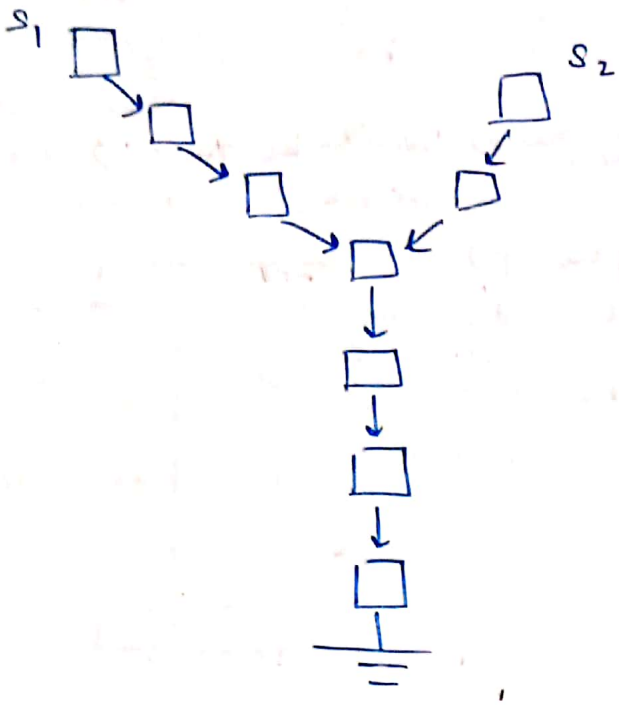
Day 22 Trees 1 (Introduction)

Q. Given 2 linked lists, return the junction.



approach 1: hash first list & then compare with the 2nd. The first collision would be the junction

↓

O(N) memory

O(N) time complexity

approach 2: if the lengths of the linked list is equal, then moving one step for each list and then intersection will surely arrive. (if present)

↓

But, the length might not be equal, so we calculate the difference between the two & according truncate one list to make them equivalent & accordingly apply the approach for equilength lists.

O(1) memory ←

O(N) tc

↓

2 passes

Trees

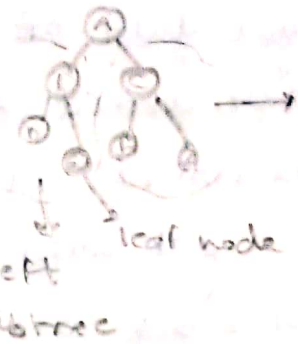
→ Binary Tree Node

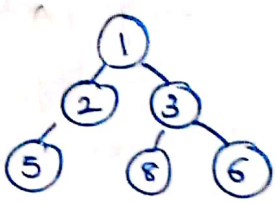
(Nary tree also used)

(*) A is parent of B, and ancestor of D, E, whereas D, E are called descendents of B, B.

↓

every node in the path from the root to the node is an ancestor.





(*) Traversals : i) inorder : LNR : 5 2 1 8 3 6

ii) preorder : NLR : 1 2 5 3 8 6

iii) postfix : LRN. : 5 2 8 6 3 1

```
inorder (node){
    if (node == NULL) return;

    inorder (node → left);
    print (node → val);
    inorder (node → right);
}
```

→ Similarly for preorder & postfix

*) Binary search tree : every element in the left subtree is smaller than root & every element in the right subtree will be larger than the root's value.

↓

Inorder traversal will be sorted values.

*) iterative approach for traversals :

i) postorder : maintain current, and in while loop if current is not NULL, then push right child of current if it exists and then current to stack, and make left child as current. If null, pop the stack, if the right child of the popped element & top of stack are same, then push popped element back to stack and make current as right. Else print current and make current = NULL.

ii) preorder : if current is not NULL, print it and push right child to stack and current equals to left child. If null, then current = st.top and continue.

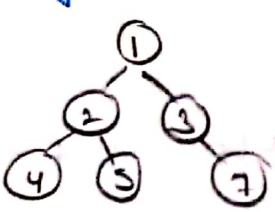
iii) inorder : push current to stack if not null and current = current → left.

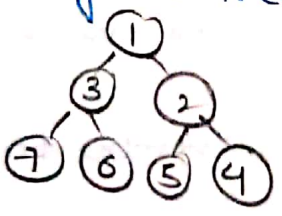
LNK

{ O(1)) space using threaded binary tree }

If null, then print top, and current = top → right and loop continues.

Q. Inverting the tree / mirror image of tree





```
if (root == NULL || (root->left == NULL &&
                     root->right == NULL))
    return root;

invert(root->right);
invert(root->left);

temp = root->left;
root->left = root->right;
root->right = NULL;
}
```

balancing trees
↑
*) AVL trees
*) Red Black trees

Q. Check if a tree is almost balanced abs. diff b/w (for every node x height of left subtree & right subtree <= 1)

height: includes root
depth: doesn't
↓
definition may vary
↓
better to clarify or test.
↓
return true or false
↓
can also be done using one variable, by returning if a true value is returned for applicable / valid ans, and -1, if not.

using struct of height & boolans
↓
we use these to not do repetitive work, which would req. repititive work of finding / computing heights.

Q. 2 traversals for tree construction.

(preorder, inorder

postorder, preorder

postorder, inorder)