Dinamikus gráfok Ep. 1

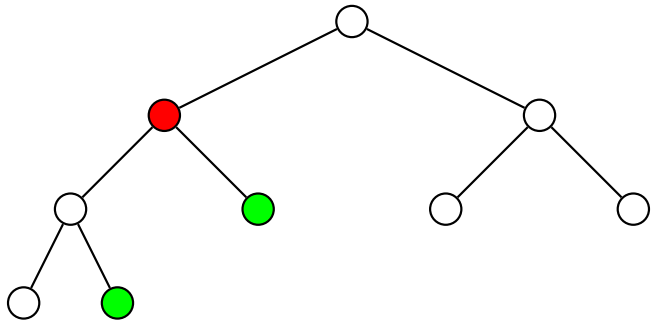
Iván Szabolcs

2016 ősz

Recap: LCA és RMQ

Lowest Common Ancestor

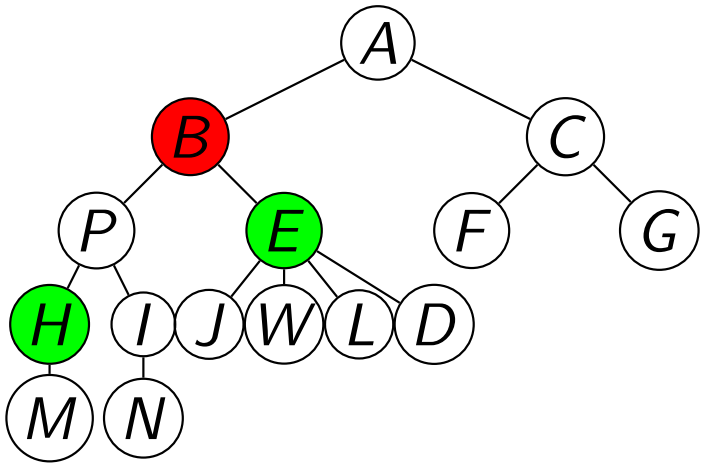
- Input: egy T (irányított, gyökeres) fa.
- Lekérdezések: (u, v) csúcspárok, adjuk vissza azt a w-t, aki őse u-nak is, v-nek is, de w-nél lejjebb már nincs ilyen csúcs.



LCA

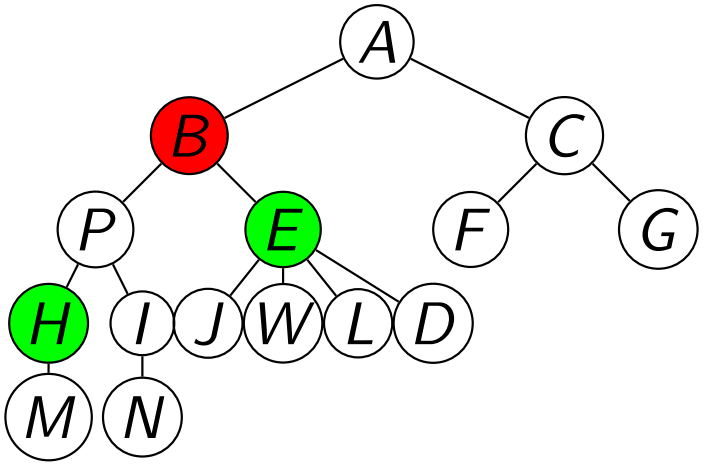
Preprocess

- Futtassunk a fában egy DFS-t és ahányszor érintünk egy csúcsot (alulról vagy felülről), írjuk ki a szintjét.
- Minden csúcshoz jegyezzük fel a (fentről) elérési és elhagyási idejét.



A	B	P	H	M	H	P	I	N	I	P	B	E	J	E	W	E	L	E	D	E	B	A	C	F	C	G	C	A
0	1	2	3	4	3	2	3	4	3	2	1	2	3	2	3	2	3	2	3	2	1	0	1	2	1	2	1	0

LCA



A	B	P	H	M	H	P	I	N	I	P	B	E	J	E	W	E	L	E	D	E	B	A	C	F	C	G	C	A
0	1	2	3	4	3	2	3	4	3	2	1	2	3	2	3	2	3	2	3	2	1	0	1	2	1	2	1	0

Ha a két intervallum tartalmazza egymást, akkor a közös ős: a csúcs a nagyobb intervallummal.

