Algorithmen und Komplexität
Musterlösung Klausur Winter 2018

Lösungsvorschlag zu Aufgabe 1

Welche der folgenden Aussagen treffen zu, und welche nicht? Für jede richtige Antwort erhalten Sie 1 Punkt (keine Minuspunkte). Hier müssen Sie die Antworten *nicht* begründen.

(a) $2^{2n} = \mathcal{O}(3^n)$.

wahr ☐ falsch ☒

(b) Gegeben eine Netzwerk $N = (V, E, \ell)$. Angenommen e ist die einzige Kante mit minimalem Kantengewicht in E, das heisst $\ell(e) < \ell(e')$ für alle Kanten $e' \neq e$. Dann ist e in jedem minimalen Spannbaum enthalten.

wahr ☒ falsch ☐

(c) Ein $(3, 5)$-Baum mit n Schlüsseln hat Höhe $\mathcal{O}(\log n)$.

wahr ☒ falsch ☐

(d) Wird bei einem (a, b)-Baum eine DELETE-Operation ausgeführt, so ist immer eine Rebalancierung notwendig.

wahr ☐ falsch ☒

(e) Fibonacci-Heaps eignen sich gut, um die asymptotische Laufzeit des Algorithmus von Prim auf $\mathcal{O}(|V| \log |V| + |E|)$ zu reduzieren.

wahr ☒ falsch ☐

(f) Union-Find Strukturen eignen sich gut, um die assymptotische Laufzeit des Algorithmus von Dijkstra auf $\mathcal{O}(|V| \log |V| + |E|)$ zu reduzieren.

wahr ☐ falsch ☒

(g) Ein Fibonacci-Heap mit n Schlüsseln hat maximal Höhe $\mathcal{O}(\log n)$.

wahr ☐ falsch ☒

(h) Die Boolesche Formel $F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2)$ ist erfüllbar.

wahr ☒ falsch ☐

(i) Sei L eine $\mathcal{NP}$-vollständige Sprache. Es gilt für alle Sprachen L', dass $L' \leq_P L$.

wahr ☐ falsch ☒

(j) Jeder Graph in dem alle Knoten Grad höchstens 2 haben ist bipartit.

wahr ☐ falsch ☒

(k) Sei $G = (V, E)$ ein Graph. Sei $\mathcal{U}$ die Menge, welche aus den Teilmengen $A \subseteq E$ mit der Eigenschaft, dass der Graph (V, A) keinen Kreis enthält, besteht. Dann ist $\mathcal{M} = (E, \mathcal{U})$, ein Matroid.

wahr ☒ falsch ☐

(l) Sei $N = (V, E, \ell)$ ein Netzwerk mit positiven Kantengewichten und sei T ein minimaler Spannbaum von N. Für jedes Paar von Knoten $u, v \in V$ gilt, dass der eindeutige $u - v$ Pfad in T ein kürzester $u - v$ Pfad in N ist.

wahr ☐ falsch ☒

(m) Es ist bekannt, dass das Problem 2-COL in $\mathcal{P}$ liegt.

wahr ☒ falsch ☐

(n) Jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ kann als Boolsche Formel in Konjunktiver Normalform dargestellt werden.

wahr ☒ falsch ☐

(o) Es gibt genau 2^n Boolesche Funktionen $f : \{0,1\}^n \to \{0,1\}$.

wahr ☐ falsch ☒

(p) Es gibt mehr unberechenbare als berechenbare Funktionen $f : \{0,1\}^* \to \{0,1\}$.

wahr ☒ falsch ☐

(q) Sei L eine Sprache mit 3-SAT $\leq_p L$. Dann ist L $\mathcal{NP}$-vollständig.

wahr ☐ falsch ☒

(r) Seien L und L' zwei Sprachen. Aus $L \leq_p L'$ folgt $L' \leq_p L$.

wahr ☐ falsch ☒

(s) Wenn ein $\mathcal{NP}$-vollständiges Problem in polynomieller Zeit gelöst werden kann, so können alle Probleme in $\mathcal{NP}$ in polynomieller Zeit gelöst werden.

wahr ☒ falsch ☐

(t) Es sei $f : \{1, 2, \ldots, n\} \to \mathbb{N}$ eine monoton steigende Funktion, welche in konstanter Zeit berechnet werden kann. Für jedes x ist in $\mathcal{O}(\log n)$ Zeit prüfbar, ob es ein $k \in \{1, 2, \ldots, n\}$ gibt mit $f(k) = x$.

wahr ☒ falsch ☐

Lösungsvorschlag zu Aufgabe 2

(a) Wir beginnen mit dem Pseudocode für unser Programm:

for $i = 1, \dots n$ **do**
  $f(i,1) = a_{i,1}$
  $f(1,i) = a_{1,i}$
end for
for $i = 2, \dots n$ **do**
  for $j = 2, \dots n$ **do**
    $f(i,j) = \text{ORAKEL}(a_{i,j}, f(i,j-1), f(i-1,j), f(i-1,j-1))$
  end for
end for
return $\max_{1 \le i,j \le n}\{a_{i,j}\}$

Korrektheit: In der ersten for-Schleife initialisieren wir die Werte $f(i,1), f(1,i)$. Mit der Definition von $f(i,j)$ ist klar, dass die Initialisierung korrekt ist. In der zweiten for-Schleife berechnen wir die Werte von $f(i,j)$ iterativ. Die Korrektheit der Berechnung folgt direkt aus der Korrektheit des Orakels und der Beobachtung, dass das Orakel nur auf Werte zugreift, die bereits (in der Initalisierung oder der for-Schleife) berechnet wurden.

Laufzeit: Man sieht leicht, dass die erste for-Schleife Laufzeit $O(n)$ hat, die zweite $O\left(n^2\right)$ (da Orakel nur konstante Zeit benötigt). Das Maximum von n^2 Zahlen lässt sich offensichtlich in Zeit $O\left(n^2\right)$ finden, sodass die Gesamtlaufzeit $O\left(n^2\right)$ beträgt.

(b) Wir behaupten, dass das Orakel folgende Funktion verwenden kann:

$$f(i,j) = \begin{cases} 0, & \text{if } a_{i,j} = 0 \\ \min\{f(i-1,j), f(i,j-1), f(i-1,j-1)\} + 1, & \text{if } a_{i,j} = 1. \end{cases}$$

Der Fall $a_{i,j} = 0$ ist klar, da es in diesem Fall keine volle quadratische Teilmatrix mit rechter unterer Ecke $a_{i,j}$ gibt.

Falls $a_{i,j} = 1$, definieren wir $m = min\{f(i-1,j), f(i,j-1), f(i-1,j-1)\}$. Wir müssen nun zeigen, dass $f(i,j) \ge m+1$ und $f(i,j) \le m+1$ gilt, daraus folgt dann, dass $f(i,j) = m+1$. Hierzu bemerken wir, dass eine quadratische Teilmatrix der Grösse a mit rechter, unterer Ecke $a_{i,j}$ genau aus dem Eintrag $a_{i,j}$ und den Einträgen der quadratischen Teilmatrizen mit Grösse $a-1$ und unteren Ecken $a_{i-1,j}, a_{i,j-1}$ bzw. $a_{i-1,j-1}$ besteht. Deshalb folgt aus $f(i,j) = a$ insbesondere $m \ge a-1$, und ebenso $a \le m+1$, was zu zeigen war.

