

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 $u \in L$ que representa a z está en L' . Y represente a z si $|z - u| \leq \epsilon$

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

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

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

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

Filtrar (L, ϵ)

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

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

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

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

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

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

return L'

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

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

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

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 $(R \circ S) \circ T \neq R \circ (S \circ T)$ } $\Theta(n^2)$
 - 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 $f(i, j, k)$

2015年12月7日(月)

Avx # 13

[PI]

Arboles 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 ellos) Esto sucede hasta quedarnos con 1 elemento.

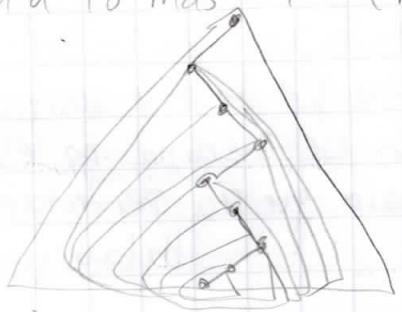
$$\Rightarrow \text{cuando } \alpha^t n \leq 1 \text{ (después de } t \text{ pasos)}$$

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

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

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

- 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.)
 $|T_L| > \alpha |T| + 1 \Rightarrow |T_R| \leq (1-\alpha)|T|$

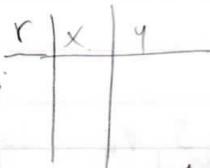
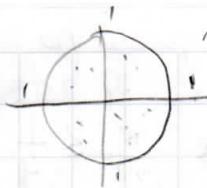
$$\Rightarrow |T_L| - |T_R| \geq \alpha |T| - (1-\alpha)|T| = (2\alpha-1)|T|$$

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

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

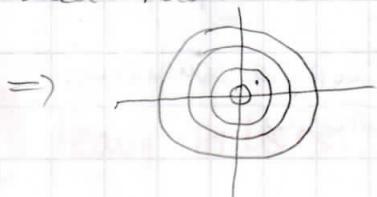
Luego esto puede pagar la reconstrucción \Rightarrow reconstrucción es "gratis"

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 $E(\cdot)$

$$E(X \geq a) \geq P(X \geq a) \quad a \text{ Pr con una cota}$$

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

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

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

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

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 $j = 1 \dots t$

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

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

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

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