UNIVERSIDAD POLITÉCNICA DE YUCATÁN

Tree

Homework 2.1

Objective

The student will build non-linear structures for data storage and retrieval.

Instructions

Modify the program implemented in class to use the following functions:

1. **Depth (15 pts)**

Write a function `depth()` that receives a binary tree (`rootPtr`) and determines its depth.

2. **Leaf Nodes (15 pts)**

Write a function `countLeaves()` that receives a pointer to the root of a binary tree (`rootPtr`) and count its leaves.

3. **Nodes at a level (15 pts)**

Write a function `printLevel()` that receives a pointer to the root of a binary tree (`rootPtr`) and print all the nodes at a certain level. If the provided level is out of range print a warning.

4. **Level Order Traversal (25 pts)**

The level order traversal of a binary tree prints the node values level-by-level starting at the root node level. The nodes on each level are printed from left to right. This traversal is not a recursive algorithm. It uses a queue data structure to control the output of the nodes. The algorithm is as follows:

```
Insert the root node in the queue.
While there are nodes left in the queue,
      Get next node in the queue.
      Print the node's value
      If the pointer to the left child of the node is not null
            Insert the left child node in the queue
      If the pointer to the right child of the node is not null
            Insert the right child node in the queue.
```

Write a function `levelOrder()` to perform a level order traversal of a binary tree. The function should take a pointer to the root node of the binary tree as an argument.

5. **Binary Tree Delete (30 pts)**

The deletion algorithm is not as straightforward as the insertion algorithm. There are three cases that are encountered when deleting an item:

a) the item is contained in a leaf node (i.e., it has no children) (Fig. 1)

b) the item is contained in a node that has one child (Fig. 2), or
c) the item is contained in a node that has two children (Fig 3 and 4).

If the item to be deleted is contained in a leaf node, the node is deleted and the pointer in the parent node is set to NULL (Fig. 1).

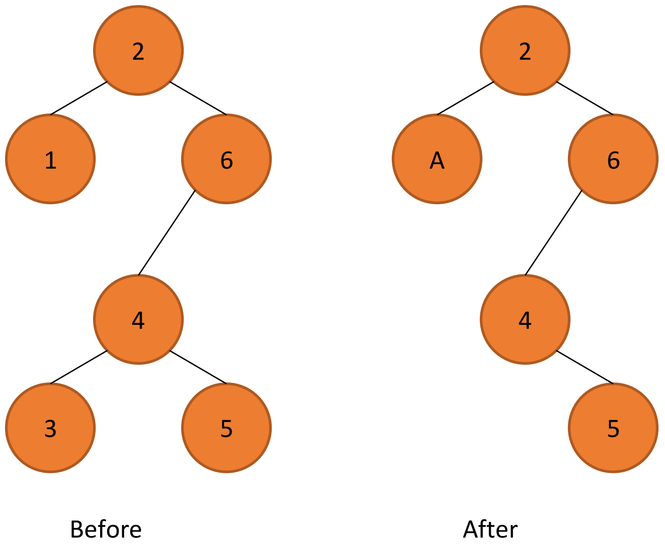


Fig. 1. Deletion of node 3.

If the item to be deleted is contained in a node with one child, the pointer in the parent node is set to point to the child node and the node containing the data item is deleted. This causes the child node to take the place of the deleted node in the tree (Fig. 2).

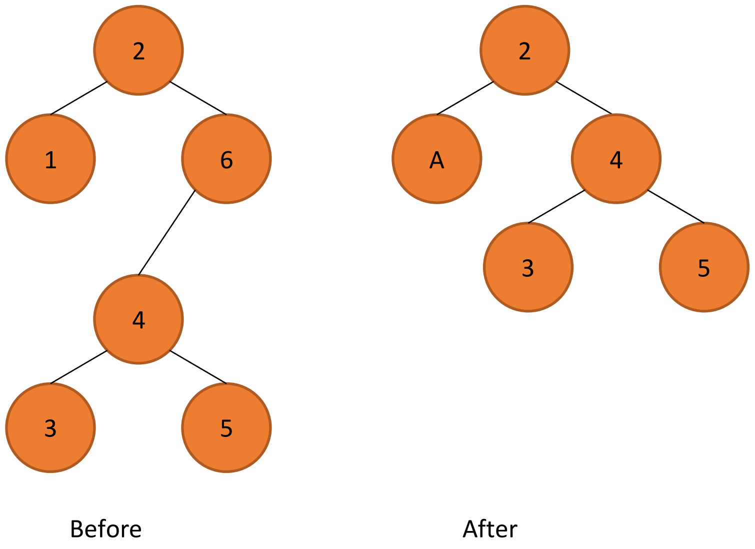


Fig. 2. Deletion of node 6.

