Anàlisi i comparació de seqüències biomoleculars

Francesc Rosselló

Curs 2007/08

Tema 0:

Introducció

Què és la biologia computacional?

- La biologia computacional és l'aplicació de mètodes computacionals per resoldre problemes posats per la biologia

Què és la biologia computacional?

- La biologia computacional és l'aplicació de mètodes computacionals per resoldre problemes posats per la biologia
  - Biologia computacional molecular: Tracta problemes relacionats amb el maneig i l'anàlisi de dades en biologia molecular
  - Biologia computacional de sistemes: Tracta problemes relacionats amb la simulació i l'anàlisi de xarxes d'interaccions biològiques
  - ...

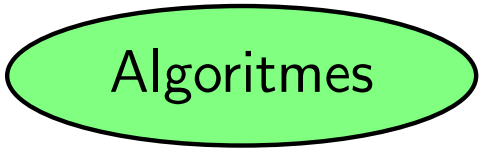
Què és la biologia computacional?

- La biologia computacional és l'aplicació de mètodes computacionals per resoldre problemes posats per la biologia
- La bioinformàtica és la implementació informàtica d'eines desenvolupades per la biologia computacional per a la seva aplicació sobre dades reals

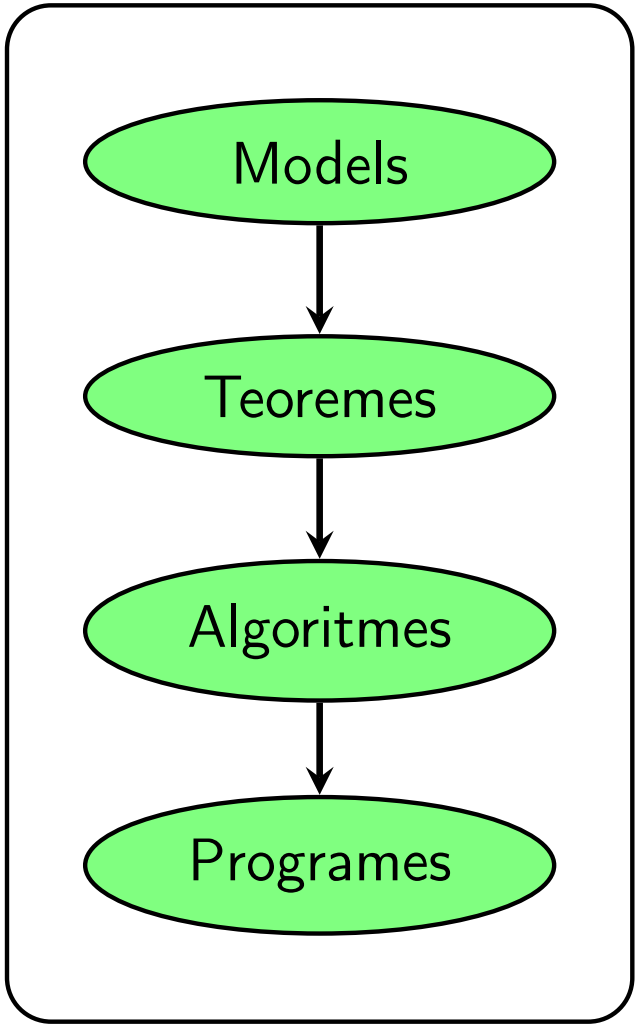
Què és la biologia computacional?

- La biologia computacional és l'aplicació de mètodes computacionals per resoldre problemes posats per la biologia
- La bioinformàtica és la implementació informàtica d'eines desenvolupades per la biologia computacional per a la seva aplicació sobre dades reals
- No les separarem, i emprarem els dos termes de manera més o manco indistinta

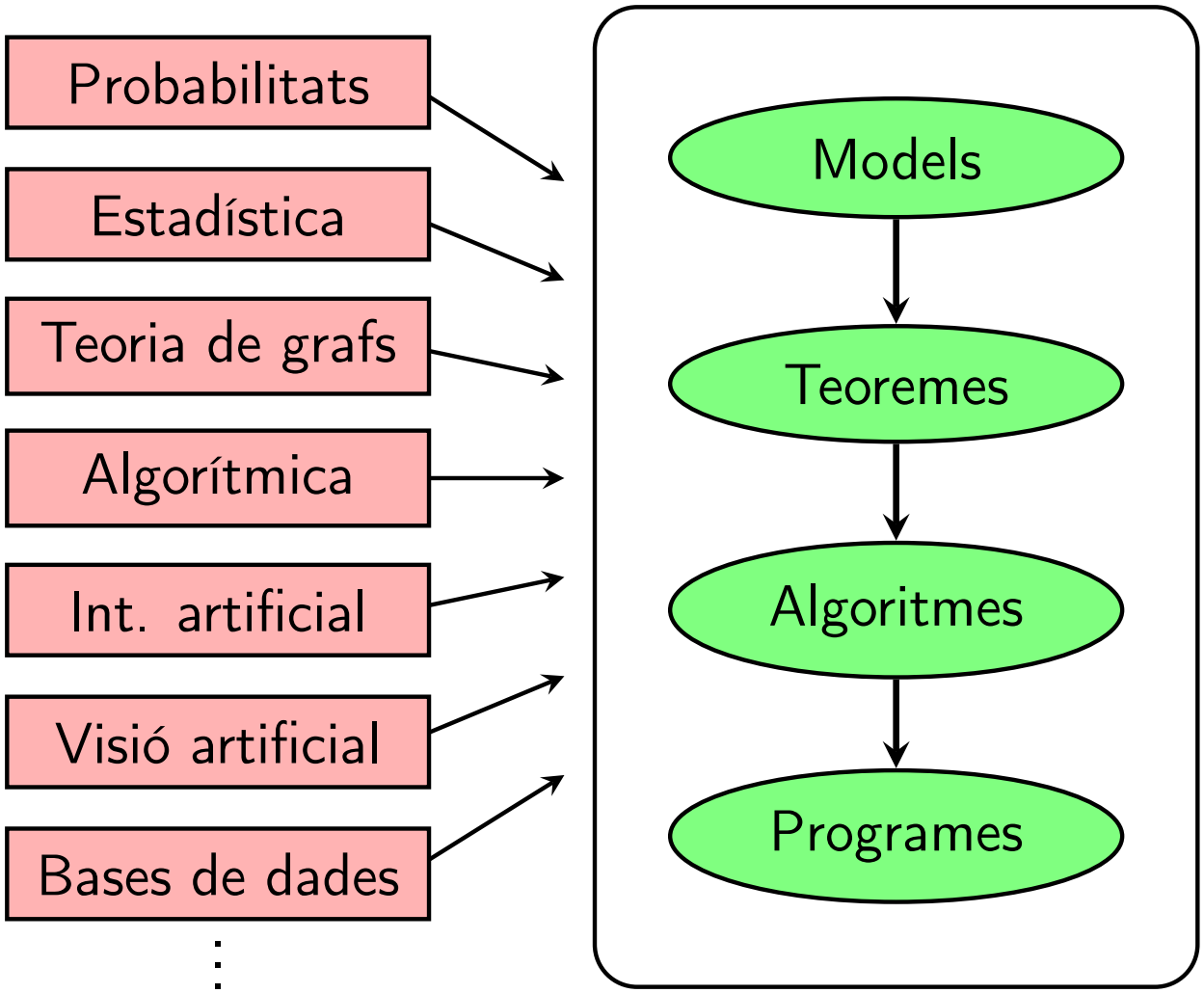
Què és la biologia computacional?



Què és la biologia computacional?



Què és la biologia computacional?



La bioinformàtica està de moda

Inicis de la biologia computacional:

- M. Dayhoff (1965): primera 'base de dades' de seqüències de proteïnes (65)
- Needleman-Wunsch (1970): primer algoritme en biologia computacional

i...

any	nombre de publicacions	
Fins 1990	1	
1991–1995	33	6.6 publ/any
1996–2000	1411	282 publ/any
2001-2005	8263	1653 publ/any
2006–2007	10048	

Resultats de cerca global a PubMed amb paraula clau `"bioinformatics" or "computational biology"`

La bioinformàtica està de moda

Inicis de la biologia computacional:

- M. Dayhoff (1965): primera 'base de dades' de seqüències de proteïnes (65)
- Needleman-Wunsch (1970): primer algoritme en biologia computacional

i...

El motiu

Els avenços en biotecnologia han portat un creixement exponencial en el descobriment d'informació genòmica i biomolecular

- Més de 700 genomes seqüenciats
- Gairebé 100 Gb en seqüències d'ADN
- Més de 300,000 seqüències de proteïnes anotades
- Més de 40,000 estructures de proteïnes
- Més de 50,000 rutes metabòliques

Els biòlegs necessiten mètodes computacionals per analitzar aquesta informació

ADN

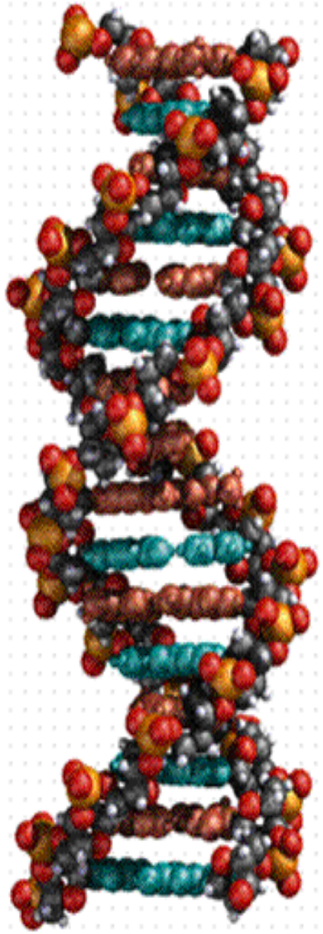
L'ADN és la molècula que conté la informació per al desenvolupament i funcionament dels éssers vius

Està format per dues cadenes de nucleòtids o ' bases':

A: Adenina C: Citosina
G: Guanina T: Timina

Aquestes cadenes formen una doble hèlix, i són complementàries

A s'enganxa a T
C s'enganxa a G

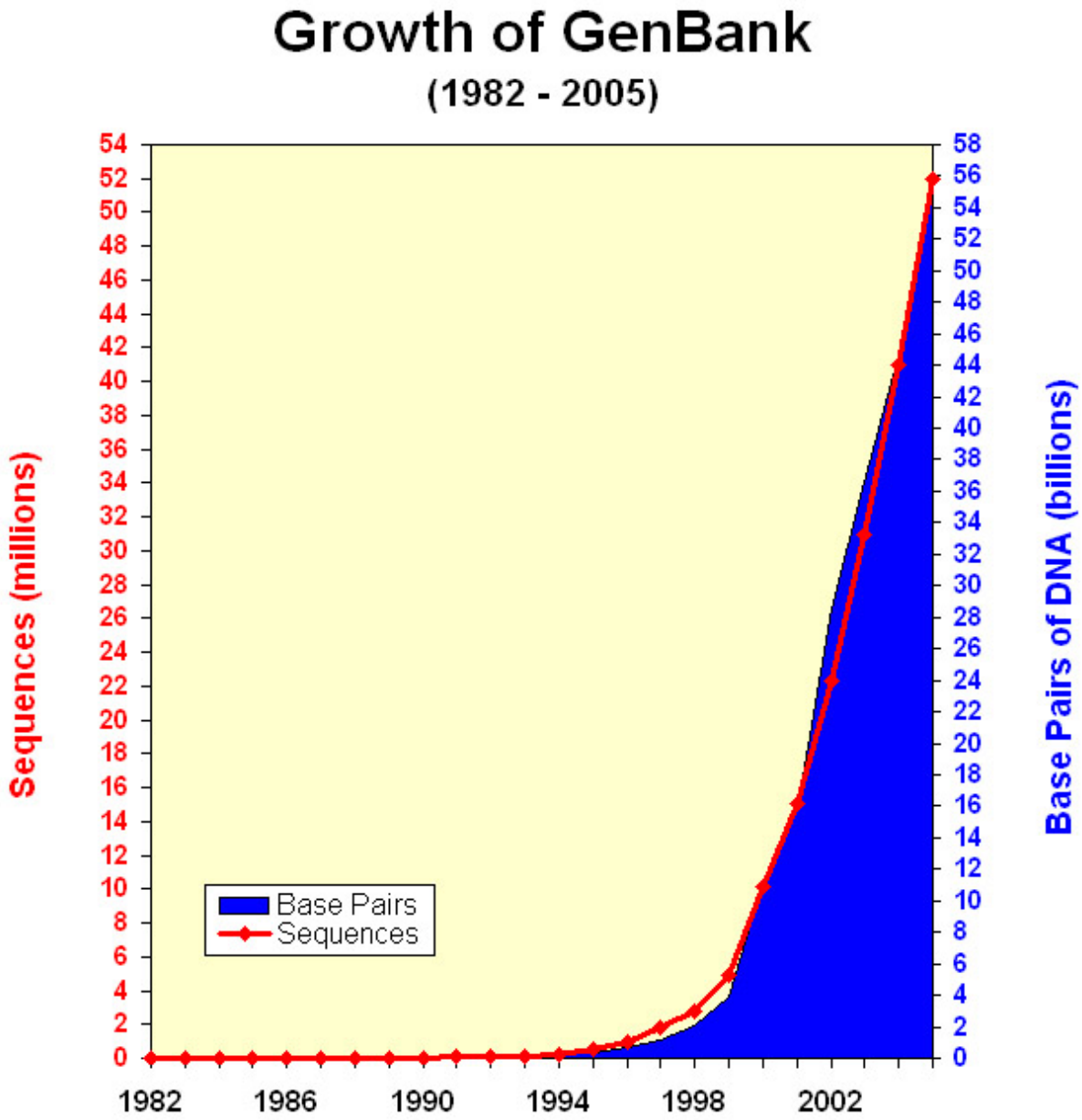


ADN

Podem modelar una molècula d'ADN com una paraula sobre l'alfabet A,C,G,T

...tgcatgcggctatgctaatgcatgcggctatgctaagctgggatccgatg
acaatgcatgcggctatgctaatgcatgcggctatgcaagctgggatccgatg
gactatgctaagctgggatccgatgacaatgcatgcggctatgctaatgaatg
gtcttgggatttaccttggaatgctaagctgggatccgatgacaatgcatgcg
ctatgctaatgaatggtcttgggatttaccttggaatatgctaatgcatgcgg
ctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgcatgcg
gctatgcaagctgggatccgatgactatgctaagctgcggctatgctaatgca
tgcggctatgctaagctgggatccgatgacaatgcatgcggctatgctaatgc
atgcggctatgcaagctgggatcctgcggctatgctaatgaatggtcttggga
tttaccttggaatgctaagctgggatccgatgacaatgcatgcggctatgcta
atgaatggtcttgggatttaccttggaatatgctaatgcatgcggctatgcta
agctgggaatgcatgcggctatgctaagctgggatccgatgacaatgcatgcg
gctatgctaatgcatgcggctatgcaagctgggatccgatgactatgctaagc
tgcggctatgctaatgcatgcggctatgctaagctcatgcggctatgctaagc
tgggaatgcatgcggctatgctaagctgggatccgatgacaatgcatgcg...

ADN


Growth of GenBank
(1982 - 2005)
Sequences (millions)
Base Pairs of DNA (billions)
Base Pairs
Sequences
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
1982
1986
1990
1994
1998
2002

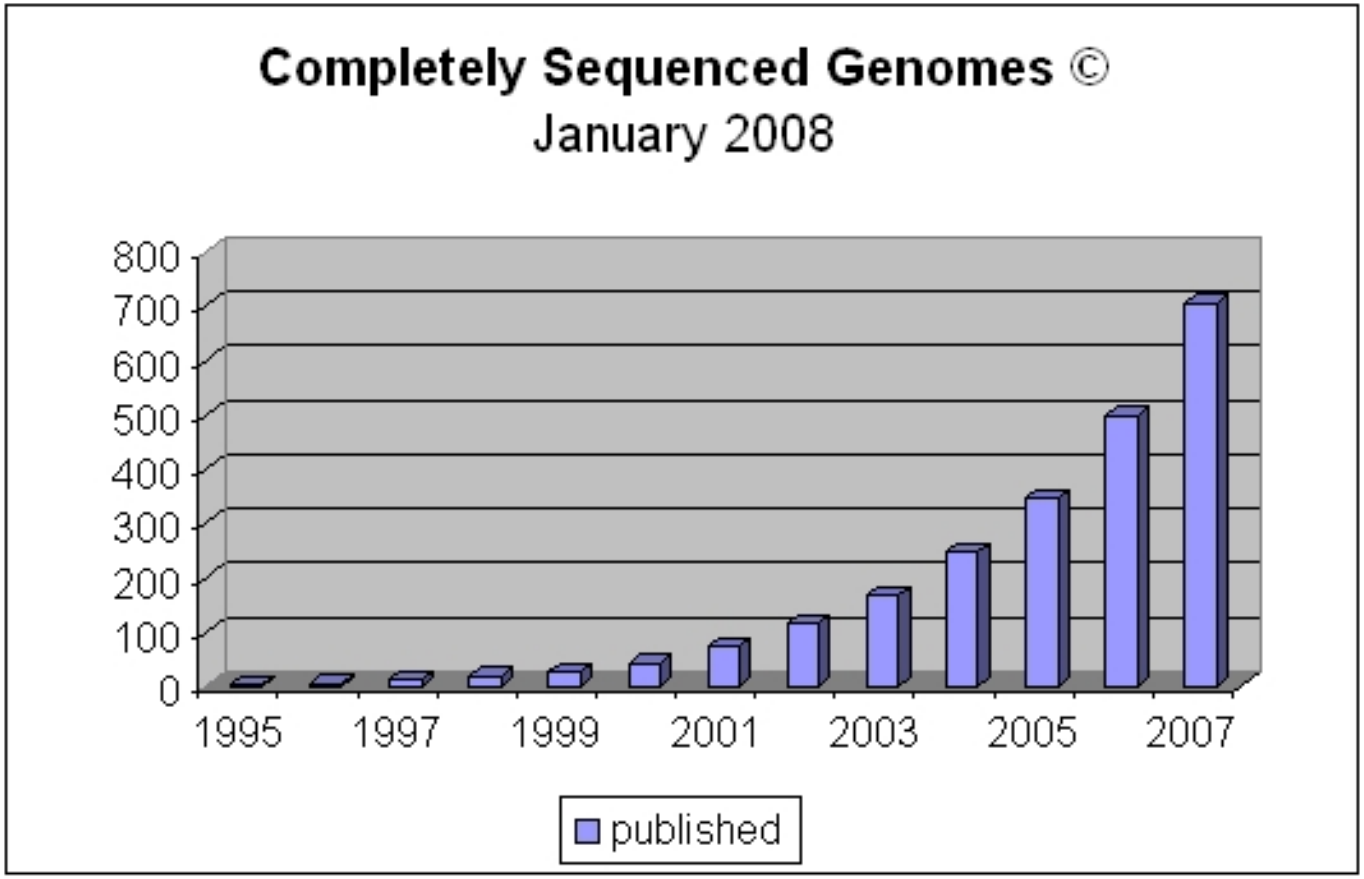
Genomes

Totes les cèl·lules d'un organisme tenen (haurien de tenir) el mateix ADN

El genoma d'un organisme és el conjunt complet de molècules d'ADN d'una cèl·lula

Els gens són seqüències d'ADN específiques que codifiquen les instruccions per produir proteïnes o ARN

Genomes


Completely Sequenced Genomes ©
January 2008
800
700
600
500
400
300
200
100
0
1995
1997
1999
2001
2003
2005
2007
published

Genomes

	Longitud	Nombre de gens
HIV	9700	9
Myc. genitallum	580 073	483
E. coli	4.6×10^6	5416
Arròs	3.9×10^8	37 544
Cans	2.4×10^9	19 300
Dro. melanogaster	1.3×10^8	13 379
Humans	3.3×10^9	$\sim 20\,500$[1]
Algunes amebes	6.9×10^{11}	11 000

[1] 01/2008

Proteïnes

Les proteïnes són part essencial dels organismes vius i participen en tots els processos cel·lulars

Són cadenes d'aminoàcids. Hi ha 20 aminoàcids.

