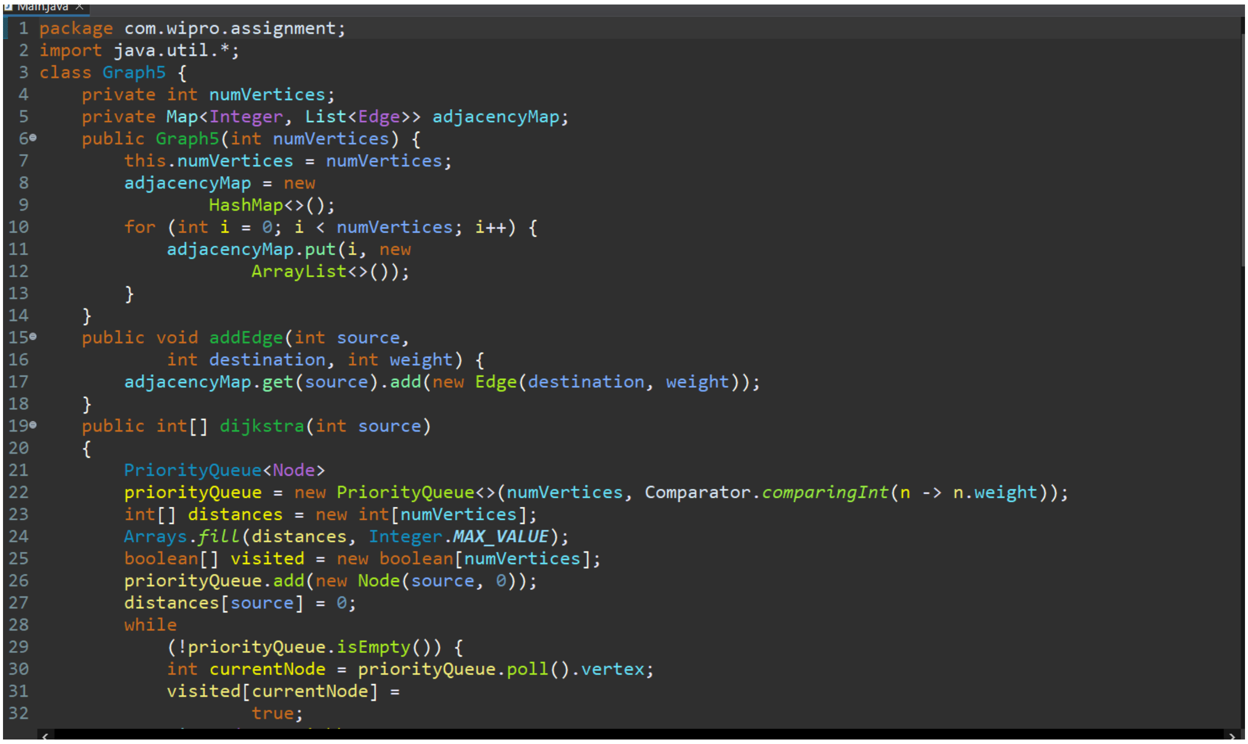
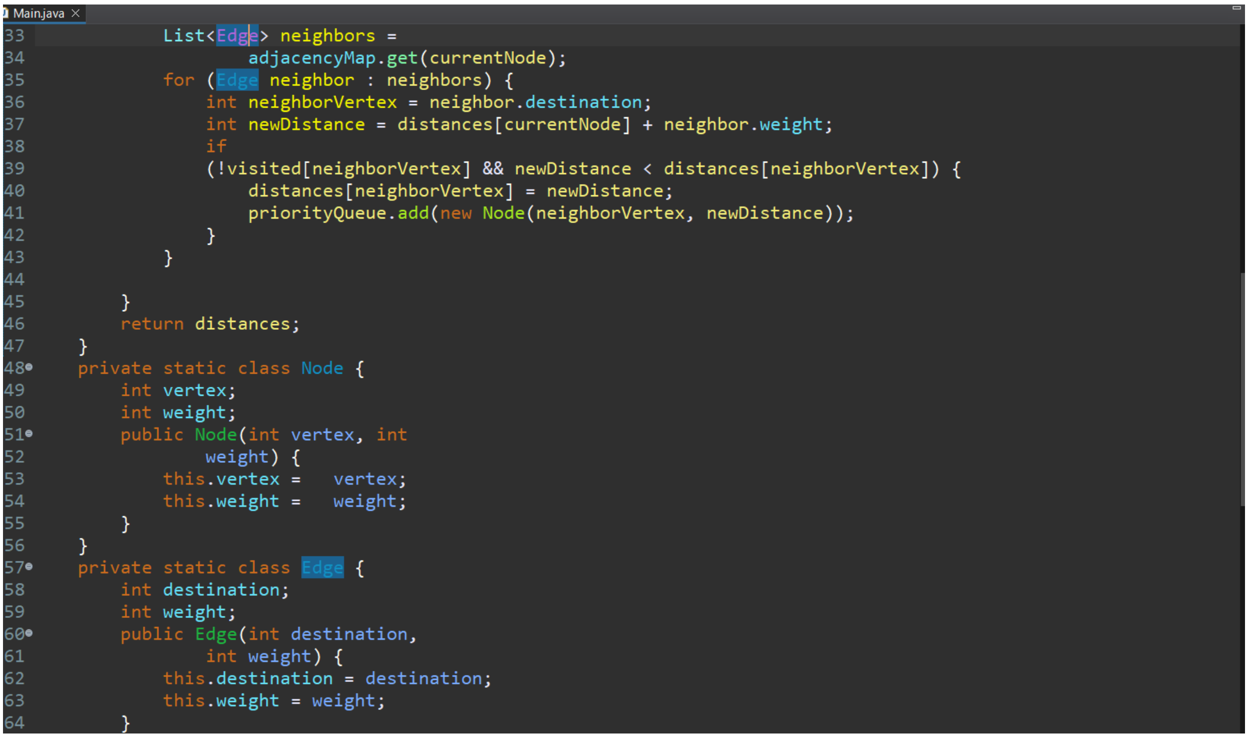
Task 1: Dijkstra’s Shortest Path Finder