The last case is the most difficult. When a node with two children is deleted, another node must take its place. However, the pointer in the parent node cannot simply be assigned to point to one of the children of the node to be deleted.

Which node is used as a replacement node to maintain this characteristic? Either the node containing the largest value in the tree less than the value in the node being deleted, or the node containing the smallest value in the tree greater than the value in the node being deleted. Let's consider the node with the smaller value.

In a binary search tree, the largest value less than a parent's value is located in the left subtree of the parent node and is guaranteed to be contained in the right-most node of the subtree. This node is located by walking down the left subtree to the right until the pointer to the right child of the current node is NULL. We're now pointing to the replacement node which is either a leaf node (Fig. 3) or a node with one child to its left (Fig. 4).

If the replacement node is a leaf node (Fig. 3), the steps to perform the deletion are as follows:

1. Store the pointer to the node to be deleted in a temporary pointer variable (this pointer is used to delete the dynamically allocated memory).
2. Set the pointer in the parent of the node being deleted to point to the replacement node.
3. Set the pointer in the parent of the replacement node to null.
4. Set the pointer to the right subtree in the replacement node to point to the right subtree of the node to be deleted.
5. Delete the node to which the temporary pointer variable points.

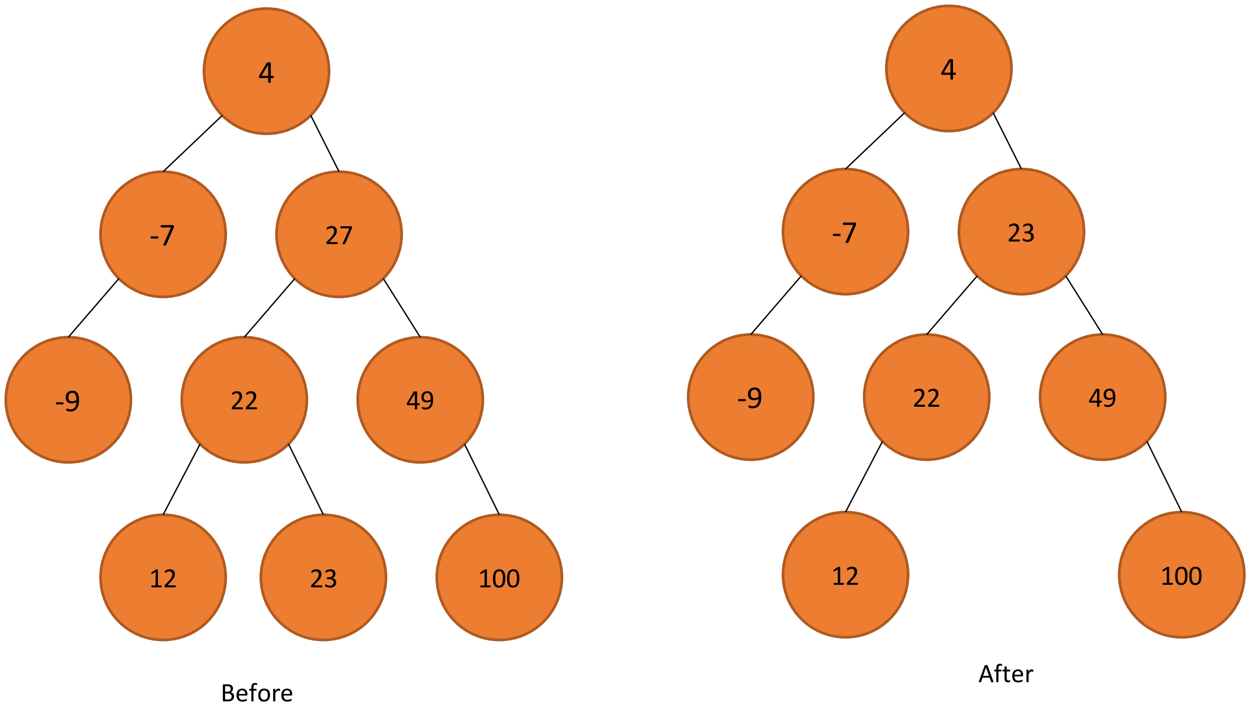
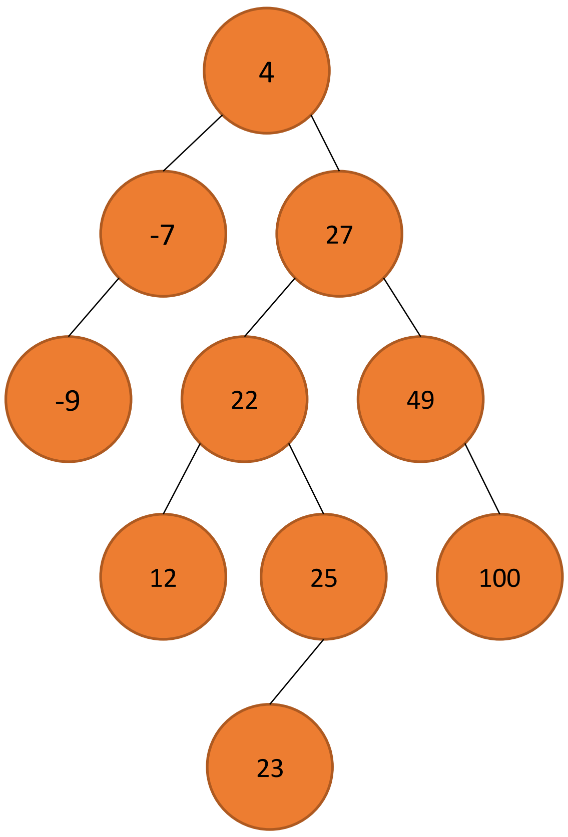


