John Fishbein - jhf5my 5:00pm lab Thursday, March 7, 2019

Post Lab 6 - Hashing

After implementing the hash table, and then attempting to optimize the word search, the big-theta complexity of the word search algorithm is in $\Theta(r * c)$ where r is the number of rows and c is the number of columns in the grid. In the quad-nested for loop, where the word search logic is performed, the loop through the rows and the loop through the columns are the only variables that impact the run time of the algorithm. The other two loops, through directions and word lengths, are insignificant in large inputs compared to the rows and columns. The other variable w, the number of words, also does not impact the run time because the hash-table look up time(the find method) theoretically should be constant. There are cases when collisions occur and find() is not exactly constant time but it is close so the number of words in the hash table does not impact complexity.

Before my optimizations, the word search performed with words2.txt on the 300x300 grid using my Mac Machine took around 30 seconds to run. After my optimizations, these same files took, about 2.5 seconds. If I were to pick a new hash function to make the process worse in terms of time, I could hash the string to the index corresponding to its length. This would drastically increase the run time because there would be a huge amount of collisions given that a lot of the words have the same size. Similarly, if I were to change the hash table size to a size much closer to the number of words inserted in it, the performance would again go down. This is because, again, the number of collisions would increase simply due to the fact that there are less spaces to put words in the hash table.

After all of my optimizations, the overall speedup that resulted was roughly 12.0 for the 300x300 grid searched using words2.txt but the speedup for the 250x250 grid on words.txt was about 125.5. These improvements were a result of several trial and error attempts to improve all aspects of my hashTable implementation. The first thing I tried to do was improve my hash function. Originally, I implemented the hash function shown in class that summed the value of each character in a string multiplied by 37 and raised to the power of the index of that character. I tested several different ideas including modifying my original loop, changing the operations I performed in my summation, and changing the data type of string to c style string. These improved my run time slightly but none resulted in drastic changes. Next, I tried to implement rehashing. This worked very well, and accounts for most of the time I was able to

improve by. Anytime I hashed a string and a collision occurred, I hashed the string again, using a completely different hash function. In the event that there was a collision after the rehash, I used linear probing to take care of the rest. This allowed me to use two of the hash functions I had experimented with in my first attempt at optimization. After that, I experimented with quadratic probing as a replacement for linear probing. I found that, after the hash and rehash, my runtime either stayed the same or slightly increased with quadratic probing so I decided to stick with linear probing. These changes brought my final run time for both the 250x250 grid on words.txt and the 300x300 grid on words2.txt to about the same time around 2-2.5 seconds.