



Präsenzaufgaben zur Vorlesung »Analyse von Algorithmen und Komplexität«

Blatt 9

Definition 1 (Knapsack (Entscheidungsvariante)). Gegeben ist eine Menge mit n Gegenständen mit Gewichten $w_1, \dots, w_n \in \mathbb{N}_{\geq 0}$ und Profiten $p_1, \dots, p_n \in \mathbb{N}_{\geq 0}$. Weiter sind eine Kapazität $K \in \mathbb{N}_{\geq 0}$ und ein Zielprofit $P \in \mathbb{N}_{\geq 0}$ gegeben. Bei dem KNAPSACK Problem muss entschieden werden, ob eine Teilmenge $S \subseteq [n]$ existiert, sodass

$$\sum_{i \in S} w_i \leq K \quad \text{und} \quad \sum_{i \in S} p_i \geq P$$

gilt.

Präsenzaufgabe 9.1 (KNAPSACK in NP)

Zeigen Sie $\text{KNAPSACK} \in \mathbf{NP}$ auf zwei verschiedene Wege.

1. Geben Sie einen nicht deterministischen Algorithmus an, der das Problem in polynomieller Zeit löst.
2. Geben Sie einen polynomiellen Verifizierer für das Problem an.

Wie hängen diese beiden Methoden zusammen?

Lösung. 1. Der Algorithmus arbeitet wie folgt: Für jeden Gegenstand entscheiden wir nicht-deterministisch, ob er in den Rucksack soll. Anschließend akzeptieren wir, falls die Kapazitäts- und Profitbedingung erfüllt sind.

Existiert eine valide Rucksackfüllung mit genügend Profit, dann gibt es auch eine Reihe von nichtdeterministischen Entscheidungen, die diese Füllung findet. Diese wird dann auch vom Algorithmus akzeptiert.

Der Algorithmus macht polynomiell viele nichtdeterministische Entscheidungen und die Überprüfung der Eigenschaften ist ebenfalls in Polynomialzeit möglich, also ist dies ein polynomieller, nichtdeterministischer Algorithmus, der KNAPSACK löst.

2. Unser Zertifikat ist eine Teilmenge $S \subseteq [n]$, die die Auswahl für den Rucksack darstellen soll. Dies kann zum Beispiel über ein Bit pro Gegenstand realisiert werden. Um dieses Zertifikat zu verifizieren, berechnen wir $\sum_{i \in S} w_i$ und $\sum_{i \in S} p_i$ und vergleichen die Werte mit K und P und akzeptieren entsprechend.

Existiert eine valide Rucksackfüllung mit genügend Profit, dann gibt es auch ein Zertifikat, das diese Füllung beschreibt. Dieses wird dann auch vom Verifizierer akzeptiert.

Das Ausrechnen der Summen und das Überprüfen ist in polynomieller Zeit möglich, also ist dies ein polynomieller Verifizierer.

In beiden Wegen wird die extra Ressource, die wir zur Verfügung haben, genutzt, um eine Teilmenge (einen möglichen Rucksack) zu bestimmen und zu überprüfen. In der Vorlesung wird noch genau gezeigt, dass diese beiden Wege tatsächlich äquivalent sind. $\square$

Präsenzaufgabe 9.2 (Transitivität von Polynomialzeitreduktionen)

Zeigen Sie, dass Polynomialzeitreduktionen transitiv sind, d.h., dass für alle Entscheidungsprobleme $L_1, L_2, L_3 \subseteq \Sigma^*$ gilt, dass

$$L_1 \preceq L_2 \wedge L_2 \preceq L_3 \implies L_1 \preceq L_3.$$

Lösung. Seien L_1, L_2, L_3 Sprachen. Zudem sei R_1 ein Algorithmus, der L_1 nach L_2 in $\mathcal{O}(n^a)$ reduziert, und R_2 ein Algorithmus, der L_2 nach L_3 in $\mathcal{O}(n^b)$ reduziert.

Sei x eine Instanz von L_1 der Größe n . Dann benötigt der Aufruf von R_1 auf x höchstens $\mathcal{O}(n^a)$ Zeit.

Sei x' die Ausgabe von R_1 . Dann hat x' eine Größe von höchstens $\mathcal{O}(n^a)$. Diese Schranke für die Ausgabe ergibt sich aus der Laufzeit der Reduktion. Der Aufruf von R_2 auf x' benötigt nun höchstens $\mathcal{O}((n^a)^b) = \mathcal{O}(n^{ab})$ Zeit. Damit ist die Reduktion von L_1 auf L_3 in polynomieller Zeit möglich.

