Algorithmen und Datenstrukturen 2

WiSe 2020 · Prof. Dr. U. Meyer
30. März 2021
Klausurnummer: **0001**

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nachname (Druckschrift)**:** _____

Vorname (Druckschrift)**:** _____

Matrikelnummer: _____

Studiengang: _____

Bitte Hinweise beachten:

- Schreiben Sie Ihren **Namen *nur* auf dieses Titelblatt**. Sollten Sie Zusatzpapier bekommen, vermerken Sie darauf unbedingt die Klausurnummer **0001**.
- Die Klausur besteht aus den Aufgaben **1 – 10**.
- Merken oder notieren Sie sich Ihre Klausurnummer **0001**, da nur unter dieser Nummer die Ergebnisse veröffentlicht werden.
- Es dürfen nur **dokumentenechte Stifte** in den Farben blau und schwarz verwendet werden. Insb. ist die Nutzung von Tintenlöschern, Tipp-Ex u. Ä. untersagt.
  Zugelassene Hilfsmittel:
  1 Blatt DIN A4 mit handschriftlichen Notizen (beidseitig).
- Das Mitbringen nicht zugelassener Hilfsmittel stellt einen Täuschungsversuch dar und führt zum Nichtbestehen der Klausur. **Schalten Sie bitte deshalb alle elektronischen Geräte, insbesondere Handys und Smartwatches, vor Beginn der Klausur aus und packen Sie diese weg.**
- Werden zu einer Aufgabe zwei oder mehr Lösungen angegeben, so gilt die Aufgabe als nicht gelöst. Entscheiden Sie sich also immer für **eine** Lösung. Begründungen sind nur dann notwendig, wenn die Aufgabenformulierung dies verlangt.
- Die Klausur ist mit Sicherheit bestanden, wenn (ohne Bonifikation aus den Übungspunkten) mindestens **50%** der Höchstpunktzahl erreicht wird.
- Die Klausur dauert **180 Minuten**.

1 0001202 200013

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Diese Seite ist nur für den internen Gebrauch bestimmt.
Bitte nicht beschreiben.

Aufgabe	1	2	3	4	5	6	7	8	9	10
Erreichbar	6	6	7	8	12	14	12	13	11	11
Erreicht										

Klausur	Bonfikation

1 0 0 2 2 0 2 2 0 0 0 1 0

Aufgabe 1: Sortieren spezieller Eingaben — $2 + 2 + 2 = 6$ Punkt(e)

Betrachten Sie die folgenden Sequenzen der Länge n bestehend aus natürlichen Zahlen. Geben Sie **in Abhängigkeit der Länge n und dem Parameter x**, mit $1 < x < n$, asymptotisch exakt an, wie lange die entsprechenden Algorithmen zum Sortieren der Sequenz benötigen.

Anmerkung: Bubblesort terminiert nach einem Durchlauf ohne Vertauschung.

a)

$$x, 1, \ldots, x-1, \quad x+1, \ldots, n$$

d.h. die Folge ist sortiert bis auf das Element x, das sich am Anfang befindet.

Bubblesort: $\Theta($ n $)$

Insertionsort: $\Theta($ n $)$

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b)

$$1, \ldots, x-1, \quad x+1, \ldots, n, x$$

d.h. die Folge ist sortiert bis auf das Element x, das sich am Ende befindet.

Bubblesort: $\Theta($ $n \cdot (n-x)$ $)$

Insertionsort: $\Theta($ n $)$

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

c)

$$x, \ldots, 1, \quad 2 \cdot x, \ldots, 1 + x, \quad \ldots \quad , n, \ldots, n + 1 - x$$

d.h. absteigend sortierte Teilfolgen der Länge x, für ein x, sodass $n = x \cdot k$ für ein $k \in \mathbb{N}$ gilt.

