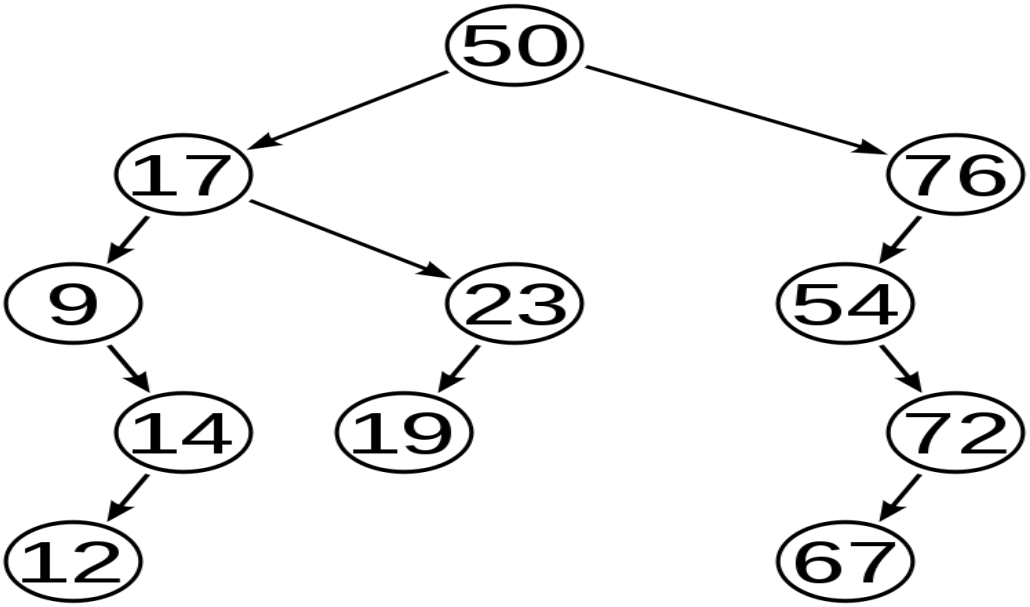
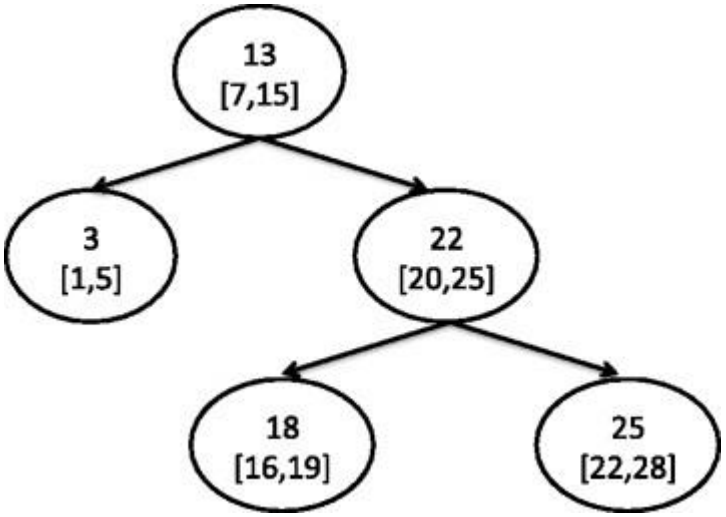
Interval tree

1. **Interval trees** are, a type of **Binary Search Tree (BST)** which carries different node structure/info then BST.

   Binary search tree



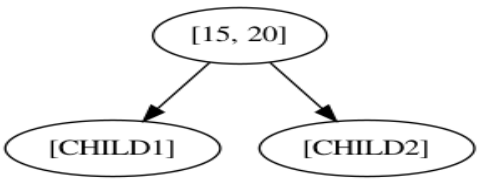


Interval tree

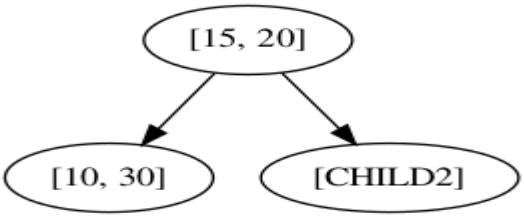
2. Interval Tree - Population / Insertion O(log N)

```
(15, 20), (10, 30), (17, 19), (5, 20), (12, 15), (30, 40)
```

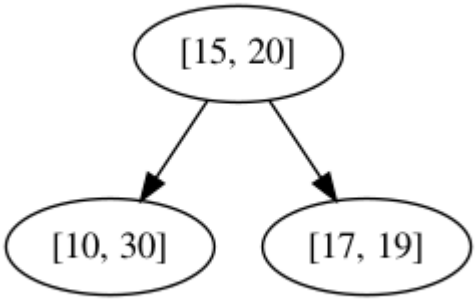
Our fist element is root element :



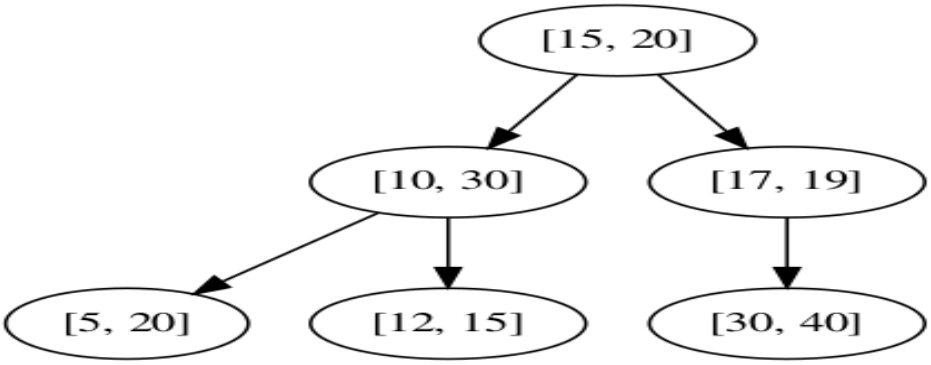
For the second element, we will compare its min value of the interval to the min value of the root node. If this min value is smaller than root node's min value than this node goes to the left of the root node or else right. So, here we have 15 and 10 where 10 is smaller than 15. So `(10, 30)` interval becomes left child of the root node.



For third element `(17, 19)`, as 17 is greater than 15 that's why this node will go on right.

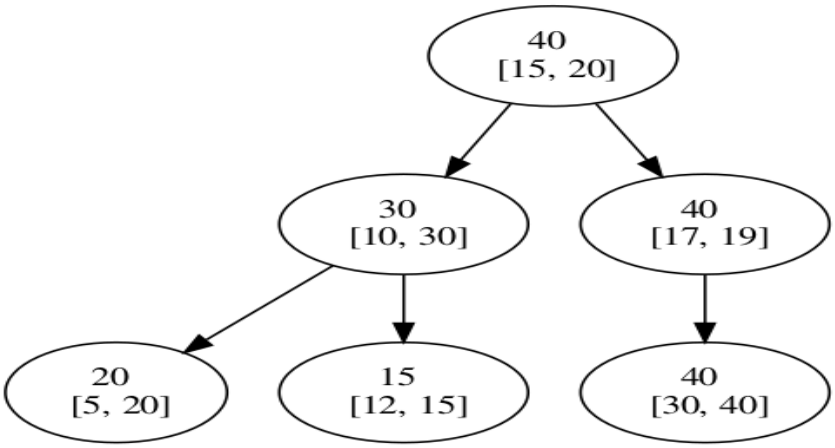


For the fourth element `(5, 20)`, 5 is smaller than 15 so we have to travel left. Now compare it will left child which is `(10, 30)`. Of course, 5 is smaller than 10 so this node will go on the left of `(10, 30)`. If do the same for all the remaining elements, your tree might look like this.



3. Max value of the sub-tree. Lets see how it can be calculated for each node in our tree.

For this, we have to start with the leaves, here the leaves are `(5, 20)`, `(12, 15)` and `(30, 40)`. Starting with `(5, 20)`, in order to find the max value of the sub-tree for this node, we have to check which is the greatest value in the sub-tree rooted with this node. Since it is a leaf, the max value will the max interval of that node. This is true of all the leaves in the tree.



- Max value is used for the deletion and searching of interval overlapping.

4. Interval Tree - Search O(log N)

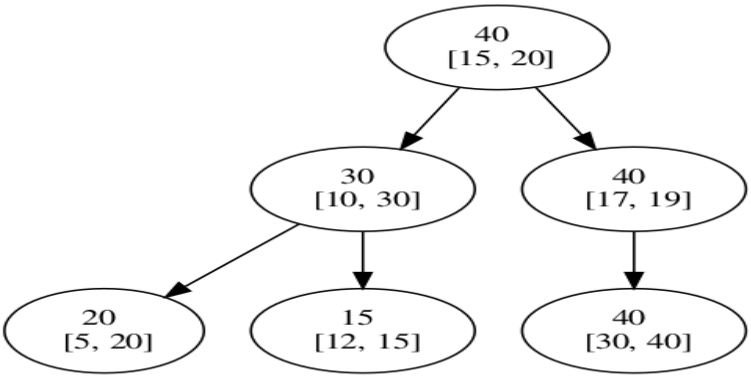
- First, we start with the root and check if there is an overlap, if it is then we will return true, if not then go ahead.
- If left child is not empty then check the max in the left child, if is greater then the query's minimum value, re-do these steps starting from left child.
- If above step does not satisfy then re-do these steps for right child.

Lets take an example, considering that we want to find the overlapping interval of `(6, 7)` exist in the tree.

Starting with the root node, and it does not overlap. So now will check if the min value of the query, which is 6, is greater than the Max value of the left child. Of course, 30 it is greater than 6 so we will follow the left child path.

We will check if `(6, 7)` overlap with the `(10, 30)`, and it not actually so we will go for its left child and check if the max value is greater than 6. 20 is greater than 6 so we will check if there is interval overlap between the `(5, 20)` and `(6, 7)`. Yay! its an overlap. The process here will return true or something positive.

Search (6,7)



5. Deleting tree node in Interval Tree O(log N)

1. Start with searching for the overlap which is same as the search operation discussed above.

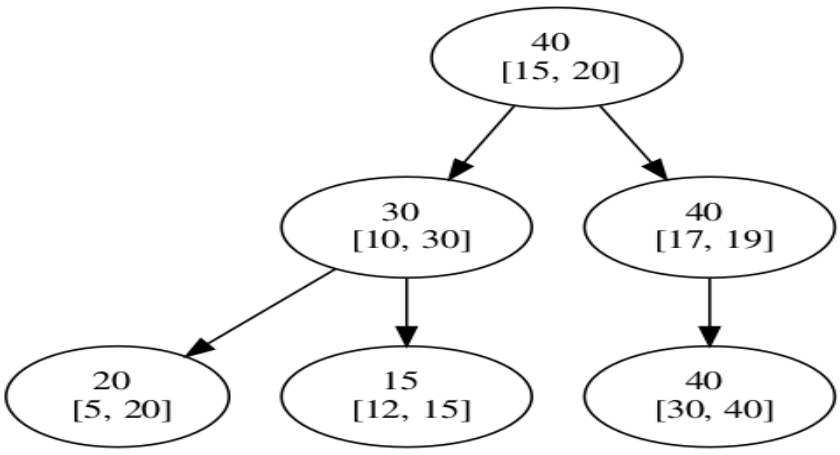
2. When you find the overlapping node, check if this node has child. If no child, then simply erase the connection with its parent.

3. But if this node has left child, then replace this node with left child with. Else, if it has only right child then replace it with it's right child.

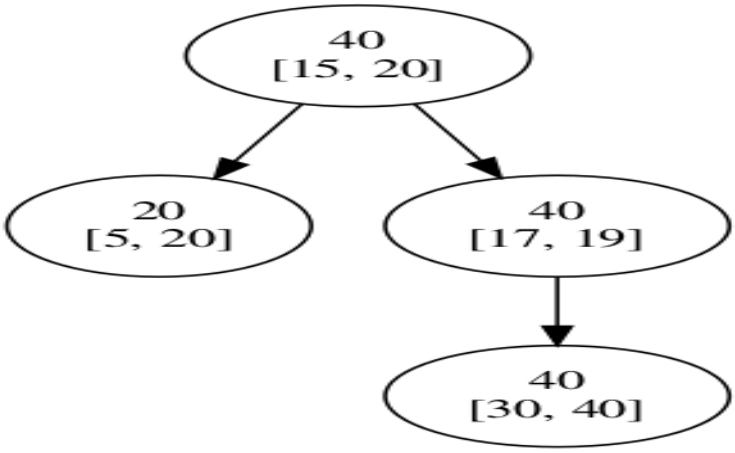
4. Complication may arise if the the deleting node has both the child and they also have their own subtree.

5. In this situation, replace the deleting node with the left child and re-insert the right child's subtree into the existing tree.

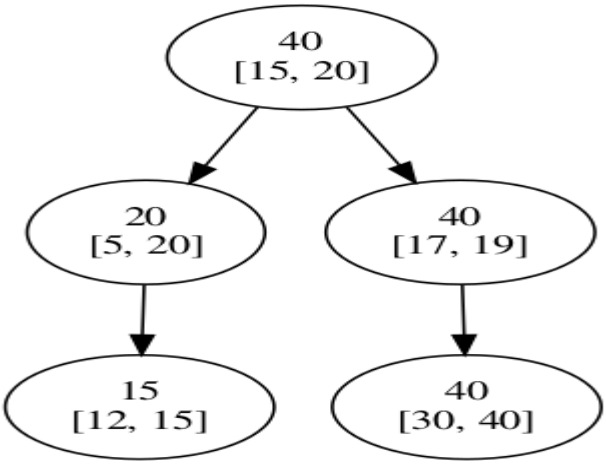
   Example : lets delete the node which intersect with `[9, 11]` on our old tree.



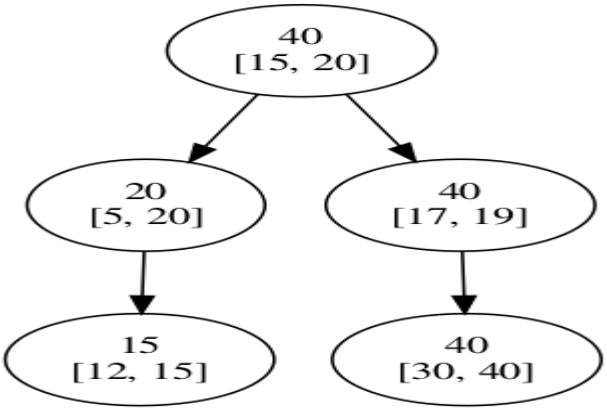
1.


40
[15, 20]
20
[5, 20]
40
[17, 19]
40
[30, 40]

2.


40
[15, 20]
20
[5, 20]
40
[17, 19]
15
[12, 15]
40
[30, 40]

3. Now we will reinsert the node `[12, 15]` in the tree, and our tree would look like this



Complexity

Time complexity of the different operations are:

- Insertion O(log N)
- Search P(log N)
- Delete O(log N)
- Initial creation: O(N * log N) as there will be N nodes

Applications of Interval Tree

Interval trees are mostly used when **dealing with geometry**.
For e.g. if you want to **find a point or line which is overlapping or enclosed in the rectangle**, then the interval tree makes it easy to find.
Putting it more formally, **Enclosing Interval Searching Problem**:
Given a set S of n intervals and a query point, q, report all those intervals containing q; and Overlapping Interval Searching Problem: Given a set S of n intervals and a query interval Q, report all those intervals in S overlapping Q.