Boris Topalov, bnt4yb

3/4/2020

After running my shell script to get the average runtime of my word search using words2.txt as the dictionary and the 300x300 grid, I found that my word search runs in between 500-600 milliseconds, typically, with most of my runs in the upper range (around 580ms). I also set my table size of my dictionary to 101 after that to see how it would affect the runtime, and I found that it basically doubled the runtime. It averaged around 1100ms, which is actually faster than I expected because the table is so much smaller (about 440x smaller). I then set the table size back to what it was original but made a really bad hash function: I set hash value = 101 * length of the string, and then returned that value mod table size. This, on average, ran in 75407 milliseconds, which is about 150x slower than with my original hash function. These results are unsurprising, as with this hash function there are only a few buckets in the hash table that are filled, and each of these buckets have many, many elements.

In general, I think that the fact that I am using a virtual machine is making my program run more slowly. Virtualbox runs quite slowly for me and is lags a lot, so I'm going to assume that the power of my computer is not being used to its full potential with the VM. I would guess that if I were to install Linux on my computer and use that as my main OS and ran this program again, it would run faster. I could be wrong though, and maybe it's just the video/display part of the virtual machine that's running slow, not computations like we are doing in the word puzzle.

I spent a lot of time trying to tweak my hash function because I thought that was why my code was running slow. I counted the collisions in my dictionary and found that there were about 6000 collisions with 25000 words, which seemed like a really bad ratio to me. After I found that, I printed out each bucket with more than one element in my hash table, but I found that the vast majority of these buckets only contained two elements. Some would contain three and even four elements, but no more than that from what I saw. I am sure that having a hash function with very few collisions would make it faster- but I don't think that it would make as significant as a difference as the differences that the optimizations I did implement made. I did try some different hash functions, though, to try and speed it up. One of my ideas was to store powers of 101 in a vector and use the indices of that vector in my hash function. That ended up being slower, though, so I just used my original hash function. With the vector of powers hash method, my program ran about 20% slower- instead of around 13.5 seconds for the 300x300 grid it was taking 16-17 seconds. I also tried to do double hashing, but I found that it wasn't working well with my find method in my hash table (or I just struggled to implement it). Only having two elements in a bucket seemed fast enough, so I ended up leaving my hash function as is.

The first step I took to optimizing my code was to use a buffer to store the output and printing it out after we had already found all the words and our timer had stopped. I used a vector of strings and just pushed back each word found in the correct format instead of printing out each word individually. This brought my timing down by a good margin but it still wasn't nearly fast enough- my runtimes for the 300x300 grid were typically between 12-15 seconds at this point. What sped my code up significantly after that was using a separate dictionary to store prefixes of each word. Even though there were a lot more collisions since there were so many more prefixes than just words, it made my word search a lot faster because I would move on to the next direction immediately if the word we were

looking at in the grid wasn't in our prefix hash table. This brought the time down to between 0.5 and 0.6 seconds. My overall speedup was around 100x since my program initially was taking 50-60 seconds for the 300x300 grid! This is a lot more than I expected and it definitely shows me how much faster a program can run if it is optimized well.

One thing I noticed is that my program ran faster when using the shell script versus calling the executable directly from the terminal. I am not sure why this is the case, since it's essentially doing the same thing- maybe it has to do with the way the program is compiled with shell scripts versus regular execution through the terminal? I would love for this question to be answered!

The big theta runtime of my word search is theta(r*c*w). I have two for loops that depend on input size (one loop for the number of rows and another nested for loop for the number of columns). Within those loops, I am calling find on each word in the grid. The worst case runtime for this would be w, since it's a lookup in a hash table and all of our elements could hash to the same spot so our hash table would basically be a linked list.