Merge Two Balanced Binary Search Trees

You are given two balanced binary search trees e.g., AVL or Red Black Tree. Write a function that merges the two given balanced BSTs into a balanced binary search tree. Let there be m elements in first tree and n elements in the other tree. Your merge function should take O(m+n) time.

In the following solutions, it is assumed that sizes of trees are also given as input. If the size is not given, then we can get the size by traversing the tree (See this).

Method 1 (Insert elements of first tree to second)
Take all elements of first BST one by one, and insert them into the second BST. Inserting an element to a self balancing BST takes Logn time (See this) where n is size of the BST. So time complexity of this method is Log(n) + Log(n+1) … Log(m+n-1). The value of this expression will be between mLogn and mLog(m+n-1). As an optimization, we can pick the smaller tree as first tree.

Method 2 (Merge Inorder Traversals)
1) Do inorder traversal of first tree and store the traversal in one temp array arr1[]. This step takes O(m) time.
2) Do inorder traversal of second tree and store the traversal in another temp array arr2[]. This step takes O(n) time.
3) The arrays created in step 1 and 2 are sorted arrays. Merge the two sorted arrays into one array of size m + n. This step takes O(m+n) time.
4) Construct a balanced tree from the merged array using the technique discussed in this post. This step takes O(m+n) time.

Time complexity of this method is O(m+n) which is better than method 1. This method takes

O(m+n) time even if the input BSTs are not balanced.
Following is C++ implementation of this method.

```
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

// A utility unction to merge two sorted arrays into one
int *merge(int arr1[], int arr2[], int m, int n);

// A helper function that stores inorder traversal of a tree in inorde
void storeInorder(struct node* node, int inorder[], int *index_ptr);

/* A function that constructs Balanced Binary Search Tree from a sorte
   See http://www.geeksforgeeks.org/archives/17138 */
struct node* sortedArrayToBST(int arr[], int start, int end);

/* This function merges two balanced BSTs with roots as root1 and root
   m and n are the sizes of the trees respectively */
struct node* mergeTrees(struct node *root1, struct node *root2, int m,
{
    // Store inorder traversal of first tree in an array arr1[]
    int *arr1 = new int[m];
    int i = 0;
    storeInorder(root1, arr1, &i);

    // Store inorder traversal of second tree in another array arr2[]
    int *arr2 = new int[n];
    int j = 0;
    storeInorder(root2, arr2, &j);

    // Merge the two sorted array into one
    int *mergedArr = merge(arr1, arr2, m, n);

    // Construct a tree from the merged array and return root of the t
    return sortedArrayToBST (mergedArr, 0, m+n-1);
}

/* Helper function that allocates a new node with the
```


Popular Posts

```
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

// A utility function to print inorder traversal of a given binary tre
void printInorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}

// A utility unction to merge two sorted arrays into one
int *merge(int arr1[], int arr2[], int m, int n)
{
    // mergedArr[] is going to contain result
    int *mergedArr = new int[m + n];
    int i = 0, j = 0, k = 0;

    // Traverse through both arrays
    while (i < m && j < n)
    {
        // Pick the smaler element and put it in mergedArr
        if (arr1[i] < arr2[j])
        {
            mergedArr[k] = arr1[i];
            i++;
        }
        else
        {
            mergedArr[k] = arr2[j];
            j++;
```

```
        }
        k++;
    }

    // If there are more elements in first array
    while (i < m)
    {
        mergedArr[k] = arr1[i];
        i++; k++;
    }

    // If there are more elements in second array
    while (j < n)
    {
        mergedArr[k] = arr2[j];
        j++; k++;
    }

    return mergedArr;
}

// A helper function that stores inorder traversal of a tree rooted wi
void storeInorder(struct node* node, int inorder[], int *index_ptr)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    storeInorder(node->left, inorder, index_ptr);

    inorder[*index_ptr] = node->data;
    (*index_ptr)++;  // increase index for next entry

    /* now recur on right child */
    storeInorder(node->right, inorder, index_ptr);
}

/* A function that constructs Balanced Binary Search Tree from a sorte
   See http://www.geeksforgeeks.org/archives/17138 */
struct node* sortedArrayToBST(int arr[], int start, int end)
{
    /* Base Case */
    if (start > end)
      return NULL;

    /* Get the middle element and make it root */
    int mid = (start + end)/2;
```

