Assignment Day-9 & 10

Core Java with DS and Algorithms

Name: Joshnitha Rangolu

Task 1: Dijkstra's Shortest Path Finder

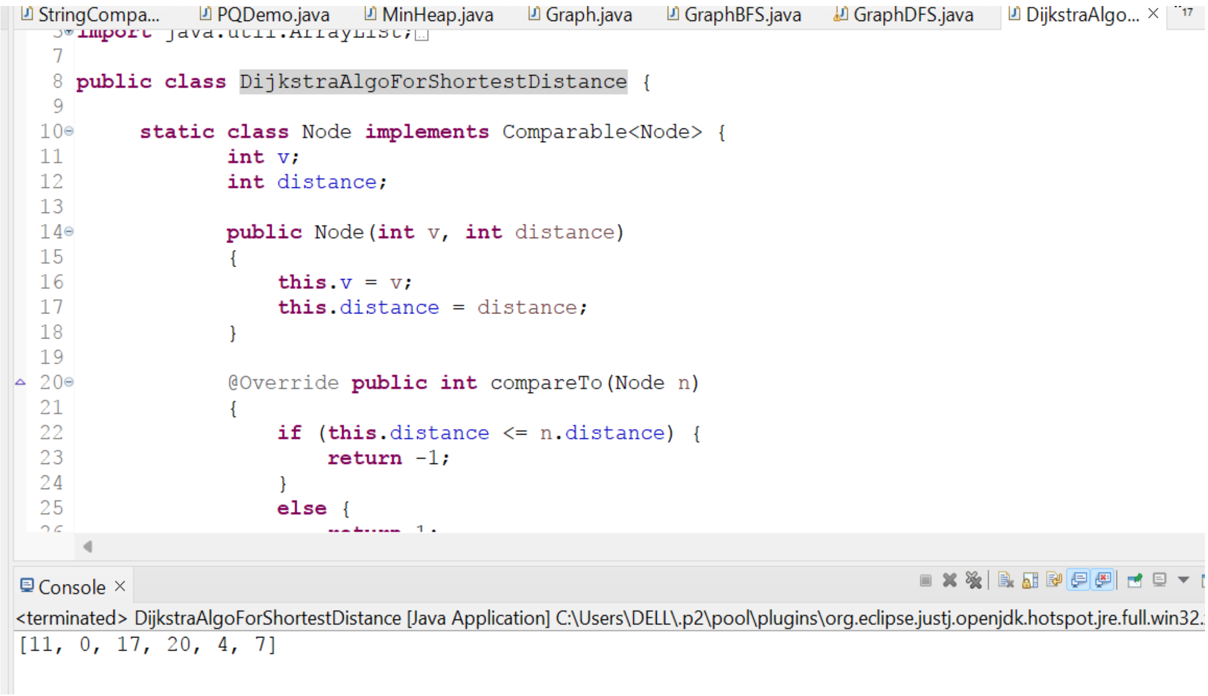
Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights.

```
package day9and10;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.PriorityQueue;

public class DijkstraAlgoForShortestDistance {

        static class Node implements Comparable<Node> {
            int v;
            int distance;

            public Node(int v, int distance)
            {
                this.v = v;
                this.distance = distance;
            }

            @Override public int compareTo(Node n)
            {
                if (this.distance <= n.distance) {
                    return -1;
        }
        else {
            return 1;
        }
    }
}

static int[] dijkstra(
    int V,
    ArrayList<ArrayList<ArrayList<Integer> > > adj,
    int S)
{
    boolean[] visited = new boolean[V];
    HashMap<Integer, Node> map = new HashMap<>();
    PriorityQueue<Node> q = new PriorityQueue<>();

    map.put(S, new Node(S, 0));
    q.add(new Node(S, 0));

    while (!q.isEmpty()) {
        Node n = q.poll();
        int v = n.v;
        int distance = n.distance;
        visited[v] = true;

        ArrayList<ArrayList<Integer> > adjList
            = adj.get(v);
        for (ArrayList<Integer> adjLink : adjList) {

            if (visited[adjLink.get(0)] == false) {
                if (!map.containsKey(adjLink.get(0))) {
```

```
               map.put(
                  adjLink.get(0),
                  new Node(v,
                         distance
                            + adjLink.get(1)));
            }
            else {
               Node sn = map.get(adjLink.get(0));
               if (distance + adjLink.get(1)
                  < sn.distance) {
                  sn.v = v;
                  sn.distance
                     = distance + adjLink.get(1);
               }
            }
            q.add(new Node(adjLink.get(0),
                       distance
                          + adjLink.get(1)));
         }
      }
   }

   int[] result = new int[V];
   for (int i = 0; i < V; i++) {
      result[i] = map.get(i).distance;
   }
   return result;
}

public static void main(String[] args)
{
```

```
ArrayList<ArrayList<ArrayList<Integer> > > adj
    = new ArrayList<>();
HashMap<Integer, ArrayList<ArrayList<Integer> > >
    map = new HashMap<>();

int V = 6;
int E = 5;
int[] u = { 0, 0, 1, 2, 4 };
int[] v = { 3, 5, 4, 5, 5 };
int[] w = { 9, 4, 4, 10, 3 };

for (int i = 0; i < E; i++) {
    ArrayList<Integer> edge = new ArrayList<>();
    edge.add(v[i]);
    edge.add(w[i]);

    ArrayList<ArrayList<Integer> > adjList;
    if (!map.containsKey(u[i])) {
        adjList = new ArrayList<>();
    }
    else {
        adjList = map.get(u[i]);
    }
    adjList.add(edge);
    map.put(u[i], adjList);

    ArrayList<Integer> edge2 = new ArrayList<>();
    edge2.add(u[i]);
    edge2.add(w[i]);

    ArrayList<ArrayList<Integer> > adjList2;
```

```
        if (!map.containsKey(v[i])) {
            adjList2 = new ArrayList<>();
        }
        else {
            adjList2 = map.get(v[i]);
        }
        adjList2.add(edge2);
        map.put(v[i], adjList2);
    }

    for (int i = 0; i < V; i++) {
        if (map.containsKey(i)) {
            adj.add(map.get(i));
        }
        else {
            adj.add(null);
        }
    }
    int S = 1;

    int[] result
        = DijkstraAlgoForShortestDistance.dijkstra(
            V, adj, S);
    System.out.println(Arrays.toString(result));
}
        }
```



```
import java.util.ArrayList;

public class DijkstraAlgoForShortestDistance {

    static class Node implements Comparable<Node> {
        int v;
        int distance;

        public Node(int v, int distance)
        {
            this.v = v;
            this.distance = distance;
        }

        @Override public int compareTo(Node n)
        {
            if (this.distance <= n.distance) {
                return -1;
            }
            else {
                return 1;
```

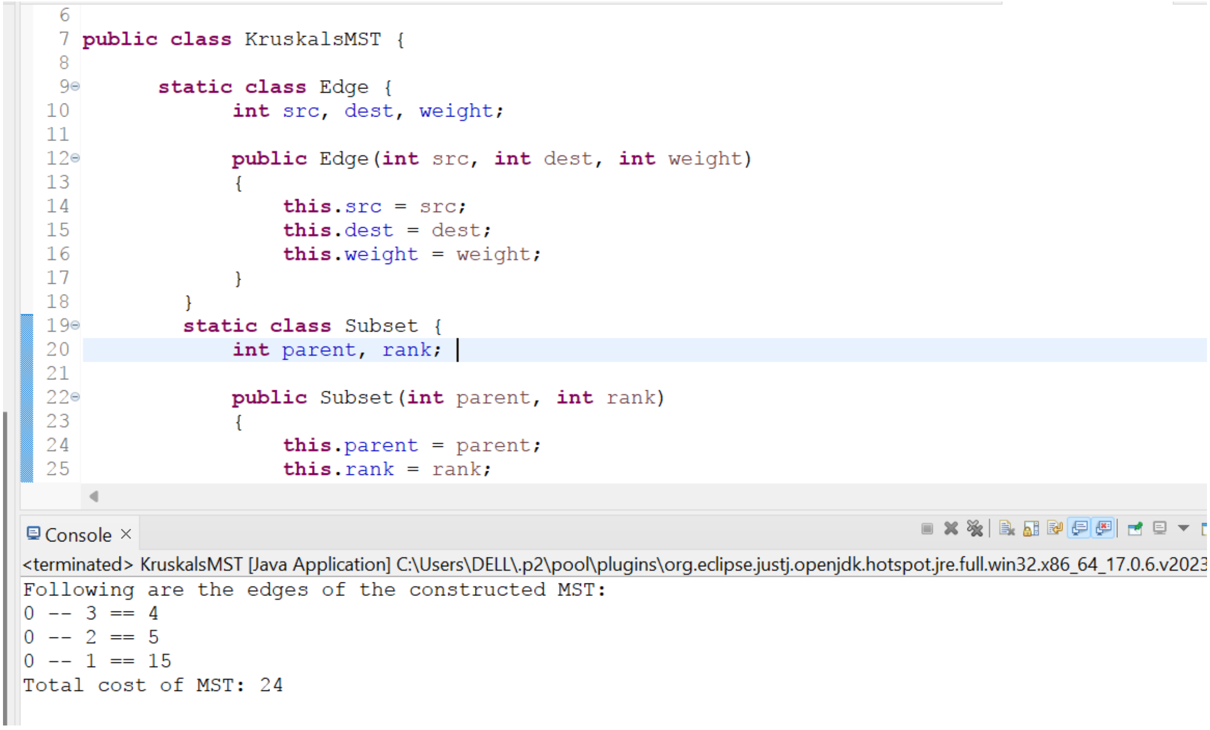
```
<terminated> DijkstraAlgoForShortestDistance [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.
[11, 0, 17, 20, 4, 7]
```

