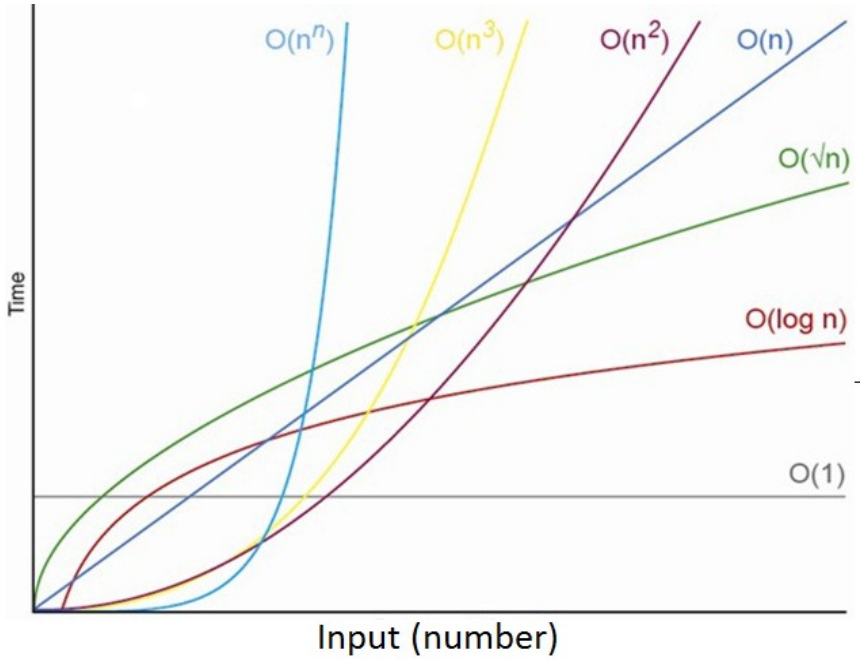
Oszd meg és uralkodj

2022.09.20

Ismétlés

Időigény

- NAGY ÖTLET: ¨ Nézzük csak azt, hogy milyen gyorsan nő a T(n), ha n → ∞.

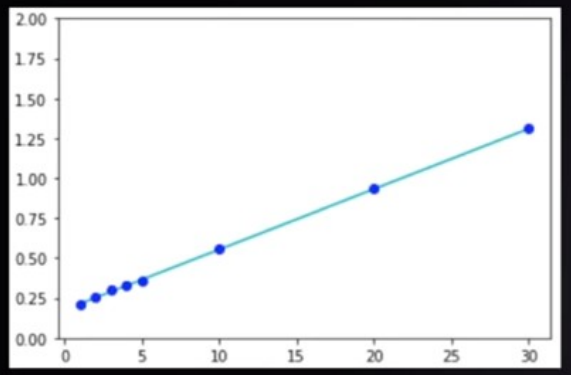


$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \cdots < 2^n < 3^n < \cdots < n^n$

Időigény

- NAGY ÖTLET: ¨ Nézzük csak azt, hogy milyen gyorsan nő a T(n), ha n → ∞.

```
given_array = [1, 4, 3, 2, ..., 10]
```



linear time $O(n)$

constant time $O(1)$

quadratic time $O(n^2)$

$T = an + b = O(n)$

$T = cn^2 + dn + e = O(n^2)$

1. find the fastest growing term
2. take out the coefficient

2. gyakorlat

- Egy adott (**rendezetlen**) tömbben elem megtalálása?

