© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

BINARY SEARCH TREE

Sonu Yadav, Jyoti

***Abstract*- Binary tree is a special tree.The maximum degree of binary tre is 2.There are two special cases of binary tree:-**

1. **Full binary tree**
2. **Complete binary tree**

The two ways of representation of binary search tree i.e linear representation and linked representation. The binary search tree is represented by a linked representation. A binary search tree (BST) is a special binary tree that is used for efficient searching of data.A BST stores data in such a way that binary search algorithm can be applied. In it we study the creation of a binary search tree ,searching in a binary search tree , insertion into a binary search tree and deletion of a node from a binary search tree.

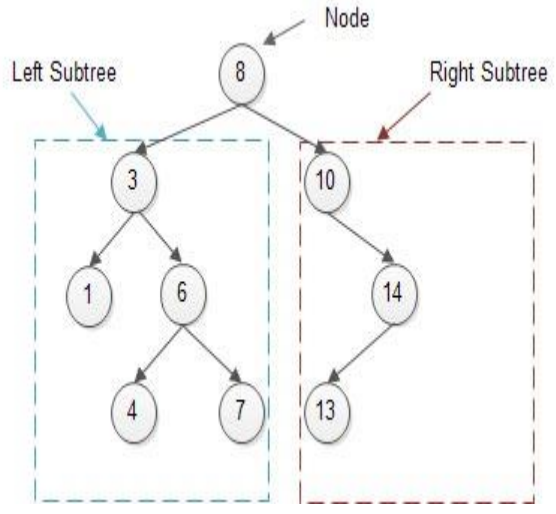
***Index Terms*- Binary tree , binary search tree, algorithm, linked representation.**

I. INTRODUCTION

Binary search tree Is a special tree. The maximum degree of this tree is 2. In binary tree ,each tree can have two or less than two children . A non empty binary tree consist of following:-

- A node called the root node
- A left sub-tree
- A right sub-tree

The root node have no parent .

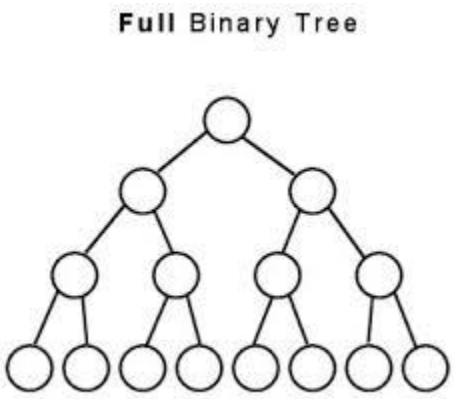


The two special cases of binary tree are:-

1. Full binary tree
2. Complete binary tree

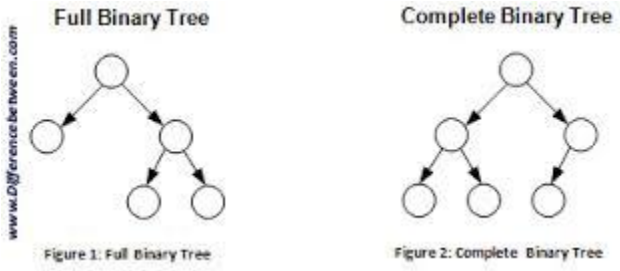
Full binary tree:-

It contains the maximum allowed number of nodes at all levels. This means that each node have exactly zero or two children.



Complete binary tree:-

It is a full tree except at the last level where all nodes must appear as far left as possible.



II. BINARY SEARCH TREE

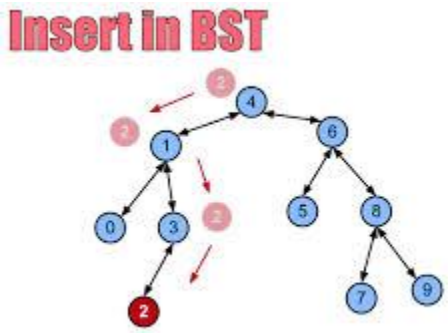
The BST is organized as per the following conditions on a binary tree:-

- The data to be stored must have unique key values.

- The key value of the nodes of right sub-tree are always more than key value of the root node.
- The left and right sub-tree are themselves BSTs.

Insertion in Binary Search Tree:

- Check whether root node is present or not(tree available or not). If root is NULL, create root node.
- If the element to be inserted is less than the element present in the root node, traverse the left sub-tree recursively until we reach T->left/T->right is NULL and place the new node at T->left(key in new node < key in T)/T->right (key in new node > key in T).
- If the element to be inserted is greater than the element present in root node, traverse the right sub-tree recursively until we reach T->left/T->right is NULL and place the new node at T->left/T->right.



Algorithm for insertion in Binary Search Tree:

```
TreeNode insert(int data, TreeNode T) {
    if T is NULL {
        T = (TreeNode *)malloc(sizeof (Struct
TreeNode));
        (Allocate Memory of new node and load
the data into it)
        T->data = data;
        T->left  = NULL;
        T->right = NULL;
    } else if T is less than T->left {
        T->left = insert(data, T->left);
        (Then node needs to be inserted in left
sub-tree.So,
         recursively traverse left sub-tree to find
the place
         where the new node needs to be inserted)
    } else if T is greater than T->right {
        T->right = insert(data, T->right);
        (Then node needs to be inserted in right
sub-tree
         So, recursively traverse right sub-tree to
find the
         place where the new node needs to be
inserted.)
    }
    return T;
}
```

Example:

Insert 20 into the Binary Search Tree.
Tree is not available. So, create root node and place 10 into it.

20

Insert 23 into the given Binary Search Tree. 23 > 20 (data in root). So, 23 needs to be inserted in the right sub-tree of 20.

```
20
  \
   23
```

Insert 13 into the given Binary Search Tree. 13 < 20(data in root). So, 13 needs to be inserted in left

sub-tree of 20.

```
      20
     /  \
    13   23
```

Insert 9 into the given Binary Search Tree.

```
      20
     /  \
    13   23
   /
  9
```

Inserting 14.

```
      20
     /  \
    13   23
   /  \
  9   14
```

Inserting 19.

```
      20
     /  \
    13   23
   /  \
  9   14
       \
        19
```

Inserting 21.

```
      20
     /  \
    13   23
   /  \  /
  9   14 21
       \
        19
```

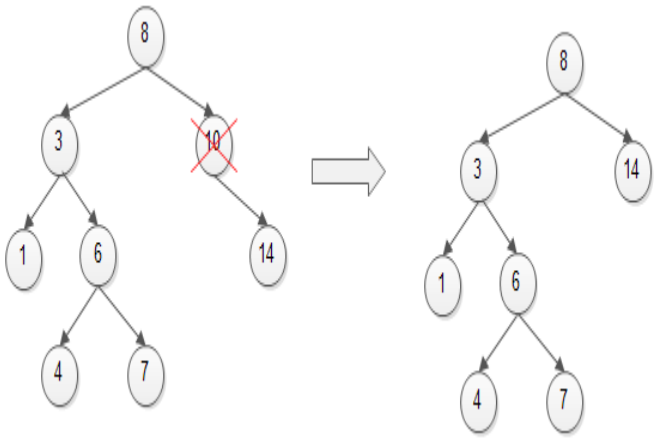
Inserting 27.

```
      20
     /  \
    13   23
   /  \  / \
  9   14 21 27
       \
        19
```

Inserting 24.

```
      20
     /  \
    13   23
   /  \  / \
  9   14 21 27
       \  /
        19 24
```

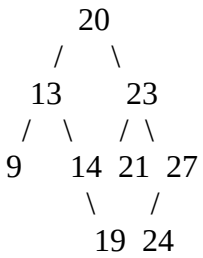
Deletion in Binary Search Tree:
How to delete a node from binary search tree? There are three different cases that needs to be considered for deleting a node from binary search tree.



