

2015年10月5日 (月)

Diccionarios en memoria secundaria.

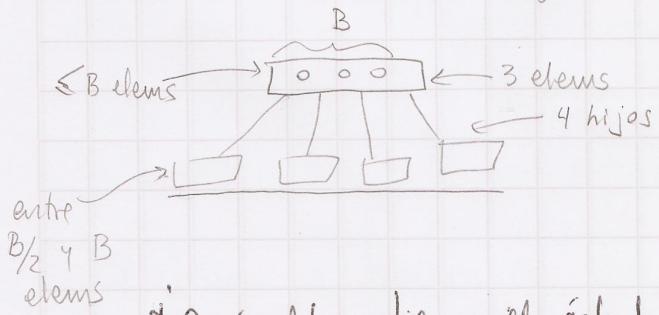
B-Tree	: insertar	sucesor
	: borrar	predecesor
	: buscar	↳ iterar en orden...

Idea básica: generalizar un árbol binario

• Cada nodo tiene tamaño B (unidad mínima de I/O)

• Invariantes:

- cada nodo almacena al menos B elementos
- salvo la raíz, cada nodo tiene al menos $B/2$ hijos
- un nodo interno con k elementos tiene $k+1$ hijos
- todas las claves en el i -ésimo hijo ($1 \leq i \leq k+1$) están, en valor, entre las claves de los elementos $i-1$ e i .
- todas las hojas tienen la misma profundidad.



¿Qué altura tiene el árbol? $h = \Theta(\log_B N)$

- Invariantes de la estructura:

- a) Cada nodo almacena a lo más B elementos.
- b) Salvo la raíz, cada nodo tiene al menos $B/2$ elementos.
- c) Un nodo interno con k elementos tiene $k+1$ hijos.
- d) Todas las claves en el i -ésimo hijo ($1 \leq i \leq k+1$) están, en valor, entre las claves de los elementos $i-1$ e i .
- e) Todas las hojas tienen la misma profundidad.

- ¿Qué altura tiene el árbol?

Debido a que todas las hojas están a la misma profundidad, se deduce que $h = O(\log_B N)$.

- Los algoritmos:

- a) Busqueda : (k)

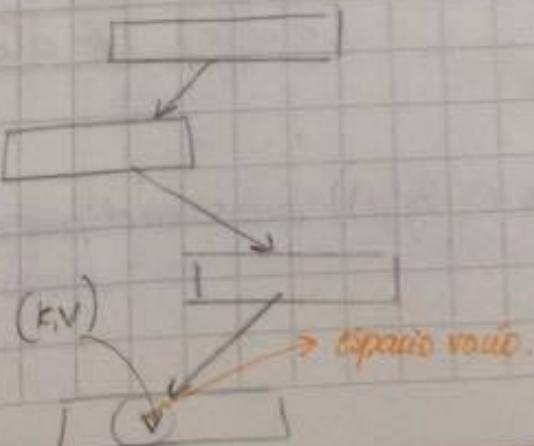
↳ Bajar por el árbol eligiendo el hijo adecuado en cada vez (toma $O(\log_B N)$).

Lo complicado es insertar / Borrar.

- a) insertar : (k,v)

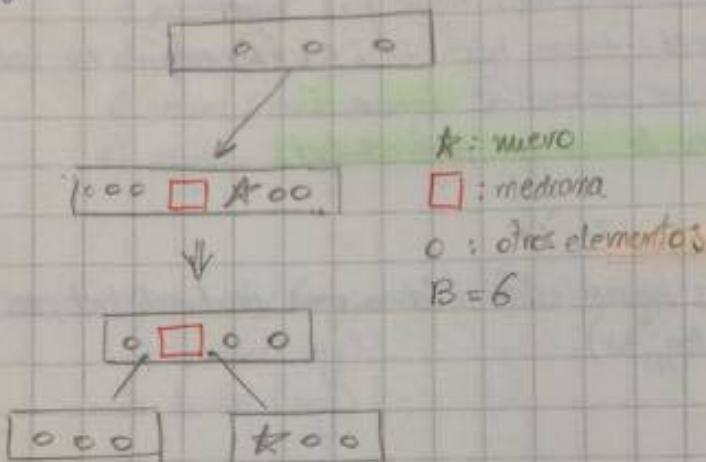
↳ Se busca la clave a insertar; al no encontrarse, encontramos el lugar donde debería estar (en una hoja). $\{O(\log_B N)\}$

a)



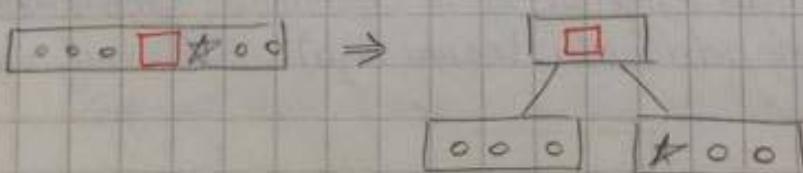
$O(\log n)$

- b) Se inserta el dato en la hoja. Si la hoja queda con $B+1$ elementos, tenemos un "overflow". Entonces dividimos la hoja en dos y se promueve la mediana al padre.



