72017912C
COMPBEB1326

GROUP A : ASSIGNMENT No.5

Page No.
Date

Title: N-Queens Problem

Objective: To solve N-Queens problem by using backtracking.

Problem Statement:
Design n-Queens matrix having first queen placed. Use backtracking to place remaining Queens to generate the final n-queen's matrix.

Software & Hardware Requirement:
1. Desktop / Laptop
2. Any operating system
3. Python
4. IDE or Code Editor

Theory:

Backtracking:
Backtracking is a technique based on algorithm to solve problem. It uses recursive calling to find the solution by building a solution step by step increasing values with time. It removes the solutions that doesn't give rise to the solution of the problem based on constraints given to solve the problem.

Backtracking algorithm is applied to some specific types of problems,
1) Decision problems used to find a feasible

solution of the problem.

2) Optimization problem used to find the best solution that can be applied.

3) Enumeration problem used to find the set of all feasible solutions of the problem.

In backtracking problem, the algorithm tries to find a sequence path to the solution which has some small checkpoints from where the problem can backtrack if no feasible solution is found for the problem.

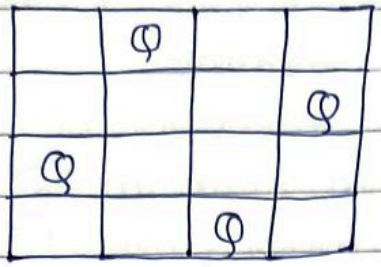
Backtracking Algorithm:

Step-1 : if current_position is goal, return success.
Step-2 : else,
Step-3 : if current_position is an end point, return failed.
Step-4 : else, if current_position is not end point, explore & repeat above steps.

N-Queen Problem:

In N-Queen problem, we are given an N×N chessboard & we have to place n queens on the board in such a way that no two queens attack each other. A queen will attack another queen if it is placed in horizontal, vertical or diagonal points in its way.

e.g for 4-Queen problem, the solution will be



Here, the binary output for n queen problem with 1's as queens to the positions are placed.

{0, 1, 0, 0}
{0, 0, 0, 1}
{1, 0, 0, 0}
{0, 0, 1, 0}

For solving n queens problem, we will try placing queen into different positions of one row. And checks if it clashes with other queens. If current positioning of queens if there are any two queens attacking each other then we will backtrack to previous location of the queen & change its positions. And check for clash of queen again.

N-Queens Algorithm:

Step 1: Start from 1st position in the array.
Step 2: Place queens in the board & check. Do,
Step 2.1: After placing the queen, mark the position as a part of the solution & then recursively check if this will lead to a solution.
Step 2.2: Now, if placing the queen doesn't lead to a solution & trackback & go to step (a) &

Page No.
Date

place the queens to other rows.

Step 2.3: If placing queen returns a lead to solution return TRUE.

Step 3: If all queens are placed return TRUE.

Step 4: If all rows are tried & no solution is Found, return False.

Testcases:

No.	Input	Expected Output	Actual Output	Result
1	1	{1}	{1}	Pass
2	2	No Sol.	No Sol.	Pass
3	4	{0,1,0,0} {0,0,0,1} {1,0,0,0} {0,0,1,0}	{0,1,0,0} {0,0,0,1} {1,0,0,0} {0,0,1,0}	Pass

Conclusion: Successfully implemented solution For N-queens problem by using backtracking approach.

Rambha
28/9/22

Code:

```
global N
N = int(input('Enter N: '))

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

def isSafe(board, row, col):

    for i in range(col):
        if board[row][i] == 1:
            return False

    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQUtil(board, col):

    if col >= N:
        return True

    for i in range(N):

        if isSafe(board, i, col):

            board[i][col] = 1

            if solveNQUtil(board, col + 1) == True:
                return True

            board[i][col] = 0

    return False

def solveNQ():
    board = [[0 for i in range(N)] for j in range(N)]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False
```

```
    printSolution(board)
    return True

solveNQ()
```

Output:

```
Enter N: 4
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```