Prof. Javier Esparza
Philipp Czerner, Kevin Kappelmann, Valentin Krasotin
Technische Universität München
Lehrstuhl für Theoretische Informatik

Einführung in die Theoretische Informatik

Sommersemester 2024 – Übungsblatt 12

- Das Übungsblatt ist in zwei Teile gegliedert: den Vorbereitungsteil, den Sie vor der Übung selbstständig bearbeiten sollen, und den Übungs-/Nachbereitungsteil, der Aufgaben enthält, die in der Übung besprochen werden und von Ihnen anschließend zur Nachbereitung verwendet werden können.

Vorbereitung (vor der Übung selbständig zu bearbeiten)

Vorbereitungsaufgabe Ü12.1. (*Wichtige Begriffe*)

Überprüfen Sie, dass Sie die folgenden Begriffe oder Notationen korrekt definieren können.

- $time_M(w)$ und $ntime_M(w)$
- $TIME(f(n))$ und $NTIME(f(n))$
- Landau-Notation (oder auch $\mathcal{O}$-Notation): $\mathcal{O}(n), \mathcal{O}(2^n), \ldots$
- P und NP
- P $\stackrel{?}{=}$ NP
- Zertifikat und polynomiell beschränkter Verifikator
- NP-hart (oder auch auch: “NP-schwer”)
- NP-vollständig
- Zertifikat und polynomiell beschränkter Verifikator
- polynomielle Reduktion
- $A \leq_p B$ (sprich: *"A ist polynomiell reduzierbar auf B"*)
- Die Probleme: SAT und 3KNF-SAT

Übung und Nachbereitung

Übungsaufgabe Ü12.2. (*Komplexe Fragen*)

Entscheiden Sie jeweils, ob die folgenden Algorithmen korrekterweise das gegebene Problem in deterministisch, polynomieller Zeit entscheiden. Begründen Sie Ihre Antwort.

(a) Gegeben: eine aussagenlogische Formel F. Frage: Ist F erfüllbar?

Algorithmus: Konstruiere die Wahrheitstabelle für F und werte F für jede Belegung aus der Tabelle aus.

(b) Gegeben: Eine natürliche Zahl $n > 1$. Frage: Ist n prim?

Algorithmus: Überprüfe für alle Zahlen k zwischen 2 und $n-1$, ob k n teilt. Wenn keine dieser Zahlen n teilt, ist n prim.

Hinweis: Sie können annehmen, dass ein Teilbarkeitstest in polynomieller Zeit durchgeführt werden kann.

(c) Sei $k \in \mathbb{N}$ fixiert.

Gegeben: Eine endliche Menge $S \subset \mathbb{N}$ mit n Elementen und eine Zahl z. Frage: Gilt für alle $(n-k)$-elementigen Teilmengen K von S, dass $\sum_{s \in K} s > z$?

Algorithmus: Wiederhole, solange es noch neue $(n-k)$-elementige Teilmengen von S gibt: Berechne die nächste solche Teilmenge K, summiere ihre Elemente und überprüfe, ob diese Summe größer als z ist. Wenn nein, antworte nein. Wenn es keine weitere $(n-k)$-elementige Teilmengen von S gibt, antworte ja.

Hinweis: Sie dürfen annehmen, dass es einen Algorithmus gibt, welcher die nächste Teilmenge in polynomieller Zeit berechnet.

Lösungsskizze.

(a) Nicht polynomiell: die Wahrheitstabelle ist exponentiell groß in der Anzahl der vorkommenden Variablen.

(b) Hängt von der Kodierung von n ab. Wir müssen $n-2$ Teilbarkeitstests durchführen. Wenn n (wie üblich) binär codiert wird hat die Eingabe Länge $\log_2(n)$. Der Algorithmus ist daher exponentiell in der Länge der Eingabe. Wenn n unär codiert wird, ist die Eingabegröße auch n und der Algorithmus polynomiell.

