UNIVERSITAT DE BARCELONA

Programació Dinàmica

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2019-2020

Programació dinàmica

- La programació dinàmica ens permet resoldre una gran quantitat de problemes (més dels que hem vist fins ara)
- Aplicat a problemes d'optimització
- Això, com veurem, té un cost en la complexitat
- El terme **programació** fa referencia a **planificació**

Teorema de Optimalidad de Mitten

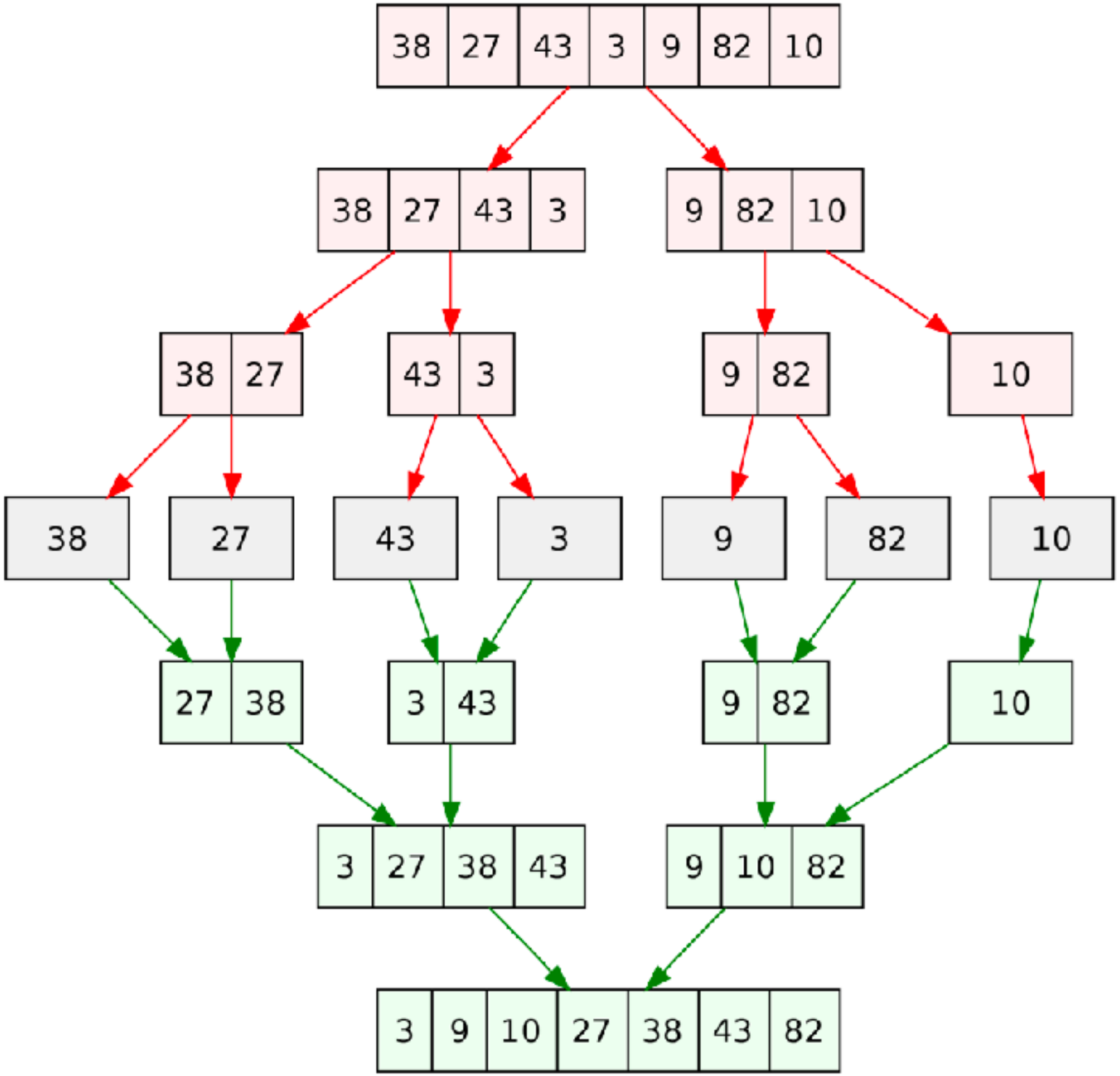
El objetivo básico en la programación dinámica consiste en 'descomponer" un problema de optimización en k variables a una serie de problemas con menor número de variables más fáciles de resolver.

Greddy vs. Programació dinàmica

- Greedy
    - Una única seqüència de la solució és generada
- Programació dinàmica
    - Múltiples seqüències són generades

Divide & Conquer vs. Programació dinàmica

- Divide and Conquer
    - Dividir el problema amb subproblemes, solucionar el problemes recursivament, i finalment combinar les seves solucions per tal de solucionar el problema original
- Programació dinàmica
    - Aplicable quan els subproblemes **no són independents**, és a dir, quan els subproblemes comparteixen subproblemes.
    - Soluciona cada subprolema un únic cop i emmagatzema la solució dins una taula, evitant així tornar-ho a calcular si un altre cop és necessita resoldre el subproblema

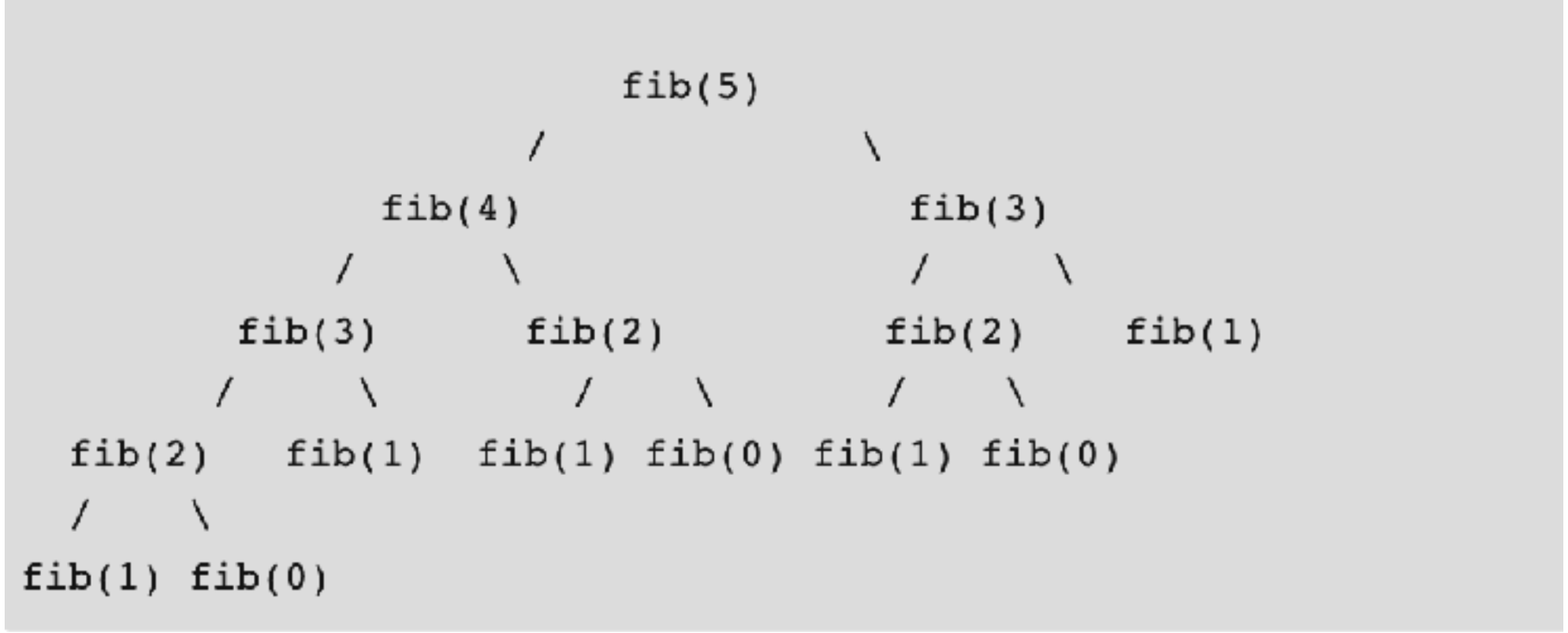

38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Examples

- Quins dels següents problemes es poden resoldre amb algoritmes greedy? i amb programació dinàmica?
  - Binary search
  - Quick Sort
  - Trobar número de fibonacci

