NAME:M.BELCIA

CLASS:II-CSE-A

ROLL NO:711524BCS020

# DATA STRUCTURE MINI PROJECT

## AIM

To implement a C++ program that assigns colors to the vertices of a graph such that no two adjacent vertices have the same color, using a backtracking algorithm.

## ALGORITHM

### Step 1:

Start the program.

### Step 2:

Represent the graph using an adjacency matrix, where graph[i][j] = 1 means there is an edge between vertex i and vertex j,and 0 means no edge.

### Step 3:

Decide how many colors (m) you want to use for coloring the graph.

### Step 4:

Start coloring from the first vertex (vertex 0).

### Step 5:

For each vertex:

- Try assigning each color (from 1 to m).
- Before assigning, check if it's safe (i.e., none of its adjacent vertices have the same color).

## Step 6:

If it's safe, assign that color to the vertex and move to the next vertex.

## Step 7:

If the next vertex cannot be colored (no color fits), then backtrack —remove the color from the current vertex and try a different color.

## Step 8:

If all vertices are successfully colored, print the color of each vertex.

## Step 9:

If no color combination works, print "No solution exists."

## Step 10:

Stop the program.

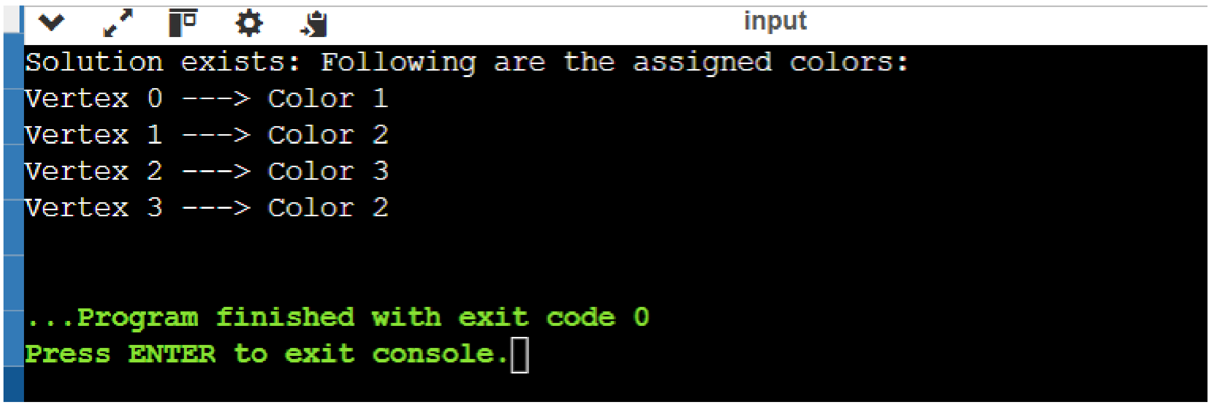
## CODE

```
#include <iostream>

#include <vector>

using namespace std;
```

```
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] && color[i] == c)
            return false;
    }
    return true;
}

bool graphColoringUtil(const vector<vector<int>>& graph, int m,
vector<int>& color, int v) {
    int n = graph.size();
    if (v == n)
        return true;
    for (int c = 1; c <= m; c++) {
        if (isSafe(v, graph, color, c)) {
            color[v] = c;
            if (graphColoringUtil(graph, m, color, v + 1))
                return true;
            color[v] = 0;
        }
    }
    return false;
}
bool graphColoring(const vector<vector<int>>& graph, int m) {
```

```cpp
    int n = graph.size();
    vector<int> color(n, 0);
    if (!graphColoringUtil(graph, m, color, 0)) {
        cout << "No solution exists.\n";
        return false;
    }
    cout << "Solution exists: Following are the assigned 
colors:\n";
    for (int i = 0; i < n; i++)
        cout << "Vertex " << i << " ---> Color " << color[i] << endl;

    return true;
}

int main() {
    vector<vector<int>> graph = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0}
    };

    int m = 3;
    graphColoring(graph, m);


    return 0;
}
```

## OUTPUT:



```
Solution exists: Following are the assigned colors:
Vertex 0 ---> Color 1
Vertex 1 ---> Color 2
Vertex 2 ---> Color 3
Vertex 3 ---> Color 2


...Program finished with exit code 0
Press ENTER to exit console.
```

## CONCLUSION

The program successfully assigns colors to all vertices of a graph so that no two adjacent vertices have the same color. It demonstrates how **backtracking** can be used to solve combinatorial optimization problems efficiently.