

recap

- Tree intro
- Terminology
- Bin Trees
- Traversals
- Iterative traversal
- Construct tree w/ traversal

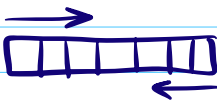
problems
new

Linear vs Hierarchical

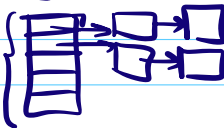
arr

HM

LL

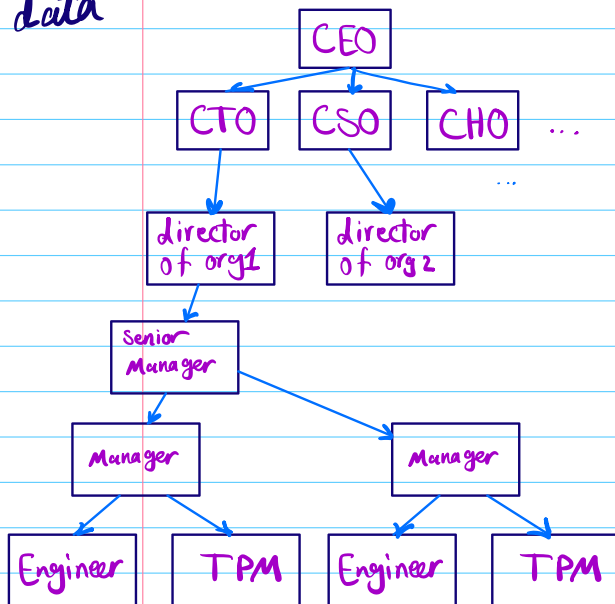
- Array 

-  linked list

- hash map 

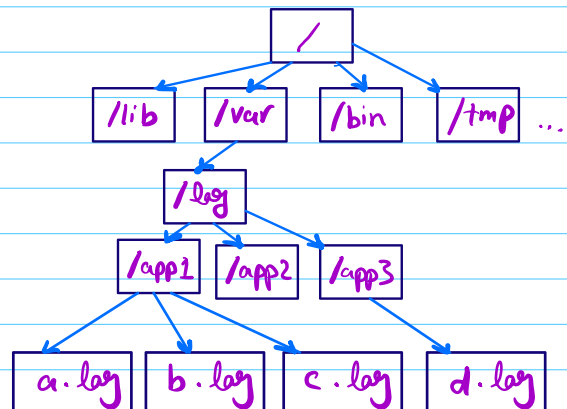
deque {
- Stack
- Queue

Hierarchical data ex: Company Org

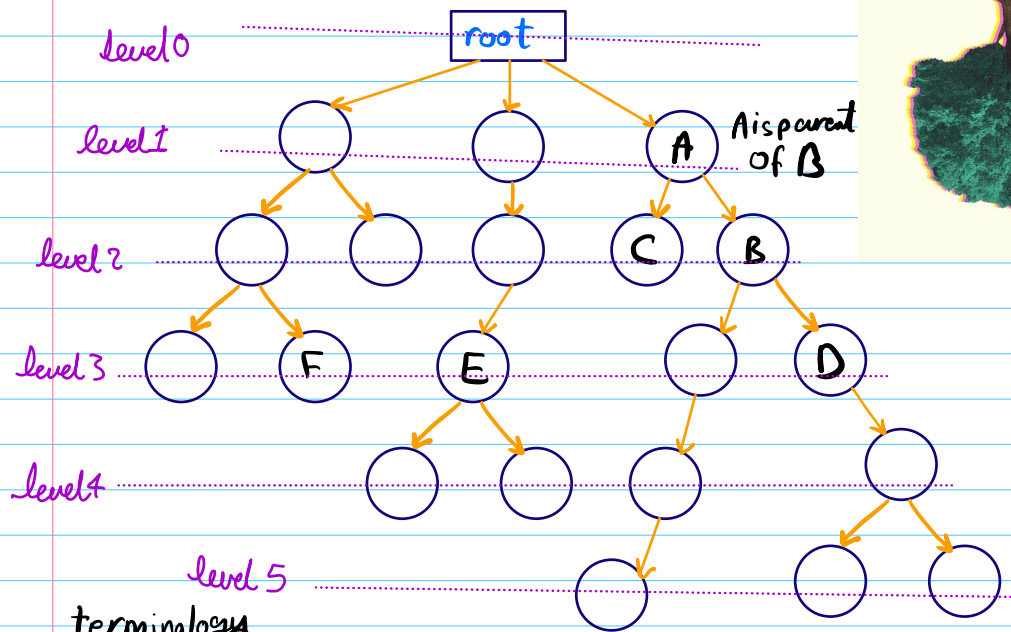


exs Family tree

exs File & Dir on Computer (linux)



Terminology



terminology

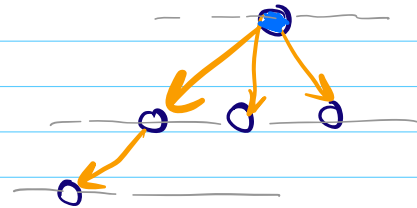
- A is parent of B
- B is child of A
- C & B siblings.
- F, E & D same level (cousins)

