# ResumenAlgoryedfinalParcial3.pdf

Anónimo

Algoritmia y Estructuras de Datos Avanzadas

2º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

ALGORYED

# PARCIAL 3

## 5.7. GAD + ORDENACIÓN TOPOLÓGICA

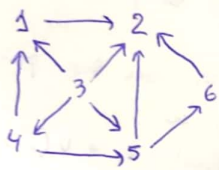
- GAD: Grafo Acíclico Dirigido
  - ↳ Proposición: G es GAD ⇔ no hay aristas ascendentes en G
- → Usamos DFS para detectar ciclos en grafos
- ORDENACIÓN TOPOLÓGICA (TS): (en un GAD) es cualquier orden total de sus vértices

$(u, v) \in E \Rightarrow u \leq v$

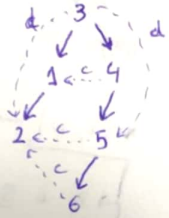
Método para obtenerla:

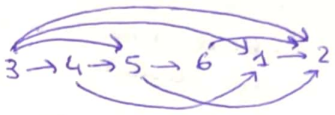
1. Aplicar DFS empezando en el nodo que no tiene ningún vértice hacia él. (inc[v] = 0) (si es GAD, siempre hay uno)
2. Añadimos el vértice al principio de la lista enlazada cuando DFS le finalice (nodo que terminemos le metemos al inicio)

Ejemplo:




d f Empezamos en el 3
2 5 1 - 2
6 3 4 2 -
1 12 3 - 1 - 2 - 4 - 5
8 11 4 - 1 - 5
7 10 5 - 2 - 6
8 9 6 - 2




TS: [3 4 5 6 1 2]

# TEMA 6: PROGRAMACIÓN DINÁMICA

## 6.1. PROBLEMA DEL CAMBIO

Input: · lista con los valores monedas
· cambio

Output: Diccionario con cuantas monedas de cada

Antes, hemos visto ALGORITMO CODICIOSO →

```
def change(c, l):
    d_change = {}
    for coin in sorted(l)[::-1]:
        d_change[coin] = c // coin
        c = c % coin
    return d_change
```

↓

Pero FALLA. Ejemplo: { cambio de 7, l = [1, 3, 4, 5] }

Nuevo enfoque →

**PROGRAMACIÓN DINÁMICA**

- Definimos $n(i, c)$: nº mínimo de monedas para cambiar c usando las primeras i monedas (l[:i])

$n(i, 0) = 0$

Es decir: $n(1, c) = c$

- → Si l[i] NO entra en el cambio → $n(i, c) = n(i-1, c)$
- → Si l[i] SÍ entra en el cambio → $n(i, c) = 1 + n(i, c - l[i])$

$$n(i, c) = \min\{n(i-1, c),\ 1 + n(i, c - v_i)\} \quad \text{si } v_i \leq c \leq C$$

$$= n(i-1, c) \quad \text{si } c < v_i$$

← dijo el mínimo entre coger esa moneda o no.

← si la moneda es muy grande (más que el cambio, no la cojo)

Entonces → Creamos Matriz $N \times C$ ⇒ coste $O(N \times C) = O(N \times 2^{\lg C})$

$\text{len}(c)$

↳ La solución de $M(N, C)$ es la esquina inferior derecha de la matriz.

- Coste: lineal en N
  exponencial en C

EJEMPLO: Cambio $c = 7$ ; $c = [1, 3, 4, 5]$

| c | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 3 | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 |
| 4 | 0 | 1 | 2 | 1 | 1 | 2 | 2 | 2 |
| 5 | 0 | 1 | 2 | 1 | 1 | 1 | 2 | (2) |

$(n(i,0) = 0 \quad n(1,c) = c)$

$n(i,c) = \min\{n(i-1,c), n(i,c-v_i) + 1\}$

$= n(i-1,c)$

1 de 3 y 1 de 4

## 6.2. El Problema de la Mochila → (Problema de Optimización con Restricciones).

(MOCHILA 0-1)
PROBLEMA: N elementos $(i_1, \dots, i_N)$, con valores $v_i$ y pesos $w_i$ — peso total max.

↳ Cogemos elementos (los elementos no se pueden dividir) t.q. $\begin{cases} \sum_j w_j \leq W \text{ (la suma de sus pesos es menor a } W) \\ \sum_j v_j \text{ es máximo} \end{cases}$

- En el tema 4 vimos un algoritmo codicioso que solo funciona si se pueden dividir los elementos.

PROGRAMACIÓN DINÁMICA → Definimos $V(i, w)$ → valor óptimo de la mochila con los primeros $i$ elementos y de cota $w$

- Si $i$ SÍ está en la selección → $v(i,w) = v_i + v(i-1, w - w_i)$
- Si $i$ NO está en la selección → $v(i,w) = v(i-1, w)$

- $v(i,0) = 0$
- $v(0,w) = 0$

→ $v(i,w) = \max\{v_i + v(i-1, w-w_i), v(i-1,w)\}$

Lo cogemos | No lo cogemos

Si $w < w_i$ : $v(i,w) = v(i-1,w)$

↳ Solución: $M(N,W)$ esquina inferior derecha.

