Data structure

① Linear D.S
② Non Linear D.S

① Arrays
② Arraylist → Dynamic Array
③ Stack & Queue
④ Linked List

} Linear

Non-Linear D.S — ① Trees —
- G-Tree
- Binary Tree
- BST
- AVL
- Graph
- Heap

Use of D.S
- → Get
- → Set
- → Remove

Generic Trees

G-Tree Creation

Data → 10, 20 (means createnode)

Algo: ① Make root

② loop

```
if(data==-1){
  pop from stack
}
else{
  ① Make a node
  ② Push yourself in children of top of stack
  ③ Push yourself in stack
}
```

Level-order

Algo:
- ★ Remove
- ★ Print
- ★ Add

TC : O(n)

SC : ~~O(Height of tree)~~ O(width of tree)

① 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120

② 10

20, 30, 40

50, 60, 70, 80, 90, 100

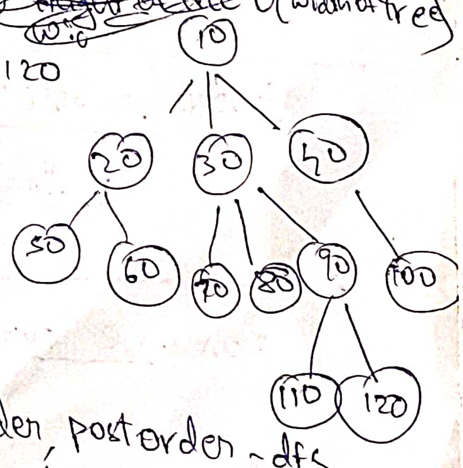
110, 120

③ 10

40 30 20

50 60 70 80 90 100

120 110

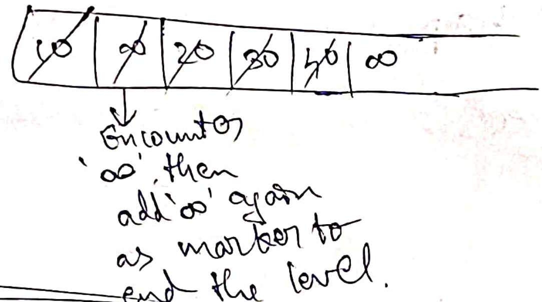


Pre order, post order - dfs

Level order - bfs

Approaches

Linewise level order: ① Use queue size to print in linewise
② Use two queue, q_1 and q_2;
q_1 represents current ele's / q_2 represents next level ele's
then keep swapping q_1 and q_2 at end. If both q_1 and q_2 is empty, then all are traversed.
③ Mark by adding '∞' at the end of level in the queue



Bonus ques.
Lintcode P. 1530

Linewise
Zig-zag traversal

even level → left to right traversal on children
odd level → right to left traversal on children

Approaches ① 2-stack
② 1-Queue, 1-Stack } Additional work: You have to add all elements of st into Q after each level

---

Mirror Generic Tree:

Faith: Every node reverses their children.

Remove leaves in generic tree: Post order

If node's children's child. size == 0, then remove it. Run loop on children in reverse (from last) to avoid removal index problem.

Linearize

In first approach **TC: O(n²)** because after each traversal at every node we are getting ~~seen~~ the tail of second last node. (which itself almost goes to O(n).)

Optimized: Instead ~~getTail~~ return tail at ~~ev~~ every traversal itself, so complexity will be **TC: O(n)**.

Find:

Just put a traversal, if found return true else return false.

Node to Root path:

- If found data then add node data and keep returning by adding parents through path. (if list size > 0 then found else not found)

LCA (Lowest Common Ancestor)

Refer self approach of t_1, t_2

Class Approach: Take two ArrayList for node To Root path. Loop in ~~until~~ list until you have common ancestors.

TC O(n) SC O(1) → Actually O(logn) Refer video

(If ignored, then SC O(1)) → Only for variable creation.

Refer self check1, check2 (with static variables)

True ✓

TC O(n) SC O(1)

Are Similar in Shape:

Approach: Check child counts

Are Mirror:

Approach: One node goes in pre order
Another node goes in post order
~~...~~ ~~...~~ And check child counts

Distance b/w two nodes:

Use node to root path for two given nodes. Then add total elements from LCA.

Is Symmetric:

Use is Mirror approach.

∴ Because a symmetric is always a mirror.

Predecessor & Successor:

State 0 → data not found

1 → data found

2 → data already found

Use Travel and change strategy

Ceil and floor

valid Candidates (V.C)

Ceil → [All elements having value greater than data] (Smallest among V.Cs)

Floor → [All elements having values smaller than data] (largest among V.Cs)

Ceil - '∞'

Floor - '-∞'

---

k^{th} largest

Make use of floor of tree for k-times to find k^{th} largest. For we need to traverse the tree 'k' times. And 'k' may by 'n' at worst case.

Leading to TC $O(n^2)$

MaxSubtree sum:

TC $O(n)$

* Using static variables of maxSum and maxSum node (Approach just return sum and check)

* Using Pair class where you return each subtree's sum, maxSum till now, maxSumNode till now

↓

This is to keep track if the total sum of subtree may be greatest sum, then we update maxSum with sum and corresponding node.

Diameter of Generic tree:

Diameter of two nodes

↓

Calculation from the Must be leaf nodes

Initial thought process:

Dch + Sdch + 2, we thought this might be answer.

(X) But there may a case that some node itself may have two very deep nodes.

(A) We have to consider this factor at every node.

Iterative pre and post order traversals:

Make use of stack. Have pair of node and class

State variable.
↳ which maintains structure of tree using stack.

LCA:

App 1: Use two Arraylist for nodeToRoot path & loop & ~~find~~ until you encounter common ancestor.

App 2: Use two variables t1, t2 in every call. Refer self code.

App 3: (With static variables) Use four static variables. Refer self code. SC O(1)