Fibonacci

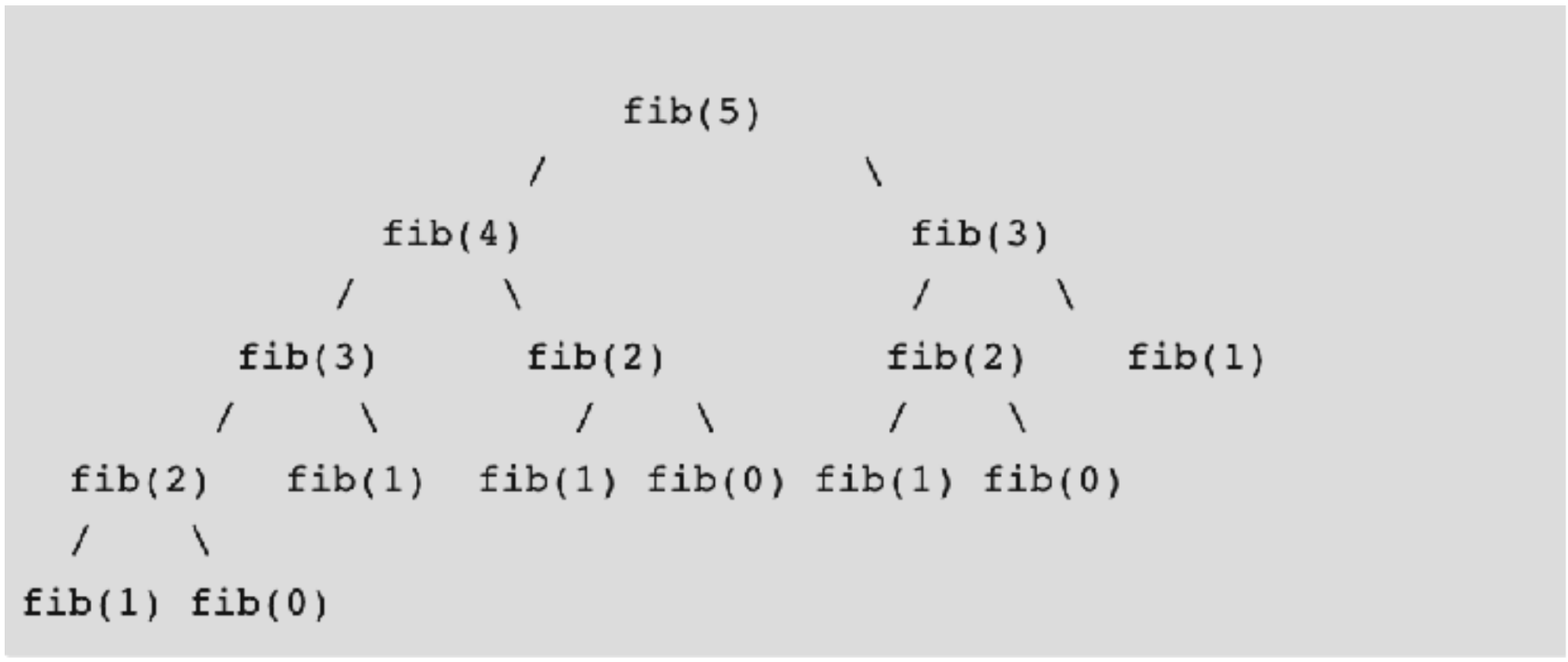
```c
/* simple recursive program for Fibonacci numbers */
int fib(int n)
{
   if ( n <= 1 )
      return n;
   return fib(n-1) + fib(n-2);
}
```



Programació dinàmica

- Passos
  - Trobar sub-estructura òptima
  - Definir recursivitat
  - Afegir casos base

- Definir recursivitat
  - - > Memoritzar



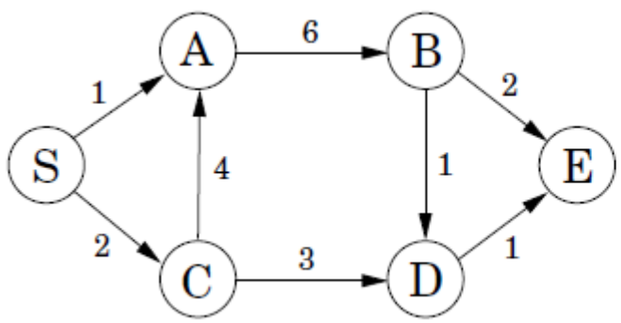
```
memo = {}
def fib(n):
    # Check if we've already solve the subproblem
    if n in memo: return memo[n]
    # Normal recursive algorithm
    if n <= 2: f = 1
    else: f = fib(n-1) + fib(n-2)
    # Remember our solution for later
    memo[n] = f
    return f
```

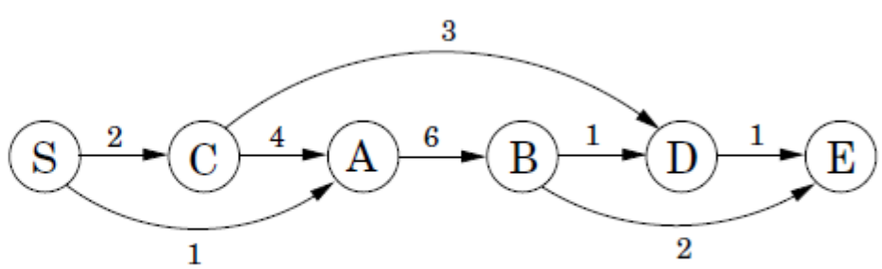
La solución óptima obtenida se ajusta al **principio de optimalidad** establecido por R. Bellman en 1957.

Una política óptima tiene la propiedad de que, cualquiera que sea el estado inicial y las decisiones iniciales, las restantes decisiones deben constituir una política óptima con respecto al estado resultante de la primera decisión

Programació dinàmica

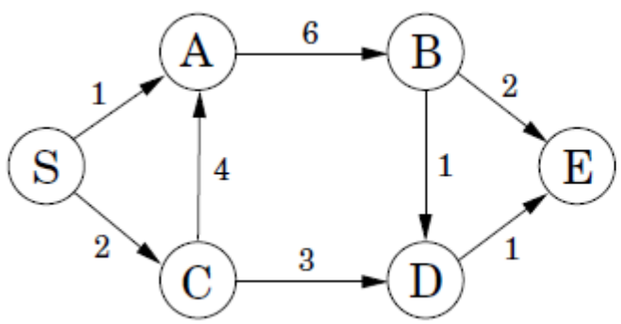
- Relació amb la linearització de grafs

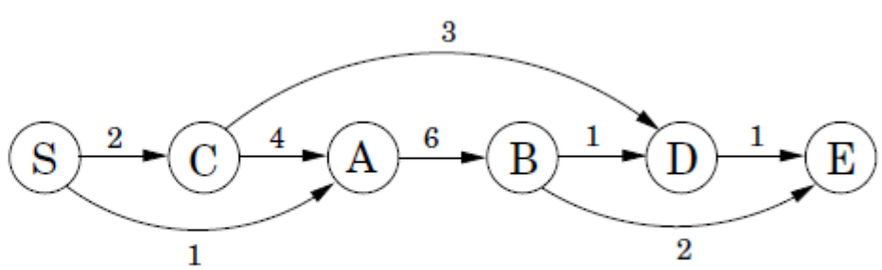




Programació dinàmica

- Relació amb la linearització de grafs





- Distància més curta a D:

$$\texttt{dist}(D) = \min\{\texttt{dist}(B) + 1, \texttt{dist}(C) + 3\}$$

Programació dinàmica

- Podem calcular la distància a tots els nodes en un pas:

```
initialize all dist(·) values to ∞
dist(s) = 0
for each v ∈ V\{s}, in linearized order:
    dist(v) = min_{(u,v)∈E}{dist(u) + l(u, v)}
```

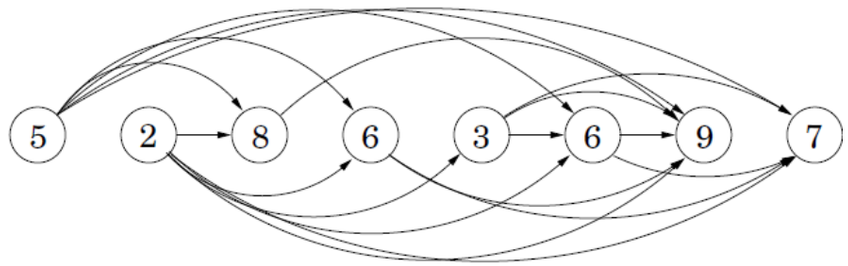
- Solucionem un conjunt de **subproblemes** fins que arribem a una solució final: tècnica molt general!!

- Tenim una funció sobre els nodes anteriors per actualitzar una resposta al node actual:

  - → Aquesta funció podria ser qualsevol! (i.e. Màxim en lloc de mínim)

Programació dinàmica

- Per obtenir una representació de programació dinàmica, suposem que el graf disposa de totes les connexions entre un node i els seus predecessors:



- Exemple: trobar el camí de longitud màxima

```
for j = 1, 2, ..., n:
    L(j) = 1 + max{L(i) : (i, j) ∈ E}
return max_j L(j)
```

Programació dinàmica

- Aplicació: **distància d'edició**

	SNOWY	SUNNY
	S – N O W Y	– S N O W – Y
	S U N N – Y	S U N – – N Y
	Cost: 3	Cost: 5

- Possibles accions: inserció, eliminació i substitució
  - Hi ha moltes combinacions!!!
- Donem una solució amb programació dinàmica

Programació dinàmica

- Taula de subproblemes

```
for i = 0, 1, 2, ..., m:
    E(i, 0) = i
for j = 1, 2, ..., n:
    E(0, j) = j
for i = 1, 2, ..., m:
    for j = 1, 2, ..., n:
        E(i, j) = min{E(i − 1, j) + 1, E(i, j − 1) + 1, E(i − 1, j − 1) + diff(i, j)}
return E(m, n)
```

Depen de l'objectiu del problemes

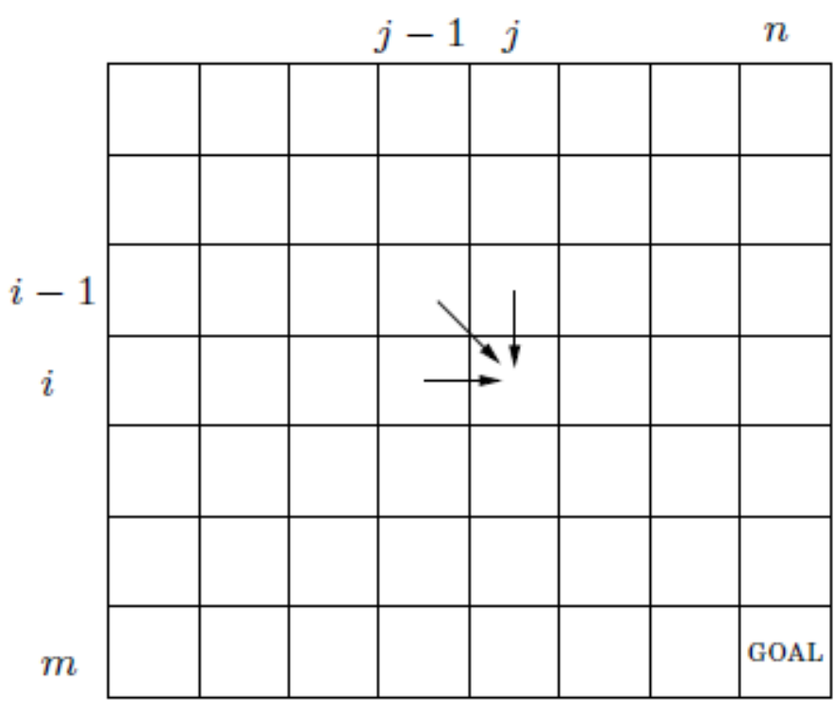
E	X	P	O	N	E	N	−	T	I	A	L
−	−	P	O	L	Y	N	O	M	I	A	L

$O(mn)$

Programació dinàmica

- Taula de subproblemes

$$E(i,j) = \min\{E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+\texttt{diff}(i,j)\}$$

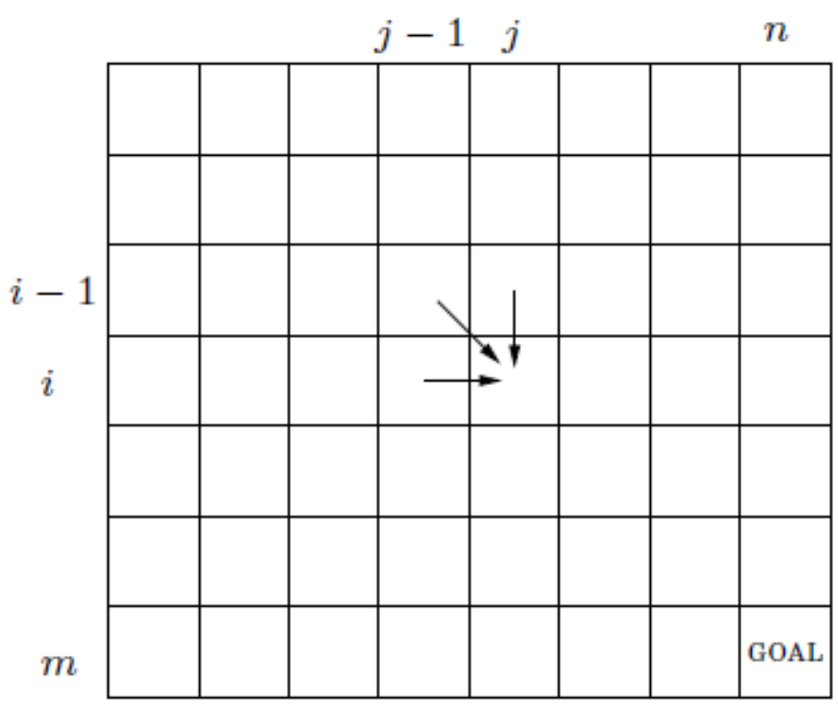


		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1										
X	2										
P	3										
O	4										
N	5										
E	6										
N	7										
T	8										
I	9										
A	10										
L	11										

Programació dinàmica

- Taula de subproblemes

$$E(i,j) = \min\{E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+\texttt{diff}(i,j)\}$$

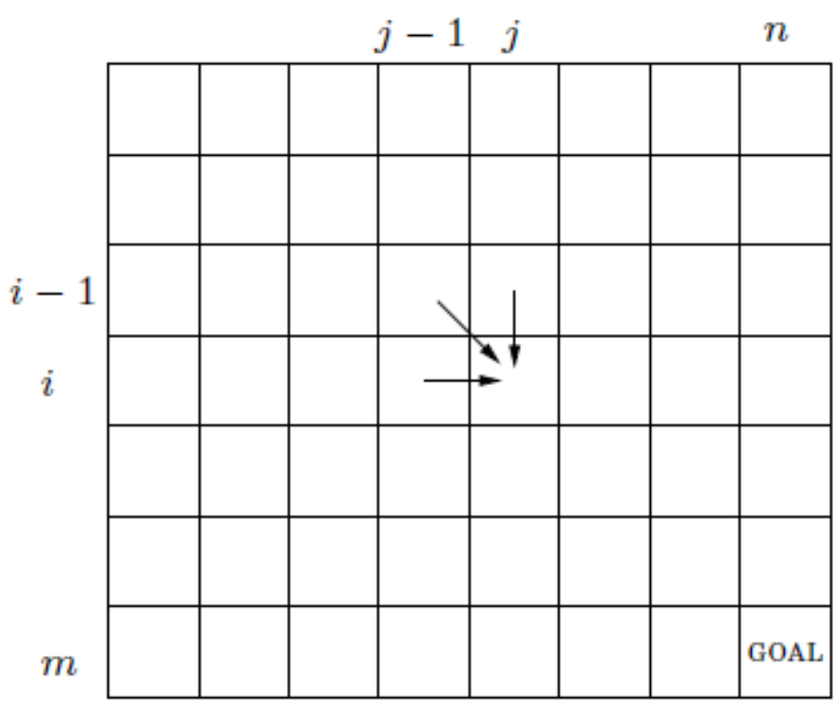


		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1									
X	2	2									
P	3	2									
O	4	3									
N	5	4									
E	6	5									
N	7	6									
T	8	7									
I	9	8									
A	10	9									
L	11	10									

Programació dinàmica

- Taula de subproblemes

$$E(i,j) = \min\{E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+\texttt{diff}(i,j)\}$$

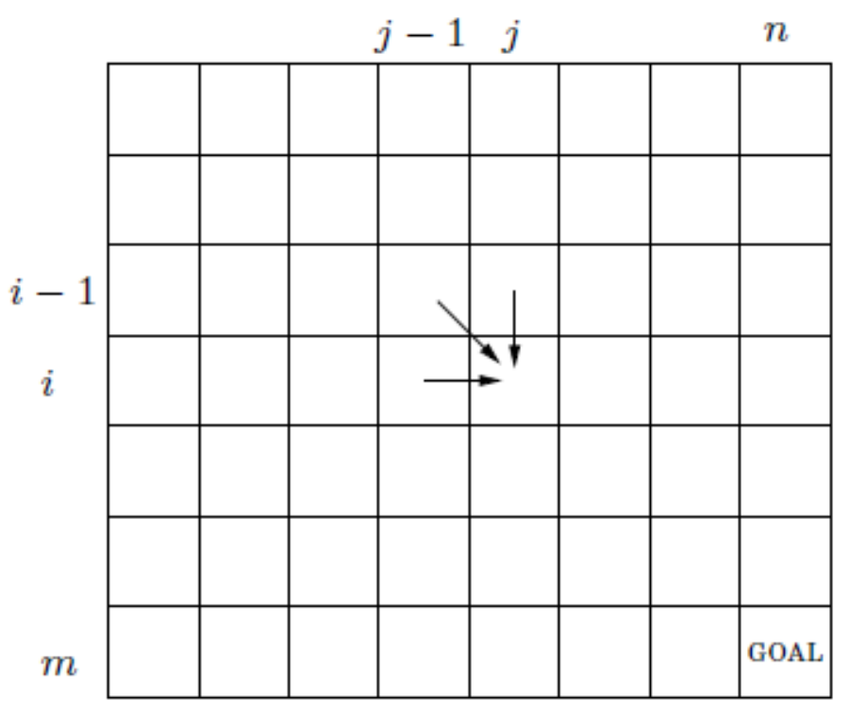


		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1									
X	2	2									
P	3	2									
O	4	3									
N	5	4									
E	6	5									
N	7	6									
T	8	7									
I	9	8									
A	10	9									
L	11	10									

Programació dinàmica

- Taula de subproblemes

$$E(i,j) = \min\{E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+\texttt{diff}(i,j)\}$$

