Exercises – Week 8

Trees

1. Write a function that takes a binary tree and returns the list of nodes that have a single child. The order of the nodes in the list will be that of the inorder traversal.

2. Write a function that takes a binary tree and returns the total number of nodes in the tree.

3. **Traversal of arbitrary trees:** Modify traversal functions to work on arbitrary trees (each node has a list of children). Use list traversal functions. For inorder, first traverse the head of the list, then the root and then tail of the list.

4. **Indented Printing:** Write a function that displays a binary tree of integers in preorder, one node per line, preceding the value of the node by a number of spaces equal to twice its depth (two spaces for each level).

5. **Removing a node:** Write a function that takes a value and a binary search tree as parameters and returns the tree from which the value was removed (if present).
Tips:

- if the tree is not empty:
    - if the given value is identical to the root key then we have found the searched node. We proceed as follows:
        - if the node is terminal (the left subtree and the right subtree are empty) this node will be deleted, and the address retained by the parent in its place becomes None;
        - if only the left subtree is non-empty the node will be deleted, and the address retained by the parent in its place becomes the address of the left subtree;
        - if only the right subtree is non-empty the node will be deleted, and the address retained by the parent in its place becomes the address of the right subtree;
        - if both subtrees are not empty:
            - the biggest node in the left subtree is identified (the rightmost node of the left subtree). This node cannot have a right subtree!
            - the information from this node is copied into our searched node;
            - the identified node is deleted. The deletion is performed as if the node is terminal or as if the node has only left subtrees;
    - if the given value is smaller than the root key, the search continues in the left subtree;

    - if the given value is greater than the root key, the search continues in the right subtree;
- if the tree is empty:
    - the searched value does not exist in the tree.