



UNIVERSITAT DE
BARCELONA



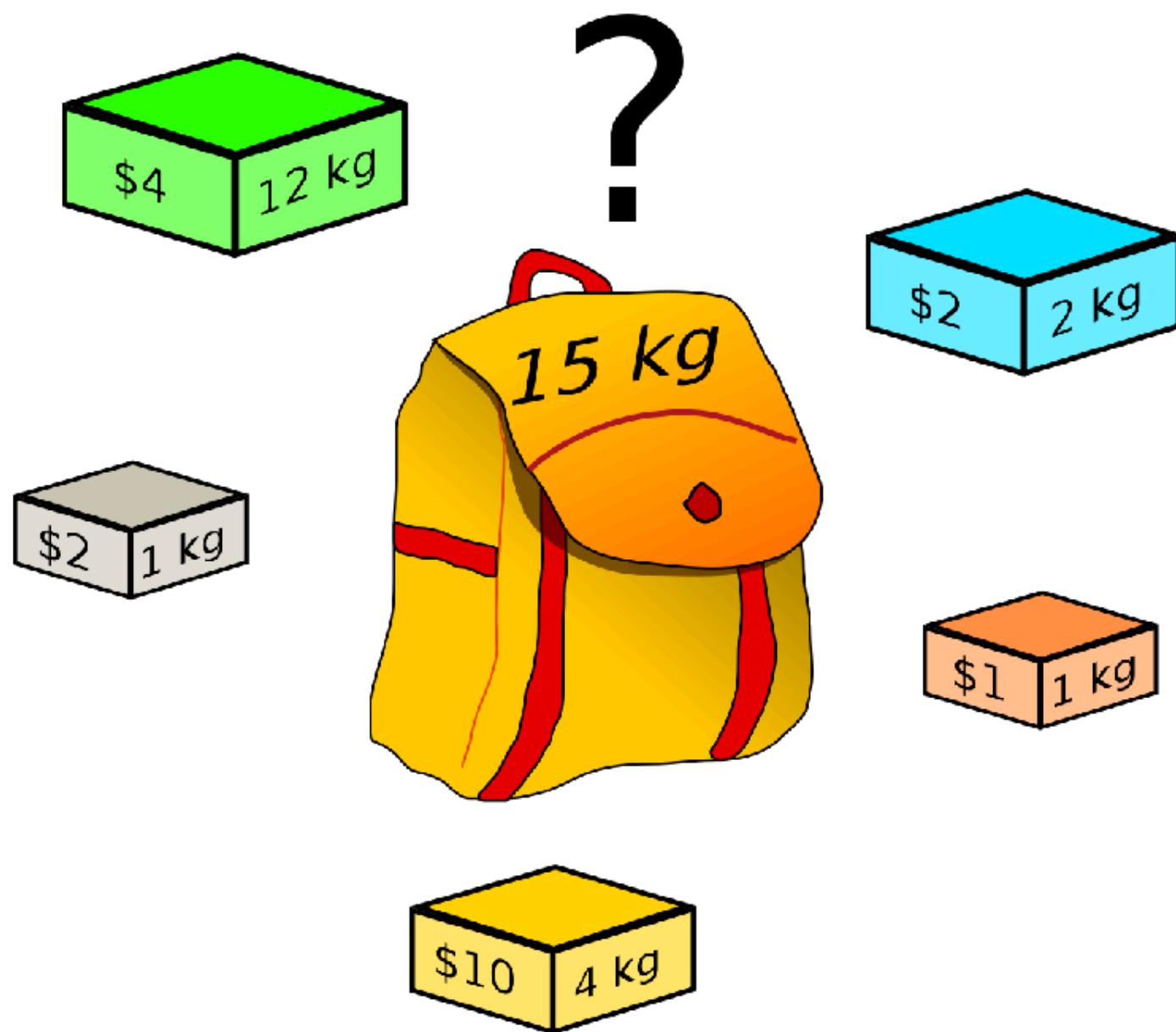
Programació Dinàmica

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí I 2024-2025

Programació Dinàmica

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 ?

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

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

Donat un conjunt d'elements numerats de l'1 al n , cadascun amb un valor de pes w i un valor de v , juntament amb una capacitat de pes màxima W .

Programació Dinàmica

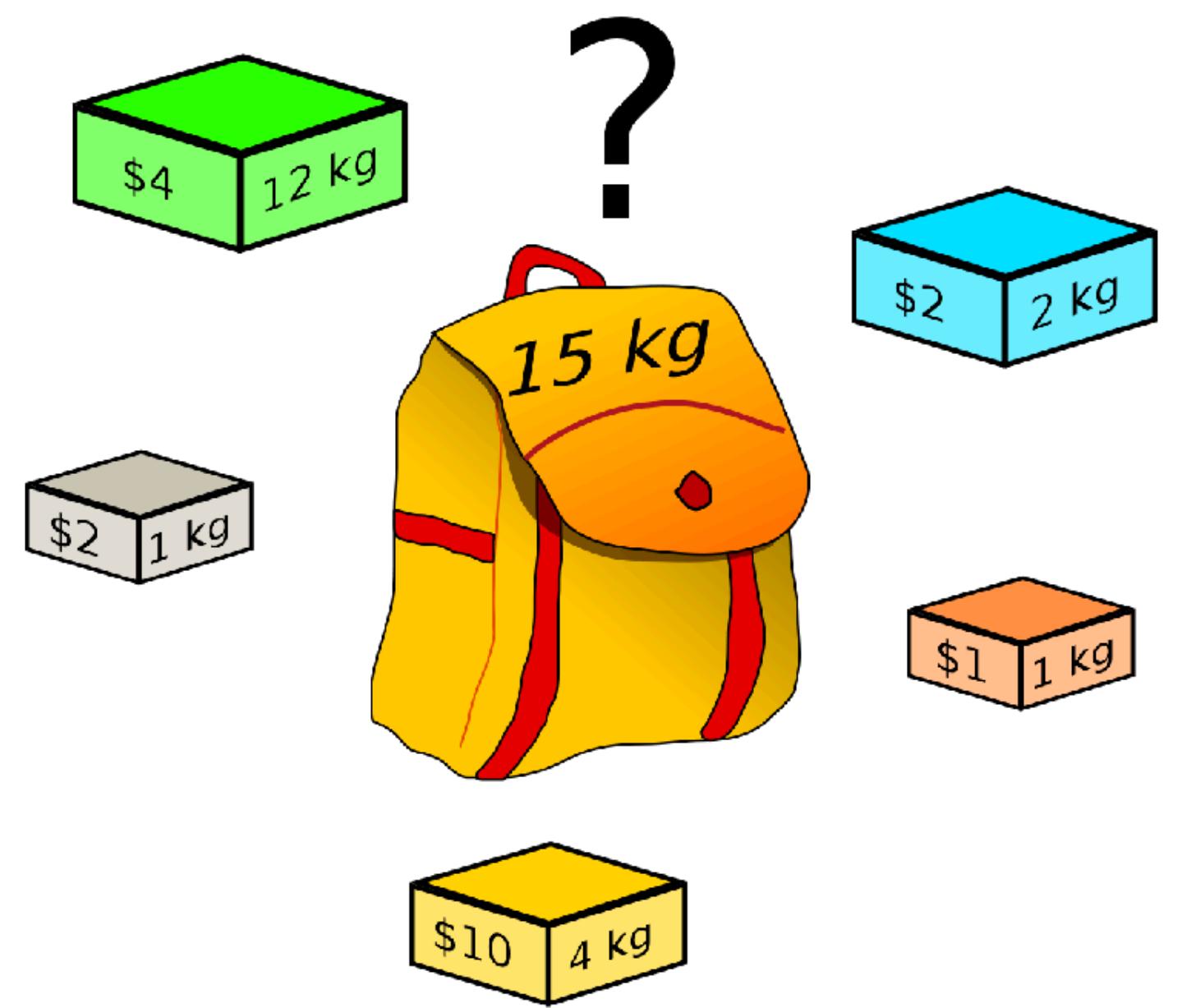
Problema de la motxilla - Solució amb Força bruta

- Donat que hi ha n elements, pot haver-hi 2^n possibles combinacions.
- Provar totes les combinacions ens assegura trobar la solució òptima
- La complexitat serà $O(2^n)$. Problema intractable quan n és gran

Solució vista a Algorísmica

Programació Dinàmica

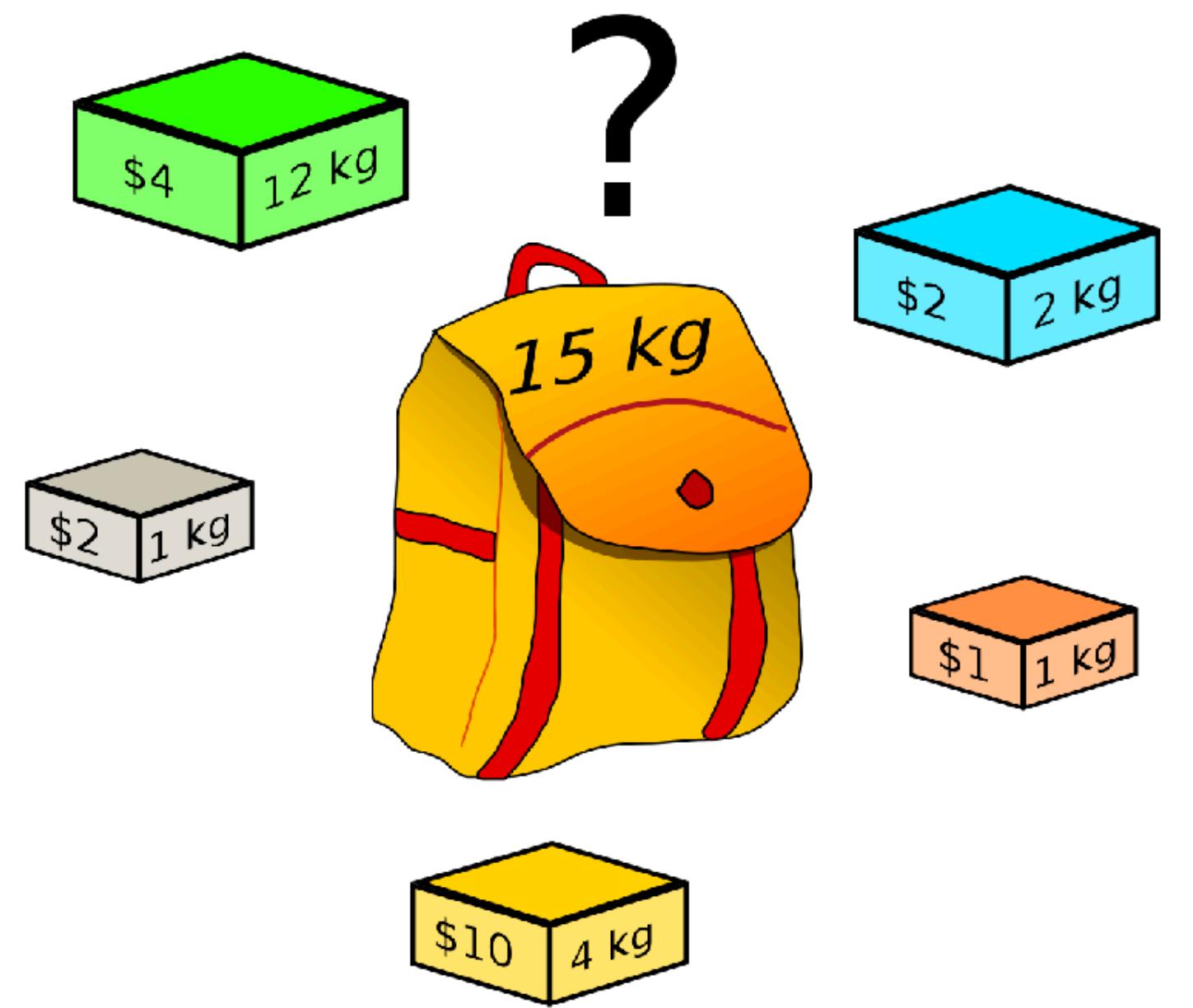
Problema de la motxilla



Greedy?

Solució ràpida però no assegura trobar la solució òptima

Problema de la motxilla



Solució amb DP?

Programació Dinàmica

Problema de la motxilla

- Com resoldre el problema amb programació dinàmica?
 1. *Definir funció de recursitat*
 2. *Trobar sub-estructura òptima*
 3. *Afegir casos base*

Programació Dinàmica

Problema de la motxilla

- Definició del sub-problema / funció de recursivitat (versió 1):
 - Si els articles s'etiqueten $1, \dots, n$, el sub-problema per trobar una solució òptima per a $S_k = \{item\ etiquetats\ 1, 2, \dots, k\}$
 - La qüestió és, podem definir la **solució final** (S_n) en termes dels **sub-problemes** (S_{n-1})?

