5 Komplexitätstheorie

Bisher haben wir Probleme mit Turingmaschinen (bzw., was ja wegen der Church-Turing-These das gleiche ist, mit einem Algorithmus) gelöst. Dabei ging es bei der Entscheidbarkeit bzw. Unentscheidbarkeit eines Problems zunächst darum, ob ein Problem *prinzipiell* gelöst werden kann oder nicht. Für die Praxis reicht diese Unterscheidung aber in den allermeisten Fällen nicht. Wenn ein Problem zwar lösbar ist, die Zeit zur Lösung des Problems aber inakzeptabel ist, dann ist das Problem *praktisch nicht lösbar*. Hat man bspw. einen Algorithmus für ein Graphenproblem, der 2^n viele Schritte zur Lösung des Problems braucht, wobei n die Anzahl der Knoten des Graphen ist. Dann ist dies für kleine Werte von n wie $n = 20$ oder $n = 30$ vielleicht noch akzeptabel, für z.B. $n = 100$ wird die Berechnung einer Lösung aber nicht mehr möglich sein, weil diese schlicht zu lange dauert.

Entsprechend verhält es sich bei dem Erfüllbarkeitsproblem der Aussagenlogik, also bei dem Problem gegeben eine aussagenlogische Formel, ist diese erfüllbar oder nicht? Hat eine Formel n verschiedene Aussagensymbole, so hat die Wahrheitstafel, aus der man die Lösung ablesen kann, 2^n viele Zeilen. Die Berechnung all dieser Zeilen dauert für größere n schlicht zu lange.

In der Komplexitätstheorie geht es nun zunächst darum, formal ein Maß für die Zeitkomplexität und die Platzkomplexität einer Berechnung zu definieren. Im Anschluss wird es dann darum gehen, wie man *untere Schranken* für ein Problem definieren kann, d.h. wie man ausdrücken kann, dass ein Problem (wie z.B. das Erfüllbarkeitsproblem der Aussagenlogik) auf keinen Fall schneller als eine bestimmte untere Schranke gelöst werden kann. Wir werden sehen, dass eine solche Schranke nicht einfach zu bestimmen ist. Tatsächlich stellt dies die Informatiker vor sehr große Probleme und wir können bisher nur Aussagen treffen, dass es mit einer hohen Wahrscheinlichkeit wohl keine Möglichkeit gibt, ein Problem schnell zu lösen. Aber Techniken, um dies mit Sicherheit zeigen zu können, sind im Moment außerhalb unserer Möglichkeiten. Diese Überlegungen werden in das Themengebiet P, NP und NP-Vollständigkeit münden.

5.1 Zeit- und Platzkomplexität

Wir werden in diesem Abschnitt die Zeit- und Platzkomplexität einer Rechnung einer Turingmaschine einführen. Um dies formal machen zu können, benötigen wir zunächst den Begriff der *Größe einer Probleminstanz*, denn die Zeit, die für

eine Berechnung benötigt wird, darf mit Sicherheit von der Größe des Problems abhängen. Um z.B. 10 Zahlen zu sortieren, wird man i.A. nicht so lange brauchen wie für das Problem, eine Million oder eine Milliarde Zahlen zu sortieren.

Für eine Turingmaschine ist eine *Probleminstanz* einfach eine Eingabe w und die *Größe der Probleminstanz* ist dann die Länge von w, also $|w|$. Will man z.B. Zahlen addieren, so ist dies als Sprache $L = \{\langle x, y, z\rangle \mid x + y = z\}$. Eine Probleminstanz ist dann ein Tripel $(2, 3, 5)$ ("Ja"-Instanz, da eine Turingmaschine für L dieses Tripel akzeptieren würde) oder $(2, 3, 6)$ ("Nein"-Instanz, da eine Turingmaschine für L dieses Tripel ablehnen würde). Die Größe einer Probleminstanz ist nun die Länge der Eingabe. Für eine Turingmaschine müsste man die Zahlen noch (bspw. als Binärzahlen) kodieren ebenso wie das Tupel. Die Länge der Eingabe wächst dann entsprechend.

Wir definieren nun den Zeit- und Platzbedarf, den eine Turingmaschine bei einer Rechnung auf einer Eingabe hat und damit dann Zeit- und Platzbeschränktheit.

