Perl Weekly Challenge - 005

2019-04-23 08:20 +00 • Mark Senn

Challenge #1

From Perl Weekly Challenge - 005 retrieved on 2019-04-22 at 14:12 +00:

> Write a program which prints out all anagrams for a given word. For more information about Anagram, please check this wikipedia page.

From the wikipedia page:

> An **anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. For example, the word *anagram* can be rearranged into *nag a ram*, or the word *binary* into *brainy*.
>
> The original word or phrase is known as the subject of the anagram. Any word or phrase that exactly reproduces the letters in another order is an anagram.

Choose list of words

I'm going to use the /usr/share/dict/words file for the list of words. Copy "/usr/share/dict/words" to "words.txt". This is to make some one-liners below short enough to fit on one physical line.

```
cp -i /usr/share/dict/words words.txt
```

This one-liner counts the number of words:

```
perl6 -e lines.elems.say words.txt
479829
```

Ugh, 479,829 lines—too much data to test the program quickly. I'm not interested in words that contain characters that are not lowercase letters. This one-liner counts the number of all-lowercase words:

```
perl6 -e 'lines.grep(/^<lower>+$/).elems.say' words.txt
355542
```

How many unique all-lowercase words are there?

```
perl6 -e 'lines.grep(/^<lower>+$/).unique.elems.say' words.txt
355542
```

Ugh, 355,542 lines—too much data to test the program quickly. I'm only interested in six-letter words to whittle the data down to a manageable size. Put all six character lowercase letter words in "words6.txt".

```
perl6 -e 'lines.grep({/^<lower>+$/ && .chars==6}).sort.map({.say})' words.txt > words6.txt
```

There are 28,447 six-letter words.

Relevant mathematics

A "set" is a list of unique elements. Let S be a set of n letters with letters s_1, s_2, ..., s_n. To make a word, choose one of the n letters, then one of the $n-1$ remaining letters, until there is one letter left that you must choose. There are

$$n \times (n-1) \times \cdots \times 1 = n! \tag{1}$$

ways to arrange the unique letters.

A "bag" is a list of unique elements and how many times they occur. Because some or all of the letters can be repeated a bag must be used to describe this challenge. Let B be a bag of n unique letters where some of the letters may be repeated. Let the unique letters be b_1, b_2, ..., b_n and c_1, c_2, ..., c_n be the count of how many times each letter occurs. There are (see (1))

$$\left(\sum_{i=1}^{n} c_i\right)!$$

ways to arrange the letters. But each unique letter can be arranged in (again, see (1))

$$\prod_{i=1}^{n} c_i!$$

so the total number of different letter arrangements is

$$\frac{\left(\sum_{i=1}^{n} c_i\right)!}{\prod_{i=1}^{n} c_i!}$$

Checking:

letters: one `a` and two `b`'s $= \{\texttt{abb}, \texttt{bab}, \texttt{bba}\}$ math: $3!/2 = (3 \times 2)/2 = 3$ checks ok!

A furthur simplification

The subject of an anagram can be one or more words and the resulting anagram can be the same or a different number of words. The subject "abcdef" can be anagrammed to "cab fed". Anagramming to a phrase that doesn't mean anything is not satisfying to me intellectually. I don't know of an easy way to determine what phrases have meaning and which don't.

Anagramming from one word to multiple words is computationally expensive.

From Partition

> A partition is a way of writing an integer n as the sum of positive integers where the order of the addends is not significant...

A six-letter word can be partitioned into words of the following lengths:

Partitions
6
5 + 1
4 + 2
4 + 1 + 1
3 + 3
3 + 2 + 1
3 + 1 + 1 + 1
2 + 2 + 2
2 + 2 + 1 + 1
2 + 1 + 1 + 1 + 1
1 + 1 + 1 + 1 + 1 + 1

That's eleven different ways—but, it is even worse than that because "5 + 1" means it could be split into "1 + 5" characters. For each partition all permutations of that partition must be tried.

From Permutations P(n,r)

> Number of different permutations of n objects where there are n_1 repeated items, n_2 repeated items, ... n_k repeated items
>
> $$\frac{n!}{n_1!n_2!\ldots n_k!}$$

So

Partition	Permutations of partition
6	1! / (1!) = 1
5 + 1	2! / (1! 1!) = 2
4 + 2	2! / (1! 1!) = 2
4 + 1 + 1	3! / (1! 2!) = 3
3 + 3	2! / (2!) = 1
3 + 2 + 1	3! / (1! 1! 1!) = 6
3 + 1 + 1 + 1	4! / (1! 3!) = 4
2 + 2 + 2	3! / (3!) = 1
2 + 2 + 1 + 1	4! / (2! 2!) = 6
2 + 1 + 1 + 1 + 1	5! / (1! 4!) = 5
1 + 1 + 1 + 1 + 1 + 1	6! / (6!) = 1

Every six letter string can be split into $1 + 2 + \cdots + 1 = 32$ character strings.

The program will only anagram six-letter one-word anagrams to other six-letter one-word anagrams.

The same program is used to solve challenges #1 and #2. See the program listing in the Challenge #2 section.

Challenge #2

From Perl Weekly Challenge - 005 retrieved on 2019-04-22 at 14:39 +00:

> Write a program to find the sequence of characters that has the most anagrams.

The Perl6 program:

```
# Perl Weekly Challenge - 005
# Challenges #1 and #2
# (This program solves both challenges.)
#
# See
#     engineering.purdue.edu/~mark/pwc-005.pdf
# for more information.

# Run using Perl 6.
use v6;


#
#  Challenge #1.
#

# Read the words in "words6.txt" to @word.
my @word = lines 'words6.txt'.IO;

# Define %hash.
# Each key is a string with the letters in lexigraphic order.
# Each value is an array of words that use those letters.
my Array %hash;

# Construct the array.
for (@word)
{
    # The key for "family" is "amfily".
    my $key = .comb(/./).sort.join;
    # Add the current word to the hash.
    %hash{$key}.push($_);
}

# Print the array.
# for (%hash.keys.sort) -> $key
# {
#     print "$key:";
#     for (%hash{$key}.Array)
#     {
#         print " $_";
#     }
#     print "\n";
# }

# Pick a random word.
my $word = @word.pick;

# Convert $word to $key.
my $key = $word.comb(/./).sort.join;

# Print the output.
print "Challenge 1\n";
print "print all anagrams for a word\n";
```

```
print "original word: $word\n";
print "anagrams:     ";
for (%hash{$key}.Array.sort)
{
    ($word eq $_)  or  print " $_";
}
print "\n";


#
#  Challenge 2.
#

my $n = 0;
for (%hash.keys)
{
    my $t = %hash{$_}.Array.elems;
    if ($t > $n)
    {
        $n = $t;
        $key = $_;
    }
}

# Print the output
print "\n";
print "Challenge 2\n";
print "sequence of letters with most anagrams\n";
print "letters:  $key\n";
print "anagrams:";
for (%hash{$key}.Array.sort)
{
    ($word eq $_)  or  print " $_";
}
print "\n";
```