

Bonyolultságelmélet

Monday 16th September, 2019, 17:46

A kurzus teljesítési követelményei

Gyakorlat

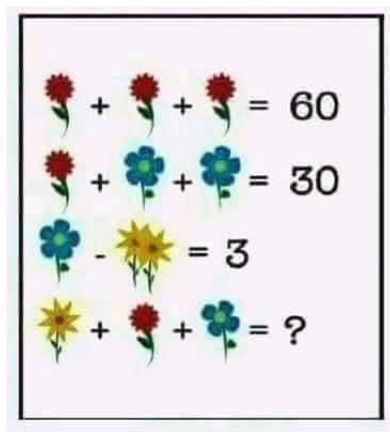
- Három kisdolgozat 6–6 pontért kb. a 4., 7. és 9. gyakorlaton
- Ha a két legjobb kisdolgozat összege kevesebb, mint 6 pont (pl. 5.975), akkor a gyakorlat nem teljesített.
- Egy nagydolgozat 28 pontért utolsó héten előadáson
- Pontszám: **két legjobb** kis- plusz a nagydolgozat pontszáma
- Ha ≥ 16 , a gyakorlat teljesítve. Ponthatárok: 16, 22, 28, 34.
- Egyébként a nagydolgozat javítása első vizsgaidőpontban
- Ha így már ≥ 16 , a gyakorlati jegy elégséges
- Ha így sem, a gyakorlat és a kurzus nem teljesített

Vizsga

- Kiskérdések összesen 50 pontért
- Ponthatárok: 21, 26, 31, 41, **szóban megvédeni**

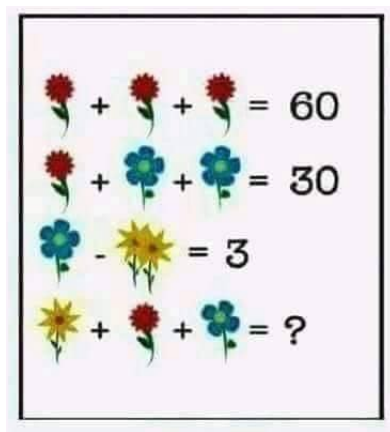
- A kiszámíthatóság- és a bonyolultságelmélet kialakulása
- Turing-gépek. A kiszámítás RAM modellje.
- Problémák egymáshoz viszonyított nehézsége: a (hatékony) visszavezetés
- Eldönthetetlen problémák
- Bonyolultsági osztályok
- Teljesség, nehézség; **NP**-teljes problémák
- Approximáció, pszeudopolinomiális algoritmusok, parametrizált komplexitás
- **PSPACE**-nehéz problémák
- Randomizált algoritmusok
- Párhuzamosítás
- Kriptográfia

Warm-up



check this

Warm-up



check this

- pirosvirág: 20
- kékvirág: 5
- sárgavirág: 1 (oh my so clever lulz)
- célfüggvény (lol): 26

95% of people cannot solve this!

$$\frac{\text{apple}}{\text{banana} + \text{pineapple}} + \frac{\text{banana}}{\text{apple} + \text{pineapple}} + \frac{\text{pineapple}}{\text{apple} + \text{banana}} = 4$$

Can you find positive whole values

for apple, banana, and pineapple?

check this

95% of people cannot solve this!

$$\frac{\text{apple}}{\text{banana} + \text{pineapple}} + \frac{\text{banana}}{\text{apple} + \text{pineapple}} + \frac{\text{pineapple}}{\text{apple} + \text{banana}} = 4$$

Can you find positive whole values

for apple, banana, and pineapple?

check this

- alma: 154476802108746166441951315019919837485664325669565431700026634898253202035277999
- banán: 36875131794129999827197811565225474825492979968971970996283137471637224634055579
- ananász: 4373612677928697257861252602371390152816537558161613618621437993378423467772036

A kiszámíthatóság elméletének kialakulása

A kiszámíthatóság elméletének kialakulása

1900: Hilbert 10. problémája (a 23-ból)

Adott $p(x_1, \dots, x_n)$ **egész együtthatós polinom**, létezik-e (egész értékű) zérushelye.

A kiszámíthatóság elméletének kialakulása

1900: Hilbert 10. problémája (a 23-ból)

Adott $p(x_1, \dots, x_n)$ **egész együtthatós polinom**, létezik-e (egész értékű) zérushelye.

pl: $x_1^3 + x_2^3 + x_3^3 - 3x_1^2x_2 - 3x_1x_2^2 - 3x_1^2x_3 - 3x_1x_3^2 - 3x_2^2x_3 - 3x_2x_3^2 - 5x_1x_2x_3$.

A kiszámíthatóság elméletének kialakulása

1900: Hilbert 10. problémája (a 23-ból)

Adott $p(x_1, \dots, x_n)$ **egész együtthetős polinom**, létezik-e (egész értékű) zérushelye.

pl: $\text{apple}^3 + \text{banana}^3 + \text{pineapple}^3 - 3\text{apple}^2\text{banana} - 3\text{apple}\text{banana}^2 - 3\text{apple}^2\text{pineapple} - 3\text{apple}\text{pineapple}^2 - 3\text{banana}^2\text{pineapple} - 3\text{banana}\text{pineapple}^2 - 5\text{apple}\text{banana}\text{pineapple}.$

A kiszámíthatóság elméletének kialakulása

1900: Hilbert 10. problémája (a 23-ból)

Adott $p(x_1, \dots, x_n)$ **egész együtthetős polinom**, létezik-e (egész értékű) zérushelye.

pl: $x_1^3 + x_2^3 + x_3^3 - 3x_1^2x_2 - 3x_1x_2^2 - 3x_1^2x_3 - 3x_1x_3^2 - 3x_2^2x_3 - 3x_2x_3^2 - 5x_1x_2x_3$.

1928: Hilbert

Találunk olyan algoritmust, amellyel a **predikátumkalkulus** tetszőleges formulájáról eldönthetjük, hogy **tautológia**-e.

A kiszámíthatóság elméletének kialakulása

1900: Hilbert 10. problémája (a 23-ból)

Adott $p(x_1, \dots, x_n)$ **egész együtthatós polinom**, létezik-e (egész értékű) zérushelye.

pl: $x_1^3 + x_2^3 + x_3^3 - 3x_1^2x_2 - 3x_1x_2^2 - 3x_1^2x_3 - 3x_1x_3^2 - 3x_2^2x_3 - 3x_2x_3^2 - 5x_1x_2x_3$.

1928: Hilbert

Találjunk olyan algoritmust, amellyel a **predikátumkalkulus** tetszőleges formulájáról eldönthetjük, hogy **tautológia**-e.

Megjegyzés

Algoritmusból sokkal kevesebb (megszámlálhatóan végtelen) van, mint eldöntési problémából (kontinuum) – Cantor, 1874 –, emiatt a problémák **túlnyomó többsége** megoldhatatlan.

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

1931, Gödel: Primitív rekurzív függvények

Mi az, hogy valami kiszámítható?

1931, Gödel: Primitív rekurzív függvények

- $f(n) = 0$;

Mi az, hogy valami kiszámítható?

1931, Gödel: Primitív rekurzív függvények

- $f(n) = 0$;
- $s(n) = n + 1$;

Mi az, hogy valami kiszámítható?

1931, Gödel: Primitív rekurzív függvények

- $f(n) = 0$;
- $s(n) = n + 1$;
- $\pi_k^i(x_1, \dots, x_k) = x_i$;

Mi az, hogy valami kiszámítható?

1931, Gödel: Primitív rekurzív függvények

- $f(n) = 0$;
- $s(n) = n + 1$;
- $\pi_k^i(x_1, \dots, x_k) = x_i$;
- $f/n, g_1/k, \dots, g_n/k \Rightarrow h(\mathbf{x}) = f(g_1(\mathbf{x}), \dots, g_n(\mathbf{x}))$;

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

1931, Gödel: Primitív rekurzív függvények

- $f(n) = 0$;
- $s(n) = n + 1$;
- $\pi_k^i(x_1, \dots, x_k) = x_i$;
- $f/n, g_1/k, \dots, g_n/k \Rightarrow h(\mathbf{x}) = f(g_1(\mathbf{x}), \dots, g_n(\mathbf{x}))$;
- $f/k, g/(k+2) \Rightarrow$

$$h(0, \mathbf{x}) = f(\mathbf{x}),$$

$$h(n+1, \mathbf{x}) = g(n, h(n, \mathbf{x}), \mathbf{x}).$$

a primitív rekurzív függvények.

Néhány primitív rekurzív függvény

Példák

Néhány primitív rekurzív függvény

Példák

- $\text{add}(0, x) = x$, $\text{add}(n + 1, x) = \text{add}(n, x) + 1$ – összeadás

Néhány primitív rekurzív függvény

Példák

- $\text{add}(0, x) = x$, $\text{add}(n + 1, x) = \text{add}(n, x) + 1$ – összeadás
- $\text{mult}(0, x) = 0$, $\text{mult}(n + 1, x) = \text{add}(\text{mult}(n, x), x)$ – szorzás

Néhány primitív rekurzív függvény

Példák

- $\text{add}(0, x) = x$, $\text{add}(n + 1, x) = \text{add}(n, x) + 1$ – összeadás
- $\text{mult}(0, x) = 0$, $\text{mult}(n + 1, x) = \text{add}(\text{mult}(n, x), x)$ – szorzás
- $\text{pow}(0, x) = 1$, $\text{pow}(n + 1, x) = \text{mult}(\text{pow}(n, x), x)$ – hatványozás

Néhány primitív rekurzív függvény

Példák

- $\text{add}(0, x) = x$, $\text{add}(n + 1, x) = \text{add}(n, x) + 1$ – összeadás
- $\text{mult}(0, x) = 0$, $\text{mult}(n + 1, x) = \text{add}(\text{mult}(n, x), x)$ – szorzás
- $\text{pow}(0, x) = 1$, $\text{pow}(n + 1, x) = \text{mult}(\text{pow}(n, x), x)$ – hatványozás
- faktoriális, különbség, egyenlőség tesztelés, előjel, egészosztás, maradék, stb

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

1934, Gödel: Általános rekurzív függvények

A primitív rekurzív függvények konstrukciói, plusz:

- $f/(k+1) \Rightarrow h(\mathbf{x}) = \min_{n \geq 0} \{n : f(n, \mathbf{x}) = 0\}$.

az általános (parciális) rekurzív függvények.

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

1934, Gödel: Általános rekurzív függvények

A primitív rekurzív függvények konstrukciói, plusz:

- $f/(k+1) \Rightarrow h(\mathbf{x}) = \min_{n \geq 0} \{n : f(n, \mathbf{x}) = 0\}$.

az általános (parciális) rekurzív függvények.

Példa: Ackermann-függvény

$$A(0, n) = n + 1$$

$$A(m + 1, 0) = A(m, 1)$$

$$A(m + 1, n + 1) = A(m, A(m + 1, n))$$

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

1934, Gödel: Általános rekurzív függvények

A primitív rekurzív függvények konstrukciói, plusz:

- $f/(k+1) \Rightarrow h(\mathbf{x}) = \min_{n \geq 0} \{n : f(n, \mathbf{x}) = 0\}$.

az általános (parciális) rekurzív függvények.

Példa: Ackermann-függvény

$$A(0, n) = n + 1$$

$$A(m + 1, 0) = A(m, 1)$$

$$A(m + 1, n + 1) = A(m, A(m + 1, n))$$

$$\text{pl. } A(1, 1) = A(0, A(1, 0)) = A(1, 0) + 1 = A(0, 1) + 1 = 2 + 1 = 3.$$

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

1934, Gödel: Általános rekurzív függvények

A primitív rekurzív függvények konstrukciói, plusz:

- $f/(k+1) \Rightarrow h(\mathbf{x}) = \min_{n \geq 0} \{n : f(n, \mathbf{x}) = 0\}$.

az általános (parciális) rekurzív függvények.

Példa: Ackermann-függvény

$$A(0, n) = n + 1$$

$$A(m + 1, 0) = A(m, 1)$$

$$A(m + 1, n + 1) = A(m, A(m + 1, n))$$

pl. $A(1, 1) = A(0, A(1, 0)) = A(1, 0) + 1 = A(0, 1) + 1 = 2 + 1 = 3$.

Az Ackermann-függvény nem primitív, de általános rekurzív függvény.

Mi az, hogy valami kiszámítható?

- 1931, Gödel: Primitív rekurzív függvények

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

- 1931, Gödel: Primitív rekurzív függvények
- 1934, Gödel: Általános rekurzív függvények

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

- 1931, Gödel: Primitív rekurzív függvények
- 1934, Gödel: Általános rekurzív függvények
- 1930-as évek eleje, Church, Kleene, Rosser (Princeton): λ -definiálhatóság

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

- 1931, Gödel: Primitív rekurzív függvények
- 1934, Gödel: Általános rekurzív függvények
- 1930-as évek eleje, Church, Kleene, Rosser (Princeton): λ -definiálhatóság

1935: AMS New York-i összejövedele

Church tézise: Az általános rekurzív függvények (matematikai fogalom) megfelelnek az algoritmikusan kiszámítható függvény fogalmának (intuitív fogalom).

A kiszámíthatóság elméletének kialakulása

Mi az, hogy valami kiszámítható?

- 1931, Gödel: Primitív rekurzív függvények
- 1934, Gödel: Általános rekurzív függvények
- 1930-as évek eleje, Church, Kleene, Rosser (Princeton): λ -definiálhatóság

1935: AMS New York-i összejövedele

Church tézise: Az általános rekurzív függvények (matematikai fogalom) megfelelnek az algoritmikusan kiszámítható függvény fogalmának (intuitív fogalom).

1936: Kleene

Az általános rekurzív függvények megegyeznek a λ -definiálható függvényekkel.

1936: Turing

Bebizonyítja, hogy a Turing-gép

- <https://www.youtube.com/watch?v=gJQTFhkhwPA>
- <https://www.youtube.com/watch?v=E3keLeMwfHY>
- <https://www.youtube.com/watch?v=FTSAiF9AHN4>

ekvivalens a λ -definiálhatósággal, és megfogalmazza azt a tézist, hogy a Turing-gép megfelel az algoritmussal kiszámítható függvényeknek.

A kiszámíthatóság elméletének kialakulása

1936: Turing

Bebizonyítja, hogy a Turing-gép

- <https://www.youtube.com/watch?v=gJQTFhkhwPA>
- <https://www.youtube.com/watch?v=E3keLeMwfHY>
- <https://www.youtube.com/watch?v=FTSAiF9AHN4>

ekvivalens a λ -definiálhatósággal, és megfogalmazza azt a tézist, hogy a **Turing-gép megfelel az algoritmussal kiszámítható függvényeknek.**

A **Church-Turing tézist** tapasztalati tények és matematikai eredmények támasztják alá:

A kiszámíthatóság elméletének kialakulása

1936: Turing

Bebizonyítja, hogy a Turing-gép

- <https://www.youtube.com/watch?v=gJQTFhkhwPA>
- <https://www.youtube.com/watch?v=E3keLeMwfHY>
- <https://www.youtube.com/watch?v=FTSAiF9AHN4>

ekvivalens a λ -definiálhatósággal, és megfogalmazza azt a tézist, hogy a **Turing-gép megfelel az algoritmussal kiszámítható függvényeknek.**

A **Church-Turing tézist** tapasztalati tények és matematikai eredmények támasztják alá:

- Minden intuitív értelemben megoldható problémáról sikerült kimutatni, hogy azok megoldhatók a matematikai modellekben is.

A kiszámíthatóság elméletének kialakulása

1936: Turing

Bebizonyítja, hogy a Turing-gép

- <https://www.youtube.com/watch?v=gJQTFhkhwPA>
- <https://www.youtube.com/watch?v=E3keLeMwfHY>
- <https://www.youtube.com/watch?v=FTSAiF9AHN4>

ekvivalens a λ -definiálhatósággal, és megfogalmazza azt a tézist, hogy a **Turing-gép megfelel az algoritmussal kiszámítható függvényeknek.**

A **Church-Turing tézist** tapasztalati tények és matematikai eredmények támasztják alá:

- Minden intuitív értelemben megoldható problémáról sikerült kimutatni, hogy azok megoldhatók a matematikai modellekben is.
- A matematikai modellek ekvivalensek.

A kiszámíthatóság elméletének kialakulása

Ezek után...

1936, '37: Church, Turing

Nem létezik olyan algoritmus, mely a predikátumkalkulus tetszőleges formulájáról eldöntené, hogy tautológia-e.

A kiszámíthatóság elméletének kialakulása

Ezek után...

1936, '37: Church, Turing

Nem létezik olyan algoritmus, mely a predikátumkalkulus tetszőleges formulájáról eldöntené, hogy tautológia-e.

1971: Matijasevič

Hilbert 10. problémája algoritmikusan **megoldhatatlan**.

Kiszámíthatóságelmélet

Melyek az **algoritmikusan** megoldható problémák?

Kiszámíthatóság és bonyolultság

Kiszámíthatóságelmélet

Melyek az **algoritmikusan** megoldható problémák?

Bonyolultságelmélet

Melyek a **gyakorlatilag** megoldható problémák? (erőforrásigény)

Kiszámíthatóság és bonyolultság

Kiszámíthatóságelmélet

Melyek az **algoritmikusan** megoldható problémák?

Bonyolultságelmélet

Melyek a **gyakorlatilag** megoldható problémák? (erőforrásigény)

1971: Cook

P és **NP** osztályok, **NP**-teljesség, **SAT** **NP**-teljes

Kiszámíthatóság és bonyolultság

Kiszámíthatóságtelmélet

Melyek az **algoritmikusan** megoldható problémák?

Bonyolultságtelmélet

Melyek a **gyakorlatilag** megoldható problémák? (erőforrásigény)

1971: Cook

P és NP osztályok, NP-teljeség, SAT NP-teljes

1972: Karp

Rámutat az NP-teljes problémák változatosságára.

Kiszámíthatóság és bonyolultság

Kiszámíthatóságtelmélet

Melyek az **algoritmikusan** megoldható problémák?

Bonyolultságtelmélet

Melyek a **gyakorlatilag** megoldható problémák? (erőforrásigény)

1971: Cook

P és NP osztályok, NP-teljeség, SAT NP-teljes

1972: Karp

Rámutat az NP-teljes problémák változatosságára.

1973: Levin

Több kombinatorikus probléma „univerzális a kimerítő keresésre”.

További bonyolultsági osztályok és számítási módok

- **L, NL, PSPACE, EXP, ...**

További bonyolultsági osztályok és számítási módok

- **L, NL, PSPACE, EXP, ...**
- Valószínűségi modellek

További bonyolultsági osztályok és számítási módok

- **L, NL, PSPACE, EXP, ...**
- Valószínűségi modellek
- Párhuzamos számítás

További bonyolultsági osztályok és számítási módok

- **L, NL, PSPACE, EXP, ...**
- Valószínűségi modellek
- Párhuzamos számítás

stb.

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

					
	C64 1Mflops	Cray Y-MP 1Gflops	mai GPU 5TFlops	Tianhe-2 34PFlops	Föld bolygó 10EFlops
n					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

					
	C64 1Mflops	Cray Y-MP 1Gflops	mai GPU 5TFlops	Tianhe-2 34PFlops	Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k				

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M			

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M		

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420				

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208			

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k		

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36				

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46			

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58		

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$					

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$	14				

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$	14	16			

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$	14	16	19		

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$	14	16	19	22	

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$	14	16	19	22	24

Mit jelent az időigény?

Az alábbi táblázat megadja, hogy adott futásidejű algoritmus adott számítási kapacitású architektúrán mekkora inputra fut még le **egy napon belül**.

	 C64 1Mflops	 Cray Y-MP 1Gflops	 mai GPU 5TFlops	 Tianhe-2 34PFlops	 Föld bolygó 10EFlops
n	86.4G	8.64×10^{13}	4.32×10^{17}	2.94×10^{21}	8.64×10^{23}
$n \log n$	2.75G	2.1×10^{12}	8.1×10^{15}	4.5×10^{19}	1.17×10^{22}
n^2	300k	9.3M	657M	54G	929G
n^3	4420	44.208	756k	14.3M	95M
2^n	36	46	58	71	79
1.1^n	264	336	426	518	578
$n!$	14	16	19	22	24

Amíg Moore törvénye (nagyjából) igaz, addig a polinomidejű algoritmusok egy-egy újabb év elteltével konstans**szor** több adattal tudnak adott időn belül elbánni.

Definíció

Definíció

Legyenek $f, g : \mathbb{N} \rightarrow \mathbb{N}$ függvények.

Definíció

Legyenek $f, g : \mathbb{N} \rightarrow \mathbb{N}$ függvények.

- $f(n) = \mathcal{O}(g(n))$, ha létezik olyan $c > 0$, hogy $f(n) \leq c \cdot g(n)$ majdnem minden n -re.

Definíció

Legyenek $f, g : \mathbb{N} \rightarrow \mathbb{N}$ függvények.

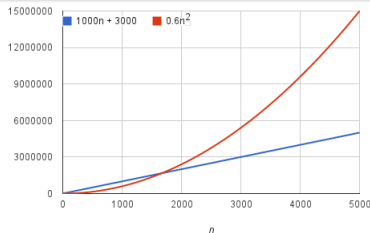
- $f(n) = \mathcal{O}(g(n))$, ha létezik olyan $c > 0$, hogy $f(n) \leq c \cdot g(n)$ majdnem minden n -re.
- $f(n) = \Omega(g(n))$, ha $g(n) = \mathcal{O}(f(n))$.

Definíció

Legyenek $f, g : \mathbb{N} \rightarrow \mathbb{N}$ függvények.

- $f(n) = \mathcal{O}(g(n))$, ha létezik olyan $c > 0$, hogy $f(n) \leq c \cdot g(n)$ majdnem minden n -re.
- $f(n) = \Omega(g(n))$, ha $g(n) = \mathcal{O}(f(n))$.
- $f(n) = \Theta(g(n))$, ha $f(n) = \mathcal{O}(g(n))$ és $g(n) = \mathcal{O}(f(n))$.

Ha $f(n) = \mathcal{O}(g(n))$, de $g(n) \neq \mathcal{O}(f(n))$, annak jele $f(n) = o(g(n))$. (és ω hasonlóan.)



$$1000n + 3000 = \mathcal{O}(0.6n^2)$$

Határértékekkel

Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \dots$

- $\dots = 0$, akkor $f(n) = o(g(n))$
- $\dots < \infty$, akkor $f(n) = \mathcal{O}(g(n))$

Határértékekkel

Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \dots$

- $\dots = 0$, akkor $f(n) = o(g(n))$
- $\dots < \infty$, akkor $f(n) = \mathcal{O}(g(n))$

Példa

Ha $p(n)$ és $q(n)$ polinomok (pozitív főegyütthatóval), akkor

- $p(n) = \mathcal{O}(q(n))$ pontosan akkor, ha p foka legfeljebb akkora, mint q -é;

Határértékekkel

Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \dots$

- $\dots = 0$, akkor $f(n) = o(g(n))$
- $\dots < \infty$, akkor $f(n) = \mathcal{O}(g(n))$

Példa

Ha $p(n)$ és $q(n)$ polinomok (pozitív főegyütthatóval), akkor

- $p(n) = \mathcal{O}(q(n))$ pontosan akkor, ha p fokszáma legfeljebb akkora, mint q -é;
- $p(n) = \Theta(q(n))$ pontosan akkor, ha fokszámaik megegyeznek.

Határértékekkel

Ha $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \dots$

- $\dots = 0$, akkor $f(n) = o(g(n))$
- $\dots < \infty$, akkor $f(n) = \mathcal{O}(g(n))$

Példa

Ha $p(n)$ és $q(n)$ polinomok (pozitív főegyütthatóval), akkor

- $p(n) = \mathcal{O}(q(n))$ pontosan akkor, ha p fokszáma legfeljebb akkora, mint q -é;
- $p(n) = \Theta(q(n))$ pontosan akkor, ha fokszámaik megegyeznek.

Legyen $a \in \mathbb{N}$, $a > 1$. Ekkor $p(n) = o(a^n)$.

A P osztály, informálisan

Definíció

Egy probléma a **P osztályba** esik, ha megoldható polinom időigényű (vagy $\mathcal{O}(n^k)$ időigényű) algoritmussal.

A P osztály, informálisan

Definíció

Egy probléma a **P osztályba** esik, ha megoldható polinom időigényű (vagy $\mathcal{O}(n^k)$ időigényű) algoritmussal.

A Cobham – Edmonds tézis

A P osztály, informálisan

Definíció

Egy probléma a **P osztályba** esik, ha megoldható polinom időigényű (vagy $\mathcal{O}(n^k)$ időigényű) algoritmussal.

A Cobham – Edmonds tézis

- Egy algoritmus akkor ad **gyakorlatilag kielégítő** megoldást egy problémára, ha polinom időigényű.

A P osztály, informálisan

Definíció

Egy probléma a **P osztályba** esik, ha megoldható polinom időigényű (vagy $\mathcal{O}(n^k)$ időigényű) algoritmussal.

A Cobham – Edmonds tézis

- Egy algoritmus akkor ad **gyakorlatilag kielégítő** megoldást egy problémára, ha polinom időigényű.
- Egy problémát akkor tekintünk **gyakorlatilag megoldhatónak**, ha a **P osztályba** esik.

Néhány ellenvetés

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.
- A **P** osztályba eső egyes problémák hatékonyan párhuzamosíthatók, míg mások (valószínűleg) nem.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.
- A **P** osztályba eső egyes problémák hatékonyan párhuzamosíthatók, míg mások (valószínűleg) nem.

Ugyanakkor

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.
- A $\mathbf{P}$ osztályba eső egyes problémák hatékonyan párhuzamosíthatók, míg mások (valószínűleg) nem.

Ugyanakkor

- A gyakorlatban előforduló $\mathbf{P}$ -beli problémák rendje kicsi.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.
- A **P** osztályba eső egyes problémák hatékonyan párhuzamosíthatók, míg mások (valószínűleg) nem.

Ugyanakkor

- A gyakorlatban előforduló **P**-beli problémák rendje kicsi. Pl. a lineáris programozás alapfeladatát megoldó első polinomidejű algoritmus, az **ellipszoid módszer** (1979, $\mathcal{O}(n^6 \cdot L)$) a **szimplex módszer** (1947) exponenciális futásidejű algoritmusánál a **gyakorlatban** lassabban futott. A szintén polinomidejű **projektív módszer** (1984, $\mathcal{O}(n^{3.5} L^2 \log L \log \log L)$) azonban már a gyakorlatban előforduló problémák esetében is gyorsabbnak bizonyult a szimplex módszernél.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.
- A $\mathbf{P}$ osztályba eső egyes problémák hatékonyan párhuzamosíthatók, míg mások (valószínűleg) nem.

Ugyanakkor

- A gyakorlatban előforduló $\mathbf{P}$ -beli problémák rendje kicsi.
- A $\mathbf{P}$ osztály elegáns matematikai elmélethez vezet.

Néhány ellenvetés

- Elfogadható-e egy $\mathcal{O}(n^{100})$ időigényű algoritmus?
- A konstansok nagysága.
- Kisméretű adatokon egy exponenciális algoritmus is jobban viselkedhet, mint egy polinomiális.
- Legrosszabb eset helyett a várható viselkedés vizsgálata is indokolt lehet.
- A $\mathbf{P}$ osztályba eső egyes problémák hatékonyan párhuzamosíthatók, míg mások (valószínűleg) nem.

Ugyanakkor

- A gyakorlatban előforduló $\mathbf{P}$ -beli problémák rendje kicsi.
- A $\mathbf{P}$ osztály elegáns matematikai elmélethez vezet.
- Fontos információkat szolgáltat a hatékonyan megoldható problémák szerkezetéről.

A kiszámítás modelljei

De milyen architektúrán polinom?

A kiszámításnak számos (matematikai) modellje létezik:

De milyen architektúrán polinom?

A kiszámításnak számos (matematikai) modellje létezik:

- Általános rekurzív függvények

De milyen architektúrán polinom?

A kiszámításnak számos (matematikai) modellje létezik:

- Általános rekurzív függvények
- λ -kalkulus

De milyen architektúrán polinom?

A kiszámításnak számos (matematikai) modellje létezik:

- Általános rekurzív függvények
- λ -kalkulus
- Turing-gép

De milyen architektúrán polinom?

A kiszámításnak számos (matematikai) modellje létezik:

- Általános rekurzív függvények
- λ -kalkulus
- Turing-gép
- RAM (Random Access Machine)

A RAM gép

RAM gép

A RAM gép

RAM gép

- rendelkezik végtelen sok **regiszterrel**

RAM gép

- rendelkezik végtelen sok **regiszterrel**
- a regiszterek típusai: **I**nput, **O**utput és **W**orking (munka-)regiszterek, minden típus regiszterei 0-tól indexeltek

RAM gép

- rendelkezik végtelen sok **regiszterrel**
- a regiszterek típusai: **I**ntput, **O**utput és **W**orking (munka-)regiszterek, minden típus regiszterei 0-tól indexeltek
- minden regiszter egy nemnegatív egész számot tartalmaz

RAM gép

- rendelkezik végtelen sok **regiszterrel**
- a regiszterek típusai: **I**ntput, **O**utput és **W**orking (munka-)regiszterek, minden típus regiszterei 0-tól indexeltek
- minden regiszter egy nemnegatív egész számot tartalmaz
- címezési módok:

RAM gép

- rendelkezik végtelen sok **regiszterrel**
- a regiszterek típusai: **I**ntput, **O**utput és **W**orking (munka-)regiszterek, minden típus regiszterei 0-tól indexeltek
- minden regiszter egy nemnegatív egész számot tartalmaz
- címezési módok:
 - k – konstans

RAM gép

- rendelkezik végtelen sok **regiszterrel**
- a regiszterek típusai: **I**ntput, **O**utput és **W**orking (munka-)regiszterek, minden típus regiszterei 0-tól indexelvek
- minden regiszter egy nemnegatív egész számot tartalmaz
- címezési módok:
 - k – konstans
 - $I[k]$, $O[k]$, $W[k]$ – direkt

RAM gép

- rendelkezik végtelen sok **regiszterrel**
- a regiszterek típusai: **I**ntput, **O**utput és **W**orking (munka-)regiszterek, minden típus regiszterei 0-tól indexelvek
- minden regiszter egy nemnegatív egész számot tartalmaz
- címezési módok:
 - k – konstans
 - $I[k]$, $O[k]$, $W[k]$ – direkt
 - $W[W[k]]$, $W[I[k]]$ stb. – indirekt

A RAM gép

RAM gép

A RAM gép

RAM gép

- program: utasítások egy véges sorozata

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)
 - JZ r, ℓ – ha $r == 0$, ugrás az ℓ . sorszámú utasításra

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)
 - JZ r, ℓ – ha $r == 0$, ugrás az ℓ . sorszámú utasításra
 - HALT, ACCEPT, REJECT – megállás

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)
 - JZ r, ℓ – ha $r == 0$, ugrás az ℓ . sorszámú utasításra
 - HALT, ACCEPT, REJECT – megállás
- a programszámláló (PC) pozitív egész, 0-ról indul

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)
 - $JZ\ r, \ell$ – ha $r == 0$, ugrás az ℓ . sorszámú utasításra
 - HALT, ACCEPT, REJECT – megállás
- a programszámláló (PC) pozitív egész, 0-ról indul
 - a gép egy lépésben végrehajtja azt az utasítást, melynek sorszáma a PC tartalma

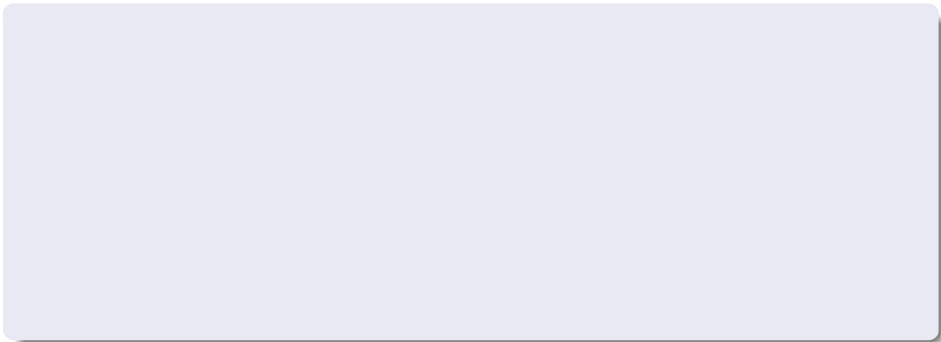
RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)
 - JZ r, ℓ – ha $r == 0$, ugrás az ℓ . sorszámú utasításra
 - HALT, ACCEPT, REJECT – megállás
- a programszámláló (PC) pozitív egész, 0-ról indul
 - a gép egy lépésben végrehajtja azt az utasítást, melynek sorszáma a PC tartalma
 - a feltételes ugrást kivéve minden lépés után eggyel nő a PC

RAM gép

- program: utasítások egy véges sorozata
- minden utasítás rendelkezik egy **sorszámmal** (egy sorszám legfeljebb egy utasításhoz tartozhat. . .)
- elemi műveletek:
 - $:=$ – értékadás (bal oldalon nem szerepelhet konstans)
 - $+$, $-$ – összeadás, kivonás ($a - b$ eredménye 0, ha $a < b$)
 - JZ r, ℓ – ha $r == 0$, ugrás az ℓ . sorszámú utasításra
 - HALT, ACCEPT, REJECT – megállás
- a programszámláló (PC) pozitív egész, 0-ról indul
 - a gép egy lépésben végrehajtja azt az utasítást, melynek sorszáma a PC tartalma
 - a feltételes ugrást kivéve minden lépés után eggyel nő a PC
 - ha nincs ilyen sorszámú utasítás, a számítás megáll

A RAM gép



A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:
 - az input az $I[1], I[2], \dots, I[m]$ számsorozat, ahol $I[m+1]$ az első 0-t tartalmazó input regiszter

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:
 - az input az $I[1], I[2], \dots, I[m]$ számsorozat, ahol $I[m+1]$ az első 0-t tartalmazó input regiszter
 - a többi regiszter tartalma 0

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:
 - az input az $I[1], I[2], \dots, I[m]$ számsorozat, ahol $I[m+1]$ az első 0-t tartalmazó input regiszter
 - a többi regiszter tartalma 0
- az output

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:
 - az input az $I[1], I[2], \dots, I[m]$ számsorozat, ahol $I[m+1]$ az első 0-t tartalmazó input regiszter
 - a többi regiszter tartalma 0
- az output
 - ACCEPT/REJECT (vagy true/false, vagy 1/0), ha a megállás egy ACCEPT/REJECT utasítás végrehajtásakor következett be

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:
 - az input az $I[1], I[2], \dots, I[m]$ számsorozat, ahol $I[m+1]$ az első 0-t tartalmazó input regiszter
 - a többi regiszter tartalma 0
- az output
 - ACCEPT/REJECT (vagy true/false, vagy 1/0), ha a megállás egy ACCEPT/REJECT utasítás végrehajtásakor következett be
 - az $O[1], O[2], \dots, O[k]$ számsorozat, ahol $O[k+1]$ az első 0-t tartalmazó output regiszter, egyébként

A RAM gép

- a számítás tetszőleges pillanatában csak véges sok regiszter értéke nem 0
- inicializálás:
 - az input az $I[1], I[2], \dots, I[m]$ számsorozat, ahol $I[m+1]$ az első 0-t tartalmazó input regiszter
 - a többi regiszter tartalma 0
- az output
 - ACCEPT/REJECT (vagy true/false, vagy 1/0), ha a megállás egy ACCEPT/REJECT utasítás végrehajtásakor következett be
 - az $O[1], O[2], \dots, O[k]$ számsorozat, ahol $O[k+1]$ az első 0-t tartalmazó output regiszter, egyébként

Az M RAM gép az $a_1, \dots, a_m$ inputon számított outputját $M(a_1, \dots, a_m)$ jelöli; ha M nem áll meg ezen az inputon, annak jele $M(a_1, \dots, a_m) = \nearrow$.

0. JZ 0, 0

if $L == R$ goto Y

1. $X := L + 1$
2. $X := X - R$
3. JZ $X, 6$
4. $X := X - 1$
5. JZ X, Y

ahol X egy másra nem használt munkaregiszter.

Feltétel nélküli ugrás, $<$, $>$, $\leq$, $\geq$ relációk szerinti feltételek hasonló módon

Stack memória, függvényhívások

Ha egy SP munkaregisztert (pl $W[0]$ -t) kinevezünk **stack pointernek**, és függvényeinket mindig **rögzített paraméterszámmal** definiáljuk, akkor lokális változókat használó rekurzív (most: „önmagukat hívó”) függvényeket is írhatunk a megszokott módon

- $W[SP] := k$, ahol k az a programsor, ahová a függvényhívás után vissza kell térnünk
- SP egyesével növelése mellett a paramétereket sorban bemásoljuk $W[SP]$ -be
- ugrunk a függvény belépési pontjára
- a függvény a verem tetején megtalálja a paramétereit, a saját lokális változóit ezek tetejére hozza létre
- visszatéréskor a megfelelő értékkel (paraméterszám+lokális változók száma) csökkenti SP-t és a megfelelő, a veremben letárolt címre ugrik vissza

Heap memória is kell?

A munkaregiszterek kettéosztásával (pl. páratlan sorszámúak a stack, párosak a heap memóriához tartoznak) többféle memóriát is lehet kezelni

- kinevezünk egy regisztert heap counternek
- dinamikus memóriafoglálásnál visszaadjuk a heap counter értékét mint „címet” és megnöveljük a heap countert a foglalni kívánt mérettel
- memória-felszabadítás:

Heap memória is kell?

A munkaregiszterek kettéosztásával (pl. páratlan sorszámúak a stack, párosak a heap memóriához tartoznak) többféle memóriát is lehet kezelni

- kinevezünk egy regisztert heap counternek
- dinamikus memóriafoglaltságnál visszaadjuk a heap counter értékét mint „címet” és megnöveljük a heap countert a foglalni kívánt mérettel
- memória-felszabadítás: nincs rá szükség

Strukturált programozás

while F do (R)

1. if $F == 0$ goto 4
2. R
3. goto 1

do (R) while F

1. R
2. if $F == 0$ goto 4
3. goto 1

for $i := 1 \dots N$ do (R)

1. $i := 1$
2. while $i \leq N$ do (
3. R
4. $i := i + 1$)

Legmagasabb bit $O(\log N)$ lépésben

```
function HIGH( $N$ )  
  if  $N == 0$  then return 0  
   $a := 1$   
  while  $a + a \leq N$  do  $a := a + a$   
  return  $a$ 
```

Pl. $\text{HIGH}(42) = 32$

Paritás $O(\log^2 N)$ lépésben

```
function ODD( $N$ )  
   $a := 0$   
  while  $a \neq N$  do  
     $N := N - a$   
     $a := \text{HIGH}(N)$   
  if  $a == 1$  then return 1  
  return 0
```

Pl.

$$42 \mapsto 42 - 32 = 10$$

$$10 \mapsto 10 - 8 = 2$$

$$2 \mapsto 2 - 2 = 0$$

return 0

Felezés $O(\log N)$ lépésben

```

function HALF( $N$ )
   $r := 0$ 
  while  $N > 1$  do
     $a := 1$ 
    while  $a + a + a + a \leq N$  do
       $a := a + a$ 
     $N := N - a - a$ 
     $r := r + a$ 
  return  $r$ 

```

Pl. HALF(45)

N	a	r
45	1	0
45	2	0
45	4	0
45	8	0
45	16	0
13	1	16
13	2	16
13	4	16
5	1	20
5	2	20
1	1	22
return 22		

Szorzás $O(\log^3 b)$ lépésben

```
function MULT( $a, b$ )  
   $r := 0$   
  while  $b > 0$  do  
    if ODD( $b$ ) then  $r := r + a$   
     $b := \text{HALF}(b)$   
     $a := a + a$   
  return  $r$ 
```

Pl. MULT(15, 11)

a	b	r
15	11	0
30	5	15
60	2	45
120	1	45
240	0	165
return 165		

Szorzás $O(\log^3 b)$ lépésben

```
function MULT( $a, b$ )  
   $r := 0$   
  while  $b > 0$  do  
    if ODD( $b$ ) then  $r := r + a$   
     $b := \text{HALF}(b)$   
     $a := a + a$   
  return  $r$ 
```

Pl. MULT(15, 11)

a	b	r
15	11	0
30	5	15
60	2	45
120	1	45
240	0	165
return 165		

Szorzás rekurzívan

```
function MULT( $a, b$ )  
  if  $b == 0$  then  
    return 0  
  if ODD( $b$ ) then  
    return MULT( $a + a, b/2$ ) +  $a$   
  return MULT( $a + a, b/2$ )
```


Szorzás $O(\log^3 b)$ lépésben

```
function MULT( $a, b$ )  
   $r := 0$   
  while  $b > 0$  do  
    if ODD( $b$ ) then  $r := r + a$   
     $b := \text{HALF}(b)$   
     $a := a + a$   
  return  $r$ 
```

Pl. MULT(15, 11)

a	b	r
15	11	0
30	5	15
60	2	45
120	1	45
240	0	165
return 165		

Szorzás rekurzívan

```
function MULT( $a, b$ )  
  if  $b == 0$  then  
    return 0  
  if ODD( $b$ ) then  
    return MULT( $a + a, b/2$ ) +  $a$   
  return MULT( $a + a, b/2$ )
```

Szorzás tail rekurzívan

```
function MULT( $a, b, r$ )  
  if  $b == 0$  then  
    return  $r$   
  if ODD( $b$ ) then  
    return MULT( $a + a, b/2, r + a$ )  
  return MULT( $a + a, b/2, r$ )
```

Duplázás / shift left konstans sok lépésben

$$R[i] := R[i] + R[i]$$

Duplázás / shift left konstans sok lépésben

$$R[i] := R[i] + R[i]$$

2^k előállítás

Inicializálás 1-gyel, majd k darab shift left – $\Theta(k)$ lépés (!)

Duplázás / shift left konstans sok lépésben

$$R[i] := R[i] + R[i]$$

2^k előállítása

Inicializálás 1-gyel, majd k darab shift left – $\Theta(k)$ lépés (!)

Az $\{1, \dots, k\}$ halmaz részhalmazzaival végzett műveletek

- k darab regiszterben, mindegyikben 0 vagy 1 a karakterisztikus függvénynek megfelelően
- elem beszúrása, törlése, lekérdezése konstans időben

Definíció

Eldöntési problémán egy tetszőleges $L \subseteq \mathbb{N}^*$ halmazt értünk (azaz számsorozatok egy halmazát).

Problémák eldöntése

Definíció

Eldöntési problémán egy tetszőleges $L \subseteq \mathbb{N}^*$ halmazt értünk (azaz számsorozatok egy halmazát).

Definíció

Az M RAM-gép **eldönti** az L problémát, ha tetszőleges $I = a_1, \dots, a_m$ inputra

$$M(I) = \begin{cases} \text{ACCEPT} & , \text{ ha } I \in L; \\ \text{REJECT} & , \text{ egyébként.} \end{cases}$$

Problémák eldöntése

Definíció

Eldöntési problémán egy tetszőleges $L \subseteq \mathbb{N}^*$ halmazt értünk (azaz számsorozatok egy halmazát).

Definíció

Az M RAM-gép **eldönti** az L problémát, ha tetszőleges $I = a_1, \dots, a_m$ inputra

$$M(I) = \begin{cases} \text{ACCEPT} & , \text{ ha } I \in L; \\ \text{REJECT} & , \text{ egyébként.} \end{cases}$$

Definíció

Az L probléma **eldönthető**, ha van őt eldöntő RAM-gép.

Problémák eldöntése

Definíció

Eldöntési problémán egy tetszőleges $L \subseteq \mathbb{N}^*$ halmazt értünk (azaz számsorozatok egy halmazát).

Definíció

Az M RAM-gép **eldönti** az L problémát, ha tetszőleges $I = a_1, \dots, a_m$ inputra

$$M(I) = \begin{cases} \text{ACCEPT} & , \text{ ha } I \in L; \\ \text{REJECT} & , \text{ egyébként.} \end{cases}$$

Definíció

Az L probléma **eldönthető**, ha van ϵ eldöntő RAM-gép.

Definíció

R jelöli az összes eldönthető probléma osztályát.

Az input mérete és a futásidő

Futásidő egy adott inputon: lépésszám

Az M RAM gép **időigénye az $a_1, \dots, a_m$ inputon** a megállásig végrehajtott utasítások száma, ha M megáll $a_1, \dots, a_m$ -en; egyébként a gép nem áll meg és időigénye végtelen

Az input mérete és a futásidő

Futásidő egy adott inputon: lépésszám

Az M RAM gép **időigénye az $a_1, \dots, a_m$ inputon** a megállásig végrehajtott utasítások száma, ha M megáll $a_1, \dots, a_m$ -en; egyébként a gép nem áll meg és időigénye végtelen

Az input mérete

Az $a_1, \dots, a_m$ input **mérete** az a_i számok **bináris reprezentációinak** hosszának összege (azaz $\sum_{i=1}^m (1 + \lfloor \log a_i \rfloor)$).

Általában n jelöli az input hosszát.

Az input mérete és a futásidő

Futásidő egy adott inputon: lépésszám

Az M RAM gép **időigénye** az $a_1, \dots, a_m$ **inputon** a megállásig végrehajtott utasítások száma, ha M megáll $a_1, \dots, a_m$ -en; egyébként a gép nem áll meg és időigénye végtelen

Az input mérete

Az $a_1, \dots, a_m$ input **mérete** az a_i számok **bináris reprezentációinak** hosszának összege (azaz $\sum_{i=1}^m (1 + \lfloor \log a_i \rfloor)$).

Általában n jelöli az input hosszát.

A gép időkorlátja

Az M RAM gép **időkorlátja** az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha tetszőleges n méretű inputon legfeljebb $f(n)$ lépésben megáll.

Definíció

P jelöli az összes **polinomidőben eldönthető** probléma osztályát, azaz melyek eldönthetőek $\mathcal{O}(n^k)$ időkorlátos RAM-géppel, valamely konstans k -ra.

A **P** osztály

Definíció

P jelöli az összes **polinomidőben eldönthető** probléma osztályát, azaz melyek eldönthetőek $\mathcal{O}(n^k)$ időkorlátos RAM-géppel, valamely konstans k -ra.

Megjegyzés

Ha RAM-gép helyett Turing-géppel definiálnánk az **R** és a **P** osztályokat, akkor is ugyanezeket az osztályokat kapnánk: **a bonyolultságelmélet független a választott architektúrától.**

(Mindaddig, amíg az architektúra realiztikus.)

A P osztály

Definíció

P jelöli az összes **polinomidőben eldönthető** probléma osztályát, azaz melyek eldönthetőek $\mathcal{O}(n^k)$ időkorlátos RAM-géppel, valamely konstans k -ra.

Megjegyzés

Ha RAM-gép helyett Turing-géppel definiálnánk az **R** és a **P** osztályokat, akkor is ugyanezeket az osztályokat kapnánk: **a bonyolultságelmélet független a választott architektúrától.**

(Mindaddig, amíg az architektúra realiztikus.)

Megjegyzés

A továbbiakban problémáink inputjainak nem csak számsorozatokot, de gráfokat és egyéb véges matematikai struktúrákat is megengedünk; ez nem gond, mert minden ilyen struktúra kódolható számsorozattal.

Polinom időben eldönthető problémák

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - ❶ $S := \{1\}$.

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - 1 $S := \{1\}$.
 - 2 Jelöljük meg az 1 csúcsot.

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - 1 $S := \{1\}$.
 - 2 Jelöljük meg az 1 csúcsot.
 - 3 Amíg S nem üres:

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - 1 $S := \{1\}$.
 - 2 Jelöljük meg az 1 csúcsot.
 - 3 Amíg S nem üres:
 - 1 Választunk egy $i \in S$ csúcsot.

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - 1 $S := \{1\}$.
 - 2 Jelöljük meg az 1 csúcsot.
 - 3 Amíg S nem üres:
 - 1 Választunk egy $i \in S$ csúcsot.
 - 2 $S := S - \{i\}$.

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - 1 $S := \{1\}$.
 - 2 Jelöljük meg az 1 csúcsot.
 - 3 Amíg S nem üres:
 - 1 Választunk egy $i \in S$ csúcsot.
 - 2 $S := S - \{i\}$.
 - 3 $j = 1..n$:
Ha $(i, j) \in E$ és j nem megjelölt,
akkor $S := S \cup \{j\}$ és jelöljük meg j -t.

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - ① $S := \{1\}$.
 - ② Jelöljük meg az 1 csúcsot.
 - ③ Amíg S nem üres:
 - ① Választunk egy $i \in S$ csúcsot.
 - ② $S := S - \{i\}$.
 - ③ $j = 1..n$:
Ha $(i, j) \in E$ és j nem megjelölt,
akkor $S := S \cup \{j\}$ és jelöljük meg j -t.
- Ha n megjelölt csúcs, adjunk ACCEPT választ,
különben adjunk REJECT választ.

Néhány kimaradt részlet

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása
 - sor: szélességi keresés

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása
 - sor: szélességi keresés
 - verem: egyfajta mélységi keresés

Polinom időben eldönthető problémák

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása
 - sor: szélességi keresés
 - verem: egyfajta mélységi keresés

Hatékonyság

Polinom időben eldönthető problémák

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása
 - sor: szélességi keresés
 - verem: egyfajta mélységi keresés

Hatékonyság

- A mátrix minden elemét legfeljebb egyszer használjuk fel.

Polinom időben eldönthető problémák

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix (Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása
 - sor: szélességi keresés
 - verem: egyfajta mélységi keresés

Hatékonyság

- A mátrix minden elemét legfeljebb egyszer használjuk fel.
- Ha az egyszerű műveletek (S egy elemének kiválasztása, megjelölése, stb.) konstans időigényűek, akkor $\mathcal{O}(n^2)$.

Függvényproblémák

Függvényprobléma egy tetszőleges $f : \mathbb{N}^* \rightarrow \mathbb{N}^*$ leképezés.

Függvényproblémák

Függvényprobléma egy tetszőleges $f : \mathbb{N}^* \rightarrow \mathbb{N}^*$ leképezés.

Az M RAM-gép az f függvényproblémát **számítja ki**, ha $M(I) = f(I)$ tetszőleges I inputra.

Függvényproblémák

Függvényprobléma egy tetszőleges $f : \mathbb{N}^* \rightarrow \mathbb{N}^*$ leképezés.

Az M RAM-gép az f függvényproblémát **számítja ki**, ha $M(I) = f(I)$ tetszőleges I inputra.

Az f függvényprobléma **rekurzív**, ha van őt kiszámító RAM-gép.

Függvényproblémák

Függvényprobléma egy tetszőleges $f : \mathbb{N}^* \rightarrow \mathbb{N}^*$ leképezés.

Az M RAM-gép az f függvényproblémát **számítja ki**, ha $M(I) = f(I)$ tetszőleges I inputra.

Az f függvényprobléma **rekurzív**, ha van őt kiszámító RAM-gép.

Megjegyzés

Mint korábban, a rekurzív függvények osztálya sem változik, ha RAM-gép helyett (például) a Turing-gép lenne a választott architektúra.

Az utazóügynök probléma

TSP

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.
Eldöntési változata: **TSP(E)**.

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.
Eldöntési változata: **TSP(E)**.
- **Triviális módszer:** az összes $\frac{(n-1)!}{2}$ lehetséges körút megvizsgálásával.

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.
Eldöntési változata: **TSP(E)**.
- **Triviális módszer:** az összes $\frac{(n-1)!}{2}$ lehetséges körút megvizsgálásával.
- **Dinamikus programozással:** $\mathcal{O}(2^n \cdot n^2)$ időigény

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.
Eldöntési változata: **TSP(E)**.
- **Triviális módszer:** az összes $\frac{(n-1)!}{2}$ lehetséges körút megvizsgálásával.
- **Dinamikus programozással:** $\mathcal{O}(2^n \cdot n^2)$ időigény

Az utazóügynök probléma

TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.
Eldöntési változata: **TSP(E)**.
- **Triviális módszer:** az összes $\frac{(n-1)!}{2}$ lehetséges körút megvizsgálásával.
- **Dinamikus programozással:** $\mathcal{O}(2^n \cdot n^2)$ időigény

Nem ismert, hogy TSP megoldható-e polinom idejű algoritmussal.
(**NP**-teljes probléma.)

Az utazóügynök probléma

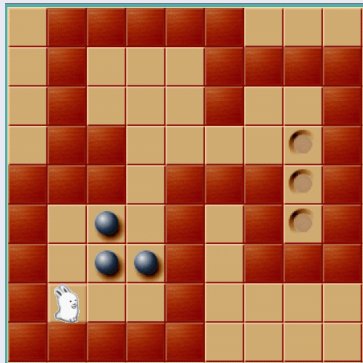
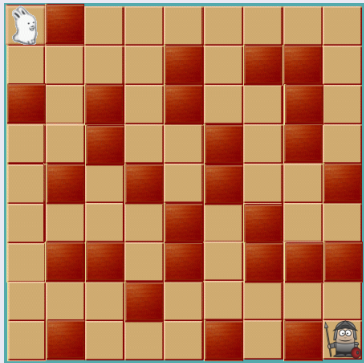
TSP

- **Adott:** az $1, 2, \dots, n$ városok és bármely két i, j város távolsága: $d_{i,j}$.
- **Kérdés:** találjunk egy, az összes várost pontosan egyszer érintő legrövidebb körutat.
Eldöntési változata: **TSP(E)**.
- **Triviális módszer:** az összes $\frac{(n-1)!}{2}$ lehetséges körút megvizsgálásával.
- **Dinamikus programozással:** $\mathcal{O}(2^n \cdot n^2)$ időigény

Nem ismert, hogy TSP megoldható-e polinom idejű algoritmussal.
(**NP**-teljes probléma.)

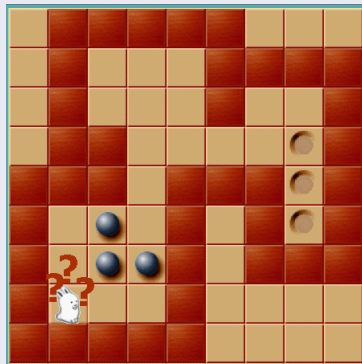
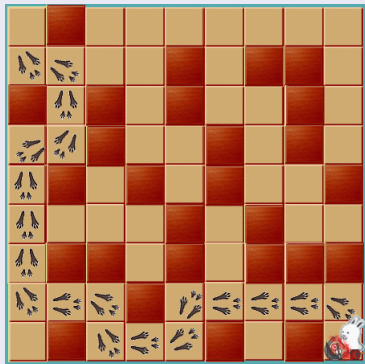
Melyik a nehezebb?

MAZE vs SOKOBAN



Melyik a nehezebb?

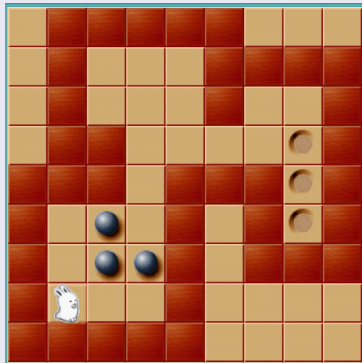
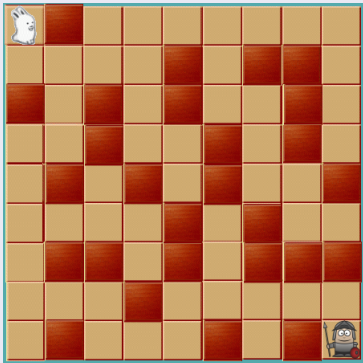
MAZE vs SOKOBAN



Úgy érzi az ember, hogy a bal oldalra van lineáris időigényű algoritmus, a jobb oldalra meg egy nagy keresési teret kell bejárni.

Melyik a nehezebb?

