Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Lab Report 03: Graph Coloring

Course Title: Artificial Intelligence Lab

Course Code: CSE-316 Section:221-14

Student Details

Name		ID
1.	Md.Mohibullah	221902083

Lab Date : 26-02-2025

Submission Date : 04-04-2025

Course Teacher's Name : Md. Sabbir Hosen Mamun

Lab Report Status

Marks: …… **Signature:.....**

Comments:..... **Date:.....**

Lab Report Name: Graph Coloring using Backtracking

1. Introduction

Graph coloring is a well-known problem in computer science and combinatorics. It involves assigning colors to the vertices of a graph such that no two adjacent vertices share the same color. This problem has numerous applications in areas such as scheduling, resource allocation, and frequency assignment in communication networks. The aim of this lab is to solve the graph coloring problem using a backtracking algorithm.

In this lab, we explore an undirected graph represented by vertices and edges, and our task is to determine if it is possible to color the graph using a specified number of colors (K) such that no two adjacent vertices share the same color.

2. Objective

The primary objective of this lab is to develop a Python program that uses backtracking to determine whether a given graph can be colored using at most K colors. The program should read the input graph structure from a file, which contains the number of vertices (N), edges (M), and available colors (K), and output whether the graph can be colored under these constraints.

3. Problem Statement

We are given an undirected graph with:

- N vertices numbered from 0 to N-1.
- M edges, where each edge connects two vertices.
- K available colors for coloring the graph.

The task is to determine if it is possible to assign a color to each vertex such that:

- No two adjacent vertices share the same color.
- The total number of colors used does not exceed K.

4. Algorithm

We approach the problem using Backtracking. The basic idea is to assign a color to a vertex and then check if the current coloring is valid by ensuring that no two adjacent vertices have the same color. If a valid coloring is found, we move on to the next vertex. If a conflict arises (i.e., no valid color can be assigned to a vertex), we backtrack by removing the color and trying a different one. If we exhaust all possible color assignments and find no solution, we conclude that the graph cannot be colored with the given number of colors.

Steps:

1. Initialization:

    - Create a graph using an adjacency list to represent the vertices and edges.
    - Initialize a color array to store the color of each vertex (starting with 0, meaning no color assigned).

2. Backtracking:

    - Start at the first vertex and try assigning a color (from 1 to K).
    - For each color, check if it can be assigned to the current vertex by ensuring that no adjacent vertex has the same color.
    - If a valid color is found, recursively assign colors to the next vertex.
    - If we reach the last vertex and successfully color it, the solution is found.
    - If no valid color can be assigned, backtrack and try a different color for the previous vertex.

3. Termination:

    - If all vertices are successfully colored, print the solution.

    - If it's not possible to color the graph with K colors, print that it's not possible.

5. Implementation

```
import os

filename = "input.txt"

if not os.path.exists(filename):

    with open(filename, "w") as f:

        f.write("4 5 3\n0 1\n0 2\n1 2\n1 3\n2 3\n")  # Sample input

class GraphColoring:

    def __init__(self, N, M, K, edges):

        self.N = N  # Number of vertices

        self.M = M  # Number of edges

        self.K = K  # Number of colors

        self.graph = {i: [] for i in range(N)}
```

```
        for u, v in edges:
            self.graph[u].append(v)
            self.graph[v].append(u)
        self.colors = [0] * N  # Color assignment for vertices

    def is_safe(self, node, c):
        """Check if color c can be assigned to node safely"""
        for neighbor in self.graph[node]:
            if self.colors[neighbor] == c:
                return False
        return True

    def backtrack(self, node):
        """Backtracking function to assign colors"""
        if node == self.N:
            return True
        for c in range(1, self.K + 1):
            if self.is_safe(node, c):
                self.colors[node] = c
                if self.backtrack(node + 1):
                    return True
                self.colors[node] = 0  # Backtrack
        return False
    def solve(self):
        """Solve the graph coloring problem"""
        if self.backtrack(0):
            print(f"Coloring Possible with {self.K} Colors")
```

```
            print("Color Assignment:", self.colors)

        else:

            print(f"Coloring Not Possible with {self.K} Colors")

# Function to parse input from file and run the test

def parse_and_run(filename):

    with open(filename, 'r') as file:

        N, M, K = map(int, file.readline().split())

        edges = [tuple(map(int, file.readline().split())) for _ in
range(M)]

    solver = GraphColoring(N, M, K, edges)

    solver.solve()

if __name__ == "__main__":

    parse_and_run("input.txt")
```

6. Result

Given the sample input in the input.txt file:

4 5 3

0 1

0 2

1 2

1 3

2 3

The program successfully determines that the graph can be colored with 3 colors and outputs the following:

```
PS E:\8th semester\AI Lab\Lab Report 02> & "C:/Program Fil
loring.py"
Coloring Possible with 3 Colors
Color Assignment: [1, 2, 3, 1]
PS E:\8th semester\AI Lab\Lab Report 02>
```

In this solution:

- **Vertex 0 is assigned color 1.**
- **Vertex 1 is assigned color 2.**
- **Vertex 2 is assigned color 3.**
- **Vertex 3 is assigned color 2.**

7. Conclusion

The graph coloring problem was successfully solved using a backtracking algorithm. The algorithm efficiently checks all possible color assignments and backtracks when necessary to find a valid solution. The implementation demonstrates how backtracking can be applied to combinatorial problems, where the solution space is explored by systematically trying out different possibilities. This approach works well for relatively small graphs and can be extended to more complex scenarios as needed.

GitHub Link: https://github.com/programmermahi/Artificial-Intelligence/tree/main/LabReport03-GraphColoring