- Para ejecutar la fila $i$ necesitamos ejecutar la fila $(i-1)$

EJEMPLO: $v = \{15, 11, 10\}$ ; $w = \{5, 4, 4\}$ ; $W = 8$

| v | $w_i$ \ W | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 4 | 1 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 |
| 11 | 4 | 2 | 0 | 0 | 0 | 0 | 11 | 11 | 11 | 11 | 21 |
| 15 | 5 | 3 | 0 | 0 | 0 | 0 | 11 | 15 | 15 | 15 | (21) |

### COSTES

- Rellenamos matriz $N \times W$
  - $v(i,w) \to O(1)$
  - Coste total: $O(N \times W) = O(N \times 2^{\lg W})$
- Tamaño problema: $O(N + \lg W)$
  ↳ coste solución DP es exponencial en tamaño del problema.

PROBLEMAS NP-completos → versiones de decisión del { problema del cambio, mochila 0-1, viajante }

subclase de NP problems

Problemas de decisión para los que podemos comprobar la corrección de las soluciones con coste polinómico

ENGINEERING THE FUTURE

indra

En Indra sentimos

# #OrgulloIngeniero

**Tú también estás a un paso de transformar el futuro.**
Pero... ¿sabes qué opina la sociedad de la personalidad de los ingenieros?

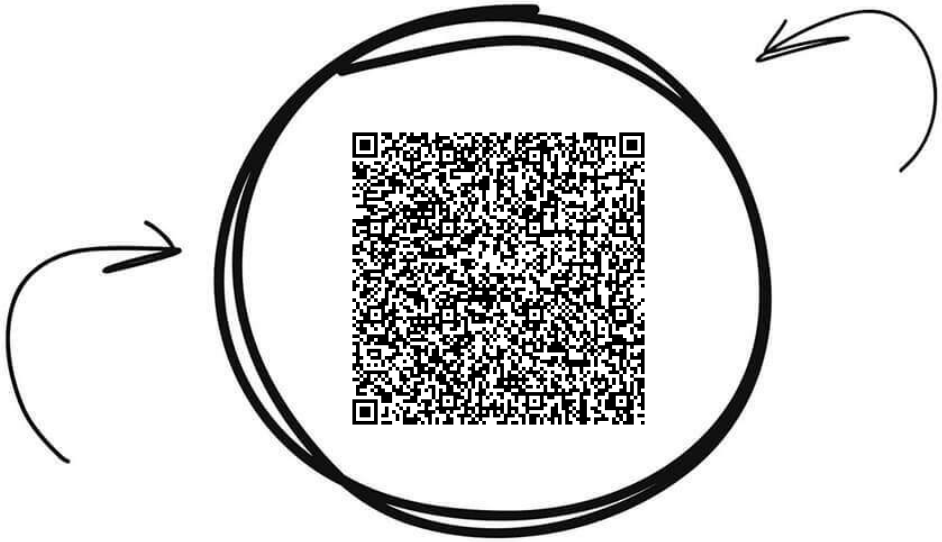
Que no tenemos creatividad, llevamos la lógica al extremo, somos indescifrables o que solo pensamos en números son algunos de los estereotipos.

¿A que no estás de acuerdo?
**¡Responde a estos estereotipos y derriba estos mitos!**


Comparte los motivos por los que sientes **#OrgulloIngeniero** y ayúdanos a poner en valor a la ingeniería.

orgullo-ingeniero.indra.es

# Algoritmia y Estructuras de...



## Banco de apuntes de la

WUOLAH

**Comparte estos flyers en tu clase y consigue más dinero y recompensas**

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

# 6.3. ALGORITMOS DE CADENA

## → DISTANCIA DE EDICIÓN

PROBLEMA: 2 cadenas S, T. Queremos transformar una en la otra con las siguientes operaciones:
- Cambiar un carácter por otro
- Eliminar un carácter
- Añadir un carácter

> i Es equivalente hacer todos los cambios en 1 cadena, que en ambas.

- len(S) = M
- len(T) = N

~~Ejemplo:~~

- Definimos DISTANCIA DE EDICIÓN $d(S,T)$ → nº mínimo de operaciones para transformar una cadena en la otra.
- Ejemplo d("unnecessarily", "unescessaraly") = 3

unne-cessari-ly
d i di
un-escessaraly

- $d_{i,j} = d(S[:i], T[:j])$

$$d_{i,j} = \begin{cases} d_{i-1,j-1} & \text{si } s_i = t_j \\ \min\{d_{i-1,j-1}+1, d_{i-1,j}+1, d_{i,j-1}+1\} & \text{si } s_i \neq t_j \end{cases}$$

$d_{i,0} = i$ → Distancia entre S/T y la cadena vacía

$d_{0,j} = j$ → Distancia entre S/T y la cadena vacía

$d_{i-1,j}+1$ ← (Eliminar $s_i$)

$d_{i,j-1}+1$ ← (Eliminar $t_j$)