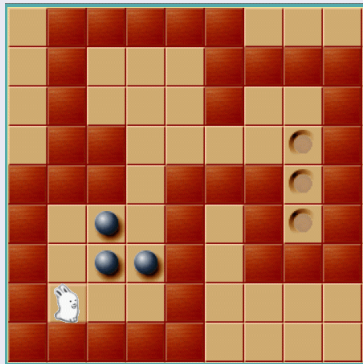
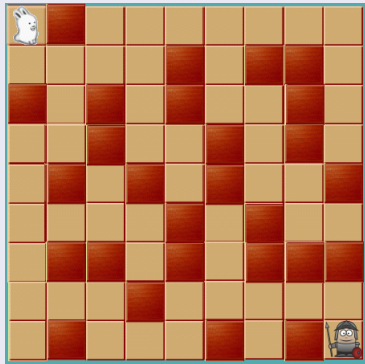
MAZE vs SOKOBAN



Úgy érzi az ember, hogy a bal oldalra van lineáris időigényű algoritmus, a jobb oldalra meg egy nagy keresési teret kell bejárni.
De ezzel két **algoritmust** hasonlítottunk össze, nem két **problémát**!

Melyik a nehezebb?

MAZE vs SOKOBAN



Úgy érzi az ember, hogy a bal oldalra van lineáris időigényű algoritmus, a jobb oldalra meg egy nagy keresési teret kell bejárni.

De ezzel két **algoritmust** hasonlítottunk össze, nem két **problémát**!

(A SOKOBANra se **talált** még senki polinomidejű algoritmust. De ez nem feltétlen jelenti, hogy nincs is. . .)

Turing-visszavezetések

Legyenek A és B eldöntési problémák.

Legyenek A és B eldöntési problémák.

B legalább olyan nehéz, mint A , ha létezik olyan hatékony módszer, azaz f rekurzív függvény, mely az A tetszőleges x bemenetéhez hozzárendeli a B egy $f(x)$ bemenetét (példányát) úgy, hogy az A -nak x akkor és csak akkor „igen” példánya, ha B -nek $f(x)$ „igen” példánya.

Turing-visszavezetések

Legyenek A és B eldöntési problémák.

B legalább olyan nehéz, mint A , ha létezik olyan hatékony módszer, azaz f rekurzív függvény, mely az A tetszőleges x bemenetéhez hozzárendeli a B egy $f(x)$ bemenetét (példányát) úgy, hogy az A -nak x akkor és csak akkor „igen” példánya, ha B -nek $f(x)$ „igen” példánya.

Jelben: $A \leq_R B$, A Turing-visszavezethető (vagy rekurzívan visszavezethető) B -re.

Turing-visszavezetések

Legyenek A és B eldöntési problémák.

B legalább olyan nehéz, mint A , ha létezik olyan hatékony módszer, azaz f rekurzív függvény, mely az A tetszőleges x bemenetéhez hozzárendeli a B egy $f(x)$ bemenetét (példányát) úgy, hogy az A -nak x akkor és csak akkor „igen” példánya, ha B -nek $f(x)$ „igen” példánya.

Jelben: $A \leq_R B$, A Turing-visszavezethető (vagy rekurzívan visszavezethető) B -re.

Ekkor:

- ha van egy B -t eldöntő algoritmusunk, akkor A -ra is van:

```
function A(x)
    return B(f(x));
```

Turing-visszavezetések

Legyenek A és B eldöntési problémák.

B legalább olyan nehéz, mint A , ha létezik olyan hatékony módszer, azaz f rekurzív függvény, mely az A tetszőleges x bemenetéhez hozzárendeli a B egy $f(x)$ bemenetét (példányát) úgy, hogy az A -nak x akkor és csak akkor „igen” példánya, ha B -nek $f(x)$ „igen” példánya.

Jelben: $A \leq_R B$, A Turing-visszavezethető (vagy rekurzívan visszavezethető) B -re.

Ekkor:

- ha van egy B -t eldöntő algoritmusunk, akkor A -ra is van:

```
function A(x)
    return B(f(x));
```

- így tehát ha $A \leq_R B$ és A -ról tudjuk, hogy eldönthetetlen, akkor B is az kell legyen.

Hatékony visszavezetés

Az f függvényre további megszorításokat teszünk.

Az f függvényre további megszorításokat teszünk.

Definíció

Azt mondjuk, hogy az A probléma (polinomidőben) **visszavezethető** a B problémára, ha létezik olyan **polinomidőben kiszámítható** f függvény, hogy minden x bemenetre

$$x \in A \Leftrightarrow f(x) \in B.$$

Az f függvényre további megszorításokat teszünk.

Definíció

Azt mondjuk, hogy az A probléma (polinomidőben) **visszavezethető** a B problémára, ha létezik olyan **polinomidőben kiszámítható** f függvény, hogy minden x bemenetre

$$x \in A \Leftrightarrow f(x) \in B.$$

Jelben: $A \leq_P B$.

$$\text{HAMILTON-ÚT} \leq_{\mathcal{P}} \text{TSP}(E)$$

HAMILTON-ÚT

HAMILTON-ÚT $\leq_{\mathcal{P}}$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.

HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosan egyszer érintő út)?

HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosan egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.

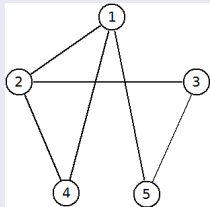
HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosan egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.



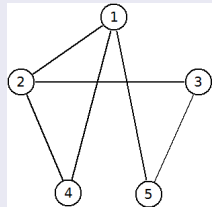
HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosan egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.



d	1	2	3	4	5
1	0	1	2	1	1
2	1	0	1	1	2
3	2	1	0	2	1
4	1	1	2	0	2
5	1	2	1	2	0

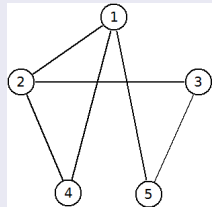
HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosán egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.



d	1	2	3	4	5
1	0	1	2	1	1
2	1	0	1	1	2
3	2	1	0	2	1
4	1	1	2	0	2
5	1	2	1	2	0

Tehát: $d_{i,j} = 1$, ha (i,j) él volt G -ben, 2 egyébként; a célérték $N+1$, ahol N a városok száma.

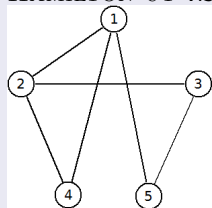
HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosán egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.



d	1	2	3	4	5
1	0	1	2	1	1
2	1	0	1	1	2
3	2	1	0	2	1
4	1	1	2	0	2
5	1	2	1	2	0

Ekkor ha van Hamilton-út G -ben, annak összköltsége $N - 1$; a két végpontját összekötve (ennek súlya 1 vagy 2) egy legfeljebb $N + 1$ súlyú körutunk van.

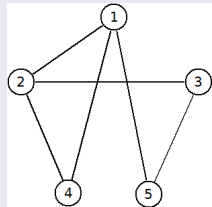
HAMILTON-ÚT $\leq_P$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosan egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.

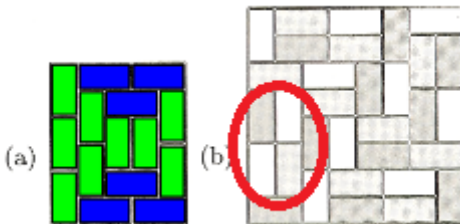


d	1	2	3	4	5
1	0	1	2	1	1
2	1	0	1	1	2
3	2	1	0	2	1
4	1	1	2	0	2
5	1	2	1	2	0

Másfelől, ha van egy legfeljebb $N + 1$ súlyú körutunk, abban legalább $N - 1$ él súlya 1 kell legyen; ezek Hamilton-utat adnak G -ben.

$\text{TATAMI} \leq_{\mathcal{P}} \text{SAT}$

- Adott egy $n \times m$ -es téglalap, melyet 2×1 -es dominókkal (forgatni ér) szeretnénk lefedni.
- A lefedésnek hogy „szép” legyen, **tatami** lefedésnek kell lennie: négy dominó nem találkozhat egy sarkon.
- Néhány dominó előre fel van rakva a táblára.
- Adjunk meg egy tatami lefedését a táblának, melyben a megadott dominók a megadott módon szerepelnek!
- Eldöntési változat: **létezik-e** ilyen lefedés?



A táblán minden **szomszédos mezőket összekötő élhez** rendelünk egy változót, pl.

- $p_{i,j}$ legyen az i . sor j . oszlopából jobbra menő él,
- $q_{i,j}$ pedig a felfele menő él,

amilyen i, j -kre ilyen élek vannak. Pl. egy 6×5 -ös táblán:

$p_{6,1}$	$p_{6,2}$	$p_{6,3}$	$p_{6,4}$	
$q_{5,1}$	$q_{5,2}$	$q_{5,3}$	$q_{5,4}$	$q_{5,5}$
$p_{5,1}$	$p_{5,2}$	$p_{5,3}$	$p_{5,4}$	
$q_{4,1}$	$q_{4,2}$	$q_{4,3}$	$q_{4,4}$	$q_{4,5}$
$p_{4,1}$	$p_{4,2}$	$p_{4,3}$	$p_{4,4}$	
$q_{3,1}$	$q_{3,2}$	$q_{3,3}$	$q_{3,4}$	$q_{3,5}$
$p_{3,1}$	$p_{3,2}$	$p_{3,3}$	$p_{3,4}$	
$q_{2,1}$	$q_{2,2}$	$q_{2,3}$	$q_{2,4}$	$q_{2,5}$
$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	$p_{2,4}$	
$q_{1,1}$	$q_{1,2}$	$q_{1,3}$	$q_{1,4}$	$q_{1,5}$
$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	

Az elképzelés:
azokat a változókat
(éleket) állítsuk 1-re,
amiket egy dominó
fed le.

	$p_{6,2}$		$p_{6,4}$	
$q_{5,1}$		$p_{5,3}$		
	$q_{4,2}$			$q_{4,5}$
$q_{3,1}$		$q_{3,3}$	$q_{3,4}$	
	$q_{2,2}$			$q_{2,5}$
$q_{1,1}$		$p_{2,3}$		
	$p_{1,2}$		$p_{1,4}$	

Azt kell megfogalmaznunk, hogy mikor felel meg egy **értékadás** egy megoldásnak, vagyis egy **tatami lefedésnek**.

- Minden **mezőt** lefed legalább egy dominó: vagyoljuk a rá illeszkedő éleket
- Minden mezőt legfeljebb egy dominó fed: az egy mezőre illeszkedő éleket **nand** kapcsolatba hozzuk: $\neg(x \wedge y)$, azaz $(\neg x \vee \neg y)$
- A tatami feltétel: minden „sarok mellett” van igaz változó – ezeket is vagyoljuk

A fenti feltételeket pedig összeésszük. Részlet:

- $(p_{4,2} \vee q_{4,2} \vee p_{4,1} \vee q_{3,2})$ – a 4. sor 2. mezőjét fedi egy dominó;
- $\neg p_{4,2} \vee \neg q_{4,2}$ – ezt a mezőt nem fedi egyszerre fentről és jobbról egy dominó;
- $(p_{3,1} \vee q_{2,1} \vee p_{2,1} \vee q_{2,2})$ – a 2. sor 1. oszlop sarkán nem érintkezik négy sarok

Plusz: az előre megadott dominóknak megfelelő változókat 1-re állítjuk

PÁROSÍTÁS

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.
- Output: van-e G -ben teljes párosítás?

PÁROSÍTÁS $\leq_P$ SAT

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.
- Output: van-e G -ben teljes párosítás?

SAT

PÁROSÍTÁS $\leq_P$ SAT

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.
- Output: van-e G -ben teljes párosítás?

SAT

- Input: konjunktív normálformájú (ítéletkalkulus-beli) formula.

PÁROSÍTÁS $\leq_P$ SAT

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.
- Output: van-e G -ben teljes párosítás?

SAT

- Input: konjunktív normálformájú (ítéletkalkulus-beli) formula.
- Output: kielégíthető-e?

PÁROSÍTÁS $\leq_P$ SAT

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.
- Output: van-e G -ben teljes párosítás?

SAT

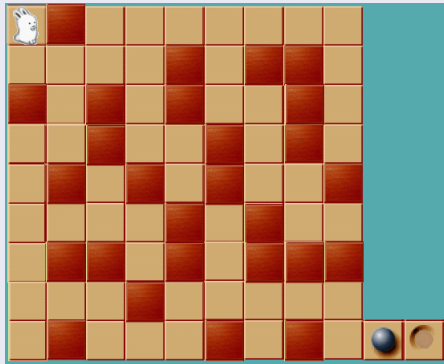
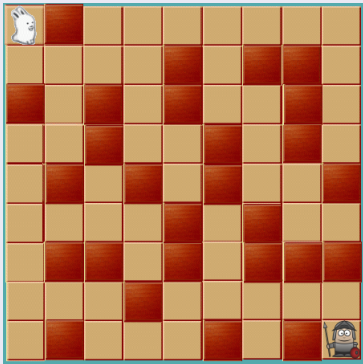
- Input: konjunktív normálformájú (ítélekalkulus-beli) formula.
- Output: kielégíthető-e?

A visszavezetés

Gyakorlaton.

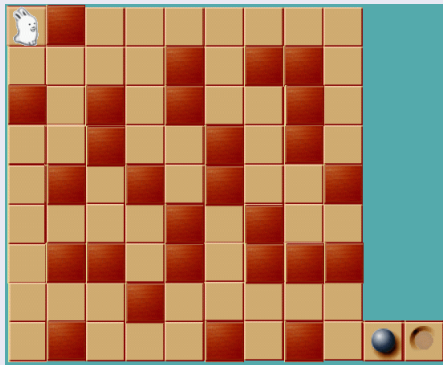
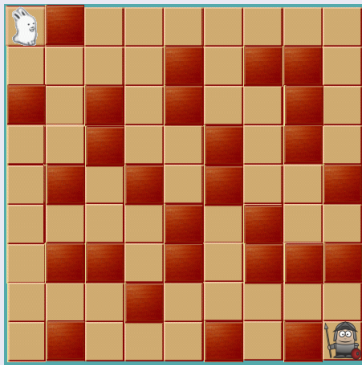
Visszatérve...

MAZE $\leq_P$ SOKOBAN



Visszatérve...

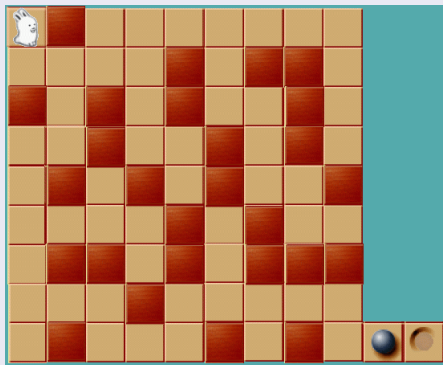
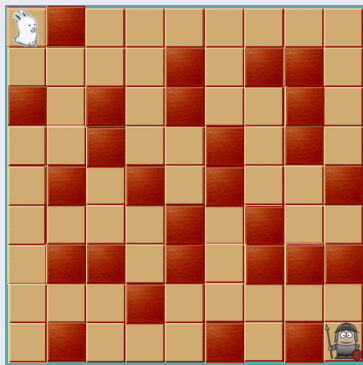
MAZE $\leq_P$ SOKOBAN



Jobb alsó sarok mellé rakni az egyetlen ládát, amellé a lyukat.

Visszatérve...

MAZE $\leq_P$ SOKOBAN



Jobb alsó sarok mellé rakni az egyetlen ládát, amellé a lyukat.

Nem ismert, hogy $\text{SOKOBAN} \leq_P \text{MAZE}$ (mely esetben „egyforma nehezek”) vagy sem.

Univerzális RAM-gép

Dacára egyszerű szintaxisának, a RAM-gép is **Turing-teljes** számítási modell: minden „hagyományos” programozási nyelven megvalósítható algoritmusra létezik RAM program is.

Univerzális RAM-gép

Dacára egyszerű szintaxisának, a RAM-gép is **Turing-teljes** számítási modell: minden „hagyományos” programozási nyelven megvalósítható algoritmusra létezik RAM program is.

Így például RAM interpretert is írhatunk RAM nyelven...

Univerzális RAM-gép

Dacára egyszerű szintaxisának, a RAM-gép is **Turing-teljes** számítási modell: minden „hagyományos” programozási nyelven megvalósítható algoritmusra létezik RAM program is.

Így például RAM interpretert is írhatunk RAM nyelven...

Tétel

Létezik olyan U **univerzális RAM-program**, amelynek ha adott egy M RAM-program és annak egy x bemenete, úgy viselkedik, mint M az x -en, vagyis $U(M; x) = M(x)$.

Univerzális RAM-gép

Dacára egyszerű szintaxisának, a RAM-gép is **Turing-teljes** számítási modell: minden „hagyományos” programozási nyelven megvalósítható algoritmusra létezik RAM program is.

Így például RAM interpretert is írhatunk RAM nyelven. . .

Tétel

Létezik olyan U **univerzális RAM-program**, amelynek ha adott egy M RAM-program és annak egy x bemenete, úgy viselkedik, mint M az x -en, vagyis $U(M; x) = M(x)$.

Természetesen M kódolva adott U számára (utasítások, tömbcímkézések. . . számokkal).

A megállási probléma – első eldönthetetlen problémánk

MEGÁLLÁS

- Input: M program, x input.
- Output: Megáll-e M az x -en futtatva?

A megállási probléma – első eldönthetetlen problémánk

MEGÁLLÁS

- Input: M program, x input.
- Output: Megáll-e M az x -en futtatva?

Tétel

A MEGÁLLÁS probléma eldönthetetlen.

A megállási probléma – első eldönthetetlen problémánk

MEGÁLLÁS

- Input: M program, x input.
- Output: Megáll-e M az x -en futtatva?

Tétel

A MEGÁLLÁS probléma eldönthetetlen.

Bizonyítás

A megállási probléma – első eldönthetetlen problémánk

MEGÁLLÁS

- Input: M program, x input.
- Output: Megáll-e M az x -en futtatva?

Tétel

A MEGÁLLÁS probléma eldönthetetlen.

Bizonyítás

Tegyük fel, hogy MEGÁLLÁS eldönthető egy M_0 programmal.

A megállási probléma – első eldönthetetlen problémánk

MEGÁLLÁS

- Input: M program, x input.
- Output: Megáll-e M az x -en futtatva?

Tétel

A MEGÁLLÁS probléma eldönthetetlen.

Bizonyítás

Tegyük fel, hogy MEGÁLLÁS eldönthető egy M_0 programmal.

Legyen D a következő (RAM-programot váró) program:

$$D(M) = \text{if } M_0(M; M) \text{ then } \nearrow \text{ else halt.}$$

A megállási probléma – első eldönthetetlen problémánk

MEGÁLLÁS

- Input: M program, x input.
- Output: Megáll-e M az x -en futtatva?

Tétel

A MEGÁLLÁS probléma eldönthetetlen.

Bizonyítás

Tegyük fel, hogy MEGÁLLÁS eldönthető egy M_0 programmal.

Legyen D a következő (RAM-programot váró) program:

$$D(M) = \text{if } M_0(M; M) \text{ then } \nearrow \text{ else halt.}$$

Ekkor:

$$D(D) = \nearrow \Leftrightarrow M_0(D; D) = \text{„igen”} \Leftrightarrow D(D) \neq \nearrow,$$

ami ellentmondás.

Visszavezetés: eldönthetetlenség bizonyítása

Ha már ismerünk egy eldönthetetlen problémát, másokról is meg tudjuk mutatni, hogy eldönthetetlenek.

Visszavezetés: eldönthetetlenség bizonyítása

Ha már ismerünk egy eldönthetetlen problémát, másokról is meg tudjuk mutatni, hogy eldönthetetlenek.

Hogyan?

Azt használjuk fel, hogy ha

Visszavezetés: eldönthetlenség bizonyítása

Ha már ismerünk egy eldönthetetlen problémát, másokról is meg tudjuk mutatni, hogy eldönthetetlenek.

Hogyan?

Azt használjuk fel, hogy ha

- A eldönthetetlen és

Visszavezetés: eldönthetlenség bizonyítása

Ha már ismerünk egy eldönthetetlen problémát, másokról is meg tudjuk mutatni, hogy eldönthetetlenek.

Hogyan?

Azt használjuk fel, hogy ha

- A eldönthetetlen és
- $A \leq_{\mathcal{R}} B$,

Visszavezetés: eldönthetlenség bizonyítása

Ha már ismerünk egy eldönthetetlen problémát, másokról is meg tudjuk mutatni, hogy eldönthetetlenek.

Hogyan?

Azt használjuk fel, hogy ha

- A eldönthetetlen és
- $A \leq_{\mathcal{R}} B$,

akkor B is eldönthetetlen.

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

- **Adott:** M RAM-program.

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

- **Adott:** M RAM-program.
- **Kérdés:** M megáll-e minden bemenetén?

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

- **Adott:** M RAM-program.
- **Kérdés:** M megáll-e minden bemenetén?

Bizonyítás

Adott M és x esetén legyen M_x olyan program, hogy az M_x minden y inputjára:
$$M_x(y) = M(x).$$

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

- **Adott:** M RAM-program.
- **Kérdés:** M megáll-e minden bemenetén?

Bizonyítás

Adott M és x esetén legyen M_x olyan program, hogy az M_x minden y inputjára:

$$M_x(y) = M(x).$$

(azaz: M_x tetszőleges y input esetén futtassa M -et x -en.)

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

- **Adott:** M RAM-program.
- **Kérdés:** M megáll-e minden bemenetén?

Bizonyítás

Adott M és x esetén legyen M_x olyan program, hogy az M_x minden y inputjára:

$$M_x(y) = M(x).$$

(azaz: M_x tetszőleges y input esetén futtassa M -et x -en.)

Így

$$M(x) \neq \nearrow \Leftrightarrow M_x \text{ megáll minden bemenetén.}$$

MINDENEN MEGÁLLÁS

Állítás

Az alábbi probléma algoritmikusan eldönthetetlen.

- **Adott:** M RAM-program.
- **Kérdés:** M megáll-e minden bemenetén?

Bizonyítás

Adott M és x esetén legyen M_x olyan program, hogy az M_x minden y inputjára:

$$M_x(y) = M(x).$$

(azaz: M_x tetszőleges y input esetén futtassa M -et x -en.)

Így

$$M(x) \neq \nearrow \Leftrightarrow M_x \text{ megáll minden bemenetén.}$$

Ez az $M; x \mapsto M_x$ hozzárendelés kiszámítható, tehát rekurzív visszavezetés; és mivel az eldönthetetlen MEGÁLLÁS problémát visszavezettük erre a problémára, ez is eldönthetetlen.

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztert 0-val inicializálunk)?

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztert 0-val inicializálunk)?

Eldönthetetlen!

Visszavezetjük rá a MEGÁLLÁS problémát.

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztert 0-val inicializálunk)?

Eldönthetetlen!

Visszavezetjük rá a MEGÁLLÁS problémát.

Adott M -hez és x -hez legyen M_x az a program, ami tetszőleges y inputra

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztert 0-val inicializálunk)?

Eldönthetetlen!

Visszavezetjük rá a MEGÁLLÁS problémát.

Adott M -hez és x -hez legyen M_x az a program, ami tetszőleges y inputra

- ha y az üres input, akkor futtatja M -et x -en;

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztert 0-val inicializálunk)?

Eldönthetetlen!

Visszavezetjük rá a MEGÁLLÁS problémát.

Adott M -hez és x -hez legyen M_x az a program, ami tetszőleges y inputra

- ha y az üres input, akkor futtatja M -et x -en;
- egyébként mondjuk megáll.

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztert 0-val inicializálunk)?

Eldönthetetlen!

Visszavezetjük rá a MEGÁLLÁS problémát.

Adott M -hez és x -hez legyen M_x az a program, ami tetszőleges y inputra

- ha y az üres input, akkor futtatja M -et x -en;
- egyébként mondjuk megáll.

Ez az M_x az $M; x$ párból elkészíthető és nyilván

M megáll x -en $\Leftrightarrow M_x$ megáll üres inputon.

ÜRES INPUTON MEGÁLLÁS

Legyen ÜRES INPUTON MEGÁLLÁS a következő probléma:

- Input: egy M program.
- Output: megáll-e M az üres inputon (azaz mikor minden input regisztrált 0-val inicializálunk)?

Eldönthetetlen!

Visszavezetjük rá a MEGÁLLÁS problémát.

Adott M -hez és x -hez legyen M_x az a program, ami tetszőleges y inputra

- ha y az üres input, akkor futtatja M -et x -en;
- egyébként mondjuk megáll.

Ez az M_x az $M; x$ párból elkészíthető és nyilván

$$M \text{ megáll } x\text{-en} \Leftrightarrow M_x \text{ megáll üres inputon.}$$

Tehát $\text{MEGÁLLÁS} \leq_{\mathcal{R}} \text{ÜRES INPUTON MEGÁLLÁS}$, így ez utóbbi is eldönthetetlen probléma.

Mit láttunk eddig?

A MEGÁLLÁS, MINDENEN MEGÁLLÁS és ÜRESSZÓN MEGÁLLÁS problémák eldönthetetlensége szerint...

Mit láttunk eddig?

A MEGÁLLÁS, MINDENEN MEGÁLLÁS és ÜRESSZÓN MEGÁLLÁS problémák eldönthetetlensége szerint...

- **nincs** olyan algoritmus, mely inputként kap egy **JAVA forráskódot** és hozzá egy **inputot**, és megválaszolja, hogy a megadott program megáll-e a megadott inputon;

Mit láttunk eddig?

A MEGÁLLÁS, MINDENEN MEGÁLLÁS és ÜRESSZÓN MEGÁLLÁS problémák eldönthetetlensége szerint...

- **nincs** olyan algoritmus, mely inputként kap egy **JAVA forráskódot** és hozzá egy **inputot**, és megválaszolja, hogy a megadott program megáll-e a megadott inputon;
- **nincs** olyan algoritmus, mely inputként kap egy forráskódot és megválaszolja, hogy a program eshet-e végtelen ciklusba egyáltalán,

Mit láttunk eddig?

A MEGÁLLÁS, MINDENEN MEGÁLLÁS és ÜRESSZÓN MEGÁLLÁS problémák eldönthetetlensége szerint...

- **nincs** olyan algoritmus, mely inputként kap egy **JAVA forráskódot** és hozzá egy **inputot**, és megválaszolja, hogy a megadott program megáll-e a megadott inputon;
- **nincs** olyan algoritmus, mely inputként kap egy forráskódot és megválaszolja, hogy a program eshet-e végtelen ciklusba egyáltalán, vagy akár csak azt, hogy paraméter nélkül elindítva végtelen ciklusba fog-e esni!

Mit láttunk eddig?

A MEGÁLLÁS, MINDENEN MEGÁLLÁS és ÜRESSZÓN MEGÁLLÁS problémák eldönthetetlensége szerint...

- **nincs** olyan algoritmus, mely inputként kap egy **JAVA forráskódot** és hozzá egy **inputot**, és megválaszolja, hogy a megadott program megáll-e a megadott inputon;
- **nincs** olyan algoritmus, mely inputként kap egy forráskódot és megválaszolja, hogy a program eshet-e végtelen ciklusba egyáltalán, vagy akár csak azt, hogy paraméter nélkül elindítva végtelen ciklusba fog-e esni!

A helyzet azonban ennél sokkal rosszabb...

Definíció

Egy A probléma **rekurzívan felsorolható** (vagy Turing-felismerhető), ha van olyan M RAM-program, amire

$$M(x) = \begin{cases} \text{ACCEPT} & , \text{ ha } x \in A \end{cases}$$

Definíció

Egy A probléma **rekurzívan felsorolható** (vagy Turing-felismerhető), ha van olyan M RAM-program, amire

$$M(x) = \begin{cases} \text{ACCEPT} & , \text{ ha } x \in A \\ \nearrow & , \text{ egyébként.} \end{cases}$$

Definíció

Egy A probléma **rekurzívan felsorolható** (vagy Turing-felismerhető), ha van olyan M RAM-program, amire

$$M(x) = \begin{cases} \text{ACCEPT} & , \text{ ha } x \in A \\ \nearrow & , \text{ egyébként.} \end{cases}$$

A rekurzíván felsorolható problémák osztályát **RE** jelöli.

Definíció

Egy A probléma **rekurzívan felsorolható** (vagy Turing-felismerhető), ha van olyan M RAM-program, amire

$$M(x) = \begin{cases} \text{ACCEPT} & , \text{ ha } x \in A \\ \nearrow & , \text{ egyébként.} \end{cases}$$

A rekurzíván felsorolható problémák osztályát **RE** jelöli.

Nyilván $\mathbf{R} \subseteq \mathbf{RE}$.

Definíció

Egy A probléma **rekurzívan felsorolható** (vagy Turing-felismerhető), ha van olyan M RAM-program, amire

$$M(x) = \begin{cases} \text{ACCEPT} & , \text{ ha } x \in A \\ \nearrow & , \text{ egyébként.} \end{cases}$$

A rekurzívan felsorolható problémák osztályát **RE** jelöli.

Nilván $\mathbf{R} \subseteq \mathbf{RE}$. (Hiszen egy RAM programot ACCEPT-től különböző válasz **helyett** végtelen ciklusba egyszerű küldeni.)

Tétel

$$\mathbf{R} \subsetneq \mathbf{RE}.$$

Tétel

$$\mathbf{R} \subsetneq \mathbf{RE}.$$

Bizonyítás

Az eldönthetetlen MEGÁLLÁS probléma rekurzívan felsorolható, pl. azzal a RAM-programmal, mely futtatja az univerzális RAM-programot az input $M; x$ páron, majd ha az megáll, ACCEPT választ ad.

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

Vagyis: ha $\emptyset \neq \mathcal{C} \subsetneq \mathbf{RE}$ a rekurzívan felsorolható problémák egy nemtriviális osztálya, akkor a következő probléma eldönthetetlen: adott egy M program, igaz-e, hogy $L(M) \in \mathcal{C}$?

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

Vagyis: ha $\emptyset \neq \mathcal{C} \subsetneq \mathbf{RE}$ a rekurzívan felsorolható problémák egy nemtriviális osztálya, akkor a következő probléma eldönthetetlen: adott egy M program, igaz-e, hogy $L(M) \in \mathcal{C}$?

Másképp mondva: automatikusan **semmi nemtriviálisat** nem tudunk eldönteni egy program **viselkedéséről** a **forráskódjának** ismeretében.

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

Vagyis: ha $\emptyset \neq \mathcal{C} \subsetneq \mathbf{RE}$ a rekurzívan felsorolható problémák egy nemtriviális osztálya, akkor a következő probléma eldönthetetlen: adott egy M program, igaz-e, hogy $L(M) \in \mathcal{C}$?

Másképp mondva: automatikusan **semmi nemtriviálisat** nem tudunk eldönteni egy program **viselkedéséről** a **forráskódjának** ismeretében.

Ezért válnak szükségessé például

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

Vagyis: ha $\emptyset \neq \mathcal{C} \subsetneq \mathbf{RE}$ a rekurzívan felsorolható problémák egy nemtriviális osztálya, akkor a következő probléma eldönthetetlen: adott egy M program, igaz-e, hogy $L(M) \in \mathcal{C}$?

Másképp mondva: automatikusan **semmi nemtriviálisat** nem tudunk eldönteni egy program **viselkedéséről** a **forráskódjának** ismeretében.

Ezért válnak szükségessé például kommentek,

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

Vagyis: ha $\emptyset \neq \mathcal{C} \subsetneq \mathbf{RE}$ a rekurzívan felsorolható problémák egy nemtriviális osztálya, akkor a következő probléma eldönthetetlen: adott egy M program, igaz-e, hogy $L(M) \in \mathcal{C}$?

Másképp mondva: automatikusan **semmi nemtriviálisat** nem tudunk eldönteni egy program **viselkedéséről** a **forráskódjának** ismeretében.

Ezért válnak szükségessé például kommentek, tervezési minták használata,

A „rosszabb helyzet”: Rice tétele

Rice tétele

A rekurzívan felsorolható problémák egyetlen nemtriviális tulajdonsága sem dönthető el algoritmikusan.

Vagyis: ha $\emptyset \neq \mathcal{C} \subsetneq \mathbf{RE}$ a rekurzívan felsorolható problémák egy nemtriviális osztálya, akkor a következő probléma eldönthetetlen: adott egy M program, igaz-e, hogy $L(M) \in \mathcal{C}$?

Másképp mondva: automatikusan **semmi nemtriviálisat** nem tudunk eldönteni egy program **viselkedéséről** a **forráskódjának** ismeretében.

Ezért válnak szükségessé például kommentek, tervezési minták használata, statikus és dinamikus tesztelés...

További eldönthetetlen problémák

Hilbert 10. problémája

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.
- **Kérdés:** Létezik-e egész értékű zérushelye?

További eldönthetetlen problémák

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.
- **Kérdés:** Létezik-e egész értékű zérushelye?

Tétel (Matijasevič)

Hilbert 10. problémája algoritmikusan megoldhatatlan.

További eldönthetetlen problémák

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.
- **Kérdés:** Létezik-e egész értékű zérushelye?

Tétel (Matijasevič)

Hilbert 10. problémája algoritmikusan megoldhatatlan.

Post megfelelezési problémája

További eldönthetetlen problémák

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.
- **Kérdés:** Létezik-e egész értékű zérushelye?

Tétel (Matijasevič)

Hilbert 10. problémája algoritmikusan megoldhatatlan.

Post megfelelezési problémája

- **Adott:** $u_1, v_1, \dots, u_n, v_n \in \Sigma^*$.

További eldönthetetlen problémák

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.
- **Kérdés:** Létezik-e egész értékű zérushelye?

Tétel (Matijasevič)

Hilbert 10. problémája algoritmikusan megoldhatatlan.

Post megfelelezési problémája

- **Adott:** $u_1, v_1, \dots, u_n, v_n \in \Sigma^*$.
- **Kérdés:** Létezik-e olyan $i_1, \dots, i_k$ ($k > 0$, $1 \leq i_j \leq n$) sorozat, hogy
$$u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k} ?$$

További eldönthetetlen problémák

Hilbert 10. problémája

- **Adott:** $p(x_1, \dots, x_n)$ egész együtthatós polinom.
- **Kérdés:** Létezik-e egész értékű zérushelye?

Tétel (Matijasevič)

Hilbert 10. problémája algoritmikusan megoldhatatlan.

Post megfelelezési problémája

- **Adott:** $u_1, v_1, \dots, u_n, v_n \in \Sigma^*$.
- **Kérdés:** Létezik-e olyan $i_1, \dots, i_k$ ($k > 0$, $1 \leq i_j \leq n$) sorozat, hogy
$$u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k} ?$$

Tétel (Post)

A fenti probléma algoritmikusan megoldhatatlan.

Post megfelelezési problémája

Példa

Pl: $\begin{pmatrix} 0 \\ 100 \end{pmatrix}$, $\begin{pmatrix} 01 \\ 00 \end{pmatrix}$, $\begin{pmatrix} 110 \\ 11 \end{pmatrix}$ -nek van:

$$\begin{pmatrix} 110 \\ 11 \end{pmatrix} \begin{pmatrix} 01 \\ 00 \end{pmatrix} \begin{pmatrix} 110 \\ 11 \end{pmatrix} \begin{pmatrix} 0 \\ 100 \end{pmatrix}$$

Egy dominó probléma

- **Adott:** Dominó típusok véges halmaza.

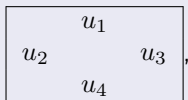
Egy dominó probléma

- **Adott:** Dominó típusok véges halmaza.
- **Kérdés:** Kirakható-e ezekkel a dominókkal az egész sík hézagmentesen?

Egy dominó probléma

- **Adott:** Dominó típusok véges halmaza.
- **Kérdés:** Kirakható-e ezekkel a dominókkal az egész sík hézagmentesen?

Típus:

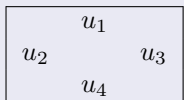


, $u_i \in \Sigma^*$.

Egy dominó probléma

- **Adott:** Dominó típusok véges halmaza.
- **Kérdés:** Kirakható-e ezekkel a dominókkal az egész sík hézagmentesen?

Típus:



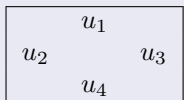
$$u_i \in \Sigma^*.$$

A dominók nem forgathatóak.

Egy dominó probléma

- **Adott:** Dominó típusok véges halmaza.
- **Kérdés:** Kirakható-e ezekkel a dominókkal az egész sík hézagmentesen?

Típus:



$$u_i \in \Sigma^*.$$

A dominók nem forgathatóak.

Tétel

A dominó probléma algoritmikusan megoldhatatlan.

Egy megoldatlan probléma

- **Adott:** m pozitív egész szám.

Egy megoldatlan probléma

- **Adott:** m pozitív egész szám.
- **Kérdés:** Kongruens-e m ?

Egy megoldatlan probléma

- **Adott:** m pozitív egész szám.
- **Kérdés:** Kongruens-e m ?

Tehát azt kérdezzük, hogy létezik-e olyan derékszögű háromszög, mely oldalai racionális hosszúságúak és melynek területe m .

Egy megoldatlan probléma

- **Adott:** m pozitív egész szám.
- **Kérdés:** Kongruens-e m ?

Tehát azt kérdezzük, hogy létezik-e olyan derékszögű háromszög, mely oldalai racionális hosszúságúak és melynek területe m .

Példa

1, 2, 3, 4 **nem** kongruensek.

Egy megoldatlan probléma

- **Adott:** m pozitív egész szám.
- **Kérdés:** Kongruens-e m ?

Tehát azt kérdezzük, hogy létezik-e olyan derékszögű háromszög, mely oldalai racionális hosszúságúak és melynek területe m .

Példa

1, 2, 3, 4 **nem** kongruensek. 5 kongruens (Fibonacci):

$$a = \frac{3}{2}, \quad b = \frac{20}{3}, \quad c = \frac{41}{6}$$

Egy megoldatlan probléma

- **Adott:** m pozitív egész szám.
- **Kérdés:** Kongruens-e m ?

Tehát azt kérdezzük, hogy létezik-e olyan derékszögű háromszög, mely oldalai racionális hosszúságúak és melynek területe m .

Példa

1, 2, 3, 4 **nem** kongruensek. 5 kongruens (Fibonacci):

$$a = \frac{3}{2}, b = \frac{20}{3}, c = \frac{41}{6}$$

Nem ismert, hogy a probléma algoritmikusan eldönthető-e.

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Ötlet

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Ötlet

- A rekurzív $\Rightarrow A$ rekurzívan felsorolható

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Ötlet

- A rekurzív $\Rightarrow A$ rekurzívan felsorolható
 A rekurzív $\Rightarrow \overline{A}$ rekurzív, így $\overline{A}$ rekurzívan felsorolható

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Ötlet

- A rekurzív $\Rightarrow A$ rekurzívan felsorolható
 A rekurzív $\Rightarrow \overline{A}$ rekurzív, így $\overline{A}$ rekurzívan felsorolható
- Tegyük fel, hogy A és $\overline{A}$ rekurzívan felsorolhatók.

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Ötlet

- A rekurzív $\Rightarrow A$ rekurzívan felsorolható
 A rekurzív $\Rightarrow \overline{A}$ rekurzív, így $\overline{A}$ rekurzívan felsorolható
- Tegyük fel, hogy A és $\overline{A}$ rekurzívan felsorolhatók. Ekkor A felismerhető egy M , $\overline{A}$ pedig egy $\overline{M}$ programmal.

Állítás

Ha A rekurzív, akkor $\overline{A}$ is az.

Ötlet

Az ACCEPT és a REJECT sorok cseréjével.

Állítás

Egy A probléma pontosan akkor rekurzív, ha A és $\overline{A}$ rekurzívan felsorolhatók.

Ötlet

- A rekurzív $\Rightarrow A$ rekurzívan felsorolható
 A rekurzív $\Rightarrow \overline{A}$ rekurzív, így $\overline{A}$ rekurzívan felsorolható
- Tegyük fel, hogy A és $\overline{A}$ rekurzívan felsorolhatók. Ekkor A felismerhető egy M , $\overline{A}$ pedig egy $\overline{M}$ programmal. Szimuláljuk M -et és $\overline{M}$ -et párhuzamosan!

Definíció

Legyen $\mathcal{C}$ problémák egy osztálya. Ekkor

$$\text{co}\mathcal{C} = \{\overline{L} : L \in \mathcal{C}\}.$$

Definíció

Legyen $\mathcal{C}$ problémák egy osztálya. Ekkor

$$\text{co}\mathcal{C} = \{\overline{L} : L \in \mathcal{C}\}.$$

Példák

Definíció

Legyen $\mathcal{C}$ problémák egy osztálya. Ekkor

$$\text{co}\mathcal{C} = \{\overline{L} : L \in \mathcal{C}\}.$$

Példák

- **coR**: azon problémák, melyek komplementere rekurzív

Definíció

Legyen $\mathcal{C}$ problémák egy osztálya. Ekkor

$$\text{co}\mathcal{C} = \{\overline{L} : L \in \mathcal{C}\}.$$

Példák

- **coR**: azon problémák, melyek komplementere rekurzív
- **coRE**: azon problémák, melyek komplementere rekurzívan felsorolható

R, **RE** és **coRE** viszonya

Következmény

$$\begin{aligned}\mathbf{R} &= \mathbf{coR} \\ \mathbf{RE} &\neq \mathbf{coRE} \\ \mathbf{R} &= \mathbf{RE} \cap \mathbf{coRE}\end{aligned}$$

Nemdeterminizmus

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Nemdeterminizmus

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Nemdeterminizmus

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy **nemdeterminisztikus** RAM-programban

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy **nemdeterminisztikus** RAM-programban

- több különböző programsor is kaphatja **ugyanazt a sorszámot**

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy **nemdeterminisztikus** RAM-programban

- több különböző programsor is kaphatja **ugyanazt a sorszámot**
- egy **lehetséges futás** minden lépésében a PC által kijelölt sorszámú lehetséges utasítások **egyike** hajtódik végre

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy **nemdeterminisztikus** RAM-programban

- több különböző programsor is kaphatja **ugyanazt a sorszámot**
- egy **lehetséges futás** minden lépésében a PC által kijelölt sorszámú lehetséges utasítások **egyike** hajtódik végre
- (utána a PC eggyel nő vagy ugrás esetén a megfelelő értékre áll be)

Nemdeterminizmus

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy **nemdeterminisztikus** RAM-programban

- több különböző programsor is kaphatja **ugyanazt a sorszámot**
- egy **lehetséges futás** minden lépésében a PC által kijelölt sorszámú lehetséges utasítások **egyike** hajtódik végre
- (utána a PC eggyel nő vagy ugrás esetén a megfelelő értékre áll be)

A program **elfogadja** az inputot, ha **van olyan** lehetséges futás, mely ACCEPT utasítással terminál, egyébként **elutasítja** azt

Nemdeterminizmus

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy **nemdeterminisztikus** RAM-programban

- több különböző programsor is kaphatja **ugyanazt a sorszámot**
- egy **lehetséges futás** minden lépésében a PC által kijelölt sorszámú lehetséges utasítások **egyike** hajtódik végre
- (utána a PC eggyel nő vagy ugrás esetén a megfelelő értékre áll be)

A program **elfogadja** az inputot, ha **van olyan** lehetséges futás, mely ACCEPT utasítással terminál, egyébként **elutasítja** azt

<https://www.youtube.com/watch?v=lufECeWtN34>

Nemdeterminisztikus generálás

Egy bit

1. $a := 0$
1. $a := 1$
2. ...

Egy n -bites szám

function ND(n)

1. $r := 0$
2. if $n == 0$ **return** r
3. $r := r + r$
4. $r := r + 1$
4. $r := r + 0$
5. $n := n - 1$
6. goto 2

Példa: HAMILTON-ÚT

- **Input:** $\vec{G} = (V, E)$ (irányított) gráf. Feltehető, hogy $V = \{1, \dots, n\}$.
- **Output:** Van-e $\vec{G}$ -ben Hamilton-út (azaz minden csúcsot érintő út)?

```
for  $i := 1 \dots n$  do  
     $W[i] := \text{ND}(1 + \lfloor \log n \rfloor)$   
for  $i := 1 \dots n - 1$  do  
    if  $(W[i], W[i + 1]) \notin E$  then REJECT  
for  $i := 1 \dots n$  do  
    for  $j := 1 \dots n$  do  
        if  $W[j] == i$  then BREAK  
    if  $j > n$  then REJECT  
ACCEPT
```

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program **időigénye** $f(n)$, ha bármely n méretű inputon **tetszőleges** lehetséges futás $f(n)$ lépésben véget ér.

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program **időigénye** $f(n)$, ha bármely n méretű inputon **tetszőleges** lehetséges futás $f(n)$ lépésben véget ér.

HAMILTON-ÚT időigénye

Az előző fólia algoritmusának időigénye például **négyzetes**:

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program **időigénye** $f(n)$, ha bármely n méretű inputon **tetszőleges** lehetséges futás $f(n)$ lépésben véget ér.

HAMILTON-ÚT időigénye

Az előző fólia algoritmusának időigénye például **négyzetes**:

- nemdeterminisztikusan generál n darab, egyenként $\approx \log n$ -bites számot, ez $n \log n$ lépés;

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program **időigénye** $f(n)$, ha bármely n méretű inputon **tetszőleges** lehetséges futás $f(n)$ lépésben véget ér.

HAMILTON-ÚT időigénye

Az előző fólia algoritmusának időigénye például **négyzetes**:

- nemdeterminisztikusan generál n darab, egyenként $\approx \log n$ -bites számot, ez $n \log n$ lépés;
- (determinisztikusan) ellenőrzi, hogy az n szám ebben a sorrendben valóban egy séta-e a gráfban, ez n lépés;

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program **időigénye** $f(n)$, ha bármely n méretű inputon **tetszőleges** lehetséges futás $f(n)$ lépésben véget ér.

HAMILTON-ÚT időigénye

Az előző fólia algoritmusának időigénye például **négyzetes**:

- nemdeterminisztikusan generál n darab, egyenként $\approx \log n$ -bites számot, ez $n \log n$ lépés;
- (determinisztikusan) ellenőrzi, hogy az n szám ebben a sorrendben valóban egy séta-e a gráfban, ez n lépés;
- (determinisztikusan) ellenőrzi, hogy minden csúcs előfordul-e a sétában, ez n^2 lépés.

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program **időigénye** $f(n)$, ha bármely n méretű inputon **tetszőleges** lehetséges futás $f(n)$ lépésben véget ér.

HAMILTON-ÚT időigénye

Az előző fólia algoritmusának időigénye például **négyzetes**:

- nemdeterminisztikusan generál n darab, egyenként $\approx \log n$ -bites számot, ez $n \log n$ lépés;
- (determinisztikusan) ellenőrzi, hogy az n szám ebben a sorrendben valóban egy séta-e a gráfban, ez n lépés;
- (determinisztikusan) ellenőrzi, hogy minden csúcs előfordul-e a sétában, ez n^2 lépés.

Megjegyzés

Az utolsó ellenőrzés hatékonyabban is elvégezhető, most a pszeudokód egyszerűségét tartottuk szem előtt.

Példa: SAT



Példa: SAT

- **Input:** konjunktív normálformájú formula

Példa: SAT

- **Input:** konjunktív normálformájú formula
- **Output:** kielégíthető-e?

Példa: SAT

- **Input:** konjunktív normálformájú formula
- **Output:** kielégíthető-e?

Algoritmus

Példa: SAT

- **Input:** konjunktív normálformájú formula
- **Output:** kielégíthető-e?

Algoritmus

- minden változónak nemdeterminisztikusan beállítjuk az értékét 1-re vagy 0-ra (lineáris időigény);

- **Input:** konjunktív normálformájú formula
- **Output:** kielégíthető-e?

Algoritmus

- minden változónak nemdeterminisztikusan beállítjuk az értékét 1-re vagy 0-ra (lineáris időigény);
- determinisztikusan ellenőrizzük, hogy a kapott kiértékelés kielégítő-e (lineáris időigény);

- **Input:** konjunktív normálformájú formula
- **Output:** kielégíthető-e?

Algoritmus

- minden változónak nemdeterminisztikusan beállítjuk az értékét 1-re vagy 0-ra (lineáris időigény);
- determinisztikusan ellenőrizzük, hogy a kapott kiértékelés kielégítő-e (lineáris időigény);
- ha igen, ACCEPT, egyébként REJECT választ adunk.

Példa: 3 – SZÍNEZÉS



Példa: 3 – SZÍNEZÉS

- **Input:** $G = (V, E)$ (irányítatlan) gráf.

Példa: 3 – SZÍNEZÉS

- **Input:** $G = (V, E)$ (irányítatlan) gráf.
- **Output:** kiszínezhető-e G **helyesen**, vagyis adhatunk-e minden csúcsnak egy-egy színt egy háromelemű színhalmazból, hogy szomszédos csúcsok különböző színt kapjanak?

Példa: 3 – SZÍNEZÉS

- **Input:** $G = (V, E)$ (irányítatlan) gráf.
- **Output:** kiszínezhető-e G **helyesen**, vagyis adhatunk-e minden csúcsnak egy-egy színt egy háromelemű színhalmazból, hogy szomszédos csúcsok különböző színt kapjanak?

Algoritmus

Példa: 3 – SZÍNEZÉS

- **Input:** $G = (V, E)$ (irányítatlan) gráf.
- **Output:** kiszínezhető-e G **helyesen**, vagyis adhatunk-e minden csúcsnak egy-egy színt egy háromelemű színhalmazból, hogy szomszédos csúcsok különböző színt kapjanak?

Algoritmus

- minden csúcsnak nemdeterminisztikusan adunk egy színt az $\{R, G, B\}$ halmazból;

Példa: 3 – SZÍNEZÉS

- **Input:** $G = (V, E)$ (irányítatlan) gráf.
- **Output:** kiszínezhető-e G **helyesen**, vagyis adhatunk-e minden csúcsnak egy-egy színt egy háromelemű színhalmazból, hogy szomszédos csúcsok különböző színt kapjanak?

Algoritmus

- minden csúcsnak nemdeterminisztikusan adunk egy színt az $\{R, G, B\}$ halmazból;
- determinisztikusan ellenőrizzük, hogy a kapott színezés helyes-e;

Példa: 3 – SZÍNEZÉS

- **Input:** $G = (V, E)$ (irányítatlan) gráf.
- **Output:** kiszínezhető-e G **helyesen**, vagyis adhatunk-e minden csúcsnak egy-egy színt egy háromelemű színhalmazból, hogy szomszédos csúcsok különböző színt kapjanak?

Algoritmus

- minden csúcsnak nemdeterminisztikusan adunk egy színt az $\{R, G, B\}$ halmazból;
- determinisztikusan ellenőrizzük, hogy a kapott színezés helyes-e;
- ha igen, ACCEPT, egyébként REJECT választ adunk.

Az NP osztály

Az NP osztályba mindazok a problémák tartoznak, melyek eldönthetők **polinom időigényű nemdeterminisztikus programmal**.

Az **NP** osztály

Az **NP** osztályba mindazok a problémák tartoznak, melyek eldönthetők **polinom időigényű nemdeterminisztikus programmal**.

Azaz, $L \in \mathbf{NP}$, ha van olyan M polinom időkorlátos nemdeterminisztikus program, melyre

Az **NP** osztály

Az **NP** osztályba mindazok a problémák tartoznak, melyek eldönthetők **polinom időigényű nemdeterminisztikus programmal**.

Azaz, $L \in \mathbf{NP}$, ha van olyan M polinom időkorlátos nemdeterminisztikus program, melyre

- ha $x \in L$, akkor M -nek **létezik** elfogadó futása x -en, és

Az NP osztály

Az **NP** osztályba mindazok a problémák tartoznak, melyek eldönthetők **polinom időigényű nemdeterminisztikus programmal**.

Azaz, $L \in \mathbf{NP}$, ha van olyan M polinom időkorlátos nemdeterminisztikus program, melyre

- ha $x \in L$, akkor M -nek **létezik** elfogadó futása x -en, és
- ha $x \notin L$, akkor M -nek minden futása elutasítja x -et.

Az NP osztály

Az **NP** osztályba mindazok a problémák tartoznak, melyek eldönthetők **polinom időigényű nemdeterminisztikus programmal**.

Azaz, $L \in \mathbf{NP}$, ha van olyan M polinom időkorlátos nemdeterminisztikus program, melyre

- ha $x \in L$, akkor M -nek **létezik** elfogadó futása x -en, és
- ha $x \notin L$, akkor M -nek minden futása elutasítja x -et.

Példa

HAMILTON-ÚT, SAT és 3 – SZÍNEZÉS **NP**-beli problémák.

Az NP osztály

Az **NP** osztályba mindazok a problémák tartoznak, melyek eldönthetők **polinom időigényű nemdeterminisztikus programmal**.

Azaz, $L \in \mathbf{NP}$, ha van olyan M polinom időkorlátos nemdeterminisztikus program, melyre

- ha $x \in L$, akkor M -nek **létezik** elfogadó futása x -en, és
- ha $x \notin L$, akkor M -nek minden futása elutasítja x -et.

Példa

HAMILTON-ÚT, SAT és 3 – SZÍNEZÉS **NP**-beli problémák.

Mivel minden $f(n)$ időigényű program felfogható $f(n)$ időigényű nemdeterminisztikus programként is, így nyilván $\mathbf{P} \subseteq \mathbf{NP}$.

Az előző példákban az algoritmusok mind a következő sémára épültek fel:

Az előző példákban az algoritmusok mind a következő sémára épültek fel:

- nemdeterminisztikusan generáltak valami „bizonyítékot” arra, hogy az input a problémának egy IGEN példánya, majd

Az előző példákban az algoritmusok mind a következő sémára épültek fel:

- nemdeterminisztikusan generáltak valami „bizonyítékot” arra, hogy az input a problémának egy IGEN példánya, majd
- determinisztikusan ellenőrizték, hogy tényleg jó bizonyítékot sikerült-e generálni.

Az előző példákban az algoritmusok mind a következő sémára épültek fel:

- nemdeterminisztikusan generáltak valami „bizonyítékot” arra, hogy az input a problémának egy IGEN példánya, majd
- determinisztikusan ellenőrizték, hogy tényleg jó bizonyítékot sikerült-e generálni.

Ez a séma **pontosan karakterizálja az NP osztályt!**

Tétel

Egy L probléma pontosan akkor van az **NP** osztályban, ha létezik egy olyan $K \subseteq \mathbb{N}^* \times \mathbb{N}^*$ reláció „inputok” és „tanúk” közt, melyre a következők fennállnak:

Tétel

Egy L probléma pontosan akkor van az **NP** osztályban, ha létezik egy olyan $K \subseteq \mathbb{N}^* \times \mathbb{N}^*$ reláció „inputok” és „tanúk” közt, melyre a következők fennállnak:

- létezik olyan k konstans, melyre ha $(x, y) \in K$, akkor $|y| \leq |x|^k$ (azaz az egyes inputokhoz tartozó tanúk „rövidek”);

Tétel

Egy L probléma pontosan akkor van az **NP** osztályban, ha létezik egy olyan $K \subseteq \mathbb{N}^* \times \mathbb{N}^*$ reláció „inputok” és „tanúk” közt, melyre a következők fennállnak:

- létezik olyan k konstans, melyre ha $(x, y) \in K$, akkor $|y| \leq |x|^k$ (azaz az egyes inputokhoz tartozó tanúk „rövidek”);
- K polinom időben eldönthető (azaz van hatékony algoritmus, mely az x, y párra eldönti, hogy y az x -hez tartozó tanú-e);

Tétel

Egy L probléma pontosan akkor van az **NP** osztályban, ha létezik egy olyan $K \subseteq \mathbb{N}^* \times \mathbb{N}^*$ reláció „inputok” és „tanúk” közt, melyre a következők fennállnak:

- létezik olyan k konstans, melyre ha $(x, y) \in K$, akkor $|y| \leq |x|^k$ (azaz az egyes inputokhoz tartozó tanúk „rövidek”);
- K polinom időben eldönthető (azaz van hatékony algoritmus, mely az x, y párra eldönti, hogy y az x -hez tartozó tanú-e);
- $x \in L$ pontosan akkor, ha $\exists y : (x, y) \in K$ (azok az inputok a probléma IGEN példányai, melyekre van tanú).