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

Code: -

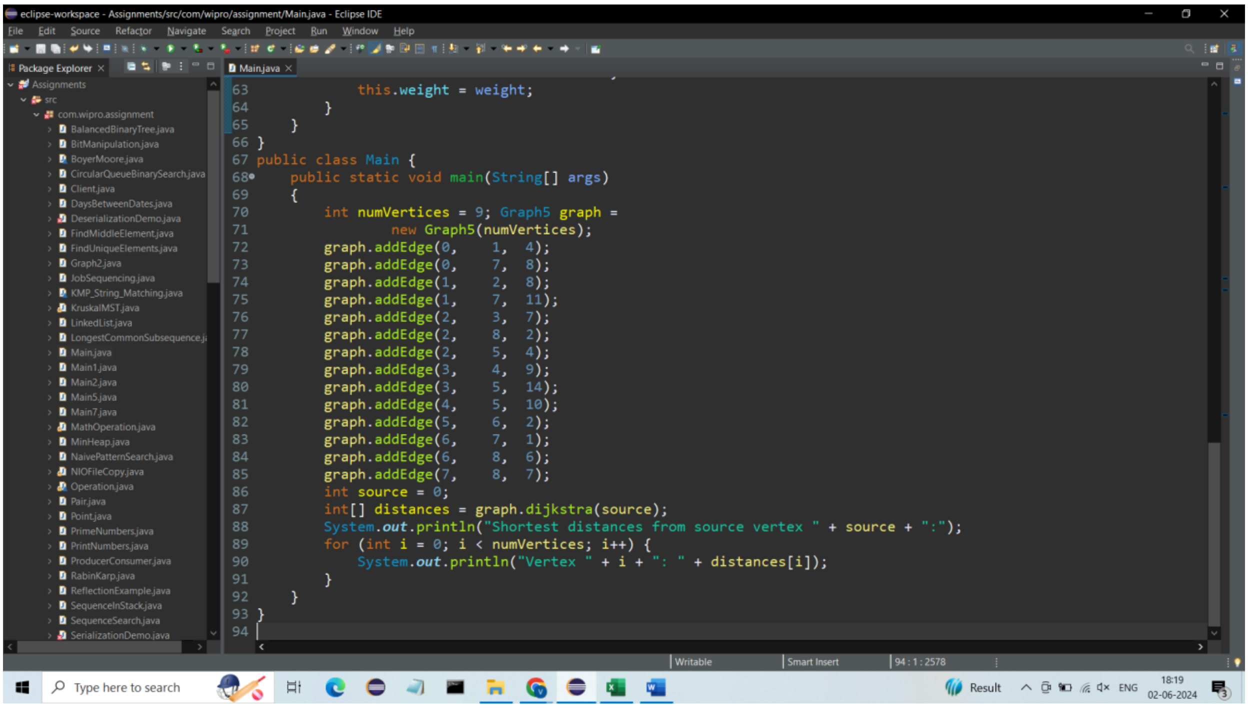


```java
package com.wipro.assignment;
import java.util.*;
class Graph5 {
    private int numVertices;
    private Map<Integer, List<Edge>> adjacencyMap;
    public Graph5(int numVertices) {
        this.numVertices = numVertices;
        adjacencyMap = new
                HashMap<>();
        for (int i = 0; i < numVertices; i++) {
            adjacencyMap.put(i, new
                    ArrayList<>());
        }
    }
    public void addEdge(int source,
            int destination, int weight) {
        adjacencyMap.get(source).add(new Edge(destination, weight));
    }
    public int[] dijkstra(int source)
    {
        PriorityQueue<Node>
        priorityQueue = new PriorityQueue<>(numVertices, Comparator.comparingInt(n -> n.weight));
        int[] distances = new int[numVertices];
        Arrays.fill(distances, Integer.MAX_VALUE);
        boolean[] visited = new boolean[numVertices];
        priorityQueue.add(new Node(source, 0));
        distances[source] = 0;
        while
            (!priorityQueue.isEmpty()) {
            int currentNode = priorityQueue.poll().vertex;
            visited[currentNode] =
                    true;
```



