Yizhou Hao

CSE100 PA2 FINAL REPORT

1.BenchDict
The Data:

hashtable benchmarking
6000 11232
7000 13608
8000 10605
9000 12680
10000 13467
11000 13940
12000 16045
13000 18494
14000 16328
15000 16620
16000 13725
17000 10067
18000 10498
19000 10440
20000 10490

bst benchmarking
6000 47384
7000 45782
8000 46871
9000 47728
10000 47167
11000 49527
12000 48913
13000 49659
14000 49983
15000 51318
16000 51468
17000 51001
18000 52970
19000 52862
20000 53013

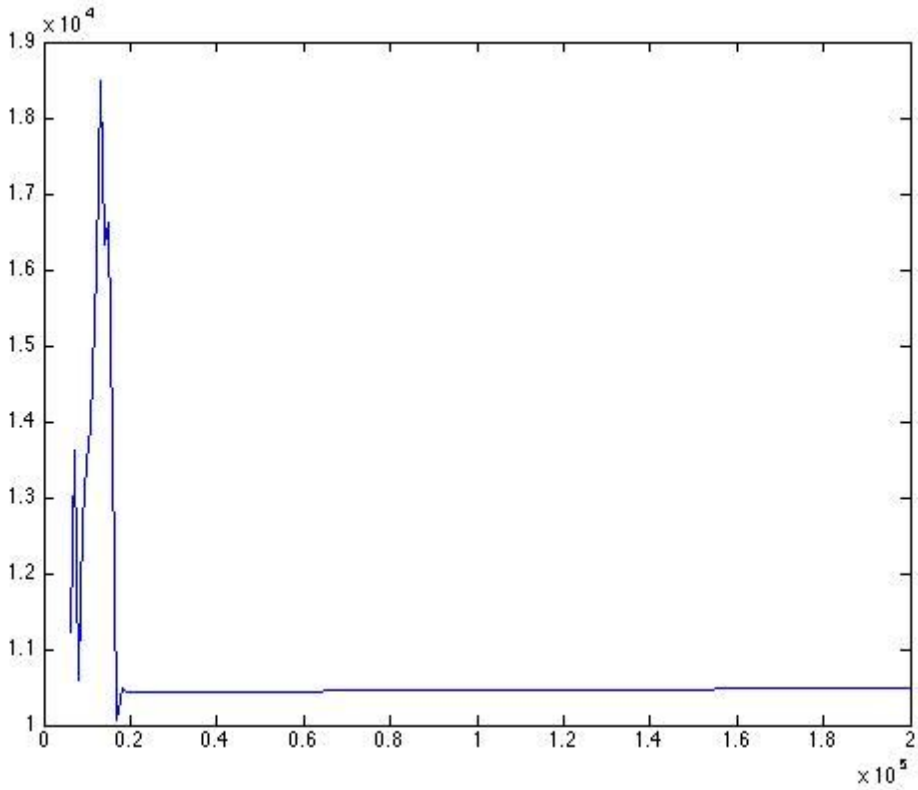
trie benchmarking
6000 22220
7000 22354
8000 21951
9000 24451
10000 25310

11000 23606
12000 23715
13000 23977
14000 24494
15000 25293
16000 24317
17000 25188
18000 25127
19000 24415
20000 26025

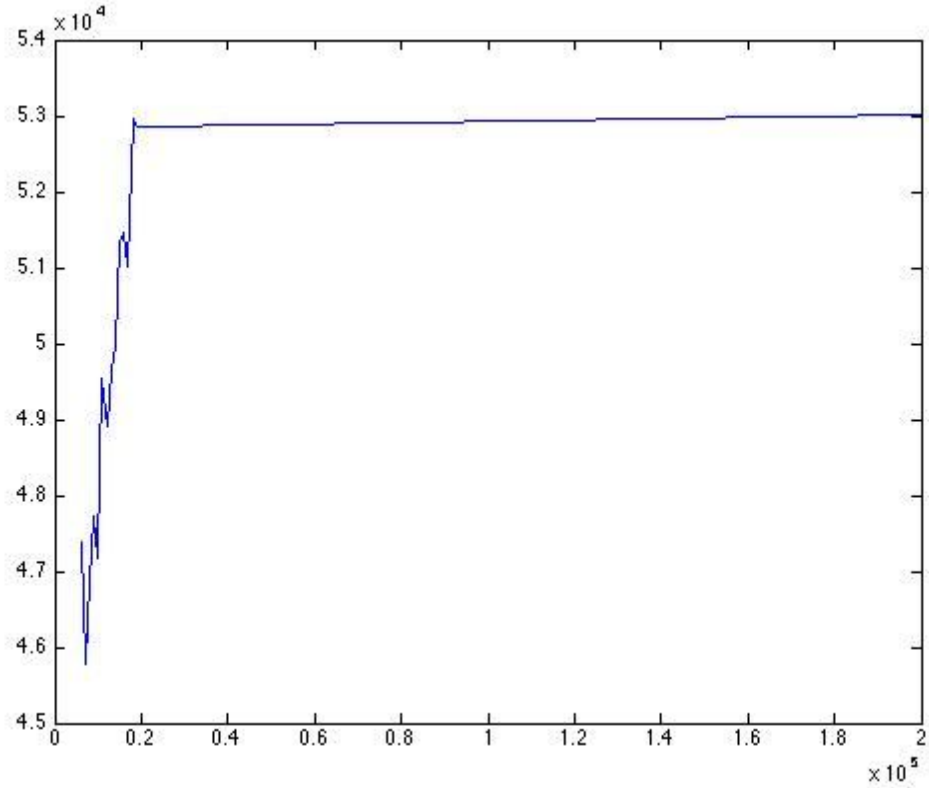
The graphs:

My graphs are consistent with theoretical running time analysis. The hashtable takes O (1). Bu t when the table is almost full, since it uses channing, it will take longer time to find a word when the size is bigger. But after the factor (N/M) reaches a limit, the hash table will resize. Hence, the running time drops and keep in almost constant time. And both bst and tst takes O(longn) to find. Those two graphs both show this relation.

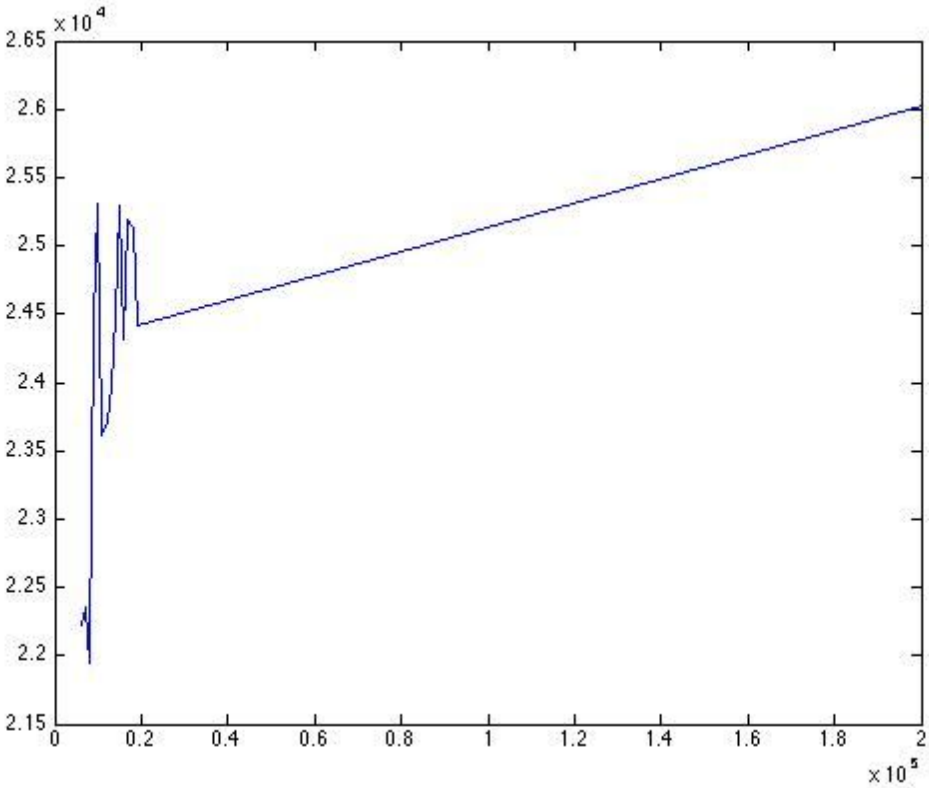
hashtable



bst


x 10^4
5.4
5.3
5.2
5.1
5
4.9
4.8
4.7
4.6
4.5
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
x 10^5

tst


x 10^4
2.65
2.6
2.55
2.5
2.45
2.4
2.35
2.3
2.25
2.2
2.15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
x 10^5

2. benchhash

a.

source:http://stackoverflow.com/questions/8317508/hash-function-for-a-string

The hashone is just to read each character's ascii value and add them up. Then, return the sum%(the size of table)

The hashtwo is that there is a contstant multiplier=131 and an initial hashvalue =0. In each iteration, the algorithm mutiplies hashvalue with the multiplier. Then it adds the ascii value of the character in that iteration to the product result (hashvalue*multiplier) to get the new hashvalue. Do it for each character and return hashvalue%(the size of table)

b. In my code, I use "a" "app" "apple" and size = 100 as input to test the correctness of two hash functions. I first calculate it by hands and compare it with the result from my program. They match, which means the program is correct.

Below are the results that I caculated by hand. (all with table size 100)

	hashone	hashtwo
a	97	97
app	21	1
apple	30	10

c.

input	average(one)	average(two)	max(one)	max(two)
(freq1,1000)	1.488	1.243	7	4
(freq1,2000)	1.911	1.2485	8	4
(freq1,5000)	3.2608	1.2512	16	5
(freq1,10000)	5.2395	1.255	26	5
(freq2,1000)	1.672	1.246	8	5
(freq2,2000)	2.2695	1.2405	10	4
(freq2,10000)	6.0734	1.2532	33	5

d.

Obviously, hashtwo is a better hash function.

When we increase the number of strings(insertion) in hashtable, the hashone will have sharply increasing average step and worst step. That means it will have much more collisions. However, hashtwo's average and worst step don't change a lot (very stable) when we increase the number of elements in the hashtable.

This is reasonable, because hashone function just add all ascii values together and return the mod. This will easily cause collision. But for hashtwo function, it has a multiplier for each char's ascii value. This decreases the possibility of collision.

3. fix of checkpoint

I fixed the seg fault, inster and find function in DictionaryTrie.cpp.

For the insert, my error was caused by that I didn't return for every recursion, I just return for the last layer of recursion.

For the find, I didn't go back and restart finding if it encounters the position that is already occupied