Algoritmica (ADA)

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Tema 1: Eficiencia

Algoritmo → serie finita de instrucciones no ambiguas que expresan un método de resolución de problemas

Traspuesta de una matriz

```
void trans (mtx &A)
  for (int i=1; i< A.rows; i++)
    for (j=0; j<i; j++)
      swap (...)
```

For externo: $1 + n + n - 1 = 2n$

For interno:

$$\sum_{i=0}^{n-1} (2+3n) = \frac{(n-1)(2+3n+1)}{2} = \frac{8n+3n^2-4}{8}$$

$T(n) = O(n^2)$

Cota superior → caso peor $(C_s) \to O$
Cota inferior → caso mejor $(C_i) \to \Omega$
Cota promedio → $C_m(n) \to$
↳ no es la media

$$O(f) = \{ g : \mathbb{N}^* \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, g(n) \leq c f(n) \}$$

Principio de invarianza

Propiedades cota superior

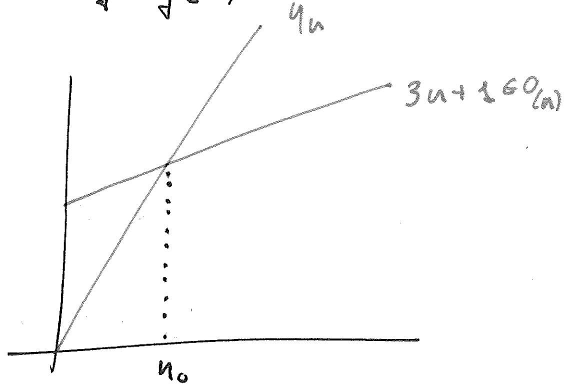
1) $f \in O(f)$

2) $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \to f_1 + f_2 \in O(\max(g_1, g_2))$

3) $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \to f \in O(g) \wedge g \notin O(f)$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \to f \notin O(g) \wedge g \in O(f)$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R} \to f \in O(g) \wedge g \in O(f)$



Coste exacto (Θ)

$\Theta(f) = O(f) \cap \Omega(f)$

$\left.\begin{array}{l} \lim_{n\to\infty} \frac{f(n)}{g(n)} = K \\ K \neq 0 \\ K \neq \infty \end{array}\right\} \longrightarrow \Theta(f) = \Theta(g)$

$\log_b(x) = \frac{\log_c(x)}{\log_c(b)}$ $\quad \log a^b = b \log(a)$ $\quad a = \log_b(c)$, $c = b^a$

Progresión geométrica

$S = \sum_{i=0}^{n} a_i = a_0 \frac{(r^n - 1)}{(r-1)}$

Resolución de ecuaciones en recurrencia

Recurrencia homogenea $\rightarrow a_k T(n-k) = 0$

$T(n) = \sum_{i=1}^{k} c_i p_i(n) r_i^n$

Ejemplo búsqueda Binaria

Complejidad $\rightarrow \log_2(n)$

$T(n) = \begin{cases} 1 & n = 1 \\ 1 + T(\frac{n}{2}) & n > 1 \end{cases}$

Primera iteración $\rightarrow T(n) = 1 + T(\frac{n}{2})$

$= 1 + 1 + T(\frac{n}{4})$

$= 3 + T(\frac{n}{8}) = K + T(\frac{n}{2^K})$

Valor de $K \rightarrow \frac{n}{2^K} = 1 \longrightarrow n = 2^K$

$a = b^c$

$c = \log_b a$

$\hookrightarrow K = \log_2(n)$ $\quad T(n) = \log_2(n) + T\left(\frac{n}{2^{\log_2(n)}}\right)$

$= T(n) = \log_2(n) + T(\frac{n}{n}) =$

$T(n) = \log_2(n) + T(1)$

$T(n) = \log_2(n) + 1 \rightarrow T(n) \in \Theta(\log_2(n))$

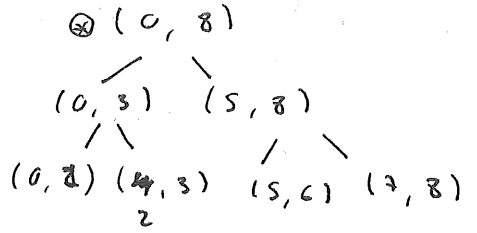
Jerarquía de funciones

$O(1) \subset O(\log(\log(n))) \subset O(\log(n)) \subset O(\log^{(a>1)}(n)) \subset O(\sqrt{n})$ en

$\subset O(n) \subset O(n\log(n)) \subset O(n^2) \subset O(n^{a(>2)}) \subset O(2^n) \subset$ [illegible]

$\subset O(n!) \subset O(n^n)$

$$T(n) = \begin{cases} 1 & n = 1 \\ n + T(n-1) & n > 1 \end{cases}$$



(1) $T(n) = n + T(n-1)$

(2) $= n + n - 1 + T(n-2)$

$= 2n - 1 + 3n - 3 + T(n-3)$

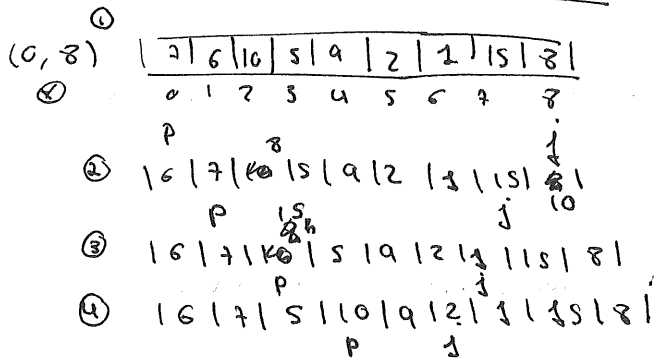
$T(n) = \sum_{i=n-k+1}^{n} i + T(n-k)$ → Acaba cuando $(n-k) = 1$

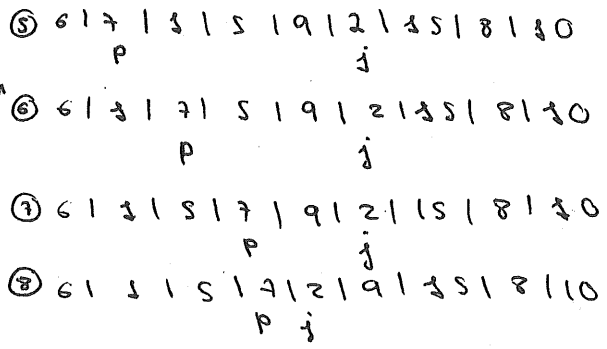
$k = n-1$

$T(n) = \sum_{i=n-n+1+1}^{n} i + T(n-n+1) = \sum_{i=2}^{n} i + T(1) \in \Theta(n^2)$ //

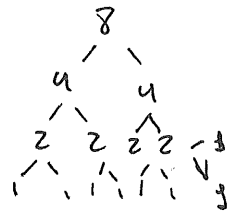
Quicksort (Primer Elemento)

~~void quicksort (int v[], int pri, int ult)~~

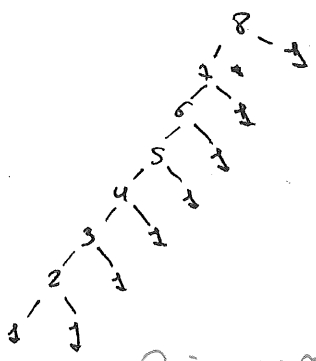




Caso mejor



Caso peor



Mejor caso

$$T(n) = \begin{cases} 1 & n \leq 1 \\ n + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

