Institut für Informatik
Prof. Dr. J. Rothe

Wintersemester 2023/24
Übungsblatt 11

Lösungsvorschläge

Kryptokomplexität 1

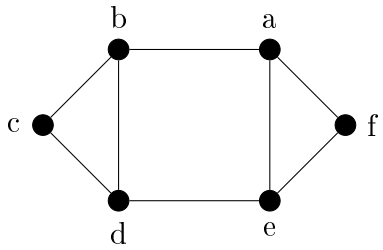
Bearbeitungszeit: 19. Dezember bis 10-12. Januar
Verantwortlich: Roman Zorn

Aufgabe 1 : Hamiltonkreis

Sei $G = (V, E)$ ein ungerichteter Graph. Ein Kreis in G, der jeden Knoten aus V genau einmal besucht, heißt *Hamiltonkreis*. Das zugehörige Entscheidungsproblem HamiltonCircuit ist wie folgt definiert:

HamiltonCircuit	
Gegeben:	Ein ungerichteter, zusammenhängender Graph $G = (V, E)$.
Frage:	Besitzt G einen Hamiltonkreis?

(a) ▶ Begründen Sie warum die folgende Instanz $G = (V, E)$ eine JA-Instanz für das Problem HamiltonCircuit ist.



(b) ▶ Entfernen Sie möglichst wenige Kanten aus G, so dass der resultierende Graph G' eine NEIN-Instanz für HamiltonCircuit ist. Begründen Sie Ihre Antwort.

(c) ▶ Geben Sie eine möglichst gute obere Schranke für die deterministische Zeitkomplexität des Problems HamiltonCircuit an, d.h., geben Sie eine geeignete und möglichst kleine deterministische Komplexitätsklasse an, die HamiltonCircuit enthält. Begründen Sie Ihre Antwort.

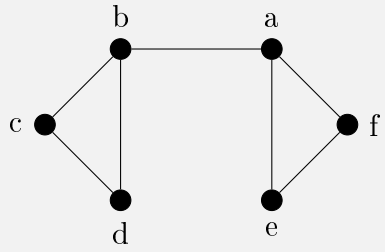
(d) ▶ Geben Sie eine möglichst gute obere Schranke für die nichtdeterministische Zeitkomplexität des Problems HamiltonCircuit an, d.h., geben Sie eine geeignete und möglichst kleine nichtdeterministische Komplexitätsklasse an, die HamiltonCircuit enthält. Begründen sie Ihre Antwort.

Bei (c) und (d) reicht es aus, wenn Sie die Anzahl der Knoten und Kanten des Graphen der Instanz als Instanzgröße auffassen und informell argumentieren.

Lösungsvorschlag:

(a) Beispielsweise der Kreis a, b, c, d, e, f, a ist ein Hamiltonkreis in G und genügt um zu begründen, warum G eine JA-Instanz ist.

(b) Beispielsweise das Entfernen der Kante $\{d, e\}$ aus G liefert eine neue Instanz G', in der es offensichtlich keinen Hamiltonkreis mehr gibt.



Jeder Kreis in G', der alle Knoten mindestens einmal besucht, besucht die Knoten a und b zweimal.

(c) Wir beschreiben einen naiven Ansatz zum Lösen des Problems. Für eine Instanz $G = (V, E)$ testen wir für jede mögliche Permutation der Knoten in V ob diese Permutation $x_1, \ldots, x_{|V|}$ ein Hamiltonkreis ist, also ob alle aufeinander folgenden Knoten der Permutation mittels Kanten aus E verbunden sind. Insgesamt gibt es $|V|!$ verschiedene Permutationen der Knoten. Für jede dieser Permutationen muss getestet werden ob die notwendigen Kanten zwischen den Knoten existieren, das geht in $\mathcal{O}(|V||E|)$. Damit benötigt dieser Ansatz eine Laufzeit in $\mathcal{O}(|V|!|V||E|)$. Für $|V| = n$ gilt

$$n! \leq n^n = 2^{\log_2(n)n} \leq 2^{n^2},$$

also $\mathcal{O}(n!) \subseteq \mathcal{O}(2^{n^2})$. Damit folgt, dass HamiltonCircuit $\in$ EXP gilt.

(d) Für das Problem HamiltonCircuit lässt sich die Zeitschranke im nichtdeterministischen Fall verbessern. Sei $G = (V, E)$ die betrachtete Instanz. Dann "raten" wir nichtdeterministisch in $\mathcal{O}(|V|)$ eine Permutation der Knoten von G und testen anschließend deterministisch in $\mathcal{O}(|V||E|)$ ob die geratene Permutation ein Hamiltonkreis ist, also ob alle notwendigen Kanten existieren. Damit brauchen wir insgesamt eine Laufzeit in $\mathcal{O}(|V| + |V||E|)$ und es folgt: HamiltonCircuit $\in$ NP.

Aufgabe 2 : Ein- vs. k-Band-Turingmaschinen

Sei M eine deterministische k-Band-Turingmaschine, die für eine Eingabe der Länge n in Zeit $t(n)$ auf Platz $s(n)$ arbeitet.

► Beschreiben Sie eine deterministische 1-Band-Turingmaschine N mit $L(N) = L(M)$, die möglichst auch auf Platz $s(n) + \mathcal{O}(1)$ arbeitet, und schätzen Sie die Laufzeit von N in Abhängigkeit von $t(n)$ ab.

Lösungsvorschlag: Wir simulieren die k-Band-DTM M mittels einer 1-Band-DTM N wie folgt.

Im ersten Schritt unterteilen wir das Band von N in k Abschnitte. Zum Unterteilen der Abschnitte verwenden wir zwei neue Symbole die jeweils Anfang und Ende eines Abschnitts symbolisieren. An das Ende der k Abschnitte ergänzen wir ein weiteres neues Symbol, um das gesamte Ende aller k Abschnitte für N zu markieren. Die k Abschnitte des Bandes von N nutzen wir, um die Inhalte der k Bänder von M zu speichern. Außerdem erweitern wir das Arbeitsalphabet von N, indem wir für jedes Symbol eine markierte Kopie des Symbols ergänzen.

In den ersten Abschnitt schreibt N anschließend die ursprüngliche Eingabe von M. Der aktuelle Zustand von N entspricht einem Tupel, dass unter Anderem auch den aktuellen Zustand von M beinhaltet.

Um nun einen Arbeitsschritt von M zu simulieren, operiert N wie folgt. N läuft über die k Abschnitte von links nach rechts und merkt sich in seinem Zustand auf welchen Symbolen die k Köpfe von M gerade stehen. Dabei ist die Position jedes Kopfes von M durch eine markierte Kopie des jeweiligen Symbols gekennzeichnet. Jetzt kennt N sowohl den Zustand von M als auch die Symbole unter allen k Köpfen von M. Damit ist klar welchen Arbeitsschritt M als nächstes machen wird.

N aktualisiert nun die k Abschnitte und schreibt für jeden Abschnitt auf die Position des Kopfes das neue Symbol. Anschließend aktualisiert N die Positionen der k Köpfe, indem die entsprechenden Symbole auf den k Bändern durch markierte Kopien ersetzt werden. Schlussendlich speichert N den neuen Zustand von M in seinem Zustand ab. Damit hat N genau einen Arbeitsschritt von M simuliert. Falls auf einem der k Bänder von M, also in einem der k Abschnitte von N, ein Leerzeichen eingefügt werden muss, so kann N alle rechts von diesem Abschnitt liegenden Abschnitte um ein Zeichen nach rechts verschieben und ein neues Leerzeichen auf dem entsprechenden Abschnitt einfügen.

Die akzeptierenden Endzustände von N entsprechen genau den akzeptierenden Endzuständen von M. Damit gilt, dass N genau dann akzeptiert, wenn M akzeptiert.

Der insgesamt notwendige Platzbedarf von N ist gleich zu dem von M zuzüglich einer Konstante in $\mathcal{O}(1)$ für die die Abschnitte begrenzenden Symbole, da N alle k Bänder von M auf einem Band statt auf N Bändern speichert.

Wir wissen, dass M in Zeit $t(n)$ arbeitet. Folglich können die k Wörter auf den k Bändern von M nicht länger als $t(n)+1$ werden. Damit ist das Wort auf dem Band von N höchstens $k(t(n)+3)+1$ lang. Für jedem Arbeitsschritt von M läuft N zweimal von links nach rechts über das gesamte Band (1.: Lesen aller Kopf-Positionen, 2.: Aktualisieren aller Köpfe) und insgesamt höchstens k Mal bis zum rechten Ende und zurück für jedes Leerzeichen, das eingefügt werden muss. Damit kostet jeder Arbeitsschritt von M maximal $(2+k)2(k(t(n)+3)+1)$ Schritte von N. Insgesamt gibt es höchstens $t(n)$ Arbeitsschritte von M, so dass N eine Laufzeit von

$$t(n)[(2+k)2(k(t(n)+3)+1)] \in \mathcal{O}(t(n)^2)$$

aufweist.

Aufgabe 3 : Raumkonstruierbarkeit

Es seien f und g raumkonstruierbare Funktionen.

► Zeigen Sie, dass $f+g$, $f \cdot g$ und $\max\{f,g\}$ raumkonstruierbar sind.