Tétel

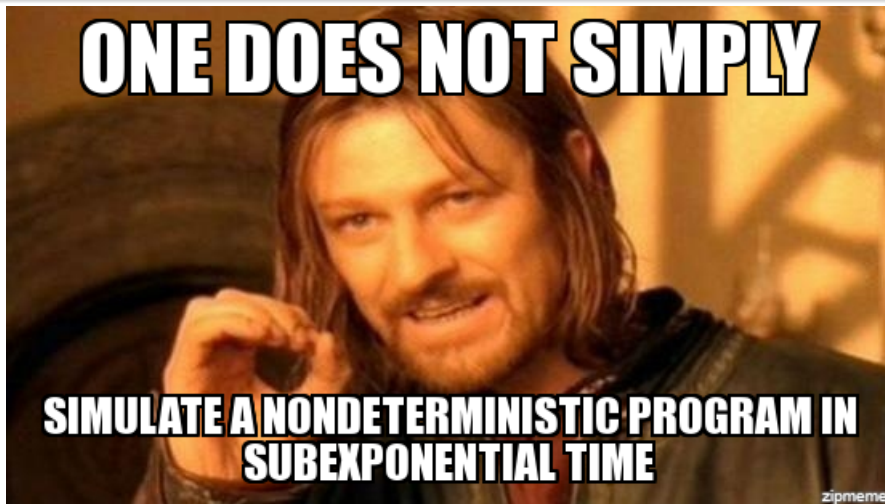
Egy L probléma pontosan akkor van az **NP** osztályban, ha létezik egy olyan $K \subseteq \mathbb{N}^* \times \mathbb{N}^*$ reláció „inputok” és „tanúk” közt, melyre a következők fennállnak:

- létezik olyan k konstans, melyre ha $(x, y) \in K$, akkor $|y| \leq |x|^k$ (azaz az egyes inputokhoz tartozó tanúk „rövidek”);
- K polinom időben eldönthető (azaz van hatékony algoritmus, mely az x, y párra eldönti, hogy y az x -hez tartozó tanú-e);
- $x \in L$ pontosan akkor, ha $\exists y : (x, y) \in K$ (azok az inputok a probléma IGEN példányai, melyekre van tanú).

A HAMILTON-ÚT problémánál egy gráfhoz tartozó tanú pl. maga egy Hamilton-út: lineáris méretű és könnyű **ellenőrizni**, hogy tényleg Hamilton-út vagy sem, és – nyilván – pontosan az IGEN példányokra van ilyen tanú.

Hatékonyak ezek a „polinomidejű” algoritmusok?

Hatékonyak ezek a „polinomidejű” algoritmusok?



Legyen $\mathcal{C}$ (eldöntési) problémák egy osztálya.

Legyen $\mathcal{C}$ (eldöntési) problémák egy osztálya.

- Az A probléma $\mathcal{C}$ -nehéz, ha $\mathcal{C}$ minden eleme visszavezethető rá.

Nehézség, teljesség

Legyen $\mathcal{C}$ (eldöntési) problémák egy osztálya.

- Az A probléma $\mathcal{C}$ -nehéz, ha $\mathcal{C}$ minden eleme visszavezethető rá.
- Ha még $A \in \mathcal{C}$ is, akkor A $\mathcal{C}$ -teljes.

Legyen $\mathcal{C}$ (eldöntési) problémák egy osztálya.

- Az A probléma **$\mathcal{C}$ -nehéz**, ha $\mathcal{C}$ **minden** eleme visszavezethető rá.
- Ha még $A \in \mathcal{C}$ is, akkor A **$\mathcal{C}$ -teljes**.

Észrevétel

Ha egy **NP**-teljes probléma polinomidőben eldönthető, akkor (és csak akkor) $P = NP$.

Legyen $\mathcal{C}$ (eldöntési) problémák egy osztálya.

- Az A probléma **$\mathcal{C}$ -nehéz**, ha $\mathcal{C}$ **minden** eleme visszavezethető rá.
- Ha még $A \in \mathcal{C}$ is, akkor A **$\mathcal{C}$ -teljes**.

Észrevétel

Ha egy **NP**-teljes probléma polinomidőben eldönthető, akkor (és csak akkor) $P = NP$.

(Hiszen akkor **NP** bármelyik problémája eldönthető egy polinomidejű visszavezetés és egy, a fenti problémát eldöntő polinomidejű algoritmus kompozíciójával, tehát polinomidőben.)

$\mathcal{C}$ -nehéz problémák keresése

Észrevétel

A hatékony visszavezetés tranzitív.

Észrevétel

A hatékony visszavezetés tranzitív.

Ezért hogy megmutassuk egy A probléma $\mathcal{C}$ -nehézségét, mindössze vissza kell rá vezetnünk egy másik, ismert $\mathcal{C}$ -nehéz B problémát. Hisz ekkor

Észrevétel

A hatékony visszavezetés tranzitív.

Ezért hogy megmutassuk egy A probléma $\mathcal{C}$ -nehézségét, mindössze vissza kell rá vezetnünk egy másik, ismert $\mathcal{C}$ -nehéz B problémát. Hisz ekkor

- minden $\mathcal{C}$ -beli visszavezethető B -re,

Észrevétel

A hatékony visszavezetés tranzitív.

Ezért hogy megmutassuk egy A probléma $\mathcal{C}$ -nehézségét, mindössze vissza kell rá vezetnünk egy másik, ismert $\mathcal{C}$ -nehéz B problémát. Hisz ekkor

- minden $\mathcal{C}$ -beli visszavezethető B -re,
- B visszavezethető A -ra,

ezért (tranzitivitás!) minden $\mathcal{C}$ -beli is visszavezethető A -ra.

Észrevétel

A hatékony visszavezetés tranzitív.

Ezért hogy megmutassuk egy A probléma $\mathcal{C}$ -nehézségét, mindössze vissza kell rá vezetnünk egy másik, ismert $\mathcal{C}$ -nehéz B problémát. Hisz ekkor

- minden $\mathcal{C}$ -beli visszavezethető B -re,
- B visszavezethető A -ra,

ezért (tranzitivitás!) minden $\mathcal{C}$ -beli is visszavezethető A -ra.

Csakhoggy...

C-nehéz problémák keresése

Észrevétel

A hatékony visszavezetés tranzitív.

Ezért hogy megmutassuk egy A probléma C -nehézségét, mindössze vissza kell rá vezetnünk egy másik, ismert C -nehéz B problémát. Hisz ekkor

- minden C -beli visszavezethető B -re,
- B visszavezethető A -ra,

ezért (tranzitivitás!) minden C -beli is visszavezethető A -ra.

Csakhogy...

... az **első** C -nehéz problémát hogy kapjuk?

Definíció

Azt mondjuk, hogy egy $\mathcal{C}$ bonyolultsági osztály **zárt a visszavezetésre**, ha valahányszor $A \leq_{\mathcal{P}} B$ és $B \in \mathcal{C}$, mindig teljesül $A \in \mathcal{C}$ is.

Visszavezetésre való zártság

Definíció

Azt mondjuk, hogy egy $\mathcal{C}$ bonyolultsági osztály **zárt a visszavezetésre**, ha valahányszor $A \leq_P B$ és $B \in \mathcal{C}$, mindig teljesül $A \in \mathcal{C}$ is.

Állítás

Az eddig látott bonyolultsági osztályok (**P**, **NP**, **R**, **RE**) zártak a visszavezetésre.

Visszavezetésre való zártság

Definíció

Azt mondjuk, hogy egy $\mathcal{C}$ bonyolultsági osztály **zárt a visszavezetésre**, ha valahányszor $A \leq_{\mathcal{P}} B$ és $B \in \mathcal{C}$, mindig teljesül $A \in \mathcal{C}$ is.

Állítás

Az eddig látott bonyolultsági osztályok (**P**, **NP**, **R**, **RE**) zártak a visszavezetésre.

Ha $\mathcal{C}$ zárt a visszavezetésre és A egy $\mathcal{C}$ -teljes probléma,

Visszavezetésre való zártság

Definíció

Azt mondjuk, hogy egy $\mathcal{C}$ bonyolultsági osztály **zárt a visszavezetésre**, ha valahányszor $A \leq_{\mathcal{P}} B$ és $B \in \mathcal{C}$, mindig teljesül $A \in \mathcal{C}$ is.

Állítás

Az eddig látott bonyolultsági osztályok (**P**, **NP**, **R**, **RE**) zártak a visszavezetésre.

Ha $\mathcal{C}$ zárt a visszavezetésre és A egy $\mathcal{C}$ -teljes probléma, akkor $\mathcal{C}$ -ben pontosan az A -ra visszavezethető problémák vannak ($\mathcal{C} = \{B : B \leq_{\mathcal{P}} A\}$).

Visszavezetésre való zártság

Definíció

Azt mondjuk, hogy egy $\mathcal{C}$ bonyolultsági osztály **zárt a visszavezetésre**, ha valahányszor $A \leq_{\mathcal{P}} B$ és $B \in \mathcal{C}$, mindig teljesül $A \in \mathcal{C}$ is.

Állítás

Az eddig látott bonyolultsági osztályok (**P**, **NP**, **R**, **RE**) zártak a visszavezetésre.

Ha $\mathcal{C}$ zárt a visszavezetésre és A egy $\mathcal{C}$ -teljes probléma, akkor $\mathcal{C}$ -ben pontosan az A -ra visszavezethető problémák vannak ($\mathcal{C} = \{B : B \leq_{\mathcal{P}} A\}$). Tehát A reprezentálja az egész osztályt!

Megjegyzés

P bármely két nemtriviális A , B elemére igaz, hogy $A \leq_{\mathcal{P}} B$.

Első **RE**-teljes problémánk

MEGÁLLÁS **RE**-teljes.

Első **RE**-teljes problémánk

MEGÁLLÁS **RE**-teljes.

Legyen $A \in \mathbf{RE}$ egy tetszőleges probléma, amit felismer az M RAM-program. Akkor tetszőleges x inputra

$$x \in A \Leftrightarrow (M; x) \in \text{MEGÁLLÁS},$$

hiszen ha M az A problémát ismeri fel, akkor pontosan az IGEN példányain áll meg (ACCEPT választ adva), így az $x \mapsto (M; x)$ leképezés egy (hatékony) visszavezetése A -nak a MEGÁLLÁS problémára.

Első **RE**-teljes problémánk

MEGÁLLÁS **RE**-teljes.

Legyen $A \in \mathbf{RE}$ egy tetszőleges probléma, amit felismer az M RAM-program. Akkor tetszőleges x inputra

$$x \in A \Leftrightarrow (M; x) \in \text{MEGÁLLÁS},$$

hiszen ha M az A problémát ismeri fel, akkor pontosan az IGEN példányain áll meg (ACCEPT választ adva), így az $x \mapsto (M; x)$ leképezés egy (hatékony) visszavezetése A -nak a MEGÁLLÁS problémára.

Így ha $\text{MEGÁLLÁS} \leq A$ valamely A problémára, akkor A is **RE**-nehéz:

- MINDENEN MEGÁLLÁS
- ÜRES INPUTON MEGÁLLÁS
- EKVIVALENCIA
- ...

Ha $\mathcal{C}, \mathcal{C}'$ zártak a visszavezetésre, A pedig egy $\mathcal{C}$ -nehéz és $\mathcal{C}'$ -beli probléma, akkor $\mathcal{C} \subseteq \mathcal{C}'$.

Ha $\mathcal{C}, \mathcal{C}'$ zártak a visszavezetésre, A pedig egy $\mathcal{C}$ -nehéz és $\mathcal{C}'$ -beli probléma, akkor $\mathcal{C} \subseteq \mathcal{C}'$.

Láttuk, hogy $\text{MEGÁLLÁS} \leq \text{EKVIVALENCIA}$.

Az is igaz, hogy $\text{MEGÁLLÁS} \leq \text{EKVIVALENCIA}$, tehát EKVIVALENCIA **RE**-nehéz és **coRE**-nehéz is egyszerre.

Mivel **RE** $\neq$ **coRE** (emiat persze egyikük sem lehet része a másiknak, hisz ha pl. **RE** $\subseteq$ **coRE**, akkor **coRE** $\subseteq$ **cocoRE** = **RE**, mely esetben egyenlőek), ez azt jelenti, hogy EKVIVALENCIA **nem RE $\cup$ coRE-beli probléma!**

Azaz sem olyan algoritmus nincs, mely felismeri, ha két program ekvivalens, sem olyan, mely felismeri, ha két program nem ekvivalens.

Definíció

Hálózat: körmentes irányított gráf,
a csúcsok címkézettek: $\wedge$, $\vee$, $\neg$, igaz, hamis, x_i

Definíció

Hálózat: körmentes irányított gráf,
a csúcsok címkézettek: $\wedge$, $\vee$, $\neg$, igaz, hamis, x_i
 $\wedge$, $\vee$ címke esetén a csúcs befoka 2,

Definíció

Hálózat: körmentes irányított gráf,
a csúcsok címkézettek: $\wedge$, $\vee$, $\neg$, igaz, hamis, x_i
 $\wedge$, $\vee$ címke esetén a csúcs befoka 2,
 $\neg$ címke esetén a csúcs befoka 1,

Definíció

Hálózat: körmentes irányított gráf,
a csúcsok címkézettek: $\wedge$, $\vee$, $\neg$, igaz, hamis, x_i
 $\wedge$, $\vee$ címke esetén a csúcs befoka 2,
 $\neg$ címke esetén a csúcs befoka 1,
igaz, hamis, x_i címke esetén a csúcs befoka 0.

Definíció

Hálózat: körmentes irányított gráf,
a csúcsok címkézettek: $\wedge$, $\vee$, $\neg$, igaz, hamis, x_i
 $\wedge$, $\vee$ címke esetén a csúcs befoka 2,
 $\neg$ címke esetén a csúcs befoka 1,
igaz, hamis, x_i címke esetén a csúcs befoka 0.

Általában megköveteljük még, hogy pontosan egy csúcs kifoka legyen 0.

Definíció

Hálózat: körmentes irányított gráf,
a csúcsok címkézettek: $\wedge$, $\vee$, $\neg$, igaz, hamis, x_i
 $\wedge$, $\vee$ címke esetén a csúcs befoka 2,
 $\neg$ címke esetén a csúcs befoka 1,
igaz, hamis, x_i címke esetén a csúcs befoka 0.

Általában megköveteljük még, hogy pontosan egy csúcs kifoka legyen 0.

Amennyiben a változók közül az $x_1, \dots, x_n$ fordulnak elő a hálózatban (legfeljebb), akkor a hálózat kiszámít egy $\{\text{igaz, hamis}\}^n \rightarrow \{\text{igaz, hamis}\}$ Boole-függvényt.

HÁLÓZAT-KIÉRTÉKELÉS

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

- **Adott:** hálózat.

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

- **Adott:** hálózat.
- **Kérdés:** a változóknak lehet-e úgy értéket adni, hogy a hálózat értéke igaz legyen?

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

- **Adott:** hálózat.
- **Kérdés:** a változóknak lehet-e úgy értéket adni, hogy a hálózat értéke igaz legyen?

SAT

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

- **Adott:** hálózat.
- **Kérdés:** a változóknak lehet-e úgy értéket adni, hogy a hálózat értéke igaz legyen?

SAT

- **Adott:** egy konjunktív normálformájú (ítéletkalkulus-beli) formula.

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

- **Adott:** hálózat.
- **Kérdés:** a változóknak lehet-e úgy értéket adni, hogy a hálózat értéke igaz legyen?

SAT

- **Adott:** egy konjunktív normálformájú (ítéletkalkulus-beli) formula.
- **Kérdés:** kielégíthető-e?

HÁLÓZAT-KIÉRTÉKELÉS

- **Adott:** változómentes hálózat.
- **Kérdés:** igaz-e az értéke?

HÁLÓZAT-KIELÉGÍTHETŐSÉG

- **Adott:** hálózat.
- **Kérdés:** a változóknak lehet-e úgy értéket adni, hogy a hálózat értéke igaz legyen?

SAT

- **Adott:** egy konjunktív normálformájú (ítéletkalkulus-beli) formula.
- **Kérdés:** kielégíthető-e?

Állítás

$$\text{HÁLÓZAT-KIELÉGÍTHETŐSÉG} \leq_P \text{SAT}.$$

A visszavezetés

Minden g csúcsnak feleljen meg a g változó.

A visszavezetés

Minden g csúcsnak feleljen meg a g változó.

$$g \text{ címkéje } x \quad \mapsto \quad x \leftrightarrow g, \text{ azaz } (\neg x \vee g) \wedge (x \vee \neg g)$$

A visszavezetés

Minden g csúcsnak feleljen meg a g változó.

g címkéje x $\mapsto x \leftrightarrow g$, azaz $(\neg x \vee g) \wedge (x \vee \neg g)$

g címkéje igaz $\mapsto g$

A visszavezetés

Minden g csúcsnak feleljen meg a g változó.

g címkéje x $\mapsto x \leftrightarrow g$, azaz $(\neg x \vee g) \wedge (x \vee \neg g)$

g címkéje igaz $\mapsto g$

g címkéje hamis $\mapsto \neg g$

A visszavezetés

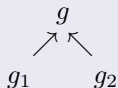
Minden g csúcsnak feleljen meg a g változó.

g címkéje $x \mapsto x \leftrightarrow g$, azaz $(\neg x \vee g) \wedge (x \vee \neg g)$

g címkéje igaz $\mapsto g$

g címkéje hamis $\mapsto \neg g$

g címkéje $\vee$:



$\mapsto g \leftrightarrow (g_1 \vee g_2)$, azaz
 $(\neg g \vee g_1 \vee g_2) \wedge (\neg g_1 \vee g) \wedge (\neg g_2 \vee g)$

A visszavezetés

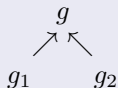
Minden g csúcsnak feleljen meg a g változó.

g címkéje $x \mapsto x \leftrightarrow g$, azaz $(\neg x \vee g) \wedge (x \vee \neg g)$

g címkéje igaz $\mapsto g$

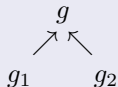
g címkéje hamis $\mapsto \neg g$

g címkéje $\vee$:



$\mapsto g \leftrightarrow (g_1 \vee g_2)$, azaz
 $(\neg g \vee g_1 \vee g_2) \wedge (\neg g_1 \vee g) \wedge (\neg g_2 \vee g)$

g címkéje $\wedge$:



$\mapsto g \leftrightarrow (g_1 \wedge g_2)$, azaz
 $(\neg g \vee g_1) \wedge (\neg g \vee g_2) \wedge (\neg g_1 \vee \neg g_2 \vee g)$

A visszavezetés

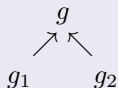
Minden g csúcsnak feleljen meg a g változó.

g címkéje $x \mapsto x \leftrightarrow g$, azaz $(\neg x \vee g) \wedge (x \vee \neg g)$

g címkéje igaz $\mapsto g$

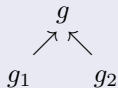
g címkéje hamis $\mapsto \neg g$

g címkéje $\vee$:



$\mapsto g \leftrightarrow (g_1 \vee g_2)$, azaz
 $(\neg g \vee g_1 \vee g_2) \wedge (\neg g_1 \vee g) \wedge (\neg g_2 \vee g)$

g címkéje $\wedge$:



$\mapsto g \leftrightarrow (g_1 \wedge g_2)$, azaz
 $(\neg g \vee g_1) \wedge (\neg g \vee g_2) \wedge (\neg g_1 \vee \neg g_2 \vee g)$

g címkéje $\neg$:



$\mapsto g \leftrightarrow (\neg g_1)$, azaz
 $(\neg g \vee \neg g_1) \wedge (g_1 \vee g)$

A visszavezetés

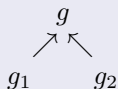
Minden g csúcsnak feleljen meg a g változó.

g címkéje $x \mapsto x \leftrightarrow g$, azaz $(\neg x \vee g) \wedge (x \vee \neg g)$

g címkéje igaz $\mapsto g$

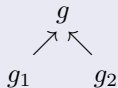
g címkéje hamis $\mapsto \neg g$

g címkéje $\vee$:



$\mapsto g \leftrightarrow (g_1 \vee g_2)$, azaz
 $(\neg g \vee g_1 \vee g_2) \wedge (\neg g_1 \vee g) \wedge (\neg g_2 \vee g)$

g címkéje $\wedge$:



$\mapsto g \leftrightarrow (g_1 \wedge g_2)$, azaz
 $(\neg g \vee g_1) \wedge (\neg g \vee g_2) \wedge (\neg g_1 \vee \neg g_2 \vee g)$

g címkéje $\neg$:



$\mapsto g \leftrightarrow (\neg g_1)$, azaz
 $(\neg g \vee \neg g_1) \wedge (g_1 \vee g)$

g kimenő kapu

$\mapsto g$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_{\mathcal{P}}$ SAT

A keresett formula: a fenti formulák konjunkciója.

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT

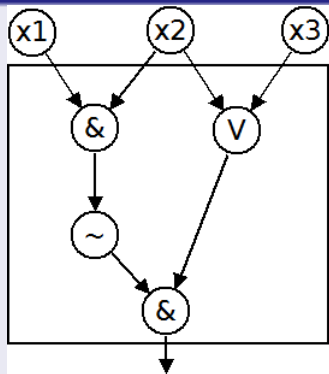
A keresett formula: a fenti formulák konjunkciója.

Adott hálózathoz a formula elkészíthető lineáris időben.

Továbbá a hálózat akkor és csak akkor kielégíthető, ha a formula az.

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

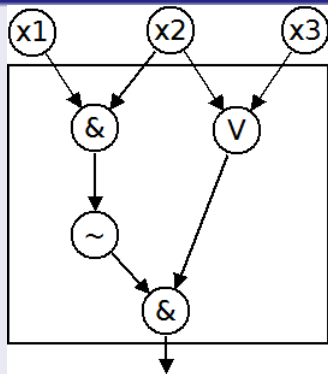
Példa



$(g_1 \leftrightarrow x_1)$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

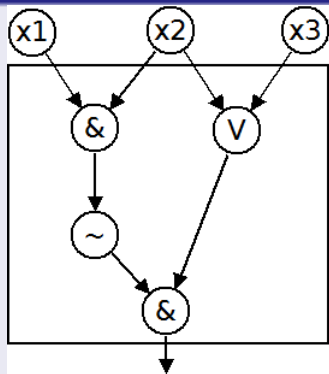
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2)$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

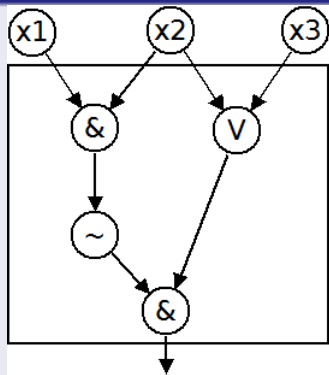
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

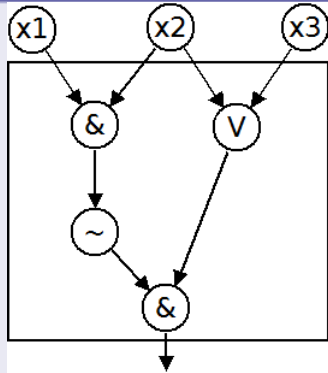
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

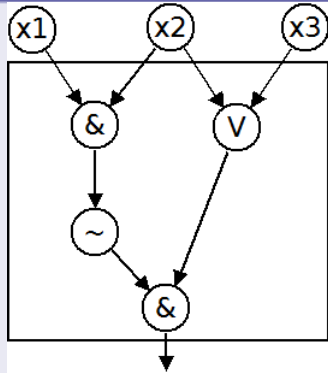
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

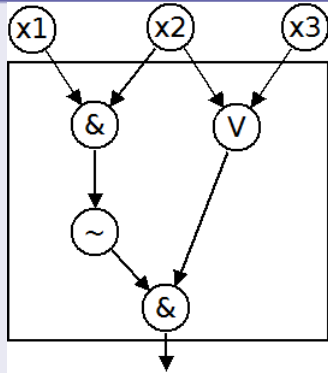
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

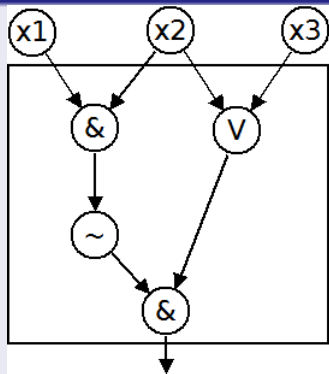
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

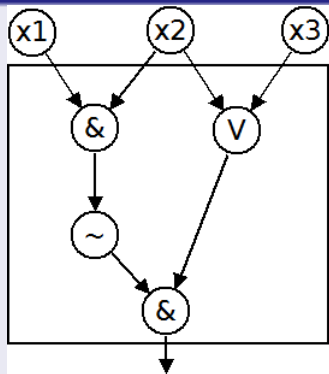
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge \\ (g_6 \leftrightarrow$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

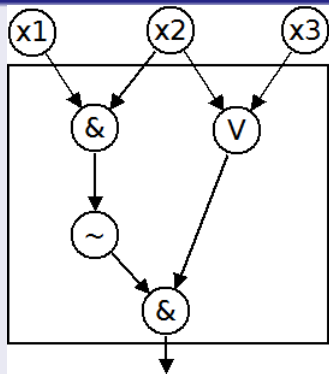
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge \\ (g_6 \leftrightarrow (\neg g_4)) \wedge$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

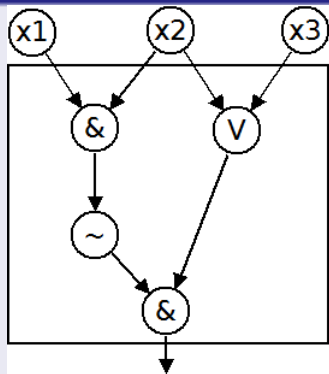
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge \\ (g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge$$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

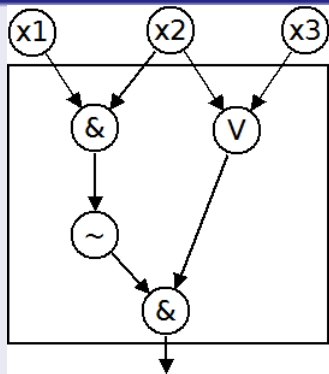
Példa



$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge \\ (g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$$

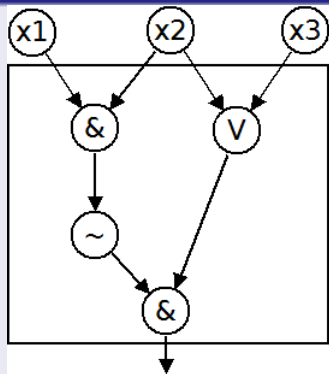
HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

Példa



$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3)$
 $\wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge$
 $(g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$
CNF alakban:
 $(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1)$

Példa

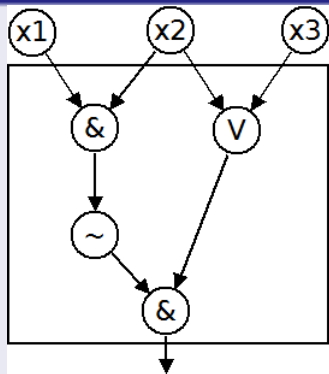


$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge \\ (g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$$

CNF alakban:

$$(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1) \wedge (\neg g_2 \vee x_2) \wedge \\ (g_2 \vee \neg x_2) \wedge (\neg g_3 \vee x_3) \wedge (g_3 \vee \neg x_3)$$

Példa

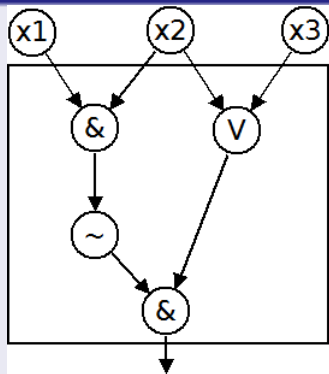


$$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3) \\ \wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge \\ (g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$$

CNF alakban:

$$(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1) \wedge (\neg g_2 \vee x_2) \wedge \\ (g_2 \vee \neg x_2) \wedge (\neg g_3 \vee x_3) \wedge (g_3 \vee \neg x_3) \wedge \\ (\neg g_4 \vee g_1) \wedge (\neg g_4 \vee g_2) \wedge (g_4 \vee \neg g_1 \vee \neg g_2)$$

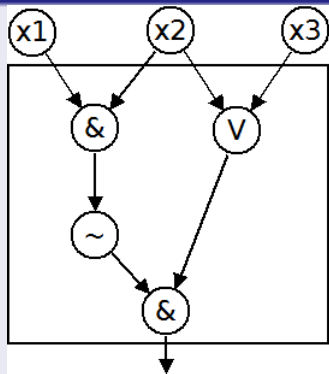
Példa



$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3)$
 $\wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge$
 $(g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$
 CNF alakban:

$(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1) \wedge (\neg g_2 \vee x_2) \wedge$
 $(g_2 \vee \neg x_2) \wedge (\neg g_3 \vee x_3) \wedge (g_3 \vee \neg x_3) \wedge$
 $(\neg g_4 \vee g_1) \wedge (\neg g_4 \vee g_2) \wedge (g_4 \vee \neg g_1 \vee$
 $\neg g_2) \wedge (\neg g_5 \vee g_2 \vee g_3) \wedge (g_5 \vee \neg g_2) \wedge (g_5 \vee$
 $\neg g_3)$

Példa

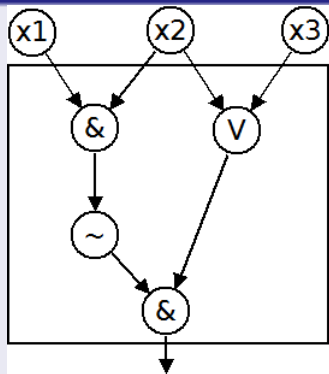


$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3)$
 $\wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge$
 $(g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$
 CNF alakban:

$(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1) \wedge (\neg g_2 \vee x_2) \wedge$
 $(g_2 \vee \neg x_2) \wedge (\neg g_3 \vee x_3) \wedge (g_3 \vee \neg x_3) \wedge$
 $(\neg g_4 \vee g_1) \wedge (\neg g_4 \vee g_2) \wedge (g_4 \vee \neg g_1 \vee$
 $\neg g_2) \wedge (\neg g_5 \vee g_2 \vee g_3) \wedge (g_5 \vee \neg g_2) \wedge (g_5 \vee$
 $\neg g_3) \wedge (\neg g_6 \vee \neg g_4) \wedge (g_6 \vee g_4)$

HÁLÓZAT-KIELÉGÍTHETŐSÉG $\leq_P$ SAT példa

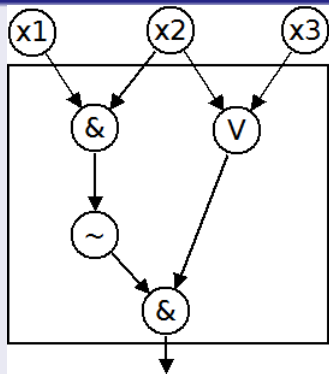
Példa



$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3)$
 $\wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge$
 $(g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$
CNF alakban:

$(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1) \wedge (\neg g_2 \vee x_2) \wedge$
 $(g_2 \vee \neg x_2) \wedge (\neg g_3 \vee x_3) \wedge (g_3 \vee \neg x_3) \wedge$
 $(\neg g_4 \vee g_1) \wedge (\neg g_4 \vee g_2) \wedge (g_4 \vee \neg g_1 \vee$
 $\neg g_2) \wedge (\neg g_5 \vee g_2 \vee g_3) \wedge (g_5 \vee \neg g_2) \wedge (g_5 \vee$
 $\neg g_3) \wedge (\neg g_6 \vee \neg g_4) \wedge (g_6 \vee g_4) \wedge (\neg g_7 \vee$
 $g_5) \wedge (\neg g_7 \vee g_6) \wedge (g_7 \vee \neg g_5 \vee \neg g_6)$

Példa



$(g_1 \leftrightarrow x_1) \wedge (g_2 \leftrightarrow x_2) \wedge (g_3 \leftrightarrow x_3)$
 $\wedge (g_4 \leftrightarrow (g_1 \wedge g_2)) \wedge (g_5 \leftrightarrow (g_2 \vee g_3)) \wedge$
 $(g_6 \leftrightarrow (\neg g_4)) \wedge (g_7 \leftrightarrow (g_5 \wedge g_6)) \wedge g_7.$
 CNF alakban:

$(\neg g_1 \vee x_1) \wedge (g_1 \vee \neg x_1) \wedge (\neg g_2 \vee x_2) \wedge$
 $(g_2 \vee \neg x_2) \wedge (\neg g_3 \vee x_3) \wedge (g_3 \vee \neg x_3) \wedge$
 $(\neg g_4 \vee g_1) \wedge (\neg g_4 \vee g_2) \wedge (g_4 \vee \neg g_1 \vee$
 $\neg g_2) \wedge (\neg g_5 \vee g_2 \vee g_3) \wedge (g_5 \vee \neg g_2) \wedge (g_5 \vee$
 $\neg g_3) \wedge (\neg g_6 \vee \neg g_4) \wedge (g_6 \vee g_4) \wedge (\neg g_7 \vee$
 $g_5) \wedge (\neg g_7 \vee g_6) \wedge (g_7 \vee \neg g_5 \vee \neg g_6) \wedge g_7.$

Első kiszámítható teljes problémáink

Emlékezzünk: egy $\mathcal{C}$ -beli probléma $\mathcal{C}$ -teljes, ha **minden** $\mathcal{C}$ -beli probléma visszavezethető rá.

Első kiszámítható teljes problémáink

Emlékezzünk: egy $\mathcal{C}$ -beli probléma $\mathcal{C}$ -teljes, ha **minden** $\mathcal{C}$ -beli probléma visszavezethető rá.

Eddig még csak eldönthetetlen, **RE**-teljes problémákkal találkoztunk: a MEGÁLLÁS ilyen volt.

Első kiszámítható teljes problémáink

Emlékezzünk: egy $\mathcal{C}$ -beli probléma $\mathcal{C}$ -teljes, ha **minden** $\mathcal{C}$ -beli probléma visszavezethető rá.

Eddig még csak eldönthetetlen, **RE**-teljes problémákkal találkoztunk: a MEGÁLLÁS ilyen volt.

A HÁLÓZAT-KIÉRTÉKELES probléma **P**-beli.

Első kiszámítható teljes problémáink

Emlékezzünk: egy $\mathcal{C}$ -beli probléma $\mathcal{C}$ -teljes, ha **minden** $\mathcal{C}$ -beli probléma visszavezethető rá.

Eddig még csak eldönthetetlen, **RE**-teljes problémákkal találkoztunk: a MEGÁLLÁS ilyen volt.

A HÁLÓZAT-KIÉRTÉKELES probléma **P**-beli.

Tétel

A HÁLÓZAT-KIELÉGÍTHETŐSÉG probléma **NP**-teljes.

Első kiszámítható teljes problémáink

Emlékezzünk: egy $\mathcal{C}$ -beli probléma $\mathcal{C}$ -teljes, ha **minden** $\mathcal{C}$ -beli probléma visszavezethető rá.

Eddig még csak eldönthetetlen, **RE**-teljes problémákkal találkoztunk: a MEGÁLLÁS ilyen volt.

A HÁLÓZAT-KIÉRTÉKELÉS probléma **P**-beli.

Tétel

A HÁLÓZAT-KIELÉGÍTHETŐSÉG probléma **NP**-teljes.

Következmény (Cook tétele)

A SAT probléma is **NP**-teljes.

SAT **NP**-teljességéről

Mivel a SAT probléma **NP**-teljes,

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.

SAT **NP**-teljességéről

Mivel a SAT probléma **NP**-teljes,

- ha polinomidőben megoldható lenne, akkor $\mathbf{P} = \mathbf{NP}$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):

SAT NP-teljeségről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):
 - **heurisztikák**

SAT NP-teljeségről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):
 - **heurisztikák**
 - **randomizált algoritmusok** alkalmazása

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):
 - **heurisztikák**
 - **randomizált algoritmusok** alkalmazása
 - a követelmények **relaxálása**. . .

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):
 - **heurisztikák**
 - **randomizált algoritmusok** alkalmazása
 - a követelmények **relaxálása. . .** (pl. nem az összes, de minél több klóz egyszerre történő kielégítése)

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):
 - **heurisztikák**
 - **randomizált algoritmusok** alkalmazása
 - a követelmények **relaxálása. . .** (pl. nem az összes, de minél több klóz egyszerre történő kielégítése)
 - . . . majd a relaxált követelmények **approximálása**

SAT NP-teljességéről

Mivel a SAT probléma NP-teljes,

- ha polinomidőben megoldható lenne, akkor $P = NP$.
(Ebben persze lehet **hinni**. De eddig még senki nem oldotta meg polinomidőben. . .)
- Nem ismert rá **szubexponenciális** algoritmus.
- Ilyenkor bevethetők (a teljesség igénye nélkül):
 - **heurisztikák**
 - **randomizált algoritmusok** alkalmazása
 - a követelmények **relaxálása. . .** (pl. nem az összes, de minél több klóz egyszerre történő kielégítése)
 - . . . majd a relaxált követelmények **approximálása**
 - vagy olyan **speciális esetek** keresése, melyre van hatékony algoritmus.

A kielégíthetőség változatai

A SAT problémának vizsgálhatjuk **speciális eseteit**, pl:

A kielégíthetőség változatai

A SAT problémának vizsgálhatjuk **speciális eseteit**, pl:

- klózonként csak **konstans sok** literált engedünk meg

A kielégíthetőség változatai

A SAT problémának vizsgálhatjuk **speciális eseteit**, pl:

- klózonként csak **konstans sok** literált engedünk meg
- csak speciális (pl. **Horn**) alakú klózokat engedünk meg

A kielégíthetőség változatai

A SAT problémának vizsgálhatjuk **speciális eseteit**, pl:

- klózonként csak **konstans sok** literált engedünk meg
- csak speciális (pl. **Horn**) alakú klózokat engedünk meg

Definíció

Legyen $k \geq 1$. A k SAT a SAT azon speciális esete, amelyben minden klóz pontosan k (nem feltétlenül különböző) literálból áll.

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Bizonyítás

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Bizonyítás

$$\ell \mapsto \ell \vee \ell \vee \ell$$

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Bizonyítás

$$\begin{aligned} \ell &\mapsto \ell \vee \ell \vee \ell \\ \ell_1 \vee \ell_2 &\mapsto \ell_1 \vee \ell_2 \vee \ell_2 \end{aligned}$$

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Bizonyítás

$$\ell \mapsto \ell \vee \ell \vee \ell$$

$$\ell_1 \vee \ell_2 \mapsto \ell_1 \vee \ell_2 \vee \ell_2$$

$$\ell_1 \vee \ell_2 \vee \ell_3 \mapsto \ell_1 \vee \ell_2 \vee \ell_3$$

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Bizonyítás

$$\ell \mapsto \ell \vee \ell \vee \ell$$

$$\ell_1 \vee \ell_2 \mapsto \ell_1 \vee \ell_2 \vee \ell_2$$

$$\ell_1 \vee \ell_2 \vee \ell_3 \mapsto \ell_1 \vee \ell_2 \vee \ell_3$$

$$\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4 \mapsto (\ell_1 \vee \ell_2 \vee x) \wedge (\neg x \vee \ell_3 \vee \ell_4)$$

Állítás

3SAT **NP**-teljes, és így $k \geq 3$ esetén k SAT **NP**-teljes.

Bizonyítás

$$\begin{aligned}
 \ell &\mapsto \ell \vee \ell \vee \ell \\
 \ell_1 \vee \ell_2 &\mapsto \ell_1 \vee \ell_2 \vee \ell_2 \\
 \ell_1 \vee \ell_2 \vee \ell_3 &\mapsto \ell_1 \vee \ell_2 \vee \ell_3 \\
 \ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4 &\mapsto (\ell_1 \vee \ell_2 \vee x) \wedge (\neg x \vee \ell_3 \vee \ell_4) \\
 &\vdots \\
 \ell_1 \vee \ell_2 \vee \dots \vee \ell_n &\mapsto (\ell_1 \vee \ell_2 \vee x) \wedge (\neg x \vee \ell_3 \vee y) \wedge \\
 &\quad (\neg y \vee \ell_4 \vee z) \wedge \dots \wedge (\neg u \vee \ell_{n-1} \vee \ell_n)
 \end{aligned}$$

Állítás

Legyen $\varphi = c_1 \wedge c_2 \wedge \dots \wedge c_k$ konjunktív normálformájú formula.

Állítás

Legyen $\varphi = c_1 \wedge c_2 \wedge \dots \wedge c_k$ konjunktív normálformájú formula.
Minden c_i klózhoz legyen c'_i az előzőekben adott kifejezés, ahol minden kifejezésben új változókat használunk.

Állítás

Legyen $\varphi = c_1 \wedge c_2 \wedge \dots \wedge c_k$ konjunktív normálformájú formula.
Minden c_i klózhoz legyen c'_i az előzőekben adott kifejezés, ahol minden kifejezésben új változókat használunk.
Ekkor φ akkor és csak akkor kielégíthető, ha $\varphi' = c'_1 \wedge c'_2 \wedge \dots \wedge c'_k$ az.

Állítás

Legyen $\varphi = c_1 \wedge c_2 \wedge \dots \wedge c_k$ konjunktív normálformájú formula.
Minden c_i klózhoz legyen c'_i az előzőekben adott kifejezés, ahol minden kifejezésben új változókat használunk.
Ekkor φ akkor és csak akkor kielégíthető, ha $\varphi' = c'_1 \wedge c'_2 \wedge \dots \wedge c'_k$ az.

Mivel $\text{SAT} \leq_P 3\text{SAT}$ és $3\text{SAT} \in \mathbf{NP}$, ezért SAT $\mathbf{NP}$ -teljességéből következik, hogy 3SAT is $\mathbf{NP}$ -teljes.

A 2SAT problémára viszont adható hatékony algoritmus.

A 2SAT problémára viszont adható hatékony algoritmus.
Legyen φ egy 2 – CNF (minden klóz két literálból áll).

A 2SAT problémára viszont adható hatékony algoritmus.
Legyen φ egy 2 – CNF (minden klóz két literálból áll).

A $G(\varphi)$ gráf

A 2SAT problémára viszont adható hatékony algoritmus.
Legyen φ egy 2 – CNF (minden klóz két literálból áll).

A $G(\varphi)$ gráf

- csúcsok: a φ -ben előforduló változók és negáltjaik.

A 2SAT problémára viszont adható hatékony algoritmus.
Legyen φ egy 2 – CNF (minden klóz két literálból áll).

A $G(\varphi)$ gráf

- csúcsok: a φ -ben előforduló változók és negáltjaik.
- élek: (α, β) él $\Leftrightarrow \bar{\alpha} \vee \beta$ vagy $\beta \vee \bar{\alpha}$ a φ tagja (azaz ha $\alpha \rightarrow \beta$ vagy $\bar{\beta} \rightarrow \bar{\alpha}$ a φ tagja.)

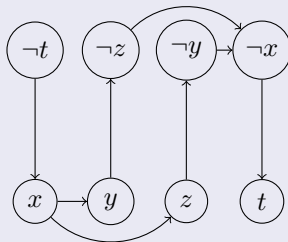
Példa

Legyen $\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t)$.

Példa

Legyen $\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t)$.

Ekkor $G(\varphi)$:



Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Így ha ℓ_1 -ből ℓ_2 elérhető, akkor $\varphi \models (\ell_1 \rightarrow \ell_2)$.

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Így ha ℓ_1 -ből ℓ_2 elérhető, akkor $\varphi \models (\ell_1 \rightarrow \ell_2)$.

Ha x -ből is elérhető $\neg x$ és fordítva, akkor $\varphi \models (x \leftrightarrow \neg x)$, így φ kielégíthetetlen.

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Így ha ℓ_1 -ből ℓ_2 elérhető, akkor $\varphi \models (\ell_1 \rightarrow \ell_2)$.

Ha x -ből is elérhető $\neg x$ és fordítva, akkor $\varphi \models (x \leftrightarrow \neg x)$, így φ kielégíthetetlen.

Ha nincs ilyen változó, akkor ciklusban:

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Így ha ℓ_1 -ből ℓ_2 elérhető, akkor $\varphi \models (\ell_1 \rightarrow \ell_2)$.

Ha x -ből is elérhető $\neg x$ és fordítva, akkor $\varphi \models (x \leftrightarrow \neg x)$, így φ kielégíthetetlen.

Ha nincs ilyen változó, akkor ciklusban:

- válasszunk egy olyan ℓ literált, amiből nem érhető el $\bar{\ell}$ és melynek még nem adtunk értéket;

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Így ha ℓ_1 -ből ℓ_2 elérhető, akkor $\varphi \models (\ell_1 \rightarrow \ell_2)$.

Ha x -ből is elérhető $\neg x$ és fordítva, akkor $\varphi \models (x \leftrightarrow \neg x)$, így φ kielégíthetetlen.

Ha nincs ilyen változó, akkor ciklusban:

- válasszunk egy olyan ℓ literált, amiből nem érhető el $\bar{\ell}$ és melynek még nem adtunk értéket;
- állítsuk 1-re ℓ -t és minden belőle elérhető literált, komplementereiket pedig nullára;

Tétel

A φ formula akkor és csak akkor kielégíthető, ha nincs olyan x változó, amelyre létezik $G(\varphi)$ -ben irányított út x -ből $\neg x$ -be és vissza.

Ötlet

Minden él egy implikációnak felel meg.

Így ha ℓ_1 -ből ℓ_2 elérhető, akkor $\varphi \models (\ell_1 \rightarrow \ell_2)$.

Ha x -ből is elérhető $\neg x$ és fordítva, akkor $\varphi \models (x \leftrightarrow \neg x)$, így φ kielégíthetetlen.

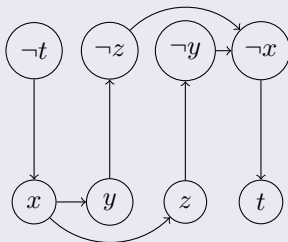
Ha nincs ilyen változó, akkor ciklusban:

- válasszunk egy olyan ℓ literált, amiből nem érhető el $\bar{\ell}$ és melynek még nem adtunk értéket;
- állítsuk 1-re ℓ -t és minden belőle elérhető literált, komplementereiket pedig nullára;

amíg minden változónak értéket nem adtunk (nem lesz ütközés, ha nincs olyan x , akiből $\neg x$ elérhető és viszont). Az így kapott értékelés kielégíti φ -t.

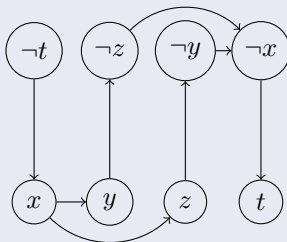
Példa

$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



Példa

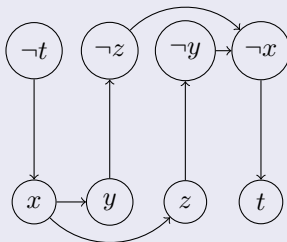
$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



Válasszuk először x -et.

Példa

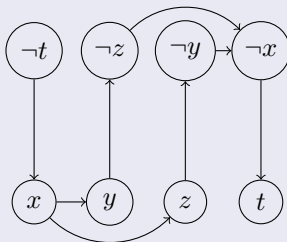
$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



Válasszuk először x -et. $G(\varphi)$ -ben x -ből elérhető $\neg x$.

Példa

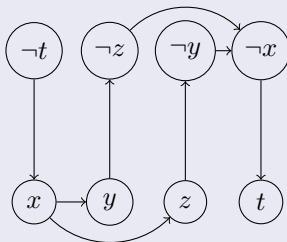
$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



Válasszuk először x -et. $G(\varphi)$ -ben x -ből elérhető $\neg x$.
igaz-ra állítjuk $\neg x$ -et és t -t, hamis-ra x -et és $\neg t$ -t.

Példa

$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



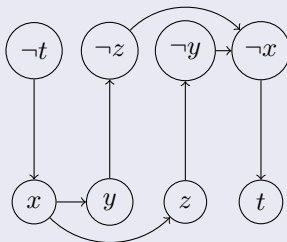
Válasszuk először x -et. $G(\varphi)$ -ben x -ből elérhető $\neg x$.

igaz-ra állítjuk $\neg x$ -et és t -t, hamis-ra x -et és $\neg t$ -t.

Ezek után válasszuk mondjuk y -t. y -ból nem érhető el $\neg y$, tehát $\alpha := y$.

Példa

$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



Válasszuk először x -et. $G(\varphi)$ -ben x -ből elérhető $\neg x$.

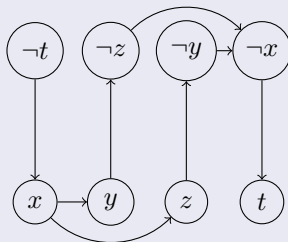
igaz-ra állítjuk $\neg x$ -et és t -t, hamis-ra x -et és $\neg t$ -t.

Ezek után válasszuk mondjuk y -t. y -ból nem érhető el $\neg y$, tehát $\alpha := y$.

igaz-ra állítjuk y -t és $\neg z$ -t, hamis-ra pedig $\neg y$ -t és z -t.

Példa

$$\varphi = (\neg x \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (x \vee t).$$



Válasszuk először x -et. $G(\varphi)$ -ben x -ből elérhető $\neg x$.

igaz-ra állítjuk $\neg x$ -et és t -t, hamis-ra x -et és $\neg t$ -t.

Ezek után válasszuk mondjuk y -t. y -ból nem érhető el $\neg y$, tehát $\alpha := y$.

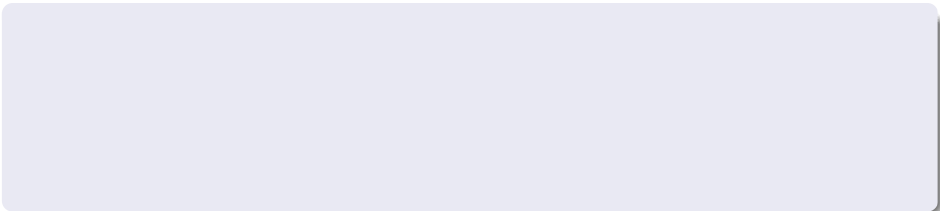
igaz-ra állítjuk y -t és $\neg z$ -t, hamis-ra pedig $\neg y$ -t és z -t.

A kapott értékekadás kielégíti φ -t.

Következmény

$2\text{SAT} \in \mathbf{P}$.

Horn-formulák és Horn-átnevezhető formulák



Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.
Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Egy CNF **Horn-átnevezhető**, ha bizonyos változói komplementálásával Horn-formulává tehető

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Egy CNF **Horn-átnevezhető**, ha bizonyos változói komplementálásával Horn-formulává tehető (vagyis ha van változók egy olyan X halmaza, hogy minden $x \in X$ -re x helyébe $\neg x$ -et, $\neg x$ helyébe x -et írva a formulába, az eredmény Horn-formula lesz).

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Egy CNF **Horn-átnevezhető**, ha bizonyos változói komplementálásával Horn-formulává tehető (vagyis ha van változók egy olyan X halmaza, hogy minden $x \in X$ -re x helyébe $\neg x$ -et, $\neg x$ helyébe x -et írva a formulába, az eredmény Horn-formula lesz).

Példa

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Egy CNF **Horn-átnevezhető**, ha bizonyos változói komplementálásával Horn-formulává tehető (vagyis ha van változók egy olyan X halmaza, hogy minden $x \in X$ -re x helyébe $\neg x$ -et, $\neg x$ helyébe x -et írva a formulába, az eredmény Horn-formula lesz).

Példa

- $x \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$ Horn-formula.

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Egy CNF **Horn-átnevezhető**, ha bizonyos változói komplementálásával Horn-formulává tehető (vagyis ha van változók egy olyan X halmaza, hogy minden $x \in X$ -re x helyébe $\neg x$ -et, $\neg x$ helyébe x -et írva a formulába, az eredmény Horn-formula lesz).

Példa

- $x \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$ Horn-formula.
- $(x \vee y) \wedge (\neg x \vee \neg y)$ Horn-átnevezhető.

Horn-formulák és Horn-átnevezhető formulák

Egy klóz **Horn-klóz**, ha benne maximum egy pozitív literál szerepel.

Egy CNF **Horn-formula**, ha benne minden klóz Horn-klóz.

Egy CNF **Horn-átnevezhető**, ha bizonyos változói komplementálásával Horn-formulává tehető (vagyis ha van változók egy olyan X halmaza, hogy minden $x \in X$ -re x helyébe $\neg x$ -et, $\neg x$ helyébe x -et írva a formulába, az eredmény Horn-formula lesz).

Példa

- $x \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$ Horn-formula.
- $(x \vee y) \wedge (\neg x \vee \neg y)$ Horn-átnevezhető.
- $(\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (x \vee y)$ **nem** Horn-átnevezhető.

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

- amíg találunk egy egyetlen literálból álló $\{\ell\}$ klózt:

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

- amíg találunk egy egyetlen literálból álló $\{\ell\}$ klózt:
- ℓ értékét 1-re állítjuk;

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

- amíg találunk egy egyetlen literálból álló $\{l\}$ klózt;
- l értékét 1-re állítjuk;
- töröljük az összes, l -t tartalmazó klózt;

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

- amíg találunk egy egyetlen literálból álló $\{\ell\}$ klózt;
- ℓ értékét 1-re állítjuk;
- töröljük az összes, ℓ -t tartalmazó klózt;
- a maradék klózokból töröljük az $\bar{\ell}$ literált.

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

- amíg találunk egy egyetlen literálból álló $\{\ell\}$ klózt;
- ℓ értékét 1-re állítjuk;
- töröljük az összes, ℓ -t tartalmazó klózt;
- a maradék klózokból töröljük az $\bar{\ell}$ literált.

Ha közben megkapjuk az üres klózt, a formula kielégíthetetlen; ellenkező esetben kielégíthető.

Horn-formulák és Horn-átnevezhető formulák

Tétel

Horn-átnevezhető formulák kielégíthetősége polinom időben eldönthető.

Ötlet

Az **egységrezolúció** alkalmazása ezekre a formulákra teljes:

- amíg találunk egy egyetlen literálból álló $\{\ell\}$ klózt;
- ℓ értékét 1-re állítjuk;
- töröljük az összes, ℓ -t tartalmazó klózt;
- a maradék klózokból töröljük az $\bar{\ell}$ literált.

Ha közben megkapjuk az üres klózt, a formula kielégíthetetlen; ellenkező esetben kielégíthető.

Az algoritmus (ésszerű reprezentációval) lineáris időben végrehajtható és Horn-átnevezhető formulákra helyes.

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Megjegyzés

„Keverni” nem tudjuk a két hatékonyan eldönthető speciális esetet:

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Megjegyzés

„Keverni” nem tudjuk a két hatékonyan eldönthető speciális esetet:
NP-teljes a következő probléma: adott egy olyan CNF, melyben minden klóz

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Megjegyzés

„Keverni” nem tudjuk a két hatékonyan eldönthető speciális esetet:
NP-teljes a következő probléma: adott egy olyan CNF, melyben minden klóz

- vagy max. két literálból áll,

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Megjegyzés

„Keverni” nem tudjuk a két hatékonyan eldönthető speciális esetet:
NP-teljes a következő probléma: adott egy olyan CNF, melyben minden klóz

- vagy max. két literálból áll,
- vagy háromelemű negatív klóz,

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Megjegyzés

„Keverni” nem tudjuk a két hatékonyan eldönthető speciális esetet:
NP-teljes a következő probléma: adott egy olyan CNF, melyben minden klóz

- vagy max. két literálból áll,
- vagy háromelemű negatív klóz,

kielégíthető-e?

