Fuat Yiğit Koçyiğit – 16429085948

CMPE343 Section 2

31.10.2021

TED ÜNİVERSİTESİ

CMPE343 Programming Homework 1
REPORT

Problem Statement and Code Design

It is expected to create a basic online dictionary application which stores all the distinct words given in input. I used both linear probing hash table and separate chaining hash table methods (as expected) to meet the needs. The linear probing hash table and separate chaining hash table has been implemented. Then, in the main class which is dictionary class, the input has been taken. While implementing the online dictionary, I recorded the run times of put and get in order to understand the performance of the code.

Implementation, Functionality and Performance Comparison

In the code, there is 5 different classes. 1 main class and 4 implementation classes. I implemented the linear probing hash table and separate chaining hash table with the SeparateChainingHashST and LinearProbingHashST classes, separately. These classes has methods such as hash, resize, put, get, getIndex and delete. Hash method is used to hash the words that user wanted to add to the online dictionary. Resize method is used to resize the hash table when it is required according to the separate chaining and linear probing algorithms. Put and get methods are used to add and get a word from the dictionary and getIndex method is used to get the index of a word in the hash table. In the main method of the program, I used the bufferedreader method in order to take the string input as one line from the user and then split it according to the spaces with the split method. Then I implemented the hash tables one by one and put the input words in them. Also, I used System.nanoTime() method to calculate the time passed in nanoseconds so that I can understand the performances of the dictionaries.

To make the **performance comparison**, I run the program 3 times with the different input (Test 1 is with the sample input, Test 2 is with the half of the sample input and Test 3 is with some extra words). The results are:

```
world world world world world world dog cat car car dog cat world window windows windo window ted cmpe 242 ted ted accommodate accommodate
Implementing linear probing hash table:
All words are added to the linear probing hash table.

Elapsed time to put words to linear probing hash table is 243700 nanoseconds
Elapsed time to get a random word from linear probing hash table is 1300 nanoseconds
////
Implementing seperate chaining hash table:
All words are added to the seperate chaining hash table.

Elapsed time to put words to seperate chaining hash table is 431300 nanoseconds
Elapsed time to get a random word from seperate chaining hash table is 1300 nanoseconds
////
```

Performance Test 1

```
world world world world world world dog cat car car dog cat world window windows windo window ted cmpe 242 ted ted accommodate accommodate
Implementing linear probing hash table:
All words are added to the linear probing hash table.

Elapsed time to put words to linear probing hash table is 74000 nanoseconds
Elapsed time to get a random word from linear probing hash table is 1400 nanoseconds
////
Implementing seperate chaining hash table:
All words are added to the seperate chaining hash table.

Elapsed time to put words to seperate chaining hash table is 404600 nanoseconds
Elapsed time to get a random word from seperate chaining hash table is 1300 nanoseconds
////
```

Performance Test 2

```
world world world world world world dog cat car car dog cat world window windows windo window ted cmpe 242 ted ted accommodate accommodate
Implementing linear probing hash table:
All words are added to the linear probing hash table.

Elapsed time to put words to linear probing hash table is 153400 nanoseconds
Elapsed time to get a random word from linear probing hash table is 1600 nanoseconds
////
Implementing seperate chaining hash table:
All words are added to the seperate chaining hash table.

Elapsed time to put words to seperate chaining hash table is 483400 nanoseconds
Elapsed time to get a random word from seperate chaining hash table is 1400 nanoseconds
```

Performance Test 3

According to the results, average elapsed time to put the words to the linear probing hash table is **~157,033.3** nanoseconds and average elapsed time to put the words to the separate chaining hash table is **~439,766.6** nanoseconds. Also, the average elapsed time to get a random word from dictionary in the linear probing hash table is **~1,433.3** nanoseconds and average elapsed time to get a random word from dictionary in the separate chaining hash table is **~1,333.3** nanoseconds.

According to these results, it can be seen that linear probing hash table is nearly 3 times faster than the separate chaining hash table for put process. In the get process, there is not a big difference between these two hash tables so the result is that the linear probing hash table is faster for our online dictionary.

