



# Dijkstra: Shortest Reach 2



by pranav9413

Problem

Submissions

Leaderboard

Discussions

Editorial

Given a graph consisting  $N$  nodes (labelled  $1$  to  $N$ ) where a specific given node  $S$  represents the starting position  $S$  and an edge between two nodes is of a given length, which may or may not be equal to other lengths in the graph.

It is required to calculate the shortest distance from the start position (Node  $S$ ) to all of the other nodes in the graph.

**Note:** If a node is unreachable, the distance is assumed as  $-1$ .

## Input Format

The first line contains  $T$ , denoting the number of test cases.

First line of each test case has two integers  $N$ , denoting the number of nodes in the graph and  $M$ , denoting the number of edges in the graph.

The next  $M$  lines each consist of three space-separated integers  $x y r$ , where  $x$  and  $y$  denote the two nodes between which the **undirected** edge exists,  $r$  denotes the length of edge between these corresponding nodes.

The last line has an integer  $S$ , denoting the starting position.

## Constraints

$$1 \leq T \leq 10$$

$$2 \leq N \leq 3000$$

$$1 \leq M \leq \frac{N \times (N-1)}{2}$$

$$1 \leq x, y, S \leq N$$

$$1 \leq r \leq 10^5$$

If there are edges between the same pair of nodes with different weights, they are to be considered as is, like multiple edges.

## Output Format

For each of the  $T$  test cases, print a single line consisting  $N - 1$  space separated integers denoting the shortest distance of  $N - 1$  nodes other than  $S$  from starting position  $S$  in increasing order of their labels.

For unreachable nodes, print  $-1$ .

## Sample Input

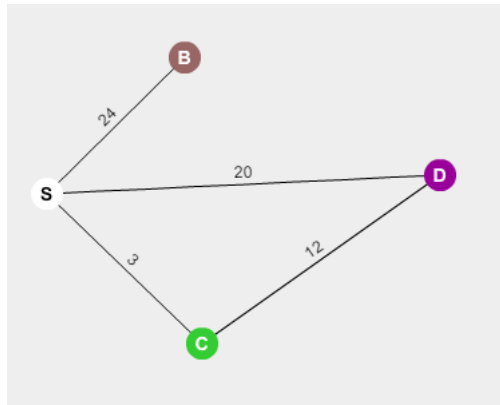
```
1
4 4
1 2 24
1 4 20
3 1 3
4 3 12
1
```

## Sample Output

```
24 3 15
```

## Explanation

The graph given in the test case is shown as :



- The straight line is a weighted edge, denoting length of edge between the corresponding nodes.
- The nodes S,B,C and D denote the obvious node 1,2,3 and 4 in the test case.

The shortest paths followed for the three nodes B,C and D are as follows :

**S->B** - Shortest Path Value : **24**

**S->C** - Shortest Path Value : **3**

**S->C->D** - Shortest Path Value : **15**

f t in

Submissions: 13871


Max Score: 60

Difficulty: Hard

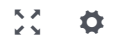
Rate This Challenge:

☆☆☆☆☆

[More](#)

Current Buffer (saved locally, editable)  

Java 8



```

1 import java.io.*;
2 import java.util.*;
3 import java.text.*;
4 import java.math.*;
5 import java.util.regex.*;
6
7
8 class BinaryMinHeap{
9
10     private List<Node> allNodes = new ArrayList<>();
11     private Map<Integer,Integer> nodePosition = new HashMap<>();
12
13     public BinaryMinHeap(){
14
15     }
16
17
18     public boolean containsData(int key){
19         return nodePosition.containsKey(key);
20     }
21
22     public void add(int weight, int key){
23
24         Node node = new Node();
25
26         node.weight = weight;
27         node.key = key;
28         allNodes.add(node);
29

