Pyung Lee

Pkl4fr

3/6/2020

Big Theta:

For my application, the big theta run time is n^3. The main component of the program that searches for words has 4 for loops nested in each other. However since the maximum word size is a small constant, relative to the rows, columns, and number of words, it is negligible when accounting for the big theta run time. If each of the for loops takes a running time of n, r*c*w would be n cubed.

Timing Results for words2.txt and 300x300.grid.txt

I ran this program on a HP Spectre with Intel Core i7-8500U CPU. These were my results for the grids from 50x50 to 300x300 grid sizes.

	words.txt	words2.txt
50x50	5.88705	3.638
140x70	28.2813	15.4092
250x250	192.462	112.788
300x300	286.23	170.256

Post-Optimizations

All the times are in seconds. This was the final runtimes with all of my optimizations. Unfortunately I was not able to get the running times listed on the txt.out documents. I believe that it had something to do with the implementation of the hashtable or the hashfunction. I tried to implement different hashfunctions from the most simple to the complicated (with varying prime numbers) but the runtimes were very similar.

Slow Hash Function

The “slow hash function” I used was taking the first letter of the string given and modding it by the table size. For a 300x300 grid using the words.2.txt I got a running time of 258.132 seconds. This function was slower because it only uses the first letter of the string to create a hash key so it is more prone to collisions with other words with the same starting letter.

Slow Table size

My “optimized” table size I used was 37571. For this optimization I used 5851, a smaller prime number hoping my runtime for 250x250 grid using words2.txt would be faster. I was surprised to see that compared to my “optimized” program, it was a whopping 1 second faster with a runtime of 112 seconds. I have to assume that it would be a similar time because the same words are being hashed so the program wouldn’t use the left over space allocated in the vector for both this table size or the optimized one.

Optimizations Attempted

I tried to optimize my program in a few different ways.

1) -O2.
    a. This brought my runtimes down for about 3 seconds from my first attempt. For 250x250 grid for words2 I had a runtime of about 240 seconds.
2) New Hash Method and Power Method.
    a. I implemented my own power method instead of the given “pow” method from the math.h library. I also implemented a new hash function, where it multiples each letter by a prime

```
int sum = 0;
for(int i = 0; i < word.length(); i++){
  sum = 37*sum + (int(word[i]) * (37^i));
}

return sum % 37571;
```

number and sums the hash together. This brought my runtime for a 250x250 grid for words2 to around 170 seconds.

3) Buffering Outputs
    a. This optimization led to my fastest runtime for words2 on a 250x250 grid. Instead of printing each word found when located, the words were concatenated into a string and printed out after the search was completed. The runtime was 112 seconds.

The biggest trouble I had with optimizations is most of the different ideas to decrease the runtime all had similar output times. For example, I experimented with different prime numbers to multiply for the hash function or to implement quadratic or double hashing methods to decrease collisions. However, these methods all had similar runtimes. I was confused why the times outputted were all similar. I did not include these times because the times were not improved by a big margin.

Overall Speed Up:

Although I did not optimize the program as much as I wanted to, my program decreased its runtime. The original runtime for words2.grid.txt for a 250x250 grid was 171.624 seconds. The optimized runtime was at 112.788 seconds. 171.624/112.788 = 1.52. Through the optimizations, the program increased its speed by a factor of around1.5.