

# ALGORISIMIA-RESUM-TOT.pdf



Arnau\_FIB



Algorítmica



3º Grado en Ingeniería Informática



Facultad de Informática de Barcelona (FIB)  
Universidad Politécnica de Catalunya

**SUBRAYAR NO  
ES ESTUDIAR**

Ambar recomienda el consumo responsable

LA AMARGA VERDAD DE UNA  
CERVEZA AMARGA DE VERDAD



LO DE ESTUDIAR PARA UN BIEN  
SUELE SALIR ~~MAL~~



LA AMARGA VERDAD DE UNA CERVEZA AMARGA

### ALGORITMIA. FÓRMULAS

$$\text{Suma Aritmética} : \sum_{i=0}^n i = \frac{n(n+1)}{2}$$

$$\text{Suma Geométrica} : \sum_{i=0}^n a^i = \frac{1-a^{n+1}}{1-a}$$

$$\sum_{i=b}^a x^i = \frac{x^{a+1} - x^b}{x - 1} \quad (\text{corrado}) \rightarrow \text{Ex: } \sum_{i=1}^{\lg n} 4^i = \frac{4^{\lg n + 1} - 4}{4 - 1}$$

Master Theorem

$$T(n) = a \cdot T(n/b) + \Theta(n^k) \rightarrow T(n) = \begin{cases} \Theta(n^k), & k > \log_b a \\ \Theta(n^k \lg n), & k = \log_b a \\ \Theta(n^{\log_b a}), & k < \log_b a \end{cases}$$

Master Theorem for Subtractions

$$T(n) = a \cdot T(n-c) + \Theta(n^k) \rightarrow T(n) = \begin{cases} \Theta(n^k), & a < 1 \\ \Theta(n^{k+1}), & a = 1 \\ \Theta(n^{k/c}), & a > 1 \end{cases}$$

Diverses propietats

$$\log_b n = 1 \quad \log_b n = \frac{\log_a n}{\log_a b} \quad \frac{1}{\log_b a} = \log_a b$$

$$a^x = P \Leftrightarrow \log_a P = x \quad n! \approx n^n \cdot e^{-n} \cdot \sqrt{2\pi n}$$

$$a^{\log_a x} = x$$

$${n \choose k} = \frac{n!}{k!(n-k)!}$$

## ALGORITMIA

### 1. BASIC ALGORITHM CONCEPTS

#### INTRODUCCIÓN

Notación asintótica

$$L = \lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)}$$

$f(n) = O(g(n))$	$L < \infty$	$f \leq g$
$f(n) = \Omega(g(n))$	$L > 0$	$f \geq g$
$f(n) = \Theta(g(n))$	$0 < L < \infty$	$f = g$
$f(n) = o(g(n))$	$L = 0$	$f < g$
$f(n) = \omega(g(n))$	$L = \infty$	$f > g$

Exemple:

1)  $2^n$  i  $n \cdot 2^n$  tenen cost asintótic  $2^n O(n)$

$$2^n \rightarrow 2^n O(n)$$

$$n \cdot 2^n = 2^{\lg n} \cdot 2^n = 2^{(\lg n + n)} = 2^n O(n)$$

2) Si una funció és  $O(n \lg n)$  també és  $O(n^2)$ , ja que si té com a límit  $n \lg n$ , també  $n^2$ .

#### classes P i NP

P → problemes que es pot solucionar en temps polinòmic

NP → problemes que es poden resoldre alguns en temps pol.

i molts altres no. Non-deterministic Polynomial time.

$P \subseteq NP \rightarrow$  open problem:  $P = NP \circ P \neq NP$

NP-complet → problemes més difícils. Si un es soluciona en temps polinòmic → tots es resolen  $\rightarrow P = NP$

NP-hard: Tot problema NP es pot transformar/reduir en temps polin.

a ell, tot i que no cal que sigui NP.

NP-complete: Es NP-hard i es NP.



# La autoescuela por fin en tu móvil



Aprende, aprueba  
y conduce.



## Curso Teórico Dribo

+2500 preguntas oficiales DGT  
Recomendaciones personalizadas  
Soporte de profesores

## Todo tu carnet con 10 prácticas

+2500 preguntas oficiales DGT  
Prácticas en zon de examen  
Queda con tu profesor/a

**dribo**  
Drive with ❤

## SELECTION AND SORTING

### Selection

Donada una llista no ordenada d'elements, volem trobar el i-èssim més petit de la llista.

