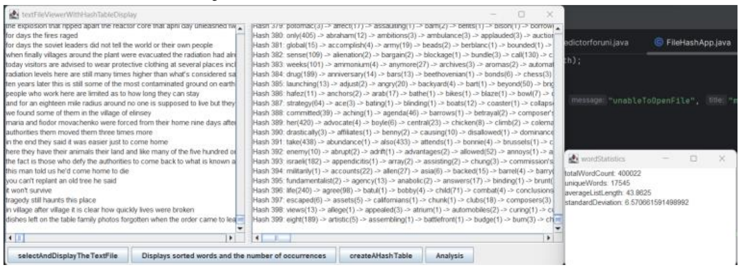
Part1:

First make sure the java version is JDK17



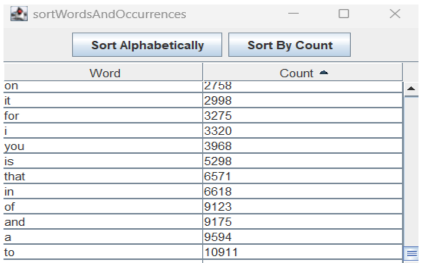
```
the explosion that ripped apart the reactor core that april day unleashed tw
for days the fires raged
for days the soviet leaders did not tell the world or their own people
when finally villages around the plant were evacuated the radiation had alr
today visitors are advised to wear protective clothing at several places incl
radiation levels here are still many times higher than what's considered sa
ten years later this is still some of the most contaminated ground on earth
people who work here are limited as to how long they can stay
and for an eighteen mile radius around no one is supposed to live but they
we found some of them in the village of elinsey
maria and fiodor movachenko were forced from their home nine days afte
authorities them moved them three times more
in the end they said it was easier just to come home
here they have their animals their land and like many of the five hundred o
the fact is those who defy the authorities to come back to what is known a
this man told us he'd come home to die
you can't replant an old tree he said
it won't survive
tragedy still haunts this place
in village after village it is clear how quickly lives were broken
dishes left on the table family photos forgotten when the order came to lea
```

```
Hash 379: potomac(3) -> affect(17) -> assaulting(1) -> barn(2) -> bells(1) -> bison(1) -> borrow
Hash 380: only(405) -> abraham(12) -> ambitions(3) -> ambulance(3) -> applauded(3) -> auction
Hash 381: global(15) -> accomplish(4) -> army(19) -> beads(2) -> berblanc(1) -> bounded(1) ->
Hash 382: sense(109) -> alienation(2) -> bargain(2) -> blockage(1) -> bundle(3) -> call(130) -> c
Hash 383: weeks(101) -> ammonium(4) -> anymore(27) -> archives(3) -> aromas(2) -> automat
Hash 384: drug(189) -> anniversary(14) -> bars(13) -> beethovenian(1) -> bonds(6) -> chess(3)
Hash 385: launching(13) -> adjust(2) -> angry(20) -> backyard(4) -> bart(1) -> beyond(50) -> bri
Hash 386: hafez(11) -> anchors(2) -> arab(17) -> bathe(1) -> bikes(1) -> blaze(1) -> bowl(7) -> c
Hash 387: strategy(64) -> ace(3) -> bating(1) -> blinding(1) -> boats(12) -> coaster(1) -> collaps
Hash 388: committed(39) -> aching(1) -> agenda(46) -> barrows(1) -> betrayal(2) -> composer's
Hash 389: her(420) -> advocate(4) -> boyle(6) -> central(23) -> chicken(8) -> climb(2) -> colema
Hash 390: drastically(3) -> affiliates(1) -> benny(2) -> causing(10) -> disallowed(1) -> dominance
Hash 391: take(438) -> abundance(1) -> also(433) -> attends(1) -> bonnie(4) -> brussels(1) -> c
Hash 392: enemy(10) -> abrupt(2) -> adrift(1) -> advantages(2) -> allowed(52) -> annoys(1) -> a
Hash 393: israeli(182) -> appendicitis(1) -> array(2) -> assisting(2) -> chung(3) -> commission's
Hash 394: militarily(1) -> accounts(22) -> allen(27) -> asia(6) -> backed(15) -> barrel(4) -> barry
Hash 395: fundamentalist(2) -> agency(13) -> anabolic(2) -> answers(17) -> binding(1) -> brunt(
Hash 396: life(240) -> agree(98) -> batul(1) -> bobby(4) -> child(71) -> combat(4) -> conclusions
Hash 397: escaped(6) -> assets(5) -> californians(1) -> chunk(1) -> clubs(18) -> composers(3)
Hash 398: views(13) -> allege(1) -> appealed(3) -> atrium(1) -> automobiles(2) -> curing(1) -> c
Hash 399: eight(189) -> artistic(5) -> assembling(1) -> battlefront(1) -> budge(1) -> bum(3) -> ch
```

```
totalWordCount: 400022
uniqueWords: 17545
averageListLength: 43.8625
standardDeviation: 6.570661591498992
```

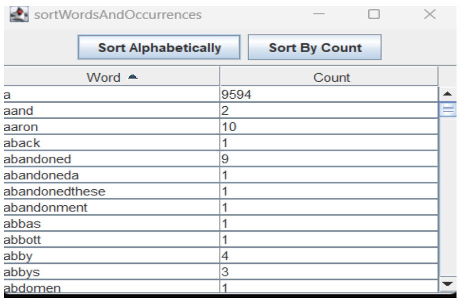
textFileViewerWithHashTableDisplay
selectAndDisplayTheTextFile
Displays sorted words and the number of occurrences
createAHashTable
Analysis
edictorforuni.java
FileHashApp.java
message: "unableToOpenFile",
title:
wordStatistics

Figure1： task4GUI



Word	Count ▲
on	2758
it	2998
for	3275
i	3320
you	3968
is	5298
that	6571
in	6618
of	9123
and	9175
a	9594
to	10911

Figure 2: Sort table GUI by Letter

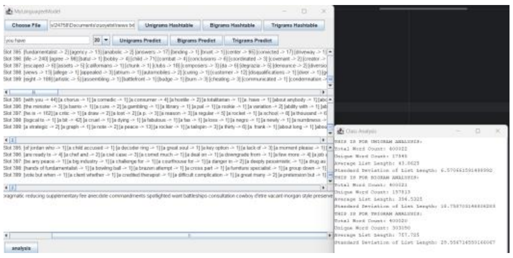


Word ▲	Count
a	9594
aand	2
aaron	10
aback	1
abandoned	9
abandoneda	1
abandonedthese	1
abandonment	1
abbas	1
abbott	1
abby	4
abbys	3
abdomen	1

Figure3: Sort Table GUI by Count

Click the button that says "selectAndDisplayTheTextFile" to choose the text file that you want to study. The software will then show you the text in the upper left window. Once you've chosen the file, hit the "createAHashTable" button to make a Word hash table from the text you chose. If you want to do text analysis, click the Analysis button. The data will show up in the lower right window. You can see summary statistics for text analysis in the wordStatistics window. These data are: This property tells you how many words are in the whole text. uniqueWords: The text's number of unique words. This is the average length of the connected list in the hash table. standardDeviation: The width of the linked list's range of values. The Sort by Count and Sort by Alphabetically buttons let you sort the hash table by words or the number of times they appear. The sortWordsAndOccurrences window will show the results of the sort.

