Introducció

Curs Tardor 2018

Curs d'algorísmia:

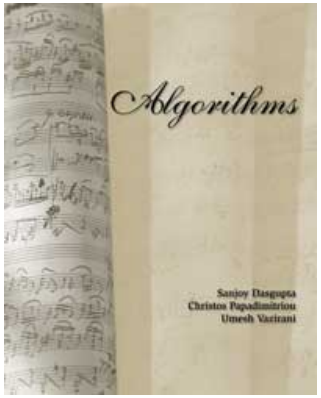
Continuació d'EDA

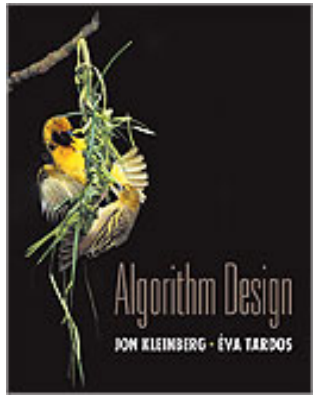
Què hauríeu de conèixer?

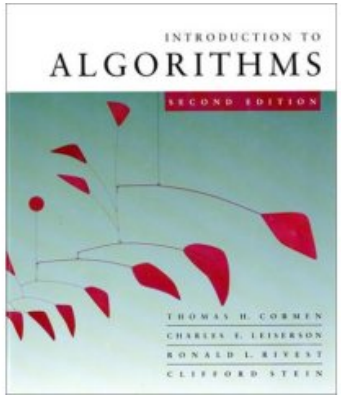
- Eines matemàtiques
  - Notació asimptòtica
  - Recurrències
- Metodologia per a dissenyar algorismes
  - Backtraking
  - Dividir i vèncer: Selecció
  - Ordenació lineal
  - Voraços i algorismes d'aproximació
  - Programació dinàmica
  - Algorismes per a fluxos en xarxes: Aplicacions
  - Programació Lineal
  - Estructura de dades avançades: Hashing,
  - Algorísmia distribuïda
- Problemes concrets
- Problemes "reals"

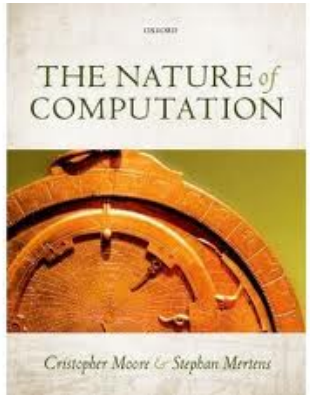
Bibliografia:

Referències principals:





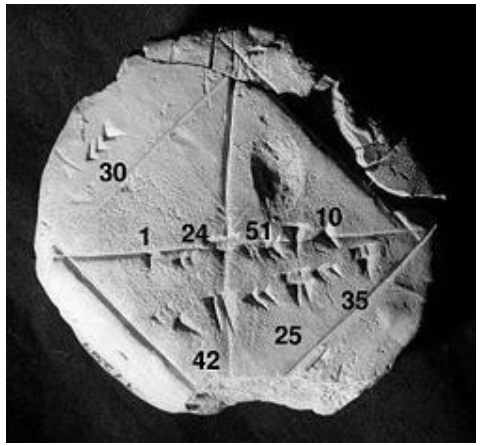




Algorismes.

Algorisme: recepta per a resoldre un problema.

Arrel (n)
$x_0 = 1$ $y_0 = n$
for $i = 1$ to 6 **do**
  $y_i = (x_{i-1} + y_{i-1})/2$
  $x_i = n/y_i$
end for



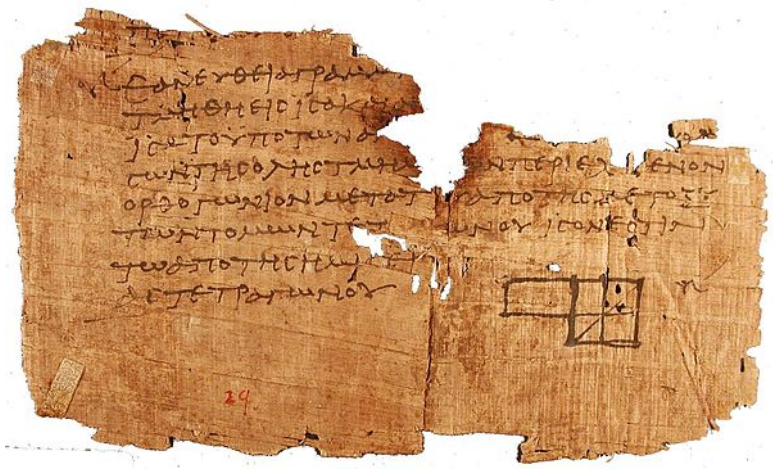
Babilònia (XVI BC)
Avui coneixem que: $\lim_{k\to\infty} x_k \to \sqrt{n}$ i ho fa ràpidament!

