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.

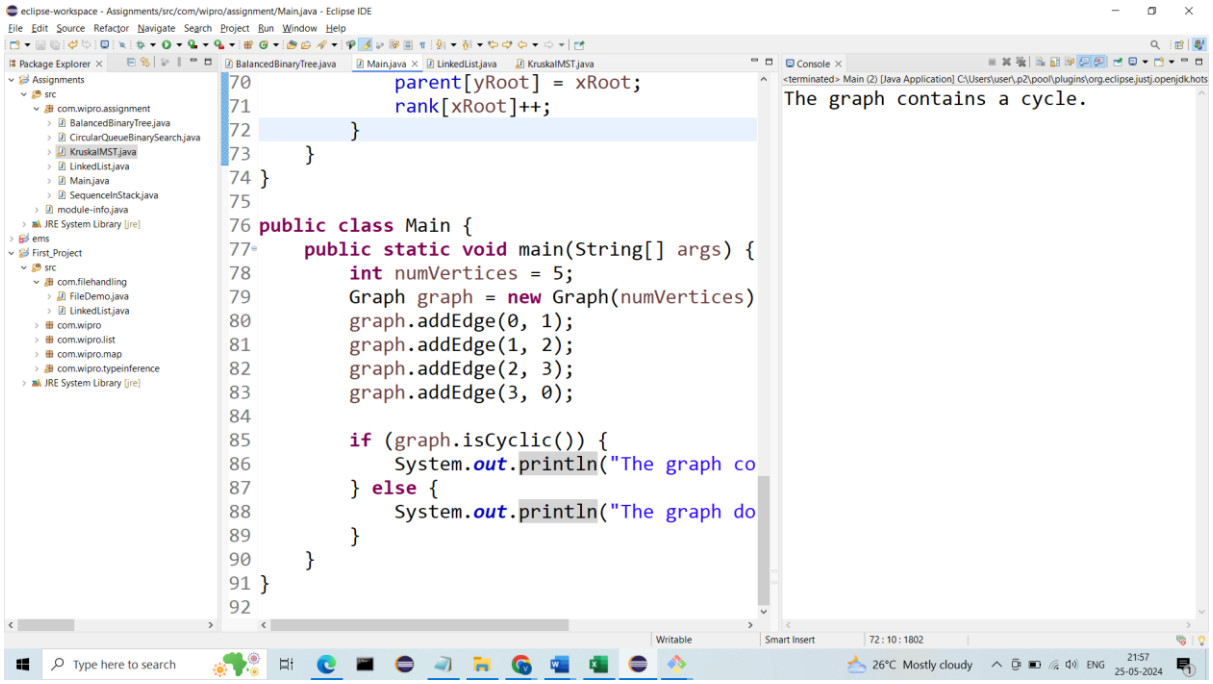
```
package com.wipro.assignment;
import java.util.*;

class Graph {
    private int numVertices;
    private List<List<Integer>> adjList;

    public Graph(int numVertices) {
        this.numVertices = numVertices;
        adjList = new
ArrayList<>(numVertices);
        for (int i = 0; i < numVertices;
i++) {
            adjList.add(new
ArrayList<>());
        }
    }

    public void addEdge(int u, int v) {
        adjList.get(u).add(v);
        adjList.get(v).add(u);
```

```
    }

    public boolean isCyclic() {
        UnionFind unionFind = new
UnionFind(numVertices);

        for (int u = 0; u < numVertices;
u++) {
            for (int v : adjList.get(u))
{
                if (unionFind.find(u) ==
unionFind.find(v)) {
                    return true; //
There's a cycle
                }
                unionFind.union(u, v);
            }
        }

        return false; // No cycle found
    }
}

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

    public UnionFind(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 x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }

    public void union(int x, int y) {
        int xRoot = find(x);
        int yRoot = find(y);

        if (xRoot == yRoot) {
            return;
        }

        if (rank[xRoot] < rank[yRoot]) {
            parent[xRoot] = yRoot;
        } else if (rank[xRoot] >
rank[yRoot]) {
            parent[yRoot] = xRoot;
        } else {
            parent[yRoot] = xRoot;
```

```
                rank[xRoot]++;
            }
        }
    }

public class Main {
    public static void main(String[]
args) {
        int numVertices = 5;
        Graph graph = new
Graph(numVertices);
        graph.addEdge(0, 1);
        graph.addEdge(1, 2);
        graph.addEdge(2, 3);
        graph.addEdge(3, 0);

        if (graph.isCyclic()) {
            System.out.println("The graph
contains a cycle.");
        } else {
            System.out.println("The graph
does not contain a cycle.");
        }
    }
}
```

Output: -


eclipse-workspace - Assignments/src/com/wipro/assignment/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Assignments
src
com.wipro.assignment
BalancedBinaryTree.java
CircularQueueBinarySearch.java
KruskalMST.java
LinkedList.java
Main.java
SequenceInStack.java
module-info.java
JRE System Library [jre]
ems
First_Project
src
com.filehandling
FileDemo.java
LinkedList.java
com.wipro
com.wipro.list
com.wipro.map
com.wipro.typeinference
JRE System Library [jre]
BalancedBinaryTree.java
Main.java
LinkedList.java
KruskalMST.java
parent[yRoot] = xRoot;
rank[xRoot]++;
}
}
}
public class Main {
public static void main(String[] args) {
int numVertices = 5;
Graph graph = new Graph(numVertices)
graph.addEdge(0, 1);
graph.addEdge(1, 2);
graph.addEdge(2, 3);
graph.addEdge(3, 0);
if (graph.isCyclic()) {
System.out.println("The graph co
} else {
System.out.println("The graph do
}
}
}
Console
<terminated> Main (2) [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hots
The graph contains a cycle.
Writable
Smart Insert
72 : 10 : 1802
Type here to search
26°C Mostly cloudy
ENG
21:57
25-05-2024