root: node with no parent

leaf: node with no child

Tree: | - only one root
| - each node has only one parent

Subtree



Q can leaf be subtree? Yes

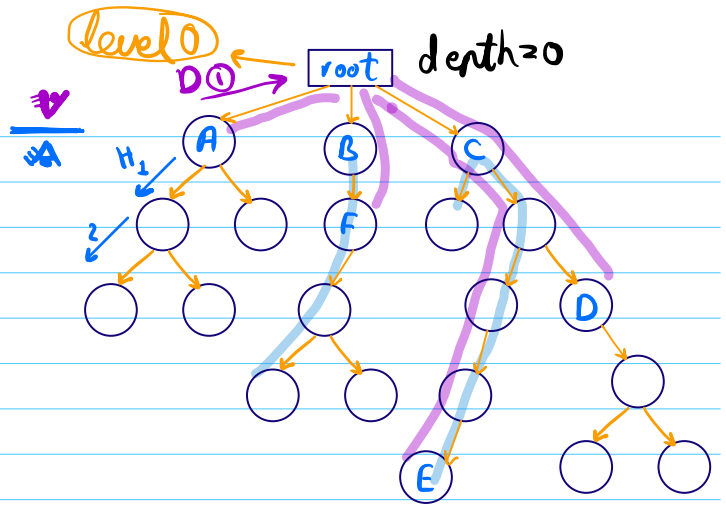
Q all nodes have parent? No (root)

Q root can be leaf? Yes

Q can tree have multiple roots? No

Height

length of longest path
from one node to any
of descendant leaf
node (Count edges)



Q $H(\text{leaf}) = 0$

$$H(A) = 2$$

$$H(B) \approx 3$$

Q $H(C) - H(E) = 4 - 0 = 4$

depth

length of path from root

$$D(A) = 1$$

$$D(F) = 2$$

$$D(E) = 5$$

$$D(D) = 3$$

max
depth
⇒ leaf

Height of tree = height of root

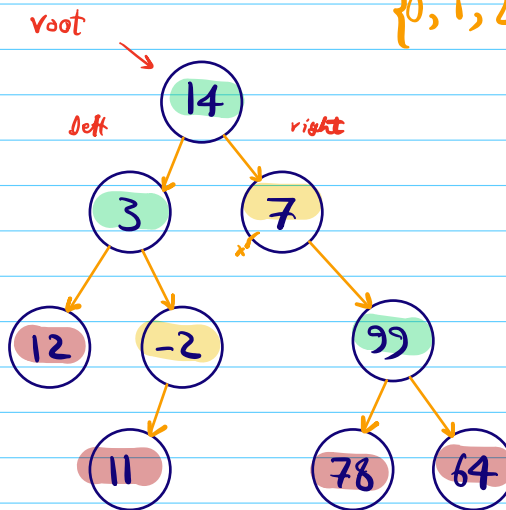
Depth of tree = ^{or} max depth of any leaf

Binary Trees

all nodes have max 2 children

$\{0, 1, 2\}$ 3 4 ...
x x

3 x



2 children
1 child
0 child
= leaf

```
class Tnode{
```

```
    int val;
```

```
    Tnode left;
```

```
    Tnode right;
```

```
    public Tnode(int v){
```

```
        this.val = v;
```

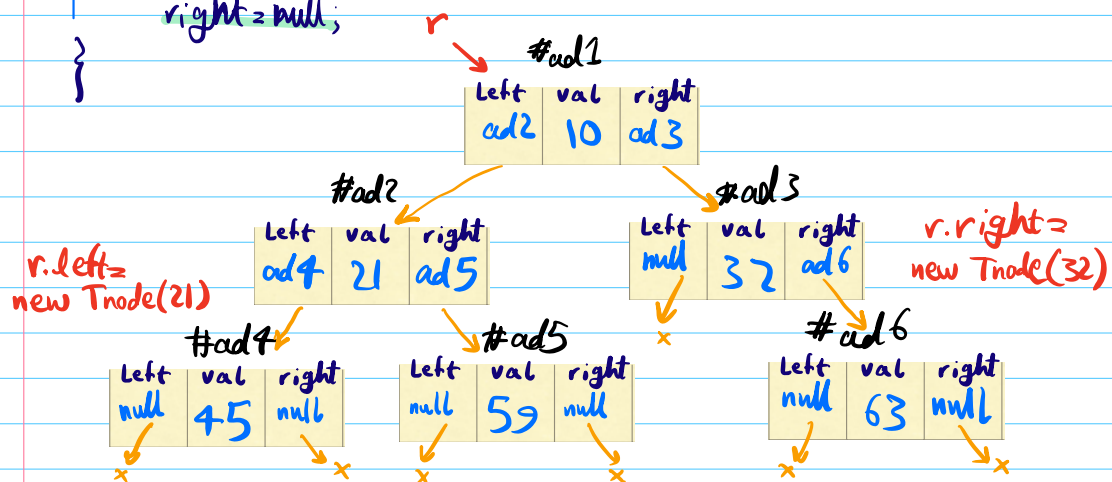
```
        left = null;
```

```
        right = null;
```

```
    }
```

Tnode

left	val	right



Tree Traversal

- preorder
- inorder
- post order
- level order

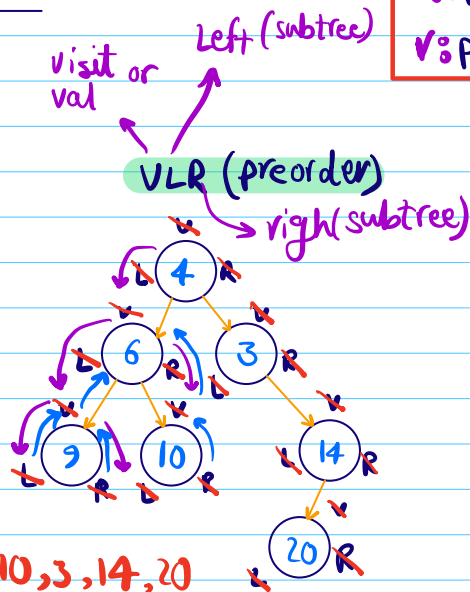
```

V: print(r.val)
L: preorder(r.left)
R: preorder(r.right)
    
```

not 100%
but useful

interview coding
tip for recursive
code on
LL and Trees good to check
null as base case
& not t.next or
t.left

Tc: $O(n)$
Sc: $O(h)$

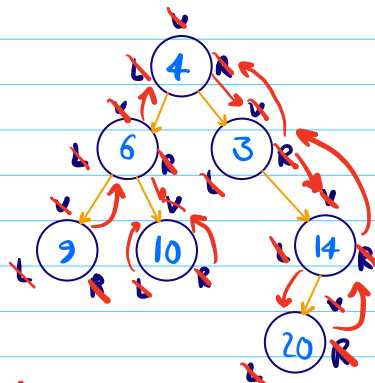


① ① ③
V L R

```

void preorder(TreeNode r){
    if(r == null) return;
    print(r.val)
    preorder(r.left)
    preorder(r.right)
}
    
```