Definition 5.1.1 (Zeit- und Platzkomplexität einer deterministischen Turingmaschine)**.** *Sei M eine deterministische Turingmaschine. Sei w ein Eingabewort.*

1. *Für eine Rechnung auf w benötigt M so viel Zeit, wie in der Rechnung einzelne Konfigurationen durchlaufen werden.*
2. *M benötigt in einer Rechnung auf w soviel Platz, wie* verschiedene *Felder besucht werden.*

Seien nun t, s Funktionen von $\mathbb{N}$ nach $\mathbb{R}$. Eine DTM M ist

1. *$t(n)$-**zeitbeschränkt** genau dann, wenn*

$$t(n) \geq \max\{k \mid \exists w \in L(M) : n = |w| \text{ und } M \text{ akzeptiert in } k \text{ Schritten}\}$$

2. *$s(n)$-**platzbeschränkt** genau dann, wenn*

$$s(n) \geq \max\{k \mid \exists w \in L(M) : n = |w| \text{ und } M \text{ akzeptiert mit Platzbedarf } k\}$$

Man beachte, dass alle Eingaben der Länge n betrachtet werden und dann $t(n)$ mindestens so groß sein muss, wie der Zeitbedarf im schlimmsten Fall (analog für den Platzbedarf und $s(n)$). Man bezeichnet die oben definierten Komplexitätsmaße daher als **worst-case**-Komplexitäten.

Noch zwei Anmerkungen zur Definition. Die Funktionen t und s sind nur deswegen Abbildungen auf die *reelle Zahlen*, damit wir später mit Funktionen

wie $n^2 + n/2$ arbeiten können, die ja nicht bei jedem $n \in \mathbb{N}$ eine natürliche Zahl liefern. Ferner werden oft Anfangswerte ignoriert und man erlaubt auch z.B. $t(n) = n^2 - 5 \cdot n$, was erst ab $n \geq 4$ einen sinnvollen Wert ergibt. Außerdem rundet man $t(n)$ und $s(n)$ ab, um auch Brüche u.ä. zu erlauben.

Hat eine deterministische Turingmaschine bspw. die Rechnung

$$z_0ab \vdash Az_1 \vdash ABz_2\# \vdash ABXz3\# \vdash ABz_3X$$

so werden für diese Rechnung 5 Zeiteinheiten benötigt und 4 Felder benutzt (das vierte Feld wird nicht geändert und bleibt leer, es wurde aber besucht und zählt daher).

Ist eine DTM $t(n)$-zeitbeschränkt mit $t(n) = n^2 - n/2$, dann darf eine Rechnung auf einem Wort w der Länge 10 nur maximal $100 - 5 = 95$ Schritte benötigen und auf einem Wort der Länge 3 nur maximal $9 - 1,5 = 7,5 \approx 7$ Schritte. Für die Zeitbeschränkung wird dann eine möglichst genaue obere Schranke t gesucht.

Der Grund warum Felder sofort bei Besuch zum Platzbedarf zählen und nicht erst, wenn sie beschrieben werden, ist, da man ansonsten mit den leeren Feldern z.B. unnär zählen könnte. So könnte man große Zahlen mit wenig Speicherbedarf (nämlich nur für die Begrenzer) kodieren und hätte später irreführende Aussagen zur Komplexität.

Wir definieren nun ganz entsprechend die Zeit- und Platzkomplexität einer *nichtdeterministischen* Turingmaschine. Hier muss aber eine Erfolgsrechnung festgehalten werden, da eine NTM auf einem Eingabewort ja (nichtdeterministisch) mehrere verschiedene Rechnungen haben kann.

Definition 5.1.2 (Zeit- und Platzkomplexität einer nichtdeterministischen Turingmaschine)**.** *Sei M eine* nicht*deterministische Turingmaschine. Sei w ein Eingabewort. In einer* fest vorgegebenen Erfolgsrechnung *auf w*

1. *benötigt eine NTM so viel Zeit, wie in der Rechnung einzelne Konfigurationen durchlaufen werden und*

2. *soviel Platz, wie* verschiedene *Felder besucht werden.*

Eine NTM M akzeptiert ein $w \in L(M)$ mit

1. *der **Zeitbeschränkung** $t \in \mathbb{R}$ genau dann, wenn die kürzestes Erfolgsrechnung $\lceil t \rceil$ Schritte hat*

2. *der **Platzbeschränkung** $s \in \mathbb{R}$ genau dann, wenn die Erfolgsrechnung mit dem geringsten Platzbedarf $\lceil s \rceil$ Felder besucht.*

Seien t, s Funktionen von $\mathbb{N}$ nach $\mathbb{R}$. Eine NTM M ist

1. $t(n)$-***zeitbeschränkt*** *genau dann, wenn*

$$t(n) \geq \max\{k \mid \exists w \in L(M) : n = |w| \text{ und } M \text{ akzeptiert mit Zeitbeschränkung } k\}$$

2. $s(n)$-***platzbeschränkt*** *genau dann, wenn*

$$s(n) \geq \max\{k \mid \exists w \in L(M) : n = |w| \text{ und } M \text{ akzeptiert mit Platzbeschränkung } k\}$$

Die Definitionen bei einer NTM sind sehr ähnlich zu denen einer DTM. Nur werden hier ausschließlich die Erfolgsrechnungen betrachtet und von diesen dann die beste genommen. Dann wird aber wieder die schlechteste hiervon bezüglich *verschiedenere* Wörter *gleicher* Länge genommen. Es ist also auch hier eine worst-case-Betrachtung. Der Nichtdeterminismus wird aber zusätzlich zu dem bisherigen nicht nur so betrachtet, dass so geraten wird, dass man zur positiven Antwort kommt, sofern dies möglich ist, sondern dass sogar so geraten wird, dass für die Rechnung der Zeit- und Platzbedarf möglichst gering ist.

Wir werden noch sehen, dass es nicht weiter schlimm ist, dass wir hier (sowohl bei DTMs als auch bei NTMs) vor allem auf Erfolgsrechnung schauen. In den uns interessierenden Fällen (nämlich den gleich folgenden Komplexitätsklassen P und NP) kann man die Schranken dann auch für die Nicht-Erfolgsrechnungen einführen. Damit werden wir dann haben:

- Eine TM ist $t(n)$-zeitbeschränkt heißt: Ein Wort der Länge n wird nach $t(n)$ Schritten akzeptiert oder abgelehnt.

- Eine TM ist $s(n)$-platzbeschränkt heißt: Eine Rechnung auf einem Wort der Länge n benutzt maximal $s(n)$ Bandzellen.

Komplexitätsklassen

Wir führen jetzt *Komplexitätsklassen* ein. Diese sind ähnlich wie REG, CF usw. Sprachfamilien. Mit ihnen werden aber Sprachen bzw. Probleme erfasst, die in einer bestimmten Zeit-/Platzschranke gelöst werden können. War also z.B. die Sprache a^*b^* in der Sprachfamilie REG, weil es einen endlichen Automaten gab, der sie akzeptiert, so werden wir sagen, dass eine Sprache dann in bspw. der Komplexitätsklasse P ist, wenn es eine DTM gibt, die die Sprache akzeptiert und $p(n)$-zeitbeschränkt ist, wobei p ein Polynom ist.

Definition 5.1.3 (Komplexitätsklasse)**.** *Für $s, t : \mathbb{N} \to \mathbb{R}$ mit $t(n) \geq n+1$ und $s(n) \geq 1$ definieren wir die Komplexitätsklassen*

$$\begin{aligned}
DTIME(t(n)) &:= \{L \mid L = L(A) \text{ für eine} \\
&\qquad t(n)\text{-zeitbeschränkte DTM } A\} \\
NTIME(t(n)) &:= \{L \mid L = L(A) \text{ für eine} \\
&\qquad t(n)\text{-zeitbeschränkte NTM } A\} \\
DSPACE(s(n)) &:= \{L \mid L = L(A) \text{ für eine} \\
&\qquad s(n)\text{-platzbeschränkte DTM } A\} \\
NSPACE(s(n)) &:= \{L \mid L = L(A) \text{ für eine} \\
&\qquad s(n)\text{-platzbeschränkte NTM } A\}
\end{aligned}$$

Während obige Komplexitätsklassen sehr allgemein sind ($DTIME(t(n))$ kann ja quasi für jedes t definiert werden), folgen nun die spezieller definierten Komplexitätsklassen P und NP sowie $PSPACE$ und $NPSPACE$. Diese Komplexitätsklassen werden uns vor allem interessieren (insb. P und NP) und diese spielen in der Informatik eine besondere, zentrale Rolle.

Definition 5.1.4 ($P, NP, PSPACE$ und $NPSPACE$)**.**

$$\begin{aligned}
P &:= \{L \mid \text{es gibt ein Polynom } p \text{ und eine} \\
&\qquad p(n)\text{-zeitbeschränkte DTM } A \text{ mit } L(A) = L\} \\
NP &:= \{L \mid \text{es gibt ein Polynom } p \text{ und eine} \\
&\qquad p(n)\text{-zeitbeschränkte NTM } A \text{ mit } L(A) = L\} \\
PSPACE &:= \{L \mid \text{es gibt ein Polynom } p \text{ und eine} \\
&\qquad p(n)\text{-platzbeschränkte DTM } A \text{ mit } L(A) = L\} \\
NPSPACE &:= \{L \mid \text{es gibt ein Polynom } p \text{ und eine} \\
&\qquad p(n)\text{-platzbeschränkte NTM } A \text{ mit } L(A) = L\}
\end{aligned}$$

Wir betonen nach diesen Definitionen noch einmal, dass in einer Komplexitätsklasse Probleme zusammengefasst werden, die in einer bestimmten Zeit- oder Platzschranke gelöst werden können. Ist ein Problem in P heißt das, es gibt ein Polynom p, so dass eine DTM existiert, die das Problem mit Zeitschranke p löst (also bei einer Eingabe der Länge n maximal $p(n)$ Schritte benötigt). Analog für die Klasse NP mit einer NTM statt einer DTM. Diese Klassen sind nachfolgend für uns besonders wichtig.

Da eine deterministische Turingmaschine, stets wie eine nichtdeterministische Turingmaschine gesehen werden kann, die den Nichtdeterminismus nie ausnutzt, gilt der folgende Satz.

Satz 5.1.5. *Es gilt*

$$\begin{aligned} DTIME(f) &\subseteq NTIME(f) \\ DSPACE(f) &\subseteq NSPACE(f) \\ P &\subseteq NP \\ PSPACE &\subseteq NPSPACE \end{aligned}$$

Zudem muss auch jede Sprache, die man mit einer Zeitschranke von t lösen kann auch mit einer Platzschranke von t gelöst werden können, denn wenn man nur $t(n)$ viele Schritte macht, kann man auch nur $t(n)$ viele Bandzellen benutzen. Daher gilt $DTIME(f) \subseteq DSPACE(f)$, $NTIME(f) \subseteq NSPACE(f)$, $P \subseteq PSPACE$ und $NP \subseteq NPSPACE$.

Damit haben wir bereits die wichtigsten Komplexitätsklassen zusammen. Bis hierhin haben wir definiert wie viel Zeit und Platz eine Turingmaschine zur Berechnung einer Antwort braucht und darauf aufbauend dann Komplexitätsklassen eingeführt, in denen Probleme zusammengefasst sind, die in bestimmten Zeit- oder Platzschranken gelöst werden können. Insbesondere haben wir die wichtigen Komplexitätsklassen P und NP eingeführt.

Man kann auch Komplexitätsklassen untersuchen, die zugleich eine Zeit- und eine Platzschranke betrachten. Dies wollen wir hier aber nicht vertiefen. Zudem gibt es weitere wichtige Komplexitätsmaße. An Bedeutung gewinnen z.B. aufgrund der aktuellen Architekturen Untersuchungen, die die Anzahl der Prozessoren berücksichtigen. Dazu muss aber erst ein Modell mit mehreren Prozessoren eingeführt werden. Auch dies wollen wir hier nicht vertiefen. In weiterführenden Veranstaltungen und Literatur zu verteilten und parallelen Algorithmen trifft man auf diese.

Wir wollen abschließend noch auf drei Punkte eingehen. Erstens auf den Zusammenhang zwischen Turingmaschinen und realen Computern. Es ist eine ähnliche Aussage wie bei der Church-Turing-These auch für Komplexitätsmaße möglich und dies lässt uns verstehen, dass oben eingeführte Komplexitätsmaße nicht nur von theoretischem Interesse sind, sondern gerade ein Bereich sind, der stark in die Praxis wirkt. Zweitens wollen wir kurz die O-Notation (oder Landau-Notation) erwähnen, die uns einige Notationen erleichtern wird. Und drittens wollen wir noch auf Entscheidungs- und Optimierungsprobleme eingehen, da alle Komplexitätsklassen oben ja über Entscheidungsprobleme definiert sind, in der Praxis ja aber viel häufiger Optimierungsprobleme auftreten (so will man oft ja nicht nur wissen, ob ein kürzester Pfad zwischen zwei Punkten in einem Graphen existiert, sondern diesen berechnen).

Bis hierhin haben wir uns mit der Zeit- und Platzkomplexität von Turingmaschinen beschäftigt, die ja durchaus recht weit entfernt scheint von einer gängigen, physikalisch implementierten Computerarchitektur. Wir haben ja aber schon im Abschnitt zur Church-Turing-These gesehen, dass ein Computer eine Turingmaschine simulieren kann *und umgekehrt* auch eine Turingmaschine

einen Computer simulieren kann. Diese Erkenntnis hat zur Übertragung von Unentscheidbarkeitsresultaten von der Turingmaschine auf gängige Computer geführt. Betrachtet man die Konstruktionen, wie eine Turingmaschine einen Computer simuliert und umgekehrt, genauer, so stellt man fest, dass diese Konstruktionen lediglich polynomiellen Mehraufwand erfordern, d.h. wenn ein Computer polynomiell zeitbeschränkt ist, dann arbeitet auch die simulierende Turingmaschine in polynomieller Zeit! Ein Problem bzw. eine Sprache ist also in P unabhängig davon, ob wir einen Algorithmus, der in Polynomialzeit arbeitet, für eine Turingmaschine oder für ein gängiges Rechnermodell angeben. Dies ist einer der Gründe für die herausragende Bedeutung der Komplexitätsklasse P und der Grund, warum sie, ganz entsprechend zu der Art wie wir dies hier für Turingmaschinen gemacht haben, bisweilen, insb. in Algorithmenbüchern, auch direkt mit einer imperativen Programmiersprache als Berechnungsmodell eingeführt wird.

Die *erweiterte Church-Turing-These* sagt aus, dass diese Simulation in Polynomialzeit nicht nur für Turingmaschinen und gängige Rechnermodelle gilt, sondern für alle sinnvollen deterministischen Rechnermodelle. Dies ist so wichtig, dass wir es noch einmal betonen wollen.

> Church-Turing-These: Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.
>
> Folgerung: Alle hinreichend mächtigen Berechnungsmodelle sind äquivalent.
>
> Erweiterte Church-Turing-These: Sind B_1 und B_2 zwei vernünftige Rechnermodelle, so gibt es ein Polynom p, so dass t Rechenschritte von B_1 bei einer Eingabe der Länge n durch $p(t, n)$ Rechenschritte von B_2 simuliert werden können.
>
> Folgerung: Zwei (Turing-)äquivalente Modelle lassen sich mit polynomiellen Mehraufwand ineinander umwandeln.

Damit erhält die Komplexitätsklasse P eine besondere Bedeutung. Jede Sprache, die in P ist, kann mit jedem Berechnungsmodell in Polynomialzeit akzeptiert werden (auch wenn die Polynome unterschiedlich sind).

Außerdem kann man statt mit Turingmaschinen bei Algorithmen auch mit gängigen Programmiersprachen argumentieren, so fern einen nur interessiert, ob ein Problem in P ist (und nicht der exakte Zeitbedarf auf einem festen Rechnermodell).

Dies zusammen mit der Tatsache, dass man i.A. davon ausgeht, dass **in P die Probleme liegen, die wir effizient lösen können**, begründet die herausragende Bedeutung von P. Letzteres liegt daran, dass Polynome meist sehr

verträglich wachsen. So wächst n^2 im Vergleich zu 2^n für steigende n deutlich langsamer. Bspw. ist $50^2 = 2500$, aber 2^{50} bereits mehr als eine Billarde.

Aus der vorherigen Betrachtung wissen wir, wie wichtig die Komplexitätsklasse P und damit Polynome sind. In der Praxis sind die genauen konstanten Faktoren im Polynom aber oft nicht interessant. Ebenso sind oft die Summanden mit kleinerem Exponenten meist nicht von Interesse. So interessiert bei einem Polynom wie $n^3 + 2 \cdot n^2 - 5 \cdot n$ oft nur, dass es sich "grob so verhält wie n^3". Um dies zu formalisieren, wird die Landau-Notation oder O-Notation eingeführt, die insb. auch in der Algorithmik zum Vergleich verschiedener Algorithmen von großer Bedeutung ist. Wir wollen uns hier nicht näher mit der O-Notation beschäftigen und wollen uns nur folgendes merken:

Ist $f(n)$ eine Funktion (z.B. $f(n) = n^2$), dann bedeutet

Die Laufzeit des Algorithmus/der Turingmaschine ist in $O(f(n))$.

dass es eine Konstante c gibt, so dass bei einer Eingabe der Länge n die Turingmaschine/der Algorithmus maximal $c \cdot f(n)$ Schritte macht.

Dies ist deswegen praktisch, da man dann bei einem Algorithmus, der z.B. bei n Zahlen alle Zahlen paarweise miteinander vergleicht, sagen kann, dass er eine Laufzeit in $O(n^2)$ hat, obwohl neben den n^2 Vergleichen vielleicht noch Dinge passieren, wie eine Schleifenvariable zu erhöhen und ähnliches. So lange dies nur konstant viele Operationen sind, die pro Vergleich passieren, ist die Laufzeit in $c \cdot n^2$ für eine Konstante c und damit in $O(n^2)$.

Oft benutzt man dann in der Algorithmenanalyse auch nicht mehr die Eingabelänge n, sondern eine *Kenngröße* n, aus der sich die eigentliche Eingabelänge leicht ermitteln lässt und die aber i.A. aussagekräftiger ist. Hat man z.B. einen Algorithmus auf Graphen, so nimmt man als Kenngröße oft die Anzahl der Knoten n und die Anzahl der Kanten m und drückt die Laufzeit dann bzgl. dieser Kenngrößen aus.

Wir wollen zuletzt noch auf die Ähnlichkeiten zwischen Entscheidungs- und Optimierungsproblemen eingehen. Bei Turingmaschinen interessieren wir uns für die Akzeptanz bzw. das Entscheiden von Sprachen. Bei einer vorgelegten Eingabe, soll die Turingmaschine im Rahmen der Zeitschranke anhalten und positiv oder negativ antworten (in diesem Abschnitt haben wir die Zeitschranken zunächst nur für Erfolgsrechnungen eingeführt, wir werden aber im nächsten Abschnitt sehen, dass wir dies auch für unsere Zwecke auf alle Rechnungen erweitern können). Z.B. erhält man als Eingabe einen Graphen G, zwei Knoten s und t und eine Schranke k und die Frage ist, *ob* ein Pfad von s nach t existiert, der höchstens den Wert k hat. In der Praxis interessiert man sich aber meist mehr dafür, den Wert für k zu optimieren oder den Pfad, von s zu t zu bestimmen. Entscheidungs- und Optimierungsprobleme lassen sich aber meist gut ineinander umwandeln, so dass eine Lösung des einen auch zu einer Lösung des anderen führt. Dass man mit einer Lösung des Optimierungsproblems auch das Entscheidungsproblem lösen kann, liegt meist auf der Hand, da

man dann ja den besten Wert kennt. Andersherum funktioniert es meist, indem man z.B. den Wert k aus dem obigen Beispiel wie bei einer binären Suche immer weiter halbiert oder indem die Probleminstanz sukzessive geändert wird, hier z.B. indem einzelne Knoten entfernt werden und geprüft wird, ob ein Entscheidungsverfahren immer noch akzeptiert. Wir merken uns an dieser Stelle einfach, dass die Entscheidungs- und die Optimierungsvariante eines Problems so ähnlich sind, dass man, wenn man das eine schnell (also mit einem Polynomialzeitalgorithmus) lösen kann, auch das andere schnell (mit einem Polynomialzeitalgorithmus) lösen kann. Es genügt daher, sich auf die, für die Theorie angenehmeren, Entscheidungsvarianten zu konzentrieren und mit diesen eine für die Praxis relevante Theorie aufzubauen.

Fragen

1. Wenn sie für eine DTM eine Zeitschranke von $t(n) = n^2 + 2n$ haben, was gilt dann bei einer Eingabe w der Länge 5?
   a) Die Rechnung dauert mindestens 35 Schritte
   b) Die Rechnung dauert maximal 35 Schritte
   c) Die Rechnung dauert genau 35 Schritte
   d) w wird nach 35 Schritten akzeptiert
2. Das P in NP steht für Polynomialzeit. Wofür steht das N?
   a) nicht(-polynomiell)
   b) nicht-deterministisch
   c) nicht-(praktikabel)
3. Was gilt?
   a) $P \subseteq NP$
   b) $NP \subseteq P$
   c) $P \cap NP = \emptyset$
   d) unvergleichbar!