CS205, Fall 2013
Computing Foundations for Computational Science
Cris Cecka

Homework 1
Due September 27th, 2013 at 11:59pm ET

This homework will be an exploration of the uses of MapReduce. In this assignment, you will run jobs locally using your machine or the CS205 cluster as well as using Amazons Elastic MapReduce (EMR).

Note: In order to run your code on Amazon, you need to finish problem 1 and provide us with a bit of time to send you your Amazon service credits. As a result, please complete problem 1 *ASAP* (i.e., today!) to minimize the potential for complications.

Download the material for HW1 here.

or

`http://iacs-courses.seas.harvard.edu/courses/cs205/resources/hw1.zip`

Problem 1 - Last bit of configuration!

Fixing SEAS Files

The first configuration script had a bug in it. We've also completed a new cluster just for CS205 which we can use in this homework and future homeworks.

Downloading and running another config script should fix everything:

```
Please back up your current SEAS folder before updating
$ wget crisco.seas.harvard.edu/CS205/CS205config2
$ bash CS205config2
```

This creates shortcuts to the `cs205.seas.harvard.edu` cluster just like the other shortcuts to the `resonance.seas.harvard.edu` cluster. The CS205 HEADNODE terminal is a shortcut for `ssh -X SEASUSER@cs205.seas.harvard.edu` and the CS205 NODE terminal is a shortcut for `ssh -X SEASUSER@cs205.seas.harvard.edu` and then running `qlogin`, which we will explain more about in class.

This also fixes the `SEAS` file sharing. Just in case, it backs up everything in your current `SEAS` folder to `SEAS_bkp`. Your `SEAS` file should now have everything in your Harvard files. **Everything in your SEAS file should also be available on each cluster**. That is, opening any of the terminal shortcuts on the desktop and listing the files with `ls` should show the same files.

Using MRJob and other course software on the clusters

To load the software we'll be using in CS205 on any of the clusters (`resonance.seas` and `cs205.seas`) execute

```
$ module load courses/cs205/2013
```

To automatically load the software when you log into a cluster edit `~/.bashrc` on the cluster and add the above command.

Amazon Elastic Compute Cloud (EC2)

As a large Internet retailer, Amazon requires a powerful computing infrastructure to support its day to day operations. While others in a variety of fields could also benefit from having access to such computing resources, the costs associated with installing and maintaining a similar infrastructure often makes this infeasible, especially when the cluster would only be used sporadically. Amazon is targeting this market with their Elastic Compute Cloud (EC2) product, which we will use in this course. A NYT article with more background about their business can be found here.

Apply for an Amazon Web Services (AWS) Account

For the class, Amazon will be providing each of you with $100 of free AWS credits. First, you must register for an AWS account, during which you will be required to enter your own personal credit card information. Once registered, we will provide you with a $100 credit code. **It is important you understand that once the provided credit code is used up, your credit card will be charged for any additional AWS usage, so it is important to keep track of your usage.**

The following steps will guide you through the registration process:

1. Sign up for AWS using either your personal Amazon account or by creating a new AWS account.

2. After signing up for AWS, sign up for EC2, which will include registration for Elastic MapReduce and a other similar services. *Some of these other services may carry a cost if you decide to use them for your own personal use.*

3. Wait for an email from us with your AWS credit code.

4. Login to your AWS Account page. Click Payment Method. At the bottom of the page, click Redeem/View AWS Credits. Then, enter your code and click redeem.

5. As mentioned in class, you may want to set up a billing alert using this link.

You can manage your account via the AWS Console.

An Introduction to MRJob for EMR

Configuring MRJob for Amazons Elastic MapReduce can be done in one of three different ways: a configuration file, the command line, and with code. For security and privacy, we recommend either one of the first two methods.

If you wish to use a configuration file, modify the `mrjob_configuration_file.txt` in the homework handout. Open it and change the access key and secret access key to your credentials. These can be found here under Security Credentials. After logging in, select "Security Credentials". Under "Access Credentials" you will see a tab titled "Access Keys". Your "Access Key ID" and "Secret Access Key" are here.

After you entered your login information, you have two possibilities to let MRJob know about your config file. One option is to rename your config file to `.mrjob.conf` and copying it to your home directory `~/.mrjob.conf`. Note that the file name starts with a dot. This means that the file is hidden and that you need the `-a` option to see it when using the `ls` command to look at the directory entry.

