## OOP Report

### Part 1: user manual

###### how to use the GUI

1. **Launching the GUI:** Upon starting the program, a JFrame titled ‘My Language Model’ will be presented. Click the designated button at the top of the frame to import a text file. Please note that only .txt files are allowed, as the program includes a filter to restrict file types.
2. **Processing Time for Large Files:** If you choose a file containing a substantial volume of words (e.g. 'news.txt'), the program might take some time to process it. Be patient during this processing phase.
3. **Viewing Text in JTextArea:** Once the text processing is complete, the entire text will display within the JTextArea component.
4. **Navigating Analysis and Statistics:** At the bottom of the JFrame, a row of buttons will become visible. These buttons provide details related to 3-grams and various statistics associated with them.
5. **Prediction Using the Language Model:** Of particular interest is the last button, which facilitates the utilization of the language model for predicting probable word sequences. The input text field continuously listens to user input and generates predictions accordingly

### Part 2: Java constructor

Task 1 — MyLinkedObject

In class MyLinkedObject, I set 3 private field -- word, count and next that they can only be accessed or modified by Getter and Setter. And in the constructor for initializing, I set 1 for the word count and null for the next. In the most important method setWord(), all possible cases of the inserted word are discussed.

Task 2 — MyHashFunction

MyHashFunction class is a abstract class with one abstract method hashcode(). For implementing any hash function, I only need to create a new class **extending** MyHashFunction and overriding hashcode().

Task 3 — MyHashTable

In class MyHashTable, I introduce 3 fields --size for the length of the table, table for storing all of the first nodes of the linked objects, and hashFunction for algorithm to work out the hashcode of each word. In the constructor, I create MyHashTable with the given size.

Task 4 — vocabulary list

For words preprocessing, I **encapsulate** various of methods in 2 classes FormatCheck and ToWordConverter. The first one is used to check if the given word is in format of money, number and abbreviation, etc. If so, then use methods in ToWordConverter to change the words into strings. In the method convertToWords(), I use **recursive structure** to deconstruct the numbers step by step, shortening their number of digits and eventually converting them into words

**Task 5 — n-gram LM**

There are two methods both called getNextWord() but with different parameters(**polymorphism**). The method with one parameter is used for bigrams -- predict word by the previous one word only. While the method with two parameters is used for trigrams -- predict word by the previous two words.

###### use of the GUI

All of the code of GUI are written in class MyLanguageModel. The entrance of the UI is set in initGUI(). A JFrame is defined as the basis border with BorderLayout. It is divided into three parts:

* a JButton in the north for selecting files to import
* a JScrollPane in the center for showing the content
* a JPanel containing several buttons in the south for switching contents displayed

For showing three grams at the same time, I build up a method called showGrams() for encapsulation. In showGrams(), two JTable are created with two JScrollPane wrapped with each. The user can change the way the hashtable sorted by clicking the button below.

In statistics part, I combine three JTables with one JScrollPane for each into a JPanel.

In prediction part, a JTextField is used for input word and a JTextArea is set for showing prediction word sequence.

### Part 3: Discussion items

**Task 1 — MyLinkedObject**

**◦ Give a brief description for each method you have implemented for the MyLinkedObject class.**

* public boolean setWord(String w)

This method is defined for inserting words into current linked list. There are 4 kinds of situation to be discuss:

1. If current linked object has stored this word, then increase the count.
2. If w is alphabetically smaller than current word, return false to addLinkedObject() in MyHashTable who will handle this exception by inserting w before current word.
3. Then if current linked object doesn’t have next object, w will be linked to the next.
4. Then if w is alphabetically smaller than the word field of the next object, w will be added between current word and the next word.
5. Finally if none of the above situations is satisfied, w will be passed to the next object by calling setWord() recursively.

* public void printAll()

This is a test function to print all of the linked object from current node.

* public int nodesCount()

It is used to count the total number of linked object from the current one, which will be useful in MyHashTable.

**Task 2 — MyHashFunction**

**◦ Describe your choice of hash function algorithm(s). Where appropriate, mathematics may be used. The source (e.g., textbook, reference, web URL) of your choice must be provided.**

I totally set up seven hash functions. The first one, as required, use the ASCII of the first letter in the word. It is the cheapest method. However, since only one letter of the whole word is considered, the distribution of hash values is not uniform, sometimes StackOverflowError occurs because too many objects are inserted in one list.

The second one, HashFunForOne, accumulate the ASCII value for each character of the first word(used for bigram). The reason why I only count for the first word is because it is more easy to find the word sequence in hashtable by the first word, if the bigrams are stored by the same first word.

Similarly, the third hash function, HashFunForTwo, accumulate the ASCII value for each character of the first two word(used for trigram).

Then, the HashFunForAll, accumulate the ASCII value for each character, which is totally reasonable but necessary.

The next one, HashFun5, is also called multiplicationHash. It is a hash algorithm based on multiplication. The core idea of this algorithm is to multiply the input data by a constant A (usually a positive real number less than 1), then extract its decimal part, and finally multiply the decimal part by the size of the hash table to obtain the hash The integer part of the hash value, used as the position of the data in the hash table. It is good but the standard deviation of linked list length is bigger than that of HashFunForOne.  
 For HashFun6 and HashFun7, their utilization of bit operations might raise concerns about potential ArrayIndexOutOfBoundsException, possibly stemming from integer overflow issues. So finally, I choose HashFunForOne for unigrams and bigrams, HashFunForTwo for trigrams.

URL:

<https://blog.csdn.net/weixin_45730532/article/details/130659166>

<https://www.codenong.com/2571683/>

<https://www.zhihu.com/question/596143597/answer/2994675530?utm_id=0>

**Task 3 — MyHashTable**

**◦ hiding the MyLinkedObject class and the MyHashFunction class within the MyHashTable class...**

Yes I can achieve it by define MyLinkedObject and MyHashFunction in MyHashTable as fields. After calling the constructor of MyHashTable, a instance of MyLinkedObject(table) and MyHashFunction(hashFunction) will be generated. Then I can use the fields and methods in these two class freely.

**Task 4 — vocabulary list**

**◦ Describe your strategy for rearranging from the alphabetically ordered vocabulary list to the list with descending frequency order.**

I just use TreeMap in the java Collections Framework. By overriding the compare method in sortByWordCountsDes() and introducing it to create a new TreeMap, a descending frequency order will be sorted automatically.

**◦ What observation(s) you have made from the vocabulary lists created?**

The word appears the most is ‘the’. On the contrary, there are many words with the fewest occurrences, and their number is all 1. So the statistics part show only the smallest alphabetical one, which is ‘abandonment’. Also, there are some error words that cannot be recognized by the system, such as ‘<p>’, ‘</s>’.

**◦ Make comparison between hash function algorithms. Discuss which algorithm is the better choice for your hash table. Also discuss any observation you have made when using various values for the hash table size m.**

Firstly, I set a method(testForFun in Test.class) to test which hash function is the best. By using the news.txt that given and assuming the size of hash table is 100, I choose the hash function with the lowest standard deviation as the best function. Then I create a method(testForSize in Test.class) to select the most suitable size of the hash table size. I set a loop from 10 to 99 as the supposed size of the table and print out the standard deviation corresponded. Finally, 70 was chosen as the most suitable size.

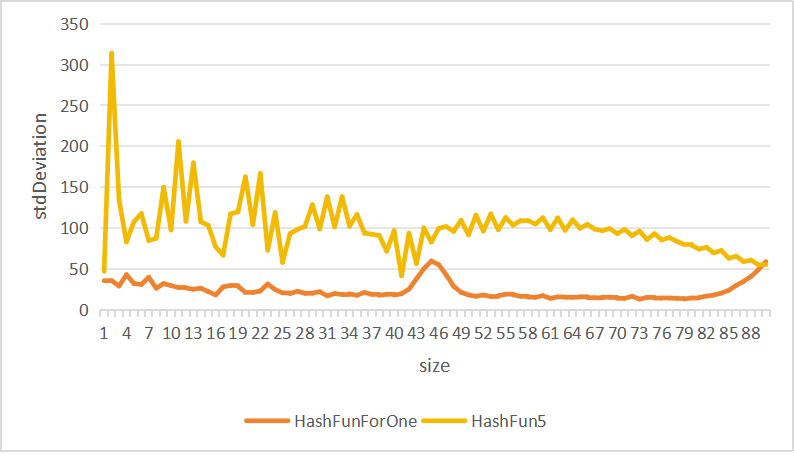


Figure1 Standard deviation corresponding to different sizes

From the figure 1, we can see the standard deviation using HashFun5 are always higher than that of HashFunForOne. Also, in the size range of 50 to 80, both of the values are stable.

**◦ For natural language processing tasks such as this, discuss potential benefit(s) of sorting ‘linked lists’ by the decreasing number of word occurrences, instead of ones ordered alphabetically.**

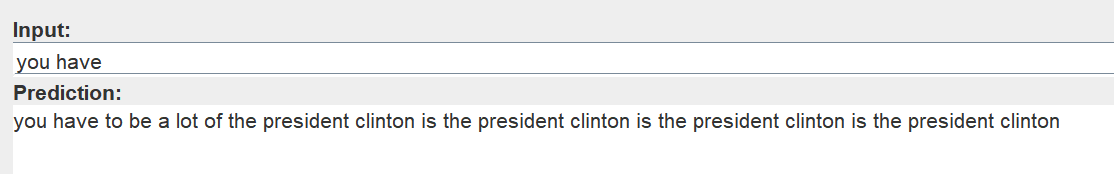
When predicting, we always want to find the word that appears most frequently. For every word a user inputs, the system needs to find the next most likely word from the LM, so the order of words stored in the LM really affects the prediction speed. By sorting words in decreasing order, we can search words faster.

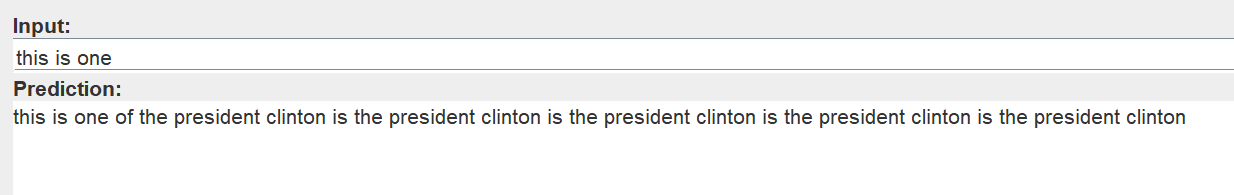
**Task 5 — n-gram LM**

**◦ What do you do with p(w1) when calculating p(w1,w2,...,wK) using unigrams and bigrams? Similarly, what do you do with p(w1) and p(w2|w1) when calculating p(w1,w2,...,wK) using up to trigrams?**

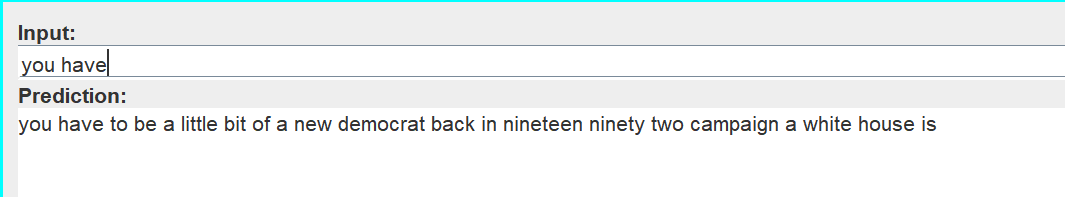
Actually, I don’t caculate p(w1) while predicting because p(w1) is a constant in current LM that I only need to consider p(wK|wK−1). The same goes for p(w1) and p(w2|w1) in trigrams.

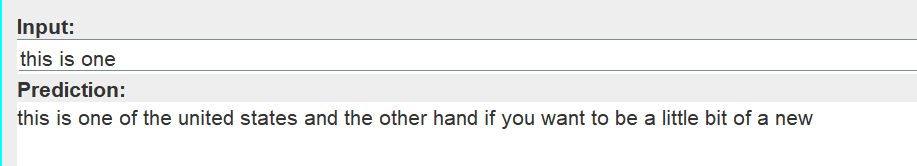
**◦ Using unigrams and bigrams only, find the first 20 words that most likely follow the following two words, ‘you have’. Repeat the same for ‘this is one’.**



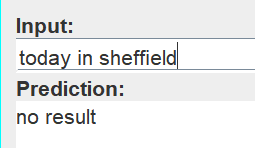
****

**◦ Now using also trigrams in addition to unigrams and bigrams, find the first 20 words that most likely follow the following two words, ‘you have’. Repeat the same for ‘this is one’.**

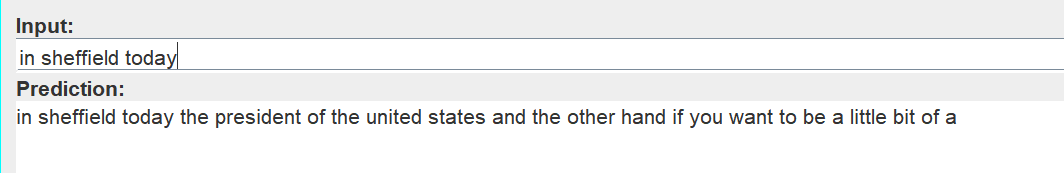
****

****

**◦ Using unigrams, bigrams and/or trigrams, are you able to find the first 20 words that most likely follow the following three words, ‘today in sheffield’? If not, why not? Repeat the same for ‘in sheffield today’.**

****

I cannot get any prediction because word ‘sheffield’ is not in the LM.



For ‘in sheffield today’ there is an answer. Due to the code logic I set -- if method getNextWord (word, nextWord) returns null, call getNextWord (word) instead. There is a return of getNextWord(today) so I can make prediction successfully.

**◦ Discuss any issue(s) of implementing n-grams with a larger value of n than ‘3’.**

If continue to expand the value of n, we need to consider such as p (wK | wk-wk 3, wK 2, wK 1) probability(when n=4), although the prediction results will be more accurate, but p will be more likely to 0. At this point, you have to switch to trigrams or less grams which is complex to set the logic.