BRACT’s

Vishwakarma Institute of Information Technology, Pune

**Practical Implementation Sheet**

| **Department:** IT | **Semester:** IV | **Academic Year:** 2024-25 | **Practical No: 7** |
| --- | --- | --- | --- |
| **Class/ Division/ Batch: SY (B)- B3** | | **Roll no: 70** | |
| **Course:** Data Structures and Analysis of Algorithms | | **Name of Student**: Anushka Kadam | |

**Aim:** Backtracking:

a) Solve Graph Coloring problem using backtracking approaches.

b) N-Queens Problem: Write a recursive program to find the solution of placing N-

queens on a chess board so that no queen takes each other.

**Code: a) Graph Coloring problem using backtracking approaches.**

#include <iostream>

#include <vector>

using namespace std;

bool isSafe(int node, vector<vector<int>>& graph, vector<int>& color, int c, int n)

{

for (int i = 0; i < n; i++)

{

if (graph[node][i] == 1 && color[i] == c)

return false;

}

return true;

}

bool solve(int node, vector<vector<int>>& graph, vector<int>& color, int m, int n)

{

if (node == n)

return true;

for (int c = 1; c <= m; c++)

{

if (isSafe(node, graph, color, c, n))

{

color[node] = c;

if (solve(node + 1, graph, color, m, n))

return true;

color[node] = 0;

}

}

return false;

}

int main()

{

int n, m;

cout << "Enter number of vertices: ";

cin >> n;

cout << "Enter number of colors: ";

cin >> m;

vector<vector<int>> graph(n, vector<int>(n));

cout << "Enter adjacency matrix:\n";

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

cin >> graph[i][j];

vector<int> color(n, 0);

if (solve(0, graph, color, m, n))

{

cout << "Color assignment:\n";

for (int i = 0; i < n; i++)

cout << "Vertex " << i << " ---> Color " << color[i] << endl;

}

else

{

cout << "No solution exists with " << m << " colors.\n";

}

return 0;

}

**Output:**

****

**Code: b) N-Queens Problem**

#include <iostream>

#include <vector>

using namespace std;

bool isSafe(vector<vector<int>>& board, int row, int col, int rows, int cols)

{

for (int i = 0; i < row; i++)

if (board[i][col] == 1)

return false;

for (int j = 0; j < col; j++)

if (board[row][j] == 1)

return false;

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

if (board[i][j] == 1)

return false;

for (int i = row - 1, j = col + 1; i >= 0 && j < cols; i--, j++)

if (board[i][j] == 1)

return false;

for (int i = row + 1, j = col - 1; i < rows && j >= 0; i++, j--)

if (board[i][j] == 1)

return false;

for (int i = row + 1, j = col + 1; i < rows && j < cols; i++, j++)

if (board[i][j] == 1)

return false;

for (int i = row + 1; i < rows; i++)

if (board[i][col] == 1)

return false;

for (int j = col + 1; j < cols; j++)

if (board[row][j] == 1)

return false;

return true;

}

bool placeQueens(vector<vector<int>>& board, int rows, int cols, int queensToPlace, int startRow = 0)

{

if (queensToPlace == 0)

return true;

for (int i = startRow; i < rows; i++)

{

for (int j = 0; j < cols; j++)

{

if (isSafe(board, i, j, rows, cols))

{

board[i][j] = 1;

if (placeQueens(board, rows, cols, queensToPlace - 1, i))

return true;

board[i][j] = 0;

}

}

}

return false;

}

void printBoard(const vector<vector<int>>& board, int rows, int cols) {

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

cout << (board[i][j] == 1 ? "Q " : ". ");

cout << endl;

}

}

int main()

{

int rows, cols, queens;

cout << "Enter number of rows: ";

cin >> rows;

cout << "Enter number of columns: ";

cin >> cols;

cout << "Enter number of queens to place: ";

cin >> queens;

if (queens > rows \* cols)