(c) Wahr: Die nächste Teilmenge lässt sich in polynomieller Zeit berechnen (Hinweis) und ist linear groß in der Größe der Eingabe. Daher ist auch die Berechnung der Summe und der Vergleich zweier Zahlen in polynomieller Zeit machbar. Jede Wiederholung ist daher polynomiell. Wir müssen also nur noch zeigen, dass nur polynomiell viele Wiederholungen gebraucht werden, d.h. dass nur polynomiell viele $(n-k)$-elementige Teilmengen geprüft werden müssen.

Die Anzahl $(n-k)$-elementiger Teilmengen von S ist

$$\binom{n}{n-k} = \frac{n!}{k! \cdot (n-k)!} \leq \frac{n!}{(n-k)!} = n \cdot (n-1) \cdot \ldots \cdot (n-k+1) \leq n^k$$

Übungsaufgabe Ü12.3. (*Guess What*)

Entscheiden Sie jeweils, ob die folgenden Algorithmen korrekterweise das gegebene Problem in nichtdeterministisch, polynomieller Zeit semi-entscheiden. Falls ja, skizzieren Sie einen Beweis. Falls nein, beschreiben Sie den Fehler.

(a) Gegeben: Rechtslineare Grammatik G für eine reguläre Sprache. Frage: ist $L(G)$ nicht leer?

Der Algorithmus ist eine NTM, die eine Ableitung in der Grammatik G rät. Genauer: Die Maschine beginnt beim Startsymbol. In jedem Schritt wählt die Maschine dann nichtdeterministisch ein Nichtterminal in der aktuellen Symbolkette und ersetzt es nichtdeterministisch durch eine passende Produktion. Falls die Symbolkette nur noch aus Terminalen besteht, wurde ein Wort in $L(G)$ gefunden und die Maschine antwortet positiv.

(b) Gegeben: Grammatik G in Chomsky Normalform für eine kontextfreie Sprache. Frage: ist $L(G)$ nicht leer?

Der Algorithmus bleibt derselbe, wie in Aufgabe (a).

Lösungsskizze.

(a) Wahr. Da G rechtslinear ist, gibt es, falls $L(G)$ nicht leer ist, eine Ableitung eines Wortes in höchstens $|P|$ Schritten, wobei $|P|$ die Anzahl der Produktionen in G ist. Der Algorithmus ist somit sogar nichtdeterministisch linear.

(b) Falsch. Es gibt kontextfreie Grammatiken, deren kürzestes Wort exponentielle Länge besitzt. Der Algorithmus braucht in solchen Fällen exponentiell viele Schritte und ist somit nicht nichtdeterministisch polynomiell. Beispiel:

$$S \to B_n B_n, \quad B_{i+1} \to B_i B_i, \quad B_0 \to b, \quad \text{für } 0 \leq i < n.$$

Übungsaufgabe Ü12.4. (*SAT-Varianten*)

Wir betrachten verschiedene Varianten von SAT, die auch NP-vollständig sind.

Zeigen Sie die NP-Vollständigkeit der folgenden Probleme A_1, A_2, indem Sie jeweils $A_i \in \text{NP}$ zeigen und eine Reduktion $\mathsf{3KNF\text{-}SAT} \leq_p A_i$ angeben.

(a) 3-OCC-KNF-SAT:

- **Eingabe**: Eine Formel F in KNF, bei der jede Variable höchstens dreimal auftritt.
- **Frage**: Ist F erfüllbar?

(b) Wir betrachten den ITE-Operator mit der Semantik $\mathsf{ITE}(x, y, z) \equiv (x \to y) \wedge (\neg x \to z)$. Eine ITE-Formel genügt der folgenden Grammatik:

$$F \to \mathsf{ITE}(F, F, F) \mid x \mid \mathsf{true} \mid \mathsf{false} \qquad \text{für Variablen } x \in \mathcal{V}$$

ITE-SAT:

- **Eingabe**: Eine ITE-Formel F.
- **Frage**: Ist F erfüllbar?

Lösungsskizze.

