ASSIGNMENT X:
Binary Search Tree Implementation
U19CS012 [D-12]

Implement the following operations in context to Binary Search Tree:

1) Creation & Insertion of Element in Binary Search Tree
2) Deletion of Element in Binary Search Tree
3) Updating of Element in Binary Search Tree
4.) Calculate the Height of Binary Search Tree

Code:

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

// General Node Structure of BST
struct node
{
    int data;
    struct node *right_child;
    struct node *left_child;
};

//Helper Functions

//Function to create a BST node
struct node *new_node(int x);

// Insertion of Node in BST
struct node *insert(struct node *root, int x);

// Minimum value in a node [LeftMost Leaf Node]
struct node *min_node(struct node *root);

// Delete a node in BST
struct node *delete (struct node *root, int x);

// Search in BST
int search(struct node *root, int x);

// Updation Of Element in BST
struct node *update(struct node *root);
```

```c
// Inorder Traversal Of BST [Will Always be Sorted]
void inorder(struct node *root);

// Height of BST
int height_of_bst(struct node *root);

#define max(a, b) (a > b) ? a : b;

struct node *root;

int main()
{
    int choice;
    printf("\nBINARY SEARCH TREE\n");

    printf(" 1 -> Create a Binary Search Tree\n");
    printf(" 2 -> Display the INORDER Traversal of BST\n");
    printf(" 3 -> Insert an Element in BST\n");
    printf(" 4 -> Delete an Element in BST\n");
    printf(" 5 -> Update an Element in BST\n");
    printf(" 6 -> Calculate the Height of BST\n");
    printf(" 7 -> Exit\n");
    int x, hgt;
    while (1)
    {
        printf("Enter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter Node Value : ");
            scanf("%d", &x);
            root = new_node(x);
            break;
        case 2:
            printf("INORDER TRAVERSAL = [");
            inorder(root);
            printf("]\n");
            break;
        case 3:
            printf("Enter Node Value : ");
            scanf("%d", &x);
            root = insert(root, x);
            break;
        case 4:
            printf("Enter Node Value to be Deleted : ");
            scanf("%d", &x);
            if (search(root, x))
```

```
            {
                root = delete (root, x);
            }
            else
            {
                printf("Given Element Doesn't Exist in BST!\n");
            }
            break;
        case 5:
            root = update(root);
            break;
        case 6:
            hgt = height_of_bst(root);
            printf("Height of BST : %d\n", hgt);
            break;
        case 7:
            exit(0);
            break;
        default:
            printf("Enter a Valid Choice!");
            break;
        }
    }

    return 0;
}

//Function to create a BST node
struct node *new_node(int x)
{
    struct node *p;
    p = malloc(sizeof(struct node));

    p->data = x;
    p->left_child = NULL;
    p->right_child = NULL;

    return p;
}

// Insertion of Node in BST
struct node *insert(struct node *root, int x)
{

    // 1. If the tree is empty, return a new, single node
    if (root == NULL)
        return (new_node(x));
    // 2. Otherwise, recur down the tree
    else
    {
```

```
        // x is greater. Should be inserted to right
        if (x > root->data)
            root->right_child = insert(root->right_child, x);
        // x is smaller should be inserted to left
        else
            root->left_child = insert(root->left_child, x);
    }

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

// Minimum value in a node [LeftMost Leaf Node]
struct node *min_node(struct node *root)
{
    // Empty BST
    if (root == NULL)
        return NULL;

    // Node with Minimum Value will have No left Child
    else if (root->left_child != NULL)
        // left most element will be minimum
        return min_node(root->left_child);

    return root;
}

// function to delete a node
struct node *delete (struct node *root, int x)
{
    // If No Tree is there, Deletion Not Possible
    if (root == NULL)
        return NULL;

    //searching for the item to be deleted
    if (x > root->data)
        root->right_child = delete (root->right_child, x);
    else if (x < root->data)
        root->left_child = delete (root->left_child, x);
    else
    {
        //CASE #1 : No Children
        if (root->left_child == NULL && root->right_child == NULL)
        {
            free(root);
            return NULL;
        }

