Day - 9 and 10

1] 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.

Solution:-

Code -

WeightedGrapheg.java

```
1  package com.wipro.graph;
2
3  /*import java.util.ArrayList;
59
60
61
62 import java.util.ArrayList;
66
67 public class WeightedGrapheg {
68     private HashMap<String, HashMap<String,Integer>> adjList = new HashMap<>();
69
70     private HashMap<String, String> previous = new HashMap<>();
71
72     public static void main(String[] args) {
73         WeightedGrapheg myGraph = new WeightedGrapheg();
74         myGraph.addVertex("A");
75         myGraph.addVertex("B");
76         myGraph.addVertex("C");
77         myGraph.addVertex("D");
78         myGraph.addVertex("E");
79         myGraph.addVertex("F");
80
81         myGraph.addEdge("A", "B", 2);
82         myGraph.addEdge("A", "D", 8);
83         myGraph.addEdge("B", "E", 6);
84         myGraph.addEdge("B", "D", 5);
85         myGraph.addEdge("E", "D", 3);
86         myGraph.addEdge("E", "F", 1);
87         myGraph.addEdge("E", "C", 9);
88         myGraph.addEdge("D", "F", 2);
89         myGraph.addEdge("F", "C", 3);
90
91         myGraph.printGraph();
92
```

```
89          myGraph.addEdge("F", "C", 3);
90
91          myGraph.printGraph();
92
93          myGraph.dijkstra("A");
94
95          ArrayList<String> shortestPathToC = myGraph.getShortestPathTo("C");
96          System.out.println("Shortest path from A to C: " + shortestPathToC);
97      }
98
99      private void addVertex(String vertex) {
100          adjList.putIfAbsent(vertex, new HashMap<>());
101     }
102
103     public boolean addEdge(String vertex1, String vertex2, int weight) {
104
105         if (adjList.containsKey(vertex1) && adjList.containsKey(vertex2)) {
106
107             adjList.get(vertex1).put(vertex2, weight);
108             adjList.get(vertex2).put(vertex1, weight);
109             return true;
110         }
111         return false;
112     }
113
114     private void printGraph() {
115         System.out.println("Graph:");
116         for (Map.Entry<String, HashMap<String, Integer>> entry : adjList.entrySet()) {
117             System.out.println(entry.getKey() + " -> " + entry.getValue());
118         }
119
120     }
121
```

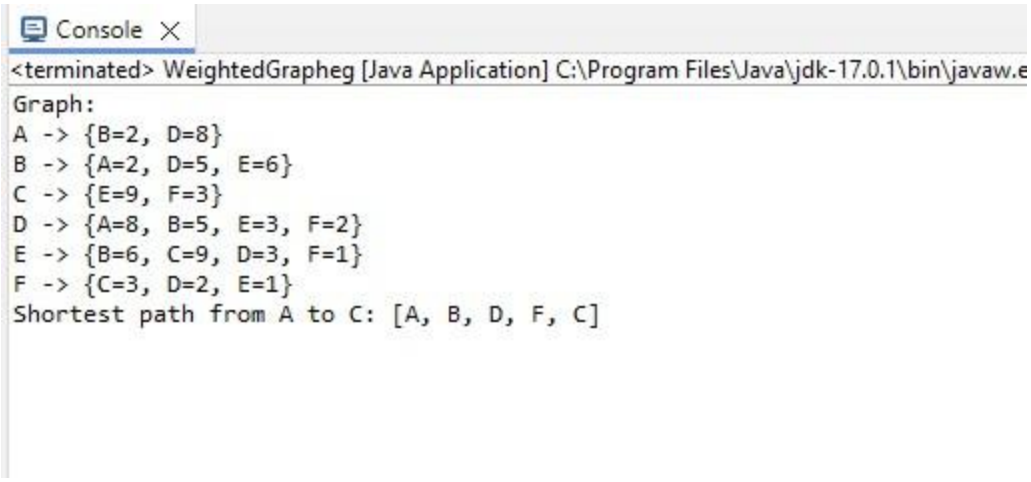
WeightedGrapheg.java

```
119
120     }
121
122     private void dijkstra(String start) {
123         HashMap<String, Integer> distance = new HashMap<>();
124         PriorityQueue<VertexDistancePair> pq = new PriorityQueue<>();
125
126         for (String vertex : adjList.keySet()) {
127             distance.put(vertex, Integer.MAX_VALUE);
128             previous.put(vertex, null);
129         }
130
131         distance.put(start, 0);
132         pq.offer(new VertexDistancePair(start, 0));
133
134         while (!pq.isEmpty()) {
135             VertexDistancePair currentPair = pq.poll();
136             String current = currentPair.vertex;
137
138             for (Map.Entry<String, Integer> neighborEntry : adjList.get(current).entrySet()) {
139                 String neighbor = neighborEntry.getKey();
140                 int weight = neighborEntry.getValue();
141                 int newDistance = distance.get(current) + weight;
142
143                 if (newDistance < distance.get(neighbor)) {
144                     distance.put(neighbor, newDistance);
145                     previous.put(neighbor, current);
146                     pq.offer(new VertexDistancePair(neighbor, newDistance));
147                 }
148             }
149         }
150     }
151
152     private ArrayList<String> getShortestPathTo(String destination) {
153         ArrayList<String> path = new ArrayList<>();
154         String current = destination;
155         while (current != null) {
156             path.add(0, current);
157             current = previous.get(current);
```

