

Algoritmusok és adatszerkezetek II.

Amotrizált költségelemzés, Fibonacci kupacok

Szegedi Tudományegyetem



- A legrosszabb költségelemzés túl pesszimista tud lenni
- Amortizált költségelemzésnél az adatszerkezetek 'életútját' vizsgáljuk
- Lehetnek költséges műveleteink, ha azok *kellően* ritkák
 - Pl. dinamikusan bővülő tömb



- A legrosszabb költségelemzés túl pesszimista tud lenni
- Amortizált költségelemzésnél az adatszerkezetek 'életútját' vizsgáljuk
- Lehetnek költséges műveleteink, ha azok *kellően* ritkák
 - Pl. dinamikusan bővülő tömb

Fontos

Ennél az elemzésnél a véletlennek nincs szerepe: az egyes műveletek átlagos költségére adunk felső korlátot a *legrosszabb esetben*.



- A legrosszabb költségelemzés túl pesszimista tud lenni
- Amortizált költségelemzésnél az adatszerkezetek 'életútját' vizsgáljuk
- Lehetnek költséges műveleteink, ha azok *kellően* ritkák
 - Pl. dinamikusan bővülő tömb

Fontos

Ennél az elemzésnél a véletlennek nincs szerepe: az egyes műveletek átlagos költségére adunk felső korlátot a *legrosszabb esetben*.

Fő megközelítések

- 1 Összesítéses elemzés
- 2 Könyvelési módszer
- 3 Potenciálmódszer



```
NÖVEL(A) {  
    i=0  
    while i < A.hossz és A[i] = 1 {  
        A[i] = 0  
        i = i+1  
    }  
    if i < A.hossz {  
        A[i] = 1  
    }  
}
```



Bináris számláló növelése

```
NÖVEL(A) {  
    i=0  
    while i < A.hossz és A[i] = 1 {  
        A[i] = 0  
        i = i+1  
    }  
    if i < A.hossz {  
        A[i] = 1  
    }  
}
```

i	3	2	1	0	\sum ktg.
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	3
	0	0	1	1	4
	0	1	0	0	7
	0	1	0	1	8
	0	1	1	0	10
	0	1	1	1	11
	1	0	0	0	15
	1	0	0	1	16
	1	0	1	0	18
	1	0	1	1	19
	1	1	0	0	22
	1	1	0	1	23
	1	1	1	0	25
	1	1	1	1	26



Összesítéses elemzés

n hosszú műveletsorra állítunk föl $T(n)$ felső korlátot

→ a műveletek átlagos költsége $T(n)/n$

Példa

k bites számlálón NÖVEL művelet n -szeri végrehajtása: $O(nk)$

Helyes, de nem éles korlát, mivel az i pozíciójú bit csak minden 2^i számú végrehajtás után változik.



Összesítéses elemzés

n hosszú műveletsorra állítunk föl $T(n)$ felső korlátot

→ a műveletek átlagos költsége $T(n)/n$

Példa

k bites számlálón NÖVEL művelet n -szeri végrehajtása: $O(nk)$

Helyes, de nem éles korlát, mivel az i pozíciójú bit csak minden 2^i számú végrehajtás után változik.



Összesítéses elemzés

n hosszú műveletsorra állítunk föl $T(n)$ felső korlátot
→ a műveletek átlagos költsége $T(n)/n$

Példa

k bites számlálón NÖVEL művelet n -szeri végrehajtása: $O(nk)$
Helyes, de nem éles korlát, mivel az i pozíciójú bit csak minden 2^i számú végrehajtás után változik.

Élesebb korlát

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n = O(n)$$

Vagyis a NÖVEL művelet amortizált költsége $O(n)/n = O(1)$



- Különböző műveletekre különböző költséget számolunk el
 - A i -edik műveletre elszámolt \hat{c}_i *amortizációs költség* tetszőlegesen eltérhet annak c_i **tényleges költségétől**



- Különböző műveletekre különböző költséget számolunk el
 - A i -edik műveletre elszámolt \hat{c}_i *amortizációs költség* tetszőlegesen eltérhet annak c_i **tényleges költségétől**
 - Azonban minden n hosszú műveletsorra teljesüljön, hogy

$$\sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i,$$

azaz a mindenkori hitelegyenleg $\left(\sum_{i=1}^n \hat{c}_i - c_i \right)$ nemnegatív



- A NÖVEL művelet működése során könyveljünk el 2 egységnyi költséget egy bit 1-re állításához
- A költség fele a majdani visszaállításra félretett „hitel”

Kérdés

A CSÖKKENT műveletet bevezetését követően is maradna az $O(1)$ amortizált költség?



