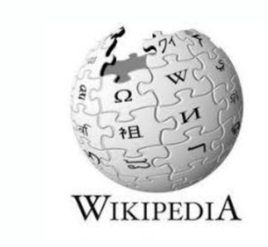
UNIVERSITAT DE BARCELONA

Algoritmes Enumeratius

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | **2025-2026**



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*

Enumeratius

- Recorregut vs. Cerca
- Backtracking
- Ramificació i poda

Enumeratius

- Tenim diverses tècniques per tal de trobar aquestes solucions:

  - **El backtracking**: El backtracking és una tècnica algorítmica que ens permet resoldre problemes de forma recursiva, intentant construir una solució de manera incremental, eliminant totes aquelles solucions parcials, (retrocedint/backtrack) que no compleixen les limitacions del problema, tan aviat com es determina que la solució no es pot completar com a solució vàlida/ òptima.

  - **Ramificació i poda (Branch and Bound)**: Un procediment enumeratiu basat en ramificació i poda requereix definir una funció d'avaluació per a cada node, per així **seleccionar** en cada pas quin és el **millor node a explorar**, i per altra banda eliminar certs nodes a ser explorats (aquells que no arribaran a la solució òptima). L'eficiència del mètode anirà fortament lligat a la funció que avalua cada node.

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

- 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

Backtracking

- El tres punts fonamentals del **backtracking**:
  - **Eleccions –>** Tenim diverses opcions a seleccionar
  - **Restriccions –>** Tenim diverses restriccions sobre les eleccions possibles
  - **Objectiu –>** Volem convergir a una solució

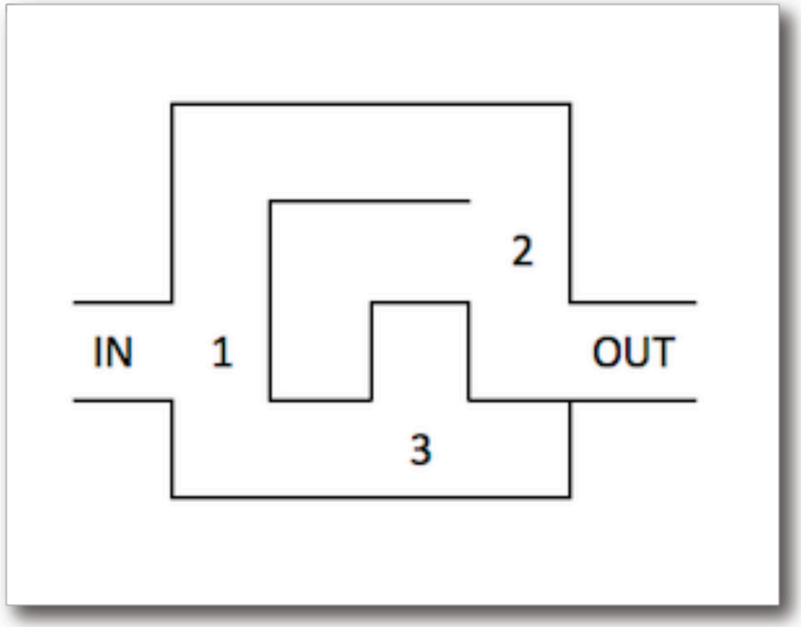
Algoritme general de backtracking

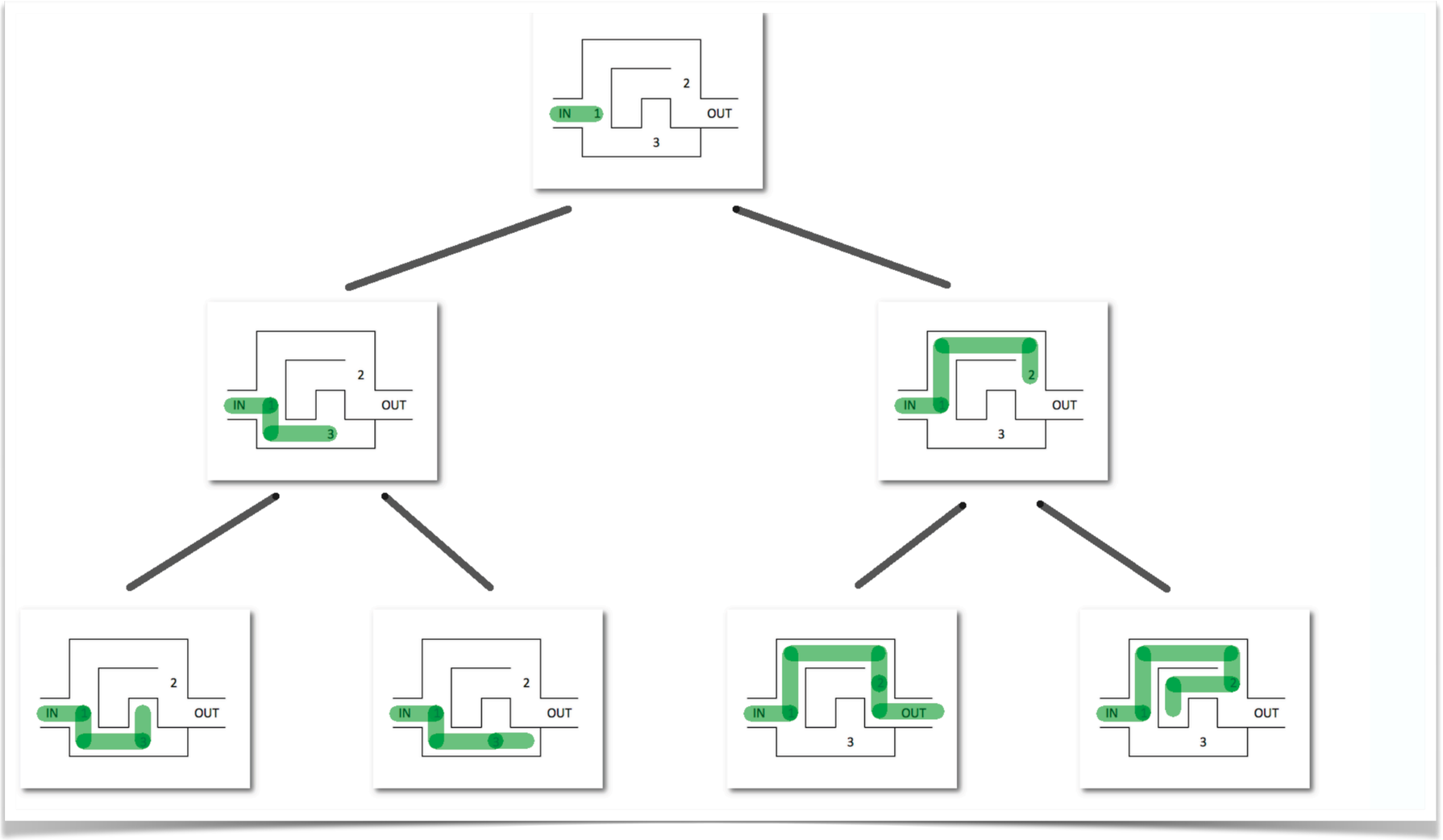
```
algorithm backtrack():
    if (solution == True)
        return True                          <-- Solution found
    for each possible moves
        if(this move is valid)
            select this move and place
            ok = call backtrack()            <-- Keep exploring
            if ok:
                return solution found
            unplace that selected move
    return False                             <-- Don't explore anymore!
                                                 no solution in this path
```

La idea és que podem ***construir una solució pas a pas utilitzant la recursivitat****; si durant el procés ens adonem que* ***no*** *serà una* ***solució vàlida****, aleshores deixem de calcular aquesta solució i* ***retrocedim*** *al pas anterior (backtrack).*

Veiem un exemple

Trobar la sortida del laberint.