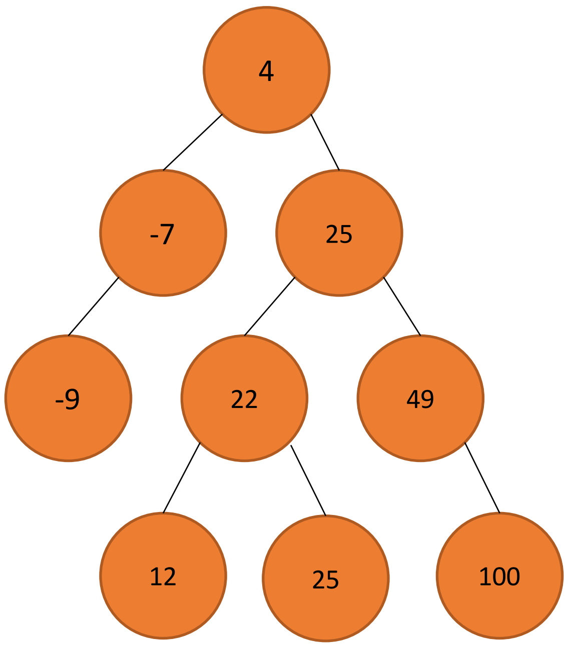
Fig. 3. Deletion of node 27. The largest value less than a parent's value (predecessor) is located in the left subtree of the parent node and is guaranteed to be contained in the right-most node of the subtree (23). Observe that 23 is a leaf node.

The deletion steps for a replacement node with a left child (Fig. 4) are similar to those for a replacement node with no children, but the algorithm also must move the child to the replacement node's position. If the replacement node is a node with a left child, the steps to perform the deletion are as follows:

```
1. Store the pointer to the node to be deleted in a
   temporary pointer variable.
2. Set the pointer in the parent of the node being
   deleted to point to the replacement node.
3. Set the pointer in the parent of the replacement node
   to point to the left child of the replacement node.
4. Set the pointer to the right subtree in the
   replacement node to point to the right subtree the
   node to be deleted.
5. Delete the node to which the temporary pointer
   variable points.
```



Before



After

Fig. 4. Deletion of node 27. The largest value less than a parent's value (predecessor) is located in the left subtree of the parent node and is guaranteed to be contained in the right-most node of the subtree (25). Observe that 25 has a left node (23).

Write function `deleteNode()` which takes as its arguments a pointer to the root node of the tree and the value to be deleted. If the value is not found in the tree, the function should print a message that indicates whether or not the value is deleted.

Homework

Submit your homework as a zip file containing the source code of each problem. Name your folder with your name, e.g. for Didier Gamboa, the file name should be:

`Didier_Gamboa`