Part2-1：



```
THIS IS FOR UNIGRAM ANALYSIS:
Total Word Count: 400022
Unique Word Count: 17545
Average List Length: 43.8625
Standard Deviation of List Length: 6.570661591498992
THIS IS FOR BIGRAM ANALYSIS:
Total Word Count: 400021
Unique Word Count: 157813
Average List Length: 394.5325
Standard Deviation of List Length: [illegible]
THIS IS FOR TRIGRAM ANALYSIS:
Total Word Count: 400020
Unique Word Count: 303050
Average List Length: 757.625
Standard Deviation of List Length: [illegible]
```

Figure 4: MylanguageModelGUI

To pick the text file you want to look at, click the Choose File button. The file's data will be shown in the text box in the upper left corner. Check out the hash table: After the text loads, you can see the one-element, two-element, and three-element hash tables, in that order: Bigrams Hashtable, Trigrams Hashtable, and Unigrams Hashtable Type your first set of words below the text box. To guess what will happen, use the following buttons: Unigrams Predict: A prediction model based on unigrams.

Bigrams Predict: Based on a prediction model for binary variables. Trigrams Predict: a guess based on the trigram model When you click the "Analysis" button, the software will look at the text that is currently loaded and the word sequence that you entered to find the most likely word sequence.

In the text box on the right, the research results will be shown. The Class Analysis window on the right shows statistical details of one-variate, binary, and ternary models, such as the number of words in each model, Linked list length, number of unique words, average linked list length, and standard deviation of linked list length.

Part2-2

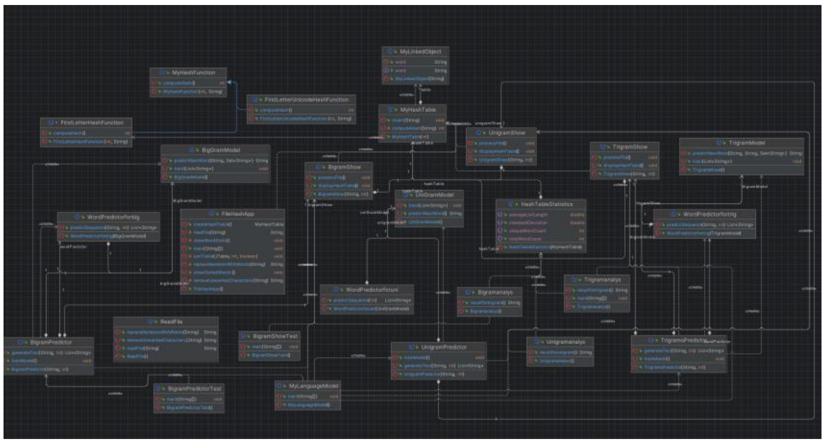


Figure 5: The UML for Whole java programming

FileHashApp:

ReadFile: This function reads the text of a file and does some preliminary work, like getting rid of extra characters, etc. MyHashTable: This class implements a hash table for saving and getting data. MyHashFunction is a utility class for figuring out hash values.

There is a hash function called FirstLetterUnicodeHashFunction that can hash the Unicode value of the first letter. To fix hash conflicts, MyLinkedObject uses a linked list structure.

MyLanguageModel:

The following classes make up the MyLanguageModel component: UnigramModel: This model is only for unigrams, which are single words. BigramModel: This language model is an expert in bigrams, which are groups of two words. TrigramModel: This language model is designed to handle trigrams, which are groups of three words. WorldPredictorforuni is a class of word prediction tools that are built on the unigrams model.

WordPredictorforbig is a class of tools for predicting words based on the bigram model.

WordPredictfortri is a class of word prediction tools that are based on the trigrams model.

UnigramPredictor: this class is used to guess text using a model of unigrams.

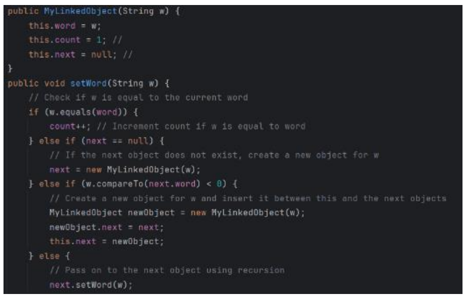
BigramPredictor: this class is used to guess text using the bigram model.

