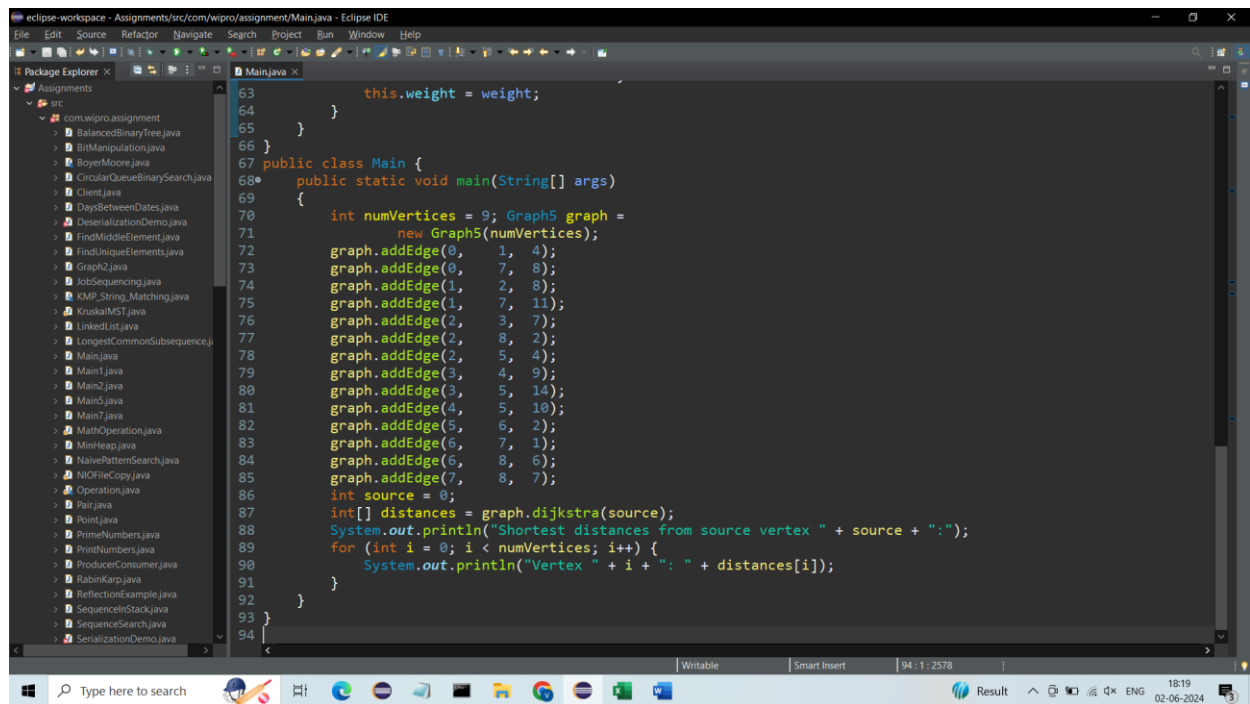


## Task 1: Dijkstra's Shortest Path Finder

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

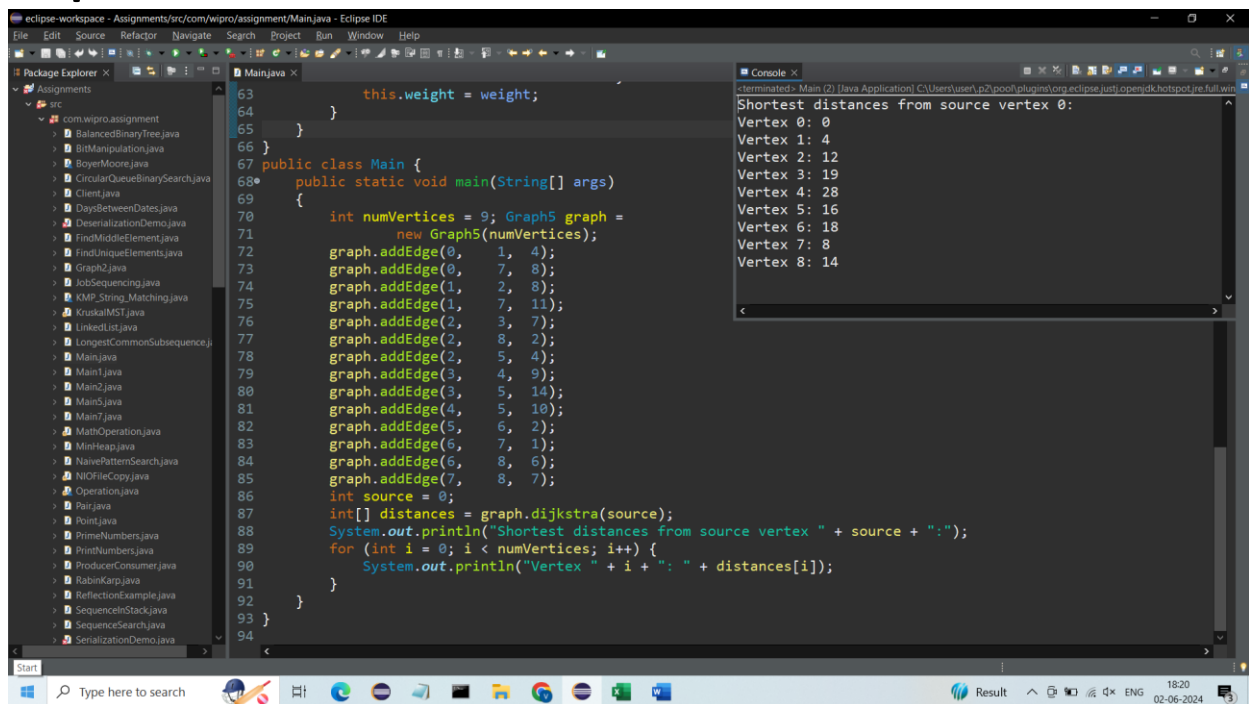
Code: -

```
1 package com.wipro.assignment;
2 import java.util.*;
3 class Graph5 {
4     private int numVertices;
5     private Map<Integer, List<Edge>> adjacencyMap;
6     public Graph5(int numVertices) {
7         this.numVertices = numVertices;
8         adjacencyMap = new
9             HashMap<>();
10        for (int i = 0; i < numVertices; i++) {
11            adjacencyMap.put(i, new
12                ArrayList<>());
13        }
14    }
15    public void addEdge(int source,
16        int destination, int weight) {
17        adjacencyMap.get(source).add(new Edge(destination, weight));
18    }
19    public int[] dijkstra(int source)
20    {
21        PriorityQueue<Node>
22        priorityQueue = new PriorityQueue<>(numVertices, Comparator.comparingInt(n -> n.weight));
23        int[] distances = new int[numVertices];
24        Arrays.fill(distances, Integer.MAX_VALUE);
25        boolean[] visited = new boolean[numVertices];
26        priorityQueue.add(new Node(source, 0));
27        distances[source] = 0;
28        while
29            (!priorityQueue.isEmpty()) {
30                int currentNode = priorityQueue.poll().vertex;
31                visited[currentNode] =
32                    true;
33                List<Edge> neighbors =
34                    adjacencyMap.get(currentNode);
35                for (Edge neighbor : neighbors) {
36                    int neighborVertex = neighbor.destination;
37                    int newDistance = distances[currentNode] + neighbor.weight;
38                    if
39                        (!visited[neighborVertex] && newDistance < distances[neighborVertex]) {