Horn-formulák és Horn-átnevezhető formulák

Megjegyzés

Az is polinom időben eldönthető egy CNF-ről, hogy Horn-átnevezhető-e.

Megjegyzés

„Keverni” nem tudjuk a két hatékonyan eldönthető speciális esetet:
NP-teljes a következő probléma: adott egy olyan CNF, melyben minden klóz

- vagy max. két literálból áll,
- vagy háromelemű negatív klóz,

kielégíthető-e?

Ötlet: $3SAT \leq ez$

Változónként $(x \vee \hat{x}) \wedge (\neg x \vee \neg \hat{x})$ és ehhez hozzá minden eredeti klózban a pozitív x literálokat $\neg \hat{x}$ -re cseréljük

Az előfordulásszám csökkentése

Azt is próbálhatjuk korlátozni, hogy egy változó (vagy egy literál) hányszor fordulhat elő legfeljebb a formulában.

Az előfordulásszám csökkentése

Azt is próbálhatjuk korlátozni, hogy egy változó (vagy egy literál) hányszor fordulhat elő legfeljebb a formulában.

Tétel

NP-teljes a következő probléma: adott egy 3CNF, melyben minden változó legfeljebb háromszor szerepel, kielégíthető-e?

Az előfordulásszám csökkentése

Azt is próbálhatjuk korlátozni, hogy egy változó (vagy egy literál) hányszor fordulhat elő legfeljebb a formulában.

Tétel

NP-teljes a következő probléma: adott egy 3CNF, melyben minden változó legfeljebb háromszor szerepel, kielégíthető-e?

P-ben van a következő probléma: adott egy CNF, melyben minden változó legfeljebb kétszer szerepel, kielégíthető-e?

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

Bizonyítás

Világos, hogy a probléma NP-ben van.

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

Bizonyítás

Világos, hogy a probléma NP-ben van. Megmutatjuk, hogy 3SAT visszavezethető a FÜGGETLEN CSÚCSHALMAZ-ra.

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

Bizonyítás

Világos, hogy a probléma NP-ben van. Megmutatjuk, hogy 3SAT visszavezethető a FÜGGETLEN CSÚCSHALMAZ-ra.

Legyen φ a 3SAT egy példánya, $\varphi = c_1 \wedge \dots \wedge c_m$.

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

Bizonyítás

Világos, hogy a probléma NP-ben van. Megmutatjuk, hogy 3SAT visszavezethető a FÜGGETLEN CSÚCSHALMAZ-ra.

Legyen φ a 3SAT egy példánya, $\varphi = c_1 \wedge \dots \wedge c_m$. A G gráf álljon m darab **háromszögből**, melyek a c_i tagoknak felelnek meg, s melyek csúcsai a c_i -kben lévő literáloknak felelnek meg. Ezen kívül még kössünk össze éllel **minden ellentétes literált**. Végül legyen $K = m$.

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcshalmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

Bizonyítás

Világos, hogy a probléma NP-ben van. Megmutatjuk, hogy 3SAT visszavezethető a FÜGGETLEN CSÚCSHALMAZ-ra.

Legyen φ a 3SAT egy példánya, $\varphi = c_1 \wedge \dots \wedge c_m$. A G gráf álljon m darab **háromszögből**, melyek a c_i tagoknak felelnek meg, s melyek csúcsai a c_i -kben lévő literáloknak felelnek meg. Ezen kívül még kössünk össze éllel **minden ellentétes literált**. Végül legyen $K = m$.

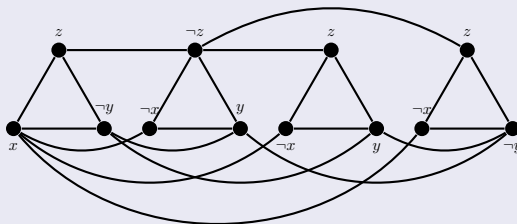
φ kielégíthető $\Leftrightarrow G$ -ben létezik K elemű független csúcshalmaz.

Példa

Legyen $\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$.

Példa

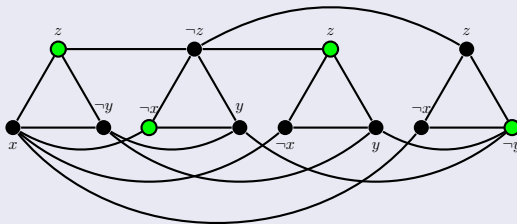
Legyen $\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$.
Ekkor a G gráf:



és $K = 4$.

Példa

Legyen $\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$.
Ekkor a G gráf:

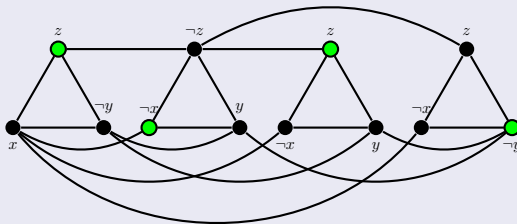


és $K = 4$.

A zölddel jelölt csúcsok egy négyelemű független csúcshalmazt alkotnak;

Példa

Legyen $\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$.
Ekkor a G gráf:



és $K = 4$.

A zölddel jelölt csúcsok egy négyelemű független csúcshalmazt alkotnak; egy kielégítő kiértékelés pedig: $x = y = 0, z = 1$.

NP-teljes gráfelméleti problémák

KLIKK

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű klikk (teljes részgráf)?

NP-teljes gráfelméleti problémák

KLIKK

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű klikk (teljes részgráf)?

Tétel

KLIKK NP-teljes.

NP-teljes gráfelméleti problémák

KLIKK

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű klikk (teljes részgráf)?

Tétel

KLIKK NP-teljes.

Bizonyítás (visszavezetéssel): gyakorlaton.

CSÚCSLEFEDÉS

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e olyan K elemű csúcshalmaz, hogy minden él illeszkedik a halmaz egy csúcsához?

NP-teljes gráfelméleti problémák

CSÚCSLEFEDÉS

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e olyan K elemű csúcshalmaz, hogy minden él illeszkedik a halmaz egy csúcsához?

Tétel

A CSÚCSLEFEDÉS probléma NP-teljes.

NP-teljes gráfelméleti problémák

CSÚCSLEFEDÉS

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e olyan K elemű csúcshalmaz, hogy minden él illeszkedik a halmaz egy csúcsához?

Tétel

A CSÚCSLEFEDÉS probléma NP-teljes.

Bizonyítás (visszavezetéssel): gyakorlaton.

Tétel

A HAMILTON-ÚT probléma NP-teljes.

NP-teljes gráfelméleti problémák

Tétel

A HAMILTON-ÚT probléma NP-teljes.

Ötlet

„Értékválasztó” gadget:

NP-teljes gráfelméleti problémák

Tétel

A HAMILTON-ÚT probléma NP-teljes.

Ötlet

„Értékválasztó” gadget:



NP-teljes gráfelméleti problémák

Tétel

A HAMILTON-ÚT probléma NP-teljes.

Ötlet

„Értékválasztó” gadget:

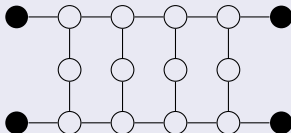


(Balról jobbra ha megy egy Hamilton-út, akkor **választanunk** kell a „fenti” és a „lenti” útvonal közül egyet.)

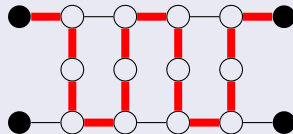
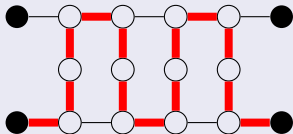
⇒ mint egy értékadás – ilyen gadgetből változónként lesz egy

Ötlet

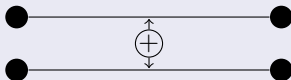
„XOR” gadget:



Minden Hamilton-út egy olyan gráfban, ami ezt a gadgetot **feszített részgráfként** tartalmazza, a következők egyike ezen a gráfon belül:



Hogy átláthatóbb legyen a rajzunk, ezt a gadgetet így rajzoljuk:



Ötlet

A XOR gadgettel egy gráfban olyan Hamilton-kört keresünk, melyben **ki tudjuk kötni bizonyos élpárokra**, hogy a két él közül a keresett Hamilton-kör **pontosan egyen** haladjon át.

A teljes konstrukció a $3SAT \leq HAMILTON-ÚT$ visszavezetéséhez:

- minden változóhoz létrehozunk egy **értékválasztó gadgetet**, az x_i -hez tartozó két párhuzamos élt x_i -vel ill. $\neg x_i$ -vel címkézzük;
- minden klózhoz létrehozunk egy **háromszöget**, az $\ell_1 \vee \ell_2 \vee \ell_3$ klózhoz létrehozott háromszög éleit rendre ℓ_1 -gyel, ℓ_2 -vel és ℓ_3 -mal címkézzük;
- az ellentétes literálokkal címkézett élek között XOR gadgetet hozunk létre;
- az értékválasztó gadgeteket láncba kötjük;
- a háromszögek csúcsaiból, az utolsó értékválasztó csúcsból és még egy plusz csúcsból pedig egyetlen nagy klikket készítünk;
- végül az előző pontbeli plusz csúcsból még egy új csúcsba lépünk.

Ez a konstrukció egy $3SAT \leq HAMILTON-ÚT$ visszavezetés lesz.

Így, mivel $\text{HAMILTON-ÚT} \leq_P \text{TSP}(E)$, így a $\text{TSP}(E)$ probléma **NP**-teljes.

Tétel

A 3-SZÍNEZÉS probléma **NP**-teljes.

Így, mivel $\text{HAMILTON-ÚT} \leq_{\mathcal{P}} \text{TSP}(E)$, így a $\text{TSP}(E)$ probléma **NP**-teljes.

Tétel

A 3-SZÍNEZÉS probléma **NP**-teljes.

Megjegyzés

Így, mivel $\text{HAMILTON-ÚT} \leq_P \text{TSP}(E)$, így a $\text{TSP}(E)$ probléma **NP**-teljes.

Tétel

A 3-SZÍNEZÉS probléma **NP**-teljes.

Megjegyzés

A 2-SZÍNEZÉS probléma **P**-ben van.

Így, mivel $\text{HAMILTON-ÚT} \leq_P \text{TSP}(E)$, így a $\text{TSP}(E)$ probléma **NP**-teljes.

Tétel

A 3-SZÍNEZÉS probléma **NP**-teljes.

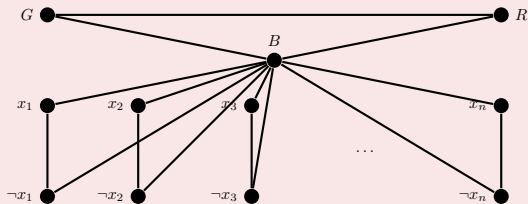
Megjegyzés

A 2-SZÍNEZÉS probléma **P**-ben van.

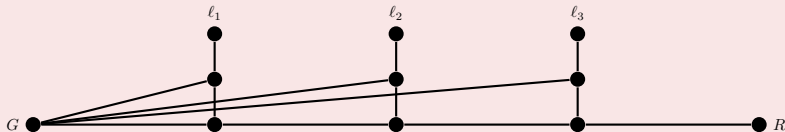
A 4-SZÍNEZÉS probléma **triviális** síkbarajzolható gráfokra.

NP-teljes problémák halmazokra és számokra

3SAT $\leq$ 3-SZÍNEZÉS: ötlet



így minden literál színe R („hamis”) vagy G („igaz”) színével kell megegyezzen, komplementere pedig egy másikkal $\Rightarrow$ kiértékelés



Ellenőrizhető: ha mindhárom literál R színét kapja, a plusz hat csúcsot nem lehet helyesen 3-színezní, ha viszont van köztük G színű, akkor igen

HÁRMASÍTÁS

Adott: Három azonos méretű halmaz, F (fiúk), L (lányok), H (házak), és egy $R \subseteq F \times L \times H$ reláció.

Kérdés: Megadható-e az R -beli hármasok egy olyan részhalmaza, melyben minden fiú, lány és ház pontosan egyszer szerepel?

NP-teljes problémák halmazokra és számokra

HÁRMASÍTÁS

Adott: Három azonos méretű halmaz, F (fiúk), L (lányok), H (házak), és egy $R \subseteq F \times L \times H$ reláció.

Kérdés: Megadható-e az R -beli hármasok egy olyan részhalmaza, melyben minden fiú, lány és ház pontosan egyszer szerepel?

Világos, hogy $\text{HÁRMASÍTÁS} \in \text{NP}$.

NP-teljes problémák halmazokra és számokra

HÁRMASÍTÁS

Adott: Három azonos méretű halmaz, F (fiúk), L (lányok), H (házak), és egy $R \subseteq F \times L \times H$ reláció.

Kérdés: Megadható-e az R -beli hármasok egy olyan részhalmaza, melyben minden fiú, lány és ház pontosan egyszer szerepel?

Világos, hogy $\text{HÁRMASÍTÁS} \in \text{NP}$.

Tétel

A HÁRMASÍTÁS probléma **NP**-teljes.

NP-teljes problémák halmazokra és számokra

HÁRMASÍTÁS

Adott: Három azonos méretű halmaz, F (fiúk), L (lányok), H (házak), és egy $R \subseteq F \times L \times H$ reláció.

Kérdés: Megadható-e az R -beli hármasok egy olyan részhalmaza, melyben minden fiú, lány és ház pontosan egyszer szerepel?

Világos, hogy HÁRMASÍTÁS $\in$ NP.

Tétel

A HÁRMASÍTÁS probléma NP-teljes.

Megjegyzés

PÁROSÍTÁS $\in$ P.

HALMAZLEFEDÉS

Adott: Egy U halmaz részhalmazainak $C = (S_1, \dots, S_n)$ rendszere és egy K szám.

Kérdés: Kiválasztható-e K darab az S_i -k közül úgy, hogy ezek egyesítése U ?

HALMAZLEFEDÉS

Adott: Egy U halmaz részhalmazainak $C = (S_1, \dots, S_n)$ rendszere és egy K szám.

Kérdés: Kiválasztható-e K darab az S_i -k közül úgy, hogy ezek egyesítése U ?

HALMAZPAKOLÁS

Adott: Egy U halmaz részhalmazainak $C = (S_1, \dots, S_n)$ rendszere és egy K szám.

Kérdés: Kiválasztható-e az S_i -k közül K darab, páronként diszjunkt halmaz?

HALMAZLEFEDÉS

Adott: Egy U halmaz részhalmazainak $C = (S_1, \dots, S_n)$ rendszere és egy K szám.

Kérdés: Kiválasztható-e K darab az S_i -k közül úgy, hogy ezek egyesítése U ?

HALMAZPAKOLÁS

Adott: Egy U halmaz részhalmazainak $C = (S_1, \dots, S_n)$ rendszere és egy K szám.

Kérdés: Kiválasztható-e az S_i -k közül K darab, páronként diszjunkt halmaz?

PONTOS LEFEDÉS HÁRMASOKKAL

Adott: Egy $3m$ elemű U halmaz és 3 -elemű részhalmazainak $(S_1, \dots, S_n)$ rendszere.

Kérdés: Kiválasztható-e m darab S_i úgy, hogy ezek egyesítése U ? (A kiválasztott S_i -k nyilván diszjunktak.)

NP-teljes problémák halmazokra

Tétel

A HALMAZLEFEDÉS, HALMAZPAKOLÁS, PONTOS LEFEDÉS HÁRMASOKKAL problémák NP-teljesek.

NP-teljes problémák halmazokra

Tétel

A HALMAZLEFEDÉS, HALMAZPAKOLÁS, PONTOS LEFEDÉS HÁRMASOKKAL problémák NP-teljesek.

Bizonyítás

A HÁRMASÍTÁS a Pontos LEFEDÉS HÁRMASOKKAL **speciális esete**.

NP-teljes problémák halmazokra

Tétel

A HALMAZLEFEDÉS, HALMAZPAKOLÁS, PONTOS LEFEDÉS HÁRMASOKKAL problémák NP-teljesek.

Bizonyítás

A HÁRMASÍTÁS a Pontos LEFEDÉS HÁRMASOKKAL **speciális esete**.
A PONTOS LEFEDÉS HÁRMASOKKAL a HALMAZLEFEDÉS, valamint a HALMAZPAKOLÁS speciális esete is.

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

Adott: Egy n -változós, egész együtthatós lineáris egyenlőtlenség-rendszer.

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

Adott: Egy n -változós, egész együtthatós lineáris egyenlőtlenség-rendszer.

Kérdés: Létezik-e **egész értékű** megoldás?

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

Adott: Egy n -változós, egész együtthatós lineáris egyenlőtlenség-rendszer.

Kérdés: Létezik-e **egész értékű** megoldás?

A HALMAZLEFEDÉS felfogható az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS speciális eseteként:

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

Adott: Egy n -változós, egész együtthatós lineáris egyenlőtlenség-rendszer.

Kérdés: Létezik-e **egész értékű** megoldás?

A HALMAZLEFEDÉS felfogható az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS speciális eseteként:

$$Ax \geq 1, \quad \sum_{i=1}^n x_i \leq B, \quad 0 \leq x_i \leq 1,$$

ahol A oszlopai a halmazrendszer elemeinek felelnek meg.

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

Adott: Egy n -változós, egész együtthatós lineáris egyenlőtlenség-rendszer.

Kérdés: Létezik-e **egész értékű** megoldás?

A HALMAZLEFEDÉS felfogható az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS speciális eseteként:

$$Ax \geq 1, \quad \sum_{i=1}^n x_i \leq B, \quad 0 \leq x_i \leq 1,$$

ahol A oszlopai a halmazrendszer elemeinek felelnek meg.

Azt is meg lehet mutatni, hogy az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS **NP**-ben van.

EGÉSZ ÉRTÉKŰ PROGRAMOZÁS

Adott: Egy n -változós, egész együtthatós lineáris egyenlőtlenség-rendszer.

Kérdés: Létezik-e **egész értékű** megoldás?

A HALMAZLEFEDÉS felfogható az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS speciális eseteként:

$$Ax \geq 1, \quad \sum_{i=1}^n x_i \leq B, \quad 0 \leq x_i \leq 1,$$

ahol A oszlopai a halmazrendszer elemeinek felelnek meg.

Azt is meg lehet mutatni, hogy az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS **NP**-ben van.

Tétel

Az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS probléma **NP**-teljes.

A RÉSZLETÖSSZEG probléma

A RÉSZLETÖSSZEG probléma

Adott: $a_1, \dots, a_n$ pozitív egészek és egy $K > 0$ célszám.

A RÉSZLETÖSSZEG probléma

Adott: $a_1, \dots, a_n$ pozitív egészek és egy $K > 0$ célszám.

Kérdés: Kiválasztható-e az a_i -k közül néhány elem úgy, hogy összegük K legyen?

NP-teljes problémák halmazokra és számokra

A RÉSZLETÖSSZEG probléma

Adott: $a_1, \dots, a_n$ pozitív egészek és egy $K > 0$ célszám.

Kérdés: Kiválasztható-e az a_i -k közül néhány elem úgy, hogy összegük K legyen?

Tétel

A RÉSZLETÖSSZEG probléma NP-teljes.

Az NP-beliség világos, az NP-nehézséget a 3SAT-ról való visszavezetéssel igazoljuk.

$3\text{SAT} \leq \text{RÉSZLETÖSSZEG}$

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.
- x_i -ből a t_i és f_i számok készülnek:

NP-teljes problémák halmazokra és számokra

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.
- x_i -ből a t_i és f_i számok készülnek:
 - t_i és f_i első m jegye csupa 0, kivéve az i . jegyet, ami 1-es;

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.
- x_i -ből a t_i és f_i számok készülnek:
 - t_i és f_i első m jegye csupa 0, kivéve az i . jegyet, ami 1-es;
 - t_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel x_i , egyébként 0;

NP-teljes problémák halmazokra és számokra

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.
- x_i -ből a t_i és f_i számok készülnek:
 - t_i és f_i első m jegye csupa 0, kivéve az i . jegyet, ami 1-es;
 - t_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel x_i , egyébként 0;
 - f_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel $\neg x_i$, egyébként 0.

NP-teljes problémák halmazokra és számokra

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.
- x_i -ből a t_i és f_i számok készülnek:
 - t_i és f_i első m jegye csupa 0, kivéve az i . jegyet, ami 1-es;
 - t_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel x_i , egyébként 0;
 - f_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel $\neg x_i$, egyébként 0.
- Továbbá, $a_i = b_i = 10^i$ minden $i = 1, \dots, n$ esetén.

NP-teljes problémák halmazokra és számokra

3SAT $\leq$ RÉSZLETÖSSZEG

- Legyen $\varphi = c_1 \wedge \dots \wedge c_n$ egy 3CNF, $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$.
- A φ -beli változók legyenek $x_1, \dots, x_m$.
- Ebből elkészítjük a RÉSZLETÖSSZEG probléma egy példányát, melyben $n + m$ -jegyű (!) számok szerepelnek.
- x_i -ből a t_i és f_i számok készülnek:
 - t_i és f_i első m jegye csupa 0, kivéve az i . jegyet, ami 1-es;
 - t_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel x_i , egyébként 0;
 - f_i utolsó n jegye közül az $m + k$. akkor 1-es, ha c_k -ban szerepel $\neg x_i$, egyébként 0.
- Továbbá, $a_i = b_i = 10^i$ minden $i = 1, \dots, n$ esetén.
- Az eredmény: a $\{f_i, t_i : 1 \leq i \leq m\} \cup \{a_i, b_i : 1 \leq i \leq n\}$ halmaz és a $K = 11 \dots 133 \dots 3$ célszám (m darab 1-es, majd n darab 3-as).

Példa

Ha $\varphi = (x_1 \vee \neg x_2 \vee x_3) \vee (x_1 \vee \neg x_2 \vee x_4) \vee (\neg x_1 \vee x_2 \vee \neg x_4)$:

$$t_1 = 1000110$$

$$f_1 = 1000001$$

$$t_2 = 0100001$$

$$f_2 = 0100110$$

$$t_3 = 0010100$$

$$f_3 = 0010000$$

$$t_4 = 0001010$$

$$f_4 = 0001001$$

$$a_1 = b_1 = 0000100$$

$$a_2 = b_2 = 0000010$$

$$a_3 = b_3 = 0000001$$

$$K = 1111333$$

Példa

Ha $\varphi = (x_1 \vee \neg x_2 \vee x_3) \vee (x_1 \vee \neg x_2 \vee x_4) \vee (\neg x_1 \vee x_2 \vee \neg x_4)$:

$$t_1 = 1000110$$

$$f_1 = 1000001$$

$$t_2 = 0100001$$

$$f_2 = 0100110$$

$$t_3 = 0010100$$

$$f_3 = 0010000$$

$$t_4 = 0001010$$

$$f_4 = 0001001$$

$$a_1 = b_1 = 0000100$$

$$a_2 = b_2 = 0000010$$

$$a_3 = b_3 = 0000001$$

$$K = 1111333$$

- t_1 és f_1 közül pontosan az egyiket kell válasszuk (K első jegye miatt);

Példa

Ha $\varphi = (x_1 \vee \neg x_2 \vee x_3) \vee (x_1 \vee \neg x_2 \vee x_4) \vee (\neg x_1 \vee x_2 \vee \neg x_4)$:

$$t_1 = 1000110$$

$$f_1 = 1000001$$

$$t_2 = 0100001$$

$$f_2 = 0100110$$

$$t_3 = 0010100$$

$$f_3 = 0010000$$

$$t_4 = 0001010$$

$$f_4 = 0001001$$

$$a_1 = b_1 = 0000100$$

$$a_2 = b_2 = 0000010$$

$$a_3 = b_3 = 0000001$$

$$K = 1111333$$

- t_1 és f_1 közül pontosan az egyiket kell válasszuk (K első jegye miatt);
- általában t_i és f_i közül is pontosan az egyiket – t_i választása feleljen meg az $x_i = 1$, f_i választása az $x_i = 0$ értékadásnak;

Példa

Ha $\varphi = (x_1 \vee \neg x_2 \vee x_3) \vee (x_1 \vee \neg x_2 \vee x_4) \vee (\neg x_1 \vee x_2 \vee \neg x_4)$:

$$t_1 = 1000110$$

$$f_1 = 1000001$$

$$t_2 = 0100001$$

$$f_2 = 0100110$$

$$t_3 = 0010100$$

$$f_3 = 0010000$$

$$t_4 = 0001010$$

$$f_4 = 0001001$$

$$a_1 = b_1 = 0000100$$

$$a_2 = b_2 = 0000010$$

$$a_3 = b_3 = 0000001$$

$$K = 1111333$$

- t_1 és f_1 közül pontosan az egyiket kell válasszuk (K első jegye miatt);
- általában t_i és f_i közül is pontosan az egyiket – t_i választása feleljen meg az $x_i = 1$, f_i választása az $x_i = 0$ értékadásnak;
- ezzel az $m + j$. jegyek mindegyike 0, 1, 2 vagy 3 lesz $j = 1, \dots, n$ -re, annak függvényében, hogy c_j -ben 0, 1, 2 vagy 3 literál vált igazgá;

Példa

Ha $\varphi = (x_1 \vee \neg x_2 \vee x_3) \vee (x_1 \vee \neg x_2 \vee x_4) \vee (\neg x_1 \vee x_2 \vee \neg x_4)$:

$$t_1 = 1000110$$

$$f_1 = 1000001$$

$$t_2 = 0100001$$

$$f_2 = 0100110$$

$$t_3 = 0010100$$

$$f_3 = 0010000$$

$$t_4 = 0001010$$

$$f_4 = 0001001$$

$$a_1 = b_1 = 0000100$$

$$a_2 = b_2 = 0000010$$

$$a_3 = b_3 = 0000001$$

$$K = 1111333$$

- t_1 és f_1 közül pontosan az egyiket kell válasszuk (K első jegye miatt);
- általában t_i és f_i közül is pontosan az egyiket – t_i választása feleljen meg az $x_i = 1$, f_i választása az $x_i = 0$ értékadásnak;
- ezzel az $m + j$. jegyek mindegyike 0, 1, 2 vagy 3 lesz $j = 1, \dots, n$ -re, annak függvényében, hogy c_j -ben 0, 1, 2 vagy 3 literál vált igazgá;
- ha az $m + j$. jegy 3, akkor jó, ha 2, akkor válasszuk ki még mondjuk a_j -t, ha 1, akkor a_j -t és b_j -t is, ha 0, akkor nem tudjuk ezen a jegyen elérni a célszámot

NP-teljes problémák halmazokra és számokra

Az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS egy másik speciális esete:

NP-teljes problémák halmazokra és számokra

Az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS egy másik speciális esete:

HÁTI ZSÁK

NP-teljes problémák halmazokra és számokra

Az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS egy másik speciális esete:

HÁTIZSÁK

Adott: n elem mindegyikének w_i súlya és c_i értéke, valamint W és C .

NP-teljes problémák halmazokra és számokra

Az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS egy másik speciális esete:

HÁTIZSÁK

Adott: n elem mindegyikének w_i súlya és c_i értéke, valamint W és C .

Kérdés: Kiválasztható-e ismétlés nélkül néhány elem úgy, hogy összértékük $\geq C$ és összsúlyuk $\leq W$?

NP-teljes problémák halmazokra és számokra

Az EGÉSZ ÉRTÉKŰ PROGRAMOZÁS egy másik speciális esete:

HÁTIZSÁK

Adott: n elem mindegyikének w_i súlya és c_i értéke, valamint W és C .

Kérdés: Kiválasztható-e ismétlés nélkül néhány elem úgy, hogy összértékük $\geq C$ és összsúlyuk $\leq W$?

Tétel

A HÁTIZSÁK probléma **NP**-teljes.

Ötlet

A RÉSZLETÖSSZEG problémában az a_i értékekből rendre készítsünk egy tárgyat $w_i = c_i = a_i$ súllyal és értékkel, továbbá legyen $C = W = K$, a célszám.

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

$$T[i, w] = \begin{cases} 0 & \text{ha } i = 0 \\ T[i - 1, w] & i > 0, w < w_i \\ \max\{T[i - 1, w], c_i + T[i - 1, w - w_i]\} & i > 0, w \geq w_i \end{cases}$$

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

$$T[i, w] = \begin{cases} 0 & \text{ha } i = 0 \\ T[i - 1, w] & i > 0, w < w_i \\ \max\{T[i - 1, w], c_i + T[i - 1, w - w_i]\} & i > 0, w \geq w_i \end{cases}$$

Ez ad egy $\mathcal{O}(n \cdot W)$ időigényű algoritmust

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

$$T[i, w] = \begin{cases} 0 & \text{ha } i = 0 \\ T[i - 1, w] & i > 0, w < w_i \\ \max\{T[i - 1, w], c_i + T[i - 1, w - w_i]\} & i > 0, w \geq w_i \end{cases}$$

Ez ad egy $\mathcal{O}(n \cdot W)$ időigényű algoritmust, ami **nem** polinom

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

$$T[i, w] = \begin{cases} 0 & \text{ha } i = 0 \\ T[i - 1, w] & i > 0, w < w_i \\ \max\{T[i - 1, w], c_i + T[i - 1, w - w_i]\} & i > 0, w \geq w_i \end{cases}$$

Ez ad egy $\mathcal{O}(n \cdot W)$ időigényű algoritmust, ami **nem** polinom, mert az input mérete $n(\log w_i + \log c_i) + \log W$.

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

$$T[i, w] = \begin{cases} 0 & \text{ha } i = 0 \\ T[i - 1, w] & i > 0, w < w_i \\ \max\{T[i - 1, w], c_i + T[i - 1, w - w_i]\} & i > 0, w \geq w_i \end{cases}$$

Ez ad egy $\mathcal{O}(n \cdot W)$ időigényű algoritmust, ami **nem** polinom, mert az input mérete $n(\log w_i + \log c_i) + \log W$.

Felfoghatjuk úgy is, hogy az algoritmusunk akkor polinomidejű, ha az inputban a súlyokat unárisan reprezentáljuk, vagy ha a súlyok a tárgyak számának egy polinomjával korlátozhatóak.

Algoritmus a HÁTIZSÁK eldöntésére

A következő rekurzív összefüggéssel számítható $T[i, w]$, ami a legfeljebb az első i tárgy felhasználásával egy w kapacitású hátizsákkal elérhető maximális haszon:

$$T[i, w] = \begin{cases} 0 & \text{ha } i = 0 \\ T[i - 1, w] & i > 0, w < w_i \\ \max\{T[i - 1, w], c_i + T[i - 1, w - w_i]\} & i > 0, w \geq w_i \end{cases}$$

Ez ad egy $\mathcal{O}(n \cdot W)$ időigényű algoritmust, ami **nem** polinom, mert az input mérete $n(\log w_i + \log c_i) + \log W$.

Felfoghatjuk úgy is, hogy az algoritmusunk akkor polinomidejű, ha az inputban a súlyokat unárisan reprezentáljuk, vagy ha a súlyok a tárgyak számának egy polinomjával korlátozhatóak.

Ez vezet el az algoritmikus bonyolultságelméletben a **pszeudopolinomialitás** fogalmához.

Pszeudopolinomiális algoritmusok, erős/gyenge NP-teljesség

Pszeudopolinomiális algoritmusok, erős/gyenge NP-teljesség

Pszeudopolinomiális algoritmusok

Legyen A egy probléma.

Pszeudopolinomiális algoritmusok, erős/gyenge NP-teljesség

Pszeudopolinomiális algoritmusok

Legyen A egy probléma. Egy A -t eldöntő algoritmus **pszeudopolinomiális**, ha tetszőleges $(a_1, \dots, a_m)$ inputon a futásideje $\mathcal{O}\left(\left(\sum_{1 \leq i \leq m} a_i\right)^k\right)$, valamely konstans k -ra.

Pszeudopolinomiális algoritmusok, erős/gyenge NP-teljesség

Pszeudopolinomiális algoritmusok

Legyen A egy probléma. Egy A -t eldöntő algoritmus **pszeudopolinomiális**, ha tetszőleges $(a_1, \dots, a_m)$ inputon a futásideje $\mathcal{O}\left(\left(\sum_{1 \leq i \leq m} a_i\right)^k\right)$, valamely konstans k -ra.

(Emlékezzünk vissza: az input **mérete** $\sum_{1 \leq i \leq m} \log a_i$ volt!

Pszeudopolinomiális algoritmusok, erős/gyenge NP-teljesség

Pszeudopolinomiális algoritmusok

Legyen A egy probléma. Egy A -t eldöntő algoritmus **pszeudopolinomiális**, ha tetszőleges $(a_1, \dots, a_m)$ inputon a futásideje $\mathcal{O}\left(\left(\sum_{1 \leq i \leq m} a_i\right)^k\right)$, valamely konstans k -ra.

(Emlékezzünk vissza: az input **mérete** $\sum_{1 \leq i \leq m} \log a_i$ volt!

Ha egy NP-teljes probléma eldönthető pszeudopolinomiális algoritmussal, úgy **gyengén NP-teljesnek**, ha pedig unáris változata is NP-teljes, akkor **erősen NP-teljesnek** nevezzük.

Pszeudopolinomiális algoritmusok, erős/gyenge NP-teljesség

Pszeudopolinomiális algoritmusok

Legyen A egy probléma. Egy A -t eldöntő algoritmus **pszeudopolinomiális**, ha tetszőleges $(a_1, \dots, a_m)$ inputon a futásideje $\mathcal{O}\left(\left(\sum_{1 \leq i \leq m} a_i\right)^k\right)$, valamely konstans k -ra.

(Emlékezzünk vissza: az input **mérete** $\sum_{1 \leq i \leq m} \log a_i$ volt!

Ha egy NP-teljes probléma eldönthető pszeudopolinomiális algoritmussal, úgy **gyengén NP-teljesnek**, ha pedig unáris változata is NP-teljes, akkor **erősen NP-teljesnek** nevezzük.

Átfogalmazás

Ha egy erősen NP-teljes problémára van pszeudopolinomiális algoritmus, akkor $P = NP$.

Pszeudopolinomiális algoritmusok, erős **NP**-teljesség

Pszeudopolinomiális algoritmusok, erős **NP**-teljesség

Példa

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.
- A TSP probléma viszont erősen **NP**-teljes.

Pszeudopolinomiális algoritmusok, erős **NP**-teljesség

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.
- A TSP probléma viszont erősen **NP**-teljes.
- Pozitív egészek faktorizálására van pszeudopolinomiális algoritmus, de nem ismert rá polinomidejű.

Pszudopolinomiális algoritmusok, erős **NP**-teljesség

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.
- A TSP probléma viszont erősen **NP**-teljes.
- Pozitív egészek faktorizálására van pszudopolinomiális algoritmus, de nem ismert rá polinomidejű.

A gyengén **NP**-teljes problémák mindazon példányai gyakorlatilag megoldhatónak tekinthetők, melyekben az inputon érkező számok **értéke** korlátozható az input **méretének** egy **polinomfüggvényével**.

Pszeudopolinomiális algoritmusok, erős **NP**-teljesség

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.
- A TSP probléma viszont erősen **NP**-teljes.
- Pozitív egészek faktorizálására van pszeudopolinomiális algoritmus, de nem ismert rá polinomidejű.

A gyengén **NP**-teljes problémák mindazon példányai gyakorlatilag megoldhatónak tekinthetők, melyekben az inputon érkező számok **értéke** korlátozható az input **méretének** egy **polinomfüggvényével**.

(Pl. a HÁTIZSÁK vagy a PARTÍCIÓ sok „kisméretű” tárgy esetén.)

Pszeudopolinomiális algoritmusok, erős **NP**-teljesség

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.
- A TSP probléma viszont erősen **NP**-teljes.
- Pozitív egészek faktorizálására van pszeudopolinomiális algoritmus, de nem ismert rá polinomidejű.

A gyengén **NP**-teljes problémák mindazon példányai gyakorlatilag megoldhatónak tekinthetők, melyekben az inputon érkező számok **értéke** korlátozható az input **méretének** egy **polinomfüggvényével**.

(Pl. a HÁTIZSÁK vagy a PARTÍCIÓ sok „kisméretű” tárgy esetén.)

PARTÍCIÓ

Pszeudopolinomiális algoritmusok, erős **NP**-teljesség

Példa

- A HÁTIZSÁK probléma gyengén **NP**-teljes.
- A TSP probléma viszont erősen **NP**-teljes.
- Pozitív egészek faktorizálására van pszeudopolinomiális algoritmus, de nem ismert rá polinomidejű.

A gyengén **NP**-teljes problémák mindazon példányai gyakorlatilag megoldhatónak tekinthetők, melyekben az inputon érkező számok **értéke** korlátozható az input **méretének** egy **polinomfüggvényével**.

(Pl. a HÁTIZSÁK vagy a PARTÍCIÓ sok „kisméretű” tárgy esetén.)

PARTÍCIÓ

- **Input:** $a_1, \dots, a_n$ pozitív egészek.

Pszeudopolinomiális algoritmusok, erős NP-teljeség

Példa

- A HÁTIZSÁK probléma gyengén NP-teljes.
- A TSP probléma viszont erősen NP-teljes.
- Pozitív egészek faktorizálására van pszeudopolinomiális algoritmus, de nem ismert rá polinomidejű.

A gyengén NP-teljes problémák mindazon példányai gyakorlatilag megoldhatónak tekinthetők, melyekben az inputon érkező számok **értéke** korlátozható az input **méretének** egy **polinomfüggvényével**.

(Pl. a HÁTIZSÁK vagy a PARTÍCIÓ sok „kisméretű” tárgy esetén.)

PARTÍCIÓ

- **Input:** $a_1, \dots, a_n$ pozitív egészek.
- **Kérdés:** Van-e $I \subseteq \{1, \dots, n\}$, melyre $\sum_{i \in I} a_i = \frac{\sum_{1 \leq i \leq n} a_i}{2}$?

A probléma (gyengén) NP-teljes.

A HÁTIZSÁK probléma egészértékű programozási feladatként felírva:

$$\sum_{i=1}^n w_i x_i \leq W$$

$$\sum_{i=1}^n c_i x_i \geq C$$

$$0 \leq x_i \leq 1$$

A HÁTIZSÁK probléma egészértékű programozási feladatként felírva:

$$\sum_{i=1}^n w_i x_i \leq W$$

$$\sum_{i=1}^n c_i x_i \geq C$$

$$0 \leq x_i \leq 1$$

Egy ILP feladat **LP-relaxációját** kapjuk, ha az egészek helyett a valósak körében oldjuk meg.

A HÁTIZSÁK probléma egészértékű programozási feladatként felírva:

$$\sum_{i=1}^n w_i x_i \leq W$$

$$\sum_{i=1}^n c_i x_i \geq C$$

$$0 \leq x_i \leq 1$$

Egy ILP feladat **LP-relaxációját** kapjuk, ha az egészek helyett a valósak körében oldjuk meg. (Azaz elhagyjuk az „ x_i egész” feltételt.)

A HÁTIZSÁK probléma egészértékű programozási feladatként felírva:

$$\sum_{i=1}^n w_i x_i \leq W$$

$$\sum_{i=1}^n c_i x_i \geq C$$

$$0 \leq x_i \leq 1$$

Egy ILP feladat **LP-relaxációját** kapjuk, ha az egészek helyett a valósak körében oldjuk meg. (Azaz elhagyjuk az „ x_i egész” feltételt.)

Mivel $LP \in \mathbf{P}$, a relaxált feladat hatékonyan megoldható.

TÖREDÉKES HÁTIZSÁK

TÖREDÉKES HÁTIZSÁK

- Mint a HÁTIZSÁK, de a tárgyak törhetőek: az i -edik tárgy x_i -ed része, $0 \leq x_i \leq 1$ egy $c_i x_i$ értékű, $w_i x_i$ súlyú tárgy.

TÖREDÉKES HÁTIZSÁK

- Mint a HÁTIZSÁK, de a tárgyak törhetőek: az i -edik tárgy x_i -ed része, $0 \leq x_i \leq 1$ egy $c_i x_i$ értékű, $w_i x_i$ súlyú tárgy.
- Erre a feladatba a hatékony **mohó** algoritmus optimális.

TÖREDÉKES HÁTIZSÁK

- Mint a HÁTIZSÁK, de a tárgyak törhetőek: az i -edik tárgy x_i -ed része, $0 \leq x_i \leq 1$ egy $c_i x_i$ értékű, $w_i x_i$ súlyú tárgy.
- Erre a feladatba a hatékony **mohó** algoritmus optimális.
- Kérdés, hogy mennyire jól „közelíti” az eredeti (függvény)problémát?

A következőkben olyan **optimalizálási** problémákra próbálunk adni hatékony **közelítő** algoritmusokat, melyeknek eldöntési változata **NP**-nehéz.

Approximáció

A következőkben olyan **optimalizálási** problémákra próbálunk adni hatékony **közelítő** algoritmusokat, melyeknek eldöntési változata **NP**-nehéz.

Az f függvény minimalizálási/maximalizálási probléma, ha $f(I) = \arg \min_{s \in S(I)} c(s) / \arg \max_{s \in S(I)} c(s)$ alakú, ahol $S(I)$ az I inputhoz tartozó **lehetséges megoldások** (nemüres) halmaza, c pedig minden megoldáshoz egy valós költséget/értéket rendelő függvény.

Ha f egy optimalizálási probléma, $\alpha > 0$ egy konstans, A pedig egy olyan polinomidejű algoritmus, melyre $A(I) \in S(I)$ minden I inputra úgy, hogy

$$\forall I : \max \left\{ \frac{c(A(I))}{c(f(I))}, \frac{c(f(I))}{c(A(I))} \right\} \leq \alpha,$$

akkor A egy **α -approximáló algoritmus f -re**.

Emlékeztető

Input: $G = (V, E)$ gráf, $K > 0$ egész.

Output: van-e olyan, legfeljebb K méretű X csúcshalmaz G -ben, melyre igaz, hogy G minden élének legalább az egyik végpontja X -beli?

Emlékeztető

Input: $G = (V, E)$ gráf, $K > 0$ egész.

Output: van-e olyan, legfeljebb K méretű X csúcshalmaz G -ben, melyre igaz, hogy G minden élének legalább az egyik végpontja X -beli?

Optimalizálási változat

Input: $G = (V, E)$ gráf.

Output: mekkora a **legkisebb** lefogó csúcshalmaz G -ben? (megoldás: egy lefogó csúcshalmaz; költsége: a mérete)

Emlékeztető

Input: $G = (V, E)$ gráf, $K > 0$ egész.

Output: van-e olyan, legfeljebb K méretű X csúcshalmaz G -ben, melyre igaz, hogy G minden élének legalább az egyik végpontja X -beli?

Optimalizálási változat

Input: $G = (V, E)$ gráf.

Output: mekkora a **legkisebb** lefogó csúcshalmaz G -ben? (megoldás: egy lefogó csúcshalmaz; költsége: a mérete)

Természetesen ha az optimalizálási változatra lenne egy hatékony módszer, akkor az eldöntésre is.

Emlékeztető

Input: $G = (V, E)$ gráf, $K > 0$ egész.

Output: van-e olyan, legfeljebb K méretű X csúcshalmaz G -ben, melyre igaz, hogy G minden élének legalább az egyik végpontja X -beli?

Optimalizálási változat

Input: $G = (V, E)$ gráf.

Output: mekkora a **legkisebb** lefogó csúcshalmaz G -ben? (megoldás: egy lefogó csúcshalmaz; költsége: a mérete)

Természetesen ha az optimalizálási változatra lenne egy hatékony módszer, akkor az eldöntésre is. Mivel az eldöntési változat **NP**-teljes, így az optimalizálási változatra sem ismert hatékony algoritmus.

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

- Legyen $X = \emptyset$.

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

- Legyen $X = \emptyset$.
- Amíg van él G -ben:

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

- Legyen $X = \emptyset$.
- Amíg van él G -ben:
 - Válasszunk egy **tetszőleges** e élt.

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

- Legyen $X = \emptyset$.
- Amíg van él G -ben:
 - Válasszunk egy **tetszőleges** e élt.
 - Vegyük be X -be e **mindkét** végpontját.

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

- Legyen $X = \emptyset$.
- Amíg van él G -ben:
 - Válasszunk egy **tetszőleges** e élt.
 - Vegyük be X -be e **mindkét** végpontját.
 - Vegyük el G -ből az összes, e bármelyik végpontjára illeszkedő élt.

Tekintsük a CSÚCSLEFEDÉS-re a következő egyszerű algoritmust:

- Legyen $X = \emptyset$.
- Amíg van él G -ben:
 - Válasszunk egy **tetszőleges** e élt.
 - Vegyük be X -be e **mindkét** végpontját.
 - Vegyük el G -ből az összes, e bármelyik végpontjára illeszkedő élt.
- Adjuk vissza X -et.

Példa

TODO

Tétel

Az előző főlán szereplő algoritmus egy 2-approximáló algoritmus a CSÚCSLEFEDÉSre.

Tétel

Az előző fólián szereplő algoritmus egy 2-approximáló algoritmus a CSÚCSLEFEDÉSre.

Ötlet

Tétel

Az előző fólián szereplő algoritmus egy 2-approximáló algoritmus a CSÚCSLEFEDÉSre.

Ötlet

- Az algoritmus nyilván egy lefogó csúcshalmazt ad vissza.

Tétel

Az előző fólíán szereplő algoritmus egy 2-approximáló algoritmus a CSÚCSLEFEDÉSre.

Ötlet

- Az algoritmus nyilván egy lefogó csúcshalmazt ad vissza.
- Ha az X halmazba rendre az $e_1, e_2, \dots, e_k$ élek végpontjai kerültek bele, akkor $e_1, \dots, e_k$ egy párosítás G -ben, mindegyiküknek legalább az egyik végpontját le kell fogni, vagyis a minimum legalább k .

Tétel

Az előző fólián szereplő algoritmus egy 2-approximáló algoritmus a CSÚCSLEFEDÉSre.

Ötlet

- Az algoritmus nyilván egy lefogó csúcshalmazt ad vissza.
- Ha az X halmazba rendre az $e_1, e_2, \dots, e_k$ élek végpontjai kerültek bele, akkor $e_1, \dots, e_k$ egy párosítás G -ben, mindegyiküknek legalább az egyik végpontját le kell fogni, vagyis a minimum legalább k .
- Az algoritmus által visszaadott eredmény pedig $2k$.

Tétel

Az előző fólíán szereplő algoritmus egy 2-approximáló algoritmus a CSÚCSLEFEDÉSre.

Ötlet

- Az algoritmus nyilván egy lefogó csúcshalmazt ad vissza.
- Ha az X halmazba rendre az $e_1, e_2, \dots, e_k$ élek végpontjai kerültek bele, akkor $e_1, \dots, e_k$ egy párosítás G -ben, mindegyiküknek legalább az egyik végpontját le kell fogni, vagyis a minimum legalább k .
- Az algoritmus által visszaadott eredmény pedig $2k$.

A CSÚCSLEFEDÉSre a mai napig **nem ismert ennél jobb** approximációs algoritmus.

Tétel

A TSP probléma nem közelíthető: **nincs** α -approximáló algoritmus, csak ha $P = NP$.

Tétel

A TSP probléma nem közelíthető: **nincs** α -approximáló algoritmus, csak ha $P = NP$.

Ötlet

Tegyük fel, hogy van egy α -approximáló A algoritmus TSP-re.

Tétel

A TSP probléma nem közelíthető: **nincs** α -approximáló algoritmus, csak ha $P = NP$.

Ötlet

Tegyük fel, hogy van egy α -approximáló A algoritmus TSP-re.

Akkor A -t felhasználva meg tudjuk oldani a Hamilton-kört a következőképpen: a $G = (V, E)$ gráfból készítsük $V \times V$ -n az alábbi $D(G)$ távolságmátrixot:

$$d_{u,v} = \begin{cases} 1 & \text{ha } (u, v) \in E; \\ |V| \cdot \alpha + 1 & \text{egyébként.} \end{cases}$$

Ekkor ha van Hamilton-kör, az optimum $|V|$; ha nincs, az optimum legalább $|V| \cdot \alpha + 1$.

Ötlet befejezése

Ekkor a következő algoritmus:

Ha $A(D(G)) \leq |V| \cdot \alpha + 1$, fogadjuk el G -t, egyébként utasítsuk el
eldönti a Hamilton-kör problémát, polinomidőben.

Ötlet befejezése

Ekkor a következő algoritmus:

Ha $A(D(G)) \leq |V| \cdot \alpha + 1$, fogadjuk el G -t, egyébként utasítsuk el
eldönti a Hamilton-kör problémát, polinomidőben.

Így várhatóan (hacsaknem $\mathbf{P} = \mathbf{NP}$) nincs approximáló algoritmus sem TSP-re.

Ötlet befejezése

Ekkor a következő algoritmus:

Ha $A(D(G)) \leq |V| \cdot \alpha + 1$, fogadjuk el G -t, egyébként utasítsuk el
eldönti a Hamilton-kör problémát, polinomidőben.

Így várhatóan (hacsaknem $\mathbf{P} = \mathbf{NP}$) nincs approximáló algoritmus sem TSP-re.
Kereshetünk viszont olyan speciális esetet, amire van.

A probléma

Input: egy D távolságmátrix,

A probléma

Input: egy D távolságmátrix, **ami teljesíti a háromszögeyenlőtlenséget:**

$$\forall i, j, k : D_{i,j} + D_{j,k} \geq D_{i,k}.$$

A probléma

Input: egy D távolságmátrix, **ami teljesíti a háromszögeyenlőtlenséget**:

$$\forall i, j, k : D_{i,j} + D_{j,k} \geq D_{i,k}.$$

Output: a minimális összsúlyú körút súlya.

A probléma

Input: egy D távolságmátrix, **ami teljesíti a háromszögeyenlőtlenséget**:

$$\forall i, j, k : D_{i,j} + D_{j,k} \geq D_{i,k}.$$

Output: a minimális összsúlyú körút súlya.

Bonyolultság

Az eldöntési probléma továbbra is **NP**-nehéz.

A probléma

Input: egy D távolságmátrix, **ami teljesíti a háromszögeyenlőtlenséget**:

$$\forall i, j, k : D_{i,j} + D_{j,k} \geq D_{i,k}.$$

Output: a minimális összsúlyú körút súlya.

Bonyolultság

Az eldöntési probléma továbbra is **NP**-nehéz.

(Ilyen mátrixot állítottunk elő a Hamilton-kör $\text{TSP}(E)$ -re való visszavezetésekor.)

METRIKUS TSP: 2-approximáló algoritmus

Algoritmus

METRIKUS TSP: 2-approximáló algoritmus

Algoritmus

- Állítsuk elő (Prim vagy Kruskal algoritmusával, például) D egy **minimális feszítőfáját**.

METRIKUS TSP: 2-approximáló algoritmus

Algoritmus

- Állítsuk elő (Prim vagy Kruskal algoritmusával, például) D egy **minimális feszítőfáját**.
- Kettőzzük meg a fa éleit.

Algoritmus

- Állítsuk elő (Prim vagy Kruskal algoritmusával, például) D egy **minimális feszítőfáját**.
- Kettőzzük meg a fa éleit.
- A kapott multigráfnak vegyük egy Euler-körvonalát.

Algoritmus

- Állítsuk elő (Prim vagy Kruskal algoritmusával, például) D egy **minimális feszítőfáját**.
- Kettőzzük meg a fa éleit.
- A kapott multigráfnak vegyük egy Euler-körvonalát.
- A körvonalból hagyjuk el a korábban már meglátogatott csúcsokat.

Algoritmus

- Állítsuk elő (Prim vagy Kruskal algoritmusával, például) D egy **minimális feszítőfáját**.
- Kettőzzük meg a fa éleit.
- A kapott multigráfnak vegyük egy Euler-körvonalát.
- A körvonalból hagyjuk el a korábban már meglátogatott csúcsokat.
- Adjuk vissza az így kapott kör összsúlyát.

METRIKUS TSP: 2-approximáló algoritmus

Algoritmus

- Állítsuk elő (Prim vagy Kruskal algoritmusával, például) D egy **minimális feszítőfáját**.
- Kettőzzük meg a fa éleit.
- A kapott multigráfnak vegyük egy Euler-körvonalát.
- A körvonalból hagyjuk el a korábban már meglátogatott csúcsokat.
- Adjuk vissza az így kapott kör összsúlyát.

Közelítés

A fenti egy 2-approximáló algoritmus a METRIKUS TSP-re.

METRIKUS TSP: 2-approximáló algoritmus

Példa

TODO

METRIKUS TSP: 2-approximáló algoritmus

2-approximálás

Az algoritmus 2-approximáló, ez a következőkből áll össze:

2-approximálás

Az algoritmus 2-approximáló, ez a következőkből áll össze:

- egy tényleges körút költségét adja vissza;

2-approximálás

Az algoritmus 2-approximáló, ez a következőkből áll össze:

- egy tényleges körút költségét adja vissza;
- bármilyen körútból elhagyva egy élt egy feszítőfát kapunk $\Rightarrow$ a minimális feszítőfa összsúlya kisebb, mint a minimális körúté;

2-approximálás

Az algoritmus 2-approximáló, ez a következőkből áll össze:

- egy tényleges körút költségét adja vissza;
- bármilyen körútból elhagyva egy élt egy feszítőfát kapunk $\Rightarrow$ a minimális feszítőfa összsúlya kisebb, mint a minimális körúté;
- az élek kettőzésével kapott gráfban ill. annak az Euler-körvonalában az élek összsúlya tehát kisebb, mint **kétszer** a minimális körúté;

2-approximálás

Az algoritmus 2-approximáló, ez a következőkből áll össze:

- egy tényleges körút költségét adja vissza;
- bármilyen körútból elhagyva egy élt egy feszítőfát kapunk $\Rightarrow$ a minimális feszítőfa összsúlya kisebb, mint a minimális körúté;
- az élek kettőzésével kapott gráfban ill. annak az Euler-körvonalában az élek összsúlya tehát kisebb, mint **kétszer** a minimális körúté;
- a már látott pontok kihagyásával **a háromszög-egyenlőtlenség miatt** nem növekszik az összsúly.

METRIKUS TSP: 2-approximáló algoritmus

2-approximálás

Az algoritmus 2-approximáló, ez a következőkből áll össze:

- egy tényleges körút költségét adja vissza;
- bármilyen körútból elhagyva egy élt egy feszítőfát kapunk $\Rightarrow$ a minimális feszítőfa összsúlya kisebb, mint a minimális körúté;
- az élek kettőzésével kapott gráfban ill. annak az Euler-körvonalában az élek összsúlya tehát kisebb, mint **kétszer** a minimális körúté;
- a már látott pontok kihagyásával **a háromszög-egyenlőtlenség miatt** nem növekszik az összsúly.

Megjegyzés: $\frac{3}{2}$ -approximálás

Az élek kettőzése helyett egy „ügyesebb” módszerrel kaphatunk egy $\frac{3}{2}$ -approximáló algoritmust is.

MAX-2SAT

A probléma

Input: egy $2CNF$, klózonként pontosan két literállal; a literálok változói különböznek.

A probléma

Input: egy $2CNF$, klózonként pontosan két literállal; **a literálok változói különböznek.**

Output: egyszerre legfeljebb hány klóz elégíthető ki benne?

MAX-2SAT

A probléma

Input: egy $2CNF$, klózonként pontosan két literállal; **a literálok változói különböznek.**

Output: egyszerre legfeljebb hány klóz elégíthető ki benne?

Bonyolultság

Tudjuk, hogy a 2SAT probléma **P**-beli.

