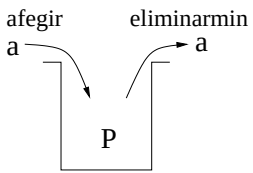
El TAD Cua de Prioritat

- Tipus abstracte de dades que té en compte l'extracció (eficient) del mínim (o màxim) d'un conjunt d'elements (comparables).
- Les operacions bàsiques són: afegir, eliminarmin i consultarmin.

$$\begin{aligned} \text{afegir} &: \text{CuaP}[a] \times a \longrightarrow \text{CuaP}[a] \\ \text{eliminarmin} &: \text{CuaP}[a] \longrightarrow \text{CuaP}[a] \\ \text{consultarmin} &: \text{CuaP}[a] \longrightarrow a \end{aligned}$$

- Normalment, les operacions modifiquen una única cua i tornen o accepten un element del tipus de base, a.



El TAD Cua de Prioritat

Com a mètodes d'una classe en llenguatges orientats a objectes se solen implementar com a:

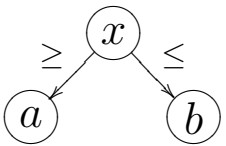
```
public interface PriorityQueue
{
void       insert( Comparable x );
Comparable findMin( )   throws Underflow;
Comparable deleteMin( ) throws Underflow;
void       makeEmpty( );
boolean    isEmpty( );
}
```

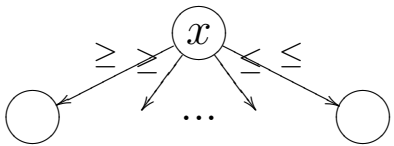
Implementació. Monticles

Els monticles són una alternativa (amb moltes variants) per tal d'implementar cues de prioritat de manera eficient.

Els podriem definir com estructrures arborescents parcialment ordenades en les quals l'element mínim (o màxim) està directament accessible.

Aquesta ordenació parcial consisteix en què els "descendents" de tot element són major o iguals que ell.



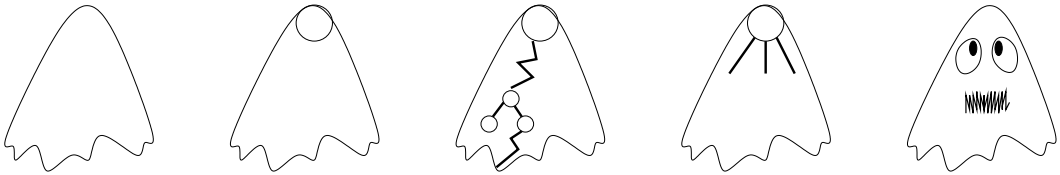


Implementació. Monticles

S'anomena **propietat de monticle** al fet que un node siga menor o igual que els seus descendents.

Un monticle (en general) és un arbre que compleix la propietat de monticle en tots els seus nodes.

Les definicions concretes de monticles inclouen algun tipus de condició d'equilibri i un ordre fix per als arbres.

Gràficament, 

Propietats generals dels monticles

La propietat de monticle implica que el mínim sempre es troba a l'arrel.

La propietat es pot restablir de manera eficient en un monticle si només s'ha perdut en un node.

Aquesta última propietat permet la implementació eficient d'operacions en monticles (afegir, eliminarmin, etc) i també la construcció incremental de monticles.

"Pujar" i "Baixar" en monticles

Siga x l'únic node d'un monticle que no compleix la propietat.

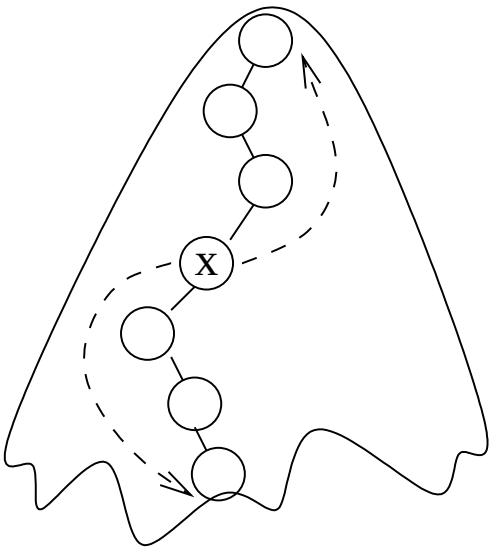
si x menor que son pare: s'intercanvia x amb son pare i es té (en el cas pitjor) la mateixa situació de partida però un nivell més amunt.

si x és major que algun dels seus fills: s'intercanvia x amb el **menor** dels fills i es té (en el cas pitjor) la mateixa situació de partida (només en aquest node!) un nivell més avall.

Qualsevol dels dos casos implica el recorregut d'una única branca de l'arbre en sentit ascendent o descendent.

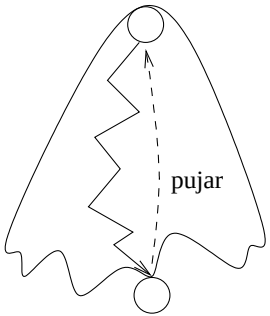
"Pujar" i "Baixar" en monticles

Les operacions corresponents s'anomenen **pujar** i **baixar**, respectivament i el seu cost serà logarítmic si el monticle està equilibrat.

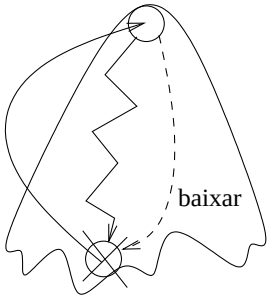


Operacions en monticles

Afegir:
- recórrer qualsevol branca i afegir un node al final.
- restablir la propietat sobre aquest node.



Eliminarmin:
- recórrer qualsevol branca fins a una fulla i eliminar-la.
- substituir l'arrel per la fulla eliminada.
- restablir la propietat sobre l'arrel.

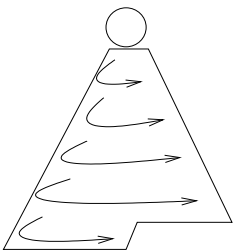


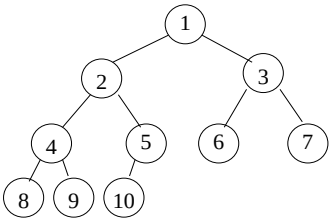
Monticles Binaris (MB)

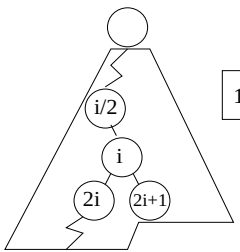
Un cas particular especialment interessant són els monticles binaris.

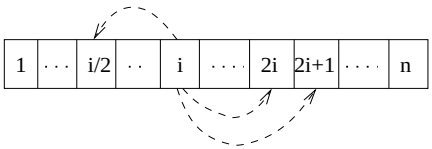
De les moltes maneres de conseguir l'equilibri, s'exigeix que els arbres siguen **complets** (tots els nivells estan complets llevat de l'últim on totes les fulles s'agrupen en un bloc contigu a l'esquerra).

