mycodeschool / BSTDeletion_CPP.cpp

Last active 2 days ago



BSTDeletion_CPP.cpp

```
/* Deleting a node from Binary search tree */
#include<iostream>
using namespace std;
struct Node {
        int data;
        struct Node *left;
        struct Node *right;
};
//Function to find minimum in a tree.
Node* FindMin(Node* root)
{
        while(root->left != NULL) root = root->left;
        return root;
}

// Function to search a delete a value from tree.
struct Node* Delete(struct Node *root, int data) {
        if(root == NULL) return root;
        else if(data < root->data) root->left = Delete(root->left,data);
        else if (data > root->data) root->right = Delete(root->right,data);
        // Wohoo... I found you, Get ready to be deleted
        else {
                // Case 1:  No child
                if(root->left == NULL && root->right == NULL) {
                        delete root;
                        root = NULL;
                }
                //Case 2: One child
                else if(root->left == NULL) {
                        struct Node *temp = root;
                        root = root->right;
                        delete temp;
                }
                else if(root->right == NULL) {
                        struct Node *temp = root;
                        root = root->left;
                        delete temp;
                }
                // case 3: 2 children
                else {
                        struct Node *temp = FindMin(root->right);
                        root->data = temp->data;
                        root->right = Delete(root->right,temp->data);
                }
        }
        return root;
}

//Function to visit nodes in Inorder
void Inorder(Node *root) {
        if(root == NULL) return;

        Inorder(root->left);       //Visit left subtree
        printf("%d ",root->data);  //Print data
        Inorder(root->right);      // Visit right subtree
```

```
}

// Function to Insert Node in a Binary Search Tree
Node* Insert(Node *root,char data) {
        if(root == NULL) {
                root = new Node();
                root->data = data;
                root->left = root->right = NULL;
        }
        else if(data <= root->data)
                root->left = Insert(root->left,data);
        else
                root->right = Insert(root->right,data);
        return root;
}

int main() {
        /*Code To Test the logic
          Creating an example tree
                             5
                            / \
                           3   10
                          / \   \
                         1   4   11
    */
        Node* root = NULL;
        root = Insert(root,5); root = Insert(root,10);
        root = Insert(root,3); root = Insert(root,4);
        root = Insert(root,1); root = Insert(root,11);

        // Deleting node with value 5, change this value to test other cases
        root = Delete(root,5);

        //Print Nodes in Inorder
        cout<<"Inorder: ";
        Inorder(root);
        cout<<"\n";
}
```

**sidhjhawar** commented on May 2, 2014

Hi, How are you making sure that once the node to be is deleted is deleted and the link to its parent is still maintained now between the child of the node deleted and its parent ?
More specifically,
else if(root->left == NULL) {
struct Node *temp = root;
root = root->right;
delete temp;
}

After deleting the temp, what about the link between temp's parent and current root ? I am little confused here.
Please explain if I am wrong.
Thanks,
Sidharth

**Merciaugust29** commented on Aug 15, 2014

I have the same issue. I thought that deleting a node with one child, we are supposed to find the parent, and link the parent to the child of the node to be deleted. You are just making the node to be deleted point to his child, and delete the node. Does that automatically make the parent of the node to be deleted point to the child of the node to be deleted? If so how does that work? Can you please clarify?

**hhuynh** commented on Oct 5, 2014

to **@sidhjhawar** and **@Merciaugust29**,

That's why the Delete function has a return statement. It's returning the same node that's being deleted, which is most of the time is NULL.

For example, let say we're deleting node 11. A recursive function ended up in node 10, line 20. The root in this context is the node 10 itself.

line 20: root(node10)->right = Delete(root->right(node11), 11)

Here, the Delete function is called again and the root in this context become node 11. Since node 11 has no child, the logic ends up on line 24. The root is deleted and set to NULL.

This NULL value is then returned to the earlier call we had above:

back to line 20: root(node10)->right = NULL;

So effectively, from node 10 perspective, node 11 is truly gone.

You could argue with the same logic for any other cases.

