CS 6103D Software Systems Laboratory

Lab Test 1 **25.10.2021**

General Instructions

- The evaluation consists of two parts - PART A and PART B. You will be allowed to proceed to PART B ONLY if you complete PART A and submit the code in EduServer.
- Design:
  - Write the design ***only*** for **PART A** and submit it in the EduServer **before 2:20 PM**.
  - No need to write a design for **PART B**.
- Implementation:
  - Implement **PART A,** make sure that your program works correctly for the given sample I/O and submit code for PART A in the EduServer **before 3:00 PM**. **If you need more time for completing PART A, you may request your instructor for the same.** But you shall be permitted to proceed to PART B ONLY if you complete the coding for PART A before **3:30 PM** and in that case, you have to complete PART B before **4:15 PM**.
  - After submitting PART A, you may inform the instructor that you have submitted and then proceed with coding for PART B.
  - No need to submit the source code in EduServer for **PART B**. You should complete the coding for PART B before **4:00 PM** and get the result verified by your evaluator before **4:30 PM.**

Mark Distribution:

Maximum Marks - 10 marks

Design - 2 marks

Viva - 1 mark

Implementation - 7 marks (Part A - 4 + Part B - 3)

Part A

Given a text file, create a list *L* of words in the file. Each element in the list should contain two attributes - the word *w* and its *frequency count* (the number of times the word occurs in the text file).

You have an option to choose any of the following data structures for implementing the list (maximum marks for implementation for each option is given in brackets):

- *Array of structures* (2 marks)
- *Singly Linked List* (3 marks)
- *Binary Search Tree* (4 marks)

Define the following functions:

1. *getFrequencyCount(w, L)*
   Returns the frequency count of *w* stored in the list *L*.
2. *getWordswithMultipleOccurrence(L)*
   Returns the list of those words in *L* that occur more than once in the text.

Design: Write the algorithm (in pseudocode) for the two functions *getFrequencyCount(w, L)* and *getWordswithMultipleOccurrence(L)*

Input Format

The input text should be read from a file '*input.txt*' which consists of a set of words each separated by a space.

Output Format

- In each line, print the words that occur more than once in the text, along with its frequency count separated by a space. This should be done by invoking *getWordswithMultipleOccurrence()* and then printing each word *w* in the returned list along with the frequency count of *w* obtained by invoking *getFrequencyCount()*

Sample Input (file *input.txt*)

The two-year post-graduate programme in Computer Science and Engineering is intended to train the students in both advanced areas in the core courses and specialized topics in the emerging technology fronts Courses offered include Algorithms and Complexity Compiler Design Foundations of Information Security Distributed Computing and Pattern Recognition The project work in the final year is intended to equip the students to go deeper into her/his area of specialization the curriculum is organized with few core courses and many electives to give the students enough choice

Output

```
the 9
in 5
```

```
and 5
is 3
intended 2
to 4
students 3
core 2
courses 3
of 2
```

Part B

Represent the course prerequisite information using a **Directed Acyclic Graph (DAG)**. A course may have zero or more other courses as prerequisites. Represent the DAG using an adjacency list. Each course is to be represented as a vertex in the graph. A directed edge from vertex *x* to vertex *y* indicates that course *x* is a prerequisite for course *y*. **Implement the following method for topological sort of a DAG, given in CLRS Exercise 22.4-5.**

Repeatedly find a vertex say *v* of in-degree 0, print *v*, and remove *v* and all its outgoing edges from the graph. (in-degree of a vertex *v* is the number of edges entering *v*). Note that removal of an edge *(u, w)* will decrease the in-degree of *w*. You may use the Queue data structure to keep vertices of in-degree 0.

Input Format

The input file consists of multiple lines.

- The first line contains an integer *n>0*, the number of courses in a semester (equal to number of vertices in the DAG).
- The next lines contain an integer *m*, the number of edges in the DAG.
- The next *n* lines contain the *code* of the *n* courses.
- The next set of lines may contain a pair of strings representing course codes, *code1 code2* indicating that course *code1* is a prerequisite for course *code2* (and hence a directed edge from vertex *code1* to vertex *code2*)

Output Format

Display an order in which the courses can be credited, in a single line, with a space separating the course codes.

Testcase 1 (file *input.txt*)

```
4
3
ZZ1004D
CS2002D
CS2006D
CS2005D
ZZ1004D CS2002D
CS2002D CS2006D
CS2006D CS2005D
```

Output

```
ZZ1004D CS2002D CS2006D CS2005D
```

Testcase 2 (file *input.txt*)

Input

```
3
0
CS6101D
CS6111D
CS6102D
```

Output

Print the vertices in any order each separated by a space.

Testcase 3 (file *input.txt*)

Input

```
4
3
ZZ1004D
CS2002D
CS2006D
CS2005D
ZZ1004D CS2005D
CS2002D CS2005D
CS2006D CS2005D
```

Output

```
CS2002D CS2006D ZZ1004D CS2005D
```

OR

The three codes CS2002D CS2006D ZZ1004D in any order followed by CS2005D as the last in the listing.