### **Homework#5**

### **Word Grouping and Frequency Analysis**

Total Points: 90

Deadline: Dec 8th, 2024 (end of the day)

#### **Task:**

Given a text file, group all words by their starting letter. For each group:

1. Count the frequency of each word (using a **Hash Table** for fast lookup).
2. Store the unique words in a **BST** for efficient retrieval in sorted order.
3. Allow querying:
   * **Query 1**: Retrieve all words starting with a given letter in sorted order along with their frequencies.
   * **Query 2**: Find the most frequent word in a group (e.g., all words starting with 'a').

### **Steps:**

1. **Input**: A text file containing several lines of text.
2. **Process**:
   * Use a **Hash Table** to store groups of words based on their starting letter (a-z). Each group has its own BST for storing unique words and their frequencies.
   * Parse the text file, insert words into the appropriate group, and update the frequency.
3. **Output**:
   * For Query 1: Return the sorted list of words starting with a specified letter and their frequencies.
   * For Query 2: Return the most frequent word starting with a specified letter.

### **Challenges:**

1. **Efficiency**: Optimize the insertion and query processes for both data structures.
2. **Edge Cases**:
   * Handle mixed case letters (e.g., "Apple" and "apple" should be treated as the same word).
   * Handle non-alphabetic characters and empty lines.

### **Solution Sketch**

#### **1. Data Structures**

* **Hash Table**: Maps each starting letter (a-z) to a BST instance.
* **BST Node**: Each node stores a word and its frequency.

#### **2. Pseudocode**

**Processing the file**:

1. Open the text file and read each line.
2. Normalize text to lowercase and extract words using regular expressions.
3. For each word:
   * Determine the starting letter.
   * Insert the word into the corresponding BST in the hash table.
   * Update its frequency.

**Query 1: Words starting with a letter**:

* Retrieve the BST for the specified letter.
* Perform an in-order traversal to get sorted words and frequencies.

**Query 2: Most frequent word in a group**:

* Traverse the BST for the specified letter to find the node with the highest frequency.

### **Sample Input**

Create a file sample.txt with the following content:

Apple apricot banana banana banana

Cat dog apple apricot

Banana apple apple cat

### **Sample Output**

**Query 1: Words starting with 'a'**:

apple: 4

apricot: 2

**Query 2: Most frequent word starting with 'b'**:

banana: 4

### Another sample input file is attached with it pride\_and\_prejudice\_sample.txt

### **Additional Challenge**

Add support for finding the least frequent word in a group.

### **Rubric for Word Grouping and Frequency Analysis (100 points total)**

#### **1. Input Processing (10 points)**

* **5 points**: Correctly reads the text file and extracts words, ignoring case and handling non-alphabetic characters.
  + Full points: Uses regular expressions or equivalent methods for extracting words and handles edge cases like punctuation and mixed-case words.
  + Partial points: Fails to normalize case, or improperly splits words.
* **5 points**: Handles empty or malformed files gracefully without crashing.
  + Full points: Implements error handling for file reading and reports an appropriate error message.
  + Partial points: Crashes or produces incorrect output for empty or invalid input files.

#### **2. Binary Search Tree (BST) Implementation (25 points)**

* **10 points**: Correct insertion of words into the BST with frequency counting.
  + Full points: BST nodes correctly track word frequencies, and duplicates increment the frequency instead of creating new nodes.
* **10 points**: In-order traversal correctly returns words in sorted order with frequencies.
  + Full points: Output is sorted alphabetically and accurately reflects word counts.
* **5 points**: Correct implementation of the "most frequent word" query.
  + Full points: Returns the most frequent word efficiently.

#### **3. Hash Table Implementation (25 points)**

* **10 points**: Proper hashing and grouping of words by their starting letter.
  + Full points: Words are correctly assigned to groups based on the first letter.
* **10 points**: Efficient retrieval of groups using the hash table.
  + Full points: Retrieval is fast and accurate, using hash table lookups.
* **5 points**: Handles edge cases like words with no valid starting letter (e.g., numbers or symbols).
  + Full points: Such words are ignored or processed correctly without affecting other groups.

#### **4. Query Functionality (10 points)**

* **5 points**: Query 1 (Retrieve words starting with a letter in sorted order) works correctly.
  + Full points: Returns an accurate and complete list of words with frequencies.
* **5 points**: Query 2 (Find the most frequent word starting with a letter) works correctly.
  + Full points: Returns the word with the highest frequency efficiently.

#### **5. Efficiency (10 points)**

* **10 points**: Efficient algorithms for BST operations and hash table lookups.
  + Full points: Code demonstrates an understanding of the time complexities of BSTs and hash tables.

#### **6. Code Quality and Documentation (10 points)**

* **5 points**: Code readability and structure.
  + Full points: Code is well-organized, uses appropriate variable names, and avoids excessive complexity.
  + Partial points: Code is disorganized or uses unclear variable names.
* **5 points**: Comments and documentation.
  + Full points: Functions and classes are well-documented, and the purpose of each block of code is clear.
  + Partial points: Sparse or unclear comments.

### **Additional Bonus (10 points)**

* **10 points**: Implements additional features, such as:
  + Querying the least frequent word in a group.
  + Handling stop words (e.g., "the," "is") or providing options to exclude them.

**Scoring Breakdown**

| **Category** | **Points** |
| --- | --- |
| Input Processing | 10 |
| BST Implementation | 25 |
| Hash Table Implementation | 25 |
| Query Functionality | 10 |
| Efficiency | 10 |
| Code Quality and Documentation | 10 |
| **Total** | **90** |
| Bonus | **+10** |