# CMPE 343
# Programming Homework 4

Halil Mert Güler

Section 2

Assignment 4

# Question 1

## 1. Problem Statement and Code Design

The task is to implement a Trie data structure in Java language with functions to able to search on the trie with given string, counting prefixes for every word and lastly suffix finding features.

**Sub tasks**

- Understanding the task: According to the problem, we should create a java code mainly using Trie data structure and necessary functions that takes input from the user and generates output according to question.

- Choose the best algorithm for the task: We are using a Trie data structure and we want to perform search, prefix and suffix operations so we can use Node implementation for the trie.

- Create the Trie data structure: Trie data structure has created with Trie Node implementation. Trie Node structure has variables "children" to check connected entities, "isEndOfWord" to check the last node as the end of the word, and count for finding count of nodes.

- Generating a code part to take input accordingly: The code part that takes inputfrom user should be in String form. After number of inputs has taken, we should take that count of inputs and show the options.

- Implementation: After we create our road map, we should start the implementation with creating "Main.Java" class that includes runnable function,the Trie class, TrieNode class and necessary algorithms.

- Testing: At this part we will test our program with the according input and checkthe result.

## 2. Implementation and Functionality

For the implementation part, "Main.java" class with runnable function created at first.
After that, the Trie and TrieNode classes was implemented.

```java
class Trie
    private TrieNode root;
    public ArrayList<String> words = new ArrayList<String>();
    public Trie() {
        root = new TrieNode();
}
```

```java
class TrieNode {
    public Map<Character, TrieNode> children;
    public boolean isEndOfWord;
    public int count;

    public TrieNode() {
        children = new HashMap<>();
        isEndOfWord = false;
        count = 0;
    }
    public Map<Character, TrieNode> getChildren() {
        return children;
    }
    public boolean isEndOfWord() {
        return isEndOfWord;
    }
    public void setEndOfWord(boolean endOfWord) {
        isEndOfWord = endOfWord;
    }
    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }}
```

The TrieNode class has an Map<Character, TrieNode> value “children” that represents the child nodes of the current TrieNode. It stores the mappings between characters and their corresponding child TrieNode objects, “isEndOfWord” is a boolean variable that indicates whether the current TrieNode represents the end of a word. When inserting a word into the trie, the last character's TrieNode is marked as the end of the word., “count” to return count of nodes, a constructor for class and getter functions.

The Trie class represents a Trie data structure, which is used for efficient string storage and retrieval operations. It has “root” which is a TrieNode object that serves as the root of the trie, “words” is a String ArrayList variable that hold inserted words to use for other operations next of the code and a constructor.

After that, the Main function that takes input, generates output and initializes the Trie is created. The main function serves as the entry point of the program, where it interacts with the user, performs operations on the Trie data structure, and manages the flow of the program.

- Initialize the Trie and Scanner objects.
- Read the number of inputs.
- Insert words into the trie using a loop.
- Present the available function options.
- Read the user's choice and perform the corresponding operation.
- Close the scanner.

After input management is done, necessary functions have generated.

- **Boolean search (String arg):** This function searches for a word in the trie. It starts from the root node and iteratively traverses through each character of the word. If any character is not found in the trie, the function returns false, indicating that the word does not exist. If all characters are found and the last node is marked as the end of a word, the function returns true, indicating that the word exists in the trie.

- **Void countPrefix(Trie trie):** This function counts the number of words with a given prefix in the trie. It starts from the root node and iteratively traverses through each character of the prefix. If any character is not found in the trie, it means there are no words with the given prefix, and the function returns 0. If the prefix is found, it returns the count of words stored in the corresponding TrieNode, excluding the word itself.

- **Void reverseFind (String suffix):** This function searches for words in the trie that end with a given suffix. It initializes a boolean variable found to keep track of whether any word with the given suffix is found. It calls the reverseFindHelper() method, passing the root node, suffix, an empty string, and the found variable as parameters. The reverseFindHelper() method recursively searches for words by appending characters to the current word and checking if the node represents the end of a word and ends with the suffix. If a word is found, it is printed, and the found variable is set to true. If no word with the suffix is found, the function prints "No result".

## 3. Final Assessments

- What were the trouble points in completing this assignment?
  The trouble points were to implement the reverseFind function.

- Which parts were the most challenging for you?
  Same with above.

