Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenment

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.

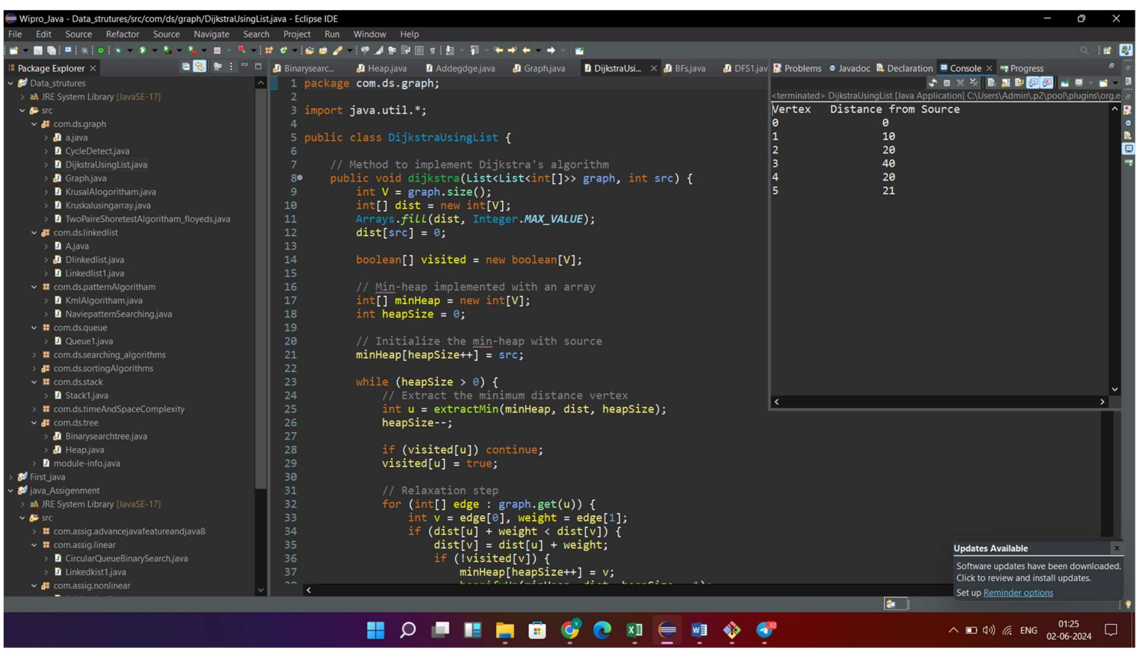
```
package com.ds.graph;

import java.util.*;

public class DijkstraUsingList {

    // Method to implement Dijkstra's algorithm
    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];

        // Min-heap implemented with an array
        int[] minHeap = new int[V];
        int heapSize = 0;

        // Initialize the min-heap with source
        minHeap[heapSize++] = src;

        while (heapSize > 0) {
            // Extract the minimum distance vertex
            int u = extractMin(minHeap, dist, heapSize);
            heapSize--;

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

            // Relaxation step
            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);
```

```
    }

    private void heapifyUp(int[] heap, int[] dist, int idx) {
        int parent = (idx - 1) / 2;
        while (idx > 0 && dist[heap[idx]] < dist[heap[parent]]) {
            int temp = heap[idx];
            heap[idx] = heap[parent];
            heap[parent] = temp;
            idx = parent;
            parent = (idx - 1) / 2;
        }
    }

    private int extractMin(int[] heap, int[] dist, int heapSize) {
        int minVertex = heap[0];
        heap[0] = heap[heapSize - 1];
        heapifyDown(heap, dist, 0, heapSize - 1);
        return minVertex;
    }

    private void heapifyDown(int[] heap, int[] dist, int idx, int
heapSize) {
        int smallest = idx;
        int left = 2 * idx + 1;
        int right = 2 * idx + 2;

        if (left < heapSize && dist[heap[left]] <
dist[heap[smallest]]) {
            smallest = left;
        }
        if (right < heapSize && dist[heap[right]] <
dist[heap[smallest]]) {
            smallest = right;
        }

        if (smallest != idx) {
            int temp = heap[idx];
            heap[idx] = heap[smallest];
            heap[smallest] = temp;
            heapifyDown(heap, dist, smallest, heapSize);
        }
    }

    private void printSolution(int[] dist) {
        System.out.println("Vertex \t Distance from Source");
        for (int i = 0; i < dist.length; i++) {
            System.out.println(i + " \t\t " + dist[i]);
        }
    }
```

Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenement

```java
    public static void main(String[] args) {
        int V = 6;
        List<List<int[]>> graph = new ArrayList<>();
        for (int i = 0; i < V; i++) {
            graph.add(new ArrayList<>());
        }

        // Add edges
        graph.get(0).add(new int[]{1, 10});
        graph.get(0).add(new int[]{2, 20});
        graph.get(1).add(new int[]{3, 50});
        graph.get(1).add(new int[]{4, 10});
        graph.get(2).add(new int[]{3, 20});
        graph.get(2).add(new int[]{4, 33});
        graph.get(3).add(new int[]{5, 2});
        graph.get(4).add(new int[]{5, 1});

        DijkstraUsingList dijkstra = new DijkstraUsingList();
        dijkstra.dijkstra(graph, 0); // 0 is the source vertex
    }
}
```

Op:



Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenment

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.

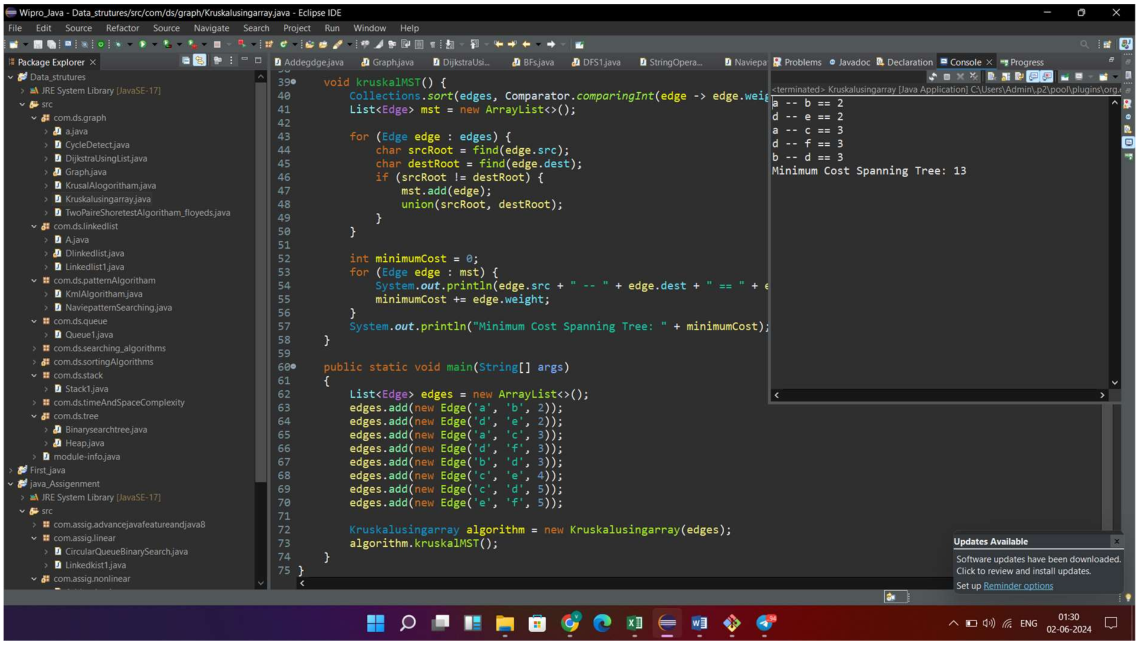
```
package com.ds.graph;



import java.util.*;

public class Kruskalusingarray {
    static class Edge {
        char src, dest;
        int weight;

        Edge(char src, char dest, int weight) {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
    }

    private List<Edge> edges;
    private Map<Character, Character> parent;

    Kruskalusingarray(List<Edge> edges) {
        this.edges = edges;
        this.parent = new HashMap<>();
    }

    private char find(char vertex) {
        if (!parent.containsKey(vertex))
            return vertex;
        return find(parent.get(vertex));
    }

    private void union(char src, char dest) {
        char srcRoot = find(src);
        char destRoot = find(dest);
        parent.put(srcRoot, destRoot);
    }

    void kruskalMST() {
        Collections.sort(edges, Comparator.comparingInt(edge ->
edge.weight));
        List<Edge> mst = new ArrayList<>();
```

Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenment

```
        for (Edge edge : edges) {
            char srcRoot = find(edge.src);
            char destRoot = find(edge.dest);
            if (srcRoot != destRoot) {
                mst.add(edge);
                union(srcRoot, destRoot);
            }
        }

        int minimumCost = 0;
        for (Edge edge : mst) {
            System.out.println(edge.src + " -- " + edge.dest + " ==
" + edge.weight);
            minimumCost += edge.weight;
        }
        System.out.println("Minimum Cost Spanning Tree: " +
minimumCost);
    }

    public static void main(String[] args)
    {
        List<Edge> edges = new ArrayList<>();
        edges.add(new Edge('a', 'b', 2));
        edges.add(new Edge('d', 'e', 2));
        edges.add(new Edge('a', 'c', 3));
        edges.add(new Edge('d', 'f', 3));
        edges.add(new Edge('b', 'd', 3));
        edges.add(new Edge('c', 'e', 4));
        edges.add(new Edge('c', 'd', 5));
        edges.add(new Edge('e', 'f', 5));

        Kruskalusingarray algorithm = new Kruskalusingarray(edges);
        algorithm.kruskalMST();
    }
}
```

Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenement

OP:



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.assig.nonlinear;

import java.util.*;

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

    // Constructor to initialize the Union-Find data structure
    public UnionFind(int size) {
        parent = new int[size];
        rank = new int[size];
        for (int i = 0; i < size; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }

    // Find with path compression
    public int find(int p) {
```

Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenment

```
        if (parent[p] != p) {
            parent[p] = find(parent[p]);
        }
        return parent[p];
    }

    // Union by rank
    public void union(int p, int q) {
        int rootP = find(p);
        int rootQ = find(q);

        if (rootP != rootQ) {
            if (rank[rootP] > rank[rootQ]) {
                parent[rootQ] = rootP;
            } else if (rank[rootP] < rank[rootQ]) {
                parent[rootP] = rootQ;
            } else {
                parent[rootQ] = rootP;
                rank[rootP]++;
            }
        }
    }

    // Method to detect cycle in an undirected graph
    public boolean hasCycle(List<int[]> edges) {
        for (int[] edge : edges) {
            int u = edge[0];
            int v = edge[1];

            int rootU = find(u);
            int rootV = find(v);

            if (rootU == rootV) {
                return true; // Cycle detected
            } else {
                union(u, v);
            }
        }
        return false; // No cycle detected
    }

    public static void main(String[] args) {
        int numberOfVertices = 5;
        UnionFind uf = new UnionFind(numberOfVertices);

        List<int[]> edges = new ArrayList<>();
        edges.add(new int[]{0, 1});
        edges.add(new int[]{1, 2});
        edges.add(new int[]{2, 3});
```

Name:Vijay Lahuraj Patil Email:Vijaylpatil454545@gmail.com
Day 9 10 assigenement

```
        edges.add(new int[]{3, 4});
        edges.add(new int[]{4, 0}); // Adding this edge creates a cycle

        if (uf.hasCycle(edges)) {
            System.out.println("Graph contains a cycle");
        } else {
            System.out.println("Graph does not contain a cycle");
        }
    }
}
```

Op: