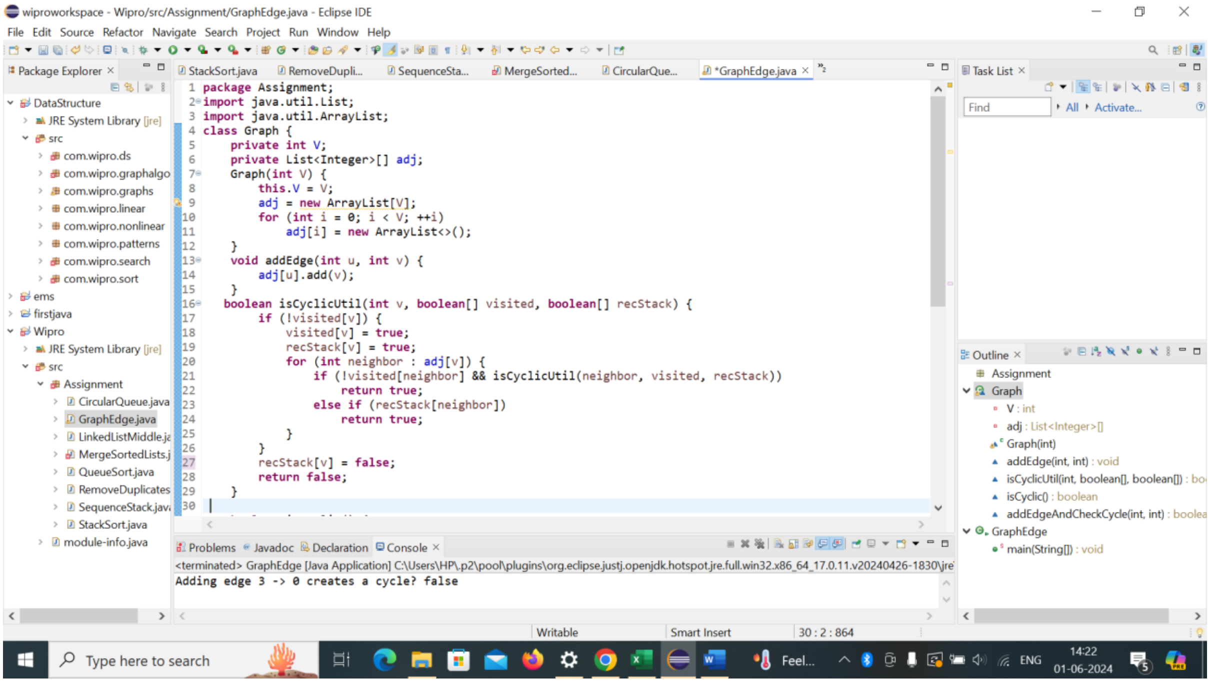
Algorithms

Task 4: Graph Edge Addition Validation

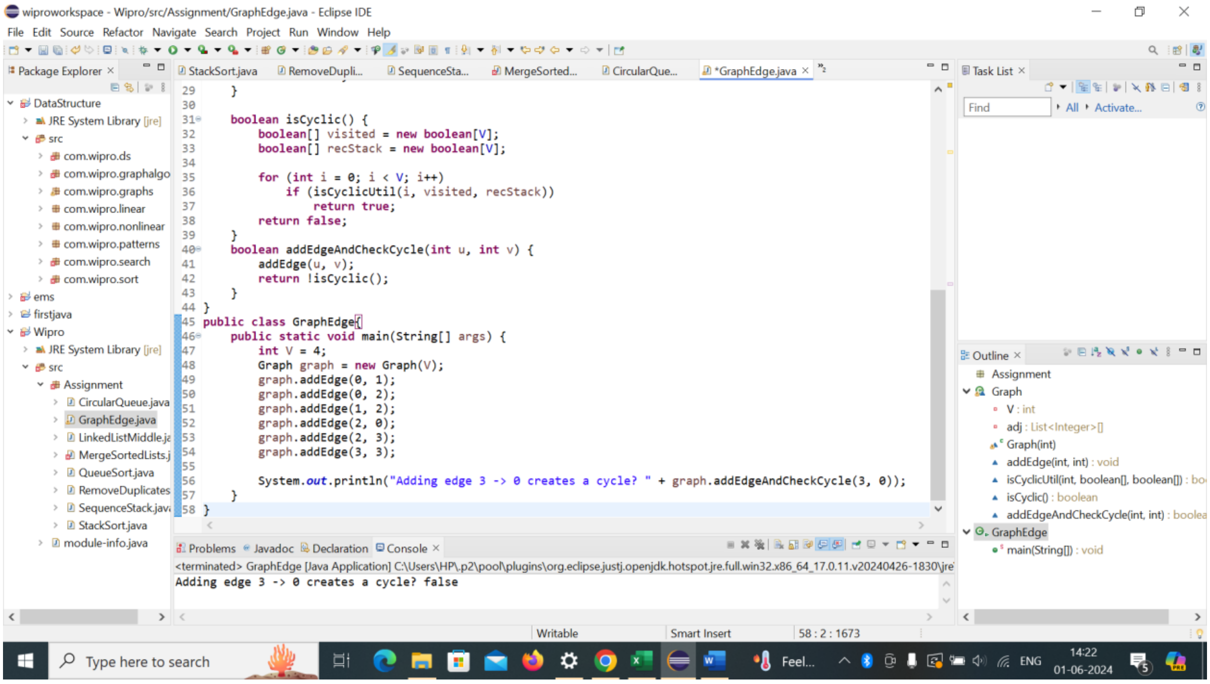
Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added.



wiproworkspace - Wipro/src/Assignment/GraphEdge.java - Eclipse IDE

```java
package Assignment;
import java.util.List;
import java.util.ArrayList;
class Graph {
    private int V;
    private List<Integer>[] adj;
    Graph(int V) {
        this.V = V;
        adj = new ArrayList[V];
        for (int i = 0; i < V; ++i)
            adj[i] = new ArrayList<>();
    }
    void addEdge(int u, int v) {
        adj[u].add(v);
    }
   boolean isCyclicUtil(int v, boolean[] visited, boolean[] recStack) {
        if (!visited[v]) {
            visited[v] = true;
            recStack[v] = true;
            for (int neighbor : adj[v]) {
                if (!visited[neighbor] && isCyclicUtil(neighbor, visited, recStack))
                    return true;
                else if (recStack[neighbor])
                    return true;
            }
        }
        recStack[v] = false;
        return false;
    }

```

```
<terminated> GraphEdge [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre
Adding edge 3 -> 0 creates a cycle? false
```



