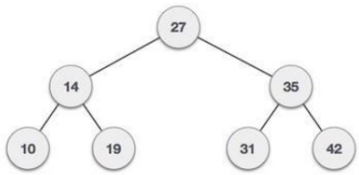
A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than or equal to its parent node's key.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as −

left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)



Basic Operations

Following are the basic operations of a tree −

- **Search** − Searches an element in a tree.
- **Insert** − Inserts an element in a tree.
- **Pre-order Traversal** − Traverses a tree in a pre-order manner.
- **In-order Traversal** − Traverses a tree in an in-order manner.
- **Post-order Traversal** − Traverses a tree in a post-order manner.

Search Operation

Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

Algorithm

- START
- 2. Check whether the tree is empty or not
- 3. If the tree is empty, search is not possible
- 4. Otherwise, first search the root of the tree.
- 5. If the key does not match with the value in the root, search its subtrees.
- 6. If the value of the key is less than the root value, search the left subtree
- 7. If the value of the key is greater than the root value, search the right subtree.
- 8. If the key is not found in the tree, return unsuccessful search.
- 9. END

Insert Operation

Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

Algorithm

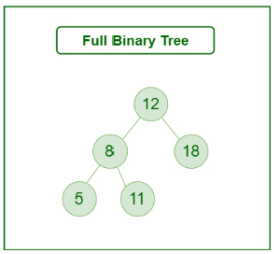
1 – START
2 – If the tree is empty, insert the first element as the root node of the tree. The following elements are added as the leaf nodes.
3 – If an element is less than the root value, it is added into the left subtree as a leaf node.
4 – If an element is greater than the root value, it is added into the right subtree as a leaf node.
5 – The final leaf nodes of the tree point to NULL values as their child nodes.
6 – END

Types of Binary Tree

Full Binary Tree

A Binary Tree is a full binary tree if every node has 0 or 2 children. The following are examples of a full binary tree. We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children.

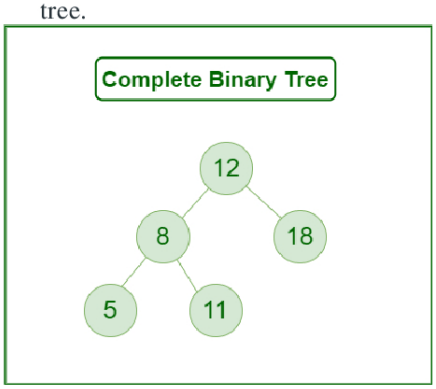
A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. It is also known as a proper binary tree.



1. Complete Binary Tree

A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible.
A complete binary tree is just like a full binary tree, but with two major differences:

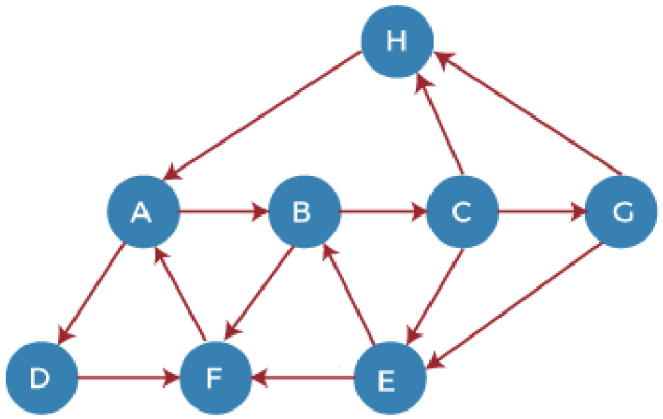
- Every level must be completely filled
- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.



2. Perfect Binary Tree

A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level.
The following are examples of Perfect Binary Trees.

Example of DFS algorithm



Adjacency Lists

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

Now, let's start examining the graph starting from Node H.

Step 1 - First, push H onto the stack.

1. STACK: H

Step 2 - POP the top element from the stack, i.e., H, and print it. Now, PUSH all the neighbors of H onto the stack that are in ready state.

1. Print: H]STACK: A

Step 3 - POP the top element from the stack, i.e., A, and print it. Now, PUSH all the neighbors of A onto the stack that are in ready state.

1. Print: A
2. STACK: B, D

Step 4 - POP the top element from the stack, i.e., D, and print it. Now, PUSH all the neighbors of D onto the stack that are in ready state.

1. Print: D
2. STACK: B, F

Step 5 - POP the top element from the stack, i.e., F, and print it. Now, PUSH all the neighbors of F onto the stack that are in ready state.

1. Print: F
2. STACK: B

Step 6 - POP the top element from the stack, i.e., B, and print it. Now, PUSH all the neighbors of B onto the stack that are in ready state.