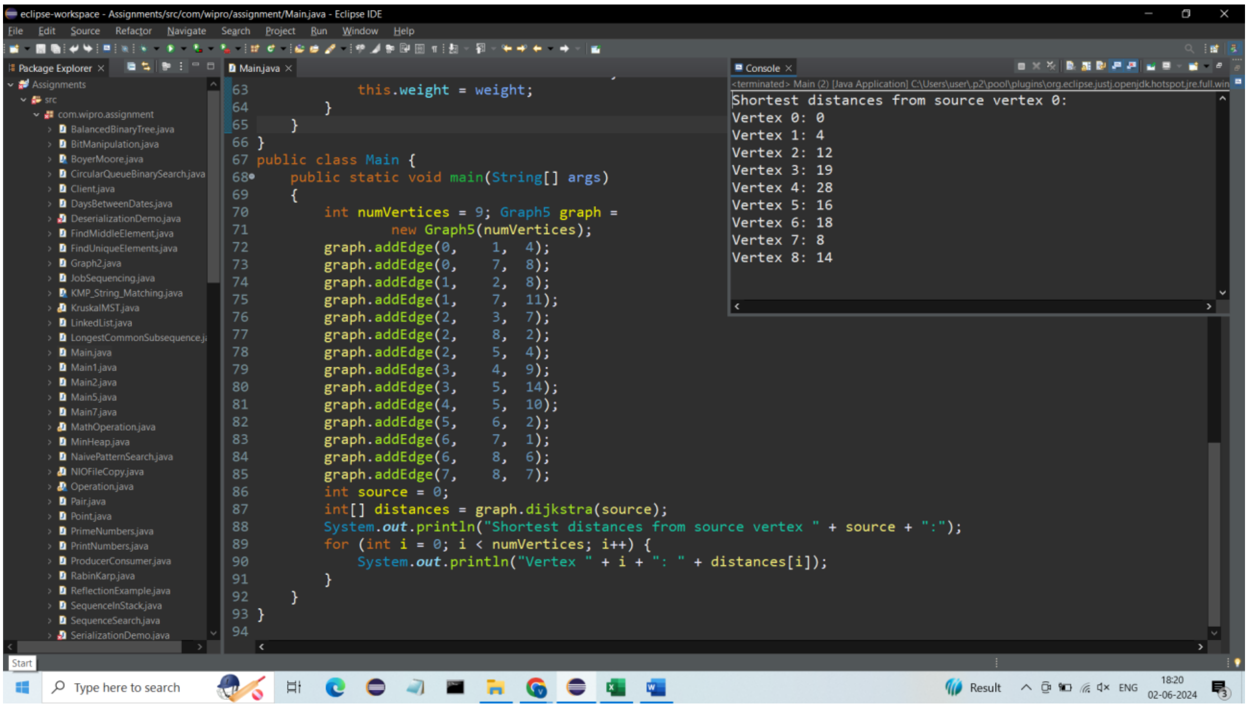
```java
            List<Edge> neighbors =
                    adjacencyMap.get(currentNode);
            for (Edge neighbor : neighbors) {
                int neighborVertex = neighbor.destination;
                int newDistance = distances[currentNode] + neighbor.weight;
                if
                (!visited[neighborVertex] && newDistance < distances[neighborVertex]) {
                    distances[neighborVertex] = newDistance;
                    priorityQueue.add(new Node(neighborVertex, newDistance));
                }
            }

        }
        return distances;
    }
    private static class Node {
        int vertex;
        int weight;
        public Node(int vertex, int
                weight) {
            this.vertex =   vertex;
            this.weight =   weight;
        }
    }
    private static class Edge {
        int destination;
        int weight;
        public Edge(int destination,
                int weight) {
            this.destination = destination;
            this.weight = weight;
        }
```



```java
      this.weight = weight;
    }
  }
}
public class Main {
  public static void main(String[] args)
  {
    int numVertices = 9; Graph5 graph =
        new Graph5(numVertices);
    graph.addEdge(0,    1,  4);
    graph.addEdge(0,    7,  8);
    graph.addEdge(1,    2,  8);
    graph.addEdge(1,    7,  11);
    graph.addEdge(2,    3,  7);
    graph.addEdge(2,    8,  2);
    graph.addEdge(2,    5,  4);
    graph.addEdge(3,    4,  9);
    graph.addEdge(3,    5,  14);
    graph.addEdge(4,    5,  10);
    graph.addEdge(5,    6,  2);
    graph.addEdge(6,    7,  1);
    graph.addEdge(6,    8,  6);
    graph.addEdge(7,    8,  7);
    int source = 0;
    int[] distances = graph.dijkstra(source);
    System.out.println("Shortest distances from source vertex " + source + ":");
    for (int i = 0; i < numVertices; i++) {
      System.out.println("Vertex " + i + ": " + distances[i]);
    }
  }
}
```

