Ve203 Discrete Mathematics

Horst Hohberger

University of Michigan - Shanghai Jiaotong University
Joint Institute

Fall Term 2016

Office Hours, Email, TAs

- Please read the Course profile, which has been uploaded to the Files section on the Canvas course site.
- We will no longer use SAKAI at all for this course..
- My office is Room 218 in in the Law School Building.
- My email is horst@sjtu.edu.cn and I'll try to answer email queries within 24 hours.
- Office hours will be announced on Canvas.
- Please also make use of the Discussions page on Canvas for asking questions, making comments or giving feedback on the course.
- The recitation class schedule and the TA office hours and contact details will be announced on Canvas.

Coursework Policy

The following applies mostly to the paper-based homework at the beginning of this term:

- Hand in your coursework on time, by the date given on each set of course work. Late work will not be accepted unless you come to me personally and I find your explanation for the lateness acceptable.
- You can be deducted up to 10% of the awarded marks for an assignment if you fail to write neatly and legibly. Messiness will be penalized!
- You are encouraged to compose your coursework solutions in LaTeX. While this is optional, there will be a 10% bonus to the awarded marks for those assignment handed in as typed LaTeX manuscripts.

  LaTeXis open-source software for mathematical typesetting, and there are various implementations available. I suggest that you use Baidu or Google to find a suitable implementation for your computer and OS. LaTeXis widely used for writing theses and scientific papers, so it may be quite useful for you to learn it.

Use of Wikipedia and Other Sources; Honor Code Policy

When faced with a particularly difficult problem, you may want to refer to other textbooks or online sources such as Wikipedia. Here are a few guidelines:

- Outside sources may treat a similar sounding subject matter at a much more advanced or a much simpler level than this course. This means that explanations you find are much more complicated or far too simple to help you. For example, when looking up the "induction axiom" you may find many high-school level explanations that are not sufficient for our problems; on the other hand, wikipedia contains a lot of information relating to formal logic that is far beyond what we are discussing here.
- If you do use any outside sources to help you solve a homework problem, you are not allowed to just copy the solution; this is considered a violation of the Honor Code.

Use of Wikipedia and Other Sources; Honor Code Policy

- The correct way of using outside sources is to understand the contents of your source and then to write in your own words and without referring back to the source the solution of the problem. Your solution should differ in style significantly from the published solution. If you are not sure whether you are incorporating too much material from your source in your solutions, then you must cite the source that you used.
- You may cooperate with other students in finding solutions to assignments, but you must write your own answers. Do not simply copy answers from other students. It is acceptable to discuss the problems orally, but you may not look at each others' written notes. Do not show your written solutions to any other student. This would be considered a violation of the Honor Code.

Use of Wikipedia and Other Sources; Honor Code Policy

In this course, the following actions are examples of violations of the Honor Code:

- Showing another student your written solution to a problem.
- Sending a screenshot of your solution via QQ, email or other means to another student.
- Showing another student the written solution of a third student; distributing some student's solution to other students.
- Viewing another student's written solution.
- Copying your solution in electronic form (LaTeX source, PDF, JPG image etc.) to the computer hardware (flash drive, hard disk etc.) of another student. Having another student's solution in electronic form on your computer hardware.

If you have any questions regarding the application of the Honor Code, please contact me or any of the TAs.

Course Grade

Please note that the grading policy for this course has been updated. Please read the following passage carefully.

The course contains four grade components:

- Three examinations,
- Course work.

The course grade will be calculated from these components using the following weighting:

- First midterm exam: 25% (based on 25 marks in the exam)
- Second midterm exam: 25% (based on 25 marks in the exam)
- Final exam: 25% (based on 25 marks in the exam)
- Course work: 25% (based on 300 marks in the homework; there will be at least 350 total marks throughout the term)

The median course grade is expected to be a B.

Class Attendance and Absence for Medical Reasons

I do not formally require that you attend every class. However, if you are unable to attend a significant number of lecture, you should notify me. The following rules have been laid down by the Academic Office:

- A student who has been absent from studies for more than one week because of illness or other emergency should consult the program advisor. **[and also talk to me!]**
- Absence for illness should be supported by a hospital/doctor's certificate. A note that a student visited a medical facility is not sufficient excuse for missing an assignment or an exam. The note must specifically indicate that the student was incapable of completing an assignment or taking the exam due to medical problems.

Class Attendance and Absence for Medical Reasons

- **Late** medical excuses must satisfy the following criteria to be valid:
  - (i) The problem must be confirmed by the doctor to be so severe that the student could not participate in the exam.
  - (ii) The problem must have occurred so suddenly that it was impractical to contact me in advance.
  - (iii) The student must be in contact with me immediately after the exam with the required documentation.

LaTeX Policy and Textbook

As engineers, you are strongly encouraged to familiarize yourselves with a mathematical typesetting program called LaTeX. This is open-source software, and there are various implementations available. I suggest that you use Baidu or Google to find a suitable implementation for your computer and OS.

While the use of LaTeX is **optional**, there will be a **10% bonus to the awarded marks** for those assignments handed in as typed LaTeX manuscripts.

The main textbook for this course is

- Rosen, K. H., *Discrete Mathematics and its Applications*, 6th Ed., McGraw-Hill International Edition 2007.

Part I

Elementary Number Theory and Applications

Basic Concepts in Logic

Basic Concepts in Set Theory

Natural Numbers and Mathematical Induction

Equivalence Relations, Integers, Rationals

Functions, Sequences, Real Numbers

Divisibility Theory of the Integers

Prime Numbers and their Distribution

The Theory of Congruences

Factorization and Verifying Primality

Basic Concepts in Logic

Propositional Logic, Statements

A ***statement*** (also called a ***proposition***) is anything we can regard as being either ***true*** or ***false***. We do not define here what the words "statement", "true" or "false" mean. This is beyond the purview of mathematics and falls into the realm of philosophy. Instead, we apply the principle that "we know it when we see it."

Contrary to the textbook, we will generally not use examples from the "real world" as statements. The reason is that in general objects in the real world are much to loosely define for the application of strict logic to make any sense. For example, the statement "It is raining." may be considered true by some people ("Yes, raindrops are falling out of the sky.") while at the same time false by others ("No, it is merely drizzling.") Furthermore, important information is missing (Where is it raining? When is it raining?). Some people may consider this information to be implicit in the statement (It is raining ***here*** and ***now***.) but others may not, and this causes all sorts of problems. Generally, applying strict logic to colloquial expressions is pointless.

The Natural Numbers

Instead, our examples will be based on numbers. For now, we assume that the set of natural numbers

$$\mathbb{N} := \{0, 1, 2, 3, ...\}$$

has been constructed. In particular, we assume that we know what a ***set*** is! If n is a natural number, we write $n \in \mathbb{N}$. (We will later discuss naive set theory and give a formal construction of the natural numbers.) We also assume that on $\mathbb{N}$ we have defined the operations of addition $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and multiplication $\cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and that their various properties (commutativity, associativity, distributivity) hold.

The Natural Numbers

1.1.1. Definition. Let $m, n \in \mathbb{N}$ be natural numbers.

(i) We say that n is ***greater than or equal to*** m, writing $n \geq m$, if there exists some $k \in \mathbb{N}$ such that $n = m + k$. If we can choose $k \neq 0$, we say n is ***greater than*** m and write $n > m$.

(ii) We say that m ***divides*** n, writing $m \mid n$, if there exists some $k \in \mathbb{N}$ such that $n = m \cdot k$.

(iii) If $2 \mid n$, we say that n is even.

(iv) If there exists some $k \in \mathbb{N}$ such that $n = 2k + 1$, we say that n is odd.

(v) Suppose that $n > 1$. If there does not exist any $k \in \mathbb{N}$ with $1 < k < n$ such that $k \mid n$, we say that n is ***prime***.

It can be proven that every number is either even or odd and not both. We also assume this for the purposes of our examples.

Statements

1.1.2. Examples.

- "$3 > 2$" is a ***true statement***.
- "$x^3 > 10$" is not a statement, because we can not decide whether it is true or not.
- "the cube of any natural number is greater than 10" is a ***false statement***.

The last example can be written using a ***statement variable*** n:

- "For any natural number n, $n^3 > 10$"

The first part of the statement is a ***quantifier*** ("for any natural number n"), while the second part is called a ***statement form*** or ***predicate*** ("$n^3 > 10$").

A statement form becomes a statement (which can then be either true or false) when the variable takes on a specific value; for example, $3^3 > 10$ is a true statement and $1^3 > 10$ is a false statement.

Working with Statements

We will denote statements by capital letters such as $A, B, C, \ldots$ and statement forms by symbols such as $A(x)$ or $B(x, y, z)$ etc.

1.1.3. Examples.

- A: 4 is an even number.
- B: $2 > 3$.
- $A(n)$: $1 + 2 + 3 + \ldots + n = n(n + 1)/2$.

We will now introduce logical operations on statements. The simplest possible type of operation is a ***unary operation***, i.e., it takes a statement A and returns a statement B.

1.1.4. Definition. Let A be a statement. Then we define the ***negation of*** A, written as $\neg A$, to be the statement that is true if A is false and false if A is true.

Negation

1.1.5. Example. If A is the statement A: $2 > 3$, then the negation of A is $\neg A$: $2 \not> 3$.

We can describe the action of the unary operation $\neg$ through the following table:

A	$\neg A$
T	F
F	T

If A is true (T), then $\neg A$ is false (F) and vice-versa. Such a table is called a ***truth table***.

We will use truth tables to define all our operations on statements.

Conjunction

The next simplest type of operations on statements are ***binary operations***. The have two statements as arguments and return a single statement, called a ***compound statement***, whose truth or falsehood depends on the truth or falsehood of the original two statements.

1.1.6. Definition. Let A and B be two statements. Then we define the ***conjunction*** of A and B, written $A \wedge B$, by the following truth table:

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

The conjunction $A \wedge B$ is spoken "A and B." It is true only if both A and B are true, false otherwise.

Disjunction

1.1.7. Definition. Let A and B be two statements. Then we define the ***disjunction*** of A and B, written $A \vee B$, by the following truth table:

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

The conjunction $A \vee B$ is spoken "A or B." It is true only if either A or B is true, false otherwise.

1.1.8. Example.

- Let A: $2 > 0$ and B: $1 + 1 = 1$. Then $A \wedge B$ is false and $A \vee B$ is true.
- Let A be a statement. Then the compound statement "$A \vee (\neg A)$" is always true, and "$A \wedge (\neg A)$" is always false.

Proofs using Truth Tables

How do we prove that "$A \vee (\neg A)$" is an always true statement? We are claiming that $A \vee (\neg A)$ will be a true statement, regardless of whether the statement A is true or not. To prove this, we go through all possibilities using a truth table:

A	$\neg A$	$A \vee (\neg A)$
T	F	T
F	T	T

Since the column for $A \vee (\neg A)$ only lists T for "true," we see that $A \vee (\neg A)$ is always true. A compound statement that is always true is called a ***tautology***.

Correspondingly, the truth table for $A \wedge (\neg A)$ is

A	$\neg A$	$A \wedge (\neg A)$
T	F	F
F	T	F

so $A \wedge (\neg A)$ is always false. A compound statement that is always false is

Implication

1.1.9. Definition. Let A and B be two statements. Then we define the ***implication*** of B and A, written $A \Rightarrow B$, by the following truth table:

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

We read "$A \Rightarrow B$" as "A implies B," "if A, then B" or "A only if B". (The last formulation refers to the fact that A can not be true unless B is true.)

To illustrate why the implication is defined the way it is, it is useful to look at a specific implication of predicates: we expect the predicate

$$A(n)\colon n \text{ is prime} \Rightarrow n \text{ is odd}, \qquad n \in \mathbb{N}, \tag{1.1.1}$$

to be false if and only if we can find a prime number n that is not odd.

Implication

By selecting different values of n we obtain the following types of statements

- $n = 3$. Then n is prime and n is odd, so we have $\mathrm{T} \Rightarrow \mathrm{T}$.
- $n = 4$. Then n is not prime and n is not odd, so we have $\mathrm{F} \Rightarrow \mathrm{F}$.
- $n = 9$. Then n is not prime, but n is odd. We have $\mathrm{F} \Rightarrow \mathrm{T}$.

None of these values of n would cause us to designate (1.1.1) as generating false statements. Therefore, we should assign the truth value "T" to each of these three cases.

However, let us take

- $n = 2$. Then n is prime, but n is not odd. We have $\mathrm{T} \Rightarrow \mathrm{F}$.

This is clearly a value of n for which (1.1.1) should be false. Hence, we we should assign the truth value "F" to the implication $\mathrm{T} \Rightarrow \mathrm{F}$.

Equivalence

1.1.10. Definition. Let A and B be two statements. Then we define the ***equivalence*** of A and B, written $A \Leftrightarrow B$, by the following truth table:

A	B	$A \Leftrightarrow B$
T	T	T
T	F	F
F	T	F
F	F	T

We read "$A \Leftrightarrow B$" as "A is equivalent to B" or "A if and only if B". Some textbooks abbreviate "if and only if" by "iff."

If A and B are both true or both false, then they are equivalent. Otherwise, they are not equivalent. In propositional logic, "equivalence" is the closest thing to the "equality" of arithmetic.

Equivalence

On the one hand, logical equivalence is strange; two statements A and B do not need to have anything to do with each other to be equivalent. For example, the statements "$2 > 0$" and "$100 = 99 + 1$" are both true, so they are equivalent.

On the other hand, we use equivalence to manipulate compound statements.

1.1.11. Definition. Two compound statements A and B are called ***logically equivalent*** if $A \Leftrightarrow B$ is a tautology. We then write $A \equiv B$.

1.1.12. Example. The two ***de Morgan rules*** are the tautologies

$$\neg(A \vee B) \Leftrightarrow (\neg A) \wedge (\neg B), \qquad \neg(A \wedge B) \Leftrightarrow (\neg A) \vee (\neg B).$$

In other words, they state that $\neg(A \vee B)$ is logically equivalent to $(\neg A) \wedge (\neg B)$ and $\neg(A \wedge B)$ is logically equivalent to $(\neg A) \vee (\neg B)$.

Contraposition

An important tautology is the ***contrapositive*** of the compound statement $A \Rightarrow B$.

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A).$$

For example, for any natural number n, the statement "$n > 0 \Rightarrow n^3 > 0$" is equivalent to "$n^3 \not> 0 \Rightarrow n \not> 0$." This principle is used in proofs by contradiction.

We prove the contrapositive using a truth table:

A	B	$\neg A$	$\neg B$	$\neg B \Rightarrow \neg A$	$A \Rightarrow B$	$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$
T	T	F	F	T	T	T
T	F	F	T	F	F	T
F	T	T	F	T	T	T
F	F	T	T	T	T	T

Rye Whiskey

The following song is an old Western-style song, called "Rye Whiskey" and performed by Tex Ritter in the 1930's and 1940's.

> *If the ocean was whiskey and I was a duck,*
> *I'd swim to the bottom and never come up.*
>
> *But the ocean ain't whiskey, and I ain't no duck,*
> *So I'll play jack-of-diamonds and trust to my luck.*
>
> *For it's whiskey, rye whiskey, rye whiskey I cry.*
> *If I don't get rye whiskey I surely will die.*

The lyrics make sense (at least as much as song lyrics generally do).

Rye Whiskey

We can use de Morgan's rules and the contrapositive to re-write the song lyrics as follows

> *If I never reach bottom or sometimes come up,*
> *Then the ocean's not whiskey, or I'm not a duck.*
>
> *But my luck can't be trusted, or the cards I'll not buck,*
> *So the ocean is whiskey or I am a duck.*
>
> *For it's whiskey, rye whiskey, rye whiskey I cry.*
> *If my death is uncertain, then I get whiskey (rye).*

These lyrics seem to be logically equivalent to the original song, but are just humorous nonsense. This again illustrates clearly why it is futile to apply mathematical logic to everyday language.

Some Logical Equivalencies

The following logical equivalencies can be established using truth tables or by using previously proven equivalencies. Here T is the compound statement that is always true, $T: A \vee (\neg A)$ and F is the compound statement that is always false, $F: A \wedge (\neg A)$

Equivalence	Name
$A \wedge T \equiv A$	Identity for $\wedge$
$A \vee F \equiv A$	Identity for $\vee$
$A \wedge F \equiv F$	Dominator for $\wedge$
$A \vee T \equiv T$	Dominator for $\vee$
$A \wedge A \equiv A$	Idempotency of $\wedge$
$A \vee A \equiv A$	Idempotency of $\vee$
$\neg(\neg A) \equiv A$	Double negation

Some Logical Equivalencies

Equivalence	Name
$A \wedge B \equiv B \wedge A$	Commutativity of $\wedge$
$A \vee B \equiv B \vee A$	Commutativity of $\vee$
$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$	Associativity of $\wedge$
$(A \vee B) \vee C \equiv A \vee (B \vee C)$	Associativity of $\vee$
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	Distributivity
$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$	Distributivity
$A \vee (A \wedge B) \equiv A$	Absorption
$A \wedge (A \vee B) \equiv A$	Absorption

These laws include all that are necessary for a ***boolean algebra generated by*** $\wedge$ ***and*** $\vee$ (identity element, commutativity, associativity, distributivity). Hence the name ***boolean logic*** for this calculus of logical statements.

Some Logical Equivalencies

We omitted de Morgan's laws from the previous table. We now list some equivalences involving conditional statements.

Equivalence
$A \Rightarrow B \equiv \neg A \vee B \equiv \neg B \Rightarrow \neg A$
$(A \Rightarrow B) \wedge (A \Rightarrow C) \equiv A \Rightarrow (B \wedge C)$ $(A \Rightarrow B) \vee (A \Rightarrow C) \equiv A \Rightarrow (B \vee C)$
$(A \Rightarrow C) \wedge (B \Rightarrow C) \equiv (A \vee B) \Rightarrow C$ $(A \Rightarrow C) \vee (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$
$(A \Leftrightarrow B) \equiv ((\neg A) \Leftrightarrow (\neg B))$ $(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$ $(A \Leftrightarrow B) \equiv (A \wedge B) \vee ((\neg A) \wedge (\neg B))$ $\neg(A \Leftrightarrow B) \equiv A \Leftrightarrow (\neg B)$

Logical Quantifiers

In the previous examples we have used predicates $A(x)$ with the words "for all x." This is an instance of a ***logical quantifier*** that indicates for which x a predicate $A(x)$ is to be evaluated to a statement.

In order to use quantifiers properly, we clearly need a universe of objects x which we can insert into $A(x)$ (a ***domain*** for $A(x)$). This leads us immediately to the definition of a ***set***. We will discuss set theory in detail later. For the moment it is sufficient for us to view a set as a "collection of objects" and assume that the following sets are known:

- the set of natural numbers $\mathbb{N}$ (which includes the number 0),
- the set of integers $\mathbb{Z}$,
- the set of real numbers $\mathbb{R}$,
- the empty set $\emptyset$ (also written $\varnothing$ or $\{\}$) that does not contain any objects.

If M is a set containing x, we write $x \in M$ and call x an ***element*** of M.

Logical Quantifiers

There are two types of quantifiers:

- the ***universal quantifier***, denoted by the symbol $\forall$, read as "for all" and
- the ***existential quantifier***, denoted by $\exists$, read as "there exists."

1.1.13. Definition. Let M be a set and $A(x)$ be a predicate. Then we define the quantifier $\forall$ by

$$\underset{x\in M}{\forall} A(x) \quad \Leftrightarrow \quad A(x) \text{ is true for all } x \in M$$

We define the quantifier $\exists$ by

$$\underset{x\in M}{\exists} A(x) \quad \Leftrightarrow \quad A(x) \text{ is true for at least one } x \in M$$

We may also write $\forall x \in M\colon A(x)$ instead of $\underset{x\in M}{\forall} A(x)$ and similarly for $\exists$.

Logical Quantifiers

We may also state the domain before making the statements, as in the following example.

1.1.14. Examples. Let x be a real number. Then

- $\forall x : x > 0 \Rightarrow x^3 > 0$ is a true statement;
- $\forall x : x > 0 \Leftrightarrow x^2 > 0$ is a false statement;
- $\exists x : x > 0 \Leftrightarrow x^2 > 0$ is a true statement.

Sometimes mathematicians put a quantifier at the end of a statement form; this is known as a ***hanging quantifier***. Such a hanging quantifier will be interpreted as being located just before the statement form:

$$\exists y : y + x^2 > 0 \qquad\qquad \forall x$$

is equivalent to $\exists y \forall x : y + x^2 > 0$.

Contraposition and Negation of Quantifiers

We do not actually need the quantifier $\exists$ since

$$\begin{aligned} \underset{x\in M}{\exists} A(x) \quad &\Leftrightarrow \quad A(x) \text{ is true for at least one } x \in M \\ &\Leftrightarrow \quad A(x) \text{ is not false for all } x \in M \\ &\Leftrightarrow \quad \neg \underset{x\in M}{\forall} (\neg A(x)) \end{aligned} \tag{1.1.2}$$

The equivalence (1.1.2) is called ***contraposition of quantifiers***. It implies that the negation of $\exists x \in M\colon A(x)$ is equivalent to $\forall x \in M\colon \neg A(x)$. For example,

$$\neg \big(\exists x \in \mathbb{R}\colon x^2 < 0\big) \qquad \Leftrightarrow \qquad \forall x \in \mathbb{R}\colon x^2 \not< 0.$$

Conversely,

$$\neg \big(\forall x \in M\colon A(x)\big) \qquad \Leftrightarrow \qquad \exists x \in M\colon \neg A(x).$$

Vacuous Truth

If the domain of the universal quantifier $\forall$ is the empty set $M = \varnothing$, then the statement $\forall x \in M\colon A(x)$ is defined to be true regardless of the predicate $A(x)$. It is then said that $A(x)$ is ***vacuously true***.

1.1.15. Example. Let M be the set of real numbers x such that $x = x + 1$. Then the statement

$$\forall_{x \in M} \; x > x$$

is true.

This convention reflects the philosophy that a universal statement is true unless there is a counterexample to prove it false. While this may seem a strange point of view, it proves useful in practice.

This is similar to saying that "All pink elephants can fly." is a true statement, because it is impossible to find a pink elephant that can't fly.

Nesting Quantifiers

We can also treat predicates with more than one variable as shown in the following example.

1.1.16. Examples. In the following examples, x, y are taken from the real numbers.

- $\forall x \forall y : x^2 + y^2 - 2xy \geq 0$ is equivalent to $\forall y \forall x : x^2 + y^2 - 2xy \geq 0$. Therefore, one often writes $\forall x, y : x^2 + y^2 - 2xy \geq 0$.
- $\exists x \exists y : x + y > 0$ is equivalent to $\exists y \exists x : x + y > 0$, often abbreviated to $\exists x, y : x + y > 0$.
- $\forall x \exists y : x + y > 0$ is a true statement.
- $\exists x \forall y : x + y > 0$ is a false statement.

As is clear from these examples, the order of the quantifiers is important if they are different.

Examples from Calculus

Let I be an interval in $\mathbb{R}$. Then a function $f : I \to \mathbb{R}$ is said to be ***continuous*** on I if and only if

$$\underset{\varepsilon>0}{\forall}\ \underset{x\in I}{\forall}\ \underset{\delta>0}{\exists}\ \underset{y\in I}{\forall}\quad |x-y|<\delta \Rightarrow |f(x)-f(y)|<\varepsilon.$$

The function f is ***uniformly continuous*** on I if and only if

$$\underset{\varepsilon>0}{\forall}\ \underset{\delta>0}{\exists}\ \underset{x\in I}{\forall}\ \underset{y\in I}{\forall}\quad |x-y|<\delta \Rightarrow |f(x)-f(y)|<\varepsilon.$$

It is easy to see that a function that is uniformly continuous on I must also be continuous on I.

If I is a closed interval, $I = [a, b]$, it can also be shown that a continuous function is also uniformly continuous. However, that requires techniques from calculus and is not obvious just by looking at the logical structure of the definitions.

Examples from Calculus

Negating complicated expressions can be done step-by-step. For example, the statement that f is not continuous on I is equivalent to

$$\neg\Big(\underset{\varepsilon>0}{\forall}\ \underset{x\in I}{\forall}\ \underset{\delta>0}{\exists}\ \underset{y\in I}{\forall}\quad |x-y|<\delta \Rightarrow |f(x)-f(y)|<\varepsilon\Big)$$

$$\Leftrightarrow \Big(\underset{\varepsilon>0}{\exists}\ \neg\ \underset{x\in I}{\forall}\ \underset{\delta>0}{\exists}\ \underset{y\in I}{\forall}\quad |x-y|<\delta \Rightarrow |f(x)-f(y)|<\varepsilon\Big)$$

$$\Leftrightarrow \Big(\underset{\varepsilon>0}{\exists}\ \underset{x\in I}{\exists}\ \neg\ \underset{\delta>0}{\exists}\ \underset{y\in I}{\forall}\quad |x-y|<\delta \Rightarrow |f(x)-f(y)|<\varepsilon\Big)$$

$$\Leftrightarrow \Big(\underset{\varepsilon>0}{\exists}\ \underset{x\in I}{\exists}\ \underset{\delta>0}{\forall}\ \neg\ \underset{y\in I}{\forall}\quad |x-y|<\delta \Rightarrow |f(x)-f(y)|<\varepsilon\Big)$$

$$\Leftrightarrow \Big(\underset{\varepsilon>0}{\exists}\ \underset{x\in I}{\exists}\ \underset{\delta>0}{\forall}\ \underset{y\in I}{\exists}\quad (|x-y|<\delta)\wedge\neg(|f(x)-f(y)|<\varepsilon)\Big)$$

$$\Leftrightarrow \Big(\underset{\varepsilon>0}{\exists}\ \underset{x\in I}{\exists}\ \underset{\delta>0}{\forall}\ \underset{y\in I}{\exists}\quad (|x-y|<\delta)\wedge(|f(x)-f(y)|\geq\varepsilon)\Big)$$

Examples from Calculus

1.1.17. Example. The ***Heaviside function*** $H\colon \mathbb{R} \to \mathbb{R}$,

$$H(x) := \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0, \end{cases}$$

is not continuous on $I = \mathbb{R}$. To see this, we need to show that there exists an $\varepsilon > 0$ (take $\varepsilon = 1/2$) and an $x \in \mathbb{R}$ (take $x = 0$) such that for any $\delta > 0$ there exists a $y \in \mathbb{R}$ such that

$$|x - y| = |y| < \delta \qquad \text{and} \qquad |H(x) - H(y)| = |1 - H(y)| \geq \varepsilon = \frac{1}{2}.$$

Given any $\delta > 0$ we can choose $y = -\delta/2$. Then $|y| = \delta/2 < \delta$ and $|1 - H(y)| = 1 > 1/2$. This proves that H is not continuous on $\mathbb{R}$.

Arguments in Mathematics

The previous example contains a ***mathematical argument*** to show that the Heaviside function is not continuous on its domain. The argument boils down to the following:

(i) We know that

$$A\colon \underset{\varepsilon>0}{\exists}\ \underset{x\in\mathbb{R}}{\exists}\ \underset{\delta>0}{\forall}\ \underset{y\in\mathbb{R}}{\exists}\quad (|x-y|<\delta)\wedge(|H(x)-H(y)|\geq\varepsilon)$$

implies

B: H is not continuous on its domain.

(ii) We show that A is true.

(iii) Therefore, B is true.

Logically, we can express this argument as

$$\big(A\wedge(A\Rightarrow B)\big)\Rightarrow B.$$

Arguments and Argument Forms

1.1.18. Definition.

(i) An ***argument*** is a finite sequence of statements. All statements except for the final statement are called ***premises*** while the final statement is called the ***conclusion***. We say that an argument is ***valid*** if the truth of all premises implies the truth of the conclusion.

(ii) An ***argument form*** is a finite sequence of predicates (statement forms). An argument form is ***valid*** if it yields a valid argument whenever statements are substituted for the predicates.

From the definition of an argument it is clear that an argument consisting of a sequence of premises $P_1, \ldots, P_n$ and a conclusion C is valid of and only if

$$(P_1 \wedge P_2 \wedge \cdots \wedge P_n) \Rightarrow C \tag{1.1.3}$$

is a tautology, i.e., a true statement for any values of the premises an the conclusion.

Arguments and Argument Forms

An argument is a finite list of premises $P_1, \ldots, P_n$ followed by a conclusion C. We usually write this list as

$$\begin{array}{ll} & P_1 \\ & P_2 \\ & \vdots \\ & P_n \\ \hline \therefore & C \end{array}$$

where the symbol $\therefore$ is pronounced "therefore". You may only use this symbol when constructing a logical argument in the notation above. Do not use it as a general-purpose abbreviation of "therefore".

Certain basic valid arguments in mathematics are given latin names and called ***rules of inference***.

Modus Ponendo Ponens

1.1.19. Example. The rule of inference

$$\begin{array}{rl} & A \\ & A \Rightarrow B \\ \hline \therefore & B \end{array}$$

is called ***modus ponendo ponens*** (latin for "mode that affirms by affriming"); it is often abbreviated simply "modus ponens". The associated tautology is

$$\big(A \wedge (A \Rightarrow B)\big) \Rightarrow B \tag{1.1.4}$$

We verify that (1.1.4) actually is a tautology using the truth table:

A	B	$A \Rightarrow B$	$A \wedge (A \Rightarrow B)$	$\big(A \wedge (A \Rightarrow B)\big) \Rightarrow B$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	T	T

Hypothetical Syllogisms

A ***syllogism*** is an argument that has exactly two premises. We first give three ***hypothetical syllogisms***, i.e., syllogisms involving the implication "$\Rightarrow$".

Rule of Inference	Name
$\begin{array}{l} A \\ A \Rightarrow B \\ \hline \therefore \quad B \end{array}$	Modus (Ponendo) Ponens ***Mode that affirms (by affirming)***
$\begin{array}{l} \neg B \\ A \Rightarrow B \\ \hline \therefore \quad \neg A \end{array}$	Modus (Tollendo) Tollens ***Mode that denies (by denying)***
$\begin{array}{l} A \Rightarrow B \\ B \Rightarrow C \\ \hline \therefore \quad A \Rightarrow C \end{array}$	Transitive Hypothetical Syllogism

Hypothetical Syllogisms

1.1.20. Examples.

(i) Modus ponendo ponens:

3 is both prime and greater than 2
If 3 is both prime and greater than 2, then 3 is odd

$\therefore$ 3 is odd.

(ii) Modus tollendo tollens:

4 is not odd
If 4 is both prime and greater than 2, then 4 is odd

$\therefore$ 4 is not both prime and greater than 2.

(iii) Transitive hypothetical syllogism:

If 5 is greater than 4, then 5 is greater than 3
If 5 is greater than 3, then 5 is greater than 2

$\therefore$ If 5 is greater than 4, then 5 is greater than 2.

Disjunctive and Conjunctive Syllogisms

There are two important syllogisms involving the disjunction "$\vee$" and the conjunction "$\wedge$":

Rule of Inference	Name
$A \vee B$, $\neg A$ / $\therefore B$	Modus Tollendo Ponens ***Mode that affirms by denying***
$\neg(A \wedge B)$, A / $\therefore \neg B$	Modus Ponendo Tollens ***Mode that denies by affirming***
$A \vee B$, $\neg A \vee C$ / $\therefore B \vee C$	Resolution

Disjunctive and Conjunctive Syllogisms

1.1.21. Examples.

(i) Modus tollendo ponens:

4 is odd or even
4 is not odd

$\therefore$ 4 is even.

(ii) Modus ponendo tollens:

4 is not both even and odd
4 is even

$\therefore$ 4 is not odd.

(iii) Resolution:

4 is even or 4 is greater than 2
4 is odd or 4 is prime

$\therefore$ 4 is greater than 2 or 4 is prime.

Some Simple Arguments

Finally, we give some seemingly obvious, but nevertheless useful, arguments:

Rule of Inference	Name
$\begin{array}{l} A \\ B \\ \hline \therefore\ A \wedge B \end{array}$	Conjunction
$\begin{array}{l} A \wedge B \\ \hline \therefore\ A \end{array}$	Simplification
$\begin{array}{l} A \\ \hline \therefore\ A \vee B \end{array}$	Addition

Examples for these are left to the reader!

Validity and Soundness

The previous rules of inference are all ***valid arguments***. In the examples we gave, the arguments always led to a correct conclusion. This was, however, only because all the premises were true statements. It is possible for a valid argument to lead to a wrong conclusion if one or more of its premises are false.

If, in addition to being valid, an argument has only true premises, we say that the argument is ***sound***. In that case, its conclusion is true.

1.1.22. Example. The following argument is valid (it is based on the rule of resolution), but not sound:

4 is even or 4 is prime
4 is odd or 4 is prime

$\therefore$ 4 is prime.

(The second premise is false, so the conclusion doesn't have to be true.)

Non Sequitur

The term ***non sequitur*** (latin for "it does not follow") is often used to describe logical fallacies, i.e., inferences that invalid because they are not based on tautologies. Some common fallacies are listed below:

Rule of Inference	Name
$\dfrac{B \quad A \Rightarrow B}{\therefore \quad A}$	Affirming the Consequent
$\dfrac{\neg A \quad A \Rightarrow B}{\therefore \quad \neg B}$	Denying the Antecedent
$\dfrac{A \vee B \quad A}{\therefore \quad \neg B}$	Affirming a Disjunct

Non Sequitur

1.1.23. Examples.

(i) Affirming the consequent:

$$\begin{array}{ll} & \text{If 9 is prime, then it is odd} \\ & \text{9 is odd} \\ \hline \therefore & \text{9 is prime.} \end{array}$$

(ii) Denying the antecedent

$$\begin{array}{ll} & \text{If 9 is prime, then it is odd} \\ & \text{9 is not prime} \\ \hline \therefore & \text{9 is not odd.} \end{array}$$

(iii) Affirming a disjunct:

$$\begin{array}{ll} & \text{2 is even or 2 is prime} \\ & \text{2 is even} \\ \hline \therefore & \text{2 is not prime.} \end{array}$$

Rules of Inference for Quantified Statements

Without proof or justification, we give the following rules of inference for quantified statements. They are often assumed as axioms in abstract logic systems.

Rule of Inference	Name
$\dfrac{\underset{x\in M}{\forall} P(x)}{\therefore \quad P(x_0) \text{ for any } x_0 \in M}$	Universal Instantiation
$\dfrac{P(x) \text{ for any arbitrarily chosen } x \in M}{\therefore \quad \underset{x\in M}{\forall} P(x)}$	Universal Generalization
$\dfrac{\underset{x\in M}{\exists} P(x)}{\therefore \quad P(x_0) \text{ for a certain (unknown) } x_0 \in M}$	Existential Instantiation
$\dfrac{P(x_0) \text{ for some (known) } x_0 \in M}{\therefore \quad \underset{x\in M}{\exists} P(x)}$	Existential Generalization

Constructing Arguments

Often, complex arguments can be broken down into syllogisms. As an example, we give a logical proof of the following theorem:

1.1.24. Theorem. Let $n \in \mathbb{N}$ be a natural number and suppose that n^2 is even. Then n is even.

Proof.
We use the following premises:

$$P_1: \quad \forall_{n \in \mathbb{N}} \neg(n \text{ even} \wedge n \text{ odd}),$$

$$P_2: n \text{ odd} \Rightarrow n^2 \text{ odd},$$

$$P_3: n^2 \text{ even} \wedge (n \text{ even} \vee n \text{ odd})$$

and we wish to arrive at the conclusion

$$C: n \text{ even}.$$

Constructing Arguments

Proof (continued).

Premise P_2 can be easily checked: if n is odd, there exists some k such that $n = 2k + 1$, so

$$n^2 = (2k+1)^2 = 2(2k^2 + 2k) + 1 = 2k' + 1$$

where $k' = 2k^2 + 2k$. Hence n^2 is also odd. We have

$$\frac{P_3 \colon n^2 \text{ even} \wedge (n \text{ even} \vee n \text{ odd})}{\therefore \quad P_4 \colon n^2 \text{ even}.}$$

by the Rule of Simplification. By Universal Instantiation, we obtain

$$\frac{P_1 \colon \underset{n \in \mathbb{N}}{\forall} \neg(n \text{ even} \wedge n \text{ odd})}{\therefore \quad P_5 \colon \neg(n^2 \text{ even} \wedge n^2 \text{ odd}).}$$

Constructing Arguments

Proof (continued).

Furthermore, by Modus Ponendo Tollens,

$$\begin{array}{ll} & P_4 : n^2 \text{ even} \\ & P_5 : \neg(n^2 \text{ even} \wedge n^2 \text{ odd}) \\ \hline \therefore & P_6 : \neg(n^2 \text{ odd}). \end{array}$$

Using Modus Tollendo Tollens,

$$\begin{array}{ll} & P_6 : \neg(n^2 \text{ odd}) \\ & P_2 : n \text{ odd} \Rightarrow n^2 \text{ odd} \\ \hline \therefore & P_7 : \neg(n \text{ odd}). \end{array}$$

Simplification yields

$$\begin{array}{ll} & P_3 : n^2 \text{ even} \wedge (n \text{ even} \vee n \text{ odd}) \\ \hline \therefore & P_8 : n \text{ even} \vee n \text{ odd}. \end{array}$$

Constructing Arguments

Proof (continued).
Finally, Modus Tollendo Ponens gives

$$\begin{array}{ll} & P_7 : \neg(n \text{ odd}) \\ & P_8 : n \text{ even} \vee n \text{ odd} \\ \hline \therefore \quad & C : n \text{ even}. \end{array}$$

This completes the proof. □

1.1.25. Remark. Of course, this proof could have been shortened and simplified if we had replaced "odd" with "not even" throughout, and we might have formulated premise P_3 slightly differently (as two separate premises) to avoid using the rule of simplification. However, our goal was to illustrate the usage of a wide variety of rules of inference and that writing down a logically valid proof is in most cases extremely tedious; in most mathematics, many of the mentioned rules of inference are used implicitly without being stated.

Naive Set Theory: Sets via Predicates

We want to be able to talk about "collections of objects"; however, we will be unable to strictly define what an "object" or a "collection" is (except that we also want any collection to qualify as an "object"). The problem with "naive" set theory is that any attempt to make a formal definition will lead to a contradiction - we will see an example of this later. However, for our practical purposes we can live with this, as we won't generally encounter these contradictions.

We indicate that an object (called an ***element***) x is part of a collection (called a ***set***) X by writing $x \in X$. We characterize the elements of a set X by some predicate P:

$$x \in X \qquad \Leftrightarrow \qquad P(x). \tag{1.2.1}$$

We write $X = \{x \colon P(x)\}$.

Notation for Sets

We define the empty set $\varnothing := \{x : x \neq x\}$. The empty set has no elements, because the predicate $x \neq x$ is never true.

We may also use the notation $X = \{x_1, x_2, \ldots, x_n\}$ to denote a set. In this case, X is understood to be the set

$$X = \{x : (x = x_1) \vee (x = x_2) \vee \cdots \vee (x = x_n)\}.$$

We will frequently use the convention

$$\{x \in A : P(x)\} = \{x : x \in A \wedge P(x)\}$$

1.2.1. Example. The set of even positive integers is

$$\{n \in \mathbb{N} : \exists_{k \in \mathbb{N}} \; n = 2k\}$$

Subsets and Equality of Sets

If every object $x \in X$ is also an element of a set Y, we say that X is a subset of Y, writing $X \subset Y$; in other words,

$$X \subset Y \qquad \Leftrightarrow \qquad \forall x \in X : x \in Y.$$

We say that $X = Y$ if and only if $X \subset Y$ and $Y \subset X$.

We say that X is a ***proper subset*** of Y if $X \subset Y$ but $X \neq Y$. In that case we write $X \subsetneq Y$.

Some authors write $\subseteq$ for $\subset$ and $\subset$ for $\subsetneq$. Pay attention to the convention used when referring to literature.

Examples of Sets and Subsets

1.2.2. Examples.

1. For any set X, $\varnothing \subset X$. Since $\varnothing$ does not contain any elements, the domain of the statement $\forall x \in X : x \in Y$ is empty. Therefore, it is vacuously true and hence $\varnothing \subset X$.
2. Consider the set $A = \{a, b, c\}$ where a, b, c are arbitrary objects, for example, numbers. The set

$$B = \{a, b, a, b, c, c\}$$

is equal to A, because it satisfies $A \subset B$ and $B \subset A$ as follows:

$$x \in A \quad \Leftrightarrow \quad (x = a) \vee (x = b) \vee (x = c) \quad \Leftrightarrow \quad x \in B.$$

Therefore, neither order nor repetition of the elements affects the contents of a set.

If $C = \{a, b\}$, then $C \subset A$ and in fact $C \subsetneq A$. Setting $D = \{b, c\}$ we have $D \subsetneq A$ but $C \not\subset D$ and $D \not\subset C$.

Power Set and Cardinality

If a set X has a finite number of elements, we define the ***cardinality*** of X to be this number, denoted by $\#X$, $|X|$ or $\operatorname{card} X$.

We define the ***power set***

$$\mathscr{P}(X) := \{A \colon A \subset X\}.$$

Here the elements of the set $\mathscr{P}(X)$ are themselves sets; $\mathscr{P}(X)$ is the "set of all subsets of X." Therefore, the statements

$$A \subset X \qquad \text{and} \qquad A \in \mathscr{P}(X)$$

are equivalent.

1.2.3. Example. The power set of $\{a, b, c\}$ is

$$\mathscr{P}(\{a, b, c\}) = \big\{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\big\}.$$

The cardinality of $\{a, b, c\}$ is 3, the cardinality of the power set is $|\mathscr{P}(\{a, b, c\})| = 8$.

Operations on Sets

If $A = \{x\colon P_1(x)\}$, $B = \{x\colon P_2(x)\}$ we define the ***union***, ***intersection*** and ***difference*** of A and B by

$$A \cup B := \{x\colon P_1(x) \vee P_2(x)\}, \qquad A \cap B := \{x\colon P_1(x) \wedge P_2(x)\},$$
$$A \setminus B := \{x\colon P_1(x) \wedge (\neg P_2(x))\}.$$

Let $A \subset M$. We then define the ***complement*** of A by

$$A^{\mathrm{c}} := M \setminus A.$$

If $A \cap B = \emptyset$, we say that the sets A and B are ***disjoint***.

Occasionally, the notation $A - B$ is used for $A \setminus B$ and A^{c} is sometimes denoted by $\overline{A}$.

1.2.4. Example. Let $A = \{a, b, c\}$ and $B = \{c, d\}$. Then

$$A \cup B = \{a, b, c, d\}, \qquad A \cap B = \{c\}, \qquad A \setminus B = \{a, b\}.$$

Operations on Sets

The laws for logical equivalencies immediately lead to several rules for set operations. For example, the distributive laws for $\wedge$ and $\vee$ imply

- $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Other such rules are, for example,

- $(A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$
- $(A \cap B) \setminus C = (A \setminus C) \cap (B \setminus C)$
- $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$
- $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$
- $A \setminus B = B^c \cap A$
- $(A \setminus B)^c = A^c \cup B$

Some of these will be proved in the recitation class and the exercises.

Operations on Sets

Occasionally we will need the following notation for the union and intersection of a finite number $n \in \mathbb{N}$ of sets:

$$\bigcup_{k=0}^{n} A_k := A_0 \cup A_1 \cup A_2 \cup \cdots \cup A_n,$$

$$\bigcap_{k=0}^{n} A_k := A_0 \cap A_1 \cap A_2 \cap \cdots \cap A_n.$$

This notation even extends to $n = \infty$, but needs to be properly defined:

$$x \in \bigcup_{k=0}^{\infty} A_k \quad :\Leftrightarrow \quad \exists_{k \in \mathbb{N}} \; x \in A_k,$$

$$x \in \bigcap_{k=0}^{\infty} A_k \quad :\Leftrightarrow \quad \forall_{k \in \mathbb{N}} \; x \in A_k.$$

Operations on Sets

In particular,

$$\bigcap_{k=0}^{\infty} A_k \subset \bigcup_{k=0}^{\infty} A_k.$$

1.2.5. Example. Let $A_k = \{0, 1, 2, \ldots, k\}$ for $k \in \mathbb{N}$. Then

$$\bigcup_{k=0}^{\infty} A_k = \mathbb{N}, \qquad \bigcap_{k=0}^{\infty} A_k = \{0\}.$$

To see the first statement, note that $\mathbb{N} \subset \bigcup_{k=0}^{\infty} A_k$ since $x \in \mathbb{N}$ implies $x \in A_x$ implies $x \in \bigcup_{k=0}^{\infty} A_k$. Furthermore, $\bigcup_{k=0}^{\infty} A_k \subset \mathbb{N}$ since $x \in \bigcup_{k=0}^{\infty} A_k$ implies $x \in A_k$ for some $k \in \mathbb{N}$ implies $x \in \mathbb{N}$.

For the second statement, note that $\bigcap_{k=0}^{\infty} A_k \subset \mathbb{N}$. Now $0 \in A_k$ for all $k \in \mathbb{N}$. Thus $\{0\} \subset \bigcap_{k=0}^{\infty} A_k$. On the other hand, for any $x \in \mathbb{N} \setminus \{0\}$ we have $x \notin A_{x-1}$ whence $x \notin \bigcap_{k=0}^{\infty} A_k$.

Ordered Pairs

A set does not contain any information about the order of its elements, e.g.,

$$\{a, b\} = \{b, a\}.$$

Thus, there is no such a thing as the "first element of a set". However, sometimes it is convenient or necessary to have such an ordering. This is achieved by defining an ***ordered pair***, denoted by

$$(a, b)$$

and having the property that

$$(a, b) = (c, d) \quad \Leftrightarrow \quad (a = c) \wedge (b = d). \tag{1.2.2}$$

We define

$$(a, b) := \{\{a\}, \{a, b\}\}.$$

It is not difficult to see that this definition guarantees that (1.2.2) holds.

Cartesian Product of Sets

If A, B are sets and $a \in A,\ b \in B$, then we denote the set of all ordered pairs by

$$A \times B := \{(a, b) : a \in A,\ \ b \in B\}.$$

$A \times B$ is called the ***cartesian product*** of A and B.

In this manner we can define an ***ordered triple*** (a, b, c) or, more generally, an ordered ***n-tuple*** $(a_1, \ldots, a_n)$ and the n-fold cartesian product $A_1 \times \cdots \times A_n$ of sets $A_k,\ k = 1, \ldots, n$.

If we take the cartesian product of a set with itself, we may abbreviate it using exponents, e.g.,

$$\mathbb{N}^2 := \mathbb{N} \times \mathbb{N}.$$

Problems in Naive Set Theory

If one simply views sets as arbitrary "collections" and allows a set to contain arbitrary objects, including other sets, then fundamental problems arise. We first illustrate this by an analogy:

Suppose a library contains not only books but also catalogs of books, i.e., books listing other books. For example, there might be a catalog listing all mathematics books in the library, a catalog listing all history books, etc. Suppose that there are so many catalogs, that you are asked to create catalogs of catalogs, i.e., catalogs listing other catalogs. In particular, you are asked to create the following:

(i) A catalog of all catalogs in the library. This catalog lists all catalogs contained in the library, so it must of course also list itself.

(ii) A catalog of all catalogs that list themselves. Does this catalog also list itself?

(iii) A catalog of all catalogs that do not list themselves. Does this catalog also list itself?

The Russel Antinomy

In the previous analogy, we can view "catalogs" as "sets" and being "listed in a catalog" as "being an element of a set". Then we have

(i) The set of all sets must have itself as an element.

(ii) The set of all sets that have themselves as elements may or may not contain itself. (This may be decidable by adding some rule to set theory.)

(iii) It is not decidable whether the "set of all sets that do not have themselves as elements" has itself as an element.

The Russel Antinomy

Formally, this paradox is known as the ***Russel antinomy:***

1.2.6. Russel Antinomy. The predicate $P(x)\colon x \notin x$ does not define a set $A = \{x\colon P(x)\}$.

Proof.
If $A = \{x\colon x \notin x\}$ were a set, then we should be able to decide for any set y whether $y \in A$ or $y \notin A$. We show that for $y = A$ this is not possible because either assumption leads to a contradiction:

(i) Assume $A \in A$. Then $P(A)$ by (1.2.1), i.e., $A \notin A$. ↯

(ii) Assume $A \notin A$. Then $\neg P(A)$ by (1.2.1), therefore $A \in A$. ↯

Since we cannot decide whether $A \in A$ or $A \notin A$, A can not be a set. □

The Russel Antinomy

There are several examples in classical literature and philosophy of the Russel antimony:

(i) Epimenides the Cretan says, "All Cretans are liars."

(ii) In a mountain village, there is a barber. Some villagers shave themselves (always) while the others never shave themselves. The barber shaves those and only those villagers that never shave themselves. Who shaves the barber?

Russel Antinomy

We will simply ignore the existence of such contradictions and build on naive set theory. There are further paradoxes (antinomies) in naive set theory, such as ***Cantor's paradox*** and the ***Burali-Forti paradox***. All of these are resolved if naive set theory is replaced by a ***modern axiomatic set theory*** such as ***Zermelo-Fraenkel set theory.***

Further Information:

- ***Set Theory***, Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/set-theory/
- P.R. Halmos, ***Naive Set Theory***,Aavilable here: http://link.springer.com/book/10.1007/978-1-4757-1645-0
- T. Jech, ***Set Theory: The Third Millennium Edition, Revised and Expanded***, Available here: http://link.springer.com/book/10.1007/3-540-44761-X

The Natural Numbers

The "counting numbers" $0, 1, 2, 3, \ldots$ are the basis of discrete mathematics. We refer to their totality as the set of ***natural numbers*** and denote it by $\mathbb{N}$. We have up to now used them to supply examples for our introduction to logic and to enumerate sets. It is time we briefly discuss how they can be formally defined.

We will represent the natural numbers as set of objects (denoted by $\mathbb{N}$) together with a relation called "succession": If n is a natural numbers, the "successor" of n, $\mathrm{succ}(n)$, is defined and also in $\mathbb{N}$. In elementary terms, 1 is the successor to 0, 2 is the successor to 1, etc.

There is no unique set of natural numbers in the sense that we can exhibit "the" set $\mathbb{N}$. Rather, any pair $(\mathbb{N}, \mathrm{succ})$, can qualify as a ***realization of the natural numbers*** if it satisfies certain axioms.

The Peano Axioms and the Induction Axiom

1.3.1. Definition. Let $\mathbb{N}$ be any set and suppose that the successor of any element of $\mathbb{N}$ has been defined. The ***Peano axioms*** are

1. $\mathbb{N}$ contains at least one object, called zero.
2. $\mathbb{N}$ is closed under the successor function, i.e., if n is in $\mathbb{N}$, the successor of n is in $\mathbb{N}$.
3. Zero is not the successor of a number.
4. Two numbers of which the successors are equal are themselves equal.
5. Induction axiom. If a set $S \subset \mathbb{N}$ contains zero and also the successor of every number in S, then $S = \mathbb{N}$.

Any set with a successor relation satisfying these axioms is called a realization of the natural numbers.

The von Neumann Realization

The Hungarian mathematician John von Neumann gave a very elegant realization of the natural numbers that is based on set theory:

Assume the empty set $\emptyset$ exists. Our idea is to define

$$
\begin{aligned}
0 &:= \emptyset, \\
1 &:= \{0\} = \{\emptyset\}, \\
2 &:= \{0, 1\} = \{\emptyset, \{\emptyset\}\}, \\
3 &:= \{0, 1, 2\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}
\end{aligned}
$$

and so on. It is clear that we want 1 to be the successor of 0, 2 the successor of 1 and so on.

The set of natural numbers $\mathbb{N}$ is then defined as consisting of the element 0 as well as the successor of every element already known to be in $\mathbb{N}$.

The von Neumann Realization

Several questions arise:

- Is it possible to define a set $\mathbb{N}$ in this ***recursive*** way?
- Does the von Neumann realization actually satisfy the Peano axioms?

The answer to both questions is affirmative:

- The possibility of defining $\mathbb{N}$ in this way is taken as an ***axiom*** of set theory.
- The proof that Peano axiom (iv) is satisfied is much harder than it looks - try it!

For a discussion of both questions (and more) we refer to P.R. Halmos, ***Naive Set Theory*** (see Slide 75). The relevant excerpt has been uploaded to SAKAI.

Addition

Once the set of natural numbers is established (e.g., through the Peano axioms), it is possible to define the operation of addition on $\mathbb{N}$,

$$+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \qquad (a, b) \mapsto a + b$$

We restrict ourselves to listing the properties that this operation then has.

For any two natural numbers $a, b \in \mathbb{N}$ we can define the natural number $c = a + b \in \mathbb{N}$ called the ***sum*** of a and b. This addition has the following properties ($a, b, c \in \mathbb{N}$):

(i) $a + (b + c) = (a + b) + c$ (***Associativity***)

(ii) $a + 0 = 0 + a = a$ (***Existence of a neutral element***)

(iii) $a + b = b + a$ (***Commutativity***)

Multiplication

Similarly, we can define ***multiplication***, where $a \cdot b \in \mathbb{N}$ is called the ***product*** of a and b. We have the following properties ($a, b, c \in \mathbb{N}$):

(i) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (***Associativity***)

(ii) $a \cdot 1 = 1 \cdot a = a$ (***Existence of a neutral element***)

(iii) $a \cdot b = b \cdot a$ (***Commutativity***)

We also have a property that essentially states that addition and multiplication are ***consistent***,

$$a \cdot (b + c) = a \cdot b + a \cdot c. \qquad (\textbf{\textit{Distributivity}})$$

Note that we are not able to define subtraction or division for all natural numbers.

Notation for Addition and Multiplication

For any numbers $a_1, a_2, \ldots, a_n$ we define the notation

$$a_1 + a_2 + \cdots + a_n =: \sum_{j=1}^{n} a_j =: \sum_{1 \le j \le n} a_j$$

and

$$a_1 \cdot a_2 \cdots a_n =: \prod_{j=1}^{n} a_j =: \prod_{1 \le j \le n} a_j.$$

For $a, b \in \mathbb{N}$ define the statement

$$a \mid b \qquad \Leftrightarrow \qquad \exists c \in \mathbb{N} \colon c \cdot a = b,$$

read as "a divides b." If $a \mid b$, then a is called a ***divisor*** of b.

Mathematical Induction

Typically one wants to show that some statement frame $A(n)$ is true for all $n \in \mathbb{N}$ with $n \geq n_0$ for some $n_0 \in \mathbb{N}$. Mathematical induction works by establishing two statements:

(I) $A(n_0)$ is true.

(II) $A(n+1)$ is true whenever $A(n)$ is true for $n \geq n_0$, i.e.,

$$\forall_{\substack{n \in \mathbb{N} \\ n \geq n_0}} \big(A(n) \Rightarrow A(n+1)\big)$$

Note that (II) does not make a statement on the situation when $A(n)$ is false; it is permitted for $A(n+1)$ to be true even if $A(n)$ is false.

The principle of mathematical induction now claims that $A(n)$ is true for all $n \geq n_0$ if (I) and (II) are true. This follows from the fifth Peano axiom (the induction axiom).

Introductory Example

1.3.2. Example. Consider the statement

$$\sum_{k=1}^{n}(2k-1) = n^2 \qquad \text{for all } n \in \mathbb{N} \setminus \{0\}.$$

This is a typical example, in that $A(n)\colon \sum_{k=1}^{n}(2k-1) = n^2$ is a predicate which is to be shown to hold for all natural numbers $n > 0$.

We first establish that $A(1)$ is true:

$$\sum_{k=1}^{1}(2k-1) = 2\cdot 1 - 1 = 1 \qquad \text{and} \qquad 1^2 = 1,$$

so $A(1)\colon 1 = 1$ is true.

Introductory Example

We next show that $A(n) \Rightarrow A(n+1)$ for all $n \in \mathbb{N} \setminus \{0\}$. This means we show that $\sum_{k=1}^{n+1}(2k-1) = (n+1)^2$ if $\sum_{k=1}^{n}(2k-1) = n^2$. Let n now be any n for which $A(n)$ is true. We then write

$$\sum_{k=1}^{n+1}(2k-1) = \sum_{k=1}^{n}(2k-1) + 2(n+1) - 1$$

If $A(n)$ is true for this specific n, we can replace the sum on the right by n^2, yielding

$$\sum_{k=1}^{n+1}(2k-1) = n^2 + 2n + 1 = (n+1)^2$$

But this is just the statement $A(n+1)$. Therefore, if $A(n)$ is true, then $A(n+1)$ will also be true. We have shown that $A(n) \Rightarrow A(n+1)$.

Foundations of Induction

Essentially, mathematical induction claims that for all $n_0 \in \mathbb{N}$,

$$\Big(A(n_0) \wedge \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} (A(n) \Rightarrow A(n+1))\Big) \Rightarrow \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} A(n). \tag{1.3.1}$$

To simplify the discussion, let us consider first the case $n_0 = 0$. Why is the conclusion (1.3.1) justified? Recall the 5th Peano axiom for the natural numbers:

> ***Induction axiom:*** *If a set $S \subset \mathbb{N}$ contains zero and also the successor of every number in S, then $S = \mathbb{N}$.*

Let us take $S = \{n \in \mathbb{N} : A(n)\}$. Then we first show that $A(0)$ is true, so $0 \in S$. Next we show that $A(n) \Rightarrow A(n+1)$ for all $n \in \mathbb{N}$. This means that if $n \in S$, then $n+1$ (the successor of n) is also in S. By the induction axiom, this means that $S = \mathbb{N}$. Thus, by showing the hypothesis in (1.3.1) we arrive at the conclusion, $A(n)$ is true for all $n \in \mathbb{N}$.

Foundations of Induction

To adapt the previous argument to the case where $n_0 > 0$, we can just take

$$S = \{n \in \mathbb{N} : n < n_0 \vee ((n \geq n_0) \wedge A(n))\}.$$

The details are left to you!

We observe that the 5th Peano axiom is basically tailor-made to ensure the validity of induction. We could paraphrase the axiom as "induction works!" In fact, there are alternative choices for the 5th axiom, such as the so-called

> ***Well-Ordering Principle:*** *Every non-empty set $S \subset \mathbb{N}$ has a least element.*

This axiom assumes that we know what "least", or, more precisely, "less than" means for the natural numbers. While we will discuss ordering relations in detail later, for now we make a provisional definition as follows.

Foundations of Induction

Let S_m be the set containing m as well as the successor of any element of S_m. Then

$$m < n \quad :\Leftrightarrow \quad (n \in S_m) \wedge (n \neq m).$$

Of course, if we use the von Neumann construction of the natural numbers, we simply have $m \leq n \Leftrightarrow m \subset n$.

We now take a "least element of a set M" to be an element $m_0 \in M$ such that $M \subset S_{m_0}$, or in other words, $m_0 \leq m$ for all $m \in M$. We then have the Well-Ordering Principle can then be written as

$$\forall_{M \subset \mathbb{N}} \; \exists_{m \in M} \; M \subset S_m.$$

(The Well-Ordering Principle is sometimes also called the Least Element Principle.)

Foundations of Induction

1.3.3. Theorem. Assume that a system of numbers satisfies the first four Peano axioms and the Well-Ordering Principle. Then the Induction Axiom holds.

Proof.
Let $S \subset \mathbb{N}$ satisfy the property that $0 \in S$ and that if $n \in S$, then $\text{succ}(n) \in S$. We need to show that $S = \mathbb{N}$. Suppose that there exists an $n_0 \in \mathbb{N}$ such that $n_0 \notin S$. Then the set $M = \{n \in \mathbb{N} : n \notin S\}$ is non-empty. By the well-ordering principle, M must have a least element m_0. Since $0 \in S$, $m_0 \neq 0$. Since $m_0 > 0$ and $m_0 \in \mathbb{N}$, there exists a number $m_0 - 1$ preceding m_0. This number is not in M, because m_0 is the least element of M. Therefore, $m_0 - 1 \in S$. However, by the definition of S, the successor of $m_0 - 1$ must also be in S. Since the successor of every number is unique, $m_0 \in S$ and hence $m_0 \notin M$. We arrive at a contradiction. □

Foundations of Induction

In fact, the Induction Axiom and the Well-Ordering Principle are equivalent: you will prove in the assignments that the Induction Axiom also implies the Well-Ordering Principle. This means that if we take one of the two as an axiom, the other becomes a theorem that can be proven. It is a common situation in mathematics that we have several equivalent choices for a system of axioms.

Further Examples of Induction

Mathematical induction can be used to prove any sort of statement on the natural numbers in a variety of contexts. Some examples follow:

1.3.4. Examples.

1. $\forall_{n\in\mathbb{N}} (1+\frac{1}{2})^n \geq 1 + n/2.$
2. $\forall_{n\in\mathbb{N}} \forall_{a,b\in\mathbb{Q}} (a+b)^n = \sum_{k=0}^{n} \frac{n!}{(n-k)!k!} a^n b^{n-k}.$
3. $\forall_{n\in\mathbb{N}} \forall_{\substack{r\in\mathbb{Q} \\ r=p/q \\ q^2>p^2}} \sum_{k=0}^{n} r^k = \frac{r^{k+1}-1}{r-1}.$
4. Let $H_n = \sum_{k=1}^{n} \frac{1}{k}$ for $n \in \mathbb{N}\setminus\{0\}$. Then $\forall_{n\in\mathbb{N}} H_{2^n} \geq 1 + n/2.$
5. Let M be a set and $A_1, \ldots, A_n \subset M$. Then

$$\left(\bigcap_{i=1}^{n} A_i\right)^{\mathsf{c}} = \bigcup_{i=1}^{n} A_i^{\mathsf{c}}.$$

Further Examples of Induction

6. Let M be a set with cardinality $\operatorname{card} M = n$, $n \in \mathbb{N}$. Then $\operatorname{card} \mathscr{P}(M) = 2^n$.
7. Shootout at the O.K. Corral: an odd number of lawless individuals, standing at mutually distinct distances to each other, fire pistols at each other in exactly the same instant. Every person fires at their nearest neighbor, hitting and killing this person. Then there is at least one survivor.

The Shootout

Let us discuss the last example in more detail: Let $P(n)$ be the statement that there is at least one survivor whenever $2n+1$ people, fire pistols at each other in the same instant. Each person fires at his nearest neighbor, and all people stand at mutually distinct distances to each other.

For $n = 1$, there are three people, A, B and C. Suppose that the distance between A and B is the smallest distance of any two of them. Then A fires at B and vice-versa. C fires at either A or B and will not be fired at, so C survives.

Suppose $P(n)$. Let $2n+3$ people participate in the shootout. Suppose that A and B are the closest pair of people. The A and B fire at each other.

- If at least one other person fires at A or B, then there remain at most $2n$ shots fired among the remaining $2n+1$ people, so there is at least one survivor.
- If no-one else fires at A or B, then there are $2n+1$ shots fired among the $2n+1$ people and by $P(n)$, there is at least one survivor.

Pitfalls in Induction

While mathematical induction is an extremely powerful technique, it must be executed most carefully. In proceeding through an induction proof it can happen quite easily that implicit assumptions are made that are not justified, thereby invalidating the result.

1.3.5. Example. Let us use mathematical induction to argue that every set of $n \geq 2$ lines in the plane, no two of which are parallel, meet in a common point.

The statement is true for $n = 2$, since two lines are not parallel if and only if they meet at some point. Since these are the only lines under considerations, this is the common meeting point of the lines.

We next assume that the statement is true for n lines, i.e., any n non-parallel lines meet in a common point. Let us now consider $n + 1$ lines, which we number 1 through $n + 1$. Take the set of lines 1 through n; by the induction hypothesis, they meet in a common point. The same is true of the lines $2, \ldots, n + 1$. We will now show that these points must be identical.

Pitfalls in Induction

Assume that the points are distinct. Then all lines $2, \ldots, n$ must be the same line, because any two points determine a line completely. Since we can choose our original lines in such a way that we consider distinct lines, we arrive at a contradiction. Therefore, the points must be identical, so all $n + 1$ lines meet in a common point. This completes the induction proof.

Where is the mistake in the above "proof" of our (obviously false) supposition?

Strong (Complete) Induction

The method of induction can be strengthened. We can replace

(I) $A(n_0)$ is true.

(II) $A(n+1)$ is true whenever $A(n)$ is true for $n \geq n_0$.

with

(I) $A(n_0)$ is true.

(II') $A(n+1)$ is true whenever all the statements $A(n_0), A(n_0+1), \ldots, A(n)$ are true.

1.3.6. Example. We will show the following statement: ***Every natural number $n \geq 2$ is a prime number or the product of primes.***

Clearly the statement is true for $n = 2$, which is prime. Next assume that $2, 3, \ldots, n$ are all prime or the product of prime numbers. Then $n+1$ is either prime or not prime. If it is prime, we are finished. If it is not prime, it is the product of two numbers $a, b < n+1$. However, a and b are themselves products of prime numbers by our assumption, and hence so is $n+1 = a \cdot b$. Therefore, by the strong induction principle the initial statement is true.

Induction vs. Strong Induction

While induction is the principle that

$$\Big(A(n_0) \wedge \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} \big(A(n) \Rightarrow A(n+1)\big)\Big) \Rightarrow \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} A(n) \tag{1.3.2}$$

strong induction states

$$\Big(A(n_0) \wedge \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} \big((A(n_0) \wedge \cdots \wedge A(n)) \Rightarrow A(n+1)\big)\Big) \Rightarrow \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} A(n). \tag{1.3.3}$$

It is clear that (1.3.3) implies (1.3.2), since of course

$$\big((A(n_0) \wedge \cdots \wedge A(n)) \Rightarrow A(n+1)\big) \Rightarrow \big(A(n) \Rightarrow A(n+1)\big)$$

Thus the "usual" induction is a special case of strong induction. However, the converse is also true, as we shall see.

Induction vs. Strong Induction

We now show that (1.3.2) implies (1.3.3), i.e., strong induction follows from induction.

We fix $n_0 \in \mathbb{N}$ and define $B(n) : A(n_0) \wedge \cdots \wedge A(n)$ for $n \geq n_0$. Then $A(n_0) = B(n_0)$ and we can write strong induction as

$$\Big(B(n_0) \wedge \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} \big(B(n) \Rightarrow A(n+1)\big)\Big) \Rightarrow \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} A(n).$$

We can write

$$\big(B(n) \Rightarrow A(n+1)\big) \equiv \big(B(n) \Rightarrow (A(n+1) \wedge B(n))\big) \equiv \big(B(n) \Rightarrow B(n+1)\big)$$

so strong induction becomes

$$\Big(B(n_0) \wedge \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} \big(B(n) \Rightarrow B(n+1)\big)\Big) \Rightarrow \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} A(n). \qquad (1.3.4)$$

Since $\underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} A(n) \equiv \underset{\substack{n\in\mathbb{N}\\ n\geq n_0}}{\forall} B(n)$ we see that (1.3.4) is just induction in B.

Recursive Definitions and the Factorial

The induction axiom also allows us to make ***recursive definitions***. For example, we wish to define a function (to be called the ***factorial***)

$$(\,\cdot\,)!\colon \mathbb{N} \to \mathbb{N}$$

having the properties that

$$0! := 1, \qquad n! := n \cdot (n-1)!, \qquad n \in \mathbb{N} \setminus \{0\}. \tag{1.3.5}$$

This is an example of a ***recursive definition*** and we may ask whether such a definition "makes sense", i.e., whether such a function exists and is unique.

That such recursive definitions define unique functions on $\mathbb{N}$ can be shown using the induction axiom. We refer again to Halmos's book for the details. In the present case, the function is simply

$$n! := \prod_{k=1}^{n} k, \qquad n \in \mathbb{N} \setminus \{0\}, \tag{1.3.6}$$

Recursive Definitions

The definition (1.3.5) is called an ***inductive*** or ***recursive formula*** for $n!$, while (1.3.6) is called a ***closed formula*** for $n!$. Recursive definitions often occur naturally in the formulation of a problem, and finding a closed formula can be extremely difficult. In some situations, a closed formula is highly desirable, while at other times, important properties are best expressed through recursive expressions.

For example, there exists a continuous extension of the factorial, given by the ***Euler gamma function***, defined for $t > 0$,

$$\Gamma(t) := \int_0^\infty z^{t-1} e^{-z}\, dz, \qquad t > 0.$$

It is possible to show that $\Gamma(1) = 1$ and that

$$\Gamma(t+1) = t\Gamma(t) = t\Gamma\big((t-1)+1\big) \qquad \text{for } t > 0.$$

Justification of Recursive Definitions

Comparing with (1.3.5), we see that

$$\Gamma(n+1) = n! \qquad \text{for } n \in \mathbb{N}.$$

Since the gamma function is defined for all strictly positive real numbers, we have a "continuous extension" of the factorial.

A slight modification allows for recursive definitions "starting" at $n = n_0$ instead of $n = 0$. Furthermore, we can define functions not just based on their preceding value, but on several such values.

1.3.7. Example. The ***Fibonacci sequence*** is defined through

$$f_0 := 0, \qquad f_1 := 1, \qquad f_n := f_{n-1} + f_{n-2}, \qquad n \in \mathbb{N} \setminus \{0, 1\}.$$

This type of recursive definition also follows from the induction axiom, much as strong induction does.

Recursive Definitions of Sets

In the same manner, we can define subsets of $\mathbb{N}$ recursively. For example, consider the set $S \subset \mathbb{N}$ such that

$$3 \in S \qquad \text{and} \qquad x, y \in S \Rightarrow x + y \in S. \tag{1.3.7}$$

(The validity of such a recursive definition (that a set S with the properties (1.3.7) exists and is unique) is based on the induction axiom.) We know that $3 \in S$, so $3 + 3 = 6 \in S$, $3 + 6 = 9 \in S$, $6 + 6 = 12 \in S$ and so on. Our goal is to prove that

$$S = \{n \in \mathbb{N} : \exists k \in \mathbb{N} \setminus \{0\} : n = 3k\}.$$

However, this requires a little preparation.

Alphabets and Strings

We introduce an example from the theory of formal languages:

1.3.8. Definition. An ***alphabet*** Σ is a finite, non-empty set of elements called ***symbols***. We define the set Σ^* of ***strings*** (or ***words***) over Σ as follows:

(i) $\lambda \in \Sigma^*$, where λ is the ***empty string*** (***null string***) containing no symbols.

(ii) If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

1.3.9. Example. Let $\Sigma = \{0, 1\}$. The elements of Σ are called ***bits*** and the words over Σ are called ***bit strings***. λ is a word, so also $\lambda 0 = 0$ is a word, as is $\lambda 1 = 1$. Since $\{0, 1\} \subset \Sigma^*$, the two-symbol words $01, 10, 11, 00 \in \Sigma^*$ and, continuing, $000, 001, 010, 011, 100, 101, 110, 111 \in \Sigma^*$. In this way, we inductively find all words over Σ.

We may interpret strings as tuples or finite sequences, but that is not necessary.

Bit Strings in Logic

Bit strings can be used in logic programming by replacing the values T and F by 1 and 0, respectively. Logical operations then become analogously defined ***bit operations***. In particular, $\neg$ becomes NOT, $\wedge$ becomes AND, $\vee$ becomes OR, where (for example)

b	NOT b
1	0
0	1

a	b	a AND b
1	1	1
1	0	0
0	1	0
0	0	0

a	b	a OR b
1	1	1
1	0	1
0	1	1
0	0	0

Given a bit string we then define corresponding ***bitwise operations*** recursively. For example,

$$\mathrm{NOT}(\lambda) := \lambda, \quad \mathrm{NOT}(wx) := \mathrm{NOT}(w)\,\mathrm{NOT}(x), \quad w \in \Sigma^*,\ x \in \Sigma.$$

Thus

$$\mathrm{NOT}(011) = \mathrm{NOT}(01)\,\mathrm{NOT}(1) = \mathrm{NOT}(0)\,\mathrm{NOT}(1)\,\mathrm{NOT}(1) = 100.$$

Concatenation of Strings

To denote a non-empty word $w \neq \lambda$, we often write wx, where $w \in \Sigma^*$ and $x \in \Sigma$.

1.3.10. Definition. We define ***concatenation of strings*** as follows:

(i) If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where λ is the empty string.

(ii) If $w_1, w_2 \in \Sigma^*$, $x \in \Sigma$, then

$$w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x.$$

This recursive definition is a bit difficult to read at first, so we give an example.

1.3.11. Example. Let $\Sigma = \{0, 1\}$. The concatenation of $110, 101 \in \Sigma^*$ according to Definition 1.3.10 is computed as follows:

$$\underset{w_1}{110} \cdot (\underset{w_2}{10}\,\underset{x}{1}) = (110 \cdot 10)1 = \big((110 \cdot 1)0\big)1$$

$$= \big(((110)1)0\big)1 = 110101.$$

Length of Strings

Definition 1.3.10 thus reduces the concatenation of strings to the concatenation of a string with a symbol, which we know how to do from Definition 1.3.8. In other words, we concatenate words by appending one symbol at a time. We make one more definition:

1.3.12. Definition. We define the ***length*** $l(w)$ ***of a string*** $w \in \Sigma^*$ as follows:

1. $l(\lambda) = 0$,
2. $l(wx) = l(w) + 1$, where $x \in \Sigma$.

1.3.13. Remark. Since the set Σ^* is not a subset of the natural numbers, it may be asked whether a set with the properties of Definition 1.3.8 exists uniquely. That this is the case is an axiom of set theory, i.e., it is assumed that such recursive definitions "work".

Structural Induction

Structural induction is a useful variant of induction that allows us to prove properties for recursively defined objects, such as the strings we have just introduced.

Structural induction establishes a statement on a recursively defined set in two steps. We call those elements specifically included in the set (e.g., the empty string in Σ^*) the basis elements of the set.

1. Establish the statement for the basis elements.
2. Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the statement holds for these new elements.

Structural Induction

As a first example, consider a proposition $P(w)$, where $w \in \Sigma^*$ is a string. In order to prove that $P(w)$ is true for all $w \in \Sigma^*$, we need to show

1. $P(\lambda)$, where λ is the empty string.
2. $\underset{w\in\Sigma^*}{\forall} \underset{x\in\Sigma}{\forall} P(w) \Rightarrow P(wx)$.

1.3.14. Example. We prove that $l(xy) = l(x) + l(y)$ for $x, y \in \Sigma^*$, where $l(w)$ is the length of a string $w \in \Sigma^*$, cf. Definition 1.3.12.

We need to formulate the statement in such a way that we can employ induction. Let us write it as

$$P(y)\colon \underset{x\in\Sigma^*}{\forall} l(xy) = l(x) + l(y).$$

We first establish $P(\lambda)$. This is the statement

$$P(\lambda)\colon \underset{x\in\Sigma^*}{\forall} l(x\lambda) = l(x) + l(\lambda).$$

Since $x\lambda = x$ and $l(\lambda) = 0$, $P(\lambda)$ is true.

Structural Induction

Next, assume that $P(y)$ is true. We must now show that $P(ya)$ is true for all $a \in \Sigma$, i.e.,

$$P(y) \Rightarrow \underset{a \in \Sigma}{\forall} P(ya)$$

or

$$\left(\underset{x \in \Sigma^*}{\forall} l(xy) = l(x) + l(y) \right) \Rightarrow \left(\underset{a \in \Sigma}{\forall} \underset{x \in \Sigma^*}{\forall} l(xya) = l(x) + l(ya) \right)$$

Since $l(xya) = l(xy) + 1$ and $l(ya) = l(y) + 1$ by Definition 1.3.12, the implication follows and the proof is complete.

The justification for structural induction lies in ordinary induction, applied to the statement

> $P(n)$: The claim is true for all elements of the set generated with n or fewer applications of the recursive rules for the set.

Structural induction first establishes $P(0)$ and then $P(n) \Rightarrow P(n+1)$.

Explicit Representation of Recursively Defined Sets

Let us return to our original example: Define $S \subset \mathbb{N}$ to be the set such that

(i) $3 \in S$,

(ii) $x, y \in S \Rightarrow x + y \in S$.

Then set

$$T = \left\{ n \in \mathbb{N} : \exists_{k \in \mathbb{N} \setminus \{0\}} \; n = 3k \right\}$$

We want to show that $S = T$.

First, we show $S \subset T$ by structural induction: $3 = 3 \cdot 1 \in T$, so the base case is established. Now for $x, y \in S$ suppose that $x, y \in T$ so that $x = 3k$ and $y = 3k'$ for $k, k' \in \mathbb{N} \setminus \{0\}$. Then

$$x + y = 3k + 3k' = 3(k + k')$$

so $x + y \in T$. This shows that $S \subset T$.

Explicit Representation of Recursively Defined Sets

Next, we show $T \subset S$ by (ordinary) induction. We claim that

$$\forall_{k \in \mathbb{N} \setminus \{0\}} \quad 3k \in S.$$

For $k = 1$, $3k = 3 \cdot 1 = 3 \in S$, so the base case is established. Now suppose that $3k \in S$. Since $3 \in S$ by definition, we can apply the recursive rule for S to deduce that

$$3(k+1) = 3k + 3 \in S.$$

This shows that $3(k+1) \in S$ if $3k \in S$. By the structural induction principle, $3k \in S$ for all $k \in \mathbb{N} \setminus \{0\}$. This established $T \subset S$.

We finally conclude that $S = T$.

Structure for Numbers

In mathematics, it is frequently useful to classify diverse objects into abstract frameworks. Understanding the abstract framework then gives an understanding of these various concrete objects, which superficially may appear quite unrelated.

An example of this from linear algebra is the concept of a vector space: By understanding concepts in general vector space (such as linear independence, bases, norms, inner products), the corresponding results can be applied to various objects, such as

- Euclidean space $\mathbb{R}^n$
- The set of continuous functions on an interval
- The set of solutions to a system of linear algebraic equations
- The set of solutions to a single homogeneous, linear differential equation

Groups

We aim to find an analogous structure for ***numbers***, in the hopes that later other objects can be subsumed in this structure. In particular, we would like the set of numbers, together with the operation of addition, to form a group.

1.4.1. Definition. A ***group*** is a pair $(G, \circ)$ consisting of a set G and a ***group operation*** $\circ\colon G \times G \to G$ such that

(i) $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a, b, c \in G$ (associativity),

(ii) there exists an element $e \in G$ such that $a \circ e = e \circ a = a$ for all $a \in G$ (existence of a unit element),

(iii) for every $a \in G$ there exists an element $a^{-1} \in G$ such that $a \circ a^{-1} = a^{-1} \circ a = e$ (existence of an inverse).

A group is called ***abelian*** if in addition to the above properties

(iv) $a \circ b = b \circ a$ for all $a, b \in G$ (commutativity).

The Natural Numbers are not a Group

It is easy to check that $(\mathbb{N}, +)$ satisfies the properties

(i) $a + (b + c) = (a + b) + c$ for all $a, b, c \in \mathbb{N}$,

(ii) $0 \in \mathbb{N}$ satisfies $a + 0 = 0 + a = a$ for all $a \in \mathbb{N}$,

(iv) $a + b = b + a$ for all $a, b \in \mathbb{N}$.

However, it is not true that

(iii) for every $a \in \mathbb{N}$ there exists an element $a^{-1} \in \mathbb{N}$ such that $a + a^{-1} = a^{-1} + a = 0$.

Therefore, we need to expand the set of natural numbers to a larger set (which we will call the ***integers***). In order to construct this set, we need to first digress to some material from Chapter 8 of Rosen's textbook.

Relations

Given two sets, the elements of these sets may be paired together in a certain way. A more precise formulation is given by the concept of a ***relation***, which we now introduce.

1.4.2. Examples.

- For $n \in \mathbb{N}$, $n \mapsto n^2$ associates to every number n its square n^2.
- $n^3 \mapsto n$ associates to some numbers $n^3 = p \in \mathbb{N}$ the number n.

We implement this by defining a relation R to be a set of ordered pairs. In the above examples, we might define

$$R = \{(a, b) \in \mathbb{N}^2 : b = a^2\},$$
$$R = \{(a, b) \in \mathbb{N}^2 : a = b^3\},$$

Relations

We define a ***relation*** R ***between two sets*** M ***and*** N to be the set

$$R = \{(m, n) \in M \times N \colon P(m, n)\} \tag{1.4.1}$$

where P is a statement frame (predicate). If $M = N$, we say that R is a ***relation on*** M.

Thereby the ***active*** concept of a relation (we associate one number to another; we ***do*** something) is defined by the ***static*** concept of a set (no action takes place; R is just an object). This is a fairly modern idea in mathematics.

Instead of writing $(a, b) \in R$, we often write $a \sim_R b$ (or $a \sim b$ if the relation R is clear from context).

We define the ***domain*** of a relation R by

$$\operatorname{dom} R = \{a \colon \exists b \colon (a, b) \in R\}$$

and the ***range*** by

$$\operatorname{ran} R = \{b \colon \exists a \colon (a, b) \in R\}.$$

Equivalence Relations

1.4.3. Definition. We say that a relation R on a set M is

(i) ***reflexive*** if $(a, a) \in R$ for all $a \in M$;

(ii) ***symmetric*** if $(a, b) \in R$ implies $(b, a) \in R$ for all $a, b \in M$;

(iii) ***anti-symmetric*** if $(a, b) \in R$ and $(b, a) \in R$ implies $a = b$ for all $a, b \in M$;

(iv) ***transitive*** if $(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$ for all $a, b, c \in M$.

A reflexive, symmetric and transitive relation on M is called an ***equivalence relation on*** M. In more intuitive notation

- reflexivity means $a \sim a$ for all $a \in M$,
- symmetry means $a \sim b \Rightarrow b \sim a$ for all $a, b \in M$,
- anti-symmetry means $(a \sim b \wedge b \sim a) \Rightarrow a = b$ for all $a, b \in M$,
- transitivity means $a \sim b \wedge b \sim c \Rightarrow a \sim c$ for all $a, b, c \in M$.

Relations

1.4.4. Examples.

- The relation $R = \{(a, b) \in \mathbb{N}^2 : a > b\}$ includes the pairs $(1, 0), (2, 1), (2, 0)$ but not the pair $(0, 1)$. We write $(a, b) \in R \Leftrightarrow a \sim b \Leftrightarrow a > b$ and notice that $\sim$ is transitive (since $a > b$ and $b > c$ implies $a > c$), but not symmetric ($a > b \not\Rightarrow b > a$) or reflexive ($a \not> a$).
- For $n \in \mathbb{N}$ we define the ***integer sum*** $I(n)$ as the sum of all integers that compose the number, e.g. $I(125) = 1 + 2 + 5 = 8$, $I(78) = 7 + 8 = 15$.

  Then the relation $R = \{(a, b) \in \mathbb{N}^2 : I(a) = I(b)\}$ includes the pairs $(22, 4), (14, 5), (3, 30)$ but not the pair $(4, 1)$. We note that R is reflexive ($I(a) = I(a)$), symmetric (if $I(a) = I(b)$, then also $I(b) = I(a)$) and transitive if $I(a) = I(b)$ and $I(b) = I(c)$, then $I(a) = I(c)$), so R is an equivalence relation. We may then call all numbers with equal integer sums ***equivalent*** in this sense.

Equivalence Classes

A ***partition*** of a set A is a family $\mathscr{F}$ of disjoint subsets of A such that their union is A. An element of a partition is called a ***fiber*** or an ***equivalence class***. An element of such an equivalence class is called a ***representative*** of the class.

1.4.5. Example. Denote by $2\mathbb{N} = \{0, 2, 4, 6, ...\} \subset \mathbb{N}$ the set of all even natural numbers and by $2\mathbb{N} + 1 = \{1, 3, 5, 7, ...\} \subset \mathbb{N}$ the set of all odd natural numbers. Since

$$2\mathbb{N} \cap (2\mathbb{N} + 1) = \emptyset \qquad \text{and} \qquad 2\mathbb{N} \cup (2\mathbb{N} + 1) = \mathbb{N}$$

it follows that $\mathscr{F} = \{2\mathbb{N}, 2\mathbb{N} + 1\}$ is a partition of $\mathbb{N}$ and $2\mathbb{N}$ and $2\mathbb{N} + 1$ are the two equivalence classes of this partition. The number 4 is a representative of $2\mathbb{N}$, as are 0, 128 and 456, and the numbers 1, 7, 457 are representatives of $2\mathbb{N} + 1$.

Equivalence Classes

We often denote equivalence classes by one of their representatives enclosed in square brackets, so we might write

$$2\mathbb{N} = [0] \qquad \text{and} \qquad 2\mathbb{N} + 1 = [1].$$

1.4.6. Theorem. Every partition $\mathscr{F}$ of M induces an equivalence relation $\sim$ on a set M by

$$a \sim b \quad :\Leftrightarrow \quad a, b \in M \text{ are in the same equivalence class.} \tag{1.4.2}$$

Proof.
It is easily seen that the relation $\sim$ defined by (1.4.2) is reflexive, symmetric and transitive. □

Fundamental Theorem on Partitions and Equivalence Relations

1.4.7. Theorem. Every equivalence relation $\sim$ on a set M induces a partition $\mathscr{F} = \{[a] : a \in M\}$ of M by

$$a \in [b] \qquad :\Leftrightarrow \qquad a \sim b. \tag{1.4.3}$$

We write $\mathscr{F} = M/\sim$.

Proof.

We need to prove that $\mathscr{F}$ is a partition,. i.e., that the union of all classes in $\mathscr{F}$ is M and that the classes are disjoint.

Since $\sim$ is **reflexive**, it follows that $a \in [a]$ for every $a \in M$, so every a is in some class. Thus, the union of all classes is M.

Fundamental Theorem on Partitions and Equivalence Relations

Proof (continued).

We next show that the classes are disjoint. Let $[a], [b]$ be two classes. Assume that there is an element $c \in M$ such that $c \in [a]$ and $c \in [b]$, so $c \sim b$ and $c \sim a$.

Let $x \in [a]$ be an arbitrary element. Then by **symmetry** and **transitivity**

$$\begin{array}{lllllll} x \in [a] & \Rightarrow & x \sim a & \Rightarrow & a \sim x & \Rightarrow & c \sim x \\ & \Rightarrow & x \sim c & \Rightarrow & x \sim b & \Rightarrow & x \in [b] \end{array}$$

so $[a] \subset [b]$.

Changing the roles of a and b, we have $[b] \subset [a]$, hence $[a] = [b]$. Thus, if two classes have a single element in common, they are the same class. Hence, the classes are disjoint. $\square$

The Integers

We will now introduce the negative numbers. One big deficiency in the natural numbers is that there is no ***inverse element*** for addition, i.e., for every $n \in \mathbb{N}$ we would like to have an element $-n$ such that

$$n + (-n) = 0$$

Such an element does not exist in $\mathbb{N}$. We therefore consider the set of ordered pairs

$$\mathbb{N}^2 = \{(n, m) \colon m, n \in \mathbb{N}\}.$$

We can consider $\mathbb{N}$ as a natural subset of $\mathbb{N}^2$ by replacing $n \in \mathbb{N}$ with $(n, 0) \in \mathbb{N}^2$. Furthermore, we define the following equivalence relation on $\mathbb{N}^2$:

$$(n, m) \sim (p, q) \qquad :\Leftrightarrow \qquad n + q = m + p. \tag{1.4.4}$$

Therefore, the pair $(5, 0)$ (which corresponds to $5 \in \mathbb{N}$) is equivalent to $(6, 1)$, because $5 + 1 = 0 + 6$.

The Integers

We have the following facts:

- (1.4.4) defines an equivalence relation, which induces a partition $\mathbb{Z} = \mathbb{N}^2/\sim$ on $\mathbb{N}^2$.
- Every pair of the form $(n, 0) \in \mathbb{N}^2$, $n \in \mathbb{N}$, is in a different equivalence class of this partition. We denote these equivalence classes by $[+n] \ni (n, 0)$.
- Every pair of the form $(0, n) \in \mathbb{N}^2$, $n \in \mathbb{N}$, $n \geq 1$, is in yet another equivalence class, denoted by $[-n] \ni (0, n)$.
- Any other pair is in a class $[+n]$ or a class $[-n]$ for some $n \in \mathbb{N}$.
- It follows that

$$\mathbb{Z} = \{[+n] \colon n \in \mathbb{N}\} \cup \{[-n] \colon n \in \mathbb{N} \setminus \{0\}\}.$$

The Integers

We now want to define addition for elements of $\mathbb{N}^2$ by

$$(n, m) + (p, q) = (n + p, m + q). \tag{1.4.5}$$

If $(n, m) \sim (\widetilde{n}, \widetilde{m})$ and $(p, q) \sim (\widetilde{p}, \widetilde{q})$, then

$$(n, m) + (p, q) \sim (\widetilde{n}, \widetilde{m}) + (\widetilde{p}, \widetilde{q}). \tag{1.4.6}$$

This means that we can define the sum of two equivalence classes by (1.4.5): let $[\pm n], [\pm m] \in \mathbb{Z}$. Then we define $[\pm n] + [\pm m]$ as the class whose representatives are obtained by adding any representative of $[\pm n]$ to any representative of $[\pm m]$ according to (1.4.5). The fact that the result does not depend on which representative we choose is expressed by (1.4.6).

We say that the so defined addition on $\mathbb{Z}$ is ***independent of the chosen representative*** and therefore ***well-defined***.

The Integers

Note that

$$(n, m) + (0, 0) = (n, m), \qquad (n, m) + (p, q) = (p, q) + (n, m)$$

and we also have the associative law of addition.Therefore, the addition on $\mathbb{Z}$ also has these properties and $[0] \in \mathbb{Z}$ is the neutral element of addition.

Furthermore, let $(n, m) \in \mathbb{N}^2$. Then

$$(n, m) + (m, n) = (n + m, n + m) \sim (0, 0).$$

In particular, $[n] + [-n] = [0]$, so we now have an inverse element for the addition defined on $\mathbb{Z}$.

The set $\mathbb{Z}$ is known as the set of ***integers***. We write n instead of $[n]$ and $-n$ instead of $[-n]$. Furthermore, we abbreviate $n + (-m)$ by $n - m$ and call the "operation" $-$ ***subtraction***. The letter $\mathbb{Z}$ is used for historical reasons: it stands for the German word for numbers, ***Zahlen***.

The Integers

Let us review our strategy up to this point:

- The set of natural numbers was too small, because it didn't include an inverse element of addition.
- We introduce pairs of natural numbers and identify the natural numbers with certain pairs $(n \leftrightarrow (n, 0))$.
- We introduce a suitable equivalence relation on $\mathbb{N}^2$ and take $\mathbb{Z} = \mathbb{N}^2/\sim$ to be the induced partition.
- We characterize the partition $\mathbb{Z} = \{[n]\} \cup \{[-n]\}$.
- We define addition on $\mathbb{N}^2$ so that it is compatible with the previously defined addition on $\mathbb{N}$, i.e., we have $n \leftrightarrow (n, 0)$, $m \leftrightarrow (m, 0)$ and $(n, 0) + (m, 0) = (n + m, 0) \leftrightarrow n + m$.
- We show that the addition on $\mathbb{N}^2$ is compatible with the equivalence classes and can hence be used to define the sum of two classes in $\mathbb{Z}$.
- We see that for the addition in $\mathbb{Z}$ we now have an inverse element for every element of $\mathbb{Z}$.

Multiplication

The initial goal was to expand the natural numbers such as the new set of numbers, together with the operation of addition, forms a group.

However, we also need to discuss the effect of our extension on the operation of multiplication. Based on the intuitive idea that

$$(m-n)\cdot(p-q) = m\cdot p + n\cdot q - m\cdot q - n\cdot p$$

we define multiplication on $\mathbb{N}^2$ by

$$(m,n)\cdot(p,q) := (m\cdot p + n\cdot q, m\cdot q + n\cdot p)$$

It will be shown in the assignments that this multiplication

- allows the multiplication of the equivalence classes $[n]$ and $[-n]$, so that it is well-defined on $\mathbb{Z} = \mathbb{N}^2/\sim$,
- coincides with the known multiplication on $\mathbb{N} \subset \mathbb{Z}$,
- is associative and commutative and has the identity element $(1,0)$,

i.e., retains the properties of multiplication on $\mathbb{N}$ (see Slide 82).

Rings

1.4.8. Definition. A ***ring*** is a triple $(R, +, \cdot)$ consisting of a set G and two ***binary operations*** $+, \cdot : R \times R \to R$ such that

(i) $(R, +)$ is an abelian group

(ii) there exists an element $1 \in R$ such that

$$a \cdot 1 = 1 \cdot a = a \qquad \text{for all } a \in R$$

(existence of a multiplicative unit element)

(iii) for any $a, b, c \in R$,

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

(asociativity)

(iv) for any $a, b, c \in R$,

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c), \qquad (b + c) \cdot a = (b \cdot a) + (c \cdot a)$$

(distributivity)

A ring is called ***commutative*** if in addition to the above properties

(iv) $a \cdot b = b \cdot a$ for all $a, b \in R$ (commutativity)

The Ring of the Integers

1.4.9. Definition. A ring $(R, +, \cdot)$ is said to be an ***integral domain*** if

$$a \cdot b = 0 \quad \Rightarrow \quad a = 0 \quad \vee \quad b = 0$$

for all $a, b \in R$.

It can be shown that $\mathbb{Z}$ is a commutative integral domain with respect to addition and multiplication, but the proof is a little technical. See, for example,

https://proofwiki.org/wiki/Ring_of_Integers_has_no_Zero_Divisors

for details.

The Rationals

We can of course now go further: the integers resulted from desiring an inverse to the addition of natural numbers. However, the multiplication still has no such inverse in the integers. We can now proceed as before: We consider $\mathbb{Z}^2$, the set of pairs of integers. We define the equivalence relation

$$(n, m) \sim (p, q) \qquad :\Leftrightarrow \qquad n \cdot q = m \cdot p. \tag{1.4.7}$$

for $(n, m), (p, q) \in \mathbb{Z} \times \mathbb{Z} \setminus \{0\}$. This is of course inspired by wanting

$$\frac{n}{m} = \frac{p}{q}.$$

We denote the set of rational numbers by $\mathbb{Q} := (\mathbb{Z} \times \mathbb{Z} \setminus \{0\})/\sim$ and we consider $\mathbb{Z}$ as a subset of $\mathbb{Z} \times \mathbb{Z} \setminus \{0\}$ by associating $n \leftrightarrow (n, 1)$.

We will ordinarily identify a representative (n, m) with its class $[(n, m)]$ and write

$$(n, m) =: \frac{n}{m} \in \mathbb{Q}.$$

Multiplication and Addition

We define the product and sum of two pairs of integers by

$$(n, m) \cdot (p, q) := (n \cdot p, m \cdot q) \tag{1.4.8}$$

$$(m, n) + (p, q) := (q \cdot m + p \cdot n, nq). \tag{1.4.9}$$

As before, it can be shown that these operations can be used to define addition and multiplication of classes in $\mathbb{Q} = (\mathbb{Z} \times \mathbb{Z} \setminus \{0\})/\sim$ and that they extend the previously defined operations on $\mathbb{Z}$ to $\mathbb{Q}$, preserving all the ring properties of Definition 1.4.8.

The neutral element of multiplication is $[(1, 1)]$ and every element $[(n, m)] \in \mathbb{Q}$ except $[(0, 1)]$ has a multiplicative inverse

$$[(n, m)]^{-1} = [(m, n)].$$

Fields

Of course, there is also a general algebraic structure to describe the properties of the rational numbers:

1.4.10. Definition. Let $(F, +, \cdot)$ be a commutative ring with unit element of addition 0 and unit element of multiplication 1. Then F is said to be a ***field*** if

(i) $0 \neq 1$

(ii) For every $a \in F \setminus \{0\}$ there exists an element a^{-1} such that

$$a \cdot a^{-1} = 1.$$

Another way of writing this definition is to say that $(F, +, \cdot)$ is a field if $(F, +)$ and $(F \setminus \{0\}, \cdot)$ are abelian groups, $0 \neq 1$ and the law of distributivity holds.

Comparing with our previous construction, it is clear that $\mathbb{Q}$ is a field.

The Modulus of a Rational Number

We introduce a final notation for later use: for any rational number $a \in \mathbb{Q}$, the symbol $|a|$ (called the ***modulus*** or ***absolute value*** of a) shall be defined as

$$|a| := \begin{cases} a & \text{if } a \geq 0, \\ -a & \text{if } a < 0. \end{cases}$$

In the following section, we will proceed to investigate some of the number sets we have just defined more closely, focusing particularly on the integers.

Functions

1.5.1. Definition. Let X and Y be sets. A ***function from*** X ***to*** Y, denoted $f : X \to Y$, is defined as a relation

$$f = \{(x, y) \in X \times Y : P(x, y)\}$$

for some predicate P with the properties that

$$\operatorname{dom} f = X \tag{1.5.1}$$

and

$$\forall_{(x_1,y_1)\in f} \; \forall_{(x_2,y_2)\in f} \; (x_1 = x_2 \;\Rightarrow\; y_1 = y_2). \tag{1.5.2}$$

The property (1.5.2) guarantees that for $(x, y) \in f$, y is uniquely determined by x. We denote this y by $y = f(x)$.

Domain, Range and Properties of Functions

1.5.2. Remark. As for general relations, the domain and range are defined by

$$\operatorname{dom} f = \{x : \exists y : (x, y) \in f\},$$
$$\operatorname{ran} f = \{y : \exists x : (x, y) \in f\}.$$

1.5.3. Example.

- $f = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} : y = x^2\}$ is a function.
- $f = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} : y^2 = x^2\}$ is not a function since $(1, 1)$ and $(1, -1)$ are both in f.

1.5.4. Definition. A function $f : X \to Y$ is said to be

- ***injective*** if $\underset{x_1, x_2 \in X}{\forall} (f(x_1) = f(x_2) \Rightarrow x_1 = x_2)$.
- ***surjective*** if $\underset{y \in Y}{\forall} \underset{x \in X}{\exists} f(x) = y$.
- ***bijective*** if f is both injective and surjective.

Inverse of a Functions

Let

$$f = \{(x, y) \in X \times Y : P(x, y)\}$$

be a function. We define

$$f^{-1} := \{(y, x) \in Y \times X : P(x, y)\}.$$

If f is surjective, f^{-1} satisfies property (1.5.1). If f is injective, f^{-1} satisfies property (1.5.2). We note:

1.5.5. Definition and Theorem. If $f : X \to Y$ is a bijective function, the relation $f^{-1} : Y \to X$ (called the ***inverse function*** to f) is a bijective function. Furthermore,

$$f^{-1}(f(x)) = x \qquad \text{and} \qquad f(f^{-1}(y)) = y$$

Sequences

One of the simplest type of functions are ***sequences of rational numbers***, functions $\mathbb{N} \to \mathbb{Q}$ or, more generally, $\Omega \to \mathbb{Q}$ where $\Omega \subset \mathbb{N}$. For example, we might have the sequence

$$\left\{ \left(n, \frac{1}{n} \right) : n \in \mathbb{N} \setminus \{0\} \right\}.$$

which has domain $\mathbb{N} \setminus \{0\}$.

We denote a sequence with domain $\mathbb{N}$ by $(a_n)_{n\in\mathbb{N}}$, $(a_n)_{n=0}^{\infty}$ or simply (a_n) and use analogous notations in case of smaller domains. We sometimes list a sequence by giving its values for $n = 0, 1, 2, 3, \ldots$.

1.5.6. Example. The sequence $(a_n) : n \mapsto n^2$ can be written as

$$(a_n) = (0, 1, 4, 9, 16, 25, \ldots),$$

$(n^2)_{n\in\mathbb{N}}$ or as "the sequence with values $a_n = n^2$."

Convergent and Cauchy Sequences

It is assumed that sequences are familiar from calculus. We recall two important concepts:

1.5.7. Definition. Let (a_n) be a sequence of (for now, rational) numbers.

(i) The sequence (a_n) is said to ***converge with limit*** $a \in \mathbb{Q}$ if

$$\forall_{\varepsilon>0} \quad \exists_{N\in\mathbb{N}} \quad \forall_{n>N} \quad |a_n - a| < \varepsilon.$$

We then write "$\lim_{n\to\infty} a_n = a$" or "$a_n \to a$ as $n \to \infty$."

(ii) The sequence (a_n) is said to be a ***Cauchy sequence*** if

$$\forall_{\varepsilon>0} \quad \exists_{N\in\mathbb{N}} \quad \forall_{m,n>N} \quad |a_n - a_m| < \varepsilon.$$

1.5.8. Remark. Since

$$|a_n - a_m| \leq |a_n - a| + |a_m - a|,$$

every convergent sequence is a Cauchy sequence,

Convergent and Cauchy Sequences

However, not every Cauchy sequence of rational numbers converges. For example, consider the sequence defined recursively as

$$a_0 = 1, \qquad a_{n+1} = \frac{4 + 3a_n}{3 + 2a_n}, \qquad n \in \mathbb{N}.$$

It is easy to see (by induction, if necessary) that $a_n \in \mathbb{Q}$ for all n. It can be shown that (a_n) is a Cauchy sequence. Then the limit a, if it existed, would satisfy

$$a = \frac{4 + 3a}{3 + 2a} \qquad \text{or} \qquad a^2 = 2.$$

But, as we will show later (Theorem 1.6.15), there is no rational number satisfying $a^2 = 2$ so the limit can not exist in $\mathbb{Q}$.

Our aim is to construct the ***completion*** of the rational numbers by "adding" all the limits of Cauchy sequences to $\mathbb{Q}$.

Convergent and Cauchy Sequences

Given $\mathbb{Q}$, we may consider the set of all sequences in $\mathbb{Q}$ that converge to a limit. Denote this set by $\mathrm{Conv}(\mathbb{Q})$. Each sequence $(a_n) \in \mathrm{Conv}(\mathbb{Q})$ is associated uniquely to a number $a \in \mathbb{Q}$, namely its limit.

We now say that two sequences are equivalent if they have the same limit, i.e.,

$$(a_n) \sim (b_n) \qquad :\Leftrightarrow \qquad \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n. \tag{1.5.3}$$

It is easy to check that this is an equivalence relation.

We denote the equivalence class of all sequences with the same limit as a sequence (a_n) by $[(a_n)]$ and denote the corresponding partition by $\mathrm{Conv}(\mathbb{Q})/\sim$.

Since each rational number is represented by a class (why?) we see that the rational numbers may be identified with the set of all classes of convergent sequences:

$$\mathbb{Q} \simeq \mathrm{Conv}(\mathbb{Q})/\sim .$$

Construction of the Real Numbers

We can now consider a larger class of sequences, that of Cauchy sequences of rational numbers, denoted by Cauchy($\mathbb{Q}$). Since every convergent sequence is a Cauchy sequence, Conv($\mathbb{Q}$) $\subset$ Cauchy($\mathbb{Q}$).

Furthermore, we say that two Cauchy sequences are equivalent not if they have the same limit (because they might not converge) but rather if their difference converges to zero:

$$(a_n) \sim (b_n) \qquad :\Leftrightarrow \qquad \lim_{n\to\infty} (a_n - b_n) = 0. \tag{1.5.4}$$

Of course, (1.5.4) is equivalent to (1.5.3) for convergent sequences. We now have the larger set

$$\text{Cauchy}(\mathbb{Q})/\sim \supset \ \text{Conv}(\mathbb{Q})/\sim \simeq \ \mathbb{Q}$$

Construction of the Real Numbers

The set $\text{Cauchy}(\mathbb{Q})/\sim$ incorporates the rational numbers and by its construction every Cauchy sequence (a_n) in $\text{Cauchy}(\mathbb{Q})/\sim$ has a limit, namely precisely the object represented by the class $[(a_n)]$. We write

$$\mathbb{R} := \text{Cauchy}(\mathbb{Q})/\sim$$

and call this set the ***real numbers***.

It can be shown that all the operations of the rational numbers can be extended to $\mathbb{R}$ and that $\mathbb{R}$ is a field

Construction of the Real Numbers

1.5.9. Example. Every rational number has a finite decimal representation. We can think of a real number as having an "infinite decimal representation." (More details on this later.) For example, the sequence

$$1,\ 1.4,\ 1.41, \ldots$$

may converge to $\sqrt{2}$ if the succeding terms are chosen appropriately.

This "infinite decimal representation" is just the way that real numbers are introduced in middle school. As another example, the sequences

$$(a_n) := (0.4,\ 0.49,\ 0.499,\ 0.4999,\ 0.49999, \ldots)$$

and

$$(b_n) := (0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5, \ldots)$$

are equivalent in the sense of (1.5.4), since

$$|a_n - b_n| = 10^{-(n+1)} \xrightarrow{n\to\infty} 0.$$

Hence, 0.499999 ... and 0.5 are considered to represent the same number.

The Division Algorithm

We have constructed the integers as an additive group; however, there is no inverse operation to multiplication. Nevertheless, the theory of divisibility of integers is very rich and full of applications.

1.6.1. Division Algorithm. Given $a, b \in \mathbb{Z}$ with $b > 0$ there exist unique $q, r \in \mathbb{Z}$ such that

$$a = q \cdot b + r \qquad \text{and} \qquad 0 \leq r < b. \tag{1.6.1}$$

The number q is called the ***quotient*** and r is called the ***remainder*** in the division of a by b.

1.6.2. Example. Given the number $a = 25$ and $b = 4$, the decomposition is

$$25 = 6 \cdot 4 + 1$$

where $q = 6$ and $r = 1$ are uniquely determined. The condition $0 \leq r < b$ is essential, or we could also write

$$25 = 5 \cdot 4 + 5 = 7 \cdot 4 - 3$$

and uniqueness is lost.

The Division Algorithm

Proof of the Division Algorithm.

Uniqueness. Given $a, b \in \mathbb{Z}$, suppose that there exist $r, \widetilde{r}$ and $q, \widetilde{q}$ such that

$$a = q \cdot b + r = \widetilde{q} \cdot b + \widetilde{r}.$$

Then

$$b \cdot |q - \widetilde{q}| = |\widetilde{r} - r|. \tag{1.6.2}$$

Since $0 \leq r, \widetilde{r} < b$ we have

$$-b < \widetilde{r} - r < b,$$

and

$$|\widetilde{r} - r| < b. \tag{1.6.3}$$

The Division Algorithm

Proof of the Division Algorithm (continued).
Now (1.6.2) and (1.6.3) imply

$$b \cdot |q - \widetilde{q}| < b$$

so

$$|q - \widetilde{q}| < 1$$

Since $q, \widetilde{q} \in \mathbb{Z}$, this implies $q = \widetilde{q}$. From (1.6.2) we then immediately obtain $\widetilde{r} = r$ and we see that r and q are unique.

The Division Algorithm

Proof of the Division Algorithm (continued).
Existence. Let

$$S(a,b) := \{n \in \mathbb{N} : n = a - x \cdot b,\ x \in \mathbb{Z}\}.$$

We prove that $S(a,b)$ is non-empty: let $x = -|a|$. Then, since $b \geq 1$,

$$n = a + |a| \cdot b \geq a + |a| \geq 0$$

and $n \in S(a,b)$.

By the Well-ordering Principle, $S(a,d)$ has a least element, which we denote by $r \geq 0$. There exists some $x_0 \in \mathbb{Z}$ such that

$$r = a - x_0 \cdot b.$$

The Division Algorithm

Proof of the Division Algorithm (continued).

Furthermore, $r < b$, because otherwise

$$a - (x_0 + 1) \cdot b = r - b \geq 0$$

would be smaller than r and also in $S(a, b)$, contradicting that r is the least element of $S(a, b)$.

Hence, there exist integers r and x_0 such that

$$a = x_0 \cdot b + r \qquad \text{and} \qquad 0 \leq r < b. \qquad \square$$

The Division Algorithm

We can now drop the requirement that $b > 0$ and allow negative non-zero b:

1.6.3. Corollary. Given $a, b \in \mathbb{Z}$ with $b \neq 0$ there exist unique $q, r \in \mathbb{Z}$ such that

$$a = q \cdot b + r \qquad \text{and} \qquad 0 \leq r < |b|.$$

Proof.

The case $b > 0$ is covered by the Division Algorithm, so we assume $b < 0$. Then $|b| = -b > 0$ and

$$a = q' \cdot |b| + r \qquad \text{with} \qquad 0 \leq r < |b|.$$

Since $|b| = -b$, we take $q' = -q$ to obtain the desired representation $\square$

Even and Odd Numbers

1.6.4. Example. Consider arbitrary $a \in \mathbb{Z}$ and $b = 2$. Then there exist unique numbers $k \in \mathbb{Z}$ such that

$$a = 2 \cdot k + r \qquad \text{with} \qquad 0 \leq r < 2$$

so either $r = 0$ or $r = 1$. If $r = 0$, the number a is said to be ***even***, if $r = 1$ then a is said to be ***odd***.

The Division Algorithm establishes that every number is either even or odd and that even numbers can be written as

$$a = 2k$$

while odd numbers can be written as

$$a = 2k + 1$$

where in each case the number k is uniquely determined.

Application of the Division Algorithm

As another application of the Division Algorithm, we show the following statement:

1.6.5. Example. If $n \in \mathbb{Z}$ is odd, then $n^2 = 8k + 1$ for some $k \in \mathbb{N}$.

To see this, we note that by the Division Algorithm n can be written in one of the forms

$$4q, \quad 4q + 1, \quad 4q + 2, \quad 4q + 3.$$

Since n is odd, $n = 4q + 1$ or $n = 4q + 3$. In either case we obtain the result:

$$(4q + 1)^2 = 8(2q^2 + 2) + 1 = 8k + 1,$$
$$(4q + 3)^2 = 8(2q^2 + 3q + 1) + 1 = 8k + 1$$

Divisors and Multiples

A special case of the division algorithm occurs when the remainder $r = 0$. It is worth investigating that case in more detail.

1.6.6. Definition. For $a, b \in \mathbb{Z}$ we define

$$a \mid b \qquad :\Leftrightarrow \qquad \exists_{c \in \mathbb{Z}} \; a \cdot c = b.$$

If $a \mid b$, it is understood that $a \neq 0$ and we say a is a ***divisor*** (or ***factor***) of b and b is a ***multiple*** of a.

If a does not divide b, we write $a \nmid b$.

Note that if $a \mid b$, then also $(-a) \mid b$ since $(-a)(-c) = ac = b$.

Divisors and Multiples

1.6.7. Theorem. For all $a, b, c \in \mathbb{Z}$ the following statements hold:

(i) $a \mid a, \quad 1 \mid a, \quad a \mid 0$,

(ii) $a \mid 1$ if and only if $a = 1$ or $a = -1$,

(iii) $a \mid b$ and $c \mid d$ implies $ac \mid bd$,

(iv) $a \mid b$ and $b \mid a$ if and only if $a = \pm b$,

(v) $a \mid b$ and $b \mid c$ implies $a \mid c$,

(vi) $a \mid b$ with $b \neq 0$ implies $|a| \leq |b|$,

(vii) $a \mid b$ and $a \mid c$ implies $a \mid (xb + yc)$ for any $x, y \in \mathbb{Z}$.

Greatest Common Divisor

If $a, b \in \mathbb{Z}$ we say that d is a ***common divisor*** of a and b if

$$d \mid a \qquad \text{and} \qquad d \mid b.$$

If $a = b = 0$, of course any integer is a common divisor. However, in all other cases, there are only a finite number of common divisors since any common divisor must satisfy

$$|d| \leq \min(|a|, |b|).$$

Greatest Common Divisor

1.6.8. Definition. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. An integer $d \in \mathbb{Z}$ is said to be the ***greatest common divisor*** of a and b if

(i) $d \mid a$ and $d \mid b$,

(ii) $c \mid a$ and $c \mid b$ implies $c \leq d$.

We denote the greatest common divisor by $\gcd(a, b)$.

We can calculate $\gcd(a, b)$ through

$$\gcd(a, b) = \max\{d \in \mathbb{N} : (d \mid a) \wedge (d \mid b)\}$$

1.6.9. Example.

- $\gcd(24, 36) = \max\{1, 2, 3, 4, 6, 12\} = 12$.
- $\gcd(17, 22) = 1$, so 17 and 22 are relatively prime.

The GCD as a Linear Combination - Bézout's Lemma

It turns out that $\gcd(a, b)$ can be expressed as a linear combination of a and b with integer coefficients:

1.6.10. Bézout's Lemma. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. Then there exist $x, y \in \mathbb{Z}$ such that $\gcd(a, b) = ax + by$.

Proof.

Let

$$S(a, b) = \left\{ n \in \mathbb{N} \setminus \{0\} : n = ax + by, \ x, y \in \mathbb{Z} \right\}.$$

The set S is not empty, since we can choose $y = 0$ and $x = a/|a|$ to ensure that $|a| \in S$. By the Well-ordering Principle, S has a least element d, which can be written as

$$d = ax_0 + by_0$$

for $x_0, y_0 \in \mathbb{Z}$. We will show that $d = \gcd(a, b)$.

Bézout's Lemma

Proof (continued).

Using the Division Algorithm, there exist $q, r \in \mathbb{Z}$ such that

$$a = q \cdot d + r, \qquad 0 \leq r < d.$$

Then

$$r = a - q \cdot d = a - q(ax_0 + by_0) = a(1 - qx_0) + b(-qy_0).$$

If $r > 0$, this would imply $r \in S(a, b)$. But $r < d$ and d is the least element of $S(a, b)$, so this gives a contradiction. Therefore, $r = 0$ and $d \mid a$.

A similar argument shows $d \mid b$, so d is a common divisor of a and b.

Bézout's Lemma

Proof (continued).
We now show that if $c \mid a$ and $c \mid b$ for any $c \in \mathbb{Z}$, then $c \leq d$. Suppose that $c > 0$. Then by Theorem 1.6.7,

$$c \mid (ax_0 + by_0) \qquad \text{so} \qquad c \mid d.$$

At the same time,

$$c = |c| \leq |d| = d,$$

completing the proof. □

1.6.11. Corollary. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. Then

$$T(a, b) = \{n \in \mathbb{Z} : n = ax + by,\ x, y \in \mathbb{Z}\}$$

is the set of all integer multiples of $\gcd(a, b)$.

The proof is left for the assignments.

Relatively Prime Integers

1.6.12. Definition. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. If $\gcd(a, b) = 1$, then a and b are said to be ***relatively prime***.

1.6.13. Theorem. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. Then a and b are relatively prime if and only if there exist $x, y \in \mathbb{Z}$ such that

$$ax + by = 1.$$

Proof.

($\Rightarrow$) If $\gcd(a, b) = 1$, then there exist x and y with $ax + by = 1$ by Theorem 1.6.10.

($\Leftarrow$) Suppose that there exist x and y with $ax + by = 1$ and that $d = \gcd(a, b)$. Then $d \mid a$ and $d \mid b$ and (by Theorem 1.6.7) $d \mid (ax + by)$, i.e., $d \mid 1$. But then $d = 1$. $\square$

Relatively Prime Integers

From this, we obtain a useful result:

1.6.14. Corollary. If $\gcd(a, b) = d$, then

$$\gcd\left(\frac{a}{d}, \frac{b}{d}\right) = 1.$$

Proof.
We note that a/d and b/d are of course integers. There exist $x, y \in \mathbb{Z}$ such that $d = ax + by$, so

$$1 = \frac{a}{d}x + \frac{b}{d}y$$

and hence $\gcd(a/d, b/d) = 1$ by Theorem 1.6.13. □

Digression into Fractions

Although we do not want to focus on the rational numbers in this section, we briefly give an application.

Suppose that $a = q \cdot a'$ and $b = q \cdot b'$, where q, a', b' are integers with $a', b' > 0$. Then

$$\frac{a}{b} = \frac{q \cdot a'}{q \cdot b'} = \frac{a'}{b'}.$$

Writing a'/b' for a/b is called ***simplifying the fraction*** a/b.

If $\gcd(a, b) = 1$, so that only the (trivial) choice $q = 1$ is possible when simplifying, we say that a/b is in ***lowest terms***.

Corollary 1.6.14 states that it is always possible to express a fraction in lowest terms by taking $q = \gcd(a, b)$. This fact is used (usually without any comment) in the proof that $\sqrt{2}$ is not a fraction.

The Square Root Problem

1.6.15. Theorem. There exists no fraction whose square is equal to 2.

Proof.
Suppose that $(a/b)^2 = 2$, where $a, b \in \mathbb{N}$ and $\gcd(a, b) = 1$. Then

$$a^2 = 2b^2,$$

so a^2 is even. By Theorem 1.1.24, this implies that a is even, so $a = 2k$ for some $k \in \mathbb{N}$. But then

$$2b^2 = 4k^2 \quad \text{implies} \quad b^2 = 2k^2 \text{ is even} \quad \text{implies} \quad b \text{ is even.}$$

Thus, a and b are both divisible by 2, giving a contradiction. □

Further Consequences of Bézout's Lemma

1.6.16. Corollary. Let $a, b, c \in \mathbb{Z}$ with $\gcd(a, b) = 1$. Then

$$a \mid c \quad \text{and} \quad b \mid c \qquad \text{implies} \qquad a \cdot b \mid c.$$

Proof.

By assumption, there exist integers $r, s, x, y \in \mathbb{Z}$ such that

$$c = a \cdot r = b \cdot s \qquad \text{and} \qquad a \cdot x + b \cdot y = 1.$$

This implies

$$\begin{aligned} c &= c(ax + by) = acx + bcy = a(bs)x + b(ar)y \\ &= ab(sx + ry), \end{aligned}$$

so $ab \mid c$. □

Further Consequences of Bézout's Lemma

1.6.17. Euclid's Lemma. Let $a, b, c \in \mathbb{Z}$ with $\gcd(a, b) = 1$. Then

$$a \mid bc \qquad \text{implies} \qquad a \mid c.$$

Proof.
There exist $x, y \in \mathbb{Z}$ such that $ax + by = 1$. Then

$$c = acx + bcy$$

Since $a \mid bc$ by assumption and $a \mid ac$ trivially, we have $a \mid c$. □

1.6.18. Remark. It is essential that $\gcd(a, b) = 1$, as can be seen from $12 \mid 9 \cdot 8$, but $12 \nmid 9$ and $12 \nmid 8$.

Alternative Definition of the GCD

In general rings one may not have a notion of "$<$", so it is necessary to recast the greatest common divisor. The following result shows that Definition 1.6.8 is equivalent to a definition not involving the ordering relation.

1.6.19. Theorem. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. Then $d \in \mathbb{N}$ is the greatest common divisor of a and b, $d = \gcd(a, b)$, if and only if

(i) $d \mid a$ and $d \mid b$

(ii) $c \mid a$ and $c \mid b$ implies $c \mid d$.

Note that the only difference to Definition 1.6.8 is that $c \leq d$ has been replaced by $c \mid d$ in (ii).

Alternative Definition of the GCD

Proof.

($\Rightarrow$) Suppose $d = \gcd(a, b)$. Then there exist $x, y \in \mathbb{Z}$ such that

$$d = ax + by.$$

If $c \mid a$ and $c \mid b$, then $c \mid d$.

($\Leftarrow$) Suppose $d \in \mathbb{N}$ satisfies $d \mid a$, $d \mid b$ and for any c such that $c \mid a$ and $c \mid b$ we know that $c \mid d$. Then by Theorem 1.6.7 (vi) any such c must satisfy $c \leq d$. Hence, d must be the greatest common divisor. $\square$

The Euclidean Algorithm

We now turn to the problem of calculating the greatest common divisor of two integers, $\gcd(a, b)$. Finding all common divisors and simply determining the greatest among them is often not practical. The Euclidean Algorithm is given in the ***Elements*** of Euclid (ca. 300 B.C.).

The procedure is it follows: we apply the Division Algorithm to a and b, yielding $q_1, r_1 \in \mathbb{Z}$ such that

$$a = bq_1 + r_1, \qquad 0 \leq r_1 < b.$$

If $r_1 = 0$, then $\gcd(a, b) = b$. Otherwise, we apply the Division Algorithm again, yielding $q_2, r_2 \in \mathbb{Z}$ such that

$$b = r_1 q_2 + r_2, \qquad 0 \leq r_2 < r_1$$

If $r_2 = 0$, we stop, otherwise continue.

The Euclidean Algorithm

We obtain a sequence of equations as follows:

$$\begin{aligned}
a &= bq_1 + r_1, & 0 \leq r_1 < b,\\
b &= r_1 q_2 + r_2, & 0 \leq r_2 < r_1,\\
r_1 &= r_2 q_3 + r_3, & 0 \leq r_3 < r_2,\\
&\vdots\\
r_{n-2} &= r_{n-1} q_n + r_n & 0 \leq r_n < r_{n-1}\\
r_{n-1} &= r_n q_{n+1} + 0,
\end{aligned}$$

Since we have a strictly decreasing sequence of remainders,

$$b > r_1 > r_2 > \cdots > r_n \geq 0,$$

we must eventually arrive at an equation with remainder zero.

The Euclidean Algorithm

We claim that

$$r_n = \gcd(a, b).$$

This follows from the claim (which we will prove momentarily) that if $a = bq + r$, then

$$\gcd(a, b) = \gcd(b, r).$$

Applying this to the system of equations, we have

$$\begin{aligned}
\gcd(a, b) &= \gcd(b, r_1) \\
&= \gcd(r_1, r_2) \\
&\vdots \\
&= \gcd(r_{n-1}, r_n) \\
&= \gcd(r_n, 0) = r_n.
\end{aligned}$$

The Euclidean Algorithm

1.6.20. Lemma. Let $a, b, q, r \in \mathbb{Z}$ with

$$a = bq + r.$$

Then

$$\gcd(a, b) = \gcd(b, r).$$

Proof.
Let $d = \gcd(a, b)$ and $a = bq + r$. By Theorem 1.6.7 (vii),

$$d \mid (a - bq) \qquad \text{so} \qquad d \mid r.$$

Hence, d is a common divisor of b and r.

Suppose $c \in \mathbb{Z}$ satisfies $c \mid b$ and $c \mid r$. Then $c \mid (bq + r)$, so $c \mid a$. Thus, c is a common divisor of a and b and $c \leq \gcd(a, b) = d$. It follows that d is not less than any divisor of b and r and we conclude $d = \gcd(b, r)$. □

The Euclidean Algorithm

We also see how to express the greatest common divisor as a linear combination of a and b,

$$\gcd(a, b) = ax + by.$$

We start from the last line and work our way upwards:

$$\begin{aligned}\gcd(a, b) = r_n &= r_{n-2} - q_n r_{n-1} \\ &= r_{n-2} - q_n(r_{n-3} - q_{n-1} r_{n-2}) \\ &= (1 + q_n q_{n-1}) r_{n-2} + (-q_n) r_{n-3}.\end{aligned}$$

Continuing in this way, we can finally express $\gcd(a, b)$ as a linear combination of a and b.

Calculating the GCD

1.6.21. Example. We calculate $\gcd(12378, 3054)$ as follows:

$$\begin{aligned} 12378 &= 4 \cdot 3054 + 162 \\ 3054 &= 18 \cdot 162 + 138 \\ 162 &= 1 \cdot 138 + 24 \\ 138 &= 5 \cdot 24 + 18 \\ 24 &= 1 \cdot 18 + 6 \\ 18 &= 3 \cdot 6 + 0. \end{aligned}$$

We see that $\gcd(12378, 3054) = 6$.

Expressing the GCD as a Linear Combination

In order to find $x, y \in \mathbb{Z}$ such that $6 = 12378 \cdot x + 3054 \cdot y$, we write

$$\begin{aligned}
6 &= 24 - 18 \\
&= 24 - (138 - 5 \cdot 24) \\
&= 6 \cdot 24 - 138 \\
&= 6 \cdot (162 - 138) - 138 \\
&= 6 \cdot 162 - 7 \cdot 138 \\
&= 6 \cdot 162 - 7 \cdot (3054 - 18 \cdot 162) \\
&= 132 \cdot 162 - 7 \cdot 3054 \\
&= 132 \cdot (12378 - 4 \cdot 3054) - 7 \cdot 3054 \\
&= 132 \cdot 12378 + (-535) \cdot 3054
\end{aligned}$$

Of course, this is not the only such representation - why?

Calculating the GCD from Smaller Numbers

A very useful result for practical calculations is the following:

1.6.22. Lemma. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$ and $k \neq 0$. Then

$$\gcd(ka, kb) = |k| \cdot \gcd(a, b).$$

Proof.
We discuss first the case $k > 0$. To find $\gcd(ka, kb)$, we multiply the equations in the Euclidean Algorithm for $\gcd(a, b)$ by k, so the final remainder is $k \cdot r_n = k \cdot \gcd(a, b)$.

For $k < 0$,

$$\gcd(ka, kb) = \gcd(-ka, -kb) = \gcd(|k|a, |k|b) = |k| \cdot \gcd(a, b). \qquad \square$$

Least Common Multiple

The counterpart to the greatest common divisor is the ***least common multiple***. Given a and b, the products $a \cdot b$ is a multiple of a and b and of course there are other multiples.

The least common multiple exists by the Well-Ordering principle, because the set of all positive multiples is non-empty (why?).

We make a formal definition:

1.6.23. Definition. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. An integer $m \in \mathbb{Z}$ is said to be the ***least common multiple*** of a and b if

(i) $a \mid m$ and $b \mid m$,

(ii) for any $c > 0$, $a \mid c$ and $b \mid c$ implies $m \mid c$.

We denote the least common multiple by $\operatorname{lcm}(a, b)$.

Relationship Between LCM and GCD

1.6.24. Lemma. Let $a, b \in \mathbb{Z}$ with $|a| + |b| \neq 0$. Then

$$\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b.$$

Proof.

Let $d = \gcd(a, b)$ and write

$$a = d \cdot r, \qquad b = d \cdot s \qquad \text{for } r, s \in \mathbb{Z}.$$

We set

$$m = \frac{a \cdot b}{d}$$

so

$$m = as = rb$$

and m is a common multiple of a and b.

Relationship Between LCM and GCD

Proof (continued).

We need to show that m is the least common multiple. Suppose that

$$c = au = bv \qquad \text{for } u, v \in \mathbb{Z}.$$

Write $d = ax + by$ for some $x, y \in \mathbb{Z}$. Then

$$\begin{aligned} \frac{c}{m} &= \frac{cd}{ab} = \frac{c(ax+by)}{ab} \\ &= \frac{c}{b}x + \frac{c}{a}y \\ &= vx + uy. \end{aligned}$$

Hence, $m \mid c$. It follows that m is the least common multiple. $\square$

Linear Diophantine Equation

With the tools now at our disposal, we can discuss the simplest so-called ***diophantine equations***. In general, these are polynomial equations in two or more variables for which integer solutions are sought. They were studied by the Greek mathematician Diophantus around 300 AD.

Today, diophantine equations play a very important role in certain cryptographic methods, such as ***elliptic curve cryptography***. We will discuss this in more detail in a later section. In the present section, we are concerned with the most simple case.

A ***linear diophantine equation in two variables*** has the form

$$ax + by = c, \qquad a, b, c \in \mathbb{Z},\ |a| + |b| \neq 0, \tag{1.6.4}$$

A solution is a pair $(x_0, y_0) \in \mathbb{Z}^2$ such that $ax_0 + by_0 = c$.

Solvability Condition and Structure of Solutions

1.6.25. Example. The equation

$$3x + 6y = 18$$

has several solutions, e.g., $(4, 1)$, $(-6, 6)$, $(10, -2)$.

The equation

$$2x + 10y = 17$$

has no solutions - why?

1.6.26. Theorem. The linear diophantine equation $ax + by = c$ has a solution if and only if $d \mid c$, where $d = \gcd(a, b)$.

Furthermore, if (x_0, y_0) is a solution, then for any $t \in \mathbb{Z}$ another solution is given by

$$x = x_0 + \frac{b}{d}t, \qquad y = y_0 - \frac{a}{d}t. \tag{1.6.5}$$

Any solution of the equation has this form, i.e., (1.6.5) yields all solutions.

Solvability Condition and Structure of Solutions

Proof.

We first prove the statement on the existence of solutions:

($\Leftarrow$) Suppose that $ax_0 + by_0 = c$, so (x_0, y_0) is a solution. There exist $r, s \in \mathbb{Z}$ such that $a = dr$, $b = ds$. Then

$$c = ax_0 + by_0 = drx_0 + dsy_0$$

so $d \mid c$.

($\Rightarrow$) Suppose that $d \mid c$. Then there exists a $t \in \mathbb{Z}$ such that $c = dt$. By Bézout's Lemma, there exist $x_0, y_0 \in \mathbb{Z}$ such that

$$d = ax_0 + by_0.$$

Then

$$c = dt = (ax_0 + by_0)t = a(tx_0) + b(ty_0).$$

and (tx_0, ty_0) is a solution to the equation.

□

Solvability Condition and Structure of Solutions

Proof (continued).

Next, we establish the structure of the solutions.

(i) For any $t \in \mathbb{Z}$,

$$x = x_0 + \frac{b}{d}t, \qquad y = y_0 - \frac{a}{d}t \tag{1.6.6}$$

solves the equation, since

$$ax + by = ax_0 + by_0 + \frac{ab}{d}t - \frac{ab}{d}t = c.$$

(ii) Finally, we show that every solution has this form. Suppose (x', y') is some solution to the equation. Then

$$ax_0 + by_0 = c = ax' + by'.$$

Solvability Condition and Structure of Solutions

Proof (continued).

(ii) Thus,

$$a(x_0 - x') = b(y' - y_0).$$

Since $d = \gcd(a, b)$, there exist $r, s \in \mathbb{Z}$ such that $a = dr$, $b = ds$. Corollary 1.6.14 implies that $\gcd(r, s) = 1$. We find

$$r(x' - x_0) = s(y_0 - y').$$

We observe that $r \mid s(y_0 - y')$ with $\gcd(r, s) = 1$. Euclid's Lemma 1.6.17 then implies

$$r \mid (y_0 - y').$$

Thus, there exists $t \in \mathbb{Z}$ such that

$$y_0 - y' = rt \qquad \text{and so} \qquad x' - x_0 = st.$$

Solvability Condition and Structure of Solutions

Proof (continued).

(ii) We can now plug everything in:

$$x' = x_0 + st = x_0 + \frac{b}{d}t, \qquad y' = y_0 - rt = y_0 - \frac{a}{d}t.$$

This completes the proof. □

Solving a Diophantine Equation

1.6.27. Example. We solve the diophantine equation

$$172x + 20y = 1000$$

We first need the greatest common divisor of the coefficients 172 and 20. Euclid's algorithm gives

$$\begin{aligned} 172 &= 8 \cdot 20 + 12, \\ 20 &= 1 \cdot 12 + 8, \\ 12 &= 1 \cdot 8 + 4, \\ 8 &= 2 \cdot 4 + 0 \end{aligned}$$

so $\gcd(172, 20) = 4$. Since $4 \mid 1000$, a solution exists.

Solving a Diophantine Equation

To find a particular solution of

$$172x + 20y = 1000$$

we express 4 as a linear combination of 172 and 20, then multiply our equation by 250.

Working backwards through Euclid's algorithm,

$$\begin{aligned} 4 &= 12 - 8 \\ &= (20 - 8) - 8 \\ &= 20 - 2(20 - 12) \\ &= -20 + 2(172 - 8 \cdot 20) \\ &= -17 \cdot 20 + 2 \cdot 172 \end{aligned}$$

so

$$172 \cdot 500 + 20 \cdot (-4250) = 1000.$$

Solving a Diophantine Equation

The general solution is then

$$x = 500 + \frac{20}{4}t = 500 + 5t,$$
$$y = -4250 - \frac{172}{4}t = -4250 - 43t$$

for $t \in \mathbb{Z}$.

If we are interested in finding a solution in $\mathbb{N}^2$, we must require

$$x = 500 + 5t > 0, \qquad y = -4250 - 43t > 0$$

or

$$-(98 + 36/43) > t > -100$$

which amounts to $t = -99$. The unique positive solution is $x = 5$, $y = 7$.

Prime Numbers

1.7.1. Definition. A number $p \in \mathbb{N} \setminus \{0, 1\}$ is said to be ***prime*** if the only positive divisors of p are 1 and p.

If p is not prime, it is said to be ***composite***.

In the assignments, you will show the following simple consequence of Euclid's Lemma:

1.7.2. Theorem. Let $p \in \mathbb{N} \setminus \{0, 1\}$ and $a, b \in \mathbb{Z}$. If p is prime and $p \mid ab$, then $p \mid a$ or $p \mid b$.

The following generalization is a straightforward application of the induction principle:

1.7.3. Corollary. Let $p \in \mathbb{N} \setminus \{0, 1\}$ and $a_1, \ldots, a_n \in \mathbb{Z}$. If p is prime and $p \mid a_1 a_2 \ldots a_n$, then $p \mid a_k$ for some k with $1 \leq k \leq n$.

Prime Numbers

Proof.
The statement is true for $n = 2$ by Theorem 1.7.2. Suppose the statement is true for $n - 1$ and $p \mid a_1 a_2 \dots a_n$. Then by Theorem 1.7.2, either

$$p \mid a_1 a_2 \dots a_{n-1} \qquad \text{or} \qquad p \mid a_n.$$

If $p \mid a_n$, we are finished. Otherwise, by the induction hypothesis, $p \mid a_k$ for some k with $1 \leq k \leq n - 1$. Thus $p \mid a_k$ for some k with $1 \leq k \leq n$. By the induction principle, the statement is true for all $n \in \mathbb{N}$. □

1.7.4. Corollary. Let $p, q_1, \dots, q_n \in \mathbb{N} \setminus \{0, 1\}$ be prime numbers. If $p \mid q_1 q_2 \dots q_n$, then $p = q_k$ for some k with $1 \leq k \leq n$.

Proof.
By Corollary 1.7.3, $p \mid q_k$ for some k with $1 \leq k \leq n$. Since q_k is prime, this means $p = 1$ or $p = q_k$. Since $p \neq 1$, we must have $p = q_k$. □

Fundamental theorem of Arithmetic

1.7.5. Fundamental Theorem of Arithmetic. Every $n \in \mathbb{N} \setminus \{0, 1\}$ is prime or the product of primes. This product is unique, apart from the order in which the primes occur.

Proof.

We have already seen in Example 1.3.6 that any $p \in \mathbb{N} \setminus \{0, 1\}$ is prime or the product of primes. We now prove the uniqueness of the factorization.

Suppose there exist primes $p_1, \ldots, p_s$ and $q_1, \ldots, q_t$ such that

$$n = p_1 p_2 \ldots p_s = q_1 q_2 \ldots q_t.$$

with

$$p_1 \leq p_2 \leq \cdots \leq p_s, \qquad q_1 \leq q_2 \leq \cdots \leq q_t.$$

Fundamental theorem of Arithmetic

Proof (continued).

By Corollary 1.7.4, $p_1 = q_k$ for some k, so $p_1 \geq q_1$. The same argument give $q_1 \geq p_1$, so we have $p_1 = q_1$. We then cancel $p_1 = q_1$ from the equality, obtaining

$$p_2 \dots p_s = q_2 \dots q_t.$$

If $t > s$, we eventually obtain

$$1 = q_{s+1} \dots q_t > 1$$

which is a contradiction. Hence, $t = s$ and all the factors are equal. □

1.7.6. Corollary. Any $n \in \mathbb{N} \setminus \{0, 1\}$ can be written in the canonical form

$$n = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$$

where $p_1 < p_2 < \dots p_r$ are prime numbers and $k_1, \dots, k_r \in \mathbb{N} \setminus \{0\}$.

The Sieve of Eratosthenes

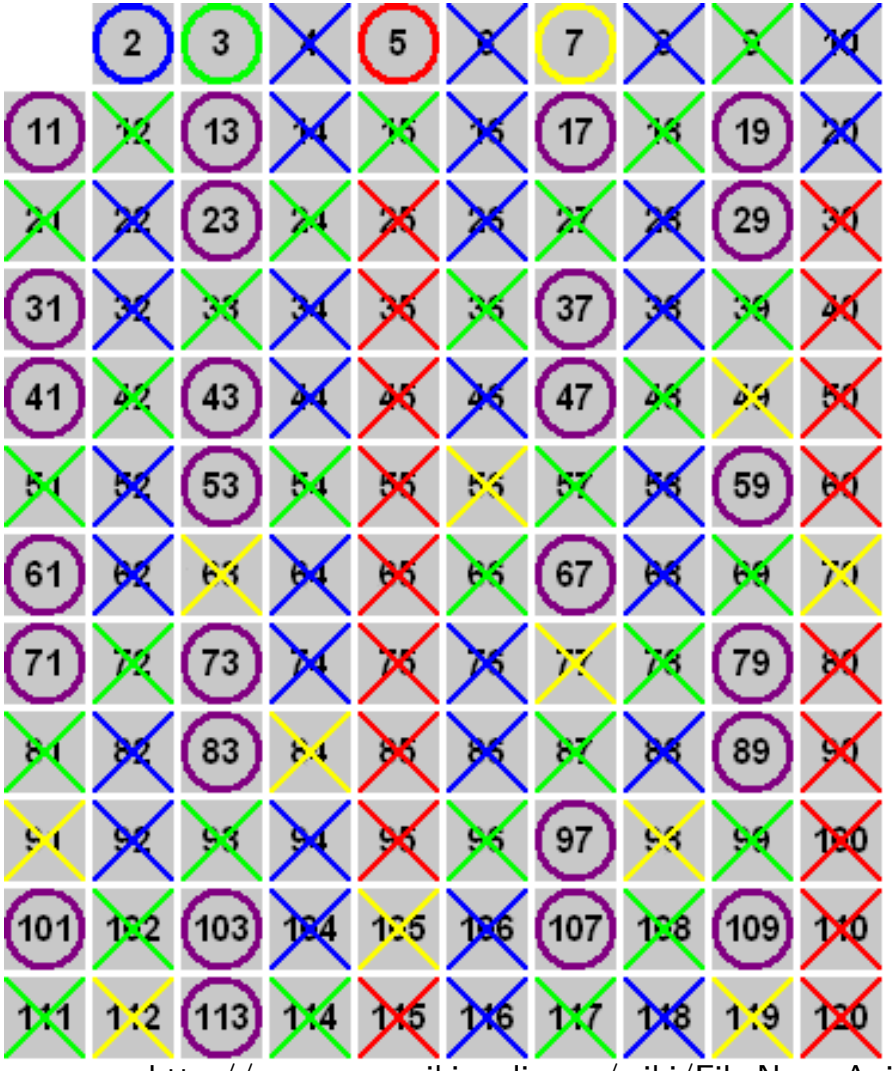
1.7.7. Theorem. If n is a composite integer, then n has a prime divisor whose square is not greater than n.

Proof.
We write $n = ab$ with $a, b \in \mathbb{N}$, $1 < a \leq b$. Then either a^2 or b^2 must be no greater than n. This divisor is either prime or the product of primes. In any case, n has a prime divisor with square no greater than n. □

1.7.8. Example. We can quickly see that 101 is prime by checking all primes no greater than $\sqrt{101}$ for whether they divide 101. Since 101 is not divisible by $2, 3, 5$ or 7 it follows that 101 is prime.

1.7.9. Example. The ***Sieve of Eratosthenes*** finds all primes less than a given n by eliminating (sifting) all multiples of primes less than $\sqrt{n}$. We illustrate this for $n = 120$.

The Sieve of Eratosthenes



Prime numbers

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		

Image source: http://commons.wikimedia.org/wiki/File:New_Animation_Sieve_of_Eratosthenes.gif
Used under the license given there

Euclid's Proof of the Infinity of the Number of Primes

1.7.10. Theorem. There are infinitely many primes.

Proof.

Assume that there are only a finite number of primes, which we list as $p_1, \ldots, p_n$. Set

$$Q = p_1 \cdot p_2 \cdots p_n + 1.$$

By the Fundamental Theorem of Arithmetic, Q is either prime or a product of primes. If Q is prime, we have found a prime not in the list $p_1, \ldots, p_n$. If Q is not prime, then it has a prime factor, say p_k. Then $p_k \mid Q$ and $p_k \mid p_1 \cdot p_2 \cdots p_n$, so

$$p_k \mid \underbrace{\left(Q - p_1 \cdot p_2 \cdots p_n\right)}_{=1},$$

which is impossible. Therefore, there must be a prime not in the list. □

Euclid's Proof of the Infinity of the Number of Primes

Note that the proof does ***not*** state that $P_n = p_1 \cdot p_2 \cdots p_n + 1$ must be a prime. However, it is interesting to note that it often seems to be the case:

- $P_1 = 2 + 1 = 3$,
- $P_2 = 2 \cdot 3 + 1 = 7$,
- $P_3 = 2 \cdot 3 \cdot 5 + 1 = 31$,
- $P_4 = 2 \cdot 3 \cdot 5 \cdot 7 + 1 = 211$,
- $P_5 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 + 1 = 2311$,
- $P_6 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 + 1 = 59 \cdot 509$ (not prime),
- $P_7 = 19 \cdot 97 \cdot 277$ (not prime), etc.

It is not known whether there are infinitely many n for which P_n is prime.

Conjectures on Prime Numbers

Prime numbers and their distribution among the integers have always fascinated mathematicians. There are many conjectures that appear simple, but have not yet been proven, or only partially proven, for example:

- The ***Twin Prime Conjecture*** (antiquity): There are infinitely many primes p such that $p + 2$ is also prime. Such primes are called ***twin primes***.

  The first hard result was obtained in May 2013, by Zhang Yitang, a Chinese-born Lecturer at the University of New Hampshire.

  Zhang proved that there exists a number $k < 70,000,000$ such that there are infinitely many prime pairs of the form $p, p + k$.

  Subsequently, his methods were improved by many mathematicians to obtain the existence of a $k \leq 246$. More details can be found, e.g., at

  http://www.wired.com/2013/11/prime/

Conjectures on Prime Numbers

- The ***Goldbach Conjecture*** (1742): Every even number can be written as the sum of two prime numbers.

  (It has been shown in 1995 that every even number is the sum of no more than 6 primes.)

- There are infinitely many primes of the form $p = n^2 + 1$, $n \in \mathbb{N}$.

  (Remains unproven.)

- There are infinitely many primes of the form $p = 4n + 3$, $n \in \mathbb{N}$.

  (See assignments.)

- For fixed $a, b \in \mathbb{N}$ with $\gcd(a, b) = 1$, there are infinitely many primes of the form $p = an + b$, $n \in \mathbb{N}$.

  (This is Dirichlet's Theorem, established in 1837.)

Congruency

1.8.1. Definition. For $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$ we say that ***a is congruent to b modulo m***, writing

$$a \equiv b \pmod{m} \qquad \text{if and only if} \qquad m \mid (a - b).$$

1.8.2. Lemma. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. Then

$$a \equiv b \pmod{m} \quad \Leftrightarrow \quad \exists_{k \in \mathbb{Z}} \; a = b + km.$$

Proof.

$$a \equiv b \pmod{m} \quad \Leftrightarrow \quad m \mid (a - b) \quad \Leftrightarrow \quad \exists_{k \in \mathbb{Z}} \; a - b = km$$

$$\Leftrightarrow \quad \exists_{k \in \mathbb{Z}} \; a = b + km$$

□

Congruency

Given $m \in \mathbb{N} \setminus \{0\}$, by the division algorithm, for every $a \in \mathbb{Z}$ there exist unique q, r such that

$$a = q \cdot m + r, \qquad 0 \leq r < m.$$

We write

$$q =: a \operatorname{div} m, \qquad r =: a \bmod m.$$

for the quotient q and the remainder r modulo m.

Lemma 1.8.2 can then be expressed as follows:

1.8.3. Theorem. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. Then

$$a \equiv b \pmod{m} \qquad \Leftrightarrow \qquad a \bmod m = b \bmod m$$

Hence, two numbers are congruent modulo m if and only if they have the same remainder modulo m.

Congruence Classes

The congruence $\pmod{m}$ is an equivalence relation on $\mathbb{Z}$:

$$a \sim b \qquad \Leftrightarrow \qquad a \equiv b \pmod{m} \tag{1.8.1}$$

is reflexive, symmetric and transitive.

By Theorem 1.4.7 every equivalence relation induces a partition of the underlying set.

1.8.4. Definition. The partition on the integers induced by the congruency relation of Definition 1.8.1 is denoted by $\mathbb{Z}_m$. The fibers (equivalence classes) of this partition are called ***congruence classes***. We write

$$\mathbb{Z}_m = \{[0]_m, [1]_m, \ldots, [m-1]_m\}$$

Congruence Classes

Hence, a single equivalence class $[a]_m$ contains all integers that have the same remainder when divided by m.

1.8.5. Example. For $m = 2$ the relation (1.8.1) induces the partition

$$\mathbb{Z}_2 = \{[0]_2, [1]_2\}$$

where the two classes contain the even and odd numbers, respectively. See also Examples 1.4.5 and 1.6.4.

1.8.6. Definition and Theorem. The partition $\mathbb{Z}_m$ together with addition and multiplication defined by

$$[a]_m + [b]_m := [a + b]_m, \qquad [a]_m \cdot [b]_m := [a \cdot b]_m$$

defines a commutative ring. We usually drop the equivalence class notation and simply write

$$\mathbb{Z}_m = \{0, 1, \dots, m - 1\}$$

Congruence Classes

The well-definedness of addition and multiplication follows from the following lemma:

1.8.7. Lemma. Let $a, \widetilde{a}, c, d \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. If $a \equiv \widetilde{a} \pmod{m}$ and $b \equiv \widetilde{b} \pmod{m}$ then

$$a + b \equiv \widetilde{a} + \widetilde{b} \pmod{m} \qquad \text{and} \qquad ab \equiv \widetilde{a}\widetilde{b} \pmod{m}.$$

Proof.
By Lemma 1.8.2 there exist $s, t \in \mathbb{Z}$ such that $\widetilde{a} = a + sm$, $\widetilde{b} = b + tm$. Then

$$\widetilde{a} + \widetilde{b} = a + b + m(s + t) \qquad \Leftrightarrow \qquad a + b \equiv \widetilde{a} + \widetilde{b} \pmod{m}$$

and

$$\widetilde{a}\widetilde{b} = ab + m(at + bs + stm) \qquad \Leftrightarrow \qquad ab \equiv \widetilde{a}\widetilde{b} \pmod{m}. \qquad \square$$

Modular Arithmetic

The ring properties of $\mathbb{Z}_p$ are left for you to prove.

As a corollary of Lemma 1.8.7 we obtain

1.8.8. Corollary. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. Then

$$a + b \equiv \big(a \bmod m + b \bmod m\big) \pmod{m}$$
$$ab \equiv \big((a \bmod m)(b \bmod m)\big) \pmod{m}.$$

Hence, to find the product (sum) of two numbers modulo m, it suffices to multiply (add) their remainders (modulo m). This is the basis of ***modular arithmetic***, a very fruitful field of number theory.

Modular Arithmetic

1.8.9. Example. We prove that $41 \mid 2^{20} - 1$. This is equivalent to showing that

$$2^{20} - 1 \equiv 0 \pmod{41}.$$

We note that $2^5 = 32 \equiv -9 \pmod{41}$. Then Corollary 1.8.8 implies

$$2^{20} = (2^5)^4 \equiv (-9)^4 \pmod{41}$$

But $(-9)^4 = 81 \cdot 81$ and $81 \equiv -1 \pmod{41}$. Again by Corollary 1.8.8,

$$2^{20} \equiv (-1)^2 \equiv 1 \pmod{41}$$

from which the assertion follows.

Modular Arithmetic

1.8.10. Example. We calculate the remainder

$$(1! + 2! + 3! + 4! + \cdots + 100!) \bmod 12.$$

Since $4! = 2 \cdot 12$, we have $12 \mid n!$ for all $n \geq 4$. This implies

$$\begin{aligned}
&(1! + 2! + 3! + 4! + \cdots + 100!) \bmod 12 \\
&= 1! \bmod 12 + 2! \bmod 12 + 3! \bmod 12 + \underbrace{4! \bmod 12}_{=0} + \cdots + \underbrace{100! \bmod 12}_{=0} \\
&= 1 + 2 + 6 = 9.
\end{aligned}$$

Division in Modular Arithmetic

We have seen that for any integers $a, b, c \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$

$$a \equiv b \pmod{m}$$

implies

$$ac \equiv bc \pmod{m}.$$

However, the converse is not necessarily true:

$$2 \cdot 1 \equiv 2 \cdot 4 \pmod{6} \qquad \text{but} \qquad 1 \not\equiv 4 \pmod{6}.$$

The next theorem gives the correct way to divide in modular arithmetic.

Division in Modular Arithmetic

1.8.11. Theorem. Let $a, b, c \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. Then

$$ac \equiv bc \pmod{m}.$$

implies

$$a \equiv b \pmod{m/d}$$

where $d = \gcd(c, m)$.

Proof.
We have

$$ac - bc = k \cdot m \qquad \text{for some } k \in \mathbb{Z}.$$

There exist integers r, s with $\gcd(r, s) = 1$ such that $c = rd$, $m = sd$.
Inserting into the equation,

$$ard - brd = k \cdot sd$$

Division in Modular Arithmetic

Proof (continued).
Canceling the common factor d,

$$r(a-b) = ks.$$

so $s \mid r(a-b)$. Since $\gcd(r,s) = 1$, Euclid's Lemma gives $s \mid (a-b)$. Hence,

$$a \equiv b \mod s.$$

Since $s = m/d$, we are finished. □

1.8.12. Corollary. Let $a, b, c \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$ and $\gcd(c,m) = 1$. Then

$$ac \equiv bc \pmod{m}.$$

implies

$$a \equiv b \pmod{m}$$

Modular Inverse

We now discuss the existence of multiplicative inverses.

