

Teorema Maestro:

recurrencia $T(n) = Kn + P T\left(\frac{n}{q}\right)$ es:

- $O(n \log^q n)$ si $P > q$
- $O(n \log n)$ si $P = q$.
- $O(n)$ si $P < q$

Sea $f = *g$, con *

$$\begin{aligned} O: & f \geq g \\ \Omega: & f \geq g \\ o: & f \geq g \\ w: & f \geq g \\ \Theta: & f \approx g \end{aligned}$$
Aproximación de Stirling:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$\ln(n!) = n \ln n - n + O(\ln n)$$

$$\sqrt{2\pi} n^{n/2} e^{-n} \leq n! \leq e^n n^{n/2} e^{-n}$$

Encontrar algoritmo $\rightarrow O(f(n))$ "óptimo"
demonstración matemática $\rightarrow \Omega(f(n))$ "ajustado"
 $O(f(n)) \wedge \Omega(f(n)) \Rightarrow \Theta(f(n))$ "determinado"

Cotas inferiores1) Estrategia del Adversario

- ↳ capacidad de cómputo "infinita"
- ↳ construye el input
- ↳ debe ser constante

↳ tiene como objetivo
elijir el peor caso
posible!

estado: estado f
costo de llegar es la cota inferior

Modelos:

- grafos (+ pesos)
- tabla de acciones
- comparaciones cruciales + comparaciones no cruciales

"Encontrar el mejor peor caso"2) Teoría de la información

- ↳ cantidad mínima de bits que caracterizan un objeto
- ↳ se aplica casi exclusivamente a modelos por comparaciones
- ↳ podría considerarse potencialmente como un algoritmo de compresión

Algoritmo de compresión de Huffman:

- 1) crear bosque con n elementos (árboles), con 1 solo elemento (nodo) de peso $w = p_i$

2) tomar los árboles de menor peso, T_1 y T_2 , sacar w_1 y w_2 estos últimos, y colocar de hijo izquierdo y derecho el un nuevo nodo de peso $w_1 + w_2$

3) Sacar T_1 y T_2 del bosque, agregar el nuevo árbol

4) Volvar a (2) hasta que quede 1 sólo árbol.

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

donde H es bits utilizados en promedio para codificar un elemento de U

H es la entropía del conjunto de probabilidades

$$\text{Si } p_i = \frac{1}{|U|}$$

$$\Rightarrow H = \log_2 |U|$$

Lógica:

- 1) encontrar número de posibles permutaciones de inputs = U
- 2) encontrar número de bits que permiten definir al input probabilidad de encontrar cada uno de aquellos inputs = p_i
- 3) Calcular la entropía a partir de $p_i = H = \sum p_i \log_2 \left(\frac{1}{p_i} \right)$
- 4) Luego por el algoritmo de compresión de:
 - Huffman: $H \leq L \leq H+1$,
 - Hu-Tucker: $H \leq L \leq H+2$, inferior con $L = \sum p_i l_i$, tenemos una cota para determinar la cantidad mínima de comparaciones que determinan a un input en particular

3) Reducciones

- ↳ el problema conocido se reduce al problema del conocido
- ↳ similar a la teoría de NP-Complejidad
- ↳ válido para cualquier cota, ya sea inferior, promedio, etc...

Sea Q un problema conocido,

$$Q = \Omega(g(n))$$

Sea P un problema desconocido.

Se quiere establecer una cota inferior de P

Complejidad problemas

- encontrar elemento en arreglo desordenado: n accesos
- encontrar elemento en arreglo ordenado: $\lceil \log_2 n \rceil$ accesos o comparaciones
- encontrar máximo en un arreglo: $n-1$ comparaciones
- encontrar min y max: $\lceil \frac{3n}{2} \rceil - 2$ comparaciones
- max y 2do max: $n + \lceil \log_2 n \rceil - 2$ comparaciones
- mediana: entre $(2+2^{-50})n$ y $2,95n$ comparaciones
(vimos $\Omega(\sqrt{\frac{n}{2}})$)
- ordenar arreglo: $n \log_2 n - O(n)$ comparaciones
- mergear 2 listas de largo $m < n$: $O(m \log \frac{n}{m})$ y $2n - O(\log n)$ si $m=n$
- Buscar en un arreglo con entropía de probabilidades H : $H+2$ comparaciones
- cápsula convexa: $O(n \log n)$ operaciones y comparaciones sobre reales
- 3SUM ($a+b+c=0$): $O(n^2 / (\log n / \log \log n)^{2/3})$, se sospecha $\Omega(n^{2-\alpha})$
- 3 puntos colineales: $\geq 3SUM - Hard$

