Grundzüge der Theoretischen Informatik, WS 21/22: Musterlösung zum 11. Übungsblatt

Julian Dörfler

Aufgabe A11.1 (Nicht-deterministischer Platz) *(4 Punkte)*
Wir definieren die folgenden beiden Komplexitätsklassen:

$$\mathsf{PSPACE} = \bigcup_{i \in \mathbb{N}} \mathsf{DSpace}(O(n^i)) \qquad \mathsf{NPSPACE} = \bigcup_{i \in \mathbb{N}} \mathsf{NSpace}(O(n^i))$$

Zeigen Sie nun die beiden folgenden Aussagen:

(a) $\mathsf{PSPACE} = \mathsf{NPSPACE}$

(b) $\mathsf{DSpace}(2^{O(n)}) = \mathsf{NSpace}(2^{O(n)})$

Sie dürfen annehmen, dass Polynome, sowie Funktionen der Form 2^{cn} für Konstanten $c > 1$ platzkonstruierbar sind.
Hinweis: Im Skript befindet sich ein Satz, mit dem die Aussagen schnell bewiesen werden können.

Lösung A11.1 (Nicht-deterministischer Platz) Wir benutzen für beide Aufgabenteile den Satz von Savitch , da sowohl Polynome, als auch Funktionen der Form 2^{cn} platzkonstruierbar sind.

(a) Für alle $i \in \mathbb{N}$ gilt direkt $\mathsf{DSpace}(O(n^i)) \subseteq \mathsf{NSpace}(O(n^i))$, also $\mathsf{PSPACE} \subseteq \mathsf{NPSPACE}$. Bleibt also nur noch zu zeigen, dass für alle $i \in \mathbb{N}$ gilt $\mathsf{NSpace}(O(n^i)) \subseteq \mathsf{PSPACE}$. Dafür verwenden wir den Satz von Savitch und sehen

$$\mathsf{NSpace}(O(n^i)) \subseteq \mathsf{DSpace}((O(n^i)^2) = \mathsf{DSpace}(O(n^{2i})) \subseteq \mathsf{PSPACE}\,.$$

(b) $\mathsf{DSpace}(2^{O(n)}) \subseteq \mathsf{NSpace}(2^{O(n)}) \subseteq \mathsf{DSpace}((2^{O(n)})^2) = \mathsf{DSpace}(2^{O(n)})$

Aufgabe A11.2 (Inklusion in NP) *(4 Punkte)*
Zeigen Sie, dass SAT und HC in NP sind. Beweisen Sie dies einmal indem Sie für *beide* Sprachen jeweilse eine nichtdeterministische Turingmaschine angeben, die diese erkennt und erneut indem Sie ebenfalls für *beide* Sprachen jeweils einen Polynomialzeit-Verifizierer angeben, der diese erkennt.

Lösung A11.2 (Inklusion in NP)

$\mathsf{SAT} \in \mathsf{NP}$*: Polyonmialzeit-Verifizierer:* Wir geben einen Polynomialzeit-Verifizierer V an: Gegeben eine Formel in CNF in n Variablen und ein Zertifikat c, testet V, ob c eine Belegung von $x_1, \ldots, x_n$ kodiert[1]. Sollte dies nicht der Fall sein, so verwirft V. Nun prüft V ob jede Klausel unter dieser Belegung erfüllt ist. Da dies in polynomieller Zeit prüfbar ist, ist V Polynomialzeit-beschränkt, also $\mathsf{SAT} \in \mathsf{NP}$

Nichtdeterministische Turingmaschine: Wir geben eine nichtdeterministische Turingmaschine N an: Gegeben eine Formel in CNF in n Variablen. Rate nun nicht-deterministisch eine Belegung von $x_1, \ldots, x_n$. Danach testet N ob in jeder Klausel mindestens ein Literal durch diese Belegung erfüllt ist. Wenn dies der Fall ist akzeptiere, ansonsten verwerfe. Dies ist in polynomieller Laufzeit prüfbar, also ist N Polynomialzeit-beschränkt und $\mathsf{SAT} \in \mathsf{NP}$.

