Übungen zur Vorlesung Grundbegriffe der Theoretischen Informatik

technische universität dortmund

Thomas Schwentick

Gaetano Geck, Lutz Oettershagen,
Christopher Spinrath, Marco Wilhelm

Lehrstuhl 1
Logik in der Informatik

SoSe 2018 | **Übungsblatt 11** | **26.06.2018**

Aufgabe 11.1 [Optimierungsprobleme vs. Entscheidungsprobleme] **7 Punkte**

Sei $G = (V, E)$ ein ungerichteter Graph. Ein *Weg* in G ist eine durch Kanten verbundene Folge von Knoten aus G in der *kein Knoten mehrfach* vorkommt. Ein Weg w' ist eine *Fortsetzung* eines Weges w, falls $w = v_1 \dots v_\ell$ und $w' = v_1 \dots v_\ell v_{\ell+1} \dots v_k$. Wir betrachten die folgenden Problemvarianten:

Problem: k-Weg-Fortsetzung
Gegeben: ungerichteter Graph G, Weg $w = v_1 \dots v_\ell$ in G mit $\ell \leq k$
Frage: Lässt sich der Weg w zu einem Weg der Länge k fortsetzen?

Problem: Weg-Fortsetzung
Gegeben: ungerichteter Graph G, Zahl $k \in \mathbb{N}$, Weg $w = v_1 \dots v_\ell$ in G mit $\ell \leq k$
Frage: Lässt sich der Weg w zu einem Weg der Länge k fortsetzen?

Problem: Weg-FortsetzungW
Gegeben: ungerichteter Graph G, Weg w in G
Gesucht: Maximale Zahl $k \in \mathbb{N}$, für die sich der Weg w zu einem Weg der Länge k fortsetzen lässt.

Problem: Weg-FortsetzungO
Gegeben: ungerichteter Graph G, Weg w in G
Gesucht: Fortsetzung von w maximaler Länge.

a) Zeigen Sie, dass sich das Problem k-Weg-Fortsetzung für festes $k \in \mathbb{N}$ in polynomieller Zeit lösen lässt. Warum folgt daraus nicht, dass sich auch das Problem Weg-Fortsetzung in polynomieller Zeit lösen lässt? **(2 Punkte)**

b) Zeigen Sie: **(5 Punkte)**

(i) Lässt sich Weg-Fortsetzung in polynomieller Zeit lösen,
dann lässt sich auch Weg-FortsetzungW in polynomieller Zeit lösen. **[2 Punkte]**

(ii) Lässt sich Weg-FortsetzungW in polynomieller Zeit lösen,
dann lässt sich auch Weg-FortsetzungO in polynomieller Zeit lösen. **[3 Punkte]**

Lösung:

a) In dem Problem k-Weg-Fortsetzung ist k eine Konstante und kein Teil der Eingabe. Um nun zu zeigen, dass k-Weg-Fortsetzung in P liegt, genügt es daher einen Algorithmus A für k-Weg-Fortsetzung anzugeben, der nur polynomiell viele Schritte in der Anzahl der Knoten macht. Bezüglich der Konstante k muss die Laufzeit von A also nicht polynomiell beschränkt sein.

Sei der ungerichtete Graph $G = (V, E)$ eine Eingabe für k-Weg-Fortsetzung. Da eine Weg-Fortsetzung w' der Länge k von einem Weg w der Länge ℓ gesucht wird, überprüft A für jede Knotenfolge $v_{\ell+1}, \ldots, v_k$ der Länge $k - \ell$, ob $wv_{\ell+1}, \ldots, v_k$ eine Wegfortsetzung von w ist.

Algorithmus A

Eingabe: Ungerichteter Graph $G = (V, E)$, Weg $v_1 \ldots v_\ell$
Ausgabe: ja/nein

```
1: for all Folgen v_{ℓ+1}, ..., v_k von Knoten aus V do
2:    valid ← true
3:    for all i ∈ {1, ..., k − 1} und j ∈ {i + 1, ..., k} do
4:       if v_i = v_j oder (v_i, v_{i+1}) ∉ E then
5:          valid ← false
6:    if valid = true then
7:       return ja
8: return nein
```

In der äußeren Schleife werden $|V|^{k-\ell}$ viele Knotenfolgen aufgezählt. Da k konstant ist, sind das polynomiell viele Folgen, $\mathcal{O}(|V|^{k-\ell}) = \mathcal{O}(|V|^k)$.

