Alg. y Complejidad - TI

① **Algoritmo**: descripción abstracta y ordenada de todas las acciones a realizar y datos a utilizar para solucionar un problema.
Además debe { • Tener un nº finito de pasos ordenados.
• Funcionar correctamente para todos los casos
• Funcionar siempre igual para unos mismos inputs.

② **Algoritmia**: estudia la eficiencia de los algoritmos de manera sistemática y trata de buscar el mejor algoritmo y sus propiedades.

③ **Eficiencia y principio de invariancia**
Notación: $T(n)$ { T = función tiempo, n = tam. input } Ppio inv.: dos implementaciones de un alg. no se diferencian en eficiencia más que en una cte. $t_1(n) \leq c\,t_2(n)$

④ **Notación asintótica**
Nos interesa saber cómo se comporta el algoritmo respecto al tpo. En el peor de los casos.
Notación: $O(f(n))$ { $f(n)$ → tasa de crecimiento en el peor caso. $O(n)$ → complejidad asintótica.

⑤ **Regla de la suma**
$T_1(n) \rightleftharpoons f(n)$, $T_2(n) \rightleftharpoons g(n)$ } $T_1(n) + T_2(n) \rightleftharpoons O(\max(f(n), g(n)))$
La complejidad está determinada por la parte más costosa.

⑥ **Regla del producto**
$T_1(n) \rightleftharpoons f(n)$, $T_2(n) \rightleftharpoons g(n)$ } $T_1(n) \cdot T_2(n) \rightleftharpoons O(f(n) \cdot g(n))$; Por tanto $O(c \cdot f(n)) = O(f(n))$ (c: cte)

⑦ **Cálculo eficiencia**
Op. básicas: sumas, asignaciones, etc.
Decisiones: seleccionar caso más costoso.
Bucles: nº {b de dentro del bucle}

Ejemplos

1) $T(n) = x^n$
2) Pasar a izq. y der
3) Sacar fact. común
4) Homogénea: raíces $\{A \cdot \text{raíz}^n + B \cdot \text{raíz}^n \dots\}$
5) Particular
6) Partic. + Homo

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

1) $x^n = 1 + n + x^{n-1}$
2) $x^n - x^{n-1} = 1 + n$
3) $x^{n-1}(x-1) = 1 + n$
4) $x^{n-1}(x-1) = 0 ; x = 1 ; x^{(H)} = A \cdot 1^n$
5) $x^{n-1}(x-1) = 1 + n ; x^{(P)} = (B + Cn)n$
6) $x^{(H)} + x^{(P)} = A \cdot 1^n + Bn + Cn^2 = A + Bn + Cn^2 \; ; \; O(n^2)$

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

1) $x^n = 4n + 3 + x^{n-1} + 2x^{n-2}$
2) $x^n - x^{n-1} - 2x^{n-2} = 4n + 3$
3) $x^{n-2}(-2 - x + x^2) = 4n + 3$
4) $x^{n-2} \cdot (-2 - x + x^2) = 0$; n raíces $\begin{cases} 1 \\ +2 \end{cases} \Rightarrow A(-1)^n + B \cdot (+2)^n = x^{(H)}$
5) $x^{n-2}(2 - x - x^2) = 4n + 3 ; x^{(P)} = (C + Dn) = x^{(P)}$
6) $x = x^{(P)} + x^{(H)} = A(-1)^n B \cdot (+2)^n + C + Dn$

el más capo $\Rightarrow O(2^n)$

(3) 1) $T(n) = 2T(n-1) - T(n-2) + 8$

3) $x^n(x^2 - 2x + 1) = 8$
4) $x^2 - 2x + 1 = 0 ; x \begin{cases} 1 \\ 1 \end{cases} \; A \cdot (1)^n + B \cdot n \cdot (1)^n = x^{(H)}$
5) $x^{(P)} = (C) \cdot n^2$
↳ n^0 y n^1 ya están cogidos
6) $x = x^{(H)} + x^{(P)} = A + Bn + Cn^2 \; ; \; O(n^2)$

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

3) $x^{n-2}(x^2 - 4) = n + 4 ; x^2 - 4 = 0 \begin{cases} 2 \\ -2 \end{cases} \; x^{(H)} = A \cdot 2^n + B \cdot (-2)^n$
4) $x^{(P)} \Rightarrow Cn + D$
5) $x = x^{(H)} + x^{(P)} = A \cdot 2^n + B \cdot (-2)^n + Cn + D \; ; \; O(2^n)$

(5) $T(n) = T(n-1) + 3n ;$

$x^{n-1}(x-1) = 3n ; x^{(H)} = A \cdot (1)^n$
$x^{(P)} = (B + Cn)n$
$x = A + Bn + Cn^2$

Algoritmia - Clase 1

- Principio de Invariancia

Eficiencia de algoritmo 1 = N · Eficiencia de algoritmo 2
(cte)

T(n) → tpo. ejecución para un input de tam. n.