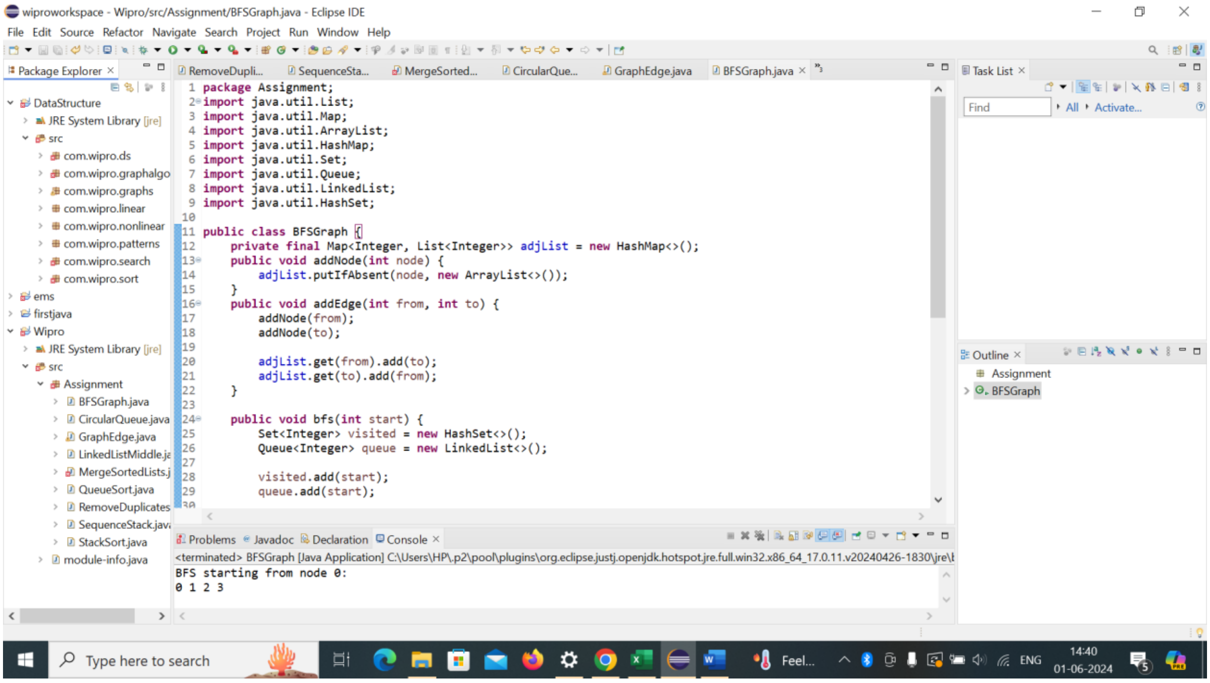
wiproworkspace - Wipro/src/Assignment/GraphEdge.java - Eclipse IDE

```java
    }

    boolean isCyclic() {
        boolean[] visited = new boolean[V];
        boolean[] recStack = new boolean[V];

        for (int i = 0; i < V; i++)
            if (isCyclicUtil(i, visited, recStack))
                return true;
        return false;
    }
    boolean addEdgeAndCheckCycle(int u, int v) {
        addEdge(u, v);
        return !isCyclic();
    }
}
public class GraphEdge{
    public static void main(String[] args) {
        int V = 4;
        Graph graph = new Graph(V);
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 2);
        graph.addEdge(2, 0);
        graph.addEdge(2, 3);
        graph.addEdge(3, 3);

        System.out.println("Adding edge 3 -> 0 creates a cycle? " + graph.addEdgeAndCheckCycle(3, 0));
    }
}
```

```
<terminated> GraphEdge [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre
Adding edge 3 -> 0 creates a cycle? false
```

Task 5: Breadth-First Search (BFS) Implementation

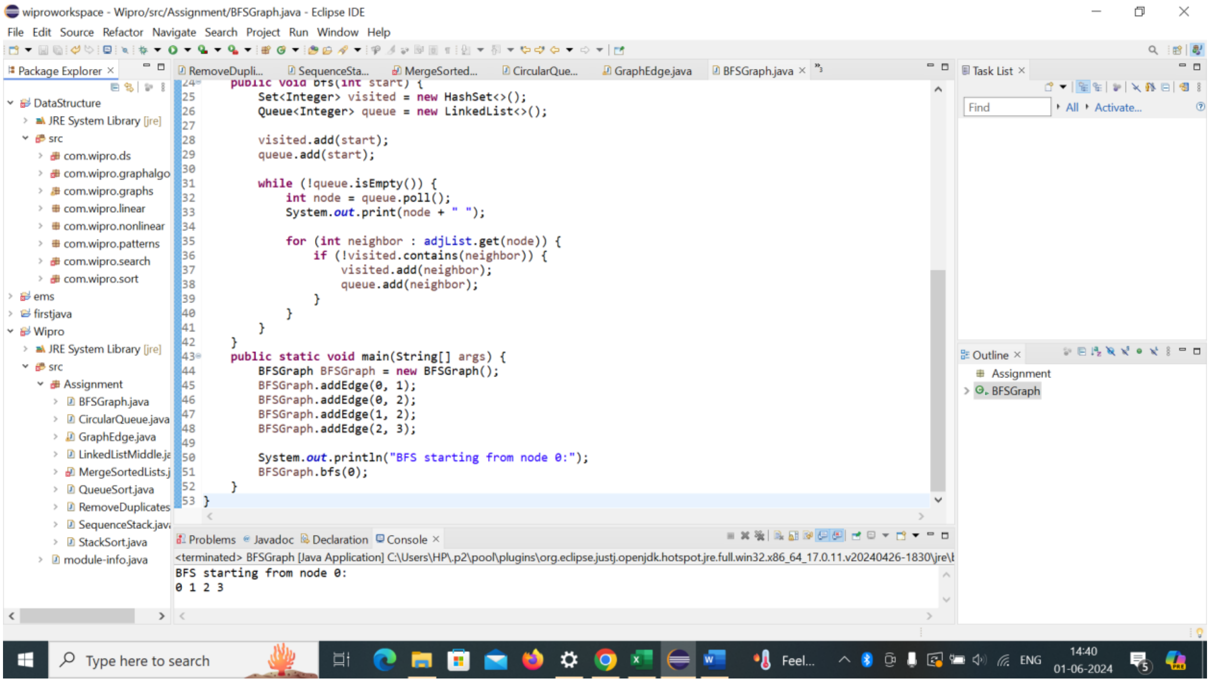
For a given undirected graph, implement BFS to traverse the graph starting from a given node and print each node in the order it is visited.



```java
package Assignment;
import java.util.List;
import java.util.Map;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;
import java.util.Queue;
import java.util.LinkedList;
import java.util.HashSet;

public class BFSGraph {
    private final Map<Integer, List<Integer>> adjList = new HashMap<>();
    public void addNode(int node) {
        adjList.putIfAbsent(node, new ArrayList<>());
    }
    public void addEdge(int from, int to) {
        addNode(from);
        addNode(to);

        adjList.get(from).add(to);
        adjList.get(to).add(from);
    }

    public void bfs(int start) {
        Set<Integer> visited = new HashSet<>();
        Queue<Integer> queue = new LinkedList<>();

        visited.add(start);
        queue.add(start);
```

<terminated> BFSGraph [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\l

```
BFS starting from node 0:
0 1 2 3
```



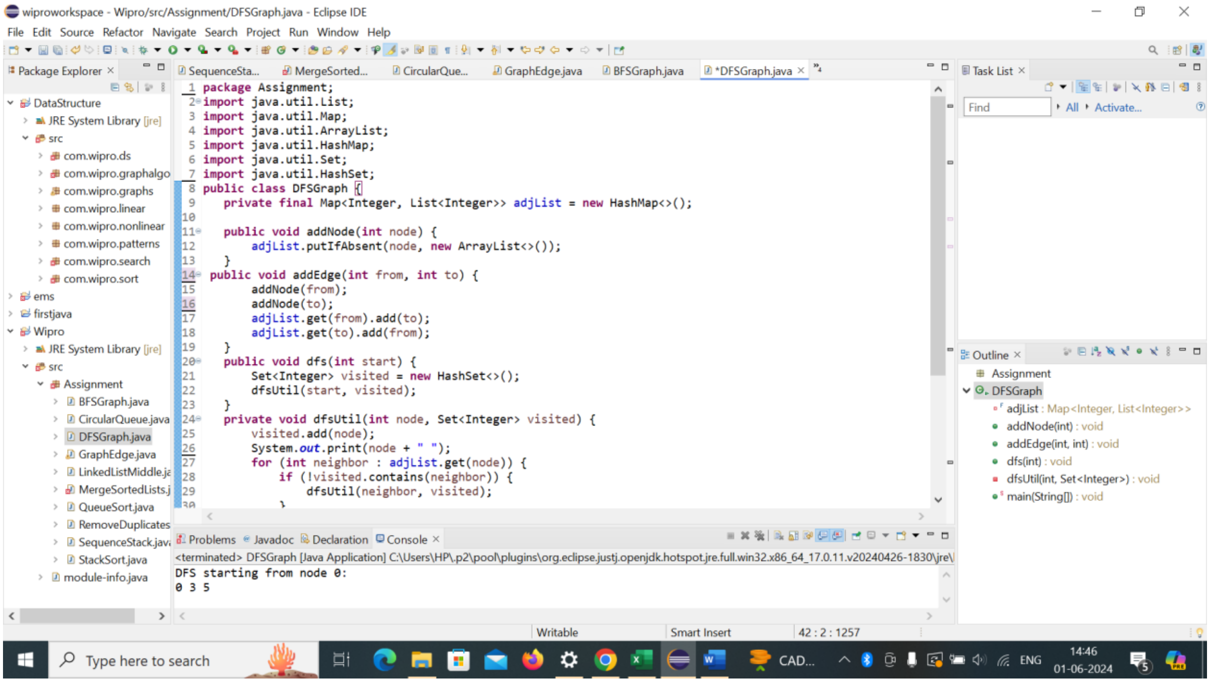
```java
    public void bfs(int start) {
        Set<Integer> visited = new HashSet<>();
        Queue<Integer> queue = new LinkedList<>();

        visited.add(start);
        queue.add(start);

        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");

            for (int neighbor : adjList.get(node)) {
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    queue.add(neighbor);
                }
            }
        }
    }
    public static void main(String[] args) {
        BFSGraph BFSGraph = new BFSGraph();
        BFSGraph.addEdge(0, 1);
        BFSGraph.addEdge(0, 2);
        BFSGraph.addEdge(1, 2);
        BFSGraph.addEdge(2, 3);

        System.out.println("BFS starting from node 0:");
        BFSGraph.bfs(0);
    }
}
```

<terminated> BFSGraph [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\l

```
BFS starting from node 0:
0 1 2 3
```

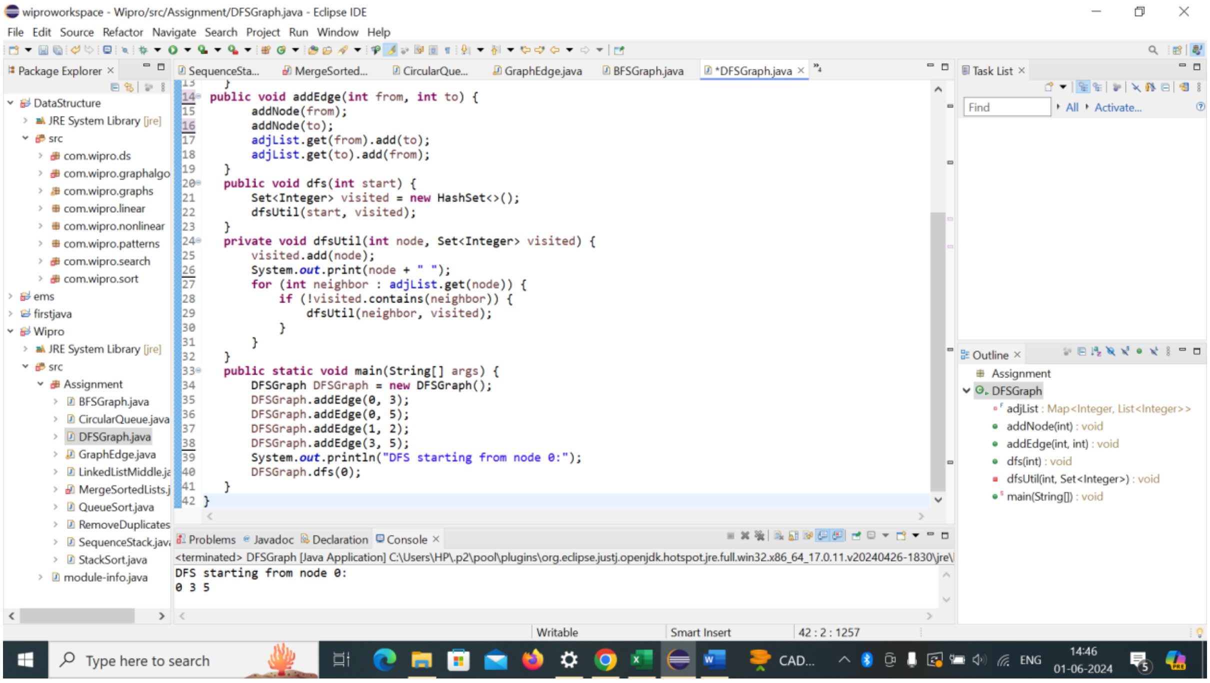
Task 6: Depth-First Search (DFS) Recursive

Write a recursive DFS function for a given undirected graph. The function should visit every node and print it out.


wiproworkspace - Wipro/src/Assignment/DFSGraph.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
SequenceSta... MergeSorted... CircularQue... GraphEdge.java BFSGraph.java *DFSGraph.java
Task List
Outline
Problems Javadoc Declaration Console

```java
package Assignment;
import java.util.List;
import java.util.Map;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;
import java.util.HashSet;
public class DFSGraph {
   private final Map<Integer, List<Integer>> adjList = new HashMap<>();

   public void addNode(int node) {
       adjList.putIfAbsent(node, new ArrayList<>());
   }
 public void addEdge(int from, int to) {
       addNode(from);
       addNode(to);
       adjList.get(from).add(to);
       adjList.get(to).add(from);
   }
   public void dfs(int start) {
       Set<Integer> visited = new HashSet<>();
       dfsUtil(start, visited);
   }
   private void dfsUtil(int node, Set<Integer> visited) {
       visited.add(node);
       System.out.print(node + " ");
       for (int neighbor : adjList.get(node)) {
           if (!visited.contains(neighbor)) {
               dfsUtil(neighbor, visited);
```

```
<terminated> DFSGraph [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\
DFS starting from node 0:
0 3 5
```