```

```

30     int nodeSize = allNodes.size();
31     int currentPosition = nodeSize - 1;
32
33     int parentIndex = (currentPosition - 1) / 2;
34     nodePosition.put(node.key, currentPosition);
35
36     while(parentIndex >= 0){
37
38         Node parentNode = allNodes.get(parentIndex);
39         Node currentNode = allNodes.get(currentPosition);
40
41         if(parentNode.weight > currentNode.weight){
42             swap(parentNode,currentNode);
43             updatePositionMap(parentNode.key,currentNode.key,parentIndex,currentPosition);
44             currentPosition = parentIndex;
45             parentIndex = (parentIndex-1)/2;          }
46         else{
47             break;
48         }
49     }
50 }
51
52
53 public int min(){
54     return allNodes.get(0).key;
55 }
56
57 public boolean empty(){
58     return allNodes.size() == 0;
59 }
60
61 public void decrease(int data, int newWeight){
62     Integer position = nodePosition.get(data);
63     allNodes.get(position).weight = newWeight;
64     int parent = (position - 1 )/2;
65     while(parent >= 0){
66         if(allNodes.get(parent).weight > allNodes.get(position).weight){
67             swap(allNodes.get(parent), allNodes.get(position));
68             updatePositionMap(allNodes.get(parent).key,allNodes.get(position).key,parent,position);
69             position = parent;
70             parent = (parent-1)/2;
71         }else{
72             break;
73         }
74     }
75 }
76
77 public Integer getWeight(int key) {
78     Integer position = nodePosition.get(key);
79     if( position == null ) {
80         return null;
81     } else {
82         return allNodes.get(position).weight;
83     }
84 }
85
86
87 public Node extractMinNode() {
88     int size = allNodes.size() -1;
89     Node minNode = new Node();
90     minNode.key = allNodes.get(0).key;
91     minNode.weight = allNodes.get(0).weight;
92
93     int lastNodeWeight = allNodes.get(size).weight;
94     allNodes.get(0).weight = lastNodeWeight;
95     allNodes.get(0).key = allNodes.get(size).key;
96     nodePosition.remove(minNode.key);
97     nodePosition.remove(allNodes.get(0));
98     nodePosition.put(allNodes.get(0).key, 0);
99     allNodes.remove(size);
100
101     int currentIndex = 0;
102     size--;

```

```

103     while(true){
104         int left = 2*currentIndex + 1;
105         int right = 2*currentIndex + 2;
106         if(left > size){
107             break;
108         }
109         if(right > size){
110             right = left;
111         }
112         int smallerIndex = allNodes.get(left).weight <= allNodes.get(right).weight ? left : right;
113         if(allNodes.get(currentIndex).weight > allNodes.get(smallerIndex).weight){
114             swap(allNodes.get(currentIndex), allNodes.get(smallerIndex));
115         }
116         updatePositionMap(allNodes.get(currentIndex).key,allNodes.get(smallerIndex).key,currentIndex,smallerIndex);
117         currentIndex = smallerIndex;
118     }else{
119         break;
120     }
121     }
122     return minNode;
123 }
124
125 public int extractMin(){
126     Node node = extractMinNode();
127     return node.key;
128 }
129
130
131 private void swap(Node node1,Node node2){
132
133     int weight = node1.weight;
134     int data = node1.key;
135
136     node1.key = node2.key;
137     node1.weight = node2.weight;
138
139     node2.key = data;
140     node2.weight = weight;
141 }
142
143
144 private void updatePositionMap(int data1, int data2, int pos1, int pos2){
145     nodePosition.remove(data1);
146     nodePosition.remove(data2);
147     nodePosition.put(data1, pos1);
148     nodePosition.put(data2, pos2);
149 }
150 }
151
152
153 class Node{
154
155     int weight;
156     int key;
157
158     public Node(){
159
160     }
161
162 }
163
164
165 class Graph{
166
167     private int V;
168     private LinkedList<Map<Integer,Integer>> adj[];
169
170     public Graph(int V){
171         this.V = V;
172         adj = new LinkedList[V];
173
174         for(int i = 0 ; i < V ; i++){

```

```

175     adj[i] = new LinkedList<Map<Integer,Integer>>();
176 }
177
178 }
179
180 public void addEdge(int srcVertex, int destVertex, int weight){
181
182     Map<Integer,Integer> srcToDest = new HashMap<Integer,Integer>();
183     Map<Integer,Integer> destToSrc = new HashMap<Integer,Integer>();
184
185     srcToDest.put(destVertex,weight);
186     destToSrc.put(srcVertex,weight);
187
188     adj[srcVertex].add(srcToDest);
189     adj[destVertex].add(destToSrc);
190
191 }
192
193 public Map<Integer,Integer> doBFS(int source){
194
195     Map<Integer,Integer> output = new HashMap<Integer,Integer>();
196
197     BinaryMinHeap minHeap = new BinaryMinHeap();
198
199     for(int i = 0 ; i < V ; i++){
200
201         if(i != source){
202             minHeap.add(20000000,i);
203             output.put(i,20000000);
204         }
205         else{
206             minHeap.add(0,i);
207             output.put(i,0);
208         }
209     }
210
211
212
213     while(!minHeap.empty()){
214
215         Node node = minHeap.extractMinNode();
216
217         Iterator<Map<Integer,Integer>> i = adj[node.key].listIterator();
218
219         while(i.hasNext()){
220
221             Map<Integer,Integer> val = i.next();
222
223             List<Integer> l = new ArrayList<Integer>(val.keySet());
224
225             if(minHeap.containsData(l.get(0))){
226
227                 if(minHeap.getWeight(l.get(0)) > node.weight + val.get(l.get(0))){
228                     minHeap.decrease(l.get(0),node.weight + val.get(l.get(0)));
229                     output.put(l.get(0),node.weight + val.get(l.get(0)));
230                 }
231             }
232         }
233     }
234
235 }
236
237 return output;
238
239 }
240
241 }
242
243 }
244
245
246 public class Solution {
247

```

```

248 public static void main(String[] args) throws IOException{
249     //1
250     //4 4
251     //1 2 24
252     //1 4 20
253     //3 1 3
254     //4 3 12
255     //1
256
257     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
258     int t = Integer.parseInt(br.readLine());
259
260     for(int a0 = 0; a0 < t; a0++){
261
262         String line = br.readLine();
263         String[] numbers = line.split("\\s");
264
265         int n = Integer.parseInt(numbers[0]);
266         int m = Integer.parseInt(numbers[1]);
267
268         Graph graph = new Graph(n);
269
270         HashMap<String,Integer> myMap = new HashMap<String,Integer>();
271
272         for(int a1 = 0; a1 < m; a1++){
273
274             line = br.readLine();
275             numbers = line.split("\\s");
276
277             int x = Integer.parseInt(numbers[0]);
278             int y = Integer.parseInt(numbers[1]);
279             int r = Integer.parseInt(numbers[2]);
280
281             if(myMap.containsKey(x + "-" + y)){
282                 int mapVal = myMap.get(x + "-" + y);
283                 if(r < mapVal){
284                     graph.addEdge((x - 1),(y - 1),r);
285                 }
286             }
287             else{
288                 myMap.put(x + "-" + y,r);
289                 myMap.put(y + "-" + x,r);
290                 graph.addEdge((x - 1),(y - 1),r);
291             }
292
293
294
295         }
296         int s = Integer.parseInt(br.readLine());
297
298         Map<Integer,Integer> map = graph.doBFS(s - 1);
299
300         for(int o : map.values()){
301
302
303
304             if(o == 20000000){
305                 System.out.print("-1 ");
306             }
307             else if(o == 0){
308                 continue;
309             }
310             else{
311                 System.out.print(o + " ");
312             }
313
314         }
315
316         System.out.println("");
317
318     }
319 }
320 }

```

8/29/2017

Dijkstra: Shortest Reach 2 | Algorithms Question | HackerRank

321

Line: 1 Col: 1

 Upload Code as File

☐ Test against custom input

Run Code

Submit Code

Copyright © 2017 HackerRank. All Rights Reserved

Join us on IRC at [#hackerrank](#) on freenode for hugs or bugs.

[Contest Calendar](#) | [Interview Prep](#) | [Blog](#) | [Scoring](#) | [Environment](#) | [FAQ](#) | [About Us](#) | [Support](#) | [Careers](#) | [Terms Of Service](#) | [Privacy Policy](#) | [Request a Feature](#)