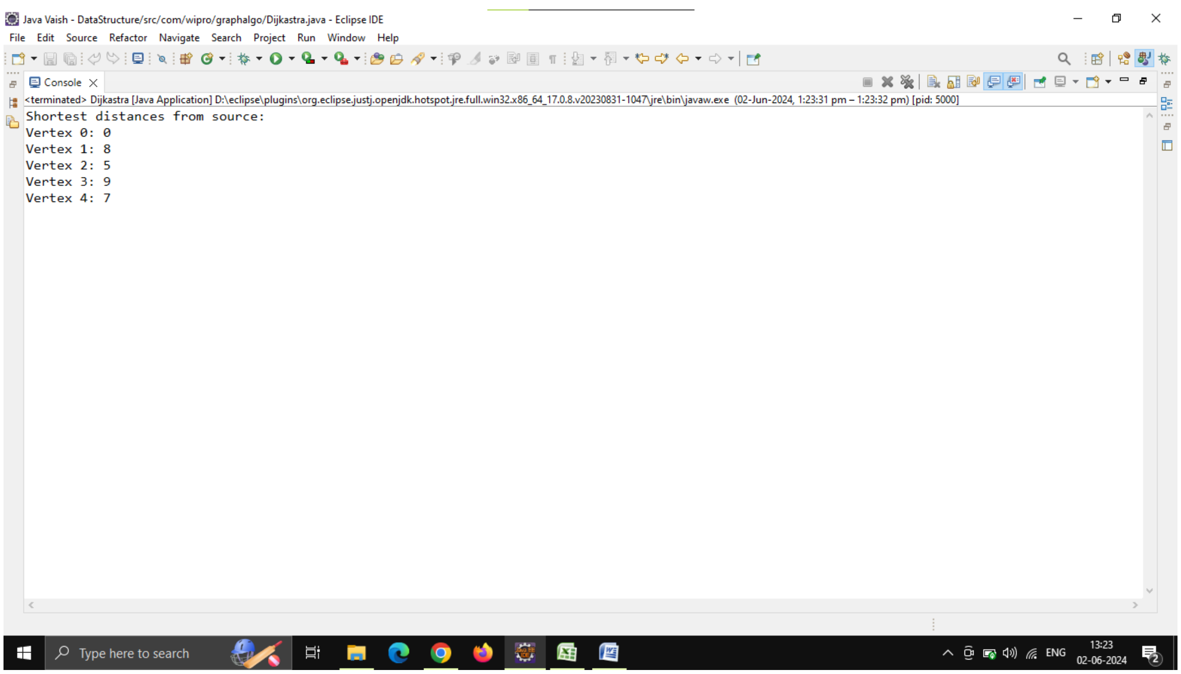
Task 1: Dijkstra's Shortest Path Finder

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

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

public class Dijkastra {


    public void dijkstra(List<List<int[]>> graph, int src) {
        int V = graph.size();
        int[] dist = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;

        boolean[] visited = new boolean[V];


        int[] minHeap = new int[V];
        int heapSize = 0;


        minHeap[heapSize++] = src;

        while (heapSize > 0) {

            int u = extractMin(minHeap, dist, heapSize);
            heapSize--;

            if (visited[u]) continue;
            visited[u] = true;

            for (int[] edge : graph.get(u)) {
                int v = edge[0], weight = edge[1];
                if (dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                    if (!visited[v]) {
                        minHeap[heapSize++] = v;
                        heapifyUp(minHeap, dist, heapSize - 1);
                    }
                }
            }
        }

        printSolution(dist);
    }
```

```java
    private int extractMin(int[] minHeap, int[] dist, int
heapSize) {
        int minIndex = 0;
        for (int i = 1; i < heapSize; i++) {
            if (dist[minHeap[i]] < dist[minHeap[minIndex]]) {
                minIndex = i;
            }
        }
        int minVertex = minHeap[minIndex];
        minHeap[minIndex] = minHeap[heapSize - 1];
        return minVertex;
    }

    private void heapifyUp(int[] minHeap, int[] dist, int index) {
        while (index > 0) {
            int parent = (index - 1) / 2;
            if (dist[minHeap[index]] < dist[minHeap[parent]]) {
                swap(minHeap, index, parent);
                index = parent;
            } else {
                break;
            }
        }
    }


    private void swap(int[] minHeap, int i, int j) {
        int temp = minHeap[i];
        minHeap[i] = minHeap[j];
        minHeap[j] = temp;
    }


    private void printSolution(int[] dist) {
        System.out.println("Shortest distances from source:");
        for (int i = 0; i < dist.length; i++) {
            System.out.println("Vertex " + i + ": " + dist[i]);
        }
    }


    public static void main(String[] args) {

        List<List<int[]>> graph = new ArrayList<>();
        int V = 5;
        for (int i = 0; i < V; i++) {
            graph.add(new ArrayList<>());
        }
        graph.get(0).add(new int[]{1, 10});
        graph.get(0).add(new int[]{2, 5});
```