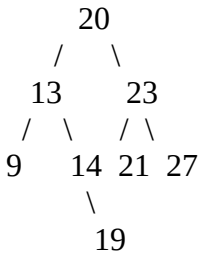
case 1: Node with no children (or) leaf node
case 2: Node with one child
case 3: Node with two children.

```
      20
     /  \
    13   23
   /  \  / \
  9   14 21 27
       \  /
        19 24
```

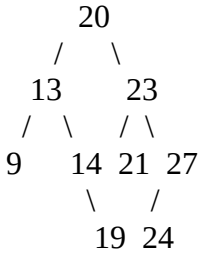
Case 1: Delete a leaf node/ node with no children.



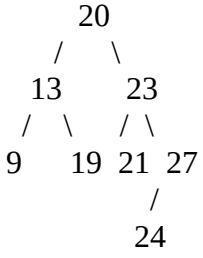
Delete 24 from the above binary search tree.



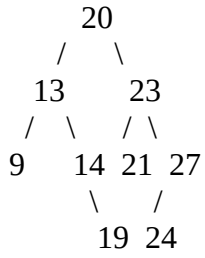
Case 2: Delete a node with one child.



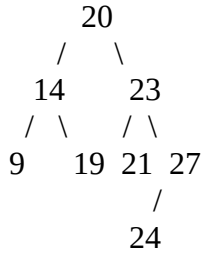
Delete 14 from above binary search tree.



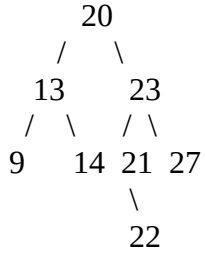
Case 3: Delete a node with two children.
Delete a node whose right child is the smallest node in the right sub-tree. (14 is the smallest node present in the right sub-tree of 13).



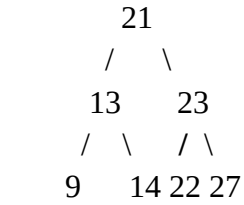
Delete 13 from the above binary tree. Find the smallest in the left subtree of 13. So, replace 13 with 14.



Delete 20 from the below binary search tree.



Find the smallest node in the right sub-tree of 20. And that smallest node is 21. So, replace 20 with 21. Since 21 has only one child(22), the pointer currently pointing to 21 is made to point to 23. So, the resultant binary tree would be the below.



III. CONCLUSION

In it we study about the binary tree and binary search tree . Both have the degree 2. In it contain the root node , left sub-tree and right sub-tree. In the binary search tree the right subtree have the greater value than the parent node and the left sub-tree have the less value than the parent node.In it we also study about the insertion and deletion a node in the binary search tree.

REFERENCES

[1] A.K Sharma, binary search tree
[2] Adel'son-Vel'skii, G. M. and Landis, E. M. An
[3] Algorithm for the Organization of Information.
[4] Soviet Mathematics Doklady. Vol. 3, 1962. PP
[5] 1259–1263.
[6] [2] Martin, W. A. and Ness D. N. Optimal Binary
[7] Trees Grown with a Sorting Algorithm.
[8] Commun. of the ACM. 15, 1972. PP 88-93.
[9] [3] Day, A. C. Balancing a Binary Tree. Computer
[10] Journal. 19, 1976. PP 360-361.
[11] [4] Eppinger, J. L. An Empirical Study of Insertion
[12] and Deletion in Binary Search Trees.
[13] Commun.of the ACM. 26, 1983. PP 663-669.
[14] [5] Chang, H. and Iyengar, S. S. Efficient
[15] Algorithms to Globally Balance a Binary Search
[16] Tree. Commun. of the ACM. 27, 1984. PP 695-
[17] 702.
[18] [6] Stout, F. and Bette, L. W. Tree Rebalancing
[19] Pearson Education Asia, 1999.