Lehrstuhl für Informatik 1
Prof. Dr. Gerhard Woeginger
Jan Böker, Tim Hartmann

WS 2018/19

Übung zur Vorlesung
Berechenbarkeit und Komplexität

Lösung Blatt 12

Tutoriumsaufgabe 12.1

Das Makespan-Scheduling-Problem ist das folgende Optimierungsproblem:

Makespan-Scheduling
Eingabe: m **Maschinen,** n **Jobs mit Laufzeiten** $p_1, \ldots, p_n$.
zulässige Lösungen: **Jede Zuteilung** $s\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$ **der Jobs auf die Maschinen.**
Zielfunktion: **Minimiere den Makespan, d.h. minimiere** $\max_{1 \le i \le m} \sum_{j:s(j)=i} p_j$.

(a) Definieren Sie die Entscheidungsvariante des Makespan-Scheduling-Problems.

Makespan-Scheduling-E
Eingabe: m Maschinen, n Jobs mit Laufzeiten $p_1, \ldots, p_n$, $b \in \mathbb{N}$
Frage: Gibt es eine Abbildung $s\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$, sodass $\sum_{j:s(j)=i} p_j \le b$ für alle $1 \le i \le m$?

(b) Beschreiben Sie eine polynomielle Reduktion von Subset-Sum auf die Entscheidungsvariante von Makespan-Scheduling und beweisen Sie ihre Korrektheit.

Hinweis: Einen NP-Härte-Beweis könnte man ganz einfach über Bin Packing führen. Hier wurde jedoch die Reduktion von Subset-Sum verlangt.

Die Eingabe für Subset-Sum sei $((a_1, \ldots, a_N), b)$. Wir konstruieren daraus eine Eingabe für Makespan-Scheduling-E. Diese hat $N + 2$ Jobs. Für alle $1 \le i \le N$ setze die Laufzeiten der Jobs auf $p_i = a_i$. Weiterhin: $p_{N+1} = 2A - b$ und $p_{N+2} = A + b$, wobei $A = \sum_{i=1}^{N} a_i$. Die Anzahl der Maschinen setzen wir auf 2. Sei $2A$ der geforderte Makespan, also die maximale Laufzeit einer Maschine. Diese dadurch gegebene transformierende Funktion ist polynomiell berechenbar.

Sei jetzt $((a_1, \ldots, a_N), b) \in$ Subset-Sum. Also gibt es eine Indexmenge $M \subseteq \{1, \ldots, N\}$ mit $\sum_{i \in M} a_i = b$. Definiere die Zuordnung $s\colon \{1, \ldots, N + 2\} \to \{1, 2\}$

mit

$$s(i) = \begin{cases} 1 & \text{falls } 1 \leq i \leq N \text{ und } i \in M \\ 2 & \text{falls } 1 \leq i \leq N \text{ und } i \notin M \\ 1 & \text{falls } i = N+1 \\ 2 & \text{falls } i = N+2 \end{cases}$$

Nachrechnen ergibt, dass dieser Schedule einen Makespan von $2A$ hat.

Sei umgekehrt $(p_1, \ldots, p_{N+2}, 2, 2A) \in Makespan - Scheduling - E$. Dann existiert eine Zuordnung s, die die Jobs auf 2 Maschinen so anordnet, dass ein Makespan von höchstens $2A$ erreicht wird. Wie leicht einzusehen ist, haben beide Maschinen damit auch eine Last von exakt $2A$. Die Jobs p_{N+1} und p_{N+2} sind keinesfalls derselben Maschine zugeordnet. Eine gültige Lösung von Subset-Sum erhalten wir jetzt über $M = \{1 \leq i \leq N \mid s(i) = s(N+1)\}$.

Tutoriumsaufgabe 12.2

Zeigen Sie, dass BinPacking stark NP-schwer ist.

Wir zeigen, dass Three-Partition $\leq_p$ BinPacking, wobei sich in der Reduktion der maximale Wert von Zahlen nur um einen polynomiellen Faktor vergrößert.