40                        distances[neighborVertex] = newDistance;
41                        priorityQueue.add(new Node(neighborVertex, newDistance));
42                    }
43                }
44            }
45        return distances;
46    }
47    private static class Node {
48        int vertex;
49        int weight;
50        public Node(int vertex, int
51            weight) {
52            this.vertex = vertex;
53            this.weight = weight;
54        }
55    }
56    private static class Edge {
57        int destination;
58        int weight;
59        public Edge(int destination,
60            int weight) {
61            this.destination = destination;
62            this.weight = weight;
63        }
64    }
```



```
63         this.weight = weight;
64     }
65 }
66 }
67 public class Main {
68     public static void main(String[] args)
69     {
70         int numVertices = 9; Graph5 graph =
71             new Graph5(numVertices);
72         graph.addEdge(0, 1, 4);
73         graph.addEdge(0, 7, 8);
74         graph.addEdge(1, 2, 8);
75         graph.addEdge(1, 7, 11);
76         graph.addEdge(2, 3, 7);
77         graph.addEdge(2, 8, 2);
78         graph.addEdge(2, 5, 4);
79         graph.addEdge(3, 4, 9);
80         graph.addEdge(3, 5, 14);
81         graph.addEdge(4, 5, 10);
82         graph.addEdge(5, 6, 2);
83         graph.addEdge(6, 7, 1);
84         graph.addEdge(6, 8, 6);
85         graph.addEdge(7, 8, 7);
86         int source = 0;
87         int[] distances = graph.dijkstra(source);
88         System.out.println("Shortest distances from source vertex " + source + ":");
89         for (int i = 0; i < numVertices; i++) {
90             System.out.println("Vertex " + i + ": " + distances[i]);
91         }
92     }
93 }
94 }
```

Output: -

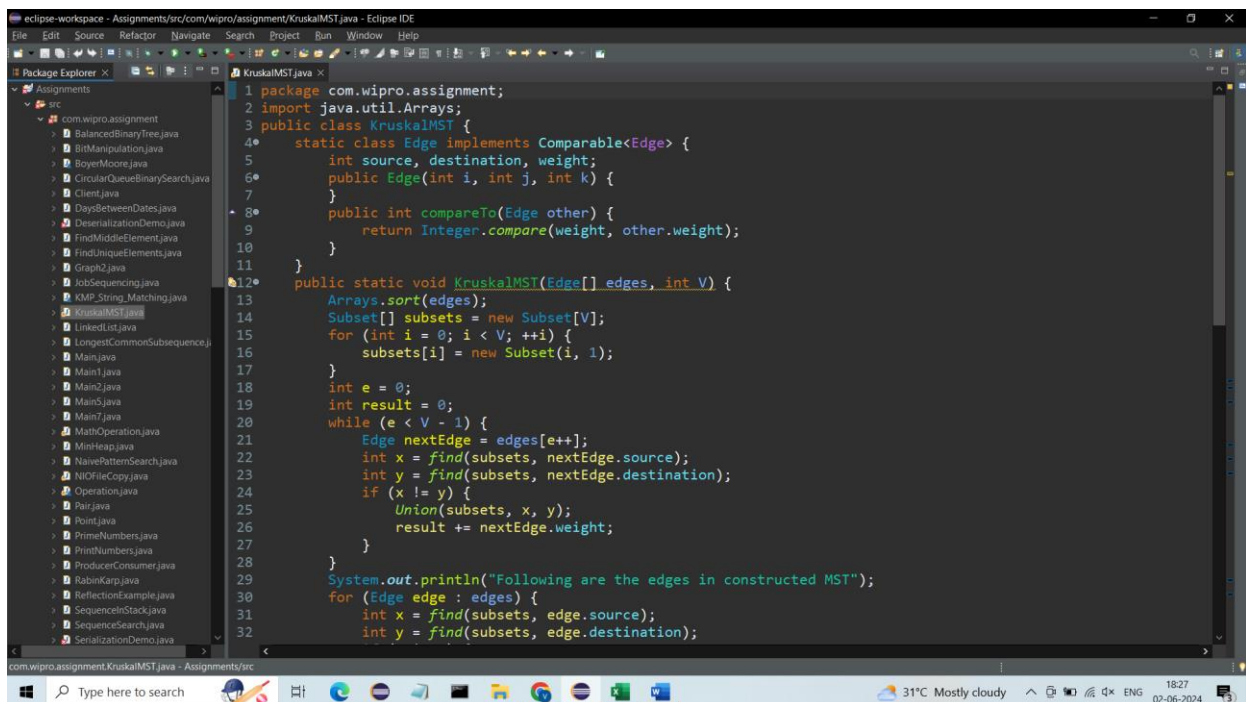


```
<terminated> Main (2) [Java Application] C:\Users\User\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win
Shortest distances from source vertex 0:
Vertex 0: 0
Vertex 1: 4
Vertex 2: 12
Vertex 3: 19
Vertex 4: 28
Vertex 5: 16
Vertex 6: 18
Vertex 7: 8
Vertex 8: 14
```

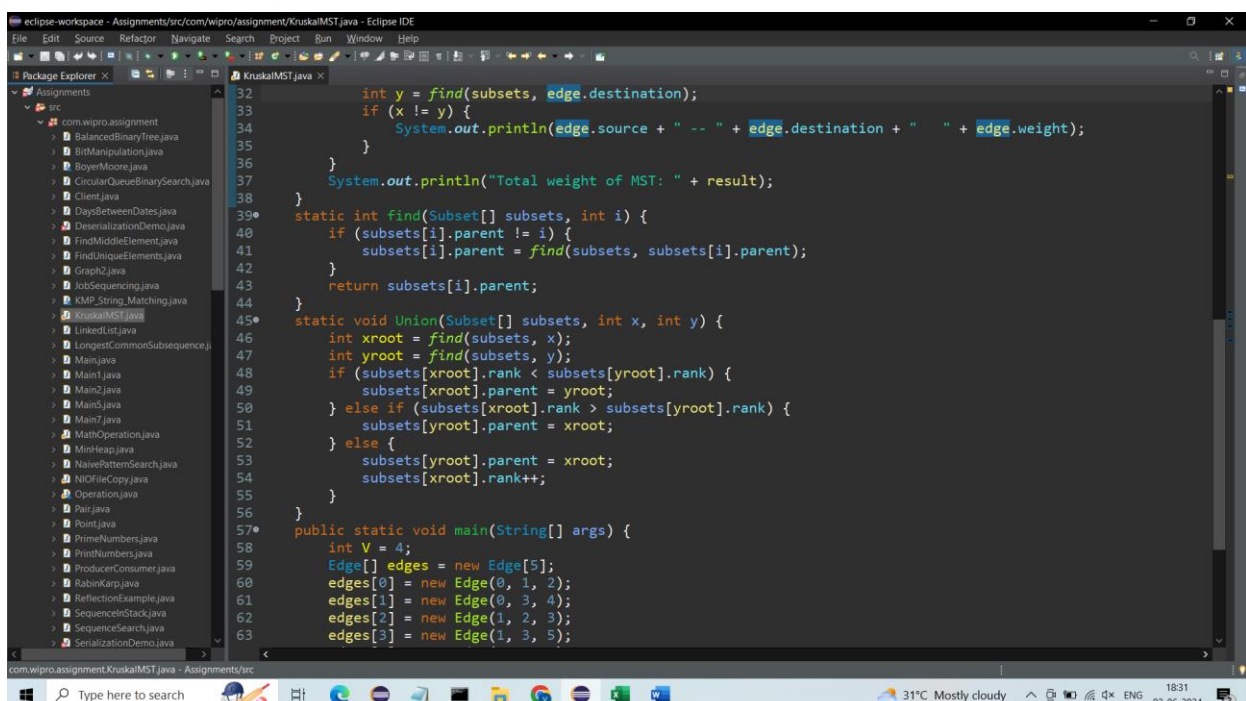
## Task 2: Kruskal's Algorithm for MST

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

Code: -

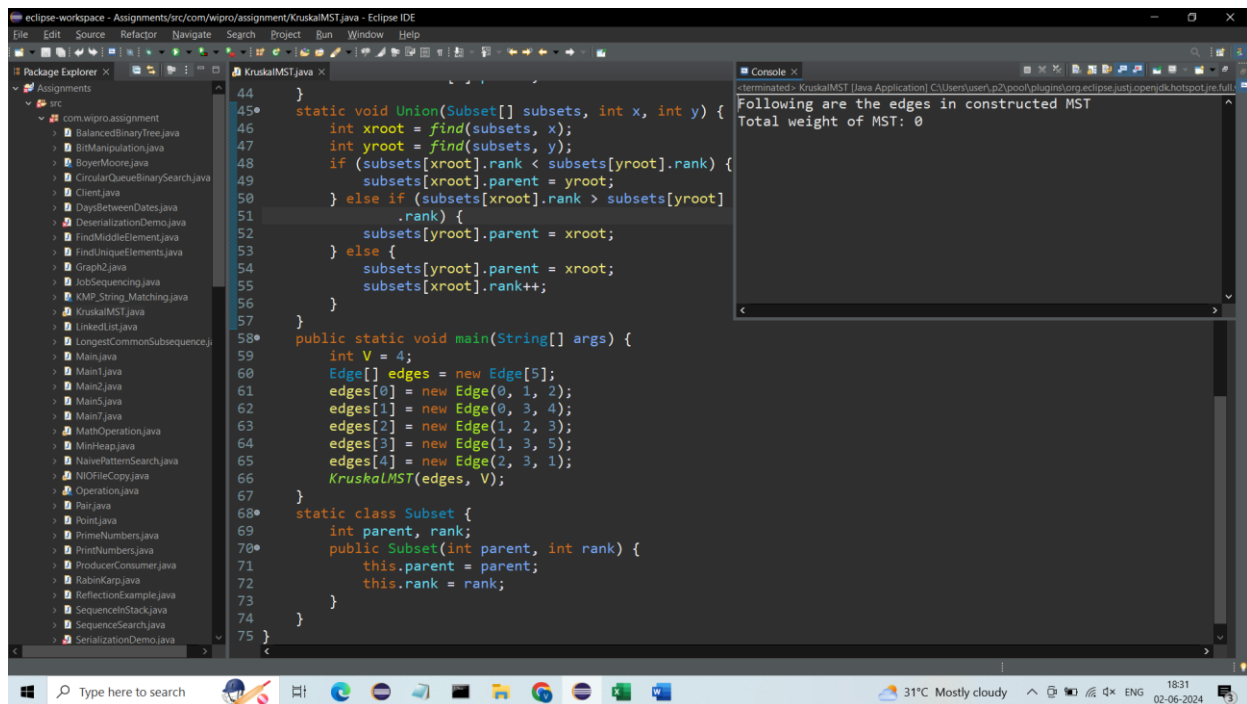


```
1 package com.wipro.assignment;
2 import java.util.Arrays;
3 public class KruskalMST {
4     static class Edge implements Comparable<Edge> {
5         int source, destination, weight;
6         public Edge(int i, int j, int k) {
7             source = i; destination = j; weight = k;
8         }
9         public int compareTo(Edge other) {
10             return Integer.compare(weight, other.weight);
11         }
12     }
13     public static void KruskalMST(Edge[] edges, int V) {
14         Arrays.sort(edges);
15         SubSet[] subsets = new SubSet[V];
16         for (int i = 0; i < V; ++i) {
17             subsets[i] = new SubSet(i, 1);
18         }
19         int e = 0;
20         int result = 0;
21         while (e < V - 1) {
22             Edge nextEdge = edges[e++];
23             int x = find(subsets, nextEdge.source);
24             int y = find(subsets, nextEdge.destination);
25             if (x != y) {
26                 Union(subsets, x, y);
27                 result += nextEdge.weight;
28             }
29         }
30         System.out.println("Following are the edges in constructed MST");
31         for (Edge edge : edges) {
32             int x = find(subsets, edge.source);
33             int y = find(subsets, edge.destination);
```



```
34             int y = find(subsets, edge.destination);
35             if (x != y) {
36                 System.out.println(edge.source + " -- " + edge.destination + " " + edge.weight);
37             }
38         }
39         System.out.println("Total weight of MST: " + result);
40     }
41     static int find(SubSet[] subsets, int i) {
42         if (subsets[i].parent != i) {
43             subsets[i].parent = find(subsets, subsets[i].parent);
44         }
45         return subsets[i].parent;
46     }
47     static void Union(SubSet[] subsets, int x, int y) {
48         int xroot = find(subsets, x);
49         int yroot = find(subsets, y);
50         if (subsets[xroot].rank < subsets[yroot].rank) {
51             subsets[xroot].parent = yroot;
52         } else if (subsets[xroot].rank > subsets[yroot].rank) {
53             subsets[yroot].parent = xroot;
54         } else {
55             subsets[yroot].parent = xroot;
56             subsets[xroot].rank++;
57         }
58     }
59     public static void main(String[] args) {
60         int V = 4;
61         Edge[] edges = new Edge[5];
62         edges[0] = new Edge(0, 1, 2);
63         edges[1] = new Edge(0, 3, 4);
64         edges[2] = new Edge(1, 2, 3);
65         edges[3] = new Edge(1, 3, 5);
```