1. Print: B
2. STACK: C

Step 7 - POP the top element from the stack, i.e., C, and print it. Now, PUSH all the neighbors of C onto the stack that are in ready state.

1. Print: C
2. STACK: E, G

Step 8 - POP the top element from the stack, i.e., G and PUSH all the neighbors of G onto the stack that are in ready state.

1. Print: G
2. STACK: E

Step 9 - POP the top element from the stack, i.e., E and PUSH all the neighbors of E onto the stack that are in ready state.

1. Print: E
2. STACK:

Now, all the graph nodes have been traversed, and the stack is empty.

Complexity of Depth-first search algorithm

The time complexity of the DFS algorithm is **O(V+E)**, where V is the number of vertices and E is the number of edges in the graph.

The space complexity of the DFS algorithm is O(V).

BFS

The applications of breadth-first-algorithm are given as follows -

- BFS can be used to find the neighboring locations from a given source location.

- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.
- BFS is used to determine the shortest path and minimum spanning tree.
- BFS is also used in Cheney's technique to duplicate the garbage collection.
- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Algorithm

The steps involved in the BFS algorithm to explore a graph are given as follows -

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

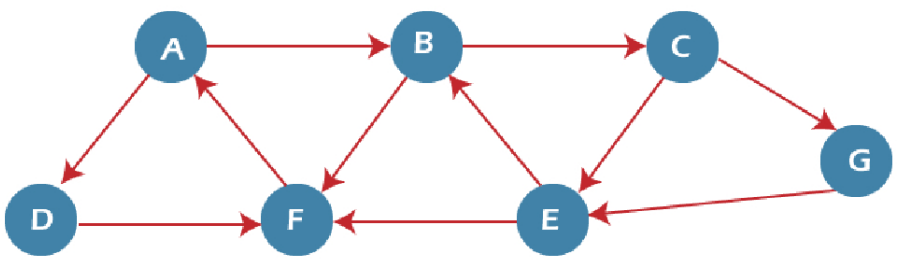
Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]



Adjacency Lists

A : B, D
B : C, F
C : E, G
G : E
E : B, F
F : A
D : F

n the above graph, minimum path 'P' can be found by using the BFS that will start from Node A and end at Node E. The algorithm uses two queues, namely QUEUE1 and QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.

Now, let's start examining the graph starting from Node A.

Step 1 - First, add A to queue1 and NULL to queue2.

1. QUEUE1 = {A}
2. QUEUE2 = {NULL}

Step 2 - Now, delete node A from queue1 and add it into queue2. Insert all neighbors of node A to queue1.

1. QUEUE1 = {B, D}
2. QUEUE2 = {A}

Step 3 - Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

1. QUEUE1 = {D, C, F}
2. QUEUE2 = {A, B}

Step 4 - Now, delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F since it is already inserted, so it will not be inserted again.

1. QUEUE1 = {C, F}
2. QUEUE2 = {A, B, D}

Step 5 - Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

1. QUEUE1 = {F, E, G}
2. QUEUE2 = {A, B, D, C}

Step 5 - Delete node F from queue1 and add it into queue2. Insert all neighbors of node F to queue1. Since all the neighbors of node F are already present, we will not insert them again.

1. QUEUE1 = {E, G}
2. QUEUE2 = {A, B, D, C, F}

Step 6 - Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

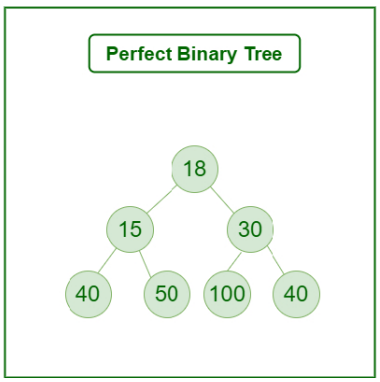
1. QUEUE1 = {G}
2. QUEUE2 = {A, B, D, C, F, E}

Complexity of BFS algorithm

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of BFS algorithm is **O(V+E)**, since in the worst case, BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

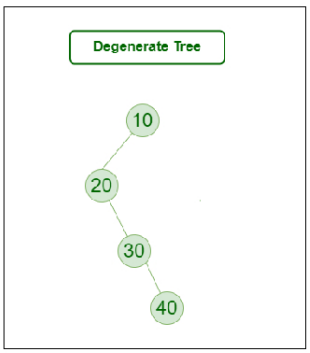
The space complexity of BFS can be expressed as **O(V)**, where V is the number of vertices.

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.



