HOMEWORK-4

1. Design a Data structure that has the following properties:

- Find median takes O(1) time

Let us consider two heaps - Min heap & and Max heap.

Initially both max heap and min heap are empty. Insert the elements - first half in Max heap and second half in minheap.

Find_median()

If len(min_heap) == len(max_heap)
AND len(min_heap) > 0:
median = [root(max_heap) + root(min_heap)] / 2
return median

If len(min_heap) > len(max_heap):
median = root(min_heap)

else:
median = root(max_heap)

finding median takes O(1) time

- Insert takes O(logn) time

IF len(max_heap) == len(min_heap) = 0 :
    insert new element in min_heap.

ELSEIF root(min_heap) < a:
    insert new element in ~~max~~ min _heap.

ELSE
    insert new-element in max_heap.

We now check a condition of lengths of max heap and min heap are differing more than one.

IF len(max_heap) − len(min_heap) > 1 :
    EXTRACT_MAX()
    insert into min_heap

ELSEIF len(min_heap) − len(max_heap) > 1:
    EXTRACT_MIN()
    insert into max_heap

This insert operation would take O(logn)

2. G (V,E) is a near tree, connected, has atmost n+k edges.

As per the given information, there are (n+k) edges. But a MST contains atmost (n-1) edges.

Extra edges = (n+k) - (n-1)
= K+1 edges.

K+1 edges are to be removed to form MST. Even after the removal the nodes are still connected by n-1 edges.

The K+1 edges may form cycles. We can remove ~~three edges~~ edge from a cycle which is of maximum cost. The nodes with n-1 form MST.

We now apply BFS for (K+1) times and remove edge from a cycle that is of highest weight.

After (K+1) times of BFS, (n-1) edges will form MST.

T.C = O(K(V+E)) = O(V+E)

Linear

3. You are given a minimum spanning tree T in a graph $G = (V, E)$. Suppose we remove an edge from G creating a new graph G_1. Assuming that G_1 is still connected, devise a linear time algorithm to find a MST in G_1.

Answer: Given a graph G and its MST as T.
If we remove an edge from G and a new graph G_1 is formed. Our goal is to find MST of new graph G_1. We have 2 cases for this.

Check whether the edge to be removed is present in the MST of graph G.

Case 1: If the edge to remove is present in MST of G, T then remove the edge and disconnect the MST into two trees T_1 and T_2.

Now find all the edges from T_1 to T_2 in the graph G_1. Each of the two nodes will be present in the graph and our goal is to find the vertices and then find the minimum cost edge and attach the two nodes to form a new MST in G_1.

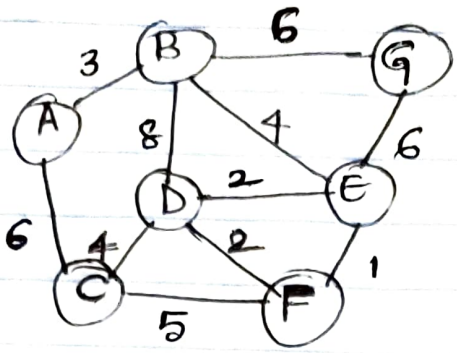
Case 2: If the edge to be removed is not present in MST of G, then the MST of G_1 will be the same as that of G.

Overall T.C = $O(V+E)$

↓ to find vertices

→ to find minimum cost edge.

4. Considering the following graph G



1. In graph G, if we use Kruskal's algorithm to find the MST, what is the third edge added to the solution? Select all correct answers.

Answer: The third edge added to the solution is

C. A-B

2. In graph G, if we use Prim's algorithm to find MST starting at A, what is the second edge added to solution?

Answer: The second edge added to the solution is

b. B-E

3. What is the cost of MST in the graph?

Answer: C. 20

5. Given, router is represented as vertex and wires connecting routers are represented as edges.

Our goal is to find path with maximum bandwidth from 's' to 't'

We modify the Dijkstra's algorithm to always pick maximum value in heap i.e MAX-HEAP.

We assign values of '0' to all nodes except 's'. 's' is assigned a value of '∞' (infinity)

S = Null.
Initialise priority queue (Max heap) Q will all node values and d(v) being key value.

```
WHILE S ≠ V
    V = EXTRACT_MAX(Q)
    S = S ∪ {v}
    for each vertex u ∈ adj(v):
        if d(u) < min(d(v), le):
            d(u) = min(d(v), le) → Increase_key
                                   (Q, u, min(d(v), le))
                update parent of u to v as well.
    end if
    end for
endwhile
```

The paths from $s \rightarrow t$ will have maximum bandwidth from $s \rightarrow t$.

6. Given an undirected graph $G = (V, E)$.
It is given that there is a faulty server.

Let G' be the graphs without the faulty server S.

Now we have to check if the graph G' is still connected after we remove the node S.
Inorder to check this, we have to run either BFS or DFS from each and every node and check whether all the nodes can be reached.

If the graph G' does not contain path / find a way to reach all other nodes, then we can say it is not connected and a solution that minimises the sum of maintenance costs of remaining edges is not possible.

If the graph G' is connected, construct Minimum spanning tree T and then find an edge that is of minimum cost ~~and connects~~ between ~~es~~ S and adj (s) and connect it to G'.

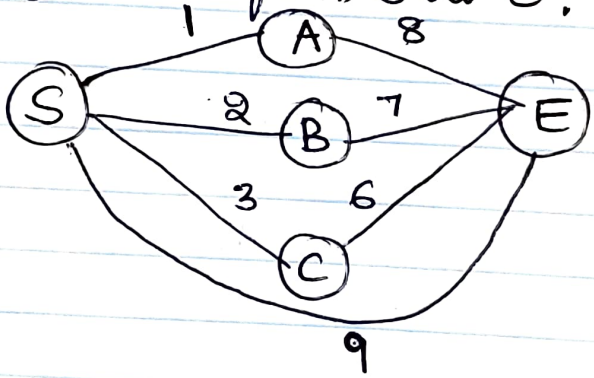
The graph will be a MST with S as a leaf.

