

Auxiliar 9 - Algoritmos Online

CC4102 - Diseño y Análisis de Algoritmos

Profesor: Gonzalo Navarro Auxiliar: Jorge Bahamonde

16 de Noviembre del 2015

1 Online Set Covering

En el problema online de set covering, se nos entrega (U, S) , donde U contiene n elementos y S contiene m subconjuntos de U . Luego se entrega de forma online una secuencia de elementos e_1, e_2, \dots, e_t . El algoritmo que resuelve el problema debe construir un subconjunto R de S . Cuando llega un nuevo elemento, si no está cubierto por los subconjuntos de R , debe escogerse un conjunto de S que lo contenga y agregarlo a R (se permite agregar más conjuntos, si se desea).

1. Demuestre que no se puede obtener (usando un algoritmo determinístico) un radio competitivo mejor que $\log_2 n$. **Hint:** Considere $n = 2^d$, $U = \{0, 1\}^d$, $S = \{S_i\}_{i=0}^d$, siendo $S_0 = \{0^d\}$ y para $1 \leq i \leq d$, considerando S_i como el conjunto de las cadenas con 1 en la i -ésima posición.
2. Demuestre que no se puede obtener un radio competitivo mejor que $\Omega(\frac{\log m}{\log n})$ con un algoritmo determinístico. **Hint:** Considere $U = [n]$ y S los subconjuntos de U con \sqrt{n} elementos (luego $m = |S| = \binom{n}{\sqrt{n}}$).

2 El problema de los k servidores

Considere el escenario donde tiene k puntos (*servidores*) en un *espacio métrico* (donde está definida una función de distancia d simétrica, no negativa y que cumple la desigualdad triangular) y una secuencia de puntos (*peticiones*) que debe atender. Cada vez que llega una petición, un servidor debe moverse hacia esa posición.

El problema *online* consiste en minimizar la distancia recorrida por todos los servidores luego de n peticiones, sin saber la secuencia de puntos a atender.

1. Sea \mathcal{A} un algoritmo online para el problema de los k servidores bajo un espacio métrico arbitrario con al menos $k + 1$ puntos. Pruebe que el radio competitivo de \mathcal{A} es al menos k .
2. Utilizaremos una función potencial para esta parte. Una función de potencial Φ demuestra un radio competitivo r de un algoritmo \mathcal{A} si satisface las siguientes condiciones:
 - Φ es no negativa.
 - Cada respuesta a una petición del algoritmo óptimo incrementa Φ no más de r veces el costo cargado al algoritmo por esa respuesta.
 - Cada respuesta de \mathcal{A} disminuye el potencial por al menos el costo cargado a \mathcal{A} por esa respuesta.

Un algoritmo es r -competitivo si existe una función de potencial que cumpla estas propiedades (Propuesto: ¿Puede demostrarlo?)

Considere el problema de k servidores en una línea y el siguiente algoritmo:

- Si todos los servidores están al mismo lado de la petición, se envía el servidor más cercano.
- Si una petición está entre dos servidores, envía los dos a velocidad constante, deteniéndose cuando uno de ellos llega al objetivo.

Utilice una función de potencial Φ que demuestre un radio competitivo k para este algoritmo.

3 Tareas y Procesadores

Tenemos m procesadores idénticos. Como entrada recibimos una lista t_1, t_2, \dots, t_n de tareas con tiempos de procesamiento $p_1, p_2, \dots, p_n > 0$, respectivamente. Las tareas son recibidas secuencialmente, y sólo cuando una tarea llega conocemos su tiempo de procesamiento. Cada tarea debe ser asignada a una máquina inmediatamente, y la decisión no puede ser cambiada. La *carga* de un procesador es la suma de los tiempos de procesamiento de todas las tareas que le son asignadas. El costo de un algoritmo que resuelve el problema de asignación es la máxima carga entre sus procesadores.

Considera el siguiente algoritmo para el problema anterior. Cada tarea es asignada al procesador con menor carga (en caso de empate, se elige cualquiera).

1. Demuestre que este algoritmo es $(2 - \frac{1}{m})$ -competitivo.
2. Demuestre que esta cota es óptima para el algoritmo.

Auxiliar #7

"Algoritmos Online"



Online Set Covering.

(d)

Universo U

Secuencia S , $|S| = m$, $x \in S \Rightarrow x \subseteq U$

Secuencia: e_1, e_2, \dots, e_n

$$n = 2^d, U = \{0, 1\}^d$$

a)

$S_0 = \{0\}^d$, $S_i = \{ \text{cadenas con un } 1 \text{ en el } i\text{ésimo dígito} \}$
con $i=1, \dots, d$.

Sea un adversario que entrega e_1, \dots, e_n

- $e_0 = 1 \dots 1 \dots 1 \dots \Rightarrow A$ elige un $S_{i_1}, i_1 \geq 0$
- e_1 es la cadena con un 0 en i_1 y $i_2 \rightarrow 1 \dots 0 \dots 1 \dots 1 \dots$
 $\Rightarrow A$ responde con S_{i_2} con $i_2 \neq i_1$
- e_2 tiene un 0 en i_1 y i_2

$\Rightarrow e_j$ tiene 0's en i_1, \dots, i_j , donde $S_{i_1} \dots S_{i_j}$ son los conjuntos que A va agregando. Cuando queda un dígito en 1 (i^{*}) paramos.

• El óptimo usa $S_{i^{*}}$ \Rightarrow uso 1 conjunto

\Rightarrow competitividad $\log_2(n)$ (?)

$$(b) \Omega = \left(\frac{\log m}{\log n} \right)$$

$$U = [n] \quad \{1, 2, \dots, n\}$$

S : subconjunto de U de tamaño \sqrt{n}

$$\Rightarrow m = |S| = \left(\frac{n}{\sqrt{n}} \right)$$

El adversario es sencillo. Para cada paso de A , elijo un x_i no cubierto.

- El adversario puede hacer esto al menos \sqrt{n} veces

- \Rightarrow Usa \sqrt{n} conjuntos

- El Opt usa $(\{x_i\})$

- \Rightarrow Es \sqrt{n} -competitivo al menos.

Queda ver que en este caso $\sqrt{n} \in \Omega \left(\frac{\log m}{\log n} \right)$

Es decir $\sqrt{n} \in \Omega \left(\frac{\log \left(\frac{m}{\sqrt{n}} \right)}{\log(n)} \right)$ \rightarrow Usar stirling en la combinatoria. $(n! \sim n \log n - n)$

P2

- Tenemos k servidores

- $d(x, y) \geq 0, d(x, y) = d(y, x)$

$$d(x, y) + d(y, z) \geq d(x, z)$$

① Peticiones en forma de puntos del espacio

Usaremos un espacio de $k+1$ puntos: $1, 2, \dots, k+1$

Pueden haber varios en un mismo punto.

\rightarrow Si tenemos k -servidores, hay al menos un punto desocupado

\rightarrow Vamos a tener k adversarios

Mostaremos que:

$$\sum_i \text{Adv}_i \leq A \Rightarrow \text{debe haber uno tal que } \text{Adv}_i \leq \frac{A}{k}$$

Sin pérdida de generalidad, "A" parte con los servidores 1, 2, ..., k.
Además, supondremos que A es lazy (sólo mueve el servidor que llegará la petición).

↳ Todo algoritmo no lazy puede ser lazyificado sin costo.

Adv: Tiene servidores en todos los puntos excepto i.

Invariantes del equipo de adversarios:

- * Cada adversario siempre tiene un servidor listo en la siguiente request.

- * Para cada punto cubierto por A, \exists un Adv que lo tiene libre.

Operación de los adversarios:

↳ Si A mueve un servidor de i_1 a i_2

$\Rightarrow \text{Adv}_{i_1}$ mueve $i_2 \rightarrow i_1$ (y pasa a ser Adv_{i_2})

- * La secuencia de peticiones es pedir siempre el punto que A no ocupa \Rightarrow Todos los adv lo tienen cubierto.

- * Si A mueve un servidor una distancia d, un único adversario hace lo mismo en la otra sección.

$$\Rightarrow \sum_{j=1}^k \text{Adv}_{j^*} = A \Rightarrow \exists \text{Adv}_{j^*} \text{ tal que } \text{Adv}_{j^*} \leq \frac{A}{k}$$

$\Rightarrow A \geq k \cdot \text{Adv}_{j^*} \geq k \cdot \text{Opt} \Rightarrow A \text{ es al menos } k\text{-competitivo.}$

$$\textcircled{2} \quad \phi \geq \theta$$

$$\int \text{op. de Opt} \rightarrow \Delta \phi \leq k \cdot \text{Copt}$$

$$\int \text{op. de A} \rightarrow \Delta \phi \leq -CA$$

$$\phi = k\psi + \theta \quad \left. \begin{array}{l} S_i: \text{Servidor de A} \\ a_i: \text{Servidor de Opt} \end{array} \right\}$$

$$\left. \begin{array}{l} \psi = \sum_{i=1}^k d(s_i, a_i) \\ \theta = \sum_{i < j} d(s_i, s_j) \end{array} \right\} \text{La idea de este } \phi \text{ es que} \\ \psi \text{ sirve para } \textcircled{1} \text{ y } \theta \text{ para } \textcircled{2}$$

- Si Opt mueve un servidor de $a_i \rightarrow a'_i \rightarrow \theta$ no cambia

$$\Delta \phi = k \cdot \Delta \psi = k(d(a'_i, s_i) - d(a_i, s_i))$$

$$\leq d(a'_i, a_i) \cdot k \text{ (por } \Delta)$$

$$= k \cdot \text{Copt} \Rightarrow \textcircled{1} \checkmark$$

- Ante una petición r :

Si r está a la izquierda de todos los servidores, sólo mueve el más cercano una distancia x .

↳ Opt ya tiene un servidor ahí

⇒ ψ disminuye en x

$$\theta \text{ crece en } (k-1)x \rightarrow \Delta \phi = k \Delta \psi + \Delta \theta \\ = -kx + (k-1)x \\ = -x \\ \Rightarrow \text{Cumple (2)} \checkmark$$

- Análogo si r está a la derecha de todos.

- 1) Si r está entre s_i y s_{i+1} y a_j es el servidor Opt en r_j
- 2) Si $j \leq i \Rightarrow$ Si se acerca a a_i una distancia x
 s_{i+1} se aleja de a_{i+1} una distancia x .
 $\Rightarrow \Delta \Psi = 0$.

θ ? Sólo cambian los términos con s_i y s_{i+1}

$d(s_i, s_j) + d(s_{i+1}, s_j)$ es constante.

(Uno se aleja y otro se acerca)

↳ Sólo queda $d(s_i, s_{i+1})$ que disminuye en $2x$ el costo.
 $\rightarrow \Delta \phi - 2x \leq -C_A //$

ALGORITMOS ALEATORIZADOS Y PROBABILÍSTICOS

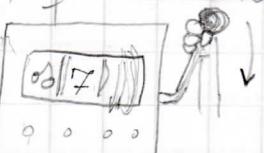
2015年11月17日(火)

(No determinísticos)

- pueden NO siempre terminar
 - pueden equivocarse
 - pueden hacer distintas cosas frente al mismo input.

{ probabilístico (Las Vegas)
{ probabilístico (Montecarlo)
{ aleatorizado.

Existen algoritmos que son probabilísticos y aleatorizados a la vez. Los que son solo probabilísticos, al equivocarse con cierto input, lo hacen consistentemente siempre. Es mejor en ese caso, que sea también aleatorizado.



Puedo estar todo el día en un casino en Las Vegas

Ej: Sacar un pez "grande" del lago
grande $\equiv \geq$ mediana

$\frac{1}{2}$ grande
 $\frac{1}{2}$ chica

$\max(k \text{ pieces})$

No es grande
con prob. $\frac{1}{2^k}$

m

c

P

en temps

Cómo sé que me equivoco?
Puedo implementar algoritmos
que responde correctamente con
alta probabilidad, pero no puedo siempre
saber si efectivamente me equivoco
o no.

