



# Programació Dinàmica

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2019-2020

# Exercici

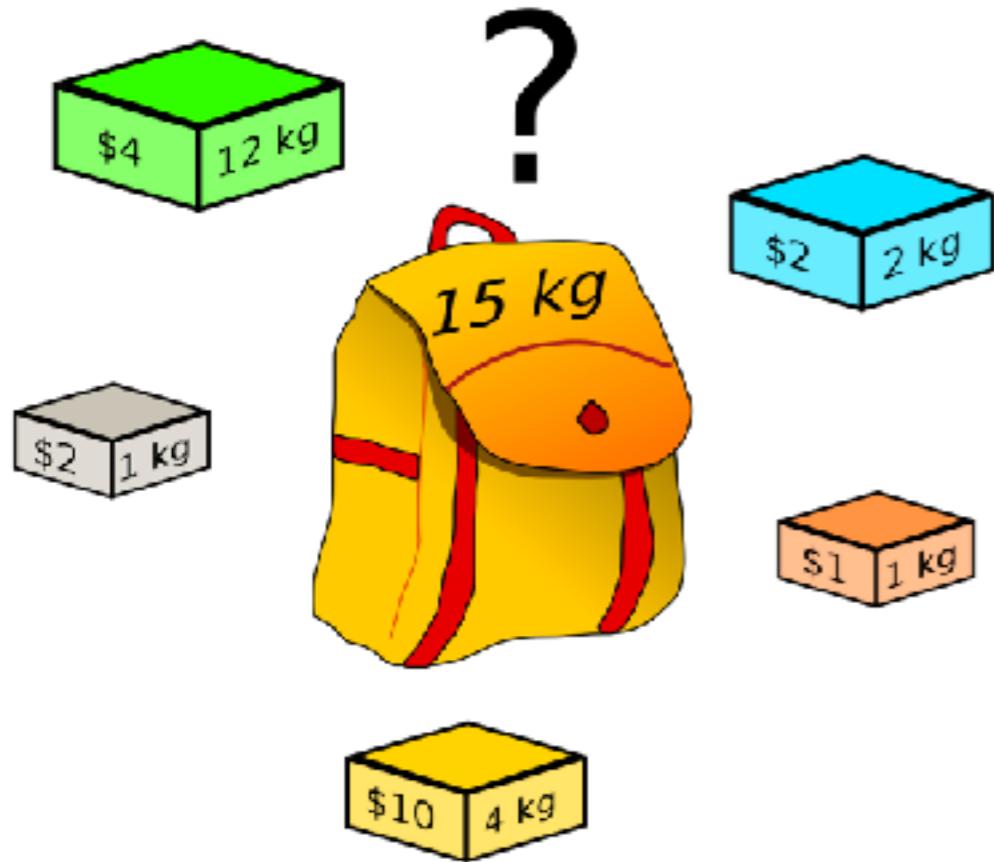
- **El viatge més barat pel riu**

- 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:

$$C(i, j) = \begin{cases} 0 & \text{si } i = j \\ \min(T(i, k) + C(k, j), T(i, j)) & \text{si } i < k \leq j \end{cases}$$

- **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 de portar el màxim valor  $v$  possible amb el mínim pes  $w$ ?

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

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

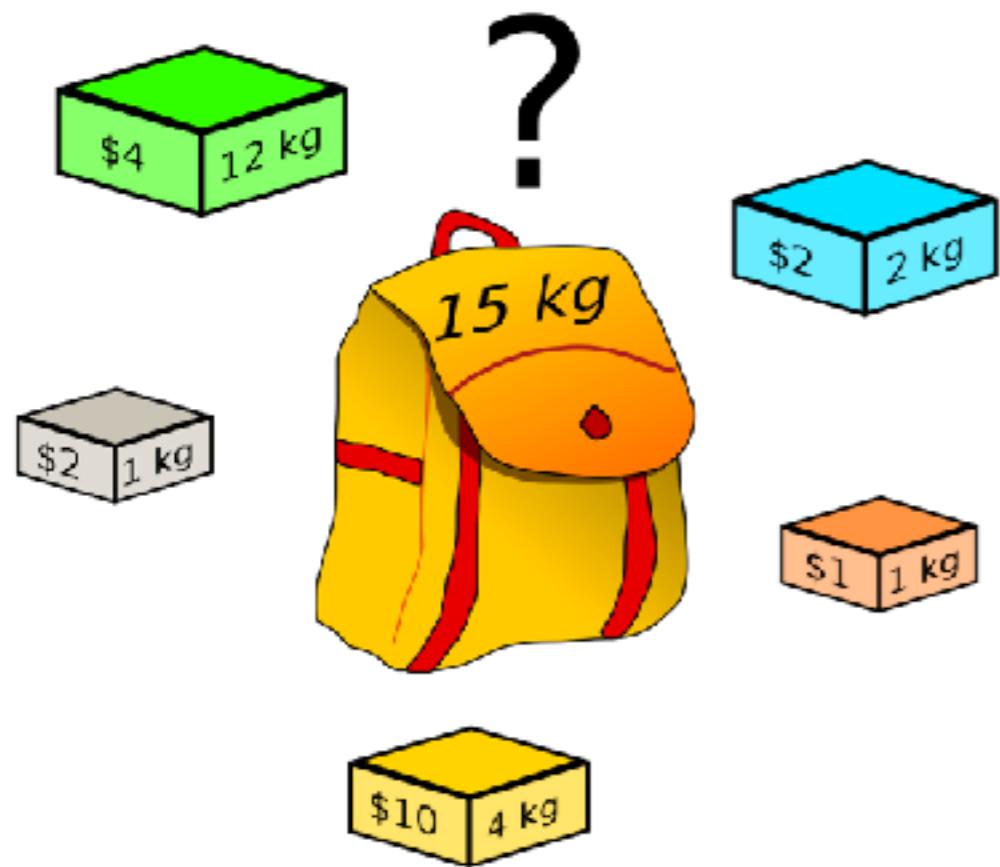
Given a set of items numbered from 1 up to  $n$ , each with a weight  $w$  and a  $v$  value, along with a maximum weight capacity  $W$ .

# Solució 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

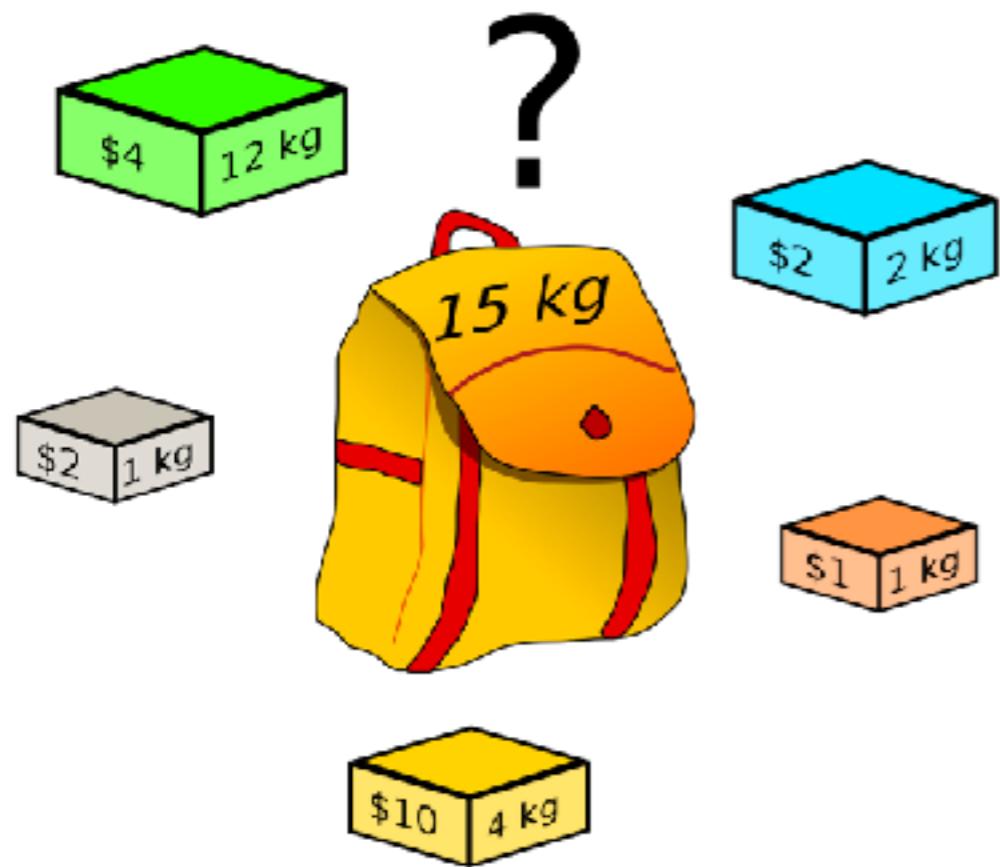
# Problema de la motxilla



**Greedy?**

Solució ràpida però no assegura torbar la solució òptima

# Problema de la motxilla



Solució amb DP?

# Problema de la motxila

- Com resoldre el problema amb programació dinàmica?
  - Hem d'identificar l'equació recursiva que associi el problema amb els subproblems

# Programació dinàmica

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

# Programació dinàmica

- Definició del sub-problema:
  - 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 questio és, podem definir la **solució final** ( $S_n$ ) en termes dels **sub-problemes** ( $S_k$ )?

