UNIVERSITAT DE BARCELONA

Enumeratius

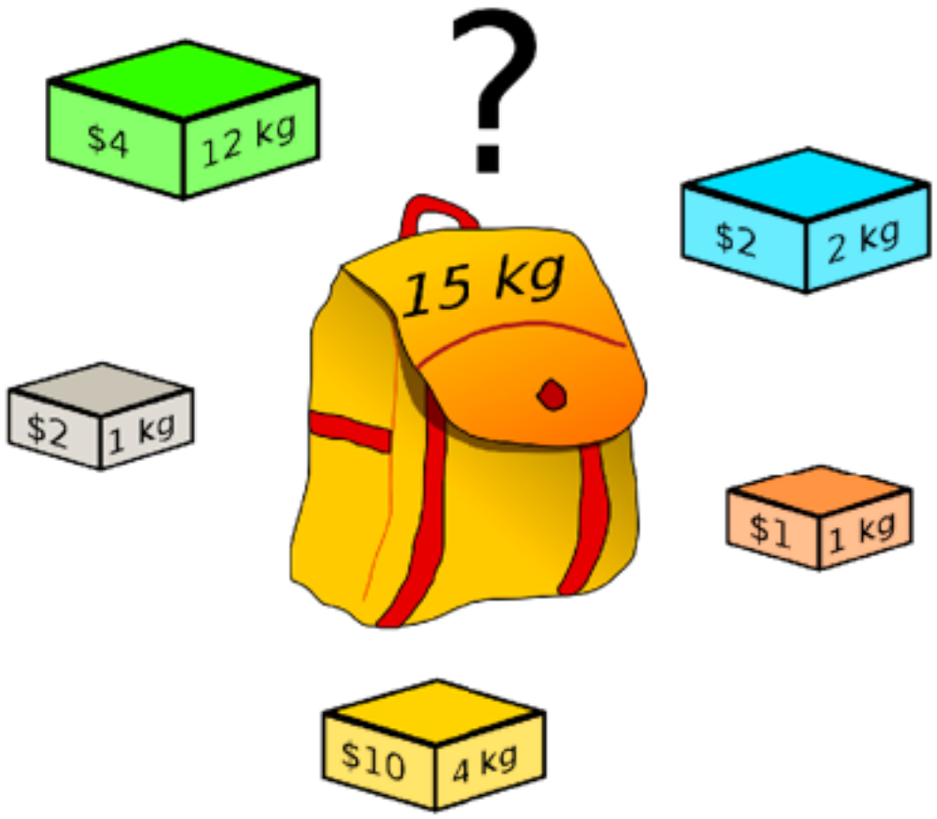
Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2019-2020

Enumeratius

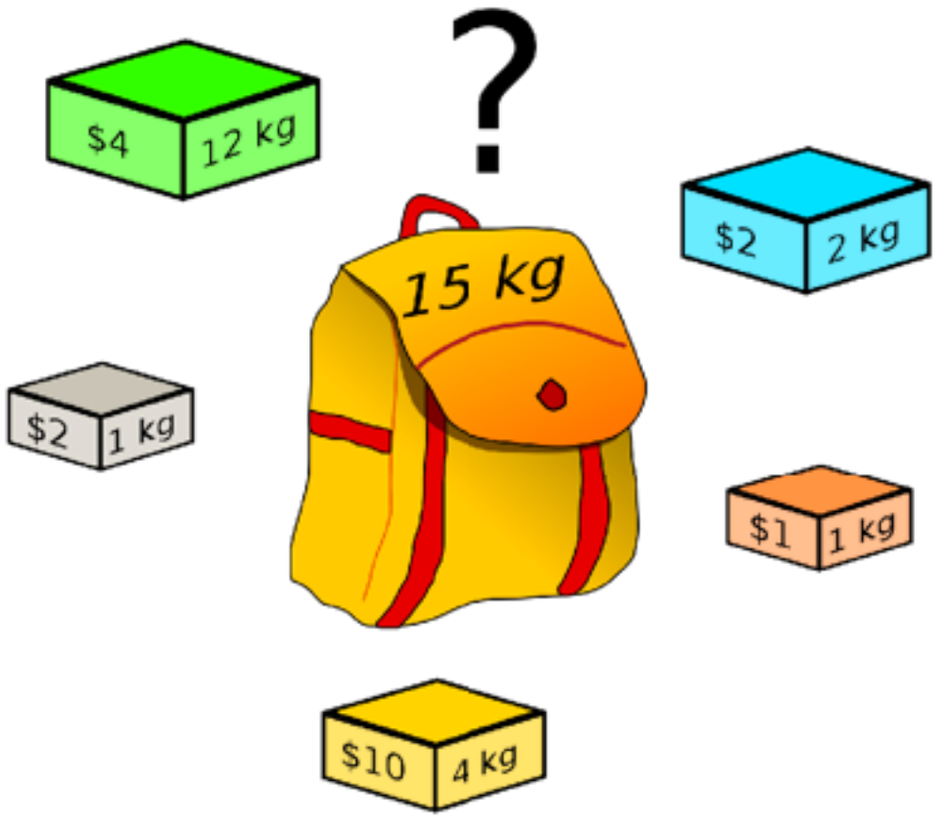
In computer science, an ***enumeration algorithm*** *is an algorithm that enumerates the answers to a computational problem. Formally, such an algorithm applies to problems that take an input and produce a list of solutions, similarly to function problems. For each input, the enumeration algorithm must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

Problema de la motxilla



Quines solucions hem vist?