Die Korrektheit ergibt sich aus der Transitivität der Äquivalenz: Sei x'' die Ausgabe von R_2 auf x' , dann gilt

$$x \in L_1 \iff x' \in L_2 \text{ und } x' \in L_2 \iff x'' \in L_3,$$

also auch $x \in L_1 \iff x'' \in L_3$. $\square$



Präsenzaufgaben zur Vorlesung »Analyse von Algorithmen und Komplexität«

Blatt 10

Präsenzaufgabe 10.1 (Fragen über Fragen)

Diskutieren Sie die folgenden Aussagen (Gelten sie allgemein? Gelten sie nicht? Gelten sie unter bestimmten Bedingungen? Welche?):

- (i) Für jede Sprache, die von einer NDTM in polynomieller Zeit akzeptiert werden kann, existiert eine DTM, die ebenfalls die Sprache in polynomieller Zeit akzeptiert.
- (ii) A ist NP -schwer $\implies A \in P$

Lösung. (i) Die Aussage gilt nur, wenn $P = NP$. Dann liegen alle Probleme in NP (von NDTM in polynomieller Zeit lösbar) auch in P (von DTM in polynomieller Zeit lösbar). Falls $P \neq NP$, dann nein, weil sonst folgender Widerspruch auftritt: ein Problem in $NP \setminus P$ ist von einer NDTM, nicht aber von einer DTM in polynomieller Zeit lösbar. Könnte eine DTM die NDTM in polynomieller Zeit simulieren, wäre das Problem auch in P .

- (ii) Falsch. Es gibt NP -schwere Probleme, die nicht in NP liegen, z.B. das HALTE-Problem. Diese liegen folglich auch definitiv nicht in P . NP -schwer ist eine untere Schranke, $\in NP$ eine obere, und wenn beide erfüllt sind, ist ein Problem NP -vollständig.

□

Definition 1 (Vertex Cover (VC)). Für einen (ungerichteten) Graphen $G = (V, E)$ ist ein Vertex Cover eine Menge von Knoten $C \subseteq V$, sodass $u \in C$ oder $v \in C$ für alle $\{u, v\} \in E$ gilt, also ist für alle Kanten mindestens ein Endpunkt in C .

Für das VERTEXCOVER Problem ist neben einem Graphen G eine Zahl $k \in \mathbb{N}_{\geq 0}$ gegeben und es wird gefragt, ob ein Vertex Cover mit Größe (Anzahl der Knoten) höchstens k in G existiert.

Präsenzaufgabe 10.2 (VERTEXCOVER ist NP -vollständig)

Zeigen Sie, dass VERTEXCOVER NP -vollständig ist.

Hinweis: Reduzieren Sie CLIQUE auf VERTEXCOVER.

Lösung. VERTEXCOVER $\in NP$ ist bekannt aus der letzten Hausaufgabe.

Vorüberlegung zur Reduktion von CLIQUE auf VERTEXCOVER: Sei eine Instanz mit $G = (V, E)$ und $k \in \mathbb{N}_{\geq 0}$ von CLIQUE gegeben. Wir invertieren zunächst den Graphen G und erhalten damit G' . Nun enthält G' genau dann eine unabhängige Menge der Größe ℓ , wenn G eine Clique der Größe ℓ enthält. Wenn ein Graph mit n Knoten eine unabhängige Menge mit mindestens k Knoten hat, dann hat der Graph auch ein Vertex Cover mit maximal $n - k$ Knoten

(das Komplement zur unabhängigen Menge). Unsere Instanz für VERTEXCOVER ist also G' und $n - k$. ($k \leq n$ können wir annehmen)

Betrachte nun folgende **Reduktion**:

Sei eine Instanz mit $G = (V, E)$ und $k \in \mathbb{N}_{\geq 0}$ von CLIQUE gegeben. Invertiere den Graphen G und erhalte damit G' . Gebe nun G' und $n - k$ als Instanz von VERTEXCOVER zurück.

Korrektheit:

$\Rightarrow$ (Falls (G, k) eine Ja-Instanz von CLIQUE ist, dann ist $(G', n - k)$ eine Ja-Instanz von VERTEXCOVER.):

G enthalte eine Clique C der Größe $\geq k$. Dann ist $V' := V \setminus C$ ein Vertex Cover von G' der Größe $\leq n - k$. Da $|C| \geq k$, gilt $|V'| \leq n - k$. Da C eine Clique ist, also alle Knoten aus C mit allen anderen Knoten aus C verbunden sind, gibt es im invertierten Graphen G' keine Kante zwischen Knoten aus C . Also gilt für jede Kante $\{u, v\} \in E'$, dass $u \notin C$ oder $v \notin C$ und somit $u \in V'$ oder $v \in V'$.