- What did you like about the assignment? What did you learn from it?
  This assignment helped to learn basic functions and the general construction of the Trie data structure.

# Question 2

## 1. Problem Statement and Code Design

The task is to implement a string sorting algorithm class using java language.

**Sub tasks**

a. Understanding the task: According to the problem, we should create a java code that takes input from user in string form and process them to generate a sorted array in desired way.

b. Create necessary functions: We need to construct two arrays and sort the first array according to the second array if the same index distances are even. If distance is odd, we will sort in ascending order. Therefore, we need functions to take strings and find numerical values, calculate distances, statements to proceed in desired way and sorting.

c. Generating a code part to take input accordingly: The code part that takes input from user should be in String form. After number of inputs has taken, we should split them and send the words to arrays.

d. Implementation: After we create our road map, we should start the implementation with creating "Main.Java" class that includes runnable function and necessary algorithms and functions.

e. Testing: At this part we will test our program with the according input and check the result.

## 2. Implementation and Functionality

For the implementation part, "Main.java" class with runnable function created at first. After that, a method created to find numerical value of words.

```java
    private static long getAlphabeticalValue(String str) {
        StringBuilder valueBuilder = new StringBuilder();
        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (Character.isLetter(c)) {
                int charValue = Character.toLowerCase(c) - 'a' + 1;
                valueBuilder.append(charValue);
            }
        }
        String valueString = valueBuilder.toString();
        long value = Long.parseLong(valueString);
        return value;
    }
```

This function's return type is long because when calculating the value of a word it can exceed the integer limit (for example: word monkey). It takes a string parameter and find the alphabetical value of it in string form and at last it converts it to long value.

After that, a function to find distances has generated.

```
    public static long calculateStringDistance(String str1, String str2) {
        long value1 = getAlphabeticalValue(str1);
        long value2 = getAlphabeticalValue(str2);
        return Math.abs(value1 - value2);
    }
```

This function simply gets difference of same indexed word's numeric values with absolute.

After, a function created to check values are even or odd, this function is used in an if-else statement to proceed.

```
String[] sorted = new String[length];
```

An array for holding the sorted values of the first array has created.

```
for (int i = 0; i < length; i++) {
            long distance = calculateStringDistance(arr1.get(i).toLowerCase(),arr2.get(i).toLowerCase());
            if (isValueEven(distance)) {
                sorted[i] = sortWithPriority(arr1.get(i), arr2.get(i));
            } else {
                sorted[i] = sortLexicographically(arr1.get(i));
            }
        }
```

In this part, it's important to send words with lower case to check when calculating the distance. After that, if the distance is even, final array's according to index will be set with priority sorting result. If the distance is odd ascending order will be applied to the word.

```
   public static String sortWithPriority(String word, String prior) {
        List<Character> charList = new ArrayList<>();
        List<Character> remainingChars = new ArrayList<>();
        for (char c : word.toCharArray()) {
            if (prior2.indexOf(c) != -1) {
                charList.add(c);
            } else {
                remainingChars.add(c);
            }
        }
        charList.sort(Comparator.comparingInt(prior::indexOf));
        charList.addAll(remainingChars);

        StringBuilder sortedString = new StringBuilder();
        for (char c : charList) {
            sortedString.append(c);
        }

        return sortedString.toString();
    }
```

The custom priority base sorting algorithm takes two string parameters which the first one is the first arrays word and the second one is the second arrays prior word. This function sorts a given string based on a specified priority order. Characters that match the priority order are placed first in the sorted string, followed by the remaining characters in their original order.

```
       System.out.println("First Array:");
        for (String word : arr1)
            System.out.print(word + " ");

        System.out.println("Second Array:");
        for (String word : arr2)
            System.out.print(word + " ");

        System.out.println("Sorted Array:");
        for (String word : sorted)
            System.out.print(word + " ");
```

Finally, code print the first array, second array, and sorted array.

## 3. Final Assessments

a. What were the trouble points in completing this assignment?
The trouble points were didn’t think about the value of a word can exceed the integer limit.

b. Which parts were the most challenging for you?
The most challenging part was to create a function to get numeric value of a word, and performing custom sorting especially when it comes to the remaining part of the word.

c. What did you like about the assignment? What did you learn from it?
This assignment helped to learn string sorting and complex sorting algorithms.