Deletion in a Binary Search Tree

We have discussed the Search and Insert operations in BST in the previous post. In this post, delete operation is discussed.

That is, given a Binary Search Tree and a node to be deleted. The task is to search that node in the given BST and delete it from the BST if it is present.

When we delete a node, three cases may arise:

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

```
              50                            50
           /     \         delete(20)      /   \
          30      70       --------->    30     70
         /  \    /  \                     \    /  \
       20   40  60   80                   40  60   80
```
2.

3. ***Node to be deleted has only one child:*** Copy the child to the node and delete the child.

```
              50                            50
           /     \         delete(30)      /   \
          30      70       --------->    40     70
            \    /  \                          /  \
            40  60   80                       60   80
```
4.
5.

6. ***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.

```
              50                            60
           /     \         delete(50)      /   \
          40      70       --------->    40    70
                 /  \                            \
                60   80                           80
```
7.
8.

Note: *The inorder successor can be obtained by finding the minimum value in the right child of the node.*

Below is the implementation of the above three cases:
C++Java

```
// C++ program to demonstrate delete
// operation in binary search tree
#include<bits/stdc++.h>
using namespace std;

// BST Node
struct node
{
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    node *temp = new 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);
        cout<<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
```

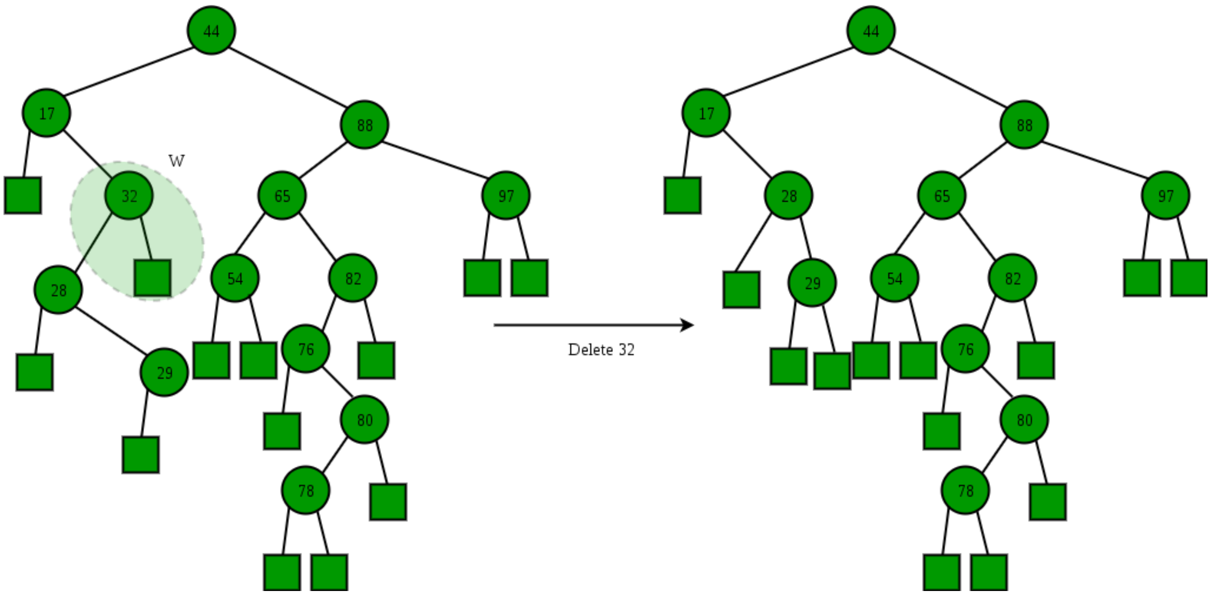
```c
    if (node == NULL)
        return newNode(key);

    // 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;
}

// Function to find the minimum
// valued node in a BST
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;
}

// Function to delete a given node from the BST
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
(smallest
        // 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
```

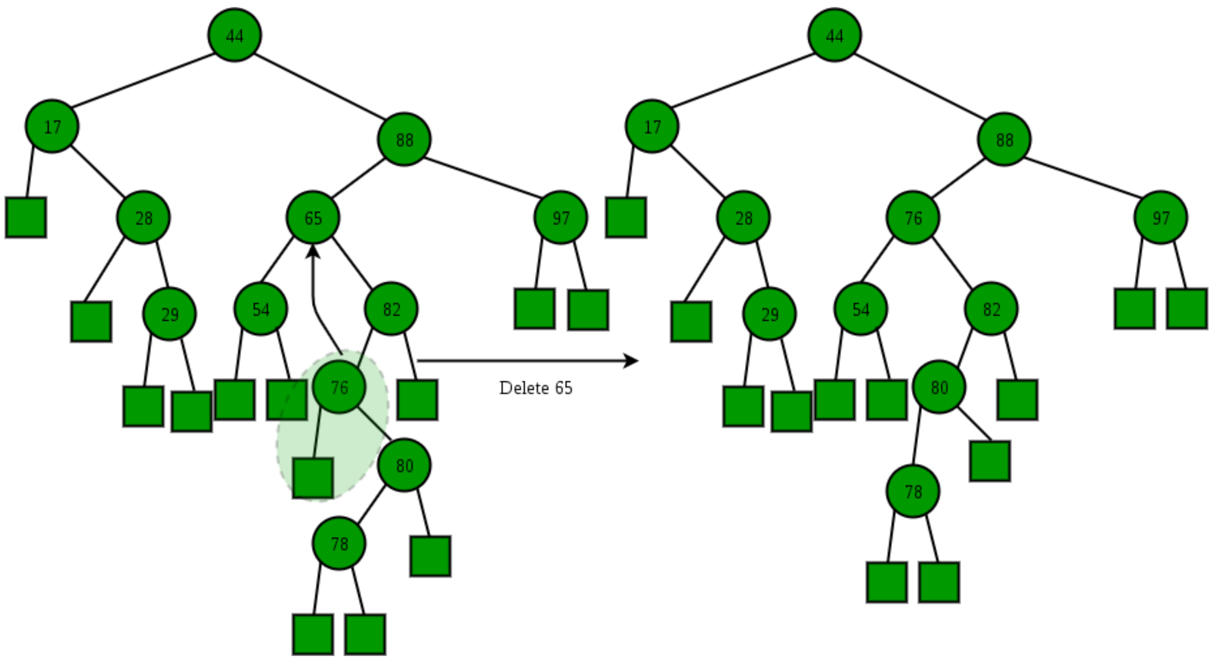
```
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         /  \    / \
       20   40  60   80 */
    struct node *root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);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

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

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

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

    cout<<"\nDelete 50\n";
    root = deleteNode(root, 50);
    cout<<"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
```

Illustration:




44
17
88
28
65
97
29
54
82
76
80
78
Delete 65
44
17
88
28
76
97
29
54
82
80
78