

# CSCI 570 - Fall 2022 - HW 4

Due: September 21, 2022

## Graded Problems

### Problem 1

[10 points] Design a data structure that has the following properties (assume  $n$  elements in the data structure, and that the data structure properties need to be preserved at the end of each operation):

- Find median takes  $\mathcal{O}(1)$  time
- Insert takes  $\mathcal{O}(\log n)$  time

Do the following:

1. Describe how your data structure will work.
2. Give algorithms that implement the Find-Median() and Insert() functions.

### Solution

We use the  $\lceil n/2 \rceil$  smallest elements to build a max-heap and use the remaining  $\lfloor n/2 \rfloor$  elements to build a min-heap. The median will be at the root of the max-heap and hence accessible in time  $\mathcal{O}(1)$  (we assume the case of even  $n$ , median is  $n/2$ -th element when elements are sorted in increasing order).

Insert() algorithm: For a new element  $x$

- Initialize len of maxheap (maxlen) and len of minheap (minlen) to 0.
- Compare  $x$  to the current root of max-heap.
- If  $x < \text{median}$ , we insert  $x$  into the max-heap. Maintain length of maxheap say maxlen, and every time you insert element into maxheap increase maxlen by 1. Otherwise, we insert  $x$  into the min-heap, and increase length of minheap say minlen by 1. This takes  $\mathcal{O}(\log n)$  time in the worst case.

- If  $\text{size}(\text{maxHeap}) > \text{size}(\text{minHeap}) + 1$ , then we call  $\text{Extract-Max}()$  on max-heap, and decrease  $\text{maxlen}$  by 1 and insert the extracted value into the min-heap and increase  $\text{minlen}$  by 1. This takes  $O(\log n)$  time in the worst case.
- Also, if  $\text{size}(\text{minHeap}) > \text{size}(\text{maxHeap})$ , we call  $\text{Extract-Min}()$  on min-heap, decrease  $\text{minlen}$  by 1 and insert the extracted value into the max-heap and increase  $\text{maxlen}$  by 1. This takes  $O(\log n)$  time in the worst case.

Find-Median() algorithm:

- If  $(\text{maxlen} + \text{minlen})$  is even: return  $(\text{sum of roots of max heap and min heap})/2$  as median
- Else if  $\text{maxlen} > \text{minlen}$ : return root of max heap as median
- return root of min heap as median

### Rubric

- 2 pt: Using max heap and min heap to store first half and second half of elements.
- 3 pt: Comparing element to the root and ~~proper~~ if conditions for inserting in appropriate heap
- 2 pt: Proper if conditions for returning the median.
- 3 pt: Correct Time Complexity

### Problem 2

[10 points] Let us say that a graph  $G = (V, E)$  is a near tree if it is connected and has most  $n + k$  edges, where  $n = |V|$  and  $k$  is a **constant**. Give an algorithm with running time  $O(n)$  that takes a near tree  $G$  with costs on its edges, and returns a minimum spanning tree of  $G$ . You may assume that all edge costs are distinct.

### Solution

1. To do this, we apply the Cycle Property  $k + 1$  times. That is, we perform BFS until we find a cycle in the graph  $G$ , and then we delete the heaviest edge on this cycle.
2. We have now reduced the number of edges in  $G$  by one while keeping  $G$  connected and (by the Cycle Property) not changing the identity of the minimum spanning tree.
3. If we do this a total of  $k + 1$  times, we will have a connected graph  $H$  with  $n - 1$  edges and the same minimum spanning tree as  $G$ . But  $H$  is a tree, and so in fact it is the minimum spanning tree.
4. The running time of each iteration is  $O(m + n)$  for the BFS and subsequent check of the cycle to find the heaviest edge; here  $m \leq n + k$ , so this is  $O(n)$ .

### Rubric

- 3 pt: Applying cycle property  $k + 1$  times.
- 2 pt: Figuring out finding LCA to find heaviest edge
- 2 pt: Figuring out that we will have to delete the heaviest edge on this cycle
- 3 pt: Correct Time Complexity

### Problem 3

[3 points] 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  $G_1$ .

### Solution

Suppose the removed edge is named  $e_r$ .

Case 1: If  $e_r$  is not in  $T$ . The MST is  $T$ . This takes linear time.

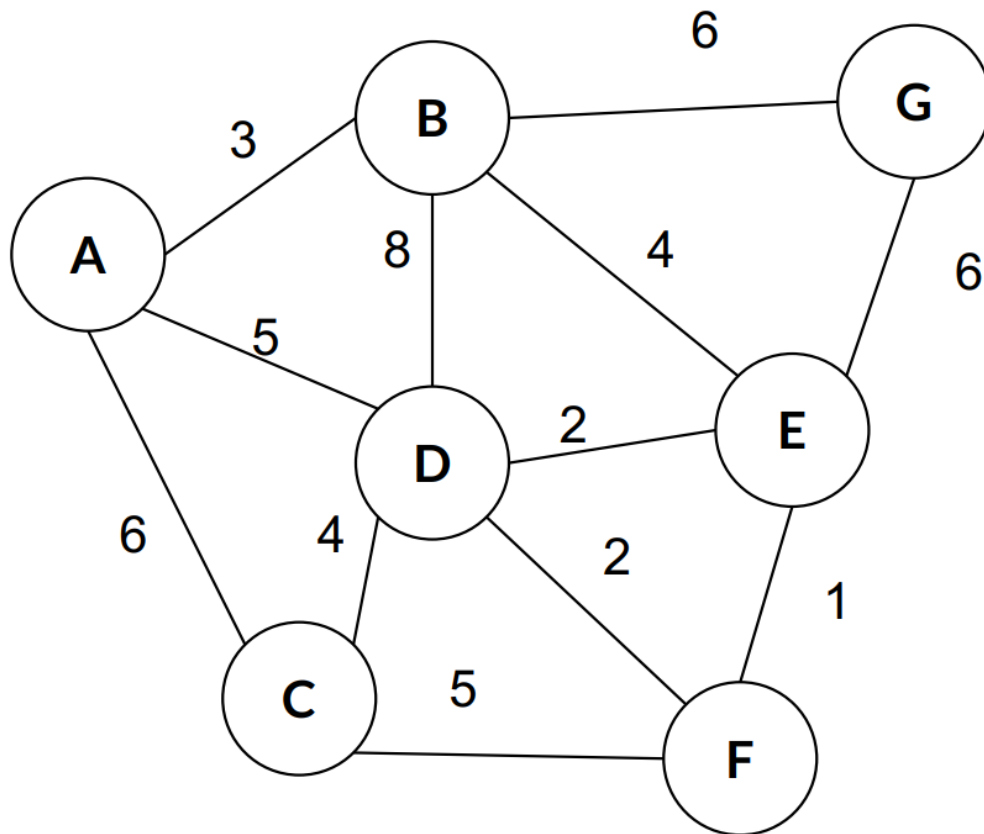
Case 2: If  $e_r$  is in  $T$ . Once we delete an edge from  $T$ , the tree becomes disconnected. We need to find the minimum weight edge that connects two components,  $T_1$  and  $T_2$ . Pick any component say  $T_2$  and find all edges going to  $T_1$ . Among them choose the one which has the smallest cost. Runtime  $\mathcal{O}(E)$ .

### Rubric

- 1 pt: The first case
- 2 pt: The second case

### Problem 4

[9 points] 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

- a. E-F
- b. D-E
- c. A-B
- d. C-F
- e. D-F

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

- a. B-G
- b. B-E

- c. D-E
  - d. A-D
  - e. E-F
3. What is the cost of the MST in the Graph?
- a. 18
  - b. 19
  - c. 20
  - d. 21
  - e. 22

**Solution**

- 1. c. A-B
- 2. b. B-E
- 3. c. 20

**Rubric**

3 points each

**Problem 5**

[10 points] A new startup FastRoute wants to route information along a path in a communication network, represented as a graph. Each vertex represents a router and each edge a wire between routers. The wires are weighted by the maximum bandwidth they can support. FastRoute comes to you and asks you to develop an algorithm to find the path with maximum bandwidth from any source  $s$  to any destination  $t$ . As you would expect, the bandwidth of a path is the minimum of the bandwidths of the edges on that path; the minimum edge is the bottleneck. Explain how to modify Dijkstra's algorithm to do this.

**Solution**

Data structure change: we'll use a max heap instead of a min heap used in Dijkstra's.

Initialization of the heap. Initially all nodes will have a distance (bandwidth) of zero from  $s$ , except the starting point  $s$  which will have a bandwidth of  $\infty$  to itself.

Change in relaxation step. Based on the definition of a path's bandwidth, the bandwidth of a path from  $s$  to  $u$  through  $u$ 's neighbor  $v$  will be  $d(u) = \min(d(v), \text{weight}(v, u))$ , because

the bandwidth of a path is equal to the bandwidth of its lowest bandwidth edge. Therefore, in the relaxation step, we will be replacing:

$$d(u) = \min(d(u), d(v) + \text{weight}(v, u))$$

with

$$d(u) = \max(d(u), \min(d(v), \text{weight}(v, u)))$$

### Rubric

- 3 points for max heap
- 3 points for initialization
- 4 points for relaxation modification

### Problem 6

[10 points] A network of  $n$  servers under your supervision is modeled as an undirected graph  $G = (V, E)$  where a vertex in the graph corresponds to a server in the network and an edge models a link between the two servers corresponding to its incident vertices. Assume  $G$  is connected. Each edge is labeled with a positive integer that represents the cost of maintaining the link it models. Further, there is one server (call its corresponding vertex as  $S$ ) that is not reliable and likely to fail. Due to a budget cut, you decide to remove a subset of the links while still ensuring connectivity. That is, you decide to remove a subset of  $E$  so that the remaining graph is a spanning tree. Further, to ensure that the failure of  $S$  does not affect the rest of the network, you also require that  $S$  is connected to exactly one other vertex in the remaining graph. Design an algorithm that given  $G$  and the edge costs efficiently decides if it is possible to remove a subset of  $E$ , such that the remaining graph is a spanning tree where  $S$  is connected to exactly one other vertex and (if possible) finds a solution that minimizes the sum of maintenance costs of the remaining edges.

### Solution

First we need to check the possibility of node  $S$  having only one neighbor in a spanning tree of the underlying graph. The best way to check this is to remove node  $S$  and all its adjacent edges to form a graph  $G'$ . If  $G'$  is a connected graph, then we can claim it is possible to have a spanning tree where node  $S$  has only one neighbor. To check the connectivity of  $G'$ , the simplest way is to run a DFS or BFS algorithm on  $G'$ .

Considering that  $G'$  is connected, we need to find the spanning tree that minimizes the maintenance cost. Therefore we need to find the MST with the additional constraint that  $S$  should be a leaf. Therefore, we remove  $S$  and all its adjacent edges to form  $G'$ . We run Prim's (or any other MST algorithm) to find the MST of  $G'$ . Among all edges adjacent to  $S$ , we find the one with the minimum maintenance cost and connect  $S$  to the MST using

this edge. The resulting graph will still be a spanning tree and in this spanning tree  $S$  will be a leaf.

#### Rubric

- 5 points: Building your spanning tree in a way that node  $S$  is a leaf
- 5 points: Making sure that the final MST is connected

#### Problem 7

[20 points] 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 explanations:

1. If all edge weights are unique, then there is a single shortest path between any two nodes in  $V$ .
2. If each edge's weight is increased by  $k$ , the shortest path cost between  $s$  and  $t$  will increase by a multiple of  $k$ .
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  $k$ .
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.

#### Solution

1. 1. False. Counter example:  $(s, a)$  with weight 1,  $(a, t)$  with weight 2 and  $(s, t)$  with weight 3. There are two shortest path from  $s$  to  $t$  though the edge weights are unique.

#### Rubric

- 2 pt: Correct T/F claim
  - 3 pt: Provides a correct counterexample as explanation
2. False. Counter example: suppose the shortest path  $s \rightarrow t$  consist of two edges, each with cost 1, and there is also an edge  $e = (s, t)$  in  $G$  with  $\text{cost}(e)=3$ . If now we increase the cost of each edge by 2,  $e$  will become the shortest path (with the total cost of 5).

#### Rubric

- 2 pt: Correct T/F claim
  - 3 pt: Provides a correct counterexample as explanation
3. False.

- Only true when we have the assumption that after decreasing all edge weights are still positive (however we don't have this in the problem). For any two nodes  $s, t$ , assume that  $P_1, \dots, P_k$  are all the paths from  $s$  to  $t$ . If  $e$  belongs to  $P_i$  then the path cost decrease by  $k$ , otherwise the path cost unchanged. Hence all paths from  $s$  to  $t$  will decrease by at most  $k$ . As shortest path is among them, then the shortest path cost will decrease by at most  $k$ .
- When 1) there is cycle in the graph, 2) and there is a path from  $s$  to  $t$  that goes through that cycle, 3) and after decreasing, the sum of edge weights of that cycle becomes negative, then the shortest path from  $s$  to  $t$  will go over the cycle for infinite times, ending up with infinite path cost, hence not “decrease by at most  $k$ ”.

#### Rubric

- 2 pt: Correct T/F claim
  - 3 pt: Considers the negative cycle case
4. False. Counter example: 1) suppose the original graph  $G$  composed of  $V = A, B, C, D$  and  $E : (A \rightarrow B) = 100, (A \rightarrow C) = 51, (B \rightarrow D) = 1, (C \rightarrow D) = 51$ , then the shortest path from  $A$  to  $D$  is  $A \rightarrow B \rightarrow D$  with length 101. 2) After squaring this path length become  $100^2 + 1^2 = 10001$ . However,  $A \rightarrow C \rightarrow D$  has path length  $51^2 + 51^2 = 5202 < 10001$  Thus  $A \rightarrow C \rightarrow D$  become the new shortest path from  $A$  to  $D$ .

#### Rubric

- 2 pt: Correct T/F claim
- 3 pt: Provides a correct counterexample as explanation

### Problem 8

[16 points] Consider a directed, weighted graph  $G$  where all edge weights are positive. You have one Star, which allows you to change the weight of any one edge to zero. In other words, you may change the weight of any one edge to zero. Propose an efficient method based on Dijkstra's algorithm to find a lowest-cost path from node  $s$  to node  $t$ , given that you may set one edge weight to zero.

**Solution** Use Dijkstra's algorithm to find the shortest paths from  $s$  to all other vertices. Reverse all edges and use Dijkstra to find the shortest paths from all vertices to  $t$ . Denote the shortest path from  $u$  to  $v$  by  $u \rightsquigarrow v$ , and its length by  $\delta(u, v)$ . Now, try setting each edge to zero. For each edge  $(u, v) \in E$ , consider the path  $s \rightarrow u \rightsquigarrow v \rightarrow t$ . If we set  $w(u, v)$  to zero, the path length is  $\delta(s, u) + \delta(v, t)$ . Find the edge for which this length



is minimized and set it to zero; the corresponding path  $s \rightarrow u \rightsquigarrow v \rightarrow t$  is the desired path.

Time complexity: The algorithm requires two invocations of Dijkstra, and an additional  $O(|E|)$  time to iterate through the edges and find the optimal edge to take for free, which is asymptotically less than Dijkstra. Thus the total running time is the same as that of Dijkstra. (For time complexity analysis, any implementation for Dijkstra's, such as with Binary heap or with Fibonacci heap works just fine.)

### Rubric

For the algorithm:

- 12 pt if the approach is of the same complexity as Dijkstra's algorithm
- 4 pt if the approach is more efficient than naive but not has the same complexity as Dijkstra's algorithm
- 0 pt if the naive approach or any other approach with larger complexity than naive is proposed.

3 pt for time complexity analysis.