

8/9/16

CC4102 - D. y A de Algo

Cotas inferiores (de pmas)

cota superior \Rightarrow encontrar un algoritmo que resuelva
cota inferior?

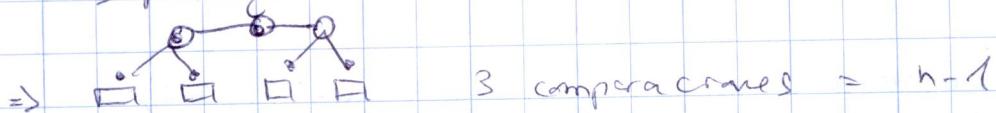
el adversario: "se encarga de asegurarse que el algoritmo
se encuentre con el peor caso"

- \hookrightarrow es como guardarse el último submarino en Guerra Naval y al final ponelo
- \hookrightarrow debe ser constante

elemento

al querer encontrar el mínimo de un arreglo
 \Rightarrow recorrer todo $\rightarrow n-1$ comparaciones

\hookrightarrow ¿para qué ese el número mínimo?



\Rightarrow mientras exista más de 1 componente conexa, puede haber
un error \Rightarrow hay que hacer que todo sea conexo
 \Rightarrow mínimo $n-1$ comparaciones (pero hacer un
grafo conexo)



no puede conocer el
mínimo global
ss no hay comparación!

Si se hacen comparaciones
ES NECESARIO HACER
al menos $n-1$.

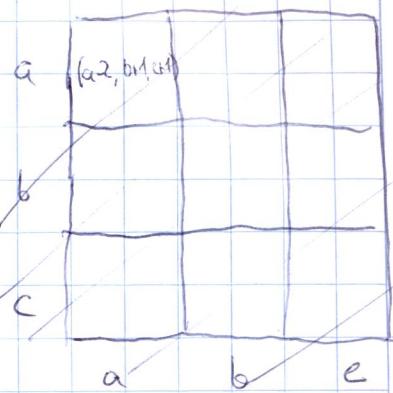
PARA ENCONTRAR EL MÁXIMO

est int est final

separar en 3 conjuntos

$$(n, 0, 0) \Rightarrow (0, 1, n-1)$$

a) No ha sido comprado



b) ha ganado todas sus compras

\Rightarrow no sirve de nada
hacer compras con c

c) ha perdido una o mas compras

| | a | b | c |
|---|-----------------|---------------|---------------|
| a | (a-2, b+1, c+1) | (a-1, b, c+1) | (a-1, b+1, c) |
| b | (a, b-1, c+1) | (a, b, c) | (a, b-1, c+1) |
| c | | (a, b, c) | |

$\Rightarrow (a, a)$ $n/2$ veces
luego (b, b) $n/2 - 1$ veces
 \circ
 (a, a) 1 vez
luego (a, b) $n-2$ veces

a) No ha sido comprado

$$(n, 0, 0, 0) \Rightarrow (0, 1, 1, n-2)$$

b) ha ganado todo

c) ha perdido todo

d) ha ganado al menos una y perdido al menos una

| | a | b | c | d |
|---|--------------------|------------------------------|------------------------------|------------------|
| a | (a-2, b+1, c+1, d) | (a-1, b, c+1 , d) | (a-1, b, c, d+1) | (a-1, b+1, c, d) |
| b | (a-1, b, c, d+1) | (a-1, b+1, c, d) | (a-1, b, c+1, d) | |
| c | (a, b-1, c, d+1) | (a, b, c, d) | (a, b, c, d) | |
| d | | (a, b-1, c+1, d+1) | (a, b-1, c, d+1) | |
| | | (a, b, c-1, d+1) | (a, b, c, d) | |
| | | | (a, b, c+1, d+1) | |
| | | | (a, b, c, d) | |

$$\lceil \frac{n}{2} \rceil + n - 2$$

$$\lceil \frac{3n}{2} \rceil - 2$$

adversario te quiere coger y elige la peor opción siempre!

! La tabla entrega cota inferior, luego al encontrar un algoritmo se vuelve cota superior y por sandwich se vuelve mejor algoritmo.

Tabla punto = Estrategia del Adversario

⇒ que necesita saber el input para avanzar
(aún con el peor input posible)

⇒ modelo debe tener algún tipo de forma de proceder.
(en el ejemplo visto, procede por comparaciones)

Reducción (como P, NP)

ej: Cápsula convexa (convex hull)

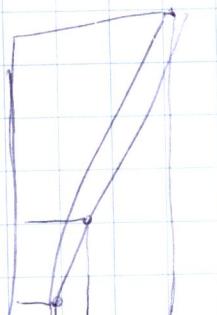
= encontrar la cápsula convexa que contiene todos los puntos.



⇒ $O(n \log n)$, n = cantidad de puntos.

$x_1^1, x_2^1, \dots, x_n^1$ ← puntos

$(x_1^1, x_1^2), (x_2^1, x_2^2), \dots, (x_n^1, x_n^2)$



Al ordenarlos (me demoro $n \log n$) y así
puedo hacer la cápsula convexa.
Por ende el ^{la} tiempo de C.C. es $\Theta(n \log n)$

Además es $\Omega(n \log n)$

⇒ $\Theta(n \log n)$

hay que asegurarse que la reducción / conversión sea
menor al orden al que queremos demostrar.

13/9/16

EC4102 - Logaritmos

Reducciones

Ej: colas de prioridad

?

| | | | |
|-------------|-------------|-------------|--------|
| insertion | $O(\log n)$ | $O(1)$ | $O(1)$ |
| min | $O(1)$ | $O(1)$ | $O(1)$ |
| extract-min | $O(\log n)$ | $O(\log n)$ | $O(1)$ |

\Leftrightarrow heap

\Leftrightarrow fibonacci

sort

imposible pq sino
podriamos ordenar un
elemento arreglo en

$O(n)$ pero sabemos
que ordenar es $\Omega(n \log n)$



concepto de NP-Hard, se utiliza
tbn para encontrar colas inferiores.

3SUM-hard (\Rightarrow el pbma es igual de
dificil que 3SUM

$\Rightarrow O(n^2)$ ordenar $n \log n$

+ recorrido con 2 punteros

+ 1 terero



$$O(n^2 / (\log n / \log \log n)^2) < O(n^2)$$

Graham & Pittie 2014

se encontró un alg^{mo} mejor!

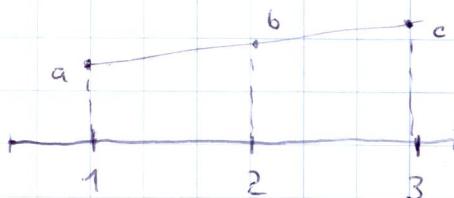
Al reducirlo a un pbma de pts. colineales.

Construyo a partir de los x_0, x_1, \dots, x_n puntos,
3n puntos. Para cada uno de ellos, los pongo
en las sig coordenadas.

$$(1, x_i)$$

$$(2, -x_{i/2})$$

$$(3, x_i)$$



$$b = \frac{a+c}{2} \Leftrightarrow -2b + a + c = 0$$

$$y \quad b = -\frac{x_b}{2} \Leftrightarrow -2b = x_b$$

Y

$$y \quad b^2 = \Rightarrow x_b + x_a + x_c = 0 //$$

Teoría de la información

ej: un algoritmo de búsqueda en un arreglo A [1, n] incluso aunque A esté ordenado, debe recibir al menos $\lceil \log_2 n \rceil$ comparaciones, pues debe ser capaz de dar n respuestas distintas.

al separar siempre por la mitad.

ordenar A [1, n]

n! órdenes posibles
(o permutaciones)



$$\lceil \log_2 n! \rceil = \lceil S(n \log n) \rceil \text{ (Stirling)}$$

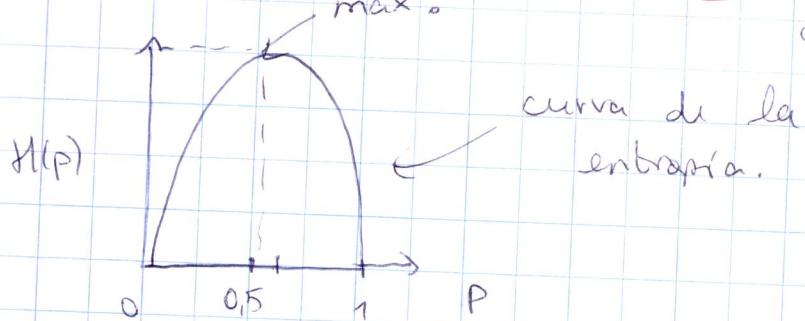
⇒ Esto me va a dar el MÍNIMO de preguntas necesarias para poder caracterizar el conjunto y por ende poder ordenarlo.

Shannon (1948)

Si una fuente emite símbolos con probabilidades $|P_i\rangle$, entonces la codificación más corta usará una permutación:

$$H = \sum_i P_i \log_2 \left(\frac{1}{P_i} \right) \text{ bits.}$$

← cuanto más predecible es el elemento menor información es necesaria!



curva de la entropía.

cuando todo es igualmente probable:

$$H = \sum_{i=1}^n \frac{1}{n} \log_2 \left(\frac{1}{n} \right) = \log_2 (n!)$$

20/9/16

CC4102 - Logaritmos

Cota inferior de Shannon:

$$H = \sum p_i \log_2 \left(\frac{1}{p_i} \right)$$

si tengo las probabilidades, soy capaz de armar la estructura ideal?

Ej. búsqueda en un arbolito

$$p_i = \frac{1}{n} \rightarrow H = \log_2 n \quad (\text{búsqueda binaria})$$

$p_i = \frac{1}{2} \rightarrow H = 2$ lo alcanzo con búsqueda secuencial.