Ellenkező esetben diszjunktak és a közös ős a két intervallum közt érintett legkisebb szintű csúcs lesz.

RMQ

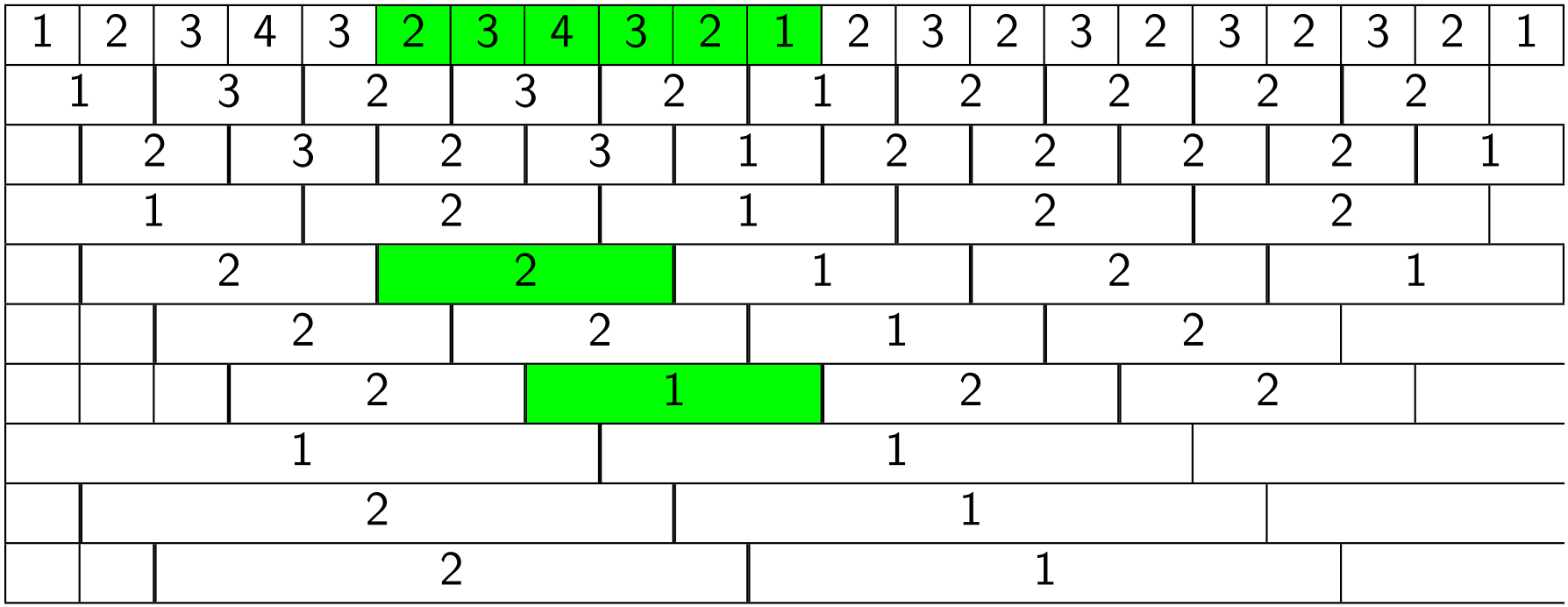
Range Minimum Query, RMQ

- Input: egy $A[1, \ldots, n]$ tömb.
- Queryk: $[i, j]$ intervallumok.
- Output: az intervallum egy legkisebb eleme.

Preprocess

- Minden i-re és 2^j-re, amire $i + 2^j \leq n$, számítsuk ki előre az $A[i \ldots i + 2^j - 1]$ tömb minimumértékét.
- Alapeset: $A[i, i]$ minimuma $A[i]$.
- Indukció: $A[i, i + 2^j - 1]$ minimuma $A[i, i + 2^{j-1} - 1]$ és $A[i + 2^{j-1}, i + 2^j - 1]$ minimuma, az $O(1)$ idő.
- Ilyen intervallumból van $O(n \log n)$.

RMQ



Query

Az $[i \ldots j]$ intervallum minumum eleme:

- vegyük a legnagyobb 2^k kettő-hatványt, ami belefér az intervallumba;
- a válasz $A[i, i + 2^k - 1]$ és $A[j + 1 - 2^k, j]$ minimuma lesz – konstans idő.

RMQ: $O(n)$ preprocess, $O(\log n)$ query

Preprocess

- Osszuk fel a tömböt $s = \frac{\log n}{2}$ hosszú blokkokra, mindnek tároljuk el a minimumát: $O(n)$ idő.
- A blokkoknak az előző algoritmussal számoljuk ki az ,,i. és $i + 2^k - 1$. közti blokkok minimumjainak minimumjait".
  - Blokkok száma: $\frac{2n}{\log n}$.
  - Preprocess erre: $O(\frac{2n}{\log n} \log(\frac{2n}{\log n})) = O(n)$ idő.

Összesen $O(n)$ preprocess idő.

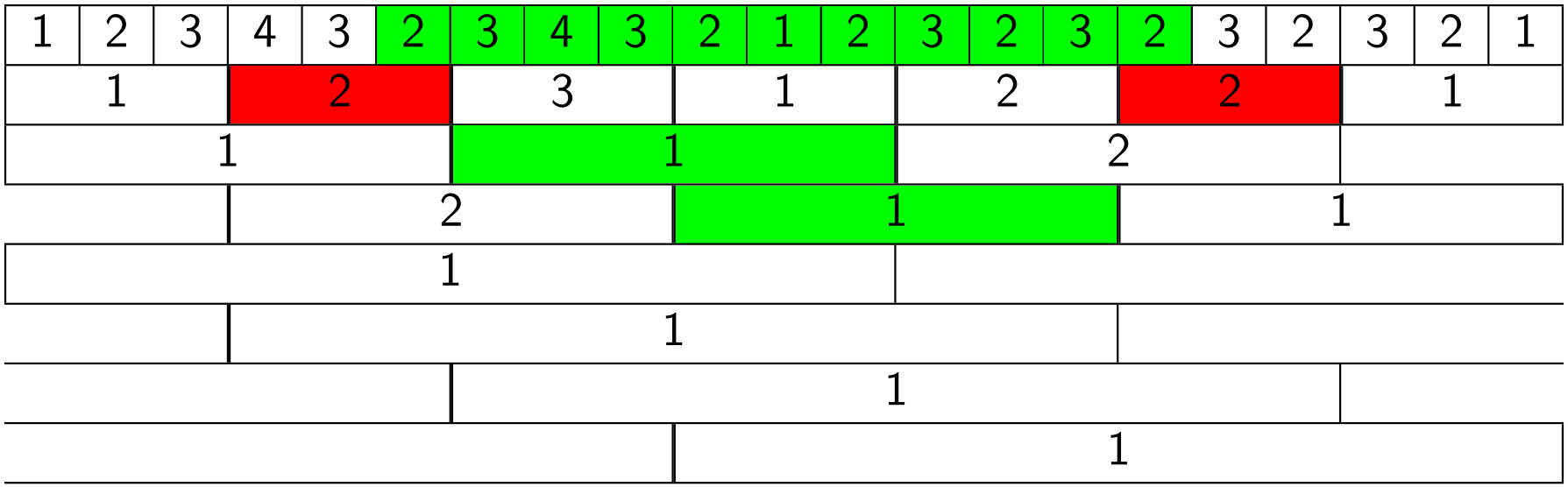
Query

Az $[i \ldots j]$ intervallum minimum eleme:

- a teljesen beleeső blokkok intervallumjainak minimuma: $O(1)$
- az elmetszett blokkokban egy linear search: $O(\log n)$.

Összesen $O(\log n)$ query idő.

RMQ: $O(n)$ preprocess, $O(\log n)$ query, block size 3



- A zöld intervallumba teljesen beleeső blokkok $(3, 1, 2)$
- Ezeknek a ,,legnagyobb beférő kettőhatvány blokkot átfogó szélső ablakok" minimuma (a két zöld) – $O(1)$ idő
- . . . plusz a ,,metszett" (piros) blokkokban egy linear search – $O(\log n)$ idő

RMQ ± 1

Linear search a metszett blokkban?

A blokkokon belüli lineáris időt akarjuk lefaragni.

- Egy LCA-ból gyártott RMQ-ban a szomszédos elemek közt mindig ± 1 a különbség.
- Ha csak a $+/-$ előjeleket ismerjük, abból is meg tudjuk mondani, hogy hol van a minimum egy intervallumon belül!

