Lab 6 Post-Lab Report

Name: Peter Haynes
Student id: prh7yc
Date: 10/22/19
Section: Tuesdays 12:30

Run-Time

While the optimizations I made did increase the actual run-time of my program, they did not affect the asymptotic run time of my program. Regardless of what optimizations I make, the program will still have to run through every row, every column, and search through the dictionary to find all the words. While my find function only runs for lengths greater than 3 and lengths less than or equal to the maximum column or row length (or 23 whichever is lower), since the runtime increases for every word added to the dictionary it must factor in to the asymptotic runtime. This element can be factored into the runtime using a coefficient of less than one to account for the fact that it doesn't run in every loop but only most loops. Ultimately, the asymptotic runtime is represented by r x c x w because for every row of the program, the search must run through every column of the program. Additionally, for every row and combination, the program must run through the dictionary using the find function which has worst case runtime of n because the elements are entered as a list. Lastly, since the directions factor will always be 8 and the length factor will always at worst be 23, these two multipliers can be treated as constants and do not factor into the asymptotic runtime.

Time Reports

Original 300x300 time: 6.00535 seconds

After optimizations time: 2.11699 seconds

Using a worse hash function: 4.94338 seconds - to make this worse hash function I raised my current value of 37^i to the ith power which made my computer do more, larger calculations ultimately slowing the runtime of the program.

Using a worse hash table: 2.50244 seconds - I switched the hash table size from 5 times the number of elements to half the number of elements which forced more collisions and thus higher run times for find.

Optimizations

The first optimization I made was changing my hash function. While I previously multiplied every letter by 37^i to get the hash value, I instead created a vector of the first 23 powers of 37 to multiply the letter values by. By doing this, I greatly reduced the number of calculations required for my hash function as previously my program was constantly multiplying 37 by itself 275 times for word lengths of 23 and smaller numbers of times for every length up to that point. After making this change, the number of calculations was cut down to the number of characters in the string thus significantly reducing how much work my computer has to do for each run. This alone took my running time from 6.00535 seconds seconds to 2.60013 seconds - a vast improvement making the program over twice as fast. Next, I increased my hash table size from 2/3rds the number of elements to 4/3rds the number of elements. The increased space in the hash table created a new load factor of 0.75 which reduced the number of collisions when inserting. While the insertion does not factor into runtime, because less collisions means shorter lists, this reduced the runtime of find. Since the find function is one of the elements impacting my runtime, reducing the time this function takes took the runtime to 2.49895 seconds. While this change doesn't change the runtime asymptotically, it did affect the program in practice. Next, rather than printing every element as I found it, I saved these elements to a new vector and printed them out after all the words had already been found. I expected this to greatly decrease the amount of time my program takes given the note given in the postlab section, however, in practice it didn't make much of a difference taking my time down to 2.32509 seconds. To implement this change I converted the word to the correct formatted string and saved it to a vector after finding it, then printed out every element of the vector after the timer had stopped. I'm not sure if there is a more efficient way to implement this that I am missing, or I the change ultimately just doesn't make that much of a difference, but 0.2 seconds is a good enough increase to be tangible improvement. Furthermore, I once again changed the hashtable size from 4/3rds the number of elements to 5 times the number of elements which gave my final runtime of 2.11699 seconds. This would not be a practical solution in the real world as it requires much larger amounts of memory to create the hash table. While this may be ok in some situations to require more memory, the increase in performance wasn't significant enough for it to be a viable solution (at least in my opinion). Lastly, I tried a different collision system changing from separate chaining to linear probing. To implement this, I first changed my hash table from a vector of lists to a vector of strings. After doing this, I changed my insert function to check the hash index of each word and if it's full add 1 to the key until it finds an index that is not full. Lastly, I changed my find function to perform a similar process but return true if the word was found rather than looking for an empty index. This was a huge mistake. I'm not entirely sure why this made such a difference on my program, but after successfully implementing linear probing, my runtime went up to 158.208 seconds for a 300x300 grid - a full 7474% increase. Armed with this information, I assumed that quadratic probing and double hashing would produce similarly disastrous results. So, rather than implementing these changes and testing, I accepted my runtime of 2.11699 and returned to separate chaining to resolve collisions. Ultimately, my optimizations produced a speedup of 2.8367.