Data Structures & Algorithms

Prepared by: Mohamed Ayman

Machine Learning Researcher

spring 2019


sw.eng.MohamedAyman@gmail.com


facebook.com/sw.eng.MohamedAyman


linkedin.com/in/eng-mohamed-ayman


codeforces.com/profile/Mohamed_Ayman

Binary Tree

Agenda

1- Binary Tree Definition

2- Binary Tree Representation

3- Binary Tree Terms

4- Binary Tree Traversal

5- BFS vs. DFS for Binary Tree

6- Time Complexity and Space Complexity

7- Search in Binary Tree

8- Delete Binary Tree


Let's
START UP

Agenda

1- Binary Tree Definition

2- Binary Tree Representation

3- Binary Tree Terms

4- Binary Tree Traversal

5- BFS vs. DFS for Binary Tree

6- Time Complexity and Space Complexity

7- Search in Binary Tree

8- Delete Binary Tree

Binary Tree Definition

- A Binary Tree is a special data structure used for data storage purposes.

A binary tree has a special condition that each node can have a maximum of two children.

- A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

- A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. A recursive definition using just set theory notions is that a (non-empty) binary tree is a tuple (L, S, R), where L and R are binary trees or the empty set and S is a singleton set.

Why Tree ?

Unlike Array and Linked List, which are linear data structures, tree is hierarchical (or non-linear) data structure.

1- One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer file system.

2- If we organize keys in form of a tree (with some ordering e.g., Binary Search Tree BST), we can search for a given key in moderate time (quicker than Linked List and slower than arrays). Self balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(Logn) for search.

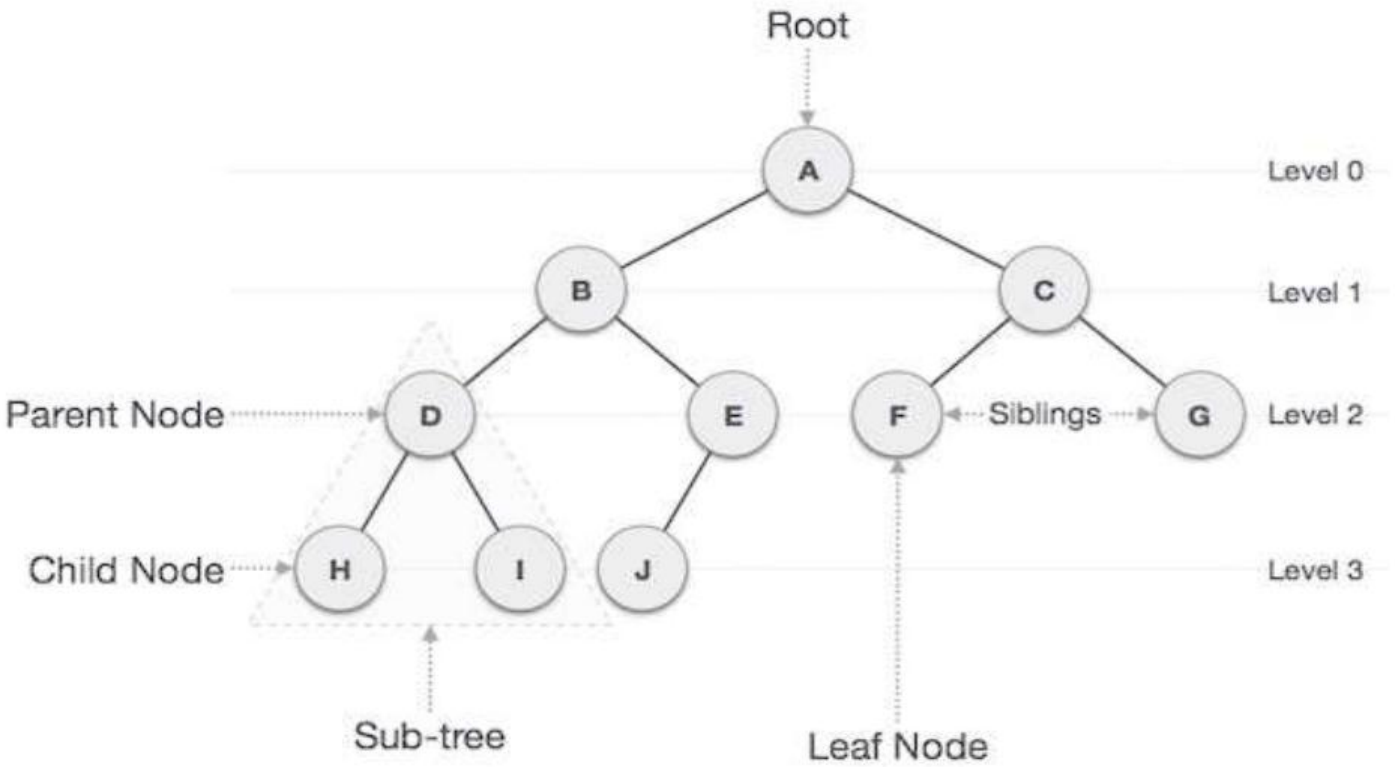
3- We can insert/delete keys in moderate time (quicker than Arrays and slower than Unordered Linked Lists). Self balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(log n) for insertion/deletion.

4- Like Linked Lists and unlike Arrays, Pointer implementation of trees don't have an upper limit on number of nodes as nodes are linked using pointers.

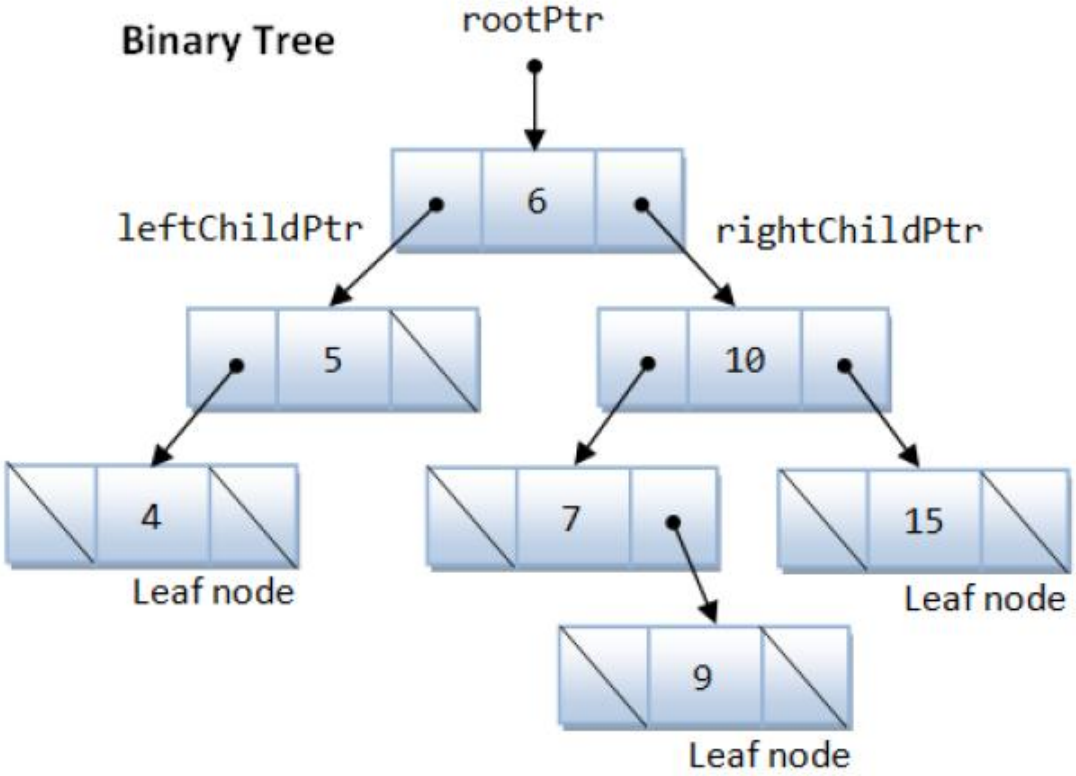
Agenda

- ✓ 1- Binary Tree Definition
- **2- Binary Tree Representation**
- 3- Binary Tree Terms
- 4- Binary Tree Traversal
- 5- BFS vs. DFS for Binary Tree
- 6- Time Complexity and Space Complexity
- 7- Search in Binary Tree
- 8- Delete Binary Tree

Binary Tree Representation



Binary Tree Representation



Binary Tree Node in C++

Link: repl.it/repls/SplendidKnowledgeableFact

```
4   // Binary Tree Node
5   struct node {
6       int data;
7       node* left;
8       node* right;
9   };
10
11  node* root;
```

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- 3- **Binary Tree Terms**
- 4- Binary Tree Traversal
- 5- BFS vs. DFS for Binary Tree
- 6- Time Complexity and Space Complexity
- 7- Search in Binary Tree
- 8- Delete Binary Tree

Binary Tree Terms

Path:	Path refers to the sequence of nodes along the edges of a tree.
Root:	The node at the top of the tree is called root.
	There is only one root per tree and one path from the root node to any node.
Parent:	Any node except the root node has one edge upward to a node called parent.
Child:	The node below a given node connected by its edge downward is called its child node.
Leaf:	The node which does not have any child node is called the leaf node.
Sub-tree:	Sub-tree represents the descendants of a node.
Visiting:	Visiting refers to checking the value of a node when control is on the node.
Traversing:	Traversing means passing through nodes in a specific order.
Levels:	Level of a node represents the generation of a node. If the root node is at level 0,
	then its next child node is at level 1, its grandchild is at level 2, and so on.
Keys:	Key represents a value of node based on which a search operation is to be carried out for a node.

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- ✓ 3- Binary Tree Terms
- **4- Binary Tree Traversal**
- 5- BFS vs. DFS for Binary Tree
- 6- Time Complexity and Space Complexity
- 7- Search in Binary Tree
- 8- Delete Binary Tree

Binary Tree Traversal

- 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 random access a node in a tree.

