UPPSALA UNIVERSITY
Mats Dahllöf

Programming courses 5LN446 5LN715
3 October 2016

5LN446/5LN715 Assignment 2

Submission only need to include concisely commented and well-designed code, written reports are not required. Please make sure that your code complies with the specification, above all that all classes, functions and methods take the right number of arguments, and return objects of the right type. In well-designed code functions and variables are named in such a way that their point is easy to grasp. (For more information see course PM.)

Assignments for both courses

(Assignment 2.1) Write a function that takes a list as its parameter and returns a list of all its permutations, e.g. `permutations([1,2,3])` returns `[[1, 2, 3], [1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]`.

(Associated exercise: make sure that you also know how to write a function that takes a *string* as its parameter and returns a list of all its permutations, e.g. `permutationsstr('abc')` returns `['abc', 'bac', 'bca', 'acb', 'cab', 'cba']`.)

(Assignment 2.2) Define a class `StackLL` which gives us the standard Stack operations and behaviour, and whose inside is defined in terms of linked lists (and a special `Node` class), i.e. no ordinary Python lists will be involved. (See the PS book.)

(Assignment 2.3) Define a function, based on the dynamic programming algorithm, which returns the list of the longest common substrings (there might be several of the same length) of two strings supplied as parameters.

Assignments compulsory only for 5LN715 (but useful for all)

(Assignment 2.4) As we have seen, Levenshtein distance can be computed and explained by means of a table. In this presentation I've added letter to each cell to say whether the its value is due to deletion, insertion, substitution, or matching (of equivalent letters).

		S	a	t	u	r	d	a	y
	0	1d	2d	3d	4d	5d	6d	7d	8d
S	1i	0m	1d	2d	3d	4d	5d	6d	7d
u	2i	1i	1s	2s	2m	3d	4d	5d	6d
n	3i	2i	2s	2s	3s	3s	4s	5s	6s
d	4i	3i	3s	3s	3s	4s	3m	4d	5d
a	5i	4i	3m	4s	4s	4s	4i	3m	4d
y	6i	5i	4i	4s	5s	5s	5s	4i	3m

Note that in case of conflicts operations have been preferred in this order: substitution, insertion, deletion.

Write a class `LevenshteinTable` that contains functions allowing us to construct instances like this:

```
lt = LevenshteinTable('Saturday','Sunday')
```

And then to compute the distance like this:

```
lt.distance()
```

After that the table should be filled with information and the `LevenshteinTable` should allow itself to be printed by means of:

```
print(lt)
```

You can start with a constructor like this:

```
class LevenshteinTable:

    def __init__(self,s1,s2):
        self.s1 = s1
        self.s2 = s2
        self.table =  [[0 for _ in range(len(self.s2)+1)] for _ in range(len(self.s1)+1)]
```

And then you have to decide how to represent the cells and compute the information in them. . .

(Assignment 2.5) Define a priority queue class. (Check what it means by googling.) Motivate your design choices concisely in the comments.