Aquesta propietat permet la numeració de nodes per nivells i la implementació eficient mijantçant un vector. Gràficament,









Pujar i baixar en MB

```
x = array[ hole ];
int hole = ++currentSize;
for( ; x.lessThan( array[ hole / 2 ] ); hole /= 2 )
    array[ hole ] = array[ hole / 2 ];
array[ hole ] = x;
```

```
Comparable tmp = array[ hole ];
for( ; hole * 2 <= currentSize; hole = child )
{
    child = hole * 2;
    if( child != currentSize &&
            array[ child + 1 ].lessThan( array[ child ] ) )
        child++;
    if( array[ child ].lessThan( tmp ) )
        array[ hole ] = array[ child ];
    else
        break;
}
array[ hole ] = tmp;
```

Operacions en MB

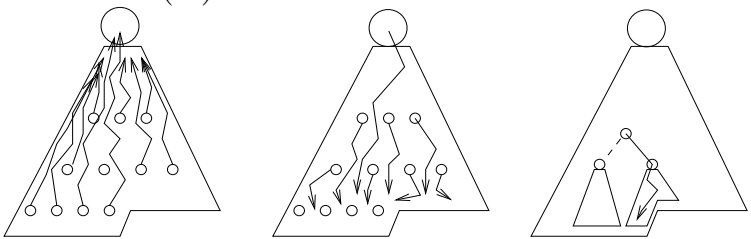
Afegir: no cal recórrer la branca. S'accedeix directament a la posició $n+1$.

Eliminarmin: només cal reduir en 1 la talla i intercanviar l'n-èssim element pel primer.

Construcció d'un monticle a partir de n elements: Es pot afegir n vegades (es fa servir "pujar"). Cost $O(n \lg n)$.

Alternativa ascendent: Es crida $n/2$ vegades a la funció "baixar" sobre monticles cada vegada més grans. Cost $O(n)$.

El cost "real" és en els dos casos $\theta(n)$!



Monticles binaris (resum)

- Els monticles binaris són molt eficients en temps i en memòria.
- Es poden implementar amb vectors extensibles.
- Són la millor opció per tal d'implementar cues de prioritat amb operacions afegir i eliminarmin.

El problema apareix quan es volen considerar altres operacions. En particular la **unió** o **mescla**.

Exercici: Trobar una operació de mescla de monticles binaris (estàtics o dinàmics) eficient.

Monticles mesclables

Hi ha aplicacions en les quals és necessaria la mescla eficient de cues de prioritat.

A més a més, si es disposara d'una mescla de monticles vertaderament eficient, es podrien implementar les altres operacions en funció de la mescla:

afegir: unió d'un monticle amb un altre monticle unitari.

eliminarmin: S'elimina l'arrel i es mesclen els (sub)monticles esquerre i dret.

Tot açò justifica el disseny de tipus de monticles basats en la mescla eficient.

Monticles a esquerres (de esquerres, esquerrans, comunistes)

Si la mescla ha de ser eficient, haurem de pensar necessàriament en estructures dinàmiques (enllaçades). Suposem que tenim monticles binaris dinàmics:

- La unió es podria basar en afegir un pare per als dos monticles a unir.

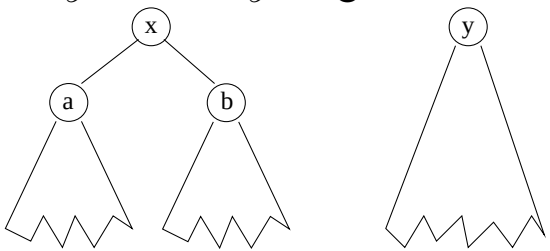
- La propietat de monticle es mantindria aplicant-hi la funció "baixar".

- La nova arrel es podria obtindre recorreguent qualsevol branca fins una fulla (com feiem en eliminarmin).

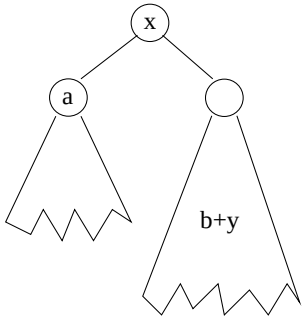
El problema és l'equilibri del monticle. Com es pot garantir?

Disseny recursiu d'un MB dinàmic

Siguen dos monticles, x i y on $x \leq y$ i siguen a i b els fills de x.



Està clar que l'arrel de $x + y$ ha de ser x. Quant als seus fills un podria no canviar (a) i l'altre es podria calcular com $b + y$ (subproblema recursiu!).



Disseny recursiu d'un MB dinàmic

Unir (x, y : monticles binaris)
<u>si</u> *buit*(x) <u>aleshores</u>
  tornar(y)
<u>si no si</u> *buit*(y) <u>aleshores</u>
  tornar(x)
<u>si no</u>
  <u>si</u> $x > y$ <u>aleshores</u> $x \leftrightarrow y$
  $dre(x) \leftarrow unir(dre(x), y)$
  tornar(x)
<u>fsi</u>

Cas millor (equilibri):
$T(n) = T(\frac{3n}{4}) + 1$

$T(n) \in \theta(\lg n)$

Cas pitjor: $\theta(n)$
(profunditat de l'arbre)

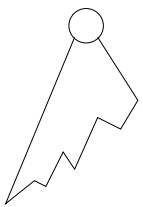
Disseny recursiu d'un MB dinàmic

En realitat, podem suposar que sempre es compleix que el fill esquerre sempre és menor que el dret (intercanvi de l'algorisme).

Aleshores, es fàcil veure que l'algorisme recursiu (lineal!) només recorre les **branques més a la dreta** dels arbres implicats.

Per tant: el cost no és en realitat proporcional a la profunditat (màxima) dels arbres sino a la longitut de la branca més a la dreta.

Monticles a esquerres: monticles binaris on la branca més a la dreta és logarítmica.



(Des)equilibri en monticles a esquerres

Longitud de camí nul, $lcn(x)$: nombre d'arcs fins al subarbre nul més pròxim. $lcn(\emptyset) = -1$ per definició.

Es compleix $lcn(x) = 1 + \min(lcn(esq(x)), lcn(dre(x)))$

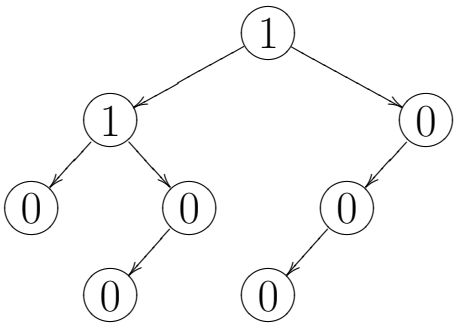
Monticle a esquerres (MESQ): monticle binari (no equilibrat) on es compleix $lcn(esq(x)) \geq lcn(dre(x))$ per a tot node x.