695

Subscribe

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 37 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 57 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 57 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 3 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 3 hours ago

```
    struct node *root = newNode(arr[mid]);

    /* Recursively construct the left subtree and make it
       left child of root */
    root->left =  sortedArrayToBST(arr, start, mid-1);

    /* Recursively construct the right subtree and make it
       right child of root */
    root->right = sortedArrayToBST(arr, mid+1, end);

    return root;
}

/* Driver program to test above functions*/
int main()
{
    /* Create following tree as first balanced BST
           100
         /  \
       50    300
      / \
    20  70
    */
    struct node *root1  = newNode(100);
    root1->left         = newNode(50);
    root1->right        = newNode(300);
    root1->left->left   = newNode(20);
    root1->left->right  = newNode(70);

    /* Create following tree as second balanced BST
            80
          /  \
        40   120
    */
    struct node *root2  = newNode(80);
    root2->left         = newNode(40);
    root2->right        = newNode(120);

    struct node *mergedTree = mergeTrees(root1, root2, 5, 3);

    printf ("Following is Inorder traversal of the merged tree \n");
    printInorder(mergedTree);

    getchar();
    return 0;
}
```

Output:

```
Following is Inorder traversal of the merged tree
20 40 50 70 80 100 120 300
```

Method 3 (In-Place Merge using DLL)
We can use a Doubly Linked List to merge trees in place. Following are the steps.

1) Convert the given two Binary Search Trees into doubly linked list in place (Refer this post for this step).
2) Merge the two sorted Linked Lists (Refer this post for this step).
3) Build a Balanced Binary Search Tree from the merged list created in step 2. (Refer this post for this step)

Time complexity of this method is also O(m+n) and this method does conversion in place.

Thanks to Dheeraj and Ronzii for suggesting this method.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

Writing code in comment? Please use **ideone.com** and share the link here.

11 Comments GeeksforGeeks

Sort by Newest

Akshay Johri • 4 months ago

How about..
1. convert both the trees to threaded binary O(n). So now inorder traversal car
2. Merge the two DLLs O(2n)
3 .Construct the tree O(2n)

Bannhi Barua • 11 months ago

another method proposal.
void insert (struct node **a, int data).
{

printf(" %d called from ", data),.

struct node *temp = *a,*prev;.

while(temp!=NULL).

{.

prev = temp;.

if(temp->data > data).

temp = temp->left;.

else if (temp->data < data).

temp = temp->right;

see more

• Reply • Share ›

sachin • 2 years ago

Can you please post -how to find the lca of two nodes in n-ary tree?

• Reply • Share ›

Prashant • 2 years ago

I have an algorithm for merging two BSTs, it compares values at the root of th accordingly. Here it is:

```
 void merge (node* root1, node* root2)
{
        if(root1 == null)
        {
                root1 = root2;
                return;
```

```
}
if(root2 == null)
        return;
if(root1->data < roo2->data)
{
        node* ptr = root2->left;
        root2->left = NULL;
        merge(root1->right,root2);
        merge(root1,ptr);
```

see more

Reply • Share ›

kartik ➦ Prashant • 2 years ago

Sometimes it is easier to get the time complexity directly by applying in
is clear that the total number of comparisons will be equal to number o
nodes in second tree. So, we can say that the complexity is O(m+n).

Reply • Share ›

leet ➦ kartik • 2 years ago

Can you please explin how it is O(m+n).I am not getting why th

```
/* Paste your code here (You may delete these lines if
```

Reply • Share ›

camster • 2 years ago

Hi everybody, Here is my recursive algorithm for merging two 2 n-ary trees (w
you find any mistakes or can optimize this better. Thank you camster.

```
struct RTreenode {
        int value;
        struct RTreenode *children[256];
```

```
};


 // Merge tree root2(aka currtwo) into tree currone
 // recursively destroy tree root2(akac  currtwo)
// in O(n + m) time complexity

void Helper(RTreenode*& root2,
            RTreenode*& currone,
            RTreenode*& currtwo){
        if (currone == NULL || currtwo == NULL){
                return;
```

see more

• Reply • Share ›

GeeksforGeeks • 2 years ago

@Ronzii & @Dheeraj: Thanks for suggesting a new method. We will add it to

• Reply • Share ›

Ronzii • 2 years ago

We could also convert both balanced binary search tree's to doubly linked lists simply merge both linked lists O(m+n) and convert them back into a balanced The complexity remains the same except everything is inplace!

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

Dheeraj • 2 years ago

1.We can convert the two tree into doubly linked list.(sorted).

2.We can merge the two sorted list.
and then
3.We can build a tree out of it.

i guess that wont take xtra space.And all the three methods..are already on the

camster • 2 years ago

I recently solved a problem where I was asked to merge 2 n-ary trees (where n
C++ to do this. However, I did not use n in-order traversals and and a two arra
wondering what the best way to merge two n-ary trees where n > 2. Thank you

```
/* Paste your code here (You may delete these lines if not writing co
```