Notes for Programming in C Lab Session #4

October 7, 2024

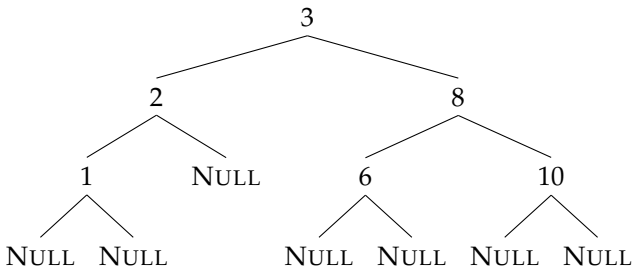
1 Introduction

The purpose of this lab session is to write some small programs that do pointer manipulations and dynamic memory allocation.

2 Overview

In this lab, you will define some functions to work with finite sets of integers, represented as (unbalanced) binary trees. Concretely, we will represent a set as a binary tree, in which each node contains an integer n, and the left subtree contains the elements which are smaller than n, and the right subtree contains the elements bigger than n.

For example, if we have the set $\{0, 1, 2, 3, 6, 8, 10\}$, we might represent it using the tree:



Note that each node of the tree contains an integer, and points to two subtrees. The subtrees can themselves be trees, or they can be the NULL value.

In C, a datatype for binary trees can be declared with the following structure declaration:

```
struct node {
  struct node *left;
  int value;
  struct node *right;
};
typedef struct node Tree;
```

This defines a type `struct node` which consists of a `left` field containing a pointer to the left subtree, and a `value` field containing an integer value, and a `right` field ointer to another `struct node`. (The typedef defines a type abbrevation `Tree` standing for the structure type `struct node`.)

A finite set is then just a pointer to this structure type. Next, you will implement a small library of functions whose prototypes and specifications are given in `list.h`, and whose implementation will go in `list.c`.

3 Instructions

1. Download the `lab4.tar.gz` file from the class website.
2. Extract the file using the command `tar xvzf lab4.tar.gz`.
3. This will extract the `lab4/` directory. Change into this directory using the `cd lab4/` command.
4. In this directory, there will be files `lab4.c`, `tree.h`, and `tree.c`.
5. There will also be a file `Makefile`, which is a build script which can be invoked by running the command `make` (without any arguments). It will automatically invoke the compiler and build the `lab4` executable.
6. Run the `lab4` executable, and see if your program works. The expected correct output is in a comment in the `lab4.c` file.

4 The Functions to Implement

4.1 Basic Exercises

The following functions should be relatively straightforward to implement. If you find yourself writing a lot of code for these functions, you should step back and rethink your approach.

- `int tree_member(int x, Tree *tree);`

  This function takes an integer `x` and a tree `tree`, and returns 0 if `x` does not occur in `tree`, and 1 if it does occur. This function should not allocate or deallocate any memory at all.

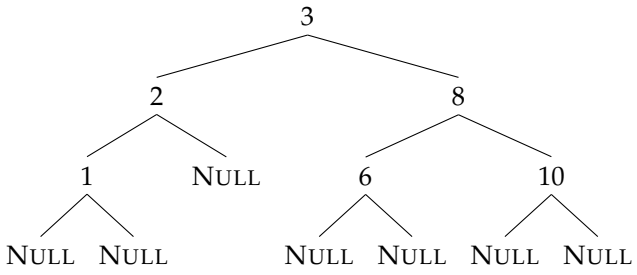
- `void tree_free(Tree *tree);`

  Given a tree `tree` as an argument, this function should free all of the memory associated with the tree. This function should recursively call `free` on each reachable node.

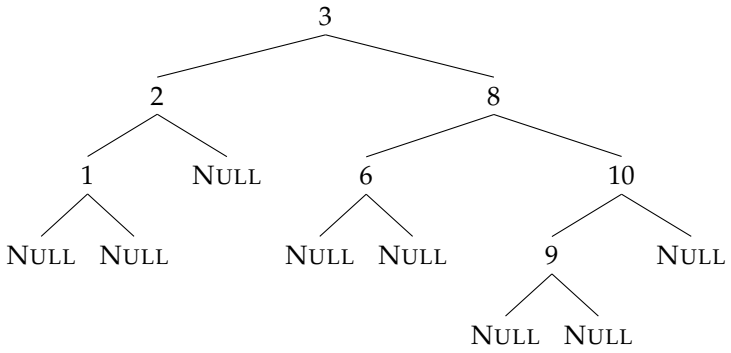
- `Tree *tree_insert(int x, Tree *tree);`

  This function should insert `x` into the tree `tree` if it is not present, and do nothing otherwise.

