CSci551/651 SP2018 Friday Section Homework 4

Assigned: Wed., 2018-03-29.

Due: **noon, Wed., 2018-04-11**.

There are $21 + 21 = 42$ points on this homework.

You are welcome to discuss your homework with other students, but each student is expected to write his or her final answer independently and in his or her own words. Students are expected to consult whatever resources are necessary to answer homework questions, including things like class papers, textbooks, other papers or RFCs, and the web. However, as above, all answers are expected to be formulated independently in your own words. If you find you must use materials from other sources, standard academic policies apply—you need to cite what material you use and clearly indicate (such as by quotes) what material was yours and what is theirs.

To submit this homework, please upload your one (ASCII) text or PDF file to the "homework 4" location on the class moodle. Simple text should be sufficient and is recommended as it is generally easier to mark up and give feedback about.

All students must include their *name* and *student id* in the body of the reply (so the file is self-contained), and please make sure you label the parts of your answer 1, 2, 3a, 3b, etc. In general, each part of a question should be answered in a few words or a few sentences. Answers that don't clearly identify what part of the question to which they apply, or that are overly long, may not get full credit.

Corrections: 2018-03-08: none yet.

1: (21pts) In this question you will take a quick sample of your personal computer files to see if they follow a heavy-tailed distribution, somewhat similar to Crovella and Bestavros' work. (We haven't read that paper as of the time this homework was given, although it's assigned before this homework is due. You're encouraged to read that paper before doing this question, but most of this question can be done before reading it. Parts (n) and (o) may require reviewing that paper.)

First, pick the computer or system with most of your files on it. Go to your home directory on Mac or Linux, or My Documents *and* Program Files on Windows, your "My Drive" on google drive, or your home directory on dropbox, and find the sizes of *all* the files under it. In Unix (or Linux or MacOS), you can do this in one line at the command prompt by running find piped into xargs with ls piped into awk or cut. (You may want to review the man pages for those commands, and remember to use the -print0 option to find and the -0 option to xargs if you have files or directories with spaces in them.) On other systems you may have to write a short program to walk the directory tree. This can be tricky to get right, although you're welcome to use libraries such as FTW in Perl, walk in Python, or ftw in C.

Now tell us a bit about your system. **(a)** (1pt) What operating system are you looking at, and how long have you used these files (approximately)?

What are the first order statistics: **(b)** (1pt) How many files did you find? **(c)** (1pt) What is the mean file size? **(d)** (1pt) What is the standard deviation of file sizes?

(e) (1pt) What is the median file size? (Recall that median is the file size "in the middle" of all files, i.e., 50% of files are larger and 50% are smaller.)

If you compare the mean and the median you can get a sense of if the distribution is skewed or not. For a normal (Gaussian, or bell-curve) distribution, mean and median should be about the same.

(f) (1pt) Based on mean and median, is the distribution of your files Gaussian, or are there more small files, or more large files?

(g) (1pt) What is the mode of all file sizes, and how often does it occur? (Recall that mode is the most common value.) **(h)** (2pts) Is this file size significant (a large percentage of files)? **(i)** (1pt) If so, what do you think caused it? (I.e., is there some application that creates files of exactly this size?) Or if not, why is it not significant?

(j) (2pts) What is the PDF of file sizes? (Recall that PDF is the probability density function, show the histogram with each bar being a given file size.) You may choose to plot the x and y axes as linear or log scale at your discretion.

(k) (2pts) What is the CDF of file sizes? Show the graph. (Recall that a CDF is the cumulative distribution function. So the CDF value at x is the probability that a measurement is less-than-or-equal-to x. Equivalently, the CDF value at X is $\int_0^X pdf(x)dx$.) Please plot the x-axis (file size) log scale and the y axis (CDF) linear scale.

(l) (2pts) What is the CCDF, the cumulative complementary CDF? Show the graph, plotting both the x and y axes as log-scale. Recall that the CCDF at X is simply $1 - CDF(X)$.

(m) (1pt) What characteristic of a CCDF indicates a heavy-tailed distribution? (In general and for any dataset, not specifically about *your* dataset.)

(n) (1pt) Do the file sizes of your home direction seem to show a heavy-tailed distribution? **(o)** (2pts) Why or why not? (Be specific about what in the graph supports or refutes your claim in part (n).)

(p) (1pt) Are you a bigger pack-rat than the professor, who found that he has 2,319,254 files in his home directory? (Partly because he uses the MH mail reader with each e-mail message in a separate file.)

Hints about statistics software: Be warned that you probably have more data items than will fit in Excel, but you're welcome to use any of the several public domain statistics packages such as R (`www.r-project.org`) or fsdb (`http://www.isi.edu/~johnh/SOFTWARE/FSDB/`), or a commercial package like S or Matlab or Mathematica, if you have access to it.