Códigos de Hoffman

| | | p_i | |
|---|----|---------------|-----|
| A | 00 | $\frac{1}{2}$ | 0 |
| C | 01 | $\frac{1}{8}$ | 110 |
| G | 10 | $\frac{1}{4}$ | 10 |
| T | 11 | $\frac{1}{8}$ | 111 |

ningún código es prefijo de otro

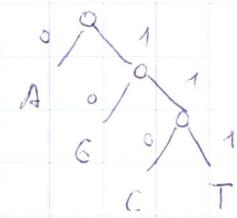
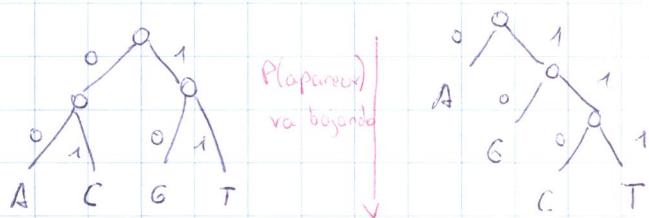
= "sr libre de prefijo"

= "código instantáneo"

$$\frac{1 \cdot \frac{n}{2}}{2} + 2 \cdot \frac{n}{4} + 3 \cdot \frac{n}{8} \cdot 2 = \frac{7}{4} n < 2n$$

largo promedio

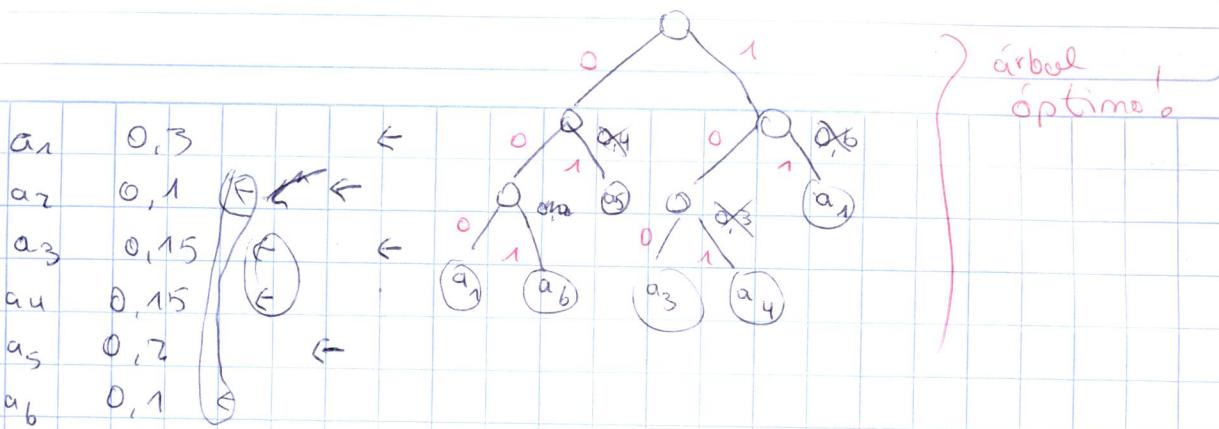
; compresión!



$\sum p_i \cdot l_i$ = Largo Promedio
 p probabilidad
 largo del orbital para
 idealmente
 esto sea =

Algoritmo de Hoffman $O(n \log n)$ a H .

- ordenar las probas de mayor a menor
- crear n nodos ai, con probabilidad pi
- mientras haya más de un árbol
 - * tomamos los árboles de menor peso probabilidad
 - * unirlos 
 - * dándole la suma de las pesos probabilidades.
- * reemplazando en el conjunto



árbol óptimo!

$$\sum p_i l_i > \sum p_i \log_2 \frac{1}{p_i}$$

$$H \leq \sum p_i l_i \leq H+1 \quad \leftarrow \text{al utilizar el algoritmo de Hoffmann.}$$

Si las probabilidades vienen ordenadas, entonces el algoritmo de Hoffmann se puede hacer en $O(n)$

\Rightarrow Hoffmann no sirve para hacer un árbol de búsqueda (sí para codificar)

\Rightarrow ¡Quiero un árbol de Hoffmann donde las hojas estén ordenadas!

Mejor algoritmo posible preservando el orden de las hojas
(Hu-Tucker / Knuth 73 / Garcia-Wachs)

$O(n \log n)$

$\sum p_i l_i < H+2$
(muy complicado)

$p_1 \dots n \rightarrow p_i = \text{prob buscar } A[i]$

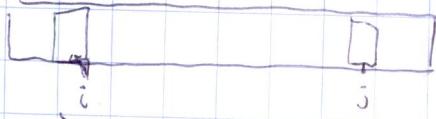
$q_1 \dots n \rightarrow q_j = \text{prob buscar entre } A[j], A[j+1]$

$A[1 \dots n], A[0] = -\infty$

$A[n+1] = +\infty$

$$\sum p_i + \sum q_i = 1$$

$$P_{ij} = q_{i-1} + p_i + q_i + p_{i+1} + \dots + q_j p_j + q_j = P_{i,j-1} + p_j + q_j$$



P (que esté aquí)

Cálculo de P_{ij} en tiempo $O(n^2)$

C_{ij} = mejor costo promedio (del mejor árbol posible) en $A[i:j]$

$$C_{i,i-1} = 0$$

$$C_{i,i} = 1$$

$$C_{i,j} = \min_{k \leq K} 1 + \frac{P_{i,k-1}}{P_{i,j}} C_{i,k-1} + \frac{P_{i,k+1}}{P_{i,j}} C_{i,k+1}$$

\Rightarrow buscamos $C_{1,n}$

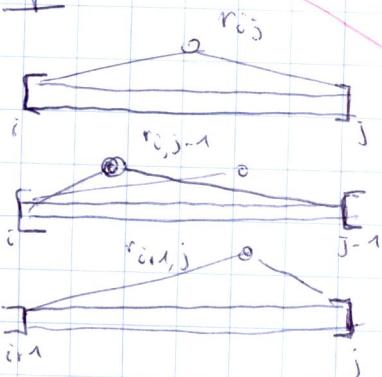
\hookrightarrow recordar el camino de tal manera de poder reconstruir el árbol.

$r_{i,j} = \arg \min (C_{i,j}) \Rightarrow$ llenar esta matriz toma $O(n^3)$

a elementos

Nota: kbn es adaptable a ~~solo~~ consultas con tiempo de acceso distinto (SSD, HDD, RAM)

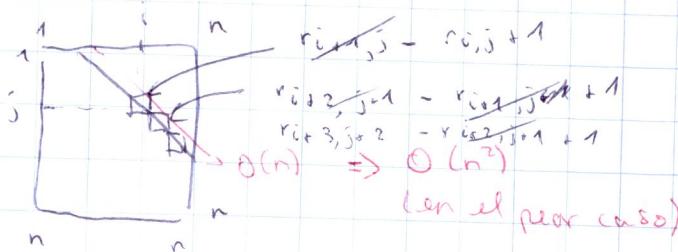
Prop:



propiedad

$$r_{i,j-1} \leq r_{i,j} \leq r_{i+1,j}$$

$$r_{i,j-1} \leq K \leq r_{i+1,j}$$



22/9/16

CC4102 - Logaritmos

Algo's en Mem 2^{r.o}

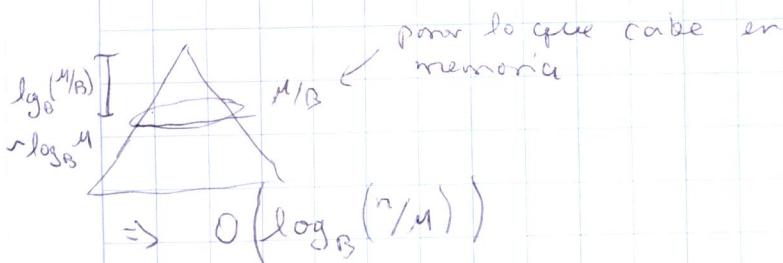


seek } para localizar a cualquier
latencia } consulta cíclonica.
trasferencia

Algoritmo del bloque

Para simplificar el análisis, supondremos que el acceso es secuencial.

- B = tamaño del bloque
- M = tamaño de la RAM
- contamos ~~bloques~~ leídos/escritos (I/O)



bloques en forma de árbol. B
(árbol 2-3 pro con $B, B+1$)
 $\Rightarrow O(\log_B n)$
(ya que altura árbol = $\log_B n$)

- ordenamiento - colas de prioridad - hashing.

Mergesort en disco

uno utiliza los bloques como unidad, y la última iteración es innecesaria (no se accede al disco una vez que la recursión se acaba)

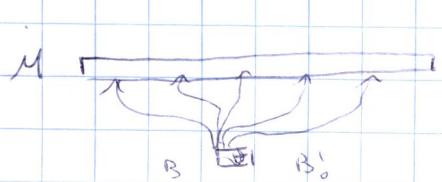
$$\log_2 n \Rightarrow \log_2 \left(\frac{n}{B} \right) \Rightarrow \frac{n}{B} \log_2 \left(\frac{n}{B} \right) \Rightarrow \frac{n}{B} \log_2 \left(\frac{n}{M} \right)$$

↑ ↑ ↑
 la unidad son los bloques cantidad de veces que se han al guardar en memoria.

$$\Rightarrow \underbrace{\frac{n!}{B!} \log_{\frac{B}{n}} \left(\frac{n!}{B!} \right)}_{\text{Sort}(n)} \text{ es final}$$

Mergesort en disco. Cota inf.

Al principio: $n!$ permutaciones posibles
 → lee un bloque de B elem^s



$\binom{M+B}{B}$ cantidad de formas en que se pueden insertar los elem^s

↑ orden interno dentro de la memoria

$\binom{M+B}{B} B!$ ya que el adversario } adversario elegira
 decide como pone las } la partición + grande

en el mejor de los casos, el algoritmo puede hacer que todas las particiones sean del mismo tamaño

$\Rightarrow \frac{n!}{\binom{M+B}{B} B!}$ es el nuevo tamaño
 (igual para cada posibilidad)

↓ si pueden separarán t pasos

$$\left(\frac{n!}{\binom{M+B}{B} B!} \right)^t = 1 \Leftrightarrow t = \frac{\log n!}{\log \binom{M+B}{B} + \log B!} = \Theta \left(\frac{n \log n}{B \log B} \right)$$

$$= \Theta \left(\frac{n \log n}{B \log B} \right) =$$

está mal!

N

$$\frac{n!}{(M+B)^t \cdot B^t} = 1$$

$$\Leftrightarrow \frac{n^t}{B^t} = \left(\frac{M+B}{B}\right)^t$$

$$\Leftrightarrow n \log n - \cancel{n \log B} = t B \log \frac{n}{B}$$

$$\Leftrightarrow \frac{n}{B} \frac{\log \frac{n}{B}}{\log \frac{M}{B}} \Leftrightarrow \boxed{\frac{n}{B} \log_{M/B} \left(\frac{n}{B}\right)}$$

$$\frac{n}{B} \left(\log_{M/B} \left(\frac{n}{M} \right) + 1 \right) = \frac{n}{B} \left(\log_{M/B} \left(\frac{n}{M} \right) + \log_{M/B} \left(M/B \right) \right)$$

$$= \frac{n}{B} \left(\log_{M/B} \left(\frac{n}{M} \cdot \frac{M}{B} \right) \right)$$

$$= \frac{n}{B} \log_{M/B} \left(\frac{n}{B} \right) = \frac{n}{B} \left(\log_{M/B} \left(\frac{n}{M} \right) + 1 \right)$$

por ende esta determinado.

$$\Rightarrow \text{sort}(n) = \Theta \left(\frac{n}{B} \log_{M/B} \left(\frac{n}{M} \right) \right)$$

28/9/16.

CC4102 - Log

Cola de prioridad en mem. Secundaria

→ B-tree $O(\log_B n)$

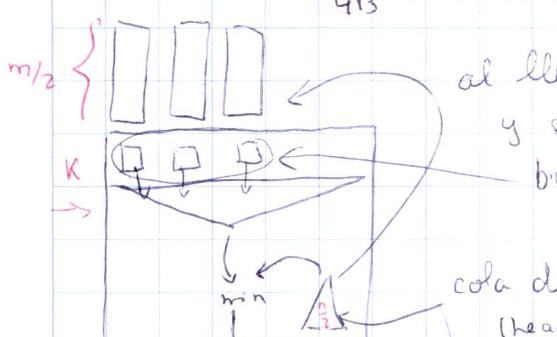
→ Obtenremos

→ Cota inferior $\Omega\left(\frac{1}{B} \log_{\frac{M}{B}}\left(\frac{n}{m}\right)\right)$

Operaciones:

- Crear
- Insertar
- Min
- Extraer Max

1) Caso $n \leq \frac{M^2}{4B}$ total de inserciones en la vida de la cola de prioridad



al llenarse la cola de inserción, se ordenan
y se escribe en disco.
buffer con info sobre los archivos en disco

$$O(1/B) \text{ (extraer)} \\ K \leq \frac{M}{2B}$$

$$n = K^r \frac{M}{2} \\ \Rightarrow K^r = \frac{2n}{M}$$

$$\Rightarrow r = \log_K \left(\frac{2n}{M}\right)$$

$$\Rightarrow O(r^2/B) = O\left(\frac{1}{B} \log_K \left(\frac{2n}{M}\right)\right)$$

el elem pasa por todos los niveles,
(peor caso)

$$rKB \leq \frac{M}{2}$$

tamaño del buffer para cada grupo de tamaño $\frac{K^r M}{2}$



$$rKB \leq \frac{\mu}{2}$$

$$K \leq \frac{\mu}{2B} \cdot \frac{1}{\log_K\left(\frac{n}{\mu}\right)}$$

$$\Rightarrow \left\{ \begin{array}{l} \frac{1}{B} \\ \log_K\left(\frac{n}{\mu}\right) \end{array} \right.$$

si $K \approx \mu/B$ llegamos a la cota inferior

Mejor caso del costo de un elem a lo largo de todo su vida.

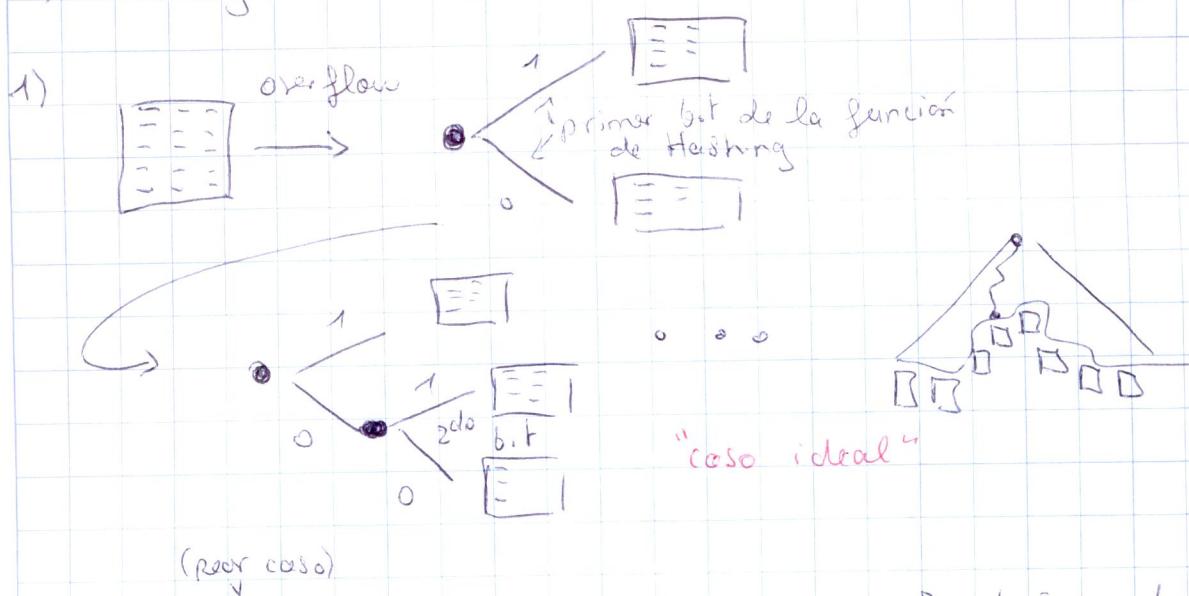
29/9/16

CC4102 - Log

Hashing en Mem ^{ra} _{2 vía}

- 1) Hashing extensibles
- 2) Hashing lineal.

c: cantidad de colisiones
 $\frac{1+c}{B}$: costo de encontrar algo



¿Qué pasa si no se puede separar en 2 hojas de hashing? (el mismo bit es el mismo para todos los valores)

⇒ expansión de k-bits para hacer una buena división

⇒ Una función bien diseñada no permite eso

Buena función de hashing se comporta de forma aleatoria sin exceso!
→ hojas llenas con ~63%.

$$\Rightarrow n_{mem} \approx \frac{n}{0,63B}$$

¿Qué pasa si se nos acaban los bits de la función de hashing?

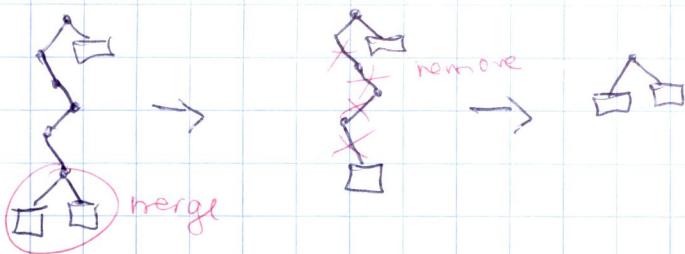
si la función de hashing es buena, en el peor caso habría que insertar $0,632^{64}$ elementos para llenar el árbol (lo que es MUCHA info)

\Rightarrow hay demasiadas colisiones

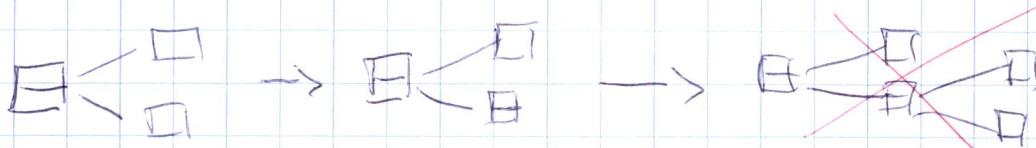
\Rightarrow si hay overflow y se acaban, linko la lista hoja a otra hoja más y sigo.

Si al eliminar elementos de una hoja, se posiblemente la hoja vecina posee una cantidad de elementos que es posible de juntar dentro de una hoja, es bueno hacer un merge (solo cuando $h_1 + h_2 \leq 10$. del espacio)

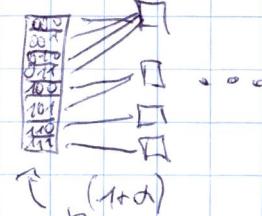
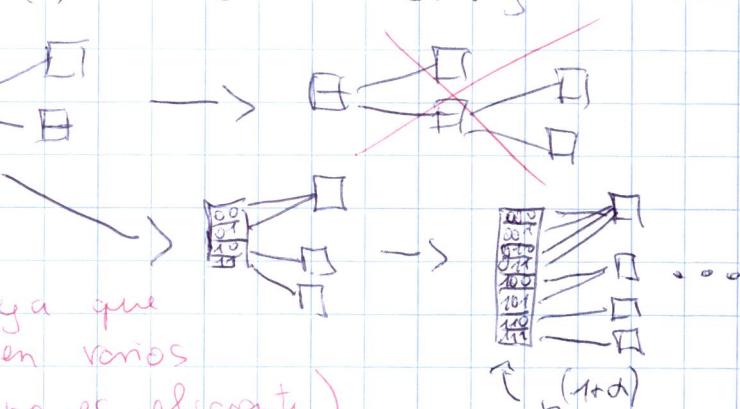
En el mismo caso, si para separar las juntar hojas, hay que revisar si el hermano es null, habrá que seguir subiendo en los nodos!



Nota: En la literatura, extensible hashing:



bueno a menos que se duplique mucho (ya que creces páginas tienen varios accesos, por lo que no es eficiente)



A) peor caso; hago 1 inserción y tengo que duplicar la tabla.

2) Hashing Lineal



$N-1$

$$N = 2^d + r, \quad 0 \leq r < 2^d$$

$\square \quad N$ páginas en disco

$$h(x) \bmod 2^{d+1}$$

(ej. $d=3, r=2 \Rightarrow N=10$ páginas)

ss $h(x) \bmod 2^{d+1} < N \Rightarrow h(x) \bmod 2^{d+1}$ } (\Rightarrow estar siempre
sino $\Rightarrow h(x) \bmod 2^d$) } en transición entre
 2^d y 2^{d+1}

ss asumiendo que tenemos 2^d elem., el siguiente
cae dentro de los páginas existentes, lo dejamos ahí.
En el caso contrario (es decir que quieren caer
fuera de lo existente) aplicamos mod 2^d , por lo
que cae dentro de lo existente (obligatoriamente). Una
vez que determinamos que hay que duplicar agarramos
el bloque en el cual cayeron los que querían
salir y vuelvo a calcular su hashing y eso
los va a separar entre ambos.

Si no se separa, si linkamos la lista a otra.

Decidimos incrementar N (expandir en 1)

$\rightarrow N \leftarrow N+1$
si $N \neq 2^{d+1}$
 $d \leftarrow d+1$
 $r \leftarrow 0$

{ con el bloque $N-2^d$
{ rehashear entre del $N-2^d$ y el N

Se decide eliminar, bajo el N ($N \leftarrow N-1$)
y se ordenan como anteriormente.

Criterios

Condicionas para aumentar / disminuir N

→ coste promedio de acceso

→ % de uso del disco.

4/10/16

CC 4102 - Log

Análisis amortizado

= algunos casos son muy caros y
obras no

→ hay que hacer un análisis
amortizado

(\Rightarrow si x cuesta \$\$\$, entonces
y cuesta \$)

$n=2^k$ bits
 $\begin{cases} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ \vdots & & & \\ 1 & 1 & 1 & 1 \end{cases}$

$O(k) \Rightarrow O(kn)$ ← dim pesimista! en ningún caso llega a
ese orden
cambios

(bits
cambiados)

es interesante contarlos por columnas!
la 1^{ra} columna siempre cambia
la 2^{da} vez por mds
la 3^{ra} cada 4

\Rightarrow cambios (n) = $n + n/2 + n/4 + \dots + n/2^{k-1} \leq 2n$ (análisis
completo)

Contabilidad de costos

- a un cambio de 0 → 1 le cobro 2

(cobro por adelan-

- a un cambio de 1 → 0 le cobro 0

(tado)

\Rightarrow cada incremento me cuesta 2.

