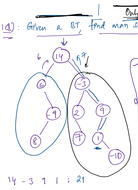
Binary Trees 4

1) Max sum path in BT
2) Sum of all nodes going down
3) Count nodes at K distance (Amazon)
4) Given a BT check if subtrees are repeating (Google)
→ Serialization/De Serialization
5) Find next parallel in your BT (VIMP)
6) Isomorphic or not
7) $E_{BT} < \begin{matrix}\text{Preorder} \\ \text{Inorder}\end{matrix}\}$ Construct BT

→ Sundays : 10 AM - 12 PM
→ Inorder/ Preorder/ Postorder
Iterative
→ LeftView/ RightView/ TopView/ Bottom
Recursion
→ Morris Traversal

→ preorder/ Inorder : postorder will be assumed
→ Trees a binary tree:

1a): Given a BT, find max sum path from root Node. (Path Value Inverse)



int maxsum (Node *root){
if(root == NULL) return 0
int l = maxsum(root→left)
int r = maxsum(root→right)
// return max(l,r) + root→data;
return [max(l, r, 0) + root→data]
↳ we need to include root
}

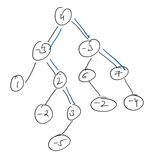
14 - 3 + 1 : 21
14 - 7 - 8 : 19

Eg:

Actual ans = 5
As per code = LST → -1
RST → -3
max(LST, RST) = 1
5 + -1 = 4

2a) Max sum path, if should contain root node in path.

Eg:



⇒ Max sum path: 3 → 2 → -5 → 4 → -3 → 7
Total Value : 10

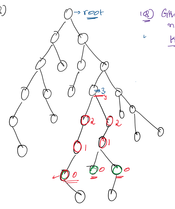
l = maxsum(root→left)
r = maxsum(root→right)

Final ans:
root→val + l + r
l>0 : root→val + l
r>0 : root→val + r
l<0 : root→val + 0
r<0 : root→val + 0

2a Solution
root→val + max(0, l) + max(0, r)
Final max path which contains root Node

// Max sum path containing a given node + Temp
int ans = INT_MIN
int maxsum (Node *root){
if(root == NULL) return 0;
int l = maxsum(root→left)
int r = maxsum(root→right)
// Considering root as Max sum path containing that node
ans = max(ans, (root→data + max(0, l) + max(0, r)))
return {max(l, r, 0) + root→val}
}
// ans contains: Max sum path in given BT
TC: O(N), SC: O(H)
↳ Height of tree

2b)



2b) Given a source node how many nodes are there at distance K, below that (Children)

int NodesBelowK(Node *S, int K){
if(S == NULL) return 0;
if (K<0) return 0;
if(K == 0) return 1;
return [NodesBelowK(S→left, K-1) + NodesBelowK(S→right, K-1)]
}
↳ pass by value

2c): Calculate no of nodes at a distance K from given source node.

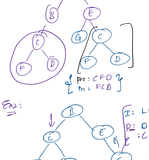
Path to S.c from root
D → C → B → A
A→right → D
Search A→left K=0
B→right → C
Search B→left K=1
C→left → D
Search C→right K=2
Note: At any point if K<0, break it
TC: O(N)

Pseudocode:
Src → Root
1) Get path from Src → dest in Vector<Node*> path
int c = NodesBelowK(Src, K);
for(int i = 1; i < path.size(); i++){
if(path[i]→left == path[i-1]){
c += NodesBelow K(path[i]→right, K-i-1)
}
else{
c += NodesBelowK(path[i]→left, K-i-1))
}
}
return c;

2d) Given BT contains data, has a single character, Based on that check if there exists 2 subtrees which are exactly same. [Node* node to subtree 2]

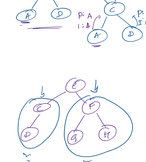
#1) If subtrees are same
#2) How many different subtrees are there

Eg:



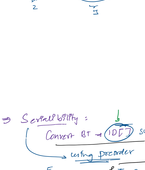
Solutions:
1) Take 2 nodes from tree & check if there are same subtrees or not
⇒ N_{C_2} → [N] paths
Overall TC: O(N²) SC: O(N + H) ⇔ recursion

Eg:



1: LOR: ACD
2: DLR: CAD

d store all N nodes in an array so that we can pick 2 pairs



2) For all nodes (Value), calculate no of nodes in left subtree.
Eg: Here: [Count nodes having K nodes to complete]

5	3
7	2
6	3
2	3
4	2
1	3

$\times N_{C_2}$

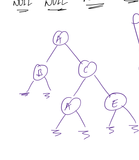
3) For all subtrees calculate Inorder/Preorder
postorderstring string s = hashtrees.
Insert < postorderstring, string s > hm.
[preorder of BT to string format] [Inorder of BT to string format]

postorderstring, string map (Node *root){
if(root == NULL) { return {"", ""}; }
postorderstring, string l = map(root→left)
postorderstring, string r = map(root→right)
// For given root how to get preorder
return {root→data + l.first + r.first , l.second + root→data + r.second}
preorder of current node, Inorder of current node
}
→ How to get postorder & preorder

⇒ Serializability:
Convert BT → IOT such that it can be reconstructed back.
using preorder

{E C D N N N F G N N H N N}

Serializability using using preorder
A B N D N N E F N N N



void serializeBy(Node root, vector<Node*> &data){
if(root == NULL) { data.push(NULL) return}
data.push(root);
serializeBy (root→left, data)
serializeBy (root→right, data);
}
// only enter node if data[i] at contains entire BT in an array

For given Question // Say we want to get serialize
// hashmap<string, int> hm;
string ser (Node *root){
if(root == NULL) { return '@'; }
↳ character which is not there in tree
string l = ser(root→left)
string r = ser(root→right)
// BT is (root→data + l + r)
string data = root→data + l + r
if(data.size() > 3){
// Subtree contains more than 1 node
// Insert in hashmap/hashset
Based on question, C++
}
}

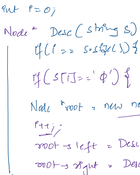
preorder serie
B ⇒ B @ @
B ⇒ B A @ @ @

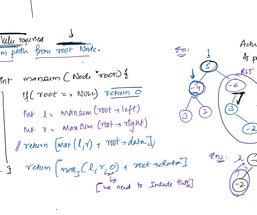
Note:
Reason for this is subtree with 1 node should not be considered, as per question.

// Given N nodes
1) Serializability: at most → number → O(2N)

// Now looking at frequencies in hashmap
we can say
Whether they are repeating or not?
How many different subtrees are there?

// In above question how many subtrees are there ⇒ N
For each subtree we calculate its preorder serializability
In hashmap how many things do we insert ⇒ N
overall TC: O(N²)
overall SC: O(N²)
Based on hashmap





int i = 0;
Node* Desc(string s){
if(i >= s.size()){ // it will never come to this stage}
if(s[i] == '@') { i++; return NULL; }
Node *root = new node(s[i]);
i++;
root→left = Desc(s)
root→right = Desc(s)
}
return root;

show root & data point on always NULL, NULL in serialize