WeightedGrapheg.java

```
                int weight = neighborEntry.getValue();
                int newDistance = distance.get(current) + weight;

                if (newDistance < distance.get(neighbor)) {
                    distance.put(neighbor, newDistance);
                    previous.put(neighbor, current);
                    pq.offer(new VertexDistancePair(neighbor, newDistance));
                }
            }
        }
    }

    private ArrayList<String> getShortestPathTo(String destination) {
        ArrayList<String> path = new ArrayList<>();
        String current = destination;
        while (current != null) {
            path.add(0, current);
            current = previous.get(current);
        }
        return path;
    }

    private static class VertexDistancePair implements Comparable<VertexDistancePair> {
        String vertex;
        int distance;

        VertexDistancePair(String vertex, int distance) {
            this.vertex = vertex;
            this.distance = distance;
        }

        @Override
        public int compareTo(VertexDistancePair other) {
            return Integer.compare(this.distance, other.distance);
        }
    }

}

```

Output -



2] 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.

Solution:-
Code -

KrusklAlgo.java

```java
package com.wipro.graph;
import java.util.*;

class Edge implements Comparable<Edge> {
    int src, dest, weight;

    public Edge(int src, int dest, int weight) {
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }

    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;
    }
}

class Subset {
    int parent, rank;
}

public class KrusklAlgo {
    int vertices, edges;
    Edge[] edge;
```

KrusklAlgo.java

```
154
155
156     KrusklAlgo(int vertices, int edges) {
157         this.vertices = vertices;
158         this.edges = edges;
159         edge = new Edge[edges];
160         for (int i = 0; i < edges; ++i) {
161             edge[i] = new Edge(0, 0, 0);
162         }
163     }
164
165
166     int find(Subset[] subsets, int i) {
167         if (subsets[i].parent != i)
168             subsets[i].parent = find(subsets, subsets[i].parent);
169         return subsets[i].parent;
170     }
171
172
173     void union(Subset[] subsets, int x, int y) {
174         int rootX = find(subsets, x);
175         int rootY = find(subsets, y);
176
177
178         if (subsets[rootX].rank < subsets[rootY].rank)
179             subsets[rootX].parent = rootY;
180         else if (subsets[rootX].rank > subsets[rootY].rank)
```

KrusklAlgo.java

```
176
177
178         if (subsets[rootX].rank < subsets[rootY].rank)
179             subsets[rootX].parent = rootY;
180         else if (subsets[rootX].rank > subsets[rootY].rank)
181             subsets[rootY].parent = rootX;
182         else {
183             subsets[rootY].parent = rootX;
184             subsets[rootX].rank++;
185         }
186     }
187
188
189     void kruskalMST() {
190         Edge[] result = new Edge[vertices];
191         int e = 0;
192         int i = 0;
193         for (i = 0; i < vertices; ++i)
194             result[i] = new Edge(0, 0, 0);
195
196
197         Arrays.sort(edge);
198
199
200         Subset[] subsets = new Subset[vertices];
201         for (i = 0; i < vertices; ++i)
202             subsets[i] = new Subset();
```

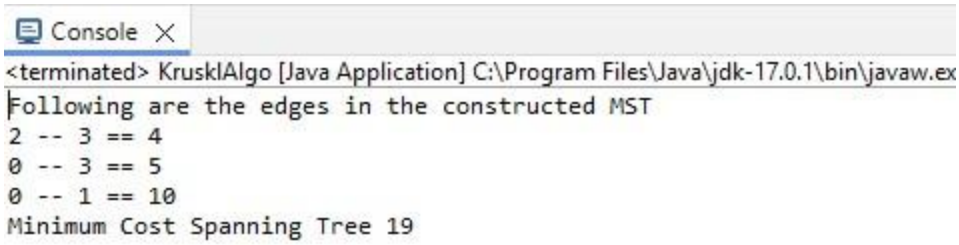
*KrusklAlgo.java

```
198
199
200         Subset[] subsets = new Subset[vertices];
201         for (i = 0; i < vertices; ++i)
202             subsets[i] = new Subset();
203
204
205         for (int v = 0; v < vertices; ++v) {
206             subsets[v].parent = v;
207             subsets[v].rank = 0;
208         }
209
210         i = 0;
211  while (e < vertices - 1) {
212
213             Edge nextEdge = edge[i++];
214
215             int x = find(subsets, nextEdge.src);
216             int y = find(subsets, nextEdge.dest);
217
218             if (x != y) {
219                 result[e++] = nextEdge;
220                 union(subsets, x, y);
221             }
222
223         }
```

