INTERVAL TREE

This is actually a generalization of AVL trees that we had. AVL tree in one sense is a one dimensional set. While here we're trying to generalize it to a 2 dimensional set, ~~we~~ where we're given the collection of closed intervals (NOT numbers). This is a dynamic set with the same kind of properties such as add, delete and ~~search~~ overlap.

So the basic idea here is to build the AVL tree based on one of the end points. We'll see that we'll build the AVL trees based on the 1st end points & then you have to somehow use the 2nd end point in order to answer the overlap query.

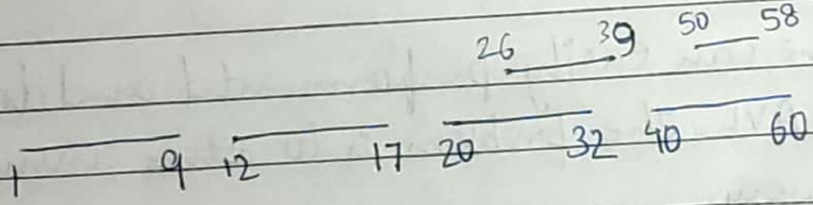
All the operations mentioned should happen in order logn or less.

—x—

Dynamic Collection of Intervals

1.) Add (Interval)
2.) Delete (Interval)

3.) Overlap (Interval) – does it overlap with any one of the intervals in the collection?

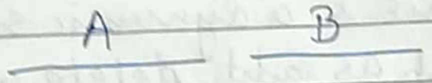


$l(a)$ = lower end point of a
$h(a)$ = high end point of a

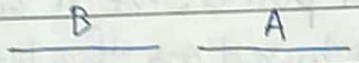
⊛ Law of Trichotomy

Given any two intervals $[l(a), h(a)]$ and $[l(b), h(b)]$, exactly one of the following should hold :-

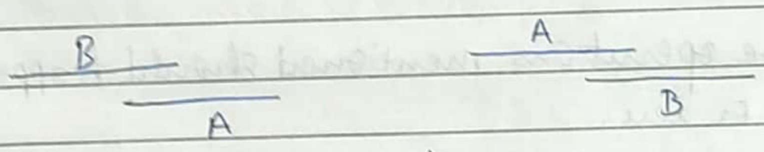
1.) A is to the left of B : $h(a) < l(b)$



2.) B is to the left of A : $h(b) < l(a)$



3.) A and B overlap : $l(a) \leq h(b)$ and $l(b) \leq h(a)$



— x —

⇒ Build a BST

Build a BBST with low end points of the intervals as the keys in the BST.

For simplicity, we'll assume that all the left end points of the intervals are unique.

So now we can easily perform add and delete through standard AVL. The problem is to ~~solve~~ answer the overlap query.

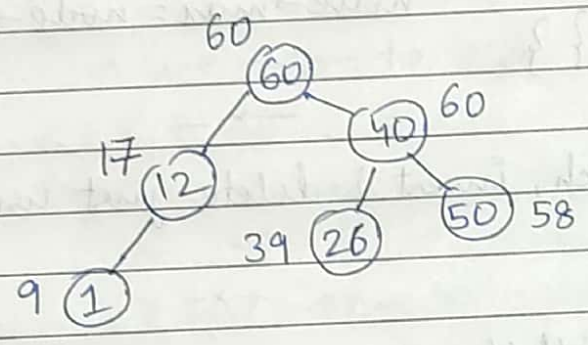
To answer this overlap query, we'll include a parameter 'max' in the structure of the node.

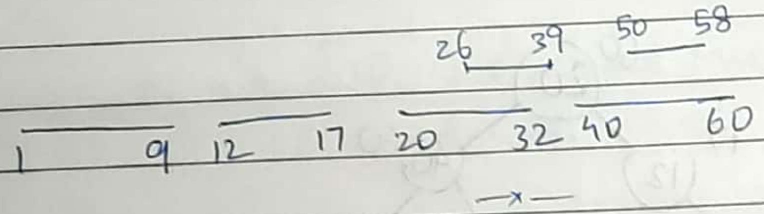
max ~~is~~ will be defined as :-

node → max = Max of :- $\begin{cases} \text{node} \to \text{left} \to \text{max} \\ \text{node} \to \text{high} \\ \text{node} \to \text{right} \to \text{max} \end{cases}$

'high' is the ~~hight~~ high end point of that node.

Ex:- max for each node is written beside the node.





—x—

(*) BBST structure

```
struct BST {
    long long int key;
    int height;
    long long int max, high;
    struct BST *left, *right;
};
```

→ After insert, delete or rotate; you have to update the 'max' of each node.

