P4 Submission

Nick Werle

March 12, 2017

1 Task 1

My program works well, with no bugs or crashes that I am aware of. To test, I made the file many times larger (copy and pasted the original 240k log until it was 300MB) and it still ran well. Additionally, using a script called memusg from `https://gist.github.com/netj/526585`[1], I was able to verify that the memory consumption of my implementation did not noticeably grow with the size of the input.

I developed this program by creating a package for parsing, which will hopefully be trivial enough to modify for the later parts of the assignment. Using streams and regex it searches each line of the log against the regex string and extracts the IP address if possible. I think the callback that I put in place will be valuable for the later portions, and what code there is will be simple to modify.

2 Task 2

3 Task 3

The source for this program is available in the Task 3 folder. I used the following input to test my program:

```
https://www.gutenberg.org/files/38280/38280-0.txt
http://www.gutenberg.org/cache/epub/20/pg20.txt
http://www.gutenberg.org/cache/epub/8209/pg8209.txt
https://www.gutenberg.org/files/25340/25340-0.txt
https://www.gutenberg.org/files/25340/25340-0.txt
http://www.gutenberg.org/cache/epub/8388/pg8388.txt
```

[1] This script is included in my repository as a submodule.

```
http://www.gutenberg.org/cache/epub/22312/pg22312.txt
http://www.gutenberg.org/cache/epub/2939/pg2939.txt
http://www.gutenberg.org/cache/epub/36446/pg36446.txt
http://www.gutenberg.org/cache/epub/5891/pg5891.txt
```

Which generates the following output (n = 10):

```
the - 57874
of - 31890
and - 29439
to - 19994
a - 16415
in - 16365
is - 9199
i - 8946
that - 8150
```

My solution works well, as best I can tell. It does not attempt to strip out html or anything similar, and so it generally assumes that input is plaintext. Additionally it will only be compatible with the standard 26/52 character English alphabet. As I understand it this solution works with ASCII as well as other UTF encodings, though I have not tested this.

The experience of developing this was overall not very different from Task 1, though the optimization phase did strain my familiarity with the Java regex/Pattern API. Once again I took the approach of making a general enough solution that I will be able to re-purpose my package for the android application, however this time I took a more straightforward approach by returning the list and allowing the package user to manipulate it as they please.

One thing I found during development is that it is considerably faster to use regex to match the words themselves (my current solution), rather than their delimiters (my initial solution). While this arguable produces less true results (e.g. hyphenated words will not be treated thusly), it simplifies the sanitation process considerably (lowercasing, removing trailing punctuation, etc.) and avoids multiple, costly passes over the dataset as different map and filter operations are applied. You could, of course, apply one very good map or filter that does the sanitation in a single step, but by grabbing the words directly, you can avoid even that single extra pass, speeding up the output considerably.