Lab 4

Graphs and Trees

1 Airline Network Shortest-Path Finder

Imagine you are tasked with developing a tool to assist airline passengers in finding the most efficient route between two airports within the airline's network. Create a Java program that models an airline network as a graph, where airports are nodes and flights are edges. Your program should enable users to input details of the flight connections between airports and find the shortest path between a specified source and destination airport.

- Implement a class to represent the airline network graph. Each airport is a node, and flights between airports are edges. You can choose an adjacency matrix or adjacency list to represent the graph.
- Implement Dijkstra's algorithm to find the shortest path between two specified airports in the airline network.
- Display the optimal route details, including the sequence of airports to visit and the total distance or time required for the journey.
- Implement error handling mechanisms to handle cases where the specified source or destination airport is not in the network or when there is no direct flight between them.

Example Input

```
Enter the list of airports: A, B, C, D, E
Enter the flights: A-B, B-C, C-D, D-E, A-C, B-D
The distance for each flight (in miles)
A-B: 500
B-C: 300
C-D: 400
D-E: 600
A-C: 700
B-D: 450
```

```
Enter source airport: A
Enter destination airport: E
```

Example Output

```
Shortest path from A to E: A - B - D - E
Total distance: 1550 miles
```

2 Class Schedule Optimization

Imagine a school with multiple classes and subject timings where certain classes cannot occur simultaneously due to shared resources or teacher availability. Develop a Java program that generates an optimized class schedule by assigning time slots to classes, ensuring that no conflicting classes occur at the same time.

- Represent the schedule information as a graph where nodes represent classes, and edges denote conflicting timings between classes.
- Implement a graph coloring algorithm to assign distinct colors (time slots) to nodes (classes) in the graph. Ensure that adjacent nodes (classes) linked by edges (conflicting timings) do not share the same color (time slot) to avoid scheduling conflicts.
- Display the timetable with color-coded class timings, ensuring that conflicting classes have different colors (non-overlapping timings).
- Use any color names as you like.

Example Input

```
Classes: A, B, C, D
Conflicting classes (cannot occur simultaneously):
  A-B
  B-C
  C-D
  D-B
```

Example Output

```
Optimized Class Schedule:
A - Blue
B - Red
C - Blue
D – Green
```

3 Tree Traversal

Implement three algorithms for Binary Tree traversal recursively or iteratively

- Preorder
- Inorder
- Postorder

Example input

```
Enter the value of the root node:
1
Enter left child value of 1 (or -1 to skip):
2
Enter right child value of 1 (or -1 to skip):
3
Enter left child value of 2 (or -1 to skip):
4
Enter right child value of 2 (or -1 to skip):
5
Enter left child value of 3 (or -1 to skip):
-1
Enter right child value of 3 (or -1 to skip):
6
Enter left child value of 4 (or -1 to skip):
-1
Enter right child value of 4 (or -1 to skip):
-1
Enter left child value of 5 (or -1 to skip):
-1
Enter right child value of 5 (or -1 to skip):
-1
Enter left child value of 6 (or -1 to skip):
-1
Enter right child value of 6 (or -1 to skip):
-1
```

Example Output

```
Preorder Traversal: 1 2 4 5 3 6
Inorder Traversal: 4 2 5 1 3 6
Postorder Traversal: 4 5 2 6 3 1
```

4 Submission

- You must work in **groups of two** and use **Java programming language** in your implementation.
- You should deliver all the coding files.
- Make sure you provide a clear and detailed report. It should contain:
  1. Problem statement.
  2. Used data structures.
  3. Sample runs and different test cases.
  4. Assumptions and details you find necessary to be clarified.

Good Luck„