Backtracking

Md. Ahanaf Arif Khan
ID: 1910676110
Session: 2018-19
Department of Computer Science & Engineering, University of Rajshahi

Topics Discussed

- What is backtracking
- State Space Tree
- Bounding Function
- Why use backtracking
- General Algorithm
- Backtracking Applications

What is Backtracking

Backtracking can be defined as a general algorithmic technique that considers ***searching every possible combination*** *in order to solve a computational problem.*

- builds a set of all the solutions **incrementally**.
- Uses the **brute force** method.
- uses **recursive calling** to find a solution set by building a solution step by step, increasing levels with time.
- uses **depth-first search** of a rooted tree.
- are generally **exponential** in both time and space complexity

What is Backtracking

The Brute force approach tries out all the possible solutions and chooses the desired/best solutions.

"The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions."

The principal idea is to construct solutions **one component at a time** and evaluate such partially constructed candidates.

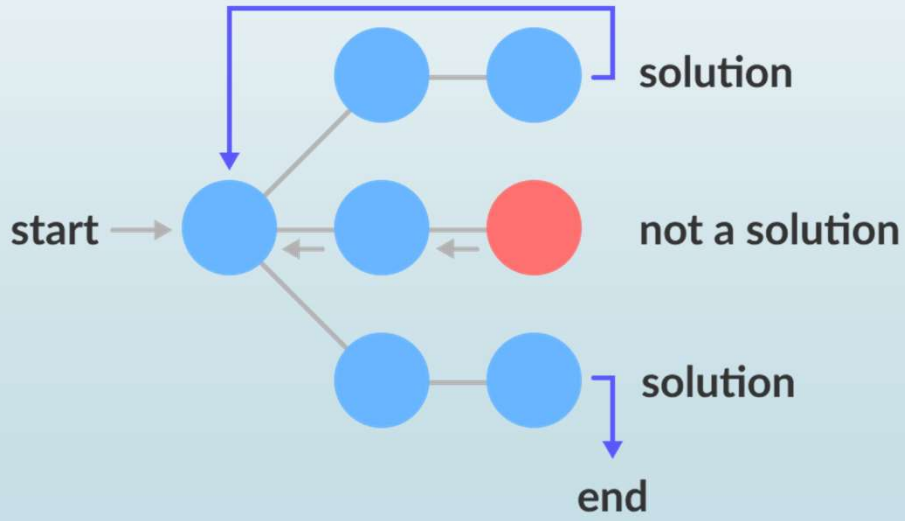
What is Backtracking

There are **three** types of problems in backtracking –

- Decision Problem – Does the problem **have** a solution?
- Optimization Problem – What is the **best solution** to the problem?
- Enumeration Problem – What are **all possible solutions** to the problem?

State Space Tree

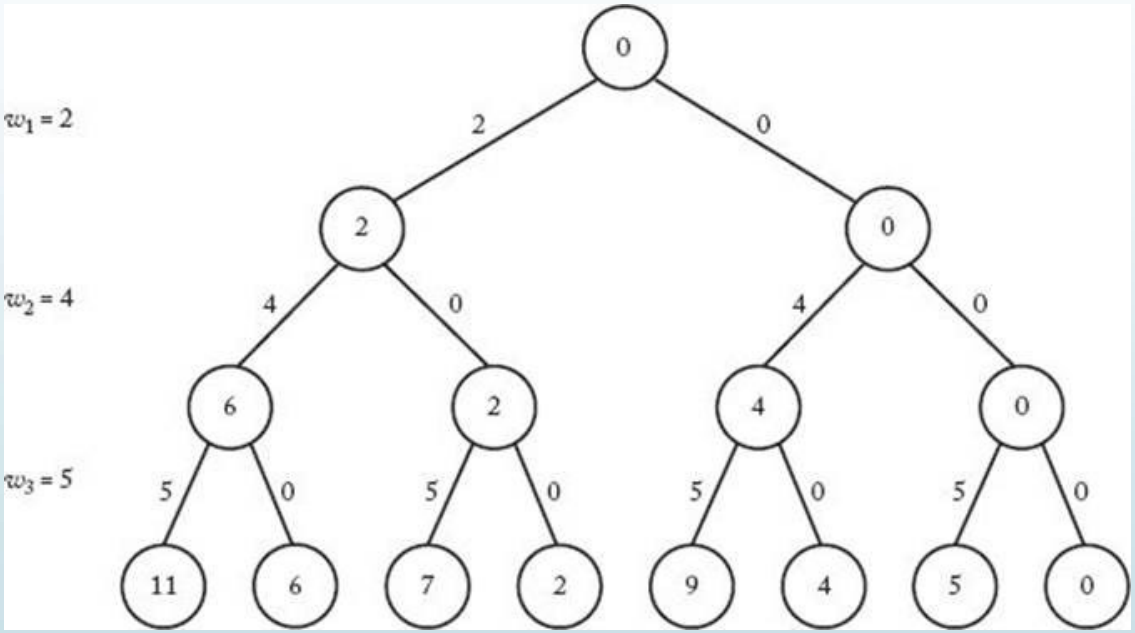
A space state tree is a tree representing ***all the possible states*** *(solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.*



State Space Tree

- The trees root represents an initial state before the search for a solution begins.
- The nodes of the first level in the tree represent the choices made for the first component of a solution
- The nodes of the second level represent the choices for the second component, and so on.

State Space Tree



State space tree for Subset Sum Problem

Bounding Function

The bounding function is a function that checks if we can get a valid solution of our problem down the path we are currently expanding.

Bounding function is needed to help **kill** some live nodes without actually expanding them.

General Algorithm

```
Backtrack(X)
If X is a solution:
        print X
else
        if X is a new solution
                add to the list of solutions
        backtrack(expand x)
```

Backtracking Applications

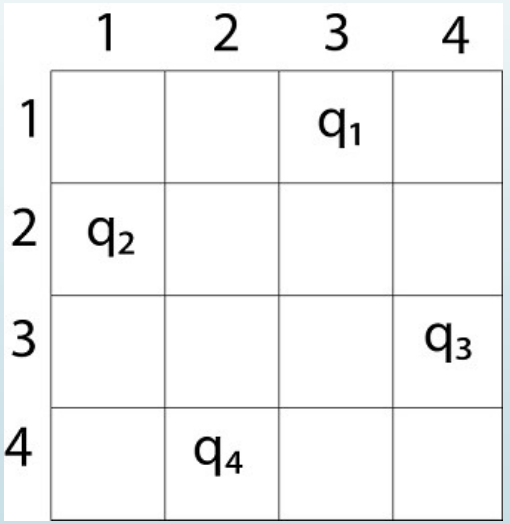
- N – Queens Problem
- Hamiltonian Paths
- Maze Solving Problem
- Subset Sum Problem

N - Queens Problem

Goal: position n queens on a n x n board such that no two queens threaten each other.

- No two queens may be in the -
  - Same row,
  - Same column,
  - Same diagonal

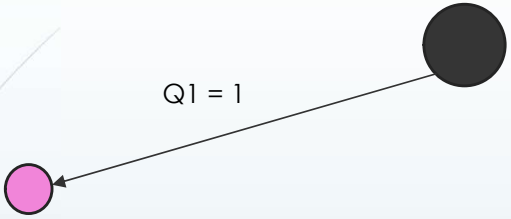
The following is a solution for 4 Queen problem.