Programació Dinàmica

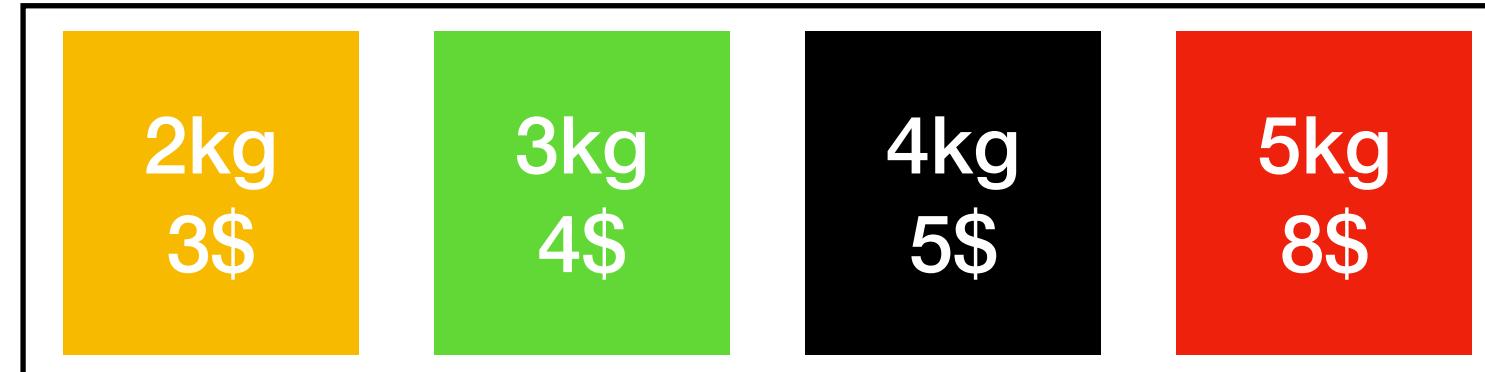
Problema de la motxilla

Objectiu: Màxim benefici donat un
Pes màxim = 20 Kg



Definim el subproblema:

S_4 :



Pes Total = 14;
Benefici màxim: 20\$

Definim el subproblema:

S_5 :



Pes Total = 20;
Benefici màxim: 26\$

Programació Dinàmica

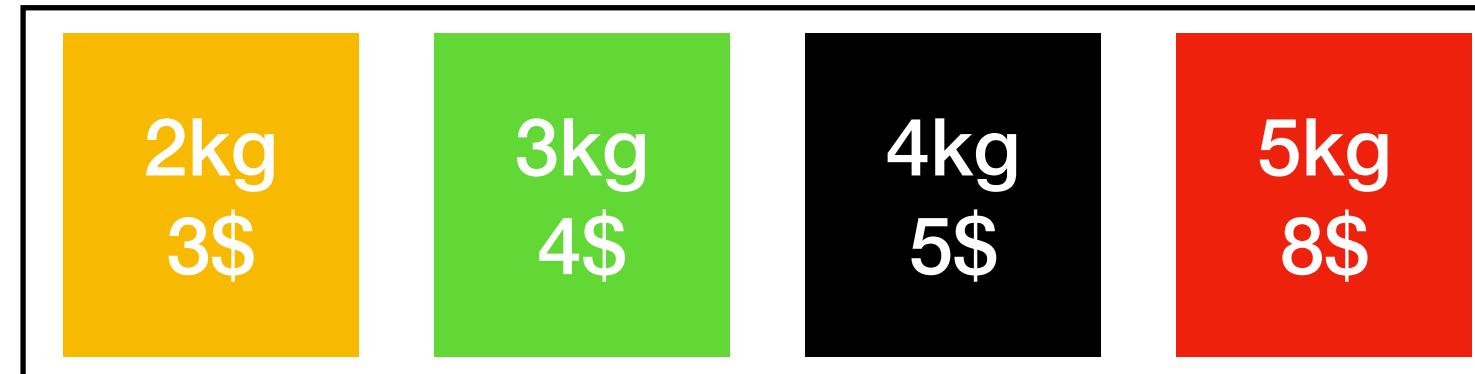
Problema de la motxilla

Objectiu: Màxim benefici donat un
Pes màxim = 20 Kg



Definim el subproblema:

S_4 :



Pes Total = 14;
Benefici màxim: 20\$

Definim el subproblema:

S_5 :



Pes Total = 20;
Benefici màxim: 26\$

la Solució S_4 no és part de la solució S_5 . Per tant el subproblema no està ben definit

Programació Dinàmica

- Definició del sub-problema:
 - calcular $V[\mathbf{k}, \mathbf{w}]$, per tal de trobar una solució òptima per a $S_k = \{item\ etiquetats\ 1, 2, \dots, k\}$ dins una motxilla de amb capacitat \mathbf{w}

Programació Dinàmica

- Definició del sub-problema:
 - calcular $V[\mathbf{k}, \mathbf{w}]$, per tal de trobar una solució òptima per a $S_k = \{item\ etiquetats\ 1, 2, \dots, k\}$ dins una motxilla de amb capacitat \mathbf{w}
 - **Qüestió?** Si assumim que coneixem $V[i, j]$, on $i = 0, 1, \dots, k - 1$, $j = 0, 1, \dots, w$, com podem derivar $V[k, w]$?

Anem a trobar la funció recursiva

Programació Dinàmica

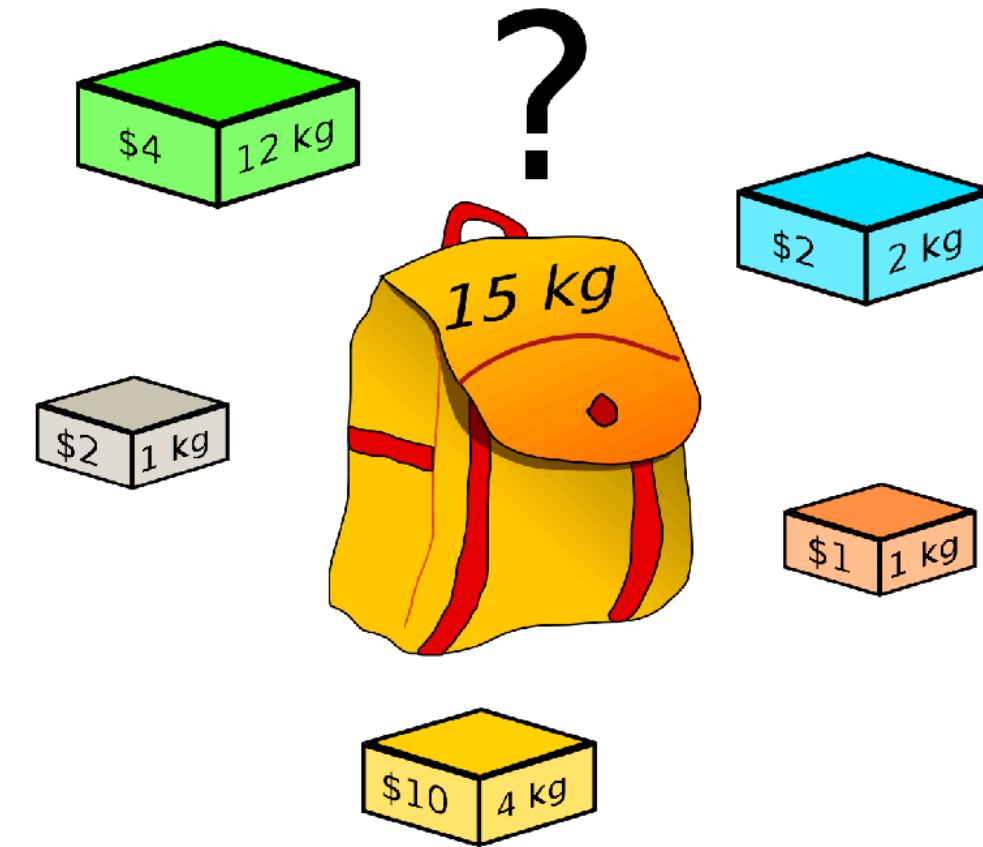
Problema de la motxilla - Funció recursiva

- El millor subconjunt S_k que té pes total $\leq w$, pot contenir l'element k.
 - **Primer cas:** $w_k > w$. L'element k **no pot ser part de la solució**, ja que si ho fos, el pes total seria $> w$.
 - **Segon cas:** $w_k \leq w$. L'element k **pot** ser part de la solució, i únicament l'agafem si forma part de la millor solució.

$$V[k, w] = \begin{cases} V[k-1, w] & \text{if } w_k > w \\ \max\{V[k-1, w], V[k-1, w - w_k] + b_k\} & \text{sino} \end{cases}$$

Programació Dinàmica

Problema de la motxilla



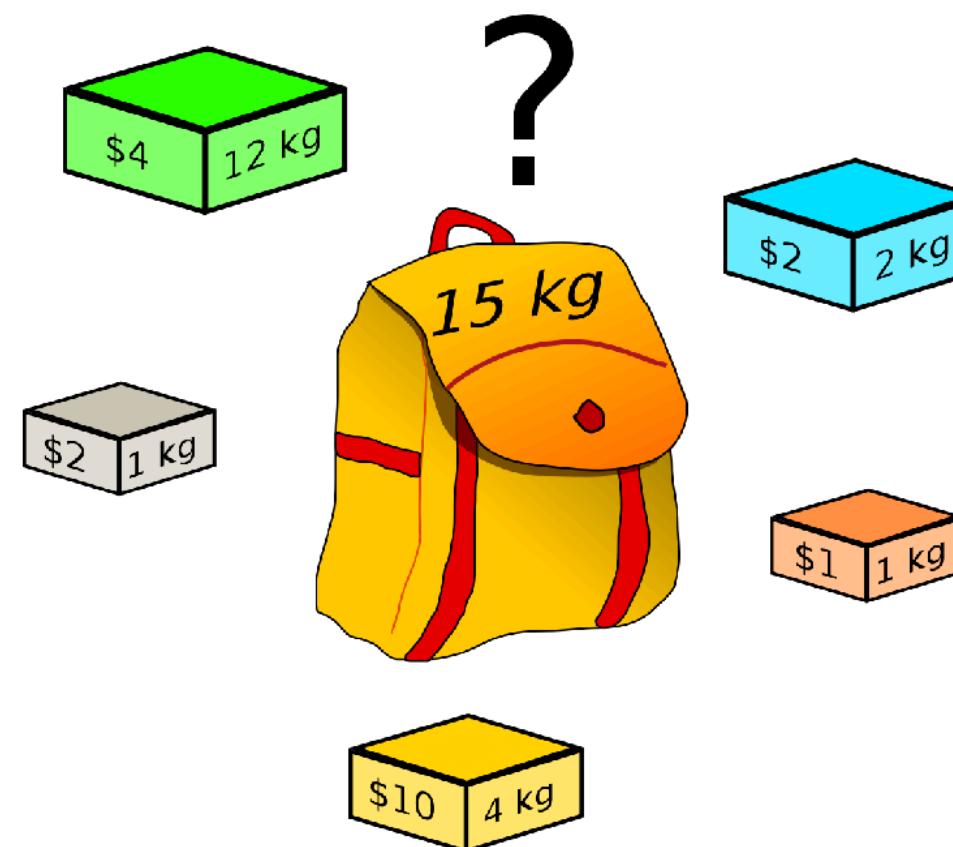
Solució per un màxim de 4 elements i una motxilla de màxim pes = 6Kg

Dibuixeu tots els càlculs que necessitaríeu calcular.

Article	Pes	Valor	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	1	0	1(A)														
B	1	2		??														
C	2	2																
D	4	10								?								
E	12	4																

Programació Dinàmica

Problema de la motxilla



Article	Pes	Valor	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	1	0	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
B	1	2	0	2(B)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)
C	2	2	0	2(B)	3(AB)	4(BC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)
D	4	10	0	2(B)	3(AB)	3(AB)	10(D)	12(BD)	13(ABD)	14(BCD)	15(ABCD)							
E	12	4	0	2(B)	3(AB)	3(AB)	10(D)	12(BD)	13(ABD)	14(BCD)	15(ABCD)							

Com trobar els elements de la motxilla?

- Tota l'informació es troba a la taula.
- **V[n,W]** és el benefici màxim dels elements que podem carregar dins la motxilla.
 - Siguin $i = n$ i $k = W$
 - Si $V[i, k] \neq V[i - 1, k]$:
 - marcar l'element i com a element present dins la motxilla.
 - $i = i - 1$
 - $k = k - w_i$
 - Si no:
 - $i = i - 1$ # Assumim que l'element i no està dins la motxilla

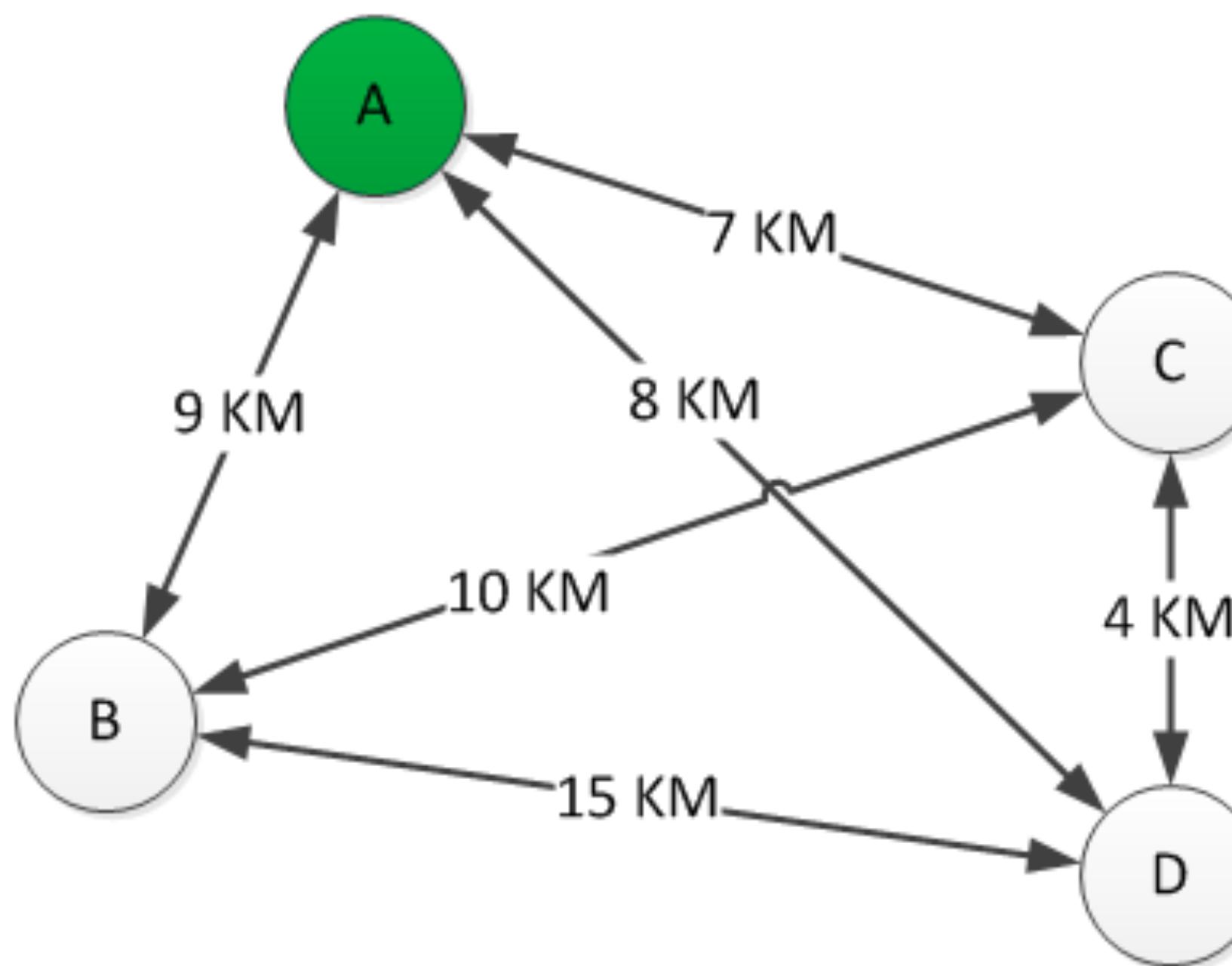
	0	1	2	3	4	5	6	7
Empty	0	0	0	0	0	0	0	0
$v_1=2, w_1=3$	1	0	0	0	2	2	2	2
$v_2=2, w_2=1$	2	0	2	2	2	4	4	4
$v_3=4, w_3=3$	3	0	2	2	4	6	6	8
$v_4=5, w_4=4$	4	0	2	2	4	6	7	9
$v_5=3, w_5=2$	5	0	2	3	5	6	7	9

	0	1	2	3	4	5	6	7
Empty	0	0	0	0	0	0	0	0
$v_1=2, w_1=3$	1	0	0	2	2	2	2	2
$v_2=2, w_2=1$	2	0	2	2	2	4	4	4
$v_3=4, w_3=3$	3	0	2	4	6	6	6	8
$v_4=5, w_4=4$	4	0	2	2	4	6	7	9
$v_5=3, w_5=2$	5	0	2	3	5	6	7	10

The diagram illustrates a grid search or dynamic programming process. The grid has columns labeled 0 through 7 and rows labeled by vertex indices v_i and weights w_i . The first row is labeled "Empty". The second row ($v_1=2, w_1=3$) contains values 1, 0, 0, 2, 2, 2, 2, 2. The third row ($v_2=2, w_2=1$) contains values 2, 0, 2, 2, 2, 4, 4, 4. The fourth row ($v_3=4, w_3=3$) contains values 3, 0, 2, 4, 6, 6, 6, 8. The fifth row ($v_4=5, w_4=4$) contains values 4, 0, 2, 2, 4, 6, 7, 9. The bottom row ($v_5=3, w_5=2$) contains values 5, 0, 2, 3, 5, 6, 7, 10. Cells are highlighted with red or green boxes and arrows show transitions between them. A red box highlights the cell at (v_1, w_1) (value 1). A green box highlights the cell at (v_2, w_2) (value 2). A red box highlights the cell at (v_3, w_3) (value 3). A green box highlights the cell at (v_4, w_4) (value 4). A red box highlights the cell at (v_5, w_5) (value 5). Arrows point from the red-highlighted cells to the green-highlighted cells, indicating a sequence of states or operations.

