ECE242 PROJECT 2: Contact Book

Due: October 30, 2014, 11PM on Moodle

Introduction

In this project, you will build a contact book to manage your contact information. For simplicity, the contact book will just store two attributes for a contact person: name and phone number. Your contact book needs to support several basic operations: insertion, search, update, and deletion.

Task Overview

Organizing the contact information in good data structures will support efficient operations. In this project, you should construct a name tree to store the names of your contacts. Figure 1 shows an example name tree. The phone number associated with the person can be stored in the appropriate leaf node for the person's name. Organizing the contact information in such a way has several advantages. First, it is space efficient, names beginning with the same characters can share those characters. Then, it supports efficient search. When you look up someone's phone number, you can have a more efficient search algorithm instead of scanning all the names in a linear fashion. Last, it supports search suggestion. Given a prefix of a name, you can quickly find all the names begin with the prefix.

You are given an input file which includes contact information for a number of persons. Each line contains the contact information for one person. The contact information has only two fields: name and phone number. These two fields are separated by a comma. You can assume that the name of your contacts is unique and case insensitive. The code (FileProcessor.java) for parsing the input file is given.

Build a name tree: take all the names in the input file and use them to construct a name tree. Each name is divided into characters. Each character is a node in your name tree except for the root which is associated with a fixed character '@'. Names that begin with the same characters share the same parent nodes in the tree, thus, searching a name will only involve searching a sub-tree that contains the names beginning with the same characters. The definition of the node class will be created by you. You need to figure out how to add each character into your name tree. Initially, you will add names to the tree from the input file and use sharing wherever possible. For example, if 'Chris Andersen' already exists in the tree, adding 'Chris Bosh' to the tree only involves adding 'Bosh' as a child branch of the node with the space character ' ' (see the example in Figure 1). In other words, if the beginning characters of two names are the same, they should share the same part in the tree which contains those characters.

After the name tree is built, you should be able to provide several basic operations of the contact book, such as searching a person by name, updating one's phone number, adding a new person, and deleting a person.

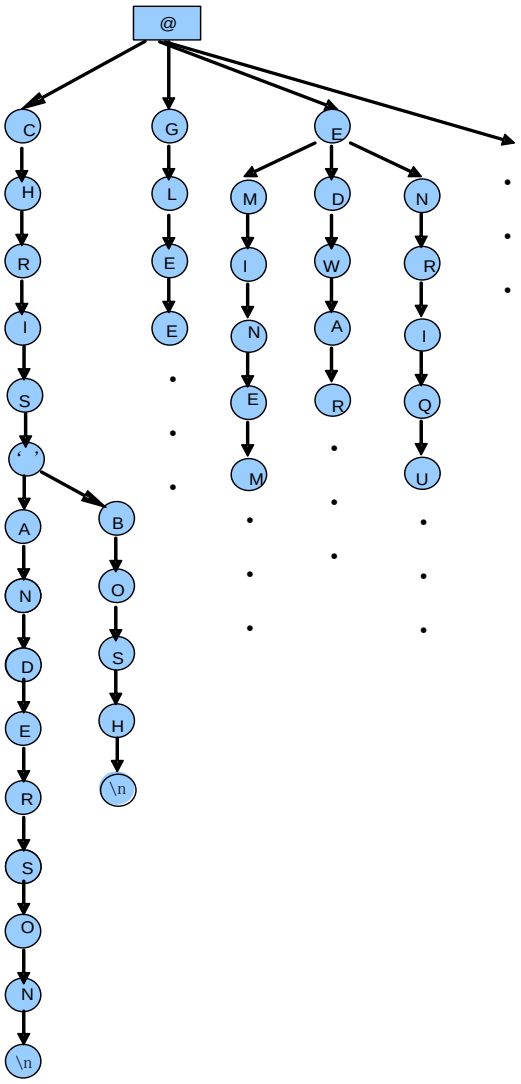

Figure 1: An example of name tree

More specifically, you will perform the following tasks.

1. Read in the input file to build a name tree. To do that, your code should include an *add person* method. For each person, you call this method once to add his/her contact information to the name tree.

2. Implement a *search person* method to find phone number by name. For an existing name in the name tree, this method returns the corresponding phone number. If the name does not exist, it returns null.

3. Implement an *update phone number* method to update one's phone number. For a name in the name tree, this method replaces the corresponding phone number with the given phone number. If the name does not exist, it does nothing.

4. Implement a *delete person* method to delete a person by name. For an existing name, this method deletes the name and the corresponding phone number from the name tree. If the name does not exist, it does nothing.

Getting started

The best way to complete any programming assignment (including this one) is to take tasks step-by-step. The first decision you need to make is the number of classes you will write. For this assignment, you should have at least four classes, including the tree node class, tree operation class (which should include all the four methods mentioned above), file processor class (which is given) and a driver class. Please do not place all the methods for your entire project in one class.

Required Output

1. After building the name tree with the input file, print out the phone number for 'Chris Bosh'. You should call your *search person* method to find the phone number.

2. Print out the phone number for 'Chris Bosh' again (by calling your *search person* method) after the following operation is performed: update the phone number for 'Chris Bosh' to be '413-111-2222' by calling your *update phone number* method.

3. Print out the phone number for 'Chris Andersen' (by calling your *search person* method) after the following operation is performed: delete 'Chris Bosh' by calling your *delete person* method.

   NOTE: make sure your methods are not limited to these two names. We may use other names to test.

Hints and suggestions

Successfully completing the project and achieving a good grade requires completing the project as described above and clearly commenting the code. As always, it makes sense to start the project early. Build your project code step by step.

- You must be clear about what is its argument for each method you need to implement. For example, the argument of the *add person* method may contain name (String), phone number (String), and root tree node.

- All the tree nodes in the name tree should be objects of the same tree node class (e.g, TreeNode). See the given FileProcessor.java file for more hints about the tree node class.

- You may need to covert a string (e.g., a name) into a char array. Java has the method to do this: toCharArray() in the String class. Since names are case insensitive, you might want to convert a name to upper case (or lower case) for easy comparison, before converting it into a char array. The toUpperCase() method in the String class can do this.

- When your *delete person* method is called, it should only try to delete the specified person (by name) only. That is, it should not delete the common nodes (if any) shared with other name(s) in the name tree. Thus, one way to implement this method is to first check whether the specified person exists. If yes, find the first tree node (on the path forming the name) that is only owned by the specified person, and delete it (by setting the corresponding pointer in its parent node to be null).

What to submit:

Please submit all .java files for your project and the screenshots for all the printout results. All code should be well commented.

Reminder: The course honesty policy requires you to write all code yourself, except for the code that we give to you. Your submitted code will be compared with all other submitted code for the course to identify similarities. Note that our checking program is not confused by changed variable or method names.

Grading

- Code works (50%)
- Comments (20%)
- Program structure (20%)
- Readability (10%)