The Art of Virtual Keyboard

Yubo Feng, yuf24@pitt.edu

1. Overview

This homework aims at building a virtual keyboard that includes one sample sentence display region, current text display region and one virtual a QWERTY layout keyboard region. The virtual keyboard contains the basic function of click-and-input, and also implements the shorthand writing (that is draw a line pass through the letters you need to get a word input). Moreover, we implement measurement interface through which users could input their expected input text then compare it with keyboard output to evaluate keyboard identify accurate.

2. Language and platform

This project is written in C# and built on visual studio 2013. Running on Win8.1 OS or Win7 OS with Win8 Developer packet plus.

3. General design

The general design is shown in Figure 1.

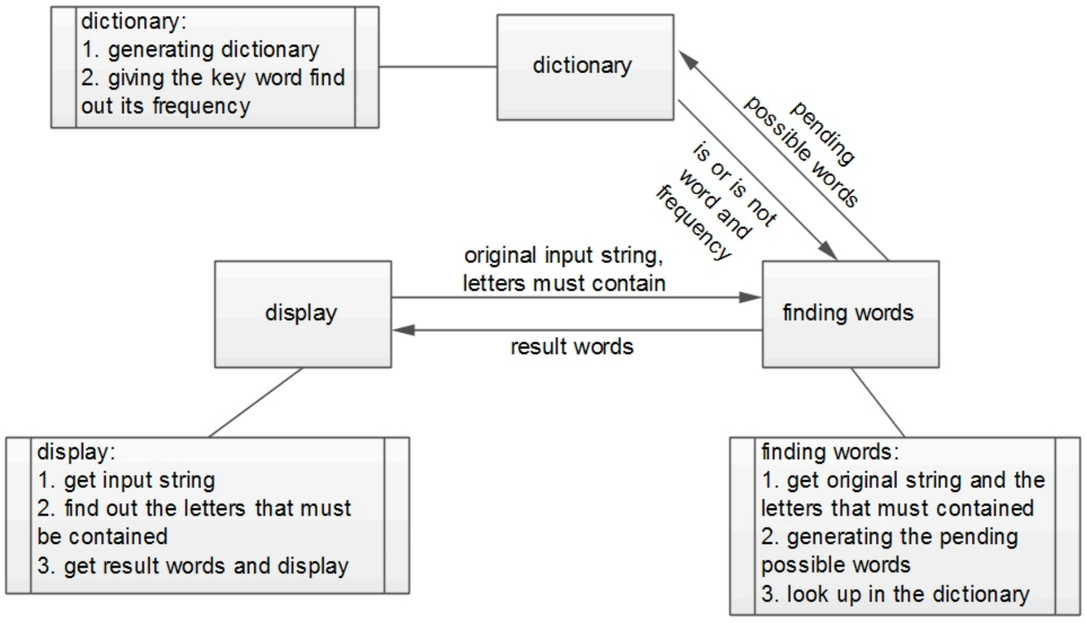
dictionary:
1. generating dictionary
2. giving the key word find out its frequency
dictionary
pending possible words
is or is not word and frequency
original input string, letters must contain
result words
display
finding words
display:
1. get input string
2. find out the letters that must be contained
3. get result words and display
finding words:
1. get original string and the letters that must contained
2. generating the pending possible words
3. look up in the dictionary

Figure 1. General design

4. Implementing details

Part 1. Display

1) The designs other than the basic keyboard layout:
   a) Lager Size: different from basic keyboard layout, the size of virtual keyboard is larger than normal keyboard including keys' size and layout since it will make identify easier: one difficulty that identify users input keys is to determine whether the key in the shorthand

writing is the one that users wants, however, the users' expectation is hard to predict, you cannot say this key is used rather than that, cause what word users input is unknown. So, in order to fix this problem, we build a keyboard with larger size which cooperates with distance filter mechanism both of which we will explain later.

b) Feedbacks: Path trace and Superscript notation. In order to let users to know whether he or she selects the letter or not, we draw the path on the keyboard that users passes: when the user uses 'shorthand writing' to input, a line will appear along with user's moving path to indicate its path as well as a number marked on the top right corner reminding users that this key is "pressed" once. Moreover, we design a kind of distance filter mechanism to identify the key that users want: if the point in the line is close enough even beyond a boundary, then we could say this point is the one that user desires.

c) Feedback in double letter: When the user try to type a word contains two same successive letters, they can click the right button on mouse with the left button keep pressing down. Then a notice of number '1' will turn to '2' appearing on the upper right corner of this key. That is to indicate user they have already typed this letter twice.

2) The letters that must be contained: the first letter, the last letter and the letters in the turning point of the drawing line, and the successive same letters.

The feature of turning point solves the problem in the homework requirement that why the different paths in figure 2 and figure 3, and one can generate the word "TEST" while the other one cannot. Because when users try to input some certain letters, they are likely to go directly from the previous letter to the next one rather than take a detour. So it is the point of users' habit we should focus on. The letters that must be contained:

$$s0[i] = \begin{cases} 1 & \text{the letters must be contained} \\ 0 & \text{other letters} \end{cases}$$

Part 2. Dictionary

The dictionary part adopts the dictionary class in C#. The dictionary represents a collection of keys and values implemented by hashing algorithms. In this project, we store the words as key and its frequency as the value. The dictionary:

Dictionary <Word, Frequency>

Part 3. Finding the right words

After getting the original string, this part need to deal with the original string into some possible combination, then check whether the combinations are in the dictionary, and sort the result words by their frequency.

1) Data structures
   Class Word: stores the result word the its frequency, and sort the words by its frequency.
   Class FindWords: Generate the possible combinations, checking the pending words in the dictionary, sort the result words by their frequency.
2) Generate possible combination from original input string.
   Using recursion to implement generating. Every recursion determines whether or not contain one letter according to the information from 'Must contained letters'. If the letter must contain, add it to the current string; if the letter is not 'must be contained' then there are two possible ways of contain and not contain, for each one generate the new prefix. Then pass the prefix down to the next recursion. Finally we get all the possible combinations.
   (i + 1)th recursion
   $$\text{(ith letter `a')} = \begin{cases} f(\mathrm{s}+'\mathrm{a}', \mathrm{i}+1) & \mathrm{s0[i]} = 1 \\ f(\mathrm{s}, \mathrm{i}+1), f(\mathrm{s}+'\mathrm{a}', \mathrm{i}+1) & \mathrm{s0[i]} = 0 \end{cases}$$
3) Finding result words.
   Check each pending word in dictionary, if it exists then creates a Word(word, frequency). Then sort these result words by frequency.
4) Sending back the result words to display.