



Home

Algo

DS

Interview

Students

C

C++

Java

Python

Contribute

Ask Q

AndroidApp

GBlog

GFact

Jobs

Arr

String

Matrix

LinkedList

Stack

Q

Hash

Heap

Tree

BST

Graph

C/C++

Bit

MCQ

Misc

O/P

AVL Tree | Set 2 (Deletion)

We have discussed AVL insertion in the [previous post](#). In this post, we will follow a similar approach for deletion.

Steps to follow for deletion.

To make sure that the given tree remains AVL after every deletion, we must augment the standard BST delete operation to perform some re-balancing. Following are two basic operations that can be performed to re-balance a BST without violating the BST property ($\text{keys}(\text{left}) < \text{key}(\text{root}) < \text{keys}(\text{right})$). 1) Left Rotation 2) Right Rotation

T1, T2 and T3 are subtrees of the tree rooted with y (on left side)
or x (on right side)

y

x



GeeksforGeeks

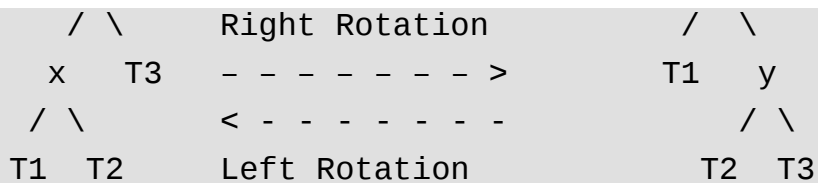


Like Page

128K likes

Be the first of your friends to like this





Keys in both of the above trees follow the following order

$\text{keys}(T1) < \text{key}(x) < \text{keys}(T2) < \text{key}(y) < \text{keys}(T3)$

So BST property is not violated anywhere.

Let w be the node to be deleted

- 1) Perform standard BST delete for w .
- 2) Starting from w , travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z , and x be the larger height child of y . Note that the definitions of x and y are different from [insertion](#) here.
- 3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z . There can be 4 possible cases that needs to be handled as x , y and z can be arranged in 4 ways. Following are the possible 4 arrangements:
 - a) y is left child of z and x is left child of y (Left Left Case)
 - b) y is left child of z and x is right child of y (Left Right Case)
 - c) y is right child of z and x is right child of y (Right Right Case)
 - d) y is right child of z and x is left child of y (Right Left Case)

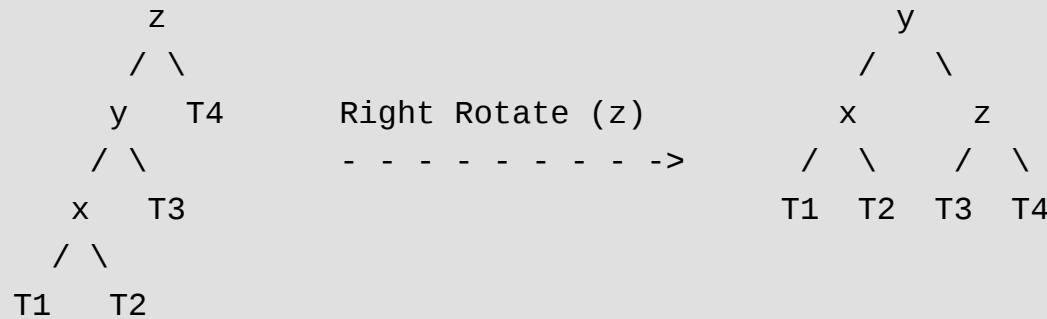
Like insertion, following are the operations to be performed in above mentioned 4 cases. Note that, unlike insertion, fixing the node z won't fix the complete AVL tree. After fixing z , we may have to fix ancestors of z as well (See [this video lecture](#) for proof)