- A: O(n)
- B: O(1)
- C: O(log n)
- D: $O(n^2)$

2. gyakorlat

Egy adott (rendezetlen) tömbben elem megtalálása

```
int linearSearch(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

2. gyakorlat

n elemű **rendezett** vektorban egy elem megkeresése?

- A: O(n)
- B: O(1)
- C: O(log n)
- D: $O(n^2)$

2. gyakorlat

n elemű rendezett vektorban egy elem megkeresése rekurzívan

```
int binarySearch(int arr[], int l, int r, int x)
{
	if (r >= l) {
	    int mid = l + (r - l) / 2;
	    if (arr[mid] == x)
	        return mid;
	    if (arr[mid] > x)
	        return binarySearch(arr, l, mid - 1, x);
	     return binarySearch(arr, mid + 1, r, x);
	}
	return -1;
}
```

2. gyakorlat

n elemű rendezett vektorban egy elem megkeresése rekurzívan

Mennyi a futási idő?

Tegyük fel $n=2^k$

$T(1)=a$

$T(n)=T(n/2)+b=(T(n/4)+b)+b=T(n/4)+2b=\ldots$
$=T(1)+k\cdot b=a+\log_2 n\cdot b$

Vagyis a futási idő O(log n).

2. gyakorlat

n elemű rendezett vektorban egy elem megkeresése iteratívan

```
int binarySearch(int arr[], int l, int r, int x)
{
      while (l <= r) {
          int m = l + (r - l) / 2;
          if (arr[m] == x)
              return m;
          if (arr[m] < x)
              l = m + 1;
          else
              r = m - 1;
      }
      return -1;
}
```

2. gyakorlat

Két n elemű vektor szorzása?

- A: O(n)
- B: O(1)
- C: O(log n)
- D: $O(n^2)$

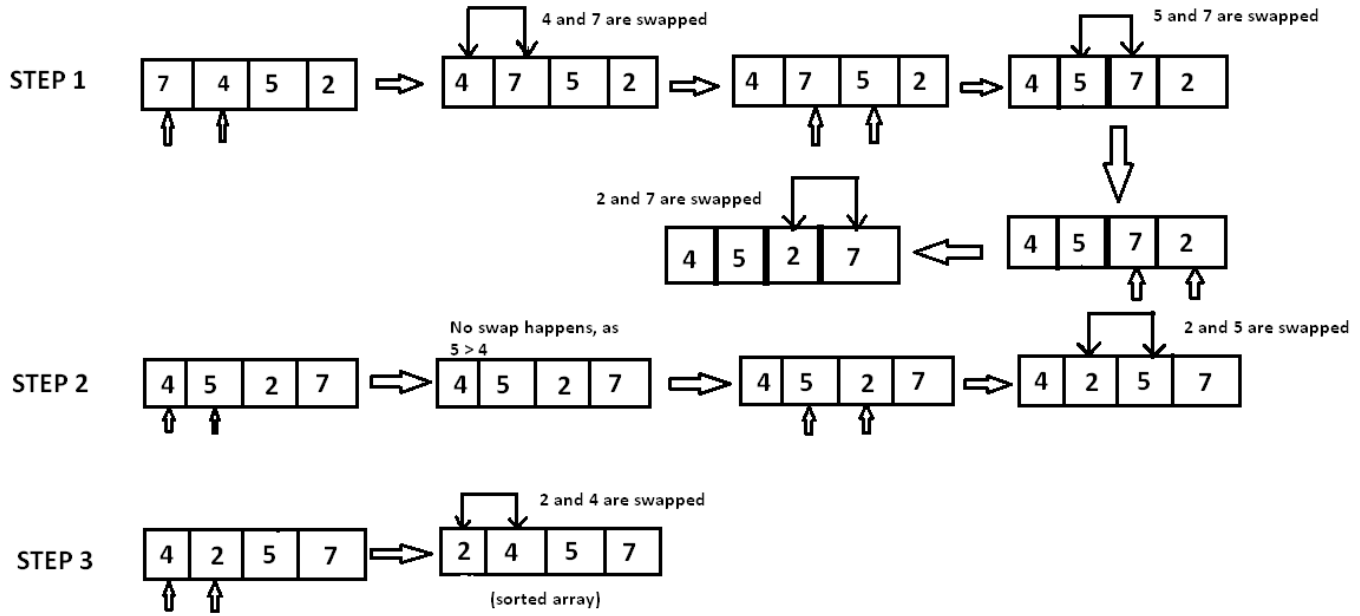
2. gyakorlat

Két n elemű vektor szorzása

```
int dotProduct(int vect_A[], int vect_B[], int n)
{
        int product = 0;
        for (int i = 0; i < n; i++)
            product = product + vect_A[i] * vect_B[i];
        return product;
}
```

Buborékos rendezés

- **Egymás melletti elemek cseréje**



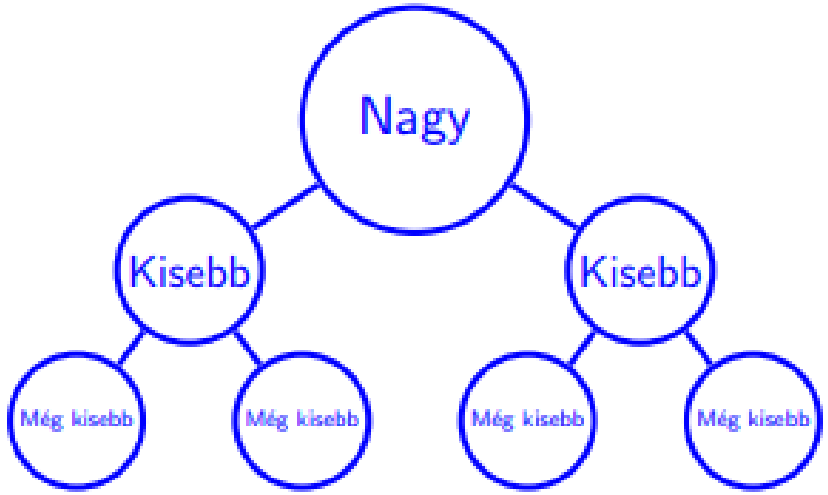
Buborékos rendezés

- Egymás melletti elemek cseréje
- (n-1) + (n-2) + (n-3) + ... + 3 + 2 + 1
- Sum = n(n-1)/2
- = $O(n^2)$ - Legrosszabb idő
- = O(n) – Legjobb idő
- = $O(n^2)$ – Átlagos idő

Oszd meg és uralkodj

Oszd meg és uralkodj

- Divide & Conquer („Oszd meg és uralkodj”) paradigma



Oszd meg és uralkodj

Oszd-meg és uralkodj:

- A feladatot ***több részfeladatra*** osztjuk, amelyek **hasonlóak az eredeti feladathoz**, de **méretük kisebb**,
- Rekurzív módon **megoldjuk a részfeladatokat**, majd **összevonjuk ezeket a megoldásokat**, hogy az eredeti feladatra megoldást adjanak.

Oszd meg és uralkodj

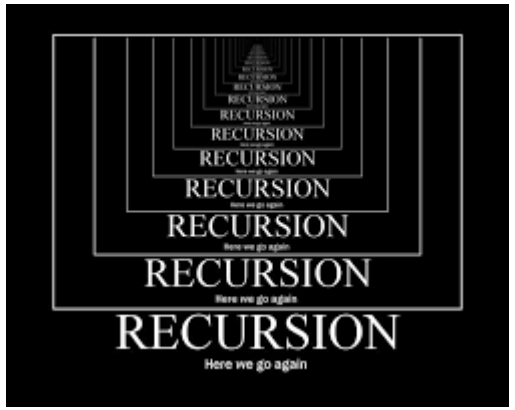
- **Felosztás:** hogyan osztjuk a feladatot több részfeladatra
- **Uralkodás:** a részfeladatokat rekurzív módon megoldjuk. Ha a részfeladatok mérete elég kicsi, akkor közvetlenül megoldja a részfeladatokat.
- **Összevonás:** a részfeladatok megoldásait összevonjuk az eredeti feladat megoldásává

  PL: vállalat adója ->részleg->munkások

Rekurzió

Rekurzió

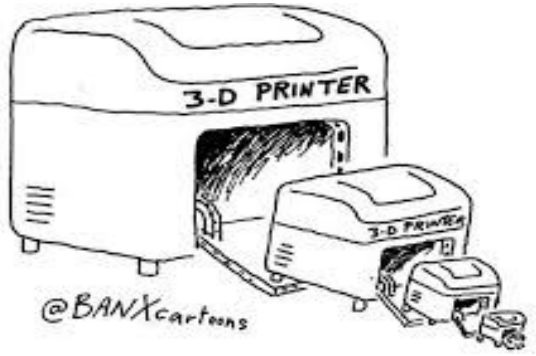