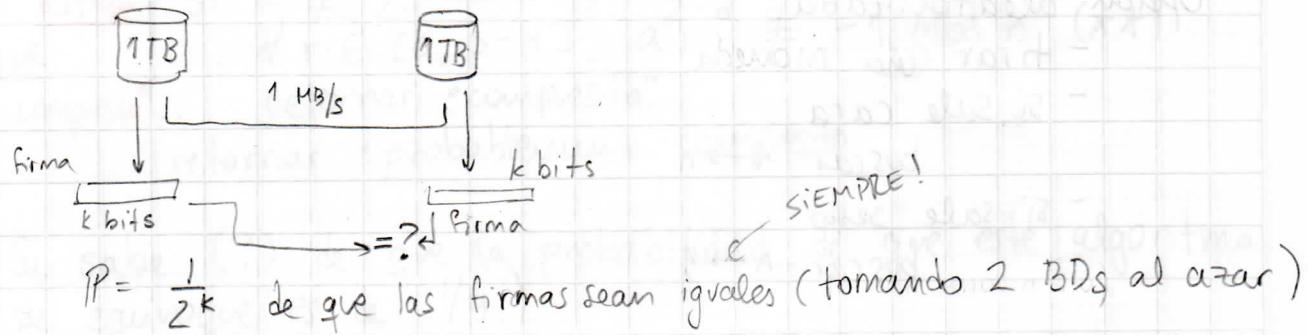
Ahora, qué pasa si sabemos que la mediana mide 1 metro? Sacamos peces hasta sacar uno grande. Problema! Puedo sacar más peces de lo que aguanta mi bote hasta que salga el pez que quiero. Puedo demorarme demasiado...

→ Las Vegas.

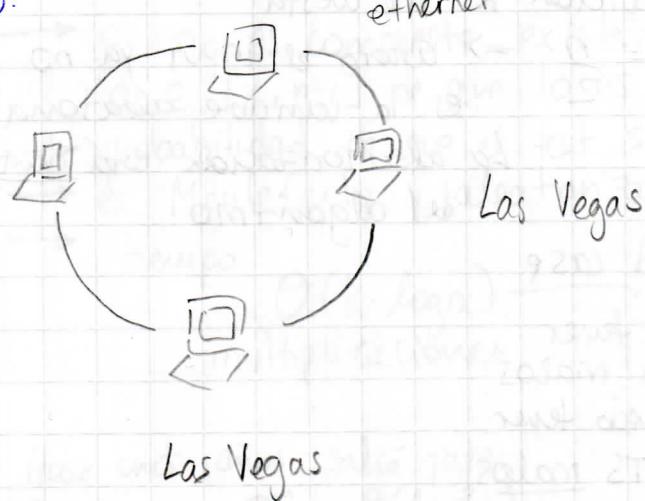
- Monte Carlo: tiempo fijo ○
prob. error conocida ○
1 sided / 2 sided errors X puede equivocarse en decir YES siempre

- Las Vegas: No hay error ○
tiempo esperado se puede "acotar" ○
peor caso NO se puede acotar X puede equivocarse en YES o NO

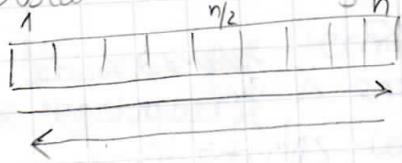
Ej. consistencia de índices de BDs.



Otro:



Otro: Buscar en un arreglo desordenado



+ importante al último

del primero al ultimo ¿y si son noticias?
del último al primero ¿y si es un
Paper?

+ importante primero

Dependiendo de la distribución de la
entrada si mi algoritmo es determinístico.
⇒ El costo promedio varía

• Opción aleatorizada:

- tirar una moneda
- si sale cara
 buscar $1 \rightarrow n$
- si sale sello
 buscar $n \rightarrow 1$

Si lo que busco está en la posición k me cuesta

$$\frac{1}{2}k + \frac{1}{2}(n-k) = \frac{n}{2} \Rightarrow \text{ahora el input ya no}$$

es la variable aleatoria.

La aleatorización está dentro
del algoritmo.

average case ≠ expected case

no me importa
el input.

El costo promedio
después de suficientes
ejecuciones tenderá
a lo mismo
siempre.

puedo tener
casos malos
pues puedo tener
INPUTS malos

Otros: primalidad, árboles binarios aleatorizados, hashing universal y perfecto, skiplists.

Primalidad Miller-Rabin

primo (n)

Sean $s, d \in \mathbb{Z}_+$

$$n-1 = 2^r \cdot d, d \text{ impar.}$$

repetir k veces

elejir $a \in [1, n-1]$ al azar

" a es testigo

de que

n es compuesto"

[si $a^d \not\equiv 1 \pmod{n}$ y

$\forall r \in [0, s-1], a^{2^r \cdot d} \not\equiv -1 \pmod{n}$ (*)

retornar "compuesto"

retornar "probablemente compuesto"

→ n NO es primo

→ n puede ser primo.

aplicaciones en criptografía

(llaves públicas y privadas, factorización)

prima de $\#s$ grandes

(Se sabe (?) de que la probabilidad de que este algoritmo se equivoque es de $1/4$)

→ Si n es compuesto, existen al menos $\frac{3}{4}n$ testigos

$a \in [1, n-1]$ de que lo es

→ probabilidad de que el test se equivoque es $1/4^k$

→ es Monte Carlo y aleatorizado, es 1 sided

→ tiempo

$\Theta(k \cdot \log n)$

multiplicaciones

(*) $\Theta(\log n) \rightarrow \begin{cases} a \\ a^2 \\ a^4 \\ a^8 \\ a^{16} \end{cases} \quad r=0$

$r=0$

(**) $\Theta(\log n) \rightarrow (a^d)^2 \quad r=1$

$r=2$

$\rightarrow (a^{2d})^2$

Hace unos años salió paper

"Primes is in P", $\Theta(\log^3 n)$

multiplicaciones

$\Rightarrow k \sim \log^7 n$

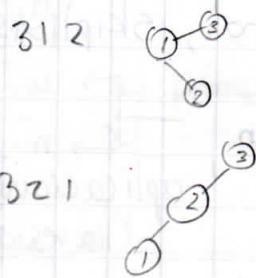
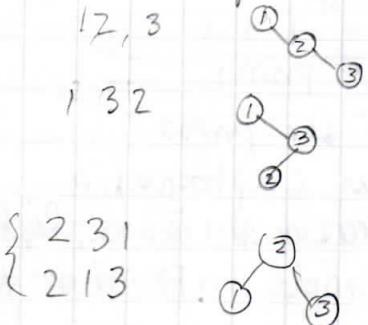
pero n es gigantesco! \Rightarrow sigue siendo preferible y menos costoso Miller-Rabin. Para k no muy grande, P de error es muy baja!

Para mismo input puedo tener árboles distintos secuencia.

2015年11月19日(木)

Árboles Aleatorizados Binarios

$S_r = 6$, depende del orden en que llegan las claves



$\text{Cut}(T, k)$ devuelve dos árboles con las claves $< k$ y $\geq k$

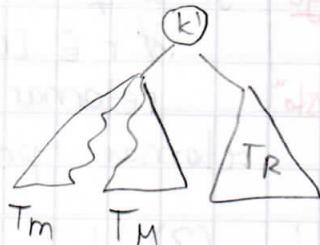
Si $T = \square$ retornar (\square, \square)

Sea $T = \begin{array}{c} (k) \\ \swarrow \quad \searrow \\ T_L \quad T_R \end{array}$

Si $k < k'$

$(T_m, T_M) \leftarrow \text{Cut}(T_L, k)$

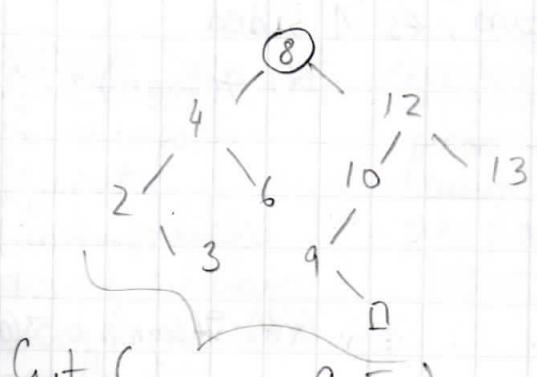
return $(T_m, \begin{array}{c} (k') \\ \swarrow \quad \searrow \\ T_L \quad T_R \end{array})$



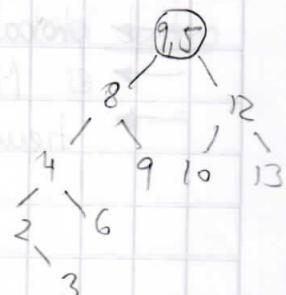
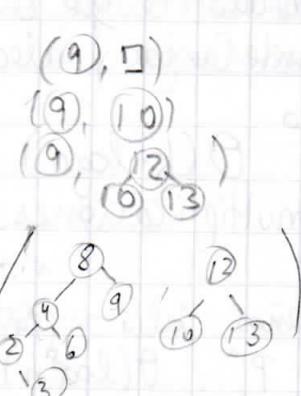
$(T_m, T_M) \leftarrow \text{Cut}(T_R, k)$

return $(\begin{array}{c} (k) \\ \swarrow \quad \searrow \\ T_L \quad T_R \end{array}, T_m)$

+ T_L

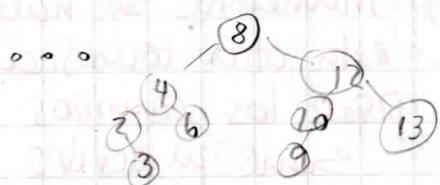
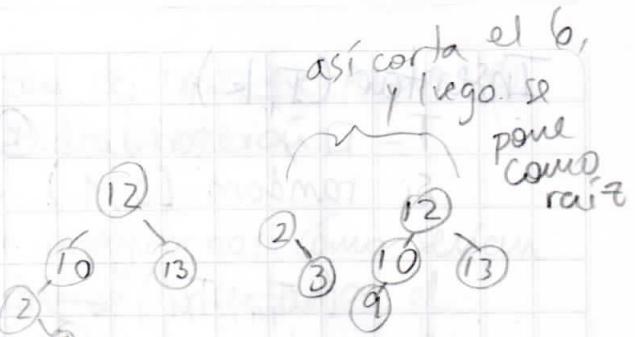


Cut (, 9,5)



Insertemos con método de Cut.

13, 9, 3, 2, 10, 12, 6, 4, 8

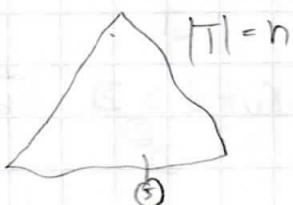


{ Es el mismo árbol que se obtiene al insertar de forma normal 8, 4, 6, 12, 10, 2, 3, 9, 13 !! }

Ahora supongamos que todas las secuencias posibles tienen dist uniforme de aparecer.

Prob (nueva clave sea la primera en insertarse)

$$= \frac{1}{n+1}$$



321 y ahora insertan 4.

En todas las secuencias de 4 números, ¿cuál es la prob. de que el 4 sea la primera en insertarse de todas las secuencias?

bajo algún algoritmo que produce árboles f's dependiendo del orden de inserción.

