Technische Universität Darmstadt
Fachgebiet Theoretische Informatik

Prof. Johannes Buchmann
Nabil Alkeilani Alkadri
Nina Bindel
Patrick Struck

# Algorithmen und Datenstrukturen

SoSe 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

**13. Lösungsblatt** — 09.07.2018
v1.0

## P1 Allgemeines

In dieser Aufgabe befassen wir uns allgemein mit Komplexitätsklassen.

a) Erklären Sie diesbezüglich kurz, aber präzise, folgende Begriffe.

   1. $P$

   2. $NP$; in welcher Relation steht $P$ zu $NP$?

   3. $NP$-schwer; geben Sie ein Beispiel für ein Problem welches $NP$-schwer ist.

   4. $NP$-vollständig; geben Sie ein Beispiel für ein Problem welches $NP$-vollständig ist.

b) Eine berühmte bisher ungelöste Frage ist, ob $P = NP$ (bzw. $P \neq NP$) gilt. Erklären Sie kurz die praktische Relevanz eines potentiellen Beweises dieser Aussage.

**Lösung.**

a) Die Begriffe sind wie folgt definiert:

   1. $P$ ist die Klasse jener Entscheidungsprobleme, die in polynomieller Laufzeit lösbar sind.

   2. $NP$ ist die Klasse jener Entscheidungsprobleme, die in polynomieller Laufzeit verifizierbar sind. Das heißt, für Probleme aus NP ist es in polynomieller Laufzeit möglich zu überprüfen, ob eine vorgeschlagene Lösung richtig ist oder nicht. Es gilt $P \subseteq NP$, da jedes Problem, dass in Polynomialzeit lösbar ist, auch in Polynomialzeit verifizierbar ist.

   3. Ein $NP$-schweres Problem ist mindestens so schwer wie alle Probleme in $NP$. Ein Algorithmus, der eine $NP$-schweres Problem löst, löst mithilfe einer Reduktion alle Probleme in $NP$. Beispiel: 3-COLORS (s. Aufgabe P2)

   4. Ein Problem ist $NP$-vollständig, wenn es in $NP$ ist und $NP$-schwer ist. Beispiel: 3-COLORS (s. Aufgabe P2)

b) Dieses Problem gilt als eines der wichtigsten ungelösten Probleme der Informatik. Viele, vor allem graphentheoretische Probleme, wären optimal in kurzer Zeit lösbar. Die meinsten Wissenschaftler und Wissenschaftlerinnen gehen davon aus, dass $P \neq NP$ gilt. Ein Beweis wurde jedoch nicht erbracht. Würde jedoch $P = NP$ bewiesen werden, hätte es enorme Konsequenzen, beispiels in der Kryptographie. Die sogenannte asymmentrische Kryptografie basiert darauf, dass Lösungen in Polynomialzeit verifizierbar sind, jedoch nicht in Polynomialzeit gelöst werden können, wie z.B. beim Problem der Primzahlfaktorisierung. In der asymmetrischen Kryptografie gibt es einen öffentlichen Schlüssel welcher zum Vertschlüsseln genutzt wird und einen geheimen Schlüssel, welcher z.B. zum Entschlüsseln genutzt wird. Damit ist es möglich, dass jeder eine Nachricht verschlüsselt, jedoch nur der/die Besitzer*in des geheimen Schlüssels den verschlüsselten Text wieder entschlüsseln kann. Würde es einen Algorithmus geben um jedes $NP$-Problem in Polynomialzeit zu lösen (also z.B. die Primfaktoren jeder beliebigen Zahl in Polynomialzeit zu finden), so könnte bspw. der geheime Schlüssel berechnete werden und eine verschlüsselte Nachricht entschlüsselt werden. Die Verfahren wäre als nicht mehr sicher.

## P2 k-COLOR

Gegeben sei ein ungerichteter Graph $G = (V, E)$. In dieser Aufgabe suchen wir eine Färbung der Knoten $V$, sodass zwei Knoten, die mit einer Kante aus $E$ verbunden sind, unterschiedlich gefärbt sind. Zur Verfügung stehen nur $k$ unterschiedliche Farben.

Beweisen Sie Folgendes: Angenommen, der 3-COLOR-Algorithmus ($k = 3$) ist $NP$-vollständig. Dann ist auch der 4-COLOR-Algorithmus ($k = 4$) $NP$-vollständig ist. Aufgrund eines in der Vorlesung bewiesenen Satzes, genügt es zu zeigen, dass

a) 4-COLOR $\in NP$ und

b) 4-COLOR ist $NP$-schwer.

*Hinweis:* Nutzen Sie für Teil (b), ähnlich der Vorlesung, als Beweismethoden einen Reduktionsbweis. Führen Sie also ein Problem auf ein anderes zurück. Dies impliziert, dass wenn es einen Algorithmus für das zweite Problem gibt, das erste auch über die Reduktion lösbar ist.

**Lösung.**

a) 4-COLOR $\in NP$ gilt, da beim (nicht-deterministischen) Raten einer Färbung der Knoten, die Korrektheit dieser Färbung polynomiell verifizierbar ist (konkret in $O(|E|)$ Schritten). Dies geschieht durch das Iteratieren über alle Kanten und der Verifikation der Färbung verbundener Knoten.

b) 4-COLOR ist $NP$-schwer, da wir 3-COLOR auf 4-COLOR reduzieren können. Sei $I$ eine Instanz von 3-COLOR mit $G = (V, E)$. Wir fügen einen Knoten hinzu, der nun gezwungenermaßen eine der vier Farben von 4-COLOR verbraucht. Dazu verbinden wir ihn mit allen Knoten in $V$. Diese 4-COLOR Instanz $I'$ ist also $G' = (V \cup \{v'\}, E \cup \{(v, v') | v \in V\})$. Ist nun $G$ mit drei Farben färbbar, belassen wir die Färbung der Knoten von $G$ in $G'$ und färben $v'$ in der vierten Farbe, wodurch $G'$ also vierfärbbar ist. Ist $G'$ vierfärbbar, so hat kein anderer Knoten dieselbe Farbe wie $v'$, da $v'$ mit allen anderen Knoten verbunden ist. Durch Entfernen von $v'$ aus $G'$ erhält man eine korrekte Färbung mit drei Farben von G. Die Reduktion ist somit gezeigt.

## P3 Greedy-Algorithmus / Optimierung

Gegeben sei die Funktion $\varphi(x, y) = (x-3)^2 + 2y^2$, die wie leicht nachzurechnen das globale Minimum bei $\boldsymbol{p_{min}} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$ hat. Um dieses Minimum zu finden verwenden wir in dieser Aufgabe die Methode des steilsten Abstiegs. Diese berechnet iterativ:

$$\boldsymbol{p_{k+1}} = \boldsymbol{p_k} + \alpha_k \boldsymbol{d_k}$$

wobei in jeder Iteration gilt:

$$\begin{aligned} \boldsymbol{d_k} &= -\nabla\varphi(\boldsymbol{p_k}) = -\begin{pmatrix} \frac{\partial\varphi(x,y)}{\partial x} \\ \frac{\partial\varphi(x,y)}{\partial y} \end{pmatrix} \\ \alpha_k \text{ Minimum von } \varphi(\alpha) &= \varphi(\boldsymbol{p_k} + \alpha \boldsymbol{d_k}) \end{aligned}$$

a) Führen Sie, ausgehend vom Startpunkt $\boldsymbol{p_0} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, zwei Iterationsschritte der Methode des steilsten Abstiegs aus.
*Hinweis:* Nach zwei Iterationen erhalten Sie noch nicht das globale Minimum $\boldsymbol{p_{min}}$.

b) Wieso handelt es sich bei der Methode des steilsten Abstiegs um einen Greedy-Algorithmus?

