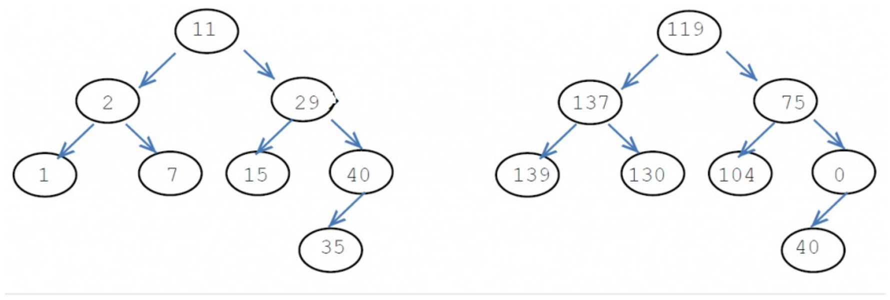
Transform a BST to greater sum tree

Given a BST, transform it into greater sum tree where each node contains sum of all nodes greater than that node.



We strongly recommend to minimize the gbrowser and try this yourself first.

Method 1 (Naive):

This method doesn't require the tree to be a BST. Following are steps.

1. Traverse node by node(Inorder, preorder, etc.)
2. For each node find all the nodes greater than that of the current node, sum the values. Store all these sums.
3. Replace each node value with their corresponding sum by traversing in the same order as in Step 1.

This takes O(n^2) Time Complexity.

Method 2 (Using only one traversal)

By leveraging the fact that the tree is a BST, we can find an O(n) solution. The idea is to traverse BST in reverse inorder. Reverse inorder traversal of a BST gives us keys in decreasing order. Before visiting a node, we visit all greater nodes of that node. While traversing we keep track of sum of keys which is the sum of all the keys greater than the key of current node.

```
// C++ program to transform a BST to sum tree
#include<iostream>
using namespace std;

// A BST node
struct Node
{
    int data;
    struct Node *left, *right;
};

// A utility function to create a new Binary Tree Node
struct Node *newNode(int item)
{
    struct Node *temp =  new Node;
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to transform a BST to sum tree.
// This function traverses the tree in reverse inorder so
// that we have visited all greater key nodes of the currently
// visited node
void transformTreeUtil(struct Node *root, int *sum)
{
   // Base case
   if (root == NULL)  return;

   // Recur for right subtree
   transformTreeUtil(root->right, sum);

   // Update sum
   *sum = *sum + root->data;
   // Store old sum in current node
   root->data = *sum - root->data;

   // Recur for left subtree
   transformTreeUtil(root->left, sum);
}

// A wrapper over transformTreeUtil()
void transformTree(struct Node *root)
{
    int sum = 0; // Initialize sum
    transformTreeUtil(root, &sum);
}

// A utility function to print indorder traversal of a
// binary tree
void printInorder(struct Node *root)
{
    if (root == NULL) return;

    printInorder(root->left);
    cout << root->data << " ";
    printInorder(root->right);
}

// Driver Program to test above functions
int main()
{
    struct Node *root = newNode(11);
    root->left = newNode(2);
    root->right = newNode(29);
    root->left->left = newNode(1);
    root->left->right = newNode(7);
    root->right->left = newNode(15);
    root->right->right = newNode(40);
    root->right->right->left = newNode(35);

    cout << "Inorder Traversal of given tree\n";
    printInorder(root);

    transformTree(root);

    cout << "\n\nInorder Traversal of transformed tree\n";
    printInorder(root);

    return 0;
}
```

Output:

```
Inorder Traversal of given tree
1 2 7 11 15 29 35 40

Inorder Traversal of transformed tree
139 137 130 119 104 75 40 0
```

Time complexity of this method is O(n) as it does a simple traversal of tree.