To make the **memory comparison**, I also run the program 3 times with the different inputs (Test 1 is with the sample input, Test 2 is with the half of the sample input and Test 3 is with some extra words). The results are:

```
////
Final table sizes for linear probing and separate chaining are 64 and 16.
Top 3 most used words, their indexes for linear probing, their node indexes for separate chaining and their number of occurrences:
```

Memory Test 1

```
Final table sizes for linear probing and separate chaining are 32 and 16.
Top 3 most used words, their indexes for linear probing, their node indexes for separate chaining and their number of occurrences:
```

Memory Test 2

```
////
Final table sizes for linear probing and separate chaining are 128 and 16.
Top 3 most used words, their indexes for linear probing, their node indexes for separate chaining and their number of occurrences:
```

Memory Test 3

According to the results, it can be seen that linear probing hash table is using much more memory compared to the separate chaining hash table. So, the result is that separate chaining hash table is better for optimizing the memory usage in our online dictionary.

Testing

To test my program, I used 3 different inputs (same inputs in the performance and memory tests). First input was the sample input, and the output was:

```
world world world world world world dog cat car car dog cat world window windows windo window ted cmpe 242 ted ted accommodate accommodate
Implementing linear probing hash table:
All words are added to the linear probing hash table.

Elapsed time to put words to linear probing hash table is 243700 nanoseconds
Elapsed time to get a random word from linear probing hash table is 1300 nanoseconds
////
Implementing seperate chaining hash table:
All words are added to the seperate chaining hash table.

Elapsed time to put words to seperate chaining hash table is 431300 nanoseconds
Elapsed time to get a random word from seperate chaining hash table is 1300 nanoseconds
////
Final table sizes for linear probing and separate chaining are 64 and 16.
Top 3 most used words, their indexes for linear probing, their node indexes for separate chaining and their number of occurrences:
world 18 7 7
synonyms 24 5 5
accommodating 44 4 4
```

The second output was the half of the sample input, and the output was:

```
world world world world world world dog cat car car dog cat world window windows windo window ted cmpe 242 ted ted accommodate accommodate
Implementing linear probing hash table:
All words are added to the linear probing hash table.

Elapsed time to put words to linear probing hash table is 74000 nanoseconds
Elapsed time to get a random word from linear probing hash table is 1400 nanoseconds
////
Implementing seperate chaining hash table:
All words are added to the seperate chaining hash table.

Elapsed time to put words to seperate chaining hash table is 404600 nanoseconds
Elapsed time to get a random word from seperate chaining hash table is 1300 nanoseconds
////
Final table sizes for linear probing and separate chaining are 32 and 16.
Top 3 most used words, their indexes for linear probing, their node indexes for separate chaining and their number of occurrences:
world 18 7 7
ted 19 3 3
cat 22 2 2
```

The third output was like same input, but with more random words and the output was:

```
world world world world world world dog cat car car dog cat world window windows windo window ted cmpe 242 ted ted accommodate accommodate
Implementing linear probing hash table:
All words are added to the linear probing hash table.

Elapsed time to put words to linear probing hash table is 153400 nanoseconds
Elapsed time to get a random word from linear probing hash table is 1600 nanoseconds
////
Implementing seperate chaining hash table:
All words are added to the seperate chaining hash table.

Elapsed time to put words to seperate chaining hash table is 483400 nanoseconds
Elapsed time to get a random word from seperate chaining hash table is 1400 nanoseconds
////
Final table sizes for linear probing and separate chaining are 128 and 16.
Top 3 most used words, their indexes for linear probing, their node indexes for separate chaining and their number of occurrences:
world 18 7 7
synonyms 88 5 5
accommodating 44 4 4
```

I also did tests with very big and very small inputs but didn't report their results but there was no error in outputs.

According to my tests, I did not expect an extra failure tests since I tested them with very small, very big and complicated inputs and there was no error.

Final Assessments

- The trouble points that I experienced in this assignment was the little errors in the implementations of the hash tables and controlling the tests.
- The implementation and modifying the algorithms parts were the most challenging parts for me because there were so many errors, and the testing was so hard.
- I learned both hash tables and their implementations with this assignment and I really like it. Also I learned that finding a key may be so easy with the help of the hash tables.