Adrian Krystian Odbierzychleb 150283238

ADSOOF Mini Project Write Up

Introduction

For the mini project the task given, was to create a date structure storing a random number of random words. In order to achieve this a sub class had to be created called WordStoreImp.java, as well as a creating a date structure class which would store the words and manipulate them regarding to the code given in the WordTest files. This had to be completed without using any Java Library built in methods or classes, such as LispLists or ArrayLists. Therefore, a greater insight with in the topic was required, using one of the examples provided, research in to hash maps begun, which would be used for this projects data structure. In order to achieve mainly texts and examples on the internet were used, but also one book become very helpful, all this can be look at in the references section at the end.

What are hash maps

Hash maps is a data structure, which stores date assigning them a key and a value, each kay is assigned to a value creating key value pairs. They use a method which takes the key and the length of the array, created and index where the data is stored. Each stored data within the hash map, also has its unique hash, hence the name hash maps. The array length it uses for the data is of fixed value, that is where the load factor comes in. It is a value which in default is 0.75, once the load factor is reached the length of the array for storing data will be increased. They are a very useful data structure, as it is one of the most efficient data structures. This is ideal in this project as it would allow to go through the WordTest files in a very quick way.

WordStoreImp

First of all, the class WordStoreImp had to be created, which is a sub class of WordStore, as it implements it. It Consists of a constructor, and three methods which are also in WordStore.

The constructor in this class, is used to take the initial number of strings cleared when running WordTest, and create a Hash map equal to its size, in this case that method has been called CustomHashMap, the variable for it has been previously initialised within the class.

```
public WordStoreImp(int initialStrings)
```

Next is the first method, which is responsible for adding a new word in to the hash map.

```
public void add(String word)
```

The second method is responsible for the count of the hashmap, which means the number of words sotred in the hashmap.

```
public int count(String word)
```

lastly is the final method which is responsible from removing the words from the created hashmap.

```
public void remove(String word)
```

The full code of the class WordStoreImp can be seen bellow:

```
class WordStoreImp implements WordStore
{
  private CustomHashMap hashmap;

  public WordStoreImp(int initialStrings)
  {
    hashmap = new CustomHashMap(initialStrings);
  }
  public void add(String word)
  {
    hashmap.add(word, word);
  }
  public int count(String word)
  {
    return hashmap.count(word);
  }
  public void remove(String word)
  {
    hashmap.remove(word);
  }

}
```

Storage

Storage is a simple class used in order to store the data of the hashmap, it consists of for variables, one of type String responsible for storing the key, one of type String for storing the value, one of type int for storing the hashcode and one of Storage for storing the next entry added to the hashmap. The variables are only declared at this point they are initialises through the constructor which consists of the same variables, the values are gathered form the Hash map class, and passed on to the constructor as arguments. The code for this class can be seen bellow:

```
class Storage
{
  String key;
  String value;
  int hashcode;
  Storage nextentry;

  public Storage(String key, String value, int hashcode,
Storage nextentry)
  {
    this.key=key;
    this.value=value;
    this.hashcode=hashcode;
    this.nextentry=nextentry;
  }
}
```

CustomHashMap

The biggest class created responsible for the whole hash maps stracture it consists of a constructor which is connected to the the WordStoreImp, a method add which is also connected to the WordStoreImp, a method remove which is connected to the WordStoreImp, a method count which is connected to the WordStoreImp and 3 in class methods, getKey, getValue and hashCode.It also consists of 4 declared variables, a int type variable count, a Storage type variable entry which is an array, int type variable max and a float type variable which is the loadfactor, using the default value for hash maps of 0.75.

```
public CustomHashMap(int size)
```

The constructor takes the value given in the WordStoreImp and creates a variable initialsize of 1, and then a while loop which moves the one further until the initial size is greater than the size value passed to the constructor. Next the variable entry is given the value of a new Storage of the size of the initialsize. Lastly the max is given the value of the initialsize multiplied by the loadfactor.

```
public void add(String key, String val)
```