IN
1
2
3
OUT

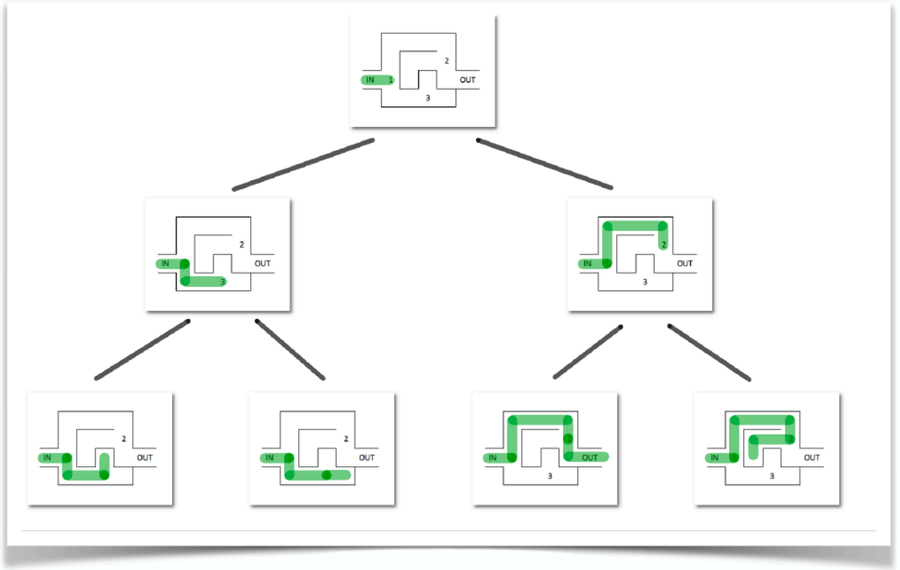
```
algorithm backtrack():
  if (solution == True)
    return True                          <-- Solution found
  for each possible moves
    if(this move is valid)
      select this move and place
      ok = call backtrack()              <-- Keep exploring
      if ok:
          return solution found
      unplace that selected move
  return False                           <-- Don't explore anymore! no solution in this path
```

```
def backtrack(junction):
    if is_exit(junction):
        return True

    for each direction of junction:
        if(backtrack(nex_junction)):
            return True
    return False
```

```python
def backtrack(junction):
    if is_exit(junction):
        return True

    for each direction of junction:
        if(backtrack(nex_junction)):
            return True
    return False
```



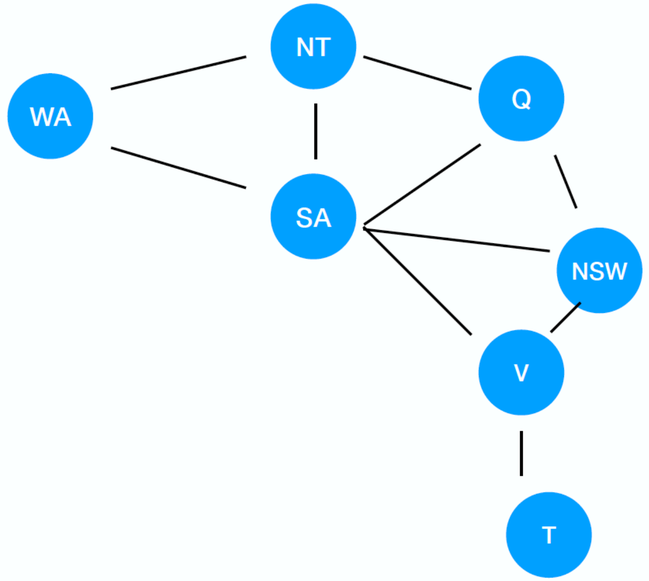
If we apply this pseudo code to the maze we saw above, we'll see these calls:

```
- 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)
```

Exercici: Pintar el mapa

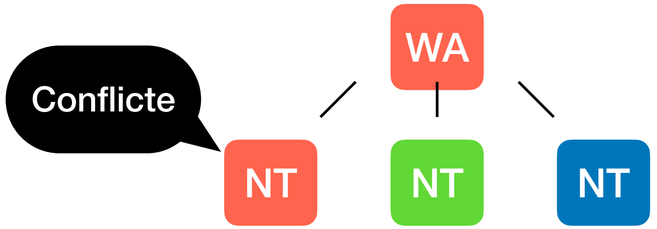
Pina el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color.

Exercici: Pintar el mapa



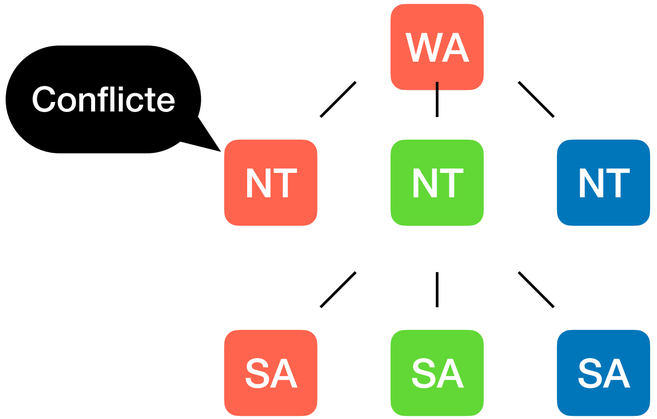
Pina el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color.

Exercici: Pintar el mapa



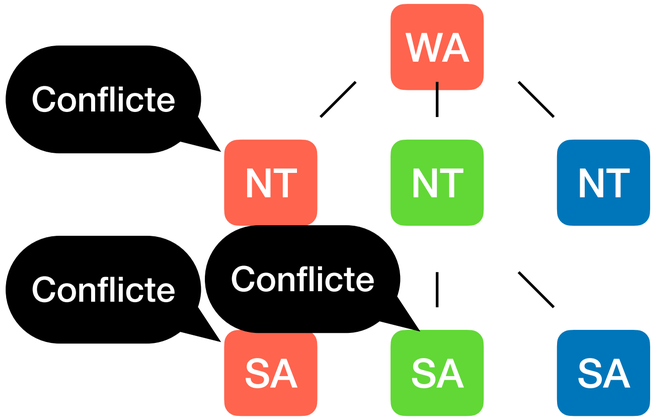
Pina el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color.