$(1,2,3) \mapsto ++$ $(4,3,2) \mapsto --$ $(3,4,3) \mapsto +-$

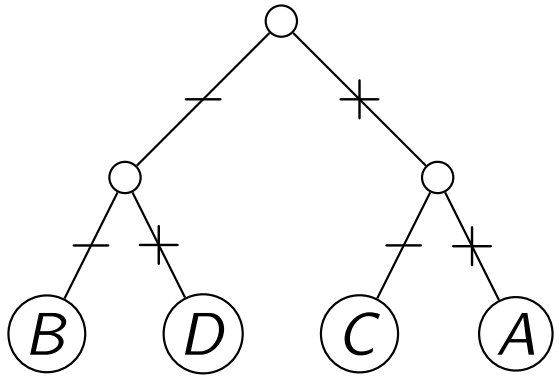
$(2,1,2) \mapsto -+$ $(3,2,3) \mapsto -+$ $(2,3,2) \mapsto +-$

$(3,2,1) \mapsto --$

- Minden típusra elég egyszer kiszámolni a query táblát.
- A blokkok $\frac{\log n}{2}$ hosszúak – ez csak $2^{\frac{\log n}{2}} = \sqrt{n}$ típus.

RMQ ±1

1	2	3	4	3	2	3	4	3	2	1	2	3	2	3	2	3	2	3	2	1
1			2			3			1			2			2			1		
A			B			C			D			D			C			B		



1	2	3
0		
A	1	

4	3	2
1		
B	2	

3	4	3
0		
C	2	

2	1	2
1		
D	1	

RMQ ± 1 időigény

Preprocess

- A $\frac{2n}{\log n}$ darab blokkra a 2^d hosszú blokk-intervallumok kiszámítása: $O(\frac{2n}{\log n} \log \frac{2n}{\log n}) = O(n)$
- Egy blokk-típusra az azon belüli 2^d hosszú intervallumoké: $\frac{\log n}{2} \log \frac{\log n}{2} = O(\log n \log \log n)$ idő
- Az $O(\sqrt{n})$ blokk-típusra ez $O(\sqrt{n} \cdot \log n \log \log n) = O(n)$ idő
- A blokkok típushoz rendelése fával: $O(n)$ idő

Összesen: $O(n)$ idő.

Query

- A teljesen benne levő blokk-intervallumokon belül legnagyobb ablakok-keresés: $O(1)$.
- Az elmetszett szélső blokkokban belül szintén egy-egy ablakkeresés: $O(1)$.

Általános RMQ?

Az LCA-t meg tudtuk oldani úgy, hogy $O(n)$ időben átkonvertáltuk[a] a problémát egy RMQ± 1 inputra, amit $O(n)$ preprocess és konstans query időben meg tudtunk oldani.