- A rekurzió olyan művelet, mely végrehajtáskor a saját maga által definiált műveletet, vagy műveletsort hajtja végre (általában a probléma egy kisebb példányára), ezáltal önmagát ismétli.



Rekurzió

- Egy rekurzív algoritmusnak két része van:
  - **Alapeset:** a megoldást már előre tudjuk vagy könnyen kiszámítható (ekkor nincs rekurzív hívás)
  - **Rekurzív eset:** a megoldást úgy kapjuk, hogy a probléma más, általában kisebb példányainak megoldását használjuk fel.

Csináljunk belőle mindig kisebbet, amíg el nem érjük az alapesetet



Rekurzió

- A rekurzív függvényhívás gondolata nagyon hasonló a matematikából ismert teljes indukciós bizonyítási módszerhez:

  az állítást belátjuk 1-re, majd n-1 alapján n-re.

- Az egyik legegyszerűbb rekurzív definíció a **faktoriálisé:**

  1 faktoriálisa 1, n faktoriálisa n* (n-1)!

Rekurzió

- Faktoriális számítás: $n! = n \cdot (n - 1) \cdot \ldots \cdot 2 \cdot 1$
- Rekurzív összefüggés: $n! = n \cdot (n - 1)!$
- Alapeset: $n = 1$
- Rekurzív eset: $n > 1$: $n \cdot (n - 1)!$

Rekurzió

- A rekurzív függvények természetesen nem a végtelenségig hívják saját magukat, **mindig van egy leállási feltétel,** azaz a rekurzív függvényeknek szükséges kelléke a feltételes elágazás.

Faktoriális

Faktoriális

Írjuk fel a lehetséges eseteket:

- **Alapeset:** 1! = 1
- **Rekurzív eset:** n!=n*(n-1)!

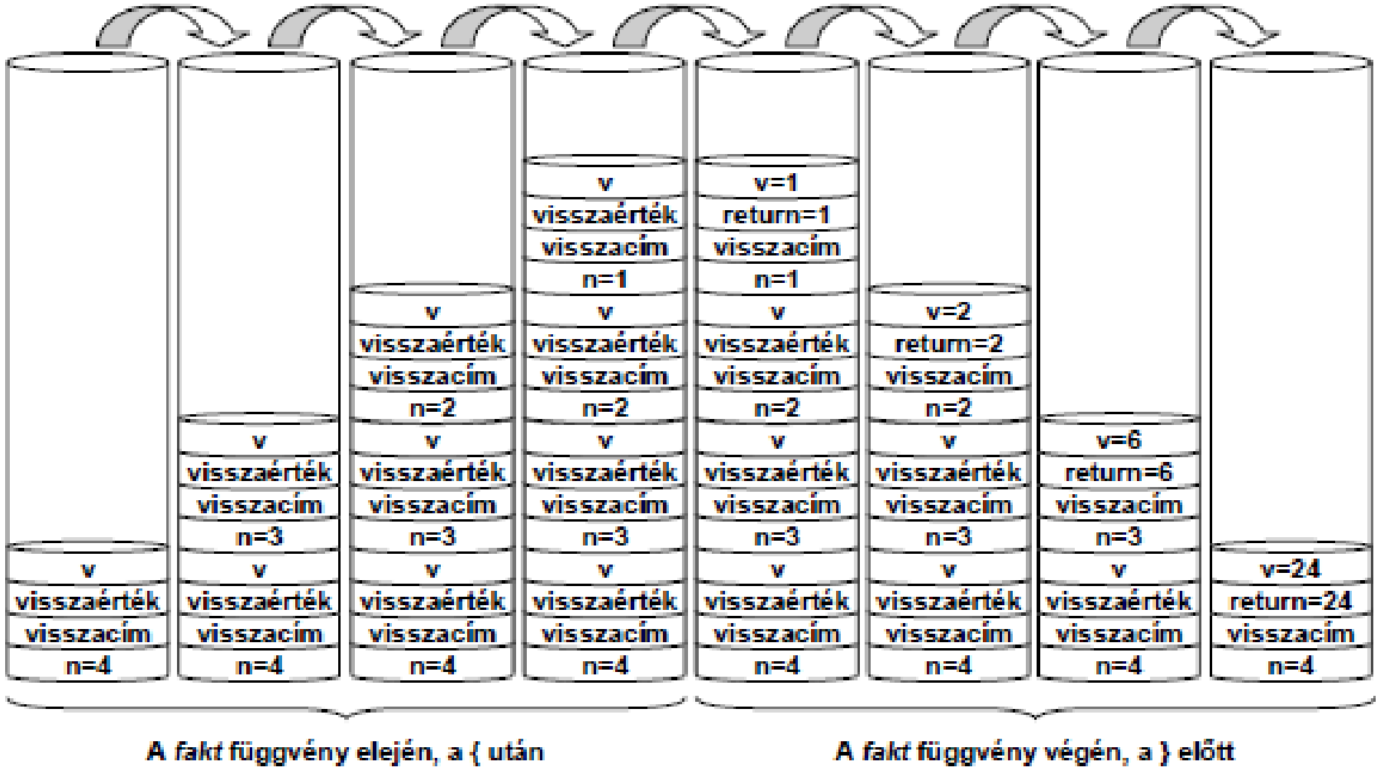
Faktoriális

Írjuk fel a lehetséges eseteket:

- **Alapeset:** 1! = 1
- **Rekurzív eset:** n!=n*(n-1)!

