The University of Chicago
Department of Computer Science

CMSC 16200 – Honors Introduction to Computer Science 2
Winter Quarter 2013
Lab #2 (01/28/2013)

Exercise 1: Using awk to grok a mail server log

In this exercise, you are provided with a log file from a mail server running the Postfix SMTP server (http://www.postfix.org/), the Amavis mail filter (http://www.amavis.org/), and the SpamAssassin SPAM filter (http://spamassassin.apache.org/). The log includes only the messages generated by Amavis, and all real e-mail addresses have been replaced by fake addresses. You do *not* need to read any official Amavis documentation to do this exercise, as the log format is explained in this handout (the URLs given above are only for those who want to find out more about mail servers).

The goal for this exercise is to process the mail log using `awk` to produce a report with interesting statistics, such as top e-mail recipients, number of SPAM messages received, etc.

Amavis log format

Like most UNIX server logs, the Amavis log is extremely verbose. However, we will only be concerned with some specific pieces of information contained in the log. Although you can safely ignore everything else, you will still need to know how to extract the correct information from the log.

When the mail server receives an e-mail, Amavis will print out ***several*** lines to its log. For the purposes of this exercise, we will assume that e-mails cannot arrive concurrently (i.e. if Amavis is writing to the log and another e-mail arrives, it won't start writing the log lines for the new e-mail until it finishes processing the current e-mail).

The first two lines contain some information we need:

```
Dec 25 00:21:39 mailhost amavis[8544]: (08544-08) SMTP::10024
/var/amavis/tmp/amavis-20051224T225515-08544: <j.random.hacker1@foobar.com> ->
<stoehr@cs.uchicago.edu> Received: from mailhost.cmsc16200.net ([127.0.0.1]) by
localhost (server.cmsc16200.net [127.0.0.1]) (amavisd-new, port 10024) with
SMTP id 08544-08 for <stoehr@cs.uchicago.edu>; Sun, 25 Dec 2005 00:21:39 -0600
(CST)

Dec 25 00:21:39 mailhost amavis[8544]: (08544-08) Checking:
<j.random.hacker1@foobar.com> -> <stoehr@cs.uchicago.edu>
```

The first two lines simply acknowledge that an e-mail has been received and will be processed. These lines state who the sender and the recipient are (this information can also be extracted from other lines). All lines start with the date that line was written to the log.

Next, Amavis prints out a variable number of lines which provide more details regarding how the e-mail is being processed. We will be interested in the ***last*** line generated by an e-mail delivery. From this line we can find out if the e-mail was blocked for being SPAM, if the e-mail was not SPAM (believe it or not, “ham” in e-mail terminology), and if the e-mail was infected with a virus.

If the e-mail was ham, then the last line will look like this:

```
Dec 25 02:59:40 mailhost amavis[27201]: (27201-09) Passed CLEAN,
[66.249.82.193] <j.random.hacker1@foobar.com> -> <stoehr@cs.uchicago.edu>,
Message-ID: <bbcbea3f0512250109l5116d765x4f3280d7b905fff7@foobar.com>, Hits:
-2.532, 7553 ms
```

If the e-mail was SPAM, then the last line will look like this:

```
Dec 25 00:21:44 mailhost amavis[8544]: (08544-08) Blocked SPAM, [78.55.0.41]
<j.random.hacker1@foobar.com> -> <stoehr@cs.uchicago.edu>, quarantine: spam-
31dd2a7f0d4d955488739a6b5b0cc6da-20051225-002139-08544-08, Message-ID:
<5006865622.7984468453@foobar.com>, Hits: 10.629, 5052 ms
```

When a message is flagged as SPAM, it is assigned a score (generated by SpamAssassin).

If the e-mail was infected with a virus, then the last line will look like this:

```
Dec 25 15:18:27 mailhost amavis[8036]: (08036-04) Blocked INFECTED
(Worm.SomeFool.P), [62.15.113.205] <j.random.hacker1@foobar.com> ->
<stoehr@cs.uchicago.edu>, quarantine: virus-20051225-151825-08036-04, Message-
ID: <20051225212829.D8CC3A323B@foobar.com>, Hits: -, 1382 ms
```

Report generation

Based on the information contained in the log, you must generate a report with the following information:

- 10 top senders
- 10 top receivers
- # of ham e-mails
- # of SPAM e-mails
- Average SPAM score (∑ SPAM Scores / # SPAM messages)
- SPAM to Ham ratio (# ham / # SPAM)
- Infected e-mails

The report could look like this:

```
10 top senders
--------------
xs.random.hacker@foobar.com: 175 emails
es.random.hacker@foobar.com: 115 emails
ae.random.hacker@foobar.com: 66 emails
[e-mail hidden]: 40 emails
cn.random.hacker@foobar.com: 30 emails
og.random.hacker@foobar.com: 29 emails
io.random.hacker@foobar.com: 24 emails
ot.random.hacker@foobar.com: 21 emails
sn.random.hacker@foobar.com: 17 emails
se.random.hacker@foobar.com: 15 emails

10 top recipients
-----------------
ba.random.hacker@foobar.com: 1932 emails
aa.random.hacker@foobar.com: 75 emails
pr.random.hacker@foobar.com: 2 emails

Stats
-----------------
Ham: 563
SPAM: 1412
Average SPAM score: 14.0758
SPAM to Ham ratio: 2.50799
Infected: 12
```

Suggested order

This `awk` problem is longer than the `awk` examples you have seen in class, and you should not try to solve the entire problem at once. A good strategy to solve programming problems, and specially text processing problems, is to start solving a subset of the problem, and then slowly building up and adding features until you reach the desired result.

