Theoretische Informatik

Janis Hutz
https://janishutz.com

1. Oktober 2025

TITLE PAGE COMING SOON

"*A funny quote from the lecture still needed*"
- A professor in TI, 2025

HS2025, ETHZ

Summary of the book Theoretische Informatik

by Prof. Dr. Juraj Hromkovic

Inhaltsverzeichnis

Note: Definitions, Lemmas, etc are often 1:1 copies from the book or paraphrased (as I did not find an easier way of stating them)

1 Combinatorics

1.1 Introduction

Combinatorics was developed from the willingness of humans to gamble and the fact that everybody wanted to win as much money as possible.

1.2 Simple counting operations

The easiest way to find the best chance of winning is to write down all possible outcomes. This can be very tedious though when the list gets longer.

We can note this all down as a list or as a tree diagram. So-called Venn Diagrams might also help represent the relationship between two sets or events. Essentially a Venn Diagram is a graphical representation of set operations such as $A \cup B$.

1.3 Basic rules of counting

1.3.1 Multiplication rule

If one has n possibilities for a first choice and m possibilities for a second choice, then there are a total of $n \cdot m$ possible combinations.

When we think about a task, and we have an **and** in between e.g. properties, we need to multiply all the options.

1.3.2 Addition rule

If two events are mutually exclusive, the first has n possibilities and the second one has m possibilities, then both events together have $n + m$ possibilities.

When we think about a task, and we have an **or** in between e.g. properties, then we need to add all the options.

1.4 Factorial

Factorial **Definition 1.1**

The factorial stands for the product of the first n natural numbers where $n \geq 1$. Notation: !

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Additionally, $0! = 1$. We read $n!$ as "*n factorial*"

1.4.1 Operations

We can rewrite $n!$ as $n \cdot (n-1)!$ or $n \cdot (n-1) \cdot (n-2)!$ and so on.

It is also possible to write $7 \cdot 6 \cdot 5$ with factorial notation: $\frac{7!}{4!}$, or in other words, for any excerpt of a factorial sequence:

$$n \cdot (n-1) \cdot \ldots \cdot m = \frac{n!}{(m-1)!}$$

1.5 Permutations

Permutations **Definition 1.2**

A permutation of a group is any possible arrangement of the group's elements in a particular order

Permutation rule without repetition: The number of n ***distinguishable*** elements is defined as: $n!$

1.5.1 Permutation with repetition

For n elements $n_1, n_2, \ldots, n_k$ of which some are identical, the number of permutations can be calculated as follows:

$$p = \frac{n!}{n_1! \cdot n_2! \cdot \ldots \cdot n_k!}$$

where n_k is the number of times a certain element occurs. As a matter of fact, this rule also applies to permutations without repetition, as each element occurs only once, which means the denominator is 1, hence $\frac{n!}{(1!)^n} = n!$

Beispiel 1.1: CANADA has 6 letters, of which 3 letters are the same. So the word consists of 3 A's, which can be arranged in 3! different ways, a C, N and D, which can be arranged in 1! ways each. Therefore, we have:

$$\frac{6!}{3! \cdot 1! \cdot 1! \cdot 1!} = \frac{6!}{3!} = 6 \cdot 5 \cdot 4 = 120$$

Since 1! equals 1, we can always ignore all elements that occur only once, as they won't influence the final result.

1.6 Variations

Variations **Definition 1.3**

A ***variation*** is a selection of k elements from a universal set that consists of n *distinguishable* elements.

Variation rule without repetition: The ${}_n\mathrm{P}_k$ function is used to ***place*** n elements on k places. In a more mathematical definition: The number of different variations consisting of k different elements selected from n distinguishable elements can be calculated as follows:

$$\frac{n!}{(n-k)!} = {}_n\,\mathrm{P}_k$$

1.6.1 Variations with repetition

If an element can be selected more than once and the order matters, the number of different variations consisting of k elements selected from n distinguishable elements can be calculated using n^k

1.7 Combinations

Combination **Definition 1.4**

A combination is a selection of k elements from n elements in total without any regard to order or arrangement.
Combination rule without repetition:

$${}_n\mathrm{C}_k = \binom{n}{k} = \frac{{}_n\mathrm{P}_k}{k!} = \frac{n!}{(n-k)! \cdot k!}$$

1.7.1 Combination with repetition

In general the question to ask for combinations is, in how many ways can I distribute k objects among n elements?

