Name: Jiwon Cha
ID: jc4va
Date: 10/19/17

Post lab report

Big theta running time

I am still not sure about the big theta representation, but I believe that it is still r * c * w, because the increases in running times seem to follow a certain pattern; for example, for my original application column below, the running time seems to be: (number of rows/100) + (number of columns/100) + around 3 or 4. However, thanks to optimization, the actual value of big theta was reduced greatly.

Timing information

The unit for the results is in seconds.

The test was run on macbook air 13'.

	ORIGINAL APPLICATION	HASH FUNCTION MODIFIED	HASH TABLE SIZE MODIFIED
3X3 GRID	0.000156	0.000215	0.000508
4X7 GRID	0.001713	0.001791	0.003201
5X8 GRID	0.003312	0.003628	0.005857
50X50 GRID	0.652201	1.00396	1.42664
140X70 GRID	3.03952	4.72415	6.47671
250X250 GRID	20.8596	33.7382	49.3914
300X300 GRID (USING WORDS2.TXT)	3.30815	4.2745	3.99477

For my hash function, I originally used the third one provided in the hash table lecture slide, but I modified it to the first one on the slide.

1. $hash(s) = s_0 \mod table\_size$ 3. $hash(s) = \left(\sum_{i=0}^{k-1} s_i * 37^i\right) \mod table\_size$

The first hash function is usually inefficient, because the first letter of string mod table size will result in many collisions, as there are simply too many words that start with the same letters. Therefore, each slot of the vector will contain a very large linked list, which obviously makes find() much more inefficient.

I modified my hash table size to a relatively smaller prime number, 31. This will also result in many collisions because there are a small number of slots in the vector, and this will increase the running time in the same way as the hash function.

Optimizations

There were two main reasons the running time of my program was very slow; one in hashTable.cpp and one in wordPuzzle.cpp.

1. Hash table file

   First of all, I made a mistake in my hash function method; instead of multiplying 37 to i, which is the integer variable used as a parameter in the for loop, I was multiplying 37 to n, which was initialized to 0. Therefore, 37 * 0 is always zero, therefore the value of n was simply s[i]. Therefore, it was an extremely inefficient hash function, which resulted in a very slow runtime. This reduced my runtime significantly (to around half). Also, instead of i < s.length(), I decided to take first three letters of the string (i<3) and let others collide. This also improved my running time significantly.

2. Word puzzle file

   First of all, I was initializing hashTable to wrows before incrementing wrows to the number of lines in the dictionary file, therefore the size of my hash table was 2 as 2 is the smallest prime number. In other words, there were only two linked lists within the vector and naturally, there would have been a great number of collisions. Therefore, find() method in hash table would have been close to its worst case running time, O(n). Since find method was called inside the quad-nested for loop, it the word search was extremely slow. Therefore, I set the size of my hash table to 991 because it seems to produce the optimal result after running my program with different prime numbers. This optimization reduced my running time by at least 3/4.

   Then, just before the switch line, I made my code to break if the length of the word found by getWordInGrid method is not equal to len using if statements so that the program does not have to proceed to the switch statement. I was hoping it would reduce my running time, but unfortunately, it did not have a significant effect on the running time.

   Moreover, Since the lab is only taking into account the running time for the word search, I made a vector of strings to hold the output of the quad-nested loop so that the slow printing process would start after the timer stops. This only slightly decreased the running time.

I was originally very confused of why my program was running so slowly, because I got so much help from TAs for this lab who had many suggestions for me, therefore I believed that

my methodology is not the problem. My original running time for 250x250 grid using words.txt dictionary was around 15 minutes (877 seconds), but I reduced it to around 20 seconds. Therefore, through optimization, I greatly reduced my running time, and my speed up is around 43.85. However, even though my program runs faster than the sample output for smaller grid, the running time for bigger grids (250x250 and 300x300) is still greater than the sample output.