Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
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 D7

Student Details

Name	**ID**
Sabbir Ahmed	221902129

Lab Date: 23-03-2025
Submission Date: 02-04-2025
Course Teacher's Name: Md. Sabbir Hosen Mamun

[For teachers use only: Don't write anything inside this box]

Lab Project Status

Marks:	**Signature:**
Comments:	**Date:**

1 Problem Title

You are given an undirected graph with N vertices and M edges. Your task is to implement a Python program that uses backtracking to determine whether the graph can be colored using K colors so that no two adjacent vertices share the same color. The input will be read from a file, where the first line contains three integers: N (number of vertices, numbered from 0 to N-1), M (number of edges), and K (number of available colors). Each of the following M lines contains two integers u and v, representing an undirected edge between vertex u and vertex v.

2 Objective

The primary objective of this problem is to deepen the understanding of Python programming by implementing a graph coloring algorithm using backtracking. The goal is to determine whether an undirected graph can be colored with K different colors such that no two adjacent vertices share the same color. By tackling this problem, the project reinforces key concepts such as recursion, iterative control structures, and basic graph theory. It also provides practical insights into solving combinatorial problems through systematic exploration of possibilities and backtracking when conflicts arise.

3 M-Coloring Algorithm Pseudocode

This algorithm is inspired by Farhan Hossan (YT channel, 2025). To get a clear understanding on the algorithm, I have collected the algorithm from the YouTube video and implemented it in Python. This algorithm has clearly demonstrated the graph coloring method to find the minimum unique colors used for adjacent nodes.

3.1 Function: nextValue

This function finds the next valid color for vertex *k*, considering previously colored vertices.

Algorithm 1: Find Next Valid Color (`nextValue`)

Purpose: Assign the next lexicographically smallest valid color to vertex k.
Input : Vertex index k
Output : Updates global array $x[k]$ with a valid color, or 0 if no valid color is found.

```
1  Function nextValue(k):
2      While True
           // Loop until a valid color is found or all are checked
3          $x[k] \leftarrow (x[k] + 1) \pmod{m+1}$;
           // Try next color cyclically (1 to m)
4          if $x[k] = 0$
5              return;
               // No more colors available, backtrack
           // Check for conflicts with adjacent, already colored
              vertices
6          let $j = 1$;
7          While $j < k$
8              if $G[k][j] \neq 0$ and $x[k] = x[j]$
9                  break;
                   // Conflict found, break inner loop to try next color
                      for k
10             $j \leftarrow j + 1$;
11         if $j = k$
               // Inner loop completed without break (no conflict)
12             return;
               // Valid color found for vertex k
```

3.2 Function: mColoring

This function recursively assigns colors to vertices using backtracking, starting from vertex k.

Algorithm 2: Recursive M-Coloring (`mColoring`)

Purpose: Find and print all valid m-colorings of the graph.
Input : Current vertex index k to color.
Output : Prints valid color assignments $x[1..n]$ to the output.

```
1 Function mColoring(k):
2     While True
          // Loop to try assigning colors to k and recurse
3         nextValue(k);
          // Find a valid color for vertex k
4         if x[k] = 0
5             return;
              // No valid color found for k, backtrack
6         if k = n
7             write x[1..n];
              // Solution found, print it
8         else
9             mColoring(k + 1);
              // Color assigned, proceed to next vertex
```

4 Python Implementation

```python
def read_input(filename):
    with open(filename, "r") as file:
        first_line = file.readline().strip()
        if not first_line:
            raise ValueError("Empty file")
        n, m, k = map(int, first_line.split())

        G = [[0 for _ in range(n)] for _ in range(n)]
        for _ in range(m):
            u, v = map(int, file.readline().split())
            G[u][v] = 1
            G[v][u] = 1
    return n, k, G

def is_safe(vertex, color, colors, G):
    n = len(G)
    for i in range(n):
        if G[vertex][i] == 1 and colors[i] == color:
            return False
    return True

def m_coloring(vertex, n, m_colors, colors, G):
    if vertex == n:
        return True

    for color in range(1, m_colors + 1):
        if is_safe(vertex, color, colors, G):
            colors[vertex] = color
            if m_coloring(vertex + 1, n, m_colors, colors, G)
                :
                return True
            colors[vertex] = 0
    return False

def solve_graph_coloring(filename):
    n, m_colors, G = read_input(filename)
    colors = [0] * n

    if m_coloring(0, n, m_colors, colors, G):
        print(f"Coloring Possible with {m_colors} Colors")
        print("Color Assignment:", colors)
    else:
        print(f"Coloring Not Possible with {m_colors} Colors"
            )

if __name__ == "__main__":
    input_filename = "input.txt"
    solve_graph_coloring(input_filename)
```