Note: As you should be running the MRJob tasks either locally, on a CS205 compute node, or on Amazon, the ~ in the configuration path corresponds to your *local* home directory and/or your SEAS directory.

The other option is to specify the path to the configuration file in the environment variable `MRJOB_CONF` by typing:

- `export MRJOB_CONF=/home/you/yourpath/fileName.txt`.

Note: Just a reminder, with these keys ANYONE can send a job to Amazon under your guise (and you will be charged). It should be fairly obvious that you therefore do not want to distribute these keys. This was why we recommended you avoid using the third method of MRJob configuration as it would require hard-coding these keys into your code. Nonetheless, if at anytime your keys are compromised, you can log into the same area, create a new pair, and deactivate the current pair.

If you decide to use AWS for your final project, a configuration file is preferable to avoid the repetition of reconfiguration. However, you can also use the command line to configure MRJob. First in your virtual box environment, you must open a terminal and set two environment variables so MRJob will have your AWS account information. Type following two commands in your terminal:

- `export AWS_ACCESS_KEY_ID=xxxxxx`
- `export AWS_SECRET_ACCESS_KEY=yyyyyy`

where the `xxxxxx` and `yyyyyy` are your Access Key ID and Secret Access Key, respectively.

To run a MRJob python script, use this command:

```
$ python myscript.py < inputfile > outputfile
```

This will run MRJob locally which is useful for testing. For this homework, locally can be your machine (with LOCAL MACHINE terminal) or a CS205 compute node (using CS205 NODE terminal).

After you set up the access keys, you can run EMR jobs using MRJob using this command:

```
$ python myscript.py -r emr < inputfile > outputfile
```

(note the `-r emr` portion). When using this command, MRJob will automatically upload any input files to the Amazon cluster and download the results to the specified output file on your local machine. Status updates will be repeatedly sent to the terminal as your job waits for the instance to start and as the job runs. You can find more detailed information on your job by looking at the AWS Console.

By default, a single "small standard on-demand" instance will be used for computation. However, these settings can be modified via any of the previously mentioned configuration methods using the "ec2_instance_type" and "num_ec2_instances" flags. See here for more details on these flags as well as others. You can find out more about MRJob here.

Note: You may see an error message stating *No handlers could be found for logger* `mrjob.conf`. This line alone does not indicate a bug in your code (its just indicating we have not set up the logger). Your code should still run fine regardless of its presence.

Important: Please make always sure that your code is bug free, before actually submitting it to amazon. Try to run the job locally first and see if it produces the desired result. Then, if this worked, you are ready to proceed to the cloud. The homework problems are small and your free credit should provide you with a lot of room for running and testing on Amazon. However, it is your responsibility to make sure the jobs terminate properly and do not cause excessive costs. You can always monitor your currently running jobs using this overview of your MapReduce job flows.

Problem 2 - MapReduce for Trapezoidal Integration [25%]

One technique for numerical integration, which you may have learned during a calculus course, is trapezoidal approximation. When using this method, the range of integration is divided into units of width h and trapezoids approximate the area under the curve for each unit.

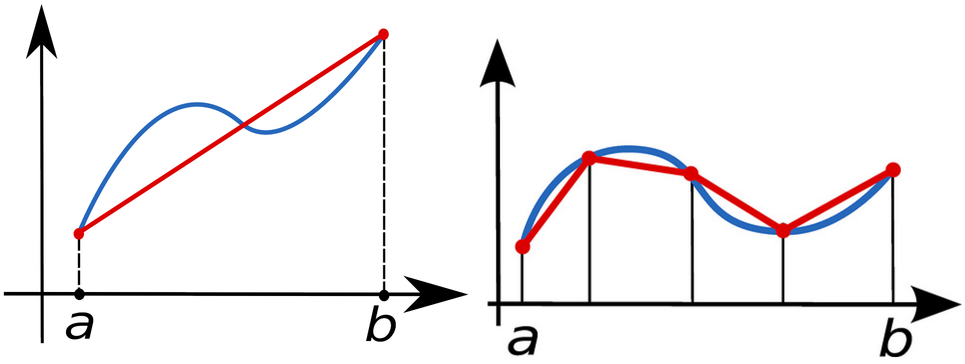


Figure 1: The function $f(x)$ (blue) is approximated by a stepwise linear function (red).