Task 2: Kruskal’s Algorithm for MST

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

```
package day9and10;


import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;


public class KruskalsMST {


        static class Edge {

            int src, dest, weight;


            public Edge(int src, int dest, int weight)

            {

                this.src = src;
```

```
    this.dest = dest;
    this.weight = weight;
  }
}
static class Subset {
  int parent, rank;

  public Subset(int parent, int rank)
  {
    this.parent = parent;
    this.rank = rank;
  }
}
public static void main(String[] args)
{
  int V = 4;
  List<Edge> graphEdges = new ArrayList<Edge>(
    List.of(new Edge(0, 1, 15), new Edge(0, 2, 5),
        new Edge(0, 3, 4), new Edge(1, 3, 18),
        new Edge(2, 3, 5)));
  graphEdges.sort(new Comparator<Edge>() {
    @Override public int compare(Edge o1, Edge o2)
    {
      return o1.weight - o2.weight;
    }
  });

  kruskals(V, graphEdges);
}
private static void kruskals(int V, List<Edge> edges)
{
      int j = 0;
      int noOfEdges = 0;

      Subset subsets[] = new Subset[V];
      Edge results[] = new Edge[V];
      for (int i = 0; i < V; i++) {
        subsets[i] = new Subset(i, 0);
      }
      while (noOfEdges < V - 1) {
        Edge nextEdge = edges.get(j);
        int x = findRoot(subsets, nextEdge.src);
        int y = findRoot(subsets, nextEdge.dest);
        if (x != y) {
          results[noOfEdges] = nextEdge;
          union(subsets, x, y);
          noOfEdges++;
        }

        j++;
      }
      System.out.println(
        "Following are the edges of the constructed MST:");
      int minCost = 0;
      for (int i = 0; i < noOfEdges; i++) {
        System.out.println(results[i].src + " -- "
                  + results[i].dest + " == "
                  + results[i].weight);
        minCost += results[i].weight;
      }
      System.out.println("Total cost of MST: " + minCost);
    }
    private static void union(Subset[] subsets, int x, int y)
    {
          int rootX = findRoot(subsets, x);
      int rootY = findRoot(subsets, y);

      if (subsets[rootY].rank < subsets[rootX].rank) {
        subsets[rootY].parent = rootX;
      }
      else if (subsets[rootX].rank
            < subsets[rootY].rank) {
        subsets[rootX].parent = rootY;
      }
      else {
        subsets[rootY].parent = rootX;
        subsets[rootX].rank++;
      }
    }
    private static int findRoot(Subset[] subsets, int i)
    {
      if (subsets[i].parent == i)
        return subsets[i].parent;

      subsets[i].parent
        = findRoot(subsets, subsets[i].parent);
      return subsets[i].parent;
    }

}
```



```

public class KruskalsMST {

       static class Edge {
             int src, dest, weight;

             public Edge(int src, int dest, int weight)
             {
                 this.src = src;
                 this.dest = dest;
                 this.weight = weight;
             }
         }
         static class Subset {
             int parent, rank;

             public Subset(int parent, int rank)
             {
                 this.parent = parent;
                 this.rank = rank;
```

```
<terminated> KruskalsMST [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v2023
Following are the edges of the constructed MST:
0 -- 3 == 4
0 -- 2 == 5
0 -- 1 == 15
Total cost of MST: 24
```