# Programació dinàmica

Objectiu: Màxim benefici  
donat un  
**Pes màxim = 20 Kg**



Definim el subproblema:

Per  $S_4$ :

Pes Total = 14;

Benefici màxim: 20

$w_1=2$	$w_2=4$	$w_3=5$	$w_4=3$
$b_1=3$	$b_2=5$	$b_3=8$	$b_4=4$

Max weight:  $W = 20$

For  $S_4$ :

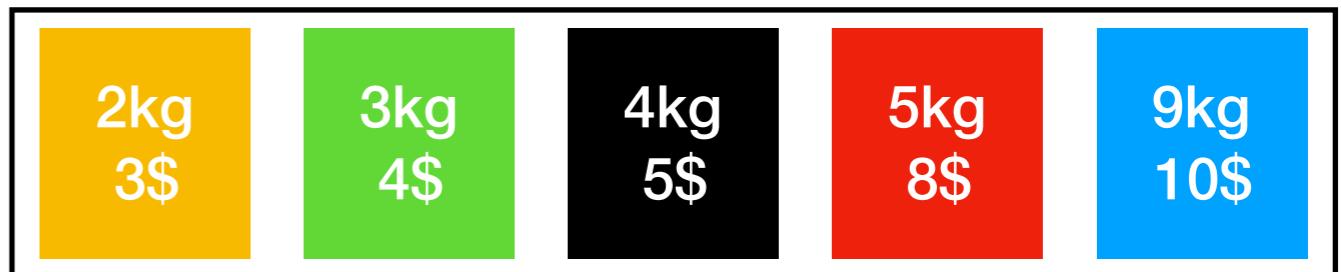
Total weight: 14;

Maximum benefit: 20

Item #	Weight $W_i$	Benefit $b_i$
1	2	3
$S_4$	2	4
3	4	5
4	5	8
5	9	10

# Programació dinàmica

Objectiu: Màxim benefici  
donat un  
**Pes màxim = 20 Kg**



Definim el subproblema:

**Per  $S_4$ :**

Pes Total = 14;

Benefici màxim: 20

**Per  $S_5$ :**

Pes Total = 20;

Benefici màxim: 26

$w_1=2$	$w_2=4$	$w_3=5$	$w_4=3$
$b_1=3$	$b_2=5$	$b_3=8$	$b_4=4$

Max weight:  $W = 20$

**For  $S_4$ :**

Total weight: 14;

Maximum benefit: 20

$w_1=2$	$w_2=4$	$w_3=5$	$w_4=3$
$b_1=3$	$b_2=5$	$b_3=8$	$b_4=4$

**For  $S_5$ :**

Total weight: 20

Maximum benefit: 26

Item #	Weight $W_i$	Benefit $b_i$
1	2	3
$S_4$	2	3
3	4	5
4	5	8
5	9	10

**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  $V[k, w]$ , per tal de trobar una solució òptima per a  $S_k = \{item\ etiquetats\ 1, 2, \dots, k\}$  dins una motxila de amb capacitat  $w$

# Programació dinàmica

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

# Fórmula recursiva

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

- 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.

```
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]

# Driver program to test above function
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(knapSack(W, wt, val, n))

# This code is contributed by Bhavya Jain
```

# Complexitat

```
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]
```

??

# Complexitat

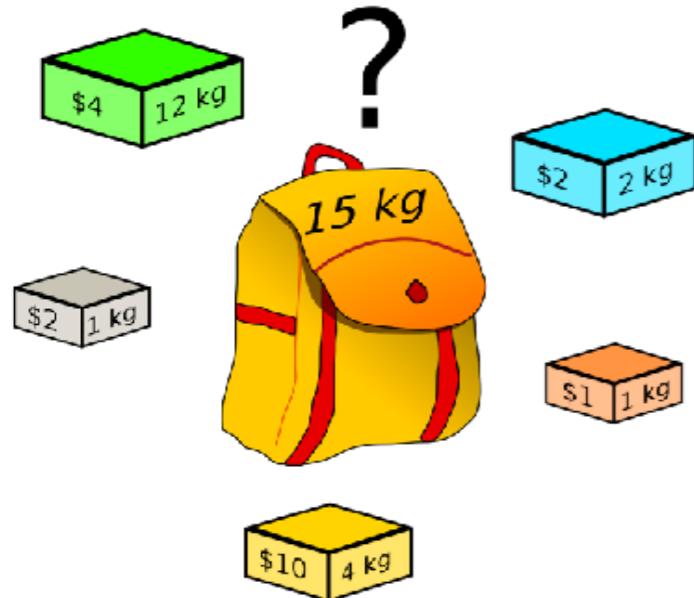
```
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]
```

$$O(nw)$$

# Problema de la motxilla

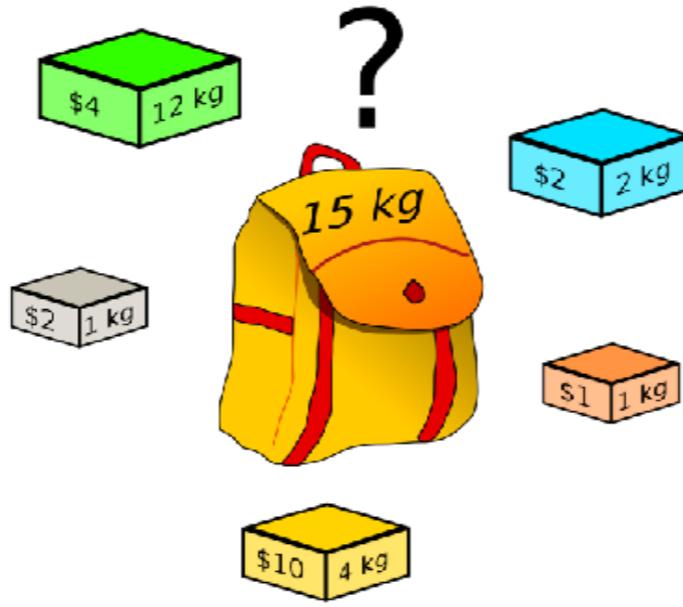


Solució per un màxim de 4 elements i una motxila de màxim pes = 6Kg

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																

# Problema de la motxilla

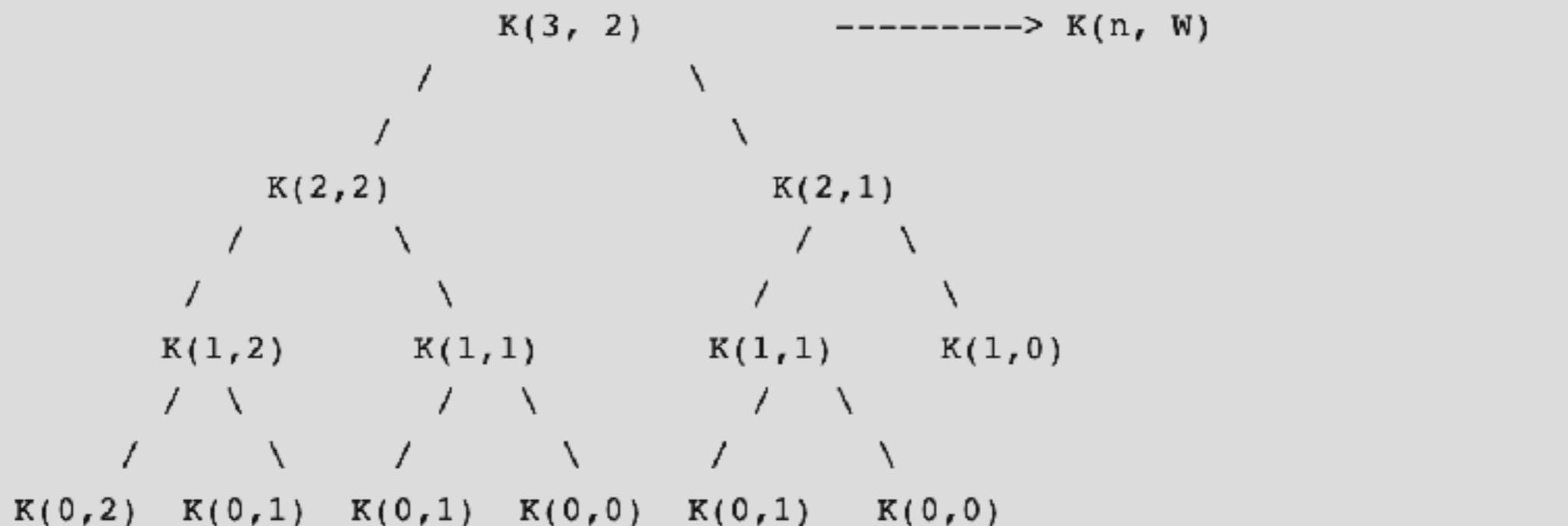


Article	Pes	Valor	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	1	0	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	
B	1	2	0	2(B)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	3(AB)	
C	2	2	0	2(B)	3(AB)	4(BC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)	5(ABC)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