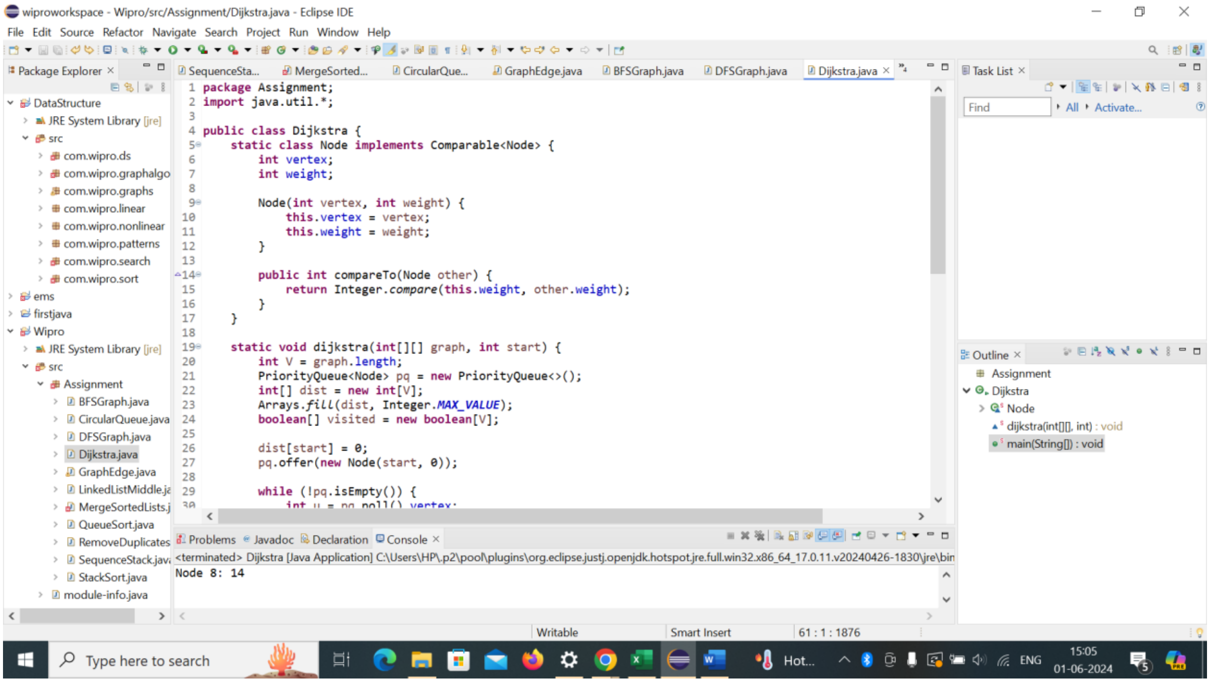

wiproworkspace - Wipro/src/Assignment/DFSGraph.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
SequenceSta... MergeSorted... CircularQue... GraphEdge.java BFSGraph.java *DFSGraph.java
Task List
Outline
Problems Javadoc Declaration Console

```java
 public void addEdge(int from, int to) {
       addNode(from);
       addNode(to);
       adjList.get(from).add(to);
       adjList.get(to).add(from);
   }
   public void dfs(int start) {
       Set<Integer> visited = new HashSet<>();
       dfsUtil(start, visited);
   }
   private void dfsUtil(int node, Set<Integer> visited) {
       visited.add(node);
       System.out.print(node + " ");
       for (int neighbor : adjList.get(node)) {
           if (!visited.contains(neighbor)) {
               dfsUtil(neighbor, visited);
           }
       }
   }
   public static void main(String[] args) {
       DFSGraph DFSGraph = new DFSGraph();
       DFSGraph.addEdge(0, 3);
       DFSGraph.addEdge(0, 5);
       DFSGraph.addEdge(1, 2);
       DFSGraph.addEdge(3, 5);
       System.out.println("DFS starting from node 0:");
       DFSGraph.dfs(0);
   }
}
```

```
<terminated> DFSGraph [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\
DFS starting from node 0:
0 3 5
```

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.

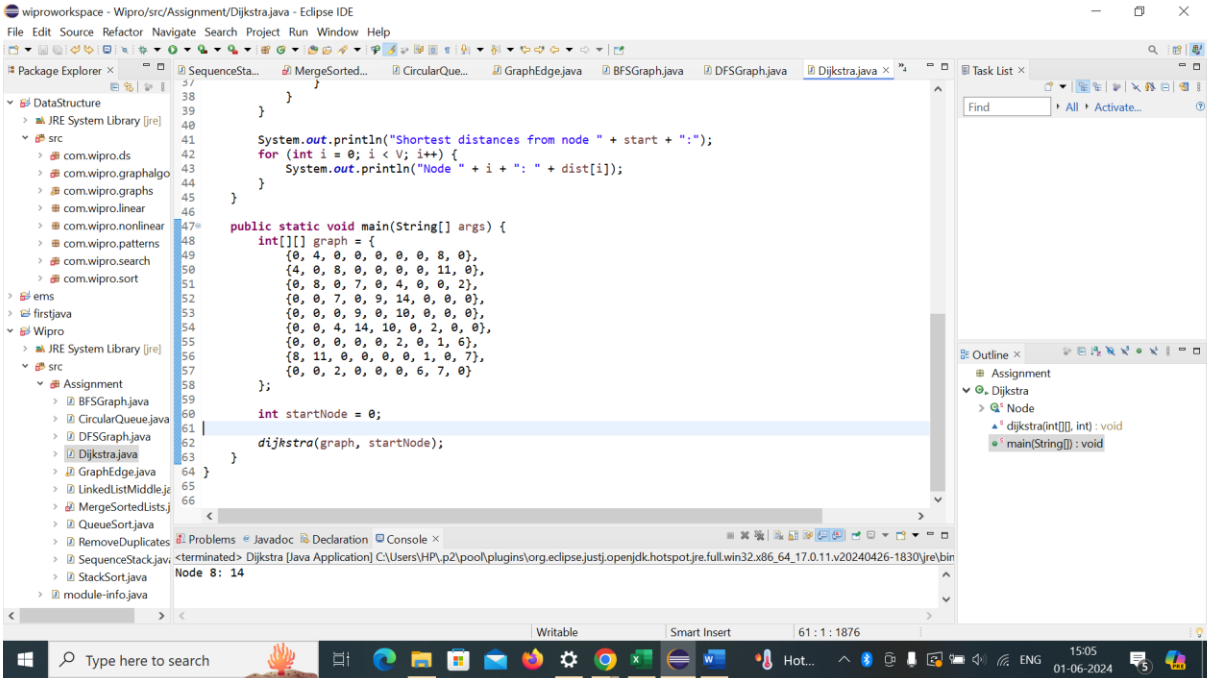


wiproworkspace - Wipro/src/Assignment/Dijkstra.java - Eclipse IDE

```java
package Assignment;
import java.util.*;

public class Dijkstra {
    static class Node implements Comparable<Node> {
        int vertex;
        int weight;

        Node(int vertex, int weight) {
            this.vertex = vertex;
            this.weight = weight;
        }

        public int compareTo(Node other) {
            return Integer.compare(this.weight, other.weight);
        }
    }

    static void dijkstra(int[][] graph, int start) {
        int V = graph.length;
        PriorityQueue<Node> pq = new PriorityQueue<>();
        int[] dist = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        boolean[] visited = new boolean[V];

        dist[start] = 0;
        pq.offer(new Node(start, 0));

        while (!pq.isEmpty()) {
            int u = pq.poll().vertex;
```

```
<terminated> Dijkstra [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin
Node 8: 14
```

