Data Structures Lab 05(b)

Course: Data Structures (CL2001) **Semester:** Fall 2024
Instructor: Sameer Faisal **T.A:** N/A

Note:

- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

BACKTRACKING

Backtracking is a search technique for solving complex problems by recursively exploring combinations of possible choices to arrive at a solution. It is commonly used to solve search, optimization, planning and gaming problems.

Sample Pseudocode:

```
void findSolutions(n, other params) :
    if (found a solution) :
        solutionsFound = solutionsFound + 1;
        displaySolution();
        if (solutionsFound >= solutionTarget) :
            System.exit(0);
        return

    for (val = first to last) :
        if (isValid(val, n)) :
            applyValue(val, n);
            findSolutions(n+1, other params);
            removeValue(val, n);
```

Example 1: Rat in a Maze

Consider a rat placed at (0, 0) in a square matrix of order N * N. It has to reach the destination at (N – 1, N – 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are ‘U'(up), ‘D'(down), ‘L’ (left), ‘R’ (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.

```c
bool isSafe(int **arr, int x, int y, int n) {
    if (x < n && y < n && arr[x][y] == 1) {
        return true;
    }
    return false;
}

bool ratinMaze(int** arr, int x, int y, int n, int** solArr) {
    if ((x == (n - 1)) && (y == (n - 1))) {
        solArr[x][y] = 1;
        return true;
    }
    if (isSafe(arr, x, y, n)) {
        solArr[x][y] = 1;
        if (ratinMaze(arr, x + 1, y, n, solArr)) {
            return true;
        }
        if (ratinMaze(arr, x, y + 1, n, solArr)) {
            return true;
        }
        solArr[x][y] = 0; //backtracking
        return false;
    }
    return false;
}
```

Example 2: N-Queen Problem

The N-Queens problem is a classic problem in computer science and combinatorial optimization. The goal is to place N queens on an N×N chessboard in such a way that no two queens threaten each other. In other words, no two queens can share the same row, column, or diagonal.

```
bool isSafe(int board[], int row, int col) {
    for (int i = 0; i < row; i++) {
        // Check if there's a queen in the same column or diagonals
        if (board[i] == col || abs(board[i] - col) == abs(i - row)) {
            return false;
        }
    }
    return true;
}
bool solveNQueens(int board[], int n, int row = 0) {
    if (row == n) {
        // All queens are placed successfully
        return true;
    }

    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col)) {
            board[row] = col; // Place the queen in this column

            // Recursively place queens in the next row
            if (solveNQueens(board, n, row + 1)) {
                return true; // If a solution is found, return true
            }

            // If placing the queen in this column doesn't lead to a solution, backtrack
            board[row] = -1;
        }
    }

    // If no safe position is found, return false (backtrack)
    return false;
}
```