```
def fakt( n ) :
    if ( n == 1 )
        return 1
    else
        return n* fakt(n-1)
```

Faktoriális



https://visualgo.net/en/recursion

Fibonacci

Fibonacci

- Keressük meg a Fibonacci sorozat (1, 1, 2, 3, 5, 8, 13...) N. elemét

- **Alapeset:**

  az első és a második elem értéke 1.

- **Rekurzív eset:**

  az N. elem az N-1 - edik és az N-2 - dik elemek összege.

Fibonacci

```
def fibo( n ) :
    if (( n == 1 ) || (n == 0)):
        return 1
    else
        return fibo(n-1) + fibo(n-2)
```

https://visualgo.net/en/recursion

Lépcső probléma

Lépcső

- Adjunk rekurzív algoritmust, ami meghatározza, hogy hányféleképpen mehetünk fel egy n lépcsőfokból álló lépcsőn, ha egyszerre csak 1 vagy 2 lépcsőfokot léphetünk!

Lépcső

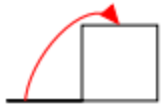
n=
4

Lépcső

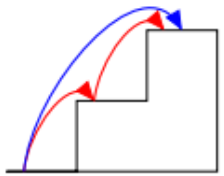
- Jelölje P(n) azt, hogy n lépcsőfokon hányféleképpen mehetünk fel.

- Ekkor a következő összefüggések állnak fent:

- P(1) = 1 (ha egy lépcső var [illegible]éképpen mehetünk)

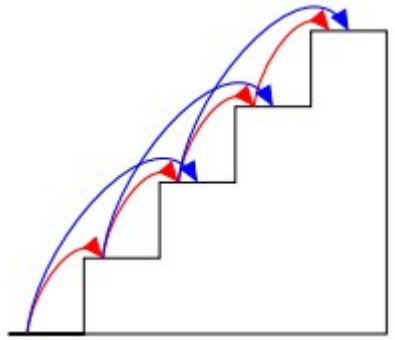


- P(2) = 2 (ha két lépcső v[illegible] tszer egyet lépünk, vagy egyszer kettőt)



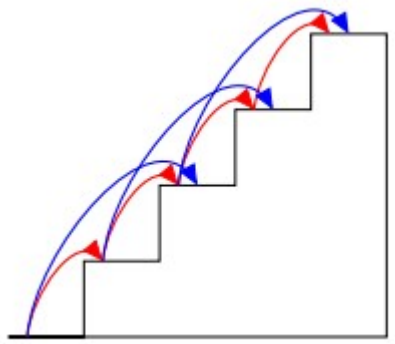
Lépcső

- $P(n) = P(n - 1) + P(n - 2)$, ha $n \geq 3$ (utolsó lépésként egyet vagy kettőt léphetünk)



Lépcső

- P(n) = P(n − 1) + P(n − 2), ha n ≥ 3 (utolsó lépésként egyet vagy kettőt léphetünk)