Memoria externa o secundaria

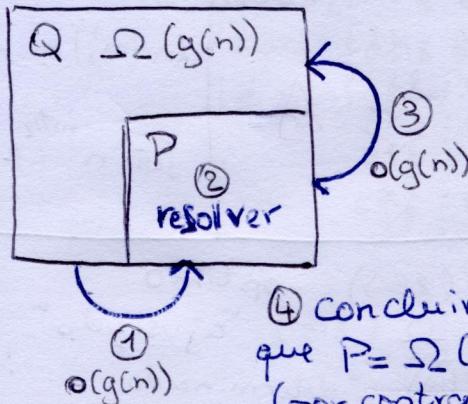
B : tamaño del bloque dentro del disco

M : tamaño memoria principal (RAM)

N : tamaño del input

$m = \frac{M}{N}$: tamaño de bloques de memoria principal

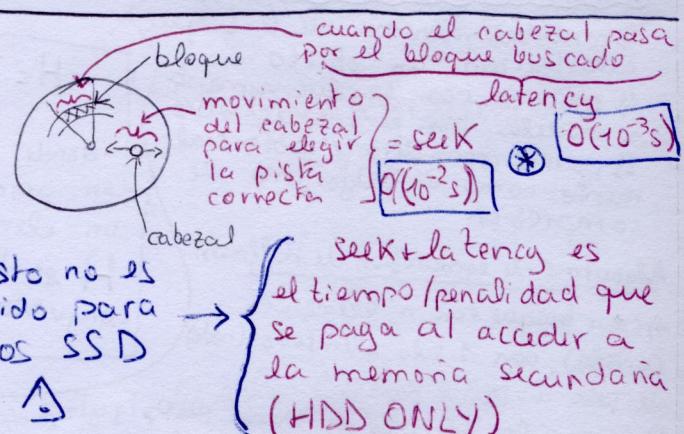
$n = \frac{N}{B}$: tamaño de bloques del input



④ Concluir que $P = \Omega(g(n))$
(por contradicción sobre límites u ordenes asintóticos)

Lógica:

- 1) transformar el input de Q en uno de P en $O(g(n))$
- 2) resolver P con el input transformado
- 3) transformar el output de P en uno de Q en $O(g(n))$
- 4) Concluir que P tiene la misma cota inferior, es decir $P = \Omega(g(n))$. De lo contrario las transformaciones nos darían una solución a Q de tiempo $O(g(n))$, lo que es imposible.



⇒ se concluye que si tenemos datos ordenados de manera secuencial en el disco, se obtienen mejores tiempos dado que no es necesario mover el cabezal y pagar el seek + latency

Nota: La memoria principal (RAM) realiza acceso a datos en $O(10^{-9} s)$

- El modelo de memoria externa abstracta de las arquitecturas de HDD y SSD (por simplicidad).
- La lectura/escritura en memoria 2^{ra} es tan cara que nos permite despreciar las operaciones del algoritmo en CPU y RAM.

Cota inferior de ordenamiento en memoria secundaria $\text{sort}(N)$:

$$\begin{aligned}\text{sort}(N) &= \frac{N}{B} \log_B \left(\frac{N}{M} \right) \\ &= n \log_B \left(\frac{N}{M} \right) \\ &= n \log_B \left(\frac{1}{m} \right)\end{aligned}$$

} distintas notaciones usando cosas distintas $\star\star$

Hashing extensible

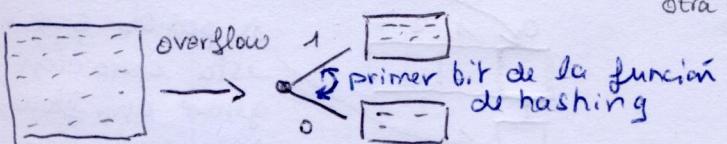
Dada una "buena" función de hashing $h(\cdot)$, se insertan elementos en una página hasta que esta se llene.

Cuando esto ocurre, tomamos los elementos de la página llena y aplicamos la función $h(\cdot)$ a cada elemento, lo que nos permite separar la información en 2 páginas.