For the left example in Figure 1 the integral with just one unit from a to b over $f(x)$ is approximated by:

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)\frac{f(a)+f(b)}{2}$$

For the example on the right we have to sum up the different units:

$$\int_a^b f(x)\mathrm{d}x \approx h\frac{f(a)+f(a+h)}{2} + h\frac{f(a+h)+f(a+2h)}{2} \dots$$

With a bit of math this can be written as:

$$\int_a^b f(x)\mathrm{d}x \approx h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{N-1} f(a+ih)\right)$$

Using MapReduce and 100,001 intervals (note this is 100,000 "points" plus the two endpoints), compute $\pi/4$ with and without in-mapper combining. Compare the results of each. Are the answers you receive the same or do they vary slightly? Explain.

Just as a reminder, the area of a unit circle equals π and a unit circle in the first quadrant can be described by the following equation:

$$f(x) = \sqrt{1-x^2}$$

Note: We have intentionally not provided you with an input file for this problem. You are expected to generate your own using any method of your choice, however, the input file should be made up of integers only.

Submission Requirements

- `P2a.py`: Completed script with standard map-reduce (mapper and reducer only, no combiner).
- `P2b.py`: Completed script with in-mapper combining.
- `P2.txt`: The input file you generated for the `P2a.py` and `P2b.py` scripts.
- `P2.pdf`: Computed values from `P2a.py` and `P2b.py` and an explanation detailing any variation between the two computed values.

Problem 3 - Anagram Solver [25%]

Jumble is a common newspaper puzzle in the United States. As seen below, it consists of a first series of anagrams that must be solved. Circled letters are then used to create an additional anagram, whose solution answers a question posed by a small cartoon. In especially difficult versions, some of the anagrams in the first set can possess multiple solutions. The correct solution can only be determined by trying to solve the subsequent cartoon (and likely failing). Therefore, when solving this puzzle, it can be quite important to know all possible anagrams of a given series of letters. In this problem, were going to compute all possible anagrams for any valid sequence of letters (i.e., one that forms a valid word).

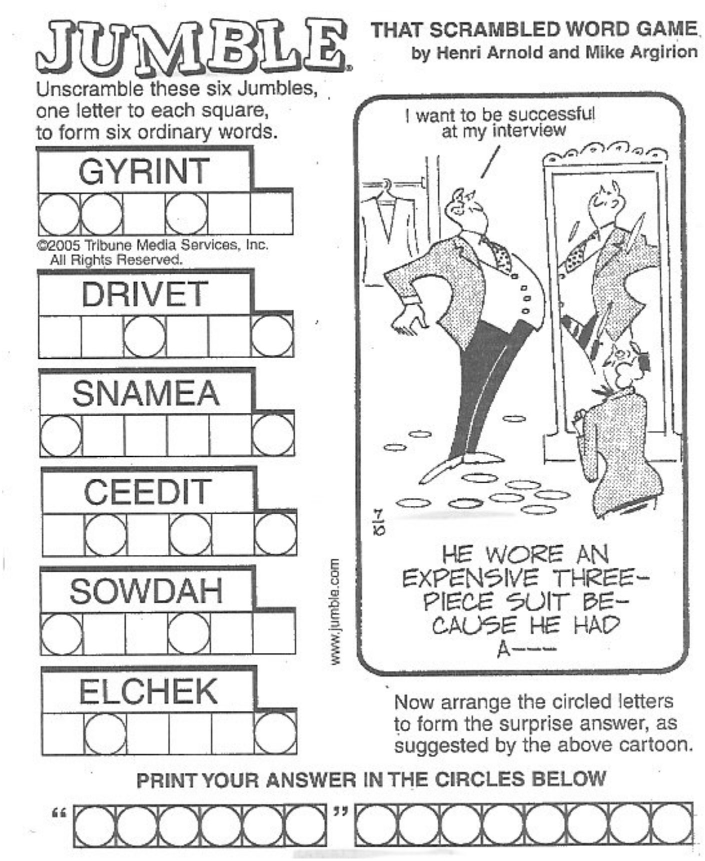

JUMBLE
THAT SCRAMBLED WORD GAME
by Henri Arnold and Mike Argirion
Unscramble these six Jumbles, one letter to each square, to form six ordinary words.
GYRINT
©2005 Tribune Media Services, Inc. All Rights Reserved.
DRIVET
SNAMEA
CEEDIT
SOWDAH
ELCHEK
I want to be successful at my interview
www.jumble.com
HE WORE AN EXPENSIVE THREE-PIECE SUIT BE-CAUSE HE HAD A---
Now arrange the circled letters to form the surprise answer, as suggested by the above cartoon.
PRINT YOUR ANSWER IN THE CIRCLES BELOW
“ ”

Figure 2: A list of all anagrams could help solve this example Jumble Puzzle.

Using MRJob and the provided Scrabble word list, write a Python script that generates sets of valid anagrams for all words in the Scrabble word list. Generate an output file of the following form (additional brackets, quotation marks, etc. in the output file are not important – it just needs to follow this general format and be readable by a person):

```
SortedLetterSequence1 NumberOfValidAnagrams1 ListOfValidAnagrams1
SortedLetterSequence2 NumberOfValidAnagrams2 ListOfValidAnagrams2
...
```

where `SortedLetterSequence` is a sequence of letters in alphabetical order (and is distinct from any other `SortedLetterSequence`), `NumberOfValidAnagrams` is the number of words that can be constructed from the letters in `SortedLetterSequence`, and `ListOfValidAnagrams` is the list of anagrams of the that can be constructed with the letters in `SortedLetterSequence`.

You should use the default instance size/type configurations as this is a relatively small job that should only take a few minutes to complete.

Submission Requirements

- `P3.py`: Complete script solving the anagram exercise
- `P3.pdf`: A single entry from the output file, namely the entry with the most anagrams. **HINT:** For our dictionary, the entry with the most anagrams should have 12 anagrams.

Problem 4 - Graph Processing Under Map/Reduce [25%]

Graphs are ubiquitous in modern society. Examples encountered by almost everyone on a daily basis include the hyperlink structure of the web (simply known as the web graph), social networks (manifest in the flow of email, phone call patterns, connections on social networking sites, etc.), and transportation networks (roads, bus routes, etc.). Search algorithms on graphs are invoked millions of times a day, e.g., whenever anyone searches for directions on the web.

For this problem, we will utilize the characters found in the Marvel Comic universe and a graph formed from those characters who appear in the same comic issues.

Generating the Marvel Graph

Alberich, Miro-Julia, & Rossello (2002) examined the characteristics of the graph formed by the characters and comics in the Marvel Comic universe, and for this problem we will be using the data that inspired their work. Begin by downloading their data file to your local system by browsing to the website hosting the dataset.

Each line in this file contains a `"CHARACTER NAME", "COMIC ISSUE"` pair, where the existence of a pair implies that that character appeared in the associated comic. We can conceptualize these relationships as a graph, where each character is represented by a vertex and an edge exists between any two vertices if the corresponding characters appear in the same comic issue.

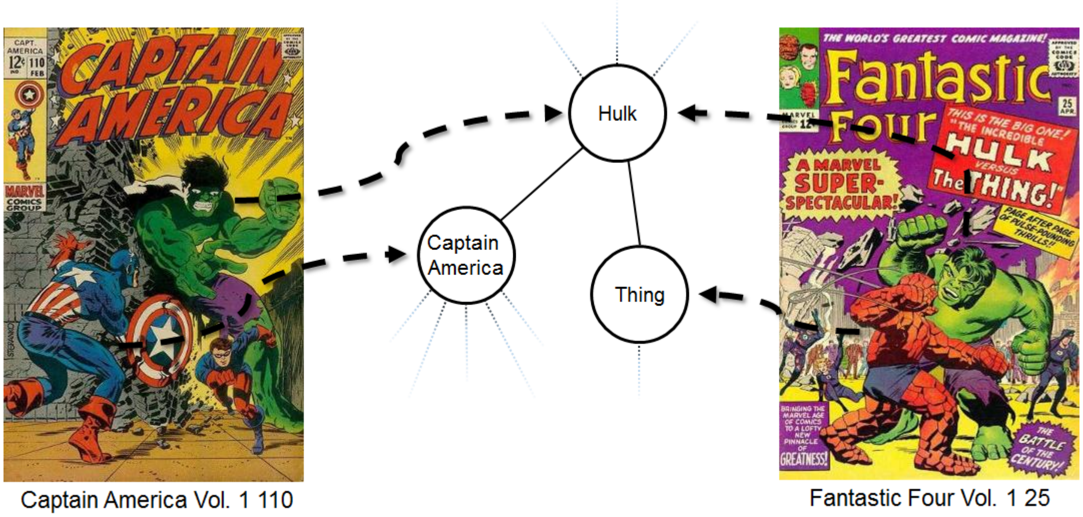
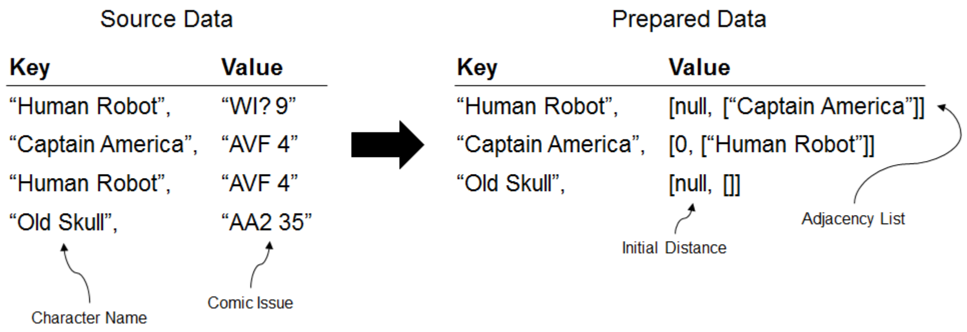


Figure 3: Interpreting character associations in a comic corpus as a graph

Your first task will be to convert this source data into a adjacency list format appropriate for further processing. After transformation, our data will consist of a set of pairs with keys that comprise all character names. Each associated value is itself a pair, with the first element indicating the currently found minimal distance of the respective node to the origin node (more on this below!), and the second component a list of characters that are associated with that vertex. The figure below illustrates this process on a small subset of the input data:



Source Data

Key	Value
“Human Robot”,	“WI? 9”
“Captain America”,	“AVF 4”
“Human Robot”,	“AVF 4”
“Old Skull”,	“AA2 35”

Prepared Data

Key	Value
“Human Robot”,	[null, [“Captain America”]]
“Captain America”,	[0, [“Human Robot”]]
“Old Skull”,	[null, []]

Figure 4: Preparing the raw Marvel data for graph processing.

Design and execute a map/reduce process which performs this transformation. For these types of map/reduce jobs, MRJob supports the ”chaining” of multiple steps together, where the output of one reduce step is fed as input to a subsequent mapper. Read the documentation on multi-step jobs for details on how this is accomplished. (Note that if the `map` argument to your `MRJob.mr` invocation is omitted, MRJob will conveniently assume an identity mapper.)

Single-Source Breadth-First Search

One of the most common and well-studied problems in graph theory is the single-source breadth-first search (SS-BFS) problem, where the task is to find the shortest paths from a source node to all other nodes in a graph (or alternatively, edges can be associated with costs or weights, in which case the task is to compute lowest-cost or lowest-weight paths).

Given the graph generated previously, you will now execute a single-source shortest-path algorithm starting from a several given root nodes to determine the shortest distance to all other (connected) nodes in the graph.

To perform this search, we will again use an iterative sequence of map/reduce jobs, where each step expands the search frontier by one “hop”. During each iteration, mappers should emit incremented distances to each neighbor, and the reducers select from the minimum of all of those distances (including the vertex’s current distance).

For the purposes of this problem, you may assume that the diameter of this graph is four (4). What does this diameter imply about the maximum number of steps in which your search must be executed? In practice, determining a graph diameter is difficult, both from a theoretical and technical (q.v. HADI diameter estimation) perspective.

Connected Components in the Marvel Universe

Execute your SS-BFS job three separate times using the prepared Marvel graph for the source vertices listed below. Before each execution, reset all of the distance values to `null`/`None` and initialize one of the following nodes to `distance=0`:

- `"CAPTAIN AMERICA"`
- `"MISS THING/MARY"`
- `"ORWELL"`

Count the number of vertices touched in each of the above searches and place these counts in your writeup. What does it mean if a vertex does not have its distance message modified during the search? Use this information to determine how many connected components are in the Marvel universe. (You may use additional one-off map/reduce jobs to count the reached vertices, or utilities such as `grep`.) Don't forget to consider isolated vertices!

Submission Requirements

- `P4-prepare.py`: Map/reduce job for converting the raw Marvel input into the adjacency form described above.
- `P4-ssbfs.py`: Map reduce code that implements SS-BFS on a given graph.
- `P4.pdf`: Vertex count for the three required SS-BFS executions, total number of connected components, and a brief description of how you determined these values.

Problem 5 - EMR Performance Scaling [25%]

MapReduce for Monte Carlo Integration

Monte Carlo methods, those which rely on the generation and intelligent use of random numbers, are often used in simulation to compute a result that would otherwise be impossible or infeasible using normal deterministic algorithms. One such application of these methods is numerical integration. When integrating over a high-dimensional space, computing definite integrals can be computationally burdensome. In these cases, it is common to see Monte Carlo integration used to compute the integral through repeated sampling of the space.

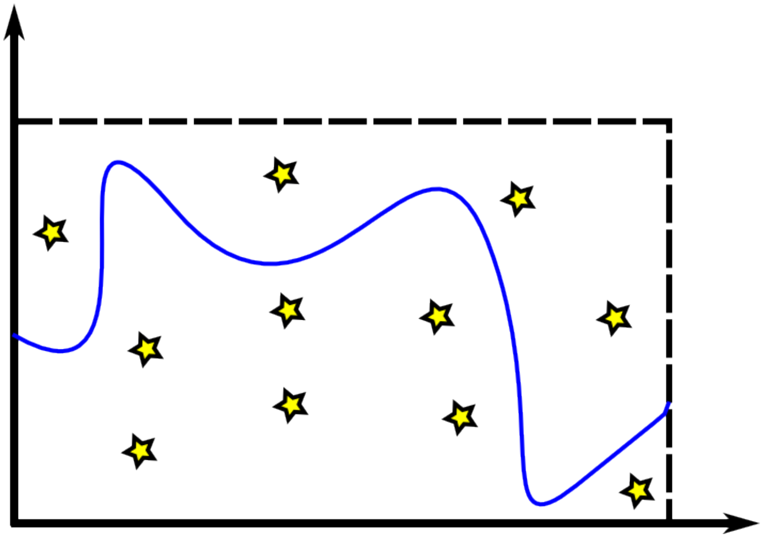


Figure 5: Monte Carlo integration works by comparing the fracting of particles under the curve to the total number of particles within an area of known size.

While many details can affect the efficacy of this method, the underlying concept is fairly simple. Consider the rectangular region shown above with known area A and the complicated curve shown in blue with definite integral I (whose value is unknown). If we were to plot uniformly-distributed random points in the rectangle, the following relation would hold.

$$\frac{p_{under}}{p_{total}} \approx \frac{I}{A}$$

Further, as the number of points plotted increases, our approximation will increase in accuracy as we sample more and more of the space, assuming we can generate sufficiently random numbers. *Note:* The increase in accuracy is independent of the dimension of the space. This is not true for many other numerical integration techniques, making Monte Carlo integration especially suitable for high-dimensional spaces.

With this relationship established, approximating the integral becomes easy. Simply multiply the sampled point ratio by the area of the enclosing region.

You should generate an input file containing a sequence of integers from 1 to 100000. For each integer listed in the file, seed a random number generator using that number as the seed. After seeding, generate 1000 uniform random pairs per seed. Use this data, along with a 1x1 square and the equation of a circle of radius 1, to approximate the value of $\pi/4$. Your script should output two labeled numbers, `numerator` and `denominator`, which when divided, will approximate the desired value.

When running your script, you should launch your job on EMR using the following configuration pairs:

- m1.small, 1 instance
- m1.small, 16 instances
- m1.large, 1 instance
- m1.large 16 instances

For each configuration, you should record the jobs run time (its listed in the text that is printed to the terminal, look carefully!). Plot these results on a labeled bar graph.

Note: If you want more frequent progress updates from EMR (5s instead of 30s), add the following line to your .mrjob config file: `check_emr_status_every: 5`

Note2: You should use in-mapper combining for this problem to reduce the bandwidth consumed when passing data to the reducer.

Submission Requirements

- `P5.py`: Completed python script
- `P5.pdf`: Bar graph of run times and computed fraction from any of the instances

Problem 6 - Intro to Culturomics [Extra credit]

One interesting feature of MRJob is the ability to run a MapReduce job from within a separate script and use the results to as input for other tasks such as plotting. See this link on runners for a brief introduction.

In the homework handout we have provided you with a few famous books throughout history from Project Gutenberg (A Tale of Two Cities, Pride and Prejudice, Wuthering Heights, Canterbury Tales, the King James Bible, Paradise Lost, and Frankenstein). Construct a line plot showing the relative frequency for each letter (i.e., there should be 26 x-coordinates plotted for the 26 English letters and a single line plotted for each book). Are there any noticeable trends? For this problem, include all code and an image of the final plot.

Submission Requirements

- `P6.py`: Completed python script containing the MRJob subclass
- `P6Driver.py`: Completed python script that runs the MapReduce job and parses the result
- `P6.pdf`: Image of the result plot and brief description of any significant similarities and differences between the curves and possible explanations for those differences.