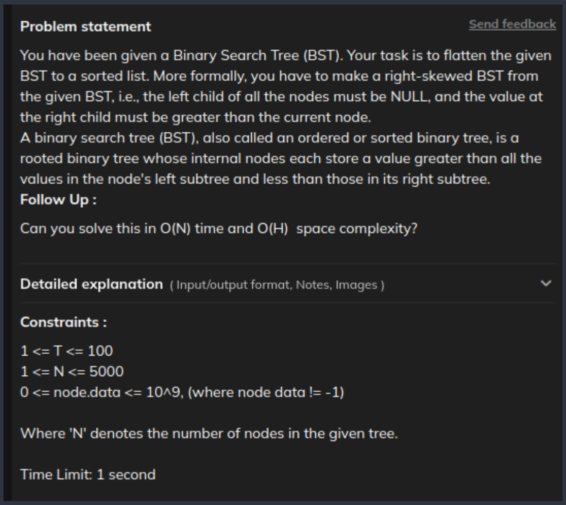
Flatten BST to Sorted Linked List



Problem statement Send feedback

You have been given a Binary Search Tree (BST). Your task is to flatten the given BST to a sorted list. More formally, you have to make a right-skewed BST from the given BST, i.e., the left child of all the nodes must be NULL, and the value at the right child must be greater than the current node.
A binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a value greater than all the values in the node's left subtree and less than those in its right subtree.
Follow Up :

Can you solve this in O(N) time and O(H) space complexity?

Detailed explanation (Input/output format, Notes, Images)

Constraints :

1 <= T <= 100
1 <= N <= 5000
0 <= node.data <= 10^9, (where node data != -1)

Where 'N' denotes the number of nodes in the given tree.

Time Limit: 1 second

→ Brute force :-

↳ Store the inorder of BST inside an array. (Morris traversal)

↳ Create a node for each element.

↳ Make left = NULL.

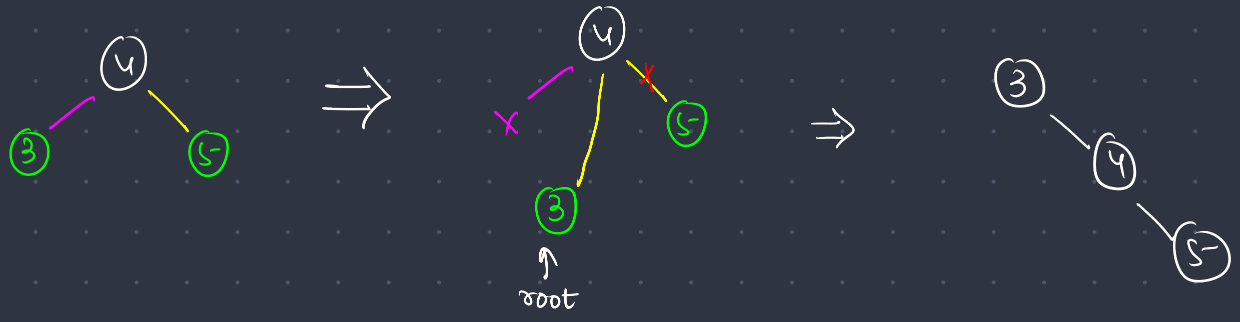
↳ & right point to next element of array.

```
TreeNode<int>* flatten(TreeNode<int>* root)
{
    vector<int> inorder;
    MorrisTraversal(root, inorder);
    int val = inorder[0];
    TreeNode<int>* head = new TreeNode(val);
    TreeNode<int>* node = head;
    int i = 1;
    while(i < inorder.size()) {
      TreeNode<int>* temp = new TreeNode(inorder[i++]);
      node->right = temp;
      node = temp;
    }
    return head;
}
```

Time Complexity : O(N)

Space Complexity : O(N)

→ Approach :-



```
TreeNode<int> *solve(TreeNode<int> *root) {
  if (!root || (!root→left && !root→right))
    return root;

  TreeNode<int> *left = solve(root→left);
  TreeNode<int> *right = solve(root→right);
  root→left = NULL;
  if (left) {
    TreeNode<int> *node = left;
    while (node→right ≠ NULL) {
      node = node→right;
    }
    node→right = root;
  }
  root→right = right;
  return left ? left : root;
}

TreeNode<int> *flatten(TreeNode<int> *root) {
  if (!root)
    return NULL;
  root = solve(root);
  return root;
}
```

Time Complexity : O(N)

Space Complexity : O(H)

Don't know why but it's giving TLE !!!