*KrusklAlgo.java

```
220                 union(subsets, x, y);
221             }
222
223         }
224 System.out.println("Following are the edges in the constructed MST");
225         int minimumCost = 0;
226         for (i = 0; i < e; ++i) {
227             System.out.println(result[i].src + " -- " + result[i].dest + " == " + result[i].weight);
228             minimumCost += result[i].weight;
229         }
230         System.out.println("Minimum Cost Spanning Tree " + minimumCost);
231     }
232
233     public static void main(String[] args) {
234         int vertices = 4;
235         int edges = 5;
236         KrusklAlgo graph = new KrusklAlgo(vertices, edges);
237
238         graph.edge[0] = new Edge(0, 1, 10);
239         graph.edge[1] = new Edge(0, 2, 6);
240         graph.edge[2] = new Edge(0, 3, 5);
241         graph.edge[3] = new Edge(1, 3, 15);
242         graph.edge[4] = new Edge(2, 3, 4);
243         graph.kruskalMST();
244     }
245 }
246
```

Output -



3] 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.

Solution:-
Code -

CycleDetect.java

```java
package com.wipro.graphalgo;

import java.util.Arrays;

class UnionFind {
    int[] parent;
    int[] rank;

    UnionFind(int n) {
        parent = new int[n];
        rank = new int[n];
        Arrays.fill(rank, 1);
        for(int i=0; i<n ;i++) {
            parent[i] =i;
        }

    }

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

    void union(int x, int y) {
        int rootX = find(x);
```

CycleDetect.java

```
25
26      void union(int x, int y) {
27          int rootX = find(x);
28          int rootY = find(y);
29
30          if (rootX != rootY) {
31              if (rank[rootX] < rank[rootY]) { // 1<2
32                  parent[rootX] = rootY;
33              } else if (rank[rootX] > rank[rootY]) {
34                  parent[rootY] = rootX;
35              } else {
36                  parent[rootY] = rootX;
37                  rank[rootX]++;
38
39              }
40
41          }
42
43      }
44
45  }
46
47  class Graph {
48      int V, E;
49      Edge[] edges;
50
```

CycleDetect.java

```
46
47 class Graph {
48     int V, E;
49     Edge[] edges;
50
51     class Edge {
52         int src, dest;
53     }
54
55     Graph(int v, int e) {
56         this.V = v;
57         this.E = e;
58         this.edges = new Edge[E];
59         for (int i = 0; i < e; i++) {
60             edges[i] = new Edge();
61             System.out.println(edges[i].src + "  -- " + edges[i].dest);
62         }
63     }
64
65     public boolean isCycleFound(Graph graph) {
66         UnionFind uf = new UnionFind(V);
67         for(int i=0; i< E ; ++i) {
68             int x = find(uf, graph.edges[i].src);
69             int y = find(uf, graph.edges[i].dest);
70
71             if(x==y) {
72                 return true;
```

CycleDetect.java

```
70
71              if(x==y) {
72                  return true;
73              }
74              uf.union(x, y);
75          }
76          return false;
77      }
78
79      private int find(UnionFind uf, int i) {
80
81          return uf.find(i);
82      }
83
84  }
85
86  public class CycleDetect {
87      public static void main(String[] args) {
88          int V = 3, E = 3;
89          //int V = 3, E = 2;
90          Graph graph = new Graph(V, E);
91
92          graph.edges[0].src = 0;
93          graph.edges[0].dest = 1;
94
95          graph.edges[1].src = 1;
96          graph.edges[1].dest = 2;
```

CycleDetect.java

```
90          Graph graph = new Graph(V, E);
91
92          graph.edges[0].src = 0;
93          graph.edges[0].dest = 1;
94
95          graph.edges[1].src = 1;
96          graph.edges[1].dest = 2;
97
98          graph.edges[2].src = 0;
99          graph.edges[2].dest = 2;
100
101         System.out.println(graph.V + " -- " + graph.E);
102         for (int i = 0; i < E; i++) {
103
104             System.out.println(graph.edges[i].src + "  -- " + graph.edges[i].dest);
105         }
106
107
108         if(graph.isCycleFound(graph)) {
109             System.out.println("Cycle Found");
110         }else {
111             System.out.println("Cycle Not Found...");
112         }
113
114     }
115 }
```

Output -

Console

<terminated> CycleDetect (1) [Java Application] C:\Program Files\Java\jdk-

```
0  -- 0
0  -- 0
0  -- 0
3 -- 3
0  -- 1
1  -- 2
0  -- 2
Cycle Found
```