- A Tree is typically traversed in two ways, BFS & DFS

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- ✓ 3- Binary Tree Terms
- ✓ 4- Binary Tree Traversal
- **5- BFS vs. DFS for Binary Tree**
- 6- Time Complexity and Space Complexity
- 7- Search in Binary Tree
- 8- Delete Binary Tree

BFS vs. DFS for Binary Tree

A Tree is typically traversed in two ways:

1. Breadth First Traversal

    Level Order Traversal

2. Depth First Traversals

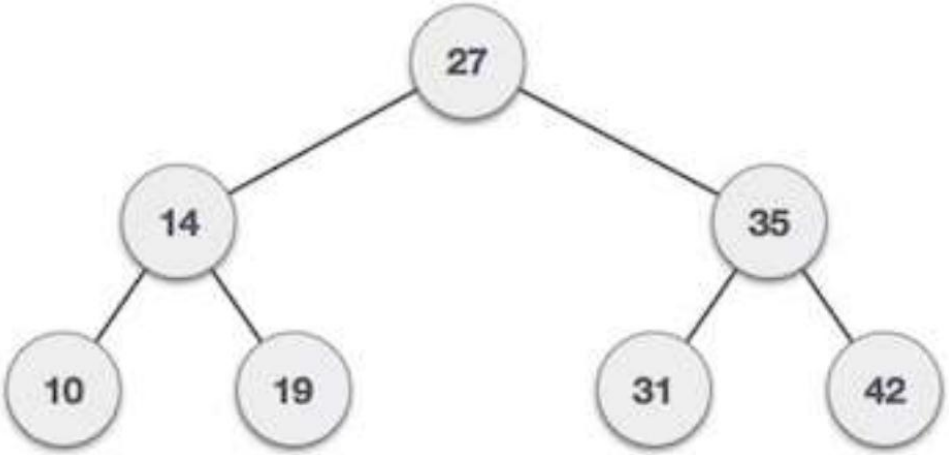
    In-order Traversal (Left-Root-Right)

    Pre-order Traversal (Root-Left-Right)

    Post-order Traversal (Left-Right-Root)

Level Order Traversal

Breadth First Traversal : 27 14 35 10 19 31 42



Level Order Traversal Algorithm

Algorithm:

For each node, first the node is visited and then it's child nodes are put in a FIFO queue.

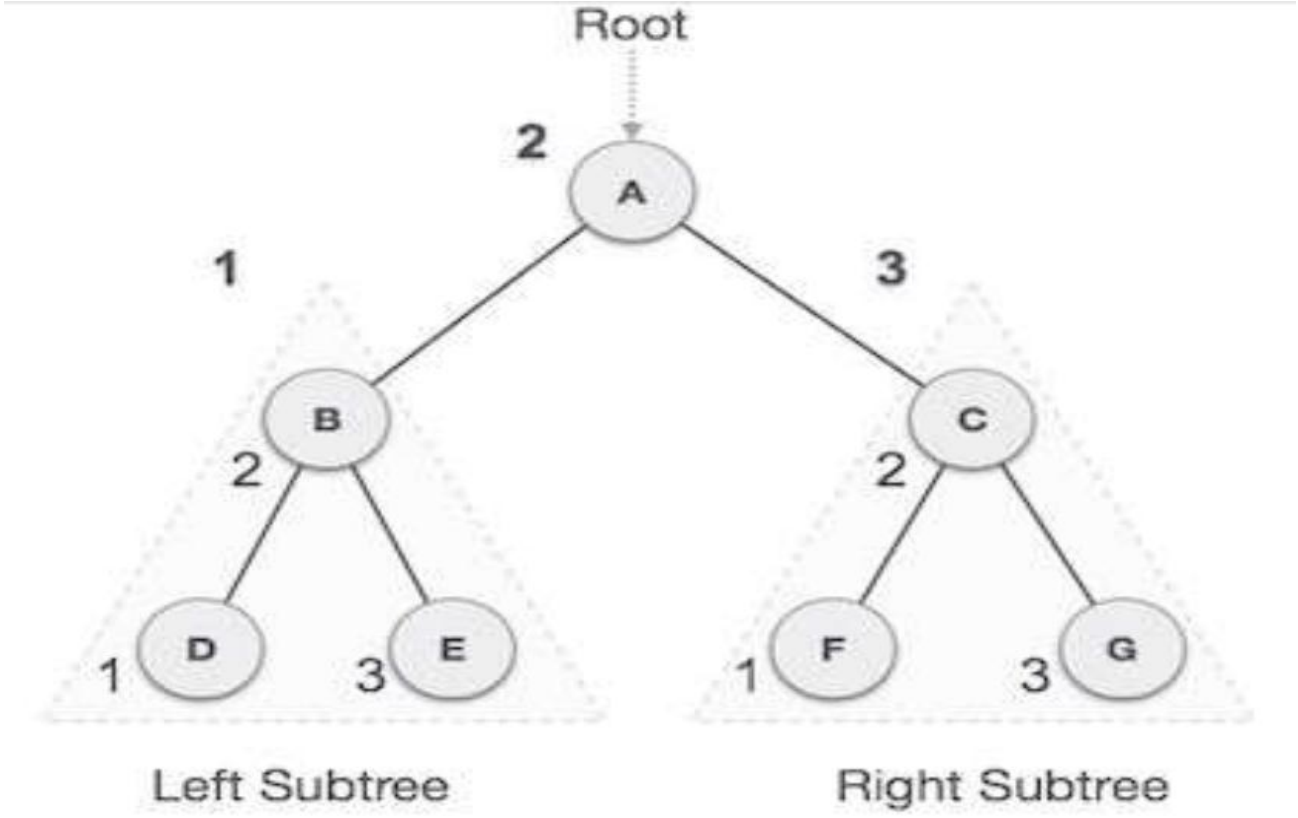
```
printLevelorder(tree)
1) Create an empty queue q
2) temp_node = root /*start from root*/
3) Loop while temp_node is not NULL
    a) print temp_node->data.
    b) Enqueue temp_node's children (first left then right children) to q
    c) Dequeue a node from q and assign it's value to temp_node
```

Level Order Traversal (BFS) in C++

Link: repl.it/repls/SplendidKnowledgeableFact

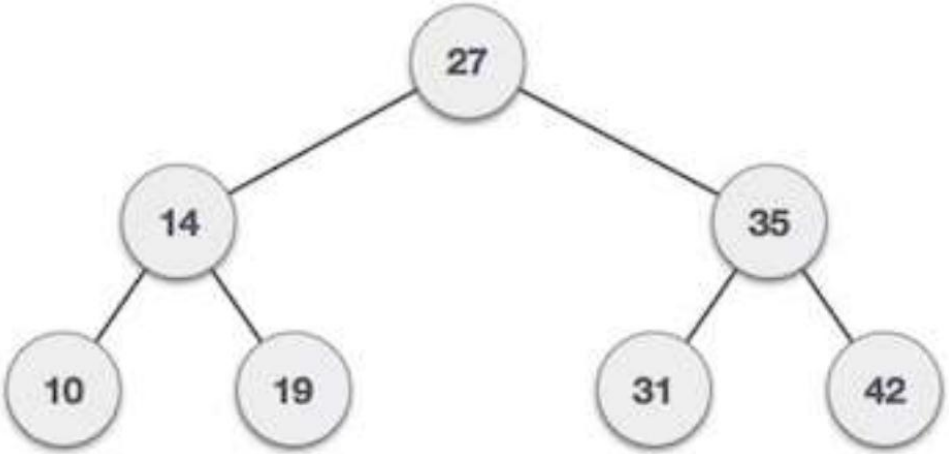
```
40  // Given a binary tree, print its nodes in level order
41  // using array for implementing queue
42  void print_level_order() { // O(n)
43      queue<node*> q;
44      node* curr = root;
45      while (curr != NULL) {
46          cout << curr->data << ' ';
47          // Enqueue left child
48          if (curr->left != NULL)
49              q.push(curr->left);
50          // Enqueue right child
51          if (curr->right != NULL)
52              q.push(curr->right);
53          // Dequeue node and make it temp
54          curr = q.front();
55          q.pop();
56      }
57  }
```

In-order Traversal (Left-Root-Right)



In-order Traversal Example

In-order Traversal : 10 14 19 27 31 35 42



In-order Traversal Algorithm

Algorithm

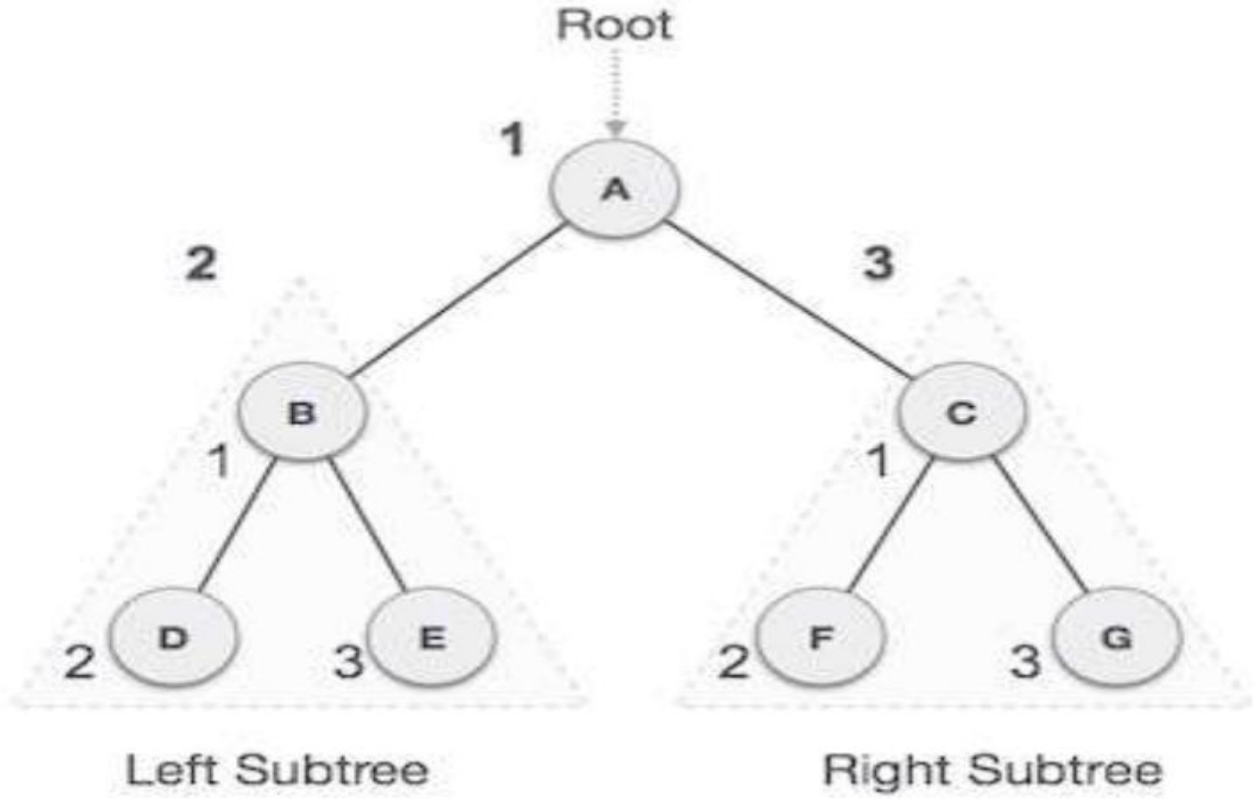
```
Until all nodes are traversed -
Step 1 - Recursively traverse left subtree.
Step 2 - Visit root node.
Step 3 - Recursively traverse right subtree.
```