The first method is add which gets as arguments two Strings of key and value. First a hashcode variable is declared which takes the value of the method hashCode with the argument key. Next the value variable is declared, and it is given the value of the getValue method, with arguments hashcode and the length of the current storage array entry.
Last the latestetry variable is created which is of type Storage and represents the value of the entry array with index of variable value. Next a for loop is created which goes on ontill the variable latest has no more values meaning that all the words have been added to the array. In the loop the latestentry is given its individual key and value and the variables are added in to a temporary array of type Storage, which once obtaining all the values,

overwrites the entry array storing all the data, and also the max length is increased each time, by multiplying the overwritten entry arrays length multiplied by 2 and the default load factor.

```
public void remove(String key)
```

The beginning of this method is identical to the add one, except the fact that latestentry is a temporary variable which is given to a new variable toremove which is the value with in the entry array to be removed. In this case a while loop is used which goes on until the toremove value is null, meaning no more values are to be removed. A new variable nextentry is created which is assigned to the array entry at a given index, or if the latest entry is not to be removed it just gains the value of the next value within the Stored data, the count is reduced at each time a data is removed, this is repeated until all the required data is removed.

```
public int count(String key)
```

This method just takes key as an argument, and initialises a int type variable of count, it just goes through an loop which gets the keys of all values adding 1 to the count each time, once the loop stops the method returns the count.

```
public int hashCode(String key)
```

A method which takes the key as its argument and is responsible of assigning it its own hashcode. It goes through a loop which is as long as the length of the key argument passed, and the hashcode is generated by adding each k value to the hashcode variable, once the loop finishes it returns the variable hashcode.

```
public int getValue(int key, int length)
```

A method which takes key and length as arguments, the length being the length of the Storage entry array, and it multiplies the two values together , and returning that value.

```
public Storage getKey(String key)
```

A method which once again takes the key as its argument, and creates two variables hashcode, which gains the value of the method hashCode and entrykey, which gets the value at entry at index value, next it goes through a while loop until the entrykey is null if it is it just returns null, and if the entrykey is equals to the key argument passed on it returns the value of the declared variable entrykey.

The full code for this class can be seen bellow:

Adrian Krystian Odbierzychleb 150283238

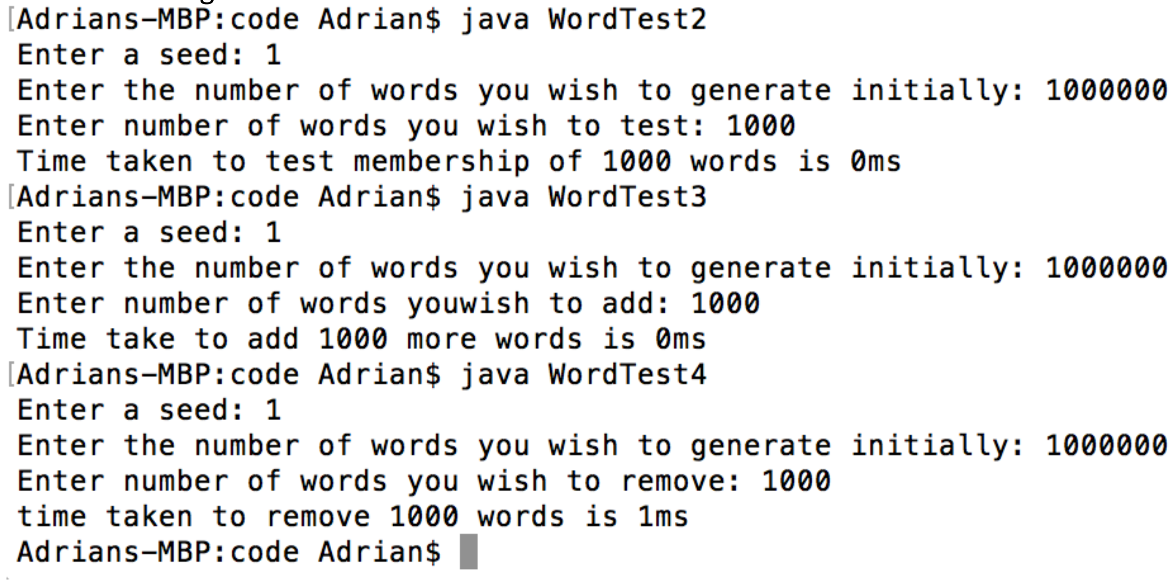
```
class CustomHashMap
{
  private int count;
  private Storage[] entry;
  private int max;
  private float loadfactor = 0.75f;

  public CustomHashMap(int size)
  {
    int initialsize = 1;
    while(initialsize<size)
    {
      initialsize <<= 1;
    }
    entry = new Storage[initialsize];
    max = (int)(initialsize*loadfactor);
  }

  public void add(String key, String val)
  {
    int hashcode = hashCode(key);
    int value = getValue(hashcode, entry.length);
    Storage latestentry = entry[value];
    for(;latestentry!=null;latestentry=latestentry.nextentry)
    {
      if(latestentry.key.equals(key))
      {
        key = latestentry.key + "1";
        break;
      }
      entry[value] = new Storage(key, val, hashcode, 
entry[value]);
      if(count++>=max)
      {
        Storage[] newEntry = new Storage[2*(entry.length)];
        for(int i=0; i<entry.length;i++)
        {
          Storage k = entry[i];
          if(k!=null)
          {
            entry[i]=null;
            while(k!=null)
            {
              Storage nextentry = k.nextentry;
              int l = getValue(hashCode(k.key), 
newEntry.length);
              k.nextentry = newEntry[l];
              newEntry[l]=k;
              k = nextentry;
            }
          }
        }
        entry = newEntry;
        max = (int)((2*entry.length)*loadfactor);
      }
    }
  }

  public void remove(String key)
  {
    int hashcode = hashCode(key);
    int value = getValue(hashcode, entry.length);
    Storage latestentry = entry[value];
    Storage toremove = latestentry;

    while(toremove!=null)
    {
      Storage nextentry = toremove.nextentry;
      if(toremove.hashcode==hashcode)
      {
        count--;
        if(latestentry==toremove)
        {
          entry[value]=nextentry;
        }
        else
        {
          toremove.nextentry=nextentry;
        }

      }
      latestentry = toremove;
      toremove = nextentry;
    }

  }

  public int count(String key)
  {
    int count = 0;
    while(getKey(key)!=null)
    {
      count++;
      key = key + "1";
    }
    return count;
  }


  public int hashCode(String key)
  {
    int hashcode=0;
    int k;
    for(int i =0; i<key.length();i++)
    {
      k = ((int)key.charAt(i)*(31^(key.length()-(i-1))));
      hashcode = hashcode + k;
    }
    return hashcode;
  }
  public int getValue(int key, int length)
  {
    int k;
    k = (int)key%length;
    return k;
  }

  public Storage getKey(String key)
  {
    int hashcode = hashCode(key);
    Storage entrykey = entry[getValue(hashcode, 
entry.length)];
    while(entrykey!=null)
    {
      if(entrykey.key.equals(key))
      {
        return entrykey;
      }
      else
      {
        entrykey = entrykey.nextentry;
      }
    }
    return null;
  }


}
```

WordTest Results

The WordTest 2, 3 and 4 have been testes with 3 different values, 1000 next with 10000 and at the end with 100000, in order to keep the results fair each time the seed number has been kept constant and the initial number of words, at values 1 and 1000000 respectively. Next graphs of time against the number of words have been created in order to see what type of gradient the line would give.