Programació Dinàmica

Travelling Salesman Problem



Programació Dinàmica

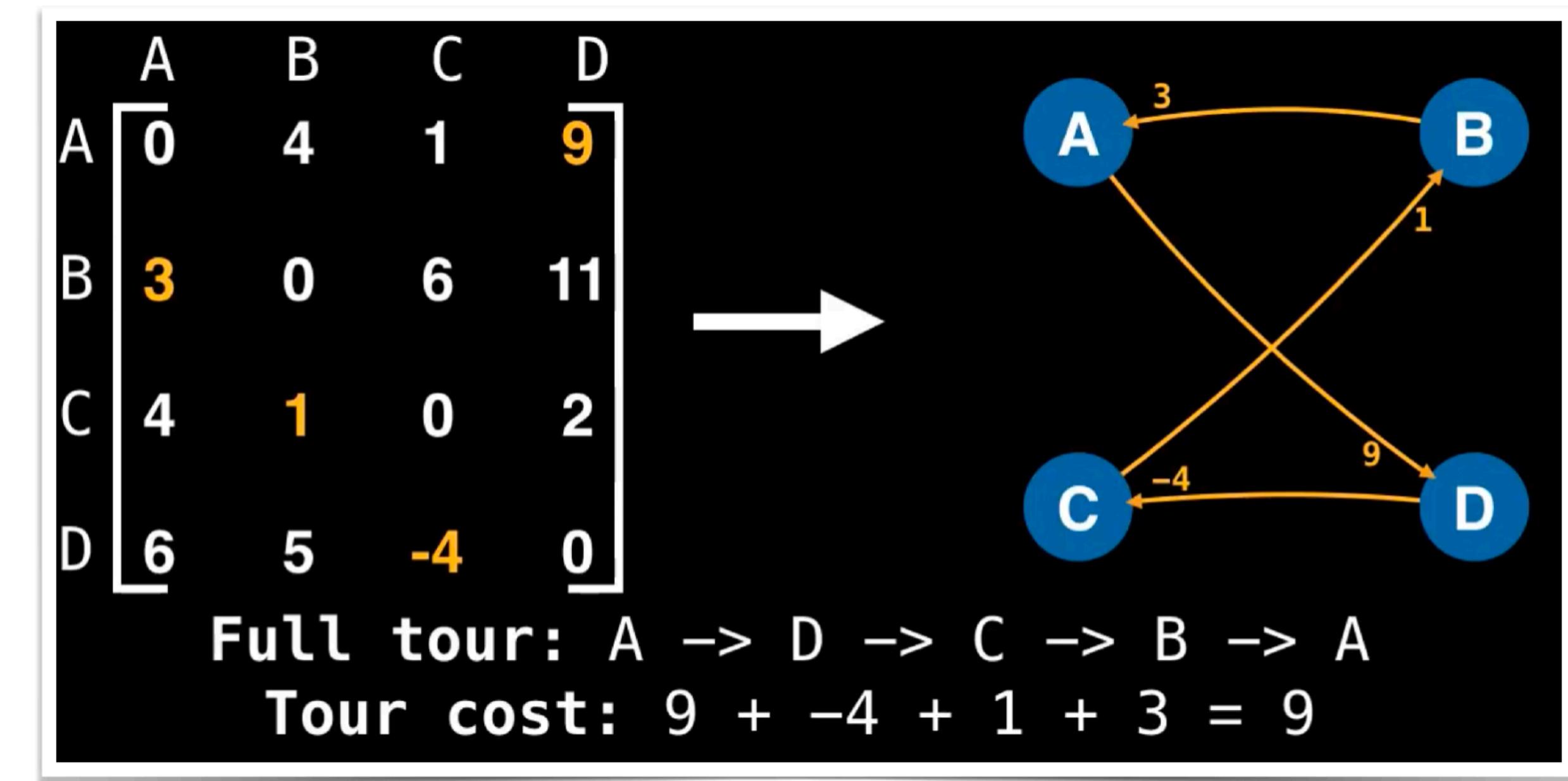
Travelling Salesman Problem

- Problema: Trobar un recorregut de longitud mínima per a un viatger que hagi de visitar diverses ciutats i després tornar al punt d'inici, on la distància existent entre cada parella de ciutats és coneguda.
- És a dir, donat un graf dirigit amb arestes amb cost positiu, es vol trobar un **circuit de longitud mínima** que **comenci i acabi** en el **mateix node passant** exactament un cop per a cada un dels **nodes restants**.

Programació Dinàmica

Travelling Salesman Problem

- Aquest problema consisteix en trobar el circuit hamiltonià



- Problema molt estudiat donat la gran complexitat computacional necessària i la gran quantitat d'aplicacions a la vida real.

Programació Dinàmica

Travelling Salesman Problem - Solució per força bruta

- Trobar la solució mitjançant una cerca exhaustiva implica mirar totes les possibles permutacions dels nodes ordenats, el que implica una complexitat de complexitat **O(n!)**.

	A	B	C	D	Tour	Cost	
A	0	4	1	9	A B C D	18	C A B D 15
B	3	0	6	11	A B D C	15	C A D B 24
C	4	1	0	2	A C B D	19	C B A D 9
D	6	5	-4	0	A C D B	11	C B D A 19
					A D B C	24	C D A B 18
					A D C B	9	C D B A 11
					B A C D	11	D A B C 18
					B A D C	9	D A C B 19
					B C A D	24	D B A C 11
					B C D A	18	D B C A 24
					B D A C	19	D C A B 15
					B D C A	15	D C B A 9

Programació Dinàmica

Travelling Salesman Problem

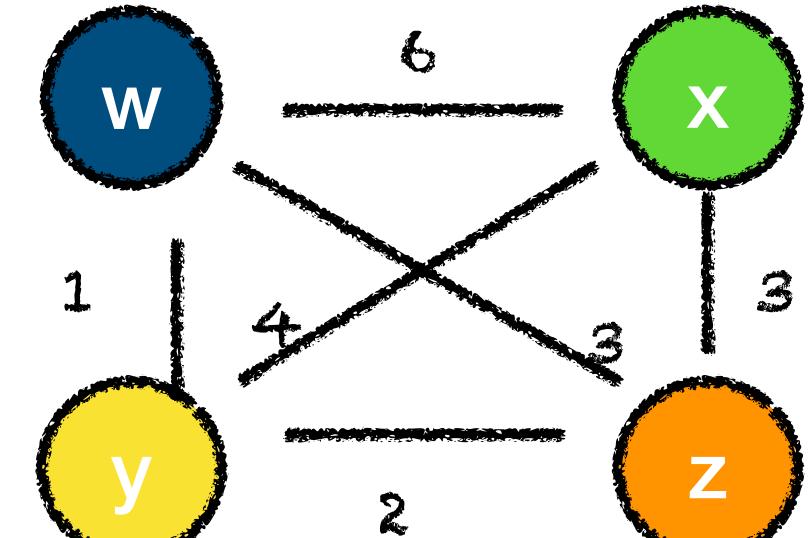
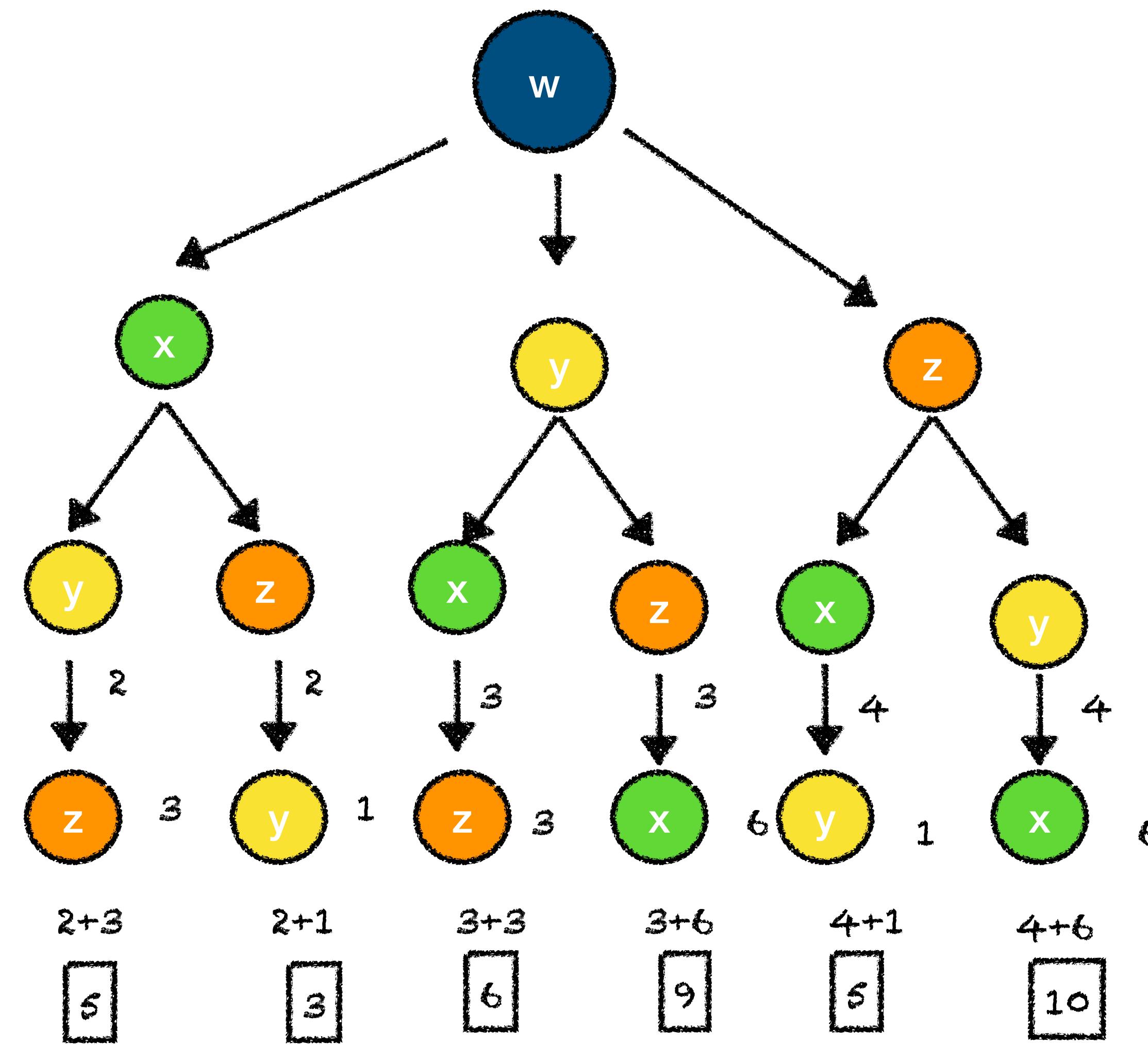
- La solució mitjançant programació dinàmica millora significativament la complexitat en el temps, passem d'una complexitat $O(n!)$ a $O(n^2 2^n)$

	n=2	n=5	n=10	n=20
Força Bruta	2	120	36.288.000	1.4e ¹⁸
DP	16	800	102.400	419.430.400

- La idea principal consisteix en calcular la solució òptima per a tots els subcamins de longitud N utilitzant informació que ja coneixem de les solicions parciales òptimes amb longitud N-1

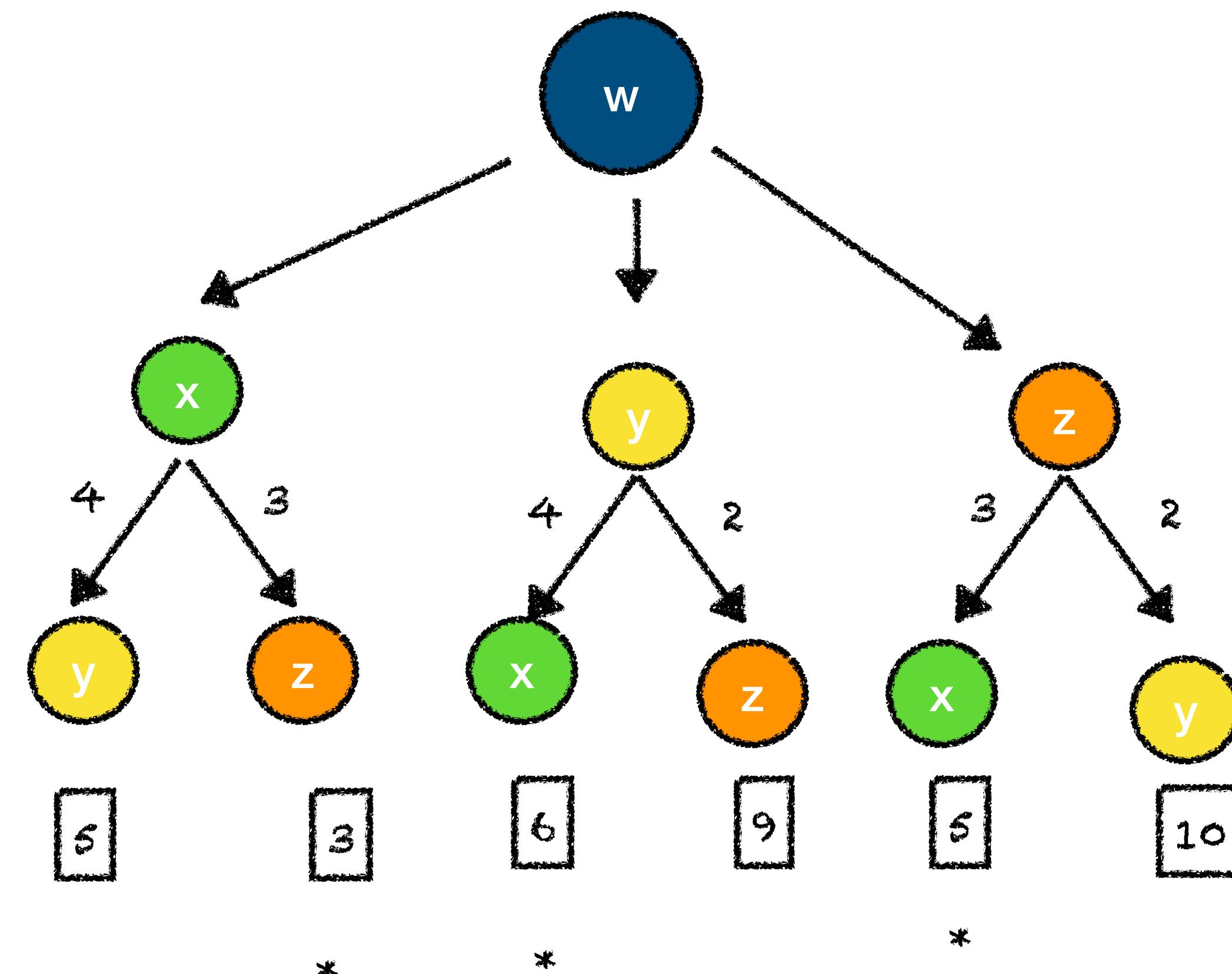
Programació Dinàmica

Travelling Salesman Problem - Solució amb força bruta

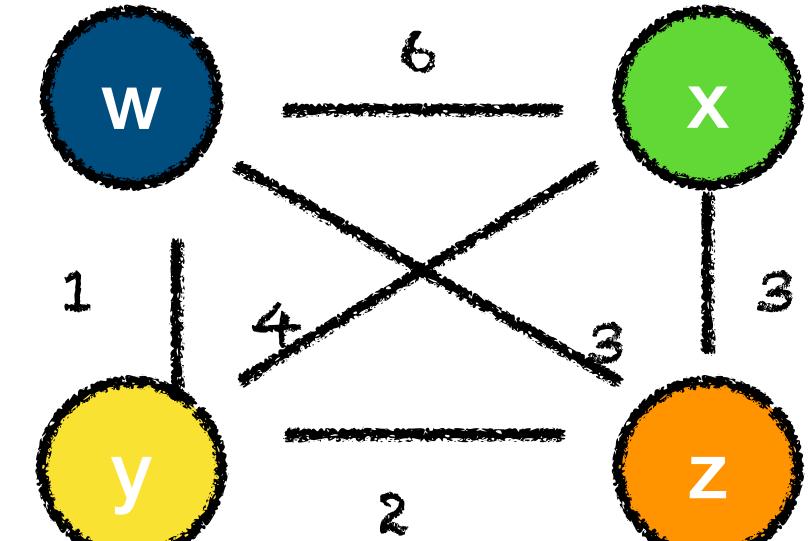


Programació Dinàmica

Travelling Salesman Problem - Solució amb força bruta

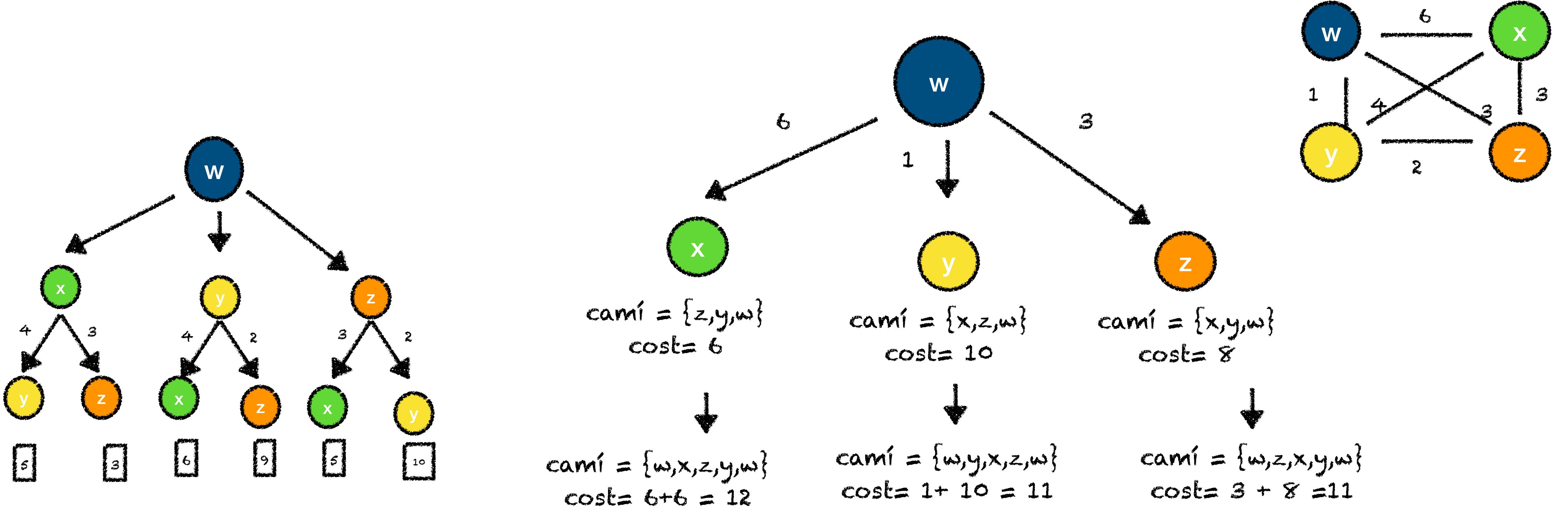


* camins mínims



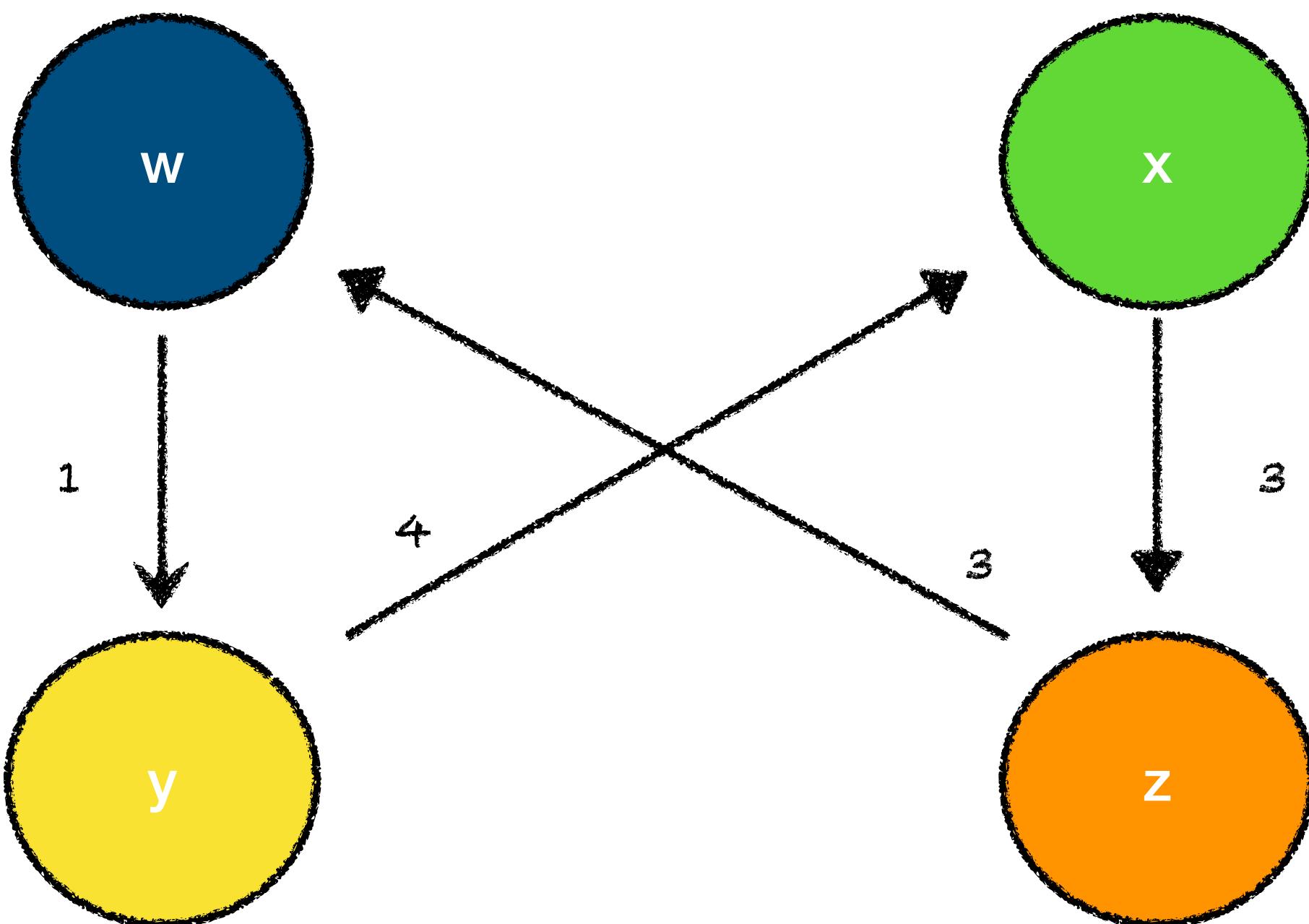
Programació Dinàmica

Travelling Salesman Problem - Solució amb força bruta



Programació Dinàmica

Travelling Salesman Problem

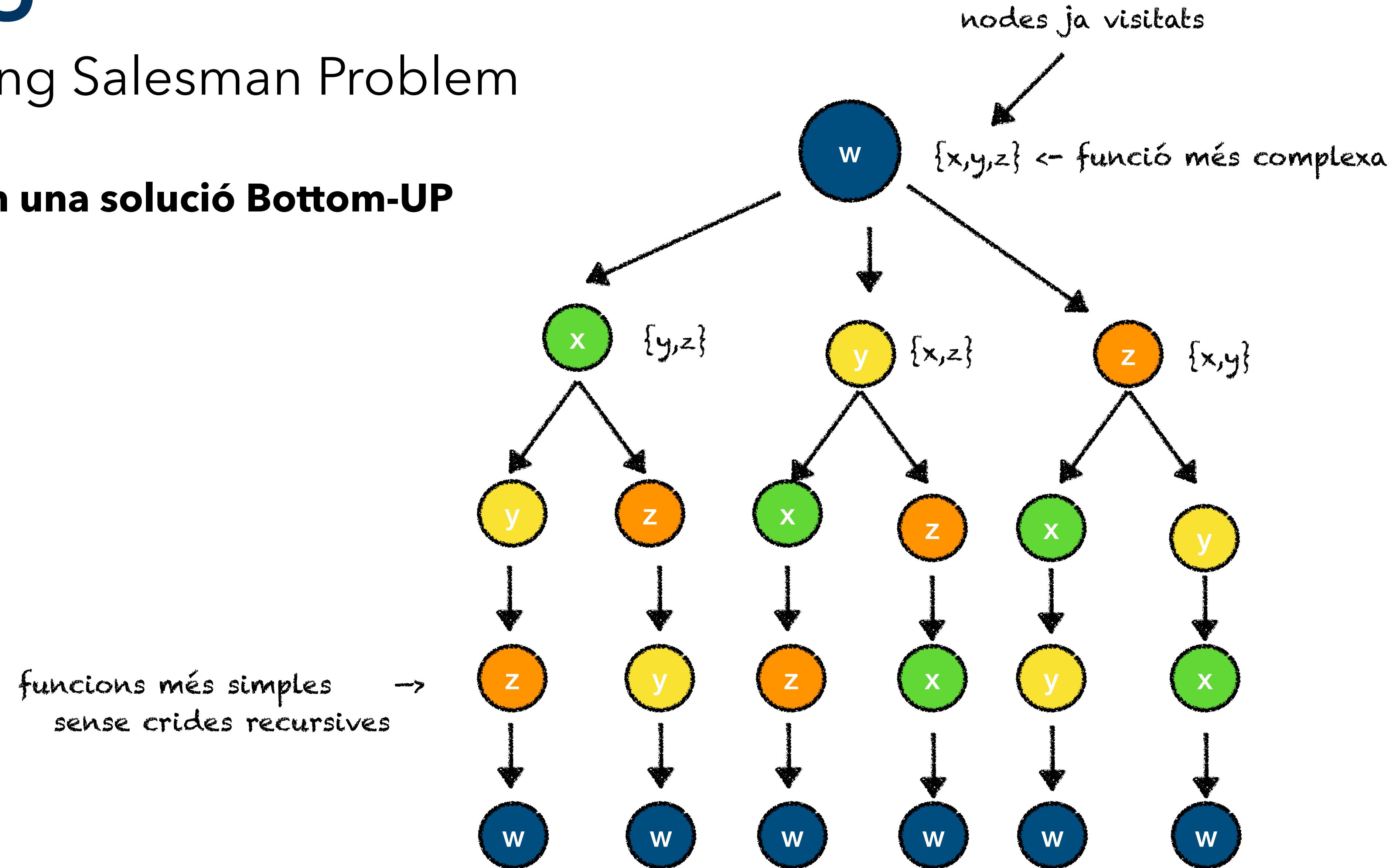


Acabem de veure una solució per força bruta utilitzant una estratègia Top-DOWN

Programació Dinàmica

Travelling Salesman Problem

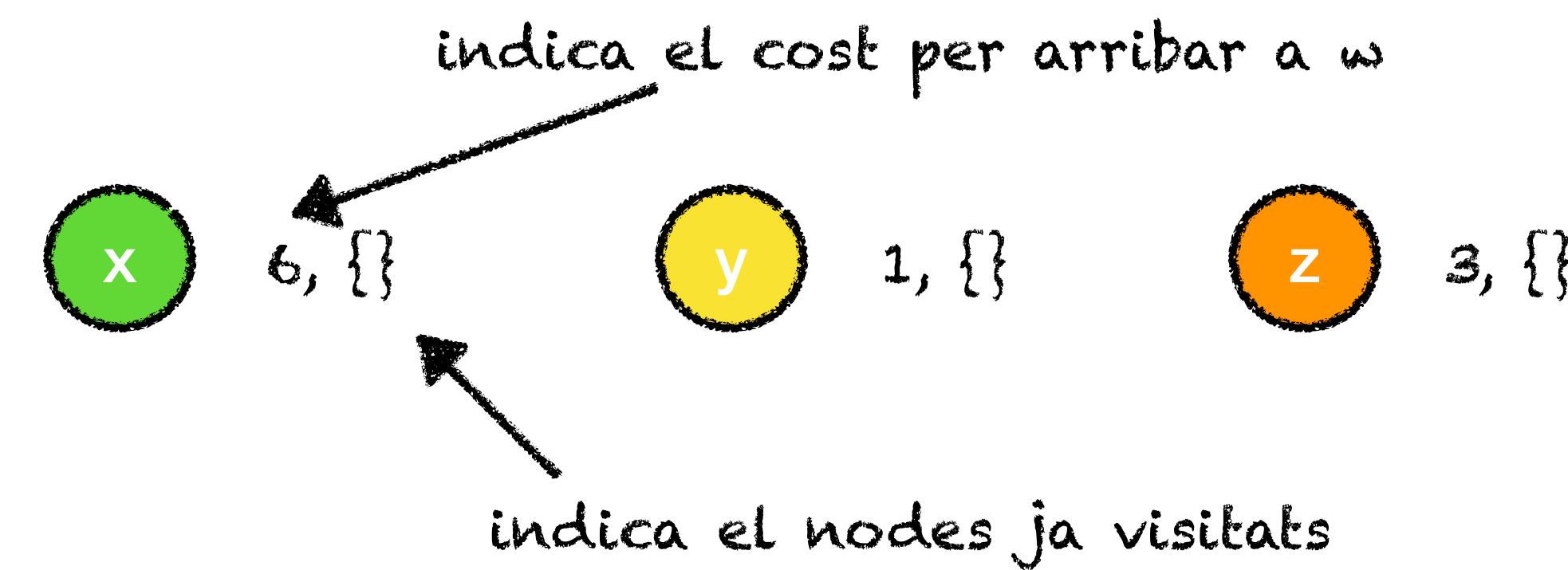
- Pensem una solució Bottom-UP



Programació Dinàmica

Travelling Salesman Problem

- Pensem una solució Bottom-UP

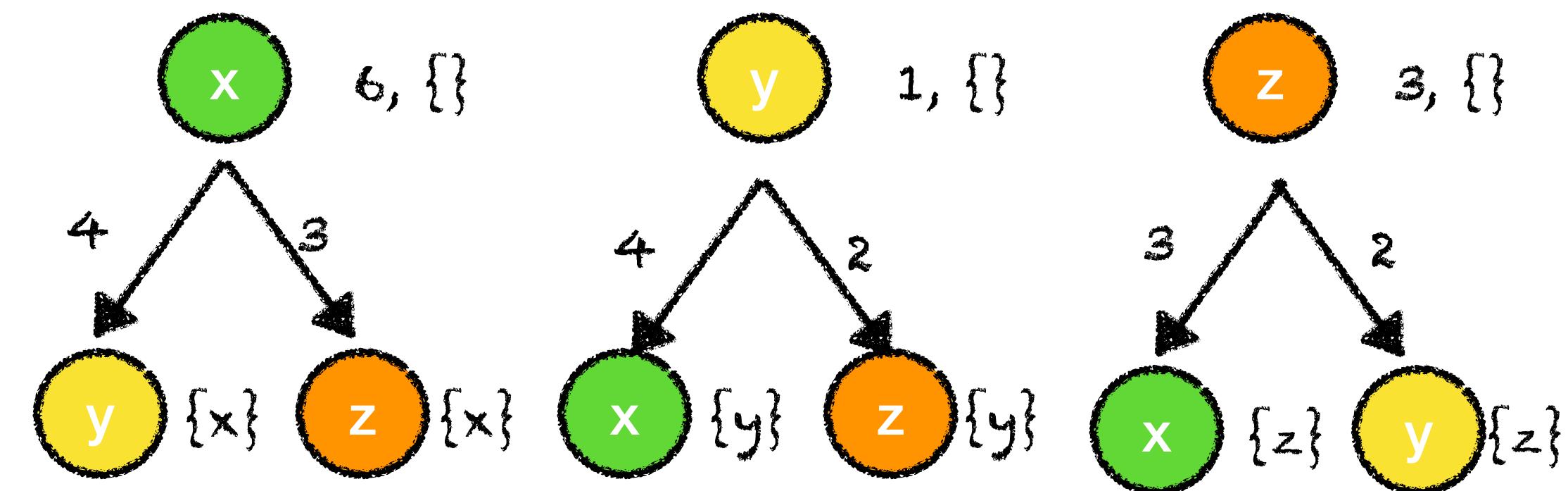


- Aquestes son les tres funcions més simples. Totes elles acaben al node **w**

Programació Dinàmica

Travelling Salesman Problem

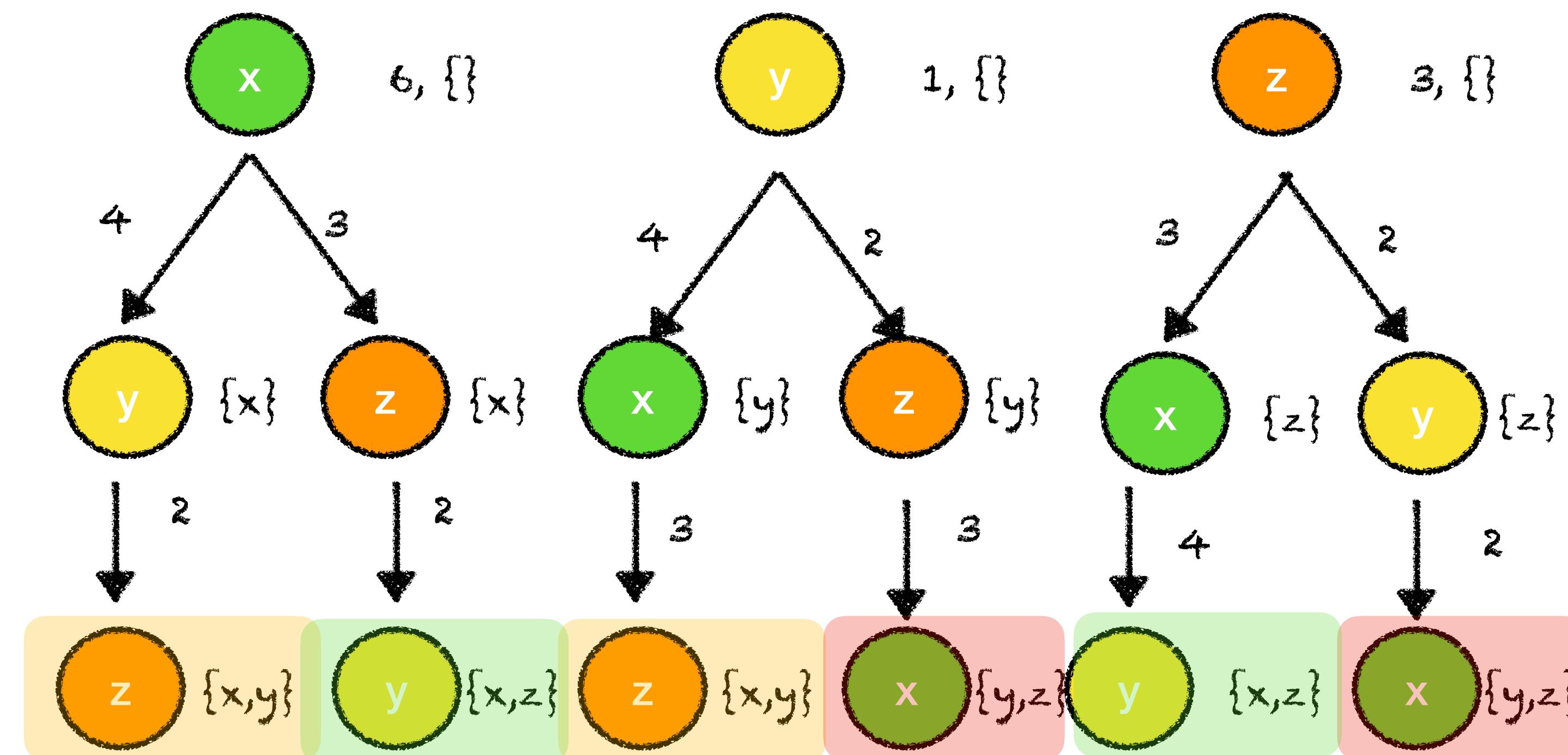
- **Pensem una solució Bottom-UP**



Programació Dinàmica

Travelling Salesman Problem

- Pensem una solució Bottom-UP

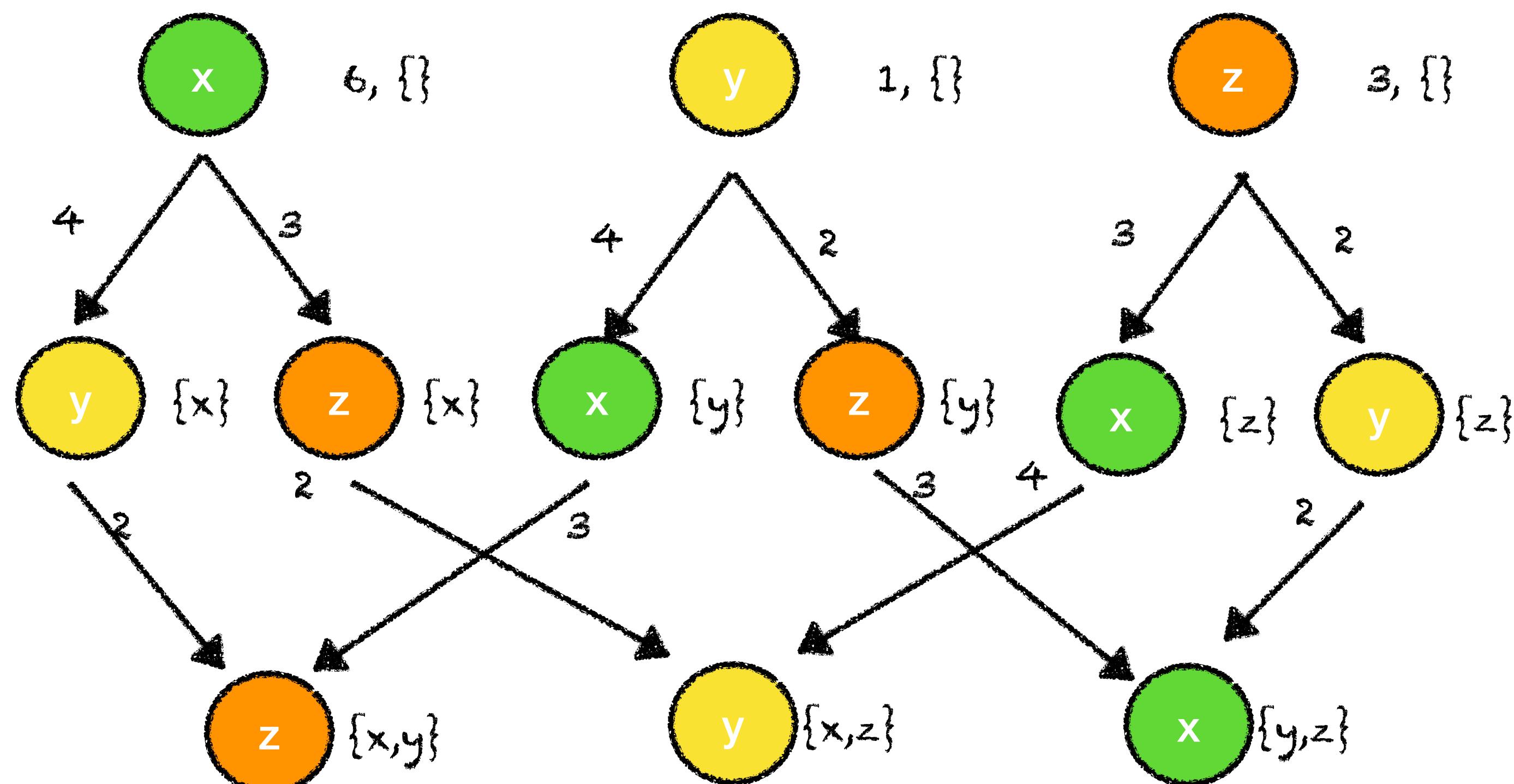


Tenim problemes superposats!!! → PODEM aplicar DP

Programació Dinàmica

Travelling Salesman Problem

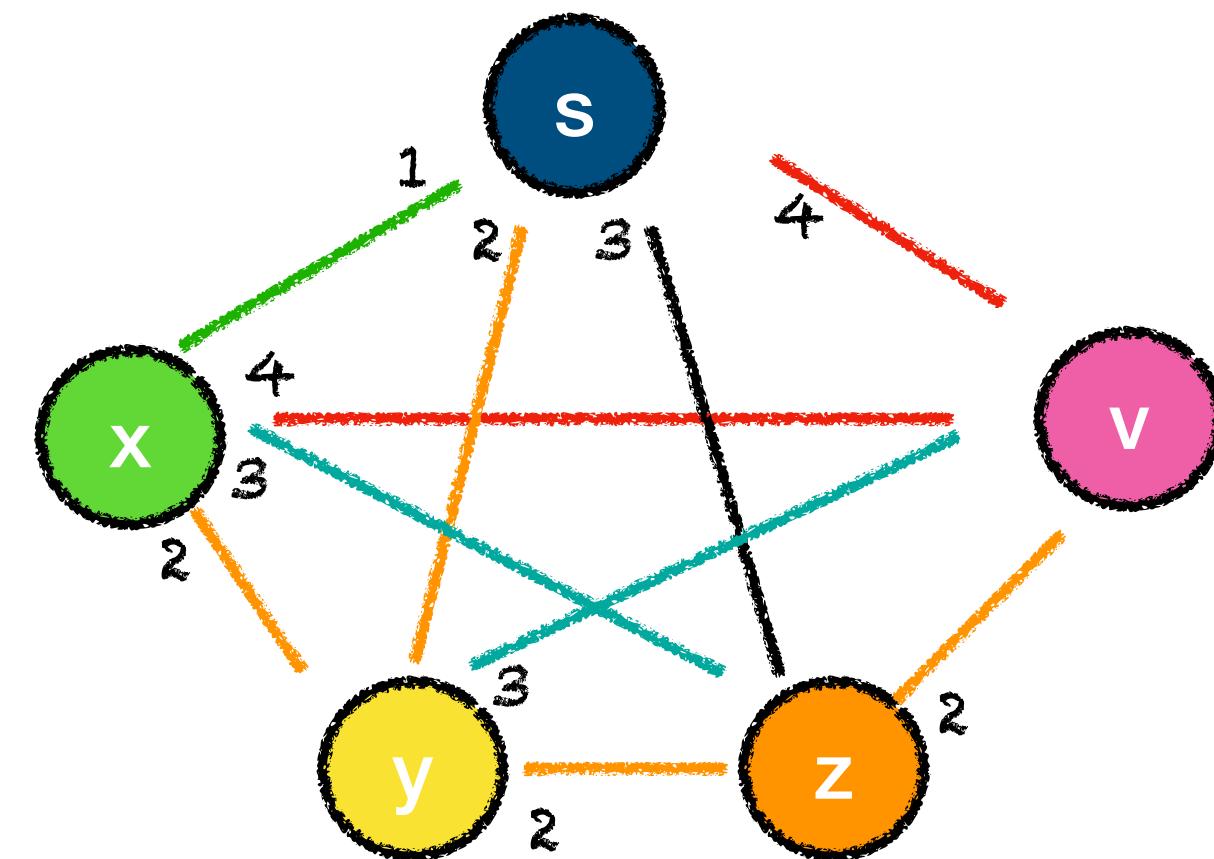
- Pensem una solució Bottom-UP



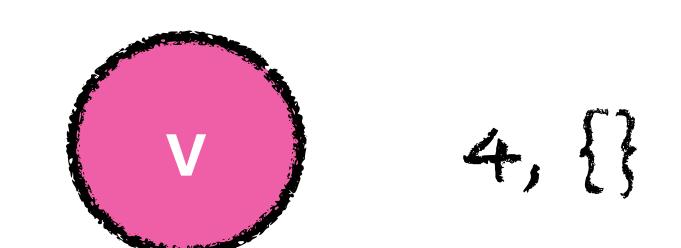
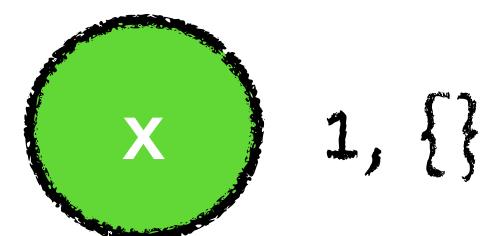
Tenim problemes superposats!!! → PODEM aplicar DP

Programació Dinàmica

Travelling Salesman Problem

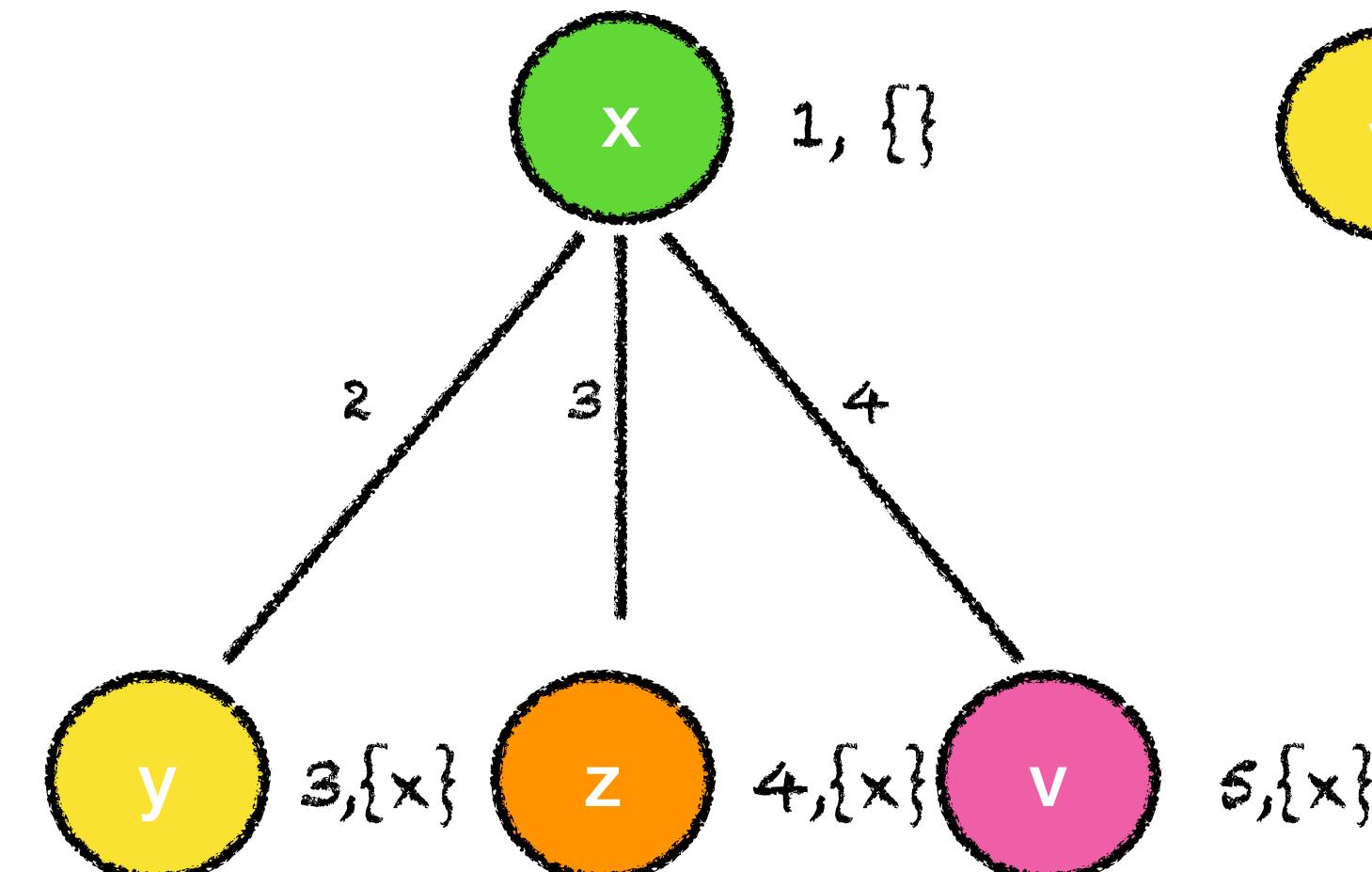
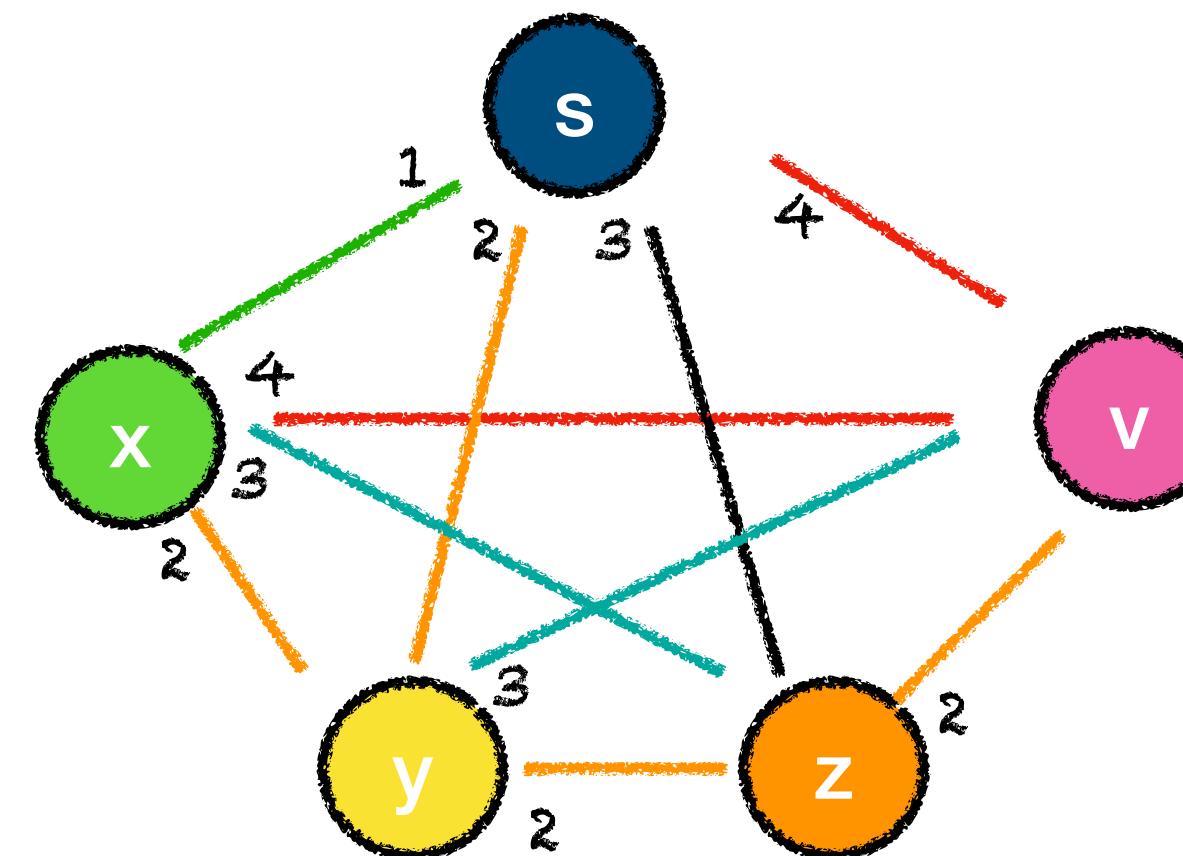


Anem a veure què passa amb
aquest exemple



Programació Dinàmica

Travelling Salesman Problem



Anem a veure què passa amb
aquest exemple



Solució amb programació Dinàmica

- La idea principal consisteix en calcular la solució òptima per a tots els subcamins de **longitud N** utilitzant informació que ja coneixem de les solucions parcials òptimes amb longitud N-1

Programació Dinàmica

Travelling Salesman Problem

- $D(V_i, S)$ és la longitud del camí mínim sortint del vèrtex V_i i passant per cadascun dels vèrtexs del conjunt S i tornat al vèrtex V_i .
- La funció de recurrència es pot definir com:

$$g(i, S) = \{L_{0i}\} \quad \text{si } S = \{\}$$

$$g(i, S) = \min_{j \in S} \{L_{ji} + g(j, S - \{j\})\}$$

- $g(i, S)$ serà la longitud del camí mínim sortint del vèrtex i que passa per tots els vèrtex del conjunt S i torna al vèrtex i

Programació Dinàmica

Travelling Salesman Problem

- Sub-estructura òptima?

Programació Dinàmica

Travelling Salesman Problem

- Sub-estructura òptima?
- Què necessitem emmagatzemar?
 - Per calcular la solució òptima per camins de longitud 3 necessitem recordar (i guardar) 2 coses per a cada camí de longitud 2:
 - el **conjunt dels nodes visitats**
 - l'**index** de l'últim node visitat
 - Quina l'espai necessari per emmagatzemar aquesta informació?

Programació Dinàmica

Travelling Salesman Problem

```
function algorithm TSP (G, n)
    for k := 2 to n do
        C({k}, k) := d1,k
    end for

    for s := 2 to n-1 do
        for all S ⊆ {2, . . . , n}, |S| = s do
            for all k ∈ S do
                C(S, k) := minm≠k, m∈S [C(S\{k}, m) + dm,k]
            end for
        end for
    end for

    opt := mink≠1 [C({2, 3, . . . , n}, k) + dk,1]
    return (opt)
end
```

Exercici

- Mirem les crides amb 5 nodes

Programació Dinàmica

Longest Increasing Subsequence (LIS)

- El problema de la seqüència creixent màxima, consisteix amb trobar una subseqüència d'una seqüència donada, tal que els elements de la subseqüència estiguin ordenats, de menor a major, i la subseqüència sigui el més gran possible. Per exemple:
 - Donada la seqüència: 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
 - Una subseqüència creixent màxima pot ser: **0, 2, 6, 9, 11 , 15**
 - Aquesta subseqüència té longitud 6, i la seqüència d'entrada donada no té cap subseqüència creixent de mida 7. La subseqüència anterior no és l'única de longitud 6, sinó que podem tindre diverses com per exemple:
- La solució d'aquest problema amb programació dinàmica ve donat per la següent **funció recurrent**:
 - $$d[i] = \max \left(1, \max_{\substack{j < i \\ a[j] < a[i]}} (d[j] + 1) \right)$$
 - amb el cas base: $d(0) = 0$

Programació Dinàmica

Longest Palindromic Subsequence

- Donada una seqüència de text, trobar la longitud del palíndrom més llarg.
- Exemple: **BBABCBCAB**
 - Solució: longitud 7 : BABCBAB
 - **Pista:** Considereu un exemple "cbebc". Si ja sabem que "**bcb**" és un palíndrom, és obvi que "cbebc" ha de ser un palíndrom ja que les lletres d'inici i finals són les mateixes.

Programació Dinàmica

Longest Palindromic Subsequence

- Exercici:
 - **Quins són els casos base?**
 - **Definiu funció recursiva**
 - Quina espai necessitem per emmagatzemar les dades?
 - Quina complexitat té el còdi?

Programació Dinàmica

Longest Palindromic Subsequence

- Exercici:

- **Quins són els casos base?**

- Si només tenim un caràcter -> **return 0**
 - Si tenim 2 caràcters, i aquests son iguals **return 2**

- **Definiu funció recursiva**

$$lps(seq, i, j) = \max(lps(seq, i, j - 1), lps(seq, i + 1, j))$$

- Quina espai necessitem per emmagatzemar les dades?
 - Quina complexitat té el còdi?

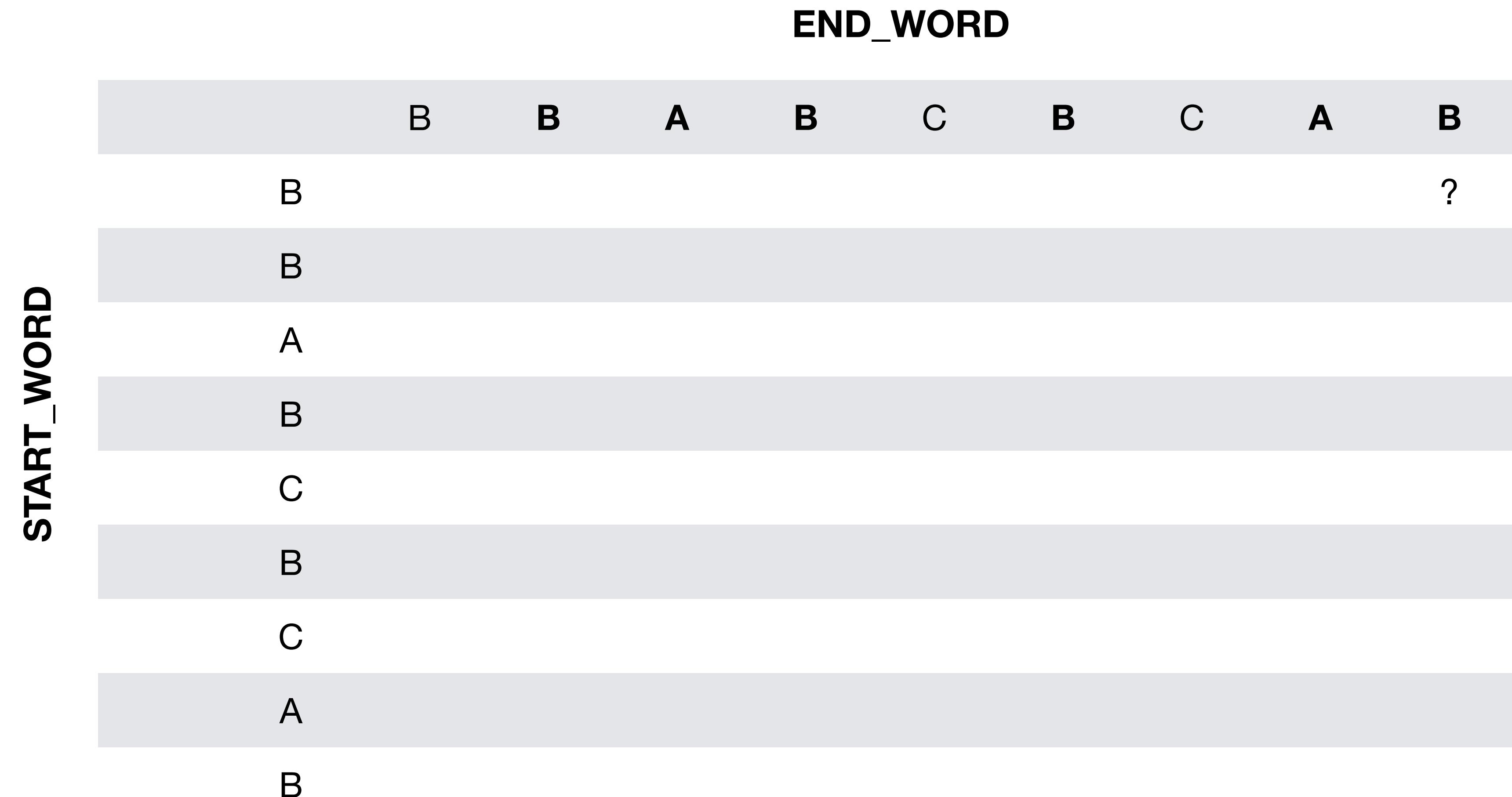
$$\max(\quad \text{lps}(ABAB), \quad \text{lps}(BABB) \quad)$$

lps(ABABB)

The diagram shows the recursive call lps(ABABB) at the top. Two arrows point downwards from it to two separate terms in the max function: lps(ABAB) and lps(BABB) .

