Binary Tree

Wednesday, 25 January 2023 12:11 AM

Binary Tree
https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/
https://www.upgrad.com/blog/5-types-of-binary-tree/
https://www.interviewcake.com/concept/python/heap
    Min-heap, max-heap
Binary Search Tree(BST)
    https://www.geeksforgeeks.org/binary-search-tree-data-structure/
Inorder traversal/Preorder traversal
https://www.geeksforgeeks.org/binary-search-tree-data-structure/
Balanced Binary Search Trees(BST)

- AVL Tree -> https://www.geeksforgeeks.org/insertion-in-an-avl-tree/
    - https://www.programiz.com/dsa/avl-tree
- Black Red Tree -> https://www.geeksforgeeks.org/introduction-to-red-black-tree/
- https://www.geeksforgeeks.org/self-balancing-binary-search-trees-comparisons/
- https://www.geeksforgeeks.org/red-black-tree-vs-avl-tree/

Binary Tree

- https://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/
- https://www.geeksforgeeks.org/binary-tree-to-binary-search-tree-conversion/
- https://www.geeksforgeeks.org/maximum-element-two-nodes-bst/
- https://www.geeksforgeeks.org/print-nodes-distance-k-given-node-binary-tree/
- https://www.geeksforgeeks.org/sorted-array-to-balanced-bst/
- https://www.geeksforgeeks.org/find-if-there-is-a-triplet-in-bst-that-adds-to-0/
    - BST to DLL is O(n)
    - find triplet in DLL is O(n^2).
- https://www.geeksforgeeks.org/find-a-pair-with-given-sum-in-bst/
    - Video -> 2-sum BST | Find a pair with given sum in a BST | 2 Methods
- **Find the maximum path sum between two leaf nodes of a binary tree**
    - https://www.geeksforgeeks.org/find-maximum-path-sum-in-a-binary-tree/
- **Connect nodes at same level**
- Convert bst to dll(flatten)
    - https://www.geeksforgeeks.org/convert-binary-tree-to-doubly-linked-list-by-keeping-
        - https://www.techiedelight.com/place-convert-given-binary-tree-to-doubly-linke
    - https://www.geeksforgeeks.org/convert-binary-tree-to-doubly-linked-list-by-fixing-lef

-track-of-visited-node/
ed-list/
ft-and-right-pointers/

        - Use **in-order traversal**
        - Assign pointers using previous nodes concept
        - Take left most traversal and use bottom-up approach to assign left pointers only to up.
            - Assign left pointers using previous concept from root to left nodes from b
        - Take right most traversal and use bottom-up approach to go left side directions assign right pointers from left nodes to root nodes.
            - assignment only for left nodes to root nodes since right nodes are already right.
        - Using **stack data structure**
- Flatten bst to linked list
    - https://www.geeksforgeeks.org/flatten-bst-to-sorted-list-increasing-order/
- **Lowest Common Ancestor in a Binary Search Tree.**
    - 2 types
        - BST
            - https://www.geeksforgeeks.org/lowest-common-ancestor-in-a-binary-sea
            - Recursively iterate both the nodes till node> n1 and node<n2
            - Otherwise iterate it if it is node<n1 & node<n2, then node.right, els
            - Node>n1 & node>n2,then node.left
        - Binary Tree
        - https://www.geeksforgeeks.org/lowest-common-ancestor-binary-tree-set-1/
            - Capture the path of both nodes in array
            - Iterate both the arrays at same time, the differing point is the LCA
            - Another approach:
                - Recursively go the node left and right of both the nodes,
                - The moment, it identifies both the nodes, that is the LCA
                - Base condition ..is both left and right nodes should not be null, if bo the LCA

- **Burn entire tree from target node**
- Print all nodes at distance K from given node: Iterative Approach
    - Iterative approach using hashmap, queues, hashset
        - Hashset -> for identifying covered nodes already
        - Hashmap -> to maintain nodes and its parent to go upwards
        - https://www.geeksforgeeks.org/print-all-nodes-at-distance-k-from-given-node-
    - Recursive - backtracking approach -> https://www.geeksforgeeks.org/print-nodes- binary-tree/
- https://www.geeksforgeeks.org/sorted-array-to-balanced-bst/