Beim Test, ob eine Knotenfolge $v_1 \ldots v_k$ eine Weg-Fortsetzung eines Weges $v_1 \ldots v_\ell$ ist, werden $\mathcal{O}(k^2)$ Knotenpaare auf Ungleichheit überprüft. Außerdem wird für $\mathcal{O}(k)$ Knotenpaare überprüft, ob sie im Graphen durch eine Kante verbunden sind. Da k konstant ist, wird dafür nur konstante Zeit benötigt.

Insgesamt ergibt sich eine polynomielle Laufzeit $\mathcal{O}(|V|^k)$.

Bei dem Problem Weg-Fortsetzung ist k Teil der Eingabe. Somit muss die Laufzeit eines Algorithmus für Weg-Fortsetzung auch polynomiell in k beschränkt sein. Dies ist für die Aufzählung der $|V|^{k-\ell}$ vielen Knotenfolgen nicht der Fall.

b) **(i)** Sei B ein Algorithmus, der das Problem Weg-Fortsetzung in polynomieller Zeit löst. Da die Länge des Weges höchstens $|V| + 1$ sein kann, kann der Algorithmus B' für das Problem Weg-FortsetzungW die $|V| - \ell + 1$ vielen Werte für k mit Hilfe des Algorithmus B überprüfen und den maximalen Wert ausgeben:

Algorithmus B'

Eingabe: ungerichteter Graph $G = (V, E)$, Weg $w = v_1 \ldots v_\ell$
Ausgabe: maximale Länge einer Weg-Fortsetzung von w

```
1: k ← ℓ
2: for all i in ℓ + 1, ..., |V| do
3:    if B(G, w, i) = ja then
4:       k ← i
5: return k
```

Da es nur $O(|V|)$ viele Schleifendurchläufe gibt und B ein Polynomialzeitalgorithmus ist, ist auch B' ein Polynomialzeitalgorithmus.

(ii) Sei C ein Algorithmus, der das Problem Weg-FortsetzungW in polynomieller Zeit löst. Wir konstruieren einen Algorithmus C' der das Problem

WEG-FORTSETZUNGO in polynomieller Zeit löst.

Wir nutzen die folgende Beobachtung. Ist für einen Weg $w = v_1 \dots v_\ell$ bekannt, dass es eine Weg-Fortsetzung der Länge $k > \ell$ gibt, dann gibt es einen Knoten $v_{\ell+1}$ mit $(v_\ell, v_{\ell+1}) \in E$ und $v_{\ell+1} \neq v_i$ für $i \leq \ell$ sodass es auch für den Weg $v_1 \dots v_\ell v_{\ell+1}$ eine Weg-Fortsetzung der Länge k gibt. Um so einen Knoten $v_{\ell+1}$ zu finden, können alle Knoten aus G durchprobiert werden. Für den Test, ob es eine Weg-Fortsetzung der Länge k für $v_1 \dots v_\ell v_{\ell+1}$ gibt, kann jeweils der Algorithmus C verwendet werden.

Algorithmus C'

Eingabe: ungerichteter Graph $G = (V, E)$, Weg $w = v_1 \dots v_\ell$
Ausgabe: maximale Weg-Fortsetzung w' von w

```
1: k ← C(G, w)
2: for all i in ℓ + 1, ..., k do
3:    for all v ∈ V − {v_1, ..., v_{i−1}} mit (v_{i−1}, v) ∈ E do
4:       if C(G, v_1, ..., v_{i−1}, v) ≥ k then
5:          v_i ← v
6: return v_1, ..., v_k
```

Da $k \leq |V|$ ist, besitzt der Algorithmus maximal $|V|^2$ Schleifendurchläufe. Hierbei wird neben konstant vielen Operationen auch C aufgerufen. Da C nach Voraussetzung eine polynomielle Laufzeit $p(|G|, |w|)$ besitzt, ist die Laufzeit des Algorithmus $O(|V|^2 p(|G|, |w|))$.

Aufgabe 11.2 [Polynomielle Reduktionen] **8 Punkte**

Aus der Vorlesung kennen Sie das Erfüllbarkeitsproblem für aussagenlogische Formeln, SAT.

Problem: SAT
Gegeben: aussagenlogische Formel φ in KNF
Frage: Gibt es eine erfüllende Belegung α für φ?

Gegeben sei ferner das folgende Problem ALLEZEICHEN.