In a Perfect Binary Tree, the number of leaf nodes is the number of internal nodes plus 1
L = I + 1 Where L = Number of leaf nodes, I = Number of internal nodes.
A Perfect Binary Tree of height h (where the height of the binary tree is the number of edges in the longest path from the root node to any leaf node in the tree, height of root node is 0) has $2^{h+1} - 1$ node.

Degenerate (or pathological) tree
A Tree where every internal node has one child. Such trees are performance-wise same as linked list. A degenerate or pathological tree is a tree having a single child either left or right.



Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40, 100

When elements are given in a sequence,

- Always consider the first element as the root node.
- Consider the given elements and insert them in the BST one by one.

The binary search tree will be constructed as explained below-

Insert 50-

Insert 70-

As 70 > 50, so insert 70 to the right of 50.

insert 60-

- As 60 > 50, so insert 60 to the right of 50.
- As 60 < 70, so insert 60 to the left of 70.
-

Insert 20-

- As 20 < 50, so insert 20 to the left of 50.

Insert 90-

- As 90 > 50, so insert 90 to the right of 50.
- As 90 > 70, so insert 90 to the right of 70.

Insert 10-

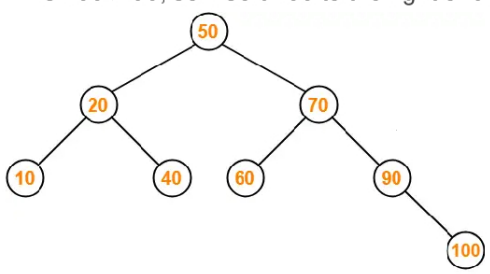
- As 10 < 50, so insert 10 to the left of 50.
- As 10 < 20, so insert 10 to the left of 20.

Insert 40-

- As 40 < 50, so insert 40 to the left of 50.
- As 40 > 20, so insert 40 to the right of 20.

Insert 100-

- As 100 > 50, so insert 100 to the right of 50.
- As 100 > 70, so insert 100 to the right of 70.
- As 100 > 90, so insert 100 to the right of 90.



Binary Search Tree

Tree Traversals (Inorder, Preorder and Postorder)

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

Inorder traversal

This technique follows the 'left root right' policy. It means that first left subtree is visited after that root node is traversed, and finally, the right subtree is traversed. As the root node is traversed between the left and right subtree, it is named inorder traversal.

The applications of Inorder traversal includes -

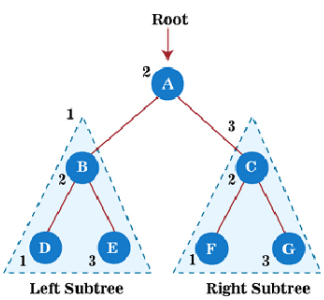
- It is used to get the BST nodes in increasing order.
- It can also be used to get the prefix expression of an expression tree.

So, in the inorder traversal, each node is visited in between of its subtrees.

Algorithm

1. Until all nodes of the tree are not visited

2. Step 1 - Traverse the left subtree recursively.
3. Step 2 - Visit the root node.
4. Step 3 - Traverse the right subtree recursively.



To know more about the inorder traversal in data structure, you can follow the link Inorder Traversal.

Preorder traversal

This technique follows the 'root left right' policy. It means that, first root node is visited after that the left subtree is traversed recursively, and finally, right subtree is recursively traversed. As the root node is traversed before (or pre) the left and right subtree, it is called preorder traversal.

So, in a preorder traversal, each node is visited before both of its subtrees.

The applications of preorder traversal include -

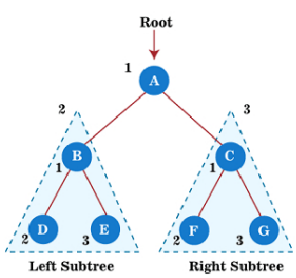
- It is used to create a copy of the tree.
- It can also be used to get the prefix expression of an expression tree.

Algorithm

1. Until all nodes of the tree are not visited
2.
3. Step 1 - Visit the root node
4. Step 2 - Traverse the left subtree recursively.
5. Step 3 - Traverse the right subtree recursively.

example

6. Now, let's see the example of the preorder traversal technique.



Now, start applying the preorder traversal on the above tree. First, we traverse the root node **A;** after that, move to its left subtree **B**, which will also be traversed in preorder.

So, for left subtree B, first, the root node **B** is traversed itself; after that, its left subtree **D** is traversed. Since node **D** does not have any children, move to right subtree **E**. As node E also does not have any children, the traversal of the left subtree of root node A is completed.

Now, move towards the right subtree of root node A that is C. So, for right subtree C, first the root node **C** has traversed itself; after that, its left subtree **F** is traversed. Since node **F** does not have any children, move to the right subtree **G**. As node G also does not have any children, traversal of the right subtree of root node A is completed.