- A NÖVEL művelet működése során könyveljünk el 2 egységnyi költséget egy bit 1-re állításához
- A költség fele a majdani visszaállításra félretett „hitel”
- A NÖVEL minden hívása során 1 bitet állítunk 1-re
 - n végrehajtás $\Rightarrow 2n$ összköltség $\Rightarrow O(1)$ költség/végrehajtás

Kérdés

A CSÖKKENT műveletet bevezetését követően is maradna az $O(1)$ amortizált költség? Nem, $O(k)$ lenne.



- Dinamikusan bővülő tömb betelésekor megduplázza méretét, és a benne aktuálisan szereplő értékeket a megnövelt méretű tömbbe másolja
- Egy elem beszúrásához rendeljünk 3 *kreditet*
 - Az első egység elhasználódik a kulcs beszúrása kapcsán
 - A második egységet saját maga majdani (első alkalommal történő) átmásolására tartsa fenn
 - A harmadik egységből minden csúcs adományoz egy már 0 egyenlegű kulcsnak a másolására (tudja, hogy vissza fogja kapni)



- Dinamikusan bővülő tömb betelésekor megduplázza méretét, és a benne aktuálisan szereplő értékeket a megnövelt méretű tömbbe másolja
- Egy elem beszúrásához rendeljünk 3 *kreditet*
 - Az első egység elhasználódik a kulcs beszúrása kapcsán
 - A második egységet saját maga majdani (első alkalommal történő) átmásolására tartsa fenn
 - A harmadik egységből minden csúcs adományoz egy már 0 egyenlegű kulcsnak a másolására (tudja, hogy vissza fogja kapni)

Észrevétel

Amikor bővítünk, ugyanannyi kulcsnak lesz 0 az egyenlege, mint ahánynak 2.



Amortizált költségelemzés – Potenciálmódszer

- Az adatszerkezet i pillanatbeli állapotát jelöljük D_i -vel
- Vezessük be a $\Phi : \mathbb{N} \rightarrow \mathbb{R}$ **potenciálfüggvényt**, ami az adatszerkezet egy D_i állapotához rendel egy **potenciált**
- Az amortizációs költség legyen $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$



Amortizált költségelemzés – Potenciálmódszer

- Az adatszerkezet i pillanatbeli állapotát jelöljük D_i -vel
- Vezessük be a $\Phi : \mathbb{N} \rightarrow \mathbb{R}$ **potenciálfüggvényt**, ami az adatszerkezet egy D_i állapotához rendel egy **potenciált**
- Az amortizációs költség legyen $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

n hosszú műveletsorra a teljes *teleszkopikus összeg*

$$\sum_{i=1}^n \hat{c}_i = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^n c_i$$



Amortizált költségelemzés – Potenciálmódszer

- Az adatszerkezet i pillanatbeli állapotát jelöljük D_i -vel
- Vezessük be a $\Phi : \mathbb{N} \rightarrow \mathbb{R}$ **potenciálfüggvényt**, ami az adatszerkezet egy D_i állapotához rendel egy **potenciált**
- Az amortizációs költség legyen $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

n hosszú műveletsorra a teljes *teleszkopikus összeg*

$$\sum_{i=1}^n \hat{c}_i = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^n c_i$$

- Olyan potenciálfüggvényt keresünk, melyre $\Phi(D_n) \geq \Phi(D_0)$, mivel ekkor nyilvánvalóan $\sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i$ is teljesül



Amortizált költségelemzés – Potenciálmódszer

- Az adatszerkezet i pillanatbeli állapotát jelöljük D_i -vel
- Vezessük be a $\Phi : \mathbb{N} \rightarrow \mathbb{R}$ **potenciálfüggvényt**, ami az adatszerkezet egy D_i állapotához rendel egy **potenciált**
- Az amortizációs költség legyen $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

n hosszú műveletsorra a teljes *teleszkopikus összeg*

$$\sum_{i=1}^n \hat{c}_i = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^n c_i$$

- Olyan potenciálfüggvényt keresünk, melyre $\Phi(D_n) \geq \Phi(D_0)$, mivel ekkor nyilvánvalóan $\sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i$ is teljesül

Kényelmes megoldás

$\Phi(D_0) = 0$, és lássuk be, hogy $\Phi(D_i) \geq 0$ minden i -re



- Legyen $\Phi(D_i) = b_i$ a NÖVEL művelet i -szeri alkalmazására a számlálóban szereplő 1 értékű bitek száma
- Jelölje t_i a NÖVEL művelet i -edik végrehajtásakor 1-ről 0-ra változó bitek számát (vagyis $c_i \leq t_i + 1$)
 - $b_i \leq b_{i-1} - t_i + 1 \Rightarrow \Phi(D_i) - \Phi(D_{i-1}) \leq 1 - t_i$
 - Így az amortizációs költség $\hat{c}_i \leq c_i + 1 - t_i = t_i + 1 + 1 - t_i = 2$

Észrevétel

Mivel $\Phi(D_0) = 0$ és minden $\Phi(D_i) \geq 0$, így az n művelet amortizált költségének összege felső korlátja a tényleges összköltségnek ($O(n)$)



- Binomiális kupachoz hasonló (annál kötetlenebbül strukturált), amortizált értelemben jobban viselkedő adatszerkezet
 - A kupacot alkotó fák *nem* rendezettek
 - A csúcsok gyerekei kétirányú ciklikus listával összekapcsoltak
 - $\text{min}[H]$ pointer a gyökérlista minimális kulcsú csúcsára mutat
 - A fák sorrendje a gyökérlistában tetszőleges
 - A kupacban található megjelölt csúcsok



Megjelölt csúcsok

- Egy csúcs megjelölt, ha már veszített el gyereket azóta, hogy egy másik csúcs gyerekévé vált
 - Létrehozásukkor jelöletlenek a csúcsok



- Egy csúcs megjelölt, ha már veszített el gyereket azóta, hogy egy másik csúcs gyerekévé vált
 - Létrehozásukkor jelöletlenek a csúcsok

A potenciálfüggvény

$$\Phi(H) = t(H) + 2m(H)$$

- $t(H)$ a kupac gyökerlistájában található fák száma
- $m(H)$ a kupacban található megjelölt csúcsok száma



Megjelölt csúcsok

- Egy csúcs megjelölt, ha már veszített el gyereket azóta, hogy egy másik csúcs gyerekévé vált
 - Létrehozásukkor jelöletlenek a csúcsok

A potenciálfüggvény

$$\Phi(H) = t(H) + 2m(H)$$

- $t(H)$ a kupac gyökerlistájában található fák száma
- $m(H)$ a kupacban található megjelölt csúcsok száma

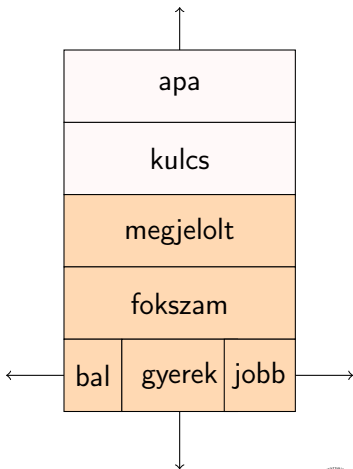
Észrevétel

A potenciál végig nemnegatív, így a teljes amortizált költség felső korlátja a műveletsorozat teljes aktuális költségének felső korlátja is



Fibonacci kupacok implementációja

```
class Node {  
    Object kulcs;  
    Node *apa;  
    int fokszam;  
    boolean megjelolt;  
    Node *gyerek;  
    Node *bal;  
    Node *jobb;  
}
```

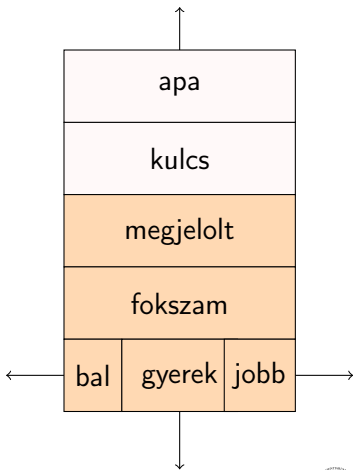


Fibonacci kupacok implementációja

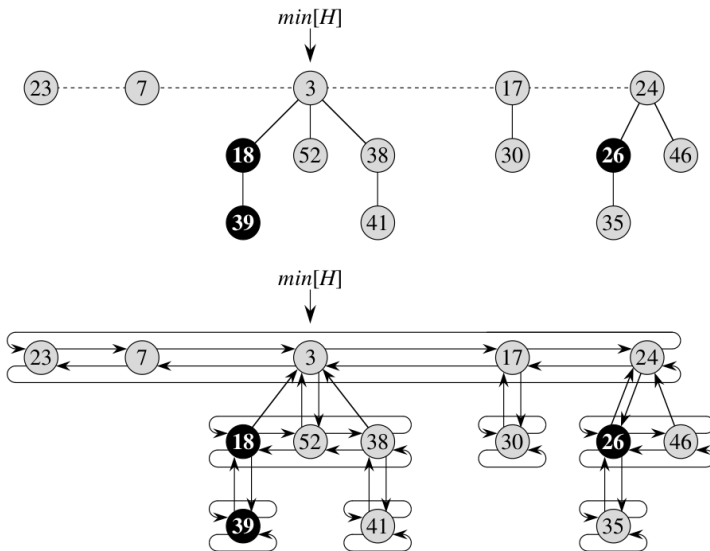
```
class Node {  
    Object kulcs;  
    Node *apa;  
    int fokszam;  
    boolean megjelolt;  
    Node *gyerek;  
    Node *bal;  
    Node *jobb;  
}
```

Emlékeztető

A megjelöltséget azt jelöli, hogy a csúcs vesztette-e el gyerekeit mióta egy másik csúcs gyerekévé vált



Fibonacci kupacok szerveződése



Forrás: CLRS: Új algoritmusok 20.1 ábrája

Kupacműveletek legrosszabb esetbeli viselkedése

Művelet	Bináris	Binomiális	Fibonacci ¹
MIN-KERES	$O(1)$	$O(\log n)$	$O(1)$
SORBOL-MIN	$O(\log n)$	$O(\log n)$	$O(\log n)$
BESZÚR	$O(\log n)$	$O(\log n)^2$	$O(1)$
KULCSOTCSÖKKENT	$O(\log n)$	$O(\log n)$	$O(1)$
EGYESÍT	$O(n)$	$O(\log n)$	$O(1)$
TÖRÖL	$O(\log n)$	$O(\log n)$	$O(\log n)$

¹amortizált költségek

²amortizált költségben $O(1)$



- Remek választás, ha tudjuk, hogy a SORBOL-MIN és TÖRÖL műveleteket keveset használjuk
 - Bizonyos gráfalgoritmusok (pl. Dijkstra) esetében a KULCSOTCSÖKKENT módszer alkalmazása dominál



A Fibonacci kupacok viselkedése

- Remek választás, ha tudjuk, hogy a SORBOL-MIN és TÖRÖL műveleteket keveset használjuk
 - Bizonyos gráfalgoritmusok (pl. Dijkstra) esetében a KULCSOTCSÖKKENT metódus alkalmazása dominál

Ha nem hajtunk végre KULCSOTCSÖKKENT és TÖRÖL műveletet

Egy (fokszám alapján) rendezetlen "binomiális kupacot" kapunk
→ $\log n$ a kupacbeli csúcsok maximális fokszámának felső korlátja



A Fibonacci kupacok viselkedése

- Remek választás, ha tudjuk, hogy a SORBOL-MIN és TÖRÖL műveleteket keveset használjuk
 - Bizonyos gráfalgoritmusok (pl. Dijkstra) esetében a KULCSOTCSÖKKENT metódus alkalmazása dominál

Ha nem hajtunk végre KULCSOTCSÖKKENT és TÖRÖL műveletet

Egy (fokszám alapján) rendezetlen "binomiális kupacot" kapunk
→ $\log n$ a kupacbeli csúcsok maximális fokszámának felső korlátja

KULCSOTCSÖKKENT és TÖRÖL műveletek végrehajtása esetén

$\log_{\phi} n = O(\log n)$ a kupacbeli csúcsok maximális fokszámának felső korlátja → innen jön a Fibonacci-kupac elnevezés is (mivel az i -edik Fibonacci szám fölírható $\frac{\phi^i - \psi^i}{\sqrt{5}}$ alakban, ahol $\phi = \frac{1+\sqrt{5}}{2}$)



EGYESÍT és BESZÚR

H_1 és H_2 Fibonacci kupacok egyesítésekor a kupacok gyökerlistáit összefűzzük, ($\text{min}[H]$ aktualizálásán túl) **más teendők nincs**

MIN-KERES

$\text{min}[H]$ explicit tárolásából adódóan $O(1)$



- Ez az a pont, amikor próbáljuk a binomiális kupachoz hasonlítóvá tenni a Fibonacci kupacunkat
 - 1 $\min[H]$ által meghatározott csúcsot eltávolítjuk a gyökérlistából
 - 2 $\min[H]$ gyerekeit a gyökérlistába delegáljuk
 - 3 Összevonjuk a gyökérlistában szereplő azonos fokszámú fákat (segéd tömb használatával)



- 1 Ha a kulcs csökkentett értéke túl kicsi, akkor a csúcsot kivágjuk és a gyökérlistába helyezzük
- 2 A kivágott csúcs szülejét (ha az nem gyökérlistabeli) megjelöltté tesszük
- 3 Amennyiben egy már megjelölt csúcs vesztene el egy újabb gyereket, úgy azt is rekurzívan a gyökérlistába visszük (megjelöltségét eltávolítjuk)



Ferde fák (Splay tree)

- Olyan keresőfa, ami bármely művelet végrehajtása után (még egy sikertelen keresés után is), az utoljára érintett csúcsot forgatások segítségével a gyökérbe viszi
- Legrosszabb esetben $O(n)$ magasságú, de amortizált tekintetben a műveletei $O(\log n)$ -beliek
- Működése mögötti intuíció: lehetnek sűrűbben érintett elemei a fának \Rightarrow ne hagyjuk őket „lesüllyedni”



Ferde fák (Splay tree)

- Olyan keresőfa, ami bármely művelet végrehajtása után (még egy sikertelen keresés után is), az utoljára érintett csúcsot forgatások segítségével a gyökérbe viszi
- Legrosszabb esetben $O(n)$ magasságú, de amortizált tekintetben a műveletei $O(\log n)$ -beliek
- Működése mögötti intuíció: lehetnek sűrűbben érintett elemei a fának \Rightarrow ne hagyjuk őket „lesülyyedni”

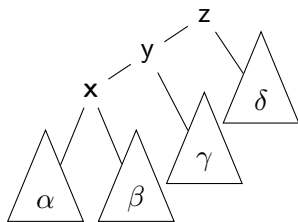
Lehetséges hátránya

Mivel még egy (sikertelen) KERES művelet is változtat a fa struktúráján, így többszálú használata problémás.

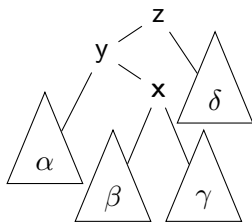


Splay tree – gyökérig való forgatások

- Ha a gyökérbe juttatni kívánt x csúcs már a jelenlegi y gyökérelem bal (jobb) fia, akkor egyszerűen forgassunk y körül jobbra (balra)
- Egyébként 2 lehetőség van (és ezek szimmetrikusai) x gyökérbe juttatására



- 1 z csúcs körül jobbra majd
- 2 y csúcs körül jobbra forgatunk



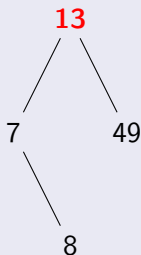
- 1 y csúcs körül balra majd
- 2 z csúcs körül jobbra forgatunk



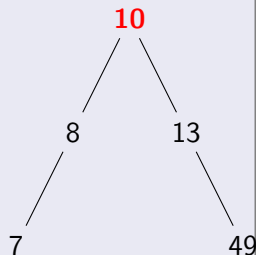
Eredeti fa



KERES(21) után



BESZÚR(10) után



- Amortizált költségelemzéssel az adatszerkezetek hosszú távú legrosszabb esetbeli viselkedését modellezhetjük
- Amortizált költségelemzés szempontjából a Fibonacci-kupac a megismert leghatékonyabb kupac
- Egyes gráfalgoritmusok implementálásához kifejezetten hasznos választás lehet

