Module: Introduction (Week 2)
Course: Algorithmic Toolbox
Specialization: Data Structures and Algorithms

Programming Assignment 1: Introduction

Revision: February 17, 2016

Introduction

Welcome to your first programming assignment! It consists of five algorithmic problems. The first three problems require you just to implement carefully the algorithms covered in the lectures. The fourth one will require you to first design an algorithm and then to implement it. The last one is an advanced problem, but we provide a clear hint.

In this programming assignment, the grader will show you the input and output 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 (please review the questions 7.4 and 7.5 in the FAQ section for a more detailed explanation of this behavior of the grader).

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Implement solutions that work much more faster than straightforward solutions for the following computational problems:
   (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 greatest common divisor of two integers;
   (e) compute the least common multiple of two integers.
2. See the huge difference between a slow algorithm and a fast one.
3. Implement algorithms covered in the lectures, design new algorithms.
4. 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. Remember the advice (link) we gave you in the first module about testing your programs and feel free to return to those videos or parts of them again while working on your assignment. In the end of this document you will find also general recommendations on solving algorithmic problems.

Passing Criteria: 3 out of 5

Passing this programming assignment requires passing at least 3 out of 5 code problems from this assignment. In turn, passing a code problem requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Problem: Small Fibonacci Number

Problem Introduction

The Fibonacci numbers are defined as follows: $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

Problem Description

Task. Given an integer n, find the nth Fibonacci number F_n.

Input Format. The input consists of a single integer n.

Constraints. $0 \leq n \leq 45$.

Output Format. Output F_n.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
3
```

Output:

```
2
```

Sample 2.

Input:

```
10
```

Output:

```
55
```

Starter Files

The starter files for this problem contain an implementation of a naive recursive algorithm for computing Fibonacci numbers in `C++`, `Java`, and `Python`. Try compiling and running a starter solution on your machine. You will see that computing, say, F_{40} already takes noticeable time.

Another way to appreciate the dramatic difference between an exponential time algorithm and a polynomial time algorithm is to use the following visualization by David Galles: http://www.cs.usfca.edu/~galles/visualization/DPFib.html. Try computing F_{20} by a recursive algorithm by entering "20" and pressing the "Fibonacci Recursive" button. You will see an endless number of recursive calls. Now, press "Skip Forward" to stop the current algorithm and call the iterative algorithm by pressing "Fibonacci Table". This will compute F_{20} very quickly. (Note that the visualization uses a slightly different definition of Fibonacci numbers: $F_0 = 1$ instead of $F_0 = 0$. This of course has almost no influence on the running time.)

What To Do

Your goal is to replace a slow recursive algorithm by a fast iterative algorithm that can easily compute F_{45}. (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

2 Problem: The Last Digit of a Large Fibonacci Number

Problem Introduction

The Fibonacci numbers are defined as follows: $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

Problem Description

Task. Given an integer n, find the last digit of the nth Fibonacci number F_n (that is, F_n mod 10).

Input Format. The input consists of a single integer n.

Constraints. $0 \leq n \leq 10^7$.

Output Format. Output the last digit of F_n.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
3
```

Output:

```
2
```

Sample 2.

Input:

```
327305
```

Output:

```
5
```

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

What To Do

Recall that Fibonacci numbers grow exponentially fast. For example, the 200th Fibonacci number equals 280571172992510140037611932413038677189525. Therefore, a solution like

```
F[0]=0
F[1]=1
for i=2 to n:
  F[i]=F[i-1]+F[i-2]
print(F[n] mod 10)
```

will turn out to be too slow, because as i grows the ith iteration of the loop computes the sum of longer and longer numbers. Also, for example, F_{1000} does not fit into the standard C++ `int` type.

To overcome this difficulty, you may want to store in $F[i]$ not the ith Fibonacci number itself, but just its last digit (that is, F_i mod 10). Computing the last digit of F_i is easy: it is just the last digit of the sum of the last digits of F_{i-1} and F_{i-2}:

```
F[i]=(F[i-1]+F[i-2]) mod 10
```

This way, all $F[i]$'s are just digits, so they fit perfectly into any standard integer type, and computing a sum of $F[i-1]$ and $F[i-2]$ is performed very quickly.

3 Problem: Greatest Common Divisor

Problem Introduction

The greatest common divisor GCD(a, b) of two non-negative integers a and b (which are not both equal to 0) is the greatest integer d that divides both a and b.

Problem Description

Task. Given two integers a and b, find their greatest common divisor.

Input Format. The two integers a, b are given in the same line separated by space.

Constraints. $1 \le a, b \le 2 \cdot 10^9$.

Output Format. Output GCD(a, b).

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
18 35
```

Output:

```
1
```

Sample 2.

Input:

```
28851538 1183019
```

Output:

```
17657
```

Starter Files

The starter files contain an implementation of a naive algorithm that just goes through all potential divisors and selects the largest one. You may want to try to submit a starter solution to the grading system to ensure that it is too slow.

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

4 Problem: Least Common Multiple

Problem Introduction

The least common multiple of two positive integers a and b is the least positive integer m that is divisible by both a and b.

Problem Description

Task. Given two integers a and b, find their least common multiple.

Input Format. The two integers a and b are given in the same line separated by space.

Constraints. $1 \le a, b \le 2 \cdot 10^9$.

Output Format. Output the least common multiple of a and b.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
6 8
```

Output:

```
24
```

Sample 2.

Input:

```
28851538 1183019
```

Output:

```
1933053046
```

Starter Files

The starter files contain an implementation that just returns the product of a and b. This is of course a common multiple of a and b, but not always the smallest one.

What To Do

Play with small datasets to find out how the least common multiple $\text{LCM}(a, b)$ is related to the greatest common divisor $\text{GCD}(a, b)$. Do you see? For any two positive integers a and b, $\text{LCM}(a, b) \cdot \text{GCD}(a, b) = a \cdot b$. Prove this property and implement an algorithm for computing the least common multiple using your solution for the greatest common divisor.

5 Advanced Problem: Huge Fibonacci Number modulo m

(Recall that advanced problems are not covered in the video lectures and require additional ideas to be solved. We therefore strongly recommend you start solving these problems only when you are done with the basic problems.)

Problem Introduction

The Fibonacci numbers are defined as follows: $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

Problem Description

Task. Given two integers n and m, output $F_n \bmod m$ (that is, the remainder of F_n when divided by m).

Input Format. The input consists of two integers n and m given on the same line (separated by a space).

Constraints. $1 \leq n \leq 10^{18}$, $2 \leq m \leq 10^5$.

Output Format. Output $F_n \bmod m$.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 1.5 sec, `Python`: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
281621358815590 30524
```

Output:

```
11963
```

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure (if you are using `C++`, `Java`, or `Python`; for `C`, you need to implement a solution from scratch).

What To Do

In this problem, the given number n may be really huge. Hence an algorithm looping for n iterations will not fit into one second for sure. Therefore we need to avoid such a loop.

To get an idea how to solve this problem without going through all F_i for i from 0 to n, let's see what happens when m is small — say, $m = 2$ or $m = 3$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_i	0	1	1	2	3	5	8	13	21	35	55	89	144	233	377	610
$F_i \bmod 2$	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
$F_i \bmod 3$	0	1	1	2	0	2	2	1	0	1	1	2	0	2	2	1

Take a detailed look at this table. Do you see? Both these sequences are periodic! For $m = 2$, the period is 011 and has length 3, while for $m = 3$ the period is 01120221 and has length 8. Therefore, to compute, say, $F_{2015} \bmod 3$ we just need to find the remainder of 2015 when divided by 8. Since $2015 = 251 \cdot 8 + 7$, we conclude that $F_{2015} \bmod 3 = F_7 \bmod 3 = 1$.

This is true in general: for any integer $m \geq 2$, the sequence $F_n \bmod m$ is periodic. The period always starts with 01 and is known as Pisano period.

6 General Instructions and Recommendations on Solving Algorithmic Problems

Your main goal in an algorithmic problem is to implement a program that solves a given computational problem in just few seconds even on massive datasets. Your program should read a dataset from the standard input and write an answer to the standard output.

Below we provide general instructions and recommendations on solving such problems. Before reading them, go through readings and screencasts in the first module that show a step by step process of solving two algorithmic problems: link.

6.1 Reading the Problem Statement

You start by reading the problem statement that contains the description of a particular computational task as well as time and memory limits your solution should fit in, and one or two sample tests. In some problems your goal is just to implement carefully an algorithm covered in the lectures, while in some other problems you first need to come up with an algorithm yourself.

6.2 Designing an Algorithm

If your goal is to design an algorithm yourself, one of the things it is important to realize is the expected running time of your algorithm. Usually, you can guess it from the problem statement (specifically, from the subsection called constraints) as follows . Modern computers perform roughly 10^8–10^9 operations per second. So, if the maximum size of a dataset in the problem description is $n = 10^5$, then most probably an algorithm with quadratic running time is not going to fit into time limit (since for $n = 10^5$, $n^2 = 10^{10}$) while a solution with running time $O(n \log n)$ will fit. However, an $O(n^2)$ solution will fit if n is up to $10^3 = 1000$, and if n is at most 100, even $O(n^3)$ solutions will fit. In some cases, the problem is so hard that we do not know a polynomial solution. But for n up to 18, a solution with $O(2^n n^2)$ running time will probably fit into the time limit.

To design an algorithm with the expected running time, you will of course need to use the ideas covered in the lectures. Also, make sure to carefully go through sample tests in the problem description.

6.3 Implementing Your Algorithm

When you have an algorithm in mind, you start implementing it. Currently, you can use the following programming languages to implement a solution to a problem: `C`, `C++`, `Java`, `Python2`, `Python3`. For all problems, we will be providing starter solutions for `C++`, `Java`, and `Python3`. If you are going to use one of these programming languages, use these starter files. If you are going to use `C` or `Python2`, you need to implement a solution from scratch.

6.4 Compiling Your Program

For solving programming assignments, you can use any of the following programming languages: `C`, `C++`, `Java`, `Python2`, or `Python3`. However, we will only be providing starter solution files for `C++`, `Java`, and `Python3`. The programming language of your submission is detected automatically, based on the extension of your submission.

Your solution will be compiled as follows. We recommend that when testing your solution locally, you use the same compiler flags for compiling. This will increase the chances that your program behaves in the same way on your machine and on the testing machine (note that a buggy program may behave differently when compiled by different compilers, or even by the same compiler with different flags).

- `C` (`gcc 5.2.1`). File extensions: `.c`. Flags:

```
gcc -pipe -O2 -std=c11
```

- `C++` (`g++ 5.2.1`). File extensions: `.cc`, `.cpp`. Flags:

```
g++ -pipe -O2 -std=c++11
```

  If your `C/C++` compiler does not recognize `-std=c++11` flag, try replacing it with `-std=c++0x` flag or compiling without this flag at all (all starter solutions can be compiled without it). On Linux and MacOS, you most probably have the required compiler. On Windows, you may use your favorite compiler or install, e.g., `cygwin`.

- `Java (Open JDK 7)`. File extensions: `.java`. No flags:

```
javac
```

- `Python 2 (CPython 2.7)`. File extensions: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python2"). No flags:

```
python2
```

- `Python 3 (CPython 3.4)`. File extensions: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python3"). No flags:

```
python3
```

6.5 Testing Your Program

When your program is ready, you start testing it. It makes sense to start with small datasets — for example, sample tests provided in the problem description. Ensure that your program produces a correct result.

You then proceed to checking how long does it take your program to process a massive dataset. For this, it makes sense to implement your algorithm as a function like `solve(dataset)` and then implement an additional procedure `generate()` that produces a large dataset. For example, if an input to a problem is a sequence of integers of length $1 \le n \le 10^5$, then generate a sequence of length exactly 10^5, pass it to your `solve()` function, and ensure that the program outputs the result quickly.

Also, check the boundary values. Ensure that your program processes correctly sequences of size $n = 1, 2, 10^5$. If a sequence of integers from 0 to, say, 10^6 is given as an input, check how your program behaves when it is given a sequence $0, 0, \ldots, 0$ or a sequence $10^6, 10^6, \ldots, 10^6$. Check also on randomly generated data. For each such test check that you program produces a correct result (or at least a reasonably looking result).

In the end, we encourage you to stress test your program to make sure it passes in the system at the first attempt. See the readings and screencasts from the first week to learn about testing and stress testing: link.

6.6 Submitting Your Program to the Grading System

When you are done with testing, you submit your program to the grading system. For this, you go the submission page, create a new submission, and upload a file with your program. The grading system then compiles your program (detecting the programming language based on your file extension, see Subsection 6.4) and runs it on a set of carefully constructed tests to check that your program always outputs a correct result and that it always fits into the given time and memory limits. The grading usually takes no more than a minute, but in rare cases when the servers are overloaded it might take longer. Please be patient. You can safely leave the page when your solution is uploaded.

As a result, you get a feedback message from the grading system. The feedback message that you will love to see is: `Good job!` This means that your program has passed all the tests. On the other hand, the three messages `Wrong answer`, `Time limit exceeded`, `Memory limit exceeded` notify you that your program failed due to one these three reasons. Note that the grader will not show you the actual test you program have failed on (though it does show you the test if your program have failed on one of the first few tests; this is done to help you to get the input/output format right).

6.7 Debugging and Stress Testing Your Program

If your program failed, you will need to debug it. Most probably, you didn't follow some of our suggestions from the section 6.5. See the readings and screencasts from the first week to learn about debugging your program: link.

You are almost guaranteed to find a bug in your program using stress testing, because the way these programming assignments and tests for them are prepared follows the same process: small manual tests, tests for edge cases, tests for large numbers and integer overflow, big tests for time limit and memory limit checking, random test generation. Also, implementation of wrong solutions which we expect to see and stress testing against them to add tests specifically against those wrong solutions.

Go ahead, and we hope you pass the assignment soon!

7 Frequently Asked Questions

7.1 I submit the program, but nothing happens. Why?

You need to create submission and upload the file with your solution in one of the programming languages `C`, `C++`, `Java`, or `Python` (see Subsections 6.3 and 6.4). Make sure that after uploading the file with your solution you press on the blue "Submit" button in the bottom. After that, the grading starts, and the submission being graded is enclosed in an orange rectangle. After the testing is finished, the rectangle disappears, and the results of the testing of all problems is shown to you.

7.2 I submit the solution only for one problem, but all the problems in the assignment are graded. Why?

Each time you submit any solution, the last uploaded solution for each problem is tested. Don't worry: this doesn't affect your score even if the submissions for the other problems are wrong. As soon as you pass the sufficient number of problems in the assignment (see in the pdf with instructions), you pass the assignment. After that, you can improve your result if you successfully pass more problems from the assignment. We recommend working on one problem at a time, checking whether your solution for any given problem passes in the system as soon as you are confident in it. However, it is better to test it first, please refer to the reading about stress testing: link.

7.3 What are the possible grading outcomes, and how to read them?

Your solution may either pass or not. To pass, it must work without crashing and return the correct answers on all the test cases we prepared for you, and do so under the time limit and memory limit constraints specified in the problem statement. If your solution passes, you get the corresponding feedback "Good job!" and get a point for the problem. If your solution fails, it can be because it crashes, returns wrong answer, works for too long or uses too much memory for some test case. The feedback will contain the number of the test case on which your solution fails and the total number of test cases in the system. The tests for the problem are numbered from 1 to the total number of test cases for the problem, and the program is always tested on all the tests in the order from the test number 1 to the test with the biggest number.

Here are the possible outcomes:

`Good job!` Hurrah! Your solution passed, and you get a point!

`Wrong answer.` Your solution has output incorrect answer for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data, the output of your program and the correct answer. Otherwise, you won't know the input, the output, and the correct answer. Check that you consider all the cases correctly, avoid integer overflow, output the required white space, output the floating point numbers with the required precision, don't output anything in addition to what you are asked to output in the output specification of the problem statement. See this reading on testing: link.

`Time limit exceeded.` Your solution worked longer than the allowed time limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data, the output of your program and the correct answer. Otherwise, you won't know the input, the output and the correct answer. Check again that your algorithm has good enough running time estimate. Test your program locally on the test of maximum size allowed by the problem statement and see how long it works. Check that your program doesn't wait for some input from the user which makes it to wait forever. See this reading on testing: link.

`Memory limit exceeded.` Your solution used more than the allowed memory limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1,

you will also see the input data, the output of your program and the correct answer. Otherwise, you won't know the input, the output and the correct answer. Estimate the amount of memory that your program is going to use in the worst case and check that it is less than the memory limit. Check that you don't create too large arrays or data structures. Check that you don't create large arrays or lists or vectors consisting of empty arrays or empty strings, since those in some cases still eat up memory. Test your program locally on the test of maximum size allowed by the problem statement and look at its memory consumption in the system.

`Cannot check answer. Perhaps output format is wrong.` This happens when you output something completely different than expected. For example, you are required to output word "Yes" or "No", but you output number 1 or 0, or vice versa. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (this is not allowed, so please follow exactly the output format specified in the problem statement). Maybe your program doesn't output anything, because it crashes.

`Unknown signal 6 (or 7, or 8, or 11, or some other).` This happens when your program crashes. It can be because of division by zero, accessing memory outside of the array bounds, using uninitialized variables, too deep recursion that triggers stack overflow, sorting with contradictory comparator, removing elements from an empty data structure, trying to allocate too much memory, and many other reasons. Look at your code and think about all those possibilities. Make sure that you use the same compilers and the same compiler options as we do. Try different testing techniques from this reading: link.

`Grading failed.` Something very wrong happened with the system. Contact Coursera for help or write in the forums to let us know.

7.4 How to understand why my program fails and to fix it?

If your program works incorrectly, it gets a feedback from the grader. For the Programming Assignment 1, when your solution fails, you will see the input data, the correct answer and the output of your program in case it didn't crash, finished under the time limit and memory limit constraints. If the program crashed, worked too long or used too much memory, the system stops it, so you won't see the output of your program or will see just part of the whole output. We show you all this information so that you get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail.

However, in the following Programming Assignments throughout the Specialization you will only get so much information for the test cases from the problem statement. For the next tests you will only get the result: passed, time limit exceeded, memory limit exceeded, wrong answer, wrong output format or some form of crash. We hide the test cases, because it is crucial for you to learn to test and fix your program even without knowing exactly the test on which it fails. In the real life, often there will be no or only partial information about the failure of your program or service. You will need to find the failing test case yourself. Stress testing is one powerful technique that allows you to do that. You should apply it after using the other testing techniques covered in this reading.

7.5 Why do you hide the test on which my program fails?

Often beginner programmers think by default that their programs work. Experienced programmers know, however, that their programs almost never work initially. Everyone who wants to become a better programmer needs to go through this realization.

When you are sure that your program works by default, you just throw a few random test cases against it, and if the answers look reasonable, you consider your work done. However, mostly this is not enough. To make one's programs work, one must test them really well. Sometimes, the programs still don't work although you tried really hard to test them, and you need to be both skilled and creative to fix your bugs. Solutions

to algorithmic problems are one of the hardest to implement correctly. That's why in this Specialization you will gain this important experience which will be invaluable in the future when you write programs which you really need to get right.

It is crucial for you to learn to test and fix your programs yourself. In the real life, often there will be no or only partial information about the failure of your program or service. Still, you will have to reproduce the failure to fix it (or just guess what it is, but that's rare, and you will still need to reproduce the failure to make sure you have really fixed it). When you solve algorithmic problems, it is very frequent to make subtle mistakes. That's why you should apply the testing techniques described in this reading to find the failing test case and fix your program.

7.6 My solution does not pass the tests? May I post it in the forum and ask for a help?

No, please do not post any solutions in the forum or anywhere on the web, even if a solution does not pass the tests (as in this case you are still revealing parts of a correct solution). Recall the third item of the Coursera Honor Code: "I will not make solutions to homework, quizzes, exams, projects, and other assignments available to anyone else (except to the extent an assignment explicitly permits sharing solutions). This includes both solutions written by me, as well as any solutions provided by the course staff or others" (link).