Un algorisme és correcte si per a qualsevol entrada s'atura i dóna una resposta correcta
Heurística: mètode per a resoldre un problema, que pot no aturar-se i pot no donar respostes exactes.

El primer algorisme no trivial

Donats $a, b \in \mathbb{Z}$:

mcd(a, b)
if $b = 0$ **then**
  return a
else if $b = 1$ **then**
  return 1
else
  mcd$(b, a \mod b)$
end if

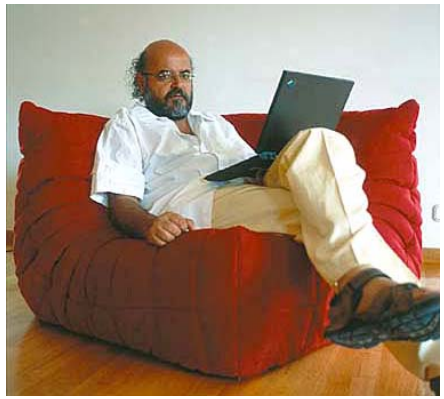


Euclides 300 BC Proposició VII-2 als *Elements*

Conseqüència que si $b|a \Rightarrow \text{mcd}(a, b) = b$,
altrament $a = bt + r \Rightarrow \text{mcd}(a, b) = \text{mcd}(b, r)$

"The algorithmic lenses"

L'algorísmia representa una nova manera de mirar els problemes en diferents àmbits de la ciència i la tecnologia. L'algorísmia estudia quines són les particularitats de l'estructura interna del problemes que ajudin a dissenyar algorismes més eficients . A més de la informàtica, l'algorísmia ha canviat la manera d'abordar i resoldre els problemes en camps com:



- Matemàtiques
- Físiques
- Biologia i epidemiologia
- Economia
- Sociologia

⋮

Donat un algorisme: Què volem estudiar?

- Si és correcte,
- Si és eficient (el seu cost)

Què vol dir cost:

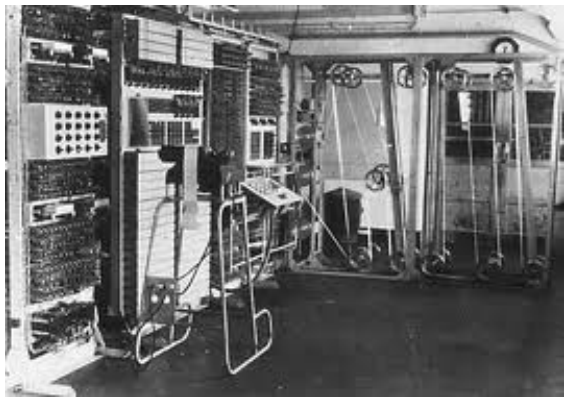
- Temps
- Memòria
- Capacitat de comunicació

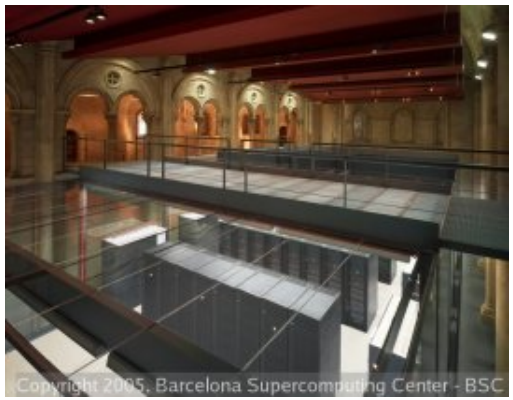
⋮

En aquest curs:
Cost d'un algorisme = temps de computació $T(n)$, parametrizat per grandària n.

Important

Les mesures de complexitat han de ser independents de les tecnologies existents





Anàlisis d'algorismes

:

Anàlisis cas pitjor: El màxim temps per a resoldre un problema amb grandària n, assumint l'entrada ve donada per un adversari.

Anàlisi mitjà: el nombre esperat de passos que un algorisme utilitza per a resoldre un problema, sobre totes les possibles entrades amb grandària n, agafades d'una distribució de probabilitat donada.

Escollir un estudiant per ajudar a la recerca

Tenim n estudiants, $\{1, \ldots, n\}$ i volem entrevistar-los per agafar el més adient (cada estudiant ve identificat per un enter entre 1 i n)
Després d'entrevistar cada estudiant hem de decidir si el pre-seleccionem o no.
A la fi del procés escollim un sol estudiant, però indemnitzem a cadascun dels pre-seleccionats no escollits, amb $S > 0$ €. Volem minimitzar el cost de les indemnitzacions. Sigui $PS(n)$ el nombre de pre-seleccionats.