$\Leftarrow$ (Falls $(G', n - k)$ eine Ja-Instanz von VERTEXCOVER ist, dann ist (G, k) eine Ja-Instanz von CLIQUE.):

G' enthalte ein Vertex Cover V' der Größe $\leq n - k$. Dann ist $C := V \setminus V'$ eine Clique der Größe $\geq k$ in G . Da $|V'| \leq n - k$, gilt $|C| \geq k$. Für jede Menge $\{u, v\} \subseteq C$ gilt, dass $\{u, v\} \notin E'$. Ansonsten wäre V' kein korrektes Vertex Cover. Da G' der invertierte Graph von G ist, gilt somit $\{u, v\} \in E$ für jede Menge $\{u, v\} \subseteq C$.

Laufzeit: Das Invertieren des Graphen kann naiv in $O(|V|^2 \cdot |E|)$ Zeit, also in polynomieller Zeit, erledigt werden.

Also existiert eine polynomielle Reduktion von CLIQUE auf VERTEXCOVER. Da CLIQUE **NP**-vollständig ist und VERTEXCOVER $\in$ **NP** gilt, ist somit VERTEXCOVER **NP**-vollständig. $\square$

Präsenzaufgabe 10.3 (Turingmaschinen)

Entwerfen Sie eine Turingmaschine für die Sprache $L = \{0^{2^n} \mid n \in \mathbb{N}\}$ über dem Alphabet $\Sigma = \{0\}$ und geben Sie die Laufzeit Ihrer Turingmaschine an. Begründen Sie die Korrektheit und die Laufzeit Ihrer Turingmaschine. Diskutieren Sie die Laufzeit Ihrer Turingmaschine im Vergleich zu einem Algorithmus (mit RAM) für diese Sprache.

Lösung. Die Turingmaschine sieht folgendermaßen aus:

- (1) Falls genau eine 0 auf dem Band steht, akzeptiere.
- (2) Bewege den Kopf von links nach rechts über die Eingabe und ersetze jede zweite 0 durch ein neues Symbol x .
- (3) Falls das letzte Symbol der Eingabe nun eine 0 ist, verwirfe.
- (4) Gehe zu Schritt (1).

Korrektheit: Die Sprache L besteht aus allen Wörtern, die nur Nullen enthalten und deren Länge eine Zweierpotenz ist. Eine Zahl ist genau dann eine Zweierpotenz, wenn sie nach wiederholtem Teilen durch 2 irgendwann eine 1 ergibt. Genau das überprüft die Turingmaschine. Wenn bei diesem wiederholten Teilen eine ungerade Zahl auftritt, ist die Zahl keine Zweierpotenz. In diesem Fall verwirft die Turingmaschine die Eingabe (siehe Schritt (3)).

Laufzeit: Die angegebene Turingmaschine hat eine Laufzeit von $O(n \log(n))$, wobei n hier die Länge der Eingabe bezeichnet. Schritt (1) und Schritt (2) benötigen jeweils eine Laufzeit von

$O(n)$. Schritt (3) geht in konstanter Laufzeit $O(1)$. Schritt (4) wiederholt die Schritte (1) bis (3) bis nur noch eine 0 auf dem Band steht oder vorher abgebrochen wird. Da in jedem Durchlauf der Schritte (1) bis (3) die Hälfte der Nullen (abgerundet) durch x Symbole ersetzt wird, steht nach $\log n$ Durchläufen nur noch eine Null auf dem Band und es wird nach Schritt (1) akzeptiert oder es steht nach $\lfloor \log n \rfloor - 1$ Durchläufen eine ungerade Anzahl von Nullen auf dem Band und es wird nach Schritt (3) verworfen. Also gibt es maximal $\log n + 1$ Durchläufe.

Diskussion: Ein Algorithmus mit RAM könnte folgendermaßen aussehen:

1. Zähle die Buchstaben und erhalte Anzahl m .
2. Falls $\log m$ ganzzahlig ist, akzeptiere. Sonst verwirfe.

Dieser Algorithmus hat eine Laufzeit von $O(n)$, wobei n hier die Länge der Eingabe bezeichnet. Ein Algorithmus mit RAM kann also den zusätzlichen Speicher nutzen, um wesentlich schneller eine Aufgabe zu erledigen, im Vergleich zu einem Algorithmus, der auf einer Turingmaschine läuft. $\square$



Präsenzaufgaben zur Vorlesung »Analyse von Algorithmen und Komplexität«

Blatt 11

Präsenzaufgabe 11.1 (Fragen über Fragen)

Diskutieren Sie nachfolgende Aussagen. Sind diese wahr oder falsch?

- (a) Für $L \in \mathbf{NP}$ gilt, aus $L \preceq 3\text{-SAT}$ folgt, dass L $\mathbf{NP}$ -vollständig ist.
- (b) Gilt für eine Sprache L , dass sowohl $L \preceq 3\text{-SAT}$ als auch $3\text{-SAT} \preceq L$, dann ist L $\mathbf{NP}$ -vollständig.
- (c) Sei L $\mathbf{NP}$ -vollständig. Dann gilt $L \in \mathbf{P}$ genau dann, wenn $\mathbf{P} = \mathbf{NP}$.
- (d) Es ist möglich, dass $3\text{-SAT} \in \mathbf{P}$ und $\text{CLIQUE} \notin \mathbf{P}$.

