TCSS 342B Fall 2004

Word Counting Project, Additional notes version 1.0

Project #3, (Homework #6)

This document contains additional hints on the project, including dealing with stack overflow and HashTable implementation hints.

StackOverflow notes:
Apparently, the standard Sun Java Virtual Machine in Windows will give you stack overflow errors if you run DictionaryApp on some of the sorted files with the binary search tree option. And I do not know of a quick fix to get it working under windows. Let me know if you know of one. This is relevant mostly for part 2 of the assignment, where you need to report timing results.

Here are your options, for reporting binary search tree results on the sorted files (in part 2). Choose one of them; choice 1 is the preferred choice.

1. Run your program on a non-Windows computer. It has been verified to work under Mac and under Linux. If you don't know Linux, now's a good a time as any to learn it the basics of how to use it. For Linux, you can use cssgate.tacoma.washington.edu. Your login name/password is the same as that of your CSS lab accounts.
2. Explain why your program crashes (when running under windows) for larger inputs. Based on your timing results for smaller inputs, estimate what the timing results should be for the larger inputs, and explain how you estimated.
3. Download and install Excelsior JET for your windows computer, and use it for all your timing results (available at http://www.excelsior-usa.com/jet.html). jet is a program that takes java class files and libraries, and compiles them into .exe files. It does some sort of optimization. I believe DictionaryApp using binary trees should work after optimization and converting into .exe files. You will have to register (it's free) to download a free trial copy.

In case you would like to use linux to run DictionaryApp, here's a short intro: Run an ssh program to logon to a unix machine. There are many ssh programs available for free; As a student, you can also download one as part of UWICK, UW's internet connectivity package. All lab computers have some ssh program installed. Then login to cssgate.tacoma.washington.edu. Your login name and password is the same as the one in the CSS lab. This means your login name is your uwnetid; your password may not be the same as the password for checking uw email). Next, copy all your .java and .txt files from your computer to cssgate. This is most easily done with a secure file transfer program. Secure file transfers run on top of the ssh communication protocol, so most ssh programs can also send files.
Use the command "`javac *.java`" to compile your programs. Use "`java DictionaryApp -b w-sorted.txt`" to run DictionaryApp on file w-sorted.txt using binary trees.

There are many other unix commands that would be useful for you to learn, like

`rm` : removes files
`ls` : lists files in the current directory
`cd` ; change directory
`cp` : copy files.

Implementing Hashing

You are required to implement a quadratic probing hash table that dynamically expands when it gets too close to full. Your hash table needs to support the SimpleMap interface. The default initial capacity of the table should be 11, and the default load factor should be 0.5. When the hash table needs to expand, you should at least double the capacity. The capacity should always remain a prime number. You should use the standard Java method hashCode() for computing a hash code. Beware that hashCode() values can be negative, and that when x is negative and y is positive, x mod y returns a negative number between –y+1 and 0 (inclusive).

One additional starter file for hash tables (QuadraticProbingHashTableMap.java) is now the website. The wordcount.zip file has been updated to include the new file and this document. You may complete this file to get a working quadratic probing hash table. The only part implemented for you is the computation of the new hash table size when expanding the table. You will need to create a HashEntry class; this class represents one entry of the HashTable. Alternatively, you may implement the hash table completely on your own.

Optional extra credit: If you would like a few points of extra credit, than you may explore the performance of hash tables using different initial capacities, different maximum load factors, and/or different hashCode generation methods. You may explore the performance of both your quadratic probing hash table and that of Java's chaining implementation of a hash table. Come up with interesting questions to ask, and answer them in your report. Some sample questions include: Which hash table implementation is more dependent on the load factor, the chaining implementation or the quadratic probing implementation? At what load factor does the performance of hash tables become unacceptably bad? Does this depend on the type of input? Which of the different hashcode() methods described in the book appears to work better in practice?