Es compleix necessàriament que els fills d'un MESQ també ho són.

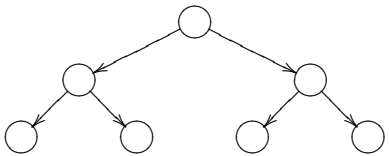
camí dret: camí més a la dreta. $lcd(x)$: longitud del camí dret.

En un MESQ es compleix que $lcn(x) = lcd(x)$.

(Des)equilibri en monticles a esquerres

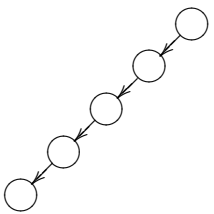


Els nodes contenen $lcn(x)$. La longitud de camí dret és 1.



pitjor MESQ

millor MESQ



(Des)equilibri en monticles a esquerres

En un MESQ amb n nodes i d nodes en el camí dret es compleix que $n \geq 2^d - 1$.

base d'inducció: un únic node té fills nuls. $1 \geq 2^1 - 1 = 1$
pas d'inducció: Siga x amb n nodes i d en el camí dret.

$dre(x)$ té n_D nodes i $d-1$ en el camí dret (hipòtesi d'I.), $n_D \geq 2^{d-1} - 1$.

Si x és MESQ el nombre de nodes en el camí dret de $esq(x)$ serà com a mínim $d-1$. Aleshores, $n_E \geq 2^{d-1} - 1$.

En resum, $n = 1 + n_E + n_D \geq 1 + 2 \cdot (2^{d-1} - 1) = 2^d - 1$

corol·lari: el camí dret d'un MESQ de talla n és menor o igual que $\lg(n+1)$.

Unió de MESQ

Com es pot garantir la propietat de MESQ alhora que es fa la mescla?

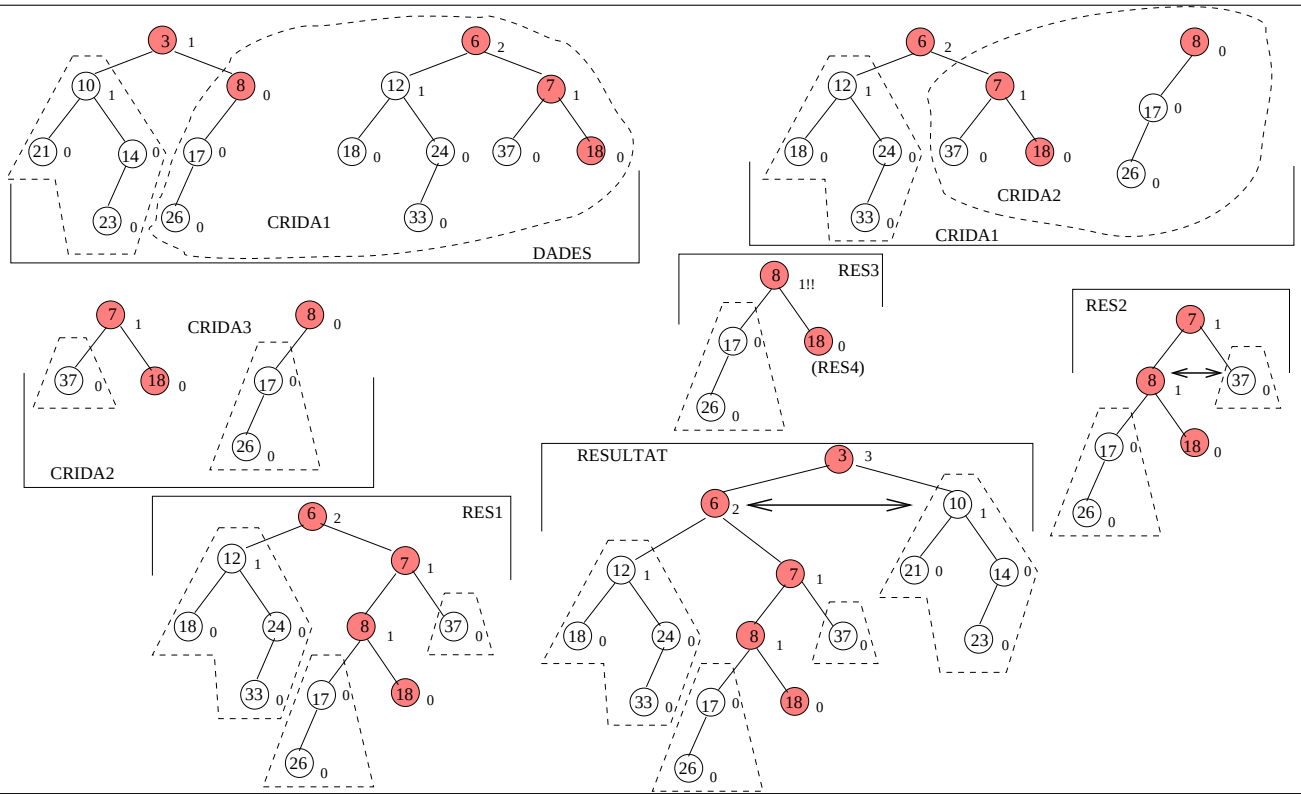
Només cal que el prodediment recursiu torne (implícitament o explícitament) la lcn.

Si després de recollir el resultat d'una crida no es compleix la propietat es torna l'arbre resultat però intercanviant els fills esquerre i dret.

la $lcn(x)$ en cada crida es recalcula sempre com a
$lcn(x) = \min[lcn(esq(x)), lcn(dre(x) + y)]$

Com a molt es fa un intercanvi de fills per crida recursiva. El cost és proporcional al nombre de crides (camins drets). Logarítmic en el pitjor cas!