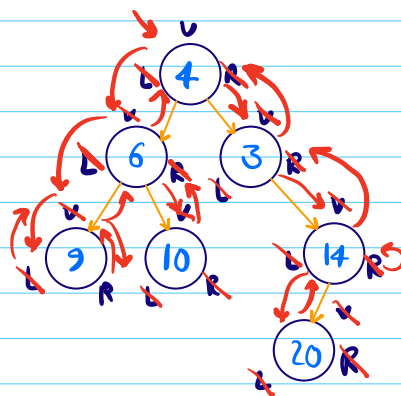
LVR



LVR

```

void inorder(TreeNode r){
    if(r == null) return;
    inorder(r.left)
    print(r.val)
    inorder(r.right)
}
    
```



LRV

```

void postorder(TreeNode r){
    if(r == null) return;
    postorder(r.left)
    postorder(r.right)
    print(r.val)
}
    
```

LRV: 9, 10, 6, 20, 14, 3, 4

Q

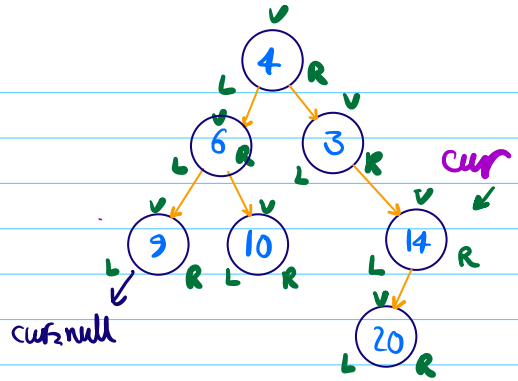
recursive

P1 Iterative inorder traversal

LVR

Stack<TNode>

20 6 3
4 10 14



Can we not use
any extra space?

Morse
Algorithm

```

cur = root
while (!st.empty() || cur != null) {
    if (cur != null) {
        ① st.push(cur)
        cur = cur.left
    } else {
        ② cur = st.pop()
        print(cur.val)
        ③ cur = cur.right
    }
}

```

TC $O(N)$
SC $O(H)$

ans 9, 6, 10, 4, 3, 20, 14

H.W itr. for LRV, VLR

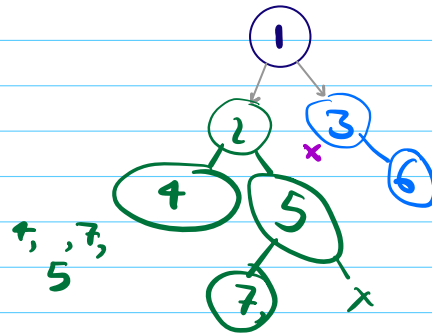
node vals are distinct

P2 Construct a tree from given inorder & Post order

0 1 2 3 4 5 6

LVR8 4, 2, 7, 5, 1, 3, 6

LRV8 4, 7, 5, 2, 6, 3, 1



hint

① last node of LRV is root

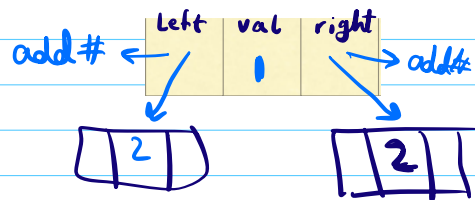
② inorder

... root ...

size of L subtree

size of R subtree

Tnode



tree 0, len-1 inclusive initial call

```
tNode tree(in[], post[], sin, ein, spost, epost){
    if( sin > ein || spost > epost){
        ret null
    }
    data ← post[epost]
```

TC: O(n)
SC: O(H)

```
{ root = new tNode(post[epost])
  rIndex = find_index_of_root_in_inorder(in[], sin, ein, data) → 4
```

a-b+1

didn't use !!

```
{ lSize = rIndex - sin // l subtree sin → rIndex-1
  rSize = ein - rIndex // r subtree rIndex+1 → ein
  root.left = tree(in, post, sin, rIndex-1, spost, spost+lSize-1)
  root.right = tree(in, post, rIndex+1, ein, spost+lSize, epost-1)
  ret root
}
```