- Notación asintótica

Nos importa un n grande

T(0) = 1
T(1) = 4
$T(n) = (n+1)^2 = n^2 + 1^2 + 2n$
Como nos importa $n \to \infty$, descartamos las ctes. y los inferiores de la función.
Es decir, cogemos el n más fuerte → $O(n^2)$
$(n+1)^2 = n^2 + 2n + 1 \leq n^2 \Rightarrow \underline{O(n^2)}$
La complejidad es la variable que más pesa.

- Regla de la adición

$T1(n) \Rightarrow O(f(n))$
$T2(n) \Rightarrow O(g(n))$
$\Big\} \; T1(n) + T2(n) \Rightarrow O(\max(f(n), g(n)))$

- Regla del producto

$T_1(n) \Rightarrow O(f(n))$
$T_2(n) \Rightarrow O(g(n))$
$\Big\} \; T1(n) \cdot T2(n) \Rightarrow O(f(n) \cdot g(n))$

$O(\varepsilon f(n)) \approx O(f(n))$
Ejemplo: $O(n^2/2) \approx O(n^2)$

- Op. elementales:
  - Aritméticas
  - Lógicas
  - Comparación
  - I/O
  - Asignación inicial
  - Returns

A = 2 + 3
B = ({[4 × 5] · Log(23)}² = 42) OR (C = 2 × 4)
} La complejidad asintótica es la misma.

S: T(A) = 1, T(B) = 4 · T(A)
T(B) = 4
⇓
$n_1 = 4 \; n_2 \approx n_1 = n_2$

La complejidad de op. elementales es siempre O(1)
(Aunque sean combinadas)

- Condicionales

If Cond then InstT else InstF eif

Coste (Cond) + Max (Coste(InsT), Coste (InstF)) = Coste - condicional

- Bucles

- Eficiencia recursiva
  - Método de la ec. característica

Ejemplo 1

```
for i:=1 to n do:
    for j:=1 to n do:
        A[i,j]:=0
```

O = Max(O1, O2) = Max(n², n) = n² ; O(n²)

Bucle 1 / Bucle 2

$$T(n) = \sum_{i=1}^{n} \cdot \sum_{j=1}^{n} \cdot 1 = n \cdot n \cdot 1 = n^2 \; ; \; O(n^2)$$

Ejemplo 2

```
for j:=i+1 to n do
    If A[j] < A[chico] then
        chico:=j
```

Coste condicional: { False: nada ; true: op. el. asignación }

$$T(n) = \sum_{j=i+1}^{n} \cdot (1 + Max(nada, 1)) = (n - i - 1) \cdot 2 \; ; \; O(n)$$

($n-(i+1)$ iteraciones)

$$T(n) = \sum_{j=1}^{n} \cdot (1+1) = 2n \; ; \; O(n)$$

Ejemplo 3

```
If A[1,1]=0 then:
    For i:=1 to n do:
        For j:=1 to n do.
            A[i,j]:=0
Else:
    For i:=1 to n do
        A[i,i]:=1
```

$$T(n) = 1 + Max\left[\left(\sum_{i=1}^{n} \cdot \sum_{j=1}^{n} \cdot 1\right), \left(\sum_{i=1}^{n} \cdot 1\right)\right] = 1 + n^2 \; ; \; O(n^2)$$

Ejemplo 4

$$T(n) = \sum_{i=1}^{n} \left(1 + \sum_{j=1}^{n} \cdot (1+1) + 3\right) = n(2n+4) = 2n^2 + 4n \; ; \; O(n^2)$$

Ejemplo 5: recursividad

```
const dim=...
tipos vector = array[1..dim] de enteros
proc Ej3(n=entero, V=vector)
var i:= entero
Si (n>0) y (n<=dim) entonces
    Desde i:=1 hasta (n-1):
        V[i] = V[i+1]
    fdesde
    Ej3(n-1, V)
fs:
```

coste loop + llam recursiva

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

$T(n) \equiv x^n$;

$x^n = 1 + n + x^{n-1} = 1 + n \Rightarrow$

$x^{n-1}(x-1) = 1 + n$

Pte. homo: $x^{n-1}(x-1) = 0 \xrightarrow{\text{roots}} x^{(H)}$ ($x=1$ ~~x=0~~ no importa: no da soluciones)

$x^{(H)} = A \cdot 1^n = A$ (A: cte, 1: root)

Particular:

$x^{n-1}(x-1) = 1 + n \Rightarrow x^{(P)} = (B + Cn)$

8 Alg. y complejidad - Clase 1

Ef. en prog. recursivas
Método de ec. característica → F. generatrices.
↑ Recursiva

