Algorithmen und Datenstrukturen 2

ALGO2 · WiSe-2021/22 · tcs.uni-frankfurt.de/algo2 · 2021-11-30 · *412c9c1*

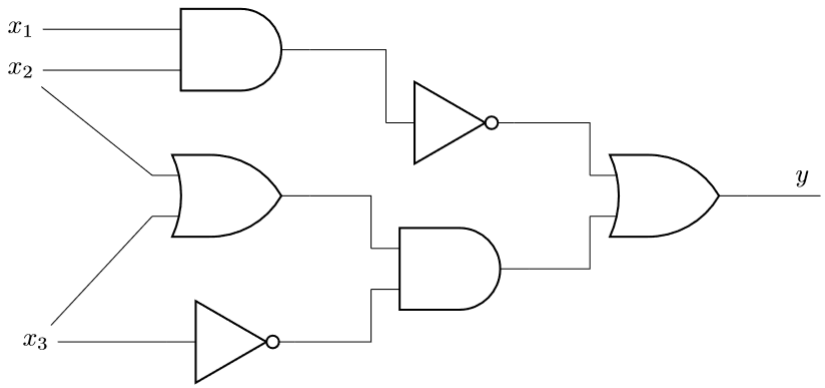
Übungen zu Woche 7: Hartnäckigkeit I

Das Übungsblatt enthält alle empfohlenen Lernaktivitäten für die aktuelle Woche.

- **Heimarbeit bis Montag 17:00.**
  - Schau die Videos an und lies die Buchkapitel.
  - Bearbeite die 🌱-Aufgabe in Moodle. (Feste Abgabefrist!)
  - Lese den Aufgabentext aller Übungsaufgaben.
- **Heimarbeit.** Bearbeite die Übungsaufgaben soweit möglich. Probier zumindest alle mal!
- **Dienstag/Donnerstag.**
  - **8:00–8:15.** Besprechung im Hörsaal.
  - **8:15–9:15.** Bearbeite jetzt die Übungen, die du noch nicht lösen konntest. Sprich mit anderen Studis! Frag das Vorlesungsteam um Hilfe!
  - **9:15–9:45.** Lösungsspaziergang zu den Aufgaben für heute.
- **Heimarbeit bis Freitag, den 03.12., 17:00.** Gib deine Lösungen zu der ⭐-Aufgabe von diesem Übungsblatt in Moodle ab. (Feste Abgabefrist!)

Dienstag

Aufgabe 7.1 (Schaltkreise). In der Vorlesung haben wir eine Reduktion von CircuitSat auf 3SAT gesehen. Gegeben sei folgender Schaltkreis:



Welche 3CNF Formel gibt die Reduktion aus, wenn die Eingabe aus dem dargestellten Schaltkreis besteht?

Aufgabe 7.2 (DNF Erfüllbarkeit). Eine aussagenlogische Formel ist in *disjunktiver Normalform* (DNF), wenn sie eine Disjunktion (OR) von Konjunktionstermen (AND) ist. Ein Beispiel für eine DNF ist:

$$(\overline{x} \wedge y \wedge \overline{z}) \vee (y \wedge z) \vee (x \wedge \overline{y} \wedge \overline{z}).$$

Gegeben ist eine aussagenlogische Formel in disjunktiver Normalform. DNF-SAT entscheidet, ob diese Funktion erfüllbar ist.

a) Beschreibe einen Algorithmus, der DNF-SAT in Polynomialzeit löst.

b) Was ist der Fehler im folgenden Argument, in dem P = NP gezeigt wird?

> Angenommen, wir haben eine aussagenlogische Formel in konjunktiver Normalform mit höchstens 3 Literalen pro Klausel. Wir wollen herausfinden, ob diese Funktion erfüllbar ist. Wir können das Distributivgesetz für Boolesche Operationen verwenden, um eine äquivalente Formel in disjunktiver Normalform zu konstruieren. Zum Beispiel:
>
> $$(x \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y}) \Leftrightarrow (x \wedge \overline{y}) \vee (y \wedge \overline{x}) \vee (\overline{z} \wedge \overline{x}) \vee (\overline{z} \wedge \overline{y}).$$
>
> Nun können wir den Algorithmus aus Aufgabenteil a) verwenden, um in Polynomialzeit herauszufinden, ob die entstandene DNF erfüllbar ist. Wir haben also 3SAT in Polynomialzeit gelöst! Da 3SAT NP-hart ist, gilt P = NP!

Aufgabe 7.3 (Perfektes Matching). Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $M \subseteq E$ ist ein *Matching*, wenn keine zwei Kanten aus M mit demselben Knoten verbunden sind. Ein Matching ist *perfekt*, wenn jeder Knoten $v \in V$ zu einer Kante $m \in M$ inzident ist. Eine andere mögliche Definition betrachtet die Größe des Matchings: M ist perfekt, wenn $|M| = |V|/2$ gilt. Wir definieren das PerfektesMatching-Problem:

> PerfektesMatching:
> Gegeben sei ein ungerichteter Graph $G = (V, E)$. Existiert in G ein perfektes Matching?

Dieses Problem kann in Polynomialzeit gelöst werden. Das ist ein fundamentales Ergebnis der kombinatorischen Optimisierung mit vielen Anwendungen in Theorie und Praxis. Es stellt sich heraus, dass das PerfektesMatching-Problem auf bipartiten Graphen einfacher zu lösen ist. Ein Graph $G = (V, E)$ ist *bipartit*, wenn die Menge der Knoten V auf zwei Teilmengen L und R verteilt werden können, sodass alle Kanten zwischen L und R liegen. In anderen Worten: L und R sind unabhängige Knotenmengen (Independent Set). Im Folgenden ist ein Algorithmus vorgeschlagen, der allgemeine Graphen auf bipartite Graphen reduziert:

> ReduceGraph:
> Gegeben sei ein ungerichteter Graph $G = (V, E)$. Erstelle daraus einen bipartiten Graphen $H = (V \times \{1, 2\}, E_H)$ wie folgt:
> Jeder Knoten $u \in V$ wird zu zwei Kopien $(u, 1)$ und $(u, 2)$. Dabei ist
>
> $$V_1 = \{(u, 1) \mid u \in V\}$$
>
> eine Seite und
>
> $$V_2 = \{(u, 2) \mid u \in V\}$$
>
> die andere Seite des bipartiten Graphen. Sei
>
> $$E_H = \{\{(u, 1), (v, 2)\} \mid \{u, v\} \in E\}.$$
>
> In anderen Worten: Wenn $\{u, v\} \in E$, fügen wir eine Kante zwischen $(u, 1)$ und $(v, 2)$ ein. Beachte, dass es keine Eigenschleifen in G gibt und es dadurch in H für alle $u \in V$ keine Kante $\{(u, 1), (u, 2)\}$ geben kann.

Ist die vorgeschlagene Reduktion korrekt? Um das herauszufinden müssen wir überprüfen, ob H nur genau dann ein perfektes Matching hat, wenn G eins hat.

a) Zeige: Wenn G ein perfektes Matching hat, dann hat auch H eins.
b) Sei G der vollständige Graph mit 3 Knoten, also ein Dreieck. Zeige: G hat kein perfektes Matching, aber H hat eins.
c) Erweitere das Beispiel aus der vorherigen Teilaufgabe: Konstruiere einen Graphen G mit einer geraden Anzahl an Knoten, sodass G kein perfektes Matching hat, H aber schon.

Damit ist die Reduktion inkorrekt, obwohl eine Richtung wahr ist.

Aufgabe 7.4 (Independent Set). Ein *Independent Set* in einem Graphen $G = (V, E)$ ist eine Teilmenge $S \subseteq V$, sodass keine zwei Knoten $u, v \in S$ durch eine Kante $\{u, v\} \in E$ verbunden sind. Du hast eine magische Blackbox, die auf wundersame Art und Weise das folgende Entscheidungsproblem in Polynomialzeit beantwortet:

- Input: Ein ungerichteter Graph G und eine Zahl $k \in \mathbb{N}$.
- Output: True, wenn G ein Independent Set der Größe k besitzt, sonst False.

a) Benutze die Blackbox, um einen Algorithmus zu beschreiben, der folgendes Optimisierungsproblem in Polynomialzeit löst:

   - Input: Ein ungerichteter Graph G.
   - Output: Die Größe des größten Independent Set in G.

b) Benutze die Blackbox, um einen Algorithmus zu beschreiben, der folgendes Suchproblem in Polynomialzeit löst:

   - Input: Ein ungerichteter Graph G.
   - Output: Ein Independent Set in G mit maximaler Größe.

Donnerstag

Aufgabe 7.5 (NP). Entscheide für die beiden folgenden Fragen, welche der Antworten „Ja“, „Nein“ oder „Unbekannt, da es P = NP beantworten würde“ zutrifft. Erkläre kurz, warum du dich für deine Antwort entschieden hast.

> IntervalScheduling:
> Sei eine Menge von Intervallen auf einer Zeitleiste und eine Schranke k gegeben. Enthält diese Menge an Intervallen eine Teilmenge von sich nicht überschneidenden Intervallen der Größe mindestens k?

Im Folgenden schreiben wir $A \leq_p B$, wenn es eine Polynomialzeit-Reduktion gibt, die das Problem A auf das Problem B reduziert. Das heißt, das Problem A soll in Polynomialzeit gelöst werden, falls es einen magischen Algorithmus gibt, der B in Polynomialzeit löst.

a) Gilt IntervalScheduling $\leq_p$ VertexCover?
b) Gilt IndependentSet $\leq_p$ IntervalScheduling?

Aufgabe 7.6 (Kundenanalyse). Zur Analyse des Kundenverhaltens benutzen Geschäfte oftmals ein zweidimensionales Array A, wo die Zeilen die Kunden und die Spalten die verkauften Produkte enthalten. Ein Eintrag $A[i, j]$ gibt an, welche Menge von Produkt j der Kunde i gekauft hat, so hat Chelsea im folgenden Beispiel siebenmal Katzenstreu gekauft.

	Waschmittel	Bier	Windeln	Katzenstreu
Raj	0	6	0	3
Alanis	2	3	0	0
Chelsea	0	0	0	7

Ein Geschäft möchte nun eine *diverse* Teilmenge der Kunden finden, weil das nützlich sein kann, um Marktforschung zu betreiben. Eine Teilmenge S der Kunden heißt *divers*, wenn keine zwei Kunden aus S dieselben Produkte gekauft haben. Mit anderen Worten gibt es für jedes Produkt nur genau einen Kunden aus S, der es gekauft hat. Das lässt sich als das Problem DiverseTeilmenge formulieren.

> DiverseTeilmenge:
> Seien ein $n \times m$ Array A, wie oben definiert, und eine Zahl k mit $k \leq n$ gegeben.
> Gibt es eine *diverse* Teilmenge S der Kunden mit $|S| \geq k$?

Zeige, dass DiverseTeilmenge NP-vollständig ist.

Sternaufgabe

Aufgabe 7.7 (⭐: Search-to-Decision).

a) Du besitzt eine magische Blackbox, die in Polynomialzeit herausfindet, wie viele Knoten ein größter vollständiger Teilgraph ein beliebiger Graph G besitzt. Beschreibe und analysiere einen Algorithmus, der in Polynomialzeit für einen beliebigen Graphen G einen vollständigen Teilgraphen maximaler Größe berechnet. Benutze dafür die Blackbox.

b) Ein ungerichteter Graph G heißt 3-färbbar, wenn es eine Funktion $c : V(G) \to \{\mathtt{R}, \mathtt{G}, \mathtt{B}\}$ gibt, die jedem Knoten eine von drei Farben zuordnet, sodass benachbarte Knoten unterschiedliche Farben erhalten (siehe erster Absatz in Erickson 12.10).
Du besitzt eine magische Blackbox, die in Polynomialzeit herausfindet, ob ein beliebiger Graph G 3-färbbar ist. Beschreibe und analysiere einen Algorithmus, der in Polynomialzeit für einen beliebigen Graphen G eine richtige 3-Färbung ausgibt oder richtigerweise ausgibt, dass keine solche Färbung existiert. Benutze dafür die Blackbox.
Tipp: Die Eingabe für die Blackbox ist ein Graph und nichts Anderes.

c) Du besitzt eine magische Blackbox, die in Polynomialzeit herausfindet, ob eine beliebige Boolesche Formel Φ erfüllbar ist (zum Beispiel $\Phi = (x \Leftrightarrow ((z \wedge y) \vee \overline{x})) \wedge (y \Rightarrow \overline{x})$). Beschreibe und analysiere einen Algorithmus, der in Polynomialzeit eine erfüllende Belegung der Variablen von Φ ausgibt, oder richtigerweise ausgibt, dass so eine Belegung nicht existiert. Benutze dafür die Blackbox.