Assignment 1

Krati Arela - EE18BTECH11050

Download all latex-tikz codes from

https://github.com/Krati012/C−and−DS/tree/main/Assignment1

1 Problem

(Q. 20) The postorder traversal of a binary tree is 8,9,6,7,4,5,2,3,1. The inorder traversal of the same tree is 8,6,9,4,7,2,5,1,3. The height of a tree is the length of the longest path from the root to any leaf. The height of the binary tree above is _____?

2 Solution

Answer: The height of the above binary tree is 4.
Explanation: **Postorder traversal** of a binary tree is a depth-first search in which we first traverse the left subtree, then traverse the right subtree, and finally visit the root node. Whereas in **inorder traversal** we first traverse the left subtree, visit the root and then traverse the right subtree.
If there are n nodes in binary tree, maximum height of the binary tree is $n-1$.
A complete binary tree will have minimum height for given n number of nodes. Then height h will lie in the range:

$$2^h \leq n \leq 2^{h+1} - 1 \tag{2.0.1}$$

Thus, solving this inequality we get:

$$h \leq log_2 n, h \geq log_2(n+1) - 1 \tag{2.0.2}$$

Since h is an integer, we finally get,

$$h = floor(log_2 n) = ceil(log_2(n+1) - 1) \tag{2.0.3}$$

Therefore, for a binary tree maximum height is $n-1$ and minimum height is $floor(log_2 n)$. For given problem, n=9, so maximum height of the binary tree possible is 8 and minimum height possible is 3.
The following algorithm allows us to construct the binary tree from given $inorder = [8,6,9,4,7,2,5,1,3]$ and $postorder = [8,9,6,7,4,5,2,3,1]$:

- First find the last node in postorder[], as root always appear in the end of postorder traversal. The root is "1" in this problem.
- Search "1" in inorder[] to find left and right subtrees of root. Everything on left of "1" in inorder[] is in left subtree and everything on right is in right subtree.
- Recur the above process for following two:
  - Recur for $inorder = [8,6,9,4,7,2,5]$ and $postorder = [8,9,6,7,4,5,2]$. The created tree will be left child of root.
  - Recur for $inorder = [3]$ and $postorder = [3]$. The created tree will be right child of root.

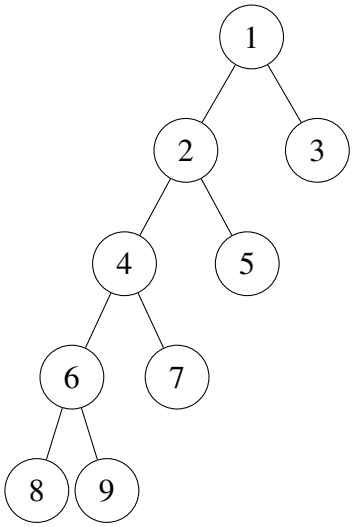

Fig. 0: Constructed Binary Tree

The following C code uses the above algorithm to construct the tree and prints the height of the tree as output.

codes/height_bst.c

The constructed tree from above algorithm is shown in Fig. 0
Thus, we can see that the height of the tree is 4.

3 Generating Order Array

The following algorithm allows us to generate in-order and postorder traversal array for any given

height h.

- Initialize array of size $2*h+1$ for inorder and postorder arrays.
- Initialize root of binary tree as 1 and recursively create child nodes of root with values as $2*(i+1)$ and $2*(i+1)+1$, with $i = 0, 1, .., h-1$.
- Insert the values of the nodes of the tree in inorder or postorder traversal as required in array and return the array.

The following code generates inorder and postorder traversal arrays for random binary tree for any given height.

codes/generate_order.c

To verify the algorithm, generate arrays for any height h, for eg, $h = 20$, and apply the function to construct and find height of binary tree in height_bst.c, which gives height of the tree as 20. Thus the algorithm is verified.

4 Complexity

The following code runs our program height_bst.c for different values of n (which is the height of the tree) and saves the execution time in a text file.

codes/run.c

5 Solution

Let us assume the time complexity to process the given input of size n to be $T(n)$ and time taken to process input $n-1$ will be $T(n-1)$. Our program recursively calls the construct function for right and left subtrees.

$$T(n) = T(n-1) + n + C \tag{5.0.1}$$

We get eq (5.0.1) as in the worst case, search in inorder array will take $O(n)$ time, and C is some constant time taken in the process.

$$T(n-1) = T(n-2) + n + C \tag{5.0.2}$$

Substituting (5.0.2) in (5.0.1),

$$T(n) = T(n-2) + 2n + 2C \tag{5.0.3}$$

On repeated substitution till $n->0$,

$$T(n) = T(0) + n*n + nC \tag{5.0.4}$$

$$\implies T(n) = n^2 + nC + T(0) \tag{5.0.5}$$

$$\implies T(n) \leq n^2 \tag{5.0.6}$$

$$\implies T(n) \in O(n^2) \tag{5.0.7}$$

Thus, our program has $O(n^2)$ time complexity. The given python file plots the time complexity using the execution time for different size of inputs.

codes/exec_time.py

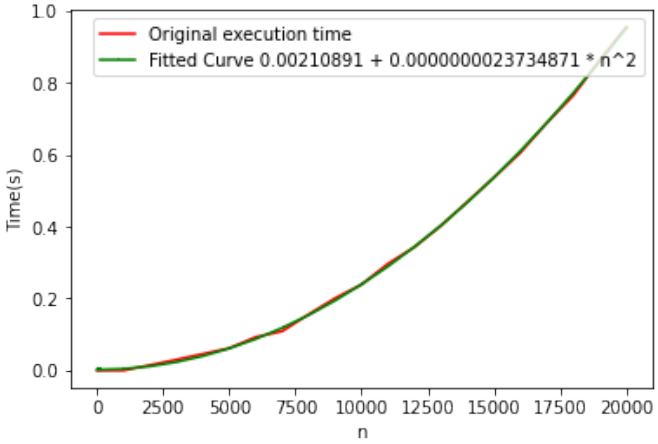

Fig. 0: $T(n)$ *vs* n

We can clearly observe that the original execution time and our mathematical time complexity align with each other.