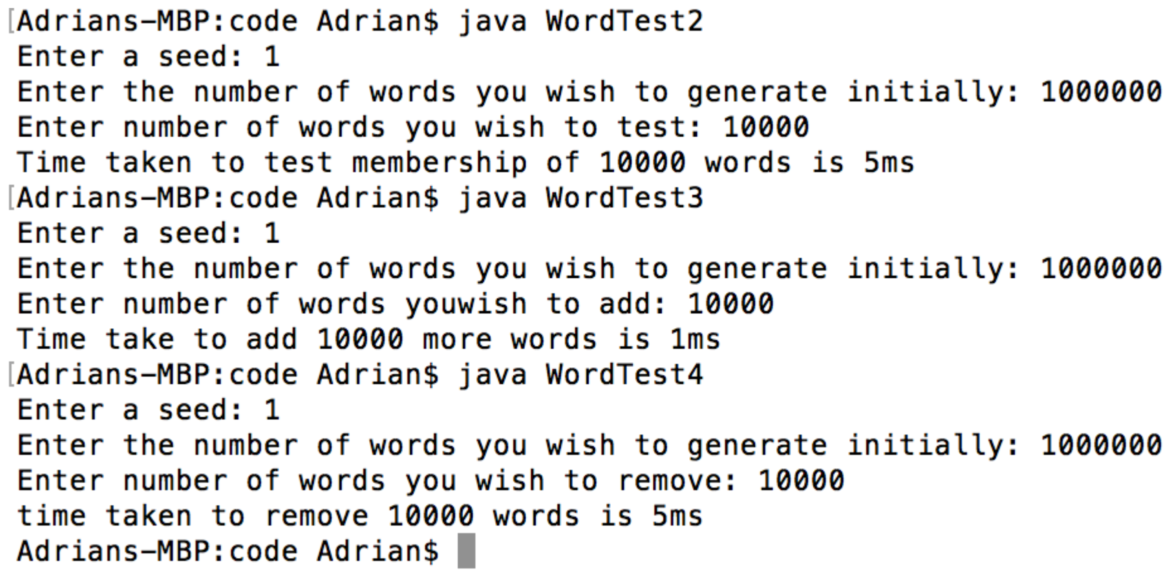
Adrian Krystian Odbierzychleb 150283238

First Test using the value of 1000 to test it.



```
[Adrians-MBP:code Adrian$ java WordTest2
Enter a seed: 1
Enter the number of words you wish to generate initially: 1000000
Enter number of words you wish to test: 1000
Time taken to test membership of 1000 words is 0ms
[Adrians-MBP:code Adrian$ java WordTest3
Enter a seed: 1
Enter the number of words you wish to generate initially: 1000000
Enter number of words youwish to add: 1000
Time take to add 1000 more words is 0ms
[Adrians-MBP:code Adrian$ java WordTest4
Enter a seed: 1
Enter the number of words you wish to generate initially: 1000000
Enter number of words you wish to remove: 1000
time taken to remove 1000 words is 1ms
Adrians-MBP:code Adrian$
```

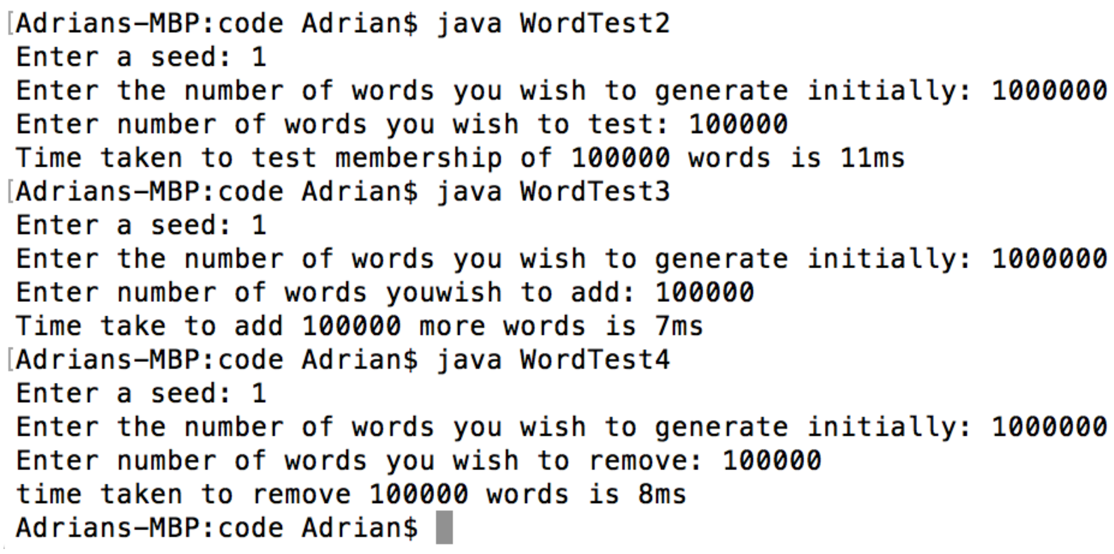
Second test using the value of 10000 to test it.



```
[Adrians-MBP:code Adrian$ java WordTest2
Enter a seed: 1
Enter the number of words you wish to generate initially: 1000000
Enter number of words you wish to test: 10000
Time taken to test membership of 10000 words is 5ms
[Adrians-MBP:code Adrian$ java WordTest3
Enter a seed: 1
Enter the number of words you wish to generate initially: 1000000
Enter number of words youwish to add: 10000
Time take to add 10000 more words is 1ms
[Adrians-MBP:code Adrian$ java WordTest4
Enter a seed: 1
Enter the number of words you wish to generate initially: 1000000
Enter number of words you wish to remove: 10000
time taken to remove 10000 words is 5ms
Adrians-MBP:code Adrian$
```

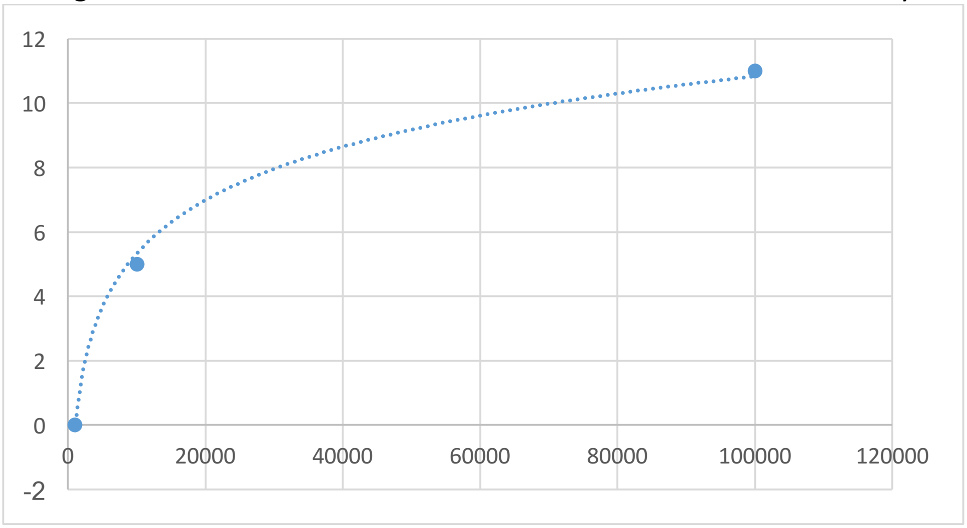
Third test using the value of 100000 to test it.

Adrian Krystian Odbierzychleb 150283238

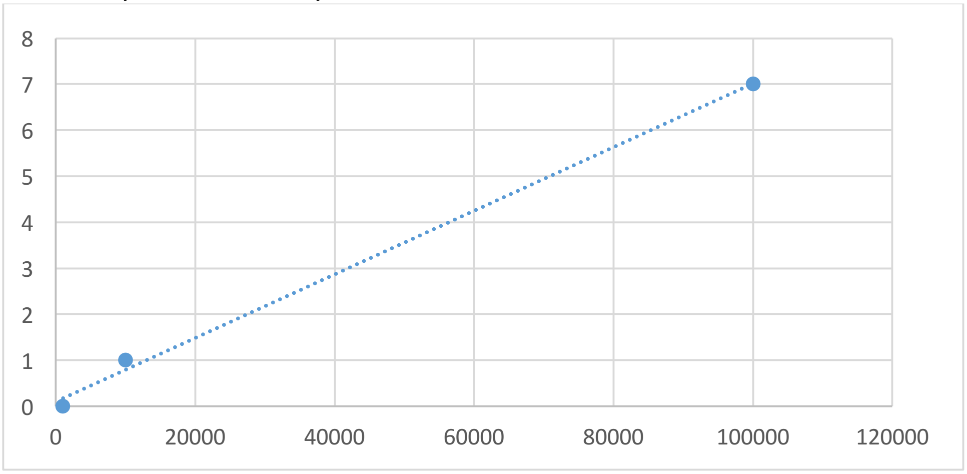


```
[Adrians-MBP:code Adrian$ java WordTest2
 Enter a seed: 1
 Enter the number of words you wish to generate initially: 1000000
 Enter number of words you wish to test: 100000
 Time taken to test membership of 100000 words is 11ms
[Adrians-MBP:code Adrian$ java WordTest3
 Enter a seed: 1
 Enter the number of words you wish to generate initially: 1000000
 Enter number of words youwish to add: 100000
 Time take to add 100000 more words is 7ms
[Adrians-MBP:code Adrian$ java WordTest4
 Enter a seed: 1
 Enter the number of words you wish to generate initially: 1000000
 Enter number of words you wish to remove: 100000
 time taken to remove 100000 words is 8ms
 Adrians-MBP:code Adrian$
```

The graph for WordTest2 can be seen below, it gives an logarithmic graph meaning that for even greater values of words the time taken will increase dramatically.

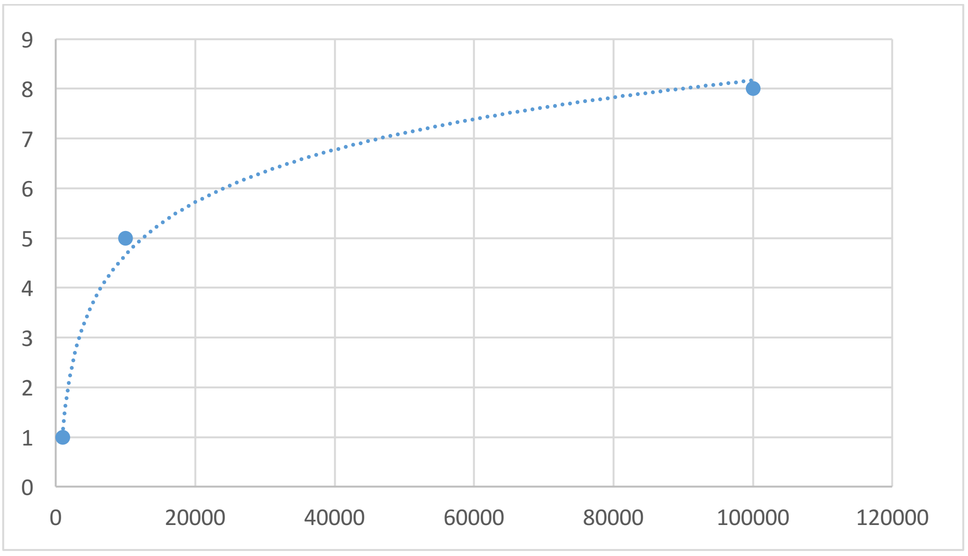


The graph for WordTest3 can be seen bellow, it gives an straight line, so the increase will be constant depending on the number of words, so if even more words were to be used it could be predicted easily what the it would take.



The graph for WordTest4 is given bellow, and its also a same type of graph as WordTest2

Adrian Krystian Odbierzychleb 150283238



Last is the results for WordTest1 which checks how many times a word has been generated, however in my case all of the tested words were not generated, meaning the count value for each was zero.

```
Enter a seed: 1
Enter the number of words you wish to generate: 1000000
Enter words to test, empty line to exit
hello
"hello" NOT generated
hi
"hi" NOT generated
world
"world" NOT generated
adsoof
"adsoof" NOT generated
data
"data" NOT generated
structure
"structure" NOT generated
adrian
"adrian" NOT generated
ako
"ako" NOT generated
ao
"ao" NOT generated
eecs
"eecs" NOT generated
_
```

Adrian Krystian Odbierzychleb 150283238

Referneces List:

1.https://en.wikipedia.org/wiki/Hash_table
2.http://javarevisited.blogspot.co.uk/2011/02/how-hashmap-works-in-java.html
3.https://beginnersbook.com/2013/12/hashmap-in-java-with-example/
4.https://stackoverflow.com/questions/730620/how-does-a-hash-table-work
5.https://www.quora.com/Whats-the-purpose-of-load-factor-in-hash-tables
6.https://www.quora.com/How-hashcode-value-is-calculated-in-Java
7.https://stackoverflow.com/questions/3795400/how-to-calculate-the-hash-code-of-a-string-by-hand
8.Introduction to Algorithhms Third Edition 2009 by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein