UNIVERSITAT DE BARCELONA

Enumeratius

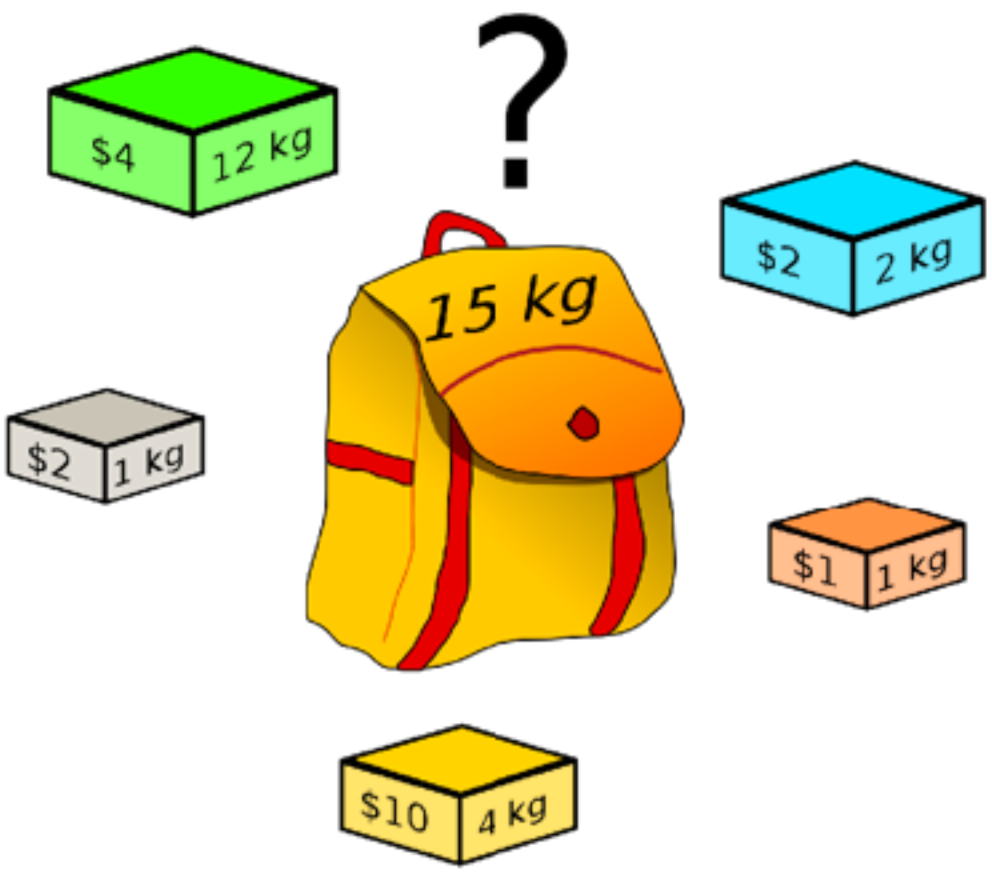
Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2019-2020

Enumeratius

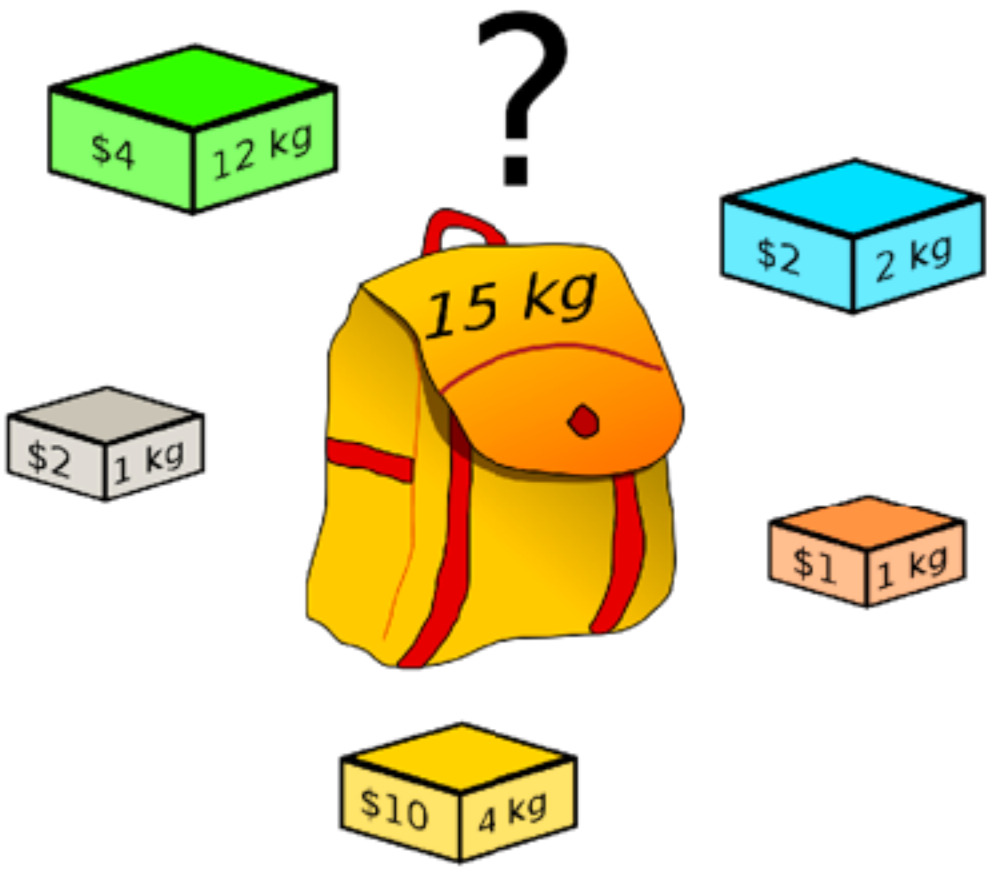
In computer science, an ***enumeration algorithm*** *is an algorithm that enumerates the answers to a computational problem. Formally, such an algorithm applies to problems that take an input and produce a list of solutions, similarly to function problems. For each input, the enumeration algorithm must produce the list of all solutions, without duplicates, and then halt*

Problema de la motxilla



Quines solucions hem vist?

Problema de la motxilla



Quines solucions hem vist?

Força bruta

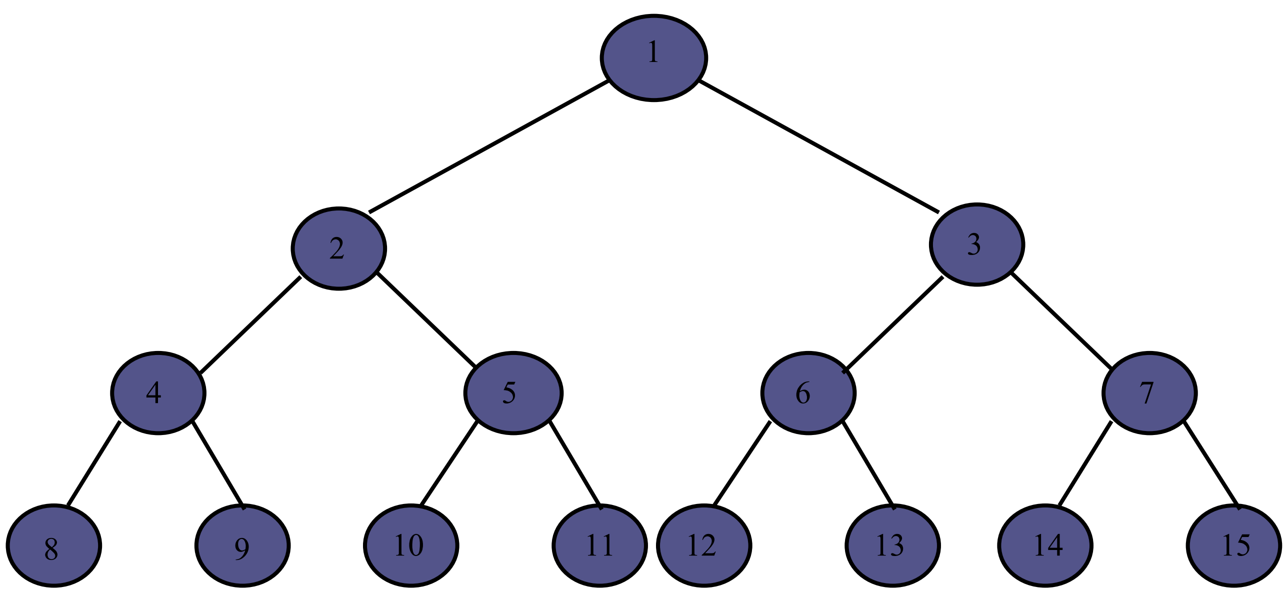
Greedy

Programació dinàmica

Enumeratius

- Recorregut vs. Cerca
- Backtracking
- Raminificació i poda

Cerca - Algorítmes amb arbres



Arbre Binari

Conjunt de nodes i arestes, on el node superior s'anomena **root** i els nodes externs s'anomenen **fulles**.

En el cas dels **arbres binaris** cada node te com a **màxim 2 arestes sortints**.

Un arbre binari, al nivell i té com a màxim 2i nodes (suposant el root i=0)

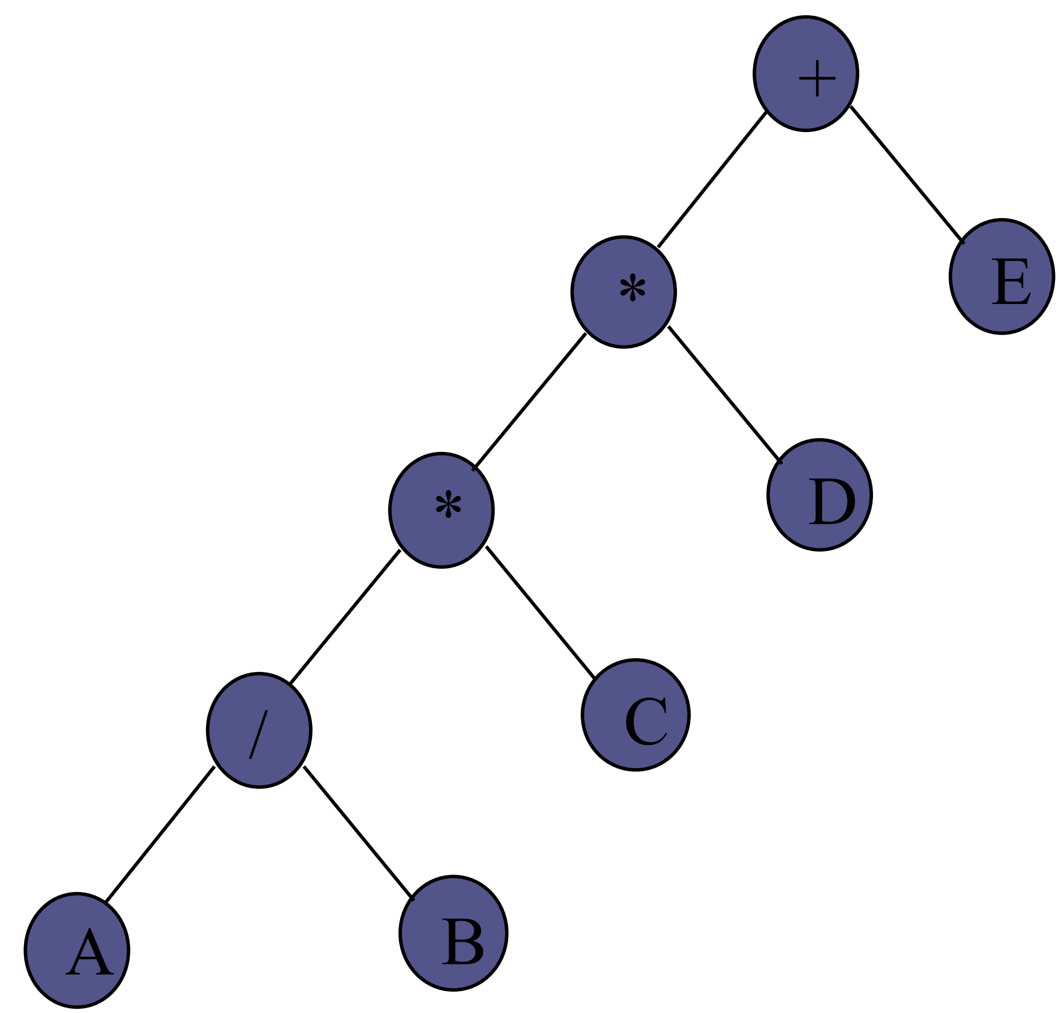
Cerca - Algorítmes amb arbres

- Com podem recórrer un arbre binari?
  - Si denominem amb
    - L: moviment a l'esquerre
    - V: "visitar" el node
    - R: moviment a la dreta
  - Tenim sis combinacions possibles de recorregut:
    - LVR, LRV, VLR, VRL, RVL, RLV
  - Si optem per realitzar primer el moviment a l'esquerra, tenim les tres primeres possibilitats
    - LVR denominarem inordre
    - LRV denominarem postordre, i
    - VLR denominarem preordre

Cerca - recorreguts

- Els recorreguts es corresponen amb les formes infixa, postfixa i prefixa d'escriure una expressió aritmètica en un arbre binari.

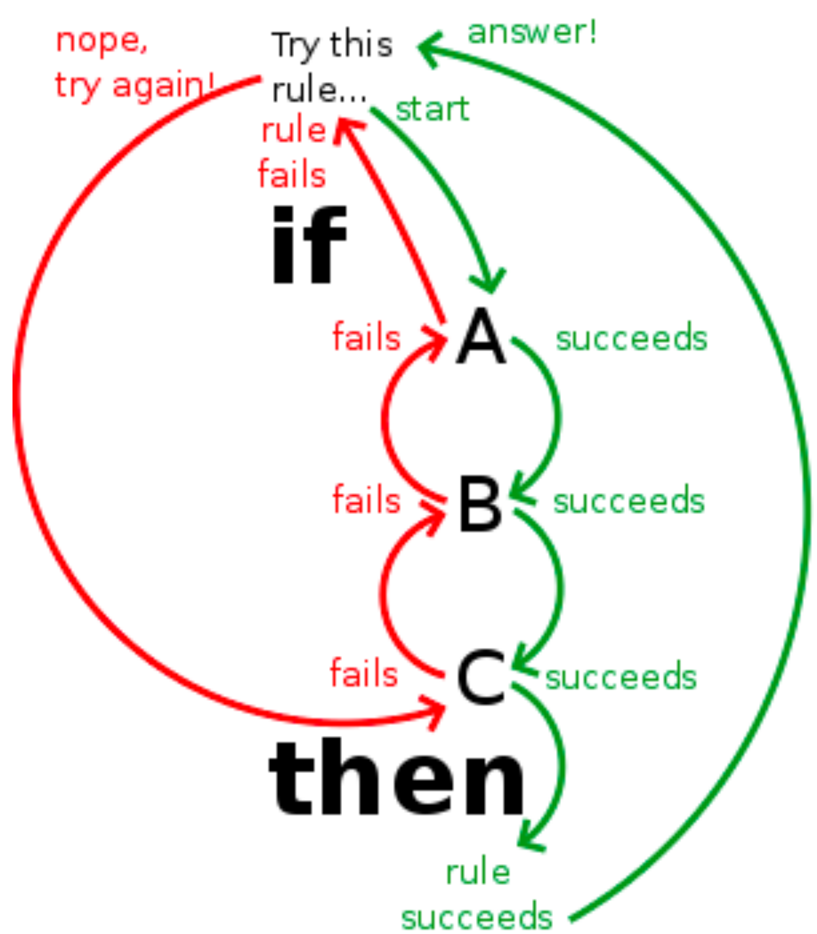
- Donat l'arbre binari, els diferents recorreguts porten a les següents formes d'escriure l'expressió:
  - Inordre LVR : A/B*C*D+E
  - Preordre VLR : +**/ABCDE
  - Postordre LRV : AB/ C*D*E+



Backtracking

Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, that incrementally builds candidates to the solutions, and abandons each partial candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

Backtracking

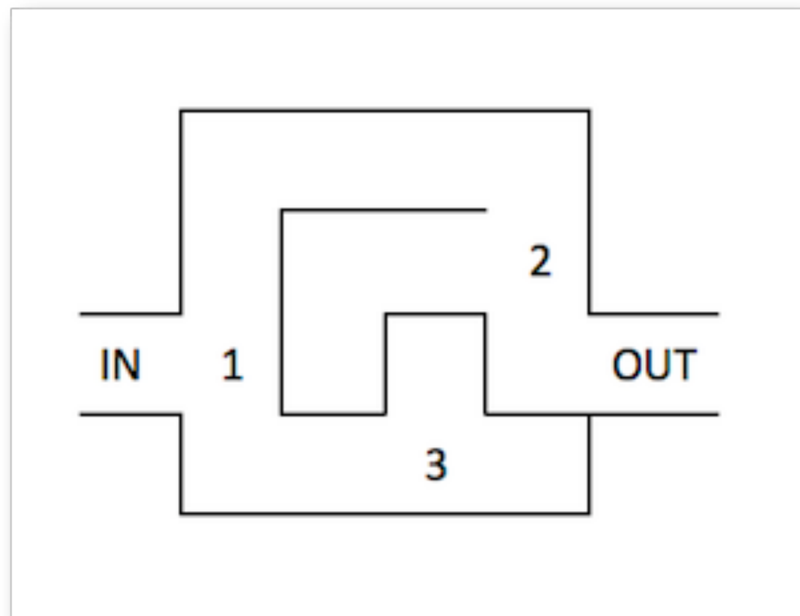


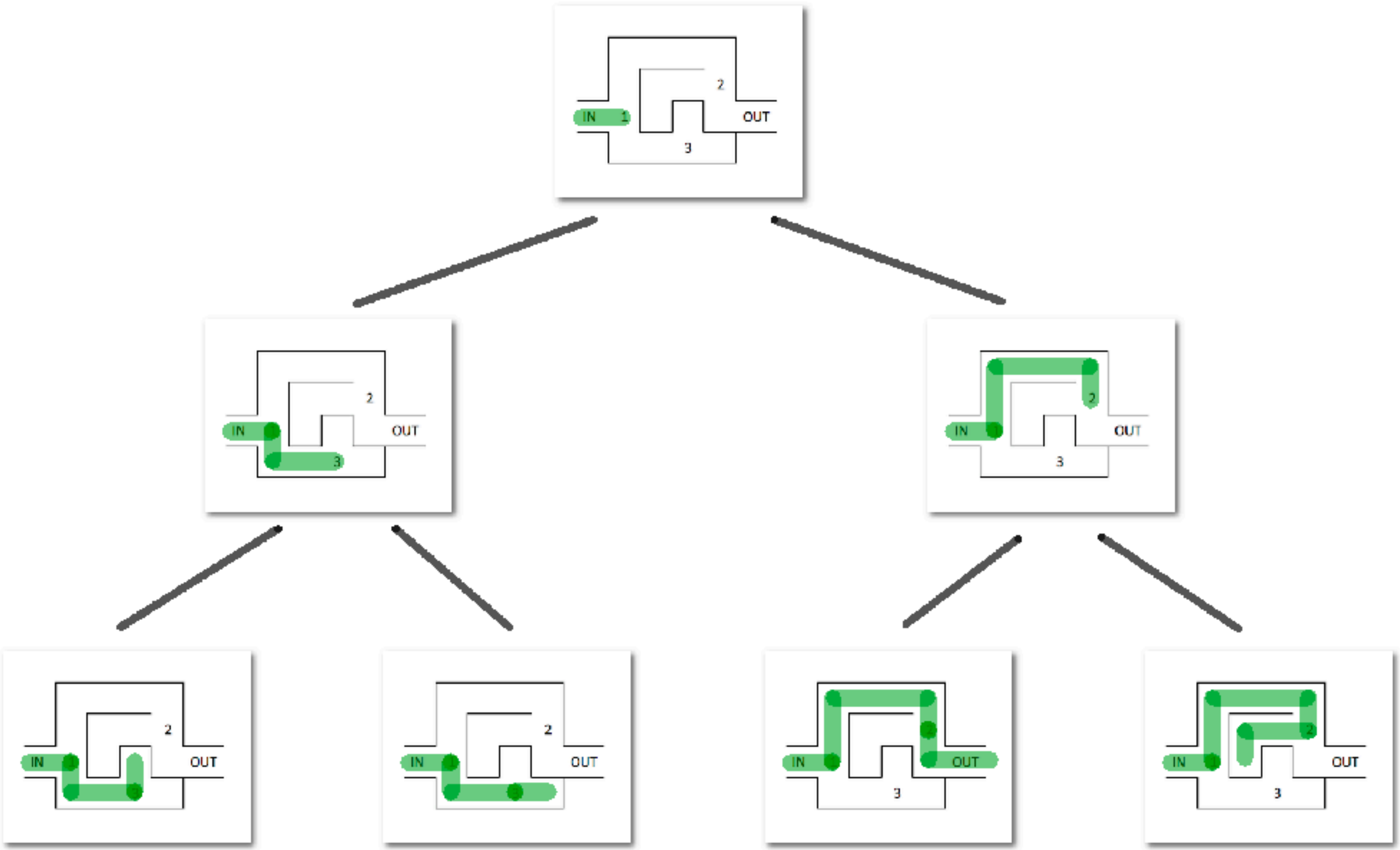
Backtracking

- El **backtracking** introdueix uns “criteris” per **reduir** la **complexitat de la cerca recursiva**.
- Aplicacions:
    - → Comprovar si un problema té solució
    - → Buscar múltiples solucions o una de totes les possibles

Veiem un exemple

Trobar la sortida del laberint.

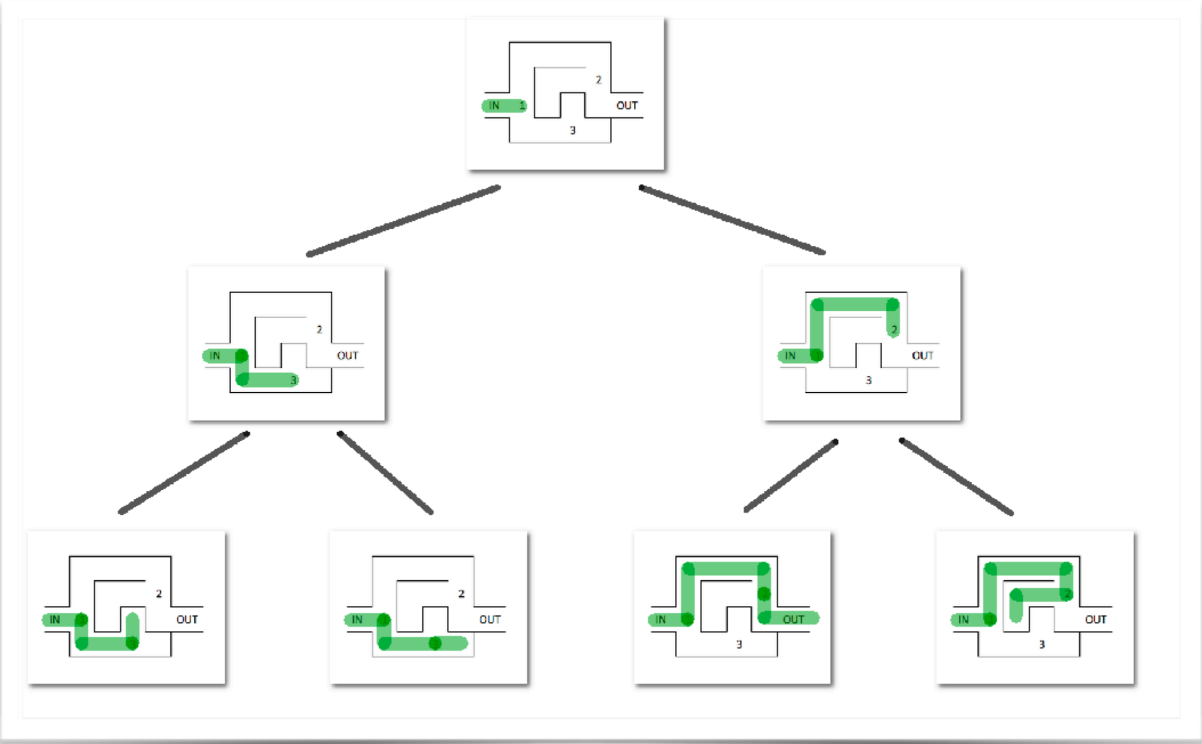




Tots els camins possibles

```
function backtrack(junction):

  if is_exit:
    return true


  for each direction of junction:
    if backtrack(next_junction):
      return true

  return false
```

```
function backtrack(junction):

  if is_exit:
    return true


  for each direction of junction:
    if backtrack(next_junction):
      return true

  return false
```



If we apply this pseudo code to the maze we saw above, we'll see these calls:

```
- at junction 1 chooses down        (possible values: [down, up])
    - at junction 3 chooses right   (possible values: [right, up])
         no junctions/exit          (return false)
    - at junction 3 chooses up      (possible values: [right, up])
         no junctions/exit          (return false)
- at junction 1 chooses up          (possible values: [down, up])
    - at junction 2 chooses down    (possible values: [down, left])
         the exit was found!        (return true)
```

*The idea is that we can **build a solution step by step using recursion**; if during the process we realise that is **not** going to be a valid **solution**, then we stop computing that solution and **we return back** to the step before (**backtrack**).*

Backtracking

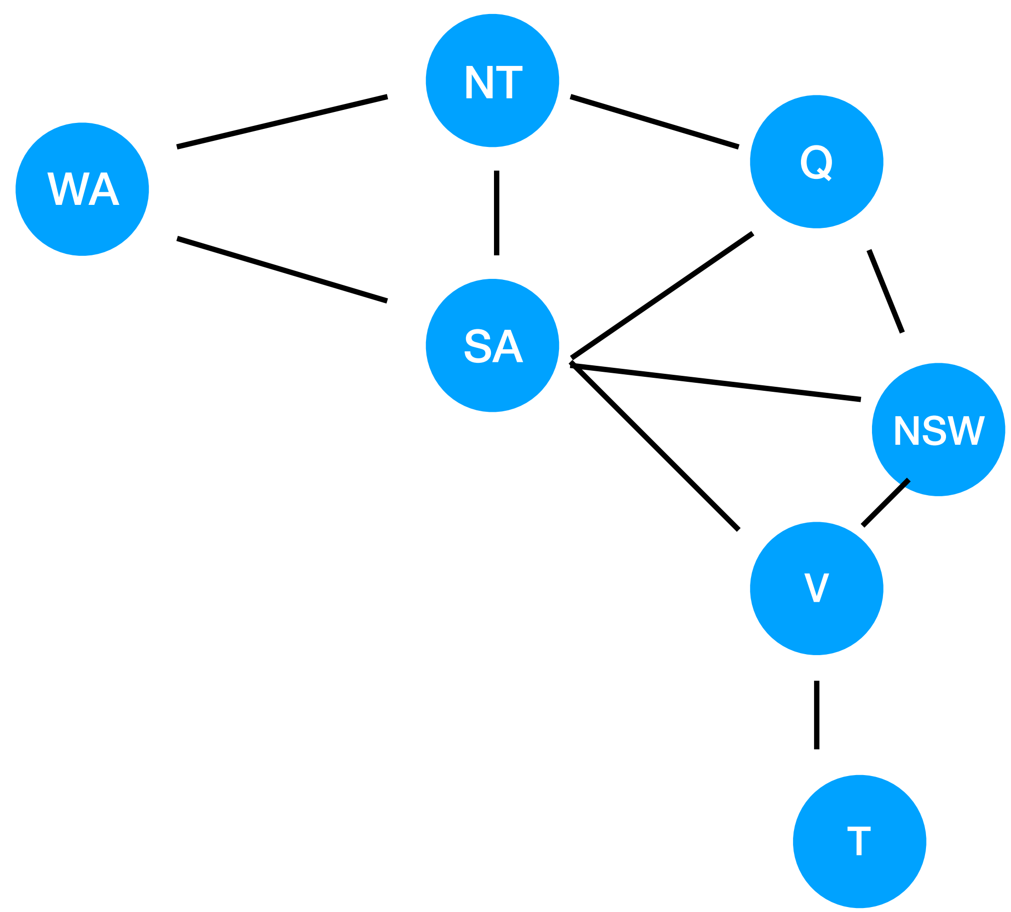
- Els esquemes de backtracking que veurem són directament aplicables a qualsevol tipus de graf (en molts exemples suposarem que són arbres)

```
Bactracking Enum(X,num)
   variables L: ListaComponentes
   inicio
        si EsSolución (X) entonces num = num+1
            mostrarSolución (X)
        sino
            L  = Candidatos (X)
            mientras ¬Vacía (L) hacer
                X[i + 1] =  Cabeza (L); L =  Resto (L)
                BacktrackingEnum (X, num)
```

Exercici: Pinta el mapa

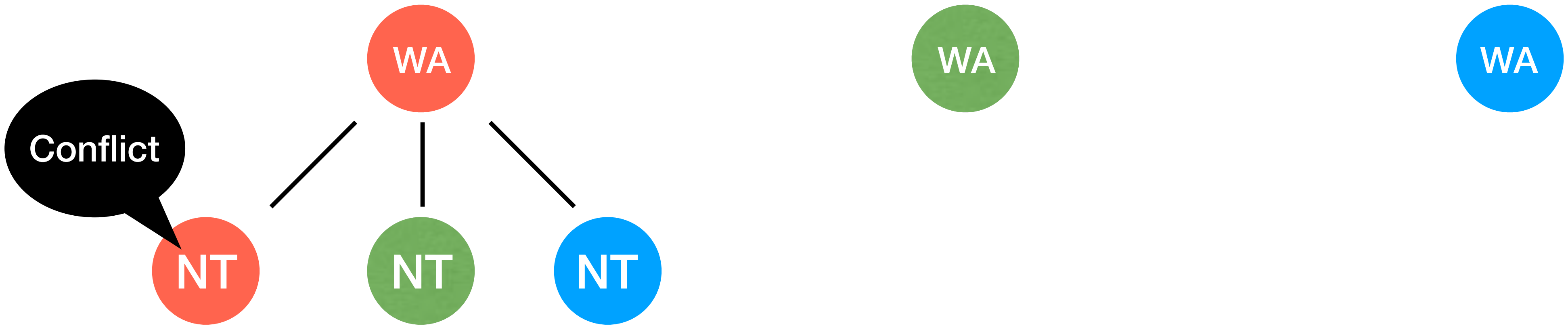
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



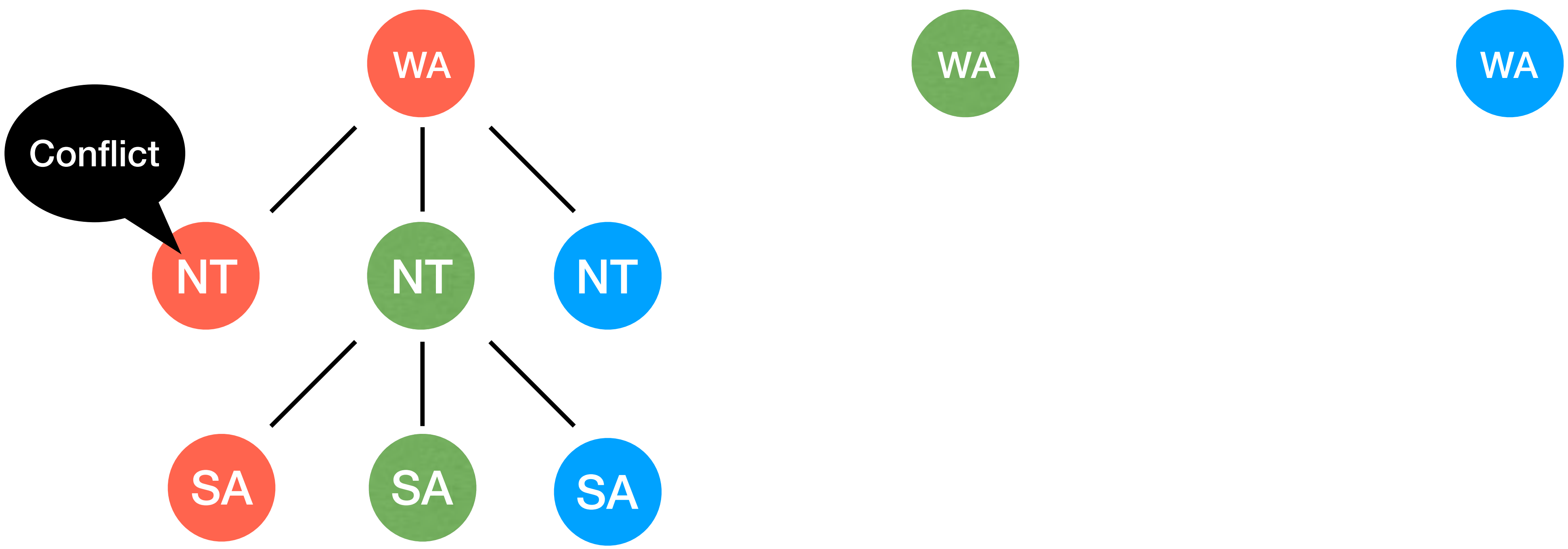
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



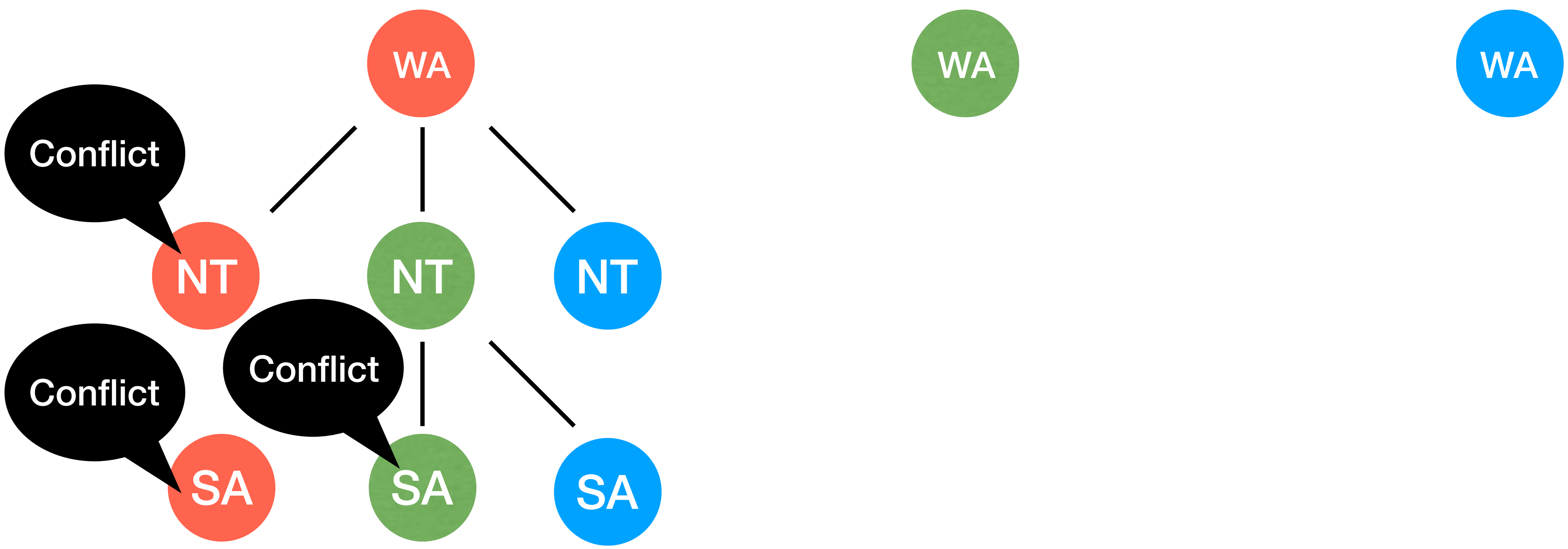
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



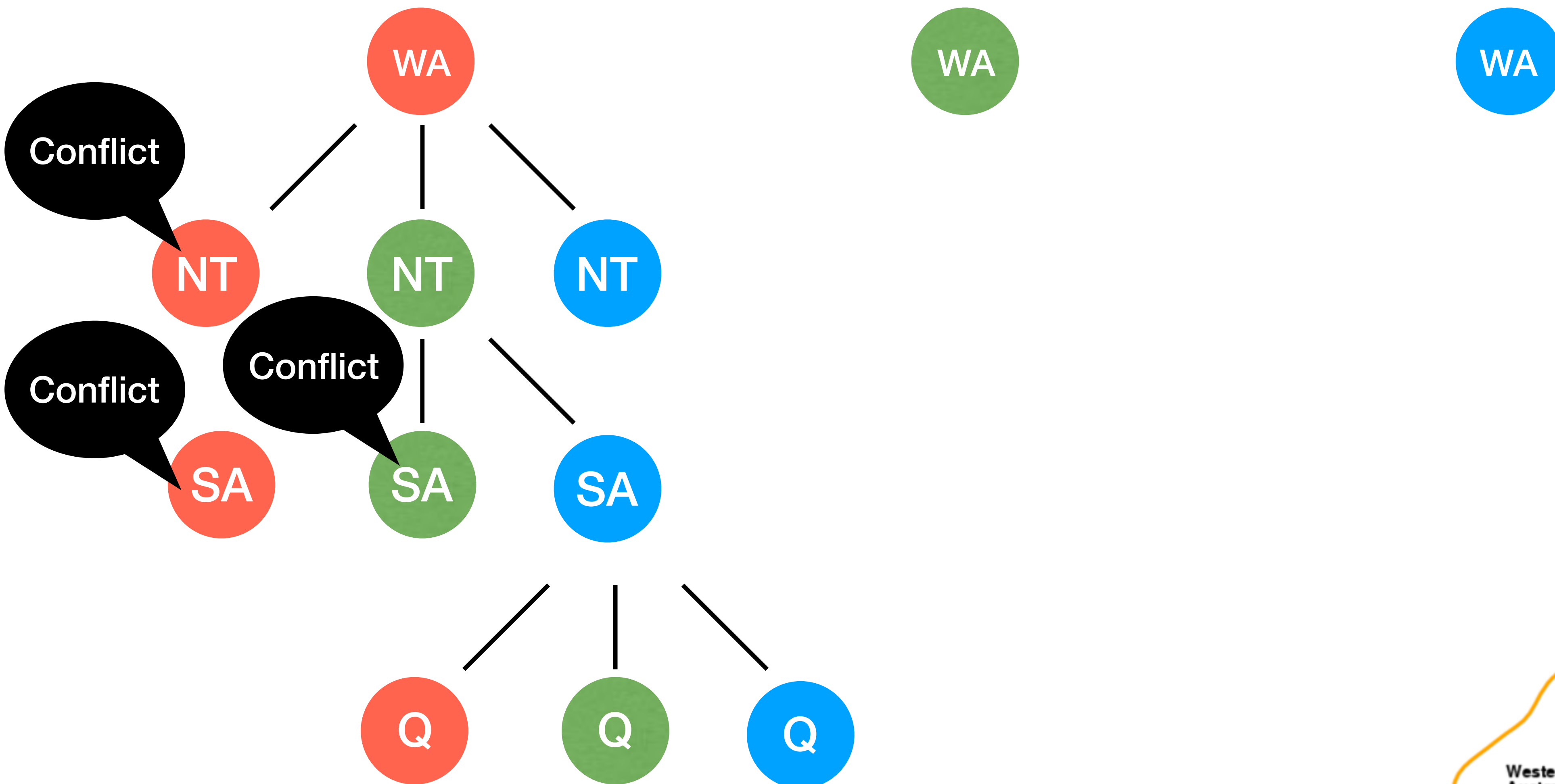
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



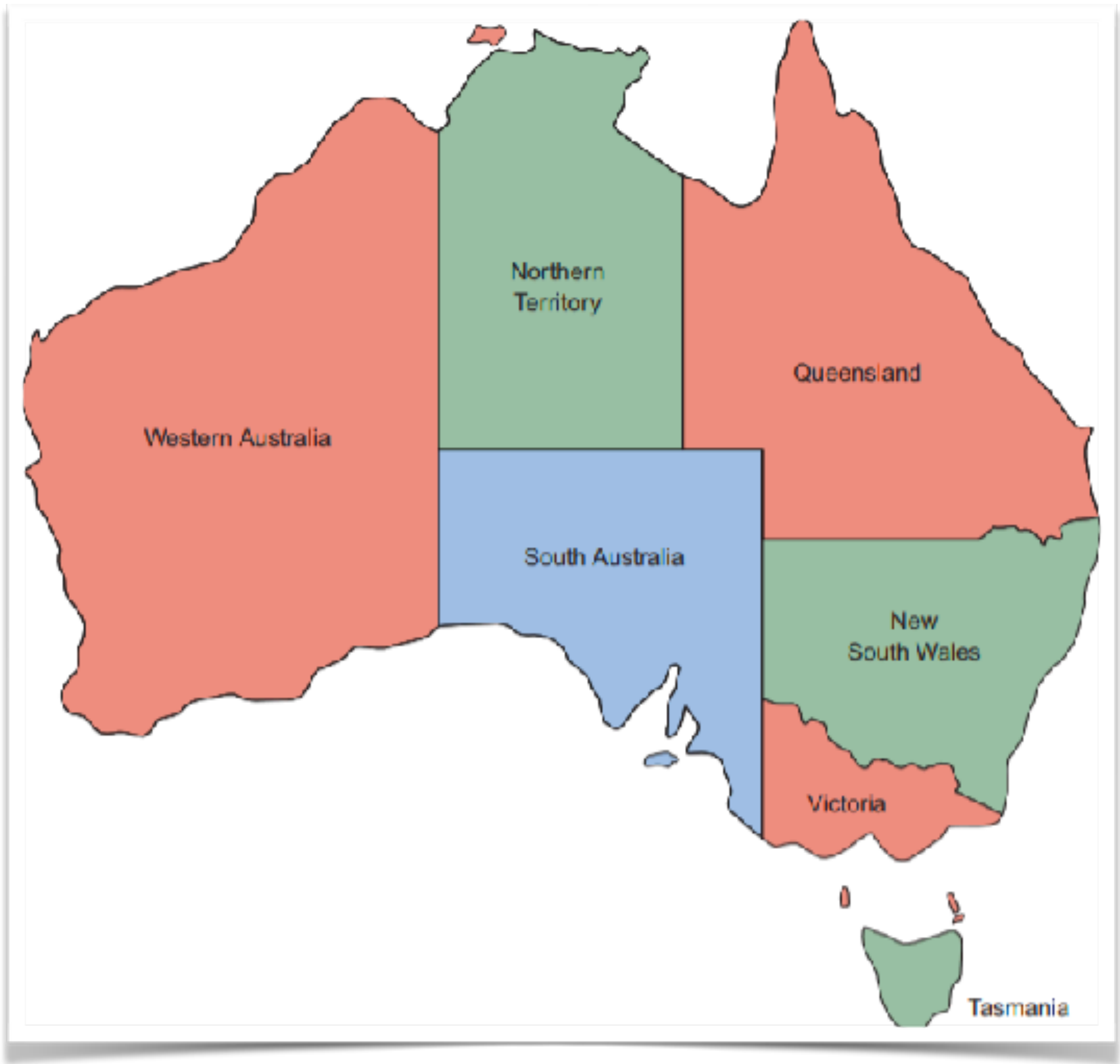
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



General Backtraking algorithm

```
algorithm backtrack():
  if (solution == True)
    return True
  for each possible moves
    if(this move is valid)
      select this move and place
      call backtrack()
      unplace that selected move
      increment the given choice in the for loop
    else
      return False
```

General Backtraking algorithm

```
algorithm backtrack():
    if (solution == True)
        return True
    for each possible moves
        if(this move is valid)
            select this move and place
            call backtrack()
            unplace that selected move
            increment the given choice in the for loop
        else
            return False
```

Solution found

Keep exploring

Don't explore anymore!
no solution in this path

Exercici: Pinta el mapa

PENSEM UNA SOLUCIÓ

Exercici: Pinta el mapa

```
If all vertexes has a color assigned,
    print vertex assigned colors
Else
    for all possible colors,
       assign a color to the vertex
       If color assignment is possible,
         recursivelty assign colors to next vertices
       If color assignment is not possible,
         de-assign color
         return False
```

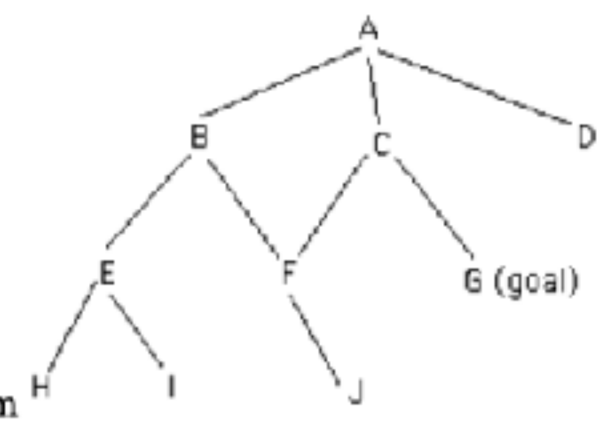
Backtracking

- Trobar ruta òptima entre node *i* i *j*

```
ruta_optima(i, j, ruta, ruta_optima)
{ Calcula la ruta óptima entre i y j y la concatena en la lista ruta_optima. }
    si i = j entonces
        medir_ruta(ruta)
        si es mejor que ruta_optima entonces ruta_optima ← ruta
    sino
        marcar i como visitado
        ∀ k : k no visitado, k adyacente a i :
        ⌈   añadir k al final de ruta
        |   ruta_optima(k, j, ruta, ruta_optima)
        ⌊   quitar k del final de ruta
        marcar i como no visitado
    fin
```

Backtracking

- The backtrack algorithm uses three lists plus one variable
  - **SL**, the state list, lists the states in the current path being tried. If a goal is found, SL contains the ordered list of states on the solution path.
  - **NSL**, the new state list, contains nodes awaiting evaluation -- i.e., nodes whose descendants have not yet been generated and searched.
  - **DE**, dead ends, lists states whose descendants have failed to contain a goal node. If these states are encountered again, they will be immediately eliminated from consideration.
  - **CS**, the current state.

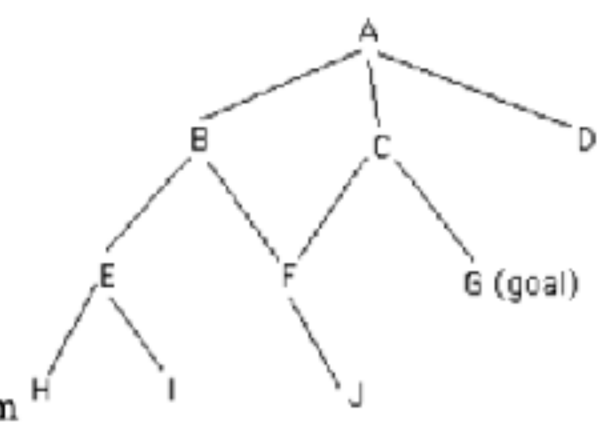


A Trace of Backtrack Algorithm

AFTER ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]

Backtracking

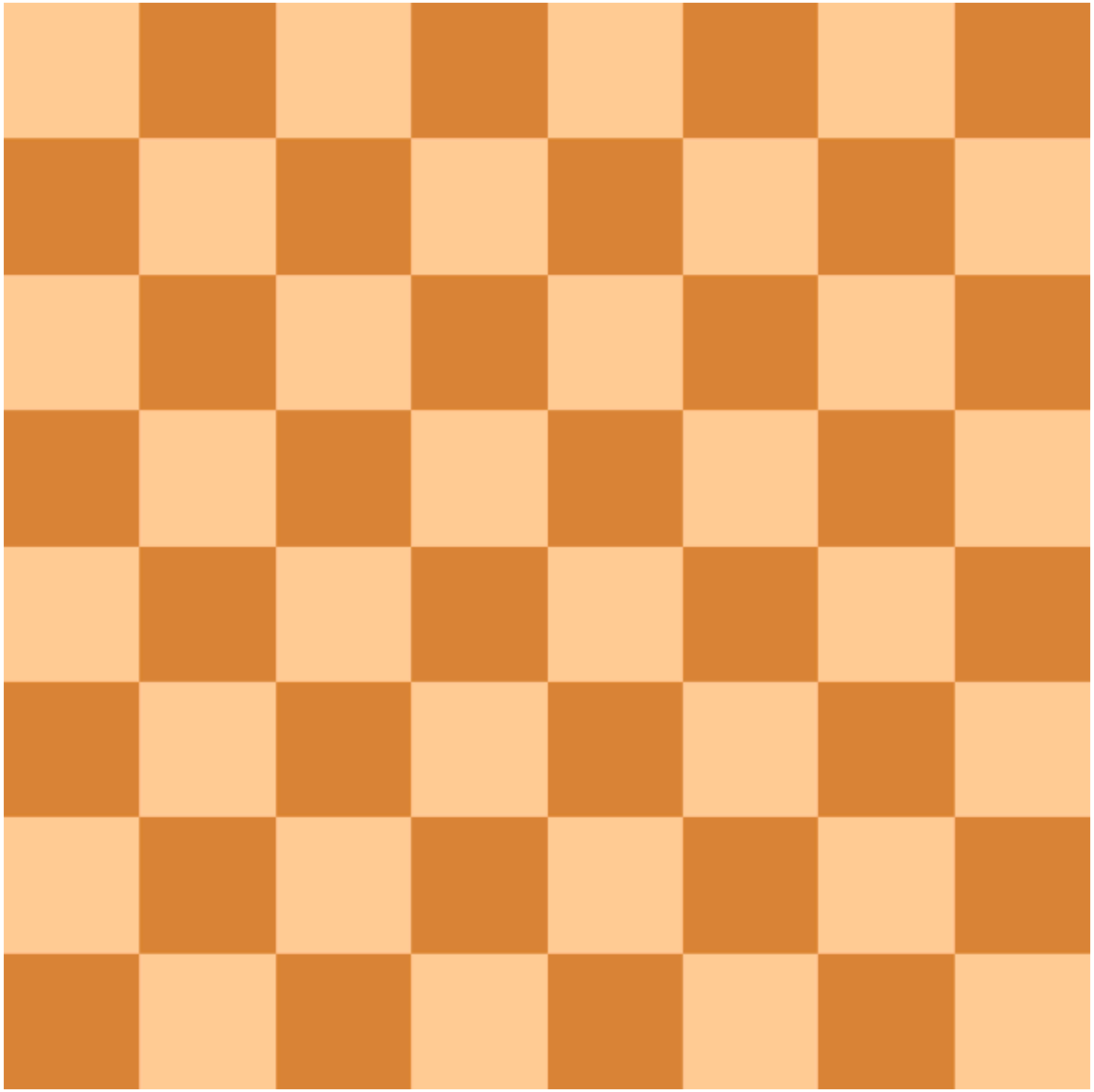
- The backtrack algorithm uses three lists plus one variable
  - **SL**, the state list, lists the states in the current path being tried. If a goal is found, SL contains the ordered list of states on the solution path.
  - **NSL**, the new state list, contains nodes awaiting evaluation -- i.e., nodes whose descendants have not yet been generated and searched.
  - **DE**, dead ends, lists states whose descendants have failed to contain a goal node. If these states are encountered again, they will be immediately eliminated from consideration.
  - **CS**, the current state.



A Trace of Backtrack Algorithm

AFTER ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]

Problema de les 8 reines:



Problema de les 8 reines:

- Solució per força bruta?

```
while there are untried configurations
{
   generate the next configuration
   if queens don't attack in this configuration then
   {
      print this configuration;
   }
}
```

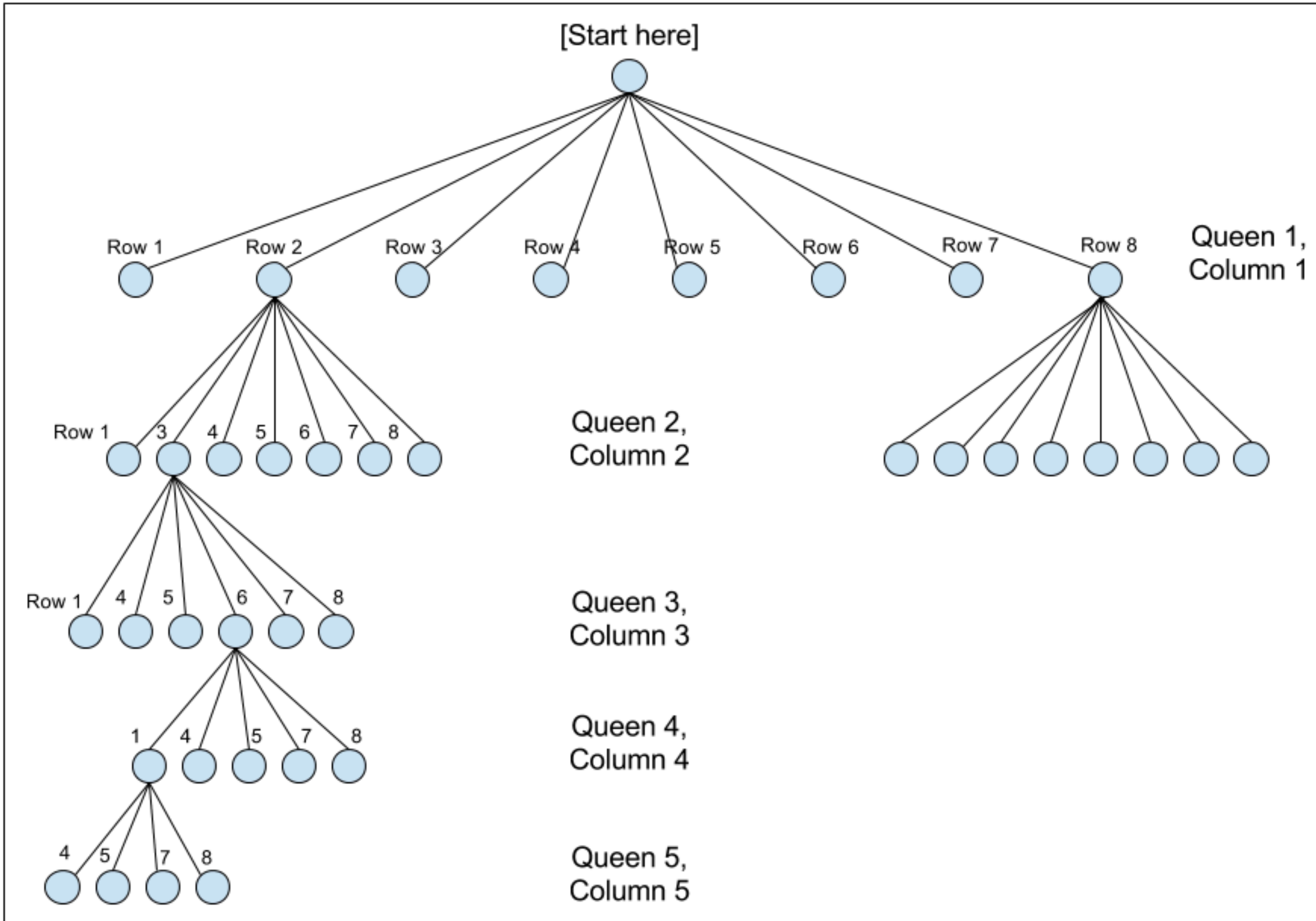
Problema de les 8 reines:

- Solució per força bruta?
  - Penseu una solució iterativa

Backtracking

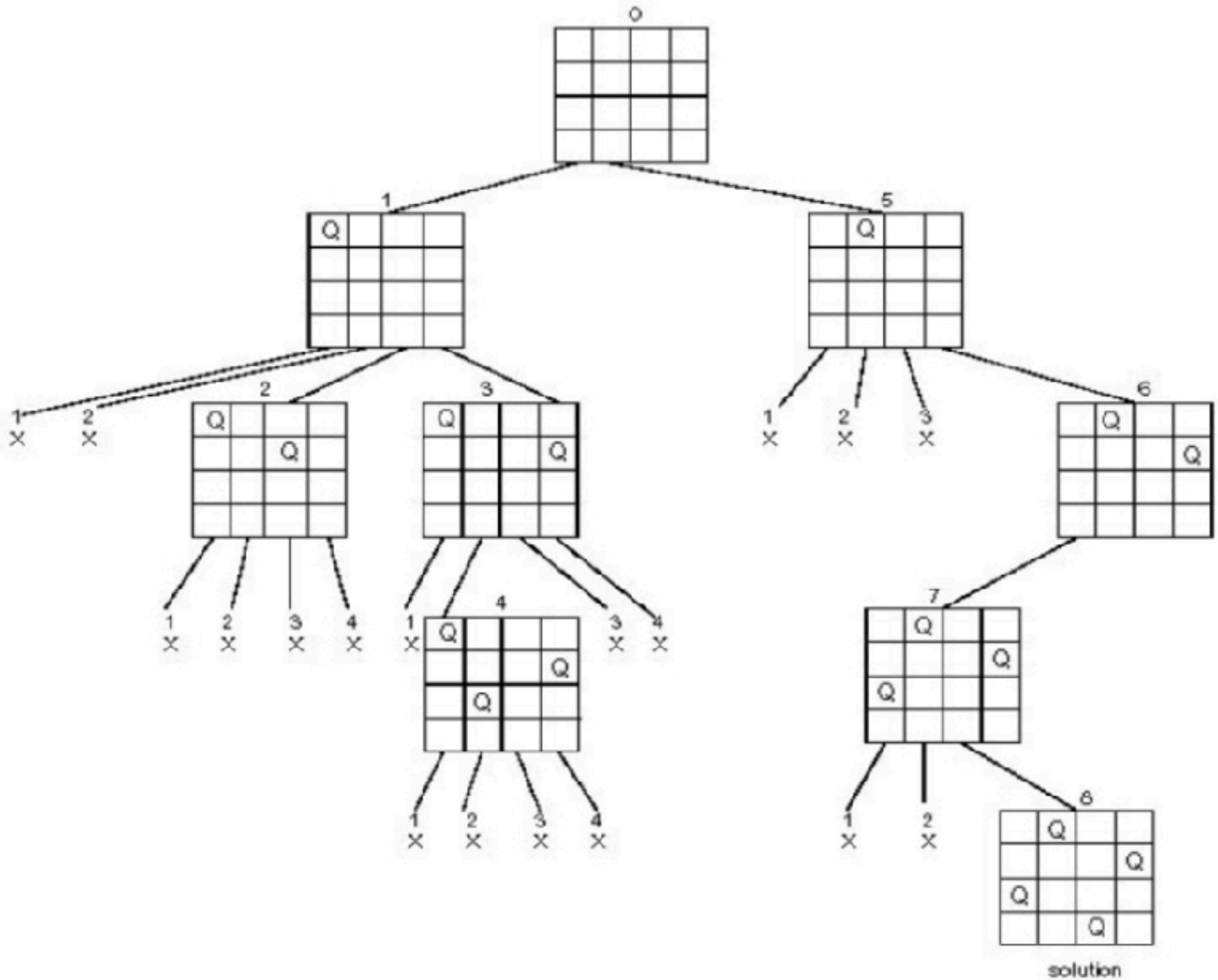
- Exemple: veure la utilitat del backtracking
- **Problema de les 8 reines:**
  - Volem col·locar 8 reines en un tauler d'escacs de 8x8 sense que hi hagi amenaça.
  - Comencem per una solució exhaustiva

```
bool ocho_reinas()
{
    for (int i=1; i<=8; i++)
        for (int j=1; j<=8; j++)
            for (int k=1; k<=8; k++)
                for (int l=1; l<=8; l++)
                    for (int m=1; m<=8; m++)
                        for (int n=1; n<=8; n++)
                            for (int o=1; o<=8; o++)
                                for (int p=1; p<=8; p++)
                                    if (solucion(i,j,k,l,m,n,o,p) return true;
    return false;
}
```


[Start here]
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7
Row 8
Queen 1,
Column 1
Row 1
3
4
5
6
7
8
Queen 2,
Column 2
Row 1
4
5
6
7
8
Queen 3,
Column 3
1
4
5
7
8
Queen 4,
Column 4
4
5
7
8
Queen 5,
Column 5

Problema de les 8 reines:

- Solució per força bruta?
  - Penseu una solució recursiva


solution

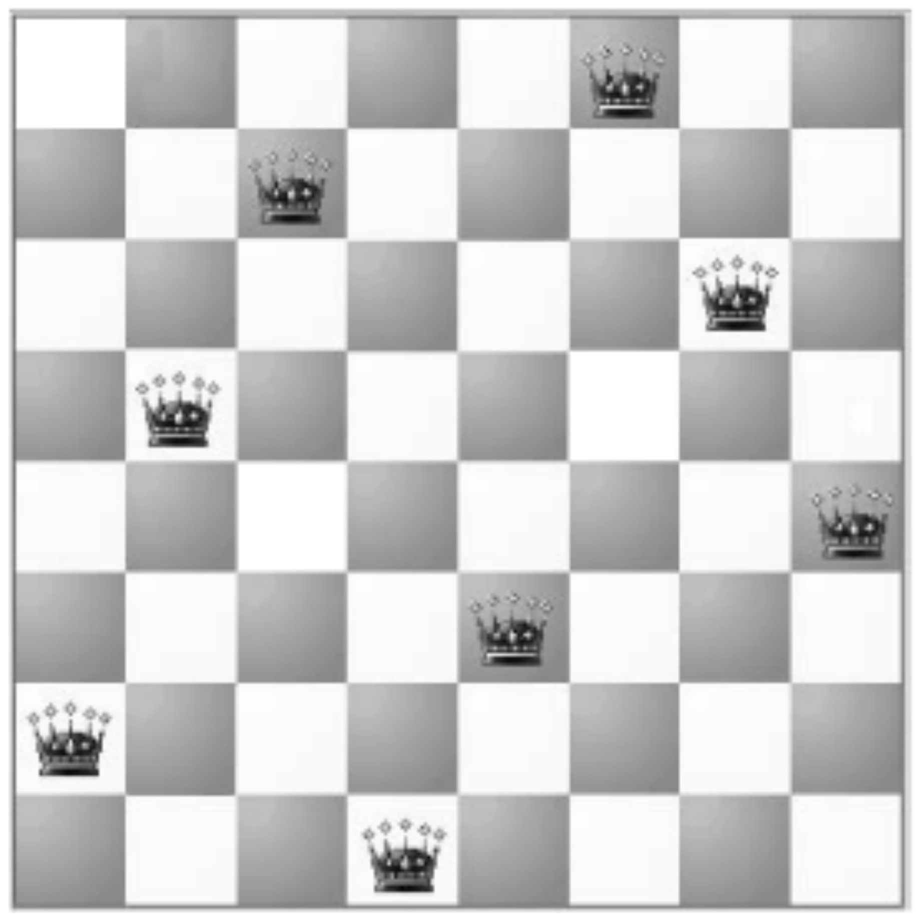
Problema de les 8 reines:

```
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.
   Do following for every tried row.
    a) If the queen can be placed safely in this row
       then mark this [row, column] as part of the
       solution and recursively check if placing
       queen here leads to a solution.
    b) If placing the queen in [row, column] leads to
       a solution then return true.
    c) If placing queen doesn't lead to a solution then
       unmark this [row, column] (Backtrack) and go to
       step (a) to try other rows.
3) If all rows have been tried and nothing worked,
   return false to trigger backtracking.
```

Backtracking

- Reduïm la complexitat: **incloure restriccions**
  - → Les reines han d'estar a diferents files i diferents columnes
  - → No podem estar a la mateixa diagonal:

Backtracking



Sudoku

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

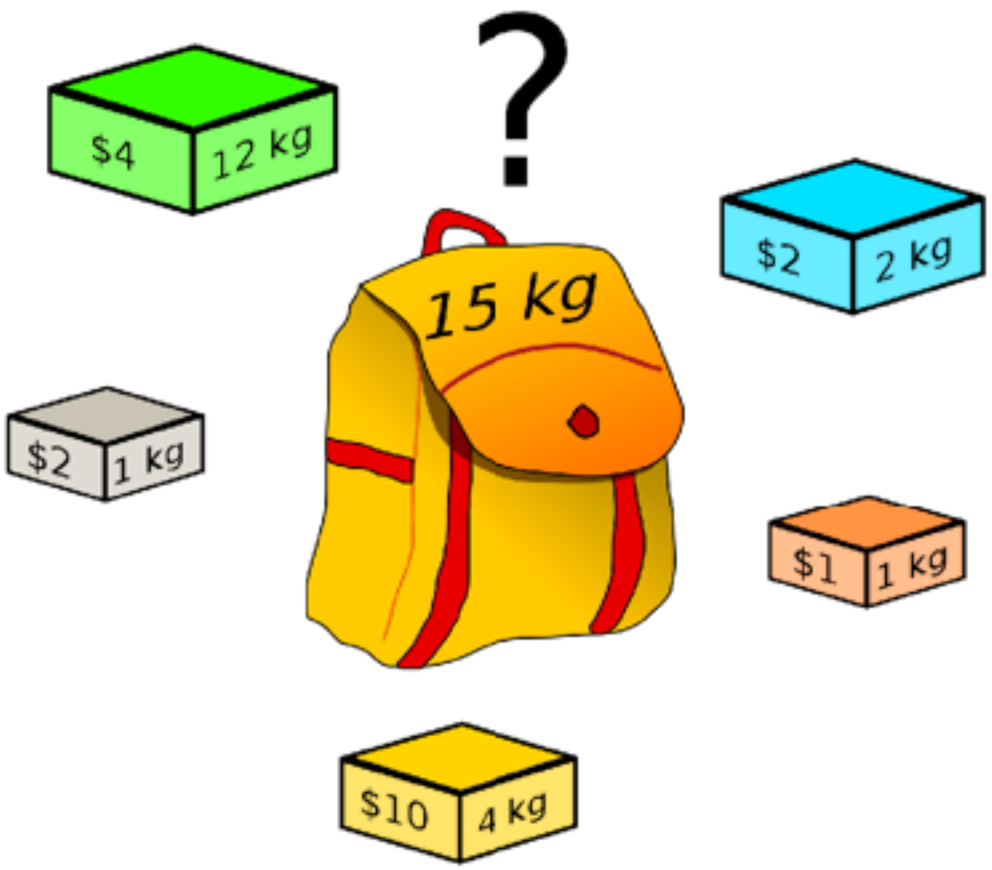
Solució amb Backtracking?

Sudoku

Solució amb Backtracking?

```
Find row, col of an unassigned cell
If there is none, return true
For digits from 1 to 9
  a) If there is no conflict for digit at row, col
       assign digit to row, col and recursively try fill in rest of grid
  b) If recursion successful, return true
  c) Else, remove digit and try another
If all digits have been tried and nothing worked, return false
```

Problema de la motxilla



Solució amb Backtracking?

Ramificació i poda

més conegut com ***Branch and Bound***

Ramificació i poda

- La **ramificació i poda implementen** l'algoritme de **backtracking**, però no a l'inversa.
- L'objectiu de ramificació i poda consisteix en reduir l'espai de cerca, per això s'introdueixen **cotes.** Guardant informació sobre l'execució d'un algoritme podem fer ús d'aquestes cotes per **ramificar** i **podar**.
- Utilitzat per a problemes d'optimització:
  - e.g.: $min_{x \in X} f(x)$

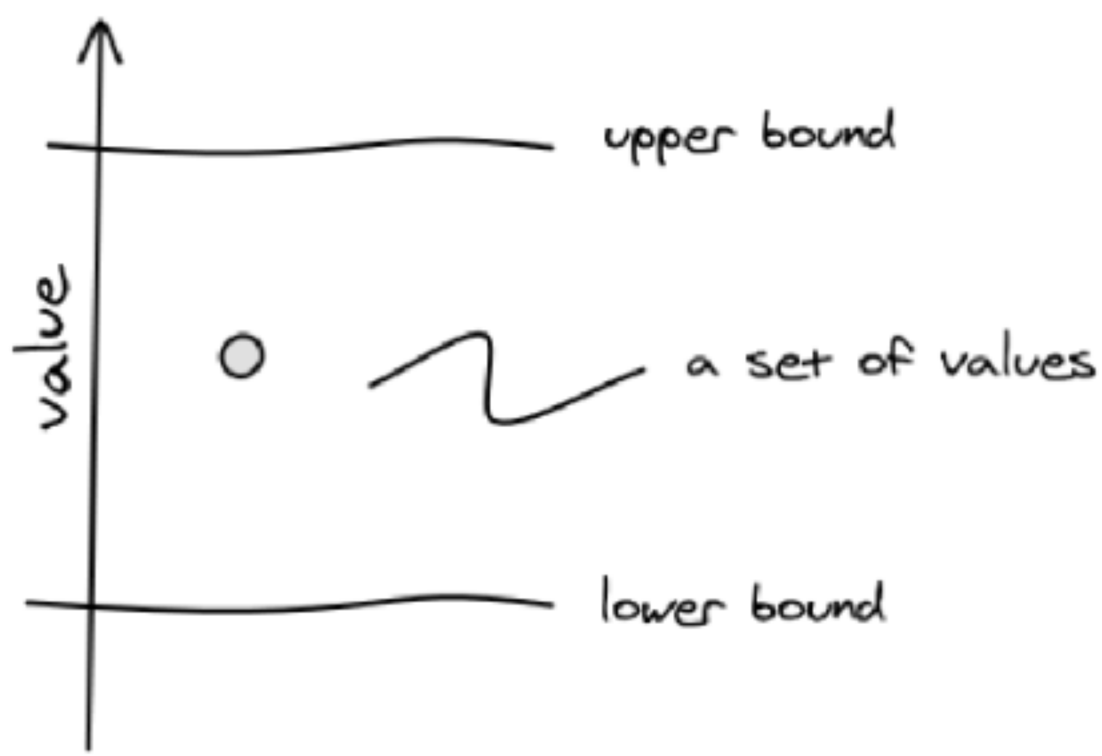
Ramificació i poda

- No limita l'exploració a la cerca amb profunditat (DFS)

Ramificació i poda

- Per a les cotes hem de guardar informació dels estats no explorats per retomar-les en el procés d’exploració.
- En problemes de **maximització**, la cota superior ens indica la solució máxima possible si seguim el node donat.
- En problemes de **minimizació**, la cota inferior ens indica la solució mínima si seguim el node donat.

Què és una cota?



Si tirem un dau 5 cops, la cota inferior és **x**
i la cota superior es **y**

Ramificació i poda

- Exemple d'assignació de costos:

→ Assignar un projecte a cada empresa **minimitzant** el cost total

	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Coste de las empresas a,b,c y d, por los proyectos A,B,C y D.

→ Inicialitzem una **cota superior**

$$a \rightarrow A, b \rightarrow B, c \rightarrow C, d \rightarrow D$$
$$z^* \leq 11 + 15 + 19 + 28 = 73$$

Ramificació i poda

- **Cota inferior:** suma mínims de cada columna

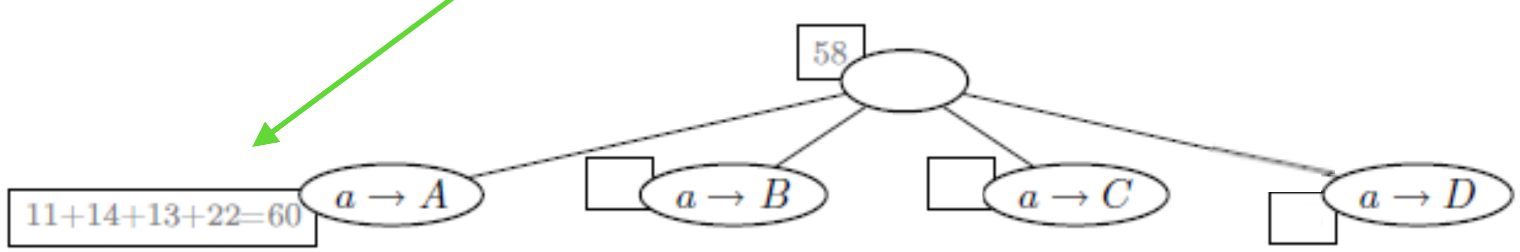
$$z^* \geq 11 + 12 + 13 + 22 = 58$$

	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Coste de las empresas a,b,c y d, por los proyectos A,B,C y D.

- Comencem la cerca

$$58 \leq z^* \leq 73$$



Ramificació i poda

- **Cota inferior**: suma mínims de cada columna

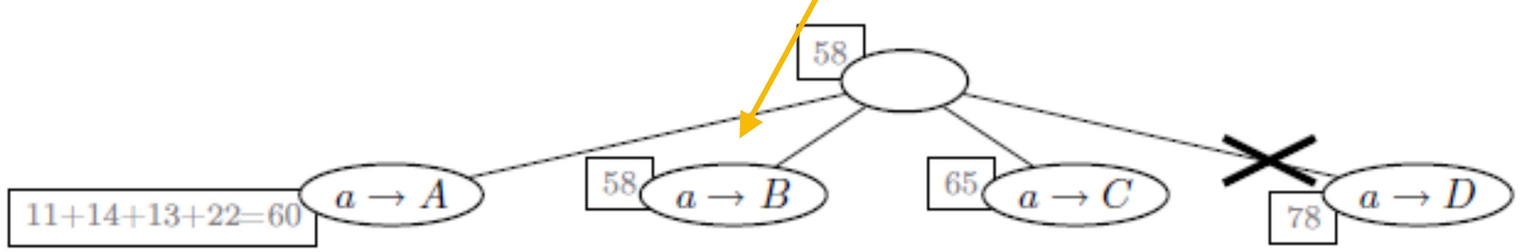
$$z^* \geq 11 + 12 + 13 + 22 = 58$$

	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Coste de las empresas a,b,c y d, por los proyectos A,B,C y D.

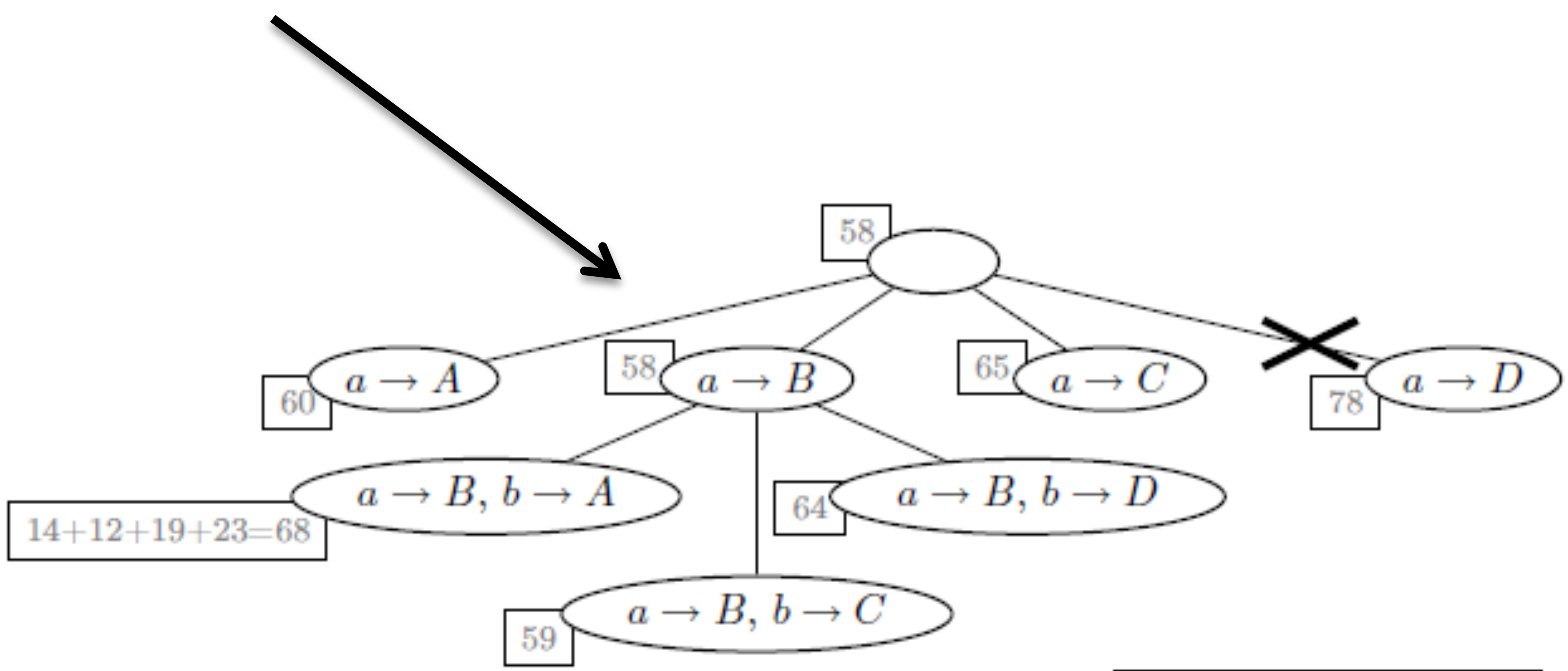
- Comencem la cerca

$$58 \leq z^* \leq 73$$



Ramificació i poda

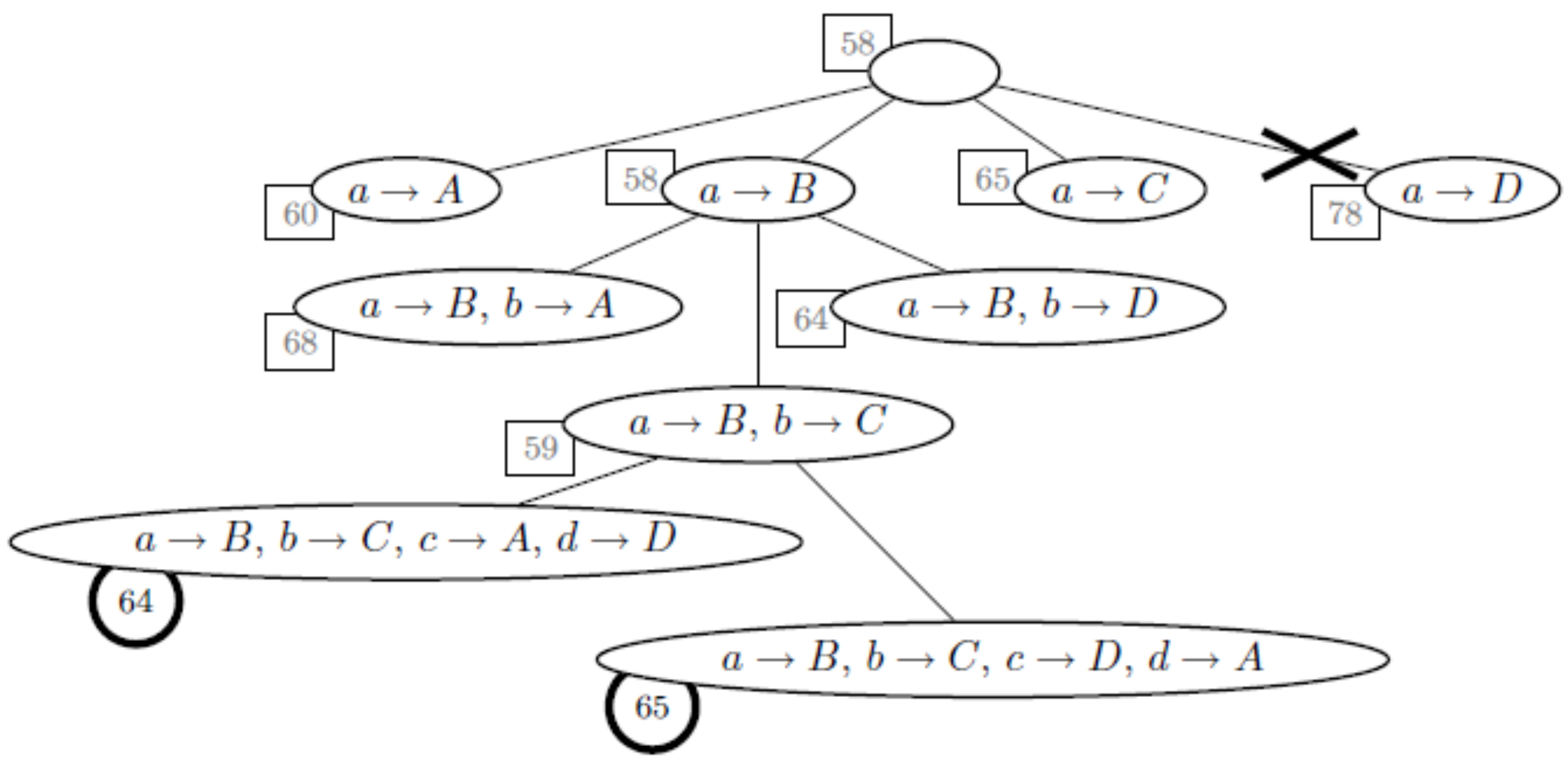
- ***Branching***: quina rama mereix la pena ser explorada?