In-order Traversal in C++

Link: repl.it/repls/SplendidKnowledgeableFact

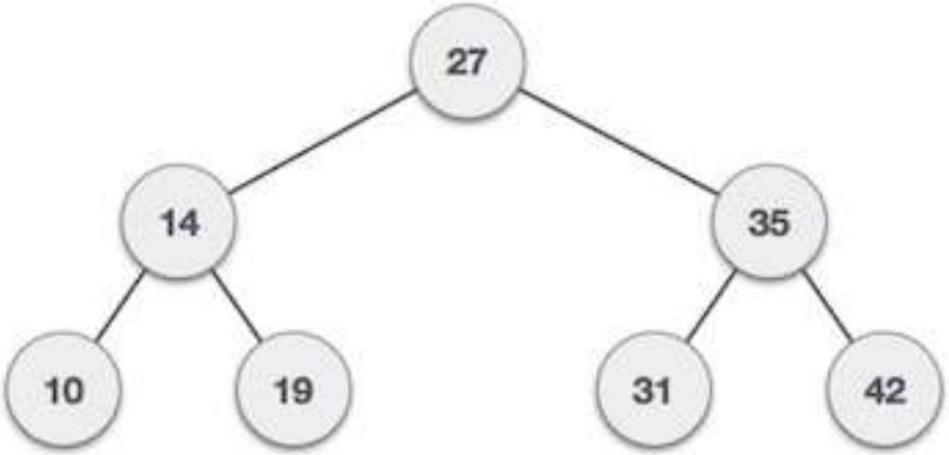
```
13  // A utility function to do inorder traversal of Binary Tree
14  void inOrder(node* curr) { // O(n)
15      if (curr == NULL)
16          return;
17      inOrder(curr->left);
18      cout << curr->data << ' ';
19      inOrder(curr->right);
20  }
```

Pre-order Traversal (Root-Left-Right)



Pre-order Traversal Example

Pre-order Traversal : 27 14 10 19 35 31 42



Pre-order Traversal Algorithm

Algorithm

Until all nodes are traversed –

Step 1 – Visit root node.

Step 2 – Recursively traverse left subtree.

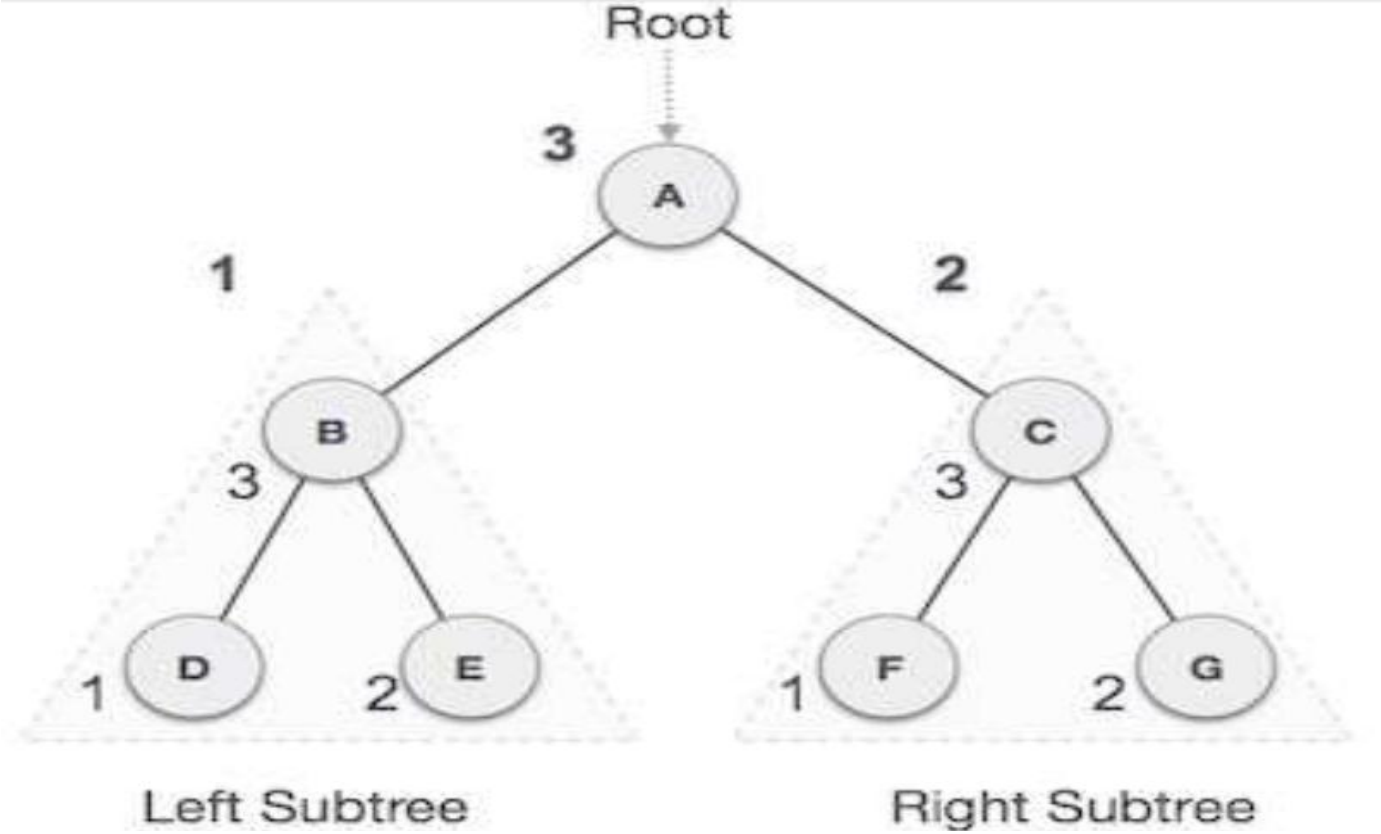
Step 3 – Recursively traverse right subtree.

Pre-order Traversal in C++

Link: repl.it/repls/SplendidKnowledgeableFact

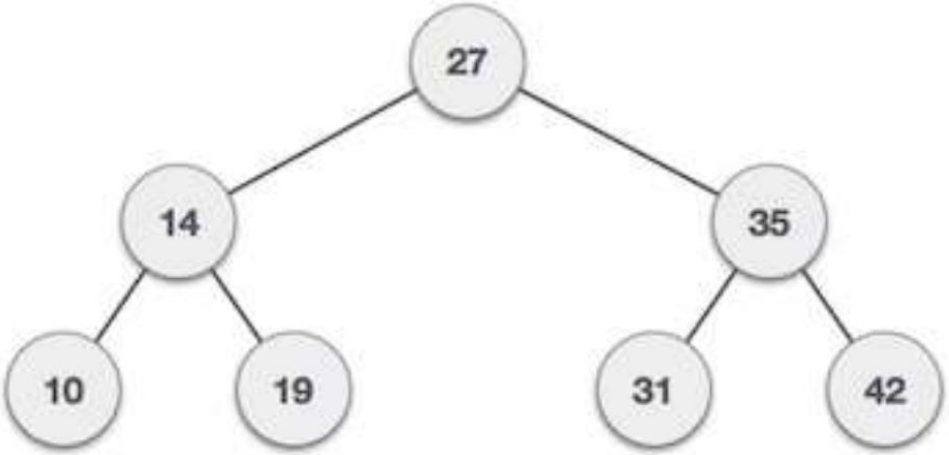
```
22  // A utility function to do inorder traversal of Binary Tree
23  void preOrder(node* curr) { // O(n)
24      if (curr == NULL)
25          return;
26      cout << curr->data << ' ';
27      preOrder(curr->left);
28      preOrder(curr->right);
29  }
```

Post-order Traversal (Left-Right-Root)



Post-order Traversal Example

Post-order Traversal : 10 19 14 31 42 35 27



Post-order Traversal Algorithm

Algorithm

Until all nodes are traversed –

Step 1 – Recursively traverse left subtree.

Step 2 – Recursively traverse right subtree.

Step 3 – Visit root node.

Post-order Traversal in C++

Link: repl.it/repls/SplendidKnowledgeableFact

```
31  // A utility function to do inorder traversal of Binary Tree
32  void postOrder(node* curr) { // O(n)
33      if (curr == NULL)
34          return;
35      postOrder(curr->left);
36      postOrder(curr->right);
37      cout << curr->data << ' ';
38  }
```

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- ✓ 3- Binary Tree Terms
- ✓ 4- Binary Tree Traversal
- ✓ 5- BFS vs. DFS for Binary Tree
- **6- Time Complexity and Space Complexity**
- 7- Search in Binary Tree
- 8- Delete Binary Tree

Time Complexity and Space Complexity

- Is there any difference in terms of Time Complexity?

All four traversals require O(n) time as they visit every node exactly once.

- Is there any difference in terms of Extra Space?

There is difference in terms of extra space required.

- Extra Space required for Level Order Traversal is O(w)

where w is maximum width of Binary Tree. In level order traversal, queue one by one stores nodes of different level.

- Extra Space required for Depth First Traversals is O(h)

where h is maximum height of Binary Tree. In Depth First Traversals, stack (or function call stack) stores all ancestors of a node.

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- ✓ 3- Binary Tree Terms
- ✓ 4- Binary Tree Traversal
- ✓ 5- BFS vs. DFS for Binary Tree
- ✓ 6- Time Complexity and Space Complexity
- 7- Search in Binary Tree
- 8- Delete Binary Tree

Search in Binary Tree in C++

Link: repl.it/repls/SplendidKnowledgeableFact

```
59  // Function that search if data in binary tree or not
60  bool search(node* curr, int data) { // O(n)
61      // base case we reach a null node so we won't find data
62      if(curr == NULL)
63          return false;
64      // return found if we find it at node curr
65      if(curr->data == data)
66          return true;
67      // search at subtree left and subtree right
68      bool left_search = search(curr->left, data);
69      bool right_search = search(curr->right, data);
70      return left_search || right_search;
71  }
```

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- ✓ 3- Binary Tree Terms
- ✓ 4- Binary Tree Traversal
- ✓ 5- BFS vs. DFS for Binary Tree
- ✓ 6- Time Complexity and Space Complexity
- ✓ 7- Search in Binary Tree
- **8- Delete Binary Tree**

Delete Binary Tree in C++

Link: repl.it/repls/SplendidKnowledgeableFact

```
73  //  This function traverses tree in post order to delete all nodes in tree
74  void delete_tree(node* curr) { // O(n)
75      if (curr == NULL)
76          return;
77      // first delete both subtrees
78      delete_tree(curr->left);
79      delete_tree(curr->right);
80      delete(curr);
81  }
```

Agenda

- ✓ 1- Binary Tree Definition
- ✓ 2- Binary Tree Representation
- ✓ 3- Binary Tree Terms
- ✓ 4- Binary Tree Traversal
- ✓ 5- BFS vs. DFS for Binary Tree
- ✓ 6- Time Complexity and Space Complexity
- ✓ 7- Search in Binary Tree
- ✓ 8- Delete Binary Tree


DO
MORE.