Problem: ALLEZEICHEN
Gegeben: regulärer Ausdruck β, Alphabet Σ
Frage: Gibt es ein Wort $w \in L(\beta)$ mit $\#_\sigma(w) > 0$ für jedes $\sigma \in \Sigma$?

a) Seien $\beta = \big(c_1c_3 + c_2\big)\big(c_1c_2 + \varepsilon\big)\big(c_2 + c_1c_3\big)$ und $\Sigma = \{c_1, c_2, c_3\}$. Gilt $(\beta, \Sigma) \in$ ALLEZEICHEN? Falls ja, geben Sie ein Wort $w \in L(\beta)$ an, dass jedes Zeichen aus Σ enthält. **(0,5 Punkte)**

b) Zeigen Sie, dass ALLEZEICHEN in NP ist. **(2 Punkte)**

Gegeben sei ferner die folgende Funktion f, die Instanzen für SAT (aussagenlogische Formeln in KNF) auf Instanzen für ALLEZEICHEN (reguläre Ausdrücke, mit Alphabet) wie folgt abbildet.

Funktion f

Jede Eingabeformel $\varphi = \psi_1 \wedge \cdots \wedge \psi_k$ mit Klauseln $\psi_1, \ldots, \psi_k$ über Variablen $x_1, \ldots, x_m$ wird abgebildet auf $\boldsymbol{f(\varphi) = (\beta_\varphi, \Sigma_\varphi)}$ mit

$$\beta_\varphi = (p(x_1) + n(x_1)) \ldots (p(x_m) + n(x_m)) \text{ und}$$
$$\Sigma_\varphi = \{c_1, \ldots, c_k\}.$$

Dabei bilden die Funktionen p und n jede Variable x_i auf einen regulären Ausdruck ab:

- $p(x_i) = \sigma_{i,1} \ldots \sigma_{i,k}$, wobei $\sigma_{i,j} = c_j$ gilt, falls x_i ein Literal von ψ_j ist, und $\sigma_{i,j} = \varepsilon$ sonst;
- $n(x_i) = \tau_{i,1} \ldots \tau_{i,k}$, wobei $\tau_{i,j} = c_j$ gilt, falls $\neg x_i$ ein Literal von ψ_j ist, und $\tau_{i,j} = \varepsilon$ sonst.

Beispielsweise bildet f die folgende Formel $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_3)$ auf die AlleZeichen-Instanz $f(\varphi) = (\beta, \Sigma)$ aus Teilaufgabe a) ab.

c) Sei $\varphi = (\neg x_2 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3 \vee x_1) \wedge (x_2 \vee x_3 \vee \neg x_3)$. Geben Sie $f(\varphi)$ an. **(0,5 Punkte)**

d) Begründen Sie, warum f in polynomieller Zeit berechenbar ist. **(1 Punkt)**

e) Zeigen Sie, dass f die Reduktionseigenschaft erfüllt, dass also für jede Eingabe φ für Sat gilt:

1. Wenn φ in Sat ist, dann ist $f(\varphi)$ in AlleZeichen.
2. Wenn $f(\varphi)$ in AlleZeichen ist, dann ist φ in Sat.

(4 Punkte)

Lösung:

a) Ja, (β, Σ) ist in AlleZeichen. Belegt wird dies beispielsweise durch das Wort $w = c_1c_3c_2$, denn $w = c_1c_3c_2 = (c_1c_3)(\varepsilon)(c_2)$ ist in $L(\beta)$ und w enthält jedes Zeichen aus Σ.

b) Ein Algorithmus, der AlleZeichen nichtdeterministisch in polynomieller Zeit entscheidet ist nachfolgend angegeben. Die Zusatzeingabe ist dabei ein Wort w. Für dieses wird überprüft, ob es alle Zeichen enthält und dem regulären Ausdruck entspricht.

Algorithmus $\mathcal{A}_{\mathsf{AZ}}$

Eingabe: regulärer Ausdruck β, Alphabet Σ
Zusatzeingabe: Wort w
Ausgabe: ja/nein

```
1: if w enthält nicht jedes Zeichen aus Σ then
2:     return nein
3: Konstruiere ε-NFA B zu RE β
4: Simuliere B bei Eingabe w
5: if B akzeptiert w then
6:     return ja
7: else
8:     return nein
```

Für die folgende Laufzeitanalyse nehmen wir an, dass die Zusatzeingabe w polynomiell lang bezüglich der Eingabe β und Σ ist. Weiter unten begründen wir, dass diese Annahme gerechtfertigt ist.

Laufzeit. Der Enthaltenseinstest für die Zeichen aus Σ kann mit einen Aufwand von $\mathcal{O}(|w||\Sigma|)$ durchgeführt werden (Schritt 1). Der ε-NFA $\mathcal{B}$ zum regulären Ausdruck β kann nach dem induktiven „Baukastenprinzip“ in Zeit $\mathcal{O}(|\beta|)$ konstruiert werden (Schritt 2). Die Simulation von $\mathcal{B}$ für das Wort w kann in Zeit $\mathcal{O}(|\mathcal{B}|^2|w|)$ durchgeführt werden (Schritt 3).[a] Der Akzeptanztest schließlich benötigt Zeit $\mathcal{O}(|\mathcal{B}|) = \mathcal{O}(|\beta|)$.

Insgesamt ergibt sich somit ein Aufwand von $\mathcal{O}(|w||\Sigma| + |\beta| + |\mathcal{B}|^2|w| + |\beta|) = \mathcal{O}(|w||\Sigma| + |\mathcal{B}|^2|w|)$. Da die Zusatzeingabe w polynomielle Länge bezüglich (β, Σ) hat, ist dies insgesamt polynomiell bezüglich (β, Σ).

Existenz einer kleinen Lösung w. Es ist noch zu begründen, dass es eine Zusatzeingabe w polynomieller Länge bezüglich (β, Σ) gibt. Dies ergibt sich aus der folgenden Überlegung.

Der ε-NFA $\mathcal{B}$ hat lineare Größe bezüglich β. Wenn es ein Wort gibt, dass von $\mathcal{B}$ akzeptiert wird und alle Zeichen von Σ enthält, dann gibt es zu jedem Zeichen $\sigma \in \Sigma$ eine Transition bei der das Zeichen σ gelesen wird. Es genügt, jede dieser Transitionen einmal zu durchlaufen. Um eine solche Transition zu erreichen, muss höchstens jede andere Transition einmal durchlaufen werden. Insgesamt ergibt sich also eine obere Schranke von $\mathcal{O}(|\Sigma| \cdot |\mathcal{B}|) = \mathcal{O}(|\Sigma| \cdot |\beta|)$ für die Länge eines Laufes in $\mathcal{B}$, in dem jedes Zeichen mindestens einmal gelesen wird. Die Länge des Laufes beschränkt auch die Länge eines kurzen bezeugenden Wortes w.

c) Die Ausgabe ist $f(\varphi) = \big(\underbrace{(c_2 + c_1)}_{x_1}\underbrace{(c_3 + c_1c_2)}_{x_2}\underbrace{(c_3 + c_2c_3)}_{x_3}, \{c_1, c_2, c_3\}\big)$.

d) Für jede Eingabeformel φ kann die Ausgabe $f(\varphi) = (\beta_\varphi, \Sigma_\varphi)$ in polynomieller Zeit bezüglich $|\varphi|$ berechnet werden.

Berechnung von β_φ. Für jede Variable x_i aus $x_1, \ldots, x_m$ muss der Teilausdruck $p(x_i) + n(x_i)$ erzeugt werden. Alle so erzeugten Teilausdrücke werden dann konkateniert. Wenn der Aufwand für jedes x_i durch ein Polynom q bezüglich $|\varphi|$ beschränkt ist, ergibt sich insgesamt eine Laufzeitschranke von $\mathcal{O}(mq(|\varphi|) = \mathcal{O}(|\varphi|q(|\varphi|))$, da die Anzahl der Variablen höchstens $|\varphi|$ ist.

Der Aufwand für jede Variable x_i ist polynomiell bezüglich $|\varphi|$, da über die Klauseln der Formel iteriert werden kann und in jeder Klausel über die Literale, sodass jedes Literal genau einmal besucht wird (die Anzahl der Literale ist höchstens $|\varphi|$). Für jede Klausel wird dabei protokolliert, ob die Variable x_i positiv auftritt oder nicht, und analog, ob sie negiert auftritt oder nicht. Entsprechend dieses Protokolls werden die Ausgaben $p(x_i)$ und $n(x_i)$ erzeugt.

Berechnung von Σ_φ. Mit linearem Zeitaufwand wird die Formel φ gelesen und die Anzahl k der Klauseln gezählt. Danach werden entsprechend viele Symbole $c_1, \ldots, c_k$ erzeugt.

e) 1. Wenn φ in SAT ist, dann ist $f(\varphi)$ in ALLEZEICHEN.
Angenommen φ ist in SAT, dann gibt es eine erfüllende Belegung für φ. Sei α eine solche Belegung. Wir definieren ein Wort $w_\alpha = w_1 \cdots w_n$ in Abhängigkeit von α: Für jedes $i \in \{1, \ldots, n\}$ sei

$$w_i = \begin{cases} p(x_i), & \text{falls } \alpha(x_i) = 1 \\ n(x_i), & \text{falls } \alpha(x_i) = 0 \end{cases}.$$

Das Wort w_α bezeugt, dass $(\beta_\varphi, \Sigma_\varphi)$ in AlleZeichen ist:

- **$w_\alpha \in L(\beta_\varphi)$:**
  Es gilt $w_i \in \{p(x_i), n(x_i)\} = L\big(p(x_i) + n(x_i)\big)$ für jedes Teilwort w_i und somit $w_\alpha = w_1 \cdots w_n \in L\big(p(x_1) + n(x_1)\big) \circ \cdots \circ L\big(p(x_n) + n(x_n)\big) = L(\beta_\varphi)$ insgesamt.
- **w_α enthält jedes Zeichen aus Σ_φ:**
  Sei $c_j \in \Sigma_\varphi$ beliebig. Dieses Zeichen kommt in w_α vor, wie die folgende Argumentation zeigt. Da Belegung α die Formel φ erfüllt, erfüllt α insbesondere die Klausel ψ_j. Es gibt also

  1. ein Literal x_i in ψ_j, sodass $\alpha(x_i) = 1$ gilt; oder
  2. ein Literal $\neg x_i$ in ψ_j, sodass $\alpha(x_i) = 0$ gilt.

  Wir betrachten den 1. Fall, im 2. Fall kann analog argumentiert werden.

  Nach Definition enthält das Gesamtwort w_α das Teilwort w_i. Dieses Teilwort ist nach Annahme des 1. Falles gerade $w_i = p(x_i)$ und $p(x_i) = \sigma_{i,1} \cdots \sigma_{i,k}$ enthält $\sigma_{i,j} = c_j$, da x_i Literal von ψ_j ist. Also enthält auch w_α das Zeichen c_j.

2. Wenn $f(\varphi)$ in AlleZeichen ist, dann ist φ in Sat.

Angenommen $f(\varphi) = (\beta_\varphi, \Sigma_\varphi)$ ist in AlleZeichen, dann gibt es ein Wort in $L(\beta_\varphi)$, das alle Zeichen aus Σ_φ enthält. Sei w ein solches Wort. Nach Definition von β_φ ist dieses Wort von der Form $w = w_1 \cdots w_n$ mit $w_i \in \{p(x_i), n(x_i)\} = L\big(p(x_i) + n(x_i)\big)$ für jedes $i \in \{1, \ldots, n\}$. Wir definieren die Belegung $\alpha_w : \{x_1, \ldots, x_n\} \to \{0, 1\}$ in Abhängigkeit von w. Für jedes $i \in \{1, \ldots, n\}$ sei

$$\alpha_w(x_i) = \begin{cases} 1, & \text{falls } w_i = p(x_i) \\ 0, & \text{falls } w_i = n(x_i) \end{cases}.$$

Belegung α_w erfüllt Formel φ, bezeugt also, dass φ in Sat ist:

Es genügt zu zeigen, dass jede Klausel $\psi_1, \ldots, \psi_k$ durch α_w erfüllt wird. Dazu sei $j \in \{1, \ldots, k\}$ beliebig gewählt. Nach Voraussetzung enthält das Wort w mindestens einmal das Zeichen c_j.

Insbesondere gibt es ein i, sodass w_i das Zeichen c_j enthält. Nach Definition ist w_i entweder $p(x_i)$ oder $n(x_i)$. Wir betrachten den Fall $w_i = p(x_i)$, im anderen Fall verläuft die Argumentation analog.

Kommt das Zeichen c_j im Wort $w_i = p(x_i)$ vor, dann ist x_i ein Literal in Klausel ψ_j. Nach Definition von α_w gilt für $w_i = p(x_i)$ aber $\alpha_w(x_i) = 1$. Mindestens ein Literal von ψ_j wird also erfüllt und somit die Klausel insgesamt.

[a] Dazu kann pro Zeichen gelesenem Zeichen die Menge der erreichbaren Zustand neu berechnet werden.

Zusatzaufgabe [Polynomielle Reduktionen] **2 Punkte**

Betrachten Sie das folgende Problem VertexCover.

Problem: VertexCover
Gegeben: ungerichteter Graph $G = (V, E)$, Zahl k
Frage: Gibt es eine Menge $S \subseteq V$ mit $|S| \leq k$, sodass jede Kante einen Knoten in S hat?

Gegeben sei ferner das folgende Problem MinMaxZeichen.

Problem: MinMaxZeichen
Gegeben: regulärer Ausdruck α, Zahlen $m_1, m_2 \in \mathbb{N}_0$, Alphabete Σ_1, Σ_2
Frage: Gibt es ein Wort $w \in L(\alpha)$, sodass

- $\#_\sigma(w) \geq m_1$ für jedes $\sigma \in \Sigma_1$ und
- $\#_\sigma(w) \leq m_2$ für jedes $\sigma \in \Sigma_2$ gilt?

Zeigen Sie VertexCover $\leq_p$ MinMaxZeichen.