6498-8280-04 CSCI 570

HW4

Q1 We will consider two Linked Lists that will contain 1 Max-Heap and 1 Min Heap. The data structure will be divided in 2 sorted parts, where the left part will contain the smaller ~~parts~~ values and the higher values will be stored in the Min Heap. We also have two ~~pointer~~ variables, lenMax and lenMin which will contain the count of elements in the respective heaps.

Algorithm to find Median:

If lenMax = lenMin and not zero:
  Pick the root element of min Heap and max Heap. Let it be x and y respectively.
  Return (x+y)/2
Else If lenMax > lenMin:
  Return Root element of max-heap
Else
  Return Root element of min-heap.

Time Complexity: O(1)

PAGE No.
DATE / /

Algorithm for Insert:

If lenMax and lenMin are zero:
    Add element to max-heap
    Increment lenMax by 1
Else If root element of min-heap < element:
    Insert an element to min-heap.
    Increment lenMin by 1
    If lenMin - lenMax > 1:
        Extract root element of min-heap and insert it into max-heap
        Decrement lenMin by 1
        Increment lenMax by 1
Else
    Insert an element to max-heap
    Increment lenMax by 1
    If lenMax - lenMin > 1:
        Extract root element from max heap and insert it into min-heap
        Decrement lenMax by 1
        Increment lenMin by 1
End If

Time Complexity:
All operations take $O(1)$. The insertion in a max/min heap takes $O(\log n)$ time. Hence, the time complexity is $O(\log n)$.

PAGE NO.

DATE / /

Q2

We know that whenever an edge with the maximum cost in a cycle is removed, the graph would still be connected.

Using this fact, with (n+k) edges in a graph, the minimum spanning tree of a graph will have (n-1) edges which would be required to connect the graph without creating a cycle.

Thus, in order to remove those k edges, we will run the BFS for k times. For each execution, we will detect a cycle and remove the highest cost edge in the cycle.

Q3

The question says that a MST T is provided in Graph G. An edge is removed from Graph G, creating a new graph G1. The assumption is Graph G1 is still connected. We need to check if removed edge was part of MST.

Algorithm:

If removed edge ~~affects~~ is a part of the MST T:
 Check edge of nodes that are connected to the removed edge.
 Select one of those edges and find MST in Graph G1.
Else:
 Remove the edge
 Retain the previous MST.

Q4

(1) ~~(a) E-B~~ ~~(b) D-E~~ (c) A-B

(2) (b) B-E

(3) (c) 20

Q5

Here, the requirement is to select the highest weighted edge first so as to avoid the bottlenecks for router communication.

Hence, in the part of Djikstra's Algorithm where we choose the most minimum edge, we will select the most heavy edge so as to maximize the bandwidth and avoiding any Bottlenecks.

The modified algorithm will be as follows:

Initially $S = \{s\}$ and $d(s) = 0$
for all other nodes $d(u) = 0$
While $S \neq V$
Select a node $v \notin S$ with atleast one edge from S for which
$d(v) = \max_{e(u,v) : u \in S} (d(u) + l_e)$
Add v to S
End while

DATE / /

Q6 We have a graph $G = (V, E)$ with vertex denoting the servers and edges representing the ~~edges~~ links to the servers.

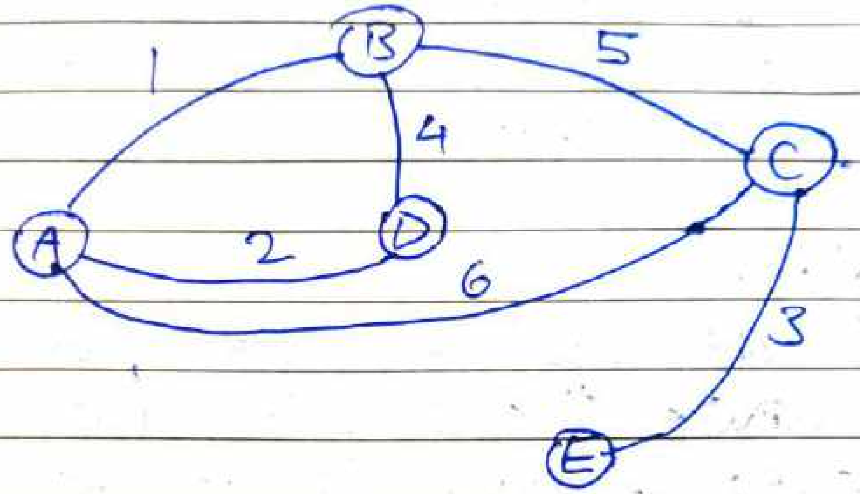
We will make another graph, G' that will ~~re~~ not contain the faulty server S.

<u>Algorithm:</u>

Run Breadth-First Search on Graph G' to check if the graph is connected. We can begin from any server.

If G' is not a connected graph:
 There is no way to eliminate server S.
 Return
Else:
 Create a Minimum Spanning Tree using Kruskal / Prim's / Reverse Delete
 Add exactly one node that will connect node S and its neighbour with the minimum cost.
 Run BFS again to compute the sum of maintenance cost of the remaining edges.
End If

Q7 We will take an example undirected graph to prove/disprove the explanations for questions 1 to



(1) False.

Here, in the above graph, we see that each edge weights are unique. But if we compute the shortest path for A-E, there are two solutions: A-B-C-E and A-C-E both having the same cost i.e., 9 (1+5+3 and 6+3).

(2) False.

If we assume k=5 and AC=13 the shortest path from A to E is A-B-C-E. New edge costs are AB=6, BC=10 and CE=8, which adds up to 24, and 24 is not a multiple of 5.

PAGE No.
DATE / /

(3) False.
If we reduce any edge by k and the weight falls down to negative numbers, provided the graph contains a circle, the Dijkstra's algorithm will be stuck in an infinite loop and thus will not be able to compute the shortest path.

(4) False.
If we assume AB=3, and now compute the shortest path between nodes B and D, it will be 4 (B-D). Squaring the weights of the edges, we get AB=9, AD=4 and BD=16. Now, the shortest path of nodes B and D will be A-B-D (9+4 i.e., 13 < 16).

Q8

For this problem, we will find the shortest path using Dijkstra's algorithm. Maintain an array that will contain the parent node to easily backtrack. Maintain another variable, 'maxWeight' to store the maximum edge weight. While running Dijkstra's, compare the edge weight to maxWeight

If it is higher, store its value in maxWeight.

Once the execution is done, ~~from~~ ~~Dijkstra~~ set the value of the highest weighted node to zero. This way, our solution is optimized.

Since the cost of the shortest path can't be lower than any other paths, reducing the maximum weighted node to 0 is the optimal choice.