Practice

Practice

1- Level order traversal line by line

2- Print all root-to-leaf paths one per line in binary tree

3- Count number of nodes in binary tree

4- Count number of leaf nodes in binary tree

5- Find maximum node value in binary tree

6- Find minimum node value in binary tree

7- Calculate maximum depth in binary tree

8- Calculate minimum depth in binary tree

9- Calculate height of each node in binary tree

10- Calculate diameter of binary tree

Practice

11- Print all full nodes in a binary tree

12- Calculate maximum path sum in binary tree

13- Calculate minimum path sum in binary tree

14- Convert a tree into it's mirror tree

15- Sum of all nodes in a binary tree

16- Check for children sum property in binary tree

17- Check if a given binary tree is sum tree

18- Check if a given number equal root to leaf path sum

19- Evaluation of expression tree

20- Replace each node in binary tree with the sum of its inorder traversal

Practice

21- Find n-th node of inorder traversal of a binary tree

22- Find n-th node in postorder traversal of a binary tree

23- Find n-th node in preorder traversal of a binary tree

24- Construct a tree from inorder and level order traversals

25- Construct a tree from preorder and postorder traversals

26- Construct a tree from postorder and inorder traversals

27- Construct a tree from inorder and preorder traversals

28- Check if all leaves are at same level

29- Check if two nodes are cousins in a binary tree

30- Check if given preorder, inorder and postorder are of same tree

Practice

31- Check if leaf traversal of two binary trees is same

32- Check if a given binary tree is perfect or not

33- Check if a binary tree is a full binary tree or not by recursive

34- Check if a binary tree is a full binary tree or not by iterative

35- Check if binary tree is complete or not by recursive

36- Check if binary tree is complete or not by iterative

37- Check if a binary tree is subtree of another binary tree

38- Check if a binary tree has duplicate values

39- Check if a binary tree contains duplicate subtrees

40- Check if two trees are mirror of each other by iterative

Practice

41- Check if two trees are identical by recursive

42- Check if two trees are identical by iterative

43- Check for symmetric binary tree by iterative

44- Check if there is a root-to-leaf path with given sequence

45- Print middle level of perfect binary tree without finding height

46- Print cousins of a given node in binary tree

47- Print the longest leaf to leaf path in a binary tree

48- Print path from root to a given node in a binary tree

49- Print root-to-leaf paths without using recursion

50- Print the nodes at odd levels of a tree

Practice

51- Find lowest common ancestor LCA in a binary tree

52- Find distance between two nodes of a binary tree

53- Print common nodes on path from root or common ancestors

54- Maximum difference between node and its ancestor in binary tree

55- Print the path common to the two paths from the root to the two given nodes

56- Check if removing an edge can divide a binary tree in two halves

57- Sum of all left leaves in a given binary tree

58- Sum of all right leaves in a given binary tree

59- Sum of nodes on the longest path from root to leaf node

60- Find largest subtree sum in a binary tree

Practice

61- Find the maximum path sum between two leaves of a binary tree

62- Find the maximum sum leaf to root path in a binary tree

63- Maximum sum from a tree with adjacent levels not allowed

64- Print all k-sum paths in a binary tree

65- Subtree with given sum in a binary tree

66- Count subtrees that sum up to a given value x

67- Difference between sums of odd level and even level nodes of a binary tree

68- Find maximum level sum in binary tree

69- Sum of all leaf nodes of binary tree

70- Sum of leaf nodes at minimum level


Let's
START UP

Practice

1- Level order traversal line by line

2- Print all root-to-leaf paths one per line in binary tree

3- Count number of nodes in binary tree

4- Count number of leaf nodes in binary tree

5- Find maximum node value in binary tree

6- Find minimum node value in binary tree

7- Calculate maximum depth in binary tree

8- Calculate minimum depth in binary tree

9- Calculate height of each node in binary tree

10- Calculate diameter of binary tree

Level order traversal line by line

- Implement function which print nodes value of each level in separate line at Binary Tree
- Function Name: print level order lines
- Parameters: None
- Return: None

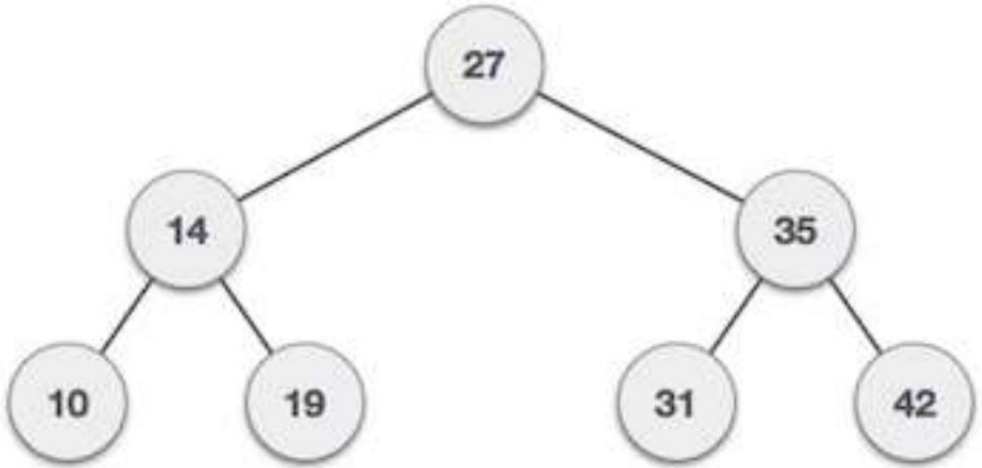
Level order traversal line by line

This Binary Tree should print:

27

14 35

10 19 31 42



Level order traversal line by line

Link: repl.it/repls/SplendidKnowledgeableFact

```
206  // Iterative method to do level order traversal line by line
207  void print_level_order_lines() { // O(n)
208      // check if tree is empty
209      if (root == NULL)
210          return;
211      // Create an empty queue for level order tarversal and Enqueue Root
212      queue<node*> q;
213      q.push(root);
214      // nodeCount (queue size) indicates number of nodes at current lelvel.
215      int nodeCount = q.size();
216      while (nodeCount > 0) {
217          // Dequeue all nodes of current level and Enqueue all nodes of next level
218          while (nodeCount > 0) {
219              node* curr_node = q.front();
220              cout << curr_node->data << ' ';
221              q.pop();
222              if (curr_node->left  != NULL) q.push(curr_node->left);
223              if (curr_node->right != NULL) q.push(curr_node->right);
224              nodeCount--;
225          }
226          cout << "\n";
227          nodeCount = q.size();
228      }
229  }
```

Print all root-to-leaf paths one per line in binary tree

- Implement function which print nodes values of each path in separate line at Binary Tree
- Function Name: print paths recursion
- Parameters: (curr, path, path size)

  curr: pointer to current node

  path: array of current path nodes

  path size: size of current path nodes
- Return: None

Print all root-to-leaf paths one per line in binary tree

Algorithm:

```
initialize: pathlen = 0, path[1000]
/*1000 is some max limit for paths, it can change*/

/*printPathsRecur traverses nodes of tree in preorder */
printPathsRecur(tree, path[], pathlen)
   1) If node is not NULL then
         a) push data to path array:
               path[pathlen] = node->data.
         b) increment pathlen
               pathlen++
   2) If node is a leaf node then print the path array.
   3) Else
        a) Call printPathsRecur for left subtree
                printPathsRecur(node->left, path, pathLen)
        b) Call printPathsRecur for right subtree.
               printPathsRecur(node->right, path, pathLen)
```

Print all root-to-leaf paths one per line in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
253   // Given a binary tree, print out all of its root-to-leaf paths, one per line.
254   void print_paths() {
255       int max_size = 1e5;
256       int path[max_size];
257       print_paths_recursion(root, path, 0);
258   }
```

Print all root-to-leaf paths one per line in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
231   // Recursive helper function given a node, and an array containing
232   // the path from the root node up to but not including this node,
233   // print out all the root-leaf paths
234   void print_paths_recursion(node* curr, int path[], int path_len) { // O(n * h)
235       if (curr == NULL)
236           return;
237       // append this node to the path array
238       path[path_len] = curr->data;
239       path_len++;
240       // it's a leaf, so print the path that led to here
241       if (curr->left == NULL && curr->right == NULL) {
242           // condition that prints out an array on a line
243           for (int i = 0; i < path_len; i++)
244               cout << path[i] << ' ';
245           cout << "\n";
246       }
247       else {
248           print_paths_recursion(curr->left, path, path_len);
249           print_paths_recursion(curr->right, path, path_len);
250       }
251   }
```

Count number of nodes in binary tree

- Implement function which count number of nodes in Binary Tree
- Function Name: size
- Parameters: (curr) pointer to current node
- Return: int number which mean number of nodes in Binary Tree

Count number of nodes in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
349   // Computes the number of nodes in a Binary Tree
350   int size(node* curr) { // O(n)
351       if (curr == NULL)
352           return 0;
353       int left_size = size(curr->left);
354       int right_size = size(curr->right);
355       return left_size + 1 + right_size;
356   }
```

Count number of leaf nodes in binary tree

- Implement function which count number of leaf nodes in Binary Tree
- Function Name: count leaf
- Parameters: (curr) pointer to current node
- Return: int number which mean number of leaf nodes in Binary Tree

Count number of leaf nodes in binary tree

Here is an algorithm to get the leaf node count.

