CSE 222 Programming Assignment #4

February 16, 2020

1 Introduction

In this assignment, you're going to write a program that builds a database of license plates and names using a binary search tree (BST). After reading in a file containing plate/name pairs, your program will let the user enter a license plate, and it will display the corresponding name. The user can also enter commands instead of a license plate, in order to further interact with the BST.

2 Behavior

You should run /tmp/plate4 (you can use /tmp/database.txt as a sample database) to observe the following behaviors. Your program's output and other behavior should match that sample program as closely as possible. Minor variations ("PLATE NOT FOUND" vs "Plate not found" for example) are fine, but anything other than minor cosmetic differences will cost points.

2.1 Start-up

Your executable program must be named "plate" and be run as:

- plate database_name

Your program should report an ERROR if it is not run correctly, and then return(1).

2.2 Reading the database

Once the program begins, it should open the database for reading and ingest data, one line at a time, building a BST from the data (keyed on **plate**). The data looks like this:

```
PTN-673 RICHELLE MUELLER
QFJ-481 LUDIVINA WOODWARD
RRW-885 VITO HUBBARD
```

and so on. Each line contains a license plate ID (no spaces, but make no other assumptions about the format), followed by a space, a first name, a space, a last name, and a newline (\n). If the database cannot be read, the program should report an ERROR and return(1).

2.3 User Interaction

The program then prompts the user to enter a license plate or a command. If a plate is entered, it should search the BST and either print the name associated with the plate, or print the message "Plate not found." If the user enters a command, it should execute the command. In any case, the program will repeatedly prompt the user and process their commands.

2.3.1 Legal Commands

There are two commands the user can enter:

***DUMP**

This will display information about the tree (see below)

***DELETE plate**

This will delete the given plate from the tree *using the exact algorithm discussed in class.* **Make sure you understand this algorithm (which is not the same as you'll find online): any variation from it will result in loss of points.**

2.3.2 Output Details

Normal plate information (in response to a plate look-up) should be printed as follows:

```
Enter command or plate: BLD-947
First name: ELIZ
Last name: FRANKLIN
Enter command or plate:
```

The *DUMP command should report the following information (**in this exact order and format**):

```
TREE HEIGHT: 7 (or whatever)
BALANCED: (YES or NO)

LNR TRAVERSAL:
Plate: <AOG-380>  Name: YATES, DORETTA
Plate: <BVJ-347>  Name: FINCH, SHELTON
Plate: <MOQ-693>  Name: MYERS, INDIRA
(... and so on)

NLR TRAVERSAL:
```

```
(show that data)

LRN TRAVERSAL:
(show that data)

Enter command or plate:
```

Please avoid extra output, changing the order of the output, etc.

The *DELETE command should either report “SUCCESS” or “PLATE NOT FOUND”

2.4 Exiting

The user exits the program by entering EOF (CTRL-D at the beginning of a line). This is the only way your program should exit: it should not exit in response to errors, and it should not seg fault. Before exiting, the program should free all memory that it allocated.

3 Implementation

You will implement a BST whose nodes are as follows:

```
struct node{
  char *plate;
  char *first;
  char *last;
  struct node *left;
  struct node *right;
};

typedef struct node* Node;
```

Remember that there is no sentinel associated with a BST. In your main program, your tree can be declared as

```
  Node root=NULL;
```

to represent your initial (empty) tree.

Note that you cannot assume pre-defined sizes of the struct node's strings: plate, first and last could be 2 characters, 20 characters, or anything else. Therefore, you will be using malloc to allocate memory for the nodes of the BST, and for the contents (3 strings) of those nodes.

Remember to free all malloc'd memory before exiting.

4 DETAILS

4.1 Required Functions

Implement the following functions, for use from your main program:

```
int height(Node root);      // return height of tree

int balanced(Node root);    // 1= means tree is balanced, 0 means tree is not balanced

Node add(Node root,
         char *plate,
         char *first,
         char *last); // Add a node to the tree and return the (new?) root of the tree


int search(Node root,
           char *plate,
           char *first,
           char *last); // search for plate; if found, populate first and last and return 1
                        // if not found, return 0 and don't change first or last

Node delete(Node root,
            char *plate); // search for plate, and if found, delete it
                          // return the (new?) root of the tree

void LNR(Node root); // print tree contents in LNR order (formatted as described above)
void NLR(Node root); // print tree contents in NLR order (formatted as described above)
void LRN(Node root); // print tree contents in LRN order (formatted as described above)

void treeFree(Node root);   // release all memory associated with this tree
```

5 SUGGESTIONS

Here is a suggested plan of attack for this assignment.

1. Write some code to make a tree by hand in memory (by using mallocs, setting ->plate etc to fixed strings, and assigning nodes or NULL to ->left and ->right in each node). This is cumbersome, but will serve you well in testing and developing your code.

2. Next, review LNR traversal and write code for that. This should only take about 4 lines of code. Run LNR on your test tree, and observe the beautiful output!

3. May as well code NLR and LRN, and run those. This is a good time to make sure the output format matches what is required.

4. Next make your search function. Again, this is only a few lines of code. You can test it with your pre-built tree. No need for anything fancy here: just do things like this:

```
int result;
char first[50],last[50];

result=search(root,"someplate",first,last);
if (result==1){printf("Found: first=%s last=%s\n",first,last);}
  else {printf("Not found");

result=search(root,"someotherplate",first,last);
if (result==1){printf("Found: first=%s last=%s\n",first,last);}
  else {printf("Not found");

result=search(root,"yetanotherplate",first,last);
if (result==1){printf("Found: first=%s last=%s\n",first,last);}
  else {printf("Not found");
```

(replace "someplate" etc with plates you have or don't have in your test tree).

SIDE NOTE

This is a good chance to get better with vi! Remember, you can say something like 4Y to yank (copy) 4 lines of code (starting from your current line) into the Yank Buffer (clipboard), and then type p to put (paste) those lines elsewhere in your file. w and b will let you move across a line one word at a time; ENTER moves down one line and moves you to the start of the line. cw replaces the current word with whatever you type (and leaves you in insert mode). If you're not comfortable with vi, start playing more with it today. A few hours spent messing around with it will save you time the rest of your life!

5. Now move on to your add function. Don't get fancy with input, don't try to ingest the database.txt file...just hardwire something like

```
root=add(root,"NewPlate","My","Name");
```

and get that working. You can call LNR(root) and it should print the new tree, with the new plate added in the correct spot.

6. Now initialize your tree to NULL, and see if your add function still works. This can be tricky, but once it's working, you can leave your hardwired tree and start building trees from the database file

7. use fopen/fgets/fclose to read /tmp/database.txt and build your tree from there. Now you have a large BST against which to test you traversal and search functions.

8. Code your height and balance functions. Do height first, and remember, it's only a few lines of code. Also, **don't sweat the MAX function.** I use MAX(a,b) in class as part of pseudocode, but there isn't a built-in MAX function. Don't get lost on stack overflow researching the MAX function!

How do you find the max of two values? How about an "if" statement?

```
if (a>b) {// do stuff for when a is the max} else
             {// do stuff for when b is max or when they're equal}
```

You're the programmer, and it's your world: you get to do what you want! (as long as you also do what the assignment asks for!)

9. Continuing...good time to work on your main program. Parse the command line argument; ingest the database file the user has asked for; and the read user commands and process them until fgets returns NULL. Then free memory and exit

10. Finally, work on the delete function. This is certainly the trickiest part of the assignment, and if you make a mistake in deleting, it can make it look like add, search, LNR etc are not working. So *make sure those other functions are working before you code delete.* This way, if things start to fail, you'll have confidence that it's the delete function you should look at.

Ask me questions if you are getting stuck, or aren't sure how to proceed. Take things in small steps, start with something you know how to do, and build it up into larger pieces. And have fun!

6 SUBMISSION

Submit via GITLab to a repository named "PlatePA4" Make sure I am added as a reporter. Include all source code, ".h" files and your Makefile. **Please do not include .o files or your executable file**.