→
```
void Updatemax(struct BST *node){
    if(node)
    {
        node→max=node→high;

        if(node→left && node→left→max>node→high)
            node→max=node→left→max;

        if(node→right && node→right→max>node→max)
            node→max=node→right→max;
    }}
```

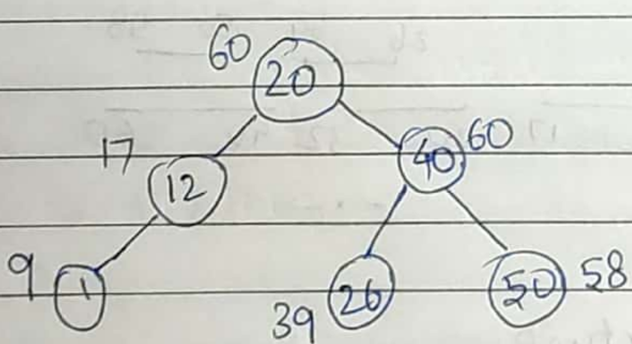
—*—

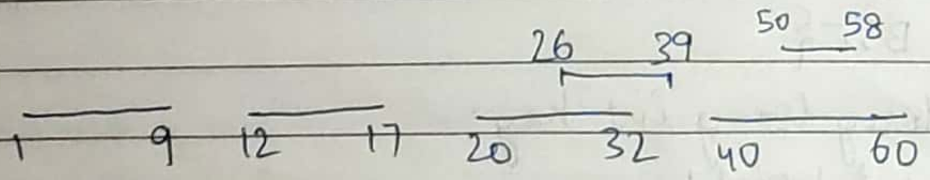
For search, insert & delete just use the lower end points.

—*—

Q. How will the overlap fn work?

Soln. Ex:-





★★ SIR explained the wrong algo but the code is right. The explanation written below is also right

1.) We need to check whether [10,11] overlaps with any interval or not.

→ We'll start at the root node

[10,11] does not overlap with [20,32]

& $10 < \textcircled{20} \rightarrow \overset{\text{left}}{\cancel{\text{node}}} \rightarrow \text{max}$

(i.e. 10 < ~~50~~ 17

⇒ we move to left

Again, [10,11] does not overlap with [12,17]

& $10 \overset{>}{\cancel{<}} \textcircled{12} \rightarrow \overset{\text{left}}{\cancel{\text{node}}} \rightarrow \text{max}$

i.e. $10 \overset{>}{\cancel{<}} \cancel{18}\ 9$

⇒ we move to ~~left~~ right which is a NULL pointer

~~[10,11] does not overlap with [1,9]~~
~~& 10 > ① → max i.e. 10 > 9~~
~~⇒ we move to right & reach a NULL pointer~~

∴ [10,11] does not overlap with any interval at all.

2.) Check whether [8,10] overlaps with any interval or not

→ We'll start at the root node

[8,10] does not overlap with [20,32]

& $8 < \textcircled{20} \rightarrow \overset{\text{left} \rightarrow \text{max}}{\cancel{\text{node}}}$ i.e. 8 < ~~50~~ 17

⇒ we move to left

[8, 10] does not overlap with [12, 17]

$\Rightarrow$ ~~we move~~ & $8 < \textcircled{12} \rightarrow \overset{\text{left} \rightarrow \text{max}}{\text{~~max~~}}$ i.e $8 < 9$

$\Rightarrow$ we move to left.

[8, 10] overlaps with [1, 9]

$\Rightarrow$ overlap fn will return '$\textcircled{1}$' node.

— * —

$\Rightarrow$ Overlap fn $\rightarrow$ O(log n)

$\rightarrow$ struct BST *Overlap(struct BST *node, long long int l, long long int r)

```
{
    while(node){
        if(l ≤ node→high && node→key ≤ r)
            return node;

        if(node→left && node→left→max ≥ l)
            node = node→left

        else node = node→right;
    }
    return NULL;
}
```

→ Proof as to why the overlap fn works correctly.

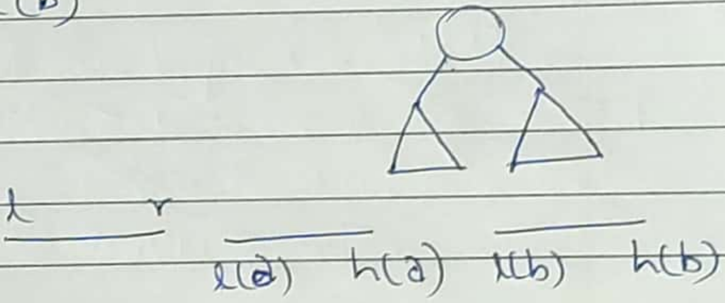
Date / /
Page

⟹ Correctness of the Overlap fn

1.) If we go to the right and did not find any overlap, then the interval cannot overlap with any interval on the left side.

→ when to the right b/c max on left < l, implies for any interval on the left side h(a) < l.

- Let a be a interval on left ~~wi~~ with left → max value
- The interval can't overlap with a.
- r < l(a) as l ≤ h(a)
- Let b be any interval on the right side, then l(a) < l(b)
- Hence r < l(a) < l(b)



2.) If we go to the left & did not find any ~~interval~~ overlap, ~~on the right side~~ then the interval can't overlap with any interval on the right side

- Let a be an interval on left with left → max value.
- The interval can't overlap with a
- r < l(a) as l ≤ h(a).