N – Queens Problem

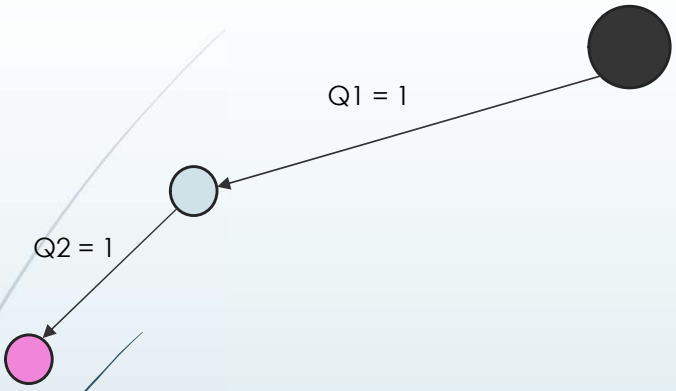
	1	2	3	4
1				
2				
3				
4				

N – Queens Problem



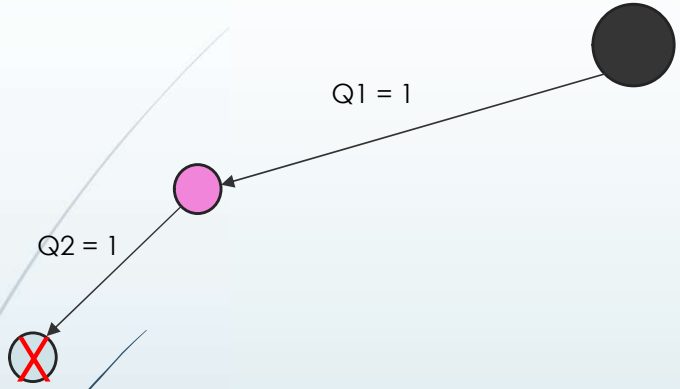
	1	2	3	4
1	Q1			
2				
3				
4				

N – Queens Problem



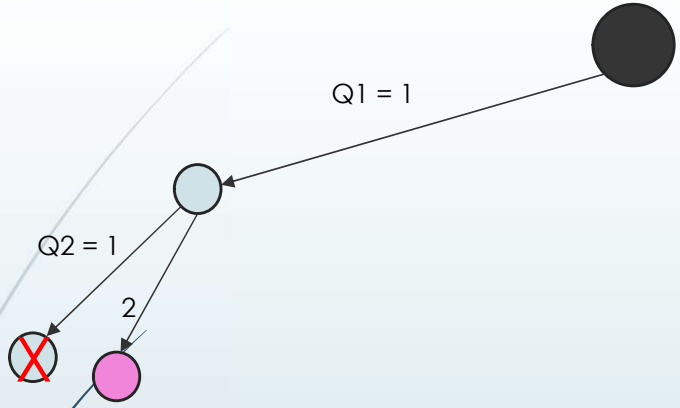
	1	2	3	4
1	Q1			
2	Q2			
3				
4				

N – Queens Problem



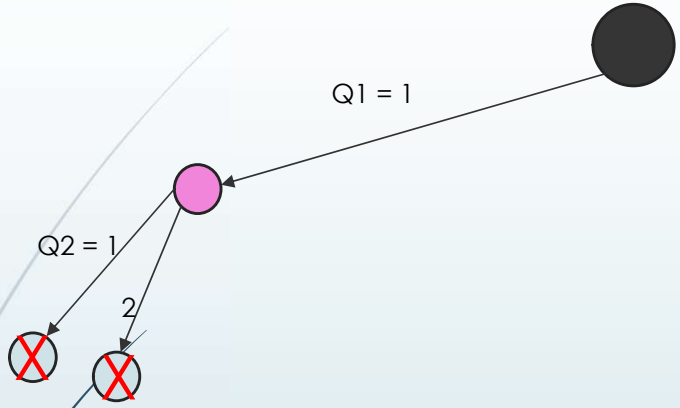
	1	2	3	4
1	Q1			
2				
3				
4				

N – Queens Problem



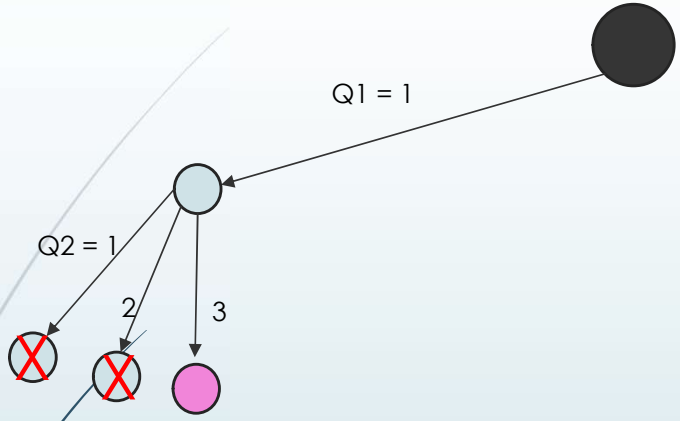
	1	2	3	4
1	Q1			
2		Q2		
3				
4				