TrigramPredictor: this is used to guess text using a model of trigrams.

HashTableStatistics is a class for figuring out hash table statistics. It has methods for finding the standard deviation and average list length.

It can use the UnigramShow, BigramShow, and TrigramShow classes to show hash table data for models that are unigrams, bigrams, or trigrams.

Part3: Task1:



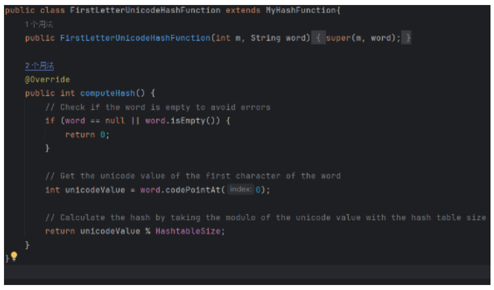
```java
public MyLinkedObject(String w) {
    this.word = w;
    this.count = 1; //
    this.next = null; //
}
public void setWord(String w) {
    // Check if w is equal to the current word
    if (w.equals(word)) {
        count++; // Increment count if w is equal to word
    } else if (next == null) {
        // If the next object does not exist, create a new object for w
        next = new MyLinkedObject(w);
    } else if (w.compareTo(next.word) < 0) {
        // Create a new object for w and insert it between this and the next objects
        MyLinkedObject newObject = new MyLinkedObject(w);
        newObject.next = next;
        this.next = newObject;
    } else {
        // Pass on to the next object using recursion
        next.setWord(w);
```

Figure 6: MyLinkedObject Class

To make a new MyLinkedObject instance, use this constructor. It takes a string w as an argument, which is the word. The first instance of the word is set to 1 when a new object is made, and next is set to null to show the next item in the linked list. Set the methodString w is the word. This method sets or changes the words in the linked list and how many times they appear. Check out the word right now: If the new word w is the same as the current object's word, the count value of the current object should be raised. Add a new word: There is no next object if next is null. In this case, a new MyLinkedObject is made for word w and linked to the current object. When w is a word that is less common than next in the dictionary, a new object is made and put between the current object and the next object. This is called an insertion sort.

Recursive call: The method will call the setWord method on the next item in the linked list if none of the other conditions are met. In other words, it goes down the list until it finds the right place.

Task2:



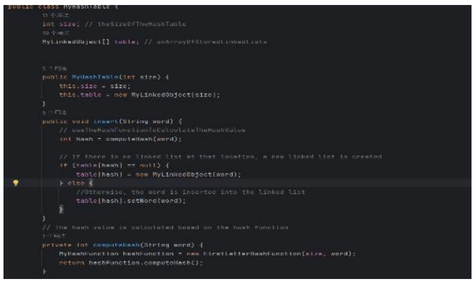
```java
public class FirstLetterUnicodeHashFunction extends MyHashFunction{
    1个用法
    public FirstLetterUnicodeHashFunction(int m, String word) { super(m, word); }

    2个用法
    @Override
    public int computeHash() {
        // Check if the word is empty to avoid errors
        if (word == null || word.isEmpty()) {
            return 0;
        }

        // Get the unicode value of the first character of the word
        int unicodeValue = word.codePointAt( index: 0);

        // Calculate the hash by taking the modulo of the unicode value with the hash table size
        return unicodeValue % HashtableSize;
    }
}
```

Figure 8: MyHashFunction



```java
public class MyHashTable {
    11个用法
    int size; // theSizeOfTheHashTable
    19个用法
    MyLinkedObject[] table; // anArrayOfStoredLinkedLists


    5个用法
    public MyHashTable(int size) {
        this.size = size;
        this.table = new MyLinkedObject[size];
    }
    5个用法
    public void insert(String word) {
        // useTheHashFunctionToCalculateTheHashValue
        int hash = computeHash(word);

        // If there is no linked list at that location, a new linked list is created
        if (table[hash] == null) {
            table[hash] = new MyLinkedObject(word);
        } else {
            //Otherwise, the word is inserted into the linked list
            table[hash].setWord(word);
        }
    }
    // the hash value is calculated based on the hash function
    1个用法
    private int computeHash(String word) {
        MyHashFunction hashFunction = new FirstLetterHashFunction(size, word);
        return hashFunction.computeHash();
    }
```

Figure 7:   FirstLetterUnicodeHashFunction

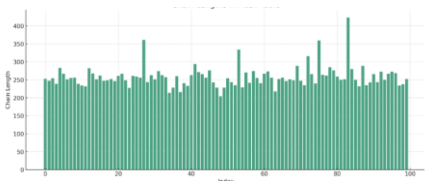


Figure 9: MyHashFunction Chart

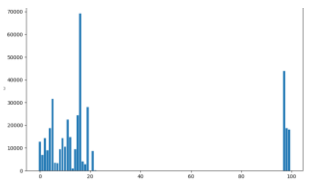


Figure 10 FirstLetterUnicodeHashFunction   Chart

The first letter hash function This class is based on MyHashFunction and uses Java's built-in hashCode method to figure out hash values. To find the hash value, it looks at the whole string and then modulos that value to make it fit the size of the hash table.

Advantage：Uniform Distribution: Because Java uses the hashCode method, it can usually come up with different hash values for each string. This makes the distribution more uniform.

A look at the FirstLetterUnicodeHashFunction. This class is also a subclass of MyHashFunction, but to figure out the hash, it only looks at the Unicode value of the string's first letter.

Advantage：This method is faster to determine because it only looks at the first character. This is especially true for long strings. However, the comparison chart shows that since only the first character is looked at, this could cause the hash numbers to be spread out unevenly. Especially if the first character of the data you give it doesn't change much, this way is more likely to make hashes. Problems with collisions, especially when a lot of lines start with the same letter.

Task3:

The parts of the hash table are stored in a MyLinkedObject array (table) by the MyHashTable class. To handle hash collisions, each entry can have a linked list.

The size of the hash table and the table array is set by the constructor.

You can add new words to the hash table with the enter method. First, the computeHash method is used to find the hash of the word. After that, the hash value is used to find the place in the table array. If the spot is empty, make a new MyLinkedObject node. If it is not empty, call the linked list's setWord method at this point to add the new word to the list.

The MyHashTable class mixes linked lists and hash functions by using MyLinkedObject to work with linked lists and FirstLetterHashFunction to figure out hash values. This makes the basic functions of a hash table possible. This lets MyHashTable keep track of words and how often they appear and handle possible hash collisions to keep data organized and speed up search. There are nodes in a linked list for each MyLinkedObject instance. If there is a hash collision, which means that two or more words have the same hash value, the nodes can be joined to make a linked list. This makes sure that all words in the hash table that have the same hash value are found in the same place.

Task4:

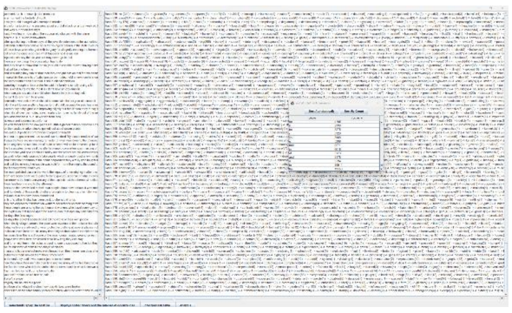
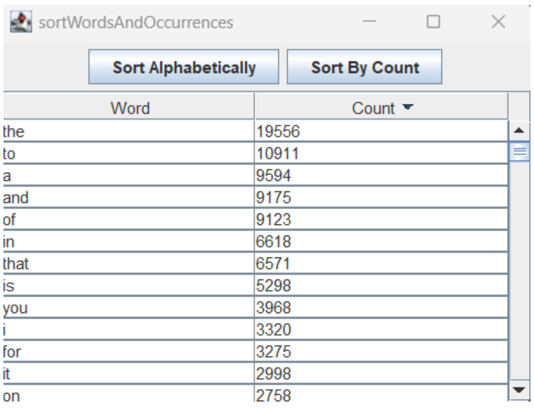