```
getLeafCount(node)
1) If node is NULL then return 0.
2) Else If left and right child nodes are NULL return 1.
3) Else recursively calculate leaf count of the tree using below formula.
    Leaf count of a tree = Leaf count of left subtree +
                           Leaf count of right subtree
```

Count number of leaf nodes in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
146   // Function to get the count of leaf nodes in a Binary Tree
147   int count_leaf(node* curr) { // O(n)
148       if(curr == NULL)
149           return 0;
150       if(curr->left == NULL && curr->right==NULL)
151           return 1;
152       int left_leaf = count_leaf(curr->left);
153       int right_leaf = count_leaf(curr->right);
154       return left_leaf + right_leaf;
155   }
```

Find maximum node value in binary tree

- Implement function which get maximum node value in Binary Tree
- Function Name: find max value
- Parameters: (curr) pointer to current node
- Return: int number which mean maximum node value in Binary Tree

Find maximum node value in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
83  // Returns maximum value in a given Binary Tree
84  int find_max_value(node* curr) { // O(n)
85      if (curr == NULL)
86        return INT_MIN;
87      // Return maximum of 3 values:
88      // 1) Root's data
89      // 2) Max in Left Subtree
90      // 3) Max in right subtree
91      int result = curr->data;
92      int left_result = find_max_value(curr->left);
93      int right_result = find_max_value(curr->right);
94      return max(result, max(left_result, right_result));
95  }
```

Find minimum node value in binary tree

- Implement function which get minimum node value in Binary Tree
- Function Name: find min value
- Parameters: (curr) pointer to current node
- Return: int number which mean minimum node value in Binary Tree

Find minimum node value in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
97   // Returns minimum value in a given Binary Tree
98   int find_min_value(node* curr) { // O(n)
99       if (curr == NULL)
100        return INT_MAX;
101      // Return minimum of 3 values:
102      // 1) Root's data
103      // 2) Min in Left Subtree
104      // 3) Min in right subtree
105      int result = curr->data;
106      int left_result = find_min_value(curr->left);
107      int right_result = find_min_value(curr->right);
108      return min(result, min(left_result, right_result));
109  }
```

Calculate maximum depth in binary tree

- Implement function which calculate maximum depth of Binary Tree paths
- Function Name: max depth
- Parameters: (curr) pointer to current node
- Return: int number which mean maximum depth in Binary Tree

Calculate maximum depth in binary tree

Algorithm:

```
 maxDepth()
1. If tree is empty then return 0
2. Else
     (a) Get the max depth of left subtree recursively  i.e.,
          call maxDepth( tree->left-subtree)
     (a) Get the max depth of right subtree recursively  i.e.,
          call maxDepth( tree->right-subtree)
     (c) Get the max of max depths of left and right
          subtrees and add 1 to it for the current node.
         max_depth = max(max dept of left subtree,
                              max depth of right subtree)
                              + 1
     (d) Return max_depth
```

Calculate maximum depth in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
111   // Function get Max Depth
112   int max_depth(node* curr) { // O(n)
113       if (curr == NULL)
114           return 0;
115       // compute the depth of each subtree
116       int left_depth = max_depth(curr->left);
117       int right_depth = max_depth(curr->right);
118       // use the larger one
119       return 1 + max(left_depth, right_depth);
120   }
```

Calculate minimum depth in binary tree

- Implement function which calculate minimum depth of Binary Tree paths
- Function Name: min depth
- Parameters: (curr) pointer to current node
- Return: int number which mean minimum depth in Binary Tree

Calculate minimum depth in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
122   // Function get Min Depth
123   int min_depth(node* curr) { // O(n)
124       if (curr == NULL)
125           return 0;
126       // compute the depth of each subtree
127       int left_depth = min_depth(curr->left);
128       int right_depth = min_depth(curr->right);
129       // use the small one
130       return min(left_depth,right_depth) + 1;
131   }
```

Calculate height of each node in binary tree

- Implement function which calculate height of each node in Binary Tree
- Function Name: height
- Parameters: (curr) pointer to current node
- Return: int number which mean height of Binary Tree

Calculate height of each node in binary tree

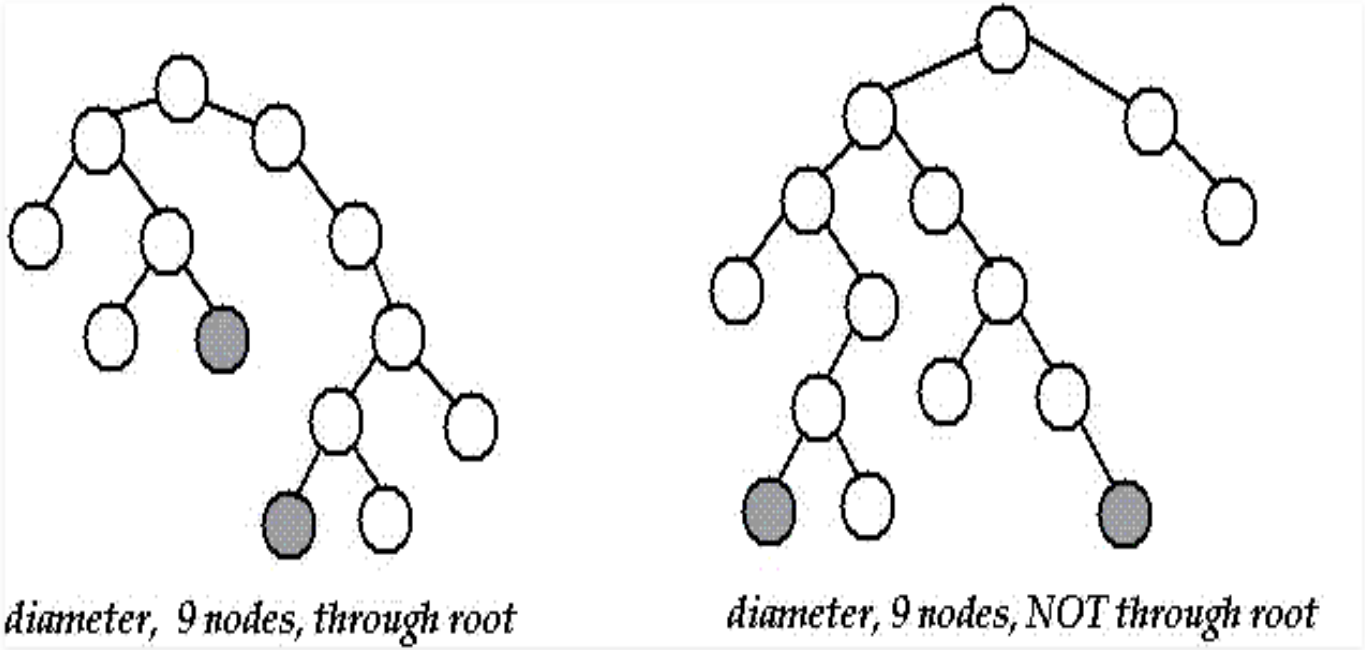
Link: repl.it/repls/SplendidKnowledgeableFact

```
318   // The function Compute the "height" of a tree.
319   // Height is the number f nodes along the longest path
320   // from the root node down to the farthest leaf node.
321   int height(node* curr) { // O(n)
322       // base case tree is empty
323       if(curr == NULL)
324           return 0;
325       // If tree is not empty then height = 1 + max of left height & right heights
326       int left_height = height(curr->left);
327       int right_height = height(curr->right);
328       return 1 + max(left_height, right_height);
329   }
```

Calculate diameter of binary tree

- Implement function which count diameter of Binary Tree
- Function Name: diameter
- Parameters: (curr) pointer to current node
- Return: int number which mean diameter of Binary Tree

Calculate diameter of binary tree



diameter, 9 nodes, through root

diameter, 9 nodes, NOT through root

Calculate diameter of binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
331   // Function to get diameter of a binary tree
332   int diameter(node* curr) { // O(n)
333       // base case where tree is empty
334       if (curr == NULL)
335           return 0;
336       // get the height of left and right sub-trees
337       int left_height = height(curr->left);
338       int right_height = height(curr->right);
339       /* Return max of following three
340       1) Diameter of left subtree
341       2) Diameter of right subtree
342       3) Height of left subtree + height of right subtree + 1
343       */
344       int left_diameter = diameter(curr->left);
345       int right_diameter = diameter(curr->right);
346       return max(left_height + right_height + 1, max(left_diameter, right_diameter));
347   }
```

Practice

1- Level order traversal line by line

2- Print all root-to-leaf paths one per line in binary tree

3- Count number of nodes in binary tree

4- Count number of leaf nodes in binary tree

5- Find maximum node value in binary tree

6- Find minimum node value in binary tree

7- Calculate maximum depth in binary tree

8- Calculate minimum depth in binary tree

9- Calculate height of each node in binary tree

10- Calculate diameter of binary tree


DO
MORE.

Practice

11- Print all full nodes in a binary tree

12- Calculate maximum path sum in binary tree

13- Calculate minimum path sum in binary tree

14- Convert a tree into it's mirror tree

15- Sum of all nodes in a binary tree

16- Check for children sum property in binary tree

17- Check if a given binary tree is sum tree

18- Check if a given number equal root to leaf path sum

19- Evaluation of expression tree

20- Replace each node in binary tree with the sum of its inorder traversal

Print all full nodes in a binary tree

- Implement function which print all nodes in Binary Tree that has two children
- Function Name: find full node
- Parameters: (curr) pointer to current node
- Return: None

Print all full nodes in a binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
372   // Traverses given tree in Inorder fashion and prints all nodes that
373   // have both children as non-empty.
374   void find_full_node(node* curr) { // O(n)
375       if (curr == NULL)
376           return;
377       find_full_node(curr->left);
378       if (curr->left != NULL && curr->right != NULL)
379           cout << curr->data << " ";
380       find_full_node(curr->right);
381   }
```

Calculate maximum path sum in binary tree

- Implement function which calculate the maximum sum in a path in Binary Tree
- Function Name: max path sum
- Parameters: (curr) pointer to current node
- Return: int, number which mean maximum sum in a path in Binary Tree

Calculate maximum path sum in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
271   // This Function get max sum of any path
272   int max_path_sum(node* curr) { // O(n)
273       // base case stop at child of leaf nodes
274       if(curr == NULL)
275           return 0;
276       // get max sum of left child
277       int left_path_sum = max_path_sum(curr->left);
278       // get max sum of right child
279       int right_path_sum = max_path_sum(curr->right);
280       // return max sum of it's children
281       return  curr->data + max(left_path_sum, right_path_sum);
282   }
```

Calculate minimum path sum in binary tree

- Implement function which calculate the minimum sum in a path in Binary Tree
- Function Name: min path sum
- Parameters: (curr) pointer to current node
- Return: int, number which mean minimum sum in a path in Binary Tree

Calculate minimum path sum in binary tree

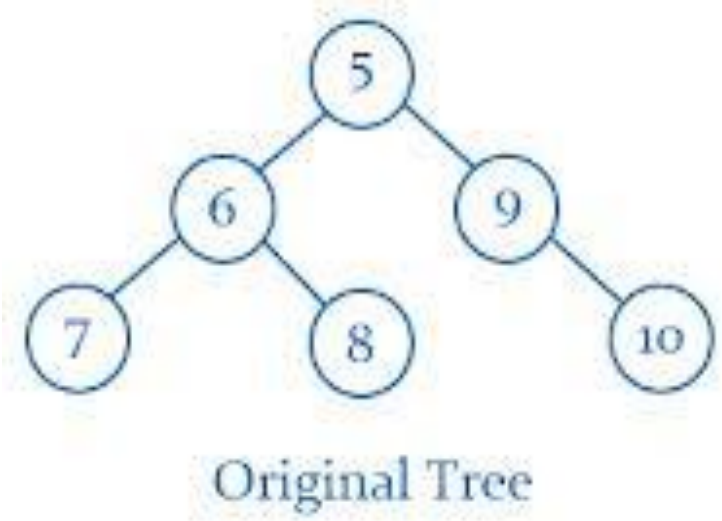
Link: repl.it/repls/SplendidKnowledgeableFact

```
284   // This Function get min sum of any path
285   int min_path_sum(node* curr) { // O(n)
286       // base case stop at child of leaf nodes
287       if(curr == NULL)
288           return 0;
289       // get min sum of left child
290       int left_path_sum = min_path_sum(curr->left);
291       // get min sum of right child
292       int right_path_sum = min_path_sum(curr->right);
293       // return min sum of it's children
294       return  curr->data + min(left_path_sum, right_path_sum);
295   }
```

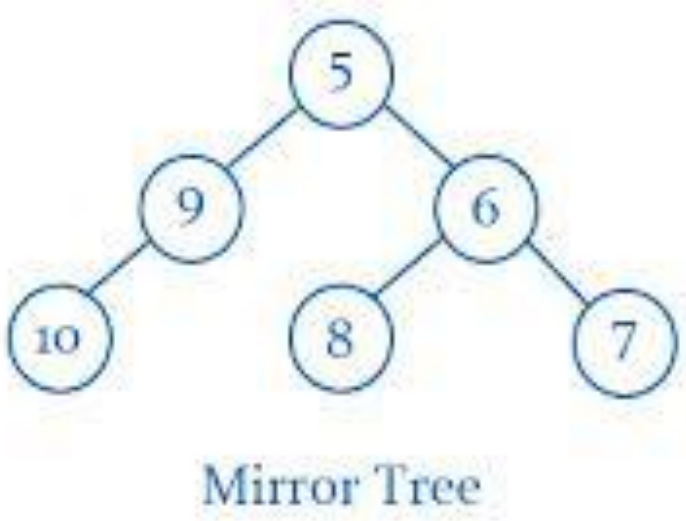
Convert a tree into it's mirror tree

- Implement function which reverse Binary Tree with its mirror
- Function Name: mirror
- Parameters: (curr) pointer to current node
- Return: None

Convert a tree into it's mirror tree



Original Tree



Mirror Tree

Convert a tree into it's mirror tree

Algorithm – Mirror(tree):

```
(1)  Call Mirror for left-subtree    i.e., Mirror(left-subtree)
(2)  Call Mirror for right-subtree  i.e., Mirror(right-subtree)
(3)  Swap left and right subtrees.
          temp = left-subtree
          left-subtree = right-subtree
          right-subtree = temp
```

Convert a tree into it's mirror tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
133   // This function mirror Binary Tree
134   void mirror(node* curr) { // O(n)
135       if (curr == NULL)
136           return;
137       // do the subtrees
138       mirror(curr->left);
139       mirror(curr->right);
140       // swap the pointers in this node
141       node* temp = curr->left;
142       curr->left = curr->right;
143       curr->right = temp;
144   }
```

Sum of all nodes in a binary tree

- Implement function which calculate sum of all nodes in Binary Tree
- Function Name: sum
- Parameters: (curr) pointer to current node
- Return: int, the sum of all nodes in Binary Tree

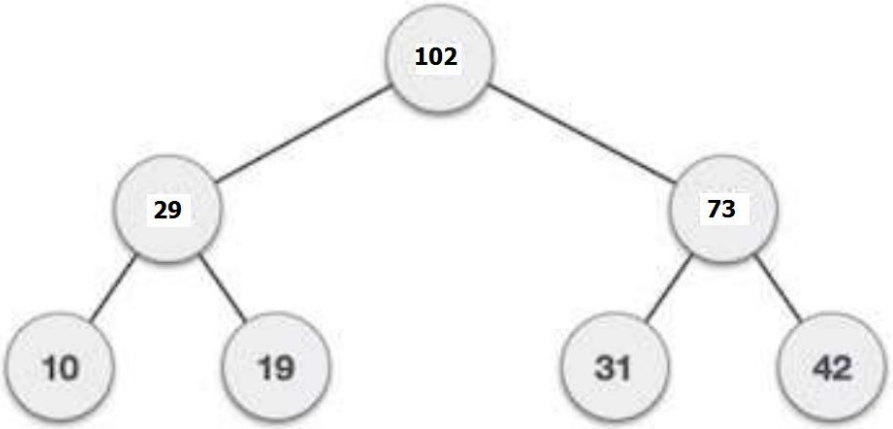
Sum of all nodes in a binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
180   // A utility function to get the sum of values in tree with root as root
181   int sum(node* curr) { // O(n)
182       if(curr == NULL)
183           return 0;
184       int left_sum = sum(curr->left);
185       int right_sum = sum(curr->right);
186       return  left_sum + curr->data + right_sum;
187   }
```

Check for children sum property in binary tree

- Implement function which determine if Binary Tree has sum property or not
- Function Name: is sum property
- Parameters: (curr) pointer to current node
- Return: boolean, true if Binary Tree has sum property, false otherwise



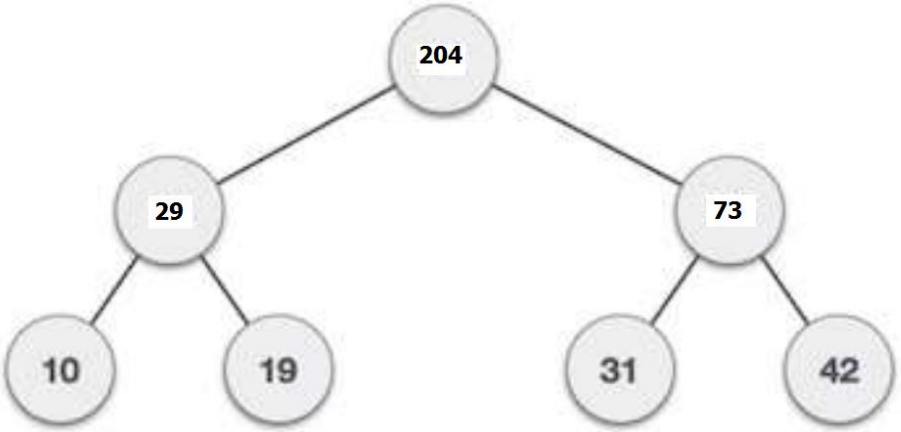
Check for children sum property in binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
157  // Function check if node equal sum of it's child
158  bool is_sum_property(node* curr) { // O(n)
159      // If node is NULL or it's a leaf node then return true
160      if(curr == NULL)
161          return true;
162      if(curr->left == NULL && curr->right == NULL)
163          return true;
164      // left_data is left child data and
165      // right_data is for right child data
166      int left_data = 0;
167      int right_data = 0;
168      // get data from left and right
169      if(curr->left != NULL)
170        left_data = curr->left->data;
171      if(curr->right != NULL)
172        right_data = curr->right->data;
173      // if the node and both of its children
174      // satisfy the property return true else false
175      return  curr->data == left_data + right_data &&
176              is_sum_property(curr->left) &&
177              is_sum_property(curr->right);
178  }
```

Check if a given binary tree is sum tree

- Implement function which determine if Binary Tree is sum tree or not
- Function Name: is sum tree
- Parameters: (curr) pointer to current node
- Return: boolean, true if Binary Tree is sum tree, false otherwise



Check if a given binary tree is sum tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
189   // Function check if sum property holds for the given node and both of its children
190   bool is_sum_tree(node* curr) { // O(n)
191       // If node is NULL or it's a leaf node then return true
192       if(curr == NULL)
193           return true;
194       if(curr->left == NULL && curr->right == NULL)
195           return true;
196       // Get sum of nodes in left and right subtrees
197       int left_sum = sum(curr->left);
198       int right_sum = sum(curr->right);
199       // if the node and both of its children
200       // satisfy the property return true else false
201       bool left_check = is_sum_tree(curr->left);
202       bool right_check = is_sum_tree(curr->right);
203       return left_check && curr->data == left_sum + right_sum && right_check;
204   }
```

Check if a given number equal root to leaf path sum

- Implement function which determine if given sum equal to sum of any path in Binary Tree or not
- Function Name: has path sum
- Parameters: (curr, sum)

  curr: pointer to current node

  sum: given sum which will be checked
- Return: boolean, true if given sum equal to sum of any path in Binary Tree, false otherwise

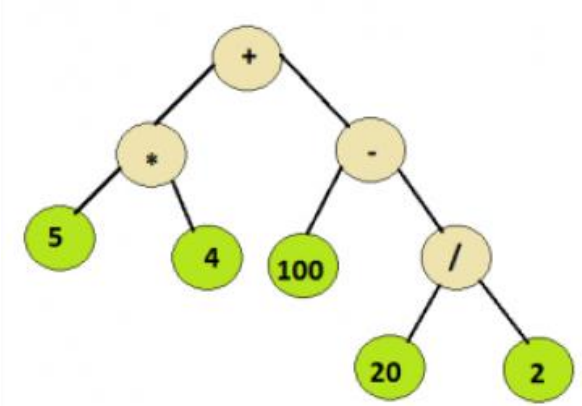
Check if a given number equal root to leaf path sum

Link: repl.it/repls/SplendidKnowledgeableFact

```
260  // function check if there exist summetion of path equal given sum
261  bool has_path_sum(node* curr, int sum) { // O(n)
262      // return true if we run out of tree and sum == 0
263      if (curr == NULL)
264          return sum == 0;
265      // check both subtrees
266      bool left_result  = has_path_sum(curr->left , sum - curr->data);
267      bool right_result = has_path_sum(curr->right, sum - curr->data);
268      return left_result || right_result;
269  }
```

Evaluation of expression tree

- Implement function which take a simple expression tree, consisting of basic binary operators i.e., + , - ,* and / and some integers, evaluate the expression tree.
- Function Name: evaluation expression
- Parameter: (curr) pointer to current node
- Return: int, the final value of expression evaluation



- Output: 110

Evaluation of expression tree