Unió de MESQ. Exemple



MESQ. Implementació iterativa

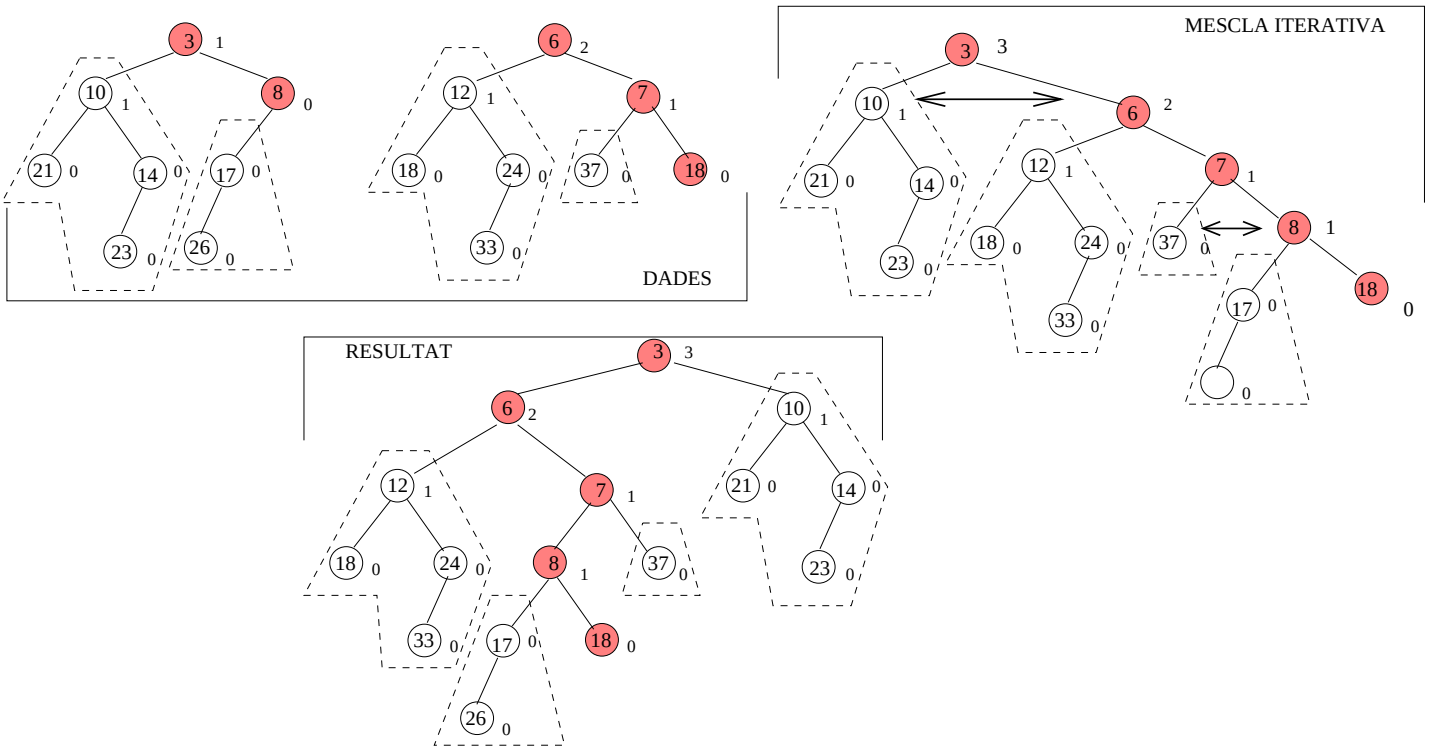
Encara que no final, la recursió del procediment "unir" es pot transformar fàcilment a iteratiu.

Si vegem els camins drets dels arbres com una llista ordenada de nodes que contenen com a informació els corresponents subarbres esquerres, la unió de MESQ es correspon exactament amb la mescla de dues llistes ordenades (mergesort).

L'única diferència és la propietat i el càlcul de les lcn.

Una vegada mesclades les "llistes" es recorre la branca al revés i es recalculen les lcn i s'intercanvien fills (si cal).

Unió iterativa. Exemple



Monticles Oblicus (o esbiaixats)

MOB: versió autoajustable dels MESQ.

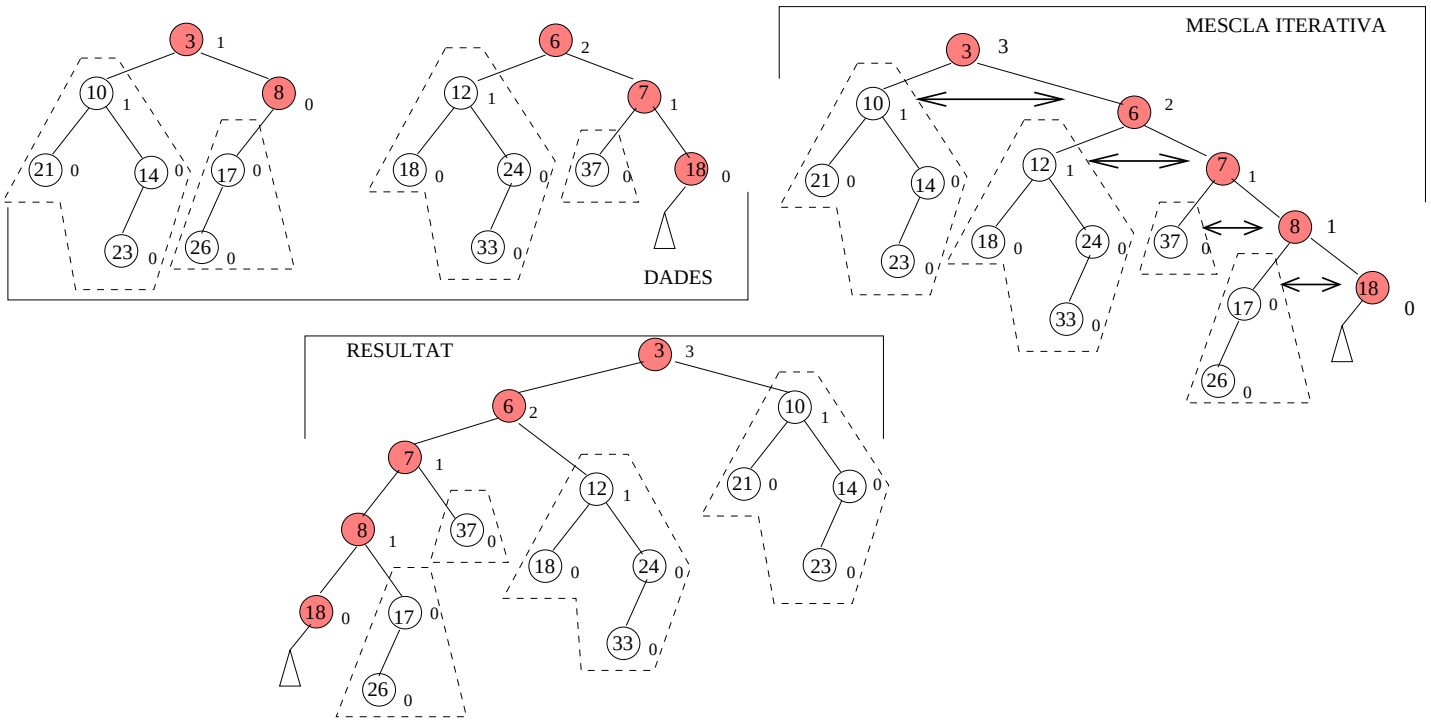
No es calculen ni s'emmagatzemen les lcn.

La unió es fa exactament com en el cas dels MESQ però s'intercanvien els fills esquerre i dret SEMPRE! (llevat de l'últim més profund).

