Introduction 1

This chapter shows what competitive programming is about, outlines the contents of the book, and discusses additional learning resources.

Section 1.1 goes through the elements of competitive programming, introduces a selection of popular programming contests, and gives advice on how to practice competitive programming.

Section 1.2 discusses the goals and topics of this book, and briefly describes the contents of each chapter.

Section 1.3 presents the CSES Problem Set, which contains a collection of practice problems. Solving the problems while reading the book is a good way to learn competitive programming.

Section 1.4 discusses other books related to competitive programming and the design of algorithms.

1.1 What is Competitive Programming?

Competitive programming combines two topics: the design of algorithms and the implementation of algorithms.

Design of Algorithms The core of competitive programming is about inventing efficient algorithms that solve well-defined computational problems. The design of algorithms requires problem solving and mathematical skills. Often a solution to a problem is a combination of well-known methods and new insights.

Mathematics plays an important role in competitive programming. Actually, there are no clear boundaries between algorithm design and mathematics. This book has been written so that not much background in mathematics is needed. The appendix of the book reviews some mathematical concepts that are used throughout the book,

A. Laaksonen, *Guide to Competitive Programming*, Undergraduate
Topics in Computer Science, https://doi.org/10.1007/978-3-319-72547-5_1

such as sets, logic, and functions, and the appendix can be used as a reference when reading the book.

Implementation of Algorithms In competitive programming, the solutions to problems are evaluated by testing an implemented algorithm using a set of test cases. Thus, after coming up with an algorithm that solves the problem, the next step is to correctly implement it, which requires good programming skills. Competitive programming greatly differs from traditional software engineering: programs are short (usually at most some hundreds of lines), they should be written quickly, and it is not needed to maintain them after the contest.

At the moment, the most popular programming languages used in contests are C++, Python, and Java. For example, in Google Code Jam 2017, among the best 3,000 participants, 79% used C++, 16% used Python, and 8% used Java. Many people regard C++ as the best choice for a competitive programmer. The benefits of using C++ are that it is a very efficient language and its standard library contains a large collection of data structures and algorithms.

All example programs in this book are written in C++, and the standard library's data structures and algorithms are often used. The programs follow the C++11 standard, which can be used in most contests nowadays. If you cannot program in C++ yet, now is a good time to start learning.

1.1.1 Programming Contests

IOI The *International Olympiad in Informatics* is an annual programming contest for secondary school students. Each country is allowed to send a team of four students to the contest. There are usually about 300 participants from 80 countries.

The IOI consists of two five-hour long contests. In both contests, the participants are asked to solve three difficult programming tasks. The tasks are divided into subtasks, each of which has an assigned score. While the contestants are divided into teams, they compete as individuals.

Participants for the IOI are selected through national contests. Before the IOI, many regional contests are organized, such as the Baltic Olympiad in Informatics (BOI), the Central European Olympiad in Informatics (CEOI), and the Asia-Pacific Informatics Olympiad (APIO).

ICPC The *International Collegiate Programming Contest* is an annual programming contest for university students. Each team in the contest consists of three students, and unlike in the IOI, the students work together; there is only one computer available for each team.

The ICPC consists of several stages, and finally the best teams are invited to the World Finals. While there are tens of thousands of participants in the contest, there are only a small number[1] of final slots available, so even advancing to the finals is a great achievement.

[1] The exact number of final slots varies from year to year; in 2017, there were 133 final slots.

In each ICPC contest, the teams have five hours of time to solve about ten algorithm problems. A solution to a problem is accepted only if it solves all test cases efficiently. During the contest, competitors may view the results of other teams, but for the last hour the scoreboard is frozen and it is not possible to see the results of the last submissions.

Online Contests There are also many online contests that are open for everybody. At the moment, the most active contest site is Codeforces, which organizes contests about weekly. Other popular contest sites include AtCoder, CodeChef, CS Academy, HackerRank, and Topcoder.

Some companies organize online contests with onsite finals. Examples of such contests are Facebook Hacker Cup, Google Code Jam, and Yandex.Algorithm. Of course, companies also use those contests for recruiting: performing well in a contest is a good way to prove one's skills in programming.

1.1.2 Tips for Practicing

Learning competitive programming requires a great amount of work. However, there are many ways to practice, and some of them are better than others.

When solving problems, one should keep in mind that the *number* of solved problems is not so important that the *quality* of the problems. It is tempting to select problems that look nice and easy and solve them, and skip problems that look hard and tedious. However, the way to really improve one's skills is to focus on the latter type of problems.

Another important observation is that most programming contest problems can be solved using simple and short algorithms, but the difficult part is to invent the algorithm. Competitive programming is not about learning complex and obscure algorithms by heart, but rather about learning problem solving and ways to approach difficult problems using simple tools.

Finally, some people despise the implementation of algorithms: it is fun to design algorithms but boring to implement them. However, the ability to quickly and correctly implement algorithms is an important asset, and this skill can be practiced. It is a bad idea to spend most of the contest time for writing code and finding bugs, instead of thinking of how to solve problems.

1.2 About This Book

The *IOI Syllabus* [15] regulates the topics that may appear at the International Olympiad in Informatics, and the syllabus has been a starting point when selecting topics for this book. However, the book also discusses some advanced topics that are (as of 2017) excluded from the IOI but may appear in other contests. Examples of such topics are maximum flows, nim theory, and suffix arrays.

While many competitive programming topics are discussed in standard algorithms textbooks, there are also differences. For example, many textbooks focus on implementing sorting algorithms and fundamental data structures from scratch, but this knowledge is not very relevant in competitive programming, because standard library functionality can be used. Then, there are topics that are well known in the competitive programming community but rarely discussed in textbooks. An example of such a topic is the segment tree data structure that can be used to solve a large number of problems that would otherwise require tricky algorithms.

One of the purposes of this book has been to *document* competitive programming techniques that are usually only discussed in online forums and blog posts. Whenever possible, scientific references have been given for methods that are specific to competitive programming. However, this has not often been possible, because many techniques are now part of competitive programming *folklore* and nobody knows who has originally discovered them.

The structure of the book is as follows:

- Chapter 2 reviews features of the C++ programming language, and then discusses recursive algorithms and bit manipulation.
- Chapter 3 focuses on efficiency: how to create algorithms that can quickly process large data sets.
- Chapter 4 discusses sorting algorithms and binary search, focusing on their applications in algorithm design.
- Chapter 5 goes through a selection of data structures of the C++ standard library, such as vectors, sets, and maps.
- Chapter 6 introduces an algorithm design technique called dynamic programming, and presents examples of problems that can be solved using it.
- Chapter 7 discusses elementary graph algorithms, such as finding shortest paths and minimum spanning trees.
- Chapter 8 deals with some advanced algorithm design topics, such as bit-parallelism and amortized analysis.
- Chapter 9 focuses on efficiently processing array range queries, such as calculating sums of values and determining minimum values.
- Chapter 10 presents specialized algorithms for trees, including methods for processing tree queries.
- Chapter 11 discusses mathematical topics that are relevant in competitive programming.
- Chapter 12 presents advanced graph techniques, such as strongly connected components and maximum flows.
- Chapter 13 focuses on geometric algorithms and presents techniques using which geometric problems can be solved conveniently.
- Chapter 14 deals with string techniques, such as string hashing, the Z-algorithm, and using suffix arrays.
- Chapter 15 discusses a selection of more advanced topics, such as square root algorithms and dynamic programming optimization.

1.3 CSES Problem Set

The *CSES Problem Set* provides a collection of problems that can be used to practice competitive programming. The problems have been arranged in order of difficulty, and all techniques needed for solving the problems are discussed in this book. The problem set is available at the following address:

https://cses.fi/problemset/

Let us see how to solve the first problem in the problem set, called *Weird Algorithm*. The problem statement is as follows:

> Consider an algorithm that takes as input a positive integer n. If n is even, the algorithm divides it by two, and if n is odd, the algorithm multiplies it by three and adds one. The algorithm repeats this, until n is one. For example, the sequence for $n = 3$ is as follows:
>
> $$3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$
>
> Your task is to simulate the execution of the algorithm for a given value of n.
>
> **Input**
>
> The only input line contains an integer n.
>
> **Output**
>
> Print a line that contains all values of n during the algorithm.
>
> **Constraints**
>
> - $1 \leq n \leq 10^6$
>
> **Example**
>
> Input:
>
> ```
> 3
> ```
>
> Output:
>
> ```
> 3 10 5 16 8 4 2 1
> ```

This problem is connected to the famous *Collatz conjecture* which states that the above algorithm terminates for every value of n. However, nobody has been able to prove it. In this problem, however, we know that the initial value of n will be at most one million, which makes the problem much easier to solve.

This problem is a simple simulation problem, which does not require much thinking. Here is a possible way to solve the problem in C++:

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    while (true) {
        cout << n << " ";
        if (n == 1) break;
        if (n%2 == 0) n /= 2;
        else n = n*3+1;
    }
    cout << "\n";
}
```

The code first reads in the input number n, and then simulates the algorithm and prints the value of n after each step. It is easy to test that the algorithm correctly handles the example case $n = 3$ given in the problem statement.

Now is time to *submit* the code to CSES. Then the code will be compiled and tested using a set of test cases. For each test case, CSES will tell us whether our code passed it or not, and we can also examine the input, the expected output, and the output produced by our code.

After testing our code, CSES gives the following report:

test	verdict	time (s)
#1	ACCEPTED	0.06 / 1.00
#2	ACCEPTED	0.06 / 1.00
#3	ACCEPTED	0.07 / 1.00
#4	ACCEPTED	0.06 / 1.00
#5	ACCEPTED	0.06 / 1.00
#6	TIME LIMIT EXCEEDED	– / 1.00
#7	TIME LIMIT EXCEEDED	– / 1.00
#8	WRONG ANSWER	0.07 / 1.00
#9	TIME LIMIT EXCEEDED	– / 1.00
#10	ACCEPTED	0.06 / 1.00

This means that our code passed some of the test cases (ACCEPTED), was sometimes too slow (TIME LIMIT EXCEEDED), and also produced an incorrect output (WRONG ANSWER). This is quite surprising!

The first test case that fails has $n = 138367$. If we test our code locally using this input, it turns out that the code is indeed slow. In fact, it never terminates.

The reason why our code fails is that n can become quite large during the simulation. In particular, it can become larger than the upper limit of an `int` variable. To

fix the problem, it suffices to change our code so that the type of n is `long long`. Then we will get the desired result:

test	verdict	time (s)
#1	ACCEPTED	0.05 / 1.00
#2	ACCEPTED	0.06 / 1.00
#3	ACCEPTED	0.07 / 1.00
#4	ACCEPTED	0.06 / 1.00
#5	ACCEPTED	0.06 / 1.00
#6	ACCEPTED	0.05 / 1.00
#7	ACCEPTED	0.06 / 1.00
#8	ACCEPTED	0.05 / 1.00
#9	ACCEPTED	0.07 / 1.00
#10	ACCEPTED	0.06 / 1.00

As this example shows, even very simple algorithms may contain subtle bugs. Competitive programming teaches how to write algorithms that really work.

1.4 Other Resources

Besides this book, there are already several other books on competitive programming. Skiena's and Revilla's *Programming Challenges* [28] is a pioneering book in the field published in 2003. A more recent book is *Competitive Programming 3* [14] by Halim and Halim. Both the above books are intended for readers with no background in competitive programming.

Looking for a Challenge? [7] is an advanced book, which present a collection of difficult problems from Polish programming contests. The most interesting feature of the book is that it provides detailed analyses of how to solve the problems. The book is intended for experienced competitive programmers.

Of course, general algorithms books are also good reads for competitive programmers. The most comprehensive of them is *Introduction to Algorithms* [6] written by Cormen, Leiserson, Rivest, and Stein, also called the *CLRS*. This book is a good resource if you want to check all details concerning an algorithm and how to rigorously prove that it is correct.

Kleinberg's and Tardos's *Algorithm Design* [19] focuses on algorithm design techniques, and thoroughly discusses the divide and conquer method, greedy algorithms, dynamic programming, and maximum flow algorithms. Skiena's *The Algorithm Design Manual* [27] is a more practical book which includes a large catalogue of computational problems and describes ways how to solve them.