$\mathsf{HC} \in \mathsf{NP}$*: Polyonmialzeit-Verifizierer:* Wir geben einen Polynomialzeit-Verifizierer V an: Gegeben eine Graphen G und ein Zertifikat c, testet V, ob c eine Permutation $v_1, \ldots, v_n$ der Knoten von G kodiert. Sollte dies nicht der Fall sein, so verwirft V. Danach testet V ob v_i und v_{i+1} für alle $i \in \{1, \ldots, n-1\}$ und v_n und v_1 benachbart sind. Wenn ja, akzeptiert N, in allen anderen Fällen verwirft V. Da dies in polynomieller Zeit prüfbar ist, ist V Polynomialzeit-beschränkt, also $\mathsf{HC} \in \mathsf{NP}$

Nichtdeterministische Turingmaschine: Wir geben eine nichtdeterministische Turingmaschine N an: Gegeben einen Graphen G, raten[2] wir eine Permutation $v_1, \ldots, x_n$ der Knoten von G. Danach testet N ob v_i und v_{i+1} für alle $i \in \{1, \ldots, n-1\}$ und v_n und v_1 benachbart sind. Wenn ja, akzeptiert N, in allen anderen Fällen verwirft N. Dies ist in polynomieller Laufzeit prüfbar, also ist N Polynomialzeit-beschränkt und $\mathsf{HC} \in \mathsf{NP}$.

Aufgabe A11.3 (NP-Reduktionen) *(4 Punkte)*
Wir bezeichnen eine Menge von k Knoten, so dass diese paarweise nicht mit Kanten verbunden sind, als ein k-Independent-set. Wir definieren das Problem

$$\mathsf{IS} = \{(G, k) \mid \text{Der (ungerichtete) Graph } G \text{ enthält ein } k\text{-Independent-set}\}$$

Wir zeigen nun, dass IS, VC und Clique unter Polynomialzeitreduktionen äquivalent sind. Konstruieren Sie explizit hierzu die folgenden Reduktionen:

(a) $\mathsf{Clique} \leq_{\mathrm{P}} \mathsf{IS}$

(b) $\mathsf{IS} \leq_{\mathrm{P}} \mathsf{VC}$

[1] Eine Möglichkeit wäre ein Bit pro Variable, wobei das i-te Bit angibt, ob die i-te Variable auf wahr gesetzt ist.

[2] *Raten* bedeutet hier, dass N nichtdeterministisch einen Bitstring rät und diesen als eine Permutation der Knoten von G interpretiert, falls dieser keine Permutation kodiert, so verwirft N.

(c) VC $\leq_{\mathrm{P}}$ Clique

Lösung A11.3 (NP-Reduktionen)
In allen diesen Teilaufgaben definieren wir für einen Graph $G = (V, E)$ den Komplementgraph $\overline{G} = (V, \binom{V}{2} \setminus E) = (V, \overline{E})$.

(a) Wir bilden (G, k) auf $(\overline{G}, k)$ ab. Dies ist offensichtlich in Polynomialzeit möglich.

Wir zeigen nun, wenn G eine k-Clique enthält, dann enthält $\overline{G}$ ein k-Independent-set. Sei also $v_1, \ldots, v_k$ eine Clique in G, dann existieren alle Kanten $\{v_i, v_j\} \in E$ mit $i \neq j$. Damit existiert aber in $\overline{G}$ keine dieser Kanten, $v_1, \ldots, v_k$ bildet also ein k-Independent-set in $\overline{G}$.

Wir zeigen nun, wenn $\overline{G}$ ein k-Independent-set enthält, dann enthält G eine k-Clique. Sei also $v_1, \ldots, v_k$ ein Independent-set in $\overline{G}$, dann existiert keine der Kanten $\{v_i, v_j\} \in E$ für $i \neq j$. Damit existieren aber in G alle diese Kanten, $v_1, \ldots, v_k$ bildet also eine k-Clique in $\overline{G}$.

(b) Wir bilden (G, k) auf $(G, |V| - k)$ ab, wobei $G = (V, E)$. Dies ist offensichtlich in Polynomialzeit möglich.

Wir zeigen nun, wenn G eine k-Independent-set enthält, dann enthält G ein $|V| - k$-Vertex Cover. Sei also $v_1, \ldots, v_k$ ein Independent-set in G, dann existiert keine der Kanten $\{v_i, v_j\} \in E$ für $i \neq j$. Somit haben alle Kanten mindestens ein Ende in $V \setminus \{v_1, \ldots, v_k\}$, dies ist also ein Vertex Cover der Größe $|V| - k$.

Wir zeigen nun, wenn G ein Vertex Cover der Größe maximal $|V| - k$ enthält, dann enthält G ein k-Independent-set. Sei also C ein Vertex Cover der Größe maximal $|V| - k$. Dann gibt es k Knoten $v_1, \ldots, v_k$ in $V \setminus C$. Nehmen wir an, zwischen zwei Knoten v_i, v_j mit $i \neq j$ existiert eine Kante. Dies ist direkt ein Widerspruch dazu, dass C ein Vertex Cover ist, also von jeder Kante mindestens einen Endpunkt enthält. Somit existieren keine Kanten $\{v_i, v_j\}$ in E und $v_1, \ldots, v_k$ ist ein k-Independent-set in G.

