COM6516: Object Oriented Programming and Software Design

Name: Vivek Shravan Tate, **Student Code:** ACP23VST

Part-1: User Manual for Application

1. **Main Screen:**

The GUI application consists of two main screens - the Main Screen and the Language Model Analysis Screen(As shown in image-01 & image-05). These screens include button panels and a split panel, each serving specific functionalities.

1. **Initial State:**
   - On application startup, all buttons in the button panel are disabled except for the "Load Document" button.
   - The split pane displays an enabled text area, and all other views are disabled (As shown in image-01).
2. **Document Upload:**
   - Click the "Load Document" button to upload a valid document from the file system.
   - The document data is displayed in the text area view upon successful upload (As shown in image-02).
3. **Processing Document:**
   - After document upload, the "Process Document" button and the Hash function selection dropdown become enabled.
   - Select a hash function from the dropdown and click "Process Document" to render the document in the vocabulary list table.
   - A warning message is displayed if unwanted words are detected and excluded from the vocabulary list table (As shown in image-03).
4. **Vocabulary Display:**
   - The vocabulary list table displays words and their counts.
   - The Main Screen buttons, including "Show Statistics," "Sort List," and "Run Language Model Analysis," become enabled(As shown in image-04).
5. **Sorting Options:**
   - The default sorting order is alphabetical, but the user can choose ascending or descending order based on word counts (As shown in image-04).
6. **Statistics Display:**
   - Clicking "Show Statistics" displays a detailed description on the far-right view using tables and bar graphs (As shown in image-04).
7. **Language Model Analysis:**
   - Clicking "Run Language Model Analysis" redirects the user to the Secondary Page.
   - All Main Screen data is retained.

2. Language Model Analysis Screen:

1. **Initial State:**
   - The "Run Language Model Analysis" button text changes to "Main Page."
   - "Show Statistics" and "Main Page" buttons are enabled.
   - The Language Model table is pre-loaded in the middle of the split view (As shown in image-05).
2. **Language Model Prediction:**
   - Enter words in the text box, select a language model from the dropdown, and click "Predict" to get the next 20 predicted words.
   - Stats related to the language model are displayed on the right view using tables and graphs upon clicking "Show Statistics"(As shown in image-06).
3. **Returning to Main Screen:**

- Clicking "Main Page" redirects the user to the Main Screen, retaining all uploaded and processed data.

Image-01

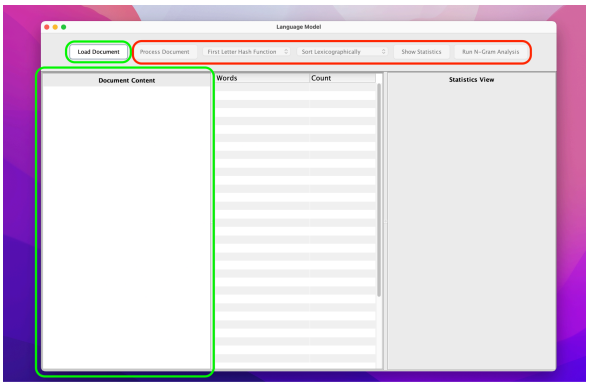


Image-02

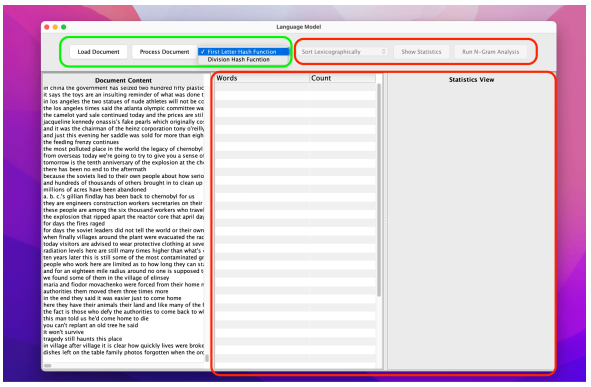


Image-03

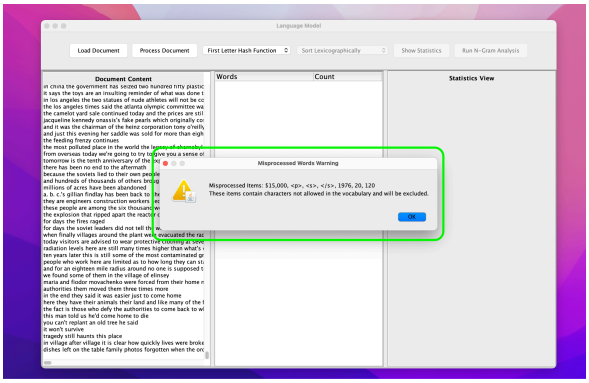


Image-04

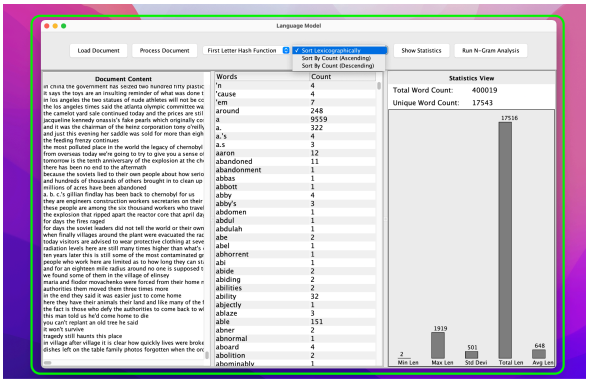


Image-05

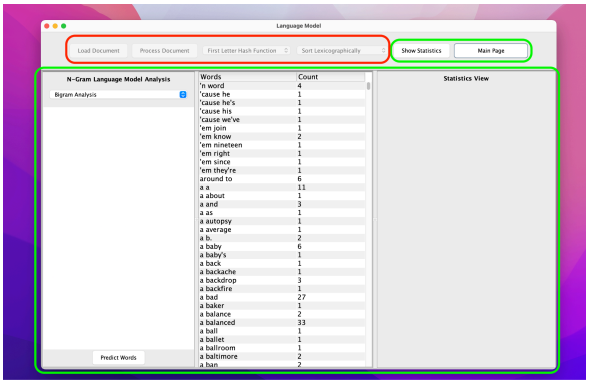
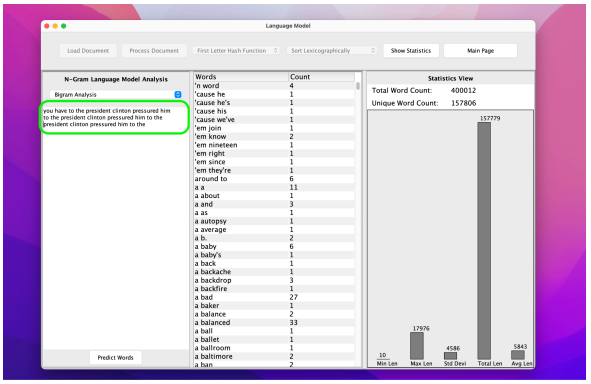


Image-06



Part 2:

1. **Data Abstraction:**
    a. Creating interfaces for HashFunction and LinkedListObject to hide implementation details and promote flexibility.
    b. Defined clear contracts for these interfaces, specifying expected behaviour.
2. **Encapsulation:**
    a. Made instance variables private, and provide public accessor/mutator methods to control access and ensure data integrity.
    b. Used constructors to manage object initialization and enforce invariants.

3. **Inheritance:**
    a. Created a hierarchy of HashFunction implementations based on different algorithms
    b. Created a base class for LinkedListObject to model a generic linked list node, allowing for adaptability in other contexts.

4. **Polymorphism:**
    a. Leverage the HashFunction interface to allow interchangeable hash functions within MyHashTable, enabling different hashing strategies without modifying core code.

Part 3 Detailed Description on Tasks:

1. **Task01- MyLinkedObject:**
    a. **Constructor:**
        i. Method Name: MyLinkedObject(String w)
        ii. Description: Initialises a new MyLinkedObject instance with the given word. Sets the word, count to 1, and initialises the next reference to null.
    b. **Methods:**
        i. **SetWord Method:**
            1. **Method Name:** public void setWord(String w)
            2. **Description:** Updates the linked list structure based on the input word. If the word already exists in the linked list, increments the count. If the word is greater than the current node's word and there's no next node, create a new node and set it as the next node. If there is a next node and the word is less than the next node's word, insert a new node before the next node. If the word doesn't match the current node's word and is greater than the next node's word, recursively calls the setWord method on the next node.
        ii. **getWordCount:**
            1. **Method Name:** private int getWordCount()
            2. **Description:** Returns the count of the word in the current MyLinkedObject instance. This method is private, indicating that external classes should not access it directly, and it's used internally to fetch the count.

2. **Task02 - MyHashFunction:**
    a. **First Letter Hash Function:**

        The first letter hash function algorithm calculates the hash code for a given word by extracting the Unicode value of its first letter and applying the modulo operation with the specified hash table size. The algorithm is succinctly expressed in the provided.Java code: **“return word.codePointAt(0) % hashTableSize;”**. Here, word.codePointAt(0) retrieves the Unicode value of the first letter, and % hashTableSize ensures the resulting hash code fits within the hash table's size.

    b. **Division Hash Function:**

        The Division Hash Function algorithm calculates the hash code for a given word by summing the Unicode values of all its characters and then applying the modulo operation with the specified hash table size. The algorithm is expressed in the provided Java code: **“return word.isEmpty() ? 0 : (word.chars().sum()) %**

hashTableSize;". Here, word.chars().sum() calculates the sum of Unicode values for all characters in the word, and % hashTableSize ensures the resulting hash code fits within the hash table's size.

