

A Project Report On

Artificial Intelligence and Robotics

SUBMITTED BY

Nand Patel

Roll No:41436

Kalyani Punpale

Roll No: 41441

CLASS: BE-4

GUIDED BY

Prof P. Rajmane



DEPARTMENT OF COMPUTER ENGINEERING

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

DHANKAWADI, PUNE-43

SAVITRIBAI PHULE PUNE UNIVERSITY

2020-21

Problem Statement:

N queens on an NxN chessboard such that no queen can strike down any other queen.

Objective

1. To learn and implement n-queens problem
2. To learn the concept of branch and bound and backtracking

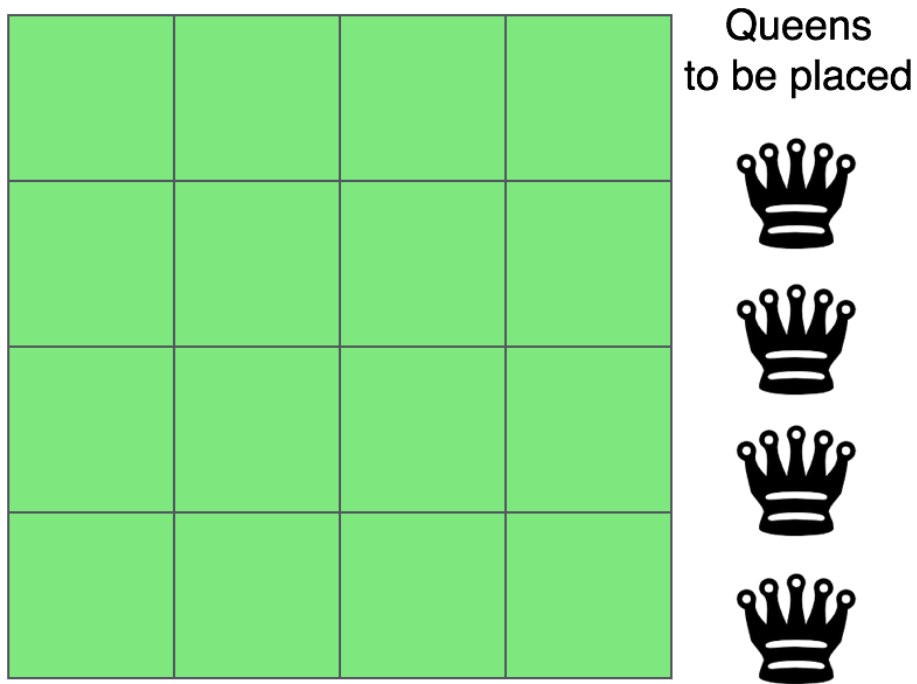
Theory

Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken. For example, in a maze problem, the solution depends on all the steps you take one-by-one. If any of those steps is wrong, then it will not lead us to the solution. In a maze problem, we first choose a path and continue moving along it. But once we understand that the particular path is incorrect, then we just come back and change it. This is what backtracking basically is.

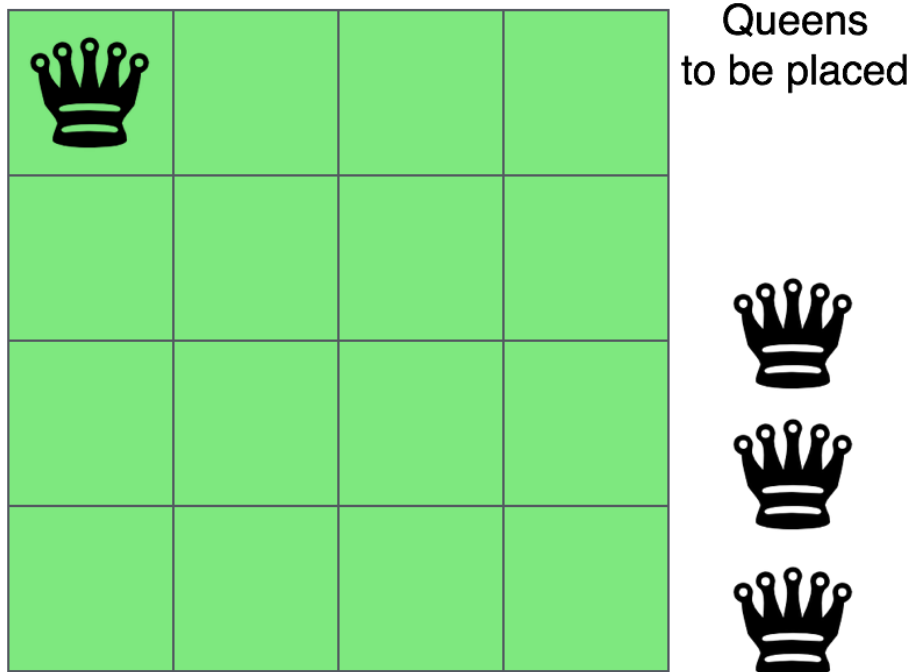
Thus, the general steps of backtracking are:

- start with a sub-solution
- check if this sub-solution will lead to the solution or not
- If not, then come back and change the sub-solution and continue again

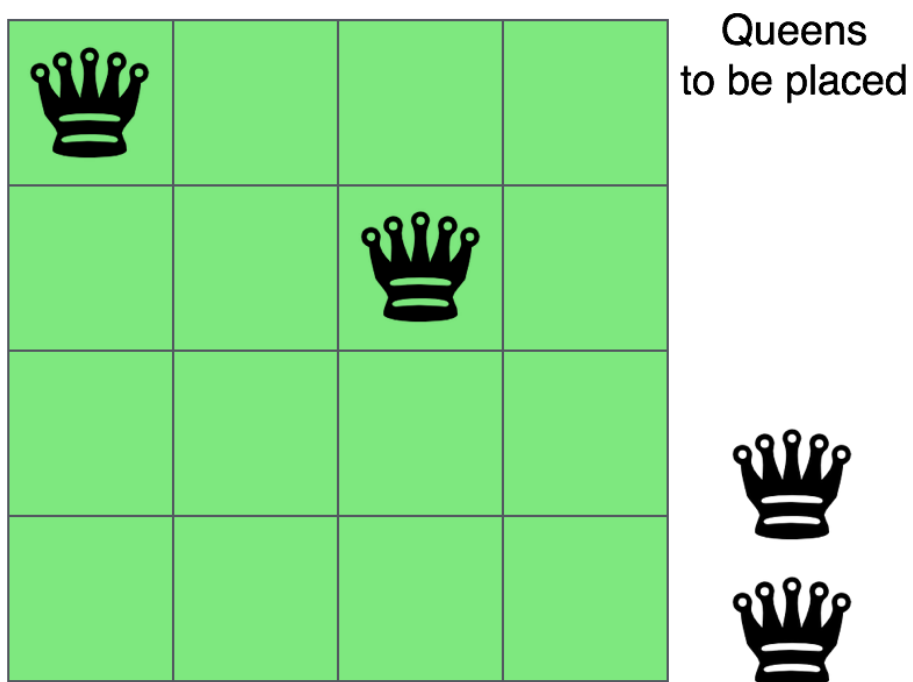
One of the most common examples of the backtracking is to arrange N queens on an $N \times N$ chessboard such that no queen can strike down any other queen. A queen can attack horizontally, vertically, or diagonally. The solution to this problem is also attempted in a similar way. We first place the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left. If no safe place is left, then we change the position of the previously placed queen.

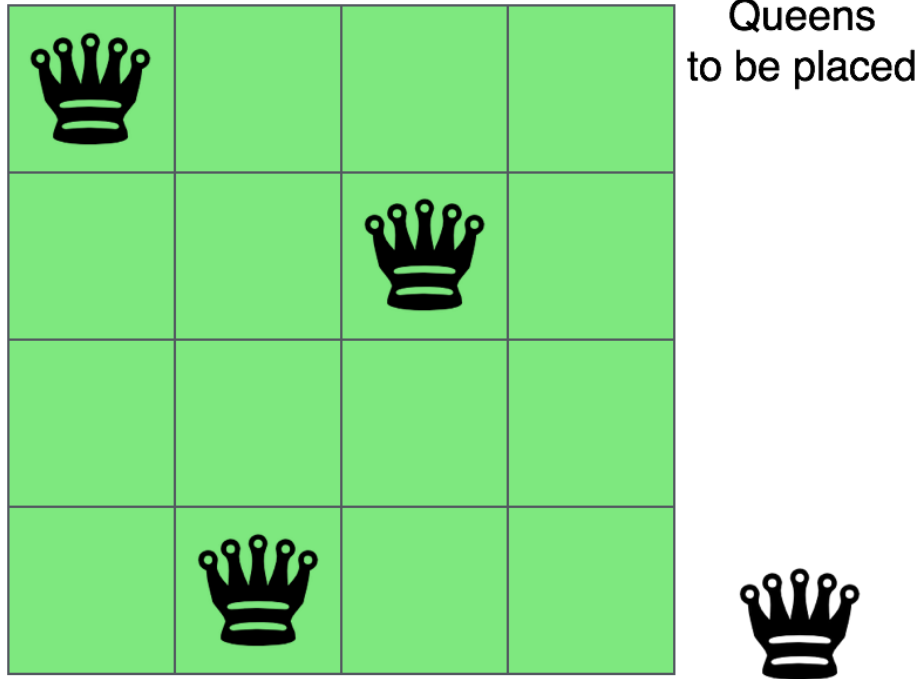


The above picture shows an $N \times N$ chessboard and we have to place N queens on it. So, we will start by placing the first queen.



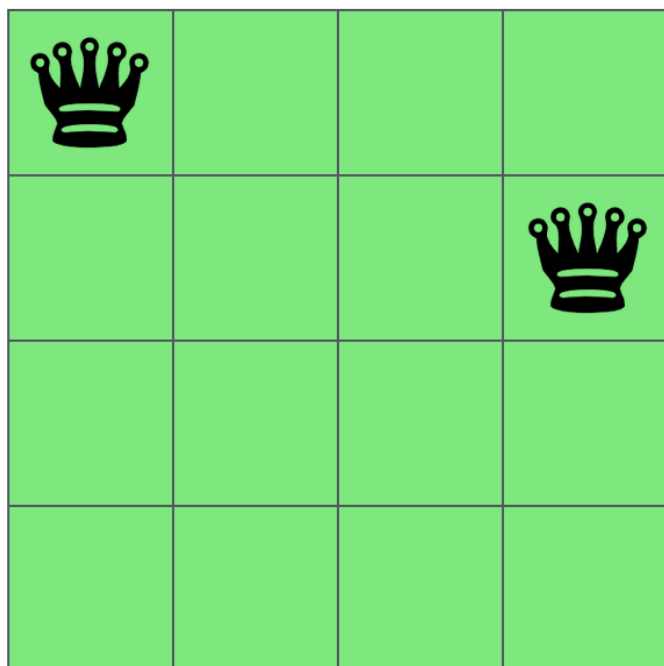
Now, the second step is to place the second queen in a safe position and then the third queen.





Now, you can see that there is no safe place where we can put the last queen. So, we will just change the position of the previous queen. And this is backtracking.

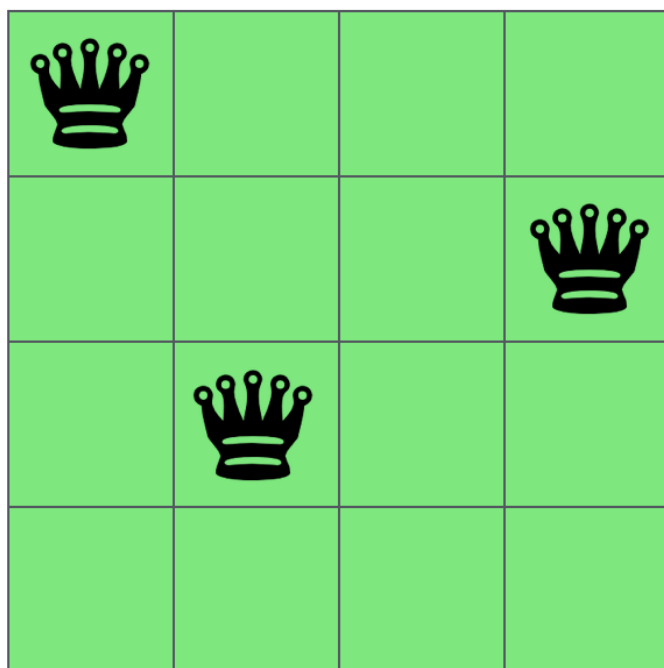
Also, there is no other position where we can place the third queen so we will go back one more step and change the position of the second queen.



Queens
to be placed



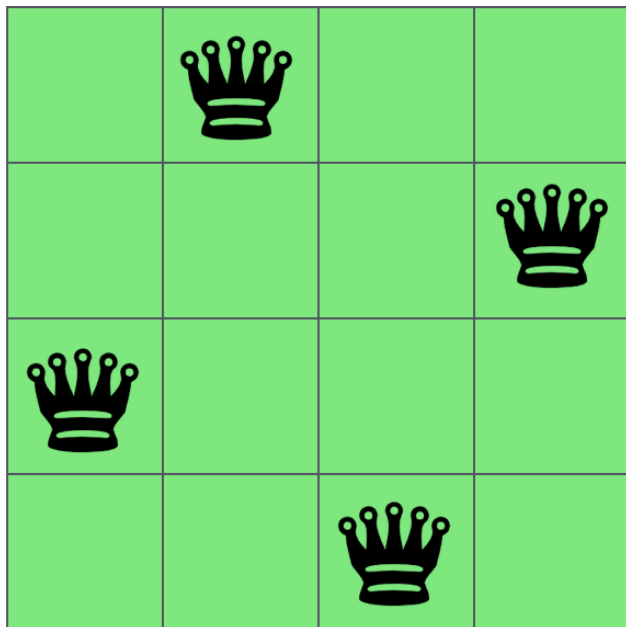
And now we will place the third queen again in a safe position until we find a solution.



Queens
to be placed



We will continue this process and finally, we will get the solution as shown below.



Source Code :

```
int is_attack(int i,int j)
{
    int k,l;
    //checking if there is a queen in row or column
    for(k=0;k<N;k++)
    {
        if((board[i][k] == 1) || (board[k][j] == 1))
            return 1;
    }
    //checking for diagonals
    for(k=0;k<N;k++)
```

```

{
    for(l=0;l<N;l++)
    {
        if(((k+l) == (i+j)) || ((k-l) == (i-j)))
        {
            if(board[k][l] == 1)
                return 1;
        }
    }
}
return 0;
}

```

```

int N_queen(int n)
{
    int i,j;
    //if n is 0, solution found
    if(n==0)
        return 1;
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            //checking if we can place a queen here or not
            //queen will not be placed if the place is being attacked
            //or already occupied
            if((!is_attack(i,j)) && (board[i][j]!=1))

```



```
{
    board[i][j] = 1;
    //recursion
    //wether we can put the next queen with this arrangment or not
    if(N_queen(n-1)==1)
    {
        return 1;
    }
    board[i][j] = 0;
}
}
}
return 0;
}
```

Screen Shot :



AIR_mini_project.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+ Code + Text



```
stat = N_queen(N)

if stat == True:
    for i in board:
        print (i)

else:
    print("No result possible")
```

Enter the number of queens

```
6
[0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0]
```

Conclusion :

Hence we learn what is backtracking and what is the N queen problem and how to solve it using backtracking.