CSE 551 - Foundations of Algorithms

Assignment 3

Submitted by: Sannidhya Pathania (spathan3)

Solution 1:

1.1.

1.1.1. True, the new MST T must still be a minimum spanning tree for this new instance.

1.1.2. The edge cost before modification is positive and distinct. So, even if these are replaced with their squared values, the order of edges will still remain the same. For example, if we assume edges were 3,4,6,8,9,10 before and later after squaring the edges will again be in the same order as before. 9,16,36,64, 81, 100. So, the MST will remain the same in both cases i.e. 'T'. The algorithm which was used earlier to create MST T will again create the same MST with the same edges but with squared values.

1.2.

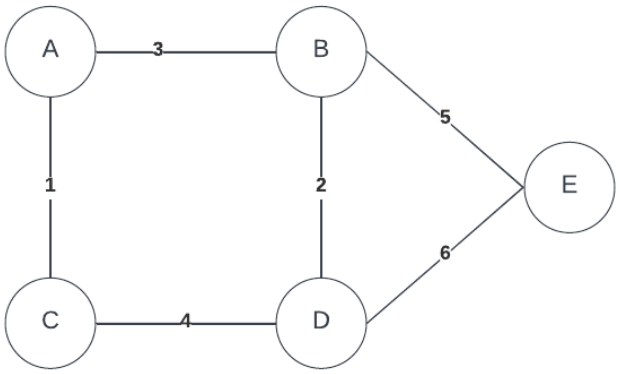
1.2.1. False, the new path P will not still be a minimum-cost s-t path for this new instance.

1.2.2. Consider a graph G with three edges: (s,v), (v,t), and (s,t), where the first two edges have a cost of 3, and the last edge has a cost of 5. Although the shortest path in G between nodes s and t is the single edge (s,t), after squaring the edge costs, the shortest path in the graph would pass through node v instead.

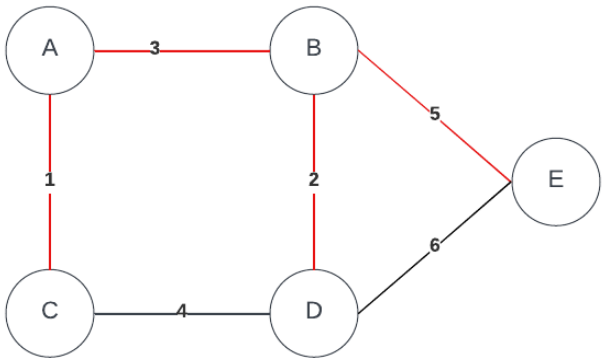
Solution 2:

2.1

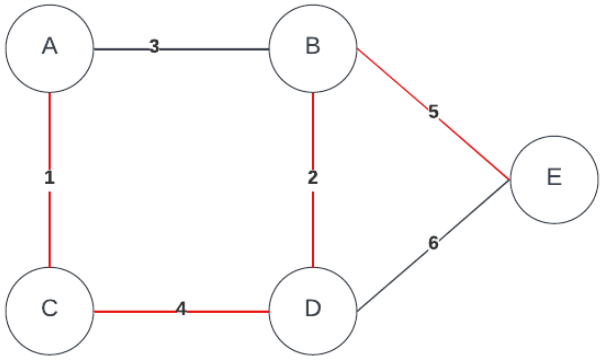
No, every minimum-bottleneck tree of G is not a minimum-spanning tree of G. For example let's take the following example of graph G with vertices as A, B, C, D, E.



The Minimum spanning tree for the given graph is as shown with a cost of 11. As we can see this is a minimum-bottleneck tree as well with a bottleneck edge of 5.



Now, consider the following spanning tree. This is also a minimum-bottleneck tree since the bottleneck edge is the same as the previous i.e 5. But, it is not a minimum spanning tree. Its cost is 12, which is greater than the previous spanning tree.



So, the claim that every minimum-bottleneck tree of G is a minimum-spanning tree of G, is **False**.

2.2

Yes, every minimum-spanning tree of G is a minimum-bottleneck tree of G.

Let T be the minimum spanning tree (MST) and B be the minimum bottleneck tree for the graph G = (V, E).

- Let us assume two vertices x and y in T. Let X be the set of vertices in G that can be reached from x without going through y. Similarly, let Y be the set of vertices in G that can be reached from y without going through x.
- Because G is a connected graph, we need an edge from x to y to connect the T.
- Since T is an MST, the edge between x and y must be a minimum edge for this connection. We started with set X and Y in such a way that the only minimum edge that should be possible should be x-y.
- But there's another edge according to the minimum bottleneck tree that has a smaller edge than x-y.
- This is a contradiction and both the spanning trees should refer to the same edge x-y. This means T is a minimum bottleneck spanning tree for graph G.

Solution 3:

3.1

Let V = the number of vertices and E = the number of edges.

The below proof is done by assuming we are **using an adjacency list.**

Begin of Algorithm

Let S be a set of explored nodes (vertices) so far; ---------- O(1)

//will take constant time because we are just defining the set.

Define the data structure for each u ∈ S, and store the earliest time d(u) when we can arrive at u;

//will add the distance for every vertex that this present as infinity. ------- V

*//if we are using a **priority queue as an additional** data structure to get the minimum later in the algorithm then it will take extra log V to put every vertex in the heap, i.e. total **VlogV**.*

Initialize S = {s}, d(s) = 0, where d(u) is the earliest time from s to u, d(u) = 0 when u = s;

------- O(1)

While S != V *//will run till the size of Set S equals the total number of vertices V.*

Select a node v ∉ S, with at least one edge from S, for which

*//If we are not using a data structure to get the minimum next vertex then this will go through at most V vertices to find the minimum. Otherwise, if we are using a priority queue as our data structure then it will take log V to extract the minimum. And then updating the neighboring edges will take at most E time in both cases assuming we are using **an adjacency list.** So time complexity without the heap is O(EV) and with the heap is O(E log(V)).*

$d'(v) = \min_{c=(u,v):u\in S} f_e(d(u))$ *// f_e function will take O(1) time*

is as small as possible. ---------- EV

Add v to S and define d(v) – d'(v) ------- O(1)

Endwhile

Define function $f_e(u)$, which generates a random number, where $f_e(t) \geq t$ for all edges e and all times t, and that $f_e(t)$ is a monotone increasing function of t, that is, you do not arrive earlier by starting later. ---------- O(1)

End of Algorithm

Without heap:

$T(E,V) = c_1(1) + c_2(V) + c_3(1) + c_4(EV) + c_5(1) + c_6(1)$

$= c_{123456}(1+V+1+EV+1+1) = c_{123456}(4+V+ EV) \Rightarrow O(EV)$.

With heap: Instead of EV it will take E log V time and overall an extra V log V time.

$T(E,V) = c_1(1) + c_2(V) + c_3(1) + c_4(E \log V) + c_5(1) + c_6(1) + c_7(V \log V)$

$= c_{1234567}(1+V+1+E \log V+1+1+V \log V) = c_{1234567}(4+V+ E \log V + V \log V)$

$\Rightarrow O(E \log V)$ since $E >= V-1$ as we are considering a connected graph.

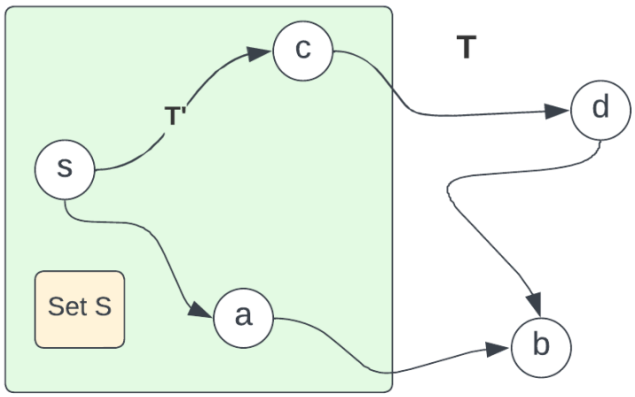
3.2

Let S be the set of explored vertices. And |S| be the size of the set.

The invariant is for every node $u \in S$, the shortest path from s to u i.e s-u is d(u). All edges are positive as evident from the question description.

Now for |S| = 1, i.e only one node is present. This is a trivial case and will be true. Since all the edges are positive, so, the minimum distance from s to s will be 0, where s is the starting node or source node.

Assuming the algorithm is true for |S| = k, where k>=1. Also, we are updating the new arrival time obtained from $f_e(t)$ function for the nodes that are reachable from the current set of nodes in S. And then in the next step we are getting the minimum time from all reachable nodes.



