Task1

A class called MyLinkedObject serves as a blueprint for the objects in a linked list. It consists of three private fields called "word", "count", and "next", which stand for word in this linked object, word occurrence in this linked list, and next linked object related to this object, respectively. The following enumerations show the explaination of each methods implemented for this class:

1. **MyLinkedObject(String w)**: Constructor method that initializes a MyLinkedObject with a given word (w). It sets the word by the given word, initializes the count to 1, and sets the next reference to null.
2. **setWord(String w)**: Adds a word to the linked list. If the word is already present, it increases the count. If the word is not present, it instantiates a new MyLinkedObject to the linked list in the appropriate alphabetical order and correctly link the object by the "next" field.
3. **toString()**: Overrides the default toString() method to provide a string representation of the MyLinkedObject, showing its word and count, and recursively displays the next MyLinkedObject in the linked list.
4. **Getter, Setter**: Because the class fields were set to private, there are some Getter and Setter methods to return the fields or set the fields.

Task2

A Hash function is to take a input sequence, using mathematics to produce a hash code in order to map linkedObjects into hash table. A good hash function can make the hash table index evenly distributed, which can reduce the adverse effects of clustering. There are some kinds of hash methods list below:

1. **MyHashFunction**: A super class of all hash function, which contains some basic properties and methods, but does not implement specific hash function logic, leaving the details of these implementations to its subclasses.
2. **Modulo Hashing**: Inherits from the MyHashFunction parent class and implements the hash method for hash computation. By taking the first character of the word sequence and converting it to a lowercase letter, take the Unicode value of the letter to modulates the hash table size and returns the result as a hash code.
3. **MD5 Hashing [1]**: A MessageDigest instance is created and specified to use the MD5 hash algorithm. Converts the input string to a byte array and updates MessageDigest. Gets a byte array of the digest and converts it to a hexadecimal string representation. The hexadecimal string is converted to an integer, and the result is returned as a hash code by taking the absolute value and then modulo the hash table size.
4. **SHA-1 Hashing [1]**: It is similar to MD5, but uses the SHA-1 hash algorithm to calculate the hash code of the input string

Task3

MyHashTable class is a class for hash table operations that integrates the MyLinkedObject class and the MyHashFunction class for storing and processing hash tables of word sequences.

The constructor MyHashTable(int hashTableSize, String hashFunction) takes as arguments the size of the hash table and the type of the hash function. According to the hash function given, select the corresponding hash function object for instantiation.

The MyHashTable class implements the insert(String wordSequence) and insertNGrams(String[] lastWord, String wordSequence, int n) methods, Using the hash function passed in when instantiating the MyHashTable object, hash word sequence as the key of that sequence in the hash table, and then split word sequence into N-grams (sequences of n words), Finally, n-grams are formed into a linked list and stored in a hash table by recursion calling the setWord(String w) method in the MyLinkedObject class.

This effectively hides the MyLinkedObject class and the MyHashFunction class. When using the MyHashTable class in the future, user only need to pass in the hash function type and HashTable size. The data can then be stored by insert or insertNGrams function.

Task4

vocabularyListSorter

```
TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<>(tableModel);
vocabularyTable.setRowSorter(sorter);

// Set sorter's rule
sorter.setSortable(1, true); // Allows the specified column to be sorted
sorter.setComparator(1, Comparator.naturalOrder()); // Sort alphabetically using the
natural order comparator
sorter.setSortKeys(Arrays.asList(new RowSorter.SortKey(1, SortOrder.DESCENDING)));
// Sort in descending order

sorter.setSortable(0, true); // Allows the specified column to be sorted
sorter.setComparator(0, Comparator.naturalOrder()); // Sort alphabetically using the
natural order comparator
sorter.setSortKeys(Arrays.asList(new RowSorter.SortKey(0, SortOrder.ASCENDING)));
//Sort in ascending order
```

Figure 1: *Vocalulary List Sorter*

Vocabulary list uses Map¡String,Integer¿ Structure and created by iterating over hash tables, which is then visualized using JTable class in the Swing framework. The defaultModel table model is then used in the JTable class to store data.

JTable has TableRowSorter¡DefaultTableModel¿ Class can assist in setting the arrangement of data, by setting Sorter parameters so that users can click on the word or word frequency

header to switch between the ascending and descending order of various types

High-frequency words may represent commonly used words or topics in the data set, while low-frequency words may indicate specialized or infrequent concepts. Since the text is not strictly preprocessed, according to the observation of vocabulary list, most of the high-frequency words are not helpful for retrieval, that is, the so-called "stop words". To solve this problem, we need to remove stop words in the preprocessing stage to improve the quality of text features.

In this program, different hash functions (MD5, Modulo, SHA1) are implemented. By counting the length of the linked list in the hash table, the average value and standard deviation of the length of the linked list are calculated as follows:

1. MD5
    - Process Speed: Fast;
    - Average Length: 14.399086651960214;
    - Standard Deviation: 19.327345684417626;
2. SHA1
    - Process Speed: Fast;
    - Average Length: 14.3954100390882;
    - Standard Deviation: 19.27567308900792;
3. Modulo
    - Process Speed: Super Slow;
    - Average Length: 15.479352916134525;
    - Standard Deviation: 807.0180077865342;

The average length represents the average length of the linked list for each slot in the hash table. A smaller average length generally indicates better performance because it means the data is more evenly distributed across the hash table, reducing hash conflicts. Standard deviation measures the difference in the length of the linked list.