Output: -



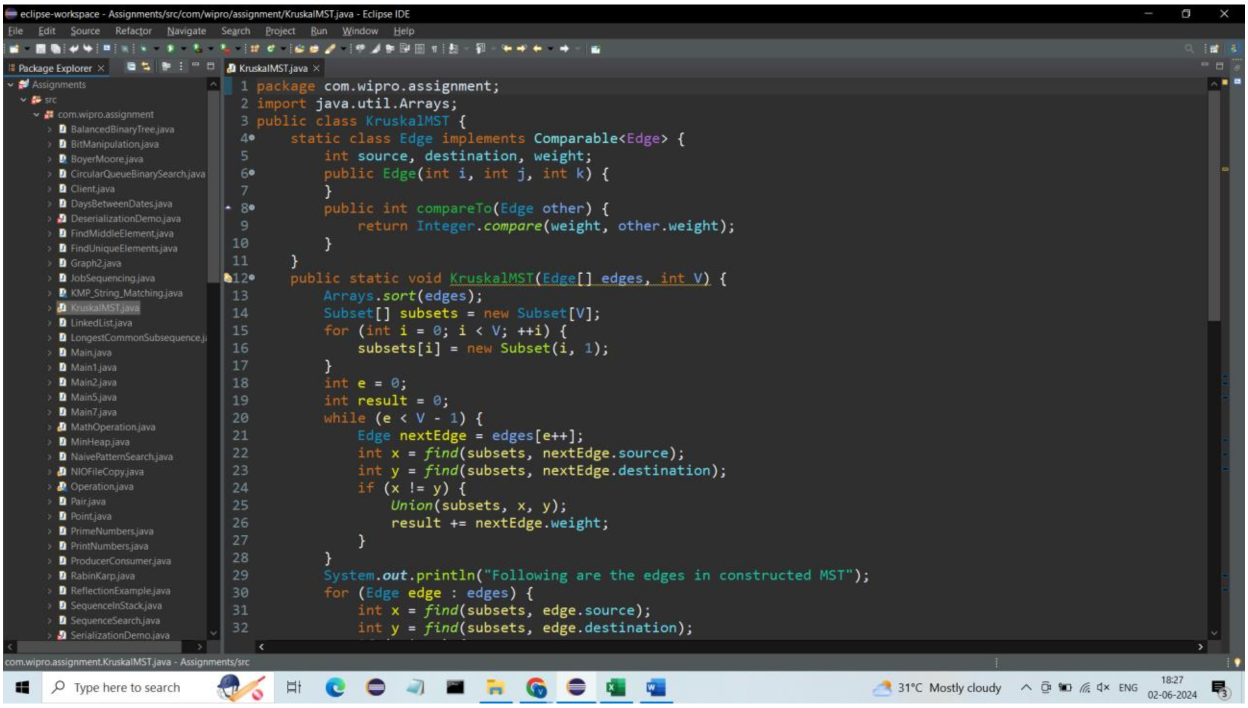
```java
      this.weight = weight;
    }
  }
}
public class Main {
  public static void main(String[] args)
  {
    int numVertices = 9; Graph5 graph =
        new Graph5(numVertices);
    graph.addEdge(0,    1,  4);
    graph.addEdge(0,    7,  8);
    graph.addEdge(1,    2,  8);
    graph.addEdge(1,    7,  11);
    graph.addEdge(2,    3,  7);
    graph.addEdge(2,    8,  2);
    graph.addEdge(2,    5,  4);
    graph.addEdge(3,    4,  9);
    graph.addEdge(3,    5,  14);
    graph.addEdge(4,    5,  10);
    graph.addEdge(5,    6,  2);
    graph.addEdge(6,    7,  1);
    graph.addEdge(6,    8,  6);
    graph.addEdge(7,    8,  7);
    int source = 0;
    int[] distances = graph.dijkstra(source);
    System.out.println("Shortest distances from source vertex " + source + ":");
    for (int i = 0; i < numVertices; i++) {
      System.out.println("Vertex " + i + ": " + distances[i]);
    }
  }
}
```

```
<terminated> Main (2) [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win
Shortest distances from source vertex 0:
Vertex 0: 0
Vertex 1: 4
Vertex 2: 12
Vertex 3: 19
Vertex 4: 28
Vertex 5: 16
Vertex 6: 18
Vertex 7: 8
Vertex 8: 14
```

