ID: 12859191938
Name: Arda Tarım
Section: 1
Assignment number: 4

Problem Statement and Code Design

In this assignment I implemented a Trie structure and various methods inside it. I also created a DataRead helper class to read the data from the .txt file.

Implementation and Functionality

Main Class

In the main class I read the data from the .txt file and insert them into a Trie. Then the specified method is called from the Trie and output is returned.

```
switch (command) {
    // search the word in the trie
    case "search":
        boolean result = trie.Search(scanner.next().toLowerCase());
        System.out.println(result ? "True" : "False");
        break;

    // find words that start with the given prefix
    case "autocomplete":
        trie.autoComplete(scanner.next().toLowerCase());
        break;

    // find words that ends with the given suffix
    case "reverse":
        trie.reverseAutoComplete(scanner.next().toLowerCase());
        break;

    // find words that start with the given prefix and ends with given suffix
    case "full":
        trie.FullAutoComplete(scanner.next().toLowerCase(), scanner.next().toLowerCase());
        break;

    // find the top k most frequent words
    case "topk":
        trie.findTopK(scanner.nextInt());
        break;

    default:
        System.out.println(x:"Invalid command.");
        break;
}
```

Trie Class

This class includes two TrieNode's as root and reversedTrieNode (reverse of the Trie for some methods).

```
void insert(String word) {
    TrieNode current = root;

    for (char c : word.toLowerCase().toCharArray()) {
        current.children.putIfAbsent(c, new TrieNode());
        current = current.children.get(c);
    }

    current.endOfWord = true;

    // inserting to reverse Trie for reverseAutoComplete method
    TrieNode reverseTrieCurrent = reverseTrieRoot;

    String reverseWord = new StringBuilder(word).reverse().toString();
    for (char c : reverseWord.toCharArray()) {
        reverseTrieCurrent.children.putIfAbsent(c, new TrieNode());
        reverseTrieCurrent = reverseTrieCurrent.children.get(c);
    }

    reverseTrieCurrent.endOfWord = true;

    // keeping track of word counts for topK method
    wordCounts.put(word, wordCounts.getOrDefault(word, defaultValue:0) + 1);
}
```

insert(String) method inserts the given word to both Trie and ReverseTrie, and updates the wordCounts hashmap accordingly.

```
boolean Search(String arg) {
    TrieNode current = root;

    for (char c : arg.toCharArray()) {
        current = current.children.get(c);
        if (current == null) {
            return false;
        }
    }

    return current.endOfWord;
}
```

search(String) method traverses the Trie and returns True if the word is found in Trie.

```
void autoComplete(String prefix) {
    TrieNode current = root;

    for (char c : prefix.toCharArray()) {
        current = current.children.get(c);
        if (current == null) {
            System.out.println(x:"No words");
            return;
        }
    }

    LinkedList<String> list = findPossibleWords(current, prefix);

    // sort the words lexographicly before printing them
    Collections.sort(list);

    if (list.isEmpty()) {
        System.out.println(x:"No words");
        return;
    }

    System.out.print(String.join(delimiter:", ", list));
}
```

autoComplete(String) method finds all the words starting with a prefix. It uses a helper method to find possible words that start with a prefix.

reverseAutoComplete(String) is very similar to the autoComplete(String) method. It does the same thing using the reversedTrieNode as the root.

```
void FullAutoComplete(String prefix, String suffix) {

    TrieNode current = root;

    for (char c : prefix.toCharArray()) {
        current = current.children.get(c);

        if (current == null) {
            System.out.println(x:"No words");
            return;
        }
    }

    // find all possible words that start with the given prefix
    LinkedList<String> possibleWords = findPossibleWords(current, prefix);

    // filter words that end with the given suffix
    LinkedList<String> list = new LinkedList<>();
    for (String word : possibleWords) {
        if (word.endsWith(suffix)) {
            list.add(word);
        }
    }

    // sort the words lexographically before printing them
    Collections.sort(list);

    if (list.isEmpty()) {
        System.out.println(x:"No words");
        return;
    }

    System.out.print(String.join(delimiter:", ", list));
}
```

fullAutoComplete(String, String) method finds the possible words starting with the prefix and then it filters the words ending with the suffix.

```
void findTopK(int k) {
    // create a list to store the words and their counts
    List<HashMap.Entry<String, Integer>> wordCountsList = new ArrayList<>(wordCounts.entrySet());

    // sort the list based on the word counts
    wordCountsList.sort((a, b) -> {
        if (!a.getValue().equals(b.getValue())) {
            return b.getValue() - a.getValue();
        }
        return a.getKey().compareTo(b.getKey());
    });

    // create a new list to store the top k words
    List<String> list = new ArrayList<>();
    for (int i = 0; i < k && i < wordCountsList.size(); i++) {
        list.add(wordCountsList.get(i).getKey().toLowerCase());
    }

    if (list.isEmpty()) {
        System.out.println(x:"No words");
    } else {
        // sort the list
        Collections.sort(list);
        // print the list
        System.out.print(String.join(delimiter:", ", list));
    }
}
```

findTopK(int) method finds the top k words that have the most occurrences. It uses an ArrayList and a Comparator to sort the list.

DataRead Class

```
class DataRead {

    LinkedList<String> list = new LinkedList<String>();

    DataRead(String fileName) {
        try {
            Scanner scanner = new Scanner(new File(fileName));
            while (scanner.hasNext()) {
                list.add(scanner.next());
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println(x:"A file error occurred.");
        }
    }

    String getNext() {
        if (list.size() > 0) {
            return list.poll();
        } else {
            return null;
        }
    }
}
```

DataRead class is used to read the data from .txt files.

Testing

test.txt The Turkish Independence War was started when the first bullet was fired against the enemy during the occupation of Izmir by the Greek forces on 15 May 1919. Victors of the First World War, by signing of the Sevres Agreement, started the occupation of their shares of the Ottoman Empire. At the beginning, resistance against occupation forces started with militia forces, namely Kuva-i Milliye (National Forces). The Turkish Grand National Assembly established the regular army and successfully led the Independence War to victory by integrating the militia forces into the regular army.

test.txt search war	True (Expected output)
test.txt autocomplete Tur	turkish (Expected output)

test.txt reverse g	during, integrating, signing (Expected output)
test.txt full a st	against (Expected output)
test.txt topk 4	by, forces, of, the (Expected output)

References

Algorithms, FOURTH EDITION, Robert Sedgewick and Kevin Wayne