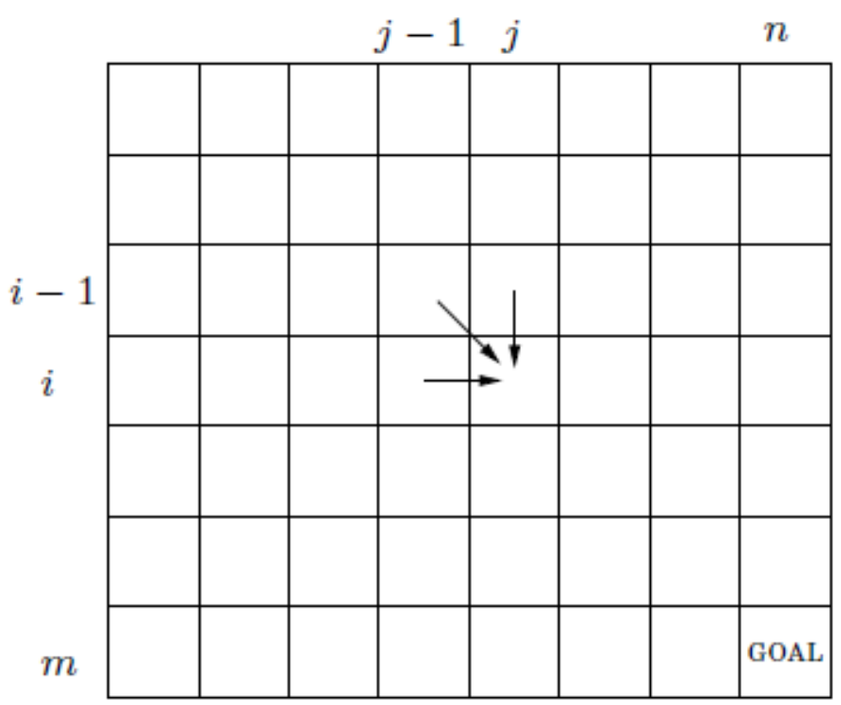


		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

Programació dinàmica

- Taula de subproblemes

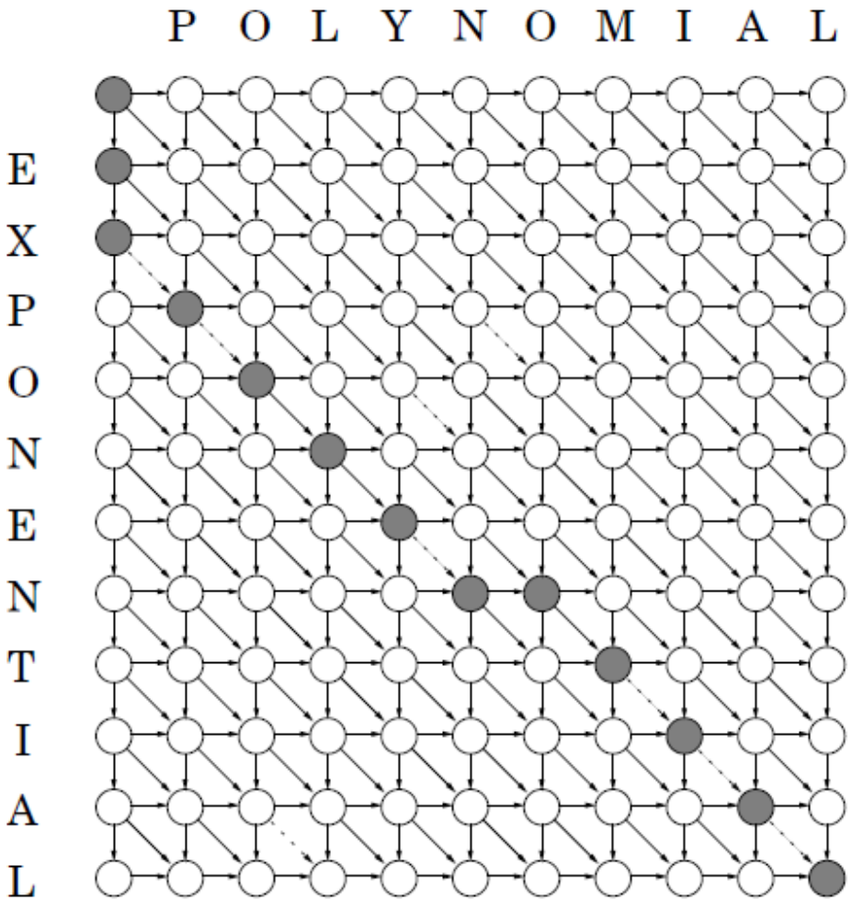
$$E(i,j) = \min\{E(i-1,j)+1, E(i,j-1)+1, E(i-1,j-1)+\texttt{diff}(i,j)\}$$



		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

Programació dinàmica

- Taula de subproblemes




```
E X P O N E N – T I A L
– – P O L Y N O M I A L
```

Hi han diferents camins amb el mateix cost associat

Els camins varien segons fixem costos diferents a diferents operacions de inserció, eliminació i substitució

Exercici: Programació dinàmica

- Associar la paraula ALGORISMICA amb la paraula AVANÇADA fent ús d'aquesta inicialització i funció de programació dinàmica

		A	L	G	O	R	I	S	M	I	C	A
	0	1										
A	1	0										
V												
A												
N												
Ç												
A												
D												
A												

Exercici: Programació dinàmica

- Associar la paraula ALGORISMICA amb la paraula AVANÇADA fent ús d'aquesta inicialització i funció de programació dinàmica

		A	L	G	O	R	I	S	M	I	C	A
	0	1	2	3	4	5	6	7	8	9	10	11
A	1	0	1									
V	2	1	1									
A	3	2	2									
N	4	3	3									
Ç	5	4	4									
A	6	5	5									
D	7	6	6									
A	8	7	7									

Exercici: Programació dinàmica

- Com modificaríeu el codi tal que el cost de d'eliminar una lletra tingui un cost de 2 enlloc de 1.

- Quin seria el resultat obtingut de transformar la paraula **Avançada** a **Algorísmica.**

		A	L	G	O	R	I	S	M	I	C	A
	0	1	2	3	4	5	6	7	8	9	10	11
A	2	0	1	2	3	4	5	6	7	8	9	10
V	4	2	1	2	3	4	5	6	7	8	9	10
A	6	4	3	2	3	4	5	6	7	8	9	9
N	8	6	5	4	3	4	5	6	7	8	9	10
Ç	10	8	7	6	5	4	5	6	7	8	9	10
A	12	10	9	8	7	6	5	6	7	8	9	9
D	14	12	11	10	9	8	7	6	7	8	9	10
A	16	14	13	12	11	10	9	8	7	8	9	9

Camins més curts

Quins algoritmes coneixem?

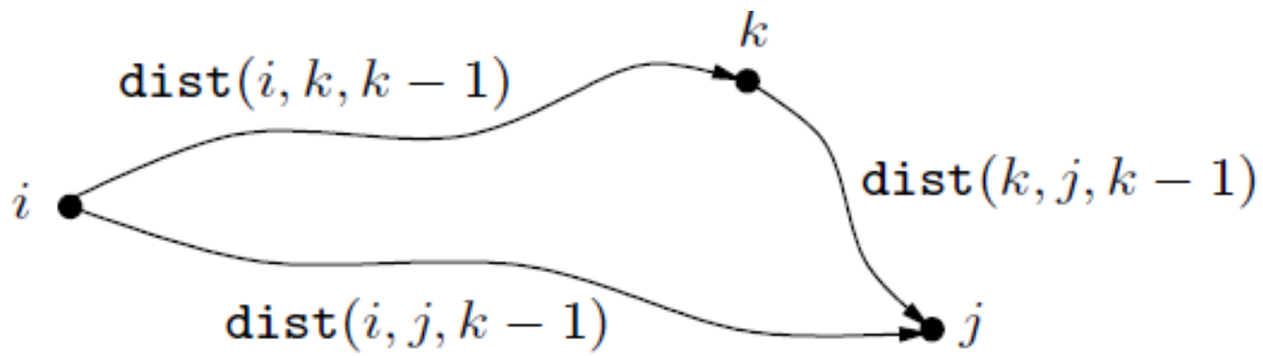
Dijkstra
Bellman Ford

Programació Dinàmica:**Floyd Warshall**

- Fem és d'una matriu tridimensional, on:

$$dist(i, j, k)\{1,2,\ldots, k\}$$

- és la distància del camí més curt entre i i j tenint en compte només k nodes.



$$\texttt{dist}(i,k,k-1) + \texttt{dist}(k,j,k-1) \;<\; \texttt{dist}(i,j,k-1),$$

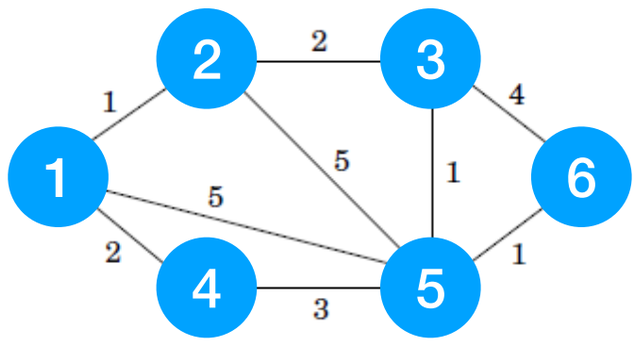
Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

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

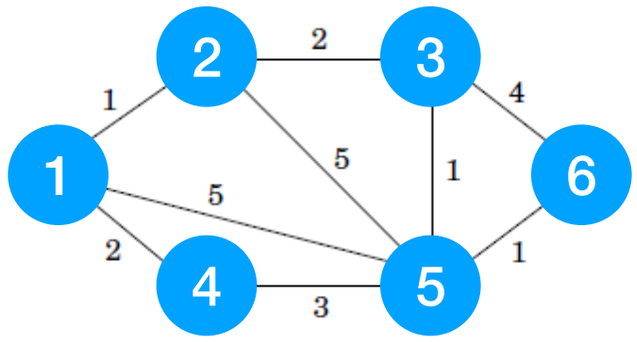


Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

k = 1

	1	2	3	4	5	6
1	-	1	-	2	5	-
2	1	2	2	3	5	-
3	-	2	-	-	1	4
4	2	3	-	4	3	-
5	5	5	1	3	10	1
6	-	-	4	-	1	-

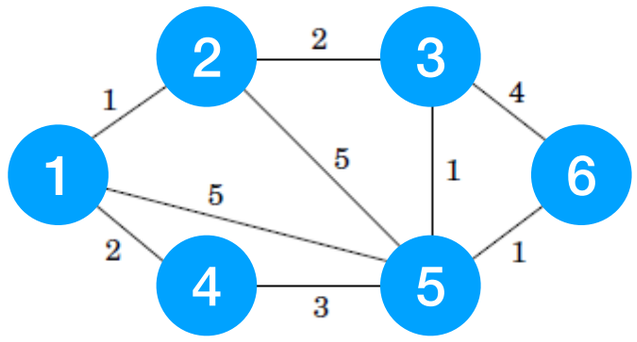


Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

k = 2

	1	2	3	4	5	6
1	2	1	3	2	5	-
2	1	2	2	3	5	-
3	3	2	4	5	1	4
4	2	3	5	4	3	-
5	5	5	1	3	10	1
6	-	-	4	-	1	-

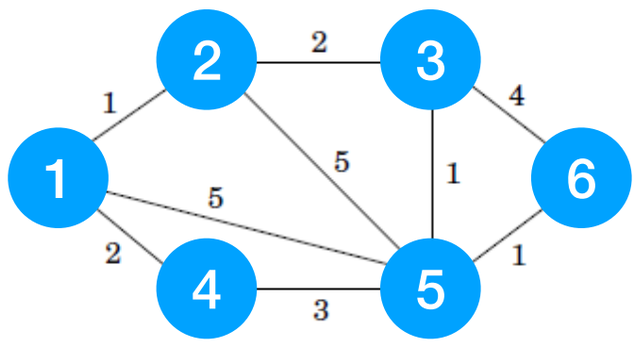


Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

k = 3

	1	2	3	4	5	6
1	2	1	3	2	4	7
2	1	2	2	3	3	6
3	3	2	4	5	1	4
4	2	3	5	4	3	9
5	4	3	1	3	2	1
6	7	6	4	9	1	8

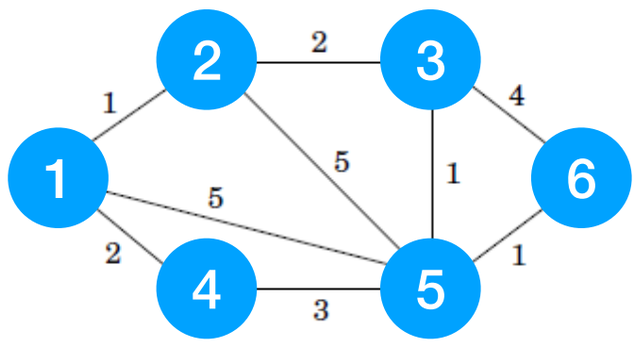


Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

k = 4

	1	2	3	4	5	6
1	2	1	3	2	4	7
2	1	2	2	3	3	6
3	3	2	4	5	1	4
4	2	3	5	4	3	9
5	4	3	1	3	2	1
6	7	6	4	9	1	8

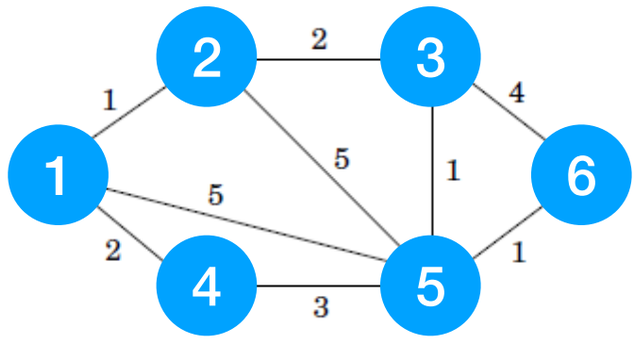


Programació Dinàmica:**Floyd Warshall**

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞
for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

k = 5

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



Programació Dinàmica:**Floyd Warshall**

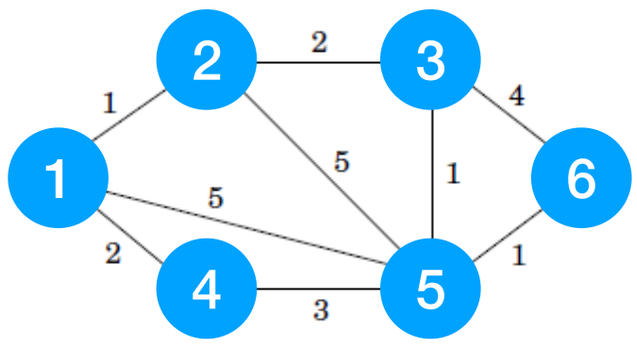
```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = ∞

for all (i, j) ∈ E:
    dist(i, j, 0) = ℓ(i, j)

for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1),  dist(i, j, k − 1)}
```

k = 6

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



Programació Dinàmica:**Floyd Warshall**

- Complexitat?

Programació Dinàmica:**Floyd Warshall**

- Complexitat : $O(|V|^3)$

Comparision

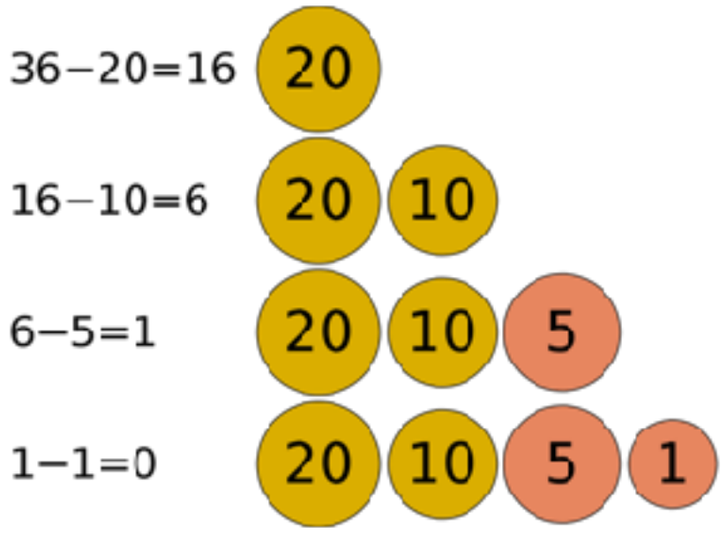
- The **Floyd–Warshall** algorithm is a good choice for computing paths between all pairs of vertices in dense graphs, in which most or all pairs of vertices are connected by edges.