Now for |S| = k+1,

- Let b be the next node that is to be added to S and let a-b be the chosen time.
- Now the shortest s-a time plus a-b is the s-b time i.e. d'(b). Here d'(b) is considering the delay in time due to weather as assumed earlier (because the time of these nodes was updated in the last iteration using the $f_e(t)$ function).

- Consider any s-b time T.
- Let c-d be the first time in T that leaves S and let T' be the subpath to c.
- T is already too long as soon as it leaves S.
    - t(T) >= t(T') + t(c,d) >= d(c) + t(c, d) >= d'(d) >= d'(b)
- Thus we observed that T is longer than d'(b) and the chosen d'(b) is the shortest time we can get. So, the algorithm is true for |S| = k+1 as well.
- Hence, proved.

3.3

IDE used = Visual Studio Code

```
import java.util.HashMap;
import java.util.PriorityQueue;
import java.util.Random;
import java.util.Scanner;

public class DijkstraWithRandomWeight {
   public static void main(String[] args) throws Exception {
       Scanner sc = new Scanner(System.in);

       System.out.println("Enter the number of vertices");
       int V = sc.nextInt();

       HashMap<Integer, Integer>[] graph = new HashMap[V];

       System.out.println("Enter the number of edges");
       int E = sc.nextInt();

       System.out.println("Enter the beginning point then the end point and then the
time needed to travel");
       while(E>0){
           int src = sc.nextInt()-1;
           int dest = sc.nextInt()-1;
           int cost = sc.nextInt();
           if(graph[src]==null)
               graph[src] = new HashMap<>();
           graph[src].put(dest, cost);
           E--;
       }

       System.out.println("Enter the starting point and destination");
       int src = sc.nextInt()-1;
```

```
        int dest = sc.nextInt()-1;

        //Map to retain the final path taken for each node as well as the time
        HashMap<Integer, Route> result = new HashMap<>();
        //Array for maintaining the distance for a node.
        int[] distance = new int[V];

        //Calling modified dijkstra algorithm
        getShortestPath(graph, result, src, distance);

        System.out.println("Time needed to travel to destination: "+
result.get(dest).cost + "\nPath taken would be "+ result.get(dest).path);

        System.out.println("All nodes shortest time as below");
        for(int node : result.keySet()){
            System.out.println((node+1)+": time: "+ result.get(node).cost + "\tPath: "+
result.get(node).path);
        }
    }
    private static void getShortestPath(HashMap<Integer, Integer>[] graph,
HashMap<Integer, Route> result, int src, int[] distance) {
        PriorityQueue<Route> heap = new PriorityQueue<>((a,b)->(a.cost - b.cost));
        //Adding all the vertices into heap.
        for(int i=0; i<graph.length; i++){
            if(i==src)
                heap.add(new Route("", 0, src));
            else{
                distance[i] = Integer.MAX_VALUE;
                heap.add(new Route("", Integer.MAX_VALUE, i));
            }
        }
        while(result.size()!=graph.length){
            //retrieving the node with minimum time to reach.
            Route minRoute = heap.poll();

            if(result.containsKey(minRoute.node))
                continue;

            minRoute.path += (minRoute.node+1);
            result.put(minRoute.node, minRoute);
            HashMap<Integer, Integer> current = graph[minRoute.node];

            if(current == null)
                continue;
```

```
            //Getting the neighbors and updating the time for each node.
            for(int adj : current.keySet()){
                int expected = minRoute.cost + current.get(adj);
                //Calling the function that will provide the new time after considering
the weather.
                int delayed = getDelay(minRoute.node, adj, expected);
                int earlier = distance[adj];

                if(earlier > delayed){
                    distance[adj] = delayed;
                    heap.add(new Route(minRoute.path, delayed, adj));
                }
                //Here we are updating the previous time with the new time after
considering weather conditions
                graph[minRoute.node].put(adj, delayed - minRoute.cost);
                System.out.println("The distance between nodes "+(minRoute.node+1) + "
and "+ (adj+1) +
                " has been changed and updated considering the weather from " +
                (expected - minRoute.cost) +" to "+ (delayed - minRoute.cost));
            }
            printGraph(graph);
            System.out.println("Time in which nodes can be reached");
            for(int i=0; i<distance.length; i++){
                System.out.print((i+1) +":");
                if(distance[i]!=Integer.MAX_VALUE)
                    System.out.print(distance[i]+ "\t");
                else
                    System.out.print("infinite\t");
            }
            System.out.println();
        }
    }
    public static int getDelay(int src, int dest, int time){
        Random r = new Random();
        int low = time;
        int high = time + 20;
        int newArrival = r.nextInt(high-low) + low;
        return newArrival;
    }
    public static void printGraph(HashMap<Integer, Integer>[] graph){
        System.out.println("New graph edges will be as follows");
        for(int i = 0; i<graph.length; i++){
            if(graph[i]!= null)
                for(int adj: graph[i].keySet()){
```

```
                    System.out.println("Edge: "+(i+1)+ "-"+(adj+1)+" (" +
graph[i].get(adj)+ ") ");
                }
        }
    }
}
class Route {
    String path;
    int cost;
    int node;
    Route(String p, int c, int n){
        this.path = p;
        this.cost = c;
        this.node = n;
    }
}
```