En nuestro caso ya NO depende el árbol. Estamos simulando todos los posibles casos en que la nueva clave podría haberse insertado primero según algún algoritmo que depende del orden

Insertar (T, k)

$T = \square$, retornar \boxed{k}

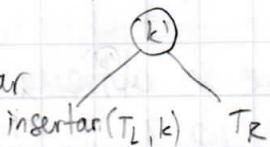
Si $\text{random}[0, 1) < \frac{1}{|T|+1}$

$(T_m, T_M) \leftarrow \text{Cut}(T, k)$
retornar \boxed{k}
 $T_m \quad T_M$

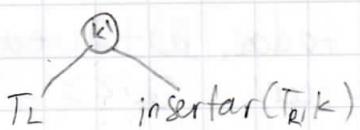
Sea $T = \boxed{k}$

$T_L \quad T_R$

Si $k < k'$, retornar



else retornar



Ej:

\square insertar(1)

insertar(2)

①

$$\xrightarrow{*} P = \frac{1}{2}$$

$$\xrightarrow{**} P = \frac{1}{2}$$

②

①

②

esos 2 árboles salen con prob. $\frac{1}{2}$ cada 1.

insertar(3) $\xrightarrow{*} P = \frac{1}{3}$ 3 es raíz \rightarrow hago Cut



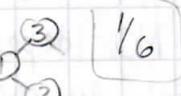
$\frac{1}{6}$

$\xrightarrow{**} P = \frac{2}{3}$ 3 no es raíz \rightarrow



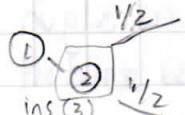
$\frac{1}{3}$

$\xrightarrow{*} P = \frac{1}{3} \rightarrow$



$\frac{1}{6}$

$\xrightarrow{**} P = \frac{2}{3} \rightarrow$



$\frac{1}{2}$

$\xrightarrow{*} P = \frac{1}{3} \rightarrow$

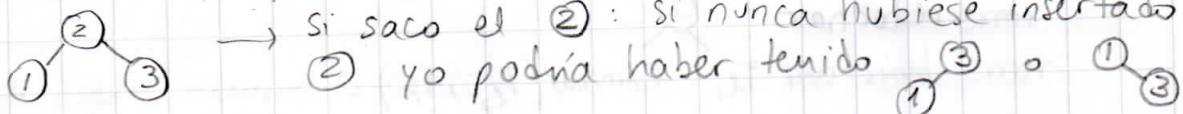


$\frac{1}{6}$

} Sin esto, el alg. es el de inserción de siempre. En cierto momento, se hace esta otra recursión, pero los caminos son parecidos, de igual costo.

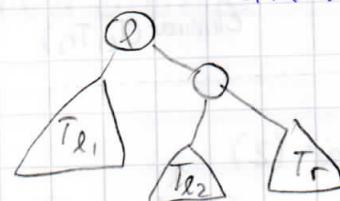
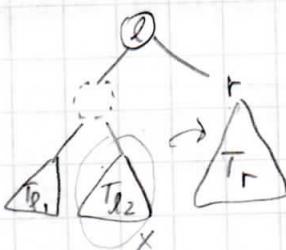
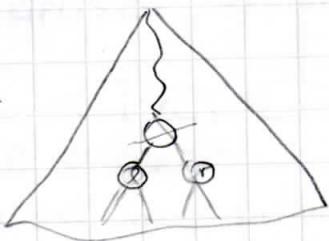
Si inserto 3 2 1 quizás el orden de cosas que pasan es distinto pero obtengo la misma distribución

¿Y cómo borramos? Tenemos que imaginarnos cómo serían los posibles árboles a los cuales nunca insertamos el elto que queremos borrar. Si queremos borrar una hoja, la borramos y listo. Pero un nodo interno?



No tenemos suficiente información !! quién fue primero?

→ simulamos con random



Borrar(T, k)

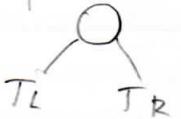
Sea $T = \begin{array}{c} k \\ / \quad \backslash \\ T_L \quad T_R \end{array}$

(borras busca, eliminas "parte" con nodo vacío y retorna árbol donde puso algo en ese nodo vacío)

si $k < k'$ retornar $\begin{array}{c} k \\ / \quad \backslash \\ T_L \quad \text{Borrar}(T, k') \end{array}$

Si $k > k'$ retornar $\begin{array}{c} k \\ / \quad \backslash \\ \text{Borrar}(T, k') \quad T_R \end{array}$
Si $|T|=1$ retornar \square
retornar Eliminar(T_L, T_R)

derivable árbol que produce raíz + T_L + T_R



Eliminar(T_L, T_R)

Sea $T_L = \begin{cases} \emptyset & \\ T_{L_1}, T_{L_2} & \end{cases}$, $T_R = \begin{cases} \emptyset & \\ T_{R_1}, T_{R_2} & \end{cases}$

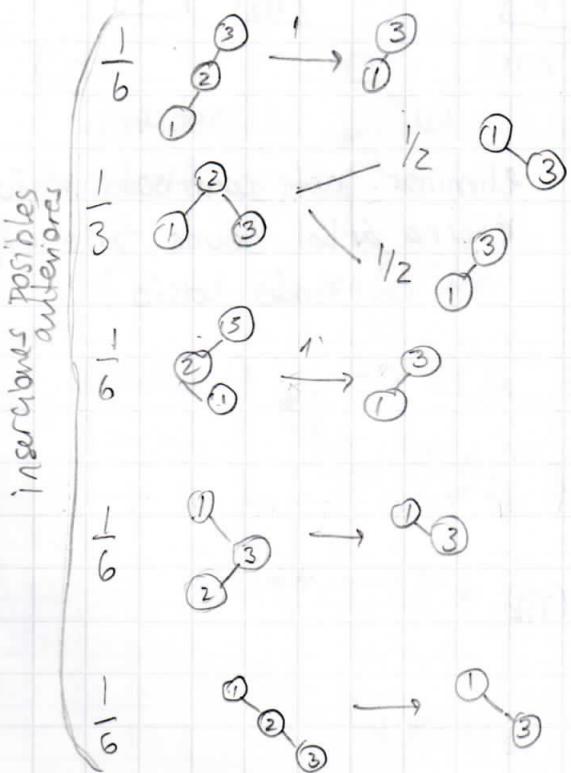
Si random $[0, 1) < \frac{|T_L|}{|T_L| + |T_R|}$

retornar $\begin{cases} \emptyset & \\ T_{L_1}, \text{ Eliminar}(T_{L_2}, T_R) & \end{cases}$

$T_{L_1}, \text{ Eliminar}(T_{L_2}, T_R)$

retornar $\begin{cases} \emptyset & \\ \text{Eliminar}(T_L, T_{R_1}), T_{R_2} & \end{cases}$

Borrar (2)



Auxiliar 10 - Algoritmos Aleatorizados y Probabilísticos

CC4102 - Diseño y Análisis de Algoritmos

Profesor: Gonzalo Navarro Auxiliar: Jorge Bahamonde

23 de Noviembre del 2015

1 Análisis de Skip Lists

2 Multiplicación de Matrices

Nos entregan 3 matrices cuadradas, A , B , C , de tamaño $n \times n$, y debemos determinar si $AB = C$. Muestre un algoritmo (probabilístico) que tome tiempo cuadrático en n .

3 Asociatividad

Se nos entrega un conjunto X con n elementos y una operación binaria \circ . Se pide determinar si \circ es asociativa en X : es decir, si para todos x, y, z en X , se cumple $(x \circ y) \circ z = x \circ (y \circ z)$. Diseñe un algoritmo (probabilístico) que resuelva este problema en $O(n^2)$ y con buena probabilidad.

Aux # 10

2015年11月23日 (月)

Algoritmos aleatorizados → tira monedas
y probabilíticos → hay P asociado al tiempo / correctitud.

Algoritmos tipo:

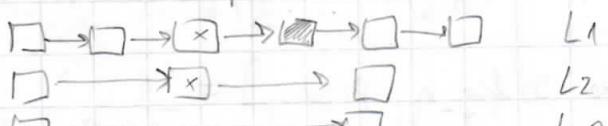
- MonteCarlo: se detienen siempre → si el algoritmo tiene 2 pero pueden equivocarse respuestas (si / no)
 - one-sided error
- Las Vegas - siempre es correcto pero pueden no detenerse
 - two-sided error



Skiplists: "aplicar búsqueda binaria a lista enlazada"



- idea 0: ordenar
- queremos los "saltos" de BB
- idea: metro expreso



$$L_1, |L_1| = l_1 = n$$

$$L_2, |L_2| = l_2$$

$$L_3, |L_3| = l_3$$

:

Si hay 2 niveles, buscar cresta:

$$\leq l_2 + \frac{l_1}{l_2} = l_2 + \frac{n}{l_2} \Rightarrow \text{minimizo c/r a } l_2 \\ (\text{¿Cuántos saltos pongo en } l_2?)$$

$$\Rightarrow C_{\min} = 2\sqrt{n}$$

Si tenemos cualquier cantidad de niveles, lo ideal sería simular un árbol binario \Rightarrow links de largo 1, 2, 4, ... ~~longitud~~

¿Cómo logro que se vea así si hay inserciones?

Idea desde árboles平衡: cada piso tiene ~~2^n nodos~~ $\frac{1}{2}$ de los nodos del piso de abajo

\Rightarrow insertar(x)

- Buscar en L_1 la posición que le corresponde a x
- Tiro una moneda hasta que salga sello
- Si tire la moneda K veces, x estará en las listas L_1, L_2, \dots, L_K
 $P[\text{un elem. esté en la lista } K] = \frac{1}{2^{K-1}} \rightarrow$ siempre tiro la moneda 1 vez.

Hay que analizar:

- Tamaño de la estructura
- Tiempo de ejecución

1) Con alta probabilidad, si hay n elementos hay $\Theta(\log n)$ pisos.

¿Por qué?

$$P[x \text{ esté en más de } c \cdot \log_2 n \text{ pisos}] = \frac{1}{2^{c \log_2 n}}$$

$$= \frac{1}{n^c}, \text{ pero esto es para 1 } x!$$

Queremos acotar la P. de que ~~c log n~~ sea más alto que $c \log_2 m$

$$\begin{aligned} P[\text{alguno de } x_1, \dots, x_n] &= P[p(x_1) \cup p(x_2) \cup \dots \cup p(x_n)] \\ &\leq \sum_i P[p(x_i)] = \frac{n}{n^c} = \frac{1}{n^{c-1}} \end{aligned}$$

el número de pisos es $> c \cdot \log_2 n$ con prob $\leq \frac{1}{n^{c-1}}$

Otra forma es ver la esperanza.

2) ¿cuántos nodos tiene la estructura? (en promedio)

• Nos gustaría saber $E(l_i)$

$$E(l_i) = \sum_{x=0}^{\infty} x \cdot P[l_i = x] = ?$$

VARS INDICADORAS: v.a con valor 0 ó 1

Definimos $X_{ij} = \text{"el elemento } x_i \in L_j"$

$$\Rightarrow E(l_i) = E\left(\sum_{j=1}^{\infty} X_{ij}\right)$$

$$= E\left(\sum_i^n \frac{1}{2^{i-1}}\right) = \frac{n}{2^{i-1}} = ?$$

$$E(n^{\circ} \text{ nodos}) = E\left(\sum_{j=1}^{\infty} l_j\right)$$

$$= \sum_{j=1}^{\infty} \frac{n}{2^{j-1}} = \sum_{i=0}^{\infty} \frac{n}{2^i} = 2n$$

$$E(n^{\circ} \text{ de pisos}) = ? \quad H_i = \begin{cases} 0 & \text{si } L_i \text{ vacía} \\ 1 & \text{si no} \end{cases}$$

$$\Rightarrow E(n^{\circ} \text{ de pisos}) = E\left(\sum_i H_i\right)$$

$$\bullet H_i \leq l_i \Rightarrow E(H_i) \leq \frac{n}{2^{i-1}}, \quad H_i \leq 1$$

$$E\left(\sum_i H_i\right) = \sum_{i=1}^{\log n} H_i + \sum_{i=\log n+1}^{\infty} H_i$$

$$\leq \sum_{i=1}^{\log n} 1 + \sum_{i=\log n+1}^{\infty} \frac{n}{2^{i-1}} \leq \log n + \sum_{i=\log n+1}^{\infty} \frac{n}{2^k} \cdot \frac{1}{2^i} = \log n + \sum_{i=0}^{\infty} \left(\frac{n}{2^k}\right) \cdot \frac{1}{2^i} \geq \log n + 1$$

$$= \log n + \sum_{i=0}^{\infty} \frac{1}{2^i} = \log n + 2$$

* Para ver el costo de la búsqueda, conviene ver el camino al revés



Partimos en L1 y nos movemos a la izq.