	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Coste de las empresas a,b,c y d, por los proyectos A,B,C y D.

Ramificació i poda

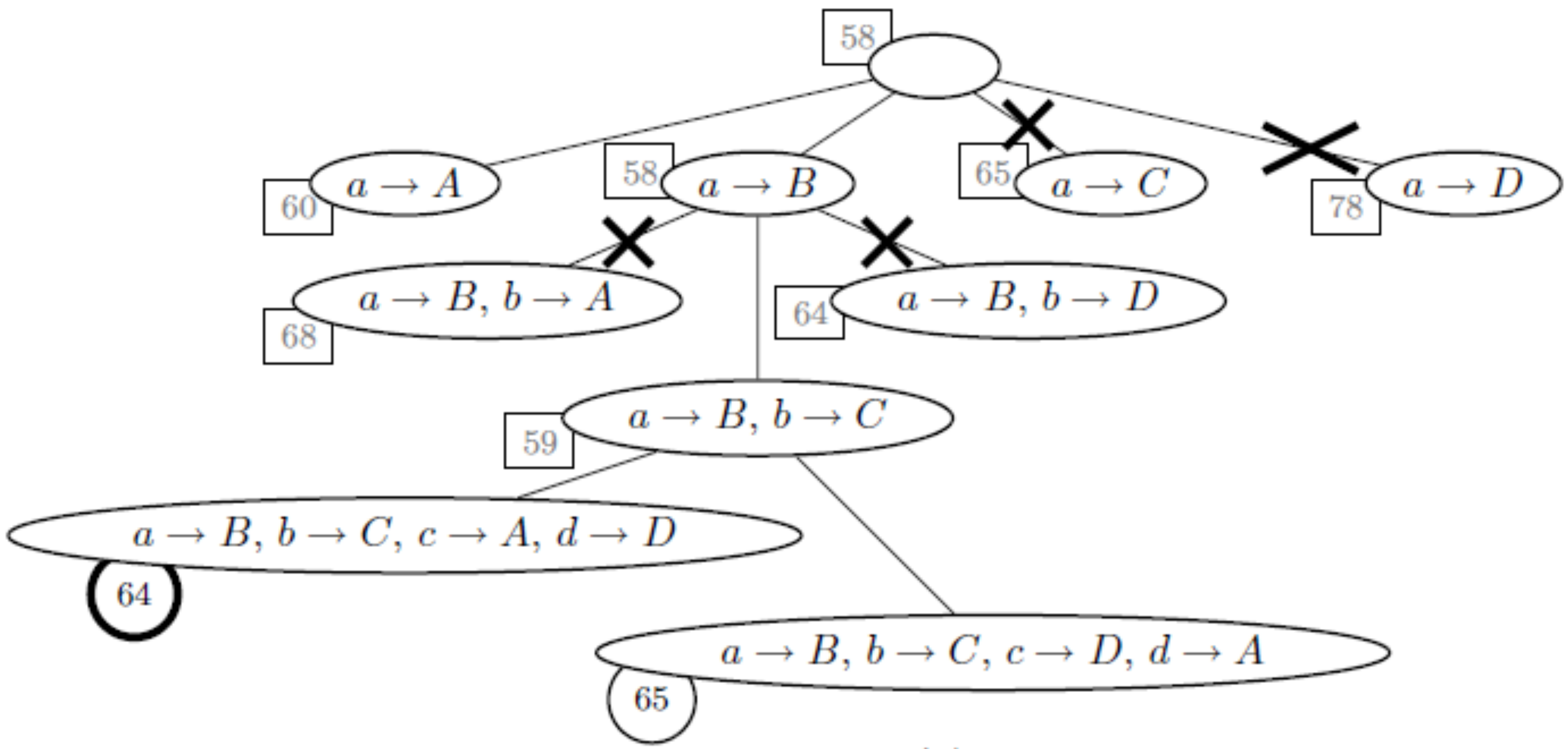


- Ja no és una cota, sinó un possible valor de la funció objectiu

$$58 \leq z^* \leq 64$$

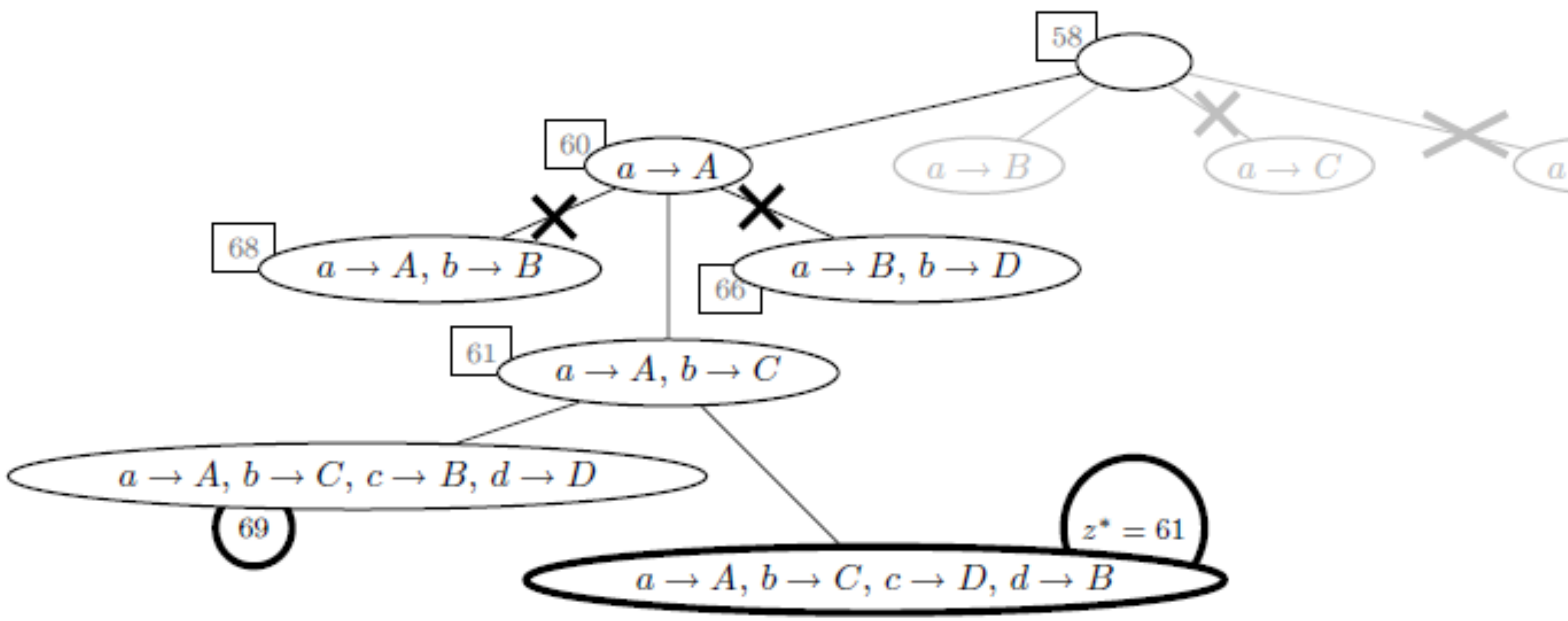
Ramificació i poda

$$58 \leq z^* \leq 64$$



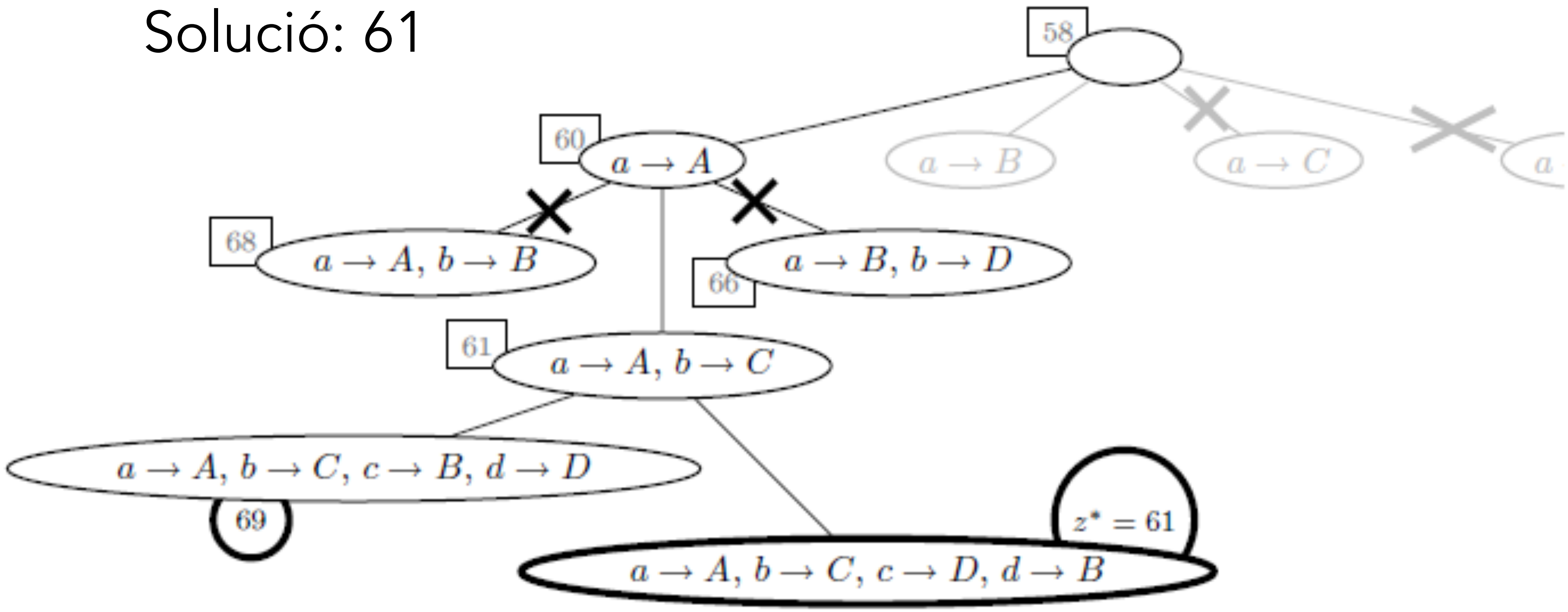
Ramificació i poda

- ***Solució:*** *61*



Ramificació i poda

Solució: 61



	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

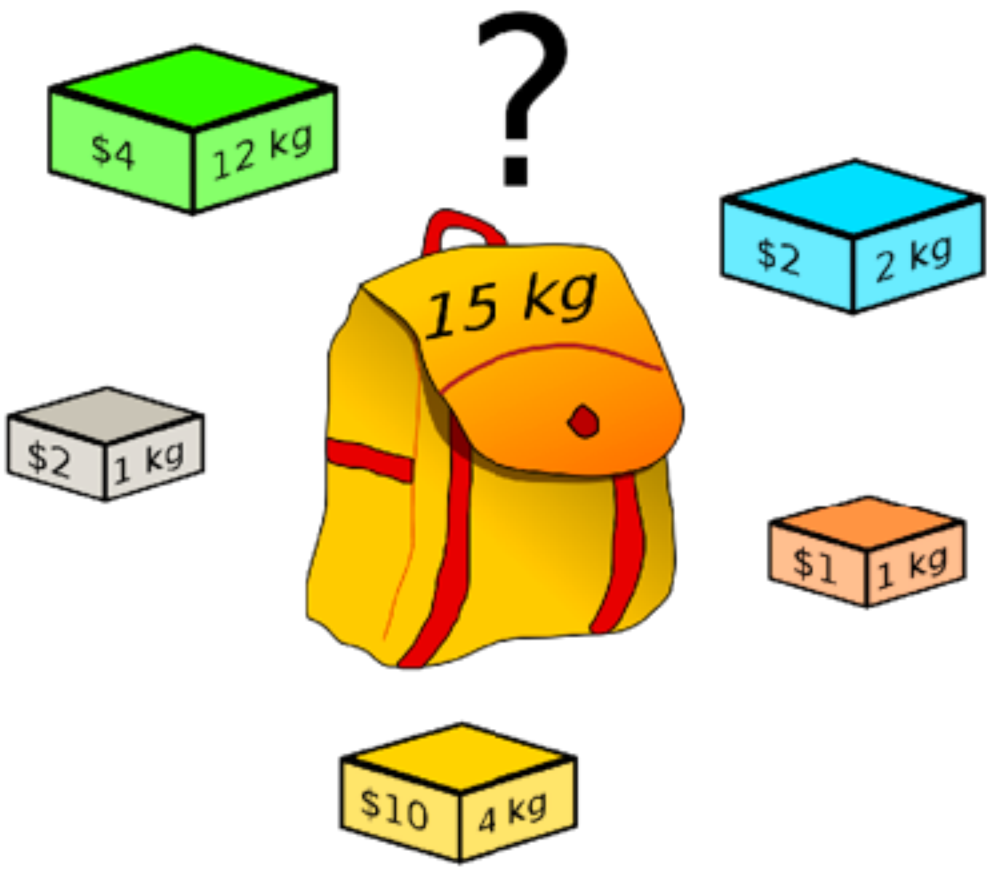
```
begin
    activeset :={∅};
    bestval:=NULL;
    currentbest:=NULL;
    while activeset is not empty do
        choose a branching node, node k ∈ activeset;
        remove node k from activeset;
        generate the children of node k, child i, i=1,...,n_k,
         and corresponding optimistic bounds ob_i;
        for i=1 to n_k do
            if ob_i worse than bestval then kill child i;
            else if child is a complete solution then
                bestval:=ob_i, currentbest:=child i;
            else add child i to activeset
        end for
    end while
end
```

Ramificació i poda

- Fes l'arbre de ramificació i poda de la següent taula (seguint el problema de l'exemple anterior). Numera els passos i actualitza els valors de les cotes.

	A	B	C	D
a	1	13	3	12
b	3	5	9	20
c	5	10	2	17
d	7	2	10	21

Problema de la motxilla



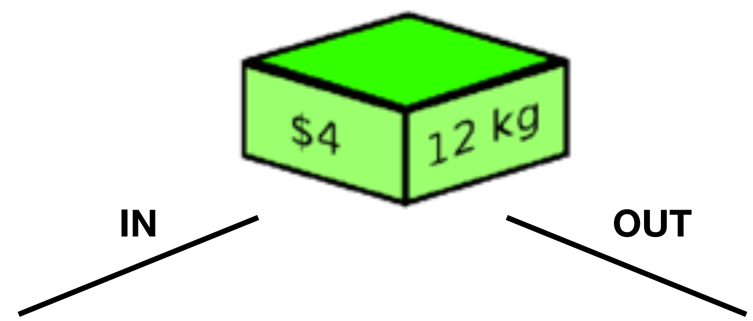
Solució amb **ramificació i poda?**

Definim cota inferior: una solució qualsevol

7$

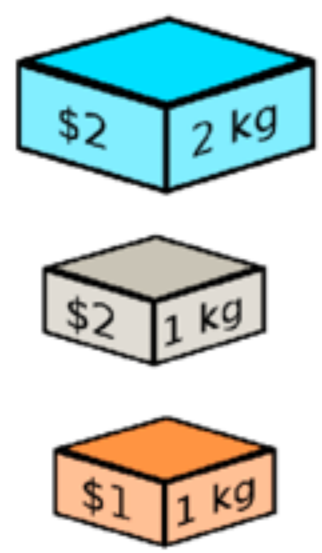
Problema de la motxilla

Definim cota inferior: 7$

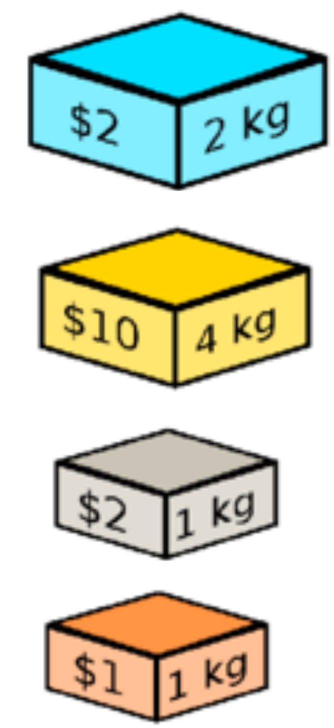


Cota superior i cota inferior dels nodes?

Cota superior = 4 + 5 = 9 $

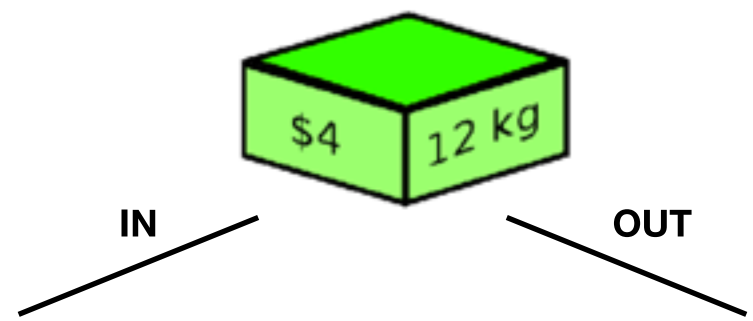


Cota superior = 12 + 10 + 2 +1 = 25$



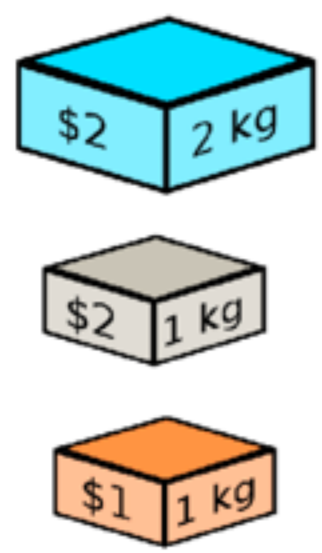
Problema de la motxilla

Definim cota inferior: 7$

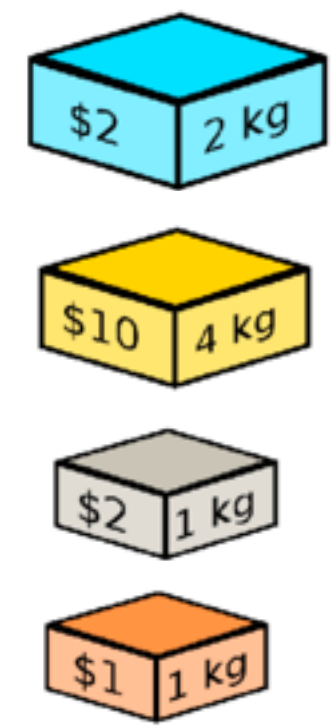


Cota superior i cota inferior dels nodes?

Cota superior = 4 + 5 = 9 $
Cota Inferior = 6 $

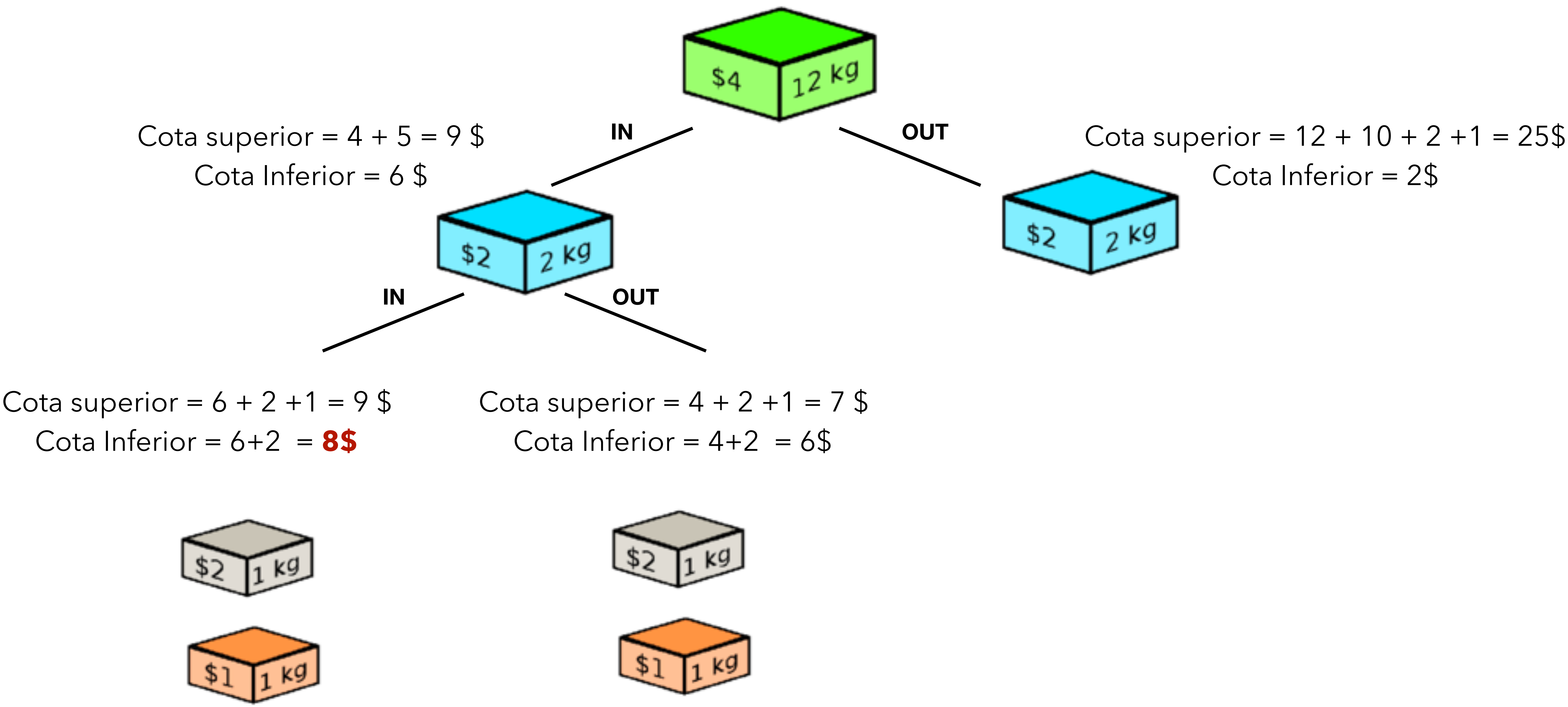


Cota superior = 12 + 10 + 2 +1 = 25$
Cota Inferior = 2$



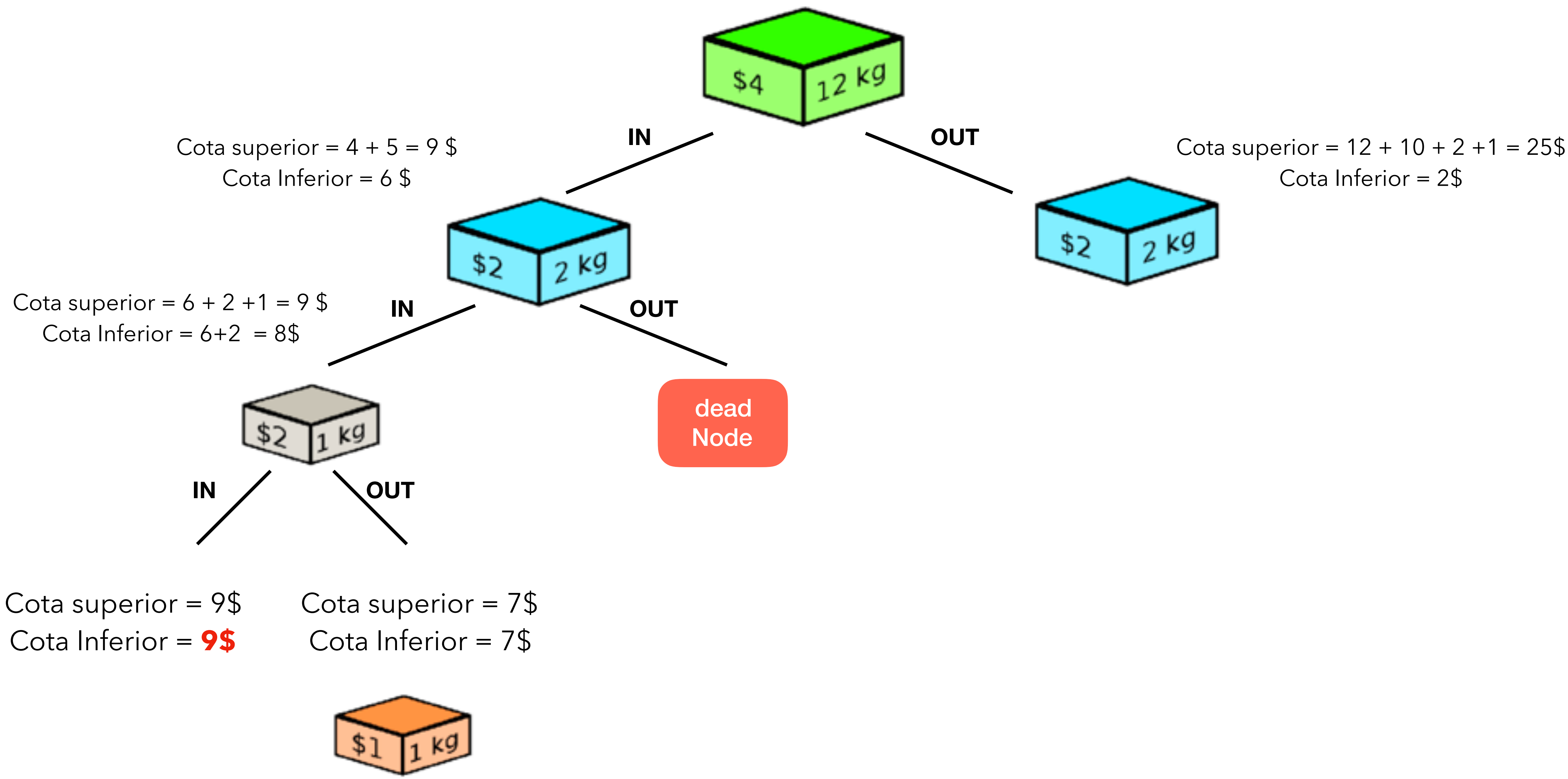
Problema de la motxilla

Definim cota inferior: 7$



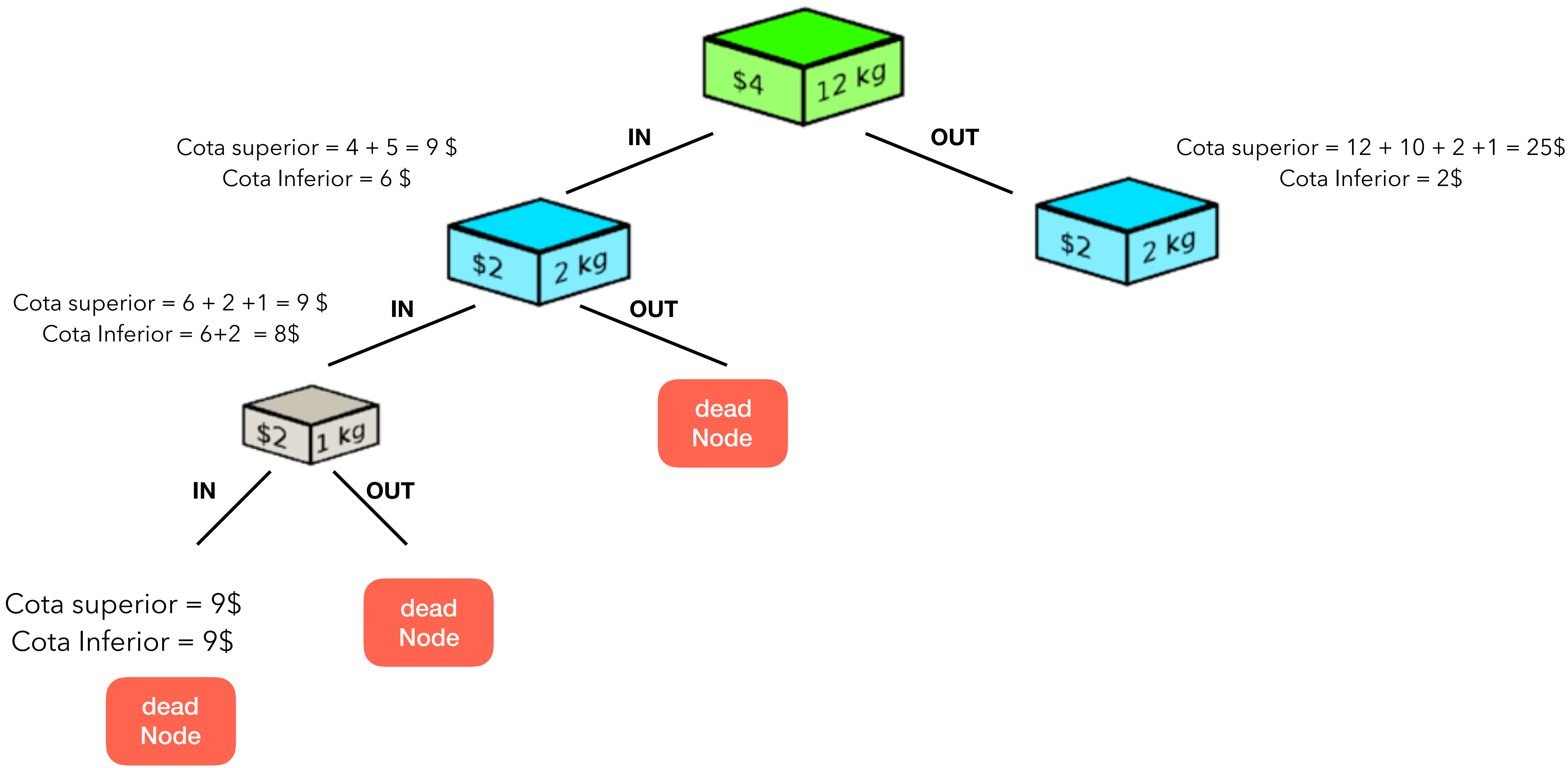
Problema de la motxilla

Definim cota inferior: 8$



Problema de la motxilla

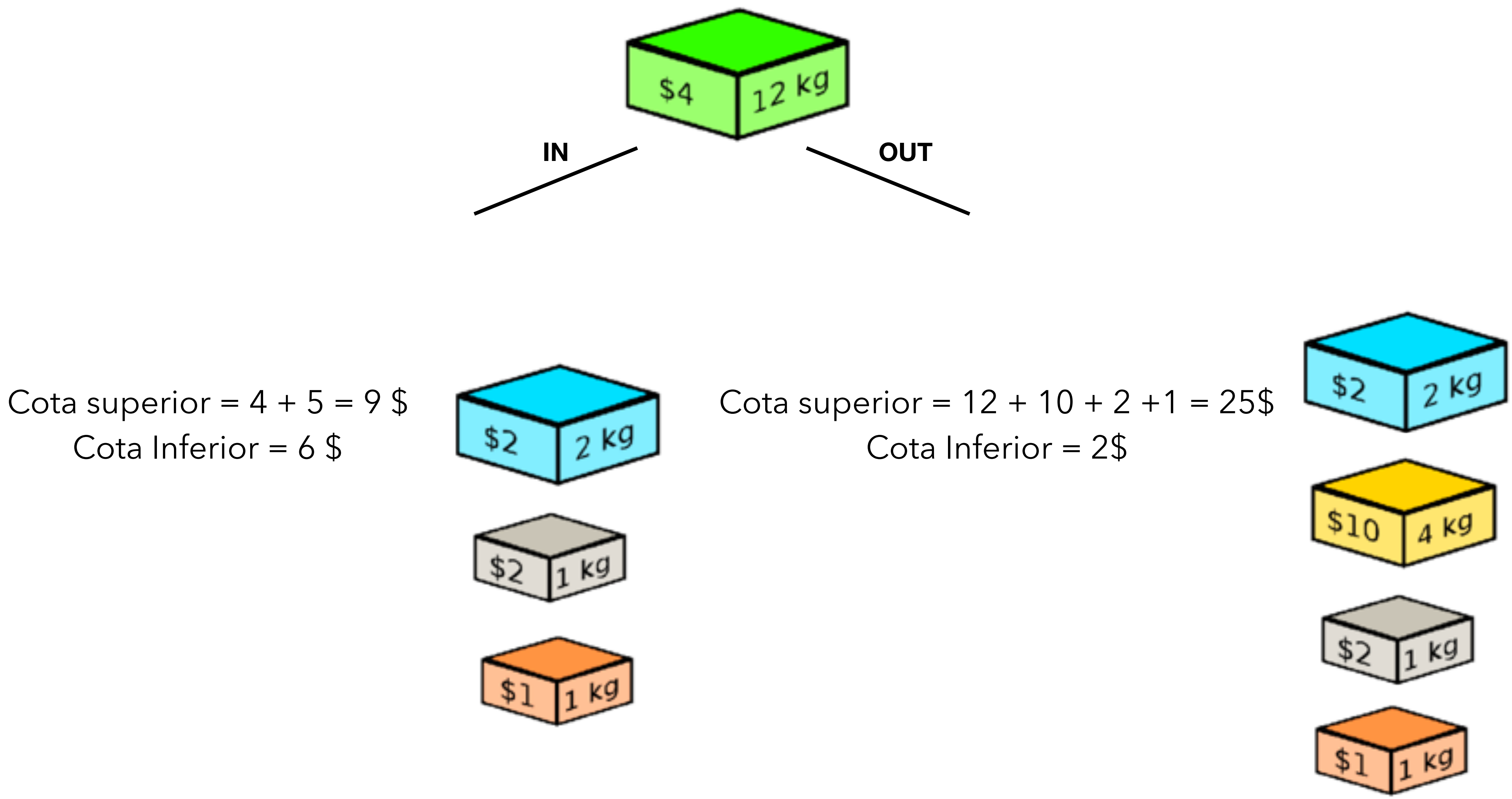
Definim cota inferior: 9$



Tornem al principi

Problema de la motxilla

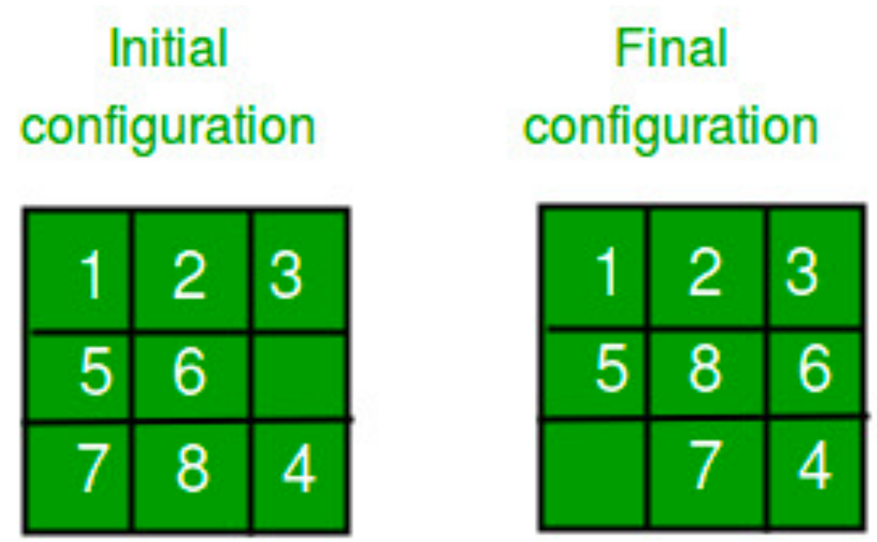
Definim cota inferior: 7$

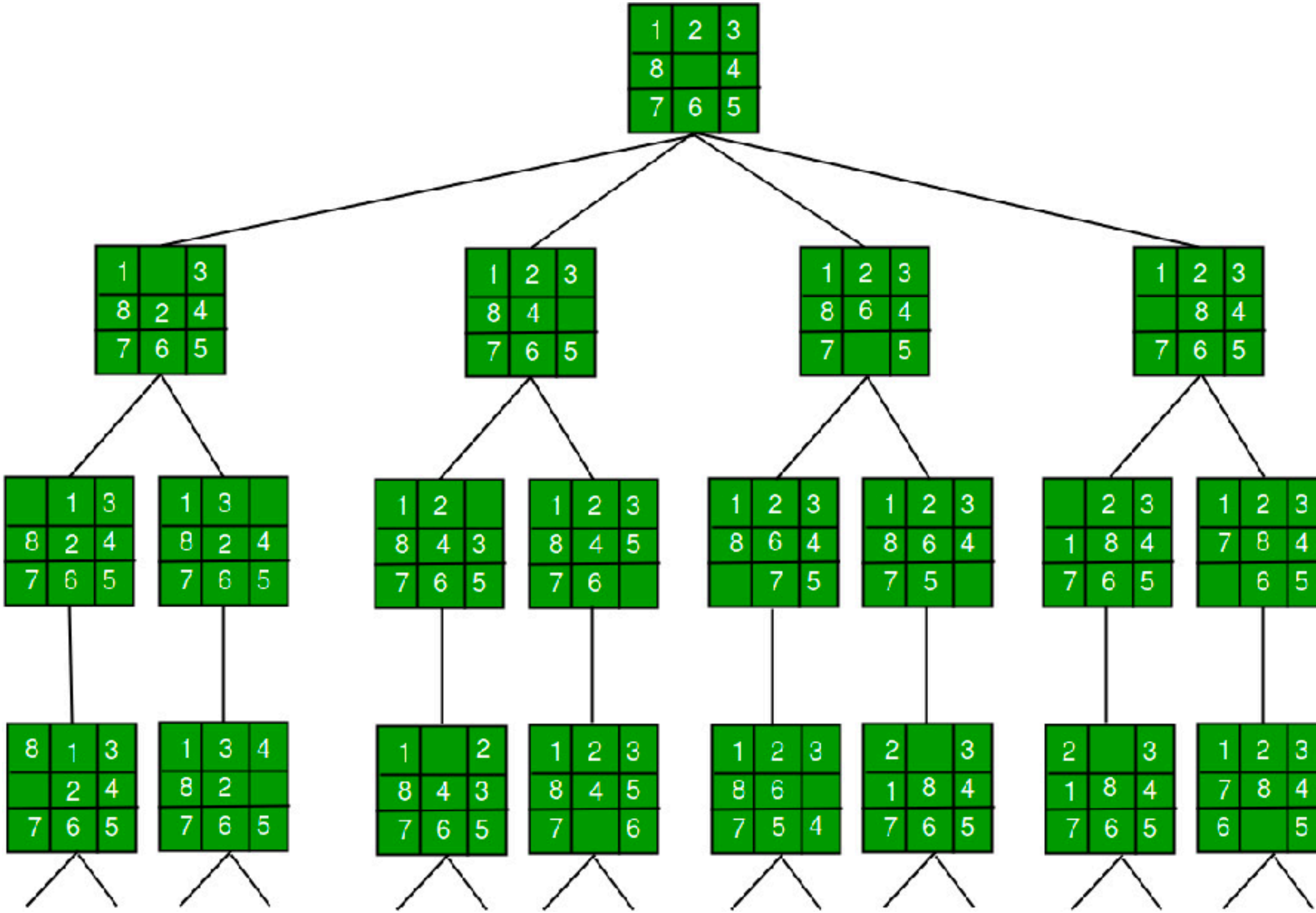


Quin node hauríem d'explorar primer???

Problema

- Donat un taulell de 3x3 amb 8 números i un espai buit. L'objectiu és col·locar els números a les cel·les perquè coincideixin amb la configuració final mitjançant l'espai buit. Podem lliscar les cel·les adjacents (dreta, esquerra, amunt i avall) de l'espai buit.





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

Solució amb ramificació i poda

A la **ramificació i poda** tenim tres tipus de nodes:
Nodes actius: és un node que s'ha generat però els fills encara no s'han generat.
E-Node: és un node viu amb els fills que s'estan explorant actualment. És a dir, un node E és un node que s'està ampliant actualment.
Dead node: és un node generat que no s'expandirà ni explorarà més.

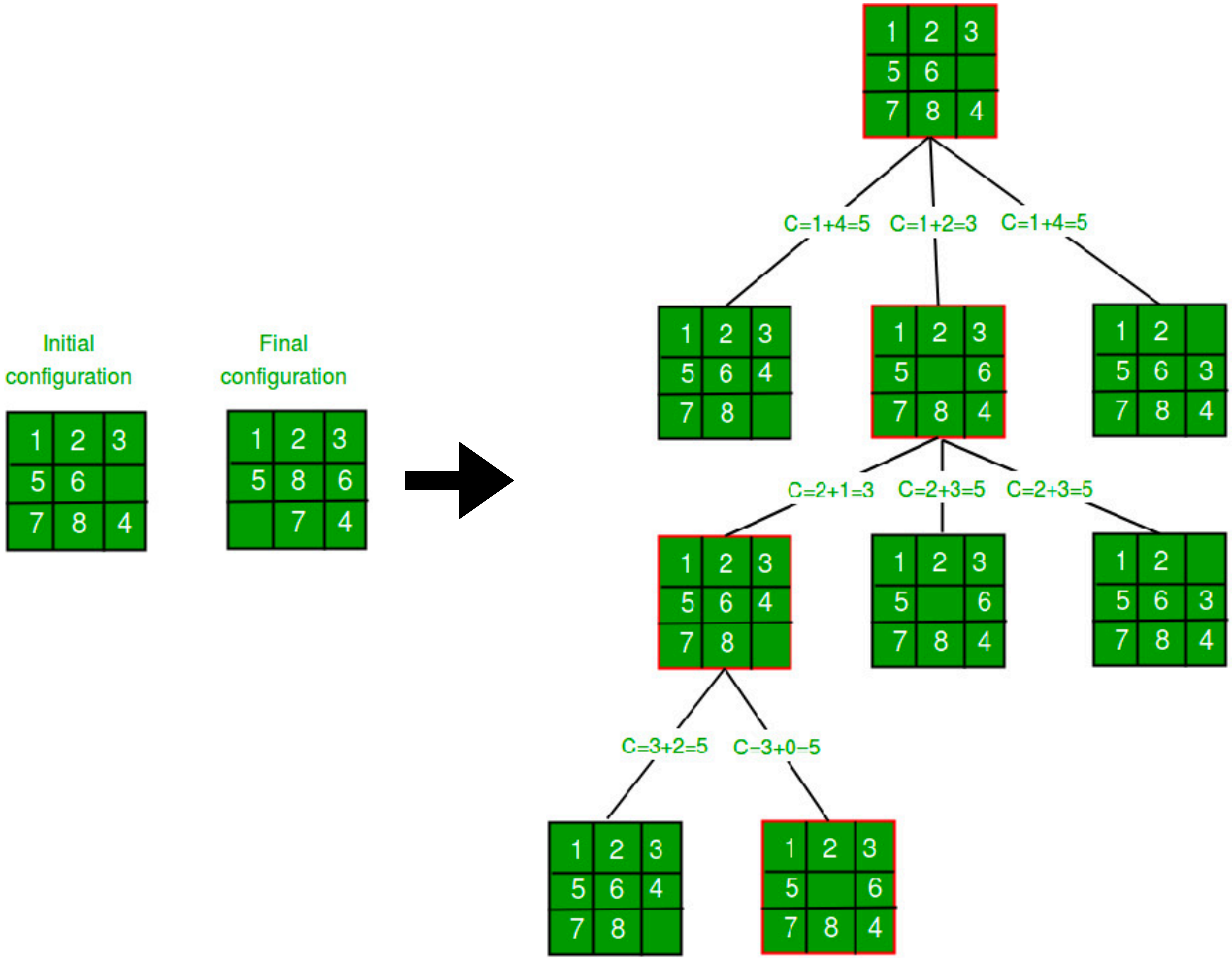
Associem a cada node de l'arbre un cost. El cost ens permetrà determinar quin quin és el següent **E-node.** El següent E-node serà aquell amb un cost menor. Definim la funció de cost de la següent manera:

```
C(X) = g(X) + h(X) where
g(X) = cost of reaching the current node
       from the root
h(X) = cost of reaching an answer node from X.
```

Solució amb ramificació i poda

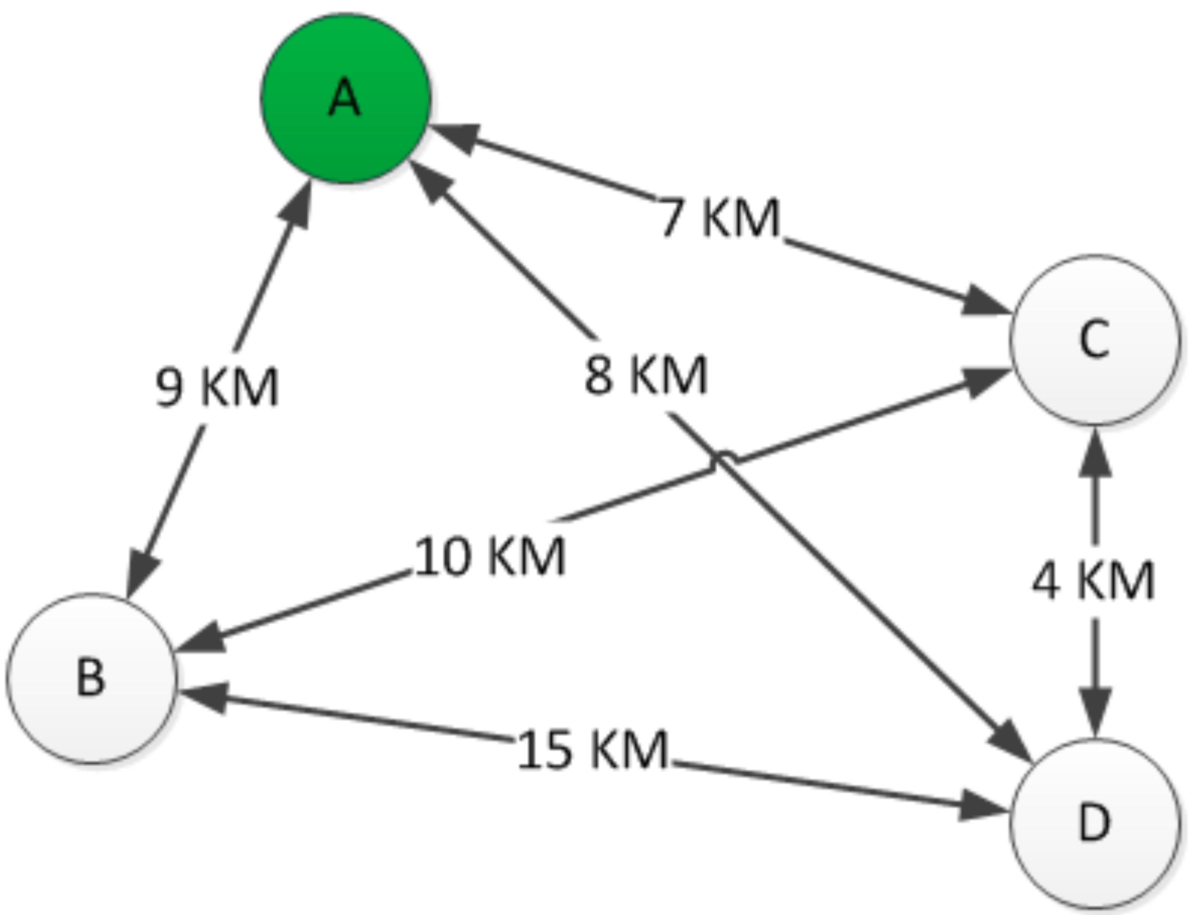
La funció de cost per al problema anterior:
Assumim que moure una cel·la cap a qualsevol direcció té un cost de 1.

```
c(x) = f(x) + h(x) where
f(x) is the length of the path from root to x
     (the number of moves so far) and
h(x) is the number of non-blank tiles not in
     their goal position (the number of mis-
     -placed tiles). There are at least h(x)
     moves to transform state x to a goal state
```


Initial configuration
Final configuration
C=1+4=5
C=1+2=3
C=1+4=5
C=2+1=3
C=2+3=5
C=2+3=5
C=3+2=5
C=3+0=5

```
lb = ∞

while(live_node_set is not null)

    choose a branching node, k, such that k ∈ live_node_set; /* k is a E-node */

    live_node_set = live_node_set - {k};

    #Generate the children of node k and the corresponding lower bounds;

    Sk={(i,zi): i is child of k and zi its lower bound}

    For each element (i,zi) in Sk

        If zi > lb

            Kill child i; /* i is a child node */

        Else

            If i is a solution

                lb = zi;

                current best = i;

            Else

                Add i to live_node_set;
```

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 distancia 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.**

Solució amb **ramificació i poda**?

Ramificació i poda

- La **clau**: funció de **prioritat i cota**
- Problema del viatjant de comerç
  - → Passar per tots els nodes **minimitzant** el cost de la ruta, passant només un cop per cada node i acabant en el node de sortida (circuit ***hamiltonià***)

Problema del viatjant de comerç

- Per tal de resoldre el problema del viatjant de comerç, mitjançant el mètode de ramificació i poda,haurem de construir un **arbre** on les **fulles** siguin **camins hamiltonians.**

Ramificació i poda

- Formalització:
  - Sean $G=(V,A)$ un grafo orientado,
    $V=\{1,2,\ldots,n\}$,
    $D[i,j]$ la longitud de $(i,j)\in A$,
    $D[i,j]=\infty$ si no existe el arco (i,j).
  - El circuito buscado empieza en el vértice 1.

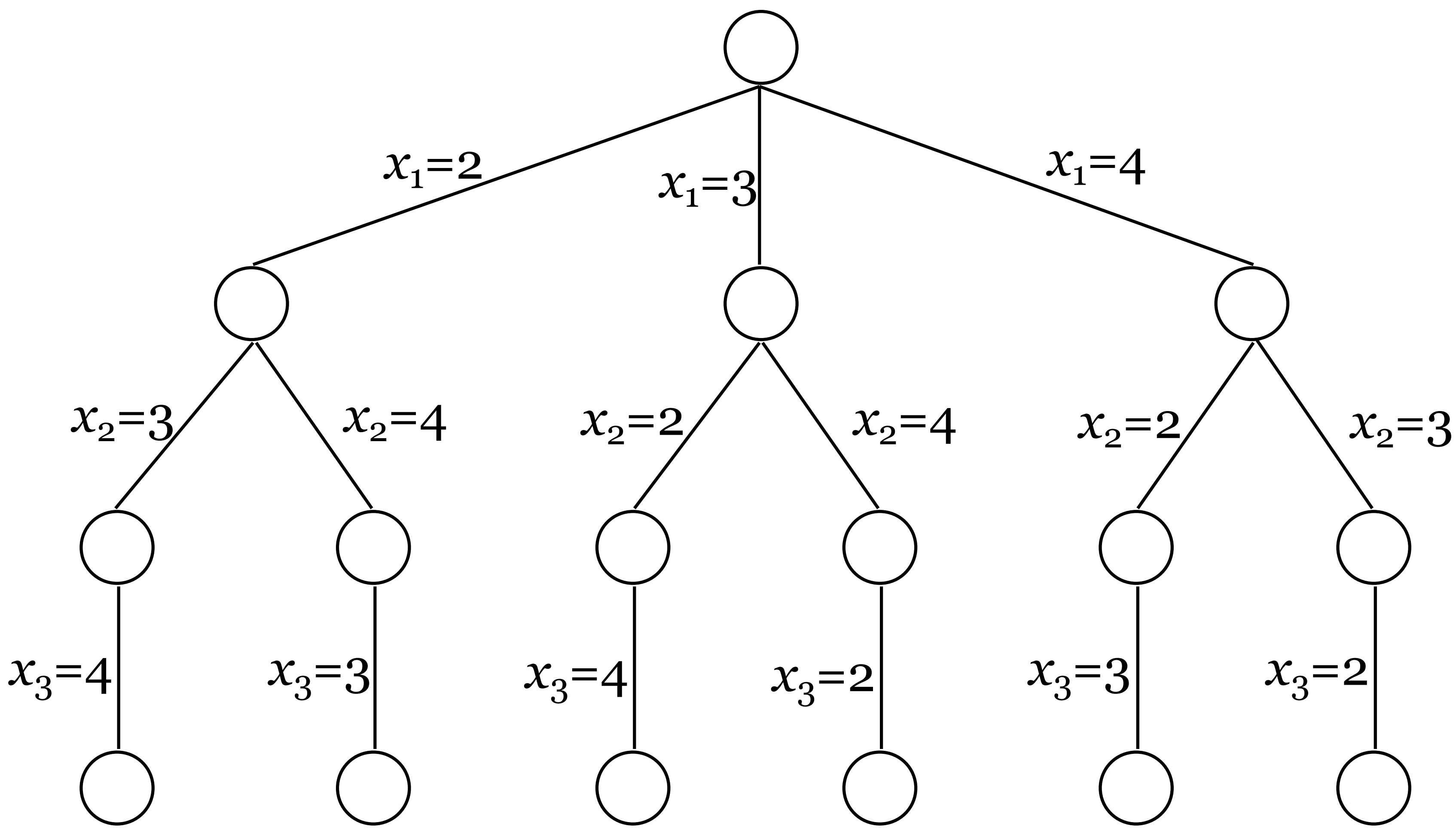
  - Candidatos:
    $E = \{ 1,X,1 \mid X \text{ es una permutación de } (2,3,\ldots,n) \}$
    $|E| = (n-1)!$

  - Soluciones factibles:
    $E = \{ 1,X,1 \mid X = x_1,x_2,\ldots,x_{n-1}$, es una permutación de $(2,3,\ldots,n)$ tal que $(i_j,i_{j+1})\in A$, $0<j<n$, $(1, x_1) \in A$, $(x_{n-1},1) \in A\}$

  - Funcion objetivo:
    $F(X)=D[1,x_1]+D[x_1, x_2] + D[x_2, x_3]+\ldots+D[x_{n-2}, x_{n-1}]+D[x_n,1]$

Ramificació i poda

- Representació per |V|=4



Ramificació i poda

- Definició d'una cota (X,k) molt senzilla:
  - Suma de les arestes ja escollides

$$cota(X,k) = D[1,X_1] + \sum_{i=1,\ldots,k-2} D[X_i, X_{i+1}]$$

  - Exemple: (n=5)

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

El problema està com calcular, de la millor manera, les **cotes**

Let S be some subset of solutions. Let

$$L(S) = \text{a lower bound on the cost of any solution belonging to } S$$

$$\text{Let } C = \text{cost of the best solution found so far}$$

If $C \leq L(S)$, there is no need to explore S because it does not contain any better solution.

If $C > L(S)$, then we need to explore S because it may contain a better solution.

Ramificació i poda

- Cota senzilla: Suma de les arestes ja escollides

$$cota(X,k) = D[1,X_1] + \sum_{i=1,\ldots,k-2} D[X_i, X_{i+1}]$$

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

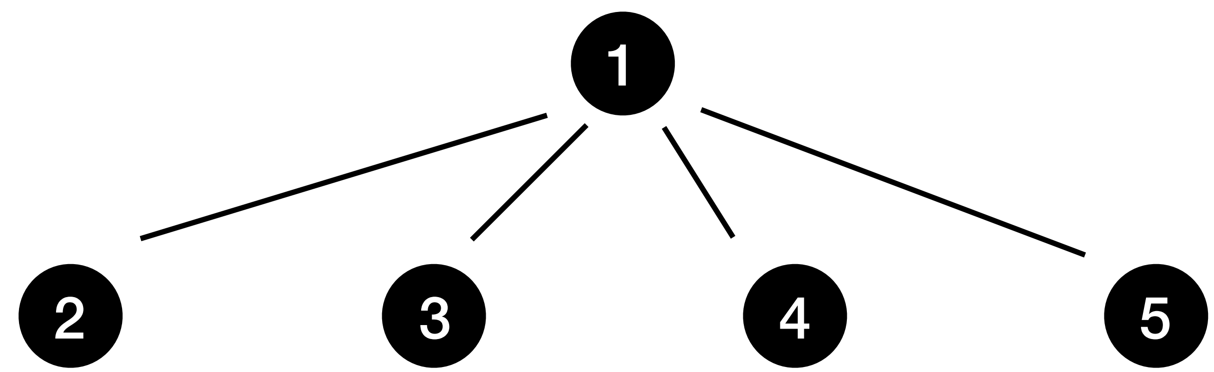
Suposem el camí: 1 - 2 - 3 - 4 - 5 - 1

Cota superior = 20 +16 + 2 + 3 + 16 = 57

Ramificació i poda

Cota superior = 20 +16 + 2 + 3 + 16 = 57

$$\begin{bmatrix} \infty & \boxed{20} & 30 & 10 & 11 \\ 15 & \infty & \boxed{16} & 4 & 2 \\ 3 & 5 & \infty & \boxed{2} & 4 \\ 19 & 6 & 18 & \infty & \boxed{3} \\ \boxed{16} & 4 & 7 & 16 & \infty \end{bmatrix}$$



20 + 16 + 2 + 3 +16 = 57

30 + 4 + 4 + 19 + 4 = 54

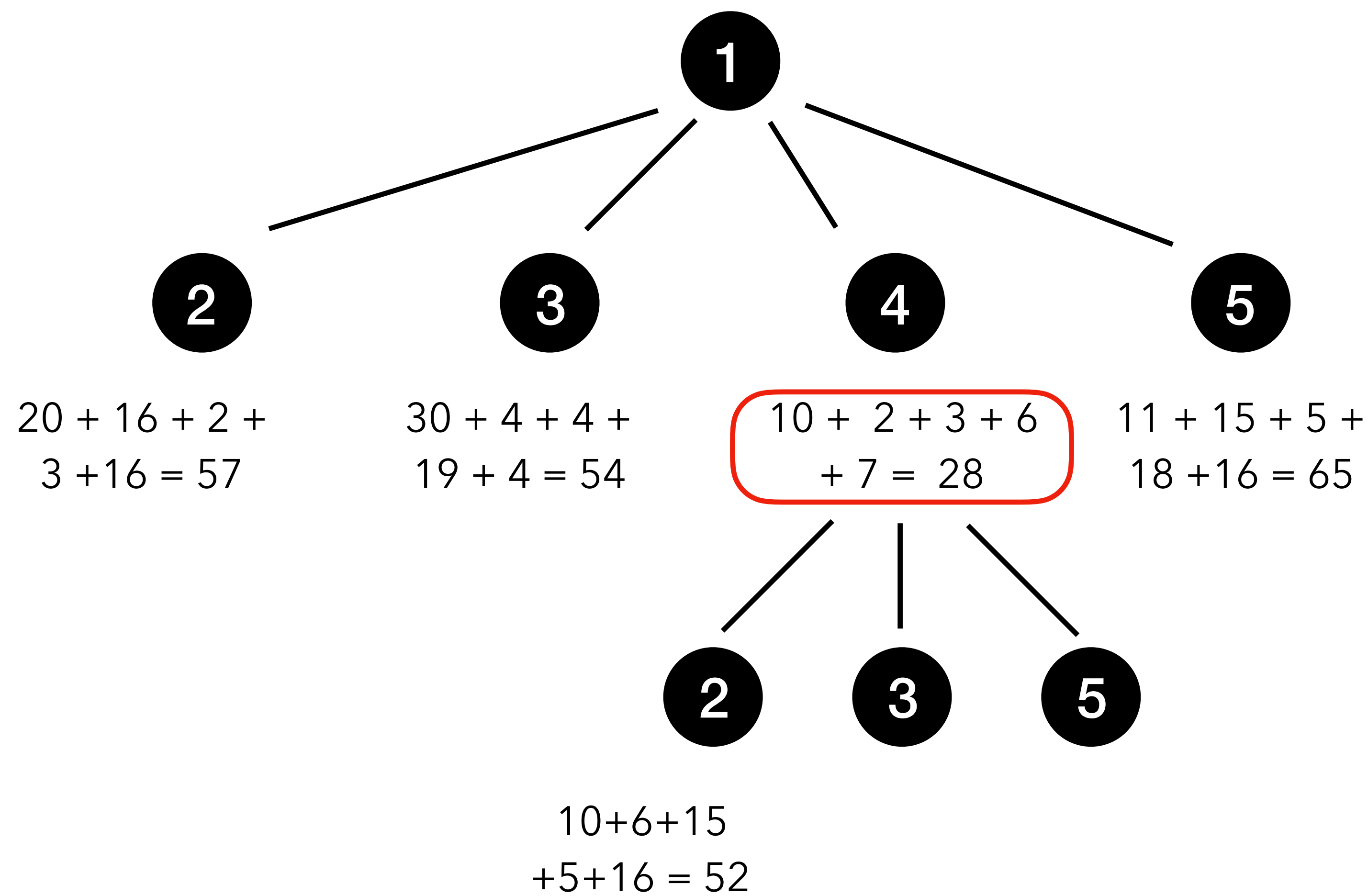
10 + 2 + 3 + 6 + 7 = 28

11 + 15 + 5 + 18 +16 = 65

Ramificació i poda

Cota superior = 20 +16 + 2 + 3 + 16 = 57

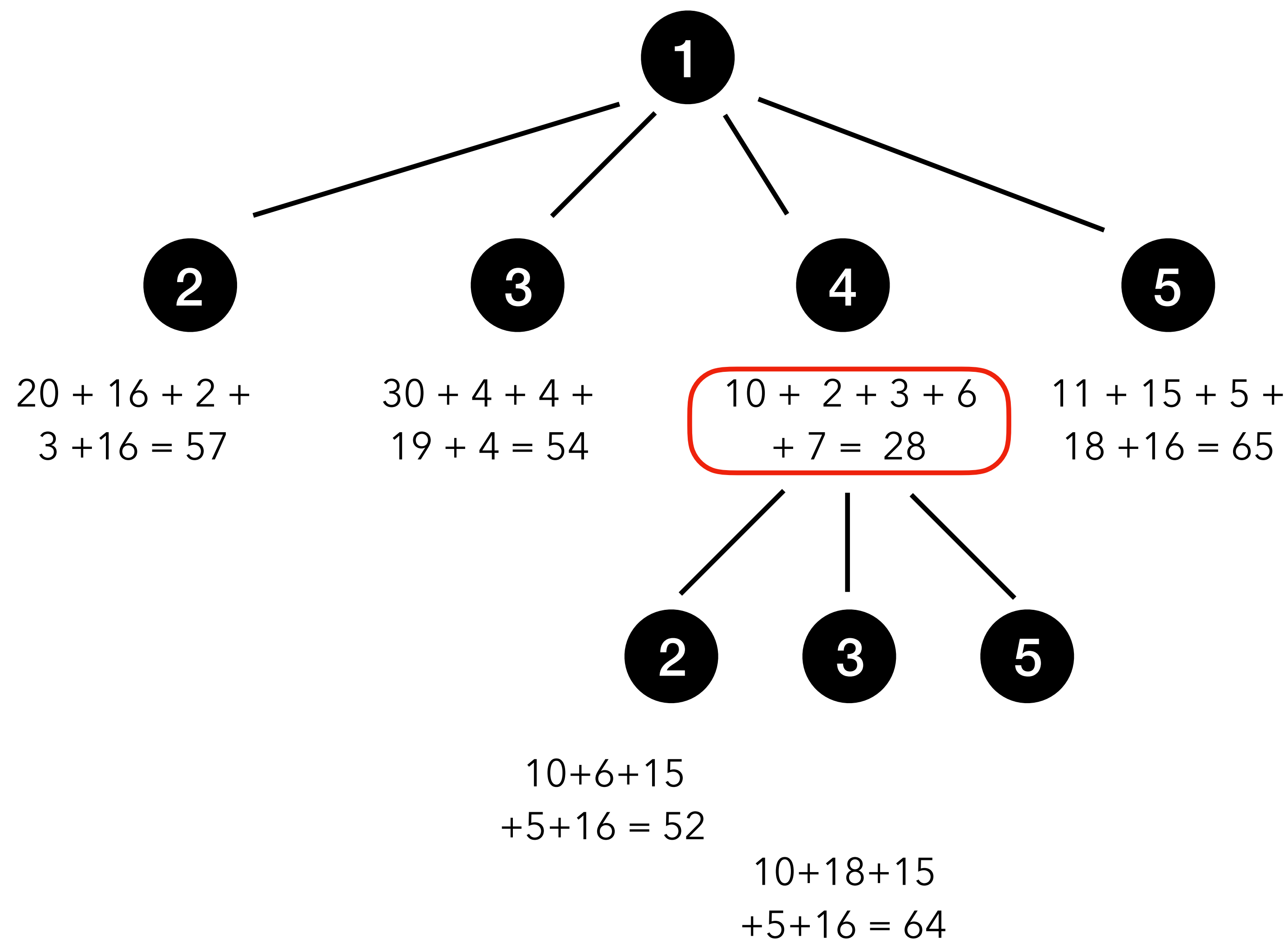
$$\begin{bmatrix} \infty & \infty & \infty & \boxed{10} & \infty \\ 15 & \infty & 16 & 4 & 2 \\ 3 & \infty & \infty & 2 & 4 \\ \infty & \boxed{6} & \infty & \infty & \infty \\ 16 & \infty & 7 & 16 & \infty \end{bmatrix}$$



Ramificació i poda

Cota superior = 20 +16 + 2 + 3 + 16 = 57

$$\begin{bmatrix} \infty & \infty & \infty & \boxed{10} & \infty \\ 15 & \infty & \infty & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ \infty & \infty & \boxed{18} & \infty & \infty \\ 16 & 4 & \infty & 16 & \infty \end{bmatrix}$$

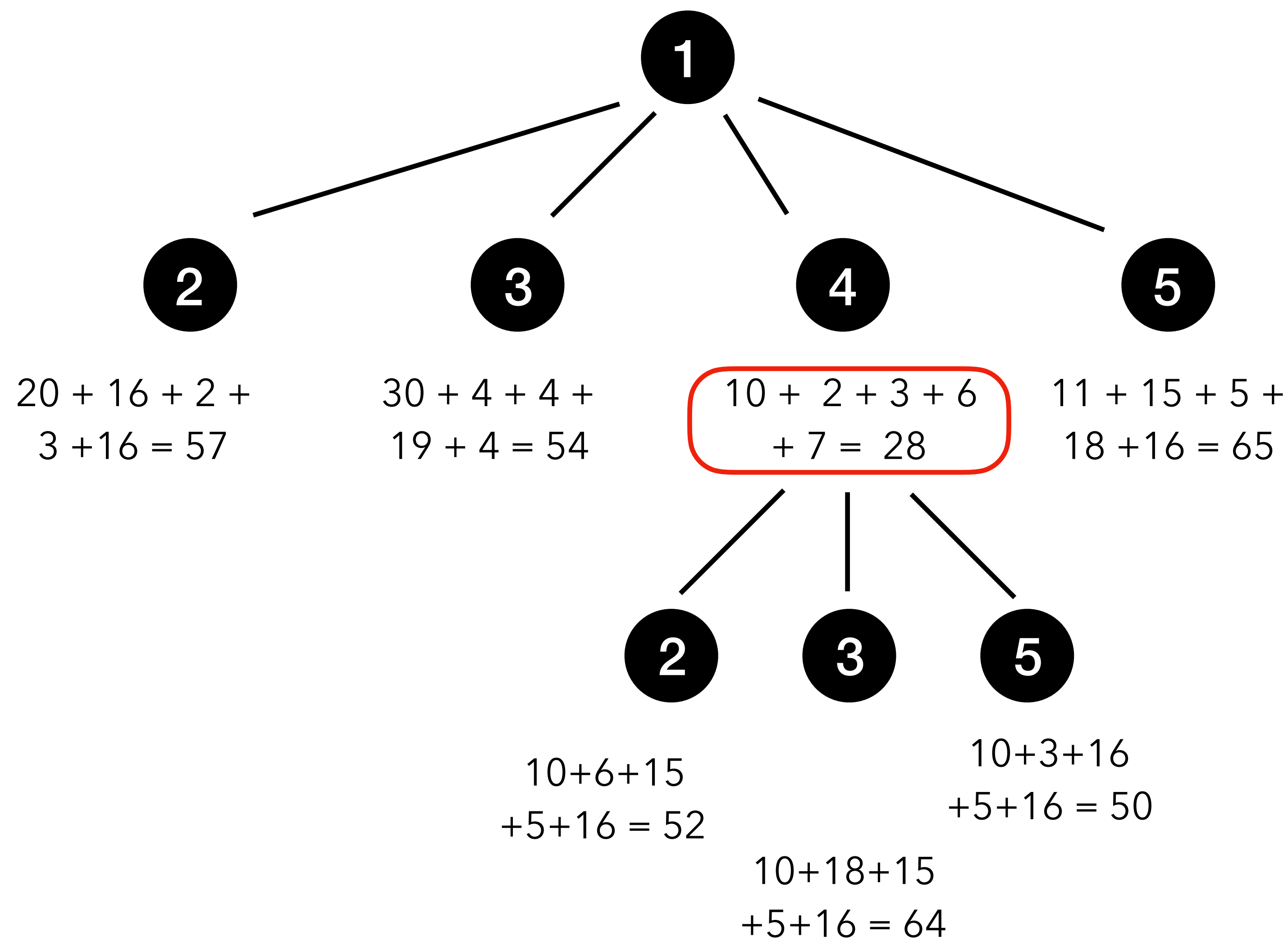


Ramificació i poda

Cota superior = 20 +16 + 2 + 3 + 16 = 57

$$\begin{bmatrix} \infty & \infty & \infty & \boxed{10} & \infty \\ 15 & \infty & 16 & 4 & \infty \\ 3 & 5 & \infty & 2 & \infty \\ \infty & \infty & \infty & \infty & \boxed{3} \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Upper Bound



Upper Bound

Una solució mitjançant la Matriu Reduïda

Ramificació i poda

- Si s'escull *t* com el mínim dels elements de la fila (columna) i-essima i es resta *t* de tots els elements de la fila (columna), la fila resultant (columna) es reduïda.
- La reducció de la matriu s'aconseguix fent la reducció per files i columnes de la matriu.
- La quantitat total ***L*** restada de files i columnes és una **cota inferior** de la longitud d'un hamiltonià de longitud mínima.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Reducció de la matriu,
L = 25

Ramificació i poda

- Càlcul de la cota inferior per al nodes diferents de la rel i de les fulles:
  - Sigui A la matriu de distàncies reduïda per al **node *y***.
  - Sigui ***x*** un fill de ***y*** que correspongui incloure l'aresta (***i***,***j***) en el recorregut i que no sigui fulla.
  - La matriu B reduïda per x, i per tant la cota(x), es calcula de la següent forma:

1. Canviar tots els elements de la fila ***i*** i de la columna ***j*** de ***A*** por ∞.

   Així evitem incloure més arestes que surtin de **i** o arribin a **j**

2. Canviar l'element (***j***,**1**) de ***A*** por ∞.

   Això evitat considerar l'arc (**j**,**1**).

3. *B* es la matriu que s'obté al reduir totes les files i columnes de la matriu resultant (excepte d'aquelles formades únicament per "∞").

   Si *r* es el valor total rest en el pas (3): cota(x)=cota(y) + D[i,j] + r

Ramificació i poda

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

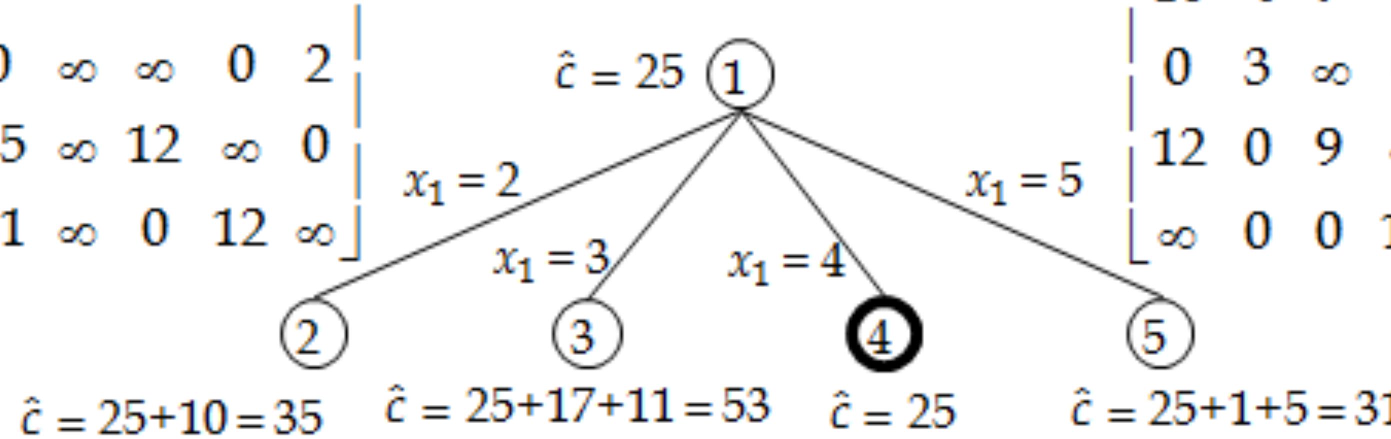
Grafo original.

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Matriz reducida, $L = 25$.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

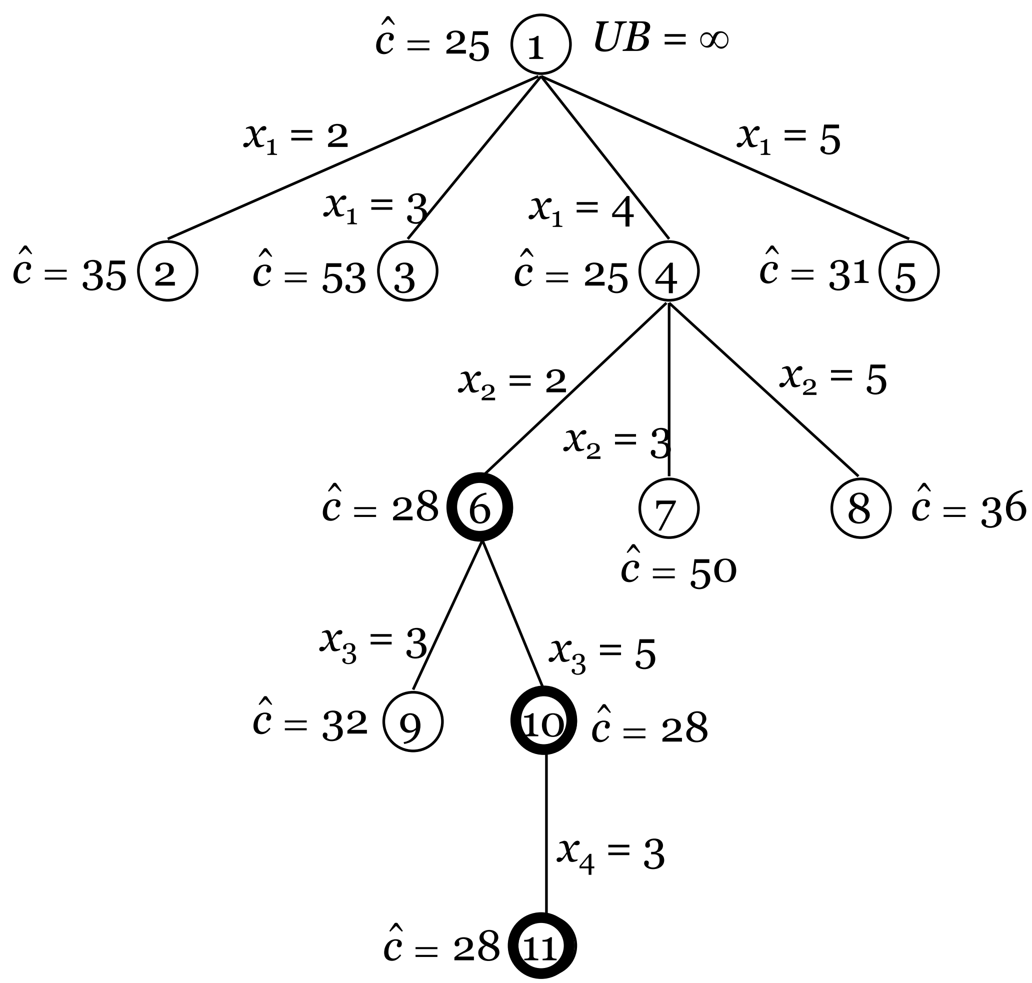
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$



$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Ramificació i poda



El següent node en curs seria el 5,
però cost(5)>UB
per tant l'algoritme termina i el
hamiltonià mínim es
1,4,2,5,3,1

Es una fulla (solució)
S'actualitza UB = 28