Hinweis: Sie können hier annehmen, dass eine Turingmaschine ein separates Eingabeband hat, das bei der Bearbeitung nicht verändert wird.

Lösungsvorschlag: Gemäß Definition ist eine Funktion h raumkonstruierbar, falls es eine deterministische Turingmaschine gibt, die für jede Eingabe n den Wert $h(n)$ auf Platz kleiner gleich $h(n)$ berechnet und dabei die Zeichenkette $\#1^{h(n)-2}\$$ auf ein Band schreibt. Um also für die obigen Kompositionen zu zeigen, dass diese raumkonstruierbar sind, genügt es eine entsprechende DTM anzugeben.

(a) $f+g$ ist raumkonstruierbar: Da f und g raumkonstruierbar sind, existieren entsprechende DTMs F und G wie oben beschrieben. Wir definieren eine neue DTM H wie folgt:

Für eine Eingabe n simuliert H zunächst die Berechnung von F auf n und schreibt das Ergebnis $\#1^{f(n)-2}\$$ auf das Ausgabeband. Anschließend simuliert H die Berechnung von G auf Eingabe n und hängt das Ergebnis an das bereits berechnete Ergebnis von F an. Danach steht auf dem Ausgabeband von H die Zeichenkette $\#1^{f(n)-2}\$\#1^{g(n)-2}\$$. H ersetzt nun $\$\#$ durch 11. Jetzt lässt sich die Zeichenkette auf dem Ausgabeband von H auch als $\#1^{f(n)+g(n)-2}\$$ ausdrücken. Offensichtlich berechnet H also die Funktion $f(n)+g(n)$ und benötigt dazu höchstens einen Platz von $f(n)+g(n)$. Damit folgt, dass die Funktion $f+g$ raumkonstruierbar ist.

(b) $f \cdot g$ ist raumkonstruierbar: Da f und g raumkonstruierbar sind, existieren entsprechende DTMs F und G wie oben beschrieben. Wir definieren eine neue DTM H wie folgt:

Für eine Eingabe n simuliert H zunächst die Berechnung von F auf n und schreibt das Ergebnis $\#1^{f(n)-2}\$$ auf das Ausgabeband. Für jedes Zeichen auf dem Ausgabeband simuliert H nun die Berechnung von G auf n und ersetzt das entsprechende Zeichen durch die Ausgabe $\#1^{g(n)-2}\$$ von G. Nach dem Ersetzen des ersten Zeichens steht auf dem Ausgabeband von H also $\#1^{g(n)-2}\$1^{f(n)-2}\$$. Nach dem Ersetzen des zweiten Zeichens steht auf dem Ausgabeband von H $\#1^{g(n)-2}\$\#1^{g(n)-2}\$1^{f(n)-3}\$$, usw.. Wenn H das Zeichen \$ ersetzt, markiert es das letzte Zeichen der Ausgabe. Nachdem H alle Zeichen der ursprünglichen Ausgabe von F ersetzt hat, ersetzt H alle Zeichenketten der Form \$# durch die Zeichenkette 11 und entfernt schließlich die Markierung vom letzten Zeichen. Nun steht auf dem Ausgabeband von H die Zeichenkette $\#1^{f(n)\cdot g(n)-2}\$$. Offensichtlich berechnet H also die Funktion $f(n) \cdot g(n)$ und benötigt dazu höchstens einen Platz von $f(n) \cdot g(n)$, so dass folgt, dass die Funktion $f \cdot g$ raumkonstruierbar ist.

(c) $\max\{f, g\}$ ist raumkonstruierbar: Da f und g raumkonstruierbar sind, existieren entsprechende DTMs F und G wie oben beschrieben. Wir definieren eine neue DTM H wie folgt:

Für eine Eingabe n simuliert H zunächst die Berechnung von F auf n und schreibt das Ergebnis $\#1^{f(n)-2}\$$ auf das Ausgabeband. Dann markiert es das letzte Zeichen (\$) der Ausgabe. Anschließend simuliert H die Berechnung von G auf n und überschreibt mit dem Ergebnis $\#1^{g(n)-2}\$$ das zuvor berechnete Ergebnis auf dem Ausgabeband. Liest es dabei an einem Punkt das markierte Dollarzeichen, gilt $g(n) \geq f(n)$ und H endet, nachdem es die Ausgabe von G vollständig auf das Band geschrieben hat. Ansonsten gilt $g(n) < f(n)$ und H überschreibt das entstandene unmarkierte Dollarzeichen wieder mit einer Eins und entfernt die Markierung vom letzten Zeichen. Anschließend steht auf dem Ausgabeband von H die Zeichenkette $\#1^{\max\{f,g\}-2}\$$. Für diese Berechnung benötigt H höchstens einen Platz von $\max\{f, g\}$ und so folgt, dass die Funktion $\max\{f, g\}$ raumkonstruierbar ist.

Aufgabe 4 : SetCover

Das Problem SetCover ist wie folgt definiert:

SetCover	
Gegeben:	Eine Menge $U = \{u_1, u_2, \ldots, u_n\}$ mit $n \geq 1$, eine Menge $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ mit $S_i \subseteq U$, für $1 \leq i \leq m$, und $\bigcup_{1 \leq i \leq m} S_i = U$ sowie eine Zahl $k \in \mathbb{N}$.
Frage:	Gibt es eine Teilmenge $\mathcal{S}' \subseteq \mathcal{S}$ mit $\lvert\mathcal{S}'\rvert \leq k$, so dass $\bigcup_{S_i \in \mathcal{S}'} = U$ gilt?

(a) Gegeben seien die folgenden zwei SetCover-Instanzen:

(i) Instanz $I = (U, \mathcal{S}, k)$ mit

$$\begin{aligned} U &= \{1, 2, \ldots, 12\}, \\ \mathcal{S} &= \{\{1,3,7\}, \{3,4,6\}, \{3,4,7\}, \{2,9,10\}, \{9,11,12\}, \\ &\quad \{6,8,10\}, \{5,6,8\}, \{5,9,11\}, \{1,2,12\}, \{2,5,8\}\} \text{ und} \\ k &= 3, \end{aligned}$$

(ii) Instanz $I'' = (U', \mathcal{Z}, k')$ mit

$$\begin{aligned} U' &= \{1, 2, \ldots, 12\}, \\ \mathcal{Z} &= \{\{1,3,4,11\}, \{1,2,4,5\}, \{6,9\}, \{3,10,11\}, \\ &\quad \{7,8,11,12\}, \{3,7,9,10,12\}\} \text{ und} \\ k' &= 4. \end{aligned}$$

► Entscheiden und begründen Sie ob es sich bei den angegebenen Instanzen um JA- oder NEIN-Instanzen für SETCOVER handelt.

(b) ► Geben Sie eine möglichst gute obere Schranke für die deterministische Zeitkomplexität von SETCOVER an und begründen Sie Ihre Antwort.

(c) ► Geben Sie eine möglichst gute obere Schranke für die nichtdeterministische Zeitkomplexität von SETCOVER an begründen Sie Ihre Antwort.

Lösungsvorschlag:

(a) Wir argumentieren für beide Instanzen einzeln:

(i) Bei I handelt es sich um eine NEIN-Instanz. Für jedes $S_i \in \mathcal{S}$ gilt $|S_i| = 3$, mit $k = 3$ dreielementingen Teilmengen aus $\mathcal{S}$ können wir aber höchstens neun der zwölf Elemente in U überdecken.

(ii) Bei I' handelt es sich um eine JA-Instanz, z.B. besteht eine Überdeckung der Größe $k' = 4$ für U' aus $\{1,2,4,5\}, \{6,9\}, \{3,10,11\}$ und $\{7,8,11,12\}$.

(b) Wir geben einen naiven, deterministischen Algorithmus für SETCOVER an: Der Algorithmus überprüft nacheinander alle möglichen Teilmengen von $\mathcal{S}$ der Größe k. Falls eine dieser Teilmengen eine Überdeckung für U ist, so gibt der Algorithmus diese zurück, andernfalls endet der Algorithmus mit "nicht erfüllbar".

Insgesamt gibt es $\binom{m}{k} \leq m^k$ Teilmengen der Größe k in $\mathcal{S}$. Für jede dieser Teilmengen können wir in Zeit in $\mathcal{O}(kn)$ testen, ob es sich um eine Überdeckung für U handelt.

Damit erhalten wir insgesamt eine Laufzeit in $\mathcal{O}(m^k kn)$. Offensichtlich gilt

$$m^k \leq m^m \leq 2^{m^2},$$

so dass $\mathcal{O}(m^k kn) \subseteq \mathcal{O}(2^{m^2+m+n})$ gilt und wir eine exponentielle Laufzeit erhalten. Damit folgt SETCOVER $\in$ EXP.

(c) Wir geben einen nichtdeterministischen Algorithmus für SetCover an: Der Algorithmus "rät" in $\mathcal{O}(k)$ eine Teilmenge $\mathcal{S}' \subseteq \mathcal{S}$ der Größe k. Anschließend überprüft der Algorithmus deterministisch in Zeit in $\mathcal{O}(kn)$ ob diese Teilmenge ein Überdeckung von U ist. Insgesamt benötigt der Algorithmus also eine Zeit in $\mathcal{O}(k + nk)$, damit folgt SetCover $\in$ NP.