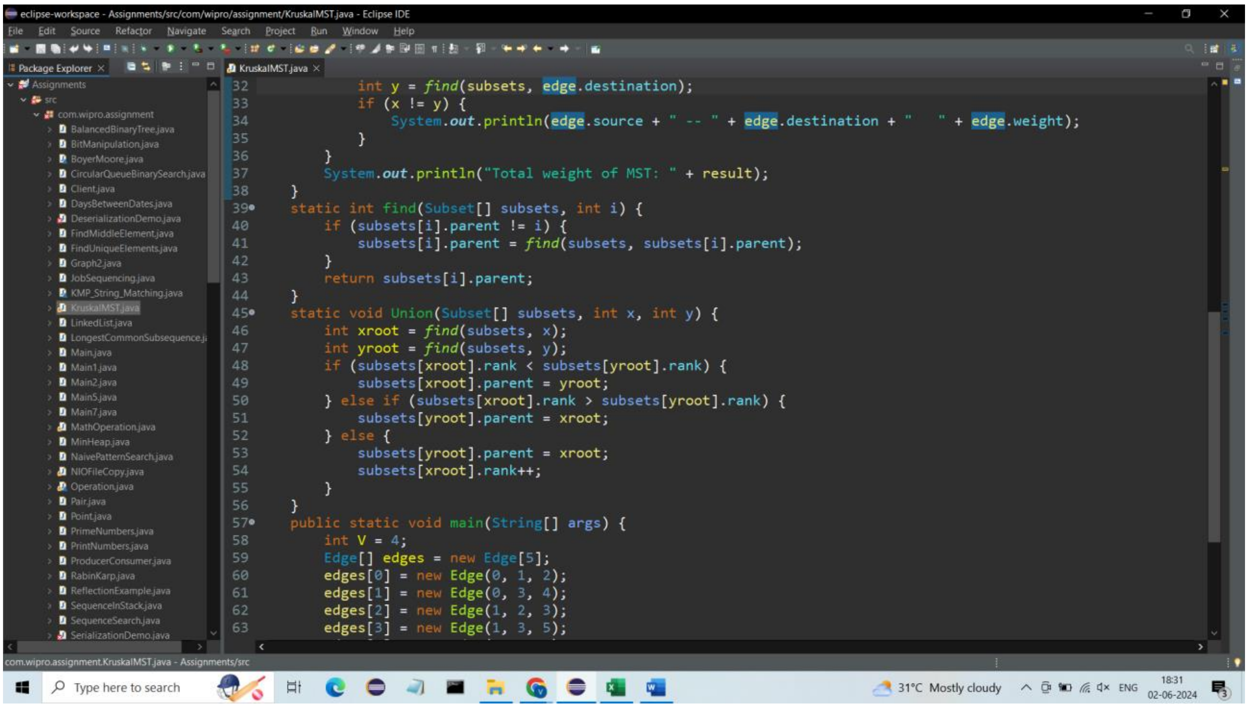
Task 2: Kruskal’s Algorithm for MST

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

Code: -

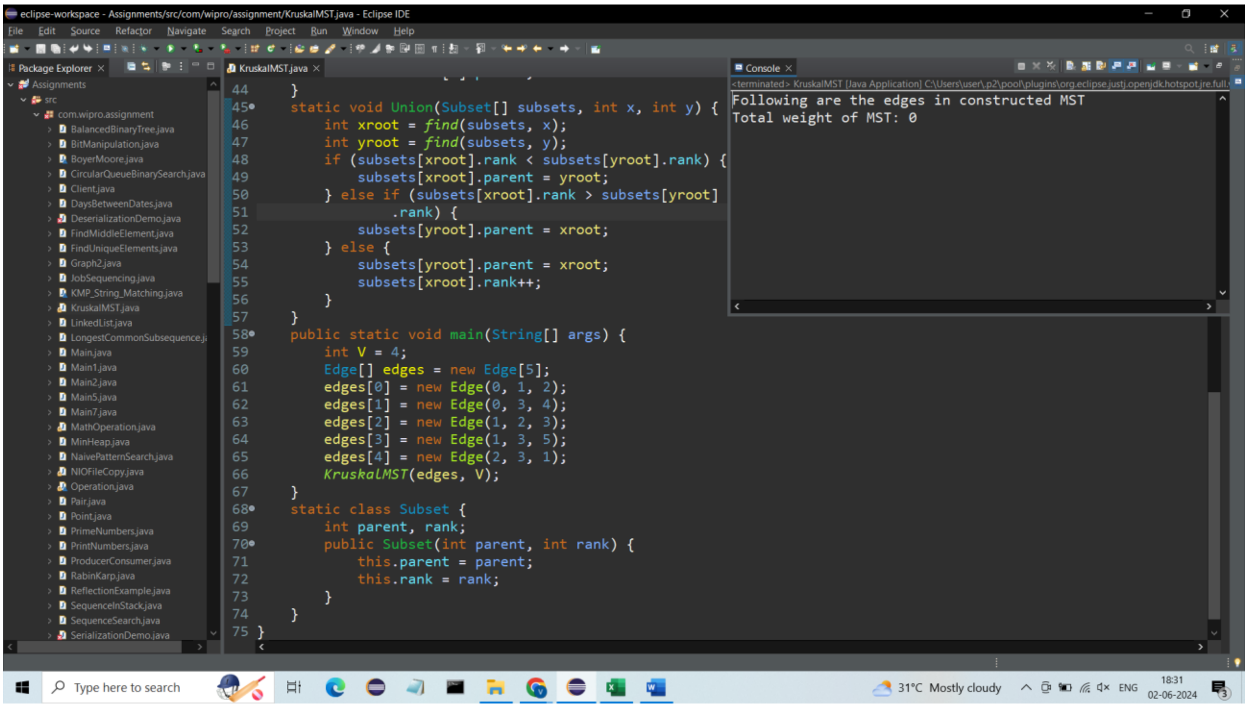


```java
package com.wipro.assignment;
import java.util.Arrays;
public class KruskalMST {
    static class Edge implements Comparable<Edge> {
        int source, destination, weight;
        public Edge(int i, int j, int k) {
        }
        public int compareTo(Edge other) {
            return Integer.compare(weight, other.weight);
        }
    }
    public static void KruskalMST(Edge[] edges, int V) {
        Arrays.sort(edges);
        Subset[] subsets = new Subset[V];
        for (int i = 0; i < V; ++i) {
            subsets[i] = new Subset(i, 1);
        }
        int e = 0;
        int result = 0;
        while (e < V - 1) {
            Edge nextEdge = edges[e++];
            int x = find(subsets, nextEdge.source);
            int y = find(subsets, nextEdge.destination);
            if (x != y) {
                Union(subsets, x, y);
                result += nextEdge.weight;
            }
        }
        System.out.println("Following are the edges in constructed MST");
        for (Edge edge : edges) {
            int x = find(subsets, edge.source);
            int y = find(subsets, edge.destination);
```



```java
            int y = find(subsets, edge.destination);
            if (x != y) {
                System.out.println(edge.source + " -- " + edge.destination + "    " + edge.weight);
            }
        }
        System.out.println("Total weight of MST: " + result);
    }
    static int find(Subset[] subsets, int i) {
        if (subsets[i].parent != i) {
            subsets[i].parent = find(subsets, subsets[i].parent);
        }
        return subsets[i].parent;
    }
    static void Union(Subset[] subsets, int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);
        if (subsets[xroot].rank < subsets[yroot].rank) {
            subsets[xroot].parent = yroot;
        } else if (subsets[xroot].rank > subsets[yroot].rank) {
            subsets[yroot].parent = xroot;
        } else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }
    public static void main(String[] args) {
        int V = 4;
        Edge[] edges = new Edge[5];
        edges[0] = new Edge(0, 1, 2);
        edges[1] = new Edge(0, 3, 4);
        edges[2] = new Edge(1, 2, 3);
        edges[3] = new Edge(1, 3, 5);
```