N – Queens Problem



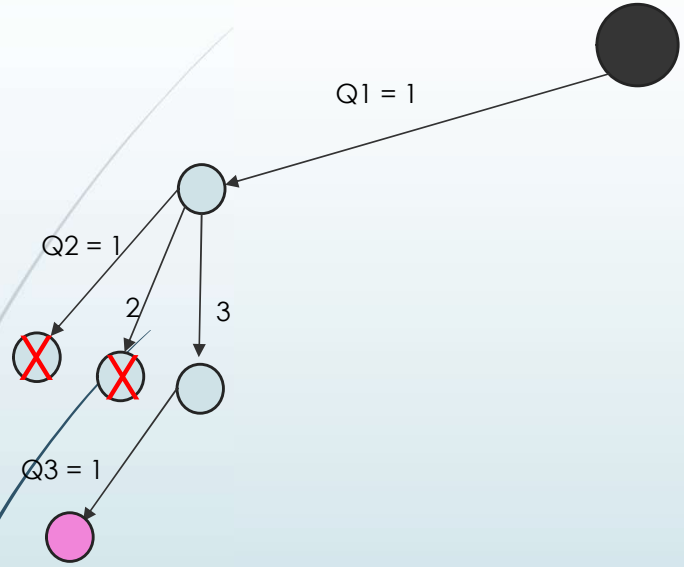
	1	2	3	4
1	Q1			
2				
3				
4				

N – Queens Problem



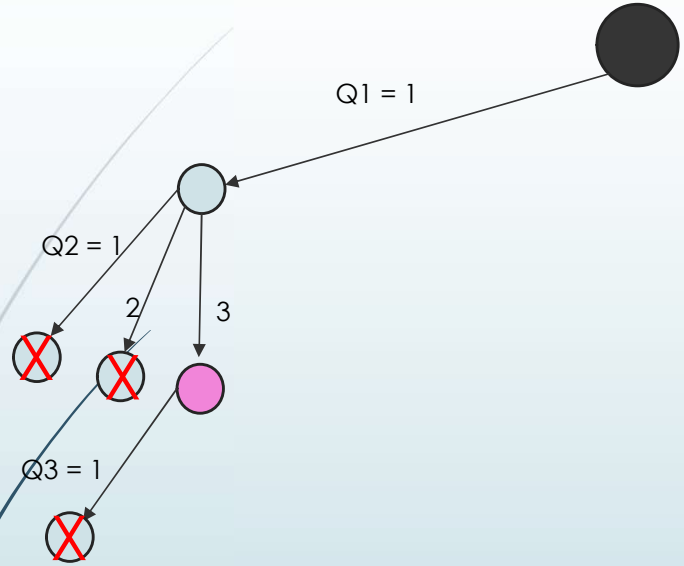
	1	2	3	4
1	Q1			
2			Q2	
3				
4				

N – Queens Problem



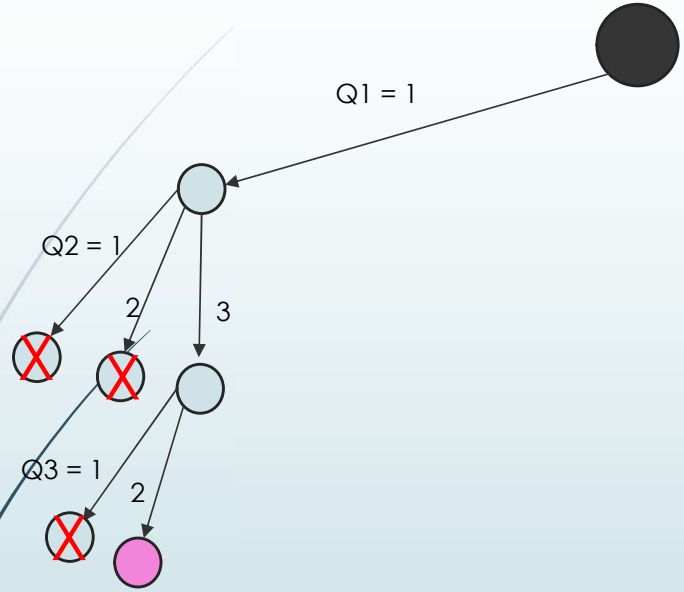
	1	2	3	4
1	Q1			
2			Q2	
3	Q3			
4				

N – Queens Problem



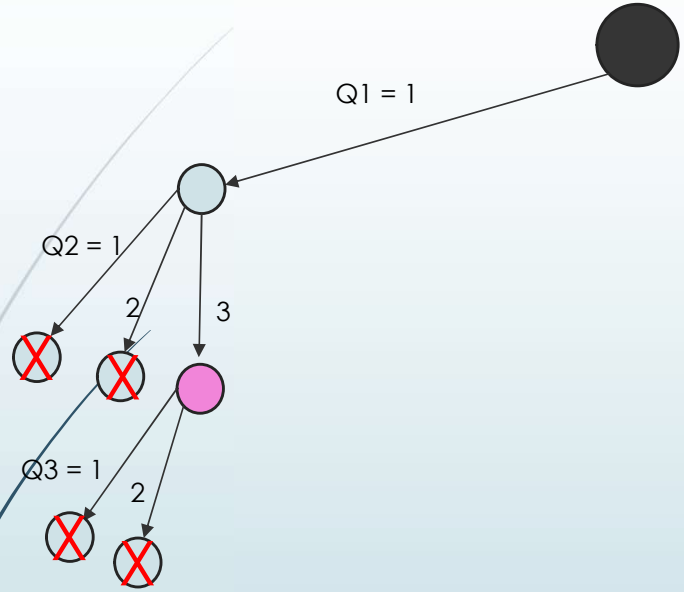
	1	2	3	4
1	Q1			
2			Q2	
3				
4				

N – Queens Problem



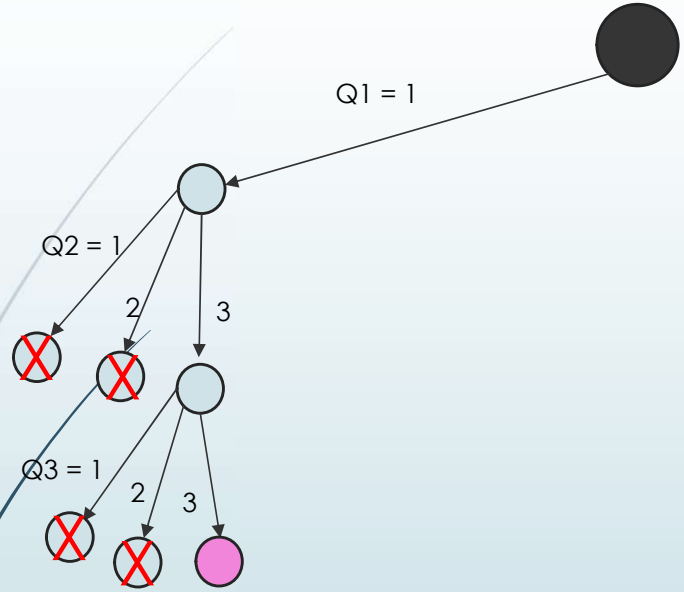
	1	2	3	4
1	Q1			
2			Q2	
