Date _____ Page _____

Day - 167

BST - 3

* Construct BST from Preorder Traversal:

0	1	2	3	4	5
8	5	1	7	10	12

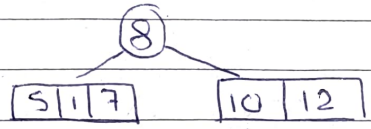
⇒ Preorder is → N L R.

⇒ So our first ~~node~~ ele. will be root Node.

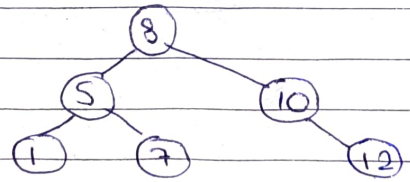
⇒ Now how to find left & right nodes.

⇒ It's very simple, we only have to find largest nodes than root node. that will come in right side & smaller nodes goes to left side.

⇒



⇒



⇒ We can also create tree by using our BST tree creation method by using array.

Date _____ Page _____

⇒ This method will take $O(n^2)$. T.C..

⇒ For solving in $O(n)$ —

⇒ we will find a range for every element that can come in that side.

Ex:

For the right side of 5 —

Element should be —

$5 <$ element < 8

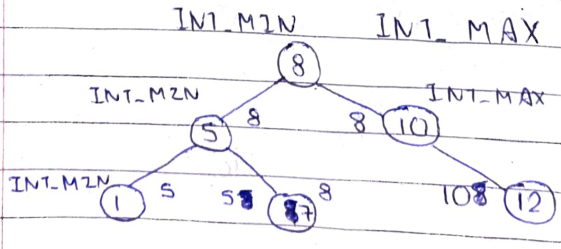
⇒ Now, how to find the range.

⇒ So, we will start with a range —

INT_MIN < element < INT_MAX

⇒ After that, for left INT_MIN < ele. < curr. node value.

⇒ For right, curr. node value < ele. < INT_MAX.



Date _____ Page _____

Code

```
Node * BST(vector<int>& preorder, int &index,
    int lower, int upper)
    if (index == preorder.size() ||
     preorder[index] < lower || preorder
      [index] > ~~right~~ upper)
      return ~~null~~ NULL;

    Node * root = new Node(preorder
                  [index++]);
    root->left = BST(preorder, index,
    lower, root -> data);
    root -> right = BST(preorder, index,
      root -> data, Upper);
    return root;
}
```

* Construct BST from 'post' order:

0	1	2	3	4	5
1	7	5	50	40	10

⇒ Here, we will do previous procedure in reverse.

⇒ we will find a range.

⇒ Then we will find or check for right side then left side by previous logic.

Date _____ Page _____

Code

```
Node * BST (int post[], int &index,
        int lower, int upper, int &size){
    if( index < 0 || post[index] <
      lower || post[index] > upper )
        return NULL;

    Node * root = new Node (post[index--]);
    root->right = BST(post, index,
        root->data, upper);
    root->left = BST(post, index,
        lower, root->data);
    return root;
}
```

* Preorder and BST:

2	4	3

⇒ We have given a preorder.

⇒ But there is a twist here that it can be of a BST Tree or not.

⇒ So, if it will create a correct BST Tree then return 1 otherwise 0.

⇒ In the first approach, we can create a BST Tree ~~the~~ from it then find the preorder.

Date _____ Page _____

⇒ If both preorder are equal then we can return 1 otherwise 0.

⇒ But this will take $O(n^2)$ T.C.

⇒ Now we can use our previous logic of lower & upper.

⇒ If the preorder is correct then our index will complete the array.

⇒ If not that means preorder is wrong.

⇒ index == size → return 1
index < size → return 0.

⇒ This approach will give segmentation fault in GFG.

⇒ This is due to the stack that used by GFG:

⇒ That stack can't handle large test cases at a time.

⇒ So, it will access ~~a~~ that memory location that we don't have the access to access.

⇒ Hence, it will give us segmentation fault.

⇒ But if we solve it by using iterative approach that will use stack, it will pass all the test cases.

Date _____ Page _____

⇒ Here, we use stack that uses heap memory.

⇒ And heap memory can easily handle all that input test cases.

⇒ So, how to do iterative approach —

⇒ As we done previous, first we push root then right then left.

⇒ In the same way, we will do here.

⇒ We will take two stack i.e. lower & upper.

⇒ We will store INT_MIN in lower & INT_MAX in upper.

⇒ Then we will push lower & upper range according to right then left.

Pseudo code

① If (arr[i] < lower.top())
    return 0;

② while(arr[i] > upper.top())
    pop upper & lower;

Date _____ Page _____

③. left = lower
right = upper
lower, upper pop
lower. push : arr[i]
upper. push : right
lower. push : left
upper. push : arr[i]