Algorithmen und Datenstrukturen 2

WiSe 2020 · Prof. Dr. U. Meyer
23. Februar 2021
Klausurnummer: **0000**

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 **0000**.
- Die Klausur besteht aus den Aufgaben **1 − 10**.
- Merken oder notieren Sie sich Ihre Klausurnummer **0000**, 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, Tippex u. Ä. untersagt.
  Zugelassene Hilfsmittel:
  1 Blatt DIN A4 mit handschriftlichen Notizen (zweiseitig).
- 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 001202 100009

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	13	12	9	11	16
Erreicht										

Klausur	Bonfikation

1 0 0 2 2 0 2 1 0 0 0 0 6

Aufgabe 1: Sortieren spezieller Eingaben — $3 + 3 = 6$ Punkt(e)

Betrachten Sie die folgenden Sequenzen der Länge n bestehend aus natürlichen Zahlen. Geben Sie in Abhängigkeit der Länge n asymptotisch exakt an, wie lange die entsprechenden Algorithmen zum Sortieren der Sequenz benötigen.

Anmerkungen: Bubblesort terminiert nach einem Durchlauf ohne Vertauschung. Sie können außerdem davon ausgehen, dass $\log_2 n$ und $\sqrt{n}$ ebenfalls natürliche Zahlen sind.

a)

$$\log_2 n, \ldots, 1, \;\; 2 \cdot \log_2 n, \ldots, 1 + \log_2 n, \;\; \ldots \;\; , n, \ldots, n + 1 - \log_2 n$$

d.h. absteigend sortierte Teilfolgen der Länge $\log_2 n$.

Bubblesort: $\Theta($ $n \log n$ $)$

Insertionsort: $\Theta($ $n \log n$ $)$

Selectionsort: $\Theta($ n^2 $)$

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b)

$$1, \ldots, \sqrt{n} - 1, \;\; \sqrt{n} + 1, \ldots, n, \sqrt{n}$$

d.h. die Folge ist sortiert bis auf das Element $\sqrt{n}$, das sich am Ende befindet.

Bubblesort: $\Theta($ n^2 $)$

Insertionsort: $\Theta($ n $)$

Selectionsort: $\Theta($ n^2 $)$

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

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

Gegeben seien n natürliche Zahlen der Form N^{k^N} für $k \leq n$ und ein beliebig großes, aber bekanntes, $N \in \mathbb{N}$.

Gibt es für solche Sequenzen eine Transformation, die jedem Element einen neuen Schlüssel zuordnet, sodass die transformierte Sequenz in Zeit $\mathcal{O}(n)$ sortiert werden kann? Geben Sie eine Transformation sowie einen Sortieralgorithmus an, mit dem dies möglich ist, oder begründen Sie, warum eine solche Transformation nicht existiert.

Lösungsskizze: Es ist möglich: Transformiere dazu die Eingabe $\sqrt[N]{\cdot} \circ \log_N : N^{k^N} \to k$, (erst $\log_N$ dann $\sqrt[N]{\cdot}$) sortiere die Exponenten mit Radixsort in Linearzeit (Basis n) und transformiere die sortierte Sequenz zurück.

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

Aufgabe 3: Sortieralgorithmen verbessern — 7 Punkt(e)

Bubblesort ist mit seiner Worst-Case Laufzeit von $\Theta(n^2)$ normalerweise nicht der Sortieralgorithmus der Wahl. Mit den Erfahrungen aus Quicksort wollen wir versuchen, Bubblesort zu verbessern.

Sei A eine Liste von n Elementen. Sie können davon ausgehen, dass n eine Zweierpotenz ist.

QuickBubbleSort:

- Falls $n = 1$: A ist sortiert, `return`.
- Falls $n > 1$:
  1. Sortiere $A[1, \ldots, \frac{n}{2}]$ und $A[\frac{n}{2} + 1, \ldots, n]$ rekursiv mit QuickBubbleSort.
  2. Mische die beiden sortieren Teilfolgen mit Bubblesort.