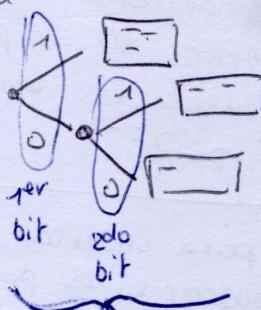
De llenarse alguna de estas páginas, se vuelve a hacer lo mismo, pero ocupando el 2do bit resultante de aplicar la función $h(\cdot)$.

En el caso en que no queden bits para comparar, se hace overflow con una lista enlazada. $\star\star$

Para el caso de eliminar elementos, hay que eliminar el elemento luego de haberlo encontrado, lo cual podría conllevar una eliminación de la página en donde estaba el elemento. También se pueden mezclar nodos hermanos.



si se llena otra página
⇒



$\star\star$ Se refiere a una función que se comporta de manera aleatoria.

tiene forma de trie

... se repite hasta que se acaban los bits de $h(\cdot)$

• costo búsqueda:

1 lectura

• costo inserción:

1 o 2 escrituras

• costo borrado:

1 o 2 lecturas + 1 escritura

este es también el criterio utilizado para saber si expandir o reducir la cantidad de páginas utilizadas. Esto implica que la inserción que causa la expansión no necesariamente se expande, pero no debería ser un problema con una "buena" función de hashing. $\star\star$

• Al insertar un elemento y , se inserta en la página $h(y) \bmod 2^t$

• En caso de que una página reballe, se genera una lista enlazada a esta con los elementos extra (no debiera ocurrir)

• Para eliminar un elemento, se busca y luego se elimina. En caso de eliminar suficientes elementos, la lista puede contraerse, por lo que los elementos en las páginas i e $i+2^t$ se vuelven a insertar en la página i $t \in t-1, p \in p-1$

↓ explicación visual al revés

$\star\star$ Si la función de hashing es buena, en el peor caso habría que insertar $0.69 \cdot 2^{64}$ elementos para llegar a overflow per lista enlazada

llenado promedio de una página

Hashing lineal

↳ almacena O(1) datos en memoria

↳ permite controlar:

- el % de llenado de los bloques
- el costo promedio de búsquedas

Sea $h(\cdot)$ una "buena" función de hashing. $\star\star$

Sea P el número total de páginas en disco, con $2^t \leq P \leq 2^{t+1}$

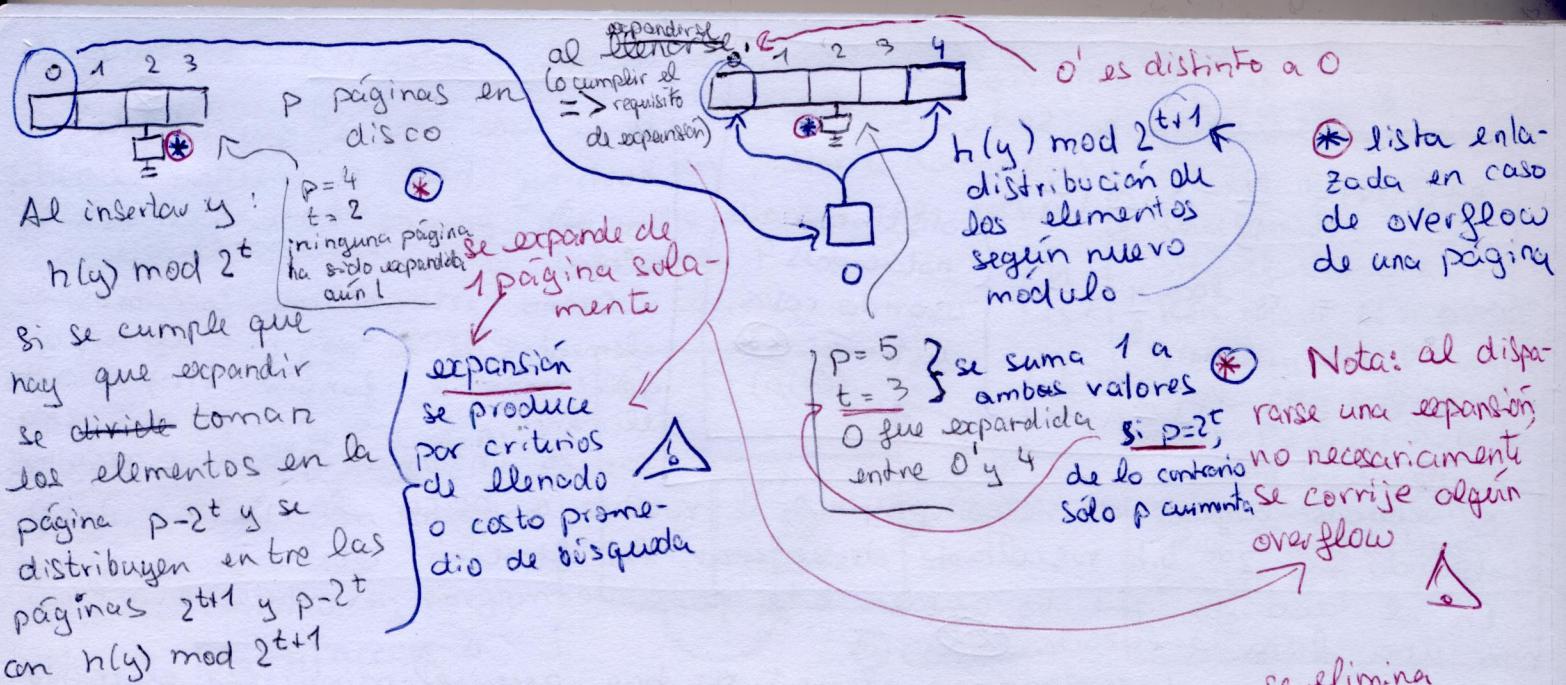
Sea y un elemento, el cual será insertado dentro del hashing lineal.

