Adarsh Solanki as5nr
3/13/12
CS 2150: Lab 6
"inlab6.pdf"

*Note: All times of computation are based on the running time experienced on a Mid 2011 MacBook Air (1.7Ghz i5, 4GB 1333 MHz DDR3, Mac OS X Lion 10.7.3)#*

**Did your implementation produce the correct results? Did you have to reformat your output?**

My implementation did produce the correct results, but the order and some of the formatting of the trivial cout statements were different than the ideal output. I changed the wording of my "Found # of words" statement as well as removed a debugging cout statement that prints out the current word being searched for in the hashTable.

Also, for some of the output files, the total number of words found was printing an erroneous value. I realized that I was not properly initializing the value of the numOfLines integer that incremented upon each successful hashTable::contains() call. Once the variable was initialized at 0, the program ran smoothly.

**How much faster was your program with the –O2 flag?**

Without the –O2 flag, the program took 2.12041 seconds to run with a dictionary of "words2.txt" and a grid of "140x70.grid.txt".

Once the optimization flag was turned on, the program completed in 1.28113 seconds with the same input files.

This is a almost a 50% increase in speed (the optimized version runs in almost half of the original time) which is a huge decrease! This enlightened me to the wonderful abilities of compilers-automated optimization, and compilers in general.

**What was the speed of your implementation?**

My a.out file completed the 250x250 grid with words.txt as a dictionary file in 37.05 seconds! This is a pretty slow computation time!

For the 300x300 grid with words2.txt, the program completed in 51.9877 seconds.

**What is the Big-Theta running complexity?**

My program runs at a Big-Theta time complexity of (R*C*W) since the length of words can be factored out as a constant.

**What problems did you encounter?**

I had a lot of trouble getting my computations to run in a reasonable amount of time. I eventually optimized my nested for-loops by including more if-statements for special cases in order to bypass extraneous calculations. For example, if the length was less than expected (the word was cut off by boundaries of the grid) then the for loops break immediately. Optimizations like this can help minimize repetition of superfluous and complex operations.

**Shell Scripting**

Shell scripting turned out to be very strange. Bash as a language seems like a very stripped-down language, for example only one arithmetic function can be used in each `expr` clause, increasing the need for multiple instructions. This is reasonable, because the design of bash and other scripting languages was not for arithmetic computation, but rather for file-system tasks, and for that it excels.

I especially liked the ease with which one can utilize shell scripts within shell scripts, along with any other command-line programs, in order to automate user-interaction with a shell. The weakly typed variables were reasonable given the uses of the language. There shouldn't be too many different types being used in the first place in a bash script, so a typeless 'var' type works well.