$${}_{n+k-1}\mathrm{C}_k = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!(n-1)!}$$

1.8 Binomial Expansion

Binomial expansion is usually quite hard, but it can be much easier than it first seems. The first term of the expression of $(a+b)^n$ is always $1a^nb^0$. Using the formula for combination without repetition, we can find the coefficients of each element:

$$\text{6th row} \begin{cases} {}_6C_0 & {}_6C_1 & {}_6C_2 & {}_6C_3 & {}_6C_4 & {}_6C_5 & {}_6C_6 \\ 1 & \frac{6}{1} & \frac{6\cdot5}{1\cdot2} & \frac{6\cdot5\cdot4}{1\cdot2\cdot3} & \frac{6\cdot5\cdot4\cdot3}{1\cdot2\cdot3\cdot4} & \frac{6\cdot5\cdot4\cdot3\cdot2}{1\cdot2\cdot3\cdot4\cdot5} & \frac{6!}{6!} \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{cases}$$

This theory is based on the Pascal's Triangle and the numbers of row n correspond to the coefficients of each element of the expanded term.

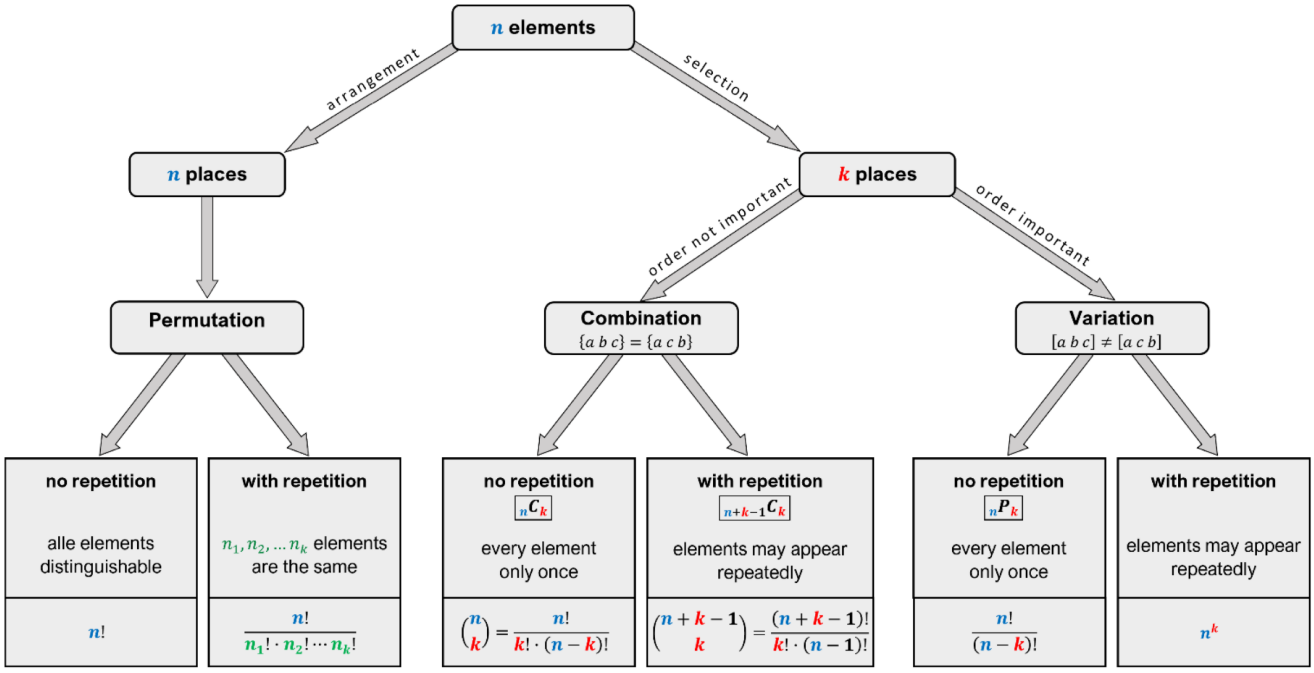
We can calculate the coefficient of each part of the expanded term k with combinatorics as follows: $\binom{n}{k}$

Binomial Expansion **Formel 1.1**

In general:

$$(a+b)^n = 1a^nb^0 + \binom{n}{1}a^{n-1}b^1 + \binom{n}{2}a^{n-2}b^2 + \ldots + \binom{n}{n-1}a^1b^{n-1} + \binom{n}{n}a^0b^n$$

1.9 Overview


n elements
arrangement
selection
n places
k places
order not important
order important
Permutation
Combination
$\{a\ b\ c\} = \{a\ c\ b\}$
Variation
$[a\ b\ c] \neq [a\ c\ b]$
no repetition
alle elements distinguishable
$n!$
with repetition
$n_1, n_2, \ldots n_k$ elements are the same
$\frac{n!}{n_1! \cdot n_2! \cdots n_k!}$
no repetition
${}_nC_k$
every element only once
$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$
with repetition
${}_{n+k-1}C_k$
elements may appear repeatedly
$\binom{n+k-1}{k} = \frac{(n+k-1)!}{k! \cdot (n-1)!}$
no repetition
${}_nP_k$
every element only once
$\frac{n!}{(n-k)!}$
with repetition
elements may appear repeatedly
n^k

2 Alphabete, Wörter, Sprachen und Darstellung von Problemen

2.1 Alphabete, Wörter, Sprachen

Alphabet **Definition 2.1**

Eine endliche, nicht leere Menge Σ. Elemente sind Buchstaben (Zeichen & Symbole).
Beispiele: Σ_{bool}, Σ_{lat} latin characters, Σ_{Tastatur}, Σ_m m-adische Zahlen (m-ary numbers, zero index)

Wort **Definition 2.2**

Über Σ eine (möglicherweise leere) Folge von Buchstaben aus Σ. Leeres Wort λ (ab und zu ε) hat keine Buchstaben.
$|w|$ ist die Länge des Wortes (Anzahl Buchstaben im Wort), während Σ^* die Menge aller Wörter über Σ ist und $\Sigma^+ = \Sigma^* - \{\lambda\}$
In diesem Kurs werden Wörter ohne Komma geschrieben, also $x_1x_2\dots x_n$ statt $x_1, x_2, \dots, x_n$. Für das Leersymbol gilt $|\text{␣}|$, also ist es nicht dasselbe wie λ

Für viele der Berechnungen in Verbindung mit der Länge der Wörter kann Kombinatorik nützlich werden. In Kapitel 1 findet sich eine Zusammenfassung über jenes Thema (in English)

Ein mögliches Alphabet beispielsweise um einen Graphen darzustellen ist folgendes:

Angenommen, wir speichern den Graphen als Adjezenzmatrix ab, dann können wir beispielsweise mit dem Alphabet $\Sigma = \{0, 1, \#\}$ diese Matrix darstellen, in dem wir jede neue Linie mit einem # abgrenzen. Das Problem hierbei ist jedoch, dass dies nicht so effizient ist, besonders nicht, wenn der Graph sparse ist, da wir dann viele # im Vergleich zu nützlicher Information haben.

Konkatenation **Definition 2.3**

$\Sigma^* \times \Sigma^* \to \Sigma^*$, so dass $\text{Kon}(x, y) = x \cdot y = xy \ \ \forall x, y \in \Sigma^*$.
Intuitiv ist dies genau das was man denkt: Wörter zusammenhängen (wie in Programmiersprachen). Die Operation ist assoziativ und hat das Neutralelement λ, was heisst, dass (Σ^*, Kon) ein Monoid ist.
Offensichtlich ist die Konkatenation nur für ein-elementige Alphabete kommutativ.
Die Notation $(abc)^n$ wird für die n-fache Konkatenation von abc verwendet

Umkehrung **Definition 2.4**

Sei $a = a_1a_2\dots a_n$, wobei $a_i \in \Sigma$ für $i \in \{1, 2, \dots, n\}$, dann ist die Umkehrung von a, $a^{\text{R}} = a_na_{n-1}\dots a_1$

Iteration **Definition 2.5**

Die i-te Iteration x^i von $x \in \Sigma^*$ für alle $i \in \mathbb{N}$ ist definiert als $x^0 = \lambda$, $x^1 = x$ und $x^i = xx^{i-1}$

Teilwort, Präfix, Suffix **Definition 2.6**

Seien $v, w \in \Sigma^*$
- v heisst ***Präfix*** von $w \iff \exists y \in \Sigma^* : w = vy$
- v heisst ***Suffix*** von $w \iff \exists x \in \Sigma^* : w = xv$
- v heisst ***Teilwort*** von $w \iff \exists x, y \in \Sigma^* : w = xvy$
- $v \neq \lambda$ heisst ***echtes*** Teilwort (gilt auch für Präfix, Suffix) von w genau dann, wenn $v \neq w$ und v ein Teilwort (oder eben Präfix oder Suffix) von w ist

Kardinalität, Vorkommen und Potenzmenge — Definition 2.7

Für Wort $x \in \Sigma^*$ und Buchstabe $a \in \Sigma$ ist $|x|_a$ definiert als die Anzahl Male, die a in x vorkommt.
Für jede Menge A ist $|A|$ die Kardinalität und $\mathcal{P}(A) = \{S | S \subseteq A\}$ die Potenzmenge von A

Kanonische Ordnung — Definition 2.8

Wir definieren eine Ordnung $s_1 < \ldots < s_m$ auf Σ. Die ***kanonische Ordnung*** auf Σ^* für $u, v \in \Sigma^*$ ist definiert als:

$$u < v \iff |u| < |v| \vee (|u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge v = x \cdot s_j \cdot v') \text{ für beliebige } x, u', v' \in \Sigma^* \text{ und } i < j$$

Oder in Worten, geordnet nach Länge und dann danach für den ersten nicht gemeinsamen Buchstaben, nach dessen Ordnung.

Sprache — Definition 2.9

$L \subseteq \Sigma^*$ ist eine Sprache, deren Komplement $L^C = \Sigma^* - L$ ist. Dabei ist $L_\emptyset$ die ***leere Sprache*** und L_λ die einelementige Sprache die nur aus dem leeren Wort besteht.
Die ***Konkatenation*** von L_1 und L_2 ist $L_1 \cdot L_2 = L_1 L_2 = \{vw \mid v \in L_1 \wedge w \in L_2\}$ und $L^0 := L_\lambda$ und $L^{i+1} = L^i \cdot L \;\; \forall i \in \mathbb{N}$ und $L^* = \bigcup_{i \in \mathbb{N}} L^i$ ist der ***Kleene'sche Stern*** von L, wobei $L^+ = \bigcup_{i \in \mathbb{N} - \{0\}} L^i = L \cdot L^*$

Für jede Sprache L gilt $L^2 \subseteq L \implies L = \emptyset \vee L = \{\lambda\} \vee L$ ist undendlich. Diese Aussage muss jedoch an der Prüfung bewiesen werden (nicht im Buch vorhanden)

Da Sprachen Mengen sind, gelten auch die Üblichen Operationen, wie Vereinigung ($\cup$) und Schnitt ($\cap$). Die Gleichheit von zwei Sprachen bestimmen wir weiter mit $A \subseteq B \wedge B \subseteq A \Rightarrow A = B$. Um $A \subseteq B$ zu zeigen reicht es hier zu zeigen dass für jedes $x \in A$, $x \in B$ hält. Wir betrachten nun, wie die üblichen Operationen mit der neu hinzugefügten Konkatenation interagieren.

Distributivität von Kon und $\cup$ — Lemma 2.1

Für Sprachen L_1, L_2 und L_3 über Σ gilt: $L_1 L_2 \cup L_1 L_3 = L_1 (L_2 \cup L_3)$

Der Beweis hierfür läuft über die oben erwähnte "Regel" zur Gleichheit. Um das Ganze einfacher zu machen, teilen wir auf: Wir zeigen also erst $L_1 L_2 \subseteq L_1 (L_2 \cup L_3)$ und dann equivalent für $L_1 L_3$.

Distributivität von Kon und $\cap$ — Lemma 2.2

Für Sprachen L_1, L_2 und L_3 über Σ gilt: $L_1 (L_2 \cap L_3) \subseteq L_1 L_2 \cap L_1 L_3$

L 2.3: Es existieren $U_1, U_2, U_3 \in (\Sigma_{\text{bool}})^*$, so dass $U_1 (U_2 \cap U_3) \subsetneq U_1 U_2 \cap U_1 U_3$

Homomorphismus — Definition 2.10

Σ_1, Σ_2 beliebige Alphabete. Ein ***Homomorphismus*** von Σ_1^* nach Σ_2^* ist jede Funktion $h : \Sigma_1^* \to \Sigma_2^*$ mit:
(i) $h(\lambda) = \lambda$
(ii) $h(uv) = h(u) \cdot h(v) \;\; \forall u, v \in \Sigma_1^*$

Erneut gilt hier, dass im Vergleich zu allgemeinen Homomorphismen, es zur Definition von einem Homomorphismus ausreichtt, $h(a)$ für alle Buchstaben $a \in \Sigma_1$ festzulegen.

2.2 Algorithmische Probleme

Ein Algorithmus $A : \Sigma_1^* \to \Sigma_2^*$ ist eine Teilmenge aller Programme, wobei ein Program ein Algorithmus ist, sofern es für jede zulässige Eingabe eine Ausgabe liefert, es darf also nicht eine endlosschleife enthalten.

Entscheidungsproblem **Definition 2.11**

Das ***Entscheidungsproblem*** (Σ, L) ist für jedes $x \in \Sigma^*$ zu entscheiden, ob $x \in L$ oder $x \notin L$. Ein Algorithmus A löst (Σ, L) (erkennt L) falls für alle $x \in \Sigma^*$: $A(x) = \begin{cases} 1, & \text{falls } x \in L \\ 0, & \text{falls } x \notin L \end{cases}$.

Funktion **Definition 2.12**

Algorithmus A berechnet (realisiert) eine ***Funktion (Transformation)*** $f : \Sigma^* \to \Gamma^*$ falls $A(x) = f(x)$ $\forall x \in \Sigma^*$ für Alphabete Σ und Γ

Berechnung **Definition 2.13**

Sei $R \subseteq \Sigma^* \times \Gamma^*$ eine Relation in den Alphabeten Σ und Γ. Ein Algorithmus A ***berechnet*** R (***löst das Relationsproblem*** R) falls für jedes $x \in \Sigma^*$, für das ein $y \in \Gamma^*$ mit $(x, y) \in R$ existiert gilt: $(x, A(x)) \in R$

2.3 Kolmogorov-Komplexität

Falls ein Wort x eine kürzere Darstellung hat, wird es ***komprimierbar genannt*** und wir nennen die Erzeugung dieser Darstellung eine ***Komprimierung*** von x.

Eine mögliche Idee, um den Informationsgehalt eines Wortes zu bestimmen, wäre einem komprimierbaren Wort einen kleinen Informationsgehalt zuzuordnen und einem unkomprimierbaren Wort einen grossen Informationsgehalt zuzuordnen.

Wenn wir also das Wort 011011011011011011011011 haben, so kann man es auch als $(011)^8$ darstellen und hat so also einen kleineren Informationsgehalt als bspw. 010110100010110100111011001O.

Die Idee mit der Komprimierung den Informationsgehalt zu bestimmen ist jedoch nicht ideal, da für jede Komprimierung bei unendlich langen Wörtern immer eine weitere Komprimierung existiert, die für unendlich viele Wörter besser geeignet ist.

Hier kommt die Kolmogorov-Komplexit zum Zuge: Sie bietet eine breit Gültige Definition des Komplexitätsmasses.

Kolmogorov-Komplexität **Definition 2.17**

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ ist die ***Kolmogorov-Komplexität*** $K(x)$ ***des Wortes*** x das Minimum der binären Längen der Pascal-Programme, die x generieren.

Hierbei ist mit der binären Länge die Anzahl Bits gemeint, die beim Übersetzen des Programms in einen vordefinierten Maschinencode entsteht.

Ein Pascal-Programm in diesem Kurs ist zudem nicht zwingend ein Programm in der effektiven Programmiersprache Pascal, sondern eine Abwandlung davon, worin es auch erlaubt ist, gewisse Prozesse zu beschreiben und nicht als Code auszuformulieren, da das nicht das Ziel dieses Kurses ist.

Kolmogorov-Komplexität **Lemma 2.4**

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ existiert eine Konstante d so dass $K(x) \leq |x| + d$

Beweis: Für jedes $x \in (\Sigma_{\text{bool}})^*$ kann folgendes Programm A_x verwendet werden:

```
$A_x$: begin
         write(x);
       end
```

Alle Teile, ausser x sind dabei von konstanter Länge, also ist die Länge der Bit-repräsentation des Programms ausschliesslich von der binären Länge des Wortes x abhängig.

□

Für regelmässige Wörter gibt es natürlich Programme, bei denen das Wort nicht als komplette Variable vorkommt. Deshalb haben diese Wörter auch (meist) eine kleinere Kolmogorov-Komplexität.

Definition 2.18: *($K(n)$ für $n \in \mathbb{N}$)* Die ***Kolmogorov-Komplexität einer natürlichen Zahl*** n ist $K(n) = K(\text{Bin}(n))$, wobei $|\text{Bin}(x)| = \lceil \log_2(x+1) \rceil$

Lemma 2.5: Für jede Zahl $n \in \mathbb{N} - \{0\}$ existiert ein Wort $w_n \in (\Sigma_{\text{bool}})^n$ so dass $K(w_n) \geq |w_n| = n$, oder in Worten, es existiert für jedes n ein nicht komprimierbares Wort.

Eine wichtige Eigenschaft der Kolmogorov-Komplexität ist, dass sie nicht wirklich von der gewählten Programmiersprache abhängt. Man kann also beliebig auch `C++`, `Swift`, `Python`, `Java` oder welche auch immer, ohne dass die Kolmogorov-Komplexität um mehr als eine Konstante wächst (auch wenn diese bei `Java` sehr gross ist):

Unterschiedliche Programmiersprachen — **Satz 2.1**

Für jede Programmiersprachen A und B existiert eine Konstante $c_{A,B}$, die nur von A und B abhängig ist, so dass für alle $x \in (\Sigma_{\text{bool}})^*$ gilt:

$$|K_A(x) - K_B(x)| \leq c_{A,B}$$

Anwendungen der Kolmogorov-Komplexität

Zufall Der Zufall ist ein intuitiver, aber nicht sehr formeller Begriff, der mit der Kolmogorov-Komplexität formalisiert werden kann:

Zufall — **Definition 2.19**

Ein Wort $x \in (\Sigma_{\text{bool}})^*$ (eine Zahl n) heisst ***zufällig***, falls $K(x) \geq |x|$ ($K(n) = K(\text{Bin}(n)) \geq \lceil \log_2(n+1) \rceil - 1$)

Existenz eines Programms vs Kolmogorov-Komplexität

Programm vs Komplexität — **Satz 2.2**

Sei L eine Sprache über Σ_{bool} und für jedes $n \in \mathbb{N} - \{0\}$ sei z_n das n-te Wort in L bezüglich der kanonischen Ordnung. Falls ein Programm A_L existiert, das das Entscheidungsproblem $(\Sigma_{\text{bool}}, L)$ löst, so gilt für alle $n \in \mathbb{N} - \{0\}$ dass

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c \qquad (c \text{ ist eine von } n \text{ unabhängige Konstante })$$

Primality testing

Primzahlensatz — **Satz 2.3**

$$\lim_{n \to \infty} \frac{\text{Prim}(n)}{\frac{n}{\ln(n)}} = 1$$

Die Annäherung von $\text{Prim}(n)$ and $\frac{n}{\ln(n)}$ wird durch folgende Ungleichung gezeigt:

$$\ln(n) - \frac{3}{2} < \frac{n}{\text{Prim}(n)} < \ln(n) - \frac{1}{2} \quad \forall n \geq 67 \in \mathbb{N}$$

Anzahl Primzahlen mit Eigenschaften — **Lemma 2.6**

Sei $n_1, n_2, \ldots$ eine stetig steigende unendliche Folge natürlicher Zahlen mit $K(n_i) \geq \frac{\lceil \log_2(n_i) \rceil}{2}$. Für jedes $i \in \mathbb{N} - \{0\}$ sei q_i die grösste Primzahl, die n_i teilt. Dann ist die Menge $Q = \{q_i \mid i \in \mathbb{N} - \{0\}\}$

Lemma 2.6 zeigt nicht nur, dass es unendlich viele Primzahlen geben muss, sondern sogar, dass die Menge der grössten Primzahlfaktoren einer beliebigen unendlichen Folge natürlicher Zahlen mit nichttrivialer Kolmogorov-Komplexität unendlich ist.

Untere Schranke für Anzahl Primzahlen — **Satz 2.4**

Für unendlich viele $k \in \mathbb{N}$ gilt

$$\text{Prim}(k) \geq \frac{k}{2^17 \log_2(k) \cdot (\log_2(\log_2(k)))^2}$$

Der Beweis hierfür ist sehr ausführlich ab Seite 42 im Buch erklärt