# Problema de la motxila

In the following recursion tree,  $K()$  refers to knapSack(). The two parameters indicated in the following recursion tree are  $n$  and  $W$ . The recursion tree is for following sample inputs.

```
wt[] = {1, 1, 1}, W = 2, val[] = {10, 20, 30}
```

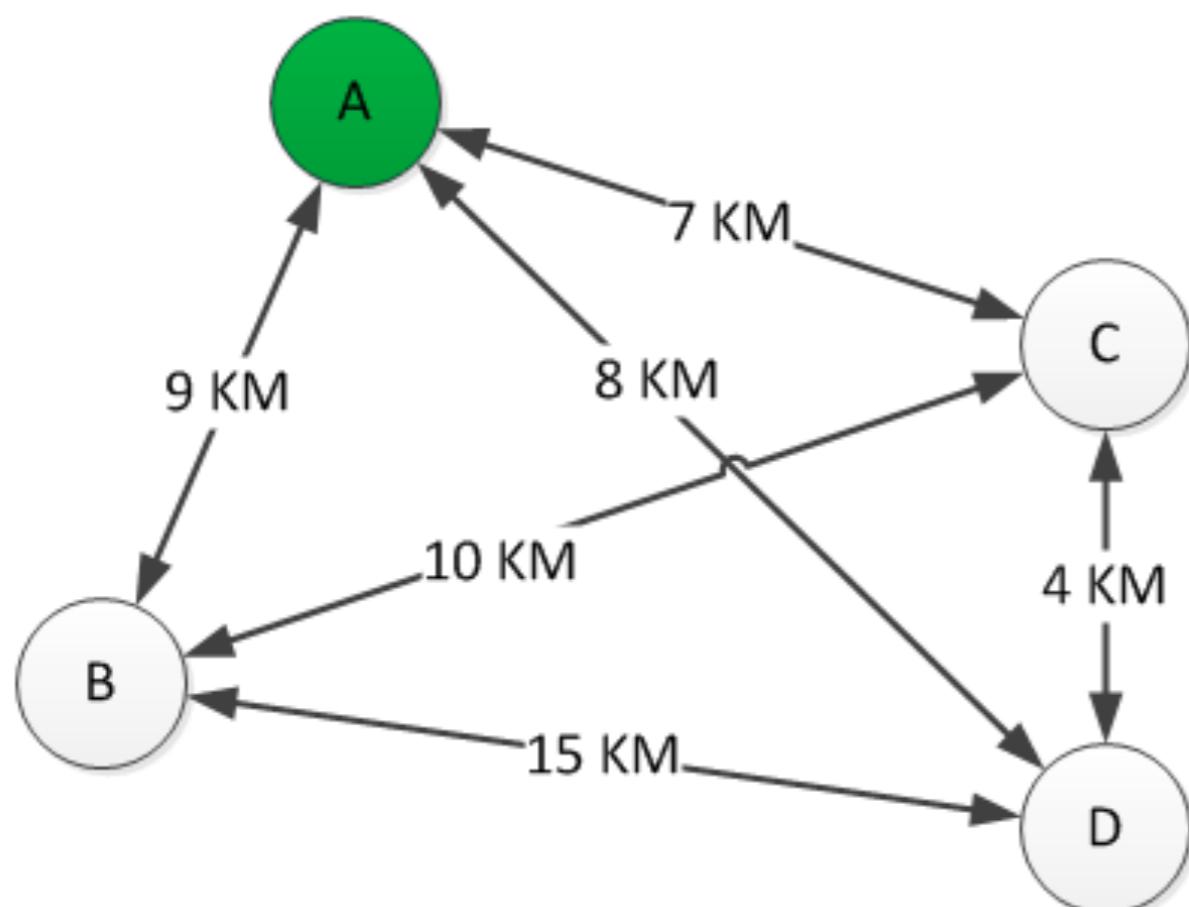


# Com trobar els elements de la motxila?

- Tota l'informació es troba a la taula.
- $V[n, W]$  és el benefici màxim dels elements que podem carregar dins la motxila.
- 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 motxila.
    - $i = i - 1$
    - $k = k - w_i$
  - Sino:
    - $i = i - 1$  # Assumim que l'element  $i$  no està dins la motxila

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

# Problema del viatjant de comerç



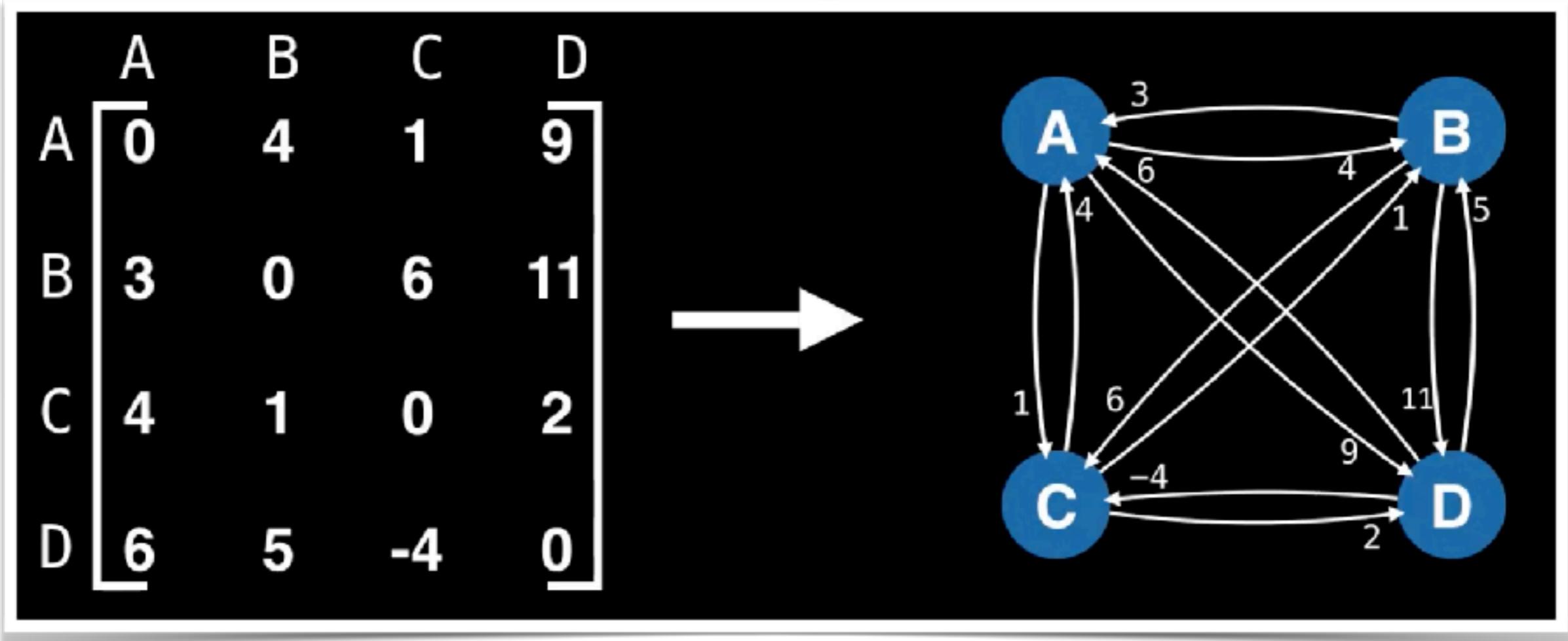
# Problema del viatjant de comerç

- 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**.