```
def P( n ) :
    if ( n == 1 )
        return 1
    elseif ( n == 2 )
        return 2
    else
        return P( n−1) + P( n−2)
```

N forint probléma

N forint probléma

4.3. Feladat n forintunk van. Minden nap veszünk pontosan egy dolgot a következők közül (zárójelben az ár szerepel) perec (1Ft), fagylalt (2Ft), csoki (2Ft). Adjunk meg egy rekurzív algoritmust, amely meghatározza, hogy hányféleképpen költhetjük el a pénzünket.

N forint probléma

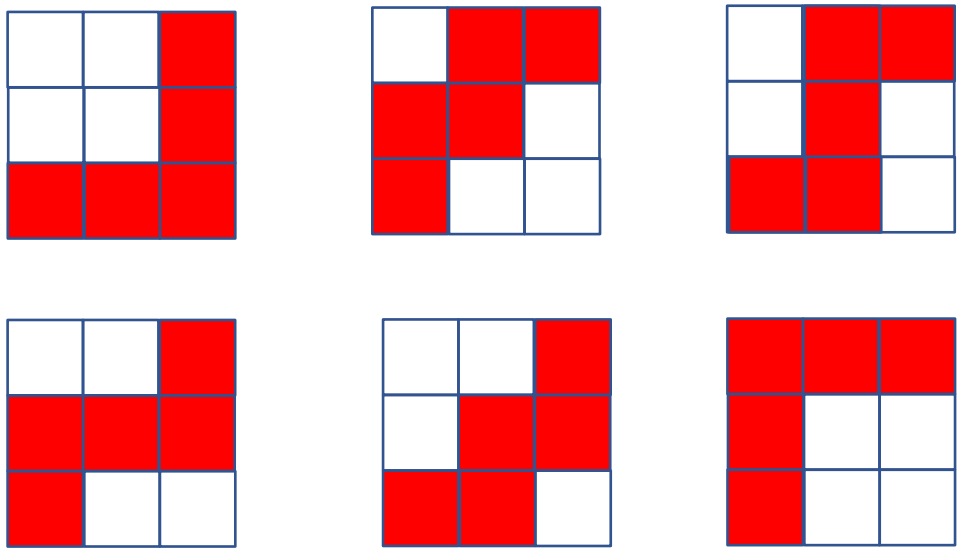
```
Function Q(n: Word) : Longint;
Begin
   If n=1 Then
        Q:=1
   Else If n=2 Then
        Q:=3
    Else
        Q:=Q(n-1)+2Q(n-2)
End;
```

Sakk probléma

Sakk

- Adjunk rekurzív algoritmust, amely meghatározza, hogy hányféleképpen juthatunk el egy **k** sorból és **n** oszlopból álló sakktábla bal alsó sarkából a jobb felső sarkába, ha csak a jobbra vagy a felfelé szomszédos mezőre léphetünk!

Sakk

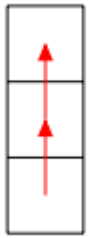


n=k=
3

Sakk

- Jelölje R(n, k) azt a számot, ahányféleképpen eljuthatunk a bal alsó sarokból a jobb felső sarokba.

- R(1, k) = 1 (csak felfelé mehetünk)

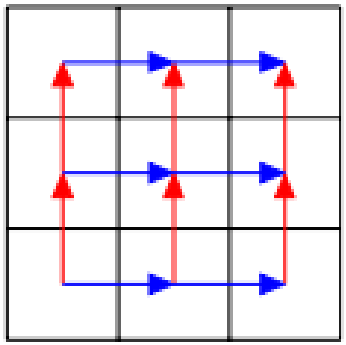


Sakk

- R(n, 1) = 1 (csak jobbra mehetünk)

Sakk

- $R(n, k) = R(n-1, k)+R(n, k-1)$
- ha $n, k \geq 2$ (az első lépés jobbra vagy felfelé történhet)



Sakk

```
def R(  n ,  k ):
    if ( ( n == 1 ) || ( k == 1 ) ):
        return 1
    else
    return R( n−1, k ) + R( n , k−1)
```

Hatvány probléma

Hatvány

- Hogyan tudnánk egy egész szám pozitív kitevős hatványát meghatározni rekurzívan?

- Alapeset:

- Rekurzív eset:

Hatvány

```
def hatvany(n,alap):
    if ( n == 1 ):
        return alap
    return alap * hatvany(n-1,alap)

hatvany( 10,2 )
```

Partició probléma

Partíció

Olyan partíciók érdekelnek bennünket, amelyekben minden elem legfeljebb m.

Jelöljük P(n,m)-mel az n szám olyan partícióinak számát, amelyekben minden szám kisebb vagy egyenlő m-mel.

Például: P(5,2)=3,

a partíciók pedig:
2 + 2 + 1,
2+1+1+1
és 1+1+1+1+1.

Természetes számok partíciója

Egy természetes szám partícióján (felbontásán) természetes számok összegére bontását értjük. Például az 5 a következőképpen bontható fel természetes számok összegére.

5
4+1
3+2
3+1+1
2+2+1
2+1+1+1
1+1+1+1+1

Így az 5-nek összesen 7 különböző partíciója van. Egy partícióban nem számít az elemek sorrendje.

Partíció

P(n,m)-re könnyen találhatunk rekurzív képletet

Felosztjuk a partíciókat két csoportra:

- egyikbe tartoznak azok, amelyekben nincsenek m-mel egyenlő számok (ezek száma $P(n,m-1)$)

- a másikba pedig azok, amelyekben van legalább egy m-el egyenlő szám (ezek száma $P(n-m,m)$).

Például: $P(5,2)=3$,

a partíciók pedig:
2 + 2 + 1,
2+1+1+1
és 1+1+1+1+1.

Partíció

Alapesetek:

N=0 ekkor 1 a visszaadott érték
N<0 ekkor 0 a visszaadott érték
M=0 ekkor 0 a visszaadott érték

A rekurzív képlet a következő:

$P(n,m) = P(n-m,m) + P(n,m-1)$, ahol $n > m > 1$

Például: P(5,2)=3,

a partíciók pedig:
2 + 2 + 1,
2+1+1+1
és 1+1+1+1+1.

Partíció

```
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        return count_partitions(n-m, m) + count_partitions(n, m-1)
```

Például: P(5,2)=3,

a partíciók pedig:
2 + 2 + 1,
2+1+1+1
és 1+1+1+1+1.

Ajánlott linkek

Ajánlott linkek

Videók

- https://www.youtube.com/watch?v=2Rr2tW9zvRg
- https://www.youtube.com/watch?v=B0NtAFf4bvU&list=PLBZBJbE_rGRV8D7XZ08LK6z-4zPoWzu5H&index=6
- **https://www.inf.u-szeged.hu/~rfarkas/Alga20/index.html**

Linkek

- http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-03.pdf