Peor caso

$$T(n) = \begin{cases} 1 & n \leq 1 \\ n + T(0) + T(n-1) & n > 1 \end{cases}$$

Segunda llamada

Primera llamada

$T(n) = n + 2T\left(\frac{n}{2}\right)$ (1) $\frac{n}{4}$

$= n + 2\left(\frac{n}{2} + 2T\left(\frac{\frac{n}{2}}{2}\right)\right) = n + 2\left(\frac{n}{2} + 2T\left(\frac{n}{4}\right)\right) = n + \frac{2n}{2} + 4T\left(\frac{n}{4}\right) = \frac{4n}{2} + 4T\left(\frac{n}{4}\right)$

(2) $= 2n + 4T\left(\frac{n}{4}\right)$

(3) $= 2n + 4\left(\frac{n}{4} + 2T\left(\frac{n}{8}\right)\right) = 3n + 8T\left(\frac{n}{8}\right)$

(k) $= kn + 2^k T\left(\frac{n}{2^k}\right)$ → Igualar a 1

k veces

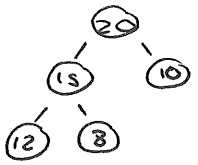
Última recursividad → $\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log_2(n)$

$T(n) = \log_2(n) \cdot n + 2^{\log_2(n)} \cdot T\left(\frac{n}{\log_2(n)}\right)$ ⊛

⊛ $\log_2(n) \cdot n + nT\left(\frac{n}{n}\right)$

$\log_2(n) \cdot n + n \rightarrow T(n) \in \Omega(n \log(n))$ //

Heapsort

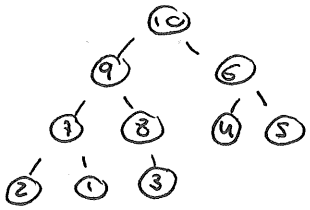


Ejemplo árbol completo

→ El de abajo tiene que ser menor al de arriba

→ Los montículos tienen que estar a la izquierda siempre sin dejar huecos

root (i) = 0
left_child (i) = 2i + 1
parent (i) = (i - 1)/2
right_child (i) = 2i + 2



top → conocer el valor de la cima → $O(1)$

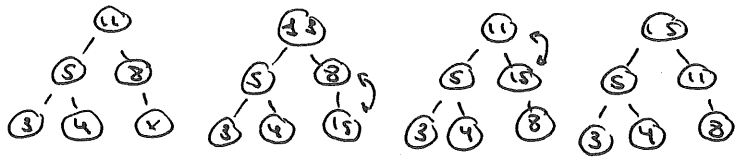
pop → borrar la cima → $O(\log n)$

↳ $\Omega(1)$

push → insertar un elemento → se coloca al final y se reorganiza el árbol → $O(\log(n))$

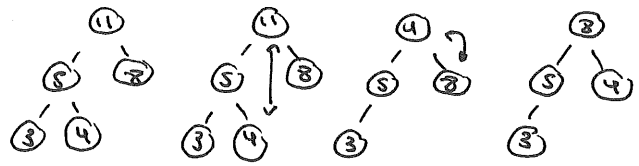
Push → insertar un elemento

1) Añadir elemento al final del vector el elemento con su padre
2) Flotarlo
3) Volver (2)



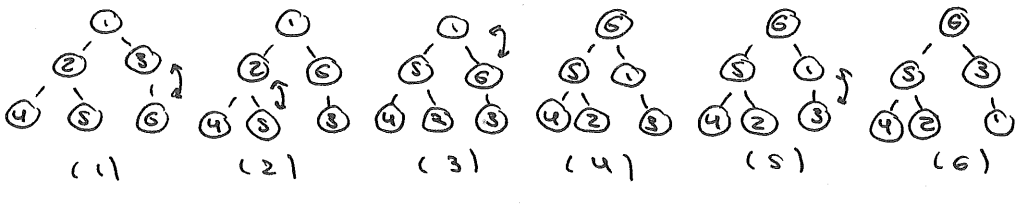
Pop → borrar la cima

1) Reemplazar el nodo raíz del árbol por el nodo que está en último lugar
2) Hundir la nueva raíz
3) Volver (2)



Heapificación

1) Asumir que el vector es un árbol esencialmente completo
2) Hundir cada nodo del árbol desde el que ocupa la posición $\left(\frac{n}{2} - 1\right)$ hasta la raíz



Ejemplo → complejidad torres de Hanoi $T(n) = 2(T(n-1) + 1$

(1) $T(n) = 2T(n-1) + 1 \rightarrow n = n-1$

(2) $= 2(2T(n-(1+1)) + 1) = 2^2 T(n-2) + 1$

(k) $= 2^k T(n-k) + 1$

$T(n-k) = 1$ $\quad k = n-1$

$n-k = 1$

$2^{n-1} T(1) + 1 \in \Theta(n, 2^n)$ //

Problemas propuestos Tema 1 libro

Jaime Hernández Delgado
u8176550

Ejercicio 2 → abstracto Tamaño → n (cte.) depende unicamente de n

Para esta función podemos ver que tiene un bucle for que imprime "*" y otro bucle for que llama a abstracto dividiendo el parametro entre 2 cuatro veces, tampoco podemos ver que haya ni peor ni mejor caso

Para el primer bucle tenemos que

$$\sum_{i=1}^{n-1} j = \frac{(n+1-1)(1+n)}{2} = \frac{2n}{2} = n \in \Theta(n)$$

↳ como es una serie aritmética usamos

$$\sum_{i=1}^{n} i = \frac{(n+1)(a_0+a_n)}{2}$$

$n + 4T\left(\frac{n}{2}\right)$ $n = n + 4T\left(\frac{n}{2}\right)$

$\downarrow$ $n = \frac{n}{2}$

(2) $T(n) = n + \left(n + 4T\left(\frac{n}{2^2}\right)\right) =$

$= 2n + 4T\left(\frac{n}{2^2}\right)$

Para el segundo bucle sin embargo vemos que se ejecuta cuatro veces teniendo así como ecuación de recurrencia:

$$T(n) \begin{cases} 1 & n \leq 1 \\ n + 4T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

(1) $T(n) = n + 4T\left(\frac{n}{2}\right)$

(2) $= n + 4\left(\frac{n}{2} + 4T\left(\frac{n}{4}\right)\right) = 3n + 4^2 T\left(\frac{n}{2^2}\right)$

(3) $= 3n + 4^2\left(\frac{n}{2^3} + 4T\left(\frac{n}{2^3}\right)\right) = 5n + 4^3 T\left(\frac{n}{2^3}\right)$

(k) $= (2k-1)n + 4^k T\left(\frac{n}{2^k}\right)$ → igualamos a 1 → $\frac{n}{2^k} = 1$; $n = 2^k$; $k = \log_2(n)$ para obtener el caso base

$(2(\log_2(n)) - 1)n + 4^{\log_2(n)} T(1)$

① ②

① $n\log_2(n)$

② $2^{2\log_2(n)}$; $a^{\log_a b} = b$; n^2

↓ Aplicamos la propiedad de los logaritmos

Tenemos entonces $O(n\log_2(n))$ y $O(n^2)$ (lineal logarítmica, polinómica)

como $O(n\log_2(n)) < O(n^2)$ la complejidad de esta función es $O(n\log_2(n))$ //