3		Q3		
4				

N – Queens Problem



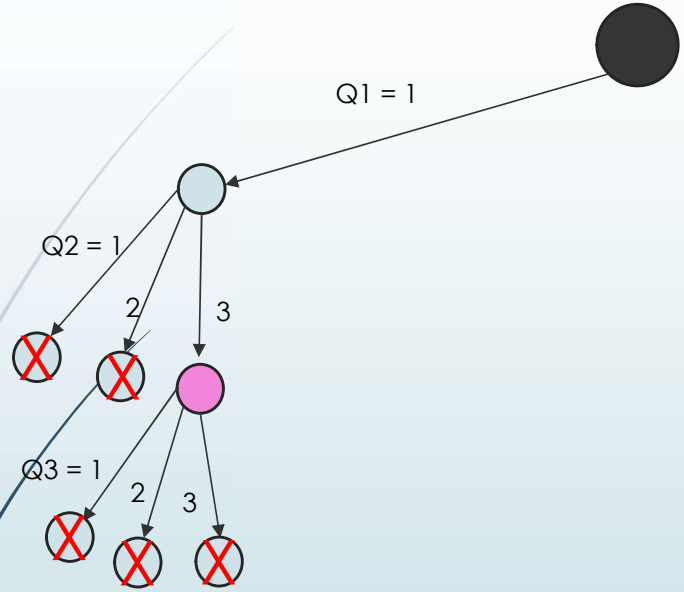
	1	2	3	4
1	Q1			
2			Q2	
3				
4				

N – Queens Problem



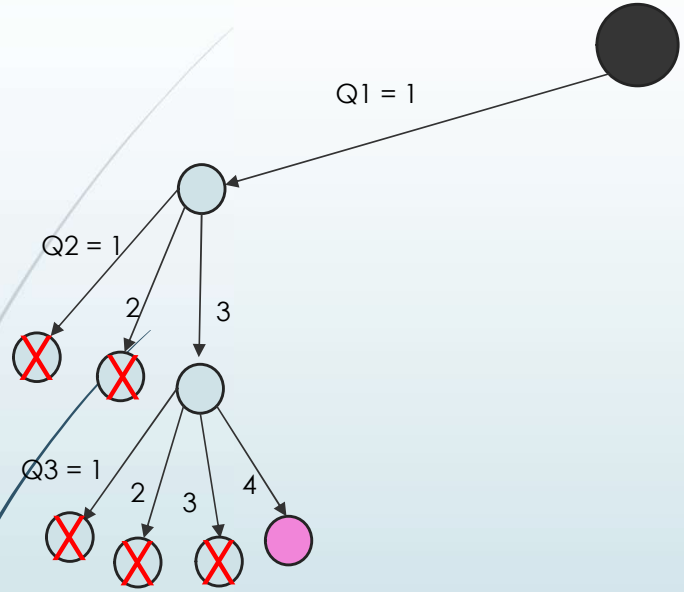
	1	2	3	4
1	Q1			
2			Q2	
3			Q3	
4				

N – Queens Problem



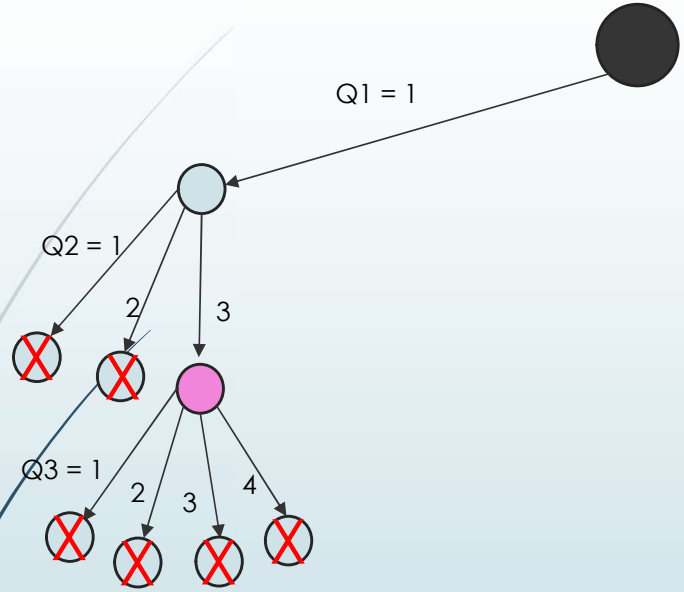
	1	2	3	4
1	Q1			
2			Q2	
3				
4				

N – Queens Problem



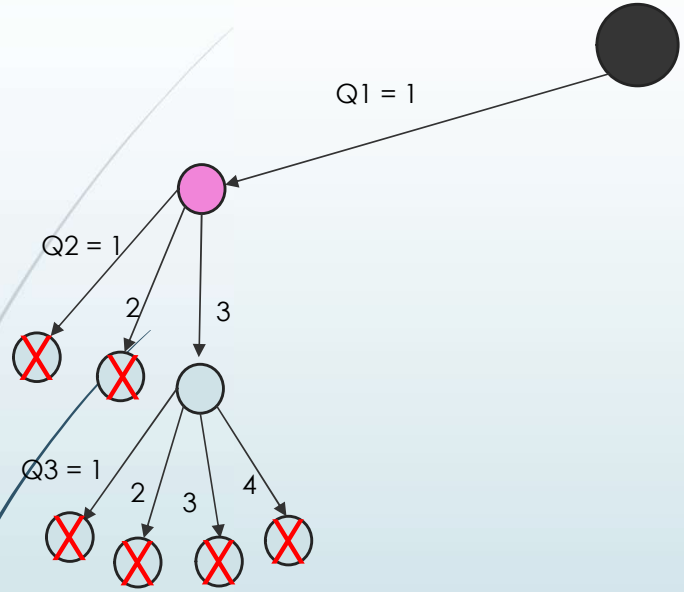
	1	2	3	4
1	Q1			
2			Q2	
3				Q3
4				

N – Queens Problem



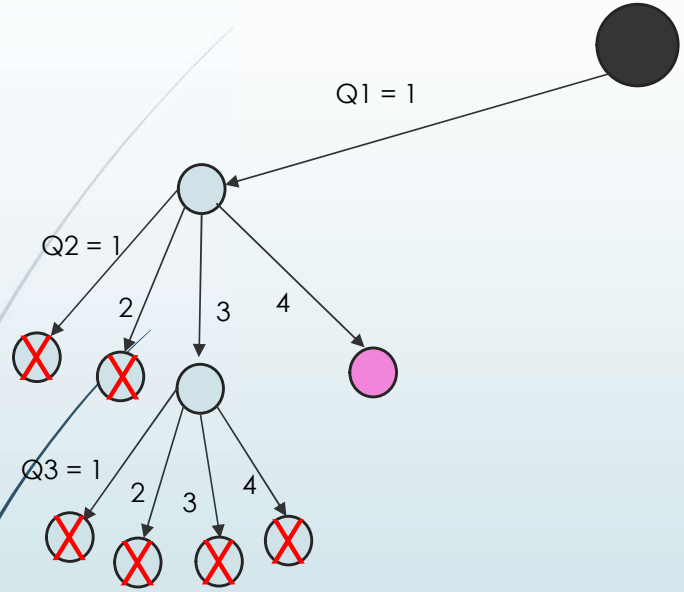
	1	2	3	4
