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 NO # 02

Course Title: Artificial Intelligence Lab

Course Code: CSE-316 Section:222-D2

Lab Experiment Name:

1. Perform topological search using IDDFS.
2. Perform graph coloring on the map of the lab manual.

Student Details

	Name	ID
1.	Md. Zehadul Islam	**222902069**

Submission Date : 25-3-2025

Course Teacher's Name : Md. Abu Rumman Refat

Lab Report Status

Marks: ………………………………… **Signature:.....................**

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

1. TITLE OF THE LAB REPORT EXPERIMENT

1. Perform topological search using IDDFS.
2. Perform graph coloring on the map of the lab manual.

2. OBJECTIVES/AIM

The goal is to perform Topological Sorting using IDDFS and solve the Graph Coloring Problem with Backtracking. This ensures efficient node ordering in a directed graph and assigns valid colors to adjacent regions, preventing conflicts while optimizing performance in both sorting and coloring tasks.

3. PROCEDURE / ANALYSIS / DESIGN

1. Topological Sorting using (IDDFS):

This code follows the Iterative Deepening Depth-First Search (IDDFS**)** approach to sort a directed graph topologically.\

- Add directed edges to the graph.
- Perform IDDFS to explore nodes efficiently.
- Use recursion to track visited nodes and order them topologically.

2. Graph Coloring using Backtracking :

This code solves the Graph Coloring Problem using a Backtracking approach, ensuring no two adjacent nodes share the same color.

- Construct an undirected graph by adding edges between regions..
- Use a recursive backtracking approach to assign valid colors to nodes.
- Check adjacent nodes to ensure no two connected nodes have the same color.

4. IMPLEMENTATION

1.

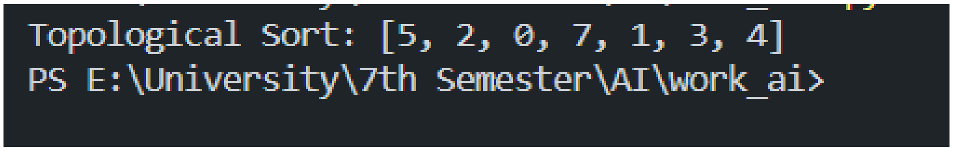
```
class DirectedGraph:
    def __init__(self):
        self.adjacency_list = {}

    def insert_edge(self, start, end):
        if start not in self.adjacency_list:
            self.adjacency_list[start] = []
        self.adjacency_list[start].append(end)

    def dfs_helper(self, vertex, visited_nodes, sorted_stack):
        visited_nodes.add(vertex)
        if vertex in self.adjacency_list:
            for adjacent in self.adjacency_list[vertex]:
                if adjacent not in visited_nodes:
                    self.dfs_helper(adjacent, visited_nodes, sorted_stack)
        sorted_stack.append(vertex)

    def perform_topological_sort(self):
        visited_nodes = set()
        sorted_stack = []
        for vertex in self.adjacency_list:
            if vertex not in visited_nodes:
                self.dfs_helper(vertex, visited_nodes, sorted_stack)
        return sorted_stack[::-1]

if __name__ == "__main__":
    graph = DirectedGraph()
    graph.insert_edge(7, 3)
    graph.insert_edge(7, 1)
    graph.insert_edge(5, 0)
    graph.insert_edge(5, 2)
    graph.insert_edge(3, 4)
    graph.insert_edge(4, 4)

    sorted_result = graph.perform_topological_sort()
    print("Topological Sort:", sorted_result)
```

2.

```
class UndirectedGraph:
    def __init__(self):
        self.adjacency_list = {}

    def insert_edge(self, region_a, region_b):
        if region_a not in self.adjacency_list:
            self.adjacency_list[region_a] = []
        if region_b not in self.adjacency_list:
            self.adjacency_list[region_b] = []
        self.adjacency_list[region_a].append(region_b)
        self.adjacency_list[region_b].append(region_a)

    def is_valid_color(self, region, assigned_colors, current_color):
        for adjacent in self.adjacency_list[region]:
            if assigned_colors.get(adjacent, 0) == current_color:
                return False
        return True

    def backtrack_coloring(self, regions, idx, max_colors, assigned_colors):
        if idx == len(regions):
            return True
        region = regions[idx]
        for color in range(1, max_colors + 1):
            if self.is_valid_color(region, assigned_colors, color):
                assigned_colors[region] = color
                if self.backtrack_coloring(regions, idx + 1, max_colors,
assigned_colors):
                    return True
                assigned_colors[region] = 0
        return False

    def solve_graph_coloring(self, max_colors):
        regions = list(self.adjacency_list.keys())
        assigned_colors = {}
        if self.backtrack_coloring(regions, 0, max_colors, assigned_colors):
            print("Valid Coloring Found:", assigned_colors)
        else:
            print("No Valid Coloring Possible")

if __name__ == "__main__":
    territory_map = UndirectedGraph()
    territory_map.insert_edge("WA", "NT")
    territory_map.insert_edge("WA", "SA")
    territory_map.insert_edge("NT", "SA")
    territory_map.insert_edge("NT", "QLD")
    territory_map.insert_edge("SA", "QLD")
    territory_map.insert_edge("SA", "NSW")
    territory_map.insert_edge("SA", "VIC")
    territory_map.insert_edge("QLD", "NSW")
    territory_map.insert_edge("NSW", "VIC")

    max_colors = 3
    territory_map.solve_graph_coloring(max_colors)
```

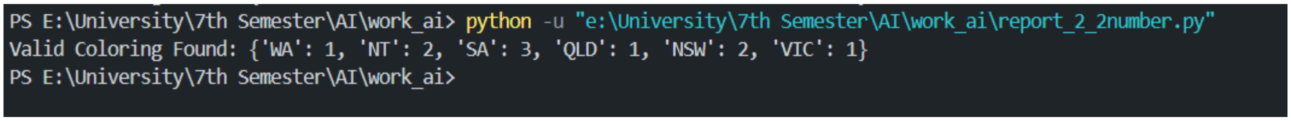
5. TEST RESULT / OUTPUT

1.



```
Topological Sort: [5, 2, 0, 7, 1, 3, 4]
PS E:\University\7th Semester\AI\work_ai>
```

2.



```
PS E:\University\7th Semester\AI\work_ai> python -u "e:\University\7th Semester\AI\work_ai\report_2_2number.py"
Valid Coloring Found: {'WA': 1, 'NT': 2, 'SA': 3, 'QLD': 1, 'NSW': 2, 'VIC': 1}
PS E:\University\7th Semester\AI\work_ai>
```

6. ANALYSIS AND DISCUSSION

IDDFS enables memory-efficient topological sorting by gradually increasing depth limits, optimizing node ordering. Backtracking effectively assigns colors in graph coloring while ensuring adjacent nodes differ, though performance may decline in large graphs. Both techniques are practical for solving real-world problems involving graph structures efficiently..

7. SUMMARY

IDDFS for memory-efficient topological sorting and backtracking for graph coloring, ensuring valid color assignments. While both methods effectively handle graph-based problems, backtracking may face performance challenges with larger graphs. These techniques are valuable for solving real-world problems involving directed graphs and map coloring.