Output: -



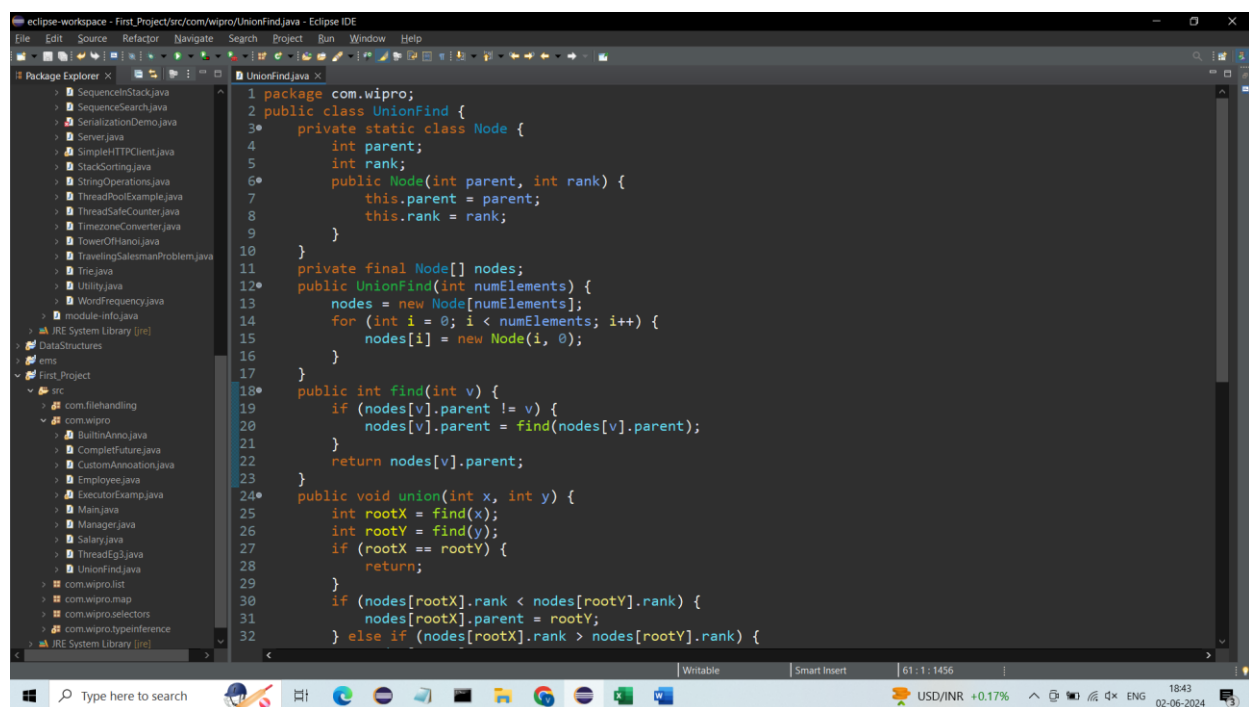
```
44 }
45 static void Union(Subset[] subsets, int x, int y) {
46     int xroot = find(subsets, x);
47     int yroot = find(subsets, y);
48     if (subsets[xroot].rank < subsets[yroot].rank) {
49         subsets[xroot].parent = yroot;
50     } else if (subsets[xroot].rank > subsets[yroot].rank) {
51         subsets[yroot].parent = xroot;
52     } else {
53         subsets[yroot].parent = xroot;
54         subsets[xroot].rank++;
55     }
56 }
57
58 public static void main(String[] args) {
59     int V = 4;
60     Edge[] edges = new Edge[5];
61     edges[0] = new Edge(0, 1, 2);
62     edges[1] = new Edge(0, 3, 4);
63     edges[2] = new Edge(1, 2, 3);
64     edges[3] = new Edge(1, 3, 5);
65     edges[4] = new Edge(2, 3, 1);
66     KruskalMST(edges, V);
67 }
68 static class Subset {
69     int parent, rank;
70     public Subset(int parent, int rank) {
71         this.parent = parent;
72         this.rank = rank;
73     }
74 }
75 }
```