Las páginas i , con $0 \leq i \leq P-2^t$, ya han sido expandidas.

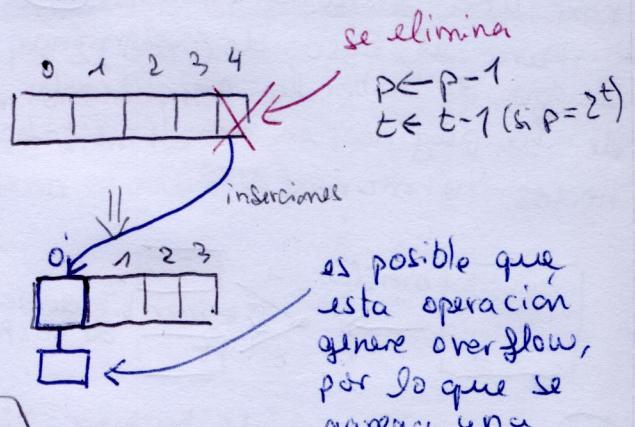
Las páginas j , con $P-2^t \leq j \leq 2^t$, aún no han sido expandidas.

Inicialmente, $p=1$ y $t=0$.

Al ser expandida, una página i reporta su contenido entre la página i y $i+2^t$.



=>
 al eliminarse un elemento y querer reducir la cantidad de páginas (contraerse)
 el proceso es similar, solo se reduce la cantidad de páginas P , y además de t , para finalmente reinsertar los elementos de la página P e insertarlos en la página $P-2^{t-1}$



- Arbol B: - $O(\log_B \frac{n}{m})$ para insertar, borrar y buscar.
 - óptimo para buscar si es por comparaciones
 - ocupación promedio 69%
 - permite recuperar los m objetos en $O(\log_B \frac{n}{m} + occ)$
 - encontrar predecesor $O(\log_B \frac{n}{m})$
- Ordenamiento: $O(n \log_m n)$ o $O(n \log_{m/m} n)$, óptimo si es por comparaciones
- Cole de prioridad: - $O(1/B \log_m n)$ (amortizado), \geq óptimo si es
 - si $\log \frac{n}{m} \leq m^d$, para $0 < d < 1$ por comparaciones
- Hashing extendible: - para $N = O(MB)$
 - inserta, borra y busca en $O(1)$ promedio (con buen hash)
 - con uno que produce t colisiones, el costo es $O(Tt/B)$
 - ocupación promedio 69%.
- Hashing lineal: se puede acotar el costo promedio $\frac{1}{2} + \frac{1}{2} \rightarrow -\Theta$
 la ocupación promedio, a costa de la otra medida Θ
- R-Trie: - $O(\log_B \frac{n}{m})$ para insertar y borrar rectángulos
 - $O(nB)$ para encontrar intersecciones o consultas de contiene (en promedio)