→ A veces encuentro un nodo que permite bajar, lo sigo
cómo pasa esto? Cuando la moneda correspondiente
salió cara.

La idea del camino inverso es llegar al último piso

=> "en cuántas monedas saco h caras?"

$$\Rightarrow E[S] = E\left[h + \sum_{r=0}^{\infty} S_r\right] = E(h) + \sum_{r=0}^{\infty} E[S_r]$$

cuantos
monedas
antes de
la 1^a cara

S_r es tal que $E(S_r) \leq 1$ y $S_r \leq \frac{n}{2^r}$ (largo esperado de

$$\Rightarrow E(S) \leq \sum_{i=0}^{\lceil \log_2 \rceil} 1 + \sum_{i=\lceil \log_2 \rceil + 1}^{\infty} \frac{n}{2^{i-1}} + E(h)$$

la lista.

$$= E(h) + \log n + 3$$

$$= 2 \log n + 5.$$

P2

$$A \ B \ C \quad AB = ?$$

\Rightarrow Naire: mult y comparar $\rightarrow O(n^3)$

- Elegir al azar un vector r
 - Calcular $A \cdot (B \cdot r)$ y comparar.
- $C \cdot r$
- Si $A(Br) = Cr \Rightarrow$ respondo si }
 ~ no }

$O(n^2)$

- Si $AB = C \Rightarrow A(Br) = Cr \quad \forall r, r$ es "testigo"

El algoritmo NO se va a equivocar cuando dice "No"
 \Rightarrow el error es one-sided

El problema es tratar de estimar la "densidad" de buenos testigos
($r = \vec{0}$ es el mal testigo)

elegimos el mal testigo

Mostraremos que si $AB \neq C$, $P[(AB)r = Cr] \leq \frac{1}{2}$

Hay que especificar la elección de r :

\rightarrow se tiene un conjunto S , con $|S| \geq 2$

\rightarrow las componentes r_i se eligen de S , con prob. i.i.d. uniforme

Sea $D = AB - C$, si $D \neq 0$, s.p.g. $d_{11} \neq 0$ (si no, permuto columnas y filas)

Si $Dr = 0 \Rightarrow (Dr) = \sum_{i=1}^n d_{1i} \cdot r_i = 0$

$$\Rightarrow r_1 = -\frac{1}{d_{11}} \left(\sum_{i=2}^n d_{1i} \cdot r_i \right)$$

único dist.
i.i.d.

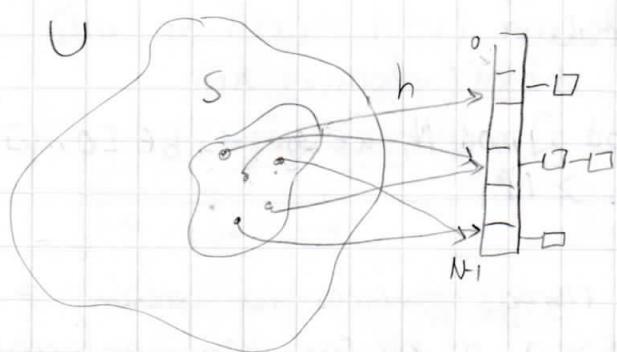
- Para cada valor de $r_2 \dots r_n$ hay sólo 1 valor posible para r_1 (si fijo $r_2 \dots r_n$, hay un solo valor para r_1 para que sea mal testigo)
- Como r_1 se elige de forma uniforme, dado indep.

$$P[r_1 \text{ sea mal testigo}] = P[r_1 = r_1^* \mid r_2 \dots r_n = \dots] \\ = \frac{1}{|S|} \leq \frac{1}{Z}$$

Si repito t veces, la prob. de equivocarse es $\frac{1}{Z^t}$
 si alguna vez dice no \Rightarrow no
 \sim \Rightarrow sí

Hashing Universal y Perfecto

2015年11月24日 (X)



S conjunto de elementos
N tamaño de la tabla
 $\frac{|S|}{N}$

Def: Una familia H de funciones de hashing es 2-universal si:

$$\forall x \neq y, \Pr_{h \in H} (h(x) = h(y)) \leq \frac{1}{N}$$
 "para todo par de ellos"

Def: Variable indicadora

3-universal es más fuerte,
sería para todo triple.

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

$$C_{xs} = |\{y \in S, C_{xy} = 1\}|$$

OBS: C_{xs} es el costo de buscar la clave x , o insertarla

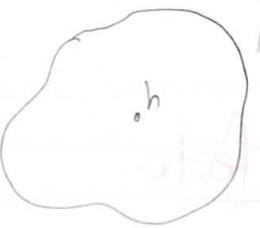
$$C_{xs} = \sum_{y \in S} C_{xy} = 1 + \sum_{\substack{y \in S \\ y \neq x}} C_{xy}$$

∴ Usando
 $N = \Theta(|S|)$
 calculas, el
 costo esperado
 es $\Theta(1)$

$$E(C_{xs}) = 1 + \sum_{\substack{y \in S \\ y \neq x}} E(C_{xy})$$

$$= 1 + \sum_{\substack{y \in S \\ y \neq x}} \Pr_{h \in H} (h(x) = h(y)) < 1 + \frac{|S|}{N}$$

2-universal



H

La prob de que h aleatoria,
para 2 (x e y) de mi conjunto fijo
de claves, cualesquiera, h se
porta bien con alta prob.

Veamos algunas familias 2-universal

$$H_p = \{ h_{ab}(x) = ((ax + b) \bmod p) \bmod N, a \in [1..p-1], b \in [0..p] \}$$

dónde p es un primo $> N$.

para cada a y b en sus resp. rangos tenemos una función \neq ,
Al crear la estructura, elegimos a y b con prob. uniforme.
(el mod N es para que queda en la tabla)
Debo hacerlo aleatorio para poder hablar de probabilidades. Dentro
de H puede haber una h muy mala, pero con cierta probabilidad
Lo bacán de 2-universal es que hay alta densidad de funciones
que se portan bien, que tienen pocas colisiones entre x e y .

Teorema: H_p es 2-universal.

Dem: Fijemos $r \neq s$ y calculemos, para $x \neq y$

$$\Pr(ax+b = r \bmod p \wedge ay+b = s \bmod p)$$

Si eso ocurre $ax+b = r \bmod p$

$$\underline{ay+b = s}$$

$$\frac{a(x-y)}{a(x-y)} = \frac{r-s}{a(x-y)} \bmod p$$

$$a = \frac{(r-s)}{(x-y)} \bmod p$$

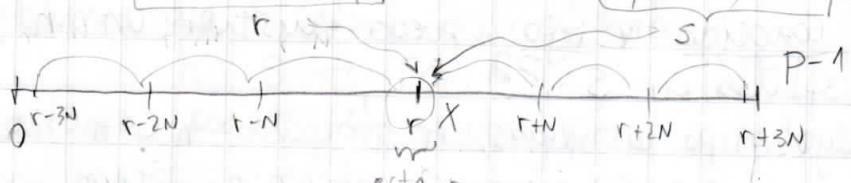
y además $b = r - ax (= s - ay)$

\therefore hay exactamente un valor de a , y un valor de b que cumplen eso

$$\Pr = \frac{1}{p(p-1)}$$

Para que $h(x) = h(y)$ necesitamos que

$$(ax + b) \bmod p \bmod N = (ay + b) \bmod p \bmod N$$



este r
no se cuenta
pues no se puede dar $r=s$
(si no $a=0$)

Hay a lo más $\lceil P/N \rceil - 1$ valores de s que colisionan con ese valor de r , $r \neq s$

→ tenemos P opciones para r

→ para cada r tenemos $\lceil P/N \rceil - 1$ opciones para s .

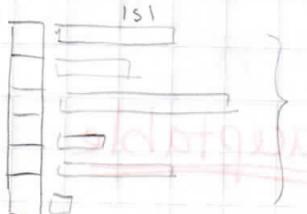
→ la prob. de que se dé cada par (r,s) específica es $\frac{1}{P(P-1)}$

$$\text{Entonces, } \Pr(h(x) = h(y)) \leq \frac{1}{P(P-1)} \cdot \underbrace{P(\lceil P/N \rceil - 1)}_{< P(P-1)} < \frac{1}{N}$$

$$\begin{aligned} \lceil P/N \rceil - 1 &= \left\lfloor \frac{P+N-1}{N} \right\rfloor - 1 \\ &\leq \frac{P+N-1}{N} - 1 \\ &= \frac{P-1}{N} \end{aligned}$$

* hay que escoger algún primo más grande que N .

Si U es más grande que $|S| \cdot N$ siempre hay un h que hace caer muchos en una misma celda.



siempre hay una celda
con $|S|$ o más elementos
colisionados

Hashing perfecto

Dado un S conocido y fijo, puedo construir un h que no genere colisiones en S ??

Puedo hacer algo tipo las Vegas, ir probando $h \in H$ hasta encontrar una función que no produzca colisiones. Pero qué pasa si H es muy grande?

Elijo $h \in H$ 2-universal

$$\Pr(h(x) = h(y)) \leq \frac{1}{N}$$

$$\Pr(\exists x \neq y, h(x) = h(y)) \leq \sum_{x \neq y \in S} \Pr(h(x) = h(y))$$

$$= |S|(|S|-1) \cdot \frac{1}{N}$$

cota para elegir
 h no perfecta

∴ si elijo $N = |S|^2$ (tabla grande)
entonces $\Pr(h \text{ no es perfecta}) < \frac{1}{2}$

Luego es poco probable

que un algoritmo Las Vegas demore mucho en probar y encontrar

h que no produzca colisiones.

(El trabajo de probar cada h toma $\Theta(|S|)$, e inicializar
puede hacerse en $\Theta(1)$, si tabla es grande y uso paos ellos)

⇒ es un alg. Las Vegas con tiempo esperado $\Theta(|S|)$

↓
número "cte" → proba
de intentos

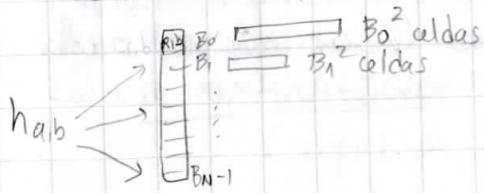
Pero $N = |S|^2$ es inaceptable

Queremos tiempo de construcción lineal y espacio lineal $\mathcal{O}(|S|)$

Probamos una función distribuidora.

$$h: S \rightarrow [0..N-1] \quad N = |S|$$

Contamos cuántos elementos pone en cada celda, B_i^2



Elegí una $h \neq$ para cada celda.

a cada celda le construyo otra tabla de tamaño cuadrático para buscar rápidamente ahí con alta probabilidad de no colisión. Los B_i^2 no son muy grandes pues ya está distribuidos

$$\text{Cuánto es el valor esperado de } \sum_{i=0}^{N-1} B_i^2 = \sum_{x,y} C_{xy}$$

$\rightarrow x, y, z \quad B^2 = 9 \quad x, y, z \text{ colisionan}$

$$\begin{array}{lll} C_{xx} = 1 & C_{yx} = 1 & C_{zx} = 1 \\ C_{xy} = 1 & C_{yy} = 1 & C_{zy} = 1 \\ C_{xz} = 1 & C_{yz} = 1 & C_{zz} = 1 \end{array}$$

$$\sum_{i=0}^{N-1} B_i^2 = \sum_{x,y} C_{xy} = N + \sum_{x \neq y} C_{xy}$$

$$\begin{aligned} E\left(\sum_{i=0}^{N-1} B_i^2\right) &= N + \sum_{x \neq y} E(C_{xy}) \\ &\leq N + \overbrace{N(N-1)}^{\text{cantidad de pares } (x,y)} \frac{1}{N} \end{aligned}$$

$$< 2N$$

(12) Puede ser que haya una función que tire todos los eltos a una misma celda i , luego $B_i^2 = N^2$ y $B_j^2 = 0$
 $\forall j \neq i$.