Per tant, una proteïna pot ser modelada com una paraula sobre un alfabet de 20 lletres:

A,R,N,D,C,E,Q,G,H,I,L,K,M,F,P,S,T,W,Y,V

(A: Alanina, R: Arginina, N: Asparagina, etc.)

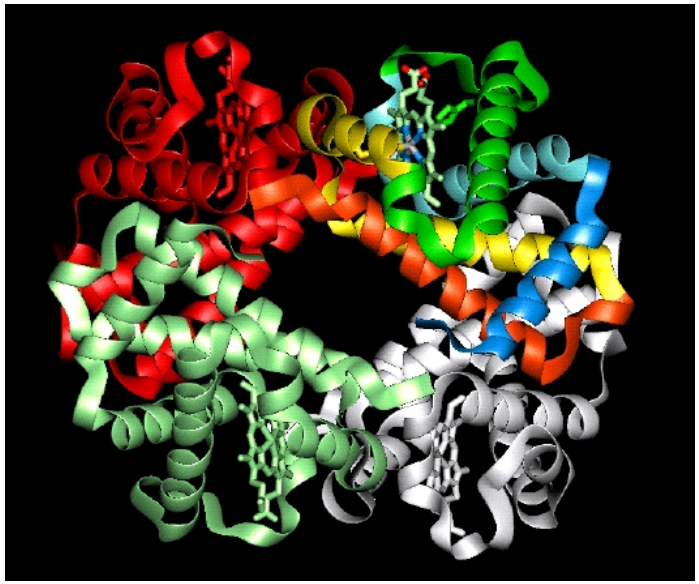
```
...KKTILAIAIPALFASAANAAVIYDKDGTTFDVYGRVQANYYGDTNEADSTA
ASGYKDVDGELKGSSRLGWSGKIALNNTWSGIAKTEWQVSAENSANKFDSRHIY
VGFDGTQYGKVIFGQTDTAFYDVLEPTDIFNEWGSEGNFYDGRQEGQVIYSNAI
GGFKGKVSYQTNDDQAVKVADVAGGIKTTVFPDVKRKYAYAAAVGYDFDFG...
```

Proteïnes

El proteïnes es recargolen dins les cèl·lules formant estructures 3D complicades que determinen la seva funció

En general, aquesta estructura és determinada per la seqüència d'aminoàcids

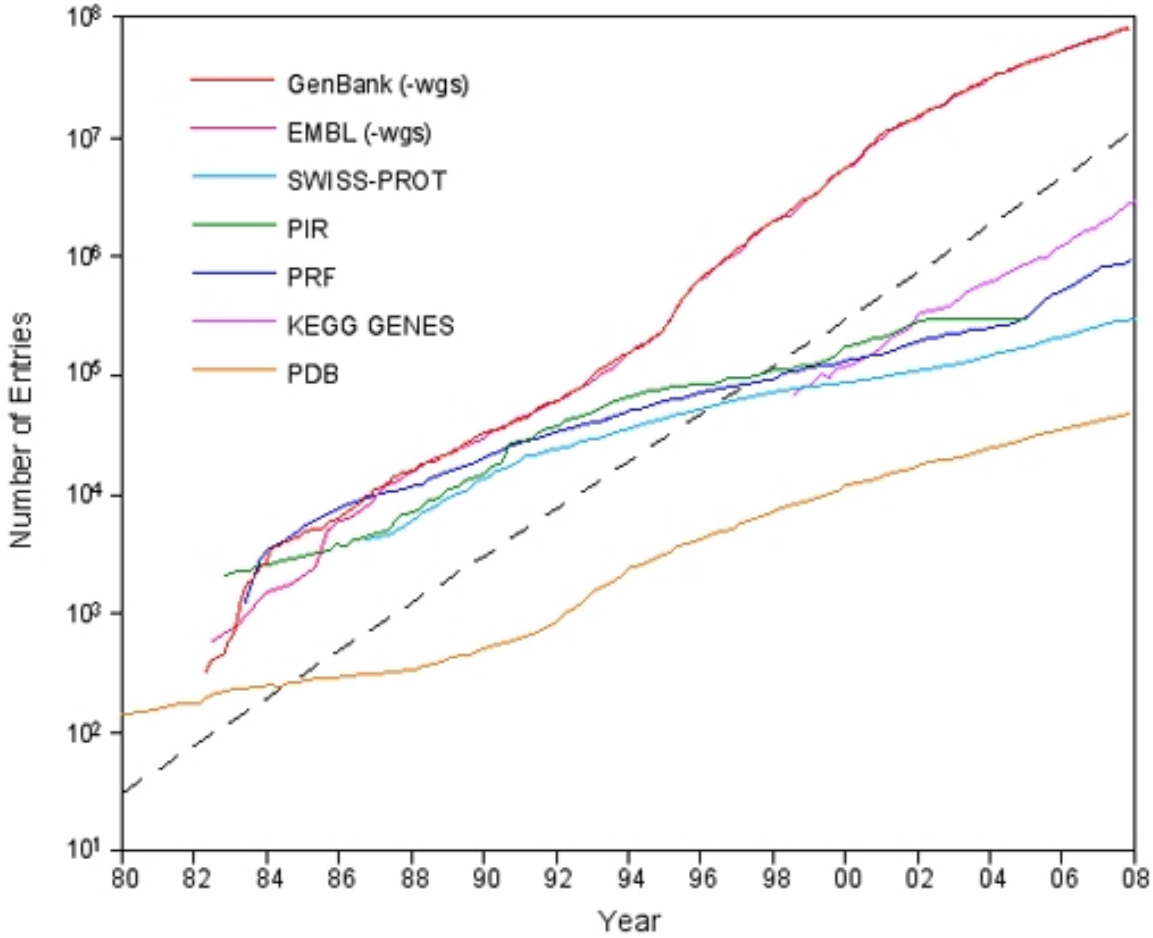
Aquestes estructures es modelen com a matrius de distàncies, com a grafs, com a camins en reticles, etc.



Estructura d'una hemoglobina

Proteïnes

Creixement de diferents bases de dades



Per què la biologia computacional?

La natura de tipus discret dels objectes i la informació que es manegen en biologia molecular (seqüències, estructures, etc.) és ideal per a la seva anàlisi emprant tècniques de matemàtica discreta

La quantitat d'informació i la incertesa en aquesta obliguen a l'ús de probabilitats i estadística

La mida i complexitat de les dades que es manegen en biologia molecular fan impossible l'anàlisi sense ordinadors

Oportunitats en biologia computacional

La biologia té més de 500 anys de problemes en què podem treballar (Don Knuth)

Les matemàtiques són el proper microscopi per a la biologia, però millor; la biologia és la propera física per a les matemàtiques, però millor (Joel Cohen)

Aplicacions de la bioinformàtica

Ha esdevingut una eina més per al biòleg:

- En la generació de dades:
  - Seqüenciat de genomes
- En l'anàlisi de les dades:
  - Cerca de gens
  - Anàlisi de la transcripció de l'ADN
  - Anàlisi de l'expressió gènica
  - Predicció de l'estructura de molècules d'ARN i proteïnes
  - Predicció de la funció de molècules d'ARN i proteïnes
  - ...
- En l'aplicació d'aquesta informació:
  - Comprensió de l'evolució
  - Reparació de gens
  - Disseny de medicaments

L'estat de l'art

...varia segons el problema

- Problemes ben entesos, amb algoritmes 'definitius' i programari a l'abast:

  Alineament de dues seqüències, cerca de llocs de restricció, ...

- Problemes avançats, amb esperança de solució:

  Cerca de gens, comparació d'estructures, filogenètica ...

- La zona gris (coneixement limitat, recerca activa, però solució molt llunyana):

  Alineament múltiple, predicció d'estructures, ...

Aquest curs

'Programa'

- Alineament de 2 seqüències
- Alineament múltiple
- Anàlisi de seqüències
- Filogenètica
- + l'anàlisi de dades necessari per poder interpretar resultats

Tema 1:

Programació dinàmica

Introducció

La programació dinàmica és una tècnica algorítmica que generalitza la recursió usual sobre $\mathbb{N}$.

