13. IMPLEMENTATION OF BINARY SEARCH TREE

Preamble

A binary search tree is a type of tree data structure that enables users to sort and store information. Because each node can only have two children, it is known as a binary tree. It is also known as a search tree because we can look up numbers in O(log(n)) time.

Steps - Creating a binary search tree

- Now, let's see the creation of binary search tree using an example.
- Suppose the data elements are - 45, 15, 79, 90, 10, 55, 12, 20, 50
- First, we have to insert 45 into the tree as the root of the tree.
- Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.
- Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

Steps - Searching in Binary search tree

Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order. The steps of searching a node in Binary Search tree are listed as follows -

- First, compare the element to be searched with the root element of the tree.
- If root is matched with the target element, then return the node's location.
- If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
- If it is larger than the root element, then move to the right subtree.
- Repeat the above procedure recursively until the match is found.
- If the element is not found or not present in the tree, then return NULL.

Steps - Deletion in Binary Search tree

In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated. To delete a node from BST, there are three possible situations occur as follows:

- The node to be deleted is the leaf node, or,
- The node to be deleted has only one child, and,
- The node to be deleted has two children

Implementation in C

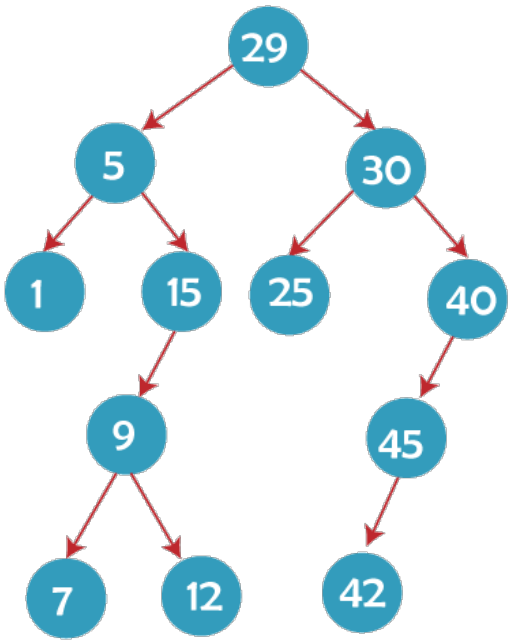
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
      int data;
      struct node *right_child;
      struct node *left_child;
};

struct node* new_node(int x)
{
      struct node *temp;
      temp = malloc(sizeof(struct node));
      temp->data = x;
      temp->left_child = NULL;
      temp->right_child = NULL;
      return temp;
}

struct node* search(struct node * root, int x)
{
      if (root == NULL || root->data == x)
            return root;
      else if (x > root->data)
            return search(root->right_child, x);
      else
            return search(root->left_child, x);
}

struct node* insert(struct node * root, int x)
{
```

```
	if (root == NULL)
		return new_node(x);
	else if (x > root->data)
		root->right_child = insert(root->right_child, x);
	else
		root -> left_child = insert(root->left_child, x);
	return root;
}
struct node* find_minimum(struct node * root)
{
	if (root == NULL)
		return NULL;
	else if (root->left_child != NULL)
		return find_minimum(root->left_child);
	return root;
}

struct node* delete(struct node * root, int x)
{

	if (root == NULL)
		return NULL;
	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
	{
		if (root->left_child == NULL && root->right_child == NULL)
		{
			free(root);
			return NULL;
```

```
            }
            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;
            }
            else
            {
                  struct node *temp = find_minimum(root->right_child);
                  root->data = temp->data;
                  root->right_child = delete(root->right_child,
                  temp->data);
            }
      }
      return root;
}
void inorder(struct node *root)
{
      if (root != NULL)
      {
            inorder(root->left_child);
            printf(" %d ", root->data);
            inorder(root->right_child);
      }
}
```

```
int main()
{
	struct node *root;
	root = new_node(20);
	insert(root, 5);
	insert(root, 1);
	insert(root, 15);
	insert(root, 9);
	insert(root, 7);
	insert(root, 12);
	insert(root, 30);
	insert(root, 25);
	insert(root, 40);
	insert(root, 45);
	insert(root, 42);
	inorder(root);
	printf("\n");
	root = delete(root, 1);
	root = delete(root, 40);
	root = delete(root, 45);
	root = delete(root, 9);
	inorder(root);
	printf("\n");
	return 0;
}
```

Sample Input and Output


29
5
30
1
15
25
40
9
45
7
12
42