**AI Assignment 4**

Sharvari Deshmukh

TY CSA 73

**Problem Statement**- Implementation of CSP Problem.

**What is CSP?**

* Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue.
* Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems.
* For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

**Graph Coloring Problem-**

**Graph coloring** refers to the problem of **coloring vertices** of a graph in such a way that **no two adjacent**vertices have the **same** **color**. This is also called the **vertex coloring** problem. If coloring is done using at most m colors, it is called m-coloring.

Graph coloring problem is both, a decision problem as well as an optimization problem.

• A decision problem is stated as, “With given M colors and graph G, whether a such color scheme is possible or not?”.

• The optimization problem is stated as, “Given M colors and graph G, find the minimum number of colors required for graph coloring.”

**Algorithm-**

**Approach**:

Basically starting from vertex 0 color one by one the different vertices.

**Base condition:**

If I have colored all the N nodes return true.

**Recursion:**

* Trying every color from 1 to m with the help of a for a loop.
* Is safe function returns true if it is possible to color it with that color i.e none of the adjacent nodes have the same color.
* Color it with color i then call the recursive function for the next node if it returns true we will return true.
* And If it is false then take off the color.
* Now if we have tried out every color from 1 to m and it was not possible to color it with any of the m colors then return false.

**Time Complexity: O( N^M)**

**Space Complexity: O(N)**

#include <iostream>

#include <vector>

#include <cstring>

using namespace std;

bool isSafe(int node, vector<int>& color, vector<vector<int>>& graph, int col) {

    for (int neighbor : graph[node]) {

        if (color[neighbor] == col) {

            return false;

        }

    }

    return true;

}

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

    if (node == N) { //base case-if every node is colored I reach the N

        return true;

    }

    for (int col = 1; col <= m; col++) {

        if (isSafe(node, color, graph, col)) {

            color[node] = col;

            if (solve(node + 1, color, m, N, graph)) return true; //no need to check other possible solutions

            color[node] = 0; // remove the previously given color, backtrack

        }

    }

    return false; //no solution

}

bool graphColoring(vector<vector<int>>& graph, int m, int N) {

    vector<int> color(N, 0);

    if (solve(0, color, m, N, graph)) {

        // Print colors assigned to each vertex

        cout << "Vertex Colors:\n";

        for (int i = 0; i < N; ++i) {

            cout << "Vertex " << i + 1 << ": Color " << color[i] << endl;

        }

        return true;

    }

    return false;

}

int main() {

    int N, m, edges;

    cout << "Enter the number of nodes: ";

    cin >> N;

    cout << "Enter the number of colors: ";

    cin >> m;

    cout << "Enter the number of edges: ";

    cin >> edges;

    vector<vector<int>> graph;

    graph.resize(N);

    cout << "Enter the edges:\n";

    for (int i = 0; i < edges; i++) {

        int node1, node2;

        cout << "\nEnter edge vertices of edge " << i + 1 << " :";

        cin >> node1 >> node2;

        node1--;

        node2--;

        graph[node1].push\_back(node2);

        graph[node2].push\_back(node1);

    }

    cout << (graphColoring(graph, m, N) ? "Yes" : "No") << endl;

    return 0;

}