1	Q1			
2			Q2	
3				
4				

N – Queens Problem



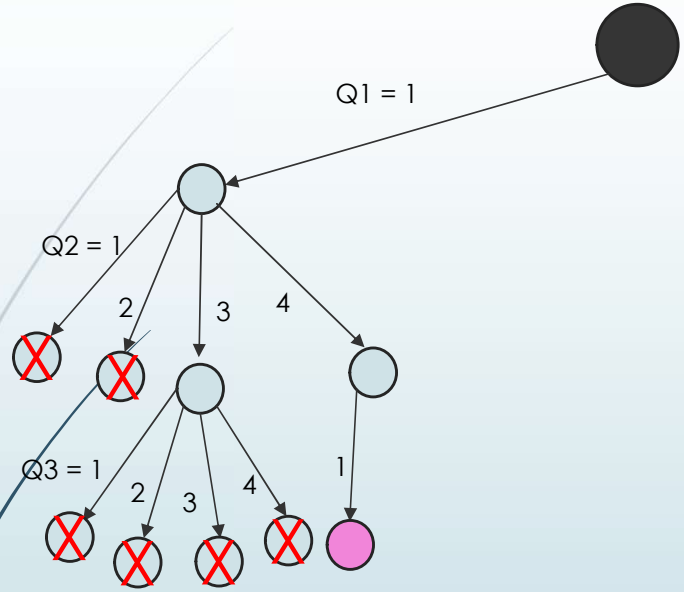
	1	2	3	4
1	Q1			
2				
3				
4				

N – Queens Problem



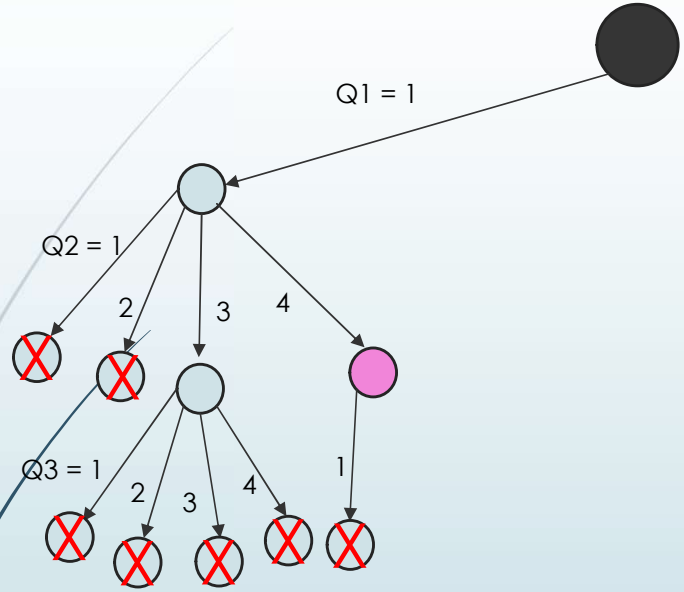
	1	2	3	4
1	Q1			
2				Q2
3				
4				

N – Queens Problem



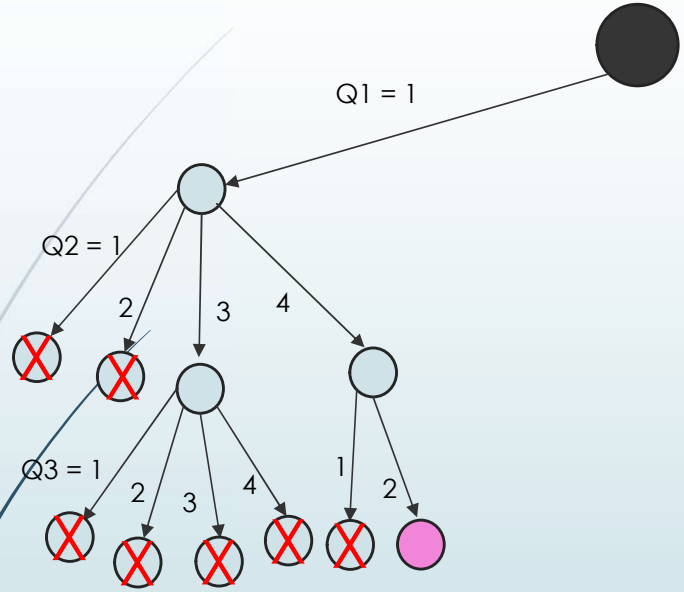
	1	2	3	4
1	Q1			
2				Q2
3	Q3			
4				

N – Queens Problem



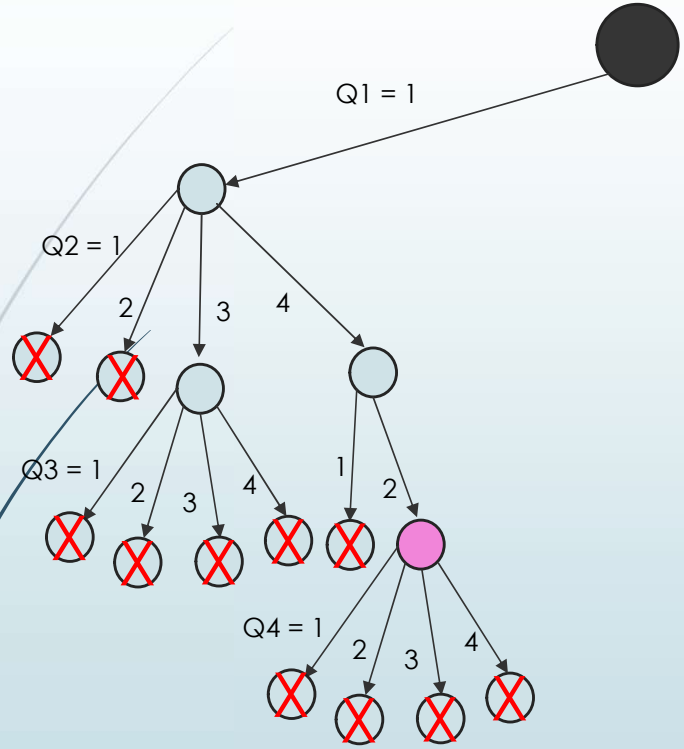
	1	2	3	4
1	Q1			
2				Q2
3				
4				

N – Queens Problem



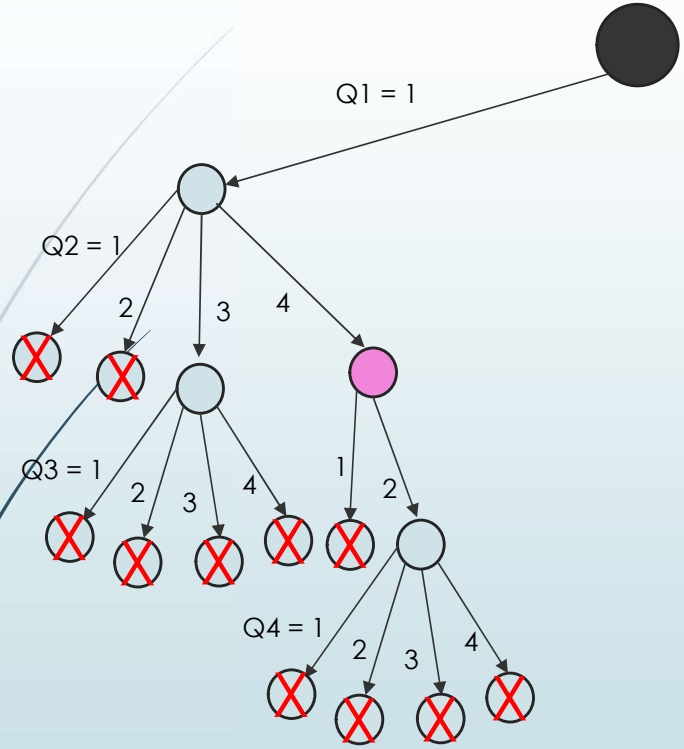
	1	2	3	4
1	Q1			
2				Q2
3		Q3		
4				

N – Queens Problem



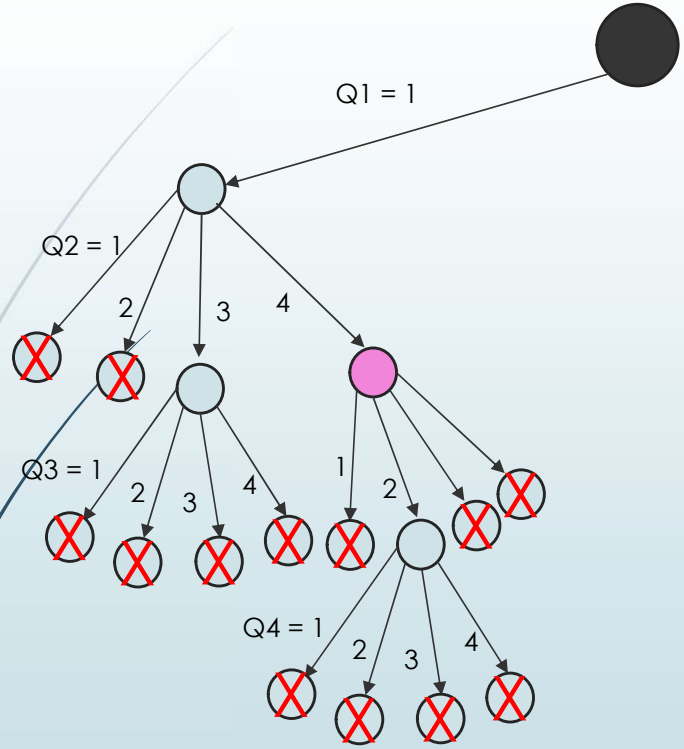
	1	2	3	4
1	Q1			
2				Q2
3		Q3		
4				

N – Queens Problem



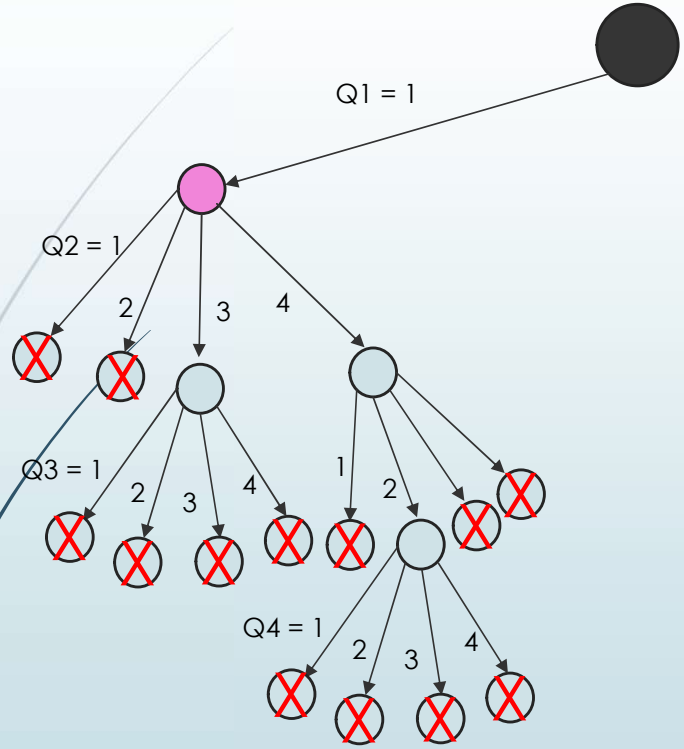
	1	2	3	4
1	Q1			
2				Q2
3				
4				

N – Queens Problem



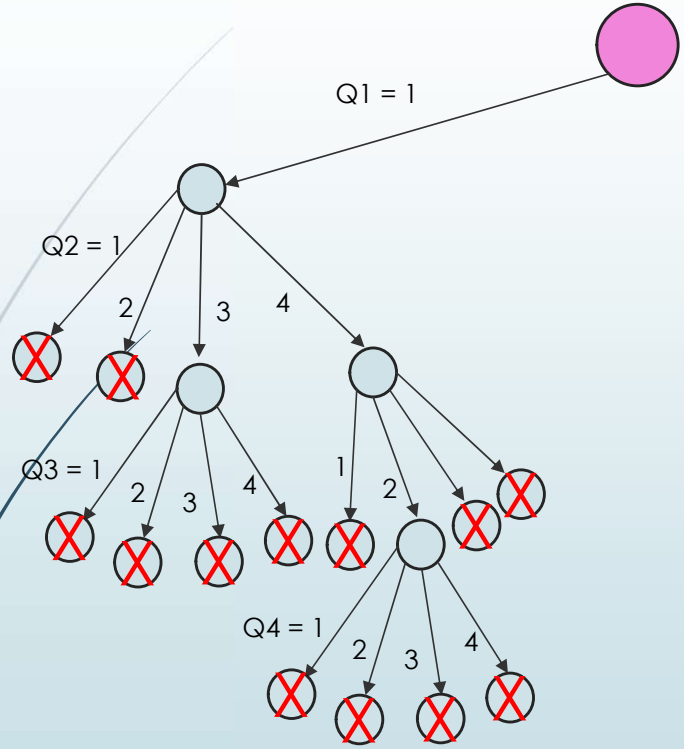
	1	2	3	4
1	Q1			
2				Q2
3				
4				

N – Queens Problem



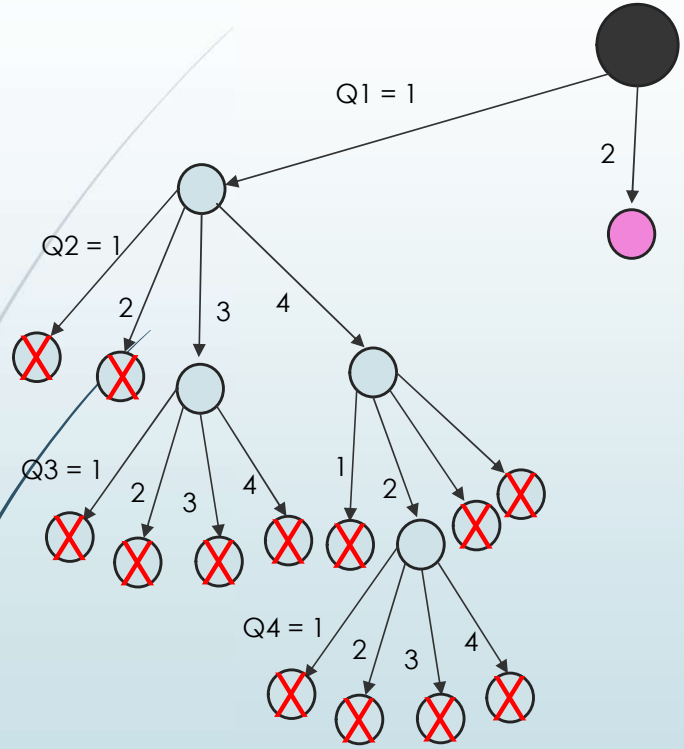
	1	2	3	4
1	Q1			
2				
3				
4				

N – Queens Problem



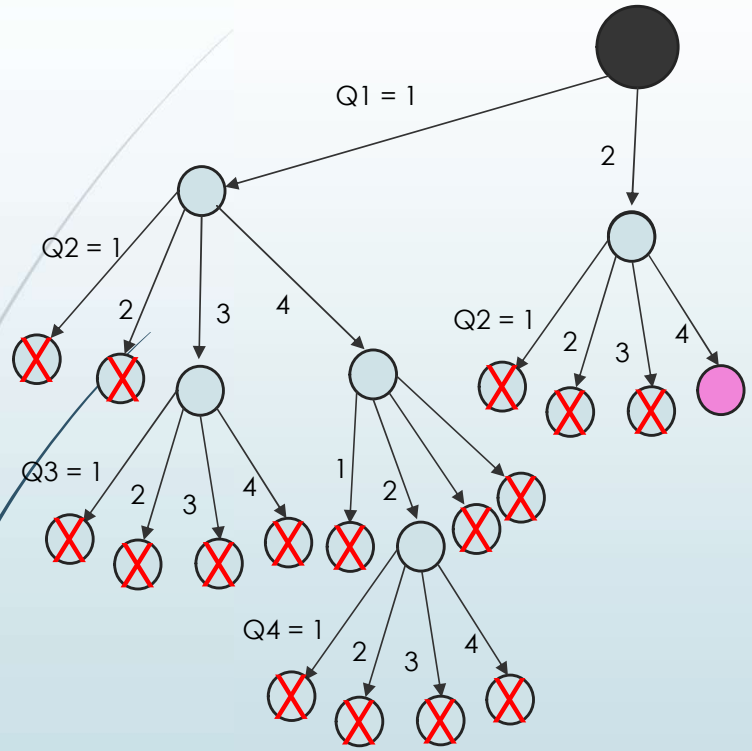