Lösung.

- (a) Nein, ein Gegenbeispiel ist $L = \emptyset$. Offensichtlich gilt $L \in \mathbf{NP}$. Aus $L \preceq 3\text{-SAT}$ folgt nur, dass sich L auf 3-SAT reduzieren lässt, nicht anders herum.
- (b) Das ist die Definition von $\mathbf{NP}$ -vollständig.
- (c) Korrekt, da sich jedes Problem auf L reduzieren lässt nach Def. $\mathbf{NP}$ -vollständig (und offensichtlich $\mathbf{P} \subseteq \mathbf{NP}$).
- (d) Nein, weil 3-SAT $\mathbf{NP}$ -vollständig ist, lässt sich CLIQUE (liegt bekannterweise in $\mathbf{NP}$, ist sogar $\mathbf{NP}$ -vollständig) auf 3-SAT reduzieren.

□

Präsenzaufgabe 11.2 ($\mathbf{NP}$ -Vollständigkeit)

In dem Problem CLIQUE-NOMEMBER ist ein Graph $G = (V, E)$, ein Knoten $v \in V$ sowie eine Zahl $k \in \mathbb{N}$ gegeben und es soll entschieden werden, ob es eine Clique mit k Knoten gibt, die den Knoten v nicht enthält.

Zeigen Sie, dass CLIQUE-NOMEMBER $\mathbf{NP}$ -vollständig ist, indem Sie eine Reduktion von CLIQUE auf CLIQUE-NOMEMBER angeben. Sie können davon ausgehen, dass CLIQUE-NOMEMBER $\in \mathbf{NP}$ ist.

Lösung. Clique-Nomember ist NP-schwer:

Sei $(G = (V, E), k)$ eine Eingabe für das Clique Problem.

Idee: Füge einen neuen isolierten Knoten v zu G hinzu.

Wir geben zuerst die Abbildung an: Sei $v \notin V$. Wir definieren die Eingabe für das Problem CLIQUE-NOMEMBER als den Graphen $G' := (V \cup \{v\}, E)$, den Knoten v als Knoten, der nicht

benutzt werden darf, und k als gesuchte Cliquengröße. (G', v, k) ist eine gültige Eingabe für CLIQUE-NOMEMBER und in polynomieller Zeit berechenbar.

Wir zeigen nun: G hat eine k -Clique genau dann, wenn G' eine k -Clique ohne v beinhaltet.

$\Rightarrow$:

Sei (G, k) eine Ja-Instanz des CLIQUE Problems. Dann enthält G eine k -Clique C . Dann ist C auch eine k -Clique in G' , da bei der Abbildung keine Kanten entfernt wurden. Da $v \notin V$, enthält C den Knoten v nicht. Damit ist C eine k -Clique in G ohne den Knoten v . (G', v, k) ist also eine Ja-Instanz des CLIQUE-NOMEMBER Problems.

$\Leftarrow$:

Sei (G', v, k) eine Ja-Instanz des CLIQUE-NOMEMBER Problems. Dann enthält G' eine k -Clique C mit $v \notin C$. Da bei der Abbildung an dem Graphen nichts geändert wurde, außer der Knoten v hinzugefügt, muss C auch eine k -Clique in G sein. Damit ist (G, k) eine Ja-Instanz des CLIQUE Problems.

Also existiert eine polynomielle Reduktion von CLIQUE auf CLIQUE-NOMEMBER. Da CLIQUE **NP**-vollständig ist und CLIQUE-NOMEMBER $\in$ **NP** nach Aufgabenstellung gilt, ist somit CLIQUE-NOMEMBER **NP**-vollständig. □

Präsenzaufgabe 11.3 (NP-Vollständigkeit)

Für das Problem SUBSET SUM CARDINALITY seien n ganze Zahlen $c_1, \dots, c_n \in \mathbb{N}_{>0}$ (wobei n gerade sei) sowie eine Zahl $K \in \mathbb{N}$ gegeben. Es soll entschieden werden, ob es eine Teilmenge $S \subseteq \{1, \dots, n\}$ gibt mit $|S| = n/2$ und $\sum_{i \in S} c_i = K$.

Zeigen Sie, dass das Problem SUBSET SUM CARDINALITY **NP**-vollständig ist. Sie können davon ausgehen, dass SUBSET SUM CARDINALITY $\in$ **NP** ist.

Lösung. Subset Sum Cardinality ist NP-schwer:

SUBSET SUM Problem: Gegeben: n ganze Zahlen $c_1, \dots, c_n \in \mathbb{N}_{>0}$ und Zahl $K \in \mathbb{N}$. Frage: Gibt es eine Teilmenge $S \subseteq \{0, \dots, n\}$ mit $\sum_{i \in S} c_i = K$?

