Task 1: Dijkstra's Shortest Path Finder

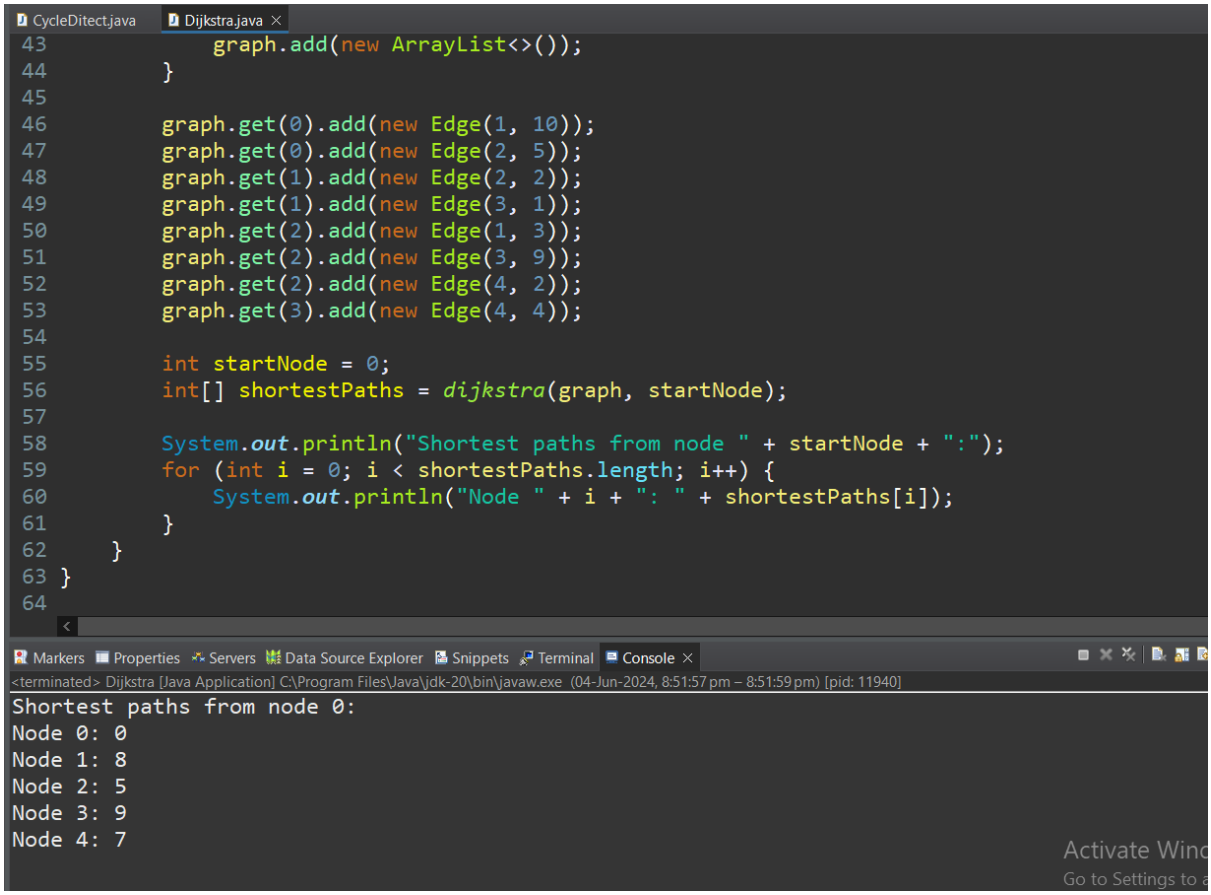
Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights

CycleDitect.java Dijkstra.java

```java
package com.assignment;

import java.util.*;

public class Dijkstra {
    static class Edge {
        int to, weight;

        Edge(int to, int weight) {
            this.to = to;
            this.weight = weight;
        }
    }

    public static int[] dijkstra(List<List<Edge>> graph, int start) {
        int n = graph.size();
        int[] dist = new int[n];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[start] = 0;

        PriorityQueue<Edge> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a.weight));
        pq.offer(new Edge(start, 0));
```

