The Technion – Faculty of Biology
Computational Approaches for Life Scientists - 134141
Spring 2017

Home assignment 2 – Due May 2nd, 23:55

Submission instructions:

- Submission is **in pairs**.
  Honor code:
  Each student is expected to be fully involved in solving all the questions. Furthermore, although you may discuss the solutions with other students, each pair must write the whole solution separately. In particular, it is not allowed to spread any piece of code / solution.

- The answers will be submitted in 2 files:
  1. A doc/docx file with the "dry" part: answers to all the questions and the required explanations.
  2. A py file with the "wet" part: you need to write the required functions in the template file **hw1_template.py**. The code in this file must support your answers and conclusions, such that when run yields the results provided in the dry part.

     Important: whenever we give execution examples, your functions must behave identically. Test your code on these (as well as additional) inputs, and verify that the output is exactly the same. This part will be checks both by executing your code on various outputs, and manually to observe how you solved the problem.

- Note: the files should be named hwX_NAME1_NAME2 where X is the HW number (e.g. hw2) and NAME1 and NAME2 are the names of the students. For example: hw2_gur_hevroni_amir_rubinstein.docx. Only one student will submit the files in Moodle.

- The assignment should be submitted via moodle. The 2 files should be uploaded to moodle by the deadline. You may submit late though, up to a maximum of 5 days for the whole semester. For that matter, submission after 23:55 is a day late.

- Note: while each question normally has a single correct answer, the code you'll write to answer them may be written in various ways. There is no single correct code.

  You are encouraged to comment (#comment) your code.

- Bonus: up to 5 points may be given as a bonus to your grade, for interesting, **non-trivial** comments, mostly biological ones, that you find relevant. For example:
  - A relevant application for the question's topic that is beyond a trivial one
  - New information that sheds interesting light on the question, or puts it in an interesting biological context

  If you added such comments and would like them to be considered for bonus, add a clear title "INSIGHTS" before the comments. Also, make sure you state your references (a website, a course you took, a book, etc.).

ATG

Question 1 – longest common substring

a. In class we wrote the function longest_common_ss(s1, s2), which finds the longest common substring that is contained in both s1 and s2. We used a simple sequential search in which we probes increasing values of *k* (*k*=1,2,3,…). As we saw, running time may be long when the strings are long and they have a long common substring.
Improve this function to make it check less values of *k*: instead of checking *k*=1,2,3,… use a more efficient approach as hinted in class. Specifically, increase the values of *k* by multiples of 2 (*k*=1,2,4,8,16,…), until you reach a *k* for which there is no common substring. At this point you have *a=k*//2 for which there is a common substring, and *b=k* for which there isn't. Now use a binary search approach for *k* in the range between *a* and *b*.
Implement your function in the skeleton file, with the new name longest_common_ss_improved(s1, s2). Copy to the skeleton file any other functions from class that you want to use.
Note: Running time of about a minute or two for each of the executions in the next section is reasonable. Solutions that run in particularly short time may get a **bonus**.

b. Find the longest common substring of each pair of the genomes of Mycobacterium tuberculosis, Salmonella enterica, and Mycobacterium leprae. Specify, for each pair, what the length of the longest common substring is, the first 10 nucleotides in this longest common substring, and how much time did it take for your solution to run, in a table of the following form. Also, give a biological explanation for the results.

Genomes	Length of longest common substring	First 10 nucleotides in the longest common substring	Running time of your solution
M.tuberculosis + S.enterica			
M.tuberculosis + M.leprae			
M.leprae + S.enterica			

c. The function `common_substring_set` uses memory (a dictionary) to speed up the search, compared to the naïve solutions we saw. However, this does not come for free. Try running this function on two strings that have a very long common substring. For example, try the extreme case of two identical strings, and k=5000, such as:

```
>>> common_substring_set(Mycobacterium_leprae,
Mycobacterium_leprae, 5000)
```

What happens? Why? Suggest, in words, ways to overcome this limitation (no need to implement them).

Question 2 – regular expressions

a. The file cap2.txt in the site contains the protein Adenylyl cyclase-associated 2 (CAP2) of the organism Orangutan in FASTA format.
The CAP2 protein contains the following pattern: the first two positions are amino acids from L, I, V or M, then any amino acid, then R, then L, then one of D or E, the next 4 positions contain any amino acid, and at the end R, then L, and finally E.
Find the occurrence of this pattern and its position within CAP2, using regular expressions. State which regular expression you used, and copy into the docx file the commands in Python that you used.

b. Note: part of this section is meant to practice understanding of existing code written by others, and tailoring it for you needs.

In this question, our goal is to compute the DNA fragments that we will get if we let a given enzyme digest the genome of a given organism. Since a genome may be very large, we will not print those fragments, but rather store them in a file.

To that end, you will find in the skeleton file a function 'digest'. This function gets the following parameters:

- in_file: the input file where the required genome is stored
- out_file: the file in which we will store the fragments, each fragment in a separate line
- pattern: a regular expression string, representing the restriction site

Most of the 'digest' function is already implemented for you. You only have to complete a portion of the code where specified, to make it work correctly. Check you function on the test case provided below here, to make sure you implemented it correctly.