	1	2	3	4
1				
2				
3				
4				

N – Queens Problem



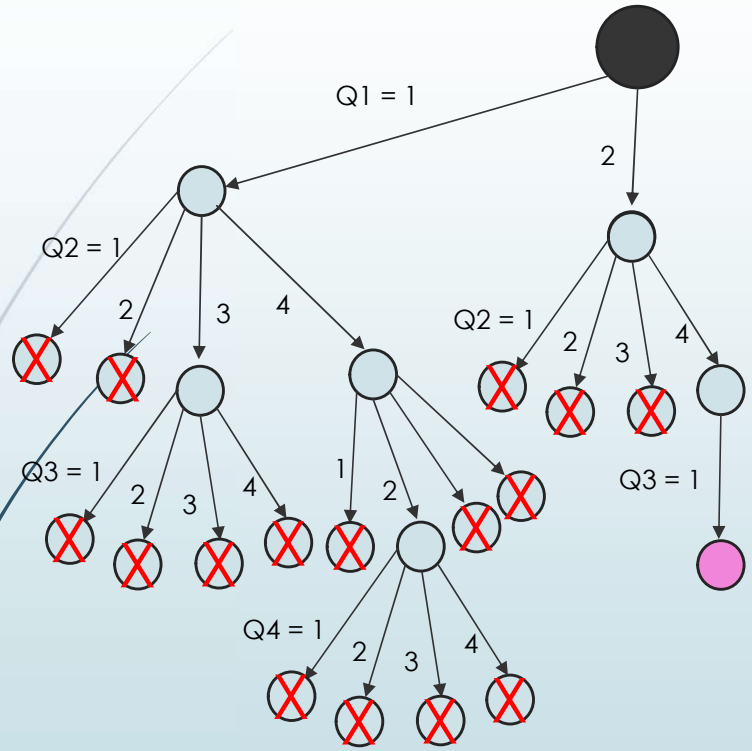
	1	2	3	4
1		Q1		
2				
3				
4				

N – Queens Problem



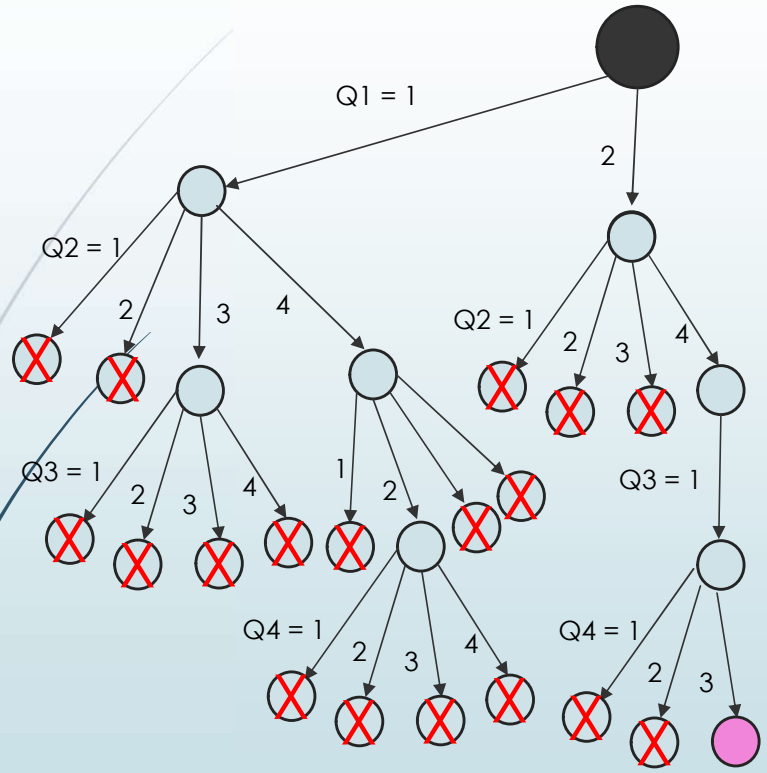
	1	2	3	4
1		Q1		
2				Q2
3				
4				

N – Queens Problem



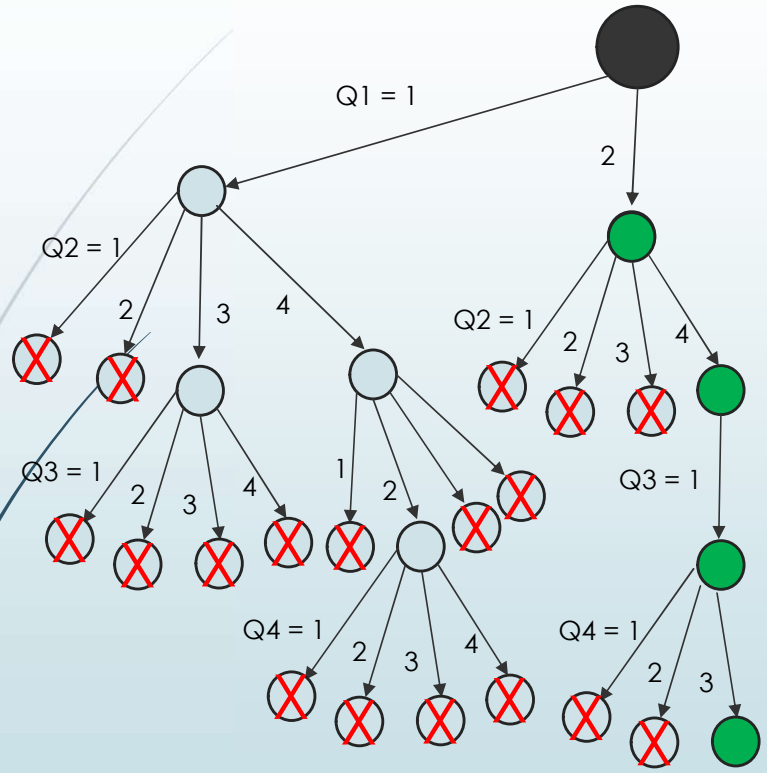
	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4				

N – Queens Problem



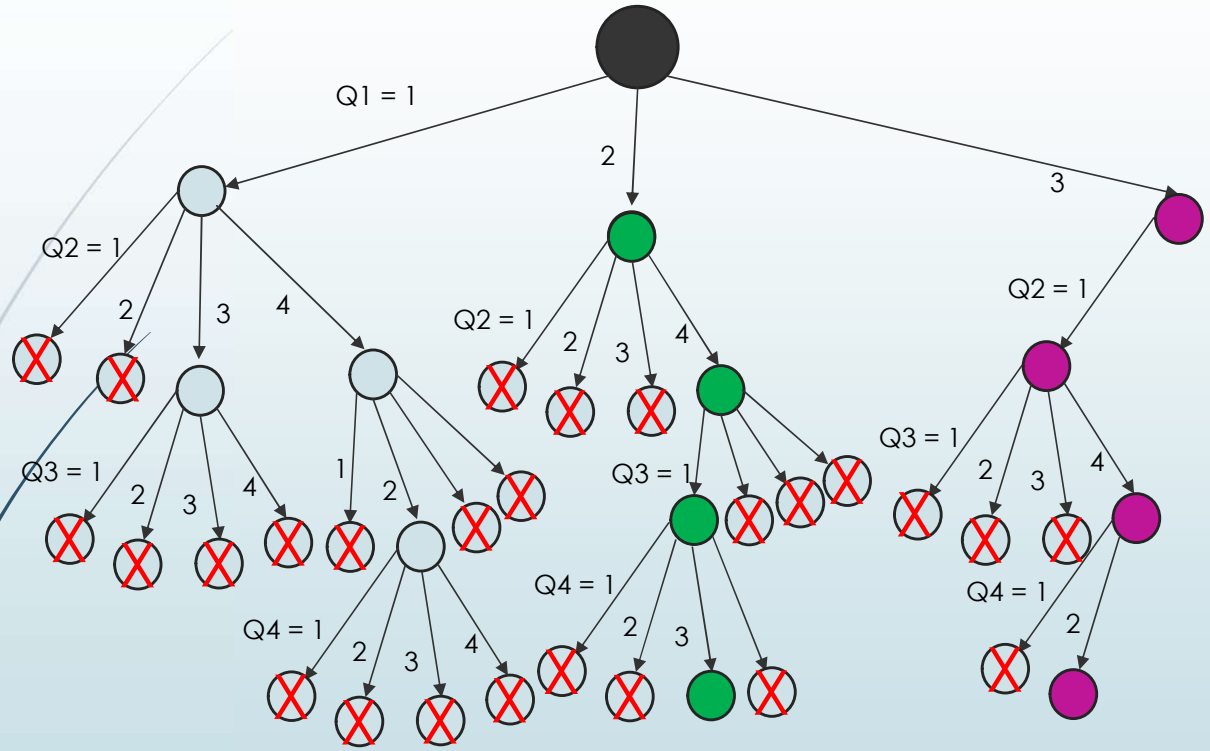
	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	