        //CASE #2 : One Child
        else if (root->left_child == NULL || root->right_child == NULL)
```

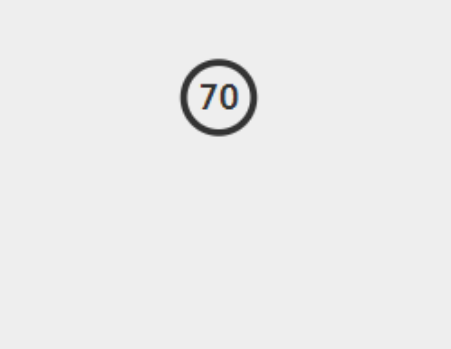
```
        {
            struct node *temp;

            if (root->left_child == NULL)
                temp = root->right_child;
            else
                temp = root->left_child;

            free(root);
            return temp;
        }

        //CASE #1 : Two Children
        else
        {
            struct node *temp = min_node(root->right_child);
            root->data = temp->data;
            root->right_child = delete (root->right_child, temp->data);
        }
    }

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

// Search in BST
int search(struct node *root, int x)
{
    // 1. Base case == empty tree
    // in that case, the target is not found so return false
    // Not Found Case
    if (root == NULL)
        return 0;

    //if root->data is x then the element is found
    if (root->data == x)
        return 1;
    // x is greater, so we will search the right subtree
    else if (x > root->data)
        return search(root->right_child, x);
    //x is smaller than the data, so we will search the left subtree
    else if (x < root->data)
        return search(root->left_child, x);
}

// Updation Of Element in BST
struct node *update(struct node *root)
{
    int old;
    printf("Enter the Element to be Updated in BST : ");
```

```
    scanf("%d", &old);
    if (search(root, old))
    {
        // Delete the Old Element
        struct node *t = delete (root, old);
        int new;
        printf("Enter the Updated Element to be Replaced : ");
        scanf("%d", &new);
        struct node *s = insert(root, new);
    }
    else
    {
        printf("Given Element Doesn't Exist in BST!\n");
    }

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

// Inorder Traversal Of BST [Will Always be Sorted]
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left_child);
        printf(" %d, ", root->data);
        inorder(root->right_child);
    }
}

// Height of BST
int height_of_bst(struct node *root)
{
    // Empty BST
    if (root == NULL)
        return 0;
    else
    {
        // Max Height of left Subtree
        int left_max = height_of_bst(root->left_child);
        // Max Height of Right Subtree
        int right_max = height_of_bst(root->right_child);
        // Take Maximum of Left & Right Sub-Tree and Add Root
        int mxn = max(left_max, right_max);
        return (mxn + 1);
    }
}
```

Test Cases:

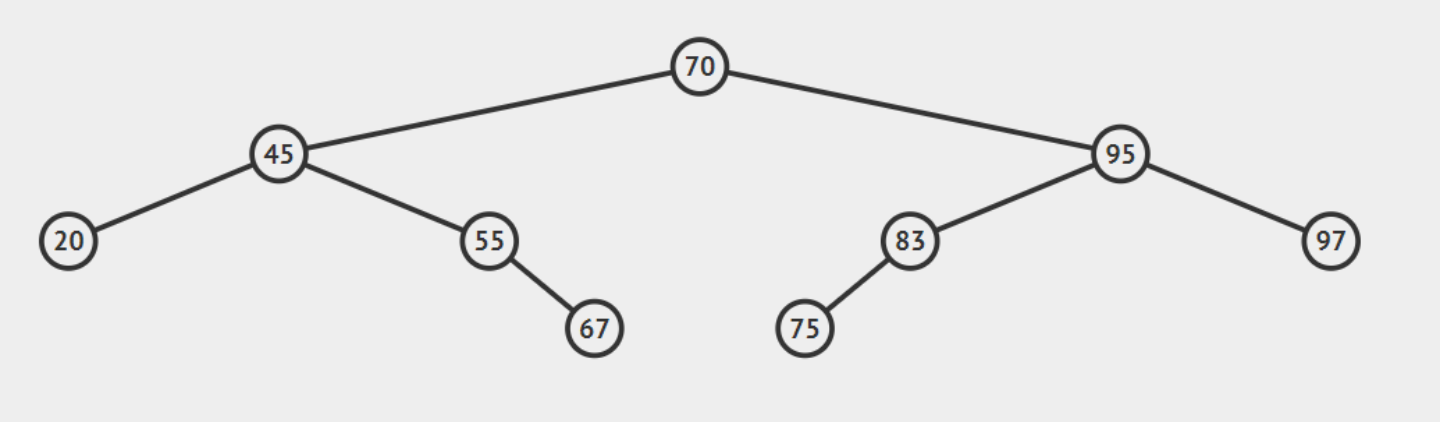
A.) Creation of Binary Search Tree