Punkteschema:

a) Initialisierung von f: **2P**
Iterative Berechnung (DP) der $f(i,j)$: **3P** (-1P falls Orakel inkorrekt aufgerufen/ nicht klar, welche Werte es benötigt)
Berechnung des Maximums: **1P**
Korrektheit (insbesondere bemerken, dass in der iterativen Berechnung nur auf bereits bekannte Werte zugegriffen wird): **1P**
Laufzeit: **1P**

b) Korrekte Formel: **2P** (1P bei kleinen Fehlern, 0P falls $f(i-1,j-1)$ nicht erwähnt wird)
Begründung: **2P** (bis zu 1P falls eine der beiden Richtungen $\le, \ge$ vergessen wurde)
Falls falsche Funktion f verwendet wurde (zum Beispiel falls 'max' in der Formel vorkommt): 0P

Lösungsvorschlag zu Aufgabe 3

Wir zeigen zuerst dass GRAD-BESCHRÄNKTER-SPANNBAUM in $\mathcal{NP}$ liegt. Falls (G, k) in der Sprache GRAD-BESCHRÄNKTER-SPANNBAUM liegt, dann besitzt G einen k-beschränkten Spannbaum T. Sei T das Zertifikat w. Da T im Graphen G enthalten ist ist $|T|$ polynomiell beschränkt in $|G|$. Gegeben T können wir in polyzeit überprüfen, ob G wirklich einen k-beschränkten Spannbaum hat. Indem wir mit breitensuche testen ob T ein Spannbaum ist und durch alle Knoten durchgehen um zu testen ob alle Knoten Grad kleinergleich k haben.

Um zu zeigen, dass GRAD-BESCHRÄNKTER-SPANNBAUM $\mathcal{NP}$-schwer ist zeigen wir HAMILTONPFAD $\leq_P$ GRAD-BESCHRÄNKTER-SPANNBAUM. Sei G eine Instanz für HAMILTONPFAD. Definiere $f(G) = (G, 2)$. Da ein Hamiltonpfad ein 2-bschränkter Spannbau ist und umgekehrt, gilt dass G genau dann in der Sprache HAMILTONPFAD liegt, wenn $f(G)$ in der Sprache GRAD-BESCHRÄNKTER-SPANNBAUM liegt.
Da f in polyzeit berechnet werden kann, gilt HAMILTONPFAD $\leq_P$ GRAD-BESCHRÄNKTER-SPANNBAUM. Da wir aus der Aufgaben stellung wissen, dass HAMILTONPFAD $\mathcal{NP}$-vollständig, also auch $\mathcal{NP}$-schwer ist, folgt daraus, dass auch GRAD-BESCHRÄNKTER-SPANNBAUM $\mathcal{NP}$-schwer ist.

Punkteschema:
GRAD-BESCHRÄNKTER-SPANNBAUM in NP: **5 Punkte**:

- **0.5P** für den Ansatz, zu testen, dass GRAD-BESCHRÄNKTER-SPANNBAUM in $\mathcal{NP}$,
- **1P** für die korrekte Beschreibung des Zeugen T (-0.5 falls die Grad-Beschränkung fehlt),
- **3P** für das Überprüfen der Eigenschaften (Spannbaum, $\Delta(T) \leq k$) inklusive einer Beschreibung wie getestet wird.
- **0.5P** für die Erkenntnis, dass wir den Zeugen in poly Zeit (in $|G|$) überprüfen können. (Auch falls die Beschreibung im vorherigen Punkt fehlte).

GRAD-BESCHRÄNKTER-SPANNBAUM ist NP-schwer: **5 Punkte**:

- **0.5P** für den Ansatz durch Reduktion zu zeigen das GRAD-BESCHRÄNKTER-SPANNBAUM $\mathcal{NP}$-schwer ist (Es mus klar werden warum die Reduktion auf eines der Probleme ausreichend ist),
- **1P** für die Auswahl des richtigen Problems,
- **3P** für das aufstellen der Transformation f und den Beweis der Äquvalenz,
- **0.5P** für die Feststelllug das die Transformation f in poly Zeit berechenbar ist/sein sollte (auch wenn keine explizite/eine falsche Transformation gegeben).

Lösungsvorschlag zu Aufgabe 4

a) Die Prozedur UPDATE(i, j) führt folgendes aus. Falls $a_{i+1,j} < a_{i,j+1}$ ist, so setzen wir $a_{i,j} = a_{i+1,j}$ und erhöhen i um 1, ansonsten setzen wir $a_{i,j} = a_{i,j+1}$ und erhöhen j um 1. Die Operation DECREASE-KEY(A) setzt zu Beginn $i = j = 1$ und führt wiederholt UPDATE(i, j) aus, solange $i \leq n$ und $j \leq n$ gilt. Damit nie auf nicht initalisierte Elemente zugegriffen wird, nehmen wir an, dass $a_{n+1,i} = a_{i,n+1} = \infty$ gilt für $1 \leq i \leq n$.

Da Update maximal $n + m$ Mal augeführt wird folgt unmittelbar die Laufzeit $\mathcal{O}(m + n)$.

Offensichtlich enthält das neu erzeugte Tableau A' dieselben Elemente wie A bis auf das kleinste. Die Eigenschaft, dass jede Spalte und jede Zeile monoton wachsend ist, ist equivalent dazu, dass $a_{i-1,j} \leq a_{i,j} \leq a_{i+1,j}$ und $a_{i,j-1} \leq a_{i,j} \leq a_{i,j+1}$ für alle $2 \leq i \leq m - 1$, $2 \leq j \leq n - 1$ gilt. Da diese Eigenschaft vor der Operation für alle $a_{i,j}$ erfülllt ist, müssen wir lediglich zeigen, dass sie auch für die $a_{i,j}$ erfüllt ist, welche während der Operation geändert wurden. Da bei jeder Ausführung von UPDATE(i, j) $a_{i,j}$ durch eine grössere Zahl ersetzt wird gilt $a_{i,j} \geq a_{i-1,j}$ und $a_{i,j} \geq a_{i,j-1}$. Da $a_{i,j}$ durch das kleinere von $a_{i+1,j}$ und $a_{i,j+1}$ ersetzt wird gilt danach auch $a_{i,j} \leq a_{i+1,j}$ und $a_{i,j} \leq a_{i,j+1}$.

b) Im folgenden setzen wir $a_{0,i} = a_{i,0} = -\infty$ für $1 \leq i \leq n$. Wir starten in einem Feld (i, j) mit $a_{i,j} = \infty$ und $a_{i-1,j} < \infty$ und $a_{i,j-1} < \infty$. So ein Feld kann sicherlich in $\mathcal{O}(n + m)$ gefunden werden indem wir von (n, n) nach oben und dann nach links laufen. Falls $\max\{a_{i-1,j}, a_{i,j-1}\} < x$ so setzen wir $a_{i,j} = x$ und beenden die Operation. Ansonsten, falls $a_{i-1,j} > a_{i,j-1}$ so setzen wir $a_{i,j} = a_{i-1,j}$ und vermindern i um 1, und sonst setzten wir $a_{i,j} = a_{i,j-1}$ und vermindern j um 1. Wir wiederholen dies, bis irgendwann x eingefügt wird.

Die Laufzeit ist $\mathcal{O}(n + m)$ da in jedem Schritt i oder j um eins kleiner wird, und x spätestens bei $i = j = 1$ eingefügt wird.

Offensichtlich enthält das neu erzeugte Tableau dieseleben Elemente und zusätzlich x. Um zu zeigen, dass die Zeilen und Spalten monoton wachsend sind, zeigen wir wie in a), dass $a_{i-1,j} \leq a_{i,j} \leq a_{i+1,j}$ und $a_{i,j-1} \leq a_{i,j} \leq a_{i,j+1}$ für jedes abgeänderte Element $a_{i,j}$ gilt. In jedem Schritt ersetzen wir $a_{i,j}$ durch ein kleineres Element, deshalb $a_{i,j}$ sicherlich kleiner als $a_{i+1,j}$ und $a_{i,j+1}$. Da wir $a_{i,j}$ jeweils durch das Maximum von $a_{i-1,j}$ und $a_{i,j-1}$ ersetzen, ist $a_{i,j}$ auch grösser als $a_{i-1,j}$ und $a_{i,j-1}$. Falls ein $a_{i,j}$ durch x ersetzen, so gilt $a_{i,j} \geq x$ da wir sonst x bereits bei der vorherigen Iteration eingesetzt hätten. Deshalb gilt $x \leq a_{i+1,j}$ und $x \leq a_{i,j+1}$. Da $x > \max\{a_{i-1,j}, a_{i,j-1}\}$, ist x grösser als die Nachbarn links und oben.

c) Wir fügen zuerst alle n^2 Elemente mit INSERT in ein $n \times n$ Young Tableau ein und führen, danach n^2 mal EXTRACT-MIN aus. Die Laufzeit ist $n^2 \cdot \mathcal{O}(n + n) = \mathcal{O}(n^3)$.

d) Setze $i = n,\ j = 1$. Wir wiederholen folgende Prozedur. Falls $x = a_{i,j}$, breche ab und gebe 'x ist in A' aus. Falls $x > a_{i,j}$, erhöhe j um 1. Falls $x < a_{i,j}$ vermindere i um 1. Falls $i = 0$ oder $j = n + 1$, so gebe 'x ist nicht in A' aus und breche ab.

Da in jeder Iteration i um 1 vermindert oder j um 1 erhöht wird, wird die Prozedur maximal $m + n$ Mal ausgeführt, was eine Laufzeit von $\mathcal{O}(m + n)$ impliziert.

Um Korrektheit zu zeigen, bemerken wir, dass falls x gefunden wird, die Ausgabe offensichtlich korrekt ist. Ansonsten gilt folgende Invariante: Nach dem Ausführen einer Iteration, gilt x befindet sich nicht in den Zeilen $i + 1, \ldots n$ und nich in den Spalten $1, \ldots j - 1$. Falls i um 1 vermindert wurde, so befindet sich x nicht in der Zeile $i + 1$, da $x < a_{i+1,j} \leq a_{i+1,j+1} \leq \ldots$ und x kann auch nicht weiter links in der Zeile stehen, da sich x nicht in den Spalten $1, \ldots, j - 1$ befindet). Falls j um 1 erhöht wurde, so befindet sich x nicht in der Spalte $j - 1$, da $x > a_{i,j-1} \geq a_{i-1,j-1} \geq \ldots$ und x kann auch nicht weiter unten in der Spalte stehen, da sich x nicht in den Zeilen $n, \ldots, i + 1$ befindet. Falls nun $i = 0$ oder $j = n + 1$ gilt, befindet sich x nicht in den Zeilen $1, \ldots n$ oder nicht in den Spalten $1, \ldots, n$ und somit nicht in A.

Punkteschema

a) Algorithmus 3 Punkte (Laufzeitanalyse fehlt -1 Punkt, kleinere Fehler -0.5 Punkte), Korrektheit 2 Punkte

b) Algorithmus 3 Punkte (Laufzeitanalyse fehlt -1 Punkt, kleinere Fehler -0.5 Punkte), Korrektheit 2 Punkte

c) Algorithmus 1 Punkt,
Laufzeitanalyse 1 Punkt

d) Algorithmus 3 Punkte (Laufzeitanalyse fehlt -1 Punkt, kleinere Fehler -0.5 Punkte), Korrektheit 3 Punkte

Lösungsvorschlag zu Aufgabe 5

Wir beobachten zuerst, dass alle Dreiecke die einen Knoten v enthalten mit Breitensuche startend in v gefunden werden können. In der ersten Runde der Breitensuche wird die Nachbarschaft von v besucht, falls eine Kante zwischen zwei Nachbarn von v existiert wird diese in der zweiten Runde der Breitensuche entdeckt, weitere Runden der Breitensuche müssen nicht berücksichtigt werden.

Etwas genauer, führen wir für jeden Knoten v folgende Prozedur durch: markiere alle Knoten in der Nachbarschaft $\Gamma(v)$ von v. Für jeden Knoten u in $\Gamma(v)$ betrachte alle Nachbarn. Für jeden Nachbarn von u, welcher bereits markiert ist erhöhe einen Zähler c um 1. Wir beobachten, dass so jedes Dreieck für jeden der drei Knoten zweimal gezählt wird. Demnach ist die Anzahl Dreieck $\frac{c}{6}$.

Die Laufzeit $\mathcal{O}(|V|(|V|+|E|))$ ist trivialerweise erfüllt, da dies genau $|V|$-mal einer (leicht) modifizierten Breitensuche entspricht, welche frühzeitig abgebrochen wird.

Eine genauere Analyse dieses Algorithmus ergibt eine (in den meisten Fällen) besser Laufzeit:

(i) um die Prozedur in jedem Knoten zu starten benötigen wir $\mathcal{O}(|V|)$,

(ii) um für jeden Knoten die Nachbarschaft zu markieren benötigen wir $\mathcal{O}(\sum_v \deg(v))$,

(iii) um die Nachbarschaften der markierten Knoten auf markierte Knoten zu durchsuchen benötigen wir $\mathcal{O}(\sum_v \deg(v)^2)$.

Wobei (iii) folgt gilt, da ein Knoten v insgesamt $\deg(v)$ mal markiert wird (für jeden seiner Nachbarn) und wir somit seine Liste $\deg(v)$ viel mal nach anderen markierten Knoten durchsuchen.

Da nun $\sum_v \deg(v) \leq \mathcal{O}(\sum_v \deg(v)^2)$ ist die Gesamtlaufzeit $\mathcal{O}(|V| + \sum_{v\in V} \deg(v)^2)$.

Punkteschema

1. **0 Punkte Lösung:**
   Da der Graph als Adj. Listen gespeichert ist, kann man nicht in konstanter Zeit überprüfen, ob zwei Knoten benachbart sind. Falls daher nicht explizit beschrieben wird, wie man die Nachbarn von v auf Kanten überprüft, erhält man sofort 0 Punkte (also auch korrekte Algorithmen, welche zu langsam sind).

   Man beachte insebsondere, dass ein Algorithmus der Form
   "für jedes $v \in V$ und all Paare $\{u, w\}$ von Nachbarn von V, überprfe, ob u und w verbunden sind"
   zu langsam ist. Die Laufzeit dieses "Brutforce" Algorithmus beträgt

   $$\mathcal{O}\left(\sum_v \deg(v)^2 \cdot \langle \text{überprüfe ob zwei Knoten verbunden sind} \rangle\right).$$

   Da der Graph als Adj. Liste gespeichert ist, ergibt dies bei $G = K_n$ z.B. eine Laufezeit von $\mathcal{O}(|V|^4)$, was um einen Faktor $|V|$ lagsamer ist als $\mathcal{O}(|V|(|V|+|E|))$.

2. **8 Punkte Lösung:** (um in diese Kategorie zu fallen muss sich die Lösung bereits entscheidend von der 0 Punkte Lösung differenzieren. Nicht anzugeben wie man Nachbarschaften überprüft gilt als falsch (0P.) und nicht als grosse Unklarheit.)

   Algorithmus 6 Punkte

   - -1 Punkt für kleine Unklarheit.
   - -2 Punkte für grosse Unklarheit.

   Laufzeitanalyse 2 Punkte

   - -1 Punkt für Zähler nicht durch 6.

- -2 Punkte für falsches zählen der Dreiecke.

3. **12 Punkte Lösung:**
   Algorithmus 6 Punkte
   - -1 Punkt für kleine Unklarheit.
   - -2 Punkte für grosse Unklarheit.

   Laufzeitanalyse 6 Punkte
   - -1 Punkt für Zähler nicht durch 6.
   - -2 Punkte für falsches zählen der Dreiecke.