MAX-2SAT

A probléma

Input: egy $2CNF$, klózonként pontosan két literállal; **a literálok változói különböznek.**

Output: egyszerre legfeljebb hány klóz elégíthető ki benne?

Bonyolultság

Tudjuk, hogy a 2SAT probléma **P**-beli.

Viszont, ennek az optimalizálási problémának az eldöntési változata (input egy F $2CNF$ és egy K szám, kielégíthető-e F -ben egyszerre K klóz?) **NP**-teljes.

MAX-2SAT

A probléma

Input: egy $2CNF$, klózonként pontosan két literállal; **a literálok változói különböznek.**

Output: egyszerre legfeljebb hány klóz elégíthető ki benne?

Bonyolultság

Tudjuk, hogy a 2SAT probléma **P**-beli.

Viszont, ennek az optimalizálási problémának az eldöntési változata (input egy F $2CNF$ és egy K szám, kielégíthető-e F -ben egyszerre K klóz?) **NP**-teljes.

Először adunk egy **randomizált $\frac{4}{3}$ -közelítő algoritmust**, aztán ezt **derandomizálva** egy determinisztikusát.

Randomizált algoritmus

Random: várható értékben approximáljon

Egy A randomizált algoritmus α -approximáló, ha **várható értéke** legfeljebb α -szor rosszabb, mint az optimális, vagyis:

$$\forall x \max \left\{ \frac{E(A(x))}{f(x)}, \frac{f(x)}{E(A(x))} \right\} \leq \alpha.$$

Randomizált algoritmus

Random: várható értékben approximáljon

Egy A randomizált algoritmus α -approximáló, ha **várható értéke** legfeljebb α -szor rosszabb, mint az optimális, vagyis:

$$\forall x \max \left\{ \frac{E(A(x))}{f(x)}, \frac{f(x)}{E(A(x))} \right\} \leq \alpha.$$

MAX-2SAT

Ebben az esetben tehát egy olyan algoritmust kell adnunk, mely legalább $\frac{3}{4}X$ klózt kielégít az input formulában, ha legfeljebb X elégíthető ki egyszerre.

Randomizált algoritmus

Random: várható értékben approximáljon

Egy A randomizált algoritmus α -approximáló, ha **várható értéke** legfeljebb α -szor rosszabb, mint az optimális, vagyis:

$$\forall x \max \left\{ \frac{E(A(x))}{f(x)}, \frac{f(x)}{E(A(x))} \right\} \leq \alpha.$$

MAX-2SAT

Ebben az esetben tehát egy olyan algoritmust kell adnunk, mely legalább $\frac{3}{4}X$ klózt kielégít az input formulában, ha legfeljebb X elégíthető ki egyszerre.

Ennél többet teszünk: olyan algoritmust fogunk adni, mely (várható értékben) a klózok $\frac{3}{4}$ -ét (tehát nem csak az egyszerre kielégíthető klózokét!) igazzá teszi.

Randomizált algoritmus

Randomizált algoritmus

Állítsunk minden változót egymástól függetlenül $\frac{1}{2} - \frac{1}{2}$ valószínűséggel igazra vagy hamisra.

Randomizált algoritmus

Állítsunk minden változót egymástól függetlenül $\frac{1}{2} - \frac{1}{2}$ valószínűséggel igazra vagy hamisra.

Várható érték

Mivel egy klózban két különböző változó van, így annak esélye, hogy a klóz igaz lesz, $\frac{3}{4}$.

Randomizált algoritmus

Állítsunk minden változót egymástól függetlenül $\frac{1}{2} - \frac{1}{2}$ valószínűséggel igazra vagy hamisra.

Várható érték

Mivel egy klózban két különböző változó van, így annak esélye, hogy a klóz igaz lesz, $\frac{3}{4}$.

A várható érték additivitása miatt így várható értékben a klózek $\frac{3}{4}$ -ét kielégítjük.

Most az előző dián szereplő véletlen algoritmusban csökkentjük a generálandó véletlen bitek számát: **derandomizálunk**.

Derandomizálás

Most az előző dián szereplő véletlen algoritmusban csökkentjük a generálandó véletlen bitek számát: **derandomizálunk**.

Determinisztikus algoritmus

Legyenek $x_1, x_2, \dots, x_n$ az input formulában szereplő változók.

Most az előző dián szereplő véletlen algoritmusban csökkentjük a generálandó véletlen bitek számát: **derandomizálunk**.

Determinisztikus algoritmus

Legyenek $x_1, x_2, \dots, x_n$ az input formulában szereplő változók.

- Először értéket adunk x_1 -nek, majd x_2 -nek, $\dots$, végül x_n -nek, az i -edik menetben x_i -nek.

Most az előző dián szereplő véletlen algoritmusban csökkentjük a generálandó véletlen bitek számát: **derandomizálunk**.

Determinisztikus algoritmus

Legyenek $x_1, x_2, \dots, x_n$ az input formulában szereplő változók.

- Először értéket adunk x_1 -nek, majd x_2 -nek, $\dots$, végül x_n -nek, az i -edik menetben x_i -nek.
- Az i -edik menetben $x_1, \dots, x_{i-1}$ értéke már rögzítve van.

Most az előző dián szereplő véletlen algoritmusban csökkentjük a generálandó véletlen bitek számát: **derandomizálunk**.

Determinisztikus algoritmus

Legyenek $x_1, x_2, \dots, x_n$ az input formulában szereplő változók.

- Először értéket adunk x_1 -nek, majd x_2 -nek, $\dots$, végül x_n -nek, az i -edik menetben x_i -nek.
- Az i -edik menetben $x_1, \dots, x_{i-1}$ értéke már rögzítve van.
- Számítsuk ki $b = 0, 1$ -re, hogy mennyi klóz elégülne ki várható értékben, ha x_i értékét b -re választjuk, $x_{i+1}, \dots, x_n$ értékét pedig $\frac{1}{2} - \frac{1}{2}$ valószínűséggel, függetlenül választjuk 0-nak ill. 1-nek.

Most az előző dián szereplő véletlen algoritmusban csökkentjük a generálandó véletlen bitek számát: **derandomizálunk**.

Determinisztikus algoritmus

Legyenek $x_1, x_2, \dots, x_n$ az input formulában szereplő változók.

- Először értéket adunk x_1 -nek, majd x_2 -nek, $\dots$, végül x_n -nek, az i -edik menetben x_i -nek.
- Az i -edik menetben $x_1, \dots, x_{i-1}$ értéke már rögzítve van.
- Számítsuk ki $b = 0, 1$ -re, hogy mennyi klóz elégülne ki várható értékben, ha x_i értékét b -re választjuk, $x_{i+1}, \dots, x_n$ értékét pedig $\frac{1}{2} - \frac{1}{2}$ valószínűséggel, függetlenül választjuk 0-nak ill. 1-nek.
- Amelyik érték nagyobb lett, arra állítjuk ténylegesen x_i értékét, és folytatjuk a ciklust.

Vegyük észre, hogy a várható értéket determinisztikusan ki tudjuk számítani.

Vegyük észre, hogy a várható értéket determinisztikusan ki tudjuk számítani. Annak esélye, hogy egy klóz értéke igaz, a következőképp számítható:

Vegyük észre, hogy a várható értéket determinisztikusan ki tudjuk számítani. Annak esélye, hogy egy klóz értéke igaz, a következőképp számítható:

- ha a klózban van olyan literál, melynek értékét 1-re rögzítettük, akkor 1;

Vegyük észre, hogy a várható értéket determinisztikusan ki tudjuk számítani. Annak esélye, hogy egy klóz értéke igaz, a következőképp számítható:

- ha a klózban van olyan literál, melynek értékét 1-re rögzítettük, akkor 1;
- különben $1 - \frac{1}{2^k}$, ahol $0 \leq k \leq 2$ a klózban levő, még nem rögzített értékű változók száma.

Vegyük észre, hogy a várható értéket determinisztikusan ki tudjuk számítani. Annak esélye, hogy egy klóz értéke igaz, a következőképp számítható:

- ha a klózban van olyan literál, melynek értékét 1-re rögzítettük, akkor 1;
- különben $1 - \frac{1}{2^k}$, ahol $0 \leq k \leq 2$ a klózban levő, még nem rögzített értékű változók száma.

A formulában igazra állított klózok értéke ezek összege.

Azt is be lehet könnyen látni, hogy minden menetben a várható érték nem csökkenhet, így az utolsó menet végére a „várható érték” továbbra is legalább a klózok $\frac{3}{4}$ -e lesz.

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

MAX-2SAT példa

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Ha $x_1 = 0$, a várható értékek: $\frac{1}{2}$, $\frac{1}{2}$, 1, 1, 1, 1, a többi $\frac{3}{4}$, összesen 9.5;

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Ha $x_1 = 0$, a várható értékek: $\frac{1}{2}, \frac{1}{2}, 1, 1, 1, 1$, a többi $\frac{3}{4}$, összesen 9.5;
ha $x_1 = 1$, a várható értékek $1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}$, a többi $\frac{3}{4}$, összesen 8.5;

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Ha $x_1 = 0$, a várható értékek: $\frac{1}{2}, \frac{1}{2}, 1, 1, 1, 1$, a többi $\frac{3}{4}$, összesen 9.5;
ha $x_1 = 1$, a várható értékek $1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}$, a többi $\frac{3}{4}$, összesen 8.5;
így legyen $x_1 = 0$.

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Ha $x_1 = 0$, a várható értékek: $\frac{1}{2}, \frac{1}{2}, 1, 1, 1, 1$, a többi $\frac{3}{4}$, összesen 9.5;

ha $x_1 = 1$, a várható értékek $1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}$, a többi $\frac{3}{4}$, összesen 8.5;

így legyen $x_1 = 0$.

Ezek után ha $x_1 = 0$ és $x_2 = 0$, a várható értékek $0, 1, 1, 1, 1, 1, \frac{1}{2}, \frac{1}{2}, 1, 1, \frac{3}{4}$ és $\frac{3}{4}$, összesen 9.5;

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Ha $x_1 = 0$, a várható értékek: $\frac{1}{2}, \frac{1}{2}, 1, 1, 1, 1$, a többi $\frac{3}{4}$, összesen 9.5;

ha $x_1 = 1$, a várható értékek $1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}$, a többi $\frac{3}{4}$, összesen 8.5;

így legyen $x_1 = 0$.

Ezek után ha $x_1 = 0$ és $x_2 = 0$, a várható értékek $0, 1, 1, 1, 1, 1, \frac{1}{2}, \frac{1}{2}, 1, 1, \frac{3}{4}$ és $\frac{3}{4}$, összesen 9.5;

ha pedig $x_1 = 0$ és $x_2 = 1$, a várható értékek $1, 0, 1, 1, 1, 1, 1, 1, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}$ és $\frac{3}{4}$, szintén 9.5. Legyen mondjuk $x_2 = 1$.

Példa

$$\begin{aligned} F = & (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge \\ & (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge \\ & (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \end{aligned}$$

Determinisztikus

Ha $x_1 = 0$, a várható értékek: $\frac{1}{2}, \frac{1}{2}, 1, 1, 1, 1$, a többi $\frac{3}{4}$, összesen 9.5;

ha $x_1 = 1$, a várható értékek $1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}$, a többi $\frac{3}{4}$, összesen 8.5;

így legyen $x_1 = 0$.

Ezek után ha $x_1 = 0$ és $x_2 = 0$, a várható értékek $0, 1, 1, 1, 1, 1, \frac{1}{2}, \frac{1}{2}, 1, 1, \frac{3}{4}$ és $\frac{3}{4}$, összesen 9.5;

ha pedig $x_1 = 0$ és $x_2 = 1$, a várható értékek $1, 0, 1, 1, 1, 1, 1, 1, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}$ és $\frac{3}{4}$, szintén 9.5. Legyen mondjuk $x_2 = 1$.

...és így tovább...

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;

TÖREDÉKES HÁTIZSÁK

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat teljesen, amennyit csak tudunk;

TÖREDÉKES HÁTIZSÁK

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat teljesen, amennyit csak tudunk;
- az első tárgyat, ami nem fér be, pedig „törjük” úgy, hogy a töredék megtöltse a hátizsák maradék kapacitását.

TÖREDÉKES HÁTIZSÁK

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat teljesen, amennyit csak tudunk;
- az első tárgyat, ami nem fér be, pedig „törjük” úgy, hogy a töredék megtöltse a hátizsák maradék kapacitását.

Approximálás?

Adódik a következő ötlet a HÁTIZSÁK problémára:

TÖREDÉKES HÁTIZSÁK

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat teljesen, amennyit csak tudunk;
- az első tárgyat, ami nem fér be, pedig „törjük” úgy, hogy a töredék megtöltse a hátizsák maradék kapacitását.

Approximálás?

Adódik a következő ötlet a HÁTIZSÁK problémára:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;

TÖREDÉKES HÁTIZSÁK

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat teljesen, amennyit csak tudunk;
- az első tárgyat, ami nem fér be, pedig „törjük” úgy, hogy a töredék megtöltse a hátizsák maradék kapacitását.

Approximálás?

Adódik a következő ötlet a HÁTIZSÁK problémára:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat, amennyit csak tudunk.

TÖREDÉKES HÁTIZSÁK

Algoritmus a TÖREDÉKES HÁTIZSÁKra

Erre a változatra a **mohó** algoritmus optimális:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat teljesen, amennyit csak tudunk;
- az első tárgyat, ami nem fér be, pedig „törjük” úgy, hogy a töredék megtöltse a hátizsák maradék kapacitását.

Approximálás?

Adódik a következő ötlet a HÁTIZSÁK problémára:

- rendezzük csökkenőbe a tárgyakat $\frac{c_i}{w_i}$ szerint;
- eszerint a sorrend szerint vegyünk be annyi tárgyat, amennyit csak tudunk.

Ez az algoritmus **nem** α -approximálja a HÁTIZSÁK problémát, semmilyen α konstansra.

Példa

Ha pl. két tárgyunk van, $w_1 = 1$, $c_1 = 1$, $w_2 = W$ és $c_2 = W - 1$:

Példa

Ha pl. két tárgyunk van, $w_1 = 1$, $c_1 = 1$, $w_2 = W$ és $c_2 = W - 1$:

- az első tárgy fajlagosan értékesebb, mint a második

Példa

Ha pl. két tárgyunk van, $w_1 = 1$, $c_1 = 1$, $w_2 = W$ és $c_2 = W - 1$:

- az első tárgy fajlagosan értékesebb, mint a második
- a töredékes változat optimumhelye $(1, \frac{W-1}{W})$, értéke $1 + \frac{(W-1)^2}{W}$

Példa

Ha pl. két tárgyunk van, $w_1 = 1$, $c_1 = 1$, $w_2 = W$ és $c_2 = W - 1$:

- az első tárgy fajlagosan értékesebb, mint a második
- a töredékes változat optimumhelye $(1, \frac{W-1}{W})$, értéke $1 + \frac{(W-1)^2}{W}$
- a mohó algoritmus lefele kerekítő változata tehát az $(1, 0)$ helyen felvett 1 értéket adja vissza

Példa

Ha pl. két tárgyunk van, $w_1 = 1$, $c_1 = 1$, $w_2 = W$ és $c_2 = W - 1$:

- az első tárgy fajlagosan értékesebb, mint a második
- a töredékes változat optimumhelye $(1, \frac{W-1}{W})$, értéke $1 + \frac{(W-1)^2}{W}$
- a mohó algoritmus lefele kerekítő változata tehát az $(1, 0)$ helyen felvett 1 értéket adja vissza
- az optimumhely $(0, 1)$, értéke $W - 1$.

Ha tehát W elég nagy, $1 \cdot \alpha < W - 1$ lesz, így az algoritmus ezen változata nem α -approximáló.

HÁTI ZSÁK 2-approximálható

HÁTIZSÁK 2-approximálható

Kis módosítás

A következő algoritmus viszont 2-approximálja a HÁTIZSÁK problémát:

HÁTIZSÁK 2-approximálható

Kis módosítás

A következő algoritmus viszont 2-approximálja a HÁTIZSÁK problémát:

- Tegyük a tárgyakat fajlagos érték szerint csökkenő sorrendbe:

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}.$$

HÁTIZSÁK 2-approximálható

Kis módosítás

A következő algoritmus viszont 2-approximálja a HÁTIZSÁK problémát:

- Tegyük a tárgyakat fajlagos érték szerint csökkenő sorrendbe:

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}.$$

- Számítsuk ki a TÖREDÉKES HÁTIZSÁK optimumhelyét:
 $(1, 1, \dots, 1, x, 0, 0, \dots, 0)$, ahol x a k . tárgy (amelyiket el kellett törnünk, mert már nem fért be).

HÁTIZSÁK 2-approximálható

Kis módosítás

A következő algoritmus viszont 2-approximálja a HÁTIZSÁK problémát:

- Tegyük a tárgyakat fajlagos érték szerint csökkenő sorrendbe:
$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}.$$
- Számítsuk ki a TÖREDÉKES HÁTIZSÁK optimumhelyét:
 $(1, 1, \dots, 1, x, 0, 0, \dots, 0)$, ahol x a k . tárgy (amelyiket el kellett törnünk, mert már nem fért be).
- Adjuk vissza vagy $\sum_{i=1}^{k-1} c_i$ -t, vagy c_k -t, amelyik nagyobb.

HÁTIZSÁK 2-approximálható

Kis módosítás

A következő algoritmus viszont 2-approximálja a HÁTIZSÁK problémát:

- Tegyük a tárgyakat fajlagos érték szerint csökkenő sorrendbe:
$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}.$$
- Számítsuk ki a TÖREDÉKES HÁTIZSÁK optimumhelyét:
 $(1, 1, \dots, 1, x, 0, 0, \dots, 0)$, ahol x a k . tárgy (amelyiket el kellett törnünk, mert már nem fért be).
- Adjuk vissza vagy $\sum_{i=1}^{k-1} c_i$ -t, vagy c_k -t, amelyik nagyobb.

Tehát: vagy fajlagosan csökkenő sorrendben beteszünk, amit csak lehet, vagy az így éppen kimaradó tárgyat egyedül rakjuk be, amelyik jobb.

HÁTIZSÁK 2-approximálható

Amiért ez legalább az optimum felét adja:

HÁTIZSÁK 2-approximálható

Amiért ez legalább az optimum felét adja:

- a töredékes hátizsák optima ennek a két értéknek az összegénél kisebb, így a kettő közül a nagyobb legalább a töredékes optimumának fele;

HÁTIZSÁK 2-approximálható

Amiért ez legalább az optimum felét adja:

- a töredékes hátizsák optima ennek a két értéknek az összegénél kisebb, így a kettő közül a nagyobb legalább a töredékes optimumának fele;
- a töredékes hátizsák optima (mivel relaxált változat) legalább akkora, mint a 0/1 hátizsáké.

HÁTIZSÁK 2-approximálható

Amiért ez legalább az optimum felét adja:

- a töredékes hátizsák optima ennek a két értéknek az összegénél kisebb, így a kettő közül a nagyobb legalább a töredékes optimumának fele;
- a töredékes hátizsák optima (mivel relaxált változat) legalább akkora, mint a 0/1 hátizsáké.

Pár kimaradt részlet

Az algoritmus így akkor nem működik, ha minden tárgy befér egyszerre (ekkor az lesz az optimumhely), vagy ha a k . tárgy egyedül se férne be (de az ilyen tárgyakat eleve nem kell felveszünk a listába), de ezek könnyen javíthatók.

HÁTIZSÁK $1 + \epsilon$ -approximálható

Nézzünk egy másik kézenfekvő algoritmust a HÁTIZSÁK probléma közelítő megoldására:

HÁTIZSÁK $1 + \epsilon$ -approximálható

Nézzünk egy másik kézenfekvő algoritmust a HÁTIZSÁK probléma közelítő megoldására:

Skálázás ϵ -ra

HÁTIZSÁK $1 + \epsilon$ -approximálható

Nézzünk egy másik kézenfekvő algoritmust a HÁTIZSÁK probléma közelítő megoldására:

Skálázás ϵ -ra

- Legyen $w_1, \dots, w_n, c_1, \dots, c_n, W$ a HÁTIZSÁK probléma egy inputja és $\epsilon > 0$ egy valós szám.

HÁTIZSÁK $1 + \epsilon$ -approximálható

Nézzünk egy másik kézenfekvő algoritmust a HÁTIZSÁK probléma közelítő megoldására:

Skálázás ϵ -ra

- Legyen $w_1, \dots, w_n, c_1, \dots, c_n, W$ a HÁTIZSÁK probléma egy inputja és $\epsilon > 0$ egy valós szám.
- Oldjuk meg a $w_1, \dots, w_n, c'_1, \dots, c'_n, W$ feladatot dinamikus programozással, ahol $c'_i = \lfloor \frac{c_i}{c_{\max}} \cdot \lfloor \frac{n}{\epsilon} \rfloor \rfloor$, $c_{\max} = \max_i c_i$ és adjuk vissza ennek az eredményét.

HÁTIZSÁK $1 + \epsilon$ -approximálható

Nézzünk egy másik kézenfekvő algoritmust a HÁTIZSÁK probléma közelítő megoldására:

Skálázás ϵ -ra

- Legyen $w_1, \dots, w_n, c_1, \dots, c_n, W$ a HÁTIZSÁK probléma egy inputja és $\epsilon > 0$ egy valós szám.
- Oldjuk meg a $w_1, \dots, w_n, c'_1, \dots, c'_n, W$ feladatot dinamikus programozással, ahol $c'_i = \lfloor \frac{c_i}{c_{\max}} \cdot \lfloor \frac{n}{\epsilon} \rfloor \rfloor$, $c_{\max} = \max_i c_i$ és adjuk vissza ennek az eredményét.

Megjegyzés

Tehát mintha a maximális hasznú tárgy haszna $\lfloor \frac{n}{\epsilon} \rfloor$ lenne, a többi haszna pedig ezzel arányosan skálázódna (egészrészét véve).

A dinamikus algoritmus

A dinamikus algoritmus

A rekurzív összefüggés most:

$$T[i, c] = \begin{cases} 0 & \text{ha } c \leq 0; \\ \infty & \text{ha } 0 = i < c; \\ \min\{T[i-1, c], T[i-1, c-c_i] + w_i\} & \text{egyébként.} \end{cases}$$

A dinamikus algoritmus

A rekurzív összefüggés most:

$$T[i, c] = \begin{cases} 0 & \text{ha } c \leq 0; \\ \infty & \text{ha } 0 = i < c; \\ \min\{T[i-1, c], T[i-1, c-c_i] + w_i\} & \text{egyébként.} \end{cases}$$

„Mekkora zsák kell legalább, hogy az első i tárgyból meglegyen legalább c haszon?”

HÁTIZSÁK $1 + \epsilon$ -approximálható

A dinamikus algoritmus

A rekurzív összefüggés most:

$$T[i, c] = \begin{cases} 0 & \text{ha } c \leq 0; \\ \infty & \text{ha } 0 = i < c; \\ \min\{T[i-1, c], T[i-1, c-c_i] + w_i\} & \text{egyébként.} \end{cases}$$

„Mekkora zsák kell legalább, hogy az első i tárgyból meglegyen legalább c haszon?”

Időigény

Ennek az algoritmusnak az időigénye

$O(\text{poly}(n, n \max c, \log W)) = O(\text{poly}(n \max c \log W))$. (Pseudopolinomiális.)

HÁTI ZSÁK $1 + \epsilon$ -approximálható

$O(\text{poly}(n \max c \log W))$

HÁTI ZSÁK $1 + \epsilon$ -approximálható

$O(\text{poly}(n \max c \log W))$

Ha $\max c = \lfloor \frac{n}{\epsilon} \rfloor$, akkor ez polinom időigény az eredeti input méretében.

HÁTIZSÁK $1 + \epsilon$ -approximálható

$O(\text{poly}(n \max c \log W))$

Ha $\max c = \lfloor \frac{n}{\epsilon} \rfloor$, akkor ez polinom időigény az eredeti input méretében.
Persze nem mindig szolgáltat optimális eredményt, az osztás utáni egészrész vételekor veszített pontosság miatt.

HÁTIZSÁK $1 + \epsilon$ -approximálható

$O(\text{poly}(n \max c \log W))$

Ha $\max c = \lfloor \frac{n}{\epsilon} \rfloor$, akkor ez polinom időigény az eredeti input méretében.

Persze nem mindig szolgáltat optimális eredményt, az osztás utáni egészrész vételekor veszített pontosság miatt.

De ha rögzítjük ϵ -t, akkor **polinom** időigényű és $1 + \epsilon$ -approximáló algoritmus!

PTAS, FPTAS

PTAS

Azt mondjuk, hogy az f optimalizálási problémára van **polinomidejű approximációs séma**, avagy PTAS, ha van olyan algoritmus, melynek f inputja és egy $\epsilon > 0$ szám a bemenete és tetszőleges rögzített ϵ -ra $(1 + \epsilon)$ -approximáló polinomidejű algoritmus.

PTAS, FPTAS

PTAS

Azt mondjuk, hogy az f optimalizálási problémára van **polinomidejű approximációs séma**, avagy PTAS, ha van olyan algoritmus, melynek f inputja és egy $\epsilon > 0$ szám a bemenete és tetszőleges rögzített ϵ -ra $(1 + \epsilon)$ -approximáló polinomidejű algoritmus.

FPTAS

Azt mondjuk, hogy az f optimalizálási problémára van **teljesen polinomidejű approximációs séma**, avagy FPTAS, ha van rá olyan PTAS, mely tetszőleges (x, ϵ) inputra $\text{poly}(|x|, \frac{1}{\epsilon})$ időben fut.

PTAS, FPTAS

PTAS

Azt mondjuk, hogy az f optimalizálási problémára van **polinomidejű approximációs séma**, avagy PTAS, ha van olyan algoritmus, melynek f inputja és egy $\epsilon > 0$ szám a bemenete és tetszőleges rögzített ϵ -ra $(1 + \epsilon)$ -approximáló polinomidejű algoritmus.

FPTAS

Azt mondjuk, hogy az f optimalizálási problémára van **teljesen polinomidejű approximációs séma**, avagy FPTAS, ha van rá olyan PTAS, mely tetszőleges (x, ϵ) inputra $\text{poly}(|x|, \frac{1}{\epsilon})$ időben fut.

Az előző dián pl. egy FPTAS-t láttunk a HÁTIZSÁK problémára.

PTAS, FPTAS

PTAS

Azt mondjuk, hogy az f optimalizálási problémára van **polinomidejű approximációs séma**, avagy PTAS, ha van olyan algoritmus, melynek f inputja és egy $\epsilon > 0$ szám a bemenete és tetszőleges rögzített ϵ -ra $(1 + \epsilon)$ -approximáló polinomidejű algoritmus.

FPTAS

Azt mondjuk, hogy az f optimalizálási problémára van **teljesen polinomidejű approximációs séma**, avagy FPTAS, ha van rá olyan PTAS, mely tetszőleges (x, ϵ) inputra $\text{poly}(|x|, \frac{1}{\epsilon})$ időben fut.

Az előző dián pl. egy FPTAS-t láttunk a HÁTIZSÁK problémára.

Ha egy problémára van FPTAS, azzal tetszőleges inputra tetszőleges pontossággal polinomidőben meg tudjuk határozni az optimumot.

FPTAS és pseudopolinomiális algoritmusok

Tétel

Ha egy egészértékű optimalizálási probléma

Tétel

Ha egy egészértékű optimalizálási probléma

- i) optima korlátozható az inputméret egy polinomjával

Tétel

Ha egy egészértékű optimalizálási probléma

- i) optima korlátozható az inputméret egy polinomjával
- ii) és van rá FPTAS,

Tétel

Ha egy egészértékű optimalizálási probléma

- i) optima korlátozható az inputméret egy polinomjával
 - ii) és van rá FPTAS,
- akkor van rá pseudopolinomiális algoritmus.

FPTAS és pseudopolinomiális algoritmusok

Tétel

Ha egy egészértékű optimalizálási probléma

- i) optimauma korlátozható az inputméret egy polinomjával
 - ii) és van rá FPTAS,
- akkor van rá pseudopolinomiális algoritmus.

Ötlet

Ha a fenti korlát n^c , akkor $\epsilon = \frac{1}{n^c}$ megfelelő választás lesz.

FPTAS és pseudopolinomiális algoritmusok

Tétel

Ha egy egészértékű optimalizálási probléma

- i) optimauma korlátozható az inputméret egy polinomjával
 - ii) és van rá FPTAS,
- akkor van rá pseudopolinomiális algoritmus.

Ötlet

Ha a fenti korlát n^c , akkor $\epsilon = \frac{1}{n^c}$ megfelelő választás lesz.

Következmény

Ha $\mathbf{P} \neq \mathbf{NP}$ és egy, a fenti i) tulajdonsággal rendelkező probléma eldöntési változata erősen $\mathbf{NP}$ -teljes, akkor nem lehet rá FPTAS.

Néhány további eredmény a **P** és **NP** osztályokról

Eddig minden **NP**-beli problémánkról vagy azt mutattuk meg, hogy **P**-ben van, vagy azt, hogy **NP**-teljes.

Néhány további eredmény a **P** és **NP** osztályokról

Eddig minden **NP**-beli problémánkról vagy azt mutattuk meg, hogy **P**-ben van, vagy azt, hogy **NP**-teljes.
Azonban...

Néhány további eredmény a P és NP osztályokról

Eddig minden NP -beli problémánkról vagy azt mutattuk meg, hogy P -ben van, vagy azt, hogy NP -teljes.
Azonban...

Tétel (Ladner, 1975)

Ha $P \neq NP$, akkor létezik olyan $L \in NP - P$, mely nem NP -teljes.

P és NPC közt?

P és NPC közt?

GRÁF-IZOMORFIZMUS

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

Output: izomorfak-e?

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

Output: izomorfak-e?

FAKTORIZÁLÁS

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

Output: izomorfak-e?

FAKTORIZÁLÁS

Input: $N, K > 0$ egészek.

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

Output: izomorfak-e?

FAKTORIZÁLÁS

Input: $N, K > 0$ egészek.

Output: Van-e N -nek K -nál kisebb prímosztója?

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

Output: izomorfak-e?

FAKTORIZÁLÁS

Input: $N, K > 0$ egészek.

Output: Van-e N -nek K -nál kisebb prímosztója?

A fenti két probléma egyikéről sem ismert, hogy **P**-beliek-e, sem az, hogy **NP**-teljesek lennének.

P és NPC közt?

GRÁF-IZOMORFIZMUS

Input: két gráf.

Output: izomorfak-e?

FAKTORIZÁLÁS

Input: $N, K > 0$ egészek.

Output: Van-e N -nek K -nál kisebb prímosztója?

A fenti két probléma egyikéről sem ismert, hogy **P**-beliek-e, sem az, hogy **NP**-teljesek lennének.

Megjegyzés

A FAKTORIZÁLÁSra persze van pszeudopolinomiális algoritmus, tehát ha $\mathbf{P} \neq \mathbf{NP}$, akkor nem lehet **erősen** **NP**-teljes; másfelől **coNP**-beli is, így ha $\mathbf{NP} \neq \mathbf{coNP}$, nem lehet **NP**-teljes.

A coNP osztály

Definíció

$$\text{coNP} = \{L : \bar{L} \in \text{NP}\}$$

A coNP osztály

Definíció

$$\text{coNP} = \{L : \bar{L} \in \text{NP}\}$$

Példák

ÉRVÉNYESSÉG

Adott: φ Boole-formula

Kérdés: Érvényes-e, azaz azonosan igaz-e φ ?

A coNP osztály

Definíció

$$\text{coNP} = \{L : \bar{L} \in \text{NP}\}$$

Példák

ÉRVÉNYESSÉG

Adott: φ Boole-formula

Kérdés: Érvényes-e, azaz azonosan igaz-e φ ?

HAMILTON-ÚT KOMPLEMENTER

Adott: G gráf

Kérdés: Igaz-e, hogy G nem tartalmaz Hamilton-utat?

A coNP osztály

Definíció

$$\text{coNP} = \{L : \bar{L} \in \text{NP}\}$$

Példák

ÉRVÉNYESSÉG

Adott: φ Boole-formula

Kérdés: Érvényes-e, azaz azonosan igaz-e φ ?

HAMILTON-ÚT KOMPLEMENTER

Adott: G gráf

Kérdés: Igaz-e, hogy G nem tartalmaz Hamilton-utat?

Állítás

Az ÉRVÉNYESSÉG és HAMILTON-ÚT KOMPLEMENTER problémák coNP-ben vannak.

Egy probléma akkor van

Egy probléma akkor van

- **NP**-ben, ha minden igen példányához létezik polinom méretű, polinom időben ellenőrizhető bizonyíték, és csak az igen példányokhoz létezik ilyen bizonyíték.

Egy probléma akkor van

- **NP**-ben, ha minden igen példányához létezik polinom méretű, polinom időben ellenőrizhető bizonyíték, és csak az igen példányokhoz létezik ilyen bizonyíték.
- **coNP**-ben, ha minden nem példányhoz létezik polinom méretű, polinom időben ellenőrizhető cáfolat, és csak a nem példányokhoz létezik ilyen cáfolat.

Egy probléma akkor van

- **NP**-ben, ha minden igen példányához létezik polinom méretű, polinom időben ellenőrizhető bizonyíték, és csak az igen példányokhoz létezik ilyen bizonyíték.
- **coNP**-ben, ha minden nem példányhoz létezik polinom méretű, polinom időben ellenőrizhető cáfolat, és csak a nem példányokhoz létezik ilyen cáfolat.

Állítás

Az ÉRVÉNYESSÉG és HAMILTON-ÚT KOMPLEMENTER problémák coNP-teljesek.

Állítás

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}.$$

Állítás

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}.$$

Állítás

Tegyük fel, hogy $\mathcal{C}$ zárt a visszavezetésre és L $\mathcal{C}$ -teljes.

Állítás

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}.$$

Állítás

Tegyük fel, hogy $\mathcal{C}$ zárt a visszavezetésre és $L \in \mathcal{C}$ -teljes.
Ekkor $\mathcal{C} = \mathbf{co}\mathcal{C} \Leftrightarrow L \in \mathbf{co}\mathcal{C}$.

Állítás

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}.$$

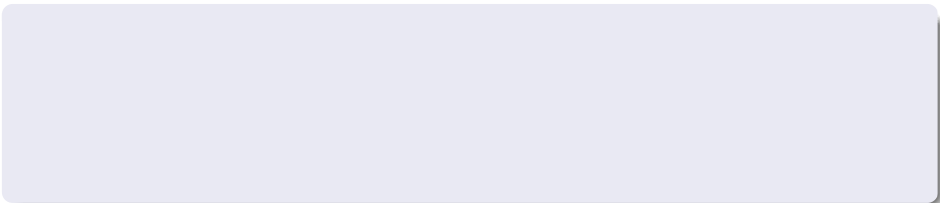
Állítás

Tegyük fel, hogy $\mathcal{C}$ zárt a visszavezetésre és L $\mathcal{C}$ -teljes.
Ekkor $\mathcal{C} = \mathbf{co}\mathcal{C} \Leftrightarrow L \in \mathbf{co}\mathcal{C}$.

Következmény

$$\mathbf{NP} = \mathbf{coNP} \Leftrightarrow \mathbf{SAT} \in \mathbf{coNP} \Leftrightarrow \mathbf{ÉRVÉNYESSÉG} \in \mathbf{NP}.$$

Determinisztikus algoritmusokkal...



- 3SAT eldönthető $\tilde{O}(1.3303^n)$ időben (Makino, Tamaki, Yamamoto, 2011),

Determinisztikus algoritmusokkal...

- 3SAT eldönthető $\tilde{O}(1.3303^n)$ időben (Makino, Tamaki, Yamamoto, 2011),
- 3-SZÍNEZÉS eldönthető $\mathcal{O}(1.3289^n)$ időben (Beigel, Eppstein, 2005),

Determinisztikus algoritmusokkal...

- 3SAT eldönthető $\tilde{O}(1.3303^n)$ időben (Makino, Tamaki, Yamamoto, 2011),
- 3-SZÍNEZÉS eldönthető $\mathcal{O}(1.3289^n)$ időben (Beigel, Eppstein, 2005),
- FÜGGETLEN CSÚCSHALMAZ eldönthető $\mathcal{O}(1.2108^n)$ időben (Robson, 1986 – Fomin, Grandoni, Kratsch 2009),

Determinisztikus algoritmusokkal...

- 3SAT eldönthető $\tilde{O}(1.3303^n)$ időben (Makino, Tamaki, Yamamoto, 2011),
- 3-SZÍNEZÉS eldönthető $\mathcal{O}(1.3289^n)$ időben (Beigel, Eppstein, 2005),
- FÜGGETLEN CSÚCSHALMAZ eldönthető $\mathcal{O}(1.2108^n)$ időben (Robson, 1986 – Fomin, Grandoni, Kratsch 2009),
- ...

Determinisztikus algoritmusokkal...

- 3SAT eldönthető $\tilde{O}(1.3303^n)$ időben (Makino, Tamaki, Yamamoto, 2011),
- 3-SZÍNEZÉS eldönthető $\mathcal{O}(1.3289^n)$ időben (Beigel, Eppstein, 2005),
- FÜGGETLEN CSÚCSHALMAZ eldönthető $\mathcal{O}(1.2108^n)$ időben (Robson, 1986 – Fomin, Grandoni, Kratsch 2009),
- ...

Az **Exponenciális Időhipotézis** azt a sejtést fogalmazza meg, hogy a 3SAT-ot (és sok más NP-teljes problémát) nem lehet **szubexponenciális** időben megoldani.

Determinisztikus algoritmusokkal...

- 3SAT eldönthető $\tilde{O}(1.3303^n)$ időben (Makino, Tamaki, Yamamoto, 2011),
- 3-SZÍNEZÉS eldönthető $\mathcal{O}(1.3289^n)$ időben (Beigel, Eppstein, 2005),
- FÜGGETLEN CSÚCSHALMAZ eldönthető $\mathcal{O}(1.2108^n)$ időben (Robson, 1986 – Fomin, Grandoni, Kratsch 2009),
- ...

Az **Exponenciális Időhipotézis** azt a sejtést fogalmazza meg, hogy a 3SAT-ot (és sok más NP-teljes problémát) nem lehet **szubexponenciális** időben megoldani.

Ez a sejtés erősebb, mint a $P \neq NP$...

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

Ismét igaz, hogy egy RAM-program esetében ha csak a használt cellák **számát** vennénk alapul, irreálisan alacsony tárigény-fogalomhoz jutnánk.

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

Ismét igaz, hogy egy RAM-program esetében ha csak a használt cellák **számát** vennénk alapul, irreálisan alacsony tárigény-fogalomhoz jutnánk.

Cellánként

Egy memóriacella igénye egy adott inputon: a benne a program futása során tárolt **legnagyobb** szám bináris alakjának a **hossza**.

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

Ismét igaz, hogy egy RAM-program esetében ha csak a használt cellák **számát** vennénk alapul, irreálisan alacsony tárigény-fogalomhoz jutnánk.

Cellánként

Egy memóriacella igénye egy adott inputon: a benne a program futása során tárolt **legnagyobb** szám bináris alakjának a **hossza**.

(Azaz $1 + \lfloor \log a \rfloor$, ha ez a maximális érték a , kivéve, ha $a == 0$, mely esetben 1).

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

Ismét igaz, hogy egy RAM-program esetében ha csak a használt cellák **számát** vennénk alapul, irreálisan alacsony tárigény-fogalomhoz jutnánk.

Cellánként

Egy memóriacella igénye egy adott inputon: a benne a program futása során tárolt **legnagyobb** szám bináris alakjának a **hossza**.

(Azaz $1 + \lfloor \log a \rfloor$, ha ez a maximális érték a , kivéve, ha $a == 0$, mely esetben 1).

PLUSZ

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

Ismét igaz, hogy egy RAM-program esetében ha csak a használt cellák **számát** vennénk alapul, irreálisan alacsony tárigény-fogalomhoz jutnánk.

Cellánként

Egy memóriacella igénye egy adott inputon: a benne a program futása során tárolt **legnagyobb** szám bináris alakjának a **hossza**.

(Azaz $1 + \lfloor \log a \rfloor$, ha ez a maximális érték a , kivéve, ha $a == 0$, mely esetben 1).
PLUSZ a cella **címének** bináris alakjának a hossza.

A futásidő mellett a felhasznált tárterület a másik fontos erőforrás.

Ismét igaz, hogy egy RAM-program esetében ha csak a használt cellák **számát** vennénk alapul, irreálisan alacsony tárigény-fogalomhoz jutnánk.

Cellánként

Egy memóriacella igénye egy adott inputon: a benne a program futása során tárolt **legnagyobb** szám bináris alakjának a **hossza**.

(Azaz $1 + \lfloor \log a \rfloor$, ha ez a maximális érték a , kivéve, ha $a == 0$, mely esetben 1).
PLUSZ a cella **címének** bináris alakjának a hossza.

Példa

Ha a $W[7]$ regiszterben a program futása során a legnagyobb érték **20**, akkor ennek a regiszternek a tárigénye 5 (a $20 = 10100_2$ string hossza) plusz 3 (a $7 = 111_2$ string hossza) **= 8**.

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

A két memóriamodell

Amint egy cellát megcímez a program (direkt vagy indirekt címezéssel, írásra vagy olvasásra), használatnak minősül.

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

A két memóriamodell

Amint egy cellát megcímez a program (direkt vagy indirekt címezéssel, írásra vagy olvasásra), használnak minősül.

Kivéve a következő esetet:

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

A két memóriamodell

Amint egy cellát megcímez a program (direkt vagy indirekt címezéssel, írásra vagy olvasásra), használnak minősül.

Kivéve a következő esetet:

- ha a program az **input** regisztereket **csak olvassa**,

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

A két memóriamodell

Amint egy cellát megcímez a program (direkt vagy indirekt címezéssel, írásra vagy olvasásra), használnak minősül.

Kivéve a következő esetet:

- ha a program az **input** regisztereket **csak olvassa**,
- az **output** regiszterekbe pedig **csak ír**, ráadásul stream módban: először $O[1]$ kap értéket, majd $O[2]$, $O[3]$ stb.

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

A két memóriamodell

Amint egy cellát megcímez a program (direkt vagy indirekt címezéssel, írásra vagy olvasásra), használnak minősül.

Kivéve a következő esetet:

- ha a program az **input** regisztereket **csak olvassa**,
- az **output** regiszterekben pedig **csak ír**, ráadásul stream módban: először $O[1]$ kap értéket, majd $O[2]$, $O[3]$ stb.

akkor az input és output regiszterek tárigényét nem kell bevenni a tárigénybe. (Ez az ún. „**offline**” vagy „lyukszalagos” eset.)

Tárigény egy adott inputon

A teljes program tárigénye egy adott inputon: az összes, a futás során **használt** cella tárigényének összege.

A két memóriamodell

Amint egy cellát megcímez a program (direkt vagy indirekt címezéssel, írásra vagy olvasásra), használnak minősül.

Kivéve a következő esetet:

- ha a program az **input** regisztereket **csak olvassa**,
- az **output** regiszterekbe pedig **csak ír**, ráadásul stream módban: először $O[1]$ kap értéket, majd $O[2]$, $O[3]$ stb.

akkor az input és output regiszterek tárigényét nem kell bevenni a tárigénybe. (Ez az ún. „**offline**” vagy „lyukszalagos” eset.)

(Ez megfelel annak a memóriamodellnek, mikor a hívó fél biztosítja az I/O területet: az inputot nem írhatjuk át, az outputot pedig egy output stream formájában kapjuk.)

A program tárigénye

A program tárigénye az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha bármely, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A program tárigénye

A program tárigénye az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha bármely, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A probléma tárigénye

Egy probléma **eldönthető $\mathcal{O}(f(n))$ tárban**, ha van őt eldöntő, $\mathcal{O}(f(n))$ tárigényű program.

Tárbonyolultság

A program tárigénye

A program tárigénye az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha bármely, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A probléma tárigénye

Egy probléma **eldönthető $\mathcal{O}(f(n))$ tárban**, ha van őt eldöntő, $\mathcal{O}(f(n))$ tárigényű program. Ezen problémák osztályát $\text{SPACE}(f(n))$ jelöli.

Tárbonyolultság

A program tárigénye

A program tárigénye az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha bármely, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A probléma tárigénye

Egy probléma **eldönthető $\mathcal{O}(f(n))$ tárban**, ha van őt eldöntő, $\mathcal{O}(f(n))$ tárigényű program. Ezen problémák osztályát $\text{SPACE}(f(n))$ jelöli.

Példa

ELÉRHETŐSÉG eldönthető **lineáris** tárban:

A program tárigénye

A program tárigénye az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha bármely, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A probléma tárigénye

Egy probléma **eldönthető $\mathcal{O}(f(n))$ tárban**, ha van őt eldöntő, $\mathcal{O}(f(n))$ tárigényű program. Ezen problémák osztályát $\text{SPACE}(f(n))$ jelöli.

Példa

ELÉRHETŐSÉG eldönthető **lineáris** tárban:

- a szélességi keresés algoritmus a egy N -csúcsú gráf esetén két, egyenként legfeljebb N méretű csúcshalmazt tárol

A program tárigénye

A program tárigénye az $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ függvény, ha bármely, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A probléma tárigénye

Egy probléma **eldönthető $\mathcal{O}(f(n))$ tárban**, ha van őt eldöntő, $\mathcal{O}(f(n))$ tárigényű program. Ezen problémák osztályát $\text{SPACE}(f(n))$ jelöli.

Példa

ELÉRHETŐSÉG eldönthető **lineáris** tárban:

- a szélességi keresés algoritmus a egy N -csúcsú gráf esetén két, egyenként legfeljebb N méretű csúcshalmazt tárol (a már elért ill. elért és kifejtett csúcso két), ez pl. N hosszú bitvektor alkalmazásával egy-egy regiszterben $\mathcal{O}(N)$ tárban megoldható.

Alapvető különbség tár és idő közt

A tár újrafelhasználható!

Alapvető különbség tár és idő közt

A tár újrafelhasználható!

Lineáris tárban...

- ...eldönthető a HAMILTON-ÚT probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes permutációját
- ...eldönthető a $TSP(E)$ probléma is, hasonlóképp
- ...eldönthető a 3-SZÍNEZÉS probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes 3-színezését
- ...

Alapvető különbség tár és idő közt

A tár újrafelhasználható!

Lineáris tárban...

- ...eldönthető a HAMILTON-ÚT probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes permutációját
- ...eldönthető a $TSP(E)$ probléma is, hasonlóképp
- ...eldönthető a 3-SZÍNEZÉS probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes 3-színezését
- ...

Megjegyzés

Nem ismert, hogy **NP** és $SPACE(n)$ közt mi az összefüggés.

Alapvető különbség tár és idő közt

A tár újrafelhasználható!

Lineáris tárban...

- ... eldönthető a HAMILTON-ÚT probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes permutációját
- ... eldönthető a $TSP(E)$ probléma is, hasonlóképp
- ... eldönthető a 3-SZÍNEZÉS probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes 3-színezését
- ...

Megjegyzés

Nem ismert, hogy NP és $SPACE(n)$ közt mi az összefüggés.
Csak annyit tudunk biztosan, hogy $NP \neq SPACE(n)$.

Alapvető különbség tár és idő közt

A tár újrafelhasználható!

Lineáris tárban...

- ... eldönthető a HAMILTON-ÚT probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes permutációját
- ... eldönthető a $TSP(E)$ probléma is, hasonlóképp
- ... eldönthető a 3-SZÍNEZÉS probléma is: ugyanazt a területet újrahasználva generáljuk a csúcsok összes 3-színezését
- ...

Megjegyzés

Nem ismert, hogy NP és $SPACE(n)$ közt mi az összefüggés.

Csak annyit tudunk biztosan, hogy $NP \neq SPACE(n)$.

(NP zárt a visszavezetésre, $SPACE(n)$ pedig – mint látni fogjuk – nem az.)

Nemdeterminisztikus tárbonyolultság

Nemdeterminisztikus program **idő**bonyolultságát a **leghosszabb** szál hosszaként definiáltuk.

Nemdeterminisztikus tárbonyolultság

Nemdeterminisztikus program **időbonyolultságát** a **leghosszabb** szál hosszaként definiáltuk.

Nemdeterminisztikus program **tárbonyolultságát** egy adott inputon hasonló módon: a **legtöbb tárat használó szál** tárigényeként definiáljuk.

Nemdeterminisztikus tárbonyolultság

Nemdeterminisztikus program **időbonyolultságát** a **leghosszabb** szál hosszaként definiáltuk.

Nemdeterminisztikus program **tárbonyolultságát** egy adott inputon hasonló módon: a **legtöbb tárat használó szál** tárigényeként definiáljuk.

A nemdeterminisztikus program tárbonyolultsága szintén akkor $f(n)$, ha tetszőleges, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

Nemdeterminisztikus tárbonyolultság

Nemdeterminisztikus program **időbonyolultságát** a **leghosszabb** szál hosszaként definiáltuk.

Nemdeterminisztikus program **tárbonyolultságát** egy adott inputon hasonló módon: a **legtöbb tárat használó szál** tárigényeként definiáljuk.

A nemdeterminisztikus program tárbonyolultsága szintén akkor $f(n)$, ha tetszőleges, legfeljebb n méretű inputon legfeljebb $f(n)$ tárat használ.

A nemdeterminisztikus programmal $\mathcal{O}(f(n))$ tárban eldönthető **problémák** osztályát $\text{NSPACE}(f(n))$ jelöli.

Példa

```
function ND-ELER( $G[1 \dots N][1 \dots N], s, t$ )  
   $u := s$   
  while  $u \neq t$  do  
     $v := \text{ND}(1 \dots N)$  //  $v$  értéke nemdet. 1 és  $N$  közti egész  
    if  $G[u][v] == 0$  then return FALSE  
     $u := v$   
  return TRUE
```

Példa

```
function ND-ELER( $G[1 \dots N][1 \dots N], s, t$ )  
   $u := s$   
  while  $u \neq t$  do  
     $v := \text{ND}(1 \dots N)$  //  $v$  értéke nemdet. 1 és  $N$  közti egész  
    if  $G[u][v] == 0$  then return FALSE  
     $u := v$   
  return TRUE
```

Tárigény: nemdeterminisztikus **logaritmikus**: $\mathcal{O}(\log n)$.

Példa

```
function ND-ELER( $G[1 \dots N][1 \dots N], s, t$ )  
   $u := s$   
  while  $u \neq t$  do  
     $v := \text{ND}(1 \dots N)$  //  $v$  értéke nemdet. 1 és  $N$  közti egész  
    if  $G[u][v] == 0$  then return FALSE  
     $u := v$   
  return TRUE
```

Tárigény: nemdeterminisztikus **logaritmikus**: $\mathcal{O}(\log n)$.

Eldönti az ELÉRHETŐSÉG problémát.

Nemdeterminisztikus tárkonyolultság

Példa

```
function ND-ELER( $G[1 \dots N][1 \dots N], s, t$ )  
   $u := s$   
  while  $u \neq t$  do  
     $v := \text{ND}(1 \dots N)$  //  $v$  értéke nemdet. 1 és  $N$  közti egész  
    if  $G[u][v] == 0$  then return FALSE  
     $u := v$   
  return TRUE
```

Tárigény: nemdeterminisztikus **logaritmikus**: $\mathcal{O}(\log n)$.

Eldönti az ELÉRHETŐSÉG problémát.

(Plusz egy, N -ig menő ciklusszámlálóval a végtelen ciklus elkerülhető, továbbra is logaritmikus tárat használva.)

A nevesített bonyolultsági osztályok

- **R**, **RE** – eldönthető ill. felismerhető problémák
- $\text{TIME}(f(n))$, $\text{NTIME}(f(n))$ – det/nemdet. $\mathcal{O}(f(n))$ időben eldönthető problémák
- $\text{SPACE}(f(n))$, $\text{NSPACE}(f(n))$ – det/nemdet. $\mathcal{O}(f(n))$ tárban eldönthető problémák
- $\mathbf{P} = \text{TIME}(n^k) = \bigcup_{k \geq 0} \text{TIME}(n^k)$ – polinomidőben eldönthető problémák
- $\mathbf{NP} = \text{NTIME}(n^k)$ – nemdet. polinomidőben eldönthető problémák
- **L** = $\text{SPACE}(\log n)$ – logaritmikus tárban eldönthető problémák
- **NL** = $\text{NSPACE}(\log n)$ – nemdet. logtárban eldönthetőek
- **PSPACE** = $\text{SPACE}(n^k)$ – polinom tárban eldönthetőek
- **NPSPACE** = $\text{NSPACE}(n^k)$ – nemdet. polinom tárban...
- **EXP** = $\text{TIME}(2^{n^k}) \dots$

Alapvető összefüggések bonyolultsági osztályok közt

Alapvető összefüggések bonyolultsági osztályok közt

i) $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$, $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$

Alapvető összefüggések bonyolultsági osztályok közt

- i) $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)), \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$
- ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Alapvető összefüggések bonyolultsági osztályok közt

- i) $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)), \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$
- ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$
- iii) $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$

Alapvető összefüggések

Alapvető összefüggések bonyolultsági osztályok közt

- i) $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)), \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$
- ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$
- iii) $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$

Ötletek

Az i) pont világos.

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Alapvető összefüggések

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

Alapvető összefüggések

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé (ennyi lépésig szimuláljuk a nemdeterminisztikus programot).

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé (ennyi lépésig szimuláljuk a nemdeterminisztikus programot).
- k minden értékére leszimuláljuk az összes k hosszú választási sorozatot (ez 2^k darab):

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé (ennyi lépésig szimuláljuk a nemdeterminisztikus programot).
- k minden értékére leszimuláljuk az összes k hosszú választási sorozatot (ez 2^k darab):
 - ha van köztük elfogadó, elfogadjuk az inputot;

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé (ennyi lépésig szimuláljuk a nemdeterminisztikus programot).
- k minden értékére leszimuláljuk az összes k hosszú választási sorozatot (ez 2^k darab):
 - ha van köztük elfogadó, elfogadjuk az inputot;
 - ha mind elutasító, elvetjük az inputot;

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé (ennyi lépésig szimuláljuk a nemdeterminisztikus programot).
- k minden értékére leszimuláljuk az összes k hosszú választási sorozatot (ez 2^k darab):
 - ha van köztük elfogadó, elfogadjuk az inputot;
 - ha mind elutasító, elvetjük az inputot;
 - különben növeljük k -t és folytatjuk a ciklust.

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Tehát egy $f(n)$ **idő**korlátos N nemdeterminisztikus programot lehet szimulálni $f^2(n)$ **tár**korlátos determinisztikus programmal.

Feltehetjük, hogy N -nek mindig **pontosan két** választása van megállásig, a 0. és az 1.

A determinisztikus algoritmus:

- Futtatunk egy k számlálót 1-től fölfelé (ennyi lépésig szimuláljuk a nemdeterminisztikus programot).
- k minden értékére leszimuláljuk az összes k hosszú választási sorozatot (ez 2^k darab):
 - ha van köztük elfogadó, elfogadjuk az inputot;
 - ha mind elutasító, elvetjük az inputot;
 - különben növeljük k -t és folytatjuk a ciklust.

(Kimaradt részlet: t utasítás végrehajtása alatt egy RAM program $\mathcal{O}(t^2)$ memóriát használ – ez pl. ismét annak a következménye, hogy a szorzás **nem** elemi művelet, az összeadás nem növelheti „túlzott ütemben” a változók értékét)

$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$, tovább

$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$, tovább

A szimulálást a tárterület újrafelhasználásával tesszük (feltehetjük, hogy N offline, így az inputot a szimuláció nem rontja el).

$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$, tovább

A szimulálást a tárterület újrafelhasználásával tesszük (feltehetjük, hogy N offline, így az inputot a szimuláció nem rontja el).

A k . menetben egy k hosszú bitvektorban tartjuk nyilván az aktuális választási sorozatot, ezt növeljük mondjuk binárisan 0-ról indítva.

$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f^2(n))$, tovább

A szimulálást a tárterület újrafelhasználásával tesszük (feltehetjük, hogy N offline, így az inputot a szimuláció nem rontja el).

A k . menetben egy k hosszú bitvektorban tartjuk nyilván az aktuális választási sorozatot, ezt növeljük mondjuk binárisan 0-ról indítva.

Mivel $f(n)$ lépésben N is csak $f^2(n)$ tárat használ, a szimulációhoz elég $f^2(n)$ tár; mivel $k \leq f(n)$, a bitvektornak is elég $f(n)$ tár.

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$$

Tehát egy $f(n)$ tárkorlátos N nemdeterminisztikus programot akarunk determinisztikusan szimulálni.

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$$

Tehát egy $f(n)$ tárkorlátos N nemdeterminisztikus programot akarunk determinisztikusan szimulálni.

Feltehetjük, hogy

- N offline;
- elfogadás előtt közvetlenül kinullázza munkaregisztereit.

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$$

Tehát egy $f(n)$ tárkorlátos N nemdeterminisztikus programot akarunk determinisztikusan szimulálni.

Feltehetjük, hogy

- N offline;
- elfogadás előtt közvetlenül kinullázza munkaregisztereit.

Nevezzük **konfigurációnak** a RAM program teljes leírását a számítás egy pillanatában, miután az n hosszú $(a_1, \dots, a_n)$ inputon elindítjuk:

- a programszámláló értéke: k_1 konstans sok lehetőség
- az összes nemnulla munkaregiszter aktuális tartalma $(i, W[i])$ párok listájának formájában: $\mathcal{O}(f(n))$ bit hosszú lista, $2^{\mathcal{O}(f(n))}$ -féle lehetőség

Összesen $k_1 \cdot 2^{\mathcal{O}(f(n))} = k^{f(n)}$ konfiguráció.

$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$, tovább

$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$, tovább

Az adott inputhoz tartozó **konfigurációs gráf**: a konfigurációk a csúcsok, C_1 -ből akkor vezet él C_2 -be, ha N átléphet C_1 -ből C_2 -be.

$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$, tovább

Az adott inputhoz tartozó **konfigurációs gráf**: a konfigurációk a csúcsok, C_1 -ből akkor vezet él C_2 -be, ha N átléphet C_1 -ből C_2 -be.

A **determinisztikus** algoritmus: elkészítjük az inputhoz tartozó konfigurációs gráfot és azon lineáris időben megoldunk egy ELÉRHETŐSÉG problémát a kezdőkonfigurációból a (kinullázás miatt egyértelmű) elfogadó konfigurációba.

$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$, tovább

Az adott inputhoz tartozó **konfigurációs gráf**: a konfigurációk a csúcsok, C_1 -ből akkor vezet él C_2 -be, ha N átléphet C_1 -ből C_2 -be.

A **determinisztikus** algoritmus: elkészítjük az inputhoz tartozó konfigurációs gráfot és azon lineáris időben megoldunk egy ELÉRHETŐSÉG problémát a kezdőkonfigurációból a (kinullázás miatt egyértelmű) elfogadó konfigurációba. (A gráfot nem kell előre elkészítenünk és letárolnunk, on-the-fly is számíthatjuk.)

Alapvető összefüggések

$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{f(n)})$, tovább

Az adott inputhoz tartozó **konfigurációs gráf**: a konfigurációk a csúcsok, C_1 -ből akkor vezet él C_2 -be, ha N átléphet C_1 -ből C_2 -be.

A **determinisztikus** algoritmus: elkészítjük az inputhoz tartozó konfigurációs gráfot és azon lineáris időben megoldunk egy ELÉRHETŐSÉG problémát a kezdőkonfigurációból a (kinullázás miatt egyértelmű) elfogadó konfigurációba. (A gráfot nem kell előre elkészítenünk és letárolnunk, on-the-fly is számíthatjuk.)

Megjegyzés

Az eddigiekből $\text{NTIME}(f(n)) \subseteq \text{TIME}(k^{f^2(n)})$, de $\text{NTIME}(f(n)) \subseteq \text{TIME}(k^{f(n)})$ is igaz.

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}(\subsetneq \mathbf{R})$$

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}(\subsetneq \mathbf{R})$$

Bizonyítás

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}(\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} (\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n) \subseteq \mathbf{TIME}(k^{\log n}) \subseteq \mathbf{P} \quad (\text{iii})$$

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}(\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n) \subseteq \mathbf{TIME}(k^{\log n}) \subseteq \mathbf{P} \quad (\text{iii})$$

$$\mathbf{P} \subseteq \mathbf{NP} \quad (\text{i})$$

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}(\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n) \subseteq \mathbf{TIME}(k^{\log n}) \subseteq \mathbf{P} \quad (\text{iii})$$

$$\mathbf{P} \subseteq \mathbf{NP} \quad (\text{i})$$

$$\mathbf{NP} = \mathbf{NTIME}(n^k) \subseteq \mathbf{SPACE}(n^k) = \mathbf{PSPACE} \quad (\text{ii})$$

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} (\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n) \subseteq \mathbf{TIME}(k^{\log n}) \subseteq \mathbf{P} \quad (\text{iii})$$

$$\mathbf{P} \subseteq \mathbf{NP} \quad (\text{i})$$

$$\mathbf{NP} = \mathbf{NTIME}(n^k) \subseteq \mathbf{SPACE}(n^k) = \mathbf{PSPACE} \quad (\text{ii})$$

$$\mathbf{PSPACE} = \mathbf{SPACE}(n^k) \subseteq \mathbf{TIME}(2^{n^k}) = \mathbf{EXP} \quad (\text{iii})$$

A **központi** kérdés, hogy $\mathbf{P} = \mathbf{NP}$ vagy sem,

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} (\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n) \subseteq \mathbf{TIME}(k^{\log n}) \subseteq \mathbf{P} \quad (\text{iii})$$

$$\mathbf{P} \subseteq \mathbf{NP} \quad (\text{i})$$

$$\mathbf{NP} = \mathbf{NTIME}(n^k) \subseteq \mathbf{SPACE}(n^k) = \mathbf{PSPACE} \quad (\text{ii})$$

$$\mathbf{PSPACE} = \mathbf{SPACE}(n^k) \subseteq \mathbf{TIME}(2^{n^k}) = \mathbf{EXP} \quad (\text{iii})$$

A **központi** kérdés, hogy $\mathbf{P} = \mathbf{NP}$ vagy sem, de a fentiek közül **egyiket** se tudjuk, hogy egyenlőség-e!

A hierarchia tételek

Megengedett függvények

Az $f : \mathbb{N} \rightarrow \mathbb{N}$ monoton növény **megengedett**, ha létezik olyan program, ami az $1^n \mapsto 1^{f(n)}$ függvényt számítja ki (n darab egyesből $f(n)$ darab egyest készít) $\mathcal{O}(f(n))$ tárban és $\mathcal{O}(n + f(n))$ időben.

A hierarchia tételek

Megengedett függvények

Az $f : \mathbb{N} \rightarrow \mathbb{N}$ monoton növény **megengedett**, ha létezik olyan program, ami az $1^n \mapsto 1^{f(n)}$ függvényt számítja ki (n darab egyesből $f(n)$ darab egyest készít) $\mathcal{O}(f(n))$ tárban és $\mathcal{O}(n + f(n))$ időben.

Időhierarchia tétel

Ha $f(n) > n$ megengedett függvény, akkor

$$\text{TIME}\left(o\left(\frac{f(n)}{\log f(n)}\right)\right) \subsetneq \text{TIME}(f(n)).$$

Emiatt pl. $\mathbf{P} \neq \mathbf{EXP}$.

A hierarchia tételek

Megengedett függvények

Az $f : \mathbb{N} \rightarrow \mathbb{N}$ monoton növény függvény **megengedett**, ha létezik olyan program, ami az $1^n \mapsto 1^{f(n)}$ függvényt számítja ki (n darab egyesből $f(n)$ darab egyest készít) $\mathcal{O}(f(n))$ tárban és $\mathcal{O}(n + f(n))$ időben.

Időhierarchia tétel

Ha $f(n) > n$ megengedett függvény, akkor

$$\text{TIME}\left(o\left(\frac{f(n)}{\log f(n)}\right)\right) \subsetneq \text{TIME}(f(n)).$$

Emiatt pl. $\mathbf{P} \neq \mathbf{EXP}$.

Tárhierarchia tétel

Ha $f(n) > \log n$ megengedett függvény, akkor

$$\text{SPACE}\left(o(f(n))\right) \subsetneq \text{SPACE}(f(n)).$$

Emiatt pl. $\mathbf{L} \neq \mathbf{PSPACE}$.

Egy algoritmus

Egy algoritmus

- Input: irányított gráf, mondjuk $G[1..N][1..N]$ szomszédsági mátrixával

Egy algoritmus

- Input: irányított gráf, mondjuk $G[1..N][1..N]$ szomszédsági mátrixával
- Output: ?

```
function  $F(x, y, d)$   
  if  $d == 0$  then return  $(x == y)$  OR  $G[x][y]$ ;  
  for  $z = 1 \dots N$  do  
    if  $F(x, z, d - 1)$  AND  $F(z, y, d - 1)$  then return TRUE  
  return FALSE  
return  $F(1, N, \lceil \log N \rceil)$ ;
```

A függvény

Az $f(x, y, d)$ hívás pontosan akkor ad TRUE-t, ha a G gráfban van x -ből y -ba legfeljebb 2^d hosszú út.

A függvény

Az $f(x, y, d)$ hívás pontosan akkor ad TRUE-t, ha a G gráfban van x -ből y -ba legfeljebb 2^d hosszú út.

- $d == 0$: legfeljebb egy hosszú út, vagyis $x == y$ vagy van él x -ből y -ba;

A függvény

Az $f(x, y, d)$ hívás pontosan akkor ad TRUE-t, ha a G gráfban van x -ből y -ba legfeljebb 2^d hosszú út.

- $d == 0$: legfeljebb egy hosszú út, vagyis $x == y$ vagy van él x -ből y -ba;
- $d > 0$: ha van x -ből y -ba egy legfeljebb 2^d hosszú út, akkor ennek felezőpontjába, z -be van x -ből egy legfeljebb 2^{d-1} hosszú út és z -ből y -ba is, ez akkor teljesül, ha $f(x, z, d - 1)$ és $f(z, y, d - 1)$ is TRUE-val tér vissza.

A függvény

Az $f(x, y, d)$ hívás pontosan akkor ad TRUE-t, ha a G gráfban van x -ből y -ba legfeljebb 2^d hosszú út.

- $d == 0$: legfeljebb egy hosszú út, vagyis $x == y$ vagy van él x -ből y -ba;
- $d > 0$: ha van x -ből y -ba egy legfeljebb 2^d hosszú út, akkor ennek felezőpontjába, z -be van x -ből egy legfeljebb 2^{d-1} hosszú út és z -ből y -ba is, ez akkor teljesül, ha $f(x, z, d - 1)$ és $f(z, y, d - 1)$ is TRUE-val tér vissza.

A belépési pont

Az $f(1, N, \lceil \log N \rceil)$ hívás tehát akkor ad TRUE-t, ha van legfeljebb $2^{\lceil \log N \rceil} \geq N$ hosszú út 1-ből N -be,

A függvény

Az $f(x, y, d)$ hívás pontosan akkor ad TRUE-t, ha a G gráfban van x -ből y -ba legfeljebb 2^d hosszú út.

- $d == 0$: legfeljebb egy hosszú út, vagyis $x == y$ vagy van él x -ből y -ba;
- $d > 0$: ha van x -ből y -ba egy legfeljebb 2^d hosszú út, akkor ennek felezőpontjába, z -be van x -ből egy legfeljebb 2^{d-1} hosszú út és z -ből y -ba is, ez akkor teljesül, ha $f(x, z, d - 1)$ és $f(z, y, d - 1)$ is TRUE-val tér vissza.

A belépési pont

Az $f(1, N, \lceil \log N \rceil)$ hívás tehát akkor ad TRUE-t, ha van legfeljebb $2^{\lceil \log N \rceil} \geq N$ hosszú **út 1-ből N -be**, vagyis egy (determinisztikus) algoritmus az ELÉRHETŐSÉG problémára.

Tárigény

Tárigény

- Az f függvény egy példányának tárigénye $\mathcal{O}(\log n)$, hisz csak az x, y, d, z változókat deklarálja, mindegyik $0 \dots N$ közti értéket vesz fel;

Tárigény

- Az f függvény egy példányának tárigénye $\mathcal{O}(\log n)$, hisz csak az x, y, d, z változókat deklarálja, mindegyik $0 \dots N$ közti értéket vesz fel;
- a hívási verem mélysége $\lceil \log N \rceil = \mathcal{O}(\log n)$, legfeljebb ennyi példánya van egyszerre f -nek a memóriában;

Tárigény

- Az f függvény egy példányának tárigénye $\mathcal{O}(\log n)$, hisz csak az x, y, d, z változókat deklarálja, mindegyik $0 \dots N$ közti értéket vesz fel;
- a hívási verem mélysége $\lceil \log N \rceil = \mathcal{O}(\log n)$, legfeljebb ennyi példánya van egyszerre f -nek a memóriában;

tehát a teljes tárigény $\mathcal{O}(\log^2 n)$.

Tárigény

- Az f függvény egy példányának tárigénye $\mathcal{O}(\log n)$, hisz csak az x, y, d, z változókat deklarálja, mindegyik $0 \dots N$ közti értéket vesz fel;
- a hívási verem mélysége $\lceil \log N \rceil = \mathcal{O}(\log n)$, legfeljebb ennyi példánya van egyszerre f -nek a memóriában;

tehát a teljes tárigény $\mathcal{O}(\log^2 n)$.

Ezzel beláttuk Savitch tételét:

Tárigény

- Az f függvény egy példányának tárigénye $\mathcal{O}(\log n)$, hisz csak az x, y, d, z változókat deklarálja, mindegyik $0 \dots N$ közti értéket vesz fel;
- a hívási verem mélysége $\lceil \log N \rceil = \mathcal{O}(\log n)$, legfeljebb ennyi példánya van egyszerre f -nek a memóriában;

tehát a teljes tárigény $\mathcal{O}(\log^2 n)$.

Ezzel beláttuk Savitch tételét:

Tétel

$$\text{ELÉRHETŐSÉG} \in \text{SPACE}(\log^2 n).$$

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)),$$

ha $f(n) \geq \log n$.

Következmény

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)),$$

ha $f(n) \geq \log n$.

Ötlet

Az ELÉRHETŐSÉG problémát a nemdeterminisztikus program **konfigurációs gráfján** oldjuk meg.

Következmény

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)),$$

ha $f(n) \geq \log n$.

Ötlet

Az ELÉRHETŐSÉG problémát a nemdeterminisztikus program **konfigurációs gráfján** oldjuk meg.

Következmény

$$\text{PSPACE} = \text{NPSPACE}.$$

Következmény

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)),$$

ha $f(n) \geq \log n$.

Ötlet

Az ELÉRHETŐSÉG problémát a nemdeterminisztikus program **konfigurációs gráfján** oldjuk meg.

Következmény

$$\text{PSPACE} = \text{NPSPACE}.$$

Következmény

$$\text{NL} \subsetneq \text{PSPACE}.$$

(Hiszen $\text{NL} \subseteq \text{SPACE}(\log^2 n) \subsetneq \text{SPACE}(n)$ Savitch tétele és a tárhierarchia tétel szerint.)

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmus

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

function $\hat{U}_T(s, t, d)$

$i := s;$

while $d \geq 0$ **do**

if $i == t$ **then return** ACCEPT

$j := \text{nd}(N);$

if NOT $G[i][j]$ **then return**

REJECT

$i := j;$

$d := d - 1;$

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

function $\hat{U}_T(s, t, d)$

$i := s;$

while $d \geq 0$ **do**

if $i == t$ **then return** ACCEPT

$j := \text{nd}(N);$

if NOT $G[i][j]$ **then return**

REJECT

$i := j;$

$d := d - 1;$

- A függvény visszaad **hat** ACCEPT-et, ha s -ből t lépésen belül elérhető.

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

function $\hat{U}_T(s, t, d)$

$i := s;$

while $d \geq 0$ **do**

if $i == t$ **then return** ACCEPT

$j := \text{nd}(N);$

if NOT $G[i][j]$ **then return**

REJECT

$i := j;$

$d := d - 1;$

- A függvény visszaad **hat** ACCEPT-et, ha s -ből t lépésen belül elérhető.
- Egyébként mindenképp REJECT-et ad vissza.

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

function $\hat{U}_T(s, t, d)$

$i := s;$

while $d \geq 0$ **do**

if $i == t$ **then return** ACCEPT

$j := \text{nd}(N);$

if NOT $G[i][j]$ **then return**

REJECT

$i := j;$

$d := d - 1;$

- A függvény visszaad **hat** ACCEPT-et, ha s -ből t d lépésen belül elérhető.
- Egyébként mindenképp REJECT-et ad vissza.
- Tárigény: $\mathcal{O}(\log n)$.

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmust hívó algoritmus

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

```
function T-IN-SK( $s, t, prev, k$ )  
     $check := 0$ ;  
    for  $u := 1 \dots N$  do  
        if  $\hat{U}_T(s, u, k - 1)$  then  
             $check := check + 1$ ;  
            if  $G[u][t]$  then return  
ACCEPT  
    if  $check == prev$  then return  
REJECT  
    THROW ERROR;
```


Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

function T-IN-SK($s, t, prev, k$)

$check := 0;$

for $u := 1 \dots N$ **do**

if $\hat{U}_T(s, u, k - 1)$ **then**

$check := check + 1;$

if $G[u][t]$ **then return**

ACCEPT

if $check == prev$ **then return**

REJECT

 THROW ERROR;

- **Ha** $prev$ az s -ből legfeljebb $k - 1$ lépésben elérhető csúcsok száma, akkor:

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

```
function T-IN-SK( $s, t, prev, k$ )  
   $check := 0$ ;  
  for  $u := 1 \dots N$  do  
    if  $\dot{U}_T(s, u, k - 1)$  then  
       $check := check + 1$ ;  
      if  $G[u][t]$  then return  
ACCEPT  
  if  $check == prev$  then return  
REJECT  
  throw ERROR;
```

- Ha $prev$ az s -ből legfeljebb $k - 1$ lépésben elérhető csúcsok száma, akkor:
- ha s -ből t legfeljebb k lépésben elérhető, akkor az eredmény ACCEPT vagy ERROR, lehet ACCEPT;

Az Immerman-Szelepcsényi tétel

Egy nemdeterminisztikus algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával.

```
function T-IN-SK( $s, t, prev, k$ )  
   $check := 0$ ;  
  for  $u := 1 \dots N$  do  
    if  $\dot{U}_T(s, u, k - 1)$  then  
       $check := check + 1$ ;  
      if  $G[u][t]$  then return  
ACCEPT  
  if  $check == prev$  then return  
REJECT  
  throw ERROR;
```

- **Ha** $prev$ az s -ből legfeljebb $k - 1$ lépésben elérhető csúcsok száma, akkor:
- ha s -ből t legfeljebb k lépésben elérhető, akkor az eredmény ACCEPT vagy ERROR, lehet ACCEPT;
- ha nem, akkor az eredmény REJECT vagy ERROR, lehet REJECT.

Az Immerman-Szelepcsényi tétel

Egy nd algoritmust hívó algoritmust hívó algoritmus

Az Immerman-Szelepcsényi tétel

Egy nd algoritmust hívó algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával és s csúcsa.

```
prev := 1;  
for  $k := 1 \dots N$  do  
    current := 0;  
    for  $t := 1 \dots N$  do  
        if T-IN-SK( $s, t, prev, k$ ) then  
            current := current + 1;  
    prev := current;  
return prev;
```

Az Immerman-Szelepcsényi tétel

Egy nd algoritmust hívó algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával és s csúcsa.

```
prev := 1;  
for  $k := 1 \dots N$  do  
    current := 0;  
    for  $t := 1 \dots N$  do  
        if T-IN-SK( $s, t, prev, k$ ) then  
            current := current + 1;  
    prev := current;  
return prev;
```

- Az algoritmus visszaadja az s -ből elérhető csúcsok számát, vagy ERROR-t dob; van, mikor a számot adja vissza.

Az Immerman-Szelepcsényi tétel

Egy nd algoritmust hívó algoritmust hívó algoritmus

- Input: gráf, $G[1 \dots N][1 \dots N]$ szomszédsági mátrixával és s csúcsa.

```
prev := 1;  
for  $k := 1 \dots N$  do  
    current := 0;  
    for  $t := 1 \dots N$  do  
        if T-IN-SK( $s, t, prev, k$ ) then  
            current := current + 1;  
    prev := current;  
return prev;
```

- Az algoritmus visszaadja az s -ből elérhető csúcsok számát, vagy ERROR-t dob; van, mikor a számot adja vissza.
- Mindezt logaritmikus tárigényben.

Az Immerman-Szelepcsényi tétel

Beláttuk az Immerman-Szelepcsényi tételt:

Az Immerman-Szelepcsényi tétel

Beláttuk az Immerman-Szelepcsényi tételt:

Tétel

Van olyan **nemdeterminisztikus** algoritmus, mely **logaritmikus tárban** kiszámítja az input gráf megadott csúcsából elérhető csúcsok számát.

Az Immerman-Szelepcsényi tétel

Beláttuk az Immerman-Szelepcsényi tételt:

Tétel

Van olyan **nemdeterminisztikus** algoritmus, mely **logaritmikus tárban** kiszámítja az input gráf megadott csúcsából elérhető csúcsok számát.

Nemdeterminisztikus függvénykiszámítás: minden szálon vagy a helyes eredményt adja vissza, vagy REJECT-et;

Az Immerman-Szelepcsényi tétel

Beláttuk az Immerman-Szelepcsényi tételt:

Tétel

Van olyan **nemdeterminisztikus** algoritmus, mely **logaritmikus tárban** kiszámítja az input gráf megadott csúcsából elérhető csúcsok számát.

Nemdeterminisztikus függvénykiszámítás: minden szálon vagy a helyes eredményt adja vissza, vagy REJECT-et; továbbá van olyan szál, mikor a helyes eredményt.

Az Immerman-Szelepcsényi tétel

Következmény

$$\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)),$$

ha $f(n) \geq \log n$.

Az Immerman-Szelepcsényi tétel

Következmény

$$\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)),$$

ha $f(n) \geq \log n$.

Ötlet

A konfigurációs gráfban először nd kiszámítjuk az elérhető konfigurációk számát, aztán nd mindet megpróbáljuk elérni.

Az Immerman-Szelepcsényi tétel

Következmény

$$\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)),$$

ha $f(n) \geq \log n$.

Ötlet

A konfigurációs gráfban először nd kiszámítjuk az elérhető konfigurációk számát, aztán nd mindet megpróbáljuk elérni. Ha annyit értünk el, amennyi csak elérhető és egyik sem elfogadó, akkor ACCEPT, különben REJECT.

Az Immerman-Szelepcsényi tétel

Következmény

$$\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)),$$

ha $f(n) \geq \log n$.

Ötlet

A konfigurációs gráfban először nd kiszámítjuk az elérhető konfigurációk számát, aztán nd mindet megpróbáljuk elérni. Ha annyit értünk el, amennyi csak elérhető és egyik sem elfogadó, akkor ACCEPT, különben REJECT.

Következmény

$$\text{NL} = \text{coNL}.$$

A logaritmikusan társított visszavezetés

A polinomidejű visszavezetésre nézve $\mathbf{P}$ minden nemtriviális eleme „ $\mathbf{P}$ -teljes”.

A logaritmikus tárban való visszavezetés

A polinomidejű visszavezetésre nézve $\mathbf{P}$ minden nemtriviális eleme „ $\mathbf{P}$ -teljes”.

Bevezetünk egy (formailag) „gyengébb” visszavezetést, melyet a $\mathbf{P}$ osztályon belül alkalmazunk problémák egymáshoz viszonyított nehézségének definiálására.

A logaritmikus tárban való visszavezetés

A polinomidejű visszavezetésre nézve $\mathbf{P}$ minden nemtriviális eleme „ $\mathbf{P}$ -teljes”. Bevezetünk egy (formailag) „gyengébb” visszavezetést, melyet a $\mathbf{P}$ osztályon belül alkalmazunk problémák egymáshoz viszonyított nehézségének definiálására.

Az f függvény az A eldöntési problémának a B eldöntési problémára való **logaritmikus tárigényű** visszavezetése, ha

A logaritmikus tárban való visszavezetés

A polinomidejű visszavezetésre nézve $\mathbf{P}$ minden nemtriviális eleme „ $\mathbf{P}$ -teljes”.

Bevezetünk egy (formailag) „gyengébb” visszavezetést, melyet a $\mathbf{P}$ osztályon belül alkalmazunk problémák egymáshoz viszonyított nehézségének definiálására.

Az f függvény az A eldöntési problémának a B eldöntési problémára való **logaritmikus tárigényű** visszavezetése, ha

- f kiszámítható logaritmikus tárban és

A logaritmiкус tárban való visszavezetés

A polinomidejű visszavezetésre nézve $\mathbf{P}$ minden nemtriviális eleme „ $\mathbf{P}$ -teljes”.

Bevezetünk egy (formailag) „gyengébb” visszavezetést, melyet a $\mathbf{P}$ osztályon belül alkalmazunk problémák egymáshoz viszonyított nehézségének definiálására.

Az f függvény az A eldöntési problémának a B eldöntési problémára való **logaritmiкус tárigényű** visszavezetése, ha

- f kiszámítható logaritmiкус tárban és
- tetszőleges I inputra $A(I) = B(f(I))$.

A logaritmikusan való visszavezetés

A polinomidejű visszavezetésre nézve $\mathbf{P}$ minden nemtriviális eleme „ $\mathbf{P}$ -teljes”. Bevezetünk egy (formailag) „gyengébb” visszavezetést, melyet a $\mathbf{P}$ osztályon belül alkalmazunk problémák egymáshoz viszonyított nehézségének definiálására.

Az f függvény az A eldöntési problémának a B eldöntési problémára való **logaritmikusan tárgényű** visszavezetése, ha

- f kiszámítható logaritmikusan tárban és
- tetszőleges I inputra $A(I) = B(f(I))$.

Ha létezik A -nak B -re való logaritmikusan tárgényű visszavezetése, annak jele **$A \leq_{\mathcal{L}} B$** : A logtárban visszavezethető B -re.

A logtáras visszavezetés

Logaritmikus tárigényű (tehát mindenképp offline) algoritmusnak legfeljebb $2^{\mathcal{O}(\log n)}$, azaz polinom sok konfigurációja lehet adott input esetén;

A logtáras visszavezetés

Logaritmikus tárigényű (tehát mindenképp offline) algoritmusnak legfeljebb $2^{\mathcal{O}(\log n)}$, azaz polinom sok konfigurációja lehet adott input esetén; mivel a visszavezetés nem eshet végtelen ciklusba, így polinomidőben meg is kell álljon.

A logtáras visszavezetés

Logaritmikus tárigényű (tehát mindenképp offline) algoritmusnak legfeljebb $2^{\mathcal{O}(\log n)}$, azaz polinom sok konfigurációja lehet adott input esetén; mivel a visszavezetés nem eshet végtelen ciklusba, így polinomidőben meg is kell álljon.

Vagyis ha $A \leq_{\mathcal{L}} B$, akkor $A \leq_{\mathcal{P}} B$ is.

A logtáras visszavezetés

Logaritmikus tárigényű (tehát mindenképp offline) algoritmusnak legfeljebb $2^{\mathcal{O}(\log n)}$, azaz polinom sok konfigurációja lehet adott input esetén; mivel a visszavezetés nem eshet végtelen ciklusba, így polinomidőben meg is kell álljon.

Vagyis ha $A \leq_{\mathcal{L}} B$, akkor $A \leq_{\mathcal{P}} B$ is.

Megjegyzés

Nem ismert, hogy $\leq_{\mathcal{L}}$ és $\leq_{\mathcal{P}}$ egybeesnek-e...

A logtáras visszavezetés

Logaritmikus tárigényű (tehát mindenképp offline) algoritmusnak legfeljebb $2^{\mathcal{O}(\log n)}$, azaz polinom sok konfigurációja lehet adott input esetén; mivel a visszavezetés nem eshet végtelen ciklusba, így polinomidőben meg is kell álljon.

Vagyis ha $A \leq_{\mathcal{L}} B$, akkor $A \leq_{\mathcal{P}} B$ is.

Megjegyzés

Nem ismert, hogy $\leq_{\mathcal{L}}$ és $\leq_{\mathcal{P}}$ egybeesnek-e... (ha igen, akkor $\mathbf{L} = \mathbf{P}$)

Legyen $\mathcal{C}$ problémák egy osztálya. Az A probléma **$\mathcal{C}$ -nehéz** a logtáras visszavezetésre nézve, ha minden $\mathcal{C}$ -beli probléma logtárban visszavezethető A -ra.

Legyen $\mathcal{C}$ problémák egy osztálya. Az A probléma **$\mathcal{C}$ -nehéz** a logtáras visszavezetésre nézve, ha minden $\mathcal{C}$ -beli probléma logtárban visszavezethető A -ra. Ha még $A \in \mathcal{C}$ is, akkor A egy $\mathcal{C}$ -teljes probléma a logtáras visszavezetésre nézve.

Legyen $\mathcal{C}$ problémák egy osztálya. Az A probléma **$\mathcal{C}$ -nehéz** a logtáras visszavezetésre nézve, ha minden $\mathcal{C}$ -beli probléma logtárban visszavezethető A -ra. Ha még $A \in \mathcal{C}$ is, akkor A egy $\mathcal{C}$ -teljes probléma a logtáras visszavezetésre nézve.

Megjegyzés

Nem ismert, hogy a logtáras visszavezetésre nézve **NP**-teljes problémák ugyanazok-e, mint a polinomidejű visszavezetésre nézve **NP**-teljesek. (A SAT egy **NP**-teljes probléma a logtáras visszavezetésre nézve is; az összes hatékony visszavezetés, melyet eddig láttunk, egyben logtáras is volt.)

A logtáras visszavezetés tranzitivitása

Tétel

A logtárban való visszavezethetőség tranzitív: ha f az A -nak B -re, g pedig a B -nek C -re való logtáras visszavezetése, akkor az $f \circ g$ összetett függvény az A -nak C -re való logtáras visszavezetése.

A gond

Legyen M_f az f -et, M_g pedig a g -t kiszámító program.

A logtáras visszavezetés tranzitivitása

Tétel

A logtárban való visszavezethetőség tranzitív: ha f az A -nak B -re, g pedig a B -nek C -re való logtáras visszavezetése, akkor az $f \circ g$ összetett függvény az A -nak C -re való logtáras visszavezetése.

A gond

Legyen M_f az f -et, M_g pedig a g -t kiszámító program.

Az nyilván igaz, hogy $I \in A \Leftrightarrow f(I) \in B \Leftrightarrow g(f(I)) \in C$.

A logtáras visszavezetés tranzitivitása

Tétel

A logtárban való visszavezethetőség tranzitív: ha f az A -nak B -re, g pedig a B -nek C -re való logtáras visszavezetése, akkor az $f \circ g$ összetett függvény az A -nak C -re való logtáras visszavezetése.

A gond

Legyen M_f az f -et, M_g pedig a g -t kiszámító program.

Az nyilván igaz, hogy $I \in A \Leftrightarrow f(I) \in B \Leftrightarrow g(f(I)) \in C$.

Csak hogy ha $M_f(I)$ -t munkaregiszterekbe írjuk, sokkal hosszabb lehet, mint $\log(|I|)$!

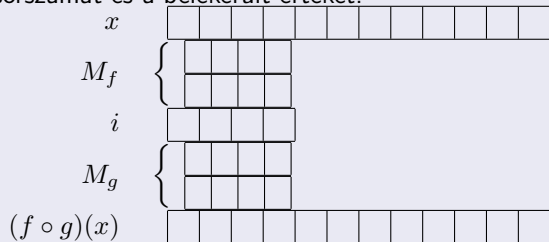
A trükk

Nem tároljuk el M_f teljes outputját, csak a legutolsóként írt output regiszter sorszámát és a belekerült értéket.

A visszavezetés tranzitivitása

A trükk

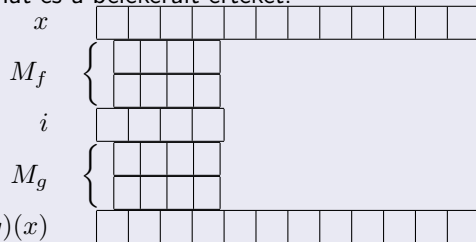
Nem tároljuk el M_f teljes outputját, csak a legutolsóként írt output regiszter sorszámát és a belekerült értéket.



A visszavezetés tranzitivitása

A trükk

Nem tároljuk el M_f teljes outputját, csak a legutolsóként írt output regiszter sorszámát és a belekerült értéket.

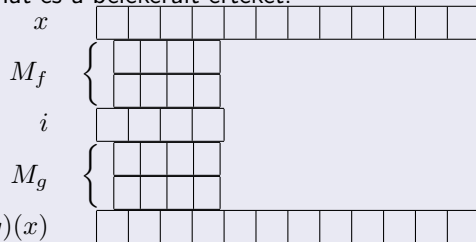


M_g -nek szüksége van egy nagyobb sorszámú regiszter tartalmára: folytatjuk M_f szimulálását.

A visszavezetés tranzitivitása

A trükk

Nem tároljuk el M_f teljes outputját, csak a legutolsóként írt output regiszter sorszámát és a belekerült értéket.



M_g -nek szüksége van egy nagyobb sorszámú regiszter tartalmára: folytatjuk M_f szimulálását.

M_g -nek szüksége van egy kisebb sorszámú regiszter tartalmára: előlről kezdjük M_f szimulálását.

P-teljesség, NL-teljesség

Korábban láttuk, hogy **P** (így **NL**) bármely két nemtriviális problémája hatékonyan visszavezethető egymásra, így a polinomidejű visszavezetésre nézve ezen osztályokon belül a „teljesség” fogalma értelmetlenné válik.

P-teljesség, NL-teljesség

Korábban láttuk, hogy **P** (így **NL**) bármely két nemtriviális problémája hatékonyan visszavezethető egymásra, így a polinomidejű visszavezetésre nézve ezen osztályokon belül a „teljesség” fogalma értelmetlenné válik.

Azt mondjuk, hogy egy probléma **NL/P**-teljes/nehéz, ha a **logtáras** visszavezetésre nézve az.

P-teljesség, NL-teljesség

Korábban láttuk, hogy **P** (így **NL**) bármely két nemtriviális problémája hatékonyan visszavezethető egymásra, így a polinomidejű visszavezetésre nézve ezen osztályokon belül a „teljesség” fogalma értelmetlenné válik.

Azt mondjuk, hogy egy probléma **NL/P**-teljes/nehéz, ha a **logtáras** visszavezetésre nézve az.

Tétel

A HÁLÓZAT-KIÉRTÉKELES probléma **P**-teljes.

P-teljesség, NL-teljesség

Korábban láttuk, hogy **P** (így **NL**) bármely két nemtriviális problémája hatékonyan visszavezethető egymásra, így a polinomidejű visszavezetésre nézve ezen osztályokon belül a „teljesség” fogalma értelmetlenné válik.

Azt mondjuk, hogy egy probléma **NL/P**-teljes/nehéz, ha a **logtáras** visszavezetésre nézve az.

Tétel

A HÁLÓZAT-KIÉRTÉKELES probléma **P**-teljes.

Az ELÉRHETŐSÉG probléma NL-teljessége

Tétel

Az ELÉRHETŐSÉG probléma NL-teljes.

Az ELÉRHETŐSÉG probléma NL-teljessége

Tétel

Az ELÉRHETŐSÉG probléma **NL**-teljes.

Ötlet

Azt már láttuk, hogy $\text{ELÉRHETŐSÉG} \in \text{NL}$.

Az **NL**-nehézséghez mint korábban, az **elérhetőségi módszert** alkalmazzuk: egy nemdeterminisztikus, $\mathcal{O}(\log n)$ tárkorlátos gép konfigurációs grájában megoldva az elérhetőségi problémát a kezdőkonfigurációtól az elfogadóig, készen vagyunk.

Következmény

Az ELÉRHETETLENSÉG probléma **NL**-teljes.

Az ELÉRHETŐSÉG probléma NL-teljesége

Tétel

Az ELÉRHETŐSÉG probléma **NL**-teljes.

Ötlet

Azt már láttuk, hogy $\text{ELÉRHETŐSÉG} \in \text{NL}$.

Az **NL**-nehézséghez mint korábban, az **elérhetőségi módszert** alkalmazzuk: egy nemdeterminisztikus, $\mathcal{O}(\log n)$ tárkorlátos gép konfigurációs grájában megoldva az elérhetőségi problémát a kezdőkonfigurációtól az elfogadóig, készen vagyunk.

Következmény

Az ELÉRHETETLENSÉG probléma **NL**-teljes.

Hiszen az Immerman-Szelepcsényi tétel szerint $\text{NL} = \text{coNL}$.

Tétel

A 2SAT probléma **NL**-teljes.

Tétel

A 2SAT probléma **NL**-teljes.

Ötlet

Azt már tudjuk, hogy $2\text{SAT} \in \mathbf{NL}$.

Tétel

A 2SAT probléma **NL**-teljes.

Ötlet

Azt már tudjuk, hogy $2SAT \in \mathbf{NL}$. Tekintsük az ELÉRHETETLENSÉG egy példányát.

Tétel

A 2SAT probléma **NL**-teljes.

Ötlet

Azt már tudjuk, hogy $2SAT \in \mathbf{NL}$. Tekintsük az ELÉRHETETLENSÉG egy példányát. Minden csúcshoz rendeljük egy x változót.

Tétel

A 2SAT probléma **NL**-teljes.

Ötlet

Azt már tudjuk, hogy $2\text{SAT} \in \mathbf{NL}$. Tekintsük az ELÉRHETETLENSÉG egy példányát. Minden csúcshoz rendeljünk egy x változót. A 2SAT azon φ példányát készítjük el, mely az összes $\neg x \vee y$ tagokból áll, ahol (x, y) él, továbbá az s és $\neg t$ tagokból, ahol s a kezdőcsúcs és t a cél.

Tétel

A 2SAT probléma **NL**-teljes.

Ötlet

Azt már tudjuk, hogy $2SAT \in \mathbf{NL}$. Tekintsük az ELÉRHETETLENSÉG egy példányát. Minden csúcshoz rendeljünk egy x változót. A 2SAT azon φ példányát készítjük el, mely az összes $\neg x \vee y$ tagokból áll, ahol (x, y) él, továbbá az s és $\neg t$ tagokból, ahol s a kezdőcsúcs és t a cél.

Így φ kielégíthető $\Leftrightarrow s$ -ből nem érhető el t .

Tétel

A 2SAT probléma **NL**-teljes.

Ötlet

Azt már tudjuk, hogy $2SAT \in \mathbf{NL}$. Tekintsük az ELÉRHETETLENSÉG egy példányát. Minden csúcshoz rendeljünk egy x változót. A 2SAT azon φ példányát készítjük el, mely az összes $\neg x \vee y$ tagokból áll, ahol (x, y) él, továbbá az s és $\neg t$ tagokból, ahol s a kezdőcsúcs és t a cél.

Így φ kielégíthető $\Leftrightarrow s$ -ből nem érhető el t .

Emiatt a 2SAT komplementere is **NL**-teljes.

QSAT

QSAT

Adott: $\exists x_1 \forall x_2 \dots Q_m x_m \varphi$ kvantifikált Boole-formula prenex normálalakban úgy, hogy a φ konjunktív normálalakú kvantormentes formula, melyben legfeljebb az $x_1, \dots, x_m$ változók fordulnak elő.

QSAT

Adott: $\exists x_1 \forall x_2 \dots Q_m x_m \varphi$ kvantifikált Boole-formula prenex normálalakban úgy, hogy a φ konjunktív normálalakú kvantormentes formula, melyben legfeljebb az $x_1, \dots, x_m$ változók fordulnak elő.

Kérdés: Igaz-e az adott formula?

A polinomiális tár

QSAT

Adott: $\exists x_1 \forall x_2 \dots Q_m x_m \varphi$ kvantifikált Boole-formula prenex normálalakban úgy, hogy a φ konjunktív normálalakú kvantormentes formula, melyben legfeljebb az $x_1, \dots, x_m$ változók fordulnak elő.

Kérdés: Igaz-e az adott formula?

Megjegyzés

QSAT

Adott: $\exists x_1 \forall x_2 \dots Q_m x_m \varphi$ kvantifikált Boole-formula prenex normálalakban úgy, hogy a φ konjunktív normálalakú kvantormentes formula, melyben legfeljebb az $x_1, \dots, x_m$ változók fordulnak elő.

Kérdés: Igaz-e az adott formula?

Megjegyzés

- A kvantorok szigorú alternálása nem lényeges.

QSAT

Adott: $\exists x_1 \forall x_2 \dots Q_m x_m \varphi$ kvantifikált Boole-formula prenex normálalakban úgy, hogy a φ konjunktív normálalakú kvantormentes formula, melyben legfeljebb az $x_1, \dots, x_m$ változók fordulnak elő.

Kérdés: Igaz-e az adott formula?

Megjegyzés

- A kvantorok szigorú alternálása nem lényeges.
- Az sem lényeges, hogy az első kvantor $\exists$.

Példa

$$\exists x \forall y ((x \vee y) \wedge (\neg x \vee \neg y))$$

Példa

$$\exists x \forall y ((x \vee y) \wedge (\neg x \vee \neg y))$$

hamis,

Példa

$$\exists x \forall y ((x \vee y) \wedge (\neg x \vee \neg y))$$

hamis, hiszen $x = 0$ esetén $y = 0$, $x = 1$ esetén pedig $y = 1$ választása mellett hamis lesz a formula magja.

Példa

$$\exists x \forall y ((x \vee y) \wedge (\neg x \vee \neg y))$$

hamis, hiszen $x = 0$ esetén $y = 0$, $x = 1$ esetén pedig $y = 1$ választása mellett hamis lesz a formula magja.

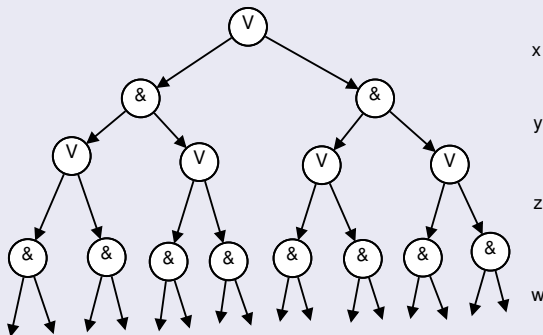
Példa

$$\exists x \forall y \exists z \forall w ((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z))$$

?

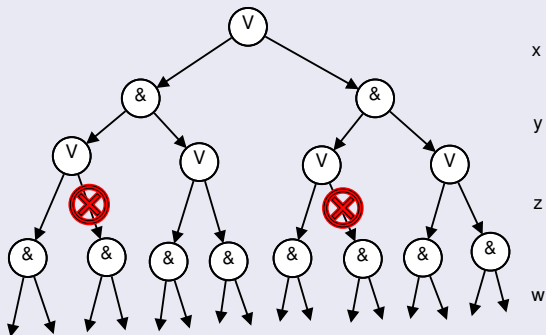
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



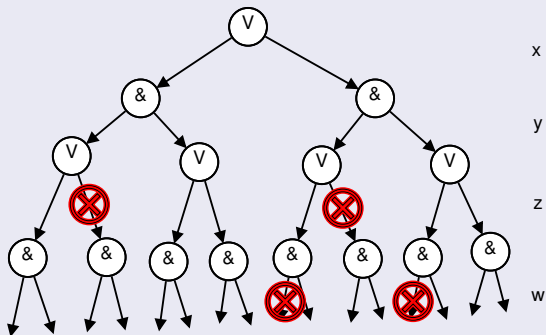
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



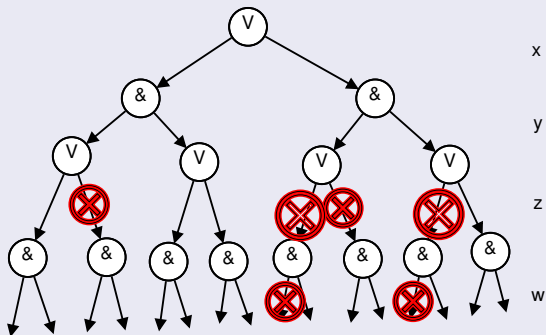
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



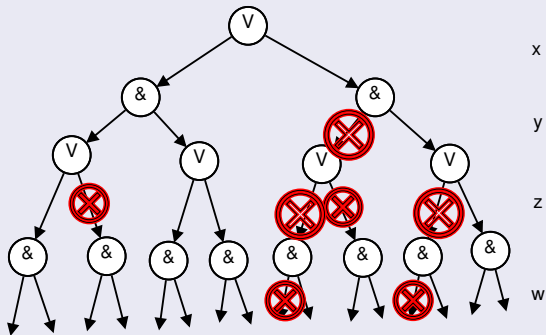
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



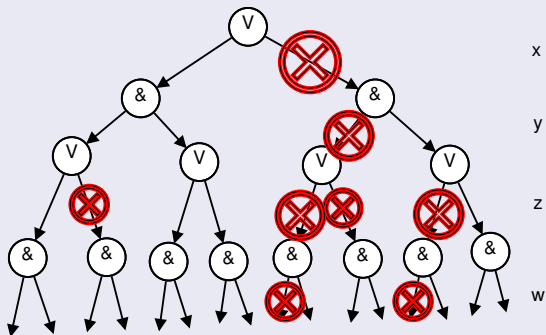
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



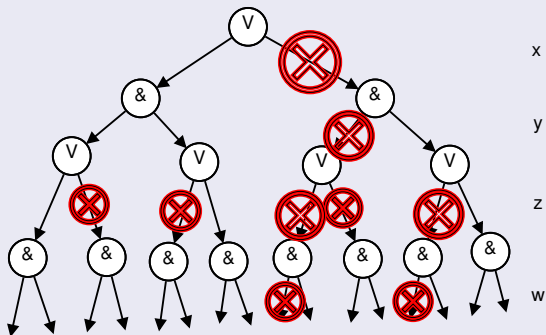
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



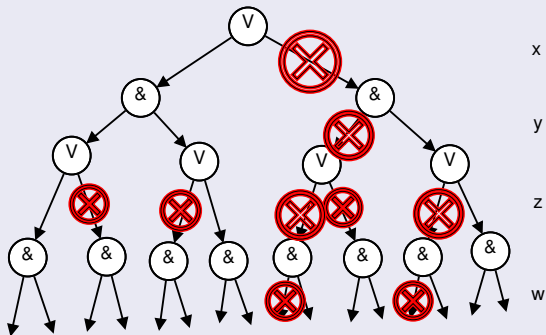
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



A hosszabb példa

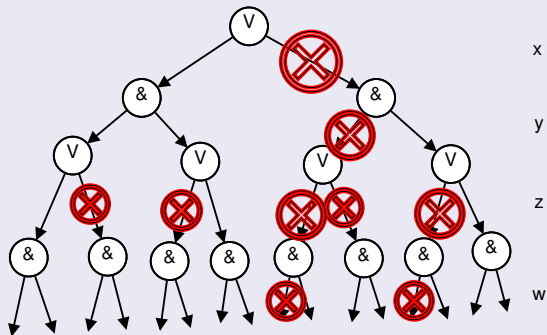
$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



A formula **igaz**: $x = 0$ választása esetén

A hosszabb példa

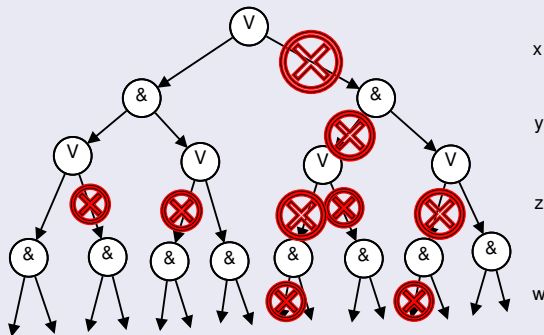
$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



A formula **igaz**: $x = 0$ választása esetén tetszőleges y -ra

A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



A formula **igaz**: $x = 0$ választása esetén tetszőleges y -ra legyen $z = y$, ekkor a formula minden w -re igaz lesz.

Állítás

A QSAT probléma **PSPACE**-ben van.

Állítás

A QSAT probléma **PSPACE**-ben van.

Algoritmus

Állítás

A QSAT probléma **PSPACE**-ben van.

Algoritmus

Bemenet: $Q_1x_1 \dots Q_mx_m\varphi$ alakú zárt formula.

Állítás

A QSAT probléma **PSPACE**-ben van.

Algoritmus

Bemenet: $Q_1x_1 \dots Q_mx_m\varphi$ alakú zárt formula.

Kimenet: a formula értéke.

Állítás

A QSAT probléma **PSPACE**-ben van.

Algoritmus

Bemenet: $Q_1x_1 \dots Q_mx_m\varphi$ alakú zárt formula.

Kimenet: a formula értéke.

```
function QSAT( $F$ )  
  if  $F == \exists x F'$  then  
    return QSAT( $F'[x/0]$ ) or QSAT( $F'[x/1]$ )  
  else if  $F == \forall x F'$  then  
    return QSAT( $F'[x/0]$ ) and QSAT( $F'[x/1]$ )  
  else  
    return EVAL( $F$ )
```

Állítás

A QSAT probléma **PSPACE**-ben van.

Algoritmus

Bemenet: $Q_1x_1 \dots Q_mx_m\varphi$ alakú zárt formula.

Kimenet: a formula értéke.

```
function QSAT( $F$ )  
  if  $F == \exists x F'$  then  
    return QSAT( $F'[x/0]$ ) or QSAT( $F'[x/1]$ )  
  else if  $F == \forall x F'$  then  
    return QSAT( $F'[x/0]$ ) and QSAT( $F'[x/1]$ )  
  else  
    return EVAL( $F$ )
```

Ez az algoritmus $\mathcal{O}(n^2)$ táiban kiszámítja a formula értékét (ahol EVAL a változómentes formulát polinom időben és lineáris táiban kiértékelő függvény).

Megjegyzés

A probléma akkor is **PSPACE**-ben van, ha

Megjegyzés

A probléma akkor is **PSPACE**-ben van, ha

- φ tetszőleges kvantormentes Boole-formula.

Megjegyzés

A probléma akkor is **PSPACE**-ben van, ha

- φ tetszőleges kvantormentes Boole-formula.
- a kvantorok nem feltétlenül váltakoznak.

Megjegyzés

A probléma akkor is **PSPACE**-ben van, ha

- φ tetszőleges kvantormentes Boole-formula.
- a kvantorok nem feltétlenül váltakoznak.
- φ -ben az $x_1, \dots, x_m$ változókon kívül egyéb változók is előfordulnak, és azt kérdezzük, kielégíthető-e az egész formula.

Megjegyzés

A probléma akkor is **PSPACE**-ben van, ha

- φ tetszőleges kvantormentes Boole-formula.
- a kvantorok nem feltétlenül váltakoznak.
- φ -ben az $x_1, \dots, x_m$ változókon kívül egyéb változók is előfordulnak, és azt kérdezzük, kielégíthető-e az egész formula.
- a formula nem feltétlenül prenex alakú.

Megjegyzés

A probléma akkor is **PSPACE**-ben van, ha

- φ tetszőleges kvantormentes Boole-formula.
- a kvantorok nem feltétlenül váltakoznak.
- φ -ben az $x_1, \dots, x_m$ változókon kívül egyéb változók is előfordulnak, és azt kérdezzük, kielégíthető-e az egész formula.
- a formula nem feltétlenül prenex alakú.
- azt kérdezzük, hogy a formula hamis-e.

Tétel

A QSAT probléma **PSPACE**-teljes.

Tétel

A QSAT probléma **PSPACE**-teljes.

Ötlet

Tétel

A QSAT probléma **PSPACE**-teljes.

Ötlet

- Egy tetszőleges n^k tárkorlátos M programhoz és x inputjához konstruálunk egy φ formulát, mely pontosan akkor igaz, ha M elfogadja x -et.

Tétel

A QSAT probléma **PSPACE**-teljes.

Ötlet

- Egy tetszőleges n^k tárkorlátos M programhoz és x inputjához konstruálunk egy φ formulát, mely pontosan akkor igaz, ha M elfogadja x -et.
- Egy konfigurációt n^k darab bináris változóval kódolunk.

Tétel

A QSAT probléma **PSPACE**-teljes.

Ötlet

- Egy tetszőleges n^k tárkorlátos M programhoz és x inputjához konstruálunk egy φ formulát, mely pontosan akkor igaz, ha M elfogadja x -et.
- Egy konfigurációt n^k darab bináris változóval kódolunk.
- Tetszőleges t -re a φ_t formulát, aminek szabad változói $x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k}$, úgy konstruáljuk meg, hogy $\varphi_t(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$ pontosan akkor igaz, ha az $(x_1, \dots, x_{n^k})$ konfigurációból az M program **2^t lépésen belül** az $(y_1, \dots, y_{n^k})$ konfigurációba jut;

Tétel

A QSAT probléma **PSPACE**-teljes.

Ötlet

- Egy tetszőleges n^k tárkorlátos M programhoz és x inputjához konstruálunk egy φ formulát, mely pontosan akkor igaz, ha M elfogadja x -et.
- Egy konfigurációt n^k darab bináris változóval kódolunk.
- Tetszőleges t -re a φ_t formulát, aminek szabad változói $x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k}$, úgy konstruáljuk meg, hogy $\varphi_t(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$ pontosan akkor igaz, ha az $(x_1, \dots, x_{n^k})$ konfigurációból az M program **2^t lépésen belül** az $(y_1, \dots, y_{n^k})$ konfigurációba jut; továbbá φ_t „nem túl hosszú”.

Tétel

A QSAT probléma **PSPACE**-teljes.

Ötlet

- Egy tetszőleges n^k tárkorlátos M programhoz és x inputjához konstruálunk egy φ formulát, mely pontosan akkor igaz, ha M elfogadja x -et.
- Egy konfigurációt n^k darab bináris változóval kódolunk.
- Tetszőleges t -re a φ_t formulát, aminek szabad változói $x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k}$, úgy konstruáljuk meg, hogy $\varphi_t(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$ pontosan akkor igaz, ha az $(x_1, \dots, x_{n^k})$ konfigurációból az M program **2^t lépésen belül** az $(y_1, \dots, y_{n^k})$ konfigurációba jut; továbbá φ_t „nem túl hosszú”.
- A keresett formula: φ_{n^k} , behelyettesítve a kezdő- és az elfogadó konfigurációt.

$$\varphi_{t+1}(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$$

Jelölje X az $x_1, \dots, x_{n^k}$ változó-vektort, Y az $y_1, \dots, y_{n^k}$ -t stb.

$$\varphi_{t+1}(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$$

Jelölje X az $x_1, \dots, x_{n^k}$ változó-vektort, Y az $y_1, \dots, y_{n^k}$ -t stb.

A Savitch-tételben látottal analóg módon:

$$\exists Z(\varphi_t(X, Z) \wedge \varphi_t(Z, Y)),$$

$$\varphi_{t+1}(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$$

Jelölje X az $x_1, \dots, x_{n^k}$ változó-vektort, Y az $y_1, \dots, y_{n^k}$ -t stb.

A Savitch-tételben látottal analóg módon:

$$\exists Z(\varphi_t(X, Z) \wedge \varphi_t(Z, Y)),$$

de ez túl hosszú! (Exponenciális hosszú formula.)

$$\varphi_{t+1}(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$$

Jelölje X az $x_1, \dots, x_{n^k}$ változó-vektort, Y az $y_1, \dots, y_{n^k}$ -t stb.

