Quiz 3: Hadoop for Fun and Profit

(Alice with her elephant!)

This quiz has 4 problems in it, worth a total of 12 points, assigned on Sep 29 2022 12:00 AM and is due on Oct 12 2022 11:59 PM. The scores will be awarded out of a max of 240 and eventually divided by 20.

1. TrafficRank [6 points]

The random surfer model of calculating the PageRank of a web page works by assuming a web surfer follows the outgoing links of the page with equal probability, but will restart every time with probability 1−β, and start at a new page, chosen uniformly at random. The probability β is called the decaying factor and influences the convergence of the computation of PageRank; typical values of β are about 0.85.

The idea of PageRank is not limited to web pages. In this problem, we model the economic rank of areas within a city under the hypothesis that inbound taxi traffic into an area might be an indicator of its economic rank. Thus we invent the notion of TrafficRank.

The main differences between *PageRank* and *TrafficRank* are that (1) for PageRank the probabilities of outgoing links are equal but for TrafficRank they are weighted by actual usage, and (2) links within an area are also counted. The resulting transition matrix is typically a full matrix, not sparse as in the case of PageRank.

A matrix of taxi trips between Chicago community areas is available publicly, downloadable from my Google Storage: `https://storage.googleapis.com/singhj-cs-119/chicago-taxi-info/chicago-taxi-rides.csv`

The city is divided into 77 "community areas," thus giving us a 77 × 77 matrix of T_{ij} entries where T_{ij} is the number of trips from community area i to community area j.

1. [1 point] The given data has rows (i, j, T_{ij}), sometimes known as COOrdinate format. It is supported by many libraries, `SciPy.sparse.coo_matrix` among them. Since the matrix is just 77 × 77, a "big data" technique is not required here. You may use any library, e.g., *scipy* or *numpy*, for reading the matrix.
2. [2½ points] Using your formulation of the TrafficRank algorithm, calculate the rankings of the Chicago community areas after 2, 4, 8, … 64 iterations of the algorithm.
3. [2½ points] An alternate measure of economic rank of an area might be the inverse of a "hardship index," defined and quantified here. How well, or poorly, do your calculations of TrafficRank for Chicago community areas correspond with the inverse of hardship index? We're looking for *qualitative analysis in plain English*.

2. Hadoop Errors [1 point]

When dealing with errors in Hadoop, where the execution is distributed to hundreds of workers, an error message could end up in a log file on any of those servers. This is a scavenger hunt question. We deliberately modify the code so it would occasionally fail and look for the error message so we can find them! In the footnote is a modified mapper for Hadoop Streaming, with the changed lines shown in red[1].

[1] The modified mapper code is:

```
#!/usr/bin/python
import sys, re
import random

def main(argv):
 line = sys.stdin.readline()
 pattern = re.compile("[a-zA-Z][a-zA-Z0-9]*")
 try:
   while line:
     for word in pattern.findall(line):
       print "LongValueSum:" + word.lower() + "\t" + "1"
       x = 1 / random.randint(0,99)
     line = sys.stdin.readline()
 except "end of file":
   return None

if __name__ == "__main__":
 main(sys.argv)
```

Run Hadoop Streaming on the five books we have been using for practice, using this modified mapper. It will fail, of course!

[½ point] Where (what server & location) did the divide-by-zero error messages show up and how many did you find?

[½ point] How many such messages did you find? Is the count you found consistent with what you might expect from random.randint(0,99)?

3. Functional Programming [1 point]

Write a functional function fcat(a, b, …) that returns a concatenation of all its inputs with dashes in between, for example: fcat(“red”, “fish”, “blue”, “fish”) returns “red-fish-blue-fish”. Your code should use recursion, not iteration.

4. Presidential Speeches [4 points]

All Presidential Speeches through history are available here (formatted as a .zip file containing a .tar.gz file for each president) for analysis[2].

Download the speeches and analyze them for the aggregate word polarity of the speeches according to the AFINN-165 valence system, which classifies about 3,382 words on a scale from -5 to +5. The exercise is to calculate the *aggregate valence per 1,000 words* for each president.

- There are at least a couple of ways of solving this problem. We would like you to use **map-side evaluation.** In the mapper, for each president, look up the valence of words in a speech in the mapper, emit sets of (president, aggregate valence) for each President. In the reducer, combine all aggregate valences for each president. [2 points]
    a. Name two Presidents whose speeches had the highest valence per 1,000 words.
    b. Name the President whose speeches had the lowest valence per 1,000 words.
- Assuming that each President name is 20 characters, each integer is 8 bytes and each word is, on average, 8 characters, Calculate the volume of data that would flow through the shuffle network[‡ര]. [2 points]

‡ You may make any other "reasonable" assumptions that you deem necessary.

ര We’re looking for 1-digit precision – an order-of-magnitude is sufficient – for example, an answer like 3.14 KB is faux-precision – 3 KB is adequate.

---

[2] The corpus was downloaded by me a few years ago from a website that no longer exists. The last speeches in the corpus were Obama’s. I have not found a more up-to-date version of the corpus.