Problema de la motxilla



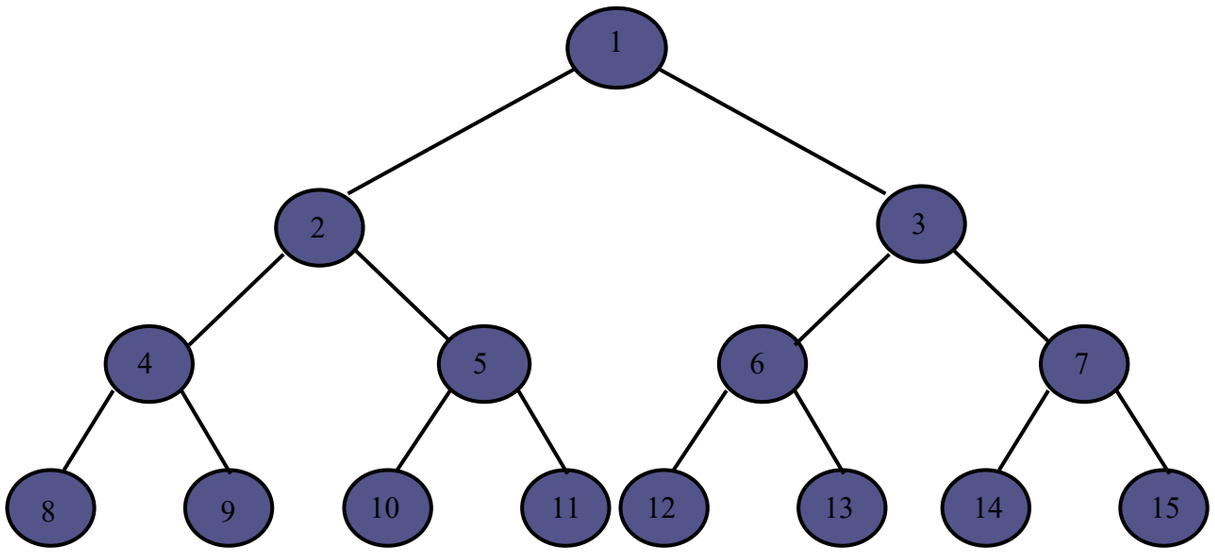
Quines solucions hem vist?

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)

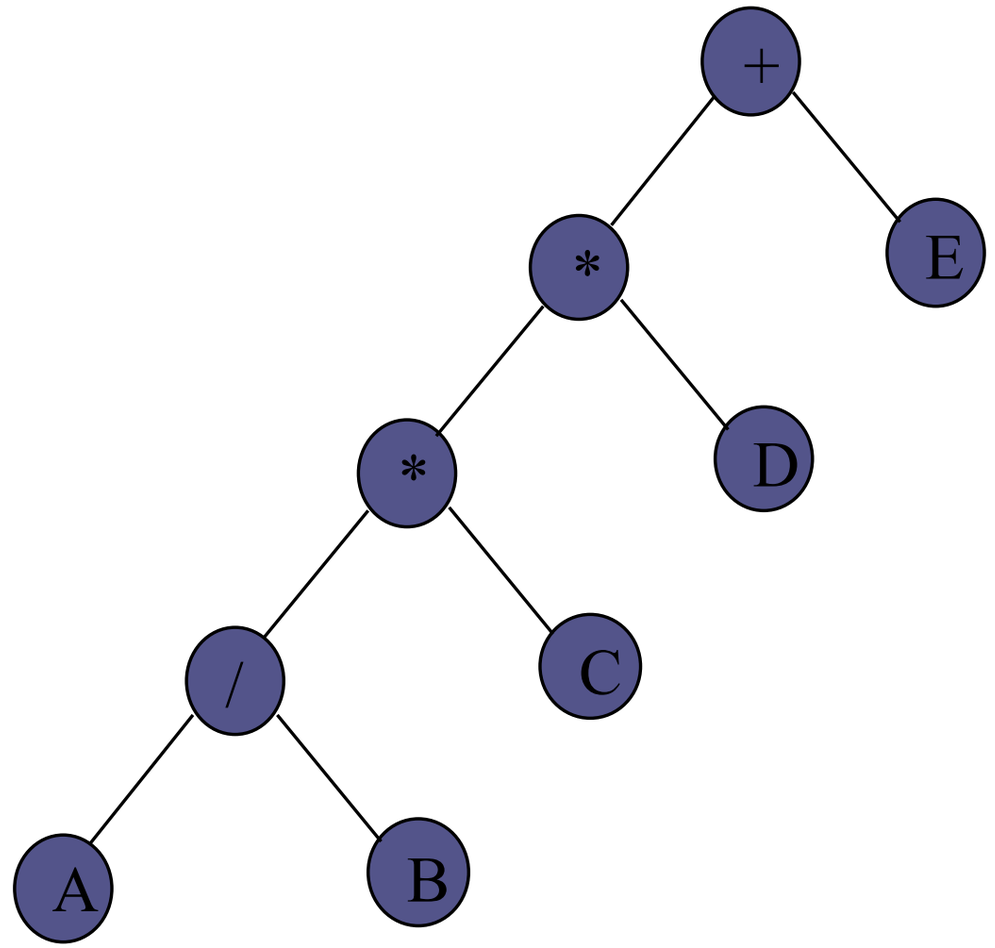
Cerca - Algorítmes amb arbres

- 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.

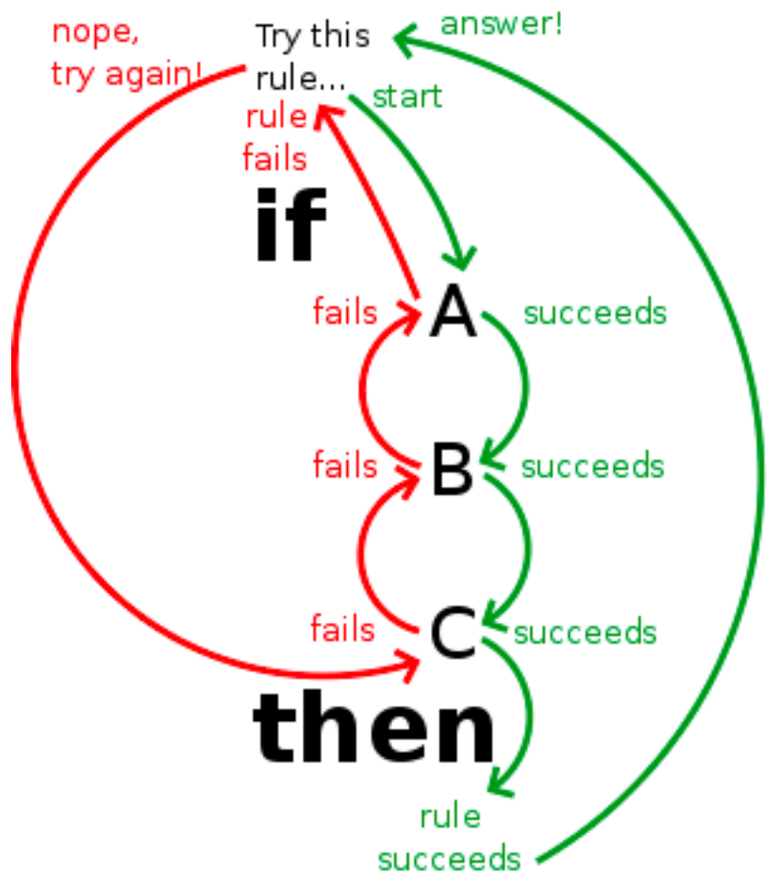
- Donat l'arbre binari, els diferents recorreguts porten a les següents formes d'escriure l'expressió:
  - Inordre LVR : A/B*C*D+E
  - Preordre VLR : +**/ABCDE
  - Postordre LRV : AB/ C*D*E+



