| | |
|---|---|
|  | Bansilal Ramnath Agarwal Charitable Trust's<br>Vishwakarma Institute of Information Technology<br>**Department of**<br>**Artificial Intelligence and Data Science** |

| | | |
|---|---|---|
| Name: Siddhesh Dilip Khairnar | | |
| Class: TY | Division: B | Roll No: 372028 |
| Semester: V | Academic Year: 2023-24 | |
| Subject Name & Code: Design and Analysis of Algorithm: ADUA31202 | | |
| Title of Assignment: Implement N- queen Problem using Back tracking method. A suitable message is to be displayed if the given Problem instance doesn't have a solution. | | |

# ASSIGNMENT NO. 6

# DAA Assignment no - 6

Name: Siddhesh Dilip Khairnar
ROLL NO: 372028
PRN NO: 22110398

Aim: Implement N-queen problem using Back tracking method. A suitable message is to be displayed if the given problem instances doesn't have a solution.

Learning ~~Objective~~ Objective: →

1) Gain a deep understanding of combinatorial problem like the N-Q problem, where the goal is to find valid combination or arrangement of element based on specific constrait.
2) Understand how recursive function are used in backtracking algorithm to explore all possible solution.
3) Explore ways to optimize backtracking algorithm for efficiency, especially for problem with a large search space.

Theory: →

The N-Queens problem can be efficiently solved using backtracking algorithm. Here are two common algorithm for solving the N-Queens problem using backtracking: →

1) Recursive Backtracking Algorithm: →

This algorithm uses a recursive approach to explore all possible configuration of queens on the chessboard. If it encounter an unsafe position, it backtracks and explores other possibilites. Here are the key step: →

- start with an empty chessboard
- place queens one by one in different columns, starting from the leftmost column.
- for each column, try to places the queen in every row of that column.
- check if the placement is safe.
- If a safe placement is found, move on to the next column recursively.
- If no safe placement is found in a column, backtrack to the previous column and continue from there.
- Repeat these steps until all queens are placed successfully.

2) Iterative Backtracking Algorithm: →

This algorithm uses an iterative approach to explore all possible configuration without explict explicit recursion. It typically employs a stack or a data structure to manage the backtracking process. Here are the key steps: →

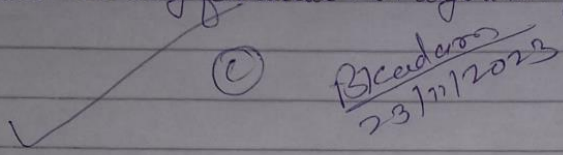
- Initialize a stack or data structure to keep track of the current state (row & column) & the partial solution.
- Start with an empty chessboard & push the first column onto the stack.
- While the stack is not empty, repeat the following: →
  - → Pop the pop top column from the stack.
  - → Try to place the queen in each row of the column.
    - → If a safe placement is found, updates the partial sol"& push the next column onto the stack.
    - → If no safe placement is found in the column, backtrack by popping columns from the stack until you can continue exploring.
- Continue this process until all sol" are found or all possibilities are exhausted.

Page No.
Date

Both algorithm use the idea of backtracking to explore possibilities solution and backtrack when an unsafe position is encountered. The recursive approaches is often simpler to implement, while the iterative approach can be more memory-efficient for large N values.

Conclusion :→

Learning about backtracking algorithm through the N-queens problem provide a strong foundation in algorithm problem solving.



Bcaderoo
23/11/2023

## Program Code:

```
#include <bits/stdc++.h>
#define N 4
using namespace std;

void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << " " << board[i][j] << " ";
        printf("\n");
    }
}

bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

bool solveNQUtil(int board[N][N], int col)
{

    if (col >= N)
        return true;

    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col))
        {
            board[i][col] = 1;

            if (solveNQUtil(board, col + 1))
```

```
                return true;

            board[i][col] = 0;
        }
    }

    return false;
}

bool solveNQ()
{
    int board[N][N] = {{0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0}};

    if (solveNQUtil(board, 0) == false)
    {
        cout << "Solution does not exist";
        return false;
    }

    printSolution(board);
    return true;
}

int main()
{
    solveNQ();
    return 0;
}
```

## Output:

```
PS D:\Program language\C++> cd "d:\Program language\C++\" ; if ($?) { g++ new.cpp -o new } ; if ($?) {
.\new }
 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0
PS D:\Program language\C++>
```