Assignment10

Gang Liu

When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 30% of your Assignment 10 grade. Ensure that your analysis document addresses the following.

1. What does the load factor λ mean for each of the two collision-resolving strategies (quadratic probing and separate chaining) and for what value of λ does each strategy have good performance?

Quadratic : λ -to determine the real cost, the fraction of the map that is full. 0 <= λ <= 1. For this condition, It is better to rehash all items if λ>=0.5.

Separate chaining: λc = average length of linked lists. -Instead of rehashing when the table is half full, rehash when λc becomes large. Before do research, I set λc = 0.8 in the java classs.

2. Give and explain the hashing function you used for BadHashFunctor. Be sure to discuss why you expected it to perform badly (i.e., result in many collisions).

```
    public int hash(String item) {
        return item.length();
    }
```

For badHashFunctor, I just return the length of the item. It will cause lots of collision because there are too many string words have same length.

3. Give and explain the hashing function you used for MediocreHashFunctor. Be sure to discuss why you expected it to perform moderately (i.e., result in some collisions).

```
    public int hash(String item) {
        int a = 0;
        for(int i = 0; i < item.length();i++){
        a = a + item.charAt(i);
        }
        return a;
    }
```

For MediocreHshFunctor, I used the sum of ASCII number for each character in a String to be the Hash Number. Most of time, if the word is not anagrams with other items, it will have unique hash number. So there are less collisions happen compared with badHashFunctor.

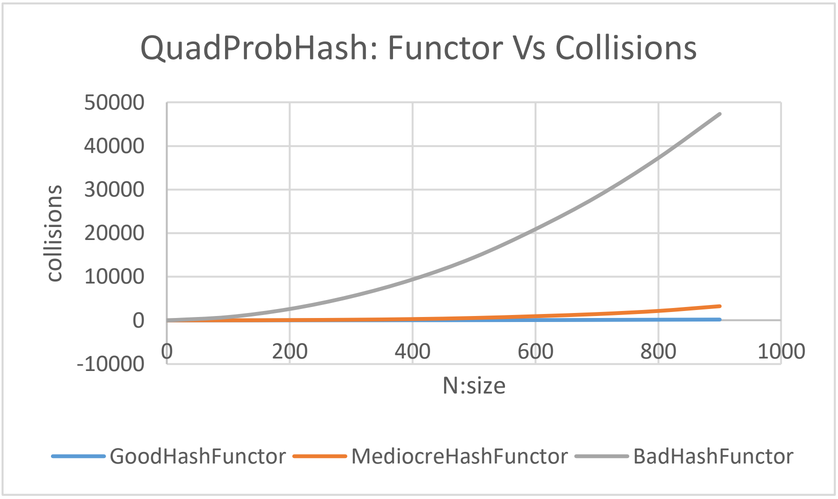
4. Give and explain the hashing function you used for GoodHashFunctor. Be sure to discuss why you expected it to perform well (i.e., result in few or no collisions).

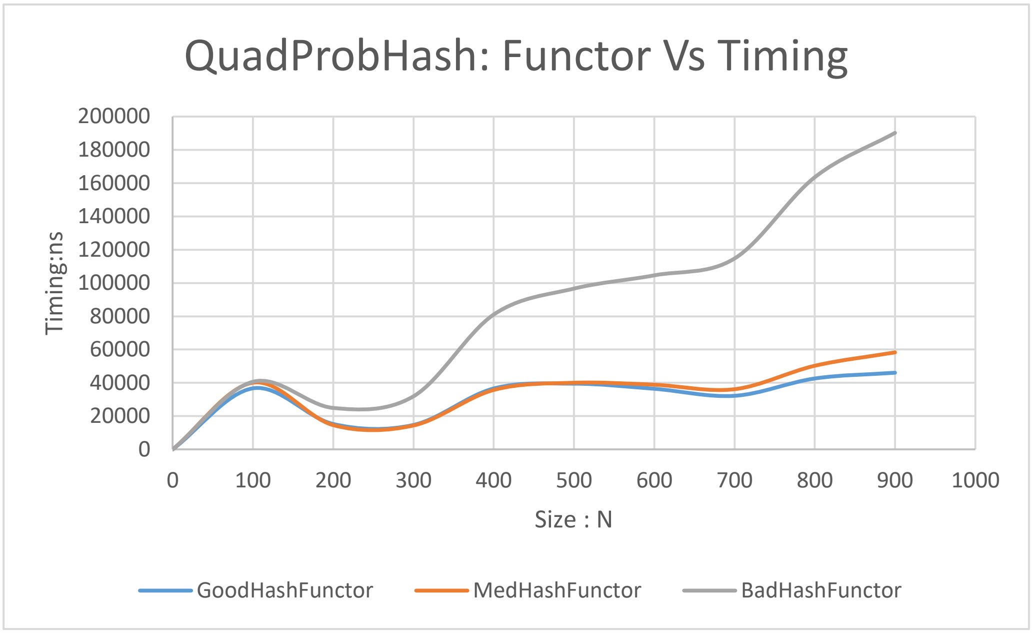
```
    public int hash(String item) {
        int hash = 7;
        for (int i = 0; i < item.length(); i++) {
            hash = hash*31 + item.charAt(i);
        }
    return hash;
    }
```

I searched this method online, and I will give my own guess.
It used a loop to deal with the number by it’s length and ASCII number. During each loop, hash number will times *31 + charAt(i),   the hash number is very huge and will be a very specific number. Under this situation, the method will just return a good hash code to make the index be different, and few collisions during add and contains method.

5. Design and conduct an experiment to assess the quality and efficiency of each of your three hash functions. Carefully describe your experiment, so that anyone reading this document could replicate your results. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is critical.
A recommendation for this experiment is to create two plots: one that shows the number of collisions incurred by each hash function for a variety of hash table sizes, and one that shows the actual running time required by each hash function for a variety of hash table sizes. You may use either type of table for this experiment.

1. I used the “dictionary.txt” to get strings , and store them into arrayList.
2. I set three different QuadProbHash tables with good,med,bad functor, the capacity is 3000
3.add different size of strings into the hashtable, and recode the time and collisions.





It easily to find that, in same plot, BadHashFunctor has lots of collisions, and GoodHashFunctor only have few collisions.

6. Design and conduct an experiment to assess the quality and efficiency of each of your two hash tables. Carefully describe your experiment, so that anyone reading this document could replicate your results. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is critical.
A recommendation for this experiment is to create two plots: one that shows the number of collisions incurred by each hash table using the hash function in GoodHashFunctor, and one that shows the actual running time required by each hash table using the hash function in GoodHashFunctor.

1.I initialized two constructor:

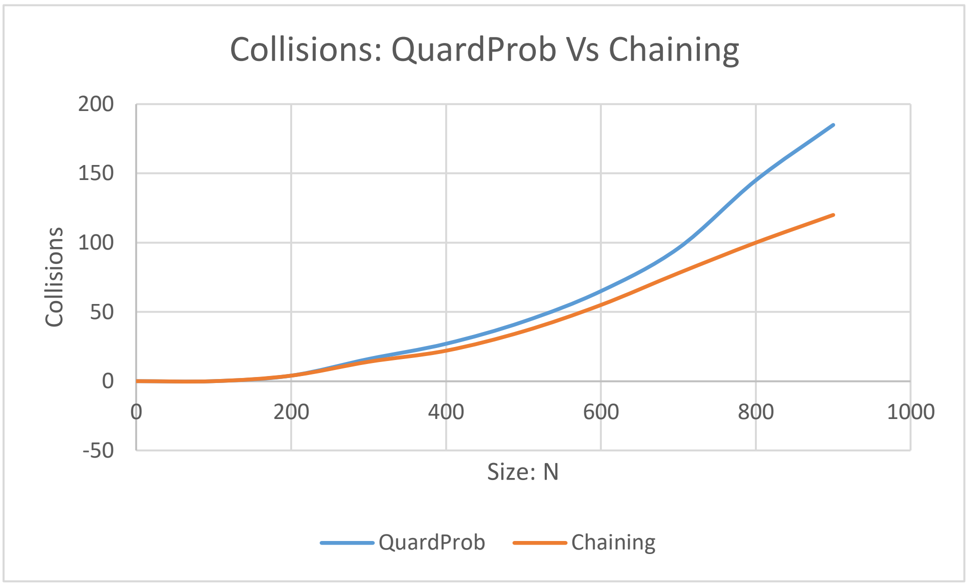
Functor1 is GoodHashFunctor.

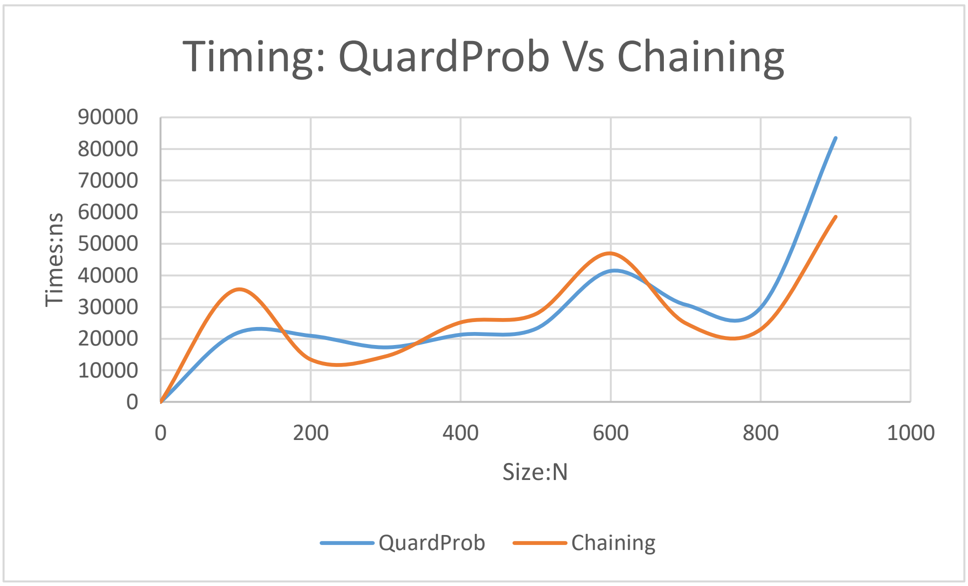
```
QuadProbeHashTable test1=new QuadProbeHashTable(3000,functor1) ;
ChainingHashTable test2 = new ChainingHashTable(3000,functor1);
```

One for QuadProbeHashTable, another for ChainingHashTable.

2. I used the “dictionary.txt” to get strings , and store them into arrayList.

3. .add different size of strings into the hashtable, and recode the time and collisions.





7. What is the cost of each of your three hash functions (in Big-O notation)? Note that the problem size (N) for your hash functions is the length of the String, and has nothing to do with the hash table itself. Did each of your hash functions perform as you expected (i.e., do they result in the expected number of collisions)? (Be sure to explain how you made these determinations.)

For BadHashFunctor, the cost is O(C), because I used the length to be he hash number. The computer only execute one time.

For MediocreHashFunctor, the cost is supposed to be O(N). I used the sum of ASCII number for each character in a String to be the Hash Number. For each character, the computer execute one loop.

For GoodHashFunctor, the cost is O(N) as well. It is very similar with the MediocreHashFunctor. Each character need one loop.

8. How does the load factor λ affect the performance of your hash tables?
The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity. Higher values decrease the space overhead but increase the lookup cost. Generally, I would rehash whole items if the load factor larger than 0.5.

9. Describe how you would implement a remove method for your hash tables.
On a delete, the actual item cannot be deleted from the table because items serve as placeholders during collision resolution. I will use lazy deletion, which marks items as deleted rather than actually removing them.

10. As specified, your hash table must hold String items. Is it possible to make your implementation generic (i.e., to work for items of AnyType)? If so, what changes would you make?
Sure it could be anyType. I would add an instructions to convert any type to String type.
For example:

```
int a = 19;
String strValue = String.valueOf(a);
System.out.println(strValue);
```

After that, I could use the hash functor which build for String type, to hash any type item.

11. How many hours did you spend on this assignment?
25 hours
Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.
Upload your solution (.pdf only) here by 11:59pm on November 9