(a)
- $\in$ **NP**: Eine erfüllende Belegung ist ein geeignetes Zertifikat, da sie maximal genauso groß ist, wie die Formel F und in polynomieller Zeit geprüft werden kann.
- **NP-schwer**: Sei F eine Formel in 3-KNF. Somit $F = \bigwedge_{i=1}^{n} K_i$ und $K_i = \bigvee_{j=1}^{m} L_{ij}$ mit $m \leq 3$. Ohne Beschränkung der Allgemeinheit kommen keine Variablen doppelt in einer Klausel vor. Andernfalls ist die Klausel entweder trivial wahr oder enthält Literale doppelt. Beides können wir in einem Preprocessing einfach entfernen.

  Sei v nun die Anzahl der in F verwendeten Variablen. Dann ersetzten wir in jeder Klausel die Variablen mit für diese Klausel spezifische Variablen und fügen eine Bedingung ein, die erzwingt, dass alle Kopien einer Variable in der Eingabe den gleichen Wert zugewiesen bekommen. Diese Bedingung hat eine Ringstruktur, um Variablenverwendungen zu sparen: $(x_{k,1} \to x_{k,2}) \wedge (x_{k,2} \to x_{k,3}) \wedge ...(x_{k,n} \to x_{k,1})$.

$$f(F) = \bigwedge_{i=1}^{n} f(K_i) \wedge \bigwedge_{k=1}^{v} \bigwedge_{i=1}^{n} (\neg x_{k,i} \vee x_{k,(i \bmod n)+1})$$

$$f(K_i) = \bigvee_{j=1}^{m} f(L_{ij})$$

$$f(L_{ij}) = \begin{cases} x_{k,i} & \text{falls } L_{ij} = x_k \\ \neg x_{k,i} & \text{falls } L_{ij} = \neg x_k \end{cases}$$

Diese Reduktion ist in polynomieller Zeit berechenbar, da nur die Eingabe einmal kopiert werden muss und ein zusätzlicher Term der Größe $n \cdot v$ erzeugt wird.

Korrektheit: Aus jeder erfüllenden Belegung σ für F können wir eine erfüllende Belegung σ' für $f(F)$ konstruieren mit $\sigma'(x_{k,i}) = \sigma(x_k)$. Auch können wir aus jeder erfüllenden Belegung σ für $f(F)$ ein Belegung für F konstruieren. Da $\sigma(x_{k,i}) = \sigma(x_{k,j})$ für alle i, j gelten muss, können wir einfach σ' über $\sigma'(x_k) = \sigma(x_{k,1})$ definieren. Somit ist die Konstruktion erfüllbarkeitserhaltend und die Reduktion korrekt.

(b)

- $\in$ **NP**: Eine erfüllende Belegung ist ein geeignetes Zertifikat, da sie maximal genauso groß ist, wie die Formel F, und in polynomieller Zeit geprüft werden kann.
- **NP-schwer**: Sei F eine Formel in 3KNF. Wir wandeln F in polynomieller Zeit unter der Verwendung der folgenden semantischen Äquivalenzen in eine ITE-Formel:

  $$\neg x \equiv \mathsf{ITE}(x, \mathsf{false}, \mathsf{true}) \qquad x \wedge y \equiv \mathsf{ITE}(x, y, \mathsf{false}) \qquad x \vee y \equiv \mathsf{ITE}(x, \mathsf{true}, y)$$

  Eine Klausel mit drei Literalen, z.B. $\neg x \vee \neg y \vee \neg z$ lässt sich damit umformen zu

  $$\begin{aligned} \neg x \vee \neg y \vee \neg z \equiv \mathsf{ITE}(&\\ &\mathsf{ITE}(x, \mathsf{false}, \mathsf{true}),\\ &\mathsf{true},\\ &\mathsf{ITE}(\\ &\quad \mathsf{ITE}(y, \mathsf{false}, \mathsf{true}),\\ &\quad \mathsf{true},\\ &\quad \mathsf{ITE}(z, \mathsf{false}, \mathsf{true})\\ &) \\ ) \end{aligned}$$

  Jede Klausel wird in eine ITE-Formel übersetzt, welche nur um einen konstanten Faktor größer ist. Entsprechend übersetzt sich jede 3KNF-Formel in eine semantisch äquivalente ITE-Formel, die nur um einen konstanten Faktor größer ist.

Die Korrektheit folgt aus der Verwendung semantisch-äquivalenten Umformungen.