(\Rightarrow puede que

\Leftrightarrow costo (n) $\leq 2n$

uno cobre más!

\Rightarrow cota superior

↓

función potencial

$n = \#$ elementos

$s =$ espacio alocado

$$s/2 \leq n \leq s$$

$$\phi = 2n - s > 0$$

$$c_i = 1$$

$$\Delta\phi_i = 2$$

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

realloc ($n=s$) $c_i \approx n$

$$\hookrightarrow 2s = 2n \rightarrow \Delta\phi_i = -n$$

$$\phi_i = 0$$

$$\phi_{i-1} = n$$

$$\tilde{c}_i = 0$$

dn 6/10/16

CC #102 - Log

Union - Find

MST (minimum spanning tree)

$$\text{Nodos} n = |V| \\ \text{Aristas} e = |E|$$

Kruskal:

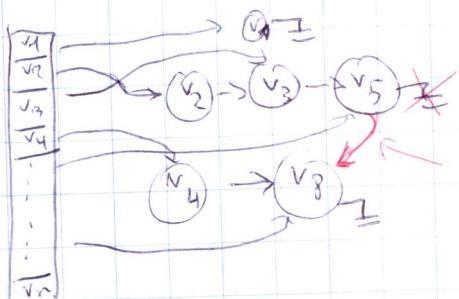
- ordenar las aristas E de menor a mayor costo
- $T \leftarrow \emptyset$ $e = \{(u, v) \mid u \in V\}$
- mientras $|T| < |V| - 1$
 - $(u, v) \leftarrow$ seg arista de E
 - si $Find(u) \neq Find(v)$
 - $T \leftarrow T \cup \{(u, v)\}$
 - $Union(Find(u), Find(v))$

{ $Find(u)$ = la componente conexa de u
Union(c, d) = unir las componentes conexas c y d

clases de equivalencia

Bon más bien árboles

Implementación posible: ~~listas enlazadas~~.



el representante de la clase es el último elemento

al unirlas, que uno de los representantes apunta al otro

Find() $O(n)$ o vice versa ...
Union $O(1)$

grande

unir la lista más grande a la más chica
 $\Rightarrow \log(n)$

Lema: Un árbol formado con camino de altura h , tiene al menos 2^h elementos.

Dem: Vale al principio, alt $h=0$, $2^0=1$ nodo.

Cuando unimos los árboles:



$$\begin{aligned} n_1 &> 2^{h_1} \\ n_2 &> 2^{h_2} \\ n_1 &> n_2 \end{aligned}$$

colgamos el árbol más chico del árbol más grande.

árbol más chico!
(lo elegimos)

$$h = \max(h_1, h_2)$$

$$n = n_1 + n_2$$

si $h = h_1$:

$$2^h = 2^{h_1} \leq n_1 \leq n \Rightarrow h = h_1, 2^h$$

si $h = h_2 + 1$:

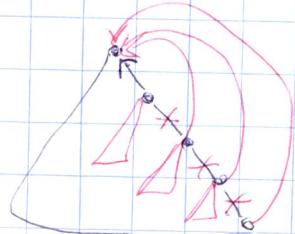
$$2^h = 2^{h_2} \leq n_2 \leq n \Rightarrow h = h_2 + 1 \text{ se cumple que } 2^{h_2+1}$$

por recurrencia funciona!

\Rightarrow Find ($\log n$)

Union (1)

\Rightarrow Kruskal es $e \log e$



al llegar al representante, se aprende cual es el de cada uno de los nodos dentro de la lista.

\rightarrow mejora? Análisis amortizado!

composición de caminos. ($\Rightarrow \log^* n$)

$$F(0) = 1$$

$$F(i) = 2^{F(i-1)}$$

$G(n)$ es mínimo si \log^*

$$G(n) = \min \{ i, F(i) \geq n \}$$

$\log^* n$

| n | $F(n)$ |
|-----|--------------------|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 16 |
| 4 | 65536 |
| 5 | 2 ⁶⁵⁵³⁶ |

$G(n)$

| n | $G(n)$ |
|---------------------------|--------|
| 0 | 0 |
| 1 | 0 |
| 2 | 1 |
| 3..4 | 2 |
| 5..16 | 3 |
| 16..65536 | 4 |
| 65536..2 ⁶⁵⁵³⁶ | 5 |

$\Rightarrow \log^* n \leq 5 \quad \forall$ todo valor

concebible

(Nota: no es la mejor cota, la mejor es la inversa de la función de Ackermann...)

↳ dem NOT PRETTY

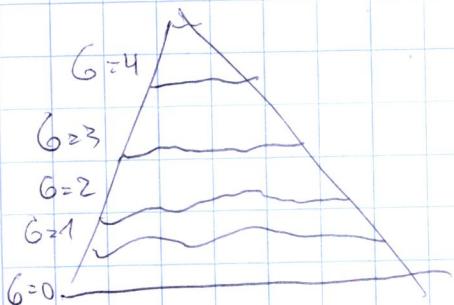
Conocemos el árbol resultante aplicando sólo \log^* un rango = largo del ^{caminocobrido} de un nodo v hasta la hoja más profunda del ~~en~~ nodo v del árbol.

Lema: Hay, a lo sumo $\frac{n}{2^r}$ nodos de rango r

Dem: Cada nodo de altura r tiene al menos 2^r descendientes, distintos de los de otro nodo de altura r .

Def: un nodo v pertenece al grupo $G(r(v))$

$$G=5$$



el máximo grupo posible es

$$\Rightarrow G(\log n) = G(n) - 1$$

Contabilidad de costos

→ Sea una operación $\text{find}(v)$ que pasa por varios nodos n

En ese camino,

- si n es la raíz o hoja de la raíz, paga f_{find}
- si n es de distinto grupo de su padre, paga f_{find}
- si no, paga n .

Obs: si v es hijo de u , en algún momento, $r(v) < r(u)$

Obs: con nuestro esquema de cobro, Find paga a lo más $G(n)+1$ veces.

Obs: Cada vez que un nodo paga, crece al menos en 1 el rango de su padre actual

Obs: un nodo g del grupo g paga a lo sumo $F(g) - F(g-1)$ veces hasta adquirir un padre de otro grupo y desde ahí no paga nunca más.

$N(g) = \#$ de nodos en el grupo g

$$N(g) = \sum_{r=F(g-1)+1}^{F(g)} \frac{n}{2^r} = \frac{n}{2^{F(g)-1}} \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) < \frac{\frac{n}{2}}{2^{F(g)-1}} = \frac{n}{F(g)}$$

Los $\frac{n}{F(g)}$ nodos del grupo g pagan en lo sumo

$F(g) - F(g-1) \leq F(g)$ cada uno
en total, pagan $\frac{n}{F(g)} \cdot F(g) = n$

Hay $G(n)$ grupos g , en cada uno se paga n .

\Rightarrow total $n \cdot G(n)$

Teorema: una secuencia de n operaciones U-F a partir de n elementos aislados.

cuesta a lo mas

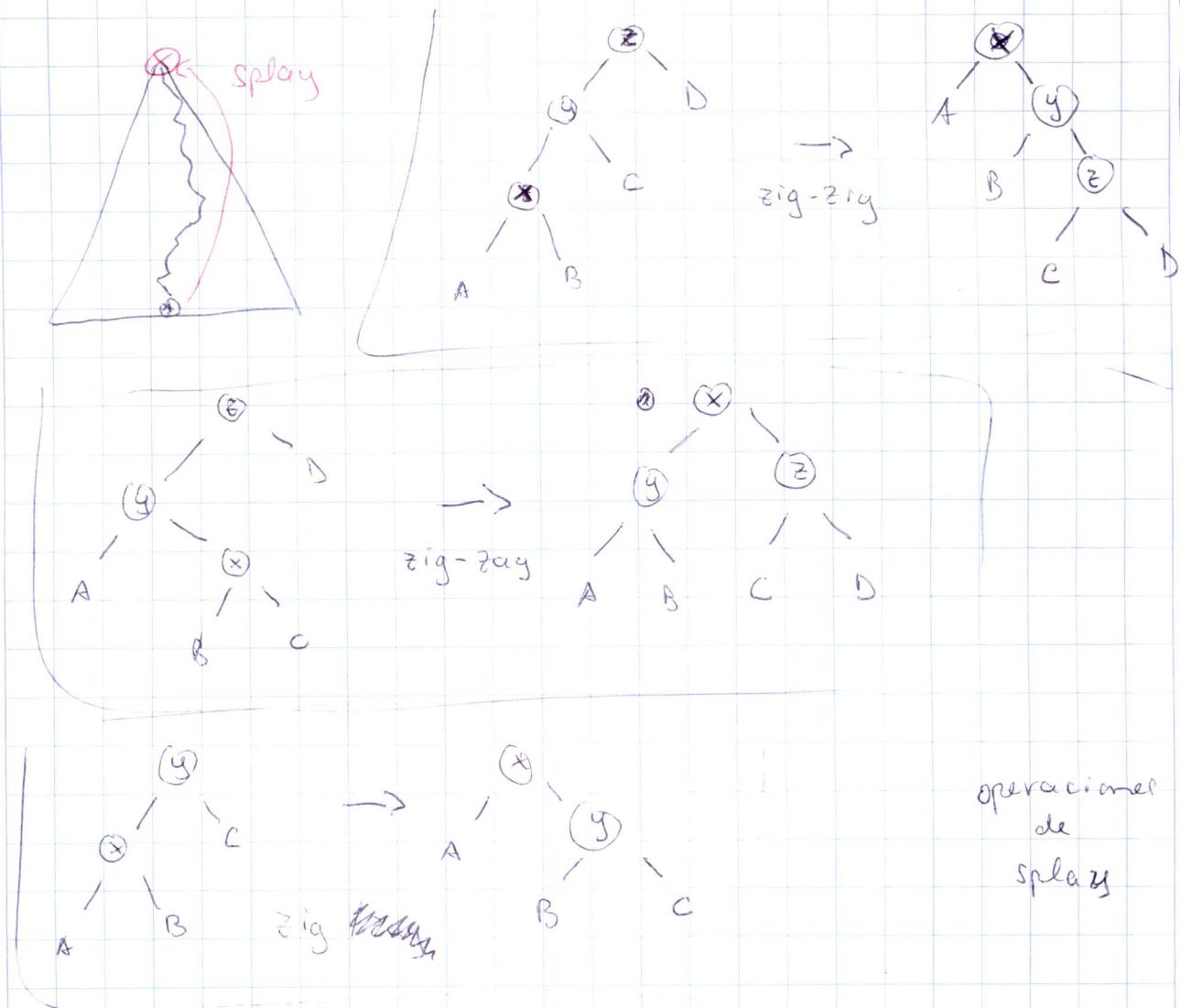
$$O(n \cdot G(n)) = O(n \log^* n)$$

el costo amortizado por operación es $O(\log^* n)$

11/10/16

CG4102 - Log

Splay Trees: al buscar un elemento y encontrarlo, este queda cerca de la raíz.



Si $f_i(x) = n^o$ nodos en el subárbol de x , luego de la operación i :

$$r_i(x) = \log_2 f_i(x) \quad \text{"rango de } x\text{"}$$

$$\phi_i = \sum_{x \in T} r_i(x)$$

Zig-Zig

$$\Delta \phi_i = r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y)$$

- $r_{i-1}(y) > r_{i-1}(x)$
- $r_i(y) \leq r_i(x)$

$$\Delta \phi_i \leq r_i(x) + r_i(z) - r_{i-1}(x) - r_{i-1}(y)$$

$$\Delta \phi_i \leq r_i(x) + r_i(z) - 2r_{i-1}(x)$$

$$\frac{\log a + \log b}{2} \leq \log \left(\frac{a+b}{2} \right)$$

$$r_{i-1}(x) + r_i(z) = \log S_{i-1}(x) + \log S_i(z)$$

$$\leq 2 \log \left(\frac{S_{i-1}(x) + S_i(z)}{2} \right)$$

$$\leq 2 \log \frac{S_i(x)}{2} = 2 \log (S_i(x)) - 2$$

$$= 2r_i(x) - 2$$

$$r_i(z) \leq 2r_i(x) - 2 - r_{i-1}(x)$$

$$\Delta \phi_i \leq r_i(x) + 2r_i(x) - 2 - r_{i-1}(x) - 2r_{i-1}(x)$$

$$\Delta \phi_i \leq 3(r_i(x) - r_{i-1}(x)) + 2$$

$$\underline{c_i = 2}$$

$$\hat{c}_i \leq 3(r_i(x) - r_{i-1}(x))$$

Zig

$$c_i = 1$$

$$\phi_i = r_i(x) + r_i(y) - r_{i-1}(x) + r_{i-1}(y)$$

Nota: al hacer muchas operaciones de splay, además de subir los nodos buscados, el árbol se ordena (y se acorta).

$$r_i(x) > r_i(y)$$

$$\Delta \phi_i \leq r_i(x) - r_{i-1}(x)$$

$$\sum c_i \leq 1 + 3(r_i(x) - r_{i-1}(x))$$

$$\sum_{i=1}^n c_i \leq 1 + \sum_{i=1}^n 3(r_i(x) - r_{i-1}(x)) \quad // \text{teléscopica!}$$

$$\sum c_i \leq 1 + 3(r_m(x) - r_0(x)) \quad \leftarrow \min \text{ es } 1 = 0$$

$$\leq 1 + 3 \log n \quad \leftarrow \max \text{ es } n$$

$$\leq 1 + 3 \log n$$

\Rightarrow el costo amortizado de efectuar una operación de splay es $O(\log n)$

$q(x)$ = frecuencia de acceso de x
m accesos en total

$$\text{cota inf: } \Theta\left(m + \sum_{x \in T} q(x) \log\left(\frac{m}{q(x)}\right)\right)$$

Definir

$$w(x) = q(x)/m$$

$$W = \sum_{x \in T} w(x) = 1$$

$$S(x) = \sum_{v \text{ descendiente de } x} w(v)$$

$$r(x) = \log(S(x))$$

$$\begin{aligned} \sum c_i &\leq 1 + 3(\log r_m(x) - r_0(x)) \\ &= 1 + 3(0 - \log\left(\frac{1}{m}\right)) \\ &= 1 + 3 \log\left(\frac{m}{q(x)}\right) \end{aligned}$$

$$= m + 3 \sum_{x \in T} \log q(x) \log\left(\frac{m}{q(x)}\right)$$

18/10/16

CC4102 - Log

Dominios discretos y finitos

Ordenamiento en tiempo lineal

Counting

Counting sort:

$O(n+u)$

$A = 1 \ 3 \ 1 \ 4 \ 2 \ 2 \ 1 \ 3 \ 3 \ 4$

$c \begin{cases} 3 & \rightarrow 1 \ 1 \ 1 \ 2 \ 2 \ 3 \ 3 \ 3 \ 4 \ 4 \\ 2 & \\ 3 & \\ 2 & \end{cases}$

$\leftarrow O(n+u)$

Bucket sort:

aquí se acumulan las apariciones (de c)

$c \begin{cases} 3 & \rightarrow 0 \rightarrow 3 \\ 2 & 4 \rightarrow 5 \\ 3 & 6 \rightarrow 8 \\ 2 & 9 \rightarrow 10 \end{cases}$

se mantiene las relaciones entre los mismos
números

$\hookrightarrow 1 \ 3 \ 1 \ 4 \ 2 \ 2 \ 1 \ 3 \ 3 \ 4$

$\Rightarrow 1 \ 1 \ 1 \ 2 \ 2 \ 3 \ 3 \ 3 \ 4 \ 4$

Radix sort $O(n \log u) \leftarrow$ no es muy bueno así...

17 1 0 0 0 1
2 0 0 0 1 0
5 0 0 1 0 1
14 0 1 1 1 0
8 0 1 0 0 0
1 0 0 0 0 1

\Rightarrow

2 0 0 0 1 0
14 0 1 1 1 0
8 0 1 0 0 0
17 1 0 0 0 1
5 0 0 1 0 1
1 0 0 0 0 1

\Rightarrow

8 0 1 0 0 0
14 1 0 0 0 1
5 0 0 1 0 1
1 0 0 0 0 1
2 0 0 0 0 1 0
14 0 1 1 1 0

\Rightarrow
8 0 1 0 0 0
17 1 0 0 0 1
1 0 0 0 0 1
2 0 0 0 1 0
5 0 0 1 0 1
14 0 1 1 1 0

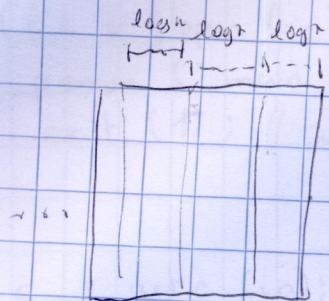
\Rightarrow

17 1 0 0 0 1
1 0 0 0 0 1
2 0 0 0 1 0
5 0 0 1 0 1
8 0 1 0 0 0
14 0 1 1 1 0

\Rightarrow

1 0 0 0 0 1
2 0 0 0 1 0
5 0 0 1 0 1
8 0 1 0 0 0
14 0 1 1 1 0
17 1 0 0 0 1

óptimo para $\log n$ bits en cada iteración



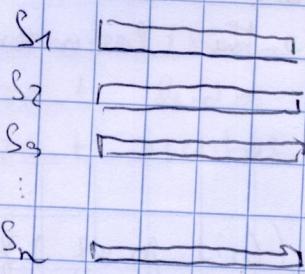
$$O(n \log \underbrace{\log n}_{\log n})$$

Ordenar n strings

$S_1 [1..m], \dots, S_n [1..m]$

$$\Sigma = [1, 0]$$

1 2 3 ... m



$$\rightarrow O(m(n + \sigma))$$

$$\Rightarrow O(m \cdot n), \text{ si } \sigma = O(n)$$

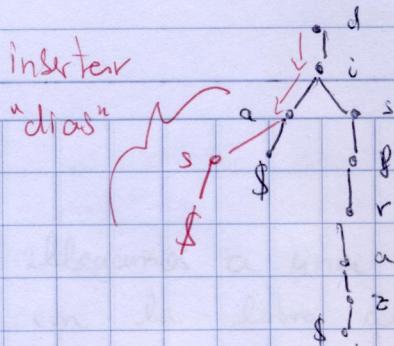
$$\hookrightarrow O\left(\frac{m \cdot n}{\log n}\right)$$

$$\log_2 n \text{ bits} = \log_2 n$$

25/10/16

CC 4102 - Log

Árboles digitales. (tries)



Dado un conjunto de strings S , terminando en un carácter especial $\$$ ($\in \Sigma$)

Un trie es un árbol con aristas rotuladas, tq:

- posee 1 hoja por string.
- La concatenación de aristas hasta la hoja es el string correspondiente
- aristas hacia nodos distintos \Rightarrow rótulos distintos
- Si tengo un patrón P ; puedo querer
 - Buscar
 - Insertar
 - Borrar
- Buscar $O(1P1)$, simplemente bajar por el trie
- Insertar:
 - buscar P y "follow"
 - Si fuera pq no hay aristas para bajar
 \Rightarrow crea la rama correspondiente
- Borrar:
 - buscar y encontrar P en una hoja
 - borro la hoja
 - mientras el padre tenga solo 1 hijo,
borro el padre y cuelgo el hijo del abuelo.

Ocupa espacio $O(L) = O(\sum_{i=1}^n |s_i|)$

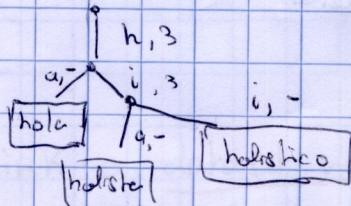
Idea:

ocupar acortar los caminos únicos que llevan a hojas.
 \Rightarrow al llegar a una hoja hay que re-cumplir P .

Árboles Patricia & Practical Algorithm To Retrieve Information Coded In Alphanumeric

- Como los arbores, pero no se permiten caminos vacíos nunca.

Si: Sí $\{hola, holista, holístico\}$



{ Patricia, solo le doy la info para elegir que camino tomar.

Un árbol patricia reemplaza caminos lunarios \nsubseteq máximos por aristas.

$$c_1 \quad c_K \\ \dots \Rightarrow \quad c_1, K$$

Un árbol Patricia con m hojas tiene a lo más m nodos internos.

Espacio: $O(|S|)$

- ¿cómo buscar?
 - * buscar desde la raíz, comparando letras del patrón con las aristas y saltando tantos caracteres como diga la arista.
 - * si llegan a una hoja, lo comparo con P.
⇒ Tiempo $O(|P|)$.

- Insertar P:

* buscar P y fallar

- si falla porque llegamos a un nodo del que no sale un hijo con la libra que venga:

* buscarnos una hoja del nodo ^{en} el que podamos bajar y RE-INSERTA MOB

- si fallamos porque si nos acabo el patrón antes de llegar a ^{una} hoja (i.e. en una arista).

* buscarnos una hoja cualquiera del hijo de esa arista y reinserturnos desde la hoja.

- si fallamos por llegar a una hoja $\neq P$, reinserturnos desde arriba.

* Re inserción de P desde una hoja:

- comparo P con la hoja y obtengo r, el mayor prefijo común

- Entro al árbol buscando r, llegaré a una arista o un nodo

* Si llego a un nodo cuelgo P como nuevo hojo

* Si llegué a una arista, la corto en 2, con un nodo de 2 hijos: la hoja P y el hijo original

- Borrar

* encontrar hoja P, borrarla

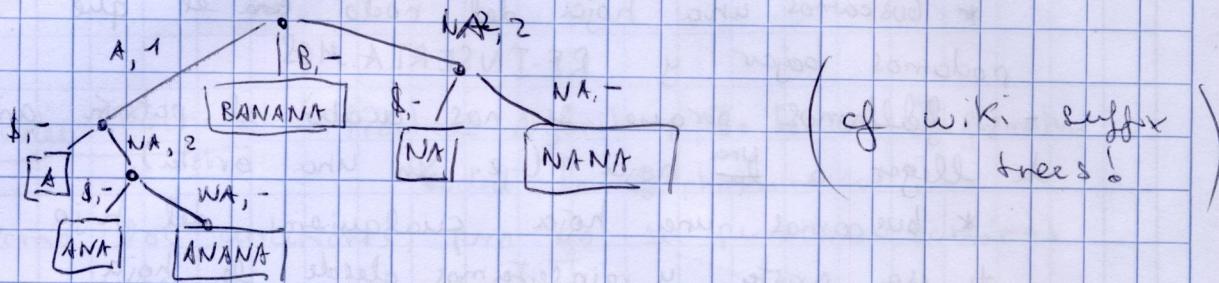
* Borrar el padre si tenía un solo hijo.

Árbol de sufijos

Sea S un string, definimos $Suf(S)$ como el conjunto de sus sufijos \Rightarrow el árbol de sufijos de S corresponde

al círculo Patricia construido sobre $\text{Suf}(S)$

- Tamaño $O(|S|)$
- Todo substring de S es prefijo de algún sufijo
⇒ esta estructura permite búsquedas en texto.



• Buscar P en S

- * Puedo per el árbol, según los caracteres de P .
 - Si no tengo la crista para bajar
⇒ " P no ocurre en S "
 - Si llego a una hoja
⇒ " P ocurre 0 o 1 vez" (resultado depende del comp^o)
 - Si P se acaba en un nodo V o en una crista hacia V :

- * Comparar P con una hoja descendiente de V .
- * Si P es prefijo de una hoja, P no ocurre en S .
- * Else, P ocurre en cada hoja descendiente de V .

Otras operaciones op^l:

- existencia → $O(|P|)$
- contar → $O(|P|)$
- reportar → $O(|P| + \#ocurrencias)$
- espacio → $O(|S|)$
- construcción → $O(|S|)$. (magia: suffix links)

• Alternativa: arreglo de sufijos

→ ordenarlos lexicográficamente. (definido por su posición en S)

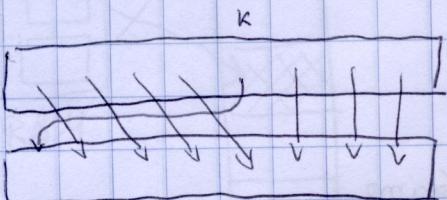
- Forman un rango
- con 2 búsquedas binarias encuentro los extremos " n "

Ocupa espacio $O(n \log n)$ construcción en $\Theta(n)$
 $O(1)$

8/11/16

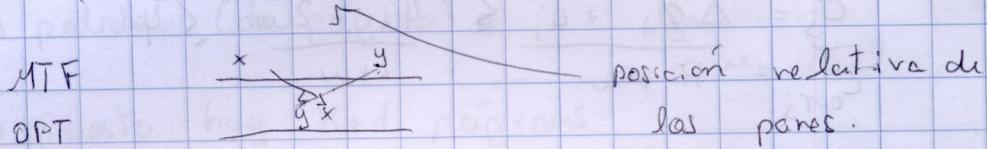
EC4102-Log

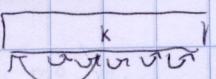
Move To Front

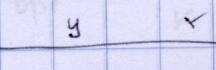
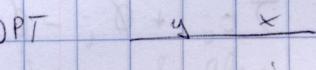


⇒ elem^s más buscados quedan al frente (parecido a un skipTree)

$\phi = 2 \cdot r$ pares invertidos (r al óptimo)



MTF  k comparaciones
 + k-1 movimientos
⇒ $c_i = 2k - 1$.

MTF  si en OPT 
 k-1 inversiones MTF crea una inversión

si en OPT 
MTF destruye una inversión

⇒ se crean r inversiones y se destruyen K-1-r

$$\begin{aligned}\Delta\phi &= 2 \cdot (r - (k-1-r)) \\ &= 2(r - k + 1 + r) \\ &= 2(2r - k + 1)\end{aligned}$$

$$c_i = 2k - 1$$

$r \leq i-1$ dado que para un i hay $i-1$ posiciones en las que podrá haber una inversión

$$\begin{aligned}\hat{c}_i &= c_i + \Delta\phi_i = 4r - 2k + 2 + 2k - 1 \\ &= 4r + 1 \leq 4i - 3 < 4c_{\text{OPT}}\end{aligned}$$

Caso dinámico: OPT puede realizar w inversiones después de la búsqueda, a costo w .

$$i \rightarrow i+w$$

↑
costo buscar
↑
costo inversiones.

$$\Delta \phi_j \rightarrow \Delta \phi_j + w$$

↓ si el algoritmo óptimo
realiza inversiones

$$\frac{\hat{c}_j}{c_{\text{OPT},j}} \leq \frac{4i}{i}$$

$$\frac{\hat{c}_j}{c_{\text{OPT},j}} = \frac{\Delta \phi_j + c_j}{i+w} \leq \frac{4i + 2w}{i+w} \leq 4$$

$\phi_0 \in O(n^2)$ en este caso!

$$\sum \hat{c}_i \leq \sum c_i + \phi_n - \phi_0$$

$$4m_{\text{opt}} + \phi_0 \leq m$$

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

$$\leq (4_{\text{opt}} + 1)m$$

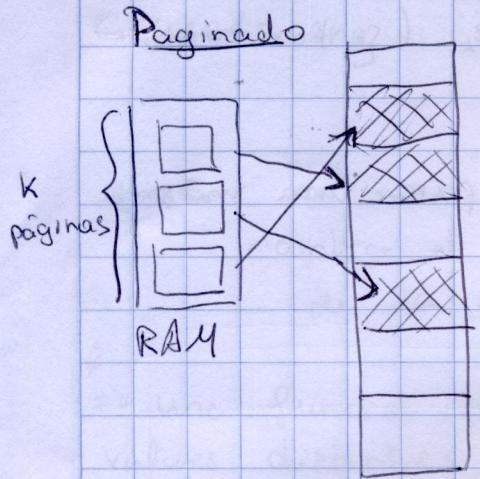
$$\sum \hat{c}_i \geq \sum c_i + \phi_n - \phi_0$$

no olvidar ϕ_0 que puede ser grande

$$\sum c_i \leq \sum \hat{c}_i + \phi_0 - \phi_n$$

⇒ en este caso MTF se demora en ajustarse, es por ello que ϕ_0 puede ser muy alto!

~~MTF~~ MTF = método de compresión adaptativa bastante utilizado.



confundirse en el paginado muestra mejor CARO.

⇒ no se puede analizar con un peor caso, pero si al comparar con el óptimo la que sale de la RAM

OPT: elige como victima la página en RAM que falta más tiempo para que vuelva a pedir

n pedidos (de página)

Supuesto: hay $K+1$ páginas distintas

el opt^{mo} elige la que más falta

⇒ tenemos K pedidos sin fallar

$$\Rightarrow C_{\text{opt}} \leq \frac{n}{K}$$

⚠ online: Existe una secuencia que me hace fallar las n veces.

↳ No se puede ser mejor que K -competitivo

LRU (least-recently-used)

→ LRU es K -competitivo.

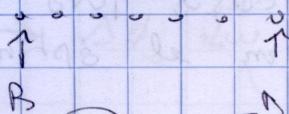


Sí en el bloque, LRU vuelve a fallar por tener

A

→ ~~desde~~ OPT falló al menos 1 vez (se han hecho $k+1$ pedidos)

- Si hay 2 fallos en el bloque para una misma página B



K pedidos \neq^s entre sí y de B.

- Si las K páginas en las que se falló son distintas entre sí y de A

- A estaba en la memoria, el óptimo habrá tenido que fallar una vez.

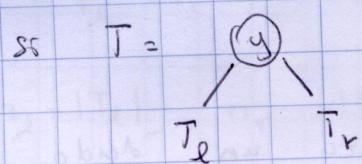
17/11/16

CC4102 - Log

Arboles aleatorizados

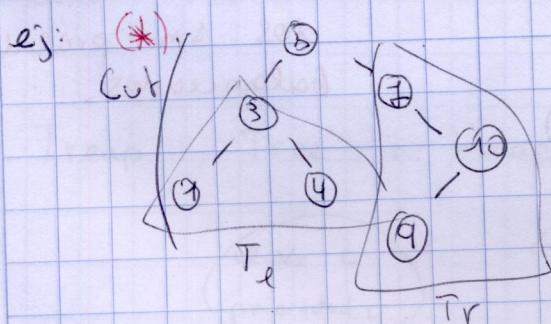
$$\text{Cut}(t, x) = (T_<, T_>)$$

si $T = \square$, ~~recibe x~~ retorna (\square, \square)



si $x < y$ $(T_<, T_>) \leftarrow \text{Cut}(T_<, x)$
retorna $(T_<, \begin{array}{c} \textcircled{y} \\ \diagdown \quad \diagup \\ T_l \quad T_r \end{array})$

sino $(T_<, T_>) \leftarrow \text{Cut}(T_r, x)$
retorna $(\begin{array}{c} \textcircled{y} \\ \diagdown \quad \diagup \\ T_l \quad T_r \end{array}, T_>)$



$$\text{Cut}(\begin{array}{c} \textcircled{10} \\ \diagdown \quad \diagup \\ 9 \quad 11 \end{array}; 3,5) = \left(\begin{array}{c} \textcircled{3} \\ \diagdown \quad \diagup \\ 2 \quad 4 \end{array}, \begin{array}{c} \textcircled{10} \\ \diagdown \quad \diagup \\ 9 \quad 11 \end{array} \right)$$

$$\text{Cut}(\begin{array}{c} \textcircled{3} \\ \diagdown \quad \diagup \\ 2 \quad 4 \end{array}; 3,5) = (\square; \textcircled{4})$$

$$\text{Cut}(\textcircled{4}; 3,5) = (\square; \textcircled{4})$$

$$\text{Cut}(\square; 3,5) = (\square, \square)$$

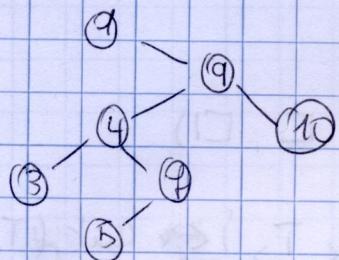
sub procesos de (*)

④ luego de hacer el Cut c/r a un valor x , obtenemos los árboles ~~en orden de inserción~~ como si nunca se hubiesen agregado valores $\geq x$; idem para $< x$. De hecho es posible agregar x como la nueva raíz.