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

$T(n) \equiv x^n$

$x^n = 1 + n \cdot x^{n-1} \Rightarrow x^n - x^{n-1} = 1+n \Rightarrow x^{n-1}(x-1) = 1+n$

HOMO: $x^{n-1}(x-1) = 0 \Rightarrow x^{(H)} = A \cdot 1^n = A$

PARTICULAR: $x^{n-1}(x-1) = 1+n \Rightarrow x^{(P)} = (B+Cn)n$

$\left.\begin{array}{l} \\ \\ \end{array}\right\} x = x^{(H)} + x^{(P)} = A + Bn + Cn^2 \Rightarrow O(n^2)$

Ej.: $T(n) = 4n+3 + T(n-1) + 2T(n-2)$;

$T(n) \equiv x^n$

$x^n = 4n+3 + x^{n-1} + 2x^{n-2}$

$x^n - x^{n-1} - 2x^{n-2} = 4n+3$

$x^{n-2}(x^2 - x - 2) = 4n+3$

HOMO: $x^{n-2}(x^2 - x - 2) = 0$; Roots $\left\{\begin{array}{l} 2 \\ -1 \end{array}\right. \Rightarrow x^{(H)} = A \cdot 2^n + B(-1)^n$

PARTI: $x^{n-2}(x^2 - x - 2) = 4n+3$; $x^{(P)} = (C + Dn)$

$x = x^{(H)} + x^{(P)} = A \cdot 2^n + B(-1)^n + C + Dn = T(n)$; $O(2^n)$

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

$n \equiv 2^k$

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

$T(2^k) = T(2^{k-1}) + 4^k + 3 \cdot 2^k$

$T(2^k) \equiv x^k$

$x^k = x^{k-1} + 4^k + 3 \cdot 2^k$

$x^k - x^{k-1} = 4^k + 3 \cdot 2^k$

HOM: $x^k(x-1) = 0$, Root = 1; $x^{(H)} = A$

PAR: $x^k(x-1) = 4^k + 3 \cdot 2^k \Rightarrow x^{(P)} = B \cdot 4^k + C \cdot 2^k$

$\left.\begin{array}{l} x = A + B \cdot 4^k + C \cdot 2^k \\ = A + B n^2 + Cn \end{array}\right\} \Rightarrow O(n^2)$

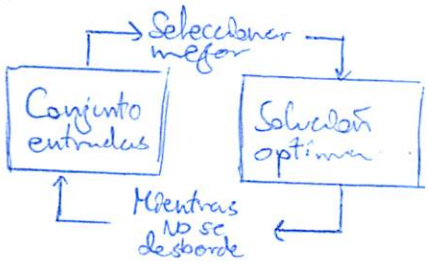
$T(n/3)$; $n \equiv 3^k$ ...

$\frac{b^k}{5} = b^{k-5}$

Alg. y Complejidad - T2

Algoritmo voraz: estructura de algoritmo de optimización.
Entrada: conjunto de n elementos
Salida: subconjunto de la entrada que optimiza una func. objetivo.

Estructura:



① Ej. 1: Problema cambio de monedas

Dadas los tipos de monedas = [100, 25, 10, 5, 1], func. que devuelve el cambio n exacto con el menor nº de monedas:

```
selección (tope, posibles, tomados):
    mejor_candidato = null
    for elemento, in posibles_elementos:
        if tope >= elemento + Σtomados y elemento > mejor:
            mejor_candidato = elemento
    return mejor_candidato
```

```
optimizar_vueltas (n):
    posibles = [100, 25, 10, 5, 1]
    solucion = [∅]
    suma = 0
    mientras suma ≠ n:
        suma = 0
        for elem in (solucion):
            suma += elem
        solucion += selección (n, posibles, suma)
```

Ej's alg. voraces:

(1) · n ← {nº's naturales} Es par.
Se toman parejas y se suman los nºs. Se busca el máximo sumando, y el menor de ellas.

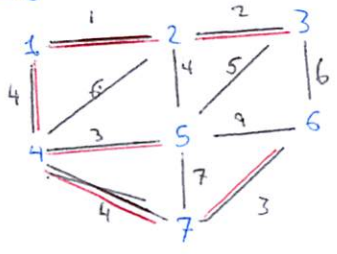
Ejemplo
n { 5, 8, 1, 4, 7, 9 }

Tomando seguidos	Tomando alternos:	Mejor opción: ordenar vector y tomar opuestos:
5+8 = 13	5+9 = 14	n = {1, 4, 5, 7, 8, 9}
1+4 = 5	8+7 = 15 ←	1+9 = 10
7+9 = 16 ←	1+4 = 5	4+8 = 12 } 12
		5+7 = 12

Algoritmos voraces en árboles

Árboles de recubrimiento mínimo:
Objetivo: dado un grafo, devolver ato con las conexiones mínimas para optimizar todos los recorridos.

Algoritmo de Kruscal



Seleccionar la arista que:
- Sea de menor peso seleccionado
- No haya sido seleccionada
- No cree un ciclo

1º: selecciono la más barata: (1,2)
2º: selecciono más barata: (2,3)
3º: (4,5)
4º: (7,6)
5º: (1,4)
6º: (5,2) → No, crea bucle
7º: (4,7) → todos unidos

H ← nº nodos
A ← nº aristas
A ← cjto. aristas
N ← cjto. nodos

1) Ordenar A : O(a log n)
2) Iniciar n cjtos: O(n)
3) Buscar y fusionar: O(a log n)
4) Demás operaciones: O(a)

} T(n) = O(a log n)

Master theorem

k ← nº veces q. se llama a la recursividad
n ← tamaño problema
b ← nº de subcasos (Log. donde a $T(n/b)$)
p ← exp. del t. independiente si es n. (Si es una cte, p=0).

$k = b^p$;

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

Algoritmia y complejidad - Clase 3

1) Algoritmos voraces:

Cjto. candidatos: todos los tipos de partes de solución
Solución: elementos elegidos
Completable: define el límite, el final del problema, cuándo termina.
Función de selección: selecciona opciones óptimas
Función objetivo: nº de elementos de la solución

Ejemplo: problema de la mochila

Tengo n objetos de w_i peso y v_i valor.
Los objetos se pueden dividir: [0...1]. n
Para un peso W, llevar el mayor valor posible tal que $\sum w_i \leq W$
Sol.: mientras no supere W meto el mejor objeto restante. (usando func. de selección).
Cuando supero el peso, meto la parte proporcional de la mejor opción. ([0...1] * mayor v_i)
¿Cuál es el mejor elemento restante? (Func. de selección).
Supongamos:

	1	2	3	4	5
valor	20	30	66	40	60
peso	10	20	30	40	50
renta	20/10	30/20	66/30	40/40	60/50

Métodos de selección:
- ¿Más valioso?
- ¿Más ligero?
- ¿Más rentable? → Este es el bueno.

Busco sacar la mayor rentabilidad para los W Kg.

El problema de los problemas voraces no es el algoritmo, sino la función de selección.

Cálculo del ritmo de crecimiento:

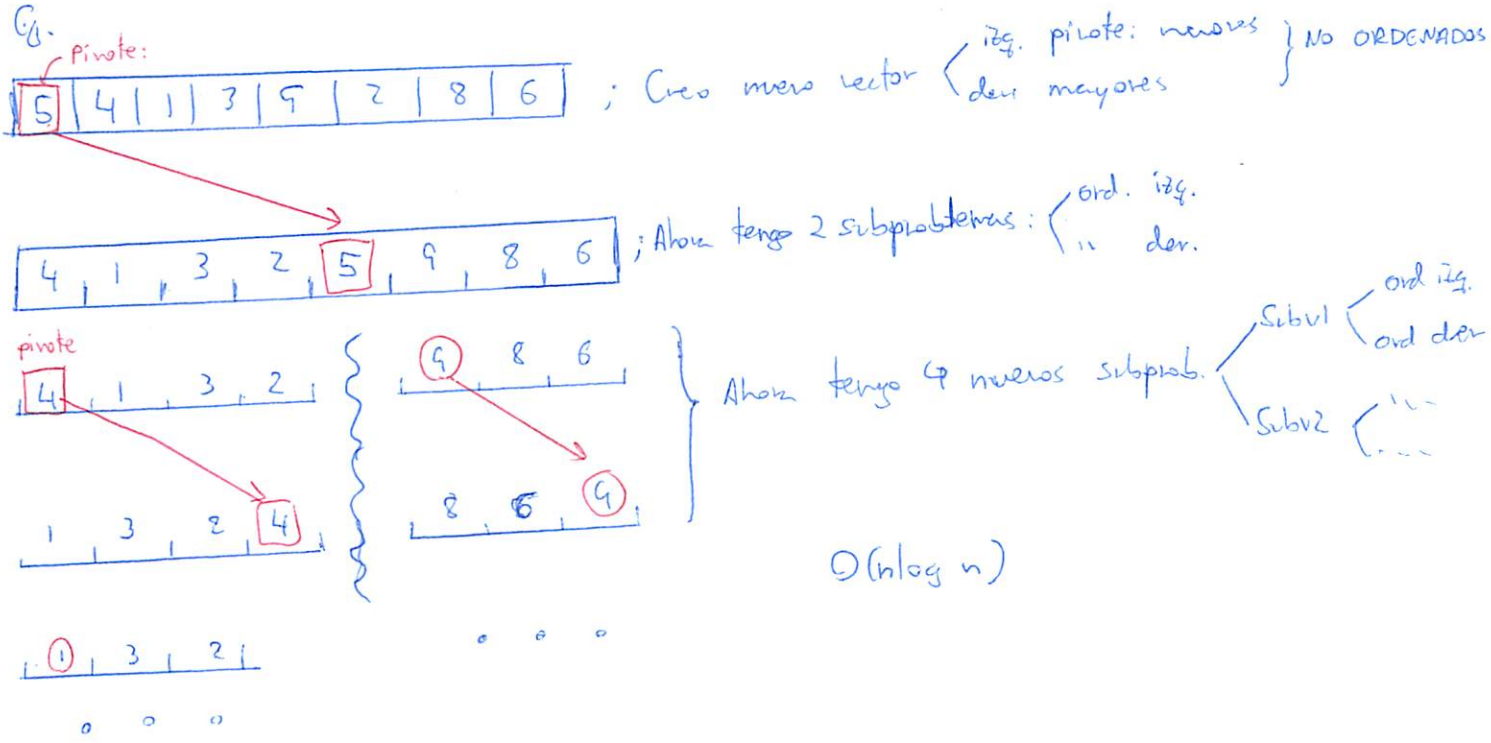
Implementación directa:
Bucle de comprobar objetos + Bucle de recorrer objetos en f. selección: $O(n) \cdot O(n) = O(n^2)$
Implementación preordenando objetos:
Bucle de comprobar objetos + bucle seleccionar obj. f. selección: $O(n) \cdot O(\log_2 n) = O(n \log n)$
$\in (\text{Max}(O(n), O(1), O(n), O(\log_2 n))$

- Divide y vencerás

Quicksort

Pivote en medio: ordena menores izq y luego derecha.

Ej.



Potenciación de enteros

$x^{25} \Rightarrow (((x^2 \cdot x)^2)^2)^2 \cdot x$ } Desplazamiento de bits.

Coste: $O(m^{\log_3 n^2})$

Mult. de Matrices

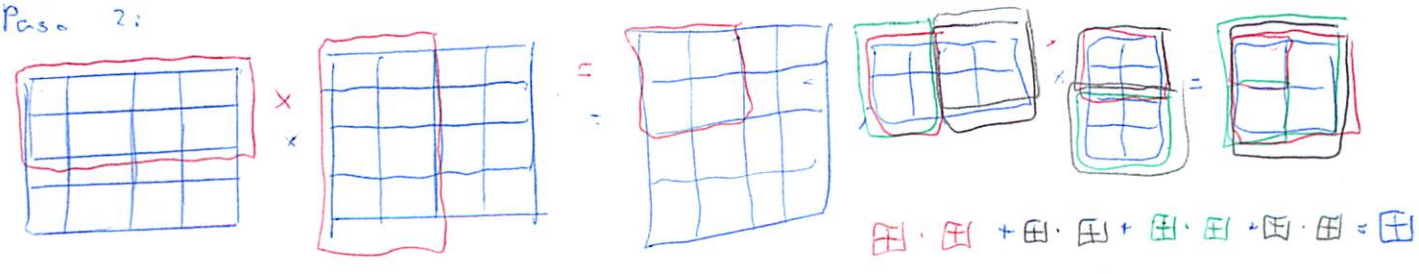
$A_{n \times n}$, $B_{n \times n}$ } $A \times B_{n \times n}$

Si n: { pot. 2 ✓ / no es pot. 2: añadir filas. columnas hasta q. sea pot 2.

Si no es pot. 2: ej 2x3. Aproximo a 4x4:

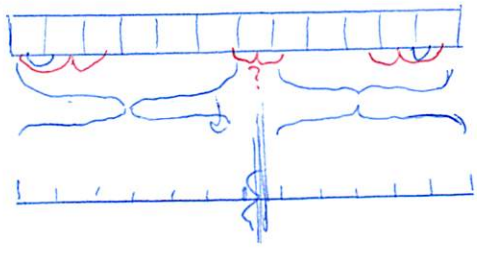
1	2	3	0
4	6	7	0
0	0	0	0
0	0	0	0

Paso 2:



- Eq. 5 dim y vencejos

Empareje de 2 en dos: con el de delante o el de detrás, tal q. la suma de puntos sea el máximo.



Max (mejor (p, m, v), mejor (m, fin, vector))

$2T\left(\frac{n}{2}\right)$ ← sobre la mitad del vector

T
2 llamadas recursivas

$p=0$
$b=2$
$k=2$

$k > b^p$; $T(n) \in O(n^{\log_b k}) = n^{\log_2 2} = n$

O (n)

[1, 2, 3, 5, 7, 9]

Tam: 18: 1, 2, 3, 5, 7

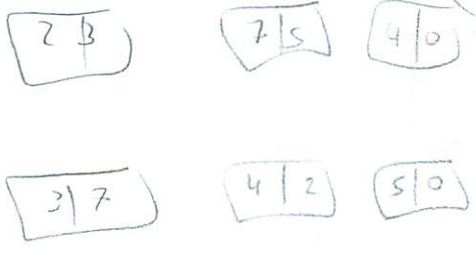
Acum: 1 + [1+2] + [1+2+3] + [1+2+3+5] + [1+2+3+5+7]

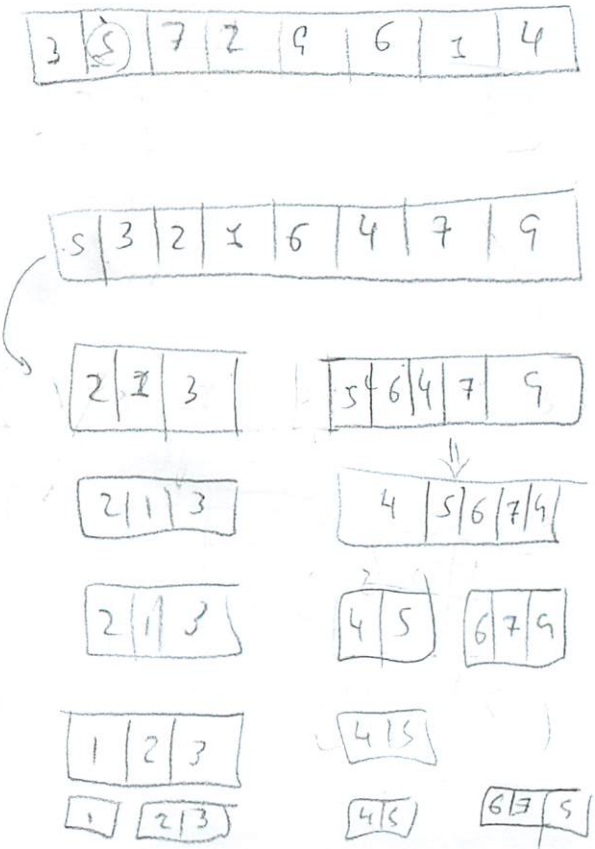
Acum = 2 * Acum + nuevo

Tam bus.: ~~7~~
1+2+3+5

5+2

[7, 3, 1, 4, 2]





T3: Programación dinámica

Problemas de optimización mediante secuencias de decisiones.
↳ Produciremos varias secuencias de decisiones y tomaremos la mejor de ellas.
↳ Empezamos de casos pequeños, y llegaremos al caso final.
Ppio de optimidad: "una solución es óptima si sus subsoluciones lo son"

Ejemplo
Problema de la mochila: llenar la mochila con objetos, maximizando el valor posible.

Maximizar $\sum x_i \cdot v_i$ con restricción $\sum_{i=1}^{n} x_i \cdot w_i \leq W$

① Generamos una tabla donde se estudian todas las posibilidades que tengo.

X	Límite de peso
Nº de elementos	Valor de los elementos introducidos

① ¿Hay algún objeto con peso igual y mejor val?
② ¿Cómo relleno el hueco?

Objetos consumibles: solo puedo tomar 1.

Para el caso con $W_{max} = 11$:

Límite:	0	1	2	3	4	5	6	7	8	9	10	11	(Solo obj 1)
$w_1=1$ $v_1=1$	0	1	1	1	1	1	1	1	1	1	1	1	(Obj 1 y 2)
$w_2=2$ $v_2=6$	0	1	6	7	7	7	7	7	7	7	7	7	(Obj 1, 2, 3)
$w_3=5$ $v_3=18$	0	1	6	7	7	18	19	24	25	25	25	25	...
$w_4=6$ $v_4=22$	0	1	6	7	7	18	22	23	28	29	29	40	
$w_5=7$ $v_5=28$	0	1	6	7	7	18	22	28	29	34	35	40	
w_6													

② Razonamos qué hacemos para obtener la mejor sol. para cada caso:
1. ¿Hay algún objeto con peso igual y mayor/menor valor? (es menor) cuyo $w_{lim} = (w_{lim} - w_i)$.
2. Sí: lo tomo y me voy al caso cuyo $w_{lim} = (w_{lim} - w_i)$.
3. No: me quedo con el caso de la fila sup.

③ Traduzco a código:
Celda = max(V[f-1][c], V[f-1][c-w_i])
f = filas
c = columnas
↑3. 2. ↗

Ejemplo 2: muchos ejs. implican el cálculo del mismo resultado varias veces.

Por ej: $\binom{n}{k} = \begin{cases} 1 & \text{si } n \text{ ó } k = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} \end{cases}$

C(4,4) → C(3,3), C(4,3)
C(3,3) → C(2,2), C(2,3)
C(4,3) → C(3,2), C(3,3)
...

Podemos utilizar el triángulo de pascal y es más eficiente

	0	1	2	———	n
0	1	0			
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	3	5	3	1
n					

Ej. 1

Matriz de F x C:

3	2	3	2	5
1	2	4	5	6
2	3	6	2	4
4	5	3	4	3

Sólo puedo { Av. derecha / Av. abajo.

Se busca llegar de [1,1] a [F,C] maximizando el coste. (coste de casillas = int dentro de estas)

Técnica: se usa una matriz de correspondencia C, cuyas casillas tienen el coste acumulado de llegar hasta ellas.

Ej.

M

5	2	6	4
3	3	8	1
2	7	4	8

→

C

5	7	13	17
8	10	18	18
10	17	21	26

Para

5	2
3	3

⇒

5	7
8	?

→ Min [v[i-1,j-1] + v[i,j-1], v[i-1,j-1] + v[i-1,j]]

Sol. C:

3	5	8	10	15
4	6	10	15	21
8	9	15	17	21
10	14	17	21	24

Ej. 2

Dos equipos (A, B) juegan 2n-1 partidas. Ganador = el que gane n partidas.

p = prob de q. gane A

q = 1 - p = prob. de q. gane B

P(i,j) = prob. de que gane A necesitando ganar i partidas y B j partidas.

Si A gana, P(0,0) = 1

Si B gana, P(i,0) = 0

Al comienzo, P(n,n)

P(i,j) = p · P(0,j) + q · P(i,0)

P(i,j)

P(i-1,j) P(i,j-1)

P(i-2,j) P(i-1,j-1) P(i,j-2)

Repe

En lugar de calcular recursiv., calculo todos los posibles valores intermedios 1 sola vez.

Ej. 3

Cambio de vueltas.

El alg. voraz pinchará: ej. dev. 8 con monedas de 1, 4, 6 { Voraz: 6+1+1 / Óptimo: 4+4

Versión progr. dinámica:

Cantidad	0	1	2	3	4	5	6	7	8
$d_1 = 1$	0	1	2	3	4	5	6	7	8
$d_2 = 4$	0	1	2	3	1	2	3	4	2
$d_3 = 6$	0	1	2	3	4	5	6	7	8

Dev 6 con monedas de 1 y de 4.

Min [c[i-1,j], 1 + c[i, j-d_i]]

↑ No añado moneda nueva

↑ Añado nueva moneda y veo cuánto me hace falta: cantidad - nueva = lo q. falta

Min [6, 1+2->]

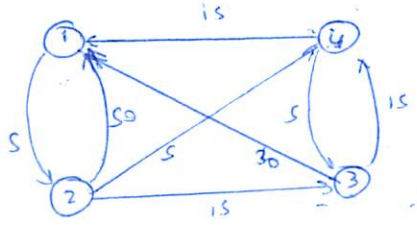
nº mínimo de monedas si tengo 1's

quedarme como estaba (no añado monedas)

Ej. 7 · Caminos mínimos

Alg. Floyd:

1) Matriz dist. directas



1)

de \ a	1	2	3	4
1	0	5	∞	∞
2	50	0	15	5
3	30	∞	0	15
4	15	∞	5	0

2) Considero el nodo 1 como nodo de paso:

	1	2	3	4
1	0	5	∞	∞
2	50	0	15	5
3	30	35	0	15
4	15	15+5	5	0

3) Considero (2) como nodo de paso:

	1	2	3	4
1	0	5	5+15	5+5
2	50	0	15	5
3	30	35	0	15
4	15	20	5	0

4) Considero (3) ....

	1	2	3	4
1	0	5	20	10
2	15+30	0	15	5
3	30	35	0	15
4	15	20	5	0

5) Considero (4) como nodo de paso

	1	2	3	4
1	0	5	20	10
2	5+15	0	5+5	5
3	30	35	0	15
4	15	20	5	0

T5 - Backtracking

Técnica: buscar exhaustivamente soluciones entre todas las posibilidades.

Nodos → Situación en t_i (soluciones parciales)

Aristas → Decisiones

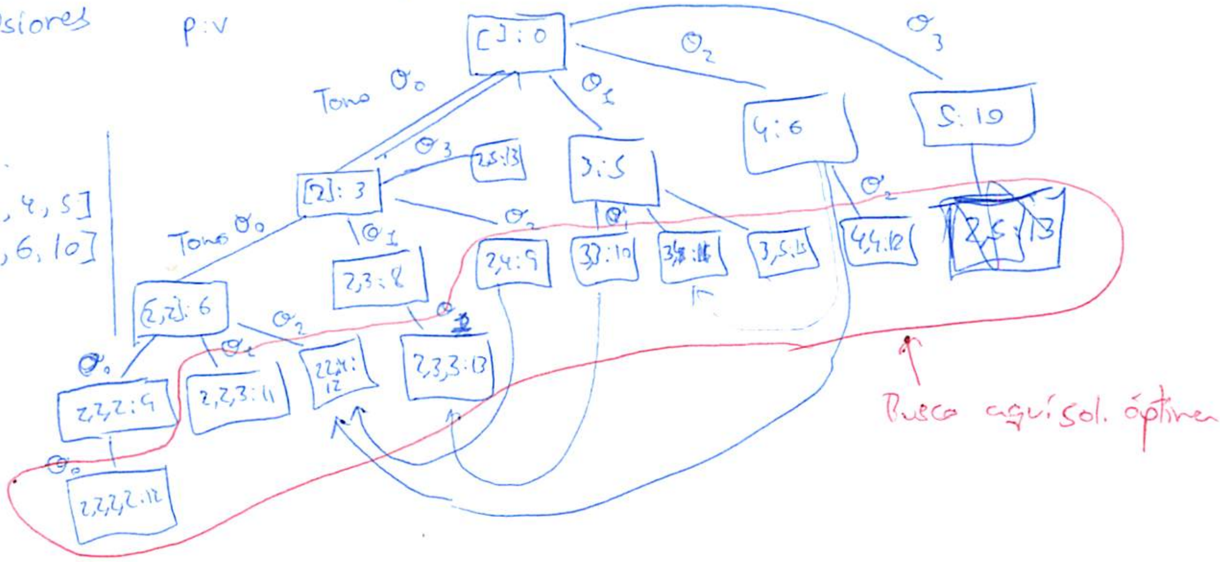
Ejemplo

Probl. mochila.

pesos = [2, 3, 4, 5]

valores = [3, 5, 6, 10]

W = 8



Ramificación y poda

① Cota inf.: max (Σ min(filas), Σ min(columnas)) Cualq. solución real posible.

② Cota sup: diagonal, diagonal inv, ... Cualq. solución real posible.

③ Rango val. admisibles = [cota inf, cota sup]

④ Mientras queden nodos sin explorar:

- ① Tomar todas las posibles opciones (menos de la que se parte) de la sig. fila.
  Estimar el mejor coste incal tomando el mejor opción de las filas restantes.
- ② Para cada una, estimar el mejor coste (2.1) Si alguna estimación supera la cota superior, eliminar dicha rama.
- (2.1) Si alguna estimación supera la cota superior, eliminar dicha rama.
- ③ Para la mejor de las estimaciones:
  - (3.1) Si no es hoja, volver al paso 4.1
  - (3.2) Si sí es hoja:
    - solución real ⇐ hoja. si hoja < solución real
    - cota superior ⇐ hoja si hoja < cota superior.
    - Seguir explorando por nivel (padre con estimación ≤ hoja) - 1

T6: Alg. no deterministas

Dos casos { Montecarlo: siempre da solución pero no siempre es correcta.
Vegas: puede fallar pero si da sol. es la correcta

Montecarlo

Partimos de tomar una muestra aleatoria y ver si satisface la condición, en lugar de comprobar todos los elementos de la población. Para ↑ prob. de que el output sea real (P_A), podemos ejecutar el algoritmo muchas veces. (Es decir, probar con varias muestras aleatorias de la población).

Ejemplo

Dado un vector, ver si un elem aparece más de 1/4 de las veces.
¿Cuántas veces hay q. ejecutar para q. $P_A \geq 90\%$?

2 casos { No hay elem rep. > 1/4 veces; $P_A = 1$.
Sí hay elem rep > 1/4 veces: tomando muestra aleatoria { $P_A = [p(\text{muestra se rep} > 1/4] = 1/4$; $P_F = -P_A + 1 = 3/4$

Para que $P_A \geq \frac{9}{16}$; $P_F \leq \frac{1}{10}$;

$(P_F)^n = \left(\frac{3}{4}\right)^n$; $\left(\frac{3}{4}\right)^{n^\circ \text{ veces}} \leq \frac{1}{10}$; $n = 9$

$n \in \mathbb{N}$

Vegas

Se ejecuta hasta que da solución correcta

$T.\text{vegas} = p(x) \cdot s(x) + (1 - p(x)) \cdot (f(x) + t(x))$

$T.\text{vegas} = s(x) + \frac{1 - p(x)}{p(x)} \cdot f(x)$ ⇕ $t(x)$

$t(x)$ = tpo. esperado total
$p(x)$ = prob. éxito
$s(x)$ = tpo. esperado de éxito
$f(x)$ = tpo. esperado de fracaso

Ejemplo

Colón desc. América:

$p(x) = 80\%$
$s(x) = 10s$
$t(x) = 60s$

$T.\text{vegas} = s(x) + \frac{1 - p(x)}{p(x)} \cdot f(x)$

$60 = \left(10 + \frac{1 - 0'8}{0'8}\right) \cdot f(x)$;

$f(x) = \frac{60}{\left(10 + \frac{1-0'8}{0'8}\right)} = \frac{60'}{10'25} = 5'85$

$50 = \frac{0'2}{0'8} f(i)$; $f(i) = \frac{50}{}$