

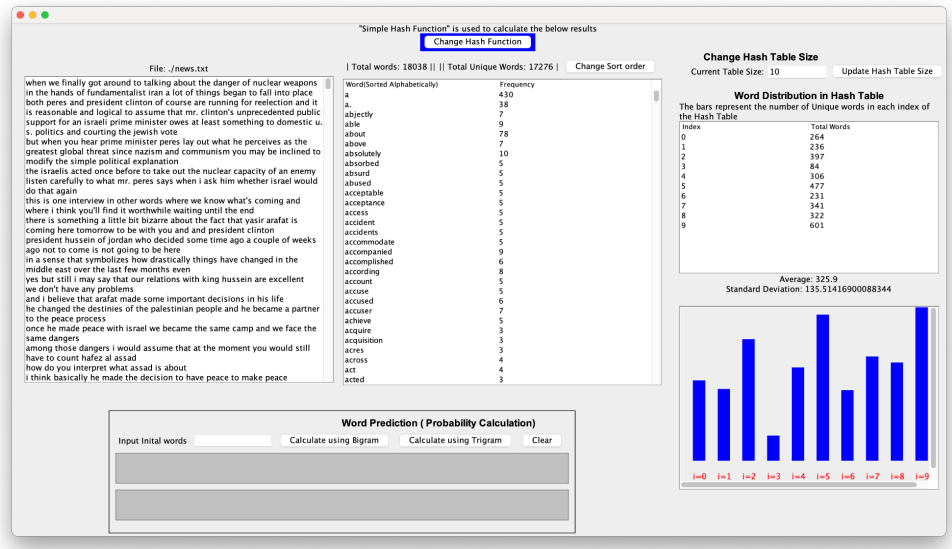
# Large language Model

Author: Manu Kenchappa Junjanna

Email: mkenchappajunjanna1@sheffield.ac.uk

## UserManual

Upon launching the program, users are presented with the initial choice of either directly selecting 'news.txt' or choosing any file using the 'Pick News' and 'Pick File' buttons, respectively. After the user makes a file selection, the program processes the data, and the results are displayed. Figure 1 illustrates the interface with the left section showcasing the contents of the selected file, the centre section presenting a table depicting words and their frequencies.



The centre section also includes a "Change Sort Order" button for toggling between alphabetical and descending frequency sorting. The rightmost section is dedicated to statistical data. A button is provided to dynamically change the hash table size by entering the new hash table and clicking "update hash table size," which will recompute the data with new update the hash table size. Below this section, the Word Distribution in the Hash Table is displayed, offering insights into the word count at each index of the Hash Table. Additionally, the average and standard deviation of the data are shown. Furthermore, a bar graph below enhances the visualisation of the distribution.

The bottom section displays the most likely 20-word sequence upon entering input. Two options are available: one using Bigram and the other using Trigram. The results for each option are shown in different text fields. Clicking the "Calculate using Bigram" button displays the next most likely 20 words in the first text area, and clicking "Calculate using Trigram" shows the next most likely 20 words in the second text area. The "Clear" button erases both the input and output, allowing users to enter new data afresh.

The final crucial feature is the "Change Hash Function" option at the top. It enables users to switch the hash function during runtime. In the current assignment, two hash functions are available: the Simple Hash Function, provided in the assignment, and the Polynomial Hash Function, explained in detail below. Clicking on "Change Hash Function" toggles between the Simple and Polynomial hash functions. The entire dataset is then reprocessed with the new hash function, and the updated results are displayed in the UI.

## Java Programming Concepts and Framework Utilisation

**Data Abstraction** is employed in the classes MyHashTable and MyLinkedList, hiding the internal details of how a hash table or a linked list is created. Instead, these classes provide functionality for adding and retrieving data in both the case of a hash table and a linked list.

**Polymorphism** is effectively applied in the hash function classes. The MyHashFunction class acts as a superclass, establishing a shared interface for various hash functions. Both the SimpleHashFunction and PolynomialHashFunction classes serve as subclasses, each implementing its version of the "public int hash(String word);" function.

**Encapsulation** is applied in both the MyHashTable and MyLinkedList classes. This involves declaring data members as private and interacting with them through public methods. For instance, in the MyLinkedList class, the word, count, and next variables are kept private and accessed or modified using dedicated getter and setter methods.

**Inheritance:** Although I have not directly implemented inheritance, I applied the concept by creating panels for various UI components like ProbabilityCalculationPanel, InputReadFilePanel, WordAndCountTablePanel, etc. These panels inherit from the JPanel class, allowing the creation of custom panels with properties inherited from JPanel.