3. **Task03 - MyHashTable:**

Yes, I have successfully achieved hiding of MLinkedObject and MyHashFunction class inside MyHashTable class by encapsulating the MyLinkedObject class and the MyHashFunction class within the MyHashTable class. Both nested classes are private and thus hidden from external access. Methods from the MyLinkedObject class and the MyHashFunction class are utilised through the MyHashTable class, aligning with the suggested design consideration.

This encapsulation enhances data hiding and ensures that the internal details of the linked object and hash function are not exposed to external code. Users interact with the hash table functionality provided by the MyHashTable class without direct access to its internal components, promoting a cleaner and more maintainable design.

4. **Task 4 - Vocabulary List:**
   a. **Describe your strategy for rearranging from the alphabetically ordered vocabulary list to the list with descending frequency order.**
      Yes, I have successfully achieved this by sorting the vocabulary list to descending frequency order by sorting the list on the basis of count of the word.

   b. **What observation(s) have you made from the vocabulary lists created?**
      In the vocabulary list created there are 400019 total number of words, 17543 Unique Words. The list also contains the un-processed data which is excluded while loading data.

   c. **Make comparison between hash function algorithms, e.g., statistics for lengths of individual linked lists**
      within the hash table. 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*.
      The Smallest linked object chain is of 2 length, Maximum Length of Linked list is 1919, Standard Deviation is 501, Average length of linked list is 648.

   d. **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.
      i. Sorting linked lists by the decreasing number of word occurrences in natural language processing tasks offers notable advantages. This approach facilitates efficient retrieval of high-frequency words, streamlines frequency-driven operations, and enhances statistical analyses of word distributions. Applications such as text summarization, sentiment analysis, and information retrieval benefit from quick access to relevant and impactful terms. Additionally, the prioritisation of common words supports streamlined text processing pipelines and enables the quick detection of unusual terms. The dynamic adaptation to changing word frequencies and optimised memory access patterns further contribute to improved overall performance in scenarios involving large and evolving text corpora.

5. **Task5 -**
    a. **What do you do with *p*(*w*1) when calculating *p*(*w*1,*w*2,...,*wK*) using unigrams and bigrams? Similarly, what do you do with *p*(*w*1) and *p*(*w*2|*w*1) when calculating *p*(*w*1,*w*2,...,*wK*) using up to trigrams?**

When calculating probabilities using unigrams (single words), bigrams (pairs of consecutive words), and trigrams (triplets of consecutive words), the probability expressions involve conditional probabilities based on the n-gram models. Let's break down the considerations for each case:

- Unigrams (p(w1, w2, ..., wK) using unigrams)

$$p(w_1, w_2, ..., w_K) = p(w_1) \times p(w_2) \times \ldots \times p(w_K)$$

- Bigrams (p(w1, w2, ..., wK) using bigrams):

$$p(w_1, w_2, ..., w_K) = p(w_1) \times p(w_2|w_1) \times p(w_3|w_2) \times \ldots \times p(w_K|w_{K-1})$$

- Trigrams (p(w1, w2, ..., wK) using trigrams):

$$p(w_1, w_2, ..., w_K) = p(w_1) \times p(w_2|w_1) \times p(w_3|w_1, w_2) \times \ldots \times p(w_K|w_{K-2}, w_{K-1})$$

    b. **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'.**
        i. **You have:** you have to the president clinton pressured him to the president clinton pressured him to the president clinton pressured him to the
        ii. **This is one:** this is one of the president clinton pressured him to the president clinton pressured him to the president clinton pressured him to the

    c. **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'.**
        i. **You have:** you have to go to the point that jim baker turned to president clinton to the point that jim baker turned to
        ii. **This is one:** this is one of the united states to make the point that jim baker turned to president clinton to the point that jim

    d. **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'.**

No, I was not able to find any words for today in Sheffield for bigram and trigram. Because the word sheffield is not present in the hash table.

    e. **Discuss any issue(s) of implementing *n*-grams with a larger value of *n* than '3'.**
        i. Implementing n-grams with larger values of n (beyond 3) introduces challenges such as increased sparsity, heightened computational complexity, and escalating memory requirements. The exponential growth in the number of possible n-grams leads to data sparsity issues, impacting accurate probability estimation. Managing the sensitivity to training data, potential lack of generalisation, and diminishing returns in performance become critical considerations. Efficient smoothing techniques and regularisation methods are essential to handle sparse data and mitigate overfitting. The choice of n requires careful consideration of the trade-offs between model complexity and the availability of sufficient, diverse training data to ensure effective language modelling.