Sei $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ und $c_1, \ldots, c_n$ die Eingabe für Three-Partition, sodass

$$\sum_{i=1}^{n} (a_i + b_i + c_i) = nS$$

gilt.

Wir konstruieren daraus folgende Instanz für BinPacking. Sei $A = \max_{1 \leq i \leq n}\{a_i, b_i, c_i\}$ der maximale Zahlenwert der gegebenen Three-Partition-Instanz. Wir setzen die Kapazität der Bins auf $b = S + 28A$, die maximale Anzahl der Bins auf $\gamma = n$ und konstruieren $3n$ Objekte mit Gewichten

$$16A + a_1, \ldots, 16A + a_n,\ 8A + b_1, \ldots, 8A + b_n,\ 4A + c_1, \ldots, 4A + c_n.$$

Offenbar können wir die neue Instanz in polynomieller Zeit berechnen. Außerdem ist der maximale Wert einer Zahl in der neuen BinPacking-Instanz polynomiell beschränkt in A.

Wir müssen also nur noch die Korrektheit zeigen. Zunächst sei (α, β) eine Lösung für die Three-Partition-Instanz, also $a_{\alpha(i)} + b_{\beta(i)} + c_i = S$ für alle $1 \leq i \leq n$. Für $1 \leq i \leq n$ befüllen wir nun den i-ten Bin mit den Gewichten $16A + a_{\alpha(i)}$, $8A + b_{\beta(i)}$ und $4A + c_i$. Dabei wird die Kapazität $b = S + 28A$ genau eingehalten.

Nehmen wir nun an, dass die konstruierte BinPacking-Instanz eine Lösung besitzt. Zunächst gilt, das kein Bin zwei Gewichte vom Typ $16A + a_i$ enthält, denn $16A + 16A = 32A > 28A + S = b$ (beachte, dass $S \leq 3A$). Also muss jeder Bin genau ein Objekt vom Typ $16A + a_i$ enthalten. Nun gilt aber, dass kein Bin zwei Gewichte vom Typ $8A + b_i$ enthält, denn $8A + 8A + 16A = 32A > 28A + S = b$ (beachte, dass jeder Bin bereits ein Gewicht von mindestens $16A$ enthält). Also muss jeder Bin genau ein Objekt vom Typ $8A + b_i$ enthalten. Mit einer weiteren Wiederholung des Arguments erhalten wir, dass kein

Bin zwei Gewichte vom Typ $4A+c_i$ enthält, denn $4A+4A+8A+16A = 32A > 28A+S = b$. Also muss jeder Bin genau ein Objekt vom Typ $4A + c_i$ enthalten. Wir nummerieren nun die Bins so, dass der i-te Bin gerade das Gewicht $4A + c_i$ enthält. Weiter definieren wir α und β, sodass der i-te die Gewichte $16A + a_{\alpha(i)}$ und $8A + b_{\beta(i)}$ enthält. Dann gilt $a_{\alpha(i)} + b_{\beta(i)} + c_i \leq S$ aufgrund der Kapazität der Bins. Da $\sum_{i=1}^{n}(a_i + b_i + c_i) = nS$ gilt, folgt weiter, dass $a_{\alpha(i)} + b_{\beta(i)} + c_i = S$ für alle $1 \leq i \leq n$.

Tutoriumsaufgabe 12.3

Wir betrachten sogenannte *primitive* Programme, die durch die Hintereinanderausführung von Zuweisungen "$x_i := x_j + c$" und If-Befehlen "IF ($x_i = c'$) THEN $x_j := x_k + c''$ ENDIF" (mit Konstanten $c, c', c'' \in \mathbb{N}$) entstehen. If-Befehle können nicht ineinander verschachtelt werden, und es gibt keine ELSE-Klauseln. Die Eingabe des Programms steht in den Variablen $x_1, \ldots, x_k$, und die Ausgabe des Programms steht am Ende in der Variable x_0. Die Variable x_0 enthält keinen Eingabewert und wird mit 0 initialisiert. Beweisen Sie, dass das folgende Problem coNP-schwer ist:

PrimitiveEq
Eingabe: Zwei primitive Programme P_1 und P_2 mit Variablen $x_0, \ldots, x_k$.
Frage: Berechnen diese beiden Programme dieselbe Funktion $f : \mathbb{N}^k \to \mathbb{N}$?