Bubblesort: $\Theta($ $n \cdot x$ $)$

Insertionsort: $\Theta($ $n \cdot x$ $)$

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

Aufgabe 2: Sortieren kleiner Werte — 6 Punkt(e)

Gegeben seien n paarweise verschiedene, gekürzte Brüche der Form $\frac{a}{b}$ mit $a \in \{1, 2, 3, 4, 5\}$ und $b \in \mathbb{N}_{>0}$.

Die Brüche sind als Paare der Form (a, b) gegeben.

Geben Sie eine Transformation sowie, falls nötig, eine möglichst großzügige Einschränkung für b an, sodass eine solche Sequenz mit Radixsort inkl. der Transformation in Zeit $\mathcal{O}(n)$ sortiert werden kann.

Hinweis: Nehmen Sie an, dass arithmetische Operationen ($+$, $-$, $\cdot$, $/$) in konstanter Zeit ausgeführt werden können.

Lösungsskizze: Die Idee: Bringe alle Brüche auf den gleichen Zähler und sortiere absteigend nach Nenner.
Bilde (a, b) auf $x = \frac{60b}{a}$ ab. Da 60 durch alle möglichen Nenner a teilbar ist, ist x eine natürlich Zahl. Sortiere diese transformierte Sequenz *absteigend* (oder aufsteigend und kehre die Reihenfolge um) und bilde x auf $(60, x)$ ab. $(60, x)$ kann noch zu (a, b) gekürzt werden. Damit Radixsort solche Zahlen in Zeit $\mathcal{O}(n)$ sortieren kann, muss $b \in \mathcal{O}(n^c)$ für ein konstantes c für alle Elemente gelten.
Die Transformation eines Elements benötigt für beide Richtungen nur eine konstante Anzahl arithmetischer Operationen und läuft somit der Annahme entsprechend in konstanter Zeit. Die ganze Sequenz lässt sich also in linearer Zeit transformieren.

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

Aufgabe 3: Sortieralgorithmen verbessern — $2 + 3 + 2 = 7$ Punkt(e)

Die Laufzeit von Quicksort hängt maßgeblich von der Wahl des Pivotelements ab. Wir haben in der Vorlesung gesehen, dass sie im Worst-Case bei $\Theta(n^2)$ liegt. Selbst bei einer zufälligen Wahl des Pivotelements bleibt die Laufzeit im Worst-Case quadratisch.

Wir wollen nun wie folgt versuchen, deterministisch ein gutes Pivotelement zu finden:

`getPivot`(Array A)

1. Teile A in disjunkte Blöcke zu je $\sqrt{n}$ Elementen auf (der letzte Block muss nicht voll sein) und sortiere jeden Block mit Bubblesort.
2. Wähle aus jedem Block das mittlere Element aus und bilde mit diesen ein Array A'.
3. Sortiere A' mit Bubblesort.
4. Gib das mittlere Element von A' als Pivot zurück.

a) Welche Worst-Case-Laufzeit hat `getPivot`, wenn das Array A die Länge n hat?

$\Theta($ $n\sqrt{n}$ $)$

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Geben Sie eine möglichst gute, nicht-asymptotische untere Schranke für die Anzahl der Elemente aus A an, die höchstens so groß wie das durch `getPivot` gewählte Element sind. Begründen Sie Ihre Antwort kurz. Gehen Sie davon aus, dass alle Elemente paarweise verschieden sind.

Hinweis: Sie können davon ausgehen, dass $\sqrt{n}$ eine ganze Zahl ist.

Lösungsskizze: A' beinhaltet $\sqrt{n}$ Elemente, eines für jeden Median der $\sqrt{n}$ Blöcke in A. Links des gewählten Elements befinden sich also $\frac{1}{2}\sqrt{n}$ der Mediane. Zu jedem der Mediane existieren wieder $\frac{1}{2}\sqrt{n}$ Elemente die höchstens so groß sind wie der Median selbst. Insgesamt sind also mindestens $\frac{1}{2}\sqrt{n} \cdot \frac{1}{2}\sqrt{n} = \frac{1}{4}n$ Elemente höchstens so groß wie das gewählte Element.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Geben Sie eine Rekursionsgleichung für die Worst-Case-Laufzeit von Quicksort an, wenn für die Pivotwahl `getPivot` verwendet wird. Sie müssen die Rekursionsgleichung nicht ausrechnen.