We suggest that you do the exercise in the following order:

- Start by printing out everything except the top 10 senders and recipients.
- Keep track of senders and recipients in your script. However, when printing the report, you can print *all* the senders and recipients in any order.
- Finally, make sure you only print the top 10 senders and recipients.

Notes

- Even though the above section suggests that you perform your work in three steps, this doesn't mean you have to hand in three separate files. Include all your work on a single `report.awk` file.
- Do *not* hand in the mail server log.
- Using the provided mail log, your report should look like the one shown on the previous page (depending on the thoroughness of your code, there might be small variations with regards to the sample report).
- You can turn your `awk` script into an executable program simply by adding the following line at the beginning of the script:

```
#!/usr/bin/awk -f

BEGIN  { ... }

END    { ... }

etc.
```

Setting the file as executable (`chmod u+x`, as described in lab #1) will allow you to run your awk script directly from the shell like this:

```
./report.awk maillog
```

Exercise 2: Text distance

In natural language processing, two texts can be compared to determine how similar they are according to multiple criteria. In this exercise, we will simplify the definition of “distance” between two texts, in such a way that it will be possible to compute this distance using a Python script.

Before computing the distance, you will need to generate the following information for each text:

- *Word frequencies*. Given the text, compute the frequency of each word. This gives a set of (word,frequency) pairs, which we will denote as (w_i, f_i) .
- *Normalized word frequencies*. Given the word frequencies, normalize them. To do this, simply take each frequency and divide it by the square root of the sum of the squares of all the frequencies:

$$(w_i, \frac{f_i}{\sqrt{\sum f_i^2}})$$

You can consider the normalized word frequencies as the specification of a point, where each word is a dimension, and the normalized frequency is the position of the text on that dimension. So, we compute the distance between two texts as a Euclidean distance:

$$distance(text_{1,} text_2) = \sqrt{\sum (fnorm_{1,i} - fnorm_{2,i})^2}$$

Note: If a text has a word that never appears in the other text, you can consider that the position along that word-dimension is 0 in the other text.

Your job is to write a single script, **`distance.py`**, which takes two command-line parameters, the text files containing the two texts we wish to compare, and outputs the distance between them. For example:

```
distance.py diary.txt newspaper.txt
```

We suggest that you write this exercise as follows:

- For each text file, read in its contents, and make a dictionary containing (word,frequency) pairs. (*Note*: You can read each text file in line by line, but there is a much easier way of reading in files which will save you a lot of time.)
- Modify these dictionaries to convert frequencies to normalized frequencies.
- Compute the distance between the two texts, using the formula above.

Example

Suppose we have the following two files:

text1

```
His cat is fat, and
that is his fat fate.
```

text2

```
The cat is fat, and that
is his feline fate.
```

The resulting (word,frequency) pairs for each text would be:

text1 freqs

```
fat          2
his          2
is           2
and          1
cat          1
fate         1
that         1
```

text2 freqs

```
is           2
and          1
cat          1
fat          1
fate         1
feline       1
his          1
that         1
the          1
```

The (word,normalized frequency) pairs are then:

text1 norm freqs

```
fate            0.25000
cat             0.25000
and             0.25000
fat             0.50000
that            0.25000
is              0.50000
his             0.50000
```

text2 norm freqs

```
feline          0.28868
fate            0.28868
the             0.28868
cat             0.28868
and             0.28868
fat             0.28868
that            0.28868
his             0.28868
is              0.57735
```

Applying the formula above to these two sets of (word,normalized frequency) pairs, the distance between the two texts is 0.5176.

You are provided with several text files you can use to test your solution:

- Distance between shrew.txt and muchado.txt: 0.358752
- Distance between shrew.txt and baskervilles.txt: 0.607597
- Distance between shrew.txt and oz.txt: 0.696045
- Distance between shrew.txt and rfc2821.txt: 0.884951

Documentation

- Make sure your code is adequately commented.
- Before any awk action, include a brief explanation of what that action is supposed to accomplish.
- If you have any functions (in awk or Python), explain their purpose, parameters, and return values.
- Any non-obvious code inside actions and functions should also be commented. The version of awk installed in the CS Linux computers will allow you to do this, but other versions of awk might not; if so, just write all your comments outside the actions.