Die Hoffnung: Da QuickBubbleSort die Liste in der Mitte teilt, hat es eine Rekursionstiefe von $\log n$ und somit eine Gesamtlaufzeit von $\mathcal{O}(n \log n)$.

Ist QuickBubbleSort wirklich schneller als Bubblesort? Begründen Sie Ihre Antwort.

Lösungsskizze: Das letzte Mischen der sortierten Teilfolgen mit Bubblesort benötigt im Worst-Case bereits $\Theta(n^2)$ Schritte, wenn z.B. die kleinste Zahl im rechten Teilarray ist. Diese muss $\frac{n}{2}$ Positionen nach links verschoben werden. Dazu kommen noch die anderen Mischphasen der rekursiven Durchläufe. Damit ist QuickBubbleSort nicht schneller als Bubblesort und läuft insbesondere nicht in $\mathcal{O}(n \log n)$.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

Aufgabe 4: Entscheidbarkeit — 8 Punkt(e)

Betrachten Sie folgende Aussage:

> Wenn L eine entscheidbare Sprache ist, ist auch ihr Komplement $\overline{L}$ entscheidbar. Da entscheidbare Sprachen rekursiv aufzählbar sind, folgt: Ist eine Sprache L rekursiv aufzählbar, dann auch ihr Komplement $\overline{L}$.

Ist diese Aussage korrekt? Begründen Sie Ihre Antwort.

Lösungsskizze: Diese Aussage ist nicht korrekt, da aus rekursiver Aufzählbarkeit nicht auch Entscheidbarkeit folgt. Beispielsweise ist das Halteproblem H rekursiv aufzählbar, nicht aber das Komplement $\overline{H}$.

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

Aufgabe 5: Turingmaschine

$1 + 4 + 5 + 2 = 12$ Punkt(e)

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

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

und der Übergangsfunktion δ definiert als

$$
\begin{aligned}
(q_0, \alpha) &\mapsto (q_0, \alpha, rechts) \quad \forall\, \alpha \in \Sigma \\
(q_0, B) &\mapsto (q_1, B, links) \\
(q_1, 1) &\mapsto (q_0, 1, rechts) \\
(q_1, 0) &\mapsto (q_2, 0, bleib)
\end{aligned}
$$

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

$w =$ 0

⇑⇑⇑ _____ / 1 Punkt(e) ⇑⇑⇑

b) Welche Sprache wird von M akzeptiert?

Lösungsskizze: M akzeptiert alle Worte der Form `(0|1)*0`, d.h. alle Binärstrings, die auf Null enden.

⇑⇑⇑ _____ / 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 nur auf Eingaben der Form `(0|1)*0` und der leeren Eingabe. Bei allen anderen Eingaben wird der Lese-Schreib-Kopf ganz nach rechts, bis zum ersten Blank-Symbol bewegt und bewegt sich dann immer eine Zelle nach links und zurück und wechselt dabei zwischen den Zuständen q_1 und q_0.

⇑⇑⇑ _____ / 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: Auf der leeren Eingabe hält M im Zustand q_1.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

1 0 0 7 2 0 2 1 0 0 0 0 1

Aufgabe 6: Scheduling — $3 + 3 + 7 = 13$ Punkt(e)

Gegeben seien zwei identische Maschinen M_1 und M_2 und fünf Aufgaben $1, \ldots, 5$ mit den Laufzeiten $t_1, \ldots, t_5$:

i	1	2	3	4	5
t_i	9	8	4	5	10

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

Betrachten Sie die folgenden zwei Strategien:

1. Die Aufgaben werden nach absteigenden Laufzeiten sortiert. Die Aufgaben werden dann nacheinander auf die Maschinen verteilt, wobei die aktuelle Aufgabe der Maschine mit der bisher geringsten Last zugeteilt wird.
2. Die Aufgaben werden in irgendeiner Reihenfolge nacheinander auf die Maschinen verteilt, wobei die aktuelle Aufgabe der Maschine mit der bisher geringsten Last zugeteilt wird.

