Assignment-1

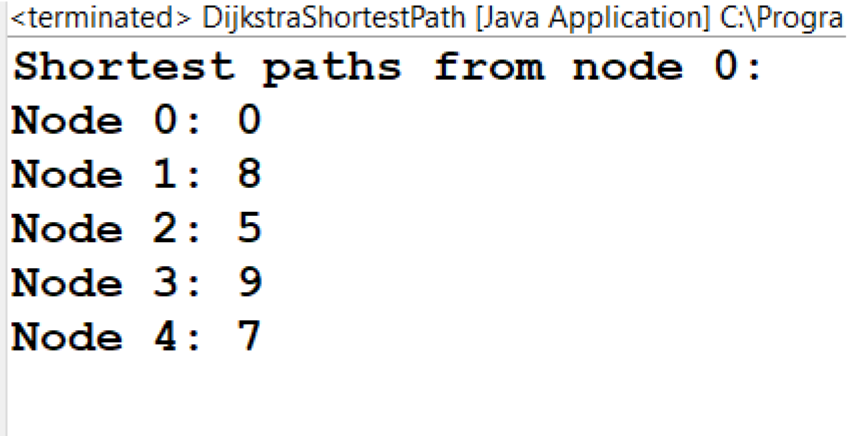
Dijkstra’s Shortest Path Finder

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:

```
package com.wipro.day9and10;

import java.util.*;

public class DijkstraShortestPath {
    static class Edge {
        int to, weight;

        Edge(int to, int weight) {
            this.to = to;
            this.weight = weight;
        }
    }

    public static int[] dijkstra(List<List<Edge>>
graph, int start) {
        int n = graph.size();
        int[] dist = new int[n];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[start] = 0;

        PriorityQueue<Edge> pq = new
PriorityQueue<>(Comparator.comparingInt(a ->
a.weight));
        pq.offer(new Edge(start, 0));

        while (!pq.isEmpty()) {
            Edge curr = pq.poll();
            int u = curr.to;
            for (Edge next : graph.get(u)) {
                int v = next.to;
                int newDist = dist[u] + next.weight;
                if (newDist < dist[v]) {
                    dist[v] = newDist;
                    pq.offer(new Edge(v, newDist));
                }
            }
        }
        return dist;
    }

    public static void main(String[] args) {
        int n = 5;
        List<List<Edge>> graph = new ArrayList<>(n);
        for (int i = 0; i < n; i++) {
            graph.add(new ArrayList<>());
        }

        graph.get(0).add(new Edge(1, 10));
        graph.get(0).add(new Edge(2, 5));
        graph.get(1).add(new Edge(2, 2));
        graph.get(1).add(new Edge(3, 1));
        graph.get(2).add(new Edge(1, 3));
        graph.get(2).add(new Edge(3, 9));
        graph.get(2).add(new Edge(4, 2));
        graph.get(3).add(new Edge(4, 4));

        int startNode = 0;
        int[] shortestPaths = dijkstra(graph,
startNode);

        System.out.println("Shortest paths from node
" + startNode + ":");
        for (int i = 0; i < shortestPaths.length;
i++) {
            System.out.println("Node " + i + ": " +
shortestPaths[i]);
        }
    }
}
```

OUTPUT:



```
<terminated> DijkstraShortestPath [Java Application] C:\Progra
Shortest paths from node 0:
Node 0: 0
Node 1: 8
Node 2: 5
Node 3: 9
Node 4: 7
```