1.8.13. Definition. Let $a \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0, 1\}$ be given. Then an integer $a^{-1} \in \mathbb{Z}$ such that

$$a^{-1}a \equiv 1 \pmod{m}.$$

is said to be an ***inverse of a modulo m***.

1.8.14. Theorem. Let $a \in \mathbb{N} \setminus \{0\}$ and $m \in \mathbb{N} \setminus \{0, 1\}$. If $\gcd(a, m) = 1$, an inverse of a modulo m exists. This inverse is unique modulo m.

Proof.
By Bézout's Lemma 1.6.10 there exist $s, t \in \mathbb{Z}$ such that $sa + tm = 1$. This means, in particular, that

$$sa \equiv 1 \pmod{m} \tag{1.8.2}$$

and s is the inverse of a modulo m. Hence, an inverse of a exists.

Modular Inverse

Proof (continued).

Now suppose $\tilde{a}^{-1}$ and a^{-1} are two modular inverses of a. Then

$$a\tilde{a}^{-1} \equiv 1 \equiv aa^{-1} \pmod{m}.$$

so

$$a(\tilde{a}^{-1} - a^{-1}) \equiv 0 \pmod{m}.$$

Since $\gcd(a, m) = 1$, Corollary 1.8.12 allows us to divide by a to obtain

$$a^{-1} \equiv \tilde{a}^{-1} \pmod{m}.$$ □

1.8.15. Remark. The proof of Theorem 1.8.14 also yields a method for finding an inverse of a modulo m: find (by using the euclidean algorithm) a linear combination of a and m that equals 1. The coefficient of a will be an inverse of a modulo m.

Congruence Classes as Fields

1.8.16. Theorem. The partition $\mathbb{Z}_p$ is a field if and only if p is a prime number.

Proof.

($\Leftarrow$) By Theorem 1.8.6, $\mathbb{Z}_p$ is a commutative ring with neural elements 0 for addition and 1 for multiplication. By the Definition 1.4.10 of a field, it remains to show that a multiplicative inverse exists for any $a \in \mathbb{Z}_p$, $a \neq 0$.

Since p is prime, any $a = 1, \ldots, p-1$ is relatively prime to p and by Theorem 1.8.14 has a multiplicative inverse.

Congruence Classes as Fields

Proof (continued).

($\Rightarrow$) Suppose that p is not prime. Then there exist $a, b \in \mathbb{Z}$ with $1 < a, b < p$ such that $p = a \cdot b$. If $\mathbb{Z}_p$ is a field, then by definition a must have a multiplicative inverse a^{-1} such that $a \cdot a^{-1} = 1$. Thus we can write:

$$b \cdot 1 = b \cdot (a \cdot a^{-1}) = (b \cdot a) \cdot a^{-1} = p \cdot a^{-1} \equiv 0 \pmod{p}$$

This contradicts $b \notin \{0, p\}$. Hence, a does not have a multiplicative inverse and $\mathbb{Z}_p$ is not a field. $\square$

Linear Congruences

1.8.17. Definition. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$. A ***linear congruence*** is an equation

$$ax \equiv b \pmod{m} \tag{1.8.3}$$

for unknowns $x \in \mathbb{Z}$.

We are interested in finding all $x \in \mathbb{Z}$ that satisfy (1.8.3), up to congruency modulo m.

Note that (1.8.3) can be rephrased as

$$ax - my = b, \tag{1.8.4}$$

which is simply the linear diophantine equation (1.6.4). Hence, we can base our theory of solvability on known results for the diophantine equation.

Solvability of Linear Congruences

1.8.18. Theorem. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$ and $d = \gcd(a, m)$. The linear congruence $ax \equiv b \pmod{m}$ has a solution if and only if $d \mid b$. In that case, it has d solutions that are mutually incongruent modulo m.

Proof.

We already know from Theorem 1.6.26 that (1.8.4) has a solution if and only if $d \mid m$, so the same is true of (1.8.3).

If a solution (x_0, y_0) exists, all other solutions of (1.8.4) are given by

$$x = x_0 + \frac{m}{d}t, \qquad y = y_0 + \frac{m}{d}t, \qquad t \in \mathbb{Z}.$$

Suppose $t = 0, \ldots, d-1$. We claim that these solutions are mutually incongruent, while all other solutions are congruent modulo m.

Solvability of Linear Congruences

Proof (continued).

Suppose first that

$$x_0 + \frac{m}{d}t_1 \equiv x_0 + \frac{m}{d}t_2 \pmod{m}, \qquad 0 \leq t_1 < t_2 \leq d-1.$$

Then

$$\frac{m}{d}t_1 \equiv \frac{m}{d}t_2 \pmod{m},$$

Since $\gcd(m/d, m) = m/d$ by Theorem 1.8.11 we can cancel the factor m/d to obtain

$$t_1 \equiv t_2 \pmod{d}.$$

But this contradicts $0 \leq t_1 < t_2 \leq d-1$. Hence, the solutions for $t = 0, \ldots, d-1$ are mutually incongruent.

Solvability of Linear Congruences

Proof (continued).

Finally, we show that any other solution is congruent to one of the d solutions with $t = 0, \dots, d-1$. Suppose an arbitrary $t \in \mathbb{Z}$ is given. By the Division Algorithm, there exist $q, r \in \mathbb{Z}$ such that

$$t = qd + r, \qquad 0 \leq r \leq d-1.$$

Then

$$\begin{aligned} x_0 + \frac{m}{d}t &= x_0 + \frac{m}{d}(qd + r) \\ &= x_0 + mq + \frac{m}{d}r \\ &\equiv x_0 + \frac{m}{d}r \pmod{m}, \end{aligned}$$

so the solution is congruent to some solution with $t = 0, \dots, d-1$. □

Solving Linear Congruences

1.8.19. Example. We solve the linear congruence

$$18x \equiv 30 \pmod{42}.$$

Since $\gcd(18, 42) = 6$ and $6 \mid 30$ there are exactly 6 mutually incongruent solutions. We guess a particular solution $x = 4$. Then the other incongruent solutions are given by

$$x = 4 + \frac{42}{6}t = 4 + 7t, \qquad t = 0, \ldots, 5.$$

or

$$x_0 = 4, \quad x_1 = 11, \quad x_2 = 18, \quad x_3 = 25, \quad x_4 = 32, \quad x_5 = 39$$

But how do we find this first solution $x = 4$ systematically, without resorting to guessing?

Solving Linear Congruences

We use the following consequence of Theorem 1.8.18,

1.8.20. Corollary. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$ and $\gcd(a, m) = 1$. Then the linear congruence $ax \equiv b \pmod{m}$ has a unique solution modulo m.

1.8.21. Example. We first attempt to reduce the problem

$$18x \equiv 30 \pmod{42}.$$

to one which has a unique solution. We divide by $\gcd(18, 42) = 6$, giving

$$3x \equiv 5 \pmod{7}.$$

Again, at this point we might guess (more easily than before) $x = 4$ and find the other solutions

$$x_0 = 4, \quad x_1 = 11, \quad x_2 = 18, \quad x_3 = 25, \quad x_4 = 32, \quad x_5 = 39$$

which are all congruent modulo 7 but incongruent modulo 42.

Solving Congruencies using the Modular Inverse

We can find a particular solution of a congruency $ax \equiv b \pmod{m}$ by multiplying with an inverse of a modulo m. By Theorem 1.8.14, this inverse exists if $\gcd(a, m) = 1$.

1.8.22. Example. We want to solve the congruence $3x \equiv 5 \pmod{7}$. We first find an inverse of 3 mod 7. Applying the euclidean algorithm,

$$\begin{aligned} 3 &= 1 \cdot 7 - 4, \\ 7 &= (-1) \cdot (-4) + 3, \\ -4 &= (-1) \cdot 3 - 1, \\ 3 &= (-3) \cdot 1 + 0, \end{aligned}$$

so $\gcd(3, 7) = 1$, as expected and

$$\begin{aligned} 1 &= (-1) \cdot 3 + 4 \\ &= (-1) \cdot 3 + (7 - 3) \\ &= (-2) \cdot 3 + 7. \end{aligned}$$

It follows that an inverse of $a = 3$ modular 7 is $a^{-1} = -2$.

Solving Congruencies using the Modular Inverse

We then multiply the congruency

$$3x \equiv 5 \pmod 7$$

by $a^{-1} = -2$, obtaining

$$-6x \equiv -10 \pmod 7$$

Since $-6 \equiv 1 \pmod 7$ and $-10 \equiv 4 \pmod 7$, we obtain

$$1 \cdot x \equiv 4 \pmod 7,$$

as expected.

Sunzi's Problem

The Chinese Remainder theorem is based on a problem posed by the Chinese mathematician Sunzi (孫子 / 孙子) and published in his book "孫子算經" (perhaps best translated as "Master Sun's Mathematical Manual") in the first or third century AD (dates by: Rosen's book and wikipedia, respectively.)

Sunzi asks:

> *There are certain things whose number is unknown. When divided by 3, the remainder is 2; when divided by 5, the remainder is 3; and when divided by 7, the remainder is 2. What will be the number of things?*

今有物不知其数，三三数之剩二，五五数之剩三，
七七数之剩二，问物几何?

The Chinese Remainder Theorem

Sunzi's problem can be written in modern mathematical notation as a system of congruences,

$$\begin{aligned} x &\equiv 2 \pmod 3, \\ x &\equiv 3 \pmod 5, \\ x &\equiv 2 \pmod 7. \end{aligned}$$

1.8.23. Chinese Remainder Theorem. Let $m_1, \ldots, m_n \in \mathbb{N} \setminus \{0\}$ be pairwise relatively prime and $a_1, \ldots, a_n \in \mathbb{Z}$. Then the system of congruences

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \\ &\vdots \\ x &\equiv a_n \pmod{m_n}. \end{aligned} \tag{1.8.5}$$

has a unique solution modulo $m = m_1 m_2 \ldots m_n$.

The Chinese Remainder Theorem

Proof.

We first show the existence of a solution by demonstrating how to construct it. Let

$$M_k := \frac{m}{m_k} = \prod_{i \neq k} m_i.$$

Then $\gcd(m_k, M_k) = 1$ and by Theorem 1.8.14 there exists an inverse y_k to M_k modulo m_k, i.e.,

$$M_k y_k \equiv 1 \pmod{m_k}.$$

Then it is easily verified that

$$x = \sum_{k=1}^{n} a_k M_k y_k$$

is a solution to (1.8.5) (noting that $M_j \equiv 0 \pmod{m_i}$ for $i \neq j$).

The Chinese Remainder Theorem

Proof (continued).

Now suppose there are two solutions x, x' to the system of congruences. Then for $k = 1, \ldots, n$,

$$x \equiv a_k \equiv x' \pmod{m_k}.$$

This implies

$$m_k \mid (x - x') \qquad\qquad k = 1, \ldots, n.$$

Since $\gcd(m_k, m_j) = 1$ for all $j, k = 1, \ldots, n$, Corollary 1.6.16 allows us to deduce

$$\underbrace{m_1 \cdots m_n}_{=m} \mid (x - x')$$

which proves uniqueness modulo m. □

Solution to Sunzi's Problem

To find the answer to Sunzi's original question, set $m = m_1 m_2 m_3 = 3 \cdot 5 \cdot 7 = 105$. Then

$$M_1 = m/3 = 35, \qquad M_2 = m/5 = 21, \qquad M_3 = m/7 = 15.$$

We then find the inverse of M_k modulo m_k,

$$y_1 = 2, \qquad y_2 = 1, \qquad y_3 = 1.$$

The solution to the system is then

$$x = 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 = 233 \equiv 23 \pmod{105}.$$

Thus 23 is the smallest integer that satisfies Sunzi's problem, but the solution is only unique modulo 105.

Solution of a System in an Elementary Fashion

1.8.24. Example. We solve the congruency

$$17x \equiv 9 \pmod{276}.$$

Instead of solving it directly, we note that $276 = 3 \cdot 4 \cdot 23$, so the congruency is equivalent to the system

$$17x \equiv 9 \pmod{3}, \quad 17x \equiv 9 \pmod{4}, \quad 17x \equiv 9 \pmod{23}.$$

(Why?) We can simplify this system to

$$x \equiv 0 \pmod{3}, \quad x \equiv 1 \pmod{4}, \quad 17x \equiv 9 \pmod{23}.$$

(Why?)

The first congruence gives $x = 3k$, $k \in \mathbb{Z}$. Substituting into the second congruence,

$$3k \equiv 1 \pmod{4}$$

The modular inverse of $a = 3$ is $\widetilde{a} = 3$, so we obtain

$$k \equiv 3 \pmod{4}$$

Solution of a System in an Elementary Fashion

We then have

$$x = 3 \cdot (3 + 4j) = 9 + 12j, \qquad j \in \mathbb{Z}.$$

Inserting into the last congruence,

$$17 \cdot (9 + 12j) \equiv 9 \pmod{23}$$

or

$$204j \equiv -144 \pmod{23}.$$

This reduces to $3j \equiv 6 \pmod{23}$. Hence, $j = 2 + 23t$, $t \in \mathbb{Z}$ and hence

$$x = 33 + 276t$$

or simply $x \equiv 33 \pmod{276}$.

Fermat's Factorization Method

One of the most important problems in number theory is the finding of factors of a composite number, or of determining that a number is prime.

The basic algorithm from antiquity is the Sieve of Eratosthenes, which is based on Theorem 1.7.7:

> *To find factors (if any exist) of a number n, one tests all primes less than $\sqrt{n}$ to see if they divide n.*

Eratosthenes lived around 200 B.C. - it took until 1643, 1800 years later, for a significant improvement on this algorithm to be found. In that year, Pierre de Fermat formulated the following idea in a letter to Mersenne:

> *Finding two factors of an odd number n is equivalent to solving the quadratic, two-variable Diophantine equation $n = x^2 - y^2$.*

Fermat's Factorization Method

This is easy to see, because if x and y satisfy $n = x^2 - y^2$, then

$$n = x^2 - y^2 = (x + y)(x - y)$$

and $x \pm y$ are factors of n. On the other hand, if $n = ab$ and n is odd, then

$$n = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2$$

and $(a \pm b)/2$ are solutions to the Diophantine equation. (Note that if n is odd, then so are a and b, so $a \pm b$ is even.)

Therefore, finding a and b such that $n = ab$ is accomplished by finding x and y such that

$$x^2 - n = y^2$$

Fermat's Factorization Method

We proceed as follows:

- Find the smallest k such that $k^2 > n$.
- Consider successively the numbers

$$k^2 - n, \qquad (k+1)^2 - n, \qquad (k+2)^2 - n, \ldots$$

  until one of these numbers is a square.
- The process must terminate, since

$$\left(\frac{n+1}{2}\right)^2 - n = \left(\frac{n-1}{2}\right)^2$$

  which corresponds to the trivial factorization $n = 1 \cdot n$.

Fermat's Factorization Method

1.9.1. Example. We apply the method to the number $n = 119143$. By any available means, we find

$$345^2 < 119143 < 346^2.$$

We need to calculate $k^2 - 119143$ for

$$346 < k < \frac{119143 + 1}{2} = 59572.$$

We find:

$$346^2 - 119143 = 573, \qquad 347^2 - 119143 = 1266,$$
$$348^2 - 119143 = 1961, \qquad 349^2 - 119143 = 2658,$$
$$350^2 - 119143 = 3357, \qquad 351^2 - 119143 = 4058,$$
$$352^2 - 119143 = 4761 = 69^2.$$

Hence, $119143 = (352 + 69)(352 - 69) = 421 \cdot 283$.

Finding Squares

For using this method, is is useful to note that

- Square numbers must end in one of the digits $0, 1, 4, 5, 6, 9$.
- By calculating k^2 mod 100 for $0 \leq k \leq 99$, the last two digits must be one of the following:

0	1	4	9	16
21	24	25	29	36
41	44	49	56	61
64	69	76	81	84
89	96			

In the previous example, this means we need only actually test the numbers 1961 and 4761 for being squares.

Fermat's Little Theorem

Fermat wrote extensively on prime numbers; one of his most important results is called his "little" theorem, to distinguish it from his "last" theorem, that for $n \geq 3$ the equation $z^n + y^n = z^n$ has no integer solutions x, y, z.

1.9.2. Fermat's Little Theorem. Let $p \in \mathbb{N}$ and $a \in \mathbb{N}$. If p is prime and $p \nmid a$, then

$$a^{p-1} \equiv 1 \pmod{p}. \tag{1.9.1}$$

More generally, for any prime $p \in \mathbb{N}$ and $a \in \mathbb{Z}$,

$$a^p \equiv a \pmod{p}. \tag{1.9.2}$$

Fermat's Little Theorem

Proof.

Consider the $p-1$ numbers

$$a, \quad 2a, \quad 3a, \quad \ldots, \quad (p-1)a \tag{1.9.3}$$

Since $\gcd(p, a) = 1$, these numbers are incongruent modulo p:

$$ra \equiv sa \pmod{p}$$

implies $r \equiv s \pmod{p}$ for any $r, s \in \mathbb{Z}$. Therefore, the integers (1.9.3) must each be congruent to one of the numbers $1, 2, 3, \ldots, p-1$, in some order. Hence,

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv (p-1)! \pmod{p}.$$

Canceling $(p-1)!$ on both sides (why can we do this?), we obtain (1.9.1).

Fermat's Little Theorem

Proof (continued).
The statement (1.9.2) is trivially true if $p \mid a$: then $a \equiv 0 \equiv a^p \pmod{p}$. If $p \nmid a$, we simply multiply (1.9.1) by a to obtain (1.9.2). □

1.9.3. Example. We calculate $5^{38} \bmod 11$. By Fermat's Little Theorem,

$$5^{10} = 5^{11-1} \equiv 1 \pmod{11}.$$

Then

$$\begin{aligned} 5^{38} &= 5^{10\cdot 3+8} = (5^{10})^3(5^2)^4 \\ &\equiv 1^3 \cdot 3^4 \equiv 81 \equiv 4 \pmod{11}. \end{aligned}$$

Fermat's Little Theorem

Another application is to show that a number n is not prime: if for some $a \in \mathbb{N} \setminus \{0\}$

$$a^n \not\equiv a \pmod{n},$$

then n can not be prime.

1.9.4. Example. Consider $n = 117$. Then

$$\begin{aligned} 2^{117} &= (2^7)^{16}2^5 \\ &= 128^{16}2^5 \\ &\equiv 11^{16}2^5 \equiv 121^8 2^5 \equiv 4^8 2^5 \pmod{117} \\ &\equiv 2^{21} \equiv (2^7)^3 \equiv 11^3 \equiv 4 \cdot 11 \equiv 44 \pmod{117} \\ &\not\equiv 2 \pmod{117}, \end{aligned}$$

so 117 is not prime. In fact, $117 = 13 \cdot 9$.

Fermat Pseudoprimes

We know that if p is prime, then

$$a^{p-1} \equiv 1 \pmod{p}$$

for any a with $p \nmid a$. But is the converse true? If the congruence holds, must p be prime?

We claim that this is not the case. In fact, we will show that

$$2^{341-1} \equiv 1 \pmod{341}$$

but $341 = 11 \cdot 31$ is not prime.

In fact, composite numbers for which $a^{p-1} \equiv 1 \pmod{p}$ are called ***Fermat pseudoprimes to base*** a.

Fermat Pseudoprimes

To verify our assertion that 341 is a Fermat pseudoprime to base 2, we need a technical result:

1.9.5. Lemma. Let $p, q \in \mathbb{N} \setminus \{0\}$ be prime numbers such that

$$a^p \equiv a \pmod{q} \qquad \text{and} \qquad a^q \equiv a \pmod{p}$$

then

$$a^{pq} \equiv a \pmod{pq}.$$

Proof.
By (1.9.2),

$$(a^q)^p \equiv a^q \pmod{p}.$$

By assumption, $a^q \equiv a \pmod{p}$. Thus,

$$a^{pq} \equiv a \pmod{p} \qquad \text{or} \qquad p \mid (a^{pq} - a).$$

Similarly, $q \mid (a^{pq} - a)$, so $pq \mid (a^{pq} - a)$ since $\gcd(p, q) = 1$. □

Fermat Pseudoprimes

We now show that $2^{340} \equiv 1 \pmod{341}$ where $341 = 11 \cdot 31$. First, note that

$$2^{10} = 1024 = 31 \cdot 33 + 1$$

so

$$2^{11} = 2 \cdot 2^{10} \equiv 2 \cdot 1 \equiv 2 \pmod{31}$$

and

$$2^{31} = 2(2^{10})^3 \equiv 2 \cdot 1^3 \equiv 2 \pmod{11}.$$

By Lemma 1.9.5,

$$2^{341} = 2^{11 \cdot 31} \equiv 2 \pmod{341}.$$

Cancelling the factor 2,

$$2^{340} \equiv 1 \pmod{341}.$$

Wilson's Theorem

Finally, we give another result on prime numbers, dating from the late 18^{th} century.

1.9.6. Wilson's Theorem. Let $p \in \mathbb{N}$ be prime. Then

$$(p-1)! \equiv -1 \pmod{p}.$$

Proof.

The cases $p = 2$ and $p = 3$ are obvious, so we suppose $p > 3$. Let a be any of the numbers

$$1, 2, 3, \ldots, p-1.$$

Then, by Theorem 1.8.14, the inverse a^{-1} modulo p of a exists and is unique modulo p,

$$a^{-1}a \equiv 1 \pmod{p}.$$

We first show that $a = a^{-1}$ if and only if $a = 1$ or $a = p-1$.

Wilson's Theorem

Proof (continued).

It is easily checked that

$$1 \cdot 1 \equiv 1 \pmod{p} \qquad \text{and} \qquad (p-1)^2 \equiv 1 \pmod{p}.$$

Now suppose that $a^2 \equiv 1 \pmod{p}$. Then

$$(a-1)(a+1) \equiv 0 \pmod{p}$$

Since p is prime, this implies $a - 1 \equiv 0$ or $a + 1 \equiv 0 \pmod{p}$. Hence, either $a = 1$ or $a = p - 1$.

Wilson's Theorem

Proof (continued).

Next, consider the remaining $p-3$ integers

$$2, 3, \ldots, p-2.$$

Since the inverse a^{-1} of a is unique, it follows that these numbers can be grouped into pairs a, a^{-1} where $a^{-1} \neq a$ and $a^{-1}a \equiv 1 \pmod{p}$.

Therefore,

$$(p-2)! = 2 \cdot 3 \cdots (p-2) \equiv 1 \pmod{p}.$$

Multiplying by $p-1$,

$$(p-1)! \equiv p-1 \equiv -1 \pmod{p}$$

which is what we wanted to show. □

Wilson's Theorem

Wilson's Theorem is mostly of theoretical interest, i.e., it can be ued to prove other results. As a practical method for determining whether a number is prime it is only of limited usefulness, since $n!$ is too large to calculate efficiently.

A particular consequence is that Wilson's theorem implies that there are an infinite number of composite numbers of the form $n! + 1$. (Since there are an infinite number of primes.) It is not known whether there are also an infinite number of primes of that form. However, it appears much rarer for $n! + 1$ to be prime than for it to be composite.

First Midterm Exam

This concludes the material that will be covered in the first midterm exam. The material is based on the following textbook chapters:

- Chapter 1;
- Chapter 2;
- Chapter 4, Sections 1 and 2;
- Chapter 3, Sections 4 – 7;
- Chapter 8, Section 5.

Some of the material (such as certain parts of the number theory and algebraic structures) may not appear in this form in the textbook; it is nevertheless part of the exam. The above textbook chapters are for reference only. The lecture slides and assignments represent the definitive exam material.

For the exam, you **may not** bring a calculator or any other electronic device, such as a cellphone or electronic dictionary. A monolingual dictionary in the form of a paper book is allowed.

Part II

Algorithms

Algorithms and Computational Complexity

Computer Arithmetic

Recurrence Relations and Divide-and-Conquer Algorithms

Combinatorics

Probabilistic Algorithms

Cryptographic Algorithms

Algorithms and Computational Complexity

Algorithms

In the previous part we have encountered many different "methods" that can be used for performing calculations, such as the Division algorithm, the Euclidean algorithm, Fermat's factorization, etc.

In the present part, we will discuss how these abstract methods can be transformed into concrete ***algorithms***, suitable for implementation on a computer. In doing so, we will study algorithms in more depth, focusing particularly on the time (number of individual operations) that an algorithm needs to perform its task.

Let us first give an informal definition:

2.1.1. Definition. An ***algorithm*** is a finite sequence of precise instructions for performing a computation or for solving a problem.

A more formal definition follows.

Definition of an Algorithms

An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function.

*Starting from an **initial state and initial input** (perhaps empty), the instructions describe a computation that, when executed, proceeds through a **finite** number of **well-defined** successive states, eventually producing **output** and terminating at a final ending state.*

The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.

Wikipedia contributors, "Algorithm," ***Wikipedia, The Free Encyclopedia***, http://en.wikipedia.org/w/index.php?title=Algorithm&oldid=629026201 (accessed October 10, 2014).

Example: Finding the Maximum

2.1.2. Example. One possible algorithm for finding the maximum of a finite sequence of integers is as follows:

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers in the sequence.
4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

Pseudocode

The description of the algorithm in Example 2.1.2 is quite cumbersome. In fact, most algorithms are intended to be realized as programs on computers. There are several programming languages that can implement such algorithms, such as Pascal or C++. It is assumed that you have at least a passing familiarity with such programming languages.

Instead of using a specific language, we will use express algorithms using ***pseudocode***.

The advantage of pseudocode is that it is as succinct as a computer program but general enough to be universally understood.

Pseudocode for Finding the Maximum

The algorithm of Example 2.1.2 can be expressed in pseudocode as follows:

2.1.3. Algorithm. ***(Maximum Element)***

Input: n integers, $a_1, a_2, \ldots, a_n$
Output: $\max(a_1, \ldots, a_n)$

```
1 max ← a_1;
2 for i ← 2 to n do
3 |   if max < a_i then
4 |   |   max ← a_i;
5 |   end if
6 end for
7 return max
```

Properties of Algorithms

There are several properties that algorithms usually share. These include:

- ***Input.*** An algorithm has input values from a specified set;
- ***Output.*** For given input, the algorithm produces output values from a specified set;
- ***Definiteness.*** The steps of the algorithm are defined precisely;
- ***Correctness.*** For each input, the algorithm produces the correct output values;
- ***Finiteness.*** For given input, the algorithm produces output after a finite number of steps;
- ***Effectiveness.*** Each step of the algorithm can be performed exactly;
- ***Generality.*** The algorithm is generally applicable, not just for certain input values.

Properties of Algorithms

2.1.4. Example. Algorithm 2.1.3 has all of these properties:

- ***Input.*** The input is a finite sequence of integers;
- ***Output.*** The output is the largest integer of the set;
- ***Definiteness.*** Only assignments, a finite loop and conditional statements occur; these are well-defined;
- ***Finiteness.*** The algorithm uses a finite number of steps because it terminates after all integers have been examined;
- ***Effectiveness.*** Each step of the algorithm can be performed exactly;
- ***Generality.*** The algorithm is generally applicable, because it can be used to find the maximum of any finite sequence of numbers;
- ***Correctness.*** The algorithm is correct, because it does return the maximum of the integers in the sequence. The proof of correctness is based on induction and requires a bit more background; we will study this in more detail and return to this example later.

Iterative and Recursive Algorithms

We can generally divide algorithms into two types:

- ***Iterative algorithms*** repeat a key step until a problem is solved.
- ***Recursive algorithms*** solve a problem by reducing it to an instance of the same problem with smaller input.

An iterative algorithm to calculate the value of $n!$, $n \in \mathbb{N} \setminus \{0\}$, is as follows:

2.1.5. Algorithm. ***(Factorial – Iterative Version)***

Input: n, a positive integer
Output: $n!$

```
1 factorial ← 1;
2 for i ← 1 to n do
3 |   factorial ← i · factorial ;
4 end for
5 return factorial
```

Recursive and Iterative Algorithms

A recursive algorithm to calculate the value of $n!$ is as follows:

2.1.6. Algorithm. ***(Factorial – Recursive Version)***

Input: n, a positive integer
Output: $n!$

```
1 Function Factorial(n):
2     if n = 1 then
3         factorial ← 1
4     else
5         factorial ← n · Factorial(n − 1)
6     end if
7     return factorial
8 end
```

We will now study several examples of algorithms.

Search Algorithms

Search algorithms are very important in many applications. The problem is to find an element x within a tuple $(a_1, \ldots, a_n)$ of elements, if it is present.

We consider x to be found if we know which $a_k = x$, i.e., if we know the value of k. The following algorithm is quite simple, comparing each element in order with x and terminating when a match is found.

2.1.7. Algorithm. ***(Linear Search)***

Input: x an integer and $a_1, a_2, \ldots, a_n$, n distinct integers
Output: i such that $a_i = x$, or 0 if no such i exists

```
1 i ← 1;
2 while i ≤ n and x ≠ a_i do
3 |   i ← i + 1;
4 end while
5 if i ≤ n then location ← i ;
6 else location ← 0;
```

Binary Search

If the tuple $(a_1, \ldots, a_n)$ consists of objects in ascending order (e.g., numbers sorted from smallest to largest) then another strategy is to successively split the list in half and only search with the sublists whose elements might contain x.

2.1.8. Example. To search for the number 7 in the list $(2, 3, 7, 8, 10, 11, 13, 16)$ we first split the list into two smaller lists, $(2, 3, 7, 8)$ and $(10, 11, 13, 16)$. Since $7 \not> 8$ we do not search the second list. We next split the first list in two, $(2, 3)$ and $(7, 8)$. Since $7 > 3$, we search the second list, which we split in two, (7) and (8). Since $7 \not> 7$, we now search the first list. Since it contains only one element, we make a comparison and find 7 in this list.

If we had been searching for the number 6, we would have completed the same steps, but the final comparison would not have yielded a match.

Binary Search

2.1.9. Algorithm. ***(Binary Search)***

Input: x an integer and $a_1 < a_2 < \ldots < a_n$, n distinct ordered integers
Output: i such that $a_i = x$, or 0 if no such i exists

```
1  i ← 1;
2  j ← n ;
3  while i < j do
4  |   m ← ⌊(i + j)/2⌋;
5  |   if x > a_m then i ← m + 1;
6  |   else j ← m;
7  end while
8  if  x = a_i  then location ← i;
9  else location ← 0;
10 return location
```

Sorting Algorithms

Another very important problem is the sorting of unsorted lists (tuples). We assume here that we are sorting numbers, but the following procedures are applicable whenever we have an ordering for a set of objects (e.g., lexicographic ordering for words).

We first introduce a very simple, but not usually very fast algorithm called the ***bubble sort***. It essentially works by pairwise comparisons and transpositions.

2.1.10. Example. To put the list $(3, 2, 1, 4)$ in ascending order, we first compare the first and the second element. Since $3 > 2$, we interchange the elements, yielding $(2, 3, 1, 4)$. Then we compare the second and the third element of this list. Since $3 > 1$, we interchange and get $(2, 1, 3, 4)$. Comparing the third and the fourth elements, we obtain $3 \not> 4$, so we do not interchange any elements. This completes the first pass.

In the second pass, we again compare the first and the second elements and interchange them. The list is now in order.

Bubble Sort

2.1.11. Algorithm. *(Bubble Sort)*

Input: $a_1, \ldots, a_n$, n unsorted elements
Output: all the $a_i, 1 \leq i \leq n$ in increasing order

```
1 for i ← 1 to n − 1 do
2 |   for j ← 1 to n − i do
3 |   |   if a_j > a_{j+1} then interchange a_j and a_{j+1};
4 |   end for
5 end for
6 return (a_1, ..., a_n) in increasing order.
```

The name bubble sort is used because the largest values "bubble" to the top. The sorting procedure is illustrated by a video taken from

http://www.youtube.com/playlist?list=PLZh3kxyHrVp_AcOanN_jpuQbcMVdXbqei

More information on these videos can be found at

http://panthema.net/2013/sound-of-sorting/

Insertion Sort

Another algorithm is called ***insertion sort***. This algorithm looks at the jth element of an n element list and inserts it in the correct position in the first $j-1$ elements.

2.1.12. Algorithm. ***(Insertion Sort)***

Input: $a_1, \ldots, a_n$, n unsorted elements
Output: all the $a_i, 1 \leq i \leq n$ in increasing order

```
1 for j ← 2 to n do
2 |   i ← 1;
3 |   while a_j > a_i do i ← i + 1;
4 |   m ← a_j;
5 |   for k ← 0 to j − i − 1 do a_{j−k} ← a_{j−k−1};
6 |   a_i ← m
7 end for
8 return (a_1, ..., a_n) in increasing order
```

Optimization Problems and Greedy Algorithms

One of the main interests in the study of algorithms is to find those that solve problems in the most efficient way possible. Problems seeking an optimal solution are known as ***optimization problems***.

In general, a problem is solved in different steps. E.g., the bubble sort algorithm compares two numbers and then interchanges them or leaves them as is. This is one step of the entire algorithm. Of course, taking the (locally) optimal course at every step does not automatically leads to a (globally) optimal solution.

Nevertheless, this is often the case and such algorithms are very important. Algorithms that find an optimal solution at every step are called ***greedy algorithms***.

Optimization of Change-Making

2.1.13. Example. Consider the problem of making change for an amount of less than 1 US$, using the standard american coins: penny (1 cent), nickel (5 cents), dime (10 cents) and quarter (25 cents)

We consider the optimization problem whereby the change should be made with the minimum number of coins possible.

For example, if 27 cents change is to be made, this can be accomplished with three coins (a quarter and two pennies) 27 pennies or various other numbers of coins.

Greedy Change-Making

If n is the amount of change required, we take the largest denomination smaller than n, then consider the remaining amount. We again take the largest denomination smaller than this amount and consider the remainder, continuing until the original amount has been realized.

2.1.14. Algorithm. ***(Greedy Change-making)***

Input: $c_1 > c_2 > ... > c_r$, coin denominations and n a positive integer
Output: M a multiset of the $c_i, 1 \leq i \leq r$, with sum of all its elements being n

```
1 M ← ∅;
2 for i ← 1 to r do
3 |   while n ≥ c_i do
4 |   |   add a coin with value c_i to M;
5 |   |   n ← n − c_i;
6 |   end while
7 end for
8 return M
```

Greedy Change-Making

Algorithm 2.1.14 is greedy, since at each step it takes the greatest allowable denomination and adds it to the change. However, it is not necessarily optimal, depending on the denominations available.

For example, if only pennies, dimes and quarters are available (no nickels) then change for 30 cents is made with one quarter and 5 pennies (6 coins) while it would be optimal to make change using 3 dimes.

It is thus not clear at all that the change-making algorithm is optimal assuming available denominations of pennies, nickels, dimes and quarters. As an illustration of the proof of optimality for algorithms, we will now show that this is in fact the case. We first need a preliminary result.

Greedy Change-Making

2.1.15. Lemma. If n is a positive integer, then n cents in change using quarters, dimes, nickels and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies and cannot have two dimes and a nickel. The amount of change in dimes, nickels and pennies cannot exceed 24 cents.

Proof.
The proof is a simple argument by contradiction. For example, if the optimal solution had more than two dimes, we could replace them by a quarter and a nickel, yielding a contradiction. The details are left to the reader. □

2.1.16. Theorem. Given coin denominations of pennies, nickels, dimes and quarters, Algorithm 2.1.14 produces change using the fewest coins possible.

Greedy Change-Making

Proof.

Suppose that there exists a number n so that there is a way to make change for n using fewer coins than the greedy algorithm produces. First we note that if q' is the number of quarters in this optimal way and q is the number of quarters in the greedy algorithm, then $q = q'$. This follows, because the greedy algorithm uses the most quarters possible ($q \geq q'$) and by Lemma 2.1.15 $q' \not< q$ (otherwise there would be ≥ 25 cents in dimes, nickels and pennies).

Now both ways of making change must contain the same value of pennies, nickels and dimes, since they contain the same number of quarters. There must be the same number of dimes, since the greedy algorithm uses the greatest number of dimes and in an optimal algorithm there are at most one nickel and four pennies. Similarly, we also have the same number of nickels and then also of pennies. □

Computational Complexity

Implementing an algorithm generally requires resources, both of a physical and a temporal type. That is, if we want to run an algorithm such as Algorithm 2.1.3 for finding the maximum element of a set, we will need computer hardware, in particular memory, as well as the time to run the algorithm. Different algorithms may have different requirements; one algorithm may require less time but more hardware resources than another.

These requirements can be compared by considering the ***space complexity*** and the ***time complexity*** of a given algorithm. The memory requirements (space complexity) depend on the data structures used. We will not cover this aspect of computer science in our course, and instead restrict our analysis to the time complexity of algorithms.

Time complexity is generally analyzed by considering the number of operations an algorithm requires for completion instead of actual time. This allows a comparison independent of specific computer characteristics (processor speeds, parallel vs. serial processing etc.)

Time Complexity

2.1.17. Example. Consider the time complexity of Algorithm 2.1.3 for finding the maximum element of a set. At each step, two comparisons are made:

- one comparison to determine whether the end of the list has been reached and
- one comparison of the temporary maximum with the current element.

If the list has n elements, these comparisons are performed $n-1$ times, with an additional comparison to exit the loop. Therefore, a total of $2n-1$ comparisons are made.

It is not essential how many computer operations are necessary to implement a single "comparison" or how long each comparison takes. What is important is that when the list size doubles, so does (essentially) the number of comparisons and hence the time needed for the algorithm. The factor 2 and the subtraction of 1 is not relevant in practice.

Integer Sequences

We now attempt to express this more formally. We associate to every input size $n \in \mathbb{N}$ a number of steps $a_n \in \mathbb{N}$ that the algorithm needs to complete. Hence, we mostly need to study ***integer sequences***, but formally we consider sequences with values in the real numbers, if necessary.

The Landau Symbol O

The integer sequences we consider will all diverge to infinity, with only very few exceptions. Hence, we are not interested in their limits (a typical problem in calculus) but rather in their relative ***speed of growth***.

2.1.18. Definition. Let (a_n) and (b_n) be sequences. Then we say that

$$a_n = O(b_n) \qquad (\text{as } n \to \infty) \tag{2.1.1}$$

(read as "(a_n) is big-oh of (b_n)") if there exist constants $C \geq 0$ and $N \in \mathbb{N}$ such that

$$|a_n| \leq C|b_n| \qquad \text{whenever } n > N. \tag{2.1.2}$$

The letter "O" in (2.1.1) is called the ***Landau symbol*** big-oh.

2.1.19. Examples.

(i) $n + n^2 = O(n^2)$ as $n \to \infty$,

(ii) $\displaystyle\sum_{k=1}^{n} \frac{1}{k} = O\big(\ln(n)\big)$ as $n \to \infty$.

The Landau Symbol O

2.1.20. Remark. The notation used in (2.1.1) is a little strange: (a_n) is a sequence and hence a_n is a number, but what is $O(b_n)$? It can not itself represent a number, because we have

$$n^2 + 1 = O(n^2) \quad \text{and} \quad n^2 + 2 = O(n^2) \quad \text{but} \quad n^2 + 1 \neq n^2 + 2.$$

So, clearly, the statement "$n^2 + 1 = O(n^2)$ as $n \to \infty$" is a fixed expression and can only be understood in its entirety. The "=" symbol has a special meaning in this context.

2.1.21. Remark. From the definition of O we see that "$a_n = O(b_n)$ as $n \to \infty$" means that the terms a_n are bounded by a constant times b_n.

We can think of this as meaning that the terms of (a_n) are "not much larger" than those of (b_n).

The Landau Symbol O

2.1.22. Remark. It is not actually necessary in (2.1.2) to require the existence of an $N > 0$ such that the inequality holds for $n > N$; since there are only finitely many terms $a_0, a_1, \ldots, a_{N-1}$ before N, there exists a constant C such that (2.1.2) holds for $n > N$ if and only if that is the case for all $n \in \mathbb{N}$. However, we have left the definition as it stands because it is then closer to the analogous definition for functions.

In most cases, it is not necessary to check the definition when comparing (a_n) with (b_n), as the following useful result holds:

2.1.23. Theorem. Let (a_n) and (b_n) be sequences. If there exists some $C \geq 0$ such that

$$\lim_{n\to\infty} \frac{|a_n|}{|b_n|} = C,$$

then $a_n = O(b_n)$ as $n \to \infty$.

Landau Symbols using Limits

Proof.

Assume that for some $C \geq 0$

$$\lim_{n\to\infty} \frac{|a_n|}{|b_n|} = C.$$

Then

$$\forall_{m\in\mathbb{N}} \exists_{N>0} \forall_{n\in\mathbb{N}} \; n > N \Rightarrow \left| \frac{|a_n|}{|b_n|} - C \right| < \frac{1}{m}.$$

Choose $m = 1$. Then there exists some $N > 0$ such that $n > N$ implies

$$\big| |a_n| - C|b_n| \big| < |b_n|,$$

i.e.,

$$|a_n| \leq (C+1)|b_n|.$$

This is just the definition of $a_n = O(b_n)$ as $n \to \infty$. □

Variants of Landau Symbols

If $a_n = O(b_n)$, then the values of a_n are not larger than a constant (independent of n) times the values of b_n when n is large enough. However, a_n can actually be much smaller than b_n, e.g., we could write

$$\frac{1}{n} = O(n^{100})$$

which would be correct, but pretty meaningless. In many practical applications, one wants to express that not only is a_n not much larger than b_n, but also it is not much smaller. Physicists have often done this by defining special variants of Landau symbols, which look identical but according to the publication in which they occur are defined differently. This can of course lead to some confusion.

To clarify the situation, the computer scientist Donald E. Knuth has introduced common variants using different symbols. To my knowledge, they are, however, not widely used in the physics and mathematics literature.

The Ω and Θ Symbols

The symbol O in the expression $a_n = O(b_n)$ essentially gives an upper growth bound for a_n. The following symbols also give lower growth bounds:

2.1.24. Definition. Let (a_n) and (b_n) be sequences. Then we define, as $n \to \infty$,

$$a_n = \Omega(b_n) \qquad :\Leftrightarrow \qquad b_n = O(a_n), \tag{2.1.3}$$

read "(a_n) is big-omega of (b_n)".

Furthermore, we define

$$a_n = \Theta(b_n) \qquad :\Leftrightarrow \qquad a_n = O(b_n) \text{ and } a_n = \Omega(b_n), \tag{2.1.4}$$

read "(a_n) is big-theta of (b_n)".

Time Complexity of Bubble Sort

2.1.25. Example. Consider the time complexity of Algorithm 2.1.11 (Bubble Sort). During the ith pass, $n-i$ comparisons are used and $n-1$ passes are made. The total number of comparisons is then

$$\sum_{i=1}^{n-1}(n-i) = n(n-1) - \frac{n(n-1)}{2} = \frac{n(n-1)}{2}$$

Since

$$\frac{1}{4}n^2 \leq \frac{n(n-1)}{2} \leq \frac{1}{2}n^2 \qquad \text{for } n \geq 2$$

we see that

$$\sum_{i=1}^{n-1}(n-i) = \Theta(n^2)$$

This tells us that the number of comparisons of list elements grows quadratically with the size of the list. That means that if the list size is doubled, the number of list element comparisons required to sort the list grows fourfold.

Time Complexity of Bubble Sort

We should also take into account that comparisons are made in each of the loops to detect whether the loop indices i and j have reached their limit. In fact, there is one such comparison for each comparison of list elements (since at each step of the algorithm, there is one list element comparison) and we hence have

$$\sum_{i=1}^{n-1}(n-i) = \Theta(n^2)$$

comparisons to see if the loops have terminated.

From this we see that there are a total of

$$\Theta(n^2) + \Theta(n^2) = \Theta(n^2)$$

comparisons. Hence, the comparisons of the loop variables do not affect the time complexity of the Bubble Sort algorithm.

Interpreting the Time Complexity of Algorithms

2.1.26. Remark. In nearly all practical applications, the number of comparisons used to see if a loop has ended is not larger than the number of comparisons (or operations) used within a loop. Therefore, in most cases, it is fine to neglect these comparisons when determining the complexity of an algorithm.

It is also important to define precisely in what terms complexity is expressed and what it measures; in Example 2.1.25, the complexity measures

the number of comparisons as a function of list length.

However, in practice, comparing two large numbers with many digits takes more time than comparing two small numbers. This is not taken into account in our analysis. This is not a problem, but we need to state clearly at all times, what we are expressing by a given complexity.

Interpreting the Time Complexity of Algorithms

Also, in Example 2.1.25, the shifts that are performed (where an element is moved to another position in the list) are not counted. It is assumed that the "cost" of these shifts (the time needed to perform them) is small compared to the time needed for comparisons.

All this should make clear that the (asymptotic) time complexities we are calculating do not completely reflect the practical speed at which different algorithms can run. The asymptotic time complexity expresses the behavior of an algorithm at infinity, that is in the limit of infinite input size. In practice, an algorithm with a worse asymptotic complexity could perform better than another one with a lower complexity, since in practice the input size is finite.

For example, both Insertion Sort and Bubble Sort need $O(n^2)$ comparisons, but their actual speed may be different when both are implemented on the same computer.

Categorization of Complexities

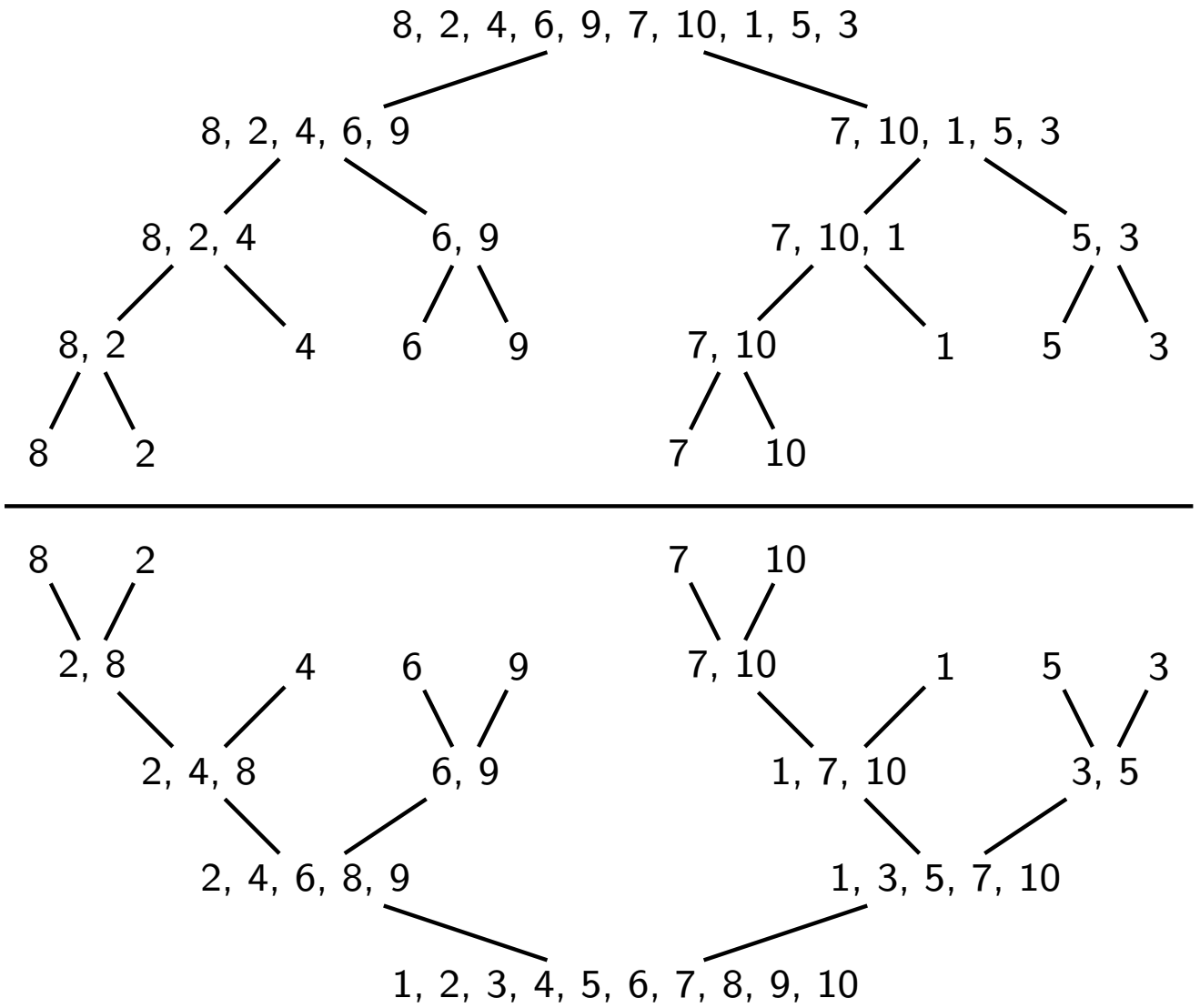
In general, we distinguish between the following complexities for algorithms (in increasing order of complexity):

Complexity	Terminology
$\Theta(1)$	constant complexity
$\Theta(\log n)$	logarithmic complexity
$\Theta(n)$	linear complexity
$\Theta(n \log n)$	$n \log n$ complexity
$\Theta(n^b)$, $b > 0$	polynomial complexity
$\Theta(b^{n^\varepsilon})$, $b > 1$, $0 < \varepsilon < 1$	sub-exponential complexity
$\Theta(b^n)$, $b > 1$	exponential complexity
$\Theta(n!)$	factorial complexity

Merge Sort

An efficient sorting algorithm that can be formulated recursively is the ***merge sort***. It works by successively splitting a list in half until only lists of length 1 remain. These are then successively joined to ordered lists.

We illustrate by showing how merge sort orders the list 8, 2, 4, 6, 9, 7, 10, 1, 5, 3 at right.



Merge Sort

The recursive pseudocode for merge sort is quite simple; it depends on having an algorithm for merging two ordered lists into a single ordered list:

2.1.27. Algorithm. ***(Merge Sort)***

Input: $L = a_1, \ldots, a_n$
Output: L, sorted into elements in nondecreasing oder

```
1 Function MergeSort(L):
2     if n > 1 then
3         m ← ⌊n/2⌋;
4         L1 ← a1, ..., am;
5         L2 ← am+1, ..., an;
6         L ← Merge(MergeSort(L1), MergeSort(L2))
7     end if
8     return L
9 end
```

Merging Two Sorted Lists

2.1.28. Algorithm. ***(Merge)***

Input: L_1, L_2, two sorted lists
Output: L a merged list of L_1 and L_2, with elements in increasing order

```
1  Function Merge(L1, L2):
2      L ← empty list;
3      while L1 and L2 are both nonempty do
4          remove smaller of first element of L1 and L2 from the list it is in
           and put it at the right end of L;
5          if removal of this element makes one list empty then
6              remove all elements from the other list and append to L
7          end if
8      end while
9      return L
10 end
```

Complexity of Merging Two Sorted Lists

2.1.29. Lemma. Two sorted lists with m and n elements, respectively, can be merged into a sorted list using no more than $n + m - 1$ comparisons.

Proof.

The least efficient way for Algorithm 2.1.28 to run is for none of the lists to become empty while the other has more than one element in it. Then $m - 1 + n - 1$ comparisons are made. The final comparison is then made between the remaining two elements. □

Complexity of Merge Sort

We now analyze the efficiency of merge sort. Recall that bubble sort needed $\Theta(n^2)$ comparisons to sort a list with n elements. Merge sort is rather more efficient.

2.1.30. Theorem. The number of comparisons needed to merge sort a list with n elements is $O(n \log n)$.

Proof.
We assume that $n = 2^m$ for simplicity. If this is not the case, the following arguments can be modified and will yield the same result.

At the first stage of the splitting procedure, the list is split into two sublists with 2^{m-1} elements each. This procedure is repeated, and after k splittings, there are 2^k sublists with 2^{m-k} elements each. After m steps, there are 2^m lists of length 1. These 2^m lists are merged into 2^{m-1} lists of length 2, requiring 2^{m-1} comparisons.

Complexity of Merge Sort

Proof (continued).

This procedure continues and generalizes as follows: at level k ($k = m, m-1, \ldots, 2, 1$) 2^k lists each with 2^{m-k} elements are merged into 2^{k-1} lists. By Lemma 2.1.29, each of the 2^{k-1} mergers requires at most $2^{m-k} + 2^{m-k} - 1 = 2^{m-k+1} - 1$ comparisons.

The total number of comparisons is required is then

$$\begin{aligned}\sum_{k=1}^{m} 2^{k-1}(2^{m-k+1} - 1) &= \sum_{k=1}^{m} 2^m - \sum_{k=1}^{m} 2^{k-1} \\ &= m2^m - (2^m - 1) \\ &= n \log n - n + 1.\end{aligned}$$

□

Efficiency of Sorting

The merge sort algorithm is "optimal" in the following sense:

2.1.31. Theorem. A sorting algorithm based on comparisons of pairs of elements needs $\Omega(n \log n)$ comparisons to sort a list of n elements.

We will prove this theorem later in the context of the theory of trees.

In the following section, we consider several computational algorithms arising in number theory and cryptography.

Representations of Integers

In the first part of the course, the properties of numbers in general and integers in particular were investigated. The results we obtained are generally applicable to integers, without regard to the way in which they are represented.

However, to implement these theorems and methods in practice, a concrete representation of integers needs to be chosen. Moreover, choosing a beneficial representation may allow algorithms to be run more efficiently than in certain other representations.

It is common to represent numbers in terms of decimal notation, e.g., we write

$$124 = 1 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0.$$

However, nothing is special about the number 10 and we could also write

$$124 = 1 \cdot 8^2 + 7 \cdot 8^1 + 4 \cdot 8^0$$

using the ***base*** 8.

Representations of Integers

2.2.1. Definition and Theorem. Let $b \in \mathbb{N} \setminus \{0, 1\}$ and $a \in \mathbb{N} \setminus \{0\}$. Then there exist unique numbers $a_0, \ldots, a_k < b$ with $a_k \neq 0$ such that

$$a = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0. \tag{2.2.1}$$

The representation (2.2.1) is called the ***base b expansion of a***. The numbers a_k are called the ***digits*** of a in base b.

The proof of Theorem 2.2.1 can be performed using induction. We will omit it here.

Several bases are in common use:

- $b = 2$ (binary)
- $b = 8$ (octal)
- $b = 10$ (decimal)
- $b = 16$ (hexadecimal)

Representations of Integers

Analogously to the case $b = 10$ we will give a number in a base b simply by listing its digits in this base. We indicate the base of the expansion by a subscript as follows:

$$124 = (124)_{10} = (174)_8.$$

For the hexadecimal expansion we represent the digits a_j in (2.2.1) through

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}.$$

For example, the number $(2\mathrm{AE0}B)_{16}$ is given by

$$2 \cdot 16^4 + 10 \cdot 16^3 + 14 \cdot 16^2 + 0 \cdot 16 + 11 = (175627)_{10}.$$

Base Conversion

The table below gives the numbers 0 through 15 in different base expansions.

Decimal	0	1	2	3	4	5	6	7
Hexadecimal	0	1	2	3	4	5	6	7
Octal	0	1	2	3	4	5	6	7
Binary	0	1	10	11	100	101	110	111
Decimal	8	9	10	11	12	13	14	15
Hexadecimal	8	9	A	B	C	D	E	F
Octal	10	11	12	13	14	15	16	17
Binary	1000	1001	1010	1011	1100	1101	1110	1111

It is very easy to convert between binary and hexadecimal expansions because each hexadecimal digit corresponds to a block of four binary digits (as listed above with leading zeroes omitted). For example,

$$(1101\ 1001\ 1100\ 0000)_2 = (\mathrm{D9C0})_{16}.$$

Base Conversion

To represent a given number a in an arbitrary base b we proceed as follows:

$$a = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

so by the Division Algorithm 1.6.1 we obtain

$$a_0 = a \bmod b.$$

Let

$$q_0 = a \operatorname{div} b = \frac{a - a_0}{b}.$$

Then

$$a_1 = q_0 \bmod b \qquad \text{and we set} \qquad q_1 = q_0 \operatorname{div} b.$$

Continuing this process until $q_k \operatorname{div} b = 0$ for some $k \in \mathbb{N}$, we find all coefficients of the base b expansion.

Base Conversion

2.2.2. Example. To find the octal expansion of $(12345)_{10}$ we first divide by 8 to obtain $12345 = 8 \cdot 1543 + 1$, so

$$\begin{aligned} 1543 &= 12345 \operatorname{div} 8, & 1 &= 12345 \bmod 8, \\ 192 &= 1543 \operatorname{div} 8, & 7 &= 1543 \bmod 8, \\ 24 &= 192 \operatorname{div} 8, & 0 &= 192 \bmod 8, \\ 3 &= 24 \operatorname{div} 8, & 0 &= 24 \bmod 8, \\ 0 &= 3 \operatorname{div} 8, & 3 &= 3 \bmod 8. \end{aligned}$$

We hence obtain $(12345)_{10} = (30071)_8$.

Pseudocode for Base Conversion

2.2.3. Algorithm. ***(Base Expansion)***

Input: a, a positive integer
Output: $A = (a_n \dots a_0)_b$, the expansion of a in base b

```
1 q ← a ; k ← 0 ; A ← ()_b;
2 while q ≠ 0 do
3 |   a_k ← q mod b;
4 |   prepend a_k to A;
5 |   q ← q div b;
6 |   k ← k + 1;
7 end while
8 return A
```

The number of operations used by the algorithm depends on the implementation of the Division Algorithm used to calculate q mod b and q div b.

Computer Arithmetic

Computers work with numbers in binary representation (also called ***binary numbers*** for short), e.g.,

$$(1000)_{10} = (1111101000)_2$$

The sequence of numbers on the right-hand side, 1111101000, is called a ***bit string*** and each digit is called a ***bit***.

A bit string of length 8 (called a ***byte***) stores a number of size between 0 and $2^8 - 1$. One byte can also be represented by two hexadecimal digits.

We will now consider several algorithms for basic arithmetic operations, starting with addition. These operations will assume that numbers are given in binary representation.

Addition of Integers

Let $a = (a_{n-1}a_{n-2}\ldots a_0)_2$ and $b = (b_{n-1}b_{n-2}\ldots b_0)_2$ be two binary numbers. We then calculate their sum

$$s = a + b = (s_n s_{n-1} s_{n-2}\ldots s_0)_2$$

as follows: We start with adding the right-most digits,

$$a_0 + b_0 = c_0 \cdot 2 + s_0 \tag{2.2.2}$$

so that s_0 is the right-most digit of $a + b$ and c_0 is called the ***carry***.

We next add

$$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1,$$

so that $(a + b)_1 = s_1$. We continue this procedure until we reach

$$a_{n-1} + b_{n-1} + c_{n-2} = c_{n-1} \cdot 2 + s_{n-1}.$$

Finally, $s_n = c_{n-1}$ is the leading digit of s.

Pseudocode for the Addition of Integers

2.2.4. Algorithm. ***(Integer Addition in Base 2)***

Input: $a = (a_{n-1}a_{n-2}\dots a_0)_2$ and $b = (b_{n-1}b_{n-2}\dots b_0)_2$
Output: $s = (s_n \dots s_0)_2 = a + b$

```
1 c ← 0 ; s ← ()_2 ;
2 for j ← 0 to n − 1 do
3 |   if a_j + b_j + c ≤ 1 then d = 0 else d = 1;
4 |   s_j ← a_j + b_j + c − 2d;
5 |   prepend s_j to s;
6 |   c ← d;
7 end for
8 append c to s;
9 return s
```

2.2.5. Remark. A similar algorithm for subtraction will be part of the assignments.

Complexity of Addition of Integers

2.2.6. Remark. The addition of two bits as in (2.2.2) is a single ***bit operation*** and we will express the complexity of algorithms in terms of such bit operations. (This is similar to our discussion of sorting algorithms, where the complexity was expressed in terms of ***elementary comparisons***.)

It is easy to see that if a and b are n-digit binary numbers, Algorithm 2.2.4 uses

$$O(n) = O(\log_2(a)) = O(\log_{10}(a))$$

additions of bits.

Multiplication also works in the same way as it is taught in school. However, since the numbers are in binary representation, there is a small advantage: instead of bit-wise multiplications, only bit-wise shifts need to be performed.

Pseudocode for the Multiplication of Integers

2.2.7. Algorithm. ***(Integer Multiplication in Base 2)***

Input: $a = (a_{n-1} \dots a_0)_2$ and $b = (b_{n-1} \dots b_0)_2$
Output: $p = a \cdot b$

```
1 for j ← 0 to n − 1 do        /* c_0, ..., c_{n−1}: partial products */
2 |   if b_j = 1 then c_j = a shifted j places;
3 |   else c_j = 0;
4 end for
5 p_0 ← 1;
6 for j ← 0 to n − 1 do
7 |   p ← p + c_j
8 end for
9 return p
```

Complexity of Integer Multiplication

For each partial product c_j, n bits need to be shifted. Since there are n partial products, this means that $O(n^2)$ bit shift operations are necessary Finally, n numbers need to be added, each addition requiring $O(n)$ bit additions.We hence have in total

$$O(n^2) + n \cdot O(n) = O(n^2)$$

bit operations.

2.2.8. Remark. There exist more efficient algorithms for multiplication. One, which we will introduce later in Example 2.3.16, uses $O(n^{\log 3}) = O(n^{1.584})$ operations.

Let us now consider the division algorithm, which is also the foundation of the base conversion Algorithm 2.2.3.

Pseudocode for the Division Algorithm

2.2.9. Algorithm. *(The Division Algorithm – div and mod)*

Input: a an integer and d a positive integer
Output: $q = a \operatorname{div} d$ and $r = a \mod d$

```
1  q ← 0 ; r ← |a|;
2  while r ≥ d do
3  |   r ← r − d;
4  |   q ← q + 1;
5  end while
6  if a < 0 and r > 0 then
7  |   r ← d − r;
8  |   q ← −(q + 1);
9  end if
10 return q, r
```

Efficiency of the Division Algorithm

Algorithm 2.2.9 uses $O(q \log a)$ bit operations.

2.2.10. Remark. There exist more efficient algorithms that require only $O(\log a \cdot \log d)$ operations.

Computer Arithmetic with Large Integers

The previously discussed operations for addition and multiplication do not scale very well when very large numbers (with thousands of digits or more) are to be manipulated. The following approach is a way around this.

Let $m_1, \dots, m_n \in \mathbb{N} \setminus \{0, 1\}$ be pairwise relatively prime, $m = \prod_{i=1}^{n} m_i$ and $a \in \mathbb{Z}$ with $0 \leq a < m$. Then it follows from the Chinese Remainder Theorem (see exercises) that a is uniquely determined by the n-tuple

$$(a \bmod m_1, \dots, a \bmod m_n).$$

2.2.11. Example. Take $m_1 = 3$, $m_2 = 4$. Then all integers less than $3 \cdot 4 = 12$ can be represented using their remainders upon division by 3 and 4 as follows:

n	0	1	2	3	4	5
$(n \bmod 3, n \bmod 4)$	(0,0)	(1,1)	(2,2)	(0,3)	(1,0)	(2,1)
n	6	7	8	9	10	11
$(n \bmod 3, n \bmod 4)$	(0,2)	(1,3)	(2,0)	(0,1)	(1,2)	(2,3)

Computer Arithmetic with Large Integers

We can now perform arithmetic operations on these tuples of integers instead of the original number.

2.2.12. Example. The numbers 95,97,98,99 are pairwise relatively prime. Then we can represent any integer less than $99 \cdot 98 \cdot 97 \cdot 95 = 89\,403\,930$ as a quadruple of the remainders with respect to these numbers.

To calculate the sum $123684 + 413456$ we first represent the summands as quadruples (n mod 99, n mod 98, n mod 97, n mod 95) and then add them:

$$\begin{aligned} & (33, 8, 9, 89) + (32, 92, 42, 16) \\ = {} & (65 \bmod 99, 100 \bmod 98, 51 \bmod 97, 105 \bmod 95) \\ = {} & (65, 2, 51, 10) \end{aligned}$$

To convert the result back to a number, we solve the system of congruences

$$x \equiv 65 \pmod{99}, \quad x \equiv 2 \pmod{98}, \quad x \equiv 52 \pmod{97}, \quad x \equiv 10 \pmod{}$$

and find $x = 537\,140$.

Modular Exponentiation

In cryptography, for example in the Diffie-Hellman Key Agreement Protocol, it is necessary to calculate $b^n \bmod m$ efficiently, where $n \in \mathbb{N}$, $b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0, 1\}$. In general, b^n will be very large, so it is not practical to calculate b^n first and then compute the modulus.

We first develop an efficient strategy for exponentiation (more efficient than multiplying as in Algorithm 2.2.7) and combine it with finding the remainder. We use the binary expansion of n to compute b^n:

$$b^n = b^{a_{k-1}2^{k-1}+\cdots+a_1\cdot 2+a_0} = \prod_{j=0}^{k-1} b^{a_j 2^j}, \qquad a_0, \ldots, a_{k-1} \in \{0, 1\}.$$

We first find all powers b^{2^j} recursively, by noting that $b^{2^{j+1}} = (b^{2^j})^2$. Next we calculate their values $\bmod m$ and finally compute their product.

Pseudocode for Modular Exponentiation

2.2.13. Algorithm. ***(Modular Exponentiation)***

Input: b an integer, $n = (n_{k-1} \dots n_0)_2$ and m two positive integers
Output: $x = b^n \mod m$

```
1 x ← 1 ; power ← b mod m;
2 for i ← 0 to k − 1 do
3 |   if n_i = 1 then x ← (x · power) mod m;
4 |   power ← (power · power) mod m;
5 end for
6 return x
```

Supposing that the more efficient division algorithm referred to in Remark 2.2.10 is used, Algorithm 2.2.13 uses $O((\log m)^2 \log n)$ bit operations to find $b^n \mod m$.

Pseudocode for the Euclidean Algorithm

We close this section with a discussion of the Euclidean algorithm:

2.2.14. Algorithm. ***(The Euclidean Algorithm)***

Input: a, b, two positive integers
Output: $x = \gcd(a, b)$

```
1 x ← a ; y ← b;
2 while y ≠ 0 do
3 |   r ← x mod y;
4 |   x ← y ; y ← r;
5 end while
6 return x
```

The following theorem is noteworthy, because its proof uses an "inverse" approach to finding the complexity: instead of asking how many operations are needed for a given input, it discusses the input size if a given number of operations are used.

Efficiency of the Euclidean Algorithm

2.2.15. Lamé's Theorem. Let $a, b \in \mathbb{N} \setminus \{0\}$ with $a \geq b$. Then the number of divisions used by the Euclidean Algorithm to find $\gcd(a, b)$ is not greater than five times the number of decimal digits in b.

Proof.

Setting $a = r_0$ and $b = r_1$, each step of the algorithm produces an equivalent equation as follows:

$$\begin{aligned}
r_0 &= r_1 q_1 + r_2, & 0 \leq r_2 < r_1,\\
r_1 &= r_2 q_2 + r_3, & 0 \leq r_3 < r_2,\\
&\vdots\\
r_{n-2} &= r_{n-1} q_{n-1} + r_n & 0 \leq r_n < r_{n-1}\\
r_{n-1} &= r_n q_n,
\end{aligned}$$

where $r_n = \gcd(a, b)$. Note that $q_1, \ldots q_{n-1} \geq 1$ and $q_n \geq 2$, since $r_{n-1} > r_n$.

Efficiency of the Euclidean Algorithm

Proof (continued).

Denote by $(f_n)_{n\in\mathbb{N}}$ the sequence of Fibonacci numbers of Example 1.3.7, $(f_n) = (0, 1, 1, 2, 3, 5, \ldots)$. Then

$$\begin{aligned} r_n &\geq 1, & r_n &\geq f_2, \\ r_{n-1} &\geq 2r_n \geq 2, & r_{n-1} &\geq f_3, \\ r_{n-2} &\geq r_{n-1} + r_n, & r_{n-2} &\geq f_2 + f_1 = f_4, \\ &\vdots & & \\ r_2 &\geq r_3 + r_4 & r_2 &\geq f_{n-1} + f_{n-2} = f_n, \\ b = r_1 &\geq r_2 + r_3, & b &\geq f_n + f_{n-1} = f_{n+1}. \end{aligned}$$

It follows that if n steps are needed to complete the Euclidean Algorithm, then

$$b \geq f_{n+1}.$$

Efficiency of the Euclidean Algorithm

Proof (continued).

It can be shown by induction that, for $n \geq 2$,

$$f_n \geq \phi^{n-2}, \qquad \phi = \frac{1+\sqrt{5}}{2}. \tag{2.2.3}$$

A quick calculation gives

$$\phi^5 = \frac{11+5\sqrt{5}}{2} > 10,$$

so $\log_{10} \phi > 1/5$. This implies

$$\log_{10} b \geq \log_{10} f_{n+1} > (n-1)\log_{10} \phi > \frac{n-1}{5}.$$

Efficiency of the Euclidean Algorithm

Proof (continued).
Hence

$$n < 5 \log_{10} b + 1 < 5k + 1,$$

where k is the number of decimal digits of b. Since n is an integer, this implies $n \leq 5k$. □

2.2.16. Remarks.

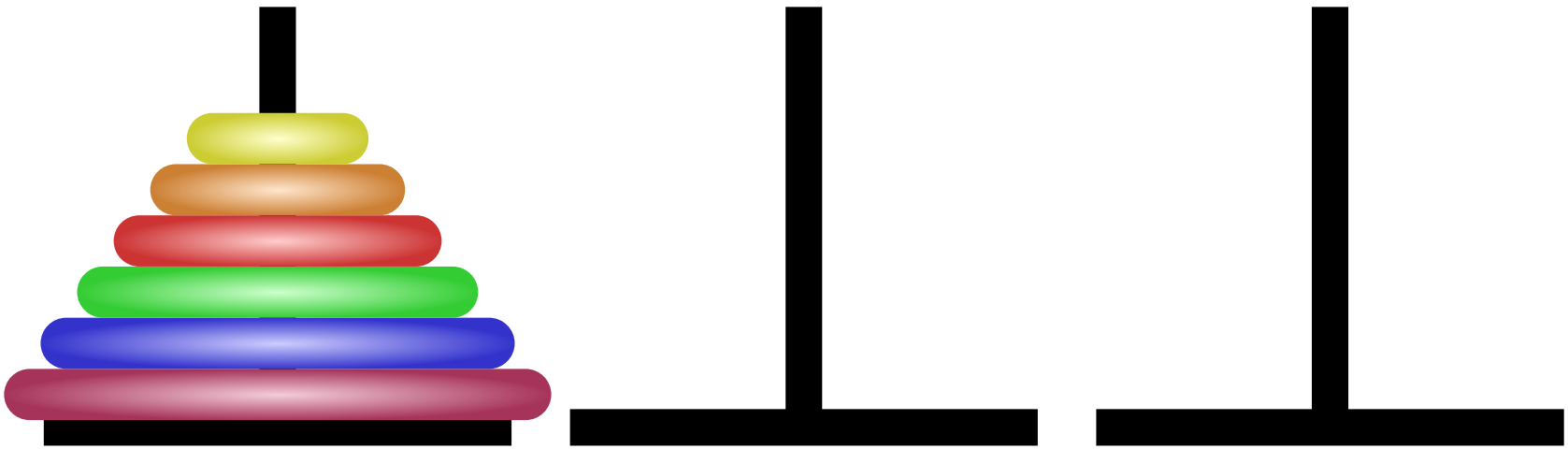
- Lamé's Theorem implies that the Euclidean algorithm uses $O(\log b)$ divisions to find $\gcd(a, b)$ when $a \geq b$.
- A crucial element of the proof is to find an estimate on the size of the nth Fibonacci number f_n. In the following section, we will consider this and other recursively defined sequences in more detail.

Tower of Hanoi

Let us begin this section with some examples.

2.3.1. Example. The tower of Hanoi is a famous problem: n concentric disks of increasing radius are placed upon one of three pegs. The goal is to move the stack of disks from Peg 1 to Peg 2, but in such a way that

- only one disk is moved at a time and
- a larger disk is never placed on top of a smaller disk.



Tower of Hanoi

Our goal is to find a ***recurrence relation*** for the number H_n of moves required to move a stack of n disks from Peg 1 to Peg 2. Suppose we need H_{n-1} moves to transfer the stack of $n-1$ disks to Peg 3:

We can then transfer the largest disk to Peg 2, and then again use H_{n-1} moves to transfer the other disks to Peg 2 on top of the largest disk. Thus,

$$H_n = 2H_{n-1} + 1, \qquad n \geq 2, \qquad H_1 = 1.$$

Tower of Hanoi

The first few terms of (H_n) are

$$1,\ 3,\ 7,\ 15,\ 31, \ldots$$

It seems that $H_n = 2^n - 1$. In fact, it is easy to prove this by induction: $H_1 = 2^1 - 1 = 1$, and if $H_{n-1} = 2^{n-1} - 1$, then

$$H_n = 2H_{n-1} + 1 = 2 \cdot (2^{n-1} - 1) + 1 = 2^n - 1.$$

Fibonacci Numbers

2.3.2. Example. Fibonacci originally posed the following problem in his ***Liber Abbaci*** (Book of Calculations) in 1202:

> *A certain man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair, which from the second month on becomes productive?*

The solution to this problem leads to the sequence of Fibonacci numbers:

- At the beginning of month 1, there is $f_1 = 1$ pair, which is not productive.
- At the beginning of month 2, there is still $f_2 = 1$ pair, which is now productive.
- At the beginning of month 3, there are now $f_3 = 2$ pairs, of which one is productive.

Fibonacci Numbers

- At the beginning of month 4, there are now $f_4 = 3$ pairs and the pair born in month 3 becomes productive.
- As the beginning of month 5, the number of pairs is equal to those of month 4, plus all those that were productive in month 4. These are all pairs that existed in month 3, since all of those will be productive in month 4. Hence, $f_5 = 3 + 2 = 5$.
- In general, at the beginning of every month the number of pairs of rabbits is equal to the number of pairs of the previous month, plus the number of pairs of two months ago, which have since become productive.

Hence, the classical Fibonacci sequence is

$$f_n = f_{n-1} + f_{n-2}, \qquad n \geq 2, \qquad f_1 = 1, \qquad f_2 = 1.$$

Recurrence Relations

In the proof of Lamé's theorem we have used a basic estimate on the Fibonacci sequence. We now discuss how to obtain this and similar estimates in more general cases. This will enable us to study the complexity of a wide class of algorithms.