Test case:
Suppose we have a synthetic restriction enzyme produced in the lab, whose recognition site is AAA (3 Adenines), then 2 or 3 repetitions of the following pattern: A then T and then either T or C. The corresponding regular expression in Python's syntax is "AAA(AT[TC]){2,3}" . The enzyme cuts right after this pattern ends. For example, if dna = "CCCAAAATTATCCCAAAAATTATTATTCCC", then our enzyme would cut between the underlined nucleotides, and we will get the three fragments "CCCAAAATTATC", "CCAAAAATTATTATT", and "CCC".

Suppose the file `test_genome.txt` contains the DNA sequence from the above example:

```
CCCAAAATTATCCCAAAAATTATTATTCCC
```

then the following execution:

```
site = "(AAA(AT[TC]){2,3})"
in_file = "./test_genome.txt"
out_file = in_file[:-3] + "fragments.txt"
digest(in_file, out_file, site)
```

will create a new file named `test_genome.fragments.txt`, which has 3 rows:

```
CCCAAAATTATC
CCAAAAATTATTATT
CCC
```

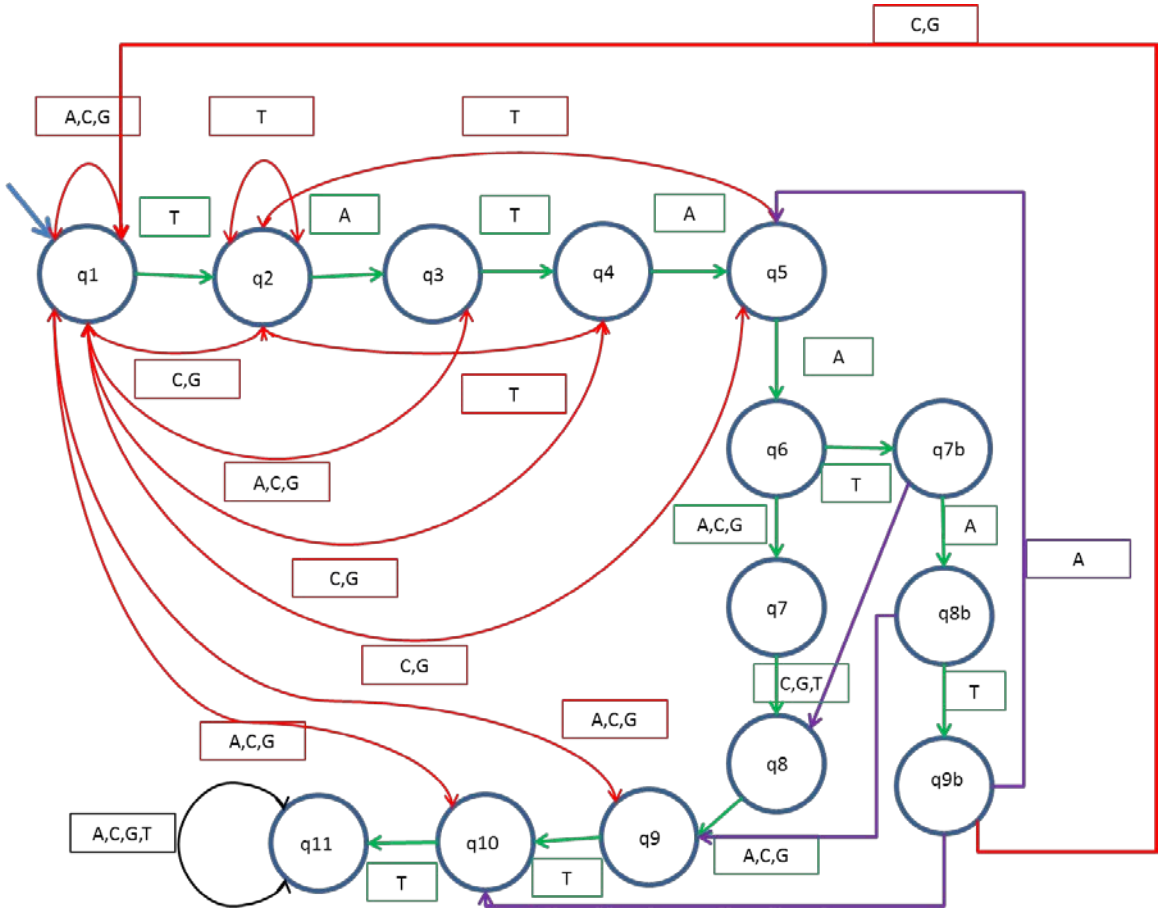
c. How many fragments will we get if we let the enzyme from the above test case digest the genome of *Salmonella enterica*? Answer this by using the function findall from the 're' library and the built-in len function. Do not try to print what findall returns – this will stuck IDLE because it is a list with potentially very long fragments! Copy the commands you used and the result into the docx file you submit.

Question 3 – finite automata

a. Draw a deterministic finite automaton[1] that accepts strings that contain any of the 6 DNA codons for Leucine. Use as few states as you can. Don't forget to specify the initial and accepting state(s).

b. Give a regular expression in Python's re syntax for the automaton from (a).

c. The following automaton[2] is supposed to recognize a TATA-like box (as it was defined in the lecture slides). The accepting state is q11. However, it contains several minor errors. Specify these errors. For each error, give an example for a DNA string for which this automaton would decide erroneously (that is, accept a string without a TATA-box or vice versa).



TAG

[1] Automaton is the single form of automata

[2] Sketched by a student in the previous offering of the course in 2013, as part of a HW assignment