Tree traversals: problems

Think about:

- Assume you have a binary tree with distinct elements, you do not know how it looks like, but you have the preorder and inorder traversal of the tree. Give an algorithm for building the tree based on these two traversals.

  e.g.:

  Preorder: A B F G H E L M

  Inorder: B G F H A L E M

- Can you rebuild the tree if you have the postorder and the inorder traversal?
- Can you rebuild the tree if you have the preorder and the postorder traversal?

  e.g.: Pre: 1 2 3 4 5

  Post: 5 4 3 2 1 or: 3 4 2 5 1

What if the tree is full?

Tree traversals: problems

Rebuild the tree if you have the preorder and the postorder traversal of a tree with distinct elements. We also know that the tree is full.

- The first element in the preorder traversal is the ***root*** of the tree.
- If there are next elements , the next one is its ***left_child***.
- We will find the ***index*** of ***left_child*** in the postorder traversal.
- All the elements to the left of this ***index*** and element at this index will be in the left subtree of ***root***. And all the elements to the right of this index will be in the right subtree of the ***root***.

  Now we have the preorder and postorder traversals for the 2 subtrees of the ***root***!

e.g.:

Preorder : 1, 2, 4, 5, 3, 6, 8, 9, 7

Postorder : 4, 5, 2, 8, 9, 6, 7, 3, 1

Tree: other problems

Think about:

Give the iterative and recursive algorithms for the following problems:

- Search for a given element in a binary tree.
- Determine the height of a binary tree.
- Determine the level at which a given element appears in a binary tree.
- Determine the parent of a node containing a given element .

Tree traversal

When we have a tree that is not binary, there are two possible traversals:

Level order (breadth first) - looks exactly the same as in case of a binary tree, we use an auxiliary queue to store the nodes.

Depth first - similar to breadth first, but we use an auxiliary stack to store the nodes (it is the generalizations of preorder traversal). By using a stack, the algorithms will always go as deep as possible in the tree and only once a whole subtree of a node was visited we pass to the next child.