3.4

Output from Code:

Enter the number of vertices

8

Enter the number of edges

15

Enter the beginning point then the end point and then the time needed to travel

1 2 9

1 6 14

1 7 15

2 3 23

3 5 2

3 8 19

4 3 6

4 8 6

5 4 11

5 8 16

6 3 18

6 5 30

6 7 5

7 5 20

7 8 44

Enter the starting point and destination

1 8

The distance between nodes 1 and 2 has been changed and updated considering the weather from 9 to 21

The distance between nodes 1 and 6 has been changed and updated considering the weather from 14 to 24

The distance between nodes 1 and 7 has been changed and updated considering the weather from 15 to 23

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (23)

Edge: 3-5 (2)

Edge: 3-8 (19)

Edge: 4-3 (6)

Edge: 4-8 (6)

Edge: 5-4 (11)

Edge: 5-8 (16)

Edge: 6-3 (18)

Edge: 6-5 (30)

Edge: 6-7 (5)

Edge: 7-5 (20)

Edge: 7-8 (44)

Time in which nodes can be reached

1:0 2:21 3:infinite 4:infinite 5:infinite 6:24 7:23 8:infinite

The distance between nodes 2 and 3 has been changed and updated considering the weather from 23 to 36

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (36)

Edge: 3-5 (2)

Edge: 3-8 (19)

Edge: 4-3 (6)

Edge: 4-8 (6)

Edge: 5-4 (11)

Edge: 5-8 (16)

Edge: 6-3 (18)

Edge: 6-5 (30)

Edge: 6-7 (5)

Edge: 7-5 (20)

Edge: 7-8 (44)

Time in which nodes can be reached

1:0 2:21 3:57 4:infinite 5:infinite 6:24 7:23 8:infinite

The distance between nodes 7 and 5 has been changed and updated considering the weather from 20 to 36

The distance between nodes 7 and 8 has been changed and updated considering the weather from 44 to 58

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (36)

Edge: 3-5 (2)

Edge: 3-8 (19)

Edge: 4-3 (6)

Edge: 4-8 (6)

Edge: 5-4 (11)

Edge: 5-8 (16)

Edge: 6-3 (18)

Edge: 6-5 (30)

Edge: 6-7 (5)

Edge: 7-5 (36)

Edge: 7-8 (58)

Time in which nodes can be reached

1:0 2:21 3:57 4:infinite 5:59 6:24 7:23 8:81

The distance between nodes 6 and 3 has been changed and updated considering the weather from 18 to 23

The distance between nodes 6 and 5 has been changed and updated considering the weather from 30 to 45

The distance between nodes 6 and 7 has been changed and updated considering the weather from 5 to 15

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (36)

Edge: 3-5 (2)

Edge: 3-8 (19)

Edge: 4-3 (6)

Edge: 4-8 (6)

Edge: 5-4 (11)

Edge: 5-8 (16)

Edge: 6-3 (23)

Edge: 6-5 (45)

Edge: 6-7 (15)

Edge: 7-5 (36)

Edge: 7-8 (58)

Time in which nodes can be reached

1:0 2:21 3:47 4:infinite 5:59 6:24 7:23 8:81

The distance between nodes 3 and 5 has been changed and updated considering the weather from 2 to 11

The distance between nodes 3 and 8 has been changed and updated considering the weather from 19 to 38

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (36)

Edge: 3-5 (11)

Edge: 3-8 (38)

Edge: 4-3 (6)

Edge: 4-8 (6)

Edge: 5-4 (11)

Edge: 5-8 (16)

Edge: 6-3 (23)

Edge: 6-5 (45)

Edge: 6-7 (15)