EJEMPLO

| | ϕ | b | i | s | c | u | i | t |
|---|---|---|---|---|---|---|---|---|
| ϕ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| s | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 |
| u | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 5 |
| i | 3 | 3 | 2 | 3 | 4 | 4 | 3 | 4 |
| t | 4 | 4 | 3 | 3 | 4 | 5 | 4 | 3 |
| c | 5 | 5 | 4 | 4 | 3 | 4 | 5 | 4 |
| a | 6 | 6 | 5 | 5 | 4 | 4 | 5 | 5 |
| s | 7 | 7 | 6 | 5 | 5 | 5 | 5 | 6 |
| e | 8 | 8 | 7 | 6 | 6 | 6 | 6 | 6 |

COSTE

TIEMPO: $O(M \times N)$ → no NP-problem, pero costoso

MEMORIA: $O(M \times N)$ → MAL, ya que el problema tiene tamaño $O(M+N)$ se puede reducir si solo nos interesa el $d_{M,N}$ final a $O(2 \times M)$

## → SUBCADENA COMÚN MÁS LARGA

PROBLEM: Dados 2 strings S, T, encontrar la subcadena común más larga de ambas (no necesariamente consecutiva) (LCS)

- Definimos $l_{i,j}$ longitud de la LCS entre $S_i, T_j$

$$l_{i,j} = \begin{cases} 0 & \text{si } i = 0 \text{ o } j = 0 \\ l_{i-1,j-1} + 1 & \text{si } s_i = t_j \\ \max\{l_{i-1,j}, l_{i,j-1}\} & \text{si } s_i \neq t_j \end{cases}$$

Para hallar la cadena:
1. Hallamos la matriz LCS
2. Recorremos hacia atrás

EJEMPLO

| | ϕ | f | o | r | c | a | j | e |
|---|---|---|---|---|---|---|---|---|
| ϕ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| f | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| o | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| r | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| c | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| a | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| j | 0 | 0 | 0 | 1 | 1 | 2 | 3 | 3 |
| e | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 3 | (3) |

$S_c = [a_{ij}]$

## 6.4. MULTIPLICACIÓN ÓPTIMA DE MATRICES

PROBLEMA: Dadas N matrices $A_1, \dots, A_N$ con tamaños $c_{i-1} \times c_i$ $(0 \le i \le N)$, el orden por el cual multiplicamos no afecta matemáticamente, pero sí al coste computacional. Queremos encontrar el orden de multiplicación para minimizar el coste.

Ejemplo: $A_1 A_2 A_3 A_4$ con tamaños $50\times10$, $10\times40$, $40\times30$, $30\times5$

- Coste de $A_1(A_2(A_3A_4))$ es 10 500 productos.
- Coste de $((A_1A_2)A_3)A_4)$ es 87 500 productos.

Estructura Óptima Explícita (EOS)

Definimos $m_{L,R}$ → nº mínimo de productos para multiplicar $A_L \dots A_R$.

→ $m_{i,i} = 0$

→ $m_{i,i+1} = c_{i-1} \times c_i \times c_{i+1}$

$$m_{LL} = 0, \quad 1 \le L \le N$$

$$m_{LR} = \min_{L \le j \le R-1} \{ c_{L-1} \times c_j \times c_R + m_{L,j} + m_{j+1,R} \}$$

$1 \le L < R \le N$

Para llegar a la propiedad de EOS, supongamos que la ordenación óptima es $(A_L \dots A_j)(A_{j+1} \times A_R)$, entonces:

$$m_{L,R} = \underbrace{c_{L-1} \times c_j \times c_R}_{(A_L \dots A_j)(A_{j+1} \dots A_R)} + m_{L,j} + m_{j+1,R}$$

EJEMPLO Dimensiones: [50, 10, 40, 30, 5]

| D \ I | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4 | 10500 | 8000 (5) | 6000 (3) | 0 |
| 3 | 27000 (4) | 12000 (2) | 0 | |
| 2 | 20000 (1) | 0 | | |
| 1 | 0 | | | |

(1) $A_1A_2 \to 50\times10\times40 = 20000$

(2) $A_2A_3 \to 10\times40\times30 = 12000$

(3) $A_3A_4 \to 40\times30\times5 = 6000$

(4) $(A_1A_2)A_3 \to \overbrace{50\times40\times30}^{60000} + 20000 = 80000$

$A_1(A_2A_3) \to \overbrace{50\times10\times30}^{15000} + 12000 = 27000$

(5) $(A_2A_3)A_4 \to \overbrace{10\times30\times5}^{1500} + 12000 = 13500$

$A_2(A_3A_4) \to \overbrace{10\times40\times5}^{2000} + 6000 = 8000$

(6) $(A_1A_2)(A_3A_4) \to \overbrace{50\times30\times5}^{7500} + 27000 = \dots$

$(A_1A_2)(A_3A_4) \to \overbrace{50\times40\times5}^{10000} + 20000 + 6000 = \dots$

$A_1(A_2(A_3A_4)) \to \overbrace{50\times10\times5}^{2500} + 8000 = 10500$

COSTE

$O(N)$ cada uno
$O(N)$ la matriz
$\}$ $O(N^3)$ en total.