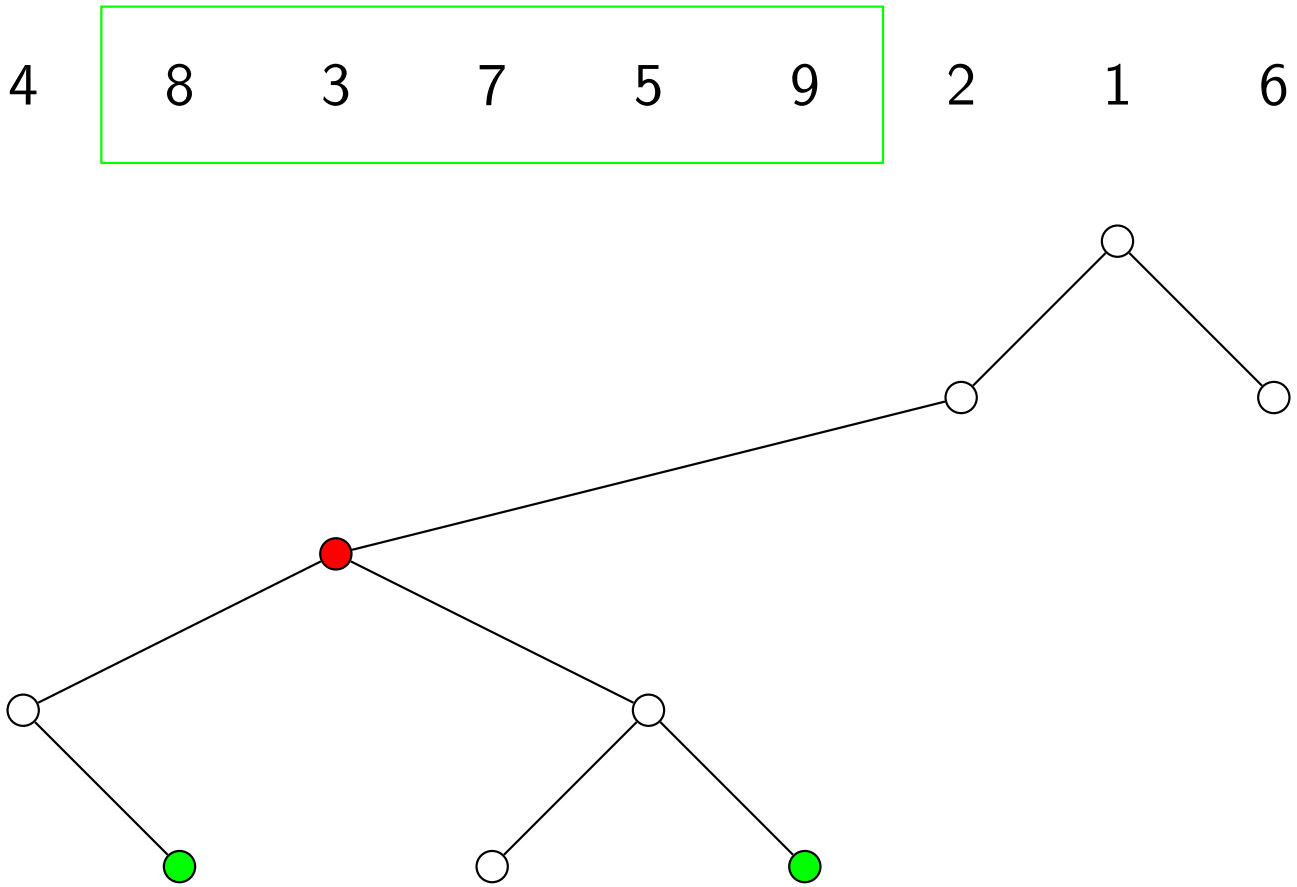
[a] „visszavezettük”

Vajon az általános RMQ-t (amikor nincs ± 1 feltétel az input tömbre) is meg lehet oldani $O(n)$ preprocess, $O(1)$ query időben?

igen!

Átkonvertáljuk LCA-ra `:D`

RMQ $\leq$ LCA: Cartesian tree építése



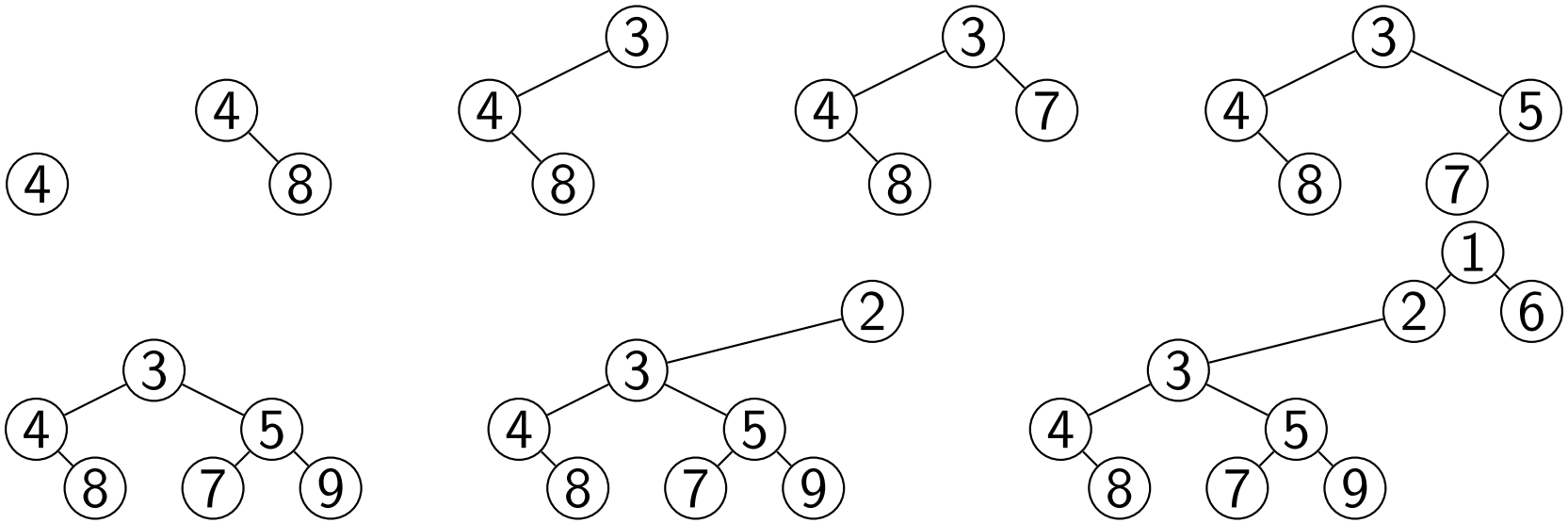
,,cartesian tree”

aztán egy RMQ query ebben egy LCA.

Cartesian tree építése

4 8 3 7 5 9 2 1 6

- x hozzáadása: a jobb legalsó csúcsból indulunk felfele
- ha az aktuális y csúcs kisebb: x ennek legyen jobb gyereke, done
- ha nagyobb: x bal gyereke legyen y, haladjunk x eredeti apjával tovább



Építés időigénye?

Amortizált elemzés

- Mindig a fa jobb ,,gerincén" megyünk felfele
- Ha valaki kisebb az új elemnél: alákötjük $O(1)$ időben jobbra, az új elem lesz a gerinc vége
- Különben haladunk felfele $O(1)$ időben – de ez a csúcs leesik a gerincről!
- Beszúráskor előre kifizetjük azt a költséget, amikor a beszúrt csúcs majd leesik a gerincről
- Egy beszúrás költsége amortizált konstans
- Az egész fa felépítése $O(n)$

RMQ–LCA

Mindkét probléma $O(n)$ preprocess mellett $O(1)$ query time-ban megoldható.

Még jobb RMQ?

Galaktikus algoritmusok

Gyakorlatban nem biztos, hogy az aszimptotikusan jobb függvény tényleg gyorsabban fut.

- n^{100} vs. 1.0001^n, $n = 100$-ra is 10^{200} vs. 1.01005 (ha n kb. húszmillió, már gyorsabb lesz – de arra n^{100} már nagyon sok)
- $6 \times 10^{23} n$ vs $n \log n$

Van értelme tehát a konstans szorzókon is próbálni faragni.

A fa egyébként is bonyolult dolog

- Az $s = \frac{\log n}{2}$ hosszú blokkok minimumaira az $n \log n$-es algoritmus rendben van: $T[i][j]$-be tesszük az $A[i \ldots i + 2^j - 1]$ rész tömb minimumának indexét (mondjuk), azzal, hogy $T[i][0] = i$ és $T[i][j+1]$-be $T[i][j]$ és $T[i + 2^j - 1][j]$ közül (ha az utóbbi létezik) azt tesszük, amelyik indexen kisebb tömbelemet találunk

Még jobb RMQ?

A blokkok minimumainak gyors lekérdezését kellene gyorsabban / kevesebb tárban megcsinálni.

Az előző algoritmus

- felépítette a blokk Cartesian fáját
- annak készített egy (kétszer hosszabb) segédtömböt, ± 1 RMQ-ra
- ennek a segédtömbnek a $\pm$-típusához számolta ki a (kis) segédtáblázatot.

Cél: ne kelljen fát építeni

Még jobb RMQ?

Ötlet

Ha két (nem feltétlenül ± 1) blokk Cartesian fája ugyanolyaz, akkor ugyanaz a típusuk!

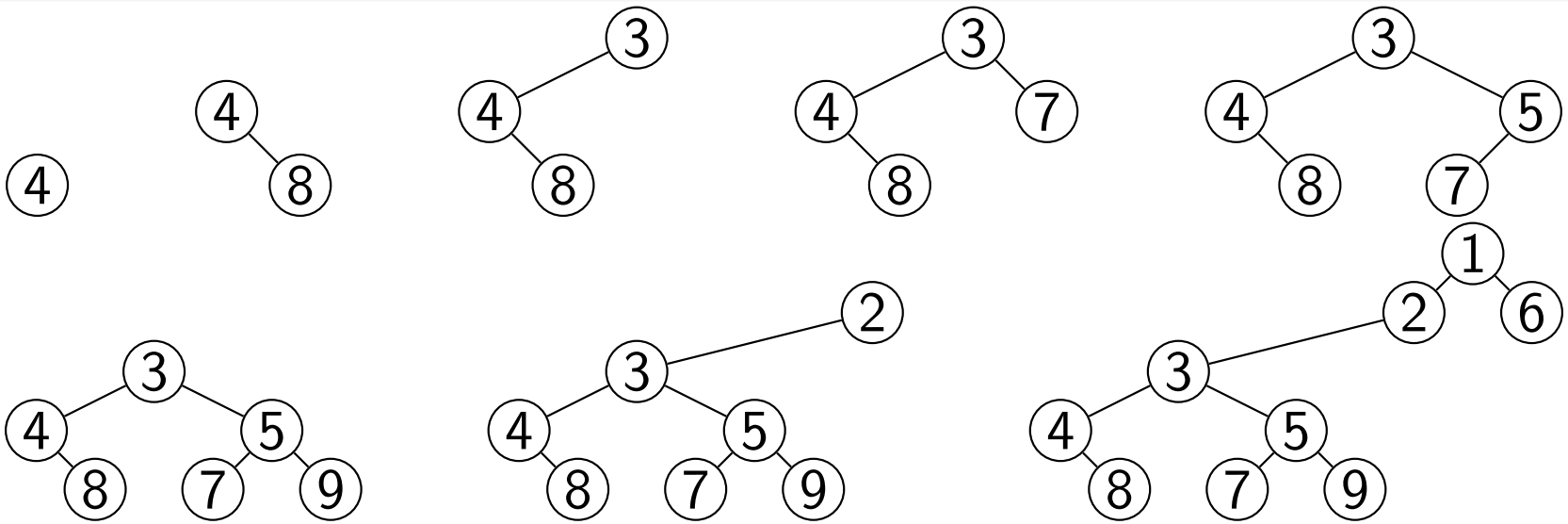
Hiszen a Cartesian fa Euler-bejárásából számoljuk a ± 1-segédtömböt.

Tehát ha tudnánk a blokkok Cartesian fájával ,,indexelni", anélkül, hogy megcsinálnánk a fát, megúsznánk pl. a kettes szorzót az Euler-bejárásból

Még jobb RMQ

A fára nincs szükségünk

- A fát egyértelműen leírja egy $\{\uparrow, \leftarrow\}^*$-sorozat
- Csak a rightmost path értékeit tároljuk mint egy verem (egy tömbben)
- Ha az aktuális beszúrt érték kisebb: $\uparrow$ és $\textsc{pop}$
- Amint nem: $\leftarrow$



$\leftarrow\leftarrow\uparrow\uparrow\leftarrow\leftarrow\uparrow\leftarrow\leftarrow\uparrow\uparrow\uparrow\leftarrow\uparrow\leftarrow\leftarrow$

Még jobb RMQ

- Mivel mindenképp $\leftarrow$-val kezdünk, a $\leftarrow = 1$, $\uparrow = 0$ ad nekünk egy legfeljebb $2s$-jegyű bináris számot, ha s a blokk hossza
- ha $s = \frac{\log n}{4}$, akkor ez megint $\sqrt{n}$-féle érték lehet
- így minden blokkra faépítés nélkül, csak egy (tömbben tárolt) veremmel ki tudjuk számítani a reprezentáns indexet és a ténylegesen létező indexekre felépíteni $\frac{\log n}{4} \log \frac{\log n}{4}$ időben a sparse tablet

Előny: nincs duplázás, nincs dinamikus struktúra, minden megy tömbben, kisebb a lineáris preprocess szorzója

Summary

- Láttunk dinamikus gráfalgoritmusokat: inkrementális, dekrementális, update time, query time
- Irányítatlan gráfokra inkrementális összefüggőség: $O(\alpha(n))$ amortizált update és query time
- Irányítatlan gráfokra fully dynamic összefüggőség: $O(\log^2 n)$ amortizált update és $O(\log n)$ query time
- Irányított gráfokra inkrementális SCC ,,mióta érhetőek el egymásból'': $O(n)$ amortizált update és $O(1)$ query time

Közben láttunk

- Amorizált időigény-elemzést
- Mester tételt
- Piros-fekete fákat
- Euler-bejárást
- Eulerian Tour forestet
- RMQ-t
- Cartesian Treet
- $\langle O(n), O(1) \rangle$ RMQ-t, LCA-t
- Sort veremmel
- Min/max/sum $O(1)$ vermet