File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
SequenceSta... MergeSorted... CircularQue... GraphEdge.java BFSGraph.java DFSGraph.java Dijkstra.java
Task List
Outline
Problems Javadoc Declaration Console
Writable
Smart Insert
61 : 1 : 1876
Type here to search
15:05
01-06-2024



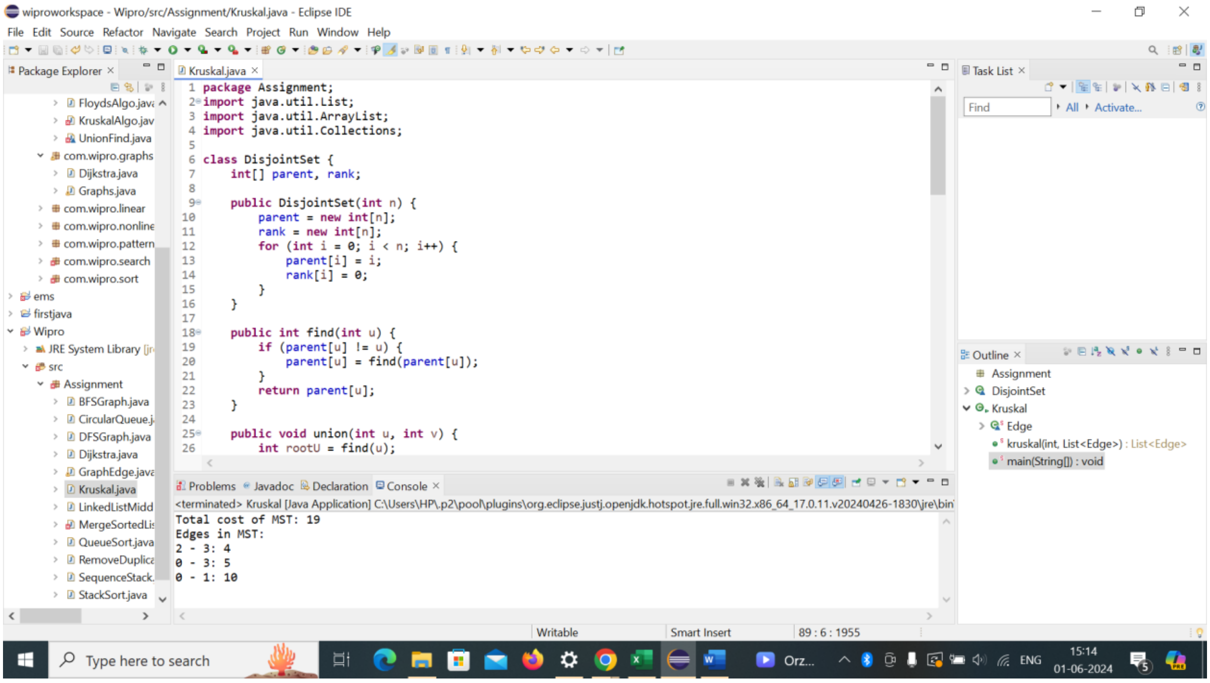
wiproworkspace - Wipro/src/Assignment/Dijkstra.java - Eclipse IDE

```java
                }
            }
        }

        System.out.println("Shortest distances from node " + start + ":");
        for (int i = 0; i < V; i++) {
            System.out.println("Node " + i + ": " + dist[i]);
        }
    }

    public static void main(String[] args) {
        int[][] graph = {
            {0, 4, 0, 0, 0, 0, 0, 8, 0},
            {4, 0, 8, 0, 0, 0, 0, 11, 0},
            {0, 8, 0, 7, 0, 4, 0, 0, 2},
            {0, 0, 7, 0, 9, 14, 0, 0, 0},
            {0, 0, 0, 9, 0, 10, 0, 0, 0},
            {0, 0, 4, 14, 10, 0, 2, 0, 0},
            {0, 0, 0, 0, 0, 2, 0, 1, 6},
            {8, 11, 0, 0, 0, 0, 1, 0, 7},
            {0, 0, 2, 0, 0, 0, 6, 7, 0}
        };

        int startNode = 0;

        dijkstra(graph, startNode);
    }
}
```

```
<terminated> Dijkstra [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin
Node 8: 14
```