Assignment-2

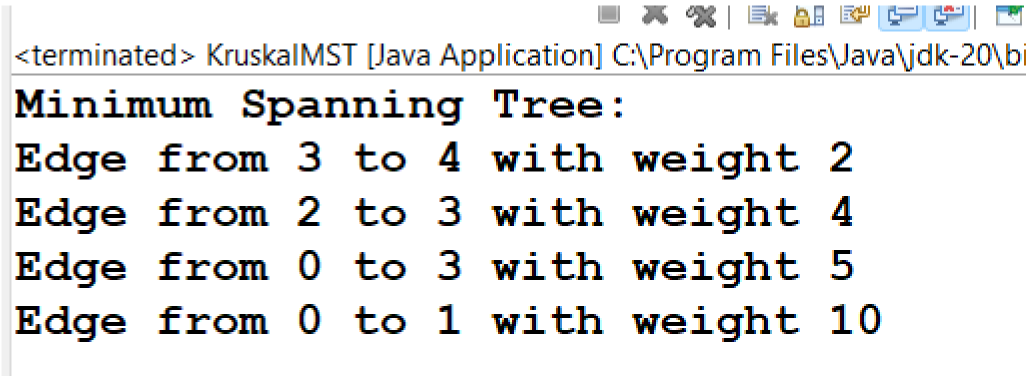
Kruskal's Algorithm for MST

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

Solution:

```java
package com.wipro.day9and10;

import java.util.*;

public class KruskalMST {
    static class Edge {
        int from, to, weight;

        Edge(int from, int to, int weight) {
            this.from = from;
            this.to = to;
            this.weight = weight;
        }
    }

    public static List<Edge> kruskal(List<Edge>
edges, int n) {
        List<Edge> mst = new ArrayList<>();
        Collections.sort(edges,
Comparator.comparingInt(a -> a.weight));
        int[] parent = new int[n];
        Arrays.fill(parent, -1);

        for (Edge edge : edges) {
            int x = find(parent, edge.from);
            int y = find(parent, edge.to);
            if (x != y) {
                mst.add(edge);
                union(parent, x, y);
            }
        }
        return mst;
    }

    public static int find(int[] parent, int i) {
        if (parent[i] == -1) {
            return i;
        }
        return find(parent, parent[i]);
    }

    public static void union(int[] parent, int x, int
y) {
        int xRoot = find(parent, x);
        int yRoot = find(parent, y);
        parent[xRoot] = yRoot;
    }

    public static void main(String[] args) {
        int n = 5;
        List<Edge> edges = new ArrayList<>();
        edges.add(new Edge(0, 1, 10));
        edges.add(new Edge(0, 2, 6));
        edges.add(new Edge(0, 3, 5));
        edges.add(new Edge(1, 3, 15));
        edges.add(new Edge(2, 3, 4));
        edges.add(new Edge(2, 4, 8));
        edges.add(new Edge(3, 4, 2));

        List<Edge> mst = kruskal(edges, n);

        System.out.println("Minimum Spanning Tree:");
        for (Edge edge : mst) {
            System.out.println("Edge from " + 
edge.from + " to " + edge.to + " with weight " + 
edge.weight);
        }
    }
}
```

OUTPUT:



```
<terminated> KruskalMST [Java Application] C:\Program Files\Java\jdk-20\bi
Minimum Spanning Tree:
Edge from 3 to 4 with weight 2
Edge from 2 to 3 with weight 4
Edge from 0 to 3 with weight 5
Edge from 0 to 1 with weight 10
```

Assignment-3

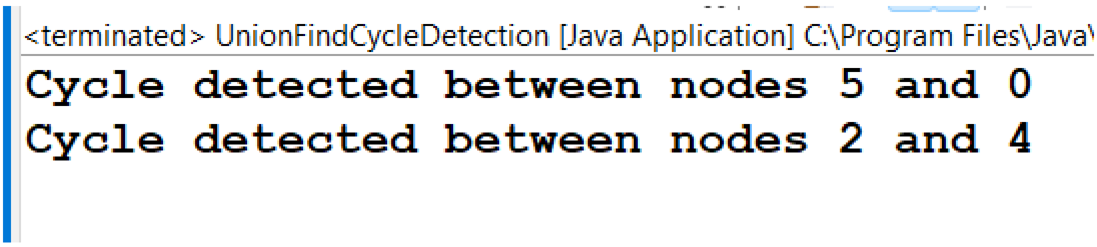
Union-Find for Cycle Detection

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

Solution:

```
package com.wipro.day9and10;

public class UnionFindCycleDetection {
    static class UnionFind {
        int[] parent;
        int[] rank;

        UnionFind(int n) {
            parent = new int[n];
            rank = new int[n];
            for (int i = 0; i < n; i++) {
                parent[i] = i;
            }
        }
        int find(int x) {
            if (parent[x] == x)
                return x;
            return parent[x] = find(parent[x]);
        }
        void union(int x, int y) {
            int rootX = find(x);
            int rootY = find(y);
            if (rootX == rootY) {
                // A cycle is detected
                System.out.println("Cycle detected
between nodes " + x + " and " + y);
                return;
            }
            if (rank[rootX] < rank[rootY]) {
                parent[rootX] = rootY;
            } else if (rank[rootX] > rank[rootY]) {
                parent[rootY] = rootX;
            } else {
                parent[rootY] = rootX;
                rank[rootX]++;
            }
        }
    }

    public static void main(String[] args) {
        int n = 6;
        UnionFind uf = new UnionFind(n);

        int[][] edges = { { 0, 1 }, { 1, 2 }, { 2, 3
}, { 3, 4 }, { 4, 5 }, { 5, 0 }, { 2, 4 } };

        for (int[] edge : edges) {
            int from = edge[0];
            int to = edge[1];
            uf.union(from, to);
        }
    }
}
```

OUTPUT:



```
<terminated> UnionFindCycleDetection [Java Application] C:\Program Files\Java\
Cycle detected between nodes 5 and 0
Cycle detected between nodes 2 and 4
```