A Savitch-tételben látottal analóg módon:

$$\exists Z(\varphi_t(X, Z) \wedge \varphi_t(Z, Y)),$$

de ez túl hosszú! (Exponenciális hosszú formula.)

Ehelyett:

$$\exists Z \forall X', Y' \left(((X' = X \wedge Y' = Z) \vee (X' = Z \wedge Y' = Y)) \rightarrow \varphi_t(X', Y') \right),$$

ahol $A = B$ az $(a_1 \leftrightarrow b_1) \wedge \dots \wedge (a_{n^k} \leftrightarrow b_{n^k})$ formula rövidítése.

$$\varphi_{t+1}(x_1, \dots, x_{n^k}, y_1, \dots, y_{n^k})$$

Jelölje X az $x_1, \dots, x_{n^k}$ változó-vektort, Y az $y_1, \dots, y_{n^k}$ -t stb.

A Savitch-tételben látottal analóg módon:

$$\exists Z(\varphi_t(X, Z) \wedge \varphi_t(Z, Y)),$$

de ez túl hosszú! (Exponenciális hosszú formula.)

Ehelyett:

$$\exists Z \forall X', Y' \left(((X' = X \wedge Y' = Z) \vee (X' = Z \wedge Y' = Y)) \rightarrow \varphi_t(X', Y') \right),$$

ahol $A = B$ az $(a_1 \leftrightarrow b_1) \wedge \dots \wedge (a_{n^k} \leftrightarrow b_{n^k})$ formula rövidítése.

A formula magja (polinom méretű) CNF-re is könnyen hozható.

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárbán?

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

Kérdés: Ugyanazt a nyelvet jelölik-e?

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

Kérdés: Ugyanazt a nyelvet jelölik-e?

VÉGES AUTOMATÁK EKVIVALENCIÁJA

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

Kérdés: Ugyanazt a nyelvet jelölik-e?

VÉGES AUTOMATÁK EKVIVALENCIÁJA

Adott: M_1 és M_2 véges **nemdeterminisztikus** automaták.

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

Kérdés: Ugyanazt a nyelvet jelölik-e?

VÉGES AUTOMATÁK EKVIVALENCIÁJA

Adott: M_1 és M_2 véges **nemdeterminisztikus** automaták.

Kérdés: M_1 és M_2 ekvivalensek-e?

További PSPACE-teljes problémák

HELYBEN ELFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

Kérdés: Ugyanazt a nyelvet jelölik-e?

VÉGES AUTOMATÁK EKVIVALENCIÁJA

Adott: M_1 és M_2 véges **nemdeterminisztikus** automaták.

Kérdés: M_1 és M_2 ekvivalensek-e?

Tétel

A fenti három probléma mind **PSPACE**-teljes.

Készülődés a **PSPACE** egy meglepő karakterizációjára

Alternálás

Visszatérünk még a **PSPACE**-re, de előbb bevezetünk egy fogalmat, az **alternálást**.

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Azt mondjuk, hogy egy C konfiguráció **végül elfogadó**, ha

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Azt mondjuk, hogy egy C konfiguráció **végül elfogadó**, ha

- C -ben a gép ACCEPT soron áll, vagy

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Azt mondjuk, hogy egy C konfiguráció **végül elfogadó**, ha

- C -ben a gép ACCEPT soron áll, vagy
- ÉS-sorszámon van, és **minden** közvetlen leszármazottja végül elfogadó, vagy

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Azt mondjuk, hogy egy C konfiguráció **végül elfogadó**, ha

- C -ben a gép ACCEPT soron áll, vagy
- ÉS-sorszámon van, és **minden** közvetlen leszármazottja végül elfogadó, vagy
- VAGY-sorszámon van, és **valamely** közvetlen leszármazottja végül elfogadó.

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Azt mondjuk, hogy egy C konfiguráció **végül elfogadó**, ha

- C -ben a gép ACCEPT soron áll, vagy
- ÉS-sorszámon van, és **minden** közvetlen leszármazottja végül elfogadó, vagy
- VAGY-sorszámon van, és **valamely** közvetlen leszármazottja végül elfogadó.

A program akkor **fogadja el** az x bemenetet, ha a kezdő konfiguráció végül elfogadó, különben elutasítja azt.

Definíció

Az **alternáló program** olyan program, melyben (a nemdeterminisztikushoz hasonlóan) egy sorszámot több sornak is kioszthatunk, **továbbá** minden sorszám vagy ÉS, vagy VAGY típusú sorszám.

Tekintsük a program számítási fáját az x bemeneten.

Azt mondjuk, hogy egy C konfiguráció **végül elfogadó**, ha

- C -ben a gép ACCEPT soron áll, vagy
- ÉS-sorszámon van, és **minden** közvetlen leszármazottja végül elfogadó, vagy
- VAGY-sorszámon van, és **valamely** közvetlen leszármazottja végül elfogadó.

A program akkor **fogadja el** az x bemenetet, ha a kezdő konfiguráció végül elfogadó, különben elutasítja azt. Az $f(n)$ idő- vagy tárkorlátos alternáló program fogalmát értelemszerűen definiáljuk.

Alternáló program

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

ATIME($f(n)$): az összes $f(n)$ időkorlátos alternáló programmal eldönthető nyelvek.

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

ATIME($f(n)$): az összes $f(n)$ időkorlátos alternáló programmal eldönthető nyelvek.

ASPACE($f(n)$): az összes $f(n)$ tárkorlátos alternáló programmal eldönthető nyelvek.

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

ATIME($f(n)$): az összes $f(n)$ időkorlátos alternáló programmal eldönthető nyelvek.

ASPACE($f(n)$): az összes $f(n)$ tárkorlátos alternáló programmal eldönthető nyelvek.

AL = **ASPACE**($\log n$)

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

ATIME($f(n)$): az összes $f(n)$ időkorlátos alternáló programmal eldönthető nyelvek.

ASPACE($f(n)$): az összes $f(n)$ tárkorlátos alternáló programmal eldönthető nyelvek.

AL = **ASPACE**($\log n$)

AP = **ATIME**(n^k)

Alternáló program

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

ATIME($f(n)$): az összes $f(n)$ időkorlátos alternáló programmal eldönthető nyelvek.

ASPACE($f(n)$): az összes $f(n)$ tárkorlátos alternáló programmal eldönthető nyelvek.

AL = **ASPACE**($\log n$)

AP = **ATIME**(n^k)

Világos, hogy $\text{NTIME}(f(n)) \subseteq \text{ATIME}(f(n))$ és
 $\text{NSPACE}(f(n)) \subseteq \text{ASPACE}(f(n))$.

Alternáló program

Definíció

Legyen $f(n)$ megengedett bonyolultsági függvény. (Idő esetén $f(n) > n$.)

ATIME($f(n)$): az összes $f(n)$ időkorlátos alternáló programmal eldönthető nyelvek.

ASPACE($f(n)$): az összes $f(n)$ tárkorlátos alternáló programmal eldönthető nyelvek.

AL = **ASPACE**($\log n$)

AP = **ATIME**(n^k)

Világos, hogy $\text{NTIME}(f(n)) \subseteq \text{ATIME}(f(n))$ és
 $\text{NSPACE}(f(n)) \subseteq \text{ASPACE}(f(n))$.

Hiszen a nemdeterminisztikus program olyan alternáló program, melynek csak VAGY típusú sorai vannak.

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Tétel

$$AP = PSPACE.$$

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Tétel

$$AP = PSPACE.$$

Az utóbbi tételt megfogalmazhatjuk így is:

PSPACE-be pontosan azok a problémák tartoznak, melyek reprezentálhatók

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Tétel

$$AP = PSPACE.$$

Az utóbbi tételt megfogalmazhatjuk így is:

PSPACE-be pontosan azok a problémák tartoznak, melyek reprezentálhatók zéró összegű,

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Tétel

$$AP = PSPACE.$$

Az utóbbi tételt megfogalmazhatjuk így is:

PSPACE-be pontosan azok a problémák tartoznak, melyek reprezentálhatók
zéró összegű, teljes információs,

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Tétel

$$AP = PSPACE.$$

Az utóbbi tételt megfogalmazhatjuk így is:

PSPACE-be pontosan azok a problémák tartoznak, melyek reprezentálhatók
zéró összegű, teljes információs, polinom lépésszámú

Alternálás és determinizmus

Az alternáló bonyolultsági osztályok és a „klasszikus” bonyolultsági osztályok közt érdekes összefüggések állnak fenn:

Tétel

$$AL = P.$$

Tétel

$$AP = PSPACE.$$

Az utóbbi tételt megfogalmazhatjuk így is:

PSPACE-be pontosan azok a problémák tartoznak, melyek reprezentálhatók zero összegű, teljes információs, polinom lépésszámú kétszemélyes játékként.

Kétszemélyes játékok

A QSAT felfogható kétszemélyes játékként:

A QSAT felfogható kétszemélyes játékként:

- Játékosok: $\exists, \forall$

Kétszemélyes játékok

A QSAT felfogható **kétszemélyes játékként**:

- Játékosok: $\exists, \forall$
- Lépések felváltva: x_1 értéke, x_2 értéke, $\dots$

A QSAT felfogható **kétszemélyes játékként**:

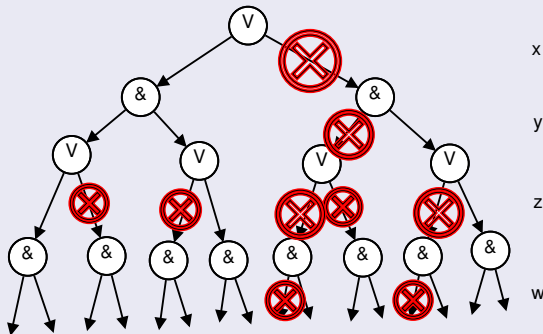
- Játékosok: $\exists, \forall$
- Lépések felváltva: x_1 értéke, x_2 értéke, ...
- $\exists$ célja: kielégíteni a formula magját

A QSAT felfogható **kétszemélyes játékként**:

- Játékosok: $\exists, \forall$
- Lépések felváltva: x_1 értéke, x_2 értéke, ...
- $\exists$ célja: kielégíteni a formula magját
- $\forall$ célja: hamissá tenni

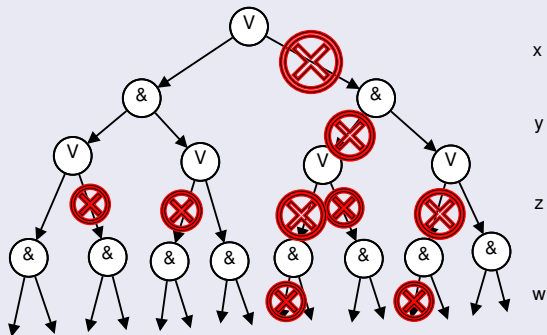
A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



A hosszabb példa

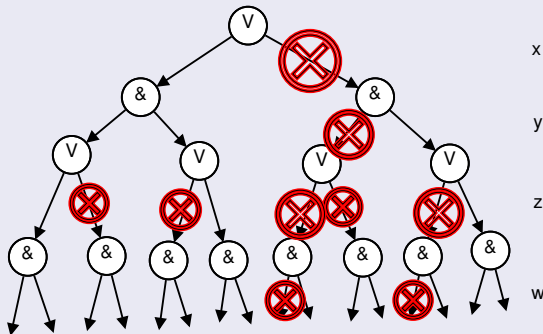
$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



„A formula **igaz**: $x = 0$ választása esetén tetszőleges y -ra legyen $z = y$ és az minden w -re jó lesz.”

A hosszabb példa

$$\exists x \forall y \exists z \forall w \left((\neg x \vee z \vee w) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee z) \right)$$



„A formula **igaz**: $x = 0$ választása esetén tetszőleges y -ra legyen $z = y$ és az minden w -re jó lesz.”

Egy nyerő stratégia $\exists$ -nek: válassza $x = 0$ -t, majd $\forall$ választja y értékét, $\exists$ ugyenezt az értéket adja z -nek, $\forall$ bárhogy is választja meg w értékét, a formula magja igaz lesz.

FÖLDRAJZI JÁTÉK

FÖLDRAJZI JÁTÉK

Adott: $G = (V, E)$ irányított gráf, $1 \in V$ kezdőcsúcs.

Egy kétszemélyes játék

FÖLDRAJZI JÁTÉK

Adott: $G = (V, E)$ irányított gráf, $1 \in V$ kezdőcsúcs.

Kérdés: Az I. játékos nyer-e az alábbi játékban?

FÖLDRAJZI JÁTÉK

Adott: $G = (V, E)$ irányított gráf, $1 \in V$ kezdőcsúcs.

Kérdés: Az I. játékos nyer-e az alábbi játékban?

- Az I. játékos kezd az 1 csúcs megnevezésével.

FÖLDRAJZI JÁTÉK

Adott: $G = (V, E)$ irányított gráf, $1 \in V$ kezdőcsúcs.

Kérdés: Az I. játékos nyer-e az alábbi játékban?

- Az I. játékos kezd az 1 csúcs megnevezésével.
- Minden lépésben minden játékosnak egy olyan csúcsot kell választani, amely még nem szerepelt, és amelybe vezet él az előzőleg megnevezett csúcsból.

FÖLDRAJZI JÁTÉK

Adott: $G = (V, E)$ irányított gráf, $1 \in V$ kezdőcsúcs.

Kérdés: Az I. játékos nyer-e az alábbi játékban?

- Az I. játékos kezd az 1 csúcs megnevezésével.
- Minden lépésben minden játékosnak egy olyan csúcsot kell választani, amely még nem szerepelt, és amelybe vezet él az előzőleg megnevezett csúcsból.
- Egy játékos veszít, ha végül nem tud ilyen csúcsot megnevezni.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.
- Létezik olyan polinom tárigényű algoritmus, mely adott állásra megkonstruálja az állás rákövetkezőit, vagy ha ilyen nincs, eldönti, melyik játékos nyert.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.
- Létezik olyan polinom tárigényű algoritmus, mely adott állásra megkonstruálja az állás rákövetkezőit, vagy ha ilyen nincs, eldönti, melyik játékos nyert.

Minden ilyen kétszemélyes játék **PSPACE**-ben van:

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.
- Létezik olyan polinom tárigényű algoritmus, mely adott állásra megkonstruálja az állás rákövetkezőit, vagy ha ilyen nincs, eldönti, melyik játékos nyert.

Minden ilyen kétszemélyes játék **PSPACE**-ben van:

- Polinom tárral elkészítjük az adott játék fáját.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.
- Létezik olyan polinom tárigényű algoritmus, mely adott állásra megkonstruálja az állás rákövetkezőit, vagy ha ilyen nincs, eldönti, melyik játékos nyert.

Minden ilyen kétszemélyes játék **PSPACE**-ben van:

- Polinom tárral elkészítjük az adott játék fáját.
- Mélységgel arányos tárral eldöntjük, kinek van nyerő stratégiája.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.
- Létezik olyan polinom tárigényű algoritmus, mely adott állásra megkonstruálja az állás rákövetkezőit, vagy ha ilyen nincs, eldönti, melyik játékos nyert.

Minden ilyen kétszemélyes játék **PSPACE**-ben van:

- Polinom tárral elkészítjük az adott játék fáját.
- Mélységgel arányos tárral eldöntjük, kinek van nyerő stratégiája.

A két algoritmus összetehető: polinom tárkorlát.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-ben van.

Ötlet

- A lépéssorozatok hossza a bemenet méretében polinomiális.
- Létezik olyan polinom tárigényű algoritmus, mely adott állásra megkonstruálja az állás rákövetkezőit, vagy ha ilyen nincs, eldönti, melyik játékos nyert.

Minden ilyen kétszemélyes játék **PSPACE**-ben van:

- Polinom tárral elkészítjük az adott játék fáját.
- Mélységgel arányos tárral eldöntjük, kinek van nyerő stratégiája.

A két algoritmus összetehető: polinom tárkorlát.

Állítás

A FÖLDRAJZI JÁTÉK probléma **PSPACE**-teljes.

$$\exists x \forall y \exists z \forall u ((y \vee \neg x) \wedge (y \vee z) \wedge (\neg y \vee u) \wedge (\neg x \vee \neg y))$$

TODO

- Az első játékos dönti el az egzisztenciálisan, a második az univerzálisan kvantifikált változó értékét, mely a címkével ellentétes.

- Az első játékos dönti el az egzisztenciálisan, a második az univerzálisan kvantifikált változó értékét, mely a címkével ellentétes.
- A második játékos választ egy tagot.

- Az első játékos dönti el az egzisztenciálisan, a második az univerzálisan kvantifikált változó értékét, mely a címkével ellentétes.
- A második játékos választ egy tagot.
- Az első játékos akkor és csak akkor nyer, ha a tag valamely literálja igaz (visszavezető él választható).

- Az első játékos dönti el az egzisztenciálisan, a második az univerzálisan kvantifikált változó értékét, mely a címkével ellentétes.
- A második játékos választ egy tagot.
- Az első játékos akkor és csak akkor nyer, ha a tag valamely literálja igaz (visszavezető él választható).
- Mivel ez minden tagra igaz, az első játékos nyer $\Leftrightarrow$ a formula igaz.

Megjegyzés

Irányítatlan gráfok esetén a probléma **P**-ben van.

További közismert **PSPACE**-teljes problémák

- Gó: input egy gó állás, melyik játékosnak van nyerő stratégiája?

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás. . .

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás. . .
- DÁMA: input egy dámaállás. . .

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás...
- DÁMA: input egy dámaállás...
- AMŐBA: egy amőbaállás...

További közismert **PSPACE**-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás. . .
- DÁMA: input egy dámaállás. . .
- AMŐBA: egy amőbaállás. . .
- SOKOBAN: megoldható-e az input Sokoban feladvány?

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás...
- DÁMA: input egy dámaállás...
- AMŐBA: egy amőbaállás...
- SOKOBAN: megoldható-e az input Sokoban feladvány? (egyszemélyes!)

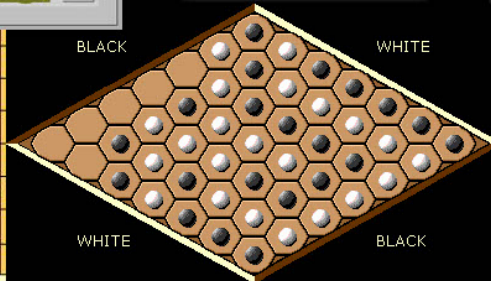
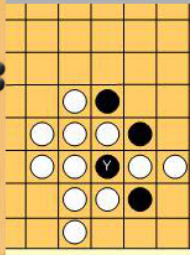
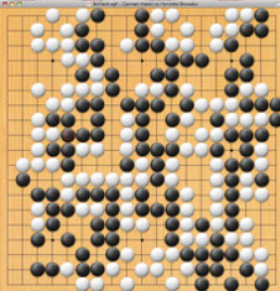
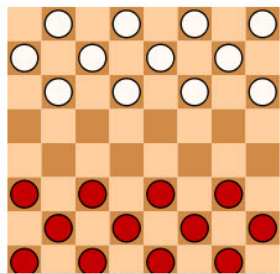
További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás. . .
- DÁMA: input egy dámaállás. . .
- AMŐBA: egy amőbaállás. . .
- SOKOBAN: megoldható-e az input Sokoban feladvány? (egyszemélyes!)
- RUSH HOUR: megoldható-e az input Rush Hour feladvány?

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás...
- DÁMA: input egy dámaállás...
- AMŐBA: egy amőbaállás...
- SOKOBAN: megoldható-e az input Sokoban feladvány? (egyszemélyes!)
- RUSH HOUR: megoldható-e az input Rush Hour feladvány?
- ÉLETJÁTÉK: kipusztul-e minden cella egy adott kezdőállásból elindítva valahány lépésben? („nulla személyes”!)

PSPACE-teljes játékok



Túl a PSPACE osztályon

Definíció

$$\mathbf{EXP} = \text{TIME}(2^{n^k})$$

$$\mathbf{NEXP} = \text{NTIME}(2^{n^k})$$

Definíció

$$\mathbf{EXP} = \text{TIME}(2^{n^k})$$

$$\mathbf{NEXP} = \text{NTIME}(2^{n^k})$$

Világos, hogy $\mathbf{EXP} \subseteq \mathbf{NEXP}$ és tudjuk, hogy $\mathbf{PSPACE} \subseteq \mathbf{EXP}$.

Definíció

$$\mathbf{EXP} = \text{TIME}(2^{n^k})$$

$$\mathbf{NEXP} = \text{NTIME}(2^{n^k})$$

Világos, hogy $\mathbf{EXP} \subseteq \mathbf{NEXP}$ és tudjuk, hogy $\mathbf{PSPACE} \subseteq \mathbf{EXP}$.
Nem ismert, hogy az $\mathbf{EXP}$ és $\mathbf{NEXP}$ osztályok megegyeznek-e.

Túl a PSPACE osztályon

Definíció

$$\mathbf{EXP} = \text{TIME}(2^{n^k})$$

$$\mathbf{NEXP} = \text{NTIME}(2^{n^k})$$

Világos, hogy $\mathbf{EXP} \subseteq \mathbf{NEXP}$ és tudjuk, hogy $\mathbf{PSPACE} \subseteq \mathbf{EXP}$.
Nem ismert, hogy az $\mathbf{EXP}$ és $\mathbf{NEXP}$ osztályok megegyeznek-e.

Tétel

Ha $\mathbf{P} = \mathbf{NP}$, akkor $\mathbf{EXP} = \mathbf{NEXP}$.

Definíció

$$\mathbf{EXPSPACE} = \mathbf{SPACE}(2^{n^k})$$

Definíció

$$\mathbf{EXPSPACE} = \text{SPACE}(2^{n^k})$$

Világos, hogy $\mathbf{NEXP} \subseteq \mathbf{EXPSPACE}$.

Definíció

$$\mathbf{EXPSPACE} = \text{SPACE}(2^{n^k})$$

Világos, hogy $\mathbf{NEXP} \subseteq \mathbf{EXPSPACE}$.

Tétel

Az alábbi probléma **EXPSPACE**-teljes: adott két reguláris kifejezés, melyekben négyzetreemelés is szerepelhet, ekvivalensek-e a kifejezések?

Definíció

$$\mathbf{ELEMI} = \text{TIME}(2^{n^k}) \cup \text{TIME}(2^{2^{n^k}}) \cup \text{TIME}(2^{2^{2^{n^k}}}) \cup \dots$$

Tétel

Az alábbi eldönthető probléma **nem elemi**: adott két reguláris kifejezés, melyekben komplementerképzés is szerepelhet, ekvivalensek-e a kifejezések?

TODO

A Cobham–Edmonds tézis, még egyszer

Egy problémát akkor tekintünk **gyakorlatilag megoldhatónak**, ha van rá polinom időigényű algoritmus.

A Cobham–Edmonds tézis, még egyszer

Egy problémát akkor tekintünk **gyakorlatilag megoldhatónak**, ha van rá polinom időigényű algoritmus.

Biztos?

A Cobham–Edmonds tézis, még egyszer

Egy problémát akkor tekintünk **gyakorlatilag megoldhatónak**, ha van rá polinom időigényű algoritmus.

Biztos?

Vannak esetek, mikor a **lineáris** időigény is **soknak** bizonyul. . .

A Cobham–Edmonds tézis, még egyszer

Egy problémát akkor tekintünk **gyakorlatilag megoldhatónak**, ha van rá polinom időigényű algoritmus.

Biztos?

Vannak esetek, mikor a **lineáris** időigény is **soknak** bizonyul. . .
...de a RAM-gépes kiszámítási modellben nem értelmezhető szublineáris időigény.

A Cobham–Edmonds tézis, még egyszer

Egy problémát akkor tekintünk **gyakorlatilag megoldhatónak**, ha van rá polinom időigényű algoritmus.

Biztos?

Vannak esetek, mikor a **lineáris** időigény is **soknak** bizonyul. . .
... de a RAM-gépes kiszámítási modellben nem értelmezhető szublineáris időigény.

A hatékony párhuzamosítás kérdése azonban gyakorlati szempontból lényeges, így szükség van olyan kiszámítási modellre, melyben pl. $\mathcal{O}(\log n)$ -es időigény is értelmezhető.

PRAM

PRAM

Parallel, random-access machine.

PRAM

Parallel, random-access machine.

- osztott memória

PRAM

Parallel, random-access machine.

- osztott memória
- sok (de „csak” polinom sok) processzor

PRAM

Parallel, random-access machine.

- osztott memória
- sok (de „csak” polinom sok) processzor
- minden processzor a memória bármelyik bitjét $\mathcal{O}(1)$ időben eléri

PRAM

Parallel, random-access machine.

- osztott memória
- sok (de „csak” polinom sok) processzor
- minden processzor a memória bármelyik bitjét $\mathcal{O}(1)$ időben eléri
- a konfliktusok kezelésének módja (EREW, CREW, CRCW) nem rögzített, de nem is lényeges igazán

PRAM

Parallel, random-access machine.

- osztott memória
- sok (de „csak” polinom sok) processzor
- minden processzor a memória bármelyik bitjét $\mathcal{O}(1)$ időben eléri
- a konfliktusok kezelésének módja (EREW, CREW, CRCW) nem rögzített, de nem is lényeges igazán

A matematikai modell

PRAM gépek helyett **logikai hálózatcsaláddal** fogjuk (ekvivalensen) definiálni a hatékony párhuzamosíthatóság fogalmát.

Hálózatcsalád

Hálózatcsaládon logikai hálózatoknak egy $\mathcal{C} = C_0, C_1, \dots$ sorozatát értjük, ahol minden i -re C_i olyan logikai hálózat, melynek input kapuinak címkéje az $\{x_1, \dots, x_i\}$ halmazba esik.

Hálózatcsaláddal való eldöntés

Hálózatcsalád

Hálózatcsaládon logikai hálózatoknak egy $\mathcal{C} = C_0, C_1, \dots$ sorozatát értjük, ahol minden i -re C_i olyan logikai hálózat, melynek input kapuinak címkéje az $\{x_1, \dots, x_i\}$ halmazba esik.

Eldöntés

A $\mathcal{C} = C_0, C_1, \dots$ hálózatcsalád **elfogadja** az $x = b_1 b_2 \dots b_n \in \{0, 1\}^*$ szót, ha $C_n(b_1, b_2, \dots, b_n) = 1$.

Hálózatsaláddal való eldöntés

Hálózatsalád

Hálózatsaládon logikai hálózatoknak egy $\mathcal{C} = C_0, C_1, \dots$ sorozatát értjük, ahol minden i -re C_i olyan logikai hálózat, melynek input kapuinak címkéje az $\{x_1, \dots, x_i\}$ halmazba esik.

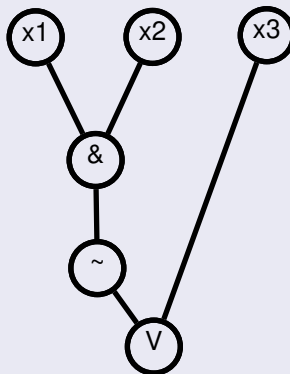
Eldöntés

A $\mathcal{C} = C_0, C_1, \dots$ hálózatsalád **elfogadja** az $x = b_1 b_2 \dots b_n \in \{0, 1\}^*$ szót, ha $C_n(b_1, b_2, \dots, b_n) = 1$.

A $\mathcal{C}$ által eldöntött $L(\mathcal{C}) \subseteq \{0, 1\}^*$ nyelvet a $\mathcal{C}$ által elfogadott szavak alkotják.

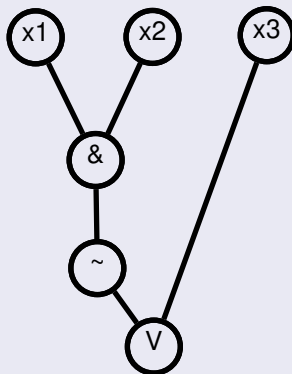
Elfogadás, elvetés

Ha C_3 a következő hálózat:



Elfogadás, elvetés

Ha C_3 a következő hálózat:



akkor $\mathcal{C} = C_0, \dots, C_3, \dots$ elfogadja a 001, 010 szavakat és elveti az 110-t.

$(01)^*$ -ot eldöntő hálózatcsalád

$(01)^*$ -ot eldöntő hálózatcsalád

- C_0 : 

$(01)^*$ -ot eldöntő hálózatcsalád

- C_0 : 
- C_{2i+1} : 

$(01)^*$ -ot eldöntő hálózatsalád

- C_0 : 
- C_{2i+1} : 
- C_{2i+2} :

TODO

Hálózatcsaláddal **minden** nyelv eldönthető

Hálózatcsaláddal **minden** nyelv eldönthető (mert minden Boole-függvényhez van
ő t reprezentáló hálózat),

Hálózatcsaláddal **minden** nyelv eldönthető (mert minden Boole-függvényhez van őt reprezentáló hálózat), így megszorításokat teszünk az „elfogadható” hálózatcsaládra n függvényében

Hálózatcsaláddal **minden** nyelv eldönthető (mert minden Boole-függvényhez van őt reprezentáló hálózat), így megszorításokat teszünk az „elfogadható” hálózatcsaládra n függvényében

- a C_n -beli **kapuk száma** szerint,

Hálózatcsaláddal **minden** nyelv eldönthető (mert minden Boole-függvényhez van őt reprezentáló hálózat), így megszorításokat teszünk az „elfogadható” hálózatcsaládra n függvényében

- a C_n -beli **kapuk száma** szerint,
- a C_n hálózat **mélysége** szerint,

Hálózatcsaláddal **minden** nyelv eldönthető (mert minden Boole-függvényhez van őt reprezentáló hálózat), így megszorításokat teszünk az „elfogadható” hálózatcsaládra n függvényében

- a C_n -beli **kapuk száma** szerint,
- a C_n hálózat **mélysége** szerint,
- a hálózatcsalád tagjaiban megengedett **kaputípusok szerint**,

Hálózatcsaláddal **minden** nyelv eldönthető (mert minden Boole-függvényhez van őt reprezentáló hálózat), így megszorításokat teszünk az „elfogadható” hálózatcsaládra n függvényében

- a C_n -beli **kapuk száma** szerint,
- a C_n hálózat **mélysége** szerint,
- a hálózatcsalád tagjaiban megengedett **kaputípusok szerint**,
- **uniformitás** szerint (egy hálózatcsalád akkor (logtár-)uniform, ha van olyan algoritmus, mely C_n -t előállítja $\mathcal{O}(\log n)$ tár felhasználása mellett).

A hatékonyan párhuzamosítható problémák osztálya

Definíció

Ha $k \geq 0$ konstans, akkor az NC^k osztályba azok a $\{0, 1\}$ fölötti nyelvek tartoznak, melyek eldönthetők

Definíció

Ha $k \geq 0$ konstans, akkor az NC^k osztályba azok a $\{0, 1\}$ fölötti nyelvek tartoznak, melyek eldönthetők

- valamely c konstansra $\mathcal{O}(n^c)$, vagyis polinom sok kaput tartalmazó,

Definíció

Ha $k \geq 0$ konstans, akkor az **NC^k osztályba** azok a $\{0, 1\}$ fölötti nyelvek tartoznak, melyek eldönthetők

- valamely c konstansra $\mathcal{O}(n^c)$, vagyis polinom sok kaput tartalmazó,
- $\mathcal{O}(\log^k n)$ mélységű,

Definíció

Ha $k \geq 0$ konstans, akkor az **NC^k osztályba** azok a $\{0, 1\}$ fölötti nyelvek tartoznak, melyek eldönthetők

- valamely c konstansra $\mathcal{O}(n^c)$, vagyis polinom sok kaput tartalmazó,
- $\mathcal{O}(\log^k n)$ mélységű,
- legfeljebb kettő befokú kapukat tartalmazó,

Definíció

Ha $k \geq 0$ konstans, akkor az **NC^k osztályba** azok a $\{0, 1\}$ fölötti nyelvek tartoznak, melyek eldönthetők

- valamely c konstansra $\mathcal{O}(n^c)$, vagyis polinom sok kaput tartalmazó,
- $\mathcal{O}(\log^k n)$ mélységű,
- legfeljebb kettő befokú kapukat tartalmazó,
- uniform

hálózatcsaláddal.

A hatékonyan párhuzamosítható problémák osztálya

Definíció

Ha $k \geq 0$ konstans, akkor az NC^k osztályba azok a $\{0, 1\}$ fölötti nyelvek tartoznak, melyek eldönthetők

- valamely c konstansra $\mathcal{O}(n^c)$, vagyis polinom sok kaput tartalmazó,
- $\mathcal{O}(\log^k n)$ mélységű,
- legfeljebb kettő befokú kapukat tartalmazó,
- uniform

hálózatcsaláddal.

$NC = \bigcup_{k \geq 0} NC^k$, a hatékonyan párhuzamosítható problémák osztálya.

NC^1 : logaritmus mélységű hálózatsalád

Példa: 1^n

Az 1^* nyelv eldönthető logaritmus mélységű hálózattal:

NC^1 : logaritmus mélységű hálózatsalád

Példa: 1^n

Az 1^* nyelv eldönthető logaritmus mélységű hálózattal: C_n az input kapukat $\wedge$ kapukkal köti össze, egy majdnem teljes bináris fába szervezett topológiával.

NC^1 : logaritmus mélységű hálózatsalád

Példa: 1^n

Az 1^* nyelv eldönthető logaritmus mélységű hálózattal: C_n az input kapukat $\wedge$ kapukkal köti össze, egy majdnem teljes bináris fába szervezett topológiával. Pl. C_{10} :

TODO

Több output kapu

Hálózatcsaládokkal **függvényeket** is kiszámíthatunk: ha a $\mathcal{C} = C_0, C_1, \dots$ családban a C_n tagnak o_n kimeneti kapuval rendelkezik (rögzített sorrendben), akkor a család értelemszerűen kiszámít egy $\{0, 1\}^* \rightarrow \{0, 1\}^*$ függvényt; az n hosszú input szavakhoz o_n hosszú output szót rendel.

Összeadás

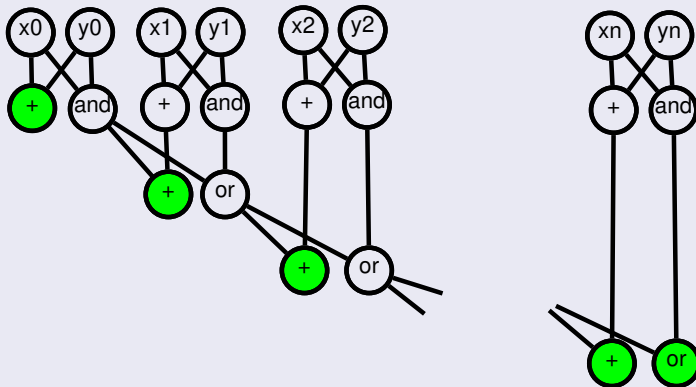
Összeadás: a C_n hálózat **input kapui** az $x_0, \dots, x_{n-1}$, $y_0, \dots, y_{n-1}$ változókkal címkézettek, output kapui $z_0, \dots, z_n$.

Összeadás

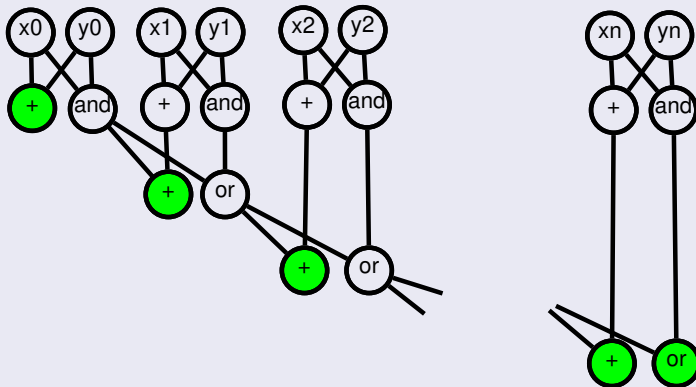
Összeadás: a C_n hálózat **input kapui** az $x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}$ változókkal címkézettek, output kapui $z_0, \dots, z_n$.

A bemeneti két n -bites szám összegét (egy $n + 1$ -bites számot) kell a kimenetre küldenie.

Triviális megvalósítás



Triviális megvalósítás



Lineáris mélység!

Összeadás logaritmikus mélységben

Elég kiszámítani a carry biteket ($c_i = 1$, ha az i . helyiértéken keletkezik átvitel, vagyis ha az $x_0x_1 \dots x_{n-1} + y_0y_1 \dots y_{i-1}$ összeg i . helyiértéke 1-es).

Összeadás logaritmikus mélységben

Elég kiszámítani a carry biteket ($c_i = 1$, ha az i . helyiértéken keletkezik átvitel, vagyis ha az $x_0x_1 \dots x_{n-1} + y_0y_1 \dots y_{i-1}$ összeg i . helyiértéke 1-es. Ezek után $z_i = x_i \oplus y_i \oplus c_{i-1}$, ahol $\oplus$ a kizáró vagy művelet (összeadás $\mathbb{Z}_2$ -ben).

Összeadás logaritmikus mélységben

Elég kiszámítani a carry biteket ($c_i = 1$, ha az i . helyiértéken keletkezik átvitel, vagyis ha az $x_0x_1 \dots x_{n-1} + y_0y_1 \dots y_{i-1}$ összeg i . helyiértéke 1-es. Ezek után $z_i = x_i \oplus y_i \oplus c_{i-1}$, ahol $\oplus$ a kizáró vagy művelet (összeadás $\mathbb{Z}_2$ -ben). (pontosabban, $z_n = c_{n-1}$ és $z_0 = x_0 \oplus y_0$.) Ez párhuzamosan, konstans mélységben elvégezhető.

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összefüggések

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összefüggések

- $c_{i,i} = x_i \wedge y_i$.

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összefüggések

- $c_{i,i} = x_i \wedge y_i$.
- $p_{i,i} = x_i \vee y_i$.

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összefüggések

- $c_{i,i} = x_i \wedge y_i$.
- $p_{i,i} = x_i \vee y_i$.
- $c_{i,i+2d} = (c_{i,i+d} \wedge p_{i+d+1,i+2d}) \vee c_{i+d+1,i+2d}$.

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összefüggések

- $c_{i,i} = x_i \wedge y_i$.
- $p_{i,i} = x_i \vee y_i$.
- $c_{i,i+2d} = (c_{i,i+d} \wedge p_{i+d+1,i+2d}) \vee c_{i+d+1,i+2d}$.
- $p_{i,i+2d} = p_{i,i+d} \wedge p_{i+d+1,i+2d}$.

Összeadás logaritmikus mélységben

Carry és propagate

Adott $x_0 \dots x_{n-1}$ és $y_0 \dots y_{n-1}$ input egészek, bitsorozat formájában és $0 \leq i \leq j \leq n$. Legyen

- $c_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ összegnél keletkezik carry;
- $p_{i,j} = 1$, ha az $x_i x_{i+1} \dots x_j + y_i y_{i+1} \dots y_j$ **+1** összegnél keletkezik carry.

Összefüggések

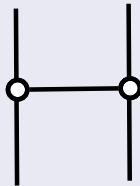
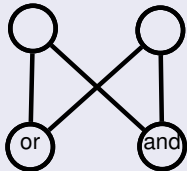
- $c_{i,i} = x_i \wedge y_i$.
- $p_{i,i} = x_i \vee y_i$.
- $c_{i,i+2d} = (c_{i,i+d} \wedge p_{i+d+1,i+2d}) \vee c_{i+d+1,i+2d}$.
- $p_{i,i+2d} = p_{i,i+d} \wedge p_{i+d+1,i+2d}$.

Ezekkel az összefüggésekkel $\mathcal{O}(\log n)$ mélységben kiszámítható a carry és a propagate az összes $(0, j)$ párra.

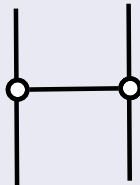
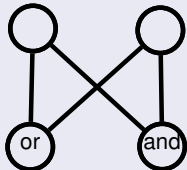
A páros-páratlan mergesort (Batcher)

```
function MERGE( $T[1..2N]$ )  
  if  $N == 1$  then compareAndSwap( $T[1], T[2]$ )  
  else  
    MERGE( $T[1, 3, 5, \dots, 2N - 1]$ )  
    MERGE( $T[2, 4, 6, \dots, 2N]$ )  
    for  $i = 2, 4, 6, \dots, 2N - 2$  do compareAndSwap( $T[i], T[i + 1]$ )  
  
function SORT( $T[1..2N]$ )  
  if  $N == 1$  then compareAndSwap( $T[1], T[2]$ )  
  else  
    SORT( $T[1..N]$ )  
    SORT( $T[N + 1..2N]$ )  
    MERGE( $T[1..2N]$ )
```

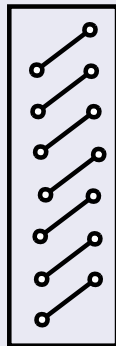
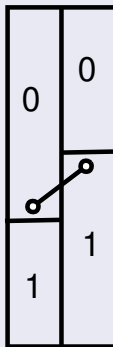
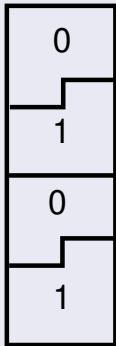
compareAndSwap



compareAndSwap



A páros-páratlan mergesort vizualizálva



Rendezés $\mathcal{O}(\log^2 n)$ mélységben

A teljes rendező hálózat mélysége: $\mathcal{O}(\log^2 n)$.

Példa: 32-bites rendező hálózat



A TÖBBSÉGI FÜGGVÉNY akkor vesz fel 1 értéket, ha argumentumainak legalább a fele 1-es.

TÖBBSÉGI FÜGGVÉNY $\in NC^2$

A TÖBBSÉGI FÜGGVÉNY akkor vesz fel 1 értéket, ha argumentumainak legalább a fele 1-es.

A függvény az előzőek szerint NC^2 -beli:

TÖBBSÉGI FÜGGVÉNY $\in NC^2$

A TÖBBSÉGI FÜGGVÉNY akkor vesz fel 1 értéket, ha argumentumainak legalább a fele 1-es.

A függvény az előzőek szerint NC^2 -beli:

- rendezzük a biteket;

TÖBBSÉGI FÜGGVÉNY $\in NC^2$

A TÖBBSÉGI FÜGGVÉNY akkor vesz fel 1 értéket, ha argumentumainak legalább a fele 1-es.

A függvény az előzőek szerint NC^2 -beli:

- rendezzük a biteket;
- visszaadjuk az $\frac{n}{2}$. elemet a rendezett sorozatból.

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Az output kapuk: $z_{i,j}$, a két mátrix szorzatának i . sorának j . oszlopának értéke

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Az output kapuk: $z_{i,j}$, a két mátrix szorzatának i . sorának j . oszlopának értéke

$$z_{i,j} = \bigvee_{k=1}^n (x_{i,k} \wedge y_{k,j}).$$

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Az output kapuk: $z_{i,j}$, a két mátrix szorzatának i . sorának j . oszlopának értéke

$$z_{i,j} = \bigvee_{k=1}^n (x_{i,k} \wedge y_{k,j}).$$

$\log n$ mélység

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Az output kapuk: $z_{i,j}$, a két mátrix szorzatának i . sorának j . oszlopának értéke

$$z_{i,j} = \bigvee_{k=1}^n (x_{i,k} \wedge y_{k,j}).$$

$\log n$ mélység

- Minden (i, k, j) hármásra kiszámítjuk az $x_{i,k} \wedge y_{k,j}$ értéket párhuzamosan: 1 mélység, n^3 kapu.

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Az output kapuk: $z_{i,j}$, a két mátrix szorzatának i . sorának j . oszlopának értéke

$$z_{i,j} = \bigvee_{k=1}^n (x_{i,k} \wedge y_{k,j}).$$

$\log n$ mélység

- Minden (i, k, j) hármasra kiszámítjuk az $x_{i,k} \wedge y_{k,j}$ értéket párhuzamosan: 1 mélység, n^3 kapu.
- Minden (i, j) párosra a fenti $z_{i,j}$ értéket párhuzamosan: $\log n$ mélység (az 1^n -nél látott majdnem teljes bináris fa topológiával), összesen n^3 kapu.

Mátrixszorzás: $\mathcal{O}(\log n)$ mélység

Az input kapuk: két $n \times n$ -es Boole-mátrix elemei, $x_{i,j}$ ill. $y_{i,j}$ a megfelelő mátrix i . sorának, j . oszlopának értéke

Az output kapuk: $z_{i,j}$, a két mátrix szorzatának i . sorának j . oszlopának értéke

$$z_{i,j} = \bigvee_{k=1}^n (x_{i,k} \wedge y_{k,j}).$$

$\log n$ mélység

- Minden (i, k, j) hármasra kiszámítjuk az $x_{i,k} \wedge y_{k,j}$ értéket párhuzamosan: 1 mélység, n^3 kapu.
- Minden (i, j) párosra a fenti $z_{i,j}$ értéket párhuzamosan: $\log n$ mélység (az 1^n -nél látott majdnem teljes bináris fa topológiával), összesen n^3 kapu.

Vagyis: $\mathcal{O}(\log n)$ mélység, polinom sok kapu.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.
- $A + I$: a főátlót 1-re állítjuk.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.
- $A + I$: a főátlót 1-re állítjuk.
- $(A + I)^k$: az (i, j) cella értéke pontosan akkor 1, ha van legfeljebb k hosszú séta i -ből j -be.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.
- $A + I$: a főátlót 1-re állítjuk.
- $(A + I)^k$: az (i, j) cella értéke pontosan akkor 1, ha van legfeljebb k hosszú séta i -ből j -be.
- Iterált négyzetreemeléssel meghatározzuk $(A + I)^{2^{\lceil \log n \rceil}}$ -t, majd ennek visszaadjuk az $(1, n)$ celláját.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.
- $A + I$: a főátlót 1-re állítjuk.
- $(A + I)^k$: az (i, j) cella értéke pontosan akkor 1, ha van legfeljebb k hosszú séta i -ből j -be.
- Iterált négyzetreemeléssel meghatározzuk $(A + I)^{2^{\lceil \log n \rceil}}$ -t, majd ennek visszaadjuk az $(1, n)$ celláját.
- Egy négyzetreemelés: $\mathcal{O}(\log n)$ mélység.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.
- $A + I$: a főátlót 1-re állítjuk.
- $(A + I)^k$: az (i, j) cella értéke pontosan akkor 1, ha van legfeljebb k hosszú séta i -ből j -be.
- Iterált négyzetreemeléssel meghatározzuk $(A + I)^{2^{\lceil \log n \rceil}}$ -t, majd ennek visszaadjuk az $(1, n)$ celláját.
- Egy négyzetreemelés: $\mathcal{O}(\log n)$ mélység.
- $\log n$ négyzetreemelés szekvenciálisan.

- Legyen $A \in \{0, 1\}^{n \times n}$ a G gráf szomszédsági mátrixa.
- $A + I$: a főátlót 1-re állítjuk.
- $(A + I)^k$: az (i, j) cella értéke pontosan akkor 1, ha van legfeljebb k hosszú séta i -ből j -be.
- Iterált négyzetreemeléssel meghatározzuk $(A + I)^{2^{\lceil \log n \rceil}}$ -t, majd ennek visszaadjuk az $(1, n)$ celláját.
- Egy négyzetreemelés: $\mathcal{O}(\log n)$ mélység.
- $\log n$ négyzetreemelés szekvenciálisan.
- Összesen $\mathcal{O}(\log^2 n)$ mélység.

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Nem ismert, hogy

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Nem ismert, hogy

- igaz-e, hogy $NC^i = NC^{i+1}$ valamely i -re?

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Nem ismert, hogy

- igaz-e, hogy $NC^i = NC^{i+1}$ valamely i -re?
- NC^1 vs. L ?

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Nem ismert, hogy

- igaz-e, hogy $NC^i = NC^{i+1}$ valamely i -re?
- NC^1 vs. L ?
- NC^1 vs. NL ?

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Nem ismert, hogy

- igaz-e, hogy $NC^i = NC^{i+1}$ valamely i -re?
- NC^1 vs. L ?
- NC^1 vs. NL ?
- NC^2 vs. NL ?

Az NC osztályok a klasszikusak közt

Tétel

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq NC^3 \subseteq \dots \subseteq NC \subseteq P.$$

Nem ismert, hogy

- igaz-e, hogy $NC^i = NC^{i+1}$ valamely i -re?
- NC^1 vs. L ?
- NC^1 vs. NL ?
- NC^2 vs. NL ?
- NC vs. P ?

Az utolsó kérdés másképp: igaz-e, hogy minden hatékonyan megoldható (P -beli) probléma hatékonyan párhuzamosítható is, vagy vannak „inherensen szekvenciális” problémák?

Alapfeladat

- Alice üzenetet akar küldeni Bobnak

Alapfeladat

- Alice üzenetet akar küldeni Bobnak



Alapfeladat

- Alice üzenetet akar küldeni Bobnak



- Chuck ne ismerhesse meg az üzenet tartalmát a hálózat lehallgatásával

Alapfeladat

- Alice üzenetet akar küldeni Bobnak



- Chuck ne ismerhesse meg az üzenet tartalmát a hálózat lehallgatásával
- Megegyeznek két **algoritmusban**, E – kódoló és D – dekódoló és két **kulcsban**, $e \in \{0, 1\}^*$ és $d \in \{0, 1\}^*$

Alapfeladat

- Alice üzenetet akar küldeni Bobnak



- Chuck ne ismerhesse meg az üzenet tartalmát a hálózat lehallgatásával
- Megegyeznek két **algoritmusban**, E – kódoló és D – dekódoló és két **kulcsban**, $e \in \{0, 1\}^*$ és $d \in \{0, 1\}^*$
- Oly módon, hogy tetszőleges $x \in \{0, 1\}^*$ üzenetre $D(d, E(e, x)) = x$ legyen

Alapfeladat

- Alice üzenetet akar küldeni Bobnak



- Chuck ne ismerhesse meg az üzenet tartalmát a hálózat lehallgatásával
- Megegyeznek két **algoritmusban**, E – kódoló és D – dekódoló és két **kulcsban**, $e \in \{0, 1\}^*$ és $d \in \{0, 1\}^*$
- Oly módon, hogy tetszőleges $x \in \{0, 1\}^*$ üzenetre $D(d, E(e, x)) = x$ legyen
- Továbbá, D és E legyen hatékonyan kiszámítható

Alapfeladat

- Alice üzenetet akar küldeni Bobnak



- Chuck ne ismerhesse meg az üzenet tartalmát a hálózat lehallgatásával
- Megegyeznek két **algoritmusban**, E – kódoló és D – dekódoló és két **kulcsban**, $e \in \{0, 1\}^*$ és $d \in \{0, 1\}^*$
- Oly módon, hogy tetszőleges $x \in \{0, 1\}^*$ üzenetre $D(d, E(e, x)) = x$ legyen
- Továbbá, D és E legyen hatékonyan kiszámítható
- Viszont $E(e, x)$ -ből d nélkül x ne legyen visszanyerhető

One-time pad

D és E

D és E

- A D , E algoritmusokról feltesszük, hogy Chuck ismeri őket

D és E

- A D , E algoritmusokról feltesszük, hogy Chuck ismeri őket
- Csak a kulcsokat nem ismeri (míg azok át nem mennek a hálózaton titkosítatlanul...)

One-time pad

D és E

- A D , E algoritmusokról feltesszük, hogy Chuck ismeri őket
- Csak a kulcsokat nem ismeri (míg azok át nem mennek a hálózaton titkosítatlanul...)

One-time pad

Ha $d = e$ egy (véletlenül generált) $n = |x|$ -bites szám, $E = D$ pedig a bitenkénti XOR művelet, ez teljesíti az előbbi elvárásokat

One-time pad

D és E

- A D , E algoritmusokról feltesszük, hogy Chuck ismeri őket
- Csak a kulcsokat nem ismeri (míg azok át nem mennek a hálózaton titkosítatlanul...)

One-time pad

Ha $d = e$ egy (véletlenül generált) $n = |x|$ -bites szám, $E = D$ pedig a bitenkénti XOR művelet, ez teljesíti az előbbi elvárásokat

Csakhogy

One-time pad

D és E

- A D , E algoritmusokról feltesszük, hogy Chuck ismeri őket
- Csak a kulcsokat nem ismeri (míg azok át nem mennek a hálózaton titkosítatlanul...)

One-time pad

Ha $d = e$ egy (véletlenül generált) $n = |x|$ -bites szám, $E = D$ pedig a bitenkénti XOR művelet, ez teljesíti az előbbi elvárásokat

Csakhogy

- a kulcsot csak egyetlen üzenet titkosítására használhatjuk

One-time pad

D és E

- A D , E algoritmusokról feltesszük, hogy Chuck ismeri őket
- Csak a kulcsokat nem ismeri (míg azok át nem mennek a hálózaton titkosítatlanul...)

One-time pad

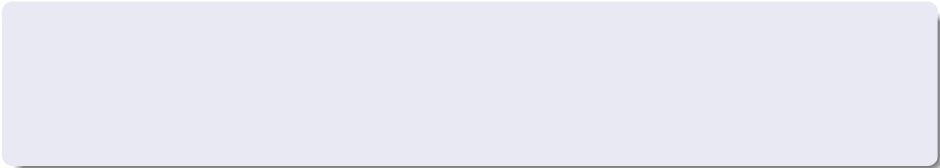
Ha $d = e$ egy (véletlenül generált) $n = |x|$ -bites szám, $E = D$ pedig a bitenkénti XOR művelet, ez teljesíti az előbbi elvárásokat

Csakhogy

- a kulcsot csak egyetlen üzenet titkosítására használhatjuk
- a kulcsot mindkét félnek ismernie kell (**szimmetrikus** titkosítás)

A kulcs titkosított átküldésének problémája pedig (most) ekvivalens az eredeti feladattal.

Publikus kulcsú titkosítások



Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,

Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,
- és e -ből d -t vagy $E(e, x)$ -ből x -et **gyakorlatilag** nem lehet d ismerete nélkül kiszámítani,

akkor nyilvános kulcsú titkosításról beszélünk.

Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,
- és e -ből d -t vagy $E(e, x)$ -ből x -et **gyakorlatilag** nem lehet d ismerete nélkül kiszámítani,

akkor nyilvános kulcsú titkosításról beszélünk.

„Gyakorlatilag”

Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,
- és e -ből d -t vagy $E(e, x)$ -ből x -et **gyakorlatilag** nem lehet d ismerete nélkül kiszámítani,

akkor nyilvános kulcsú titkosításról beszélünk.

„Gyakorlatilag”

- Az ONE-TIME PAD elméletileg is feltörhetetlen.

Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,
- és e -ből d -t vagy $E(e, x)$ -ből x -et **gyakorlatilag** nem lehet d ismerete nélkül kiszámítani,

akkor nyilvános kulcsú titkosításról beszélünk.

„Gyakorlatilag”

- Az ONE-TIME PAD elméletileg is feltörhetetlen.
- A publikus kulcsú titkosítások azonban **nem** azok:

Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,
- és e -ből d -t vagy $E(e, x)$ -ből x -et **gyakorlatilag** nem lehet d ismerete nélkül kiszámítani,

akkor nyilvános kulcsú titkosításról beszélünk.

„Gyakorlatilag”

- Az ONE-TIME PAD elméletileg is feltörhetetlen.
- A publikus kulcsú titkosítások azonban **nem** azok:
- Ha Chuck ismeri $y = E(e, x)$ -et és E -t, akkor **nemdeterminisztikusan** megsejtve x -et tudja ellenőrizni, hogy eltalálta-e, hiszen e számára is ismert

Publikus kulcsú titkosítások

- Ha az e titkosító kulcs **mindenki** számára ismert, d pedig **csak Bob** számára,
- és e -ből d -t vagy $E(e, x)$ -ből x -et **gyakorlatilag** nem lehet d ismerete nélkül kiszámítani,

akkor nyilvános kulcsú titkosításról beszélünk.