Wir zeigen UNSAT $\leq_p$ PrimitiveEq.

Aus einer CNF-Formel φ konstruieren wir ein primitives Programm P_φ, das φ auf der gegebenen Eingabe auswertet, wie folgt. Es seien $v_1, \ldots, v_n$ die Variablen von φ und $C_1, \ldots, C_m$ die Klauseln.

Das Programm P_φ verwendet die Variablen x_0 (für das Endergebnis), $x_1, \ldots, x_n$ (zur Eingabe der Variablenwerte), $y_1, \ldots, y_m$ (zum Speichern der Klauselzwischenergebnisse) und y (zum Zwischenspeichern des Endergebnisses). Um die Zuweisung von Konstanten zu simulieren, verwenden wir die Variable x_0.

Für jede Klausel C_i werden nun Befehle eingefügt, die diese auswerten und das Ergebnis in der Variable y_i speichern. Dies wird hier am Beispiel der Klausel $C_4 = v_1 \vee v_2 \vee \neg v_3$ beschrieben:

$y_4 := x_0 + 0$;
IF $(x_1 = 1)$ THEN $y_4 := x_0 + 1$ ENDIF;
IF $(x_2 = 1)$ THEN $y_4 := x_0 + 1$ ENDIF;
IF $(x_3 = 0)$ THEN $y_4 := x_0 + 1$ ENDIF

Hinter den Befehlen zum Auswerten der Klauseln folgen nun Befehle, um die gesamte Formel auszuwerten. Dazu gehen wir wie folgt vor:

$y := x_0 + 1$;
IF $(y_1 = 0)$ THEN $y := x_0 + 0$ ENDIF;
IF $(y_2 = 0)$ THEN $y := x_0 + 0$ ENDIF;
$\vdots$

IF $(y_m = 0)$ THEN $y \coloneqq x_0 + 0$ ENDIF;
$x_0 \coloneqq y + 0$

Die neue Instanz für PrimitiveEq ist jetzt (P_φ, P_0), wobei P_0 ein Programm ist, das die gleichen Variablen wie P_φ verwendet, aber immer 0 als Ergebnis liefert. Dieses Programmpaar ist offensichtlich in polynomieller Zeit berechenbar.

Korrektheit: Zeige, dass φ unerfüllbar ist genau dann, wenn die beiden Programme dieselbe Funktion berechnen. Angenommen, φ ist erfüllbar. Für eine erfüllende Belegung $x_1, \ldots, x_n$ berechnet P_φ auf der Eingabe $(x_1, \ldots, x_n, 0, \ldots, 0)$ eine 1, weshalb die Programme P_φ und P_0 nicht dieselbe Funktion berechnen. Umgekehrt sei φ so, dass die Programme P_φ und P_0 nicht dieselbe Funktion berechnen. Dann existiert eine Eingabe $(x_1, \ldots, x_n, y_1, \ldots, y_m, y)$, auf der von P_φ eine 1 berechnet wird; andere Werte kann das Programm nicht ausgeben. In diesem Fall enthielten also alle Variablen $y_1, \ldots, y_m$ den Wert 1. Dies bedeutet, das für jede Klausel eine der Eingaben $x_1, \ldots, x_n$ auf 0 oder 1 gesetzt ist und zwar so, dass diese Klausel erfüllt ist. Indem man nun für die Werte der $x_1, \ldots, x_n$, die ungleich 0 und 1 sind, beliebig 0 oder 1 wählt, erhält man eine erfüllende Belegung für φ.