5, 7, 10, 3, 4, 9, 1

están invertidos

si insertamos y hacemos Cut con el nuevo número:



insertar en el orden inverso!

1, 9, 4, 3, 10, 7, 5

Al insertar nuevos nodos, se lanza un "dado" de $n+1$ caras, aquél que sale se vuelve la nueva raíz (haciendo una operación Cut (T , nueva raíz))

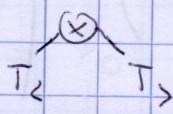
Insert (T, x)

$a \leftarrow \text{random}(1, \dots, |T| + 1)$

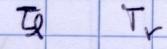
si $a = 1$

$(T_L, T_R) \leftarrow \text{Cut}(T, x)$

return



sea $T =$



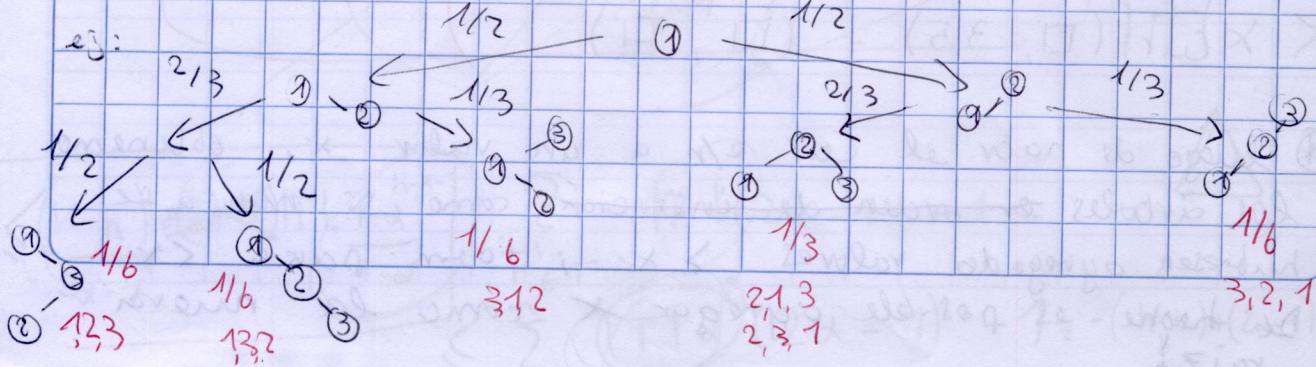
si $x < y$, retornar

$\text{Insert}(T_L, x) \quad T_R$

retornar

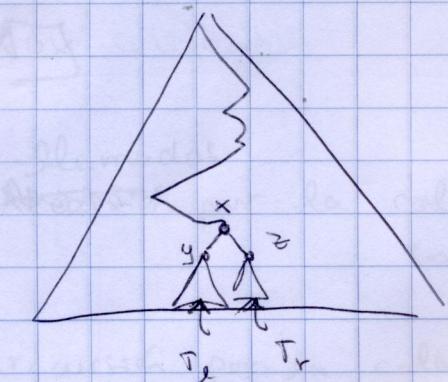
$T_L \quad \text{Insert}(T_R, x)$

ej:



⇒ árboles aleatorizados son probablemente balanceados.

Borrado



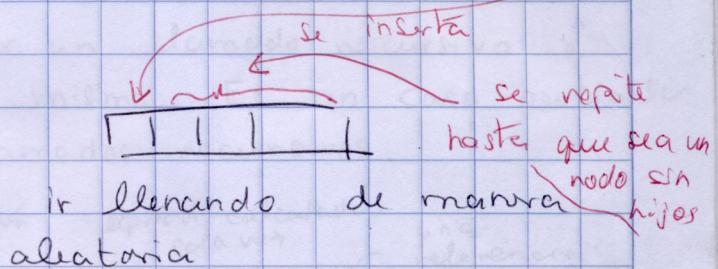
$$n_l = |T_l|, n_r = |T_r|$$

$$xy\dots z \Leftrightarrow xz\dots y$$

$$\frac{n_l}{n_l+n_r} \rightarrow \boxed{}_{n_l}$$

se elige cual de los 2 usar

$$\frac{n_r}{n_l+n_r} \rightarrow \boxed{}_{n_r}$$

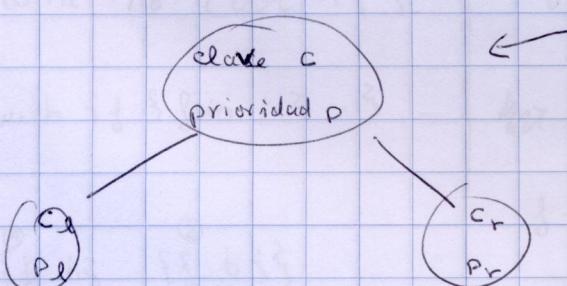


\Rightarrow la idea es preservar la invariante de permutaciones aleatorias!

Cartesian Tree construction \Leftarrow construcción NSV (C2)

Treap : Tree + Heap

árbol determinístico



$$c_l \leq c \leq c_r$$

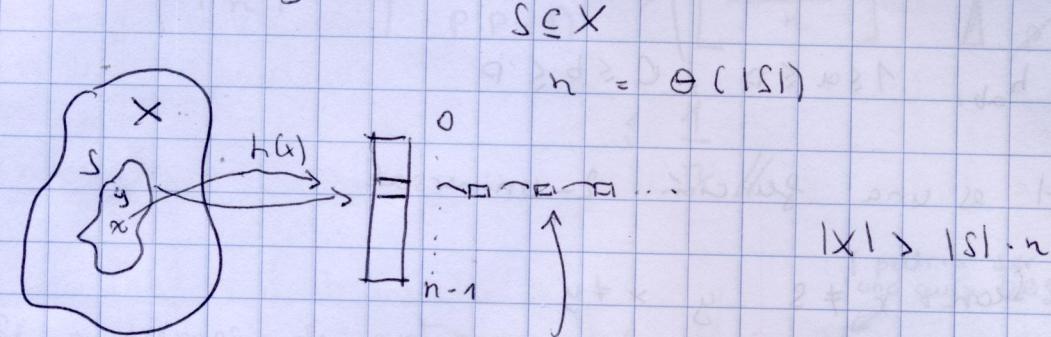
$$p > p_l \rightarrow p > p_r$$

Forma en un Treap:

- encontrar el elemento con mayor prioridad en el rango $[x_1, x_2]$ de claves.
- Idem los top-k

22/11/16

CC4102 - Log



\Rightarrow elegir una función de hashing aleatoria en el momento

\Rightarrow elegimos h aleatoriamente h v.a.i.d

prop: 2-independencia:

Dadas dos claves $x \neq y$

$$\Pr_r[h(x) = h(y)] \leq 1/n$$

$h \in H$

\Rightarrow Decimos que H es una familia 2-universal.

$$C_{xy} = \begin{cases} 1 & \text{si } h(x) = h(y) \\ 0 & \text{si no} \end{cases}$$

$$C_{xs} = \sum_{y \in S} C_{xy} = \text{largo de la lista.}$$

$$E(C_{xy}) = \Pr_r(C_{xy} = 1) \begin{cases} 1 & \text{si } x = y \\ & \text{si } x \neq y, y \in H \text{ es universal} \\ & \Rightarrow \leq 1/n \end{cases}$$

$$E(C_{xs}) \leq 1 + \frac{|S|-1}{n} = O\left(1 + \frac{|S|}{n}\right)$$

$$= O(1) \text{ si } n = \Theta(|S|)$$

$$h_{ab} = ((ax + b) \bmod p) \bmod n$$

$$H = \{ h_{ab}, \quad 1 \leq a \leq n, \quad 0 \leq b \leq p$$

Teo: H es una función 2-universal

Dern: Sean r ≠ s y × ≠ y

$$\Pr_{\substack{r \\ p}} \left((ax+b = r \bmod p) \mid (ax+b = s \bmod p) \right)$$

$$ax + b \equiv r \pmod{p}$$

$$ax + b \equiv s \pmod{p}$$

$$a(x-y) \equiv r-s \pmod{p}$$

$$a = (r-s)(x-y)^{-1} \pmod{p}$$

es la única selección para a. (dado que p es primo)

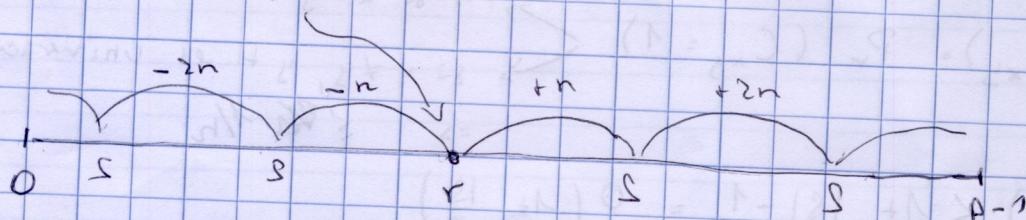
$$b = r - ax \pmod{p}$$

es la única solución para b

$$\therefore P_r = \frac{1}{r(r-1)}$$

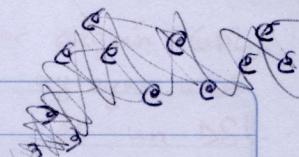
$$P_r(h(x) = h(y))$$

$$= \Pr_r \left(\underbrace{(ax+b) \bmod p}_{r} \bmod n = \underbrace{(ay+b) \bmod p}_{r} \bmod n \right)$$



p valores para
 r

cantidad de colisiones
posibles entre p y S



$$\leq p \left(\left\lceil \frac{p}{n} \right\rceil - 1 \right) \frac{1}{p(p-1)} = p \left(\left\lfloor \frac{p+x-1}{n} \right\rfloor - 1 \right) \frac{1}{p(p-1)}$$

$$\leq \frac{1}{n}$$

¡podría ser el caso de
una guía telefónica!

Si conocemos de antemano el conjunto S , podríamos querer garantizar un tiempo de consulta $O(1)$.

① Usamos una tabla de tamaño $n = |S|^2$

$\Pr(\exists x \neq y \in S \text{ que colisionan})$

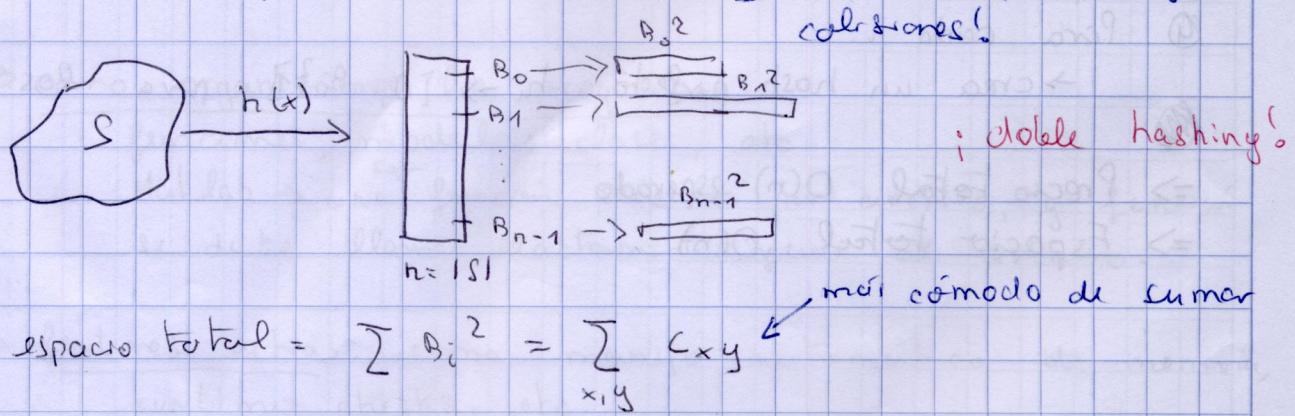
$$\leq \sum_{x \neq y \in S} \Pr(C_{xy} = 1) \leq \sum_{x \neq y \in S} \frac{1}{|S|^2} = \frac{|S|(|S|-1)}{2} \cdot \frac{1}{|S|^2} \leq \frac{1}{2}$$

(\Rightarrow hay una probabilidad de $1/2$ que no hagan colisiones!)

∴ el nro esperado de $h \in H$ que debemos probar es 2.

∴ en tiempo $O(|S|)$ esperado encontramos un h perfecto para S .

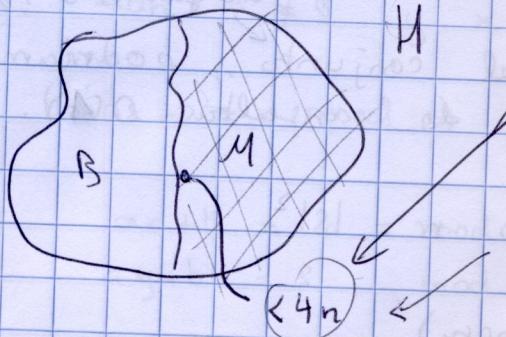
② Usando espacio $O(|S|) \Rightarrow$ ¡muy probable que haya colisiones!



$$= n + \sum_{x,y} c_{xy}$$

$$\leq n + k(n-1) \cdot \frac{1}{n} < 2n$$

construir todas las tablas toma tiempo ~~$O(n^2)$~~ $O(181)$.



H

Tamaño esperado de la doble función de hashing en promedio

el mínimo de las funciones malas debe cumplir con la condición, si no no podemos llegar al promedio $\leq 2n$

Si nos aseguramos que el tamaño cumpla la condición, entonces podemos asegurar el tiempo $O(1)$, esto con $1/2$ de probabilidades de sacar una buena función de hashing.

Hashing perfecto: (algoritmo resumido)

① elige una función de hashing distribución $h \rightarrow [0, n-1]$ de una familia universal H

② calcula los B_i

③ repite hasta que $\sum B_i^2 \leq 4n$

④ Para cada i :

→ crea un hash perfecto $h_i \rightarrow [0, B_i^2]$ para los B_i

⇒ Precio total $O(n)$ esperado

⇒ Espacio total $O(n)$

29/11/16

CC 4102 - Log

Algo más Aproximados

→ pbmas NP-completos

→ decisión → app°.

Def°: dada una fct° de costo C , un pbma de optimización es encontrar el obj de un conjunto de mínimo o máximo costo.

Un algoritmo que encuentra una solución de costo $A(I)$ para un input I , es una:

$\rho(n)$ -optimización si $\forall n$

$$\max_{|I|=n} \frac{A(I)}{\text{Opt}(I)} \leq \rho(n)$$

(para minimización)

$$\max_{|I|=n} \frac{\text{Opt}(I)}{A(I)} \leq \rho(n)$$

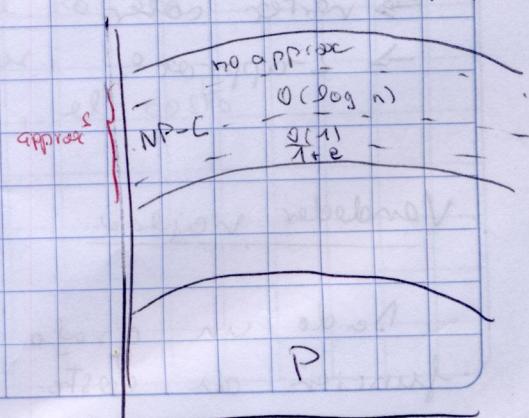
(para maximización)

$A(I)$ es approx
↳ debe usar algo más
 T (polinomiales)
 $\text{Opt}(I)$ es la sol° optim
↳ puede ser $T(\log p)$.

donde $\text{Opt}(I)$ es el costo óptimo.

Def°: un esquema de appr° polinomial, es un algo° de aproximación que recibe un parámetro adicional $\epsilon > 0$, y entrega una $(1+\epsilon)$ -appr°. Su costo es polinomial en n (siendo ϵ una ct).

ej: $O(n^{2/\epsilon})$

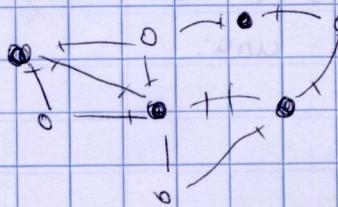


DNNP

Def: Un esquema de appr^o completamente polinomial es un esquema de appr^o polinomial cuyo costo también es un polinomio en la variable $1/\epsilon$:
ej: $O\left(\frac{1}{\epsilon^2} \cdot n^3\right)$

Vertex Cover

Dado un grafo $G = (V, E)$, encontrar un $V' \subseteq V$ de tamaño mínimo tq $\forall (u, v) \in E, u \in V'$ ó $v \in V'$.



Una 2-appr^o:

$$V' \leftarrow \emptyset$$

mientras $E \neq \emptyset$

elijo $e(u, v) \in E$ cualquiera

$$V' \leftarrow V' \cup \{u, v\}$$

elimino de E toda arista incidente en u o en v .

\Rightarrow vertex cover

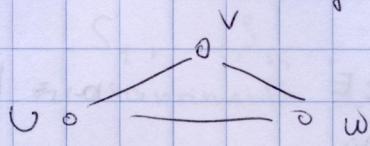
\Rightarrow 2-appr^o.

Vendedores viajeros

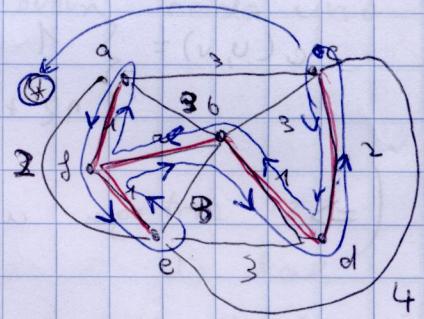
Dado un grafo $G = (V, E)$ completo con una función de costo $c: E \rightarrow \mathbb{R}^+$, asociando a los

aristas, encuentre un circuito hamiltoniano de costo mínimo

→ caso particular: los costos satisfacen la desigualdad del triángulo $c(u,w) \leq c(u,v) + c(v,w)$



2-approc°:



① construimos un MST del grafo como todo circuito es un camino + una arista y un camino es un árbol que conecta G, el costo del MST es \leq el menor costo de un camino hamiltoniano.

② construimos un tour euleriano del MST

cdb f a f e f b d c ④ (ida vuelta)

el resultado es un circuito que toca todos los nodos aunque puede repetir nodos
toda arista del MST se recorre 2 veces
(al bajar y al subir)
si su costo es $\leq 2 \cdot \text{opt}$

③ eliminaremos los nodos del tour que no son en primera ocurrencia de izq a der.
por la desigualdad triangular, el costo no crece.

Vendedor viajero, caso general



Supongamos que tengo una ℓ -aprox.

Dado un grafo $G = (V, E)$, puede determinar si tiene o no un circuito hamiltoniano definiendo un grafo completo con los nodos de V , y estos costos

$$c(u, v) = \begin{cases} 1 & \text{si } (u, v) \in E \\ n+1 & \text{si no} \end{cases} \quad n = |V|$$

(\Rightarrow si entregar un costo $n+1$, entonces si es cierto)

$$C(n) = \max_{\{I \mid |I|=n\}} C[A(I)]$$

costo alta

$$C[O(I)]$$

costo óptimo

$$|S| = n$$

K conjuntos S_1, \dots, S_K de r elem^s cada uno
 ↳ no necesariamente disjuntos.

$$\text{se cumple que } \frac{k}{2^r} \leq \frac{1}{4}$$

⇒ colorear los elem^s de rojo o azul de tal manera que ningún conjunto quede monocromático

(a) MC, color válido con $P(\cdot) = 1/2$
 + analizar peor caso

indicatriz:

$$c_i = \begin{cases} 1 & \text{si los colores } S_i \text{ no es monocromático.} \\ 0 & \text{si no} \end{cases}$$

Para cada $\&$ conjunto S_i :

para cada elemento a_1, \dots, a_k : < si no este coloreado
 coloreamos azul con $P(\cdot) = 1/2$
 (coloreamos rojo con $P(\cdot) = 1/2$) rojo sea

$$P(\text{monocromático}) = \frac{1}{2^K} \quad \text{para un subconjunto } S_i$$

$$P(\text{todos monocromáticos}) = \frac{K}{2^r} \leq \frac{1}{4}$$

además

$$\begin{aligned} E(X) &= E\left(\sum_k c_i\right) = \sum_i E(c_i) \\ &= \sum_i \frac{1}{2^r} = \frac{K}{2^r} \leq \frac{1}{4} < 1/2 // \end{aligned}$$