Problem-1) Write an efficient algorithm to check if two binary tree are equal or not.Two binary trees are identical if they have identical structure and their contents are also the same

We will try to parallely traverse both the trees in the same manner(either inorder, preorder or postorder). If at any point in time, the value of the current node is different; or if one of 2 pointers of 2 trees is null but the other one is not null then the trees are not the same.

Here, I'm using inorder traversal.

Node of tree has 3 fields.

value - stores the data
left - stores the reference to left child
right - stores the reference to right child

Input parameters : root of both trees

Algorithm isTreeSame(r1, r2):

```
IF r1 is null and r2 is null:
        Return True
IF (r1 is null and r2 is not null) or (r1 is not null and r2 is null):
        Return False
IF r1.value == r2.value:
        Return isTreeSame(r1.left, r2.left) and isSameTree(r1.right, r2.right)
Return False
```

Time Complexity:

Since we are using inorder traversal here, the worst case complexity is O(max(n1, n2)) where n1 and n2 are the number of nodes in tree 1 and 2 respectively.

However, note that since we are using inorder traversal, the search would stop as soon as a "non-identical" node is found in the parallel inorder search path of trees. Here by non-identical node, I mean the corresponding nodes in both trees where the value field is not the same or where node is not present in one tree but corresponding node is present in other tree while doing the parallel inorder traversal.

Space Complexity:

Since, we are using recursion here, the max. depth of recursive call stack would be equal to minimum height among both the trees. Hence it is log(min(n1, n2)) where n1 and n2 are the number of nodes in tree 1 and 2 respectively on an average but could be O(min(n1, n2)) when trees are skew.

Problem 2.) A binary tree is given. Check whether it is a BST or not. Consider the following for example which is not a BST.

Generalize the solution for any tree and estimate the time complexity of the algorithm.
The code is not expected-just give the solution/algorithm.

It is a well known property of BST that the inorder traversal of a BST processes the elements in ascending order. Also, if a binary tree has its inorder traversal in ascending order then that tree is a binary search tree.

We can use this fact to check whether the given tree is BST or not. We will find the inorder traversal of the given tree and if it is in ascending order then we can say that the tree is BST. So, we will use a FIFO structure to store the elements while doing inorder traversal and after the inorder traversal, we will check whether the elements of FIFO structure are in strictly ascending order or not.

Node of tree has 3 fields.

- value - stores the data
- left - stores the reference to left child
- right - stores the reference to right child

Input parameters : root of tree

Algorithm checkBST(root):

```
Initialize a vector V
inorder(V, root)
FOR i = 2 to V.length:
    IF V[i] <= V[i - 1]:
        Return False
Return True
```

Input parameters : a Vector and root of tree

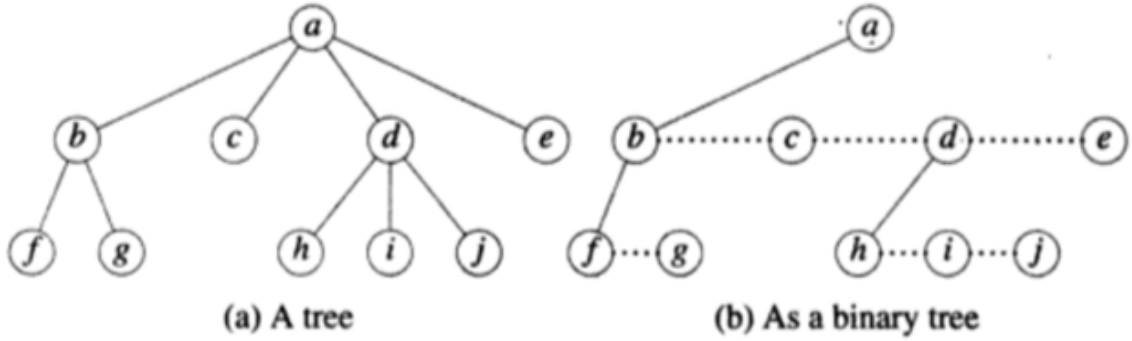
Algorithm inorder(V, root):

```
IF root == null:
        Return
inorder(V, root.left)
V.append(root.value)
inorder(V, root.right)
```

Time Complexity:

Since, we are doing inorder traversal, the time complexity will be O(n) where n = number of nodes in the given tree.

Problem 3- .Convert a general tree into a binary tree using following scheme (see the figure):



The leftmost child is made as the left child and all the nodes of that level become the right child and its children in sequence (Source: Sartaj Sahni, go through the book for more details if required). What is the time complexity of the conversion algorithm?

We can write the given algorithm, described in words, formally as follows.

I'm assuming that the general tree has the following structure for its node.