Introducció

Algoritmes que calculen F_n per recursió

```
Algoritme FIB(n)
begin
if n = 0 then
   return 1
if n = 1 then
   return 1
a := n − 1
b := n − 2
f1 := FIB(n − 1)
f2 := FIB(n − 2)
return f1 + f2
end
```

Introducció

Algoritmes que calculen F_n per recursió

```
Algoritme FIB(n)
begin
if n = 0 then
   return 1
if n = 1 then
   return 1
a := n − 1
b := n − 2
f1 := FIB(n − 1)
f2 := FIB(n − 2)
return f1 + f2
end
```

```
Algoritme FIB2(n)
begin
F := (); i := 0
for i = 0 to n do
   if i = 0 then
      F[i] := 1
   else
      if i = 1 then
         F[i] := 1
      else
         F[i] := F[i − 1] + F[i − 2]
return F[n]
end
```

Introducció

La programació dinàmica és una tècnica algorítmica que generalitza la recursió usual sobre $\mathbb{N}$.

Introducció

La programació dinàmica és una tècnica algorítmica que generalitza la recursió usual sobre $\mathbb{N}$.

La base de la programació dinàmica és:

- Reducció del problema en subproblemes sobre instàncies 'anteriors'
- Resolució de tots aquests subproblemes
- Resolució del problema original a partir d'aquestes

Útil en problemes de cerca de solució òptima i amb els subproblemes hiper-utilitzats

Aquí només emprarem una versió senzilla de la PD.

El problema del canvi

Objectiu: Repartir una certa quantitat M de doblers en monedes d'un certs valors $c_1, \ldots, c_d$ prefixats, emprant el nombre mínim de monedes

Input: Un nombre $M \in \mathbb{N}$ i un vector de d nombres $c = (c_1, \ldots, c_d) \in \mathbb{N}^d$ amb $c_1 > \ldots > c_d$

Output: Un vector $(i_1, \ldots, i_d) \in \mathbb{N}^d$ tal que

$$i_1 c_1 + \cdots + i_d c_d = M$$

i $i_1 + \cdots + i_d$ mínim amb aquesta propietat, si existeix (un canvi òptim de M)

El problema del canvi

El que solen fer a les botigues (*greedy*: a cada moment tria el que sembla millor en aquell moment)

```
Real(M,c,d)
```
begin
$r := M$; $k := 1$
for $k = 1$ **to** d **do**
  $i_k := \lfloor r/c_k \rfloor$
  $t := r - i_k$
return $(i_1, \ldots, i_d)$
end

El problema del canvi

El que solen fer a les botigues (*greedy*: a cada moment tria el que sembla millor en aquell moment)

```
Real(M,c,d)
```
begin
$r := M$; $k := 1$
for $k = 1$ **to** d **do**
  $i_k := \lfloor r/c_k \rfloor$
  $t := r - i_k$
return $(i_1, \ldots, i_d)$
end

Incorrecte: Provau $M = 30$, $c = (25, 10, 1)$

El problema del canvi

```
ForçaBruta(M,c,d)
begin
sumamenoris := ∞
(i_1, ..., i_d) = (0, ..., 0)
for (i_1, ..., i_d) from (0, ..., 0)
                     to (⌊M/c_1⌋, ..., ⌊M/c_d⌋) do
   sumatotal := i_1 c_1 + ··· + i_d c_d
   if sumatotal = M then
      sumais := i_1 + ··· + i_d
      if sumais < sumamenoris then
         sumamenoris := sumais
         canviòptim := (i_1, ..., i_d)
return canviòptim
end
```

El problema del canvi

```
ForçaBruta(M,c,d)
begin
sumamenoris := ∞
(i_1, ..., i_d) = (0, ..., 0)
for (i_1, ..., i_d) from (0, ..., 0)
                     to (⌊M/c_1⌋, ..., ⌊M/c_d⌋) do
   sumatotal := i_1 c_1 + ··· + i_d c_d
   if sumatotal = M then
      sumais := i_1 + ··· + i_d
      if sumais < sumamenoris then
         sumamenoris := sumais
         canviòptim := (i_1, ..., i_d)
return canviòptim
end
```

Correcte si existeix canvi (fàcil d'arreglar), però massa costós $O(M^d/c_1 \cdots c_d)$

El problema del canvi

Observació clau: Si a un canvi òptim de M li llevam una moneda de valor c_i, tenim un canvi òptim de $M - c_i$

El problema del canvi

Observació clau: Si a un canvi òptim de M li llevam una moneda de valor c_i, tenim un canvi òptim de $M - c_i$

$$\text{canviòptim}(M) = \text{òptim de} \begin{cases} \text{canviòptim}(M - c_1) + \delta_1 \\ \text{canviòptim}(M - c_2) + \delta_2 \\ \dots \\ \text{canviòptim}(M - c_d) + \delta_d \end{cases}$$

on $\delta_i = (\underbrace{0, \dots, 0, \overset{i}{1}, 0, \dots, 0}_{d})$

El problema del canvi

```
Recursiu(M,c,d)
begin
if M = 0 then
   return (0,...,0)
else
   sumamenoris := ∞; i = 1
   for i = 1 to d
      if M ⩾ c_i then
         canvi := Recursiu(M − c_i, c, d)
         sumacanvi := suma de les entrades de canvi
         if sumacanvi + 1 < sumamenoris then
            canviòptim := canvi + δ_i
   return canviòptim
end
```

El problema del canvi

```
Recursiu(M,c,d)
begin
if M = 0 then
   return (0, ..., 0)
else
   sumamenoris := ∞; i = 1
   for i = 1 to d
      if M ⩾ c_i then
         canvi := Recursiu(M − c_i, c, d)
         sumacanvi := suma de les entrades de canvi
         if sumacanvi + 1 < sumamenoris then
            canviòptim := canvi + δ_i
   return canviòptim
end
```

Correcte si existeix canvi (fàcil d'arreglar), però recalcula massa

El problema del canvi

```
Recursiu(M,c,d)
begin
if M = 0 then
   return (0,...,0)
else
   sumamenoris := ∞; i = 1
   for i = 1 to d
      if M ⩾ c_i then
         canvi := Recursiu(M − c_i, c, d)
         sumacanvi := suma de les entrades de canvi
         if sumacanvi + 1 < sumamenoris then
            canviòptim := canvi + δ_i
   return canviòptim
end
```

Correcte si existeix canvi (fàcil d'arreglar), però recalcula massa; millor guardar i invocar

El problema del canvi

```
CanviPD(M,c,d)
begin
canviòptim(0) := (0,...,0)
sumacanviòptim(0) := 0; m := 1
for m = 1 to M do
   canviòptim(m) := (error,...,error)
   sumacanviòptim(m) := ∞; i := 1
   for i = 1 to d
      if m ⩾ c_i then
         if sumacanviòptim(m − c_i) + 1 < sumacanviòptim(m)
                                                    then
            canviòptim(m) := canviòptim(m − c_i) + δ_i
            sumacanviòptim(m) := sumacanviòptim(m − c_i) + 1
return canviòptim(M)
end
```

El problema del canvi

```
CanviPD(M,c,d)
begin
canviòptim(0) := (0,...,0)
sumacanviòptim(0) := 0; m := 1
for m = 1 to M do
   canviòptim(m) := (error,...,error)
   sumacanviòptim(m) := ∞; i := 1
   for i = 1 to d
      if m ⩾ c_i then
         if sumacanviòptim(m − c_i) + 1 < sumacanviòptim(m)
                                                    then
            canviòptim(m) := canviòptim(m − c_i) + δ_i
            sumacanviòptim(m) := sumacanviòptim(m − c_i) + 1
return canviòptim(M)
end
```

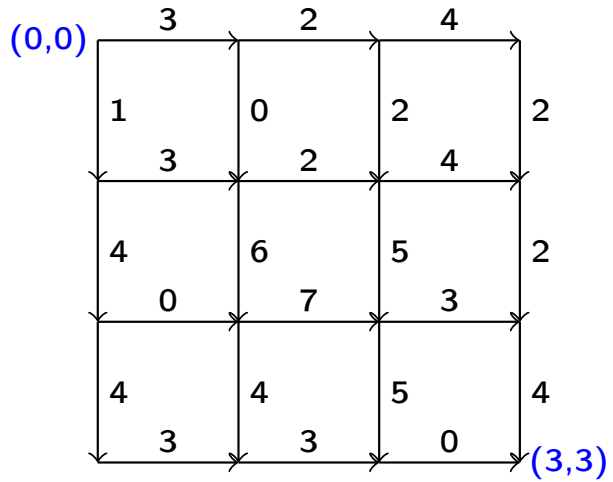
Cost: $O(Md)$

El problema del turista de Manhattan

Objectiu: Trobar el camí més llarg entre dos punts donats d'una graella amb pesos

Input: Una graella orientada G de mida $n \times m$ amb pesos w a les arestes i dos vèrtexos distingits $S = (0, 0)$ i $A = (n, m)$

Output: Un camí de pes màxim dins G de S a A



El problema del turista de Manhattan

```
Greedy(G,n,m)
begin
(x, y) := (0, 0)
camí := ((0, 0))
pestotal := 0
while x < n and y < m do
    if w((x, y), (x + 1, y)) ⩾ w((x, y), (x, y + 1)) then
        camí := (camí, (x + 1, y))
        pestotal := pestotal + w((x, y), (x + 1, y))
        (x, y) := (x + 1, y)
    else
        camí := (camí, (x, y + 1))
        pestotal := pestotal + w((x, y), (x, y + 1))
        (x, y) := (x, y + 1)
```

El problema del turista de Manhattan

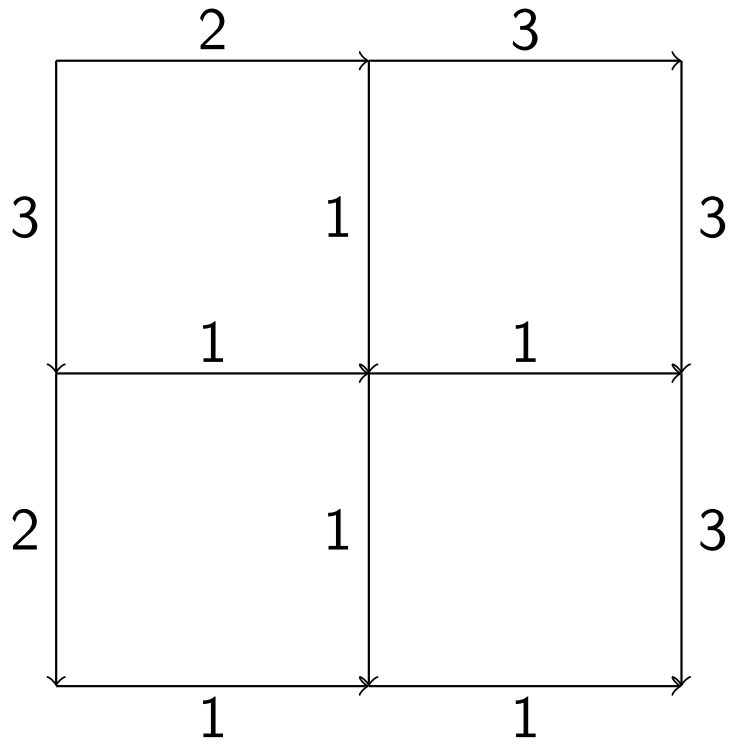
if $x = n$ **then**
  while $y < m$ **do**
    camí $:= (\text{camí}, (x, y+1))$
    pestotal $:=$ pestotal $+ w((x,y),(x,y+1))$
    $(x,y) := (x, y+1)$
else
  while $x < n$ **do**
    camí $:= (\text{camí}, (x+1, y))$
    pestotal $:=$ pestotal $+ w((x,y),(x+1,y))$
    $(x,y) := (x+1, y)$
output camí, pestotal
end

El problema del turista de Manhattan

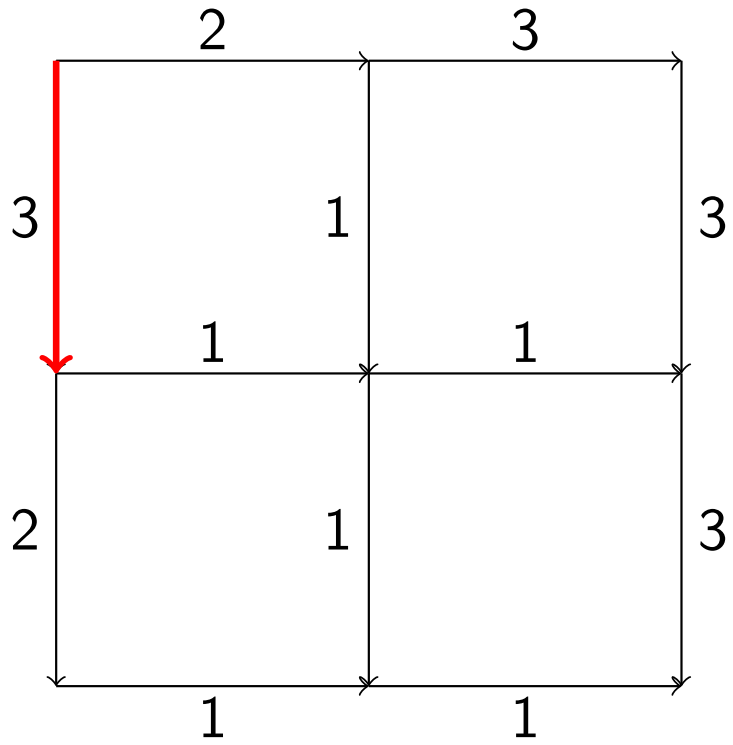
if $x = n$ **then**
  while $y < m$ **do**
    camí $:= (\text{camí}, (x, y+1))$
    pestotal $:=$ pestotal $+ w((x,y),(x,y+1))$
    $(x,y) := (x, y+1)$
else
  while $x < n$ **do**
    camí $:= (\text{camí}, (x+1, y))$
    pestotal $:=$ pestotal $+ w((x,y),(x+1,y))$
    $(x,y) := (x+1, y)$
output camí, pestotal
end

Incorrecte

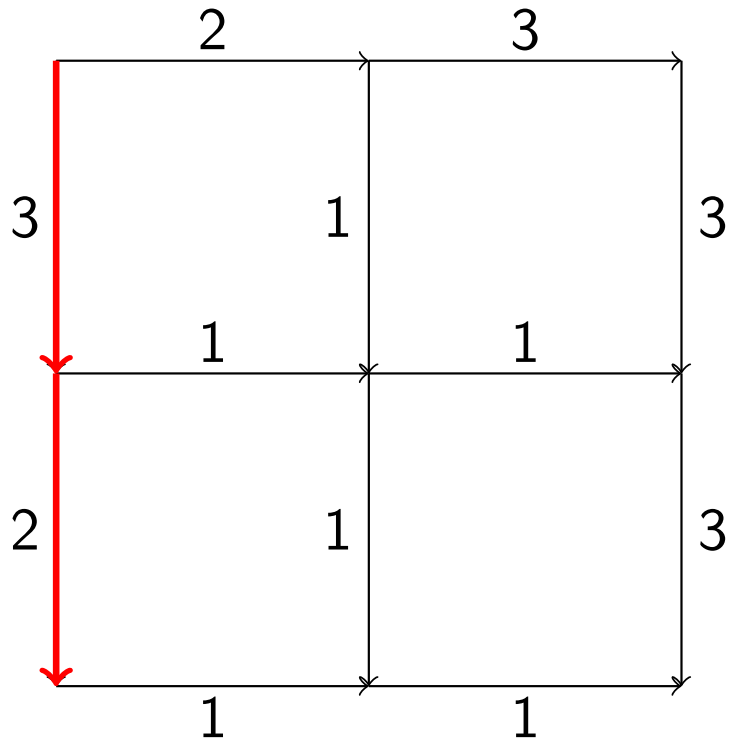
El problema del turista de Manhattan



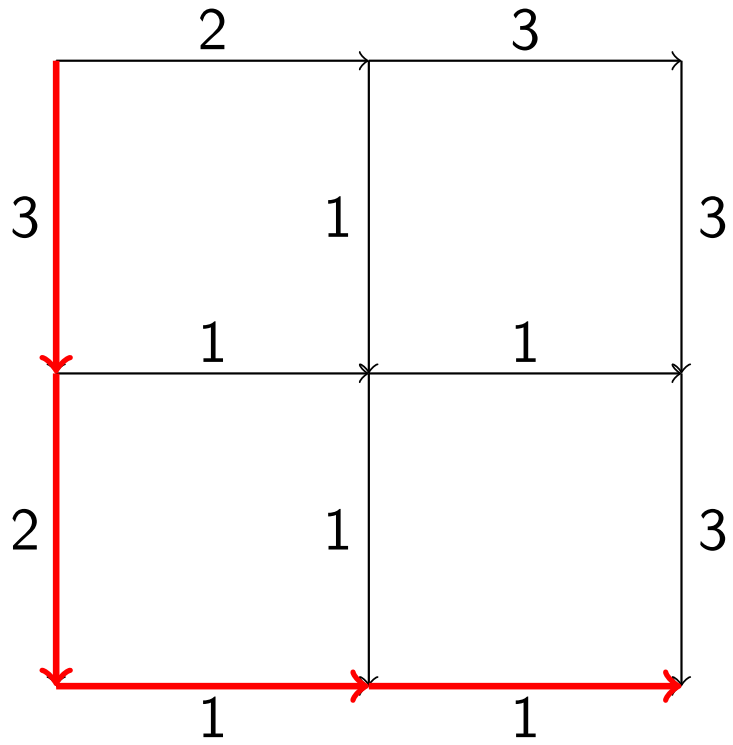
El problema del turista de Manhattan



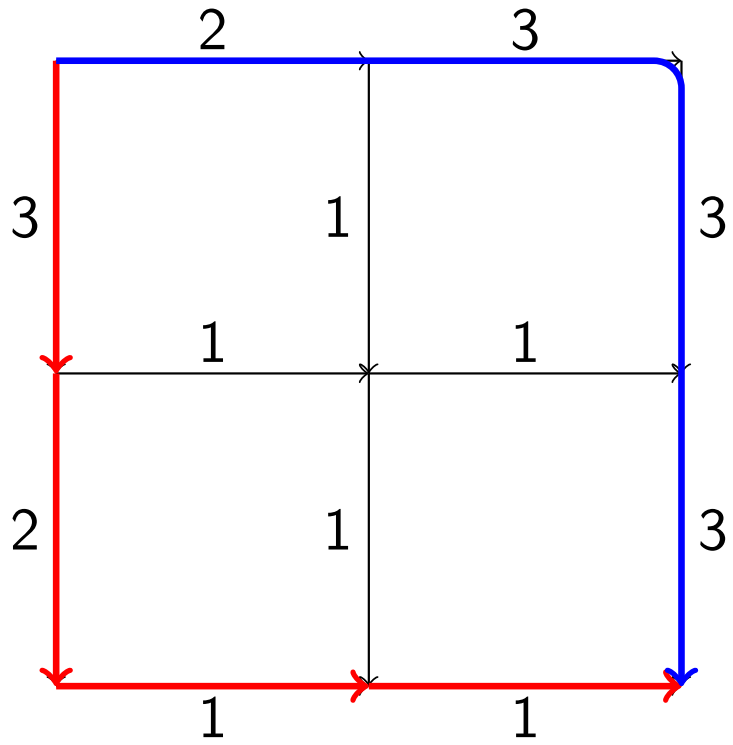
El problema del turista de Manhattan



El problema del turista de Manhattan



El problema del turista de Manhattan



El problema del turista de Manhattan

Hi ha també un algoritme de força bruta, que prova tots els camins de $(0, 0)$ a (n, m) i es queda amb el que tingui pes total més gran. Cost: $O(2^{n+m})$.

El problema del turista de Manhattan

Observació clau: Si un camí de pes màxim $(0,0) \rightsquigarrow (n,m)$ visita el node (i_0, j_0), aleshores el bocí $(0,0) \rightsquigarrow (i_0, j_0)$ és de pes màxim.

Si tenim un camí de pes màxim $(0,0) \rightsquigarrow (i_0, j_0)$, el camí $(0,0) \rightsquigarrow (i_0 - 1, j_0)$ o $(0,0) \rightsquigarrow (i_0, j_0 - 1)$ fins el node anterior a (i_0, j_0) és de pes màxim.

El problema del turista de Manhattan

El que farem serà calcular el pes màxim d'un camí de $(0,0)$ a cada (i,j), com a

$$\begin{aligned}&\text{pesmàximcamí}(i,j)\\ &\quad = \max\begin{cases}\text{pesmàximcamí}(i-1,j) + w((i-1,j),(i,j))\\ \text{pesmàximcamí}(i,j-1) + w((i,j-1),(i,j))\end{cases}\end{aligned}$$

anirem guardant aquests valors i a més quan es dóna el màxim

Al final, mirarem $\text{pesmàximcamí}(n,m)$ i com s'hi arriba

El problema del turista de Manhattan

El que farem serà calcular el pes màxim d'un camí de $(0,0)$ a cada (i,j), com a

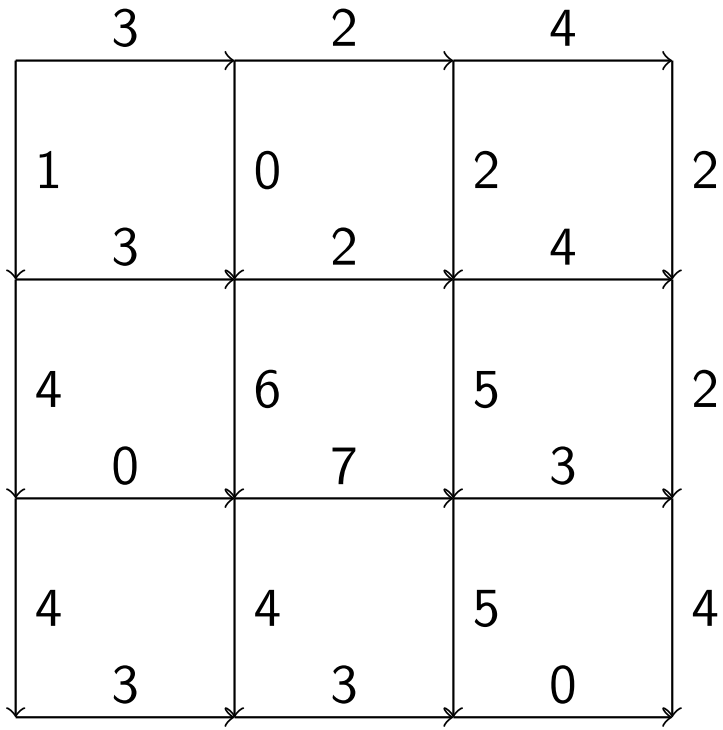
$$\begin{aligned}&\text{pesmàximcamí}(i,j)\\ &\quad = \max\begin{cases}\text{pesmàximcamí}(i-1,j) + w((i-1,j),(i,j))\\ \text{pesmàximcamí}(i,j-1) + w((i,j-1),(i,j))\end{cases}\end{aligned}$$

anirem guardant aquests valors i a més quan es dóna el màxim

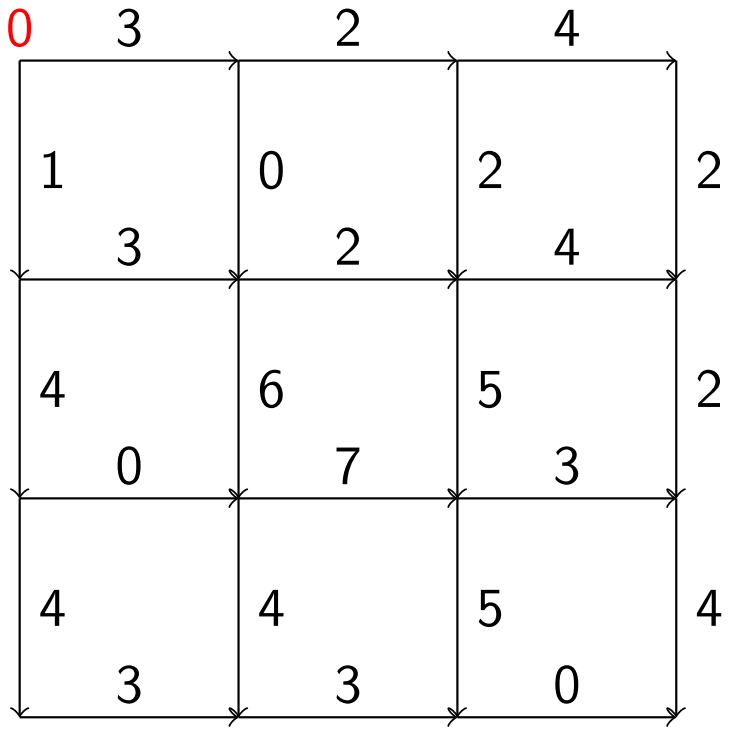
Al final, mirarem $\text{pesmàximcamí}(n,m)$ i com s'hi arriba