Hiring (n)
best:=0
$PS(n) = 0$
for $i = 1$ to n **do**
  interview i
  if i is better than best **then**
    best:=i and pre-select i
    $PS(n) = PS(n) + 1$
  end if
end for

Complexitat

A la llista de l'adversari els estudiants estan escollits de manera que ordenadament van de menys apte a més apte. A cada pas l'algorisme es veu forçat a pre-seleccionar cada estudiant

Complexitat cas pitjor: $PS(n) \leq c_1 n$ (a on c_1 és una constant)

Notació asimptòtica.

Estudiem el comportament de $T(n)$ quan n pot prendre valors molt grans ($n \to \infty$)

if $n = 10$, $n^2 = 100$ i 2^n: 1024;

if $n = 100$, $n^2 = 10000$ i
$2^n = 1267650600228224401469670320537 6$;

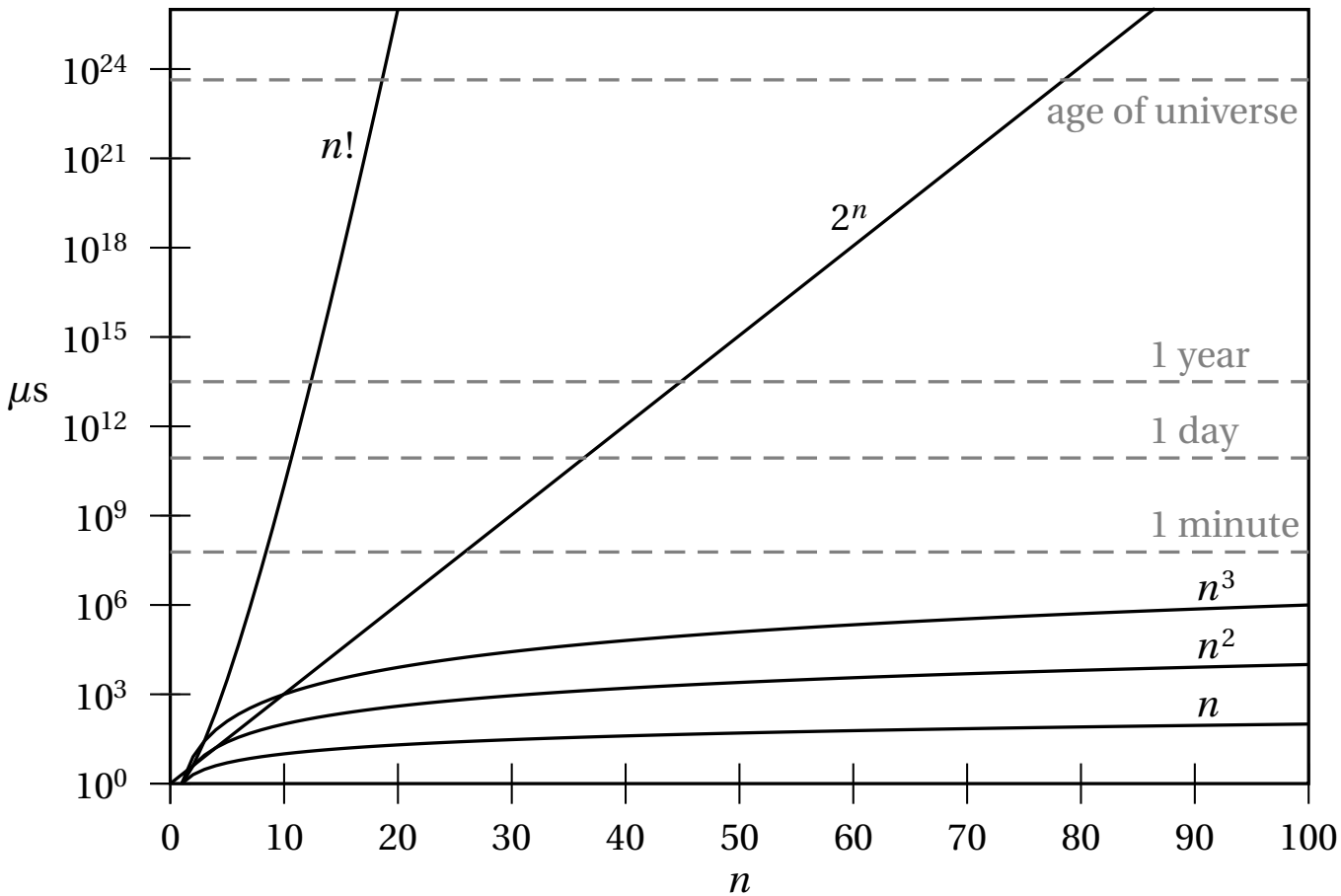
if $n = 10^3$ $n^2 = 10^6$ 2^n és un numero amb 302 dígits

10^{64} = nombre d'àtoms en la terra ($< 2^{213}$)

Notatció:
$\lg \equiv \log_2$; $\ln \equiv \log_e$; $\log \equiv \log_{10}$.