CycleDitect.java Dijkstra.java

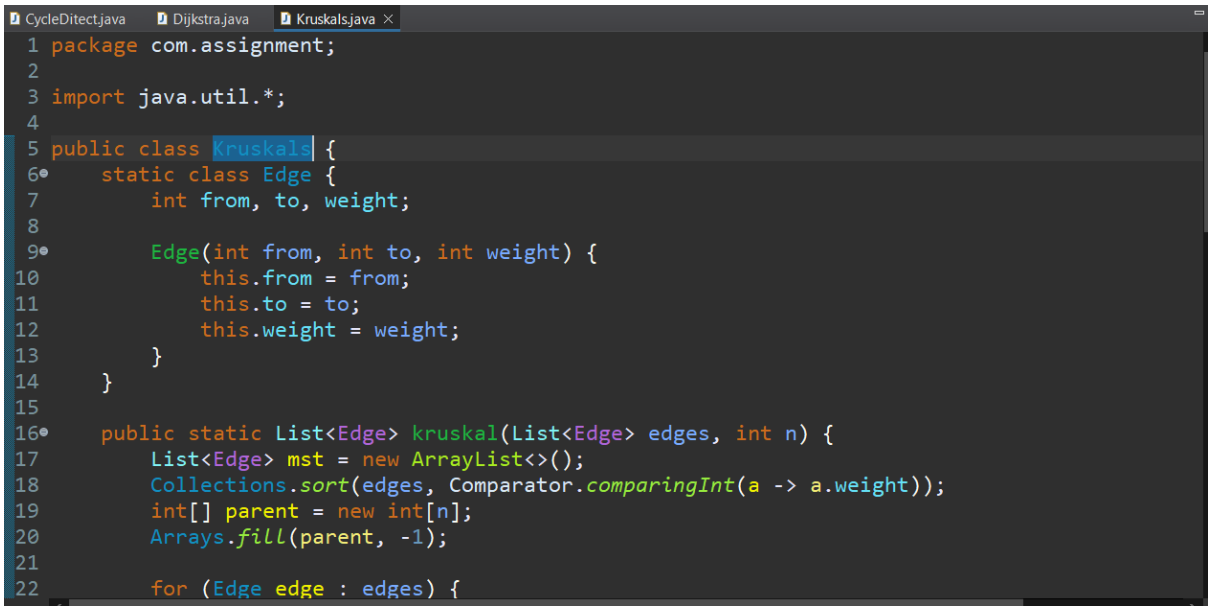
```java

        while (!pq.isEmpty()) {
            Edge curr = pq.poll();
            int u = curr.to;
            for (Edge next : graph.get(u)) {
                int v = next.to;
                int newDist = dist[u] + next.weight;
                if (newDist < dist[v]) {
                    dist[v] = newDist;
                    pq.offer(new Edge(v, newDist));
                }
            }
        }
        return dist;
    }

    public static void main(String[] args) {
        int n = 5;
        List<List<Edge>> graph = new ArrayList<>(n);
        for (int i = 0; i < n; i++) {
            graph.add(new ArrayList<>());
        }
```



```
CycleDitect.java  Dijkstra.java ×
43              graph.add(new ArrayList<>());
44          }
45
46          graph.get(0).add(new Edge(1, 10));
47          graph.get(0).add(new Edge(2, 5));
48          graph.get(1).add(new Edge(2, 2));
49          graph.get(1).add(new Edge(3, 1));
50          graph.get(2).add(new Edge(1, 3));
51          graph.get(2).add(new Edge(3, 9));
52          graph.get(2).add(new Edge(4, 2));
53          graph.get(3).add(new Edge(4, 4));
54
55          int startNode = 0;
56          int[] shortestPaths = dijkstra(graph, startNode);
57
58          System.out.println("Shortest paths from node " + startNode + ":");
59          for (int i = 0; i < shortestPaths.length; i++) {
60              System.out.println("Node " + i + ": " + shortestPaths[i]);
61          }
62      }
63  }
64
Markers  Properties  Servers  Data Source Explorer  Snippets  Terminal  Console ×
<terminated> Dijkstra [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe  (04-Jun-2024, 8:51:57 pm – 8:51:59 pm) [pid: 11940]
Shortest paths from node 0:
Node 0: 0
Node 1: 8
Node 2: 5
Node 3: 9
Node 4: 7
Activate Windo
Go to Settings to a
```

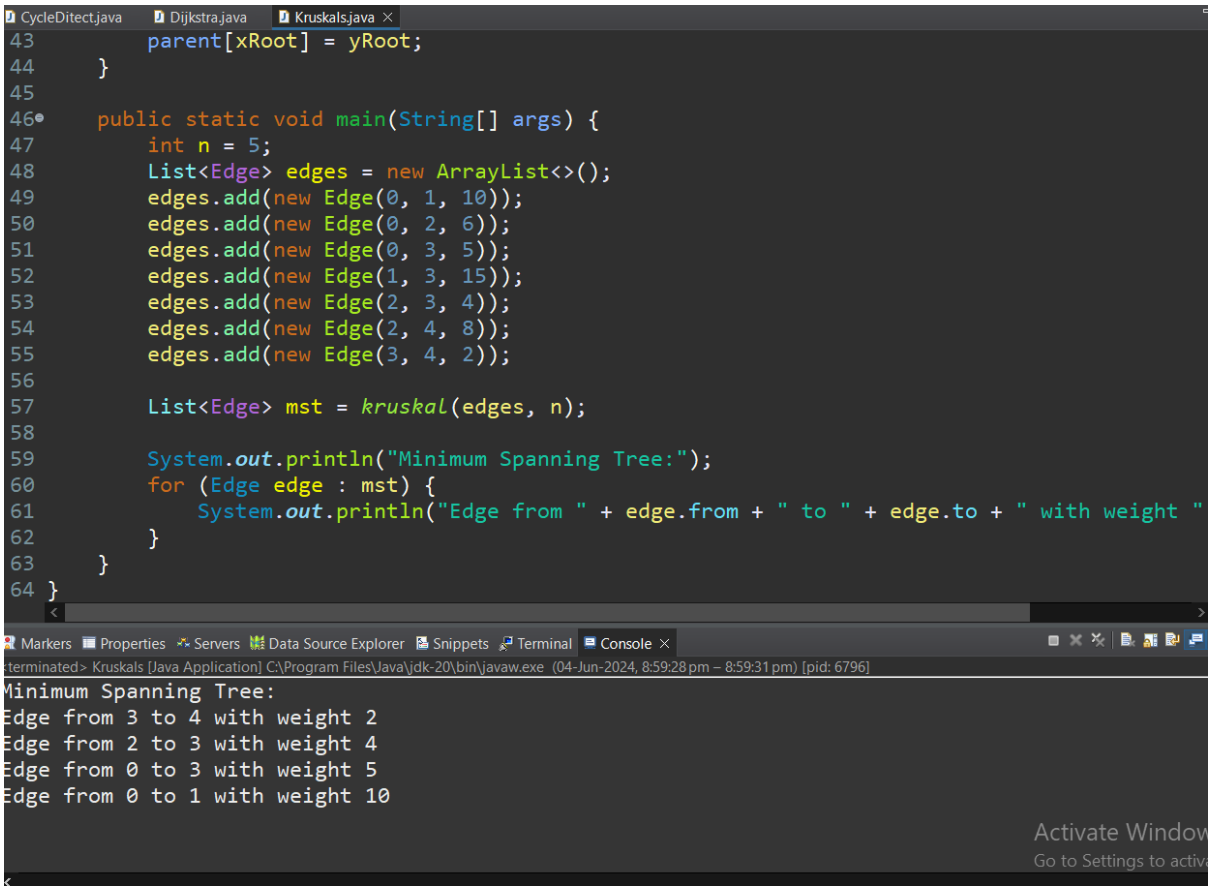
## Task 2: Kruskal's Algorithm for MST

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



```
CycleDitect.java  Dijkstra.java  Kruskals.java ×
1  package com.assignment;
2
3  import java.util.*;
4
5  public class Kruskals {
6      static class Edge {
7          int from, to, weight;
8
9          Edge(int from, int to, int weight) {
10             this.from = from;
11             this.to = to;
12             this.weight = weight;
13         }
14     }
15
16     public static List<Edge> kruskal(List<Edge> edges, int n) {
17         List<Edge> mst = new ArrayList<>();
18         Collections.sort(edges, Comparator.comparingInt(a -> a.weight));
19         int[] parent = new int[n];
20         Arrays.fill(parent, -1);
21
22         for (Edge edge : edges) {
```

```
CycleDitect.java  Dijkstra.java  Kruskals.java
22          for (Edge edge : edges) {
23              int x = find(parent, edge.from);
24              int y = find(parent, edge.to);
25              if (x != y) {
26                  mst.add(edge);
27                  union(parent, x, y);
28              }
29          }
30          return mst;
31      }
32
33      public static int find(int[] parent, int i) {
34          if (parent[i] == -1) {
35              return i;
36          }
37          return find(parent, parent[i]);
38      }
39
40      public static void union(int[] parent, int x, int y) {
41          int xRoot = find(parent, x);
42          int yRoot = find(parent, y);
43          parent[xRoot] = yRoot;
```



```
CycleDitect.java  Dijkstra.java  Kruskals.java
43          parent[xRoot] = yRoot;
44      }
45
46      public static void main(String[] args) {
47          int n = 5;
48          List<Edge> edges = new ArrayList<>();
49          edges.add(new Edge(0, 1, 10));
50          edges.add(new Edge(0, 2, 6));
51          edges.add(new Edge(0, 3, 5));
52          edges.add(new Edge(1, 3, 15));
53          edges.add(new Edge(2, 3, 4));
54          edges.add(new Edge(2, 4, 8));
55          edges.add(new Edge(3, 4, 2));
56
57          List<Edge> mst = kruskal(edges, n);
58
59          System.out.println("Minimum Spanning Tree:");
60          for (Edge edge : mst) {
61              System.out.println("Edge from " + edge.from + " to " + edge.to + " with weight "
62          }
63      }
64  }
```

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Terminal  Console
<terminated> Kruskals [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe  (04-Jun-2024, 8:59:28 pm – 8:59:31 pm) [pid: 6796]
Minimum Spanning Tree:
Edge from 3 to 4 with weight 2
Edge from 2 to 3 with weight 4
Edge from 0 to 3 with weight 5
Edge from 0 to 1 with weight 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.

CycleDitect.java

```java
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]);
        }
```

CycleDitect.java

```java
        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]++;

            }

        }

    }

```

CycleDitect.java

```
46
47 class Graph {
48     int V, E;
49     Edge[] edges;
50
51     class Edge {
52         int src, dest;
53     }
54
55     Graph(int v, int e) {
56         this.V = v;
57         this.E = e;
58         this.edges = new Edge[E];
59         for (int i = 0; i < e; i++) {
60             edges[i] = new Edge();
61             System.out.println(edges[i].src + "   -- " + edges[i].dest);
62         }
63     }
64
65     public boolean isCycleFound(Graph graph) {
66         UnionFind uf = new UnionFind(V);
67         for(int i=0; i< E ; ++i) {
```

CycleDitect.java

```
64
65     public boolean isCycleFound(Graph graph) {
66         UnionFind uf = new UnionFind(V);
67         for(int i=0; i< E ; ++i) {
68             int x = find(uf, graph.edges[i].src);
69             int y = find(uf, graph.edges[i].dest);
70
71             if(x==y) {
72                 return true;
73             }
74             uf.union(x, y);
75         }
76         return false;
77     }
78
79     private int find(UnionFind uf, int i) {
80
81         return uf.find(i);
82     }
83
84 }
85
```

CycleDitect.java

```
85
86 public class CycleDitect {
87     public static void main(String[] args) {
88         //int V = 3, E = 3;
89         int V = 3, E = 2;
90         Graph graph = new Graph(V, E);
91
92         graph.edges[0].src = 0;
93         graph.edges[0].dest = 1;
94
95         graph.edges[1].src = 1;
96         graph.edges[1].dest = 2;
97
98         //graph.edges[2].src = 0;
99         //graph.edges[2].dest = 2;
100
101        System.out.println(graph.V + " -- " + graph.E);
102        for (int i = 0; i < E; i++) {
103
104            System.out.println(graph.edges[i].src + "  -- " + graph.edges[i].dest);
105        }
106
```

CycleDitect.java

```
96          graph.edges[1].dest = 2;
97
98          //graph.edges[2].src = 0;
99          //graph.edges[2].dest = 2;
100
101         System.out.println(graph.V + " -- " + graph.E);
102         for (int i = 0; i < E; i++) {
103
104             System.out.println(graph.edges[i].src + "  -- " + graph.edges[i].dest);
105         }
106
107
108         if(graph.isCycleFound(graph)) {
109             System.out.println("Cycle Found");
110         }else {
111             System.out.println("Cycle Not Found...");
112         }
113
114     }
115 }
116
117
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Terminal | Console

<terminated> CycleDitect [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe (04-Jun-2024, 9:01:33 pm – 9:01:35 pm) [pid: 10312]

```
0  -- 0
0  -- 0
3 -- 2
0  -- 1
1  -- 2
Cycle Not Found...
```