Following are the edges in constructed MST  
Total weight of MST: 0

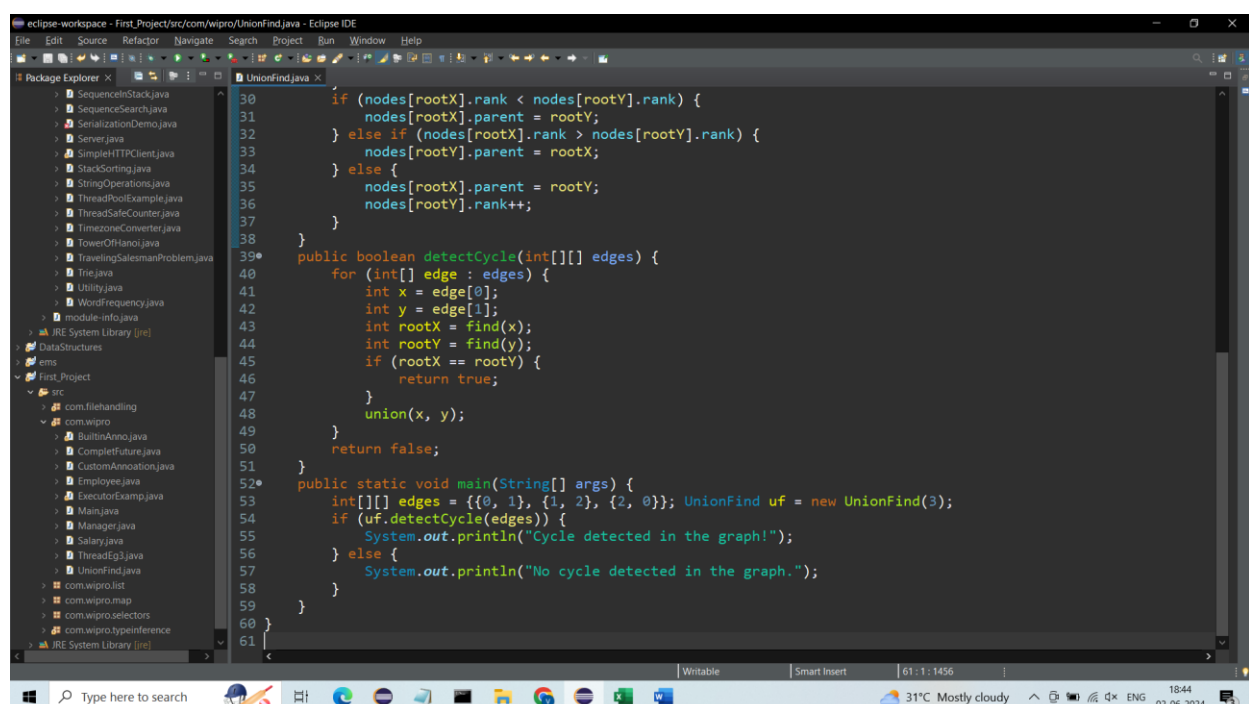
## Task 3: Union-Find for Cycle Detection

