Day 16 Assignments

1. **Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights.**

```
public int shortestDistance(U a, U b) {

    class Node {
        U v;
        int dis;

        Node(U v, int dis) {
            this.v = v;
            this.dis = dis;
        }
    }

    if (!adjlist.containsKey(a) || !adjlist.containsKey(b)) {
        return -1;
    }

    PriorityQueue<Node> pq = new PriorityQueue<Node>((c, d) -> c.dis - d.dis);
    pq.offer(new Node(a, 0));

    HashMap<U, Integer> distance = new HashMap<>();
    HashMap<U, U> pathTrace = new HashMap<>();
    distance.put(a, 0);
    pathTrace.put(a, a);
    while (!pq.isEmpty()) {
        Node node = pq.poll();
        int dis = node.dis;
        U u = node.v;

        for (U v : adjlist.get(u).keySet()) {
            int cost = adjlist.get(u).get(v);
            if (dis + cost > -2147483648 && dis + cost <
distance.getOrDefault(v, Integer.MAX_VALUE)) {
                distance.put(v, dis + cost);
                pq.offer(new Node(v, dis + cost));
                pathTrace.put(v, u);
            }
        }
    }

    if (!distance.containsKey(b)) {
        return -1;
    }
    System.out.println("Path to take is : ");
```

```
    U dest = b;
    while (dest.compareTo(a) != 0) {
        U source = pathTrace.get(dest);
        System.out.println(dest + " --- " + adjlist.get(dest).get(source) + "
---> " + source);
        dest = source;
    }

    return distance.getOrDefault(b, -1);

}
```

```
package m5_core_java_programming.day_16;

/*
    Code Dijkstra's algorithm to find the shortest path from a start node to
every other node
    in a weighted graph with positive weights.
*/

import m5_core_java_programming.mycollection.Graph;

public class Assignment_1 {
    public static void main(String[] args) {
        Graph<String> graph = new Graph<>();
        graph.addVertex("Delhi");
        graph.addVertex("Mumbai");
        graph.addVertex("Bangalore");
        graph.addVertex("Chennai");
        graph.addVertex("Kolkata");
        graph.addVertex("Hyderabad");

        graph.addEdge("Delhi", "Mumbai", 1400);
        graph.addEdge("Delhi", "Bangalore", 2150);
        graph.addEdge("Delhi", "Kolkata", 1500);
        graph.addEdge("Delhi", "Hyderabad", 1550);
        graph.addEdge("Mumbai", "Bangalore", 980);
        graph.addEdge("Mumbai", "Kolkata", 2050);
        graph.addEdge("Mumbai", "Hyderabad", 710);
        graph.addEdge("Bangalore", "Chennai", 350);
        graph.addEdge("Bangalore", "Hyderabad", 570);
        graph.addEdge("Chennai", "Kolkata", 1670);
        graph.addEdge("Chennai", "Hyderabad", 630);

        System.out.println("Using Dijkstra Distance between Kolkata and
Hyderabad is " + graph.shortestDistance("Kolkata", "Hyderabad"));
    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrai
Path to take is :
Hyderabad --- 630 ---> Chennai
Chennai --- 1670 ---> Kolkata
Distance between Kolkata and Hyderabad is 2300

Process finished with exit code 0
```

---

2. **Implement Kruskal's algorithm to find the minimum spanning tree of a given connected, undirected graph with non-negative edge weights.**

```
public int[] minimumSpanningTree() {
    int[] info = new int[2];

    class Node {
        int v;
        int u;
        int dis;

        Node(int v, int u, int dis) {
            this.u = u;
            this.v = v;
            this.dis = dis;
        }
    }

    //par is just index book to map it with generic objects
    List<U> par = adjlist.keySet().stream().toList();

    int[] parent = new int[par.size()];
    int[] ranks = new int[par.size()];

    for (int i = 0; i < par.size(); i++) {
        parent[i] = i;
    }

    Integer[][] adjMat = createAdjacencyMatrix();

    PriorityQueue<Node> pq = new
PriorityQueue<Node>(Comparator.comparingInt((Node c) -> c.dis));

    for (int i = 0; i < adjMat.length; i++) {
        for (int j = 0; j < adjMat.length; j++) {
```

```
            if (adjMat[i][j] != Integer.MAX_VALUE) {
                pq.offer(new Node(i, j, adjMat[i][j]));
            }
        }
    }

    while (!pq.isEmpty()) {
        Node node = pq.poll();
        int u = node.u;
        int v = node.v;
        int parA = setPar(u, parent);
        int parB = setPar(v, parent);
        if (parA != parB && adjMat[u][v] != Integer.MAX_VALUE) {
            union(u, v, parent, ranks);
            info[0]++;
            info[1] += adjMat[u][v];
            System.out.println(par.get(u) + "----( " + adjMat[u][v] + " )---->"
+ par.get(v));
        }
    }

    return info;
}
```

```
package m5_core_java_programming.day_16;

import m5_core_java_programming.mycollection.Graph;

public class Assignment_2 {
    public static void main(String[] args) {
        Graph<String> graph = new Graph<>();
        graph.addVertex("Delhi");
        graph.addVertex("Mumbai");
        graph.addVertex("Bangalore");
        graph.addVertex("Chennai");
        graph.addVertex("Kolkata");
        graph.addVertex("Hyderabad");

        graph.addEdge("Delhi", "Mumbai", 1400);
        graph.addEdge("Delhi", "Bangalore", 2150);
        graph.addEdge("Delhi", "Kolkata", 1500);
        graph.addEdge("Delhi", "Hyderabad", 1550);
        graph.addEdge("Mumbai", "Bangalore", 980);
        graph.addEdge("Mumbai", "Kolkata", 2050);
        graph.addEdge("Mumbai", "Hyderabad", 710);
        graph.addEdge("Bangalore", "Chennai", 350);
        graph.addEdge("Bangalore", "Hyderabad", 570);
        graph.addEdge("Chennai", "Kolkata", 1670);
        graph.addEdge("Chennai", "Hyderabad", 630);
```

```
        System.out.println("Minimum Spanning Tree of the given graph ");
        int[] info = graph.minimumSpanningTree();
        System.out.println("Number of edges are : " + info[0] + " Total weight
of the edges : " + info[1]);
    }
}
```

Output

```
Minimum Spanning Tree of the given graph
Chennai----( 350 )---->Bangalore
Hyderabad----( 570 )---->Bangalore
Hyderabad----( 710 )---->Mumbai
Delhi----( 1400 )---->Mumbai
Delhi----( 1500 )---->Kolkata
Number of edges are : 5 Total weight of the edges : 4530

Process finished with exit code 0
```

---

3. **Write a Union-Find data structure with path compression. Use this data structure to detect a cycle in an undirected graph.**

```
public boolean hasCycle() {
    class Node {
        int v;
        int u;
        int dis;

        Node(int v, int u, int dis) {
            this.u = u;
            this.v = v;
            this.dis = dis;
        }
    }

    //par is just index book to map it with generic objects
    List<U> par = adjlist.keySet().stream().toList();

    int[] parent = new int[par.size()];
    int[] ranks = new int[par.size()];
```

```
    for (int i = 0; i < par.size(); i++) {
        parent[i] = i;
    }

    Integer[][] adjMat = createAdjacencyMatrix();

    PriorityQueue<Node> pq = new
PriorityQueue<Node>(Comparator.comparingInt((Node c) -> c.dis));

    for (int i = 0; i < adjMat.length; i++) {
        for (int j = 0; j < adjMat.length; j++) {
            if (adjMat[i][j] != Integer.MAX_VALUE) {
                pq.offer(new Node(i, j, adjMat[i][j]));
            }
        }
    }

    while (!pq.isEmpty()) {
        Node node = pq.poll();
        int u = node.u;
        int v = node.v;
        int parA = setPar(u, parent);
        int parB = setPar(v, parent);
        if (parA != parB && adjMat[u][v] != Integer.MAX_VALUE) {
            union(u, v, parent, ranks);
        } else if (parA == parB) {
            System.out.println(par.get(u) + "----( " + adjMat[u][v] + " )---->"
+ par.get(v));
            return true;
        }
    }

    return false;
}
```

```
package m5_core_java_programming.day_16;

import m5_core_java_programming.mycollection.Graph;

public class Assignment_3 {
    public static void main(String[] args) {
        Graph<String> graph = new Graph<>();
        graph.addVertex("Delhi");
        graph.addVertex("Mumbai");
        graph.addVertex("Bangalore");
        graph.addVertex("Chennai");
        graph.addVertex("Kolkata");
        graph.addVertex("Hyderabad");
```

```
        graph.addEdge("Delhi", "Mumbai", 1400);
        graph.addEdge("Delhi", "Bangalore", 2150);
        graph.addEdge("Delhi", "Kolkata", 1500);
        graph.addEdge("Delhi", "Hyderabad", 1550);
        graph.addEdge("Mumbai", "Bangalore", 980);
        graph.addEdge("Mumbai", "Kolkata", 2050);
        graph.addEdge("Mumbai", "Hyderabad", 710);
        graph.addEdge("Bangalore", "Chennai", 350);
        graph.addEdge("Bangalore", "Hyderabad", 570);
        graph.addEdge("Chennai", "Kolkata", 1670);
        graph.addEdge("Chennai", "Hyderabad", 630);

        System.out.println("Possible Edge Causing Cycle in the graph ");
        graph.hasCycle();
    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Pro
Possible Edge Causing Cycle in the graph
Bangalore----( 350 )---->Chennai

Process finished with exit code 0
```

Tools Used :

IntelliJ IDE

java version "1.8.0_411"

Java(TM) SE Runtime Environment (build 1.8.0_411-b09)

Java HotSpot(TM) Client VM (build 25.411-b09, mixed mode, sharing)