Edge: 7-5 (36)

Edge: 7-8 (58)

Time in which nodes can be reached

1:0 2:21 3:47 4:infinite 5:58 6:24 7:23 8:81

The distance between nodes 5 and 4 has been changed and updated considering the weather from 11 to 17

The distance between nodes 5 and 8 has been changed and updated considering the weather from 16 to 26

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (36)

Edge: 3-5 (11)

Edge: 3-8 (38)

Edge: 4-3 (6)

Edge: 4-8 (6)

Edge: 5-4 (17)

Edge: 5-8 (26)

Edge: 6-3 (23)

Edge: 6-5 (45)

Edge: 6-7 (15)

Edge: 7-5 (36)

Edge: 7-8 (58)

Time in which nodes can be reached

1:0 2:21 3:47 4:75 5:58 6:24 7:23 8:81

The distance between nodes 4 and 3 has been changed and updated considering the weather from 6 to 6

The distance between nodes 4 and 8 has been changed and updated considering the weather from 6 to 13

New graph edges will be as follows

Edge: 1-2 (21)

Edge: 1-6 (24)

Edge: 1-7 (23)

Edge: 2-3 (36)

Edge: 3-5 (11)

Edge: 3-8 (38)

Edge: 4-3 (6)

Edge: 4-8 (13)

Edge: 5-4 (17)

Edge: 5-8 (26)

Edge: 6-3 (23)

Edge: 6-5 (45)

Edge: 6-7 (15)

Edge: 7-5 (36)

Edge: 7-8 (58)

Time in which nodes can be reached

1:0 2:21 3:47 4:75 5:58 6:24 7:23 8:81

Time needed to travel to destination: 81

Path taken would be 178

All nodes shortest time as below

1: time: 0 Path: 1

2: time: 21 Path: 12

3: time: 47 Path: 163

4: time: 75 Path: 16354

5: time: 58 Path: 1635

6: time: 24 Path: 16

7: time: 23 Path: 17

8: time: 81 Path: 178

Screenshot of output:

```
spats@Sannidhyas-Air Question_3 %  /usr/bin/env /Library/Java/JavaVirtualMachines/amazon-corretto-19.jdk/
eDetailsInExceptionMessages -cp /Users/spats/Desktop/ASU/Sem\ 2/CSE\ 551\ FOA/Assignments/AS3/Question_3/
Enter the number of vertices
8
Enter the number of edges
15
Enter the beginning point then the end point and then the time needed to travel
1 2 9
1 6 14
1 7 15
2 3 23
3 5 2
3 8 19
4 3 6
4 8 6
5 4 11
5 8 16
6 3 18
6 5 30
6 7 5
7 5 20
7 8 44
Enter the starting point and destination
1 8
The distance between nodes 1 and 2 has been changed and updated considering the weather from 9 to 21
The distance between nodes 1 and 6 has been changed and updated considering the weather from 14 to 24
The distance between nodes 1 and 7 has been changed and updated considering the weather from 15 to 23
New graph edges will be as follows
Edge: 1-2 (21)
Edge: 1-6 (24)
Edge: 1-7 (23)
Edge: 2-3 (23)
Edge: 3-5 (2)
Edge: 3-8 (19)
Edge: 4-3 (6)
Edge: 4-8 (6)
Edge: 5-4 (11)
Edge: 5-8 (16)
Edge: 6-3 (18)
Edge: 6-5 (30)
Edge: 6-7 (5)
Edge: 7-5 (20)
Edge: 7-8 (44)
Time in which nodes can be reached
1:0     2:21    3:infinite      4:infinite      5:infinite      6:24    7:23    8:infinite
```

```
The distance between nodes 2 and 3 has been changed and updated considering the weather from 23 to 36
New graph edges will be as follows
Edge: 1–2 (21)
Edge: 1–6 (24)
Edge: 1–7 (23)
Edge: 2–3 (36)
Edge: 3–5 (2)
Edge: 3–8 (19)
Edge: 4–3 (6)
Edge: 4–8 (6)
Edge: 5–4 (11)
Edge: 5–8 (16)
Edge: 6–3 (18)
Edge: 6–5 (30)
Edge: 6–7 (5)
Edge: 7–5 (20)
Edge: 7–8 (44)
Time in which nodes can be reached
1:0     2:21    3:57    4:infinite      5:infinite      6:24    7:23    8:infinite
The distance between nodes 7 and 5 has been changed and updated considering the weather from 20 to 36
The distance between nodes 7 and 8 has been changed and updated considering the weather from 44 to 58
New graph edges will be as follows
Edge: 1–2 (21)
Edge: 1–6 (24)
Edge: 1–7 (23)
Edge: 2–3 (36)
Edge: 3–5 (2)
Edge: 3–8 (19)
Edge: 4–3 (6)
Edge: 4–8 (6)
Edge: 5–4 (11)
Edge: 5–8 (16)
Edge: 6–3 (18)
Edge: 6–5 (30)
Edge: 6–7 (5)
Edge: 7–5 (36)
Edge: 7–8 (58)
Time in which nodes can be reached
1:0     2:21    3:57    4:infinite      5:59    6:24    7:23    8:81
```

```
The distance between nodes 6 and 3 has been changed and updated considering the weather from 18 to 23
The distance between nodes 6 and 5 has been changed and updated considering the weather from 30 to 45
The distance between nodes 6 and 7 has been changed and updated considering the weather from 5 to 15
New graph edges will be as follows
Edge: 1–2 (21)
Edge: 1–6 (24)
Edge: 1–7 (23)
Edge: 2–3 (36)
Edge: 3–5 (2)
Edge: 3–8 (19)
Edge: 4–3 (6)
Edge: 4–8 (6)
Edge: 5–4 (11)
Edge: 5–8 (16)
Edge: 6–3 (23)
Edge: 6–5 (45)
Edge: 6–7 (15)
Edge: 7–5 (36)
Edge: 7–8 (58)
Time in which nodes can be reached
1:0     2:21    3:47    4:infinite      5:59    6:24    7:23    8:81
The distance between nodes 3 and 5 has been changed and updated considering the weather from 2 to 11
The distance between nodes 3 and 8 has been changed and updated considering the weather from 19 to 38
New graph edges will be as follows
Edge: 1–2 (21)
Edge: 1–6 (24)
Edge: 1–7 (23)
Edge: 2–3 (36)
Edge: 3–5 (11)
Edge: 3–8 (38)
Edge: 4–3 (6)
Edge: 4–8 (6)
Edge: 5–4 (11)
Edge: 5–8 (16)
Edge: 6–3 (23)
Edge: 6–5 (45)
Edge: 6–7 (15)
Edge: 7–5 (36)
Edge: 7–8 (58)
Time in which nodes can be reached
1:0     2:21    3:47    4:infinite      5:58    6:24    7:23    8:81
```

```
The distance between nodes 5 and 4 has been changed and updated considering the weather from 11 to 17
The distance between nodes 5 and 8 has been changed and updated considering the weather from 16 to 26
New graph edges will be as follows
Edge: 1–2 (21)
Edge: 1–6 (24)
Edge: 1–7 (23)
Edge: 2–3 (36)
Edge: 3–5 (11)
Edge: 3–8 (38)
Edge: 4–3 (6)
Edge: 4–8 (6)
Edge: 5–4 (17)
Edge: 5–8 (26)
Edge: 6–3 (23)
Edge: 6–5 (45)
Edge: 6–7 (15)
Edge: 7–5 (36)
Edge: 7–8 (58)
Time in which nodes can be reached
1:0     2:21    3:47    4:75    5:58    6:24    7:23    8:81
The distance between nodes 4 and 3 has been changed and updated considering the weather from 6 to 6
The distance between nodes 4 and 8 has been changed and updated considering the weather from 6 to 13
New graph edges will be as follows
Edge: 1–2 (21)
Edge: 1–6 (24)
Edge: 1–7 (23)
Edge: 2–3 (36)
Edge: 3–5 (11)
Edge: 3–8 (38)
Edge: 4–3 (6)
Edge: 4–8 (13)
Edge: 5–4 (17)
Edge: 5–8 (26)
Edge: 6–3 (23)
Edge: 6–5 (45)
Edge: 6–7 (15)
Edge: 7–5 (36)
Edge: 7–8 (58)
Time in which nodes can be reached
1:0     2:21    3:47    4:75    5:58    6:24    7:23    8:81
Time needed to travel to destination: 81
Path taken would be 178
All nodes shortest time as below
1: time: 0      Path: 1
2: time: 21     Path: 12
3: time: 47     Path: 163
4: time: 75     Path: 16354
5: time: 58     Path: 1635
6: time: 24     Path: 16
7: time: 23     Path: 17
8: time: 81     Path: 178
```