Homework #1: CMPT-825

Due in class on Sep 12, 2003

Anoop Sarkar – `anoop@cs.sfu.ca`

(1) Text processing using Perl: For all sub-parts of this question, use the text file called `hw1.txt` from the location provided on the course web page. Each program should be run from the command line as follows:
`% perl your-program.pl hw1.txt` ⟨*cr*⟩

a. Write a Perl program to print out only the lines which contain the word ␣*plants* (e.g. it should not match *plant* or *implants* or even *plantocracy*), where ␣ is a whitespace character. There should be 45 lines in your output. You can count the lines in your output using `wc`:

```
perl your-program.pl hw1.txt | wc -l
```

Here's another way to use standard Unix tools to achieve the same output as your Perl code:

```
cat hw1.txt | tr ' ' '\n' | grep "^plants$" | wc -l
```

b. Write a Perl program that takes the input file and prints out those lines which contain the word *plant* with any kind of suffix (e.g. it should match ␣*plant* and ␣*plants* but not *implants*) **and** any word containing the string *flower* (e.g. it should match *flowers* and *wildflowers*). Similarly, print out the lines matching the word *plant* with any suffix **and** any word containing the string *manufacturing*.

Notice that the two subsets of the original text provided by your Perl program correspond in a very crude way to two different meanings of the word *plant*, one meaning is synonymous to *plant-life* and the other to *factory* .

c. Write a Perl program that collects the frequency of each word and prints it out sorted by descending frequency. The first few lines of the output should look like this:

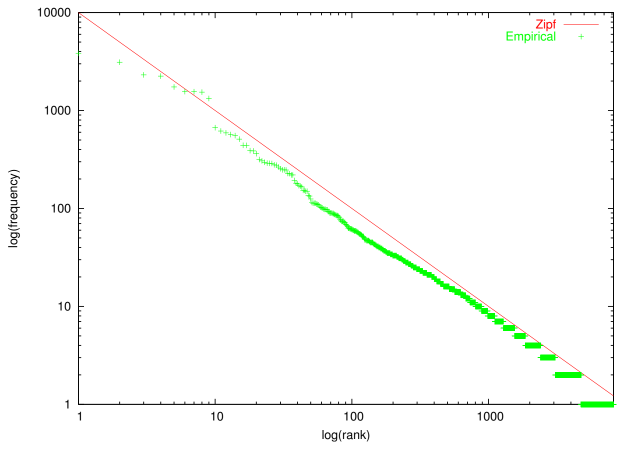
```
3842 ,
3116 the
2310 plant
2244 .
1740 a
...
```

(2) (20pts) **Zipf's Law** (see page 23 of the textbook). Modify your code from Q.1c to print out the line numbers for each word and its frequency as shown below:

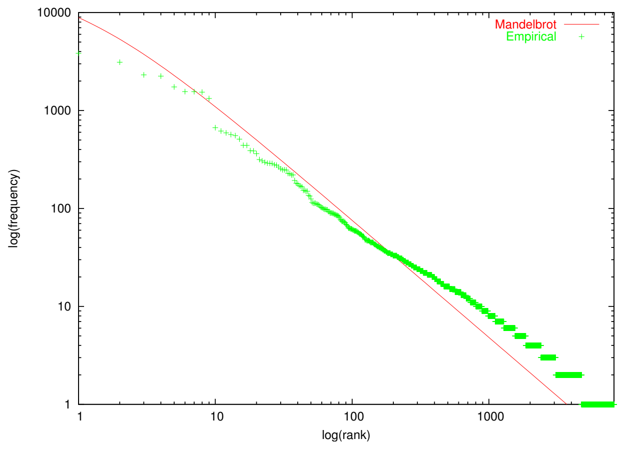
```
1     3842 ,
2     3116 the
3     2310 plant
4     2244 .
5     1740 a
...
```

Save the entire output of your program to a file called `hw1-2.plot`:

```
perl hw1-1.pl hw1.txt > hw1-2.plot
```



(a)



(b)

Figure 1: (a) Zipf's formula compared with the empirical distribution of word frequencies (b) Mandelbrot's formula compared with the empirical distribution.

The line numbers correspond to the sorted rank r of each word based on its frequency f. Zipf's law states that $f \propto \frac{1}{r}$ or, equivalently, that $f = \frac{k}{r}$, for some constant factor k. In this question, we will use the GNU tool `gnuplot` to plot Zipf's formula against the empirical relationship between a word's rank and its frequency.