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

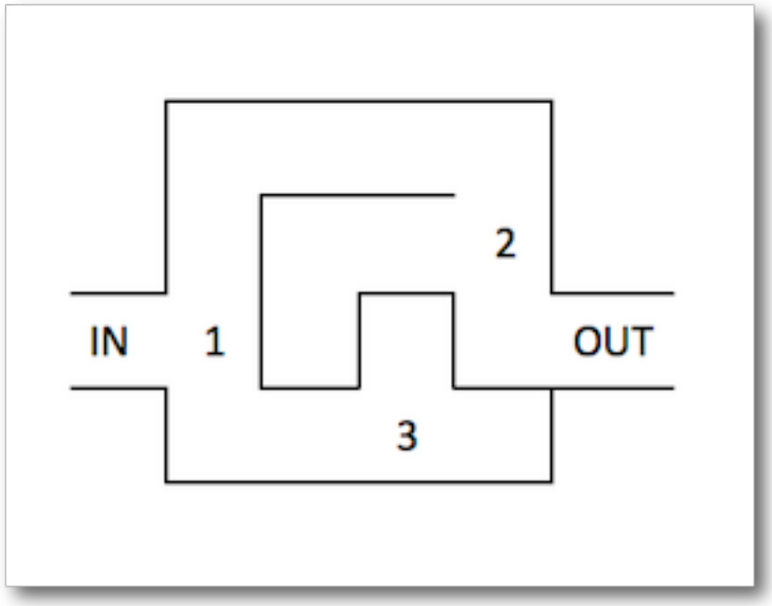


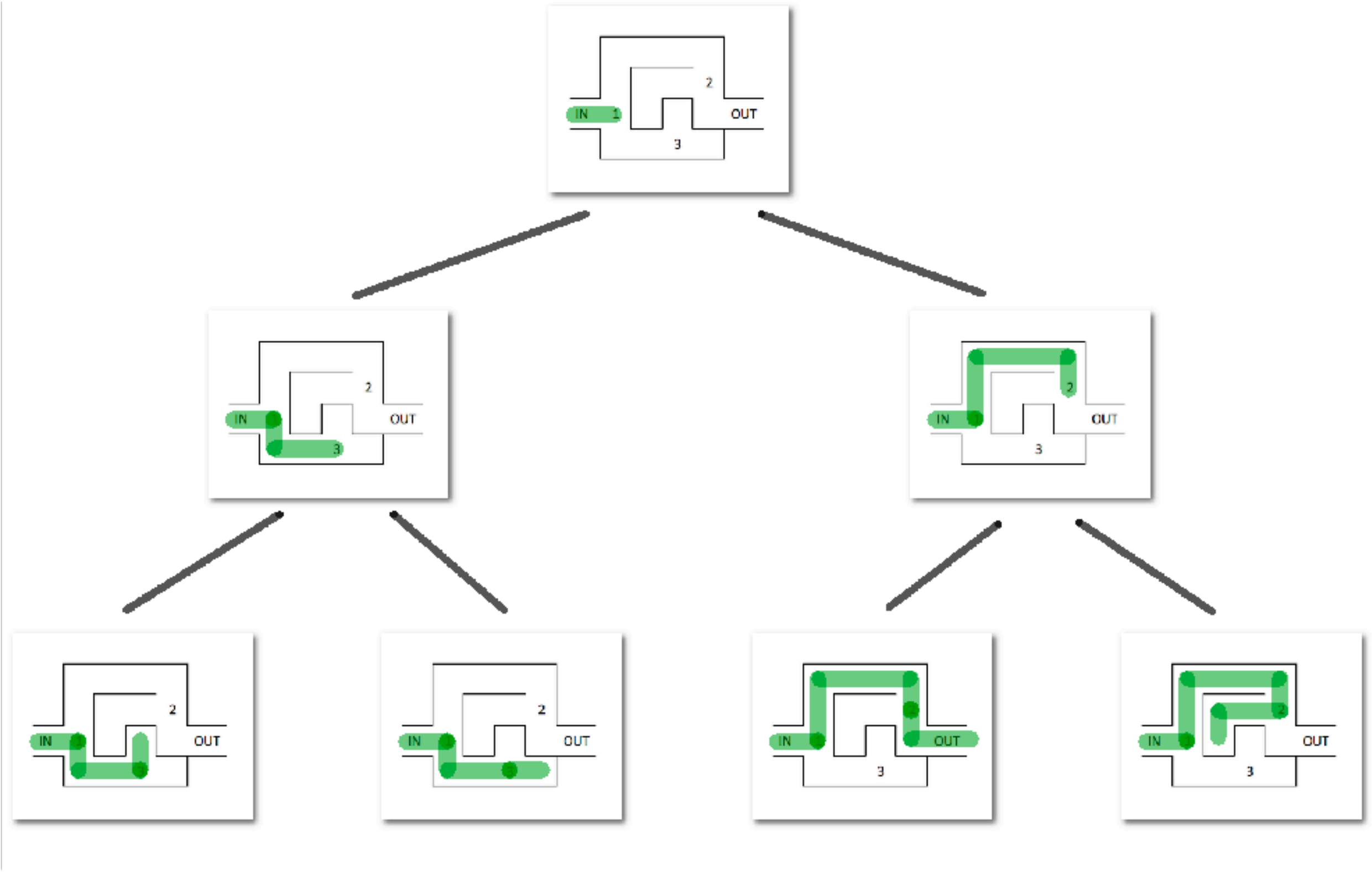
Backtracking

- 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.

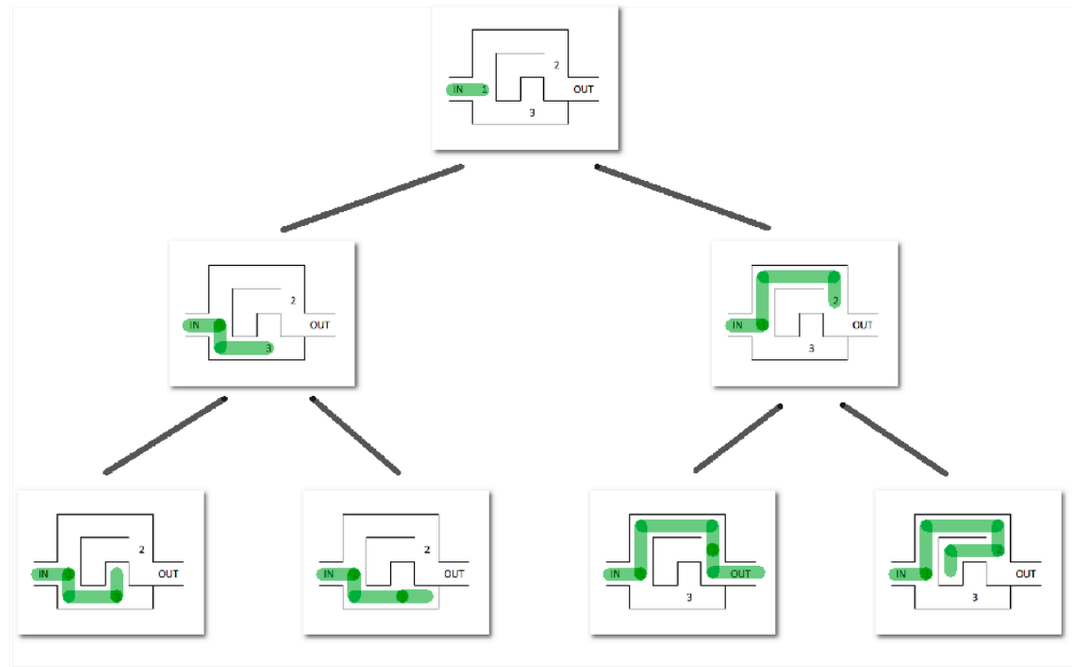




Tots els camins possibles

```
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

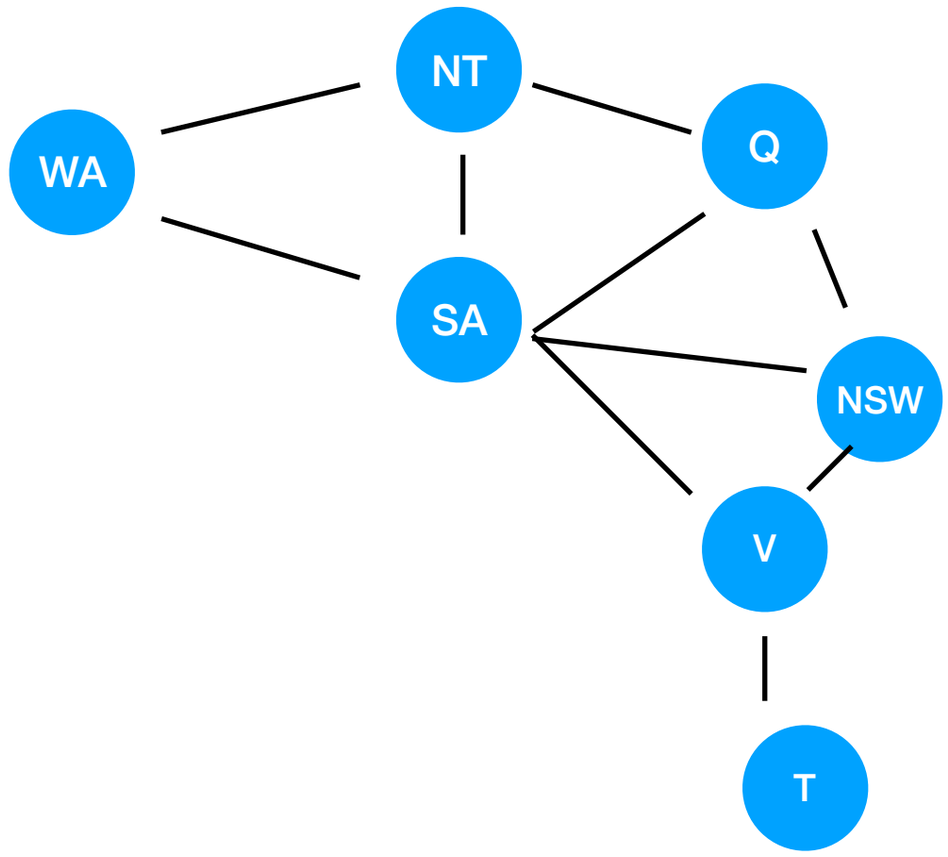
- Els esquemes de backtracking que veurem són directament aplicables a qualsevol tipus de graf (en molts exemples suposarem que són arbres)

```
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

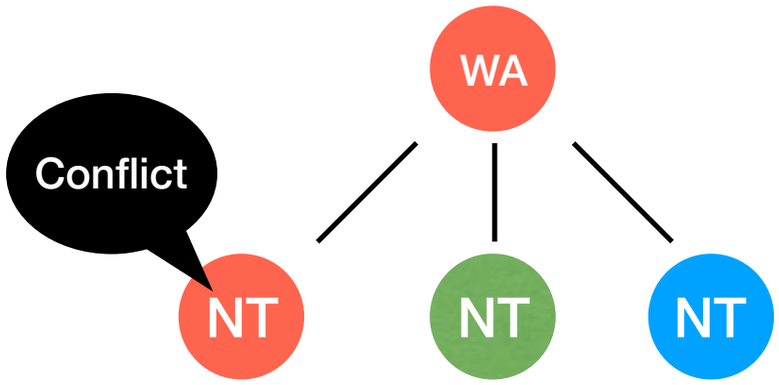
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



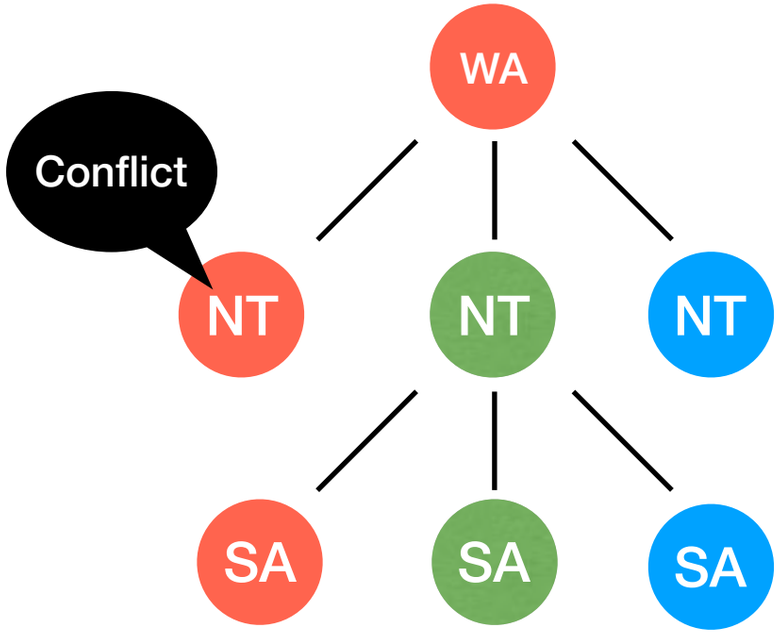
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



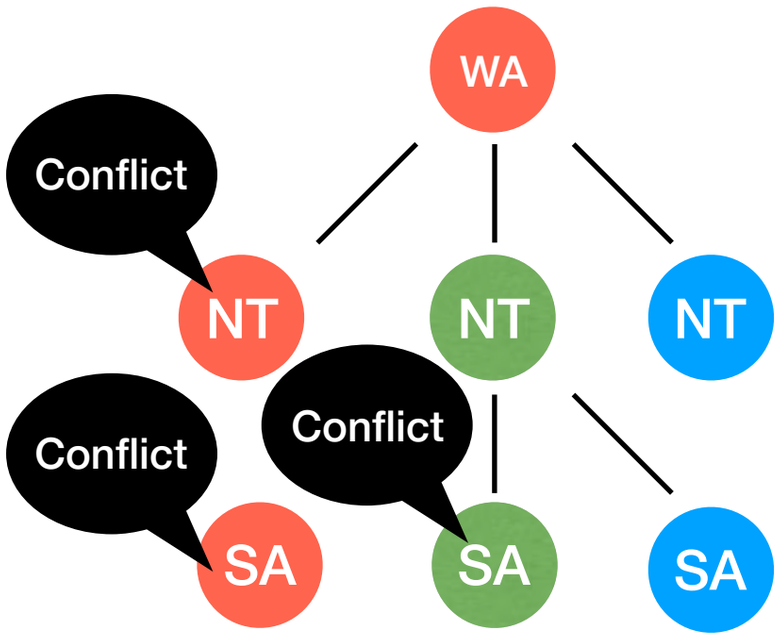
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



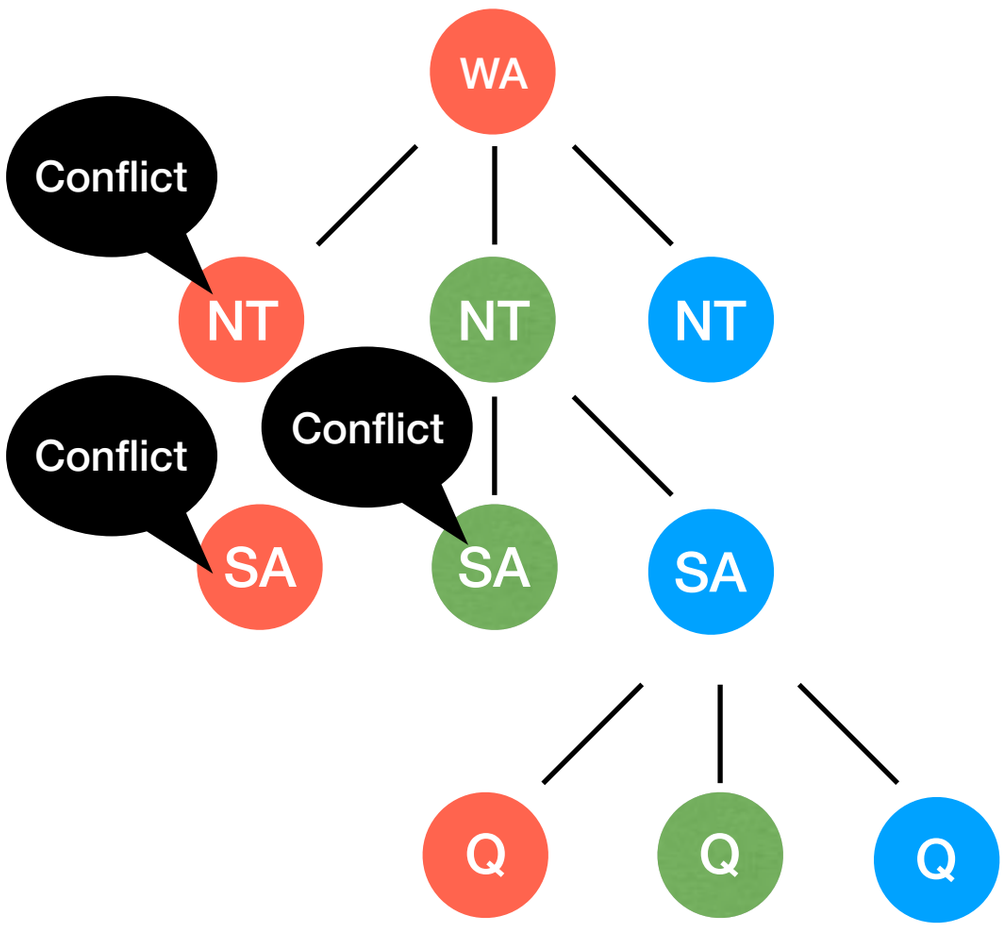
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

Exercici: Pinta el mapa



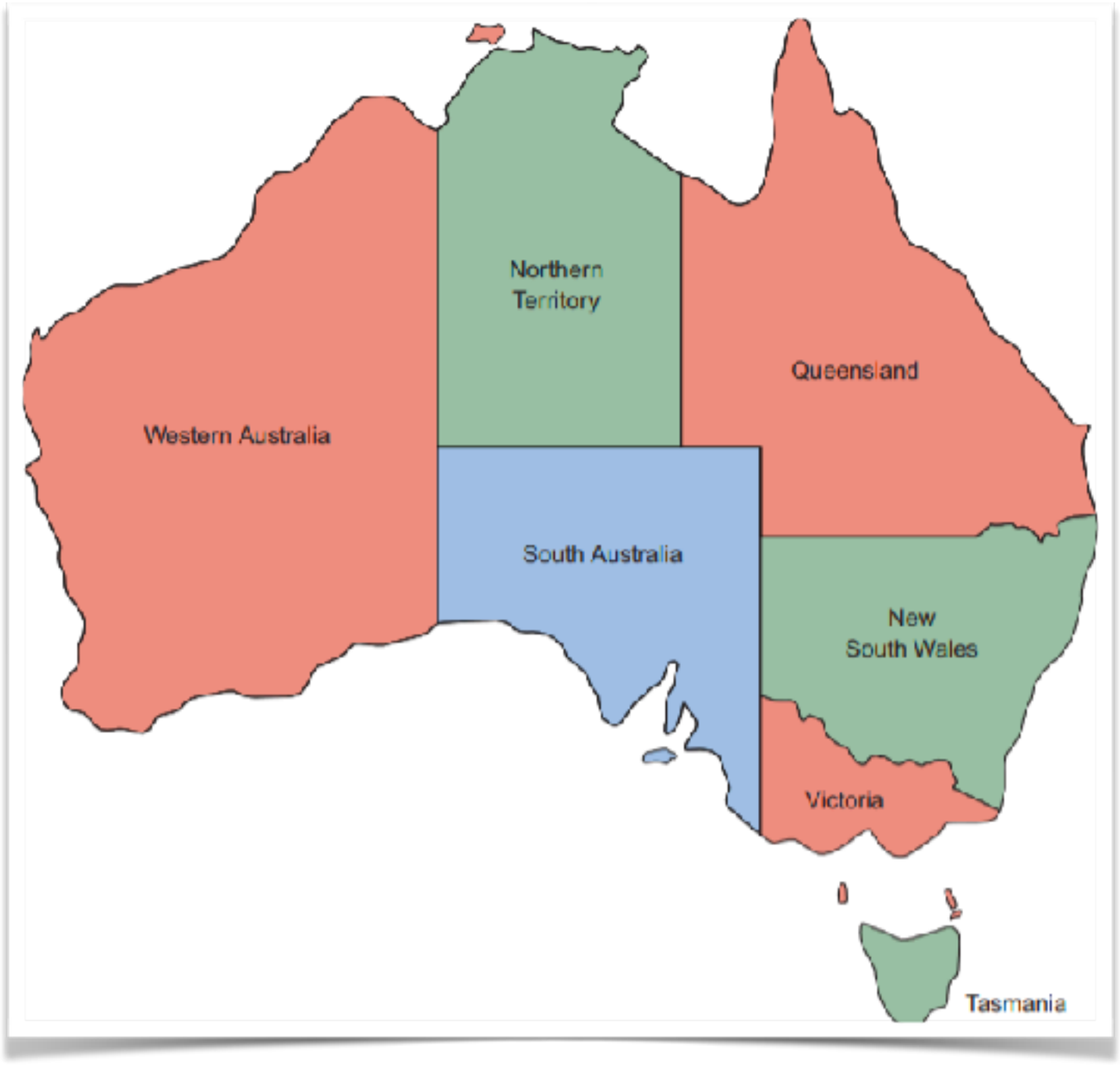
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color

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                          <-- Solution found
    for each possible moves
    if(this move is valid)
        select this move and place
        call backtrack()                     <-- Keep exploring
        unplace that selected move
        increment the given choice in the for loop
    else
        return False                         <-- 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

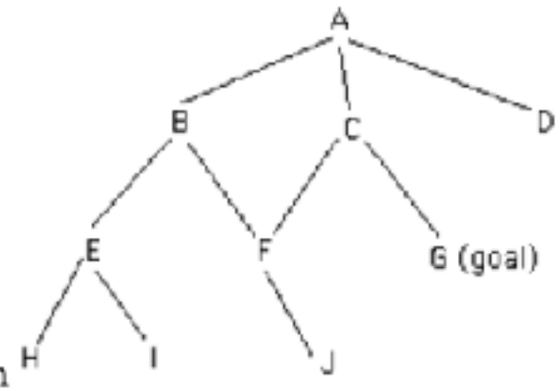
- The backtrack algorithm uses three lists plus one variable

  - **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.

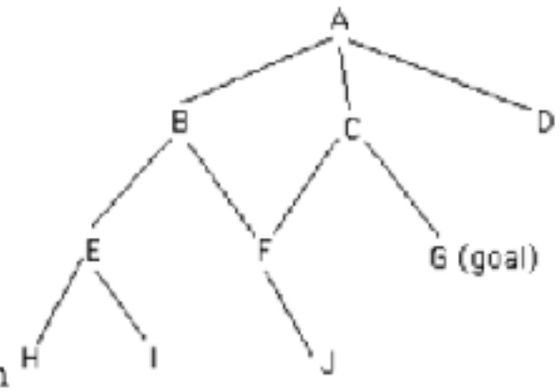


A Trace of Backtrack Algorithm

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

Backtracking

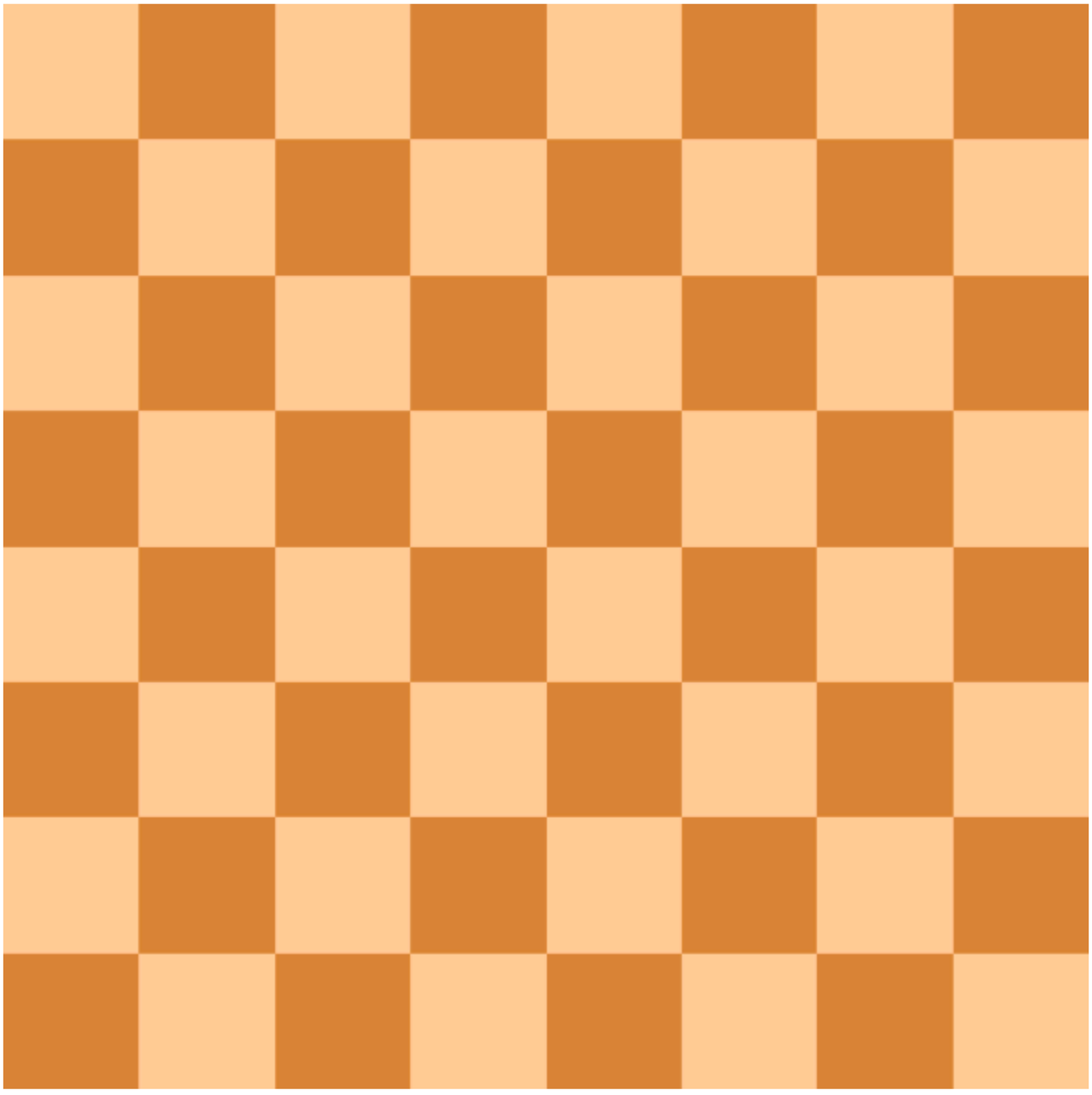
- The backtrack algorithm uses three lists plus one variable
  - **SL**, the state list, lists the states in the current path being tried. If a goal is found, SL contains the ordered list of states on the solution path.
  - **NSL**, the new state list, contains nodes awaiting evaluation -- i.e., nodes 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:

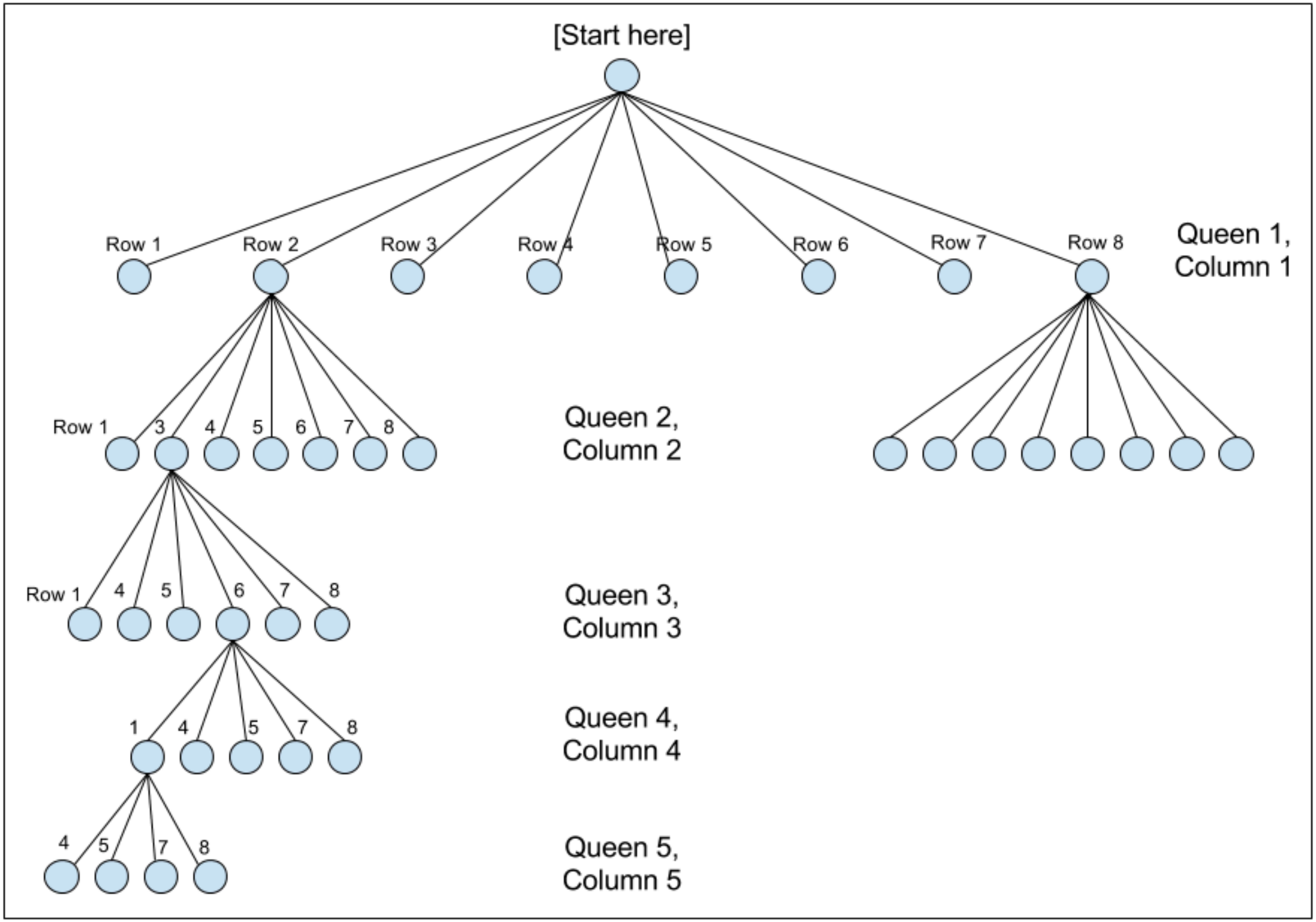
- Solució per força bruta?

  - 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

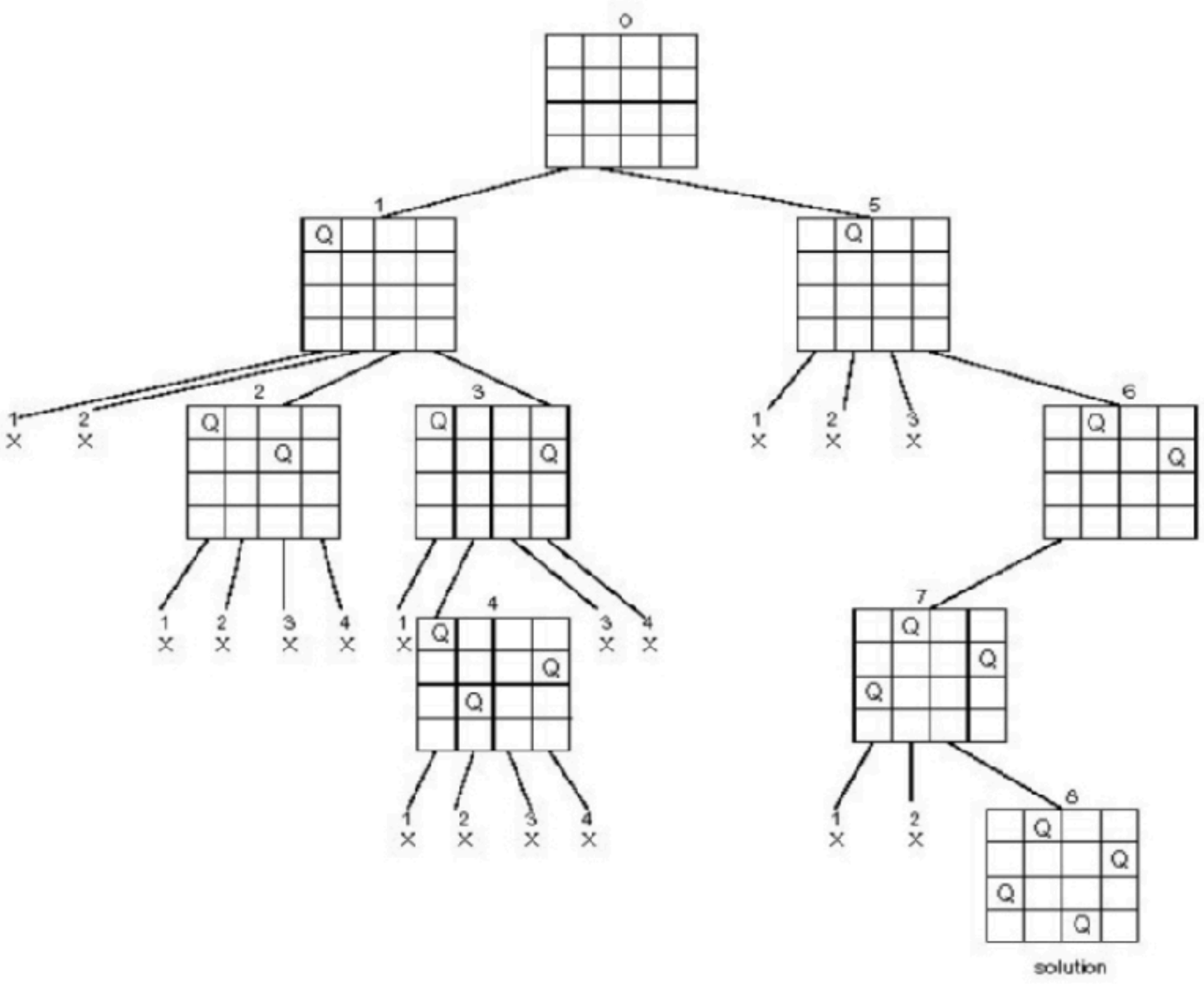
```
bool ocho_reinas()
{
    for (int i=1; i<=8; i++)
        for (int j=1; j<=8; j++)
            for (int k=1; k<=8; k++)
                for (int l=1; l<=8; l++)
                    for (int m=1; m<=8; m++)
                        for (int n=1; n<=8; n++)
                            for (int o=1; 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


0
1
Q
1
X
2
X
2
Q
Q
1
X
2
X
3
X
4
X
3
Q
Q
1
X
2
X
3
X
4
X
4
Q
Q
Q
1
X
2
X
3
X
4
X
5
Q
1
X
2
X
3
X
6
Q
Q
7
Q
Q
Q
1
X
2
X
8
Q
Q
Q
Q
solution