1. Dividim la llista en particions de 5 elements  $O(n)$
2. Trobem les medianes de cada grup  $O(n)$
3. Trobem la mediana de les medianes  $\rightarrow x \quad T(n/5)$
4. Partitionem la llista al voltant de  $x$ .  $O(n)$
5. Si  $l[i] = x$ , ja el tenim
- si  $l[k] = x$  i  $i > k$ , cridem recursivament a la dreta  
Altrament cridem a l'esquerra.  $T(n) = \Theta(n)$

### Sorting

Assumim que comparar dos elements es temps constant.

COUNTING SORT      Counting Sort ( $A, r$ );  $0 \leq A[i] \leq r, i \in \{0, \dots, n\}$

Considera els possibles valors  $i \in [0, r]$ . Per cada un d'ells conta quants elements més petits que ell hi ha a  $A$ . Ho utilitza per saber on va l'element en qüestió.

$$T(n) = \Theta(n + r) \quad n = |A| \quad r = \text{rang} \rightarrow [0, r]$$

És estable: Els números amb el mateix valor apareixen en el mateix ordre que a l'entrada

RADIX SORT      RADIX-LSD ( $A, d, b$ )

Donat vector  $A$  amb  $n$  elements amb  $d$  díigits en base  $b$ , fem un Counting Sort per cada un dels díigits.

$$T(n, d) = \Theta(d(n+b))$$

## GREEDY ALGORITHMS

Algorisme veraq obté una solució òptima a un problema fent una seqüència de decisions.

Un algorisme veraq no fa mai 'back tracking'.

Perquè funcioni correctament l'algorisme:

- capaç d'arribar a la sol. òptima fent la tria local.
- capaç arribar sol. òptima amb el camí fet fins el mom.

## INTERVAL SCHEDULING

- Solució inicial bàsica. Anar mirant les que acaben abans i en van posant a la llista. Si empata, s'agafa la correcta que no solapei. El cost és  $O(n^2)$
- Solució amb ordenació. Ordenem per finalització i anem afegint les que no es solapin. El cost és  $O(n \log n)$   
↳ Si sabem rang valors → ordenem amb RADIX / Counting
- Afegim pesos: No en coneix algorisme veraq per trobar sol.

## JOB SCHEDULING

Cal ordenar en funció del criteri que vulguem, el millor aparentment es ordenar per deadline. Així minimitzem el retard. A més, no creem espais on no es fa cap treball.

# LO DE ESTUDIAR PARA UN BIEN

→ SUELE SALIR ~~MAL~~



DE VERDAD

CERVEZA AMARGA

VERDAD

AMBAR  
CERVEZAS INDEPENDIENTES

Ambar recomienda el consumo responsable

## MINIMUM SPANNING TREE

ParticiS de vèrtexs

Blue rule: Donat un cut-set entre  $S$  i  $V-S$  sense arcs blaus, selecciona del cut-set un arc sense color amb el mínim pes.

Red rule: Donat un cicle  $C$  sense aretes vermelles, seleccionar una areta no pintada amb el pes màxim i pintar-la de vermell.

### Prim's Algorithm

Començant per un vèrtex  $v$ , incrementa  $T$  anant afegint cada vegada el vèrtex connectat a qualsevol de  $T$  amb el mínim pes, aplicant la "Blue Rule". Priority queue (heap) per guardar les aretes i agafar la de menys pes.

Cost: Depèn de la implementació de  $Q$  (arestes a explorar)

$Q$  és vector desordenat  $\rightarrow T(n) = O(|V|^2)$

$Q$  és heap  $\rightarrow T(n) = O(|E| \lg |V|)$

$Q$  és Fibonacci Heap  $\rightarrow T(n) = O(|E| + |V| \lg |V|)$

### Kruskal's Algorithm

Considera tots els vèrtexs i fa créixer un arbre utilitzant les "Blue Rule" i "Red Rule" per afegir o descartar "e". Ordena les aretes per pes creixent i les va afegint sempre que no formin un cicle.

Per detectar els cicles eficientment s'utilitza l'estructura d'una partici d'un set)

de dades Union-Find (colecció dinàmica d'un set)

Operacions Union-Find: MAKE SET( $x$ ) :  $O(1)$   $\rightarrow$  creat new set.