Output: -



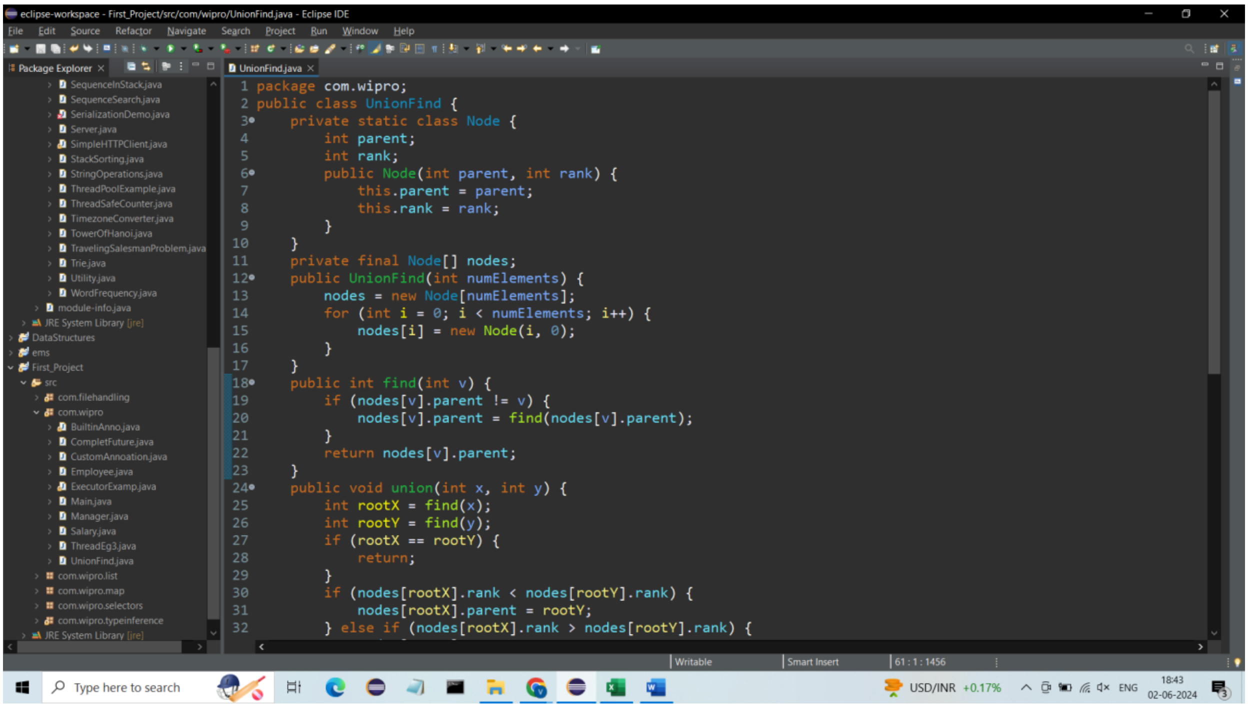
```java
    }
    static void Union(Subset[] subsets, int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);
        if (subsets[xroot].rank < subsets[yroot].rank) {
            subsets[xroot].parent = yroot;
        } else if (subsets[xroot].rank > subsets[yroot]
                .rank) {
            subsets[yroot].parent = xroot;
        } else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }
    public static void main(String[] args) {
        int V = 4;
        Edge[] edges = new Edge[5];
        edges[0] = new Edge(0, 1, 2);
        edges[1] = new Edge(0, 3, 4);
        edges[2] = new Edge(1, 2, 3);
        edges[3] = new Edge(1, 3, 5);
        edges[4] = new Edge(2, 3, 1);
        KruskalMST(edges, V);
    }
    static class Subset {
        int parent, rank;
        public Subset(int parent, int rank) {
            this.parent = parent;
            this.rank = rank;
        }
    }
}
```

```
<terminated> KruskalMST [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
Following are the edges in constructed MST
Total weight of MST: 0
```