Falls beide Maschinen die gleiche Last haben, dürfen Sie die aktuelle Aufgabe einer beliebigen Maschine zuteilen.

a) Bestimmen Sie eine optimale Lösung. Zeigen Sie die Optimalität dieser Lösung.

Lösungsskizze: M_1 erhält Aufgaben $1, 3, 4$ mit Gesamtlaufzeit von 18. M_2 erhält Aufgaben $2, 5$ mit Gesamtlaufzeit von 18. Die Lösung ist optimal, da die Last gleichverteilt auf allen Maschinen ist.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Verteilen Sie die Aufgaben an die Maschinen unter Verwendung der 1. Strategie. Geben Sie dabei für jede Aufgabe an, welcher Maschine sie zugeteilt wird. Berechnet die 1. Strategie eine optimale Lösung? Begründen Sie Ihre Antwort.

Lösungsskizze: Nein. Sortiert nach absteigenden Laufzeiten ist die Reihenfolge der Aufgaben $5, 1, 2, 4, 3$. Strategie 1 verteilt die Aufgaben wie folgt:
Aufgabe 5 mit $t_5 = 10$ an M_1 (aktuelle Last 10),
Aufgabe 1 mit $t_1 = 9$ an M_2 (aktuelle Last 9),
Aufgabe 2 mit $t_2 = 8$ an M_2 (aktuelle Last 17),
Aufgabe 4 mit $t_4 = 5$ an M_1 (aktuelle Last 15),
Aufgabe 3 mit $t_3 = 4$ an M_1 (aktuelle Last 19).

Der berechnete Makespan ist also 19, wobei 18 optimal wäre.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Hier ist eine Kopie der Tabelle:

i	1	2	3	4	5
t_i	9	8	4	5	10

Geben Sie eine Reihenfolge der Aufgaben an, sodass mit der 2. Strategie der Makespan 23 oder schlechter ist. Geben Sie dabei für jede Aufgabe an, welcher Maschine sie zugeteilt wird.

Lösungsskizze: Eine mögliche Permutationen der Laufzeiten: $1, 2, 4, 3, 5$ (mit Laufzeitenreihenfolge $9, 8, 5, 4, 10$). Die 2. Strategie verteilt die Aufgaben wie folgt:
Aufgabe 1 mit $t_1 = 9$ an M_1 (aktuelle Last 9),
Aufgabe 2 mit $t_2 = 8$ an M_2 (aktuelle Last 8),
Aufgabe 4 mit $t_4 = 5$ an M_2 (aktuelle Last 13),
Aufgabe 3 mit $t_3 = 4$ an M_1 (aktuelle Last 13),
Aufgabe 5 mit $t_5 = 10$ an M_1 (aktuelle Last 23).

Der berechnete Makespan ist also 23.

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

Lösungsskizze: Ja, da der Approximationsfaktor $3/2 = 1 + 1/2$ ist, setze $\varepsilon = 1/2$. Der Skalierungsalgorithmus aus der Vorlesung bietet für $\varepsilon = 1/2$ eine Laufzeit von $\mathcal{O}(n^3/\varepsilon)$. Wenn $1/\varepsilon$, hier 2, konstant ist, hat $1/\varepsilon$ keinen asymptotischen Einfluss auf die Laufzeit.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Sei SP das Problem zu entscheiden, ob eine Sequenz bestehend aus n natürlichen Zahlen aufsteigend sortiert ist. Dann gibt es eine polynomielle Reduktion von SP auf KNF-SAT.

Lösungsskizze: Wahr. Da das SP in polynomieller Zeit lösbar ist (gehe ein Mal durch die Sequenz durch und prüfe, ob die nachfolgende Zahl größer ist als die aktuelle Zahl), gehört es zur Klasse $\mathcal{P} \subseteq \mathcal{NP}$. Da es auch zu $\mathcal{NP}$ gehört und KNF-SAT $\mathcal{NP}$-vollständig ist, gibt es eine polynomielle Reduktion.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Angenommen $\mathcal{P} \neq \mathcal{NP}$. Dann gibt es eine polynomielle Reduktion von CLIQUE auf 3-SAT.

Lösungsskizze: Wahr, unabhängig von $\mathcal{P} \neq \mathcal{NP}$, da beides $\mathcal{NP}$-vollständige Probleme sind.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

d) Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit nicht-negativen Kantengewichten und ein Knoten $s \in V$. Es gibt einen effizienten Algorithmus, der zu jedem Knoten $v \in V$ einen Distanzwert berechnet, der höchstens doppelt so groß ist wie die Länge des kürzesten Weges von s nach v.

Lösungsskizze: Wahr. Dijkstra löst das kürzeste Wege Problem exakt also ist der Algorithmus auch 2-approximativ.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

Aufgabe 8: SubsetSum — $2 + 7 = 9$ Punkt(e)

Im $\mathcal{NP}$-vollständigen SubsetSum Problem ist die Eingabe durch eine Menge M von n ganzen Zahlen gegeben. Die Eingabe gehört genau dann zur Sprache SubsetSum, wenn eine nicht-leere Teilmenge $M' \subseteq M$ existiert mit folgender Eigenschaft:

$$\sum_{m \in M'} m = 0.$$

Wir wollen eine Einschränkung betrachten: Im 3-Sum Problem ist die Eingabe auch durch eine Menge M von n ganzen Zahlen gegeben. Die Eingabe gehört genau dann zur Sprache, wenn 3 paarweise verschiedene Zahlen $a, b, c \in M$, das heißt $a \neq b, a \neq c$ und $b \neq c$, mit folgender Eigenschaft existieren:

$$a + b + c = 0.$$

a) Zeigen Sie, dass 3-Sum zur Sprache $\mathcal{NP}$ gehört.

Lösungsskizze: Wir raten 3 Zahlen a, b, c aus M und überprüfen, ob $a + b + c = 0$ ergibt. Die Überprüfung ist in konstanter Laufzeit möglich.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Alice behauptet:

> „3-Sum ist so ähnlich zu SubsetSum, also ist 3-Sum bestimmt auch $\mathcal{NP}$-vollständig."

Ist Alice's Behauptung wahr?
Zeigen Sie die $\mathcal{NP}$-Vollständigkeit mit SubsetSum als Reduktionspartner oder geben Sie einen effizienten deterministischen Algorithmus an, der 3-Sum entscheidet.

Lösungsskizze: 3-Sum ist nicht $\mathcal{NP}$-vollständig, außer wenn $\mathcal{P} = \mathcal{NP}$. Prüfe für alle $\binom{n}{3} = \mathcal{O}(n^3)$ Zahlenkombinationen aus 3 Zahlen, ob sie als Summe 0 ergeben. Da die Prüfung in Konstantzeit möglich ist, hat der Algorithmus eine Laufzeit von $\mathcal{O}(n^3)$.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

Aufgabe 9: Das Freundebuch — $2 + 9 = 11$ Punkt(e)

Luca geht in die erste Klasse. Er hat ein Freundebuch, in das sich alle seine Klassenmitglieder eintragen sollen. Dazu soll das Freundebuch herumgereicht werden, sodass jedes Klassenmitglied das Freundebuch genau ein Mal erhält und es am Ende wieder bei Luca ist. Leider gibt es in der Klasse Konflikte. Daher darf jedes Klassenmitglied das Freundebuch nur an ein Klassenmitglied weiterreichen, mit dem es keinen Konflikt hat.

Luca bezeichnet die Klassenmitglieder als $K_1, \ldots, K_n$. Luca selbst ist K_1. Nun erstellt er eine Konfliktliste L bestehend aus zweielementigen Mengen $\{K_i, K_j\}$ von Klassenmitgliedern, die einen Konflikt miteinander haben.

Die Sprache FB sei wie folgt definiert:

$$\text{FB} = \left\{ (n, L) \;\middle|\; \begin{array}{l} \text{Bei } n \text{ Klassenmitgliedern mit Konfliktliste } L \text{ gibt es eine Reihenfolge,} \\ \text{angefangen und beendet bei Klassenmitglied } K_1, \\ \text{in der jedes } K_i \text{ das Freundebuch genau ein Mal erhält und} \\ \text{eine Übergabe von } K_i \text{ an } K_j \text{ nur erlaubt ist, wenn } \{K_i, K_j\} \notin L. \end{array} \right\}$$

a) Gehört $(7, \{\{K_1, K_2\}, \{K_1, K_3\}, \{K_4, K_6\}\})$ zur Sprache FB? Begründen Sie Ihre Antwort.

Lösungsskizze: Ja, K_1 gibt das Buch an K_4, danach gibt K_4 es an K_2, K_2 an K_3. Danach kann eine beliebige Permutation von K_5, K_6, K_7 gewählt werden, weil niemand mehr Konflikte miteinander hat. Am Ende übergibt man es zurück an K_1.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Zeigen Sie, dass die Sprache FB $\mathcal{NP}$-vollständig ist.
Hinweis: Das Hamiltonische Kreis-Problem HC ist $\mathcal{NP}$-vollständig.

Lösungsskizze:
In $\mathcal{NP}$: Rate nichtdeterministisch eine Reihenfolge $(r_2, \ldots, r_n)$ der Klassenmitglieder ohne K_1, und prüfe, ob $\{K_1, r_2\} \notin L, \{K_1, r_n\} \notin L$ und ob für alle $2 \leq i \leq n-1$ gilt: $\{r_i, r_{i+1}\} \notin L$. Dies ist in $\mathcal{O}(n \cdot |L|)$ möglich.
Reduktion: Wir benutzen HC als Reduktionspartner.
Transformation: Sei $G = (V, E)$ der Graph als Eingabe für HC. Wir betrachten zu G den Komplementgraphen $\overline{G} = (V, \overline{E})$, wobei $\overline{E} = \big\{\{u, v\} | u, v \in V, u \neq v, \{u, v\} \notin E\big\}$ ist. Dann sind ist $n := |V|$ die Anzahl der Klassenmitglieder und $\overline{E} = L$ die Konfliktliste für FB.
Korrektheit:
$G \in HC \Rightarrow (n, L) \in$ FB: Wenn G einen Hamiltonischen Kreis besitzt, dann werden die benutzten Kanten nicht im Komplementgraphen $\overline{G}$ auftauchen. Damit werden die benutzten Kanten nicht in der Konfliktliste L auftauchen und das Freundebuch kann in der Reihenfolge des Kreises weitergegeben werden. Da ein Hamiltonische Kreis alle Knoten besucht, gibt es insbesondere einen Kreis, der bei K_1 beginnt und endet.
$G \in HC \Leftarrow (n, L) \in$ FB: Es gibt eine konflikfreie Übergabe, wobei jedes Klassenmitglied das Freundebuch ein Mal erhält: Also gibt es einen Kreis, der jeden Knoten genau ein Mal besucht ohne die Benutzung der Kanten aus $\overline{G}$. Da die Übergangen keine Konflikte haben, gibt es einen Hamiltonischen Kreis in G.
Laufzeit: $|V|$ bestimmen und das Komplement (ohne Eigenkanten) von E bilden ist in $O(|V| + |V|^2)$ möglich. Die Laufzeit ist also polynomiell in der Eingabegröße des Graphen G.

⇑⇑⇑ _____ / 9 Punkt(e) ⇑⇑⇑

Ein VertexCover in einem ungerichteten Graphen $G = (V, E)$ ist eine Teilmenge C von Knoten, sodass jede Kante einen Endpunkt in der Menge C besitzt. Das Ziel ist, die Anzahl der Knoten in C zu minimieren.

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit $n = |V|$ Knoten und $m = |E|$ Kanten, wobei $V = \{1, \ldots, n\}$. Beachten Sie, dass ungerichtete Graphen keine Eigenkanten haben.

Betrachten Sie nun den folgenden Algorithmus.

```
guter-greedy {
  C_g = ∅;
  Solange E ≠ ∅ {
    Wähle eine beliebige Kante {u, v} ∈ E;
    Füge Knoten u und v zu C_g hinzu;
    Entferne alle zu u und v inzidenten Kanten aus E;
  };
  Ausgabe: C_g;
};
```

a) Zeigen Sie, dass die Ausgabe C_g von Algorithmus `guter-greedy` immer höchstens 2-mal größer als eine optimale Lösung ist.

Lösungsskizze: Für jede Kante $\{u, v\} \in E$ muss mindestens einer der beiden Knoten in jedem VC vorhanden sein, insbesondere auch im optimalen VC der Größe Opt. Da der Algorithmus `guter-greedy` immer beide Knoten der Kante aufnimmt, ist: $|C_g|/2 \geq Opt$.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Bob hat einen Verbesserungsvorschlag für den Algorithmus `guter-greedy`: Statt beide Knoten u, v einer Kante zu C_g hinzuzufügen, kann man auch nur einen der beiden Knoten hinzufügen. Er schlägt vor immer den Knoten mit dem kleineren Index auszuwählen. Daraus ergibt sich der folgende Algorithmus:

```
schlechter-greedy {
  C_s = ∅;
  Solange E ≠ ∅ {
    Wähle eine beliebige Kante {u, v} ∈ E;
    Finde den Knoten mit dem kleineren Index w = min(u, v);
    Füge Knoten w zu C_s hinzu;
    Entferne alle zu w inzidenten Kanten aus E;
  };
  Ausgabe: C_s;
};
```

Zeigen Sie, dass der Algorithmus `schlechter-greedy` eine Ausgabe C_s produzieren kann, die $(n-1)$-mal größer ist als die optimale Lösung. Der Approximationsfaktor $n-1$ soll für eine beliebige Anzahl an Knoten $n \geq 3$ gelten.

Lösungsskizze: Betrachte den Sterngraph mit einem Zentralknoten n und $n-1$ Sateliten $1, \ldots, n-1$. Beachte, dass die Benennung der Knoten so ist, dass der Zentralknoten den höchsten Index n hat. Die optimale Lösung wählt nur den Zentralknoten und hat somit Größe 1. Der Algorithmus wählt alle $n-1$ Sateliten, weil diese immer einen kleineren Index haben als der Zentralknoten. Damit ist die von `schlechter-greedy` produzierte Lösung $(n-1)$-mal schlechter als die optimale Lösung.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

c) Zeigen Sie, dass für Ausgabe C_s von `schlechter-greedy` aus Aufgabenteil b) gilt: $|C_s| \leq n-1$. Folgern Sie, dass C_s garantiert höchstens $(n-1)$-mal größer ist als die optimale Lösung. Sie dürfen annehmen, dass $E \neq \emptyset$.

Lösungsskizze: Der Graph hat n Knoten. Falls C_s aus weniger als $n-1$ Knoten besteht, ist die Aussage trivial. Da es keine Eigenkanten in einem ungerichteten Graphen gibt, wird nach der Aufnahme von beliebigen $n-1$ Knoten in C_s jede Kante aus E abgedeckt sein, weil alle Nachbarn des einzigen nicht-ausgewählten Knotens u mit $\{u\} = V \setminus C_s$ gewählt sind. Damit sind auch alle zu u inzidenten Kanten abgedeckt. Daher wird `schlechter-greedy` spätestens wenn $|C_s| = n-1$ terminieren. Da $E \neq \emptyset$, ist die optimale Größe $Opt \geq 1$. Dadurch folgt die obere Schranke an den Approximationsfaktor.

⇑⇑⇑ _____ / 5 Punkt(e) ⇑⇑⇑

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Hinweis auf Seite 16 platziert wurde.

1016202100003