Pero si es un conjunto Σ universal, la esperanza de la suma de B_i^2 es lineal. Pero no es suficiente% fuerte
Puedo que hay una función muy muy buena
y el resto son relativamente malas.



todas las buenas deben producir $\sum B_i^2 < 4N$.
porque si \exists buena $\geq 4N \Rightarrow$ todas las malas $\geq 4N$
 $\Rightarrow E(\cdot) \geq 2N$

Si hay una buena que es mala, luego todas las malas son peores.
(y al más hay $\frac{1}{2}$ buenas, $\frac{1}{2}$ malas.)

Entonces debo usar Las Vegas para encontrar distribuidora que produzca $\sum B_i^2 < 4N$

(Recordar que en Las Vegas estoy escogiendo h al azar con repetición,
por eso pude demorar ∞)

Encontrar h distribuidora toma $\Theta(|S|)$ y pasar celda por celda asignando las otras h es $\Theta\left(\frac{\sum B_i^2}{4N}\right) = \Theta(|S|)$
(pues $N = |S|$)

Una vez que la estructura se construyó, es determinista.

2015年11月26日(木)

ALGORITMOS APROXIMADOS

Problemas de decisión NP-completos (clique tamaño K)

☰ Problemas de optimización también difíciles. (máx clique?)
(minimizar/maximizar)

Todo probl. de opt., tiene un probl. de decisión asociado. Podría usar uno para responder el otro.

Def: Un algoritmo es un $\rho(n)$ -aproximación a un problema de optimización si,

• input de tamaño n ,

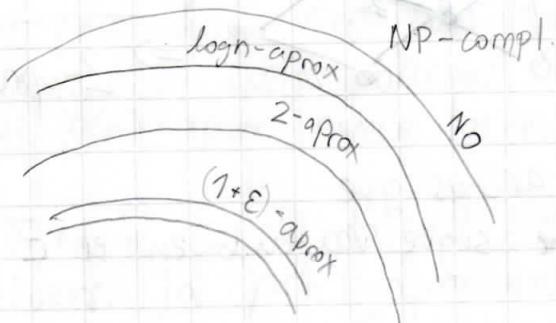
$$\max \left(\frac{C}{C^*}, \frac{C^*}{C} \right) \leq \rho(n)$$

alg de
minimizació
n > 1 alg de
máx.

donde $C =$ valor que encuentra el algoritmo

$C^* =$ valor óptimo.

probl. de decisión NP-compl,
se dejan aproximar con
un probl. de opt. 2-aprox,
log n-aprox, $1+\epsilon$ aprox



más deseable
 \sim
 $\mathcal{O}(n^{2/\epsilon})$, $\mathcal{O}\left(\frac{1}{\epsilon^2} n^3\right)$ ← (el costo crece polinomialmente cuando ϵ disminuye)

Def: Un esquema de aproximación polinomial es un algoritmo de aproximación que recibe un parámetro extra ϵ y produce una $(1+\epsilon)$ -aproximación.
Su costo es una función de n y de ϵ , polinomial en n para todo ϵ fijo.

Si la función también es un polinomio en $1/\epsilon$, se llama esquema de aproximación completamente polinomial.

Nota: los problemas de decisión se dejan "trasladar" de uno NP a otro NP.
 Pero los prob. de optimización quizás no.

VERTEX COVER: (prob. de decisión: existe vertex cover de tamaño k ?)

Dado un grafo $G(V, E)$ elegir un subconjunto mínimo $V' \subseteq V$ que cubra todas las aristas.

Vamos a hacer una 2-aprox. muy simple.

VC:

$$V' \leftarrow \emptyset$$

mientras $E \neq \emptyset$

elegir $(u, v) \in E$

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

sacar de E toda arista incidente en $u \circ v$.

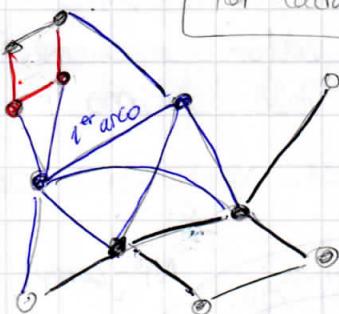
puedo usar esta
aproximación para resolver
(de manera aprox también)
el problema de decisión.

→ incluyo ambos nodos, en vez
de elegir uno solo.

Por cada par de nodos

que meto, el
conjunto óptimo
tiene al menos
1 de esos 2.

⇒ 2-aprox



Es una 2-aprox pues cada vez que mete los nodos u y v en V' , un VC óptimo debe contener $u \circ v$ (o ambos).

Al eliminar las aristas que $u \circ v$ cubren, el invarianta sigue valiendo en el E resultante.

Recordemos que: $VC \ k \Leftrightarrow$ clique de $n-k$ en \bar{G}

ej: 100 nodos (n)

clique de tam 10 → \bar{G} VC de tam 90

clique de \emptyset ← la aprox encuentra $\leq 180 / 100$

X

No puedo reducir algoritmos de opt. aproximados.

Camino Hamiltoniano

Dado $G(V, E)$, existe un circuito que pase por cada nodo exactamente una vez? Sí? No?

Problema del viajante de comercio (prob. de opt) ("vendedor viajero")

Además las aristas tienen un costo, y quiero un circuito que minimice la suma de los costos.

(Supondremos que existe camino Hamiltoniano)

Probaremos que este problema NO se deja aproximar.

Supongamos que existe una $p(n)$ -aproximación. La uso para resolver un problema de circ. Hamiltoniano en $G(V, E)$

Creo un grafo completo $G'(V, V \times V)$

con costo $C(u, v) = \begin{cases} 1 & \text{si } (u, v) \in E \\ |V|p(|V|) + 1 & \text{si } (u, v) \notin E. \end{cases}$

La idea es que si tengo una $p(n)$ -aprox cualquiera, puedo responder el problema de circ. hamiltoniano SIEMPRE en tiempo polinomial escogiendo costos correctos. (Aristas muy costosas, de manera que el $p(n)$ -aprox nunca los escoge) → que no están en G , y si en G'

Si existe un circuito en G , el costo del mejor camino en G' es $|V|$

∴ la $p(n)$ -aproximación me encontrará un camino de costo $\leq |V|p(|V|)$

Si no existe un circuito Hamiltoniano en G , entonces toda solución en G' necesita usar al menos una arista que NO está en G , la cual tiene costo $|V|p(|V|) + 1$

∴ el costo total es siempre, usando cualquier solución aprox o no, $> |V|p(|V|)$.

Ahora ejecuto solv ϕ aprox, veo si es $> \leq |V|p(|V|)$ y siempre puedo resolver prob. de camino Hamiltoniano.



Perna

Desigualdad Triangular

$$C(u, v) \leq C(u, w) + C(w, v)$$

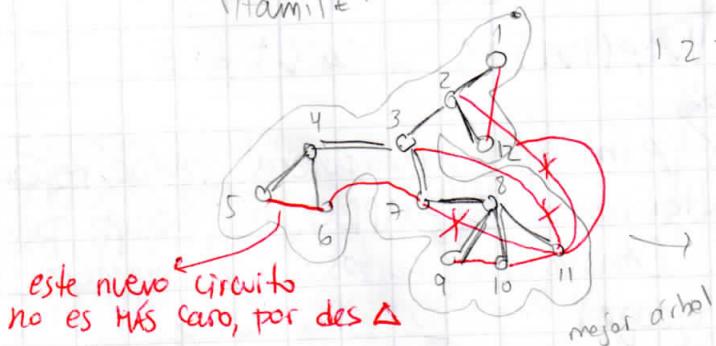
Si los costos del problema del viajante de comercio satisfacen la desigualdad triangular, con $E = V \times V$, entonces existe una **2-aprox.** (y una $\frac{3}{2}$, Christofides)

Un circuito es un camino + 1 arista.

$$C(\text{circ.}_{\text{Hamilt}}) \geq C(\text{camino Hamilt}) \geq C(\text{MST})$$

mín spanning tree

1 2 3 4 5 6 7
8 9 10 11 12 13
14 15 16 17 18 19



→ no hay problema pues existen todas las aristas.

$$C'(\text{Euler}) \leq C(\text{Euler}) = 2C(\text{MST}) \leq 2C(\text{circ.}_H.)$$

↳ pasa 2 veces por cada arista de MST

2-aprox:

- 1) Construir un MST de G
- 2) Hago un circuito Euleriano del MST.

Vertex Cover con costos

Cada nodo pesa $w(v)$. Quiero minimizar la suma de los pesos.

$$\text{Sea } x(v) = \begin{cases} 1 & \text{si elijo } v \\ 0 & \sim \end{cases}$$

Quiero encontrar $x(\cdot)$ tal que

$$\forall v \quad x(v) \geq 0$$

$$x(v) \leq 1$$

$$\forall u, v \in E, \quad x(u) + x(v) \geq 1$$

Mínimizar $\sum_{v \in V} x(v) w(v)$

Es un problema de programación lineal que se resuelve en tiempo polinomial. El óptimo lineal, C_L , es $C_L \leq C^*$

De esa solución $x(v) \in [0, 1]$, obtenemos la de VC como:

elegir v sii $x(v) \geq 0.5$.

- 1) es v en VC? si, pues $(u, v) \in E \Rightarrow x(u) + x(v) \geq 1$
 $\Rightarrow x(u) \geq 0.5$ ó $x(v) \geq 0.5$
 \Rightarrow elijo u o v .

2) qué aproximación resulta?

$$\sum_{v \in V'} w(v) = \sum_{v \in V'} y(v) w(v)$$

$$y(v) = \begin{cases} 1 & \text{si } v \in V' \\ 0 & \sim \end{cases}$$

$$\text{si } v \in V', \quad y(v) = 0 \leq x(v)$$

$$\text{si } v \notin V', \quad y(v) = 1 \leq 2 \cdot x(v) \quad x(v) \geq 0.5 \text{ porque lo elegí.}$$

∴ La suma de los pesos es

$$\sum_{v \in V'} w(v) \leq 2 \cdot C_L \leq 2 \cdot C^*$$

Auxiliar 11 - And Now For Something Completely Different

CC4102 - Diseño y Análisis de Algoritmos

Profesor: Gonzalo Navarro Auxiliar: Jorge Bahamonde

30 de Noviembre del 2015

1 Inicialización de Arreglos

Considere que tiene un arreglo V de tamaño D (en la forma de un bloque continuo de memoria), que debe soportar las siguientes operaciones:

- $\text{INIT}(V, D, v)$: inicializa el arreglo con el valor v en $1, \dots, D$.
- $\text{READ}(V, i)$: retorna el valor almacenado en la posición i del arreglo.
- $\text{WRITE}(V, i, v)$: almacena el valor v en la posición i .

Considere que cuando pide espacio de memoria al sistema, el contenido de éste puede estar sucio. Diseñe una estructura de modo que las tres operaciones mencionadas tomen tiempo constante:

1. Utilizando $O(D \log D)$ bits de espacio adicional.
2. Utilizando $O(D)$ bits de espacio adicional.
3. **Propuesto:** Utilizando $o(D)$ bits de espacio adicional.

2 Problemas NP-completos

Algunos problemas NP-completos pueden ser resueltos de forma aproximada. Sin embargo, en ocasiones una respuesta aproximada puede no ser posible de obtener, o puede simplemente no resultar útil. Luego, es interesante conocer formas de buscar soluciones exactas para estos problemas.

3 Bin packing

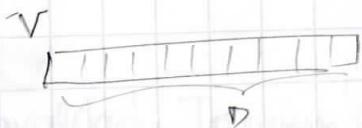
Tenemos un conjunto $I = \{1, \dots, n\}$ de items, donde cada item $i \in I$ tiene un *tamaño* $s_i \in (0, 1]$; además, tenemos un conjunto $B = \{1, \dots, n\}$ de *bins* con capacidad 1. Se pide encontrar una asignación $a : I \rightarrow B$ de modo que el número de bins utilizados (no vacíos) sea mínima. Para acortar notación, para un $J \subset I$ denotaremos $s(J) = \sum_{j \in J} s_j$. Este problema es NP-completo, pero es fácil obtener una 2-aproximación.

1. Muestre una 2-aproximación.
2. Muestre que no es posible lograr un factor de aproximación menor a $3/2$ en tiempo polinomial si $P \neq NP$. Para ello, recuerde que el problema de partir X en dos conjuntos de igual suma es NP-completo.
3. Muestre una $3/2$ -aproximación.

2015年11月30日 (A)

Aux # 11

PI



1) Usando $O(D \log D)$ bits adicionales

Ingredientes

* Un arreglo $V[1, D]$

* Un stack $S[1, D]$ (en forma de arreglo) } estamos en un
+ un puntero al tope, t . } modelo de muy bajo
nivel, donde puede haber
slots con basura

* El valor de inicialización

Init(V, D, v): # inicializar de forma "lazy"

$t \leftarrow 0$

$I \leftarrow v$

Write(V, i, v)

si $V[i]$ no está

2) Usar $f(D)$

Idea: usemos un arreglo B de D bits, tal que

$$B[i] = 1 \Leftrightarrow V[i] \text{ ha sido modificado.}$$

Claramente nos encontramos con el problema de inicializar B .

Para ello, consideremos B como un arreglo $B' [1, D]$

con $D' = \frac{D}{w} \leftarrow$ tamaño de palabra del procesador

Supongamos $w \geq \log D$

Usamos la técnica anterior para inicializar B' en 0's

$$\Rightarrow \text{esta usa } 2D' \log D' = \frac{2D}{w} \log D' \leq \frac{2D}{w} \log D \leq 2D \text{ bits}$$

Sunto a B , esto usa $3D$ bits ($\in \Theta(D)$)

• Branch & Bound



"Búsqueda exhaustiva inteligente"

1) Backtracking.

- Mirando parte de una solución puedo descartar gran parte del espacio de soluciones.

Ej: Si tengo $(x, v x_1 \vee x_1)$, ya que sé que $x_1 = F$ no lleva a nada.

Para Backtracking necesitamos un "test" que mire un subproblema

y diga:

- \exists solución
- Encontré una solución
- No sé

acotar su costo

Luego el backtracking se ve algo así:

- Dado un problema P_0
- $S \leftarrow \{P_0\}$ es el conjunto de subproblemas activos.
- Mientras S no esté vacío:
 - escoger un subproblema $P \in S$ y quitarlo de S .
 - expandir P en P_1, \dots, P_k subproblemas menores.
 - para cada P_i :
 - Si $\text{test}(P_i)$:
 - encuentra sol global \Rightarrow lo retorno
 - no encuentra solución \Rightarrow descarto P_i
 - else se agrega P_i a S .
- Se anuncia "No solución"

2) Branch & Bound \Rightarrow para optimización (s.p.g. minimización)

- mirando parte de una solución puedo acotar su costo

Aquí necesitamos una función lower-bound que acote por abajo el costo de cualquier solución completa que parte de una solución incompleta dada.

ej) TSP. Si tenemos un tour parcial que pasa por un cijo S de nodos, completarlo necesitará un camino en $(V-S)$ y arcos de $(V-S)$ a a y b (el inicio y fin del tour).

Luego Branch & Bound se ve algo así:

- Dado un problema P_0
- $\text{best} \leftarrow \infty$
- $S \leftarrow \{P_0\}$ es el conjunto de subproblemas activos
- Mientras S no esté vacío:
 - escoger un subproblema $P \in S$ y quitarlo de S .
 - expandir P en P_1, \dots, P_k soluciones parciales
 - para cada P_i :
 - siguiente

ESQUEMA DE ALGORITMOS

- Para cada P_i

* Si P_i es una sol. completa \Rightarrow actualizar best.

* Si no,

si $\text{lower-bound}(P_i) < \text{best}$.

agregar P_i a S

- Se retorna best.

\Rightarrow costo de completar \geq + arco más liviano de a a $(V-S)$

S

+ arco más liviano de b a $(V-S)$

+ pesos del mínimo subgrafo

que conecte todos los nodos de $(V-S)$

MST

2015年12月1日 (X)

ESQUEMA DE APROXIMACIÓN

($1+\epsilon$)

PROBLEMA DE LA MOCHILA

NP-COMPLETO, se deja aproximar (KNAPSACK)

subset-sum

Versión de decisión: $x_1, x_2, \dots, x_n > 0$ bits para repr t
 tope t. $|input| = \Theta(n \cdot \log t)$ puede ser 32 bits
 Se puede encontrar un conjunto que suma exactamente t.

Versión de opt: t es maximo

Forma Greedy: recibo elemento y veo qué pasa si lo meto o no a la bolsa.

Nota: $\Theta(n \cdot t)$ prog. dinámica, pres t es gigantesco,
 ↳ $2^{32} ?!$
 NO es polinomial en el
 input $\Theta(n \log t) \dots$ es exponencial!

Veamos algoritmo exacto:

Exacto:

$L \leftarrow \langle \rangle$ (lista ordenada crecientemente siempre)

for $i \leftarrow 1$ to n (lo incluyo o no?)

a cada etapa de L , sumas $x_i \rightarrow L \leftarrow \text{merge}(L, L + x_i)$ (unión de todos los subconjuntos posibles)
 truncar (L, t) (eliminar subconjuntos que se pasan) de t
 return $\max L$ (máx t)

$$x = 1, 3, 2$$

$$L \leftarrow \langle \rangle \quad L+1 = \langle 1 \rangle$$

$$L \leftarrow \langle 0, 1 \rangle \quad L+3 = \langle 3, 4 \rangle$$

$$L \leftarrow \langle 0, 1, 3, 4 \rangle \quad L+2 = \langle 2, 3, 5, 6 \rangle$$

$$L \leftarrow \langle 0, 1, 2, 3, 4, 5, 6 \rangle$$

$$= \langle 0, 1, 3, 4 \rangle \quad \text{agrego}$$

$$\langle 2, 3, 5, 6 \rangle \quad \text{no agrego}$$

en el merge se eliminan repetidas

Lo anterior era
Fuerza Bruta ...

$$2^2 \cdot 2^{n-1} + 2^{n-2}$$



CON

¿Cuánto tiempo requiere? $\approx \Theta(2^n)$ pueden estar todos los subconjuntos posibles en L
INACEPTABLE.

Vamos a aproximar. La idea es hacer que L no se agrande mucho

• Versión aproximada: Dado un elemento $z \in L$, tendremos una L' donde garantizamos que $\forall y \in L$ que representa a z está en L' . Y represente a z si $\left| \frac{z}{1+\delta} \leq y \leq z \right|$

z es representado por alguien en foco MENOR (no podes usar mayor porque me podria pasar de t)

Aprox (ϵ)

$L \leftarrow \langle \rangle$

for $i \leftarrow 1$ to n

$L \leftarrow \text{Merge}(L, L + x_i)$

$\text{truncar}(L, t)$

$\text{filtrar}(L, \delta) \leftarrow$ borro los otros representados por otras
return max L

Filtrar (L, δ)

$L' \leftarrow \langle L[1] \rangle$ tomo el min; por definición nadie lo pude representar.

$\text{last} \leftarrow L[1]$

for $i \leftarrow 2$ to $|L|$

if $L[i] > \text{last} \cdot (1+\delta)$

$L' \leftarrow L' \text{concat } L[i]$

$\text{last} \leftarrow L[i]$

return L'

Este error relacionado a $1+\delta$ se va acumulando?

$$5 \geq 4 \geq 3 \geq \frac{5}{(1+\delta)}$$

Este error relacionado a $(1+\delta)$ se va acumulando.

Después de filtrar tenemos:

$$L[1] \geq 1$$

$$L[i] > (1+\delta) \cdot L[i-1]$$

$$\text{si } l = |L|, \quad L[l] \geq (1+\delta)^{l-1}$$

pero solo trabajamos mientras $L(l) \leq t$ (al truncar)

$$\log(1). (1+\delta)^{l-1} \leq t \quad (\log(1+\delta) \cdot (l-1) \leq \log t)$$

$$l \leq \frac{\log t}{\log(1+\delta)} + 1 \quad \begin{array}{l} \text{tenemos esta sup.} \\ \text{para el largo!} \end{array}$$

Entonces en cada iteración del for pago $\Theta\left(\frac{1 + \log t}{\log(1+\delta)}\right)$

$$\cancel{\Theta(n^2)} \rightarrow \Theta\left(\frac{n \log t}{\log(1+\delta)}\right) \quad \begin{array}{l} \text{y } n \log t \text{ es} \\ \text{el tamaño del input.} \end{array}$$

Pareciera ser lineal en el tamaño del input, pero debemos estudiar bien qué sorpresas salen del $(1+\delta)$.
Aunque sean números enteros, hay que dejar que $\delta > 0 \in \mathbb{R}$ si no explota.

¿Cuál es la relación entre ϵ y δ ? + peor caso

$$\frac{z}{1+\delta} \leq y \leq z \quad \begin{array}{l} \downarrow \\ y \text{ pasa a 2ª etapa. Se reemplaza luego con } x \end{array}$$

$$\frac{z}{(1+\delta)^2} \leq \frac{y}{1+\delta} \leq x \leq y \quad \begin{array}{l} \downarrow \\ \text{pasa a 3ª etapa y se reemplaza por } w \end{array}$$

$$\frac{z}{(1+\delta)^3} \leq \frac{x}{1+\delta} \leq w \leq x \quad ; \text{después de } n \text{ iteraciones, entrego } z^* \text{ que} \\ \text{representa a } z \text{ óptimo}$$

$$\frac{z}{(1+\delta)^n} \leq z^* \leq z$$

x_1 , solución en el máx de todos, s.t.

Lo peor es que el primer élt sea $x_1 = t$, luego viene $x_2 = t/(1+\delta)$ y el alg. elimina x_1 , luego llega $x_3 = \frac{t}{(1+\delta)^2}$, se elimina x_2 y se reemplaza por x_3 , y así.

\Rightarrow Luego digo $\underbrace{\frac{z}{(1+\delta)^n}}_{1+\epsilon \text{ aproximado}} \leq z^* \leq z$

$$\begin{aligned} 1 + \delta &= (1 + \epsilon)^n \\ \Rightarrow (1 + \delta)^{1/n} &= 1 + \epsilon \\ \Rightarrow | \delta &= (1 + \epsilon)^{1/n} - 1 | \quad \text{Carmen demuestra que } \delta = \epsilon/2n \text{ funciona} \end{aligned}$$

Si originalmente había mucha distancia entre los elementos, querás con filtrar no elimino ninguno de la lista, pero este problema era fácil per sé! porque al sumar los costos se llega rápido a t . El problema es difícil cuando los costos entre éltos son muy parecidos.

Reemplazando δ , el costo del apróx es

$$\Theta\left(\frac{n^2 \log t}{\log(1+\epsilon)}\right) \quad \text{usaremos } \frac{\epsilon}{1+\epsilon} \leq \ln(1+\epsilon) \leq \epsilon$$

$$\Rightarrow \Theta\left(\frac{(1+\epsilon) n^2 \log t}{\epsilon}\right) = \boxed{\Theta\left(\frac{n^2 \log t}{\epsilon}\right)}$$

está en función de $\frac{1}{\epsilon}$, que era b. que queríamos

El problema es polinomial en n , en $\log t$ y en $\frac{1}{\epsilon}$.

Fin

NOT

Auxiliar 12 - Problemas para el Control

CC4102 - Diseño y Análisis de Algoritmos

Profesor: Gonzalo Navarro Auxiliar: Jorge Bahamonde

3 de Diciembre del 2015

1 Análisis Amortizado: Árboles α -balanceados

Un *árbol α -balanceado*, para $1/2 < \alpha < 1$, es un árbol binario de búsqueda donde todo subárbol $T = (\text{root}, T_l, T_r)$, cumple $|T_l| \leq \alpha|T|$ y $|T_r| \leq \alpha|T|$. Las operaciones para buscar y mantener un árbol α -balanceado son las mismas que para un árbol binario de búsqueda, excepto que luego de insertar o borrar un nodo, se busca el nodo más alto en el camino del punto de inserción/borrado hacia la raíz, que no esté α -balanceado, y se lo reconstruye como árbol perfectamente balanceado (el costo es proporcional al tamaño del subárbol que se reconstruye).

