TREE, TYPES OF TREES, TRAVERSAL TECHNIQUES AND ITS VARIOUS OPERATIONS IN DATA STRUCTURE

By

Prof. B J Gorad, BECSE, M.Tech CST, PHD(CSE)*
Assistant Professor, CSE, SITCOE, Ichalkaranji, Kolhapur, Maharashtra

http://www.simplycs.in

http://www.sitcoe.org.in
Tree Basics - Introduction

- Tree is a collection of nodes, one of which can be called the root node, say r, along with a relation of parenthood that creates a hierarchical structure on the nodes.
- The root can have zero or more nonempty subtrees, $T_1, T_2, T_3, \ldots, T_k$.
- Each root of a subtree is said to be a child of r and r is the parent of each subtree root. This can be recursive in nature.

![Tree Diagram]

- In any tree with N nodes, there will be a maximum of N-1 edges.
- In a tree, every individual element is called a 'node'.

http://www.simplycs.in  
http://www.sitcoe.org.in
Basic Terms and Terminologies

1. **Root**
   In a tree data structure, the first node is called as **Root Node**.
   Every tree must have root node. We can say that root node is the origin of tree data structure.
   In any tree, there must be only one root node. We never have multiple root nodes in a tree.

2. **Edge**
   In a tree data structure, the connecting link between any two nodes is called as **EDGE**.
   In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.
3. Parent

In a tree data structure, the node which is predecessor of any node is called as **PARENT NODE**.

In simple words, the node which has branch from it to any other node is called as parent node.

Parent node can also be defined as "**The node which has child / children**".

4. Child

In a tree data structure, the node which is descendant of any node is called as **CHILD Node**.

In simple words, the node which has a link from its parent node is called as child node.

In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.
Basic Terms and Terminologies

5. Siblings
In a tree data structure, nodes which belong to same Parent are called as SIBLINGS.
In simple words, the nodes with same parent are called as Sibling nodes.

6. Leaf
In a tree data structure, the node which does not have a child is called as LEAF Node.
In simple words, a leaf is a node with no child.
In a tree data structure, the leaf nodes are also called as External Nodes.
External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.
7. Internal Nodes

In a tree data structure, the node which has at least one child is called as **INTERNAL Node**.

In simple words, an internal node is a node with at least one child.

In a tree data structure, nodes other than leaf nodes are called as **Internal Nodes**.

The root node is also said to be **Internal Node** if the tree has more than one node. Internal nodes are also called as 'Non-Terminal' nodes.

8. Degree

In a tree data structure, the total number of children of a node is called as **DEGREE** of that Node.

In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as 'Degree of Tree.'
Basic Terms and Terminologies

9. Level
In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...

In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).

10. Height
In a tree data structure, the total number of edges from leaf node to a particular node in the longest path is called as HEIGHT of that Node.

In a tree, height of the root node is said to be height of the tree.

In a tree, height of all leaf nodes is '0'.

http://www.simplycs.in
11. Depth
In a tree data structure, the total number of edges from root node to a particular node is called as DEPTH of that Node.

In a tree, the total number of edges from root node to a leaf node in the longest path is said to be Depth of the tree.

In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree.

In a tree, depth of the root node is '0'.

12. Path
In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes. Length of a Path is total number of nodes in that path.

In below example the path A - B - E - J has length 4.

http://www.simplycs.in
13. Sub Tree
In a tree data structure, each child from a node forms a subtree recursively.
Every child node will form a subtree on its parent node.
A tree data structure can be represented in two methods. Those methods are as follows...

- List (Linked List) Representation
- Left Child - Right Sibling Representation

Consider the right shown tree...

1. List Representation

In this representation, we use two types of nodes, one for representing the node with data and another for representing only references.

We start with a node with data from root node in the tree.

Then it is linked to an internal node through a reference node and is linked to any other node directly. This process repeats for all the nodes in the tree.

The above tree example can be represented using List representation as shown in right...
2. Left Child - Right Sibling Representation

In this representation, we use list with one type of node which consists of three fields namely Data field, Left child reference field and Right sibling reference field. Data field stores the actual value of a node, left reference field stores the address of the left child and right reference field stores the address of the right sibling node.

Graphical representation of that node is as shown at right...

In this representation, every node's data field stores the actual value of that node. If that node has left child, then left reference field stores the address of that left child node otherwise that field stores NULL.

If that node has right sibling then right reference field stores the address of right sibling node otherwise that field stores NULL.

The above tree example can be represented using Left Child - Right Sibling representation as shown...
**Binary Tree**

In a normal tree, every node can have any number of children.