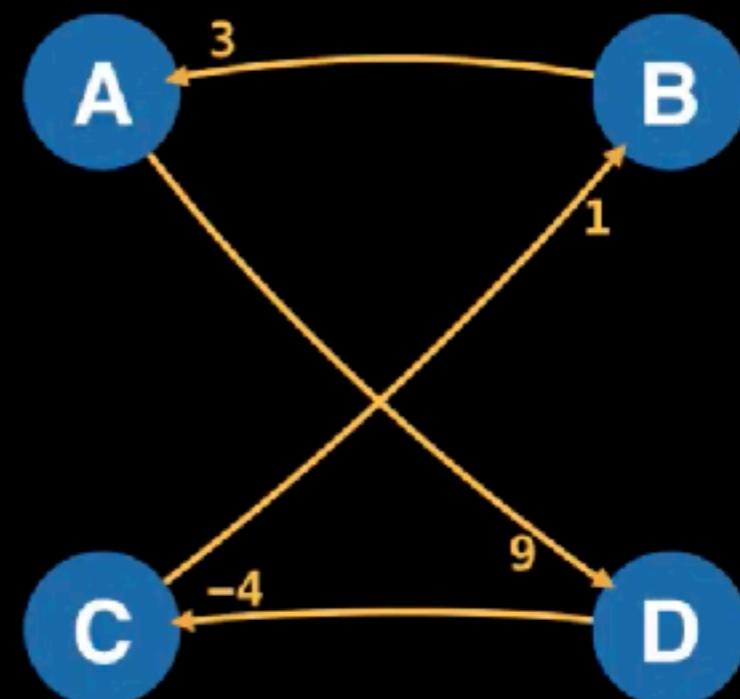
# Problema del viatjant de comerç

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

Donat un **graf complet** amb pesos quin és el **camí hamiltonià** de mínim cost



	A	B	C	D
A	0	4	1	9
B	3	0	6	11
C	4	1	0	2
D	6	5	-4	0



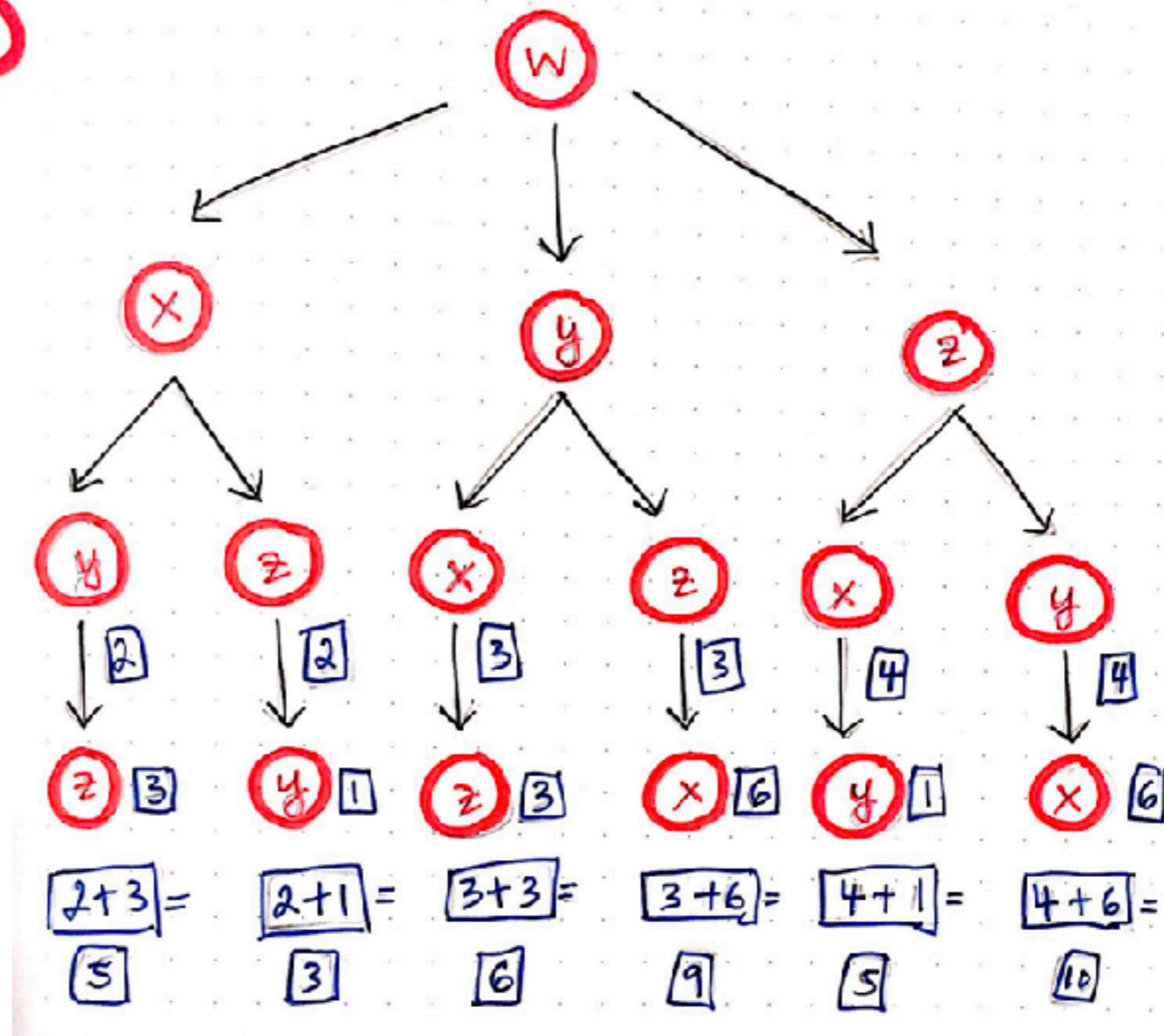
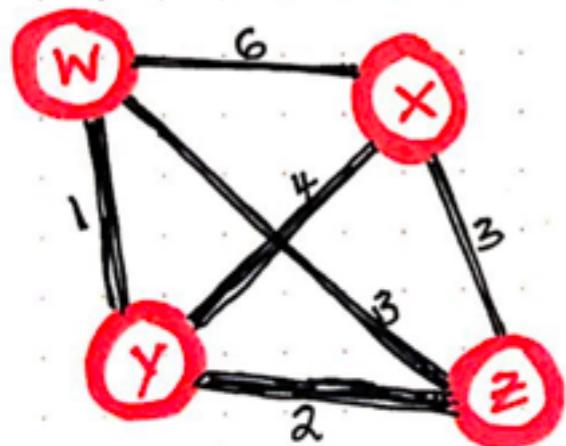
**Full tour:** A  $\rightarrow$  D  $\rightarrow$  C  $\rightarrow$  B  $\rightarrow$  A

**Tour cost:**  $9 + -4 + 1 + 3 = 9$

# Solució per força bruta?

- Buscar totes les possibles permutacions i mirar quina d'aquestes és la que té un cost menor. Això 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
					A B D C	15	C A D B 24
					A C B D	19	<b>C B A D 9</b>
					A C D B	11	C B D A 19
B	3	0	6	11	A D B C	24	C D A B 18
					<b>A D C B 9</b>		C D B A 11
					B A C D	11	D A B C 18
C	4	1	0	2	<b>B A D C 9</b>		D A C B 19
					B C A D	24	D B A C 11
D	6	5	-4	0	B C D A	18	D B C A 24
					B D A C	19	D C A B 15
					<b>B D C A 9</b>		<b>D C B A 9</b>



$$2+3=$$

$$\boxed{5}$$

$$2+1=$$

$$\boxed{3}$$

$$3+3=$$

$$\boxed{6}$$

$$3+6=$$

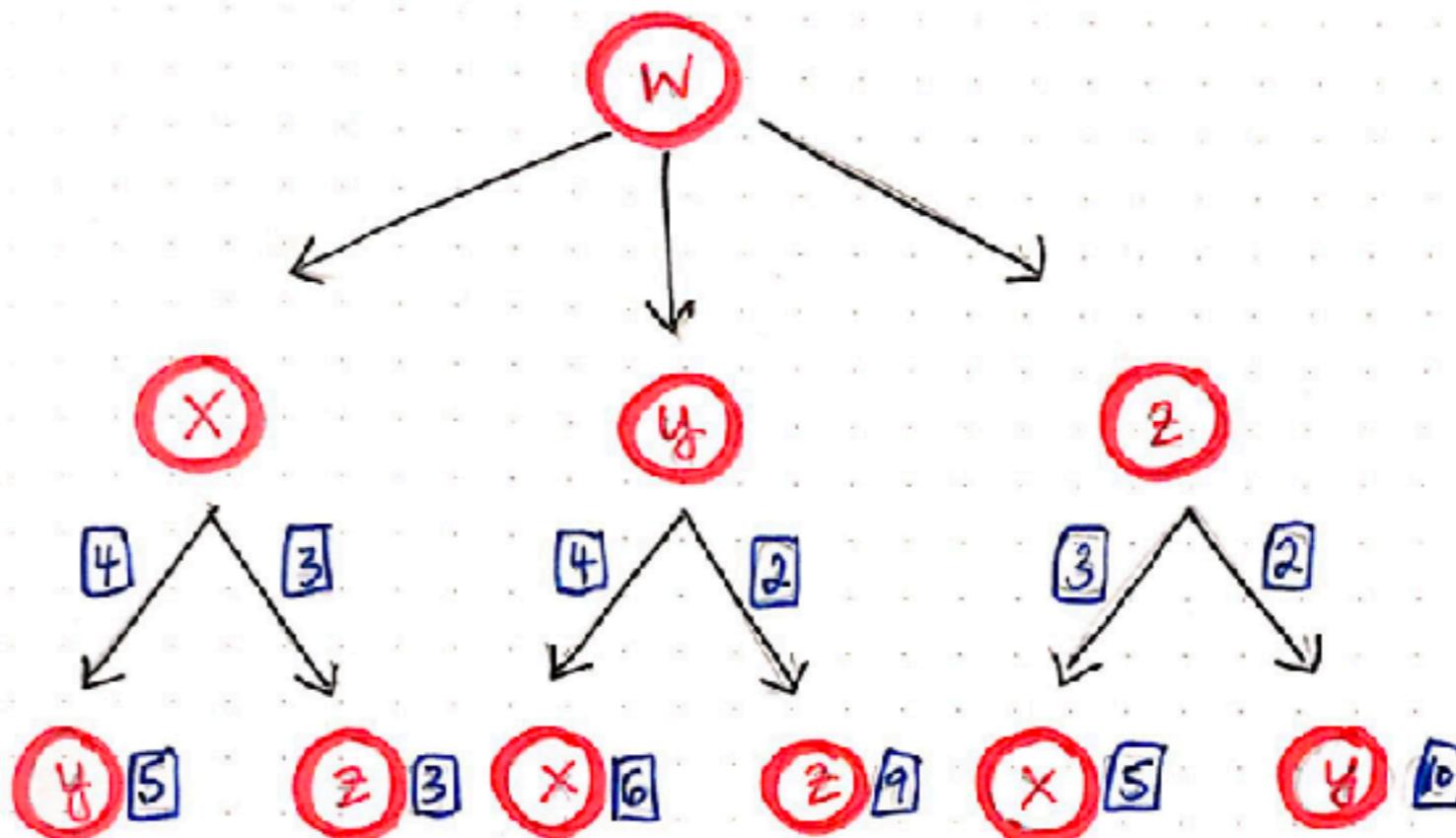
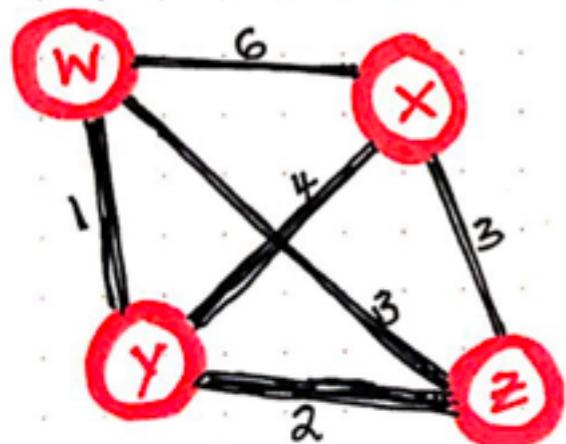
$$\boxed{9}$$

