Algorithms and Data Structures for Biology
13 May 2019 — Lab Session

Ugo Dal Lago Thomas Leventis

This assignment will be the second one to be marked. Please carefully read the instruction below before starting to work on the assignment

1 Instructions

This assignment must be completed individually, by May 27th 2019 at 23.59 CET. To complete the assignment, you need to send the following to the email addresses `ugo.dallago@unibo.it` and `thomas.leventis@irif.fr`:

- One Python file named `FIRSTNAME_LASTNAME.py`, with the code you have developed.
- One PDF file name `FIRSTNAME_LASTNAME.pdf`, preferably produced by way of LaTeX, including a description of the algorithms you designed, the Python code you wrote, and the experimental results.

The email should have the following subject: "`[ADSB] Assignment II`".

2 The Problem

We want to deal with the following problem. Suppose you are the head of just-launched genomics research lab, and you need to decide which ones (between many) software packages to buy. After some analysis, you conclude that the market offers n software packages, each of them with a price of P_i Euros, and offering some functionalities: unfortunately, not all packages are equivalent! We can model the functionalities your lab needs as a set $A = \{a_1, \dots, a_m\}$ where each a_i is distinct. As an example a_1 could be some form of genome sequencing, while a_2 could be a 3D molecule rendering. Each software package thus offers a set of functionalities $S_i \subseteq A$. As an example, we could have that

$$P_3 = 156 \qquad S_3 = \{a_1, a_3, a_4\}$$

meaning that the third package costs 156 Euros, and offers the functionalities a_1, a_3 and a_4. Your goal, of course, is to decide *which* software packages you want to buy, so that *all* functionalities are somehow covered by at least one of the software packages you buy. Moreover, you want to do that *minimizing* the price.

This assignment asks you to:

- model the problem described above as a combinatorial optimization problem, abstracting away from unessential details. In doing so, try to turn your problem into one the algorithmics literature is already aware of, by doing some online search.
- Think about exact, but also approximate greedy solutions to the problem. In doing so, you are encouraged to actively look for such solutions rather than developing them from scratch, since the literature on the subject is rich.
- Implement the exact and approximate greedy algorithms you designed in Python.

- Run your algorithm(s) on the problem instances you can find before the end of this week from the course's webpage. Use this opportunity to check that the approximatio ratio of your greedy solutions (if any) is within the bounds you expected.
- Finally use `cProfile` to experimentally test the runtime of your program on the instances above.