Sorted Linked List to Balanced BST

Given a Singly Linked List which has data members sorted in ascending order. Construct a Balanced Binary Search Tree which has same data members as the given Linked List.

Examples:

```
Input:  Linked List 1->2->3
Output: A Balanced BST
     2
   /  \
  1    3


Input: Linked List 1->2->3->4->5->6->7
Output: A Balanced BST
        4
      /   \
     2     6
   /  \   / \
  1   3  4   7

Input: Linked List 1->2->3->4
Output: A Balanced BST
      3
    /  \
   2    4
 /
```


```
1

Input:  Linked List 1->2->3->4->5->6
Output: A Balanced BST
      4
    /   \
   2     6
 /  \   /
1   3  5
```

Method 1 (Simple)
Following is a simple algorithm where we first find the middle node of list and make it root of the tree to be constructed.

```
1) Get the Middle of the linked list and make it root.
2) Recursively do same for left half and right half.
       a) Get the middle of left half and make it left child of the root
          created in step 1.
       b) Get the middle of right half and make it right child of the
          root created in step 1.
```

Time complexity: O(nLogn) where n is the number of nodes in Linked List.

See this forum thread for more details.

Method 2 (Tricky)
The method 1 constructs the tree from root to leaves. In this method, we construct from leaves to root. The idea is to insert nodes in BST in the same order as the appear in Linked List, so that the tree can be constructed in O(n) time complexity. We first count the number of nodes in the given Linked List. Let the count be n. After counting nodes, we take left n/2 nodes and recursively construct the left subtree. After left subtree is constructed, we allocate memory for root and link the left subtree with root. Finally, we recursively construct the right subtree and link it with root. While constructing the BST, we also keep moving the list head pointer to next so that we have the appropriate pointer in each recursive call.
Following is C implementation of method 2. The main code which creates Balanced BST is

Popular Posts

highlighted.

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

/* Link list node */
struct LNode
{
    int data;
    struct LNode* next;
};

/* A Binary Tree node */
struct TNode
{
    int data;
    struct TNode* left;
    struct TNode* right;
};

struct TNode* newNode(int data);
int countLNodes(struct LNode *head);
struct TNode* sortedListToBSTRecur(struct LNode **head_ref, int n);


/* This function counts the number of nodes in Linked List and then ca
   sortedListToBSTRecur() to construct BST */
struct TNode* sortedListToBST(struct LNode *head)
{
    /*Count the number of nodes in Linked List */
    int n = countLNodes(head);

    /* Construct BST */
    return sortedListToBSTRecur(&head, n);
}

/* The main function that constructs balanced BST and returns root of
       head_ref -->  Pointer to pointer to head node of linked list
       n  --> No. of nodes in Linked List */
struct TNode* sortedListToBSTRecur(struct LNode **head_ref, int n)
{
    /* Base Case */
    if (n <= 0)
        return NULL;

    /* Recursively construct the left subtree */
    struct TNode *left = sortedListToBSTRecur(head_ref, n/2);
```

```
    /* Allocate memory for root, and link the above constructed left
       subtree with root */
    struct TNode *root = newNode((*head_ref)->data);
    root->left = left;

    /* Change head pointer of Linked List for parent recursive calls *
    *head_ref = (*head_ref)->next;

    /* Recursively construct the right subtree and link it with root
      The number of nodes in right subtree  is total nodes - nodes in
      left subtree - 1 (for root) which is n-n/2-1*/
    root->right = sortedListToBSTRecur(head_ref, n-n/2-1);

    return root;
}
```

```
/* UTILITY FUNCTIONS */

/* A utility function that returns count of nodes in a given Linked Li
int countLNodes(struct LNode *head)
{
    int count = 0;
    struct LNode *temp = head;
    while(temp)
    {
        temp = temp->next;
        count++;
    }
    return count;
}

/* Function to insert a node at the beginging of the linked list */
void push(struct LNode** head_ref, int new_data)
{
    /* allocate node */
    struct LNode* new_node =
        (struct LNode*) malloc(sizeof(struct LNode));
    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
```


Recent Comments

```
    (*head_ref)    = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct LNode *node)
{
    while(node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct TNode* newNode(int data)
{
    struct TNode* node = (struct TNode*)
                         malloc(sizeof(struct TNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return node;
}

/* A utility function to print preorder traversal of BST */
void preOrder(struct TNode* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preOrder(node->left);
    preOrder(node->right);
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct LNode* head = NULL;

    /* Let us create a sorted linked list to test the functions
     Created linked list will be 1->2->3->4->5->6->7 */
    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
```

```
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    printf("\n Given Linked List ");
    printList(head);

    /* Convert List to BST */
    struct TNode *root = sortedListToBST(head);
    printf("\n PreOrder Traversal of constructed BST ");
    preOrder(root);

    return 0;
}
```

Time Complexity: O(n)

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

Related Tpoics:

- Given a linked list, reverse alternate nodes and append at the end

- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List
- Delete N nodes after M nodes of a linked list
- Design a stack with operations on middle element
- Swap Kth node from beginning with Kth node from end in a Linked List

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

57 Comments GeeksforGeeks

Sort by Newest

Abhishek • 3 months ago

Awesome solution

Gaurav Baingalia • 7 months ago

can some one help me in this code ..>>

having prblm in sorting out the error..>>

```
#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;
```

```
int i=0;

struct lnode

{

int item;

struct lnode* next;
```

see more

Reply • Share ›

BePositive • 8 months ago

Method 2 is also wrong based on the same logic as told in earlier comment.

1 Reply • Share ›

BePositive • 8 months ago

Method 1 is wrong. It doesn't work for 9 elements 1,2,....,9
In order to make it right, we first need to find the number of elements which are
balanced BST. And then only, we can apply method 1 to those.
Let's say n be the number of elements. And x be the number of elements less
balanced BST (e.g. 1 or 3 or 7), then,

x = pow(2, ceil(log2(n))) - 1.

e.g. for n = 9, x = 7. n = 5, x = 3

Now we need to create complete balanced BST using these x elements (using
then need to add the (n - x) elements to the leaf nodes, using post order traver

Reply • Share ›

BePositive • 8 months ago

Order of the first solution should be O(N2) and not O(nlogn) as we are not disc

• Reply • Share ›

Guest ➚ BePositive • 5 months ago

Yes, had the list been doubly linked list, complexity is O(nlogn)else it is O(n^2+nlogn) which is nothing but O(n^2)....

• Reply • Share ›

Swastik Sahu • 9 months ago

How is the complexity of first method O(n log n) ?
If we create a hash of pointers to each node, i.e- hash[1] = pointer to 1st node

Then it can be done in O(n).

• Reply • Share ›

Sanjith Sakthivel • 9 months ago

```
node* sortll2bst(node* head).
{
node *curr=head;
while(curr->next!=NULL)
curr=curr->next;
int n=count(head);
return sortll2bstfunc(head, curr, n);
}
int count(node* head).
{
int c=0;
while(head!=NULL)
{
head=head->next;
c++;
}
}
```

node * sortll2bstfunc(node * start, node* end, int n).

see more

• Reply • Share ›

Akshay Jindal • 10 months ago

The C implementation of Algorithm1--->.

Node *middle(Node *p, Node *r).
{
if(p<r)
{
Node *s=p;Node *m=p;.
Node *q=p->next->next;.
while(q->item<=r->item||q!=NULL).
{.

m=p;.

p=p->next;.

q=q->next->next;.
}.
root=p;.
root->left=middle(s, m);.

see more

• Reply • Share ›

Vartul Gupta • 10 months ago

struct treeNode* makeTreeFromList(struct node*x, int length).
{
if(x==NULL)
return NULL;

return NULL;
if(length == 0).
return NULL;
else
{
struct node*temp = x;.
int counter = length/2;.
int i=0;
while(i<counter)
{
temp= temp->link;
i++;
}
struct treeNode*ans=NULL;
if(temp!=NULL)

see more

• Reply • Share ›

Santunu Patro • 10 months ago

acha acha

• Reply • Share ›

Himanshu • 10 months ago

```
 /* Method 1 Implementation */
#include<stdio.h>
#include<stdlib.h>
struct LNode
{
    int data;
    struct LNode* next;
};
```

```
struct TNode
{
    int data;
    struct TNode* left;
    struct TNode* right;
};

struct TNode* newNode(int data);
```

see more

• Reply • Share ›

Nirdesh Mani Sharma. • 11 months ago

In the second example, the tree created from 1->2->3->4->5->6->7 seems inc
repeated twice (5 is missing).

• Reply • Share ›

hsg92 • 11 months ago

Below I have coded Method-2 in Java, but it gives different definitely incorrect a
wrong?

```
/* Paste your code here (You may delete these lines if not writing c
        private TreeNode listTotreeBottomUp(Node start, int n){
                if ( n <= 0 ) return null;
                TreeNode left = listTotreeBottomUp(start, n/2);
                TreeNode root = new TreeNode(start.data);
                root.left = left;
                start = start.next;
                root.right = listTotreeBottomUp(start, n-n/2-1);
                return root;
        }
```

```
public void listTotreeBottomUp(){
        TreeNode root = listTotreeBottomUp(head, length(head)
        System.out.println("\nBottomUp\nInorder: ");
        inorder(root);
        System.out.println("\nPreorder: ");
```


Linuxwc ➦ hsg92 • 10 months ago

I have not seen your implementation but it seems that pointers are beir
pointers. The C++ code that corresponds your Java code uses *head_
when **head_ref should be used.

Each activation of your subroutine
Makes a left subtree that starts from the node "start" (and puts its data
Makes a root node that again has the data of the node "start"
Makes a right subtree that starts from the node next to "start" (and pos
node may already be in the left subtree)

So the original start-node of your tree or subtree may be in the tree sev
for the next node and so on.

One solution is to use a pointer "startptr" as a class variable, initialize i
as in the code below

```
TreeNode listTotreeBottomUp(Node start, int n){
       if ( n <= 0 ) return null;
       int leftsize = sizeofleft(n);
```

see more

Sandeep Jain • a year ago

Thanks for pointing this out. We have updated the comment.

• Reply • Share ›

Sarthak Mall 'shanky' • a year ago

I think in the comments the list is written in reverse.. 7->6->5->4->3->2->1
It should be 1->2->3->4->5->6->7.

• Reply • Share ›

Xiaoge Yuan • a year ago

neat.

• Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

• Reply • Share ›

Vallabh Patade • a year ago

Passing (n-n/2-1) as second argument while building right sub tree, why not tc

• Reply • Share ›

Ankit Jain • a year ago

In above code, in the recursive calls instead of passing head_ref we need to p
the method has a double pointer as its first argument..

• Reply • Share ›

Pavan • a year ago

In the posted program, don't we need to include another base case as if((*hea
Please comment if this becomes a redundant base case.
Thanks

• Reply • Share ›

Faisal • 2 years ago

```
[sourcecode language="C++"]
#include<iostream>
using namespace std;
class Node
{
public:int key;
Node* next;
Node(int k,Node* n = 0)
{
key = k;
next = n;
}
};
class Tnode
{
public:int data;
Tnode* left;
Tnode* right;
```

see more

Reply • Share ›

Ankit Gangal • 2 years ago

I think the complexity for solution 1 is O(n) where n is the size of the linked list.
go through n-1 recursive function calls and 1 call to the first function that starts

In merge sort we have all the n elements on each of log(n) levels .. so in that c
here we have 2^(h-1) elements on each 1 of h = log(n) levels ... so it is not the

Please tell me if i m wrong..

Reply • Share ›

venu gopal • 2 years ago

Can you please show me a code which doesnot uses pointer to a pointer ** ie

Chiranjeev Kumar • 2 years ago

```
 /* Paste your code here (You may delete these lines if not writing c
// Selection Sort
#include<stdio.h>
typedef struct node
{
    int value;
    struct node *next;
}mynode;
void add(mynode **head,int data)
{
    mynode *temp = (mynode *)malloc(sizeof(struct node));
    temp->value = data;
    temp->next = NULL;
    mynode *t = *head;
    if(!t)
    {
        printf("..............Creating SLL.......\n");
        *head = temp;
```

see more

Mady • 2 years ago

```
 //1. Let pointer s be the middle of the list. find using fast,slow
//left = head;
// right = s;
```

```
void list_2_BST(struct node *left,struct node **right){

    if(left == NULL)
       return;

     if(left == s){
       createTree(left);
       return ;
     }
     list_2_BST(left->next,right);
     *right = (*right)->next;
     createTree(left);
     createTree(right);
}
```

^ | v • Reply • Share ›

shiv • 2 years ago

y this code is not working if i take a global *head and call function sortListToBS
struct TNode* sortedListToBSTRecur(struct LNode *head_ref, int n)and also r
as head->data...i have done all the progrms by taking head as global y it is not

^ | v • Reply • Share ›

kartik ➚ shiv • 2 years ago

Please post the code that your tried.

^ | v • Reply • Share ›

shiv ➚ kartik • 2 years ago

here is my code

#include

