CSE 216 Homework IV

This is the fourth and final assignment of the semester, and as we have done in past semesters, this is a significantly shorter homework. The goal of this assignment is to learn how to

(a) use a language designed for quick prototyping
(b) to sift through large amounts of information
(c) using functional programming techniques to filter out irrelevant information, and
(d) exploit the advantages of lazy evaluation to arrive at the desired results faster (i.e., with fewer computations).

A. Language for quick prototyping

For faster prototype development, we will use the dynamically typed language we have studied in this course: Python. This faster speed of development is a big reason (although not the only one) behind the popularity of Python, and I hope this assignment will be able to demonstrate that to some extent.

B. Large amount of data

The data we will be using for this assignment is a single large file of server log information. Every single line in this file is one act of a client communicating with the server, and it provides the following pieces of information:

date and time, duration, protocol, source IP address, source point, destination IP address, destination point, number of packets transferred, bytes transferred, flows, flags, Tos, class, attack type, attack ID, attack description

We don't need to understand what every piece of information means (in fact, getting to our final answer without worrying about other details is the very point of this exercise). But from the above field names, we can still get an overall picture of what kind of information is getting stored in this server. You will notice that some fields have very few specific values. For example, the protocol is almost always TCP (which stands for *transmission control protocol*) or UDP (which stands for *user datagram protocol*). On the other hand, the "class" is always either marked *normal* or *suspicious*.

C. The task

Our task is to sift through this data and identify those communication instances that are "suspicious" and use the user datagram protocol. Further, we are interested in situations where there are multiple very quick and very short bursts of UDP communications to the server. This vague description needs to be refined, of course! So … what exactly do we mean by "multiple", "very quick", and "very short"?

- By multiple, we simply mean "more than one".
- We will quantify "very short" to mean that the duration of each communication is less than a millisecond.
- And finally, we will quantify "very quick" to mean that the communication is happening within a second of the previous communication timestamp.

These are hallmarks of attacks on a server, which is why we want to identify them as quickly as possible[1]. In particular, we want to issue a notification as soon as we spot the very first instance of such potential attack on the server. Therefore, our task is as follows:

Sift through the given data to find the very first instance of a communication that satisfies ALL the following properties:

1. It uses the UDP protocol.
2. It is marked "suspicious".
3. It has a duration less than a millisecond
4. The previous communication using UDP (and marked "suspicious") happened within one second before this one.

You must write a class called ServerAttackDetector. The constructor for this class should take the path to the data file as its argument. For example:

```
d = ServerAttackDetector("/User/myname/location/of/downloaded/datafile.csv")
```

This class be defined in a module named **`serverattackdetector`**, and must have a method called **`detect`**, which must return a tuple of the form **`(a, b)`**, where

- **`a`** is an int value showing how many lines of the data file it actually had to read in order to detect the first potential attack, and
- **`b`** is the entire line from the data file, which describes this potential attack.

Your code must take advantage of functional programming techniques and lazy evaluation in order to avoid needless computation[2]. Specifically, we (i.e., the graders) will expect the following to work:

```
from serverattackdetector import ServerAttackDetector

d = ServerAttackDetector("...")  # to be replaced by actual path
a, b = d.detect()
print(a)  # to verify whether your code is, indeed, performing lazy evaluation
print(b)  # to verify whether your code is, indeed, able to detect the first
          # instance of a potential attack
```

There is a small sample test file we have provided to you already, called **`hw4testfile.csv`**. In this file, line no. 6 is the very first line that satisfies all the four criteria, so the correct output should be

```
print(a)
>>> 5
print(b)
>>> 2017-03-21 21:21:43.508,   0.000,UDP  ,14278_26,57110,EXT_SERVER,1434,
1,    46,   1,......,  0,suspicious,---,---,---
```

Note that when printing 'a', you are including the actual line being returned (i.e., the one bound to the variable 'b') but you are NOT including reading the header line that has the column titles.

[1] We have simplified the criteria a lot. Detecting actual server attacks requires techniques that are a lot more sophisticated, and often include machine learning coupled with data science analyses that happen in conjunction with the kind of programming this homework is introducing.
[2] Recall from our lectures, how the use of higher order functions with streams in Java needed fewer computations when there was a terminal operation.

Rubric

The rubric for this assignment is very straightforward. Your code will be tested with ten cases (some will be large amounts of data, like the complete file given to you while some will be small files, like the test file given to you). Each test case will have the exact same format as the files already provided to you (i.e., same column names and column organization; the actual values in the rows will differ from one test case to another). Each test case is worth 10 points: 5 points for iterating through the correct number of lines as per what we expect from a lazy evaluation, and 5 points for returning the corresponding correct line from the server logs.

- Late submissions or uncompilable code will not be accepted.
- Please remember to verify what you are submitting. Make sure you are, indeed, submitting what you think you are submitting!
- **What to submit?** A single .zip file comprising the completed codebase. *You do not have to submit the script shown above in the box. That piece of code is given here just so that you can test the output of your code. Please do NOT submit any data file either.*