CS2121/9643 – Project 1
due Mar. 1, 2016 (latest to submit: Mar. 4)

1. (100pt) Write a Python program `editDist.py` to compute the edit distance between two strings. Recall that the edit distance gives the *minimum* number of operations (indels or replacements) to transform one string into another. Therefore, the dynamic programming algorithm in the alignments.pdf lecture can be used with `min` instead of `max`. The recurrence relation is:

$$DP[i,j] = \min \begin{cases} DP[i-1,j-1] + f(S[i],T[j]) \\ DP[i-1,j]+1 \\ DP[i,j-1]+1 \end{cases}$$

where

$$f(S[i],T[j]) = \begin{cases} 0, & \text{if } S[i]=T[j] \text{ (match: no operation required)} \\ 1, & \text{if } S[i]\neq T[j] \text{ (mismatch: one replacement required)} \end{cases}$$

and the initialization is

$$DP[i,0] = i, DP[0,j] = j, \text{ for all } i,j.$$

Your program will read the two strings from two files in the command line. You can use either

```
python editDist.py <file1.txt> <file2.txt>
```

or, in Canopy,

```
run editDist.py <file1.txt> <file2.txt>
```

You'll need `import sys` and open the first file with something like `f1 = open(sys.argv[1], "r")`. The program will print on the screen the edit distance between the given strings and a corresponding alignment. For example, if one file contains "`once upon a time`" and the other contains "`one pony is mine`," then the output should look like this:

```
edit distance = 7
optimal alignment:

once upon- -a time
|| || ||| |  | | |
on-e -pony is mine
```

For longer input strings, the alignment itself is to be broken in lines of 60 characters; here is an example:

```
edit distance = 34
optimal alignment:

ACATACGATACAGACGATCGGCTAGAATCCACCAGCTACAGCTAG-T-C---GATACA-G
||||||||||  | ||  |||||||||||||||||||||||||| | |   |||  | |
ACATACGATAC--A-GA--GGCTAGAATCCACCAGCTACAGCTAGTTACAAGGATCGATG

CACGAATCGCTAAACAG-CTCGATCGATCGCTAGCTGATCGATACTTACCACAGCTGATC
||||||   |||||||| || | |   |||||||||||||||||||||||||||||| |
CACGAA---CTAAACAGACTAG-TTTCTCGCTAGCTGATCGATACTTACCACAGCTAAAA

GATGCTATT-TAGCTAGCT-CGTAGTA
|||||||| | |  ||||   |  ||
GATGCTATTATTG-GAGCTAATTTTTA
```

To simplify the production of the output alignment, if the input files have multiple lines, then the newline characters will be replaced by spaces: `stringName.replace("\n", " ")`.

The structure of the code will not be considered for grading but it is strongly advised to implement clear logic, use meaningful names, and provide useful comments.

2. (bonus: 20pt) Use *linear space* as explained in the alignments3.pdf lecture.

3. (bonus: 20pt) Implement also the *search* variant where one string, called *pattern*, is searched for within the other one, called *text*. (See also the alignments3.pdf lecture.) Compute and return all substrings of *text* that have the smallest edit distance to *pattern*. (Quadratic space is fine here as *pattern* is assumed much shorter than *text*.)

Note Submit your solution on `owl.uwo.ca`:

- include all necessary Python files
- include a `readme` file explaining how you ran the program
- you may want to include your test files as well