```java
        graph.get(1).add(new int[]{2, 2});
        graph.get(1).add(new int[]{3, 1});
        graph.get(2).add(new int[]{1, 3});
        graph.get(2).add(new int[]{3, 9});
        graph.get(2).add(new int[]{4, 2});
        graph.get(3).add(new int[]{4, 4});
        graph.get(4).add(new int[]{0, 7});

        Dijkastra dijkstra = new Dijkastra();
        int src = 0;
        dijkstra.dijkstra(graph, src);
    }
}
```

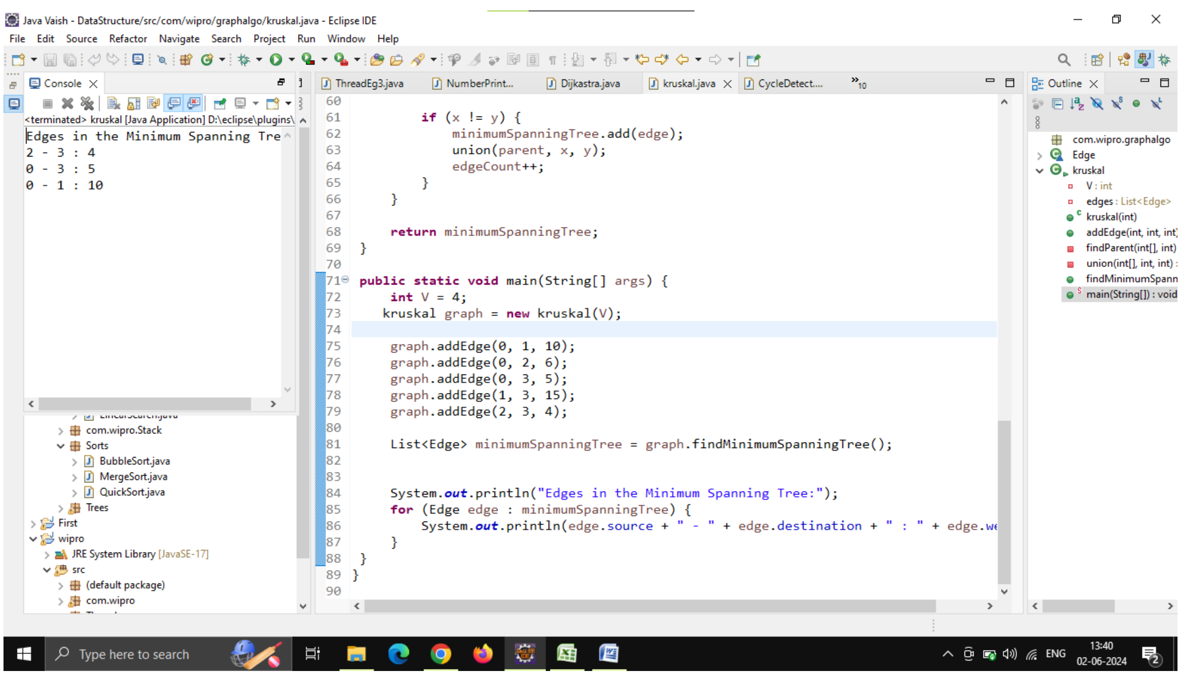


Task 2: Kruskal's Algorithm for MST

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

```
package com.wipro.graphalgo;
import java.util.*;
class Edge implements Comparable<Edge> {
 int source, destination, weight;

 public Edge(int source, int destination, int weight) {
     this.source = source;
     this.destination = destination;
     this.weight = weight;
 }

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

public class kruskal{
 private int V;
 private List<Edge> edges;

 public kruskal(int V) {
     this.V = V;
     this.edges = new ArrayList<>();
 }

 public void addEdge(int source, int destination, int weight) {
     edges.add(new Edge(source, destination, weight));
 }
 private int findParent(int[] parent, int vertex) {
     if (parent[vertex] == vertex)
         return vertex;
     return findParent(parent, parent[vertex]);
 }

 private void union(int[] parent, int x, int y) {
     int xSet = findParent(parent, x);
     int ySet = findParent(parent, y);
     parent[ySet] = xSet;
 }


 public List<Edge> findMinimumSpanningTree() {
     List<Edge> minimumSpanningTree = new ArrayList<>();
```

```java
        Collections.sort(edges);

        int[] parent = new int[V];
        for (int i = 0; i < V; i++) {
            parent[i] = i;
        }

        int edgeCount = 0;
        for (Edge edge : edges) {
            if (edgeCount == V - 1)
                break;

            int x = findParent(parent, edge.source);
            int y = findParent(parent, edge.destination);

            if (x != y) {
                minimumSpanningTree.add(edge);
                union(parent, x, y);
                edgeCount++;
            }
        }

        return minimumSpanningTree;
    }

    public static void main(String[] args) {
        int V = 4;
       kruskal graph = new kruskal(V);

        graph.addEdge(0, 1, 10);
        graph.addEdge(0, 2, 6);
        graph.addEdge(0, 3, 5);
        graph.addEdge(1, 3, 15);
        graph.addEdge(2, 3, 4);

        List<Edge> minimumSpanningTree = 
graph.findMinimumSpanningTree();


        System.out.println("Edges in the Minimum Spanning Tree:");
        for (Edge edge : minimumSpanningTree) {
            System.out.println(edge.source + " - " + edge.destination 
+ " : " + edge.weight);
        }
    }
}
```


Java Vaish - DataStructure/src/com/wipro/graphalgo/kruskal.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Console
<terminated> kruskal [Java Application] D:\eclipse\plugins\
Edges in the Minimum Spanning Tre
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10
com.wipro.Stack
Sorts
BubbleSort.java
MergeSort.java
QuickSort.java
Trees
First
wipro
JRE System Library [JavaSE-17]
src
(default package)
com.wipro
ThreadEg3.java
NumberPrint...
Dijkastra.java
kruskal.java
CycleDetect...
if (x != y) {
minimumSpanningTree.add(edge);
union(parent, x, y);
edgeCount++;
}
}
return minimumSpanningTree;
}
public static void main(String[] args) {
int V = 4;
kruskal graph = new kruskal(V);
graph.addEdge(0, 1, 10);
graph.addEdge(0, 2, 6);
graph.addEdge(0, 3, 5);
graph.addEdge(1, 3, 15);
graph.addEdge(2, 3, 4);
List<Edge> minimumSpanningTree = graph.findMinimumSpanningTree();
System.out.println("Edges in the Minimum Spanning Tree:");
for (Edge edge : minimumSpanningTree) {
System.out.println(edge.source + " - " + edge.destination + " : " + edge.we
}
}
}
Outline
com.wipro.graphalgo
Edge
kruskal
V : int
edges : List<Edge>
kruskal(int)
addEdge(int, int, int)
findParent(int[], int)
union(int[], int, int)
findMinimumSpann
main(String[]) : void
Type here to search
ENG
13:40
02-06-2024

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.graphalgo;
import java.util.Arrays;
class UnionFind{
	int[] parent;
	int[] rank;

	UnionFind(int n){
		parent=new int[n];
		rank=new int[n];
		Arrays.fill(rank,1);
		for(int i=0;i<n;i++) {
			parent[i]=i;
		}
	}
	int find(int i) {
		if(parent[i] != i) {
			parent[i]=find(parent[i]);
		}
		return parent[i];
	}
	void union(int x,int y) {
		int rootX = find(x);
		int rootY = find(y);

		if(rootX != rootY) {
			if (rank[rootX] < rank[rootY]) { // 1<2
				parent[rootX] = rootY;
			} else if (rank[rootX] > rank[rootY]) {
				parent[rootY] = rootX;
			} else {
				parent[rootY] = rootX;
				rank[rootX]++;

			}
		}
	}


}
```

```
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 boolean isCycleFound(Graph graph) {
		UnionFind uf = new UnionFind(V);
		for(int i=0;i<E;++i) {
		int x=find(uf,graph.edges[i].src);
		int y =find(uf,graph.edges[i].src);

		if(x==y) {
			return true;
		}
		uf.union(x,y);

		}
		return false;
	}
	private int find(UnionFind uf,int i) {
		return uf.find(i);
	}
}

public class CycleDetect {
	public static void main(String[] args){
		int V=3,E=2;

		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;
```

```java
		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);
		}

		if(graph.isCycleFound(graph)) {
			System.out.println("Cycle Found");
		}else {
			System.out.println("Cycle Not Found...");
		}
	}
}
```