Name:Shahroz Imtiaz

Email ID:si6rf

File Name: postlab6.pdf

Date:10/25/2018

1. Big-Theta Running time

Since we have four loops, once would assume the big-theta is O(n^4), but the two loops of the for loops rely on constants (direction and word length), so in terms of big theta, we have O(R*C). With the creating of the hash table, the big-theta is O(R*C+W) for the application, but that turns into O(R*C) for the word search.

2. Hashing

All of the results below are calculated for words2.txt and the 300x300.grid.txt using my MacBook Air. My original application found all of the words on an average of .787732 seconds. To make my hashing worse using my hash function, I simply just multiplied my original hash function by 0. This hashed all of the values to the 0 index of my vector. This increased the time need to find all of the words to 43.7173 seconds! When all of the words were hashed to the same value of zero, my program had to linearly go through my words and see if the words it was looking at matched the word that the word search was looking for. To make my hashing worse using my hash table size, I made my table size 6. Not only is 6 not prime, it's also a low value. This implementation caused more collusions and required my program take 9.5685 seconds to find all of the words.

3. Optimization

When I first started my lab, it took my program over 20 mins to find all of the words for words.txt and the 250x250.grid.txt! After some debugging, I quickly realized that the problem was with my hash function. For my hash function, I was originally using the method pow() from the math class. Not only was this slowing down my program because the method pow() is really slow at calculating the result for large values, but I was also overflowing and hashing to "0" a lot.

The biggest change by far in terms of speeding up my code was removing the pow() method. I decided that just adding the values of each char in the word string was much better than using the pow() method. After making the change to my hash function, my program found all of the words for words.txt and the 250x250.grid.txt in 12 seconds. I was really amazed by how one little change can completely change the time it takes for you program to finish! Feeling more confident about my code, I decided to see what other optimizations I could make.

After reading what was recommended for optimization on the class website, I decided to try using the -O2 flag to optimize my code. After adding the -O2 flag to my makefile, I quickly noticed the change in time. My program now took under 4 seconds to find all of the words for words.txt and the 250x250.grid.txt. After reading some more tips on the class website for optimizing my code, I decided that printing out the word after I found a word was taking a lot of time and it would be much more effective to store the words in a linked list and print the list out after the timer had stopped.

Storing the words, rather than immediately printing them out, knocked off some more time from what it took to finish finding all of the words. My program now took around 3 seconds to find all of the words for words.txt and the 250x250.grid.txt. Next, I decided to take another look at what I could do to improve my hash function. After reviewing the slides from lecture and researching on the internet, I decided to multiply my hash function by a prime number. By multiply my hash function by a prime number, I ended up getting less collusions because the hash values were more unique. I also took a look at the for loop I was using to traverse my string, and I decided that if I look at ever other letter in my string, I would actually save time because now my for loop took O(log(n)) to finish instead of O(n) like it did originally, and doing this wouldn't increase my collisions by a lot. After doing all of this, my program took around 1.3 seconds to find all of the words for words.txt and the 250x250.grid.txt.

I then started thinking about my hash table's size and if I would benefit from doubling my hash table's size in the beginning, before I inserted all of my words. And since we were concerned with speed and not how much memory the program takes up, I decided that by doubling my hash table's size in the beginning, I would have less collusions and thus my program would take less time to find all of the words. So, I doubled the size of my hash table, checked if it was a prime number, and if it wasn't, I found the next highest prime number.

Finally, I checked to see what the largest size of each word was in both dictionaries. The largest word in words.txt was 22 and the largest word in words.txt was 16. With this new knowledge, I updated my inner most loop in my wordPuzzle.cpp file to stop once it went over the highest largest length of the word in the dictionary used. Doing this didn't improve the results for when my program used words.txt since my original code for the inner most for loop

stopped after it went over 22, but it did improve the speed of my program for when the dictionary used was words2.txt because the largest word in that dictionary was 16 not 22.

Throughout the process, there was a lot of success and failure, such as seeing if a list of lists would be faster than a vector of lists, but trying different possible optimizations led from my program finding all of the words for words2.txt and the 300x300.grid.txt in 6.67819 seconds to find all of the words in .787732 seconds. This was a speed-up of 8.477!