UNION ( $x, y$ ) :  $O(k \cdot \alpha(n))$   $\rightarrow$  merge two sets.

FIND ( $x$ ) :  $O(\alpha(n))$   $\rightarrow$  returns representative.

Cost:  $T(n) = O(n + m \lg n)$

## DATA COMPRESSION

Donat un text  $T$  sobre un alfabet  $\Sigma$  volem representar el text amb els mínims bits possibles.

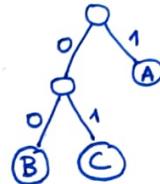
Propietat de prefix: Sent  $\phi: \Sigma \rightarrow \{0,1\}^*$  aleshores per tot  $x, y \in \Sigma$   $\phi(x)$  no és prefix de  $\phi(y)$

Exemple:  $\Sigma = \{A, B, C\} \rightarrow \phi(A) = 1 \quad \phi(B) = 00 \quad \phi(C) = 01$

### Arbre prefix

Arbre binari que compleix:

- Una fulla per símbol
- Camí arrel - fulla és  $\phi(\text{fulla})$



### Mida codificació

$$n = |T|$$

$$\forall x \in \Sigma \text{ la freqüència d'aparició } f(x) = \frac{|T_x|}{n}$$

$$\text{Mida codificació } B(T) = n \cdot \alpha(T)$$

$$\alpha(T) = \sum_{x \in \Sigma} f(x) \cdot |\phi(x)|$$

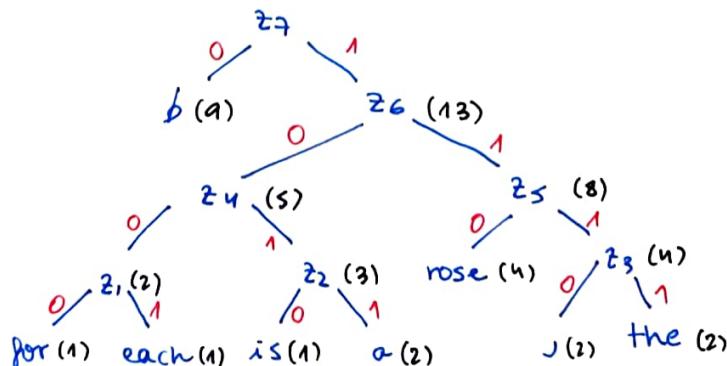
### Greedy Huffman's Algorithm $T(n) = n \lg n$

Tenim  $f(x)$  per tot  $x \in \Sigma$ . Ordenem els símbols per  $f(x)$ , en una priority queue. Construïm un arbre de baix a dalt, agafem els dos primers elements de PQ i creem el pare, que tindrà com a  $f(x)$  la suma dels dos fills.

Quan la PQ estigui buida, l'arbre estarà complet.

Ex: "for each rose, a rose is a rose"

$$f(x) = \frac{\text{aparicions } x}{\text{paraula + espais}}$$



El de més per a l'esquerra, i si igual de pes, per ordre de aparició. (Si creem  $z_i$ , aquest no ha aparegut mai a la seua)

## Algoritmos de Aproximación

Greedy aproxima la solució amb un valor a prop de la solució óptima.

Donada una  $r > 1$ , una  $r$ -aproximació de l'algorisme si

$$\forall x \in \text{Input}(\text{id}) \quad \frac{1}{r} \leq \frac{\text{id}(x)}{\text{opt}(x)} \leq r \quad \left\{ \begin{array}{l} \text{MAX prob: } \text{id}(x) \leq \text{opt}(x) \leq r \cdot \text{id}(x) \\ \text{MIN prob: } \text{opt}(x) \leq \text{id}(x) \leq r \cdot \text{opt}(x) \end{array} \right.$$

cost de

$\text{id}(x)$  → Algorisme polinòmic que

produceix una sol. aproximaçió

$\text{opt}(x)$  → cost solució óptima

### Vertex Cover problem

problema 31 llista 2.

Donat  $G = (V, E)$  trobar  $S \subseteq V$  mínim tq  $\forall \{u, v\} \in E$

$u \in S \vee v \in S$  on  $u, v \in V$ .