La función lineal-algorítmica crece más rápido que la polinómica

Jaime Hernández Delgado
48776551W

Ejercio 3 → Palin

Tamaño → n = ult - puim

En esta función sí que hay un peor y mejor caso, ya que dependiendo del string la complejidad temporal varía

Mejor Caso

Se da cuando pal[puim] es distinto de pal[ult] ya que no entra dentro de la recursión por lo tanto el mejor caso es una constante ya que sólo ha ejecutado una operación elemental (OE)

$\Omega(1)$

Peor Caso

$1 + T(n-2)$ $\quad n = 1 + T(n-2)$ $\quad \downarrow n = n-2$

$1 + (1 + T(n-2-2))$

$$T(n) \begin{cases} 1 & n \leq 1 \\ 1 + T(n-2) & n > 1 \end{cases}$$

(1) $T(n) = 1 + T(n-2)$
(2) $= 1 + (1 + T(n-2-2)) = 2 + T(n-4)$
(3) $= 2 + (1 + T(n-4-2)) = 3 + T(n-6)$
(K) $= K + T(n-2K)$

↳ Ah no depender de n no lo tenemos en cuenta

⊛ $T(n-2K)$ → igualamos a 1 para obtener el caso base

$n - 2K = 1$
$-2K = 1 - n$
$K = \frac{1-n}{-2}$

$\frac{1-n}{-2} + T(n - 2(\frac{1-n}{-2}))$:

$= \frac{1-n}{-2} + T(n - (-1+n))$:

$= \frac{1-n}{-2} + T(n+1-n) = \frac{1-n}{-2} + T(1) \in O(n)$ //

Jaime Hernández Delgado
48776050W

Práctica 3: Ejercicio 2

Para la resolución del problema tenemos dos bucles uno externo while → ejecuta $i \leq w$
Donde el bucle interno decrementa j hasta 0
También tenemos que tener en cuenta incrementa tres veces el anterior

En esta situación no hay caso mejor o peor ya que depende de (n) todo el resto, y no se puede llegar a un caso el cual sea mejor que otro

Para el análisis del algoritmo tenemos:

Nº	i	p	j	q	w
(1)	1	0	1	0	6
(2)	3	1	0	1	6
(3)	9	0	1	0	6
(k)	3^k		∅		

$$T(n) = \sum_{i=1}^{\lfloor \log_3(n) \rfloor} 3^k \rightarrow \text{esto es una serie geométrica}$$

$3^k = n$
$a = b^c$
$c = \log_b(a)$

$$k = \log_3(n) \qquad \sum_{k=0}^{n} ar^k = \frac{a_0(r^{n+1} - 1)}{r - 1}$$

despejando

$$= 3\frac{(3^{(\log_3(n)+1} - 1)}{3 - 1} = \frac{3(n-1)}{2} = T(n) \in O(n) //$$

despeje de k

Jaime Hernández Delgado

48776654 W

Ejercicio 3

Tamaño → n = V.size()

En este ejercicio tenemos que calcular la complejidad tenemos un mejor caso y peor caso

El mejor caso es cuando el tamaño del vector es ($n \leq 1$) ya que no entra en ningún momento en el bucle (for), es decir es de orden constante $c_i(n) = 1 \in \mathcal{O}(1)$ ~~es de orden constante~~.

En este caso al depender del tamaño del bucle entra dentro del bucle (for) donde hay 1 OE ya que es lineal, de esta manera calculando todo quedaría

$$T(n) = \sum_{i=1}^{n} \left(\sum_{j=n-1}^{i} 1 \right)$$

② ①

① El primer sumatorio es aritmético, para resolverlo tenemos

$$\sum_{i=n-1}^{i} 1 = \text{Invertimos el sumatorio desde } i=0 \text{ hasta } (n-1) \rightarrow \sum_{i=0}^{n-1} 1 = \left\{ \begin{array}{l} \text{usamos la} \\ \text{fórmula de} \\ \text{la serie aritmética} \end{array} \right\} \rightarrow$$

$$\rightarrow \sum_{i=1}^{n} i = \frac{(n+1)(a_0+a_n)}{2} = \frac{(n-1+1)(1+1)}{2} = \frac{2n}{2} = n \in$$

② Una vez hecho el sumatorio interno realizamos el externo

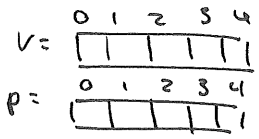
$$\sum_{i=1}^{n} n = \frac{(n+1)(n+n)}{2} = \frac{n^2+n^2+n+n}{2} = \frac{2(n^2+n)}{2} = n^2+n \in \mathcal{O}(n^2) \;//$$

Práctica 4

Jaime Hernández Delgado : 48776554W

Ejercicio 1 → Mochila

Tamaño $n = i + 1$



p = peso máximo

Mejor caso $\forall p[i] > P$ a $0 \le i \le n-1$

$$T(n) = \begin{cases} 1 & n \le 0 \\ 1 + T(n-1) & n > 0 \end{cases}$$

Mejor Caso

(1) $T(n) = 1 + T(n-1)$

(2) $= 1 + 1 + T(n-1-1) = 2 + T(n-2)$

(3) $= 2 + 1 + T(n-2-1) = 3 + T(n-3)$

(K) $= K + T(n-K)$

$n - K = 1 \rightarrow -K = -n$, $K = n$

$T(n) = n + T(0) = n + 1 \in \Omega(n)$ //

Peor Caso

→ El peso máximo es mayor al peso total de la mochila

$\sum p[i] \le P$ a $0 \le i \le n-1$

$$T(n) = \begin{cases} 1 & n \le 0 \\ 1 + 2T(n-1) & n > 0 \end{cases}$$

(1) $T(n) = 1 + 2T(n-1)$

(2) $= 1 + 2\left(1 + 2T(n-2)\right) = 3 + 4T(n-2)$

(3) $= 3 + 4\left(1 + 2T(n-3)\right) = 7 + 8T(n-3)$

(K) $= 2^K - 1 + 2^K T(n-K)$

$n - K = 1$, $n = K$

$T(n) = 2^n - 1 + 2^n T(0) = T(n) = 2^{n+1} \in O(2^n)$ //

Jaime Hernández Delgado
48776654W

sucio //

Ejercicio 2 → Abstracto

"Para el cálculo de este Algoritmo tenemos que tener en el mejor caso y peor caso para ello el mejor caso tenemos que el paso de $n \leq 1$

$$T(n) \begin{cases} 1 & n \leq 1 \\ T\left(\frac{n}{2}\right)+1 & n > 1 \end{cases}$$

para el mejor caso tenemos $T(n) = 1 \in \Omega(1)$

(1) $T(n) = 1 + T\left(\frac{n}{2}\right)$

(2) $= 1 + 1 + T\left(\frac{n}{4}\right) = 2 + T\left(\frac{n}{4}\right)$

(3) $= 2 + 1 + T\left(\frac{n}{8}\right) = 3 + T\left(\frac{n}{8}\right)$

(k) $= k + T\left(\frac{n}{2^k}\right)$

$n = 2^k$
$a = b^c$
$c = \log_b(a)$

$k = \log_2(n)$

$\longrightarrow (k) = \log_2(n) + T\left(\frac{n}{\log_2(n)}\right)$

Ejercicio 2 → abstracto ~~[illegible]~~ Jaime

(1) $T = n + 1 \cdot T\left(\frac{n}{2}\right)$

(2) $T = n + 1\left(\frac{n}{2^2}\right)$

(1) $T(n) = n + 2T(n-1)$

(2) $= n + 2(n-1-1) + n + n + 2T(n+n-1) = 2n + 2T(2n-1)$

(3) $= 2n + 2(n +$

ADA (Repaso Examen parcial)

$$T(n) = \begin{cases} 1 & n = 1 \\ 1 + T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

$n = \frac{n}{2}$

(1) $T(n) = 1 + T\left(\frac{n}{2}\right)$

(2) $T(n) = 1 + T\left(\frac{n}{2^2}\right) + 1 = 2 + T\left(\frac{n}{2^2}\right)$

$= 2 + 1 + T\left(\frac{n}{2^3}\right) = 3 + T\left(\frac{n}{2^3}\right)$

$= k + T\left(\frac{n}{2^k}\right)$ $\quad \frac{n}{2^k} = 1$ $\quad n = 2^k$

$k = \log_2(n)$

$T(n) \in \Theta(\log_2(n))$ //

$$T(n) = \begin{cases} 1 & n = 1 \\ n + T(n-1) & n > 1 \end{cases} \qquad n = n - 1$$

(1) $T(n) = n + T(n-1)$

(2) $= n + (n-1) + T(n-2) = 2n - 1 + T(n-2)$

(3) $= 2n - 1 - 1 + n + T(n-1-2) = 3n - 3 + T(n-3)$

(k) $= kn - k + T(k-3)$

$$T(n) = \begin{cases} 1 & n \le 1 \\ n + 4T\left(\frac{n}{2}\right) & \end{cases} \qquad n = \frac{n}{2}$$

(1) $= n + 4T\left(\frac{n}{2}\right)$

$n + 4\left(\frac{n}{2} + 4T\left(\frac{n}{2^2}\right)\right) = n + 2n + 4^2 T\left(\frac{n}{2^2}\right)$

$= 3n + 4\left(\frac{n}{2} + 4T\left(\frac{n}{2^3}\right)\right) = 3n + 2n + 4^3 T\left(\frac{n}{3^3}\right)$

$5n + 4\,4^3 T\left(\frac{n}{2^3}\right)$

(k) $= (2k-1)n + 4^k T\left(\frac{n}{2^k}\right)$ $\quad n = 2^k$

$k = \log_2(n)$

$(2\log_2(n) - 1)n + 4^{\log_2(n)}$

$\hookrightarrow n(\log_2(n))$

Repaso (final test parcial)

$$T(n) = \begin{cases} 1 & n = 1 \\ n + 4T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

$n = \frac{n}{2}$

(0) $T(n) = n + 4T\left(\frac{n}{2}\right)$

(2) $= n + 4\left(\frac{n}{2} + 4T\left(\frac{n}{2^2}\right)\right) = n + 2n + 4T\left(\frac{n}{2^2}\right) = 3n + 4T\left(\frac{n}{2^2}\right)$

(3) $= 3n + 4\left(\frac{n}{2^2} + 4T\left(\frac{n}{2^3}\right)\right) = 3n + 2n + 4^2 T\left(\frac{n}{2^3}\right)$

(k) $= (2k-1)n + 4^k T\left(\frac{n}{2^k}\right)$ 1

$\frac{n}{2^k}$ $n = 2^k$ $k = \log_2(n)$

$2(\log_2(n)-1)n$

$\sum_{i=0}^{j/2} n =$

$$T(n) \begin{cases} \Theta(n^k) \\ \Theta(n \log_2 n) \\ \Theta(n \log_2 n) \end{cases} \quad T(n) = n + 4T\left(\frac{n}{2}\right) + \Theta(n) \quad \begin{matrix} n > b^k \\ n = b^k \\ n < b^k \end{matrix}$$

$$T(n) = \begin{cases} 1 & n = 1 \\ n + 2T(n-1) & n > 1 \end{cases}$$

$n = n-1$

(1) $T(n) = n + 2T(n-1)$

(2) $= n + 2(n-1) + 2T(n-1-1) = n + 2n - 2 + 2T(n-2)$

$= 3n - 2 + 2T(n-2)$

(3) $= 3n - 2 + 2(n-1+2T(n-2-1)) = 3n - 2 + 2n - 2 + 4T(n-3)$

$= 5n - 4 + 4T(n-3)$

$(k) = (2k-1)n - 2^{k-1} + 2^{k-1} T(n-k)$ $n-k = 1$

$k = 1-n$ $-k = 1-n$

$k = n-1$

$k-1 = n-1-1$

$n = 2$

$(2(n-1)-1)n$

$(2(n-1)-1)n$

$2n-1\,n$

$2n^2 - n - 2$

Repaso examen ADA

$$f(n) = \begin{cases} 1 & n \leq 1 \\ n + 4f\left(\frac{n}{2}\right) & n > 1 \end{cases} \qquad n + 4\left(\frac{n + 4T(n/2)}{2^2}\right)$$

$2n + 4$

(1) $T(n) = n + 4T\left(\frac{n}{2}\right)$

(2) $= n + 4\left(\frac{n + 4T(n/2)}{2^2}\right) = n + n + 4T\left(\frac{n}{2}\right)$

(3) $= n + n + 4\left(\frac{n + 4T\left(\frac{n}{2}\right)}{2^2}\right) = 2n + 4\left(n + 4T\left(\frac{n}{2}\right)\right)$

$2n + 4n + 16T\left(\frac{n}{2}\right)$

$6n + 16\,4^2 T\left(\frac{n}{2}\right)$

(k) $= 2^k n + 4^{k-1} T\left(\frac{n}{2^k}\right) \quad k = \log_2(n)$

$n 2^{\log(n)} + 4^{\log(n)-1} \log_2$

(2) $T(n) = n + 2T\left(\frac{n}{2}\right)$

(1) $= n + 2T\left(\frac{n + 2T(n/2)}{2^2}\right) = \frac{2n + 2T\left(\frac{n}{2^2}\right)}{2} = n + T\left(\frac{n}{2}\right)$

$$f(n) \begin{cases} 1 \\ f\left(\frac{n}{2}\right) + 1 \end{cases}$$

(1) $T(n) = 1 + T\left(\frac{n}{2}\right)$

(2) $= 1 + T\left(\frac{1 + T\left(\frac{n}{2}\right)2^2}{2^2}\right) = 1 + \frac{1}{2^2} +$

$$\sum_{j=1}^{i}$$

(1) $T(n) = n + 4T\left(\frac{n}{2}\right)$

(2) $= n + 4T\left(\frac{n + 4T\left(\frac{n}{2^2}\right)}{2^2}\right) = 2n + 4T\left(\frac{n}{2^2}\right)$

(3) $= 2n + 4T\left(\frac{n + 4T\left(\frac{n}{2^3}\right)}{2^3}\right) = 2n + \frac{n + 4T\left(\frac{n}{2^3}\right)}{2} = \frac{3n + 4T\left(\frac{n}{2^3}\right)}{2}$

$T(n) = n + 2T\left(\frac{n}{2}\right)$

$= n + 2T\left(\frac{n + 2T\left(\frac{n}{2^2}\right)}{2^2}\right) = \frac{2n + n + 2T\left(\frac{n}{2^2}\right)}{2} = \frac{3n + 2T\left(\frac{n}{2^2}\right)}{2}$

Repaso final (Domingo)

15/04/2024

$$\sum_{i=2}^{i=n-1} \sum_{j=1}^{j=i} n \rightarrow \frac{n(n-i)}{2} = \frac{n^2 - ni}{2}$$

Solución Recursiva con Almacen

$M[i][j] = \max(M[i-1][j]+k, M[i-1][j-1]+v)$

Versión iterativa

$$T(n,w) = 1 + \sum_{i=1}^{n} 1 + \sum_{i=0}^{w} 1 + \sum_{i=1}^{n} \sum_{j=1}^{w} 1 = 1 + n + w + 1 + w(n+1)$$

$T(n,w) \in \Theta(nw)$

$S(n,w) \in \Theta(nw)$ → complejidad espacial mejorable (vectorial)

Masther Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + g(n)$$

$$T(n) = \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_b(n)) & a = b^k \\ \Theta(n^{\log_b(a)}) & a > b^k \end{cases}$$

Reducción → $a = b = a$

$$T(n) = aT\left(\frac{n}{a}\right) + g(n) \qquad g(n) \in \Theta(n^k)$$

$$T(n) = \begin{cases} \Theta(n) & k < 1 \\ \Theta(n \log(n)) & k = 1 \\ \Theta(n^k) & k > 1 \end{cases}$$

Masther Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + g(n)$$

$$T(n) = \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_b(n)) & a = b^k \\ \Theta(n^{\log_b(a)}) & a > b^k \end{cases}$$

Reducción → $a = b = a$

$$T(n) = aT\left(\frac{n}{a}\right) + g(n) \qquad g(n) \in \Theta(n^k)$$

$$T(n) = \begin{cases} \Theta(n) & k < 1 \\ \Theta(n \log n) & k = 1 \\ \Theta(n^k) & k > 1 \end{cases}$$

Reducción

$T(n) = aT\left(\frac{n}{a}\right) + g(n) \quad g(n) \in \Theta(n^k)$

$$T(n) \begin{cases} n^k & k > 1 \\ n \log n & k = 1 \\ n & k < 1 \end{cases}$$

Master Theorem

$T(n) = aT\left(\frac{n}{b}\right) + g(n)$

$$T(n) = \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_b(n)) & a = b^k \\ \Theta(n^{\log_b(a)}) & a > b^k \end{cases}$$

Master theorem

$T(n) = aT\left(\frac{n}{b}\right) + g(n)$

$$T(n) \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_b(n)) & a = b^k \\ \Theta(n^{\log_b(a)}) & a > b^k \end{cases}$$

Reducción → $b = a = a$

$T(n) = aT\left(\frac{n}{a}\right) + g(n) \quad g(n) \in \Theta(n^k)$

$$T(n) \begin{cases} \Theta(n) & k < 1 \\ \Theta(n \log n) & k = 1 \\ \Theta(n^k) & k > 1 \end{cases}$$

Master theorem

$T(n) = aT\left(\frac{n}{b}\right) + g(n)$

$$T(n) \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_b(n)) & a = b^k \\ \Theta(n^{\log_b(a)}) & a > b^k \end{cases}$$

Master theorem

$T(n) = aT\left(\frac{n}{b}\right) + g(n)$

$$T(n) \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_b(n)) & a = b^k \\ \Theta(n^{\log_b(a)}) & a > b^k \end{cases}$$

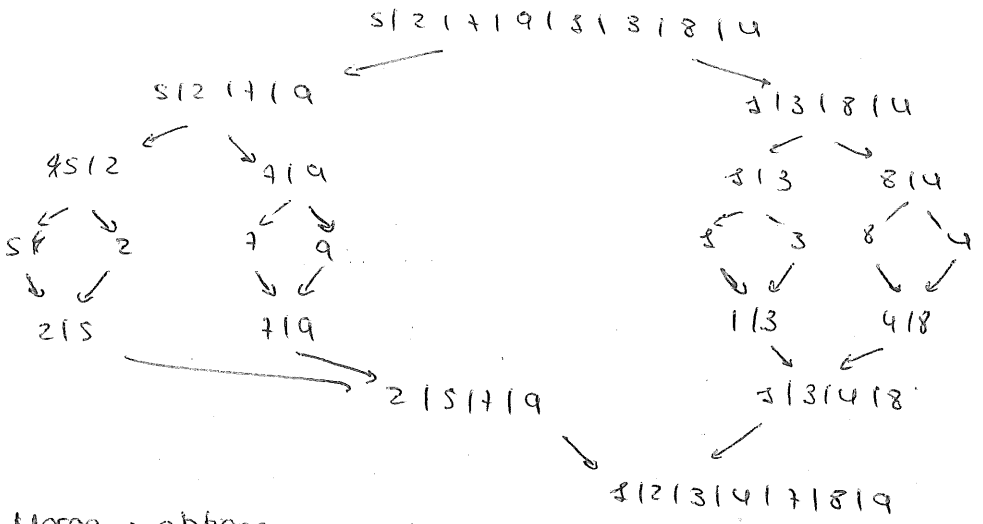
07/03/2024

Tema 3: Divide y vencerás

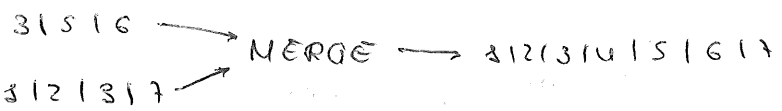
Algoritmo Mergesort

→ Ordeno de forma ascendente un vector V de n elementos

→ Solución usando el algoritmo: divide y vencerás



Merge → obtiene un vector ordenado como unión de dos vectores también ordenados




```
void merge (vector<int> &V, size_t first, midle, last){
    vector<int> v_merged;
    v_merged.reserve (last - first);
    size_t f = first; size w = middle;
    while (f != midle && r != last){
      if (v[f] < v[r]){
        v_merged.push back (v[f]); ++f;
      }else{
        v_merged.pushback (v[r]); ++r;
      }
    }
    for ( ; f != midel; ++f ) v_merged.pushback(v[f])
    for( ; r != last; ++r) v_merged.pushback (v[r])
```

29/02/2024

~~Tema 2~~:

→ un árbol completo con (n) nodos $\exists p \in \mathbb{N} \mid n = 2^p - 1$ $\quad a = b^c$ $\quad c = \log_b a$

Este árbol tiene $\left(\frac{n+1}{2} - 1\right)$ nodos internos

Caso más favorable → El vector de entrada ya es un montículo

$$c_i = \frac{n+1}{2} - 1 \in \Omega(n)$$

Nivel del árbol	Nodos (nivel)	Pasos	Total pasos en nivel
k (hojas)	2^k	$\log_2\left(\frac{n+1}{2^k - 1}\right)$	$2^k \times \log_2\left(\frac{n+1}{2^{k+1}}\right)$

$$c_s(n) = \sum_{k=0}^{\log_2(n+1)-1} 2^k \log_2\left(\frac{n+1}{2^{k+1}}\right) = n - \log_2(n+1) \in O(n) \;//$$

$$c_s(n) = \sum_{k=0}^{\log_2(n+1)-1} 2^k \log_2\left(\frac{n+1}{2^{k+1}}\right) = \sum_{k=0}^{\log_2(n+1)-1} 2^{\log_2(n+1)-1} \log_2\left(\frac{n+1}{2^{\log_2(n+1)-k+1}}\right)$$

$$\begin{cases} n+1 = 2^{k+1} \\ k+1 = \log_2(n+1) - 1 \end{cases} = \sum_{k=0}^{\log_2(n+1)-1} 2^{\log_2(n+1)-1} \log_2\left(\frac{n+1}{n+1}^{1}\right) = 0$$

$$\sum_{k=0}^{\log_2(n+1)-1} 2^{\log_2(n+1)-1} \cdot 0 \to 0 \;//$$

Algoritmo sink → auxiliar 4 2 //

~~void sink (int *v, size_t n, size_t i) {~~

Construcción de un montículo

$$c_s(n) = \sum_{k=0}^{\log_2(n+1)-1} 2^k \log_2\left(\frac{n+1}{2^{k+1}}\right) = \sum_{k=0}^{\log_2(n+1)-1} 2^k (\log_2(n+1) - (k+1)) =$$

$$= (\log_2(n+1) - 1) \sum_{k=0}^{\log_2(n+1)-1} 2^k - \sum_{k=0}^{\log_2(n+1)-1} k 2^k$$

$$S = \sum_{k=0}^{M} k 2^k \qquad rS = 0 \cdot 1 + 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + (M-1) 2^{(M-1)} + M 2^M$$

$$2S = 0 \cdot 2^1 + 1 \cdot 2^2 + 2^3 + \dots + (M-1) 2^M + M 2^{M-1}$$

$r = 2$

$a = 1$

$$S - 2S = 0 \cdot 1 - M 2^{M+1} + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + \dots + 1 \cdot 2^M$$

$$-S = -M \cdot 2^{M+1} + \sum_{k=1}^{M} 2^k \longrightarrow S = M 2^{M-1} - 2^1 \frac{(2^M - 1)}{2 - 1} = 2^{M+1}(M-1) + 2$$

$$\left[S = \sum_{k=0}^{M} 2^k k = 2^{M+1}(M-1) + 2 \right]$$

```
void mergesort ( vector<int> &v, size_t first, size_t last){
    if (last - first < 2)
      return;

    size_t midel = first + (last - first) / 2;
    mergesort (v, first, midel);
    mergesort (v, midel, last);

    merge ( v, first, midel, last);
}
```

⊛ Ejemplo

$T(n) = 8T\left(\frac{n}{2}\right) + n^5$ (8: h, 2: b, 5: k)

$b^k = 2^5 = 32 \quad T(n) \in \Theta(n^5)$

Complejidad

Tamaño → last - first

$\Theta(n)$ → merge

No hay mejor ni peor caso → recorre todos los elementos

Ecuación de recurrencia

↳ $T(n) = \begin{cases} 1 & n \leq 1 \\ n + 2T\left(\frac{n}{2}\right) & n > 1 \end{cases}$

Complejidad temporal → $f(n) = \Theta(n \log(n))$

Complejidad espacial → $\max_{0 \leq i \leq \log_2(n)} \frac{n}{2^i} = n$ → sin tener encuenta la pila de recursión (por bloque de memoria)

→ si añadimos una pila añadir un orden $(\log(n))$

Técnica divide y vencerás

Consiste:
1) Descomponer problemas más pequeños
2) Resolver cada subproblema
3) Combinar solución de los subproblemas

Consideraciones:
1) No siempre un problema de talla menor es fácil
2) La solución de los subproblemas no implica necesariamente que la solución del problema original se pueda obtener fácilmente

Análisis de eficiencia

Depende de:
1) Nº de subproblemas
2) Tamaño subproblemas
3) Grado intersección entre subproblemas

Ecuaciones de recurrencia:
- $g(n)$ → tiempo del esquema tamaño (n)
- b → cte división tamaño del problema

$$T(n) = nT\left(\frac{n}{b}\right) + g(n)$$

Solución general → si $\exists k \mid g(n) \in \Theta(n^k)$ (master theorem)

IMPORTANTE !!! ⊛ $T(n)$:

$$\begin{cases} \Theta(n^k) & \text{si } n < b^k \text{ manda el segundo término} \\ \Theta(n^k \log_b n) & \text{si } n = b^k \\ \Theta(n^{\log_b n}) & \text{si } n > b^k \text{ manda el primer término} \end{cases}$$

Teorema de reducción

↳ Los mejores resultados en cuanto a coste se consiguen cuando los subproblemas son aproximadamente del mismo tamaño (y no contienen subproblemas comunes)

Caso especial → si se cumple la condición del teorema de reducción

$(b = n = a)$

$$\boxed{T(n) = aT\left(\frac{n}{a}\right) + g(n)} \qquad g(n) \in \Theta(n^k)$$

$$T(n) \begin{cases} \Theta(n^k) & k > 1 \\ \Theta(n \log n) & k = 1 \\ \Theta(n) & k < 1 \end{cases}$$

Demostración:

Cálculo potencia enésima:

$$x^n = \begin{cases} 1 & n = 0 \\ x^{\frac{n}{2}} x^{\frac{n}{2}} & n \text{ (par)} \\ x^{\frac{n-1}{2}} x^{\frac{n-1}{2}} \times & n \text{ (impar)} \end{cases}$$

$$T(n) = 1 + 2T\left(\frac{n}{2}\right) \in \Theta(n)$$

$r = x^{\frac{n}{2}}$

$x^n = r^2$

$1 + T\left(\frac{n}{2}\right) \in \Theta(\log(n))$

Resolución ejercicios (k-ésimo mínimo)

1) Selección del k-ésimo mínimo (k no es una constante (si n crece k también))
   a) Recorrer el vector (buscar el menor k veces)
   b) Complejidad $\Theta(kn) \rightarrow \Theta(n^2)$
2) Construir un Heap, realizar k-1 (pop) y una operación top
   a) Complejidad $O(n + k \log(n))$
   b) Mejor que lo anterior pero mejorable
3) Quickselect → basado en Quicksort
   a) Complejidad $\Omega(n)$ y $O(n^2)$

Búsqueda Binaria - Dicotómica

```
int bb (Elem v[], int p, int u, Elem e) {
    if (p > u) return -1
    int m = (p+u)/2
    if (e == v[m]) return m;
    if (e < v[m]) return bb (v, p, m-1, e);
    return bb (v, m+1, u, e);
}
```

Peor caso

$$T(n) = \begin{cases} 1 & n = 1 \\ 1 + T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

(1) $T(n) = 1 + T\left(\frac{n}{2}\right)$

(2) $= 2 + T\left(\frac{n}{2^2}\right)$

(k) $= k + T\left(\frac{n}{2^k}\right)$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log(n)$

$T(n) \in O(\log(n))$

Temas: Algoritmos Voraces

Problema mochila continua

→ Se permite fraccionar los objetos → el valor disminuye proporcionalmente a lo que cogemos

→ Maximizar la función objetivo

Ejemplo $P=12$ $p=(6,5,2)$ $v=(48,35,20)$ $\frac{v}{p}=(8,7,10)$

Criterios posibles

	Solución	Peso	Valor
1) Valor decreciente →	$(1,1,\frac{1}{2})$	12	93
2) Peso creciente →	$(\frac{5}{6},1,1)$	12	95
3) Valor específico decreciente $\left(\frac{v_i}{p_i}\right)$ →	$(1,\frac{4}{5},1)$	12	96

(↓ → Cuanto más valor mejor)

3) Valor específico: cuanto vale por unidad de peso

→ Tomada una decisión siempre se toma el mismo criterio

Formalización

$X=(x_0, x_1, x_2 \dots x_n)$ $x_i \in [0,1]$

$x_i=0$ → coger objeto i

$0<x_i<1$ → fraccionar

$x_i \geq 1$ → coger objeto completo

Función objetivo

$$\max\left(\sum_{i=1}^{n} x_i \cdot v_i\right)$$

valor transportado

restricción

↳ $\sum_{i=1}^{n} x_i \cdot p_i \leq P$ W

Algoritmo voraz → $O(N \log(n))$

master theorem

$T(n)=aT\left(\frac{n}{b}\right)+g(n)$

$$T(n)=\begin{cases}\Theta(n^k) & a<b^k \\ \Theta(n^k \log n) & a=b^k \\ \Theta n^{\log_b(a)} & a>b^k\end{cases}$$

reducción → $a=b=a$

$T(n)=aT\left(\frac{n}{a}\right)+g(n)$ $g(n) \in \Theta(n^k)$

$$T(n)\begin{cases}\Theta(n) & k<1 \checkmark \\ \Theta(n \log n) & k=1 \checkmark \\ \Theta(n^k) & k>1 \checkmark\end{cases}$$

Teorema: Algoritmo encuentra solución Óptima

$X = (x_1, x_2 \dots x_n)$ | $\sum_{i=1}^{n} x_i p_i = P$

Algoritmo voraces → greedy

Optimización por selección discreta

Problemas { satisfaga una restricción / Optimiza una función

1) Solución factible → cumple las restricciones del programa
2) Solución optima → solución factible que optimiza la función objetivo del problema

Pasos

~~1) Coje el un elemento i para añadir a la solución~~

2)

Ejemplos
- mochila discreta
- problema del cambio
- Árboles recubrimiento
- fontanero diligente
- Asignación de tareas

Problema Mochila discreta (sin fraccionamiento) → El método voraz no resuelve el problema

Se usa como una aproximación hacia el óptimo

Sea n objetos con valores V_i y pesos W_i y una mochila con capacidad máxima de transporte peso W. Seleccionar un conjunto de objetos de forma que:

- No sobrepasa W
- El valor transportado sea máximo

Valor transportado → $\max \left(\sum_{i=1}^{n} x_i v_i \right)$ (valor transportado)

Restricciones

$\sum_{i=1}^{n} x_i w_i \leq W \qquad x_i \in [0, 1]$

$x_i = 0$ → no selecciona el objeto i

$x_i = 1$ → si selecciona el objeto i

Problema Del cambio

→ Formar una suma N con el número mínimo de monedas tomadas con repetición de un conjunto C

→ Una solución es una secuencia de decisiones

$$S = (S_1, S_2, S_3 \dots S_n)$$

→ Función objetivo

$$\min(S)$$

→ Restricción

$$\sum_{i=1}^{n} \text{valor}(S_i) = N$$

→ solución voraz: tomar en cada momento la moneda de mayor valor posible

Árbol Recubrimiento Coste mínimo → Prim y Kruskal

Partimos de un grafo $g = (V, A)$

2 tipos de algoritmos: Prim → en ambos se van añadiendo arcos de uno en una solución
Kruskal

→ Diferencia → ~~foro~~ forma de elegir los arcos

Prim: Buscar el arco de mínimo peso que va de un vértice explorado a uno que no lo está
Añadir el arco a la solución y el vértice a los explorados

Complejidad → $O(n^2)$ $O(n^3)$

Algoritmo voraz → Sigue una estrategia consistente a elegir la opción local óptima en cada caso con la esperanza de llegar a una solución general óptima

De otra manera:

1) Descompone el problema a un conjunto de decisiones locales
2) Coges las más prometedoras

→ Si la solución de un problema, especialmente si se es de optimización

a) Hay numerosas alternativas

b) Búsqueda óptima de tomas decisiones

Identificación

1) Diseñar una solución recursiva al problema → top-down

2) Análisis → la complejidad temporal es prohibitiva

a) Subproblemas superpuestos

b) Reparto no equitativo de las tallas de los subproblemas

3) Si el problema es un problema de optimización, verificar que se puede establecer una estructura óptima

PD recursiva Vs iterativa

→ La complejidad temporal asintótica suele ser la misma

a) La versión recursiva puede ser más eficiente

→ resuelve los subproblemas que necesita

→ la versión iterativa ha de resolverse todos los subproblemas porqué no sabe los que necesitará más adelante

→ Las dos hacen uso del almacenamiento

→ Iterativa → menor complejidad espacial (depende del problema)

→ Recursiva → no es posible reducir el tamaño del almacenamiento

→ La programación dinámica no implica necesariamente una transformación a iterativa

Problemas clásicas para programación dinámica

- Mochila 0-1
- Cálculo número de fibonacci
- Problemas con cadenas
  - → Subsecuencia común máximo
  - → Distancia de edición entre dos cadenas
- Problemas de grafos

Coeficiente Binomial

Identidad de Pascal → $\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$; $\binom{n}{0} = \binom{n}{n} = 1$

Sol analítica → $\binom{n}{r} = \frac{n!}{r!(n-r)!}$

Complejidad temporal

$$T(n,r) = \begin{cases} 1 & r=0 \quad n=r \\ 1 + T(n-1,r-1) + T(n-1,r) & \text{otro caso} \end{cases}$$

→ Si hacemos uso de programación dinámica en vez de divide y vencerás → Cuando se incrementa la eficiencia

→ Problema recursiva → Divide y vencerás

→ Programación dinámica → se puede ahorrar cálculos guardando en un almacén

→ Programación dinámica / Dominios discretizables ya que las tablas han de ser indexadas
\ Disminuye drásticamente el número de subproblemas repetidos

→ Cota mochila discreta → valor mochila discreta mediante voraz

→ Casos de usar Divid y Venceras → principio de optimidad
↘ subproblemas son de tamaño diferente

→ Divide y vencerás → será más eficiente cuanto más equitativa sea la división en subproblemas

Voraz / Arbol, grafo no dirigido
/ Mochila discreta sin limitación
/ Asignación de tareas
/ Mochila continua (mejor)

Programación dinámica / Cadenas
/ fibo (menor coste)
/ ~~Búsqueda Binaria~~ Búsqueda Binaria

Divide y vencerás / fibonacci
/ Mochila sin fraccionamiento
\ Késimo

→ DR mejor en memoria que DI

Coste temporal tabas

$$T(n) = \begin{cases} 1 & \text{si } n = 0 \\ n + \sum_{j=0}^{n-1} T(j) & n > 0 \end{cases}$$

$$T(n-1) = n - 1 + \sum_{j=0}^{n-2} T(j) \rightarrow 2T(n-1) = n - 1 + \sum_{j=0}^{n-1} T(j)$$

$$T(n) = n + \sum_{j=0}^{n-1} T(j)$$

resolviendo $\sum_{j=0}^{n-2} T(j)$

$$T(n) = n + \sum_{j=0}^{n-2} T(j)$$

$$T(n-1) = n - 1 + \sum_{j=0}^{n-2} T(j)$$

$$\sum_{j=0}^{n-1} T(n-1)$$

$$2T(n-1) = n - 1 + n - 1 + \sum_{j=0}^{n-2} (j) + \sum_{j=0}^{n-2} (j) \rightarrow T(0) + T(1) + \cdots + T(n-2) + T(n-1)$$

$$\sum_{j=0}^{n-2} T(j) = T(n-1)$$

$$2T(n-1) = n - 1 + \sum_{j=0}^{n-1} T(j)$$

$$T(n) = n + 1 - 1 + \sum_{j=0}^{n-1} T(j) = 1 + 2T(n-1)$$

(1) $T(n) = 1 + 2T(n-1)$

(2) $= 1 + 2\left(1 + 2T(n-2)\right) = 1 + 2 + 4T(n-2) = 3 + 4T(n-2)$

(3) $= 3 + 4\left(1 + 2T(n-3)\right) = 7 + 8T(n-3)$

(k) $= (2^k - 1) + 2^k T(n-k)$

caso base

$n - k = 0$

$k = n$

$\rightarrow (2^n - 1) + 2^n T(0) \cdot 1 \in O(2^n)$

21/03/2024

Programación Dinámica

Punto de partida resolver PD → solución recursiva → ineficiente (naive)

↓ versión ingenua

$$Knapsack(0, c) = 0$$

$$Knapsack(n, c) = \max \begin{cases} Knapsack(n-1, c) \\ Knapsack(n_s - 1, c - p_n) + v_n \end{cases}$$

(n: ítems, c: peso)

→ solución matemática. Información relevante

1) Subproblemas
2) Parametros que caracterizan e identifican el subproblema
3) Dominio y recorrido de la función
4) Subproblemas del caso base
5) Orden el cual ha de resolverse los problemas en un algoritmo

Memoización → vector<vector<int>>

→ en el almacenamiento se hacen menos cálculos que la iterativa

→ se puede reducir el coste espacial

Decisiones óptimas

Conclusiones

1) La complejidad temporal es $\Theta(nW)$
2) Si W es grande → la solución iterativa no es eficiente
3) Complejidad espacial $\Theta(W)$
4) La solución de memoización y almacenamiento es más eficiente que la iterativa

Problema 2 → Cortando tubos / (longitud → n)

Longitud	1	2	3	4	5	6	7	8	9	10
precio	1	5	8	9	10	17	17	20	24	30

Espacial → $O(n)$

Temporal → $O(n^2)$

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i}) \qquad r_0 = 0$$

$$r_{10} = \max \begin{cases} p_9 + r_1 \\ p_2 + r_8 \\ p_3 + r_7 \\ \dots \\ p_{n+1} + r_{n-1} \end{cases}$$

Mejor caso → $T(n) \in \Omega(n)$

Peor caso → $T(n) = \begin{cases} 1 & n = 0 \\ 1 + 2T(n-1) & n > 0 \end{cases}$

$$(k)\quad T(n) = 2^k - 1 + 2^k T(n-k) \qquad n - k = 0,\ n = k$$

$$T(n) = 2^n - 1 + 2^n \in O(2^n)$$

Jaime

Soluciones Voraces

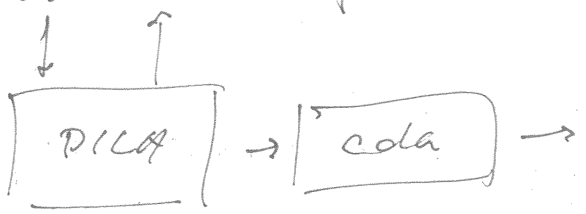
Busqueda ~~est~~ exhaustiva

~~Estrategia~~

estrategia busqueda

- LIFO → pila (cola prioridad)
- FIFO → cola
- LC (dirigido) → cola de prioridad

no es necesario usar la lista de nodos vivos



$f(x) = 3x \quad g(x) = \lfloor x/2 \rfloor \quad x = 3 \quad y = 6$

$(g \circ f \circ g \circ f \circ f \circ g)(3) = 6$

$(f \circ g \circ g \circ f)(3) = 6$

$3\left(\frac{\frac{3(3)}{2}}{2}\right) \rightarrow 3\left(\frac{4}{2}\right) = 6$

NV: (1)(1)(2,27)(2,4)

nodo, resultado

objetivo: 6

$f(9) = 27$

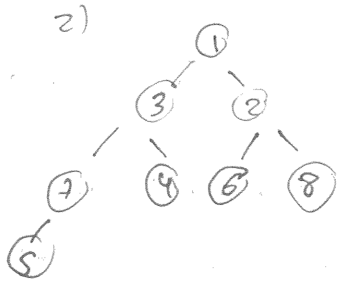
$g(9) = 4$

se hace mediante cola de prioridad

Esquema ramificación y Poda

1) Initial_node(p) → obtiene nodo inicial

2)



$f(n) = \{ \; 6(1)$

16/05/2024

Ramificación y Poda → Mejora Backtracking

2 tipos de cota:
- optimista → relajado
- pesimista → mediante método voraz

Hacer una estructura de Datos → cola de prioridad

NV → nodos vivos → nodos que pueden cambiar → solución parcial que vale la pena explorar

Cota → aproximación, no garantiza el óptimo

Ejemplo mochila
P = 16
p = (7, 8, 2)
v = (49, 40, 20)

NV: ¿? ? ? > 1 0 ? ?

1 1 1 → 109

Mejor sol: ~~60~~ 89

① 0 ? ? → 60
1 ? ? → 109
- 1 0 ? → 69
- 1 1 ? → 109
  - 1 1 0 → 89
  - ~~1 1 1 → 109~~ ← no factible

HEAP
~~? ? ?~~
0 ? ?
~~1 ? ?~~
1 0 ?
~~1 1 ?~~
1 1 0

using Node = tuple<(...)> → similar a un struct → puntero guarda el padre, ~~pun~~ altura, puntero (vector) hijos
priority_queue<node> pq.

break → rompe el bucle
continue → sigue a la siguiente iteración

Mochila:
- cota optimista → mochila continua
- cota pesimista → voraz de la mochila (mochila discreta)

Si k = v.size → nodo hoja

Cota pesimista parcial → permite mejorar la mejor solución por cada iteración
↳ A veces puede no ser fructífero debido al tiempo de su cálculo
→ Depende del problema de lo ajustado que está el problema y la eficiencia de su cálculo

Con la técnica Backtracking no resulta tan útil

optimo global = ~~140~~ 155 ↑ actualizar (solo)

0 ? ? ? → pesimista → (0, 1, 1, 1) = 155
→ optimista → (0, 1, 1, 1) = 155 > 155 (no prometedor) → no actualiza el optimo global ya que no es mejor
↳ indica que si se actualiza o se hace poda

1 ? ? ? → pesimista → (1, 1, 0, 0) = 140
→ optimista → (1, 1, 0, 4/5) = 172 > 150 (conti)

se sabe que será 155 ó ⑦

No igualar pesimista y optimista

si optimista <= pesimista → no prometedor

Práctica examen final

master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + g(n) \qquad g(n) \in \Theta(n^k)$$

$$T(n) \begin{cases} \Theta(n^k) & n < b^k \\ \Theta(n^k \log_2(n)) & n = b^k \\ \Theta(n^{\log_b(n)}) & n > b^k \end{cases}$$

$$T(n) = aT\left(\frac{n}{a}\right) + g(n) \qquad b = a$$

$$T(n) = \begin{cases} \Theta(n) & k < 1 \\ \Theta(n \log n) & k = 1 \\ \Theta(n^k) & k > 1 \end{cases}$$