Programming and algorithms

Practice 5 - bioinformatics

Objectives for the practice:

- Distance algorithm for sequences
- Find code online, adapt it to the specifics of our problem
- Algorithm for sequence alignment (local and global)
- Working on graphs

As always, encode all your functions in a file.
For your tests, you will use the following randomly generated DNA sequences:

```
DNA_samples = ['ACCATACCTTCGATTGTCGTGGCCACCCTCGGATTACACGGCAGAGGTGC',
               'GTTGTGTTCCGATAGGCCAGCATATTATCCTAAGGCGTTACCCCAATCGA',
               'TTTTCCGTCGGATTTGCTATAGCCCCTGAACGCTACATGCACGAAACCAC',
               'AGTTATGTATGCACGTCATCAATAGGACATAGCCTTGTAGTTAACAG',
               'TGTAGCCCGGCCGTACAGTAGAGCCTTCACCGGCATTCTGTTTG',
               'ATTAAGTTATTTCTATTACAGCAAAACGATCATATGCAGATCCGCAGTGCGCT',
               'GGTAGAGACACGTCCACCTAAAAAAGTGA',
               'ATGATTATCATGAGTGCCCGGCTGCTCTGTAATAGGGACCCGTTATGGTCGTGTTCGATCAGAGCGCTCTA',
               'TACGAGCAGTCGTATGCTTTCTCGAATTCCGTGCGGTTAAGCGTGACAGA',
               'TCCCAGTGCACAAAACGTGATGGCAGTCCATGCGATCATACGCAAT',
               'GGTCTCCAGACACCGGCGCACCAGTTTTCACGCCGAAAGCATC',
               'AGAAGGATAACGAGGAGCACAAATGAGAGTGTTTGAACTGGACCTGTAGTTTCTCTG',
               'ACGAAGAAACCCACCTTGAGCTGTTGCGTTGTTGCGCTGCCTAGATGCAGTGG',
               'TAACTGCGCCAAAACGTCTTCCAATCCCCTTATCCAATTTAACTCACCGC',
               'AATTCTTACAATTTAGACCCTAATATCACATCATTAGACACTAATTGCCT',
               'TCTGCCAAAATTCTGTCCACAAGCGTTTTAGTTCGCCCCAGTAAAGTTGT',
               'TCAATAACGACCACCAAATCCGCATGTTACGGGACTTCTTATTAATTCTA',
               'TTTTTCGTGGGGAGCAGCGGATCTTAATGGATGGCGCCAGGTGGTATGGA']
```

1 Distance algorithm

Task 1 *Warm-up : implement the hamming distance. Test it on the first two DNA samples above. You should obtain 42.*

Task 2 *Implement the Levenshtein distance. Test it on the example seen in class (python and kryptonite). Store in a matrix the distance between any two of the DNA samples above. The first line (and first column) of the matrix should be:* `[ 0. 31. 27. 31. 29. 29. 29. 38. 28. 27. 27. 36. 30. 31. 28. 28. 33. 32.]`. *Save this matrix in a text file.*

There is a package called Levenshtein already implemented in python. If you use your own computer, you may want to use it to check your results. Here are the links for this library:

- `https://pypi.org/project/python-Levenshtein/` package website
- `https://rawgit.com/ztane/python-Levenshtein/master/docs/Levenshtein.html` documentation

2 Sequence alignment

When you start coding known algorithm, that can take some time. Instead of hard-coding everything yourself, which can easily be time-consuming, the most efficient way is often to google the algorithm first. That is what we will do here.

!! Warning !! One must not do this without **checking the code first**. I shall insist on this point. If you do not read the code first (and understand at least very roughly what each part does), you may easily **get hacked**. Using a code that you don't understand is just like eating something without checking that it is edible first: anything can happen.

This warning may seem excessive when considering the fact that most of the code online is most often double checked by other users which would report a hack. However better be safe than sorry. Now let us go on.

One last point, look for **multiple codes** before chosing one. Use the simplest one. The longer / more complicated it is, the harder to decode and modify !

2.1 Local alignment

Smith-Waterman algorithm tries to find the best local alignment of two sequences.

Task 3 *Google a python3 implementation of the Smith-Waterman algorithm. Roughly check the code, then copy it in a file and make it work. This last part (making it work) is often what takes time because some adjustment may need to be done.*

Task 4 *Test your code on the example seen in class. You should obtain the same alignment. Then try it on some of the* `DNA_samples`*, see if it works.*

Task 5 *Modify the output so it looks like:*

```
A C - T A
| |   | |
A C A T A
```

2.2 (optional) Global alignment

The algorithm Needleman-Wunsch finds the best global alignment, it works much like Smith-Waterman algorithm. The only differences are:

	Smith–Waterman algorithm	Needleman–Wunsch algorithm
Initialization	First row and first column are set to 0	First row and first column are subject to gap penalty
Scoring	Negative score is set to 0	Score can be negative
Traceback	Begin with the highest score, end when 0 is encountered	Begin with the cell at the lower right of the matrix, end at top left cell

For more details, go check online (the wikipedia page is quite good).

Task 6 *Create a copy of the Smith-Waterman algorithm and modify it so that it does the Needleman-Wunsch algorithm. Try it on some examples.*