#include <iostream>

#include <vector>

using namespace std;

// Function to check if it is safe to color the vertex v with color c

bool isSafe(int v, const vector<vector<int>>& graph, const vector<int>& color, int c) {

for (int i = 0; i < graph.size(); i++) {

if (graph[v][i] && c == color[i])

return false;

}

return true;

}

// Function to recursively solve the graph coloring problem using backtracking

bool graphColoringUtil(int v, const vector<vector<int>>& graph, vector<int>& color, int m) {

if (v == graph.size())

return true;

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

if (isSafe(v, graph, color, c)) {

color[v] = c;

if (graphColoringUtil(v + 1, graph, color, m))

return true;

color[v] = 0; // backtrack

}

}

return false;

}

// Function to solve the graph coloring problem using branch and bound with backtracking

void graphColoring(const vector<vector<int>>& graph, int m) {

vector<int> color(graph.size(), 0);

if (graphColoringUtil(0, graph, color, m)) {

cout << "Graph can be colored using at most " << m << " colors.\n";

cout << "Coloring scheme:\n";

for (int i = 0; i < graph.size(); i++) {

cout << "Vertex " << i << ": Color " << color[i] << "\n";

}

} else {

cout << "Graph cannot be colored using " << m << " colors.\n";

}

}

int main() {

int n, m; // n: number of vertices, m: number of colors

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

cin >> n;

// Initialize the adjacency matrix of the graph

vector<vector<int>> graph(n, vector<int>(n, 0));

cout << "Enter the adjacency matrix (0 or 1):\n";

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

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

cout<<"Enter the edges between"<<i<<" "<<j;

cin >> graph[i][j];

}

}

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

cin >> m;

graphColoring(graph, m);

return 0;

}

// v=4

//clor=3

//1->2,1->3,1->4,2->4,2->1,3->1,3->4,4->1,4->2,4->3

// abhisek

|  |
| --- |
| #include <iostream> |
|  | #include<list> |
|  | #include <algorithm> |
|  | #include <vector> |
|  | using namespace std; |
|  |  |
|  | #define MAX\_COLORS 1001 |
|  | vector<int> Color(MAX\_COLORS,-1); |
|  |  |
|  | class Graph{ |
|  | int V; |
|  | int maxColors; |
|  | list<int> \*adj; |
|  | public: |
|  | Graph(int V,int m); |
|  | void addEdge(int u,int v); |
|  | bool coloring(int u); |
|  | bool isSafe(int u,int color); |
|  | void printResult(); |
|  | }; |
|  | Graph::Graph(int V,int m){ |
|  | this->maxColors = m; |
|  | this->V = V; |
|  | adj = new list<int>[this->V]; |
|  | } |
|  | void Graph::printResult(){ |
|  | for(int i=0;i<this->V;i++){ |
|  | cout << "Vertex " << i << " has color " << Color[i] << endl; |
|  | } |
|  | cout << endl << endl; |
|  | } |
|  | void Graph::addEdge(int u,int v){ |
|  | adj[u].push\_back(v); |
|  | adj[v].push\_back(u); |
|  | } |
|  | bool Graph::isSafe(int u,int color){ |
|  | list<int>::iterator it; |
|  | //if any adjacent vertex to u has the same color return false. |
|  | //else is safe |
|  | for(it=adj[u].begin();it!=adj[u].end();++it){ |
|  | if(Color[\*it] == color) return false; |
|  | } |
|  | return true; |
|  | } |
|  | //Backtrack function |
|  | //u represents the number of vertices colored. The verices are colored in the order they are labeled. 0,1...V-1 |
|  | bool Graph::coloring(int u){ |
|  | if(u==this->V) return true; |
|  | for(int i=0;i<this->maxColors;i++){ |
|  | //i represents the ith color. |
|  | if(isSafe(u,i)){ |
|  | Color[u] = i; //color of u equals i |
|  | if(coloring(u+1)) return true; |
|  |  |
|  | Color[u] = -1; //Backtrack |
|  | } |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | int main(){ |
|  | int V = 5; |
|  | int maxColors = 3; |
|  | Graph g(V,maxColors); |
|  | g.addEdge(0,1); |
|  | g.addEdge(0,2); |
|  | g.addEdge(0,3); |
|  | g.addEdge(1,2); |
|  | g.addEdge(3,4); |
|  |  |
|  | int vertexColored = 0; |
|  | if(g.coloring(vertexColored)){ |
|  | cout << "Success! " << endl << endl; |
|  | g.printResult(); |
|  | } |
|  | else{ |
|  | cout << "Failed coloring the graph" << endl; |
|  | } |
|  | return 0; |
|  | } |