CS324e - Elements of Graphics and Visualization

Java Intro / Review

A1 Demo

- Demo of A1 expected behavior
- Crack a substitution cipher
- assumes only letters encrypted and assumes upper and lower case substitutions the same
- initial key based on standard frequencies
- allow changes to be made

Java Intro / Review

- Instead of going over syntax of language we will write a program to solve a non trivial problem and discuss the syntax and semantics as we go

Zipf's Law

- Empirical observation - word frequency
- Named after George Zipf, a linguist
- Zipf's Law: The frequency of a word is inversely proportional to its rank among all words in the body of work

Zipf's Law Example

- Assume **the** is the most frequent word in a text and it occurs 10,000 times
- 2nd most frequent word expected to occur 5,000 times (if top ranked word's frequency is as expected)

  ½ * 10,000 = 5,000
- 3rd most frequent word expected to occur 3,333 times

  1/3 * 10,000 = 3,333
- Expected number of occurrences of 100th most frequent word?

Zipf's Law

- Out of a work with N distinct words, the predicated probability of the word with rank k is:

$$f(k;s,N) = \frac{1/k^s}{\sum_{n=1}^{N}(1/n^s)}.$$

- s is constant based on distribution.
- In classic version of Zipf's law s = 1

Zipf's Law

- Assume 35,000 words
  - N = 35,000
- assume s = 1
- 35,000th harmonic number is about 11
- expected frequency of 10th word, k = 10
- Assume 1,000,000 words

$$\frac{1}{\sum_{n=1}^{N}(1/n^s)}$$

1,000,000 / 10 / 11 = 9,090

Alternate Formula

- Probability of a given word being the word with rank r
- R = number of distinct words

$$P(r) \approx \frac{1}{r \ln(1.78\,R)}$$

- Multiply by total number of words in word to get expected number of words

Approach

- Read "words" from a file
- determine frequency of each word
- sort words by frequency
- Compare actual frequency to expected frequency
  - many ways to define expected frequency
  - freq * rank = constant
  - estimate constant, simple
  - or use formulas

Java Program

- Eclipse IDE
- Create Project
- Create Class(es)
  - procedural approach
  - object based approach
  - object oriented approach

Calculating Frequencies

- Reading from a file
  - Scanner class
  - built in classes
  - documentation
  - exceptions
- Try reading into native array
- Try reading into ArrayList
  - show some of "words"
- better delimiter: "[^a-zA-Z']+"
  - regular expressions

Calculate Frequencies

- Don't need to store multiple copies of every word
- Just the number of times a given word appears
- Another class / data structure is useful
  - A Map, aka a Dictionary
  - key, value pairs
  - HashMap or TreeMap, order of keys

Using the Map

- Read in words, count frequencies
  - "wrapper" classes
- Read in and print out some of the map
- TreeMap
  - ordered by keys
- HashMap
  - seemingly Random order
- We want sorted by frequency
  - why can't we use another map?

Sorting by Frequency

- Create another class, WordPair
- Have the class implement the Comparable interface
  - define compareTo method
  - 2 objects / variables involved
- Add to ArrayList, use Collections.sort
- Now list start of ArrayList

Does Zipf's Law Hold?

- plot rank vs. frequency on a log - log scale
  - should be a near straight line
- recall freq * rank = constant
- Estimate constant
  - simple average of first 1000 terms?
  - simple average of all words with freq > 10?
  - Simple linear regression, best fit line to log - log plot

Viewing Results

- Compare predicted frequency and actual frequency of top 100 words and % error