Additional concepts, such as **Composition**, involve the Frame class containing a ( "has a" relationship ) BarGraphPanel and a DistributionBarPanel. There is also an **Association** where MyHashTable class utilises a MyHashFunction class to calculate hash values. It's important to note that these two objects have separate lifecycles. Additionally, the assignment employs the use of **Anonymous classes** to set addActionListener for Buttons.

The **Collection Framework**, including classes like Map and TreeMap, as well as utilities like Comparator, is primarily used for sorting words alphabetically and in descending order of frequency count in the "WordAndCountTablePanel" class.

In the **GUI implementation** (Frame.java), I've utilised Swing components like JFrame, JPanel, JButton, JFileChooser, and SwingWorker. Additionally, custom panels (BarGraphPanel and DistributionBarPanel) extend JPanel. The ProbabilityCalculationPanel and WordAndCountTablePanel classes also extend JPanel and incorporate various Swing components.

For **Exception Handling**, I've employed this concept in the validation of input during the prediction of the most likely 20 words. I throw an IllegalArgumentException in methods like get20MostFrequentWordsUsingBigram and get20MostFrequentWordsUsingTrigram if the constraints are not met. The exceptions are then caught ( in ProbabilityCalculationPanel class ) to display appropriate messages in the UI.

## User Interface Design and Features

### MyLinkedList Methods

1. Set and get methods of data members "word","count" and "next"
  - a. public void setNext(MyLinkedList newNext)
  - b. public MyLinkedList getNext()
  - c. public void seCount(int count)
  - d. public int getCount()
  - e. String getWord()
  - f. public int setWord(String w)

The setWord method is used to insert a word into a linked list. It compares the input word with the current word. If they are equal, it increments a count. If the input word is greater, it inserts the word further into the list. If it is less, it inserts the word before the current node. The method returns 0 if the word was already in the list and 1 otherwise. This return type indicates that a new word has been added.

2. Clone method "*public MyLinkedList clone()*"

The clone method in the MyLinkedListObject class creates a new object with the same word, count, and next values as the current object and returns this new object

### 3. toString “*public String toString()*”

The toString method in MyLinkedListObject class returns a string representation of the object, combining the word and count fields, separated by " | ", and ends with " | --->". This is used for debugging purpose

## Choice of hash function algorithm | Polynomial Hash Function

The Below hash function is inspired by the method toHashCode of Java, and taken from <https://computinglife.wordpress.com/2008/11/20/why-do-hash-functions-use-prime-numbers/>

```
1 @Override
2 public int hash(String word) {
3     int hash = 0;
4     for (int i = 0; i < word.length(); i++) {
5         hash = (hash * 31 + word.charAt(i)) % hashTableSize;
6     }
7     return hash;
8 }
```

*assignment/hash\_table/PolynomialHashFunction.java*

The hash method in the PolynomialHashFunction class calculates a hash value for a given word by iterating over each character in the word. It multiplies the current hash by 31, adds the ASCII value of the character, and takes the modulus of hashTableSize to ensure the value remains within the range of 0 to hashTableSize. The resulting hash value can be used as an index into the hash table.

## Encapsulating MyLinkedListObject and MyHashFunction

This has been accomplished by employing package and default access specifiers. The “hash\_table” package in the assignment encapsulates the classes MyLinkedListObject and MyHashFunction (SimpleHashFunction and PolynomialHashFunction). These classes are only accessible within this package, ensuring restricted visibility, i.e., they are “package-private” classes. In contrast, the MyHashTable class employs a public access specifier, enabling it to be accessed externally. Consequently, MyLinkedListObject and MyHashFunction can only be utilised outside the package through the MyHashTable object.

## Sorting Strategy: Alphabetical & Descending Frequency

To effectively implement this feature, I have made use of TreeMap<String, Integer>, where the key (String) represents the word and the value (Integer) represents the frequency. TreeMap maintains the elements in sorted order based on their natural ordering or according to a specified comparator.

### 1. Alphabetical Sorting:

```
TreeMap<String, Integer> map = new TreeMap<>();
```

The TreeMap sorts entries based on their natural ordering by default, When elements are added, the words are sorted alphabetically.