**NLababidi** commented on Dec 14, 2014

Very Clean and helpful code. Thank You,

**samarpanda** commented on May 2, 2015

May be a small thing. While running the above program i get this error

```
root@runnable:~# g++ /root/main.cpp
/root/main.cpp: In function 'void Inorder(Node*)':
/root/main.cpp:44:26: error: 'printf' was not declared in this scope
```

I think we need to include `#include<cstdio>` . It compiles without any error after adding the package.

```
root@runnable:~# g++ /root/main.cpp
root@runnable:~# /root/a.out
Inorder: 1 3 4 10 11
```

You can checkout the code in runnable link

**jigarshah2811** commented on Jul 28, 2016

Great explanation and elegant code. Thanks a lot.

**sakib1061** commented on Oct 21, 2016

What need of temp?
Isn't it enough
root=root->left
for the case 2

**dhinapak** commented on Oct 28, 2016

HI while i m running the above i m getting error as it is in below link [http://stackoverflow.com/questions/29491024/crash-this-may-be-due-to-a-corruption-of-the-heap](click here)

how to resolve plz help me out

thanks in advance
dina

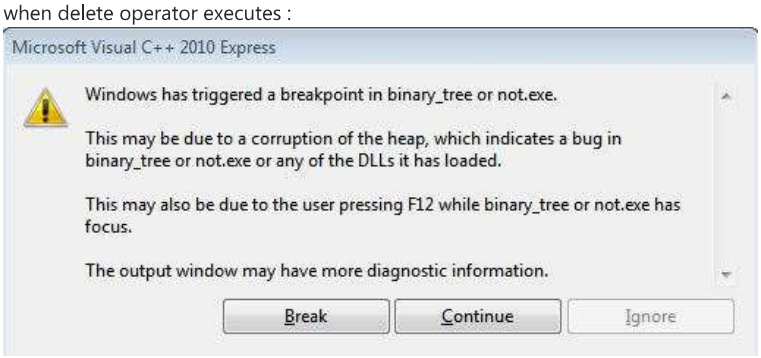
**dhinapak** commented on Oct 28, 2016

``

**dhinapak** commented on Oct 28, 2016

Hi i m getting below error while running above code

when delete operator executes :




**afnancute** commented on Jun 18, 2017

Thnaks

**VaibhavS22** commented on Aug 5, 2017

shouldn't the second case be like this :
else if(root->right==NULL){
Node* temp=root->left;
delete root;
root=temp;
}

**meet2mky** commented on Mar 6, 2018

For everyone who are not getting how the link between the parent of deleted node and and child of deleted node is maintainted. You can easily get it just by looking where the return value of each recursion is going.

**doveral** commented on Mar 28, 2018

this code is quite very clean to understand,thank you very much ,you did a great job,atleast better than what my lecturer does

**rak96** commented on Apr 30, 2018

i have a doubt.. how the ctrl trasfers to else if to else

**vageeesh** commented on Jul 22, 2018

good explanation.

**Pasha54** commented on Sep 29, 2018

In Insert function, data has input as char data type, why?



**sheikhazad** commented on Dec 20, 2018 • edited

> Hi, How are you making sure that once the node to be is deleted is deleted and the link to its parent is still maintained now between the child of the node deleted and its parent ?
> More specifically,
> else if(root->left == NULL) {
> struct Node *temp = root;
> root = root->right;
> delete temp;
> }
>
> After deleting the temp, what about the link between temp's parent and current root ? I am little confused here.
> Please explain if I am wrong.
> Thanks,
> Sidharth

Let me copy paste the funtion with comments so that u may understand:

```
Node* deleteNode(Node* temp, int data)
{
(temp)? std::cout<<"\nIn deleteNode temp->data = " <data : std::cout<<"\nIn deleteNode temp = NULL" ;
if(temp == NULL)
{
std::cout<<"It's NULL";
return temp;
}
else if( data < temp->data)
{
std::cout<<"\n Data to be delted is in left of : " << temp->data << ", so sending next left node ";
temp->left = deleteNode(temp->left, data); //Assign to temp->left because u are sure now u r not deleting current node but in left of current.
temp->left will be parent of what deleteNode() returns when stack unwinds
(temp->left) ? std::cout<<"\nReturned temp->left Node: " <left->data : std::cout<<"\nReturned temp->left = NULL";
}
else if(data > temp->data)
{ std::cout<<"\n Data to be delted is in right of : " << temp->data << ", so sending next right node ";
temp->right = deleteNode(temp->right, data); //Assign to temp->right because u are sure now u r not deleting current node but in right of
current. temp->right will be parent of what deleteNode() returns when stack unwinds
(temp->right) ? std::cout<<"\nReturned temp->right Node: " <right->data : std::cout<<"\nReturned temp->right = NULL";
}
else //found data to be deleted
{
Node* current = temp; //for clarity
if(current->left == NULL && current->right == NULL) // If no children
{
delete current;
temp = NULL; // return NULL to be linked to previous parent's left or right in previous "else if" when stack unwinds
std::cout<<"\n No children";
}
//Node to be deleted has left or right or both childred
else if(current->left == NULL)
{
temp = current->right;
delete current; // return current->right to be linked to previous parent's left or right in previous "else if" when stack unwinds, U dont need to
track parent in recursion as parent will be there when current recursion unwinds to previous
std::cout<<"\n Has right children";
}
else if(current->right == NULL)
{
temp = current->left;
delete current; // return current->left to be linked to previous parent's left or right in previous "else if" when stack unwinds, U dont need to track
parent in recursion as parent will be there when current recursion unwinds to previous
std::cout<<"\n Has left children";
}
else//if left and right children
{
std::cout<<"\n Has left and right children";
Node* minNodeInRight = findMinNode(current->right); //Find min node in right of current node, and that min node will be deleted in future.
std::cout<<"\n Min Node found to be deleted is " << minNodeInRight->data;
current->data = minNodeInRight->data; //Copy min node data here and dont delete current but delete the found node in right of current


                //current->right = deleteNode(minNodeInRight->right, minNodeInRight->data); // U cant pass minNodeInRight->right as
  argument becoz its much below current node, if u send it link with parent will be broken,
                //U need to traverse from Current node until u find  minNodeInRight->data
                current->right = deleteNode(current->right, minNodeInRight->data);      //Assign to current->right because u are
  sure now u r not deleting current node but in right of current.

        }

  }
  (temp) ? std::cout<<"\n END OF DELTE, returning Node: " <<temp->data : std::cout<<"\nReturning temp = NULL";
  return temp;
```

```
}
```

**hardik4uonly** commented on Sep 17, 2019

> What need of temp?
> Isn't it enough
> root=root->left
> for the case 2

for making memory free . because that data is not useful for us now. but it is stored in Heap section.

**himu59** commented on Oct 31, 2019 • edited

why we assign delete function to root in line 87.
it is necessary?

**paras062** commented on Jan 28

Looks like it fail if you try to delete 12 from the BST, as 12 is equal to root.data, and we don't have condition to handle when data = root.data. Please have a look.
We have condition to check if data > root.data or data < root.data, what missing is data = root.data

I tried this on my end, below is the python version of doing the same.

## CASE 1: Delete root of the binary tree

```
elif (data == root.data):
    temp = root
    newRoot = root.left
    root = root.left


    # Move current root to the end of right most node of the left subtree
    while(root.right != None):
        root = root.right


    if root.right == None:
        root.right = temp.right


    del temp, root
    return newRoot
```

**paras062** commented on Jan 28

There is another issue with the code, in your example you've been deleting elements from the right sub tree, if you try to delete from left subtree, your FindMin function will fail.

```
def minValue(node, data):
current = node
if (data < node.data):
# loop down to find the right most leaf
while(current.right is not None):
current = current.right
elif (data > node.data):
# loop down to find the left most leaf
while(current.left is not None):
current = current.left


  return current.data
```

This is what I did to fix it.

**pnguptacs7** commented 22 days ago

Hi paras062,
we don't need to make that change.The code is similar to search element in a binary tree ie element can be in a left sub tree or right sub tree,all FindMin function does is to find min element in a right sub tree.I executed this code and it works fine.