1 Introduction

> You, my young friend, are going to learn to program the algorithms of this book, and then go on to win programming contests, sparkle during your job interviews, and finally roll up your sleeves, get to work, and greatly improve the gross national product!

Mistakenly, computer scientists are still often considered the magicians of modern times. Computers have slowly crept into our businesses, our homes and our machines, and have become important enablers in the functioning of our world. However, there are many that use these devices without really mastering them, and hence, they do not fully enjoy their benefits. Knowing how to program provides the ability to fully exploit their potential to solve problems in an efficient manner. Algorithms and programming techniques have become a necessary background for many professions. Their mastery allows the development of creative and efficient computer-based solutions to problems encountered every day.

This text presents a variety of algorithmic techniques to solve a number of classic problems. It describes practical situations where these problems arise, and presents simple implementations written in the programming language Python. Correctly implementing an algorithm is not always easy: there are numerous traps to avoid and techniques to apply to guarantee the announced running times. The examples in the text are embellished with explanations of important implementation details which must be respected.

For the last several decades, programming competitions have sprung up at every level all over the world, in order to promote a broad culture of algorithms. The problems proposed in these contests are often variants of classic algorithmic problems, presented as frustrating enigmas that will never let you give up until you solve them!

1.1 Programming Competitions

In a programming competition, the candidates must solve several problems in a fixed time. The problems are often variants of classic problems, such as those addressed in this book, dressed up with a short story. The inputs to the problems are called *instances*. An instance can be, for example, the adjacency matrix of a graph for a shortest path problem. In general, a small example of an instance is provided, along with its solution. The source code of a solution can be uploaded to a server via

a web interface, where it is compiled and tested against instances hidden from the public. For some problems the code is called for each instance, whereas for others the input begins with an integer indicating the number of instances occurring in the input. In the latter case, the program must then loop over each instance, solve it and display the results. A submission is accepted if it gives correct results in a limited time, on the order of a few seconds.

Figure 1.1 The logo of the ICPC nicely shows the steps in the resolution of a problem. A helium balloon is presented to the team for each problem solved.

To give here a list of all the programming competitions and training sites is quite impossible, and such a list would quickly become obsolete. Nevertheless, we will review some of the most important ones.

ICPC The oldest of these competitions was founded by the *Association for Computing Machinery* in 1977 and supported by them up until 2017. This contest, known as the ICPC, for *International Collegiate Programming Contest*, is organised in the form of a tournament. The starting point is one of the regional competitions, such as the *South-West European Regional Contest* (SWERC), where the two best teams qualify for the worldwide final. The particularity of this contest is that each three-person team has only a single computer at their disposal. They have only 5 hours to solve a maximum number of problems among the 10 proposed. The first ranking criterion is the number of submitted solutions accepted (i.e. tested successfully against a set of unknown instances). The next criterion is the sum over the submitted problems of the time between the start of the contest and the moment of the accepted submission. For each erroneous submission, a penalty of 20 minutes is added.

There are several competing theories on what the ideal composition of a team is. In general, a good programmer and someone familiar with algorithms is required, along with a specialist in different domains such as graph theory, dynamic programming, etc. And, of course, the team members need to get along together, even in stressful situations!

For the contest, each team can bring 25 pages of reference code printed in an 8-point font. They can also access the online documentation of the Java API and the C++ standard library.

Google Code Jam In contrast with the ICPC contest, which is limited to students up to a Master's level, the Google Code Jam is open to everyone. This more recent annual competition is for individual contestants. Each problem comes in

general with a deck of small instances whose resolution wins a few points, and a set of enormous instances for which it is truly important to find a solution with the appropriate algorithmic complexity. The contestants are informed of the acceptance of their solution for the large instances only at the end of the contest. However, its true strong point is the possibility to access the solutions submitted by all of the participants, which is extremely instructive.

The competition *Facebook Hacker Cup* is of a similar nature.

Prologin The French association Prologin organises each year a competition targeted at students up to twenty years old. Their capability to solve algorithmic problems is put to test in three stages: an online selection, then regional competitions and concluding with a national final. The final is atypically an endurance test of 36 hours, during which the participants are confronted with a problem in Artificial Intelligence. Each candidate must program a "champion" to play a game whose rules are defined by the organisers. At the end of the day, the champions are thrown in the ring against each other in a tournament to determine the final winner.

The website `https://prologin.org` includes complete archives of past problems, with the ability to submit algorithms online to test the solutions.

France-IOI Each year, the organisation France-IOI prepares junior and senior high school students for the International Olympiad in Informatics. Since 2011, they have organised the 'Castor Informatique' competition, addressed at students from Grade 4 to Grade 12 (675,000 participants in 2018). Their website `http://france-ioi.org` hosts a large number of algorithmic problems (more than 1,000).

Numerous programming contests organised with the goal of selecting candidates for job offers also exist. The web site `www.topcoder.com`, for example, also includes tutorials on algorithms, often very well written.

For training, we particularly recommend `https://codeforces.com`, a well-respected web site in the community of programming competitions, as it proposes clear and well-prepared problems.

1.1.1 Training Sites

A number of websites propose problems taken from the annals of competitions, with the possibility to test solutions as a training exercise. This is notably the case for Google Code Jam and Prologin (in French). The collections of the annals of the ICPC contests can be found in various locations.

Traditional online judges The following sites contain, among others, many problems derived from the ICPC contests.

`uva` `http://uva.onlinejudge.org`
`icpcarchive` `http://icpcarchive.ecs.baylor.edu`, `http://livearchive.onlinejudge.org`

Chinese online judges Several training sites now exist in China. They tend to have a purer and more refined interface than the traditional judges. Nevertheless, sporadic failures have been observed.

poj `http://poj.org`
tju `http://acm.tju.edu.cn` (Shut down since 2017)
zju `http://acm.zju.edu.cn`

Modern online judges Sphere Online Judge `http://spoj.com` and Kattis `http://open.kattis.com` have the advantage of accepting the submission of solutions in a variety of languages, including Python.

spoj `http://spoj.com`
kattis `http://open.kattis.com`
zju `http://acm.zju.edu.cn`

Other sites

codechef `http://codechef.com`
codility `http://codility.com`
gcj `http://code.google.com/codejam`
prologin `http://prologin.org`
slpc `http://cs.stanford.edu/group/acm`

Throughout this text, problems are proposed at the end of each section in relation to the topic presented. They are accompanied with their identifiers to a judge site; for example [spoj:CMPLS] refers to the problem '*Complete the Sequence!*' at the URL `www.spoj.com/problems/CMPLS/`. The site `http://tryalgo.org` contains links to all of these problems. The reader thus has the possibility to put into practice the algorithms described in this book, testing an implementation against an online judge.

The languages used for programming competitions are principally C++ and Java. The SPOJ judge also accepts Python, while the Google Code Jam contest accepts many of the most common languages. To compensate for the differences in execution speed due to the choice of language, the online judges generally adapt the time limit to the language used. However, this adaptation is not always done carefully, and it is sometimes difficult to have a solution in Python accepted, even if it is correctly written. We hope that this situation will be improved in the years to come. Also, certain judges work with an ancient version of Java, in which such useful classes as `Scanner` are not available.

1.1.2 Responses of the Judges

When some code for a problem is submitted to an online judge, it is evaluated via a set of private tests and a particularly succinct response is returned. The principal response codes are the following:

Accepted Your program provides the correct output in the allotted time. Congratulations!

Presentation Error Your program is almost accepted, but the output contains extraneous or missing blanks or end-of-lines. This message occurs rarely.

Compilation Error The compilation of your program generates errors. Often, clicking on this message will provide the nature of the error. Be sure to compare the version of the compiler used by the judge with your own.