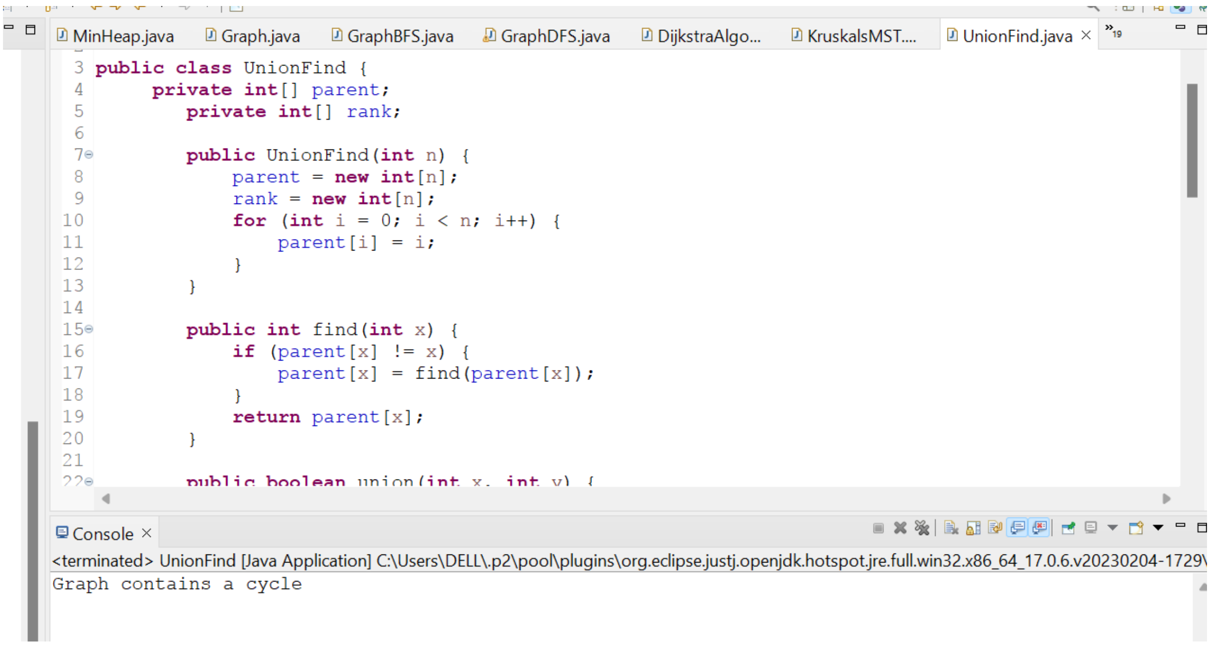
Task 3: Union-Find for Cycle Detection

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

```
package day9and10;

public class UnionFind {

private int[] parent;

private int[] rank;

public UnionFind(int n) {

parent = new int[n];

rank = new int[n];

for (int i = 0; i < n; i++) {

parent[i] = i;

}

}

public int find(int x) {
```

```
if (parent[x] != x) {

parent[x] = find(parent[x]);

}

return parent[x];

}

public boolean union(int x, int y) {

int rootX = find(x);

int rootY = find(y);

if (rootX == rootY) {

return false;

}

if (rank[rootX] < rank[rootY]) {

parent[rootX] = rootY;

} else if (rank[rootX] > rank[rootY]) {

parent[rootY] = rootX;

} else {

parent[rootY] = rootX;

rank[rootX]++;

}

return true;

}

public boolean hasCycle(int[][] edges) {

for (int[] edge : edges) {

int x = edge[0];

int y = edge[1];

if (!union(x, y)) {

return true;

}
```

```java
}
return false;
}
public static void main(String[] args) {
int[][] edges = {{0, 1}, {1, 0}, {2, 1}};
UnionFind uf = new UnionFind(edges.length);
if (uf.hasCycle(edges)) {
System.out.println("Graph contains a cycle");
} else {
System.out.println("Graph does not contain a cycle");
}
}
}
```



```java
public class UnionFind {
     private int[] parent;
        private int[] rank;

        public UnionFind(int n) {
            parent = new int[n];
            rank = new int[n];
            for (int i = 0; i < n; i++) {
                parent[i] = i;
            }
        }

        public int find(int x) {
            if (parent[x] != x) {
                parent[x] = find(parent[x]);
            }
            return parent[x];
        }

        public boolean union(int x, int y) {
```

```
<terminated> UnionFind [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\
Graph contains a cycle
```