Name: Chaoyu Li
USC ID:6641732094
Date:9/18/2021

CSCI 570 - Fall 2021 - HW4
Due: Sep 23rd

1.1. First, sort the entire sequence from smallest to largest. Take the first half of the sorted sequence to build a max-heap. The root node of this max-heap is the median. Then use the remaining elements to build a min-heap. When we need to take the median, we only need to take the root node of the max-heap. This operation is O(1) complexity. When we need to insert an element, whether it is on max-heap or min-heap, it is O(logn) complexity.

1.2. Find-Median():

Pop the root node of the max-heap

Insert(x):

If x>the root node of the max-heap:

Insert x to the min-heap

Else:

Insert x to the max-heap

Endif

If the size of the max-heap>the size of the min-heap+1:

Pop the root node of max-heap

Insert this node into the min-heap

If the size of the max-heap<the size of the min-heap:

Pop the root node of the min-heap

Insert this node into the max-heap

2. The data structure of the sever is min-heap.

Algorithm:

When the sever is given the first number, build a min-heap. Then every time the server gets a new number, insert it into the min-heap until it is full, all the insert operation takes O(logk) times.

If the min-heap is full, it means that there are k numbers in the min-heap now. And these k numbers are the k largest numbers.

While the server gets a new number x:

If x>the root of the root node of the min-heap:

Pop the root node of the min-heap(O(1))

Insert x into the min-heap(O(logk))

Endif

Endwhile

Therefore, the min-heap always stores k largest numbers. The complexity is O(logk).

3. Let us suppose that there are two different minimum spanning trees T_1 and T_2. Because there is at least one edge that is contained in T_1 but not contained in T_2. Let e_1 be the one with the minimum weight.

Because T_2 is a minimum spanning tree, $T=T_2 \cup \{e1\}$ must contain a cycle.

Therefore, T must contain an edge e_2 that does not belong to T_1.

Because we suppose that the weight of e_1 is minimum, $w(e_2)>w(e_1)$.

If we replace e_2 to e_1 in T_2, it will still be a minimum spanning tree and the sum of the weights become less.

Therefore, T_2 is not a minimum spanning tree, this is a contraction.

4. Algorithm:

```
Set sum as the number of the edges in G. sum=n+8 initial.
While run BFS:
    If there is a cycle in G:
        Delete the edge has the greatest weight
        sum=sum-1
    Endif
     If sum=n-1:
        Return
    Endif
Endwhile
```

Complexity analysis:

Because there is alway a cycle in G until sum=n-1, this algorithm will run nine times at most. For each iteration, the complexity of BFS is O(m+n), here m=n+8. Therefore, the complexity of BFS is O(n) here. Every time a cycle is found, this algorithm will find the edge with the largest weight in the cycle. However, it will not take more time. Therefore, the complexity of the whole algorithm is O(9n)=O(n).

5.1. False. Although all edge weights are unique, the sum of them may be the same. For example, there are four nodes {A,B,C,D}, edge weights are {A->B:2, B->C:5, A->D:3, D->C: 4}. Then there are two different shortest paths from A to C with the same weight.

5.2. False. The shortest path may be changed if each edge's weight is increased k. For example, there are three nodes {A,B,C}, edge weights are {A->B:2, B->C:2, A->C:5}. For the shortest path from A to C, we will choose A->B->C initially, and the cost is 2+2=4. However, if we add k=10 to all the edges, the shortest path changes to A->C, and the cost is 15<12+12. 15-4=9 is not a multiple of 10.

5.3. False. For example, there are three nodes {A,B,C}, edge weights are {A->B:2, B->C:2, A->C:5}. For the shortest path from A to C, we will choose A->B->C initially, and the cost is 2+2=4. However, if we reduce k=1 to A->B and B->C, the cost of the shortest path reduces 1+1=2>k.

5.4. False. For example, there are five nodes {A,B,C,D,E}, edge weights are {A->B:1, B->C:1, C->D:1, D->E:1, A->E:3}. For the shortest path from A to E, we will choose A->E and the cost is 3. However, if we let all edge weight square, the shortest path will become A->B->C->D->E with cost $1^2*4=4<3^2=9$.

6. Algorithm:

Run Dijkstra from s to any other nodes, save the distances to array dis1[].
Make all edges reverse
Run Dijkstra from t to any other nodes, save the distances to array dis2[].
Set ans as the cost of the shortest path from s to t
For each edge {u->v} in G:
If dis1[u]+dis2[v](let the weight of u->v=0)<ans:
ans=dis1[u]+dis2[v]
Endif
Endfor
Return ans

Complexity analysis:

For two Dijkstra, the complexity is O(mlogn), which m is the number of edges and n is the number of nodes.

Reversing all the edges takes m times, the complexity is O(m).

The loop will consider every edge to set its weight to zero, so it takes m times at most, the complexity is O(m).

Therefore, the complexity of the whole algorithm is O(mlogn), which is the same as the complexity of Dijkstra.