CSL 201 Data Structures
Lab 1 Due on August 14, 11.55pm

Instructions

- You are to use Java as the programming language. Use Eclipse as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. You are encouraged engage in high level discussions with your classmates; however ensure to include their names in the report. If you refer to any source on the Internet, include the corresponding citation in the report. If we find that you have copied code from your classmate or from the Internet, you will get a straight F grade in the course.
- The submission must be a zip file with the following naming convention - rollnumber.zip. The Java files should be contained in a folder named after the question number.
- Include appropriate comments to document the code. Include a `read me` file containing the instructions on for executing the code. The code should run on institute linux machines.
- Upload your submission to moodle by the due date and time. Do not email the submission to the instructor or the TA.

The goal of this lab is to get you familiar with Java programming language and environment. We will be using Java for all the labs. Hence it is important that you spend a couple of weeks getting familiar with Java. This lab consists of 5 programming questions. You have to write code for all of them.

1 Smart Imposition (5 points)

When I was young, my typing instructor teacher used to give imposition as punishment whenever I made a typing mistake. I always wondered if there was a smart way to complete the imposition to make it look natural by randomly inserting a fixed number of spelling mistakes. I could not use copy paste as it gives the same result every time and the outcome does not appear natural. Write a simple Java function which takes as input the sentence to be repeated, the number of repetitions and the number of errors and outputs the sentence the specified number of times with specified number of random errors. The program must number each sentence and it should make a certain number of random typos so that it looks like typed by a human. However there can be only one error in each repetition.

Input

I will always use OOP programming
5
3

Output

1 I will always use OOP programming
2 I will always use OaP programming
3 I will klways use OOP programming
4 I will always use OOP programmisg
5 I will always use OOP programming

2 Toppers in CSL201(5 points)

Write a class that maintains information about students securing the top ten ranks in CSL201. Student information consists of student id and rank. Use a doubly linked list to maintain this information that is sorted on basis of rank. You can refer to Code Fragments 3.16 and 3.17 in the textbook for the starter code. The student information and their percentile rankings can be obtained from the standard input.
You have to first create a class `StudentRecord` that provides the interface for a student record with the attributes student id and rank. Define and implement the constructors, `set` update and `get` access methods for this class. You will also implement comparators `isEqual`, `isGreater` and `isLesser` that will be helpful while sorting the list of student records based on rank.
The input to the program will be in the following format `1 id_val rank`. 1 indicates that the record has to be inserted (if possible) into the list. After every insert, the program should output the current list of top ten rankers. The program is exited with `0` as the input. Below is sample input and output of the program.

Input	Output
1 Ram 10	Ram 10
1 Rahul 20	Ram 10 Rahul 20
0	

3 Recursion (5 points)

Given an unsorted array, A, of integers and an integer k, write a recursive Java program for rearranging the elements in A so that all elements less than or equal to k come before any elements larger than k. Write the pseudocode of the program and determine the running time of your algorithm on an array of n values.
The elements of A and the value of k are obtained from the standard input. The number of elements in the array should be the first input, followed by the elements of the array and finally the value of k.
Input
8 1 5 10 7 6 9 8 2 5
7 1 2 5 7 6 9 8 4
Possible Output
1 5 2 10 7 6 9 8
1 2 5 8 7 6 9

4 The Matrix (5 points)

In this problem, we will conduct an empirical (experimental) analysis of a widely used function - matrix multiplication. Write a Java program for multiplying two $n \times n$ randomly generated matrices. Given n, the size of a matrix, your program must perform 20 matrix multiplications, each time on two random matrices of size n. Run experiments to find the average time required for multiplication of the matrices for varying values of $n = 5, 10, 20, 30, 40, 50$. Use GNU plot to see the trend of growth rate in time with n. Analyze the result and make a small report.

5 Term Frequency-Inverse Document Frequency Computation (10 points)

Term frequency-inverse document frequency (tf-idf) is a numerical statistic that quantifies the importance of a word to a document in a corpus. It is a very popular tool for document analysis and is

commonly used as a mechanism to characterize documents. The tf-idf value of a word is proportional to the number of times a word appears in a document, but is offset by the frequency of the word in the corpus. This helps to adjust for the fact that some words appear more frequently in general.
tf-idf is the product of two statistics, term frequency and inverse document frequency. While there are different ways to compute these statistics, we will use the following definitions. Let t denote a word or a term and d denote a document.

- Term frequency $tf(t, d)$: The number of times the term t occurs in the document d, i.e., the frequency of term t in the document d is represented as $f(t, d)$. Then term frequency can be computed as
$$tf(t, d) = 0.5 + 0.5\frac{f(t, d)}{\max f(t, d)}$$
- Inverse document frequency $idf(t, d)$: It is the logarithmically scaled fraction of the documents in the corpus that contain the term t. This is obtained as follows
$$idf(t, d) = \log \frac{N}{|\{d \in D : t \in d\}|}$$
where
  - N is the total number of documents in the corpus
  - $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears (i.e., $tf(t, d) \neq 0$). In case this value turns out to be 0, to avoid division by 0, treat the value as 1.

  Mathematically the base of the log function does not matter and constitutes a constant multiplicative factor towards the overall result.

For example, consider two documents, d_1 and d_2 containing the following contents respectively

- d_1 - `Data structures are cool concepts. I love programming using intelligent data structures.`
- d_2 - `Data structures are fun to program. Data structures are useful in programming games. There are different kinds of data structures.`

$$\begin{aligned} f(\texttt{structures}, d_2) &= 3 \\ tf(\texttt{structures}, d_2) &= 0.5 + 0.5\frac{3}{3} = 1 \\ idf(\texttt{structures}, d_2) &= \log\frac{2}{2} = 0 \\ tfidf(\texttt{structures}, d_2) &= 0 \end{aligned}$$

The goal of this question is to implement a program that takes as input a list of document names and computes tf-idf statistic for each document. The text documents that have to be taken as input are provided in the zip file under the folder data. Ignore the following words that may appear in the documents - `a, an, the, this, that, and, or, is, are, then, there, they, of, on, in` and `to` while computing the statistic. You must ignore all punctuation marks. The terms should not be case sensitive. The output of the program must be a matrix $tfidf$, where each element of the matrix $tfidf(t, d)$ corresponds to the tf-idf value of the t^{th} term and d^{th} document. The terms and documents correspond to the columns and rows of the matrix. You must consider the union of terms occurring in all documents of the corpus for computing the statistic characterizing a single document.

6 References

1. http://en.wikipedia.org/wiki/Tf-idf
2. M Goodrich, R Tamassia, and M. Goldwasser, "Data Structures and Algorithms in Java", 6^{th} edition, Wiley, 2011.