Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
L. Hübschle-Schneider, T. Maier

12. Übungsblatt zu Theoretische Grundlagen der Informatik im WS 2015/16

http://algo2.iti.kit.edu/TGI2015.php
{sanders,huebschle,t.maier}@kit.edu

Musterlösungen

Aufgabe 1 (*FREE EDGE, 2 Punkte*)

Wir definieren das Entscheidungsproblem FREE EDGE: Gegeben sei ein (ungerichteter) Graph $G = (V, E)$ und $k \leq |V| \in \mathbb{N}$. Gibt es für jedes $V' \subseteq V$ mit $|V'| = k$ eine Kante $\{u, v\} \in E$, sodass weder u noch v in V' enthalten sind?
Formulieren Sie das komplementäre Problem co-FREE EDGE und zeigen Sie, dass FREE EDGE $\in$ co-NP.

Musterlösung:
In mathematischer Schreibweise lautet das FREE EDGE-Problem

$$\forall_{V' \subseteq V,\ |V'|=k} \exists_{\{u,v\} \in E} : (u \notin V' \wedge v \notin V')?$$

Für das co-Problem müssen wir nun diese Formel negieren und erhalten

$$\exists_{V' \subseteq V,\ |V'|=k} \forall_{\{u,v\} \in E} : (u \in V' \vee v \in V')?$$

Dieses Problem ist aus der Vorlesung schon als VERTEX COVER bekannt.

VERTEX COVER (co-FREE EDGE): Gegeben sei ein (ungerichteter) Graph $G = (V, E)$ und $k \leq |V| \in \mathbb{N}$. Existiert eine Teilmenge V' von V mit $|V'| = k$, sodass für jede Kante $\{u, v\} \in E$ mindestens einer ihrer beiden Endpunkte in V' liegt?

Um zu zeigen, dass FREE EDGE $\in$ co-NP genügt es zu zeigen, dass VERTEX COVER $\in$ NP. Dies ist aber aus Vorlesung und Übung bereits bekannt.

Aufgabe 2 (*NP-vollständige Wegeprobleme in Graphen, 2 + 3 Punkte*)

Gegeben seien folgenden Probleme:

HAMILTON PATH (NP-vollständig): Gegeben einem ungerichteten Graph $G = (V, E)$, gibt es einen einfachen (zykelfreien) Pfad der jeden Knoten des Graphen durchläuft (beachte: ein einfacher Pfad muss kein einfacher Zykel sein $\Rightarrow$ HAMILTON PATH $\neq$ HAMILTON CYCLE)?

LONGEST PATH: Gegeben einem ungerichteten Graph $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{N}$ und einer Zahl $k \in \mathbb{N}$, gibt es einen einfachen Pfad mit Länge $\geq k$ in G?

DEGREE-CONSTRAINED SPANNING TREE: Gegeben einem ungerichteten Graph $G = (V, E)$ und einer Zahl $k \in \mathbb{N}$, existiert ein Spannbaum T_k in G dessen Knoten einen Grad von maximal k aufweisen?

Zeigen sie (Gehen Sie davon aus, dass HAMILTON PATH NP-vollständig ist):

a) das Problem LONGEST PATH ist NP-vollständig.

b) das Problem DEGREE-CONSTRAINED SPANNING TREE ist NP-vollständig.

Musterlösung:

a) Zunächst müssen wir zeigen, dass LONGEST PATH in NP liegt. Hierzu raten wir nichtdeterministisch eine Folge von Knoten $(v_1, \dots v_\ell)$. Für diese Folge überprüfen wir, ob zwischen je zwei Knoten v_i und v_{i+1} eine Kante existiert und ob die Summe der zugehörigen Kantengewichte $\geq k$ ist. Außerdem muss überprüft werden, dass die Folge keinen Knoten doppelt enthält. Dieser Test ist trivial in $\mathcal{O}(n)$ durchführbar ($\Rightarrow$ LONGEST PATH ist in NP).

Nun müssen wir zeigen, dass LONGEST PATH NP-schwer ist. Hierzu reduzieren wir HAMILTON PATH auf LONGEST PATH (HAMILTON PATH $\leq_p$ LONGEST PATH). Sei eine HAMILTON PATH-Instanz I_{HAM} gegeben durch den Graph G_{HAM}. Wir konstruieren die LONGEST PATH-Instanz I_{LP} indem wir den gegebenen Graphen übernehmen ($G_{HAM} = G_{LP}$), die Kostenfunktion setzen wir auf konstant $\equiv 1$ ($\forall_{v \in V} c(v) = 1$) und $k = n - 1$. Es ist leicht erkennbar, dass ein einfacher Graph mit $n - 1$ Kanten ein Hamiltonpfad ist. Dadurch lässt sich leicht zeigen: Die Instanz I_{LP} ist genau dann lösbar wenn G_{HAM} einen Hamiltonpfad besitzt.

Wir haben somit gezeigt, dass LONGEST PATH NP-vollständig ist, denn es liegt in NP und ist NP-schwer.

b) Zunächst müssen wir zeigen, dass DEGREE-CONSTRAINED SPANNING TREE in NP liegt. Hierzu raten wir (nichtdeterministisch) eine Kantenmenge E_T und überprüfen ob der Graph $G_T = (V, E_T)$ ein korrekter Spannbaum ist, der die gegebenen Einschränkungen erfüllt (zusammenhängend, Knotengrade kleiner k). Alle Tests lassen sich trivial in polynomieller Zeit durchführen.

Nun müssen wir zeigen, dass DEGREE-CONSTRAINED SPANNING TREE NP-schwer ist. Hierzu reduzieren wir erneut HAMILTON PATH, diesmal auf das Problem DEGREE-CONSTRAINED SPANNING TREE. Sei I_{HAM} eine Instanz von HAMILTON PATH. Wir erzeugen eine DEGREE-CONSTRAINED SPANNING TREE-Instanz I_{DCST}, indem wir erneut den Graphen übernehmen ($G_{HAM} = G_{DCST}$). Außerdem setzen wir $k = 2$. Um zu begründen, warum diese Transformation funktioniert genügt es festzustellen, dass ein zusammenhängender Graph mit Grad 1 und 2 Knoten ein einfacher Pfad oder ein einfacher Zykel ist. Daraus folgt dass ein Spannbaum mit Grad-1- und -2-Knoten den gewünschten Hamiltonpfad liefert.

Wir haben somit gezeigt, dass das Problem DEGREE-CONSTRAINED SPANNING TREE NP-vollständig ist, indem wir gezeigt haben, dass es in NP liegt und NP-schwer ist.

Aufgabe 3 (*Eindeutige Gewichte,* 5 *Punkte*)

Gegeben seien die Probleme BIN PACKING und UNIQUE PACKING.

BIN PACKING (NP-vollständig): eine Instanz von BIN PACKING sei gegeben durch eine Menge $M = \{v_1, \dots, v_n)$ von Elementen, eine Gewichtsfunktion $c : M \to \mathbb{N}$, eine Kapazität $W \in \mathbb{N}$ und einer Zahl $k \in \mathbb{N}$. Gibt es eine Verteilung ($d : V \to \{1, \dots, k\}$) aller Elemente auf maximal k Behälter sodass keine Teilmenge ein Gewicht größer W besitzt?

UNIQUE PACKING: Die Definition von UNIQUE PACKING entspricht der von BIN PACKING mit dem Unterschied, dass die Gewichtsfunktion injektiv sein muss, d.h. keine zwei Elemente können das selbe Gewicht haben.

Zeigen Sie, dass UNIQUE PACKING NP-vollständig ist (unter der Annahme, dass BIN PACKING NP-vollständig ist).

Musterlösung:
Zunächst zeigen wir UNIQUE PACKING $\in$ NP. Hierzu genügt es eine nichtdeterministisch geratene Lösung (Verteilungsfunktion) in polynomieller Zeit zu verifizieren. Dies ist einfach machbar durch Aufaddieren des Gewichts in jedem Behälter.

Nun müssen wir BIN PACKING auf UNIQUE PACKING reduzieren. Informell: Wir müssen die Gewichte aller Elemente eindeutig machen, ohne die Erfüllbarkeit der Instanz zu verändern. Dies geschieht in mehreren Schritten. Es sei I_{BP} eine BIN PACKING Instanz (M_{BP}, c_{BP}, W_{BP} und k_{BP}). Außerdem sei ℓ die Anzahl aller Elemente mit uneindeutigem Gewicht.
Wir wählen $M_{UP} = M_{BP}$ und $k_{UP} = k_{BP}$. Die folgenden Umformungen verändern die Erfüllbarkeit nicht:

- $c'_{BP} = (\ell^2 + 1) \cdot c_{BP} \quad W'_{BP} = (\ell^2 + 1) W_{BP}$

  Diese Veränderung verändert die Erfüllbarkeit nicht (klar)!

- Es sei $pre(v_i) = |\{v_j \in M \mid \ j < k \wedge c_{BP}(v) = v_{BP}(v_i)\}|$, also die Anzahl der Elemente vor v_i mit dem selben Gewicht wie v_i

  Wir wählen $c_{UP}(v_i) \ = \ c'(v_i) + pre(v_i) \ = \ (\ell^2 + 1) \cdot c_{BP}(v_i) + pre(v_i)$ und $W_{UP} = W_{BP} + \ell^2$

  Es bleibt zu zeigen, dass diese Veränderung das Erfüllbarkeitsverhalten nicht verändert.

  (I'_{BP} erfüllbar $\Rightarrow$ I_{UP} erfüllbar) Betrachten wir eine erfüllende Verteilung von I'_{BP}. In jedem einzelnen Behälter können nur maximal ℓ Elemente liegen, die ein uneindeutiges Gewicht besitzen. Jedes davon wiegt maximal $c'_{BP} + \ell$. Daraus folgt: In I_{UP} wiegen die Elemente dieses Behälters maximal $W + \ell^2$.

  (I_{UP} erfüllbar $\Rightarrow$ I'_{BP} erfüllbar) Betrachten wir eine erfüllende Verteilung von I_{UP}. Seien T_j alle Elemente des j-ten Behälters. Diese wiegen insgesammt $w_j = r \cdot (\ell^2 + 1) + q$ (wobei $q \leq \ell^2 + 1$). Um zu zeigen, dass $r = \sum_{v \in T_j} c_{BP}(v)$ genügt zu zeigen, dass alle erhöhten Gewichte zusammen weniger als ℓ^2 ausmachen.

Wir haben somit gezeigt, dass das Problem UNIQUE PACKING NP-vollständig ist, indem wir gezeigt haben, dass es in NP liegt und NP-schwer ist.

Aufgabe 4 (*Pseudopolynomialzeitalgorithmus für PARTITION, 5 Punkte*)

Geben Sie einen Pseudopolynomialzeitalgorithmus für PARTITION an (Tipp: dynamische Programmierung). Geben Sie eine obere Schranke für die Laufzeit an (diese muss polynomiell in der Anzahl der Zahlen in der Eingabe und der größten vorkommenden Zahl sein).

Hinweis: Sie müssen dabei nicht auf Turingmaschinenebene argumentieren, sondern können sich am RAM-Modell orientieren. Das bedeutet, dass Sie beispielsweise annehmen dürfen, dass Addition und Vergleich zweier Zahlen in Zeit $\mathcal{O}(1)$ möglich sind.

Musterlösung:
Sei $M = \{x_1, \ldots, x_n\}$ die PARTITION-Instanz. Beachte, dass M eine Multimenge ist, doppelte Elemente in der Eingabe sind also erlaubt. Wir suchen eine Partitionierung von M in zwei Mengen M_1 und M_2 mit identischen Summen. Wenn die Gesamtsumme $\sum_{x \in M} x$ ungerade ist, ist die Instanz offensichtlich unlösbar und wir sind fertig. Ansonsten reicht es, M_1 so zu bestimmen, dass die Summe der Elemente gleich der Hälfte der Summe aller Elemente entspricht; M_2 ist dann $M \setminus M_1$. Das Zielgewicht der beiden Mengen beträgt also $S := \frac{1}{2} \sum_{i=1}^{n} x_i$. Zudem definieren wir die Indikatorfunktion $Q(i, s)$, die genau dann wahr ist, wenn eine Teilmultimenge von $x_1, \ldots, x_i$ existiert, deren Summe genau s ist. Die Lösung der PARTITION-Instanz ist also $Q(n, S)$.

Offensichtlich ist $Q(n, s) = \textbf{false}$ falls $s < 0$ oder $s > 2S$ und uninteressant für $s > S$. Diese Werte können wir also getrost ignorieren. Für $Q(0, \cdot)$ ist keine Lösung möglich, da keine Elemente zur Auswahl stehen; $Q(\cdot, 0)$ hingegen ist trivial erfüllbar. Wir definieren Q nun als dynamisches Programm wie folgt:

$Q(0, s) := \textbf{false}$	für $0 \leq s \leq S$
$Q(i, 0) := \textbf{true}$	für $1 \leq i \leq n$
$Q(i, s) := Q(i-1, s) \vee Q(i-1, s - x_i)$	für $1 \leq i \leq n$ und $0 < s \leq S$

Dazu speichern wir bereits berechnete Werte in einem zweidimensionalen Array, in dem der Wert von $i \in \{0, \ldots, n\}$ die Zeilen und $s \in \{0, \ldots, X\}$ die Spalten adressiert. Die Zeile $i = 0$ ist dabei ein Dummy, der die Sonderfallbehandlung vereinfacht. Jede mögliche rechte Seite von $Q(i, s)$ können wir dann entweder trivial berechnen, auf Einträge zurückführen, die weiter "oben links" im Array stehen und daher bereits bekannt sind (im Fall dass $s - x_i < 0$ wird, ist der Wert als **false** bekannt). Wenn wir uns für jeden Eintrag des Arrays noch merken, mit welcher der drei Möglichkeiten dieser erzeugt wurde, können aus dem Eintrag in Position (n, S) auch leicht die Teilmenge berechnen, die die Instanz löst.

Die Laufzeit dieses dynamischen Programms ist $\mathcal{O}(nS)$. Um die Laufzeit abhängig von n und der größten Zahl g in der Eingabe anzugeben, können wir $S \leq ng$ abschätzen und erhalten $\mathcal{O}(gn^2)$. Dies ist *pseudo*polynomiell, da g nicht polynomiell in der *Größe des Eingabeproblems* ist, sondern exponentiell in der Anzahl Bits einer Respräsentation der Instanz.

Ein schnellerer Pseudopolynomialzeitalgorithmus mit Komplexität $\tilde{\mathcal{O}}(\sqrt{n}S)$ wurde von Koiliaris und Xu in *A Faster Pseudopolynomial Time Algorithm for Subset Sum*, arXiv:1507.02318 (2015) vorgestellt. Letztlich ist PARTITION nichts anderes als ein Spezialfall von SUBSET SUM.

Aufgabe 5 (*Approximierbarkeit von MAX2SAT,* 3 *Punkte*)

Wir definieren das Optimalwertproblem MAX2SAT: Gegeben sei eine Menge von n 2SAT-Klauseln. Wie viele dieser Klauseln sind maximal gleichzeitig erfüllbar?

Sie dürfen annehmen, dass das Entscheidungsproblem zu MAX2SAT (*sind mehr als k Klauseln gleichzeitig erfüllbar?)* NP-vollständig ist. Nehmen Sie außerdem an, dass P $\neq$ NP.

Zeigen Sie, dass dann es keinen Polynomialzeitalgorithmus gibt, der MAX2SAT mit einem konstanten Maximalfehler e löst. Ein solcher Algorithmus darf bei k erfüllbaren Klauseln eine Antwort im Bereich $k - e$ bis $k + e$ geben.

Musterlösung:
Wenn es einen solchen Algorithmus gäbe, dann könnten wir eine neue Klauselmenge ϕ' konstruieren, in der jede Klausel der ursprünglichen Instanz (nennen wir sie ϕ) $(2e + 1)$-mal vorkommt. Diese Klauselmenge ist nur um einen konstanten Faktor größer als ϕ, da e eine Konstante des Approximationsalgorithmus ist. Durch die Vervielfachung der Klauseln können wir jetzt aber ablesen, wie viele Klauseln von ϕ erfüllbar sind, da sich die Ergebnisintervalle des Approximationsalgorithmus nicht überschneiden: Wenn in ϕ maximal k Klauseln gleichzeitig erfüllbar sind, liegt das Ergebnis für ϕ' zwischen $(2e + 1)k - e > (2e+1)k-e-1 = (2e+1)(k-1)+e$ und $(2e+1)k+e < (2e+1)k+e+1 = (2e+1)(k+1)-e$. Wenn also k Klauseln in ϕ gleichzeitig erfüllbar sind, dann können wir das aus dem approximierten Ergebnis für ϕ' eindeutig von $k - 1$ oder $k + 1$ erfüllbaren Klauseln unterscheiden, denn die Intervalle überschneiden sich nicht. Dies ist aber ein Widerspruch zur Annahme P $\neq$ NP, denn mit dem Algorithmus für das Optimalwertproblem können wir leicht das Entscheidungsproblem lösen.