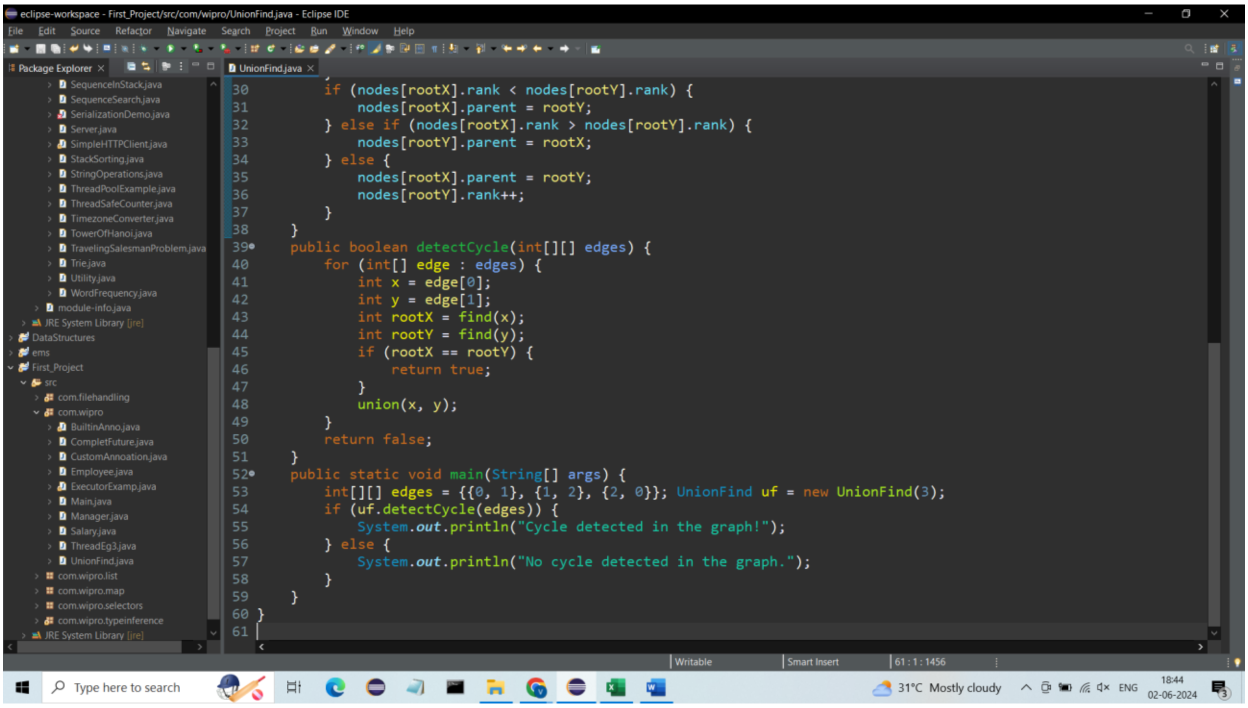
Task 3: Union-Find for Cycle Detection

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

Code: -

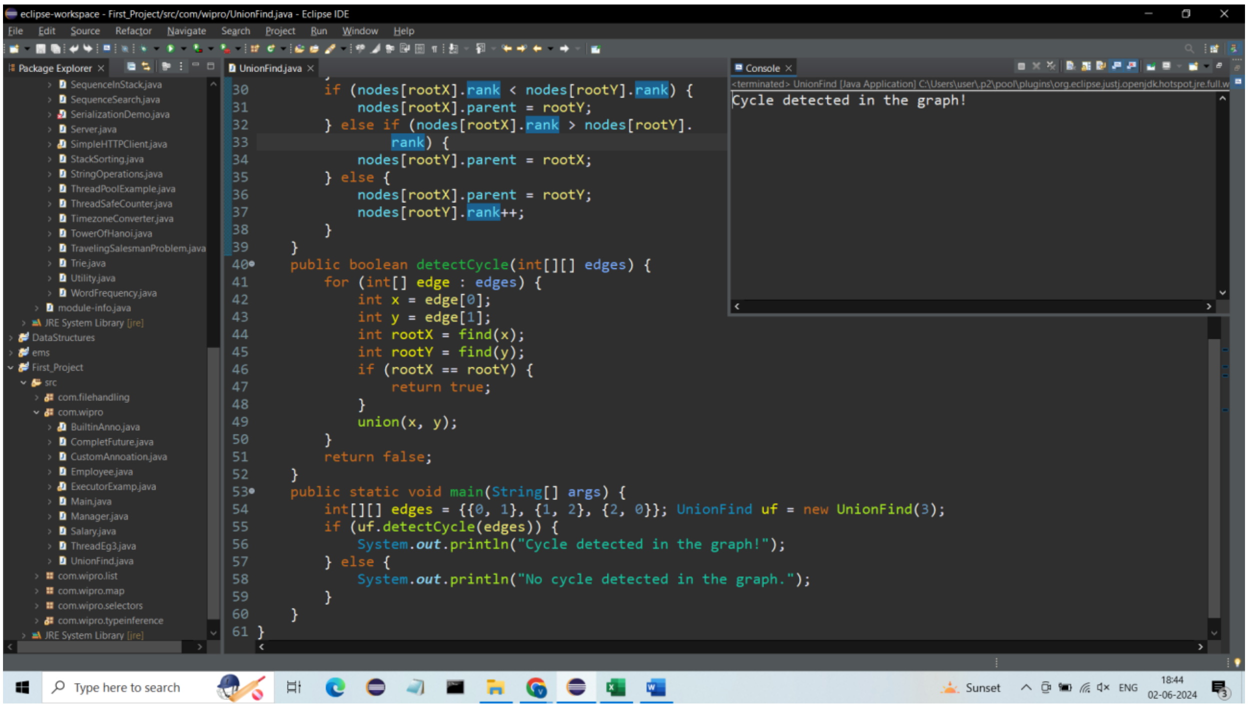

eclipse-workspace - First_Project/src/com/wipro/UnionFind.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
UnionFind.java
Writable
Smart Insert
61 : 1 : 1456
Type here to search
USD/INR +0.17%
ENG
18:43
02-06-2024

```java
package com.wipro;
public class UnionFind {
    private static class Node {
        int parent;
        int rank;
        public Node(int parent, int rank) {
            this.parent = parent;
            this.rank = rank;
        }
    }
    private final Node[] nodes;
    public UnionFind(int numElements) {
        nodes = new Node[numElements];
        for (int i = 0; i < numElements; i++) {
            nodes[i] = new Node(i, 0);
        }
    }
    public int find(int v) {
        if (nodes[v].parent != v) {
            nodes[v].parent = find(nodes[v].parent);
        }
        return nodes[v].parent;
    }
    public void union(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX == rootY) {
            return;
        }
        if (nodes[rootX].rank < nodes[rootY].rank) {
            nodes[rootX].parent = rootY;
        } else if (nodes[rootX].rank > nodes[rootY].rank) {
```


eclipse-workspace - First_Project/src/com/wipro/UnionFind.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
UnionFind.java
Writable
Smart Insert
61 : 1 : 1456
Type here to search
31°C Mostly cloudy
ENG
18:44
02-06-2024

```java
        if (nodes[rootX].rank < nodes[rootY].rank) {
            nodes[rootX].parent = rootY;
        } else if (nodes[rootX].rank > nodes[rootY].rank) {
            nodes[rootY].parent = rootX;
        } else {
            nodes[rootX].parent = rootY;
            nodes[rootY].rank++;
        }
    }
    public boolean detectCycle(int[][] edges) {
        for (int[] edge : edges) {
            int x = edge[0];
            int y = edge[1];
            int rootX = find(x);
            int rootY = find(y);
            if (rootX == rootY) {
                return true;
            }
            union(x, y);
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] edges = {{0, 1}, {1, 2}, {2, 0}}; UnionFind uf = new UnionFind(3);
        if (uf.detectCycle(edges)) {
            System.out.println("Cycle detected in the graph!");
        } else {
            System.out.println("No cycle detected in the graph.");
        }
    }
}
```

Output: -



```java
        if (nodes[rootX].rank < nodes[rootY].rank) {
            nodes[rootX].parent = rootY;
        } else if (nodes[rootX].rank > nodes[rootY].
                rank) {
            nodes[rootY].parent = rootX;
        } else {
            nodes[rootX].parent = rootY;
            nodes[rootY].rank++;
        }
    }
    public boolean detectCycle(int[][] edges) {
        for (int[] edge : edges) {
            int x = edge[0];
            int y = edge[1];
            int rootX = find(x);
            int rootY = find(y);
            if (rootX == rootY) {
                return true;
            }
            union(x, y);
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] edges = {{0, 1}, {1, 2}, {2, 0}}; UnionFind uf = new UnionFind(3);
        if (uf.detectCycle(edges)) {
            System.out.println("Cycle detected in the graph!");
        } else {
            System.out.println("No cycle detected in the graph.");
        }
    }
}
```

```
<terminated> UnionFind [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.w
Cycle detected in the graph!
```