**Binary/Binary Search Tree**

Note : All complex questions on trees require you to traverse it in some way. So command on tree traversal will go a long way in solving those problems.

1. Creating a binary/binary search tree form array/linked list
2. Given preorder and postorder traversal create BT. DO for preorder, inorder traversal as well. Use recursion.
3. Traversal: preorder, inorder and postorder recursively, iteratively and using **Morris Algorithm.** Morris uses the idea of threaded binary tree. Post order using Morris is complex…is done using dummy node.
4. Level order traversal, reverse level order traversal, spiral level order traversal (using two stacks or using deque), vertical level order traversal.
5. Printing left view, right view, top view, bottom view
6. Find min-max in BST.
7. Check if given tree is a binary search tree. (Best approach – use range check at every node to report failure case and reduce search space)
8. Find **largest BST** in a BT(use min-max approach to check if a subtree is bst).
9. Find all paths from root to leaf with a given sum X.(use of DFS/preorder or backtracking) Now print the leaf to root path with given sum. Print leaf to leaf path with given sum. Print maximum sum path(start and end may be any node). <http://www.geeksforgeeks.org/find-the-maximum-sum-path-in-a-binary-tree/>

<http://www.geeksforgeeks.org/find-maximum-path-sum-in-a-binary-tree/>

Find maximum level sum(level order traversal)

1. Find height and **diameter** of the tree. Diameter is longest path between any two nodes in the tree which may/mayn’t include the root node).
2. Find **LCA**(Lowest Common Ancestor) of two nodes in a BT and BST.(Recursive , path printing approach, RMQ approach)
3. **Find distance between two given nodes in bt.**

**Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2\*Dist(root, lca)**

1. Check if two nodes in a BT are **cousins**, i.e. they must be at the same level and their immediate parents must be different.
2. **Vertical Level Order Traversal** :

Approach 1 :Keep HashMap<Integer, List<Node>> a map of horizontal distance form root and list of nodes at same horizontal distance hd. Do preorder traversal and decrement hd while moving left and incrementing hd moving right. This approach prints nodes at same horizontal dist together but may not in the order as looked from the top,i.e. nodes at a particular vertical line may not be in exact order as in the tree.

Approach 2: This approach prints nodes at same vertical line in exact order as in the tree. Keep the same hashmap as above but do level order traversal instead of preorder. In the queue of level order, store OueueItem(Node, hd). This way nodes that are higher in the vertical line will be printed before lower nodes. While dequeing, put left child as QueuItem(Node.left, hd-1) and right child as QueueItem(Node.right, hd+1)

1. Top view of tree: Do vertical level order traversal using horizontal level order approach but don’t save the entire hashmap, just keep HashSet(integer) to keep track of vsisted nodes. Whille dequeing check if this node exists in the hashset. If it does this means the first node of this vertical line has been visited so we will ignore other nodes of this line. If not visited, add to the hashset and continue.
2. Bottom view of tree : Differs from top view in that we need to store last nodes of every vertical line. We will still do vertical level order sing horizontal level order approach but instead of HashSet we wil use TreeMap sorted on hd to store required nodes. The level order will vist nodes of vertical lines from top to high and wil replace the previous nodes in treemap so that at the end we have in treemap last nodes of all vertical lines. Now, the treemap will have bottom nodes from left to right due to its sorted nature. Note : if we’re allowed to modify the tree node structure we can keep hd in the tree to do away with defining QueItem(Node, hd) for level order traversal.
3. Left view : Can be done by simple preorder traversal while keeping the last level seen. If the current level > last level seen visit the node and print(also update level seen) else visit left and right subtrees. This version uses stack of recursion. We can also use queue and do level order traversal.
4. Right view : Use same ideas as in left view except that visit right subtree before visiting left subtree.
5. **Find maximum path sum in a bt, the path may begin and end at any** **node** : Uses the same idea as used in diameter with a difference that recursive call will return max path including root instead of max path for the subtree rooted at the current root. The path can be soley in left subtree(case1) or solely in right subtree(case2) or may pass through the root nodes(case3) or only root(case4). Do postorder traversal b/c for comparision we need in advance answers for subtrees and then find max of four cases. Note : a trick is that every recursive call need to return max path some including at most one child and root so that in parent call we can extend the optimal path otherwise path will get broken between recursive calls if we simply returned max of all four cases in recursion. The max of four cases shall be updated in a global variable ad we move up the tree in post order visiting. <http://www.geeksforgeeks.org/find-maximum-path-sum-in-a-binary-tree/>
6. Recursion practice – Given a leaf node and root node, print leaf to root path. Use recursion while returning Boolean in postorder traversal if leaf found. Note : same trick, every recursive call shall return max path that pass with left/right subtree and root node to be used in parent call so that path doesn’t get broken. Also globally update optimal path sum in every recursive call. <http://www.geeksforgeeks.org/find-the-maximum-sum-path-in-a-binary-tree/>
7. **Print nodes at K distance from root, from any given node, k distance from all leaf nodes**. <http://www.geeksforgeeks.org/print-nodes-at-k-distance-from-root/>

<http://www.geeksforgeeks.org/print-nodes-distance-k-leaf-node/>

<http://www.geeksforgeeks.org/print-nodes-distance-k-given-node-binary-tree/>

http://algorithms.tutorialhorizon.com/print-all-the-nodes-which-are-x-distance-from-the-given-node/

1. Mirror a BT(use postorder). Find if a BT is mirror of another BT. Find if a BT can be folded.
2. **Find max no of distinct nodes in a root to leaf pat**h. Follow up – in any binary tree path- <http://stackoverflow.com/questions/39940607/max-number-of-distinct-nodes-in-a-binary-tree-path>
3. Given a leaf node, print all its ancestors. Use of backtracking, make recursive call return Boolean if leaf found and pass list as parameter in call to store path and backtrack
4. Find the sum of leaf nodes at minimum depth of a tree. Return -1 if root is NULL. Find min depth and then find sum of leaves at the level. Before recursing, handle failure cases first.
5. Sum of nodes at a given depth. Recursive , bfs/level order traversal.

---------------------------------------------------**Nary Tree Questions**------------------------

**1. Mirror an n-ary tree. Find maximum in n-ry tree**

**2. Find maximum non-adjacent sum in n-ry tree.(https://stackoverflow.com/questions/28871860/algorithm-to-find-the-maximum-non-adjacent-sum-in-n-ary-tree)**

**2. Count all traversals of an n-ary tree.(product of factorial(number of children of each node));**

**3. Check if a given n-ry tree is a sum tree(using postorder and level order both). If not how to make it a sum tree.**

**4. Find the maximum avg score of an n-ry tree. Avg score = (sum of child nodes at a node)/no. of nodes.**

**4. Serialize and de-serialize and n-ry tree**

**5. Find LCA of an n-ry tree(http://www.geeksforgeeks.org/lca-n-ary-tree-constant-query-o1/) 4 methods**

**6. Diameter of an n-ry tree.**

**7. Longest path in an undirected tree**