a) Left Left Case

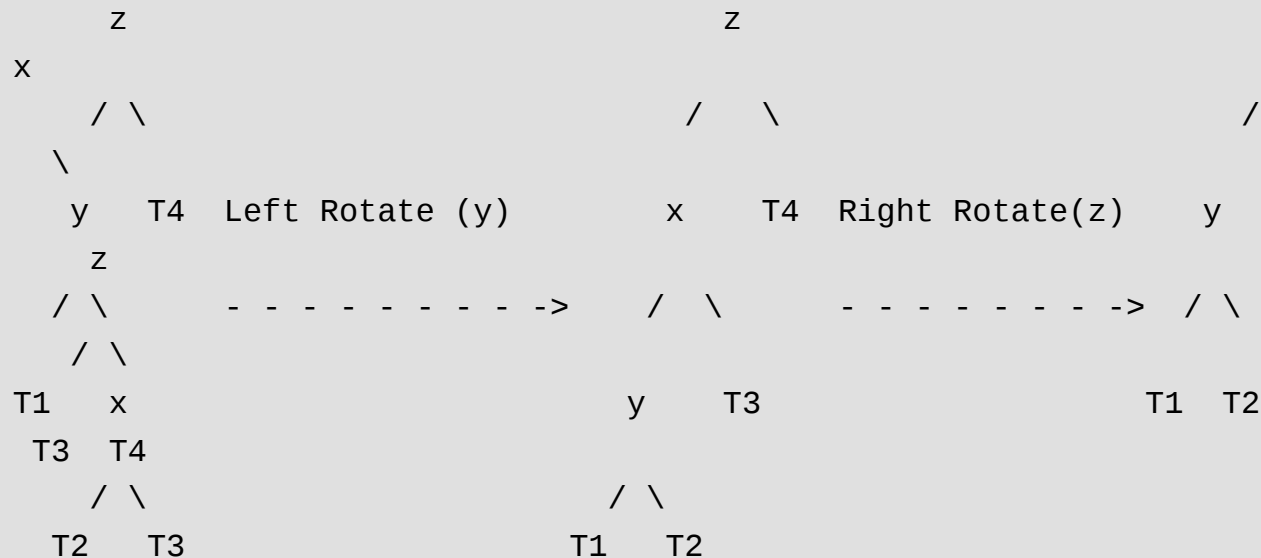
Popular Posts

- [Top 10 Algorithms and Data](#)

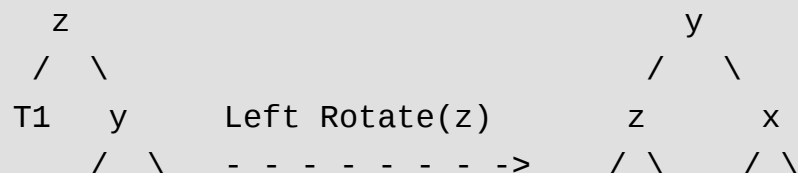
T1, T2, T3 and T4 are subtrees.



b) Left Right Case



c) Right Right Case



Structures for Competitive Programming

- Top 10 algorithms in Interview Questions
- How to begin with Competitive Programming?
- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Heavy Light Decomposition
- Sorted Linked List to Balanced BST
- Comb Sort
- Aho-Corasick Algorithm for Pattern Searching



```

      T2   x
       / \
      T3  T4

      T1  T2 T3  T4

```

d) Right Left Case

```

      z               z               x
     / \             / \             / \
    T1  y   Right Rotate (y)   T1  x   Left Rotate(z)   z
    x
   / \
  x   T4
 / \
T4
/ \
T2  T3

      T2  y               T1  T2  T3
       / \
      T3  T4

```

Unlike insertion, in deletion, after we perform a rotation at z, we may have to perform a rotation at ancestors of z. Thus, we must continue to trace the path until we reach the root.

C implementation

Following is the C implementation for AVL Tree Deletion.

The following C

implementation uses the recursive BST delete as basis. after deletion, we get pointers to all ancestors one by one in bottom up manner. So we don't need parent pointer to travel up. The recursive code itself travels up and visits all the ancestors of the deleted node.

1) Perform the normal BST deletion.

2) The current node must be one of the ancestors of the deleted node. Update the

Find out more

invitr

Subscribe and Never Miss an Article

Email Address

Subscribe

- [Common Interview Puzzles](#)
- [Interview Experiences](#)
- [Advanced Data Structures](#)
- [Dynamic Programming](#)
- [Greedy Algorithms](#)
- [Backtracking](#)
- [Pattern Searching](#)
- [Divide & Conquer](#)
- [Geometric Algorithms](#)
- [Searching](#)
- [Sorting](#)
- [Hashing](#)
- [Analysis of Algorithms](#)
- [Mathematical Algorithms](#)
- [Randomized Algorithms](#)

height of the current node.

3) Get the balance factor (left subtree height – right subtree height) of the current node.

4) If balance factor is greater than 1, then the current node is unbalanced and we are either in Left Left case or Left Right case. To check whether it is Left Left case or Left Right case, get the balance factor of left subtree. If balance factor of the left subtree is greater than or equal to 0, then it is Left Left case, else Left Right case.

5) If balance factor is less than -1, then the current node is unbalanced and we are either in Right Right case or Right Left case. To check whether it is Right Right case or Right Left case, get the balance factor of right subtree. If the balance factor of the right subtree is smaller than or equal to 0, then it is Right Right case, else Right Left case.

C

Java

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

// An AVL tree node
struct node
{
    int key;
    struct node *left;
    struct node *right;
    int height;
};
```

• Recursion

All Categories

Recent Comments

SurajThanks ashutosh :) For Boosting

How to find number of perfect squares between two given numbers · 1 minute ago

yogesh upadhyay This is same as...

Weighted Job Scheduling · 12 minutes ago

i guess:2 i connecting...

Detect Cycle in a Directed Graph · 40 minutes ago

Shambhavi Shinde The output for example one is -1, but should be...

Find k such that all elements in k'th row are 0 and k'th column are 1 in a boolean matrix · 58 minutes ago

any variable in c...

```

// A utility function to get maximum of two integers
int max(int a, int b);

// A utility function to get height of the tree
int height(struct node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

// A utility function to get maximum of two integers
int max(int a, int b)
{
    return (a > b)? a : b;
}

/* Helper function that allocates a new node with the given
key and
NULL left and right pointers. */
struct node* newNode(int key)
{
    struct node* node = (struct node*)
        malloc(sizeof(struct node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1; // new node is initially added at le
af
    return(node);
}

```

Write a program that produces different results in C and C++ · 1 hour ago

Ashish Jaiswal

C code:

#include<stdio.h>

#include<stdlib.h>...

Detect Cycle in a Directed Graph

· 2 hours ago

Tags

Adobe Advance Data
 Structures Advanced Data
 Structures **Amazon**
 array Backtracking Bharti SoftBank (HIKE)
 Bit Magic C++ CN c puzzle D-E-
 Shaw DBMS Divide and
 Conquer **Dynamic**
 Programming Flipkart
GATE GATE-CS-2012 GATE-CS-C-
 Language **GATE-CS-DS-&-**
Algo GATE-CS-Older GFacts
 Goldman Sachs Google **Graph**
 Greedy Algorithm Hashing
Interview
Experience Java MAQ
 Software
MathematicalAlgo

```

}

// A utility function to right rotate subtree rooted with y
// See the diagram given above.
struct node *rightRotate(struct node *y)
{
    struct node *x = y->left;
    struct node *T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;

    // Update heights
    y->height = max(height(y->left), height(y->right))+1;
    x->height = max(height(x->left), height(x->right))+1;

    // Return new root
    return x;
}

// A utility function to left rotate subtree rooted with x
// See the diagram given above.
struct node *leftRotate(struct node *x)
{
    struct node *y = x->right;
    struct node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;
}

```

Matrix
 Stanley
 Oracle
 puzzle
 Samsung
 stack

MICROSOTI
 Operating systems
 Pattern Searching
 Python
 SAP Labs
 Stack-Queue

Morgan
 Recursion
 SnapDeal

- [GeeksQuiz](#)
- [GeeksforGeeksIDE](#)
- [GeeksforGeeks Practice](#)
- [Data Structures](#)
- [Algorithms](#)
- [C Programming](#)
- [C++ Programming](#)
- [Java Programming](#)
- [Books](#)
- [Interview Experiences](#)
- [GATE CS](#)
- [GATE CS Forum](#)
- [Android App](#)

```

    // Update heights
    x->height = max(height(x->left), height(x->right))+1;
    y->height = max(height(y->left), height(y->right))+1;

    // Return new root
    return y;
}

// Get Balance factor of node N
int getBalance(struct node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

struct node* insert(struct node* node, int key)
{
    /* 1. Perform the normal BST rotation */
    if (node == NULL)
        return(newNode(key));

    if (key < node->key)
        node->left = insert(node->left, key);
    else
        node->right = insert(node->right, key);

    /* 2. Update height of this ancestor node */
    node->height = max(height(node->left), height(node->right)) + 1;
}

```



```
/* 3. Get the balance factor of this ancestor node to check whether
   this node became unbalanced */
int balance = getBalance(node);