Hinweis: Sie können $f(n)$ für die Worst-Case-Laufzeit von `getPivot` und a für den Anteil der Elemente, die höchstens so groß wie das gewählte Pivotelement sind, verwenden.

$T(n) =$ $T(\frac{n}{4}) + T(\frac{3n}{4}) + \Theta(n\sqrt{n})$ oder $T(n) = T(an) + T((1-a)n) + \Theta(f(n))$

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

Seien L_1 und L_2 rekursiv aufzählbare Sprachen und sei

$$L_1 \circ L_2 = \{uv \ : \ u \in L_1, \ v \in L_2\}$$

die Sprache der konkatenierten Wörter aus L_1 und L_2.

Zeigen oder widerlegen Sie: $L_1 \circ L_2$ ist rekursiv aufzählbar.

Lösungsskizze: $L_1 \circ L_2$ ist rekursiv aufzählbar.
Da L_1 und L_2 rekursiv aufzählbar sind, existieren Turingmaschinen T_1 und T_2, die die jeweiligen Worte in einer bestimmten Reihenfolge auf das Band schreiben. Seien $u_0, u_1, \ldots$ die Wörter aus L_1 und $v_0, v_1, \ldots$ entsprechend für L_2. Dann gibt es auch eine Turingmaschine T, die alle Kombinationen aus zwei Wörtern wie folgt durchprobieren:
Beginnend bei $k = 1$, zähle alle Wörter $u_i v_{k-i}$ auf, mit $0 \leq i < k$, $u_j \in L_1$ und $v_j \in L_2$.
Bei dieser Reihenfolge alle möglichen Kombinationen durchlaufen, auch wenn eine der Sprachen unendlich viele Wörter beinhalten sollte.

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

Gegeben sei die folgende Turingmaschine $M = (Q, \Sigma, \delta, q_0, \Gamma, F)$ mit

$$Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{0, 1\}$$
$$\Gamma = \{B\} \cup \Sigma$$
$$F = \{q_2\}$$

und der Übergangsfunktion δ definiert als

$$(q_0, B) \mapsto (q_2, B, rechts)$$
$$(q_0, 1) \mapsto (q_1, 1, rechts)$$
$$(q_1, 1) \mapsto (q_2, 1, rechts)$$
$$(q_2, 1) \mapsto (q_1, 1, rechts)$$
$$(q_2, 0) \mapsto (q_1, 0, bleib)$$

a) Geben Sie ein Wort $w \in \Sigma^*$ an, das von M akzeptiert wird.

$w =$ 11

⇑⇑⇑ _____ / 1 Punkt(e) ⇑⇑⇑

b) Welche Sprache $L \subseteq \Sigma^*$ wird von M akzeptiert?

Lösungsskizze: M akzeptiert alle Binärstrings gerader Länge, die ausschließlich aus Einsen bestehen, und den leeren String.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

c) Hält M auf allen Eingaben? Begründen Sie Ihre Antwort und geben Sie ggf. eine Eingabe an, auf der M nicht hält.

Lösungsskizze: M hält auf allen Eingaben.
Immer wenn eine 1 gelesen wird bewegt sich der Lese-Schreib-Kopf nach rechts. Nach endlich vielen Schritten ist das Ende der Eingabe erreicht oder es wird eine 0 gelesen. Sobald etwas anderes als eine 1 gelesen wird, macht die Turingmaschine noch maximal einen Schritt.

- Wenn M im Zustand q_0 ist und ein B gelesen wird wechselt sie nach q_2 und geht nach rechts. Da rechts von einem B wieder ein B stehen muss und δ auf (q_2, B) nicht definiert ist hält sie.
- Wenn M im Zustand q_2 ist und eine 0 gelesen wird wechselt sie nach q_1 und bleibt an der stelle. Da δ auf $(q_1, 0)$ nicht definiert ist hält sie.
- In allen anderen Fällen ist δ nicht definiert und M hält ohne einen weiteren Schritt.

⇑⇑⇑ _____ / 5 Punkt(e) ⇑⇑⇑

d) Gibt es Eingaben, auf denen M in einem nicht-akzeptierenden Zustand hält? Geben Sie ein Beispiel an oder begründen Sie, warum es keine solche Eingabe gibt.

Lösungsskizze: Für die Eingabe `0` ist δ nicht definiert und M hält ohne zu akzeptieren.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

Aufgabe 6: Scheduling — $3 + 4 + 7 = 14$ Punkt(e)

Gegeben seien drei identische Maschinen M_1, M_2 und M_3 und sechs Aufgaben $1, \ldots, 6$ mit den Laufzeiten $t_1, \ldots, t_6$:

i	1	2	3	4	5	6
t_i	2	13	7	5	10	3

Alle Aufgaben sollen auf die Maschinen verteilt werden mit dem Ziel, den Makespan (frühester Zeitpunkt, zu dem alle Aufgaben fertiggestellt sind) zu minimieren.

Für die folgenden Teilaufgaben gilt: Falls beide Maschinen die gleiche Last haben, dürfen Sie die aktuelle Aufgabe einer beliebigen Maschine zuteilen.

a) Verteilen Sie die Aufgaben an die Maschinen unter Verwendung der Greedy-Offline-Heuristik für das Lastverteilungsproblem aus der Vorlesung. Geben Sie dabei für jede Aufgabe an, welcher Maschine sie zugeteilt wird. Geben Sie den resultierenden Makespan an.

Lösungsskizze: Permutationen der Aufgaben: $2, 5, 3, 4, 6, 1$ (mit Laufzeitenreihenfolge $13, 10, 7, 5, 3, 2$). Die Greedy-Offline-Heuristik verteilt die Aufgaben wie folgt:
Aufgabe 2 mit $t_2 = 13$ an M_1 (aktuelle Last 13),
Aufgabe 5 mit $t_5 = 10$ an M_2 (aktuelle Last 10),
Aufgabe 3 mit $t_3 = 7$ an M_3 (aktuelle Last 7),
Aufgabe 4 mit $t_4 = 5$ an M_3 (aktuelle Last 12),
Aufgabe 6 mit $t_6 = 3$ an M_2 (aktuelle Last 13).
Aufgabe 1 mit $t_1 = 2$ an M_3 (aktuelle Last 14).
m1: [13] m2: [10, 3] m3: [7, 5, 2].
Der resultierende Makespan ist also 14.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Berechnet die Greedy-Offline-Heuristik auf dieser Instanz eine optimale Lösung? Beweisen Sie Ihre Antwort.

Lösungsskizze: Ja, der Makespan 14 ist optimal. Da wir nur Aufgaben mit ganzzahligen Laufzeiten haben, muss auch der Makespan ganzzahlig sein. Angenommen, 14 wäre nicht optimal, dann müsste ein Makespan von höchstens 13 erreicht werden können. Wenn der Makespan 13 ist, dann können die Maschinen höchstens $3 \cdot 13 = 39$ Zeiteinheiten abarbeiten. Da die Gesamtlaufzeit aller Aufgaben beträgt jedoch 40 beträgt ist ein Makespan von 13 oder besser nicht möglich.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

c) Nun stehen nur noch zwei Maschinen M_1 und M_2 zur Verfügung und es sollen folgende Aufgaben verteilt werden:

i	1	2	3	4	5
t_i	2	13	7	5	10

Geben Sie eine Reihenfolge der Aufgaben an, sodass mit der Greedy-Online-Heuristik für das Lastverteilungsproblem aus der Vorlesung der Makespan 25 oder schlechter ist. Geben Sie dabei für jede Aufgabe an, welcher Maschine sie zugeteilt wird.

Lösungsskizze: Eine mögliche Permutationen der Aufgaben: $5, 3, 4, 1, 2$ (mit Laufzeitenreihenfolge $10, 7, 5, 2, 13$). Die Greedy-Online-Heuristik verteilt die Aufgaben wie folgt:
Aufgabe 5 mit $t_5 = 10$ an M_1 (aktuelle Last 10),
Aufgabe 3 mit $t_3 = 7$ an M_2 (aktuelle Last 7),
Aufgabe 4 mit $t_4 = 5$ an M_2 (aktuelle Last 12),
Aufgabe 1 mit $t_1 = 2$ an M_1 (aktuelle Last 12),
Aufgabe 2 mit $t_2 = 13$ an M_1 (aktuelle Last 25).

Der berechnete Makespan ist also 25.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

Aufgabe 7: NP-Vollständigkeit und Approximation — $3+3+3+3=12$ Punkt(e)

Entscheiden Sie, ob die folgenden Aussagen stimmen, und begründen Sie kurz Ihre Antwort. Es ist kein Beweis gefordert.

a) Sei $\mathcal{B}$ die Klasse der ungerichteten Bäume mit mindestens drei Knoten. Es ist ein $\mathcal{NP}$-vollständiges Problem zu entscheiden, ob ein gegebener Baum $B \in \mathcal{B}$ einen Hamiltonischen Kreis enthält.

Lösungsskizze: Falsch. Ein Baum ist ein kreisfreier Graph, daher ist die Antwort dieses Entscheidungsproblems bei jeder Eingabe „nein“und lässt sich in Zeit $\mathcal{O}1$ entscheiden.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Für das Rucksackproblem gibt es einen $\frac{2}{3}$-approximativen Algorithmus mit polynomieller Laufzeit in n.

Lösungsskizze: Falsch, der Approximationsfaktor $\frac{2}{3}$ würde bedeuten, dass der Algorithmus eine bessere Lösung als eine optimale Lösung findet. Da die optimale Lösung die bestmögliche ist, geht das nicht.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Angenommen $\mathcal{P} \neq \mathcal{NP}$. Dann gibt es eine polynomielle Reduktion von KNF-SAT auf die leere Sprache.

Lösungsskizze: Falsch. Wenn $\mathcal{P} \neq \mathcal{NP}$, gibt es für KNF-SAT keinen polynomiellen Algorithmus, wohingegen für die leere Sprache schon. Eine polynomielle Reduktion würde einen polynomiellen Algorithmus für KNF-SAT implizieren.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

d) Falls es einen Algorithmus mit polynomieller Laufzeit für VertexCover gibt, dann ist $\mathcal{P} = \mathcal{NP}$.

Lösungsskizze: Wahr, falls der Algorithmus deterministisch ist. Dann gibt es für alle Probleme aus $\mathcal{NP}$ einen deterministischen Polynomialzeitalgorithmus: Erst die polynomielle Reduktion zum $\mathcal{NP}$-vollständigen Problem VertexCover und dann das Lösen des VertexCover in Polynomialzeit.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

Im gewichtetem Vertex-Cover-Problem GVC ist ein ungerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$ und Knotengewichten w_v für jeden Knoten $v \in V$ gegeben. Desweiteren ist eine Gewichtsschranke W gegeben.

Gesucht ist eine Knotenüberdeckung $C \subseteq V$, sodass

$$\sum_{i \in C} w_i \leq W.$$

Im gewichtetem Set-Cover-Problem GSC sind m endliche Mengen $A_1, \ldots, A_m$, wobei Menge A_i das Gewicht s_i hat, und eine Gewichtsschanke S gegeben. Gesucht ist eine Indexmenge $I \subseteq \{1, \ldots, m\}$, sodass

$$\bigcup_{i \in I} A_i = \bigcup_{j \in \{1,\ldots,m\}} A_j \qquad \text{und} \qquad \sum_{i \in I} s_i \leq S.$$

a) Zeigen Sie, dass GVC $\in \mathcal{NP}$ ist.

Lösungsskizze: Rate eine Knotenmenge $C \subseteq V$. Prüfe, ob für jede Kante $e \in E$ gilt: $e \cap C \neq \emptyset$. Prüfe, ob $\sum_{i \in C} w_i \leq W$. Falls alle diese Bedingungen erfüllt sind, akzeptiere, falls nein, verwerfe. Die Überprüfung ist in Laufzeit $\mathcal{O}(|E| \cdot |V| + |V|)$ möglich, also in polynomieller Laufzeit.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Zeigen Sie, dass GSC $\in \mathcal{NP}$ ist.

Lösungsskizze: Rate eine Indexmenge $I \subseteq \{1, \ldots, m\}$. Berechne $\bigcup_{i \in I} A_i$ und $\bigcup_{j \in \{1,\ldots,m\}} A_j$ und prüfe, ob sie gleich sind. Prüfe, ob $\sum_{i \in I} s_i \leq S$. Falls alle diese Bedingungen erfüllt sind, akzeptiere, falls nein, verwerfe. Die Überprüfung ist in Laufzeit $\mathcal{O}(\sum_{j \in \{1,\ldots,m\}} |A_j| + m)$ möglich, also in polynomieller Laufzeit.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Zeigen Sie, dass es eine polynomielle Reduktion GVC $\leq_p$ GSC gibt. Zeigen Sie die Korrektheit dieser Reduktion.

Lösungsskizze:
Transformation: Sei $G = (V, E)$ der Graph, w_i die Knotengewichte und W die Schranke als Eingabe für GVC. Wir setzen für jeden Knoten $i \in V$:

- $A_i := \{e \in E : i \in e\}$
- $s_i := w_i$

Zusätzlich sei $S := W$. Damit ist übrigens $n = m$.
Laufzeit: Bilden von einem A_i und s_i dauert $\mathcal{O}(|E| + 1)$, Setzen von S ist in $\mathcal{O}(1)$. Damit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(|V| \cdot |E|)$. Die Laufzeit ist also polynomiell in der Eingabegröße des Graphen G.
Korrektheit:
$C \subseteq V$ ist eine Knotenüberdeckung.
$\Leftrightarrow$ Jede Kante $e \in E$ besitzt einen Endpunkt in C.
$\Leftrightarrow$ Die Indexmenge C hat folgende Eigenschaft: $\bigcup_{i \in C} A_i = E = \bigcup_{j \in \{1,\ldots,m\}} A_j$.
Da $w_i = s_i \; \forall i$ und $W = S$ gilt:

$$\sum_{i \in C} w_i \leq W \Leftrightarrow \sum_{i \in C} s_i \leq S$$

Somit ist C genau dann eine Lösung für GVC , wenn C auch eine Lösung für GSC ist.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

Aufgabe 9: Approximativer Durchmesser — $3 + 4 + 4 = 11$ Punkt(e)

Gegeben sei ein ungerichteter, ungewichteter, zusammenhängender Graph $G = (V, E)$.

Bei einem ungewichteten Graphen messen wir die Länge eines Weges durch die Anzahl der Kanten entlang des Weges. Für je zwei Knoten $u, v \in V$ sei $d(u, v)$ die Länge des kürzesten Weges von u nach v. Der Durchmesser D eines Graphen ist die Länge des längsten kürzesten Weges im Graphen:

$$D = \max_{u,v \in V} d(u, v).$$

Gegeben sei folgender Algorithmus, der einen approximativen Durchmesser D_a berechnet: Wir führen Breitensuche mit Startknoten 1 aus und bestimmen die Tiefe t des resultierenden Baumes. Die Ausgabe ist $D_a = 2 \cdot t$.

a) Zeigen Sie, dass der Algorithmus nicht immer die optimale Lösung berechnet.

Lösungsskizze: Betrachte den vollständigen Graphen mit 3 Knoten K_3: Breitensuche hat Tiefe 1, also ist $D_a = 2$, dabei kann jeder Knoten jeden anderen Knoten durch nur eine Kante erreichen, folglich ist $D = 1$.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Zeigen Sie, dass $D_a \leq 2 \cdot D$ ist.

Lösungsskizze: $D_a = 2 \cdot t$. Es ist $D \geq t$, da der kürzeste Weg von Knoten 1 zu einem tiefsten Knoten v die Länge $d(1, v) = t$ hat. Damit ist $2D \geq 2t = D_a$.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

c) Zeigen Sie, dass $D_a \geq D$ ist.

Lösungsskizze: Da t die Tiefe des Baumes ist, kann man jeden anderen Knoten durch Hochklettern der Baumkanten zur Wurzel und dann Hinabsteigen zu dem eigentlichen Knoten erreichen. Beide Wege haben Länge höchstens t. Daher ist $D \leq t + t = D_a$.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

Aufgabe 10: Clique — $8 + 3 = 11$ Punkt(e)

Gegeben sei ein ungerichteter, zusammenhängender Graph $G = (V, E)$ mit $|V| \geq 2$.

Eine Clique C in G ist eine Teilmenge von V, innerhalb derer alle Knotenpaare durch eine Kante miteinander verbunden sind. Das Ziel ist, die Anzahl der Knoten in C zu maximieren.

a) Beschreiben Sie einen effizienten Greedyalgorithmus, der eine nicht-erweiterbare Clique ausgibt. Begründen Sie die Korrektheit Ihres Algorithmus und zeigen Sie, dass ihr Algorithmus effizient ist.

Lösungsskizze:
Algorithmus:
$C = \emptyset$;
`Solange` $V \neq \emptyset$ `{`
  `Füge einen Knoten` $v \in V$ `zu` C `hinzu;`
  `Entferne` v `und alle Knoten nicht adjazent zu` v `aus` V`;`
`};`
`Ausgabe:` C`;`

Korrektheit: Wir fügen zuerst einen beliebigen Knoten v zu C hinzu. Durch das Löschen der nicht adjazenten Knoten wird sichergestellt, dass nur Knoten zu C hinzugefügt werden, die zu allen vorher eingefügten Knoten in C adjazent sind. Somit ist C eine Clique. C ist nicht erweiterbar, da wir den Vorgang wiederholen bis $V = \emptyset$, also kein Knoten mehr übrig ist, der zu allen Knoten in C adjazent ist.
Laufzeit: Der Algorithmus hat höchstens $|V|$-viele Schleifendurchläufe, wobei jeder Durchlauf höchstens $\mathcal{O}(|V| + |V|^2)$ zum Löschen eines Knoten und der nicht zu ihm adjazenten Knoten braucht. Damit ist die Laufzeit $\mathcal{O}(|V|^3)$, also ist er effizient.

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

b) Zeigen Sie, dass für eine optimale Clique C_{opt} in G und eine nicht-erweiterbare Clique C in G gilt:

$$\frac{|C_{opt}|}{|C|} \leq \frac{|V|}{2}$$

Lösungsskizze: Offensichtlich ist $|C_{opt}| \leq |V|$, denn mehr als $|V|$ paarweise verbundene Knoten kann es in einem Graphen mit $|V|$ Knoten nicht geben. Da $|V| \geq 2$ und der Graph zusammenhängend ist, sind in einer nicht-erweiterbaren Clique immer mindestens zwei Knoten, da ein Knoten immer zu mindestens einem anderen Knoten verbunden ist. Es ist also $|C| \geq 2$. Daraus folgt $\frac{|C_{opt}|}{|C|} \leq \frac{|C_{opt}|}{2} \leq \frac{|V|}{2}$.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Hinweis auf Seite 15 platziert wurde.