```
Algorithm :
Let t be the syntax tree
If  t is not null then
      If t.info is operand then
         Return  t.info
      Else
         A = solve(t.left)
         B = solve(t.right)
         return A operator B
         where operator is the info contained in t
```

Evaluation of expression tree

Link: repl.it/repls/SqueakyComplicatedPoint

```
13  // Utility function to return the integer value of a given string
14  int toInt(string s) {
15      int num = 0;
16      for (int i = 0; i < s.length(); i++)
17          num = num * 10 + int(s[i]-'0');
18      return num;
19  }
```

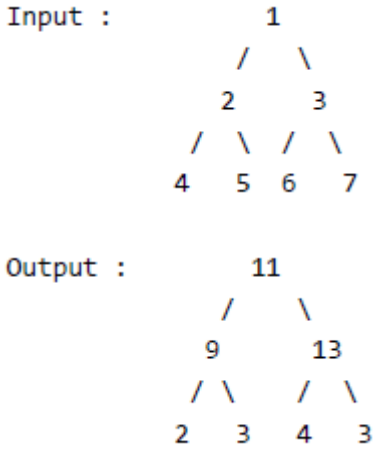
Evaluation of expression tree

Link: repl.it/repls/SqueakyComplicatedPoint

```
20  // This function receives a node of the syntax tree
21  // and recursively evaluates it
22  int eval(node* curr) {
23      if (curr == NULL)
24          return 0;
25      // leaf node i.e, an integer
26      if (curr->left == NULL && curr->right == NULL)
27          return toInt(curr->data);
28      // Evaluate left subtree
29      int l_val = eval(curr->left);
30      // Evaluate right subtree
31      int r_val = eval(curr->right);
32      // Check which operator to apply
33      if (curr->data=="+") return l_val+r_val;
34      if (curr->data=="-") return l_val-r_val;
35      if (curr->data=="*") return l_val*r_val;
36      if (curr->data=="/") return l_val/r_val;
37      return -1;
38  }
```

Replace each node in binary tree with the sum of its inorder traversal

- Implement function which replace each node with the sum of left & right subtree
- Function Name: replace node with sum
- Parameter: (curr) pointer to current node
- Return: int, the final value of each node



Replace each node in binary tree with the sum of its inorder traversal

Link: repl.it/repls/SplendidKnowledgeableFact

```
383   // function to replace each node with the sum of its
384   // inorder predecessor and successor
385   int replaceNodeWithSum(node* curr) { // O(n)
386       if (curr == NULL)
387           return 0;
388       int sumLeft = replaceNodeWithSum(curr->left);
389       int sumRight = replaceNodeWithSum(curr->right);
390       curr->data += sumLeft + sumRight;
391       return curr->data;
392   }
```

Practice

~~11- Print all full nodes in a binary tree~~

~~12- Calculate maximum path sum in binary tree~~

~~13- Calculate minimum path sum in binary tree~~

~~14- Convert a tree into it's mirror tree~~

~~15- Sum of all nodes in a binary tree~~

~~16- Check for children sum property in binary tree~~

~~17- Check if a given binary tree is sum tree~~

~~18- Check if a given number equal root to leaf path sum~~

~~19- Evaluation of expression tree~~

~~20- Replace each node in binary tree with the sum of its inorder traversal~~


DO
MORE.

Practice

21- Find n-th node of inorder traversal of a binary tree

22- Find n-th node in postorder traversal of a binary tree

23- Find n-th node in preorder traversal of a binary tree

24- Construct a tree from inorder and level order traversals

25- Construct a tree from preorder and postorder traversals

26- Construct a tree from postorder and inorder traversals

27- Construct a tree from inorder and preorder traversals

28- Check if all leaves are at same level

29- Check if two nodes are cousins in a binary tree

30- Check if given preorder, inorder and postorder are of same tree

Find n-th node of inorder traversal of a binary tree

Problem Link: geeksforgeeks.org/find-n-th-node-inorder-traversal

Find n-th node in postorder traversal of a binary tree

Problem Link: geeksforgeeks.org/find-n-th-node-in-postorder-traversal-of-a-binary-tree

Find n-th node in preorder traversal of a binary tree

Problem Link: geeksforgeeks.org/find-n-th-node-in-preorder-traversal-of-a-binary-tree

Construct a tree from inorder and level order traversals

Problem Link: geeksforgeeks.org/construct-tree-inorder-level-order-traversals

Construct a tree from preorder and postorder traversals

Problem Link: geeksforgeeks.org/full-and-complete-binary-tree-from-given-preorder-and-postorder-traversals

Construct a tree from postorder and inorder traversals

Problem Link: geeksforgeeks.org/construct-a-binary-tree-from-postorder-and-inorder

Construct a tree from inorder and preorder traversals

Problem Link: geeksforgeeks.org/construct-tree-from-given-inorder-and-preorder-traversal

Check if all leaves are at same level

Problem Link: geeksforgeeks.org/check-leaves-level

Check if two nodes are cousins in a binary tree

Problem Link: geeksforgeeks.org/check-two-nodes-cousins-binary-tree

Check if given preorder, inorder and postorder are same tree

Problem Link: geeksforgeeks.org/check-if-given-preorder-inorder-and-postorder-traversals-are-of-same-tree

Practice

~~21- Find n-th node of inorder traversal of a binary tree~~

~~22- Find n-th node in postorder traversal of a binary tree~~

~~23- Find n-th node in preorder traversal of a binary tree~~

~~24- Construct a tree from inorder and level order traversals~~

~~25- Construct a tree from preorder and postorder traversals~~

~~26- Construct a tree from postorder and inorder traversals~~

~~27- Construct a tree from inorder and preorder traversals~~

~~28- Check if all leaves are at same level~~

~~29- Check if two nodes are cousins in a binary tree~~

~~30- Check if given preorder, inorder and postorder are of same tree~~


DO
MORE.

Practice

31- Check if leaf traversal of two binary trees is same

32- Check if a given binary tree is perfect or not

33- Check if a binary tree is a full binary tree or not by recursive

34- Check if a binary tree is a full binary tree or not by iterative

35- Check if binary tree is complete or not by recursive

36- Check if binary tree is complete or not by iterative

37- Check if a binary tree is subtree of another binary tree

38- Check if a binary tree has duplicate values

39- Check if a binary tree contains duplicate subtrees

40- Check if two trees are mirror of each other by iterative

Check if leaf traversal of two binary trees is same

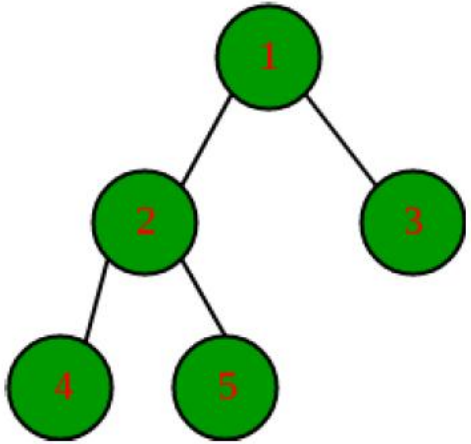
Problem Link: geeksforgeeks.org/check-if-leaf-traversal-of-two-binary-trees-is-same

Check if a given binary tree is perfect or not

Problem Link: geeksforgeeks.org/check-weather-given-binary-tree-perfect-not

Check if a binary tree is a full binary tree or not by recursive

- Implement function which check if binary tree is full or not
- Function Name: is full tree
- Parameter: (curr) pointer to current node
- Return: boolean, true if binary tree is full, false otherwise



A full binary tree is defined as a binary tree in which all nodes have either zero or two child nodes.

Check if a binary tree is a full binary tree or not by recursive

Link: repl.it/repls/SplendidKnowledgeableFact