$E' = E$

cost:  $O(n+m)$

$S = \emptyset$

while  $E' \neq \emptyset$  do

$|S| \leq 2 \text{opt}(G)$

Pick  $e = (u, v)$

$r \leq 1.36$  per aprox un

$S = S \cup \{u, v\}$

NP-complet

$E' = E' - \{(u, v) \mid u, v \in \text{edges adj a } u, v\}$

end while

# DYNAMIC PROGRAMMING

## Fibonacci Recurrence

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

0, 1, 1, 2, 3, 5, 8, 13, ...

$$\lim_{n \rightarrow \infty} \frac{F_{n+1}}{F_n} = \varphi = 1.6180 \dots$$

DP - Fibonacci (n) {

$$F[0] = 0$$

$$F[1] = 1$$

for  $i = 2$  to  $n$  do

$$F[i] = F[i-1] + F[i-2]$$

return  $F[n]$

}

## Weighted Activity Selection

Set activitats  $S = \{1, 2, \dots, n\}$  on cada elem  $\in S$  té temps  $start_i$  o  $tempfin_i$ .  
 $i$  un per  $w_i$ . Trobar el set d'activitats compatibles

que maximiza  $\sum w_i$

Definim  $p(i)$  com  $j < i$  on  $j$  és el enter més gran

que l'activitat  $j$  no coincideix amb la  $i$ .

$opt(j)$  és la suma amb el major nombre d'activitats que es poden fer

$$opt(j) = \begin{cases} 0 & \text{if } j=0 \\ \max \{(opt(p[j]) + w_j), opt[j-1]\} & \text{if } j \geq 1 \end{cases}$$

## Memoization:

```
R - Opt(j) {
    if w[j] != -1 then
        ret (w[j])
    else
        ret max(w[j] + R-opt(P[j]), R-opt(j-1))
```

conjunt d'activitats ja ordenat.

Tots els  $p(j)$  calculats a  $P[]$

Taula  $W[n+1]$  per guardar valors. Inicialment -1.

cost:  $\Theta(n \lg n + n)$

WUOLAH

# LO DE ESTUDIAR PARA UN BIEN

→ SUELE SALIR ~~MAL~~



DE VERDAD

Multiplying a Sequence of Matrices

com multiplicar? → com fer parèntesi

$$A_1 \times A_2 \times A_3 \quad \text{on} \quad \begin{aligned} A_1 & (10, 100) & \rightarrow (A_1 \cdot A_2) \cdot A_3 \rightarrow 7500 \text{ op} \\ A_2 & (100, 5) & \rightarrow A_1 \cdot (A_2 \cdot A_3) \rightarrow 75000 \text{ op} \\ A_3 & (5, 50) & \text{cal tenir un bon ordre!} \end{aligned}$$

Per parentitzar  $(A_1 \times \dots \times A_n)$ :

$n=1 \rightarrow$  No fem res

$n>1 \rightarrow$  trobar una  $K$  amb  $1 \leq K \leq n$  tq  $((A_1 \times \dots \times A_K) (A_{K+1} \times \dots \times A_n))$

Fer-ho amb força bruta trigaria massa.

Hem de fer-ho amb una recurrència:

$$m[i, j] = \begin{cases} 0, & i=j \\ \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j \}, & i \neq j \end{cases}$$

Algorisme recursiu te cost  $\mathcal{O}(2^n)$ ,

Tenim subproblemes, donats un  $(i, j)$ ,

com  $1 \leq i \leq j \leq n$ , hi ha  $\mathcal{O}(n^2)$  subproblemes

Per tant, podem utilitzar Dynamic Programming → Memorització

→ Recurs

→ Iterat

Tabulating

(Algorismes a les transparències A/I/O)

## 0-1 Knapsack

Donats  $n$  items que no poden ser fraccionats. Un item  $i$  té pes  $w_i$ , valor  $v_i$ . El màxim pes permès és  $W$ .

Objectiu: Trobar  $S \subseteq I$  tq es maximitzi  $\sum_{i \in S} v_i$

Definim la recurrència:

$V[i, x]$  valor màxim que podem obtenir amb els objectes  $\{1, \dots, i\}$  amb màxim pes total  $\leq x$ .

$$V[i, x] = \begin{cases} 0 & \text{if } i == 0 \text{ or } w == 0 \\ \max \{ V[i-1, x - w_i] + v_i, V[i-1, x] \} & \text{altrament} \end{cases}$$

Algorisme:

$P[n+1, W+1]$  inicialment tot a 0.  $V[i]$  conté  $v_i$

for  $i=1$  to  $n$  do

    for  $x=0$  to  $W$  do

$$P[i, x] = \max \{ P[i-1, x - w_i] + v_i, P[i-1, x] \}$$

return  $P[n, W]$

$O(n \cdot W)$

primera fila i columna

Recuperar la solució:

Creem una altra taula  $K[n+1, W+1]$

on guardem  $K[i, x] = 1$  quan el

màxim de  $P[i, x]$  és perquè hem posat

l'element  $i$ . La recorrem així

$S = \emptyset$

for  $i=n$  to 0 do

    if  $K[i, x] = 1$  do

$S = S \cup \{i\}$

$x = x - w_i$

return  $S$

$O(n \cdot W)$

Complexitat:

Te cost pseudo-polinòmic en relació al valor numèric de l'entrada, però exponencial en la mida de l'entrada

(mida en bits)

## SHORTEST PATHS PROBLEMS

Distancia :  $\delta(u, v) = \min \{ w(p) \mid u \xrightarrow{p} v \}$

si no existeix camí  $u \xrightarrow{p} v \rightarrow \delta(u, v) = +\infty$

- Una distància entre tots els parells de vèrtexs es pot definir en un graf si no existeix un cicle amb pes negatius ( $w(c) \leq 0$ )

### Shortest Path Tree $O(n+m)$

Directed sub-tree. Es té d'arrel  $s$ , i tots els fills estan connectats pel camí més curt.

### Shortest Path Problems

- Dijkstra : Funciona per pesos positius.
- Bellman-Ford: Funciona per tots els pesos. També detecta cicles de pes negatiu.

### SSSP: Single Source Shortest Path

- Dijkstra: Funciona si  $w(e) \geq 0$ . Utilitza priority queue. Alg pàg 40 (All)  
Cost :  $\Theta(m \lg n) \rightarrow P.Q.$   
 $\Theta(m + n \lg n) \rightarrow$  Fib. Heap

- Bellman-Ford: Funciona per tots els pesos. Detecta els cicles negatius. Alg pàg 45  
Cost :  $O(n \cdot m)$

- DAG : Directed Acyclic Graphs. No tenim cicles, fem en ordre topològic per millorar eficiència.

Ara ja podem calcular el camí més curt des del 'source'. Alg pg 67  
Cost :  $\Theta(n+m)$

## All Pairs Shortest Path

graf representat amb matrius d'adjacència  $\rightarrow w_{ij} = \begin{cases} 0 & , i=j \\ w_{ij} & , (i,j) \in E \\ +\infty & , i \neq j \wedge (i,j) \notin E \end{cases}$

Input: nxn matrix  $W = (w_{ij})$

Output: nxn matrix  $D$  on  $D[i,j] = \delta(i,j)$  i

nxn matrix  $P$  on  $P[i,j]$  és el predecessor de  $j$  en un camí mínim de  $i$  a  $j$ .

## FLOYD - WARSHALL

S'aprofita de l'estructura recursiva del camí més curt entre dos vèrtexs per resoldre el problema.

- Tot subcamí d'un camí mínim és mínim.

$P = P_0, P_1, \dots, P_{r-1}, P_r$        $d_{ij}^{(k)}$ : mínim per d'un camí de  $i$  a  $j$ , on hi ha  $\{1, \dots, k\}$  vèrtex entre  $i$  i  $j$ .

subcamí

- La recurrència:

Quan  $k=0$ ,  $d_{ij}^{(0)} = w_{ij}$

Suposem un camí  $i \rightsquigarrow j$  amb vèrtexs intermitjós  $\{1, \dots, k\}$  i pes  $d_{ij}^{(k)}$

$\rightarrow$  Si  $k$  no és intermid  $\Rightarrow d_{ij}^{(k)} = d_{ij}^{(k+1)}$

$\rightarrow$  Si  $k$  és intermid  $\Rightarrow d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

$$P = i \rightsquigarrow k \rightsquigarrow j$$

- Algorisme pàg 77 (All). Cost:  $O(n^3)$   
amb camins a pàg 80.

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & , k=0 \\ \min \{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \} & , k \geq 1 \end{cases}$$

- També cal guardar els predecessors, per fer-ho, inicialitzem la matriu  $P^{(0)}$ .

$\rightarrow$  Per  $k \geq 1$ :

$$P_{i,j}^{(0)} = \begin{cases} NIL & \text{if } i=j \vee w_{ij} = +\infty \\ i & \text{if } i \neq j \wedge w_{ij} \neq +\infty \end{cases} \parallel P_{i,j}^{(k)} = \begin{cases} P_{i,j}^{(k-1)} & , k \text{ not used} \\ P_{k,i}^{(k-1)} & , k \text{ used} \end{cases}$$

$O(n^3)$  per construir  $P$  (recuperar predecessors)

WUOLAH

LO DE ESTUDIAR PARA UN BIEN

→ SUELE SALIR ~~MAL~~



DE VERDAD

### JOHNSON

Més ràpid per grafs esparsos  $\rightarrow m = o(n^2)$

Graf donat amb mister d'adjacència.

Utilitza BF per detectar que no hi hagi cicles negatius, després executa n vegades Dijkstra.

$$\text{Cost : } \Theta(n \cdot m) + \Theta(n \cdot (m + n \lg n)) = \Theta(nm + n^2 \cdot \lg n)$$

L'algorisme transforma el graf inicial mantenint les relacions de pesos però sense ser negatius.

### Longest common Subsequence (ADN)

Busquem caràcters consecutius semblants entre dos strings. que poden ser

$$X = \langle x_1, \dots, x_n \rangle$$

$$Y = \langle y_1, \dots, y_m \rangle$$

$$\text{sol: } Z = \langle x_{i_1}, \dots, x_{i_k} \rangle = \langle y_{j_1}, \dots, y_{j_k} \rangle$$

Per la recurrència:

$$x(i) = \langle x_1, \dots, x_i \rangle \quad y(j) = \langle y_1, \dots, y_j \rangle \quad i \leq n, j \leq m$$

$$c[i, j] = \text{màxima longitud de subseqüència}$$

Recurrència:

$$c[i, j] = \begin{cases} 0 & , i=0 \text{ || } j=0 \\ c[i-1, j-1] + 1 & , x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & , x_i \neq y_j \end{cases}$$

$$\text{Algorisme recursiu} \rightarrow T(n, m) = 3^{O(n+m)}$$

$$\text{Tabulació} \rightarrow \text{pàg 16 (A12)} \rightarrow T(n, m) = \Theta(n \cdot m)$$

$$\text{Després, per accedir a la solució} \rightarrow \Theta(n+m)$$

## Longest common substring

DEA DBBEEF :  $x$        $Z = \langle x_i, \dots, x_{i+k} \rangle = \langle y_j, \dots, y_{j+k} \rangle$   
EAT BEEF :  $y$

BEEF :  $Z \rightarrow$  Subcadena más larga

$$S[i,j] = \begin{cases} 0 & , i=0 \text{ || } j=0 \\ 0 & , x_i \neq x_j \\ S[i-1, j-1] + 1 & , x_i = x_j \end{cases}$$

Tabulating  $\rightarrow$  algoritme pàg 26 (A12)  $\rightarrow O(nm)$

## Edit Distance Problem

Mínim d'operacions / minimitzar cost per transformar una cadena  $x$  en una cadena  $y$ .

Operacions : insert, delete, modify

$$E[i,j] = \begin{cases} i & \text{if } j=0 (\lambda \rightarrow y[i]) \\ j & \text{if } i=0 (x[i] \rightarrow \lambda) \\ \min \begin{cases} E[i-1, j] + 1 & \text{if D (delete)} \\ E[i, j-1] + 1 & \text{if I (insert)} \\ E[i-1, j-1] + \delta(x_i, y_j) & \text{otherwise} \end{cases} & \end{cases}$$

$$\delta(x_i, y_j) = \begin{cases} 0 & x_i = y_j \\ 1 & \text{otherwise} \end{cases}$$

Tabulating  $\rightarrow$  Alg pàg 44 (A12)  $\rightarrow$  Cost :  $O(n \cdot m)$

