UMEÅ UNIVERSITY **March 19, 2014**
Department of Computing Science
Obligatory Exercise

Programing Languages

Obligatory Exercise 2

Short Message Service

Name Simon Nilsson
E-mail c09snn@cs.umu.se
Path ~c09snn/edu/5dv086/lab2

Grader

Jan Erik Moström
Petter Ericson

1 Introduction

T9 is a technology used in phones to quickly type text without using a keyboard, even though most new phones uses a virtual keyboard via touch screens. The process is based on having many letters and symbols on each key. Most of the latter systems, such as T9, were dictionary based and reduces the number of key buttons pressed by mapping the input to the dictionary. The dictionary also picks, unless the user choose something else, the most common word.

The goal is to implement a T9 system using Haskell. The system will take a message input and return a minimum sequence of key pressed that produces a given message for a particular dictionary augmented with word counts.

The system must have the following properties:

- A message is a string of upper-case letters A-Z and blanks.
- An input string is a sequence of digits (0,2-9) and the special character ^.
- A prefix is a sequence of digits (2-9) representing letters.
- A prefix can be modified any number of times.
- Each (modified) prefix is mapped into a word as follows.
  - The prefix together with a dictionary determines a set of possible words.
  - The possible words are sorted in decreasing order of frequency.
  - Ties are broken by ordering the words lexicographically.
  - If there are no possible words, the prefix maps into the special character ?.
  - A prefix which has been modified N times maps into the (N+1)-th word in the list.
  - If (N+1) exceeds the length of the list, then it wraps around the list.

This implementation uses a pre-defined dictionary that can be found here: `http://www8.cs.umu.se/kurser/5DV086/VT14/resources/Dictionary.hs`

The original specification can be found at:
`http://www8.cs.umu.se/kurser/5DV086/VT14/xlab2.html`

2 User Guide

The implementation can be found at `c09snn/edu/5DV086/lab2` and also on `GitHub`[1] at the address `https://github.com/SNilsson/5dv086-lab2` The solution requires a file named `Dictionary.hs` to work correctly. For verification and testing purpose the files `Dictionary.hs`, `test.hs` and `Messages.hs` is added to the final submission.

[1] Type of revision control, also known as version control and source control is the management of changes to documents, computer programs, large web sites, and other collections of information.

The implementation follows the specification provided with additional modifications:

The implementation sorts the sequences of words by their priority and then by name.

To run the T9 implementation the user must have `ghci`[2] installed and through it load `T9.hs`. Then execute the method `get_t9_sequence` that accepts a string as input. When imported from other files the only accessible method that can be run is `get_t9_sequence` and it should be the only one the user executes.

`test.hs` can be used to check all the messages in `Messages.hs`.

3 Implementation

The implementation is fairly straight forward with few algorithms. `get_t9` uses other hidden functions that helps it decipher:

- The amount of keys pressed to generate the word
- The shortest sequence of keys pressed to generate the word
- How many ^that should be used to get the correct word or word sequence given

3.1 `get_shortest_sequence`

This function takes a sequence of characters, i.e a word and a dictionary as input and returns a sequence of keys pressed to create the word.

4 Tests

The following section will test the messages in the `Messages.hs` file and print the result in a file. This file is attached in the file. Note that the program will not write the file for the user! The file just creates output called a, while in the program use `writeFile "file_name" a` to create a nicely printed file of the results. The file `test.txt` is attached at the end of this report.

[2]GHC is a state-of-the-art, open source, compiler and interactive environment for the functional language Haskell. GHCi is GHC's interactive environment, in which Haskell expressions can be interactively evaluated and programs can be interpreted.