5 Output:

5.1 Test Case 1

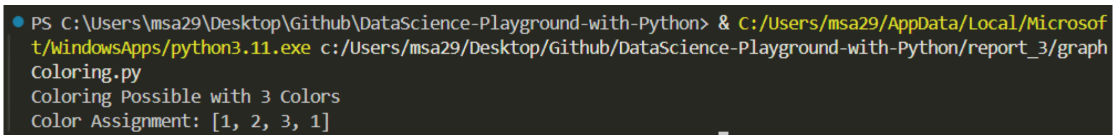
Input.txt

```
4 5 3
0 1
0 2
1 2
1 3
2 3
```

Output:

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

Screenshot



```
PS C:\Users\msa29\Desktop\Github\DataScience-Playground-with-Python> & C:/Users/msa29/AppData/Local/Microsof
t/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Github/DataScience-Playground-with-Python/report_3/graph
Coloring.py
Coloring Possible with 3 Colors
Color Assignment: [1, 2, 3, 1]
```

Figure 1: Test Case 1 - Graph Coloring Implementation

5.2 Test Case 1

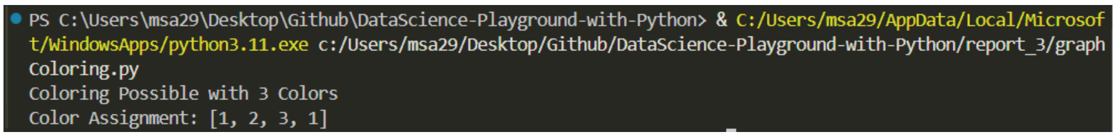
Input_2.txt

```
8 14 3
0 1
0 2
1 2
1 3
2 3
4 5
4 6
5 6
5 7
6 7
0 4
1 5
2 7
3 6
```

Output:

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

Screenshot



```
PS C:\Users\msa29\Desktop\Github\DataScience-Playground-with-Python> & C:/Users/msa29/AppData/Local/Microsof
t/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Github/DataScience-Playground-with-Python/report_3/graph
Coloring.py
Coloring Possible with 3 Colors
Color Assignment: [1, 2, 3, 1]
```

Figure 2: Test Case 2 - Graph Coloring Implementation

6 Discussion

In this implementation, an undirected graph is represented using an adjacency matrix, and backtracking is employed to explore all possible color assignments for each vertex. The algorithm checks, for every vertex, if assigning a particular color violates the graph coloring constraint. If a conflict is detected, the algorithm backtracks to try alternative colors. I have used several functions and built I functions of python. Lets summarize and see what we have:

- **solve_graph_coloring**, it intializes colors and call m_coloring function
- **m_coloring**, this function Recursively tries to color the graph using m_colors. This function is analogous to mColoring(k) in the pseudocode.
- **is_safe**, this function checks if assigning the given color to the current vertex violates any adjacency constraint. This function plays a similar role as checking G[k][j] in the nextValue function.

- Lastly **read_input**, Reads the graph input from a file. The first line contains: N (number of vertices), M (number of edges), K (number of colors) Each of the next M lines contains an edge: u v

7 Conclusion

The code successfully demonstrates how backtracking can be applied to solve the graph coloring problem in Python. The approach effectively navigates through potential color assignments while ensuring that adjacent vertices do not share the same color. This not only reinforces core programming concepts but also provides a foundation for tackling more complex problems in graph theory and combinatorial optimization. While the backtracking method works well for moderate-sized graphs, future improvements might include exploring heuristic-based methods or constraint propagation techniques to handle larger graphs more efficiently. Overall, this code serves as a practical example of combining algorithmic thinking with Python programming skills to solve real-world problems in computer science.

Bibliography

Farhan Hossan. (Youtube). *Graph Coloring Problem | Backtracking | Data Structure & Algorithm | Bangla Tutorial*.

www.interviewbit.com. (2025). *Graph Coloring Problem*.