## MAX - FLOW AND MIN - CUT

### FLOW NETWORK

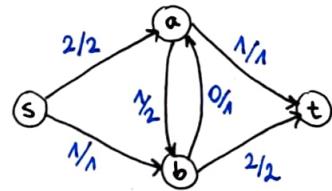
$N = (V, E, c, s, t)$  format per un digraf, vèrtexs, sink vèrtexs (on acaba el flux) i capacitat dels arcs  $\rightarrow c(u,v)$  per  $(u,v) \in E$  (on comença el flux)

Flux: Segueix les normes de Kirchoff

- $\forall (u,v) \in E \quad 0 \leq f(u,v) \leq c(u,v)$
- $\forall v \in V - \{s,t\} \quad \sum_{u \in V} f(u,v) = \sum_{z \in V} f(v,z)$

$$|f| = \sum_{v \in V} f(s,v) = f(s,V) = f(V,t)$$

$$\frac{f(e)}{c(e)}$$



### MAX - FLOW PROBLEM

Donada una xarxa  $N = (V, E, c, s, t)$ . Trobar el màx flow

$(S,T)$ -cut: Dividir la xarxa en 2.  $V = S \cup T$  i  $S \cap T = \emptyset$

i  $s \in S$  i  $t \in T$ . Es un cut qualsevol.

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

Capacitat de les aretes que van de  $S$  a  $T$

flow across the cut:  $\rightarrow$  Propietat: És igual independentment dels tall que es facin.

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{v \in T} \sum_{u \in S} f(v,u)$$

Flux de les aretes de  $S$  a  $T$  menys el flux de les aretes de  $T$  a  $S$ .

## RESIDUAL GRAPH

Xarxa formada pel graf  $G_f = (V_f, E_f, c_f)$  a partir d'una xarxa  $N$  amb flux  $f$ . És el graf format per els mateixos aretes i vèrtexs, però  $c_f = c - f$  ( $c_f > 0$  sempre)

forward edges  $\rightarrow$  Aretes lliures. Tenen la capacitat que poden donar.

backward edges  $\rightarrow$  Aretes ocupades que ja tenen el flux utilitzat.

$c_f \rightarrow$  Residual capacity

## AUGMENTING PATHS

Camí  $P$  en  $G_f$  que va de  $s$  a  $t$ .  $P$  pot tenir forward i backward edges. Serveix per incrementar el flow

Alg. pàg 29 (A13)  $\rightarrow$  Incrementa el flow de la xarxa.

Bottleneck  $b(P)$ : capacitat residual mínima de les aretes del augmenting path  $P$ .

## MAX-FLOW MIN-CUT THEOREM

$$\max_f \{ |f| \} = \min_{(S,T)} \{ c(S,T) \}$$

El flow màxim és igual al mínim de les capacitats dels  $(S,T)$ -cuts.

## FORD - FULKERSON

Ford-Fulkerson ( $G, s, t, c$ )

for all  $(u,v) \in E$  let  $f(u,v) = 0$

$$G_f = G$$

while there is an  $s-t$  path  $P$  in  $G_f$  do

$f = \text{Augment}(P, G_f)$

Compute  $G_f$

return  $f$

$\min \text{ capacitat}$

$\uparrow$   
num iteracions  $\leq C$

Constr.  $G_f$   $E(G_f) \leq 2m \rightarrow O(m)$

Aum. path  $\rightarrow O(n+m)$

$\downarrow$

$O(c(n+m))$

$\downarrow$   
 $O(c \cdot m)$

(pseudopolinomial)

LO DE ESTUDIAR PARA UN BIEN  
SUELE SALIR ~~MAL~~



## LA AMARGA VERDAD DE UNA CERVEZA AMARGA DE VERDAD

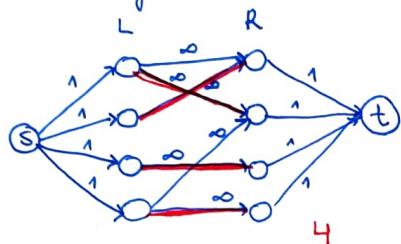
### MAXIMUM MATCHING

Problema: Donat graf  $G = (V, E)$  un subconjunt  $M \subseteq E$  és un matching si cada node apareix com a molt en una aresta de  $M$ .

Graf Bipartits: Graf  $G = (V, E)$  si hi ha una partició de  $V$  tq  $V = L \cup R \wedge L \cap R = \emptyset$  i  $\forall e \in E \quad e = (u, v) \quad u \in L \wedge v \in R$ .

