Walter Cazzola

Home Page
ADAPT Lab.
Curriculum Vitae
Research Topic

Didactics

Publications

Funded Projects

Research Projects

Related Events

Exam of Advance in Programming

3 September 2012

Disclaimer. Note that to have a running solution for an exercise is not enough: you need a well-cooked solution that proves your ability to use what explained during the classes. The worth of the 2 exercises is the same (15 points). To pass the exam you have to do both exercises. The submissions with only one exercise will not be evaluated at all.

Exercise 1: Swyping on the Phone.

Swype is an input method for touch screens where the user enters words by sliding a finger or stylus from letter to letter, lifting only between words. It uses error-correcting algorithms and a language model to guess the intended word. Swype is designed for use on touchscreen devices with a traditional QWERTY keyboard. In the picture you have the swype for the word «quick» as an example.

The algorithm implemented in the smartphones is really advanced but it is not difficult to get a good approximation that extracts some suggestions from a swype, i.e., the sequence of keys you touch. To do that we can do some undertakings and considerations:

1. all the characters in the intended word are in the swype;
2. the first and last letter of the swype are the first and last letter of the intended word;
3. between two characters of the word there are many possible useless characters in the swype; and
4. The number of traversals between different rows of the keyboard gives us a fair idea about the length of the word.

Of course the suggestions come out from a dictionary and the above considerations must be used to skim all the words that absolutely cannot be the intended word.

In this exercise, you have to write a function `get_suggestion` that given a swype produces a list of possible intended words for that swype by considering the above suggestions.

The following is an example of the expected computation.

```
from swype import *

if __name__ == '__main__':
    test_cases = \
       ['heqerqllo',                          # hello
        'qwertyuihgfcvbnjk',                  # quick
        'wertyuioiuytrtghjklkjhgfd',          # world
        'dfghjioijhgvcftyuioiuytr',           # doctor
        'aserfcvghjiuytedcftyuytre',          # architecture
        'asdfgrtyuijhvcvghuiklkjuytyuytre',   # agriculture
        'mjuytfdsdftyuiuhgvc',                # music
        'vghjioiuhgvcxsasdvbhuiklkjhgfdsaserty', # vocabulary
       ]

    for test in test_cases:
```

```
[15:35]cazzola@surtur:~/pa/es1>python3 main-swype.py
['hello','hero','ho']
['quick']
['wed','weird','weld','wild','wold','word','world','would']
['doctor','door','dour']
['architecture']
['acute','adjure','afire','agile','agriculture','argue','article','astute
['music','mystic']
```

Exercise 2: Tagging the Clouds.

Every day, thousands new pages are created on the Internet and people are looking for the most varied topics online. The real problem is to help the users to find what they are looking for. Many search engines carry out this task but to do that the huge amount of data available must be semantically classified. The search engines are used really advanced algorithms.

A quite naïve approach to semantic classification can be realized by digging out some tags as the most frequently used terms in the web page we are classifying Of course, not all terms are significant, e.g., for sure in any English written page the most used word is «the» but it does not give any hint on the real content of the web page; normally the most significant are the nouns.

In this exercise we have to implement a function `tags` that given a file representing the text of a web page returns a list of all the nouns found in the text that occur more than 9 times. The list must be sorted by frequency (from higher to lower) apart those nouns with the same frequency. Consider the following set of nouns and remember to filtering out the whole punctuation and to consider the words independently of the case when looking for the tags. To double check each word in the list must contain its frequency as well.

The following is an example of the expected behavior. The test files are Australia, Aardvark and Computer.

```
from tagging import *

if __name__ == "__main__":
  t1 = tags("australia.txt")
  t2 = tags("computer.txt")
  t3 = tags("aardvark.txt")
  print(t1)
  print(t2)
```

```
[15:38]cazzola@surtur:~/pa/es2>python3 main-tagging.py
[('australia', 100), ('australian', 39), ('territory', 20),
 ('government', 18), ('country', 15), ('east', 14), ('continent', 14),
 ('state', 13), ('governor', 11), ('island', 11), ('asia', 10), ('land',
[('computer', 96), ('memory', 32), ('machine', 27), ('instruction', 18),
 ('architecture', 16), ('time', 16), ('language', 16), ('control', 16),
 ('arithmetic', 15), ('use', 14), ('unit', 11), ('number', 11),
 ('system', 10), ('software', 10)]
```

Last Modified: Mon, 24 Sep 2012 15:37:59 ADAPT Lab.