CS310: Advanced Data Structures and Algorithmns

Fall 2018 Programming Assignment 1

Due: Wednesday, Oct. 10, 2018 before midnight

Goals

This assignment aims to help you:

- Learn about hash tables
- Review packages
- Start thinking about performance and memory use

Preliminaries

Reading

- S&W Chapter 3.4 (hashing)
- S&W code instructions http://algs4.cs.princeton.edu/code/ .
- Java documentation about packages: https://docs.oracle.com/javase/tutorial/java/package/packages.html.

Installation and compilation

- Create a directory named pa1 (small letters) under your cs310 directory. Remember that Unix is case sensitive!
- Create the subdirectories classes and src, each with cs310 subdirectories for the package files. Here is an illustration of the class hierarchy:

  ```
  │
  ├classes
  │   └cs310
  │
  └──src
      └cs310
  ```

  The source files you create for this assignment should reside in a package called cs310. When creating a package, all the source files should reside in a directory whose name is the same as the package name, and the java source files should have a the following declaration at the top:

  `package cs310;`

- Download the S&W code in a jar file. See here: http://algs4.cs.princeton.edu/code/ . At the bottom of the page there are installation and running instructions. Create a lib directory under pa1, and copy the algs4.jar library there.

- To compile the code you first change to the src directory:

  ```
  cd src (the top-level source directory)
  javac -cp .:../lib/algs4.jar -d ../classes cs310/*.java
  ```

  The -d flag redirects the class files to the ../classes subdirectory. The -cp flag tells the compiler to load the jar file during compilation. If you omit this part, it will not compile since your source will not find the S& W classes it needs. Notice that on Windows you should put a ; (semi-colon) instead of a : (colon).

Questions

1. In class we discussed deleting from a hash table that uses linear probing. The text approaches the problem by deleting the item and moving all the subsequent cluster of hash elements one position up. This is necessary in order to not disconnect a collision chain, but it may harm the performance, especially if there are large clusters.

   An alternative way to delete from a hash table is as follows: Every item in the table has an extra boolean flag, active or not active. When an item is "deleted" it is not physically removed from the table, but rather, its active flag is set to false. For this to work, several other things have to be modified from the original implementation:

   - When searching for an item in the table, you should return it only if its active flag is set to true.
   - When calculating the table size for rehash, you have to take into account "deleted" items as well, since they are in the table and take up space.
   - When rehashing, you obviously don't need the deleted objects anymore, and therefore you insert only active objects into the new table.

   Implement this alternative deletion. Do the following:

   (a) Start from S&W's implementation of hash with linear probing at:
   http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/LinearProbingHashST.java.

   (b) Copy it over to your cs310 package/subdirectory, change the package declaration on top to cs310, and change the class name (and file name, of course), to LinearProbingHashST2.java.

   (c) Notice that since it's a different package now, you may have to import some classes from the algs4 library.

   (d) Make the changes noted above. Notice that it's not only the delete function that you should change.

   (e) Test the delete function: In the main function, have the LinearProbingHashST2 read a small file, tinyST.txt which you can find at http://algs4.cs.princeton.edu/34hash/tinyST.txt . You can use the LinearProbingHashST's main function as a starting point: It reads a file string by string and puts the <string, index> pair in the table (in this case every string is just one character).

   (f) Print out all the characters, then delete "S" and "A" and print again.

   (g) Have the main function print out the size() of the table and the values of the n and m fields (as appear in the original source file) before and after deletion.
   Usage: `java -cp .:../lib/algs4.jar cs310.LinearProbingHashST2 < tinyST.txt`

2. Performance of a hash table: Compare the performance of the S&W lookup tables: linear probing (LinearProbingHashST), separate chaining (SeparateChainingHashST), sorted symbol table (ST), and a sequential search (SequentialSearchST), which is just an array list. Use the Linear Probing implementation provided by S&T, not the one you defined in the previous question. Don't use Java's standard hash tables. You will create a word counter, reading in a long document and map each word to its number of occurrences in the text. Specifically, do the following:

   (a) Create a class TestPerf in the cs310 package.

(b) In your class, define four lookup tables as described above. Each one will have <String, Integer> as key and value types.

(c) You can use the In class from the S&W to parse an entire text file into an array of Strings using the function readAllStrings. For simplicity, assume that every two words are separated by a whitespace, and don't worry about capital and small letters, punctuations, quotations etc. It means that for example, "word" and "word;" will be considered two different words, but that's ok for the sake of this assignment.

(d) Use the file http://introcs.cs.princeton.edu/java/data/tale.txt. It is also attached as a handout. It is probably big enough to show the differences in performance on most PCs and laptops.

(e) Word counter: Every time you read a word, you should search it in the table. If it is not there, add it with a value of 1 (because it's the first time you see it). If it is in the table, just increment its value by 1.

(f) To compare the performance, you should time the above word counter. The simplest way to measure the time is to use System.currentTimeMillis() before and after you read the file into each table, and subtract the end time from the start time. Have your function return the result. It will be in milliseconds, and it is a long, not an int.

(g) Have the main function take the file name as a command line parameter (NOT read from the standard input), so the usage is :

```
java -cp .:../lib/algs4.jar cs310.testPerf tale.txt
```

I don't believe I still have to say it, but tale.txt should reside in your running directory for this to work. If it isn't, specify its relative or absolute path. Notice that during runtime you should also specify the classpath as you did during compilation to avoid a runtime error.

(h) Notice that S&W don't believe in inheritance, which may force you to duplicate some parts of your code. It is annoying, I know.

(i) SequentialSearchST is really slow (by now you should not be surprised). Just be patient.

The memo.txt file

Create a memo.txt file in your pa1 directory. It's a plain text file with your name and student ID. If you took late days, specify it clearly too. Also, answer the following questions:

1. For the alternative deletion (question 1): How many items does the table physically have before and after deletion? How many "valid" items does the table physically have before and after deletion?

2. For question 2: How many ms did every table take?

3. Does it (approximately) correspond to the expected insert and search performance for each table? No need to perform a detailed analysis, but please mention what you know about the expected runtime for each one of the lookup table types.

4. How many words in total does the file tale.txt have? You can figure it out using `wc -w tale.txt` . wc is the word count unix command. Again, don't worry about punctuations, spaces etc.

5. How many unique words does the file have? For this, you'll need to examine one of your tables. Hint: You can print out the table from one of your tests – strings and frequency, into a file, one entry per line, and see how many lines it has. Again, you can use the wordcount command in unix: `wc -l <file name>`.

6. What is the most common word and how many times does it appear? You can use the file you created in the previous question. Make sure you print into it not only the words, but the frequencies as well. To find the most common word you can sort by frequency. The Unix command `sort -n -k 2 <file name>` can help, or you can use excel if you wish. The Unix command sorts an input file, the `-n` flag tells it to sort numerically (the default is lexicographic), and the `-k 2` flag sorts by the second field, which is the frequencies.

Delivery

Before the due date, assemble files in the pa1 subdirectory of your provided cs310 directory on our UNIX site. Remember again that UNIX is case sensitive, so the directory name has to be in small letters. This is important since the TA uses automated scripts. Make sure the sources compile and run on UNIX! They should, since Java is very portable. It's mainly a test that the file transfer worked OK and that you know how to compile and run from a command line.

- pa1/memo.txt (plain txt file, try "`more memo.txt`" on UNIX)
- pa1/src/cs310/TestPerf.java
- pa1/src/cs310/LinearProbingHashST2.java
- pa1/lib/algs4.jar

Check your filenames: login on UNIX: cd cs310/pa1, then "ls" should show memo.txt, src, classes. "`ls src/cs310`" should show all the source filenames. Remember that case counts on UNIX!