Reverse Binary Tree Traversal Algorithm Discovery

Contributed by:
Howard Whitston (modification of Pete Henderson’s Reverse Binary Tree)

Keywords:
In-order and pre-order binary tree traversals, generalization, algorithm discovery, counting sequences, graph isomorphism, functional programming, recursive problem solving

Problem Statement:
1. Using the in-order and pre-order traversal lists, determine the binary tree structure and a series of questions lead the student to discovery the general pattern for the solution
2. Using the general pattern, the student can show the algorithm with a functional programming program solution (optional)

Type:
In-class group experiment, computer lab

Prerequisite Knowledge:
Tree Traversal, Graph Theory, Binary Trees

Learning Outcomes:
To reinforce the tree traversals and general tree construction. To demonstrate recursive algorithm design in a non-standard problem. To show, again, how to construct a general solution from a specific problem.

Connections:
Algorithm design strategies, binary trees construction, complexity classes

Pedagogical Notes:
This problem can be done at the same time as Pete Henderson’s Reverse Binary Tree Traversal thus showing the similarities and differences between pre-order and post-order traversals of a binary tree. It sets up the environment for working on the third part of this problem. It isn’t a trivial problem.
Brief Solution:
From the pre-order list, one can find the root and the right-most leaf. From the in-order list, you can split the list at the root and now recursively backtrack to the pre-order list to get the right and left sub-trees roots. Continue building the tree until you have placed all items from the lists. Now, double check your solution binary tree by doing the pre- and in-order traversals. Next, generalize the method to get the algorithm and build the function(s) using a functional programming language.

Sources:
D. Knuth, “Art of Computer Programming”, Vol 1, Chapter 2; Vol 3, Chapter 6

Complete Problem Statement:
The following are the in-order and pre-order traversals of a single binary tree whose 10 nodes are labeled 0, 1, 2, . . ., 9.

in-order:   3  1  7  5  0  4  8  2  9  6
pre-order: 0  1  3  5  7  2  4  8  6  9

a. Draw the corresponding tree T with the nodes labeled.
b. Consider 2 nodes labeled 0 and 1. If the sequences in-order and pre-order are any permutations of 0 and 1, is it always possible to construct a corresponding binary tree? If yes, give an argument. If no, give a counterexample.
c. Consider 3 nodes labeled 0, 1 and 2. If the sequences in-order and pre-order are any permutations of 0, 1 and 2, is it always possible to construct a corresponding binary tree? If yes, give an argument. If no, give a counterexample.
d. Give a brief explanation for your answer to (c) above.
e. Give a binary tree with 5 nodes 0, 1, 2, 3 and 4 whose in-order and pre-order sequences are the same. How many such, non-isomorphic trees with 5 nodes are there?
f. What binary tree corresponds to in-order sequence = pre-order sequence = empty?
g. Given the in-order and pre-order traversals in (a) above, give the set of nodes in the left subtree of T.
h. Given the in-order and pre-order traversals in (a) above, give the set of nodes in the right subtree of T.
i. Given the in-order and pre-order traversals in (a) above, give the in-order traversal of the left subtree of T.
j. Given the in-order and pre-order traversals in (a) above, give the in-order traversal of the right subtree of T.
k. Given the in-order and pre-order traversals in (a) above, give the pre-order traversal of the left subtree of $T$.
l. Given the in-order and pre-order traversals in (a) above, give the pre-order traversal of the right subtree of $T$.
m. What is the label of the node of the left subtree of $T$?
n. What is the label of the node of the right subtree of $T$?

**Generalizing what has been discovered:**
Let $n \geq 0$ with traversal sequences $\text{in-order} = i_0 \ i_1 \ i_2 \ \ldots \ i_n$ and $\text{pre-order} = p_0 \ p_1 \ p_2 \ \ldots \ p_n$, where there are no duplicates in the sequences {NOTE: to better discover these generalizations, try some examples if you get stuck}.

a. Which node is the root of the corresponding binary tree?
b. What nodes are in the left subtree?
c. What nodes are in the right subtree?
d. Give the sequence corresponding to the in-order traversal of the left subtree?
e. Give the sequence corresponding to the in-order traversal of the right subtree?
f. Give the list corresponding to the pre-order traversal of the left subtree?
g. Give the list corresponding to the pre-order traversal of the right subtree?
h. What node is the root of the left subtree?
i. What node is the root of the right subtree?
j. Using the terminology of (g), what general constraints on the sets must be true for the existence of a corresponding binary tree?

**Towards a general algorithmic solution:**
Represent the in-order and pre-order traversal sequences as lists in Standard ML (e.g, 
val in-order = [3, 1, 7, 5, 0, 4, 8, 2, 9, 6]; ) The goal is to develop a Standard ML function definition

\[
\text{createBT : 'a list X 'a list } \rightarrow \ 'a \ BinaryTree
\]
which creates the corresponding binary tree $T$ given the two traversal sequences/lists.

Identify and name the required helper functions using the generalized concepts you discovered in the previous question. Give a brief description of the purpose of each helper function.

**Solution hints:**
a. Function 'root' returns the root of the tree $T$ (ie, the last element of the pre-order list)
b. Functions that return subsequences/sublists corresponding to the in-order traversals of the left and right subtrees of $T$ [obtained by splitting 'in-order' into two sublists
determined by the root element from (a)]
i) in-order_left_subtree : list X splitting_node → list (e.g., in-order_left_subtree( [1, 2, 3, 4, 5, 6, 7, 8], 5 ) = [1, 2, 3, 4] for splitting node 5)

ii) in-order_right_subtree : list X splitting_node → list (e.g., in-order_right_subtree( [1, 2, 3, 4, 5, 6, 7, 8], 5 ) = [6, 7, 8] for splitting node 5)

C. Functions that return subsequences/sublists corresponding to the pre-order traversals of the left and right subtrees of T [extracted by knowing the number of nodes in the left and right subtrees - length of lists in (b-i) and (b-ii) above, and corresponding positions of these sublists in the pre-order traversal]

iii) pre-order_left_subtree : list X number_of_nodes LST → list (e.g., pre-order_left_subtree( [1, 2, 3, 4, 5, 6, 7, 8], 4 ) = [1, 2, 3, 4])

iv) pre-order_right_subtree : list X number_of_nodes RST X n → list (e.g., pre-order_right_subtree( [1, 2, 3, 4, 5, 6, 7, 8], 3, 8 ) = [5, 6, 7])

Note 1:
You might use a function permutation : 'a list X 'a list → bool to determine if a valid tree exists.

d. Compose, verify, and test each of these helper functions.

e. Compose the definition of the recursive function:
createBT 'a list X 'a list → BinaryTree
given the function bt : node_label X left_subtree X right_subtree → BinaryTree for creating a binary tree given root, left subtree and right subtree, the empty binary tree 'bt_empty' and the functions root( pre-order ), left_sublist(in-order, root ) and right_sublist( in-order, root )

fun createBT( [], [] ) = bt_empty
| createBT( in-order, pre-order ) =
  bt( root,
  createBT( in-order_left_subtree, pre-order_left_subtree ),
  createBT( in-order_right_subtree, pre-order_right_subtree ) );

4