To use fsdb (these commands are for fsdb-2.x), print out the size of each file on a blank line, then turn that into a fsdb file with `dbcoldefine size >sizes.fsdb`. You can get mean and median with `dbcolstats -q 4 size <sizes.fsdb`. You can count how popular each size is with `cat sizes.fsdb|dbsort -n size|dbrowuniq -c >counted_sizes.fsdb`. From the popularity count you can find the mode (actually, the 10 most popular, not just the most popular) via count `cat counted_sizes.fsdb | dbsort -rn count | head -11`. You can compute the CDF from popularity with `cat counted_sizes.fsdb | dbsort -n size | dbrowaccumulate -c count | dbcol size accum >cumulative_sizes.fsdb`. You can compute the CCDF from, assuming you have 1551735 files, as `cat cumulative_sizes.fsdb | dbcolrename accum compaccum | dbroweval '_compaccum = 1551735-_compaccum;' >complementary_cumulative_sizes.fsdb`. You can plot the various values by reading these files into gnuplot.

To use R, load your data with `v <- scan("sizes")`, get basic statistics with `summary(v)` and `sd(v)`, find the mode with `vt <- table(v); which(vt == max(vt))`. You can plot the PDF with `plot(vt,log="x")`. Computing the CDF is trickier, but this does it: `cdf_v <- vt; for (i in 2:length(cdf_v)) { cdf_v[i] = cdf_v[i-1] + cdf_v[i]; };` . CCDF is a bit cleaner: `m=max(cdf_v); ccdf_v <- cdf_v; for (i in 1:length(ccdf_v)) { ccdf_v[i] = m - ccdf_v[i]; }`. But getting a plot out requires some hackery: `ccdf_v_matrix = cbind(names(ccdf_v),ccdf_v); plot(ccdf_v_matrix,log="xy");`.

2: (21pts) Something you hopefully studied in your undergraduate networking class was marshaling—how general data structures are sent between computers that may be of different types. (If you didn't This question explores how protocol headers are processed in real operating systems.

First, find the Internet Request for Comments (RFC) that specifies the ICMP protocol. a) (1pt) What RFC number is the official reference for this protocol? b) (2pts) What is the official specification of the message format for an ICMP "echo reply" message? (Cut and paste from the RFC.)

Find the ICMP header representation for some common operating system in the system's C header files. c) (1pt) What operating system and version did you pick? d) (2pts) What is the relevant structure that represents the ICMP header? (copy and paste it into your answer)

A challenge in implementing protocols is doing so *portably*. The C language leaves many details up to the compiler, including structure packing and size of integers. (An int can be 32 or 64 bits long [or evey 16 bits] depending on the computer and operating system.)

e) (2pts) Does that structure have code to handle compiling on different architectures (yes or no)?

If it does, f) (3pts) How does it handle different sizes of "int"?

If it does not, then f) (3pts) How would you modify it to handle sizes of "int"?

Pick some *other* operating system than the one you chose in step (a). (Other means a completely different OS, not just a different version of the same OS.) g) (1pt) What other OS and version did you pick? h) (2pt) What is the relevant structure that represents the ICMP header? (copy and paste it into your answer) i) (2pts) How does the approach to handling portability taken by this other OS compare to what you found in the first OS?

Although most operating systems deal with marshaling by a combination of carefully crafted C structures and macros like ntohs, one can do all the packing and unpacking by hand without too much trouble. j) (5pts) What goes in the following C-language function that converts a packet header passed in as an array of unsigned chars into the following output structure, using only bit shifts and/or multiplications and additions (*not* carefully formatted structures)? (You take the input data as per the RFC in network byte order and should unpack it into the given structure into the native unisgned integer format of your local machine.) Some additional requirements: (1) Your code must be *portable* to any computer where integers are at least 16 bits, either big or little endian! (2) Please note that the required structure does *not define* the encoded format (some fields in the C structure

are different sizes [larger] than their encoded format). That's defined in the RFC that you look up as question part (a). (3) If your code is longer than about ten lines, you've probably done something wrong.

Required structure:

```
struct decoded_icmp {
        unsigned int type,
                code,
                checksum,
                id,
                seqno;
};
void
demarshall(unsigned char bytes[], struct decoded_icmp *out)
{
   /* your code goes here */
}
```

As sample input, if you run it on this array of bytes, a correct solution should find an increasing pattern in the output of the fields when printed in hexadecimal:

```
unsigned char sample_bytes[] = { 1, 2, 3, 69, 6, 120, 9, 171 };
```