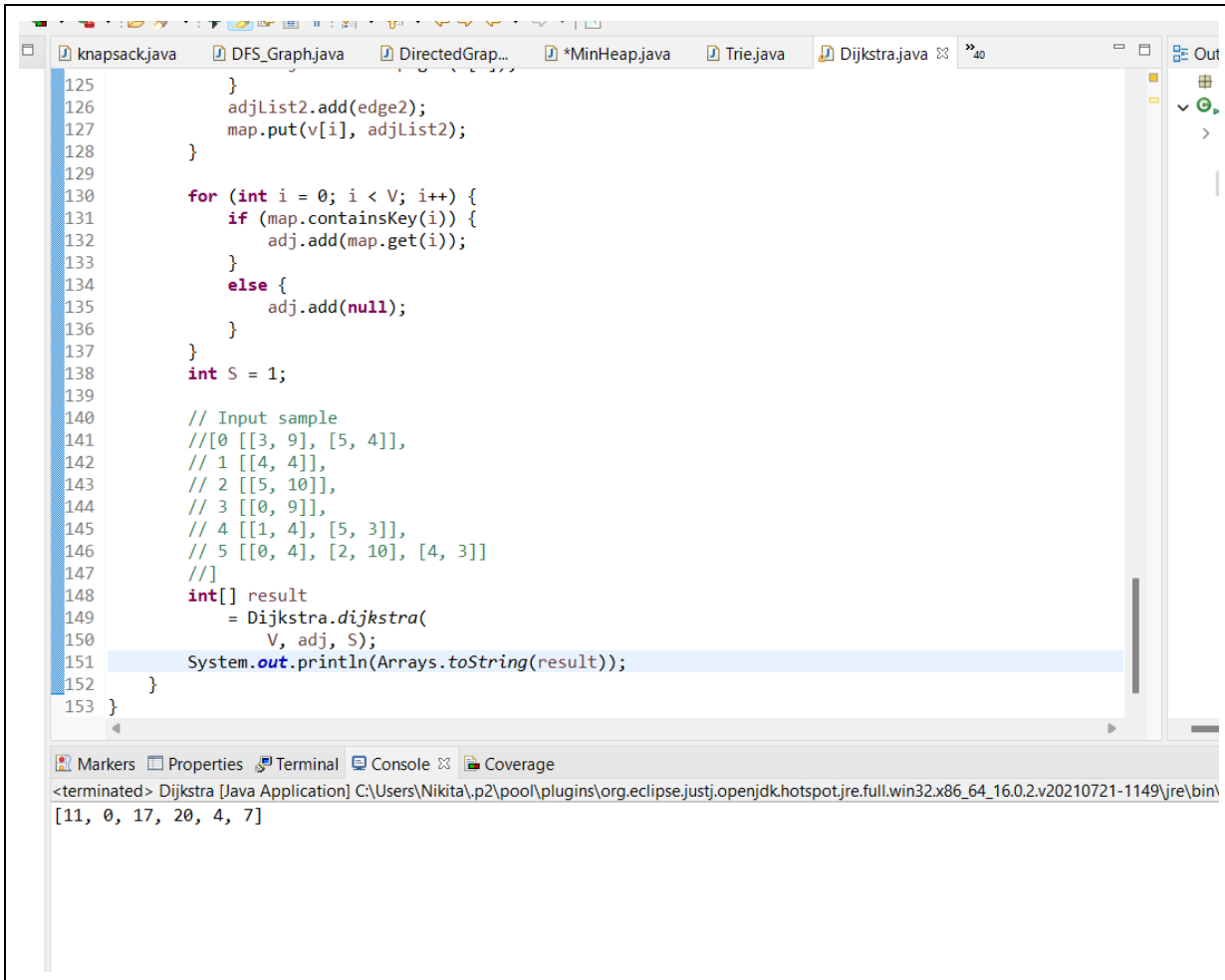
Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights.