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

Code: -

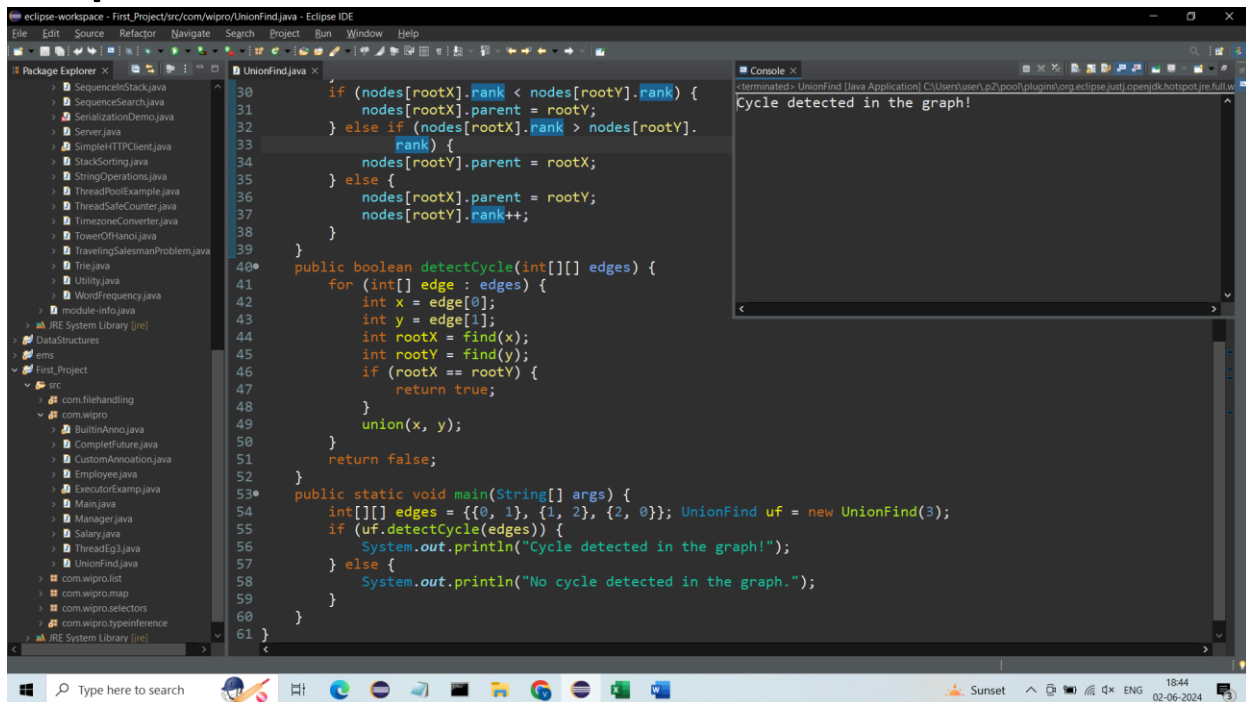


```
1 package com.wipro;
2 public class UnionFind {
3     private static class Node {
4         int parent;
5         int rank;
6         public Node(int parent, int rank) {
7             this.parent = parent;
8             this.rank = rank;
9         }
10    }
11    private final Node[] nodes;
12    public UnionFind(int numElements) {
13        nodes = new Node[numElements];
14        for (int i = 0; i < numElements; i++) {
15            nodes[i] = new Node(i, 0);
16        }
17    }
18    public int find(int v) {
19        if (nodes[v].parent != v) {
20            nodes[v].parent = find(nodes[v].parent);
21        }
22        return nodes[v].parent;
23    }
24    public void union(int x, int y) {
25        int rootX = find(x);
26        int rootY = find(y);
27        if (rootX == rootY) {
28            return;
29        }
30        if (nodes[rootX].rank < nodes[rootY].rank) {
31            nodes[rootX].parent = rootY;
32        } else if (nodes[rootX].rank > nodes[rootY].rank) {
```



```
30        if (nodes[rootX].rank < nodes[rootY].rank) {
31            nodes[rootX].parent = rootY;
32        } else if (nodes[rootX].rank > nodes[rootY].rank) {
33            nodes[rootY].parent = rootX;
34        } else {
35            nodes[rootX].parent = rootY;
36            nodes[rootY].rank++;
37        }
38    }
39    public boolean detectCycle(int[][] edges) {
40        for (int[] edge : edges) {
41            int x = edge[0];
42            int y = edge[1];
43            int rootX = find(x);
44            int rootY = find(y);
45            if (rootX == rootY) {
46                return true;
47            }
48            union(x, y);
49        }
50        return false;
51    }
52    public static void main(String[] args) {
53        int[][] edges = {{0, 1}, {1, 2}, {2, 0}};
54        UnionFind uf = new UnionFind(3);
55        if (uf.detectCycle(edges)) {
56            System.out.println("Cycle detected in the graph!");
57        } else {
58            System.out.println("No cycle detected in the graph.");
59        }
60    }
61 }
```

## Output: -



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'First\_Project' with a source folder 'src' containing several Java files, including 'UnionFind.java'. The main editor window shows the code for 'UnionFind.java'. The code implements a Union-Find data structure with methods for finding roots, unioning nodes, and detecting cycles in a graph. The 'main' method initializes an array of edges and calls 'detectCycle' to check for a cycle. The Console window on the right shows the output of the program, which is 'Cycle detected in the graph!'.

```
30 if (nodes[rootX].rank < nodes[rootY].rank) {
31     nodes[rootX].parent = rootY;
32 } else if (nodes[rootX].rank > nodes[rootY].
33     rank) {
34     nodes[rootY].parent = rootX;
35 } else {
36     nodes[rootX].parent = rootY;
37     nodes[rootY].rank++;
38 }
39 }
40 public boolean detectCycle(int[][] edges) {
41     for (int[] edge : edges) {
42         int x = edge[0];
43         int y = edge[1];
44         int rootX = find(x);
45         int rootY = find(y);
46         if (rootX == rootY) {
47             return true;
48         }
49         union(x, y);
50     }
51     return false;
52 }
53 public static void main(String[] args) {
54     int[][] edges = {{0, 1}, {1, 2}, {2, 0}}; UnionFind uf = new UnionFind(3);
55     if (uf.detectCycle(edges)) {
56         System.out.println("Cycle detected in the graph!");
57     } else {
58         System.out.println("No cycle detected in the graph.");
59     }
60 }
61 }
```

Console Output:

```
terminated> UnionFind [Java Application] C:\Users\user\p2\pool\plugin1.org.eclipse.justi.openjdk.hotspot.jre.full.j
Cycle detected in the graph!
```