DISCRETE MATHEMATICS

with applications in Computer Science

Nikolai Chukhin
Alexander S. Kulikov

About This Textbook

Thank you for joining us! This textbook accompanies our year-long course on Discrete Mathematics in the Computer Science and Artificial Intelligence B.S. program at Neapolis University Pafos. It covers all the classical topics: combinatorics, graph theory, probability theory, logic, and set theory. It also presents many applications of these topics in different areas of computer science: artificial intelligence, bioinformatics, coding theory, complexity theory, cryptography, data structures and algorithms, programming language theory, and formal methods. Together, we will see many ways to use these ideas in practice. Each chapter also includes advanced topics that are not covered in standard discrete mathematics courses.

This book was last updated on September 16, 2026; get its latest version at Leanpub.

Contents at a Glance

Contents

About the Book

Problems and Puzzles

The textbook contains more than 500 exercises and 500 theory problems. Their difficulty varies: some problems help you check your understanding of a new definition, while others ask you to apply the ideas and methods you have studied. For almost all exercises in this textbook, we provide detailed solutions.

For many methods in discrete mathematics, we will invite you to solve an interactive puzzle and try to discover the underlying idea on your own. This serves several purposes. First, it is engaging! Second, discovering an idea yourself makes it more memorable than simply being shown it. Third, few things compare to the joy of solving a math problem!

Python

You will see many examples of Python code. These code snippets serve as interactive examples. For instance, textbooks often explain an algorithm with a picture showing its steps on a specific input. An implementation, however, allows you to trace the algorithm on *any* input.

Why do we need code in a math course?

Applications. Python examples will show you a wide range of practical applications of discrete mathematics.

Interactive examples. Code can be used to create interactive examples: copy the code, change its parameters, run it, and see what happens.

Deeper understanding. Implementing a method from discrete mathematics will help you understand all its details.

I find that I don't understand things unless I try to program them.

Donald Knuth (1938–)

But why Python rather than another language?

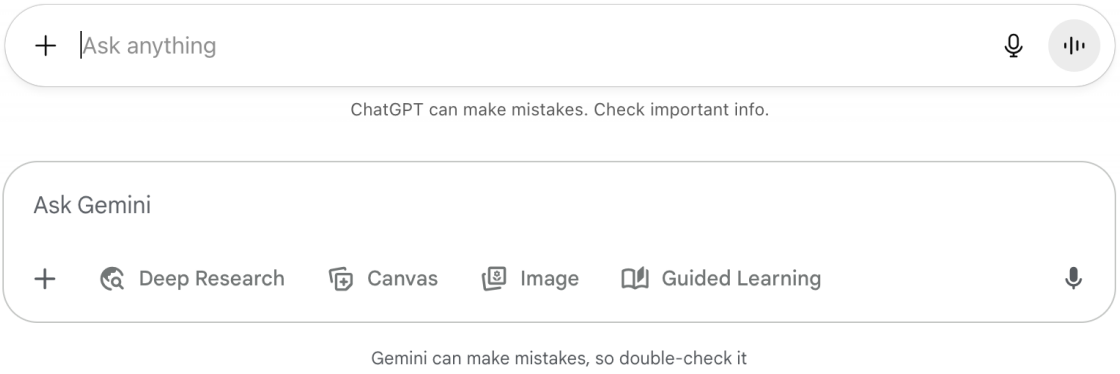
High-level language. It is relatively easy to start programming in Python, even if you have never programmed before: the code is compact and easy to read.

Interactive mode. Python can be used interactively to communicate with the computer through a REPL (read-eval-print loop).

Batteries included. Many libraries allow you to draw a graph, plot a function, generate a random sequence, and much more in just a few lines of code.

AI Era

Is it still worth studying discrete mathematics in the era of large language models? Indeed, large language models are becoming increasingly good at solving many standard mathematical problems. However, they still make mistakes, both in calculations and in reasoning.



Thus, responsibility for verifying the result still lies with the user, and the ability to distinguish a correct solution from an incorrect one becomes even more important. To evaluate a solution, one must understand the mathematical *language* in which it is written; studying this language is one of the main goals of this textbook.

For example, a model might think that $8.8 - 8.11 = -0.31$. If you don't know how addition and subtraction work, you won't be able to recognize that the model has made

a mistake. As another example, suppose you asked a model to prove that the n-th Fibonacci number is not less than $2^{n/2}$ for all $n \geqslant 6$. The model might produce the following solution:

> We prove by induction on n. Base case $n = 6$: $F_6 = 8 \geqslant 8 = 2^{6/2}$. Induction step $n - 1 \to n$:
>
> $$F_n = F_{n-1} + F_{n-2} \geqslant 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} = 2^{\frac{n}{2}-1}(\sqrt{2}+1) > 2^{\frac{n}{2}-1} \cdot 2 = 2^{\frac{n}{2}}.$$

If you don't know what mathematical induction is, or if you have little experience working with it, you might not notice the mistake in this proof. A similar incorrect argument could even be used to "prove" the obviously false statement that all Fibonacci numbers are even.

At the same time, large language models often solve problems correctly, and discussing ideas and solutions with them can be useful. We recommend the following approach. Whenever you need to solve a problem, don't rush to read our solution or ask a large language model for a solution:

- Don't deprive yourself of the unique satisfaction of solving a problem!
- If you solve a problem yourself, both the solution and its methods will stick in your memory much better.
- Even if you fail to solve the problem, the time will not be wasted: you will try different approaches and identify exactly what makes the problem difficult. This, in turn, will help you study the given solution with greater interest and appreciation.

Once you have solved the problem or come up with an idea, it is perfectly fine to discuss it with a large language model.

Notation

We use the following standard notation and facts.

Notation

- $[P]$ is the Iverson bracket: $[P] = 1$ if P is true, and $[P] = 0$ otherwise ($[2 < 3] = 1$, $[2 \leqslant 3] = 1$, $[2 > 3] = 0$)
- $\lfloor x \rfloor$ is the floor of x, obtained by rounding x down to the nearest integer ($\lfloor 3 \rfloor = 3$, $\lfloor 3.01 \rfloor = 3$, $\lfloor 3.873 \rfloor = 3$)
- $\sum_{i \in [n]} f(i) = f(1) + f(2) + \cdots + f(n)$ is the sum of $f(i)$ over all i from 1 to n
- $a \equiv_m b$ states that the integers a and b are congruent modulo m (i.e., they give the same remainder when divided by m; equivalently, their difference is divisible by m)

Sets

- $[n] = \{1, 2, \ldots, n\}$ is the set of positive integers from 1 to n
- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ is the set of all integers

- $\mathbb{Z}_{\geqslant n} = \{n, n+1, n+2, \dots\}$ is the set of integers not less than n
- $\mathbb{P} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, \dots\}$ is the set of prime numbers
- $\mathbb{R}$ is the set of real numbers

Inequalities

- $1 + x \leqslant e^x$ for all $x \in \mathbb{R}$

Set theory

- $|A|$ is the size (or cardinality) of the set A
- $x \in A$ states that the element x belongs to the set A (or the set A contains the element x)
- $A \cup B$ is the union of the sets A and B, i.e., all elements that lie in at least one of the sets A and B
- $A \cap B$ is the intersection of the sets A and B, i.e., all elements that lie in both sets A and B

Acknowledgments

This book has benefited greatly from the contributions of many individuals, to whom we are deeply grateful. We are especially thankful to Maksim Levitskii, Georgie Levtsov, Ivan Mihajlin, Maksim Nikolaev, and Ivan Pavlov for their invaluable contributions.

Chapter 1

Mathematical Thinking

1.1 Proofs of Existence and Optimality

How can we prove that an object with given properties exists, or show that it cannot exist? We explore constructive and nonconstructive arguments, prove impossibility and optimality, and see how the same ideas build error-correcting codes.

1.1.1 Constructive Proofs of Existence

In discrete mathematics and computer science, *rigorous proofs* are essential because they provide certainty in a domain where intuition can be misleading. Many problems involve combinatorial explosion, subtle edge cases, or non-obvious structures, so only a formal proof can guarantee that a statement holds in all cases—not just for tested examples. *Proofs of existence* show that a desired object or solution actually exists, which is crucial when designing algorithms or systems, whereas *proofs of optimality* ensure that no better solution is possible, justifying efficiency claims. Without such guarantees, algorithms might fail on unseen inputs or perform suboptimally, which can have serious consequences in practice. Rigorous reasoning thus forms the foundation for correctness, reliability, and performance in theoretical and applied computer science.

When you yourself are responsible for some new application in mathematics... then your reputation... and possibly even human lives, may depend on the results you predict. It is then the need for mathematical rigor will become painfully obvious to you.

Richard Hamming (1915–1998)

Problem 1.1.1 Multiple choice, 5 points

Mark the correct statements:

☐ There exists a positive integer that is divisible by 13 and ends in 15.

☐ There exists a positive integer that is divisible by 15 and ends in 13.

Yes, a number ending in 15 and divisible by 13 exists: $715 = 13 \cdot 55$. This is a clear example of a *constructive* proof of existence: to prove that the desired object exists, we simply present it. In the proof itself, we may not explain how exactly the desired object was found. The number 715 could have been found by manual enumeration, with the help of a computer, or even seen in a dream. The code below finds all such numbers in the interval $[0, 9\,999]$.

```
for n in range(10 ** 4):
    if n % 13 == 0 and n % 100 == 15:
        print(n)
```

```
715
2015
3315
4615
5915
7215
8515
9815
```

The following code finds just the first such number.

```
from itertools import count

print(next(n for n in count() if n % 13 == 0 and n % 100 == 15))
```

```
715
```

However, a number ending in 13 and divisible by 15 does not exist: if a number is divisible by 15, it ends in 0 or 5.

Problem 1.1.2 Choice, 5 points

Recall that a number $p \in \mathbb{Z}_{>0}$ is called *prime* if it is divisible only by one and itself. The number $p = 1$ is traditionally not considered prime. Here are some of the first prime numbers.

```
from sympy import primerange

print(*primerange(130))
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127
```

Euler discovered that the polynomial $p(n) = n^2 - n + 41$ has an amazing property: for small n, the value of $p(n)$ is prime.

```python
from sympy import isprime


def p(n):
    return n ** 2 - n + 41


for n in range(1, 35):
    print(n, p(n), isprime(p(n)))
```

```
1 41 True
2 43 True
3 47 True
4 53 True
5 61 True
6 71 True
7 83 True
8 97 True
9 113 True
10 131 True
11 151 True
12 173 True
13 197 True
14 223 True
15 251 True
16 281 True
17 313 True
18 347 True
19 383 True
20 421 True
21 461 True
22 503 True
23 547 True
24 593 True
25 641 True
26 691 True
27 743 True
28 797 True
29 853 True
30 911 True
31 971 True
32 1033 True
33 1097 True
34 1163 True
```

Is it true that $p(n)$ will be prime for all non-negative integers n? (In other words, does there *exist* $n \in \mathbb{Z}_{\geqslant 0}$ such that $p(n) = n^2 - n + 41$ is not prime?)

