Rahul Singh 2019MT10285

Assignment-2 MTL-342

Question 1.1

Sol^n. Given :- $G = (V, E)$ be a graph,
T be a MST of G.
e be an edge in T.

Show :- There exist some cut $(S, V \setminus S)$ s.t e' is the smallest weight edge crossing the cut.

Claim 1 :- On removing e' from T forms 2 connected component such that $T \setminus \{e\}$ is unconnected.

Proof :- As trees are minimally connected, removing an edge e' from T would give us two connected components.
Also, the cut corresponding to the 2 connected components has no edge. If it has a edge, then it will be 1 connected component.

Claim 2 :- On removing e from T, Let the two components be S and $V \setminus S$ such that e' is the minimum weight edge in the cut $(S, V \setminus S)$.

Proof :- By Contradiction

Assume there exist another edge e' which belongs to the cut $(S, V \setminus S)$ such that $w(e') < w(e)$
$\because (w(e)$ represent weight of edge$)$.

Now as $T \setminus \{e\}$ is a 2-connected component and e' is an edge which also belongs in the cut of the two connected components.

Then $T \setminus \{e\} \cup \{e'\}$ is a tree. $\because (w(e') < w(e))$.

since $w(T \setminus \{e\} \cup \{e'\}) < w(T)$

So, T can't be an MST. (Contradiction).

Question 1.2. Given:- $G = (V, E)$ is a Graph such that every cut of G has a unique light edge crossing the cut.

Know :- G has a unique MST.

Proof :- From Lemma proved in class, we know that for every cut of G, the unique light edge crossing the cut must be part of every MST of G.

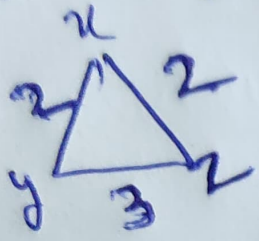
Consider the set T of edges, containing the unique light edge crossing each cut C of G.
We know that T is a subset of every MST.
∴ (by above part).

Claim :- T must be equal to every MST.

Proof :- Since there is exactly one edge of T crossing every cut of G, the graph (V, T) must have exactly one connected component. If there were two unconnected component (V_1, V_2), then the cut (V_1, V_2) would not be crossed by any edge of T. which contradicts the defination of T.

Also, T cannot contain any cycles since the heaviest edge on a cycle in T cannot be the unique lightest edge in any cut.
Then T itself must be a spanning tree of G.
we know T is a subset of all MST of G.
It must be the unique MST of G, as any additional edge added to T would form a cycle, making it no longer be a tree.

Counterexample:- Consider a graph with 3 vertices x, y, z and weights $w(x,y) = w(x,z) = 2$.
$w(y,z) = 3$.



The graph has unique minimal spanning tree containing edges (x,y) and (x,z). However cut $(\{x\}, \{y,z\})$ doesn't have a unique light edge crossing the cut.

Question 1-3

Solution:-

Let the 2 sorted array be X and Y.

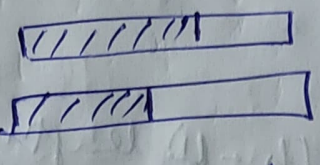
Claim: The first n elements comprise first k elements of X and n-k elements of Y. where $0 \leq k \leq n$.

Proof:-

If k^{th} element of X belongs to first n elements. then 1, 2 . . . , k elements of X also belongs to first n elements.

Similarly, this holds for array Y.

So,



These n elements belongs in first 'n'.

Now if we find 'k'. Then by taking maximum value among X_k and Y_{n-k} we get the median value.

Pseudo code:-

```
func^n Median(X, Y):
    l = 0
    u = n
    while l < u.
        m = (l+u)/2.
        if X_{m+1} < Y_{n-m} and X_m ≤ Y_{n-m+1}.
            set l ← m+1.
```

else if $Y_{n-m+1} < X_m$ and $Y_{n-m} \leq X_{m+1}$
set $u \leftarrow m-1$.

else if $X_{n-m+1} < X_m$ and $X_{m+1} < Y_{n-m}$
break // Null case.

else if $X_{m+1} \geq Y_{n-m}$ and $Y_{n-m+1} \geq X_m$.
return m.

endwhile.

end fun^n.

Time Analysis :-

At every iteration, $(h-l)$ length reduces by atleast half.
Initially $h-l = n$
So, max. no. of iterations $= O(\log_2 n)$.

Now, at every iteration in while loop,
we require 4 accesses to X_m, X_{m+1},
Y_{n-m}, Y_{n-m+1}.

So, Total Time $= 4\log_2 n + C$.
$= O(\log_2 n)$.

Correctness of Algo :-

Case 1:- if $Y_{n-m+1} \geq X_m$ and $X_{m+1} \geq Y_{n-m}$.

Then algo returns m.

claim :- $k = m$.

Proof :- Since $Y_{n-m+1} \geq X_m$.

$Y_i \geq X_m \quad \forall i = n-m+1, \ldots, n$

and $X_i \geq X_m \quad \forall i = m+1, \ldots, n$.

As X_0 and Y are sorted always.

Similarly, we can say. as $X_{m+1} \geq Y_{n-m}$.

$X_i \geq Y_{n-m} \quad \forall i = m+1, \ldots, n$

$Y_i \geq Y_{n-m} \quad \forall i = n-m, \ldots, n$.

Now $X_{m+1}, X_{m+2} \ldots X_n, Y_{n-m+1}, Y_{n-m+2} \ldots Y_n$

are all greater than any of $X_1 \ldots X_m$

and $Y_1, Y_2 \ldots Y_{n-m}$

They are in total n elements.

So, $k = m$

Case 2:- $X_{m+1} < Y_{n-m}$ and $X_m \leq Y_{n-m+1}$.

Then algo sets $l = m+1$.

claim :- $k > m$.

Proof :- Assume $k \leq m$.

Then X_{m+1} will not be in first n.

And Y_{n-m} is in first n.

But $X_{m+1} < Y_{n-m}$

Contradiction.

Hence $k > m$. and l should be set to $m+1$.

Case 3:- $Y_{n-m+1} < X_m$ and $Y_{n-m} \leq X_{m+1}$.

Then algo sets $u = m-1$.

Claim:- $k < m$.

Proof:- Assume $k \geq m$.

This means X_m is in first n.

and Y_{n-m+1} is not in first n,

But $Y_{n-m+1} < X_m$ (Contradiction).

So, $k < m$, and 'u' should be set to '$m-1$'.

Case 4:- $Y_{n-m+1} < X_m$ and $X_{m+1} < Y_{n-m}$.

By the above cases, we get

$k < m$ and $k > m$.

contradiction.

This case can't happen. (Null case).

Algo at each situation, either increases l or decreases 'u' until it finds 'k' or $l = u$.

After finding 'k', we can find maximum value among X_k and Y_{n-k} we can get our desired result, i.e, median value (nth element).

Hence proved.

Question 1.4.

Soln.

Assuming that all edge costs are distinct.

Let $e = (v, w)$ be the new edge being added (given).

Representing T using adjacency list.

Now we find v-w path P in T in linear time in the number of nodes and edges of T, which is $O(|V|)$.

If every node on this path in T has cost less than c, then the cycle property implies that the new edge $e = (v, w)$ is not in MST. since it is most expensive edge on the cycle C formed formed from P and e.

So, MST has not changed.

Now, if some edge on this path has cost greater than c, then the cycle property implies that the most expensive such edge 'f' cannot be in the MST.

So, T is no longer MST.

We replace the heaviest edge on the v-w path P in T with the edge $e = (v, w)$ obtaining a new spanning Tree T'.

Claim :- T' is a MST.

Proof :- Consider an edge e' not in T'.

So, let $e' = (v', w')$.

Adding e' to T' gives us a cycle C' consisting of the v'-w' path P' in T', plus e'.

we consider one further cycle: the cycle K formed by adding e' to T. By the cycle property, e' is the most expensive edge on K.

Now, we have 3 cycles :- C, C' and K.
where edge f is the most expensive edge on C,
and edge e' is the " " " " K.

If the edge e does not belong to C', then C' = K.
and so e' is the most expensive edge on C'.

Otherwise, the cycle K includes f (as C' needed to use e instead) and C' uses a portion of C (including e) and a portion of K. In this case, e' is ~~most~~ more expensive than f (since f lies on K). and hence it is more expensive than everything on C (since f is the most expensive edge on C). It is also more expensive than everything else on K. and it is the most expensive edge on C', as desired.

Hence, T' is MST.

Now, If the edge costs are not all distinct. we ~~pur~~ perturb all edge costs by extremely small amounts so they become distinct. we ~~do~~ add a small quantity ε to the new edge e and we perturb the costs of all other edges f by even much smaller, distinct, quantities ε_f.

Let c(T) denote real cost.
C'(T) denote its perturbed cost.

Perturbation has following properties:-

(i) for trees T_1 & T_2,

if $c'(T_1) < c'(T_2)$ then $c(T_1) \leq c(T_2)$.

(ii) if $c(T_1) = c(T_2)$ and T_2 contains e but T_1 doesn't then

$c(T_2) > c(T_1)$.

for first part of question,

since $c'(T') < c'(T)$ and T' contains e but T doesn't.

Property (i) implies $c(T') \leq c(T)$

Property (ii) implies $c(T') < c(T)$.

for second part of question,

we compute a MST w.r.t to perturbed costs. which, by property 1(i), is also one of several possible minimal spanning Tree w.r.t to the real costs.

Hence proved.

Question 1.5.

Solution.

Claim :- If an element 'r' appears more than $n/2$ times in an array, then it has to appear in majority $n/4$ times in one of left half or right half of array.

Proof:

Suppose, it does not happen,

then r occurs $\leq n/4$ times in both left and right half.

$\Rightarrow$ 'r' occurrence $\leq n/4 + n/4$

$\leq n/2$ times in array.

Contradiction

Hence, the majority element in array is majority in atleast one of left or right half. also.

We use Divide and Conquer Approach.

Pseudo code :- count() returns the element occuring in majority and also its no. of occurences.

```
func^n  count(Array, begin, end)
(x, n) = count(Array, begin, (begin+end)/2)
(y, m) = count(Array, (begin+end)/2, end).
```

Notation.

- occurence of x in Array[begin, (begin+end)/2] is n
- " " " in Array[(begin+end)/2, end] is s
- " y in Array[(begin+end)/2, end] is m.
- " y in Array[begin, (begin+end)/2] is t.

```
if (m + t < n + r)
    return (x, n+r)
else return (y, m+t).

end func^n.
```

Correctness.

Proof by strong Induction.

Basestep:- Holds for array of size '1' sub-array.

Hypothesis:- Holds for sizes atleast 'k' sub-array.

Induction step:- shows for the k+1 sub-array.

func^n count (Array, begin, end) returns the element occuring in majority and also its no. of occurences.

By hypothesis, count (Array, begin, end) returns correct answer if end − begin ≤ k.

By previous claim, we know majority element has to be in majority in one of left or right half. One of x or y is our required answer. as count (Array, begin, $\frac{begin+end}{2}$) and count (Array, $\frac{begin+end}{2}$, end) gives correct answers by Induction Hypothesis.

Now, the occurence of 'x' in the complete sub-array is
n + Occurence of x in Array[$\frac{begin+end}{2}$, end].

the occurence of y in the complete sub-array is
m + Occurence of y in Array [begin, $\frac{begin+end}{2}$].

Occurence of an element is counted by linearly traversing in our case.

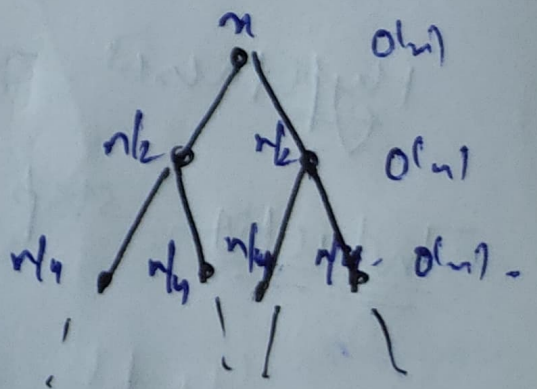
Due to this, correct count of x & y in Array is findout.
Now, the majority count is simply the largest value of count of x & y which is done by the algo. In this way we get the correct count.

Time Complexity :-

Linear Traversal takes $O(n)$ time.

Total time, $T(n) = 2T(n/2) + O(n)$.

$$\boxed{T(n) = O(n \log n)}$$



Question 1.6

Pseudo code. Here Root represent root node of Tree.

```
func^n  LocalMinima(ROOT):
    if ( x(left child of ROOT) < x(ROOT) ):
            Local Minima(left child of ROOT).
    else if ( x(right child of root) < x(ROOT) ):
            Local Minima(right child of root).
    else:
            return root
```

Time Analysis:-

In a complete binary tree with $2^d - 1$ nodes, there will be 'd' passes, as in each pass, goes down a level. and at each pass there are 3 checks

$\Rightarrow$ $O(3d)$ checks

$\Rightarrow$ $O(d)$ checks

$\Rightarrow$ $O(\log n)$

Invariant :-

$x(v) < x(\text{parent of } v)$ if parent of v exist - for any call to LocalMinima(v).

Proof :- First call trivially holds as it does not have any parent.

Now, At any call to LocalMinima(v), we recursively call to LocalMinima(w) only if $x(w) < x(v)$

$\Rightarrow$ Invariant holds.

Existence Proof :-

Claim :- Every complete binary tree has a local minima.

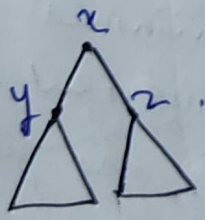
Proof :- By Induction.

Base step : $n = 1$ Trivial.

Hypothesis : $n = 2^d - 1$ statement holds

Induction step :- $n = 2^{d+1} - 1$.

complete Binary with $n = 2^{d+1} - 1$.



** $\triangle$ represent complete binary tree with $n = 2^d - 1$.

Case 1 :- $x_x > x_y$

Subtree rooted at y is complete binary tree and by hypothesis has a local minima. Also, local minima could be 'y' as $x_y < x_x$. (invariant holds).

Case 2:- $x_x > x_z$ and $x_x < x_y$

Subtree rooted at 'z' is a complete binary tree and by hypothesis, it has a local minima. The local minima could be 'z' as

$x_z < x_x$

Case 3:- $x_x < x_y$ and $x_x < x_z$.

$\Rightarrow$ So, x is local minima.

Hence proved.

Claim!- At any node v, LocalMinima(v) gives local minima of subtree rooted at v.

Proof!- If x(left child of v) $< x(v)$.
According to algo, we do LocalMinima(left child of v).
as local minima can be found in left child by previous claim & as left child of v could also be a local minima.

Now if x(right child of v) $< x(v)$.
According to algo, we do LocalMinima(Right child of v)
for similar reason mentioned above.

if v is a leaf, then v is local minima as x(parent of v) $> x(v)$ by Invariant.

Also, if both left and right child of v are greater than v. and invariant holds then v is the local minima.

So, in order to find local minima for complete binary tree, we initialised it from root node.