Discrete Structures

(Algorithmic Approach)

Alexander Gamkrelidze

Contents

Introduction

In our everyday life, people do not need to understand what an algorithm is. But still, algorithms are omnipresent and play a much more important role as one can think. Literrally, they follow us at every step: The process of walking is also an algorithm such as "Lean forward a little, put the left foot forward, lean yourself forward a little, put the right foot forward, and repeat this process as many times as desired". This process of walking sounds to be very easy but in fact this is one of the most important unsolved problems in robotics.

Another example is a Khinkali recipe:

Input data:

Meat (mutton, can be mixed with beef), onions, red basil, savory, red pepperoni, black pepper, salt, flour, water

Output: Khinkali, Pshavian style

Procedure of the algorithm:

Algorithm **Pshavian Khinkali**

1. Make a stock: wash the bones, cover with water and boil until the taste is not watery;
   Sieve to clean from bone rests and put aside.
2. Cut the meat and all greens and vegetables in small peaces;
3. Mix salt into the warm stock (should taste like sea water) and cover the meat with it. mix well and repeat the procedure until the water is no longer absorbed;
4. Mix in the cut onions, red pepperoni and the greens, add some salt and black pepper to taste;
5. Take *exactly* the same amount of the stock used for the meat and make a hard dough: mix well with flour, roll out, roll into a compact cylinder, flatten, roll out, roll into a compact cylinder and repeat this procedures several time (ca. 10 min.). Cover and put into the refrigerator for ca. 20 min;
6. Repeat the above (last) step to get a hard but elastic dough (about 4 times);
7. Fill the rest of the stock with water into a large bowl and bring to a boil;
8. Cut a long peace from the dough, roll like a thick rope and cut into small peaces. Flatten the small peaces to thin, round discs;
9. Put 1Tbsp soft meat mixture on each disc, bind together khinkali-style, put into the boiling water and mix carefully to prevent from sticking;
10. Boil until khinkalis are inflated and swim on the surface for several minutes (ca. 10 - 12 min. in total);
11. Take out with a skimming ladle and — bon appetit!

end of the algorithm

Of course, the above recipe can not be considered as an algorithm in its exact definition, because it has many unclear moments, such as the lack of the stopping criterions while making the dough or adding salt and pepper "to taste" and so on. It needs some improvisation and the intervention of human creativity and feeling that is illegitimate in automatisation.

But it has some important similarities to that we call "an algorithm": it contains the exact steps one has to follow to make khinkali.

In general, an algorithm can be considered as a list of steps needed to solve a problem with given initial values (its input) without any intervention from outside.
Also, the following three points should be considered:

1. An algorithm should contain one or several steps;
2. When finishing one step, the next step (that follows immediately after it) should be executed;
3. The steps can be repeated several times (iterative execution).

Remark: To solve a problem, the total number of steps must be finite – an algorithm must terminate at some time, but as we will see later, there are some algorithms that never terminate, thus not giving an answer at all. Of course, such algorithms are useless, but for many problems, no algorithm can exist that finish their computation in finite time (such problems are called unsolvable, but this is a topic of our further courses).

In algorithm design, two aspects are crucial:

1. Correctness — does it do the required job? Does it give the right output to each input?
2. Computational time — how fast is the algorithm? What is the maximal number of steps needed to get the answer for any input?

Obviously, nobody needs an algorithm that does not do its job, or contains some (or many) mistakes.
Also, there is no use of an algorithm that gives the answer in several years or even centuries and millennia. The computation should be guaranteed in reasonable time, the faster the better.
There are many problems around us: from making khinkali to sending rockets into the space. A natural question is wether we can write an algorithm for *any* problem? Unfortunately, the answer is *No.* There are algorithms that can not be solved automatically by an algorithm (at least from our actual scientific viewpoint).
More shocking news is that there are "much more" unsolvable problems than solvable! That means that there are more problems that can not be tackled by artificial intelligence — a sort of feeling and human intuition is needed to master them.

And what about the solvable problems? Can we get an answer for them efficiently?
The prospects here are not satisfiable as well: There are many very important problems that can not be solved in reasonable time using modern knowledge (not meaning that such solutions can not be discovered in future).

The moral from these stories is that, getting a new problem, one has to determine if it is solvable and in case it is, wether it can be solved in reasonable time (there are some well known cases that many bright heads spent years in searching a solution to a problem that turned out to be unsolvable).
But how can we know wether a problem is solvable or intractable? Unfortunately, we do not have one method to determine such things. But we have some theoretical insight into the design and analysis of algorithms that helps us to understand such questions. And this course is the beginning of that theory. It intends to help students to start to understand the basics of problem solving by algorithms and their analysis.
Another question is how to tackle very important problems that turns out to be difficult or even unsolvable? People have developed methods to write approximating algorithms (the answer will be *almost* correct), or probabilistic approaches (the answer will be *most of the time* correct) or restriction of data etc. But this is the topic of our further courses.

Chapter 1

Simpe Algorithms to Begin With

1.1 The Wolf, the Goat and the Cabbage

As Kernighan and Ritchie wrote in their seminal book "The C Programming Language", the best way to learn a programming is to write programs. The same can be said about algorithms, so let us start with some simple problems to see the most important points in algorithm design.
As a simplest example consider a well-known kids riddle (WGC for short):

A man wants to transport a wolf, a goat and a cabbage from one bank of a river to another by a boat (fig. 1.1).

Figure 1.1: Initial an final states of the WGC problem

As long as the man is on the same side of the river with the animals, they behave well and do not hurt each other. But if the wolf and the goat are left unattended (the man is on the other side of the river), the wolf eats the goat. Similarly, if the goat and the cabbage are left unattended, the goat eats the cabbage. If the wolf and the cabbage are left unattended, no incident will happen.

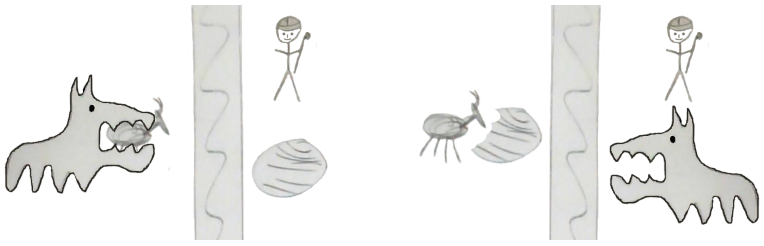


Figure 1.2: Forbidden states

What is the algorithm to transport the animals and the plant from one bank of the river to another?
First of all, let us state the problem (input, output, and restrictions).

Thing to rememter: **A WELL-STATED PROBLEM IS WORTH HALF THE SOLUTION.**

Given: A river and a man, a boat, a wolf, a goat and a cabbage, all on its one bank (fig. 1.1 left).
Output: All of them on the other bank of the river (fig. 1.1 right).
Restrictions: Only the man and one animal (or plant) fit in the boat (first restriction);
Wof and goat or/and goat and cabbage can not be left unattended (second restriction).

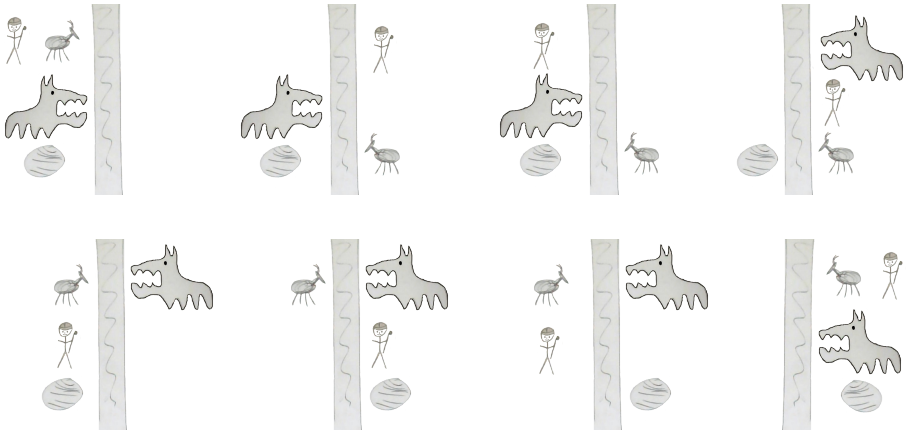


Figure 1.3: Steps of the algorithm

To solve this problem, we can use the algorithm (steps to be executed to solve the problem) in fig. 1.3.

Algorithm 1.1**:** Wolf, Goat and Cabbage
Given: a river and a man, a boat, a wolf, a goat, and a cabbage on one bank;

1: Promote the goat to the right bank;
2: Go back to the left bank;
3: Promote the wolf to the right bank;
4: Bring the goat to the left bank;
5: Promote the cabbage to the right bank;
6: Go back to the left bank;
7: Promote the goat to the right bank.
End of the algorithm

First of all, we should prove the correctness of the algorithm: Show that executing the steps one after the other with given input yelds the desired result without violating any restriction point.

Exercise 1.1: Prove the correctness of this algorithm (hint: show what happens after each step).

To estimate the quality (the speed) of the algorithm, we should count its steps. To do so, one should first define what a "step" means. In general, this is a conventional question (depends on what the authors mean under this term), but as we will see, in the theory of algorithms, there are some generally accepted simple actions considered as steps. In our case let us assume that one crossing of the river with the boat is one step. Note that this definition is independent from the time one needs to cross: we are interested in general questions such as *how many steps are needed*, that are independent from *who* is executing the task and how fast can he execute each step.
In algorithm analysis, the *method* should be analysed, independent from the tool (computer) one uses to execute the algorithm.

Exercise 1.2: Count the number of steps in the above algorithm.

Usually, in everyday-life problems, most information is redundant. For example, we do not care how large the river is, how scary the wolf is, what is the color of the boat etc. We are interested only in the information that is necessary to solve the problem.
We can describe the left bank as a set A, and the right bank as the set B. Further, assign a lathin letter to each animal: Human $\mapsto$ H, Wolf $\mapsto$ W, Goat $\mapsto$ G and Cabbage $\mapsto$ C (fig. 1.4).

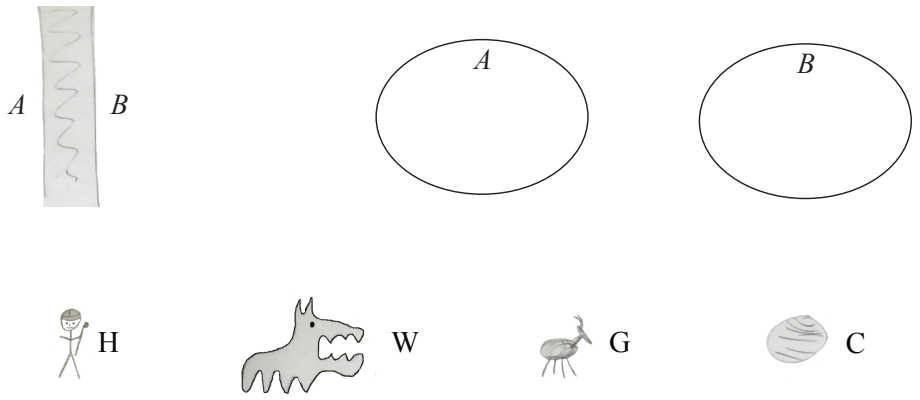


Figure 1.4: Formalisation of the above pictures

Then the input and the output will be $A = \{\text{W,G,C,H}\}, B = \emptyset$ and, respectively, $A = \emptyset, B = \{\text{W,G,C,H}\}$ (fig. **??**).

More formally, we have:

Input: Two sets $A = \{\text{W,G,C,H}\}$ and $B = \emptyset$.

Output: $A = \emptyset$ and $B = \{\text{W,G,C,H}\}$.

Restrictions: In each step, the letter H should be put from one set to the other together with exactly one or no other letter. The set that does **not** contain the letter H should **not** contain the letters G, C and/or W, G.

The statment will be slightly simplified by using numbers instead of letters. For example, Human $\mapsto$ 11, Wolf $\mapsto$ 1, Goat $\mapsto$ 2, Cabbage $\mapsto$ 3. In this case, the term "wolf and goat are alone on the same bank" or "goat and cabbage are on the same bank" means that the sum in one of the sets is odd. And the term "the human is on the bank" means that the sum of the elements of the appropriate set is greater than or equal to 10.

Exercise 1.3: What does it mean that the sum of numbers in one set is 6 ?

Exercise 1.4: State the WGC problem in terms of numbers.

Exercise 1.5: Write an algorithm for the problem stated in the previous exercise. Write the configurations of the sets after each step.

Exercise 1.6: Prove the correctness of the algorithm from the above exercise (Hint: show that no restriction is violated after each step and the input and output are as requested).

Exercise 1.7: Consider the following algorithm:

Algorithm 1.2**:** WGC (fast version)
Input: A river and a human, wolf, goat and a cabbage on one bank;

1: Promote the goat to the right bank;
2: Return to the left bank;
3: Promote the wolf to the right bank;
4: Return to the left bank;
5: Promote the cabbage to the right bank.
End of the algorithm

Is this a correct algorithm? Explain your answer.

Remark: In the above problem, the order of the input does not affect the result. Such problems are called *completely symmetric* (or sometimes *partially symmetric* if only some input elements can be exchanged).
Often, the problems are not symmetric at all, as shown in the following simple example:

Given two natural numbers a and b, calculate $\frac{a}{b}$.

Exercise 1.8: Consider the problem of sorting n whole numbers in ascending order. What is its input and output? Is it completely symmetric? Partially symmetric? Non-symmetric?

Exercise 1.9: Give an example of a partially symmetric problem.

Exercise 1.10: Write an algorithm for the following problem: For the given a sequence of 10 whole numbers, calculate the sum of its odd elements. What can be the maximal (minimal) number of steps for this algorithm? Assume that checking the number for being odd and addition are one steps. Is this problem completely symmetric? Partially symmetric? Non-symmetric? What is its input and output? Are there any restrictions?

Last but not least, a small puzzle to solve:

Two men meet on the street. They haven't seen each other for many years. They talk about various things, and then after some time one of them says: "Since you're a mathematician, I'd like to give you a problem to solve. You know, today's a very special day for me: All three of my sons celebrate their birthday this very day! So, can you tell me how old each of them is?" "Sure," answers the mathematician, "but you'll have to tell me something about them". "OK, I'll give you some hints", replies the father of the three sons, "The product of the ages of my sons is 36". "That's fine," says the mathematician, "but I'll need more than just this". "The sum o f their ages is equal to the number o f windows i n that building", says the father pointing at a structure next to them. The mathematician thinks for some time and replies, "Still, I need an additional hint to solve your puzzle". "My oldest son has blue eyes", says the other man. "Oh, this is sufficient!" exclaims the mathematician, and he gives the father the correct answer.

What are the ages of his three sons?

1.2 Summary

Considering some simple examples, we showed how to construct algorithms – a list of actions necessary to solve a given problem.
To state a problem, one needs to clearly define its input and output as well as the restrictions that should be regarded during the execution of the algorithm.
After writing an algorithm, one shold proof its correctness (show somehow that after the execution of the algorithm with given input the appropriate output will be obtained without violating any restriction).
Last but not least – to estimate the goodness of an algorithm – one has to count (or estimate) the (maximal and/or minimal) number of steps needed to complete the algorithm with any input. Tis process is called the estimation of the time complexity of the algorithm. In the above examples, the estimation process was quite simple. As we will se later, it can become much more complex and will need some techniques to be done.

Chapter 2

Recursion and Iteration

2.1 The Boats problem

Let's consider the well-known boats problem:

Input: A narrow river with a small bay; A long black boat on the left side of the bay and a small white boat on the right side of it.

Output: The white boat on the left side and the black one on the right side of the bay (the boats should pass one another).

Restrictions: The river is narrow enough to pass only one (black or white) boat; The black boat can not enter the bay but the white one can.

Fig. 2.1 shows the input and output configurations as well as the restrictions.

Figure 2.1: Input, output and restrictions

In order to pass the boats, the the following algorithm should be completed:

Algorithm 2.1**: "Bypassing one boat"**
Input: A narrow river with a bay; A long black boat on the left side and a small white boat on the right side of the bay

---

1: The white boat enters the bay;
2: The black boat passes by;
3: The white boat exits the bay.
End of the Algorithm

It is easy to show that this algorithm gives us the desired result and its steps do not violate the restrictions (fig. 2.2).
The above algorithm can be denoted as A_1 (bypassing 1 boat). Its steps are schematically shown in fig. 2.2. Obviously, if we apply the A_1 algorithm on the initial input configuration, the correct configuration will be obtained. Now consider the initial condition with two white boats on the rigt side of the bay shown in fig. 2.3 (left) and the desired output (same figure right). Let us call it "bypassing two boats".

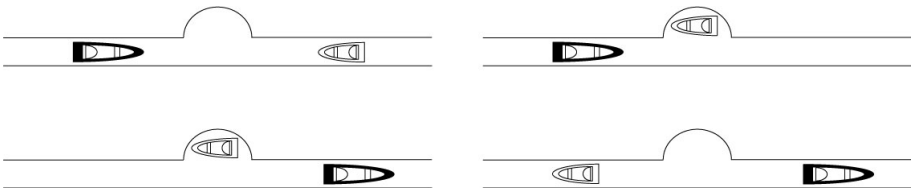


Figure 2.2: The One-Boat Algorithm

Figure 2.3: Initial and desigred positions with two boats

Applying the three steps described above we will obtain the configuration shown in fig. 2.4 (left). After this, if the black boat navigates back to its initial position (this procedure is called U), we obtain the same configuration that is initial for A_1. Thus, applying A_1 we will reach the desired output configuration for the two boats problem fig. 2.4 (right).

Figure 2.4: Configuration after applying A_1

That means that if the initial condition is as shown in fig. 2.3 (left) and we apply A_1 we obtain the configuration shown in 2.4 (left). Applying U leads to the position shown in 2.4 (right) and the initial condition for the one boat problem has been established, thus after applying A_1 we should obtain the desired final configuration.
Hence, the algorithm tor the two boats problem A_2 can be described as $A_2 = A_1, U, A_1$ (in words: execute A_1, then U and A_1 again).

Now suppose that the algorithm A_n manages to bypass n white boats (fig. 2.5). Note that we already considered this algorithm for the special cases $n = 1$ and $n = 2$.

Consider the $n+1$ boat problem with initial and final configurations as shown in fig. 2.6 (upper left and right) and execute the algorithms A_1, U, we obtain the initial configuration for the A_n algorithm (its proper input) as shown in fig. 2.6 lower left and wight).

Obviously, after executing A_n we obtain the desired final configuration (see fig. 2.6 upper right).

As a result, we have $A_{n+1} = A_1, U, A_n$. As we know the exact description of A_1, we can easily calculate A_2 (here, $n + 1 = 2$ and $n = 1$: First execute A_1, then U and then A_1 again; To compute A_3, execute A_1, then U and then A_2 as described above: $A_3 = A_1, U, A_1, U, A_1$.
Hence, we can compute the exact description of the "recursive" algorithm A_n for any $N \in \mathbb{N}$:

$$\begin{aligned} A_n &= A_1, U, A_{n-1} \\ &= A_1, U, A_1, U, A_{n-2} \\ &= A_1, U, A_1, U, A_1, U, A_{n-2} \\ &= \underbrace{A_1, U, A_1, U, \cdots, A_1}_{n\text{-times}} \end{aligned}$$

Exercise 2.1: What is A_7? (example: $A_4 = A_1, U, A_3 = A_1, U, A_1, U, A_1, U, A_1$)

Note that the algorithm A_n "uses itself" with lower parameter (e.g., $A_2 = A_1, U, A_1$; $A_7 = A_1, U, A_6$ etc.)
If an algorithm "uses itself", it is called **recursive**. Hence $A_n = A_1, U, A_{n-1}$ is a recursive algorithm.
Note that any recursive algorithm can be described non-recursively as a so-called **iterative** algorithm where one or more operations are executed consecutive and repeated several times.
In our example, the n-boat algorithm can be described as follows:

Figure 2.5: Initial and final configurations for the general algorithm A_n

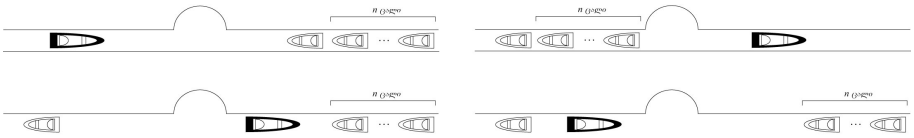


Figure 2.6: Configurations for A_{n+1}

The part of the algorithm that is repeated is called a **cycle** and the sequence of instructions to be repeated is called its **body**.
Exercise 2.2: prove that the number of steps of A_n is $4n$ (formally we write $T(A_n) = 4n$).
Exercise 2.3: How can we change the algorithm to have $T(A'_n) = 4n - 1$?

Algorithm 2.2**: Boats Problem**
(Non-recursive – iterative – version)
Input: $n \in \mathbb{N}$ (number of boats)

```
1: repeat n-times:
2:  {
3:    White boat enters the bay;
4:    Black boat passes by;
5:    White boat comes out of the bay;
6:    Black boat goes backwards
7:  }
   End of the Algorithm
```

2.2 Towers of Hanoi

In 1883, a french mathematician Eduard Lukas stated the following problem:

Input: Three rods A, B, C. On A we have n disks of different sizes to form a pyramid (smaller on larger, see fig. 2.7 (a)).

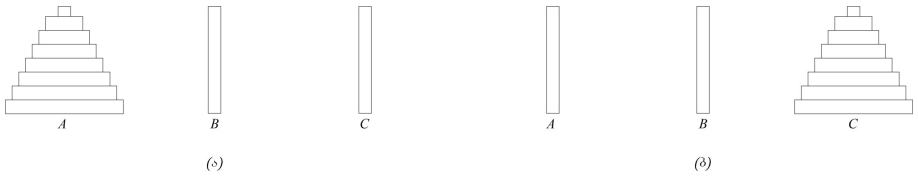


Figure 2.7: Initial and final states of the Towers of Hanoi problem

Output: The initial pyramid on rod C (fig. 2.7 (b)).

Restriction: One and only one upper disk should be taken from top of one rod and moved to the top of another rod; No larger disk can be placed on top of the smaller one.

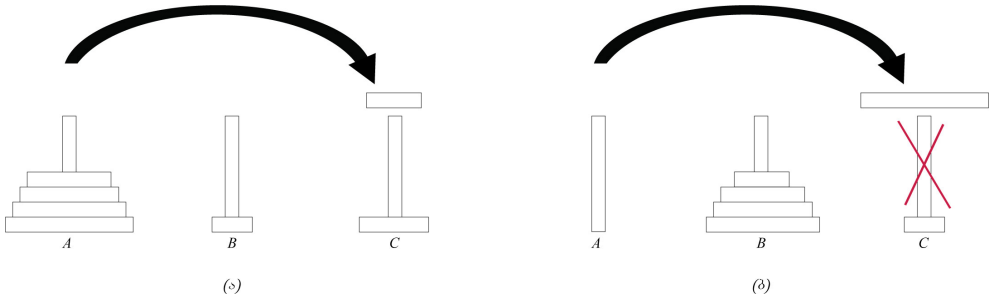


Figure 2.8: Proper *(a)* and improper *(b)* moves

Consider the simplest case of a one-disk pyramid ($n = 1$). Obviously, one move suffices to complete the task. For formal description, moving one disk from A to C will be described as $A_1^{A,C}$.
In order to move a two-disk pyramid from A to C, the following steps must be completed:

1. From A, move the upper disk to B (algorithm $A_1^{A,B}$, fig. 2.9 *(b)*);
2. From A, move the upper disk to C (algorithm $A_1^{A,C}$, fig. 2.9 *(c)*);
3. From B, move the upper disk to C (algorithm $A_1^{B,C}$, fig. 2.9 *(d)*).

Formally, we denote the above process of transportation of a two-disk pyramid as $A_2^{A,C}$.
Generally, the algorithm that moves an n-disk pyramid from one rod X_1 to another X_2 can be described as $A_n^{X_1,X_2}$.
Here, $n \in \mathbb{N}$, $X_1, X_2 \in \{A, B, C\}$ da $X_1 \neq X_2$.
For example, $A_{13}^{C,A}$ denotes an algorithm that moves a 13-disk pyramid from C to A, and $A_{108}^{B,A}$ denotes an algorithm that moves 108-disk pyramid from B to A.
Knowink how to move a two-disk pyramid from one rod to another, we can easily construct an algorithm $A_3^{A,C}$ to move a 3-disk pyramid:
Consider a three-disk pyramid as a two-disk pyramid put on one largest disk (fig. 2.12 *(a)*).

Obviously, due to $A_2^{A,B}$, the upper two-disk pyramid can be moved to B (fig. 2.12 *(b)*), and then by $A_1^{A,C}$, the largest disk will be moved from A to C (fig. 2.12 *(g)*) and finally by $A_2^{B,C}$, the two-disk pyramid will be moved from B to C (fig. 2.12 *(d)*).

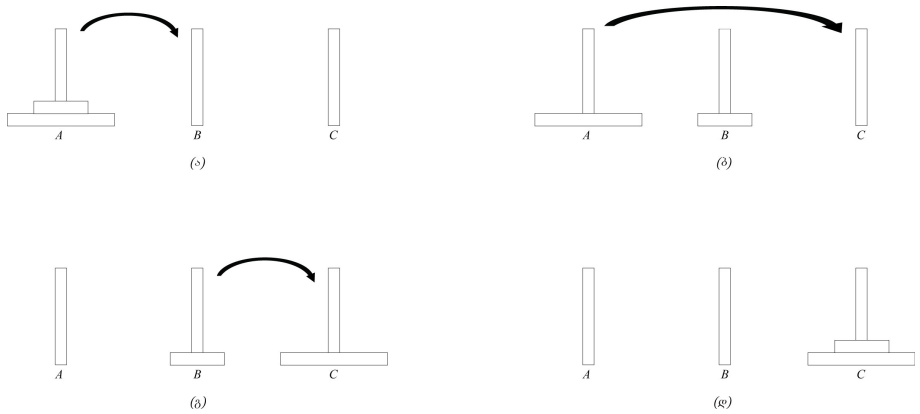


Figure 2.9: Operations needed to move a two-disk pyramid

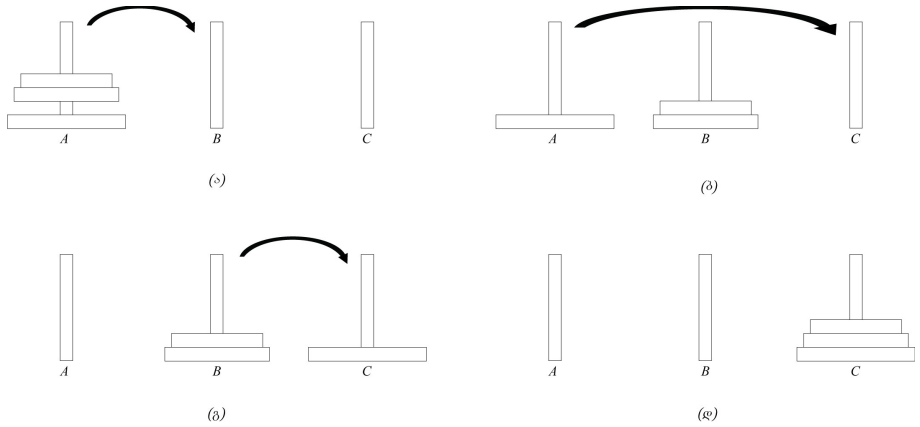


Figure 2.10: moves for A_3

By the recursive definition we have $A_3^{A,B} = [A_2^{A,B}, A_1^{A,C}, A_2^{B,C}]$ (first execute $A_2^{A,B}$, then $A_1^{A,C}$ and finally $A_2^{B,C}$). Note that $A_2^{A,B}$ and $A_2^{B,C}$ consist of several steps: $A_2^{A,B} = [A_1^{A,C}, A_1^{A,B}, A_2^{C,B}]$ and $A_2^{B,C} = [A_1^{B,A}, A_1^{B,C}, A_2^{A,C}]$.

Exercise 2.4: Describe recursively $A_3^{B,C}$, $A_3^{C,A}$, $A_3^{A,B}$, $A_3^{B,A}$ da $A_3^{C,B}$.

Knowing how to move a three-disk pyramid, we can easily describe an algorithm for four disks by recursion: $A_4^{X_1,X_2}$ (e.g., $A_4^{A,C} = [A_3^{A,B}, A_1^{A,C}, A_3^{B,C}]$.

Exercise 2.5: Describe recursively $A_4^{B,C}$, $A_4^{C,A}$, $A_4^{A,B}$, $A_4^{B,A}$ and $A_4^{C,B}$.

Knowing how to move an n-disk pyramid, we can easily describe an algorithm for $n+1$ disks by recursion: $A_{n+1}^{X_1,X_2}$ (see fig. 3.1):

$$A_{n+1}^{X_1,X_2} = [A_n^{X_1,X_3}, A_1^{X_1,X_2}, A_n^{X_3,X_2}], \quad X_1 \neq X_2 \neq X_3, \quad X_1, X_2, X_3 \in \{A, B, C\}.$$

Like above, to make things easier, in this example we show the move of n disks at once despite the fact that it consist of many steps.

Exercise 2.6: Explain the meaning of the following notations: $A_7^{B,C}$, $A_{12}^{C,B}$, $A_4^{B,C}$.

It is easy to show that during the execution of $A_1^{X_1,X_2}$, no restriction is violated. According to the recursive description of $A_2^{A,C}$, the first steps are from $A_1^{A,B}$. Obviously, no restrictions are violated during its execution. Next, $A_1^{A,C}$ should be executed. Since C is empty, the disk from A can be moved without violations and C will contain the largest disk. Due to this, the last steps, $A_1^{B,C}$, can be executed without problems (the largest disk does not block the rod).

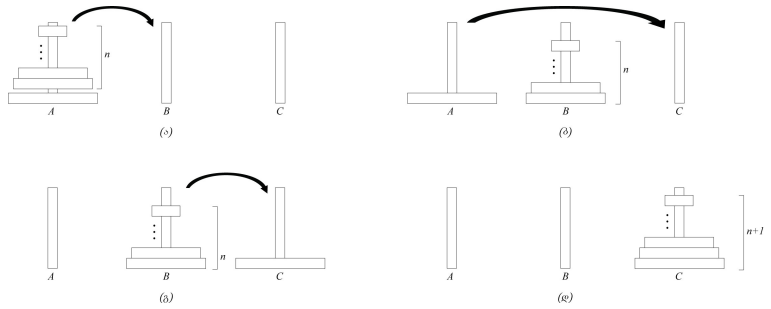


Figure 2.11: Operations needed to move an $n+1$-disk pyramid

Similarly we can reason that if $A_3^{A,C}$ described as above is executed, no restrictions are violated: First, $A_2^{A,B}$ is executed (fig. 2.12(a)). It does not violate the restrictions because B da C are empty and A has the largest disk on the bottom. as a result, we obtain the largest disk on A and a two-disk pyramid on B (fig. 2.12i(b)). The second stage is $A_1^{A,C}$ that obviously does not violate any condition. The resulting configuration (the largest disk on C and a two-disk pyramid on B with ampty A, fig. 2.12(g)) is the input of the last sub-algorithm $A_2^{B,C}$.

Exercise 2.7: Show that the execution of $A_2^{B,C}$ with the described input does not violate any restriction and the resulting configuration will be the solutionn of a 3-disk Towers of Hanoi problem (fig. 2.12(d)).

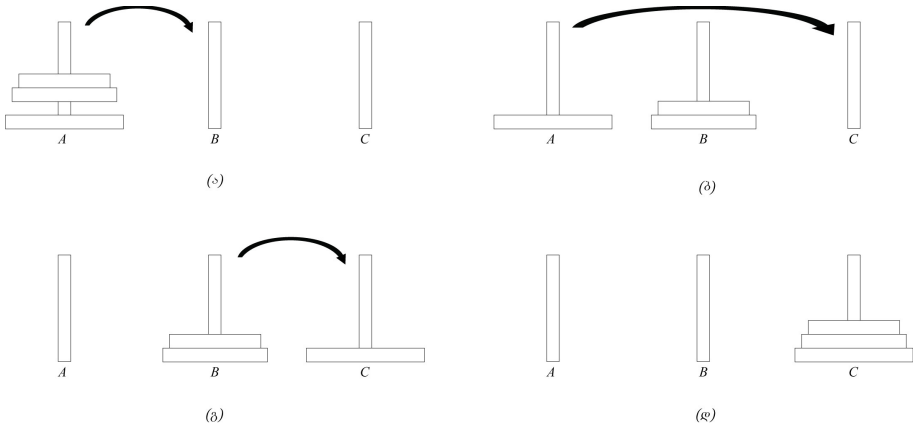


Figure 2.12: Operations needed to solve the 3-disk Towers of Hanoi problem

Exercise 2.8: Explain the operations that will be executed according to the term $H_3^{A,C} = [H_1^{A,B}, H_2^{A,C}, H_1^{B,C}]$. Is there any cviolation of restrictions possible during the computation?

Now consider the iterative description of the Tower of Hanoi algorithm axla ki ganvixiloT hanois koSkebis iteraciuli algoriTmi:

Algorithm 2.3**: Towers of Hanoi**
(Non-recursive – iterative – version)
Input: $n \in \mathbb{N}$ (The number of disks forming a pyramid on the rod A, the rods B and C are empty)

1: Repeat until the n-disk pyramid is moved from A to another rod
2: {
3: Move the minimal disk to right (of from C to A);
4: Move the non-minimal disk (there will be only one possibility)
5: }
End of the Algorithm

Note that, despite its simple description, the prove of the correctness of the iterativealgorithm is quite complicated.

Also, the estimation the number of its steps is also difficult because the number of the repetition of the cycle body is not clear.
These questions will be discussed in the next chapter.

Exercise 2.9: Consider the above iterative algorithm. To which rod will it move 3, 4, 5 and 6 disk pyramids beginning with rod A?
In general, to which rod will it move an odd-disk and even-disk pyramid?

2.3 Ancient Greek Problems: Ruler-and-Compass Constructions

Since ancient Greece, the so-called "ruler-and-compass construction" problems were stated, some of them remaining unsolved for several thousand years until novel techniques and theories were developed in the 19th sentury with deep insight into the core of modern mathematics. The answers to the unsolved problems were intriguing: they all turned out to be unsolvable with available methods (resources). Thus, they can be considered as the oldest unsolvable problems of human history. However, it should be mentioned that, in the contrary to other generally unsolvable problems, they are not solvable using only compass and ruler but can be solved with other methods.

Input: Compass, ruler and two points on the plane (the distance between them assumed to be 1);
Any geometric figure;
A real number $\xi \in \mathbb{R}$.

Output: Define if the given figure or two points with distance between them equal ξ are constructable with compass and ruler.

Restriction: With a ruler, one can only draw a line between two points A, B or extend an existing line as desired;
Given any three points A, B and O, one can draw a circle with center O and radius $|A, B|$ (fig. 2.13).

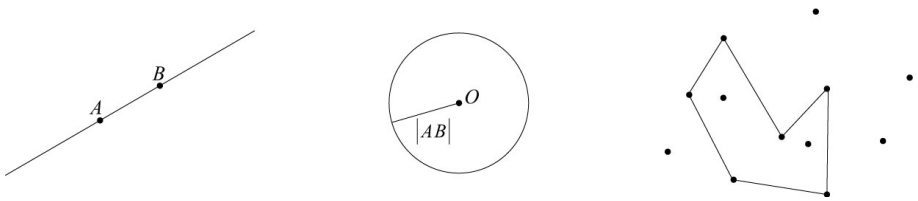


Figure 2.13: Geometric objects constructed with a ruler and a compass

Given a set of (already constructed) points $\mathcal{S} = \{A_1, A_2, ..., A_n\}$, a closed path of broken lines on some of them with its interior form a geometric figure that can be constructed by ruler and compass.

Definition 2.1: Here we determine some formal notations:

- $\forall$ for all;
- $\exists$ exists;
- $\in$ belongs to;
- $\Rightarrow$ it follows

A new point A_{n+1} is said to be constructed with ruler and compass if:

- $\exists\ A_i, A_j, A_k, A_l \in \mathcal{S}$ (to be read as: *there exist* points A_i, A_j, A_k, A_l, that *belong to* the set $\mathcal{S}$) and A_{n+1} is the intersection point of the lines (A_i, A_j) and (A_k, A_l) (fig. 2.14 left);
- $\exists\ A_i, A_j, A_k, A_l, O \in \mathcal{S}$ and A_{n+1} is an intersecting point of (A_i, A_j) and a circle with centre O and radius $|A_k, A_l|$ (fig. 2.14 middle);
- $\exists\ A_i, A_j, A_k, A_l, O_1, O_2 \in \mathcal{S}$ amd A_{n+1} is an intersection point of the circles with centre O_1, radius $|A_k, A_l|$ and centre O_2, radius $|A_i, A_j|$ (fig. 2.14 right).

Note that some of the points $A_i, A_j, A_k, A_l, O_1, O_2 \in \mathcal{S}$ can coinside and $\mathcal{S}$ is the set of already constructed points.

A geometric figure is said to be constructed with ruler and compass if it is bordered by a closed sequence of consecutive line segments whose endpoints are already constructed with ruler and compass.

a real number $\xi \in \mathbb{R}$ is said to be constructible by ruler and compass if two points A and B with the distance $|AB| = \xi$ are constructible.

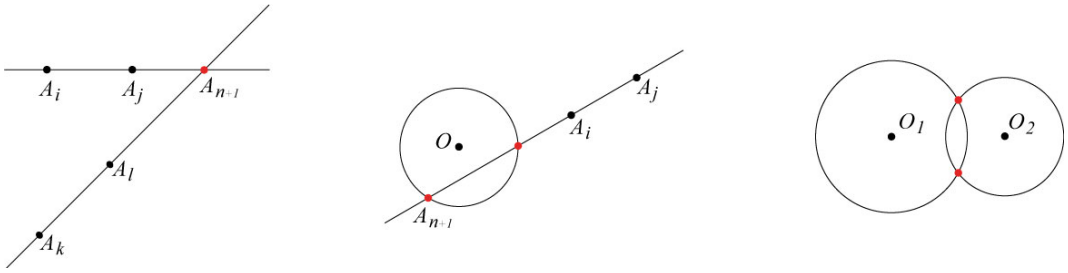


Figure 2.14: Construction of new points with compass and ruler

Initially, two points A' adn B' are given with the distance $|A', B'| = 1$ between them (the unit 1 is constructed). To construct the number 2 (to construct two points with the distance of 2 units between them), the following algorithm can be applied (in general, for already constructed points A and B, this algorithm will construct the number $|A, B| + 1$):

Input: Two points A and B

1. draw a line on A and B;
2. Draw a circle with center B and radius 1;

   This circle will intersect the AB line in two points: D (in the direction to A) and C (in the opposite direction to A).
3. Output: A and C.

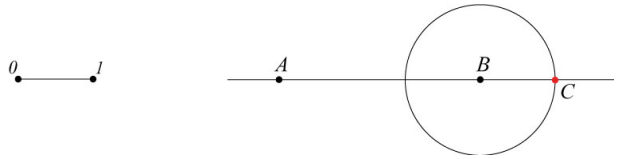


Figure 2.15: Constructing $|A, B| + 1$

Denoting this algorithm by N, with the input A and B, we have $N(A, B) = (A, C)$. Obviously, $|A, C| = |A, B| + 1$. Given two initial points A and B with distance 1 between them, one can construct any natural $n \in \mathbb{N}$ with the following recursive algorithm:

- $P_1 = (A, B)$;
- $P_n = N(P_{n-1})$.

Exercise 2.1: With given points A, B, C, D, write an algorithm that constructs a line segment with the length $|A, B| + |C, D|$. Prove its correctness and count the number of steps (assume that one step is either drawing a line or a circle).

Exercise 2.2: With given points A, B with $|A, B| > 1$, write an algorithm that constructs a line segment with the length $|A, B| - 1$. Prove its correctness and count the number of steps (assume that one step is either drawing a line or a circle).

Given two points A and B, it is easy to construct the linear bisector and also the middle point of the line segment $[A, B]$:

Input: Two points A and B (fig. 2.16 (a)).

- Draw a circle with center A and radius $|A, B|$;
- Draw a circle with center B and radius $|A, B|$ (fig. 2.16 (b))

  Result 1: Intersection points of the circle C and D.
- Draw a line on C and D (fig. 2.16 (g)).

  Result 2: Intersection point K of this line and the segment A, B.
- Output: Two points C da K (fig. 2.16 (d)).
  K is the middle point and (C, K) is the linear bisector.

Exercise 2.3: Prove the correctness of the above algorithm and count the number of its steps.

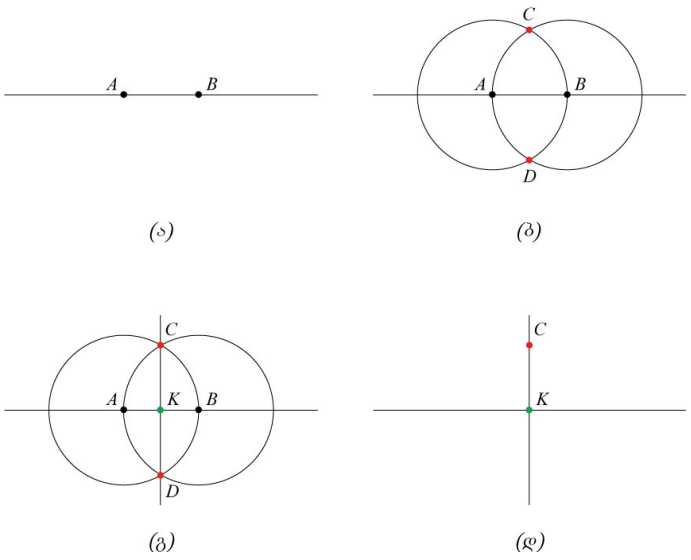


Figure 2.16: Construction of the linear bisector and the middle point of $[A, B]$

Denoting this algorithm by $P(A, B)$ we have $P(A, B) = (C, K)$.

Given two points A, B and C outside the line (A, B), we can construct a perpendicular line to (A, B) on C. That means that we construct a point D such that the line (CD) is perpendicular to the initial line (AB):

Input: Three points A, B and C not on (A, B) line (fig. 2.17 (a)).

- Draw a circle with centre C and radius $|A, C|$ (fig. 2.17 (b));

  Result 1: An intersection point L of this circle and (A, B);
- Execute $P(A, L)$;

  Result 2: Points K and T, where T lies on (A, B) line (fig. 2.17 (g)).
- Oitput: Two points C and T (fig. 2.17 (d)).

Exercise 2.4: In the algorithm above, $P(A, L)$ should be executed. Give a detailed description of the complete process by drawings.

Exercise 2.5: What happens if the circle with centce in C and radius $|A, C|$ touches the (A, B) line in one point without the second point L?

Exercise 2.6: Explain why we obtain two additional points K, T after the execution of $P(A, L)$.

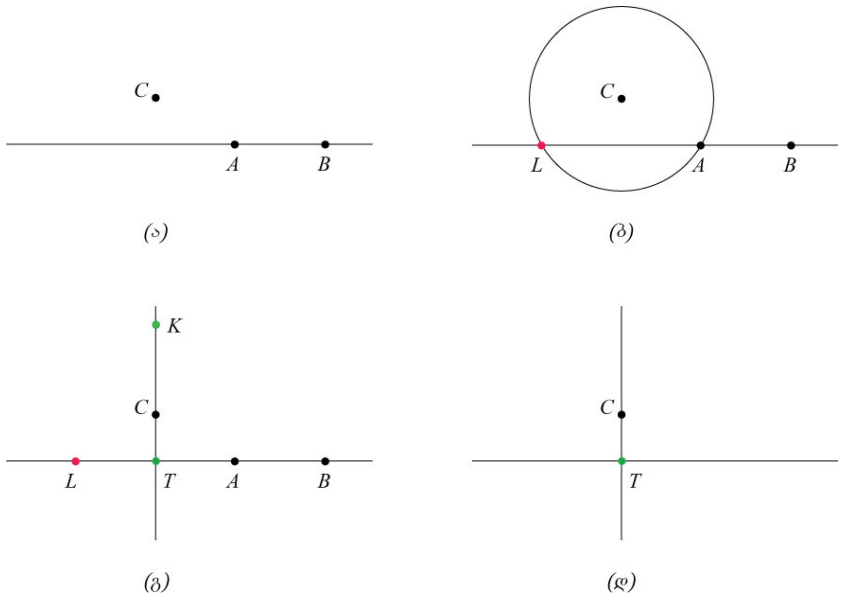


Figure 2.17: Constructing a perpendicular line from a given point

Exercise 2.7: Prove that the line (C, T) is perpendicular to the line (A, B).

Exercise 2.8: Given three co-linear points A, B and C between them. Write an algorithm that constructs a perpendicular line to (AB) that contains C.

Exercise 2.9: Given three points A, B and C not on (AB). Write an algorithm that constructs a line parallel to AB containing C.

For two constructed numbers $a_1, a_2 \in \mathbb{N}$ we can construct two points B_1, B_2 so that $|B_1, B_2| = a_1 \cdot a_2$:

Input: Four points A_1, A_2, A_3, A_4 with $|A_1, A_2| = a_1$ and $|A_3, A_4| = a_2$;

- On A_1, construct a line perpendicular to (A_1, A_2) (fig. 2.18 (a)) ;
- Mark a new point E on it with $|A_1, E| = 1$ (fig. 2.18 (b));
- On the same new line mark a point F with $|A_1, F| = |A_3, A_4|$ (fig. 2.18 (b)).
- Construct a line on E and A_2;
- Construct a line on F parallel to (E, A_2);

  Result: Intersection point of this line with (A_1, A_2) (fig. 2.18 (g));
- Output: Two points A_1 and K (fig. 2.18 (d)).

Exercise 2.10: Prove that $|A_1, K| = a_1 \cdot a_2$. Hint: use the triangle similarity properties.

Exercise 2.11: Prove that the algorithm is correct if $a_2 < 1$.

Similarly, for given four points A_1, A_2, A_3, A_4 with $|A_1, A_2| = a_1$ and $|A_3, A_4| = a_2$, we can construct $\frac{a_1}{a_2}$:

Input: Four points A_1, A_2, A_3, A_4 with $|A_1, A_2| = a_1$ da $|A_3, A_4| = a_2$;

- On A_1, construct a line parallel to (A_1, A_2) (fig. 2.19 (a));
- Mark a new point E on this line with $|A_1, E| = 1$ (fig. 2.19 (b)) .

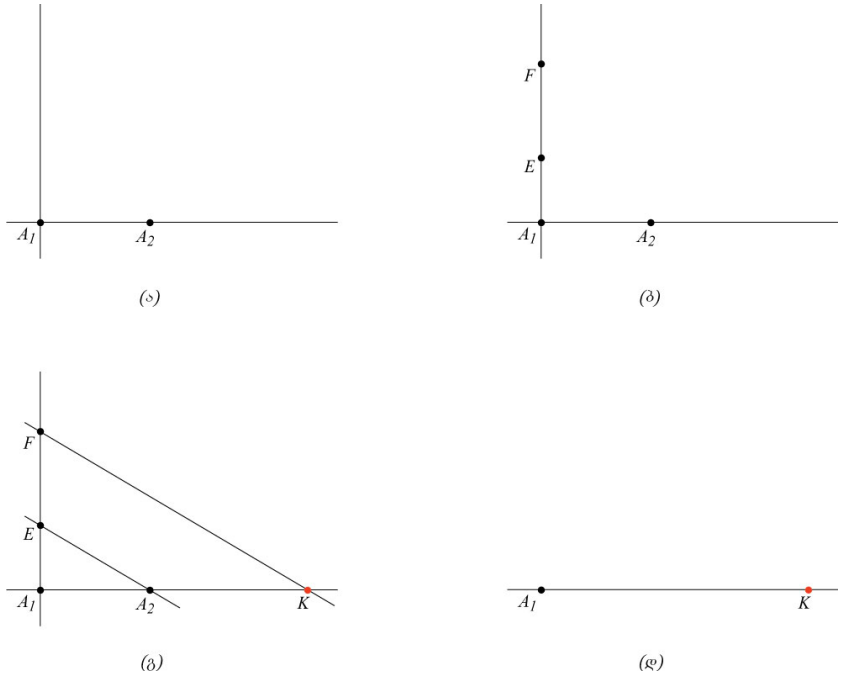


Figure 2.18: Construction of a line segment with the length $|A_1, A_2| \cdot |A_1, F|$

- On the same line, mark a new point F with $|A_1, F| = |A_3, A_4|$ (fig. 2.19 (b));
- Construct a line on F and A_2;
- E, construct a line parallel to $|F, A_2|$;

  Result: An intersection point K of this line with (A_1, A_2) (fig. 2.19 (g));
- Output: Two points A_1 and K (fig. 2.19 (d)).

Exercise 2.12: Prove that $|A_1, K| = \frac{a_1}{a_2}$. Hint: use the triangle similarity properties.

Thus we can construct any positive natural $n \in \mathbb{N}$ and rational number $a \in \mathbb{Q}$ using the above algorithms recursively.

A natural question arises: Can we construct irrational numbers using only ruler and a compass?
The first irrational number that can be constructed using Pithagora's theorem is $\sqrt{2}$.

Exercise 2.13: Write an algorithm $S(A, B)$ with $|AB| = 1$ that constructs $\sqrt{2}$. Draw the diagrams for each step of the algorithm (similar to the algorithms above).
Prove its correctness and count the number of steps.

Exercise 2.14: Write an algorithm $S'(A, B)$ with $|AB| = \sqrt{a}$ that constructs $\sqrt{a+1}$. Draw the diagrams for each step of the algorithm (similar to the algorithms above).
Prove its correctness and count the number of steps.

Obviously, the following algorithm $H(n)$ gives two points with the distance beetween them equal to $\sqrt{n},\ n \in \mathbb{N}$:

Algorithm $H(n)$:

- If $n = 1$, Output: A, B with $|A, B| = 1$ End of the algorithm;
- If $n > 1$:

  Execute $S(H(n-1))$.

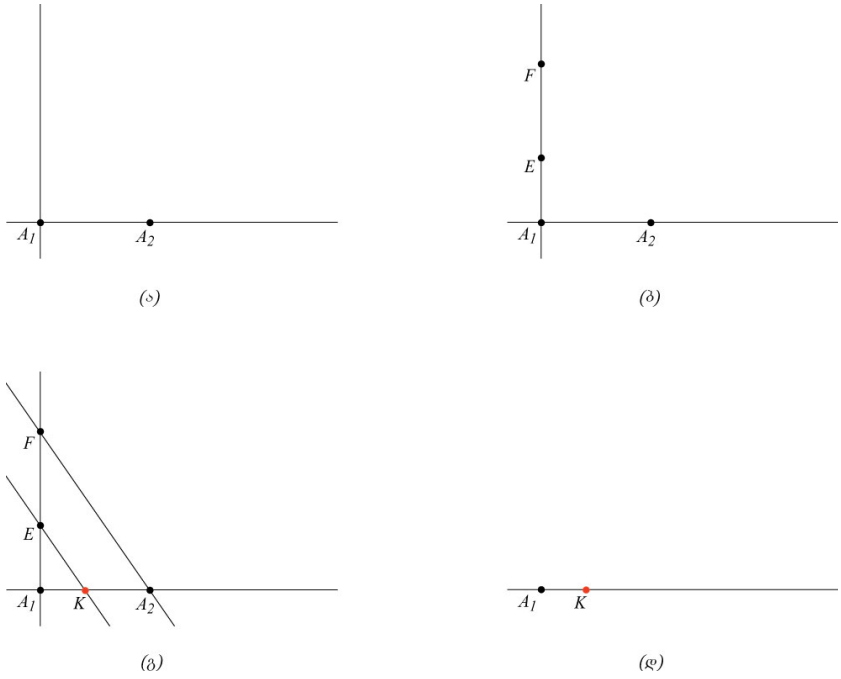


Figure 2.19: Construction of $\frac{|A_1,A_2|}{|A_1,F|}$

Exercise 2.15: Using the above algorithms, construct an algorithm that computes $\sqrt{a}$ for any $a \in \mathbb{Q}^+$.

Now consider the following algorithm:

Input: A_1, A_2 with $|A_1, A_2| = \xi$ (fig. 2.20 i(a)).

- On the left-hand side of A_1 mark a new point B on (A_1, A_2) with $|B, A_1| = 1$ (fig. 2.20 (b));
- Construct a circle with diameter $[B, A_2]$ (fig. 2.20 i(g));
- Construct a perpendicular line to (A_1, A_2) on A_1 (fig. 2.20 (d));

  Result: An intersection point P of this line and the circle (fig. 2.20 (d));
- Output: A_1 and P.

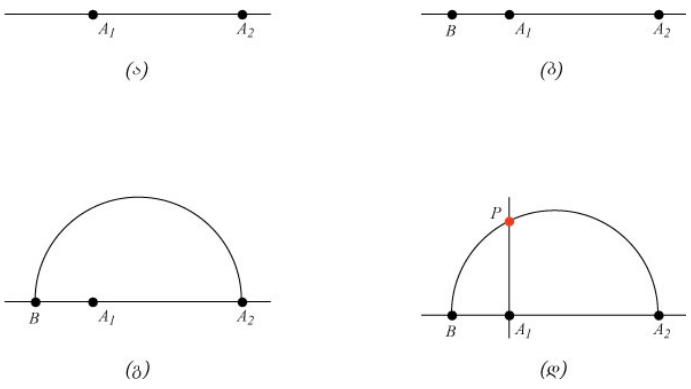


Figure 2.20: r$\sqrt{|A_1, A_2|}$ sigrZis monakveTis agebis procesi

Exercise 2.16: Prove that $|A_1, P| = \sqrt{|A_1, A_2|} = \sqrt{\xi}$. Hint: use the triangle similarities properties.

Exercise 2.17: Given two points A and B, how can we construct a circle with the diameter $[A, B]$?

Exercise 2.18: Consider three points O, A and B. The rays $[O, A)$ and $[O, B)$ from O form an angle $\angle\, AOB = \alpha$. Construct an algorithm that, given the points O, A and B, computes the points O, A, C such that $\angle\, AOC = \frac{\alpha}{2}$.

Now we can state some ancient problems that remained unsolved for over to thousand years. They were very important in practical engineering (e.g. huge building construction) already known to the Babylonians:

- **Squaring the circle:** Consider a circle with centre O and radius 1. *How* can we construct a square with the same area as the above circle?

  A fundamental fact in geometry (known to the ancient Egyptians) says that the area of a circle is proportional to the sqare of its diameter, and in ancient Greece it was proved to be equal to π (the ratio of the length of a circle to its diameter).

- **Cube root:** Consider two points with a distance a between them. *How* can we construct two points with distance $\sqrt[3]{a}$ between them?

- **Angle trisection:** Given three points O, A and B and two rays $[O, A)$ and $[O, B)$ starting from O, and creating romelic O wertilSi qmnis kuTxes $\angle\, AOB = \alpha$. *How* can we construct a point C such that $\angle\, AOC = \frac{\alpha}{3}$?

- **Regular n-Sided Polygons:** Given a number $n \in \mathbb{N}$, *how* can we construct an n-Sided Polygon with equal sides?

The problems above remained unsolved for more than two millennia until the first three were proved to be unsolvable. The last was solved by the famous Gauß formula.
Here we sketch the main ideas of the proofs.

A new point can be constructed only as an intersection of two already constructed lines (solution of a first-order equations), or two already constructed circles (solution of a second-order equations) or one already constructed line and already constructed circle (solution of a second-order equations). Hence (using the composition method) the coordinates of a new point should be the solutions of power-of-two-order polinomial equations with rational coefficients: $a_{2^n} x^{2^n} + a_{2^n-1} x^{2^n-1} + \cdots + a_1 x + a_0 = 0$, $n \in \mathbb{N}$, $a_i \in \mathbb{Q}$.
Since $\sqrt[3]{a}$ is not a solution of this kind of equation, this construction should be impossible with compass and ruler.

As shown by German mathematician Ferdinand von Lindemann in the 19th senctury, π is a transcendental number, meaning that it is not a solution of any polynomial equation with rational coefficients. It can be expressed only by an endless series (endless sum – endless polynomial). Since the construction process of a point with compass and ruler should be completed in finite time, its coordinates should be expressed by finite polynomial equation with rational coefficients. This completes the proof that no two points with the distance π can be constructed. Since a square with area of a unite circle (with radius 1) should have a side of the length $\sqrt{\pi}$, and π is not constructable, this kind of square should not be constructable using only compass and ruler.

At the end of the 18th century, the 18 year old Karl Friedrich Gaußdiscovered a method to construct a regular 17-gon (heptadecagon) thus solving a problem that remained open for over 2000 years. The general formula for constructable regular polygons is:
An n-gon is constructable iff (if and only if) $\exists m, q_1, ..., q_l \in \mathbb{N}_0$ such that

$$n = 2^m \cdot (2^{2^{q_1}} + 1) \cdot (2^{2^{q_2}} + 1) \cdots (2^{2^{q_l}} + 1).$$

Due to this formula we conclude that a pentagon, 17-gon, and a 65537-gon are all constructable, but a 7-gon is not constructable with ruler and compass.

The problem of polygon construction is connected to the very important problem of n-th roots of unity that is also applied in algorithm design (to be discussed in later courses).

Exercise 2.19: What is the method to construct a hexagon? An octagon?

Exercise 2.20: (Not-so-easy exercise) Give a method to construct a pentagon.

Note that for the ancient problems, no construction method exists *using only* a ruler and a compass. That **does not** mean that there do not exist construction methods using other resources. In the contrary to other problems we will discuss later and that turned out to be unsolvable with any tools, the unsolvability of the ancient problems are due to the limited resources.

Open problem: In the above formula, $2^{2^q} + 1$ is a so called Fermats prime number. It was widely believed for a long time that this formula delivers only the primes. But this belief was disapproved. In fact, it mostly delivers combined numbers.
An important open question is wether this formula delivers an infinite amount of primes.

2.4 Summary

In this chapter we introduced the concepts of the so called recursive and iterative algorithms. In the recursive description, the algorithm recalls itself with lower parameters, in the iterative one it repeats a sequence of instructions several times building a so-called cycle.
Both methods have their advantages ind dissadvantages depending on the special problem and its realisation needs to be discussed in details later.
As a nice example of the recursive algorithms we considered ancient problems of ruler-cpmpass constructions that remained unsolved for over 2000 years until their solutions with modern mathematical techniques in the 19th century.
These problems are also interested in the computational sense. Here we see the unsolvability concept, however only with the restriction of resources. With other tools it may be solvable.
In general, there exist some problems that can not be solved in finite time using any resources, but we will discuss this kind of problems later.

Chapter 3

Mathematical Induction and its Applications

3.1 Mathematical Induction

Consider a sequence of all odd numbers:

$a_1 = 1, a_2 = 3, a_3 = 5, a_4 = 7, ...$
Obviously, its elements can be described as $a_i = 2 \cdot i - 1$.
Now let us calculate the sum of the first n elements:

$$S_n = a_1 + a_2 + a_3 + \cdots + a_n.$$

Recursively, it can be desribed as

$$S_n = S_{n-1} + a_n$$

(first $n-1$ odd numbers plus the n-th odd number).

Exercise 3.1: Give the recursive descriptions for $S_{n+1}, S_{n-1}, S_{n-2}$ and S_{n-3}.
First let us calculate few elements:

$$\begin{array}{lll} S_1 = & a_1 & = 1 \\ S_2 = & a_1 + a_2 & = 4 \\ S_3 = & a_1 + a_2 + a_3 & = 9 \\ S_4 = & a_1 + a_2 + a_3 + a_4 & = 16 \\ S_5 = & a_1 + a_2 + a_3 + a_4 + a_5 & = 25 \\ \cdots & & \end{array}$$

Examining the right side of the equations we can see some regularity: the sums build squares of natural numbers: $S_1 = 1^2, S_2 = 2^2, S_3 = 3^2, S_4 = 4^2, S_5 = 5^2$.
This gives us the first clue about the general rule of the sums: $S_i = i^2$.
Note that this is only a clue, the observation of some experimental results. Even checking the hypothesis for millions of cases will only strengthen the proposition. Mathematically, we need a formal proof for every n.
To accomplish such a proof, a special technique called **"mathematical induction"** is needed. It can be fulfilled in three steps:

1. Verification: Verify the conjecture for $n = 1$;
2. Assumption: Assume the conjecture is true $\forall k = 1, 2, ..., n$;
3. Induction step: Prove the conjecture for $n + 1$.

After thes steps we obtain: If the conjecture is true for $n = 1$, assume $n = 1$ in the second step of the above process; If it is proved to be true for $n + 1$ (step 3), then it should be true for $n = 2$; now assume $n = 2$ in the second step and we can prove that it is true for $2 + 1 = 3$ and so on.

In our example above we have:

1. Verification: $n = 1$: $S_1 = 1^2 = 1$;
2. Assumption: Assume $S_n = n^2$;
3. Induction step: Show $S_{n+1} = (n + 1)^2$.
   According to the recursive formula, $S_{n+1} = S_n + a_{n+1} = S_n + 2 \cdot (n + 1) - 1 = S_n + 2 \cdot n + 1$.
   According to the assumption, $S_n = n^2$ and we have: $S_{n+1} = n^2 + 2 \cdot n + 1 = (n + 1)^2$
   that proves the assumption for every n.

If the recursive formula is written in non-recursive form (with only constants and variables), we say that the recursion has been solved.

Using the above principles we will prove one more mathematical conjecture:

A sequence is given recursively as $S_1 = 1, S_n = S_{n-1} + n$. Prove that $S_n = \frac{n\cdot(n+1)}{2}$.

1. Verification: $n = 1$: $S_1 = \frac{1\cdot(1+1)}{2} = 1$;
2. Assumption: $S_n = \frac{n\cdot(n+1)}{2}$;
3. Induction step: Prove $S_{n+1} = \frac{(n+1)\cdot(n+2)}{2}$.
   Since $S_{n+1} = S_n + (n + 1)$, due to the assumption, $S_{n+1} = \frac{n\cdot(n+1)}{2} + (n + 1) = (n + 1)(\frac{n}{2} + 1) = \frac{(n+1)\cdot(n+2)}{2}$.

Exercise 3.2: Using mathematical induction prove that every odd number can be described as $a_i = 2 \cdot i - 1$.

Exercise 3.3: Solve the following recursion: $S_1 = 3$, $S_n = S_{n-1} + n$.

Exercise 3.4: Solve the following recursion: $K_1 = 7$, $K_n = K_{n-1} + 2n$.

Exercise 3.5: Solve the following recursion: $P_1 = 1$, $P_n = P_{n-1} + 2^n$.

Exercise 3.6: Solve the following recursion: $L_1 = 7$, $L_n = 2 \cdot L_{n-1}$.

3.2 Applications: Correctness and Complexity

Consider the recursive description of the boats problem $A_n = A_1, U, A_{n-1}$ from the previous chapter. We can prove its correctness using mathematical induction as follows:

- Verification: Obviously, A_1 is correct;
- Assumption: A_n correctly passes n boats (provided the initial configuration is as desired);
- Induction step: Show that $A_{n+1} = A_1, U, A_n$ is correct.

If we can prove the correctness of A_{n+1} (provided that A_n is correct), knowing that A_1 is correct, A_2 will be correct. Since A_2 is correct, according to the third step of the mathematical induction, A_3 will be proved to be correct etc. Hence, the assumption will be correct for every natural n.

To prove the correctness of $A_{n+1} = A_1, U, A_n$, consider the configuration after executing the steps A_1, U. Obviously, it is the proper input for the n-boats problem and can be correctly solved (assumption) by A_n, meaning that A_1, U, A_n will solve the $n + 1$ boats problem correctly.

Q.E.D.

Exercise 3.7: Can we solve the n-boats problem by $A_n = A_{n-1}, U, A_1$?

After the prove of correctness of a given algorithm A, we should estimate the number of steps $T(A_n)$.
First of all, A_1 has to be executed, then U and at last A_{n-1}, hence $T(A_n) = T(A_1) + T(U) + T(A_{n-1})$ (as many as needed for A_1, then as many as needed for U and at last as many as needed for A_{n-1}).
This recursive formula can be solvedas follows:

We know that $T(A_1) = 3$ amd $T(U) = 1$ (by a simple check) and we get:

$$T(A_n) = T(A_{n-1}) + 4.$$

By its own, $T(A_{n-1}) = T(A_{n-2}) + 4$, $T(A_{n-2}) = T(A_{n-3}) + 4$ ...

Hemce,

$$\begin{aligned} T(A_n) = &\quad T(A_{n-1}) + 1 \cdot 4 = T(A_{n-2}) + 2 \cdot 4 = T(A_{n-3}) + 3 \cdot 4 = \cdots = \\ = &\quad T(A_1) + (n-1) \cdot 4 = 3 + (n-1) \cdot 4 = 4 \cdot n - 1. \end{aligned}$$

Exercise 3.8: Prove that no faster algorithm can exist for this problem.

By similar considerations, we can prove the correctness and estimate the number of steps of the Hanoi Towers algorithm

$$H_n^{X_1,X_2} = H_{n-1}^{X_1,X_3}, H_1^{X_1,X_2}, H_{n-1}^{X_3,X_2}.$$

- Verification: $H_1^{X_1,X_2}$ is correct (obvious);
- Assumption: Assume that $H_k^{X_1,X_2}$ is correct for all $k \leq n \in \mathbb{N}$;
- Induction Step: Prove that $H_{n+1}^{X_1,X_2} = H_n^{X_1,X_3}, H_1^{X_1,X_2}, H_n^{X_3,X_2}$ is correct (assumed H_n is).

  First, the algorithm $H_n^{X_1,X_3}$ moves the upper n disks from X_1 to X_3 (fig. 3.1(a)). According to the assumption, this procedure will be completed correctly (note that during this process, the largest disk remains on X_1 and any other disk can be moved upon it thus not violating the restrictions). After this, we move the largest disk to X_2 (fig. 3.1(b)) creating the initial configuration for $H_n^{X_3,X_2}$. applying it we move the remaining n disks from X_3 to X_2 (fig. 3.1(g)). As in the steps above, due to the assumption, $H_n^{X_3,X_2}$ will move the disks correctly without violating any restriction (since the largest disk is already on X_2, it can be ignored). As a result we obtain all the complete $n+1$ pyramid moved (fig. 3.1(d)).
  In the diagram below we have $X_1 = A, X_2 = C, X_3 = B$.

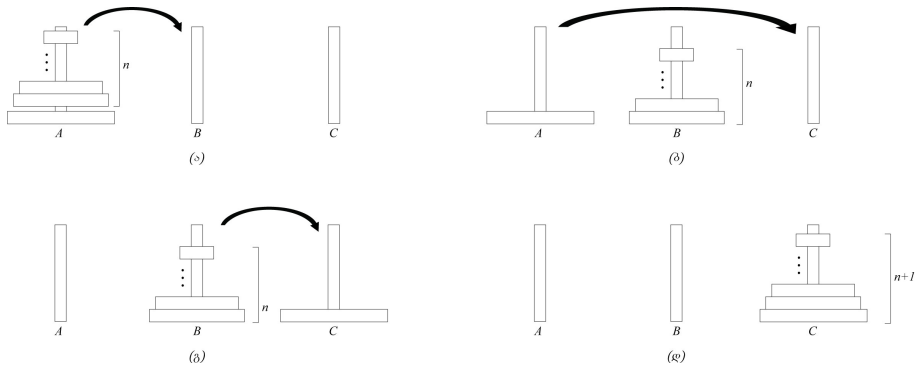


Figure 3.1: Operations needed to move $n+1$ disks

To estimate the number of needed steps, consider the recursive description

$H_n^{X_1,X_2} = H_{n-1}^{X_1,X_3}, H_1^{X_1,X_2}, H_{n-1}^{X_3,X_2}$.

Obviously,

$T(H_n^{X_1,X_2}) = T(H_{n-1}^{X_1,X_3}) + T(H_1^{X_1,X_2}) + T(H_{n-1}^{X_3,X_2})$.

Exercise 3.9: Explain the meaning of $T(H_{n+3}^{A,C})$ and $T(H_3^{C,B})$ and $T(H_7^{A,C})$.
Exercise 3.10: What are $T(H_1^{A,C})$ and $T(H_2^{A,C})$ equal to?
Exercise 3.11: Prove that $T(H_1^{A,C}) = T(H_1^{B,C})$ and, in general, $T(H_n^{X_1,X_2}) = T(H_n^{Y_1,Y_2})\ \forall X_1, X_2, Y_1, Y_2 \in \{A, B, C\}$ (no matter from where to where we move the disks – the number of steps remains the same).

Since $H_n^{X_1,X_2} = H_{n-1}^{X_1,X_3}, H_1^{X_1,X_2}, H_{n-1}^{X_3,X_2}$, first $H_{n-1}^{X_1,X_3}$ should be executed, after it $H_1^{X_1,X_2}$ and, at the end, $H_{n-1}^{X_3,X_2}$. Hence

$$T(H_n^{X_1,X_2}) = T(H_{n-1}^{X_1,X_3}) + T(H_1^{X_1,X_2}) + T(H_{n-1}^{X_3,X_2}) = 2 \cdot T(H_{n-1}^{X_1,X_2}) + 1$$

(see the above exercises).

Exercise 3.12: Prove by mathematical induction:

$$T(H_n^{X_1,X_2}) = 2^n - 1.$$

Exercise 3.13: Consider the n-boats problem. Prove its correctness and estimate its number of steps.

Exercise 3.14: Consider the iterative algorithm for the n-disk Hanoi Towers problem and estimate its number of steps.
Note that the above exercise is not easy because it is not clear when the cycle terminates.

This example shows us the negative sides of the iterative description of algorithms. In contrary to recursion, neither the proof of correctness nor the estimation of steps is easy.
But, as we will see in short, there exist examples where the recursive description leads to specific problems. Hence one of the main questions in algorithm design is to determine the correct description and data structures to make them efficient.
One of such problems is the computation of the so called Fibonacci sequence that we will discuss now.

3.3 The Fibonacci Sequence

The famous Italian mathematician of the 12th - 13th cenctury Leonardo da Pisa, better known as Fibonacci, tried to solve a very practical problem:

A peasant has rabbits that win offspring – exactly one doe a month after two months of age. How many does will he have starting with one newborn and assuming that the rabbits do not die?

For small n the computation is easy: The first two months only one, then they become 2, then 3 (the offspring of the eldest, the second is too young), then 5 (offspring of first two, the third too young) etc. In the n-th month the number of offsprings should be equal to the number of the rabbits that are at least 2 months of age. Hence, denoting the number of rabbits by F_n, we obtain

$$F_1 = F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

It is also called the Fibonacci Sequence.

We know also that $F_1 = 1, F_2 = 1, F_3 = 3, F_4 = 5$. For some technical reasons, the first element of this sequence is sometimes defined as $F_0 = 0$ thus describing the sequence as

$$F_0 = 0;\ F_1 = 1;\ F_n = F_{n-1} + F_{n-2}$$

for $n > 1$. The first members of this sequence are:

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,$
$10946, 17711, 28657, 46368, \ldots$

The natural numbers computed by the Fibonacci sequence are called the Fibonacci numbers, F_n and F_{n+1} are the neighbouring numbers.

An interesting fact is that this sequence was also known in ancient Greece and the middle-age India. Some authors also consider it with $F_1 = 1$ and $F_2 = 2$.
As it turned out, this sequence does by no means calculate the number of rabbits but turned out to have many very useful properties so that the Fibonacci sequence has many applications in the wide fields of modern theoretical and practical problem solving.
This fact is also showing how important is to analyse the *methods* how a problem is tackled. It can totally fail to solve the initial problem but can open wide horizons and perspectives.
The fibonacci sequence is also very common in nature. For example, the seeds of a sunflower are positioned in rounded curves in a pattern shown in fig. 3.2.

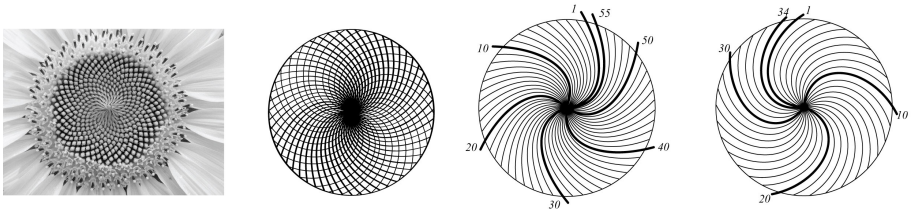


Figure 3.2: Pattern of seeds in a sunflower

As one can see, the seedings can be considered in two different ways each of them building 55 and 34 curves, two neighbouring Fibonacci numbers.
Besides this, the number of tree branching or that of the petals are mostly Fibonacci numbers (there are many flowers with 3, 5, 8, or 13 petals, but exeptionally with 4, 6, 7 or 9).

It is proved that for any $n \in \mathbb{N}$, there is an explicit (one and only one) sequence of non-neighbouring Fibonacci numbers $F_{i_1}, F_{i_2}, ..., F_{i_k}$ that sum up as n: $n = F_{i_1} + F_{i_2} + ... + F_{i_k}$.
For example, $n = 67$ can be represented as $67 = 1 + 3 + 8 + 55 = F_1 + F_4 + F_6 + F_{10}$ and there exists no other such representation (of course, we have $67 = 1 + 3 + 8 + 21 + 34$, but here are 21 and 34 the neighbouring numbers that contradicts the statement).
Such an explicit representation creates the so-called Fibonacci code for each natural number that is widely used in the modern theory of information.

One of the most important open problem of the 20st century, the so-called Hilbert's 10th problem on the solution of Diophantine equations, has been solved using the Fibonacci sequence. It is interesting to know that this problem has initiated the development of the modern theory of algorithms: to solve it, many novel techniques and notions has been introduced that are also used nowadays.
Worth to mention that this problem turned out to be *unsolvable*.

The Fibonacci sequence has also many applications in combinatorics.
Consider the following question: In how many different ways can we climb the n-step stairs if we can go one or two steps at once?
Obviously, if $n = 1$, there is only one way to climb. If $n = 2$, two ways are possible: twice one step or two steps at once. In case of $n = 3$, three different ways are possible: 1+1+1 or 1+2 or 2+1. $n = 4$: 1+1+1+1 or 1+1+2 or 1+2+1 or 2+1+1 or 2+2, 5 possibilities in total.
If we denote the number of disfferent ways to climb an n-step stair by G_n, G_{n+1} can be calculated by the following argument: Given an $n + 1$ step stairs, we can promote by 1 step and then climb the n step stairs (by as many different ways as possible) or we can promote by two steps at once and then run up the $n - 1$ step stairs by as many methods as possible. In total, we get $G_{n+1} = G_n + G_{n-1}$ different possibilities that coincides exactly with the Fibonacci sequence. In general,

$$G_n = F_{n+1}.$$

As a second application consider the problem of covering of a chessboard by domino stones. Given a $2 \times n$ chessboard and n domino stones, each covering exactly two neighbouring fields of the chessboard. In how many different ways can we cover a $2 \times n$ chessboard with n domino stones?

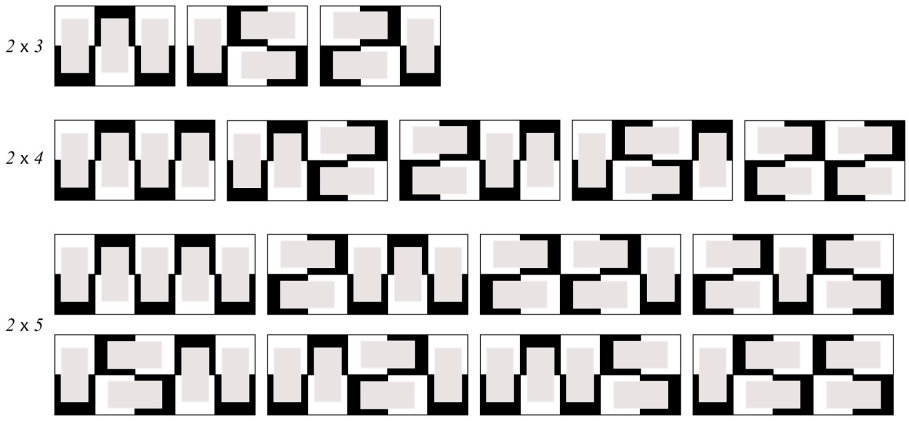


Figure 3.3: Covering a small chessboard by the domino stones

Exercise 3.15: Denote the number of ways we can cover a $2 \times n$ board by P_n (in the examples above, $P_3 = 3$, $P_4 = 5$ da $P_5 = 8$).
What is the recursive description of P_n? How is it related to the Fibonacci sequence?

The Fibonacci sequence has following interesting properties (to mention only few):

- $F_1 + F_2 + \cdots + F_n = F_{n+2} - 1$;
- F_{3n} is even;
- F_{5n} is dividible by 5;
- $F_1 + F_3 + F_5 + \cdots + F_{2n-1} = F_{2n}$;
- $F_0 - F_1 + F_2 - F_3 + \cdots - F_{2n-1} + F_{2n} = F_{2n-1} - 1$;
- $F_1^2 + F_2^2 + \cdots + F_n^2 = F_n \cdot F_{n+1}$;
- $F_{n-1} \cdot F_{n+1} - F_n^2 = (-1)^n$;

Exercise 3.16: Prove the above equations.

Much more harder to prove are the following properties:

- If $n > 4$ and F_n is prime, then n is prime (the inverse is not true: $\exists p$ prime so that F_p is not prime);
- If $n, m \in \mathbb{N}$ and $gcd(m, n)$ is their greatest common divider, then $gcd(F_m, F_n) = F_{gcd(m,n)}$
- $F_{n+m} = F_{n-1}F_m + F_nF_{m+1}$;
- $F_{(k+1)n} = F_{n-1}F_{kn} + F_nF_{kn+1}$;
- $F_n = F_lF_{n-l+1} + F_{l-1}F_{n-1}$;
- $F_n = F^2_{(n+1)/2} + F^2_{(n-1)/2}$, if n is odd;
- $F_n = F^2_{n/2+1} + F^2_{n/2-1}$, if n is even.

Open problem: Are there infinitely many primes in the Fibonacci sequence?

Exercise 3.17: What is $gcd(46368, 21)$?

Exercise 3.18: Write an algorithm that determines if its input is a fibonacci number.

A natural question is: How many steps are needed to compute F_n?

Exercise 3.19: Compute $T(F_n)$ (number of steps – arithmetical operations – needed to compute F_n by a recursive formula).
Now consider the iterative version of the Fibonacci algorithm that repeats a sequence of operations in a cycle:

Algorithm 3.1**:** Fibonacci Numbers (iterative version)
Input: $n \in \mathbb{N}$

---

1 r Repeat n-times: { $c = a + b$; $b = a$; $a = c$; }
End of the Algorithm

Exercise 3.20: Prove that after the execution of this algorithm, $a = F_n$. What will be the values of the variables c and b?

Exercise 3.21: Prove that the above algorithm computes F_n in $4n + 2$ steps (consider the addition and assigning as one step).

Due to this algorithm we can see the clear advantage of the iterative method compared to the recursive one.

An interesting question is, what caused such a great difference in the computation efficiency?
To analyze this question, consider the graphical representation of the computational process of the recursive algorithm:

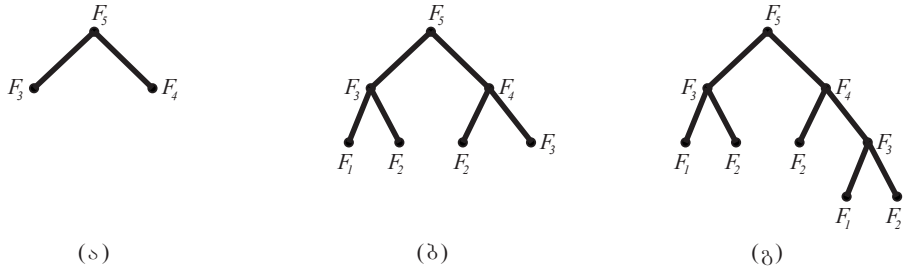


Figure 3.4: Computation of F_5

Each point in the diagram represents the corresponding Fibonacci number that is connected by a line segment with two lower points representing numbers necessary to compute it. Obviously, to compute F_5, the recursive algorithm computes F_3 twice. On its own, F_3 consists of the sum of two other numbers, that means that several computations are *redundant*: it would be possible to compute F_3 only once and to store the result somewhere in order to use it later in case of need. That would avoid redundant computations.
This example shows the negative sides of the "blind" recursion: in frequent cases the algorithm proceeds with the computations that could be avoided.
This was the main reson for the developers of programming languages in the 1960's and 1970's to abandon recursion. Later, after the development of compiler optimisers, this problem has been solved and the recursive functions play a central role in modern programming theory and practice due to their positive features.

Exercise 3.22: Draw the computation diagrams of F_6 and F_7 as shown in the above example. How many redundant steps are proceeded in each case?

Naturally, it would be much more convenient and fast to compute the Fibonacci numbers by a non-recursive formula. In fact, such a formula can be represented as

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right).$$

Exercise 3.23: Prove the correctness of this formula using mathematical induction.
A natural question would be: how could anibody deduce this kind of formula?
The most plausible way could be the induction: observation of the sequence, experimenting with its members and deducing some regularities. At a glance, no regularity can be discovered except the defining equation $F_n = F_{n-1} + F_{n-2}$:

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,$
$10946, 17711, 28657, 46368, \ldots$

But if we analyse the fractions of the neighbouring numbers, a sort of regularity can be observed:

$T_n = \frac{F_n}{F_{n-1}} \quad (n > 1)$:

$T_2 = \frac{1}{1} = 1;$ $T_3 = \frac{2}{1} = 2;$ $T_4 = \frac{3}{2} = 1,5;$ $T_5 = \frac{5}{3} \approx 2,666667;$

$T_6 = \frac{8}{5} = 1,6;$ $T_7 = \frac{13}{8} = 1,625;$ $T_8 = \frac{21}{13} \approx 1,615;$ $T_9 = \frac{34}{21} \approx 1,619;$

$T_{10} = \frac{55}{34} \approx 1,6176;$ $T_{11} = \frac{89}{55} \approx 1,618;$ $T_{12} = \frac{144}{89} \approx 1,61798;$ $T_{13} = \frac{233}{144} \approx 1,61805;$

$T_{14} = \frac{377}{233} \approx 1,618026;$ $T_{15} = \frac{610}{377} \approx 1,618037;$ $T_{16} = \frac{987}{610} \approx 1,618033;$ $T_{17} = \frac{1597}{987} \approx 1,618034;$

$T_{18} = \frac{2584}{1597} \approx 1,618034;$ $T_{19} = \frac{4181}{2584} \approx 1,618034;$ $T_{20} = \frac{6765}{4181} \approx 1,6180339887;$ $T_{21} = \frac{10946}{6765} \approx 1,6180339887;$

$T_{22} = \frac{17711}{10946} \approx 1,6180339887;$ $T_{23} = \frac{28657}{17711} \approx 1,6180339887;$ $T_{24} = \frac{46368}{28657} \approx 1,6180339887...$

The sequence T_n (the fracton of the neighbouring numbers) converges to some fixed number (meaning that the difference between the Fibonacci numbers and this specific number is getting smaller and smaller and tends to go closer to zero as we observe higher Fibonacci numbers). In fact, it is proved that

$$\lim_{n\to\infty} \frac{F_n}{F_{n-1}} = \Phi \approx 1,6180339887.$$

Here $\Phi = \frac{1+\sqrt{5}}{2}$ is the so called "Golden ratio" already known to the ancient Greeks (see the exercise to construct a pentagon with compass and ruler).

In our case that means that the starting from some point, the Fibonacci sequence behaves very similarly to a geometric progression.
Hence we can suppose that there are more similar properties and there exist a sequence

$$G_n = c \cdot q^n,$$

that coincides with the Fibonacci sequence (at least from some point on) for some specific $c, q \in \mathbb{N}$. But how can we determine these numbers?

Of course, the sequence $(G_n)_{n=1}^{\infty}$ itself should satisfy the Fibonacci condition

$$G_n = G_{n-1} + G_{n-2}$$

and

$$c \cdot q^n = c \cdot q^{n-1} + c \cdot q^{n-2}$$

Dividing both sides of the equation by $c \cdot q^{n-2}$, we obtain an equation due to which we can define q:

$$q^2 = q + 1.$$

Its solutions are

$$q_1 = \frac{1+\sqrt{5}}{2} \text{r} \quad \text{da} \quad q_2 = \frac{1-\sqrt{5}}{2}.$$

As a result, we obtain two sequences

$$G'_n = c \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n \quad \text{and} \quad G''_n = c \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n,$$

both of them satisfying the Fibonacci condition.
All we have to do is to choose one of them or some combination of both and to determine the parameter c so that the resulting sequence coincides the Fibonacci sequence $(F_n)_{n=1}^{\infty}$.

Consider the sequence G'_n. If $n = 1$ then $G'_1 = c \cdot \frac{1+\sqrt{5}}{2}$ and because we want $G'_1 = F_1 = 1$,

$$c \cdot \frac{1+\sqrt{5}}{2} = 1.$$

Hence $c = \frac{2}{1+\sqrt{5}}$. But in this case $G'_2 = \frac{2}{1+\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^2 \neq F_2 = 1$ meaning that $(G'_n)_{n=1}^{\infty}$ by itself can not coincide to the Fibonacci sequence.

Exercise 3.24: Prove by similar arguments that $(G''_n)_{n=1}^{\infty}$ does not coincide the Fibonacci sequence as well.

Now consider the sequence

$$H_n = G'_n - G''_n = c \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

Exercise 3.25: Prove that the above sequence $(H_n)_{n=1}^{\infty}$ satisfies the Fibonacci condition.

Exercise 3.26: Consider $H'_n = G'_n + G''_n$. Does it satisfy the Fibonacci condition?

Obviously, $H_0 = 0$ and the starting elements coinside. Now consider H_1:

$$H_1 = c \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right).$$

Since it shoul coincide with the first element of the Fibonacci sequence, we solve the following equation:

$$H_1 = c \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right) = 1$$

and obtain $q = \frac{1}{\sqrt{5}}$.
Hence

$$H_1 = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right).$$

Since the sequence $(H_n)_{n=1}^{\infty}$ satisfies the Fibonacci condition and its first two members also coincide with that of Fibonaccis, they should coinside comletely.:

$$H_n = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

Q. E. D.

Exercise 3.27: Consider $H'_n = G'_n + G''_n = c' \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$.
Can we choose c' so that the sequence H' coincides with the Fibonacci sequence?

Analysing the process of deducing the non-recursive formula, several fundamental moments should be emphasized:

- First of all, we "carry out" an experiment: like physicists observe natural events, biologists – living organisms, mineralogists – minerals, we observe the sequence of numbers and their relations and try to find some regularities in it. This observation and deduction process is called **induction** not to be confused with the notion of mathematical induction above. In "experiments" we mean exactly the test of different relations – the difference of neighbours, their sum, multiplications, fractions etc.;

- During one of such experiments we discovered certain regularity: the fraction of neighbouring Fibonacci numbers tend to go closer to a particular number (converge to this number);
- This regularity gave us the hint to make some *analogy:* Since the fraction converges to some particular number – like any geometric progression does – it is possible for this sequence to have also other similar properties;
- Due to this, we have a new object to analyse: a geometric progression with Fibonacci property.

The above explicit formula has been deduced after the closer look, investigation and analysis of this kind of sequence.

In general, the idea of science can be formulated similarly: observing the objects, experimenting with them, discovering regularities and searching their reasons.
Our object of investigation was the sequence of numbers, and the discovered regularity – its analogy to geometric progressions. The reason of this regularity could be that our initial sequence (the Fibonacci sequence) in its core is the analogon of geometric progressions.

3.4 Pascal's Triangle

The so called Pascal's triangle is a triangular list of natural numbers where each number is the sum of its two upper neighbours, starting with 1:

														1																
												1				1														
										1				2				1												
								1				3				3				1										
						1				4				6				4				1								
					1			5				10				10				5				1					
				1				6				15				20				15				6				1	
			1			7				21				35				35				21				7			1
		1			8			28				56				70				56				28			8			1
	1			9			36			84				126				126				84			36			9			1
1			10			45			120			210				252				210			120			45			10			1

For example, the second element of the fourth raw is equal to 3 because its upper neighbours (the first and second element of the third raw) are 1 and 2.
The Pascal's triangle has wide range of applications in theory and practice. However known already in the ancient Greece and the middle-age Iran, its applications are related to the French mathematician Blaise Pascal thus its name used in Europe (on the contrary, in the Middle-Eastern world it is known as the triangle of the famous Iranian poet and philosopher Omar Khayyam).

What is the use of the Pascal's Triangle?
As its widest application, the computation of the n-th power of the sum of two numbers can be considered:

$$(x+y)^n = a_{n,0}x^n \cdot y^0 + a_{n,1}x^{n-1} \cdot y^1 + a_{n,2}x^{n-2} \cdot y^2 + \cdots + a_{n,n-1}x^1 \cdot y^{n-1} + a_{n,n}x^0 \cdot y^n$$

For any fixed n, we get a so called polynome of two variables of power n (a sum of $n+1$ elements where the sum of the powers of the variables is constantly equal n).
For example,

$(a+b)^0 = 1$,
$(a+b)^1 = a+b = 1a^1b^0 + 1a^0b^1$,
$(a+b)^2 = a^2 + 2ab + b^2 = 1a^2b^0 + 2a^1b^1 + 1a^0b^2$,
$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b3 = 1a^3b^0 + 3a^2b + 3ab^2 + 1a^0b^3$

Observing the above polynoms we can see that their coefficients are exactly the rows of the Pascal's triangle.
As it turned out, this is always the case: the coefficients of $(a+b)^n$ coincide to the $n+1$st row of the Pascal's triangle:

$$a_{i,j} = P_{i+1,j},$$

where $P_{k,l}$ is the l-th element in the k-th row.

Moreover, each element $P_{i,j}$ of the Pascal's triangle is the so called binomial coefficient:

$$P_{i,j} = \begin{pmatrix} i-1 \\ j-1 \end{pmatrix}$$

The number $C_p^k \equiv \begin{pmatrix} k \\ p \end{pmatrix}$ denotes the number of different choices of p elements from a k element set. As an example, if $k = 4$ and $p = 2$, consider a 4-element set $\{a_1, a_2, a_3, a_4\}$ (the names of the elements are immaterial). Its 2-element choices are:

$\{a_1, a_2\}, \{a_1, a_3\}, \{a_1, a_4\}, \{a_2, a_3\}, \{a_2, a_4\}, \{a_3, a_4\}$, 6 in total: $C_p^k \equiv \begin{pmatrix} k \\ p \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} = 6.$

This are the topics of Combinatorics we will discuss later.

Exercise 3.28: Write a recursive formula to compute $P_{n,m}$.

Note that there exists a general formula $C_p^k = \frac{k!}{p!(k-p)!}$. Its meaning and the way how it can be deduced will be discussed later.

Exercise 3.29: Write a recursive algorithm that computes $P_{n,m}$. Hint: Use the formula deduced in the previous exercise.

3.5 Summary

In this chapter we discussed the principle of the mathematical induction that is widely used in the proof of the properties of recursive and ordered structures. This principle has been applied to prove the correctness and estimate the number of computational steps of recursive algorithms.
Besides this, we have introduced and analysed two very important sequences: The Fibonacci sequence and Pascal's triangle, showing basic ideas of the applications of mathematical induction techniques to deduce general rules and regularities.
Analysing the iterative and recursive algorithms of Fibonacci numbers, we have shown the main differences between these two approaches, the main weaknesses of recursion and their reasons.

Chapter 4

Sets and their Cardinality

4.1 Bijection and Countable Sets

The Little Prince owns a herd of black and white sheeps. Mathematically, we can consider it as a *finite* set (fig. 4.1 left). The Little Prince wants to determine which kind of sheeps more: white or black. To do this, he separates them in two *subherds:* the set is divided into two subsets (fig. 4.1 middle).

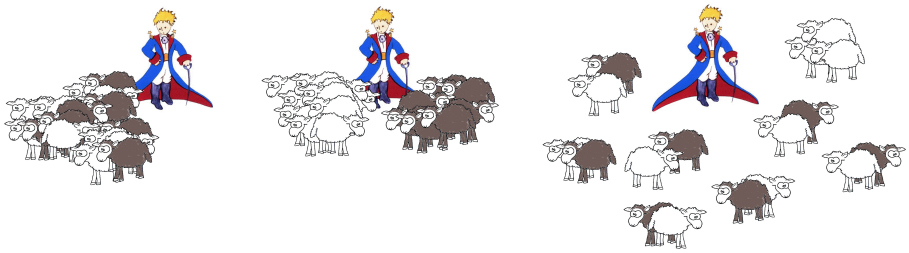


Figure 4.1: The Little Prince and his sheeps

The groups of white and black sheeps on their own can be considered as two sets. To determine which set consists of more elements, the Little prince *pairs* their elements with one another (one black and one white sheep) (fig. 4.1 right).
In other words, one element of one set *corresponds* to one element of the second set so that two different elements of the first set correspond to two different elements of the second (two elements do not correspond to one and the same – two different white seeps correspond to two different blacks). Since after this kind of pairing there are white sheeps left we can conclude that there are more white sheeps than black ones.

Obviously, this kind of pairing is always possible between two finite sets. Important is that *two different* elements of the first set are paired with *two different* elements of the second. If after this pairing neither set has unpaired elements left, they should have equal number of elements.

Such a pairing is called **Bijection**.

A natural question is whether or not is it possible to pair the elements of infinite sets?
Considering the set of natural numbers $\mathbb{N} = \{1, 2, 3, 4, ...\}$ and the set of even numbers $2\mathbb{N} = \{2, 4, 6, 8, ...\}$, such a pairing could be $(1 \leftrightarrow 2)$, $(2 \leftrightarrow 4)$, $(3 \leftrightarrow 6)$, $(4 \leftrightarrow 8)$, ... , $(n \leftrightarrow 2n)$, ...

Despite the fact that this process will last forever, we can definitely say that to each element of the set $\mathbb{N}$ corresponds exactly one element of the set $2\mathbb{N}$ no "redundant" element will be left in any set (by the same arguments we can conclude that to each element of $2\mathbb{N}$ corresponds an *explicit* element from $\mathbb{N}$. As mentiond, this kind of mapping is called a *bijection* (or on-by-one and onto).

In mathematical language it can be expressed as follows:
Definition 4.1: Consider a mapping $f : A \to B$ where A and B are some sets and to *each* element of A corresponds exactly one element from B.

Given any element $a \in A$, which is mapped to some $b \in B$ by the function f, formally denoted as $f : a \mapsto b$, then b is called the **value** of a and and a is called an **argument** of b.
The sets A and B are called the **domain** and **image** of the function f.

- A function $f : A \to B$ is called a *surjection* (or to be surjective) if to each element of B there exists an element of A so that a is mapped to b. Formally, $\forall b \in B, \exists a \in A, f(a) = b$
  (in other words, there does not exist an element in the image that has no argument – no element will be "missed";
- $f : A \to B$ is called **injection** if two different elements are mapped into two different elements. Formally, $\forall a \neq b, f(a) \neq f(b)$;
- $f : A \to B$ is a **bijection** if it is both bijective and surjective (one-by-one and onto)

To conclude, two (finite or infinite) sets have the same "amount" of elements if there exists a bijective mapping between them (it is possible to establish a one-by-one correspondence).
Note that it is not correct to speak about the amount of elements in an infinite set. Thats why it is said that two infinite sets have the same **cardinality** if there is a bijection between them.
If there is no bijection between A and B but there is a bijection from A to some subset of B it is said that the cardinality of A is more than the cardinality of B and alternatively that the cardinality of B is less than that of A.

It is worth to mention that we established a bijection between a set and its proper subset. This fact is always false for finite sets and can only established in case of infinite sets.

This can be used as the definition of an *infinite* set:

Definition 4.2: A set is called infinite iff (if and only if) we can establish a bijection between this set and its proper subset.
From this point we can also define what infinity is: Infinity is the number of elements of an infinite set.

The number of elements of a set is also called its **cardinal number** or **cardinality.**

If A is a finite set, it is of finite cardinality. The number of its elements is denoted by $|A|$ and we write $|A| < \infty$.

If there is a bijection between a given set A and the set of natural numbers $\mathbb{N}$, it is said to be **countable** (we can "count" its elements).

Obviously, if there exists a bijection between two sets, many (in case of infinite sets infinite) similar bijections (pairings) exist.
For example, another bijection between the sets of natural and even numbers can be $(1 \leftrightarrow 4)$, $(2 \leftrightarrow 6)$, $(3 \leftrightarrow 2)$, $(4 \leftrightarrow 8)$, $(5 \leftrightarrow 10)$, $(6 \leftrightarrow 12)$, ... , $(n \leftrightarrow 2n)$, ...

In general, for two finite sets A and B with $|A| = |B| = n$ there exist $n! = 1 \cdot 2 \cdot 3 \cdots n$ different bijections, and between infinite sets there are either infinitely many or no bijection at all.

Exercise 4.1: Show that there exists a bijection between the sets of natural numbers and the set of whole numbers $\mathbb{Z} = \{..., -n, ..., -3, -2, -1, 0, 1, 2, 3, ..., n, ...\}$.

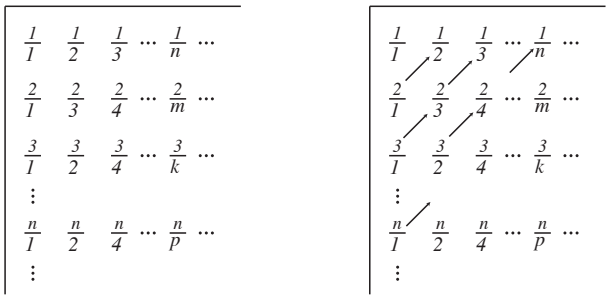
Exercise 4.2: Show that the set of odd numbers is countable.

Now consider the set of positive rational numbers $\mathbb{Q}^+$. This set is **everywhere dense**: in $]0; 1[$ there are $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, ..., \frac{1}{n}, ...$ and, in general, between any $]q_1; q_2[$ there are infinitely many rationals $q_1 + \frac{q_2 - q_1}{2}, q_1 + \frac{q_2 - q_1}{3}, q_1 + \frac{q_2 - q_1}{4}, ..., q_1 + \frac{q_2 - q_1}{n}, ...$).
Due to this fact one can think that there are "more" elements in $\mathbb{Q}^+$ as in $\mathbb{N}$, because there are infinitely many rationals between any two naturals. However, the following is true:

Theorem 4.1: The set $\mathbb{Q}^+$ of positive rationals is countable.

Proof: To show this, we explicitly construct such a counting method (this technique is called proof by construction: to prove something we construct and show it).

First of all, we "write down" all the positive rationals:



The first row contains *all* the positive rationals with numerator 1 and denominator $n \in \mathbb{N}$; The second row contains all the *simple* fractions with the numerator 2, the third with numerator 3 etc.

Obviously, this *infinite* table contains all the positive rationals (however, it can not be written completely, it is just a theoretical structure. But note that *any* positive rational can is listed there).
We count the positive rationals as follows: As the first element we will take the upper leftmost rational $\frac{1}{1}$. the second will be the first element of the second row, after which we proceed up-and-right, numerating the rationals until getting to the border of the table after which we go to the first element of the next row and repeat the operation of proceeding up-and-right until reaching the border, going to the first element of the next row and so on infinitely. It is easy to see that by this method we assign a positive rational to any natural number and vice versa.

Thus we can pair all the positive rationals wit all the natural numbers.

Q.E.D.

Exercise 4.3: Using the above theorem prove that the set $\mathbb{Q}$ of all rationals is countable.

This result is interesting but not astonishing: both sets $\mathbb{Q}$ and $\mathbb{N}$ are infinite, so no wonder that they can be paired. Here a natural question arises: Are all the infinite sets "equal" (countable)?

Here we have an astonishing and quite uneqpected counterintuitive answer: There exist infinite sets that **are not** countable!
Roughly speaking, there are infinite sets that "contain more elements" (have greater cardinality) than others.

4.2 Diagonalization: Not all infinite sets are equal!

Theorem 4.2: The set of real numbers $\mathbb{R}$ is not countable.

The proof of this theorem is an easy corollary of the following

Lemma 4.1: (Cantor's Diagonal Argument) The set $]0;1[$ is not countable.

Proof: Here we will use two milestones of the proof techniques in Mathematics and mathematical reasoning in general: proof by contradiction (Reductio ad absurdum) and the diagonalisation method.
First of all we want to mention that any irrational number can be expressed as a decimal fraction, e.g. 327,12345679875645345... or 0,12128494576... etc. (formally, this fact should be proved too).

Assume that all the reals in $]0;1[$ are countable (this is the first and most important step in this proof technique: assumption of the contrary to construct a contradiction by logical reasoning). Hence any real number will be represented as a member $0, d_{i,1}d_{i,2}d_{i,3}...d_{i,n}...$ of an infinite list:

General representation	Example
$D_1 = 0, d_{1,1}d_{1,2}d_{1,3}...d_{1,n}...$	$D_1 = 0, 1298736178...$
$D_2 = 0, d_{2,1}d_{2,2}d_{2,3}...d_{2,n}...$	$D_2 = 0, 8913467255...$
$D_3 = 0, d_{3,1}d_{3,2}d_{3,3}...d_{3,n}...$	$D_3 = 0, 9871367513...$
$\vdots$	
$D_n = 0, d_{n,1}d_{n,2}d_{n,3}...d_{n,n}...$	$D_n = 0, 8734368646...$
$\vdots$	

Here each $d_{i,j}$ denotes the j 'th digit after the decimal point of the i 'th number in the list. In particular, $d_{n,n}$ is the n 'th digit after the decimal point of the n 'th number in the list. Note that these elements are on the *diagonal* of this list (in our example the diagonal digits are $1, 9, 7$ etc.)

If we construct a real number C that can not be a member of any such list, this will be a contradictin since the assumption was that a list exists that contains *any* real number in $]0; 1[$. In other words, we should prove that to any such list a real number exist that can not be listed there.
Given any such list (that is complete meaning that it contains all the rationals in $]0 : 1[$), we construct $C = 0, [d_{1,1} + 1][d_{2,2} + 1][d_{3,3} + 1]...[d_{n,n} + 1]...$, it should be different from the first number in the given list because they have different first digits after the decimal point; It should also be different from the second number because they differ in the second digit and so on for all the numbers in the list (note that here $9 + 1 \equiv 0$). For our example above we obtain $C = 0, 208...7....$
Thus we have shown that no such list can be complete that leads to the contradiction to the initial assumption that such a complete list with all the rationals in $]0; 1[$ can exist.

Hence such a complete list can not exist. Q.E.D.

We started our argumentation by the assumption of the counterargument (the negation of the term to be proved). With a chain of logical decisions we constructed a contradiction: a term that contradicts a known fact (in this case the assumption itself). Since there was no mistake in the logical argumentations (that can be easily checked), and the main axiom of logic says that from truth there can be deduced only truce or, in other words, if we deduce a false term by logical reasoning that contain no errors, the starting assumption should be false.
This proof technique is called **proof by contradiction (reductio ad absurdum).**
In our case we assumed that *all* the reals in $]0; 1[$ can be counted in a list (no real number in $]0; 1[$ is missed there) and showed that there should exist a real that is not listed there.

Furthermore, we have used a proof technique called **diagonalisation:** we gave the pattern of *all possible* lists and constructed a decimal fraction by changing the *diagonal* elements thus constructing a contradiction.

The above theorem proves an extremely important fact: Not all infinite sets are countable (bijective). To say it nonformally, some infinite sets have "more" elements than others.
This fact is very hard to realise. Even some great thinkers of modern times (among them the founder of the 20th century philosophy Ludwig Wittgenstein) refused to recognise this fact and searched a mistake in the argumentation. Since ancient times there was a vision of unique emptiness and a unique infinity, and the realising a different fact was very difficult.

Due to the diversity of infinity we will deduce very important facts in the theory of algorithms.

Exercise 4.4: Apply the diagonalization method to prove the uncountability of $\mathbb{Q}^+$. Find a mistake in the argumentation.

Exercise 4.5: In the proof above, where did we use the fact that the representation of the reals is infinite?

A natural question is, if the representation of reals is infinite, how can we compute them by algorithms? The answer is: They *are not* computable!
However, we can compute them by any approximation (as exactly as we like): we can write an algorithm that computes any real number so that as many digits as needed after the decimal point will be correct.

As an example consider the recursive formula to compute $\sqrt{2} = 1, 41421356237310...$:

1. $a_0 = n > 0$;
2. $a_{n+1} = \frac{a_n + \frac{2}{a_n}}{2} = \frac{a_n}{2} + \frac{1}{a_n}$.

Note that the first element of the recursion can be any natural number. The special choice does not affect the result, however taking different starting values could affect the number of computational steps.

Starting with $a_0 = 1$, we obtain:

a_1=0,5+1=1,5;

a_2≈**1,41**667...;

a_3≈**1,4142**15...;

a_4≈**1,41421356237**46...

Exercise 4.6: Compute the numbers starting with $a_0 = 3$. How many steps will be needed to compute the correct result up to eight digits after the decimal point?

Exercise 4.7: Consider the computations starting with $a_0 = 1$. How many steps will be needed to compute the correct answer up to 15, 17, 20, 23, 25, 27, 30 digits after the decimal point?
In general, (approximately) how many steps are needed to compute the correct result up to $k \in \mathbb{N}$ digits after the decimal point?

Exercise 4.8: Prove that $\sqrt{2} \notin \mathbb{Q}$. *Hint:* Assume the contrary: There exist $a, b \in \mathbb{N}$ so that $\frac{a}{b}$ is irreducible, $\frac{a}{b} = \sqrt{2}$ and construct a contradiction.

Given a finite set A, we can construct a set of all its subsets 2^A. For example, if $A = \{a_1, a_2, a_3\}$, the set of its all subsets will be
$\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$. In general, the set of all subset of an n element set contains 2^n elements.
Obviously, no bijection exists between a finite set and the set of all its subsets. As can be proved by diagonalization, such a bijection does not exist for infinite sets.
Hence, $|A| < |2^A|$ meaning that for each set there exists a "mightier" set (with "more elements", or greater cardinality).

Exercise 4.9: Prove that for any set A, $|A| < |2^A|$.

4.3 Summary

In this chapter, we introduced the so called bijective mappings between sets and proved that not all infinite sets are "equal": some infinite sets have "more elements" than others. To be precise, infinite sets can have different cardinalities: there does not exist the pairings between them so that no elements of any set are left over.
There exist finite, countable and uncountable sets, and for any set there exist a set with greater cardinality ("more elements").
This very counterintuitive fact will be applied later to prove that there exist unsolvable problems. And even more: There exist "much more" unsolvable problems than solvable ones.

Chapter 5

Encoding the Data: Alphabets and Languages

5.1 Encoding the Data

Consider the words "alphabet" and "language". These are words of the English language having some meaning (semantics). Another example is "ghwpy" – it is not a word of in English (and, maybe in no language), but is constructed using Latin letters. This is the main difference between a "word of a language" and "a word constructed by an alphabet".
"A word constructed by an alphabet" is just any sequence of the letters of that alphabet that could have or not have some *meaning* (semantics), whereas "a word of a language" *must have* a meaning in the langauage (it is a selection from the set of all possible words).
A word can be finite or infinite. In our everyday life we deal with finite words.
Finite words have finite length defined by the number of elements it is constructed of.
For example, | alphabet | = 8 and | languages | = 9. The length of a given word $w = w_1w_2...w_n$ (the number of its elements) is denoted by $|w| = n$. $w(i)$ is its i 'th element. So, for example, "alphabet(4) = "h" and "electrification(7) = "i".
Given two words w_1 and w_2, its *concatenation* is $w_1 \circ w_2 = w_1w_2$ (these words combined). For example, "foot"∘"ball="football". If $w = u \circ v$, then u is the **prefix** of w and v is the **suffix** of w: $u \prec w$ and $v \succ w$.
The n-element prefix of w is denoted by $w[n]$, and its m-element suffix is denoted by $w\{m\}$ (not to be confused with $w(n)$!!!).

Exercise 5.1: Explain the meaning of the following: $|w|$, $w[|w|]$, $w(|w|)$, $w[|w|-1]$, $w[0]$, $w\{|w|\}$, $w\{|w|\}$, $w\{|w|-1\}$, $w\{0\}$.

Exercise 5.2: Explain the meaning of the following: $|w|$, $w[|w|]$, $w(|w|)$, $w[|w|-2]$, $w[0]$, $w\{|w|\}$, $w\{|w|\}$, $w\{|w|-3\}$, $w\{0\}$, where w ="electrofication"

Definition 5.1: Given any alphabet $\Sigma = \{a_1, a_2, ..., a_n\}$, Σ^n denotes the set of all words that are built by the elements of Σ and have length n:

$$\Sigma^n = \{w \mid w(i) \in \Sigma, (1 \leq i \leq n), |w| = n\}.$$

Σ^* is the set of all *finite* words on Σ:

$$\Sigma^* = \bigcup_{i=1}^{\infty} \Sigma^i = \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \Sigma^n \cup \cdots.$$

Besides the finite and infinite words there exists an **empty word** ϵ, that has no elements at all: $|\epsilon| = 0$, $\epsilon \prec w$ and $w \circ \epsilon = \epsilon \circ w = w$ for any word w.

All the above can be stated in

Definition 5.2: Any (finite) set Σ can be considered as an alphabet. A word based on the alphabet is a sequence of its elements. Given a word w on Σ, its length $|w|$ is the number of its elements. If $|w| = 0$, it is called the *empty word* and is denoted by ϵ.
If $|w| = \infty$, it is called an infinite word.

Given two words w and v (w finite), $w \circ v = wv$ is called the *concatenation* of the two words.
w is called *prefix of* w, $(u \prec w)$ if $\exists v$ such that $w = u \circ v$.
u is suffix of w, $(u \succ w)$, if $\exists v$ so that $w = v \circ u$. For any word w, $w(n)$ is its n 'th element, $w[n]$ its prefix of length n, and wn its suffix of length n.
For a given alphabet Σ, $\Sigma^n = \{w \,|\, |w| = n\}$ and $\Sigma^* = \{w \,|\, |w| < \infty\}$.

Exercise 5.3: Given two words $w_1 \in A^*$ and $w_2 \in B^*$, with A and B alphabets. What is the alphabet of $w_1 \circ w_2$?

Exercise 5.4: Given the words $w_1 = 00134$, $w_2 = 65430$, $w_3 = 001$, $w_4 = 346$, is it true that $w_3 \circ w_4 = w_1 \circ w_4[6]$? Check your answer.

A word $w \in A^*$ *contains* $v \in A^*$ if $\exists w_1, w_2 \in A^*$ and $w = w_1 \circ v \circ w_2$ (w_1 or w_2 could be empty).
In this case v is said to be a subword of w. For example, for the word "modification" the subwords are "modification", dific", "modi" to list a few.
Besides, "modi" is its prefix and "cation" its suffix.
On the contrary, "modication" is not its subword, however it is combined by two subwords.

Exercise 5.5: Which of the following is true: $w \in \Sigma^{|w|}$, $w \in \Sigma^{|w|-1}$, $w[k] \in \Sigma^k$ if w is built on the alphabet Σ and $k \in \mathbb{N}$? Prove your answer.

Words can be constructed on any (finite) alphabet (set). Given the base-10 (or decimal) alphabet $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, we can represent any natural number with it. This representation is called the *decimal* or *base-10* system because these 10 digits are used to describe any natural number.
Obviously, there are infinitely many (finite) alphabets. If Σ_1 and Σ_2 are different alphabets, there exists a bijective mapping $\Sigma_1^* \to \Sigma_2^*$ meaning that the choice of the alphabet is irrelevant: we can translate one set of words to the other one-by-one and onto.
For example, english words can be coded in decimal as follows:
First of all, we should code the latin alphabet in decimal:

a → 00	b → 01	c → 02	d → 03	e → 04	f → 05	g → 06	h → 07
i → 08	j → 09	k → 10	l → 11	m → 12	n → 13	o → 14	p → 15
q → 16	r → 17	s → 18	t → 19	u → 20	v → 21	w → 22	x → 23
y → 24	z→ 25						

As a result, we can substitute each element of an english word by an according tuple. For example, "concept" will be coded as "02141302041519".

Exercise 5.6: What will be the code of "electrofication"? Which word corresponds to the code "0304031403081306"?

As seen from the previous exercise, coding and decoding is easy if we know which code corresponds to which character. It is a little bit trickier if we do not know the correspondence, especially if the correspondence is taken randomly as shown in the next table.

a → 39	b → 27	c → 99	d → 03	e → 38	f → 21	g → 76	h → 78
i→ 87	j → 90	k → 10	l → 11	m → 13	n → 31	o → 37	p → 65
q → 16	r → 17	s → 18	t → 19	u → 47	v → 51	w → 66	x → 08
y → 24	z → 25						

If the letter-number correspondence is known, we can build a function $f : \Sigma \to A \times A$ (here Σ is any alphabet and $A = \{0, 1, 2, ..., 9\}$). In the last example, $f(\text{a}) = 39$, $f(\text{b}) = 27$, $f(\text{l}) = 11$, $f(\text{s}) = 18$ etc.
Any $w \in \Sigma^n$ can be coded by the following algorithm:

Algorithm 5.1: $P(w)$
Input: $w \in \Sigma^{|w|}$

```
1 r  if(w = ε)   {   End of Algorithm   }
2    else
3    {   return( P(w[|w| − 1]) ∘ f(w(|w|)) )   }
     (f is defined by the above table).
```

Exercise 5.7: Describe in detail each computational step of $P(\text{"}force\text{"})$.

Given an english word coded using the above method, it will be tricky to encode it (restore the original word) without knowing the relation table.

It is worth to mention that there exist efficient methods to decode the words coded by the upper method: If we know that an English text is coded, we look up for the most frequent tuples. Since we know that in English texts the most frequent letter is "e", then "a", followed by "r", then "i", "o" etc. we can assume that the most frequent code corresponds to "e", the second to "a" etc. experimenting with some patterns we can obtain real English text. One of the branches of computer science and mathematics – cryptography – deals with the methods of encoding and decoding information.

Definition 5.3: For a given alphabet Σ, any subset $L \subset \Sigma^*$ is called a **language** on the alphabet Σ.

Alphabets and languages play a central role in Informatics. Any problem can be coded in some language and the solution of this problem is equivalent to the problem of finding some specific words in that language.
In computer science a crucial role plays the so called "binary" alphabet $\mathbb{B} = \{0, 1\}$. Any information (as complex as desired) can be coded using only two symbols.
Given any natural number $n \in \mathbb{N}$, we can rewrite it in binary using the following method:

Algorithm 5.2**:** Binary
Input: $n \in \mathbb{N}$

```
1 r  if(n = 0)
2    {    return(0)    }
3   else if(n = 1)
4    {    return(1)    }
5   else
6    {    Binary(⌊n/2⌋)   (repeat the algorithm for input ⌊n/2⌋)    return the remainder after the division n/2
     (if(n is odd) return(1) ;
     (if(n is even) return(0) ;    }
```

Note that this algorithm returns the binary digits (bits) of the appropriate representation one-by-one.

Exercise 5.8: Consider the following algorithm:

Algorithm 5.3**:** Binary
Input: $n \in \mathbb{N}$

```
1 r  if(n = 0)   {    return(0)    }
2   else if(n = 1)   {    return(1)    }
3   else    Binary(⌊n/2⌋)   (repeat the algorithm for input ⌊n/2⌋)    return the remainder after the division n/2
     (if(n is odd) return(1) ;
     (if(n is even) return(0) ;    }
```

What is its output?

Exercise 5.9: Write the following numbers in binary: 13, 127, 17, 8, 16, 0.

Given any binary number, e.i. $a = 110110_2$, its individual digits (bits) can be enumerated *from right to left* starting with zero:$a = a_5a_4a_3a_2a_1a_0$. Here $a_5 = 1, a_4 = 1, a_3 = 0, a_2 = 1, a_1 = 1, a_0 = 0$. Of course, we could enumerate them from left to rigt, or beginning from one etc., but the above method has been established as standard.

Similarly we could encode any number with any alphabet.
If a number is encoded in a k-digit alphabet, it is said to be written in k-base:

Exercise 5.10: Consider the numbers given in the previous exercise. Write them in octadecimal (base-8), hexadecimal (base-16) and binary systems. Hint: for hexadecimal use the alphabet $\{0, 1, ..., 9, A, B, ..., F\}$.

Exercise 5.11: Write an algorithm that converts a binary number into decimal.

Algorithm 5.4: k-ary
Input: $n \in \mathbb{N}$

```
1 r  if(n < k)   {    return(k)   }
2    else
3    {
4     Binary(⌊n/k⌋)   (repeat this process for input ⌊n/k⌋)
5     return the remainder of n/k
6    }
```

5.2 Simulating Infinity with Finite Structures: Modular Arithmetics

One of the most natural procedures nowadays in our life is reading a clock. Assumed there are 24 hours in day-and-night, after 1 o'clock there will be 2, then 3, 4, 5,..., 23 and after it 0 (same as 24). Hence we have $0+1=1, 1+1=2, 2+1=3, 3+1=4, ..., 23+1=0$ and the cycle starts again.
So we have the following rule: $0+1=1$, $1+2=3$, ..., $22+1=23$, $23+1=0$ and we repeat the cycle. Since 24 numbers are used in total (0, 1, ..., 23), it is said that the addition is defined *modulo* 24. In this case $12+7\,mod\,24=19$, $23+1\,mod\,24=0$, $12+15\,mod\,24=3$, $503+20167\,mod\,24=6$. The general principle is: We add two numbers as usual, divide the result by 24 and take the *remainder*.
Similarly we can define multiplication: We multiply two numbers as usual, divide the result by 24 and take the *remainder*. For example, $3\cdot 7\,mod\,24=21$, $103\cdot 17\,mod\,24=23$.

Exercise 5.12: Calculate $13+17\,mod\,24$, $9+23\,mod\,24$, $23\cdot 5\,mod\,24$, $5\cdot 17\,mod\,24$.

In our finite set $\mathbb{Z}_{24}=\{0,1,2,3,,4...,23\}$ we have a so called *the identity element under addition* 0 that leaves any number unchanged after addition: $a+0\,mod\,24=a$. Hence we can state the following question: Given a number $a\in\mathbb{Z}_{24}$, does there exist a number $b\in\mathbb{Z}_{24}$ so that $a+b\ mod\ 24=0$? In if it exists, b is called *the inverse element of a regarding addition.* As usual, the inverse element regarding addition is denoted by $-a$.
In fact, we can find an inverse regarding addition to any given number $a\in\mathbb{Z}_k=\{0,1,...,k-1\}$. For example, $11+13\,mod\,24=0$, $23+1\,mod\,24=0$, $7+17\,mod\,24=0$ etc. In general, $-a\in\mathbb{Z}_k$ can be calculated by $(k+1)-a$.
Similarly we can define *the identity element under multiplication* denoted by 1: for any $a\in\mathbb{Z}_k$, $a\cdot 1=a$.
The inverse to an element a regarding multiplication would be a^{-1} such that $a\cdot a^{-1}=1$. For example, $13\cdot 3\,mod\,24=1$ so that 13 and 3 are inverse to each other regarding multiplication *mod* 24.
By a simple check we can see that $6\in\mathbb{Z}_{24}$ has no inverse regarding multiplication *mod* 24. This example shows an important fact: Some elements can have no inverses regarding multiplication.

Exercise 5.13: Write down all the numbers in $\mathbb{Z}_{24}$ that have an inverse regarding multiplication. Compare them with the numbers that do not have the inverses. Is there any regularity in the sets of these numbers?

A deep result of algebra states that in *modular arithmetics* (arithmetics in the set $\mathbb{Z}_k$ modulo k), an element $a\in\mathbb{Z}_k$ has an inverse regarding multiplication iff a and k are relatively prime.

An interesting fact is that, in modular arithmetics, the inverses of some elements are the elements themselves. For example, $1\cdot 1\,mod\ 24=1$ and $7\cdot 7\,mod\ 24=1$.

Exercise 5.14: Calculate $(-13)\,mod\ 24$, $(-1)\,mod\ 24$, $13^{-1}\ mod\,24$, $-13\,mod\ 24$, $17^{-1}mod\ 24$.

We can also define the arithmetical operations modulo any $m\in\mathbb{N}$: For any $a,b\in\mathbb{Z}_m$, calculate $c=a+b$ or $d=a\cdot b$, divide the result by m and compute the residue.

Exercise 5.15: Compute $(-13)\,mod\,27$, $(-1)\,mod\,9$, $13^{-1}\,mod\,27$, $-13\,mod\,31$, $17^{-1}mod\,41$, $13+17\,mod\,34$, $9+23\,mod\,4$, $23\cdot 5\,mod\,32$, $5\cdot 17\,mod\,47$.

An important question can arise quite naturally: What use has modular arithmetics? This question was stated long time ago and for some period of time the investigation of modular arithmetics and final structures in general (such as finite rings, fields, groups etc.) were regarded as purely theoretical without any interesting application. After the deep results from the 19th century, mostly based on the results of Galois concerning the non-solvability of the equations with the degree higher that four in radicals, or the solution of the ancient Greek problems mentioned above, the applications of finite structure arithmetics seemed to be impossible. But after the development of computers in the 20st century it became actual again.

Since the memory of any computer is limited, it is impossible to represent infinite structures (such as the set of natural numbers, to say nothing about the reals), it is impossible to perform absolutely reliable computations without losing any information.
One way to solve this problem would be the use of arithmetics on finite structures to mimic the arithmetics on infinite structures: Take a finite ring with a unit (a structure with addition and multiplication as well as a unit regarding multiplication and one regarding addition), such as $\mathbb{Z}_k$ (here k is prime) and perform computations modulo k.
In real life we can choose k so large that the computations in $\mathbb{Z}_k$ are satisfiable *in most cases,* so that we can mimic the computations in $\mathbb{N}$ without the loss of information.
Of course, modular arithmetics can not give us the right answer: in some cases we can deal with large numbers that no longer fit in the finite set $\mathbb{Z}_k$, but in most practical cases such divergencies can not greatly influence correct computational results.
Besides this, there exist methods (like the Chinese Remainder theorem) due to which one can obtain computational results in a large ring combining the results from several smaller rings.
These are problems of computational Algebra we will deal with in later courses.

5.3 Elements of the Binary Arithmetics

Consider the binary numbers $A = (a_3a_2a_1a_0) = (1101)$ and $B = (b_3b_2b_1b_0) = (1110)$. Like the numbers written in decimal, we can add them by the school method:

0	1	1	0	1	13
0	1	1	1	0	14
x_4	x_3	x_2	x_1	x_0	

First of all, the "lower" (rightmost) bit must be added: $x_0 = 1 \oplus 0 = 1$ (here, $\oplus$ is the binary addition – addition modulo 2).
Like decimal numbers, we have "carry bit" 0 or 1 that should be added with higher bits. In our example we have $c_1 = 0$ since we have not two 1 in the first bits. To compute the second bit of the sum we have $x_1 = 0 \oplus 1 \oplus c_1 = 0 \oplus 1 \oplus 0 = 1$ (sum up the actual bits and the previous carry bit). In this step, the carry bit is $c_2 = 0$, since there were less than two 1s in the three summands of x_1. After this, we calculate $x_3 = 1 \oplus 1 \oplus d_2 = 1 \oplus 1 \oplus 0 = 0$, and the carry bit is $c_3 = 1$, because we have tho 1s in the three summands of x_2.
Compute $x_3 = 1 \oplus 1 \oplus c_3 = 1 \oplus 1 \oplus 1 = 1$ and $c_4 = 1$ (same considerations).
In the last step, $x_4 = 0 \oplus 0 \oplus c_4 = 0 \oplus 0 \oplus 1 = 1$ and we obtain the final result:

0	1	1	0	1	13
0	1	1	1	0	14
1	1	0	1	1	27

Exercise 5.16: Describe in details the computational process (the results and carrybits in every step) of the following sums: $(10010101)_2 + (10010101)_2$, $(11110101)_2 + (00010101)_2$ and $(10010001) +_2 (10011101)_2$.

In general, we have the following algorithm:

Algorithm 5.5: Addition of Binary Numbers

1: **procedure** $SumBinary((a_n, ..., a_0), (b_n, ..., b_0))$ $c_0 = 0$;
2: for ($i = 0, i \leq n, i++$)
3: {
4: $x_i = a_i \oplus b_i \oplus c_i$;
5: if (there are two or three 1s in the variables a_i, b_i and c_i)
6: then $c_{i+1} = 1$;
7: else $c_{i+1} = 0$;
8: }
9: $x_{n+1} = c_{n+1}$;

Exercise 5.17: Explain why is it necessary to compute x_{n+1} but not x_{n+2} or x_{n+3}?

Exercise 5.18: Prove the correctness of the above algorithm and compute the number of its steps.

Exercise 5.19: Following the pattern of addition, write an algorithm for the multiplication of two n-bit numbers. Prove its correctness and compute the number of its steps.

If we are given n bit numbers and no longer numbers can be represented, modulo 2^n arithmetics will be constructed over the set $\mathbb{Z}_{2^n}$, since we can represent 2^n different numbers with n bitis.
Consider an 8-bitnumber $11111111_2 = 255_{10}$. Obviously, $11111111_2 + 1 \, mod \, 2^8 = 0$, so $-255 \, mod \, 256 = 1$.

Exercise 5.20: Prove the following statement: given an n-bit binary number $(111...1)_2$, its inverse in respect of addition is 1.

Now consider another 8-bit binary number $x = 11010010$. How can we calculate its inverse regarding addition?
According to the above considerations, we should calculate a number y with the following property: $x+y = 11111111$ to obtain $-x \, mod \, 2^n = y + 1$.
It is easy to see that such y (in our example $y = 00101101$) Should consist of the inverted bits of the initial number x: For any $x_{n-1}...x_1x_0$, the corresponding "reverse number" will be $y = \overline{x_{n-1}}...\overline{x_1}, \overline{x_0}$, where $\overline{x_i}$ is the negation of the x_i bit: $\overline{1} = 0$, $\overline{0} = 1$.
Hence we have the following rule to construct the inverse regarding addition modulo 2^n: Construct the reverse number and add 1:

$$-(x_{n-1}...x_1x_0) \, mod \, 2^n = (\overline{x_{n-1}}...\overline{x_1}, \overline{x_0}) + 1.$$

Chapter 6

Relations and Sorting

6.1 Relations

Consider the set of the citizens of Georgia $A = \{w \mid w \text{ is a citizen of Georgia}\}$. Obviously, there will exist subsets of friends. If $a, b \in A$ and a is a friend of b, then b is a friend of a. We can represent their friendship as (a, b). Writing all such pairs of friends we obtain a ser $R = \{(a, b) \mid a, b \in A,\ a \text{ and } b \text{ are friends}\}$.
It is clear that the set R shows some relation between the elements of the set A that gives us some information about its elements. Obviously, $(a, b) \in R \Leftrightarrow (b, a) \in R$.
Now consider the same set A and a relation
$R_1 = \{(a, b) \mid a, b \in A,\ b \text{ is an ancestor of } a\}$. Here $(a, b) \in R_1 \Rightarrow (b, a) \notin R_1$. It gives us different information about the relation of elements of A and has a fundamental difference from R above (to be explained later).

After these examples we want to define some fundamental notions:

Definition 6.1: For any two sets A and B, their Cartesian product is defined as $A \times B = \{(a, b) \mid a \in A, b \in B\}$.

For example, if $A = \{1, 2, 3, 4\}$ and $B = \{a, b, c\}$, then

$$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c), (4, a), (4, b), (4, c)\}.$$

Note that the sequence in which the elements appear in the pairings are crucial for the relations: the elements of A followed by the elements of B.

Obviously, $A \times A$ is the cartesian product of A with itself.

For example, if $A = \{a_1, a_2, a_3\}$,
$A \times A = \{(a_1, a_1), (a_1, a_2), (a_1, a_3), (a_2, a_1), (a_2, a_2), (a_2, a_3), (a_3, a_1), (a_3, a_2), (a_3, a_3)\}$.

Exercise 6.1: given a natural number $n \in \mathbb{N}$, construct $A \times A$, where $A = \{a_1, a_2, ..., a_n\}$.
What is the input of this problem? What is its result?
Write an algorithm that solves the above problem.

Definition 6.2: Given any two sets A and B (often is $A = B$), the set $R \subset A \times B$ is called the relation on the sets A and B.

The "friendship relation" R and the "ancestor relation" R_1 defined above give us two different relationships of the same set A.

As an example of a relation defined on the cartesian product of two different sets consider the relation of georgian cities with their regions:
A = {Kartli, Kakheti, Racha, Imereti, Megrelia} da B = {Ozurgeti, Oni, Poti, Agara, Zugdidi, Vani, Telavi, Gurjaani, Kutaisi}.
The relation connecting eich city to the proper region will be:

$$\begin{aligned} R_2 \quad = \{ \quad & \text{(Kartli, Agara), (Kakheti, Telavi), (Kakheti, Gurjaani), (Racha, Oni), (Imereti, Vani),} \\ & \text{(Imereti, Kutaisi), (Megrelia, Poti), (Megrelia, Zugdidi) } \}. \end{aligned}$$

Definition 6.3: Any relation $R \subset A \times B$ (often $A = B$) can have the following properties:

- For any $a_1 \in A, a_2 \in B$, $a_1 \neq a_2$, $(a_1, a_2) \in R$ or $(a_2, a_1) \in R$ (or both), then R is *complete*;
- If $\forall a \in A$, $(a, a) \in R \subset A \times B$, then R is *reflexive*;
- If $\forall a_1 \in A, a_2 \in B$, $a_1 \neq a_2$, $(a_1, a_2) \in R \Leftrightarrow (a_2, a_1) \in R$, then R is *symmetric*;
- If $\forall a_1 \in A, a_2 \in B$, $a_1 \neq a_2$, $(a_1, a_2) \in R \Rightarrow (a_2, a_1) \notin R$, then R is *antisymmetric*;
- If $\forall a_1, a_2, a_3 \in A$, $a_1 \neq a_2$, $a_3 \neq a_2$, $(\,(a_1, a_2) \in R$, $(a_2, a_3) \in R\,) \Rightarrow (a_1, a_3) \in R$, then R is *transitive*.

The above (friendship) relation R is symmetric but not complete because there exist two people $a, b \in A$ that are not friends: $(a, b) \notin R$.
The second (ancestors) relation R_1 is transitive: if b is ancestor of a ($(a, b) \in R_1$) and c is ancestor of b ($(b, c) \in R_1$), c is ancestor of a, $(a, c) \in R_1$. It is also antisymmetric: $(a, b) \in R_1 \Leftrightarrow (b, a) \notin R_1$.
The third relation R_2 is antisymmetric and not complete.

Exercise 6.2: Prove that the relation $R_3 \subset \mathbb{N} \times \mathbb{N}$, $R_3 = \{(a, b) | a \leq b\}$ is reflexive and complete.

Exercise 6.3: Explicitly describe the following sets:

(a) $\{1\} \times \{1, 2\} \times \{1, 2, 3\}$;

(b) $\emptyset \times \{1, 2, 3\}$;

(g) $2^{\{1,2\}}$ that is the set of all subsets of $\{1, 2\}$;

(d) $2^{\{1,2\}} \times \{1, 2\}$.

It is convenient to depict not too large relations as small circles connected with arrows: Given set A and a relation R defined on it, $(a, b) \in R$, the elements a and b can be represented as small circuits connected by an arrow from a to b (fig. 6.1 (a)).
If $(b, b) \in R$, the element b is represented by a small circle connected by an arrow with itself (fig. 6.1 (b)).
For $A = \{a, b, c, d, e\}$, the relation
$R = \{(a, a), (a, b), (b, b), (b, a)(c, a), (c, d), (d, e), (e, b), (e, c), (e, d)\}$ is represented in fig. 6.1 (g).

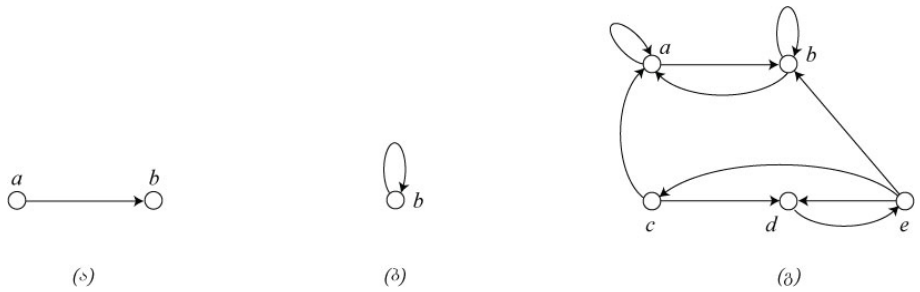


Figure 6.1: Depiction of relations

Definition 6.4: A reflexive, symmetric and transitive relation is called "equivalence" relation.
Note that an equivalence relation classifies and unites "similar" elements into different subsets.

For example, if we are given a set A of living organisms, then $R' = \{(a, b) \,|\, a$ da b are both mammals$\}$ is an equivalence relation.

Exercise 6.4: Prove that the above relation R' is reflexive, symmetric and transitive.

As mentioned before, the equivalence relations divides a given set into so called "equivalence classes" – in subsets that contain elements that are similar in certain way.

If an equivalence relation R is defined on a set $A \neq \emptyset$, it defines such subsets $B_\alpha \subset A$ that $B_\alpha = \{a, b \in A \,|\, (a, b) \in R\}$ (each subset contains only such elements are "similar" by the definition of R).

Example: The relation $R = \{(a, b)|\, a$ and b are both even or a and b are both odd$\}$ divides the set of natural numbers $\mathbb{N}$ into two disjoint classes (sets) of odd and even numbers: $N_1 = \{a_i|a_i$ is odd, $i \in \mathbb{N}\}$, $N_2 = \{a_i|a_i$ is even, $i \in \mathbb{N}\}$.

Given a set $A = \{0, 1, 2, 3, 4, 5, 6, 7\}$, we can depict the relation $R = \{(a, b) \in A \times A \,|\, a$ and b are even or a and b are odd$\}$ as

Figure 6.2: Division of a set in two independent classes

Obviously, the relation R divides A into two independent classes (subsets).

The equivalence of the elements depends on the relation: two different relations can determine differend equivalence classes.
Example: Given the set $\mathbb{N}$, the relation $R' = \{(a, b) \,|\, a$ and b are both divisible by 3 or either a or b are not divisible by 3$\}$.

Exercise 6.5: Depict the relation R'.

The equivalence classes of a set A defined by an equivalence relation R are denoted as $[a] = \{b \,|\, (a, b) \in R\}$.
If $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $R' = \{(a, b) \,|\, a$ and b are both divisible by 3 or either a or b are not divisible by 3$\}$, $[6] = \{0, 3, 6, 9\}$ da $[2] = \{1, 2, 4, 5, 7, 8\}$ (note that $[6] = [3] = [0] = [9]$ and $[1] = [2] = [4] = [5] = [7] = [8]$).

Exercise 6.6: Give an example of an equivalence relation that divides the set of natural numbers into three equivalence classes.

Now consider two natural numbers 307 and 509 denoted in a decimal alphabet. As we all know, the relation between them is $307 < 509$. This relation should be either defined somewhere or be a logical consequence of some basic definitions (alternatively we could define $509 < 307$).
As a basic definition we have the relations $0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9$ that are not obvious, it is just a common convention.
So we have the following relation on the set $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$:

$$
\begin{aligned}
R \;= \{ & \\
& (0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8),(0,9) \\
& (1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(1,9) \\
& (2,3),(2,4),(2,5),(2,6),(2,7),(2,8),(2,9) \\
& (3,4),(3,5),(3,6),(3,7),(3,8),(3,9) \\
& (4,5),(4,6),(4,7),(4,8),(4,9) \\
& (5,6),(5,7),(5,8),(5,9) \\
& (6,7),(6,8),(6,9) \\
& (7,8),(7,9) \\
& (8,9) \\
\} &
\end{aligned}
$$

It determines the so called *order* and gives the rule how to sort the elements in ascending order.

Definition 6.5: A complete, antisymmetric and transitive relation is called *total order*. An incomplete, antisymmetric and transitive relation is called *partial order*.

Exercise 6.7: Prove that the relation R above is a total order.

Exercise 6.8: Give an example of a partial order on the above set A.

If $(a, b) \in R$ and R is order then we can write $a < b$.

Given an order of the above set A, it is easy to order all the elements of A^* due to the following algorithm:

Algorithm 6.1: $C(w, v)$
Input: $w = (w_1, w_2, ..., w_n), v = (v_1, v_2, ..., v_m) \in A^*$

---

1 If $|w| = |v| = 0$ then $w = v$ end of algorithm
2 If $w(1) < v(1)$ then $(w, v) \in R$ (or, equivalently, $w < v$) end of algorithm
3 If $v(1) < w(1)$ then $(v, w) \in R$ (or, equivalently, $v < w$) end of algorithm
4 Execute $C(w\{|w| - 1\}, v\{|v| - 1\})$ (the same algorithms for the suffixes of the words w and v).

Note that $\epsilon < a, \forall a \neq \epsilon \in A$.

Exercise 6.9: Explain the meaning of the mathematical notations in the previous algorithm: "If $w(|w|) < v(|v|)$ then..." and "$C(w\{|w| - 1\}, v\{|v| - 1\})$".

Exercise 6.10: Prove the correctness of the above algorithm. Count the number of steps for $|w| = |v|$ and $|w| \neq |v|$.

Exercise 6.11: Write explicitly the complete order relation R on the above set A that sorts the elements as $1 \leq 3 \leq 2 \leq 5 \leq 8 \leq 4 \leq 0 \leq 9 \leq 7 \leq 6$.

Exercise 6.12: Give two examples of partial order relations on the set A. Can $R = \emptyset$ be its partial order?

Exercise 6.13: Is the relation $R = \emptyset$ defined on any set X transitive?

Exercise 6.14: Can there exist a relation $R \neq \emptyset$ defined on a set X that is both symmetric and antisymmetric?

Exercise 6.15: Prove that for any partial order relations R_1 and R_2 defined on some set X, $R_1 \cap R_2$ is a partial order defined on the same set.

Exercise 6.16: Consider a set S whose elements are sets. Prove that the relation $R_S = \{(A, B) \mid A, B \in S, A \subseteq B\}$ is a partial order.

Exercise 6.17: Consider a set $S = 2^{\{1,2,3\}}$ that is a set of all subsets of $\{1, 2, 3\}$.
Explicitly write its elements and depict the relation R_S as defined in the previous exercise.
Write its minimal and maximal elements (such that $(min, b) \in R_S$, $\forall b \in S$ and $(b, max) \in R_S$, $\forall b \in S$).

Exercise 6.18: Formally define the minimal and maximal elements of an ordered set.
Can such elements always exist?

Now consider the relation given in fig. 6.3 (a).

Exercise 6.19: Show that the relation given in fig. 6.3 (a) is neither reflexive nor transitive.

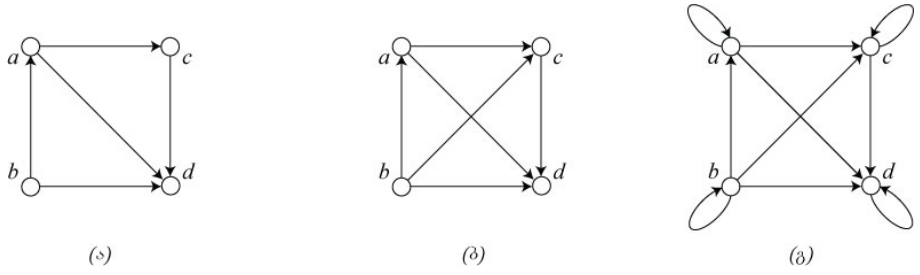


Figure 6.3: Reflexive and transitive closure of a relation

Adding appropriate new pairs of elements (or arrows in the diagrams) one can obtain a transitive relation (fig. 6.3 (b). On the other hand, adding the pairs (a, a) for each element a the relation becomes reflexive (fig. 6.3 (g)). The process of adding pairs of elements to the relation set is called the ***reflexive and transitive closure*** and can be applied to any relation.

Definition 6.6: For any relation R, its reflexive and transitive closure R^* is called a relation such that $R \subset R^*$ and the number of elements of R^* is minimal (R^* is obtained form R adding minimal number of pairs).

Exercise 6.20: Write an algorithm that constructs a reflexive and transitive closure from any relation R defined on a set A (constructs an appropriate set). Prove its correctness and count the number of steps if $|A| = n$.

6.2 Applications of sorting and equivalences: Searching in sets and residue classes

Total and partial orderings play a central role in computer science and our everyday life: many algorithms can be performed efficiently on sorted sets.

As an example consider the total ordering defined on the lathin alphabet Q: $a < b < c < d < \cdots < y < z$.
If we succeed to sort all english words based on the sorting principle of Q by defining some sorting on the set Q^*, searching any word w in any english dictionary will be much faster: first, we flip the book in the middle and consider the first word v there. If $w = v$, we are done. If $w < v$, repeat the same operation recursively with the first (left) half, or the right half (if $w > v$) of the pages considered.

Exercise 6.21: Write an algorithm that, for any two words $w, v \in Q^*$, defines $w = v$ or $w < v$ or $v < w$.
Remark: Use the analogy to the number comparison problem.

Exercise 6.22: Proof the correctness of the algorithm from the previous exercise and count the number of steps if $|w| = n$ da $|v| = m$.

Given any alphabet A and an ordered set of its words $S = \{u_1, u_2, ..., u_n \in A^*\}$, the following algorithm searches any given word $w \in A^*$ in S:

Algorithm 6.2: Algorithm $L(S, w)$
Input: A set of words $S = \{u_1, u_2, ..., u_n \in A^*\}$ and a word w
Output: Index i such that $u_i \in S$, $u_i = w$ (if it exists)

1: **procedure** $L(S, w)$
2: If $S = \emptyset$, *print*($w \notin S$); **End of the algorithm**
3: If $u_{\lfloor \frac{|S|}{2} \rfloor} = w$, *print*($\lfloor \frac{|S|}{2} \rfloor$-th element); **End of the algorithm**
4: If $u_{\lfloor \frac{|S|}{2} \rfloor} < w$, execute $L(\{u_{\lfloor \frac{|S|}{2} \rfloor + 1}, ..., u_n\}, w)$;
5: If $u_{\lfloor \frac{|S|}{2} \rfloor} > w$, execute $L(\{u_1, ..., u_{\lfloor \frac{|S|}{2} \rfloor}\}, w)$;
6:

Algorithm 6.3: Algorithm $L(S, w)$
Input: A set of words $S = \{u_1, u_2, ..., u_n \in A^*\}$ and a word w
Output: Index i such that $u_i \in S$, $u_i = w$ (if it exists)

1: If $S = \emptyset$, *print*($w \notin S$); **End of the algorithm**
2: If $u_{\lfloor \frac{|S|}{2} \rfloor} = w$, *print*($\lfloor \frac{|S|}{2} \rfloor$-th element); **End of the algorithm**
3: If $u_{\lfloor \frac{|S|}{2} \rfloor} < w$, execute $L(\{u_{\lfloor \frac{|S|}{2} \rfloor + 1}, ..., u_n\}, w)$;
4: If $u_{\lfloor \frac{|S|}{2} \rfloor} > w$, execute $L(\{u_1, ..., u_{\lfloor \frac{|S|}{2} \rfloor}\}, w)$;

Exercise 6.23: Prove the correctness of the above algorithm usting mathematical induction. Count the number of steps assuming $|S| = n$.

Well-ordered sets are also effective solving mathematical questions: Finding the minimum or maximum of the values of a given function, or finding the union, intersection or difference of two sets can be performed much faster on sorted sets.

Exercise 6.24: Given two ordered sets A and B, compute $A \cap B$, $A \cup B$ and $A \backslash B$ (for the sake of simplicity assume that both sets contain natural numbers). Prove its correctness and count the number of steps.

As an example of the application of equivalence relation, consider the idea of modular arithmetics discussed in the previouse chapter: A given set of numbers $\mathbb{Z}_k = \{0, 1, ..., k-2, k-1\}$ can be generated from the set of whole numbers $\mathbb{Z}$ by dividing each $z \in \mathbb{Z}$ by k and taking its residue. In other words, two numbers $z_1, z_2 \in \mathbb{Z}$ are equivalent (belong to the same equivalence class), if after division by k, their residues coinside: $z_1 \equiv z_2 \Leftrightarrow \{\frac{z_1}{k}\} = \{\frac{z_2}{k}\}$.

$$\begin{aligned} 0 \equiv [0] &= \{0, k, -k, 2k, -2k, 3k, -3k, 4k, -4k, ...\}, \\ 1 \equiv [1] &= \{1, -1, k+1, -(k+1), 2k+1, -(2k+1), ...\}, \end{aligned}$$

$$\begin{aligned} &. \quad . \quad . \\ \end{aligned}$$

$$\begin{aligned} k-2 \equiv [k-2] &= \{k-2, -(k-2), 2k-2, -(2k-2), ...\}, \\ k-1 \equiv [k-1] &= \{k-1, -(k-1), 2k-1, -(2k-1), ...\} \end{aligned}$$

Note that theequivalence (or residue) classes are denoted by rectangle brackets, in our case it is denoted as $[z_1] = [z_2] = \{x \mid \{\frac{x}{k}\} = \{\frac{z_1}{k}\} = \{\frac{z_2}{k}\}\}$.

It is a common practice in Algebra to consider the so called factor rings by dividing all elements of the ring by its one specific element and taking the quotients that creates certain equivalence classes.

6.3 Summary

In this chapter, we introduced the notion of relations and its important examples: equivalence relation and sorting with their applications. Dividing sets into equivalence classes is important in natural sciences (i.e. the classification of biological objects) as well as in mathematics and computer science (residue classes), discussed in the previous chapter as modular arithmetics. On the other hand, sorting has a wide range of applications in almost all problems: from searching in ordered sets the classification of archeological data.