Reduktion: Sei $I = (c_1, \dots, c_n, K)$ eine Eingabe für das SUBSET SUM Problem. Definiere $c'_i := c_i + 1$ für $i \leq n$ und $c'_i := 1$ für $i \in \{n+1, \dots, 2n\}$. Wir definieren die Eingabe für das Problem SUBSET SUM CARDINALITY als $I' = (c'_1, \dots, c'_{2n}, K+n)$.

Wir zeigen nun: Es gibt eine Lösung für I genau dann, wenn es eine Lösung für I' gibt.

$\Rightarrow$:

Sei I eine Ja-Instanz des SUBSET SUM Problems. Dann gibt es eine Menge $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} c_i = K$. Es gilt also $\sum_{i \in S} c'_i = \sum_{i \in S} (c_i + 1) = K + |S|$. Sei nun $S' = \{n+1, \dots, 2n - |S|\}$. Es gilt $|S \cup S'| = n$ und $\sum_{i \in S \cup S'} c'_i = \sum_{i \in S} c'_i + \sum_{i \in S'} c'_i = \sum_{i \in S} (c_i + 1) + \sum_{i \in S'} 1 = K + |S| + (n - |S|) = K + n$. $S \cup S'$ ist also eine Lösung für I' und I' damit eine Ja-Instanz.

$\Leftarrow$:

Es sei I' eine Ja-Instanz. Dann gibt es also eine Menge $S' \subseteq \{1, \dots, 2n\}$ mit $\sum_{i \in S'} c'_i = K + n$ und $|S'| = n$. Sei $S := S' \cap \{1, \dots, n\}$. Es gilt $\sum_{i \in S'} c'_i - \sum_{i \in S} c'_i = n - |S|$, da die Items in $\{n+1, \dots, 2n\}$ das Gewicht 1 haben, und genau $n - |S|$ Items übrig sind. Damit gilt $\sum_{i \in S} (c_i + 1) = \sum_{i \in S} c'_i = K + n - (n - |S|) = K + |S|$, woraus folgt, dass $\sum_{i \in S} c_i = K$. Damit ist S eine Lösung für I und somit I eine Ja-Instanz.

Laufzeit: Die Reduktion kann naiv in einer Zeit von $O(n)$ implementiert werden. Das ist eine polynomielle Laufzeit.

Also existiert eine polynomielle Reduktion von SUBSET SUM auf SUBSET SUM CARDINALITY. Da nach Skript SUBSET SUM **NP**-vollständig ist und SUBSET SUM CARDINALITY $\in$ **NP** nach Aufgabenstellung gilt, ist somit SUBSET SUM CARDINALITY **NP**-vollständig. $\square$



Präsenzaufgaben zur Vorlesung »Analyse von Algorithmen und Komplexität«

Blatt 12

Präsenzaufgabe 12.1

Im Folgenden sollen Sie herausfinden (und begründen), welche Lower Bounds sich über ETH und die Reduktionen aus der Vorlesung ergeben. Nutzen Sie hierfür auch das Sparsification Lemma.

Welche Schranken für die Laufzeit in Abhängigkeit von den natürlichen Parametern (Anzahl Knoten, Anzahl Kanten, Anzahl Items, Anzahl unterschiedlicher Items, Anzahl Tripel, Kardinalität der Grundmenge, Anzahl unterschiedlicher Tripel, Anzahl unterschiedlicher auftretender Zahlen, größte auftretende Zahl, ...) ergeben sich jeweils aus folgenden (zum Großteil aus der Vorlesung bekannten) Reduktionen (Laufzeitfaktoren, die polynomiell in der Eingabekodierung sind, dürfen vernachlässigt werden)?

Schätzen Sie dazu zunächst die verschiedenen Parameter der resultierenden Instanz mithilfe der Parameter der eingegebenen Instanz geeignet ab.

1. $3\text{-SAT} \preceq k\text{-CLIQUE}$
2. $k\text{-CLIQUE} \preceq k\text{-INDEPENDENTSET}$ (Problemdefinition für die Antwort nicht relevant; in der Reduktion wird der Graph $G = (V, E)$ auf den Komplementgraph $(V, \bar{E})$ abgebildet)
3. $\text{SAT} \preceq 3\text{-DIMENSIONALMATCHING}$ (Hinweis: $3\text{-SAT} \subseteq \text{SAT}$)
4. $3\text{-DIMENSIONALMATCHING} \preceq 3\text{-EXACTCOVER}$
5. $3\text{-EXACTCOVER} \preceq \text{SUBSETSUM}$

Lösung. Angenommen ETH gilt. Das Sparsification Lemma impliziert, dass man 3-SAT nicht in $2^{o(m)} \cdot |I|^{O(1)}$ lösen kann.

1. In der Reduktion werden $|V| = O(m)$ Knoten und $|E| = O(m^2)$ Kanten erstellt und $k = m$. Außerdem gilt $n \leq 3m \in O(m)$.
 - Lower bound bzgl. der Anzahl an Knoten: $2^{o(|V|)} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der $k\text{-Clique}$ in $2^{o(|V|)} \cdot |I|^{O(1)}$ löst. Dann gibt es auch einen Algorithmus, der 3-SAT in $2^{o(m)} \cdot |I|^{O(1)}$ löst (Reduktion auf $k\text{-Clique}$ + $k\text{-Cliquen}$ Algorithmus). Nach dem Sparsification Lemma gibt es dies nur, wenn die ETH nicht gilt.

- Lower bound bzgl. der Anzahl an Kanten: $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der k -Clique in $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$ löst. Da $|E| \in O(m^2)$, gibt es dann auch einen Algorithmus, der 3-SAT in $2^{o(m)} \cdot |I|^{O(1)}$ löst (Reduktion auf k -Clique + k -Cliques Algorithmus). Nach dem Sparsification Lemma gibt es dies nur, wenn die ETH nicht gilt.
 - Lower bound bzgl. k : $2^{o(k)} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der k -Clique in $2^{o(k)} \cdot |I|^{O(1)}$ löst. Dann gibt es auch einen Algorithmus, der 3-SAT in $2^{o(m)} \cdot |I|^{O(1)}$ löst (Reduktion auf k -Clique + k -Cliques Algorithmus). Nach dem Sparsification Lemma gibt es dies nur, wenn die ETH nicht gilt.
2. In der Reduktion werden genau $|V|$ Knoten und maximal $|E| = O(|V|^2)$ Kanten erstellt.
- Lower bound bzgl. der Anzahl an Knoten: $2^{o(|V|)} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der k -Independent-Set in $2^{o(|V|)} \cdot |I|^{O(1)}$ löst. Dann gibt es auch einen Algorithmus, der k -Clique in $2^{o(|V|)} \cdot |I|^{O(1)}$ löst (Reduktion auf k -Independent-Set + k -Independent-Set Algorithmus). Nach Aufgabenteil 1, gibt es diesen nur, wenn die ETH nicht gilt.
 - Lower bound bzgl. der Anzahl an Kanten: $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der k -Independent-Set in $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$ löst. Da $|E| \leq |V|^2$, gibt es dann auch einen Algorithmus, der k -Clique in $2^{o(|V|)} \cdot |I|^{O(1)}$ löst (Reduktion auf k -Independent-Set + k -Independent-Set Algorithmus). Nach Aufgabenteil 1, gibt es diesen nur, wenn die ETH nicht gilt.
3. In der Reduktion werden Mengen mit $|U| = |V| = |W| = O(mn)$ Elementen erstellt und $|T| = O(m^2n^2)$ Tupel. Es gilt $n \leq 3m \in O(m)$ für 3-SAT.
- Lower bound bzgl. der Anzahl an Elementen: $2^{o(\sqrt{|V|})} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der 3-Dimensional-Matching in $2^{o(\sqrt{|V|})} \cdot |I|^{O(1)}$ löst. Da $|V| \in O(mn) \leq O(m^2)$, gibt es dann auch einen Algorithmus, der 3-SAT in $2^{o(m)} \cdot |I|^{O(1)}$ löst (Reduktion von SAT auf 3-Dimensional-Matching angewandt auf 3-SAT Instanzen + 3-Dimensional-Matching Algorithmus). Nach dem Sparsification Lemma geht dies nur, wenn die ETH nicht gilt.
 - Lower bound bzgl. der Kardinalität von T : $2^{o(\sqrt[4]{|T|})} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der 3-Dimensional-Matching in $2^{o(\sqrt[4]{|T|})} \cdot |I|^{O(1)}$ löst. Da $|T| \in O(m^2n^2) \leq O(m^4)$, gibt es dann auch einen Algorithmus, der 3-SAT in $2^{o(m)} \cdot |I|^{O(1)}$ löst (Reduktion von SAT auf 3-Dimensional-Matching angewandt auf 3-SAT Instanzen + 3-Dimensional-Matching Algorithmus). Nach dem Sparsification Lemma geht dies nur, wenn die ETH nicht gilt.
4. Das 3-dimensionale Matching ist ein Spezialfall vom 3-Exact Cover. Daher gelten die Lower Bounds von 3-dimensionalem Matching auch für 3-Exact Cover. Da $|U| = 3m \in O(|V|)$, ergibt sich, dass es für 3-Exact Cover unter der ETH keinen Algorithmus mit Laufzeit $2^{o(\sqrt{m})} \cdot |I|^{O(1)}$ geben kann, da sonst 3-dimensionales Matching in $2^{o(\sqrt{|V|})} \cdot |I|^{O(1)}$ lösbar wäre. Und da $|F| = |T|$, ergibt sich, dass es für 3-Exact Cover unter der ETH keinen Algorithmus mit Laufzeit $2^{o(\sqrt[4]{|F|})} \cdot |I|^{O(1)}$ geben kann, da sonst 3-dimensionales Matching in $2^{o(\sqrt[4]{|T|})} \cdot |I|^{O(1)}$ lösbar wäre.
5. In der Reduktion wird $n = |F|$ gesetzt. Lower bound bzgl. n : $2^{o(\sqrt[4]{n})} \cdot |I|^{O(1)}$
Angenommen, es existiert ein Algorithmus, der SubsetSum in $2^{o(\sqrt[4]{n})} \cdot |I|^{O(1)}$ löst. Dann

gibt es auch einen Algorithmus, der 3-Exact Cover in $2^{o(\sqrt[4]{|F|})} \cdot |I|^{O(1)}$ löst. Nach Aufgabenteil 4, gibt es diesen nur, wenn die ETH nicht gilt. $\square$



Präsenzaufgaben zur Vorlesung »Analyse von Algorithmen und Komplexität«

Blatt 13

Präsenzaufgabe 13.1 (Vertex Cover)

Ein Vertex Cover in einem Graphen $G = (V, E)$ ist eine Menge von Knoten $C \subseteq V$, so dass für jede Kante $\{v, w\} \in E$ gilt, dass $v \in C$ oder $w \in C$. Beim Optimierungsproblem Minimum Vertex Cover soll die kardinalitätskleinste Menge $C \subseteq V$ gefunden werden, die ein Vertex Cover von G ist.

Gegeben ist der folgende approximative Algorithmus mit Güte 2 für VertexCover:

Idee: Wähle eine beliebige Kante. Wenn einer der beiden Endknoten bereits in C ist, verwirf diese Kante. Sonst, füge beide Endknoten zur Menge C hinzu. Mache dies für jede Kante.

Algorithmus 2APPROXCV($G = (V, E)$)

```
1   $C = \{\}$ 
2  foreach  $\{u, v\} \in E$  do
3    if  $u \notin C$  and  $v \notin C$  then
4       $C = C \cup \{u, v\}$ 
5    fi
6  od
7  return  $C$ 
```

Beweisen Sie die Güte.

Lösung. Beweis: Sei C^* ein minimales Vertex Cover. Wir zeigen $|C| \leq 2|C^*|$.

Sei A die Menge der Kanten, für die beide Endknoten zu C hinzugefügt wurden. Es gilt $|C| = 2|A|$, da beide Knoten jeder ausgewählten Kante zu C hinzugefügt wurden und alle anderen Kanten nicht weiter betrachtet werden. Damit folgt auch, dass keine zwei Kanten aus A einen Knoten gemeinsam haben können. Sei nun C^* ein minimales Vertex Cover. C^* enthält nach Definition mindestens einen Knoten jeder Kante, also insbesondere einen Knoten jeder Kante aus A . Da keine zwei Kanten aus A vom gleichen Knoten aus C^* überdeckt werden können, gilt $|C^*| \geq |A|$. Es folgt $|C| = 2|A| \leq 2|C^*|$. Die Lösung C von **2ApproxCV** ist also maximal doppelt so groß wie die optimale Lösung C^* . Damit berechnet der Algorithmus eine 2-Approximation.

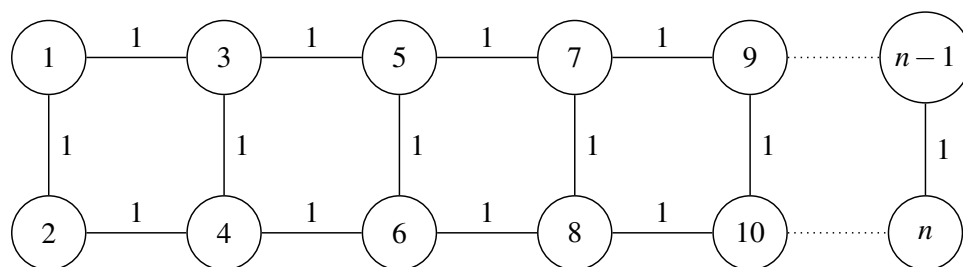
□

Präsenzaufgabe 13.2 (TSP)

In der Vorlesung haben Sie zwei approximative Algorithmen für das TSP-Problem kennen gelernt. Der erste Algorithmus ΔTSP_1 hat eine Approximationsrate von 2, während der zweite Algorithmus ΔTSP_2 von Christofides eine Approximationsrate von $3/2$ besitzt. In dieser Übung soll gezeigt werden, dass die zweite Rate scharf ist. D.h., dass es eine Folge von Instanzen gibt, auf denen der Algorithmus immer näher an diese Rate heran kommt.

Wir betrachten den vollständigen Graphen $K_n = (V, E)$ für ein $n \in \mathbb{N}_{\geq 6}$ mit $n \bmod 4 = 2$ und die nachfolgende Gewichtsfunktion $d_2: E \rightarrow \mathbb{R}_+$. Zeigen Sie, dass

- $\text{OPT}((K_n, d_2)) = n$ gilt und
- eine mögliche Ausführung von ΔTSP_2 existiert, die auf Eingabe (K_n, d_2) eine Tour der Länge $(n-1) + n/2$ berechnet.

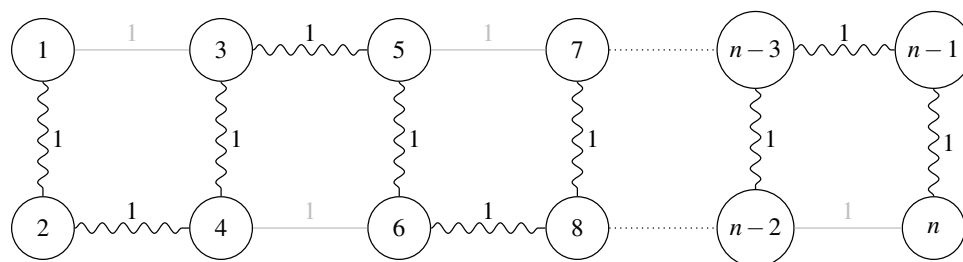


Wir definieren $E_2 := \{\{i, i+2\} | i \in \{1, \dots, n-2\}\} \cup \{\{2i+1, 2i+2\} | i \in \{0, \dots, n/2-1\}\}$ und d_2 , so dass alle Kanten aus E_2 auf 1 abgebildet werden und alle Kanten aus $E \setminus E_2$ auf den Wert des kürzesten Wegs zwischen diesen Knoten in dem oben abgebildeten Graphen abgebildet werden.

Lösung. Wichtig bei dieser Aufgabe ist der folgende Fakt: Bei beiden Algorithmen wird *irgendein* MST genommen und dann *irgendeine* Reihenfolge der Knoten. Typischerweise wird diese Reihenfolge durch bestimmte Knoten- oder Kanten-Labels festgelegt (wird zum Beispiel der Kruskal-Algorithmus zur Konstruktion des MST genutzt, so werden dort die Kanten nach Gewicht sortiert. Kanten gleichen Gewichts müssen also auch noch durch irgendeine Reihenfolge festgelegt werden.) Wir gehen also davon aus, dass wir diese Reihenfolge auch wählen können. Sollte der Algorithmus also einen “besseren” MST oder eine “bessere” Knotenreihenfolge nutzen, passen wir die Labels dementsprechend an.

Ein möglicher MST ist durch die unten angegebene Zick-Zack-Tour gegeben oder formal durch

$$E_T = \left[\bigcup_{\substack{1 \leq j \leq n-5 \\ j \bmod 4 = 1}} \{\{j, j+1\}, \{j+1, j+3\}, \{j+3, j+2\}, \{j+2, j+4\}\} \right] \cup \{\{n-1, n\}\}.$$



Es gilt $d(T) = n - 1$ für die Kosten $d(T)$ des MST T , da nur Kanten mit Kosten 1 gewählt werden und $n - 1$ Kanten gewählt werden müssen, um den Zusammenhang sicher zu stellen.

Da jeder Knoten im MST außer 1 und n geraden Grad hat, ist das Matching K sehr simpel, nämlich $K = \{\{1, n\}\}$. Die Kosten der Kante $\{1, n\}$ sind $n/2$ (der kürzeste Pfad von 1 zu n) und somit gilt $d(K) = n/2$ für die Kosten von K . Die Rundreise R entspricht einfach dem MST vereinigt mit der Matchingkante und hat somit Kosten: $d(R) = n - 1 + n/2$.

Die optimale Tour würde erst “oben” und dann “unten” entlang laufen, also von Knoten 1 zu Knoten 3 gehen, dann zu Knoten 5 und so weiter, bis Knoten $n - 1$ erreicht wurde. Danach folgen Knoten $n, n - 2, n - 4$ und so weiter. Diese Tour hat offensichtlich Kosten n , da wir n Kanten mit Gewicht 1 nutzen. Somit gilt $\text{OPT}((K_n, d_2)) = n$ und wir haben eine Rate von

$$\frac{\Delta TSP_2((K_n, d_2))}{\text{OPT}((K_n, d_2))} = \frac{n - 1 + n/2}{n} = 3/2 - \frac{1}{n} \leq 3/2.$$

Da $\lim_{n \rightarrow \infty} 3/2 - \frac{1}{n} = 3/2$, kommen wir der Rate von $3/2$ also beliebig nahe.

□