Wrong Answer Re-read the problem statement, a detail must have been overlooked. Are you sure to have tested all the limit cases? Might you have left debugging statements in your code?

Time Limit Exceeded You have probably not implemented the most efficient algorithm for this problem, or perhaps have an infinite loop somewhere. Test your loop invariants to ensure loop termination. Generate a large input instance and test locally the performance of your code.

Runtime Error In general, this could be a division by zero, an access beyond the limits of an array, or a `pop()` on an empty stack. However, other situations can also generate this message, such as the use of `assert` in Java, which is often not accepted.

The taciturn behaviour of the judges nevertheless allows certain information to be gleaned from the instances. Here is a trick that was used during an ICPC / SWERC contest. In a problem concerning graphs, the statement indicated that the input consisted of connected graphs. One of the teams doubted this, and wrote a connectivity test. In the positive case, the program entered into an infinite loop, while in the negative case, it caused a division by zero. The error code generated by the judge (Time Limit Exceeded ou Runtime Error) allowed the team to detect that certain graphs in the input were not connected.

1.2 Python in a Few Words

The programming language Python was chosen for this book, for its readability and ease of use. In September 2017, Python was identified by the website `https://stackoverflow.com` as the programming language with the greatest growth in high-income countries, in terms of the number of questions seen on the website, notably thanks to the popularity of machine learning.[1] Python is also the language retained for such important projects as the formal calculation system SageMath, whose critical portions are nonetheless implemented in more efficient languages such as C++ or C.

Here are a few details on this language. This chapter is a short introduction to Python and does not claim to be exhaustive or very formal. For the neophyte reader we recommend the site `python.org`, which contains a high-quality introduction as well as exceptional documentation. A reader already familiar with Python can profit

[1] `https://stackoverflow.blog/2017/09/06/incredible-growth-python/`

enormously by studying the programs of David Eppstein, which are very elegant and highly readable. Search for the keywords `Eppstein PADS`.

Python is an interpreted language. Variable types do not have to be declared, they are simply inferred at the time of assignment. There are neither keywords `begin/end` nor brackets to group instructions, for example in the blocks of a function or a loop. The organisation in blocks is simply based on indentation! A typical error, difficult to identify, is an erroneous indentation due to spaces used in some lines and tabs in others.

Basic Data Types

In Python, essentially four basic data types exist: Booleans, integers, floating-point numbers and character strings. In contrast with most other programming languages, the integers are not limited to a fixed number of bits (typically 64), but use an arbitrary precision representation. The functions—more precisely the *constructors*: `bool, int, float, str`—allow the conversion of an object to one of these basic types. For example, to access the digits of a specific integer given its decimal representation, it can be first transformed into a string, and then the characters of the string can be accessed. However, in contrast with languages such as C, it is not possible to directly modify a character of a string: strings are *immutable*. It is first necessary to convert to a list representation of the characters; see below.

Data Structures

The principal complex data structures are dictionaries, sets, lists and n-tuples. These structures are called *containers*, as they contain several objects in a structured manner. Once again, there are functions `dict`, `set`, `list` and `tuple` that allow the conversion of an object into one of these structures. For example, for a string `s`, the function `list(s)` returns a list `L` composed of the characters of the string. We could then, for example, replace certain elements of the list `L` and then recreate a string by concatenating the elements of `L` with the expression `''.join(L)`. Here, the empty string could be replaced by a separator: for example, `'-'.join(['A','B','C'])` returns the string `"A-B-C"`.

1.2.1 Manipulation of Lists, n-tuples, Dictionaries

Note that lists in Python are not linked lists of cells, each with a pointer to its successor in the list, as is the case in many other languages. Instead, lists are arrays of elements that can be accessed and modified using their index into the array. A list is written by enumerating its elements between square brackets `[` and `]`, with the elements separated by commas.

Lists The indices of a list start with 0. The last element can also be accessed with the index -1, the second last with -2 and so on. Here are some examples of operations to extract elements or sublists of a list. This mechanism is known as *slicing*, and is also available for strings.

The following expressions have a complexity linear in the length of L, with the exception of the first, which is in constant time.

`L[i]`	the `i`th element of `L`
`L[i:j]`	the list of elements with indices starting at `i` and up to (but not including) `j`
`L[:j]`	the first `j` elements
`L[i:]`	all the elements from the `i`th onwards
`L[-3:]`	the last three elements of `L`
`L[i:j:k]`	elements from the `i`th up to (but not including) the `j`th, taking only every `k`th element
`L[::2]`	the elements of `L` with even indices
`L[::-1]`	a reverse copy of `L`

The most important methods of a list for our usage are listed below. Their complexity is expressed in terms of n, the length of the list `L`. A function has *constant amortised complexity* if, for several successive calls to it, the average complexity is constant, even if some of these calls take a time linear in n.

`len(L)`	returns the number of elements of the list `L`	$O(1)$
`sorted(L)`	returns a sorted copy of the list `L`	$O(n \log n)$
`L.sort()`	sorts `L` in place	$O(n \log n)$
`L.count(c)`	the number of occurrences of `c` in `L`	$O(n)$
`c in L`	is the element `c` found in `L`?	$O(n)$
`L.append(c)`	append `c` to the end of `L`	amortised $O(1)$
`L.pop()`	extracts and returns the last element of `L`	amortised $O(1)$

Thus a list has all the functionality of a stack, defined in Section 1.5.1 on page 20.

***n*-tuple** An n-tuple behaves much like a list, with a difference in the usage of parentheses to describe it, as in `(1, 2)` or `(left, 'X', right)`. A 1-tuple, composed of only one element, requires the usage of a comma, as in `(2,)`. A 0-tuple, empty, can be written as `()`, or as `tuple()`, no doubt more readable. The main difference with lists is that n-tuples are immutable, just like strings. This is why an n-tuple can serve as a key in a dictionary.

Dictionaries Dictionaries are used to associate objects with values, for example the words of a text with their frequency. A dictionary is constructed as comma-separated key:value pairs between curly brackets, such as `{'the': 4, 'bread': 1, 'is': 6}`, where the keys and values are separated by a colon. An empty dictionary is obtained with `{}`. A membership test of a key `x` in a dictionary `dic` is written `x in dic`. Behind a dictionary there is a hash table, hence the complexity of the expressions `x in dic, dic[x], dic[x] = 42` is in practice constant time, even if the worst-case complexity is linear, a case obtained in the improbable event of all keys having the same hash value.

If the keys of a dictionary are all the integers between 0 and $n-1$, the use of a list is much more efficient.

A loop in Python is written either with the keyword `for` or with `while`. The notation for the loop `for` is `for x in S:`, where the variable `x` successively takes on the values of the container `S`, or of the keys of `S` in the case of a dictionary. In contrast, `while L:` will loop as long as the list `L` is non-empty. Here, an implicit conversion of a list to a Boolean is made, with the convention that only the empty list converts to `False`.

At times, it is necessary to handle at the same time the values of a list along with their positions (indices) within the list. This can be implemented as follows:

```
for index in range(len(L)):
    value = L[index]
    # ... handling of index and value
```

The function `enumerate` allows a more compact expression of this loop:

```
for index, value in enumerate(L):
    # ... handling of index and value
```

For a dictionary, the following loop iterates simultaneously over the keys and values:

```
for key, value in dic.items():
    # ... handling of key and value
```

1.2.2 Specificities: List and Dictionary Comprehension, Iterators

List and Dictionary Comprehension

The language Python provides a syntax close to the notation used in mathematics for certain objects. To describe the list of squares from 0 to n^2, it is possible to use a *list comprehension*:

```
>>> n = 5
>>> squared_numbers = [x ** 2 for x in range(n + 1)]
>>> squared_numbers
[0, 1, 4, 9, 16, 25]
```

which corresponds to the set $\{x^2 | x = 0, \ldots, n\}$ in mathematics.

This is particularly useful to initialise a list of length n:

```
>>> t = [0 for _ in range(n)]
>>> t
[0, 0, 0, 0, 0]
```

or to initialise counters for the letters in a string:

```
>>> my_string = "cowboy bebop"
>>> nb_occurrences = {letter: 0 for letter in my_string}
>>> nb_occurrences
{'c': 0, 'o': 0, 'w': 0, 'b': 0, 'y': 0, ' ': 0, 'e': 0, 'p': 0}
```

The second line, a *dictionary comprehension*, is equivalent to the following:

```
nb_occurrences = {}
for letter in my_string:
    nb_occurrences[letter] = 0
```

Ranges and Other Iterators

To loop over ranges of integers, the code `for i in range(n):` can be used to run over the integers from 0 to $n-1$. Several variants exist:

`range(k, n)`	from k to $n-1$
`range(k, n, 2)`	from k to $n-1$ two by two
`range(n - 1, -1, -1)`	from $n-1$ to 0 (-1 excluded) in decreasing order.

In early versions of Python, `range` returned a list. Nowadays, for efficiency, it returns an object known as an *iterator*, which produces integers one by one, if and when the `for` loop claims a value. Any function can serve as an iterator, as long as it can produce elements at different moments of its execution using the keyword `yield`. For example, the following function iterates over all pairs of elements of a given list:.

```
def all_pairs(L):
    n = len(L)
    for i in range(n):
        for j in range(i + 1, n):
            yield (L[i], L[j])
```

1.2.3 Useful Modules and Packages

Modules

Certain Python objects must be imported from a module or a package with the command `import`. A package is a collection of modules. Two methods can be used; the second avoids potential naming collisions between the methods in different modules:

```
>>> from math import sqrt
>>> sqrt(4)
2
>>> import math
>>> math.sqrt(4)
2
```

math This module contains mathematical functions and constants such as `log`, `sqrt`, `pi`, etc. Python operates on integers with arbitrary precision, thus there is no limit on their size. As a consequence, there is no integer equivalent to represent $-\infty$ or $+\infty$. For floating point numbers on the other hand, `float('-inf')` and `float('inf')` can be used. Beginning with Python 3.5, `math.inf` (or `from math import inf`) is equivalent to `float('inf')`.

fractions This module exports the class `Fraction`, which allows computations with fractions without the loss of precision of floating point calculations. For example, if `f` is an instance of the class Fraction, then `str(f)` returns a string similar to the form `"3/2"`, expressing `f` as an irreducible fraction.

bisect Provides binary (dichotomous) search functions on a sorted list.

heapq Provides functions to manipulate a list as a heap, thus allowing an element to be added or the smallest element removed in time logarithmic in the size of the heap; see Section 1.5.4 on page 22.

string This module provides, for example, the function `ascii_lowercase`, which returns its argument converted to lowercase characters. Note that the strings themselves already provide numerous useful methods, such as `strip`, which removes whitespace from the beginning and end of a string and returns the result, `lower`, which converts all the characters to lowercase, and especially `split`, which detects the substrings separated by spaces (or by another separator passed as an argument). For example, `"12/OCT/2018".split("/")` returns `["12", "OCT", "2018"]`.

Packages

One of the strengths of Python is the existence of a large variety of code packages. Some are part of the standard installation of the language, while others must be imported with the shell command `pip`. They are indexed and documented at the web site `pypi.org`. Here is a non-exhaustive list of very useful packages.

tryalgo All the code of the present book is available in a package called `tryalgo` and can be imported in the following manner: `pip install tryalgo`.

```
>>> import tryalgo
>>> help(tryalgo)         # for the list of modules
>>> help(tryalgo.arithm)  # for a particular module
```

collections To simplify life, the class `from collections import Counter` can be used. For an object `c` of this class, the expression `c[x]` will return 0 if `x` is not