- value : which stores the data
- child : which is an array of references to child nodes of this node

Also I'm assuming that the binary tree has the following structure for its node.

- value : which stores the data
- left : reference to left child
- right : reference to right child

Input : root of general tree

Algorithm convertToBT(root):

```
IF root == null:
        Return null
node = create a binary tree node with root.value

IF root has >= 1 child node:
        node.left = convertToBT(root.child[1])

next = node.left
FOR i = 2 to root.child.length
        next.right = convertToBT(root.child[i])
        next = next.right
Return node
```

We can see that the function convertToBT() is called exactly once for every node in the general tree. Hence, its time complexity is O(n) where n = nodes in the general tree.

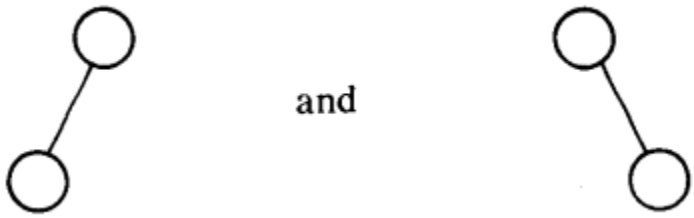
Problem 4: How many binary trees can be generated for a given input (n nodes, Briefly explain using example)?

Let b_i denote the number of structurally distinct binary trees that can be constructed using i number of indistinguishable nodes.

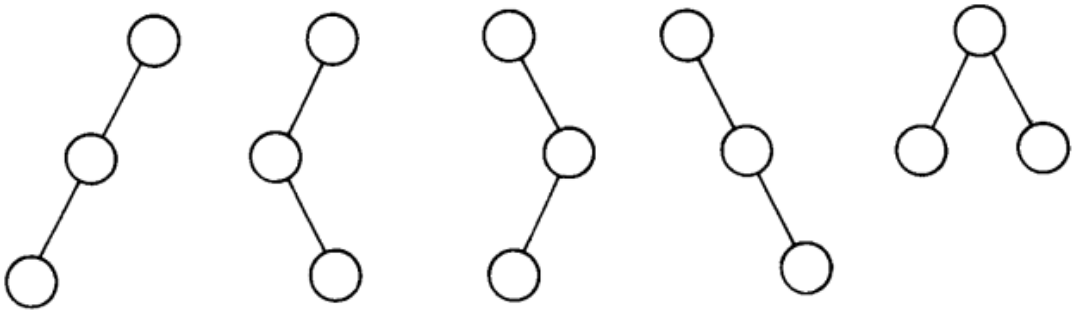
Then it is easy to see that only one binary tree can be constructed with 0 or 1 node.
So, $b_0 = b_1 = 1$.

Also, it is also relatively easy to enumerate the number of distinct binary trees that could be formed using 2, 3 nodes. These are shown as below.

2 distinct binary trees using 2 nodes:

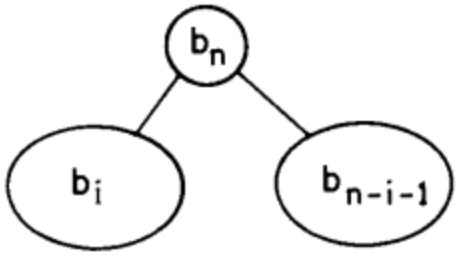


5 distinct binary trees using 3 nodes:



Now, it is also possible to enumerate all possible distinct binary trees with 4 or more nodes but we can see that it quickly becomes very difficult and tedious.

Since the subtree of a tree are also binary trees, we can formulate a recursive relationship on the number of distinct binary trees possible having n nodes as follows.



i.e., for a binary tree having n nodes, it is possible to have 0 to n - 1 nodes in the left subtree of root. Similarly, it is possible to have 0 to n - 1 nodes in the right subtree of the root.

So, the total number of distinct binary trees with n nodes will be,

$$b_n = \sum_{i=0}^{n-1} b_i * b_{n-i-1}$$

This recursive relation can be solved using generating functions.

Let the generating function be defined as follows.

$$B(x) = \sum_{i>=0} b_i * x^i$$

This can be simplified as follows.

$$xB^2(x) = B(x) - 1$$

This quadratic equation can be solved easily which will give the following value to B(x).

$$B(x) = (1 - \sqrt{1 - 4x})/2x$$

We can use the binomial theorem to expand the $(1\text{-}4x)^{1/2}$. It will give the following result.

$$b_n = \frac{1}{n+1}\binom{2n}{n}$$

We can also verify this result for b_2 which will be ${}^4C_2/3 = 2$ and similarly for b_3 it will be ${}^6C_3/4 = 5$ which is correct.

References:

Images taken from book “Fundamentals of Data Structure”.