$$4+1=$$

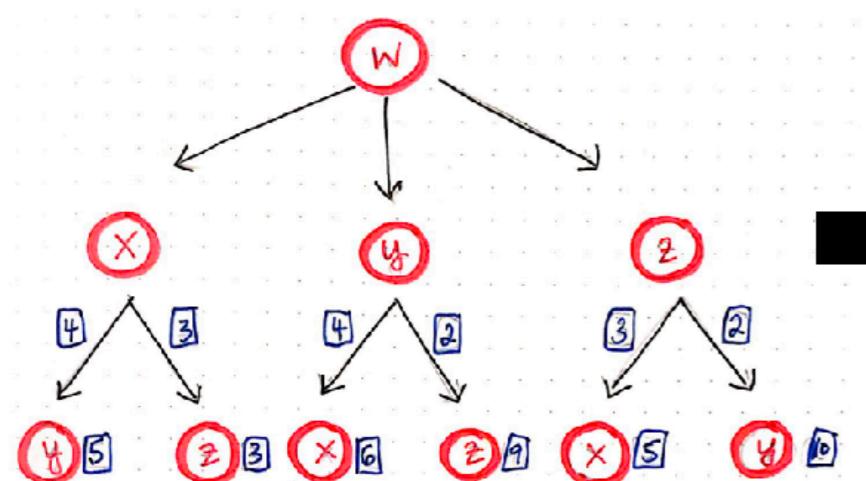
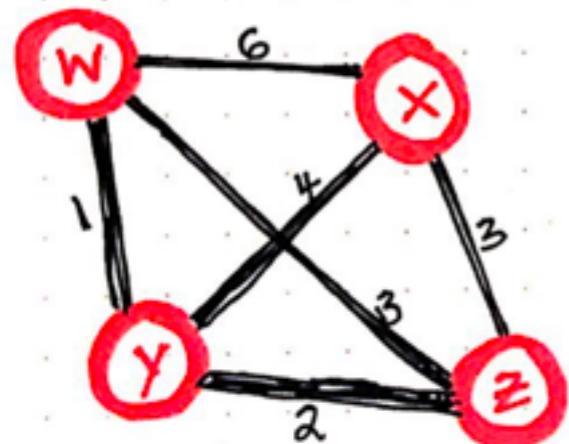
$$\boxed{5}$$

$$4+6=$$

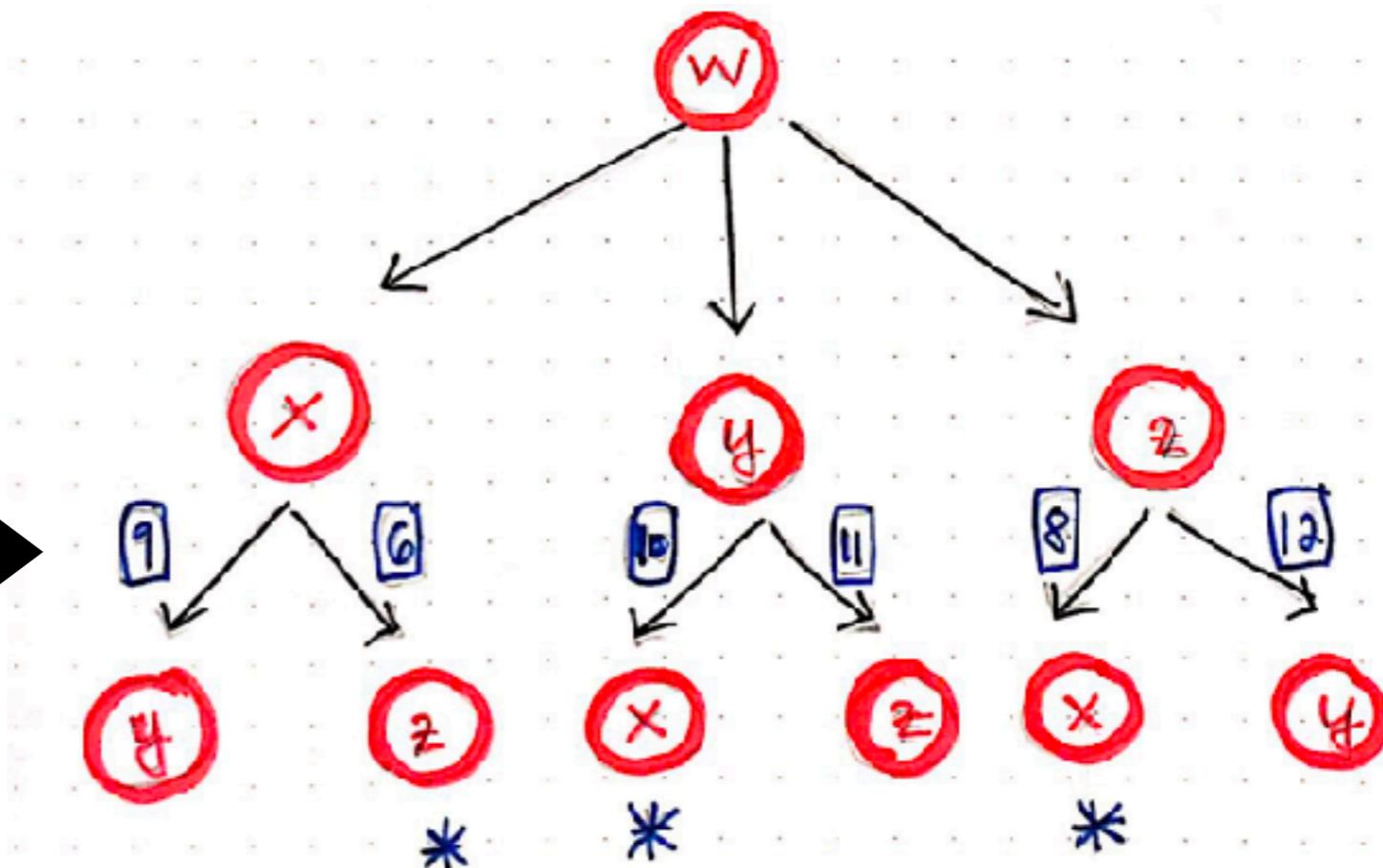
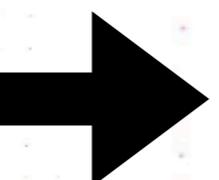
$$\boxed{10}$$



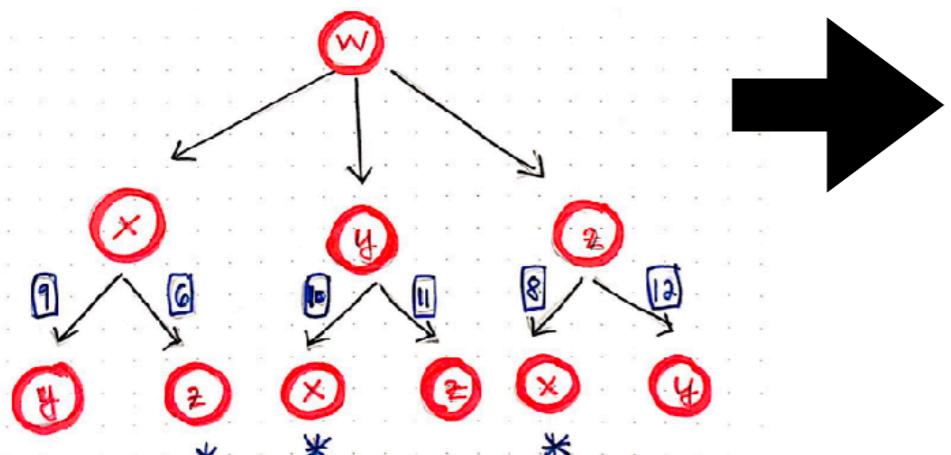
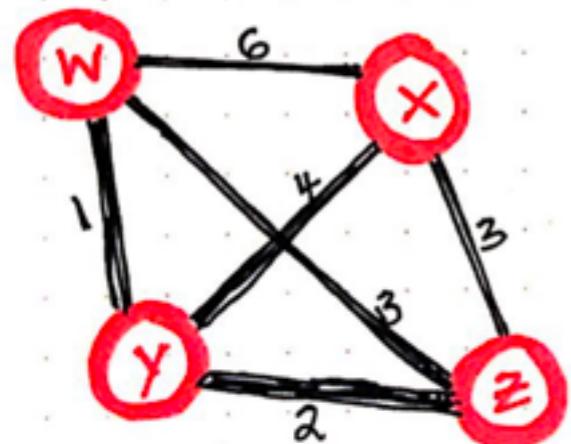
\*Remember: each time we sum & build up the distance between 2 nodes, we remove one level from this recursive tree.



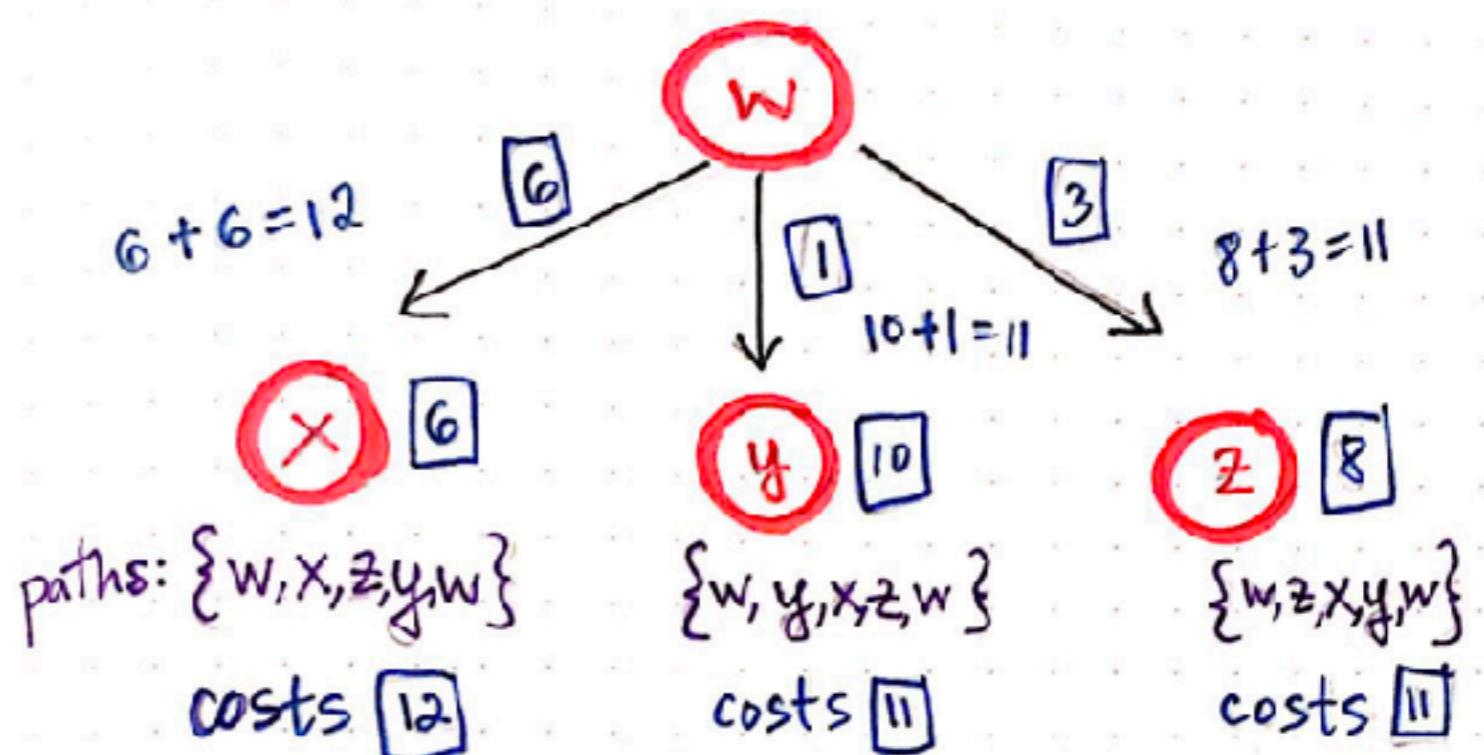
\*Remember: each time we sum + build up the distance between 2 nodes, we remove one level from this recursive tree.



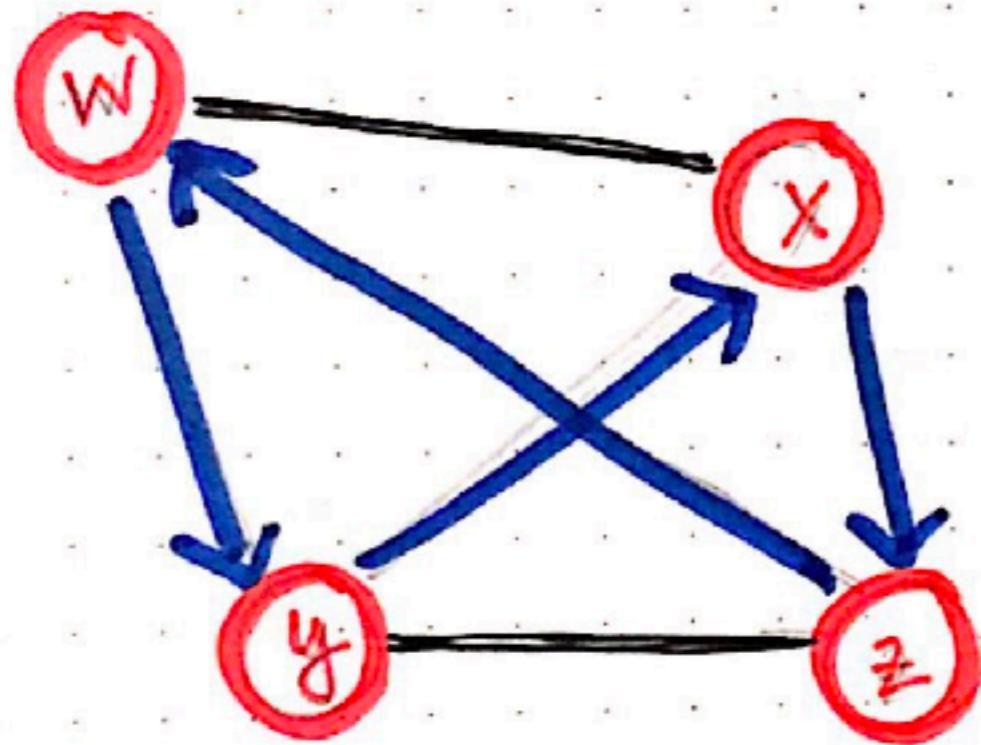
\*When we get to this point, we have two options of which possible permutation we want to sum up.  
 ⇒ We'll choose the shortest/lowest cost of each pair, which is marked here using a \*.



\*When we get to this point, we have two options of which possible permutation we want to sum up.  
 → We'll choose the shortest/lowest cost of each pair, which is marked here using a \*.



We've found 3 of the shortest paths! Since two of them both cost  $11$ , we can choose either as our solution.



We've solved for a Hamiltonian circuit by finding the shortest path from  $w$ , which costs us  $11$ .