Exercici: Pintar el mapa



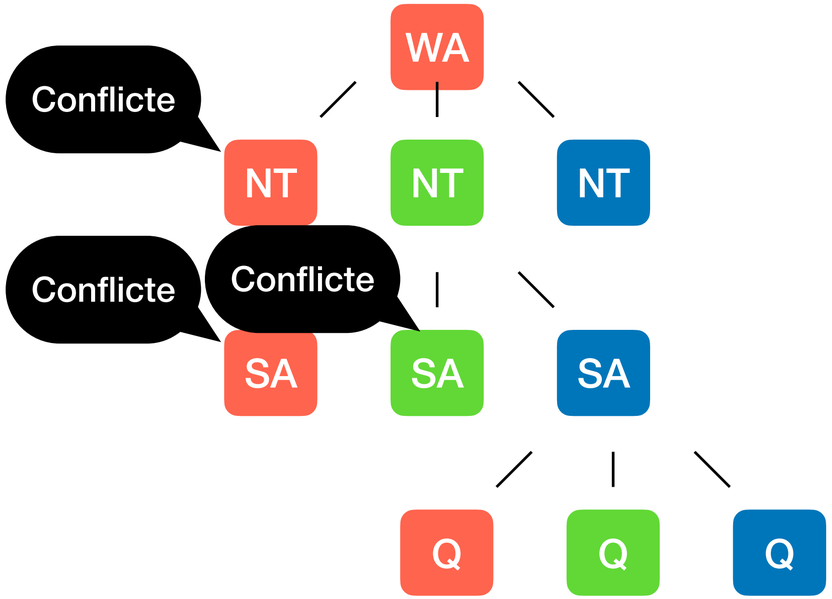
Pina el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color.

Exercici: Pintar el mapa



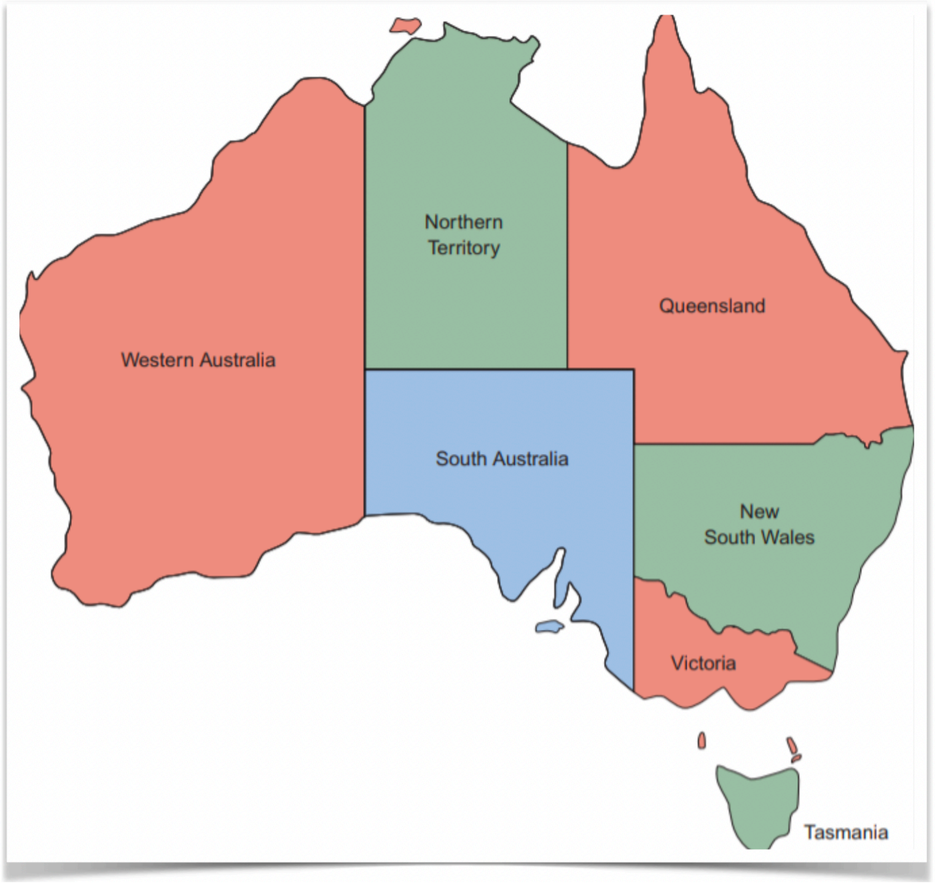
Pina el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color.

Exercici: Pintar el mapa



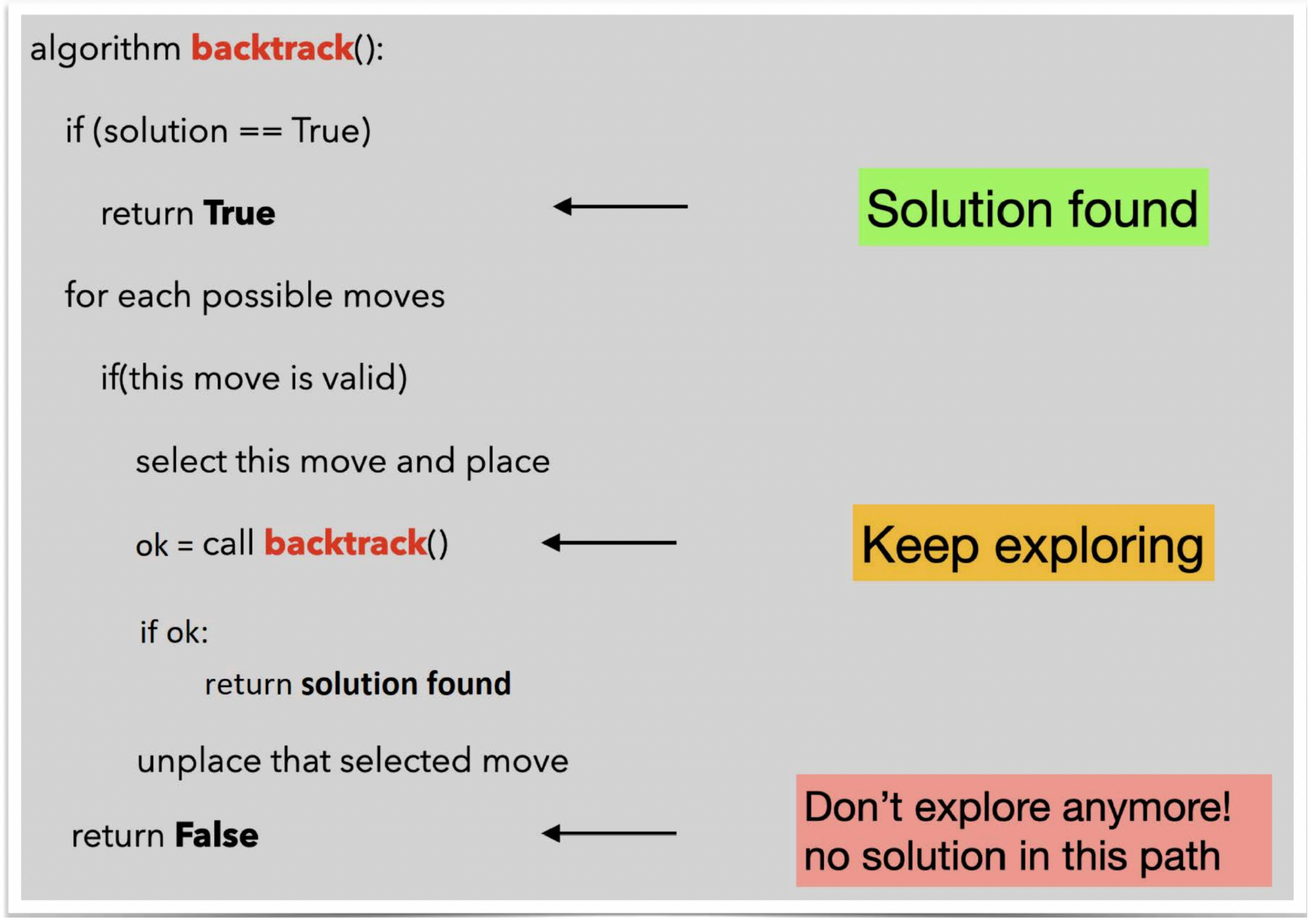
Pinta el mapa amb màxim 3 colors
on cap estat adjacent tingui el mateix color.

