Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 7-2-2021	Date of Submission: 21-2-2022

To implement N Queens Problem

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.
The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following are two solutions for 4 Queen problem.

Algorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

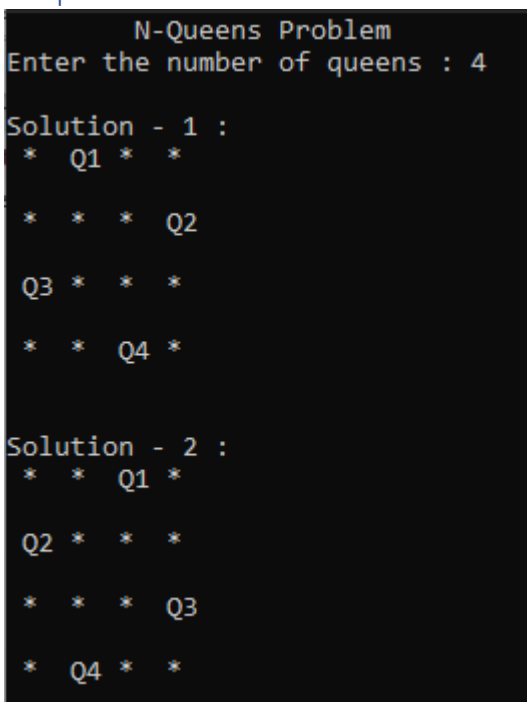
1) Start in the leftmost column
2) If all queens are placed return true
3) Try all rows in the current column. Do following for every tried row.
   a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
   b) If placing 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.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

Code

```
#include <stdio.h>
#include <stdlib.h>

int NoSoln(int k, int col[])
{
   int i;
   for(i=1;i<=k-1;i++)
   {
      if(col[k]==col[i] || (abs(i-k)==abs(col[i] - col[k])))
         return 1;
   }
   return 0;
}


int NQueen(int n)
{
   int k = 1;
   int count=0;
   int i,j,col[n+1];
   col[k]=0;
   while(k!=0)
```

```
    {
        col[k] += 1;
        while(col[k]<=n && NoSoln(k,col))
            col[k]=col[k]+1;
        if(col[k]<=n)
        {
            if(k==n)
            {
                count++;
                printf("\nSolution - %d : \n",count);
                for(i=1;i<=n;i++)
                {
                    for(j=1;j<=n;j++)
                        if(col[i] == j)
                            printf(" Q%d",i);
                        else
                            printf(" * ");
                    printf("\n\n");
                }
            }
        else
        {
            k++;
            col[k]=0;
        }
    }
    else
        k--;
    }
    return count;
}

int main()
{
    int n,solutions;
    printf("\tN-Queens Problem");
    printf("\nEnter the number of queens : ");
    scanf("%d",&n);
    solutions=NQueen(n);
    if(solutions==0)
        printf("No solution!!");
    return 0;
}
```

Output


N-Queens Problem
Enter the number of queens : 4
Solution - 1 :
* Q1 * *
* * * Q2
Q3 * * *
* * Q4 *
Solution - 2 :
* * Q1 *
Q2 * * *
* * * Q3
* Q4 * *

Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 21-2-2021	Date of Submission: 21-2-2022

To implement MiniMax Algorithm

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Algorithm

```
function minimax(node, depth, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
maxEva= -infinity
 for each child of node do
 eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva)       //gives Maximum of the values
return maxEva

else                          // for Minimizer player
```

```
minEva= +infinity
for each child of node do
eva= minimax(child, depth-1, true)
minEva= min(minEva, eva)        //gives minimum of the values
return minEva
```

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Code

```
import math
def minimax (curDepth, nodeIndex,maxTurn, scores,targetDepth):
    if (curDepth == targetDepth):
        return scores[nodeIndex]
    if (maxTurn):
        print (scores[nodeIndex])
        print ("->")
        return max(minimax(curDepth + 1, nodeIndex * 2,
                    False, scores, targetDepth),
                minimax(curDepth + 1, nodeIndex * 2 + 1,
                    False, scores, targetDepth))
    else:
        print (scores[nodeIndex])
        print (" ->")
        return min(minimax(curDepth + 1, nodeIndex * 2,
                    True, scores, targetDepth),
                minimax(curDepth + 1, nodeIndex * 2 + 1,
                    True, scores, targetDepth))

scores = [2, 3, 5, 9, 0, 1, 7, 5]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")
result=minimax(0, 0, True, scores, treeDepth)
print(result)
```

Output

```
PS E:\TY\SEM VI\AI> & C:/Users/varun/AppData/Local/Programs/Python/Python310/python.exe "e:/TY/SEM VI/AI/minimaxalgo.py"
The optimal value is : 3
 ->
3
 ->
3
 ->
5
 ->
5
 ->
2
 ->
9
 ->
12
->
3
->
3
 ->
5
->
9
->
3
```