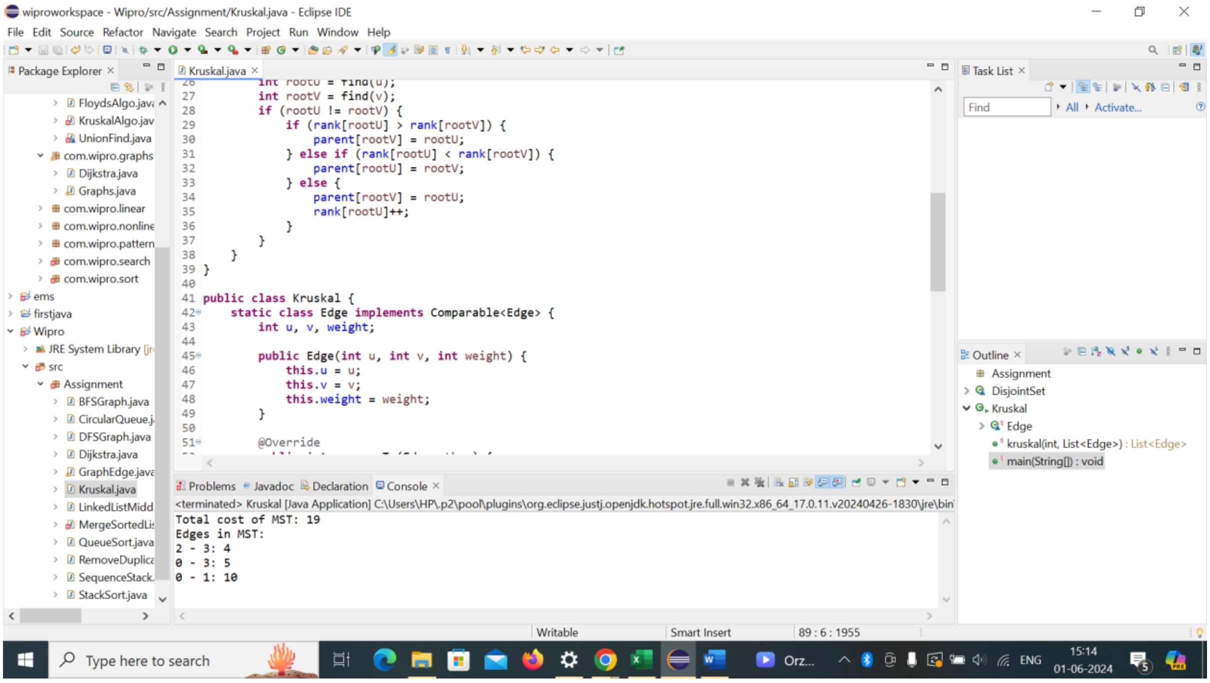
Task 2: Kruskal’s Algorithm for MST

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



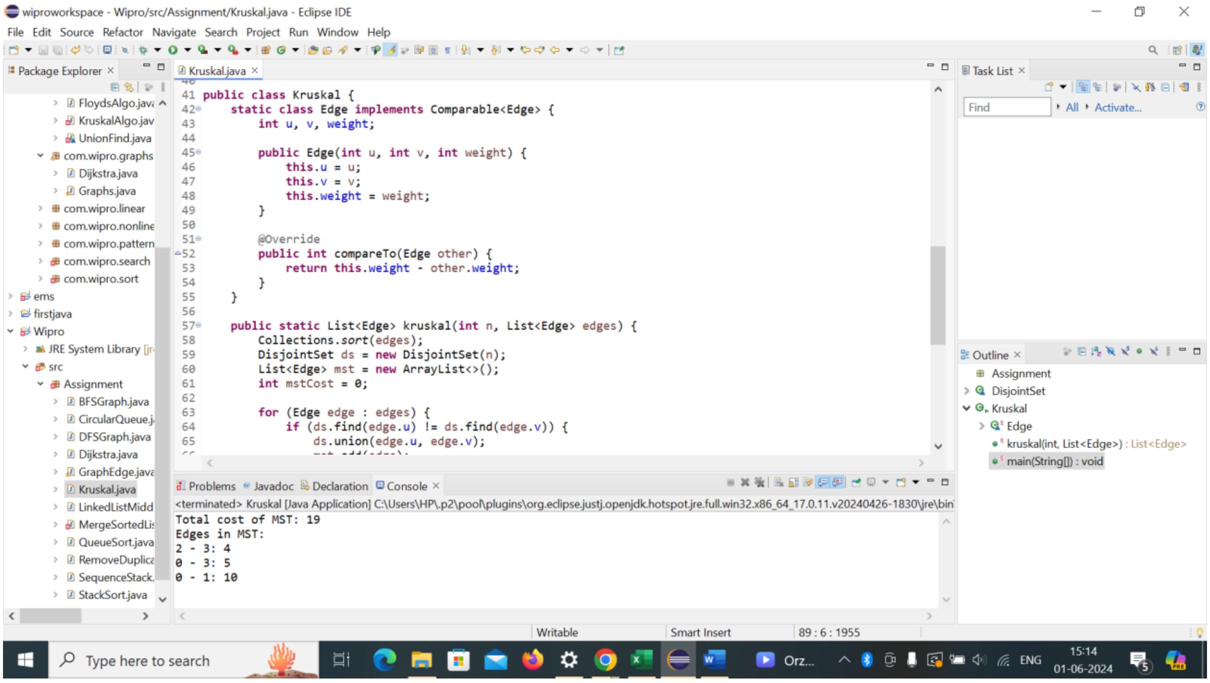
```java
package Assignment;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

class DisjointSet {
    int[] parent, rank;

    public DisjointSet(int n) {
        parent = new int[n];
        rank = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }

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

    public void union(int u, int v) {
        int rootU = find(u);
```

```
<terminated> Kruskal [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin
Total cost of MST: 19
Edges in MST:
2 - 3: 4
0 - 3: 5
0 - 1: 10
```



```java
        int rootU = find(u);
        int rootV = find(v);
        if (rootU != rootV) {
            if (rank[rootU] > rank[rootV]) {
                parent[rootV] = rootU;
            } else if (rank[rootU] < rank[rootV]) {
                parent[rootU] = rootV;
            } else {
                parent[rootV] = rootU;
                rank[rootU]++;
            }
        }
    }
}

public class Kruskal {
    static class Edge implements Comparable<Edge> {
        int u, v, weight;

        public Edge(int u, int v, int weight) {
            this.u = u;
            this.v = v;
            this.weight = weight;
        }

        @Override
```

```
<terminated> Kruskal [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin
Total cost of MST: 19
Edges in MST:
2 - 3: 4
0 - 3: 5
0 - 1: 10
```



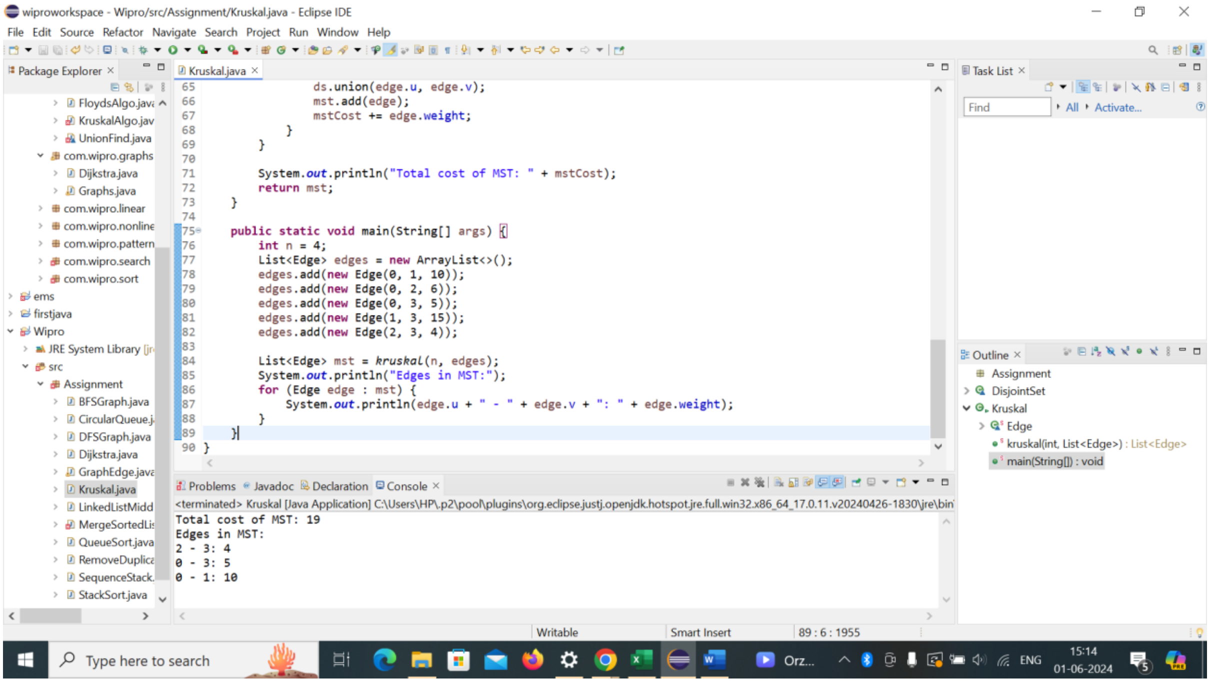
wiproworkspace - Wipro/src/Assignment/Kruskal.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Kruskal.java
Task List
Outline
Problems Javadoc Declaration Console
Writable
Smart Insert
89 : 6 : 1955

```java
public class Kruskal {
    static class Edge implements Comparable<Edge> {
        int u, v, weight;

        public Edge(int u, int v, int weight) {
            this.u = u;
            this.v = v;
            this.weight = weight;
        }

        @Override
        public int compareTo(Edge other) {
            return this.weight - other.weight;
        }
    }

    public static List<Edge> kruskal(int n, List<Edge> edges) {
        Collections.sort(edges);
        DisjointSet ds = new DisjointSet(n);
        List<Edge> mst = new ArrayList<>();
        int mstCost = 0;

        for (Edge edge : edges) {
            if (ds.find(edge.u) != ds.find(edge.v)) {
                ds.union(edge.u, edge.v);
```