N – Queens Problem



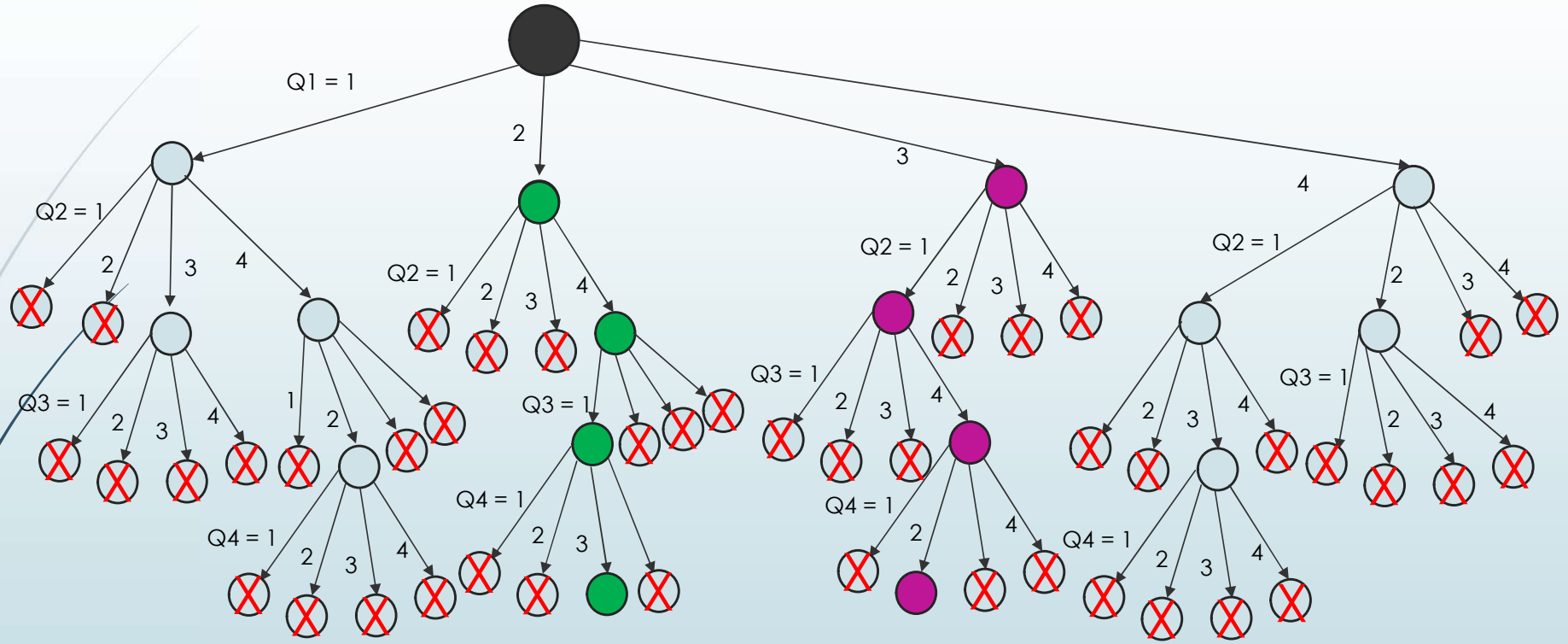
	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	

N – Queens Problem

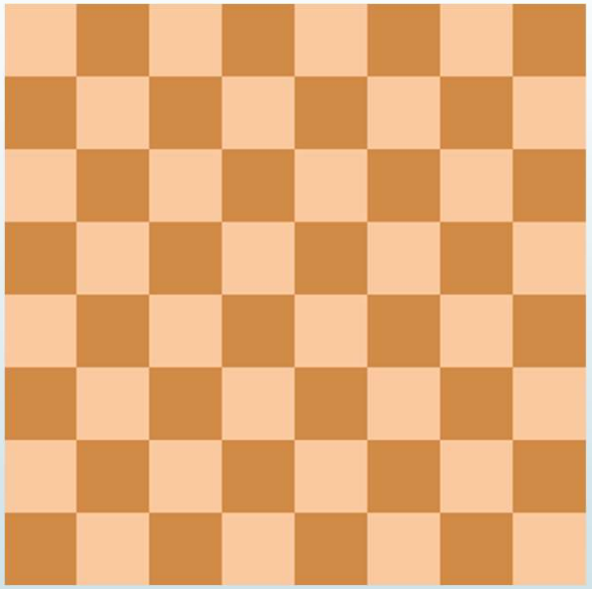


	1	2	3	4
1			Q1	
2	Q2			
3				Q3
4		Q4		

N – Queens Problem

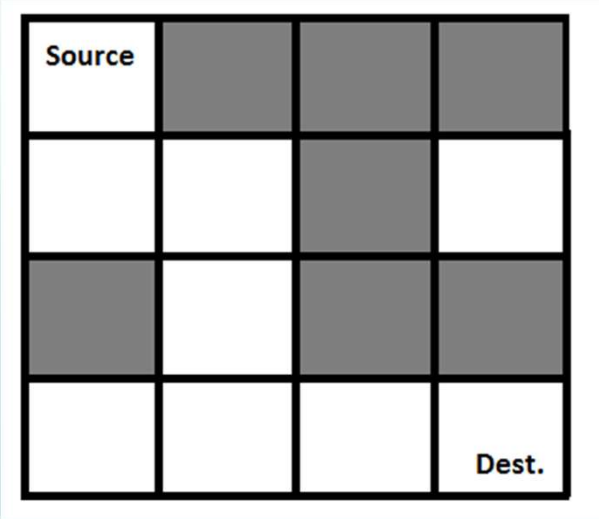


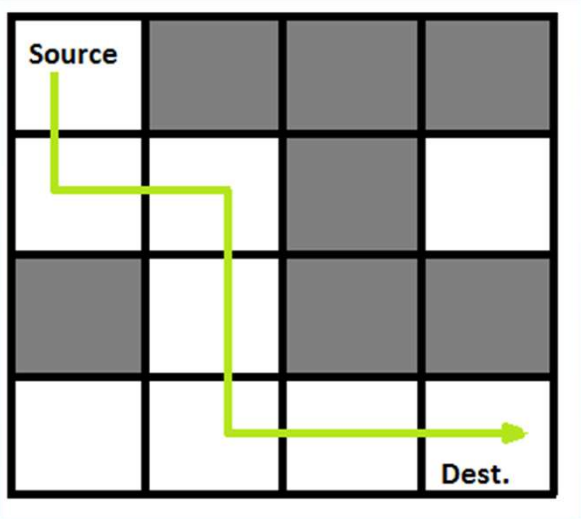
N – Queens Problem



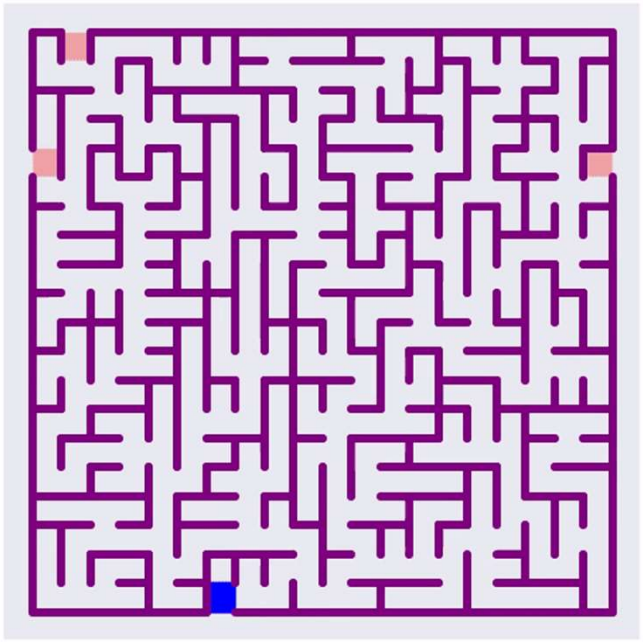
Maze Solving Problem

A Maze is given as N*N binary matrix of blocks where source block is the upper left most block and destination block is lower rightmost block. We want to start from the source and explore the maze to reach the destination. We can move only in two directions: forward and down.





Maze Solving Problem



Subset Sum Problem

Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique