Homework 4

Due: Friday, Sept 26, 11:59 PM PT

1. 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):
    a. Find-median takes O(1) time
    b. Extract-Median takes O(log n) time
    c. Insert takes O(log n) time

    Do the following:
    a) Describe how your data structure is designed. (5 points)
    b) Give algorithms that implement the Extract-Median() and Insert() functions. (8 points)

    Solution:
    For general n, we use the ceiling(n/2) smallest elements to build a max-heap and use the remaining floor(n/2) elements to build a min-heap. The median will be at the root of the max-heap for odd n, and average of the two roots for even n, and hence accessible in time O(1).
    For extraction, if n is odd, we remove the root from the max_heap in O(log n), and if n is even, we just output the median value. (Note that the 'extract' operation is not applicable for even n by default, as the median is not an element itself - one can introduce a convention as applicable, such as "simply output median value for even n" or "extract middle two for even n" etc leading to slightly different implementations - this was not specified in the question).
    For inserting an element, we do the following for even n (i.e., after insertion, n will be odd). First, we compare it with the root of the min-heap: If it is smaller, we insert it in the max-heap. This is consistent with our design to have the max-heap have one extra element for odd n. If it is bigger, we add it to the min-heap, but along with that, we extract the root of the min-heap and add it to the max-heap.
    If n is odd, we compare the element with the root of the min_heap. If it is bigger than the root of the min_heap, we remove the root, add it to the max_heap, and add the new element to the min_heap. If it is smaller, then we add the new element to the max_heap. Since all insertion and extract_min can be done in O(log n) in a heap, this operation also takes O(log n) time.

    Rubrics:
    - 5 points for using two heaps for the smaller half and one for the bigger half.
    - 3 points for the correct Extract-Median.
    - 5 points for the correct Insert.

2. Given an undirected graph $G = (V, E)$ where a set of data centers V is connected by a network of optical links E. Each link (u, v) has a positive latency cost $l(u, v)$. There is a

proposal to add a new optical link to the network. The proposal specifies a list C of candidate pairs of data centers between which the new link may be established. Your task is to choose the link that yields the greatest reduction in end-to-end latency between a given source data center s and target data center t. Design an efficient algorithm for this problem. No proof is required. Give the runtime complexity of your algorithm. (Note that your algorithm's time complexity should not be worse than Dijkstra's shortest path algorithm.) (12 points)

Solution:

- Run Dijkstra's algorithm from s (using l as the edge weights) to compute $dist(s, x)$ for every $x \in V$.
- Run Dijkstra's algorithm from t (treating the graph as undirected or equivalently run on the same graph with weights l) to compute $dist(t, x)$ for every $x \in V$.
- For each candidate link $\{u, v\} \in C$ (with proposed latency $l(u, v)$), the best possible new latency between s and t that uses that link is
  $(dist(s, u) + l(u, v) + dist(t, v),\ dist(s, v) + l(u, v) + dist(t, u))$.
- Choose the link in C whose insertion achieves the smallest such new latency:
  - If this minimum exceeds the original $dist(s, t)$, then no candidate link can reduce the latency.
  - Otherwise, pick the $\{u, v\}$ that attains this minimum (breaking ties arbitrarily).

Complexity: $O((|V| + |E|) \log |V|)$ for the two Dijkstra runs, plus $O(|C|)$ to scan candidates, so in total $O((|V| + |E|) \log |V| + |C|)$.

Rubric:

- 2 points for running Dijkstra algorithm from s.
- 2 points for running Dijkstra from t.
- 3 points for calculating the shortest path between s and t given two candidate data centers (u and v).
- 3 points for considering all possible cases for calculated latencies.
- 2 points for time complexity. Since the time complexity of Dijkstra depends on its implementation details, any correct time complexity for Dijkstra is accepted.

3. 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. (10 points)

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 Prime'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 for making sure that the final MST is connected
- 5 points for building your spanning tree in a way that node S is a leaf