CSE 303, Winter 2006, Assignment 7
Due: Friday 10 March, 9:00AM

Last update: 18 February

Summary: You will write a Java program and a bash script to test the effectiveness of your homework 6 solution (concluding that it is not good). This is an *individual* assignment. You will perform a "takeout validation" test like this:

- There are two directories of files, the "test directory" and the "control directory".
- Divide the test-directory files into random groups of 5 (with one smaller group of leftovers if necessary).
- For each group, take the files out of the test-directory and run `spam_detector` on the two directories and each file in the group (i.e., run `spam_detector` 5 times or maybe fewer for the last group). Then put the files back in the test-directory.
- Record how well `spam_detector` does at "guessing" which directory the files came out of. (With the listMail example directory as the "test directory" and the spamMail example directory as the "control directory" your instructor found that depending on the random groups, the program got between 1 and 3 of the 33 right, meaning we would be better off flipping a coin. But negative results are better than no results.)

Requirements:

- Write a Java program that takes two command-line arguments, a directory d and a number n, and prints to stdout all the files in d, with n on each row (and possibly less than n on the last row). You should not print `.` or `..` (though this won't be a problem — see advice for what libraries to use). What files are on what line should be *random*; you should expect different output every time you run your program. (See advice for how to create a random permutation.) Print an appropriate error-message to stderr if the command line has two few arguments, the first is not an accessible directory, or the second cannot be converted to a positive integer.
- Write a bash script `takeoutTester` that takes two command-line arguments (first the "test directory" second the "control directory") and performs the "takeout test" putting 5 files in each group. A skeletal solution is provided; complete the skeleton without changing the code provided. Sample output is also provided: You should print out a group of filenames, then the result of `spam_detector` for each file in the group, then the next group of filenames, and so on. Finally you should print the total number right and the total number of files in the test directory.
- Extend your homework-six Makefile with two new targets: (1) for building the Java code as necessary and (2) a "phony" target `run` for running `takeoutTester` using the listMail example directory as the "test directory" and the spamMail example directory as the "control directory".

Advice:

- For the Java program:
  - The sample solution uses 3 static methods (as a matter of style; 1 would have worked) and is about 55 lines.
  - Use the `java.io.File`, `java.util.Random`, and `java.lang.Integer` classes, particularly the `nextInt`, `parseInt`, and `list` methods. A documention source for the Java standard library is: `http://java.sun.com/j2se/1.5.0/docs/index.html` in particular the "lang&util" and "I/O" parts of the library.
  - Catch a `NumberFormatException` error; ask if you have never caught exceptions before.

  - Here is one easy way to take an array and switch the order of the elements so you have a random permutation. *Use this algorithm; one you make up is probably wrong.* For each position i of the array from 0 to the right in order, swap the contents of i with the contents of a random position $j \leq i$ (if $j == i$ obviously the swap does not do anything, but that's okay).
  - Create a random permutation of the files and then print out the file names, putting spaces and newlines in as appropriate.

- For the bash script:
  - The sample solution is 45 lines including provided code.
  - Follow the "TO DO" comments in the provided skeleton.
  - There may be many ways to store the i^{th} line of a file in a shell variable; the sample solution uses `sed` and back-quotes for a simple one-line solution.
  - Store the result line of running `spam_detector` in a variable using back-quotes.
  - To determine if `spam_detector` thought the test-directory was closer, use `grep` (sending output to `/dev/null`) and then examine `grep`'s exit code.
- For the Makefile: Your phony target should have one or more sources, but it should not itself be a source for `all`, i.e., you should have to type `make run` to run a test.

Extra Credit: Do all the following:

- Extend `spam_detector` to support at least 5 distance functions (if you did the extra credit in homework 6, you will already have most of this).
- Extend `takeoutTester` to use 5 for the file-per-line only if there is no third command-line argument (else use the value of this argument). It can also take a fourth argument for how to run `spam_detector`.
- Write a program (in any language) called `allApproachesTester` that uses `takeoutTester` as follows:
  - Like `takeoutTester` it takes two directories
  - It runs `takeoutTester` with the directories, but for every combination of group-size (from 1 to the number of files in the test directory) and distance function.
  - It stores all the results in comma-separated format suitable for pasting into a spreadsheet. For example, one line might look like `4,0,3,33` indicating that the group size was 4, distance-function 0 was used (you can number them however you want), and 3 of 33 files were correctly categorized.
- Make a graph displaying your results (group-size on the x-axis, percentage-right on the y-axis, one line for each distance function). You can turn in a spread-sheet file and/or a picture holding your graph.

Assessment: Your solutions should be:

- Correct Java and bash code
- In good style, including indentation and line breaks
- A Makefile that rebuilds fairly precisely what is necessary
- Of reasonable size and efficiency

Turn-in:

- Turn-in all source code, including the `.c` and `.h` files for homework 6. Do not turn in compiled code.
- The grader should be able to type `make run` and have all the code compile and one test run. (Of course, typing `make run` again will run another test but not require any compilation.)
- Do not turn in any example files.
- Use `turnin` for course cse303 and project hw7. If you use late-days, use project hw7late1 (for 1 late day) or hw7late2 (for 2) instead of hw7.