- For **sparse graphs** with non-negative edge weights, a better choice is to use **Dijkstra's algorithm** from each possible starting vertex, since the running time of repeated Dijkstra (using Fibonacci heaps) is better than the running time of the Floyd–Warshall algorithm when is significantly smaller than.

Problema canvi monedes

- **Problema**: Donat un conjunt n tipus de monedes, cada una d'elles amb un valor c_i, i una quantitat P, trobar el número mínim de monedes a utilitzar per obtenir la quantitat exacte.

- Alguna proposta amb un algorítme greedy? Solució molt eficient, però no sempre la òptima



Problema canvi monedes

- Definició de l'equació recurrent:
  - $Canvi(i, Q)$: calcula el número mínim de monedes necessàries per a retornar una quantitat Q, utilitzant els i primer tipus de monedes (és a dir, 1…i),
  - La solució de $Canvi(i, Q)$ pot utilitzar k monedes de tipus **i** o pot ser que no utilitzi cap.
    - Sinó s'utilitza cap moneda d'aquest tipus: $Canvi(i, Q) = Canvi(i - 1, Q)$.
    - Si s'utilitza k monedes de tipus i: $Canvi(i, Q) = Canvi(i, Q - k * c_i) + k$
  - En qualsevol cas, el valor sempre serà el mínim:
    - $Canvi(i, Q) = min_{k=0,1,\ldots,Q/C_i}\{Canvi(i - 1, Q - k * c_i) + k\}$
- Casos base:
  - Si (i<1) o (Q<0) no hi ha cap solució al problema i $Canvi(i, Q) = +\inf$
  - Per a qualsevol i>0, $Canvi(i, 0) = 0$

Problema canvi monedes

- Definició de les taules utilitzades:
  - Necessitem emmagatzemar els resultats de tots els subproblemes.
  - El problema a resoldre es: Canvi (n, P)
  - Necessitem una taula de n*x*P, d'enters, que anomenarem D, on D[i,j]= Canvi(i,j)
- Exemple. n = 3, P = 8, v =(1,4,6)

Quantitat a retornar

	0	1	2	3	4	5	6	7	8
V1 = 1	0	1	2	3	4	5	6	7	8
V2 = 4	0	1	2	3	1	2	3	4	2
V3 = 6	0	1	2	3	1	2	1	2	2

- De dalt a baix, i d'esquerra a dreta, aplicar la següent funció de recurrència:
  - $D[i,j] = min_{k=0,1,...,Q/Ci}\{D[i-1, Q-k*c_i] + k\}$

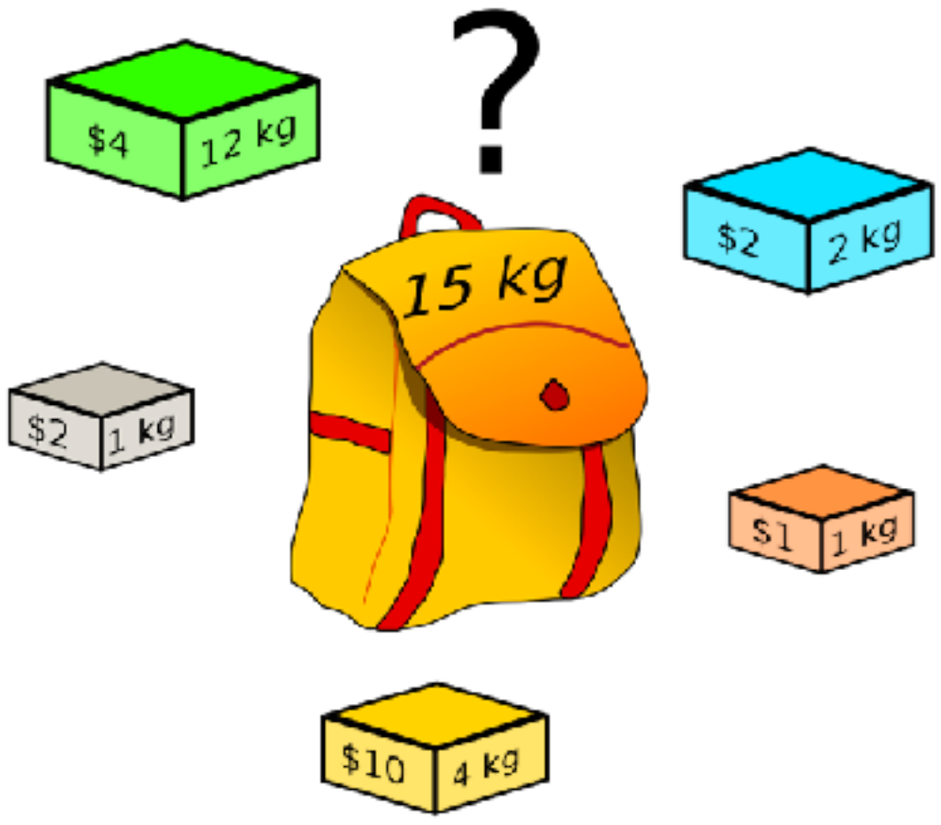
Exercici

- **El viatge més barat pel riu**
  - En un riu hi ha **n** embarcadors, en cadascun d'ells quals es pot llogar un bot per anar a un altre embarcador situat a una zona més baixa del riu. Suposem que no es pot remuntar el riu. Una taula de tarifes indica els costos de viatjar entre els diferents embarcadors. Es suposo que un viatge entre *i* i *j* pot sortir més barat fent diverses escales que no pas directament.
  - El problema consisteix amb determinar el **cost mínim** per a anar d'una embarcador a un altre. Els costos venen definits per la taula de tarifes, T. Així, T [i, j] serà el cost d'anar de l'embarcador i a l'j. La matriu serà triangular superior d'ordre n, on n és el nombre d'embarcadors.
  - La idea recursiva és que el cost es calcula de la següent manera:
    - $C(i, j) = T[i, k] + C(k, j)$
  - La expressió recurrent per a la solució pot ser definida com:

```
            0    si i = j
C(i, j)=
             Min(T(i,k) + C(k,j), T(i,j))    si i < k <= j
```

  - **Troba l'algoritme !!**

Problema de la motxilla



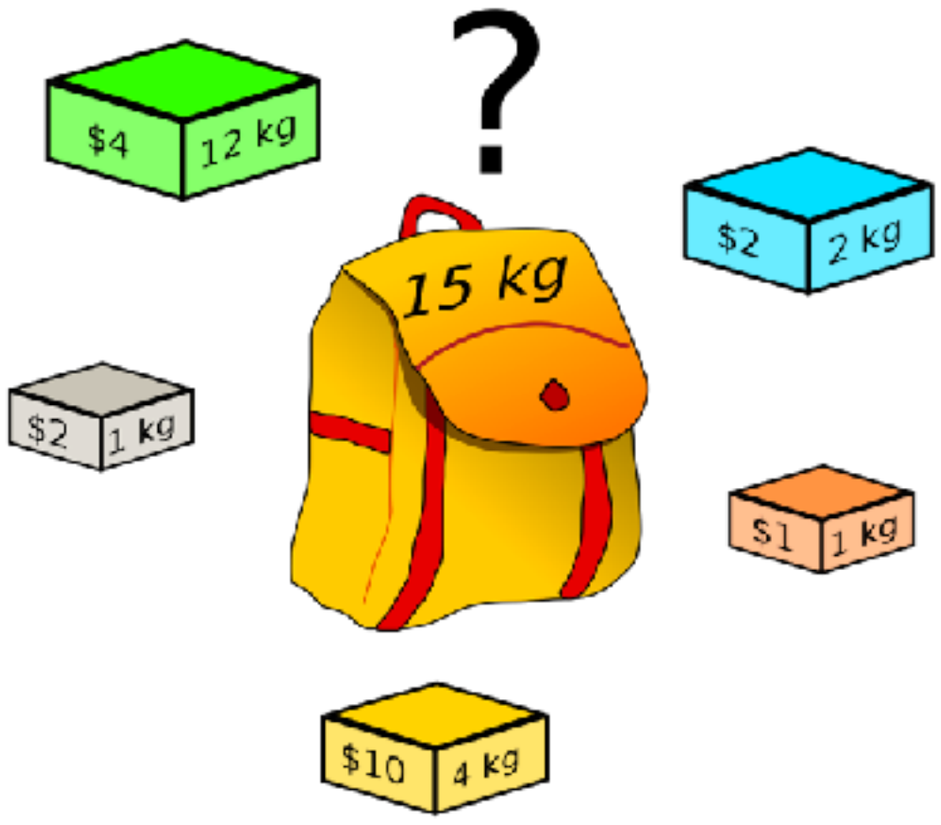
Tenim diversos objectes, amb el seu pes, i tenim la capacitat total de la motxilla. Quins objectes hauríem de posar per tal portar el màxim valor ***v*** possible amb el mínim pes ***w***?

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1\}.$$

Given a set of items numbered from 1 up to , each with a weight *w* and a *v* value , along with a maximum weight capacity *W*.

Problema de la motxilla



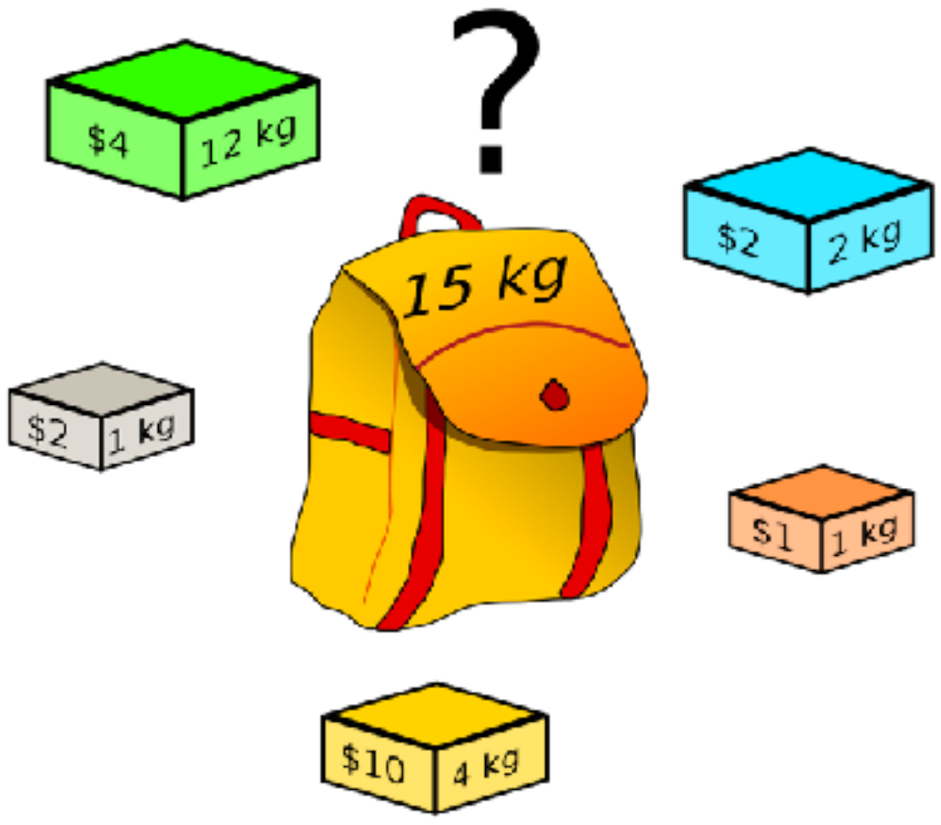
Tenim diversos objectes, amb el seu pes, i tenim la capacitat total de la motxilla. Quins objectes hauríem de posar per tal portar el màxim valor possible amb el mínim pes?

Pensem una solució amb un mètode:

a) Greedy

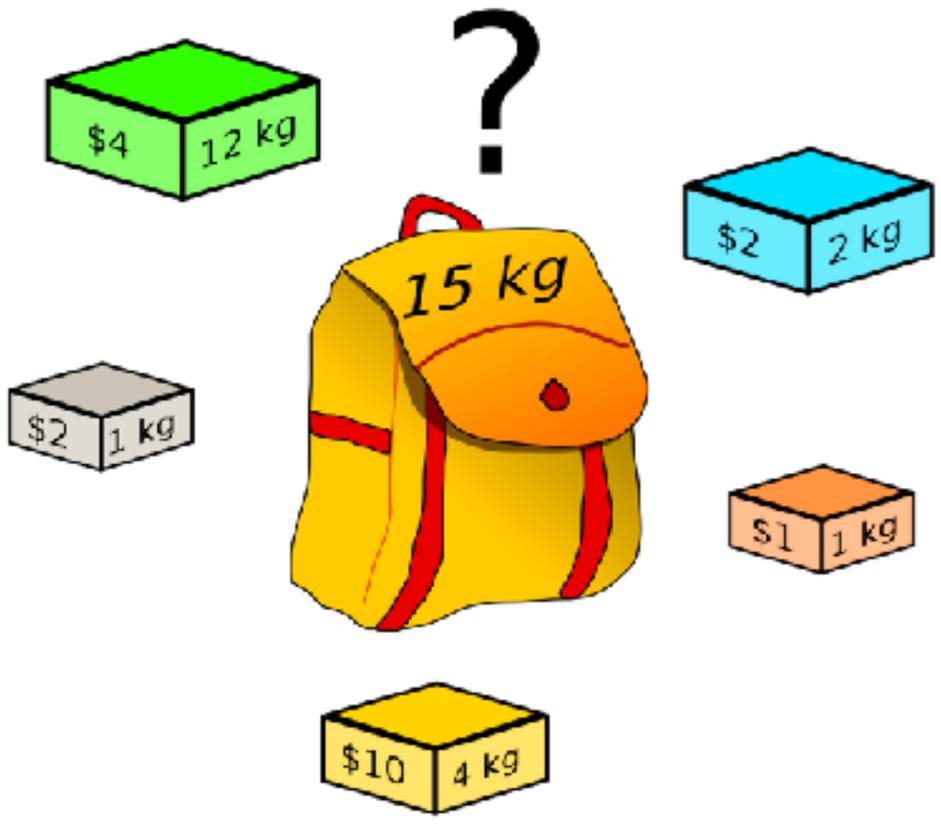
b) Programació dinàmica

Problema de la motxilla



Solució (1) Greedy: Agafa els objectes amb un major valor fins que quedi capacitat dins la motxilla

Problema de la motxilla



Solució (1) Greedy: Agafa els objectes amb un major valor fins que quedi capacitat dins la motxilla

Solució (2) Greedy: Agafa els objectes on els seu valor/pes sigui major fins que quedi capacitat