2.3.3. Definition. A ***recurrence relation*** for a sequence (a_n) is an equation of the form

$$a_n = f(a_0, \dots, a_{n-1}), \qquad \text{for } n \geq k \tag{2.3.1}$$

for some $k \in \mathbb{N}$.

A sequence (a_n) is called a solution to the recurrence relation if the terms of the sequence satisfy (2.3.1).

Numbers $a_0, \dots, a_{k-1}$ that are given together with (2.3.1) are called ***initial conditions*** for the recurrence relation.

Existence and Uniqueness for Recurrence Relations

2.3.4. Lemma. Let $a_0, \ldots, a_{k-1} \in \mathbb{R}$ be given. Then there exists a unique sequence (a_n) such that a_n satisfies (2.3.1).

Proof.

The existence of a unique sequence (a_n) can be shown by strong induction in n: first, $a_0, \ldots, a_{k-1}$ are given (and exist uniquely). Suppose that for some $n \geq k$ we know that $a_k, \ldots, a_{n-1}$ exist uniquely such that (2.3.3) is satisfied with n replaced by $k, \ldots, n-1$, respectively. Then we define

$$a_n := f(a_0, \ldots, a_{n-1})$$

and we see that a_n exists uniquely. This is precisely the procedure used to verify that inductive definitions make sense. □

Linear Recurrence Relations

2.3.5. Definition. A linear recurrence relation of degree $k \in \mathbb{N} \setminus \{0\}$ with constant (real) coefficients is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k} + F(n) \qquad \text{for } n \geq k, \tag{2.3.2}$$

where $c_1, \ldots, c_k \in \mathbb{R}$, $c_k \neq 0$, and $F \colon \mathbb{N} \to \mathbb{R}$ is a sequence.

If $F(n) = 0$ for all n, (2.3.2) is said to be ***homogeneous***, otherwise ***inhomogeneous***.

We will first treat the homogeneous case

$$a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}. \tag{2.3.3}$$

The basic approach for solving (2.3.3) is to make the ansatz

$$a_n = r^n \tag{2.3.4}$$

and attempt to find a number $r \in \mathbb{R}$ such that (2.3.4) satisfies (2.3.3).

Linear Homogeneous Recurrence Relations of Degree 2

This leads to the ***characteristic equation*** for (2.3.3),

$$r^k - c_1 r^{k-1} - \cdots - c_{k-1} r - c_k = 0, \tag{2.3.5}$$

whose left-hand side is called the ***characteristic polynomial***. The solutions of the characteristic equation are called ***characteristic roots*** of (2.3.3).

We will first treat recurrence relations of degree 2; those of degree 1 are easily solvable, while those of higher degree imply discussing roots of polynomials of degree ≥ 3, which is fairly involved.

2.3.6. Theorem. Let $c_1, c_2 \in \mathbb{R}$ such that $r^2 - c_1 r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence (a_n) is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if

$$a_n = \alpha_1 \cdot r_1^n + \alpha_2 \cdot r_2^n, \qquad \alpha_1, \alpha_2 \in \mathbb{R}, \ n \in \mathbb{N}. \tag{2.3.6}$$

Linear Homogeneous Recurrence Relations of Degree 2

Proof.

The proof is in two parts: first, we show that (2.3.6) actually solves the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$; then, we show that every solution of $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ is of the form (2.3.6).

Suppose that $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$. Since r_1, r_2 are roots of $r^2 - c_1 r - c_2 = 0$ we have

$$\begin{aligned} c_1 a_{n-1} + c_2 a_{n-2} &= c_1(\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + c_2(\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2}) \\ &= \alpha_1 r_1^{n-2}(c_1 r_1 + c_2) + \alpha_2 r_2^{n-2}(c_1 r_2 + c_2) \\ &= \alpha_1 r_1^{n-2} r_1^2 + \alpha_2 r_2^{n-2} r_2^2 \\ &= a_n \end{aligned}$$

so the recurrence relation is satisfied. This shows that (2.3.6) solves the recurrence relation.

Linear Homogeneous Recurrence Relations of Degree 2

Proof (continued).

Now let (a_n) be a solution to the recurrence relation. By Lemma 2.3.4 this sequence is unique and determined by a_0 and a_1. We thus need to show that we can find α_1 and α_2 such that

$$a_0 = \alpha_1 + \alpha_2, \qquad a_1 = \alpha_1 r_1 + \alpha_2 r_2.$$

If this is possible, it follows that $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for all $n \in \mathbb{N}$. A simple calculation yields

$$\alpha_1 = \frac{a_1 - a_0 r_2}{r_1 - r_2}, \qquad \alpha_2 = \frac{a_0 r_1 - a_1}{r_1 - r_2}$$

so

$$a_n = \frac{a_1 - a_0 r_2}{r_1 - r_2} r_1^n + \frac{a_0 r_1 - a_1}{r_1 - r_2} r_2^n.$$

□

Linear Homogeneous Recurrence Relations of Degree 2

2.3.7. Example. The Fibonacci sequence satisfies $f_0 = 0$, $f_1 = 1$ and $f_n = f_{n-1} + f_{n-2}$. Setting $f_n = r^n$, we obtain

$$r^n - r^{n-1} - r^{n-2} = 0$$

or

$$r^2 - r - 1 = 0.$$

The characteristic roots are

$$r_{1,2} = \frac{1 \pm \sqrt{5}}{2},$$

so, finding suitable constants to satisfy the initial conditions, we obtain

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

Homogeneous Recurrence Relations of Degree 2

If there is only a single characteristic root of multiplicity 2, then we have the following result:

2.3.8. Theorem. Let $c_1, c_2 \in \mathbb{R}$, $c_2 \neq 0$ such that $r^2 - c_1 r - c_2 = 0$ has a single roots r_0. Then the sequence (a_n) is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if

$$a_n = \alpha_1 \cdot r_0^n + \alpha_2 \cdot n r_0^n, \qquad \alpha_1, \alpha_2 \in \mathbb{R}, \ n \in \mathbb{N}.$$

The proof is left to the exercises.

2.3.9. Example. We consider the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$ with initial conditions $a_0 = 1$ and $a_1 = 6$. There is only one characteristic root, $r = 3$. Then

$$a_n = \alpha_1 3^n + \alpha_2 n 3^n.$$

To find α_1, α_2 we use $a_0 = \alpha_1 = 1$ and $a_1 = 3\alpha_1 + 3\alpha_2 = 6$. This gives

$$a_n = (n+1)3^n.$$

Homogeneous Recurrence Relations of Degree k

We will no prove the following result, which generalizes the solution ansatz to recurrence relations of degree k:

2.3.10. Theorem. Let $c_1, c_2, \ldots, c_k \in \mathbb{R}$ such that

$$r^k - c_1 r^{k-1} - \cdots - c_{k-1} r - c_k = 0$$

has t distinct roots $r_1, \ldots, r_t$ with multiplicities $m_1, \ldots, m_t$, respectively. Then the sequence (a_n) is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ if and only if

$$\begin{aligned} a_n = {} & (\alpha_{1,0} + \alpha_{1,1} n + \cdots + \alpha_{1,m_1-1} n^{m_1-1}) \cdot r_1^n \\ & + (\alpha_{2,0} + \alpha_{2,1} n + \cdots + \alpha_{2,m_2-1} n^{m_2-1}) \cdot r_2^n \\ & + \cdots + (\alpha_{t,0} + \alpha_{t,1} n + \cdots + \alpha_{t,m_t-1} n^{m_t-1}) \cdot r_t^n, \quad n \in \mathbb{N}. \end{aligned}$$

where $\alpha_{i,j} \in \mathbb{R}$, $1 \leq i \leq t$, $0 \leq j \leq m_i - 1$.

Homogeneous Recurrence Relations of Degree k

2.3.11. Example. We want to solve

$$a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3}, \qquad a_0 = 1, \qquad a_1 = -2, \qquad a_2 = -1.$$

The characteristic root is found from

$$r^3 + 3r^2 + 3r + 1 = (r+1)^3 = 0$$

so we have $r = -1$ with multiplicity 3. The solution of the recurrence relation is then

$$a_n = \alpha_{1,0}(-1)^n + \alpha_{1,1}n(-1)^n + \alpha_{1,2}n^2(-1)^n.$$

To find the constants we solve

$$\begin{aligned} a_0 &= \alpha_{1,0} = 1, \\ a_1 &= -\alpha_{1,0} - \alpha_{1,1} - \alpha_{1,2} = -2, \\ a_2 &= \alpha_{1,0} + 2\alpha_{1,1} + 4\alpha_{1,2} = -1 \end{aligned}$$

to obtain

$$a_n = (1 + 3n - 2n^2)(-1)^n.$$

Inhomogeneous Recurrence Relations

In the inhomogeneous case we want to solve

$$a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k} + F(n) \qquad \text{for } n \geq k \qquad (2.3.7)$$

with constants $c_1, \ldots, c_k \in \mathbb{R}$. The general solution to (2.3.7) will be the sum of a particular solution and solutions of the associated homogeneous equation. This is analogous to the solution of inhomogeneous systems of linear algebraic equations or of inhomogeneous systems of linear ordinary differential equations.

2.3.12. Theorem. If (a_n^{part}) is a particular solution of (2.3.7), then every solution is of the form $(a_n) = (a_n^{\text{part}} + a_n^{\text{hom}})$, where (a_n^{hom}) solves (2.3.7) with $F(n) = 0$ for all n.

We omit the proof, which is similar to the corresponding statements for systems of ODEs or linear equations.

Inhomogeneous Recurrence Relations

2.3.13. Example. Consider the recurrence equation $a_n = 3a_{n-1} + 2n$ with initial value $a_1 = 3$. The associated homogeneous equation $a_n = 3a_{n-1}$ has general solution $a_n^{\text{hom}} = \alpha \cdot 3^n$. A particular solution can be found by educated guessing: since $F(n) = 2n$ is a polynomial of degree 1, try setting $a_n^{\text{part}} = cn + d$ for $c, d \in \mathbb{R}$ to be determined. Then

$$cn + d = 3(c(n-1) + d) + 2n \qquad \Rightarrow \qquad (2 + 2c)n + (2d - 3c) = 0,$$

so we have $c = -1$ and $d = -3/2$. Thus,

$$a_n = \alpha \cdot 3^n - n - 3/2$$

solves the recurrence relation. The initial condition $a_1 = 3$ then gives $\alpha = 11/6$, so

$$a_n = \frac{11}{2} 3^{n-1} - n - \frac{3}{2}.$$

Inhomogeneous Recurrence Relations

It is no accident that in this example we were able to find a particular solution by guessing it to be a polynomial. In fact, if $F(n)$ is an exponential function, then we can also find a particular solution of exponential form. In fact, the following theorem holds.

Inhomogeneous Recurrence Relations

2.3.14. Theorem. Consider the inhomogeneous equation (2.3.7) with

$$F(n) = s^n \sum_{j=0}^{t} b_j n^j, \qquad s, b_0, \dots, b_t \in \mathbb{R}.$$

A particular solution to (2.3.7) then has the following form:

- If s is not a characteristic root of the associated homogeneous equation, then there exist numbers $p_0, \dots, p_t \in \mathbb{R}$ such that

$$a_n^{\text{part}} = s^n \sum_{j=0}^{t} p_j n^j.$$

- If s is a characteristic root of multiplicity m, then there exist numbers $p_0, \dots, p_t \in \mathbb{R}$ such that

$$a_n^{\text{part}} = n^m s^n \sum_{j=0}^{t} p_j n^j.$$

Inhomogeneous Recurrence Relations

2.3.15. Example. Consider the recurrence relation

$$a_n = 6a_{n-1} - 9a_{n-2} + F(n),$$

where we are interested in the cases

1. $F(n) = 3^n$,
2. $F(n) = n3^n$,
3. $F(n) = n^2 2^n$,
4. $F(n) = (n^2 + 1)3^n$.

The associated homogeneous equation is $a_n = 6a_{n-1} - 9a_{n-2}$ which has a single characteristic root $r = 3$ of multiplicity 2. We then have particular solutions of the following forms:

1. $a_n^{\text{part}} = p_0 n^2 3^n$,
2. $a_n^{\text{part}} = (p_0 + p_1 n)n^2 3^n$,
3. $a_n^{\text{part}} = (p_0 + p_1 n + p_2 n^2)2^n$,
4. $a_n^{\text{part}} = (p_0 + p_1 n + p_2 n^2)n^2 3^n$.

Divide-and-Conquer Algorithms

The principle of "divide and conquer", (originating from ***divide et impera***, divide and rule, in Latin) goes back to the ancient Romans who used political, economic and military means to break up alliance of opponents or concentrations of power so that each part was less powerful than themselves.

In computer algorithms, ***divide-and-conquer*** algorithms are those that break up a problem into several smaller instances and tackle each instance separately. They are similar to recursive algorithms, but "multi-branched", i.e., referring to several instances of themselves, not just one, with smaller input.

We are interested in analyzing the time complexity of divide-and-conquer algorithms. It turns out that recurrence relations are well-suited to describing the number of steps needed for such an algorithm to complete and that we can find quite general formulas for big-Oh estimates for many classes of algorithms.

Divide-and-Conquer Algorithms

2.3.16. Examples.

(i) The Binary Search Algorithm 2.1.9 reduces the problem of locating an element of a list to locating it in a list of half the length. Two comparisons are needed to do this, one to determine which half-list to search, the other to determine whether any terms remain in the list. Therefore, the number of comparisons needed to search a list of length n is given by

$$f(n) = f(n/2) + 2.$$

Since only one list of length $n/2$ is searched, this is actually a recursive algorithm, a special case of a divide-and-conquer algorithm.

(ii) A divide-and-conquer algorithm to find the minimum and maximum of a list of length $n = 2k$ might split the list into two equally sized sublists, find the minimum and maximum in each sublist and compare the values of the two sublists. The number of comparisons needed is then given by

$$f(n) = 2f(n/2) + 2.$$

Divide-and-Conquer Algorithms

(iii) The ***Karatsuba algorithm*** is a divide-and-conquer algorithm for integer multiplication which is faster than Algorithm 2.2.7. Suppose that a and b are integers with binary representations of length $2n$ (adding leading zeroes if necessary),

$$a = (a_{2n-1} \dots a_1 a_0)_2, \qquad b = (b_{2n-1} \dots b_1 b_0)_2.$$

Setting

$$A_0 = (a_{n-1} \dots a_0)_2, \qquad A_1 = (a_{2n-1} \dots a_n)_2,$$
$$B_0 = (b_{n-1} \dots b_0)_2, \qquad B_1 = (b_{2n-1} \dots b_n)_2$$

we can write

$$a = 2^n A_1 + A_0, \qquad b = 2^n B_1 + B_0.$$

Divide-and-Conquer Algorithms

Then

$$a \cdot b = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0,$$

so the multiplication of two integers of length $2n$ is reduced to summing three multiplications of integers of length n. The number of operations required for the multiplications by powers of 2 (shifts in the binary expansion) and addition are proportional to n, so the total number of operations is given by

$$f(2n) = 3f(n) + Cn$$

for some $C \in \mathbb{N}$.

(iv) The Merge Sort Algorithm 2.1.27 reduces the problem of sorting a list with n elements to that of sorting two lists with $n/2$ elements. It then uses fewer than n comparisons to merge the two sorted lists. Therefore, it uses fewer than $M(n)$ comparisons, where

$$M(n) = 2M(n/2) + n.$$

Time Complexity of Divide-and-Conquer Algorithms

2.3.17. Theorem. Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + c, \qquad a \geq 1,\ b \in \mathbb{N} \setminus \{0, 1\},\ c > 0$$

whenever n is divisible by b. Then

$$f(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > 1, \\ O(\log n) & \text{if } a = 1. \end{cases}$$

Furthermore, whenever $a > 1$ and $n = b^k$ for some $k \in \mathbb{Z}_+$,

$$f(n) = C_1 n^{\log_b a} + C_2 \tag{2.3.8}$$

with

$$C_1 = f(1) + \frac{c}{a-1}, \qquad C_2 = -\frac{c}{a-1}.$$

Time Complexity of Divide-and-Conquer Algorithms

Proof.

We first note that if $f(n) = af(n/b) + g(n)$ for some function g,

$$\begin{aligned} f(n) &= af\left(\frac{n}{b}\right) + g(n) = a\left(af\left(\frac{n}{b^2}\right) + g\left(\frac{n}{b}\right)\right) + g(n) \\ &= a^2 f\left(\frac{n}{b^2}\right) + ag\left(\frac{n}{b}\right) + g(n) \\ &\vdots \\ &= a^k f\left(\frac{n}{b^k}\right) + \sum_{j=0}^{k-1} a^j g\left(\frac{n}{b^j}\right) \end{aligned}$$

for any $k \in \mathbb{N}$. If $n = b^k$ for some $k \in \mathbb{N}$, we obtain

$$f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j g\left(\frac{n}{b^j}\right). \tag{2.3.9}$$

Time Complexity of Divide-and-Conquer Algorithms

Proof (continued).

We will now prove the various statements of the theorem.

(i) Suppose that $a = 1$. If $n = b^k$ for some $k \in \mathbb{N}$, (2.3.9) gives

$$f(n) = f(1) + c \cdot k = f(1) + c \cdot \log_b n = O(\log n).$$

Furthermore, if $b^k < n < b^{k+1}$ for some $k \in \mathbb{N}$ we use that f is increasing to obtain

$$f(n) \leq f(b^{k+1}) = f(1) + c(k+1) = f(1) + c + c \cdot \log_b n = O(\log n).$$

This proves the statement of the theorem for $a = 1$.

Time Complexity of Divide-and-Conquer Algorithms

Proof (continued).

(ii) Now let $a > 1$ and $n = b^k$. Then (2.3.9) gives

$$\begin{aligned} f(n) &= a^k f(1) + c\sum_{j=0}^{k-1} a^j = a^k f(1) + c\frac{a^k - 1}{a-1} \\ &= a^k\left(f(1) + \frac{c}{a-1}\right) - \frac{c}{a-1} \\ &= C_1 n^{\log_b a} + C_2, \end{aligned}$$

where we have used $a^k = a^{\log_b n} = n^{\log_b a}$. This proves (2.3.8). If $b^k < n < b^{k+1}$ for some $k \in \mathbb{N}$ we use that f is increasing to obtain

$$f(n) \le f(b^{k+1}) = a^k\left(af(1) + \frac{ac}{a-1}\right) - \frac{c}{a-1} = O(n^{\log_b a}). \quad \square$$

Time Complexity of Divide-and-Conquer Algorithms

We now apply Theorem 2.3.17 to the first two of the Examples 2.3.16:

2.3.18. Examples.

(i) The comparisons needed by the Binary Search algorithm satisfy $f(n) = f(n/2) + 2$, so

$$f(n) = O(\log n).$$

(ii) The comparisons needed for finding the minimum and maximum of a list of integers satisfy $f(n) = 2f(n/2) + 2$, so

$$f(n) = O(n^{\log_2 2}) = O(n).$$

In order to treat the other examples, we need a more general theorem.

The Master Theorem

The following result is called the "Master Theorem" because it can be used to obtain the time complexity of many divide-and-conquer algorithms. There exist more powerful variants that give big-Θ estimates.

2.3.19. Master Theorem. Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d, \qquad a \geq 1,\ b \in \mathbb{N} \setminus \{0, 1\},\ c > 0,\ d \geq 0$$

whenever $n = b^k$. Then

$$f(n) = \begin{cases} O(n^d) & \text{if } a < b^d, \\ O(n^d \log n) & \text{if } a = b^d, \\ O(n^{\log_b a}) & \text{if } a > b^d, \end{cases}$$

The proof of the Master Theorem is relegated to the exercises.

The Master Theorem

We now apply the Master Theorem to the remaining two Examples 2.3.16:

2.3.20. Examples.

(i) The number of comparisons needed by the Merge Sort algorithm is bounded by $M(n)$, where $M(n) = 2M(n/2) + n$, so

$$M(n) = O(n \log n).$$

(ii) The number of operations needed for the fast integer multiplication satisfies $f(n) = 3f(n/2) + Cn$, so

$$f(n) = O(n^{\log_2 3}) = O(n^{1.585}).$$

This is a significant improvement on the "classical" algorithm, which uses $O(n^2)$ operations.

Counting

In mathematics, as well as in applications, it is important to know how to count. The act of counting can be described as finding the "number of elements" in a set. How is this done? In the final analysis, by comparing one set with a set where the number of elements is known.

For example, consider the set of bit strings of length 2,

$$A = \{00, 01, 10, 11\}.$$

Counting the elements in the set is done as follows:

$$00 \to 1,$$
$$01 \to 2,$$
$$10 \to 3,$$
$$11 \to 4.$$

We have found a one-to-one correspondence between the element of A and the elements of the set $\{1, 2, 3, 4\}$. We conclude that A has 4 elements.

Cardinality

We can make this more precise as follows:

2.4.1. Definition. Let A, B be arbitrary sets.

(i) The sets A and B have ***equal cardinality*** if there exists a bijective function $\varphi\colon A \to B$. In this case we write

$$\operatorname{card} A = \operatorname{card} B.$$

(ii) The set A is said to have a ***cardinality no greater than*** B if there exists an injective function $\varphi\colon A \to B$. In this case we write

$$\operatorname{card} B \geq \operatorname{card} A.$$

(iii) The set B is said to have a ***cardinality strictly greater than*** A if there exists an injective function $\varphi\colon A \to B$ but there does not exist a bijective function from A to B. In this case we write

$$\operatorname{card} B > \operatorname{card} A.$$

Cardinality

2.4.2. Remarks. Although we want to think of cardinality as meaning "number of elements", we have not actually defined this. In fact, we have not even stated what cardinality ***actually is***. We have only defined what it means that two sets have equal cardinality.

We are very careful with our definitions because a set may have an "infinite number of elements" and in fact it turns out that there are ***different sizes of infinity***! Therefore, terms such as "number of elements" need to be handled with extreme care.

This also means that "seemingly obvious" statements require non-trivial proofs or further assumptions. For example, the fact that

$$\operatorname{card} A \geq \operatorname{card} B \quad \wedge \quad \operatorname{card} B \geq \operatorname{card} A \quad \Rightarrow \quad \operatorname{card} A = \operatorname{card} B$$

is not trivial (***Cantor-Schröder-Bernstein Theorem***) and is in fact not accepted by a minority of mathematicians.

Finite, Infinite, Countable and Uncountable Sets

We can define

$$A \sim B \quad \Leftrightarrow \quad \operatorname{card} A = \operatorname{card} B \tag{2.4.1}$$

and this is in fact an equivalence relation (check this!). The set of all possible cardinalities of a set is equal to the set of equivalence classes induced by (2.4.1).

2.4.3. Definition. Let A be a set and $\sim$ the equivalence relation (2.4.1).

(i) If $A \sim \{1, 2, \dots, n\}$ for some $n \in \mathbb{N}$, we write $\operatorname{card} A = n$.
A is said to be ***finite***.

(ii) If $A \sim \mathbb{N}$, we write $\operatorname{card} A = \aleph_0$.
A is said to be ***countably infinite***.

If A is not finite, A is said to be ***infinite***.

If A is finite or countably infinite, A is said to be ***countable***, otherwise A is ***uncountable***.

Cardinalities of Infinite Sets

At this point, we rapidly start to leave the confines of naive set theory and enter territory where each statement has to be preceded by saying which axioms of set theory need to be assumed for the statement to be true.

For example, supposing a general, set-theoretic ***well-ordering principle***, the set of possible cardinalities can be ordered and the cardinalities of the infinite sets denoted by

$$\aleph_0 < \aleph_1 < \ldots$$

(Again, there are mathematicians who dispute this because they do not accept the general well-ordering principle.) Then it is not difficult to show that $\mathbb{N}$ is the "smallest" infinite set and is assigned the cardinality $\aleph_0$, as we have done.

We will prove that $\operatorname{card} \mathbb{R} > \aleph_0$ (but our proof is not accepted by everyone...). Then the question is whether

$$\operatorname{card} \mathbb{R} = \aleph_1?$$

Cardinalities of Infinite Sets

In other words, is there an infinite set with "more" elements than $\mathbb{N}$ but "fewer" elements than $\mathbb{R}$? Supposing that $\operatorname{card}\mathbb{R} = \aleph_1$ is known as the ***continuum hypothesis***.

The situation gets even more confusing: using commonly accepted axioms of set theory (the ZFC axioms introduced to eliminate the Russell paradox and the problems with naive set theory),

> *It has been proven that the continuum hypothesis can be neither proven nor disproven within the ZFC axioms, assuming that the ZFC axioms are consistent.*

Hence, a choice can be made: either $\operatorname{card}\mathbb{R} = \aleph_1$ or $\operatorname{card}\mathbb{R} > \aleph_1$. Either choice is acceptable and leads to a "consistent" branch of mathematics. Since human intuition fails to tell us which choice is "correct" and mathematics can (provably!) not provide an answer, the choice is essentially arbitrary.

Counting and Cardinality

2.4.4. Examples.

1. The set of bit strings of length 2, $M = \{00, 01, 10, 11\}$ is finite and has cardinality $\operatorname{card} M = 4$, because we can define the map $\varphi\colon M \to \{1, 2, 3, 4\}$ by

$$00 \mapsto 1, \qquad 01 \mapsto 2, \qquad 10 \mapsto 3, \qquad 11 \mapsto 4.$$

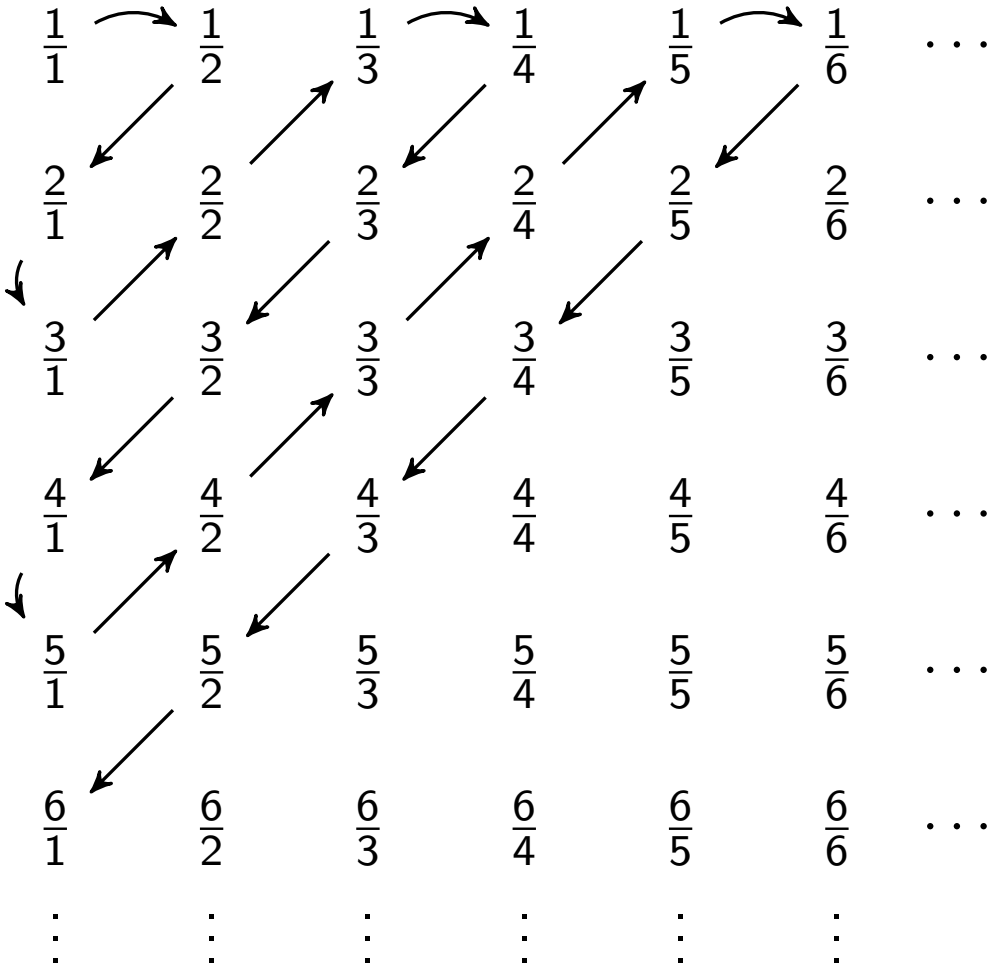
   It is clear that this map is bijective, so we have counted M. (We say that we have counted M when we have found a bijection φ.)

2. The set of even natural numbers $2\mathbb{N} = \{n \in \mathbb{N}\colon n = 2k,\ k \in \mathbb{N}\}$ is countably infinite; we simply define the bijection

$$\varphi\colon n \mapsto \frac{n}{2}.$$

Counting and Cardinality

2.4.5. Example. The positive rationals are countably infinite:



Uncountability of the Set of Bit Strings

Note that on the one hand there is a bijection $\mathbb{N} \to \Omega$, where $\Omega = \{p/q \in \mathbb{Q} : q = 1,\ p \in \mathbb{Z}\}$ is a strict subset of $\mathbb{Q}$, while on the other hand there is a bijection $\mathbb{N} \to \mathbb{Q}$. According to our definition of cardinality, the "number of elements" of Ω and $\mathbb{Q}$ are therefore equal. This contradicts the classical principle that "the size of the whole can not be equal to the size of a part".

We next give a famous result of Georg Cantor concerning the set T of bit strings of infinite length. We may regard such strings as infinite sequences whose entries take on only the values zero and one, i.e.,

$$T = \big\{(a_n) : \mathbb{N} \to \{0, 1\}\big\}.$$

2.4.6. Theorem. The set T has cardinality greater than $\aleph_0$, i.e., is uncountable.

Uncountability of the Set of Bit Strings

Proof.

The following method of proof is known as ***Cantor's diagonal argument***. Suppose that T is countable. Then we can find a sequence $(x_n)\colon \mathbb{N} \to \mathbb{C}$ and list the elements of this sequence, e.g.,

$$\begin{aligned} x_0 &= (0,0,0,0,0,0,0,\ldots),\\ x_1 &= (1,1,1,1,1,1,1,\ldots),\\ x_2 &= (0,1,0,1,0,1,0,\ldots),\\ x_3 &= (1,0,1,0,1,0,1,\ldots),\\ &\vdots \end{aligned}$$

We now define

$$y = (y_n) = (1,0,1,1,\ldots)$$

such that $y_n \neq (x_n)_n$. Since $y \neq x_n$ for all n, we arrive at a contradiction. □

Uncountability of the Set of Real Numbers

2.4.7. Corollary. The set $\mathbb{R}$ of real numbers is uncountable.

Proof.
We first show that $\operatorname{card}\mathbb{R} = \operatorname{card}(0,1)$, where $(0,1) \subset \mathbb{R}$ is the unit interval, by exhibiting a suitable bijection:

$$\varphi\colon (0,1) \to \mathbb{R}, \qquad \varphi(x) = \tan\left(\frac{(2x-1)\pi}{2}\right)$$

Next, we represent every number in the interval $[0,1]$ in base 2, so that its binary digits may be considered to be a bit string. There is a small technical difficulty at this point: two different bit strings can represent the same real number, e.g., $0.0111\cdots = 0.10000\ldots$. However, because there are only countably many such numbers (why?) they can be accounted for separately. We omit the details here. Then, since T is uncountable, so is $\mathbb{R}$. □

Cardinality of Power Sets

2.4.8. Theorem. Let M be a finite set with card $M = n$. Then the power set of M, $\mathscr{P}(M)$, has cardinality 2^n.

Proof.

Assume that the elements of M are numbered, writing $M = \{a_1, \dots, a_n\}$. We associate any element of $\mathscr{P}(M)$ with a bit string of length n as follows: an element $S \in \mathscr{P}(M)$ is a subset $S \subset M$. We associate to S the bit string of length n where the ith bit is 1 if $a_i \in S$ and 0 if $a_i \notin S$. Thus we obtain a bijective map from $\mathscr{P}(M)$ to the bit strings of length n.

Now a bit string of length n is the binary representation of a number between 0 and $2^n - 1$. We associate to each bit string this number plus 1, obtaining a bijection $\mathscr{P}(M) \to \{1, \dots, 2^n\}$. □

Cardinality of Power Sets

2.4.9. Remark. The same argument shows that

$$\operatorname{card} \mathscr{P}(\mathbb{N}) = \operatorname{card} T = \operatorname{card} \mathbb{R}.$$

More generally, a theorem due to Cantor states that

$$\operatorname{card} \mathscr{P}(A) > \operatorname{card} A$$

for any set A and hence

$$\operatorname{card} P(\mathbb{R}) > \operatorname{card} \mathbb{R}.$$

We conclude that the set of functions $f : \mathbb{R} \to \mathbb{R}$ has a larger cardinality than $\mathbb{R}$.

Cardinality of Cartesian Products

2.4.10. Theorem. Let M, N be finite sets with $\operatorname{card} M = m$, $\operatorname{card} N = n$. Then the cartesian product of M and N, $M \times N$, has cardinality $m \cdot n$.

Proof.

Suppose that $M = \{a_1, \dots, a_m\}$ and $N = \{b_1, \dots, b_n\}$. Then we define

$$\varphi \colon M \times N \to \{1, \dots, m \cdot n\}, \qquad (a_i, b_j) \mapsto (i-1)n + j.$$

The inverse map is

$$\varphi^{-1} \colon \{1, \dots, m \cdot n\} \to M \times N, \quad x \mapsto \left(a_{1+((x-1)\operatorname{div} n)}, b_{1+((x-1) \mod n)}\right),$$

and it is then easy to see that φ is bijective. □

Cardinality of Cartesian Products

2.4.11. Corollary. Let $M_1, \ldots, M_n$ be a finite number of finite sets with $\operatorname{card} M_k = m_k$. Then the cardinality of the cartesian product is

$$\operatorname{card}(M_1 \times M_2 \times \cdots \times M_n) = \prod_{i=1}^{n} m_i.$$

This follows from Theorem 2.4.10 by induction. We can use Corollary 2.4.11 to obtain an alternate proof for Theorem 2.4.8.

2.4.12. Example. There are 32 microcomputers in a computer center. Each microcomputer has 24 ports. How many ports are there in total?

Each port can be identified by the pair

$$(\text{computer no.}, \text{port no. on computer}).$$

Thus the set of all ports is the cartesian product of the set of all computers with the set of ports on each computer and there are $24 \cdot 32 = 768$ total ports.

Cardinality of Unions of Sets

2.4.13. Theorem. Let M, N be finite disjoint sets with $\operatorname{card} M = m$, $\operatorname{card} N = n$, $M \cap N = \emptyset$. Then the union of M and N, $M \cup N$, has cardinality $m + n$.

Proof.
Suppose that $M = \{a_1, \dots, a_m\}$ and $N = \{b_1, \dots, b_n\}$. Then we define

$$\varphi \colon M \cup N \to \{1, \dots, m+n\}, \qquad c \mapsto \begin{cases} i & \text{if } c = a_i, \\ m+j & \text{if } c = b_j. \end{cases}$$

The proof of bijectivity is left to the reader. □

2.4.14. Remark. Theorem 2.4.13 is the basis for the "sum rule" on page 338 of the textbook.

Cardinality of Unions of Sets

2.4.15. Corollary. Let $M_1, \dots, M_n$ be a finite number of disjoint finite sets with $\operatorname{card} M_k = m_k$. Then the cardinality of the union is

$$\operatorname{card}\Big(\bigcup_{i=1}^{n} M_i\Big) = \sum_{i=1}^{n} m_i.$$

2.4.16. Theorem. Let M, N be finite sets with $\operatorname{card} M = m$, $\operatorname{card} N = n$, $\operatorname{card}(M \cap N) = k$. Then the union of M and N, $M \cup N$, has cardinality $m + n - k$.

Proof.

Suppose that $M = \{a_1, \dots, a_m\}$ and $N = \{a_1, \dots, a_k, b_1, \dots, b_{n-k}\}$, where $k \geq 1$. Then

$$M \cup N = \{a_1, \dots, a_m, b_1, \dots, b_{n-k}\}$$

so $\operatorname{card}(M \cup N) = m + n - k$. □

Cardinality of Unions of Sets

Another way of writing Theorem 2.4.16 is

$$\operatorname{card}(M \cup N) = \operatorname{card} M + \operatorname{card} N - \operatorname{card}(M \cap N). \tag{2.4.2}$$

The previous theorems are applied in a wide variety of counting problems. We will just give a few examples here; many more can be found in the textbook and the exercises.

Number of Bit Strings with Specified Bits

2.4.17. Example. How many bit strings of length 8 start with a 1 bit or end with the two bits 00?

A bit string of length 8 is an octuple (8-tuple). There is a natural bijection from the set of octuples with first entry equal to 1 to the heptuples (7-tuples):

$$\phi\colon (1, x_1, x_2, \dots, x_7) \mapsto (x_1, x_2, \dots, x_7).$$

Since the set of heptuples is $\{0,1\}^7$, by Corollary 2.4.11 the cardinality of the set of bit strings starting with a 1-bit is 2^7. An analogous argument shows that the cardinality of the set of bit strings ending with 00 is 2^6. The problem asks for the union of the these two sets. The cardinality of the set of bit strings both starting with 1 and ending in 00 (the intersection of the previous two sets) is 2^5. Therefore,

$$2^7 + 2^6 - 2^5 = 160$$

bit strings start with a 1 bit or end with the two bits 00.

IPv4

Every computer or other device connected to the internet is assigned an ***internet address*** or ***IP address*** (IP stands for "internet protocol"). In the original system still used today, called IPv4, this address consists of 32 bits, or 4 bytes. Each byte (set of 8 bits) can be represented as a decimal number between 0 and 255, and in the ***dot-decimal*** notation, these numbers are separated by periods. For example, the JI's web server for the SAKAI system has the internet address 202.120.46.185.

The IP address is divided into several parts: the ***class*** followed (usually) by the ***netid*** and the ***hostid***. The bits in the IP address are numbered 0 through 31.

- For Class A network addresses, the first bit is 0, the next 7 bits give the netid, followed by a 24 bit hostid.
- For Class B network addresses, the first two bits are 10, the next 14 bits give the netid, followed by a 16 bit hostid.
- For Class C network addresses, the first three bits are 110, the next 21 bits give the netid, followed by a 8 bit hostid.

IPv4

- For Class D and E network addresses, the initial bits are 1110 and 11110, respectively, followed by private or multicast addresses. These classes are not used as general internet addresses.

2.4.18. Example. The SAKAI server's address 202.120.46.185 in binary representation is

$$11001010011110000010111010111001.$$

Since the first bits are 110, it is a Class C network address.

There are certain restrictions on the precise addresses permitted:

- 1111111 is not available as a netid for Class A networks.
- No hostids consisting only of 1s or 0s are available on any network.

2.4.19. Example. Under the above restrictions, what is the combined number of Class A, B and C addresses available?

IPv4

First, we calculate the number of Class A addresses. There are $2^7 - 1 = 255$ possible netids, and $2^{24} - 2 = 16\,777\,214$ possible hostids, giving a total of $255 \cdot 16\,777\,214 = 2\,130\,706\,178$ possible Class A addresses. In a similar manner, we calculate that there are $1\,073\,709056$ Class B and $532\,676\,608$ Class C addresses, giving a total of $3\,737\,091\,842$ addresses of Class A, B or C.

This number of addresses has proved too small, and several technical method have been implemented to resolve the problem. The long-term solution is to switch from the IPv4 system to a new 128-bit system called IPv6, which is slowly being implemented. More information can be found at HTTP://EN.WIKIPEDIA.ORG/WIKI/IPV4 and HTTP://EN.WIKIPEDIA.ORG/WIKI/IPV6.

The Pigeonhole Principle

2.4.20. Pigeonhole Principle. Let M be a finite set and $f: M \to M$ a map. Then f is surjective if and only if it is injective.

The name "pigeonhole principle" comes from the idea of a set of n pigeons landing in n pigeonholes. Numbering the pigeons $1, \ldots, n$ and the locations $1, \ldots, n$, the association of pigeon to pigeonhole is given by a map

$$f: \{1, \ldots, n\} \to \{1, \ldots, n\}.$$

Image source: http://en.wikipedia.org/wiki/File:TooManyPigeons.jpg. Used under the license given there

The Pigeonhole Principle states that one pigeon will land in each pigeonhole (f is surjective) if and only if no two pigeons land in the same location (f is injective).

The Pigeonhole Principle

The following theorem is another, equivalent formulation of the pigeonhole principle. We will prove it first, and then show that it implies the Pigeonhole Principle 2.4.20.

2.4.21. Theorem. Let M, N be finite sets and $f : M \to N$ surjective. Then f is injective if and only if $\operatorname{card} M = \operatorname{card} N$.

Proof.
Suppose $M = \{a_1, \dots, a_m\}$, $\operatorname{card} M = m$ and $\operatorname{card} N = n$. Since f is surjective, it follows that

$$n = \operatorname{card} N = \operatorname{card} \operatorname{ran} f = \operatorname{card}\{f(a_1), \dots, f(a_m)\} \leq m$$

If f is injective, then the numbers $f(a_1), \dots, f(a_m)$ are all distinct, and $\operatorname{card}\{f(a_1), \dots, f(a_m)\} = m$. Thus, $m = n$.

Suppose that $m = n$. Then $\operatorname{card}\{f(a_1), \dots, f(a_m)\} = m$ so $f(a_1), \dots, f(a_m)$ are distinct. This implies that f is injective. □

The Pigeonhole Principle

Proof of the Pigeonhole Principle 2.4.20.
Suppose that M is finite and $f : M \to M$ is injective. Then $f : M \to \operatorname{ran} f$ is surjective and injective. By Theorem 2.4.21, $\operatorname{card}(\operatorname{ran} f) = \operatorname{card} M$. Since $\operatorname{ran} f \subset M$, this implies (why?) that $\operatorname{ran} f = M$. Thus $f : M \to M$ is surjective.

Suppose that $f : M \to M$ is surjective. Then, by Theorem 2.4.21, f is injective. □

Yet another version of the pigeonhole theorem is the following:

2.4.22. Theorem. Let M, N be finite sets with $\operatorname{card} M > \operatorname{card} N$ and $f : M \to N$. Then f is not injective.

The proof is left to the exercises.

The pigeonhole principle seems almost trivial, but it is actually a very deep theorem about the structure of finite sets. It does not hold true for infinite sets, as the injective (but not surjective) map $f : \mathbb{N} \to \mathbb{N}$, $n \mapsto n + 1$ shows.

Applications of the Pigeonhole Principle

The pigeonhole principle can be used to prove a wide variety of theorems, some of which are quite surprising.

2.4.23. Lemma. Let $n \in \mathbb{N} \setminus \{0\}$. Then there exists a multiple of n such that its decimal expansion contains only the digits 1 and 0.

Proof.
Let $n \in \mathbb{N} \setminus \{0\}$ and consider the set of $n+1$ integers

$$M = \Big\{1,\ 11,\ 111, \dots, \underbrace{11\dots1}_{n \text{ times}},\ \underbrace{11\dots11}_{n+1 \text{ times}}\Big\}.$$

Define the map $f : M \to \{0, \dots, n-1\}$, $f(x) = x \bmod n$. By Theorem 2.4.22, f is not injective, so there exist two elements in M with the same remainder when divided by n. The larger of these less the smaller is an integer containing only 0s and 1s. Furthermore, it must have remainder 0 when divided by n, i.e., it is a multiple of n. □

Applications of the Pigeonhole Principle

2.4.24. Remark. In many practical applications, the pigeon hole theorem and related statements with respect to counting are formulated as "putting m objects into k boxes." It should be obvious that this is equivalent to finding a function from a set M of card $M = m$ objects to a set N of card $N = n$ boxes.

Given arbitrary sets X, Y and some map $f : X \to Y$ it is convenient to define the pre-image of a set $V \subset Y$ as

$$f^{-1}(V) := \{x \in X : f(x) \in V\}.$$

If V contains only one element $V = \{v\}$, we often omit the braces, writing $f^{-1}(v)$ instead of $f^{-1}(\{v\})$. For example, defining the function $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x, y) = x^2 + y^2$, the set

$$f^{-1}(r^2) = \{(x, y) \in \mathbb{R}^2 : f(x, y) = r^2\}$$

is a circle of radius r centered at the origin.

The Generalized Pigeonhole Principle

2.4.25. Generalized Pigeonhole Principle. Let M, N be finite sets with cardinalities $\operatorname{card} M = m$ and $\operatorname{card} N = n$, respectively. Then for any function $f : M \to N$ there is at least one $n_0 \in N$ such that $\operatorname{card}(f^{-1}(n_0)) \geq \lceil m/n \rceil$.

Proof.

We prove the theorem by contradiction. Suppose that for all $y \in \operatorname{ran} f$

$$\operatorname{card}(f^{-1}(y)) < \lceil m/n \rceil.$$

Then also $\operatorname{card}(f^{-1}(y)) < m/n$, since $\operatorname{card}(f^{-1}(y)) \in \mathbb{N}$. Using $f^{-1}(y) \cap f^{-1}(y') = \emptyset$ for $y \neq y'$,

$$\operatorname{card} M = \operatorname{card}\Big(\bigcup_{y \in \operatorname{ran} f} f^{-1}(y) \Big) = \sum_{y \in \operatorname{ran} f} \operatorname{card}(f^{-1}(y)) < n \frac{m}{n} = m,$$

which is a contradiction. □

Ramsey Theory

We can apply the generalized pigeonhole theorem to a problem that occurs in ***Ramsey theory***. In general, Ramsey theory is concerned with the number of elements in a set such that a given property will hold. A particular example is the following problem:

2.4.26. Example. At a party consisting of 6 people, any two people are either friends or enemies. Then there are either three mutual friends or three mutual enemies at the party.

Consider one member of the party, A. The five other members of the party are either friends or enemies of A. By the generalized pigeonhole principle, at least three of them are either friends or enemies of A. Suppose that B, C, D are friends of A. If any two of these three are friends, then they together with A are a group of three mutual friends. If none of B, C, D are friends, then they form a group of three mutual enemies.

If B, C, D are a group of three enemies of A, the proof proceeds in a similar manner.

Ramsey Numbers

For $m, n \in \mathbb{N} \setminus \{0, 1\}$ the ***Ramsey number*** $R(m, n)$ is defined as the minimum number of people at a party so that there are either m mutual friends or n mutual enemies. By Example 2.4.26,

$$R(3, 3) \leq 6.$$

In the assignments you will prove $R(3, 3) > 5$, so $R(3, 3) = 6$.

While a few interesting properties of Ramsey numbers are known, e.g., $R(m, n) = R(n, m)$ and $R(2, n) = n$, only a few Ramsey numbers are actually known. In fact, only nine Ramsey numbers with $3 \leq m \leq n$ have been calculated! For example, $R(4, 4) = 18$. For other numbers, only bounds are known, e.g., $43 \leq R(5, 5) \leq 49$.

Permutations

2.4.27. Definition. Let $\{x_1, \dots, x_n\}$ be a set of n distinguishable elements ($x_k \neq x_j$ for $j \neq k$). Then an injective map

$$\pi : \{x_1, \dots, x_n\} \to \{x_1, \dots, x_n\}$$

is called a ***permutation*** of these elements.

By the pigeonhole principle, any permutation is automatically bijective.

2.4.28. Remark. A permutation is defined on a set of n distinguishable elements; instead of $\{x_1, \dots, x_n\}$ we can also simply write $\{1, \dots, n\}$, replacing the permutation of elements with a permutation of indices.

Recall that a function f is defined as a set of pairs of the form $(x, f(x))$, where x is the independent variable. Thus we could define a permutation π through a set of pairs $\{(1, \pi(1)), \dots, (n, \pi(n))\}$. In fact, we do represent permutations in this way, but use a different notation, writing

$$\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix}$$

Permutations

2.4.29. Example. Consider the set of 3 objects, $\{1, 2, 3\}$. Then one permutation is given by the identity map,

$$\pi_0 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

while other permutations are given by

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}.$$

2.4.30. Remark. Instead of writing out the map as above, we often simply give the ordered n-tuple $(\pi(1), \ldots, \pi(n))$. For example, instead of

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \qquad \text{we might write} \qquad (3, 1, 2).$$

Permutations

2.4.31. Remark. A permutation can be regarded as an ***arrangement*** of a set; using the notation of Remark 2.4.30, the tuple $(3, 1, 2)$ can be considered on the one hand as the values of a permutation; on the other hand, it represents a specific ordering (or arrangement) of the elements of the set $\{1, 2, 3\}$. In practice, the interpretation as an ordering is used in applications.

Permutations

2.4.32. Lemma. There are $n!$ permutations π of the set $\{1, \dots, n\}$.

Proof.
We consider the number of possible values of $\pi(k)$, $k = 1, \dots, n$. The first value, $\pi(1)$ can be any of the n elements of $\{1, \dots, n\}$. The second value can be any element of $\{1, \dots, n\} \setminus \{\pi(1)\}$. Hence there are n possible values of $\pi(1)$, but only $n - 1$ possible values of $\pi(2)$. In general, there are $n - k + 1$ possible values for $\pi(k)$, so in total, there are

$$n \cdot (n-1) \cdots (n-n+1) = n!$$

different choices for the values $(\pi(1), \dots, \pi(n))$ of a permutation. □

Permutations as Arrangements

In combinatorics, permutations are regarded as ***arrangements in a definite order***. Clearly, the ordered tuple $(\pi(1), \dots, \pi(n))$ is an arrangement of $(1, \dots, n)$ is a definite order, i.e., the definition based on permutations realizes this goal.

Often one is interested in first selecting $r \leq n$ objects from $\{1, \dots, n\}$ and then arranging these objects in a definite order. We realize this by the following, more general definition:

2.4.33. Definition. Let $n, r \in \mathbb{N} \setminus \{0\}$ with $r \leq n$. Then an injective map

$$\pi \colon \{1, \dots, r\} \to \{1, \dots, n\}$$

is called an ***r-permutation*** of r elements from $\{1, \dots, n\}$.

Permutations as Arrangements

2.4.34. Notation. We again write π as

$$\begin{pmatrix} 1 & 2 & \dots & r \\ \pi(1) & \pi(2) & \dots & \pi(r) \end{pmatrix} \qquad \text{or} \qquad (\pi(1), \dots, \pi(r)),$$

where $\pi(k) \in \{1, \dots, n\}$, $k = 1, \dots, r$.

2.4.35. Example. Let $n = 3$, $r = 2$. Then the following are permutations of two elements from $\{1, 2, 3\}$:

$$(1, 2), \qquad (2, 1), \qquad (1, 3), \qquad (3, 1), \qquad (2, 3), \qquad (3, 2).$$

2.4.36. Theorem. There are

$$n \cdot (n-1) \cdots (n-r+1) = \frac{n!}{(n-r)!}$$

different permutations of r elements from $\{1, \dots, n\}$.

Combinations

A related question arising in combinatorics is the number of ***selections*** of r elements from a set of n elements, where we do not care about the order. Such selections are called ***r-combinations***.

2.4.37. Example. Let $n = 3$, $r = 2$. Then the following are combinations of two elements from $\{1, 2, 3\}$:

$$\{1, 2\}, \qquad \{1, 3\}, \qquad \{2, 3\}.$$

Note that a selection consists of sets (which are unordered) while a permutation consists of ordered tuples.

2.4.38. Definition. A combination of r elements from $\{1, \dots, n\}$ is subset $A \subset \{1, \dots, n\}$ with $\operatorname{card} A = r$ elements.

Combinations

2.4.39. Theorem. There are

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

combinations of r elements from $\{1, \ldots, n\}$.

Proof.

We first consider permutations of r objects from $\{1, \ldots, n\}$. Every permutation gives us a combination, by identifying

$$(\pi(1), \ldots, \pi(r)) \longrightarrow \{\pi(1), \ldots, \pi(r)\}.$$

Obviously, more than one permutation will give us the same combination. Note that any permutation of $(\pi(1), \ldots, \pi(r))$ will be a permutation of r objects from $\{1, \ldots, n\}$. Furthermore, any permutation of $(\pi(1), \ldots, \pi(r))$ will yield the same combination. For each tuple $(\pi(1), \ldots, \pi(r))$, there are $r!$ permutations of $(\pi(1), \ldots, \pi(r))$, so we need to divide the total number of permutations of r objects from $\{1, \ldots, n\}$ by $r!$. □

Combinations as Permutations of Indistinguishable Objects

A combination of r elements of $\{1, \ldots, n\}$ is similar to a permutation, but without regard to order. Fundamentally, if we do not order the selected objects, we regard them as indistinguishable once they have been selected. (We can not say "Take the fifth selected object," but merely, "Take a selected object." We do not distinguish between the selected objects.)

Therefore, a combination can be regarded as ***a permutation of*** r ***indistinguishable objects from*** $\{1, \ldots, n\}$***.***

Effectively, this corresponds to a division of $\{1, \ldots, n\}$ into two classes, a subset consisting of r elements and its complement.

Sorting Into Classes

We can hence reformulate Theorem 2.4.39 on combinations with regard to the problem of sorting elements of $\{1, \dots, n\}$ into two classes (sets) A_1 and A_2 of specified size.

2.4.40. Theorem. Let $\mathscr{N} := \{1, \dots, n\}$. Then there are

$$\frac{\operatorname{card} \mathscr{N}!}{\operatorname{card} A_1! \operatorname{card} A_2!} = \frac{n!}{n_1! n_2!} = \frac{n!}{r!(n-r)!} = \binom{n}{r}$$

possible ways of dividing $\mathscr{N}$ into two sets A_1 and A_2, where

- $A_1 \subset \mathscr{N}$, $\operatorname{card} A_1 = n_1 = r$,
- $A_2 = \mathscr{N} \setminus A_1$, $\operatorname{card} A_2 = n_2 = n - r$.

Of course, this result can be generalized!

Sorting Into Classes

2.4.41. Theorem. Let $\mathscr{N} := \{1, \dots, n\}$. Then there are

$$\frac{n!}{n_1! n_2! \dots n_k!}$$

possible ways of dividing $\mathscr{N}$ into k sets $A_1, \dots, A_k$, where

- $\mathscr{N} = \bigcup_{i=1}^{k} A_i, \quad A_i \cap A_j = \emptyset$ for $i \neq j$,
- card $A_i = n_i$, $i = 1, \dots, k$.

2.4.42. Remark. Since we are only interested in dividing a number of elements into classes, and do not distinguish between the elements within each class, this sorting into classes is also called ***permutation of indistinguishable objects***.

Sorting Into Classes

Proof of Theorem 2.4.41.

We first count the number of ways of sorting n_1 elements into A_1:

$$\binom{n}{n_1} = \frac{n!}{n_1!(n-n_1)!}.$$

Next we count the ways of sorting n_2 elements of the remaining $n - n_1$ elements into A_2:

$$\binom{n-n_1}{n_2} = \frac{(n-n_1)!}{n_2!(n-n_1-n_2)!}.$$

The total number of ways of sorting n_1 elements into A_1 and n_2 elements into A_2 is then the product

$$\binom{n}{n_1}\binom{n-n_1}{n_2} = \frac{n!}{n_1!(n-n_1)!}\frac{(n-n_1)!}{n_2!(n-n_1-n_2)!} = \frac{n!}{n_1!n_2!(n-n_1-n_2)!}$$

Sorting Into Classes

Proof of Theorem 2.4.41 (continued).

Proceeding in this manner, we find there are

$$\begin{aligned}
\binom{n}{n_1}\prod_{i=1}^{k-1}\binom{n-\sum_{j=1}^{i}n_j}{n_{i+1}} &= \frac{n!}{n_1!(n-n_1)!}\prod_{i=1}^{k-1}\frac{\left(n-\sum_{j=1}^{i}n_j\right)!}{n_{i+1}!\left(n-\sum_{j=1}^{i+1}n_j\right)!} \\
&= \frac{n!}{n_1!(n-n_1)!}\frac{\prod_{i=1}^{k-1}\left(n-\sum_{j=1}^{i}n_j\right)!}{\prod_{i=1}^{k-1}n_{i+1}!\prod_{i=1}^{k-1}\left(n-\sum_{j=1}^{i+1}n_j\right)!} \\
&= \frac{n!}{\prod_{i=1}^{k}n_i!}\frac{\prod_{i=2}^{k-1}\left(n-\sum_{j=1}^{i}n_j\right)!}{\prod_{i=1}^{k-2}\left(n-\sum_{j=1}^{i+1}n_j\right)!} \\
&= \frac{n!}{\prod_{i=1}^{k}n_i!}
\end{aligned}$$

ways of sorting all objects into classes $A_1, \ldots, A_k$, $\operatorname{card} A_i = n_i$. □

Repetition

Recall that r-permutations are arrangements of r objects from a set of $n \geq r$ objects, represented through r-tuples. Similarly, r-combinations are selections of r objects but without regard to order, hence represented as sets of cardinality r. These two problems have one thing in common, which we only mentioned in passing: the r selected/arranged objects are all ***distinct***.

What happens if we drop this requirement, i.e., we allow identical objects to appear in our selections/arrangements? This is known as selection/arrangement ***with repetition***.

In the case of arrangements with repetition, this is quite easy: in an r-tuple, we now allow any entry to contain an arbitrary element of our set $\{1, \ldots, n\}$, regardless of what is contained in the other entries. Thus, the set of all ***r-permutations with repetition*** is just $\{1, \ldots, n\}^r$ and has cardinality n^r by Theorem 2.4.10. There are n^r different r-permutations with repetition.

Repetition and Multisets

Dealing with combinations with repetition is a bit more subtle: A 2-combination is, for example, the set $\{a, b\}$, but a 3-combination with repetition might be $\{a, b, b\} = \{a, b\}$. So the "information" that the element b is to be repeated (or selected twice) is lost if we just use set formalism.

A solution is to introduce ***multisets***. These are unordered collections of objects (like sets) but also associate to each object a so-called ***multiplicity***. For example, the set

$$\{2a, 4b, c\}$$

contains the elements a, b, c with multiplicities $2, 4, 1$, respectively. The multiplicities are generally integers and keep track of "how many times" an element is included in the set. Multisets are related to fuzzy sets (that you encountered in the exercises) and follow a similar calculus. We will define the cardinality of a multiset as the sum of the multiplicities of its elements. Then an ***r-combinations with repetition*** is simply a multiset of cardinality r, or an r-multiset.

Combinations and Permutations with Repetition

Note that in allowing r-permutations and r-combinations of n elements with repetition, we no longer require $r \leq n$.

2.4.43. Example. Consider the set $S = \{a, b, c\}$. Then

$$(b, a, b, c, a, b, b)$$

is a 7-permutation with repetition of S. The 7-multiset

$$\{2a, 4b, c\}$$

represents the corresponding 7-combination of S.

We have already discussed that there are $n^r = 3^7 = 2187$ possible 7-permutations with repetition. But how many different 7-combinations are there?

Counting Multisets

Let us stay with the special case of Example 2.4.43. It is clear that the number of possible 7-combinations is equal to the number of multisets of cardinality 7. Suppose that we write

$$\{2a, 4b, c\} = \{a, a, b, b, b, b, c\}.$$

Now if we agree to always list the elements of a multiset containing elements of $S = \{a, b, c\}$ in a definite order, we need only write

$$x, x, |, x, x, x, x, |, x$$

to indicate the set $\{2a, 4b, c\}$. By our convention, the first xs will stand for as until a bar $|$ is reached, then the xs will stand for bs etc. We can also leave out the commas. Thus,

$|xxx|xxxx$ stands for the 7-multiset $\{3b, 4c\}$.

Counting Multisets

We see that the multisets are determined entirely by the position of the bars among the xs. In our example, we need 2 bars to indicate the separation between a, b, c and have 7 xs to indicate the elements of the 7-multiset. We can regard this as follows: we have 9 "slots" and each slot is either filled by a bar or an x. Since the xs by themselves can be any element of $S = \{a, b, c\}$, the number of multisets is determined by number of ways to select the two slots that hold the bars. In our example, there are $\binom{9}{2} = 36$ ways to select these slots.

This argument immediately generalizes to yield the following theorem:

2.4.44. Theorem. For $r, n \in \mathbb{N} \setminus \{0\}$ here are

$$\left(\begin{matrix} r+n-1 \\ n-1 \end{matrix}\right) = \left(\begin{matrix} r+n-1 \\ r \end{matrix}\right)$$

r-combinations of n elements with repetition. (Or that many r-multisets of n elements.)

Counting Multisets

2.4.45. Example. Consider the equation

$$x_1 + x_2 + x_3 = 11, \qquad x_1, x_2, x_3 \in \mathbb{N}.$$

We are interested in finding the number of solutions to this equation. Note that this equation is equivalent to the following: select a total of 11 items, where x_1 items are of type 1, x_2 are of type 2 and x_3 are of type 3. Thus, each solution corresponds to an 11-multiset of 3 three elements and there are

$$\binom{11+3-1}{11} = 78$$

different solutions.

The Inclusion-Exclusion Principle

In Theorem 2.4.16 we have proven that for two finite sets A, B,

$$\operatorname{card}(A \cup B) = \operatorname{card} A + \operatorname{card} B - \operatorname{card}(A \cap B). \tag{2.4.3}$$

For the following arguments we will write $|A|$ for $\operatorname{card} A$ for brevity. It turns out that (2.4.3) can be generalized; for instance,

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|.$$

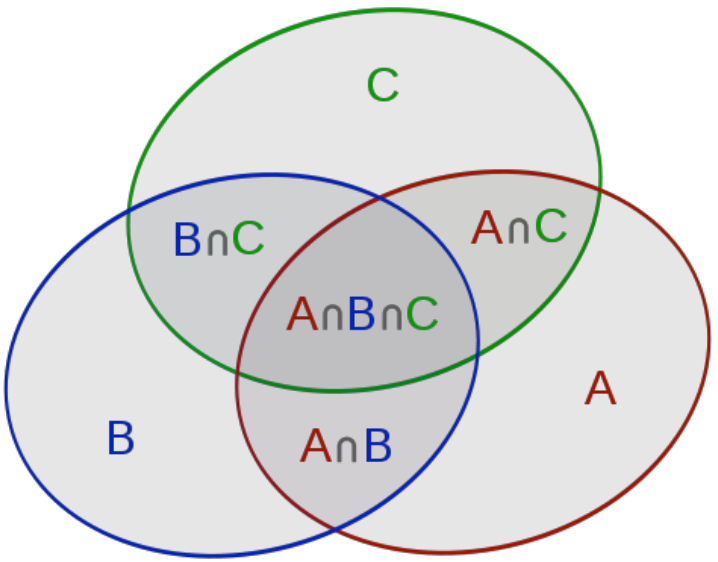


Image source: http://commons.wikimedia.org/wiki/File:Inclusion-exclusion.svg Used under the license given there

The Inclusion-Exclusion Principle

The general theorem which governs the cardinality of the union of n sets is called the ***Inclusion-Exclusion Principle***

2.4.46. Inclusion-Exclusion Principle. For any sets $A_1, \ldots, A_n$, $n \in \mathbb{N}$,

$$|A_1 \cup A_2 \cup \ldots \cup A_n| = \sum_{1 \le i \le n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - + \ldots + (-1)^{n+1} |A_1 \cap A_2 \cap \ldots \cap A_n| \tag{2.4.4}$$

The Inclusion-Exclusion Principle

Proof.

We will show that the right-hand side of (2.4.4) counts every element of the union $\bigcup A_i$ exactly once. Suppose that an element a is contained in exactly r of the sets $A_1, \ldots, A_n$, $1 \leq r \leq n$. Then the first sum counts a exactly $r = \binom{r}{1}$ times. The second sum counts a $\binom{r}{2}$ times, since there are $\binom{r}{2}$ pairs of sets which both contain a. In general, a sum involving the intersection of m sets will count a $\binom{r}{m}$ times.

Therefore, the number of times that the right-hand side (2.4.4) counts a is

$$\begin{aligned}\binom{r}{1} - \binom{r}{2} + - \cdots + (-1)^{r+1}\binom{r}{r} &= 1 - \sum_{m=0}^{r}(-1)^m\binom{r}{m} \\ &= 1 - (1-1)^r = 1\end{aligned}$$

□

Inclusion-Exclusion (Alternative Form)

Suppose that S is a finite set and $A_1, \ldots, A_n \subset S$. Then

$$A_1 \cap A_2 \cap \cdots \cap A_n = S \setminus (A_1^c \cup A_2^c \cup \cdots \cup A_n^c)$$

and, therefore,

$$\begin{aligned} |A_1 \cap A_2 \cap \ldots \cap A_n| &= |S| - |A_1^c \cup A_2^c \cup \ldots \cup A_n^c| \\ &= |S| - \sum_{1 \le i \le n} |A_i^c| + \sum_{1 \le i < j \le n} |A_i^c \cap A_j^c| \\ &\quad - + \ldots + (-1)^n |A_1^c \cap A_2^c \cap \ldots \cap A_n^c|. \end{aligned} \tag{2.4.5}$$

We can apply (2.4.5) to certain counting problems.

2.4.47. Example. We are interested in the number of primes less than or equal to n for a given $n \in \mathbb{N} \setminus \{0, 1\}$. By Theorem 1.7.7, to check whether a number p is prime, it suffices to check whether it is a multiple of any number less than $\sqrt{p}$.

Counting Prime Numbers

Suppose we are interested in the number of primes less than $n = 120$. Then we need to eliminate from $S = \{2, 3, \dots, 120\}$ all composite numbers, i.e., all numbers with prime factors less than $\lfloor \sqrt{120} \rfloor = 10$. These are the numbers $2, 3, 5, 7$, so let

$$A_1 := \{q \in S : 2 \mid q\}, \qquad A_2 := \{q \in S : 3 \mid q\},$$
$$A_3 := \{q \in S : 5 \mid q\}, \qquad A_4 := \{q \in S : 7 \mid q\}.$$

Then a prime number in S will be equal to 2, 3, 5 or 7 or an element of $A_1^c \cap A_2^c \cap A_3^c \cap A_4^c$. We apply (2.4.5):

$$|A_1^c \cap A_2^c \cap A_3^c \cap A_4^c| = |S| - \sum_{i=1}^{4} |A_i| + \sum_{1 \le i < j \le 4} |A_i \cap A_j| - \sum_{1 \le i < j < k \le 4} |A_i \cap A_j \cap A_k| + |A_1 \cap A_2 \cap A_3 \cap A_4| \tag{2.4.6}$$

Counting Prime Numbers

We use that

$$|\{q \in S \colon n \mid q\}| = 120 \operatorname{div} n = \lfloor 120/n \rfloor$$

Then

$$|S| = 119, \qquad |A_1| = 60, \qquad |A_2| = 40, \qquad |A_3| = 24, \qquad |A_4| = 17.$$

Furthermore,

$$\begin{aligned}
A_1 \cap A_2 &= \{q \in S \colon 6 \mid q\}, & |A_1 \cap A_2| &= 20,\\
A_1 \cap A_3 &= \{q \in S \colon 10 \mid q\}, & |A_1 \cap A_3| &= 12,\\
A_1 \cap A_4 &= \{q \in S \colon 14 \mid q\}, & |A_1 \cap A_4| &= 8,\\
A_2 \cap A_3 &= \{q \in S \colon 15 \mid q\}, & |A_2 \cap A_3| &= 8,\\
A_2 \cap A_4 &= \{q \in S \colon 21 \mid q\}, & |A_2 \cap A_4| &= 5,\\
A_3 \cap A_4 &= \{q \in S \colon 35 \mid q\}, & |A_3 \cap A_4| &= 3,
\end{aligned}$$

Counting Prime Numbers

$$A_1 \cap A_2 \cap A_3 = \{q \in S : 30 \mid q\}, \qquad |A_1 \cap A_2 \cap A_3| = 4,$$
$$A_1 \cap A_2 \cap A_4 = \{q \in S : 42 \mid q\}, \qquad |A_1 \cap A_2 \cap A_4| = 2,$$
$$A_1 \cap A_3 \cap A_4 = \{q \in S : 70 \mid q\}, \qquad |A_1 \cap A_3 \cap A_4| = 1,$$
$$A_2 \cap A_3 \cap A_4 = \{q \in S : 105 \mid q\}, \qquad |A_2 \cap A_3 \cap A_4| = 1$$

and, finally,

$$A_1 \cap A_2 \cap A_3 \cap A_4 = \{q \in S : 210 \mid q\}, \qquad |A_1 \cap A_2 \cap A_3 \cap A_4| = 0.$$

Inserting these values into (2.4.6), we obtain

$$\begin{aligned} |A_1^c \cap A_2^c \cap A_3^c \cap A_4^c| &= 119 - 60 - 40 - 24 - 17 \\ &\quad + 20 + 12 + 8 + 8 + 5 + 3 - 4 - 2 - 1 - 1 + 0 \\ &= 26 \end{aligned}$$

Hence, there are $4 + 26 = 30$ primes less than or equal to 120. We have found these primes through the Sieve of Eratosthenes in Example 1.7.9.

The Prime Number Theorem

More generally, the number of prime numbers less than a given n is of significant interest. The following result is quite difficult to prove and we only state it here for reference:

2.4.48. Prime Number Theorem. Let $\pi(n)$ denote the number of prime numbers less than or equal to $n \in \mathbb{N}$. Then

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln(n)} = 1,$$

where ln denotes the natural logarithm (to base $e = 2.71828\ldots$).

Classical Definition of Probability

In this section we will introduce some methods based on the simple, classical concept of probability.

2.5.1. Definition. Let A be a random outcome of an experiment that may proceed in various ways. Assume each of these ways is equally likely. Then the probability of the outcome A, denoted $P[A]$, is

$$P[A] = \frac{\text{number of ways leading to outcome } A}{\text{number of ways the experiment can proceed.}}$$

2.5.2. Examples.

(i) The experiment consists of flipping a coin. We are interested in the probability of the coin landing heads up. The experiment can proceed in two ways: the coin land heads up or tails up. We assume each event is equally likely, so the classical definition gives

$$P[\text{coin lands heads up}] = \frac{1}{2} = 0.5$$

Classical Definition of Probability

(ii) The experiment consists of rolling two 6-sided dice and summing the results, so the possible outcomes are the numbers $S = 2, 3, \ldots, 12$. We are interested in the outcome $S = 3$. Each die will give results 1, 2, 3, 4, 5 or 6. We assume that each result is equally likely. There are two ways we can get the outcome $S = 3$: either the first die's result is 1 and the second die's result is 2, or the first die gives 2 and the second die gives 1. In total, the experiment can proceed in $6 \times 6 = 36$ different ways. Hence

$$P[3] = \frac{2}{36} = \frac{1}{18} = 0.056$$

Sample Spaces and Sample Points

The techniques of counting permutations, combinations and sorting into classes are very useful for evaluating probabilities by the classical definition. First, however, we need to translate physical situations into a mathematical context.

2.5.3. Definition. A ***sample space*** for an experiment is a set S such that each physical outcome of the experiment corresponds to exactly one element of S.

An element of S is called a sample point.

2.5.4. Remark. Not every element of a sample space needs to correspond to the outcome of an experiment. It is often convenient to select a very large, but simple sample space. For example, for the roll of a six-sided die, we can take $S = \mathbb{N}$, where the result of the die corresponds to the natural numbers 1,2,3,4,5 or 6 and all other natural numbers are not used. All natural numbers in this example are sample points, even though only six actually correspond to physical reality.

Events

2.5.5. Definition. Any subset A of a sample space S is called an ***event***.

2.5.6. Example. We roll a four-sided die 10 times. Then a possible sample space is $S = \mathbb{N}^{10}$ and a sample point is a 10-tuple, for example $(1, 2, 3, 2, 3, 3, 1, 1, 4, 4) \in \mathbb{N}^{10}$.

This sample point corresponds to first rolling a one, then a 2, next a 3, followed by a 2, a 3, a 3, two ones and two fours.

An event might correspond to "rolling at least two fours" in which case this would be a subset $A \subset S$ such that each 10-tuple in A has at least two entries equal to 4. For example, $(1, 2, 3, 2, 3, 3, 1, 1, 4, 4) \in A$ but $(1, 2, 3, 2, 3, 3, 1, 1, 3, 4) \notin A$.

Image source:
http://commons.wikimedia.org/wiki/File:4-sided_dice_250.jpg.
Used under the license given there

Example

2.5.7. Example. We roll a four-sided die 10 times. What is the probability of obtaining 5 ones, 3 twos, 1 three and 1 four?

There are $4^{10} = 1048576$ possibilities for the 10-tuple of results of the die rolls, corresponding to that many sample points in $S = \mathbb{N}^{10}$ that correspond to physical results. The event A consists of all ordered 10-tuples containing 5 ones, 3 twos, 1 three and 1 four. There are

$$\frac{10!}{5!3!1!1!} = 5040$$

possible ways of obtaining 5 ones, 3 twos, 1 three and 1 four, so there are that many elements in A. The probability is

$$\frac{5040}{1048576} \approx 0.00481 \approx 0.5\%.$$

Properties of Events

We can interpret events as follows: Suppose S is a sample space and $A_1, A_2 \subset S$.

(i) The event $A_1 \cap A_2$ contains all sample points corresponding to outcomes common to A_1 and A_2.

A_1, A_2 are said to be ***mutually exclusive*** if $A_1 \cap A_2 = \emptyset$.

(ii) The event $A_1 \cup A_2$ contains all sample points corresponding to outcomes in either A_1 or A_2.

If our sample space S contains a finite number of sample points (i.e., $\operatorname{card} S < \infty$) and $P[A_1]$ and $P[A_2]$ are given, we can calculate $P[A_1 \cap A_2]$ and $P[A_1 \cup A_2]$.

However, some sample spaces are countably or even uncountably infinite in size. It is then easier to simply ***define*** the properties that a "probability" is supposed to have in such a way that the definition coincides with the results obtained by counting whenever S is finite.

Definition of Probability

2.5.8. Definition. Let S be a sample space and $\mathscr{P}(S)$ the power set of S. Then a function

$$P : \mathscr{P}(S) \to \mathbb{R}, \qquad A \mapsto P[A]$$

is called a ***probability function*** (or just ***probability***) on S if

(i) $P[A] \geq 0$ for all $A \subset S$,

(ii) $P[S] = 1$,

(iii) For any set of events $\{A_k\} \subset \mathscr{P}(A)$ such that $A_i \cap A_k = \emptyset$ for $i \neq k$,

$$P\left[\bigcup A_k\right] = \sum P[A_k].$$

This equation holds also if $\{A_k\}$ is an infinite set of A_k, in which case the union and the sum are also infinite.