Figure 11: Task4 GUI



Word	Count
the	19556
to	10911
a	9594
and	9175
of	9123
in	6618
that	6571
is	5298
you	3968
i	3320
for	3275
it	2998
on	2758

Figure 12: SortWordsAndCount

To implement a comparison function, the function first compares the frequency of the words, and then compares the lexicographic order of the words if the frequencies are the same. The word list can then be rearranged using a sorting algorithm such as merge sort or quick sort, which can handle sorting of linked lists.

The distribution of word frequencies, such as particular words appearing exceptionally frequently or a large number of terms appearing only once, is one example of an observation. Misspelled words can also be seen, as well as the influence these errors may have on word frequency data.

Sorting by word frequency in descending order may be more effective for natural language processing tasks than alphabetical order because it offers faster access to high-frequency terms, which may be more significant for specific studies. Furthermore, it can expose the topic of the document or the author's usage habits.

Task 5: When calculating p(w1,w2.....wK) using unigrams and bigrams, p(w1) is the probability of the first word in the sequence. This probability can be calculated independently because it does not depend on other words in the sequence. Specifically, p(w1) can be obtained by calculating the frequency of word w1 in the corpus.

Calculate the probability: p(w1) is the frequency of word w1 divided by the total number of words. Expressed as a mathematical formula: p(w1) $=\frac{the\ total\ count\ for\ w1}{Total\ number\ of\ words\ in\ the\ corpus}$.

When p(w1,w2,...,wK) is calculated using up to trigrams (i.e. considering unigrams, bigrams and trigrams), p(w1) is still the probability of the first word in the sequence, while p(w2|w1) is the conditional probability that the second word w2 appears given the first word w1

p(w2|w1)$=\frac{bigrams(w2|w1)the\ number\ of\ occurence}{the\ number\ of\ occurence\ of\ w1}$.

You Have:Unigrams: you have heightens henstra russian k. expeditious raisin window less fletcher bigots joining anger fatiguing rescuing broussette brush choice owned.

Bigrams: you have me like dr. billy zane hits that villages often twice they speaks except that three had hopping of.

Trigrams: you have secondly a and graham kerr formerly great job so times he throughout the life seems once do you get quoted.

This is one :Unigrams: this is one reflected optic allow racking stockholder aligned arrests th parody realized curve paranoid ford accomplish bothered frost osos.

Bigrams: this is one jersey would overnight hostile buyout bid by carrying shown in but landed near the stuck give another.

Trigrams: this is one practical man whatever prayed in the leslie abramson will be born who be interviewed on c. even wait till Wednesday.

It can be found that when you have two words, bigrams output is the best, and then when this is one and three words are predicted, trigrams are the best prediction.

Today in Sheffield/in Sheffield Today: cant be predicted. Because "today in sheffield" and "in sheffield today" are specific phrases, they may not appear directly in the provided n-gram data.

Problem implementing n-grams (n > 3):

1. 1. Data sparsity: As n increases, the frequency of specific n-gram combinations in the corpus will decrease significantly, causing the data to become very sparse.
2. 2. Increased computational complexity: Longer n-grams require more storage space and computing resources to process.
3. 3. Decreased generalization ability: Longer n-grams may overfit to specific sequences in the training data, resulting in decreased generalization ability.

Reference:

Team, C. (2023). *What Is Hashing, and How Does It Work?* [online] Codecademy Blog. Available at: https://www.codecademy.com/resources/blog/what-is-hashing/.