Exercici: Pintar el mapa



Exercici: Pintar el mapa

- **Pensem una solució**



Exercici: Pintar el mapa

```
def map_painting(city_colors, cities, adjMatrix, index, colors):
    # if a solution has been found
    if(index== len(cities)):
        printSolution(city_colors)
        return True

    for c in colors:
        if(valid_movement(adjMatrix, city_colors, c, index)):
            city_colors[index] = c
            if(map_painting(city_colors, cities, adjMatrix, index+1, colors)):
                return True
            city_colors[index] = 0
    return False


colors = ['R','G','B']
cities = ['WA','NT','SA','Q','NSW','V','T']
city_colors = [0 for i in range(len(cities))]

adjMatrix = [[ 0, 1, 1, 0, 0, 0, 0],
             [ 1, 0, 1, 1, 0, 0, 0],
             [ 1, 1, 0, 1, 1, 1, 0],
             [ 0, 1, 1, 0, 1, 0, 0],
             [ 0, 0, 1, 1, 0, 1, 0],
             [ 0, 0, 1, 0, 1, 0, 1],
             [ 0, 0, 0, 0, 0, 1, 0]]
```

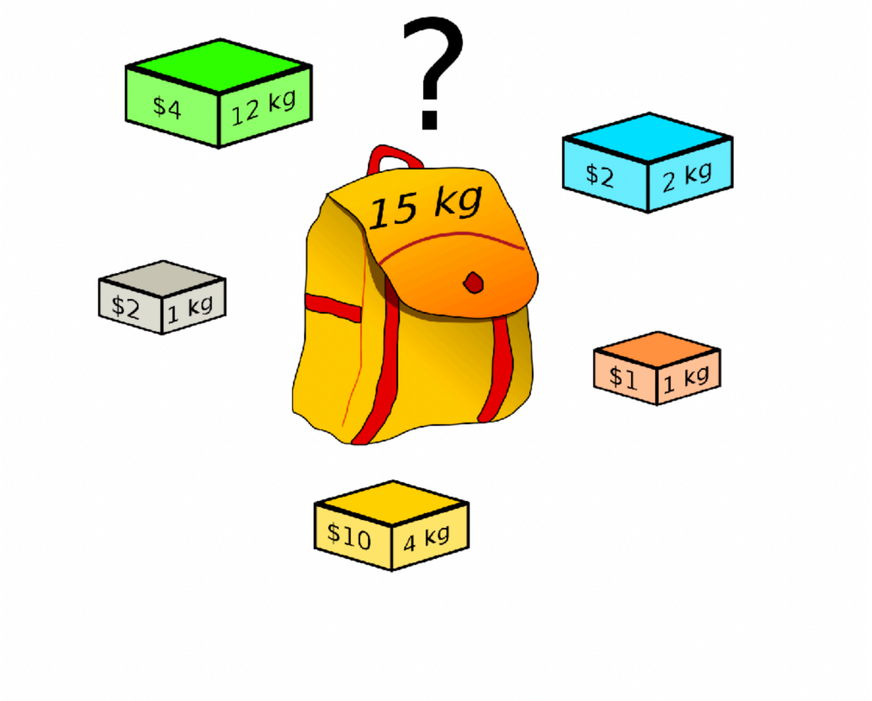
Exercici: Pintar el mapa

```
def printSolution(results):
    print("Solution Exists:" " Following are the assigned colors ")
    for i in range(len(results)):
        print(results[i],end=" ")


def check_if_solution_valid(adjMatrix, solution):
    for i in range(len(solution)):
        for j in range(i + 1, len(solution)):
            if (adjMatrix[i][j] and solution[j] == solution[i]):
                return False
    return True
```

Bracktracking

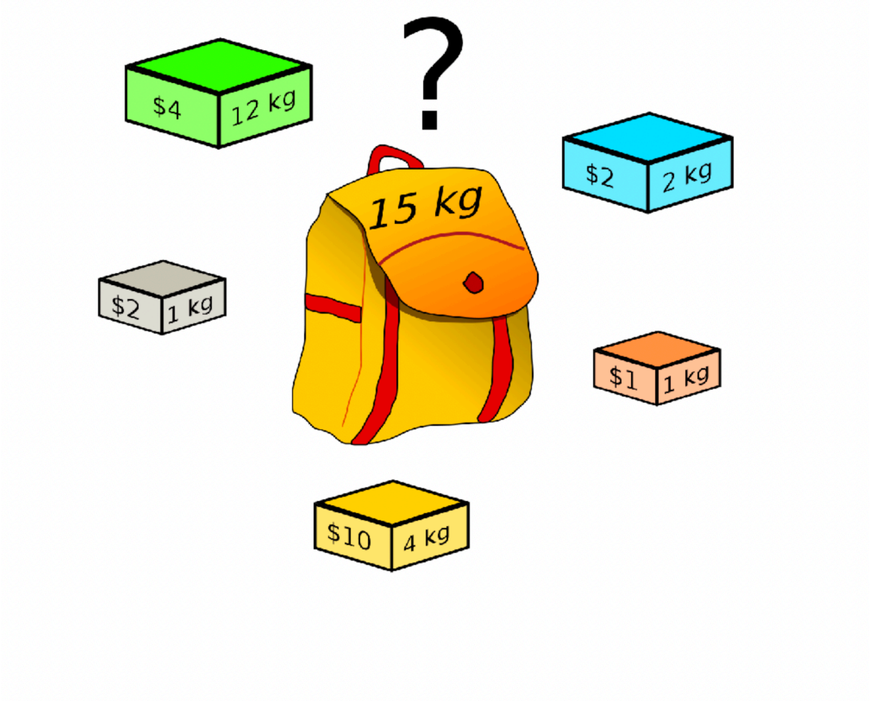
Problema de la motxilla



Quines solucions hem vist?