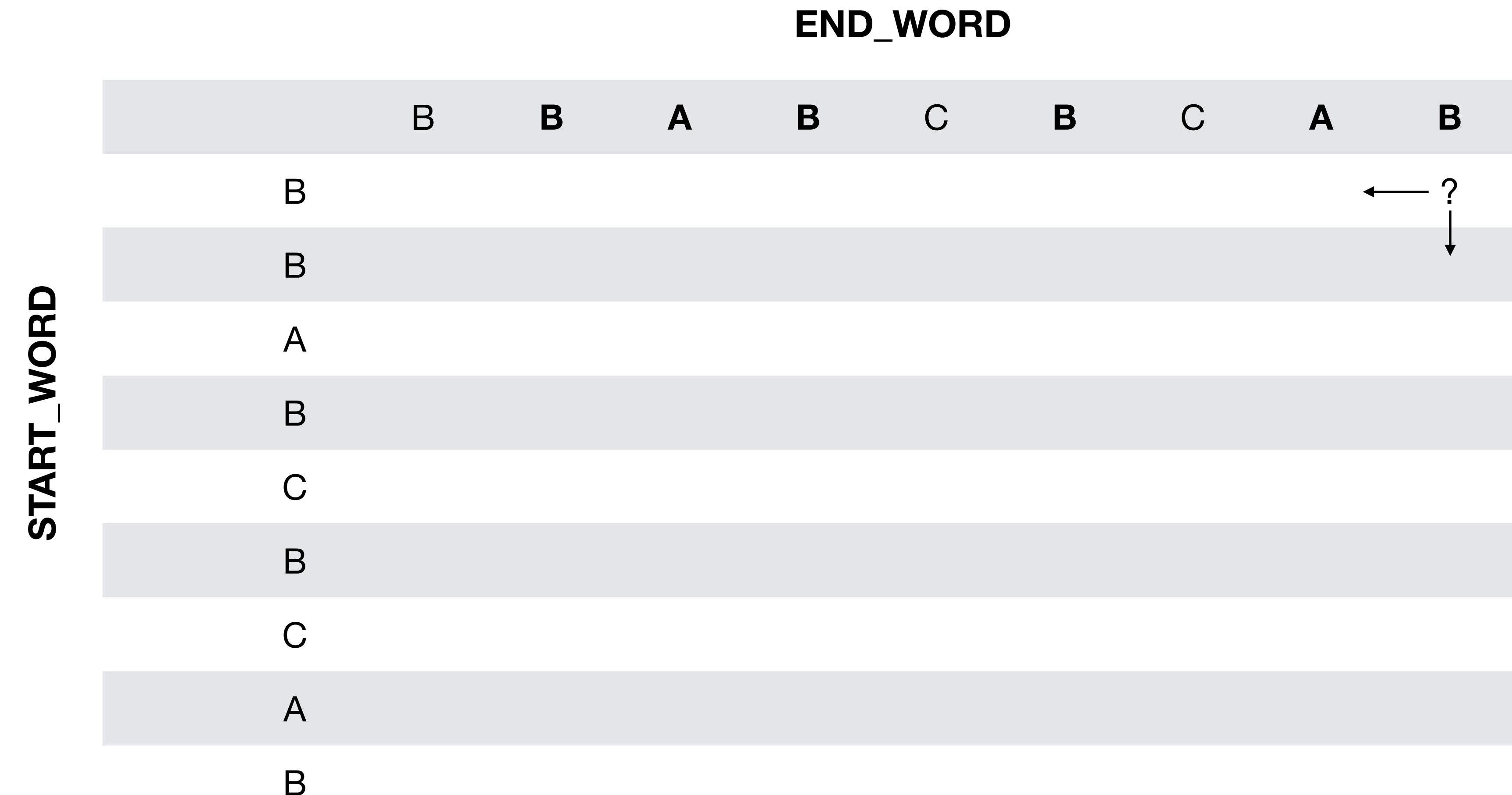
Programació Dinàmica

Longest Palindromic Subsequence



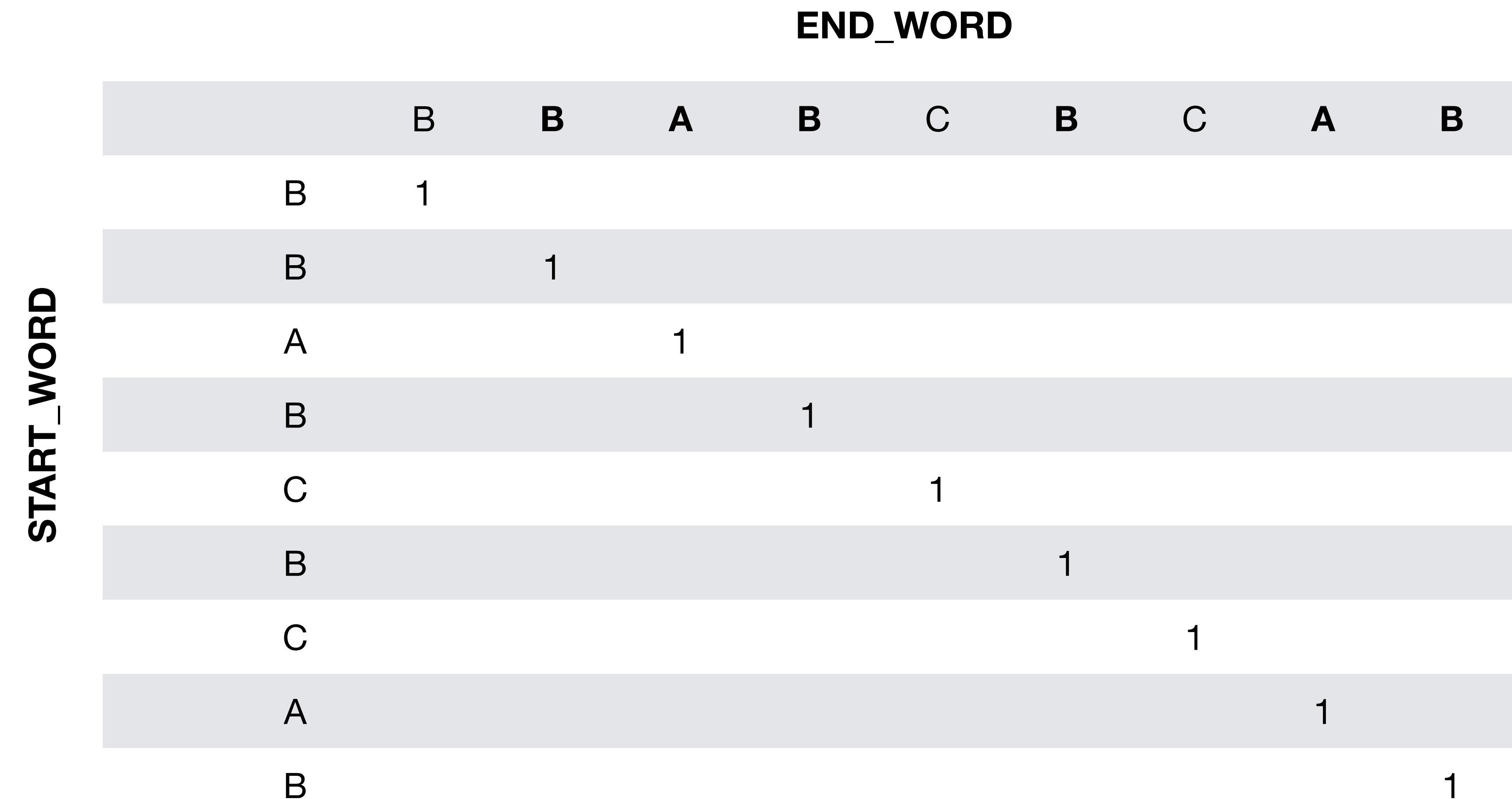
Programació Dinàmica

Longest Palindromic Subsequence



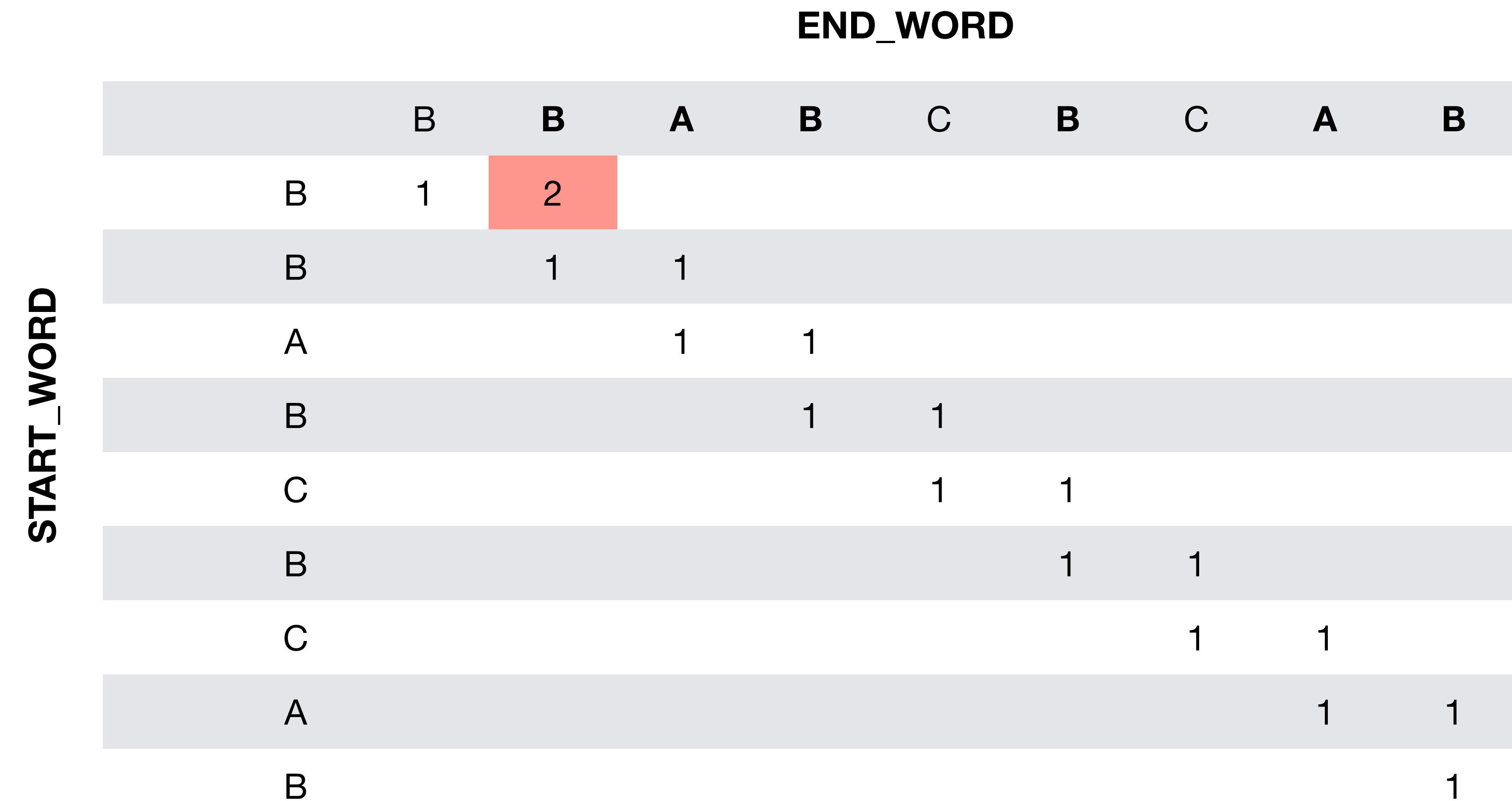
Programació Dinàmica

Longest Palindromic Subsequence



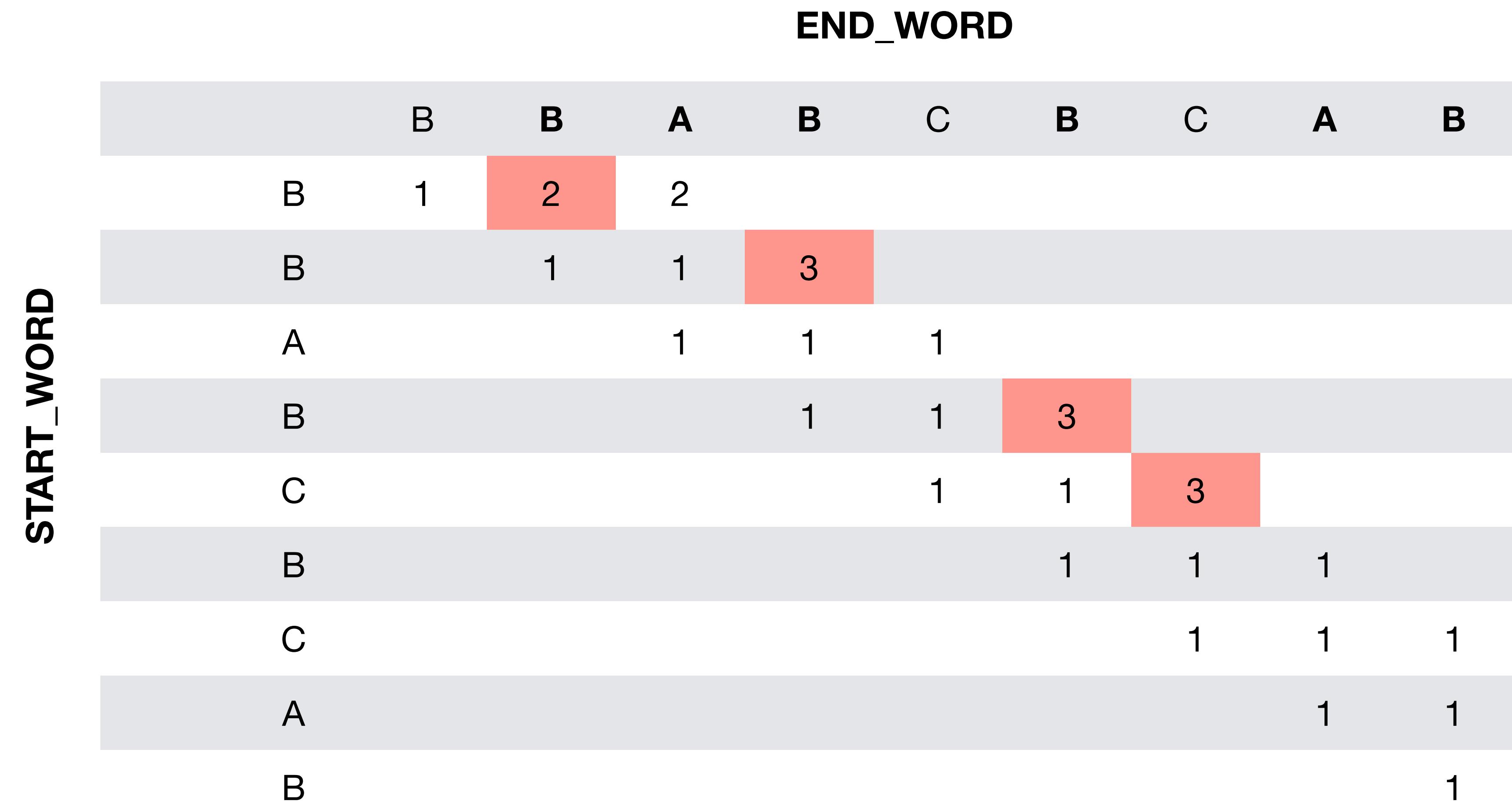
Programació Dinàmica

Longest Palindromic Subsequence



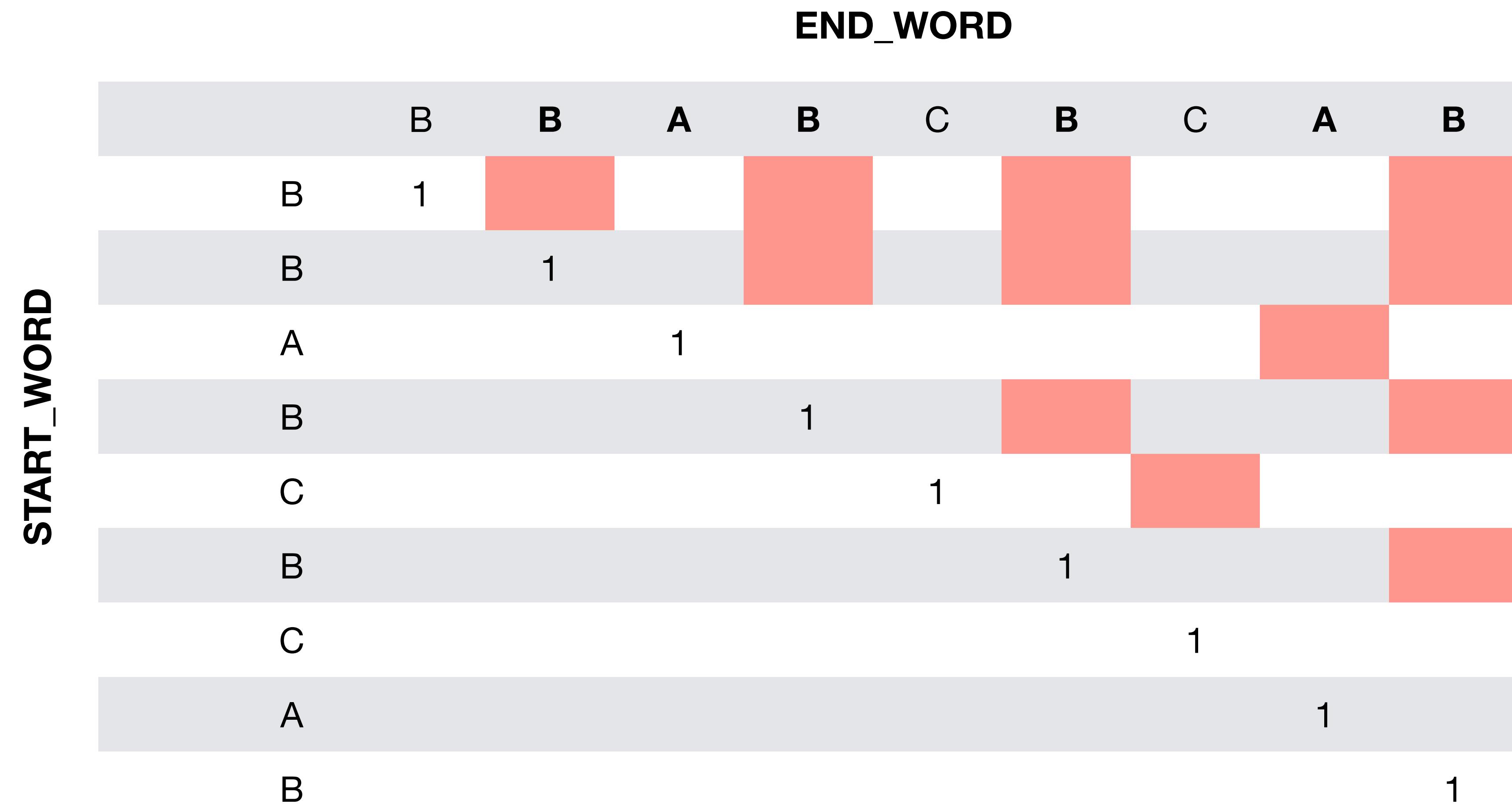
Programació Dinàmica

Longest Palindromic Subsequence



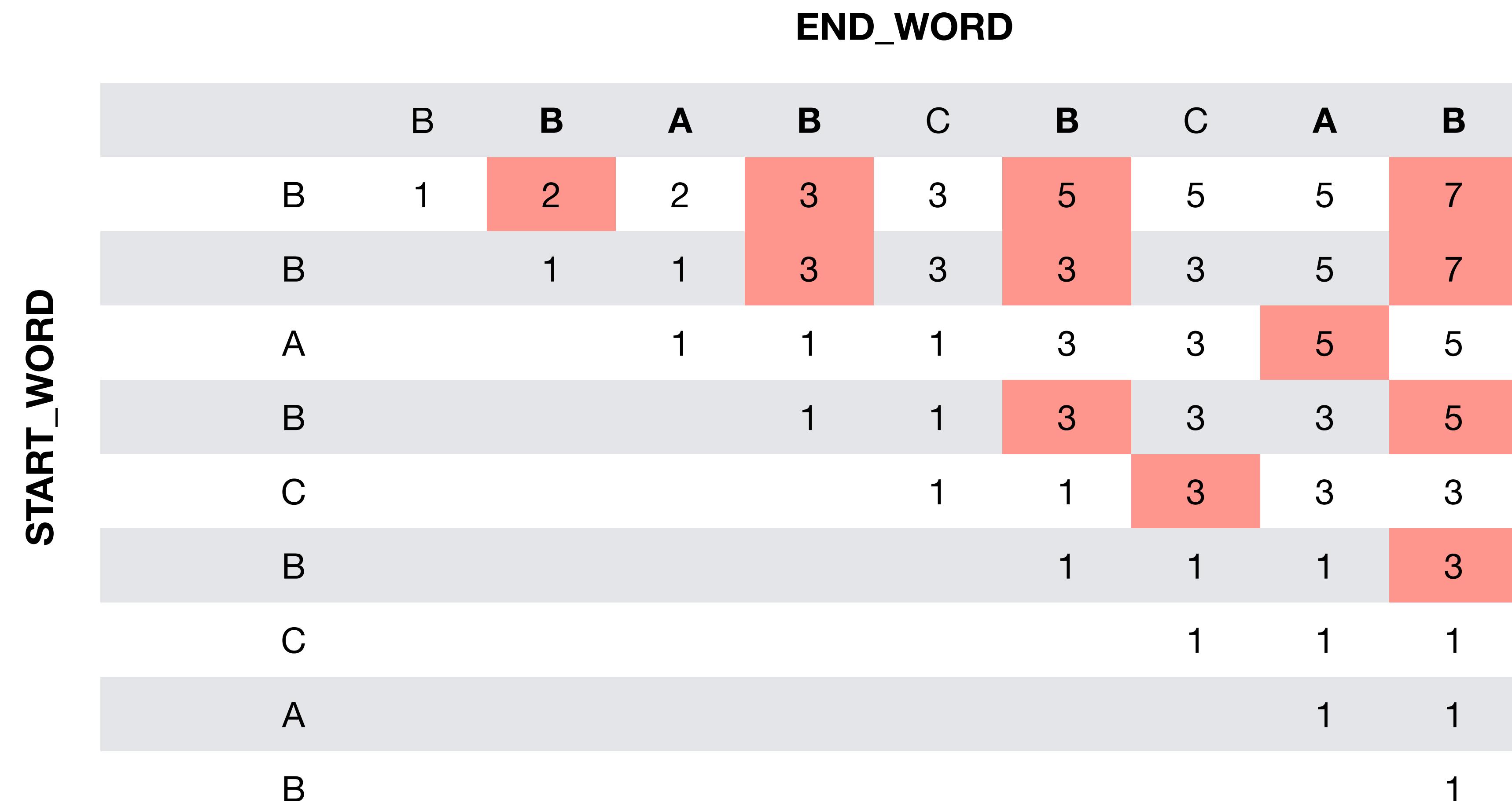
Programació Dinàmica

Longest Palindromic Subsequence



Programació Dinàmica

Longest Palindromic Subsequence



Programació Dinàmica

◀ Hide

Keyboards in Concert

Olav has some electronic keyboards and would like to play a tune. Unfortunately all of Olav's keyboards are broken so each of them can only play some of the notes. By switching which instrument he is using he will be able to play the whole tune, but moving keyboards around is annoying so he would like to minimize the amount of times he has to switch. Can you help Olav figure out the minimum number of keyboard switches needed to play the entire song?



CC-BY 2.0, Daniel Spils via Flickr

Input

The first line of input is two space separated integers; n ($1 \leq n \leq 1\,000$) the number of instruments, and m ($1 \leq m \leq 1\,000$), the number of notes in the tune. This is followed by n lines, each starting with an integer k_i ($1 \leq k_i \leq 1\,000$), the number of notes playable by instrument i , followed by k_i pairwise distinct integers $\ell_1, \ell_2, \dots, \ell_{k_i}$, the notes that instrument i can play ($1 \leq \ell_j \leq 1\,000$). Finally, there is a line with m space-separated integers – the notes of the tune in order.

Output

The minimum number of times Olav needs to switch the instrument he is using during the tune.

Sample Input 1

```
2 10
2 1 2
2 2 3
1 2 1 2 3 3 2 3 1 3
```

Sample Output 1

```
3
```

Programació Dinàmica

Cent Saving

- Quan arribo a la caixa registradora, poso tots els meus n articles a la cinta transportadora i espero fins que tots els altres clients que tinc davant meu siguin atesos.
- Mentre espero, m'adono que aquest supermercat recentment va començar a arrodonir el preu total d'una compra al múltiple més proper de 10 cèntims (amb 5 cèntims arrodonit a l'alça). Per exemple, 94 cèntims s'arrodoneixen a 90 cèntims, mentre que 95 s'arrodoneixen a 100.
- És possible dividir la meva compra en grups i pagar les peces per separat. Em pregunto on col·locar els d divisors per minimitzar el cost total de la meva compra. Donat que s'esgota el temps, **no reordenem els elements de la cinta.**



Programació Dinàmica

Exemple

```
items, divisors possibles  
[13,21,55,60,42], 1
```

Possibles solucions

```
[ ] [13 21 55 60 42] -> round(0) + round(191) = 0 + 190 = 190  
[13] [21 55 60 42] -> round(13) + round(178) = 10 + 180 = 190  
[13 21] [55 60 42] -> round(34) + round(157) = 30 + 160 = 190  
[13 21 55] [60 42] -> round(89) + round(102) = 90 + 100 = 190  
[13 21 55 60] [42] -> round(149) + round(42) = 150 + 40 = 190
```



```
items, divisors possibles  
[2,2,2,2], 1
```

Possibles solucions

```
[2] [2 2 2] -> round(2)+ round(6) = 10  
[2 2] [2 2] -> round(4)+ round(4) = 0  
[2 2 2] [2] -> round(6)+ round(2) = 10  
[2 2 2 2] [] -> round(8)+ round(0) = 10
```

```
items, divisors possibles  
[2,2,2,2,2,2], 2
```

```
[2] [2] [2 2 2 2] -> round(2)+ round(2) + round(8) = 10  
[2 2] [2 2] [2 2] -> round(4)+ round(4) + round(4) = 0  
...
```