**Lösung.**

a) Der Gradient ergibt sich zu:

$$\nabla\varphi(x,y) \quad = \quad \begin{pmatrix} 2(x-3) \\ 4y \end{pmatrix}$$

Wir berechnen nun $\boldsymbol{p_1}$:

Einsetzen von $\boldsymbol{p_0} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ liefert die Abstiegsrichtung:

$$\boldsymbol{d_0} = -\nabla\varphi(\boldsymbol{p_0}) \quad = \quad -\begin{pmatrix} 2(1-3) \\ 4 \cdot 1 \end{pmatrix} = \begin{pmatrix} 4 \\ -4 \end{pmatrix}.$$

Wir berechnen nun $\alpha_0$ als Minimum der Funktion $\varphi(\alpha) = \boldsymbol{p_0} + \alpha\boldsymbol{d_0}$:

$$\boldsymbol{p_0} + \alpha\boldsymbol{d_0} = \begin{pmatrix} 1+4\alpha \\ 1-4\alpha \end{pmatrix}$$
$$\Rightarrow \quad \varphi(\alpha) = (1+4\alpha-3)^2 + 2(1-4\alpha)^2.$$

Durch Vereinfachen von $\varphi(\alpha)$ sowie der ersten und zweiten Ableitung, erhalten wir das Minimum an der Stelle $\alpha = \frac{1}{3}$.

Somit erhalten wir:

$$\boldsymbol{p_1} \quad = \quad \boldsymbol{p_0} + \alpha_0\boldsymbol{d_0} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \frac{1}{3}\begin{pmatrix} 4 \\ -4 \end{pmatrix} = \begin{pmatrix} \frac{7}{3} \\ -\frac{1}{3}. \end{pmatrix}$$

Um $\boldsymbol{p_2}$ zu berechnen, führen wir dieselbem Schritte noch einmal aus:

$$\begin{aligned} \boldsymbol{p_1} &= \begin{pmatrix} \frac{7}{3} \\ -\frac{1}{3} \end{pmatrix} \\ \boldsymbol{d_1} = -\nabla\varphi\boldsymbol{p_1} &= \begin{pmatrix} \frac{4}{3} \\ \frac{4}{3} \end{pmatrix} \\ f(\alpha) &= \left(\frac{7}{3} + \frac{4}{3}\alpha - 3\right)^2 + 2\left(-\frac{1}{3} + \frac{4}{3}\alpha\right)^2 \\ \Rightarrow \alpha &= \frac{1}{3}. \end{aligned}$$

Somit erhalten wir

$$\boldsymbol{p_2} = \boldsymbol{p_1} + \alpha_1\boldsymbol{d_1} = \begin{pmatrix} \frac{7}{3} \\ -\frac{1}{3} \end{pmatrix} + \frac{1}{3}\begin{pmatrix} \frac{4}{3} \\ \frac{4}{3} \end{pmatrix} = \begin{pmatrix} \frac{25}{9} \\ \frac{1}{9} \end{pmatrix} \approx \begin{pmatrix} 2,778 \\ 0,110 \end{pmatrix}$$

Wir erhalten somit nach zwei Iterationen nicht das globale Minimum $\boldsymbol{p_{min}} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$ aber dennoch eine gute Approximation.

b) Ein Greedy-Algorithmus ist dadurch charakterisiert, dass er nach jedem Iterationsschritt das derzeitige, optimale Ergebnis (gemäß Invariante) zurückgibt. Oder anders gesagt: Das Prinzip eines Greedy-Algorithmus (gieriger Algorithmus) ist es, in jedem Teilschritt so viel wie möglich zu erreichen. Wir betrachten sogenannte gierige Algorithmen zur schrittweisen Berechnung optimaler Elemente aus einer bestimmten Grundmenge. Das Prinzip ist in jedem Schritt das optimale Ergebnis (gemäß Invariante) zurück zu geben. Bei günstiger Aufgabenstruktur entsteht so ein globales Optimum.

Bei der Methode des steilsten Abstiegs handelt es sich daher um einen Greedy-Algorithmus, da in jeder Iteration entlang der maximalen Abstiegsrichtung ($\boldsymbol{d_k}$) mit einer maximalen Schrittweite ($\alpha_k$) gesucht wird. In jeder Iteration $k$ erhalten wir somit mit $\boldsymbol{p_k}$ ein lokales Minimum und somit liefert der Algorithmus–per Definition "gierig"–stets das aktuelle Optimum bezüglich der Problemstellung.

## H1 Euler-Tour und Euler-Pfade

Das Problem der Euler-Tour ist eng mit dem aus der Vorlesung bekannten Problem der Königsberger Brücken verwandt: Gegeben sei ein zusammenhängender, gerichteter Graph $G = (V, E)$. Eine Euler-Tour ist ein Zykel in $G$, der jede Kante $e \in E$ genau einmal durchläuft. Ein Euler-Pfad ist ein Pfad im Graph, der alle Kanten genau einmal durchläuft. Er muss nicht zwangsläufig zum Ausgangspunkt zurückkehren.
Sei $G = (V, E)$ ein gerichteter, zusammenhängender Graph. Dann gilt:

a) Wenn $G$ eine Euler-Tour enthält, so gilt für alle Knoten $v \in V$ Eingangsgrad($v$)=Ausgangsgrad($v$).
*Hinweis:* Eine Euler-Tour kann als Vereinigung einfacher kantendisjunkter Zyklen aufgefasst werden , d.h. jeder Zykel durchläuft einen Knoten höchstens einmal und jede Kante kommt in höchstens einem Zykel vor.

b) Wenn $G$ einen Euler-Pfad enthält, so gilt entweder Eingangsgrad($v$)=Ausgangsgrad($v$) für alle Knoten $v \in V$ oder es gibt genau zwei Knoten, für die dies nicht gilt.

*Bemerkung*: In beiden Fällen gilt auch die Umkehrung (der Beweis ist etwas kompliziert). Es gibt auch einen Algorithmus, der auf jedem Graphen mit den obigen Eigenschaften eine Euler-Tour bzw. einen Euler-Pfad findet. Die Laufzeit dieses Algorithmus liegt in $O(|E|)$. Beide Probleme sind also in polynomieller Zeit lösbar.

**Lösung.**

a) Eine Euler-Tour kann einen Knoten $v \in V$ mehrmals durchlaufen. Jedoch kann eine Euler-Tour in eine Reihe von einfachen Zykeln $z_1, \ldots, z_l$ zerlegt werden, für die gilt:

- Jeder Zykel $z_i$ durchläuft einen Knoten $v \in V$ höchstens einmal.
- Jede Kante $e \in E$ ist in genau einem Zykel $z_i$ enthalten.

"⇒": Wir nehmen nun an, dass auf dem Graphen $G$ eine Euler-Tour $E$ existiert. Wir zerlegen diese wie oben beschrieben in eine Reihe einfacher Zykel $z_1, \ldots, z_l$. In jedem der einfachen Zyklen $z_i$ gilt für alle Knoten $v \in V$ Eingangsgrad($v$)=Ausgangsgrad($v$). Genauer gesagt gilt

- Eingangsgrad($v$)=Ausgangsgrad($v$)=1 $\Leftrightarrow$ der Zykel $z_i$ durchläuft $v$.
- Eingangsgrad($v$)=Ausgangsgrad($v$)=0 $\Leftrightarrow$ der Zykel $z_i$ durchläuft $v$ nicht.

Durch Aufsummieren über alle Teilzykel erhält man die Behauptung.

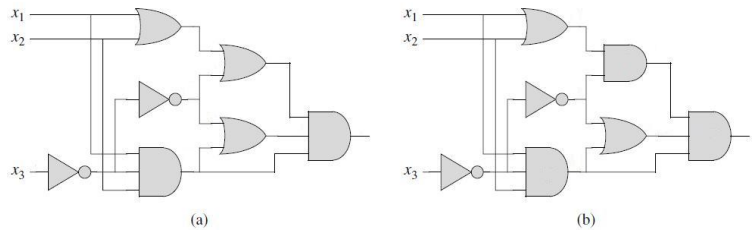
b) Sei nun ein Eulerpfad $W$ in $G$ gegeben. Wir unterscheiden zwei Fälle:

(1) $W$ ist eine Euler-Tour, d.h. Anfangs- und Endpunkt stimmen überein. In diesem Fall gilt nach Teil a) der Aufgabe Eingangsgrad($v$)=Ausgangsgrad($v$) für alle $v \in V$.

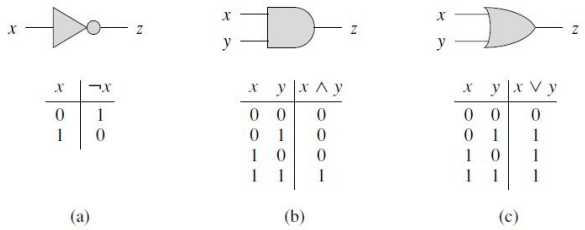
(2) Anfangs- und Endpunkt stimmen nicht überein. Sei $W = (v_0, v_1, \ldots, v_{n-1})$ ein Eulerpfad in $G$ mit $v_0 \neq v_{n-1}$. $W$ durchläuft nach Definition alle Kanten in $E$. Wir bilden einen neuen Graphen $G'$ unter Hinzunahme der Kante $(v_{n-1}, v_0)$. Damit ist $W' = (v_0, v_1, \ldots, v_{n-1}, v_0)$ eine Euler-Tour in $G'$ und es gilt Eingangsgrad($v$) = Ausgangsgrad($v$) für alle Knoten $v \in V$ (für den Graphen $G'$). Im Graphen $G$ haben alle Knoten außer $v_0$ und $v_{n-1}$ den gleichen Eingangs- und Ausgangsgrad wie in $G'$, für $v_0$ ist der Eingangs-, für $v_{n-1}$ der Ausgangsgrad um 1 niedriger.

## H2 Schaltkreis-SAT

Gegeben seien die folgenden beiden Schaltkreise.




Dabei werden die folgenden logischen Gatter verwendet.




| $x$ | $\neg x$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

(a)

| $x$ | $y$ | $x \wedge y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(b)

| $x$ | $y$ | $x \vee y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

(c)

Das NICHT-Gatter (a) gibt genau dann wahr zurück, wenn sein Eingabewert falsch ist. Das UND-Gatter (b) gibt wahr zurück, wenn alle seine Eingabewerte wahr sind. Das ODER-Gatter (c) gibt wahr zurück, wenn mindestens einer seiner Eingabewerte wahr ist. UND- und ODER-Gatter können mehr als zwei Eingabewerte haben.

a) Finden Sie eine Lösung des Schaltkreises a).

b) Zeigen Sie, dass Schaltkreis b) keine Lösung hat.

**Lösung.**

a) Die Besetzung $x_1 = x_2 =$ wahr, $x_3 =$ falsch liefert eine Lösung des Schaltkreises a).

b) Die einfachste Lösung wäre, sämtliche Besetzungen von $x_1, x_2$ und $x_3$ durchzuprobieren. Dafür müssten wir $2^3 = 8$ Möglichkeiten durchprobieren.
Wir leiten statt dessen einen Widerspruch her. Angenommen, das UND-Gatter C gibt den Wert wahr zurück. Dann muss jeder der logischen Ausdrücke $u$, $v$ und $t$ wahr sein. Ausdruck $u$ ist wahr, wenn sowohl $r$ als auch $s$ wahr sind (Gatter A). $s$ ist genau dann wahr, wenn $x_3$ wahr ist (zwei NICHT-Gatter). Dagegen kann $t$ nur dann wahr sein, wenn $x_3$ falsch ist (UND-Gatter $B$). $x_3$ müsste also gleichzeitig wahr und falsch sein. Widerspruch.