## 2. Frequency-Based Sorting:

```
TreeMap<String,Integer>frequencyOrderMap=new  
TreeMap<>(Comparator.<String,Integer>comparing(map::get).reversed()).thenComp  
aring(Comparator.naturalOrder());
```

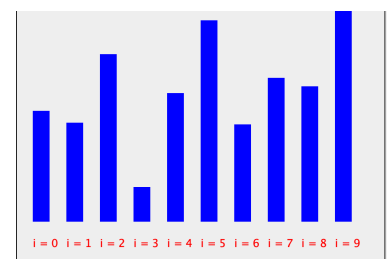
To sort based on frequencies, I have to provide a custom comparator to sort entries by their values (frequencies). This comparator first compares entries based on frequency in descending order (`reversed()`), and in case of a tie (equal frequencies), it resorts to natural order.

When Creating Vocabulary list one thing which is noticeable for the given data set most of the high frequency words are These are typically common words known as "stop words" like "the," "to," "a," "of," and "and" with more data set we can get more data which can be analysed

## Analysing Hash Function Algorithms : Linked List Statistics and Observations

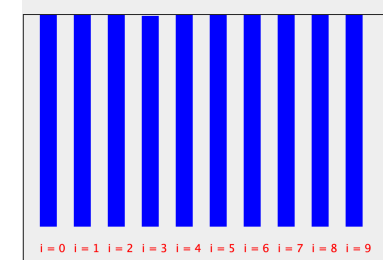
1. SimpleHashFunction: When using this hash function, the words are unevenly distributed in the hash table. For example, as shown in the image, indices 5 and 9 have more words compared to other indices

Index	Total Words
0	264
1	236
2	397
3	84
4	306
5	477
6	231
7	341
8	322
9	601



2. PolynomialHashFunction: When using the polynomial hash function, the words in each index are evenly distributed, providing a more balanced way to distribute the data

Index	Total Words
0	325
1	333
2	319
3	294
4	330
5	351
6	344
7	355
8	303
9	305



When comparing both hash functions, PolynomialHashFunction is the better choice. PolynomialHashFunction considers all characters of the given word to calculate the hash, unlike SimpleHashFunction, which only considers the first character. This results in higher collision resistance

When **adjusting hash table sizes** for Simple and Polynomial Hash Functions, both maintain consistent averages. However, the Polynomial Hash Function demonstrates lower standard deviations, ensuring more controlled word distribution. The choice of hash function has a greater influence than table size, highlighting the Polynomial Hash Function's stability and predictability in distribution.

**Sorting words by frequency** improves accessibility, enhances pattern recognition and trend spotting, outperforming alphabetical sorting. This optimization significantly boosts overall efficiency in processing and interpreting textual data

## Analysing N-Gram Models in Natural Language Processing

When predicting the top 20 words using unigram and bigram probabilities ( $p(w_2|w_1)$ ), we can skip calculating  $p(w_1)$ . With the user inputting at least one word ( $w_{k-1}$ ), we can directly compute the probability of  $w_k$  given  $w_{k-1}$  using  $p(w_k/w_{k-1}) = c(w_k, w_{k-1}) / c(w_{k-1})$ .

Similarly, for trigrams, omitting the need for  $p(w_1)$  and  $p(w_2|w_1)$ , if the user provides at least two words ( $w_{k-2}$  and  $w_{k-1}$ ), we can determine the probability of  $w_k$  given  $w_{k-2}$  and  $w_{k-1}$  as  $p(w_k/w_{k-2}, w_{k-1}) = c(w_{k-2}, w_{k-1}, w_k) / c(w_{k-2}, w_{k-1})$ .

### Predictive Analysis:

#### 1. Using Unigrams and Bigrams :

"you have" : "you have to be a lot of the president clinton when you know what the president clinton when you know what the"

"this is one" : "this is one of the president clinton when you know what the president clinton when you know what the president clinton when you"

#### 2. Using Trigrams

"you have" : "you have to have better investment policies you have to have better investment policies you have to have better investment policies you"

"this is one" : "this is one of the c. i. a. station chief colby became director of the c. i. a. station chief colby became director"

When entering "today in Sheffield," the system fails to generate the next 20 likely words using unigrams, bigrams, or trigrams. An error message in the UI states, "Cannot predict the next word since 'today in Sheffield' is not in the dataset." This issue arises because predicting the next word ( $w_{k+1}$ ) requires the presence of the preceding word ( $w_k$ ). In this context, "Sheffield" is absent, leading to prediction failure. In contrast, inputting "in Sheffield today" successfully predicts subsequent words, leveraging the presence of the preceding word "today."

### Challenges of Larger n-grams:

Implementing n-grams with a value greater than '3' presents significant challenges, primarily due to memory constraints and computational complexity. Firstly, considering more than 4 words in a sequence increases uniqueness, demanding more memory—impractical for large datasets. Secondly, calculating probabilities for larger n-grams becomes computationally intensive, impacting real-time processing and user experience.