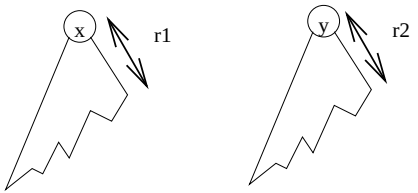
Ara ja no estan garantits els camins drets logarítmics. El cost (pitjor) pot ser (és) lineal!

Es pot garantir un cost amortitzat logarítmic?

MOB. Exemple

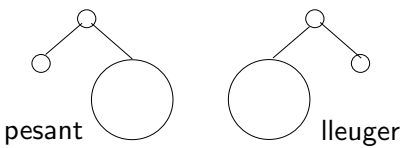


Anàlisi amortitzada dels MOB



El cost real serà proporcional a la suma dels camins drets, $r_1 + r_2$

Un node x s'anomena **pesant** si $|esq(x)| < |dre(x)|$



- Els únics nodes que poden passar de pesants a lleugers o al revés són els dels camins drets.
- Les fulles són sempre lleugeres.
- El nombre de nodes lleugers en el camí dret és $1 + \log n$ com a molt.

Anàlisi amortitzada dels MOB

El nombre de nodes lleugers en el camí dret és $1 + \log n$ com a molt.

Siga $L(n)$ el nombre de nodes lleugers en el camí dret d'un MOB x de n nodes.

El cas pitjor és que x siga lleuger. Aleshores $|dre(x)| \leq \frac{n}{2}$.

Per tant, una fita superior per a $L(n)$ vindrà donada per la recurrència:

$$L(n) = L(\frac{n}{2}) + 1$$

la solució de la qual és $1 + \lg n$.

* Açò ho compleix tot AB!

Anàlisi amortitzada dels MOB

Potencial: nodes pesants en els monticles implicats.

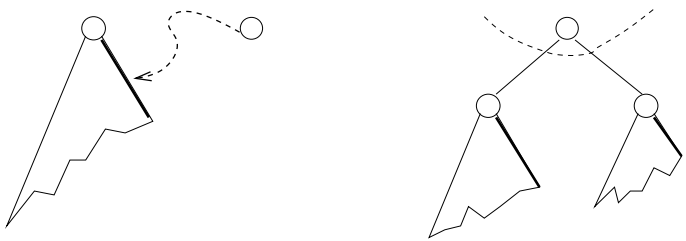
Donats dos monticles, siguen p_1, p_2, ℓ_1, ℓ_2 els nombre de nodes pesants i lleugers en els respectius camins drets. Cost de la unió: $p_1 + \ell_1 + p_2 + \ell_2$.

- els nodes pesants **sempre** esdevenen lleugers.
- els nodes lleugers, potser si o potser no (per la igualtat).
- Cas pitjor: tots els lleugers esdevenen pesants!

$$\Delta\phi \leq \underbrace{(\ell_1 + \ell_2)}_{\text{n. de pesants després}} - \underbrace{(p_1 + p_2)}_{\text{n. de pesants abans}}$$

$$\hat{c}_u = c_u + \Delta\phi = [p_1 + \ell_1 + p_2 + \ell_2] + [\ell_1 + \ell_2 - (p_1 + p_2)] = 2(\ell_1 + \ell_2) \in O(\lg n)$$

MOB (MESQ): les altres operacions



afegir: es recorre la branca dreta i s'insereix el nou node en el lloc que li toque. MESQ: es recalculen les lcn per d'alt i s'intercanvien fills si cal. MOB: s'intercanvien els fills de $tots$ els nodes en el camí dret.

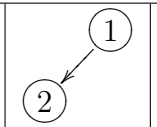
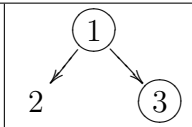
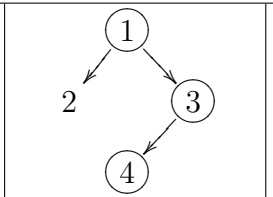
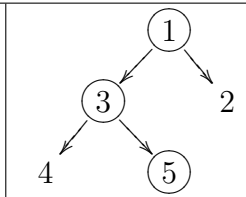
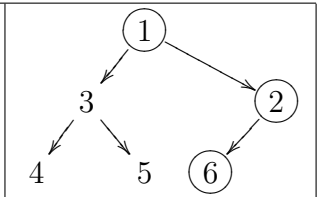
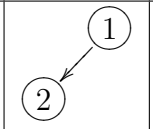
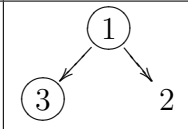
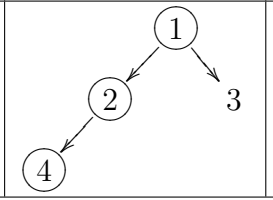
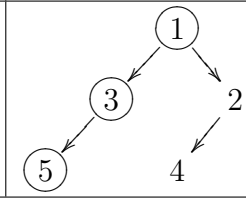
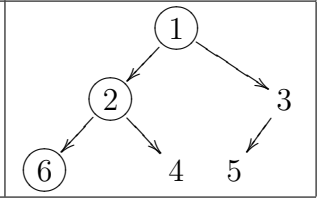
eliminarmin: s'elimina l'arrel i apareixen dos monticles que s'unieixen. MESQ: els dos fills són necessàriament MESQ i per tant els seus camins drets seran logarítmics. MOB: els fills no han de complir cap condició.

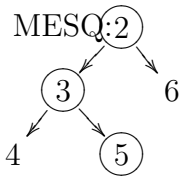
MOB (MESQ): les altres operacions

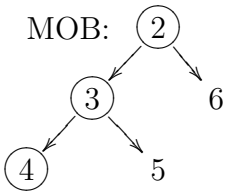
Anàlisi amortitzada d'afegir: es tracta (exactament) d'una unió de dos MOB.

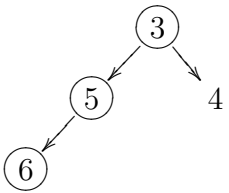
Anàlisi amortitzada d'eliminarmin: L'única diferència és el cost constant d'eliminar l'arrel i la modificació de $\Delta\phi$ en 1 pel fet que l'arrel puguera ser pesant. El cost amortitzat és logarítmic.

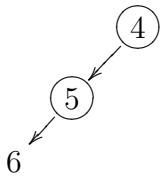
Exemples

MESQ					
MOB					

MESQ: 

MOB: 





MOB (MESQ): Conclusions

El cost amortitzat de totes les operacions sobre els monticles oblicus és logarítmic.