```
#include
#include
struct node{
int data;
struct node *next;
}*head=NULL;
struct tree{
int data;
struct tree *left;
struct tree *right;
}*root=NULL;
int i=1;
struct node *build()
{
int n;
```

see more

• Reply • Share ›

shiv shiv • 2 years ago

above root->left=left is.....

if(ndata=head->data;

sorry by mistake it was copd wrongly

• Reply • Share ›

saniaz • 2 years ago

I think in method2 base case should be

```
/* Base Case */
if (n == 0)
return NULL;
```

instead of

/* Base Case */
if (n <= 0)
return NULL;

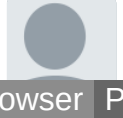
Correct me if I am wrong.

saniaz • 2 years ago

Type error!!

Input: Linked List 1->2->3->4->5->6->7
Output: A Balanced BST
4
/ \
2 6
/ \ / \
1 3 4 7

It should be
Input: Linked List 1->2->3->4->5->6->7
Output: A Balanced BST
4
/ \
2 6
/ \ / \
1 3 5 7



avinash • 2 years ago

```
I think there is no need of allocating using new for root, since the
```

```
    /* Allocate memory for root, and link the above constructed left
        subtree with root */
    struct TNode *root = *head_ref;
    root->left = left;

    /* Change head pointer of Linked List for parent recursive calls
    *head_ref = (*head_ref)->next;
I tried it & it worked
```

Reply • Share ›

Ravi • 2 years ago

Method 2 - Awesome.

Reply • Share ›

abbie • 2 years ago

i tried the second method, but every time it is giving one node less in the corre
-
1->2->3->4->5->6

i am getting o/p as-
3 2 1 5 4 (this is preorder)
but 6 is missing in the tree.
please check the code and clarify me whether i am making a mistake or this c

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

Reply • Share ›

kartik ➦ abbie • 2 years ago

There is no problem in the code. See this run. Going forward, please c
comment like this. That saves everybody's time.

Reply • Share ›

abbie kartik • 2 years ago

thanx

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

Reply • Share ›

naveen • 2 years ago

your solution is wrong.... as in linked list we can not use binary search or strict element at one... we need to traverse the linked list.. also we can not go back i complexity is o(nlogn) ??

Reply • Share ›

GeeksforGeeks naveen • 2 years ago

@naveen: Please take a closer look at algorithm. It doesn't use Binary of time complexity calculation.

1 Reply • Share ›

pb • 2 years ago

what if we need to do the task for doubly list that too in place i.e. converting ne pointers....any thoughts??

Reply • Share ›

GeeksforGeeks pb • 2 years ago

@pb: This has been published as separate post. Please see In-place c BST

Reply • Share ›

kunalgupta1991 • 2 years ago

@ I found second method wrong... for 1,2,3,4,5 it will construct.

3
/ \
2 4
/ /
1 5

which is not a bst
this is because we always focus on constructing left subtree first
please correct if i am wrong.

• Reply • Share ›

Aaman ➚ kunalgupta1991 • a year ago

I agree wid kunal, trying hard to dry run but right subtree seems wrong

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

• Reply • Share ›

Linuxwc ➚ kunalgupta1991 • a year ago

If you want a complete tree (other levels full and nodes in the last level
left subtree is not always n/2. See the recursive calls in the sortedListT
function sizeofleft(int) below to calculate the left subtree is probably not

```
 include <math.h>  add this line
int sizeofleft (int treesize) {
    // Size of left subtree in a complete tree

    if (treesize<=1) return(0);
    // If at most 1 node, there is no left subtree

    int levelsabovelast = floor(log(treesize)/log(2));
    // At least 1
```

```
int firstinlastlevel = pow(2,levelsabovelast);
/* Other left nodes in the last level:
 if n >= nodes in the previous levels + 1 + half of the ma
 then previous power of two - 1 (-1 since the first alread
```

see more

Reply • Share ›

Avinash kunalgupta1991 • 2 years ago

@kunal: I think u misunderstood the algorithm. It will construct followin

Preorder:- 3,2,1,5,4

Please check on the algorithm once again.
Advice:- Always dry run an algorithm to understand better.

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

Reply • Share ›

Karthick • 2 years ago

Method 2 is cool. Took me a long time to understand it. :)
Code,written beautifully.
Thanks.

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

Reply • Share ›

geeky • 2 years ago

Guys,
You are doing good job. But if you can provide tutorials on how to find the com

relation then it will really helpful for the beginners like me. Also please suggest to get started with it

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

• Reply • Share ›

randy geeky • 2 years ago

Introduction to Algorithms and Concrete Mathematics

• Reply • Share ›

randy • 2 years ago

method 2, beautiful code!

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your site