```
BINARY SEARCH TREE
 1 -> Create a Binary Search Tree
 2 -> Display the INORDER Traversal of BST
 3 -> Insert an Element in BST
 4 -> Delete an Element in BST
 5 -> Update an Element in BST
 6 -> Calculate the Height of BST
 7 -> Exit
Enter your choice : 1
Enter Node Value : 70
Enter your choice : 2
INORDER TRAVERSAL = [ 70, ]
```



B.) Insertion of Element in Binary Search Tree

```
BINARY SEARCH TREE
 1 -> Create a Binary Search Tree
 2 -> Display the INORDER Traversal of BST
 3 -> Insert an Element in BST
 4 -> Delete an Element in BST
 5 -> Update an Element in BST
 6 -> Calculate the Height of BST
 7 -> Exit
Enter your choice : 1
Enter Node Value : 70
Enter your choice : 2
INORDER TRAVERSAL = [ 70, ]
Enter your choice : 3
Enter Node Value : 45
Enter your choice : 3
Enter Node Value : 20
Enter your choice : 3
Enter Node Value : 55
Enter your choice : 3
Enter Node Value : 67
Enter your choice : 3
Enter Node Value : 95
Enter your choice : 3
Enter Node Value : 97
Enter your choice : 3
Enter Node Value : 83
Enter your choice : 3
Enter Node Value : 75
Enter your choice : 2
INORDER TRAVERSAL = [ 20,  45,  55,  67,  70,  75,  83,  95,  97, ]
```

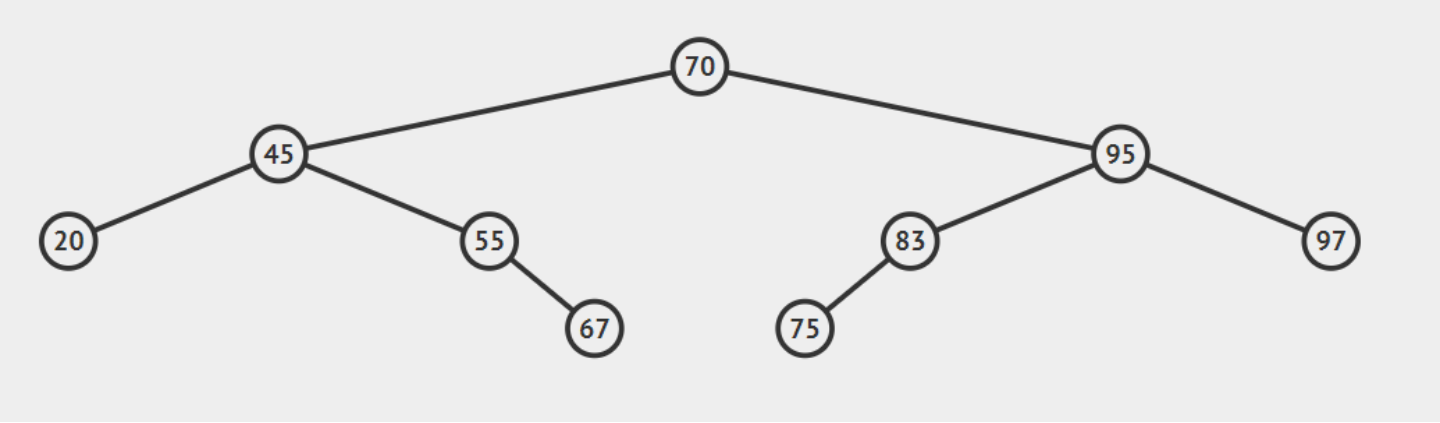


C.) Deletion of Element in Binary Search Tree

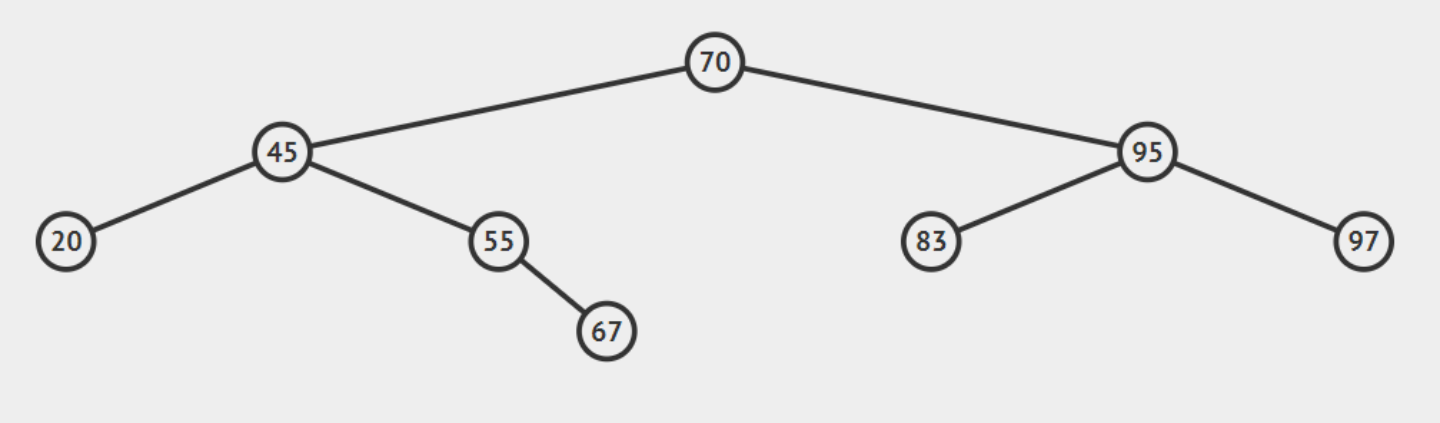
```
INORDER TRAVERSAL = [ 20,  45,  55,  67,  70,  75,  83,  95,  97, ]
Enter your choice : 4
Enter Node Value to be Deleted : 80
Given Element Doesn't Exist in BST!
Enter your choice : 4
Enter Node Value to be Deleted : 75
Enter your choice : 2
INORDER TRAVERSAL = [ 20,  45,  55,  67,  70,  83,  95,  97, ]
Enter your choice : 4
Enter Node Value to be Deleted : 55
Enter your choice : 2
INORDER TRAVERSAL = [ 20,  45,  67,  70,  83,  95,  97, ]
Enter your choice : 4
Enter Node Value to be Deleted : 45
Enter your choice : 2
INORDER TRAVERSAL = [ 20,  67,  70,  83,  95,  97, ]
```

1.) Invalid Query: Deletion of Element ["80"]

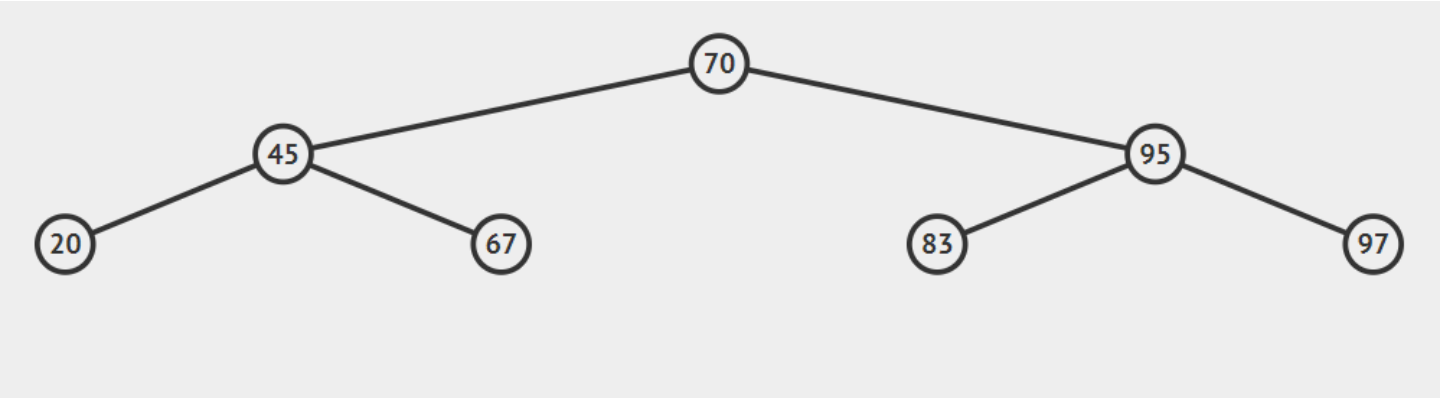
Output: "Given Element Doesn't Exist in BST!"



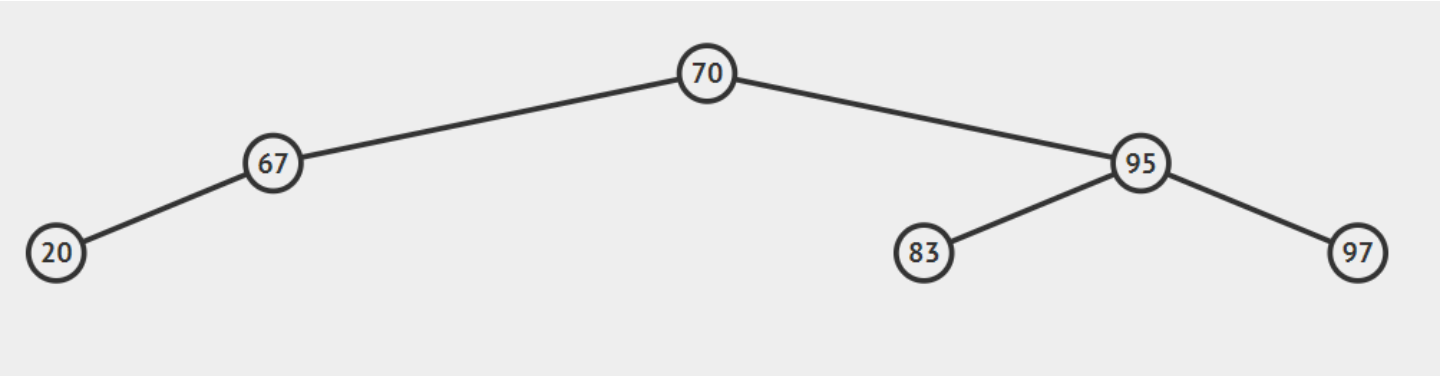
2.) Deletion of Element [Leaf Node "75"] [0 Children]



3.) Deletion of Element ["55"] [1 Children]

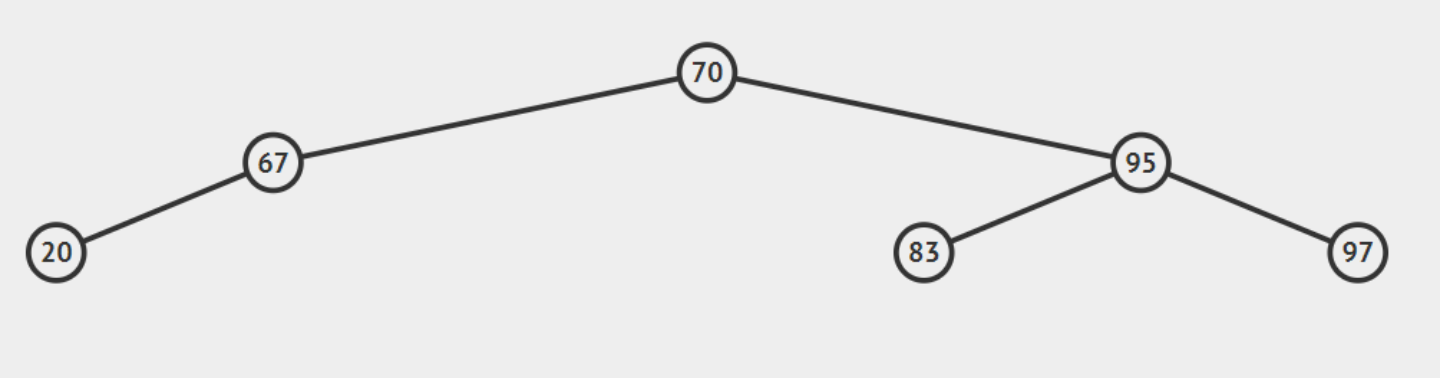


4.) Deletion of Element ["45"] [2 Children]

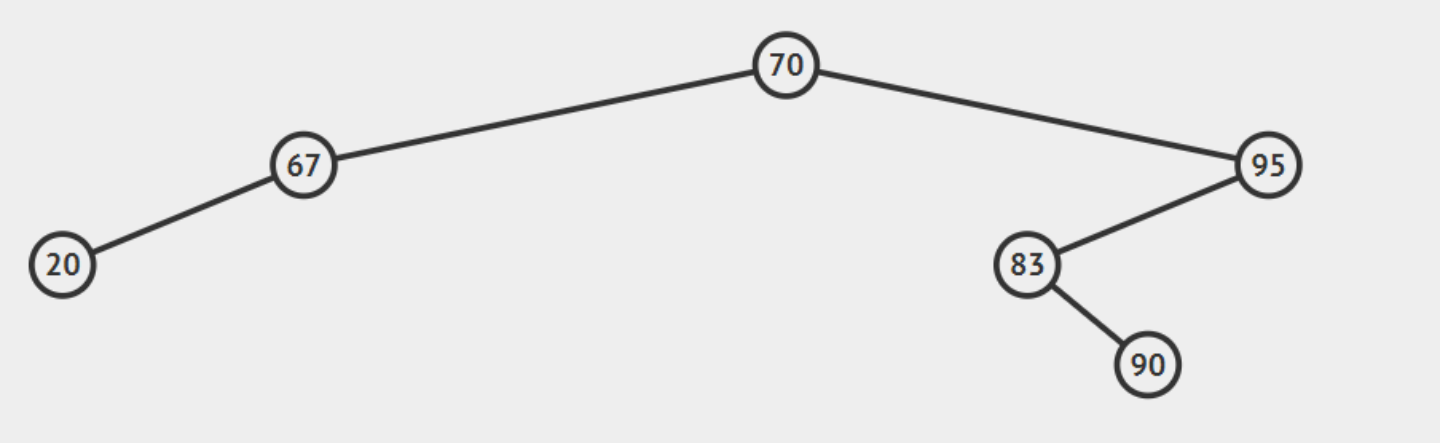


D.) Updating of Element in Binary Search Tree

```
Enter your choice : 5
Enter the Element to be Updated in BST : 97
Enter the Updated Element to be Replaced : 90
Enter your choice : 2
INORDER TRAVERSAL = [ 20,  67,  70,  83,  90,  95, ]
```

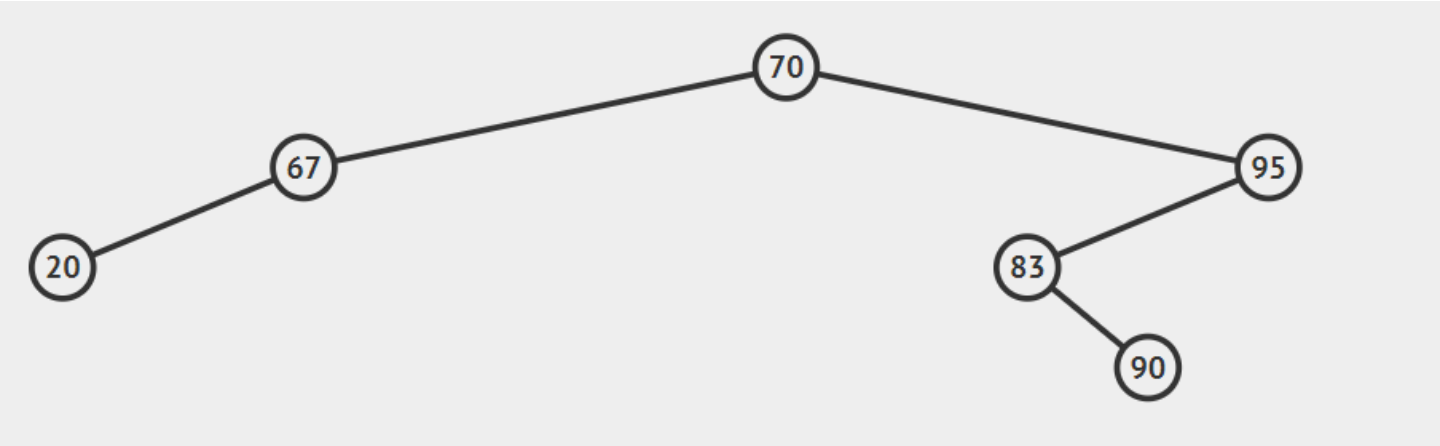


Let's Update the Element "97" to "90".



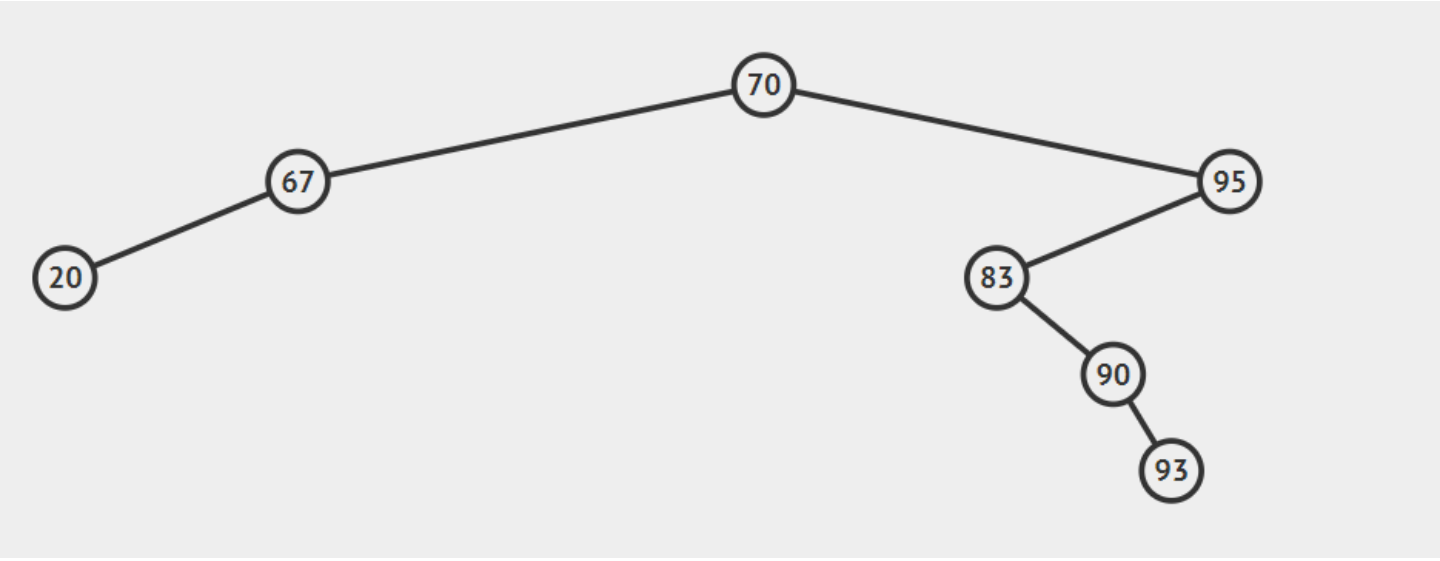
E.) Calculation of Height of Binary Search Tree

```
INORDER TRAVERSAL = [ 20,  67,  70,  83,  90,  95, ]
Enter your choice : 6
Height of BST : 4
Enter your choice : 3
Enter Node Value : 93
Enter your choice : 6
Height of BST : 5
Enter your choice : 7
```



Height of Above Tree is 4.

Now, Let's Insert "93" So that our Height Increases by 1.



Height of Above Tree is 5.