	MB pitjor	MESQ pitjor	MOB amortitzat
afegir	$\lg n$	$\lg n$	$(\lg n)$
eliminarmin	$\lg n$	$\lg n$	$(\lg n)$
unir	n	$\lg n$	$(\lg n)$

Monticles Binomials (MBINO)

(RESUM)

- *Arbres binomials: una forma capritxosa d'equilibri*
- *Unió d'arbres binomials*
- *Arbres i boscos: monticles binomials*
- *Unió de MBINO*
- *Fites logarítmiques associades als MBINO*
- *Notes sobre la implementació dels MBINO*
- *El cost amortitzat d'***afegir** *és constant!*
- *L'operació* **eliminarmin**. *Implementació.*

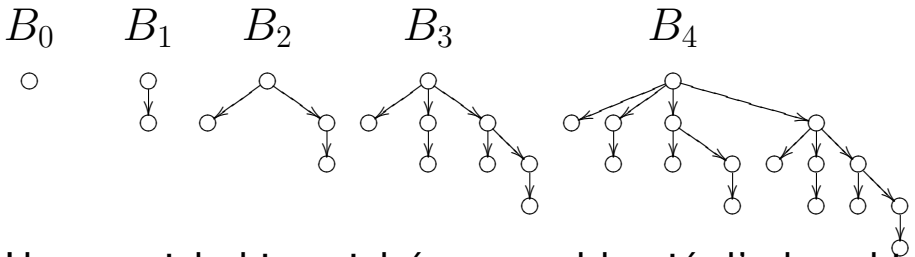
Monticles Binomials (MBINO)

Arbre binomial d'ordre k (B_k): una arrel amb k subarbres que són arbres binomials d'ordres $0, 1, 2, \ldots, k-1$.

Un arbre binomial d'ordre 0 és un únic node.

- $|B_k| = 2^k$
- $h(B_k) = k + 1$
- nodes en B_k a profunditat i: $\binom{k}{k-i-1}$,

Monticles Binomials (MBINO)



Un monticle binomial és una col·lecció d'arbres binomials d'ordres diferents on es compleix la propietat de monticle.

Si hi ha n elements, aleshores $n = c_0 2^0 + c_1 2^1 + \cdots + c_m 2^m$.

La profunditat d'un monticle de n elements és $\lfloor \lg n \rfloor + 1$

i el nombre d'arbres també

Monticles Binomials (MBINO)

La unió de dos arbres binomials d'ordre k és trivial: només cal afegir un d'ells com a $k+1$-èsim fill de l'altre per obtindre un nou arbre binomial d'ordre $k+1$.

unió de monticles binomials: unió dels seus arbres de menor a major ordre. Cada unió d'arbres dóna lloc a arbres d'ordre superior.

L'algorisme és equivalen a la suma de nombres binaris

El cost de la unió de monticles és lineal en el nombre d'arbres $\Rightarrow$ logarítmic

Monticles Binomials (MBINO)

afegir: és la unió de dos monticles binomials un dels quals només té un element.

eliminarmin: Si s'elimina el mínim d'un arbre d'ordre k, el que queda (els fills) constitueix un monticle de $2^k - 1$ elements. Només caldria unir aquest monticle amb el monticle resultant d'eliminar l'arbre d'ordre k i cercar el mínim entre les arrels.

unir, afegir, i eliminarmin són logarítmiques en els MBINO.

Es pot demostrar que l'operació afegir té un cost amortitzat constant!

Monticles Binomials Pereosos (MBINOP)

(RESUM)

- *Una versió relaxada dels MBINO.*
- *MBINOP: un nombre no necessàriament logarítmic d'arbres de profunditats logarítmiques.*
- **afegir** *i* **unir** *són constants però* **eliminarmin** *és lineal!*
- *Implementació d'***eliminarmin***: l'operació privada* **reorganitzar**.
- *Anàlisi amortitzada.*

Eliminamin en MBINOP

Algorisme Eliminamin (M : llista d'arbres)
Mètode:

a=Llegirmin(M)	// s'apunta el mínim
M'=Convertir(fills(a))	// convertir llista de fills en monticle
EliminaNode(a)	// s'elimina a de la llista d'arbres
M=Unir(M, M')	// Unió pereosa de monticles (llistes)
Reorganitzar(M)	

Amb una implementació mijantçant llistes lligades, totes les operacions són constants llevat de la reorganització.

Reorganització de MBINOP

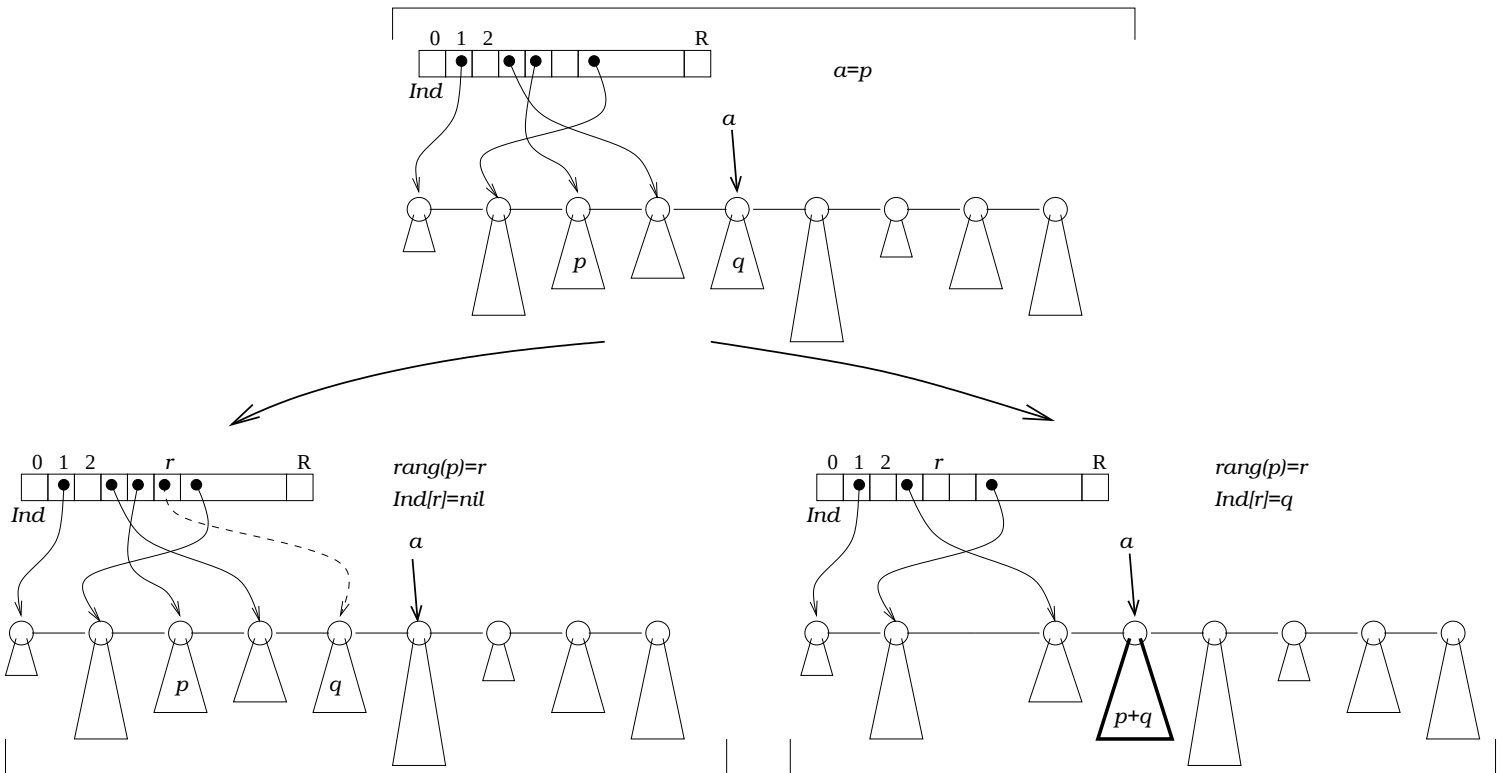
Es parteix d'un únic monticle format per arbres binomials resultat d'haver afegit els fills del mínim a la resta d'arbres. La col·lecció d'arbres no està ordenada.

L'algorisme es basa en un únic recorregut fent servir un vector d'apuntadors a arbres indexat per rangs.

En un moment donat, s'està sobre un determinat arbre i a l'esquerra no poden haver arbres de rangs repetits. Tots aquests arbres a l'esquerra són accessibles com a $Ind[r]$ (si és que algun d'ells té rang r).

En cada pas, l'algorisme: o avança si no hi ha arbres a l'esquerra del mateix rang de l'actual; o fa la unió si no. En tots dos casos, s'actualitza l'apuntador corresponent.

Reorganització de MBINOP



Reorganització de MBINOP

Algorisme Reorganitzar (M : llista d'arbres)
Aux: Ind : vector $[1..R]$ de Arbres // Inicialment, $Ind[r]$=nil, $0 \leq r \leq R \leq \lg |M|$
Mètode:

```
a=primer(M)                    // s'inicia l'apuntador que recorre M
mentre (a ∈ M) fer
    r=rang(a)
    si Ind[r] ∈ M fer
        a=unir(Ind[r], a)
        Ind[r]=nil             // ja no hi haurà arbre de rang r a l'esquerra
    sino
        Ind[r]=a               // serà l'únic arbre de rang r a l'esquerra
        a= seg(a)              // s'avança al següent arbre
    fsi
fmentre
```

Reorganització de MBINOP

- L'algorisme recorre una única vegada la col·lecció d'arbres però el nombre d'iteracions pot ser, com a molt, dues vegades el nombre inicial d'arbres.
- Com que totes les operacions són constants el cost serà de l'ordre del nombre d'arbres.
- El cost espacial és logarítmic respecte del nombre d'arbres (pel vector Ind).

El cost en el pitjor cas és lineal!

Reorganització de MBINOP

Anàlisi amortitzada. Potencial = nombre d'arbres en els monticles.

l'única operació no trivial és **eliminarmin**.

Si inicialment hi ha A arbres i el mínim es troba en un arbre d'ordre r, hi haurà $A + r - 1$ arbres abans de la reorganització i A' després ($1 \leq A' \leq \lg n$). La diferència de potencial serà $A' - A \leq \lg n - A$.

El cost de la reorganització és $A + r - 1$ i la cerca del mínim costa A'.

$$\hat{c}_i = c_i + \Delta\phi_i = A + r - 1 + A' + (A' - A) = 2A' + r - 1 \leq 3 \lg n$$

MBINOP. Resum

Noves operacions sobre cues de prioritat: **eliminar** i **decrementar**

	MB	MESQ	MOB		MBINO		MBINOP	
	pitjor	pitjor	pitj.	amort.	pitj.	amort.	pitj.	amort.
afegir	$\lg n$	$\lg n$	n	$\lg n$	$\lg n$	$O(1)$	1	
eliminarmin	$\lg n$	$\lg n$	n	$\lg n$	$\lg n$		n	$\lg n$
unir	n	$\lg n$	n	$\lg n$	$\lg n$		1	
eliminar	$\lg n$	$\lg n$	n	$\lg n$	$\lg n$		n	$\lg n$
decrementar	$\lg n$	$\lg n$	$\lg n$		$\lg n$		$\lg n$	

Monticles de Fibonacci (MFIBO)

Una versió més relaxada dels Monticles Binomials Pereosos:

- L'objectiu és millorar l'operació `decrementar`.
- En lloc de recórrer una branca per reorganizar després de modificar una clau la proposta consisteix a convertir el subarbre corresponent en un nou arbre del monticle.
- $\uparrow$ el cost seria constant!
- $\downarrow$ el nombre d'arbres augmenta (encara més)!
- $\downarrow\downarrow$ els arbres ja no poden ser binomials!
  Exercici: Valdria encara l'anàlisi amortitzada dels MBINOP en aquest cas?

Monticles de Fibonacci (MFIBO)

Monticle de Fibonacci (F-Monticle): Col·lecció d'arbres **no ordenats** que compleixen la propietat de monticle.

- S'associa a cada node (arbre) un **rang** (nombre de fills) de manera que dos arbres del mateix rang es poden unir per formar un arbre de rang una unitat superior.
- En lloc d'imposar una estructura als arbres de manera explícita es dissenyaran les operacions per tal que:
  a) els nodes no perden massa fills, b) decrementar siga (quasi) constant.

Operacions sobre els F-monticles

- Si un node perd un fill, s'ha de marcar.
- Si un node marcat per un fill (el segon) cal convertir-lo en un nou arbre del monticle.
- Això implica que son pere perd un fill i podria (si estiguera marcat) ser també tallat.

Això és el que es coneix com **talls en cascada**.

Inicialment els nodes no estan marcats. Les arrels mai poden estar marcades.

Ara l'operació decrementar té un cost (pitjor) logarítmic (altra volta).

Operacions sobre els F-monticles

unió/afegir: pereosa, com els MBINOP.

eliminarmin: eliminació, unió de la lista de fills del mínim i reorganització. Ara podran haver-hi més arbres però la reorganització desfarà (en part) l'efecte dels talls sobre els arbres.

decrementar: talls en cascada mentre en el camí a l'arrel es troben nodes marcats.

eliminar: decrementar + eliminarmin

Quina relació hi ha entre el rang, el nombre de fills i la profunditat dels arbres?

Propietats

Siguen $y_1, \ldots, y_k$ els fills del node x en l'ordre en què han sigut afegits, aleshores

$$\text{rang}(y_1) \geq 0, \qquad \text{rang}(y_i) \geq i - 2 \;\; \forall i \geq 2$$

Considerem un dels nodes y_i. En el moment de la seua inserció s'ha de complir: a) $\text{rang}(x) = \text{rang}(y_i)$, b) $y_1, \ldots, y_{i-1}$ han de ser ja fills de x (juntament amb altres, posiblement).