Probabilities of Events

A subset $A \in \mathscr{P}(S)$ is an event in sample space. We therefore interpret events as follows:

- $A = \emptyset$ is the event that "nothing happens" (an experiment has no outcome found in S)
- $A = S$ is the event that "something happens" (the experiment has some outcome in S)
- $A^c = S \setminus A$ represents all outcomes that do not include those associated to A

By Definition 2.5.8,

$$P[S] = 1, \qquad P[\emptyset] = 0, \qquad P[A^c] = 1 - P[A],$$

where the first equation is given by definition and the latter two follow easily from the properties of the probability function.

It is also not difficult to show that

$$P[A_1 \cup A_2] = P[A_1] + P[A_2] - P[A_1 \cap A_2], \quad A_1, A_2 \in \mathscr{P}(S). \qquad (2.5.1)$$

Probabilities of Events

Note that (2.5.1) implies ***Boole's inequality***,

$$P[A_1 \cup A_2] \leq P[A_1] + P[A_2], \tag{2.5.2}$$

for all $A_1, A_2 \in \mathscr{P}(S)$.

Another easy property is that if $A_1 \subset A_2$, then $P[A_1] \leq P[A_2]$.

2.5.9. Example. Suppose that two four-sided dice are rolled. We take the sample space

$$S = \{(1,1), (1,2), (1,3), (1,4), (2,1), \ldots, (4,3), (4,4)\},$$

where each tuple has the form (1^{st} result, 2^{nd} result). We then assign the probability function $P[\{(i,j)\}] = 1/16$ for $i, j = 1, 2, 3, 4$. By defining the probability for each one-element subset of S we are fixing the probability for any event $A \subset S$. Our definition has been chosen to coincide with the probability of obtaining each sample point according to Definition 2.5.1 if the dice are fair.

Probabilities of Events

Let A_1 be the event that corresponds to the outcome "the sum of the two die rolls is at most 3" and A_2 correspond to the outcome "the two die rolls give the same number". Then

$$A_1 = \{(1,1),(1,2),(2,1)\}, \qquad A_2 = \{(1,1),(2,2),(3,3),(4,4)\}.$$

The probability of these events is calculated as

$$P[A_1] = P[\{(1,1)\}] + P[\{(1,2)\}] + P[\{(2,1)\}] = \frac{3}{16},$$

$$P[A_2] = \frac{4}{16} = \frac{1}{4}.$$

Then we can calculate for example that the probability of "the sum of two die rolls is at most three and both rolls are the same" is

$$P[A_1 \cap A_2] = P[\{1,1\}] = \frac{1}{16},$$

the probability of "the two die rolls are distinct" is

$$P[A_2^c] = 1 - P[A_2] = \frac{3}{4}.$$

The Probabilistic Inclusion-Exclusion Principle

We note that the generalization of (2.5.1) is given by the inclusion-exclusion principle,

2.5.10. Inclusion-Exclusion Principle. Let S be a sample space and $A_1, \ldots, A_n \subset S$. Then

$$P[A_1 \cup A_2 \cup \cdots \cup A_n] = \sum_{1 \le i \le n} P(A_i) - \sum_{1 \le i < j \le n} P[A_i \cap A_j] + \sum_{1 \le i < j < k \le n} P[A_i \cap A_j \cap A_k] - + \ldots + (-1)^{n+1} P[A_1 \cap A_2 \cap \cdots \cap A_n] \tag{2.5.3}$$

Since S need not be a countable set, this version of the theorem is best proven using induction in n. We omit the proof here.

Conditional Probability

We have seen from this example how the relations derived from the axioms of Definition 2.5.8 allow us to calculate the probability that

- "event A occurs",
- "event A does not occur",
- "events A and B occur" and
- "event A or event B occurs".

The axioms do not, however, provide us with a way to calculate the probability that "event B occurs if event A has occurred." For this, we need to make an additional definition, which we now introduce.

Let us denote by $P[B \mid A]$ the probability that "B occurs given that A has occurred". Let us use Example 2.5.9 for illustration. Suppose we are interested in $P[A_1 \mid A_2]$, i.e., the probability that the sum of the die rolls is less than four given that the rolls are equal. Effectively, 4 out of 16 outcomes lead to event A_2, and only one out these 4 gives rise to an element in $A_1 \cap A_2 = \{(1, 1)\}$.

Conditional Probability

If we count the number of sample points ("outcomes") in A_2 that give $A_1 \cap A_2$ to obtain the probability of obtaining A_1 given that A_2 has occurred, we essentially have

$$P[A_1 \mid A_2] = \frac{\operatorname{card}(A_1 \cap A_2)}{\operatorname{card} A_2} = \frac{P[A_1 \cap A_2]}{P[A_2]}.$$

This motivates the following definition:

2.5.11. Definition. Let $A, B \subset S$ be events and $P[A] \neq 0$. Then

$$P[B \mid A] := \frac{P[A \cap B]}{P[A]}.$$

is said to be the ***conditional probability*** of B given A.

Independence of Events

If one event does not influence another, then we say that the two events are ***independent***. Mathematically, we express this in the following way.

2.5.12. Definition. Let A, B be two events. We say that A and B are independent if

$$P[A \cap B] = P[A]P[B]. \tag{2.5.4}$$

Equation (2.5.4) is equivalent to

$$P[A \mid B] = P[A] \quad \text{if } P[B] \neq 0,$$
$$P[B \mid A] = P[B] \quad \text{if } P[A] \neq 0.$$

The Matching Problem

We illustrate these principles by considering the classic ***matching problem***, which was first analyzed in the 18^{th} century.

2.5.13. Example. Suppose that someone writes n letters and addresses n corresponding envelopes, but then puts the letters into the envelopes randomly. Of interest is the probability that there is at least one match, i.e., that at least one letter is put into the correct envelope. In particular, how does this probability behave as n becomes large?

Let A_i be the event that the ith letter is put into the correct envelope. We want to find $P[A_1 \cup A_2 \cup \cdots \cup A_n]$, so we apply the inclusion-exclusion formula. First we note that $P[A_i \cap A_j]$ does not depend on i and j, i.e., the probability that the correct letters are put into any given two envelopes doesn't depend on their numbers. We set $P[A_i \cap A_j] =: p_2$, so

$$\sum_{1 \le i < j \le n} P[A_i \cap A_j] = \binom{n}{2} p_2.$$

A similar argument applies to the other probabilities in (2.5.3).

The Matching Problem

We apply the inclusion-exclusion principle,

$$\begin{aligned}
&P[A_1 \cup A_2 \cup \ldots \cup A_n] \\
&= \sum_{1 \le i \le n} P(A_i) - \sum_{1 \le i < j \le n} P[A_i \cap A_j] + \sum_{1 \le i < j < k \le n} P[A_i \cap A_j \cap A_k] \\
&\quad - + \ldots + (-1)^{n+1} P[A_1 \cap A_2 \cap \ldots \cap A_n] \\
&= \binom{n}{1} p_1 - \binom{n}{2} p_2 + \binom{n}{3} p_3 - + \cdots + (-1)^{n+1} \binom{n}{n} p_n.
\end{aligned}$$

It is clear that $p_1 = P[A_i]$, the probability of putting the correct letter into the ith envelope, is $1/n$. However, $p_2 = P[A_i \cap A_j] \neq P[A_i]P[A_j]$ because the events A_i and A_j are not independent.

Let us consider the probability p_r of $P[A_{i_1} \cap \cdots \cap A_{i_r}]$. There are $n!$ ways of matching letters to envelopes, but we require r envelopes to have the correct letters. Thus, there remain $(n-r)!$ ways of matching the other letters to envelopes.

The Matching Problem

It follows that

$$p_r = \frac{(n-r)!}{n!}.$$

Then

$$P[A_1 \cup A_2 \cup \ldots \cup A_n] = \binom{n}{1} p_1 - \binom{n}{2} p_2 + - \cdots + (-1)^{n+1} \binom{n}{n} p_n$$
$$= -\sum_{r=1}^{n} (-1)^r \binom{n}{r} \frac{(n-r)!}{n!} = -\sum_{r=1}^{n} \frac{(-1)^r}{r!}.$$

Since $e^{-1} = \sum_{r=0}^{\infty} (-1)^r / r!$, we have

$$P[A_1 \cup A_2 \cup \ldots \cup A_n] \approx 1 - \frac{1}{e} \approx 0.63212.$$

The approximation is actually quite good for small n, e.g., for $n = 7$ we obtain a probability of 0.63214.

Monte Carlo Algorithms

All the algorithms that we have encountered so far are ***deterministic***, i.e., they follow a fixed sequence of steps. Given some input, it is possible to describe precisely what actions the algorithm will take and the output is completely determined.

However, for some applications this procedure is too rigid. A certain randomness in performing one or more steps of the algorithm may be desirable. Such algorithms are called ***probabilistic algorithms***.

We first study a specific type of probabilistic algorithm here, called a ***Monte Carlo algorithm***. The name stems from the famous Monte Carlo casino in the principality of Monaco, which is located along the mediterranean coast of France.

Image source:
http://commons.wikimedia.org/wiki/File:Monte_Carlo_Casino.jpg.
Used under the license given there

Monte Carlo Algorithms

Monte Carlo algorithms always yield an output, but it is possible that the output is false. The steps involved in these algorithms may be considered to be similar to gambling, whence the name.

A Monte Carlo algorithm performs several iterations, or tests, to determine the truth of some statement. Each test will give either "true" or "unknown". If at least one of the tests yields "true", the algorithm will output "true". If all the tests yield "unknown", then the algorithm will output "false".

The probability that the actual statement is true but a test yields "unknown" instead of "true" is small, and by performing many tests, the probability that the final output is "false" while the statement is actually true can be made very small.

Miller-Rabin Test for Primality

2.5.14. Example. The Miller-Rabin test is a Monte Carlo Algorithm that determines whether or not an integer is prime.

Let $n \in \mathbb{N}$ be an odd integer. Then

$$n - 1 = 2^s t \qquad \text{for some } s \in \mathbb{N} \text{ and an odd integer } t \in \mathbb{N}.$$

The integer n is said to pass the ***Miller-Rabin test to base*** b if either

$$b^t \equiv 1 \pmod{n} \qquad \text{or} \qquad b^{2^j t} \equiv -1 \pmod{n}$$

for some j with $0 \leq j \leq s - 1$.

- If n is prime and $1 < b < n$, then n passes the Miller-Rabin test to base b.
- If n is composite, then there are fewer than $n/4$ bases b with $1 < b < n$ such that n passes Miller's test to base b.

Miller-Rabin Test for Primality

The Monte Carlo algorithm now randomly selects k bases b and performs the Miller-Rabin test.

- If n fails the test for any of the bases used, the algorithm will return "true" (n is composite).
- If n passes each test, the answer is still unknown. Nevertheless, the algorithm will return "false" (n is prime).

The probability that n is composite and still passes the test each of the k times is

$$p_k = \frac{1}{4^k}.$$

If, e.g., $k = 30$ tests are performed, $p_k < 10^{-18}$. It is almost certain that a number that the algorithm returns as prime actually is prime.

Quick Sort

Among the different sorting algorithms studied so far the most efficient one is Merge Sort. However, Merge Sort requires a significant amount of spatial resources. Furthermore, although Merge Sort is asymptotically optimal, there are other algorithms that perform better in practice.

One of these is called ***Quick Sort***. Like Merge Sort, it uses a divide-and-conquer strategy. But instead of merging two lists, Quick Sort reorders each sublist according to a chosen element:

- From a list of elements, choose an element, called the ***pivot***. Often, this is the last element of the list.
- Rearrange the list by ordering all elements with values smaller than the pivot to its left and all larger elements to its right.
- The pivot is now in its final position and there are two sublists, one on either side.
- Recursively apply the above procedure to these two sublists.

Pseudocode for Quick Sort

2.5.15. Algorithm. ***(Quick Sort)***

Input: $L = \langle l_1, \cdots, l_n \rangle$ a list of n elements
Output: L with its elements in increasing order

```
1  Function qsort(L, j, k):
2      if j < k then
3          for i ← j to k − 1 do
4              if l_i < l_k then swap l_i and l_j; j ← j + 1 ;
5          end for
6          swap l_j and l_k;
7          qsort(L, 1, j − 1); qsort(L, j + 1, k)
8      end if
9      else return L;
10 end
```

Quick Sort

2.5.16. Examples.

(i) To order the list $L = \langle 4, 3, 1, 2 \rangle$, 2 is selected as the pivot. The following comparisons are performed: $4 \not< 2$, $3 \not< 2$, $1 < 2$. 1 and 4 are swapped, and so are 2 and 3 such that 2 is in its final position and $L = \langle 1, 2, 4, 3 \rangle$. The sublist on the left of 2 consists of only one element and as such is already ordered. The one on the right of 2 is $\langle 4, 3 \rangle$, which is ordered using one comparison $3 < 4$. $L = \langle 1, 2, 3, 4 \rangle$ is now ordered.

(ii) To order the list $L = \langle 1, 2, 3, 4, 5 \rangle$, 5 is selected as the initial pivot. Four comparisons are performed $1 < 5$, $2 < 5$, $3 < 5$ and $4 < 5$. 5 is already in its final position. Only the left sublist with pivot 4 is to be considered, which leads to three comparisons. Then two comparisons for 3 as a pivot and 1 when it is 2. This easily extends to a list of n elements that are already ordered. In this case $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$ comparisons are perfomed.

Problems of Quick Sort with Fixed Pivot

The second example shows that the worst-case complexity of Quick Sort is $O(n^2)$. Even worse, the case of a sorted or almost-sorted list is fairly common in applications, so that this worst case occurs fairly often in practice. On the other hand, for "random" lists, Quick Sort is quite fast.

While we could decide to choose a different pivot (for example, the first instead of the last element of a list), this would still leave the algorithm vulnerable to being slow for certain special cases.

A better solution is to choose the pivot ***randomly***!

Las Vegas Algorithms

This leads us to a new type of probabilistic algorithm is called a ***Las Vegas algorithm***, named after the famous desert city in Nevada, US, where gambling is legal.

While a Monte Carlo algorithm sometimes returns a wrong result, albeit with low probability, a Las Vegas algorithm always gives the correct output. What is random in a Las Vegas algorithm is the ***running time*** of the algorithm.

In a deterministic algorithm, given specified input and the steps of the algorithm, the time complexity can be calculated precisely. In a Las Vegas algorithm, even for specified input, this is not possible.

One often uses Las Vegas algorithms for comparatively small input size, since for large input their running time can be quite long.

Pseudocode for Quick Sort with Random Pivot

2.5.17. Algorithm. ***(Quick Sort with Ransom Pivot)***

Input: $L = \langle l_1, \cdots, l_n \rangle$ a list of n elements
Output: L with its elements in increasing order

```
1  Function qsort_r(L, j, k):
2      if j < k then
3          r ← rand(k); swap l_r and l_k;  /* 0 < r ≤ k: rand pivot */
4          for i ← j to k − 1 do
5              if l_i < l_k then swap l_i and l_j; j ← j + 1; ;
6          end for
7          swap l_j and l_k;
8          qsort_r(L, 1, j − 1); qsort_r(L, j + 1, k);
9      end if
10     else return L;
11 end
```

Time Complexity of Quick Sort with Random Pivot

By choosing a random pivot, one essentially forces a given list to be "random", even if it is structured in some way originally. Hence, Quick Sort with a random pivot is equivalent to Quick Sort with a fixed pivot acting on a randomly ordered list.

The random pivot serves only to prevent "worst" cases (which may be common practical configurations) from occurring often.

Therefore, the (worst-case) time complexity of Quick Sort with random pivot remains $O(n^2)$. However, as we now show, the "average case" complexity is $O(n \log n)$.

Let $t(n)$ denote the number of comparisons needed to Quick Sort a list of n elements. Then

$$t(0) = t(1) = 1$$

since lists of one or zero elements are trivially sorted and only one comparison is needed to find the size of the list.

Time Complexity of Quick Sort with Random Pivot

Suppose that the chosen pivot is the ith-largest element of the list. Then

$$t(n) = t(i-1) + t(n-i) + n, \qquad i = 1, \ldots, n, \qquad n \geq 2.$$

since the left sublist will have length $i-1$, the right sublist will have length $n-i$ and $n-1$ comparisons are needed to place each element into the appropriate sublist. Furthermore, one comparison is used to check if the list size is at least two; if not, the contents of the list are returned.

If the pivot is chosen randomly, the probability that it will be the ith-largest element in the list is equal to $1/n$ for each i. We calculate the ***average time complexity*** as the ***expected number of comparisons*** by averaging over all possible sizes of the chosen pivot:

$$\bar{t}(n) = \frac{1}{n} \sum_{i=1}^{n} \left(\bar{t}(i-1) + \bar{t}(n-i) \right) + n$$

Time Complexity of Quick Sort with Random Pivot

Since

$$\sum_{i=1}^{n} \bar{t}(i-1) = \sum_{i=1}^{n} \bar{t}(n-i)$$

(only the order of summation is reversed) we see that

$$\bar{t}(n) = \frac{2}{n} \sum_{i=1}^{n} \bar{t}(i-1) + n.$$

Multiplying by n, we obtain

$$n \cdot \bar{t}(n) = 2 \sum_{i=1}^{n} \bar{t}(i-1) + n^2 \tag{2.5.5}$$

We also have

$$(n-1) \cdot \bar{t}(n-1) = 2 \sum_{i=1}^{n-1} \bar{t}(i-1) + (n-1)^2 \tag{2.5.6}$$

Time Complexity of Quick Sort with Random Pivot

Subtracting (2.5.6) from (2.5.5),

$$n \cdot \bar{t}(n) - (n-1) \cdot \bar{t}(n-1) = 2\bar{t}(n-1) + 2n - 1$$

or

$$\frac{1}{n+1}\bar{t}(n) = \frac{1}{n}\bar{t}(n-1) + \frac{2}{n+1}$$

This is a linear, inhomogeneous, first-order recurrence relation with variable coefficients. To find a solution, we substitute:

$$\frac{\bar{t}(n)}{n+1} = \frac{\bar{t}(n-1)}{n} + \frac{2}{n+1} = \frac{\bar{t}(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$\vdots$$

$$= \frac{t(1)}{2} + 2\sum_{k=1}^{n+1} \frac{1}{k}.$$

Time Complexity of Quick Sort with Random Pivot

Since

$$\frac{1}{k} \leq \int_{k-1}^{k} \frac{1}{x}\, dx, \qquad k \geq 2,$$

we have

$$\begin{aligned}
\frac{\bar{t}(n)}{n+1} &\leq \frac{1}{2} + 2 + 2\sum_{k=2}^{n+1} \int_{k-1}^{k} \frac{1}{x}\, dx \\
&= \frac{1}{2} + 2 + 2\int_{1}^{n+1} \frac{1}{x}\, dx \\
&= \frac{1}{2} + 2 + 2\ln(n+1) \\
&= O(\log n)
\end{aligned}$$

and hence

$$\bar{t}(n) = O(n \log n).$$

Worst-Case and Average-Case Time Complexity

This shows our assertion: that ***on average***, Quick Sort needs $O(n \log n)$ comparisons to sort a list. This is true no matter whether the pivot is chosen randomly or deterministically. Random choice of pivot only serves to prevent common worst-case scenarios from occurring often.

Let us summarize several issues that we need to be aware of when discussing time complexity:

- Time complexity does not make any statement about the spatial resources required.
- Complexity expressed using Landau symbols is an ***asymptotic statement***. For finite n, an algorithm that is asymptotically slow can be very fast.
- Complexity as we have previously calculated it is actually ***worst-case complexity***. An algorithm with bad worst-case complexity may still be very useful in practice, if the worst case occurs very rarely. For that reason, ***average-case complexity*** is often at least as important as worst-case complexity.

The Probabilistic Method

The probabilistic method is a technique for showing the existence of objects with specified properties. It is non-constructive, i.e., it does not give information on how to find/construct these objects but only establishes that they exist.

The method is based on assigning probabilities to all elements of a set S. An object with specified properties exists in S if its probability is greater than zero.

2.5.18. Probabilistic Method. If the probability that an element of a set S does not have a specified property is less than 1, then there exists an element in S with this property.

As a demonstration, we will prove the following statement:

2.5.19. Example. Let $k \in \mathbb{N} \setminus \{0, 1\}$. Then $R(k, k) \geq 2^{k/2}$, where R denotes the Ramsey number introduced on Slide 384.

The Probabilistic Method

It is clear that this statement is true for $k = 2$ and $k = 3$, since $R(2, 2) = 2 = 2^{2/2}$ and $R(3, 3) = 6 > 2^{3/2}$. We will use the probabilistic method to show that if a party has $n < 2^{k/2}$ members, it is possible that no k members are mutual friends or enemies.

Suppose that a party of n people is assembled randomly in such a way that it is equally likely for any two people at the party to be friends or enemies. There are $\binom{n}{k}$ different sets of k people at this party, which we denote by $S_1, S_2, \ldots, S_{\binom{n}{k}}$. Let E_i be the event that all k members of S_i are either mutual friends or enemies. Then the probability that there are either k mutual friends or enemies in S is

$$P[\text{there exist } k \text{ mutual friends or enemies}] = P\left[\bigcup_{i=1}^{\binom{n}{k}} E_i\right].$$

The Probabilistic Method

There are $\binom{k}{2} = k(k-1)/2$ pairs of people in each S_i, and the probability that they are enemies is $1/2$. Thus, the probability that all members if S_i are enemies is $2^{-k(k-1)/2}$. The probability that they are all friends is also $2^{-k(k-1)/2}$, so

$$P[E_i] = 2 \cdot 2^{-k(k-1)/2}.$$

Using Boole's inequality (2.5.2) and $\binom{n}{k} \leq n^k/2^{k-1}$ (see assignments),

$$P\left[\bigcup_{i=1}^{\binom{n}{k}} E_i\right] \leq \sum_{i=1}^{\binom{n}{k}} P[E_i] \leq 2^{1-k(k-1)/2} \frac{n^k}{2^{k-1}} = 2^{2-k/2} \frac{n^k}{2^{k^2/2}}.$$

Now if $n < 2^{k/2}$ and $k \geq 4$,

$$P\left[\bigcup_{i=1}^{\binom{n}{k}} E_i\right] < 1.$$

The Probabilistic Method

We have therefore shown that for a party whose members are randomly assigned friendship or enmity with each other,

$$P[\text{there exist } k \text{ mutual friends or enemies in a party of } n \text{ members}] < 1.$$

However, by Definition 2.5.1,

$$= \frac{P[\text{there exist } k \text{ mutual friends or enemies in a party of } n \text{ members}]}{\text{number of parties containing groups of } k \text{ mutual friends/enemies}}{\text{number of all possible parties with } n \text{ members}}$$

Since the quotient is less than 1 if $n < 2^{k/2}$ and $k \geq 4$ it follows that there must exist some party of n members not containing a group of k mutual friends or enemies and we see that $R(k, k) \geq 2^{k/2}$.

Mathematics and Cryptography

At the end of World War II many mathematicians decided to refocus their work on less applicable mathematics such as abstract algebra, number theory or algebraic geometry. By doing so they expected their work not to lead to new destructive weapons.

Only a few decades later their work found major applications in the field of ***cryptography***, the "science of secrets", which encompasses various aspects of information security such as authentication, confidentiality or data integrity.

A message that is to be hidden is called the ***plain text*** while the obscured method is called the ***cipher text***. The process of generating the cipher text from the plain text is called ***encryption***, while the reverse process is called ***decryption***.

With the advance of technology, cryptography has become an integral part of our lives, being present in ATM cards, passports, computers, websites...

A Brief History of Cryptography - Ancient Origins

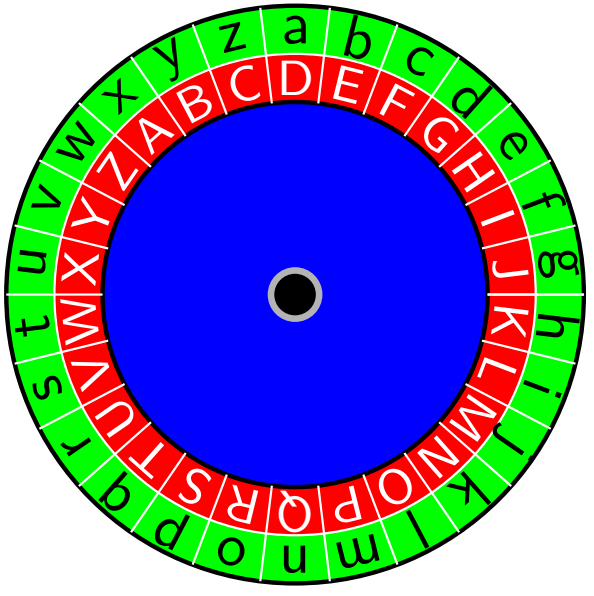
Cryptography is a very old science, the earliest proper record of its use having been found in Mesopotamia and dated near 1500 B.C.

Later cryptography was in common use by the Greeks and the Romans. One of the simplest and most widely known encryption technique is called ***Caesar's cipher***, which is based on shifting the letters of the alphabet:

Such a shift can be performed manually using two rings, as shown at right, or through modular arithmetic by associating numbers to letters,

$$A \mapsto 0, \qquad B \mapsto 1, \qquad \ldots \qquad Z \mapsto 25,$$

adding k modulo 26 to perform a shift by k letters, and then converting back to letters.



A Brief History of Cryptography - Caesar's Cipher

2.6.1. Example. The plain text MEET YOU IN THE PARK is to be encrypted using Caesar's cipher with $k = 3$. We first convert the text into numbers,

12 4 4 19 24 14 20 8 13 19 7 4 15 0 17 10

Then we add 3 modulo 26 to these numbers,

15 7 7 22 1 17 23 11 16 22 10 7 18 3 20 13

and translate back into letters, obtaining the cipher text PHHW BRXL QWKH. Decryption is performed by reversing the process.

Note that if $k = 13$, then $x + 13 \bmod 26 = x - 13 \bmod 26$ and decryption and encryption use the same procedure.

A Brief History of Cryptography - Caesar's Cipher

Caesar's cipher is of course easily decoded. Apart from being very simple to reverse if it is known that it was used, it suffers from a separate problem: it is a ***fixed substitution cipher***, where every letter is substituted with a fixed, different letter.

As such, it is vulnerable to ***frequency analysis***: Every language features a different letter distribution. For instance, in English the most common letters are E, T, A, O and I. It is therefore easy to recover those letters by analyzing the frequency of the letters in an encrypted message.

The Roman emperor Julius Caesar used the cipher with a shift of 3 letters in the first century B.C. For $k = 13$ it is known as ROT-13 and has found use in newsgroups as a method for hiding answers to riddles, movie spoilers, etc.

A Brief History of Cryptography - One-Time Pads

During World War I and II, cryptography was heavily used on all sides. An important breakthrough was made in 1917 with the discovery of the one-time pad, an theoretically breakable encryption technique. It works as follows:

A message is represented as a number, most conveniently a string of zeros and ones, e.g.,

$$\text{plaintext message} = 1100001111100$$

Suppose that a truly random number (called the secret key) is known to both sender and receiver, e.g.,

$$\text{secret key (random number)} = 0001111001000.$$

The sender encrypts the message by adding the two numbers modulo 2, digit by digit, yielding

$$\text{encrypted message} = 1101110110100$$

A Brief History of Cryptography - One-Time Pads

Since the secret key was a random number, the digit-by-digit sum modulo two is random as well. Without the secret key, the message can not be decoded. The encryption is theoretically secure; however there are certain disadvantages:

- Obtaining a truly random number is difficult. Furthermore, the length of the random number must equal the message length.
- The secret key must be distributed to the receiver before the secret message is sent. This presents its own problems in practice.

Cryptographic techniques where both sender and receiver need to know the decryption key are said to be ***symmetric***. Up to the 1970s, all techniques were of this type.

A Brief History of Cryptography - Asymmetric Encryption

To overcome the issue of key-sharing, the notion of ***asymmetric encryption*** was introduced. Asymmetric cryptography works in a similar fashion as a padlock: everybody is able to lock it, but only the owner of the key can unlock it.

The common strategy in public key cryptography consists for a user to make publicly available his ***public key*** such that everybody is able to encrypt a message. Then the receiver only needs to apply his ***private key*** on the ciphertext in order to recover the initial message.

In order to discuss certain asymmetric encryption techniques, we first introduce some further background.

Cryptographic Functions

2.6.2. Definition. Let f be a bijective function from a set X to a set Y.

- For given $x \in X$, the computation of $f(x)$ is said to be a ***tractable problem*** if it can be performed with at most polynomial time complexity, i.e., with $O(n^b)$ operations for input size n and some $b > 0$.
- The function f is said to be a ***one-way function*** if for all $x \in X$, the calculation of $f(x)$ is tractable while for "essentially all" $y \in Y$, the calculation of $f^{-1}(y)$ is intractable.
- The function f is a ***trapdoor one-way function*** if f is a one-way function with the additional property that given some extra information the computation of $f^{-1}(y)$ becomes tractable for all $y \in Y$.

Public-Key Cryptography

The above definition of tractability is based on currently known algorithms. It is usually not possible to prove that a given problem is intractable in principle. It may happen that a hitherto intractable problem becomes tractable due to advances in software (new algorithms) or hardware (quantum computers).

Public-key cryptography is based on a trapdoor one-way function f:

- A message x is encrypted to a ciphertext $f(x)$, using the public key (The function f is essentially the public key.)
- Retrieving the message x requires inverting f, which is an intractable problem.
- However, the recipient has a "trapdoor" which allows the tractable inverting of f and retrieval of the message.

The Integer Factorization problem

Suppose that two (large) prime numbers p and q are given. Calculating their product,

$$f(p, q) = p \cdot q =: n$$

is a tractable problem. However, inverting f, i.e., factoring n to find

$$(p, q) = f^{-1}(n),$$

is not tractable, as we now show.

The most straightforward approach is ***trial division***: given an integer n, we attempt to divide n by prime numbers smaller than $\sqrt{n}$. If n is composite, one of these numbers must divide n (by Theorem 1.7.7).

Let us now discuss the time complexity of this approach.

Complexity of Trial Division

By the Prime Number Theorem 2.4.48 the number of primes less than n is given asymptotically by

$$\pi(n) = O(n/\ln(n)) \qquad \text{as } n \to \infty$$

so we see that

$$O(\sqrt{n}/\ln(n))$$

divisions need to be performed in trial division, supposing that all prime numbers less than $\sqrt{n}$ are known. It appears that the complexity of trial division is not exponential after all...

However, if we express the complexity in terms of input size, this is another matter: if the binary expansion of n has k bits, then $n \approx 2^k$ and

$$O(2^{k/2}/k)$$

divisions are required.

Factorization Strategies

If an algorithm runs in polynomial time with respect to the numerical value of the input, but in exponential time with respect to the input size, we say it runs in ***pseudo-polynomial time***. Trial division is of this type.

2.6.3. Remarks. Given a large random integer n, the probability for n to be divisible by 2 is 1/2; by 3, 1/3; by 5, 1/5 etc. One can deduce that about 88% of integers have a factor smaller than 100 and 92% a factor smaller than 1000.

Therefore, despite its exponential complexity, trial division is used in almost all factoring programs. All the small factors are first removed before more advanced strategies are employed to totally factor n.

In practice, trial division is implemented through a large table containing all the primes, or alternatively the difference between two consecutive primes, up to 10 million. Then even for a 1000 digit long integer it only takes a few seconds to perform all the trial divisions and ensure that n is free of any small factor.

Factorization Strategies

Another simple idea in order to remove small factors consists in computing $\gcd(n, P)$ where P corresponds to the product of all the prime numbers below a given bound B. Compared to trial division this strategy seems appealing since computing a gcd can be done in polynomial time. In practice, this method is much more efficient when considering primes below 1000 but it becomes extremely slow when checking prime factors of size around one million.

In the first case the product of all the primes is about 1,400 bits while in the second case it is approximately 1,500,000! Computing the gcd of an integer around 2^{2048} and P, then takes much longer.

This simple example highlights how practical cases can highly diverge from the theoretical asymptotic analysis.

The RSA Problem

The one-way function $f(p, q) = n$ can be used to create a trapdoor one-way function. This is the basis of the RSA cryptosystem, which is named after its (public) inventors, Rivest, Shamir and Adleman.

Let $n = pq$, with p and q two primes, and a positive integer e such that $\gcd(e, (p-1)(q-1)) = 1$ be given. Then the function $f : \mathbb{Z} \to \mathbb{Z}$ given by

$$f(m) = m^e \mod n$$

is thought to be a trapdoor one-way function.

This means that calculating $f(m)$ for given m is easy, but finding

$$m = f^{-1}(c) = \sqrt[e]{c} \mod n$$

for a given $c \in \mathbb{N}$ is hard. However, if p and q are known (the trapdoor), then f^{-1} becomes easy to compute, as we now show.

The RSA Problem

Suppose the integer $c \in \mathbb{N}$ is given and that e, p and q are known.

Let $d \in \mathbb{N} \setminus \{0\}$ be an inverse to e modulo $(p-1)(q-1)$. (d exists by Theorem 1.8.14 since e and $(p-1)(q-1)$ are coprime.)

Then

$$\exists_{k \in \mathbb{Z}} \; de = 1 + k(p-1)(q-1)$$

and

$$c^d \equiv (m^e)^d = m^{de} = m^{1+k(p-1)(q-1)} \pmod{n}$$

From this we obtain

$$c^d \equiv m \cdot (m^{p-1})^{k(q-1)} \pmod{p}$$
$$c^d \equiv m \cdot (m^{q-1})^{k(p-1)} \pmod{q}.$$

The RSA Problem

By Fermat's Little Theorem,

$$m^{p-1} \equiv 1 \pmod{p}, \qquad m^{q-1} \equiv 1 \pmod{q}.$$

Hence,

$$c^d \equiv m \pmod{p} \qquad \text{and} \qquad c^d \equiv m \pmod{q}.$$

By the Chinese Remainder Theorem,

$$c^d \equiv m \pmod{n}.$$

This shows that the inverse of f exists,

$$f^{-1}(c) = c^d \mod n$$

and is unique modulo n.

The RSA Problem

If p and q are known, it is easy to calculate $(p-1)(q-1)$ and find the inverse of e modulo $(p-1)(q-1)$.

However, if only n is known, finding $(p-1)(q-1)$ is hard.

The integers e and n for the function f can be published and used for creating a ciphertext c from a plaintext m. However, inverting the function requires the trapdoor, consisting of the integers p and q.

It is useful to think of the RSA problem in terms of groups: The function

$$f : \mathbb{Z}_n \to \mathbb{Z}_n, \qquad f(m) = m^e$$

needs to be inverted by finding the eth root. However, finding roots in the groups $\mathbb{Z}_n$ is hard.

The RSA Protocol

Based on this mathematical background, it is possible to specify some precise sequence of actions allowing different entities to preserve or share a secret. These actions are organized into sets which all together form a ***protocol***.

The RSA cryptosystem is implemented by the following protocol:

Key generation:

1) Generate two large random (and distinct) primes p and q, each roughly the same size.
2) Compute $n = pq$.
3) Select a random integer e, $1 < e < (p-1)(q-1)$, with $\gcd(e, (p-1)(q-1)) = 1$.
4) Compute the unique integer d, $1 < d < (p-1)(q-1)$ such that $ed \equiv 1 \pmod{(p-1)(q-1)}$.
5) Release the public key (n, e), keep secret the private key d.

The RSA Protocol

In order for a user A to secretly send a message to a user B, and user B to be able to recover the initial message, the following sequence of steps is to be followed.

Encryption:

1) Retrieve B's public key.
2) Represent the message as an integer $0 \leq m \leq (n-1)$.
3) Compute $c = m^e \mod n$.
4) Send the ciphertext c to B.

Decryption:

1) Obtain the ciphertext c from user A.
2) Use the private key d to compute $m = c^d \mod n$.
3) Recover the initial message from the integer m.

The key generation has to be run by the user once to setup the parameters. Later only the encryption and decryption parts on the protocol are to be used.

Security of the RSA Algorithm and Complexity of Factoring

The security of the RSA algorithm is based on the fact that knowing only $n = p \cdot q$, it is not possible to find $d = (p-1)(q-1)$ without factoring n. We have seen that trial division has exponential time complexity, so for sufficiently large prime numbers p and q, factoring n is not practical using this method.

Of course, there are more sophisticated approaches, many suing so-called general ***General Number Field Sieves***. Their complexity is typically lower than exponential, but still higher than polynomial - hence factoring remains an intractable problem.

Security of the RSA Algorithm and Complexity of Factoring

To describe the complexity of factoring algorithms, one uses the function

$$L_n(\alpha, c) = e^{(c+o(1))\left((\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}\right)}$$

where $c > 0$ and $0 \leq \alpha \leq 1$. Here $o(1)$ denotes some function f such that $\lim_{n\to\infty} f(n) = 0$.

When $\alpha = 0$, $L_n(\alpha, c)$ describes polynomial complexity while $\alpha = 1$ indicates exponential complexity. Thus, the parameter α can be used to describe intermediate (sup-exponential) complexities.

The fastest known algorithm for factoring an integer has complexity

$$L_n\left(\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right).$$

Security in Practice

It is generally assumed that currently no existing computer can perform more than 2^{128} operations within a reasonable time span. In other words, if the encryption parameters are carefully selected such that the best algorithm available to solve a problem requires more than 2^{128} operations, then the encryption may considered to be secure.

The following diagram clarifies how the different security levels are (currently) divided. It will need to be updated to take increases of computational power into account.

Security in Practice

In order to find the size of the required product of primes $n = pq$, we set

$$2^{128} = e^{\sqrt[3]{\frac{64}{9}}(\ln n)^{1/3}(\ln \ln n)^{2/3}}. \tag{2.6.1}$$

which gives approximately $n = 7.65 \cdot 10^{763}$. In terms of bit length, n should be about 2500 bits long. In practice, lengths of powers of 2 are used, so that a bit length of 2048 bits is considered to be "secure".

The largest product ever factorized occurred in the breaking (over a period of two years) of RSA-768:

123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732 630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3

=

33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489

×

36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917

Finite and Cyclic Groups

For the following discussion, it will be useful to introduce some concepts from group theory:

2.6.4. Definition. Let $(G, \circ)$ be a group.

- $(G, \circ)$ is said to be a ***finite group*** if the set G has a finite number of elements. In that case this number of elements is denoted by $|G|$ and said to be the ***order*** of the group.
- $(G, \circ)$ is said to be ***cyclic*** if

$$\underset{g\in G}{\exists}\ \underset{b\in G}{\forall}\ \underset{k\in\mathbb{N}}{\exists} \quad b = \underbrace{g \circ g \circ \cdots \circ g}_{k \text{ times}} =: g^k$$

  Such an element g is called a ***generator*** of the group $(G, \circ)$.

2.6.5. Example. For all $n \in \mathbb{N} \setminus \{0\}$, the groups $(\mathbb{Z}_n, +)$ are finite and cyclic with generator $g = 1$.

The group $\mathbb{Z}_3$ also has generator $g = 2$, but 2 is not a generator for $\mathbb{Z}_4$.

The Discrete Logarithm Problem (DLP)

Let G be a finite, cyclic group with n elements and some generator g. Then any element $b \in G$ can be written as

$$b = g^k$$

for some k. Given g, the discrete logarithm problem consists of finding the inverse to the function

$$f(k) = g^k$$

in the group G, i.e., to find k if g and g^k are given. We call

$$\log_g(b) := k$$

the ***discrete logarithm*** of b to base g. Depending on the group, evaluating the logarithm may be easy or hard, i.e., f may or may not be a one-way function.

The Discrete Logarithm Problem (DLP)

2.6.6. Examples.

(i) Recall that $(\mathbb{Z}_n, +)$, $n \in \mathbb{N} \setminus \{0, 1\}$, consists of the set $\{0, 1, \ldots, n-1\}$ with addition modulo n. Let g be a generator so that any $b \in Z_n$ can be written as

$$b = \underbrace{g + g + \cdots + g}_{k \text{ times}} \equiv k \cdot g.$$

Since g is a generator and $1 \in \mathbb{Z}_n$, there exists some c such that $g \cdot c \equiv 1 \pmod{n}$. This number c is an inverse of g modulo n, so

$$b \cdot c \equiv kgc \equiv k$$

and we immediately regain k from b.

The Discrete Logarithm Problem (DLP)

(ii) Consider now the group $(\mathbb{Z}_p \setminus \{0\}, \cdot)$, $p \in \mathbb{N} \setminus \{0, 1\}$ a prime number, which consists of the set $\{1, \ldots, p-1\}$ with the group operation being multiplication modulo p. We will write $\mathbb{Z}_p^*$ for $(\mathbb{Z}_p \setminus \{0\}, \cdot)$.

Let g be a generator so that any $b \in \mathbb{Z}_p^*$ can be written as

$$b = \underbrace{g \cdot g \cdots g}_{k \text{ times}} \equiv g^k.$$

In this situation, finding k if b is given is much more difficult and

$$f(k) = g^k$$

is a one-way function.

Secret Key Agreement

We have seen that the RSA algorithm uses the factorization problem to create a trapdoor one-way function, which in turn can be applied to create an asymmetric encryption protocol.

Asymmetric encryption is one approach to solving the basic problem of symmetric encryption schemes: both parties must share a common secret key. There is, however, another approach to this problem, which is to devise a way for the two parties to agree on a common secret key by sharing only public information.

One such scheme is the ***Diffie-Hellman protocol***, which is based on the discrete logarithm problem.

The Diffie-Hellman Problem (DHP)

Let G be a finite cyclic group with a generator g. Then the function

$$f : \mathbb{Z}_{|G|} \times \mathbb{Z}_{|G|} \to G \times G, \qquad (a, b) \mapsto (g^a, g^b)$$

is a one-way function (if G is suitably chosen).

The Diffie-Hellman problem is as follows: suppose g^a and g^b are given. Then finding

$$g^{ab} = (g^a)^b = (g^b)^a$$

is hard, unless a and b are known.

Diffie-Hellman Key Agreement Protocol

To set up the protocol, the two parties need to agree on a common prime p and a generator g of the multiplicative group $\mathbb{Z}_p^*$.

Then the following steps are performed by each party:

1) A selects a random $a \in \mathbb{Z}_p^*$, and sends $\alpha = g^a \mod p$ to B.
2) B selects a random $b \in \mathbb{Z}_p^*$, and sends $\beta = g^b \mod p$ to A.
3) Upon receiving α, B computes $\alpha^b \equiv g^{ab} \pmod p$.
4) Upon receiving β, A computes $\beta^b \equiv g^{ab} \pmod p$.

Parties A and B now possess a common secret key, g^{ab}. Any other party knows only g^a and g^b and can not calculate g^{ab} without first solving the discrete logarithm problem to obtain a and b.

Second Midterm Exam

This concludes the material that will be covered in the second midterm exam. The material is based on the following textbook chapters:

- Chapter 4, Section 3;
- Chapter 5;
- Chapter 6, Sections 1 and 2;
- Chapter 7.

Some of the material (such as cryptographic algorithms) may not appear in this form in the textbook; it is nevertheless part of the exam. The above textbook chapters are for reference only. The lecture slides and assignments represent the definitive exam material.

For the exam, you **may not** bring a calculator or any other electronic device, such as a cellphone or electronic dictionary. A monolingual dictionary in the form of a paper book is allowed.

Part III

Selected Topics of Discrete Mathematics

Graphs

Paths and Circuits in Graphs

Planar Graphs

Trees

Applications of Graph Theory

Generating Functions

Graphs

Graphs

3.1.1. Definition. A pair $G = (V, E)$ is said to be an ***(undirected) graph*** if

- V is a non-empty set of objects called ***vertices***.
- E is a set or multiset of one- or two-element subsets of V called ***edges***. One-element subsets are called ***loops***.

The set V is called the ***vertex set***, E the ***edge set*** of G.

An edge $e = \{u, v\} \in E$ with $u, v \in V$, $u \neq v$, is said to ***connect*** the ***endpoints*** u and v.

If card $V < \infty$ we say that G is a ***finite graph***, otherwise an infinite graph.

If E is a set (every element of E occurs at most once) and does not contain one-element sets, then G is said to be a ***simple graph***.

If E is a multiset, G is said to be a ***multigraph***.

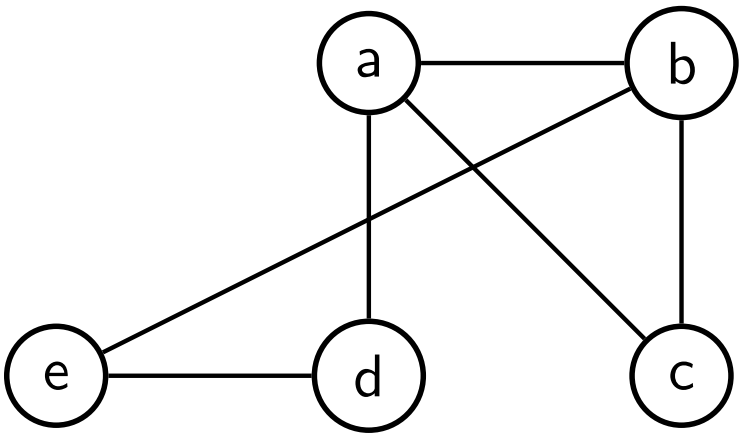
If E contains one-element subsets of V, G is said to be a ***pseudograph***.

Drawing Graphs

We usually represent a graph by drawing the vertices as points and the edges as lines joining the vertices. For example, the graph $G = (V, E)$ with

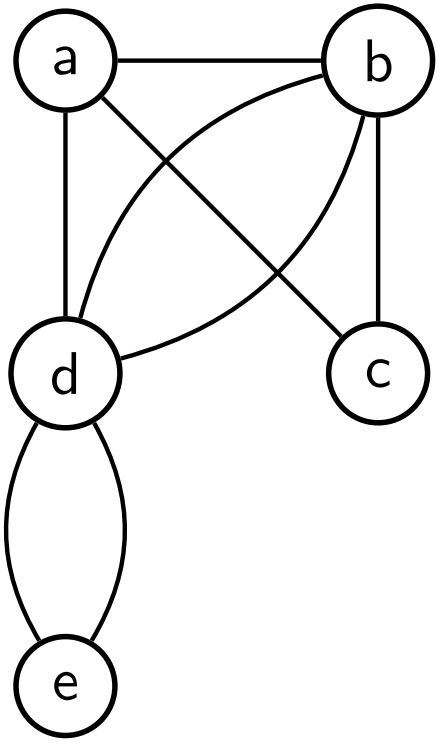
$$V = \{a, b, c, d, e\},$$
$$E = \big\{\{a, b\}, \{b, c\}, \{a, c\}, \{a, d\}, \{d, e\}, \{b, e\}\big\}$$

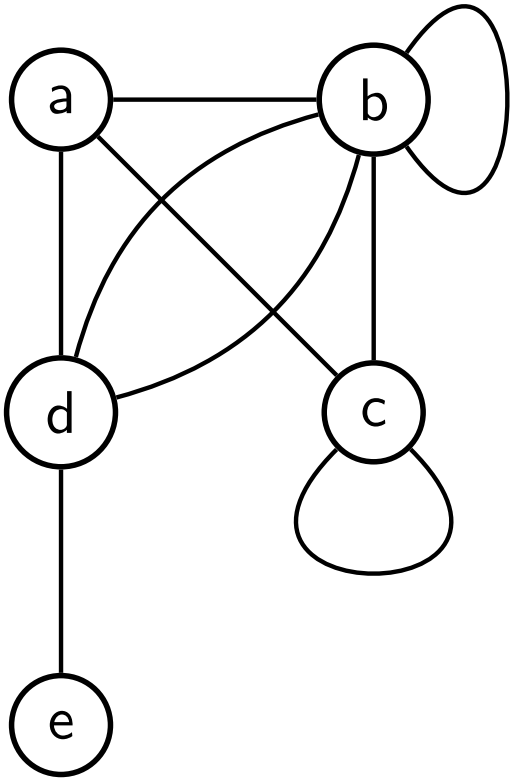
can be represented as



Drawing Graphs

Below are drawings of a multigraph and a pseudograph:





Directed Graphs

3.1.2. Definition. A pair $G = (V, E)$ is said to be a ***directed graph*** or ***digraph*** if

- V is a non-empty set of objects called ***vertices***.
- E is a set or multiset of pairs of elements of V called ***directed edges*** or ***arcs***.

An edge $e = (u, v) \in E$ with $u, v \in V$, $u \neq v$, is said to ***start at*** u and ***end at*** v.

If card $V < \infty$ we say that G is a ***finite directed graph***, otherwise an infinite directed graph.

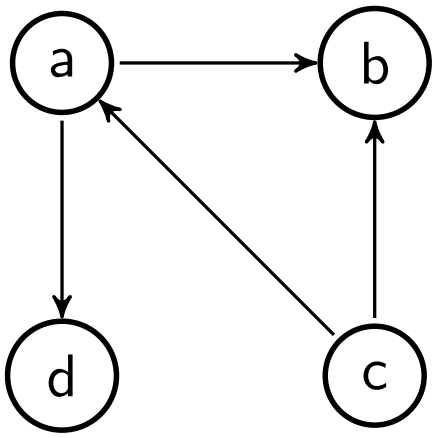
If E is a set (every element of E occurs at most once) and does not contain pairs with equal components, then G is said to be a ***simple directed graph***.

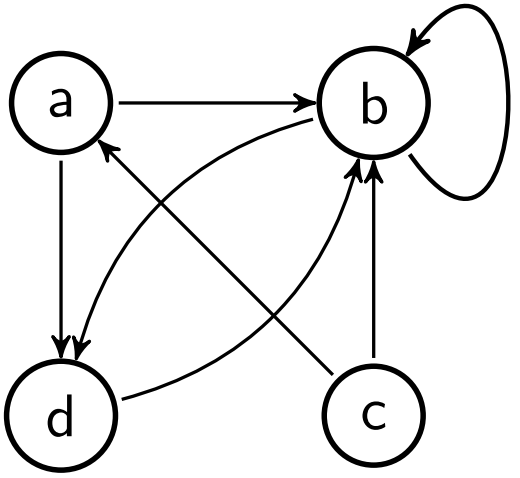
If E is a multiset or contains pairs with equal components, G is said to be a ***directed multigraph***.

Directed Graphs

3.1.3. Remark. Note that while a directed multigraph may contain loops, an undirected multigraph may not (such an undirected graph would be called a pseudograph). Since graph theory is still a fairly young discipline, the terminology used varies widely and is not always logical or consistent.

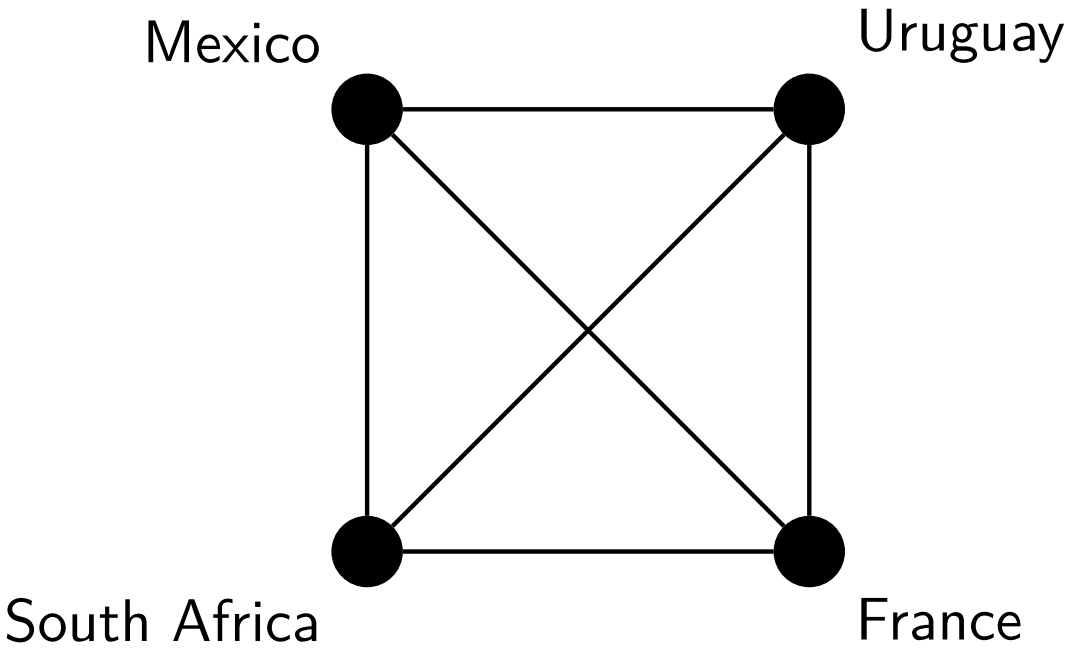
3.1.4. Example. A simple directed graph and a directed multigraph





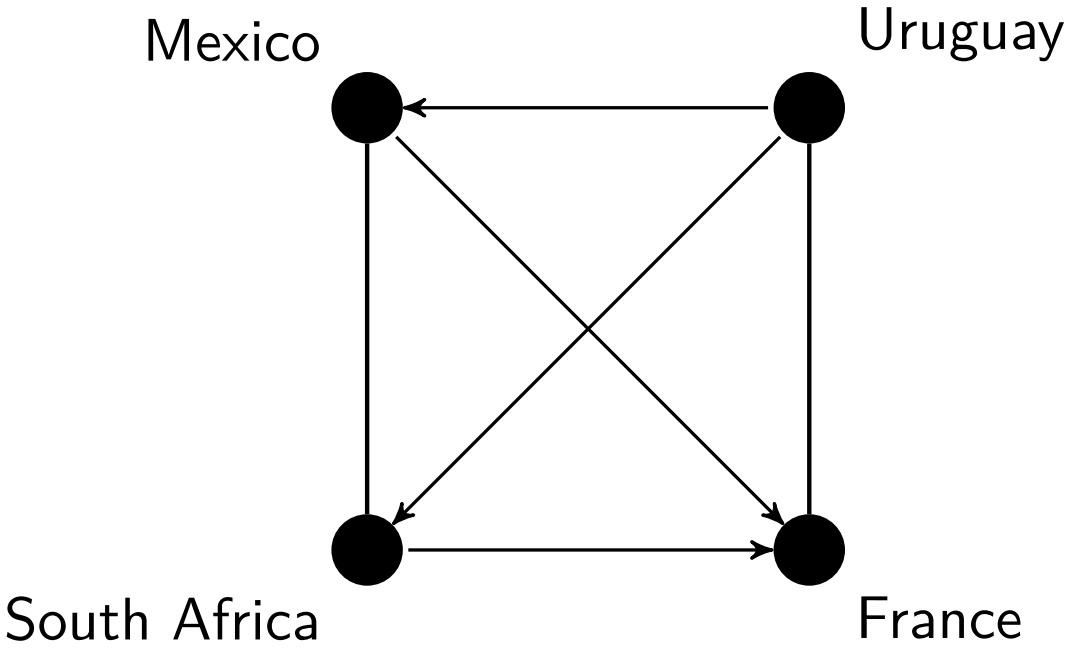
Example: Graphs in Football

3.1.5. Example. In the group stage of the 2010 World Cup, Group A consisted of Uruguay, Mexico, South Africa and France. Each of the teams played against each of the other three teams. (This is called a ***round-robin tournament***.) The following undirected simple graph represents the matches:



Example: Graphs in Football

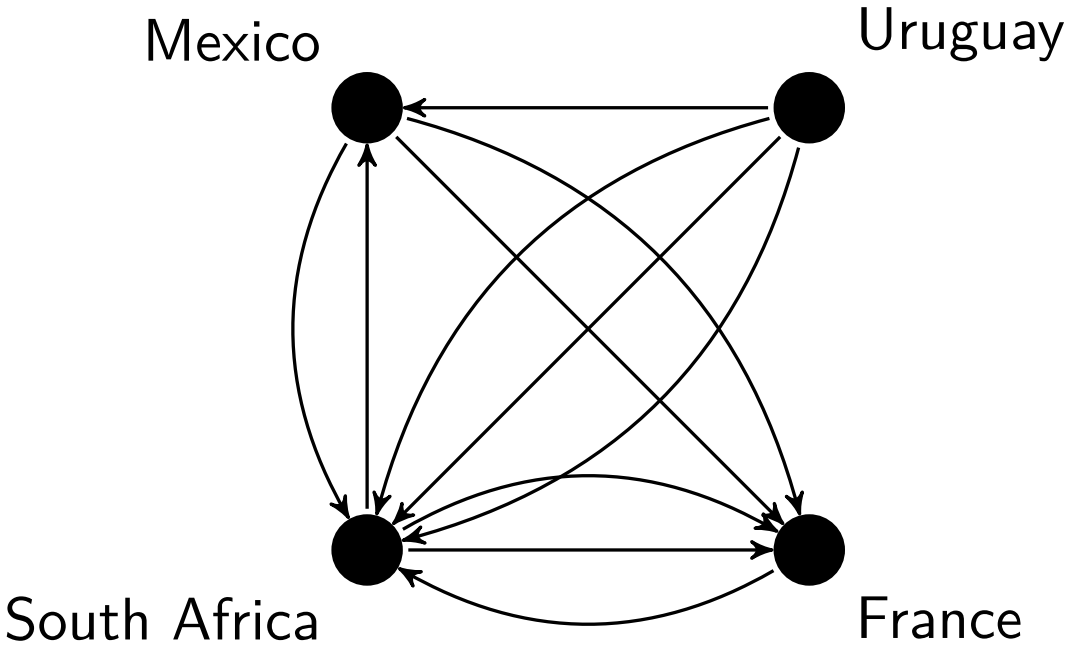
By adding arrows to the edges we can indicate which team eventually won each match:



The graph shows that Uruguay won its match against Mexico, but drew against France. A graph containing directed as well as undirected edges is called a ***mixed graph***.

Example: Graphs in Football

Alternatively, we can indicate the goals scored by each team against the other teams:



The graph shows that Uruguay did not concede a single goal and scored three goals against South Africa. The match between France and Uruguay ended without a goal scored. This is a directed multigraph.

Degree of Vertices in an Undirected Graph

3.1.6. Definition. Let G be an undirected graph and $\{u, v\}$ and edge of G. Then

- $\{u, v\}$ is said to be ***incident*** to u and v.
- u and v are said to be adjacent.

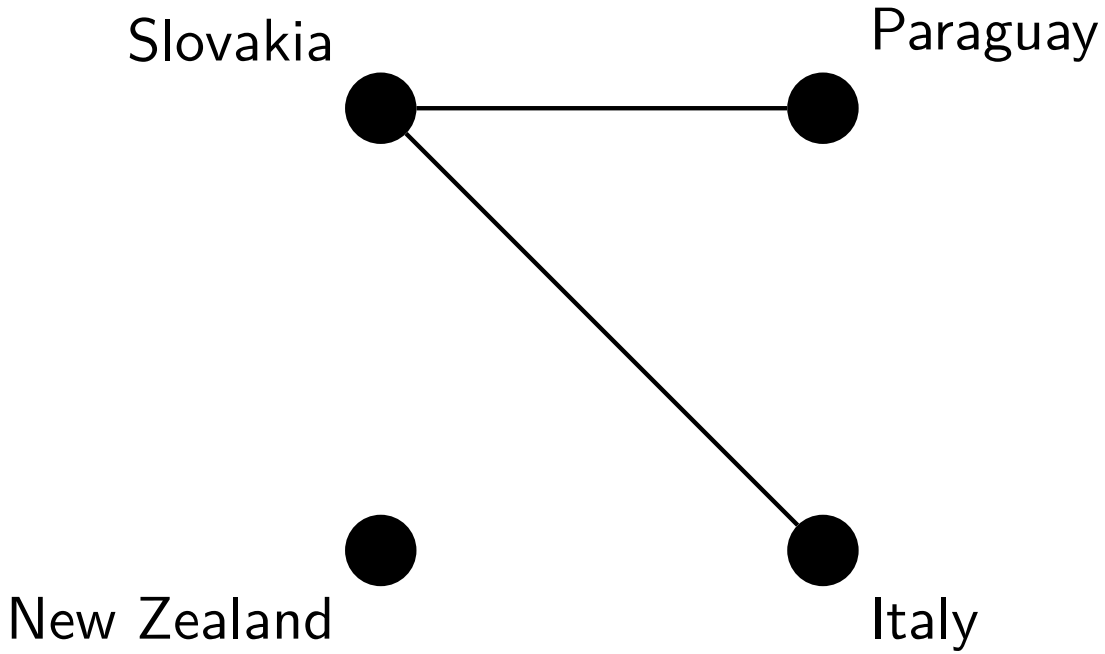
The degree $\deg(v)$ of a vertex v is the sum of the number of edges that are not loops and twice the number of loops incident to v.

A vertex is said to be ***isolated*** if it has degree zero.

A vertex is said to be ***pendant*** if it has degree one.

Example: Graphs in Football

3.1.7. Example. In the group stage of the 2010 World Cup, Group F consisted of Paraguay, Slovakia, New Zealand and Italy. The following undirected simple graph represents the matches that did not end in a draw:



Here, New Zealand is isolated, while Italy and Paraguay are pendant. The vertex Slovakia has degree 2. Italy and Slovakia are adjacent, but Italy and Paraguay are not.

The Handshaking Theorem

The sum of the degrees of all the vertices of a graph must equal twice the number of edges, since each edge "contributes" two degrees. Suppose that the graph $G = (V, E)$ has $|E|$ edges. Then

$$2|E| = \sum_{v \in V} \deg v.$$

This is known as the ***handshaking theorem***. A slightly less obvious statement is the following:

3.1.8. Theorem. An undirected graph has an even number of vertices of odd degree.

Proof.
Let V_1 and V_2 be the set of vertices of even and odd degree in $G = (V, E)$, respectively. Then

$$\underbrace{2|E|}_{\text{even}} = \sum_{v \in V_1} \underbrace{\deg v}_{\text{even}} + \sum_{v \in V_2} \underbrace{\deg v}_{\text{odd}}.$$

□

Degree of Vertices in a Directed Graph

3.1.9. Definition. Let G be a directed graph and (u, v) and edge of G. Then

- (u, v) is said to have ***initial vertex*** u and ***terminal vertex*** v.
- u and v are said to be adjacent.

The ***in-degree*** of a vertex v, denoted $\deg^-(v)$, is the number of edges with v as a terminal vertex. The ***out-degree*** of a vertex v, denoted $\deg^+(v)$, is the number of edges with v as an initial vertex.

Note that a loop contributes one to both the in- and the out-degree of a vertex.

In a digraph (V, E) the sum of the edges is equal to the sum of the in-degrees and equal to the sum of the out-degrees:

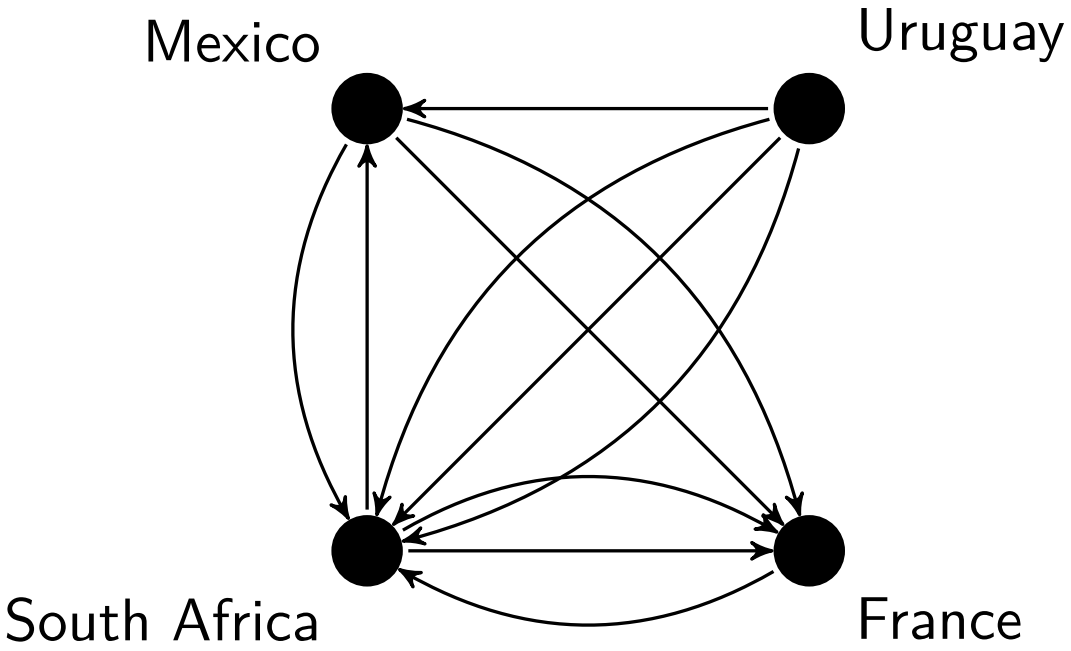
$$|E| = \sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v).$$

Example: Graphs in Football

3.1.10. Example. In the graph below,

$$\deg^-(\text{Uruguay}) = 0, \qquad \deg^+(\text{Uruguay}) = 4.$$

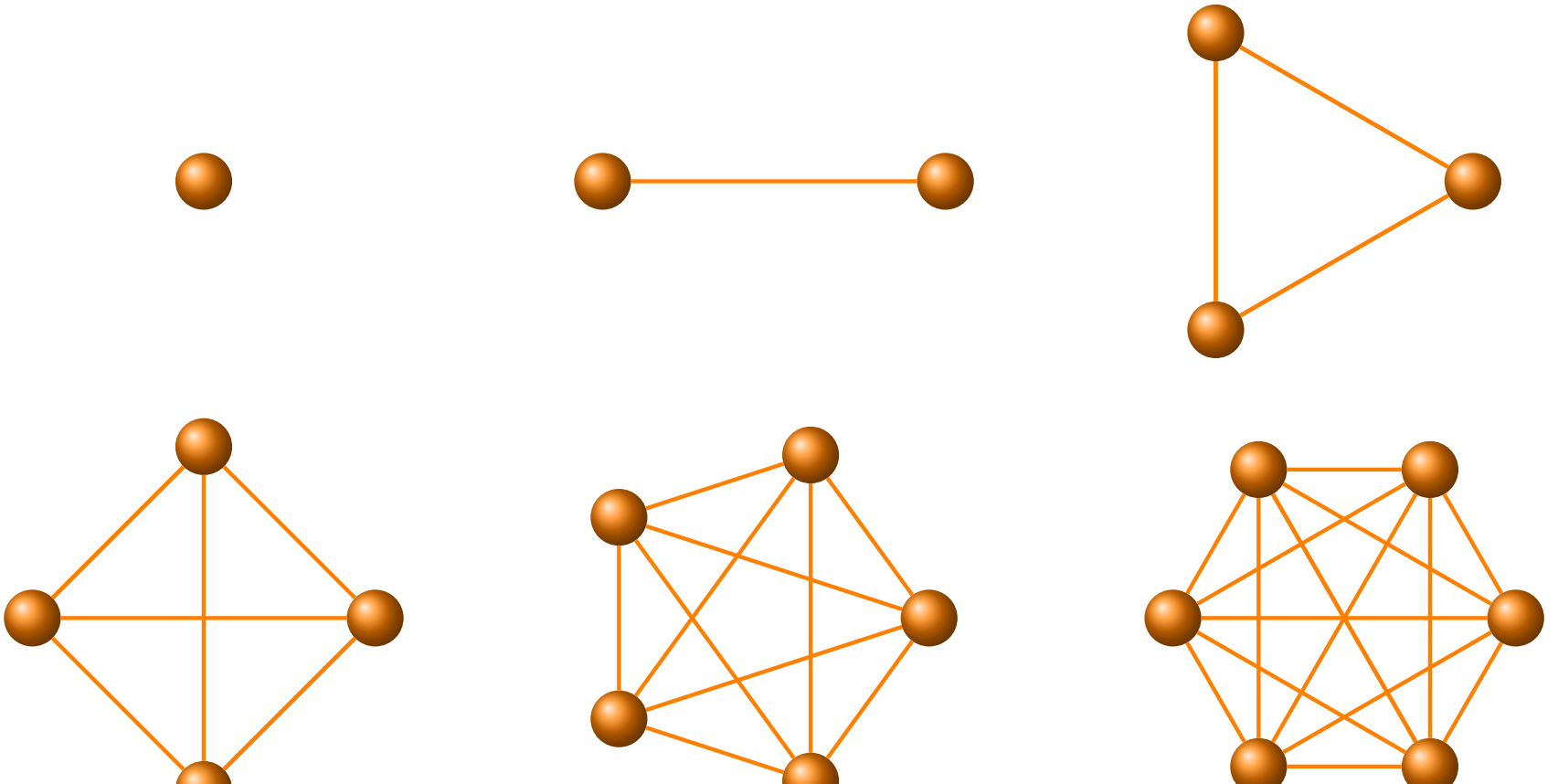
Uruguay and France are the only vertices that are not adjacent.



Complete Graphs

3.1.11. Definition. A ***complete graph*** K_j, $j \in \mathbb{N} \setminus \{0\}$ is the simple graph with j vertices that contains exactly one edge between each pair of vertices.

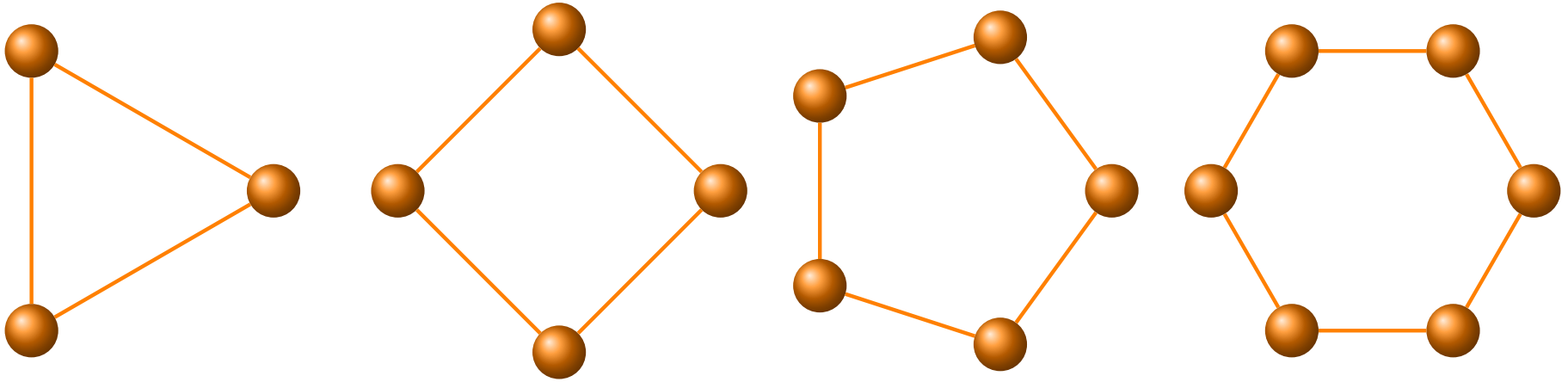
3.1.12. Examples. The complete graphs K_j, $j = 1, 2, 3, 4, 5, 6$:



Cycle Graphs

3.1.13. Definition. A ***cycle*** C_n, $n \geq 3$, is the graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and edges $\{v_j, v_{(j+1) \bmod n}\}$, $j = 1, \ldots, n$.

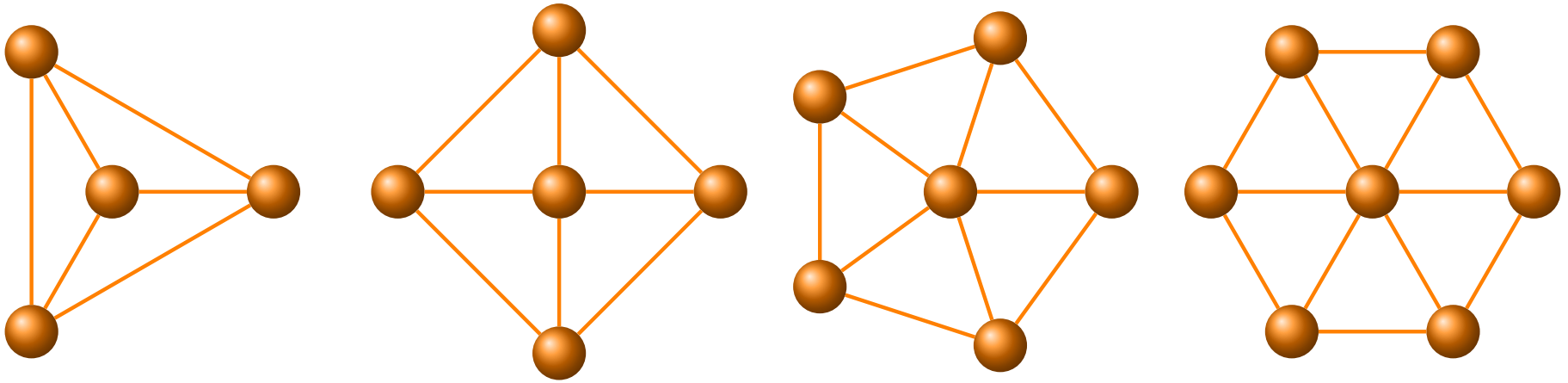
3.1.14. Examples. The cycles C_j, $j = 3, 4, 5, 6$:



Wheel Graphs

3.1.15. Definition. A ***wheel*** W_n, $n \geq 3$, is the graph $G = (V, E)$ consisting of a cycle C_n with one additional vertex connected to all other vertices.