Binary tree is a special type of tree data structure in which every node can have a **maximum of 2 children**.

One is known as left child and the other is known as right child.

- A tree in which every node can have a maximum of two children is called as Binary Tree.
- In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

**Example**

![Binary Tree Diagram](https://www.sitcoe.org.in)
Types of Binary Tree

There are different types of binary trees and they are...

1. Strictly Binary Tree
   - In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none.
   - That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follows...
   - A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree
   - Strictly binary tree is also called as Full Binary Tree or Proper Binary Tree or 2-Tree
   - Strictly binary tree data structure is used to represent mathematical expressions.

http://www.simplycs.in
TYPES OF BINARY TREE

2. Complete Binary Tree

- In a binary tree, every node can have a maximum of two children.
- But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be $2^\text{level}$ number of nodes.
- For example at level 2 there must be $2^2 = 4$ nodes and at level 3 there must be $2^3 = 8$ nodes.
- A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.
- Complete binary tree is also called as Perfect Binary Tree.
3. Extended Binary Tree

- A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.
- The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary tree.
- In below figure, a normal binary tree is converted into full binary tree by adding dummy nodes (In pink colour).
4. Skewed Binary Tree

- A Skewed Binary tree could be skewed to the left or it could be skewed to the right.
- In left Skewed Binary tree most of the nodes have left child without having corresponding right child.
- In Right Skewed Binary tree most of the nodes have Right child without having corresponding left child.
A binary tree data structure is represented using two methods. Those methods are as follows...

- **Array Representation**
- **Linked List Representation**

Consider the binary tree shown at right side.

1. **Array Representation**

In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree.

Nodes are numbered sequentially level by level and left to right.

Consider the above example of binary tree and it is represented as follows...

<table>
<thead>
<tr>
<th>Tree Elements</th>
<th>15</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>K</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Position of Left Child of $i = 2 \times i$;
Position of Right Child of $i = 2 \times i + 1$
Position of parent of $i = i/2$
Position of right sibling of $i = i+1$
Position of left sibling of $i = i - 1$
Binary Tree Representation Techniques

Linked List Representation of Binary Tree

- Linked List Representation of Binary Tree is More Efficient than Array Representation.
- Node in Binary tree consists of 3 fields
  - Data
  - Address of Left Child
  - Address of Right Child
- Left and Right are pointer type fields. Left holds address of Left Child and Right holds the address of right child.
- In c Language, Structure can be used to define node of tree.

```c
typedef struct tnode
{
    int data;
    struct tnode *left;
    struct tnode *right;
}tnode;
```
Binary Tree Traversal

- When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed.
- In any binary tree displaying order of nodes depends on the traversal method.
- Most of the Tree requires traversing a tree in particular order.
- Traversing a tree means it is the process of visiting every nodes of tree and exactly once.
- Since Binary tree is defined recursive in nature, tree traversal can be also in recursive in nature.

3 ways of Traversing a tree

- **Pre-order (VLR)** – Visit Root First, Traverse Left Sub tree in preorder, then Right Sub tree in Pre-order
- **In-Order (LVR)** - Traverse Left Sub tree in In-order, Visit Root Node, then Right Sub tree in In-order
- **Post-Order (LRV)** - Traverse Left Sub tree in post-order, then Right Sub tree in Post-order, Visit Root Node.
Binary Tree Traversal

Example -
Consider given tree and traverse the tree using Preorder, post order and in order methods.

Pre-Order Traversal –
Example 1:
Example 2:

In-Order Traversal –
Example 1:
Example 2:

Post-Order Traversal -
Example 1:
Example 2:
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, In-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.

```c
#include<stdio.h>

typedef struct tnode
{
    int data;
    struct tnode *left;
    struct tnode *right;
}tnode;

tnode *create();
int count(tnode *p);
int count_leaf(tnode *p);
int cal_height(tnode *p);
void display_in(tnode *p);
void display_post(tnode *p);
void display_pre(tnode *p);

void main()
{
    int value, choice,n,h;
    tnode *R;
    clrscr();
    while(1)
    {
        1:1
    }
}
```