Therefore, all the nodes of the tree are traversed. So, the output of the preorder traversal of the above tree is

A → B → D → E → C → F → G

To know more about the preorder traversal in the data structure, you can follow the link Preorder traversal.

Postorder traversal

This technique follows the 'left-right root' policy. It means that the first left subtree of the root node is traversed, after that recursively traverses the right subtree, and finally, the root node is traversed. As the root node is traversed after (or post) the left and right subtree, it is called postorder traversal.

So, in a postorder traversal, each node is visited after both of its subtrees.

The applications of postorder traversal include -

- It is used to delete the tree.
- It can also be used to get the postfix expression of an expression tree.

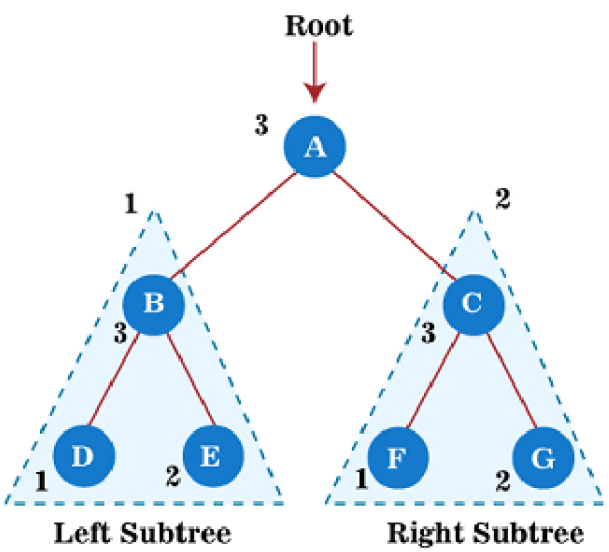
Algorithm

1. Until all nodes of the tree are not visited
2.
3. Step 1 - Traverse the left subtree recursively.
4. Step 2 - Traverse the right subtree recursively.
5. Step 3 - Visit the root node.

Example

1. Now, let's see the example of the postorder traversal technique.
2. Now, start applying the postorder traversal on the above tree. First, we traverse the left subtree B that will be traversed in postorder. After that, we will traverse the right subtree **C** in postorder. And finally, the root node of the above tree, i.e., **A**, is traversed.
3. So, for left subtree B, first, its left subtree **D** is traversed. Since node **D** does not have any children, traverse the right subtree **E**. As node E also does not have any children, move to the root node **B.** After traversing node **B,** the traversal of the left subtree of root node A is completed.
4. Now, move towards the right subtree of root node A that is C. So, for right subtree C, first its left subtree **F** is traversed. Since node **F** does not have any children, traverse the right subtree **G**. As node G also does not have any children, therefore, finally, the root node of the right subtree, i.e., **C,** is traversed. The traversal of the right subtree of root node A is completed.
5. At last, traverse the root node of a given tree, i.e., **A**. After traversing the root node, the postorder traversal of the given tree is completed.

6. Therefore, all the nodes of the tree are traversed. So, the output of the postorder traversal of the above tree is -
7. **D → E → B → F → G → C → A**
8. To know more about the postorder traversal in the data structure, you can follow the link Postorder traversal.



InOrder(root) visits nodes in the following order:

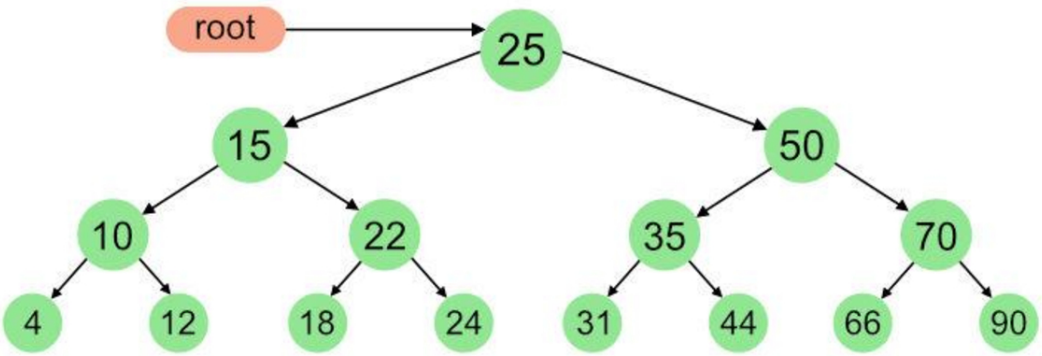
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:

4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Chapter 12 DFS

The step by step process to implement the DFS traversal is given as follows -

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

Algorithm

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT