

Homework 4

3] Let $G = (V, E)$ be the graph and (u, v) be the edge that is deleted in order to form G' . Suppose after deletion the spanning tree T still remains connected then we won't do anything as it will now be the ~~Spanning~~ minimum Spanning tree for G' , as well.

But if T is not connected then T would be but naturally divided into T_1 and T_2 .

Now we will find all the edges that will connect T_1 and T_2 . After which we will find the one with minimum weight and add that edge to the minimum Spanning tree (T) as that is the required ^{minimum} Spanning tree.

Time Complexity:-

First we have to find T_1 and T_2 after which we have to find all the edges.

Lastly traverse through them to find the minimum one.

So, the overall time complexity would $O(V + E)$.

4] ① c] A-B

② b] B-E

③ c] 20

6] Let graph $G(V, E)$ be the graph with n servers and G_1 be the graph (V, E_1) where S be not present.

∴ So, G_1 won't be having S and all its adjacent edges.

We then either BFS or DFS to check whether G_1 is connected or not. Starting at any server.

If G_1 is a connected graph :

compute the MST using any MST algorithm on G_1 .

Add the edge between S and it's adjacent edges which has the minimum cost to MST.

Else:

Impossible to compute.

By doing so in a way we are ensuring that S would be a leaf ~~no~~ in the MST.

7] i) This statement is False.

Let's say there ~~is~~^{an} edge from S to c (S, c) which weighs 1 ~~and~~, we have one more edge S to t which weighs 2 and one edge from (S, t) which weighs 3.

So, there are now two shortest paths from S to t both weighing 3.

ii) This Statement is also false.

Let there be a shortest path from (S, t) which has 2 edges and each of them ~~is~~ of cost 1.

Let there be one more edge connecting S and t having cost 3. Now if we increase the cost by 1 each, then the shortest ~~ap~~ path will have a cost of 5 solid

is not the multiple of k

iii) This statement may be true when we assume that after decreasing the weights by k , the weights are still positive. Let's say the edges which form paths from s to t decrease by k , so in that case the shortest path would also decrease its cost by at most "k" as it is one of the edges paths.

If we consider a case where by decreasing the weights we might end up with negative weights. If we do get negative weights and let there also be a cycle in our graph which includes a path from s to t . So, in order to find the shortest path will end up going in that cycle infinite times. and so, our cost won't be decreasing by at most k times.

iv) This statement is false. Let's say there are 2 paths from

s to t. One having one edge (b) which costs 5 and the other having 2 edges, each of 3 cost. ~~s → b → t~~ would

\textcircled{b} $s \xrightarrow{b} t$ this would be the shortest path. When we square the costs, edge b's cost becomes 25 whereas edges a and c become 9 each. So, now, the shortest path is from edges a and c. Thus on squaring the ~~costs~~ shortest path changed.

- 5] We will use a max priority queue instead of a min as we want maximum bandwidth. In the relax step we will also modify the insert step in the ~~min~~ priority queue because the path quality is the minimum of edges weights. All the above changes won't be affecting the correctness of Dijkstra's so exploring the vertices would be like Dijkstra's.

2] In order to make sure that the graph G becomes a MST, we can use the cycle property 'k' times. This means that we run BFS till the time we don't find a cycle in the graph and once we do we remove the edge having highest cost. Till we continue doing this now we have removed one edge from G and we are also ensuring that it is connected thus not violating the MST property. Now, if we continue to do this 'k' times, we will end up getting another graph G' which in fact is a tree and it is a MST.

The time complexity for each iteration would be $O(m+n)$ for BFS and subsequent check to remove the edge having high cost. Here $m \leq n+k$ so the $O(n)$ is the time complexity.

1] We will use max heap to store first half of the elements and min heap to store the second half.

The median would be the root of max heap and hence we can access it in $O(1)$ time. Here we are assuming ~~the~~

n to be even and median to be $\frac{n}{2}$ when the array is sorted in increasing order.

Insert () :

$$\text{len}(\text{Max heap}) = 0$$

$$\text{len}(\text{Min heap}) = 0$$

If $x < \text{median}$, we insert in max-heap.
Everytime we insert an element in max-heap, we increase it's length by 1.
Otherwise we insert in min-heap
and increase it's length by 1.
This would take $O(\log n)$ in worst case.

If $\text{size}(\text{max heap}) > \text{size}(\text{min heap}) + 1$, then
we remove one element from
max heap and decrease it's size
by 1. We then add the removed
element in min heap and increase it's
size by 1. This would also take
 $O(\log n)$ in worst case.

If $\text{size of } (\text{min heap}) > \text{size}(\text{max heap})$,
we remove one element from
min heap and decrease it's size

by 1. We add the extracted element in max heap and increase it's length by 1. This would also take $O(\log n)$ time.

Find Median () :

If $[len(max\text{heap}) + len(min\text{heap})]$ is even,
then return (sum of roots of maxheap
and minheap by 2) as the median.

Else if $len(max\text{heap}) > len(min\text{heap})$.
Return the root of maxheap as median

Else return the root of minheap as
median

8]

We will be using Dijkstra's. We reverse the edges and use Dijkstra's from s to all vertices to t . We let the shortest distance $v \xrightarrow{\sim} v$ be denoted like this as the length by $l(v, v)$. We will try to set each edge as zero.

So for $(v, v) \in E$, we consider

$s \rightarrow v \xrightarrow{\sim} v \rightarrow t$ if $w(v, v) = 0$

$w(v, v) = 0$ then $l(s, v) + l(v, t)$ is the final path length. We will keep on doing this until we find the length which is minimized.

Complexity : - Since we are running the two invocations of Dijkstra's and an additional traversal through the edges would take $O(|E|)$ time, the final complexity would be same as Dijkstra's.