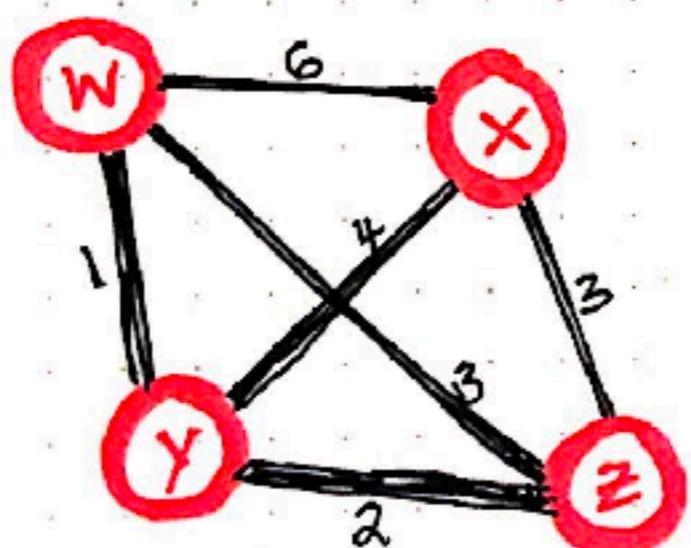
# Solució amb programació Dinàmica

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

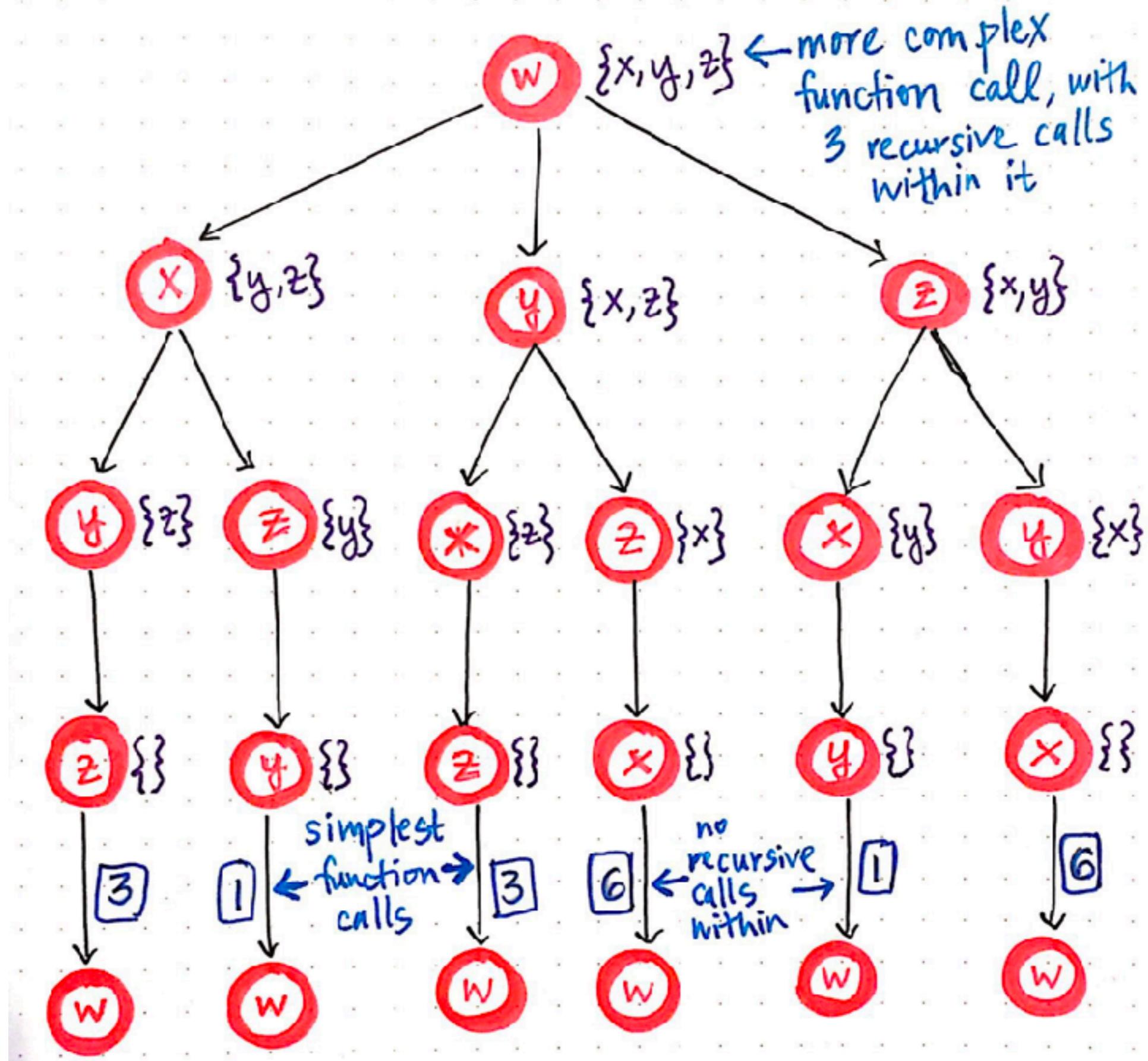
	$n=2$	$n=5$	$n=10$	$n=20$
<b>Força Bruta</b>	2	120	36.288.000	$1.4e^{18}$
<b>DP</b>	16	800	102.400	419.430.400

\* We first tried to solve TSP by using a brute-force method, using a top-down approach.

→ We started with one node, and a single function call, using it to generate more calls, recursively.



We can use an opposite approach to solve TSP for the exact same graph!



# Solució amb programació dinàmica

6  $\times$  {}

1 y {}

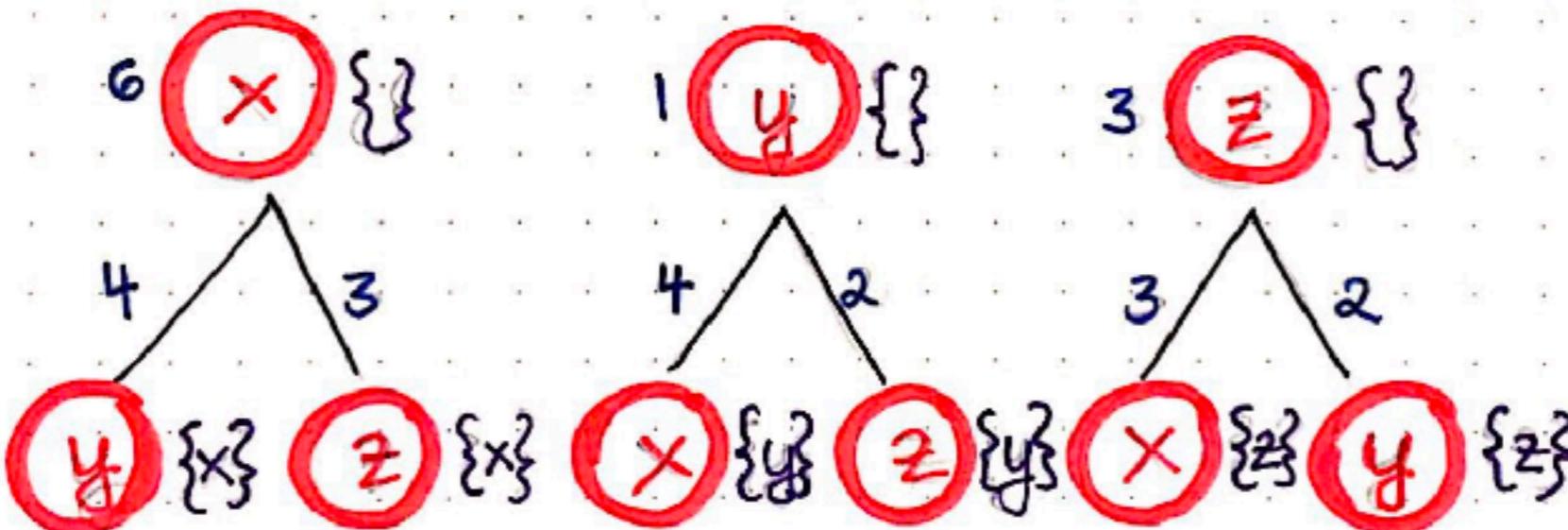
3 z {}

⇒ These are our three simplest function calls, each of which terminate at our start node, w.

\* The number next to each node tells us the cost to get to w.

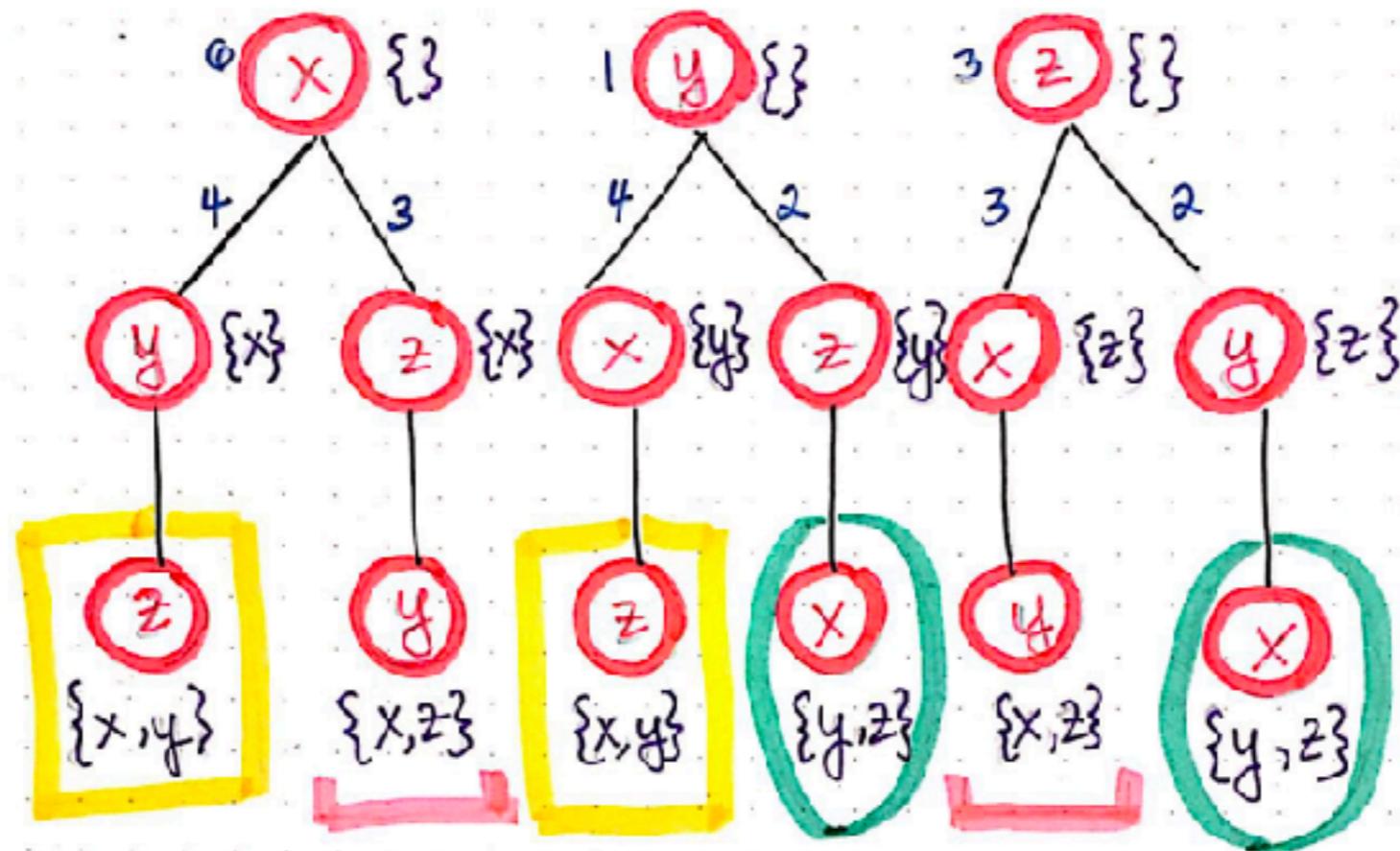
But how do we get to any of these three nodes? Which nodes lead to them?

# Solució amb programació dinàmica



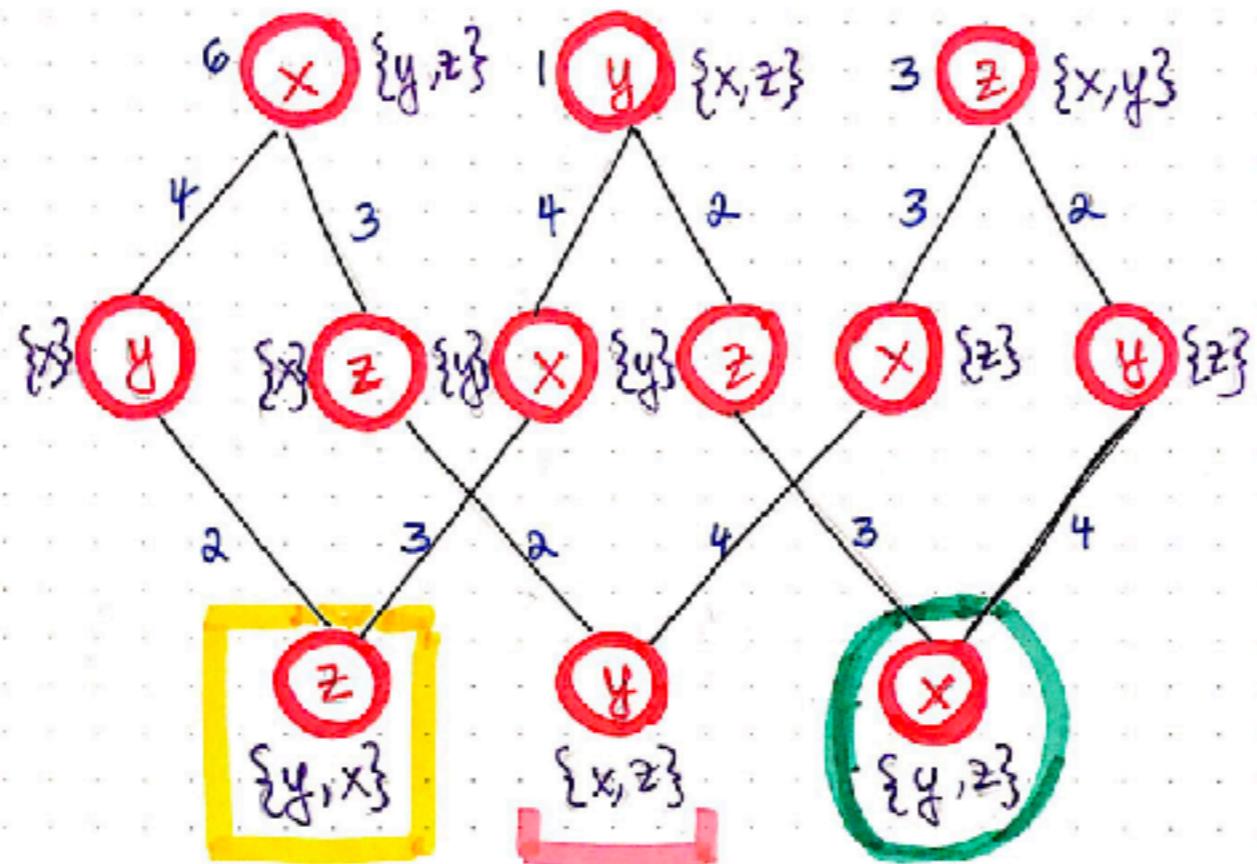
	w	x	y	z
w	0	6	1	3
x	6	0	4	3
y	1	4	0	2
z	3	3	2	0

Using our adjacency matrix, we can enumerate **NOT** the possible paths, but in this case, all the function calls that lead to one another.



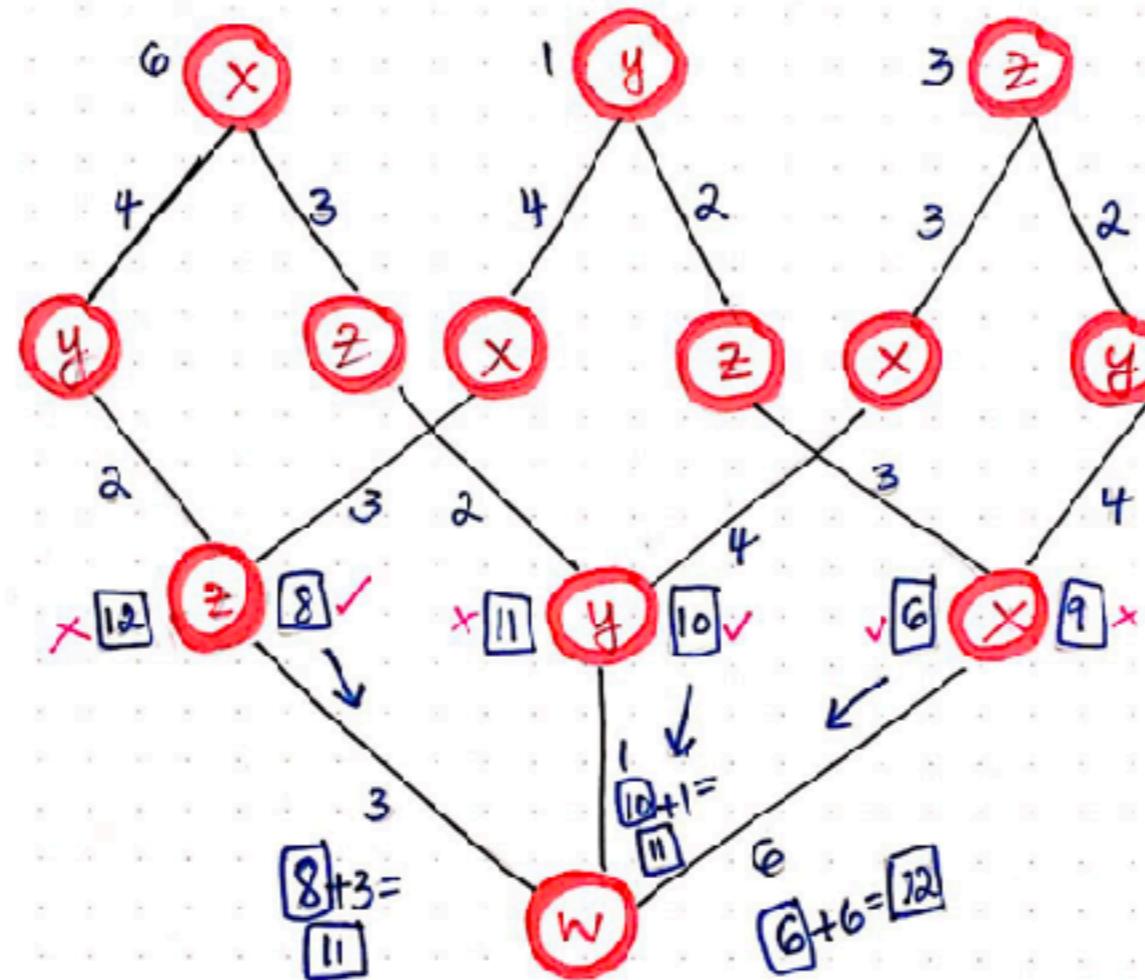
\* There are so many overlapping subproblems here!

→ This is a sign that we can use dynamic programming here to help us out.



→ We no longer need to generate an entire tree to figure out all the recursive function calls and determine all possible paths

\* We saw subproblems, cut out our repeated work.



⇒ We end up with the same results as our brute-force method!

⇒ We no longer have 6 recursive calls being made, and we're not generating a giant tree structure!

# 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

# Solució amb programació Dinàmica

- $D(V_i S)$  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:

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

$$g(i, S) = \text{Min}_{j \in S} \{L_{ij} + g(j, S - \{j\})\}$$

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

# Solució amb programació Dinàmica

```
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