```
394  // This function tests if a binary tree is a full binary tree.
395  bool isFullTree(node* curr) { // O(n)
396      if (curr == NULL)
397          return true;
398      if (curr->left == NULL && curr->right == NULL)
399          return true;
400      // If both left and right are not NULL, and left & right subtrees are full
401      if (curr->left != NULL && curr->right != NULL)
402          return isFullTree(curr->left) && isFullTree(curr->right);
403      // We reach here when none of the above if conditions work
404      return false;
405  }
```

Check if a binary tree is a full binary tree or not by iterative

Problem Link: geeksforgeeks.org/check-whether-binary-tree-full-binary-tree-not-iterative-approach

Check if binary tree is complete or not by recursive

Problem Link: geeksforgeeks.org/check-if-a-given-binary-tree-is-complete-tree-or-not

Check if binary tree is complete or not by iterative

Problem Link: geeksforgeeks.org/check-if-a-given-binary-tree-is-complete-tree-or-not

Check if a binary tree is subtree of another binary tree

Problem Link: geeksforgeeks.org/check-binary-tree-subtree-another-binary-tree-set-2

Check if a binary tree has duplicate values

Problem Link: geeksforgeeks.org/check-binary-tree-not-bst-duplicate-values

Check if a binary tree contains duplicate subtrees

Problem Link: geeksforgeeks.org/check-binary-tree-contains-duplicate-subtrees-size-2

Check if two trees are mirror of each other by iterative

Problem Link: geeksforgeeks.org/check-if-two-trees-are-mirror

Problem Link: geeksforgeeks.org/iterative-method-check-two-trees-mirror

Practice

~~31- Check if leaf traversal of two binary trees is same~~

~~32- Check if a given binary tree is perfect or not~~

~~33- Check if a binary tree is a full binary tree or not by recursive~~

~~34- Check if a binary tree is a full binary tree or not by iterative~~

~~35- Check if binary tree is complete or not by recursive~~

~~36- Check if binary tree is complete or not by iterative~~

~~37- Check if a binary tree is subtree of another binary tree~~

~~38- Check if a binary tree has duplicate values~~

~~39- Check if a binary tree contains duplicate subtrees~~

~~40- Check if two trees are mirror of each other by iterative~~


DO
MORE.

Practice

41- Check if two trees are identical by recursive

42- Check if two trees are identical by iterative

43- Check for symmetric binary tree by iterative

44- Check if there is a root-to-leaf path with given sequence

45- Print middle level of perfect binary tree without finding height

46- Print cousins of a given node in binary tree

47- Print the longest leaf to leaf path in a binary tree

48- Print path from root to a given node in a binary tree

49- Print root-to-leaf paths without using recursion

50- Print the nodes at odd levels of a tree

Check if two trees are identical by recursive

Problem Link: geeksforgeeks.org/write-c-code-to-determine-if-two-trees-are-identical

Check if two trees are identical by iterative

Problem Link: geeksforgeeks.org/iterative-function-check-two-trees-identical

Check for symmetric binary tree by iterative

Problem Link: geeksforgeeks.org/check-symmetric-binary-tree-iterative-approach

Check if there is a root-to-leaf path with given sequence

Problem Link: geeksforgeeks.org/check-root-leaf-path-given-sequence

Print middle level of perfect binary tree without finding height

Problem Link: geeksforgeeks.org/print-middle-level-perfect-binary-tree-without-finding-height

Print cousins of a given node in binary tree

Problem Link: geeksforgeeks.org/print-cousins-of-a-given-node-in-binary-tree

Print the longest leaf to leaf path in a binary tree

Problem Link: geeksforgeeks.org/print-longest-leaf-leaf-path-binary-tree

Print path from root to a given node in a binary tree

Problem Link: geeksforgeeks.org/print-path-root-given-node-binary-tree

Print root-to-leaf paths without using recursion

Problem Link: geeksforgeeks.org/print-root-leaf-path-without-using-recursion

Print the nodes at odd levels of a tree

Problem Link: geeksforgeeks.org/print-nodes-odd-levels-tree

Practice

~~41- Check if two trees are identical by recursive~~

~~42- Check if two trees are identical by iterative~~

~~43- Check for symmetric binary tree by iterative~~

~~44- Check if there is a root-to-leaf path with given sequence~~

~~45- Print middle level of perfect binary tree without finding height~~

~~46- Print cousins of a given node in binary tree~~

~~47- Print the longest leaf to leaf path in a binary tree~~

~~48- Print path from root to a given node in a binary tree~~

~~49- Print root-to-leaf paths without using recursion~~

~~50- Print the nodes at odd levels of a tree~~


DO
MORE.

Practice

51- Find lowest common ancestor LCA in a binary tree

52- Find distance between two nodes of a binary tree

53- Print common nodes on path from root or common ancestors

54- Maximum difference between node and its ancestor in binary tree

55- Print the path common to the two paths from the root to the two given nodes

56- Check if removing an edge can divide a binary tree in two halves

57- Sum of all left leaves in a given binary tree

58- Sum of all right leaves in a given binary tree

59- Sum of nodes on the longest path from root to leaf node

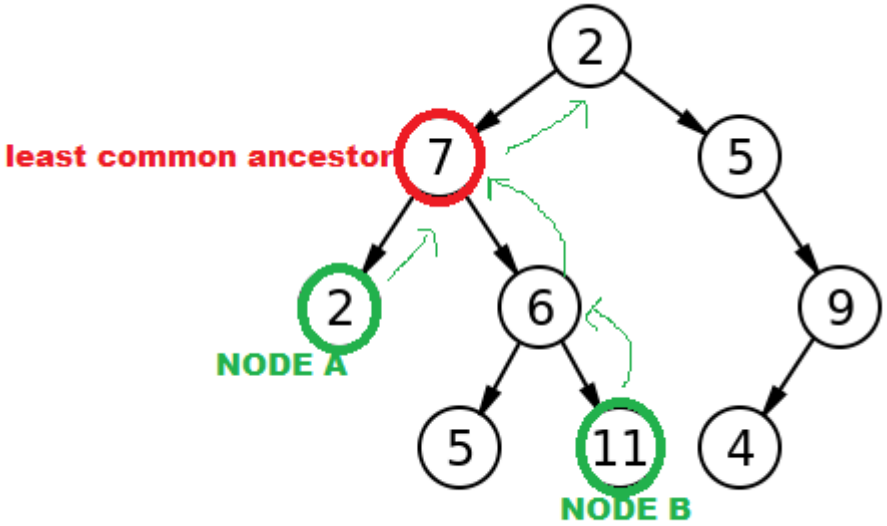
60- Find largest subtree sum in a binary tree

Find lowest common ancestor LCA in a binary tree

- Let T be a rooted tree. The lowest common ancestor between two nodes n1 and n2 is defined as the lowest node in T that has both n1 and n2 as descendants (where we allow a node to be a descendant of itself).

- The LCA of n1 and n2 in T is the shared ancestor of n1 and n2 that is located farthest from the root. Computation of lowest common ancestors may be useful, for instance, as part of a procedure for determining the distance between pairs of nodes in a tree: the distance from n1 to n2 can be computed as the distance from the root to n1, plus the distance from the root to n2, minus twice the distance from the root to their lowest common ancestor.

Find lowest common ancestor LCA in a binary tree



Find lowest common ancestor LCA in a binary tree

Link: repl.it/repls/SplendidKnowledgeableFact

```
297   // This function returns pointer to LCA of two given values n1 and n2.
298   // This function assumes that n1 and n2 are present in Binary Tree
299   node* find_LCA(node* curr, int n1, int n2) { // O(n)
300       if (curr == NULL)
301           return NULL;
302       // If either n1 or n2 matches with root's key, report the presence by returning root
303       if (curr->data == n1 || curr->data == n2)
304           return curr;
305       // Look for keys in left and right subtrees
306       node* left_Lca  = find_LCA(curr->left, n1, n2);
307       node* right_Lca = find_LCA(curr->right, n1, n2);
308       // If both of the above calls return Non-NULL, then one key is present
309       // in once subtree and other is present in other, So this node is the LCA
310       if (left_Lca != NULL && right_Lca != NULL)
311           return curr;
312       // Otherwise check if left subtree or right subtree is LCA
313       if(left_Lca != NULL)
314           return left_Lca;
315       return right_Lca;
316   }
```

Find distance between two nodes of a binary tree

Problem Link: geeksforgeeks.org/find-distance-between-two-nodes-of-a-binary-tree

Print common nodes on path from root or common ancestors

Problem Link: geeksforgeeks.org/print-common-nodes-path-root-common-ancestors

Maximum difference between node and its ancestor in binary tree

Problem Link: geeksforgeeks.org/maximum-difference-between-node-and-its-ancestor-in-binary-tree

Print the path common to the two paths from the root to the two given nodes

Problem Link: geeksforgeeks.org/print-path-common-two-paths-root-two-given-nodes

Check if removing an edge can divide a binary tree in two halves

Problem Link: geeksforgeeks.org/check-if-removing-an-edge-can-divide-a-binary-tree-in-two-halves

Sum of all left leaves in a given binary tree

Problem Link: geeksforgeeks.org/find-sum-nodes-given-perfect-binary-tree

Sum of all right leaves in a given binary tree

Problem Link: geeksforgeeks.org/find-sum-right-leaves-given-binary-tree

Sum of nodes on the longest path from root to leaf node

Problem Link: geeksforgeeks.org/sum-nodes-longest-path-root-leaf-node

Find largest subtree sum in a binary tree

Problem Link: geeksforgeeks.org/find-largest-subtree-sum-tree

Practice

~~51- Find lowest common ancestor LCA in a binary tree~~

~~52- Find distance between two nodes of a binary tree~~

~~53- Print common nodes on path from root or common ancestors~~

~~54- Maximum difference between node and its ancestor in binary tree~~

~~55- Print the path common to the two paths from the root to the two given nodes~~

~~56- Check if removing an edge can divide a binary tree in two halves~~

~~57- Sum of all left leaves in a given binary tree~~

~~58- Sum of all right leaves in a given binary tree~~

~~59- Sum of nodes on the longest path from root to leaf node~~

~~60- Find largest subtree sum in a binary tree~~


DO
MORE.

Practice

61- Find the maximum path sum between two leaves of a binary tree

62- Find the maximum sum leaf to root path in a binary tree

63- Maximum sum from a tree with adjacent levels not allowed

64- Print all k-sum paths in a binary tree

65- Subtree with given sum in a binary tree

66- Count subtrees that sum up to a given value x

67- Difference between sums of odd level and even level nodes of a binary tree

68- Find maximum level sum in binary tree

69- Sum of all leaf nodes of binary tree

70- Sum of leaf nodes at minimum level

Find the maximum path sum between two leaves of a binary tree

Problem Link: geeksforgeeks.org/find-maximum-path-sum-two-leaves-binary-tree

Find the maximum sum leaf to root path in a binary tree

Problem Link: geeksforgeeks.org/find-the-maximum-sum-path-in-a-binary-tree

Maximum sum from a tree with adjacent levels not allowed

Problem Link: geeksforgeeks.org/maximum-sum-tree-adjacent-levels-not-allowed

Print all k-sum paths in a binary tree

Problem Link: geeksforgeeks.org/print-k-sum-paths-binary-tree

Subtree with given sum in a binary tree

Problem Link: geeksforgeeks.org/subtree-given-sum-binary-tree

Count subtrees that sum up to a given value x

Problem Link: geeksforgeeks.org/count-subtress-sum-given-value-x

Difference between sums of odd level and even level nodes of a binary tree

Problem Link: geeksforgeeks.org/difference-between-sums-of-odd-and-even-levels

Find maximum level sum in binary tree

Problem Link: geeksforgeeks.org/find-level-maximum-sum-binary-tree

Sum of all leaf nodes of binary tree

Problem Link: geeksforgeeks.org/sum-leaf-nodes-binary-tree

Sum of leaf nodes at minimum level

Problem Link: geeksforgeeks.org/sum-leaf-nodes-minimum-level

Practice

~~61- Find the maximum path sum between two leaves of a binary tree~~

~~62- Find the maximum sum leaf to root path in a binary tree~~

~~63- Maximum sum from a tree with adjacent levels not allowed~~

~~64- Print all k-sum paths in a binary tree~~

~~65- Subtree with given sum in a binary tree~~

~~66- Count subtrees that sum up to a given value x~~

~~67- Difference between sums of odd level and even level nodes of a binary tree~~

~~68- Find maximum level sum in binary tree~~

~~69- Sum of all leaf nodes of binary tree~~

~~70- Sum of leaf nodes at minimum level~~


DO
MORE.

Assignment

References

[01]	Online Course YouTube Playlists	https://bit.ly/2Pq88rN
[02]	Introduction to Algorithms Thomas H. Cormen	https://bit.ly/20NhuSn
[03]	Competitive Programming 3 Steven Halim	https://nus.edu/2z40vyK
[04]	Fundamental of Algorithmics Gilles Brassard and Paul Bartley	https://bit.ly/2QuvwbM
[05]	Analysis of Algorithms An Active Learning Approach	http://bit.ly/2EgCCYX
[06]	Data Structures and Algorithms Annotated Reference	http://bit.ly/2c37XEv
[07]	Competitive Programmer's Handbook	https://bit.ly/2APAbeG
[08]	GeeksforGeeks	https://geeksforgeeks.org
[09]	Codeforces Online Judge	http://codeforces.com
[10]	HackerEarth Online Judge	https://hackerearth.com
[11]	TopCoder Online Judge	https://topcoder.com

Questions ?