AlaOperator - assignment solution

Luminita Moruz

September 29, 2013

1 Problem formulation

An algorithm should be implemented that takes as input a list of operators, with each operator including a collection of offers expressed as prefix, price. For a given phone number, the algorithm should output the cheapest operator that has a prefix matching that phone number. When several prefixes of an operator match a number, the longest should be chosen.

2 Implementations

Two straightforward solutions were implemented, both briefly described below.

2.1 Solution 1 - simple approach

Let n be the number of operators. Since each operator may potentially offer a large number of prefixes, I chose to store their prefixes and prices as dictionaries, with prefixes as keys and prices as values. This implies that I store the data in a list $D = [D_1, D_2, ..., D_n]$, where each D_i is a dictionary storing the prefixes and prices of operator i.

Let $x = x_1x_2...x_m$ be the phone number we search, where m is the number of digits. My algorithm starts by searching the full number $x_1..x_m$ in all the n dictionaries. In the next step, the last digit of the number is dropped, meaning that only $x_1..x_{m-1}$ is searched in the dictionaries of the operators. Following, in the next step only $x_1..x_{m-2}$ is searched and so on. If at a certain point the number is found in the dictionary D_i of operator i, then no search will be performed in D_i in the next iterations. This way I make sure that the longest match is returned for a given operator. The procedure is repeated until we reached x_1, or until the number was found in all $D_i, i = 1..n$.

The search in a dictionary in python has in average $O(1)$ complexity. Since we have n dictionaries, and a search is performed for each prefix of the phone number, the complexity of the algorithm would be $O(mn)$, where m is the length of the phone number.

2.2 Solution 2 - optimized approach

When implementing the solution above, I realized that in practice the phone number will be much shorter than the prefixes. Thus, instead of starting by searching the full word

$x_1x_2...x_m$in all the dictionaries, I start instead by searching $x_1x_2...x_p$, where p is the length of the longest prefix observed at any of the operators. p is calculated when the data is loaded.

In this case, the complexity is reduced to $O(np)$.

3 Limitations and possible improvements of current implementation

The solutions provided here can be improved in various ways. In addition, depending on the characteristics of the input data, more efficient solutions can be designed. Some of my own thoughts regarding these are given below.

Depending on the data, an improved implementation may include building one dictionary storing the information of all the operators, rather than n separate dictionaries. Also, if a larger number of phone numbers are to be searched, then a dictionary including solutions for the prefixes searched up to that point can be maintained.

The current implementation is a small proof-of-principle. As an example, no proper validation of the input data is performed, which would be a must in a real world application. In addition, my program does not implement a proper exception handling, neither it provides informative error messages. It rather relies on the user to provide correct input data :).

Regarding the unit tests, they are by no means complete. In addition, they make use of real files when testing some of the functions. In a real application, this may slow down the tests significantly. Instead, mock objects could be used.

The system tests evaluate the functionality of the program for only a few simple examples. In reality, one should test the program for more complex input. Also, it should be tested that the correct error messages are returned when erroneous input data is provided.