1. Muestre que la búsqueda en un árbol α -balanceado cuesta $O(\log n)$, y que lo mismo ocurre con las inserciones y borrados, si no consideramos las reconstrucciones. ¿Qué constante obtiene multiplicando el $\log n$?
2. Muestre que el costo amortizado de las inserciones y borrados, ahora considerando las reconstrucciones, es también $O(\log n)$. Para ello, considere la función potencial

$$\Phi(T) = \frac{1}{2\alpha - 1} \sum_{T' \in T} \max\{|T'_l| - |T'_r| - 1, 0\}$$

donde $T' \in T$ significa que T' es un subárbol de T . **Propuesto:** encontrar la constante que multiplica a este $\log n$, y recomendar un óptimo.

2 Dominios Discretos: Consultas de Rango

Describa un algoritmo que, dados n enteros en $[0, \dots, k]$, preprocese su entrada y responda cuántos de estos enteros se encuentran en el rango $[a, \dots, b]$ en tiempo constante. Su algoritmo debería tomar tiempo $\Theta(n + k)$ en el preprocesamiento.

3 Algoritmos Online: Tareas y Procesadores

Tenemos m procesadores idénticos. Como entrada recibimos una lista t_1, t_2, \dots, t_n de tareas con tiempos de procesamiento $p_1, p_2, \dots, p_n > 0$, respectivamente. Las tareas son recibidas secuencialmente, y sólo cuando una tarea llega conocemos su tiempo de procesamiento. Cada tarea debe ser asignada a una máquina inmediatamente, y la decisión no puede ser cambiada. La *carga* de un procesador es la suma de los tiempos de procesamiento de todas las tareas que le son asignadas. El costo de un algoritmo que resuelve el problema de asignación es la máxima carga entre sus procesadores.

Considere el siguiente algoritmo para el problema anterior. Cada tarea es asignada al procesador con menor carga (en caso de empate, se elige cualquiera).

1. Demuestre que este algoritmo es $(2 - \frac{1}{m})$ -competitivo.
2. Demuestre que esta cota es óptima para el algoritmo.

4 Algoritmos Probabilísticos: Asociatividad

Se nos entrega un conjunto X con n elementos y una operación binaria \circ . Se pide determinar si \circ es asociativa en X : es decir, si para todos x, y, z en X , se cumple $(x \circ y) \circ z = x \circ (y \circ z)$. Diseñe un algoritmo (probabilístico) que resuelva este problema en $O(n^2)$ y con buena probabilidad.

AUX #12

2015年12月3日(木)

P2

n enteros $\in \{0, \dots, k\}$

$\rightarrow n(a, b) \rightarrow \# \text{ en t. cte.}$

Tiempo $\Theta(n+k)$ de proproc.

Sol: Si constuyo la tabla de conteos y luego hago sumas parciales en una tabla T.

$$T[i] = \sum_{j=0}^i x_j \quad \# \text{ elementos } \leq i$$

$\hookrightarrow \# \text{ de apariciones de } j, \text{ suma de frecuencias acumuladas}$

Dado $A[1, n]$

$\Theta(k) \left\{ \begin{array}{l} \text{for } i = 0 \dots k \\ C[i] \leftarrow 0 \end{array} \right. \quad A[i] = j \rightarrow C[j]++ \right.$

$\Theta(n) \left\{ \begin{array}{l} \text{for } i = 1 \dots n : \\ C[A[i]]++ \end{array} \right. // \text{Aquí } C[i] = x_i, \text{ uso } \underline{\text{Counting Sort.}}$

$\Theta(k) \left\{ \begin{array}{l} \text{for } i = 0 \dots k \\ C[i] = C[i] + C[i-1] \end{array} \right. // \text{considerando } C[-1] = 0. \rightarrow$

$$\text{Luego } n(a, b) = C[b] - C[a-1]$$

Esta respuesta es la que toma tiempo cte.

P3

m procs.

lista de tareas t_1, \dots, t_n

con tiempos de procesamiento $p_1, \dots, p_n > 0$

• Algoritmo online conoce p_i solo cuando t_i llega:

• Carga $\sum_{t_i \in \text{proc}} p_i$

Algoritmo: enviar la tarea al proc. con menor carga.

1.- Demuéstrese que es $(2 - \frac{1}{m})$ -competitivo.

- "Hint". Si s es el procesador con mayor carga, ¿qué ocurre con el tiempo de ocio de los demás?
- Si no soy s , mi tiempo de ocio es \leq a la última tarea asignada a s .
→ pues de lo contrario esa tarea me habría sido asignada
 $\Rightarrow t_{\text{ocio}} \leq \max_{1 \leq i \leq n} p_i$

Por otro lado

$$m \cdot c(I) = \sum \text{cargas} + \sum t_{\text{ocio}}$$

↑ costo total del alg.

$$\Rightarrow m \cdot c(I) \stackrel{(a)}{\leq} \sum p_i + (m-1) \max_{1 \leq i \leq n} p_i$$

Por otro lado,

$$\text{OPT}(I) \geq \frac{1}{m} \sum_{i=1}^n p_i \quad \begin{array}{l} \text{(no puedo demorar menos que un caso ideal} \\ \text{con } t_{\text{ocio}} = 0 \end{array}$$

Además, $\text{OPT}(I) > \max_{1 \leq i \leq n} p_i$ (no puedo demorar menos que la tarea + larga)

$$\Rightarrow C(I) \leq \text{OPT}(I) + \left(\frac{m-1}{m}\right) \cdot \text{OPT}(I) = \left(2 - \frac{1}{m}\right) \text{OPT}(I)$$

⇒ el algoritmo es $(2 - \frac{1}{m})$ -competitivo.

2. Demostrar que la cota es óptima para el algoritmo.
 O sea, que \exists un caso I^* en que $C(I^*) = \left(2 - \frac{1}{m}\right) OPT(I^*)$

Consideraremos $m \cdot (m-1)$ tareas de costo 1 y luego otra de costo por determinar.
 El algoritmo online va a dejar los m procs con carga $(m-1)$
 Ahora llega una tarea de costo x
 \Rightarrow costo = $(m-1+x)$ (uno de ellos tendrá carga m+1+x)

¿Qué haría OPT? OPT conoce todas las tareas.

• Por ejemplo, sabiendo que x puede ser grande, reservar un proc para x .

\Rightarrow uso $(m-1)$ procs para las de tamaño 1 $\rightarrow m-1$ procs con carga m

\Rightarrow costo = $\max(m-1, x)$

$$\Rightarrow \frac{C(I)}{OPT(I)} = \frac{m-1+x}{\max(m-1, x)} \stackrel{\text{buscando } 2 - \frac{1}{m}}{\Rightarrow} x = m \Rightarrow \frac{2m-1}{m} = 2 - \frac{1}{m}$$

P4

$X, |X| = n, \circ, x \circ y \dots$
 $\forall x, y, z \in X, (x \circ y) \circ z = x \circ (y \circ z)$

- Naive: chegar todos las tuplas $(x, y, z) \Rightarrow$ tiempo $\Theta(n^2)$
- Como con otros casos (mult. de matrices, etc) podría buscar un festigo (x^*, y^*, z^*) similar. El problema es que los festigos pueden no ser densos. Existe una familia de ops binaria tq el número de festigos es cte.

Escogemos conjuntos de tuplas como testigos

Sea $P = P(X)$

Cada $R \in P$ puede verse como un vector binario de largo n , donde el i -ésimo bit denota si el i -ésimo elemento de X está o no en P
 $(R = \sum_{i \in X} r_i \cdot i)$

Definimos: $R + S = \sum_{i \in X} (r_i \mid s_i) \cdot i$

$$R \circ S = \sum_{i,j \in X} (r_i \mid s_j) (i \circ j) \in P$$

$$\alpha R = \sum_i (\alpha r_i) i \text{ para } \alpha \text{ cte.}$$

• Algoritmo:

- elegir R, S, T de P (uniforme, indep)
- si $(R \circ S) \circ T \neq R \circ (S \circ T)$ $\Rightarrow \Theta(n^2)$
 - responder no \leftarrow seguro
 - else responder sí \leftarrow Puede equivocarse.

• \circ es asociativa en $X \Leftrightarrow \circ$ es asociativa en P (fácil de mostrar (?)

Prop: Si \circ NO es asociativa, al menos $\frac{1}{8^n}$ de las tuplas R, S, T son testigos.

$$\text{O sea, } P[(R \circ S) \circ T = R \circ (S \circ T)] \leq \frac{7}{8}$$

• Vamos a particionar P^3 en grupos de 8 , tq cada grupo tiene al menos 1 testigo.

Si \circ no es asociativa, $\exists (i^*, j^*, k^*)$ tq $(i^* \circ j^*) \circ k^* \neq i^* \circ (j^* \circ k^*)$

• Sea R_0 tq $i^* \notin R_0$, S_0 tq $j^* \notin S_0$, T_0 tq $k^* \notin T_0$

Luego llamaremos

$$R_1 = R_0 \cup \{i\}$$
$$S_1 = S_0 \cup \{j\}$$
$$T_1 = T_0 \cup \{k\}$$

Sea $f(R, S, T) = \sum_{\substack{i \in R \\ j \in S \\ k \in T}} f(i, j, k)$ con $f(i, j, k)$

Auxiliar 13 - Auxiliar Sandwich

CC4102 - Diseño y Análisis de Algoritmos

Profesor: Gonzalo Navarro Auxiliar: Jorge Bahamonde

7 de Diciembre del 2015

1 Árboles α -balanceados

Un *árbol α -balanceado*, para $1/2 < \alpha < 1$, es un árbol binario de búsqueda donde todo subárbol $T = (\text{root}, T_l, T_r)$, cumple $|T_l| \leq \alpha|T|$ y $|T_r| \leq \alpha|T|$. Las operaciones para buscar y mantener un árbol α -balanceado son las mismas que para un árbol binario de búsqueda, excepto que luego de insertar o borrar un nodo, se busca el nodo más alto en el camino del punto de inserción/borrado hacia la raíz, que no esté α -balanceado, y se lo reconstruye como árbol perfectamente balanceado (el costo es proporcional al tamaño del subárbol que se reconstruye).

1. Muestre que la búsqueda en un árbol α -balanceado cuesta $O(\log n)$, y que lo mismo ocurre con las inserciones y borrados, si no consideramos las reconstrucciones. ¿Qué constante obtiene multiplicando el $\log n$?
2. Muestre que el costo amortizado de las inserciones y borrados, ahora considerando las reconstrucciones, es también $O(\log n)$. Para ello, considere la función potencial

$$\Phi(T) = \frac{1}{2\alpha - 1} \sum_{T' \in T} \max\{|T'_l| - |T'_r| - 1, 0\}$$

donde $T' \in T$ significa que T' es un subárbol de T .

2 Dominios... ¿Discretos?

Se desea ordenar n puntos que pertenecen al círculo unitario según su distancia al origen, de menor a mayor. Diseñe un algoritmo que en promedio tome tiempo $O(n)$, suponiendo que los puntos se distribuyen de manera uniforme en este espacio.

3 Coloreo de Elementos

Sea S un conjunto de n elementos. Se tienen k conjuntos S_1, \dots, S_k de r elementos cada uno. Los conjuntos no necesariamente son disjuntos, y se cumple que $k/2^r \leq 1/4$. Se desea colorear los elementos de rojo o azul, de modo que ningún S_i quede monocromático (todos los puntos rojos o todos azules).

1. Diseñe un algoritmo de tipo MonteCarlo que obtenga un colooreo válido con probabilidad $1/2$ y analícelo.
2. Mejore su algoritmo para obtener un colooreo válido con probabilidad de fallar a lo más $1/2^t$, para cualquier t dado, y analícelo.
3. Convierta el algoritmo en uno de tipo las Vegas y analice su costo esperado.