y for all nodes from bottom

ottom
from top to bottom to

assigned right from root to

arch-tree/

e

th are not null, then that is

iterative-approach/
distance-k-given-node-

- https://www.geeksforgeeks.org/bfs-vs-dfs-binary-tree/?ref=lbp
- https://www.geeksforgeeks.org/level-order-tree-traversal/?ref=lbp
- https://www.geeksforgeeks.org/count-bst-nodes-that-are-in-a-given-range/?ref=rp
- https://takeuforward.org/data-structure/kth-largest-smallest-element-in-binary-search-tree
    - Travel inorders in normal and reverse order based on kth largest or smallest
- BST iterator
    - https://www.geeksforgeeks.org/implementing-forward-iterator-in-bst/
    - Iterate inorder and implement all left nodes recursively into the stack datastructure fo
      Complexity O(H)
- https://takeuforward.org/data-structure/flatten-binary-tree-to-linked-list/
    - Condition - we have to do preorder traversal(root-> left-> right)
    - Using stack data structure
        - Incase of stack DS -> we have to follow reverse of pre order -> (root -> right inse
        - Insert root
        - Pop out the element from the stack
        - Insert right elements first and then left elements
        - Map current node to top element of the stack
    - Using recursion
        - Go recursively **right** to the last of the BT
        - Then recursively go to the **left** of the BT
        - Finally assign the **previous** to the **right** of **current node**
        - Assign **left node** as null
        - Assign current node as previous
- Level order traversal in spiral form
    - https://www.geeksforgeeks.org/level-order-traversal-in-spiral-form/
    - Use 2 stacks..in 1 stack -> insert from right to left
    - In another stack while polling..-> insert from left to right
- Matrix in spiral form
    - https://www.geeksforgeeks.org/print-a-given-matrix-in-spiral-form/
    - Reduce the index in all 4 directions

We have already discussed AVL tree, Red Black Tree and Splay Tree. In this artic
efficiency of these trees:

Metric	RB Tree	AVL Tree	Splay Tr
Insertion in	**O(logn)**	**O(logn)**	**Amortiz**

e/

or the optimized Space

rt -> left insert)

cle, we will compare the

ree

ed O(logn)

worst case			
Maximum height of tree	**2*log(n)**	**1.44*log(n)**	**O(n)**
Search in worst case	**O(logn), Moderate**	**O(logn), Faster**	**Amortiz Slower**
Efficient Implementation requires	**Three pointers with color bit per node**	**Two pointers with balance factor per node**	**Only tw no extra**
Deletion in worst case	**O(logn)**	**O(logn)**	**Amortiz**
Mostly used	**As universal data structure**	**When frequent lookups are required**	**When sa retrieve**
Real world Application	**Database Transactions**	**Multiset, Multimap, Map, Set, etc.**	**Cache in Garbage**

Basis of comparison	Red Black Trees	AVL Trees
Lookups	[Red Black Trees](#) has fewer lookups because they are not strictly balanced.	[AVL trees](#) provide Red-Black Trees b strictly balanced.
Colour	In this, the color of the node is either Red or Black.	In this, there is no
Insertion and removal	Red Black Trees provide faster insertion and removal operations than AVL trees as fewer rotations are done due to relatively relaxed balancing.	AVL trees provide removal operation done due to relati
Storage	Red Black Tree requires only 1 bit of information per node.	AVL trees store ba with each node th an integer per noc
Searching	It does not provide efficient searching.	It provides efficier
Uses	Red-Black Trees are used in most of the	AVL trees are usec

Basis of comparison	Red Black Trees	AVL Trees
Lookups	Red Black Trees has fewer lookups because they are not strictly balanced.	AVL trees provide faster lookups than Red-Black Trees because they are more strictly balanced.
Colour	In this, the color of the node is either Red or Black.	In this, there is no color of the node.
Insertion and removal	Red Black Trees provide faster insertion and removal operations than AVL trees as fewer rotations are done due to relatively relaxed balancing.	AVL trees provide complex insertion and removal operations as more rotations are done due to relatively strict balancing.
Storage	Red Black Tree requires only 1 bit of information per node.	AVL trees store balance factors or heights with each node thus requiring storage for an integer per node.
Searching	It does not provide efficient searching.	It provides efficient searching.
Uses	Red-Black Trees are used in most of the language libraries like map, multimap, multiset in C++, etc.	AVL trees are used in databases where faster retrievals are required.
Balance Factor	It does not gave balance factor	Each node has a balance factor whose value will be 1,0,-1
Balancing	Take less processing for balancing i.e.; maximum two rotation required	Take more processing for balancing