http://www.simplycs.in
http://www.sitcoe.org.in
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, in-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.

```c
while(1)
{
    printf("\n\tEnter Choice:");
    scanf("%d", &choice);
    switch(choice)
    {
    case 0: exit(0);
    case 1: R=create();
            printf("\n\tRoot node at: %d", R);
            break;
    case 2: display_pre(R);
            break;
    case 3: display_post(R);
            break;
    case 4: display_in(R);
            break;
    case 5: n=count(R);
            printf("\n\tNumber of Nodes: %d", n);
            break;
    case 6: n=count_leaf(R);
            printf("\n\tNumber of Leaf Nodes: %d", n);
    }
}
```
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, In-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.

```c
#include <stdio.h>

typedef struct node
{
    int data;
    struct node *left;
    struct node *right;
} node;

node *create()
{
    node *p;
    int x;
    printf("Enter Data (-1 for No Data): ");
    scanf("%d", &x);
    if (x == -1)
        return NULL;
    else
    {
        p = (node *) malloc(sizeof(node));
        p->data = x;
        p->left = create();
        p->right = create();
    }
    return p;
}

int cal_height(node *p)
{
    if (p == NULL) return -1;
    else
    {
        int left_height = cal_height(p->left);
        int right_height = cal_height(p->right);
        return 1 + (left_height > right_height ? left_height : right_height);
    }
}

int count_nodes(node *p)
{
    if (p == NULL) return 0;
    else
    {
        return 1 + count_nodes(p->left) + count_nodes(p->right);
    }
}

int count_leaf_nodes(node *p)
{
    if (p == NULL) return 0;
    else
    {
        if (p->left == NULL && p->right == NULL) return 1;
        else
        {
            return count_leaf_nodes(p->left) + count_leaf_nodes(p->right);
        }
    }
}

int main()
{
    node *root = create();
    printf("Height of Tree: %d\n", cal_height(root));
    printf("Number of Nodes: %d\n", count_nodes(root));
    printf("Number of Leaf Nodes: %d\n", count_leaf_nodes(root));
    return 0;
}
```
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, In-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.

```c
p=(tnode*)malloc(sizeof(tnode));
p->data=x;
printf("Next Enter Left Child of %d",x);
p->left=create();
printf("Next Enter Right Child of %d",x);
p->right=create();
return p;

void display_pre(tnode *p)
{
    if(p!=NULL)
    {
        printf("%d",p->data);
        display_pre(p->left);
        display_pre(p->right);
    }
}

void display_post(tnode *p)
{
    if(p!=NULL)
    {
```

http://www.simplycs.in

http://www.sitcoe.org.in
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, in-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, in-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.

```c
if (p == NULL)
    return 0;
int i = 1 + count(p->left) + count(p->right);
    printf("\n\t Number of Nodes:%d", i);
return i;
}
int count_leaf(tnode *p)
{
    int i = 0;
    if (p == NULL)
        return 0;
    if (p->left == NULL && p->right == NULL)
        return 1;
    i = i + count_leaf(p->left) + count_leaf(p->right);
    printf("\n\t Number of Leaf Nodes:%d", i);
    return i;
}
int cal_height(tnode *p)
{
    int h1, hr;
    if (p == NULL)
    ```
Write C Program to Create Binary Tree and Perform Traverse (Pre-order, Post-order, in-order), Calculate Height, Count No of Nodes, No of Leaf Nodes, etc.

```c
int cal_height(tnode *p)
{
    int hl, hr;
    if(p==NULL)
        return 0;
    if(p->left==NULL && p->right==NULL)
        return 0;
    hl=cal_height(p->left);
    hr=cal_height(p->right);
    if(hl>hr)
        return (hl+1);
    return (hr+1);
}
```
Creation of Binary Tree from Traversal Sequence

1. Preorder and In order is given
2. Post order and In order is given

Creation of Binary Tree from preorder and in order

In order – E A C K F H D B G
Preorder – F A E K C D H G B

Creation of Binary Tree from post order and in order

In order – B I D A C G E H F
Post order – I D B G C H F E A
Creation of Binary Tree from Traversal Sequence

Create Binary tree for Following Examples-

Creation of Binary Tree from preorder and in order

- Preorder – G, B, Q, A, C, K, E, F, D, E, R, H

Creation of Binary Tree from post order and in order

- In order – B, C, A, E, D, G, H, F, I
- Post order – C, B, E, H, G, I, F, D, A
Binary Search Tree
Binary Search Tree is Binary Tree, which is either empty or in which each node contains a key, that satisfies the following condition:

1. All keys are distinct.
2. For every node X, values which are smaller than its node are in the left subtree.
3. For every node X, values which are larger than its node are in the right subtree.

Example:

```
   X
   / \
  Y   Z
```

X > Y
X < Z
Binary Search Tree

Examples

Binary search trees

Not a binary search tree

http://www.simplycs.in

http://www.sitcoe.org.in
A binary tree is simply a tree in which each node can have at most two children.

A binary search tree is a binary tree in which the nodes are assigned values, with the following restrictions:

- No duplicate values.
- The left subtree of a node can only have values less than the node.
- The right subtree of a node can only have values greater than the node and recursively defined.
- The left subtree of a node is a binary search tree.
- The right subtree of a node is a binary search tree.

http://www.simplycs.in  http://www.sitcoe.org.in
Example Binary Searches

- search (root, 25 )

10 < 25, right
30 > 25, left
25 = 25, found

5 < 25, right
45 > 25, left
30 > 25, left
10 < 25, right
25 = 25, found
Algorithm for Binary Search Tree

• A) compare ITEM with the root node N of the tree
  • i) if ITEM<N, proceed to the left child of N.
  • ii) if ITEM>N, proceed to the right child of N.
• B) repeat step (A) until one of the following occurs
  • i) we meet a node N such that ITEM=N, i.e. search is successful.
  • ii) we meet an empty sub tree, i.e. the search is unsuccessful.
Binary Tree Insertion
## Binary Search Tree Insertion

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>10</th>
<th>8</th>
<th>4</th>
<th>6</th>
<th>3</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
</table>

http://www.simplycs.in

http://www.sitcoe.org.in
Binary Search Tree Insertion
Binary Search Tree Insertion

1 10 8 4 6 3 2 5
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1
Binary Search Tree Insertion
Binary Search Tree Insertion

http://www.simplycs.in
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
## Binary Search Tree Insertion

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>8</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

1

10

[http://www.simplycs.in](http://www.simplycs.in)
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
Binary Search Tree Insertion

http://www.simplycs.in
Binary Search Tree Insertion

```
1  10  8  4  6  3  2  5
```

Diagram:
```
1
|
10
```

http://www.simplycs.in
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
Binary Search Tree Insertion

http://www.simplycs.in
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
Binary Search Tree Insertion

http://www.simplycs.in
Binary Search Tree Insertion

| 1 | 10 | 8 | 4 | 6 | 3 | 2 | 5 |

http://www.simplycs.in
Binary Search Tree Insertion

http://www.simplycs.in
Binary Search Tree Insertion
Binary Search Tree Insertion

```
1 10 8 4 6 3 2 5
```

http://www.simplycs.in
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

http://www.simplycs.in
Binary Search Tree Insertion

1 | 10 | 8 | 4 | 6 | 3 | 2 | 5

1 → 10 → 8 → 4 → 6

Diagram showing the insertion of 3 into a binary search tree.
Binary Search Tree Insertion

1 | 10 | 8 | 4 | 6 | 3 | 2 | 5

1

10

8

4

6

Binary Search Tree Insertion
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8

4

6
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8

4

6

Binary Search Tree Insertion

1 10 8 4 6 3 2 5
Binary search  Tree Insertion
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8

4

6

2

5
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8

4

3

6

Binary Search Tree Insertion
Binary Search Tree Insertion

1 10 8 4 6 3 2 5
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

Tree representation:
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

1

10

8

4

3

6

3

6
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

Diagram of a binary search tree with nodes 1, 10, 8, 4, 6, 3, and 2. Insertion process is shown.
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

```
1
  \-- 10
    \-- 8
       \-- 4
         \-- 3
           \-- 6
```

2

```
Binary Search Tree Insertion

1 10 8 4 6 3 2 5
Binary Search Tree Insertion

```
1 10 8 4 6 3 2 5
```

```
1
  \-----
   \   \  \ 10
    \   \  \
     \   \\
      \  \\ 8
       \ \\
        \4
         \ \\
          \3
           \ \\
            \6
```

---

1. Insert 2 into the binary search tree.
2. Insert 5 into the binary search tree.
Binary Search Tree Insertion
Binary Search Tree Insertion

1 10 8 4 6 3 2 5
Binary Search Tree Insertion
Binary Search Tree Insertion

1 10 8 4 6 3 2 5

Diagram showing the insertion of the number 5 into a binary search tree.
Binary Tree Insertion
Binary Tree Insertion

1 10 8 4 6 3 2 5

2 3 4 5 6 8 10 1

Binary Tree Insertion
Binary Tree Insertion

1 10 8 4 6 3 2 5

1
  
10

8

4

6

3

2

6

5
Binary Tree Insertion

1 10 8 4 6 3 2 5

1  
   10
  / 
 8  
 /  
4  
/  
3  
/  
2  
/  
6
Binary Tree Insertion
Binary Search Tree Deletion
Binary Tree Deletion
Binary Tree Deletion
Binary Tree Deletion
Binary Tree Deletion
Binary Tree Deletion

```
1 10 8 4 6 3 5
```

```
1
  10
    8
     4
      3 6
        5
```
Binary Tree Deletion
Binary Tree Deletion

1 10 8 4 6 3 5
Binary Tree Deletion

1 10 8 4 6 3 5

Diagram of a binary tree with nodes 1, 10, 4, 3, 5.
Binary Tree Deletion

1 10 8 4 6 3 5

1 → 10
10 → 4
4 → 3 6
3 → 5
6
Binary Tree Deletion

1 10 8 4 6 3 5
Binary Tree Deletion
Binary Tree Deletion
Binary Tree Deletion
Binary Tree Deletion
Binary Tree Deletion

1 10 4 6 3 5

Diagram of a binary tree with nodes labeled 1, 10, 4, 6, 3, 5.
Binary Tree Deletion

1  10  4  6  3  5

Diagram of a binary tree with nodes labeled 1, 10, 4, 6, 3, 5.
Binary Tree Deletion

1 10 4 6 3 5

Diagram of a binary tree with nodes 1, 10, 5, 3, and 6.
Binary Tree Deletion
Binary Tree Deletion

```
1 10 4 6 3 5
```

Diagram of a binary tree with nodes labeled 1, 10, 4, 6, 3, and 5.
Binary Tree Deletion
Binary Tree Deletion

http://www.simplycs.in
Types of BST

- AVL Tree
- Red-Black Tree
- Splay Tree
AVL Tree

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

The tree became unbalanced after inserting key 95.

After the tree is rebalanced using rotation we have:
Red Black Tree

• Every node has a color either red or black.
• Root of tree is always black.
• There are no two adjacent red nodes (A red node cannot have a red parent or red child).
• Every path from root to a NULL node has same number of black nodes.

A chain of 3 nodes is nodes is not possible in Red-Black Trees.
Following are NOT Red-Black Trees

\[
\begin{array}{ccc}
30 & 30 & 30 \\
/ & / & / \\
20 & 20 & 20 \\
/ & / & / \\
10 & 10 & 10 \\
\end{array}
\]
Violates Violates Violates

Property 4. Property 4 Property 3

Following are different possible Red-Black Trees with above 3 keys

\[
\begin{array}{ccc}
20 & 20 \\
/ & / \\
10 & 10 \\
/ & / \\
NIL & NIL \\
\end{array}
\]

\[
\begin{array}{ccc}
20 & 20 \\
/ & / \\
10 & 10 \\
/ & / \\
NIL & NIL \\
\end{array}
\]

http://www.simplycs.in
http://www.sitcoe.org.in
Splay Tree

Automatically moves frequently accessed elements nearer to the root for quick to access
# Complexity in BST

<table>
<thead>
<tr>
<th>Operation</th>
<th>Average</th>
<th>Worst Case</th>
<th>Best Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Insertion</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Deletion</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Applications of BST

• Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
• Storing a set of names, and being able to lookup based on a prefix of the name. (Used in internet routers.)
• Storing a path in a graph, and being able to reverse any subsection of the path in $O(\log n)$ time. (Useful in travelling salesman problems).
• Finding square root of given number
• Allows you to do range searches efficiently.
Applications of BST

- Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- Storing a set of names, and being able to lookup based on a prefix of the name. (Used in internet routers.)
- Storing a path in a graph, and being able to reverse any subsection of the path in $O(\log n)$ time. (Useful in travelling salesman problems).
- Finding square root of given number
- Allows you to do range searches efficiently.
The B-tree is a generalization of a binary search tree in that a node can have more than two children.

Nodes have multiple values and are in sorted order.

In computer science, a B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.

Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. B-trees are a good example of a data structure for external memory.

It is commonly used in databases and filesystems.
Properties of B-Tree
1) All leaves are at same level.
2) A B-Tree is defined by the term *minimum degree* ‘t’. The value of t depends upon disk block size.
3) Every node except root must contain at least t-1 keys. Root may contain minimum 1 key.
4) All nodes (including root) may contain at most 2t - 1 keys.
5) Number of children of a node is equal to the number of keys in it plus 1.
6) All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in range from k1 and k2.
7) B-Tree grows and shrinks from root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
8) Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(\log n).

Following is an example B-Tree of minimum degree 3. Note that in practical B-Trees, the value of minimum degree is much more than 3.

http://www.simplycs.in
http://www.sitcoe.org.in