Assignment overview

This lab includes a few programming exercises to gain experience writing algorithms using set and map implementations from the Java Collections Framework.

This lab requires Java programming. You may (and should!) discuss the lab and coding techniques with your classmates, but all of the code you submit to Learning Hub must be your own.

READ THE TIPS AND FAQS SECTION BELOW FOR HELPFUL ADVICE!

Submission information

Due date: As shown on Learning Hub. Late assignments will not be graded.

What to submit:

- Java source file
- Screen print of your output

Please do not zip or compress your submissions.

Grading

This lab is graded on a 10-point scale. Point breakdown:

- Part 1: Word frequency [2 points]
- Part 2: Distinct words [3 points]
- Part 3: Bad excuses [5 points]

Part 1

Write a program that uses a Map to count the number of times each word appears in the Beatles song “All You Need is Love”.

Input

- love.txt – Lyrics of the Beatles song

Note that punctuation has already been removed from the input file, but it is still mixed case and has whitespace. “Love” and “love” should be considered the same word.

Output

The output from your program should be sorted in alphabetical order, and should look something like this:

```
all - 20
be - 7
but - 2
can - 11
...(etc.)
```

Part 2

Write a program that uses a HashSet to determine if all the words in a file are distinct (i.e. different from each other). Your program should print `DISTINCT` or `NOT DISTINCT` based on the contents of the file.

Input

- q2input.txt – a list of words that may or may not be distinct

Output

Either “`DISTINCT`” or “`NOT DISTINCT`”, based on the contents of the input file.

Part 3

A BCIT instructor is having a problem with students giving fictitious excuses when they are late with their homework. In order to reduce the amount of time spent listening to fictitious excuses, the instructor has asked you to write a program that will search a list of excuses for keywords to help identify the fictitious excuses.

Input

- q3test.txt

Input is from a file, with a single test case per file.

The first line contains exactly two integers: K (which defines the number of keywords), followed by E (which defines the number of excuses).

The next K lines each contain exactly one keyword, and the last E lines each contain exactly one excuse.

Keywords will be formed of contiguous lower case alphanumeric characters (ie: no white space or punctuation). Excuses are strings of blank delimited lower case words. The words in the excuses contain only lower case alphanumeric characters.

Output

Your algorithm will print the worst excuse(s), one per line, formatted exactly as read in. We define the worst excuse(s) as the excuse(s) which contains the largest number of keywords.

If a keyword occurs more than once in an excuse, each occurrence of the keyword is counted. A keyword "occurs" in an excuse if and only if it exists in the string in contiguous form and is delimited by the beginning or end of the line or a space.

Note: If there is more than one worst excuse, they must all be printed in lexicographical order. Duplicate excuses are only printed once.

Sample Input

```
4 5
goat
ate
homework
canary
this excuse is so good that it contains no keywords
```

```
a goat ate my homework
the canary ate my goat
my canary was killed by a goat
can you believe goat died
```

Sample Output

```
a goat ate my homework
the canary ate my goat
```

Tips and FAQs

- It's easy reading input with the Scanner class in Java. You'll need to put this in a try/catch for IOException:

```
File fileIn = new File("inputfile.txt");
Scanner sc = new Scanner(fileIn);
while (sc.hasNext()) {
    String str = sc.next();
    System.out.println(str);
}
sc.close();
```

- I did Q1 fairly easily with a TreeMap. Some potentially useful methods: get, put, remove, replace. You might not need all of these; it depends on your algorithm.
  However, you may also be able to use a different map implementation (I didn't try).
- Here's a code snippet for printing out the contents of a TreeMap with an iterator. The entrySet() method transforms the contents of a TreeMap into a set of map entries:

```
for (Map.Entry<String, Integer> entry : myTmap.entrySet()) {
    System.out.println(entry.getKey() + " - " + entry.getValue());
}
```

- You can also use the Scanner class to read individual words out of a string!