// If this node becomes unbalanced, then there are 4 cases

// Left Left Case
if (balance > 1 && key < node->left->key)
    return rightRotate(node);

// Right Right Case
if (balance < -1 && key > node->right->key)
    return leftRotate(node);

// Left Right Case
if (balance > 1 && key > node->left->key)
{
    node->left = leftRotate(node->left);
    return rightRotate(node);
}

// Right Left Case
if (balance < -1 && key < node->right->key)
{
    node->right = rightRotate(node->right);
    return leftRotate(node);
}
```

```

    /* return the (unchanged) node pointer */
    return node;
}

/* Given a non-empty binary search tree, return the node with minimum
   key value found in that tree. Note that the entire tree
   does not
   need to be searched. */
struct node* minValueNode(struct node* node)
{
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current->left != NULL)
        current = current->left;

    return current;
}

struct node* deleteNode(struct node* root, int key)
{
    // STEP 1: PERFORM STANDARD BST DELETE

    if (root == NULL)
        return root;

    // If the key to be deleted is smaller than the root's
    key,
    // then it lies in left subtree
    if ( key < root->key )

```

```

        root->left = deleteNode(root->left, key);

    // If the key to be deleted is greater than the root's
    key,
    // then it lies in right subtree
    else if( key > root->key )
        root->right = deleteNode(root->right, key);

    // if key is same as root's key, then This is the node
    // to be deleted
    else
    {
        // node with only one child or no child
        if( (root->left == NULL) || (root->right == NULL) )
        {
            struct node *temp = root->left ? root->left : r
oot->right;

            // No child case
            if(temp == NULL)
            {
                temp = root;
                root = NULL;
            }
            else // One child case
                *root = *temp; // Copy the contents of the non
-empty child

            free(temp);
        }
        else

```

```

    {
        // node with two children: Get the inorder succ
        essor (smallest
            // in the right subtree)
            struct node* temp = minValueNode(root->right);

            // Copy the inorder successor's data to this no
            de
            root->key = temp->key;

            // Delete the inorder successor
            root->right = deleteNode(root->right, temp->key
        );
    }
}

// If the tree had only one node then return
if (root == NULL)
    return root;

// STEP 2: UPDATE HEIGHT OF THE CURRENT NODE
root->height = max(height(root->left), height(root->rig
ht)) + 1;

// STEP 3: GET THE BALANCE FACTOR OF THIS NODE (to chec
k whether
// this node became unbalanced)
int balance = getBalance(root);

// If this node becomes unbalanced, then there are 4 ca
ses

```

```

// Left Left Case
if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);

// Left Right Case
if (balance > 1 && getBalance(root->left) < 0)
{
    root->left = leftRotate(root->left);
    return rightRotate(root);
}

// Right Right Case
if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);

// Right Left Case
if (balance < -1 && getBalance(root->right) > 0)
{
    root->right = rightRotate(root->right);
    return leftRotate(root);
}

return root;
}

// A utility function to print preorder traversal of the tree.
// The function also prints height of every node
void preOrder(struct node *root)
{

```

```

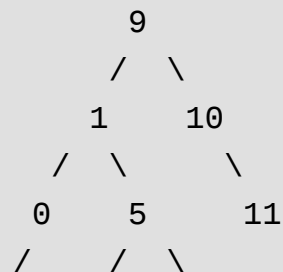
    if(root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

/* Driver program to test above function*/
int main()
{
    struct node *root = NULL;

    /* Constructing tree given in the above figure */
    root = insert(root, 9);
    root = insert(root, 5);
    root = insert(root, 10);
    root = insert(root, 0);
    root = insert(root, 6);
    root = insert(root, 11);
    root = insert(root, -1);
    root = insert(root, 1);
    root = insert(root, 2);

```

/* The constructed AVL Tree would be



```

        -1    2    6
    */

    printf("Pre order traversal of the constructed AVL tree
is \n");
    preOrder(root);

    root = deleteNode(root, 10);

    /* The AVL Tree after deletion of 10
        1
       / \
      0   9
     /   / \
    -1   5  11
       / \
      2   6
    */

    */

    printf("\nPre order traversal after deletion of 10 \n")
;
    preOrder(root);

    return 0;
}

```

Output:

```

Pre order traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11

```

```
Pre order traversal after deletion of 10
```

```
1 0 -1 9 5 2 6 11
```

Time Complexity: The rotation operations (left and right rotate) take constant time as only few pointers are being changed there. Updating the height and getting the

same as BST delete which is $O(h)$ where h is height of the tree. Since AVL tree is balanced, the height is $O(\log n)$. So time complexity of AVL delete is $O(\log n)$.

References:

[IITD Video Lecture on AVL Tree Insertion and Deletion](#)

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

Related Posts:

- Centroid Decomposition of Tree
- Gomory-Hu Tree | Set 1 (Introduction)
- kasai's Algorithm for Construction of LCP array from Suffix Array
- Overview of Data Structures | Set 3 (Graph, Trie, Segment Tree and Suffix Tree)
- Heavy Light Decomposition | Set 2 (Implementation)
- Heavy Light Decomposition | Set 1 (Introduction)
- Count Inversions of size three in a give array
- Count inversions in an array | Set 3 (Using BIT)

Previous post in category

Next post in category

(Login to Rate and Mark)

3.6

Average Difficulty : 3.6/5.0
Based on 3 vote(s)



Add to TODO List



Mark as DONE



30 people like this. [Sign Up](#) to see what your friends like.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

 Recommend 2  Share

Join the discussion...



pranav adarsh • 16 days ago

THIS IS ALSO WORKING !!!!

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

```
#include<malloc.h>
```

```
struct avl
```

```
{
```

```
int data;
```

```
struct avl*lf;
```

```
struct avl*rt;
```

```
int height;
```

```
};
```

```
int height(struct avl*n)
```

see more

^ | v • Reply • Share ›



Chaitanya • a month ago

If duplicate key comes, you are inserting at right side. But while checking for duplicate keys....

I think in insertion it should also check for duplicate keys...

if (balance < -1 && key >= node->right->key) //For right-right case
if (balance > 1 && key >= node->left->key) //For left-right case

Please comment if I am wrong...

^ | v • Reply • Share ›



TeamGfG Mod → Chaitanya • 25 days ago

According to the definition, duplicate keys are not present i then it must be handled explicitly.

^ | v • Reply • Share ›



nitin → TeamGfG • 20 days ago

While doing the deletion I am not seeing anywhere height I am not able to understand the reason b

^ | v • Reply • Share ›



sahil wadhwa • 8 months ago

Nice Tutorial :)



^ | v · Reply · Share ›



lucy · 10 months ago

how to handle if key is not present in tree

^ | v · Reply · Share ›



Klaus → lucy · 10 months ago

If you reach NULL in the above deletion procedure, that me

1 ^ | v · Reply · Share ›



AMIT JAMBOTKAR · a year ago

@GeeksforGeeks

```
if (key < node->key)
node->left = insert(node->left, key);
else
node->right = insert(node->right, key);
```

In the else part if keys are duplicated it will keep on visiting insert ti
Correct me if I am wrong .If you find it's wrong please update code

^ | v · Reply · Share ›



Nick · a year ago

I think the avl tree after the deletion of 10 is wrong. Accordint to lef
you can contract me for the right avl tree.

^ | v · Reply · Share ›



helper · a year ago



helpful · 2 years ago

now i got the importance of that one line of java, which creates a l

^ | v · Reply · Share ›



Hero · 2 years ago

I think we can improve the code more concisely by combining the balancing node part. Like this replace the balancing node code wil

```
Node *balanceNode(Node *n)
```

```
{
```

```
if (!n) return n;
```

```
updateHeight(n);
```

```
int balance = getBalance(n);
```

```
if (balance > 1 && getBalance(n->left) >= 0)
```

```
return rightRotate(n);
```

```
if (balance < -1 && getBalance(n->right) <= 0)
```

```
return leftRotate(n);
```

[see more](#)

3 ^ | v · Reply · Share ›



Harshil Sukhadia → Hero · 2 years ago



a little improvement can be done by including the code of f
node in the routine (and node *n should be inserted or dele
done both in insertion and deletion

^ | v • Reply • Share ›



prashant jha • 2 years ago

```
#include<iostream>
using namespace std;
struct tnode
{
tnode* lchild;
int data;
tnode* rchild;
tnode(int d)
{
lchild=NULL;
data=d;
rchild=NULL;
}
};
int get_height(tnode* root)
{
if(!root)
return 0;
```

[see more](#)

^ | v • Reply • Share ›



prashant jha · 2 years ago

here during unwinding phase of recursion is balance factor of a node has to perform double rotation else single right rotate but in insertion must be either -1 or 1 but here it is not necessary it may be -1 or 0 .. rotation

^ | v · Reply · Share ›



ArafatX · 2 years ago

If anyone here wants to have the complete source code with this function twitter.com/arafatx and pm me there.

[img] <http://codegix.com/imagex/avll...> [/img]

Credit to geeksforgeeks to with the previous explanation in order to

^ | v · Reply · Share ›



mallard · 2 years ago

can't we check the check left->left->right etc case of deleted node

^ | v · Reply · Share ›



ArafatX · 2 years ago

I believe this code has error but I'm not sure what (trying to figure out) sequence 10, 20, 30, 40, 50, 60, 70, 80, 90, 99. And we delete 40
30, 20, 10, 80, 60, 50, 70, 90, 99. But the result from your code:

Edit: It's ok. Now I understand. There are 2 ways of deleting the AVL

^ | v · Reply · Share ›



Guest → ArafatX • 2 years ago

Pardon me, now I understand. There are 2 ways of deletin

^ | v • Reply • Share ›



Dimitris S. • 2 years ago

Hey, one small question:

What do the different height values stand for?

- Does height == 1 mean "the node is balanced" e.g. a node with 1 child and 1 child themselves.
- What height would a node that is left-balanced have, for example
- In relation to the upper bulletpoint, what value of height would a node have if it is right-balanced?

^ | v • Reply • Share ›



Castle Age → Dimitris S. • 2 years ago

Height is the height of the current subtree. Leave nodes alone if the difference between the height of the left subtree and the right subtree is -1 and 1.

You need to reread every thing about trees and their definitions.

^ | v • Reply • Share ›



Jaini • 3 years ago

No doubt, this tutorial is awesome but to make it more self explanatory, could you add more subtrees in diagrams?

^ | v • Reply • Share ›



abhishek08aug · 3 years ago

Intelligent :D

^ | v · Reply · Share ›



Pranshu Tomar · 3 years ago

not bad...

^ | v · Reply · Share ›



Doubt · 3 years ago

When you delete the node , don't you need to reset it's parent's pc

```
/* Paste your code here (You may delete these lines if no
```

^ | v · Reply · Share ›



GeeksforGeeks → Doubt · 3 years ago

This is handled in as we have recursive code and returned a closer look at following lines

```
// If the key to be deleted is smaller than the  
// then it lies in left subtree
```

```
if ( key < root->key )  
    root->left = deleteNode(root->left, key);
```

```
// If the key to be deleted is greater than the  
// then it lies in right subtree
```

```
else if ( key > root->key )
```

```
else if( key > root->key )
    root->right = deleteNode(root->right, key);
```

^ | v • Reply • Share ›



Julian • 3 years ago

Your deletion code seems to only check for imbalance on the root. However, wikipedia says you need to do it for all ancestors of the node. Is this? because I'm not seeing it.

3 ^ | v • Reply • Share ›



Castle Age → Julian • 2 years ago

The power of recursion. For every recursive call, it checks if the node is unbalanced and then moves up.

1 ^ | v • Reply • Share ›



mallard → Julian • 2 years ago

we are backtracing when we have deleted the node, hence we do appropriate

1 ^ | v • Reply • Share ›



BlackMath • 4 years ago

Something wrong with the code.

At the line while deleting the node, in the case of two children :

```
// node with two children: Get the inorder successor (smallest
// in the right subtree)
```

```
struct node* temp = minValueNode(root->right);
```

```
struct Node { int val; struct Node *left, *right; }
```

What if the right subtree is null ?

when the right subtree of a node is null and we want to find the su
parent which is a left child of its parent.

Code for minValueNode is not handling this case and if we call mi
will throw null pointer exception :

```
/* loop down to find the leftmost leaf */  
while (current->left != NULL)  
current = current->left;
```

Please correct me is i am wrong.

[see more](#)

^ | v • Reply • Share ›



TulsiRam → BlackMath • a year ago

This will never be the case. . Because you will enter in this
children

^ | v • Reply • Share ›



GeeksforGeeks → BlackMath • 4 years ago

@BlackMath:

We travel up to find the inorder successor only when right
we come to minValueNode() function only when both left a
an inorder successor which is also an ancestor.

Following link can also be useful:

<http://www.geeksforgeeks.org/archives/9999/comment-pa>

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

Privacy



@geeksforgeeks, Some rights reserved

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)