Due date: 11:00pm Friday October 19, 2007

Introduction

Unsolicited email (spam) is annoying and clutters your mailbox. You are to write a *spam classifier* - a program that reads email messages of regular ASCII characters (32-127) and assigns each message a rating that indicates the likelihood that the message is spam.

How can you assign a message a rating? Spam contains words and phrases that are not common in genuine email messages. For example, the phrase

```
MAKE MONEY FAST
```

You are to implement a classifier that scans each message for common spam phrases. Each phrase is assigned a numerical value that indicates a *spam weight* for that phrase. For example, the phrase `MAKE MONEY FAST` might be assigned a spam weight of 20, while another phase `MEET HOT SINGLES` might be assigned a weight of 31. The *spam rating* for a message is the sum of the weights of all phrases found in the message.

For example, if a message contains both `MAKE MONEY FAST` and `MEET HOT SINGLES`, it would have a spam weight of 51 (using the weights from above).

Unfortunately, spammers are pretty tricky. They like to mix up the words in the phrases, and include special characters in an attempt to foil the spam filters. This means that permutations of phrases, such as `MAKE FAST MONEY`, or `HOT SINGLES MEET` should also be matched and included in the spam rating, and the case of characters in the message should be ignored.

Your classifier should also ignore whitespace and word separators when matching phrases to messages. For example, the following lines in messages should all generate a match with the spam phrase `MEET HOT SINGLES`:

```
HOT_SINGLES_MEET
HOTSINGLESMEET
S.I.N.G.L.E.S.M.E.E.T.H.O.T
[TAB]meet___HOT___SiNgLeS!!
```

Program Input

Input is read from a file.

The first line of input contains 2 positive integers: $m \leq 1000$, the number of spam phrases in the spam dictionary, and $n \leq 10000$, the number of messages to be scanned. Next there will be m lines, each containing a spam phrase (a string of up to 5 blank delimited words consisting of lower case letters [ASCII 97-122] and/or the special characters [ASCII 33-42], followed by an integer spam value v $(0<v<100)$. For example:

```
make money fast 20
```

Following the spam phrases is a line with a single colon (`:`), followed by the n messages to be scanned. Each message consists of exactly one line of text, which is comprised of words and word separators. Each message contains up to 100 words made from valid ASCII characters (32-127).

Words are delimited by word separators. Valid word separators include any whitespace character (`isWhitespace(char ch)`), or the dot "`.`", dash "`-`", or underscore "`_`" characters. Messages are separated by lines containing a single colon (`:`).

Program Output

Output is to the console.

One line containing the *spam rating* is printed for each message in the input. Output lines must be displayed in the order the messages occur in the input. No additional headings, punctuation, or comments are to be output.

Sample Input

```
7 6
haha 3
pharmacy 1
hot single 4
make money fast 13
increase your size 7
! 21
now is the time 55
:
Make money fast and MEET HOT SINGLE COWBOYS!!
:
Do you want to increase your size?    Now is the time!
:
Single hot people waiting for you. Don't be shy. It is the time to meet hot singles.
:
The_time_is_now - P.H.A.R.M.A.C.Y!!!!
:
There is no hot money for single people.
:
ha ha ha
```

Sample Output

```
59
83
8
140
0
6
```

Other Requirements and Guidelines

1. You may work in pairs, or by yourself. If you work with someone else, each person is expected to contribute equally to the assignment. Reports of partner-exploitation should be directed to the course instructor. The names and ID's of both partners must be included as per the submission guidelines below.
2. For this assignment you must use Java, and your program must compile using the Java 6 release.
3. Your program must read the input from a text file. The name of the text file will be passed as the only command line argument (as per the following submission requirements).
4. You are free to use any of the data structures or algorithms in the Java Collections Framework, or you can develop your own structures and algorithms if you prefer.
5. You are expected to implement and use both a pattern matching algorithm and a permutation generating algorithm in your solution. You will be graded (partly) on the sophistication and efficiency of your algorithms.
6. Your program will be evaluated against instructors test files, which will not be provided in advance. It is up to you to understand the requirements and make sure your solution meets them.
7. Programming style, design, technique, and inline documentation are up to you. This is not a course in software engineering or software design, however, you are expected to write code that can be easily understood by the instructor. Also, since this is an algorithm course, we will be looking at the overall efficiency and sophistication of the algorithms and structures that you use.
8. Each source code file in your submission must include a header block listing the author(s), date, and description of the file contents.
9. These requirements are subject to change. Any clarifications or changes will be announced in WebCT, and documented in a new version of this file. It is your responsibility to monitor WebCT so that you are aware of any assignment updates or changes.

Submission Details

The assignment must be submitted to WebCT before: ***11:00pm Friday October 19, 2007.*** WebCT will be configured such that assignments cannot be submitted after this time.

Your submitted assignment will consist of a single zip file. When your submitted zip file is unzipped, it should create a directory with your real name(s) (initial & last name). For example, if I was working with Albert on the assignment, unzipping will create a directory called: rneilson_awei (or something similar).

Within this directory I expect to find:

1. An MS-Word 2003 compatible document that includes:
    a. A short (one to two page) description of the algorithms and data structures used to solve the problem (including why you chose to design your program the way you did).
    b. A short (one to two) page analysis of the worst case time-efficiency of your program. In your time-based analysis you should analyze each of the algorithms used in your solution. Your evaluation must take into account the efficiency of any Java collections data structures or algorithms used in your solution.
2. A directory containing all the Java source files required to build your program. Your submission must not include java SDK files, and must meet the following criteria:
    a. Your "main" function must be in a class called Main, in a package called "spam", and in a subdirectory called "spam" under the directory created when your submitted file is unzipped. In the above example, this means main should be in the file rneilson_awei\spam\Main.java
    b. Note that packages correspond to subdirectories. This means you need to put the line: package spam; in Main.java. You will probably need to do the same with the other source files.

How do I test your assignment?

1. I don't use NetBeans. I just compile your program using javac. Your program must compile fine when I use the following commands to compile it:

    javac spam\Main.java (assuming, for the given example, I'm in the directory rneilson_awei).

2. Next I will execute your program, passing the name of one of my input files, and redirecting the output to a file. For example:

    java spam\Main a1data.txt > rneilson_awei_a1data_out.txt

Grading and Evaluation

Your program will be graded according to the marks distribution shown on the next page. The instructor's test data is designed to make sure you understood and followed the requirements. Don't assume anything!

Within each category, marks will be assigned for choice of data structures and algorithms, as well as correctness and efficiency of the solution(s).

Programs that do not compile will receive a grade of zero.

Programs that are submitted late will receive a grade of zero.

Programs that do not read the input file (ie: crash on input) will receive a grade of zero.

Programs that do not accept the file name as a command line argument will receive a maximum grade of 13/100.

Mark Distribution

Program Design 20%

The program design portion of your grade will be based on (but not limited to) things like

- Choice of algorithms and data structures.
- Modularity, readability, and overall organization of the program.

Program Efficiency 30%

The program efficiency portion of your grade will be based on (but not limited to) things like

- Algorithm used to create phrase permutations.
- String pattern matching algorithm.
- Time-based efficiency analysis is complete, understandable, and correct.
- Overall efficiency of Solution (more efficient solutions receive higher grades).

Correctness of Solution 50%

The correctness portion of your grade will be based on (but not limited to) things like

- Program runs without error / exception.
- Correct permutations are generated.
- Correct phrases are matched.
- Processing of word separators.
- Formatting of output.

Summary of Changes to this File

- none - this is v1 of the file