UNIT - V

BACK TRACKING

OUTLINE

- **Backtracking**
  - n-Queens Problem

Backtracking

- Backtracking is mainly used to solve problems which have more than one solutions.
- The principal idea is to construct solutions by considering one component at a time and evaluate such partially constructed candidates as follows.
- If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option **for the next component**.
- If there is no legitimate option for the next component, no alternatives for *any* remaining component need to be considered.
- In this case, **the algorithm backtracks** to replace the last component of the partially constructed solution with its next option.

Backtracking

- This kind of processing is often implemented by constructing a tree of choices being made, called the **state-space tree.**

- Its **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.

- A node in a state-space tree is said to be ***promising*** if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise, it is called ***nonpromising***.

Backtracking

- **Leaves** represent either **non promising dead ends** or **complete solutions** found by the algorithm.

- If the current node turns out to be **non promising**, the algorithm **backtracks to the node's parent** to consider the next possible option for its last component;

- if there is no such option, **it backtracks one more level up** the tree, and so on.

- Finally, if the algorithm reaches a complete solution to the problem, it either stops (if just one solution is required) or continues searching for other possible solutions.

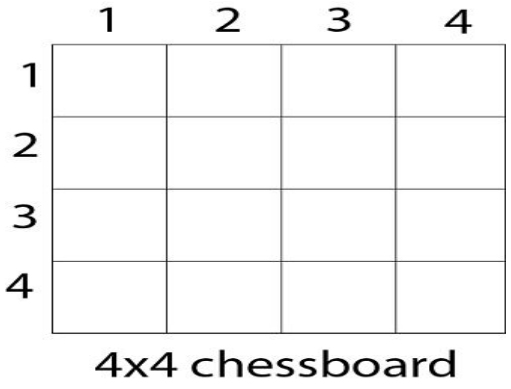
n Queens problem

N – Queens problem:

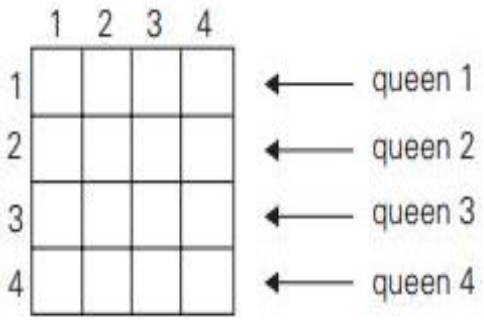
- Problem statement : The n-queens problem is to place n queens on an n × n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.
- We solve n queens problem by backtracking technique.
- It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.
- For 4 queen problem we can get 2 distinct solutions.
- For 8 queen problem we can get 92 distinct solutions.

4 queen problem:

- So let us consider the four-queens problem and solve it by the backtracking technique. Since each of the four queens has to be placed in its own row, all we need to do is to assign a column for each queen on the board presented in the following figure.

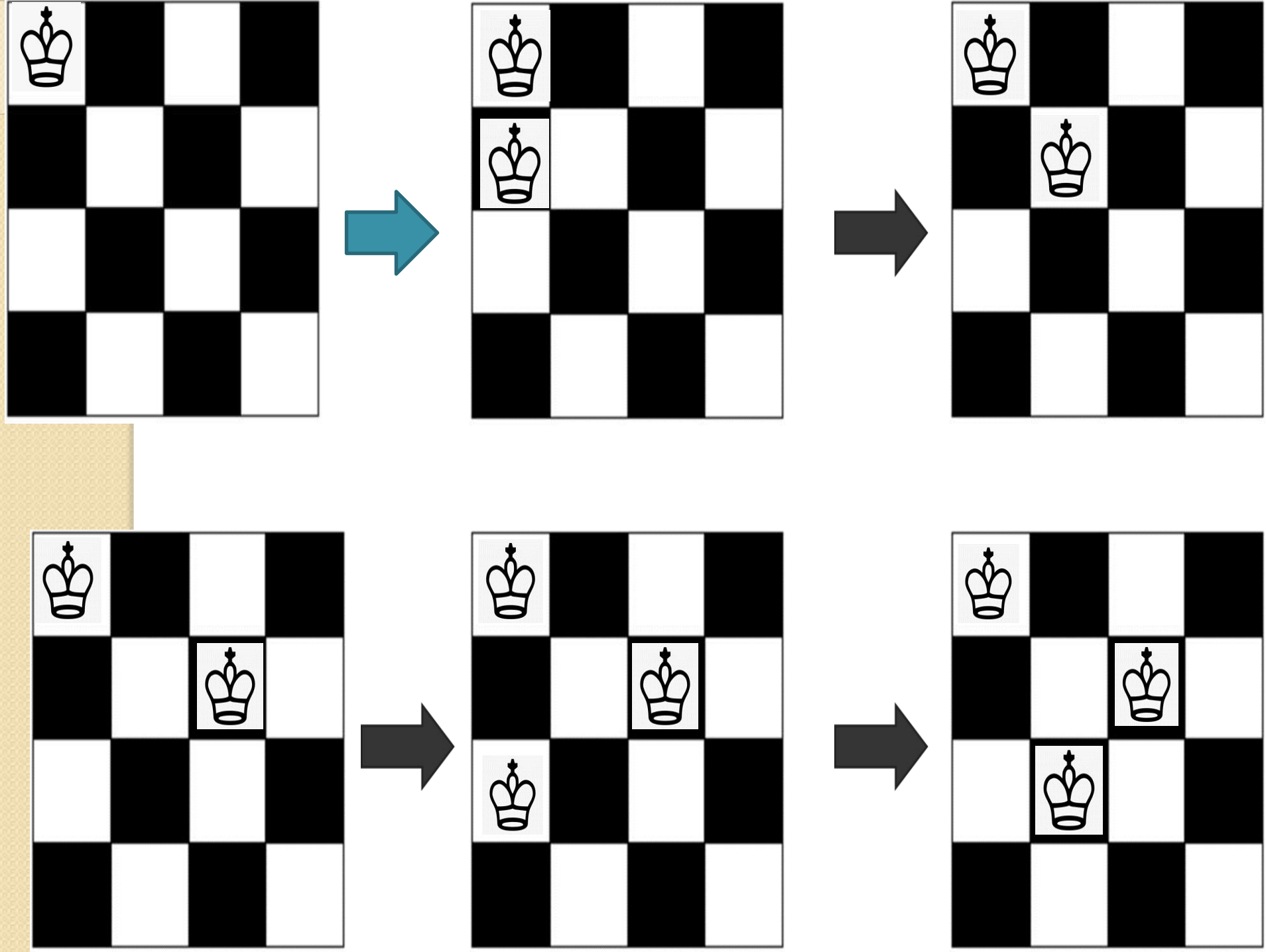


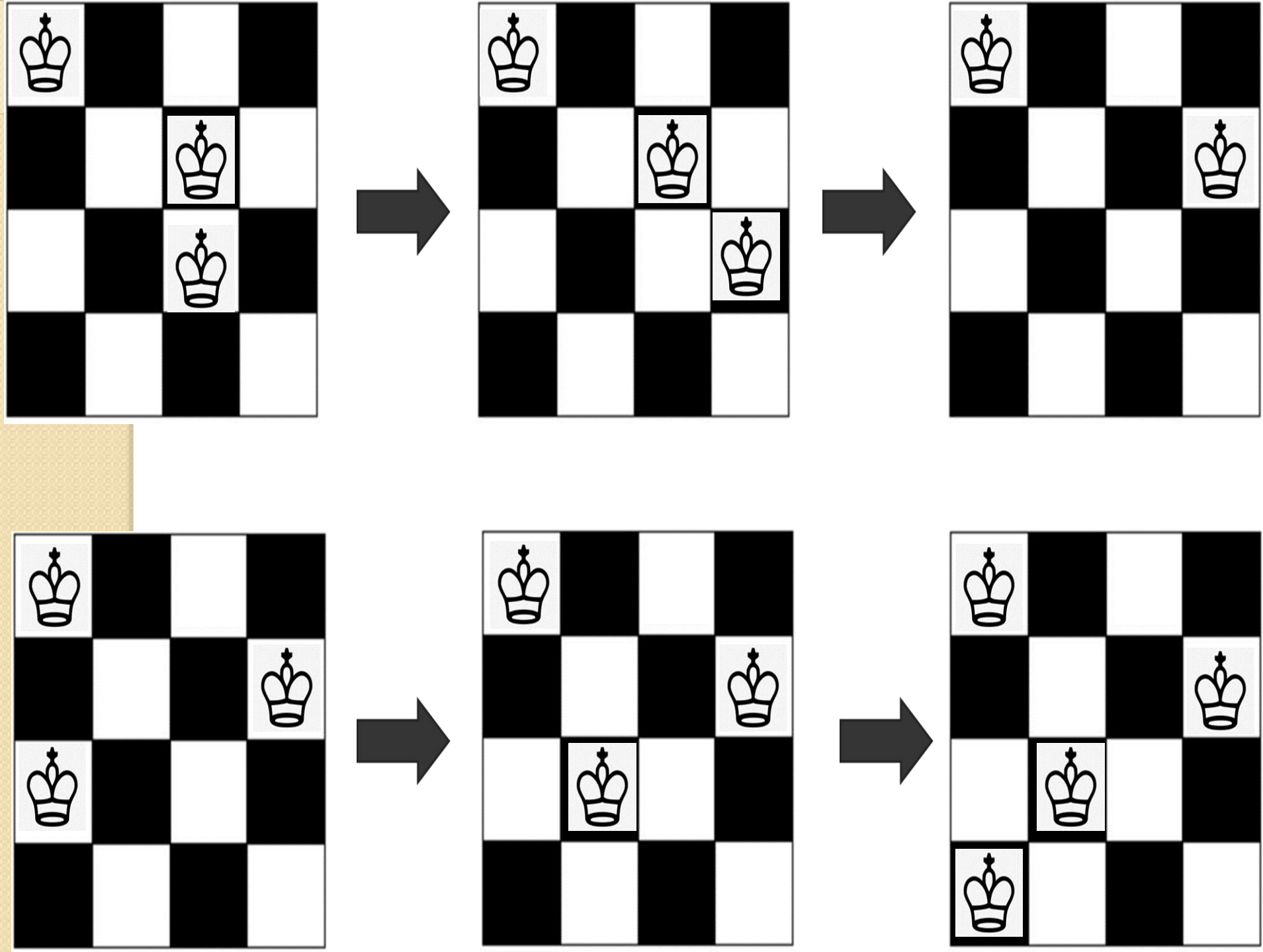
4x4 chessboard

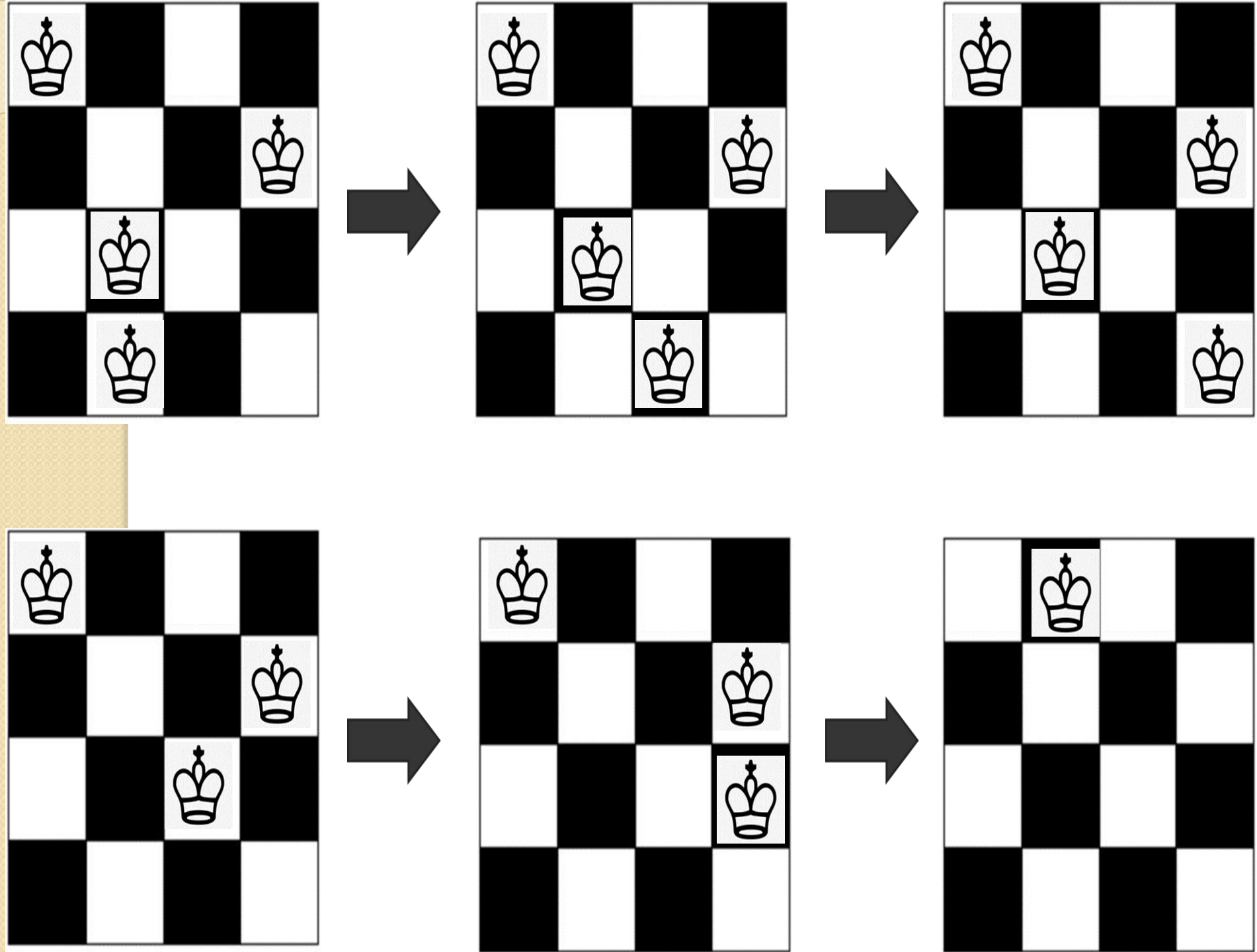


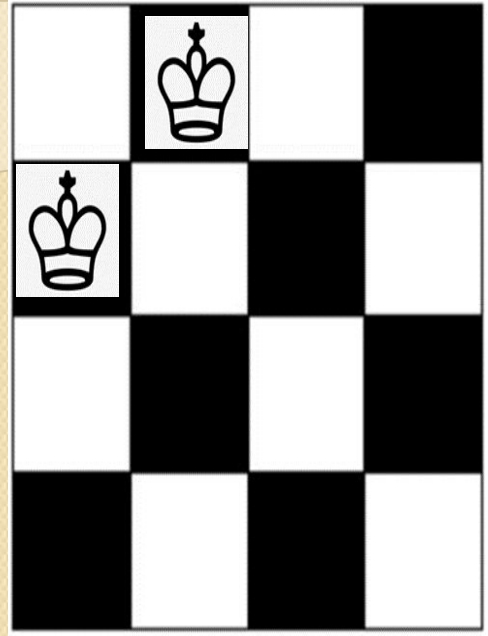
- We assume here that first queen is to placed in first row, second queen is to placed in second row and so on.. Such that there is no attack

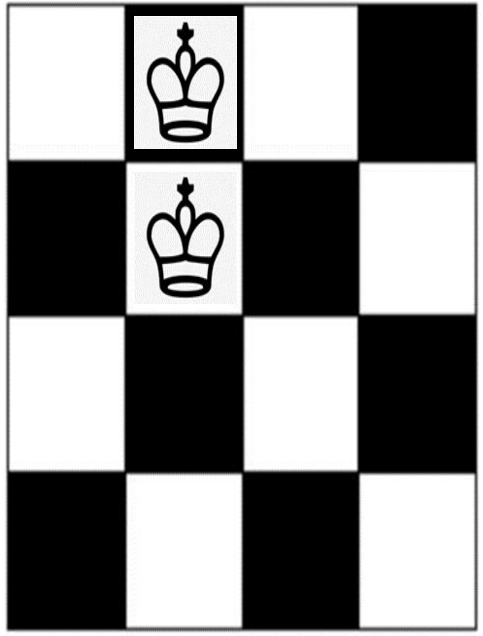
- We start with the empty board and then place queen 1 in the first possible position of its row, which is in column 1 of row 1.
- Then we place queen 2, after trying unsuccessfully for columns 1 and 2, in the first acceptable position for it, which is square (2, 3), the square in row 2 and column 3. This proves to be a dead end because there is no acceptable position for queen 3.
- So, the algorithm backtracks and puts queen 2 in the next possible position at (2, 4). Then queen 3 is placed at (3, 2), which proves to be another dead end.
- The algorithm then backtracks all the way to queen 1 and moves it to (1, 2). Queen 2 then goes to (2, 4), queen 3 to (3, 1), and queen 4 to (4, 3), which is a solution to the problem.
- If other solutions need to be found the algorithm can simply resume its operations at the leaf at which it stopped.

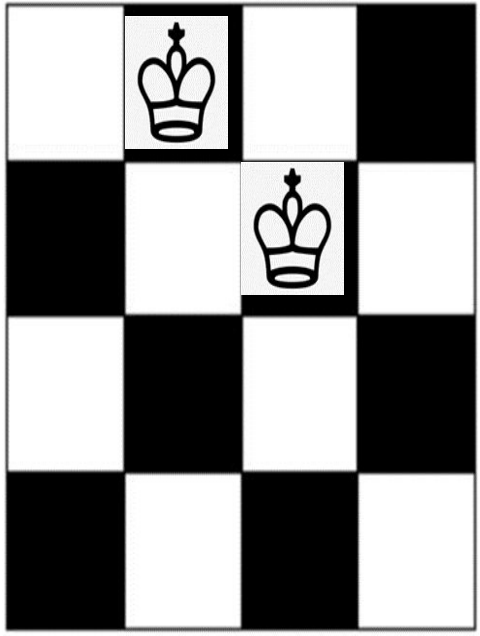
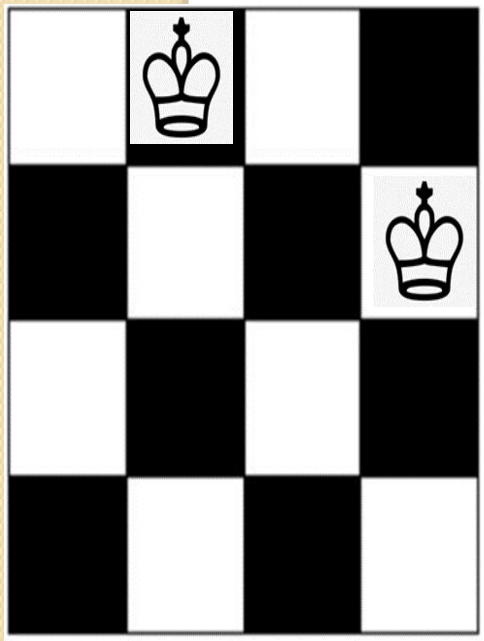


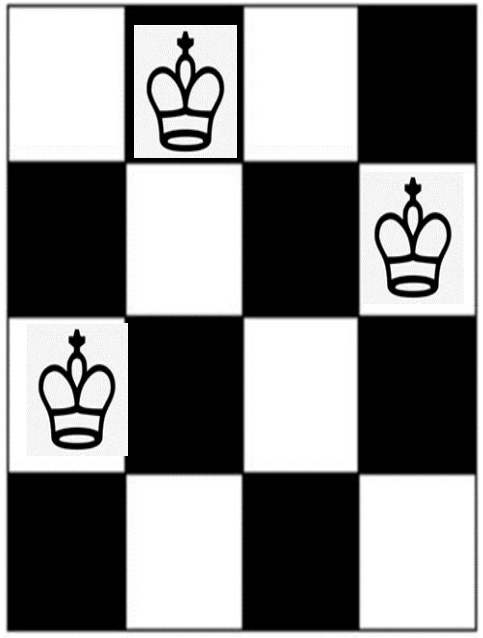


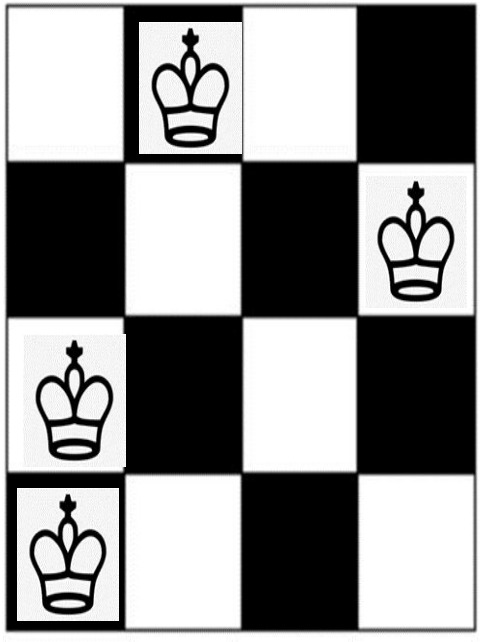


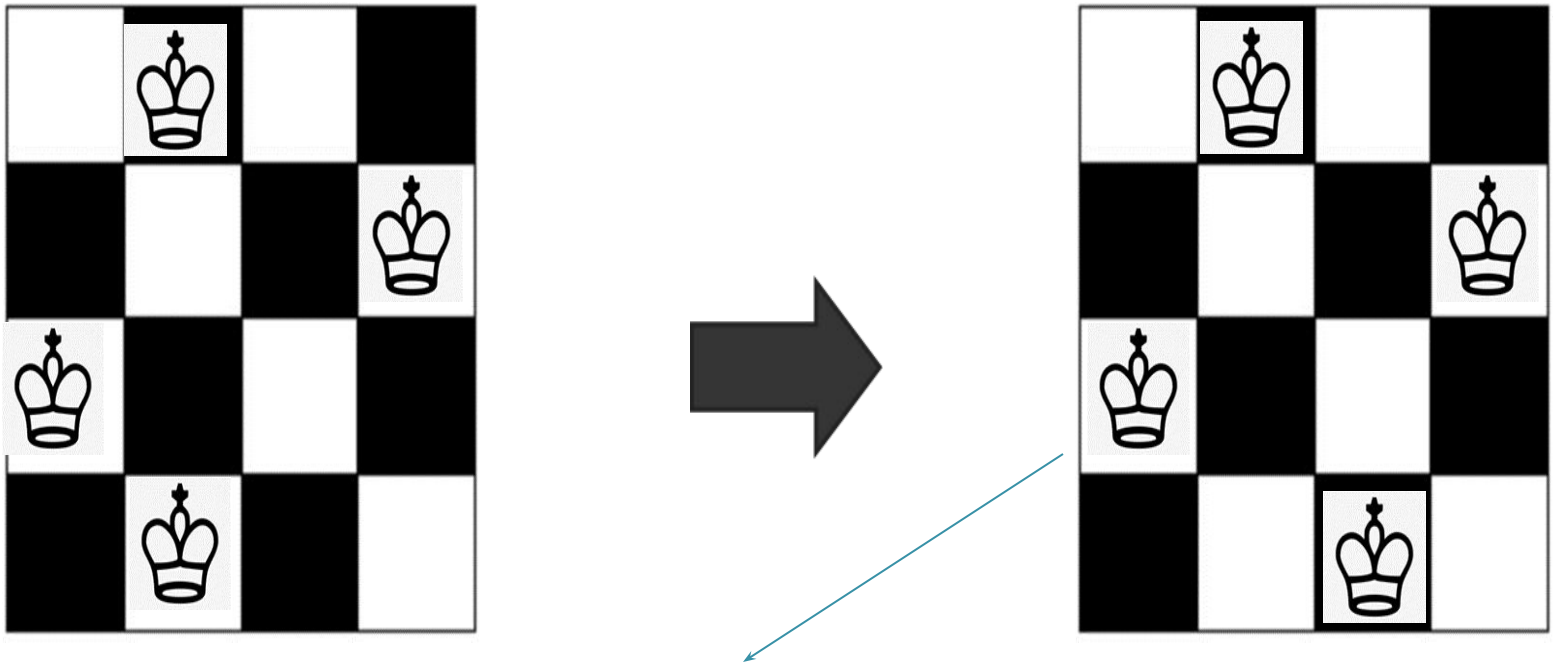








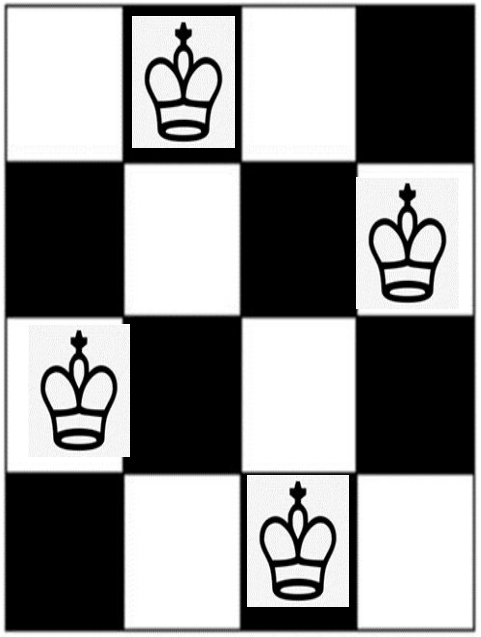


Solution here is : (2,4,1,3)

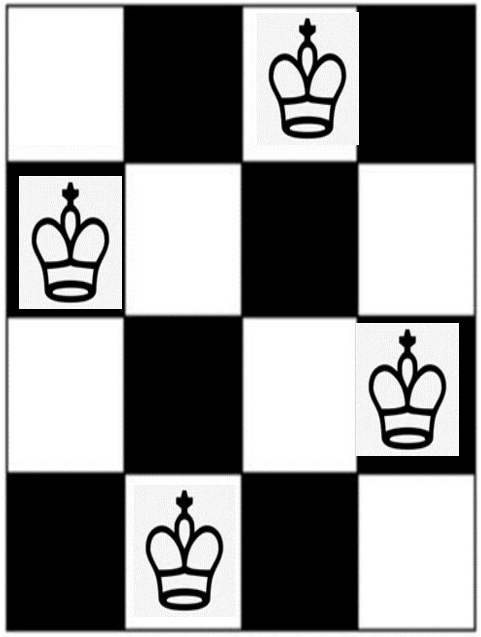
Which indicates:

- ✔**First queen is placed in I row second column**
- ✔**Second queen is placed in II row fourth column**
- ✔**Third queen is placed in III row first column**
- ✔**Fourth queen is placed in IV row third column**

Two solutions are:



Solution 1 is : (2,4,1,3)



Solution 2 is : (3,1,4,2)

State-Space Tree of the 4-Queens Problem

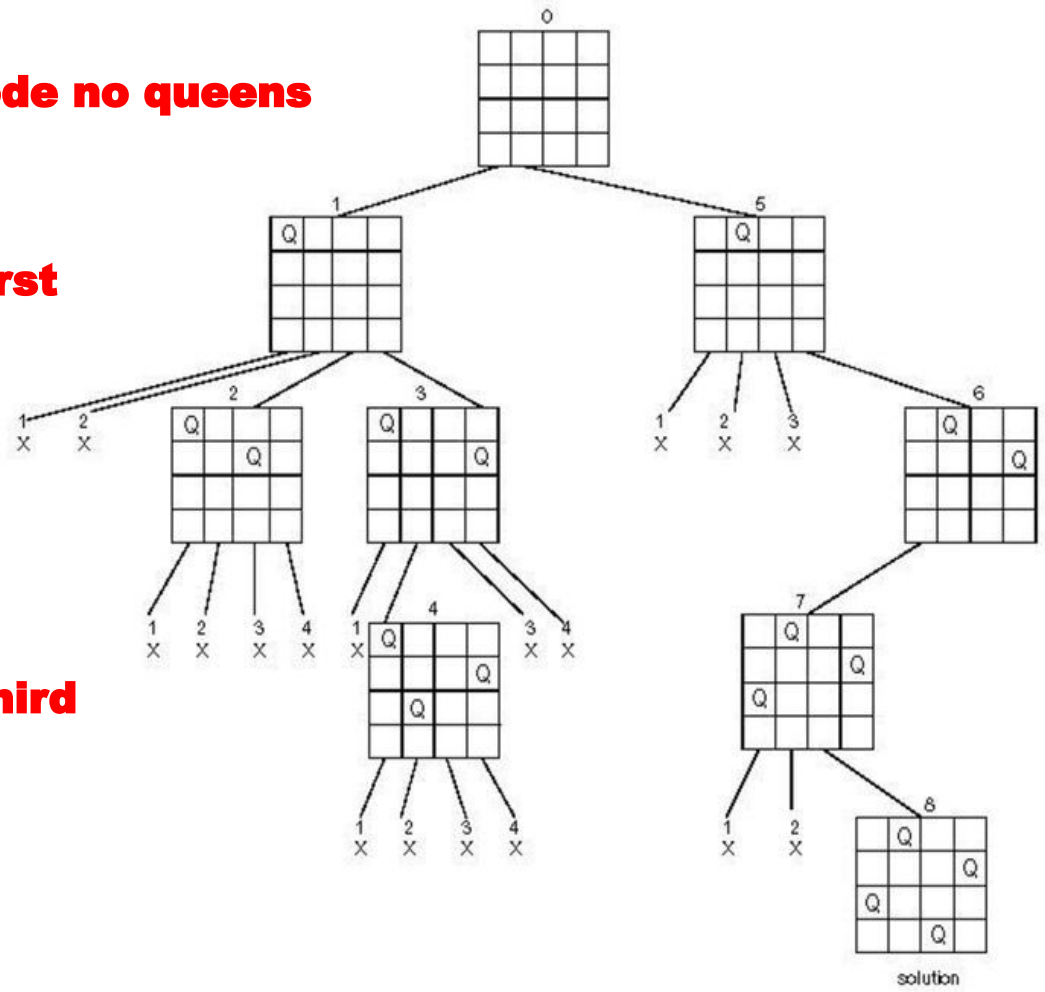
Node 0: root node no queens are placed

Level 1: Placing First queen

Level 2: Placing Second queen

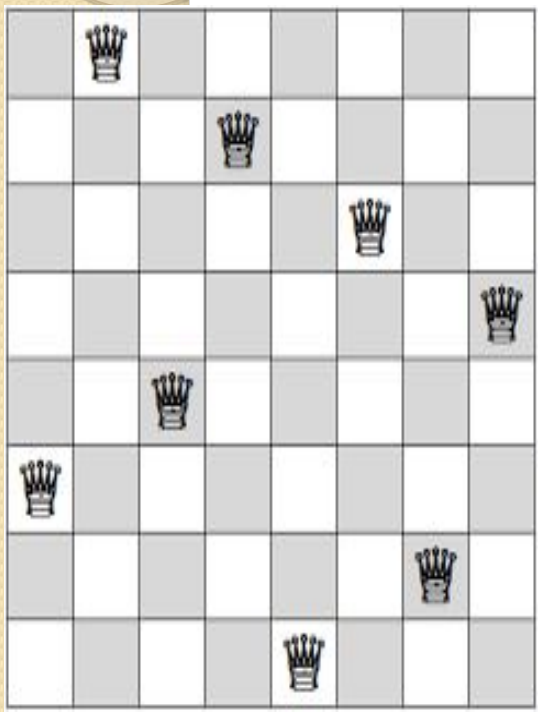
Level 3: Placing Third queen

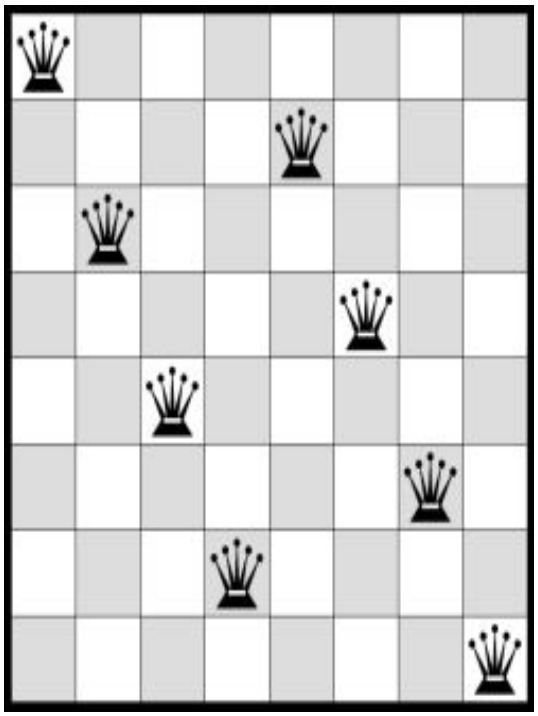
Level 4: Placing Fourth queen

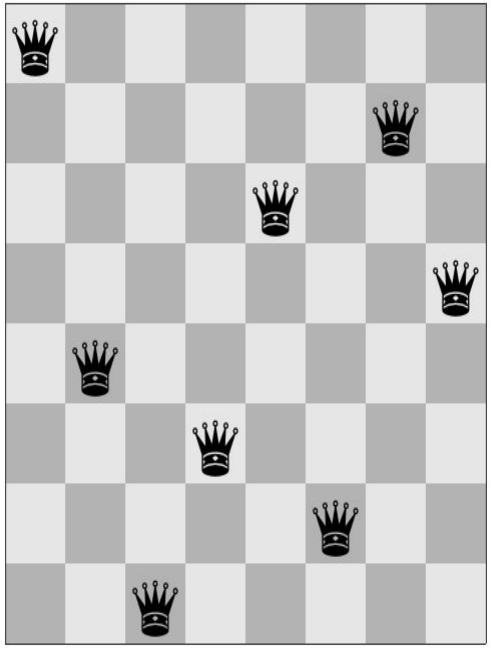


State-space tree of solving the four-queens problem by backtracking. × denotes an unsuccessful attempt to place a queen in the indicated column. The numbers above the nodes indicate the order in which the nodes are generated.

Few unique solutions for 8 queen problem







N-queens prblem

Algorithm to find all solutions of n-queens problem

```
Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all
// possible placements of n queens on an n × n
// chessboard so that they are nonattacking.
{
     for i := 1 to n do
     {
          if Place(k, i) then
          {
               x[k] := i;
               if (k = n) then write (x[1 : n]);
               else NQueens(k + 1, n);
          }
     }
}
```

N-queens problem

```
Algorithm Place(k, i)
// Returns true if a queen can be placed in kth row and
// ith column. Otherwise it returns false. x[ ] is a
// global array whose first (k – 1) values have been set.
// Abs(r) returns the absolute value of r.
{
    for j := 1 to k – 1 do
        if ((x[j] = i) // Two in the same column
            or (Abs(x[j] – i) = Abs(j – k)))
                // or in the same diagonal
            then return false;
    return true;
}
```

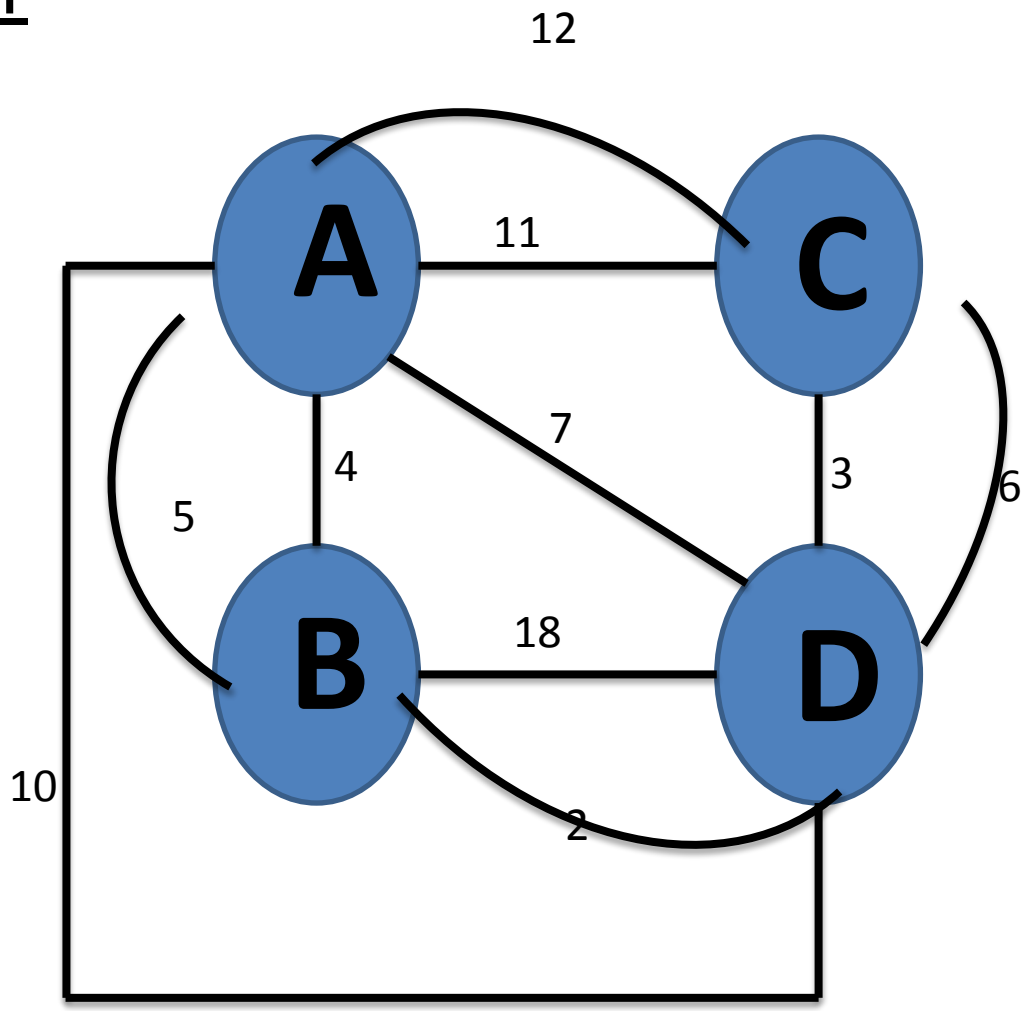
is_safe()

Travelling Salesman Problem

- Travelling Salesman Problem (TSP): Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

- Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exist a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

ASSIGNMENT




THANK
YOU