Programming Assignment 2: Algorithmic Warm-up

Revision: May 28, 2020

Introduction

Welcome to your second programming assignment of the Algorithmic Toolbox at Coursera! It consists of eight programming challenges. Three of them require you just to implement carefully the algorithms covered in the lectures. The remaining challenges will require you to first design an algorithm and then to implement it. For all the challenges, we provide starter solutions in `C++`, `Java`, and `Python3`. These solutions implement straightforward algorithms that usually work only for small inputs. To verify this, you may want to submit these solutions to the grader. This will usually give you a "`time limit exceeded`" message for `Python` starter files and either "`time limit exceeded`" or "`wrong answer`" message for `C++` and `Java` solutions (the reason for wrong answer being an integer overflow issue). Your goal is to replace a naive algorithm with an efficient one. In particular, you may want to use the naive implementation for stress testing your efficient implementation.

In this programming assignment, the grader will show you the input data if your solution fails on any of the tests. This is done to help you to get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail. However, for all the following programming assignments, the grader will show the input data only in case your solution fails on one of the first few tests.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. See the huge difference between a slow algorithm and a fast one.
2. Play with examples where knowing something interesting about a problem helps to design an algorithm that is much faster than a naive one.
3. Implement solutions that work much more faster than straightforward solutions for the following programming challenges:
   (a) compute a small Fibonacci number;
   (b) compute the last digit of a large Fibonacci number;
   (c) compute a huge Fibonacci number modulo m;
   (d) compute the last digit of a sum of Fibonacci numbers;
   (e) compute the last digit of a partial sum of Fibonacci numbers;
   (f) compute the greatest common divisor of two integers;
   (g) compute the least common multiple of two integers.
4. Implement the algorithms covered in the lectures, design new algorithms.
5. Practice implementing, testing, and debugging your solution. In particular, you will find out how in practice, when you implement an algorithm, you bump into unexpected questions and problems not covered by the general description of the algorithm. You will also check your understanding of the algorithm itself and most probably see that there are some aspects you did not think of before you had to actually implement it. You will overcome all those complexities, implement the algorithms, test them, debug, and submit to the system.