Assignment Three
DYPL/ID2005, Dynamiska programmeringsspråk
Spring Term 2010

Beatrice Åkerblom
`beatrice@dsv.su.se`

6th May 2010

Introduction

The purpose of this assignment is to revisit some results of Lutz Prechelt's [1]. Prechelt has made empirical comparisons of different languages, including C, C++, Java, Perl, Python, Rexx, and Tcl. Interesting results from Prechelt's experiment include different productivity ratings in terms of LOC/hour (LOC = Lines Of Code) in different languages, program lengths, etc.

In the third assignment, you are to implement a simplified version of the test program used in Prechelt's experiment twice in two different languages: once in a systems programming language that you are familiar with and once in a dynamic programming language. If you do not know any systems programming language, talk to Beatrice and we'll figure something out.

I suggest that you start with the language that you are most familiar with, e.g., starting with Java and moving on to Erlang.

Be sure not to miss the last section that details what simplifications you may make.

Language Selection

You are allowed to use any of the following languages (others may be OK too):

Systems PLs: C, C++, C#, Java, Object Pascal, Pascal, Objective-C

Dynamic PLs: Io, Perl, PHP, Python, Ruby, Smalltalk, Tcl, Erlang, Lisp, Scheme

If you do not know any of the systems programming languages listed, talk to Beatrice and we'll figure something out.

What To Hand In

In addition to the two solutions, we want a short paper (about 1.000 words) describing what you feel the differences are in programming in the different languages. Are there strenghts in the systems languages that you felt were missing in the dynamic ones, and vice versa? What language felt best suited for the task? Do you feel that there were any differences in how you approached the problem in different languages?

We also want you to clock the time it takes you to complete the programs (please be careful about this, and don't lie – it does not affect grading and we won't make it public). We suggest that you start with the language that you are most familiar with to even the odds a little. A possible result would be: Java 8 hours, Python 8 hours.

Task Description

Attention: Please follow these instructions super accurately.

First read through the whole of it in order to get an overview. Concentrate on the details only upon second reading.

The following mapping from letters to digits is given:

```
E | JNQ | RWX | DSY | FT | AM | CIV | BKU | LOP | GHZ
e | jnq | rwx | dsy | ft | am | civ | bku | lop | ghz
0 |  1  |  2  |  3  |  4 |  5 |  6  |  7  |  8  |   9
```

We want to use this mapping for encoding telephone numbers by words, so that it becomes easier to remember the numbers.

Functional Requirements

Your task is writing a program that finds, for a given phone number, all possible encodings by words, and prints them. A phone number is an arbitrary string of dashes and digits. The words are taken from a dictionary which is given as an alphabetically sorted ASCII file (one word per line).

All encodings that are possible from this dictionary and that matches the phone number exactly shall be printed. Thus, possibly nothing is printed at all (e.g., if the dictionary is empty). The words in the dictionary contain letters (capital or small, but the difference is ignored in the sorting). For the encoding only the letters are used, but the words must be printed in exactly the form given in the dictionary.

Encodings of phone numbers can consist of a single word or of multiple words separated by spaces. The encodings are built word by word from left to right. If and only if at a particular point no word at all from the dictionary can be inserted, a single digit from the phone number can be copied to the encoding instead. Two subsequent digits are never allowed, though. To put it differently: In a partial encoding that currently covers k digits, digit k+1 is encoded by itself if and only if, first, digit k was not encoded by a digit and, second, there is no word in the dictionary that can be used in the encoding starting at digit k+1.

Your program must work on a series of phone numbers; for each encoding that it finds, it must print the phone number followed by a colon, a single space, and the encoding on one line; trailing spaces are not allowed. All remaining ambiguities in this specification will be resolved by the following example. (Still remaining ambiguities are intended degrees of freedom.)

Sample dictionary:

```
an
blau
Bo
Boot
bos
da
Fee
fern
Fest
fort
je
jemand
mir
Mix
Mixer
Name
neu
od
Ort
so
Tor
Torf
Wasser
```

Sample phone number list:

```
112
562482
4824
07216084067
107835
10789135
381482
04824
```

Corresponding correct program output (on screen):

```
562482: mir Tor
562482: Mix Tor
4824: Torf
4824: fort
4824: Tor 4
107835: neu od 5
107835: je bos 5
107835: je Bo da
381482: so 1 Tor
04824: 0 Torf
04824: 0 fort
04824: 0 Tor 4
```

Any other output would be wrong (except for different ordering of the lines).

Wrong outputs for the above example would be e.g.

1. **4824**: 4 Ort, because in place of the first digit the words Torf, fort, Tor could be used,
2. **10789135**: je Bo 9 1 da , since there are two subsequent digits in the encoding, 3. 04824: 0 Tor , because the encoding does not cover the whole phone number, and
3. **562482**: mir Torf , because the encoding is longer than the phone number.

Quantitative Requirements

1. Length of the individual words in the dictionary: 50 characters maximum
2. Number of words in the dictionary: 200.000 maximum

3. Length of the phone numbers: 50 characters maximum
4. Number of entries in the phone number file: unlimited

Qualitative Requirements:

- Your implementation must be fairly efficient in terms of memory use/data structures/time complexity of searches. For example, making one huge flat list and search for matches in that is (most likely) a bad implementation that will be failed (unless it can be sufficiently motivated).

In short, write good programs that does not just give the correct answer, but does it in a good way, without consuming too much computer power. If your test programs takes more than a couple of minutes for a small data sample (i.e., the small one given below), that's a dead give-away that your program is not good enough.

A More Interesting Word List

You may use any other English wordlist instead. Here you find one listing the top 1000 words on the internet:

`http://www.andreas.com/faq-thousandwords.html`

Permitted Simplifications

1. You may assume that words in the dictionary are all lowercase and match the regexp /[a-z]+/ (this is assumed by the test program described below).
2. You need only search for completions that create complete matches of one or more words, without digits (in the examples above, only 562482: mir Tor, 562482: Mix Tor, 4824: Torf, 4824: fort and 107835: je Bo da would be required.)

NOTE: both your implementations must make the exact same simplifications. State which ones you make use of!

Test Program

`http://people.dsv.su.se/~tobias/attach/validate.rb` Here you find a very silly test program that simply checks that the data generated by your solution was correct. Note that it gives an error if it sees the same number appear twice in an instream, which actually might happen.

Run the program like this:

```
 ./validate.rb outfile
```

or simply pipe the result of your program to the validate program, e.g.,

```
 ./mysolution dict nums | ./validate.rb
```

Not that the validator does not check that your program covered all cases – it just checks that the cases you did cover were valid.

Bibliography

[1] Lutz Prechelt, *"An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl"*, `http://projects.mi.fu-berlin.de/w/bin/view/Main/LutzPrechelt`