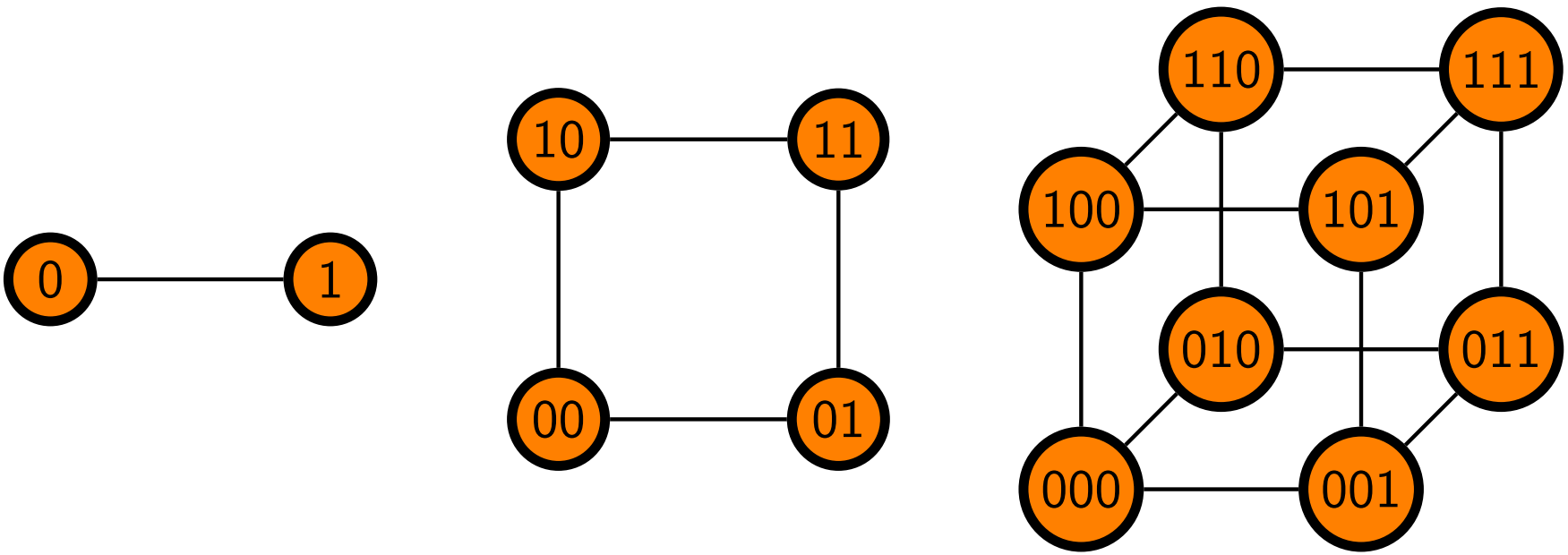
3.1.16. Examples. The wheels W_j, $j = 3, 4, 5, 6$:



Hypercube Graphs

3.1.17. Definition. A ***hypercube*** Q_n, $n \geq 1$, is the graph $G = (V, E)$ where V is the set of bit strings of length n and any two vertices are joined by an edge if they differ in a single digit.

3.1.18. Examples. The cubes Q_1, Q_2, Q_3:



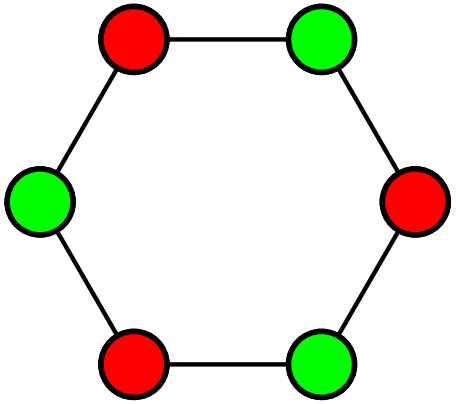
A hypercube Q_n can be constructed from two hypercubes Q_{n-1} by prefixing their vertices with zero and 1, respectively, and then joining the vertices that are identical except for the (new) first digit.

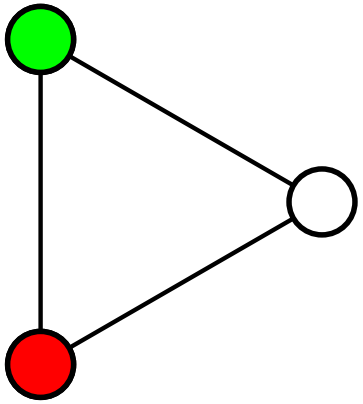
Bipartite Graphs

3.1.19. Definition. A simple graph $G = (V, E)$ is said to be ***bipartite*** if its vertex set can be partitioned into two disjoint subsets V_1 and V_2 such that every edge in E connects a vertex in V_1 to a vertex in V_2. The pair (V_1, V_2) is said to be a ***bipartition*** of G.

We often write $G = (V_1, V_2, E)$ for a bipartite graph.

3.1.20. Example. The cycle C_6 is bipartite, while the cycle C_3 is not bipartite.





Bipartite Graphs

Example 3.1.20 suggests the following theorem:

3.1.21. Theorem. A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Proof.

($\Rightarrow$) Assume that the graph $G = (V, E)$ is bipartite. Then $V = V_1 \cup V_2$, where every vertex in V_1 is adjacent only to an edge in V_2 and vice-versa. We assign one color to all vertices in V_1 and the other color to all vertices in V_2.

($\Leftarrow$) Suppose that $G = (V, E)$ and $V = V_1 \cup V_2$, where V_1 consist of vertices of one color and V_2 consists of vertices of the other color and no two vertices of the same color are adjacent. Then every vertex in V_1 is adjacent only to a vertex in V_2 and vice-versa, so (V_1, V_2) is a bipartition of G.

□

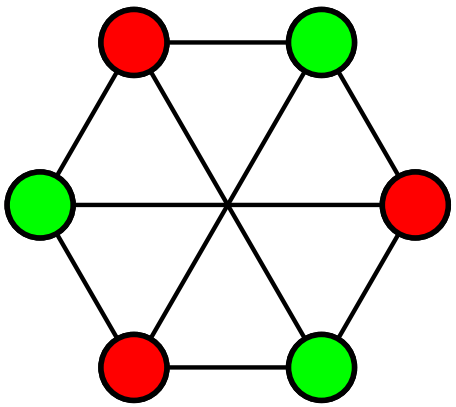
Complete Bipartite Graphs

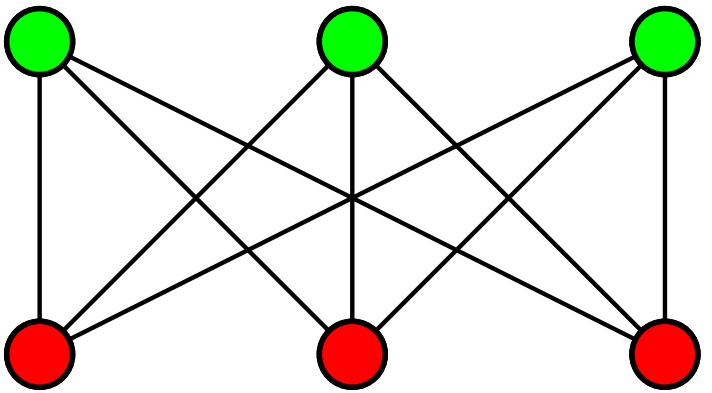
Recall from Definition 3.1.11 that a complete graph is a graph where any two vertices are connected by exactly one edge.

3.1.22. Definition. The ***complete bipartite graph*** $K_{m,n}$, $m, n \in \mathbb{N} \setminus \{0\}$, is a simple graph having a bipartition (V_1, V_2) such that $|V_1| = m$, $|V_2| = n$, and every vertex in V_1 is adjacent to every vertex in V_2.

3.1.23. Example.

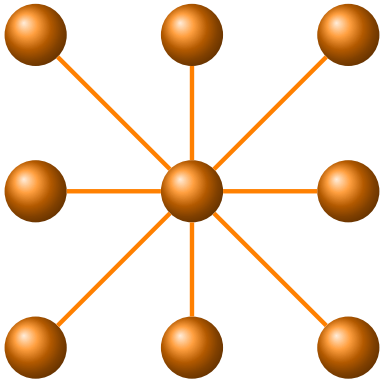
Below are two ways of drawing the graph $K_{3,3}$:



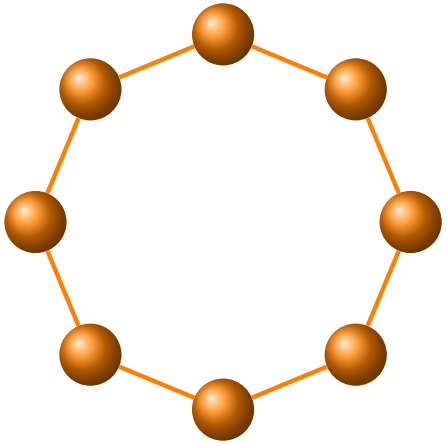


Some Applications of Graphs

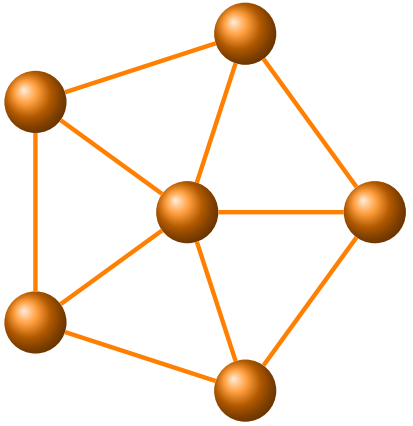
3.1.24. Example. Local-Area Networks (LANs) are interconnections of computers or computing devices (such as printers, routers etc.). There are three basic ways (topologies) of arranging the network connections, represented through graphs:



Star Topology (bipartite complete graph $K_{1,n}$)



Ring Topology (ring graph C_n)



Hybrid Topology (wheel graph W_n)

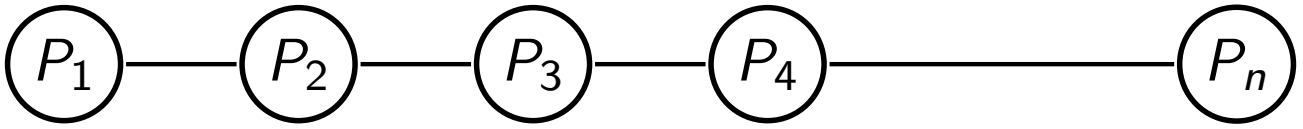
Some Applications of Graphs

3.1.25. Example. In modern computers, calculationally intensive tasks are not handled by a single processor (***serial processing***) but split into multiple sub-tasks and handled by several processors concurrently (***parallel processing***). Since the sub-tasks depend on each other, the processors need to be connected so that they can communicate with each other. The processors thus represent a network. The type of connection within the network will influence the speed of the calculation as well as the type of parallel software that can be used.

The simplest possible network would connect any two processors with each other, forming a complete graph K_n with $\binom{n}{2}$ edges. For 64 processors, this means there would be 2016 connections and any processor would be connected to 63 others. This configuration is often not possible in practice and will always be expensive.

Linear Arrays

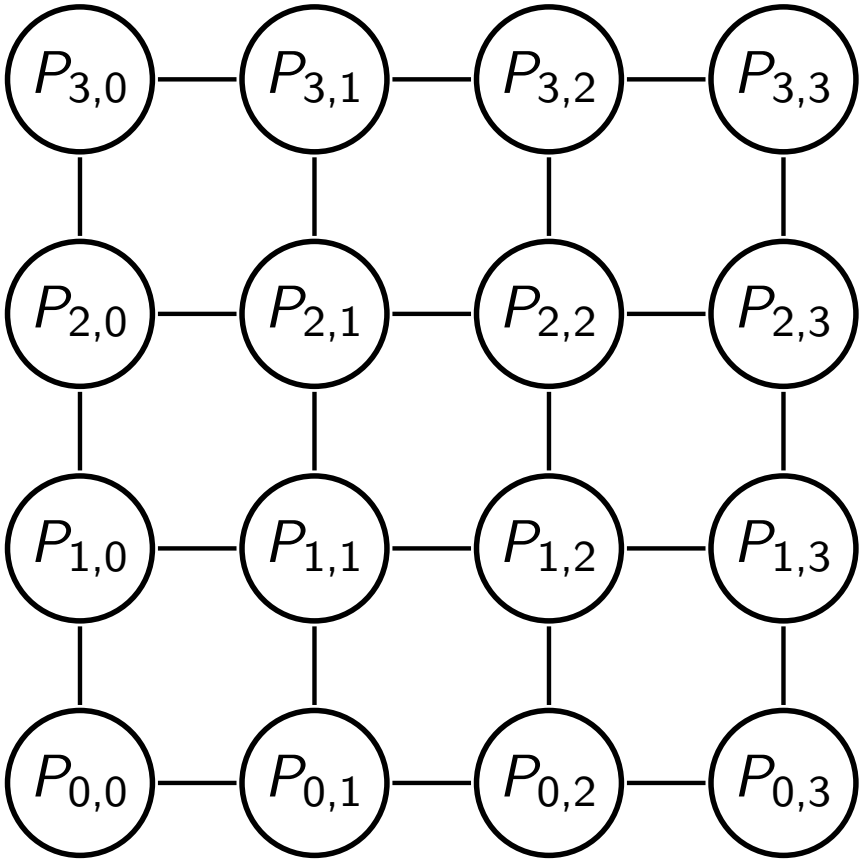
Another simple arrangement is called a ***linear array***, where the ith processor is connected to the $(i-1)$st and the $(i+1)$st processor, except that the first is only connected to the second, and the last is only connected to the penultimate processor. If we label our processors $P_1, \ldots, P_n$, the linear array is shown below:



The disadvantage of the linear array is that communication between two processors requires sending signals across many intermediate processors, i.e., requiring many ***hops***.

Mesh Networks

If there are $n = m^2$ processors, they can be arranged in a ***mesh***. The processors are labeled $P_{i,j}$, $0 \leq i, j \leq m - 1$, and each $P_{i,j}$ is connected to $P_{i\pm1,j\pm1}$, i.e., four other processors, as long as they are in the mesh.



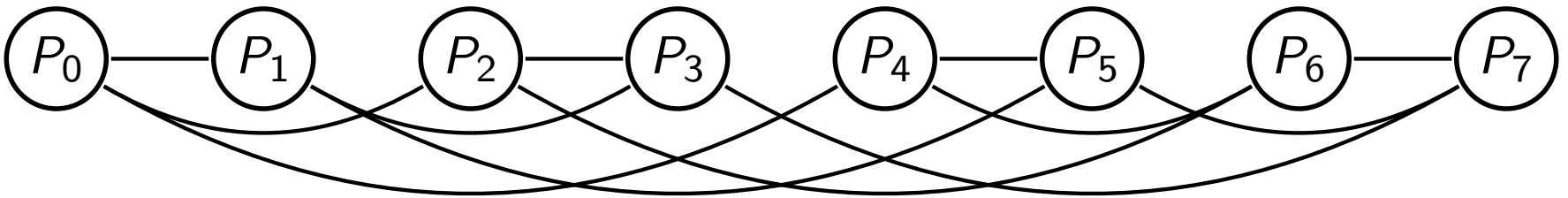
Nodes at the edges of the network are only connected to two or three other nodes.

There is a generalization of the mesh network that ensures that every node is connected to exactly four other nodes (see exercises).

Communication between two arbitrary nodes requires $O(\sqrt{n}) = O(m)$ intermediate links.

Hypercube Networks

Many networks with $n = 2^m$ nodes are arranged in ***hypercubes***. The hypercube configuration balances the number of direct connections of each processor with the number of intermediate connections required for communication. A hypercube network for $m = 3$ is shown below:



This is another representation of the hypercube in Example 3.1.18.

Subgraphs

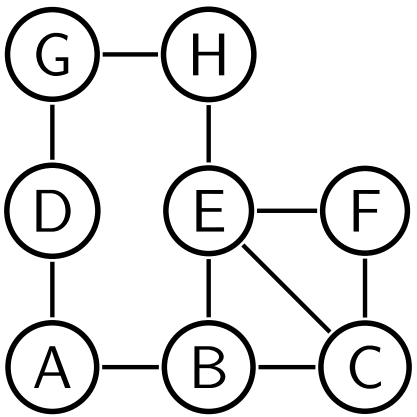
3.1.26. Definition. let $G = (V, E)$ be a graph. Then $H = (W, F)$ is called a ***subgraph*** of G if $W \subset V$ and $F \subset E$. If $H \neq G$ then H is called a ***proper subgraph*** of G.

3.1.27. Example. The cycle C_6 is a subgraph of the wheel W_6 which is a subgraph of the complete graph K_7.

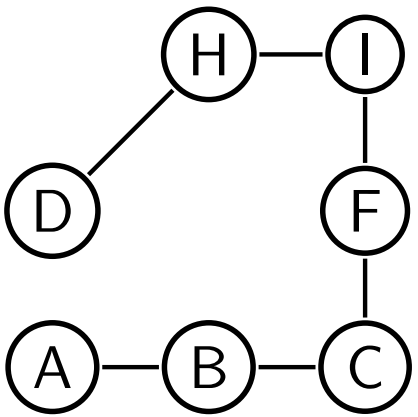
Union of Graphs

3.1.28. Definition. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs. Then the ***union*** of G_1 and G_2, denoted by $G_1 \cup G_2$, is the simple graph with vertices $V_1 \cup V_2$ and edges $E_1 \cup E_2$.

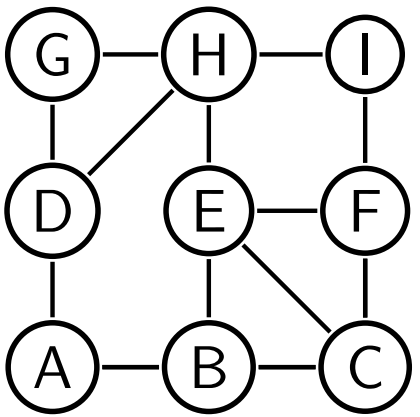
3.1.29. Example.



G_1



G_2

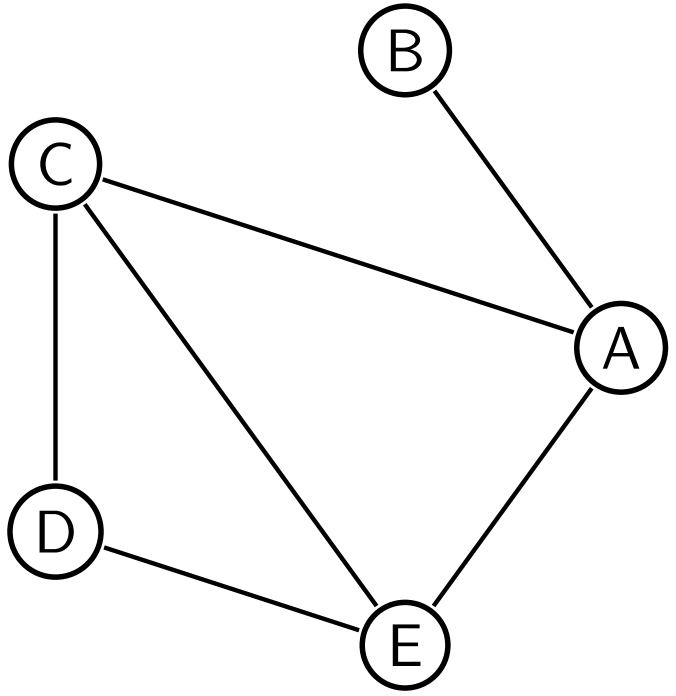


$G_1 \cup G_2$

Adjacency Tables

We will now introduce several ways of representing finite graphs. A simple way to do so is using an ***adjacency table***, which lists all the vertices of the graph and the vertices adjacent to them.

3.1.30. Example. The following is a simple graph and its adjacency table:

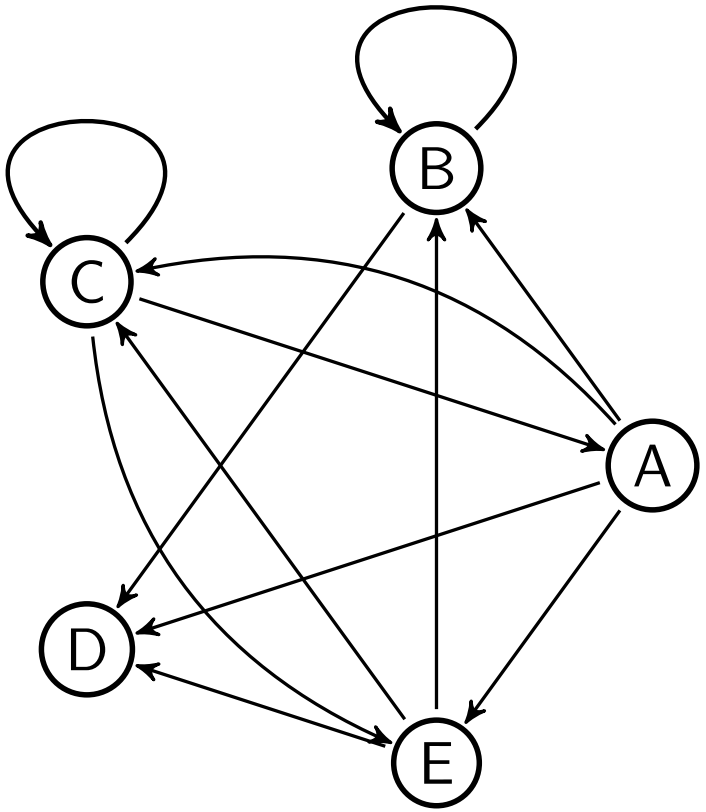


Vertex	Adjacent Vertices
A	B, C, E
B	A
C	A, D, E
D	C, E
E	A, C, D

Adjacency Tables

A similar table can be used for directed graphs.

3.1.31. Example. The following is a directed graph and its adjacency table:



Initial Vertex	Terminal Vertices
A	B, C, D, E
B	B, D
C	A, C, E
D	
E	B, C, D

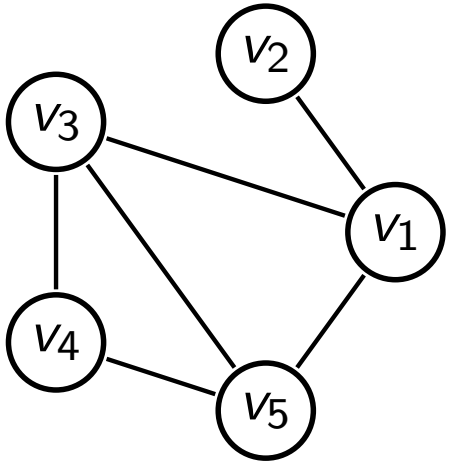
Adjacency Matrices

Instead of tables, matrices can also be used to describe graphs. Assume that $G = (V, E)$, where $V = (v_1, \dots, v_n)$ has been ordered in some way. We then define the ***adjacency matrix*** A_G by

$$A_G = (a_{ij})_{i,j=1}^{n}, \qquad a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the adjacency matrix is always symmetric, i.e., $a_{ij} = a_{ji}$.

3.1.32. Example. The following is a simple graph and its adjacency matrix:



$$A_G = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Adjacency Matrices

In the case of simple graphs, adjacency matrices are zero-one matrices with vanishing trace. However, they can also be used to describe multigraphs and pseudographs, in which case the entries a_{ij} are equal to the multiplicity of the edge $\{v_i, v_j\}$. The adjacency matrix remains symmetric for any type of undirected graph.

An adjacency matrix A_G for a directed graph $G = (V, E)$ with card $V = n$ is defined by

$$A_G = (a_{ij})_{i,j=1}^{n}, \qquad a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, it does not need to be symmetric.

Adjacency Matrices

For card $V = n$, an adjacency matrix will contain n^2 entries, while an adjacency table will have $c \cdot n$ entries, where $c = \max_{1 \leq i \leq n} \deg(v_i)$.

Therefore, if a graph has only few edges, $c \ll n$, (it is ***sparse***) it may be more efficient to use adjacency tables to describe the graph. On the other hand, in this case most of the entries of the adjacency matrix will be zero (such a matrix is also called sparse) and there are special techniques to work with sparse matrices efficiently.

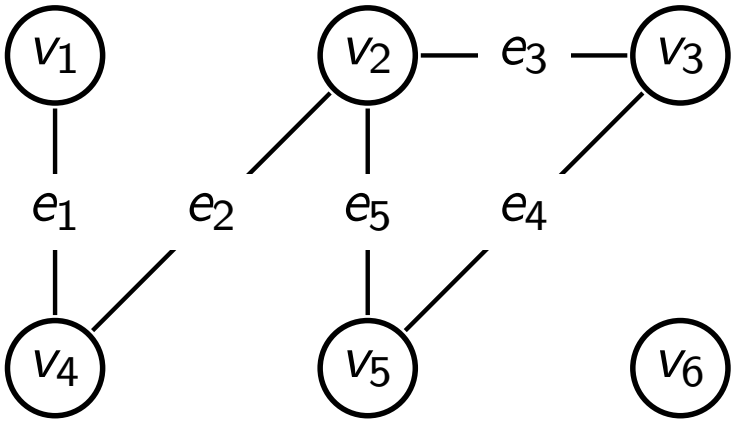
If the graph is ***dense*** (contains many edges) it is preferable to use an adjacency matrix instead of an adjacency table as information is contained in a more accessible form in the matrix.

Incidence Matrices

Graphs can also be represented using ***incidence matrices***. If $G = (V, E)$ is an undirected (pseudo-)graph, $V = \{v_1, \dots, v_n\}$, and $E = \{e_1, \dots, e_m\}$, we define $M_G = (m_{ij})$ by

$$m_{ij} = \begin{cases} 1 & \text{when the edge } e_j \text{ is incident with the vertex } v_i \\ 0 & \text{otherwise.} \end{cases}$$

3.1.33. Example. The following is a simple graph and its incidence matrix:



$$M_G = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Isomorphisms of Graphs

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be simple graphs. Suppose we have a bijective map

$$\varphi \colon V_1 \to V_2,$$

i.e., to every vertex in V_1 we associate precisely one vertex in V_2 and vice-versa. This map induces an injective map

$$\varphi_* \colon E_1 \to V_2 \times V_2, \qquad (a, b) \mapsto (\varphi(a), \varphi(b)).$$

If $\operatorname{ran} \varphi_* = E_2$, we can regard the graph G_2 as being the image of G_1 under (φ, φ_*), where φ maps the vertices of G_1 into those of G_2 and φ_* does the same for the edges.

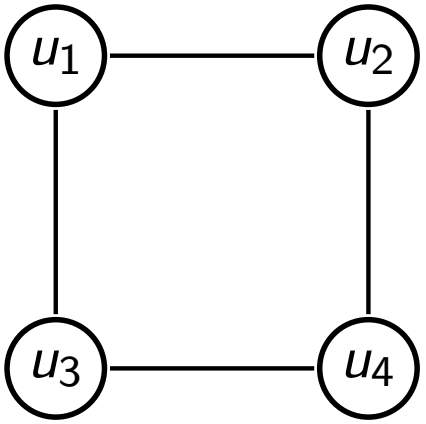
Isomorphisms of Graphs

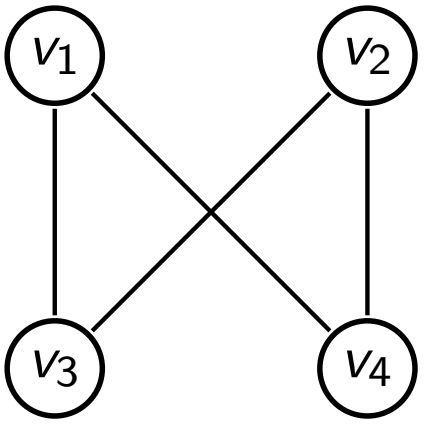
3.1.34. Definition. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two simple graphs. Then we say that G_1 and G_2 are ***isomorphic*** if there exists a bijective function $\varphi\colon V_1 \to V_2$ such that the induced map

$$\varphi_*\colon E_1 \to E_2, \qquad (a, b) \mapsto (\varphi(a), \varphi(b))$$

is bijective. Such a function φ is called a ***(graph) isomorphism***.

3.1.35. Example. Consider the two graphs $G = (U, E)$, $H = (V, F)$ below:





Isomorphisms of Graphs

We now show that the graphs G and H are isomorphic: define $\varphi\colon U \to V$ by setting

$$\varphi\colon u_1 \mapsto v_1, \qquad u_2 \mapsto v_4, \qquad u_3 \mapsto v_3, \qquad u_4 \mapsto v_2.$$

Now

$$E = \big\{\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_4\}, \{u_3, u_4\}\big\}$$

and

$$\begin{aligned}\varphi_* E &= \big\{\{\varphi(u_1), \varphi(u_3)\}, \{\varphi(u_1), \varphi(u_2)\}, \{\varphi(u_3), \varphi(u_4)\}, \{\varphi(u_2), \varphi(u_4)\}\big\} \\ &= \big\{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}\big\} \\ &= F\end{aligned}$$

Therefore, φ is a graph isomorphism and G and H are isomorphic.

Isomorphisms of Graphs

An equivalent way to see that φ is an isomorphism is to compare the adjacency matrix for (U, E) with that of $(\varphi(U), F)$. In the previous example,

$$\begin{aligned} U &= (u_1, u_2, u_3, u_4), \\ \varphi(U) &= (v_1, v_4, v_3, v_2), \\ E &= \big\{\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_4\}, \{u_3, u_4\}\big\}, \\ F &= \big\{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}\big\}. \end{aligned}$$

If the matrices are identical, then φ is an isomorphism. In our case, we have

$$A_{(U,E)} = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccc} u_1 & u_2 & u_3 & u_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} = \begin{array}{c} \\ v_1 \\ v_4 \\ v_3 \\ v_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_4 & v_3 & v_2 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} = A_{(\varphi(U),F)}$$

Graph Invariants

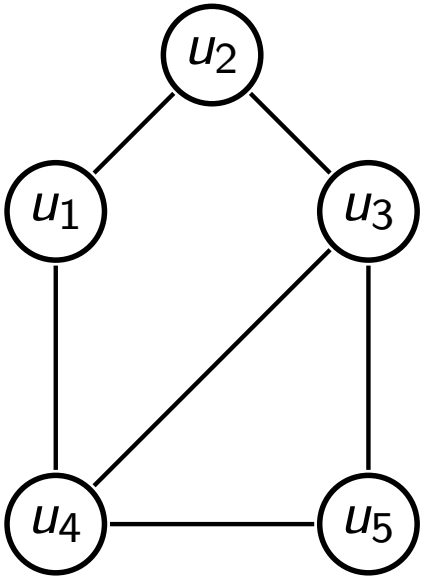
Properties of graphs that are preserved by graph isomorphisms are called ***graph invariants***. Two obvious examples of graph invariants are

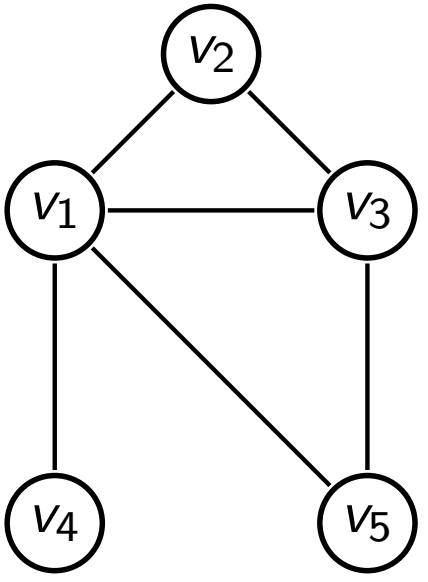
- The number of edges of a graph,
- The total number of vertices in a graph

The second invariant can be made more precise:

- For any $k \in \mathbb{N}$, the number of vertices of degree k is a graph invariant.

3.1.36. Example. Consider the two graphs $G = (U, E)$, $H = (V, F)$ below:

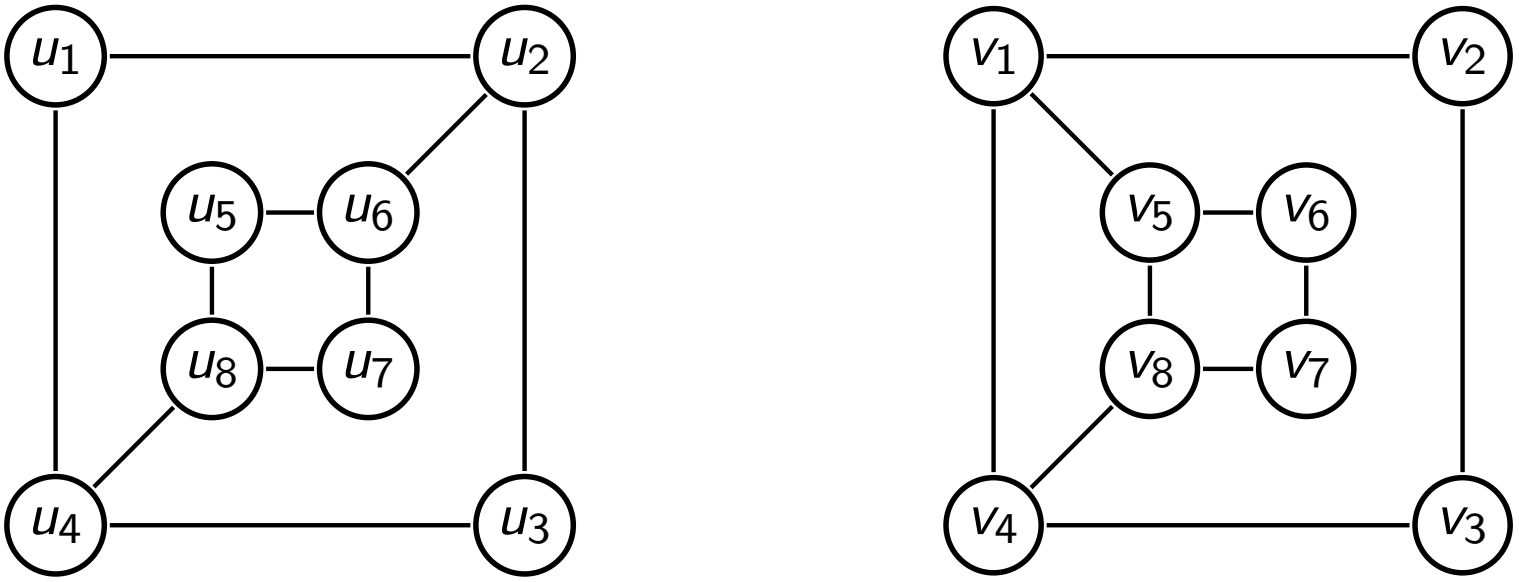




Graph Invariants

The two graphs are not isomorphic because H has a vertex of degree 4 (v_1), while G has no such vertex.

3.1.37. Example. Consider the two graphs $G = (U, E)$, $H = (V, F)$ below:



Both graphs have the same number of vertices of degrees 1, 2 and 3, so they might conceivably be isomorphic. In order to show that they are in fact not isomorphic, we need a stronger argument.

Isomorphism of Subgraphs

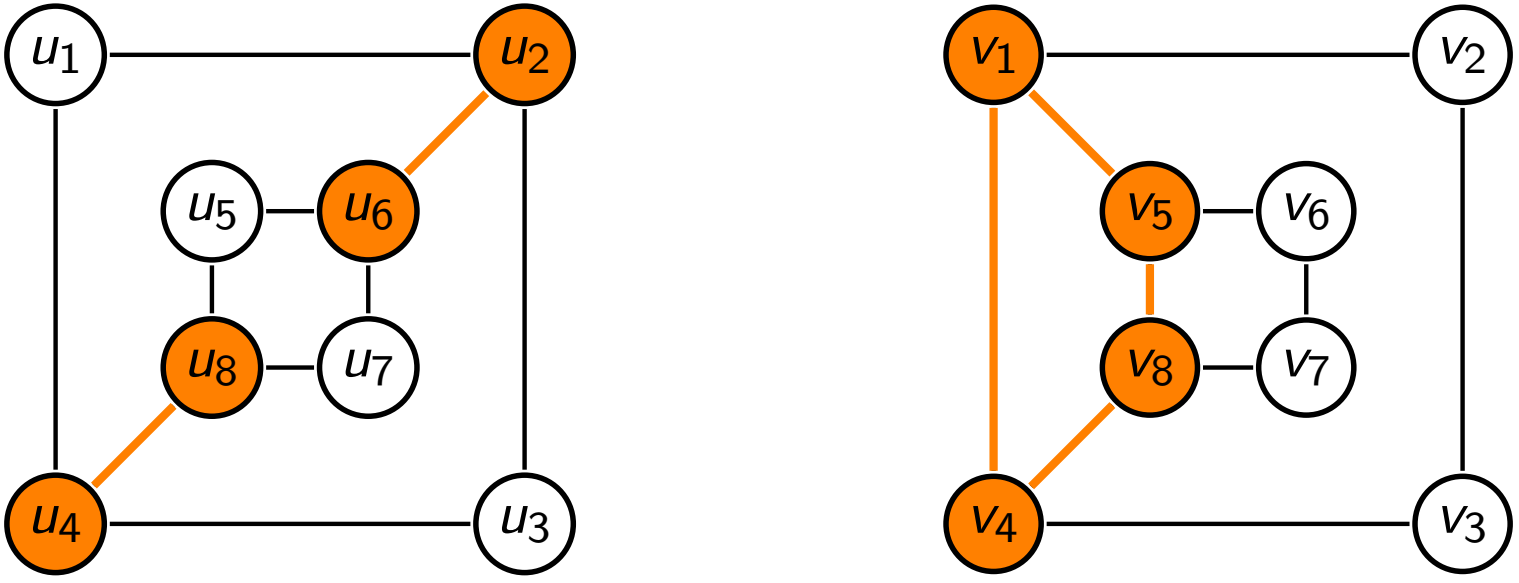
Suppose that there exists a graph isomorphism $\varphi\colon U \to V$, $\varphi_*\colon E \to F$. Let $U' \subset U$ and $G' = (U', E')$ be a subgraph of (U, E). Then φ restricted to U', denoted $\varphi|_{U'}$, has range contained in V and induces an isomorphism onto a subgraph of (V, F).

In our example, consider the subgraph (U', E') consisting of all vertices of degree $k \in \mathbb{N}$. Then a graph isomorphism φ, if one exists, can be restricted to U' and its range must be the subgraph of V consisting of vertices of degree k. We hence make the following note:

3.1.38. Lemma. If two graphs are isomorphic, then all subgraphs consisting of vertices of degree $k \in \mathbb{N}$ must be isomorphic.

Isomorphism of Subgraphs

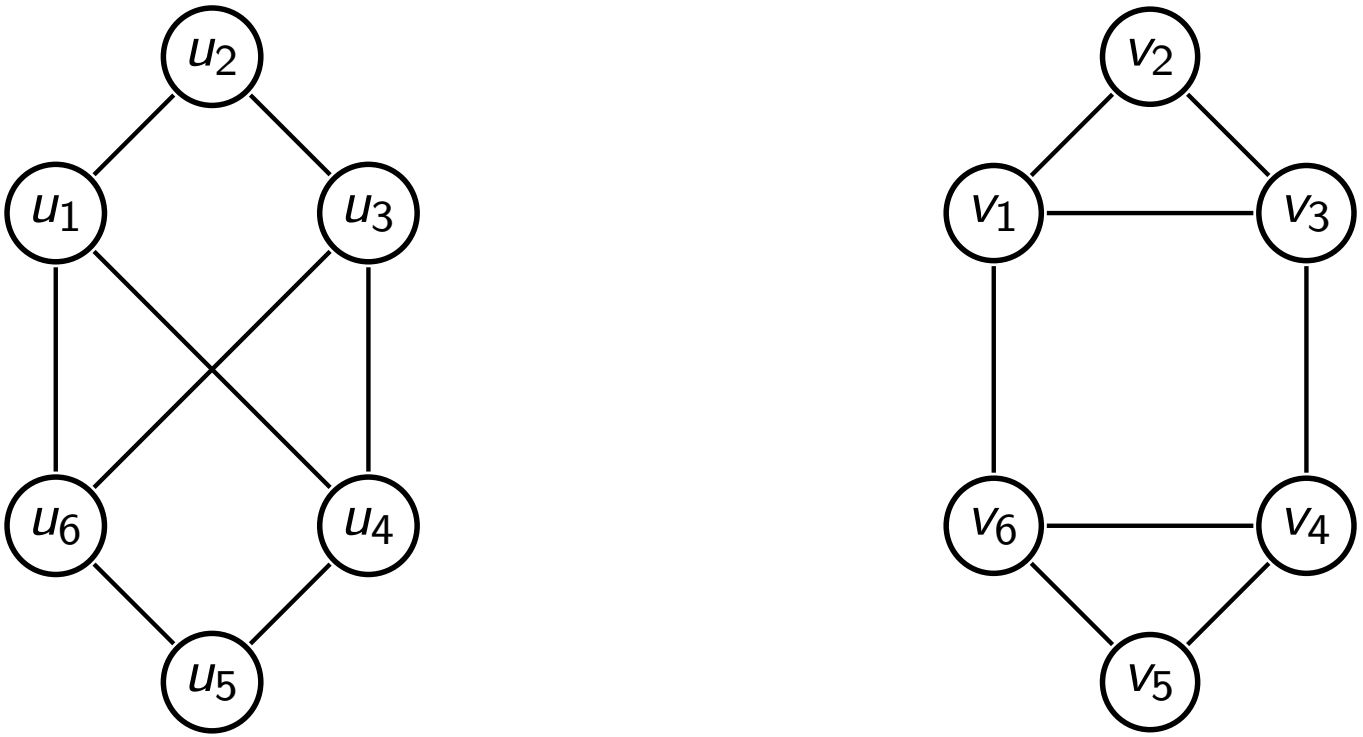
Returning to Example 3.1.37, the subgraphs consisting of vertices of degree 3 and their common edges are marked in orange:



It is obvious that the two subgraphs are not isomorphic, so the graphs G and H can not be isomorphic either.

A More Difficult Problem of Isomorphism

3.1.39. Example. Consider the two graphs $G = (U, E)$, $H = (V, F)$ below:



Here the subgraphs of of vertices of degree 2 and 3 are isomorphic (for the latter, see Example 3.1.35). However, we may suspect that G and H are not themselves isomorphic; proving this requires us to introduce the concept of paths and circuits, which we do in the following section.

Paths

3.2.1. Definition. Let $G = (U, E)$ be an undirected graph. Let $u, v \in U$.

- A ***path of length*** $n = 1$ ***from*** u ***to*** v is a set containing the single edge $e \in E$ associated with $\{u, v\}$.
- A ***path of length*** $n \geq 2$ ***from*** u ***to*** v is a tuple of edges $(e_1, e_2, \ldots, e_n)$ such that
  - e_1 is associated with $\{u, x_1\}$,
  - e_k is associated with $\{x_{k-1}, x_k\}$ for $k = 2, \ldots, n-1$,
  - e_n is associated with $\{x_{n-1}, v\}$

  where $x_1, \ldots, x_{n-1} \in U$ and $e_1, \ldots, e_n \in E$.
- A path is said to ***pass through*** the vertices $u, x_1, \ldots, x_{n-1}, v$ and ***traverse*** the edges $e_1, \ldots, e_n$.
- If G is a simple graph, we denote the path simply by the tuple of vertices it passes through, i.e., by $(u, x_1, \ldots, x_{n-1}, v)$.
- A path with $u = v$ is called a ***circuit***.
- If the tuple of edges does not contain any edge more than once, the path is said to be ***simple***.

The Number of Paths Joining Two Vertices

The number of paths from one vertex to another can be found from the adjacency matrix:

3.2.2. Theorem. Let G be a multigraph with adjacency matrix A_G and ordered set of vertices $V = (v_1, \ldots, v_n)$. Then the number of different paths of length $k \in \mathbb{N} \setminus \{0\}$ between v_i and v_j is equal to the (i, j)th coefficient of A_G^k.

Proof.
We will prove the theorem by induction. For $k = 1$ the statement is obviously true.

We now assume that the (i, j)th entry of A^k gives the number of paths connecting v_i to v_j. Since $A^{k+1} = A^k \cdot A$, the (i, j)th entry of A^{k+1} is

$$b_{i1}a_{1j} + b_{i2}a_{2j} + \cdots + b_{in}a_{nj} \tag{3.2.1}$$

where $A^k = (b_{ij})_{1 \leq i,j \leq n}$.

Simple Circuits as Graph Invariants

Proof.

A path of length $k+1$ from v_i to v_j consists of a path of length k from v_i to some v_m and a path of length 1 from v_m to v_j. Summing over the product of all paths of length 1 from $v_m \in V$ to v_j with the number of all paths of length k from v_i to v_m gives exactly (3.2.1). □

In the assignments, you will prove the following result:

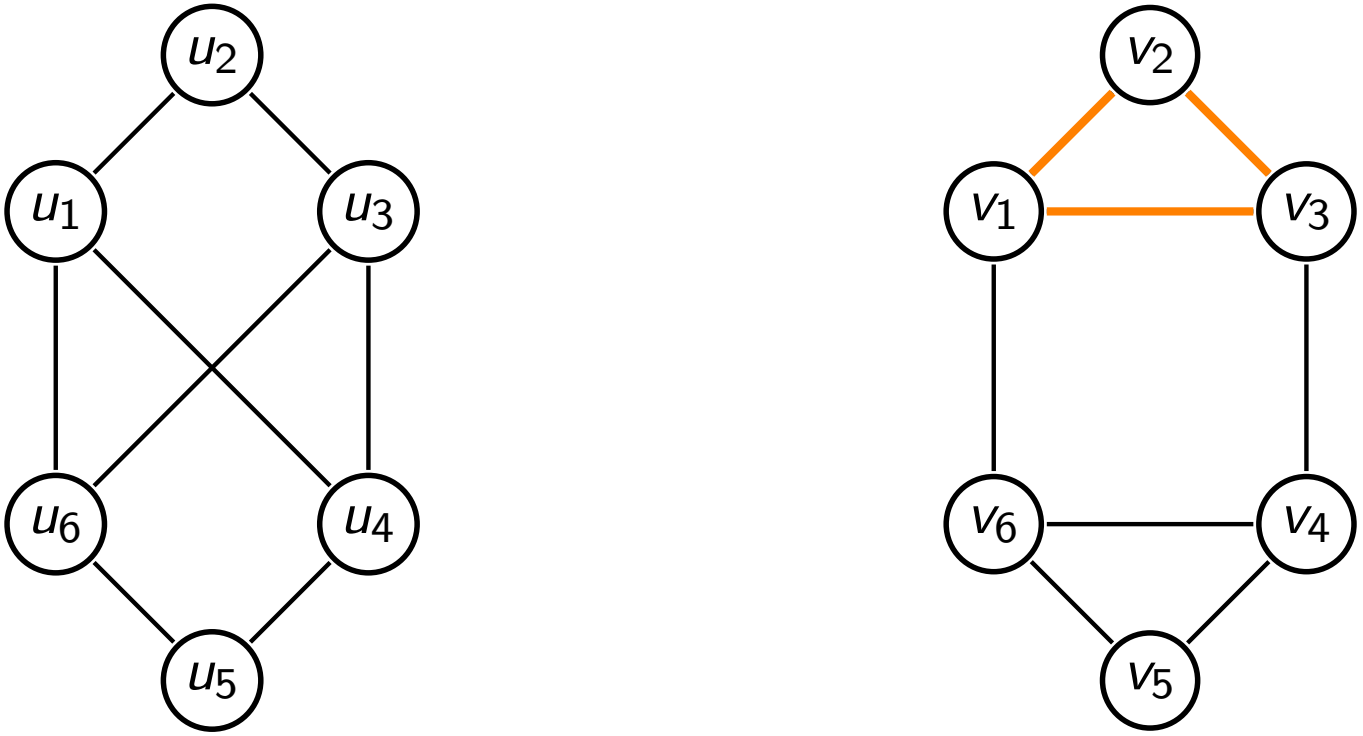
3.2.3. Theorem. The existence of a simple circuit of length $k \in \mathbb{Z}_+$ is a graph invariant.

This means that if two graphs are isomorphic, then they must either both have or both not have a simple path of length k for each $k \in \mathbb{Z}_+$.

A graph with adjacency matrix A will have a circuit of length k if $\operatorname{tr} A^k \neq 0$.

Simple Circuits as Graph Invariants

3.2.4. Example. Consider again the graphs shown below:



The graph on the right has a simple circuit of length 3, while the graph on the left does not appear to have such a circuit. To see that this is the case, we need to write out the adjacency matrix for the graph on the left.

Simple Circuits as Graph Invariants

$$A = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{array} \begin{array}{c} \begin{array}{cccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

An easy calculation shows that

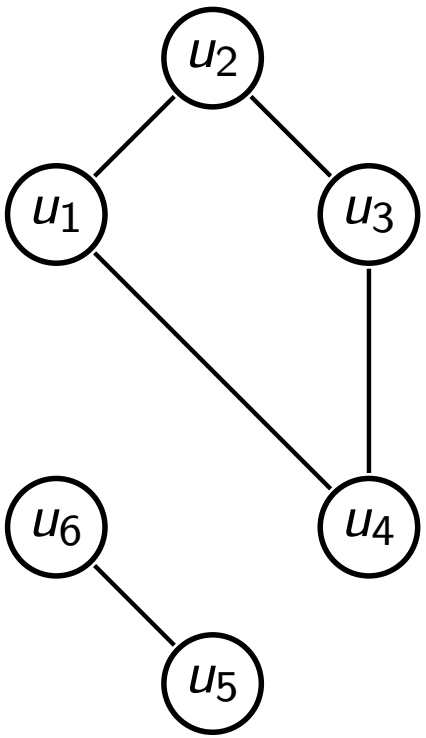
$$A^3 = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{array} \begin{array}{c} \begin{array}{cccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{array} \\ \begin{pmatrix} 0 & 6 & 0 & 8 & 0 & 8 \\ 6 & 0 & 6 & 0 & 4 & 0 \\ 0 & 6 & 0 & 8 & 0 & 8 \\ 8 & 0 & 8 & 0 & 6 & 0 \\ 0 & 4 & 0 & 6 & 0 & 6 \\ 8 & 0 & 8 & 0 & 6 & 0 \end{pmatrix} \end{array}$$

Connectedness

3.2.5. Definition. An undirected graph is called ***connected*** if there is a path between any two distinct vertices.

A ***connected component*** of a graph is a connected subgraph that is not a proper subgraph of another connected subgraph.

3.2.6. Example. Consider the graph $G = (U, E)$ shown below:



The graph is not connected, because there is no path from u_1 to u_5 that traverses edges in E.

There are two connected components: the subgraph given by $U' = \{u_5, u_6\}$ with edge set $E' = \{\{u_5, u_6\}\}$ and the subgraph $U'' = \{u_1, u_2, u_3, u_4\}$ with edge set $E'' = \{\{u_1, u_2\}, \{u_2, u_3\}, \{u_3, u_4\}, \{u_1, u_4\}\}$.

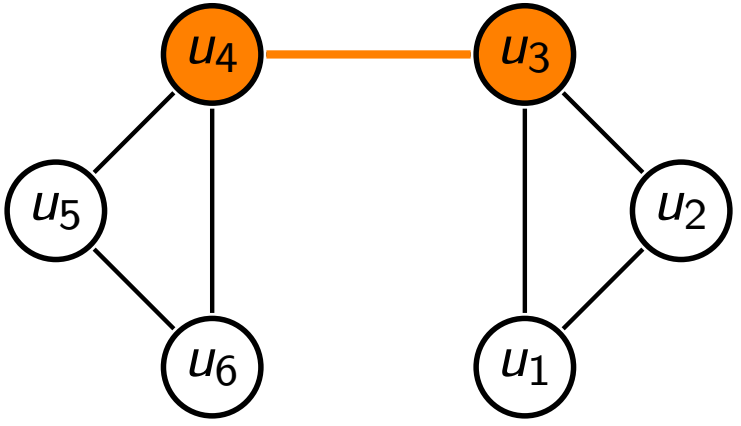
$U''' = \{u_1, u_2, u_3\}$ with $E''' = \{\{u_1, u_2\}, \{u_2, u_3\}\}$ is not a connected component, because it a subgraph of the connected subgraph (U'', E'').

Cut Vertices and Cut Edges

The removal of a vertex v of a graph G together with all incident edges gives a subgraph of G. If this subgraph has more connected components than G, we say that v is a ***cut vertex*** or an ***articulation point***.

Similarly, an edge whose removal gives a subgraph with more connected components is called a ***cut edge***.

3.2.7. Example. In the graph below, the cut edges and cut vertices have been marked in orange:



Euler Paths and Circuits

3.2.8. Definition. Let G be a finite multigraph.

1. An ***Euler circuit*** in G is a simple circuit containing every edge of G.
2. An ***Euler path*** in G is a simple path containing every edge of G.

Famously, Leonhard Euler studied the seven bridges of Königsberg in 1735. An Euler path corresponds to walking across each bridge exactly once:

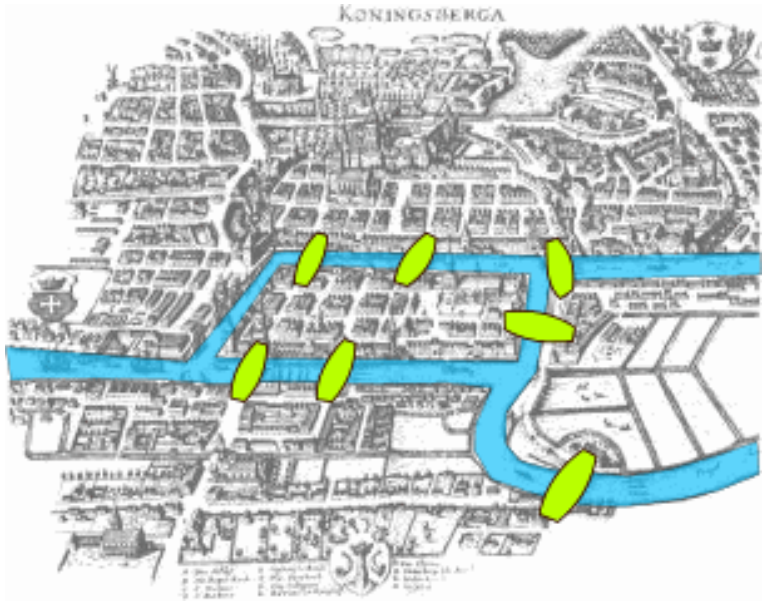
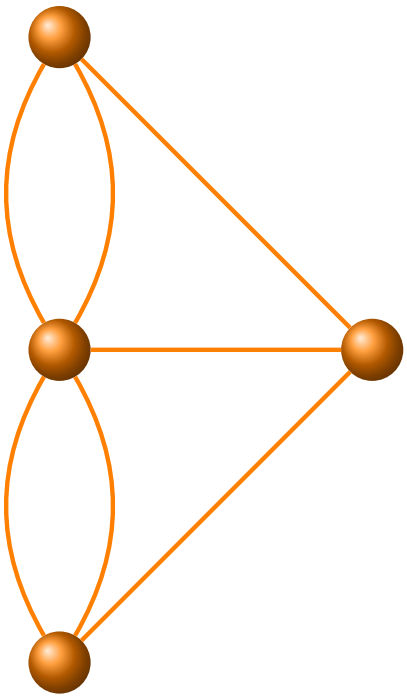


Image source:
http://en.wikipedia.org/wiki/File:Konigsberg_bridges.png.
Used under the license given there



Euler Paths and Circuits

3.2.9. Theorem. A finite connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree.

Proof.

($\Rightarrow$) Suppose that the multigraph has an Euler circuit starting at some vertex v_1. We then "follow" the Euler circuit and count 1 degree whenever we "enter" a vertex and 1 more degree when we "leave" each vertex. We count one degree for v_1 as we start by leaving v_1. Entering some other vertex, we count 1 as we enter and 1 as we leave. Hence, by traversing the edges of the graph along the circuit, we count two degrees whenever we pass through a vertex. Finally, we "enter" the vertex v_1 at the final edge of the circuit, counting another degree. Therefore, each vertex of the graph must have an even degree.

Euler Paths and Circuits

($\Leftarrow$) Suppose the multigraph G is connected and that every vertex has an even degree. We choose some vertex v_0 and arbitrarily choose an edge $\{v_0, v_1\}$. This is possible, because the graph is connected and has at least two vertices. We then choose an edge incident to v_1, which is also incident to some vertex v_2. We continue to choose edges in this way until we reach a vertex v_n which has no more edges incident to it that we have not yet used in our path.

This will occur because G is assumed to be finite and hence contains only a finite number of edges.

We now claim that $v_n = v_0$: The path "uses" one degree of each vertex to enter and one degree to "leave" each vertex. Since every vertex has an even degree, for every edge that is used to "enter" the vertex, there exists exactly one edge to "leave" the vertex. Therefore, if the path arrives at a vertex that can not be "left" by an unused edge, it must be v_0.

Euler Paths and Circuits

We have now constructed a circuit in the graph G. However, it may not be an Euler circuit, because it might not include every edge in G. Let H be the subgraph of G that contains the edges not already used. Then we have an Euler circuit of $H \setminus G$. Let w_0 be a vertex in H that is connected to a vertex in $G \setminus H$. Such a vertex exists because G is connected. Repeat the above procedure to find a circuit in H starting and ending at w_0 and then splice this circuit into the circuit of $H \setminus G$. We then obtain a new (large) circuit in G.

If there remain unused edges, we repeat the above step until no more edges remain and our circuit becomes an Euler circuit in G. Our procedure terminates because the graph G is finite. □

The construction of an Euler circuit can be made into a workable algorithm. We also note that every circuit is of course a path, so the existence of an Euler circuit implies the existence of an Euler path.

Euler Paths and Circuits

3.2.10. Theorem. A finite connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

Proof.

($\Rightarrow$) Suppose that the multigraph has an Euler path starting at some vertex v_1 and ending at some vertex v_n. Then the vertex v_1 must have odd degree, because it is left once more than it is entered. A similar statement applies to v_n. All other vertices are entered and then left an equal number of times. Thus, v_1 and v_n have odd degree while all other vertices have even degree.

($\Leftarrow$) Suppose that the multigraph G has two vertices, v_1 and v_n, of odd degree and that all other vertices have even degree. Add an edge $\{v_1, v_n\}$ to G. Then every vertex in G has even degree and there exists an Euler circuit starting at v_1 and ending at v_n. Removing the edge $\{v_1, v_n\}$ from the Euler circuit produces the desired Euler path. (Why exactly?) □

Hamilton Paths and Circuits

3.2.11. Definition. Let G be a finite graph.

1. A ***Hamilton path*** in G is a simple path that passes through every vertex of G exactly once.
2. A ***Hamilton circuit*** in G is a simple circuit that passes through every vertex of G exactly once.

There are no known useful necessary and sufficient conditions for the existence of Hamilton circuits, but there are a few giving sufficient conditions:

3.2.12. Dirac's Theorem. Let G be a simple graph with $n \geq 3$ vertices such that the degree of each vertex is at least $n/2$. Then G has a Hamilton circuit.

3.2.13. Ore's Theorem. Let G be a simple graph with $n \geq 3$ vertices such that the sum of the degree of any two non-adjacent vertices is at least n. Then G has a Hamilton circuit.

Hamilton Paths and Circuits

Clearly, Dirac's Theorem is a Corollary of Ore's Theorem. Ore's Theorem will be proven in the exercises.

Hamiltonian circuits appear in many applications. A famous problem that we will look at later is the ***traveling salesman problem***, which asks for the perfect route a traveling salesman should take to visit a set of cities.

The Icosian Game

William Hamilton studied such paths and circuits and their symmetry properties. Out of his research he developed a game called the ***Icosian game*** or "A Voyage Round the World" which challenges a person to connect pegs on the edges of a dodecahedron using a string in such a way that all pegs are touched exactly once by the string.

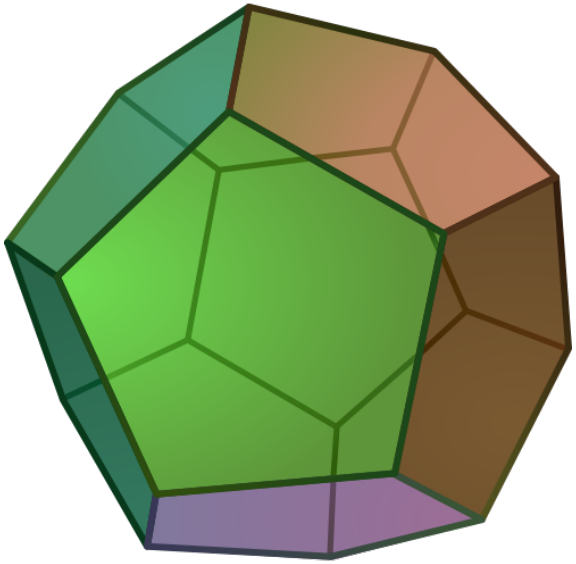


Image source:
http://commons.wikimedia.org/wiki/File:
POV-Ray-Dodecahedron.svg
Used under the license given there

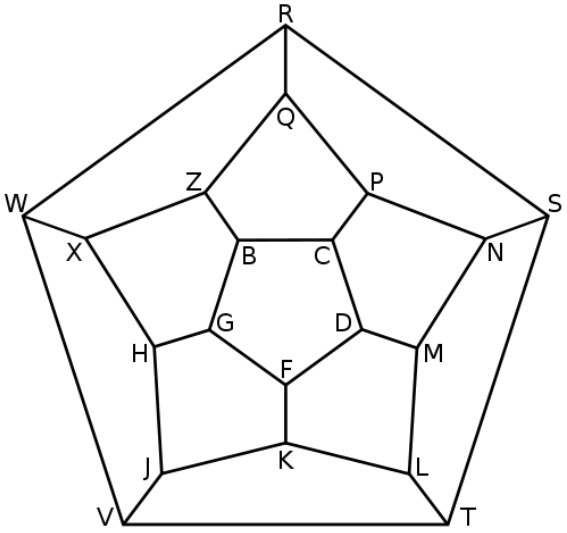


Image source:
http://commons.wikimedia.org/wiki/File:
Icosian_grid_small_with_labels.svg
Used under the license given there

The Icosian Game

A solution is shown below:

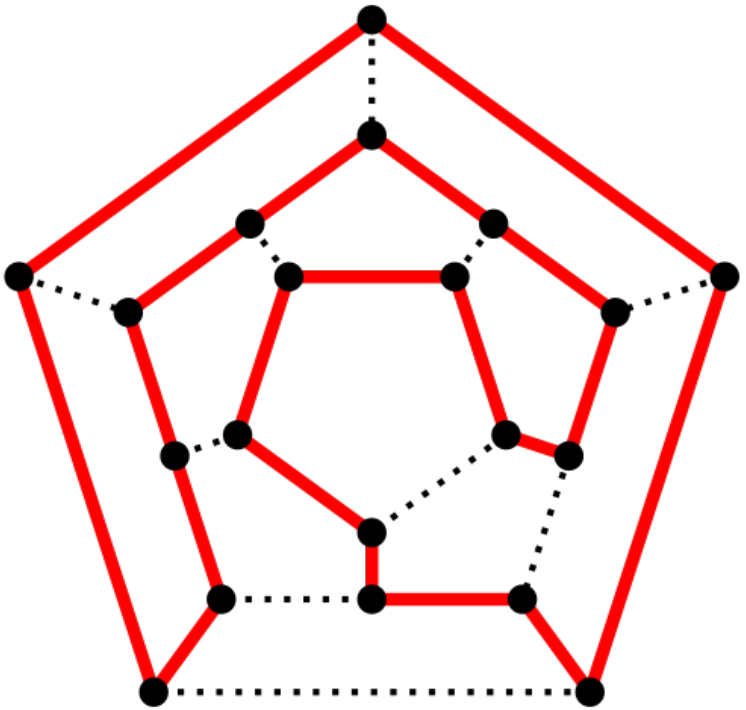


Image source:
http://commons.wikimedia.org/wiki/File:Hamiltonian_path.svg
Used under the license given there

A three-dimensional image of the solution can be found at http://mathworld.wolfram.com/IcosianGame.html

Weighted Graphs

A weighted graph is a graph where each edge is assigned a weight. For example, the image below gives the distances (in miles) between various cities in the United States:

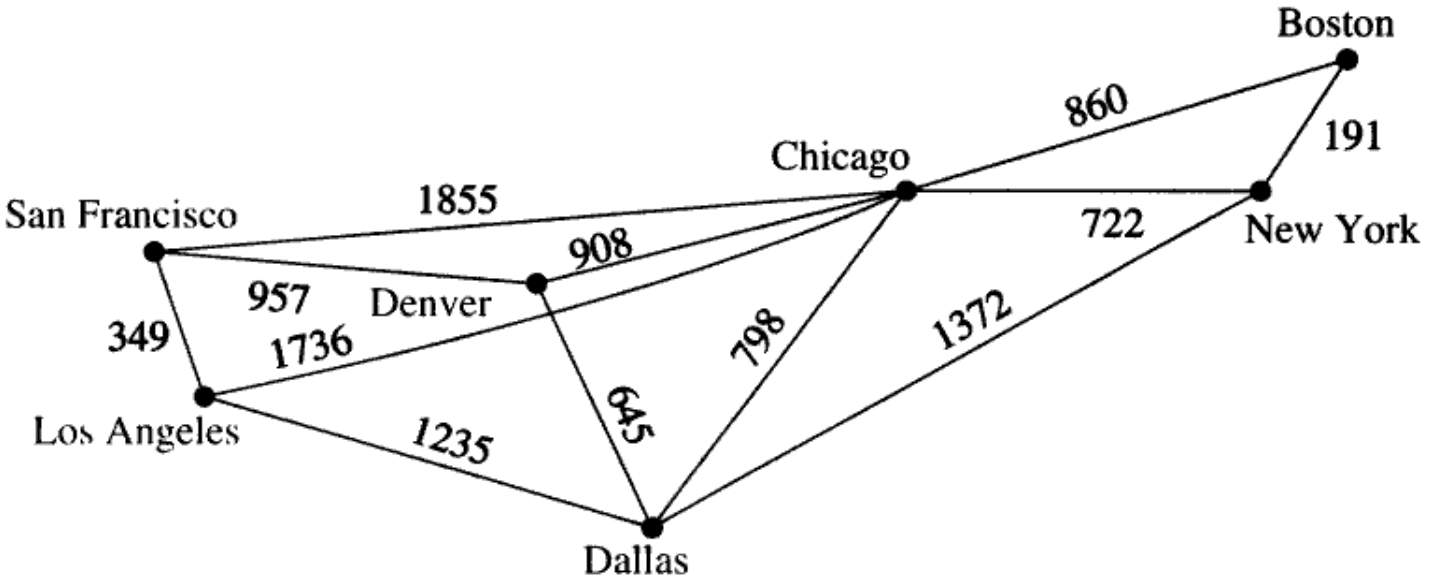


Image source: Rosen, K.H., ***Discrete Mathematics***, 6th Ed., McGraw-Hill International Edition 2007
Used under fair use exemption.

We are interested in paths in weighted graphs. The ***length*** of a path in a weighted graph is given by the sum of the weights of the edges comprising the path. An empty path consisting of no edges is said to have length infinity.

Shortest Paths in Weighted Graphs

Suppose we are given an undirected, connected, simple, weighted path. Given two vertices a and z, our goal is to find the shortest path (i.e., the path of least length) that joins a to z. We will here present an algorithm due to Dijkstra that finds this shortest path.

The algorithm proceeds in several iterations. In the kth iteration, we will form a distinguished set of vertices, called S_k. Furthermore, in each iteration we will assign the labels $L_k(v)$ to the vertex v.

We start with the $k = 0$th iteration, setting

$$S_0 := \emptyset, \qquad L_0(v) = \begin{cases} 0 & v = a, \\ \infty & v \neq a. \end{cases}$$

We find S_{k+1} from S_k, $k \in \mathbb{N}$, by adding a vertex u to S_k with the smallest label. Then, we update the labels of all vertices not in S_{k+1} so that

$$L_{k+1}(v) = \min\big\{L_k(v), L_k(u) + w(u, v)\big\} \tag{3.2.2}$$

where $w(u, v)$ denotes the weight of the edge $\{u, v\}$.

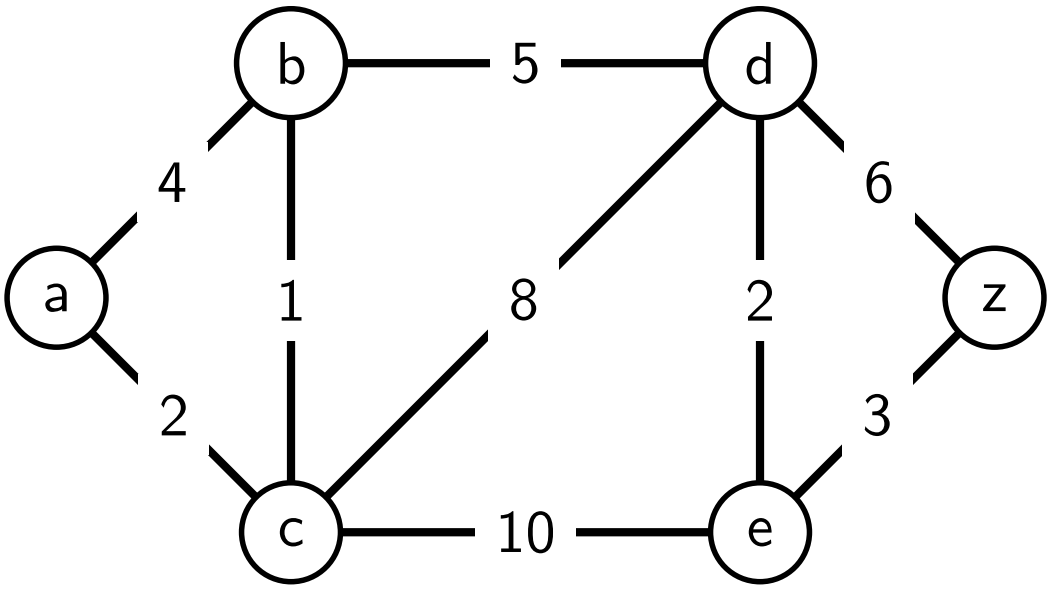
Dijkstra's Algorithm

We claim that

$$L_k(v) = \text{length of the shortest path from } a \text{ to } v \text{ that contains only vertices in } S_k.$$

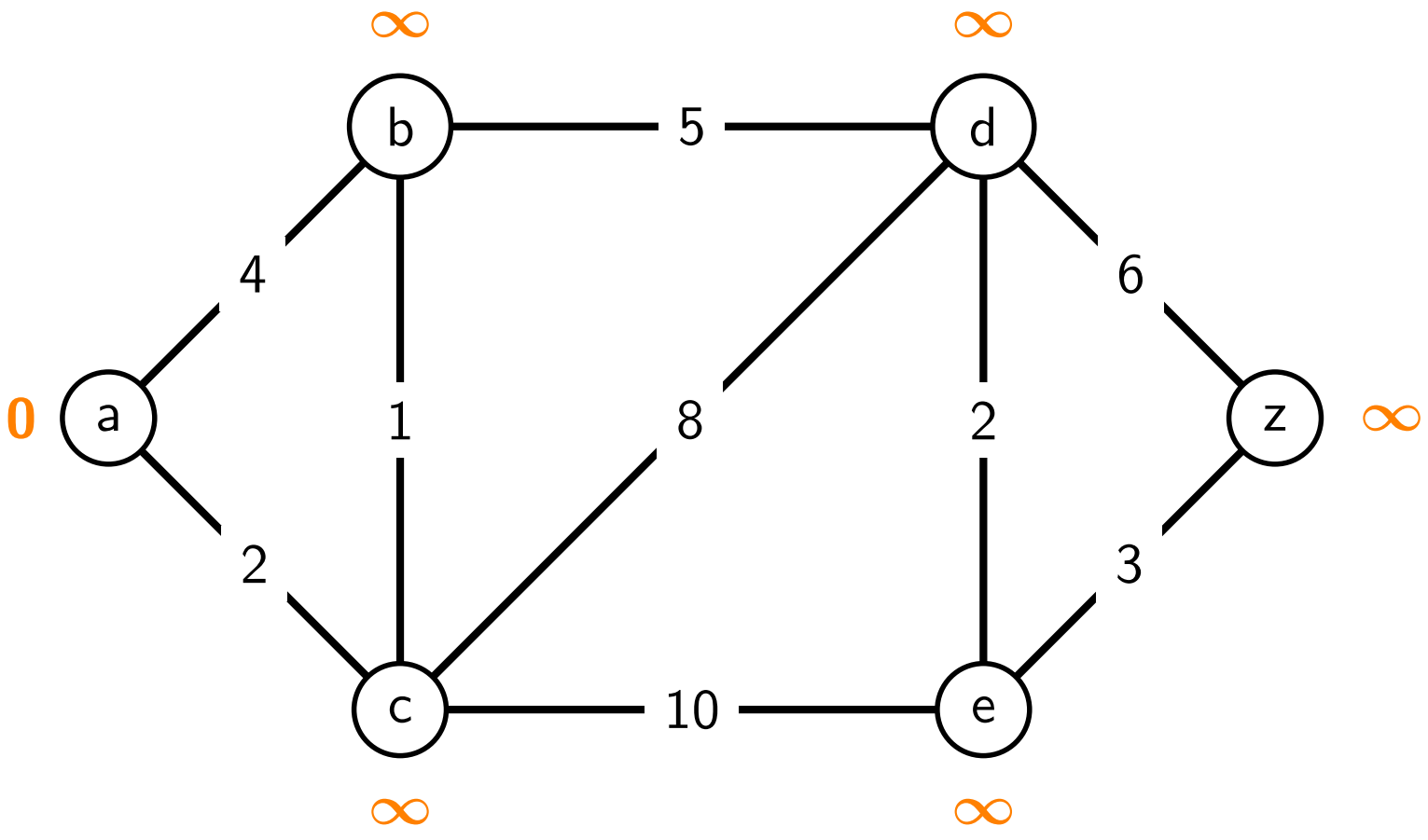
Once z is added to S_k, the length of the shortest path is given by the label of z.

3.2.14. Example. We will find the shortest path from a to z in the graph given below:



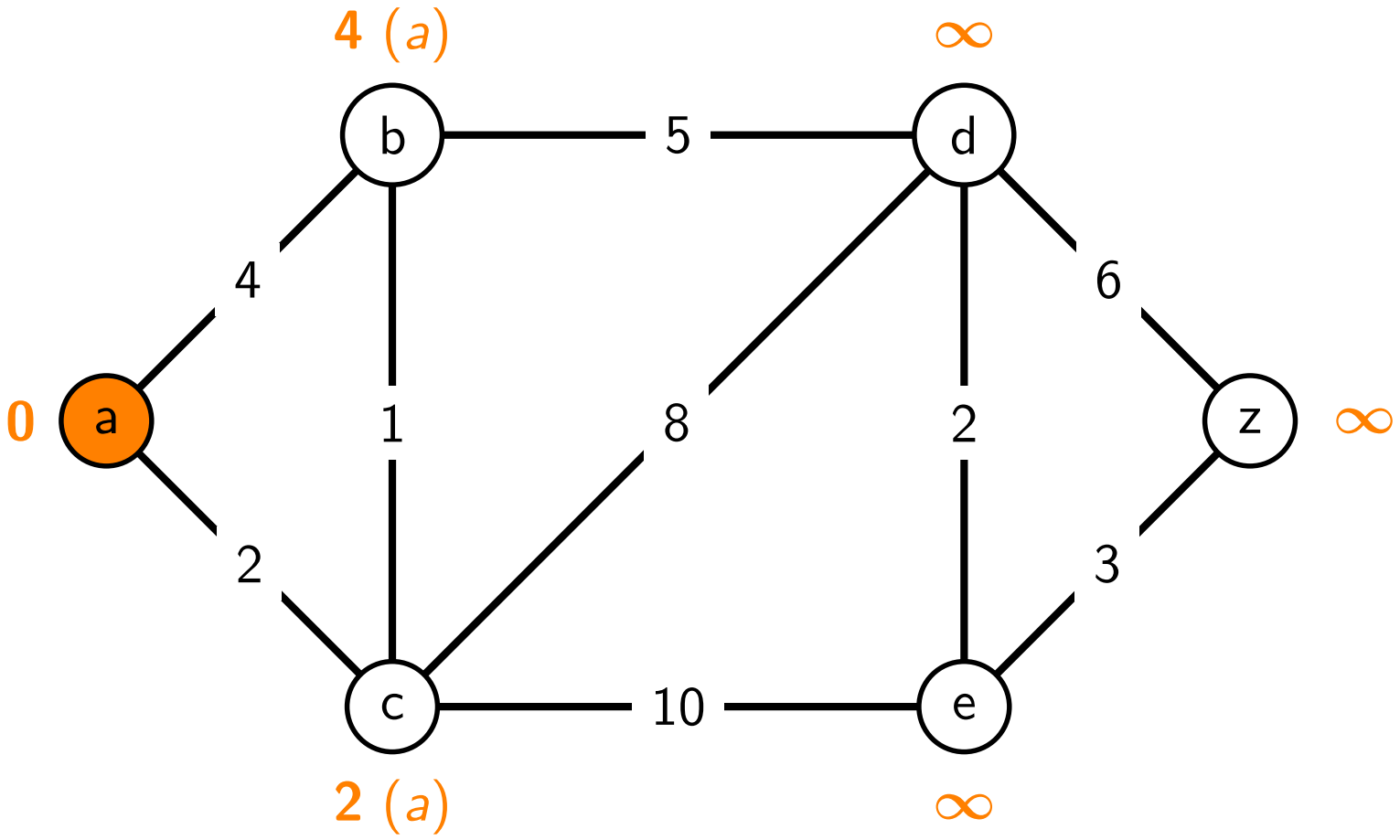
Dijkstra's Algorithm - 0^{th} Iteration

$S_0 = \emptyset$.



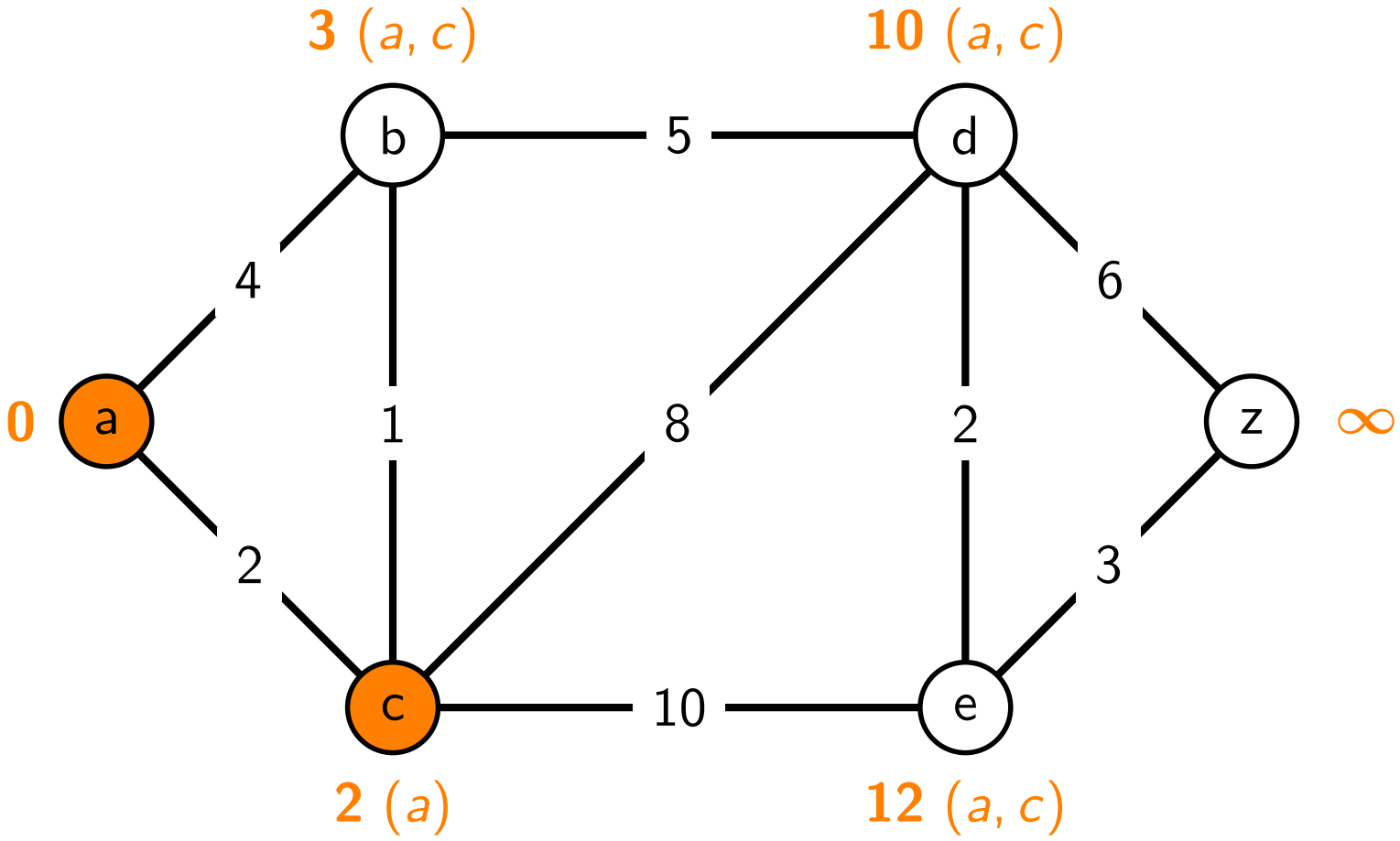
Dijkstra's Algorithm - 1st Iteration

$S_1 = \{a\}$.



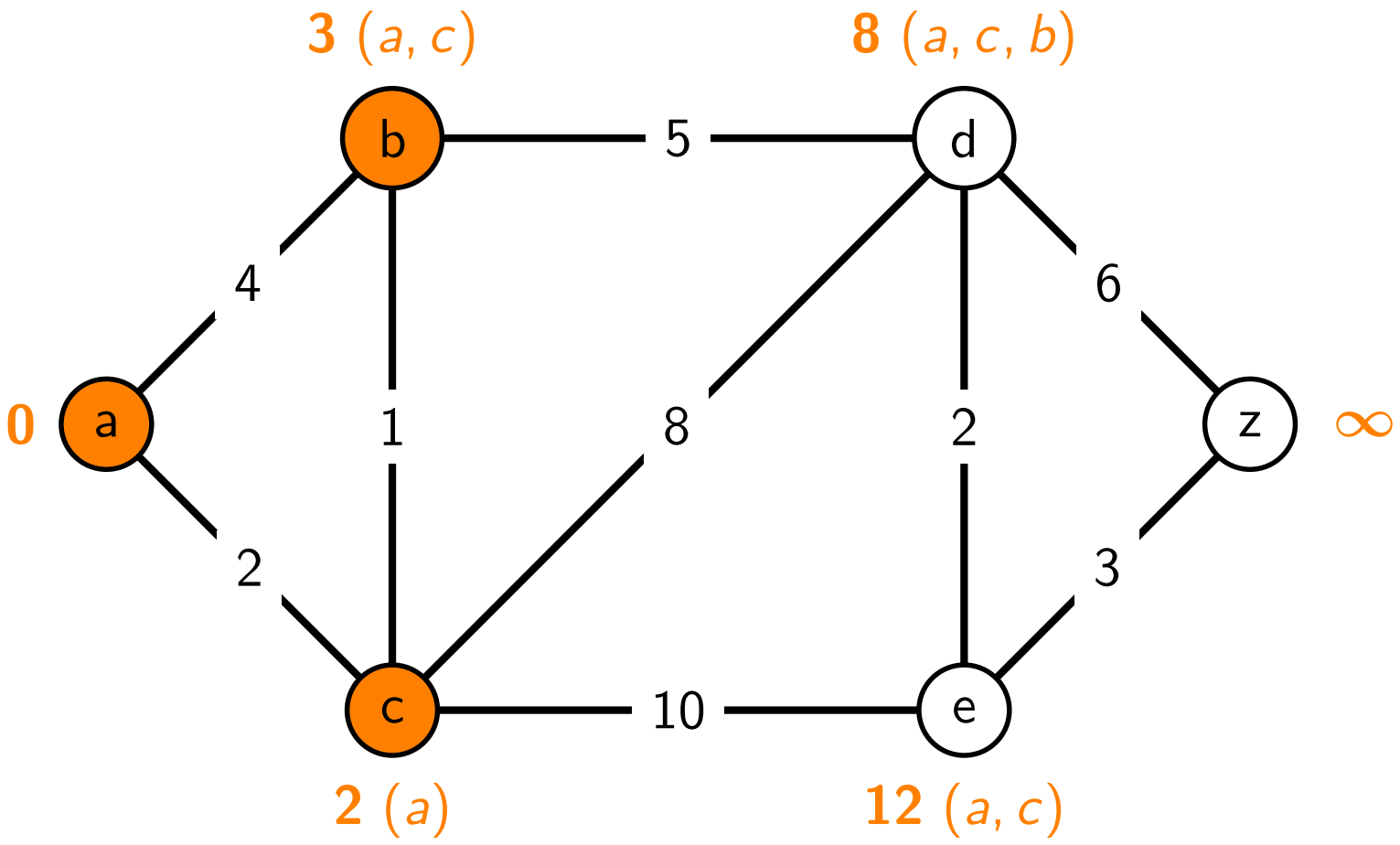
Dijkstra's Algorithm - 2nd Iteration

$S_2 = \{a, c\}$.



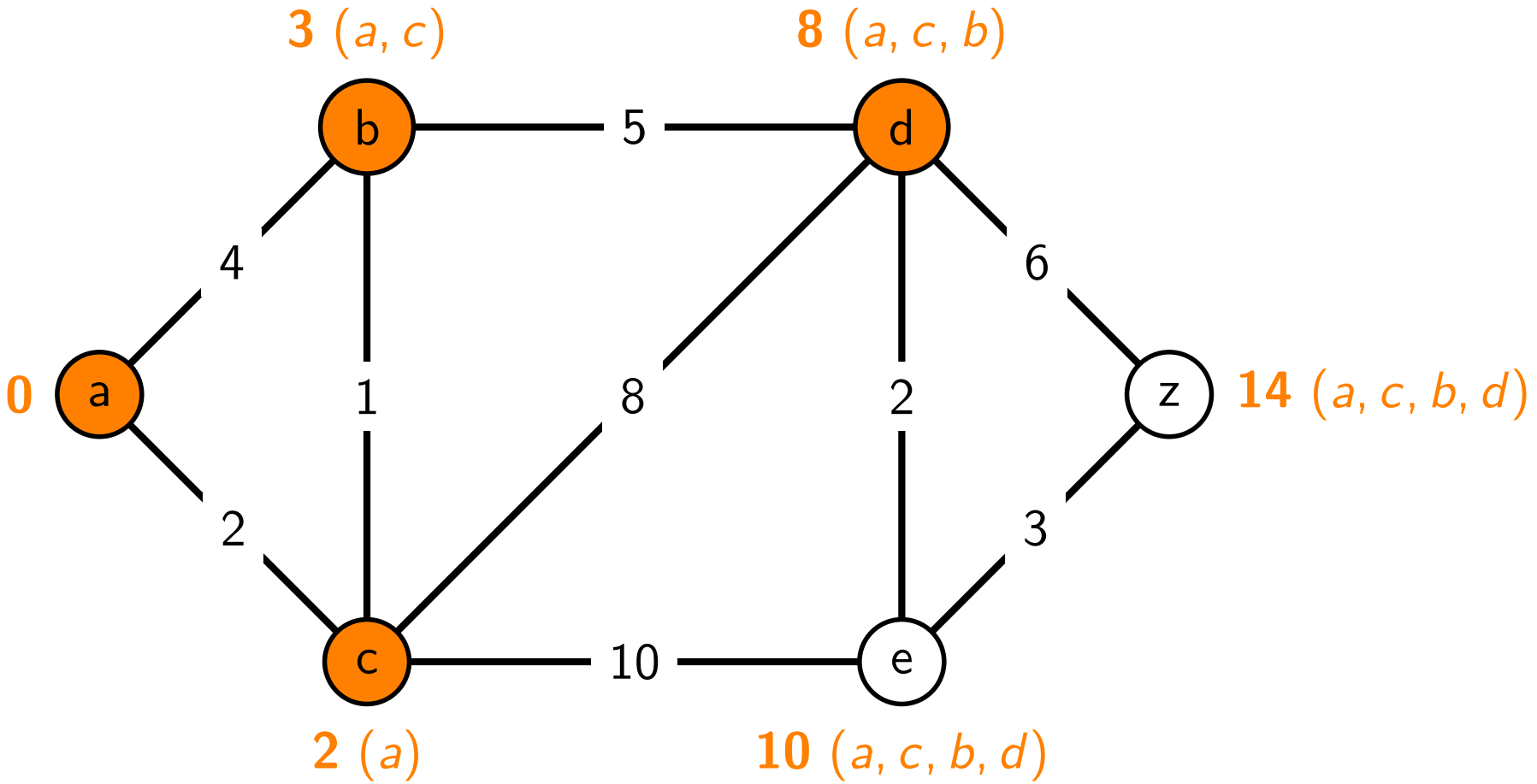
Dijkstra's Algorithm - 3rd Iteration

$S_3 = \{a, b, c\}$.



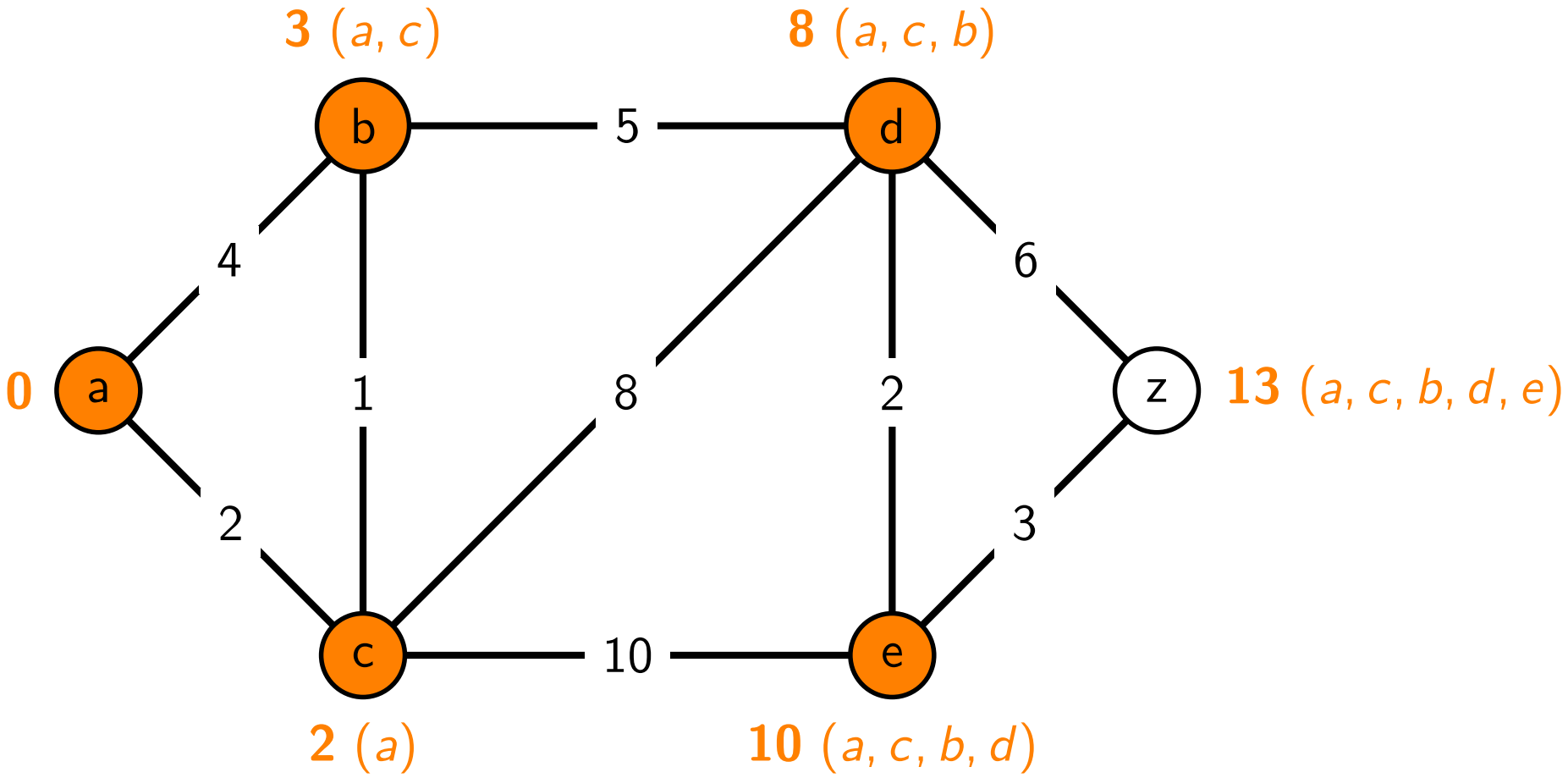
Dijkstra's Algorithm - 4th Iteration

$S_4 = \{a, b, c, d\}$.



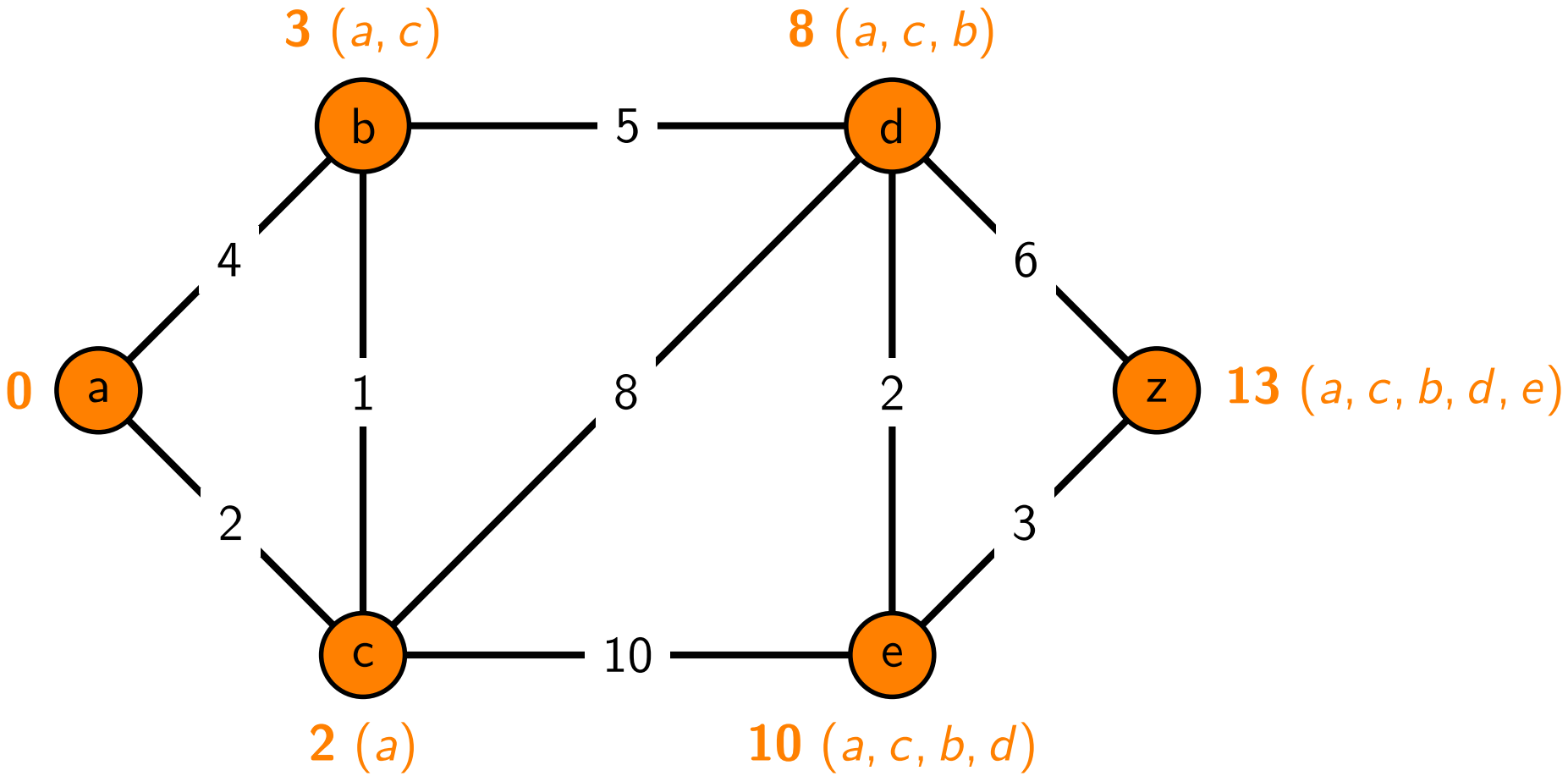
Dijkstra's Algorithm - 5th Iteration

$S_5 = \{a, b, c, d, e\}$.



Dijkstra's Algorithm - 6th Iteration

$S_5 = \{a, b, c, d, e, z\}$.



Dijkstra's Algorithm

To establish that Dijkstra's algorithm actually gives shortest path from a to z, we will prove the following result:

3.2.15. Theorem. Let G be an undirected, weighted, connected graph and a a vertex in G. At the kth iteration of Dijkstra's algorithm,

(i) The label $L_k(v)$ of every vertex $v \in S_k$ is the length of a shortest path from a to v;

(ii) The label $L_k(w)$ of every vertex $w \notin S_k$ is the length of a shortest path from a to w that contains only (besides w itself) vertices in S_k.

Proof.

We prove these statements by induction in $k \in \mathbb{N}$. For $k = 0$, $S_0 = \emptyset$, so (i) is vacuously true. Furthermore, there is no path from a to a vertex other than a using vertices in S_0, so, by definition, its length is ∞. It follows that (i) and (ii) are true for $k = 0$.

Dijkstra's Algorithm

Proof (continued).

Assume that (i) and (ii) are true for $k \in \mathbb{N}$. The vertices in S_k are hence labeled with the length of the shortest path from a. Let $u \in S_{k+1}$ but $u \notin S_k$, so u is the vertex added in the kth iteration. Thus, u has a smallest label of all vertices not in S_k.

It is clear (from (ii)) that u is labeled with the length of a shortest path containing only vertices in S_k. We claim that the label of u is actually the length of a shortest path without regard to where the path's vertices lie. We show this by contradiction: Suppose that there exists a path of length less than $L_k(u)$ from a to u containing a vertex not in S_k. Let v be the first vertex along this path which is not in S_k. Then $L_k(v) < L_k(u)$ so v should have been added to S_k instead of u, giving a contradiction. Hence the label of u is the shortest path from a to u. This implies (i) with k replaced by $k + 1$.

Dijkstra's Algorithm

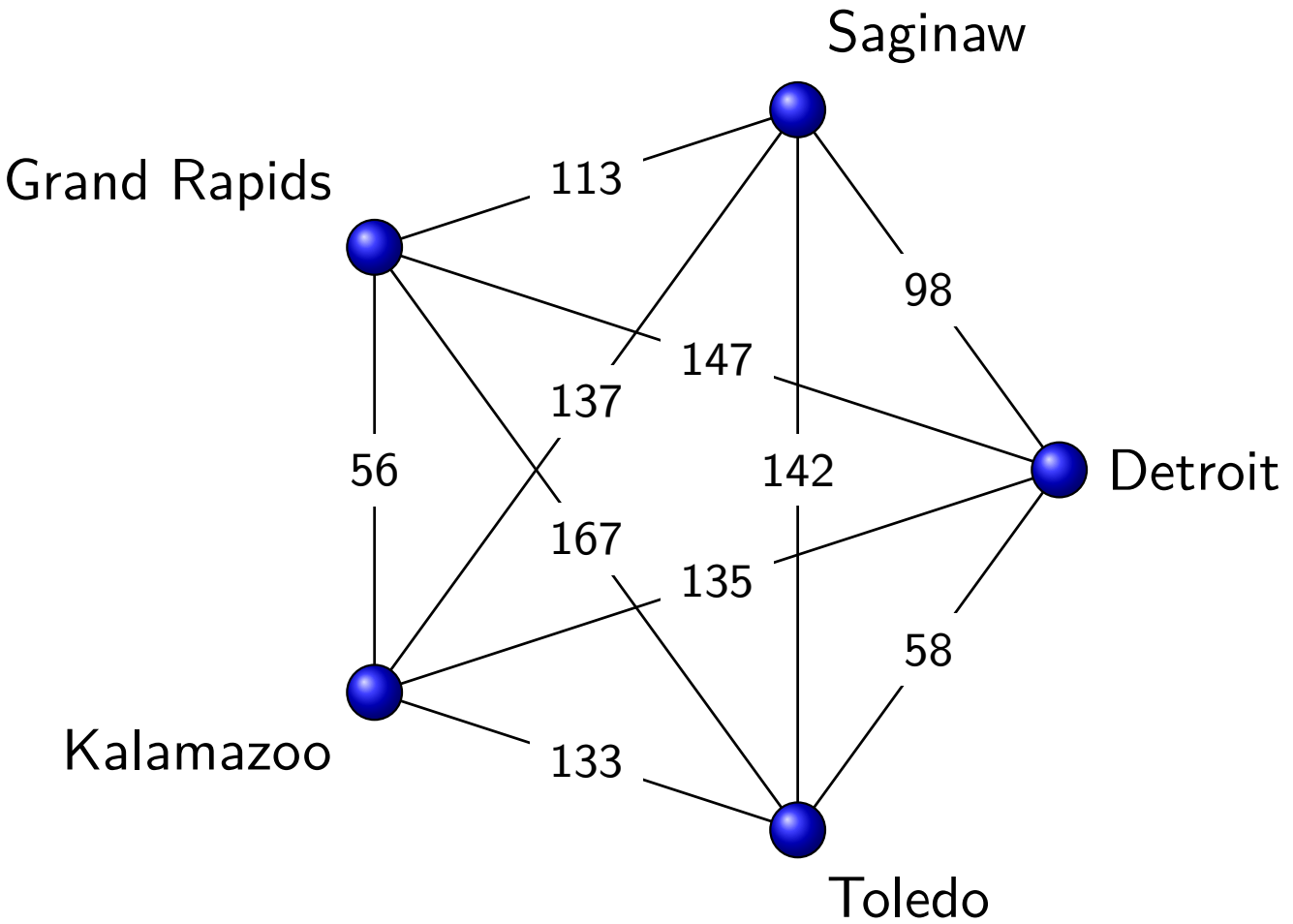
Proof (continued).
Let $v \notin S_{k+1}$. A shortest path from a to v containing only elements of S_{k+1} either contains u or it does not. If it does not contain u, then $L_{k+1}(v) = L_k(v)$ is the length of the shortest path from a to v since we assumed that (ii) is true for k. If it does contain u, the path is made up of a shortest path from a to u followed by the edge from u to v. (Why?) But then the length of this path is just given by the right-hand side of (3.2.2), which is equal to the label $L_{k+1}(v)$. Thus, (ii) is true with k replaced by $k + 1$. □

3.2.16. Corollary. Dijktsra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.

3.2.17. Theorem. Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected simple undirected weighted graph consisting of n vertices.

The Traveling Salesman Problem

Consider the following problem: a traveling salesman needs to visit the cities Detroit, Toledo, Grand Rapids, Saginaw and Kalamazoo, shown in the graph below with the distances from each other:



The Traveling Salesman Problem

The traveling salesman problem asks: in which order should the cities be visited to minimize the total distance travelled, assuming that he starts and ends his tour in Detroit? In other words, we are looking for the shortest Hamilton circuit of the graph starting and ending in Detroit. In our example, the graph is complete, so we can visit the cities in any order. By checking each possible route individually, it turns out that the circuit

Detroit - Grand Rapids - Toledo - Saginaw - Kalamazoo - Detroit

is longest, taking 728 miles, while

Detroit - Toledo - Kalamazoo - Grand Rapids - Saginaw - Detroit

is shortest with a length of 458 miles.

The Traveling Salesman Problem

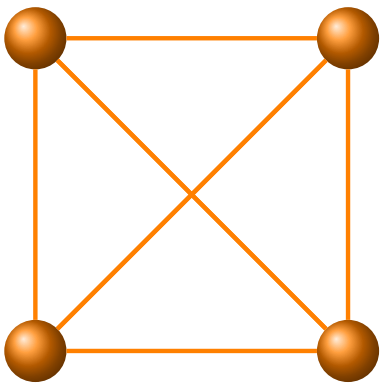
To solve the traveling salesman problem naively in a graph with n vertices, one must examine $(n-1)!/2 = O(n!)$ different Hamilton circuits – a prohibitive task for any realistic n. Due to the great practical importance of the problem, more efficient algorithms have been developed. These, however, are all exponential in time, and in fact, it has been shown that the problem belongs to the class of NP-complete problems.

There exist polynomial-time algorithms that do not find the shortest possible route (of weight $w_{\min}$) but rather a route of weight less than $c \cdot w_{\min}$, for example with $c = 3/2$. In practice, there are algorithms that solve the problem in graphs of 1000 vertices within 2% of the shortest path length in a few minutes of computer time.

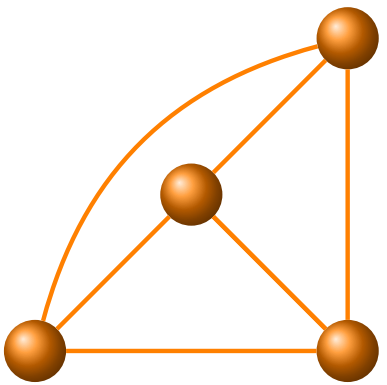
Planar Graphs

3.3.1. Definition. A graph is said to be ***planar*** if it can be drawn in the plane without any edges crossing. Such a drawing is called a ***planar representation*** of the graph.

3.3.2. Example. The complete graph K_4 is planar:

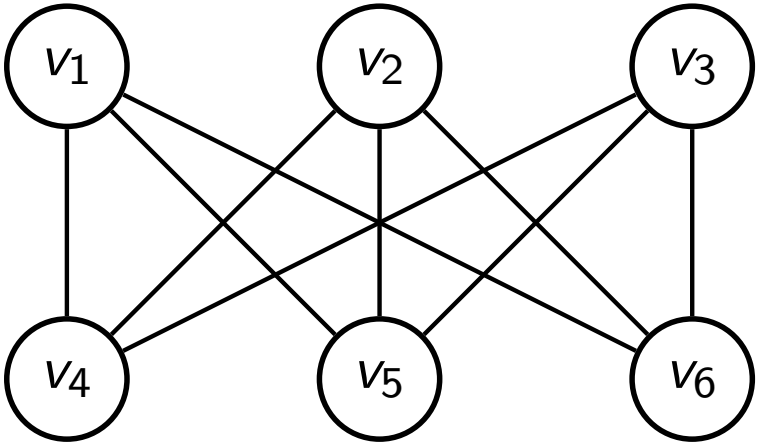


can be drawn as

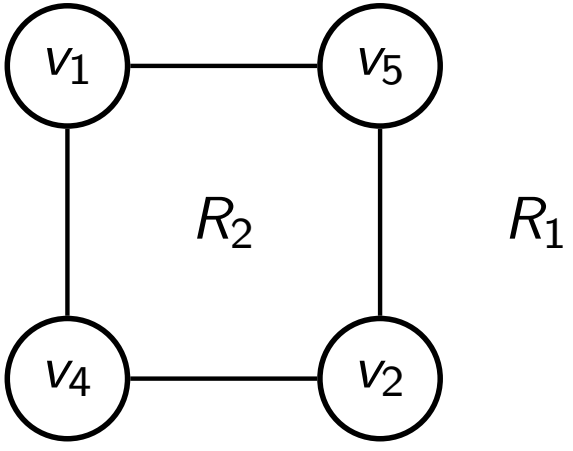


Planar Graphs

3.3.3. Example. The complete bipartite graph $K_{3,3}$ (see Example 3.1.23) is shown at right:

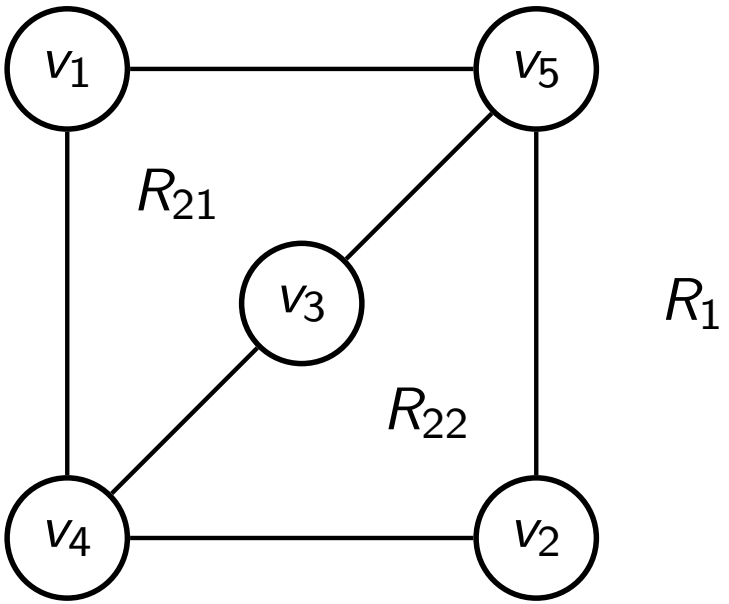


We will show that it is non-planar. In any planar representation, the vertices v_1 and v_2 are connected to both v_4 and v_5, splitting the plane into two regions, R_1 and R_2:



Planar Graphs

Now suppose the vertex v_3 is in the region R_2. The region R_2 is then divided into two subregions, R_{21} and R_{22}, as shown below:



Now there is no way to place the final vertex v_6 without two edges crossing: If $v_6 \in R_1$, the edge $\{v_3, v_6\}$ must cross one of the edges of the path $(v_1, v_5, v_2, v_4, v_1)$; if $v_6 \in R_{22}$, the edge $\{v_1, v_6\}$ must cross another edge; if $v_6 \in R_{21}$, the edge $\{v_2, v_6\}$ must cross another edge. A similar construction applies if $v_3 \in R_1$.

Euler's Formula

A planar representation of a finite graph in the plane splits the plane into ***regions***: polygons whose edges are the edges of the graph, as well as one unbounded region.

3.3.4. Euler's Formula. Let G be a finite connected planar simple graph with e edges and v vertices. Let r be the number of regions in a planar representation of G. Then

$$r = e - v + 2.$$

Proof.

Let $G = (V, E)$ be given in a planar representation. We will construct a finite sequence $G_1, G_2, \ldots, G_e = G$ of subgraphs of G as follows: Let G_1 be the subgraph consisting of one edge of G and the two vertices incident to this edge. $G_{k+1} = (V_{k+1}, E_{k+1})$ is obtained from $G_k = (V_k, E_k)$ as follows: select any edge in $E \setminus E_k$ incident to a vertex in V_k and add it to E_k. If this edge is incident to a vertex not in V_k, add the vertex to V_k. This construction is possible because G is connected.

Euler's Formula

Let r_k, e_k and v_k denote the number of regions, edges and vertices, respectively, of G_k. We will prove by induction that

$$r_k = e_k - v_k + 2 \tag{3.3.1}$$

for all $k \in \mathbb{Z}_+$. For $k = 1$ we have one edge and two vertices and a single region, so the formula holds. Now assume that $r_k = e_k - v_k + 2$ and that the edge $\{a_{k+1}, b_{k+1}\}$ is added to G_k. We consider two cases:

(i) $a_{k+1}, b_{k+1} \in V_k$. Since the graph is planar, the vertices a_{k+1} and b_{k+1} must lie on the boundary of a common region R (otherwise the edge $\{a_{k+1}, b_{k+1}\}$ would intersect some other edge). The addition of the edge $\{a_{k+1}, b_{k+1}\}$ then splits R into two subregions. Hence,

$$r_{k+1} = r_k + 1, \qquad e_{k+1} = e_k + 1, \qquad v_{k+1} = v_k$$

and (3.3.1) remains true for k replaced with $k + 1$.

Euler's Formula

Proof (continued).

(ii) $a_{k+1} \in V_k, , b_{k+1} \notin V_k$. In this case, b_{k+1} must lie in a region that has a_{k+1} on its boundary. Hence, the edge $\{a_{k+1}, b_{k+1}\}$ does not produce a new boundary to a region and

$$r_{k+1} = r_k, \qquad e_{k+1} = e_k + 1, \qquad v_{k+1} = v_k + 1.$$

Again, (3.3.1) remains true for k replaced with $k+1$ and the proof is complete. □

3.3.5. Corollary. If G is a connected planar simple graph with e edges and $v \geq 3$ vertices, then $e \leq 3v - 6$.

The proof of Corollary 3.3.5 uses the concept of the ***degree*** of a region: this is the number of edges that lie on the boundary of the region, where an edge that occurs twice on the boundary contributes two to the degree.

Euler's Formula

Proof.

Let G be a connected simple planar graph with at least three vertices. Then G divides the plane into $r \in \mathbb{Z}_+$ regions, where the degree of each region is at least three. The sum of the degrees of all regions is twice the number of edges. It follows that

$$2e = \sum_{R_i} \deg(R_i) \geq 3r.$$

Using Euler's formula,

$$e - v + 2 = r \leq \frac{2e}{3}$$

from which $e \leq 3v - 6$ follows. $\square$

Euler's Formula

3.3.6. Corollary. If G is a connected planar simple graph then G has a vertex of degree not exceeding five.

Proof.
If G has one or two vertices the result is obviously true. If G has three or more vertices, we can apply Corollary 3.3.5 to give

$$2e \leq 6v - 12.$$

If the degree of every vertex were at least six, by the Handshaking Theorem we would have

$$2e = \sum_{v_i} \deg(v_i) \geq 6v,$$

giving a contradiction. □

Euler's Formula

3.3.7. Corollary. Let G be a connected planar simple graph with e edges and $v \geq 3$ vertices. If G has no circuits of length three, then $e \leq 2v - 4$.

The proof is left to the reader.

3.3.8. Examples.

(i) The complete graph K_5 is non-planar, because it has $v = 5$ vertices and $e = 10$ edges, so $3v - 6 = 9 \not\geq 10 = e$.

(ii) From Example 3.3.3 we know that $K_{3,3}$ is non-planar. However, it has $v = 6$ vertices and $e = 9$ edges and hence satisfies $e \leq 3v - 6$, so Corollary 3.3.5 can not be used to show that it is non-planar.

(iii) The graph $K_{3,3}$ is bipartite, so it has no circuits of length three (why?). By Corollary 3.3.7 it should satisfy the inequality $e \leq 2v - 4$ if it were planar. However, $e = 9 \not\leq 8 = 2v - 4$, so it is non-planar.

Elementary Subdivisions and Homeomorphic Graphs

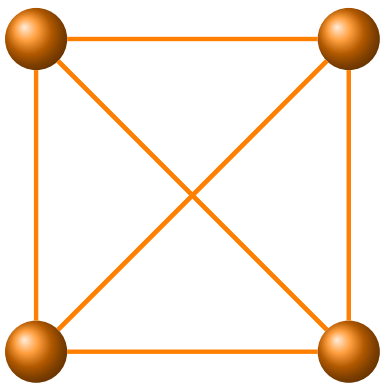
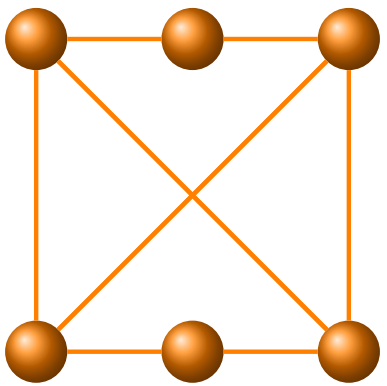
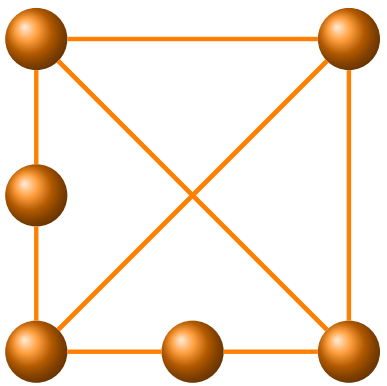
3.3.9. Definition. Let $G = (V, E)$ be a graph, $v_1, v_2 \in V$ and $\{v_1, v_2\} \in E$. Then the graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ given by

$$\widetilde{V} = V \cup \{w\}, \qquad \widetilde{E} = (E \setminus \{v_1, v_2\}) \cup \big\{\{v_1, w\}, \{w, v_2\}\big\}$$

is said to be obtained from G through an ***elementary subdivision***. We say that G is obtained from itself through an ***empty elementary subdivision***.

Two graphs G_1, G_2 are said to be ***homeomorphic*** if there exists a graph G such that G_1 and G_2 can each be obtained from G through successive (possibly empty) elementary subdivisions.

3.3.10. Example. The following graphs are all homeomorphic:

Elementary Subdivisions and Homeomorphic Graphs

Two graphs that are homeomorphic are either both planar or non-planar:

3.3.11. Lemma. Let G_1 and G_2 be homeomorphic graphs. Then G_1 is planar if and only if G_2 is planar.

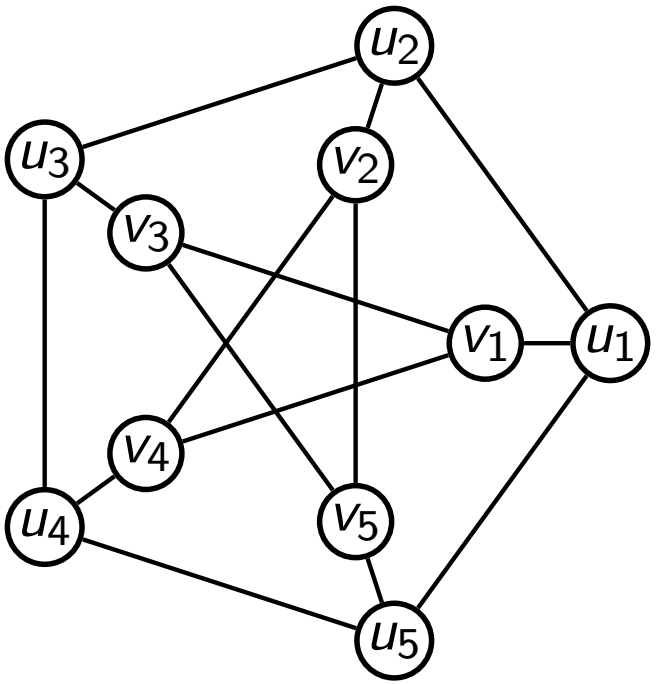
Proof.
It is sufficient to show that if G_1 is planar, then G_2 is also planar. Let G be the graph from which G_1 and G_2 were obtained through elementary subdivisions. Suppose that G_1 is planar. Then G must also be planar (it simply contains fewer vertices through which the edges pass; the geometry of the edges is the same). For the same reasons the elementary subdivisions of G that yield G_2 do not cause a crossing of edges, so G_2 is also planar. □

It is clear that a graph G is non-planar if a subgraph of G is non-planar. In particular, if a subgraph of G is homeomorphic to a non-planar graph, then G is non-planar. There is a (perhaps surprising) result from Kuratowski that actually gives a specific criterion along these lines:

Kuratowski's Theorem

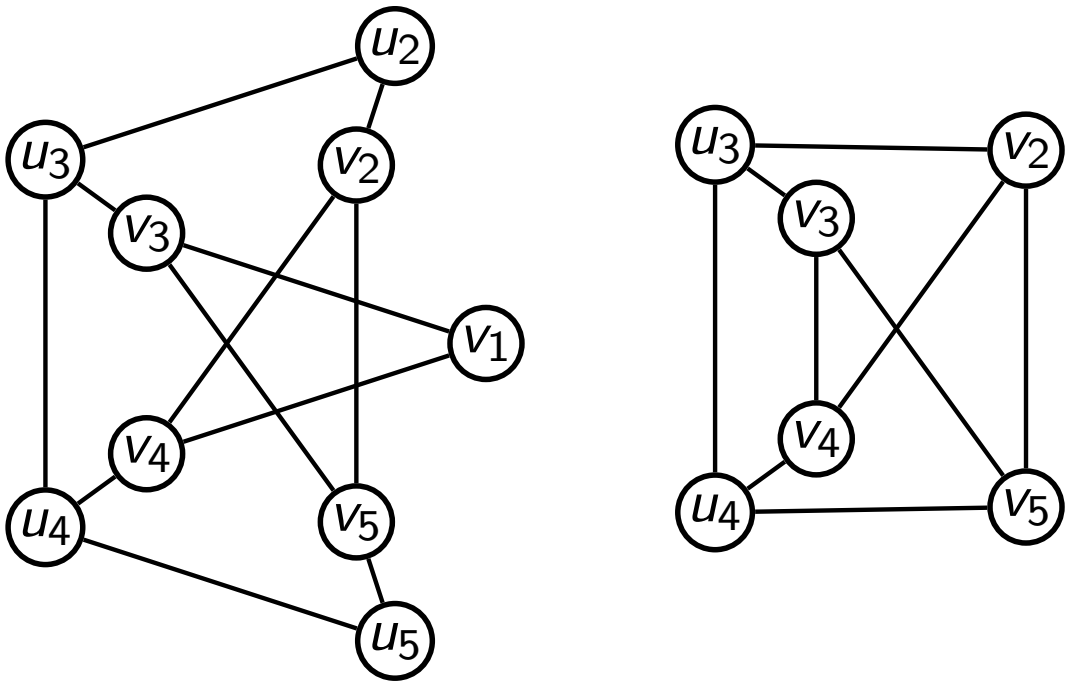
3.3.12. Kuratowski's Theorem. A graph is non-planar if and only if it contains a subgraph that is homeomorphic to $K_{3,3}$ or K_5.

3.3.13. Example. Consider the ***Petersen graph***, shown below. It has no circuits of length 3, $v = 10$ vertices and $e = 15$ edges. It satisfies $e \leq 2v - 4$, so Corollary 3.3.7 does not preclude it from being planar. We will establish that it is actually non-planar.



Kuratowski's Theorem

Consider the subgraph obtained by removing u_1 and the three edges incident to it (below left):



Removing the vertices u_2, u_5 and v_1 yields a homeomorphic graph. It is easy to see that this graph is actually (isomorphic to) the complete bipartite graph $K_{3,3}$ (with bipartition $(\{u_3, v_4, v_5\}, \{v_2, v_3, u_4\})$), so the Petersen graph is non-planar.

Coloring of Graphs

Many applications revolve around assigning colors to vertices of a graph.

3.3.14. Definition. A ***coloring*** of a graph is an assignment of colors to vertices so that no two adjacent vertices have the same color.

If a graph admits a coloring using k colors, we say that is is ***k-colorable***.

The ***chromatic number*** $\chi(G)$ of a graph G is the least number of colors needed for a coloring of the graph.

3.3.15. Examples.

1. $\chi(K_n) = n$,
2. $\chi(C_n) = 2$ if $n \geq 2$ and n is even,
3. $\chi(C_n) = 3$ if $n \geq 3$ and n is odd,
4. $\chi(W_n) = \chi(C_n) + 1$,
5. $\chi(K_{m,n}) = 2$.

2-Coloring of Graphs

3.3.16. Theorem. A graph is 2-colorable if and only if it has no circuits of odd length.

Proof.

($\Rightarrow$) Suppose a graph is two-colorable. Then the vertices along every path must have alternating colors: If the first color is indicated by "color -1" and the second color by "color 1", then the kth vertex in a path will have color $(-1)^k$. In a path of length n, the final vertex will have the color $(-1)^{n+1}$. In a circuit of length n, the final vertex coincides with the first vertex, so $-1 = (-1)^{n+1}$, which implies that n is even.

($\Leftarrow$) Suppose a graph has no circuits of odd length. We choose an arbitrary vertex a and color it white. All adjacent vertices are colored black. These adjacent vertices are not joined by an edge, since that would lead to a triangle (a circuit of length 3). Now all vertices adjacent to the black vertices are colored white. We need to show that there is no edge joining two white vertices.

2-Coloring of Graphs

Proof (continued).

($\Leftarrow$) This is the case, because the new white vertices are not adjacent to the initial vertex a. There is also no edge joining the new white vertices, since otherwise we would obtain a simple circuit of length 3 or 5.

We claim that, continuing in this way, we obtain a 2-coloring of the graph. We see this as follows:

Suppose the above procedure leads to two adjacent black vertices. Then each of these is joined to the initial vertex a or to some other vertex by a path of the same length modulo 2, i.e., either even or odd length. Since the two vertices are adjacent, we obtain a simple circuit of length $n_1 + n_2 + 1$ where n_1 and n_2 are the lengths of the two paths. Since n_1 and n_2 are either both even or both odd, the circuit has odd length, which is a contradiction. $\square$

k-Coloring of Graphs

The following theorem gives an answer to the question of how many colors are at most needed to give a coloring of a graph:

3.3.17. Brooks's Theorem. If every vertex of a finite graph has degree at most d, then the graph can be colored using $d + 1$ colors.

Proof.

We will prove the theorem is true for graphs with n vertices of degree at most d by induction in n.

- A graph with fewer than d vertices can be trivially colored with at most $d + 1$ colors.
- Suppose any graph with at most $n - 1$ vertices of degree at most d can be colored using $d + 1$ colors. Let G be a graph of n vertices of degree at most d. Remove an arbitrary vertex v and all incident edges. The resulting graph has n vertices of degree less than or equal to d, since the removal of the edges does not increase the degree of any remaining vertex.

k-Coloring of Graphs

Proof (continued).

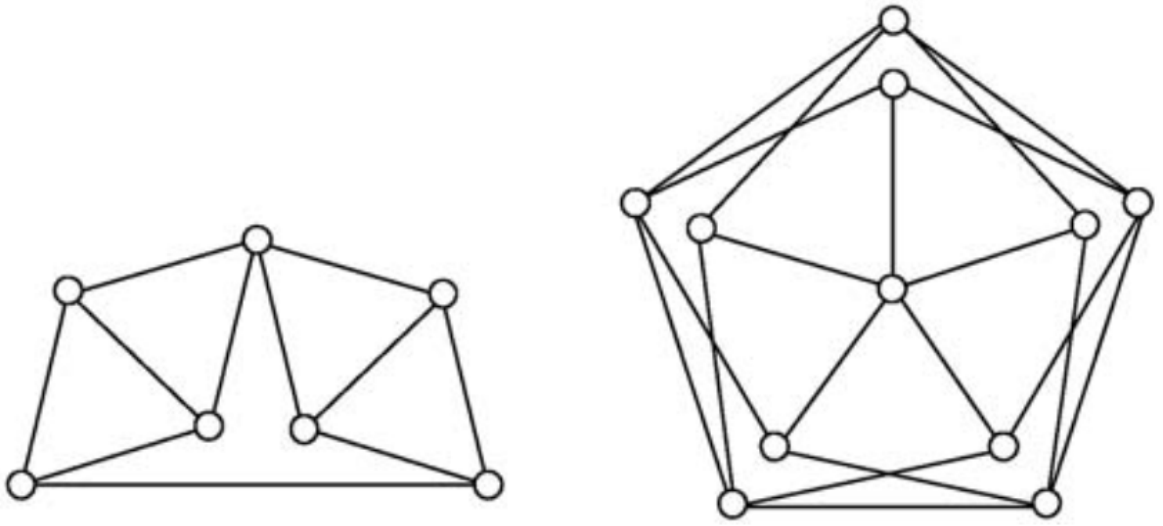
We will prove the theorem is true for graphs with n vertices of degree at most d by induction in n.

- The reduced graph can be colored using $d+1$ colors by our induction hypothesis. Since the degree of the removed vertex was at most d, it was originally joined to d vertcies colored using at most dd of the $d+1$ colors. Hence, at least one of the $d+1$ colors can be used to color v. We we have obtained a coloring of our original graph using $d+1$ colors.

By the induction theorem, any graph of n vertices of degree at most d can be colored with at most $d+1$ colors. □

k-Coloring of Graphs

3.3.18. Example. Both of the graphs below are not 3-colorable. The graph on the right does not even contain a triangle!



Maps and Dual Graphs

Consider a map of regions, as shown below:

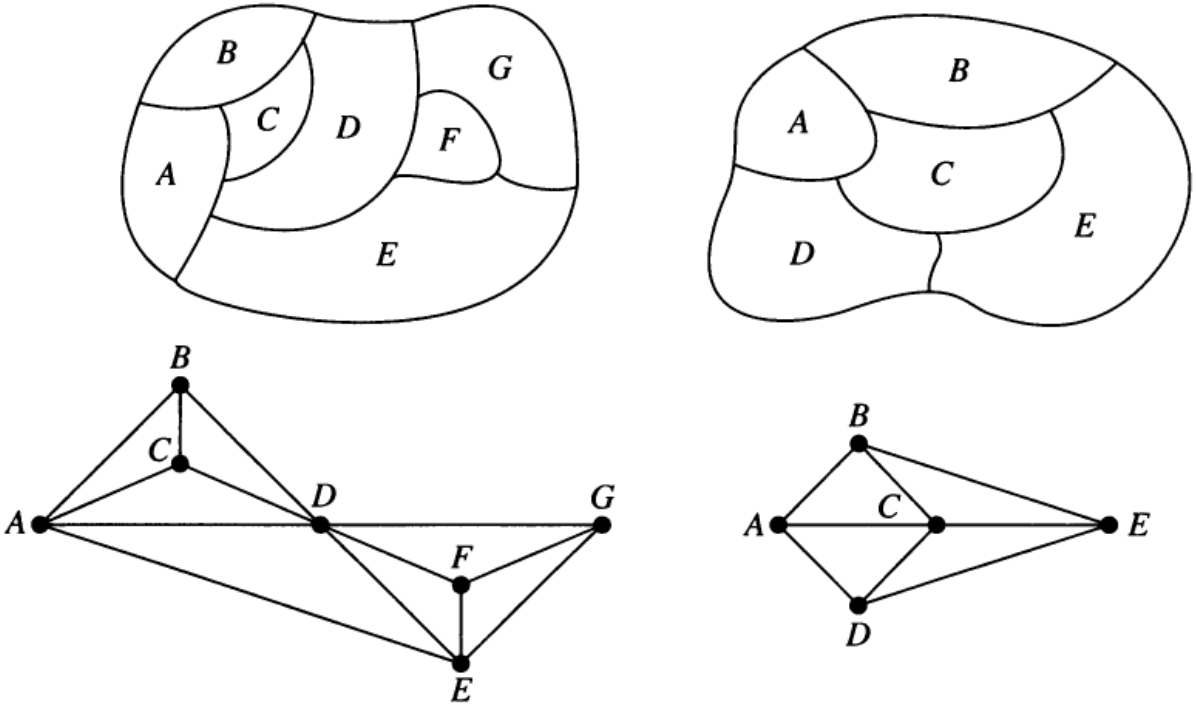


Image source: Rosen, K.H., ***Discrete Mathematics***, 6th Ed., McGraw-Hill International Edition 2007. Used under fair use exemption.

Maps and Dual Graphs

The ***dual graph*** is found as follows: choose an arbitrary point in each region (the dual vertex) and connect the points by drawing a single line joining them through each distinct border. If a border has two or more separate components (e.g., the border between Russia and China, which is interrupted by Mongolia), a single line is sufficient. The system of points (vertices) and lines (edges) then gives the dual graph, which is always planar (why?).

A famous, classic question now is: How many colors are at most necessary to color a map, or equivalently, it's dual graph?

The Four-Color Theorem

The following theorem was first formulated as a conjecture in the 1850s and was only proven in 1976 with the aid of computers. The (at the time controversial) proof is the first example of a significant theorem that was proven in a way that would make it all but impossible for a human being to verify the result by hand.

3.3.19. Four Color Theorem. The chromatic number of a planar graph is no greater than four.

3.3.20. Example. GSM cell phone networks operate in different frequency ranges, as adjacent regions need to have different frequencies to prevent interference. GSM uses only four such ranges, which (by the Four Color Theorem) is sufficient.

PSLQ, BBP & More Computer Math

The issues raised by the proof of the Four Color Theorem have become more apparent in the last few years:

- "Proof algorithms" have been designed that can automate the proof of theorems in logic. By referring to truth tables and elementary logical rules, in principle it should be possible to automate the process of establishing he truth or falsehood of statements, provided that is possible in the first place.There have been several successes in this approach, which remains an active area of research.
- Rather than just proving theorems, computers have also been used to ***discover*** theorems. The best-known example is the PSLQ algorithm, that finds relations linking constants to algebraic expressions automatically. In fact, one surprising formula to be discovered in this way is the BBP (Borwein-Bailey-Plouffe) formula in 1996,

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right).$$

PSLQ, BBP & More Computer Math

The BBP formula for π is remarkable because despite its simple form, it has escaped detection by humans until it was "discovered" by the PSLQ algorithm. Once discovered, the proof is actually not difficult (see assignments).

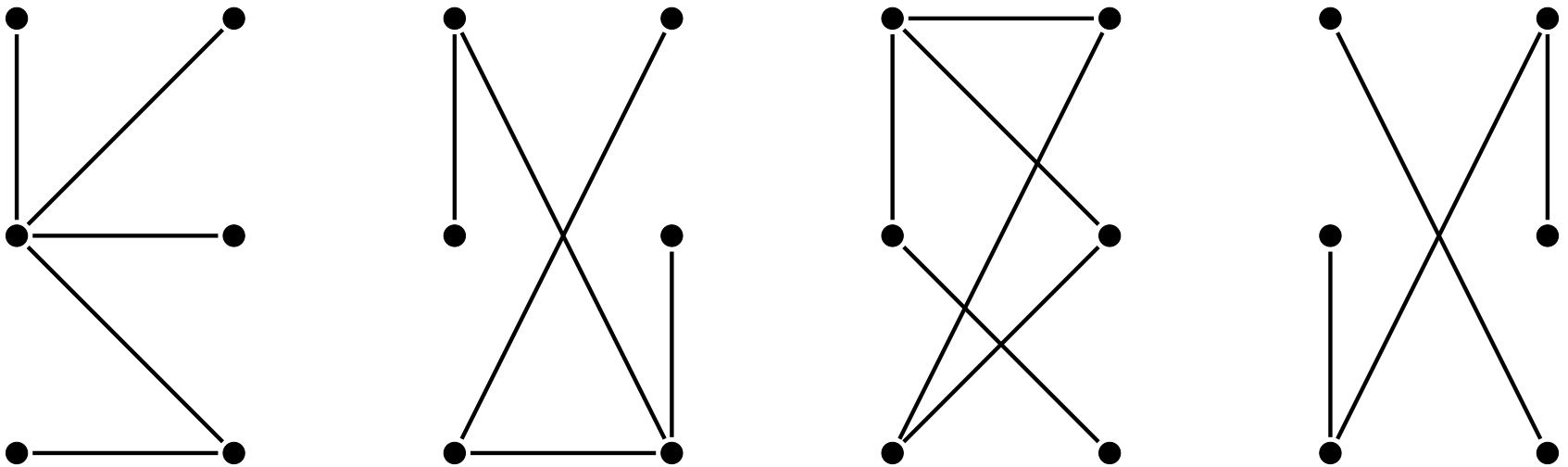
Furthermore, the formula is also useful, since it can be used to calculate the nth decimal of π in hexadecimal expansion without first calculating the preceding digits. Such formulas are very rare and have recently been designated "spigot" formulas.

It appears that computers have become an integral part of mathematics, far beyond performing simple calculations or visualizations.

Trees

3.4.1. Definition. A ***tree*** is a connected undirected graph with no simple circuits. A not necessarily connected graph with no simple circuits is called a ***forest***.

3.4.2. Example. Of the graphs below, G_1 and G_2 are trees, while G_4 is a forest.



G_1 G_2 G_3 G_4

Trees and Simple Paths

Trees can also be characterized by paths. We first give a theorem on the existence of simple paths in general undirected graphs.

3.4.3. Theorem. There is a simple path between every pair of distinct vertices of a connected undirected graph.

Proof.
Let $G = (V, E)$ be a connected undirected graph and $u \neq v$ two vertices. Since G is connected, there exists at least one path joining u and v. Let $(u, x_1, \dots, x_{n-1}, v)$ be the tuple of vertices associated to such a path having least length. We then claim that this path is simple. If it is not simple, the $x_i = x_j$ for some $i < j$. However, then the tuple of vertices $(x_{i+1}, \dots, x_j)$ can be eliminated from the path to give a shorter path, yielding a contradiction. □

Trees and Simple Paths

The following theorem gives an equivalent characterization of trees which is sometimes used as a definition.

3.4.4. Theorem. An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Proof.

($\Rightarrow$) Suppose that T is a tree. Then T is a connected graph with no simple circuits. Let x, y be distinct vertices. Then, since T is connected, there exists a simple path joining x and y by Theorem 3.4.3. This path must be unique, for if there were a second such path the two paths could be joined to yield a circuit. This circuit could further be reduced to a simple circuit (see assignments), yielding a contradiction.

Trees and Simple Paths

Proof (continued).

($\Leftarrow$) Suppose that T is a graph and that there exists a unique simple path between any two vertices of T. Then T is clearly connected. Furthermore, T can not have a simple circuit: if there were such a circuit starting and ending at a vertex x and passing through a vertex y, then there would be two distinct simple paths joining x and y. But this contradicts the assumption that there exists a unique simple path between x and y. $\square$

Rooted Trees

3.4.5. Definition. A ***rooted tree*** is a tree in which one vertex has been designated as the root and every edge is imbued with a direction in such a way that the root is the initial vertex of any path joining the root to any other vertex.

Definition 3.4.5 suffices to imbue each edge of a tree with a unique direction (why?).

Rooted Trees

We next introduce some terminology:

3.4.6. Definition. Let T be a rooted tree with root r.

- If $v \neq r$ is a vertex in T, the ***parent*** of v is the unique vertex u such that there is a directed edge (u, v) in T.
- If u is the parent of v, the v is a ***child*** of u.
- If v_1, v_2 have the same parent u, they are called ***siblings***.
- If $v \neq r$, the ***ancestors*** of v are vertices in the unique directed path joining r to v, excluding v and r.
- The ***descendants*** of a vertex u are all those vertices which have u as an ancestor.
- A vertex with no children is called a ***leaf***, otherwise it is called an ***internal vertex***.
- If a is a vertex in T, the ***rooted subtree*** with root a is the tree whose vertices comprise a and all its descendants and whose edges comprise all the edges joining a to these descendants.

m-ary Trees

3.4.7. Definition. A rooted tree is called an ***m-ary tree*** if every internal vertex has no more than m children. The tree is called a ***full m-ary tree*** if every internal vertex has exactly m children. A (full) m-ary tree with $m = 2$ is called a (full) ***binary tree***.

3.4.8. Definition. An ***ordered rooted tree*** is a tree where the children of each vertex are ordered.

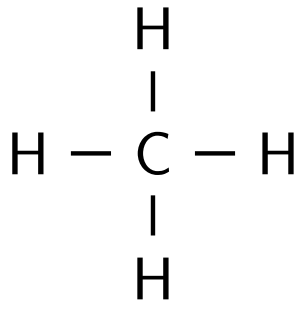
When drawing a rooted tree in the conventional way, we consider the children to be ordered from left to right. Unless stated otherwise, all depicted trees are considered to be ordered in this way.

For (ordered) binary trees we define the ***left child*** and the ***right child*** of a root in the obvious way. If u is an internal vertex, the (rooted) subtree with the left/right child of u as its root is called the ***left/right subtree of u***.

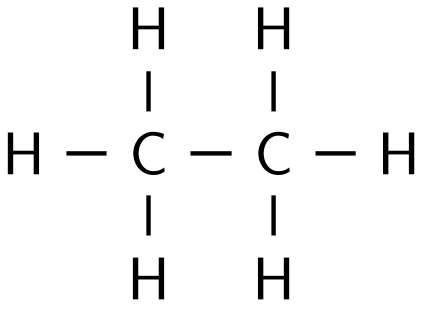
Application to Saturated Hydrocarbons

A ***saturated hydrocarbon***, or ***alkane***, has the chemical formula C_nH_{2n+2}. Examples include

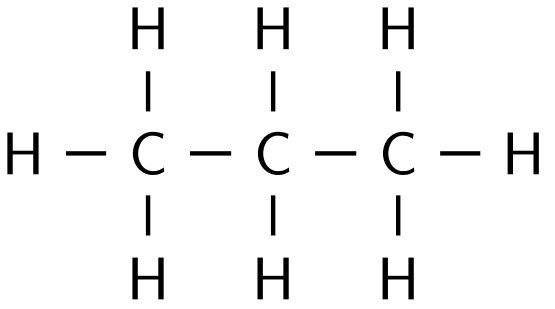
Methane (CH_4)



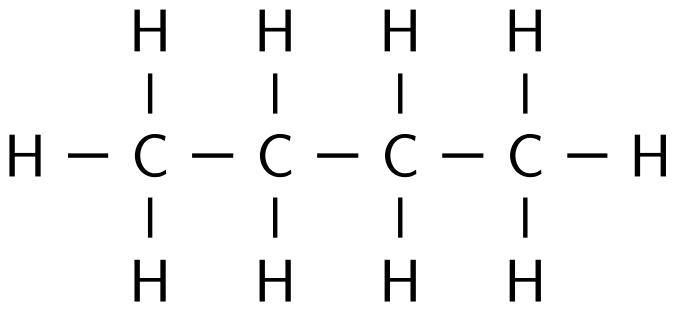
Ethane (C_2H_6)



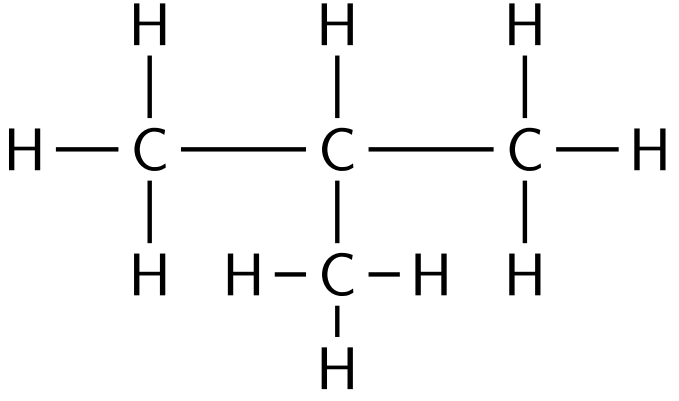
Propane (C_3H_8)



n-Butane (C_4H_{10})



iso-Butane (Methylpropane; C_4H_{10})



Application to Saturated Hydrocarbons

We can define saturated hydrocarbons inductively as follows:

3.4.9. Definition.

(i) The rooted tree consisting of one root and four children is an alkane.
(ii) If T is a rooted tree, then the tree obtained by adding three children to a leaf is an alkane.

In this definition, the internal vertices will represent carbon atoms, while the leaves represent hydrogen atoms.

Definition 3.4.9 will yield alkanes that are different regarded as rooted trees, but whose underlying undirected graphs are isomorphic. These are considered to represent the same molecule.

Alkanes with the same number of internal vertices and the same number of leaves that are not isomorphic are called ***stereoisomers***. Finding the number of stereoisomers of a given compound is a non-trivial problem involving graphs and counting. More information on this problem can be found at http://www.cs.uwaterloo.ca/journals/JIS/cayley.html

Counting the Vertices and Edges of Trees

3.4.10. Theorem. A tree with n vertices has $n-1$ edges.

Proof.
The proof follows easily by induction: for $n = 1$ the tree consists only of a single vertex and no edges, so the statement is true.

Suppose the statement is true for any tree with n vertices and let T be a tree with $n+1$ vertices. Then T has a leaf, which can be removed along with the vertex joining it to its parent. The resulting tree has n vertices, and, by the induction hypothesis, $n-1$ edges. Restoring the eliminated leaf and its edge, we see that T has n edges. □

Counting the Vertices and Edges of Trees

3.4.11. Theorem. A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Proof.

There are two types of vertices in a full m-ary tree: the root and vertices that are children of other vertices. Since there are i internal vertices, there must be $m \cdot i$ children. Adding the root, we obtain $n = mi + 1$. □

3.4.12. Theorem. A full m-ary tree with

(i) n vertices has $i = (n-1)/m$ internal vertices and $l = [(m-1)n + 1]/m$ leaves,

(ii) i internal vertices has $n = mi + 1$ vertices and $l = (m-1)i + 1$ leaves,

(iii) l leaves has $n = (ml - 1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices.

Proof.

The theorem follows from the two equations $n = mi + 1$ and $n = i + l$. □

Counting the Vertices and Edges of Trees

3.4.13. Example. Suppose that someone starts a chain letter. Each person who receives the letter is asked to send it on to four other people. Some people do this, but others do not send any letters. How many people have seen the letter, including the first person, if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out? How many people sent out the letter?

Balanced m-ary Trees

3.4.14. Definition. Let T be a rooted tree with root r.

- The ***level*** of a vertex $v \neq r$ is the length of the path joining r to v. The level of r is defined to be zero.
- The ***height*** of T is the maximum of all the levels of the vertices of T.
- A rooted m-ary tree of height h is said to be ***balanced*** if all leaves have levels h or $h-1$.

3.4.15. Theorem. An m-ary tree of height h has at most m^h leaves.

Proof.

We prove the statement by strong induction in h. The statement is true for $h = 0$, because the tree then consists of a single vertex, the root. This is a leaf, so the number of leaves is $m^0 = 1$.

Balanced m-ary Trees

Proof (continued).

Assume T is a tree of height h. Removing the root and all edges originating at the root, one obtains a forest of at most m tress of height at most $h-1$. Each tree has at most m^{h-1} leaves by the induction hypothesis. Hence, the total number of leaves is at most $m \cdot m^{h-1} = m^h$. □

3.4.16. Corollary.

(i) If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$.

(ii) If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$.

Proof.

Statement (i) follows directly from $l \leq m^h \Leftrightarrow h \geq \log_m l$. Statement (ii) will be proven in the assignments. □

Binary Search Trees

Suppose we have a totally ordered set of objects that we want to enter into a database for easy lookup. One technique involves the use of binary trees as follows:

1. The first object becomes the root of the search tree.
2. The second object becomes a left child if it precedes the root, or a right child if it succeeds it.
3. The third and further objects are sorted along the tree, moving left if they precede a vertex or right if they succeed it.

3.4.17. Example. We enter the words mathematics, physics, geography, zoology, meteorology, geology, psychology, and chemistry (in this order) into a search tree, using the standard lexicographic ordering of words.

Locating and Adding Items to a Binary Search Tree

3.4.18. Algorithm. ***(Locating or adding items to a binary search tree)***

Input: T a binary search tree and x an item to be found or added
Output: v the position of x in the binary tree

```
1  v ← root of T;
2  if v = NULL then v ← x; return v;
3  while v.label ≠ x.label do
4  |   if x.label < v.label then
5  |   |   if v.left = NULL then v.left ← x;
6  |   |   v ← v.left;
7  |   else
8  |   |   if v.right = NULL then  v.right ← x;
9  |   |   v ← v.right;
10 |   end if
11 end while
12 return v
```

Locating and Adding Items to a Binary Search Tree

We now analyze the computational complexity of Algorithm 3.4.18. First, given a binary tree T with n labeled vertices, we add unlabeled vertices to it so that every labeled vertex has two children. This gives a full binary tree, which we denote by U. The labeled vertices are the interior vertices of U. The advantage of this procedure is that when locating or adding an item, we never increase the number of vertices in U.

The most comparisons needed to add an item is the length of the longest path in U from its root to a leaf, i.e., the height h of U. The vertices of T are the interior vertices of U, so U has n interior vertices. By Theorem 3.4.12, U then has $n+1$ leaves. Corollary 3.4.16 gives

$$h \geq \lceil \log_2(n+1) \rceil.$$

If T is balanced we actually have equality, so the algorithm requires at most $\lceil \log(n+1) \rceil = O(\log n)$ comparisons. As items are added to a search tree, it may become unbalanced. There are algorithms (discussed in courses on data structures) that can "rebalance" a tree in this case.

Decision Trees

A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a ***decision tree***. The possible solutions of the problem correspond to the paths to the leaves of this rooted tree. This is illustrated through the following example:

3.4.19. Example. Suppose that among eight nominally identical coins one is counterfeit and lighter than the others. This coin may be found by weighing any number of the eight coins on a balance scale. We wish do determine the minimum number of weighings needed to find the counterfeit coin.

Finding a Counterfeit Coin

Every weighing corresponds to an internal vertex in a decision tree and every result is a child of this weighing and also corresponds to the next weighing. Each weighing has three possible outcomes:

- "left side is lighter"
- "right side is lighter"
- "no side is lighter"

so the decision tree is a 3-ary tree. The final result after the weighings ("coin no. n is counterfeit") corresponds to a leaf in the tree. There are 8 possible leaves, so, by Corollary 3.4.16 the height of the tree is at least $\lceil \log_3 8 \rceil = 2$. Hence, at least 2 weighings are necessary.

Finding a Counterfeit Coin

A naive approach easily shows that the coin can be found in three weighings. However, it is also possible to do so in just two weighings, as illustrated below:

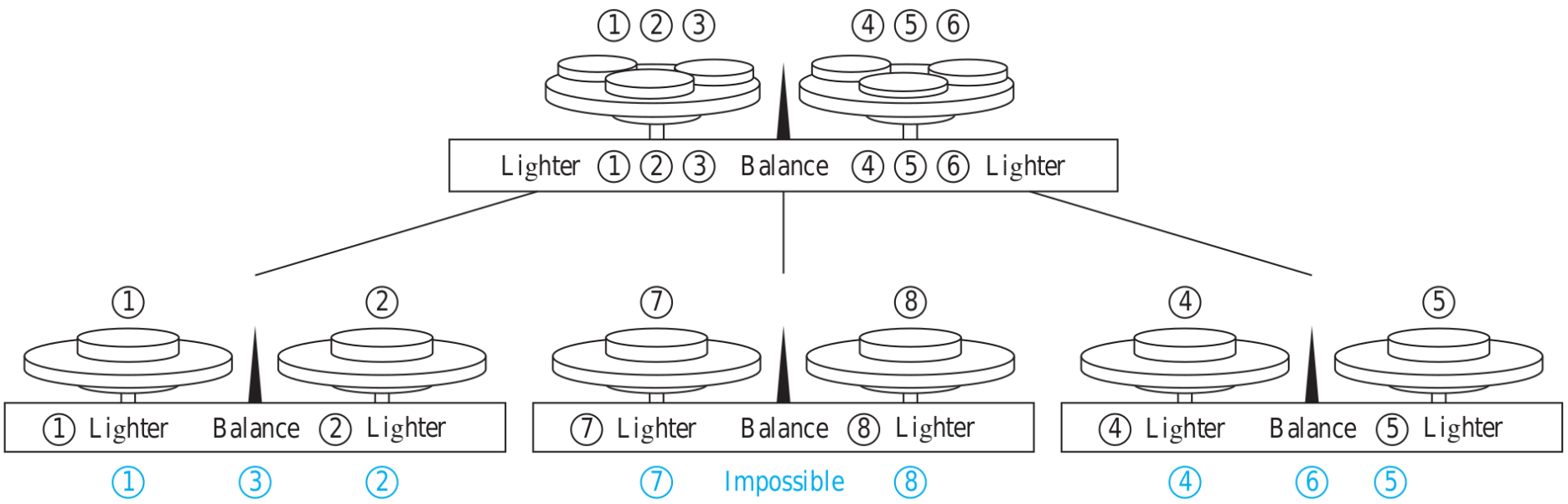


Image source: Rosen, K.H., ***Discrete Mathematics***, 6th Ed., McGraw-Hill International Edition 2007. Used under fair use exemption.

Complexity of Binary Sorting Algorithms

Consider a sorting algorithm based on comparing two configurations of a given list (usually by comparing two possible positions of a single list item). Such an algorithm is called a ***binary sorting algorithm***, because each comparison will designate one of two possible list configurations as being "more sorted". All of the sorting algorithms we have studied are of this type.

We can model any binary sort algorithm using a binary decision tree. For a list of n distinct elements there are $n!$ different arrangements, one of which will correspond to the sorted list. Hence, the decision tree has $n!$ leaves and, by Corollary 3.4.16 the height of the tree is at least $\lceil \log_2(n!) \rceil$ and at least that many comparisons are needed.

We now conduct a brief excursion into calculus to obtain an estimate for $\log_2(n!)$.

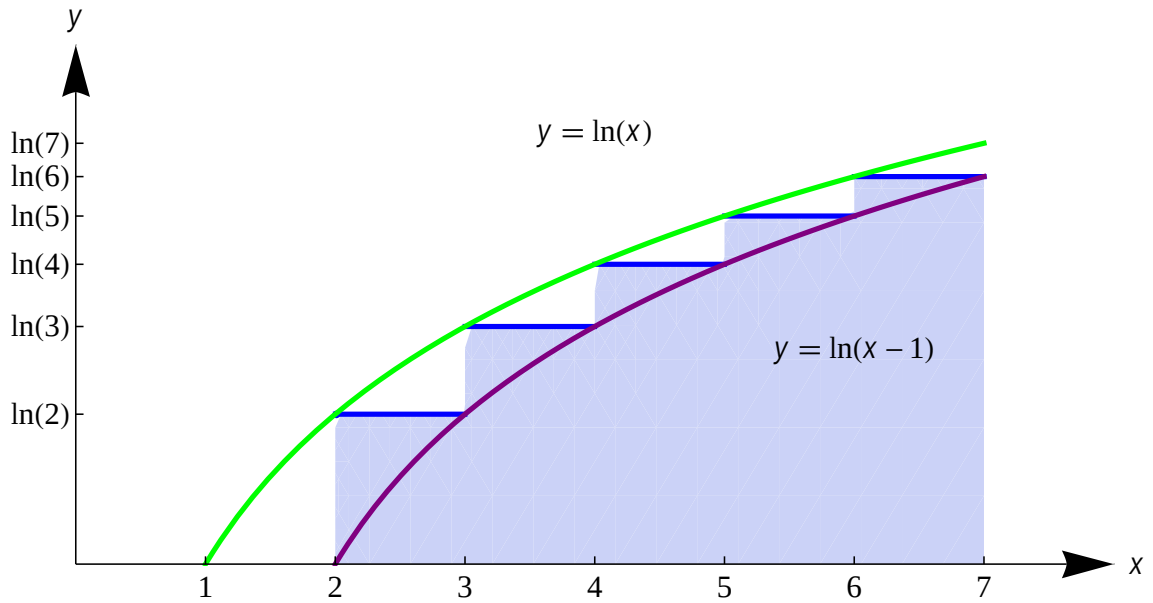
Elementary Estimate on the Factorial

We may obtain an elementary estimate as follows: Let

$$S_n = \ln(n!) = \sum_{k=1}^{n} \ln k.$$

Then it is easy to see that

$$\int_1^{n+1} \ln x \, dx > S_n > \int_2^{n+1} \ln(x-1) \, dx = \int_1^n \ln x \, dx. \tag{3.4.1}$$



Elementary Estimate on the Factorial

Since $\int \ln(x)\,dx = x\ln(x) - x$, we immediately obtain

$$n\ln(n) - n - e + 1 < \ln(n!) < (n+1)\ln(n+1) - n - e.$$

This implies that $\log n! = \Theta(n \log n)$. It follows that the worst-case time complexity of any binary sort is $\Omega(n \log n)$. Thus, it is impossible to find a binary search algorithm that has a lower time complexity than $O(n \log n)$. The merge sort (Algorithm 2.1.27) may be considered optimal in this sense.

Prefix Codes

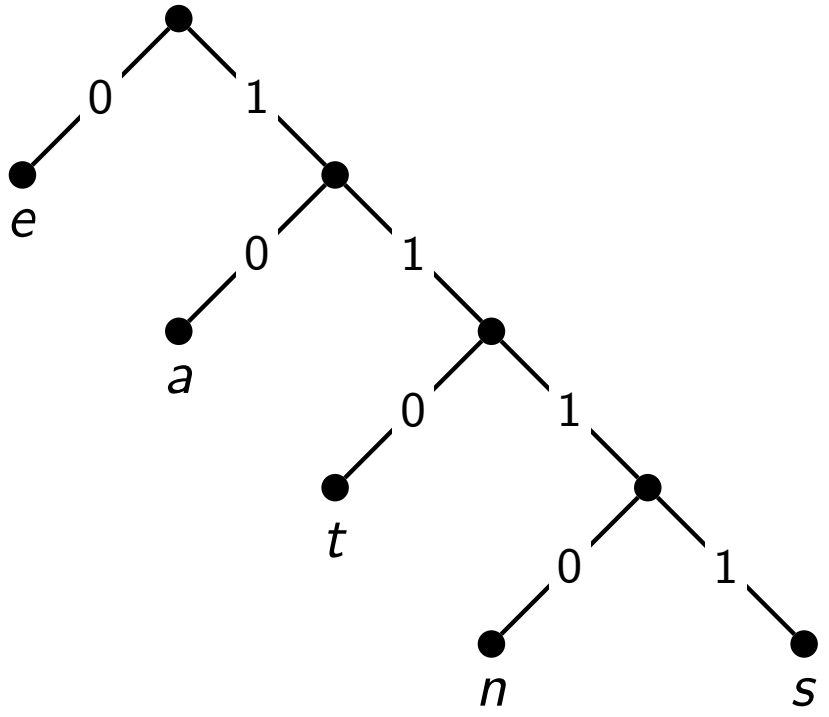
Another type of decision tree occurs when encoding letters as binary digits. Suppose that we want to encode the letters e, a and t. If we simply set

$$e \mapsto 0, \qquad a \mapsto 1, \qquad t \mapsto 01,$$

then the string 0101 could represent either "eaea" or "tt" or "eat". We would therefore like to find a method that encodes letters as strings without ambiguity. More precisely, we want to ensure that the bit string encoding of a letter never occurs as the first part of the encoding for any other letter. Encodings with this property are called ***prefix codes***

We can construct prefix codes from binary trees as follows: suppose we want to encode the letters e, a, t, n and s. Then we assign each letter the code found by concatenating the labels of the edges of the path joining the root to the letter in the following tree.

Prefix Codes



Hence we have

$$e \mapsto 0, \qquad a \mapsto 10, \qquad t \mapsto 110, \qquad n \mapsto 1110, \qquad s \mapsto 1111.$$

Huffman Coding

Clearly, a prefix code uses longer bit strings to ensure the non-ambiguity of the encoding. Given a text that is to be encoded, we would like to minimize the total number of bits necessary for the encoding. Thus, we would like to encode common letters using short bit strings and rarely occurring letters using longer bit strings. One algorithm that implements this goal is called ***Huffmann coding***.

Huffman encoding works as follows: an initial forest of rooted trees consisting of a single vertex labeled with the letters and their frequencies is given. In each step, the two trees with the smallest labels are joined to a new root, which is labeled with the sum of the labels of the roots of the two trees. This is repeated until a single tree is obtained. The left children are labeled with zeroes while the right children are labeled with ones.

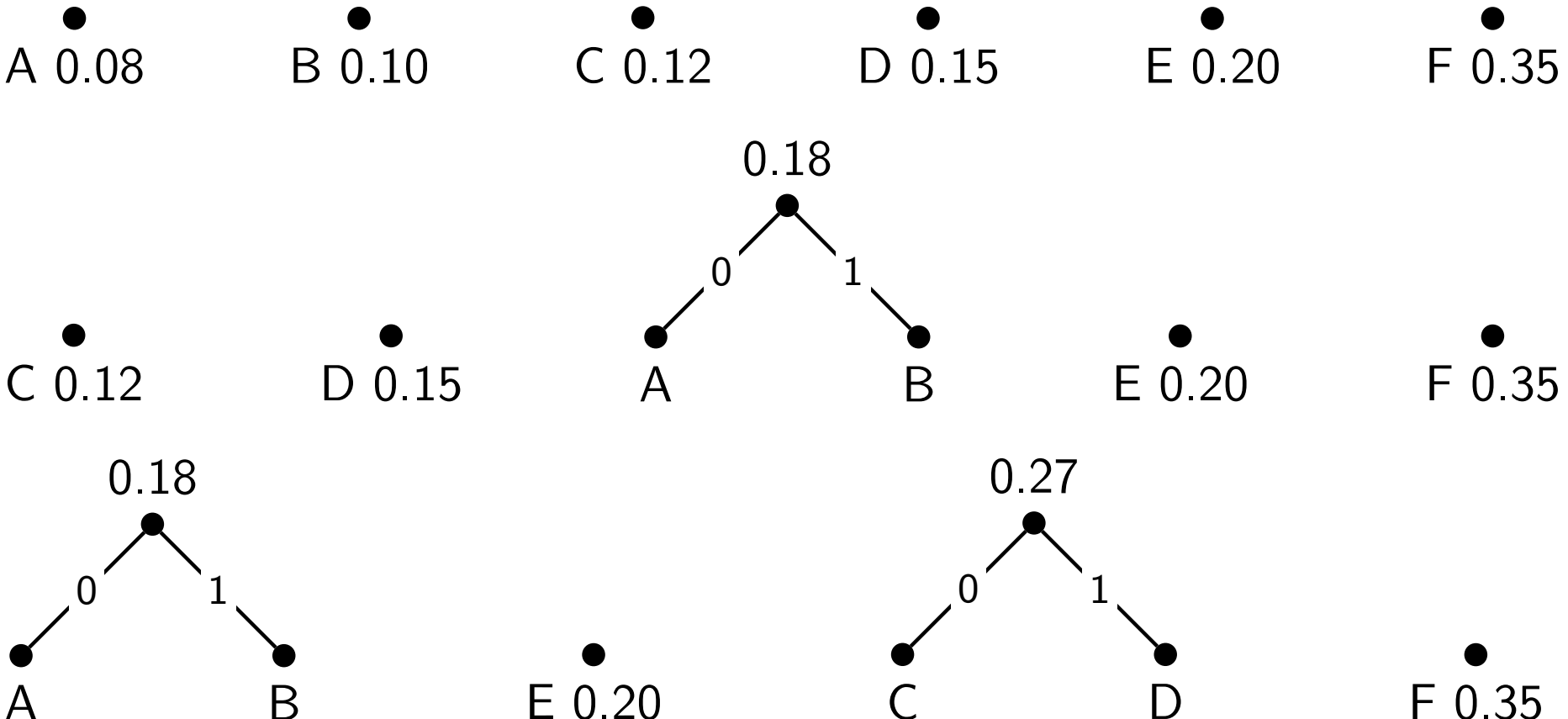
Huffman coding is a greedy algorithm and it is optimal in the sense that no binary prefix code can encode symbols (letters) using fewer bits. This will be shown in the assignments.

Huffman Coding

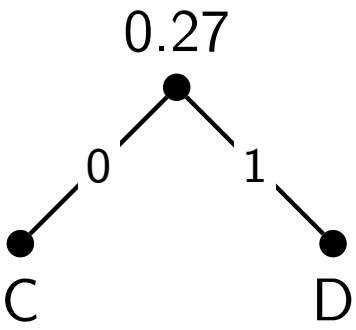
3.4.20. Example. Suppose it is known that the following letters appear with the given frequencies:

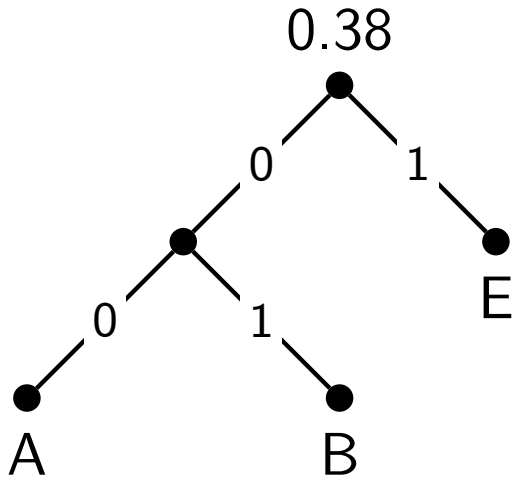
A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35.

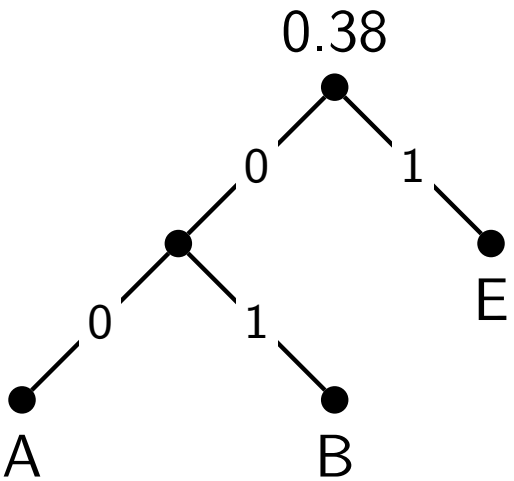
We then have the following iterative trees:

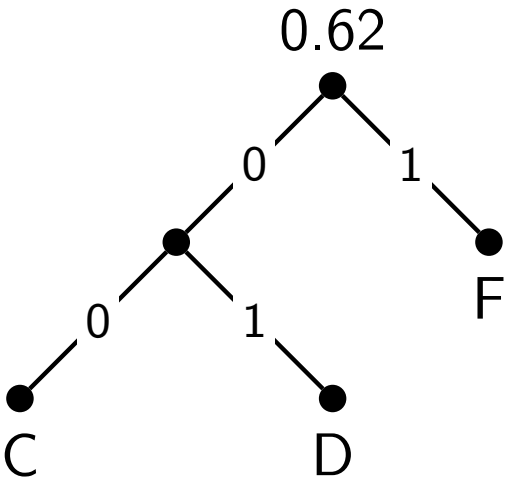


Huffman Coding

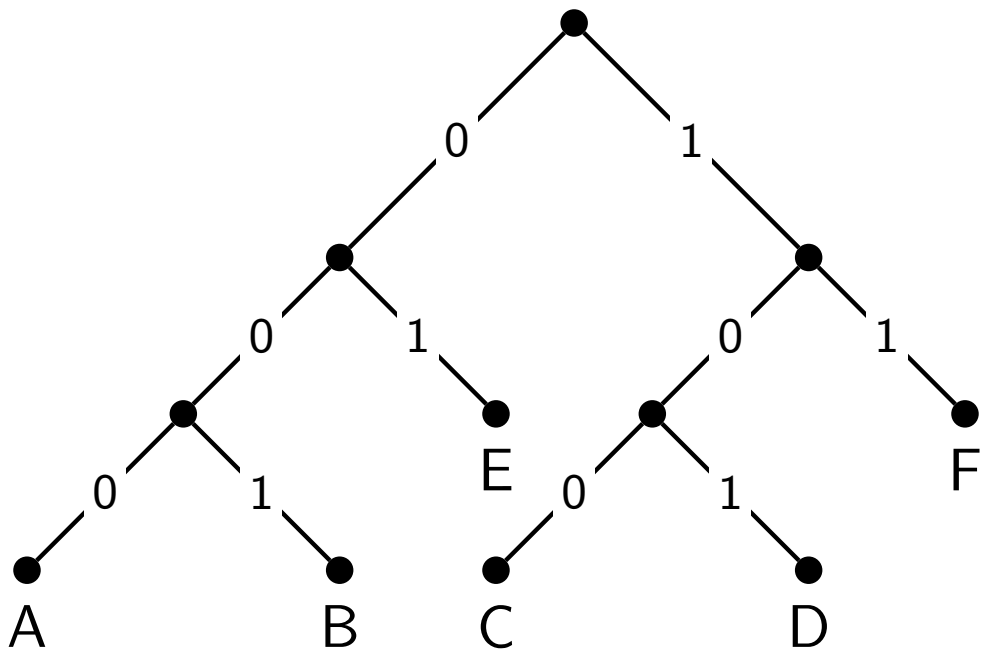








Huffman Coding



Huffman coding thus gives

$$A \mapsto 000, \quad B \mapsto 001, \quad C \mapsto 100, \quad D \mapsto 101, \quad E \mapsto 01, \quad F \mapsto 11.$$

Huffman coding leads to the shortest number of bits used for encoding the text with the given symbol frequencies (see assignments). The scheme has been refined in various ways, for example to account for initially unknown symbol frequencies that are continually updated as more text is encoded.

Universal Address System

Many types of trees (such as binary search trees) store information, so handling this data becomes important. The basis is a systematic way of labeling the vertices of trees.

3.4.21. Universal Address System.

(i) Label the root with the integer 0. Then label its k children (at level 1) from left to right with $1, 2, 3, \ldots, k$.

(ii) For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with $A.1, A.2, \ldots, A.k_v$.

The Universal Address System gives a total ordering of the vertices in a tree through the lexicographic ordering of its labels.

Tree Traversal (Preorder)

We now consider the problem of ***traversing*** an entire tree, i.e., listing (***visiting***) each vertex of a tree. There are three basic algorithms that achieve this: ***preorder traversal***, ***inorder traversal*** and ***postorder traversal***. We will give recursive definitions for each of them.

3.4.22. Definition. Let T be an ordered rooted tree with root r.

(i) If T consists only of r, then r is the preorder traversal of T.

(ii) If $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right in T, the preorder traversal begins by visiting r. Then, $T_1, T_2, \ldots, T_n$ are sequentially traversed in preorder.

3.4.23. Remark. In preorder traversal, the vertices of a tree are visited in the lexicographic ordering of the Universal Address System.

Tree Traversal (Inorder)

3.4.24. Definition. Let T be an ordered rooted tree with root r.

(i) If T consists only of r, then r is the inorder traversal of T.

(ii) If $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right in T, the inorder traversal begins by traversing T_1 in inorder, then visiting r. Then, $T_2, T_3, \ldots, T_n$ are sequentially traversed in inorder.

Tree Traversal (Postorder)

3.4.25. Definition. Let T be an ordered rooted tree with root r.

(i) If T consists only of r, then r is the postorder traversal of T.

(ii) If $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right in T, the postorder traversal sequentially traverses $T_1, \ldots, T_n$ in postorder traversal and then visits r.

An Easy Shortcut for Tree Traversal

To find the different types of traversal of a tree easily, we can use the following shortcut:

(i) Draw a counter-clockwise curve around the ordered rooted tree starting at the root, moving along the edges.

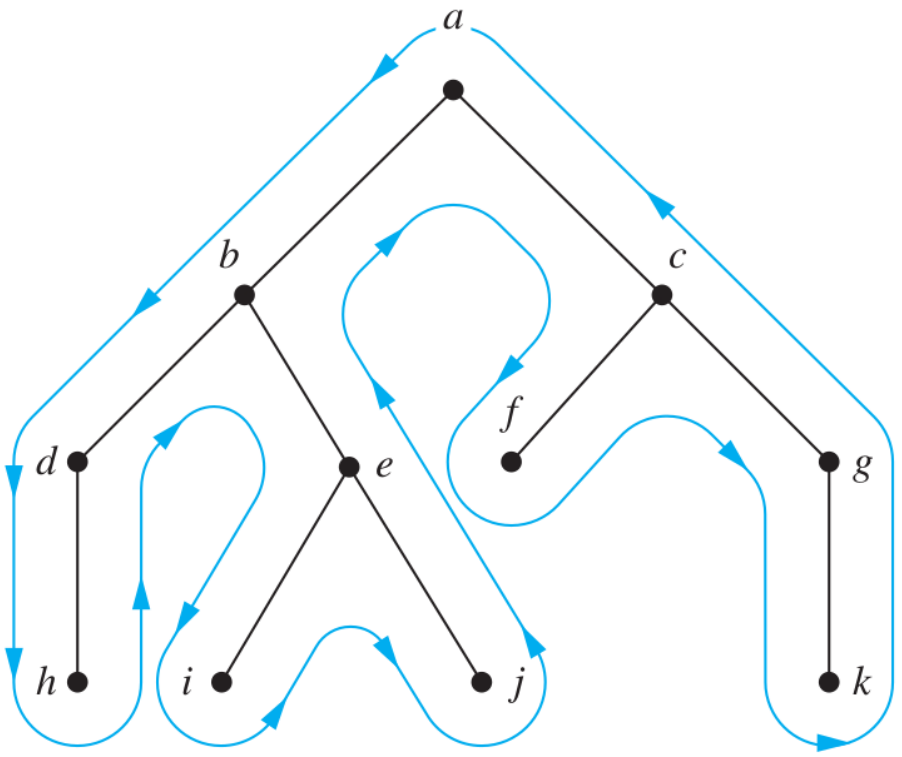
(ii) We can list the vertices in preorder by listing each vertex the first time this curve passes it.

(iii) We can list the vertices in inorder by listing a leaf the first time the curve passes it and listing each internal vertex the second time the curve passes it.

(iv) We can list the vertices in postorder by listing a vertex the last time it is passed on the way back up to its parent.

An Easy Shortcut for Tree Traversal

3.4.26. Example. Consider the following tree:

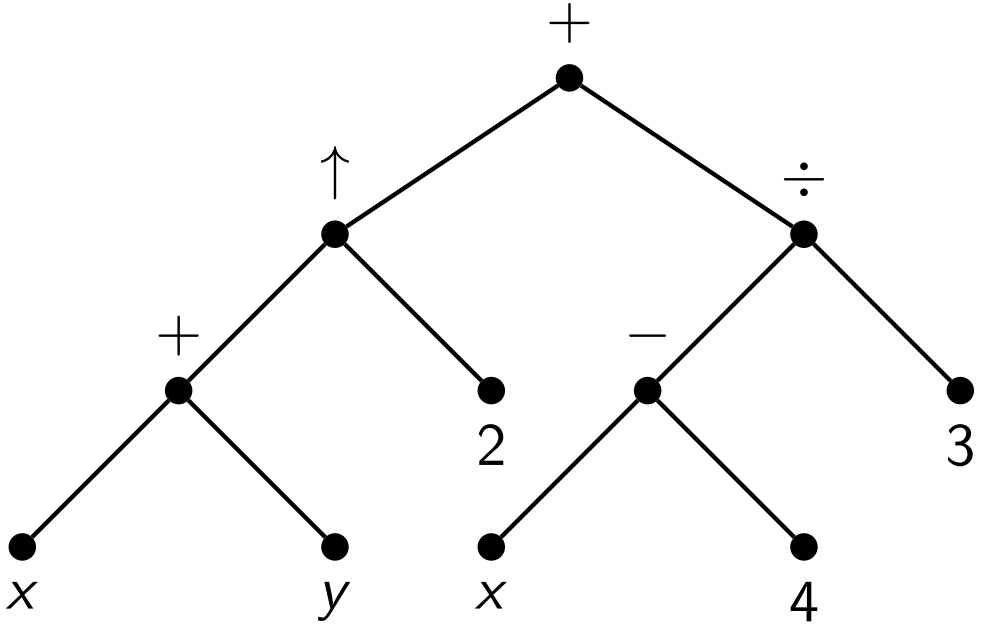


- Preorder traversal gives
  a, b, d, h, e, i, j, c, f, g, k.
- Inorder traversal gives
  h, d, b, i, e, j, a, f, c, k, g.
- Postorder traversal gives
  h, d, i, j, e, b, f, k, g, c, a.

Arithmetic Operations as Trees

Various mathematical expressions, such as compound statements in logic, unions, intersections and differences in set theory and arithmetic operations can be represented as ordered trees. As an example, we will discuss the arithmetic operations $+, -, \cdot, \div, \uparrow$ (the last representing exponentiation).

3.4.27. Example. The expression $(x+y)^2 + (x-4)/3$ can be written as $(x+y) \uparrow 2 + (x-4) \div 3$. As an ordered tree, the expression can be represented as follows:



Infix Notation

It is clear that an ordered binary tree of this type uniquely determines the arithmetic expression. Note that the internal vertices of the tree correspond to ***operators*** while the leaves are ***operands***.

We can list the vertices of the tree according the inorder traversal; this gives the operators and operands of the original expression in their original order and is known as ***infix notation***. However, the expression itself is not determined by its infix notation unless parentheses are added:

3.4.28. Example. The expression $(x+y)^2+(x-4)/3$ in infix notation is $x \ + \ y \ \uparrow \ 2 \ + \ x \ - \ 4 \ \div \ 3$.

Prefix Notation

We can also list the vertices of the tree, for example, in preorder; this is known as ***prefix notation***.

3.4.29. Example. The expression $(x+y)^2+(x-4)/3$ in prefix notation is $+\ \uparrow\ +\ x\ y\ 2\ \div\ -\ x\ 4\ 3$.

The prefix notation corresponds to listing in the order of the Universal Address System and in this system, each binary operator will precede its operands. thus, one can evaluate a prefix expression from right to left:

- Start at the right end of the expression and read the symbols one by one. When an operator is encountered, its operands are the two symbols immediately to the right.
- Perform the operation on the operands; the result gives a new operand.

It can be shown (see assignments) that the prefix notation uniquely determines the arithmetic expression without the need for parentheses.

Postfix Notation

Finally, listing the vertices in postorder is known as ***postfix notation***. As prefix notation, it does not require parentheses. A postfix expression is evaluated from left to right:

- Start at the left end of the expression and read the symbols one by one. When an operator is encountered, its operands are the two symbols immediately to the left.
- Perform the operation on the operands; the result gives a new operand.

3.4.30. Example. The expression $(x+y)^2+(x-4)/3$ in postfix notation is $x\ y\ +\ 2\ \uparrow\ x\ 4\ -\ 3\ \div\ +$.

3.4.31. Remark. Postfix notation is also known as ***Reverse Polish Notation***. It is used by many scientific calculators, known as ***RPN calculators***.

Spanning Trees

In many applications it is of interest to find a subgraph of a graph that includes every vertex and is also a tree.

3.4.32. Definition. Let G be a simple graph. A ***spanning tree*** of G is a subgraph of G that is a tree containing every vertex of G.

3.4.33. Theorem. A simple graph is connected if and only if it has a spanning tree.

Proof.

($\Rightarrow$) Let G be a connected graph. If G has no simple circuits, then G is a spanning tree of itself. If G has a simple circuit, remove an edge from this circuit. The resulting subtree will remain connected. Continue removing edges until there are no more simple circuits. The result is a subgraph that is a tree and still contains all vertices of G. This is a spanning subtree of G.

Spanning Trees

Proof (continued).

($\Leftarrow$) If a graph G has a spanning tree T, then any two vertices of G are connected by a path in T. Since the edges of T are also edges of G, these vertices are connected by a path in G, so G is a connected graph. □

The method outlined in the proof of Theorem 3.4.33 (finding simple circuits and removing edges) is not useful for finding spanning trees in practice due to the large number of steps required to identify simple circuits. Instead, two methods are commonly used, called ***depth-first search*** and ***breadth-first search***.

Depth-First Search

We can build a spanning tree for a connected simple graph using ***depth-first search***. We will form a rooted tree, and the spanning tree will be the underlying undirected graph of this rooted tree.

(i) Arbitrarily choose a vertex of the graph as the root.

(ii) Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.

(iii) Continue adding vertices and edges to this path as long as possible.

(iv) If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree. If not, move back to the previous vertex in the path and attempt to add another edge as per (ii) and (iii).

Depth-First Search

Depth-first search is also called ***backtracking*** (for obvious reasons). The edges selected by depth-first search of a graph are called ***tree edges***. All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree (see assignments). These edges are called ***back edges***.

We say that we ***explore*** from a vertex v when we carry out the steps of depth-first search beginning when v is added to the tree and ending when we have backtracked back to v for the last time.

When we add an edge connecting a vertex v to a vertex w, we finish exploring from w before we return to v to complete exploring from v. This makes it easy to write a recursive algorithm for the depth-first search.

Depth-First Search

Each edge is considered at most twice for inclusion with one of its endpoints in T: once for each vertex to which it is incident. Since a simple graph with n vertices has at most $n(n-1)/2$ edges, the algorithm requires $O(n^2)$ steps.

3.4.34. Theorem. The depth-first search yields a spanning tree T of a connected graph G.

Sketch of Proof.
Let T_0 be the tree consisting of a single vertex of G. Then T_{n+1} is found from T_n by adding an edge connecting a vertex in T_n with a vertex not already in T_n together with this second vertex to T_n and exploring from this vertex. For each n, T_n contains no simple circuits (prove this by induction!) and T_{n+1} is connected if T_n is connected, so each T_n is a tree. Since G is connected, each vertex of G will be contained in some T_n (check this!), so the tree T obtained after the depth-first search terminates is a spanning tree. □

Breadth-First Search

We can also produce a spanning tree of a simple graph by the use of ***breadth-first search***. Again, a rooted tree will be constructed, and the underlying undirected graph of this rooted tree forms the spanning tree.

(i) Arbitrarily choose a vertex of the graph as the root.

(ii) Add all edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in the spanning tree. Arbitrarily order them.

(iii) For each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit. Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree.

(iv) Repeat (iii) until all vertices in the tree have been added.

We omit the pseudocode; like the depth-first search, this algorithm requires $O(n^2)$ steps.

Representing Relations as Digraphs

One application of graphs is the visualization of relations, which were first introduced on Slide 118. For a relation on a set M, we can use a directed multigraph where we take the vertex set $V = M$ and the edge set $E = R$.

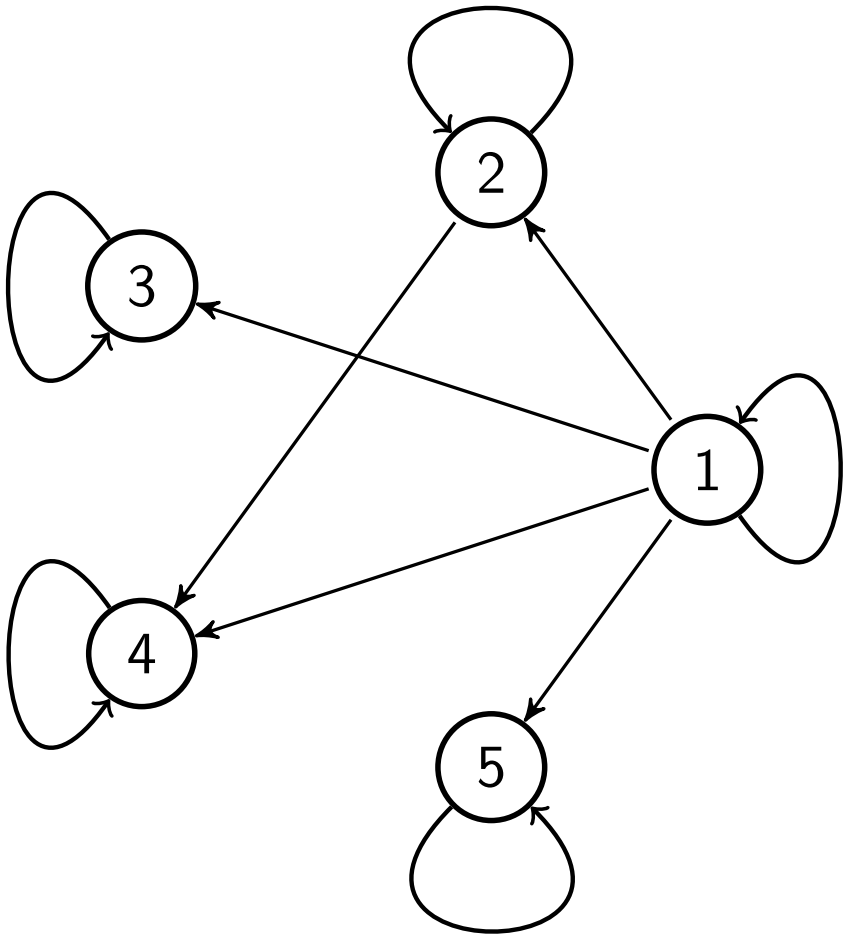
3.5.1. Example. Consider the relation on

$$M = \{1, 2, 3, 4, 5\}$$

given by

$$(a, b) \in R \quad \Leftrightarrow \quad a \mid b.$$

The figure at right shows R as a directed graph with vertices $V = M$ and edges $E = R$.



Partial Orderings

We have previously studied equivalence relations in some depth. We now introduce another important class of relations.

3.5.2. Definition. Let M be a set and R a relation on M that is reflexive, transitive and antisymmetric (see Definition 1.4.3). Then R is called a ***partial ordering*** on M and the pair (M, R) is called a ***partially ordered set*** or ***poset***.

3.5.3. Examples.

(i) The relation $\leq$ is a partial ordering on $\mathbb{Z}$. Here "the relation $\leq$" is supposed to mean "the set of all pairs $(a, b) \in \mathbb{Z}^2$ such that $a \leq b$." We will use this type of shorthand throughout this section.

(ii) Let M be a set. The relation $\subset$ is a partial ordering on the power set $\mathscr{P}(M)$.

(iii) The relation $|$ is a partial ordering on $\mathbb{N} \setminus \{0\}$.

Total Orderings

Given a poset M we will use the general symbol $\preccurlyeq$ to indicate the ordering relation if no specific set/relation (as in the above examples) is provided. Thus, a poset is denoted $(M, \preccurlyeq)$.

3.5.4. Definition. Let $(M, \preccurlyeq)$ be a poset.

(i) Let $a, b \in M$. If $a \preccurlyeq b$ or $b \preccurlyeq a$, then a and b are called ***comparable***, otherwise they are called ***incomparable***.

(ii) If any two elements of M are comparable we say that $(M, \preccurlyeq)$ is a ***totally ordered*** or ***linearly ordered*** set and the relation $\preccurlyeq$ is called a ***total*** or ***linear*** ordering. A totally ordered set is also called a ***chain***.

3.5.5. Examples.

(i) The relation $\leq$ is a total ordering on $\mathbb{Z}$.

(ii) Let M be a set. The relation $\subset$ is not a total ordering on the power set $\mathscr{P}(M)$.

(iii) The relation $|$ is not a total ordering on $\mathbb{N} \setminus \{0\}$.

Well-Ordered Sets

3.5.6. Definition. Let $(M, \preccurlyeq)$ be a poset. Then $(M, \preccurlyeq)$ is called a ***well-ordered set*** if $\preccurlyeq$ is a total ordering and every non-empty subset of M has a least element.

3.5.7. Examples.

(i) The poset $(\mathbb{Z}, \leq)$ is not well-ordered, because it does not have a least element.

(ii) The poset $(\mathbb{N}, \leq)$ is well-ordered.

(iii) The poset $(\mathbb{N}^2, \preccurlyeq)$ with

$$(a_1, a_2) \preccurlyeq (b_1, b_2) \quad \Leftrightarrow \quad (a_1 < b_1) \vee \big((a_1 = b_1) \wedge (a_2 \leq b_2)\big)$$

is well-ordered. (This is called the ***lexicographic ordering***.)

General Lexicographic Ordering

We can define a lexicographic ordering on the cartesian product of n posets as follows:

3.5.8. Definition. Let $(A_1, \preccurlyeq_1), \dots, (A_n, \preccurlyeq_n)$ be a family of n posets. Then the set $A_1 \times \cdots \times A_n$ together with the ordering relation

$$(a_1, \dots, a_n) \preccurlyeq (b_1, \dots, b_n) \quad :\Leftrightarrow \quad \Big(\underset{1 \le i \le n}{\forall} a_i = b_i \Big) \vee \big(a_1 \prec_1 b_1 \big) \vee \Big(\underset{j \in \{2, \dots, n\}}{\exists} a_j \prec_j b_j \wedge \underset{i < j}{\forall} a_i = b_i \Big). \tag{3.5.1}$$

becomes a poset. The ordering relation (3.5.1) is called the ***lexicographic ordering*** for the cartesian product $A_1 \times \cdots \times A_n$.

If $A_1 = A_2 = \cdots = A_n$ and we have have the same partial order $\preccurlyeq$ on each A_k, we use the same symbol $\preccurlyeq$ for the partial order on the cartesian product $A_1 \times \cdots \times A_n$ given by (3.5.1).

Lexicographic Ordering of Strings

3.5.9. Definition. Let $\mathscr{A}$ be an alphabet and $\preccurlyeq$ an ordering relation on the alphabet. Let $a = a_1 a_2 \ldots a_m$ and $b = b_1 b_2 \ldots b_n$ be strings of length n and m from $\mathscr{A}$. Let $t = \min(m, n)$. Then we define

$$\begin{aligned} a \prec b \quad :\Leftrightarrow \quad & \big((a_1, \ldots, a_t) = (b_1, \ldots, b_t) \wedge m < n\big) \\ & \vee \big((a_1, \ldots, a_t) \prec (b_1, \ldots, b_t)\big). \end{aligned}$$

3.5.10. Example. Let $\mathscr{A} = \{0, 1\}$ with the partial ordering $\preccurlyeq$ induced by $0 \prec 1$. Consider the bit strings $a = 110011$, $b = 11001$, $c = 1100001$, $d = 1110$. Then $c \prec b \prec a \prec d$.

Well-Ordered Induction

Recall that the principle of induction on the natural numbers is equivalent to the well-ordering principle. It seems plausible that a version of induction should hold on general well-ordered sets, and this is indeed the case.

If $(M, \preccurlyeq)$ is a poset and $a, b \in M$, we define

$$a \prec b \qquad :\Leftrightarrow \qquad (a \preccurlyeq b) \wedge (a \neq b).$$

3.5.11. Theorem. Suppose that $(M, \preccurlyeq)$ is a well-ordered set. Then the predicate $P(x)$ is true for all $x \in M$ if

$$\forall_{y \in M} \quad \left(\forall_{x \prec y} P(x) \right) \Rightarrow P(y) \tag{3.5.2}$$

Well-Ordered Induction

Proof.

Suppose that (3.5.2) is true and that there exists some $z \in M$ so that $P(z)$ is false. Then the set $A = \{x \in M : \neg P(x)\}$ is non-empty. Since $(M, \preccurlyeq)$ is well-ordered, there exists a least element a of A. Since a is the least element of A, it follows that $P(x)$ is true for all $x \prec a$. But then by (3.5.2) $P(a)$ must be true. This gives a contradiction. □

3.5.12. Remark. For this type of induction, we do not need a "basis step", i.e., we do not need to show that $P(x)$ is true for some initial element of M. For example, suppose that a is the least element of M. Then the statement $\forall_{x \prec a} P(x)$ in (3.5.2) is vacuously true (there is no element $x \prec a$) so once (3.5.2) has been established, $P(a)$ is true automatically.

Well-Ordered Induction

3.5.13. Example. Suppose that for $n, m \in \mathbb{N}$ we define

$$a_{m,n} := \begin{cases} 0 & \text{if } n = m = 0, \\ a_{m-1,n} + 1 & \text{if } n = 0 \text{ and } m > 0, \\ a_{m,n-1} + n & \text{if } n > 0. \end{cases}$$

We want to prove that

$$a_{m,n} = m + \frac{n(n+1)}{2} \qquad m, n \in \mathbb{N}. \tag{3.5.3}$$

We can use the lexicographic ordering on $\mathbb{N}^2$ introduced in Example 3.5.7 to apply well-ordered induction. In particular,

$$(m, n) \prec (m_0, n_0) \quad \Leftrightarrow \quad (m < m_0) \vee \big((m = m_0) \wedge (n < n_0)\big).$$

Take $(m_0, n_0) \in \mathbb{N}^2$ and suppose that (3.5.3) holds for all $(m, n) \prec (m_0, n_0)$.

Well-Ordered Induction

1. If $(m_0, n_0) = (0, 0)$ the statement follows immediately.
2. Consider the case $n_0 = 0$ and $m_0 > 0$. Then $a_{m_0,n_0} = a_{m_0-1,n_0} + 1$. Since $(m_0 - 1, n_0) \prec (m_0, n_0)$, we have

$$a_{m_0,n_0} = a_{m_0-1,n_0} + 1 = m_0 - 1 + \frac{n_0(n_0+1)}{2} + 1 = m_0 + \frac{n_0(n_0+1)}{2}$$

   so (3.5.3) is true for $(m, n) = (m_0, n_0)$ in this case.
3. Now consider the case $n_0 > 0$. Then $a_{m_0,n_0} = a_{m_0,n_0-1} + n_0$. Since $(m_0, n_0 - 1) \prec (m_0, n_0)$, we have

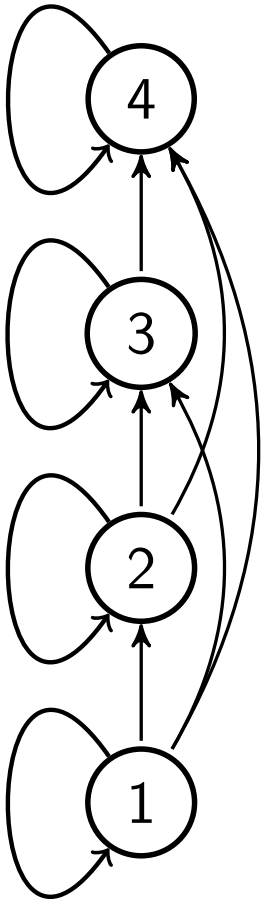
$$a_{m_0,n_0} = a_{m_0,n_0-1} + n_0 = m_0 + \frac{n_0(n_0-1)}{2} + n_0 = m_0 + \frac{n_0(n_0+1)}{2}$$

   so (3.5.3) is true for $(m, n) = (m_0, n_0)$ in this case also.

This completes the proof by induction.

Hasse Diagrams

Consider the digraph for the poset $(\{1, 2, 3, 4\}, \leq)$:

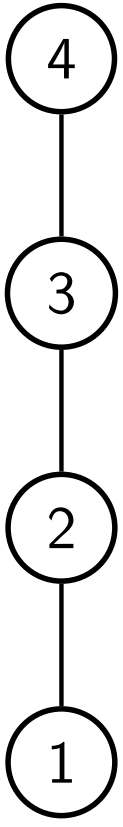


If we know that this is the graph of a partial ordering, we do not need to show all the edges:

- Since $\leq$ is reflexive, we know that loops must be present and we can omit them,
- Since $\leq$ is transitive, we can omit the edges that must be present due to transitivity.
- If we agree that the arrows always point upwards, we can omit the arrowheads.

Hasse Diagrams

Applying these principles we end up with the following graph, called a ***Hasse diagram***:

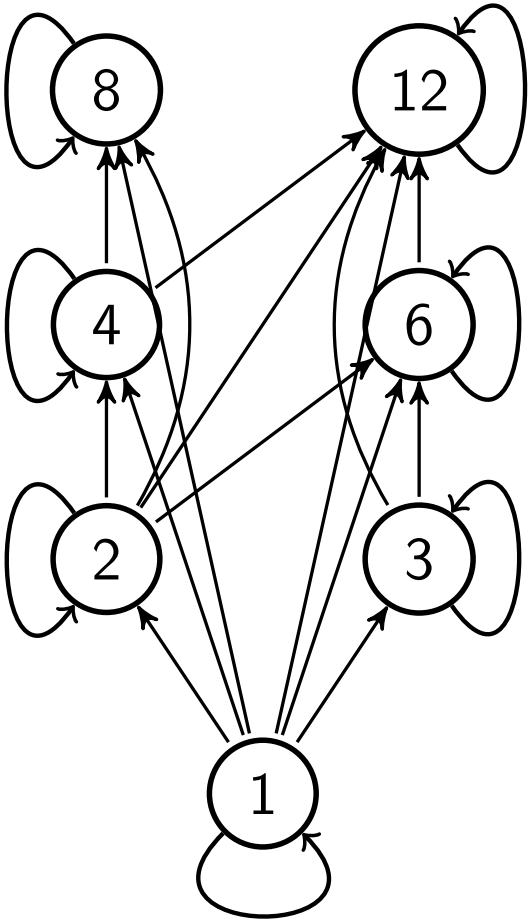


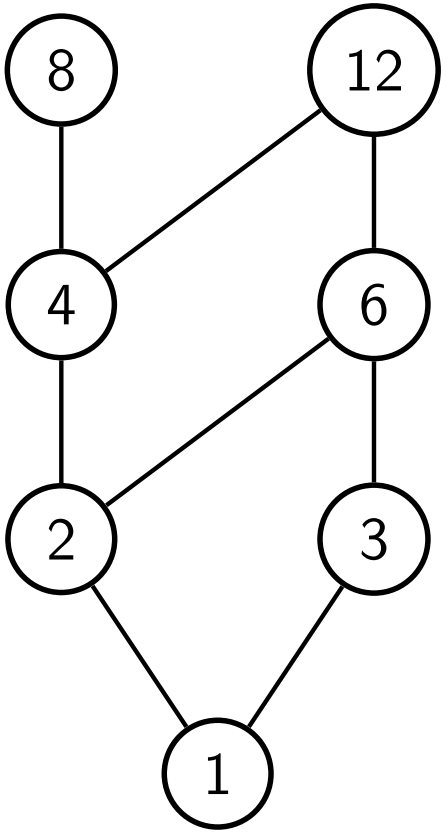
We can perform this basic procedure with any graph of an ordering relation. It is easy to see that the original graph of the partial ordering relation can be regained from the Hasse diagram, so no information is lost by simplifying the graph in this way.

The Hasse diagran can be very useful to identify basic properties of the relation

Hasse Diagrams

3.5.14. Example. Consider the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$. Then the graph and the Hasse diagram of the relation are





Maximal and Minimal Elements

3.5.15. Definition. Let $(M, \preccurlyeq)$ be a poset and suppose $a \in M$. We say that

(i) The element a is ***maximal*** if there exists no $b \in M$ such that $a \prec b$.

(ii) The element a is ***minimal*** if there exists no $b \in M$ such that $b \prec a$.

(iii) The element a is the ***greatest element of*** M if $b \preccurlyeq a$ for all $b \in M$.

(iv) The element a is the ***least element of*** M if $a \preccurlyeq b$ for all $b \in M$.

3.5.16. Remark. The greatest and least elements, if they exist, are unique (see exercises).

3.5.17. Example. Consider the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$ of Example 3.5.14. There,

- The element 1 is minimal;
- The elements 8 and 12 are maximal;
- The element 1 is the least element;
- There is no greatest element.

Subsets and Bounds

3.5.18. Definition. Let $(M, \preccurlyeq)$ be a poset and suppose that $A \subset M$.

(i) An element $u \in M$ such that $a \preccurlyeq u$ for all $a \in A$ is is called an ***upper bound for*** A.

(ii) An element $l \in M$ such that $l \preccurlyeq a$ for all $a \in A$ is is called a ***lower bound for*** A.

(iii) An upper bound x for A such that $x \preccurlyeq u$ for all upper bounds u of A is called a ***least upper bound for*** A. We then write $x = \sup A$.

(iv) A lower bound x for A such that $l \preccurlyeq x$ for all lower bounds l of A is called a ***greatest lower bound for*** A. We then write $x = \inf A$.

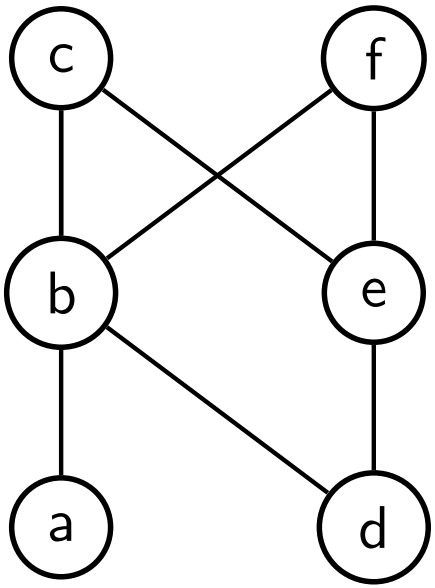
3.5.19. Remark. The greatest and least upper bounds, if they exist, are unique (see exercises).

Subsets and Bounds

3.5.20. Example. Consider the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$ of Example 3.5.14 and let $A = \{1, 2, 3\}$. Then

- The elements 6 and 12 are upper bounds for A;
- The element 1 is the (only) lower bound for A;
- $\inf A = 1$, $\sup A = 6$.

3.5.21. Example. Consider the poset with Hasse diagram as shown below:



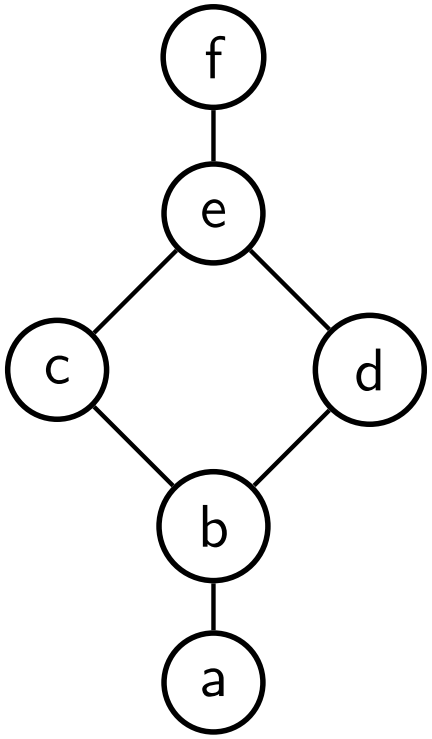
The subset $A = \{a, b, e\}$ has two upper bounds, c and f, but no least upper bound.

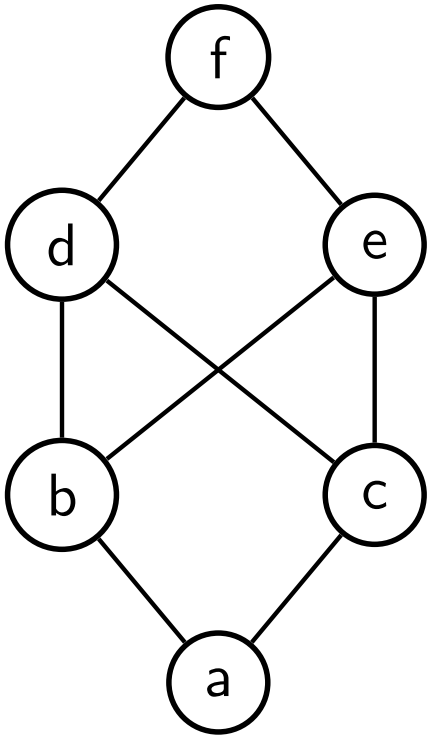
There is no lower bound for A.

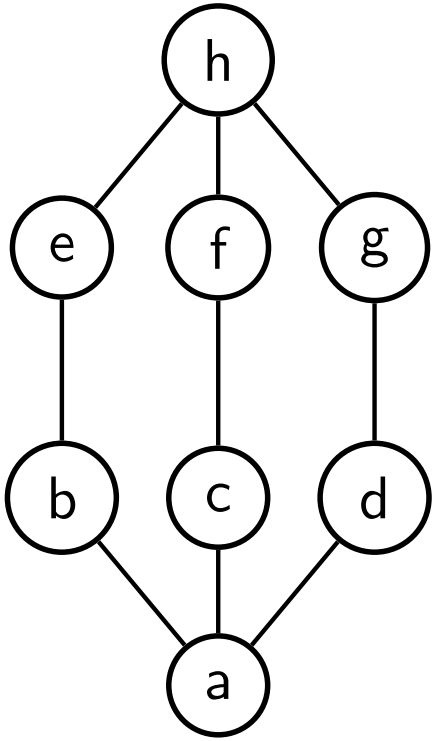
Lattices

3.5.22. Definition. Let $(M, \preccurlyeq)$ be a poset and suppose that every pair of elements of M has a greatest lower and a least upper bound. Then M is called a ***lattice***.

3.5.23. Example. The left and right diagrams are lattices, the middle one is not ($\sup\{b, c\}$ does not exist) .







Topological Sorting

Lattices are used in problems involving information flow - we will study some examples in the exercises.

We now turn to another problem: Let $(M, \preccurlyeq)$ be a poset which is not totally ordered. A total ordering $\preccurlyeq_t$ such that $a \preccurlyeq_t b$ whenever $a \preccurlyeq b$ is said to be ***compatible*** with $\preccurlyeq$. The problem is to find a compatible total ordering for a poset $(M, \preccurlyeq)$. This is called ***topological sorting*** of $(M, \preccurlyeq)$. In our discussion, the following result will be useful.

3.5.24. Lemma. Let $(M, \preccurlyeq)$ be a finite, non-empty poset. Then there exists a minimal element of M.

Proof.
Choose some $a \in M$. If a is minimal, we are finished. If it is not minimal, there exists some $a_1 \prec a$. We repeat this process with a_1. Since M is finite, we must obtain a minimal element after a finite number of iterations. □

Topological Sorting

We now give a topological sorting algorithm as follows: Let $(M, \preccurlyeq)$ be a finite, non-empty poset.

1. Find a minimal element a_1 of $(M, \preccurlyeq)$.
2. Define $(M', \preccurlyeq')$, where $M' = M \setminus \{a_1\}$ and $\preccurlyeq'$ actually is the restriction of $\preccurlyeq$ from M to M'. This is also a poset.
3. If $M' \neq \emptyset$, repeat Step 1 with $(M, \preccurlyeq)$ replaced by $(M', \preccurlyeq')$, obtaining a least element a_2 of M'.

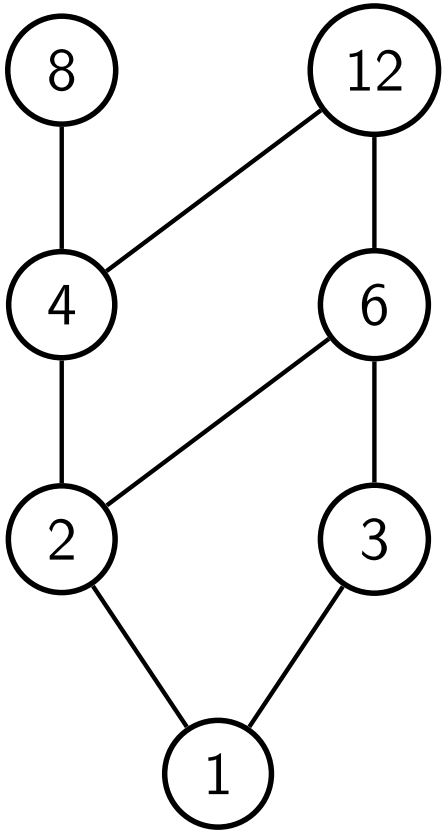
The above loop continues until $M = \emptyset$. We obtain a sequence of elements $a_1, \ldots, a_n$, $n = \operatorname{card} M$, and define a total ordering $\preccurlyeq_t$ by

$$a_1 \prec_t a_2 \prec_t a_3 \prec_t \cdots \prec_t a_n.$$

It is easy to see that $\preccurlyeq_t$ is compatible with $\preccurlyeq$: if $a \preccurlyeq b$, then a will have been selected before b, so $a \preccurlyeq_t b$.

Topological Sorting

3.5.25. Example. Consider the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$ of Example 3.5.14. Then a total ordering can be found by successively choosing the least (bottom) elements from the Hasse diagram:



One possible total ordering is given by

$$1 \prec_t 2 \prec_t 3 \prec_t 4 \prec_t 6 \prec_t 8 \prec_t 12$$

Another possibility is

$$1 \prec_t 3 \prec_t 2 \prec_t 6 \prec_t 4 \prec_t 12 \prec_t 8$$

A Simple Proof of Fermat's Little Theorem

We now compile some more applications of graph theory. Recall the Fermat's Little Theorem 1.9.2 from the first part of the course:

3.5.26. Fermat's Little Theorem. Let $p \in \mathbb{N}$ and $a \in \mathbb{N}$. If p is prime, then

$$p \mid (a^p - a). \tag{3.5.4}$$

We now give a quick and elegant proof using graph theory.

A Simple Proof of Fermat's Little Theorem

Proof.

Let $G = (V, E)$ be the graph with vertex set

$$V = \left\{ (a_0, \ldots, a_{p-1}) \in \mathbb{N}^p : \underset{k=1,\ldots,p}{\forall} \; 1 \leq a_k \leq a \text{ and } \underset{1 \leq i,j \leq p}{\exists} \; a_i \neq a_j \right\}$$

Then

$$\operatorname{card} V = a^p - a.$$

For any $u = (u_0, \ldots, u_{p-1}) \in V$ we also have $v = (u_{p-1}, u_0, \ldots, u_{p-2}) \in V$ and we let $\{u, v\} \in E$. In particular, E is precisely the set of all such two-element sets of p-tuples.

A Simple Proof of Fermat's Little Theorem

Proof (continued).

Each vertex of G has degree two: the two adjacent vertices to $(u_k)_{k=1,\dots,p}$ are $(u_{(k-1) \bmod p})$ and $(u_{(k+1) \bmod p})$. Then each connected component of G is a cycle with vertices

$$\{(u_{(k+j) \bmod p})_{k=1,\dots,p} : j = 0, \dots, p-1\}.$$

We now claim that each cycle has exactly p elements. Suppose that a cycle has n elements, then after shifting the entires of a tuple $(u_0, \dots, u_{p-1})$ to the right $n > 0$ times, the original tuple is obtained. If m of the entries of a tuple are equal, i.e., $u_{k_1} = u_{k_2} = \cdots = u_{k_m}$, then for each i, there is a j so that $u_{(k_i+n) \bmod p} = u_{k_j}$, or $(k_i + n) \bmod p = k_j$. Summing over all i, we have

$$k_1 + \cdots + k_m + m \cdot n \equiv k_1 + \cdots + k_m \pmod{p}.$$

A Simple Proof of Fermat's Little Theorem

Proof (continued).

This implies

$$m \cdot n \equiv 0 \pmod{p}$$

Since $m < p$ and p is prime, this implies $n \equiv 0 \pmod{p}$, so $n = p$ (taking $n > 0$). Hence, the connected components of G are cycles of length p. Since the number of connected components is an integer, we must have

$$p \mid (a^p - a).$$

□

A Proof of the Cantor-Schröder-Bernstein Theorem

Recall that we defined the size (cardinality) of sets by comparing them using maps (see Definition 2.4.1):

3.5.27. Definition. Let A, B be arbitrary sets. Then

$$\begin{aligned} \operatorname{card} A \leq \operatorname{card} B \quad &:\Leftrightarrow \quad \exists\ \varphi\colon A \to B \quad \text{injective,} \\ \operatorname{card} A = \operatorname{card} B \quad &:\Leftrightarrow \quad \exists\ \varphi\colon A \to B \quad \text{bijective.} \end{aligned}$$

We would then expect that

$$\operatorname{card} A \leq \operatorname{card} B \quad \wedge \quad \operatorname{card} B \leq \operatorname{card} A \quad \Rightarrow \quad \operatorname{card} A = \operatorname{card} B.$$

or, in words:

3.5.28. Cantor-Schröder-Bernstein Theorem. Let A, B be arbitrary sets. If there exist injective maps from A to B and from B to A, then there exists a bijective map between A and B.

A Proof of the Cantor-Schröder-Bernstein Theorem

Proof.

Suppose that there exist injective maps $\varphi\colon A \to B$, $\psi\colon B \to A$. We define a 2-colored (bipartite) graph $G = (A, B, E)$ as follows: one set of vertices is colored red and labeled with the elements of A, the other set is colored blue and labeled with the elements of B.

We denote these vertex sets by A and B also, keeping in mind that due to their colorings they are now disjoint. In fact, we can now assume without loss of generality that the original sets A and B are disjoint. (If they are not, we just add the color of each element to distinguish identical elements.)

Of course, G may have an infinite (even uncountable) number of vertices.

A Proof of the Cantor-Schröder-Bernstein Theorem

Proof (continued).

Let us define a relation on $A \cup B$ by setting

$$a \sim b \qquad :\Leftrightarrow \qquad b = \varphi(a) \quad \vee \quad a = \psi(b).$$

The edge set of G is now defined by letting G be the directed graph representing this relation.

Every vertex in G is the initial vertex of an edge, since φ is defined on all of A and ψ is defined on all of B. Furthermore, every vertex is the terminal vertex of at most one edge, since φ and ψ are injective. Hence, the degree of each vertex is either 1 or 2.

A Proof of the Cantor-Schröder-Bernstein Theorem

Proof (continued).

We now examine the connected components of G. These can take the following forms:

- A finite, simple circuit of even length,
- An infinite path with initial vertex v_0, $(v_0, v_1, v_2, \ldots)$,
- An infinite path with no initial vertex, $(\ldots, v_{-2}, v_{-1}, v_0, v_1, v_2, \ldots)$.

Since every vertex is the initial vertex of a directed edge, there can be no finite paths that are not circuits. All circuits must be of even length by Theorem 3.3.16, so the above list is complete.

In each connected component we now remove every second directed edge. The remaining edges then give a one-to-one pairing between vertices of A and of B in each connected component. Hence, the cardinalities of the vertex sets associated to A and B in each component is the same and so the cardinalities of A and B are equal overall. □

Counting Full Binary Trees

Let us ask a seemingly simple question: how many different full binary trees with $n \in \mathbb{N} \setminus \{0\}$ internal vertices are there?

This question is related to the number of ways of sorting a list on elements, since a binary sorting algorithm can be represented as a binary tree with leaves corresponding to final arrangements of the list after sorting.

In order to get a grasp on the question, it is useful to introduce an alternative, recursive definition of full binary trees:

3.6.1. Definition.

(i) The tree consisting of a single vertex r is a ***full binary tree*** with root r.

(ii) If T_1 and T_2 are full binary trees, then the rooted tree formed by adding a new vertex r and edges joining r to the roots of T_1 and T_2 is a full binary tree with root r.

The equivalence with Definition 3.4.7 can be shown using structural induction and strong induction.

Counting Full Binary Trees

We can now argue as follows: A binary tree with n internal vertices has a root and two subtrees with a total of $n-1$ vertices. If C_n denotes the number of full binary trees with n internal vertices, we then have

$$C_n = \sum_{i+j=n-1} C_i C_j = \sum_{k=0}^{n-1} C_k C_{n-1-k}, \qquad n \geq 2. \tag{3.6.1}$$

where the sum on the right is found by counting all possible forms of the two subtrees of the root. We also have

$$C_0 = C_1 = 1$$

so (3.6.1) defines a (non-linear) recurrence relation for C_n. In order to solve the relation, we need to develop a new and useful tool: generating functions.

Generating Functions

In this section, we introduce a very powerful tool from calculus that can be used to solve recurrence relations or obtain combinatorial identities: the generating function of a sequence.

3.6.2. Definition. Let (a_n) be a sequence of real numbers. If the formal power series

$$G(x) = \sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + \dots$$

has a radius of convergence $\varrho > 0$ we say that G is a ***generating function*** for (a_n).

If $(a_0, \dots, a_k)$ is a finite tuple, we define the generating function in the same way, regarding the tuple as a sequence (a_n) with $a_n = 0$ for $n > k$.

Generating Functions

3.6.3. Examples.

(i) The geometric series formula yields

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x} =: G(x) \qquad \text{for } |x| < 1$$

so G is the generating function for the sequence (a_n) with $a_n = 1$ for all $n \in \mathbb{N}$.

(ii) Consider the $(1, a, a^2, a^3, \ldots)$ for some $a \in \mathbb{R}$. This sequence has generating function

$$G(x) = \frac{1}{1-ax} = \sum_{n=0}^{\infty} a^n x^n$$

where the series has radius of convergence $\varrho = 1/a$.

Generating Functions

(iii) By the binomial theorem,

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

so the function $G(x) = (1+x)^n$ is the generating function for the $n+1$-tuple

$$\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}.$$

The Cauchy Product

Generating functions are power series with non-vanishing radii of convergence. In order to work with them effectively, we need to recall some background from calculus. In particular, we have the following result regarding the product of two power series:

3.6.4. Theorem. Let $\sum a_k$ and $\sum b_k$ be series that both converge absolutely. Then the ***Cauchy product*** $\sum c_k$,

$$c_k := \sum_{i+j=k} a_i b_j$$

converges absolutely and $\sum c_k = \left(\sum a_k\right)\left(\sum b_k\right)$. The sequence

$$(a_k) * (b_k) := (c_k), \qquad c_k := \sum_{i+j=k} a_i b_j,$$

is called the ***convolution*** of the sequences (a_k) and (b_k).

The Cauchy Product

3.6.5. Example. The Cauchy product is often used in calculus to establish the functional equation for the exponential function. The exponential function can be defined to be

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$

Then

$$\begin{aligned}(\exp x)(\exp y) &= \left(\sum_{n=0}^{\infty} \frac{x^n}{n!}\right)\left(\sum_{m=0}^{\infty} \frac{y^m}{m!}\right) = \sum_{n=0}^{\infty}\left(\left(\frac{x^k}{k!}\right) * \left(\frac{y^k}{k!}\right)\right)_n \\ &= \sum_{n=0}^{\infty} \sum_{l+m=n} \frac{1}{l!m!} x^l y^m = \sum_{n=0}^{\infty} \frac{1}{n!} \sum_{l+m=n} \frac{n!}{l!m!} x^l y^m \\ &= \sum_{n=0}^{\infty} \frac{1}{n!}(x+y)^n \\ &= \exp(x+y)\end{aligned}$$

The Cauchy Product

This yields the functional equation

$$(\exp x)(\exp y) = \exp(x + y).$$

We also remark that $\exp x = e^x$ is the generating function for the sequence (a_n) with $a_n = 1/n!$.

3.6.6. Corollary. Let $\sum a_k x^k$ and $\sum b_k x^k$ be power series that both converge absolutely for $|x| < \varrho$. Then

$$\Big(\sum_{l=0}^{\infty} a_l x^l\Big)\Big(\sum_{m=0}^{\infty} b_m x^m\Big) = \sum_{n=0}^{\infty} c_n x^n$$

with

$$c_n = \sum_{i+j=n} a_i b_j$$

The Cauchy Product

3.6.7. Example. For any $n \in \mathbb{Z}_+$ and $k = 0, \dots, n$,

$$\binom{n}{0}\binom{n}{k} + \binom{n}{1}\binom{n}{k-1} + \cdots + \binom{n}{k}\binom{n}{0} = \binom{2n}{k}.$$

We can write this identity as

$$\sum_{i=0}^{k} \binom{n}{i}\binom{n}{k-i} = \sum_{i+j=k} \binom{n}{i}\binom{n}{j} = \binom{2n}{k} \tag{3.6.2}$$

To prove (3.6.2), we consider the generating function of the finite sequence $\binom{2n}{k}$, $k = 0, \dots, 2n$, given by

$$G(x) = (1+x)^{2n} = \sum_{k=0}^{2n} \binom{2n}{k} x^k. \tag{3.6.3}$$

The Cauchy Product

For $k = 0, \ldots, n$ the right-hand-side of (3.6.2) is given by the coefficient of x^k in $G(x)$. Now

$$\begin{aligned}(1+x)^{2n} &= \left((1+x)^n\right)^2 \\ &= \Big(\sum_{l=0}^{n} \binom{n}{l} x^l\Big)\Big(\sum_{m=0}^{n} \binom{n}{m} x^m\Big) \\ &= \sum_{k=0}^{2n}\Big(\sum_{i+j=k} \binom{n}{i}\binom{n}{j}\Big)x^k.\end{aligned}$$

Comparing with (3.6.3) gives (3.6.2). The formula (3.6.2) was known to the Chinese mathematician Zhu Shijie (朱世杰) who lived around AD 1300. Setting $k = n$ we obtain the special case

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n},$$

which is known as ***Vandermonde's convolution formula***.

The Binomial Series

We will need some more background from basic calculus to employ generating functions to their full potential. Recall that the generalized binomial coefficients are defined by

$$\binom{\alpha}{0} := 1, \qquad \binom{\alpha}{j} := \frac{\alpha(\alpha-1)\dots(\alpha-j+1)}{j!}, \qquad j \in \mathbb{N},\ \alpha \in \mathbb{R}.$$

The following theorem should be known from calculus:

3.6.8. Theorem. Let $-1 < x < 1$ and $\alpha \in \mathbb{R}$. Then

$$(1+x)^\alpha = \sum_{n=0}^{\infty} \binom{\alpha}{n} x^n$$

where the series on the right converges absolutely.

The Binomial Series

3.6.9. Example. For $\alpha = 1/2$, we have the binomial series

$$\sqrt{1+x} = \sum_{n=0}^{\infty} \binom{1/2}{n} x^n.$$

The first term, $n = 0$, evaluates to

$$\binom{1/2}{0} x^0 = 1.$$

For $n \geq 1$ we have

$$\begin{aligned}
\binom{1/2}{n} &= \frac{1/2(1/2-1)(1/2-2)\dots(1/2-(n-1))}{n!} \\
&= (-1)^{n-1}\frac{1/2(1-1/2)(2-1/2)\dots(n-1-1/2)}{n!} \\
&= \frac{(-1)^{n-1}}{2^n}\frac{1(2-1)(4-1)\dots(2(n-1)-1)}{n!}
\end{aligned}$$

The Binomial Series

Continuing,

$$\begin{aligned}\binom{1/2}{n} &= \frac{(-1)^{n-1}}{2^n}\frac{1(2-1)(4-1)\ldots(2(n-1)-1)}{n!}\\ &= \frac{(-1)^{n-1}}{2^n}\frac{1\cdot 1\cdot 3\cdot 5\cdots(2n-3)}{n!}\\ &= \frac{(-1)^{n-1}}{2^n}\frac{1}{2^{n-1}(n-1)!}\frac{1\cdot 2\cdot 3\cdot 4\cdot 5\cdots(2n-2)}{n!}\\ &= -2\frac{(-1)^n}{4^n}\frac{1}{n}\frac{(2n-2)!}{(n-1)!(n-1)!} = -2\frac{(-1)^n}{4^n}\frac{1}{n}\binom{2n-2}{n-1}.\end{aligned}$$

It follows that

$$\sqrt{1+x} = 1 - 2\sum_{n=1}^{\infty}\frac{(-1)^n}{4^n}\binom{2n-2}{n-1}\frac{x^n}{n} \tag{3.6.4}$$

for $|x| < 1$.

Counting Full Binary Trees

We will now find a closed formula for the numbers C_n. Recall that they satisfy the recurrence relation (3.6.1),

$$C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k} = \sum_{j+k=n} C_k C_j, \qquad n \in \mathbb{N}.$$

with $C_0 = 1$. We define the generating function of the sequence (C_n) to be the formal power series

$$c(x) := \sum_{n=0}^{\infty} C_n x^n.$$

Then

$$c(x)^2 = \Big(\sum_{j=0}^{\infty} C_j x^j\Big)\Big(\sum_{k=0}^{\infty} C_k x^k\Big) = \sum_{n=0}^{\infty}\Big(\sum_{j+k=n} C_k C_j\Big)x^n$$

$$= \sum_{n=0}^{\infty} C_{n+1} x^n = \frac{1}{x}(c(x) - 1)$$

Counting Full Binary Trees

It follows that

$$c(x) = 1 + xc(x)^2.$$

This implies that

$$c(x) = \frac{1 - \sqrt{1-4x}}{2x} = \frac{2}{1 + \sqrt{1-4x}}$$

From the last quotient we see that $c(x)$ actually has a power series expansion at $x = 0$. Expanding the first quotient using (3.6.4),

$$c(x) = \frac{1}{2x}(1 - \sqrt{1-4x}) = \frac{1}{2x} \cdot 2 \sum_{n=1}^{\infty} \frac{(-1)^n}{4^n} \binom{2n-2}{n-1} \frac{(-4x)^n}{n}$$

$$= \sum_{n=1}^{\infty} \binom{2n-2}{n-1} \frac{x^{n-1}}{n}$$

$$= \sum_{n=0}^{\infty} \binom{2n}{n} \frac{x^n}{n+1} = \sum_{n=0}^{\infty} C_n x^n.$$

The Catalan Numbers

This shows that the number of full binary trees with n internal vertices is given by

$$C_n = \frac{1}{n+1}\binom{2n}{n}, \qquad n \in \mathbb{N}.$$

These numbers are called the ***Catalan numbers***. They appear in many contexts, e.g., as

- The number of ways to parenthesize the product of $n+1$ numbers (cf. Example 8, page 456 of ***Rosen***),
- The number of non-crossing handshakes of $2n$ people seated across a circular table,
- The number of triangulations of a polygon with $n+2$ sides,
- The number of rooted trees with n edges.

Final Exam

The preceding material completes the final third of the course material. It encompasses everything that will be the subject of the Final Exam.

The exact exam date will be announced on SAKAI.

No calculators or other aids will be permitted during the exam. A sample exam with solutions has been uploaded to SAKAI. Please study it carefully, including the instructions on the cover page.