Backtracking

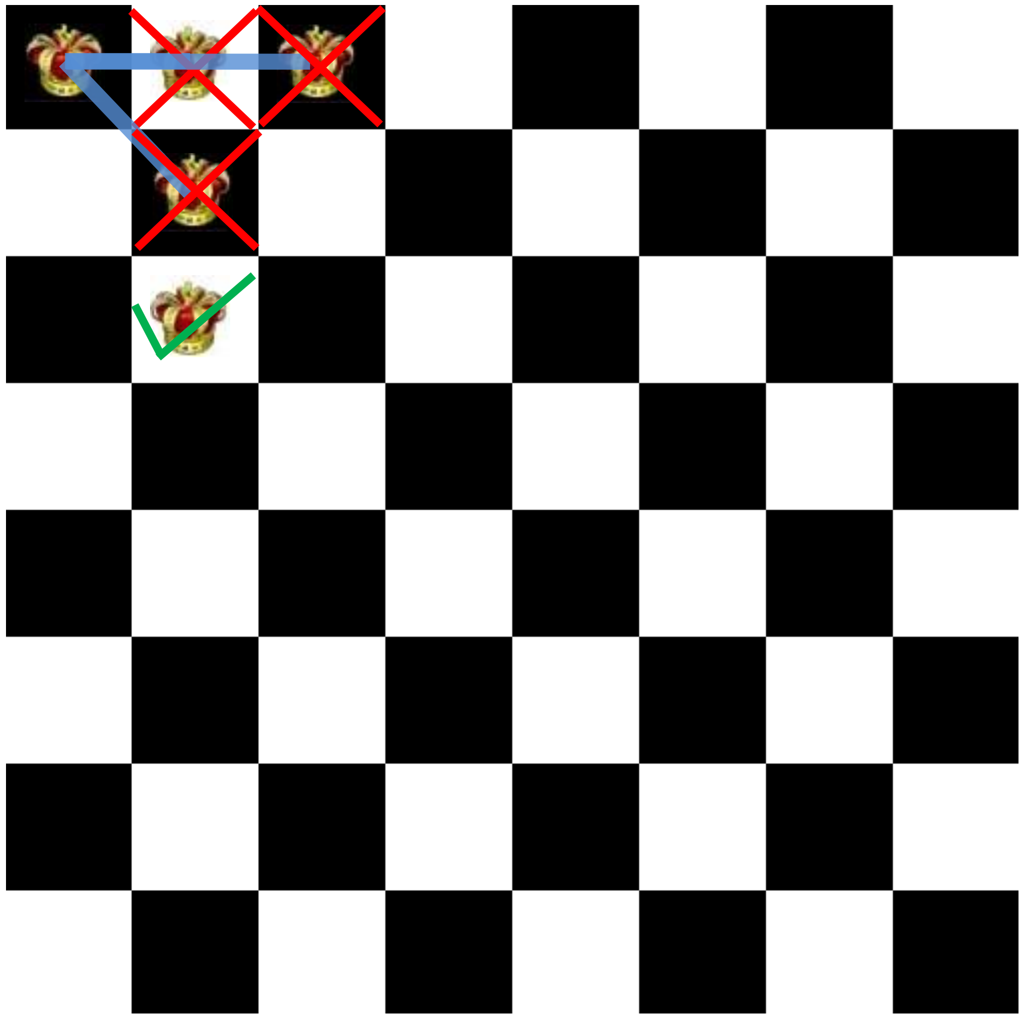
8 Queen’s Problem

Backtracking

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

- Backtracking is an important tool for solving constraint satisfaction problems, such as *crosswords*, *verbal arithmetic*, *Sudoku*, and many other puzzles.

8 Queen’s Problem



8 Queen's Problem

- The **eight queens puzzle** is the problem of placing eight chess queens on an 8 X 8 chessboard so that no two queens attack each other.
- Thus, a solution requires that no two queens share the same row, column, or diagonal.
- The eight queens puzzle is an example of the more general **n-queens problem** of placing n queens on an *n X n* chessboard, where solutions exist for all natural numbers *n* with the exception of *1, 2 and 3*.

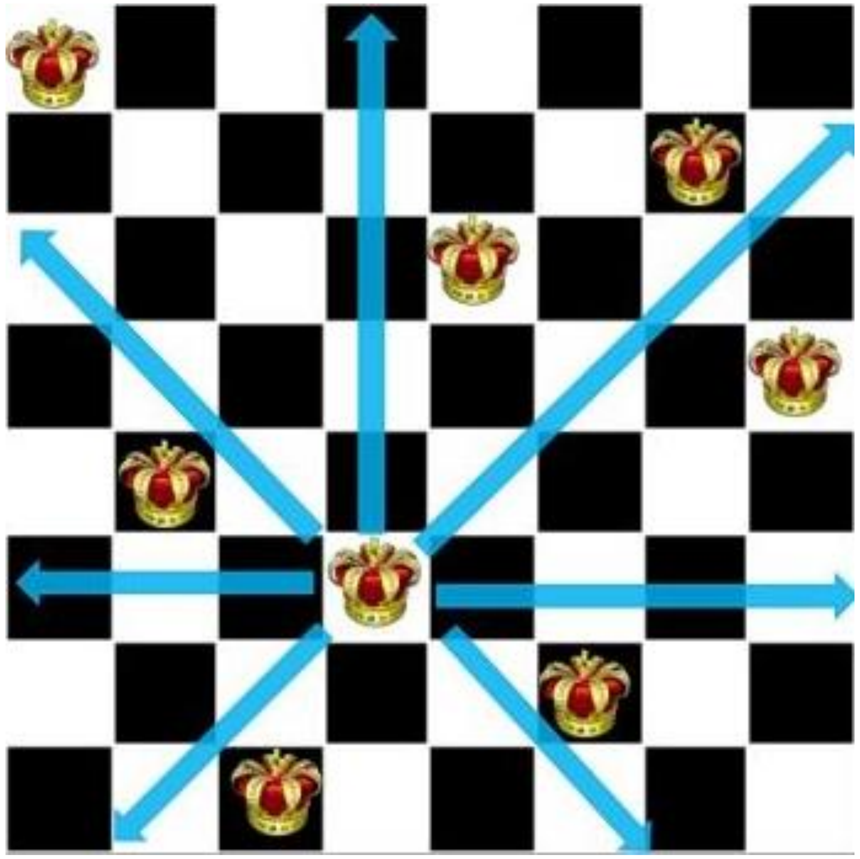
Formulation

States: any arrangement of 0 to 8 queens on the board

Initial state: 0 queens on the board

Successor function: add a queen in any square

Goal test: 8 queens on the board, none attacked



Backtracking Concept

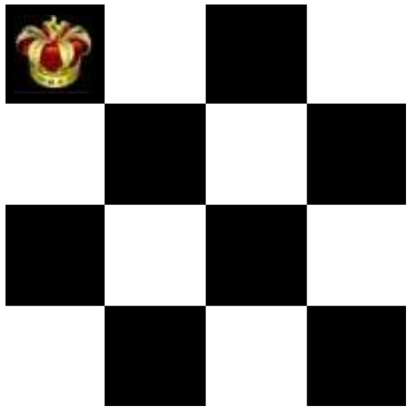
- Each recursive call attempts to place a queen in a specific column.
- For a given call, the state of the board from previous placements is known (i.e. where are the other queens?)
- **Current step backtracking:** If a placement within the column does not lead to a solution, the queen is removed and moved "down" the column.
- **Previous step backtracking:** When all rows in a column have been tried, the call terminates and backtracks to the previous call (in the previous column).

Backtracking Concept

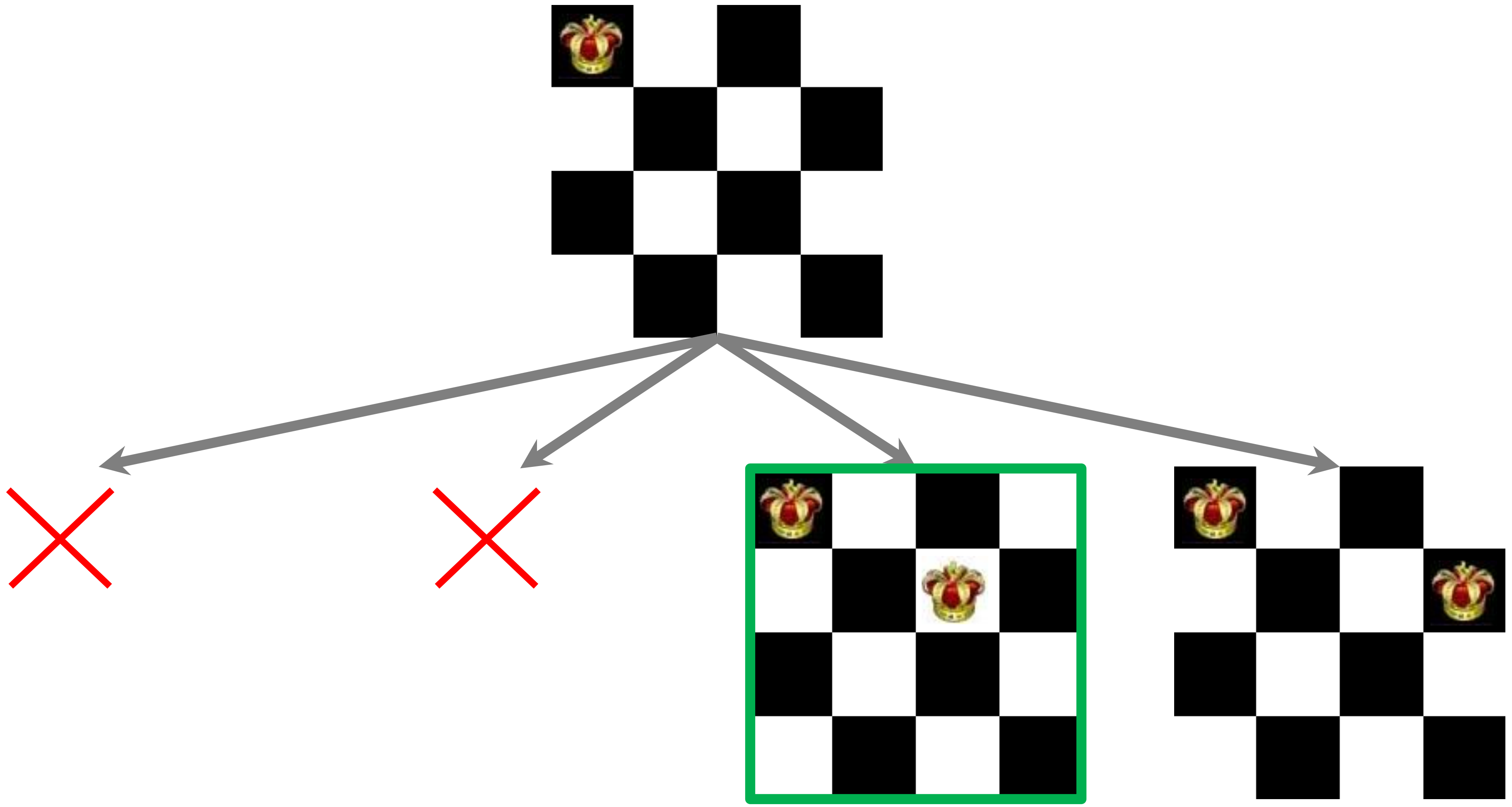
- **Pruning:** If a queen cannot be placed into column ***i***, do not even try to place one onto column ***i+1*** – rather, backtrack to column ***i-1*** and move the queen that had been placed there.

- Using this approach we can reduce the number of potential solutions even more.

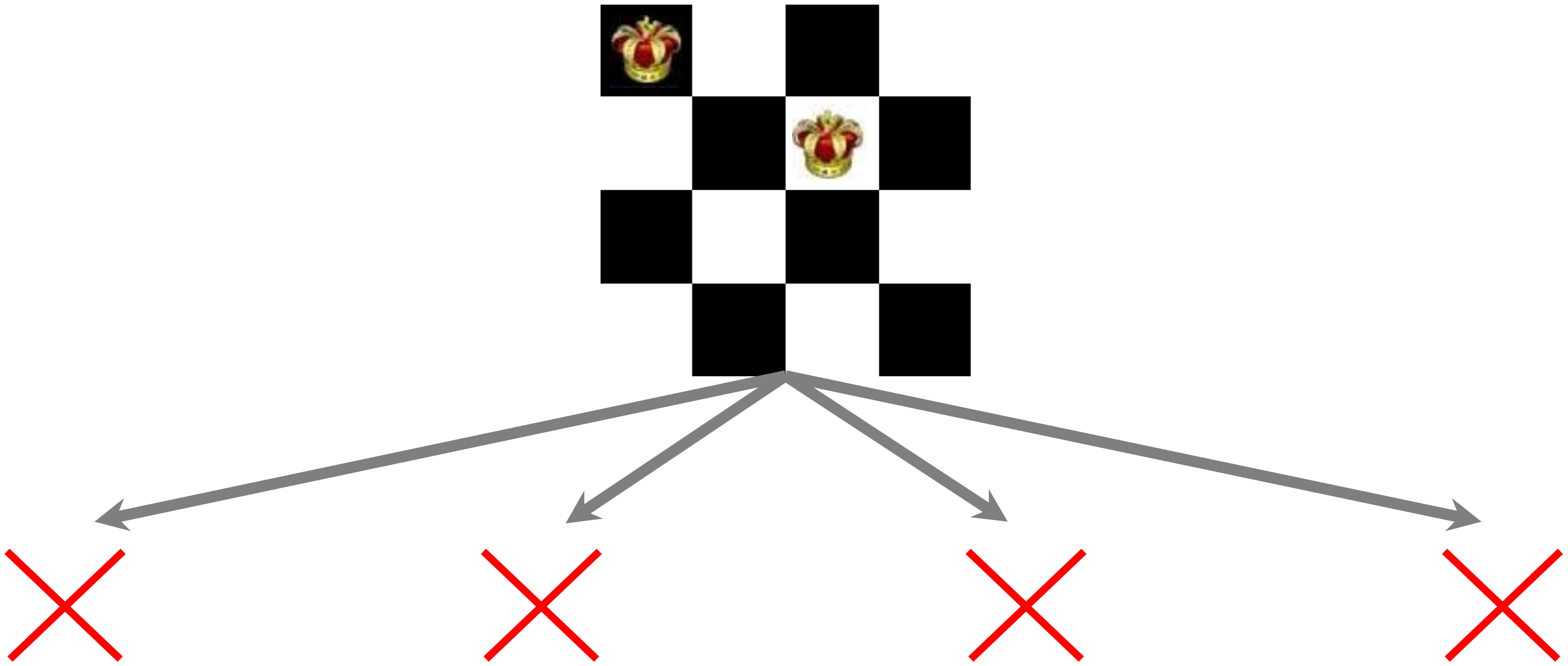
Example- 4 Queens



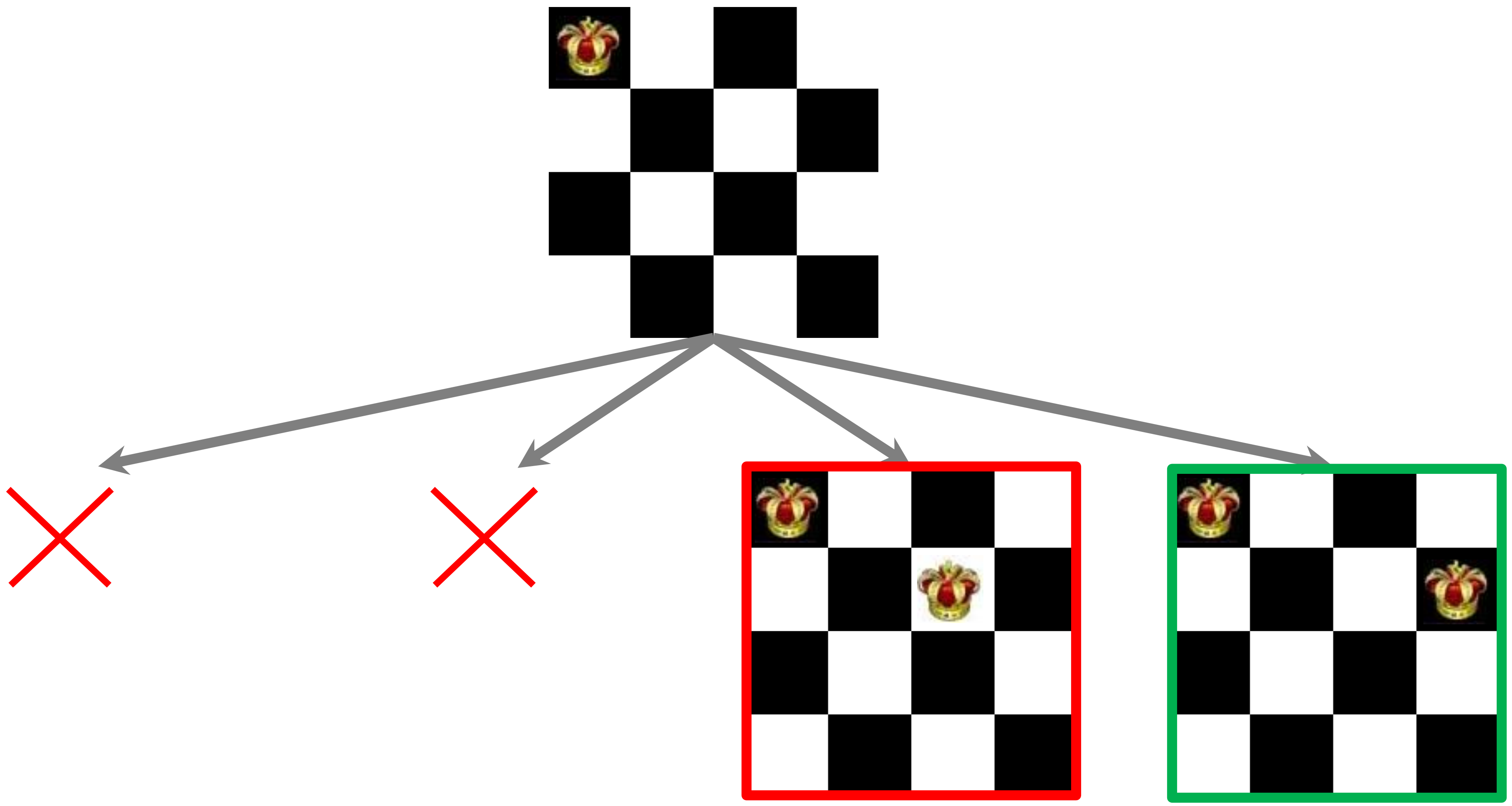
Example- 4 Queens



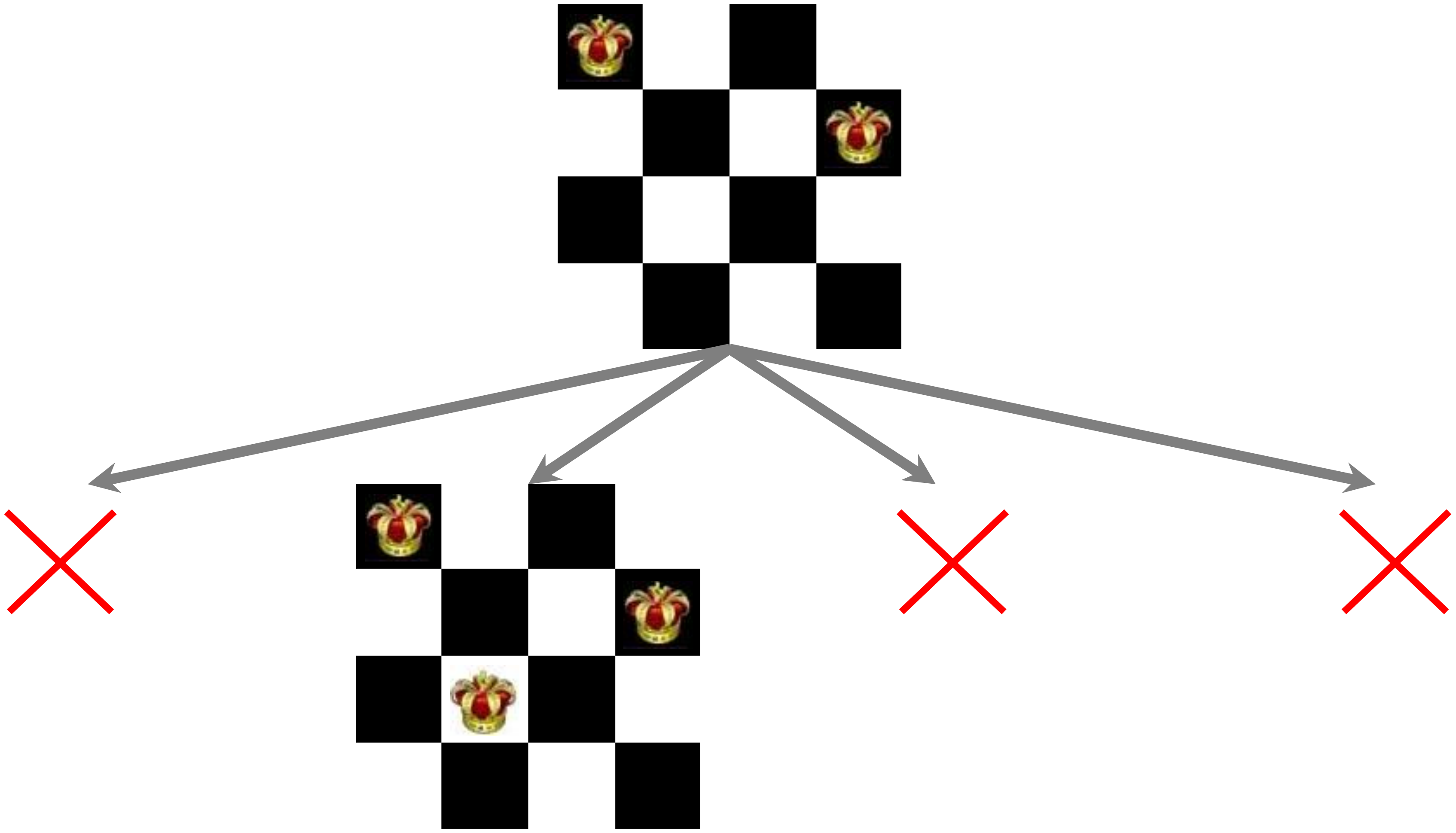
Example- 4 Queens



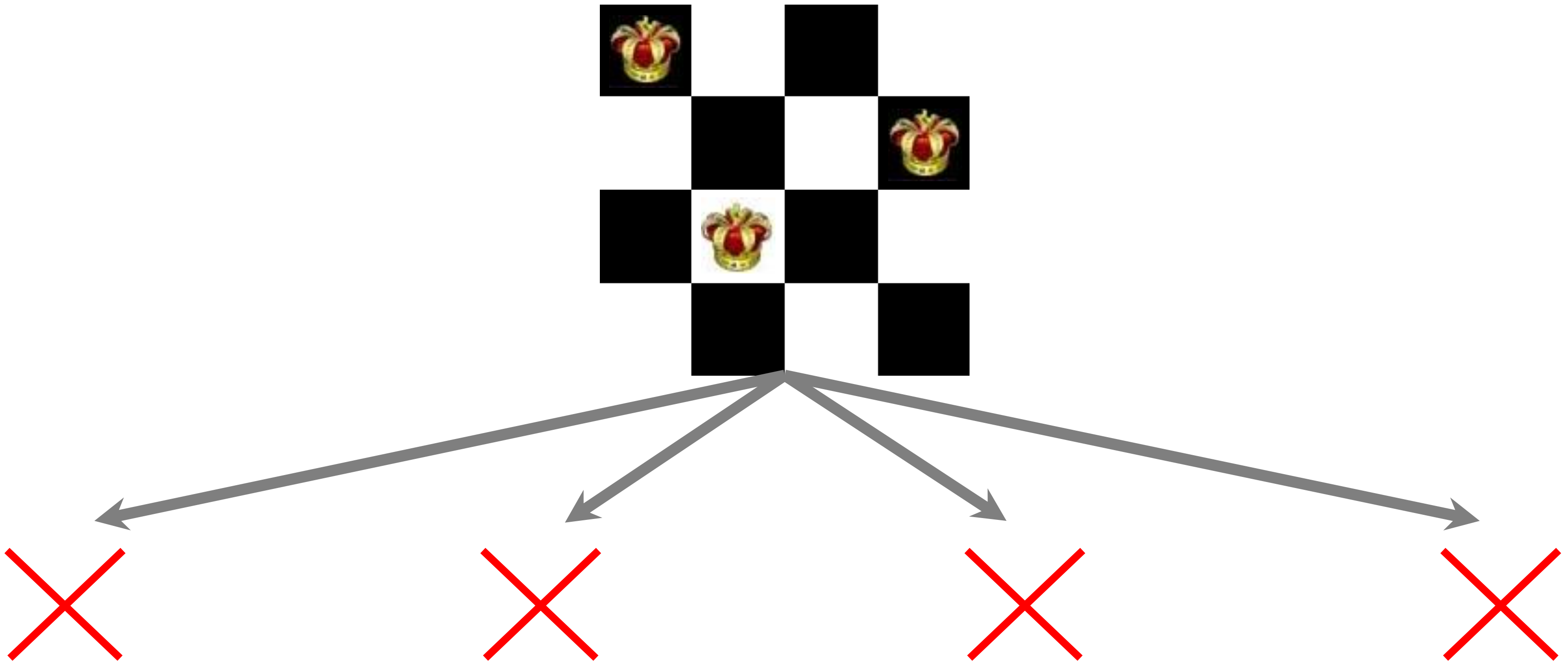
Example- 4 Queens



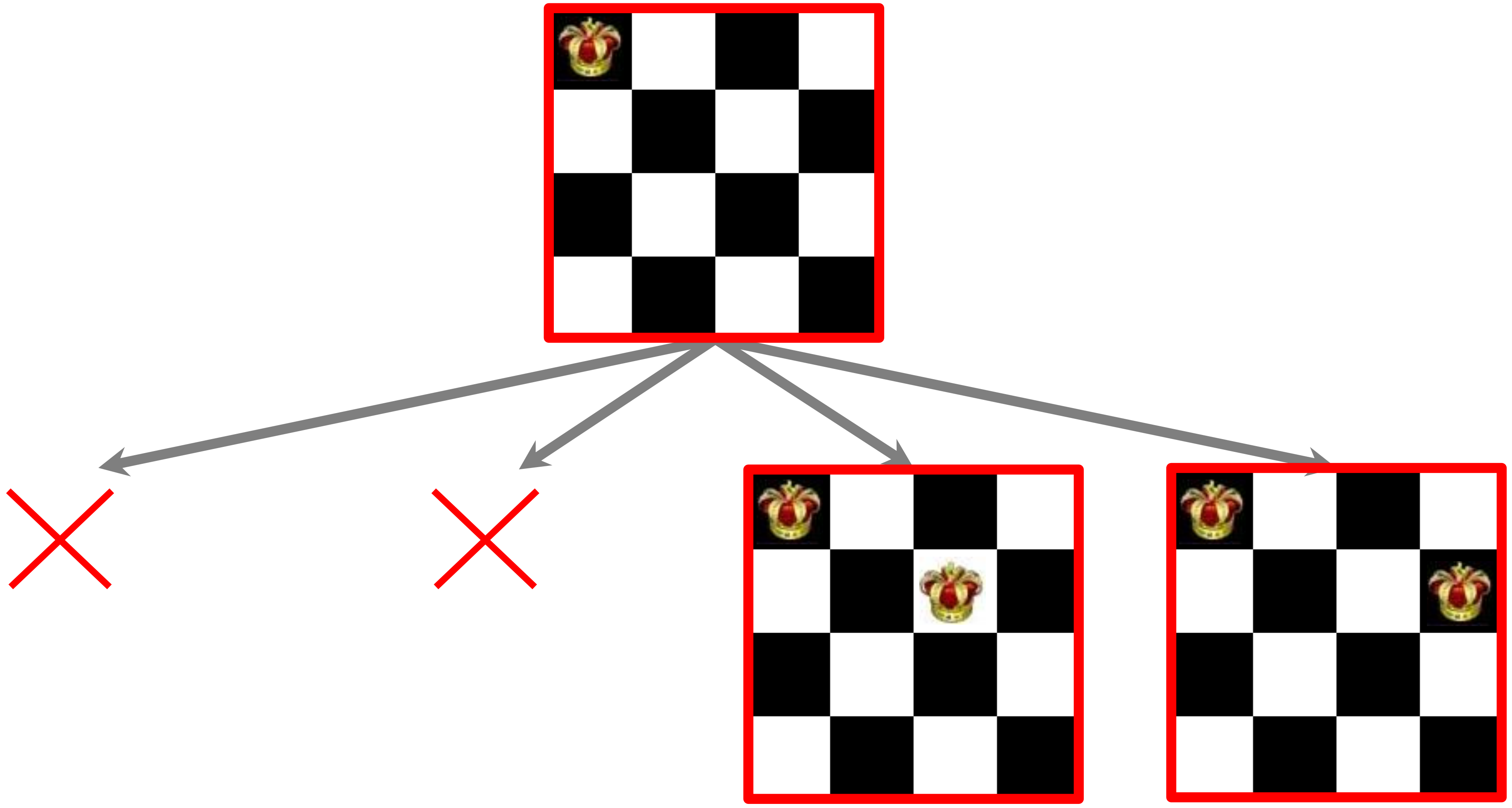
Example- 4 Queens



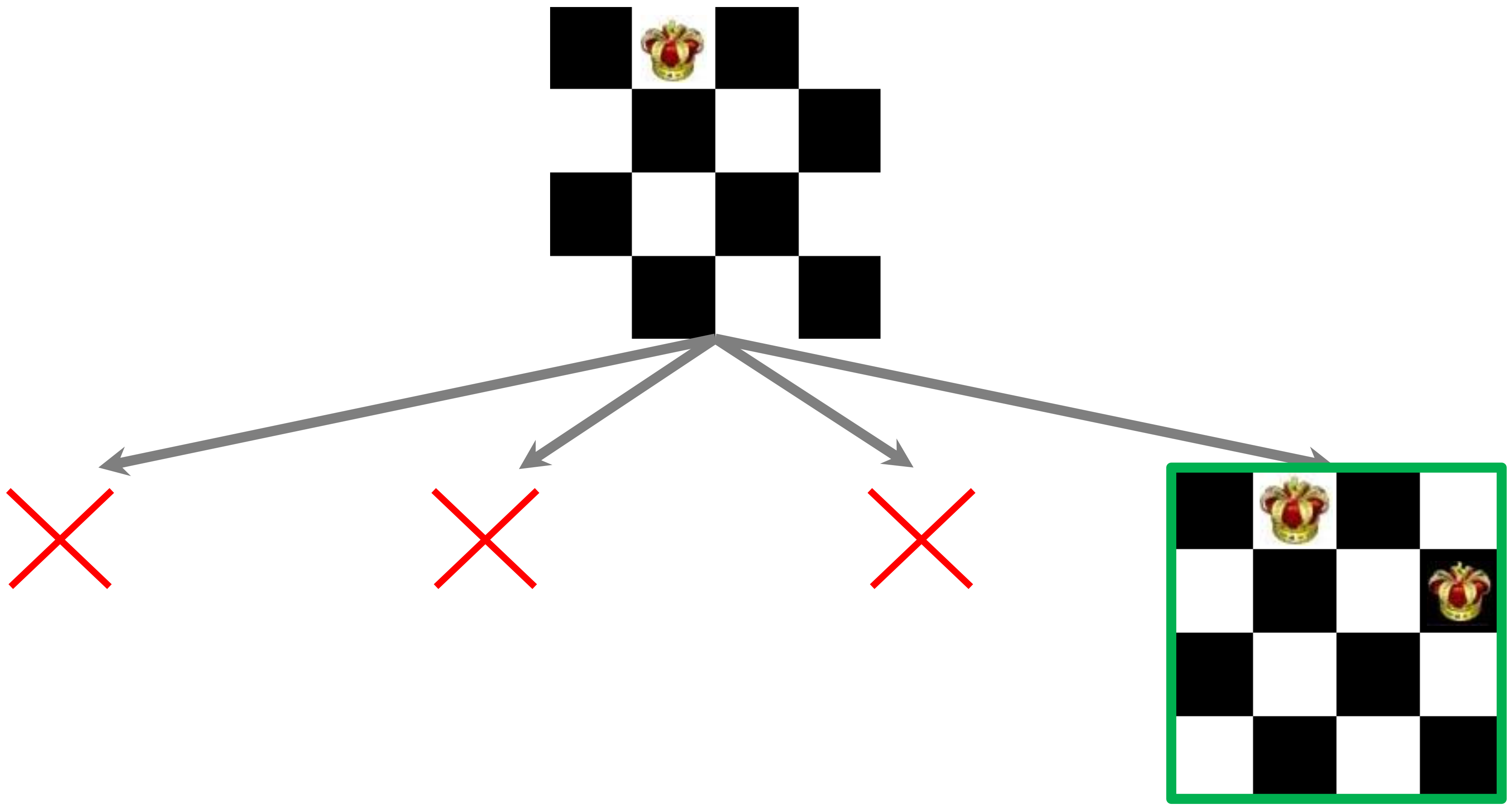
Example- 4 Queens



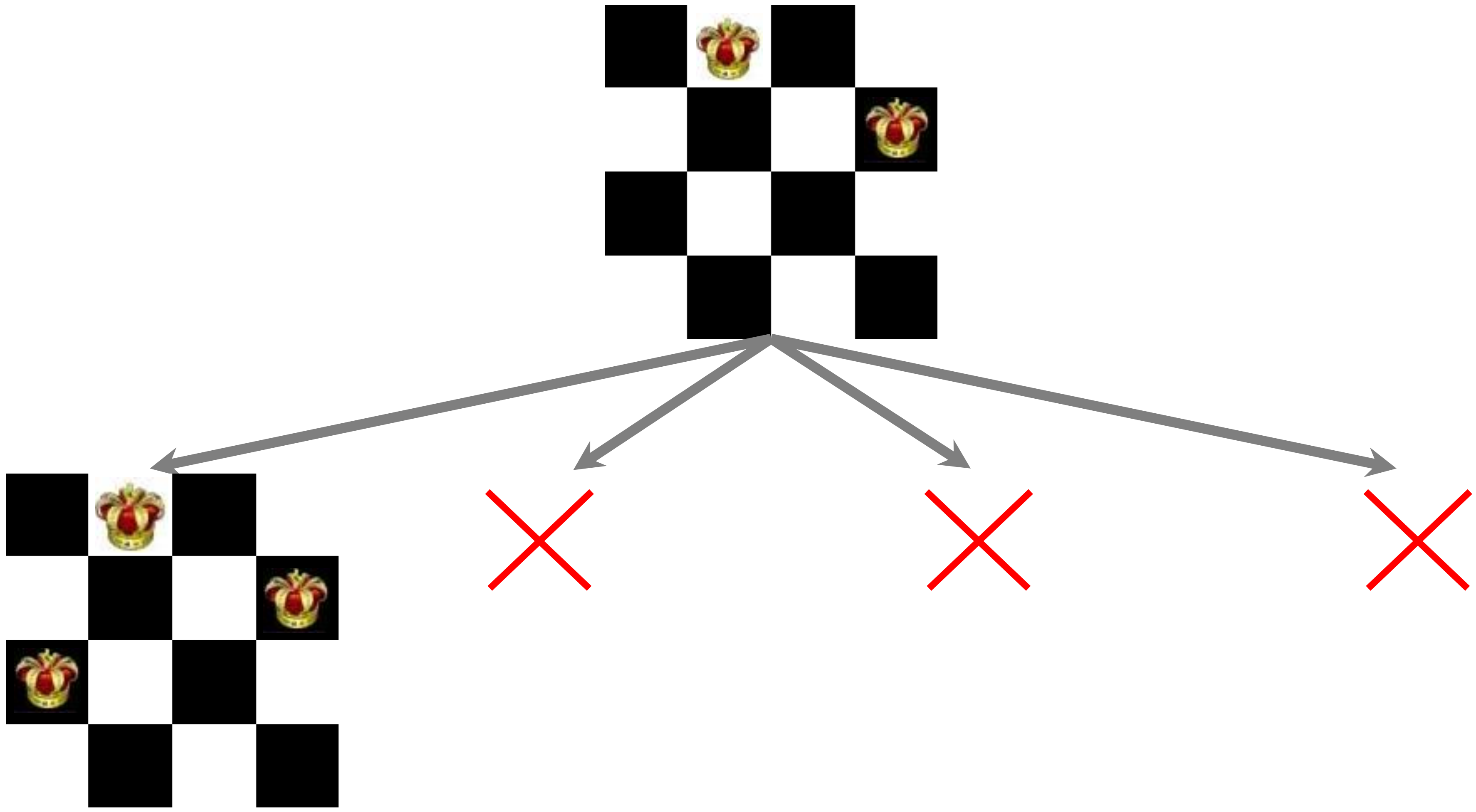
Example- 4 Queens



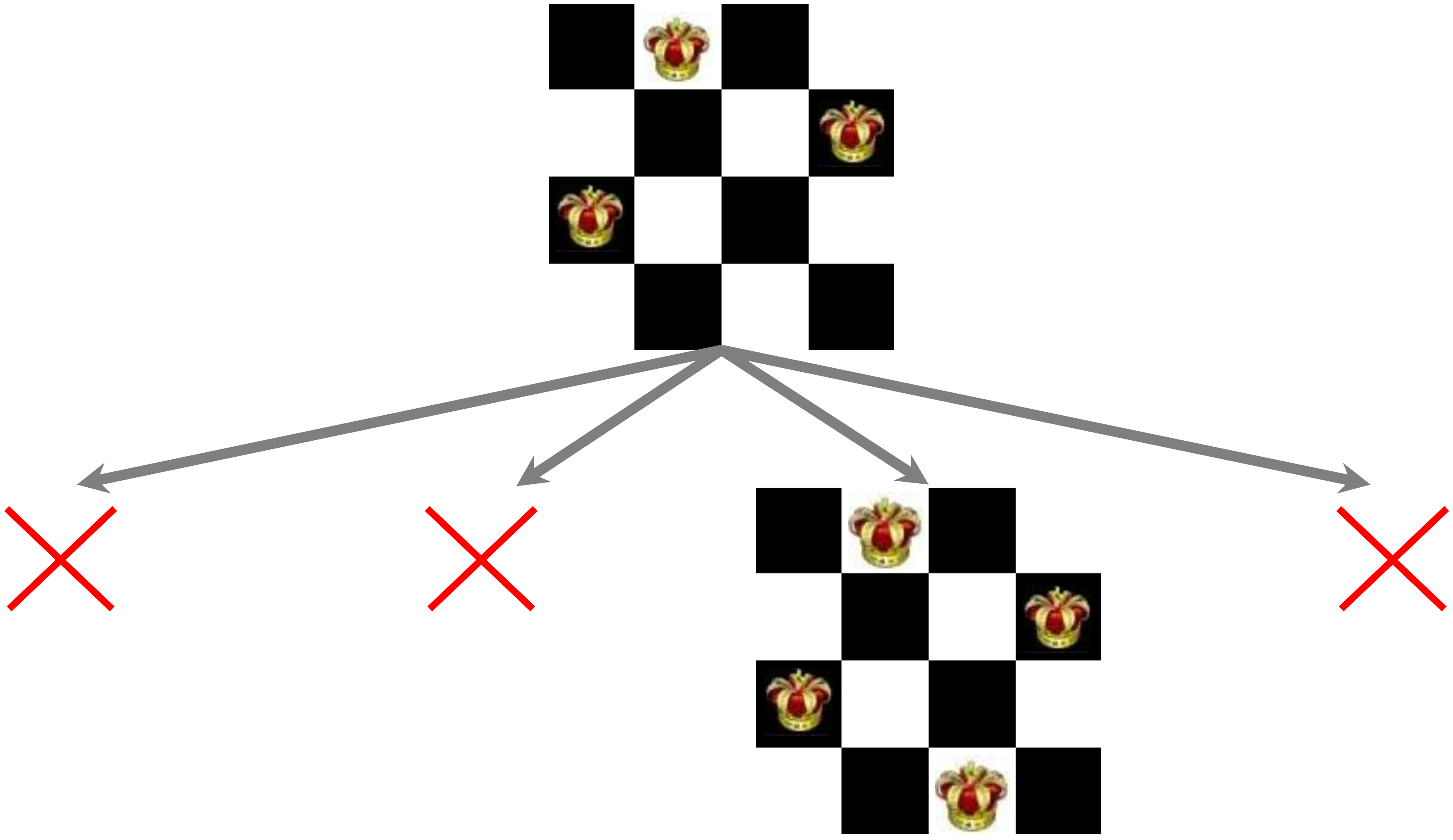
Example- 4 Queens



Example- 4 Queens



Example- 4 Queens



Final Unique Solution

Backtracking Algorithm

1. Place the first queen in the left corner of the table.
2. Save the attacked positions.
3. Move to the next queen (which can only be placed to the next line).
4. Search for a valid position. If there is one, go to step 8.
5. If there is no valid position for the queen, delete it.
6. Move to the previous queen.
7. Go to step 4.
8. Place it to the first valid position.
9. Save the attacked positions.
10. If the queen processed is the last, stop; otherwise go to step 3.

Pseudocode

```
PutQueen(row){
    for every position col on the same row
        if position col is available
            place the next queen in position col
                if (row < 8)
                    PutQueen(row + 1)
                else  "success"
     remove the queen from position col
}
```

Backtracking

Sum Of Subsets

Sum Of Subsets Problem

- The problem is to find a subset of a given set **$S=\{s_1, s_2, \ldots, s_n\}$** of ***n*** positive integers whose sum is equal to a given positive integer ***d***.

- **Observation:** It is convenient to sort the set's elements in increasing order, **$s_1 < s_2 < \ldots < s_n$**. And each set of solutions don't need to be necessarily of fixed size.

- **Example:** For S={3, 5, 6, 7} and d=15, the solution is:

  Solution = {3, 5, 7}.

Formulation

- We will assume a **binary state space tree**.
- The nodes at depth 1 are for including (yes, no) item 1, the nodes at depth 2 are for item 2, etc.
- The left branch includes $\mathbf{s}_i$ and the right branch excludes $\mathbf{s}_i$.
- The nodes contain the sum of the integers included so far.

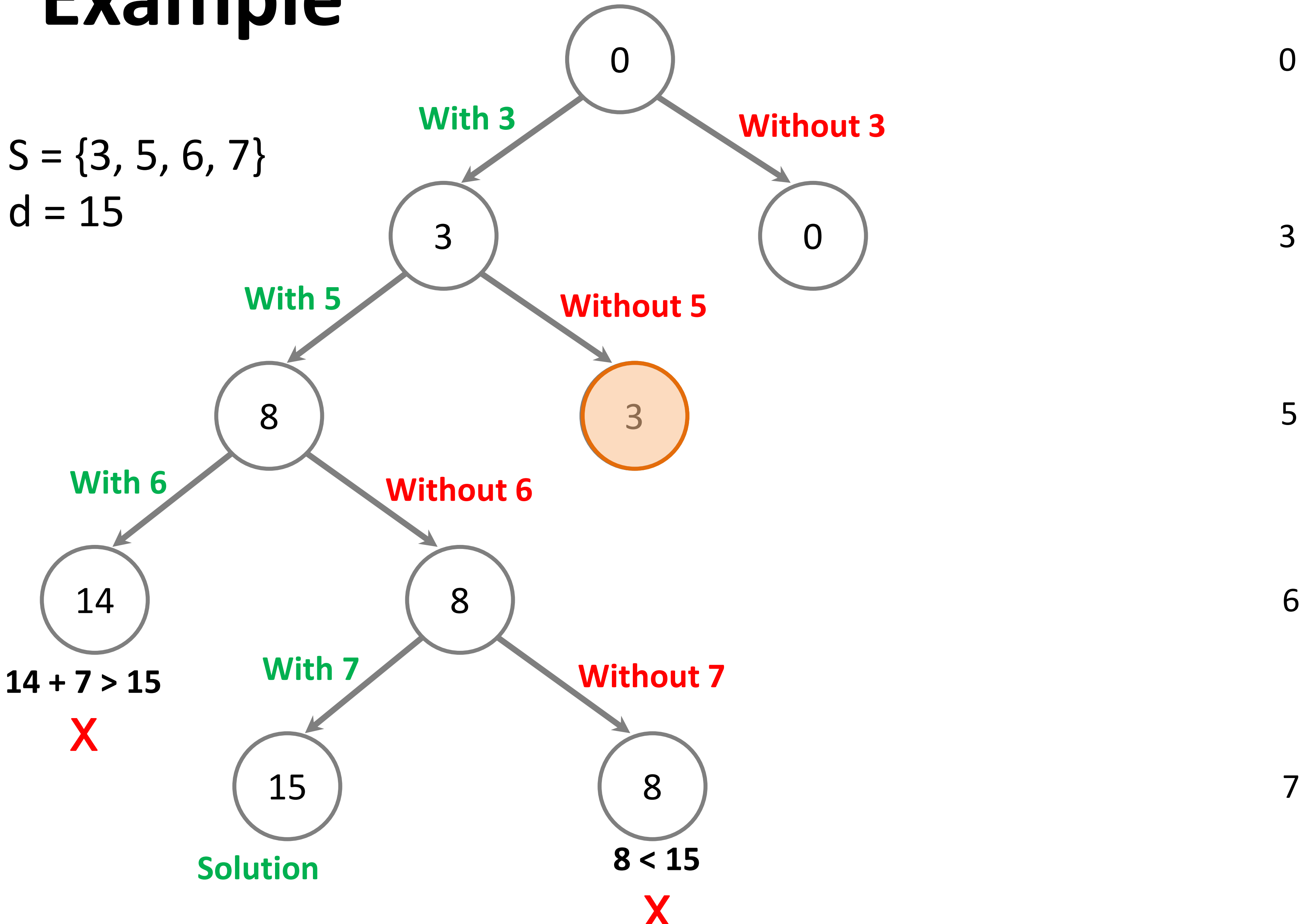
Sum Of Subsets Using Backtracking

- For *n* positive integers in a set, there are 2^n possible subsets.

- Using backtracking concept we don’t have to observe all the subsets rather we can prune the tree rooted at the node which will not give any solution and backtrack to its previous node.

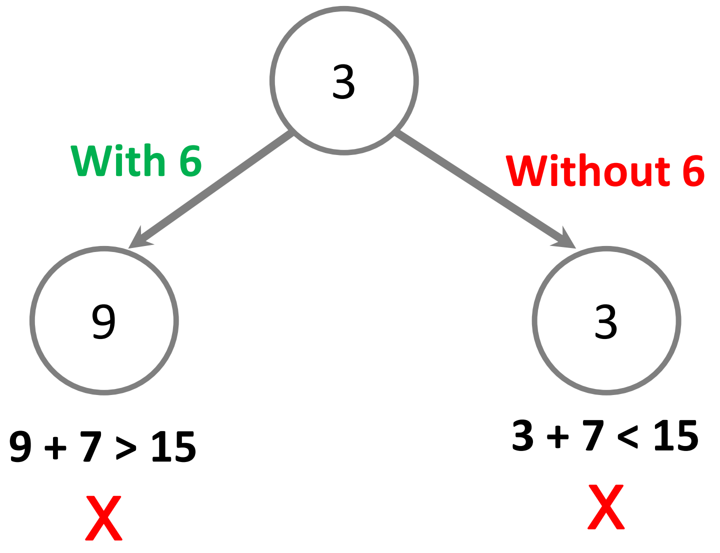
Example

S = {3, 5, 6, 7}
d = 15



Example

S = {3, 5, 6, 7}
d = 15



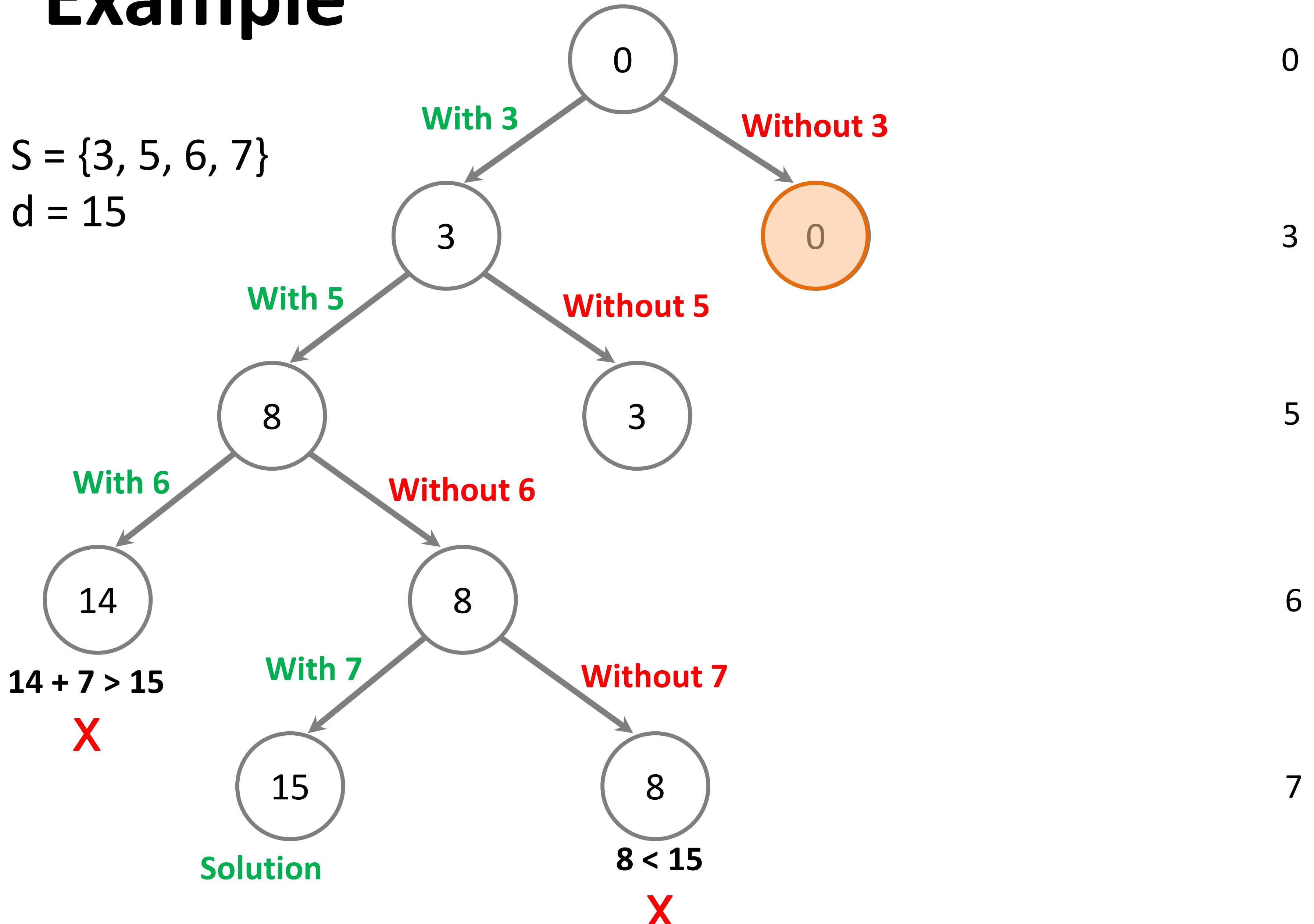
5

6

7

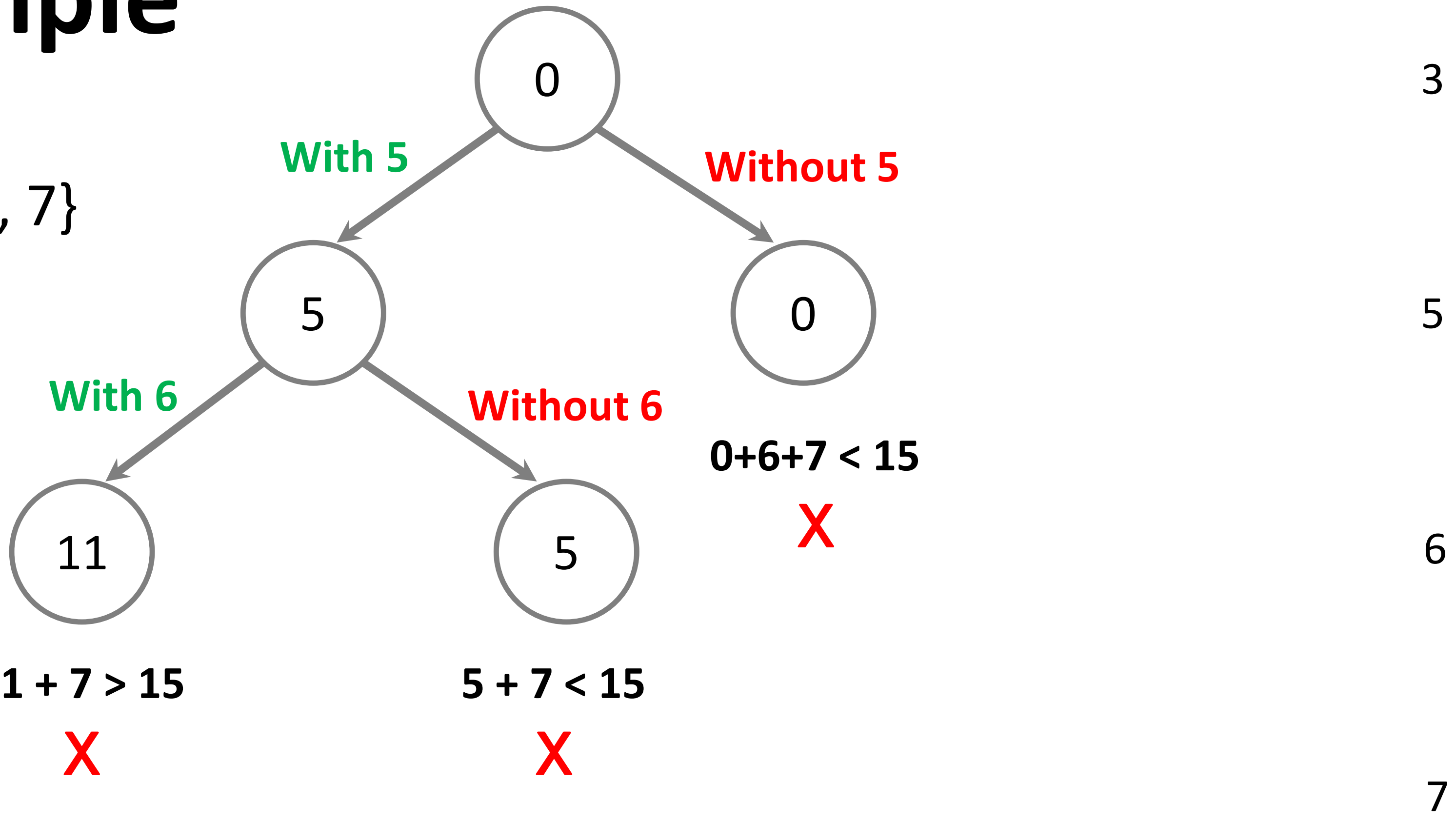
Example

S = {3, 5, 6, 7}
d = 15



Example

S = {3, 5, 6, 7}
d = 15



Backtracking

Graph Coloring

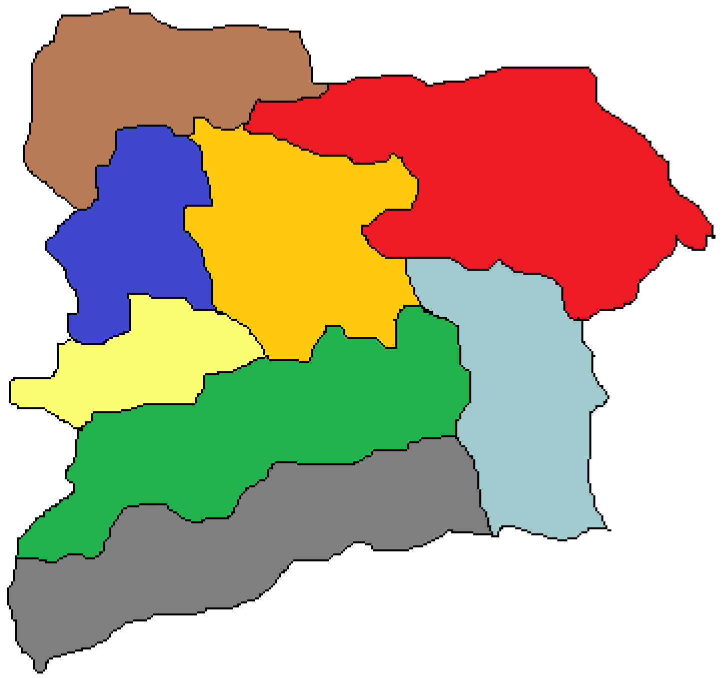
Graph Coloring Problem

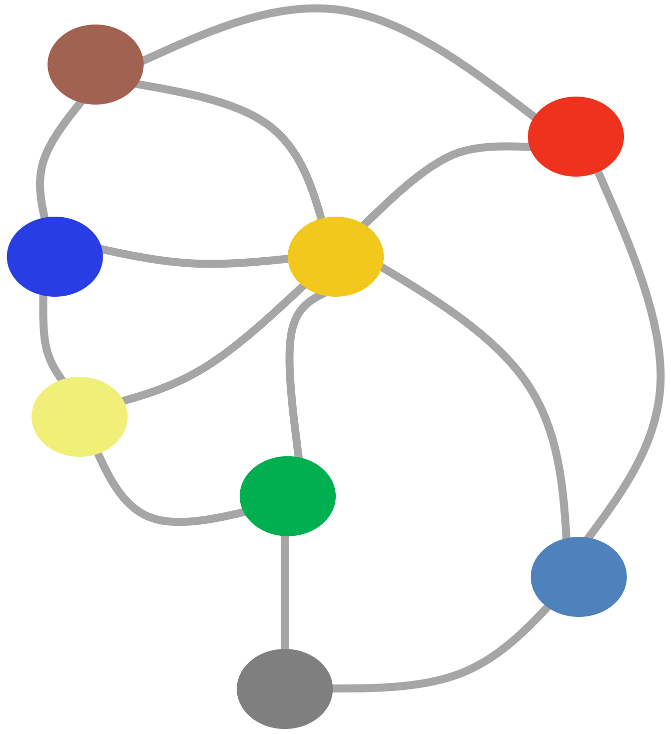
- Graph coloring problem of states each vertex in a graph in such a way that no two adjacent vertices have same color. This problem is called as *m-coloring problem*. This is also called a *vertex coloring*.

- If the degree of a given graph is *d* then we can color it with *d+1* color.

- A graph is a planer if it can be drawn such that no edges cross when drawn on plane surface A map can easily be converted to a planner graph.

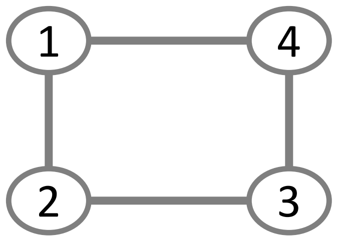
Map Coloring

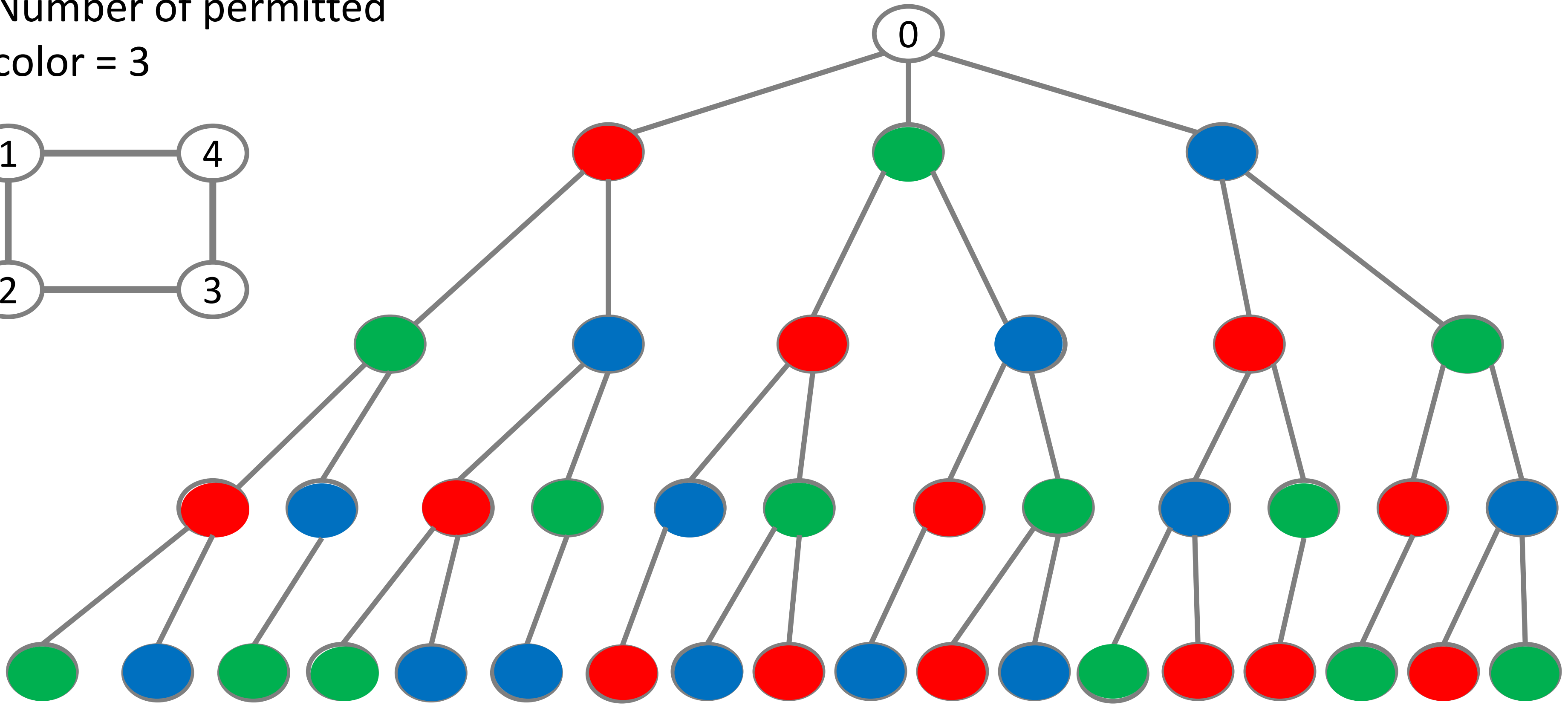




Graph Coloring - Example

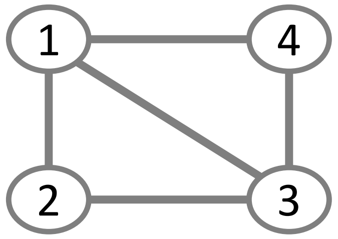
Number of permitted color = 3

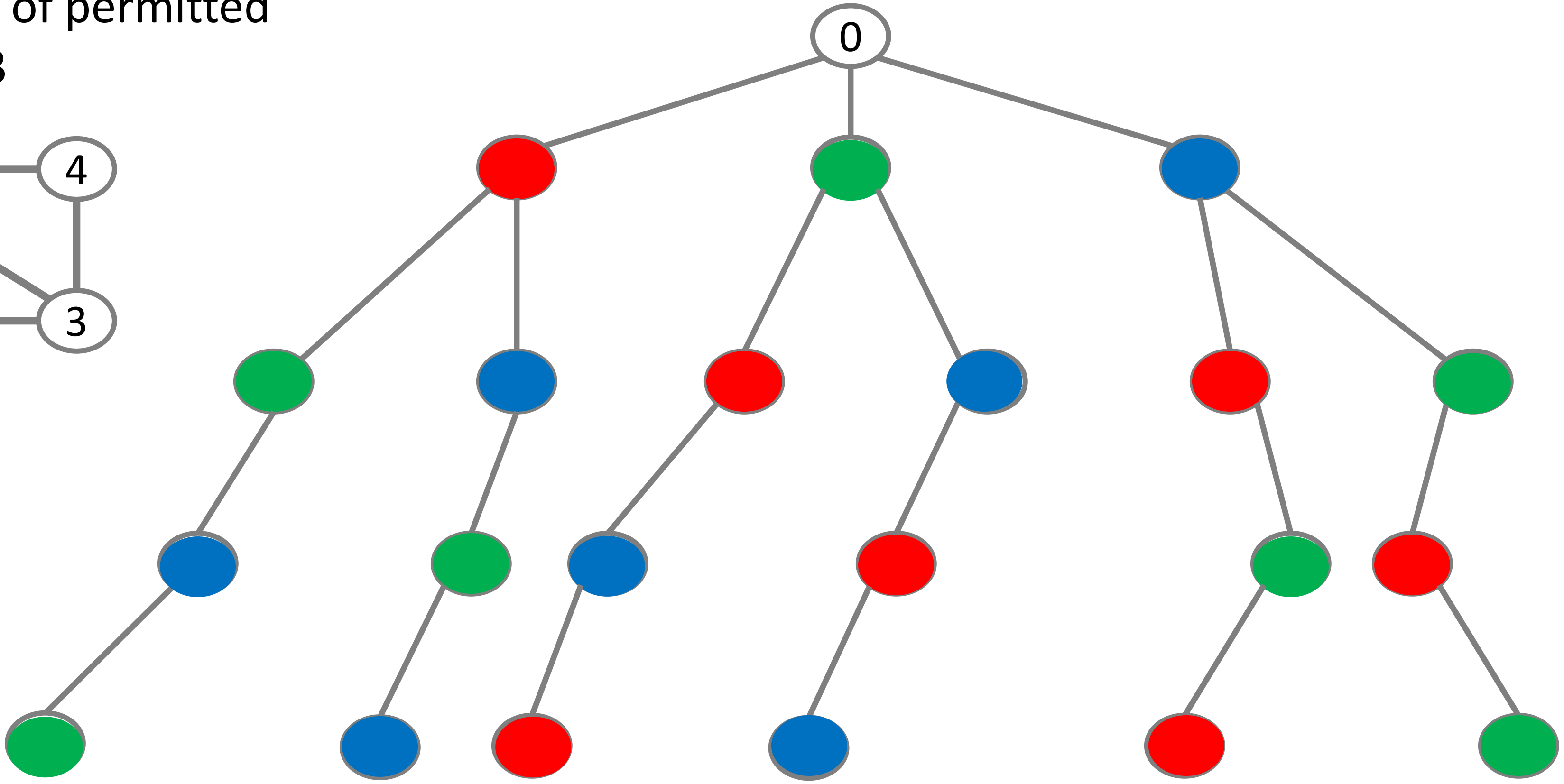




Graph Coloring - Example

Number of permitted color = 3





Backtracking

Hamiltonian Circuit

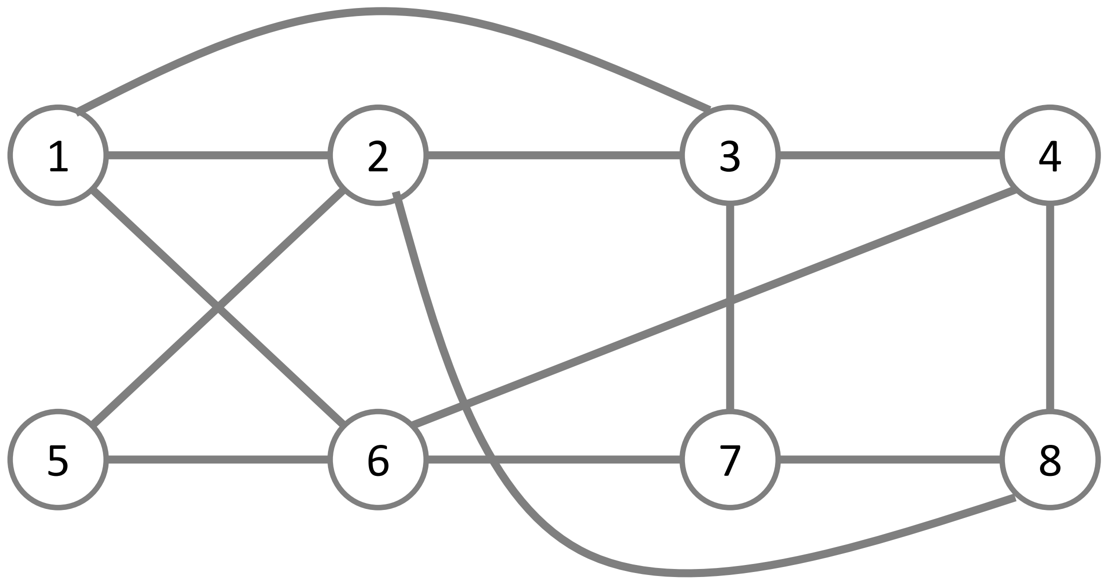
Hamiltonian Circuit Problem

- This problem is concern about finding a Hamiltonian circuit in a given graph.

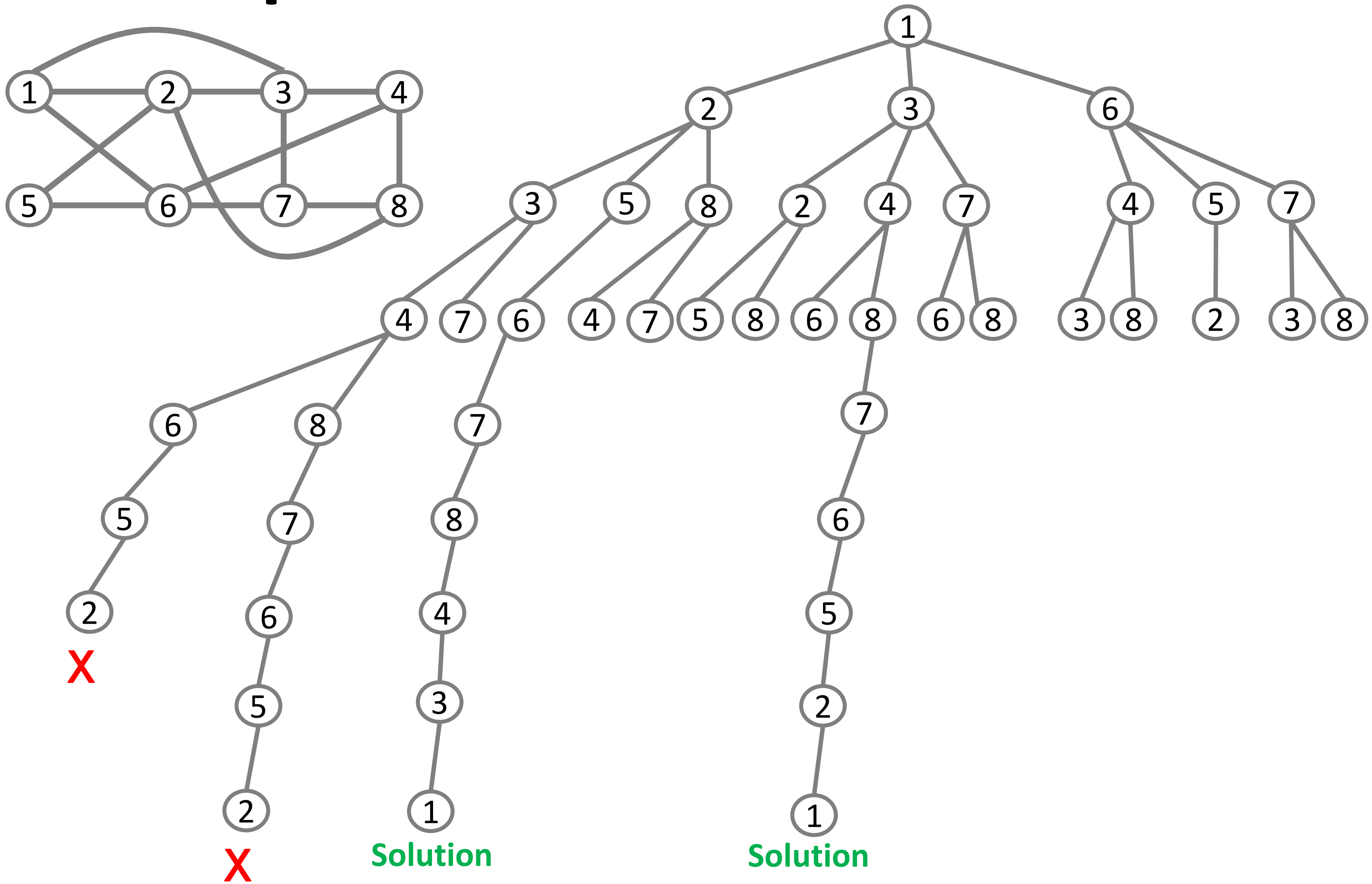
- **Hamiltonian circuit:** It is defined as a cycle that passes to all the vertices of the graph exactly once except the starting and ending vertices that is the same vertex.

Example

How to find Hamiltonian cycle by using backtracking in given graph.



Example


X
X
Solution
Solution