- ↳ Se agregó un elemento al padre, luego, puede recursivamente haber overflow en los ancestros de la hoja que tuvo problemas.

Si la raíz sufre overflow, se crea una nueva raíz de 1 sólo elemento, y por ende, la altura del árbol crece en 1.

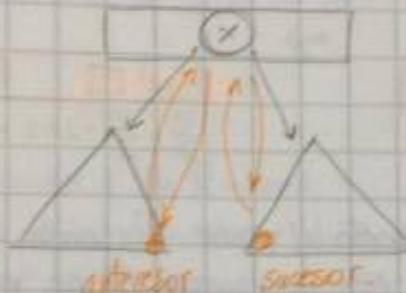


⇒ Tenemos que insertar es $O(\log_B N)$ incluso en el peor caso.

3) Borrado (e):

Buscar y encontrar la clave.

- Si es un nodo interno, se intercambia el elemento con su sucesor o antecesor (que estará en una hoja).

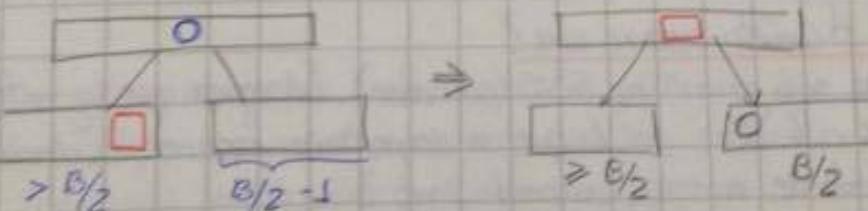


⇒ sin pérdida de generalidad, el borrado **ocurre en las hojas**.

- En cambio, si tenemos que borrar en una hoja.
Se borra, y si la hoja queda con menos de $B/2$ elementos.

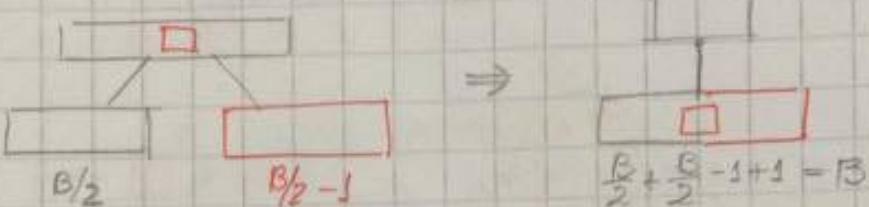
↳ Pido los vecinos y le pido un elemento a alguno :

En el caso del izquierdo:



Análogo en el caso de vecino derecho. Esto funciona siempre y cuando cada bloque tenga más de $B/2$ elementos.

- Si ambos tienen $\frac{B}{2}$ elementos, no puedo pedirle elementos. Entonces elijo uno de ellos y me fusiono con él (y con el elemento "separador" en el padre).



Notemos que le quitamos un elemento al padre. Por ende, el padre puede sufrir underflow. \Rightarrow Esto recursivamente se va propagando a la raíz.

- Como la raíz no tiene restricciones, fusionamos y creamos una raíz más grande.



\Rightarrow De nuevo es $O(\log_B N)$.

• Problemas: ¿Cuánto espacio usa?

la estructura sólo garantiza 50% (por caso). [Asegura que la mitad de los nodos está llena, por lo que se desperdicia espacio, es decir, si tenemos 1 mb de números, ocuparía 2 mb.]

El caso promedio $\approx 69\%$ (similar a logaritmo (2)).

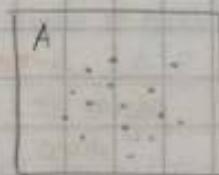
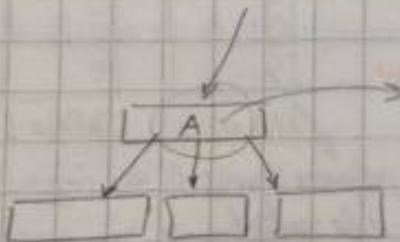
• Una variante útil son los Bt árboles.

- ↳ Todos los elementos (datos) sólo están en las hojas. En los nodos internos tenemos sólo punteros ("copias" de llaves).
- ↳ Las hojas tienen punteros entre ellas, por lo que se puede hacer un recorrido secuencial a través de ello.

R-trees

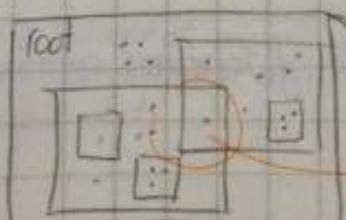
- ↳ Son muy parecidos a los R-trees.

- 1) Cada nodo interno almacena hasta B rectángulos.
- 2) Salvo la raíz, todo nodo interno tiene al menos $B/2$ rectángulos.
- 3) Todas las hojas tienen igual profundidad.
- 4) Las hojas almacenan la info geométrica (valores).
- 5) Si tenemos k rectángulos \rightarrow hay k hijos.
- 6) Cada rectángulo cubre los rectángulos de sus hijos. (totalmente)



↳ La raíz tendrá el bounding box (el rectángulo mayor) que cubre a todos los demás.