Comparació dels temps de computació amb els processadors actuals

Assumint una entrada amb grandària $n = 1$ es pot resoldre en 1 μsegon:



Temps de computació en funció de la grandària d'entrada n

	n	$n \lg n$	n^2	1.5^n	2^n
10	< 1s	< 1s	< 1s	< 1s	< 1s
50	< 1s	< 1s	< 1s	11m	36y
100	< 1s	< 1s	< 1s	12000y	10^{17}y
1000	< 1s	< 1s	< 1s	$> 10^{25}y$	$> 10^{25}y$
10^4	< 1s	< 1s	< 1s	$> 10^{25}y$	$> 10^{25}y$
10^5	< 1s	< 1s	< 1s	$> 10^{25}y$	$> 10^{25}y$
10^6	< 1s	20s	12d	$> 10^{25}y$	$> 10^{25}y$

Algorismes eficients i algorismes pràctics

Temps de computació: $n^{10^{10}}$ és un polinomi, però el temps de computació pot ser elevat.
De la mateixa manera, si tenim cn^2 per una constant $c = 10^{64}$, la constant té rellevància fins a entrades amb grandària $n = 10^{64}$.

A l'algorísmia, *factible (feasible)= temps polinòmic*, altrament *no factible (unfeasible)*.

A la pràctica és difícil implementar computacions amb valors més grans que n^4, per a valors "reals" de n.

A la pràctica, les constants grans tenen rellevància, però quan considerem asimptòtiques ($n \to \infty$) sempre podem agafar valors de n més grans que qualsevol constant.

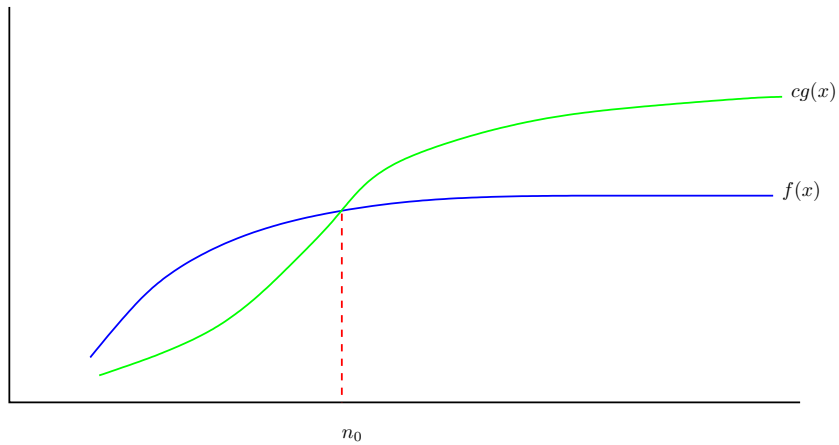
Big Oh: O.

Definition

Donades $f, g : \mathbb{Z} \to \mathbb{Z}$, definim:

$$O(g(n)) = \{f(n) | \exists c > 0, n_0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

$f(n) \in O(g(n))$ or $f(n) = O(g(n))$



$f(x) = O(g(x))$

$$\limsup_{n \to \infty} \frac{f(n)}{g(n)} \leq c.$$

$n^{10} = O(e^n)$: Escollir $c = 1$ and $n_0 = e^5$. Per tant,
$\forall n \geq \lceil e^5 \rceil, 10 \ln n \leq n, \Rightarrow e^{10 \ln n} \leq \lceil e^n \rceil$.

Notem que una constant $k > 0$, $k = O(1)$,
(agafem $c = k$, $n_0 = 1 \Rightarrow k \leq 1k$).

$\Rightarrow$ la funció constant $f(x) = k$ per tot enter x i un valor constant k també és $O(1)$.

Però, la funció $f(x) = x$ no és $O(1)$!!!

