Julia Shea

CS 2150

Oct. 24th, 2019

Post-lab 6

Big-Theta Analysis:

The big-theta runtime complexity for my program is a function of the number of rows and columns in the grid that the nested for loops must iterate through. The number of words in the dictionary is only relevant when inserting into my hash table, so for the word search portion of the assignment, words (*w*) should not be included in the calculation. The for loops run in a linear fashion through rows (r) and columns (c), so this gives us a runtime complexity of r*c. I am not including the number of directions in this analysis (8) or the word length (some small constant), because these are insignificant constants that can be factored out for purpose of overall big-theta analysis.

Running time:

The code I wrote for the pre-lab ran the 300x300 grid word puzzle with words2.txt in an average time of .879 seconds–for the purposes of the post-lab, I found it relatively difficult to significantly reduce this time. I ran my code on a VM (Virtual Box Linux Ubuntu) using a macOS as my host machine. I used the separate chaining collision resolution technique in the prelab, and because my code ran fast, I decided not to change my technique and instead attempted to decrease running time using a different hash function and a load factor. I eventually got my code to run in an average time of .721 seconds. This was done by increasing the number of buckets in my hash table to the number of inputs*5, instead of the number of inputs*2. I tried

several different hash functions to better spread out the number of inputs, but I couldn't find one that caused my program to run faster than the one I used for the prelab, which was the summation of all the characters in the string % the table size. Even the third hash function given on the slides, which was (∑k−1i=0si∗37i) caused my program to run in ~.9 seconds, which was slower than the original prelab function. I also tried to modify the data structure of my hash table and make it an array of lists instead of a vector of lists, but this had no measurable effect on my runtime. All in all, I finished this lab with an improvement of 1.26x my original un-optimized time.

NOTE: I sincerely hope that the graders of this lab will understand that I tried all the techniques I saw fit, despite the lack of significant improvement in runtime.

Manipulating Running time via Hash Functions:

NOTE: all times listed are using the 300x300 grid/words2.txt and 250x250 grid/words.txt respectively

1. Original running time of my application (after post-lab optimization): 0.721 seconds and 2.85 seconds.
2. Running time with "worse" hash function: 57.311 seconds and 299.897 seconds.
    a. I used the following hash function: s0 %table size. This caused such a significant increase in running time because the words ended up mapping to the same 26 buckets in the hash table, and iterating through them all was incredibly difficult and slow.
3. Running time with "worse" hash table size: 1.29 seconds and 3.72 seconds.

a. I decreased the hash table size to half the number of inputs (words in the dictionary), because this would cause fewer possible buckets for the values to be placed in, and would therefore slow down lookup times for the word search. This caused minor, yet measurable, differences in runtime.