### MAXIMUM MATCHING BIPARTITE GRAPH

Donat un graf bipartit  $G = (L \cup R, E)$ , trobar un maximum matching.



El flux màxim en la xarxa que hem construit resol el problema de trobar el maximum matching.

Cost:  $O(n \cdot (n+m))$

### DISJOINT PATH

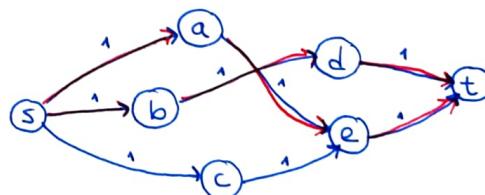
Problema: Donat  $G = (V, E)$  i dos vertexs  $s, t \in V$ , un conjunt de camins es edge-disjoint si les sevesarestes són disjunts (tot i que poden compartir vertexs).

Cal trobar el màxim número de camins disjunts.

Cal construir una xarxa  $N$  assignant capacitat a cada aresta.

El màxim número de camins disjunts es igual al màxim flux de  $s$  a  $t$ .

Cost:  $O(n(n+m))$



2

## EDMON - KARP ALGORITHM

FF algorithm però utilitzant BFS : trobar el augmenting path amb menys arestes.

Edmon-Karp ( G, c, s, t ) {

For - all  $e = (u, v) \in E$  let  $f(u, v) = 0$ ;

$G_f = G$

while there is an  $s \rightarrow t$  path in  $G_f$  do

$P = \text{BFS}(G_f, s, t)$

$f = \text{Augment}(f, P)$

Compute  $G_f$

return  $f$

{

- Necesita cost  $O(n+m)$  per fer el BFS
- Necesita fer  $O(n \cdot m)$  augmentations.

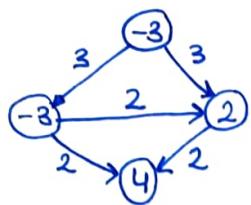
Cost :  $O(m \cdot n (n+m))$

## CIRCULATION WITH DEMANDS

No hi ha un node source ( $s$ ) i un sink ( $t$ ) sinó que tots els nodes poden produir o consumir.

$$N = (V, E, c, d) \quad c \rightarrow \text{capacitat } \oplus \text{ de cada aresta}$$

$$d \rightarrow \text{demanda d'un vèrtex } \ominus$$

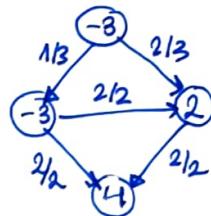


- $d(v) > 0 \rightarrow$  pot rebre  $d(v)$  més de les que dona
- $d(v) < 0 \rightarrow$  pot donar  $(d(v))$  més de les que rep
- $d(v) = 0 \rightarrow$  no pot ni donar ni rebre més del que rep ni done.

### Una circulació en $N$ :

$$1. \text{ Capacitat: } e \in E, 0 \leq f(e) \leq c(e)$$

$$2. \text{ Conservació: } \forall v \sum_{(u,v) \in E} f(u,v) - \sum_{(v,z) \in E} f(v,z) = d(v)$$



Una circulació pot no existir.

$$\text{Si una circulació existeix} \Rightarrow \sum_{v \in V} d(v) = 0$$

### Reducció a Max-Flow

$$\text{A partir de } N = (V, E, c, d) \text{ definim } N' = (V', E', c', s, t)$$

$$V' = V \cup \{s, t\}$$

$$D = \sum_{v \in T} d(v)$$

$$\forall v \in S \text{ afegim } (s, v) \text{ amb } c = -d(v)$$

$$S = \{v \in V \mid d(v) < 0\}$$

$$\forall v \in T \text{ afegim } (v, t) \text{ amb } c = d(v)$$

$$T = \{v \in T \mid d(v) > 0\}$$

Les aretes les mantenim i el capacitat també.

$$1. \text{ Cada flux a } f' \text{ verifica } |f'| \leq D$$

2. Si hi ha una circulació  $f$  a  $N$ , tenim max-flow en  $N'$  amb  $|f'| = D$

$$\text{amb } |f'| = D$$

3. Si tenim max-flow  $f'$  en  $N'$  amb  $|f'| = D$ ,  $N$  té circulació

Es pot resoldre una circulació en temps polinòmic.