7. Given a connected graph $G=(V,E)$ with positive edge weights. In V, s and t are two nodes for shortest path computation, prove or disprove with explanation

1. If all edge weights are unique, then there is a single shortest path between any two nodes in V.

Answer: I would like to discuss this with a counter example that the statement is FALSE

Consider the following graph G which contains 3 paths between the starting node S and end node E via A, B and C and a direct node from S to E.



From the above graph G we could see that there is more than one (1) shortest path.

$S-A-E = 9$

$S-B-E = 9$

$S-C-E = 9$

$S-E = 9$

So the above given statement is FALSE.

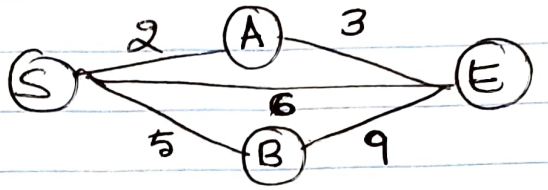
7

2.

2. If each edge's weight is ~~replaced by its~~ increased by k the shortest path cost between s and t will increase by a multiple of k.

Answer: FALSE.

By using a counter example, Consider a graph G shown below.

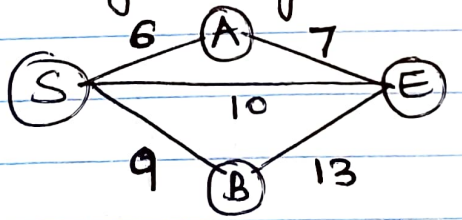


From the graph G, the shortest path from S–E is via node A.

$S \rightarrow A \rightarrow E$

$Cost_1 = 2 + 3 = 5$

Now let us say that we increase the each edge weight by a value $k = 4$,



The shortest path from S–E now will be the direct path from $S \rightarrow E$ and the $Cost_2 = 10$.

Increase in Cost = $Cost_2 - Cost_1 = 10 - 5$

increased Cost = 5

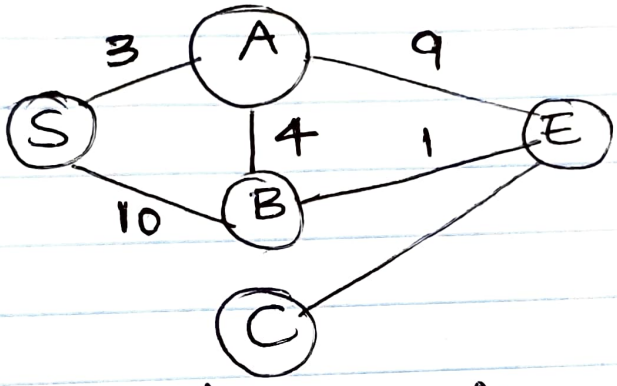
But 5 is not a multiple of 4(k)

Hence, the above statement is false

3. If the weight of some edge e decreases by k then the shortest path cost between s and t will decrease by at most k.

Answer: FALSE

By using a counter example, consider a graph G shown below.



Now if we decrease the edge values by a value $k = 4$, then the graph would turn into an infinite loop as there is a cycle in the graph and the edges S – A would become negative.

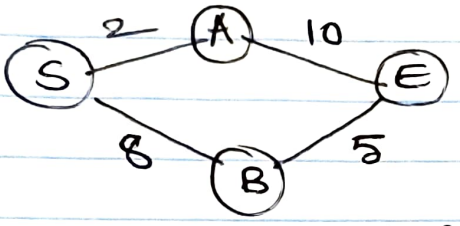
If there is a cycle, then the shortest paths would go into a cycle infinite times resulting in infinite path cost.

Hence, the above statement is FALSE

4. If each edge's weight is replaced by its square, i.e w to w^2, then the shortest path between s and t will be the same as before but with different costs

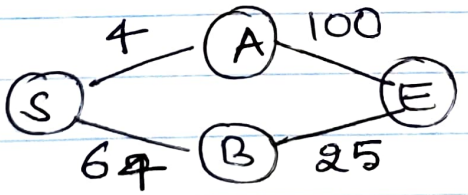
Answer: FALSE

Consider a graph G shown below.



Now, the shortest path between S and E is via node A. $S \rightarrow A \rightarrow E$ with cost of $2+10=12$.

Now if square the edge values then the graph would look like



Now, the shortest path is $S \rightarrow B \rightarrow E$ with cost of $64+25 = 89$. As we see that the path has changed when we squared each of the edge values, we can say that the above given statement is false.

8. Given a weighted graph G where all edge weights are positive.

We can set the edge weight of any one edge to zero.

The time complexity of the naive solution would be $O(E^2 \log V)$

But we solve this in $O(E \log V)$ time.

Our goal is to find the lowest-cost path from node s to node t.

Initially run the Dijkstra's algorithm from node 's' to every other node. Now reverse all the edges and a new graph G' is formed.

Apply dijkstra's algorithm from 't' to find shortest distance from every other node to 't'.

Now the current edge be (a,b) between 's' and 't'.

$$\text{Cost} = \text{distance}(s \to a) + \text{dist}(a \to b) + \text{dist}(b \to t)$$

Now our goal is to find the lowest cost by setting the edge value to zero that minimises the cost.

Then cost = dist (s → a) + dist (b → t)

The time complexity = $2 * O(E \log V) + O(E)$

$2 * O(E \log V)$ — Dijkstra's algorithm

$O(E)$ — finding optimal edge.

$$\boxed{T.C = O(E \log V)}$$