$s_{i,j} := \text{pesmàximcamí}(i,j)$

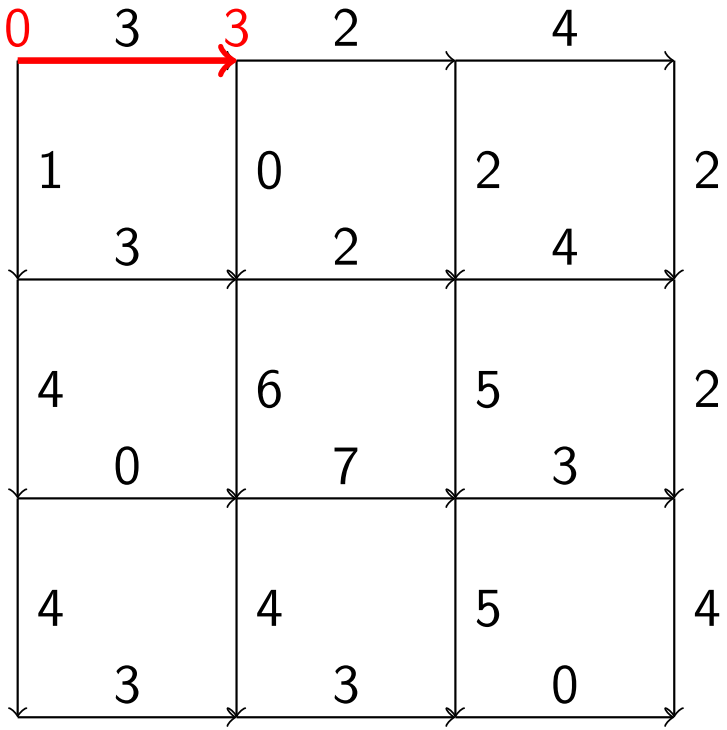
El problema del turista de Manhattan



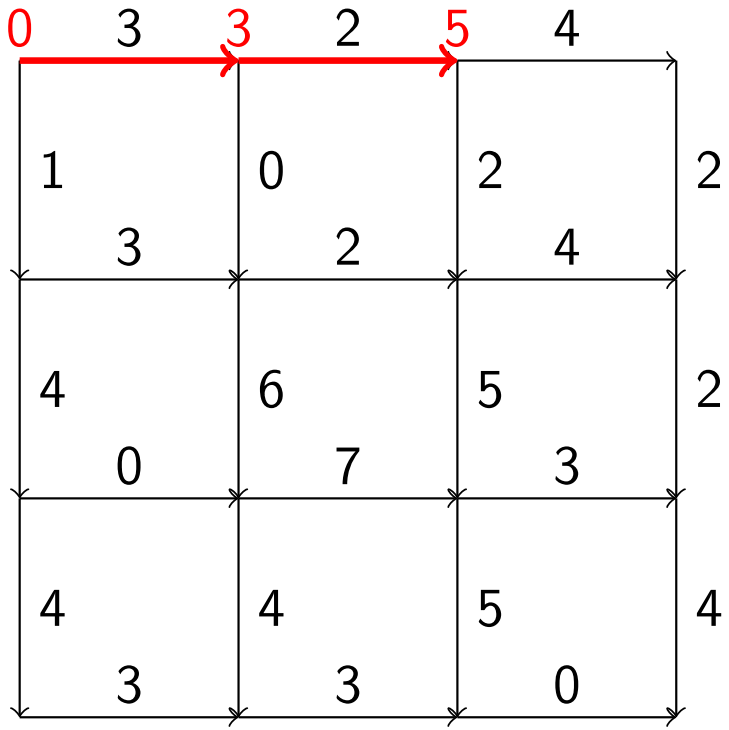
El problema del turista de Manhattan



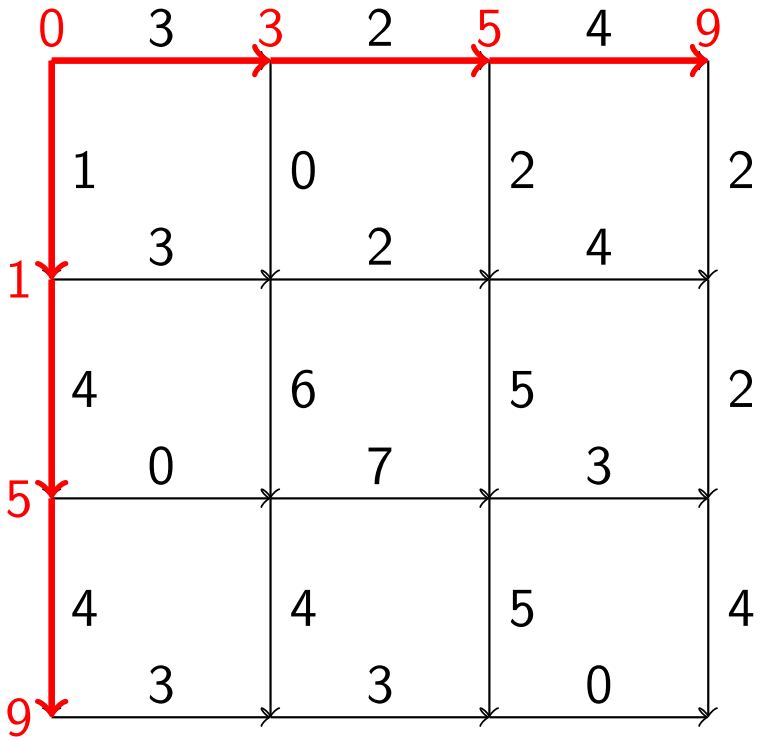
El problema del turista de Manhattan



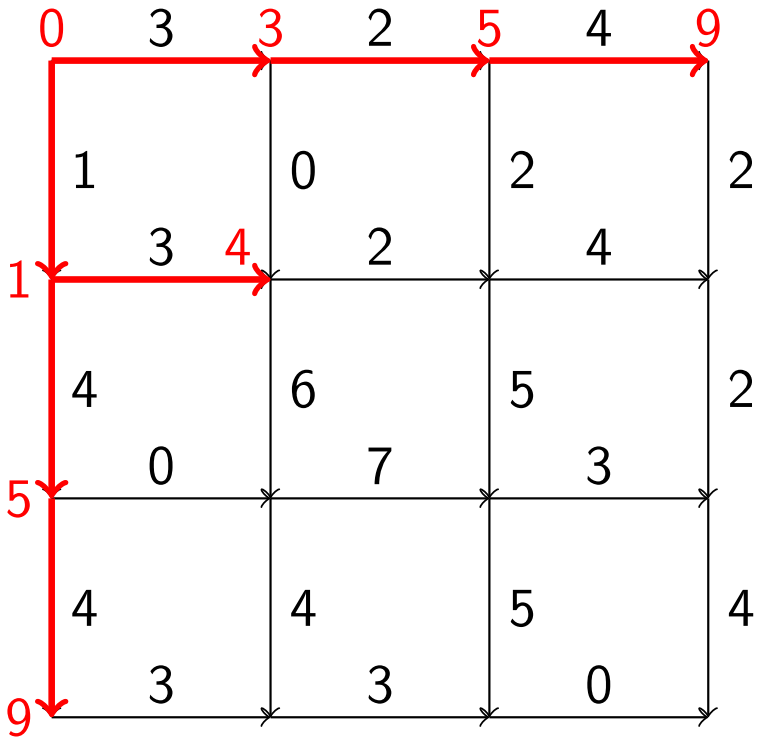
El problema del turista de Manhattan



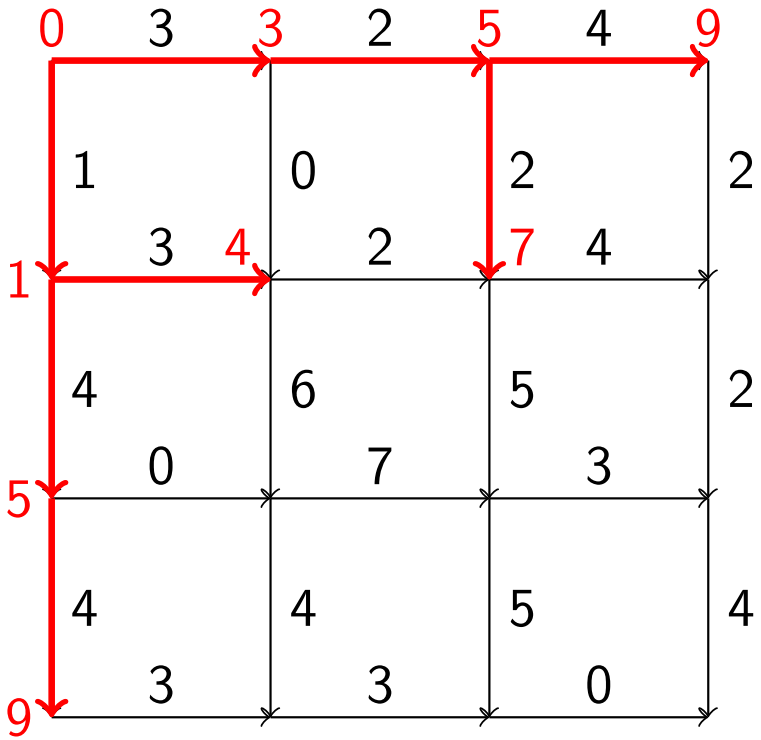
El problema del turista de Manhattan



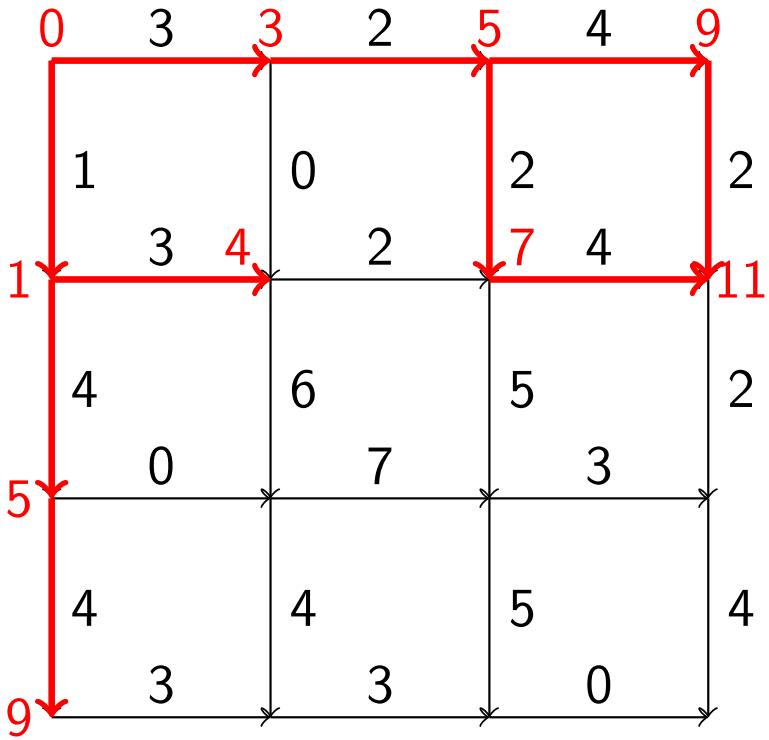
El problema del turista de Manhattan



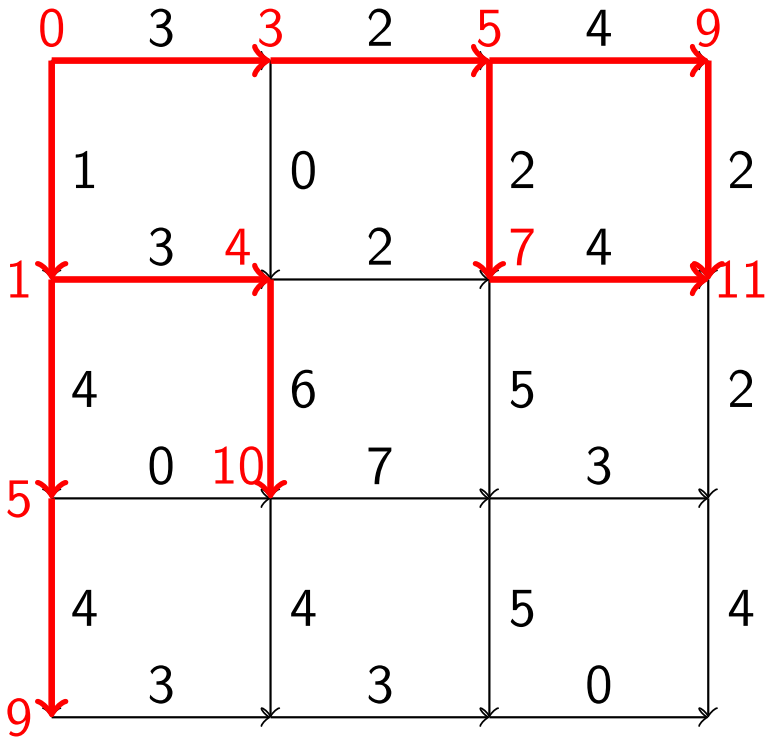
El problema del turista de Manhattan



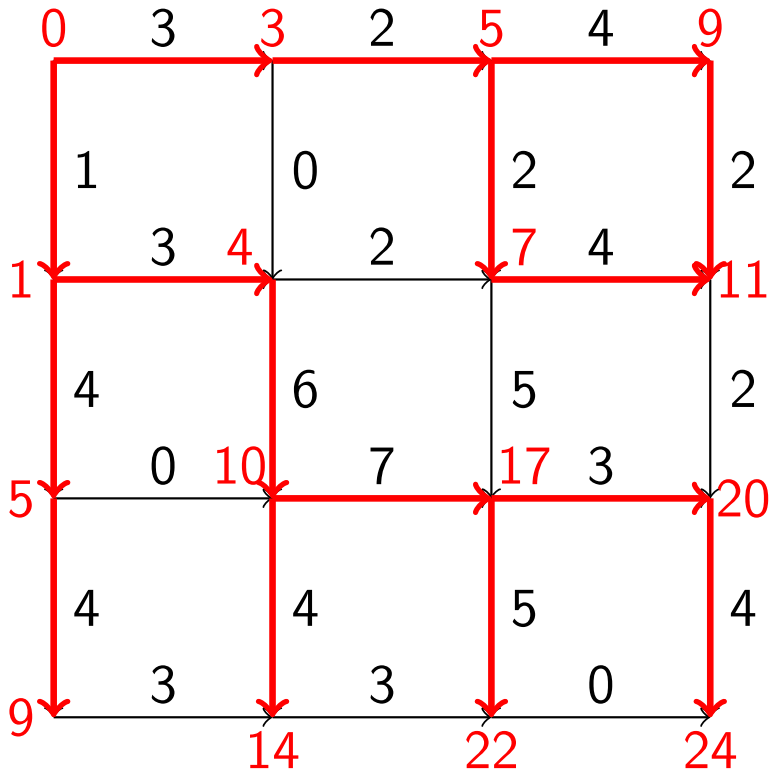
El problema del turista de Manhattan



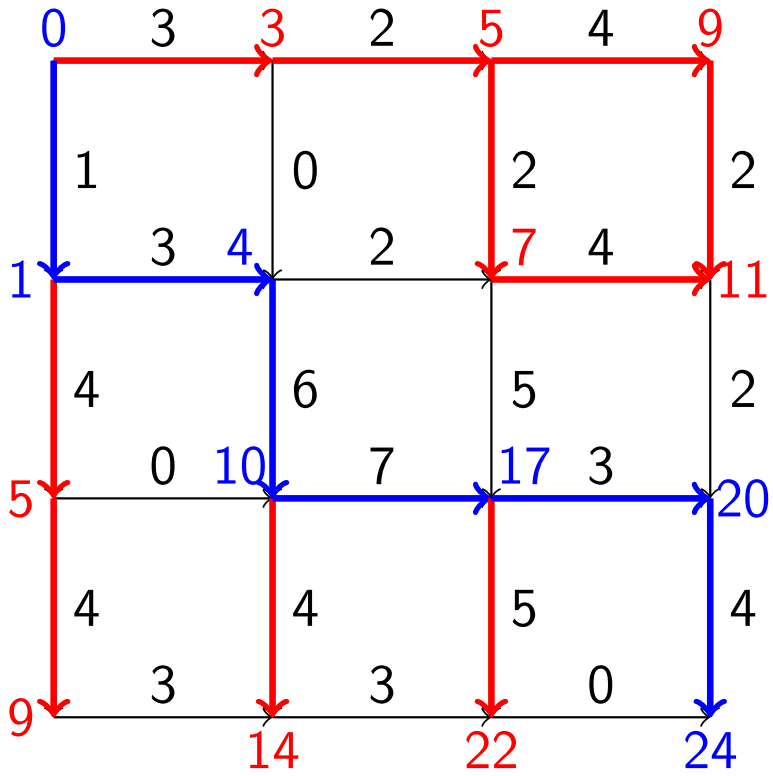
El problema del turista de Manhattan



El problema del turista de Manhattan



El problema del turista de Manhattan



El problema del turista de Manhattan

MahtattanPD(G,n,m)
begin
$s_{0,0} := 0$
for $i = 1$ **to** n **do**
  $s_{i,0} := s_{i-1,0} + w((i-1,0),(i,0))$
  $p_{i,0} := (i-1,0)$
for $j = 1$ **to** m **do**
  $s_{0,j} := s_{0,j-1} + w((0,j-1),(0,j))$
  $p_{0,j} := (0,j-1)$
for $i = 1$ **to** n **do**
  for $j = 1$ **to** m **do**

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + w((i-1,j),(i,j)) \\ s_{i,j-1} + w((i,j-1),(i,j)) \end{cases}$$

  if $s_{i,j} = s_{i-1,j} + w((i-1,j),(i,j))$ **then**
    $p_{i,j} = (i-1,j)$
  else
    $p_{i,j} = (i,j-1)$

El problema del turista de Manhattan

Ara cada $s_{i,j}$ conté el pes d'un camí de pes màxim de $(0,0)$ a (i,j) i $p_{i,j}$ una possible arribada d'un camí de pes màxim a (i,j). Per acabar...

output $s_{n,m}$ $(x,y) := (m,m)$
camíòptim $= ((n,m))$
while $(x,y) \neq (0,0)$ **do**
  camíòptim $:=$ $(p_{x,y}$, camíòptim$)$
  $(x,y) := p_{x,y}$
output camíòptim

Cost total: temps $O(n \cdot m)$, espai $O(n \cdot m)$.

Exercici: Reformar-ho per trobar tots els camins òptims

El problema del turista de Manhattan

Si en lloc d'una graella estàssim en un graf dirigit acíclic arrelat (rDAG) $G = (V, E)$, empraríem la recurrència

$$s(v) = \max\{s(u) + w(u, v) \mid (u, v) \in E\}$$

però cal decidir l'ordre amb què visitam els nodes: per calcular $s(v)$, el s de tots els seus pares ha d'haver estat calculat abans.

El problema del turista de Manhattan

Ordenació topològica d'un DAG $G = (V, E)$: ordenació dels nodes de manera que si $(u, v) \in E$, aleshores $u < v$.

Els nodes d'un DAG es poden ordenar topològicament en temps $O(|E| + |V|)$

El problema del turista de Manhattan

```
Topologicalordering(G)
begin
L := () %Llista dels nodes ordenats topològicament
Q := {arrels de G}
while Q ≠ ∅ do
   triam v ∈ Q
   Q := Q − {v}
   L := (L, v)
   for each u ∈ V such that (v, u) ∈ E
      E := E − {(v, u)}
      if u es queda sense arcs d'entrada then
         Q := Q ∪ {u}
if E ≠ ∅ then
   output error (G conté cicles)
else
   output L
end
```

El problema del turista de Manhattan

ManhattanDAGPD(G,v)
begin
Ordenam topològicament G
$n :=$ lloc de v en l'ordre topològic de G
$s_0 := 0$
for $i = 1$ **to** n **do**
  $s_i := \max\{s_j + w(v_j, v_i) \mid (v_j, v_i) \in E\}$
  for $j = 1$ **to** i **do**
    if $s_i = s_j + w(v_j, v_i)$ **then**
      $p_i := j$
output s_n
$x = n$
camíòptim $= (n)$
while $x \neq 0$ **do**
  camíòptim $:=$ $(p_x,$ camíòptim$)$
  $x := p_x$
output camíòptim

LCS

Donades dues paraules $a = a_1 \dots a_n$ i $b = b_1 \dots b_m$ sobre un alfabet fixat, una subseqüència en comú més llarga (Longest Common Subsequence, LCS) és una parella de vectors de posicions

$$\begin{aligned} \underline{a} &= (i_1, \dots, i_l) \quad 1 \leqslant i_1 < i_2 < \dots < i_l \leqslant n \\ \underline{b} &= (j_1, \dots, j_l) \quad 1 \leqslant j_1 < j_2 < \dots < j_l \leqslant m \end{aligned}$$

tal que $a_{i_k} = b_{j_k}$ per a tot $k = 1, \dots, l$, i k és màxim amb aquesta propietat.

LCS

Exemple: Considerem $a = \texttt{actggac}$ i $b = \texttt{atgctc}$

LCS

Exemple: Considerem $a = \texttt{actggac}$ i $b = \texttt{atgctc}$

```
a-ctggac--
at--g--ctc
```

és CS maximal, però no una de L

LCS

Exemple: Considerem $a = \texttt{actggac}$ i $b = \texttt{atgctc}$

```
a-ctggac--
at--g--ctc
```

és CS maximal, però no una de L

```
actggac--
a-tg--ctc
```

és LCS

LCS

Exemple: Considerem $a = \texttt{actggac}$ i $b = \texttt{atgctc}$

```
a-ctggac--
at--g--ctc
```

és CS maximal, però no una de L

```
actggac--
a-tg--ctc
```

és LCS

```
actgga--c
a-t-g-ctc
```

és una altra LCS

LCS

1. Reconèixer el problema de trobar una (o totes les) LCS com un problema de trobar camí òptim en un rDAG
2. Donar un algoritme de PD per trobar tots els LCS de dues paraules
3. Implementar-lo en Perl (amb alfabet a,c,g,t): l'output ha de ser 'user friendly'