# 03-60-254: Data Structures and Algorithms
**Lab Assignment 1: Word Count I**

## 1 Due date

In your lab section during the week starting September 24. To be submitted 15 minutes before the end of your lab section. You can also submit in the labs one week earlier.

## 2 Regulations

This assignment must be done individually. Any similarity found between your code/answers and another student's, or a carbon-copy of an answer found in the web will be considered cheating. For the part where coding is required, you must implement your own program.

## 3 Objective

The aim of this assignment is to understand the performance difference of Array and LinkedList. Get prepared for more efficient algorithms.

## 4 The problem

Your task is to count the frequency of words in a text file, and return the most frequent word with its count. For example, given the following text,

```
there are two ways of constructing a software design one way
is to make it so simple that there are obviously no deficiencies
and the other way is to make it so complicated that there are no
obvious deficiencies
```

Your program should printout the following along with the milliseconds to finish the computing:

```
The most frequent word is "there" with 3 occurrences.
```

## 5 The algorithms

**Algorithm Based on LinkedList** Algorithm 1 is based on the data structure LinkedList. It maintains a list for word frequencies. The algorithm runs by scanning every token in the input stream, and incrementing its corresponding frequency count.

**Algorithm using Arrays** Algorithm 1 is very slow. It can be improved in many different ways. Algorithm 2 is one possible improvement. It uses two arrays to keep track of the counts.

A template program written in Java is listed below in Appendix. It can be downloaded by right-clicking **this**. You can also use other programming languages of your choice, such of Python, C, C++.

```
Input: Array of string tokens
Output: The most frequent word and its frequency
begin
    wordFreqList =empty;
    foreach token in the input do
        for i =1; i<wordFreqList.length; i++ do
            if wordFreqList.get(i) equals token then
                increment the freq of the word in wordFreqList;
            end
        end
        if token not in wordFreqList then
            insert token into the wordFreqList with freq=1;
        end
    end
    Find the most frequent word from wordFreqList;
end
```

**Algorithm 1:** Use LinkedList

```
Input: Array of string tokens
Output: The most frequent word and its frequency
begin
    Initialize wordArray and countArray;
    for each token in the input array do
        if token in wordArray with index i then
            increment countArray[i];
        end
        else
            find j the last position of wordArray;
            wordArray[j]=token;
            countArray[j]=1;
        end
    end
    Find the most frequent word;
end
```

**Algorithm 2:** Use Array

## 6 Tasks

1. **Demonstrate the performance of the algorithms** You are required to run these two algorithms, and observe their performance differences. We prepared a set of test datasets of different sizes. You can click the following data sets to download them. Try to run the data sets in increasing order up to the point when it is unbearably long.

   **200 lines** Link to dblp200

   **500 lines** Link to dblp500

   **1k lines** Link to dblp1k]

   **5k lines** Link to dblp5k]

   **10k lines** Link to dblp10k]

   **100k lines** Link to dblp100k]

   **1m lines** Link to dblp1000k]

   **3.6m lines** Link to dblp]

   Note that you can download those files using unix commands like:

   ```
   wget http://cs.uwindsor.ca/~jlu/254/dblp200.txt
   ```

   Report what is the biggest data you can run.

2. **Analyze the complexities of the algorithms** Use the asymptotic analysis to give the complexity of these two algorithms.

3. **Improve Algorithm 1** Algorithm 1 is very slow because of the *get*($i$) method. Improve this algorithm while keeping the LinkedList data structure. A hint is to avoid to use *get*($i$) method while iterating through the list.

## 7 Submission

1. Your assignment must be submitted during the lab session in the section you are registered in. Any submission within the last 15 minutes before the end of the lab session will not be accepted and a zero mark will be given. Late assignments will be given a zero mark. Submissions by email will not be accepted.

2. Provide the source code for your programs (for Task 3 only).

3. Run your programs during the lab for various data sizes. Print out the count (most frequent word and its occurrences) and the performance (run time in millisecond). Lab instructor will test your code with new data.

4. Explain complexity of the algorithms in Big Oh notation.

## 8 Bonus Mark

This part is optional, and worth one mark. The task is to improve the algorithms to have a better performance. You need to demonstrate the improvement by side-by-side comparison with Algorithm 2. You need to show that empirically it is *obviously* better.By 'obvious' we mean that it uses only a fraction of the time in terms of millisecond compared with other algorithms. The submission process is the same as the other part, i.e., you need to show the source code, demonstrate the result, and explain the reasons.

## 9 Marking Scheme

The total marks for this assignment is 5+1, 1 is the bonus mark. Your marks will be calculated using the following 'algorithm':

```
Input: Your assignment
Output: Your score
begin
    yourScore=0;
    foreach two algorithms do
        if run correctly for several data sets then
            yourScore++;
        end
        if Explain complexity correctly then
            yourScore++;
        end
    end
    if Improve count_LINKED_LIST then
        yourScore++;
    end
    if Bonus is done then
        yourScore++;
    end
end
```

**Algorithm 3:** Marking scheme

## 10 Appendix: Code template

Here is the code template written in Java. You can download the code by clicking on **this**. Note that you may need to adjust the program, e.g., when running large data, CAPACITY may need to be increased.

```java
import java.io.File;
import java.util.Scanner;
import java.util.Map.Entry;
import java.util.AbstractMap;
import java.util.LinkedList;

public class WordCountLinkedList254 {
        public static Entry<String, Integer> count_ARRAY(String[] tokens) {
                int CAPACITY = 10000;
                String[] words = new String[CAPACITY];
                int[] counts = new int[CAPACITY];
                for (int j = 0; j < tokens.length; j++) {
                        String token = tokens[j];
                        for (int i = 0; i < CAPACITY; i++) {
                                if (words[i] == null) {
                                        words[i] = token;
                                        counts[i] = 1;
                                        break;
                                } else if (words[i].equals(token))
                                        counts[i] = counts[i] + 1;
                        }
                }

                int maxCount = 0;
                String maxWord = "";
                for (int i = 0; i < CAPACITY & words[i] != null; i++) {
                        if (counts[i] > maxCount) {
                                maxWord = words[i];
                                maxCount = counts[i];
                        }
                }
                return new AbstractMap.SimpleEntry<String, Integer>(maxWord, maxCount);
        }

        public static Entry<String, Integer> count_LINKED_LIST(String[] tokens) {
                LinkedList<Entry<String, Integer>> list = new LinkedList<Entry<String, Integer>>();
                for (int j = 0; j < tokens.length; j++) {
                        String word = tokens[j];
                        boolean found = false;
                        for (int i = 0; i < list.size(); i++) {
                                Entry<String, Integer> e = list.get(i);

                                if (word.equals(e.getKey())) {
                                        e.setValue(e.getValue() + 1);
                                        list.set(i, e);
                                        found = true;
                                        break;
                                }
                        }
                        if (!found)
                                list.add(new AbstractMap.SimpleEntry<String, Integer>(word, 1));
                }

                int maxCount = 0;
                String maxWord = "";
                for (int i = 0; i < list.size(); i++) {
                        int count = list.get(i).getValue();
                        if (count > maxCount) {
                                maxWord = list.get(i).getKey();
                                maxCount = count;
                        }
                }
                return new AbstractMap.SimpleEntry<String, Integer>(maxWord, maxCount);
        }

        static String[] readText(String PATH) throws Exception {
                Scanner doc = new Scanner(new File(PATH)).useDelimiter("[^a-zA-Z]+");
                int length = 0;
                while (doc.hasNext()) {
                        doc.next();
                        length++;
                }

                String[] tokens = new String[length];
                Scanner s = new Scanner(new File(PATH)).useDelimiter("[^a-zA-Z]+");
                length = 0;
                while (s.hasNext()) {
                        tokens[length] = s.next().toLowerCase();
                        length++;
                }
                doc.close();

                return tokens;
        }

        public static void main(String[] args) throws Exception {

                String PATH = "dblp200.txt";
                String[] tokens = readText(PATH);
                long startTime = System.currentTimeMillis();
                Entry<String, Integer> entry = count_LINKED_LIST(tokens);
                long endTime = System.currentTimeMillis();
                String time = String.format("%12d", endTime - startTime);
                System.out.println("time\t" + time + "\t" + entry.getKey() + ":" +
                    entry.getValue());

                tokens = readText(PATH);
                startTime = System.currentTimeMillis();
                entry = count_ARRAY(tokens);
                endTime = System.currentTimeMillis();
                time = String.format("%12d", endTime - startTime);
                System.out.println("time\t" + time + "\t" + entry.getKey() + ":" +
                    entry.getValue());
        }
```