Green University of Bangladesh
Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: Spring Year:2025

Course Title: Artificial Intelligence Lab
Course Code: CSE(316) Section: 221-D22

Student Details

Name	ID
Md Moinul Hasan	221902281

Date: 01/03/2025

Submission Date: 06/04/2025

Course Teacher's Name: Md.Sabbir Hosen Mamun

Project Proposal Status

Marks: .... Signature: ....

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

Objective

The primary objective of this lab is to design and implement a Python-based solution to the **Graph Coloring Problem** using a **recursive backtracking approach**. Graph coloring is one of the most significant problems in the field of **graph theory**, and it has wide-ranging applications in areas such as **scheduling**, **resource allocation**, **register assignment in compilers**, **map coloring**, and **network frequency assignment**.

This lab emphasizes developing a practical understanding of:

- **Backtracking algorithms** and how they are applied to constraint satisfaction problems.
- Representing an **undirected graph** using data structures such as an **adjacency matrix**.
- Writing modular, readable, and efficient Python code for recursive solutions.
- Ensuring that color assignments follow strict constraints, where no two connected vertices share the same color.

By the end of this lab, students should be able to:

- Understand and implement a recursive solution to explore all possible colorings of a graph.
- Analyze and visualize how backtracking eliminates invalid choices and searches for a feasible solution.
- Demonstrate the ability to transform theoretical knowledge of graph algorithms into functional code.
- Evaluate whether a given graph can be colored using no more than `K` colors while maintaining adjacency constraints.

This lab not only tests problem-solving and programming skills but also builds a foundational understanding of recursive design and optimization techniques crucial for real-world software development and competitive programming.

Problem Statement

You are given a simple, **undirected graph** that consists of `N` vertices and `M` edges. The vertices are numbered from `0` to `N-1`, and each edge connects a pair of distinct vertices. Additionally, you are provided with an integer `K`, which represents the number of available colors that can be used to color the graph.

The goal is to determine whether it is possible to assign a color to each vertex in the graph such that:

- **No two adjacent vertices** (vertices connected directly by an edge) are assigned the **same color**, and

- The total number of distinct colors used does not exceed `K`.

To solve this, you must implement a **backtracking algorithm**. This approach systematically tries all possible color assignments for the vertices, starting from the first vertex. If at any step a color assignment leads to a conflict (i.e., a vertex has the same color as one of its neighbors), the algorithm **backtracks** and tries a different color.

The key constraints are:

- A vertex can only be colored if the color is **not already used** by any of its adjacent vertices.
- The program must efficiently explore the color space and **terminate early** when a valid coloring is found or all options are exhausted.

The program should:

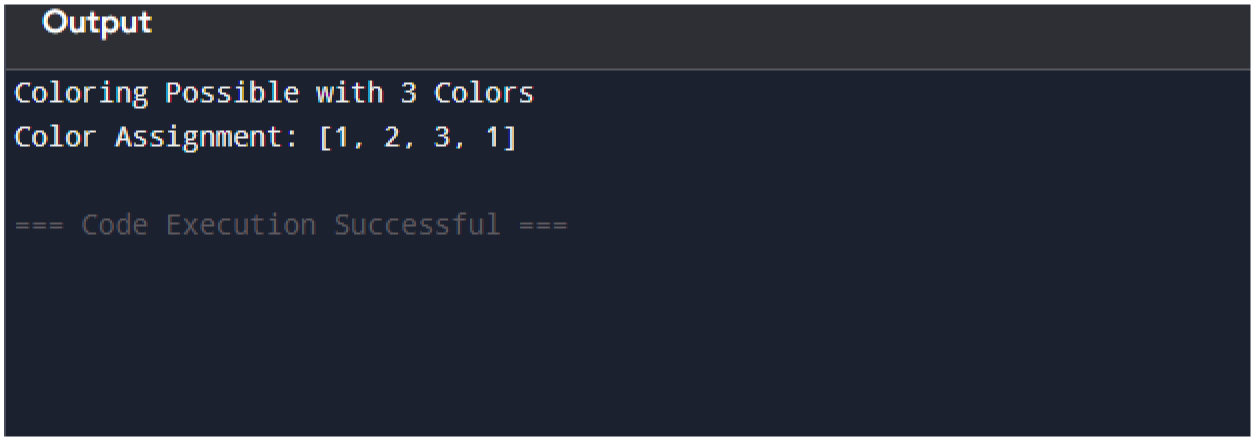
1. Accept the number of vertices `N`, number of edges `M`, and number of colors `K` as input.
2. Accept `M` pairs of integers, where each pair represents an undirected edge between two vertices.
3. Construct the graph using an **adjacency matrix** or any suitable data structure.
4. Apply the recursive backtracking method to attempt coloring the graph.
5. Output whether a valid coloring is possible using at most `K` colors.
6. If coloring is possible, display the color assigned to each vertex.

This problem tests your understanding of recursion, graph theory, algorithm design, and Python programming — all essential topics in computer science.

Let me know if you'd like me to expand **any other section** like the conclusion, or help format it into a Word or PDF version!

Python Code

```
def is_safe(graph, color, c, v):

  for i in range(len(graph)):

    if graph[v][i] == 1 and color[i] == c:

      return False

  return True


def solve(graph, m, color, v):

  if v == len(graph):
        return True
    for c in range(1, m + 1):
        if is_safe(graph, color, c, v):
            color[v] = c
            if solve(graph, m, color, v + 1):
                return True
            color[v] = 0
    return False

def can_color(graph, m):
    color = [0] * len(graph)
    if solve(graph, m, color, 0):
        return True, color
    return False, []

def create_matrix(n, edges):
    g = [[0] * n for _ in range(n)]
    for u, v in edges:
        g[u][v] = 1
        g[v][u] = 1
    return g

# Test case 1
n, m, k = 4, 5, 3
edges = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3)]

graph = create_matrix(n, edges)
ok, result = can_color(graph, k)

if ok:
    print(f"Coloring Possible with {k} Colors")
    print("Color Assignment:", result)
else:
print(f"Coloring Not Possible with {k} Colors")
```



```
Coloring Possible with 3 Colors
Color Assignment: [1, 2, 3, 1]

=== Code Execution Successful ===
```

Conclusion:

This lab effectively demonstrates the practical application of a backtracking algorithm to solve the graph coloring problem, providing valuable insights into recursive problem-solving techniques. The Python program developed in this lab can efficiently determine whether a valid coloring exists for a given undirected graph using exactly **K** colors. The program works by attempting to color each vertex of the graph while ensuring that no two adjacent vertices share the same color, adhering to the constraints of the problem.

By utilizing backtracking, the algorithm systematically explores potential color assignments for each vertex. If a valid coloring is found at any point, the algorithm terminates with a solution. If a conflict arises, the algorithm backtracks and tries alternative colorings. This approach showcases the power of recursion, where the algorithm incrementally builds a solution and revisits previous decisions when necessary.

The lab also provides an opportunity to better understand the importance of constraints in problem-solving. In the case of graph coloring, the algorithm must respect the rule that adjacent nodes cannot share the same color. This constraint-driven approach highlights how backtracking can be used to efficiently explore all possible solutions while pruning invalid ones along the way.

Overall, this lab not only helps in understanding the backtracking technique but also emphasizes the role of recursion and constraint satisfaction in graph-based problem-solving. It serves as a foundation for tackling more complex problems in fields such as scheduling, map coloring, and optimization.