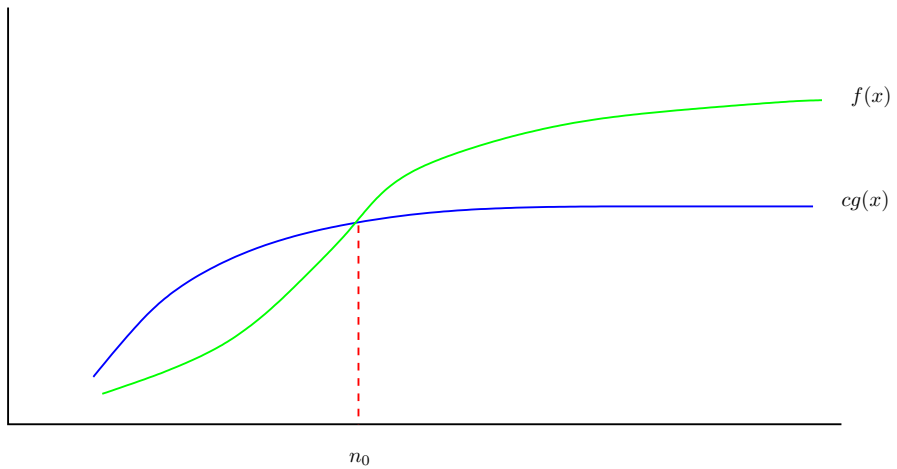
Omega: Ω.

Definition

Donades $f, g : \mathbb{Z} \to \mathbb{Z}$, definim:

$$\Omega(g(n)) = \{f(n) | \exists c > 0, n_0 : 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$

$f(n) \in \Omega(g(n))$ o $f(n) = \Omega(g(n))$



$f(x) = \omega(g(x))$

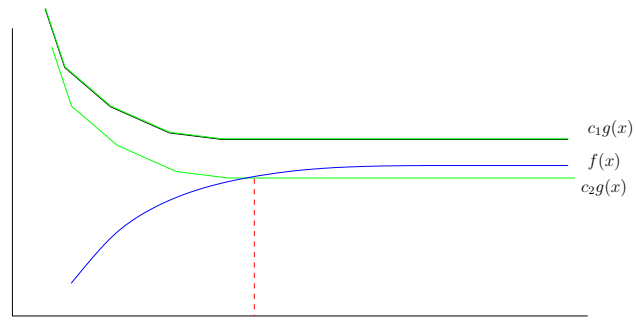
$$\liminf_{n \to \infty} \frac{f(n)}{g(n)} \geq c > 0.$$

Definition

Donades $f, g : \mathbb{Z} \to \mathbb{Z}$, definim:

$$\Theta(g(n)) = \{f(n) | \exists c_1, c_2, n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0\}$$

$f(n) = \Theta(g(n))$ sii $f(n) = \Omega(g(n))$ i $f(n) = O(g(n))$



$f(x) = \Theta(f(x))$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c, \text{ on } c = \max\{c_1, c_2\}$$

Donat $f(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$, llavors $f(x) = \Theta(x^k)$ (si $a_k \neq 0$).

Notem: La notació asimptòtica es pot utilitzar dintre d'una equació

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2).$$

Little-oh: o

Definition

Donades $f, g : \mathbb{Z} \to \mathbb{Z}$, definim:

$$o(g(n)) = \{f(n) | \exists n_0, \forall c > 0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}.$$

$f(n) = o(g(n))$

$$\limsup_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

$\forall \epsilon > 0, n^{1-\epsilon} = o(n)$ però $\forall \epsilon > 0, n^{1+\epsilon} \notin o(n)$

Little omega: ω

Definition

Donades $f, g : \mathbb{Z} \to \mathbb{Z}$, definim:

$$\omega(g(n)) = \{f(n) | \forall c > 0, \exists n_0 : 0 \leq cg(n) \leq f(n) cg, \forall n \geq n_0\}.$$

$f(n) = \omega(g(n))$

$$\liminf_{n \to \infty} \frac{f(n)}{g(n)} = \infty.$$

Notem: $n^2/2 = \omega(n)$ però $n^2/2 \notin \omega(n^2)$

Quadre resum

Símbol	$L = \lim_{n\to\infty} \frac{f(n)}{g(n)}$	intuïció ..
$f(n) = O(g(n))$	$L < \infty$	$f \leq g$
$f(n) = \Omega(g(n))$	$L > 0$	$f \geq g$
$f(n) = \Theta(g(n))$	$0 < L < \infty$	$f = g$
$f(n) = o(g(n))$	$L = 0$	$f < g$
$f(n) = \omega(g(n))$	$L = \infty$	$f > g$

Noms utilitzats per classes de funcions

classe	definition
polylogarirthmic	$f = O(\log^c n)$ for cte. c
polynomial	$f = O(n^c)$ for cte. c or $n^{O(1)}$
subexponential	$f = o(2^{n^\epsilon})\ \forall 1 > \epsilon > 0$
exponential	$f = 2^{\mathrm{poly}(n)}$

Igualtat asimptòtica

Definition

Given functions $f, g : \mathbb{Z} \to \mathbb{Z}$, definim: $f(n) \sim g(n)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

Per tant, $f(n) \sim g(n)$ vol dir que quan n creix, les dues funcions esdevenen molt similars.

Recordant els grafs

Graf: $G = (V, E)$ a on V és el conjunt de vèrtexs, $|V| = n$.
$E \subset V \times V$ és el conjunt d'arestes, $|E| = m$,

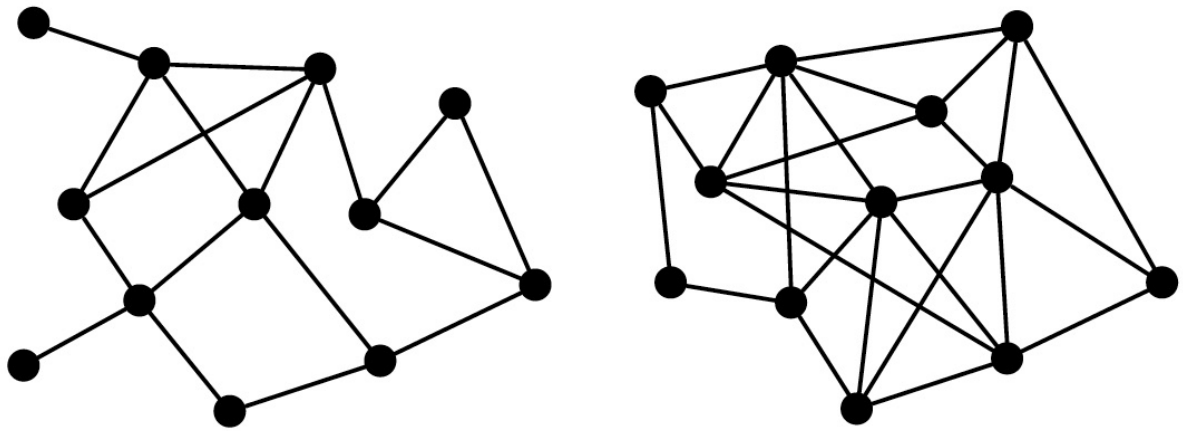
- Grafs: *dirigits* (digrafs) or *no-dirigits*.
- G no-dirigit es diu connex si hi ha un camí entre qualsevol 2 vèrtexs.
- Si G és connex, aleshores $\frac{n(n-1)}{2} \geq m \geq n - 1$.
- El grau d'un vèrtex v, $d(v)$ és el nombre d'arestes incidents a v.
- Una clica (clique) K_n amb n vèrtexs és un graf complert.

Digrafs

- Grafs amb arestes dirigides.
- El concepte de connectivitat en digrafs és el de connectivitat forta (strongly connected): hi ha un camí entre qualsevol 2 vèrtexs.
- Un digraf té com a màxim $n(n-1)$ arestes.

Densitat

Un graf G amb $|V| = n$ vèrteos es diu que és dens si $|E| = \Theta(n^2)$;
Es diu que és espars si $|E| = o(n^2)$.



Estructura de dades per a grafs.

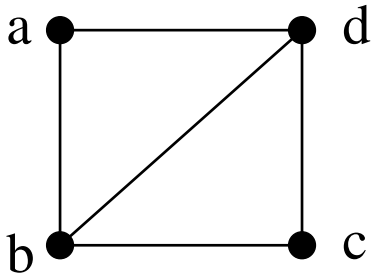
Els grafs són una de les eines més importats per a simular moltes situacions de la vida real.
Es necessiten maneres per a enmagatzemar i manipular de manera eficient els grafos.

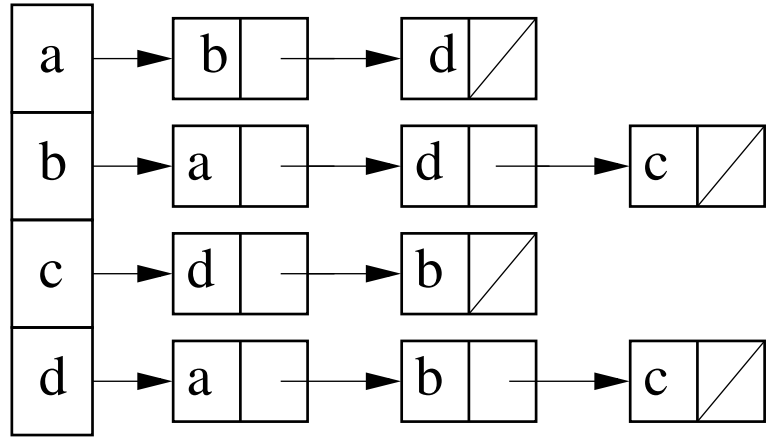
Sigui G un graf amb $V = \{1, 2, \ldots, n\}$. Les dues maneres més importants de representar grafs:

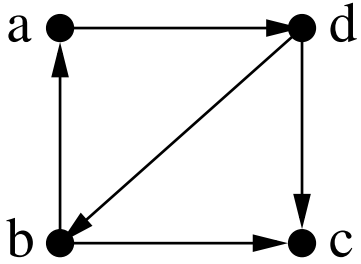
Llista d'adjacència

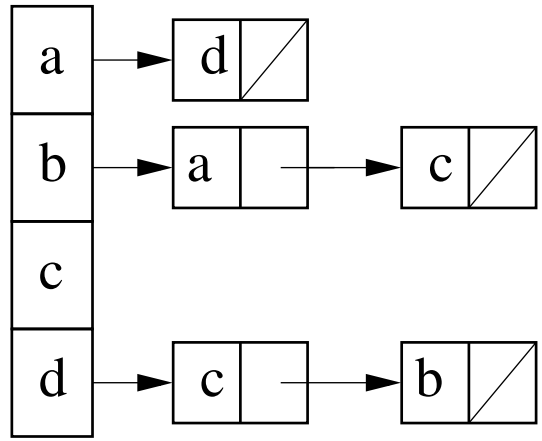
Matriu d'adjacència

Llista d'adjacència





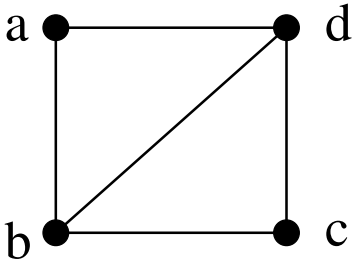




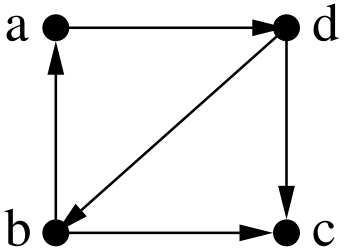
Matriu d'adjacència

Donat un digraf $\vec{G} = (V, \vec{E})$, definim la seva matriu d'adjacència $n \times n$:

$$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{if } (i,j) \notin E. \end{cases}$$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Matriu d'adjacència

- Si G no és dirigit, la matriu d'adjacència és simètrica
- Si A és la matriu d'adjacència de G, aleshores A^2 dóna tots el nombre de camins amb longitud $= 2$ a G.
- Per a graphs amb pesos, $w_{i,j}$ a cada $(i, j) \in E$, la matriu d'adjacència té w_{ij} a la posició i, j.
- L'us de la matriu d'adjacència per representar un graf permet utilitzar L'àlgebra de matrius.

Comparació entre la representació amb llistes i amb matrius

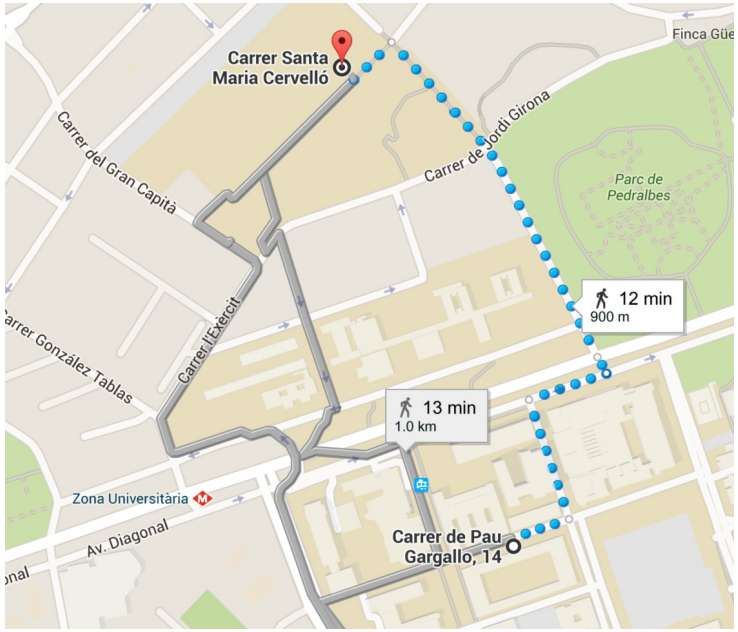
- Llistes utilitzen únicament un registre per vèrtex i un registre per aresta . Cada registre necessita 2×64 bits, per tant $128m$ bits $= \Theta(m)$.
- Matriu necessita $n \times n$ entrades, cadascuna pot ser de 1 bit: $\Theta(n^2)$ bits. Si el grafs te pesos, aleshores es necessita $64n^2$ bits.
- En grafs sense pesos, la representació amb matriu es millor per grafs densos, la representació per llistes és millor per grafs esparsos.

Nombre passos (complexitat):

- Afegir una aresta a G: tant l'estructura de dades de matrius com les llistes necessiten $\Theta(1)$.
- "Query" si G te una aresta d'u a v:
  Matriu: $\Theta(1)$.
  Llista $O(n)$
- Explorar tots els veïns del vèrtex v:
  Matriu: $\Theta(n)$
  Llista: $O(|d(v)|)$

Searching a graph: Breadth First Search

1. start with vertex v, visit and list all their neighbors at distance=1
2. then all their neighbors at distance 2 from v.
3. Repeat until all vertices visited



BFS use a QUEUE, (FIFO) cua

Problem: If unknowingly the BFS revisits a vertex, the algorithm could yield the wrong notion of distance.
The solution to avoid that loop is to label each vertex we visit for first time, and ignore it when we revisit it.

Searching a graph: Depth First Search

explore

1. From current vertex, move to another
2. Until you get stuck
3. Then backtrack till new place to explore.

DFS use a STACK, (LIFO) pila

Time Complexity of DFS and BFS

- DFS:

For undirected and directed graphs: $O(|V| + |E|)$
In the case of sparse graphs $T(n) = O(n)$
For the case of a dense graph $T(n) = O(n^2)$

- BFS:

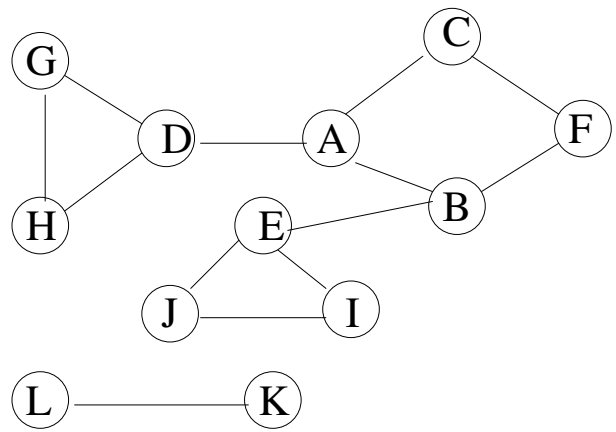
For undirected and directed graphs: $O(|V| + |E|)$

Therefore, the complexity of both procedures is linear in the size of the graph.

Connected components un undirected graphs.

Undirected Connected Components
INPUT: undirected graph G
QUESTION: Find if G is connected (if there is a path between any pair of vertices in $V(G)$.

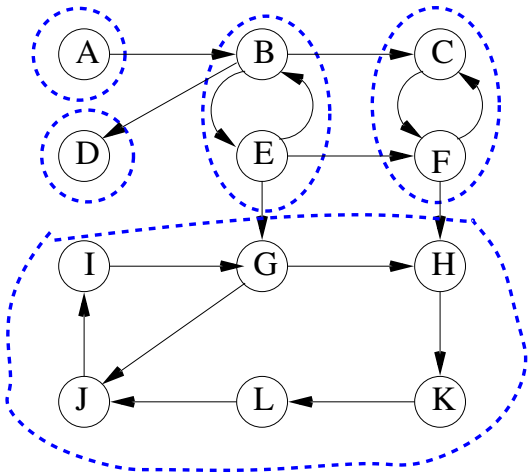
To find connected components in G apply DFS and count how many times **explore** is called. each time DFS calls **explore** on a vertex, it yields exactly the connected component to which the vertex belongs.

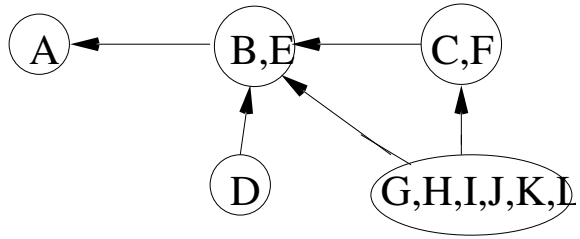


The problem can be solved in $O(|V| + |E|)$.

Strongly connected components in a digraph

Every digraph is a *directed acyclic graph (dag)* of its strongly connected components.





Complexity strongly connected components: $T(n) = O(|V| + |E|)$