Lab 1, Part 1: Stretches

Due date: April 1, at the beginning of lab period.

Lab Assignment

Assignment Preparation

This is a pair programming lab. Students who are officially registered for the course should partner only with those who are registered as well. Those on the waiting list should partner with other students from the wait list.

You are responsible for finding your teammates for this assignment, and you need to do it fast. I will assign partners for everyone who has not been able to find one.

Please note, that (a) only one submission per pair is required, and (b) the names of BOTH students must be found at the top of every file submitted.

The Task

This assignment repeats some of the things you have done in CPE 103. The goal is to let you remember Java (in case it's been some time since you wrote Java programs), and to let you remember some of the preliminaries.

For this assignment, each team will implement the Binary Search Tree abstract data type. To remind you:

> A Binary Search Tree (BST) is a binary tree data structure that stores values at each node of the tree and has the following property:
>
> - All values stored in the *left subtree* of every node N in a BST are smaller than the value stored in N. All values stored in the *right subtree* of a node N are greater than the value stored in N.

The Binary Search Tree ADT you are implementing will store `integer` values in the tree nodes.

The ADT (abstract data type) for Binary Search Trees includes the following functions:

- initialize(): creates an empty BST.
- int insert(int value): inserts value into the tree. Returns 1 if the insertion was successful and 0 if value already exists in the tree.
- int find(int value): checks if value exists in the BST. Returns 1 if it is found and 0 if it is not found.
- int delete(int value): removes the specified value from the BST. Returns 1 if the value was found and removed, 0 if the value was not found.

In addition, you will use the ADT you create to implement three functions[1]:

- void sort(int[] A): sort an array A, **and** output the elements of A in sorted order (ascending) (note, that this is actually a procedure that relies on side effects (print statements)).

  The sort() procedure must use the BST ADT implementation to achieve its goal.
- int findMax(int[] A): This function works as follows. It creates a BST for the input array and then finds the maximum element in the array and returns it.
- int findSecondMax(int[] A): This function creates a BST for the input array and then finds the second largest number in the array and returns it.

Implementation Instructions

In this course we are using Java as the programming language. **Solutions in other languages will not be accepted!** You can use any means of building Java programs you are comfortable with: text editor, simple IDE, complex IDE like Eclipse, etc.

You are allowed to consult the textbook, and/or any on-line sources you can find concerning the nature of Binary Search Tree ADT and the algorithms you need to implement for it. You can observe/use any pseudocode you can find.

HOWEVER, you are **prohibited from using existing Java implementations** of any functionality required for the lab. (This is a general requirement for the course, but since this is the first lab, let me stress it

[1] You are using Java, so you will be implementing *methods*.

specifically). All code you produce for this lab (and other labs!) **must be written by you** and noone else.

In addition to the code implementing the BST ADT and the functions listed above, you will also implement a simple *testing framework* for your implementation. The testing framework for the BST ADT shall take as input a sequence of integer numbers stored in a file and a list of commands, stored in another file. It should create a BST ADT from the values provided, and should then execute the commands and report the results of each of the command.

For testing the sort/finMax/findSecondMax functions, the testing framework, shall read the list of numbers into an array, and then execute the three functions using the array as input and report the results.

While you are welcome to use any means for creating your programs, your submission should compile and run from command line. In particular, you shall name your test frameworks `BSTTest.java` and `OPTest.java`. It should be possible to compile your code using simply

```
$ javac BSTTest.java
$ javac OPTest.java
```

commands. (I strongly prefer that you do not create packages to store the ADT implementation and then place `BSTTest.java` and `OPTest.java` outside the package, or use other structure for your submission, that requires a more involved compilation process. However, if you do need something else in order to be able to successfully compile and run your program, please submit a README file which describes the compilation process, or, even better, a shell script `compile.sh` which compiles your code, or a `make` file).

Note, that the programs will be run as follows:

```
$ java BSTTest numbers.csv commands.txt
...
$ java OPTest number.csv
```

Here, `numbers.csv` is a CSV (comma-separated values) file containing the list of numbers and `commands.txt` is a file containing a list of commands that need to be executed.

The format of the commands file is as follows:

```
command;
command;
...
command;
```

A `command` is one of the following:

```
insert(<N>)
```

```
delete(<N>)
find(<N>)
```

Each command has the obvious semantics. `<N>` is an integer value.

Example. Consider two files: `numbers.csv` and `commands.csv`.

`numbers.csv`:

```
10,3,5,19,22,34,34,27,6,3
```

`commands.txt`:

```
find(34);
find(27);
delete(34);
find(34);
delete(34);
insert(4);
find(4);
find(17);
```

Consider the following run of your `BSTTest` program:

```
$ java BSTTest numbers.csv commands.txt
```

First, your program reads the numbers from `numbers.csv` and *silently* creates the initial BST that contains all the numbers (in your case: 10, 3,5,19, 22,34,27, 6 and 3 arranged as shown in Figure **??**. Note, that only one copy of 34 is stored).

Next, your program executes each command found in `commands.txt` in turn and reports the results:

- `find(34)`: success;
- `find(27)`: success;
- `delete(34)`: success;
- `find(34)`: not found;
- `delete(34)`: not found;
- `insert(4)`: success;
- find(4): success;
- find(17): not found;

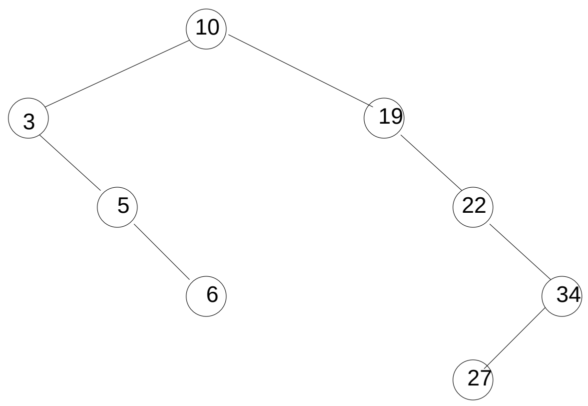

Figure 1: Sample Binary Search Tree.

Note: The `BSTTest` program, strictly speaking, does not force you to create a proper BST - you could perform all the same activities even without creating it. The sorting framework, on the other hand, will require BST. **Also**, the modifications to your program that you will do in Part 2 of the lab, will only work if you *faithfully implement the BST ADT here*.

The data for your lab is available from the Lab 1 data page:

http://www.csc.calpoly.edu/~dekhtyar/349-Spring2010/labs/lab01.html

Testing and Deliverables

You have to submit your code and instructions on how to compile (if needed) and run your program. Also, submit a README file, that, in additon to any compilation instructions, includes the names of everyone on the team. (each Java program should contain the same names in the header comment as well).

Use `handin` to submit:

```
$ handin dekhtyar-grader lab01-1 <files>
```

Good Luck!