At the unix shell, enter the command `gnuplot` and enter the following commands at the `gnuplot` shell. This will produce a plot that compares Zipf's formula with the empirical distribution you have stored in `hw1-2.plot`. We plot on the log scale, plotting $log(r)$ on the x-axis and $log(f)$ on the y-axis. Since there are 8200 tokens in our file `hw1.txt`, we vary $r = 1 \ldots 8200$ and set the value $k = 10000$. The `gnuplot` commands are:

```
set logscale
set xlabel "log(rank)"
set ylabel "log(frequency)"
plot [x=1:8200] 10000/x title "Zipf", 'hw1-2.plot' using 1:2 title "Empirical"
```

Now compare Zipf's formula with the Mandelbrot formula

$$f = P(r + \rho)^{-B}$$

or

$$log(f) = log(P) - B \cdot log(r + \rho)$$

This particular command to `gnuplot` uses the parameter settings: $P = 10000$, $\rho = 100$ and $B = 1.15$.

```
set logscale
set xlabel "log(rank)"
set ylabel "log(frequency)"
plot [x=1:8200] [1:] 10000*(x + 100)**(-1.15) title "Mandelbrot", \
  'hw1-2.plot' using 1:2 title "Empirical"
```

You can save your output as a postscript file by using the following commands *before* you enter the plot command:

```
set terminal postscript eps color
set output "filename.eps"
```

a. Compare your output with Figures 1.1 and 1.2 from the textbook (pages 26–27). Change the parameters P, B and ρ in the Mandelbrot function to match the observed distribution of word frequencies for `hw1.txt`.

b. What happens to the Mandelbrot equation when $B = 1$ and $\rho = 0$?

c. Ponder the implications of Zipf's law for natural language processing. Let's assume you have an NLP application which uses a dictionary of words. Based on Zipf's law or Mandelbrot's law or the empirical distribution plotted above, approximately how many words taken from a new (previously unseen) text do you expect to match in your dictionary.

d. Is Zipf's Law some deep property of language, or is it simply that any random (stochastic) process would have the same property? Confirm or deny this hypothesis using the following experiment: use the file `genRandomWords.pl` which creates random words out of the (lowercase) English alphabet. Running `perl genRandomWords.pl 100000` should provide enough material to plot a graph which you can compare to real English data.

(3) (80pts) **Minimum Edit Distance**. Download the perl program `edit-distance.pl` from the location provided on the course web page. It implements a function `minEditDistance` which computes the minimum edit distance between two strings. Here are two examples of the output produced by `edit-distance.pl`:

```
% perl edit-distance.pl gamble gumbo
levenshtein distance = 5

% perl edit-distance.pl "recognize speech" "wreck a nice beach"
levenshtein distance = 14
```

Copy `edit-distance.pl` to a new file `edit-distance-align.pl` and then extend `edit-distance-align.pl` by modifying the function `minEditDistance` so that you can memorize or trace back your steps from the final score for the minimum distance alignment (for each substring record whether it was an insertion, deletion or substitution that provided the best alignment). Then, pass this information to a single new function `printAlignment` which produces the following visual display of the best (minimum distance) alignment:

```
% perl edit-distance-align.pl gamble gumbo
levenshtein distance = 5
g a m b l e
|   | |
g u m b _ o

% perl edit-distance-align.pl "recognize speech" "wreck a nice beach"
levenshtein distance = 14
_ r e c _ _ o g n i z e   s p e e c h
  | | |         | |   | |     |   | |
w r e c k   a   n i c e   _ b e a c h

% perl edit-distance-align.pl execution intention
levenshtein distance = 8
_ e x e c u t i o n
      |     | | | |
i n t e _ n t i o n
```

The 1st line of the visual display shows the *target* word or phrase and the 3rd line shows the *source* word or phrase. Deletion from the source is represented as an underscore '_' in the 1st line with the letter being deleted from the source in the 3rd line (aligned with the underscore char). Insertion in the target is represented as an underscore in the 3rd line aligned with the inserted letter in the target on the 1st line. Finally, if a letter is unchanged between target and source then a vertical bar (the pipe symbol '|') is printed aligned with the letter in the 2nd line.

Apart from the use of references (used to pass multiple arrays to a function in `edit-distance.pl`), this program does not require any advanced knowledge of Perl.