  As an example, inserting 9 into the following tree:



  should result in an updated tree:



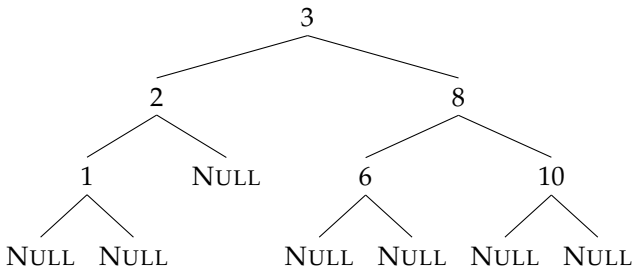
4.2 Challenge Exercises

Once you have done the basic exercises, you can try the challenge problem of *removing* an element from a set, which involves more complex pointer manipulations and pattern of memory deallocations.

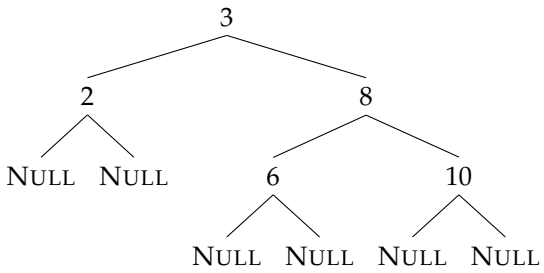
- `void pop_minimum(Tree *tree, int *min, Tree **new_tree);`

  This function should take a nonempty tree `tree` as its first argument, and then it should (a) return the minimum value held in the tree in the contents of `min`, and (b) modify `tree` so that it no longer contains `min`, returning an updated pointer in `new_tree`.

  As an example, calling `pop_minimum` on the following tree



  should return the value 1 in `min`, and modify the tree so it has the shape



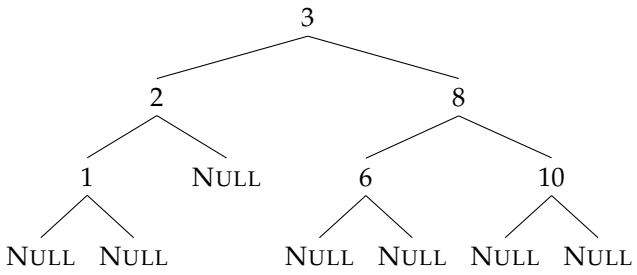
- `Tree *tree_remove(int x, Tree *tree);`

  This function should remove `x` from the tree `tree` if it is present, and do nothing otherwise.

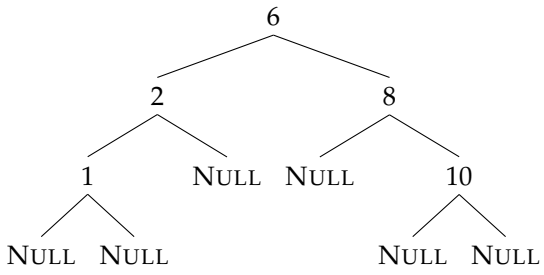
  Hints:

  1. The difficult case is when you have reached the node which needs to be removed.

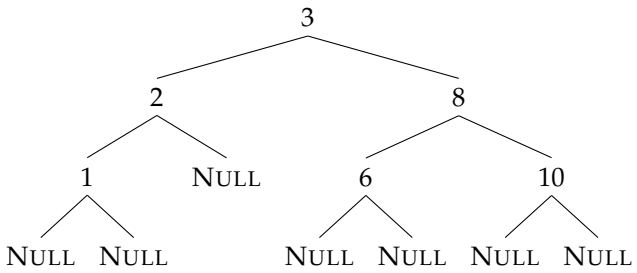
2. It will be very helpful to use `pop_minimum` as a subroutine – but remember you can only call it on nonempty trees!
3. If you remove 3 from the tree



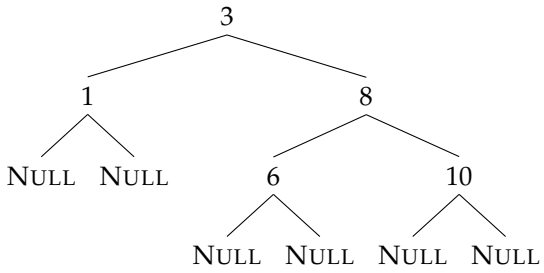
you should get



4. If you remove 2 from the tree



you should get



(Revised DJG, October 2020).