Bracktracking

Problema de la motxilla

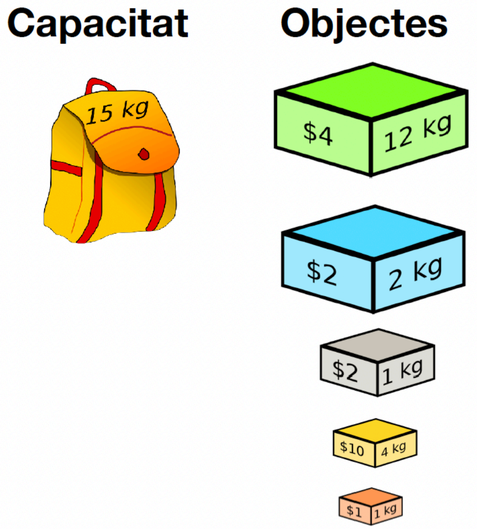


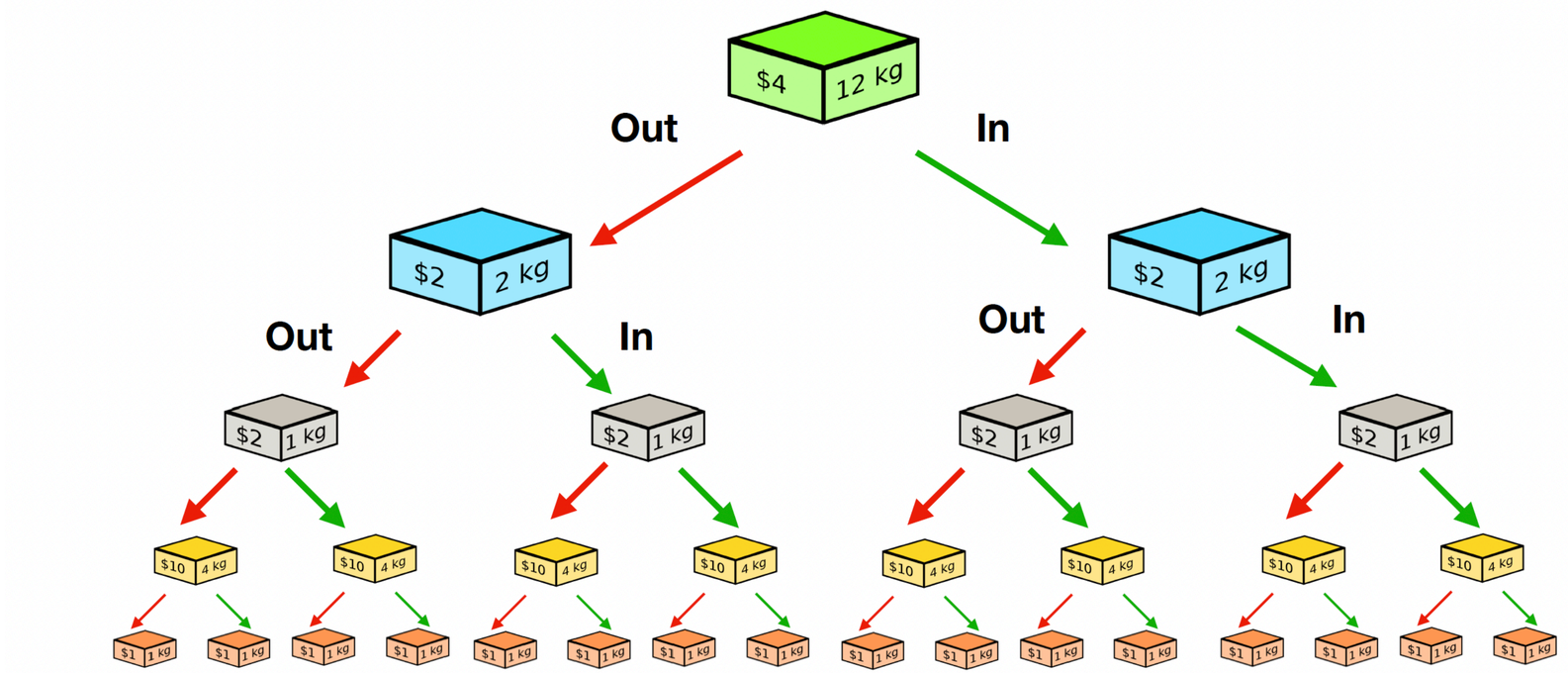
Quines solucions hem vist?

força bruta
greedy
programació dinàmica

Backtracking

Problema de la motxilla - Solució Exhaustiva





Amb la força bruta hauríem d'explorar totes les possibles solucions i avaluar-les així com és mostra a la imatge.

Backtracking

Problema de la motxilla

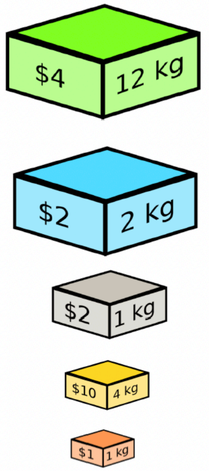
- Amb **backtracking** tenim tres punts importants:
    - Les **eleccions**.
    - Les **restriccions**.
    - **L'objectiu**.
- En el cas del problema de la motxilla. Ens queda d'aquesta manera:
    - Les eleccions: Incloure o no incloure l'objecte.
    - Les restriccions: No pot superar un cert pes.
    - L'objectiu: Els objectes seleccionats han de tenir el màxim valor.

Backtracking

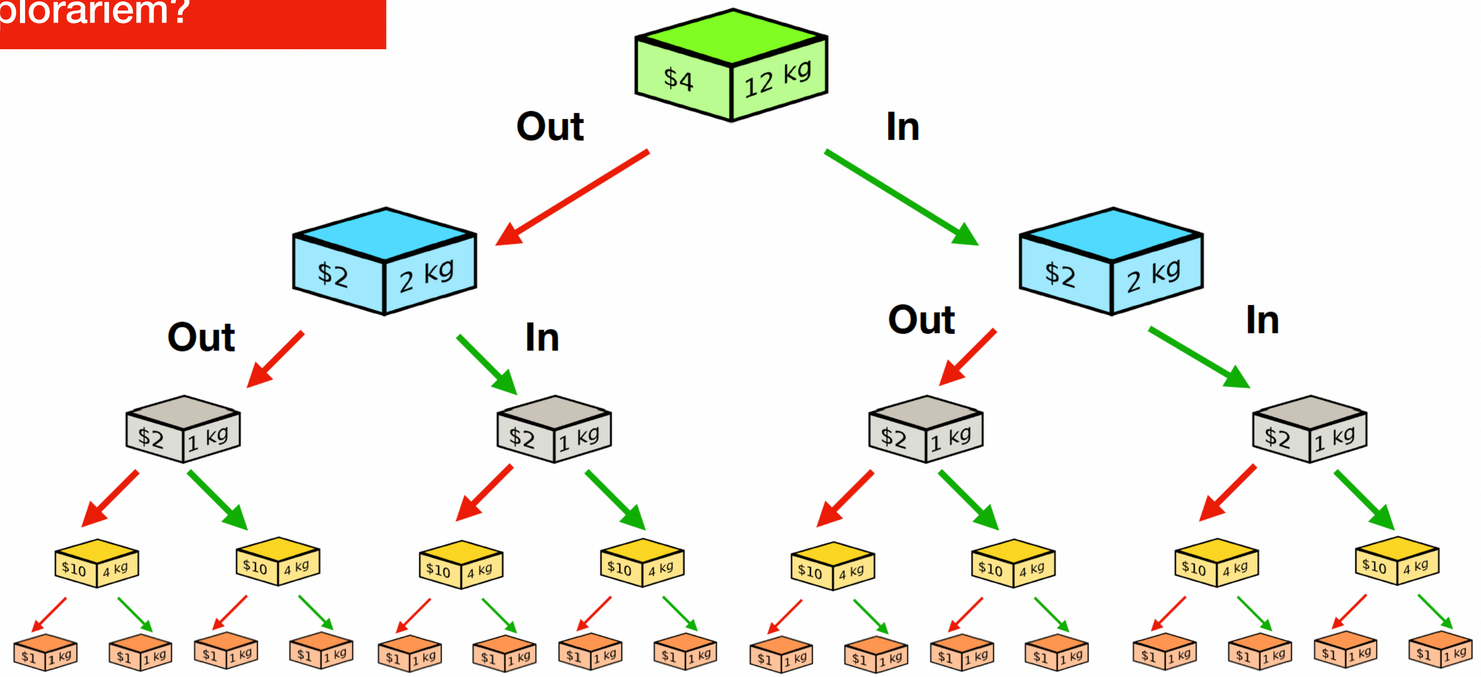
Problema de la motxilla - **Solució amb backtrackin**

Capacitat

Objectes



Quina part de l'arbre no exploriem?



Amb la força bruta hauríem d'explorar totes les possibles solucions i avaluar-les així com és mostra a la imatge.

```python
max_weight = 23
item_values = [16, 15, 4, 3, 2]
item_weights =[14, 13, 7, 2, 1]
num_items = len(item_weights)

def knapsack(max_weight, item_values, item_weights, num_items,
             current_weight, current_value, index, items,
             best_weight, best_value, best_items):
    # Hem trobat una solució.
    if index == num_items:
        # Si la solució trobada és millor que l'anterior, actualizem.
        if current_value > best_value:
            best_weight, best_value, best_items = current_weight, current_value, items
            print("SOLUTION: ",best_weight, best_value, best_items, items, index)
        return best_weight, best_value, best_items

    # Possibles moviments - > Posem l'ítem o no el posem
    for (add_weight, add_value, add_item) in  [(0, 0, -1), (item_weights[index], item_values[index], index)]:
        if current_weight + add_weight <= max_weight: # si el moviment es vàlid
            current_weight += add_weight
            current_value += add_value
            index += 1
            items.append(add_item)

            best_weight, best_value, best_items = knapsack(max_weight, item_values, item_weights, num_items,
                                                           current_weight, current_value, index, items,
                                                           best_weight, best_value, best_items)
            index -= 1
            current_value -= add_value
            current_weight -= add_weight
            items.pop()
    return best_weight, best_value, best_items

knapsack(max_weight, item_values, item_weights, num_items, 0, 0, 0, [], 0, 0, [])
```

```
SOLUTION:  1 2 [-1, -1, -1, -1, 4] [-1, -1, -1, -1, 4] 5
SOLUTION:  2 3 [-1, -1, -1, 3, -1] [-1, -1, -1, 3, -1] 5
SOLUTION:  3 5 [-1, -1, -1, 3, 4] [-1, -1, -1, 3, 4] 5
SOLUTION:  8 6 [-1, -1, 2, -1, 4] [-1, -1, 2, -1, 4] 5
SOLUTION:  9 7 [-1, -1, 2, 3, -1] [-1, -1, 2, 3, -1] 5
SOLUTION:  10 9 [-1, -1, 2, 3, 4] [-1, -1, 2, 3, 4] 5
SOLUTION:  13 15 [-1, 1, -1, -1, -1] [-1, 1, -1, -1, -1] 5
SOLUTION:  14 17 [-1, 1, -1, -1, 4] [-1, 1, -1, -1, 4] 5
SOLUTION:  15 18 [-1, 1, -1, 3, -1] [-1, 1, -1, 3, -1] 5
SOLUTION:  16 20 [-1, 1, -1, 3, 4] [-1, 1, -1, 3, 4] 5
SOLUTION:  21 21 [-1, 1, 2, -1, 4] [-1, 1, 2, -1, 4] 5
SOLUTION:  22 22 [-1, 1, 2, 3, -1] [-1, 1, 2, 3, -1] 5
SOLUTION:  23 24 [-1, 1, 2, 3, 4] [-1, 1, 2, 3, 4] 5

(23, 24, [])
```

algorithm **backtrack**():

if (solution == True)

return **True** ← Solution found

for each possible moves

if(this move is valid)

select this move and place

ok = call **backtrack**() ← Keep exploring

if ok:

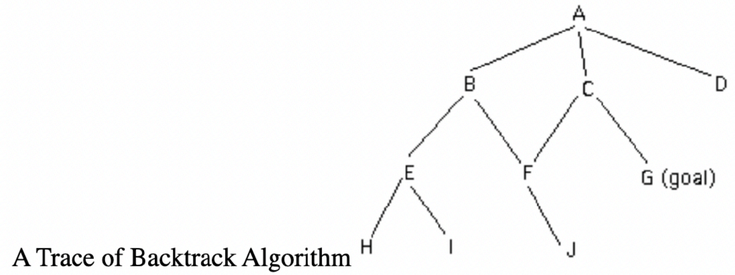
return **solution found**

unplace that selected move

return **False** ← Don't explore anymore! no solution in this path

Backtracking

- L'algoritme de backtracking utilitza tres llistes i una variable
  - **SL**, la llista d'estats (state list), enumera els estats del camí actual que s'està provant. Si es troba un objectiu, SL conté la llista ordenada d'estats al camí.
  - **NSL**, la nova llista d'estats (new state list), conté els nodes en espera d'avaluació, és a dir, els nodes descendents els quals encara no s'han explorat.
  - **DE,** nodes terminals (dead ends), enumera els nodes descendents dels quals no han pogut contenir una solució vàlida. Si es tornen a trobar aquests estats, s'eliminaran immediatament de la consideració.
  - **CS**, l'estat actual (the current state).

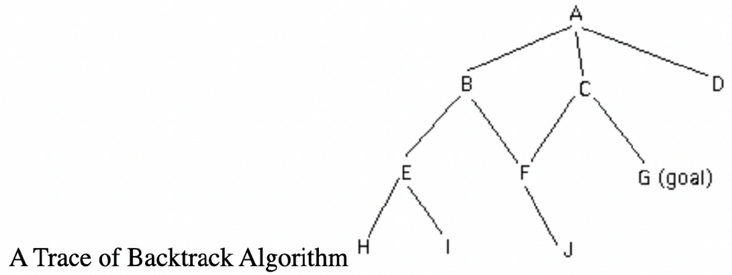


A Trace of Backtrack Algorithm

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]

Backtracking

- L'algoritme de backtracking utilitza tres llistes i una variable
  - **SL**, la llista d'estats (state list), enumera els estats del camí actual que s'està provant. Si es troba un objectiu, SL conté la llista ordenada d'estats al camí.
  - **NSL**, la nova llista d'estats (new state list), conté els nodes en espera d'avaluació, és a dir, els nodes descendents els quals encara no s'han explorat.
  - **DE,** nodes terminals (dead ends), enumera els nodes descendents dels quals no han pogut contenir una solució vàlida. Si es tornen a trobar aquests estats, s'eliminaran immediatament de la consideració.
  - **CS**, l'estat actual (the current state).



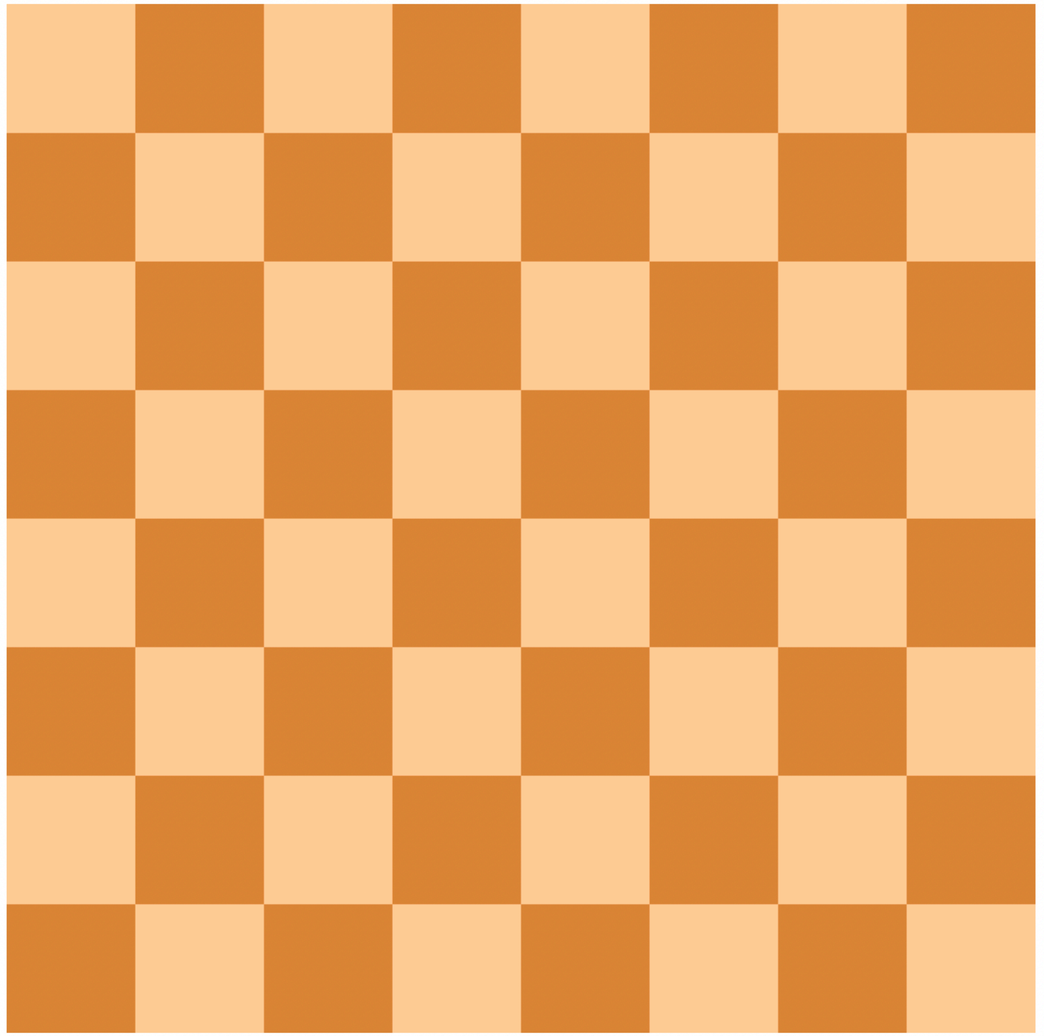
A Trace of Backtrack Algorithm

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

Backtracking

Problema de les N-Reines

- Volem col·locar N reines en un tauler d'escacs de NxN sense que hi hagi amenaça.



Backtracking

Problema de les N-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;
   }
}
```

Backtracking

Problema de les N-Reines

- Solució per força bruta?
  - Penseu una solució iterativa

```
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;
}
```

Backtracking

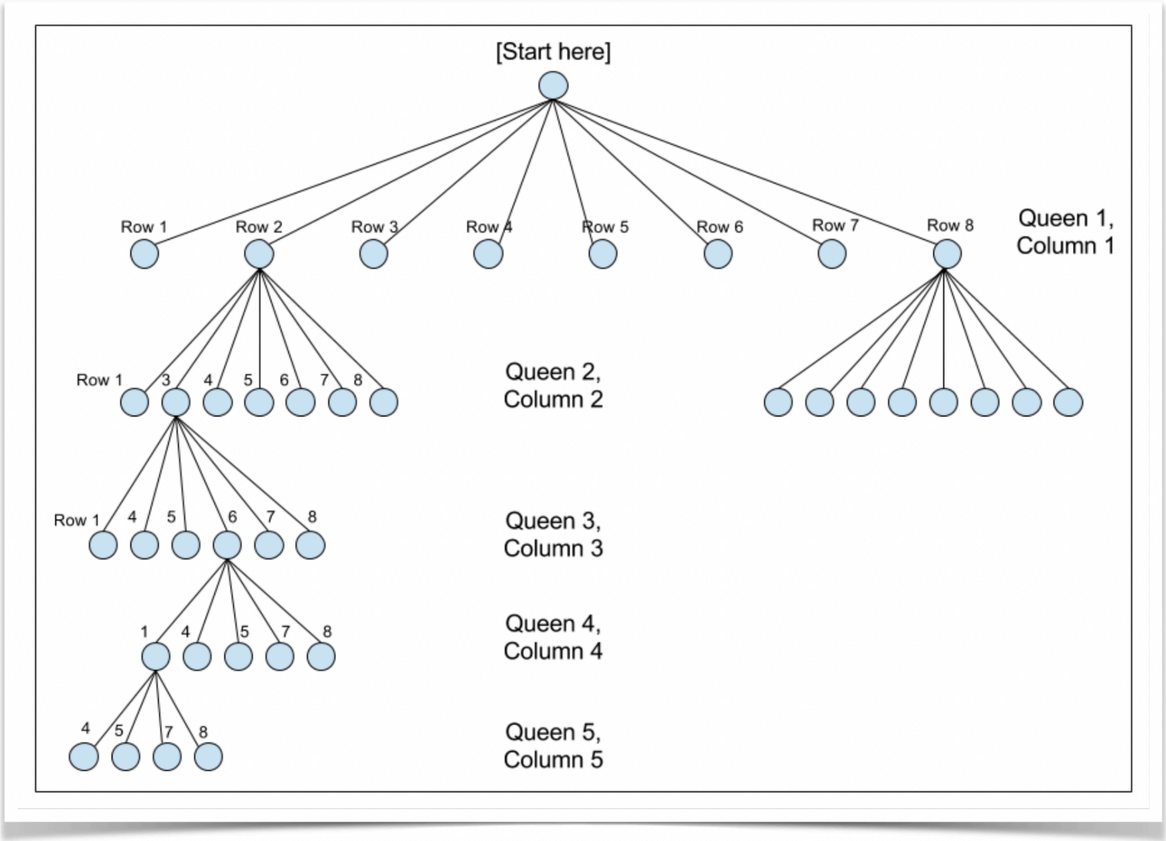
Problema de les N-Reines

- Reduïm la complexitat: incloure restriccions
  - –> Les reines han d'estar a diferents files i diferents columnes
  - –> No poden estar a la mateixa diagonal

Backtracking

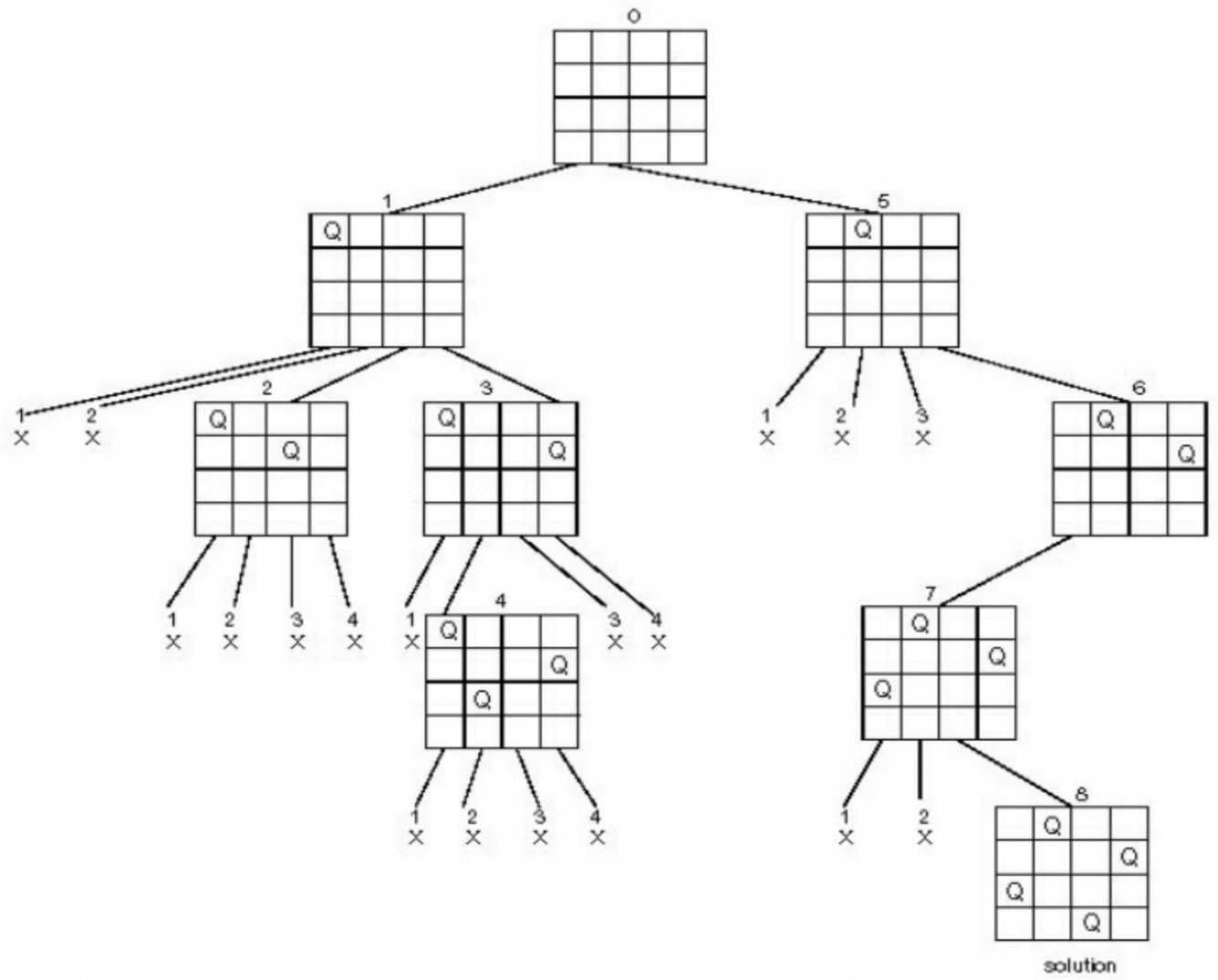
Problema de les N-Reines: **Solució amb backtracking**

- Pensem una solució mitjançant backtracking



Backtracking

Problema de les N-Reines: **Solució amb backtracking**



Backtracking

Problema de les N-Reines: **Solució amb backtracking**