- Al insertar, uno debe escoger un criterio para ver por qué hijo hay que descender. (Puede ser por el que crece menos en área, o bien, el que me hace más overlapping, es decir, el que me contenga más).



↳ Contiene a todos los demás rectángulos.

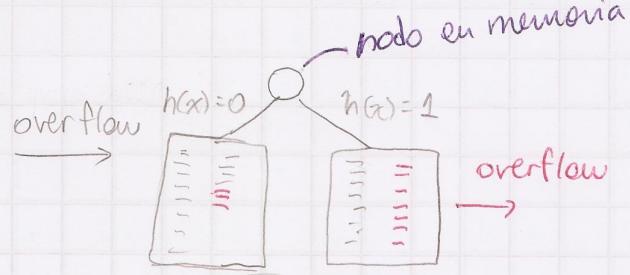
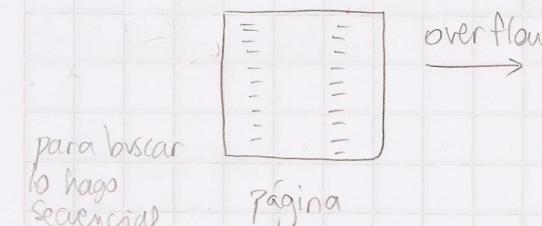
Se pueden solapar.

Hashing en Disco

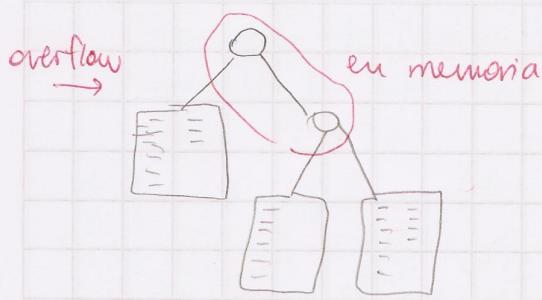
2015年10月6日(木)

- Hashing extendible → exige cierto espacio en mem. (desv: puede haber mucho espacio vacío en los bloques)
- Hashing lineal → no exige RAM

Hashing Extendible:



es esperable que cada pág. esté llena a la mitad

$$\sim M > \frac{2^n}{B}$$


Las hojas son punteros a disco
⇒ búsqueda = 1 acceso
(pues los nodos están en memoria.)

⇒ inserción = 3 accesos (?)

2 sin overflow: leer y escribir

3 con " " : leer y escribir 2 págs

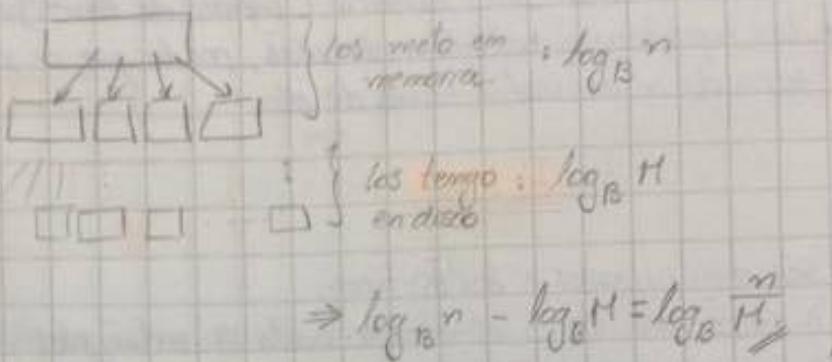
Si hay varios overflows seguidos, las páginas vacías las dejo NULL.

⇒ garantizado 50% ocupación

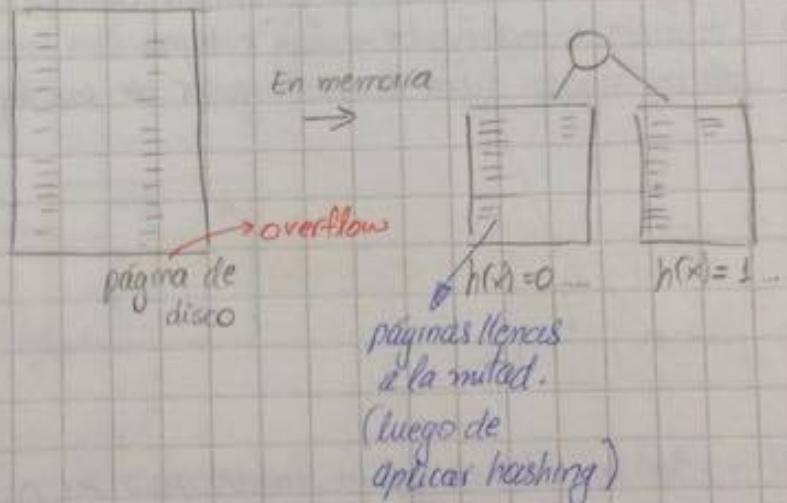
Hashing en disco

06/0ctubre/2015

- Hashing extendible
- Hashing lineal



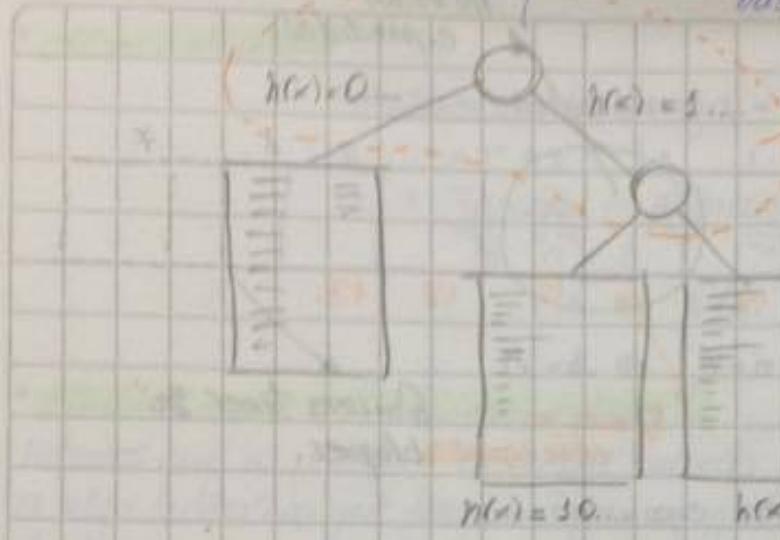
• Hashing extendible.



Luego, para ello necesitamos tener $M \geq \frac{2}{B} n$.

¿Qué pasa si seguimos insertando, y tenemos overflow?

* Tries (Arboles que permiten búsquedas de strings)



* Índice que está en memoria, y que tiene páginas (punteros a ellas).

(En promedio es $\sim 67\%$).

* Tenemos una ocupación de un 50% (la mitad de las páginas ocupadas)

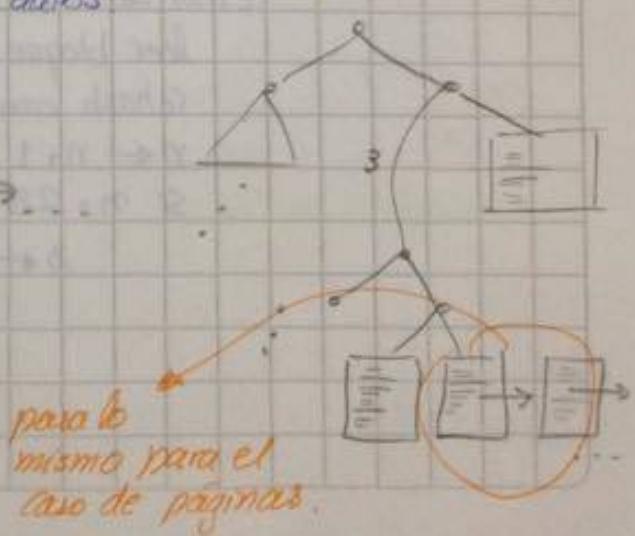
- Búsqueda en esta estructura es 1 acceso.
- Inserción son 3 accesos. (leer / escribir si no hay overflow son 2 accesos, y si leemos / 2 escribimos con overflow, pues tenemos 2 páginas que escribir)

↳ Sin embargo : 1) ¿Qué pasa si se acaba el hash ?
2) ¿Memoria interna suficiente ?

Si la función de hash es mala, entonces hay colisiones. No hay mucho que hacer, al igual que en estructura de datos.



todos tienen el mismo $h(x)$, no difieren en ningún bit.

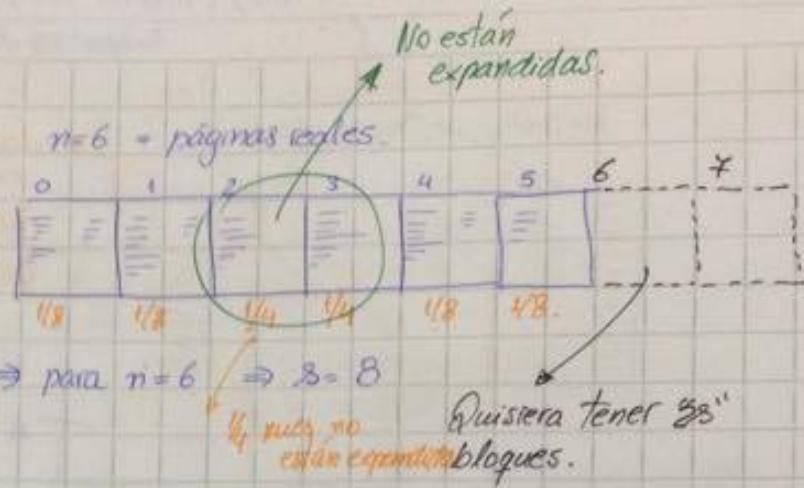


para lo mismo para el caso de páginas.

Hashing Lineal:

Arreglo de bloques en discos

$$2s = 2^{\lceil \log_2 n \rceil}$$



↳ hashing lineal va haciendo una expansión de la tabla de a poco. (Siempre voy buscando $2s$ =potencia próxima de dos). Supongamos que tenemos 8 celdas como arriba. Si queremos que un elemento caiga en una celda "virtual", lo envío a la celda aun no expandida. El algoritmo es:

$S: h(x) \bmod s < n \bmod s$ return $h(x) \bmod 2s$ return $h(x) \bmod s$	}	supondremos $s \leq n < 2s$
--	---	-----------------------------

• ¿Cómo expandir?

Expandir Tabla:

```

leer bloque  $n-s$ 
(rehash con  $h(x) \bmod 2s$ )
 $n \leftarrow n+1$ 
S:  $n=2s$ 
 $s \leftarrow 2s$ 
  
```

• **¿Cómo contraer?**

Contraer - Tabla :

$$n \leftarrow n-1$$

agregar la página n a la $n-3$

$$\text{si } n = 3$$

$$z \leftarrow z/2$$

• **¿Cuando expandir? ¿Cuando contraer?**

Notemos que no podemos expandir cuando hay overflow, ya que es algo restringente que trae consigo un gasto grande. Entonces las páginas con overflow se enlazarán en una lista, esperando su turno. Cuando toque la expansión, se leerá toda la lista y se trasladarán los elementos (todos los de la lista enlazada) en las nuevas posiciones (incluidas las celdas extendidas).

Estrategia (1): expandir cuando sube el tiempo promedio de búsqueda, y contraer en el caso contrario.

Estrategia (2): contraer cuando la tasa de ocupación es baja, y expandir en el caso contrario.

$$\square^* = \left(\frac{1}{1-\square} \right)$$

$$\sum_{i=1}^n a^i = \frac{1-a^{i+1}}{1-a}$$

2015年10月8日 (木)

Análisis amortizado.

Promedio de una secuencia de operaciones ≠ ^{promedio sobre inputs posibles}

→ Análisis de costo de una secuencia de operaciones. Algunos casos se contrarrestan con otros. Se puede llegar a hablar de costos constante amortizado.

- ▶ Análisis global
- ▶ Contabilidad de costos
- ▶ Función potencial.

ANÁLISIS GLOBAL

Ej: contador de bits.

peor caso = k (ancho de k bits)

costo total = kn , $n=2^k$

$$1 \cdot \frac{n}{2} + 2 \cdot \frac{n}{4} + 3 \cdot \frac{n}{8} + \dots \quad (a=\frac{1}{2})$$

$$\leq \sum_{i \geq 1} i \cdot \frac{n}{2^i} = n \sum_{i \geq 1} \frac{i}{2^i} = an \sum_{i \geq 1} (a^i)^{-1} = an \sum_{i \geq 1} (a^i)^{-1}$$

$$= na \left(\sum_{i \geq 1} a^i \right)^{-1} = an \left(\frac{1}{1-a} - 1 \right)^{-1} = \frac{an}{(1-a)^2} = 2n$$

k bits	
000	000
000	001 1
000	010 2
000	011 1
000	100 3
000	101 1
...	2
0111111111111111	1
1000000000000000	(k)

o también $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots = 2n$ (ver por columnas)

⇒ costo amortizado de incrementar es 2.

No tengo más
Los otros

(3) BARRA DE COSTOS

costos de cambio estrictos

CONTABILIDAD DE COSTOS

EJ: Contador de bits de nro.

- Por cada $0 \rightarrow 1$ le voy a cobrar 2 a la operación, adelantándose a su próximo cambio de $1 \rightarrow 0$
- los cambios $1 \rightarrow 0$ son gratis.

000 ... 000
000 ... 001
000 ... 010
000 ... 011
000 ... 100
000 ... 101
000 ... 110
000 ... 111

hay 1 rectángulo por cambio de contador
→ después de n cambios se ha cobrado $2n$

→ alcancía

Función potencial: ϕ es como hacer contabilidad también. Tengo un saquito de ahorro.. ϕ es un valor numérico, en función del estado de la estructura de datos.

c_n = costo real de la n -ésima operación

ϕ_n = el valor de ϕ (la alcancía) después de la n -ésima op.

\hat{c}_n = costo amortizado de la n -ésima operación.

$$\hat{c}_i = c_i + \phi_i - \phi_{i-1} = c_i + \Delta\phi_i$$

Luego

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n c_i + \underbrace{\phi_1 - \phi_0}_{\geq 0} \geq \sum_{i=1}^n c_i$$

cota superior al costo real.



Depth

Ej: Contador de bits.

$$\phi = \sum 1s \text{ en la cadena (total)}$$

$$\begin{array}{r} _ _ _ _ _ \quad 0111 \\ \downarrow +1 \\ \underline{\quad \quad \quad 1000} \end{array}$$

$c_i = \# 1s$ seguidos de der a izq + 1.

Luego $\Delta \phi_i$ es la variación de 1's

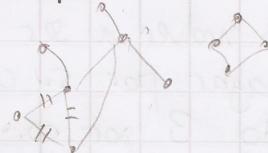
($-c_{i-1}$)

$$\Delta \phi_i = -\# 1s + 1 \quad (\text{la cantidad de } 1s \text{ en el estado anterior})$$

$$\hat{c}_i = c_i + \Delta \phi_i = 2$$

$$\sum \hat{c}_i = 2^n \geq \sum c_i$$

Otro ejemplo DFS en un grafo para detectar componentes conexas.

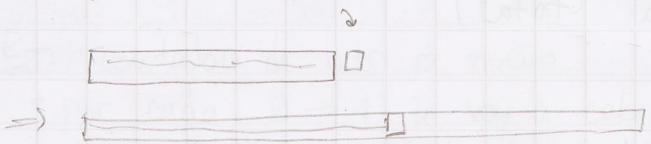


$$\sum_{v \in V} \text{arity}(v) = 2e$$

en vez de ver qué pasa en los nodos, es más fácil cobrar 2 (por adelantado) a cada arista que tenemos.

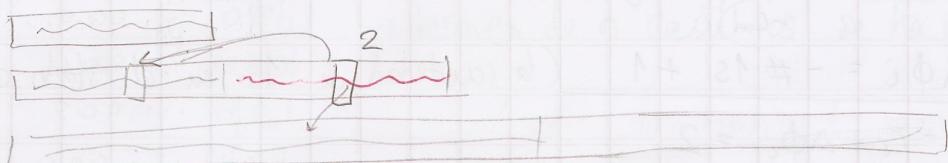
⇒ hemos usado una técnica de análisis amortizado para calcular costo global $\Theta(n+e)$

Ej: reallocar duplicando el tamaño.



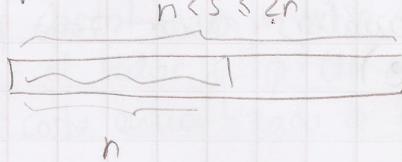
• Contabilidad de costo.

para n ops, cuántas veces se duplica el elemento? peor caso $\log n$...



A los elementos que son insertados en la segunda mitad le cobro 1 por insertar + 2 por su futura copia y por copiar el otro elemento simétrico en la primera mitad.
La gracia es que un elto rojo nunca va a volver a ser rojo de nuevo, y siempre alguien va a pagar por su copia posterior
⇒ n ops cuesta $3n$ → costo amortizado 3 por op.

• Función potencial



$$\phi = 2n - s \rightarrow \text{en una inserción normal (sin reallocación)}$$

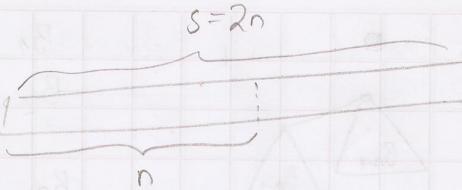
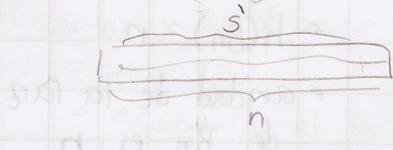
$$c_i = 1$$

$$\rightarrow \Delta c_i = 2 \quad \text{s no varía}$$

$$\hat{c}_i = 3$$

Admonitida

Cuando el arojo se realoca



$$C_i = n$$

$$\begin{aligned}\Delta\phi_i &= \phi_i - \phi_{i-1} = (2n - s) - (2n - s') \\ &= (2n - 2n) - (2n - n) \\ &= -n\end{aligned}$$

\Rightarrow

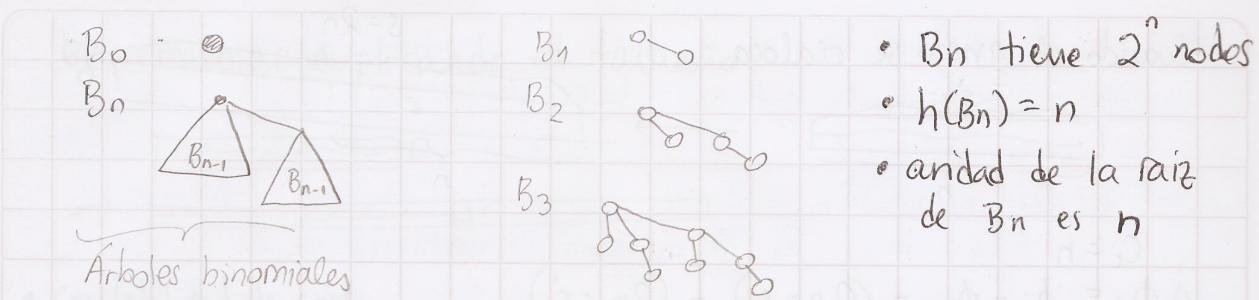
$$\hat{c}_i = 0 \quad \text{Pago con todos los ahorros.}$$

ya que me da 0 , no
debo preocuparme de
cuántas veces realoco
para

En el fondo ϕ está relacionado al espacio libre que queda

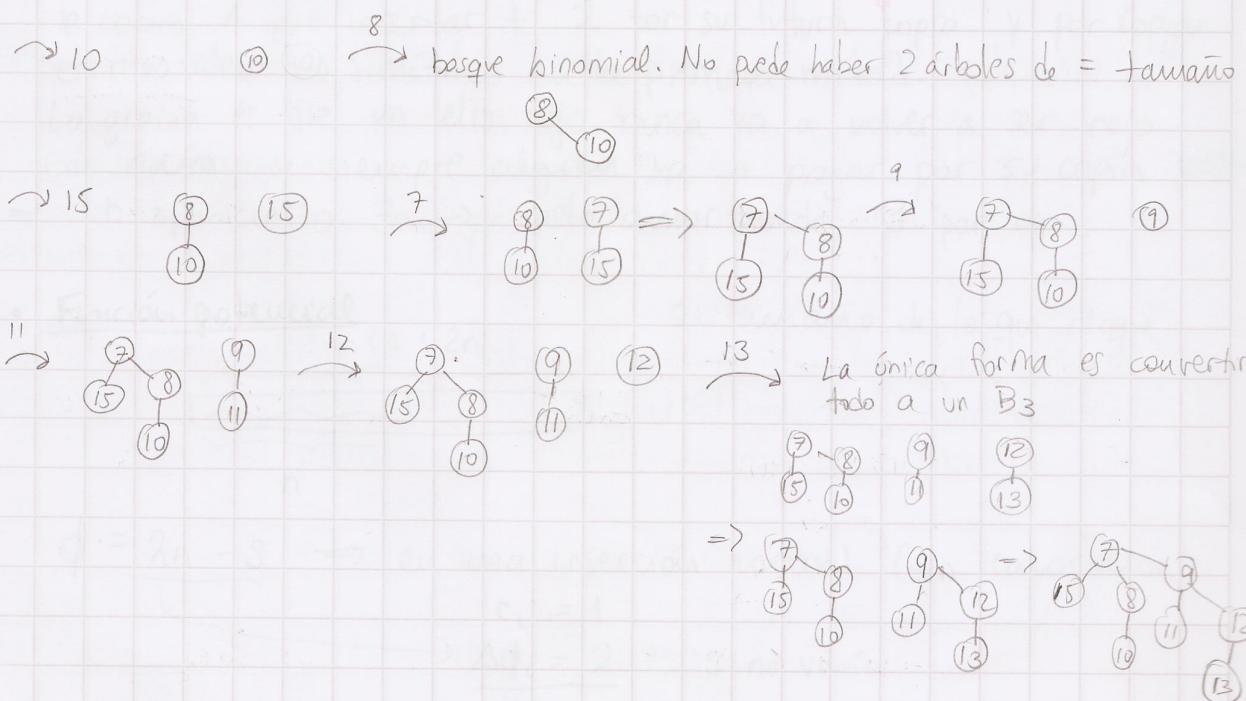
Colas Binomiales

2015年10月13日 (K)



→ Un bosque binomial es un conjunto de árboles binomiales, todos de distinto tamaño.

→ Una cola binomial es un bosque binomial con una clave asociada a cada nodo, donde la clave de un padre no puede ser mayor que la de un hijo.



Entonces ...

-> Insertar x en el bosque

- creo \otimes

- "sumo" \otimes al bosque

↳ si no hay un árbol del tamaño de \otimes
agregamos \otimes

↳ si hay, unimos el árbol de x con el de su mismo tamaño
colgando el de mayor raíz del de menor raíz.

(exactamente la misma mecánica de sumar 1 a un contador binario)

↳ Se obtiene un árbol del doble de tamaño, y se itera.

OBS: Una cola binomial con n claves tiene a lo sumo $\lceil \log_2 n \rceil$ árboles.

∴ La inserción cuesta $\Theta(\log n)$

OBS: podemos hacer heapify mediante insertar n elementos, a costo $\Theta(n)$

(producir esta estructura)

también es
peor caso
↳ pensar en el costo
amortizado de cada
operación de contador
binario)

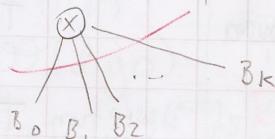
• Encontrar min: Buscar mínimo entre las raíces

↳ $\Theta(\log n)$ podría ser $\Theta(1)$ si recalcular min

• Extraer min: - Buscar raíz de clave mínima, sea

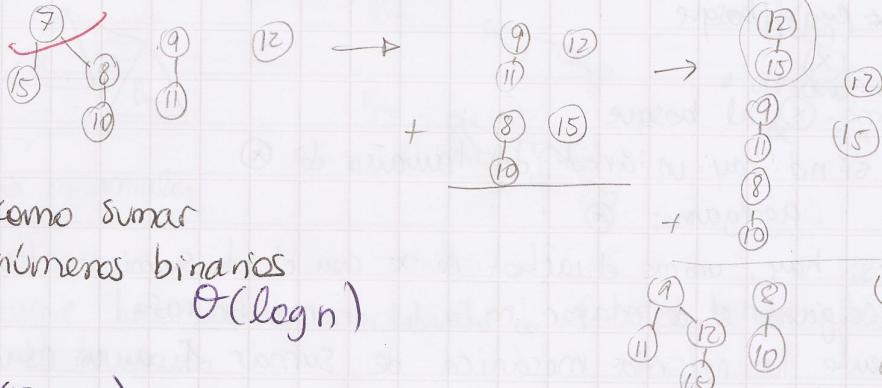


- Eliminar x y quedarse con los k subárboles



Colas Binomiales

- "Sumar" esos k subárboles al resto del bosque que → "reserva"



- Merge (T_1, T_2)

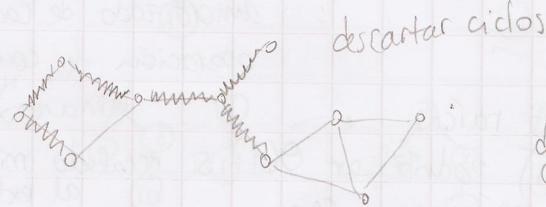
"Sumar" los bosques de T_1 y T_2
 $\Theta(\log(|T_1| + |T_2|))$

(o también
podía dejar
otra pareja
a fuera)

Union Find

Ej: MST (Minimum Spanning Tree)

Kruskal



Al principio, en C
cada nodo por si solo es una comp conexa

todavía
no tenemos
árbol que
conecte todos
el grafo

Kruskal (V, E)

dr a costa
de arista

heapify (E)

$\rightarrow C \leftarrow \{\{v\}, v \in V\}$

$T \leftarrow \emptyset$

\rightarrow while $|C| > 1$

$(u, v) \leftarrow \text{extractMin}(E)$

$U \leftarrow \text{componente de } u \text{ en } C$

$V \leftarrow \text{componente de } v \text{ en } C$

\rightarrow if $U \neq V$

$T \leftarrow T \cup \{(u, v)\}$

$E \leftarrow E - \{(u, v)\} + \{(U \cup V)\}$

return T



MST se puede ver de manera más general.

Problema: mantener un conjunto de clases de equivalencia sobre n elementos

- inicialmente, cada elemento forma una clase separada.
- en cada clase, elegiremos un representante.
- Find(x) me da el representante de la clase a la que x pertenece.
- Union(x,y) une las clases representadas por x e y.

Kruskal (V,E)

heapify(E)

C ← classes(V)

T ← ∅

while |E| > 1

{ (u,v) ← extractMin(E)

x ← Find(u)

y ← Find(v)

if x ≠ y

T ← T ∪ {(u,v)}

Union(x,y)

return T

$\Theta(\log n)$

Se puede ver como árboles
cuyos nodos apuntan a
su padre → representante

Complejidad: Al haber n nodos, todo árbol tiene altura a lo más $\lceil \log n \rceil$

Union: $\Theta(1)$

Find: $\Theta(\log n)$

m finds → $\Theta(m \log^* n)$ amortizado

$$\log^*(65536) = 1 + \log^*(16) = 2 + \log^*(4) = 3 + \log^*(2) = 4$$

Lema: todo árbol de altura h tiene al menos 2^h nodos

Dem: Caso base, $h=0$ y $n=1$ ✓

Caso inductivo: Unimos T_1 y T_2
 $n_1 \geq n_2$.

∴ Colgamos T_2 de T_1

$$T = \begin{array}{c} \triangle \\ T_1 \quad T_2 \end{array} \quad h(T) = \max(h(T_1), 1+h(T_2))$$

2 casos: a) $h(T) = h(T_1)$

$$h = h_1 + h_2 \geq h_1 \stackrel{H.I.}{\geq} 2^{h(T_1)} = 2^{h(T)}$$

b) $h(T) = 1 + h(T_2)$

$$h = h_1 + h_2 \geq 2n_2 \stackrel{H.I.}{\geq} 2 \cdot 2^{h(T_2)} = 2^{1+h(T_2)} = 2^{h(T)}$$

$n = n_1 + n_2$

$\geq 2n_2$

$\geq 2 \cdot 2^{h(T_2)}$

$= 2^{1+h(T_2)}$

$= 2^{h(T)}$

va aprendiendo
modifica la estructura
al leer.

Evaluación de costo de la obra de un elevador.

Se requiere una máquina que levante la carga de 1000 kg.

La máquina debe tener una velocidad constante de 1 m/s.

El motor tiene una potencia de 1000 W y una eficiencia de 80%.

El sistema de frenado es de tipo hidráulico y se requiere una fuerza de frenado de 1000 N para detener la máquina.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

El sistema de control es de tipo digital y se requiere una velocidad de respuesta de 0.1 s.

A) BZ A01 #202

Jificaciones en numeros romanos

Los Agentes

1) BISQUEDA (E)

- Buscar el mejor resultado de acuerdo a la regla

- 2) (logística) buska en resultados random

2) INSECCION (el sistema basico) 3) observar cada uno

- 1) solo debe haber una insercion, ya no se consideran mas

- 2) solo se inserta una vez en la lista

- 3) solo se inserta una vez en la lista

- 4) solo se inserta una vez en la lista

- 5) solo se inserta una vez en la lista

- 6) solo se inserta una vez en la lista

- 7) solo se inserta una vez en la lista

- 8) solo se inserta una vez en la lista



$$(n_{\text{coll}}) \odot = d \quad ? \quad \text{Lo que se ve es que } d \text{ es}$$