„Gyakorlatilag”

- Az ONE-TIME PAD elméletileg is feltörhetetlen.
- A publikus kulcsú titkosítások azonban **nem** azok:
- Ha Chuck ismeri $y = E(e, x)$ -et és E -t, akkor **nemdeterminisztikusan** megsejtve x -et tudja ellenőrizni, hogy eltalálta-e, hiszen e számára is ismert
- Ez tehát **FNP**-beli probléma $\Rightarrow$ abban az extrém esetben, ha **$P = NP$** , minden ilyen rendszer polinomidőben törhető

Egy f (parciális) **injektív** függvény **egyirányú**, ha

Egy f (parciális) **injektív** függvény **egyirányú**, ha

- minden x -re $f(x)$ legfeljebb polinomiálisan hosszabb/rövidebb x -nél;

Egy f (parciális) **injektív** függvény **egyirányú**, ha

- minden x -re $f(x)$ legfeljebb polinomiálisan hosszabb/rövidebb x -nél;
- f kiszámítható polinomidőben;

Egyirányú függvények

Egy f (parciális) **injektív** függvény **egyirányú**, ha

- minden x -re $f(x)$ legfeljebb polinomiálisan hosszabb/rövidebb x -nél;
- f kiszámítható polinomidőben;
- f^{-1} nem számítható ki polinomidőben.

Egyirányú függvények

Egy f (parciális) **injektív** függvény **egyirányú**, ha

- minden x -re $f(x)$ legfeljebb polinomiálisan hosszabb/rövidebb x -nél;
- f kiszámítható polinomidőben;
- f^{-1} nem számítható ki polinomidőben.

Nyilván az inverz mindenképpen **FNP**-beli (megsejtjük, ellenőrizzük).

Egyirányú függvények

Egy f (parciális) **injektív** függvény **egyirányú**, ha

- minden x -re $f(x)$ legfeljebb polinomiálisan hosszabb/rövidebb x -nél;
- f kiszámítható polinomidőben;
- f^{-1} nem számítható ki polinomidőben.

Nilván az inverz mindenképpen **FNP**-beli (megsejtjük, ellenőrizzük).

Ha a titkosító algoritmus egyirányú függvénnyel történik, akkor a kulcs ismerete nélkül nem lehet visszaállítani az inputot.

Prímek szorzása

- Input: $p < q$ prímek
- Output: $p \cdot q$

Prímek szorzása

- Input: $p < q$ prímek
- Output: $p \cdot q$

A függvény nyilván injektív, polinomidőben kiszámítható és **jelen állás szerint** nem ismert, hogy inverze kiszámítható lenne polinomidőben.

Prímek szorzása

- Input: $p < q$ prímek
- Output: $p \cdot q$

A függvény nyilván injektív, polinomidőben kiszámítható és **jelen állás szerint** nem ismert, hogy inverze kiszámítható lenne polinomidőben.

Moduláris hatványozás

- Input: p prím, r **primitív gyöke**, $0 < x < p$ egész
- Output: $r^x \bmod p$.

Prímek szorzása

- Input: $p < q$ prímek
- Output: $p \cdot q$

A függvény nyilván injektív, polinomidőben kiszámítható és **jelen állás szerint** nem ismert, hogy inverze kiszámítható lenne polinomidőben.

Moduláris hatványozás

- Input: p prím, r **primitív gyöke**, $0 < x < p$ egész
- Output: $r^x \bmod p$.

Mivel r primitív gyök, a függvény injektív;

Prímek szorzása

- Input: $p < q$ prímek
- Output: $p \cdot q$

A függvény nyilván injektív, polinomidőben kiszámítható és **jelen állás szerint** nem ismert, hogy inverze kiszámítható lenne polinomidőben.

Moduláris hatványozás

- Input: p prím, r **primitív gyöke**, $0 < x < p$ egész
- Output: $r^x \bmod p$.

Mivel r primitív gyök, a függvény injektív; mivel van iterált moduláris hatványozás, polinomidőben kiszámítható,

Prímek szorzása

- Input: $p < q$ prímek
- Output: $p \cdot q$

A függvény nyilván injektív, polinomidőben kiszámítható és **jelen állás szerint** nem ismert, hogy inverze kiszámítható lenne polinomidőben.

Moduláris hatványozás

- Input: p prím, r **primitív gyöke**, $0 < x < p$ egész
- Output: $r^x \bmod p$.

Mivel r primitív gyök, a függvény injektív; mivel van iterált moduláris hatványozás, polinomidőben kiszámítható, és **jelen állás szerint** nem ismert, hogy inverze kiszámítható lenne polinomidőben.

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek,

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt),

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt), és szintén nem ismert, hogy inverze polinomidőben kiszámítható lenne.

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt), és szintén nem ismert, hogy inverze polinomidőben kiszámítható lenne.

Publikus kulcsú titkosításként

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt), és szintén nem ismert, hogy inverze polinomidőben kiszámítható lenne.

Publikus kulcsú titkosításként

- Bob generál p, q prímeket, és egy e számot, ami relatív prím $\Phi(pq)$ -hoz

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt), és szintén nem ismert, hogy inverze polinomidőben kiszámítható lenne.

Publikus kulcsú titkosításként

- Bob generál p, q prímeket, és egy e számot, ami relatív prím $\Phi(pq)$ -hoz
- Bob kiszámítja azt a d számot, amire $e \cdot d \equiv 1 \bmod \Phi(pq)$

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt), és szintén nem ismert, hogy inverze polinomidőben kiszámítható lenne.

Publikus kulcsú titkosításként

- Bob generál p, q prímeket, és egy e számot, ami relatív prím $\Phi(pq)$ -hoz
- Bob kiszámítja azt a d számot, amire $e \cdot d \equiv 1 \bmod \Phi(pq)$
- Publikus: pq és e

Adott: p, q prímszámok, e szám úgy, hogy e és $\Phi(pq) = (p-1)(q-1)$ relatív prímek.

- Input: $0 \leq x < pq$ szám
- Output: $x^e \bmod pq$

A függvény **injektív**, ha e és $\Phi(pq)$ tényleg relatív prímek, polinomidejű (ismét iterált moduláris hatványozás miatt), és szintén nem ismert, hogy inverze polinomidőben kiszámítható lenne.

Publikus kulcsú titkosításként

- Bob generál p, q prímeket, és egy e számot, ami relatív prím $\Phi(pq)$ -hoz
- Bob kiszámítja azt a d számot, amire $e \cdot d \equiv 1 \bmod \Phi(pq)$
- Publikus: pq és e
- Privát: d

Trapdoor függvények

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.

Trapdoor függvények

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.
Ezért az f egyirányú függvényre további megszorításokat teszünk:

Trapdoor függvények

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.

Ezért az f egyirányú függvényre további megszorításokat teszünk:

Ha még az is igaz, hogy

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.

Ezért az f egyirányú függvényre további megszorításokat teszünk:

Ha még az is igaz, hogy

- az f értelmezési tartományából **hatékonyan vehetünk mintát** és

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.

Ezért az f egyirányú függvényre további megszorításokat teszünk:

Ha még az is igaz, hogy

- az f értelmezési tartományából **hatékonyan vehetünk mintát** és
- létezik olyan d függvény, melynek ismeretében f^{-1} kiszámítása hatékony,

Trapdoor függvények

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.

Ezért az f egyirányú függvényre további megszorításokat teszünk:

Ha még az is igaz, hogy

- az f értelmezési tartományából **hatékonyan vehetünk mintát** és
- létezik olyan d függvény, melynek ismeretében f^{-1} kiszámítása hatékony, akkor f egy trapdoor (csapda-)függvény.

Trapdoor függvények

Ha az egyirányú függvénnyel elkódolt üzenetet Bob sem tudja semmivel hatékonyan visszafejteni, az nem túl hasznos.

Ezért az f egyirányú függvényre további megszorításokat teszünk:

Ha még az is igaz, hogy

- az f értelmezési tartományából **hatékonyan vehetünk mintát** és
- létezik olyan d függvény, melynek ismeretében f^{-1} kiszámítása hatékony, akkor f egy trapdoor (csapda-)függvény.

Az RSA **talán** trapdoor függvény, ezért alkalmas publikus kulcsú titkosításra (a prímszorzás és a moduláris hatványozás önmagukban nem azok).

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

- a „nem” példányokat minden szálon elveti,

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

- a „nem” példányokat minden szálon elveti,
- az „igen” példányokat **pontosan egy** szálon fogadja el.

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

- a „nem” példányokat minden szálon elveti,
- az „igen” példányokat **pontosan egy** szálon fogadja el.

(vessük össze: **RP**-nél pont hogy **sok** elfogadó szálát kötöttünk ki.)

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

- a „nem” példányokat minden szálon elveti,
- az „igen” példányokat **pontosan egy** szálon fogadja el.

(vessük össze: **RP**-nél pont hogy **sok** elfogadó szálát kötöttünk ki.)

Nyilván $P \subseteq UP \subseteq NP$.

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

- a „nem” példányokat minden szálon elveti,
- az „igen” példányokat **pontosan egy** szálon fogadja el.

(vessük össze: **RP**-nél pont hogy **sok** elfogadó szálát kötöttünk ki.)

Nyilván $P \subseteq UP \subseteq NP$.

Például FAKTORIZÁLÁS $\in UP$.

Egy probléma akkor van az **UP** osztályban, ha eldönthető olyan polinomidejű **nemdeterminisztikus** A programmal, mely

- a „nem” példányokat minden szálon elveti,
- az „igen” példányokat **pontosan egy** szálon fogadja el.

(vessük össze: **RP**-nél pont hogy **sok** elfogadó szálát kötöttünk ki.)

Nyilván $P \subseteq UP \subseteq NP$.

Például $FAKTORIZÁLÁS \in UP$.

Tétel

$P \neq UP$ pontosan akkor, **ha van egyirányú függvény**.

„Quantum mechanics: Real Black Magic Calculus.”

– Albert Einstein, 1925

„Quantum mechanics: Real Black Magic Calculus.”

– Albert Einstein, 1925

„God does not play dice with the universe.”

– Albert Einstein

„Quantum mechanics: Real Black Magic Calculus.”

– Albert Einstein, 1925

„God does not play dice with the universe.”

– Albert Einstein

„Not only does God play dice but... he sometimes throws them where they cannot be seen.”

– Stephen Hawking

„Quantum mechanics: Real Black Magic Calculus.”

– Albert Einstein, 1925

„God does not play dice with the universe.”

– Albert Einstein

„Not only does God play dice but... he sometimes throws them where they cannot be seen.”

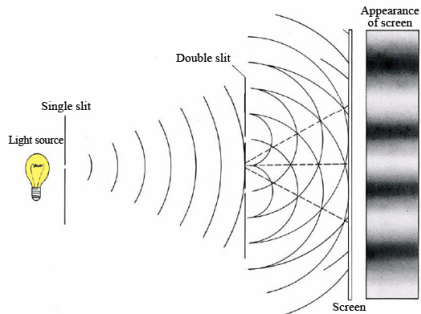
– Stephen Hawking

Mottó

„How I want a drink, alcoholic of course, after the heavy lectures involving quantum mechanics.”

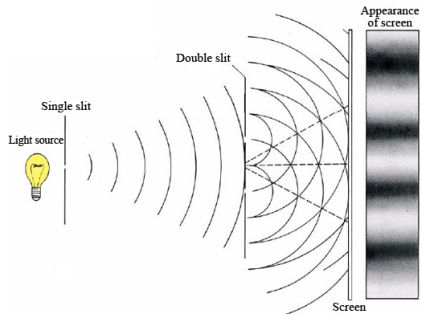
– Sir James Jeans

A kétrés kísérlet: hullám, mérés, részecske



Hullámtermészet? Csakhogy...

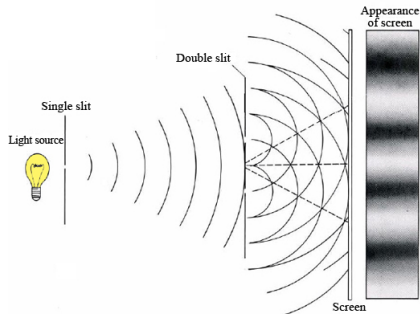
A kétrés kísérlet: hullám, mérés, részecske



Hullámtermészet? Csakhogy...

- Egyetlen foton is interferál.

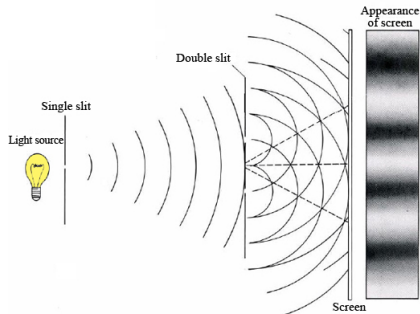
A kétrés kísérlet: hullám, mérés, részecske



Hullámtermészet? Csakhogy...

- Egyetlen foton is interferál.
- Ha detektort teszünk valamelyik részbe, az interferencia megszűnik. (Vagy ha lecsukjuk valamelyik rést.)

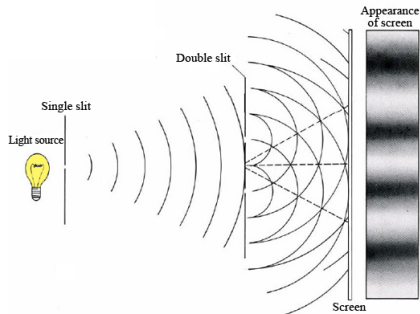
A kétrés kísérlet: hullám, mérés, részecske



Hullámtermészet? Csakhogy...

- **Egyetlen** foton is interferál.
- Ha **detektort** teszünk valamelyik részbe, az interferencia megszűnik. (Vagy ha lecsukjuk valamelyik rést.)
- Ha **egyszerre** csak egy rést hagyunk nyitva és a becsapódási pozícióból nem számítható ki, hogy melyik résen „kellett” áthaladjon, akkor

A kétrés kísérlet: hullám, mérés, részecske



Hullámtermészet? Csakhogy...

- **Egyetlen** foton is interferál.
- Ha **detektort** teszünk valamelyik részbe, az interferencia megszűnik. (Vagy ha lecsukjuk valamelyik rést.)
- Ha **egyszerre** csak egy rést hagyunk nyitva és a becsapódási pozícióból nem számítható ki, hogy melyik résen „kellett” áthaladjon, akkor **interferál**.

Akkor...

Akkor...

- Egy foton (elektron, macska...) egyszerre több lehetséges állapot **szuperpozíciójában** van...

Akkor...

- Egy foton (elektron, macska...) egyszerre több lehetséges állapot **szuperpozíciójában** van...
 - „melyik rész felé ment”

Akkor...

- Egy foton (elektron, macska...) egyszerre több lehetséges állapot **szuperpozíciójában** van...
 - „melyik rész felé ment”
 - „életben van vagy elpusztult”

Akkor...

- Egy foton (elektron, macska...) egyszerre több lehetséges állapot **szuperpozíciójában** van...
 - „melyik rész felé ment”
 - „életben van vagy elpusztult”
- ...amíg meg nem mérjük (ekkor **összeomlik** a hullámfüggvény és beáll **egy** állapotba).

Akkor...

- Egy foton (elektron, macska...) egyszerre több lehetséges állapot **szuperpozíciójában** van...
 - „melyik rész felé ment”
 - „életben van vagy elpusztult”
- ... amíg meg nem mérjük (ekkor **összeomlik** a hullámfüggvény és beáll **egy** állapotba).
 - ha a részhez teszünk detektort, ott omlik össze és nem lesz interferencia

Akkor...

- Egy foton (elektron, macska...) egyszerre több lehetséges állapot **szuperpozíciójában** van...
 - „melyik rész felé ment”
 - „életben van vagy elpusztult”
- ... amíg meg nem mérjük (ekkor **összeomlik** a hullámfüggvény és beáll **egy** állapotba).
 - ha a részhez teszünk detektort, ott omlik össze és nem lesz interferencia
 - ha az ernyőhöz, akkor csak ott és lesz interferencia

Kvantum összefonódás (entanglement)

Amit lehet tenni

Kvantum összefonódás (entanglement)

Amit lehet tenni

- Fotonokat gyártani vízszintes vagy függőleges polarizáltsággal

Kvantum összefonódás (entanglement)

Amit lehet tenni

- Fotonokat gyártani vízszintes vagy függőleges polarizáltsággal
- Megmérve ezeket 50-50%-ban gyártjuk

Kvantum összefonódás (entanglement)

Amit lehet tenni

- Fotonokat gyártani vízszintes vagy függőleges polarizáltsággal
- Megmérve ezeket 50-50%-ban gyártjuk
- Egy fotonból **két összefonódottat** gyártani (prizmával)

Kvantum összefonódás (entanglement)

Amit lehet tenni

- Fotonokat gyártani vízszintes vagy függőleges polarizáltsággal
- Megmérve ezeket 50-50%-ban gyártjuk
- Egy fotonból **két összefonódottat** gyártani (prizmával)
- Összefonódottak: ha megmérjük mindkettő polarizáltságát, **ugyanazt** kapjuk

Kvantum összefonódás (entanglement)

Amit lehet tenni

- Fotonokat gyártani vízszintes vagy függőleges polarizáltsággal
- Megmérve ezeket 50-50%-ban gyártjuk
- Egy fotonból **két összefonódottat** gyártani (prizmával)
- Összefonódottak: ha megmérjük mindkettő polarizáltságát, **ugyanazt** kapjuk
- Akkor is, ha a két mérés közt annyi idő telik el, ami a fénysebességnek is kevés, hogy „átadja” az információt

Einstein

„Spooky action at a distance” – ilyen nem lehet
Akkor a polarizáltság már **benne van** a részecskében, „rejtett” módon

Kvantum összefonódás (entanglement)

Bell

Kvantum összefonódás (entanglement)

Bell

- Engedjük át mindkét fotont egy-egy $\Theta = 45^\circ$ -os **polarizátoron** és nézzük meg, ugyanaz történik-e velük

Kvantum összefonódás (entanglement)

Bell

- Engedjük át mindkét fotont egy-egy $\Theta = 45^\circ$ -os **polarizátoron** és nézzük meg, ugyanaz történik-e velük
- Tudjuk: ha a polarizátor szöge Θ az eredeti polarizáláshoz képest, azt a polarizátor $\cos^2 \Theta = 0.5$ részben engedi át

Kvantum összefonódás (entanglement)

Bell

- Engedjük át mindkét fotont egy-egy $\Theta = 45^\circ$ -os **polarizátoron** és nézzük meg, ugyanaz történik-e velük
- Tudjuk: ha a polarizátor szöge Θ az eredeti polarizáláshoz képest, azt a polarizátor $\cos^2 \Theta = 0.5$ részben engedi át
- ... tehát a 0° -ost is és a 90° -ost is 0.5 részben engedné át mindkét oldal, függetlenül, akkor 0.5 kellene legyen azon részecske-párok aránya, melyekkel ugyanaz történik

Kvantum összefonódás (entanglement)

Bell

- Engedjük át mindkét fotont egy-egy $\Theta = 45^\circ$ -os **polarizátoron** és nézzük meg, ugyanaz történik-e velük
- Tudjuk: ha a polarizátor szöge Θ az eredeti polarizáláshoz képest, azt a polarizátor $\cos^2 \Theta = 0.5$ részben engedi át
- ... tehát a 0° -ost is és a 90° -ost is 0.5 részben engedné át mindkét oldal, függetlenül, akkor 0.5 kellene legyen azon részecske-párok aránya, melyekkel ugyanaz történik

DE NEM ANNYI

Kvantum összefonódás (entanglement)

Bell

- Engedjük át mindkét fotont egy-egy $\Theta = 45^\circ$ -os **polarizátoron** és nézzük meg, ugyanaz történik-e velük
- Tudjuk: ha a polarizátor szöge Θ az eredeti polarizáláshoz képest, azt a polarizátor $\cos^2 \Theta = 0.5$ részben engedi át
- ... tehát a 0° -ost is és a 90° -ost is 0.5 részben engedné át mindkét oldal, függetlenül, akkor 0.5 kellene legyen azon részecske-párok aránya, melyekkel ugyanaz történik

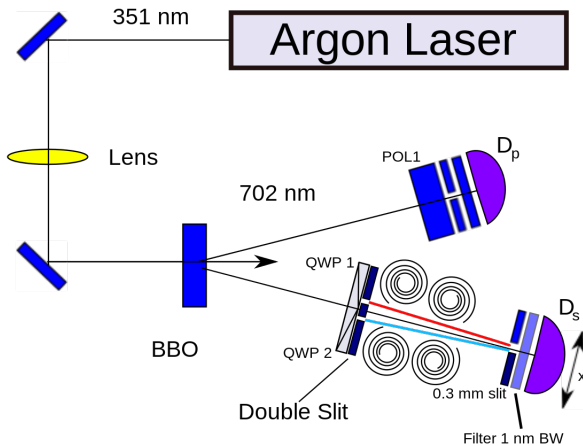
DE NEM ANNYI
hanem **minddel ugyanaz történik.**

Kvantum összefonódás (entanglement)

Bell

- Engedjük át mindkét fotont egy-egy $\Theta = 45^\circ$ -os **polarizátoron** és nézzük meg, ugyanaz történik-e velük
- Tudjuk: ha a polarizátor szöge Θ az eredeti polarizáláshoz képest, azt a polarizátor $\cos^2 \Theta = 0.5$ részben engedi át
- ... tehát a 0° -ost is és a 90° -ost is 0.5 részben engedné át mindkét oldal, függetlenül, akkor 0.5 kellene legyen azon részecske-párok aránya, melyekkel ugyanaz történik

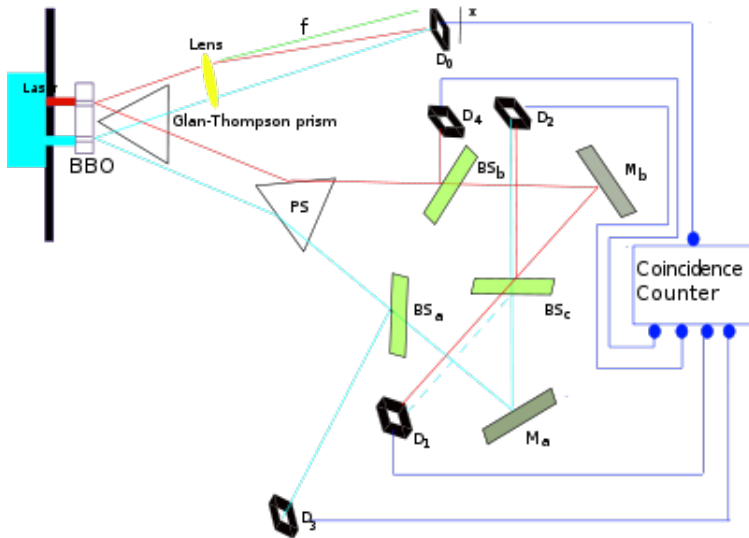
DE NEM ANNYI
hanem **minddel ugyanaz történik**. Spooky.



Walborn, Cunha, Padua, and Monken, Double slit quantum eraser experiment with polarizer (POL1) present. In this configuration, interference is detected at D_s.

Delayed choice quantum eraser, 2000

Delayed choice quantum eraser, 2000



A kísérletek szerint

- A rendszer az összes lehetséges állapot egy **szuperpozíciójában van**...
- ...amíg rá nem mérünk...
- ...akkor össze is omlik a hullámfüggvény és „beáll” a mérés eredményére...
- ...egyszerre az Univerzum összes összefonódott kvantumjában
- ...kivéve, ha a mérés eredményét garantáltan „kiradírozzuk”, akkor nem is omlik össze.

A kísérletek szerint

- A rendszer az összes lehetséges állapot egy **szuperpozíciójában van**...
- ...amíg rá nem mérünk...
- ...akkor össze is omlik a hullámfüggvény és „beáll” a mérés eredményére...
- ...egyszerre az Univerzum összes összefonódott kvantumjában
- ...kivéve, ha a mérés eredményét garantáltan „kiradírozzuk”, akkor nem is omlik össze.

Érteni ugyan nem értjük, **miért**,

A kísérletek szerint

- A rendszer az összes lehetséges állapot egy **szuperpozíciójában van**...
- ...amíg rá nem mérünk...
- ...akkor össze is omlik a hullámfüggvény és „beáll” a mérés eredményére...
- ...egyszerre az Univerzum összes összefonódott kvantumjában
- ...kivéve, ha a mérés eredményét garantáltan „kiradírozzuk”, akkor nem is omlik össze.

Érteni ugyan nem értjük, **miért**, de elég érdekesnek látszik ahhoz, hogy jó ötletnek tűnjön **számítógépet építeni** belőle.

Építeni?

Építeni?

- Eddig 7 – 10 **qubites** rendszereket sikerült minden kutatócsoportnak építenie...

Építeni?

- Eddig 7 – 10 **qubites** rendszereket sikerült minden kutatócsoportnak építenie...
- ... kivéve a kanadai D-Wave Systemst,

Építeni?

- Eddig 7 – 10 **qubites** rendszereket sikerült minden kutatócsoportnak építenie...
- ... kivéve a kanadai D-Wave Systemst,
- akik eladtak a Google-nak és a NASA-nak egy 512-qubites rendszert 2013-ban, miután a Lockheed Martinnak egy 128-qubitest 2011-ben

Építeni?

- Eddig 7 – 10 **qubites** rendszereket sikerült minden kutatócsoportnak építenie...
- ... kivéve a kanadai D-Wave Systemst,
- akik eladtak a Google-nak és a NASA-nak egy 512-qubites rendszert 2013-ban, miután a Lockheed Martinnak egy 128-qubitest 2011-ben
- amely rendszerekről „jelenleg nem világos, hogy i) történik-e bennük kvantum-jellegű számítás és ii) ha igen, attól gyorsabb-e a rendszer egyáltalán, mint egy laptop

Építeni?

- Eddig 7 – 10 **qubites** rendszereket sikerült minden kutatócsoportnak építenie...
- ... kivéve a kanadai D-Wave Systemst,
- akik eladtak a Google-nak és a NASA-nak egy 512-qubites rendszert 2013-ban, miután a Lockheed Martinnak egy 128-qubitest 2011-ben
- amely rendszerekről „jelenleg nem világos, hogy i) történik-e bennük kvantum-jellegű számítás és ii) ha igen, attól gyorsabb-e a rendszer egyáltalán, mint egy laptop

Lássuk a matekot.

Valószínűségi amplitúdók

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Valószínűségi amplitúdók

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Szuperpozíció

Ehelyett a 0 állapothoz egy α_0 , az 1 állapothoz egy α_1 **amplitúdót** rendelünk:

Valószínűségi amplitúdók

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Szuperpozíció

Ehelyett a 0 állapothoz egy α_0 , az 1 állapothoz egy α_1 **amplitúdót** rendelünk:

- α_0, α_1 komplex számok

Valószínűségi amplitúdók

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Szuperpozíció

Ehelyett a 0 állapothoz egy α_0 , az 1 állapothoz egy α_1 **amplitúdót** rendelünk:

- α_0, α_1 komplex számok
- **mérés**kor 0-t $|\alpha_0|^2$, 1-et pedig $|\alpha_1|^2$ valószínűséggel kapunk

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Szuperpozíció

Ehelyett a 0 állapothoz egy α_0 , az 1 állapothoz egy α_1 **amplitúdót** rendelünk:

- α_0, α_1 komplex számok
- **mérés**kor 0-t $|\alpha_0|^2$, 1-et pedig $|\alpha_1|^2$ valószínűséggel kapunk
- ennek megfelelően $|\alpha_0|^2 + |\alpha_1|^2 = 1$ kell legyen

Valószínűségi amplitúdók

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Szuperpozíció

Ehelyett a 0 állapothoz egy α_0 , az 1 állapothoz egy α_1 **amplitúdót** rendelünk:

- α_0, α_1 komplex számok
- **méréskor** 0-t $|\alpha_0|^2$, 1-et pedig $|\alpha_1|^2$ valószínűséggel kapunk
- ennek megfelelően $|\alpha_0|^2 + |\alpha_1|^2 = 1$ kell legyen
- mérés során a hullámfüggvény **összeomlik**: $\alpha_i = 1$ lesz, ha a mérés eredménye i (α_{1-i} pedig 0)

Valószínűségi amplitúdók

A hullámtermészetből eredő **fázis** modellezéséhez kevés, ha a 0 állapothoz egy p_0 , az 1 állapothoz egy p_1 **valószínűséget** rendelünk.

Szuperpozíció

Ehelyett a 0 állapothoz egy α_0 , az 1 állapothoz egy α_1 **amplitúdót** rendelünk:

- α_0, α_1 komplex számok
- **méréskor** 0-t $|\alpha_0|^2$, 1-et pedig $|\alpha_1|^2$ valószínűséggel kapunk
- ennek megfelelően $|\alpha_0|^2 + |\alpha_1|^2 = 1$ kell legyen
- mérés során a hullámfüggvény **összeomlik**: $\alpha_i = 1$ lesz, ha a mérés eredménye i (α_{1-i} pedig 0)

Technikai jelölés: $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ (ez a (α_0, α_1) értékű **qubit**)

Több qubites rendszer állapotai

Több qubites rendszer állapotai

Ha a rendszerben **két** qubit van, akkor egy mérés eredménye lehet $|00\rangle$, $|01\rangle$, $|10\rangle$ vagy $|11\rangle$

Több qubites rendszer állapotai

Ha a rendszerben **két** qubit van, akkor egy mérés eredménye lehet $|00\rangle$, $|01\rangle$, $|10\rangle$ vagy $|11\rangle$

Összefonódás!

(Általában) nem kezelhetjük függetlennek az egyes qubitek amplitúdóit

Több qubites rendszer állapotai

Ha a rendszerben **két** qubit van, akkor egy mérés eredménye lehet $|00\rangle$, $|01\rangle$, $|10\rangle$ vagy $|11\rangle$

Összefonódás!

(Általában) nem kezelhetjük függetlennek az egyes qubitek amplitúdóit
Pl. ha a két qubit **összefonódott**, akkor a mérés eredménye vagy $|00\rangle$, vagy $|11\rangle$ lehet csak

Megoldás

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

Megoldás

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

- $\sum_{u \in \mathbf{2}^2} |\alpha_u|^2 = 1$

Megoldás

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

- $\sum_{u \in \mathbf{2}^2} |\alpha_u|^2 = 1$
- $|u\rangle$ mérésének valószínűsége $|\alpha_u|^2$

Megoldás

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

- $\sum_{u \in \mathbf{2}^2} |\alpha_u|^2 = 1$
- $|u\rangle$ mérésének valószínűsége $|\alpha_u|^2$
- $|u\rangle$ -t mérve a rendszer ténylegesen beáll az $|u\rangle$ állapotba

Több qubites rendszer állapotai

Megoldás

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

- $\sum_{u \in 2^2} |\alpha_u|^2 = 1$
- $|u\rangle$ mérésének valószínűsége $|\alpha_u|^2$
- $|u\rangle$ -t mérve a rendszer ténylegesen beáll az $|u\rangle$ állapotba

Még több qubit

Ha a qubitek száma n , az egy **2^n -tagú** formális összeg

$$|\psi\rangle = \sum_{u \in 2^n} \alpha_u |u\rangle, \quad \sum |\alpha_u|^2 = 1;$$

Több qubites rendszer állapotai

Megoldás

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

- $\sum_{u \in 2^2} |\alpha_u|^2 = 1$
- $|u\rangle$ mérésének valószínűsége $|\alpha_u|^2$
- $|u\rangle$ -t mérve a rendszer ténylegesen beáll az $|u\rangle$ állapotba

Még több qubit

Ha a qubitek száma n , az egy **2^n -tagú** formális összeg

$$|\psi\rangle = \sum_{u \in 2^n} \alpha_u |u\rangle, \quad \sum |\alpha_u|^2 = 1;$$

ez 500 qubit esetén 2^{500} darab komplex szám!

A $|0\rangle = (1, 0)$ és $|1\rangle = (0, 1)$ azonosítással a $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ állapot épp a $|\psi\rangle = (\alpha_0, \alpha_1)$ alakot ölti

Tenzor szorzat

A $|0\rangle = (1, 0)$ és $|1\rangle = (0, 1)$ azonosítással a $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ állapot épp a $|\psi\rangle = (\alpha_0, \alpha_1)$ alakot ölti

Továbbmenve, $|00\rangle = (1, 0, 0, 0)$, $|01\rangle = (0, 1, 0, 0)$, $|10\rangle = (0, 0, 1, 0)$ és $|11\rangle = (0, 0, 0, 1)$ adná, hogy

$$\sum_{u \in \mathbf{2}^2} \alpha_u |u\rangle = (\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}).$$

Tenzor szorzat

A $|0\rangle = (1, 0)$ és $|1\rangle = (0, 1)$ azonosítással a $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ állapot épp a $|\psi\rangle = (\alpha_0, \alpha_1)$ alakot ölti

Továbbmenve, $|00\rangle = (1, 0, 0, 0)$, $|01\rangle = (0, 1, 0, 0)$, $|10\rangle = (0, 0, 1, 0)$ és $|11\rangle = (0, 0, 0, 1)$ adná, hogy

$$\sum_{u \in 2^2} \alpha_u |u\rangle = (\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}).$$

Vektorok tenzor szorzata

Legyen $u = (a_1, \dots, a_n)$ és $v = (b_1, \dots, b_k)$. Akkor

$$u \otimes v = (a_1 b_1, a_1 b_2, \dots, a_1 b_k, a_2 b_1, \dots, a_n b_k)$$

a két vektor **tenzor** szorzata.

(mint $(a_1 v, a_2 v, \dots, a_n v)$, „elhagyva a zárójeleket”)

Miért jó a tenzor szorzat

Miért jó a tenzor szorzat

Asszociatív

$$u \otimes (v \otimes w) = (u \otimes v) \otimes w$$

Miért jó a tenzor szorzat

Asszociatív

$$u \otimes (v \otimes w) = (u \otimes v) \otimes w$$

Konzisztens a ket-jelöléssel

$$|u\rangle \otimes |v\rangle = |uv\rangle, \text{ ha } u \in \mathbf{2}^n, v \in \mathbf{2}^k$$

Miért jó a tenzor szorzat

Asszociatív

$$u \otimes (v \otimes w) = (u \otimes v) \otimes w$$

Konzisztens a ket-jelöléssel

$$|u\rangle \otimes |v\rangle = |uv\rangle, \text{ ha } u \in \mathbf{2}^n, v \in \mathbf{2}^k$$

Szuperpozíciók tenzor-szorzata szuperpozíció

Ha $|\psi\rangle = \sum_{u \in \mathbf{2}^n} \alpha_u |u\rangle$ és $|\varphi\rangle = \sum_{v \in \mathbf{2}^k} \beta_v |v\rangle$ úgy, hogy $\sum |\alpha_u|^2 = \sum |\beta_v|^2 = 1$, akkor

$$|\psi\rangle \otimes |\varphi\rangle = \sum_{uv \in \mathbf{2}^{n+k}} \alpha_u \beta_v |uv\rangle,$$

$$\text{és } \sum |\alpha_u \beta_v|^2 = 1.$$

Unitér transzformációk

Unitér mátrixok

Egy A négyzetes mátrix **unitér**, ha a konjugált transzponáltja az inverze

Unitér transzformációk

Unitér mátrixok

Egy A négyzetes mátrix **unitér**, ha a konjugált transzponáltja az inverze

Példák

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Unitér transzformációk

Unitér mátrixok

Egy A négyzetes mátrix **unitér**, ha a konjugált transzponáltja az inverze

Példák

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Ezek éppen valós mátrixok, ekkor unitér = ortonormált.

Miért?

A kvantummechanika második posztulátuma

Zárt kvantumrendszer fejlődése unitér transzformációval írható le:

A kvantummechanika második posztulátuma

Zárt kvantumrendszer fejlődése unitér transzformációval írható le:

Tetszőleges t_1, t_2 időpontokhoz létezik egy U unitér operátor, melyre ha a rendszer t_1 -ben a $|\psi\rangle$ állapotban van, akkor t_2 -ben $U|\psi\rangle$ -ben.

Miért?

A kvantummechanika második posztulátuma

Zárt kvantumrendszer fejlődése unitér transzformációval írható le:

Tetszőleges t_1 , t_2 időpontokhoz létezik egy U unitér operátor, melyre ha a rendszer t_1 -ben a $|\psi\rangle$ állapotban van, akkor t_2 -ben $U|\psi\rangle$ -ben.

Avagy: a Schrödinger-egyenlet

Zárt kvantumrendszer fejlődése unitér transzformációval írható le: létezik **egy** H Hermite-mátrix, amire

$$i\hbar \frac{d|\psi\rangle}{dt} = H|\psi\rangle.$$

Bitváltás, fázisváltás

$X|0\rangle, X|1\rangle$ – bitváltás

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

és hasonlóan, $X|1\rangle = |0\rangle$.

$X|0\rangle, X|1\rangle$ – bitváltás

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

és hasonlóan, $X|1\rangle = |0\rangle$.

Általában $X(\alpha_0, \alpha_1) = (\alpha_1, \alpha_0)$.

$X|0\rangle, X|1\rangle$ – bitváltás

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

és hasonlóan, $X|1\rangle = |0\rangle$.

Általában $X(\alpha_0, \alpha_1) = (\alpha_1, \alpha_0)$.

$Z|0\rangle, Z|1\rangle$ – fázisváltás

$$Z|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle,$$

de

$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle!$$

$X|0\rangle, X|1\rangle$ – bitváltás

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

és hasonlóan, $X|1\rangle = |0\rangle$.

Általában $X(\alpha_0, \alpha_1) = (\alpha_1, \alpha_0)$.

$Z|0\rangle, Z|1\rangle$ – fázisváltás

$$Z|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle,$$

de

$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle!$$

Általában $Z(\alpha_0, \alpha_1) = (\alpha_0, -\alpha_1)$.

Hadamard keverés – szuperpozíció létrehozása

Hadamard keverés – szuperpozíció létrehozása

$H|0\rangle$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}.$$

Hadamard keverés – szuperpozíció létrehozása

$H|0\rangle$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}.$$

Tehát egy qubitet $|0\rangle$ -ra inicializálva, majd a H Hadamard-transzformációt alkalmazva rajta az $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ állapotba kerül a qubit; ezt 50 – 50%-kal mérjük $|0\rangle$ -nak ill. $|1\rangle$ -nek

Klasszikus áramkörök kvantumgépen

Klasszikus áramkörök kvantumgépen

Klasszikus, $f : 2^n \rightarrow 2$ (pl. logikai áramkörökkel megvalósított) függvényeket is szeretnénk kvantumszámításunk során használni (pl. mikor a feladat épp az, hogy f egy zérushelyét keressük)

Klasszikus áramkörök kvantumgépen

Klasszikus, $f : 2^n \rightarrow 2$ (pl. logikai áramkörökkel megvalósított) függvényeket is szeretnénk kvantumszámításunk során használni (pl. mikor a feladat épp az, hogy f egy zérushelyét keressük)

A probléma

A klasszikus kapuk **nem mind invertálhatóak!** Az unitér transzformációk viszont igen.

Klasszikus áramkörök kvantumgépen

Klasszikus, $f : 2^n \rightarrow 2$ (pl. logikai áramkörökkel megvalósított) függvényeket is szeretnénk kvantumszámításunk során használni (pl. mikor a feladat épp az, hogy f egy zérushelyét keressük)

A probléma

A klasszikus kapuk **nem mind invertálhatóak!** Az unitér transzformációk viszont igen.

A megoldás

Kiegészítő qubiteket alkalmazunk: a T Toffoli kapura (mátrixra) igaz, hogy

$$T|a\rangle|b\rangle|c\rangle = |a\rangle|b\rangle|c \text{ xor } (a \wedge b)\rangle.$$

Klasszikus áramkörök kvantumgépen

Klasszikus, $f : 2^n \rightarrow 2$ (pl. logikai áramkörökkel megvalósított) függvényeket is szeretnénk kvantumszámításunk során használni (pl. mikor a feladat épp az, hogy f egy zérushelyét keressük)

A probléma

A klasszikus kapuk **nem mind invertálhatóak!** Az unitér transzformációk viszont igen.

A megoldás

Kiegészítő qubiteket alkalmazunk: a T Toffoli kapura (mátrixra) igaz, hogy

$$T|a\rangle|b\rangle|c\rangle = |a\rangle|b\rangle|c \text{ xor } (a \wedge b)\rangle.$$

Ha a nand b -t akarjuk kiszámítani, akkor a c qubitet $|1\rangle$ -gyel inicializálva, majd $|abc\rangle$ -t T -vel transzformálva c éppen az a nand b állapotba kerül.

Klasszikus áramkörök kvantumgépen

Klasszikus, $f : 2^n \rightarrow 2$ (pl. logikai áramkörökkel megvalósított) függvényeket is szeretnénk kvantumszámításunk során használni (pl. mikor a feladat épp az, hogy f egy zérushelyét keressük)

A probléma

A klasszikus kapuk **nem mind invertálhatóak!** Az unitér transzformációk viszont igen.

A megoldás

Kiegészítő qubiteket alkalmazunk: a T Toffoli kapura (mátrixra) igaz, hogy

$$T|a\rangle|b\rangle|c\rangle = |a\rangle|b\rangle|c \text{ xor } (a \wedge b)\rangle.$$

Ha a nand b -t akarjuk kiszámítani, akkor a c qubitet $|1\rangle$ -gyel inicializálva, majd $|abc\rangle$ -t T -vel transzformálva c éppen az a nand b állapotba kerül.

Klasszikus áramkörök kvantumgépen

Klasszikus áramkörök kvantumgépen

Nincs kész!

A hagyományos áramkörökben nem gond az output **másolása**.

Klasszikus áramkörök kvantumgépen

Nincs kész!

A hagyományos áramkörökben nem gond az output **másolása**.

A no-cloning tétel

Nincs olyan U unitér mátrix és $|\varphi\rangle$ állapot, melyre
$$U(|\psi\rangle|\varphi\rangle) = |\psi\rangle|\psi\rangle$$
teljesülne minden $|\psi\rangle$ állapotra.

Klasszikus áramkörök kvantumgépen

Nincs kész!

A hagyományos áramkörökben nem gond az output **másolása**.

A no-cloning tétel

Nincs olyan U unitér mátrix és $|\varphi\rangle$ állapot, melyre
$$U(|\psi\rangle|\varphi\rangle) = |\psi\rangle|\psi\rangle$$
teljesülne minden $|\psi\rangle$ állapotra.

Klasszikus eset

Klasszikus biteket tudunk Toffoli kapuval másolni:

$$T|1\rangle|a\rangle|0\rangle = |1\rangle|a\rangle|a\rangle.$$

A Toffoli-kapuhoz hasonló elven tetszőleges $f : \mathbf{2}^n \rightarrow \mathbf{2}$ függvény szimulálható egy

$$U_f : |b_1 \dots b_n\rangle |c\rangle \mapsto |b_1 \dots b_n\rangle |c \text{ xor } f(b_1, \dots, b_n)\rangle$$

kvantum áramkörrel.

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés

Hadamard-keverés

- Inicializáljuk az n -qubites rendszerünket a $|000\dots 0\rangle$ állapotban.

Hadamard-keverés

- Inicializáljuk az n -qubites rendszerünket a $|000\dots 0\rangle$ állapotban.
- Minden qubiten hajtsuk végre a H Hadamard-transzformációt.

Hadamard-keverés

- Inicializáljuk az n -qubites rendszerünket a $|000\dots 0\rangle$ állapotban.
- Minden qubiten hajtsuk végre a H Hadamard-transzformációt.

Ekkor a kapott állapot:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \dots \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$

vagyis $\frac{1}{\sqrt{2^n}} \sum_{u \in \mathbf{2}^n} |u\rangle$, tökéletes szuperpozíció.

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés

- Inicializáljuk az n -qubites rendszerünket a $|000\dots 0\rangle$ állapotban.
- Minden qubiten hajtsuk végre a H Hadamard-transzformációt.

Ekkor a kapott állapot:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \dots \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$

vagyis $\frac{1}{\sqrt{2^n}} \sum_{u \in 2^n} |u\rangle$, tökéletes szuperpozíció.

Mérés?

Ha most mérnénk, minden u állapot azonos valószínűséggel állna be.

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés

- Inicializáljuk az n -qubites rendszerünket a $|000\dots 0\rangle$ állapotban.
- Minden qubiten hajtsuk végre a H Hadamard-transzformációt.

Ekkor a kapott állapot:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \dots \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$

vagyis $\frac{1}{\sqrt{2^n}} \sum_{u \in 2^n} |u\rangle$, tökéletes szuperpozíció.

Mérés?

Ha most mérnénk, minden u állapot azonos valószínűséggel állna be.
Ez nem túl hasznos. (Ugyanaz, mint a random beállítás.)

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés + U_f

Alkalmazzuk a Hadamard-keverés után az U_f áramkört a rendszeren, a $|00 \dots 0\rangle$ állapotból indítva?

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés + U_f

Alkalmazzuk a Hadamard-keverés után az U_f áramkört a rendszeren, a $|00 \dots 0\rangle$ állapotból indítva?

Az eredmény ...

$$\frac{1}{\sqrt{2^n}} \sum_{u \in 2^n} |u\rangle |f(u)\rangle.$$

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés + U_f

Alkalmazzuk a Hadamard-keverés után az U_f áramkört a rendszeren, a $|00 \dots 0\rangle$ állapotból indítva?

Az eredmény ...

$$\frac{1}{\sqrt{2}^n} \sum_{u \in 2^n} |u\rangle |f(u)\rangle.$$

Mérés?

Ha most mérnénk, egy $|u\rangle |f(u)\rangle$ alakú állapotba állna be a rendszer, minden u -ra azonos valószínűséggel.

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés + U_f

Alkalmazzuk a Hadamard-keverés után az U_f áramkört a rendszeren, a $|00 \dots 0\rangle$ állapotból indítva?

Az eredmény ...

$$\frac{1}{\sqrt{2}^n} \sum_{u \in 2^n} |u\rangle |f(u)\rangle.$$

Mérés?

Ha most mérnénk, egy $|u\rangle |f(u)\rangle$ alakú állapotba állna be a rendszer, minden u -ra azonos valószínűséggel.

Függvénykiszámítás egyszerre minden lehetséges inputra

Hadamard-keverés + U_f

Alkalmazzuk a Hadamard-keverés után az U_f áramkört a rendszeren, a $|00 \dots 0\rangle$ állapotból indítva?

Az eredmény ...

$$\frac{1}{\sqrt{2}^n} \sum_{u \in 2^n} |u\rangle |f(u)\rangle.$$

Mérés?

Ha most mérnénk, egy $|u\rangle |f(u)\rangle$ alakú állapotba állna be a rendszer, minden u -ra azonos valószínűséggel.

Ez sem túl hasznos. (Ugyanaz, mint egy random helyen kiértékelés.)

A „jó” függvényértékek fázisának változtatása

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

Felfoghatjuk úgy is, hogy ha $f(u) = 0$, akkor c nem változik, ha $f(u) = 1$, akkor bitváltás történik.

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

Felfoghatjuk úgy is, hogy **ha** $f(u) = 0$, akkor c nem változik, ha $f(u) = 1$, akkor bitváltás történik.

Inicializálva az U_f kimeneti qubitjét $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ -vel...

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

Felfoghatjuk úgy is, hogy ha $f(u) = 0$, akkor c nem változik, ha $f(u) = 1$, akkor bitváltás történik.

Inicializálva az U_f kimeneti qubitjét $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ -vel...

- ha $f(u) = 0$: marad $H|1\rangle$

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

Felfoghatjuk úgy is, hogy **ha** $f(u) = 0$, akkor c nem változik, ha $f(u) = 1$, akkor bitváltás történik.

Inicializálva az U_f kimeneti qubitjét $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ -vel...

- ha $f(u) = 0$: marad $H|1\rangle$
- ha $f(u) = 1$: $-H|1\rangle$ lesz

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

Felfoghatjuk úgy is, hogy **ha** $f(u) = 0$, akkor c nem változik, ha $f(u) = 1$, akkor bitváltás történik.

Inicializálva az U_f kimeneti qubitjét $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ -vel...

- ha $f(u) = 0$: marad $H|1\rangle$
- ha $f(u) = 1$: **$-H|1\rangle$ lesz**

Azaz:

$$|u\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \mapsto |u\rangle (-1)^{f(u)} \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

A „jó” függvényértékek fázisának változtatása

U_f , ismét

$$U_f(|u\rangle|c\rangle) = |u\rangle|c + f(u)\rangle.$$

Felfoghatjuk úgy is, hogy **ha** $f(u) = 0$, akkor c nem változik, ha $f(u) = 1$, akkor bitváltás történik.

Inicializálva az U_f kimeneti qubitjét $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ -vel...

- ha $f(u) = 0$: marad $H|1\rangle$
- ha $f(u) = 1$: $-H|1\rangle$ lesz

Azaz:

$$|u\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \mapsto |u\rangle (-1)^{f(u)} \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

A továbbiakban a $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ konstanst elhagyjuk:

$$|u\rangle \mapsto (-1)^{f(u)} |u\rangle.$$

$|0 \dots 0\rangle$ -ra tükrözés...

Vegyük a következő (unitér) operátort:

$$P(|u\rangle) = \begin{cases} |u\rangle & , \text{ ha } u = 0 \dots 0; \\ -|u\rangle & , \text{ egyébként.} \end{cases}$$

(Az operátor megvalósítható $O(n)$ kvantum kapuval.)

$|0 \dots 0\rangle$ -ra tükrözés...

Vegyük a következő (unitér) operátort:

$$P(|u\rangle) = \begin{cases} |u\rangle & , \text{ ha } u = 0 \dots 0; \\ -|u\rangle & , \text{ egyébként.} \end{cases}$$

(Az operátor megvalósítható $O(n)$ kvantum kapuval.)

... kétoldali Hadamard-keveréssel

Ha a $D = HPH$ transzformációt tekintjük, kiírással igazolható, hogy

$$D(\alpha_1, \dots, \alpha_{2^n}) = (2\mu - \alpha_1, 2\mu - \alpha_2, \dots, 2\mu - \alpha_{2^n}),$$

ahol $\mu = \sum \frac{\alpha_i}{2^n}$ az amplitúdók **átlaga**.

$|0 \dots 0\rangle$ -ra tükrözés...

Vegyük a következő (unitér) operátort:

$$P(|u\rangle) = \begin{cases} |u\rangle & , \text{ ha } u = 0 \dots 0; \\ -|u\rangle & , \text{ egyébként.} \end{cases}$$

(Az operátor megvalósítható $O(n)$ kvantum kapuval.)

... kétoldali Hadamard-keveréssel

Ha a $D = HPH$ transzformációt tekintjük, kiírással igazolható, hogy

$$D(\alpha_1, \dots, \alpha_{2^n}) = (2\mu - \alpha_1, 2\mu - \alpha_2, \dots, 2\mu - \alpha_{2^n}),$$

ahol $\mu = \sum \frac{\alpha_i}{2^n}$ az amplitúdók **átlaga**.

Tehát D végrehajtása egy „átlag körüli inverzió”.

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

U_f egy tükrözés $|\alpha\rangle$ -ra,

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

U_f egy tükrözés $|\alpha\rangle$ -ra,
 D pedig egy tükrözés $|\psi\rangle$ -re,

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

U_f egy tükrözés $|\alpha\rangle$ -ra,

D pedig egy tükrözés $|\psi\rangle$ -re,

ezek eredménye pedig egy forgatás a két vektor síkjában!

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

U_f egy tükrözés $|\alpha\rangle$ -ra,

D pedig egy tükrözés $|\psi\rangle$ -re,

ezek eredménye pedig egy forgatás a két vektor síkjában!

- Mégpedig, ha $|\psi\rangle$ és $|\alpha\rangle$ bezárt szöge Θ , akkor 2Θ -val való forgatás.

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

U_f egy tükrözés $|\alpha\rangle$ -ra,

D pedig egy tükrözés $|\psi\rangle$ -re,

ezek eredménye pedig egy forgatás a két vektor síkjában!

- Mégpedig, ha $|\psi\rangle$ és $|\alpha\rangle$ bezárt szöge Θ , akkor 2Θ -val való forgatás.
- Ha ezt a forgatást $\approx \frac{\pi}{4} \sqrt{\frac{2^n}{M}}$ -szer ismétljük, akkor a kapott vektor $|\alpha\rangle$ -val (tehát a „rossz” vektorok alterével) bezárt szöge közel $\frac{\pi}{2}$ lesz

Előbb U_f , majd D

M darab „jó” u vektor (melyekre $f(u) = 1$) esetén

- ha $|\psi\rangle$ az uniform szuperpozíció és $|\alpha\rangle = \frac{1}{\sqrt{2^n - M}} \sum_{u \text{ rossz}} |u\rangle$ a rossz vektorok uniform szuperpozíciója, akkor:

U_f egy tükrözés $|\alpha\rangle$ -ra,

D pedig egy tükrözés $|\psi\rangle$ -re,

ezek eredménye pedig egy forgatás a két vektor síkjában!

- Mégpedig, ha $|\psi\rangle$ és $|\alpha\rangle$ bezárt szöge Θ , akkor 2Θ -val való forgatás.
- Ha ezt a forgatást $\approx \frac{\pi}{4} \sqrt{\frac{2^n}{M}}$ -szer ismétljük, akkor a kapott vektor $|\alpha\rangle$ -val (tehát a „rossz” vektorok alterével) bezárt szöge közel $\frac{\pi}{2}$ lesz
- **Ekkor** mérve tehát jó eséllyel **nem** a rossz vektorok közül kapunk egyet.

Összefoglalva: van olyan kvantumalgoritmus, mely az f függvényt csak $O(\sqrt{2^n})$ -szer meghívva, nagy valószínűséggel ad egy olyan $u \in \mathbf{2}^n$ -t, melyre $f(u) = 1$.

Összefoglalva: van olyan kvantumalgorithmus, mely az f függvényt csak $O(\sqrt{2^n})$ -szer meghívva, nagy valószínűséggel ad egy olyan $u \in 2^n$ -t, melyre $f(u) = 1$.

Ez bizonyíthatóan **jobb**, mint egy klasszikus valószínűségi algoritmus (ott $o(2^n)$ hívás nem ad konstans valószínűséggel jó eredményt, ha pl $M = 1$)

Összefoglalva: van olyan kvantumalgorithmus, mely az f függvényt csak $O(\sqrt{2^n})$ -szer meghívva, nagy valószínűséggel ad egy olyan $u \in 2^n$ -t, melyre $f(u) = 1$.

Ez bizonyíthatóan **jobb**, mint egy klasszikus valószínűségi algoritmus (ott $o(2^n)$ hívás nem ad konstans valószínűséggel jó eredményt, ha pl $M = 1$)

Ha az **összes** jó vektort meg akarjuk határozni: $O(\sqrt{M2^n})$.

Összefoglalva: van olyan kvantumalgorithmus, mely az f függvényt csak $O(\sqrt{2^n})$ -szer meghívva, nagy valószínűséggel ad egy olyan $u \in 2^n$ -t, melyre $f(u) = 1$.

Ez bizonyíthatóan **jobb**, mint egy klasszikus valószínűségi algoritmus (ott $o(2^n)$ hívás nem ad konstans valószínűséggel jó eredményt, ha pl $M = 1$)

Ha az **összes** jó vektort meg akarjuk határozni: $O(\sqrt{M2^n})$.

Ezek a korlátok (ebben a kvantumszámítási modellben) aszimptotikusan **optimálisak** is.

Közel az 1 valószínűséghez

A valószínűség javítása

ϵ megnövelése: $\log n$ számú iterált végrehajtással

Közel az 1 valószínűséghez

A valószínűség javítása

ϵ megnövelése: $\log n$ számú iterált végrehajtással

A Grover keresés egy-egy f hívása mellé még $O(\log n)$ egyéb kvantum-kapu műveletet végez

Done.