```
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.
```

```
algorithm backtrack():
  if (solution == True)
    return True                          <-- Solution found
  for each possible moves
    if(this move is valid)
      select this move and place
      ok = call backtrack()              <-- Keep exploring
      if ok:
          return solution found
      unplace that selected move
  return False                           <-- Don't explore anymore!
                                             no solution in this path
```

Backtracking

Problema de les N-Reines: **Solució amb backtracking**

Sudoku

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

Solució amb Backtracking?

Backtracking

Sudoku

- Amb **backtracking** tenim tres punts importants:
    - Les **eleccions**.
    - Les **restriccions**.
    - **L'objectiu**.
- En el cas del problema del **SUDOKU**. Ens queda d'aquesta manera:
    - **Les eleccions**: Un número (1,..,9) a cada una de les cel·les del taulell
    - **Les restriccions**: No pots haver números repetits en una columna/fila, ni a la sub-graella a la que pertany la cel·la.
    - **L'objectiu**: Omplir totes les cel·les buides

Sudoku

```
algorithm backtrack():
    if (solution == True)
        return True                          <-- Solution found
    for each possible moves
        if(this move is valid)
            select this move and place
            ok = call backtrack()            <-- Keep exploring
            if ok:
                return solution found
            unplace that selected move
    return False                             <-- Don't explore anymore!
                                                 no solution in this path
```

Sudoku

```
algorithm backtrack():
    if (solution == True)
        return True                          <-- Solution found
    for each possible moves
        if(this move is valid)
            select this move and place
            ok = call backtrack()            <-- Keep exploring
            if ok:
                return solution found
            unplace that selected move
    return False                             <-- Don't explore anymore! no solution in this path
```

Solució amb Backtracking?

Hi ha alguna cel·la sense emplenar?

per els digits del 0 al 9

genera cap conflicte?

Sudoku

```
def solve(board,i,j):
    if(i==9): #solution found
        printBoard(board)
        return True
    if board[i][j] != 0: # the cell is already filled -> go to next cell
        if j == 8:
            solve(board, i+1,0)
        else:
            solve(board, i,j+1)
    else:
        for val in range(1, 10):
            if isPossible(board, i, j, val):
                board[i][j] = val # select the move
                if j == 8:
                    solve(board, i+1,0)
                else:
                    solve(board, i,j+1)
                board[i][j] = 0 # Bad choice, unplace
        return False

    # We found a solution, print it

solve(board, 0,0)
```

```
- - - - - - - - - - - - -
| 0 0 0 | 8 0 0 | 4 0 3 |
| 2 0 0 | 0 0 4 | 8 9 0 |
| 0 9 0 | 0 0 0 | 0 0 2 |
- - - - - - - - - - - - -
| 0 0 0 | 0 2 9 | 0 1 0 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 7 0 | 6 5 0 | 0 0 0 |
- - - - - - - - - - - - -
| 9 0 0 | 0 0 0 | 0 8 0 |
| 0 6 2 | 7 0 0 | 0 0 1 |
| 4 0 3 | 0 0 6 | 0 0 0 |
```

```
- - - - - - - - - - - - -
| 7 5 1 | 8 9 2 | 4 6 3 |
| 2 3 6 | 1 7 4 | 8 9 5 |
| 8 9 4 | 5 6 3 | 1 7 2 |
- - - - - - - - - - - - -
| 6 4 5 | 3 2 9 | 7 1 8 |
| 1 2 9 | 4 8 7 | 3 5 6 |
| 3 7 8 | 6 5 1 | 2 4 9 |
- - - - - - - - - - - - -
| 9 1 7 | 2 3 5 | 6 8 4 |
| 5 6 2 | 7 4 8 | 9 3 1 |
| 4 8 3 | 9 1 6 | 5 2 7 |
```