Aleshores, el rang de y_i en eixe instant ha de ser major o igual que el nombre de fills que tinguera (major o igual que $i - 1$).

A partir d'ahí i fins a que x tinga els k fills definitius, y_i no pot perdre **més** d'un fill (sense ser tallat!). Per tant, $\text{rang}(y_i) \geq i - 2$

Propietats

Si rang$(x) = k$ aleshores $|x| \geq F_{k+2} \geq \phi^k$

F_k és k-èssim nombre de Fibonacci

Anomenem t_k la menor talla de qualsevol node de rang k.

Com que tot x està format per k subarbres que compleixen la propietat anterior:

$$|x| = \Sigma_{i=1}^{k} |y_1| \geq t_0 + \Sigma_{i=2}^{k} t_{i-2}$$

en particular, el mínim (t_k):

$$t_k \geq \begin{cases} 1 & k = 0 \\ 2 & k = 1 \\ 2 + \Sigma_{i=2}^{k} t_{i-2} & k > 1 \end{cases}$$

Propietats

La solució de la recurrència ens donarà doncs una fita inferior per a t_k.

Aquesta recurrència es pot transformar en la de Fibonacci, $t_k = t_{k-1} + t_{k-2}$, però amb diferents condicions inicials. Es compleix,

$$t_k = F_{k+2} \geq c \cdot \phi^k$$

Corol·lari: Si rang$(x) = k$ en un F-monticle de n elements, aleshores

$$k \leq c' \cdot \lfloor \log_\phi n \rfloor \in O(\lg n)$$

Anàlisi. Operacions crítiques

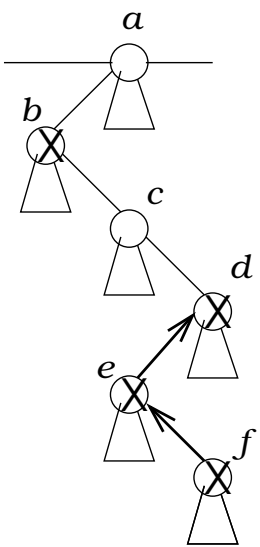
decrementar: El cost dels talls en cascada serà una quantitat C entre 1 i $\lg n$ que és igual al nombre de nodes marcats seguits en la branca de l'element concret.

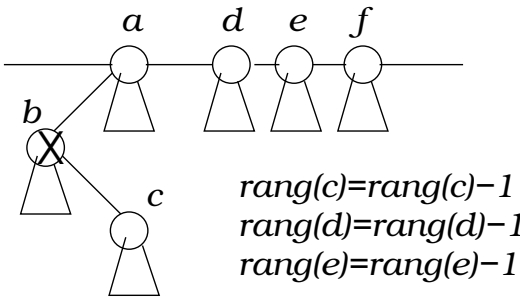
Cost pitjor: $\lg n$ (millor: $O(1)$)

eliminarmin: Es tenen A arbres ($1 \leq A \leq n$). L'arbre que conté el mínim (de rang r) se substitueix per els seus r subarbres. Després de la reorganització, no pot haver-hi dos arbres amb el mateix rang. Això implica, pel corol·lari anterior que el nombre final d'arbres, A_f, serà logarítmic. El cost serà proporcional al nombre total d'arbres que cal reorganitzar, $A + r - 1$.

Cost pitjor: n (millor: $\lg n$)

Anàlisi. Operacions crítiques





Anàlisi amortitzada

O "Fibonacci Jones a la recerca del potencial perdut"

Potencial: nombre d'arbres, A (com en els MBINOP)

decrementar: Com més es recorre la branca, més (nous) arbres apareixen:

$$\hat{c}_i = c_i + \Delta\phi_i = C + C = 2C \in O(\lg n)$$

eliminarmin: Com que el nombre final de nodes, A_f és logarítmic,

$$\hat{c}_i = c_i + \Delta\phi_i = A + r - 1 + [A_f - A] = r + A_f - 1 \in O(\lg n)$$

Aquestes fites no representen cap diferència respecte dels MBINOP!

Anàlisi amortitzada

Potencial: nombre de nodes marcats, N

decrementar: Com més es recorre la branca, més marques desapareixen (perque esdevenen arrels):

$\hat{c_i} = c_i + \Delta\phi_i = C + [(N - C) - N] = 0 \in O(1)$

eliminarmin: La reorganització no modifica marques. Però, alguns dels r fills del mínim podrien estar marcats i perdre les marques:

$\hat{c_i} = c_i + \Delta\phi_i = A + r - 1 + [(N - r') - N] \leq A + r \in O(n)$

S'ha millorat la fita sobre la primera però s'ha empitjorat (massa) la segona!

Anàlisi amortitzada

Potencial: Combinació lineal dels anteriors, $A + 2N$.

decrementar: Com més es recorre la branca, més marques desapareixen (perque esdevenen arrels) i arbres apareixen:

$$\hat{c_i} = c_i + \Delta\phi_i = C + [(A + C) + 2(N - C) \ - \ (A + 2N)] = 0 \in O(1)$$

eliminarmin: Es poden perdre les marques de r' fills del mínim i el nombre final d'arbres, A_f és logarítmic:

$$\hat{c_i} = c_i + \Delta\phi_i = A + r - 1 + [A_f + 2(N - r') - (A + 2N)] = A_f - r - 1 \in O(\lg n)$$

Resum final

	MB	MESQ	MOB	MBINO	MBINOP	MFIBO
	pitjor	pitjor	pitj. amort.	pitj. amort.	pitj. amort.	pitj. amort.
afegir	$\lg n$	$\lg n$	n $\lg n$	$\lg n$ $O(1)$	1	1
eliminarmin	$\lg n$	$\lg n$	n $\lg n$	$\lg n$	n $\lg n$	n $\lg n$
unir	n	$\lg n$	n $\lg n$	$\lg n$	1	1
eliminar	$\lg n$	$\lg n$	n $\lg n$	$\lg n$	n $\lg n$	n $\lg n$
decrementar	$\lg n$	$\lg n$	$\lg n$	$\lg n$	$\lg n$	$\lg n$ 1

Existeixen alternatives més eficients (en la pràctica) al F-Monticle (els *pairing heaps*).

Aplicacions dels F-Monticles

Entre les aplicacions més famoses dels F-Monticles estan els algorismes de Prim (per a l'obtenció de l'arbre d'extensió minimal graf) i Dijkstra (per l'obtenció dels camins més curts des d'un mateix origen) sobre grafs ponderats.

En tots dos algorismes (golafres), cal fer servir una cua de prioritat d'arcs on la prioritat són pesos de camins o arcs en el graf.

En cada iteració cal decrementar per tal d'actualitzar el conjunt d'eleccions factibles per construir la solució global.

El cost constant de decrementar resulta esencial per tal d'obtenir un cost global (en els dos casos) proporcional a $a + n \lg n$.