(c) Wir bilden (G, k) auf $(\overline{G}, |V| - k)$ ab, wobei $G = (V, E)$. Dies ist offensichtlich in Polynomialzeit möglich.

Wir zeigen nun, wenn G ein Vertex Cover der Größe maximal k enthält, dann enthält $\overline{G}$ eine $|V| - k$-Clique. Sei also C ein Vertex Cover der Größe maximal k in G. Dann gibt es $|V| - k$ Knoten $v_1, \ldots, v_{|V|-k}$ in $V \setminus C$. Nehmen wir an, zwischen zwei Knoten v_i, v_j mit $i \neq j$ existiert keine Kante in $\overline{G}$. Dann existiert eine Kante zwischen v_i und v_j in G. Dies ist direkt ein Widerspruch dazu, dass C ein Vertex Cover ist. Somit existieren alle Kanten $\{v_i, v_j\}$ in $\overline{G}$ und $v_1, \ldots, v_{|V|-k}$ ist eine $|V| - k$-Clique in $\overline{G}$.

Wir zeigen nun, wenn $\overline{G}$ eine $|V| - k$-Clique enthält, dann enthält G ein Vertex Cover der Größe maximal k. Sei $v_1, \ldots, v_{|V|-k}$ also eine $|V| - k$-Clique in $\overline{G}$. Dadurch existieren in $\overline{G}$ also alle Kanten $\{v_i, v_j\}$, diese existieren also in G nicht. Wählen wir nun die restlichen k Knoten in V als C, dann existieren nach dem entfernen von C aus G also keine Kanten mehr. C ist somit ein Vertex Cover der Größe k in G.

Aufgabe A11.4 (2SAT) *(4 Punkte)*
Sei

$$\mathsf{2SAT} = \{\phi \mid \phi \text{ ist eine erfüllbare Formel in 2CNF}\}.$$

Beweisen Sie: $\mathsf{2SAT} \in \mathsf{P}$.
Hinweise: Konstruieren Sie zu einer Formel ϕ einen gerichteten Graphen $G = (V, E)$ mit $V = \{x, \overline{x} \mid x \text{ ist Variable in } \phi\}$. Die Kanten ergeben sich aus den Klauseln. Es könnte hierbei nützlich sein, eine Klausel $(\ell \vee \ell')$ äquivalent aufzufassen als $(\overline{\ell} \to \ell')$. Zeigen Sie, dass ϕ genau dann nicht erfüllbar ist, wenn es eine Variable x gibt, so dass es einen Weg von x nach $\overline{x}$ und einen Weg von $\overline{x}$ nach x in G gibt.

Bonusaufgabe (2 Bonuspunkte)*:* Zeigen Sie: $\mathsf{2SAT} \in \mathsf{NL}$. (Dies ist eine stärkere Aussage, da aus den Simulationssätzen folgt, dass $\mathsf{NL} \subseteq \mathsf{P}$.) Hierzu dürfen Sie den Satz von Immerman und Szelepcsényi benutzen, auch wenn wir diesen in der Vorlesung nicht bewiesen haben.

Lösung A11.4 (2SAT)
Eine Klausel $\ell \vee \ell'$ ist äquivalent zu $\overline{\ell} \to \ell'$ bzw. $\overline{\ell}' \to \ell$. Wir fügen entsprechende gerichtete Kanten $(\overline{\ell}, \ell')$ und $(\overline{\ell}', \ell)$ in den Graph ein.

Hierraus folgt insbesondere, dass wenn es einen Pfad von einem Literal ℓ zu einem Literal ℓ' gibt, dass es dann ebenfalls einen Pfad von $\overline{\ell}'$ nach $\overline{\ell}$ gibt.

Gibt es eine Variable x, so dass es in dem Graph für ϕ einen Pfad von $\overline{x}$ nach x und einen Pfad von x nach $\overline{x}$ gibt, dann ist ϕ nicht erfüllbar: Wäre ϕ erfüllbar mit $x = 0$, dann würde dies $x = 1$ implizieren und umgekehrt.

Es bleibt zu zeigen, dass ϕ ansonsten erfüllbar ist. Dazu konstruieren wir iterativ eine erfüllende Belegung für ϕ:

Sei x ein Literal, dem wir noch keinen Wert zugewiesen haben und bei dem es keinen Pfad von x nach $\overline{x}$ gibt. Wir setzen $x = 1$. Alle Knoten y, die von x aus erreichbar sind, setzen wir ebenfalls auf 1. (D.h. wir weisen den entsprechenden Literalen 1 zu, den entsprechenden Variablen also 0 oder 1.) Deren Negationen $\overline{y}$ setzen wir auf 0.

Dies ist immer möglich: Gäbe es Pfade von x nach y und $\overline{y}$, dann gäbe es auch Pfade von y und $\overline{y}$ nach $\overline{x}$, also einen Pfad von x nach $\overline{x}$. Gäbe es einen Pfad von x zu einem Knoten y, der bereits auf 0 gesetzt wäre, dann gibt es auch einen Pfad von $\overline{y} = 1$ zu $\overline{x}$, also wäre $x = 0$ bereits gewählt.

Da keine Konflikte entstehen, weisen wir jedem Literal x einen Wert mit $x = 1 - \overline{x} \in \{0, 1\}$ zu. Es gibt keine Kante (x, y) mit $x = 1$ und $y = 0$ nach Konstruktion. Also haben wir eine erfüllende Belegung konstruiert, also ist ϕ erfüllbar.

Folgender Algorithmus ist ein Polynomialzeitalgorithmus für 2SAT. Die Erreichbarkeit zu Testen ist in linearer Zeit möglich.
Input: 2CNF ϕ mit Variablen U
1: Konstruiere den Implikationsgraphen $G = (V, E)$ wie beschrieben.
2: **for** $x \in U$ **do**
3: Gibt es Pfade von x nach $\overline{x}$ und von $\overline{x}$ nach x in G, verwerfe.
4: **od**

5: Akzeptiere

Für die Bonusaufgabe geben wir anhand obiger Beobachtungen zuerst einen Algorithmus für $\overline{\mathsf{2SAT}} \in \mathsf{NL}$ an:

Input: 2CNF ϕ mit Variablen U

1: Rate nicht-deterministisch ein Literal x aus U.
2: Prüfe, ob ein Pfad von x nach $\overline{x}$ im Implikationsgraphen von ϕ existiert.
3: Akzeptiere wenn solch ein Pfad akzeptiert, ansonsten verwerfe.

Wir müssen hierbei den Implikationsgraphen nicht explizit speichern, sondern können uns mit logaritmischem Platz die relevanten Kanten jeweils bei der Prüfung von Adjazenz neu berechnen. Es gilt also $\mathsf{2SAT} \in \mathsf{co\text{-}NL} = \{\overline{L} \mid L \in \mathsf{NL}\}$.

Nach dem Satz von Immerman und Szelepcsényi gilt aber $\mathsf{co\text{-}NL} = \mathsf{NL}$, wir haben also $\mathsf{2SAT} \in \mathsf{NL}$.

Aufgabe A11.5 (Limits der Platzkonstruierbarkeit) *(4 Bonuspunkte)*
Sei s eine platzkonstruierbare Funktion mit $s(n) = o(\log n)$ und sei M eine $O(s)$-platzbeschränkte deterministische Turingmaschine (mit Extra-Eingabeband), die s berechnet. Der Einfachheit halber nehmen wir an, dass M den Kopf auf dem Eingabeband nicht nach links bewegen darf, den Input also genau einmal liest.

a) Aus einer Konfiguration von M erhält man die zugehörige *kleine* Konfiguration, indem man die Position auf dem Eingabeband weglässt. Zeigen Sie: Auf Eingaben der Länge n kann M nur $o(n)$ viele verschiedene kleine Konfiguration durchlaufen.

b) Zeigen Sie: Es gibt einen String x, so dass M für unendlich viele Eingaben der Form 1^i den String x ausgibt.
(Hinweis: Erinnern Sie sich an das Pumpinglemma.)

c) Zeigen Sie: $\lfloor \log \log n \rfloor$ ist nicht platzkonstruierbar durch eine Turingmaschine, die den Kopf auf dem Eingabeband nicht nach links bewegen darf.

d) *(*)* Verallgemeinern Sie die Argumentation auf Turingmaschinen, die den Kopf auf dem Eigabeband auch nach links bewegen dürfen.

Lösung A11.5 (Limits der Platzkonstruierbarkeit) (a) Die Anzahl kleiner Konfigurationen von M ist gegeben durch $c^{s(n)}$, wobei c eine Konstante ist, die nur von M abhängt. Sei nun $\varepsilon > 0$ beliebig. Dann gibt es ein n_0, so dass für alle $n \geq n_0$ gilt $c^{s(n)} \leq c^{\varepsilon \log n} = n^{\varepsilon \log c}$. Da ε beliebig war und $\log c$ konstant ist, gilt also $c^{s(n)} = o(n)$.

(b) Da M nur $o(n)$ viele kleine Konfigurationen durchläuft gibt es ein n_0, so dass M weniger als n_0 kleine Konfigurationen durchläuft bei Eingabe 1^{n_0}. Nun müssen wir zwei Fälle unterscheiden:

M liest die gesamte Eingabe ein: Dann wiederholt sich eine kleine Konfiguration bei der Berechnung von M auf Eingabe 1^n. Nehmen wir an zwischen diesen beiden Vorkommen der gleichen kleinen Konfiguration wurden k Zeichen der Eingabe

einglesen. Es gilt $k > 0$, da M ansonsten nicht terminieren würde und somit nicht s berechnen könnte. Da M deterministisch ist, wird M nach weiteren k eingelesenen Zeichen der Eingabe aber wieder in der selben kleinen Konfiguration sein, induktiv verhält sich M also für alle $i \in \mathbb{N}$ auf den Eingaben $1^{n_0+i\cdot k}$ identisch zu der Eingabe 1^{n_0}, produziert also auf allen diesen unendlich vielen Eingaben die gleiche Ausgabe.

M liest nicht die gesamte Eingabe ein: Dann wird M bei allen Eingaben 1^n mit $n \geq n_0$ die selbe Ausgabe ausgeben, da M nur maximal die ersten $n_0 - 1$ Zeichen der Eingabe einliest und diese für beide Eingaben identisch sind.

(c) Sei M eine Maschine wie in der Aufgabenstellung, die $\lfloor \log \log n \rfloor$ berechnet. Da $\lfloor \log \log n \rfloor$ monoton und unbeschränkt ist, gibt es keinen Wert, den $\lfloor \log \log n \rfloor$ für unendlich viele Eingaben annimmt. Weiterhin ist $\lfloor \log \log \rfloor n = o(\log n)$, es muss also nach Aufgabenteil (b) eine Ausgabe geben, die von unendlich vielen Eingaben produziert wird. Dies ist ein Widerspruch, also kann M nicht existeren und s somit nicht platzkonstruierbar sein durch eine DTM, die ihren Kopf nie nach links bewegt.

(d) Wir wählen n_0 wie in (b).

Nun vergleichen wir wie sich M auf Eingabe 1^{n_0} und Eingabe $1^{n_0+i\cdot n_0!}$ für $i \in \mathbb{N}$ verhält. Nun wird M auf Eingabe 1^{n_0} das linke und das rechte Ende der Eingabe in einer gewissen Abfolge besuchen. Nun zeigen wir induktiv, dass M auf Eingabe $1^{n_0+i\cdot n_0!}$ das linke und rechte Ende der Eingabe in der gleichen Abfolge besuchen und dabei immer in der selben kleinen Konfiguration sind:

I.A. Am Anfang ist M offensichtlich immer am linken Ende der Eingabe und in der selben kleinen Konfiguration.

M besucht das selbe Ende direkt erneut: Nun bewegt sich der Kopf von M auf dem Eingabeband um maximal als n_0 Positionen vom entsprechenden Ende der Eingabe weg. Da $1^{n_0+i\cdot n_0!}$ aber mehr als n_0 Einsen enthält wird sich M bei beiden Eingaben identisch verhalten und somit auch in das gleiche Ende der Eingabe in der selben kleinen Konfiguration erreichen.

M besucht das andere Ende der Eingabe als nächstes: Hierzu müssen mindestens n_0 Schritte verwendet werden, daher existiert eine kleine Konfiguration die sich in weniger als n_0 Schritten wiederholen muss. Nun ist aber $n_0!$ offensichtlich durch diese Anzahl an Schritten teilbar, also wird M auf Eingabe $1^{n_0+i\cdot n_0!}$ das andere Ende der Eingabe in der selben Konfiguration wie bei Eingabe 1^{n_0} erreichen.

Am Ende kann M eventuell noch eine restliche Berechnung ausführen, bei der M kein Ende der Eingabe mehr erreicht. Mit der selben Argumentation wie oben verhält sich M nun weiterhin auf beiden Eingaben identisch, wird also die selbe Ausgabe produzieren für alle $i \in \mathbb{N}$.