<terminated> Kruskal [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin

```
Total cost of MST: 19
Edges in MST:
2 - 3: 4
0 - 3: 5
0 - 1: 10
```



wiproworkspace - Wipro/src/Assignment/Kruskal.java - Eclipse IDE

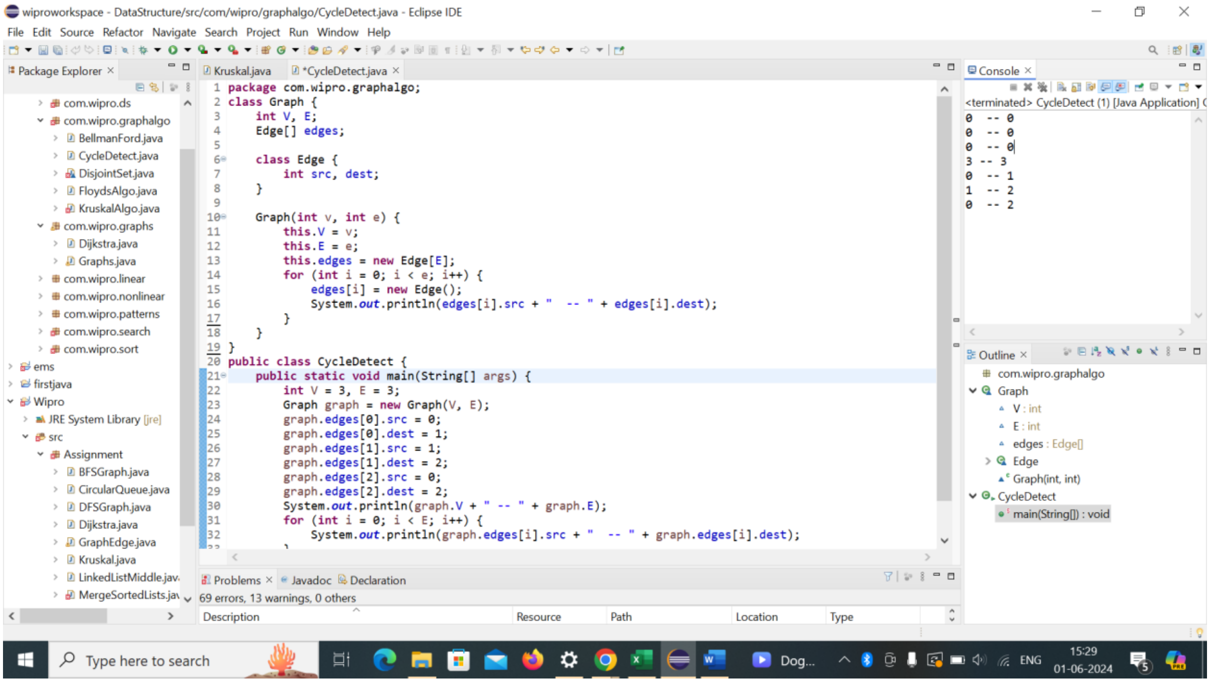
```java
                ds.union(edge.u, edge.v);
                mst.add(edge);
                mstCost += edge.weight;
            }
        }

        System.out.println("Total cost of MST: " + mstCost);
        return mst;
    }

    public static void main(String[] args) {
        int n = 4;
        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));

        List<Edge> mst = kruskal(n, edges);
        System.out.println("Edges in MST:");
        for (Edge edge : mst) {
            System.out.println(edge.u + " - " + edge.v + ": " + edge.weight);
        }
    }
}
```

<terminated> Kruskal [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin

Total cost of MST: 19
Edges in MST:
2 - 3: 4
0 - 3: 5
0 - 1: 10

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.



wiproworkspace - DataStructure/src/com/wipro/graphalgo/CycleDetect.java - Eclipse IDE

```java
package com.wipro.graphalgo;
class Graph {
    int V, E;
    Edge[] edges;

    class Edge {
        int src, dest;
    }

    Graph(int v, int e) {
        this.V = v;
        this.E = e;
        this.edges = new Edge[E];
        for (int i = 0; i < e; i++) {
            edges[i] = new Edge();
            System.out.println(edges[i].src + "  -- " + edges[i].dest);
        }
    }
}
public class CycleDetect {
    public static void main(String[] args) {
        int V = 3, E = 3;
        Graph graph = new Graph(V, E);
        graph.edges[0].src = 0;
        graph.edges[0].dest = 1;
        graph.edges[1].src = 1;
        graph.edges[1].dest = 2;
        graph.edges[2].src = 0;
        graph.edges[2].dest = 2;
        System.out.println(graph.V + " -- " + graph.E);
        for (int i = 0; i < E; i++) {
            System.out.println(graph.edges[i].src + "  -- " + graph.edges[i].dest);
```

