Homework Assignment 4 (Theory)

Submitted by
Vidhi Sharma 2019286
Dolly Sidar 2019304

Bonus Problem:

Explanation:
Let the shortest path s-t be ShortDist.
Let's define each edge as the edge(u, v) with weight w.
IDEA:

- Since, the shortest path s-t contains all vertices, it is a **spanning tree.** We can apply the **cut property** of the spanning tree here.
- If an edge is on all the possible shortest paths from s to t, then it is a **vital arc.** If this vital arc is removed from the graph, then the shortest path s-t increases.
  b ∈ E such that b is a vital arc, then
  Shortest path s-t in G \ b > Shortest path s-t in G (ShortDist)
- If an edge is not a vital arc, then the shortest path s-t does not change.

ALGORITHM:

- First we need to calculate the shortest path of all vertices from source s and target t using Dijkstra's algorithm.

```
Ds = new int[];
Dt = new int[];
//Initialize Ds and Dt
  Dijkstra( s, Ds)  // shortest path from s to all vertices
  Dijkstra( t, Dt)   // shortest path from t to all vertices
```

- Then we need to find all edges that are on any shortest path s-t

```
ArrayList<Edge> pathEdges;
for all e Edge(u, v):
        if(Ds(u) + w + Dt(v) == ShortDist)
                pathEdges.add(e);
```

- Now to find vital arcs we know that they will be present in the pathEdges arraylist.

For an edge(u1, v1) to be a vital arc we need to check if it is present in pathEdges and also check if for all other edges(u2, v2) in pathEdges

Ds[v2] < Ds[u1]   or   Ds[v1] < Ds[u2]

To do this, sort Arraylist pathEdges according to Ds(u) and Ds(v) values using Comparator.

```
Collections.sort(pathEdges, new EmpComparator()); //EmpComparator
                                        here sorts according to Ds(u) and Ds(v) values
HashSet<Edge> bridge;
Int r = -1;
for i=0 to pathEdges.size()
      if (Ds[pathEdges.get(i).u]>=r)
          if(pathEdges.size()>i+1)
                if(Ds[pathEdges.get(i).v] <= Ds[pathEdges.get(i+1).u])
                        bridge.add(pathEdges.get(i));
          else
                bridge.add(pathEdges.get(i));
      r = max(r, Ds[pathEdges.get(i).v);
End Loop;
```

- Now, we have to see how the removal of the vital arc changes (increases) the shortest path s-t. Let's take a vital arc bi such that it divides the graph G into 2 separate graphs SepGraph i and i+1. Let Eb be the set of all edges(u, v) which join SepGraphs 0...i to SepGraphs i+1….i+n . Then, the shortest path s-t with bi edge removed will be:

  min { Ds(u) + w + Dt(v) }   where edge(u, v) $\in$ Eb

  First, for SepGraph 0, we have to see all the vertices that are on the shortest path without any bridge. For SepGraphs i > 0, shortest path must include bridge i-1.

  We need to,

```
Collections.sort(bridge, new EmpComparator) //sort bridges in the order
                                                   they appear in the shortest path
run depth-first search on s
For all edge(u, v) in bridge
        If( v is not visited )
                run depth-first search on v
```

- Let there be removal= new int[];
  removal[i] stores the shortest path s-t such that bridge edge bi is removed.
  Now create a SegmentTree SegTree such that it has two functions:

1) update function: update(i,j,value) updates removal[k]= min{removal[k],value} for all k=i...j-1.
2) Query function: query(e) returns removal[e] where e is a bridge edge

Now,

For all e edge(u,v) such that e is not bridge edge

if (SepGraph[u] <= SepGraph[v] - 1)

SegTree.update(SepGraph[u], SepGraph[v] - 1, Ds[u] + w + Dt[v]);

- Now create an array of size E:

  ans = new int[E];

  For all e edge(u,v) ∈ E

  if(bridge.contains(e)) // Bridge.contains() complexity is O(1) for hashset

  ans[i] = SegTree.query(e);

  else

  ans[i] = ShortDist // shortest distance s-t of graph G unchanged

  Return ans;

Time Complexity:

Dijkstra algorithm: O(E log V)
Finding edges in shortest path: O(E)
Finding vital arc edge: O(E)
Finding SepGraph using DFS: O(E log V)
(maximum time dfs is called is E)
Segment tree construction: O(n) = O(E)
Segment tree updation: O(log(E))
Segment tree search (query): O(1)
Creation of array: O(E)

So, total time complexity is O(ElogV)