

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));
}

}

```

```
StringCompa... PQDemo.java MinHeap.java Graph.java GraphBFS.java GraphDFS.java DijkstraAlgo... × 17
import java.util.ArrayList;
7
8 public class DijkstraAlgoForShortestDistance {
9
10     static class Node implements Comparable<Node> {
11         int v;
12         int distance;
13
14         public Node(int v, int distance)
15         {
16             this.v = v;
17             this.distance = distance;
18         }
19
20         @Override public int compareTo(Node n)
21         {
22             if (this.distance <= n.distance) {
23                 return -1;
24             }
25             else {
26                 return 1;
27             }
28         }
29     }
30
31     // Main method
32     public static void main(String[] args) {
33         // Create a graph with 7 nodes and 11 edges
34         ArrayList<Node> nodes = new ArrayList<Node>();
35         for (int i = 0; i < 7; i++) {
36             nodes.add(new Node(i, 0));
37         }
38         // Create edges
39         ArrayList<Edge> edges = new ArrayList<Edge>();
40         edges.add(new Edge(0, 1, 11));
41         edges.add(new Edge(0, 2, 4));
42         edges.add(new Edge(1, 3, 7));
43         edges.add(new Edge(2, 3, 20));
44         edges.add(new Edge(3, 4, 4));
45         edges.add(new Edge(3, 5, 17));
46         edges.add(new Edge(4, 6, 7));
47         edges.add(new Edge(5, 6, 17));
48
49         // Run Dijkstra's algorithm
50         DijkstraAlgoForShortestDistance algo = new DijkstraAlgoForShortestDistance();
51         algo.run(nodes, edges, 0);
52     }
53
54     // Run Dijkstra's algorithm
55     public void run(ArrayList<Node> nodes, ArrayList<Edge> edges, int source) {
56         // Create a priority queue
57         PriorityQueue<Node> pq = new PriorityQueue<Node>();
58         // Add source node to the queue
59         pq.add(nodes.get(source));
60         // Create a visited array
61         boolean[] visited = new boolean[nodes.size()];
62         // Run the algorithm
63         while (!pq.isEmpty()) {
64             Node u = pq.poll();
65             visited[u.v] = true;
66             // Get all edges of u
67             ArrayList<Edge> edgesOfu = new ArrayList<Edge>();
68             for (Edge e : edges) {
69                 if (e.src == u.v) {
70                     edgesOfu.add(e);
71                 }
72             }
73             // For each edge, calculate the distance to the destination
74             for (Edge e : edgesOfu) {
75                 Node v = nodes.get(e.dest);
76                 int newDistance = u.distance + e.weight;
77                 if (!visited[v.v] && newDistance < v.distance) {
78                     v.distance = newDistance;
79                     pq.add(v);
80                 }
81             }
82         }
83     }
84 }
```

Console ×

<terminated> DijkstraAlgoForShortestDistance [Java Application] C:\Users\DELL\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jdk\bin\java.exe -Xmx1024m -Xms128m -Djava.library.path=C:\Users\DELL\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jdk\bin\java.exe [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;
}
}

```

```

6
7 public class KruskalsMST {
8
9     static class Edge {
10         int src, dest, weight;
11
12         public Edge(int src, int dest, int weight)
13         {
14             this.src = src;
15             this.dest = dest;
16             this.weight = weight;
17         }
18     }
19     static class Subset {
20         int parent, rank;
21
22         public Subset(int parent, int rank)
23         {
24             this.parent = parent;
25             this.rank = rank;
26         }
27     }
28 }

```

```
<terminated> KruskalsMST [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v2023-09-20\bin\java.exe -Xmx2G -Djava.awt.headless=true -jar ...  
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;

}

}

```

```

    }

    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");


    }

}

}

}

```



The screenshot shows the Eclipse IDE with the 'UnionFind.java' file open. The code in the file is as follows:

```

3 public class UnionFind {
4     private int[] parent;
5     private int[] rank;
6
7     public UnionFind(int n) {
8         parent = new int[n];
9         rank = new int[n];
10        for (int i = 0; i < n; i++) {
11            parent[i] = i;
12        }
13    }
14
15    public int find(int x) {
16        if (parent[x] != x) {
17            parent[x] = find(parent[x]);
18        }
19        return parent[x];
20    }
21
22    public boolean union(int x, int y) {

```

The console at the bottom shows the output of the program:

```

<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

```