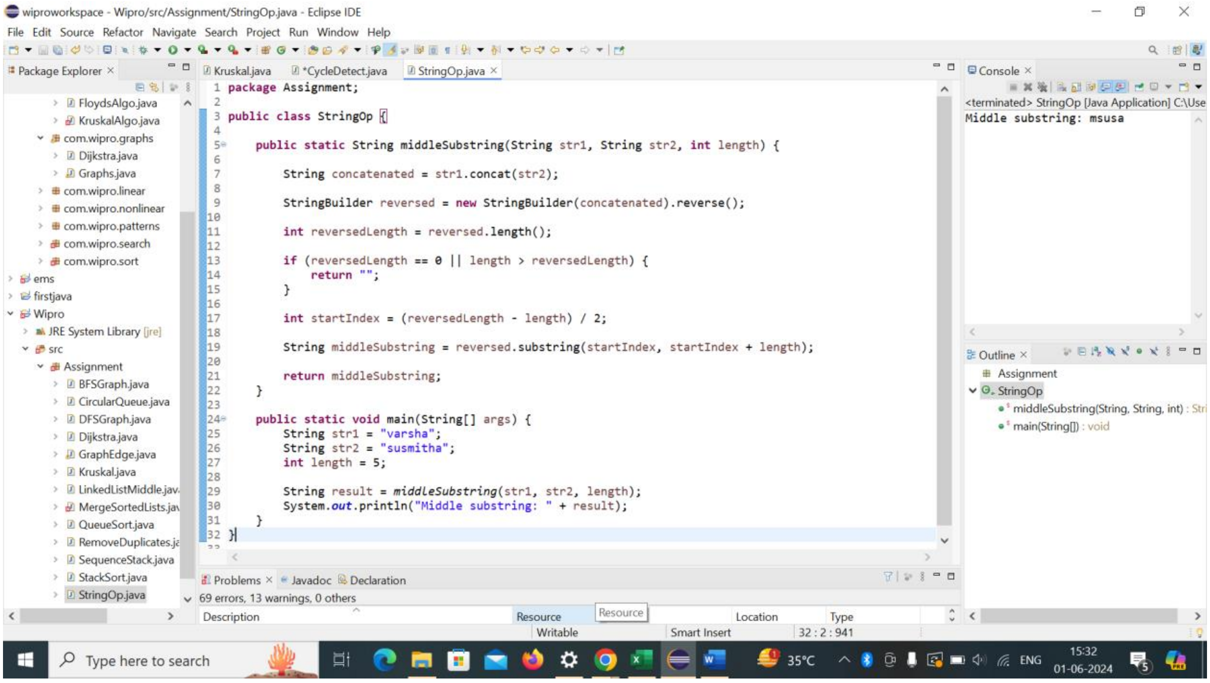
<terminated> CycleDetect (1) [Java Application]

```
0  -- 0
0  -- 0
0  -- 0
3 -- 3
0  -- 1
1  -- 2
0  -- 2
```

69 errors, 13 warnings, 0 others

Task 1: String Operations

Write a method that takes two strings, concatenates them, reverses the result, and then extracts the middle substring of the given length. Ensure your method handles edge cases, such as an empty string or a substring length larger than the concatenated string.



wiproworkspace - Wipro/src/Assignment/StringOp.java - Eclipse IDE

```java
package Assignment;

public class StringOp {

    public static String middleSubstring(String str1, String str2, int length) {

        String concatenated = str1.concat(str2);

        StringBuilder reversed = new StringBuilder(concatenated).reverse();

        int reversedLength = reversed.length();

        if (reversedLength == 0 || length > reversedLength) {
            return "";
        }

        int startIndex = (reversedLength - length) / 2;

        String middleSubstring = reversed.substring(startIndex, startIndex + length);

        return middleSubstring;
    }

    public static void main(String[] args) {
        String str1 = "varsha";
        String str2 = "susmitha";
        int length = 5;

        String result = middleSubstring(str1, str2, length);
        System.out.println("Middle substring: " + result);
    }
}
```

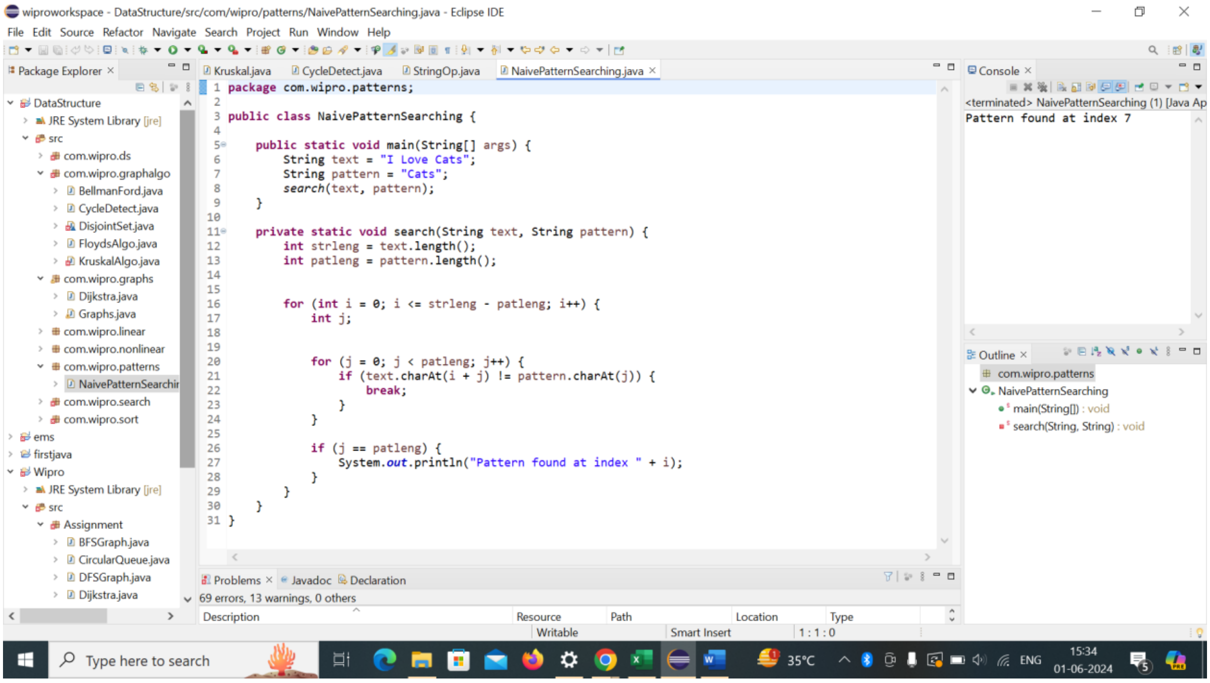
<terminated> StringOp [Java Application]

```
Middle substring: msusa
```

69 errors, 13 warnings, 0 others

Task 2: Naive Pattern Search

Implement the naive pattern searching algorithm to find all occurrences of a pattern within a given text string. Count the number of comparisons made during the search to evaluate the efficiency of the algorithm



wiproworkspace - DataStructure/src/com/wipro/patterns/NaivePatternSearching.java - Eclipse IDE

```java
package com.wipro.patterns;

public class NaivePatternSearching {

    public static void main(String[] args) {
        String text = "I Love Cats";
        String pattern = "Cats";
        search(text, pattern);
    }

    private static void search(String text, String pattern) {
        int strleng = text.length();
        int patleng = pattern.length();


        for (int i = 0; i <= strleng - patleng; i++) {
            int j;


            for (j = 0; j < patleng; j++) {
                if (text.charAt(i + j) != pattern.charAt(j)) {
                    break;
                }
            }

            if (j == patleng) {
                System.out.println("Pattern found at index " + i);
            }
        }
    }
}
```

<terminated> NaivePatternSearching (1) [Java Ap

```
Pattern found at index 7
```

69 errors, 13 warnings, 0 others