{

cout << "Too many queens for the board size!" << endl;

return 0;

}

vector<vector<int>> board(rows, vector<int>(cols, 0));

if (placeQueens(board, rows, cols, queens))

{

cout << "\nSolution:\n";

printBoard(board, rows, cols);

}

else

{

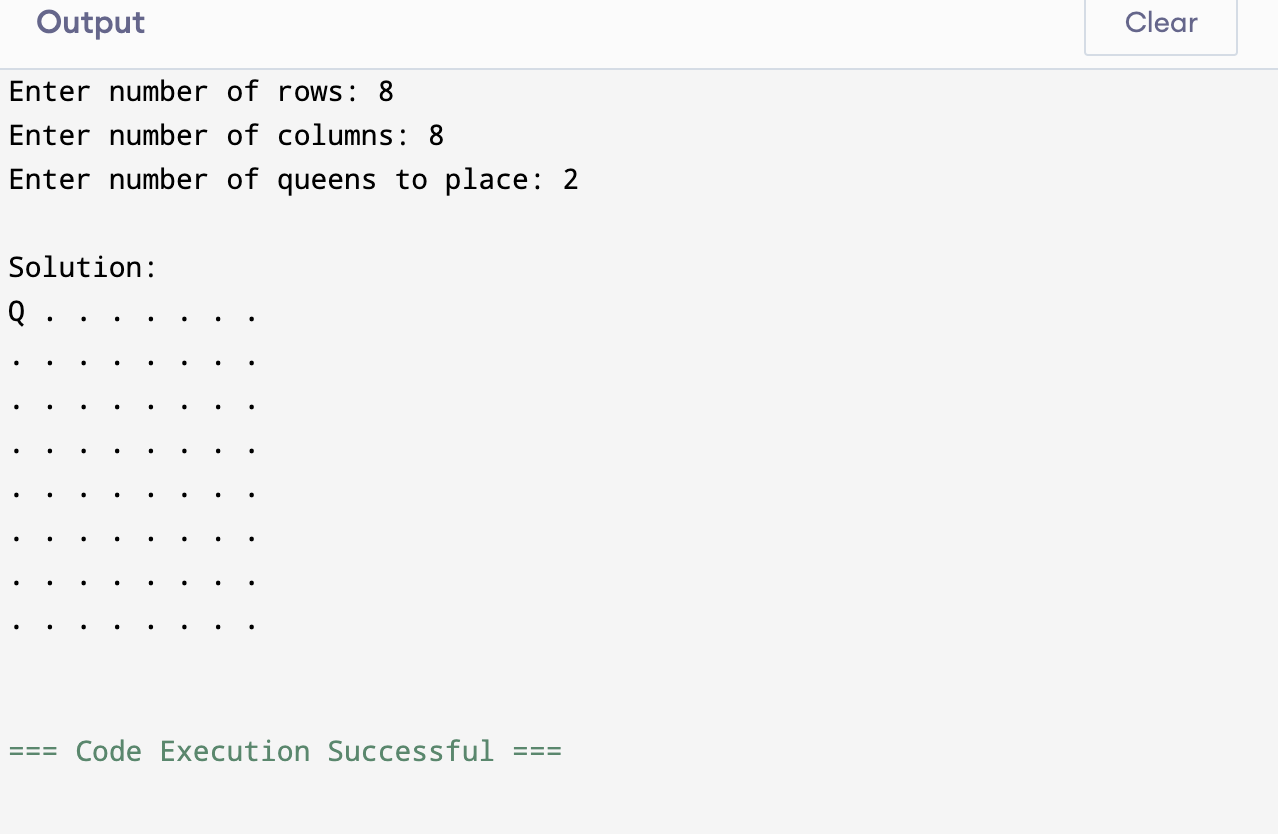
cout << "No solution possible for given input.\n";

}

return 0;

}

**OUTPUT:**

****