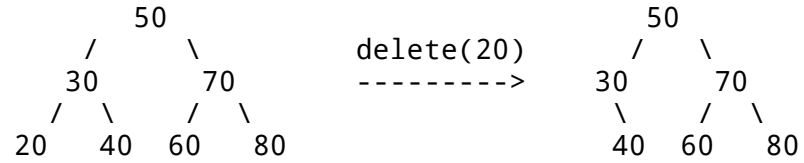
Binary Search Tree | Set 2 (Delete)

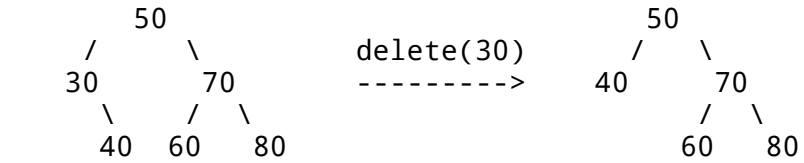
January 30, 2014

We have discussed BST search and insert operations. In this post, delete operation is discussed. When we delete a node, there possibilities arise.

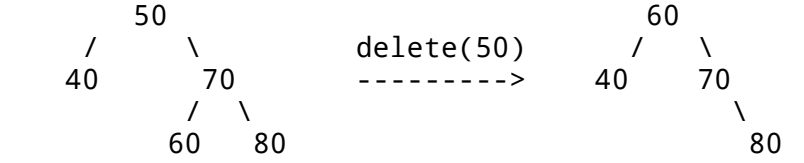
1) ***Node to be deleted is leaf:*** Simply remove from the tree.



2) ***Node to be deleted has only one child:*** Copy the child to the node and delete the child



3) ***Node to be deleted has two children:*** Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.



The important thing to note is, inorder successor is needed only when right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in right child of the node.

```
// C program to demonstrate delete operation in binary search tree
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp =  (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else
        node->right = insert(node->right, key);

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

/* Given a non-empty binary search tree, return the node with minimu
   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;
}

/* Given a binary search tree and a key, this function deletes the k
   and returns the new root */
struct node* deleteNode(struct node* root, int key)
{
    // base case
    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)
        {
            struct node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct node *temp = root->left;
            free(root);
            return temp;
        }

        // node with two children: Get the inorder successor (smalle
        // in the right subtree)
        struct node* temp = minValueNode(root->right);

        // Copy the inorder successor's content to this node
        root->key = temp->key;

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

```
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
       20   40  60   80 */
    struct node *root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    printf("Inorder traversal of the given tree \n");
    inorder(root);

    printf("\nDelete 20\n");
    root = deleteNode(root, 20);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    printf("\nDelete 30\n");
    root = deleteNode(root, 30);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    printf("\nDelete 50\n");
    root = deleteNode(root, 50);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    return 0;
}
```

Output:

```
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80
```

Time Complexity: The worst case time complexity of delete operation is O(h) where h is height of Binary Search Tree. In worst case, we may have to travel from root to the deepest leaf node. The height of a skewed tree may become n and the time complexity of delete operation may become O(n)

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

Related Questions:

- Convert left-right representation of a bianry tree to down-right
- Print level order traversal line by line
- C Program for Red Black Tree Insertion
- Can we reverse a linked list in less than O(n)?
- A Programmer's approach of looking at Array vs. Linked List
- Threaded Binary Tree
- Priority Queue | Set 1 (Introduction)
- Stack | Set 4 (Evaluation of Postfix Expression)

1) Smallest of bigger elements than deleted node i.e. left most node of right sub-tree
OR
2) Biggest of smaller elements than deleted node i.e. rightmost node of left sub-tree.

Explanation should mention both for clear understanding.

4 ▲ | ▼ • Reply • Share ›

shashank • 6 months ago

struct node *temp = root->right;
free(root);
return root->right;

shouldnt you be returning root here ?
also if you have freed root's space , how does root->right not cause a core?

2 ▲ | ▼ • Reply • Share ›

Shankar Narayanan ➦ shashank • 6 months ago

It should be return temp. That is the purpose of creating a temp variable there. I think the author missed it.

1 ▲ | ▼ • Reply • Share ›

GeeksforGeeks Mod ➦ Shankar Narayanan • 5 months ago

Shashank & Shankar Narayanan,

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

1 ▲ | ▼ • Reply • Share ›

rahul • a month ago

update the tree after deleting 30 in the diagram

1 ▲ | ▼ • Reply • Share ›

GeeksforGeeks Mod ➦ rahul • 16 days ago

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

▲ | ▼ • Reply • Share ›

Akheel • 10 days ago

In this case if we delete nodes 50, then 70. the node 80 is also lost alongwith the 70

There should be a condition for node->right==NULL with the node->left==NULL, because otherwise it simply deletes the node and loses the right subtree of the tree

▲ | ▼ • Reply • Share ›

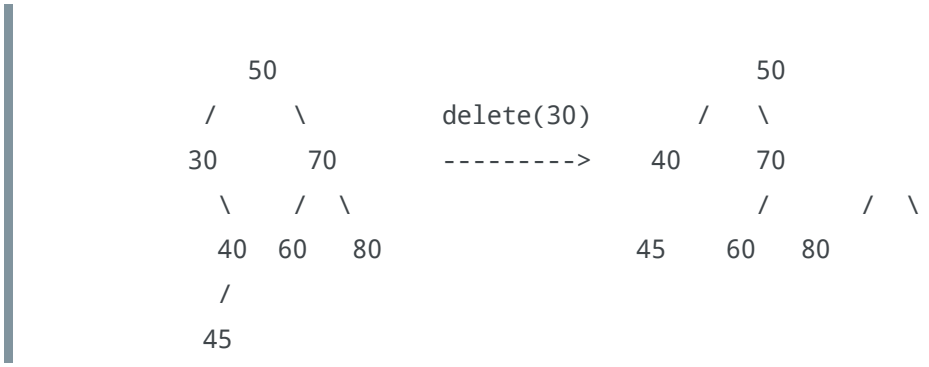
Phanijella • 16 days ago

how to delete 30 if 40 has a right child 45 ?

• Reply • Share ›

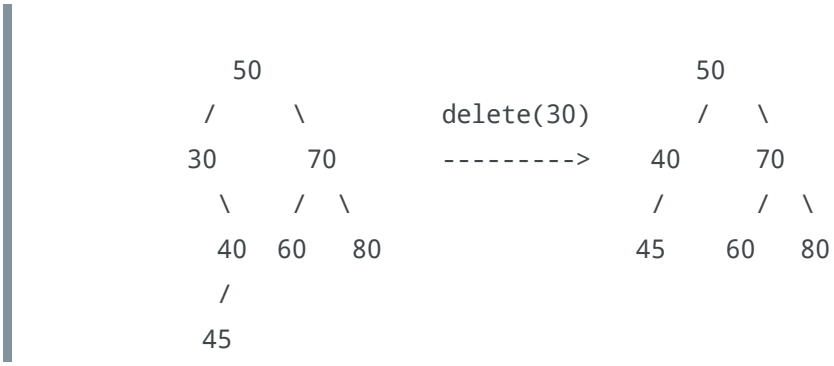
Kartik ➚ Phanijella • 16 days ago

45 would go move with 40



• Reply • Share ›

Kartik ➚ Kartik • 16 days ago



• Reply • Share ›

mb1994 • 3 months ago

in the case where the left child is NULL, you have simply replaced the node with the right child. Cant' you replace the node's value with the LEAST of right subtree as well?
that's what you've done in the last case, as I see.

• Reply • Share ›

bhupendra • 3 months ago

please give the link to the next post in the topic below every post

• Reply • Share ›

Subscribe Add Disqus to your site

- **Categories**
  - Articles (77)
    - Algorithms (8)
    - C (9)
    - C++ (15)
    - Data Structures (24)
    - DBMS (1)
    - Java (2)
    - Operating Systems (1)
    - Searching and Sorting (10)
  - Programs (35)
  - Quizzes (1,410)
    - Aptitude (1)
    - Computer Science Quizzes (1,409)
      - Algorithms (146)

- C (205)
- C++ (127)
- Data Structures (131)
- DBMS (2)
- GATE (719)
- Java (51)
- Operating Systems (28)

-

Recent Discussions

- coder

  Code is not correct...!!! Check for input :...

  Convert left-right representation of a bianry tree to down-right · 10 minutes ago

- Sai Pavan Pothuri

As we have not defined the copy constructor....

Copy Constructor in C++ · 10 hours ago

- krishna

  with out sorting,and finding rotations...

  Lexicographically minimum string rotation · 1 day ago

- Kartik

  Please take a closer look, the given string is...

  Lexicographically minimum string rotation · 1 day ago

- hh

  Order of array wiil be disturbed if we sort the...

  Lexicographically minimum string rotation · 1 day ago

- hh

  Order of array will be disturbed if we sort the...

  Lexicographically minimum string rotation · 1 day ago