According to the calculated data, MD5 and SHA1 look similar and outperform Modulo and Multiplicative. The higher standard deviation of Modulo and Multiplicative indicates that the length of the linked list changes greatly, which may lead to more hash conflicts. Considering the average length and standard deviation, the performance of MD5 and SHA1 is relatively good, so the program chooses SHA-1 as the hash function.

Sorting linked lists by word frequency in descending order rather than just alphabetical order highlights the importance of frequently occurring words, sorting by frequency puts more relevant or important words at the top of the list, which are generally more important in the language and more likely to affect the meaning of the sentence, so placing them first

can produce more meaningful and relevant linguistic predictions. Not only that, ranking by word frequency can also improve search and retrieval efficiency, and high-frequency words are easier to access quickly when performing a search operation or retrieving data from a linked list sorted by frequency. High-frequency words often provide important context and understanding in a sentence, and prioritizing them better captures the context. Low-frequency words are usually noisier, and sorting the end of the list helps to potentially reduce noise in the data.

Task5

In unigrams, all words are assumed to be independent of each other. The total number of words in the corpus is obtained by adding the frequency values of all words, and then traversing the vocabulary to obtain the frequency of occurrence of a given word in the corpus. Finally, p(w1) is calculated by the frequency of a given word/the total number of words in the corpus.

To calculate the conditional probability based on bigrams (binary model), that is, to calculate the probability of the occurrence of word2 given the previous word word1. Firstly, the frequency of simultaneous occurrence of the bigrams words and the frequency of the previous word are obtained from the vocabulary, and then the divide them follow the formula p(wk— WK-1) = c(WK-1, wk)/c(WK-1), and finally the probability p(wk— wK-1) is obtained.

Similar to the calculation of bigrams, trigrams calculates the probability of the third word according to the occurrence of the first two words, obtains the frequency of trigrams and the frequency of bigrams respectively from the vocabulary, and then calculates the probability of the occurrence of the word when given the first two words.

For input "you have":

- 1-grams: "you have the the the the the the the the the the the the the the the the the the the the the the the the the the;"
- 2-grams: "you have to be a lot of the president clinton is a lot of the president clinton is a lot of the; "
- 3-grams: "you have to be a little bit of a sudden chris says jesus it's gunfire and i think that the president of; "

For input "this is one":

- 1-grams: "this is one the the the the the the the the the the the the the the the the the the the the the the the the the the; "
- 2-grams: "this is one of the president clinton is a lot of the president clinton is a lot of the president clinton is a;"
- 3-grams: "this is one of the united states and the other hand if you want to be a little bit of a sudden chris; "

For input ”today in sheffield”:

- 1-grams: ”today in sheffield the the the the the the the the the the the the the the the the the the the the; ”
- 2-grams: ”today in sheffield;”
- 3-grams: ”today in sheffield;”

For input ”in sheffield today”:

- 1-grams: ”in sheffield today the the the the the the the the the the the the the the the the the the the the;”
- 2-grams: ”in sheffield today the president clinton is a lot of the president clinton is a lot of the president clinton is a lot;”
- 3-grams: ”in sheffield today;”

In this example, when input ”today in sheffield”, neither 2-gram nor 3-gram can find similar words, which probably means that there is no combination of ”sheffield” or” in sheffield” in the data set. Therefore, the calculated conditional probabilities are all 0 and the next word cannot be predicted, which also indicates that the vocabulary in the data set is not diversified enough and the data behind some word sequences is insufficient. For the input of ”in sheffield today”, 2-gram only considers the previous word ”today” to predict, and the data set contains the combination of” today”, so 2-gram can predict the next word. The 3-gram considers the first two words, ”sheffield today,” unpredictable for the same reason that the combination does not exist in the aforementioned dataset.

For the 1-gram model, since it is independent and not affected by the previous words, it will always return the word with the highest frequency and therefore the highest probability in the word list, so that the predicted word is generally meaningless. The 2-gram calculates the probability of the occurrence of a bigrams word based on the occurrence of the previous word, and then takes the word with the greatest probability as the next word in the prediction. This pattern only takes into account the context of the current word, and ultimately results in phrases that may fall into repetition in the prediction. The 3-gram model takes into account a wider range of contextual information, which reduces the likelihood of repeating similar fragments, so that the predicted sentences are relatively more meaningful.

When N-grams are implemented with values of n larger than ’3’, the model may suffer from sparsity problems, resulting in many N-grams never appearing in the training data, resulting in the model not being able to accurately predict these sequences, and therefore requiring more data to train the model. At the same time, higher-order n-grams means more combinations and possible sequences, which increases the complexity and storage requirements of the model, and consumes a lot of time and resources in the process of handling model training and prediction.

[1] Messagedigest (java platform se 8), docs.oracle.com. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html