- Yes, $p(n)$ is prime for all non-negative integers n.
- No, there exists a non-negative integer n such that $p(n)$ is not prime.

It is not difficult to use the code provided above to find the first n for which $p(n)$ is not prime.

```python
from itertools import count
from sympy import isprime
```

```
def p(n):
    return n ** 2 - n + 41


print(next(n for n in count() if not isprime(p(n))))
```

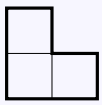
```
41
```

If you look closely, it is not so surprising that $p(41)$ is composite: after all, $p(41) > 41$ and $p(41)$ is divisible by 41. What is surprising is that for all integers $0 \leqslant n < 41$, the number $p(n)$ is prime.

> **For the curious.** It is known that the number 41 is the largest Euler's lucky number. These are positive integers k such that the value of the polynomial $n^2 - n + k$ is prime for all integers $0 \leqslant n < k$.

Problem 1.1.3 Multiple choice, 5 points

Can this figure be cut into several equal parts? (In other words, does such a partition *exist*?)

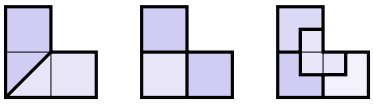


We call two parts equal if they can be matched by rotating, shifting, and reflecting. Cutting can be done *not* only along the grid lines.

Mark the correct statements:

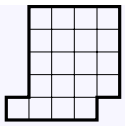
- ☐ It can be cut into two equal parts.
- ☐ It can be cut into three equal parts.
- ☐ It can be cut into four equal parts.

Yes, it can be cut into two, three, and four parts.



Problem 1.1.4 Multiple choice, 5 points

Now suppose cutting is only allowed along the grid lines. Can the following figure be cut into two or three equal parts?

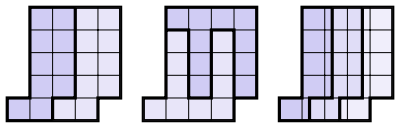


Mark the correct statements:

☐ It can be cut into two equal parts.

☐ It can be cut into three equal parts.

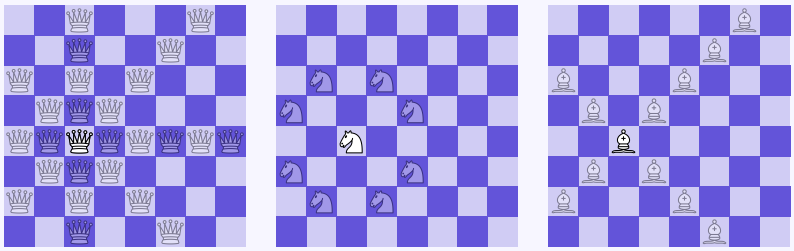
Yes, it can be cut into two parts (below we provide two ways). But it cannot be cut into three parts along the grid lines, as the figure consists of twenty cells, and twenty is not divisible by three. However, if cutting is not restricted to the grid lines, it can be cut into three parts.



Problem 1.1.5 Multiple choice, 5 points

Is it possible to place eight non-attacking queens on a chessboard? How about fifteen bishops? Or thirty-two knights?

The diagram below shows how these pieces move: the queen moves any number of cells vertically, horizontally, or diagonally; the knight moves in an L-shape; the bishop moves any number of cells diagonally.



Mark all correct statements.

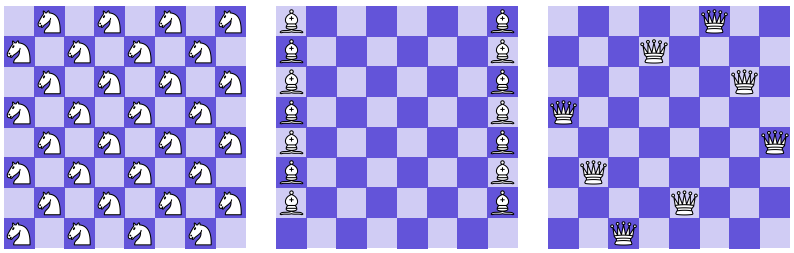
☐ One can place eight non-attacking queens.

☐ One can place thirty-two non-attacking knights.

☐ One can place fifteen non-attacking bishops.

Eight queens and thirty-two knights can be placed. To verify this, just look at the diagram below. It is easy to see directly that the queens do not attack each other. To verify

that no two knights attack each other, note that, on the one hand, they all stand on squares of the same color, and on the other hand, a knight only attacks squares of the opposite color. However, fifteen bishops cannot be placed on the board; we will soon prove this rigorously. But fourteen bishops can be placed, as shown in the diagram.



For the curious. The diagram above is a rigorous mathematical proof that the required number of pieces can be placed. However, the question remains: how to find these configurations? For example, finding a configuration of queens is not that simple. Especially if you need to place, for example, 20 queens on a 20×20 board. It is not easy to do this manually (try it!), nor is it easy to write a program for this task. At the same time, explicit configurations of n queens on an $n \times n$ board are known for all $n \geqslant 4$. For example, if n is an even number that does not leave a remainder of 2 when divided by 6, then the following configuration will work: for all $1 \leqslant i \leqslant n/2$, place queens in the cells $(i, 2i)$ and $(n/2 + i, 2i - 1)$.

1.1.2 Difficult Problems in Number Theory (Optional)

This section explores number theory questions that begin with "Does there exist a number…?"—some of the simplest to state, yet notoriously difficult to solve (and in some cases still open).

Treat these as curiosities rather than core material: each problem is worth just one point, so don't spend too much time on them. Feel free to skip this lesson entirely—nothing here will be needed later.

Problem 1.1.6

Number, 1 point

Pierre de Fermat conjectured that, for any $n \in \mathbb{Z}_{\geqslant 0}$, the number $2^{2^n} + 1$ is prime, and Leonhard Euler found the smallest n for which this is not the case (in times when there were no computers!). Find this value of n.

Pierre de Fermat (1607–1665)

Leonhard Euler (1707–1783)

For the curious. Numbers of the form $2^{2^n} + 1$ (for $n \in \mathbb{Z}_{\geqslant 0}$) are called *Fermat numbers*. There are many open problems around them:

- Are there infinitely many Fermat primes?
- Are there infinitely many composite Fermat numbers?
- Does a Fermat number exist that is not square-free?

The code below shows that for $n = 5$ the corresponding number is composite.

```
from sympy import isprime

for n in range(7):
    f = 2 ** (2 ** n) + 1
    print(n, f, isprime(f))
```

```
0 3 True
1 5 True
2 17 True
3 257 True
4 65537 True
5 4294967297 False
6 18446744073709551617 False
```

Problem 1.1.7

String, 1 point

Fermat's Last Theorem (formulated by Pierre de Fermat in 1637 and proved in 1994 by Andrew Wiles) states that the equation

$$a^n + b^n = c^n$$

has no solutions in positive integers for any integer $n \geqslant 3$.

In 1769, Leonhard Euler proposed a hypothesis generalizing Fermat's theorem and

stating that for $n \in \mathbb{Z}_{>2}$, representing an n-th power as a sum of n-th powers requires at least n terms. In particular, the equations

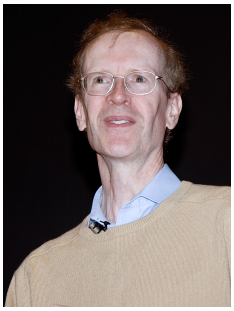
$$a^4 + b^4 + c^4 = d^4 \text{ and } a^5 + b^5 + c^5 + d^5 = e^5$$

have no solutions for $a, b, c, d, e \in \mathbb{Z}_{>0}$.

It turns out that Euler's hypothesis is false: there *exists* a counterexample. To disprove the hypothesis, enter four or five numbers (separated by spaces) that satisfy one of the equations above.

Pierre de Fermat
(1607–1665)

Leonhard Euler
(1707–1783)



Andrew Wiles
(born 1953)

As a counterexample to Euler's hypothesis, one may take the following set of numbers:

$$a = 95800,\ b = 217519,\ c = 414560,\ d = 422481\,.$$

It is easy to verify its correctness:

```
print(95800 ** 4 + 217519 ** 4 + 414560 ** 4 == 422481 ** 4)
```

```
True
```

It is known that this is the only such quadruple of numbers where all numbers are less than a million. For $n = 5$, the minimal counterexample will have smaller numbers—see the screenshot of one of the shortest scientific articles.

COUNTEREXAMPLE TO EULER'S CONJECTURE ON SUMS OF LIKE POWERS

BY L. J. LANDER AND T. R. PARKIN

Communicated by J. D. Swift, June 27, 1966

A direct search on the CDC 6600 yielded

$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

as the smallest instance in which four fifth powers sum to a fifth power. This is a counterexample to a conjecture by Euler [1] that at least n nth powers are required to sum to an nth power, $n > 2$.

REFERENCE

1. L. E. Dickson, *History of the theory of numbers*, Vol. 2, Chelsea, New York, 1952, p. 648.

For the curious. This counterexample can be found by exhaustive search. The code below simply iterates over all quintuplets of numbers $1 \leqslant a, b, c, d, e < 150$. As soon as the required quintuplet is found, it is printed and the program terminates.

```
from itertools import product

for a, b, c, d, e in product(range(1, 150), repeat=5):
    if a ** 5 + b ** 5 + c ** 5 + d ** 5 == e ** 5:
        print(a, b, c, d, e)
        break
```

This code will run for a long time. It can be sped up, for example, as follows:

1. If a quintuplet of numbers a, b, c, d, e satisfies the condition, then the numbers a, b, c, d can be permuted in any way. To avoid unnecessary iterations, we can agree that $a \leqslant b \leqslant c \leqslant d$. To iterate over such quadruples, we can use the `combinations` generator instead of `product`.
2. Precompute an array of fifth powers of numbers from the interval $[1, 149]$, and then check whether the sum of any four of its elements lies in this array.

The resulting optimized code will run much faster.

```
from itertools import combinations

powers = {i ** 5 for i in range(1, 150)}
print(next(c for c in combinations(powers, 4) if sum(c) in powers))
```

Problem 1.1.8

String, 1 point

Find integers a, b, c such that

$$a^3 + b^3 + c^3 = 42.$$

Such a triplet of numbers was found in 2019. More than a million hours of machine time were spent on the search. But verifying the result is instantaneous!

```
x = -80538738812075974
y = 80435758145817515
z = 12602123297335631

print(x ** 3 + y ** 3 + z ** 3)
```

```
42
```

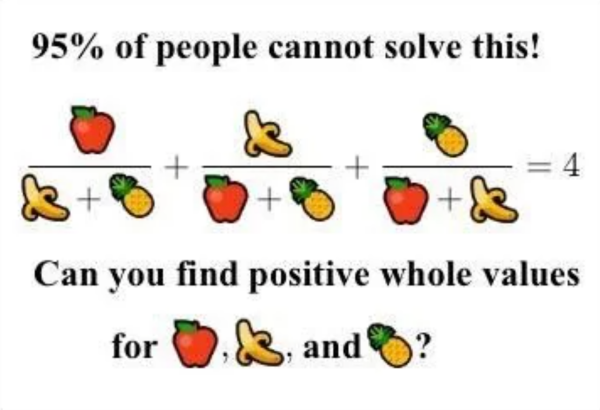
Problem 1.1.9

String, 1 point

Find positive integers a, b, c such that

$$\frac{a}{b+c} + \frac{b}{c+a} + \frac{c}{a+b} = 4\,.$$

You may encounter this problem on the internet in the following humorous form:



In decimal notation, the minimal triplet of such numbers will have more than eighty digits!

```
from fractions import Fraction

a = 1544768021087461664419513150199198374856643256695654317000266348982532020352779999
b = 3687513179412999982719781156522547482549297996897197099628313747163722463405557 9
c = 4373612677928697257861252602371390152816537558161613618621437993378423467772036

print(Fraction(a, b + c) + Fraction(b, c + a) + Fraction(c, a + b))
```

```
4
```

Problem 1.1.10 String, 1 point

Find a positive integer n such that the greatest common divisor of the numbers $n^{17} + 9$ and $(n+1)^{17} + 9$ is not one.

Below is the smallest such n.

```
from math import gcd


def f(n):
    return n ** 17 + 9


n = 84244329255928893292881973223089006724594204607924 33
print(gcd(f(n), f(n + 1)))
```

```
893658223791571665995096225335894563579345325693555 9
```

Finally, here are some examples of open problems in number theory (that is, questions to which no one knows the answer yet).

Goldbach's Conjecture. Does there exist an even number greater than two that cannot be expressed as the sum of two primes?

Collatz Conjecture. For a number $n \in \mathbb{Z}_{>0}$, consider the following sequence: if n is even, replace it with $n/2$, and if it is odd, replace it with $3n + 1$, and then continue the same process. Does there exist a number $n \in \mathbb{Z}_{>0}$ for which this sequence does not encounter one?

Mathematics may not be ready for such problems.

Paul Erdős (1913–1996), on the Collatz conjecture

Twin Primes. Does there exist an infinite number of pairs of prime numbers that differ by two?

Perfect Numbers. Does there exist an odd number that is equal to the sum of its proper positive divisors?

1.1.3 Proofs of Nonexistence

As we have seen, to prove that an object with the desired properties exists, it is enough to simply present such an object. And for this to be considered a rigorous mathematical proof of existence, it is not even necessary to explain how exactly this object was found or constructed.

But how do we prove that an object with the desired properties does not exist? There is no general recipe for this. The desired object may not exist for a variety of reasons. This can even manifest within a single problem. Let's give an example.

Problem 1.1.11 Multiple choice, 5 points

You and your friend are going on a hike and have determined a list of items to take. Now you want to distribute these items between two backpacks so that the weights of the backpacks are equal. For which of the given sets is this possible?

- ☐ 1, 2, 3
- ☐ 1, 2, 3, 4, 5, 6
- ☐ 1, 2, 3, 4, 5, 7
- ☐ 2, 4, 6, 8, 10, 12
- ☐ 1, 2, 3, 4, 5, 17

The sets $\{1, 2, 3\}$ and $\{1, 2, 3, 4, 5, 7\}$ can be split: $1 + 2 = 3$ and $4 + 7 = 1 + 2 + 3 + 5$. In the remaining three cases, it is not possible, and each time the reason is different:

- the sum of the numbers in the set $\{1, 2, 3, 4, 5, 6\}$ is odd;
- in the set $\{2, 4, 6, 8, 10, 12\}$ the sum of the numbers is even, but it is obtained from the previous set by doubling all the numbers;
- the set $\{1, 2, 3, 4, 5, 17\}$ has a number that is too large.

> **For the curious.** We have seen three different obstacles that prevent splitting a set of numbers into two sets with equal sums. Naturally, one might ask: is there such an exhaustive list of obstacles? Unfortunately, we still do not know the answer to this question. If such a list of (easily checkable) obstacles existed, we could solve the corresponding problem, known as the knapsack problem, algorithmically quickly. At the same time, no one knows whether a fast algorithm for this problem exists or not. The question of the existence of such an algorithm is the question of the equality of the complexity classes P and NP. This is one of the most important open questions in computer science and mathematics. The Clay Mathematics Institute has offered a prize of one million dollars for its solution.

A real number x is called *rational* if there exist integers p, q with $q \neq 0$ such that $x = \frac{p}{q}$. It is known that the number $\sqrt{2}$ is irrational: *there do not exist* integers p, q with $q \neq 0$ such that $\sqrt{2} = \frac{p}{q}$.

Problem 1.1.12 Choice, 5 points

Is this a correct proof?

> Assume $\sqrt{2}$ is rational. Then, $\sqrt{2} = \frac{a}{b}$. Without loss of generality, one may assume that this fraction is irreducible, that is, a and b do not share non-trivial divisors (that is, $\gcd(a, b) = 1$). Squaring both sides, one gets $2 = \frac{a^2}{b^2}$, hence $a^2 = 2b^2$. This implies that a^2 is even and hence a is even, that is, $a = 2k$ for some $k \in \mathbb{Z}$. Then, $a^2 = 4k^2 = 2b^2$, hence $b^2 = 2k^2$ and b^2 is even, and thus, b is also even. But then, $\gcd(a, b) \geqslant 2$, a contradiction.

☐ Yes, it is correct. ☐ No, it is incorrect.

> **For the curious.** There is also the following (on the one hand, humorous, on the other hand, correct) proof of the irrationality of the number $\sqrt[3]{2}$. Suppose $\sqrt[3]{2} = \frac{a}{b}$, then $b^3 + b^3 = a^3$, but this contradicts Fermat's Last Theorem!

As we have already seen, it is possible to place eight non-attacking queens, thirty-two non-attacking knights, or fourteen non-attacking bishops on a chessboard. Think about how to mathematically rigorously prove that it is not possible to place more pieces.

In other words, we want to prove that *there do not exist* such arrangements:

- nine non-attacking queens;
- fifteen non-attacking bishops;
- thirty-three non-attacking knights.

Problem 1.1.13 Choice, 5 points

Is this a correct proof?

> There cannot be more than eight queens because in each of the eight columns there cannot be more than one queen.

☐ Yes, it is correct. ☐ No, it is incorrect.

Problem 1.1.14 Choice, 5 points

Is this a correct proof?

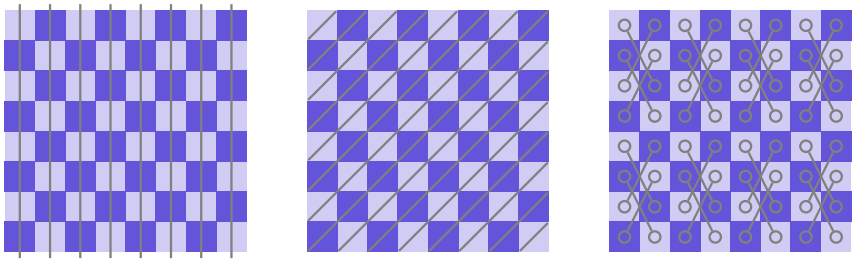
> Let's prove that it is not possible to place more than 32 non-attacking

knights on the board. A knight always attacks squares of the opposite color. Therefore, it is advantageous to place knights on all, for example, black squares. There will then be exactly thirty-two knights, and they will not attack each other because they are on squares of the same color. At the same time, all white squares are attacked, so it is not possible to place any additional knight. Therefore, it is not possible to place more than 32 knights.

☐ Yes, it is correct. ☐ No, it is incorrect.

Let's provide a mathematically rigorous proof.

- For the queens, this is the simplest: in each column, there can be no more than one queen, so there will definitely be no more than eight queens.
- To show that there cannot be more than 14 bishops, one can consider 15 diagonals in one direction (covering the entire board) and notice that it is not possible to place bishops on the first and last of them simultaneously.
- To show that more than 32 knights cannot be placed, one can divide all the squares of the board into pairs (see the figure), so that in each pair no more than one square can be used.



$\leqslant$ 8 queens $\leqslant$ 14 bishops $\leqslant$ 32 knights

For the curious. Another common method of proving nonexistence is the *diagonal argument*. Below, we provide the statements of several theorems that can be proven using this method.

Theorem (Cantor, 1874). *There is no bijection between* $\mathbb{Z}$ *and* $\mathbb{R}$.

Theorem (Turing, 1936). *The halting problem is algorithmically unsolvable: there is no computer program that would take as input the source code* P *of a program and the input* I *and (in finite time) determine whether* P *will halt on* I.

In other words, it is impossible to write a program that would tell us whether another program will loop indefinitely or not. If such a program existed, we

could potentially use it to prove Fermat's Last Theorem: we would simply run an infinite search that would stop upon finding the first counterexample. We write "potentially" here because such a program, even if it existed, could run for millennia and provide no practical benefit. In any case, Turing proved in 1936 that such a program does not exist. Interestingly, from this result, one can derive Gödel's first incompleteness theorem, which Gödel published five years earlier. Informally, it states that no matter which (sufficiently strong) system of axioms we choose, there will always be statements that we cannot prove or disprove within this system of axioms.

Theorem (Gödel's first incompleteness theorem, 1931). *If formal arithmetic is consistent, then there exists a formula in it that is neither provable nor refutable.*

Proving that something does not exist can be challenging. For most computational problems, it is an open problem whether a fast enough algorithm exists.

- Does there exist an algorithm that multiplies two n-bit numbers in $O(n)$ time?
- Does there exist an algorithm that multiplies two $n \times n$ matrices in $O(n^{2+o(1)})$ time (i.e., in time that grows only slightly faster than n^2, for example, $O(n^2 \log^5 n)$)?
- Does there exist an algorithm that checks whether a graph with n vertices has a Hamiltonian cycle in $n^{O(1)}$ time (i.e., in polynomial time with respect to n)?

1.1.4 Non-constructive Proofs of Existence

Existence proofs may also be *non-constructive*: in such arguments, we establish the existence of an object without explicitly exhibiting it. A common approach is *proof by contradiction*: we assume that the desired object does not exist and derive a contradiction.

For example, it is easy to show that at least one of the numbers $\sin^2(10^{1000})$ and $\cos^2(10^{1000})$ is at least $1/2$. Indeed, their sum equals one, so at least one of them must be at least $1/2$ (otherwise, if both were less than $1/2$, their sum would be less than one, a contradiction). However, determining which of the two satisfies this inequality is difficult (it would require very precise knowledge of the value of π).

To give another example, we prove that there exist irrational numbers x, y such that x^y is rational. To do this, consider two cases.

- The number $\sqrt{2}^{\sqrt{2}}$ is rational. Then we can take $x = \sqrt{2}$ and $y = \sqrt{2}$.

- The number $\sqrt{2}^{\sqrt{2}}$ is irrational. Then we can take $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$:

$$x^y = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^2 = 2\,.$$

Thus, we have proven that such x, y exist, but we have not presented a specific pair of such numbers (although we have significantly narrowed the search space).

For this problem, a constructive proof can also be given:

$$\left(\sqrt{2}\right)^{\log_2 9} = 3\,.$$

It is easy to show the irrationality of the number $\log_2 9$: suppose $\log_2 9 = a/b$ for positive integers a, b, then $9^b = 2^a$, but this is impossible, since these numbers have different parities.

Later we will see non-constructive existence proofs for which no constructive analogs are still known.

> **For the curious.** But what about $\sqrt{2}^{\sqrt{2}}$? Is it irrational? Yes, it is even transcendental (it is not a root of any non-zero polynomial with integer coefficients). This follows from the Gelfond–Schneider theorem, which provides a positive answer to Hilbert's seventh problem.

Another popular method for non-constructive proofs is the *pigeonhole principle*. It states that if there are $n + 1$ pigeons in n holes, then one of the holes must contain more than one pigeon. As you can see, we show that a hole with more than one pigeon exists, but we do not explicitly present it. A more general statement is: if $nm + 1$ objects are divided into n classes, then one of the classes must contain at least $m + 1$ objects.

Problem 1.1.15 Multiple choice, 5 points

Mark all correct statements.

☐ Among any five hundred people, there are two born on the same day of the year.

☐ Among any 101 integers, there are two whose difference is divisible by 100.

☐ If 1001 balls are placed into 20 boxes, then at least one box contains at least 51 balls.

☐ If 1000 balls are placed into 20 boxes, then at least one box contains at least 51 balls.

Problem 1.1.16 Choice, 5 points

Consider the following problem.

> Prove that there exists a positive integer consisting only of zeros and ones that is divisible by 19.

Is the proof given below correct?

> Consider nineteen numbers:
>
> $$1, 11, 111, \ldots, \underbrace{111\cdots111}_{19\text{ digits}}.$$
>
> If one of them is divisible by 19, then this number suits us. Otherwise, all these numbers are not divisible by 19, that is, they give non-zero remainders when divided by 19. Since there are nineteen numbers, and only eighteen non-zero remainders, there will be two numbers giving the same remainder:
>
> $$\underbrace{111\cdots111}_{n\text{ digits}} \bmod 19 = \underbrace{111\cdots111}_{m\text{ digits}} \bmod 19$$
>
> for $n < m$ (the operation $\pmod{19}$ denotes the remainder when divided by 19). It remains to note that the difference of these numbers is divisible by 19 (since the remainders of the two numbers coincide) and has the form
>
> $$\underbrace{111\cdots111}_{m-n\text{ ones}}\underbrace{000\cdots000}_{n\text{ zeros}}.$$

☐ Yes, it is correct. ☐ No, it is not correct.

We conclude this section by proving a bit more involved result.

Theorem. *For any set of thirty seven-digit numbers, there are two disjoint non-empty subsets with the same sum.*

A more formal statement looks like this. Let S be a set of thirty seven-digit numbers:

$$S \subseteq \{1000000, 1000001, \ldots, 9999999\}$$

and $|S| = 30$. Then there will be non-empty sets $A, B \subseteq S$ such that $A \cap B = \varnothing$ and $\sum_{a \in A} a = \sum_{b \in B} b$.

For example, for the set $S = \{7, 31, 3, 12, 5, 14, 20\}$, such subsets would be: $A = \{7, 3, 12, 14\}$ and $B = \{31, 5\}$.

When there are not seven, but thirty numbers, and the numbers themselves are larger, it becomes more difficult to find such subsets. Here, for example, are thirty such numbers:

```
from random import randint, seed

seed(14)

for i in range(30):
    print(randint(10 ** 6, 10 ** 7 - 1), end=' ')
    if i % 6 == 5:
        print()
```

```
2792285 9843470 5142886 5548558 5291201 5882438
2218459 8545410 6083294 8828556 7655079 7607029
2986415 5421072 4745783 6295863 7006791 5368826
7051040 9661551 3511608 3701978 5619271 3767372
1177927 2173344 3064039 6655032 1466196 2395998
```

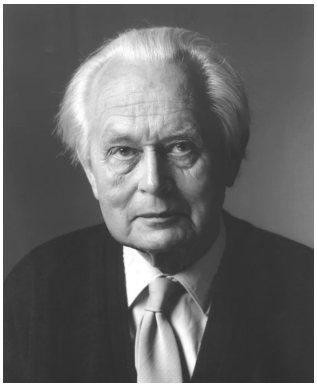
It is difficult to find the required two subsets manually, and it is also challenging to write a program that would find the required two subsets quickly. However, it can be proven that such two subsets exist, *without explicitly presenting them*. Such proofs are called *non-constructive*.

Proof. The total number of non-empty subsets is $2^{30} - 1 > 10^9$ (the total number of subsets, including the empty one, is 2^{30}, because each element can be included or not). On the other hand, the sum of the elements of each such subset is not more than $30 \cdot 10^7 = 0.3 \cdot 10^9$. Therefore, there will be two non-empty sets with the same sum. If they intersect, then the repeated elements can be removed from both sets. It is easy to see that non-empty disjoint sets with the same sum will be obtained. □

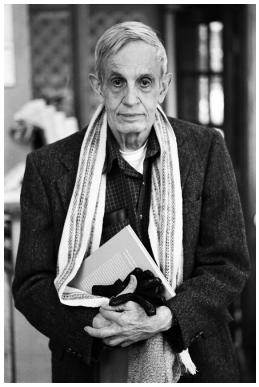
1.1.5 More Complex Non-constructive Proofs of Existence (Optional)

For some games, it can be non-constructively shown that the second player cannot have a winning strategy. This is done roughly as follows: assuming that the second player has a winning strategy, we show that the first player also has a winning strategy, and we arrive at a contradiction. If it can also be shown that there are no draws in the game, then we immediately get that the first player has a winning strategy. At the same time, constructing an explicit strategy for the first player can be difficult.

We will illustrate this with the game of Hex. This is a game with relatively simple rules, invented by Piet Hein in 1942 and reinvented by John Nash in 1948. It turns out that in this game, the first player has a winning strategy, but no one knows it!



Piet Hein
(1905–1996)



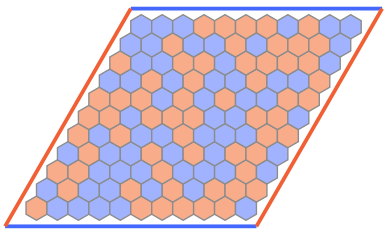
John Nash
(1928–2015)

The rules of the game are the following. Two players place their colored pieces on a board of hexagons of size $n \times n$. In the common variant of this game, $n = 11$. The goal of the player is to connect the sides of their color with a path of hexagons of their own color.

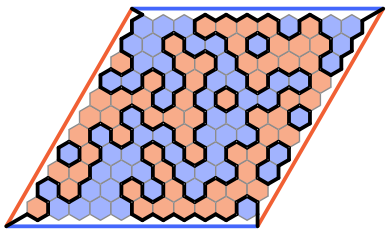
It turns out that the first player has a winning strategy! We will prove this non-constructively, that is, without presenting an explicit strategy.

Theorem. *In the Hex game, the first player has a winning strategy.*

Proof. **No draws.** First, let's prove that there are no draws in the game. Consider a board completely filled with cells. Think of the four sides of the field as four additional colored regions: two blue ones and two red ones. To such a filled board, correspond the following graph. The vertices of this graph are all the vertices of the hexagons, as well as the four corners of the field. Draw an edge between two neighboring vertices whenever the two regions on the sides of this edge have different colors. In other words, the graph traces the boundary between red and blue regions, including the boundary between a cell and an outer side of the field. Then, each internal vertex has a degree of zero or two, and the four corner vertices have a degree of one. This means that our graph is a set of cycles, as well as two paths with ends in the corner cells, and these paths do not intersect.

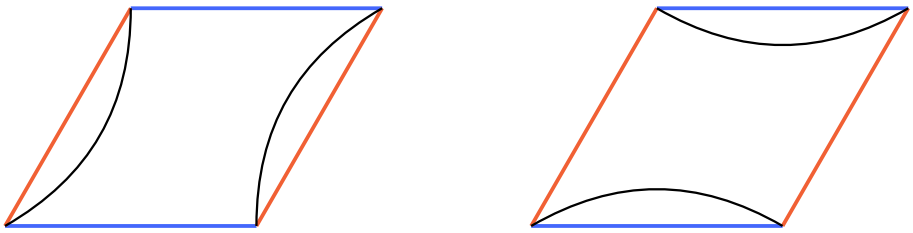


A filled board



The corresponding graph

The way these corner vertices are connected by paths determines the winner.



Blue player wins

Red player wins

In the example above, the corner-to-corner paths connect the corners vertically, so the blue player is the winner.

Stealing a strategy. Now, assume that the second player has a strategy. The first player can "borrow" it from the second player as follows: the first player makes their initial move arbitrarily, and then starts playing according to the second player's strategy. This guarantees a win for the first player, which cannot be (since the second player has a strategy). Therefore, the second player cannot have a winning strategy, and since there is no draw, the first player must have one.

In this reasoning, we implicitly used the fact that the initial move of the first player (which they make arbitrarily) cannot prevent them from winning. More carefully, it should be said as follows: when the first player needs to make a move according to the second player's strategy, they mentally erase one of their pieces (so that the number of pieces on the field becomes odd) and see how the second player would move in such a situation; if the second player's strategy suggests placing a piece in the spot where the first player's (mentally erased) piece already stands, the first player simply places a piece in an arbitrary spot again.

□

Another common way to non-constructively prove the existence of something is to take a random object and prove that it has the desired properties with non-zero probability. This is called the *probabilistic method*.

Theorem (Erdős, 1963)**.** *For any* $k \in \mathbb{Z}_{\geqslant 1}$*, there exists a tournament* $G(V, E)$ *in which for any* k *vertices* $u_1, \dots, u_k \in V$*, there exists a vertex* $v \in V$ *such that* $(v, u_i) \in E$ *for all* $i \in [k]$*. (That is, for any* k *teams, there is a team that has beaten all of them.)*

Proof. Consider a random tournament $G(V, E)$ on n vertices: the outcome of each game (the direction of each edge) is chosen randomly with probability $1/2$. For a set $S \subseteq V$ of size k, let A_S denote the event "no $v \notin S$ has beaten all of S". Each $v \notin S$ beats all teams from S with probability 2^{-k}. There are $n - k$ such v and these probabilities are independent, so

$$\Pr[A_S] = (1 - 2^{-k})^{n-k}.$$

Then

$$\begin{aligned}
\Pr\left[\bigcup_S A_S\right] &\leqslant \sum_S \Pr[A_S] && \text{(union bound)}\\
&= \binom{n}{k}(1-2^{-k})^{n-k}\\
&\leqslant n^k(1-2^{-k})^{n-k} && \left(\binom{n}{k} \leqslant n^k\right)\\
&\leqslant n^k\left(e^{-2^{-k}}\right)^{n-k} && (1+x \leqslant e^x,\ x=-2^{-k})\\
&= n^k e^{-\frac{n-k}{2^k}}\\
&= e^{\frac{k}{2^k}} \cdot \frac{n^k}{e^{\frac{n}{2^k}}}.
\end{aligned}$$

Recall now that k is fixed, and we can take n arbitrarily large. Then, $e^{\frac{k}{2^k}}$ is a constant, n^k is a polynomial, and $e^{n/2^k} = (e^{1/2^k})^n$ is an exponential function (as functions of n). An exponential function grows faster than a polynomial function, hence, for large enough n, the considered probability is less than one. Therefore, there exists a tournament in which the event A_S does not occur for any set S. □

For the curious. In continuous mathematics, there are also many proofs that can be classified as non-constructive.

Theorem (Bolzano — Cauchy, 1821)**.** *If a function continuous on a segment takes values of different signs at the ends of this segment, then at some point within the segment it equals zero.*

The Bolzano — Cauchy theorem is somewhat similar to the pigeonhole principle: it provides certain conditions under which the desired object (a hole with more than one pigeon or a point where the function equals zero) must exist. These two statements are non-constructive because they do not specify which object has the desired property.

Theorem (Borsuk — Ulam, 1933)**.** *For any continuous mapping of an n-dimensional sphere into n-dimensional Euclidean space, there exist two diametrically opposite points on the sphere that have the same value.*

From this theorem, the following (discrete!) result can be derived.

Theorem. *Let G be an undirected graph on $3n$ vertices, which is the union of a Hamiltonian cycle and n disjoint triangles. Then, G has an independent set of size n.*

In 1888, Hilbert proved that there exist nonnegative polynomials (with real coefficients) that cannot be represented as a sum of squares of polynomials. The first explicit example of such a polynomial was constructed by Theodor Motzkin only in 1967:

$$P(x, y) = 1 + x^4y^2 + x^2y^4 - 3x^2y^2.$$

In 1900, David Hilbert formulated a list of twenty-three open problems, the seventeenth of which asked: can any nonnegative polynomial be represented as a sum of squares of rational functions? A positive non-constructive answer to this question was given by Emil Artin in 1927, and a constructive one by Charles Delzell in 1984.

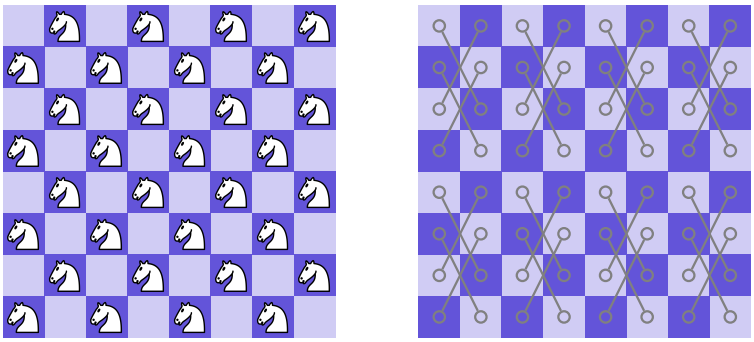
1.1.6 Proofs of Optimality

How can we be certain that a solution is not just good, but the best possible? In this section, we develop techniques for proving optimality—showing that no alternative solution can outperform the one at hand.

A proof of optimality consists of two parts. To prove that the optimal value of some objective function is M, one needs to:

1. prove that there *exists* an object for which the value of the objective function is M;
2. prove that there *does not exist* an object for which the value of the objective function is better than M (or, in other words: prove that *for any* other object, the value of the objective function will not be better than M).

For example, we already know that the maximum number of knights on a chessboard that do not attack each other is 32: there *exists* an arrangement with exactly 32 knights (see the figure below), and there *does not exist* an arrangement with more than 32 knights (because all 64 cells of the board can be divided into 32 pairs, so that each pair can contain at most one knight).



Problem 1.1.17 Number, 5 points

There are 90 cards with all two-digit numbers:

$$10, 11, 12, \ldots, 98, 99.$$

You can take some of these cards, and for each of them, you will receive one euro, but only if the following condition is satisfied: you must not have two cards in your hand whose numbers sum to 100. What is the maximum number of euros you can earn?

The answer is 50. Let's prove this. First, let's show that there *exists* such a set of fifty cards. It is not difficult to present it:

$$50, 51, \ldots, 98, 99.$$

Here, the sum of any two numbers is greater than one hundred.

Now let's show that there *does not exist* a set of more than fifty cards. Among our ninety cards, there are forty pairs, in each of which the sum is 100:

$$(10, 90), (11, 89), \ldots, (49, 51).$$

Thus, from these eighty numbers, it will not be possible to take more than forty numbers. There are still ten numbers left that did not fall into any pairs:

$$50, 91, 92, \ldots, 99.$$

It is easy to see that they can always be taken into the hand. Thus, it will be possible to take no more than $40 + 10 = 50$ cards.

Problem 1.1.18 Number, 5 points

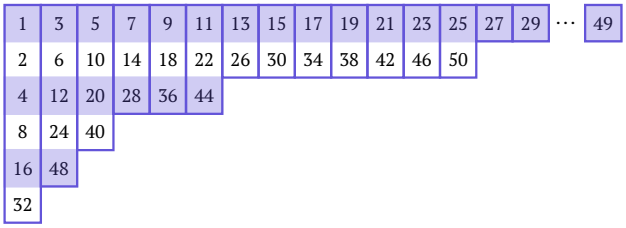
What is the maximum number of integers that can be marked among the numbers $1, 2, \ldots, 50$ so that none of the marked numbers equals twice another marked number?
Try it! Using set theory notation, one may state this problem as follows: find

$$\max\{|S|\colon S \subseteq \{1, \ldots, 50\} \text{ and } \forall x, y \in S,\ x \neq 2y\}.$$

It is easy to see that the numbers break into *independent* chains, in each of which no more than one of the two neighboring numbers can be taken. For example,

$$3, 6, 12, 24, 48.$$

Thus, from a chain of length k, at most $\lceil \frac{k}{2} \rceil$ elements can be taken (this is the number $k/2$ rounded up): you need to take every second element, starting from the first. In total, 33 elements will be collected. Below are the chains, with the elements that should be taken highlighted. The shown solution is not the only one: for example, from the first chain, you could take the elements $2, 8, 32$.

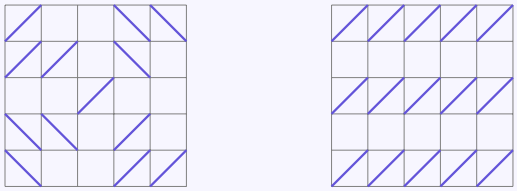


Finally, let's consider a more complex problem.

Problem 1.1.19 Number, 5 points

In each of the twenty-five squares of a 5×5 grid, a diagonal can be drawn in one of two ways. What is the maximum number of diagonals that can be drawn so that no two have common points? Try it!

Below are two examples of arrangements with thirteen and fifteen diagonals.



1.1.7 Application: Error Correcting Codes (Optional)

In this section, we will look at an important application: error-correcting codes. While analyzing such codes, we will prove existence, correctness, and optimality. At the same time, we only briefly touch on this topic: we will see several beautiful and classical examples, but this will only be the tip of the iceberg (there are separate courses and books dedicated to error-correcting codes).

Such codes are needed so that we can recover data even when this data is somehow corrupted. The nature of the damage can be very different:

- When transmitting data over a network, some packets may be lost or corrupted.
- The disk on which the data is stored may be physically damaged (e.g., scratched).
- A QR code in which the data is encoded may be smudged or poorly photographed.

In all these situations it is important for us to be able to recover the data despite possible damage. The methods of such recovery may vary. For example, if we are afraid of losing data from the disk, we can store one or several backups — and this is often done, although it is highly redundant. Or if some packets did not reach the network, we can resend them. Below we will be interested in how redundant the encoding must be so that it still allows effective error correction.

We call a *code* a set of binary strings of the same length: $C \subseteq \{0,1\}^n$. Elements of the set C are called *codewords*. These words will encode the messages we need to send. The

model is as follows: the message is sent through an unreliable channel, in which *symmetric substitution errors* may occur (a bit b is flipped, that is, replaced with $1 \oplus b$).

The *distance* of a code is defined as the minimum distance between its codewords:

$$d = \min_{u,v \in C} \{\mathrm{dist}(u, v) \colon u \neq v\}.$$

Here, dist is the *Hamming distance*, i.e., the number of positions in which the two strings differ:

$$\mathrm{dist}(u, v) = |\{i \in [n] \colon u_i \neq v_i\}|.$$

It is easy to see that this distance satisfies the *triangle inequality*: for any $u, v, w \in \{0,1\}^n$ we have

$$\mathrm{dist}(u, w) \leqslant \mathrm{dist}(u, v) + \mathrm{dist}(v, w).$$

A (n, k, d)-*code* is a code $C \subseteq \{0,1\}^n$ with distance d and cardinality at least 2^k. One says that an (n, k, d)-code *detects* $d - 1$ errors and *corrects* $\lfloor \frac{d-1}{2} \rfloor$ errors. This terminology is explained as follows. If instead of the codeword $x \in C$ we receive a word x' with at most $d - 1$ errors (that is, $\mathrm{dist}(x, x') \leqslant d - 1$), then we can surely detect that these errors occurred, since x' cannot coincide with another codeword: all of them are at least d away from x. If moreover $\mathrm{dist}(x, x') \leqslant \lfloor \frac{d-1}{2} \rfloor$, then from x' we can even recover the original codeword x (i.e., correct all errors), since all other codewords are surely farther from x' than x. Indeed, if there were another codeword $y \in C$ at distance at most $\lfloor \frac{d-1}{2} \rfloor$ from x', then, by the triangle inequality, we would have

$$\mathrm{dist}(x, y) \leqslant \mathrm{dist}(x, x') + \mathrm{dist}(x', y) \leqslant \left\lfloor \frac{d-1}{2} \right\rfloor + \left\lfloor \frac{d-1}{2} \right\rfloor \leqslant d - 1.$$

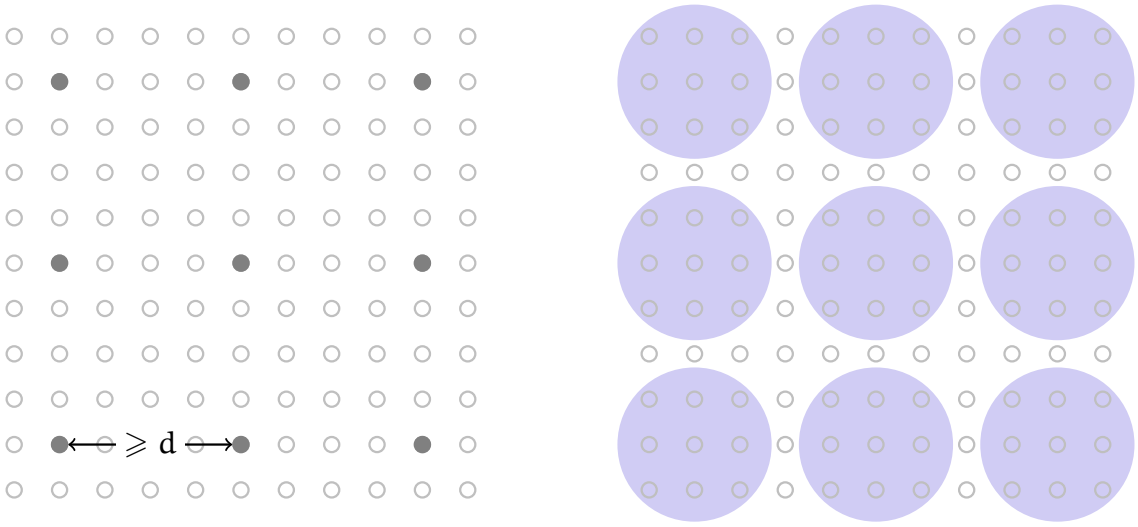
Let's illustrate this with a simple example. Consider the following code:

$$C = \{x \in \{0,1\}^n \colon x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0\}$$

(the codewords are length n strings with even sum of bits). It is easy to check that the distance of this code equals two. This code *detects* one error: if through a noisy channel, where at most one substitution error can occur, we receive a message x' with parity sum 0, then we know there was no error; if instead we get a message with parity sum 1, then we know there was one error (but do not know in which bit it occurred!). In this case, $|C| = 2^{n-1}$, that is, $k = n - 1$. This means that to *detect* one error, it is enough to extend the original message by just one bit (you can simply append the parity bit of the original message). But to *correct* (rather than just detect) one error, the message needs to be extended by at least $\log_2 n$ bits. Intuitively, this happens because detecting one error gives us one bit of information: error or no error. But correcting one error gives us $\log_2(n + 1)$ bits of information: whether an error occurred and, if so, in which of the n positions.

When designing a code, we want it to have as many codewords as possible (i.e., large k) at as large a distance from each other as possible (i.e., large d). Of course, the larger one of these parameters is, the smaller the other becomes. Below, we prove bounds on the

relationship of these parameters. Before formulating them, let's also give some geometric intuition. We want to select in $\{0,1\}^n$ as many points as possible with pairwise distance at least d. Or equivalently: we want to choose in $\{0,1\}^n$ as many pairwise disjoint balls of radius $\lfloor \frac{d-1}{2} \rfloor$ as possible.

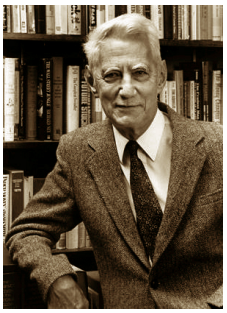


We call a *ball* of radius r centered at $u \in \{0,1\}^n$ the set of all strings at distance at most r from u:

$$\mathrm{Ball}(u,r) = \{v \in \{0,1\}^n \colon \mathrm{dist}(u,v) \leqslant r\}.$$

Its volume (cardinality) is denoted by $\mathrm{Vol}(r)$ (clearly, it does not depend on the center). The volume of a ball of radius $r = \alpha n$ for $\alpha \leqslant \frac{1}{2}$ can be estimated by the following formula (which follows from Stirling's formula):

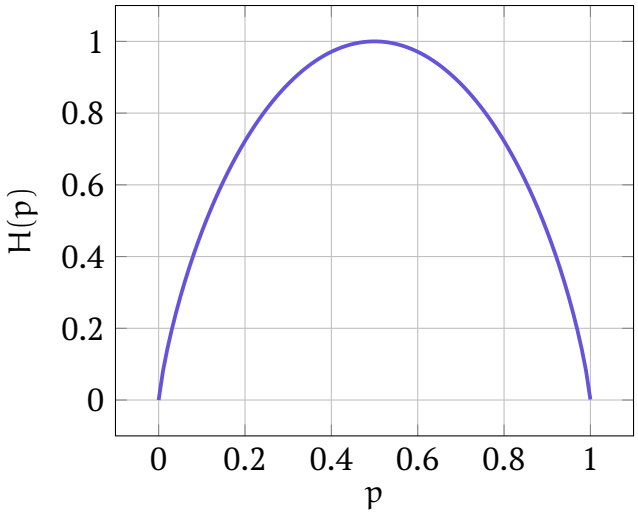
$$\mathrm{Vol}(r) = \sum_{i=0}^{\alpha n} \binom{n}{i} \approx 2^{H(\alpha)n}.$$



Claude Elwood Shannon (1916–2001)

The approximate equality here should be read as: up to a polynomial (in n) factor. The function $H(p)\colon [0,1] \to [0,1]$ is called the *Shannon entropy function*:

$$H(p) = -p \log_2 p - (1-p)\log_2(1-p).$$



Problem 1.1.20 Formula, 1 point

Find $\mathrm{Vol}(0^n, 1)$, that is, the volume of the ball of radius one centered at the word $00\ldots0$ (of length n).

Problem 1.1.21 Formula, 1 point

Find $\mathrm{Vol}(0^n, 2)$, that is, the volume of the ball of radius two centered at the word $00\ldots0$ (of length n).

Hamming and Gilbert Bounds

Theorem (Hamming bound). *For an* (n, k, d)*-code,*

$$2^k \cdot \mathrm{Vol}\left(\left\lfloor \frac{d-1}{2} \right\rfloor\right) \leqslant 2^n .$$

Proof. Indeed, 2^k disjoint balls of radius $\lfloor \frac{d-1}{2} \rfloor$ would otherwise not fit into $\{0,1\}^n$. □

From the Hamming bound one can formally derive what we discussed informally above: to be able to correct one error, the message must be lengthened by at least $\log_2 n$ bits. Indeed, $\mathrm{Vol}(1) = n + 1$ (the ball contains its center and n words obtained by flipping one coordinate). Therefore $2^k \cdot (n+1) \leqslant 2^n$. Taking logarithms gives $k \leqslant n - \log_2(n+1)$. Soon we will see how to achieve equality in this bound. For now we give a bound from the other side.

Theorem (Gilbert bound). *If*

$$(2^k - 1) \cdot \mathrm{Vol}\,(d-1) < 2^n ,$$

then there exists an (n, k, d)*-code.*

Proof. We add words to the code arbitrarily, ensuring that any two words have distance at least d. If at some step we cannot add a word, then the whole space is covered by balls of radius $d-1$. The inequality above guarantees that we make at least 2^k steps, i.e., construct the code. □

Linear Codes

A code $C \subseteq \{0,1\}^n$ is called *linear* if it is a *linear subspace* of $\{0,1\}^n$: if $u, v \in C$, then $u \oplus v \in C$ (in particular, this implies that the zero word belongs to any linear code: $u \oplus u = \mathbf{0} \in C$). A linear code can be specified by a *parity-check matrix* $H \in \{0,1\}^{(n-k)\times n}$ and then

$$C = \{x \in \{0,1\}^n \colon Hx = \mathbf{0}\}.$$

Some properties of linear codes:

1. The distance of a linear code is the weight of its minimal nonzero word:

   $$d = \min\{\|u\| \colon u \in C,\ u \neq \mathbf{0}\}.$$

   Indeed, for any $u, v \in C$ we have $\operatorname{dist}(u,v) = \operatorname{dist}(u \oplus v, \mathbf{0}) = \|u \oplus v\|$.

2. Any $\leqslant d-1$ columns of the matrix H are linearly independent, while there exist d linearly dependent columns. Indeed, if $\leqslant d-1$ columns were linearly dependent, then there would be a codeword of weight $\leqslant d-1$. Conversely, since there is a codeword of weight d, there are d linearly dependent columns.

Problem 1.1.22 Multiple choice, 1 point

Mark all linear codes.

- ☐ $C = \{x \in \{0,1\}^n \colon |x| \equiv_2 0\}$
- ☐ $C = \{000, 001, 010, 100\}$
- ☐ $C = \{0^n, 1^n\}$

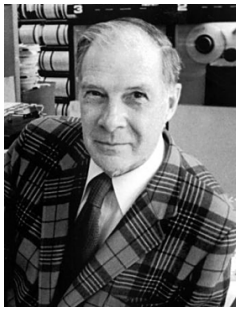
Problem 1.1.23 Formula, 1 point

What is the largest number of errors a code (containing at least two codewords) of length n can correct? Assume that n is odd.

Hamming Code

The basic example of a linear code is the *Hamming code*. It has many remarkable properties. Its only drawback is that its distance equals three, meaning it corrects only one

error. Nevertheless, it is based on a beautiful combinatorial construction. Understanding its properties is key to understanding more complex codes. To illustrate the properties, we use a toy example with $n = 7$, then discuss why these properties hold in general.



Richard Hamming (1915–1998)

The parity-check matrix of the Hamming code $C_7 \subseteq \{0,1\}^7$ consists of all possible nonzero vectors of length 3:

$$H_7 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

In other words, the set C_7 consists of words $u \in \{0,1\}^7$ that satisfy the system of linear equations:

$$u_4 \oplus u_5 \oplus u_6 \oplus u_7 = 0, \quad u_2 \oplus u_3 \oplus u_6 \oplus u_7 = 0, \quad u_1 \oplus u_3 \oplus u_5 \oplus u_7 = 0\,.$$

It is easy to see that these three equations (the rows of H_7) are linearly independent: indeed, u_4 appears only in the first equation, u_2 only in the second, u_1 only in the third (so no combination of these rows can sum to zero). It follows that $|C| = 2^{7-3} = 2^4 = 16$: informally, each equation reduces the degrees of freedom by one. The code distance equals three: there are no zero columns in the parity-check matrix, any two columns are different (and therefore cannot sum to zero), and one can easily find three linearly dependent columns (any two columns sum to another column in the matrix).

Now we list the remarkable properties of the Hamming code, introducing a few new definitions along the way.

Perfectness. A code is called *perfect* if it meets the Hamming bound. Clearly, the code C_7 has this property: balls of radius one centered at codewords of C_7 optimally pack the space $\{0,1\}^7$.

Systematic form. A code is called *systematic* if encoding is done by appending some information to the original message. The bits of the original message are called *information bits*, and the appended bits are *check bits*. More formally, an $(n, k, \cdot)$-code C is systematic if there exists an encoding $\phi\colon \{0,1\}^k \to C$ and coordinates

$1 \leqslant i_1 \leqslant \cdots \leqslant i_k \leqslant n$, such that for any message $m \in \{0,1\}^k$, the projection of its codeword on these coordinates equals m: $\phi(m)|_{i_1,\ldots,i_k} = m$.

Clearly, the code C_7 is systematic. Encoding can be done as follows. Denote the message by (u_3, u_5, u_6, u_7) (this numbering will be convenient for decoding). For any such message we can uniquely find the bits u_1, u_2, u_4:

$$u_4 = u_5 \oplus u_6 \oplus u_7, \quad u_2 = u_3 \oplus u_6 \oplus u_7, \quad u_1 = u_3 \oplus u_5 \oplus u_7\,.$$

Thus, the encoding is:

$$\phi(u_3, u_5, u_6, u_7) = (u_3 \oplus u_5 \oplus u_7, u_3 \oplus u_6 \oplus u_7, u_3, u_5 \oplus u_6 \oplus u_7, u_5, u_6, u_7)\,.$$

In fact, any linear code is systematic. One can express one variable from each parity-check equation, substitute it into the others, and so on.

Simple encoding. This is essentially described in the previous point: the encoding is as simple and efficient as possible.

Simple decoding. Decoding is not only simple and efficient but also elegant. Given a word $v \in \{0,1\}^7$ differing from a codeword in at most one bit, compute the *syndrome* $s = H_7 v$: if $s = \mathbf{0}$, there is no error; otherwise, s is the binary representation of the erroneous bit!

Example: suppose we receive $v \in \{0,1\}^7$ with $s = (0,1,1)^T$. Then s is the binary representation of three. Take $e = (0,0,1,0,0,0,0)$, where the 1 is at position three. Clearly, $H_7 e = s$. Therefore,

$$H_7(v \oplus e) = H_7 v \oplus H_7 e = s \oplus s = \mathbf{0}\,,$$

meaning $v \oplus e$ is the original codeword.

```
from itertools import product


def syndrome(u):
    return ((u[3] + u[4] + u[5] + u[6]) % 2,
            (u[1] + u[2] + u[5] + u[6]) % 2,
            (u[0] + u[2] + u[4] + u[6]) % 2)


def is_code_word(u):
    return syndrome(u) == (0, 0, 0)


def min_non_zero_weight(code):
    return min(sum(u) for u in code if sum(u) > 0)


def encode(m):
    return ((m[0] + m[1] + m[3]) % 2,
            (m[0] + m[2] + m[3]) % 2,
            m[0],
```

```
            (m[1] + m[2] + m[3]) % 2,
            m[1],
            m[2],
            m[3])


def decode(v):
    s = syndrome(v)
    if s == (0, 0, 0):
        return v

    i = s[2] + 2 * s[1] + 4 * s[0]
    u = list(v)
    u[i - 1] = (u[i - 1] + 1) % 2
    return u


hamming_code = [u for u in product(range(2), repeat=7) if is_code_word(u)]
print('Length of Hamming code:', len(hamming_code))
print('Minimum non-zero weight:', min_non_zero_weight(hamming_code))
print('Encode (1, 0, 1, 0):', encode((1, 0, 1, 0)))

for word in ((1, 0, 1, 1, 0, 1, 0), (1, 0, 1, 0, 0, 1, 0), (1, 1, 1, 1, 0, 1, 0)):
    print(f'Decode {word}: {decode(word)}')
```

```
Length of Hamming code: 16
Minimum non-zero weight: 3
Encode (1, 0, 1, 0): (1, 0, 1, 1, 0, 1, 0)
Decode (1, 0, 1, 1, 0, 1, 0): (1, 0, 1, 1, 0, 1, 0)
Decode (1, 0, 1, 0, 0, 1, 0): [1, 0, 1, 1, 0, 1, 0]
Decode (1, 1, 1, 1, 0, 1, 0): [1, 0, 1, 1, 0, 1, 0]
```

Now, for the general case. For $n = 2^t - 1$, the Hamming code is constructed the same way and has all the same properties. In this case, H_n contains all distinct nonzero columns of height $t = \log_2(n+1)$. Then $|C_n| = 2^{n-t} = \frac{2^n}{n+1}$. Hence, $|C_n|$ balls of radius one (each containing $n+1$ points of $\{0,1\}^n$) fill $\{0,1\}^n$ completely, so the code is perfect. As any linear code, C_n is systematic: the check bits are conveniently chosen as those at positions with binary representation having exactly one 1. Thus, encoding a message of length $n-t$ appends t checksums. In decoding, the syndrome is still the binary representation of the erroneous bit.

For $2^{t-1} \leqslant n < 2^t - 1$, the Hamming code is no longer perfect (here, H_n has n columns of height t, corresponding to numbers from 1 to n), but no other code can be perfect for distance three: for that, 2^n must be divisible by $n+1$. This means some points of $\{0,1\}^n$ will be at distance greater than one from any codeword. In such words the error cannot be corrected, but it is not required. All other properties remain: the code is still systematic, simply encoded, and simply decoded (for $e \in \{0,1\}^n$, $\|e\| = 1$, we must have $H_n e = s$; if H_n has no column equal to s, the received message cannot be decoded).

Problem 1.1.24 String, 1 point

The parity-check matrix for the code $C \subseteq \{0,1\}^6$ is defined as follows:

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Enter the following seven values, separated by commas.

1. The number k for this code,
2. The number d for this code,
3. The codeword for the message 011,
4. The message $m \in \{0,1\}^3$ if the codeword is 010100 (the code can be corrupted),
5. Is this code linear (1 if yes, 0 if no),
6. Is this code perfect (1 if yes, 0 if no),
7. Is this code systematic (1 if yes, 0 if no).

Reed–Muller code

The Reed–Muller code allows one to achieve higher distances.

Theorem. *Let* $0 \leqslant r \leqslant m$. *Then,*

$$C_{r,m} = \{\textit{truth table of } P\colon P \in \mathbb{F}_2[x_1, \ldots, x_m],\ P \textit{ is multilinear},\ \deg(P) \leqslant r\}$$

is a linear $(n = 2^m, k = \sum_{t \leqslant r} \binom{m}{t}, d = 2^{m-r})$ *code. Encoding is polynomial-time.*

Before proving, consider a toy example. Let $m = 2$. Here we use polynomials in two variables. The set $C_{1,2}$ is shown below, its distance is two. The set $C_{0,2}$ consists of its first two vectors, distance is four.

x_1	x_2	0	1	x_1	$1 \oplus x_1$	x_2	$1 \oplus x_2$	$x_1 \oplus x_2$	$1 \oplus x_1 \oplus x_2$
0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0
1	1	0	1	1	0	1	0	0	1

Proof. **Linearity.** The sum of two polynomials of degree $\leqslant r$ has degree $\leqslant r$.

Dimension. Basis: truth tables of all multilinear monomials of degree $\leqslant r$:

$$\{1\} \cup \{x_1, \ldots, x_m\} \cup \{x_1x_2, \ldots, x_{m-1}x_m\} \cup \cdots \cup \{x_{i_1} \cdots x_{i_r} \colon 1 \leqslant i_1 < \cdots < i_r \leqslant m\}.$$

Hence $k = \sum_{t \leqslant r} \binom{m}{t}$.

Encoding. Map $\phi \colon \{0,1\}^k \to \{0,1\}^n$: input bits are coefficients of a degree-$\leqslant r$ polynomial; output is its truth table.

Distance. Since the code is linear, it suffices to show any nonzero codeword has weight $\geqslant 2^{m-r}$, which follows from the lemma below.

□

Lemma. *Let $P \in \mathbb{F}_2[x_1, \ldots, x_m]$ be a nonzero multilinear polynomial of degree $\leqslant r$. Then,*

$$|\{x \in \{0,1\}^m \colon P(x) = 1\}| \geqslant 2^{m-r}.$$

Proof. Induction on m. Base: $m = 1$, then $P \in \{1, x_1, 1 \oplus x_1\}$, checked directly. Step $m - 1 \to m$: without loss of generality, assume P depends on x_m. Then

$$P(x) = Q(x_1, \ldots, x_{m-1}) \oplus x_m R(x_1, \ldots, x_{m-1}),$$

where $R \neq 0$, $\deg R \leqslant r - 1$. By induction, $R = 1$ on at least 2^{m-r} points. Each such $(v_1, \ldots, v_{m-1})$ extends to $(v_1, \ldots, v_{m-1}, 1 \oplus Q(v))$, and

$$P(v_1, \ldots, v_{m-1}, 1 \oplus Q(v)) = Q \oplus 1 \oplus Q = 1.$$

□

Reed–Solomon Code

For the Reed–Solomon code we need a larger field. So, let $\mathbb{F}_q$ be a field of size q. Now the symbols of our codewords will be elements of $\mathbb{F}_q$ (while earlier they were bits, i.e., elements of $\mathbb{F}_2$). The definitions of various code properties generalize from the case $q = 2$ naturally. We call a code the set $C \subseteq \mathbb{F}_q^n$, the distance of the code is defined as $d = \min\{d_H(u, v) \colon u \neq v \in C\}$, and the dimension as $\lfloor \log_q |C| \rfloor$. We call a code linear if it is a linear space: if $u, v \in C$, then $\alpha u + \beta v \in C$ for any $\alpha, \beta \in \mathbb{F}_q$. It is easy to see that the distance is still equal to the minimum weight of a nonzero codeword.

Reed–Solomon codes are actively used in practice: for data storage (in particular, on CDs and DVDs), in barcodes, and in space communications. As we will see below, for given n and k they achieve the optimal distance. Their main drawback is that they use an alphabet of large size.

Theorem. *Let $k \leqslant n \leqslant q$ and let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be pairwise distinct elements. Then*

$$C = \{(P(\alpha_1), \ldots, P(\alpha_n)) \colon P \in \mathbb{F}_q[x], \deg(P) < k\}$$

is a linear $(n, k, d = n - k + 1)$-code. Encoding can be performed in polynomial time.

Note that this code achieves the *Singleton bound*: for any (n, k, d)-code over an alphabet of size q the inequality $d \leqslant n - k + 1$ holds. This is easy to prove: since we have at least q^k (encoded) messages, some two of them must coincide in the first $k-1$ symbols; thus the distance between these two codewords is at most $n - (k-1)$. Hence, for given n and k this code achieves the optimal distance.

Proof. **Linearity.** Clearly the code is linear:

$$\beta_1 \cdot (P_1(\alpha_1), \ldots, P_1(\alpha_n)) + \beta_2 \cdot (P_2(\alpha_1), \ldots, P_2(\alpha_n)) \\ = ((\beta_1 P_1 + \beta_2 P_2)(\alpha_1), \ldots, (\beta_1 P_1 + \beta_2 P_2)(\alpha_n)) .$$

Dimension. If $P_1 \not\equiv P_2$, then the vectors $(P_1(\alpha_1), \ldots, P_1(\alpha_n))$ and $(P_2(\alpha_1), \ldots, P_2(\alpha_n))$ are distinct: otherwise the polynomial $P_1 - P_2$ would have at least n roots, which is impossible since $\deg(P_1 - P_2) < k \leqslant n$. Hence, the dimension of the code is k.

Encoding. Encoding $\phi\colon \mathbb{F}_q^k \to \mathbb{F}_q^n$ is simple: the given k elements of $\mathbb{F}_q$ are considered as the coefficients of the polynomial P. We then compute the values of this polynomial at the required points.

Distance. A nonzero polynomial of degree less than k can have at most $k-1$ roots. Therefore, any nonzero vector $u \in C$ has at least $n - k + 1$ nonzero components.

□

Theorem. *Decoding the Reed–Solomon code can be done in polynomial time.*

Proof. We are given a function $\tilde{P}$ which differs from P in at most $e = \lfloor \frac{n-k}{2} \rfloor$ positions. Consider the error vector:

$$D(x) = \prod_{i\colon P(\alpha_i) \neq \tilde{P}(\alpha_i)} (x - \alpha_i) .$$

Let $Q(x) = P(x)D(x)$ — a polynomial of degree less than $k + e$. It is easy to see that

$$Q(x) = \tilde{P}(x)D(x) \text{ for all } x \in \{\alpha_1, \ldots, \alpha_n\}.$$

In this equation we know $\alpha_1, \ldots, \alpha_n$ and the values of $\tilde{P}$ at these points. But we do not know the coefficients of the polynomials Q and D. Hence, the equality can be viewed as a system of linear equations for these coefficients. We know it has a solution, and we can find it. Nobody guarantees, however, that we find the original polynomials Q and D. So let us assume we found polynomials $\tilde{Q}$ and $\tilde{D}$ such that $\deg(\tilde{Q}) < k + e$, $\deg(\tilde{D}) \leqslant e$ and $\tilde{Q}(x) = \tilde{P}(x)\tilde{D}(x)$ for all $x \in \{\alpha_1, \ldots, \alpha_n\}$. It turns out that even if we found different polynomials, it is enough to divide $\tilde{Q}$ by $\tilde{D}$ to obtain P! Indeed, consider the polynomials $P\tilde{D}$ and $\tilde{Q}$. We know that $\tilde{Q}$ coincides with $\tilde{P}\tilde{D}$ at all n points, and P coincides with $\tilde{P}$ at least in $n - e$ points. Therefore, $P\tilde{D}$ and $\tilde{Q}$ coincide in at least $n - e \geqslant k + e$ points, and hence they are equal. □

Problem 1.1.25 Programming, 1 point

Implement decoding for the Reed–Solomon code.

The first line contains integers $1 \leqslant k \leqslant n \leqslant q \leqslant 5000$, where q is prime. The next line contains a codeword consisting of n elements of $\mathbb{F}_q$ (space-separated), encoding a message of length k. The encoding procedure is based on evaluating a polynomial at the points $1, 2, \ldots, n \in \mathbb{F}_q$. It is guaranteed that the number of errors in the codeword does not exceed $\lfloor \frac{n-k}{2} \rfloor$. Output the original message of length k.

1.1.8 Theory Problems

Basic Problems

Problem 1.1.26 10 points

Does there exist a sequence of 10 numbers such that the sum of any five consecutive numbers is positive, whereas the sum of any seven consecutive numbers is negative?

Hint. It *is* possible!

Problem 1.1.27 10 points

Is it possible to place the numbers -1, 0, and $+1$ in a 6×6-grid so that the sums of all six rows, six columns, and the two main diagonals are pairwise distinct?

Hint. One needs 14 different sums.

Problem 1.1.28 10 points

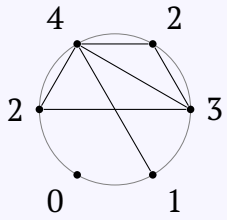
In the nodes of an integer grid, five points are marked. Prove that at least one of the segments connecting these points passes through a grid node distinct from its endpoints.

Hint. Consider a segment joining points (x_1, y_1) and (x_2, y_2). Its mid-point is $((x_1 + x_2)/2, (y_1 + y_2)/2)$. This is a grid point if x_1 and x_2 have the same parity as well as y_1 and y_2 have the same parity.

Problem 1.1.29 10 points

n points are marked on a circle, and some pairs of these points are connected by segments. Prove that there are two points having the same number of incident segments.

For example, in the figure below, there are two points with two adjacent segments.



Hint. If all the n numbers are different, then there are numbers 0 and $n-1$.

Problem 1.1.30 10 points

Five points are placed inside a unit square (a square with side length 1). Prove that there exist two points that are no more than $\frac{\sqrt{2}}{2}$ apart.

Hint. Divide the square into four smaller, equal-sized squares.

Problem 1.1.31 10 points

Prove that for any $n+1$ numbers selected from $[2n] = \{1, 2, \ldots, 2n\}$, there are two co-prime numbers.

Hint. Try to partition the set $[2n]$ into n "pigeonholes" such that any two numbers drawn from the same pigeonhole are co-prime.

Problem 1.1.32 10 points

Prove that for any $n+1$ numbers selected from $[2n] = \{1, 2, \ldots, 2n\}$, there are two numbers such that one of them divides the other one.

Hint. Consider classifying numbers based on their prime factorization. Specifically, try writing every integer x in the form $x = 2^k \cdot m$, where m is an odd number. How many possible values can the odd part, m, take?

Problem 1.1.33 10 points

Prove that in any sequence of n integers, there exists a continuous subsequence whose sum is divisible by n. (Formally: Let $a_1, a_2, \ldots, a_n$ be a sequence of integers; then there exist $1 \leqslant l \leqslant r \leqslant n$ such that $a_l + a_{l+1} + \cdots + a_r$ is divisible by n.) For example, in the sequence $(7, 2, 4, 9, 7)$, there are two such subsequences (whose sum is divisible by five): $(2, 4, 9)$ and $(4, 9, 7)$.

What are their remainders modulo n?

Hint. Consider $n+1$ prefix sums:

$$S_0 = 0, S_1 = a_1, S_2 = a_1 + a_2, \cdots, S_n = a_1 + \cdots + a_n.$$

Problem 1.1.34 10 points

Prove that, for any $x_1, \ldots, x_{11} \in \mathbb{Z}$, there exist $\alpha_1, \ldots, \alpha_{11} \in \{-1, 0, 1\}$ not all being zero, such that

$$2025 \text{ divides } \alpha_1 x_1 + \cdots + \alpha_{11} x_{11}.$$

Hint. Consider all 2^{11} linear $0/1$-combinations of the x values.

Problem 1.1.35 15 points

A chessmaster has 77 days to prepare for a tournament. She desires to play at least one game every day but not more than 132 games in total. Prove that there exists a sequence of consecutive days during which she plays exactly 21 games.

Hint. Let g_i be the total number of games played up to and including day i. Consider the sequence $g_1, g_2, \ldots, g_{77}$ and the sequence $g_1+21, g_2+21, \ldots, g_{77}+21$.

Problem 1.1.36 15 points

What is the minimum number of colors needed to color all positive integers under a condition that n and $2n$ have different colors as well as n and $n+2$ have different colors (for all n)?

Hint. To establish a lower bound, attempt to find a small set of integers that must all have different colors from each other. For example, consider the numbers 4, 6, and 8.

Problem 1.1.37 15 points

A row of n coins is laid out: heads, tails, heads, tails, and so on. In one move, it is allowed to flip any number of consecutive coins. What is the minimum number of moves required to ensure that all the coins are heads up?

Hint. Consider the number of boundaries (i.e., the number of neighboring pairs of different sides) and how it changes when one flips the consecutive sequence of coins.

Problem 1.1.38 15 points

Prove that, for any $n, m \in \mathbb{Z}_{\geqslant 1}$, any sequence of distinct real numbers of length at least $mn + 1$ contains a monotonically increasing subsequence of length $m + 1$ or a monotonically decreasing subsequence of length $n + 1$.

Hint. For each element x_k, where $1 \leqslant k \leqslant mn + 1$, in the sequence, associate it with an ordered pair of integers (i_k, d_k). Here, i_k represents the length of the longest increasing subsequence ending at x_k, and d_k represents the length of the longest decreasing subsequence ending at x_k. What can you say about these pairs?

Advanced Problems

Problem 1.1.39 20 points

Let $S_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. Prove that $S_n \notin \mathbb{Z}$ for any $n \geqslant 2$.

Hint. Focus on the highest power of 2 that is less than or equal to n.

Problem 1.1.40 20 points, Dirichlet's approximation theorem, 1879

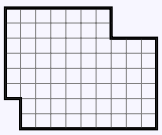
Prove that for any $x \in \mathbb{R}$ and $n \in \mathbb{Z}_{>0}$, there exists a rational number p/q, such that $1 \leqslant q \leqslant n$ and

$$\left| x - \frac{p}{q} \right| \leqslant \frac{1}{nq}.$$

Hint. The inequality can be rewritten as $|qx - p| \leqslant \frac{1}{n}$. This implies that we need to find a multiple of x, specifically qx (for $1 \leqslant q \leqslant n$), that is sufficiently close to an integer p.
Consider the fractional parts of the $n+1$ numbers $0, x, 2x, \ldots, nx$. Utilize the Pigeonhole Principle to demonstrate that two of these fractional parts must be arbitrarily close to each other.

Problem 1.1.41 20 points

Is it possible to cut this figure into two equal parts?



Hint. It *is* possible (though challenging).

Problem 1.1.42 20 points

Let $\mathcal{F} \subseteq 2^{[n]}$ be a family of subsets of an n-element set satisfying the following two properties:

- The size of each set in $\mathcal{F}$ is odd.
- The size of the intersection of any two distinct sets in $\mathcal{F}$ is even.

Find the maximum possible size of $\mathcal{F}$.

Problem 1.1.43 20 points

Let G be a bipartite graph with parts of size n that has a unique perfect matching. What is the maximum number of edges in G?

Hint. The answer is $\frac{n \cdot (n+1)}{2}$.

Optional Problems

Problem 1.1.44 20 points

Two players are playing the following game. On the board, there are integers from 1 to n. In each move, a player is allowed to cross out a number and all of its divisors. The player who cannot make a move loses. Who wins with optimal play?

Hint. Think how the first player can steal the strategy of the second player.

Problem 1.1.45 20 points

In the game "Chomp", players take turns breaking off pieces from a rectangular chocolate bar of size $n \times m$. On each turn, a player is allowed to choose any piece and break it off, as well as everything above and to the right of it (in other words, on a turn, you can choose a piece (x, y) and break off all the pieces (x', y') where $x' \geqslant x$ and $y' \geqslant y$). The player who eats the bottom-left piece loses. Who wins when playing optimally?

Hint. Think how the first player can use the strategy of the second player to win.

Problem 1.1.46 20 points

Compute, up to a polynomial factor, the minimum number of balls of radius $n/4$ needed to cover $\{0,1\}^n$.

Hint. For the upper bound, calculate how many *random* balls are sufficient.

Problem 1.1.47 20 points

Suppose C_1 is a (n, k_1, d_1) code and C_2 is a (n, k_2, d_2) code over $\mathbb{F}_2$. Let

$$C_3 = \{x \circ (x+y) : x \in C_1,\ y \in C_2\}.$$

Prove that C_3 is a $(2n, k_1 + k_2, \min\{2d_1, d_2\})$ code. Also, prove that if C_1, C_2 are linear and over $\mathbb{F}_q$, then C_3 is also linear.

Problem 1.1.48 30 points

Let C be a $(2d, \log m, d)$ code over $\mathbb{F}_2$. Prove that $m \leqslant 2d$.

Hint. Place the vectors in $\mathbb{R}^n$ such that any two vectors c_i and c_j, for $i \neq j$, form a large angle between them.