```
package Day9_10;

import java.util.*;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.PriorityQueue;

public class Dijkstra {

    static class Node implements Comparable<Node> {
        int v;
        int distance;

        public Node(int v, int distance)
        {
            this.v = v;
            this.distance = distance;
        }

        @Override public int compareTo(Node n)
        {
            if (this.distance <= n.distance) {
                return -1;
            }
            else {
                return 1;
            }
        }
    }

    static int[] dijkstra(
        int V,
        ArrayList<ArrayList<ArrayList<Integer> > > adj,
        int S)
    {
        boolean[] visited = new boolean[V];
        HashMap<Integer, Node> map = new HashMap<>();
        PriorityQueue<Node> q = new PriorityQueue<>();

        map.put(S, new Node(S, 0));
        q.add(new Node(S, 0));

        while (!q.isEmpty()) {
            Node n = q.poll();
            int v = n.v;
            int distance = n.distance;
            visited[v] = true;

            ArrayList<ArrayList<Integer> > adjList
                = adj.get(v);
            for (ArrayList<Integer> adjLink : adjList) {

                if (visited[adjLink.get(0)] == false) {
                    if (!map.containsKey(adjLink.get(0))) {
```

```
                        map.put(
                            adjLink.get(0),
                            new Node(v,
                                     distance
                                         + adjLink.get(1)));
                    }
                    else {
                        Node sn = map.get(adjLink.get(0));
                        if (distance + adjLink.get(1)
                            < sn.distance) {
                            sn.v = v;
                            sn.distance
                                = distance + adjLink.get(1);
                        }
                    }
                    q.add(new Node(adjLink.get(0),
                                   distance
                                       + adjLink.get(1)));
                }
            }
        }

        int[] result = new int[V];
        for (int i = 0; i < V; i++) {
            result[i] = map.get(i).distance;
        }

        return result;
    }

    public static void main(String[] args)
    {
        ArrayList<ArrayList<ArrayList<Integer> > > adj
            = new ArrayList<>();
        HashMap<Integer, ArrayList<ArrayList<Integer> > >
            map = new HashMap<>();

        int V = 6;
        int E = 5;
        int[] u = { 0, 0, 1, 2, 4 };
        int[] v = { 3, 5, 4, 5, 5 };
        int[] w = { 9, 4, 4, 10, 3 };

        for (int i = 0; i < E; i++) {
            ArrayList<Integer> edge = new ArrayList<>();
            edge.add(v[i]);
            edge.add(w[i]);

            ArrayList<ArrayList<Integer> > adjList;
            if (!map.containsKey(u[i])) {
                adjList = new ArrayList<>();
            }
            else {
                adjList = map.get(u[i]);
            }
            adjList.add(edge);
            map.put(u[i], adjList);

            ArrayList<Integer> edge2 = new ArrayList<>();
```

```
            edge2.add(u[i]);
            edge2.add(w[i]);

            ArrayList<ArrayList<Integer> > adjList2;
            if (!map.containsKey(v[i])) {
                adjList2 = new ArrayList<>();
            }
            else {
                adjList2 = map.get(v[i]);
            }
            adjList2.add(edge2);
            map.put(v[i], adjList2);
        }

        for (int i = 0; i < V; i++) {
            if (map.containsKey(i)) {
                adj.add(map.get(i));
            }
            else {
                adj.add(null);
            }
        }
        int S = 1;

        // Input sample
        //[0 [[3, 9], [5, 4]],
        // 1 [[4, 4]],
        // 2 [[5, 10]],
        // 3 [[0, 9]],
        // 4 [[1, 4], [5, 3]],
        // 5 [[0, 4], [2, 10], [4, 3]]
        //]
        int[] result
            = Dijkstra.dijkstra(
                V, adj, S);
        System.out.println(Arrays.toString(result));
    }
}
```


knapsack.java
DFS_Graph.java
DirectedGrap...
*MinHeap.java
Trie.java
Dijkstra.java
Out
}
adjList2.add(edge2);
map.put(v[i], adjList2);
}
for (int i = 0; i < V; i++) {
if (map.containsKey(i)) {
adj.add(map.get(i));
}
else {
adj.add(null);
}
}
int S = 1;
// Input sample
//[0 [[3, 9], [5, 4]],
// 1 [[4, 4]],
// 2 [[5, 10]],
// 3 [[0, 9]],
// 4 [[1, 4], [5, 3]],
// 5 [[0, 4], [2, 10], [4, 3]]
//]
int[] result
= Dijkstra.dijkstra(
V, adj, S);
System.out.println(Arrays.toString(result));
}
}
Markers
Properties
Terminal
Console
Coverage
<terminated> Dijkstra [Java Application] C:\Users\Nikita\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\
[11, 0, 17, 20, 4, 7]