4 Uniendo (casi) todo: Heavy Hitters

Considere un *stream* A de elementos a_1, \dots, a_m de m objetos, donde cada $a_i \in 1, \dots, n$. Esto significa que el tamaño de cada elemento es cercano a $\log n$, y que contar el número de objetos ya vistos requiere espacio $\log m$ (en el caso exacto). En este caso, n puede ser *muy* grande.

Denotaremos $f_j = |\{a_i \in A \mid a_i = j\}|$, es decir, el número de elementos en el stream que toman el valor j . También denotaremos $F_1 = \sum_j f_j$, el número de elementos ya vistos, y $F_0 = \sum_j f_j^0$ el número de elementos distintos.

Queremos resolver el problema de encontrar aquellos elementos con una frecuencia mayor a ϕ , es decir, que $f_j > \phi \cdot m$ (análogamente, el problema puede ser planteado como encontrar los k elementos más frecuentes).

Analizaremos una versión aproximada del problema, de modo que se permite que elementos con una frecuencia entre $\phi - \epsilon$ y ϕ también sean retornados.

Resolveremos este problema con un algoritmo online y probabilístico que entrega una solución aproximada. El análisis requerirá pensar, además, en las representaciones de los elementos.

2015年12月7日(月)

Avx # 13

[PI]

Árboles α -balanceados

$$\frac{1}{2} < \alpha < 1$$

$$\begin{array}{c} T \\ / \quad \backslash \\ T_L \quad T_R \end{array} \Rightarrow |T_L| \leq \alpha |T|$$
$$|T_R| \leq \alpha |T|$$

- Al insertar un nodo, debes volver por el camino que bajé y verificar que cada subárbol sea α -balanceado. Si ese subárbol no está α -balanceado \Rightarrow reconstruye subárbol para que sea perfectamente balanceado.
- Costo de rebalancear T es $\Theta(|T|)$

1) Al bajar por el árbol, en cada descenso descarto al menos $(1-\alpha)$ de los elementos. (Suponemos que en cada descenso, en el peor caso bajo por el subárbol más grande, el que tiene α de los). Esto sucede hasta quedarnos con 1 elemento.

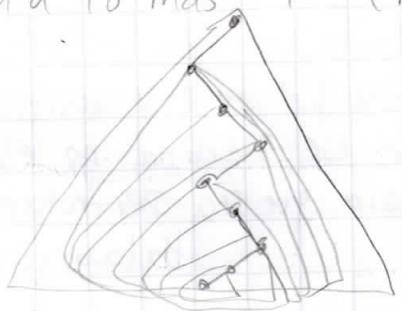
$$\Rightarrow \text{cuando } \alpha^t n \leq 1 \text{ (después de } t \text{ pasos)}$$

$$\Rightarrow \frac{1}{\alpha^t} \sim n$$

$$\Rightarrow \log n \sim t \log \left(\frac{1}{\alpha}\right)$$

$$\Rightarrow t \sim \frac{\log n}{\log \left(\frac{1}{\alpha}\right)} = \log \left(\frac{1}{\alpha}\right)^{-1} \dots$$

- 2) $\phi(T) = \frac{1}{2\alpha-1} \sum_{T' \in T} \max(||T_L|| - ||T_R|| - 1, 0)$ representa cuán desbalanceada está la estructura, en cada punto uno de los sumandos de ϕ se va a ver afectado. La diferencia entre $|T_L|$ y $|T_R|$ va a cambiar en lo más 1 ($\max(||T_L|| - ||T_R|| - 1, 0)$)



Como inserto en $\Theta(\log n)$ subárboles (largo del camino) el $\Delta\phi$ total es a lo más $\Theta(\log n)$
 \Rightarrow puede cargarse este costo al descenso en el árbol.

Para la reconstrucción, si rebalanceo T^* ,
 $\rightarrow \phi_f(T^*) = 0$ (para un árbol perfectamente balanceado, $\phi = 0$)
 Por otro lado, antes de un rebalanceo,
 $\phi_i = \Theta(|T^*|)$? Pg?

\rightarrow Sabemos que si T^* fue reconstruido, (S.P.g.)
 $|T_L| > \alpha |T| + 1 \Rightarrow |T_R| \leq (1-\alpha)|T|$

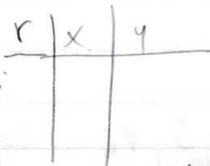
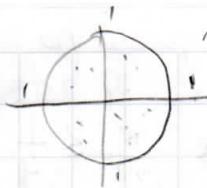
$$\Rightarrow |T_L| - |T_R| \geq \alpha |T| - (1-\alpha)|T| = (2\alpha-1)|T|$$

$$\Rightarrow \frac{1}{2\alpha-1} \max(|T_L| - |T_R| - 1, 0) \geq |T| - 1.$$

$$\Rightarrow \phi_i(T) \geq |T| - 1 \Rightarrow |\Delta\phi| = -\Theta(|T|)$$

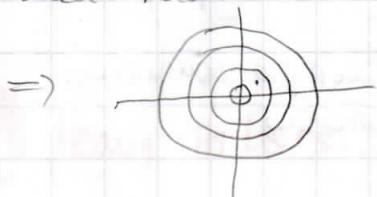
Luego esto puede pagar la reconstrucción \Rightarrow reconstrucción es "gratis"

P2



A pesar de ser un dominio continuo, lo dividimos en "cajones" y trabajamos de manera discreta.

Idea: hacer buckets en forma de anillos concéntricos.



Luego, BucketSort.

Idealmente, en cada bucket queda 1 elemento (construyo tantos buckets como sea necesario para ello)

\Rightarrow Usaremos n buckets con la misma área = $\frac{\pi}{n}$ (para aprovechar distribución uniforme)

\Rightarrow La primera zona, la central, es un círculo de radio r_1 .

$$\Rightarrow A_1 = \pi r_1^2 = \frac{\pi}{n} \Rightarrow r_1 = \frac{1}{\sqrt{n}}$$



$$\Rightarrow A_j = \pi r_j^2 - \pi r_{j-1}^2$$

Las demás áreas son anillos

$$\text{para } A_2 = \pi r_2^2 - \frac{\pi}{n} = \frac{\pi}{n} \Rightarrow r_2 = \sqrt{\frac{2}{n}}$$

$$\text{De hecho, } r_j = \sqrt{\frac{j}{n}}$$

Luego el algoritmo es simplemente usar bucket sort con estos buckets $\Rightarrow O(n)$ en promedio (en promedio habrá un punto por bucket).

P3

$$|S| = h$$

k ejtos S_1, \dots, S_k , $|S_i| = r$; se cumple $\frac{k}{2^r} \leq \frac{1}{4}$

$S_i \subseteq S$ $\forall i$, los S_i pueden no ser disjuntos

1.- Pintaremos los elementos al azar (en forma indep y uniforme)

Esto toma $\Theta(n + kr)$

→ verificar que se cumpla condición.

Para un S_i sea $X_i = \begin{cases} 1 & \text{si } S_i \text{ es monocromático} \\ 0 & \text{no} \end{cases}$

Queremos (para una sol correcta), $\sum X_i = 0$.

$$\text{Para un } S_i \quad E(X_i) = \sum_{\text{rojo, azul}} \left(\frac{1}{2}\right)^r = \frac{1}{2^{r-1}}$$

$$\Rightarrow E(\sum X_i) = \sum E(X_i) = \frac{k}{2^{r-1}}$$

Ahora, por la desigualdad de Markov (sirve para pasar de $E(\cdot)$ a \Pr con una cota)

$$\Rightarrow E(\sum X_i \geq 1) = \frac{k}{2^{r-1}} \xrightarrow{\text{anunciado, }} \frac{k}{2^r} \leq \frac{1}{4}$$

$$\Rightarrow \Pr(\sum X_i \geq 1) \leq \frac{k}{2^r} \leq \boxed{\frac{1}{2}}$$

2.- OK, repito t veces verificando. Si alguna vez obtengo un coloreo válido, lo retorno

$$\Rightarrow \Pr[\text{fallar } t \text{ veces}] = \prod_{i=1}^t [1 - \Pr[\text{fallar 1 vez}]] = \left(\frac{1}{2}\right)^t$$

tiempo $\Theta(t(n + kr))$

3.- Repetir por siempre ($t \rightarrow \infty$) \leftarrow fallar $i-1$ veces \leftarrow achuntarle

$$E(t) = \sum i \cdot P(i) = \sum_{i=1}^{\infty} i \cdot p^{i-1} (1-p)$$

$$= \sum_{i=1}^{\infty} \frac{i}{2^i}$$

para resolver usar

$$\frac{d}{da} \left(\sum a^x \right)$$

74 Heavy hitters.

Stream A = $a_1 \dots a_m$, $a_i \in \{1 \dots n\}$

Encuentrar los i tq $f_i > \phi \cdot m$, para ϕ dado.

Me gustaría tener un contador para ca $a_i \in \{1 \dots n\}$, pero n puede ser gigante! Uso "buckets" contadores donde a_i 's caen en el mismo bucket \rightarrow hash con colisiones. Usaremos varios hash

\Rightarrow Algoritmo Count-Min

- Usamos t funciones de hash $h_i: n \rightarrow k$, con $k < n$.
- También tenemos una tabla C:

h_1	C_{11}	C_{12}	\vdots	$\boxed{10}$	\vdots	C_{1k}
h_2	C_{21}	C_{22}	\vdots	\vdots	\vdots	C_{2k}
\vdots						
h_t		\vdots	\vdots	\vdots		C_{tk}

Count-Min:

$$C_{i,j} \leftarrow 0 \quad \forall i, j$$

cuando llega un a_i ,
for $j = 1 \dots t$

$$C_{j,h_j(a_i)} ++$$

$$\Rightarrow \hat{f}_q = \min_{j=1 \dots t} C_{i,h_j(q)}$$

$$C_{j,h_j(q)} \geq f_q$$

Claramente $\hat{f}_q \geq f_q$ } queremos acotar \hat{f}_q en un rango
 Ahora, $f_q < \hat{f}_q + W$ } aceptable

Definimos $Y_{i,j} = \text{overcount en } h_i \text{ debido a } j$.

$$Y_{i,j} = \begin{cases} f_j & \text{con prob } 1/k \\ 0 & \text{resto} \end{cases} \text{ pues } h \text{ es buena fn. de hash.}$$

$\Rightarrow E[Y_{i,j}] = f_j/k$ lo que hace fallar 1 elemento al contar de q

$$X_i = \sum_{j \neq q} Y_{i,j} \Rightarrow E[X_i] = \sum_{j \neq q} \frac{f_j}{k} = \frac{F_1}{k} \leftarrow \text{cuanto falla la función } h_i \text{ en total.}$$

(*) Luego $P[X_i \geq \epsilon F_1] \leq \frac{1}{2}$ (Markov)

$$\begin{aligned} P[\hat{f}_q - f_q \geq \epsilon F_1] &= P[\min_i X_i \geq \epsilon F_1] = P[\bigcap_{i=1}^t (X_i \geq \epsilon F_1)] \\ &= \prod_{i=1}^t P[X_i \geq \epsilon F_1] \leftarrow \text{aqui ganamos por usar muchas fn. de hash } \neq's \\ &\stackrel{(*)}{=} \frac{1}{2}^t \end{aligned}$$

Notación $\frac{\epsilon F_1}{2}$. Para que fallo el conjunto deben fallar todas por separado.

$$\text{Definimos } \delta \text{ } t = \log \frac{1}{\delta} \Rightarrow P[\text{fallar}] = \frac{1}{2} \log \frac{1}{\delta} = \frac{1}{2} \log \frac{1}{\delta} = \delta$$

$$\Rightarrow f_q \leq \hat{f}_q \leq \hat{f}_q + \epsilon F_1$$

con $P \geq 1 - \delta$ al menos.

ϵ relacionado con k
 δ relacionado con t .

Falta análisis de espacio