Binary Trees 3

Saturday, 11 March 2023 10:00 AM

1a) Calculate no. of leaf nodes

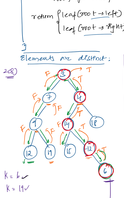
```
int leaf(Node *root) {
    if(root == NULL) return 0;
    if(root -> left == NULL && root -> right == NULL) return 1;
    return (leaf(root -> left) + leaf(root -> right))
}
```

Aim: Calculate no. of leaf nodes in given BT

Methodology: Traverse to get leaf nodes in LST + leaf(root->left) ; in RST : leaf(root->right)

Base Condition:

2a) Elements are distinct:



k = 6

k = 15

Solutions:

1) Apply any traversal & check

TC: O(N) SC: O(H)

6 is from root to parent

```
bool search(Node* root, int k) {
    if(root == NULL) {return false}
    if(root -> val == k) {
        return true;
    }
    return (search(root->left, k) || search(root->right, k))
}
```


TC: O(N) SC: O(H)

Obs: All Nodes which are returning true are on the path from $\boxed{\text{Root} \to (\text{Node})}$

In above example nodes returning true are:

$\boxed{3, 7, 14, -12, 6}$

// find path from root → (Node)

```
bool path(Node *root, int k, vector<Node*> &P)
    if(root == NULL) {return false}
    if(root -> data == k) {
        P.push(root)
        return true;
    }
    bool temp = path(root->left, k, P) || path(root->right, k, P)
    if(temp) {// cannot reach returning true
        P.push(root)
        return true;
    }
    else {return false}
}
```

passed by reference

we can use for as well

If k = 6

Tree we have with P looks like:

$\boxed{6 \mid -12 \mid 14 \mid 4 \mid 3}$

Note: It is from given k to root Node, if at all there exists a path

If see root from Root → Given

Reverse

Duplicates not there.

3a) Given a value to BT find shortest path.

Find path between 12 & -7

12 18 6 7 4 -7 ✓ ?

12 18 6 7 3 7 4 -7 ✗

Not shortest

Approach:

path 12 from 3 : 3 7 6 18 12

path 7 from 3 : 3 7 4 7

path 12 from 3 : 3 7 6 18 12

path 25 from 3 : 3 7 6 14 25

path 12 → 25 : 12 18 6 14 25

Find : 2

7 → 6 in ans1

3 → 7 in ans2

Iterate on both arrays & see last common index, say = Find

In first arr[] : A is size of 1st arr1 : O(H)

In 2nd arr[] : B is size of 2nd arr2 : O(H)

Note: ans1[A-1] → arr1[Find] → ans2[Find+1] → ans2[B-1]

TC: O(N + N + N) SC: O(H)

Given 12 & -7,

what is first node which contains both

Then node = 7 (LCA of both nodes)

Given 12 & 25

First node contains both : 6

Given 7 & 18

First contains both : 7

Least Common Ancestor

12 : 3 7 6 18 12

25 : 3 7 6 14 25

LCA of 12 & 25 is basically nothing but (last common element in both paths)

TC: O(N) SC: O(H)

LCA:

1) Do it using recursion (Try on your own)

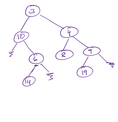
→ Search both Nodes at same time

2) Sparse Table + Binary lifting : O(log N) SC: O(N log N) no of nodes

3) Sparse Table + Euler path : O(1) ↳ Time is for precomputation

Advanced Concepts

4a) Given BT/ Invert it



```
void Invert(Node *root) {
    if(root == NULL) return;
    Invert(root->left);
    Invert(root->right);
    Node *temp = root->right;
    root->right = root->left
    root->left = temp
}
```

Aim: Invert Given BT

Meth: → Invert LST

→ Invert RST

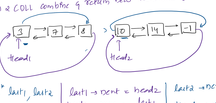
→ Swap root->left & root->right

Swap

5a) CDLL:

Given 2 CDLL combine & return new head Node?

Eg:



Node* last1, last2

last1 = head1->prev

last2 = head2->prev

last1->next = head2

head2->prev = last1

?

last2->next = head1

head1->prev = last2

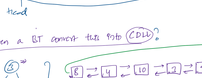
return head1

Note: Union of Sep Lists : if head1 == NULL: return head2

if head2 == NULL: return head1

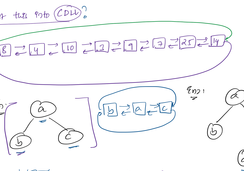
```
Node* combine2CDLL(Node *head1, Node *head2) {

}

Node* C3CDLL(Node *head1, Node *head2, Node *head3) {
    return combine(head1, combine(head2, head3))
}
```

output:



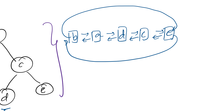
6a) Given a BT convert this into CDLL?

Eg:



Eg:

Eg:



Sol: Either inorder traversal or use of re-arrange links to get CDLL.

Don't use extra space

```
Node* BTCD(Node *root) {
    if(root == NULL) return NULL;
    Node *temp1 = BTCD(root->left)
    Node *temp2 = BTCD(root->right)
    //Make root a CDLL
    root->left = root->right = root
    return C3CDLL(temp1, root, temp2)
}
```

Aim: Given root Node of BT, convert the CDLL & return head node

CDLL (root->left):

: 8 ⇄ 4 ⇄ 10

Note: Make root single CDLL

CDLL (root->right)

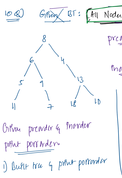
: 7 ⇄ 7 ⇄ 25 ⇄ 14

7a) Given BT, convert to DLL & return head?

Up → Try on your own → CDLL → DLL

2) Think of another way solution

8a) Given BT: All Nodes are unique

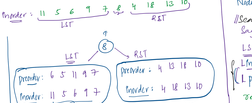


	0	1	2	3	4	5	6	7	8	9
preorder:	8	6	5	11	7	9	4	13	18	10
inorder:	11	5	6	7	9	8	4	18	13	10

LST RST

Given preorder & inorder find postorder

1) Build tree & print postorder



We can construct LST

We can construct RST

	0	1	2	3	4	5	6	7	8	9
preorder:	8	6	5	11	7	9	4	13	18	10

Build BT ✓

inorder: 11 5 6 7 9 8 4 18 13 10

inorder & LST

inorder & RST

Build BT ✓

postorder: 11 5 7 9 6 18 10 13 4 8

post order LST

post order RST

L&R:

Build LST ✓

Build RST ✓ → Build BT ✓

	0	1	2	3	4	5	6	7	8	9
preorder:	8	6	5	11	7	9	4	13	18	10
postorder:	11	5	7	9	6	18	10	13	4	8

We know root = 8

But we don't know what elements are there in left part & what elements are there in right part, we cannot construct

Obs:

1) BT: Nodes are distinct

{pre[] in[]} → BT is fixed

or

{post[] in[]} → BT is fixed

Note: Should have to be unique, if not partitioning into LST[] & RST[] is not possible (there is an ambiguity)

```
Node* constructBT(int pre[], int in[], N) {
    if (N == 0) {return NULL}
    Node *root = new Node(pre[0])
    //Search for pre[0] in inorder[] //
    Say it is present at find
    root->left =
        constructBT(L.preorder, L.inorder, m)
        ↳ Head node of LST
    root->right =
        constructBT(R.preorder, R.inorder, N-find-1)
        ↳ Head node of RST
    return root;
}
```

LST:

L.inorder[] : (0, find-1) in in[]

L.preorder[] : (1, find) in pre[]

RST

R.inorder[] : (find+1, n-1) in in[]

R.preorder[] : (find+1, n-1) in pre[]

```
Node* construct(int pre[], int sp, int ep, int in[], int si, int ei) {
    if (sp > ep) & return NULL}
    // root node is pre[sp], Node *root = new Node(pre[sp]); → O(1)
    // search pre[sp] in in[] & say it is present at find : ?  [This part can be optimized using hashmap ✓]  → O(1)
    [Keep inorder in this]
    LST: si ei
    inorder[] : [si, find-1]
    How many elements in LST = ?
    find - si - 1 = find - si
    preorder[] : [sp+1, x]
    x = (sp+1) + (find - si)
    x = find - si + sp

    RST si ei
    inorder[] : [find+1, ei]
    preorder[] : [x+1, ep]
    x = find - si + sp
    Calculation/Nothing more than this find.
    root -> left = construct(pre, sp+1, x, in, si, find-1) → T(LST)
    root -> right = construct(pre, x+1, ep, in, find+1, ei); → T(RST)
    return root → O(1)
}
```

→ O(1)

OVERALL TC: O(N)

OVERALL SC: O(N)