Algorithmen und Datenstrukturen 1

SoSe 2023 · Prof. Dr. Dell
24. Juli 2023
Klausurnummer: **0000**

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nachname (Druckschrift)**:** _____

Vorname (Druckschrift)**:** _____

Matrikelnummer: _____

Studiengang: _____

Bitte Hinweise beachten:

- Schreiben Sie Ihren Namen nur auf dieses Titelblatt.
- Zusatzpapier darf nicht mit abgegeben werden.
- Sie dürfen Stifte in den Farben blau und schwarz verwenden.
- Zugelassenes Hilfsmittel: 1 Blatt DIN A4 mit handschriftlichen Notizen (beidseitig).
- Nicht zugelassene Hilfsmittel (z.B. Handys, Smartwatches, andere Geräte) stellen eine Täuschung dar und führen zum Nichtbestehen der Klausur. Schalten Sie daher alle elektronischen Geräte aus und verstauen Sie diese in Ihrer Tasche.
- 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 gilt mit 50% der Höchstpunktzahl als bestanden.
- Die Klausur dauert 180 Minuten.

Please note:

- *Write your name on this cover sheet only.*
- *Additional paper must not be submitted.*
- *You may use pens in the colors blue and black.*
- *Approved aid: 1 sheet DIN A4 with handwritten notes (on both sides).*
- *Non-approved aids (e.g. mobile phones, smartwatches, other devices) constitute deception and lead to failing the exam. Turn off your electronic devices before the exam and store them in your bag.*
- *If two or more solutions are given for a problem, the problem counts as unsolved. So always choose **one** solution.*
- *Justifications are only necessary if the problem wording requires it.*
- *The exam is passed with 50% of the maximum number of points.*
- *The exam lasts 180 minutes.*

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Diese Seite ist für den internen Gebrauch.
Bitte leer lassen.

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Diese Seite ist für den internen Gebrauch.
Bitte leer lassen.

Aufgabe	1	2	3	4	5	6
Erreichbar	20	8	12	20	20	20
Erreicht						

Summe	Note
/100	

Aufgabe 1: Landau-Notation und Laufzeit — $5 + 2 + 2 + 2 + 9 = 20$ Punkte

a) Welche der folgenden Aussagen sind korrekt? Es könnten zwischen 0 und 8 Antworten korrekt sein. Kreuzen Sie genau die korrekten Antworten an:
Which of the following statements are correct? There could be between 0 and 8 correct answers. Mark exactly the correct answers:

☐ $2^z = O(z^2)$ ☑ $m^2 = \Omega(m^2)$ ☑ $\log z = O(z^2)$ ☑ $M = o(M^2)$

☑ $2^m = \omega(m)$ ☐ $\log y \in o(\log y)$ ☐ $2^m = \Theta(m)$ ☐ $\log n = \omega(n^2)$

⇑⇑⇑ _____ / 5 Punkte ⇑⇑⇑

b) Sei $f(n) = 100 \cdot (n + \sqrt{n}) \cdot (\log n + n^2)$. Welches asymptotische Wachstum ist für diese Funktion richtig? Kreuzen Sie genau eine Antwort an:
Let $f(n) = 100 \cdot (n + \sqrt{n}) \cdot (\log n + n^2)$. Which asymptotic growth is correct for this function? Mark exactly one answer:

☐ $\Theta(\log n)$ ☐ $\Theta(\log n + n^2)$ ☑ $\Theta(n^3)$ ☐ $\Theta(n \log n)$ ☐ $\Theta(\sqrt{n})$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

c) Geben Sie eine Funktion $f(n)$ an, sodass $f(n) = \omega(n^2)$ und $f(n) = o(n^3)$ gilt:
Enter a function $f(n)$ such that $f(n) = \omega(n^2)$ and $f(n) = o(n^3)$:

$f(n) =$ $n^{2.5}$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

d) Geben Sie eine Funktion $g(k)$ an, sodass $g(k) = \omega(\log k)$ und $g(k) = o(k)$ gilt:
Enter a function $g(k)$ such that $g(k) = \omega(\log k)$ and $g(k) = o(k)$:

$g(k) =$ $\sqrt{k}$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

e) Geben Sie für die folgenden Algorithmen jeweils die Laufzeit abhängig von n in Θ-Notation an. Sie können hierbei annehmen, dass die artihmetischen Operationen (einschließlich Wurzel ziehen, Potenzieren und Logarithmus bilden) in konstanter Zeit berechnet werden.
For the following algorithms, provide the running time in terms of n in Θ-notation. You may assume that arithmetic operations (including square root, exponentiation, and logarithm) are computed in constant time.

```
for i = 1, 2, ..., n do
    if i == ⌊log n⌋ then
        print "*"
```

```
s = √n
while s > 2 do
    s = s/2
```

```
s = 1
while s ≤ (log n)^5 do
    s = 4s
```

$\Theta($ n $)$ $\Theta($ $\log n$ $)$ $\Theta($ $\log \log n$ $)$

⇑⇑⇑ _____ / 9 Punkte ⇑⇑⇑

a) Betrachten Sie die folgende Funktion FOO.
Consider the following function FOO.

```
function FOO(n)
    if n ≤ 0 then
        print "*"
    else
        FOO(n/3)
        print "*"
        FOO(n/3)
        print "*"
        FOO(n/3)
        FOO(n/3)
```

Zum Beispiel gibt FOO(0) genau einen Stern (*) aus.
For example, FOO(0) prints exactly one star (*).

Geben Sie eine Rekursionsgleichung für die genaue Anzahl $A(n)$ von Sternen an, die ein Aufruf von FOO(n) insgesamt ausgibt. Der Basisfall ist $A(0) = 1$.
Enter a recurrence equation for the exact number $A(n)$ of stars that a call to FOO(n) prints in total. The base case is $A(0) = 1$.

Für alle $n \geq 1$ gilt:
For all $n \geq 1$, we have: $A(n) =$ $4 \cdot A(n/3) + 2$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

b) Geben Sie für folgende Rekursionsgleichungen eine geschlossene Form an.
For the following recurrence equations, give a closed form.

Sie können bei B und C davon ausgehen, dass $n = 9^k$ für eine natürliche Zahl $k > 0$ gilt, und bei D, dass $n = 9k$ für eine natürliche Zahl $k > 0$ gilt. Geben Sie das Ergebnis in Θ-Notation abhängig von n an.
In B and C, you can assume $n = 9^k$ for a natural number $k > 0$, and in D, you can assume $n = 9k$ for a natural number $k > 0$. Give the result in Θ-notation depending on n.

- $B(n) = B(\frac{n}{9}) + n^9, \quad B(1) = 1.$

  $B(n) = \Theta($ n^9 $)$

- $C(n) = 81 \cdot C(\frac{n}{9}) + n^2, \quad C(1) = 1.$

  $C(n) = \Theta($ $n^2 \log n$ $)$

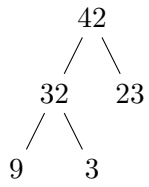
- $D(n) = D(n-9) + n^2, \quad D(0) = 1.$

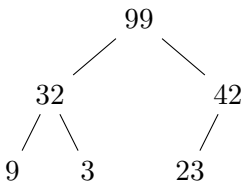
  $D(n) = \Theta($ n^3 $)$

⇑⇑⇑ _____ / 6 Punkte ⇑⇑⇑

Aufgabe 3: Anwendungsübungen $1+1+1+1+1+1+1+1+1+1+1+1=12$ Punkte

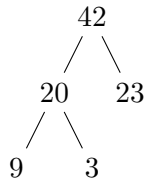
a) Links abgebildet ist ein **Max-Heap**. Wir rufen INSERT(99) auf. Zeichnen Sie rechts den Max-Heap, der dadurch entsteht.
On the left is a Max-Heap. We call INSERT(99). Draw the resulting Max-Heap on the right.

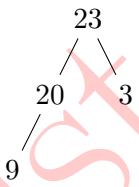


Musterlösung:


⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

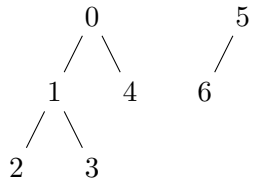
b) Links abgebildet ist ein **Max-Heap**. Wir rufen EXTRACTMAX auf. Zeichnen Sie rechts den Max-Heap, der dadurch entsteht.
On the left is a Max-Heap. We call EXTRACTMAX on this Max-Heap. Draw the resulting Max-Heap on the right.



Musterlösung:


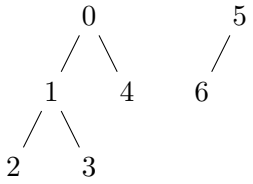
⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

c) Links abgebildet ist ein Zustand der **Quick-Union** Datenstruktur. Wir nehmen an, dass die Operation UNION(i, j) immer den durch FIND(i) spezifizierten Knoten als Kind des von FIND(j) spezifierten Knoten anhängt. Wir rufen UNION$(3, 6)$ auf. Zeichnen Sie rechts den Zustand, der dadurch entsteht.
*On the left is a state of the **Quick-Union** data structure. We assume that the operation UNION(i, j) always appends the node specified by FIND(i) as a child of the node specified by FIND(j). We call UNION$(3, 6)$. Draw the resulting state on the right.*



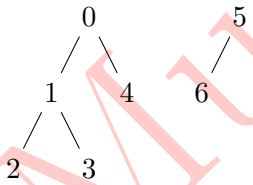
⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

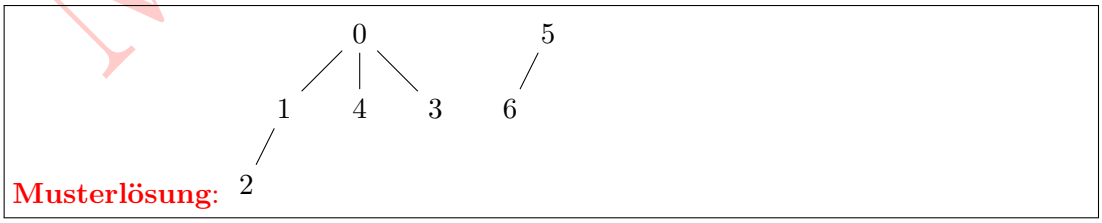
d) Links abgebildet ist ein Zustand der **Weighted Quick-Union** Datenstruktur. Wir rufen UNION(3, 6) auf. Zeichnen Sie rechts den Zustand, der dadurch entsteht.
*On the left is a state of the **Weighted Quick-Union** data structure. We call UNION(3, 6). Draw the resulting state on the right.*



⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

e) Links abgebildet ist ein Zustand der **Weighted Quick-Union** Datenstruktur mit Pfadverkürzung. Wir rufen FIND(3) auf. Zeichnen Sie rechts den Zustand, der dadurch entsteht.
*On the left is a state of the **Weighted Quick-Union** data structure with path compression. We call FIND(3). Draw the resulting state on the right.*





⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

f) Wir betrachten jetzt die Datenstruktur **Hashing mit linearem Sondieren**. Als Hashfunktion benutzen wir die Funktion $h\colon \mathbb{N} \to \{0, 1, \ldots, 10\}$ mit $h(x) = (2x) \bmod 11$.
*We now consider the data structure **Hashing with linear probing**. As a hash function we use the function $h\colon \mathbb{N} \to \{0, 1, \ldots, 10\}$ with $h(x) = (2x) \bmod 11$.*

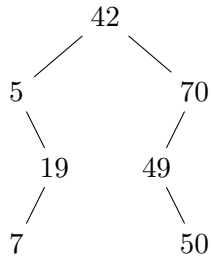
Die Hash-Tabelle hat Platz für elf Einträge und ist derzeit wie folgt gefüllt:
The hash table has room for eleven entries and is currently filled as follows:

0	1	2	3	4	5	6	7	8	9	10
				2			20	26	4	

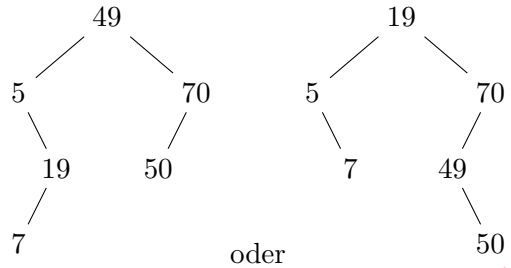
Fügen Sie die Zahlen 4 und 2 in dieser Reihenfolge in die Hashtabelle ein.
Add the numbers 4 and 2 in this order to the hash table.

⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

g) Links abgebildet ist ein (nicht notwendigerweise balancierter) **binärer Suchbaum**. Wir rufen DELETE(42) auf. Zeichnen Sie rechts den Zustand, der dadurch entsteht.
*On the left is a (not necessarily balanced) **binary search tree**. We call DELETE(42). Draw the resulting state on the right.*

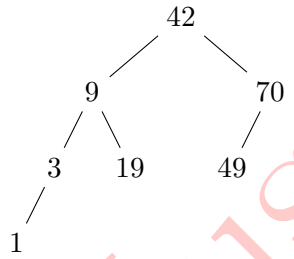


Musterlösung:

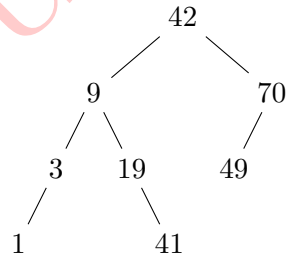


⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

h) Links abgebildet ist ein (nicht notwendigerweise balancierter) **binärer Suchbaum**. Wir rufen INSERT(41) auf. Zeichnen Sie rechts den Zustand, der dadurch entsteht.
*On the left is a (not necessarily balanced) **binary search tree**. We call INSERT(41). Draw the resulting state on the right.*

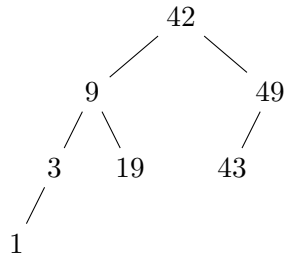


Musterlösung:

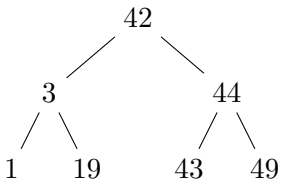


⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

i) Links abgebildet ist ein **AVL-Baum**. Wir rufen DELETE(9) und INSERT(44) in dieser Reihenfolge auf. Zeichnen rechts den AVL-Baum, der dadurch entsteht.
*On the left is an **AVL tree**. We call DELETE(9) and INSERT(44) in this order. Draw the resulting AVL-tree on the right.*

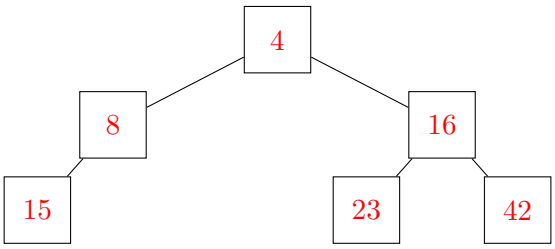


Musterlösung:



⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

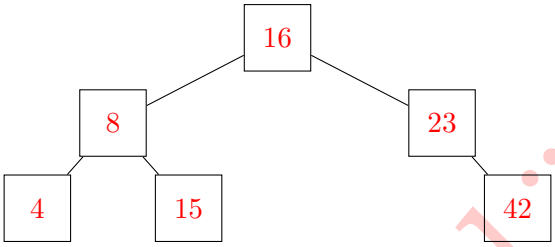
j) Hier ist die Struktur eines Baumes, an dessen Knoten Zahlen gespeichert sind:
Here is the structure of a tree whose nodes store numbers:



Die Präorder-Traversierung (*preorder traversal*) dieses Baums gibt die Zahlen 4, 8, 15, 16, 23, 42 in dieser Reihenfolge aus. Füllen Sie die Zahlen richtig in die Knoten ein.
The preorder traversal of this tree outputs the numbers 4, 8, 15, 16, 23, 42 in this order. Correctly fill in the numbers in the nodes.

⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

k) Hier ist die Struktur eines Baumes, an dessen Knoten Zahlen gespeichert sind:
Here is the structure of a tree whose nodes store numbers:



Die Inorder-Traversierung (*inorder traversal*) dieses Baums gibt die Zahlen 4, 8, 15, 16, 23, 42 in dieser Reihenfolge aus. Füllen Sie die Zahlen richtig in die Knoten ein.
The inorder traversal of this tree outputs the numbers 4, 8, 15, 16, 23, 42 in this order. Correctly fill in the numbers in the nodes.

⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

l) Gegeben ist ein Text, der aus den Buchstaben `a`, `b`, `c`, `d`, `e`, `f`, `g` besteht. Die Häufigkeit der Buchstaben ist wie folgt:
Given is a text consisting of the letters `a`, `b`, `c`, `d`, `e`, `f`, `g`. The frequency of the letters is as follows:

a	b	c	d	e	f	g
22	2	1	4	100	8	17

Geben Sie einen optimalen präfixfreien Code an, der diese Buchstaben in Bitfolgen übersetzt:
Give an optimal prefix-free code that translates these letters into bit strings:

a	b	c	d	e	f	g
10	110000	110001	11001	0	1101	111

⇑⇑⇑ _____ / 1 Punkte ⇑⇑⇑

Aufgabe 4: Platzsparende Dynamische Programmierung — 20 Punkte

Die *Tribonacci-Folge* ist die Folge $0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, \ldots$, wobei das n-te Glied t_n der Folge definiert ist als $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ für $n \geq 4$ und $t_1 = 0$ sowie $t_2 = t_3 = 1$. Geben Sie einen Algorithmus **in Code oder Pseudocode** an, der mit platzsparender dynamischer Programmierung die n-te Glied der Folge berechnet. Der Algorithmus darf nicht rekursiv sein, muss eine Laufzeit von $O(n)$ haben und darf nur konstant viel Speicherplatz verwenden.

The Tribonacci sequence *is the sequence* $0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, \ldots$, *where the* n*-th element* t_n *of the sequence is defined as* $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ *for* $n \geq 4$ *and* $t_1 = 0$ *and* $t_2 = t_3 = 1$. *Give an algorithm* ***in code or pseudocode*** *that uses space-efficient dynamic programming to compute the* n*-th element of the Tribonacci sequence. The algorithm must not be recursive, must have a running time of* $O(n)$*, and may only use constant space.*

```
function Trib(n)
    if n == 1: return 0
    if n == 2: return 1
    a, b, c := 0, 1, 1
    repeat n − 3 times:
        a, b, c := b, c, a + b + c
    return c
```

Aufgabe 5: Beweis: Aktualisierung eines Minimalen Spannbaums — $5 + 15 = 20$ Punkte

Wir betrachten einen Algorithmus UPDATEMST, der einen minimalen Spannbaum aktualisiert, wenn das Gewicht einer einzelnen Kante im Graphen verkleinert wird. Dieser Algorithm erhält als Eingabe also einen ungerichteten Graph G mit Kantengewichten $w\colon E \to \mathbb{Z}$, einen minimalen Spannbaum T von (G, w), sowie eine Kante $e_0 \in E(G)$ und $a \in \mathbb{Z}$ mit $a > 0$. Seien w' neue Kantengewichte für G, wobei Gewichte $w'(e) = w(e)$ für Kanten mit $e \neq e_0$ aus w übernommen werden, aber $w'(e_0) = w(e_0) - a$ gesetzt wird. Der Algorithmus berechnet nun effizient einen minimalen Spannbaum von (G, w').

```
1: function UPDATEMST(G, w, T, e_0, a)
2:     if e_0 ∈ T then
3:         return T
4:     else
5:         Sei w' definiert wie oben
6:         Berechne den eindeutigen einfachen Kreis C in T ∪ {e_0}
7:         Sei e_max die schwerste Kante in C bezüglich w'
8:         T' ← (T ∪ {e_0}) \ {e_max})
9:         return T'
```

We consider an algorithm UPDATEMST that updates a minimal spanning tree when the weight of a single edge in the graph is decreased. This algorithm is given an undirected graph G with edge weights $w\colon E \to \mathbb{Z}$, a minimal spanning tree T of (G, w), as well as an edge $e_0 \in E(G)$ and $a \in \mathbb{Z}$ with $a > 0$. Let w' be new edge weights for G, where the weight $w'(e) = w(e)$ of edges with $e \neq e_0$ is taken from w, but we set $w'(e_0) = w(e_0) - a$. The algorithm now efficiently computes a minimal spanning tree of (G, w').

```
1: function UPDATEMST(G, w, T, (e_0, a))
2:     if e_0 ∈ T then
3:         return T
4:     else
5:         Let w' be defined as above
6:         Compute the unique simple cycle C in T ∪ {e_0}
7:         Let e_max be the heaviest edge in C with respect to w'
8:         T' ← (T ∪ {e_0}) \ {e_max}
9:         return T'
```

a) Beweisen Sie, dass die Laufzeit von UPDATEMST höchstens $O(n)$ ist. Erklären Sie insbesondere, wie der Kreis C in Zeile 6 in dieser Zeit berechnet wird.

Prove that the running time of UPDATEMST is at most $O(n)$. Explain in particular how to compute the cycle C in line 6 in this time.

Musterlösung: T ist zunächst als unsortierte Liste von Kanten gespeichert und wir brauchen $O(n)$ Zeit, um Zeile 2 auszuführen. Die Laufzeit des else-Blocks ist dominiert von Zeile 6. Um diese in Zeit $O(n)$ auszuführen, berechnen wir zunächst die Adjazenzliste des von T induzierten Teilgraphen. Sei $e_0 = \{v, w\}$. Zeile 6 konkretisieren wir, indem wir in dem Graphen T eine Breitensuche von v aus starten. Wenn wir bei w ankommen, führen wir back-tracking durch, um den eindeutigen Pfad von v nach w zu erhalten. Breitensuche in einem Graphen mit n Knoten und $n - 1$ Kanten benötigt $O(n)$ Zeit.

⇑⇑⇑ _____ / 5 Punkte ⇑⇑⇑

b) Beweisen Sie die Korrektheit von UPDATEMST. Nehmen Sie dafür an, dass in w und w' alle Kantengewichte paarweise verschieden sind.
Proof the correctness of UPDATEMST. Assume here that all edge weights in w and w' are pairwise different.

Musterlösung:
Fall 1: $e_0 \in T$. Angenommen es gäbe einen Spannbaum T'' von G mit

$$w'(T'') < w'(T).$$

Falls T'' die Kante e_0 nicht enthält, so gilt somit

$$w(T) \geq w'(T) > w'(T'') = w(T''). \quad \text{(Widerspruch)}$$

Falls T'' die Kante e_0 enthält, so gilt andererseits

$$w(T) = w'(T) + a > w'(T'') + a = w(T'') - a + a = w(T''). \quad \text{(Widerspruch)}$$

Fall 2: $e_0 \notin T$. Nach der Kreiseigenschaft kann $e_{\max}$ in keinem minimalen Spannbaum von (G, w') enthalten sein. Wir zeigen, dass alle Kanten in $E(G) \setminus (T \cup \{e_0\})$ in keinem minimalen Spannbaum von (G, w') enthalten sind. Damit ist dann $(T \cup \{e_0\}) \setminus \{e_{\max}\}$ der einzige noch mögliche minimale Spannbaum von (G, w').
Sei $e \in E(G) \setminus (T \cup \{e_0\})$ und $e = \{v, w\}$. Dann ist der eindeutige v-w-Pfad in T zusammen mit der Kante e ein Kreis in G, wobei darauf alle Kanten außer e aus T stammen. Nach der Kreiseigenschaft kann die schwerste Kante jedes Kreises in keinem minimalen Spannbaum enthalten sein. Wegen der Minimalität von T muss also e die schwerste Kante auf diesem Kreis sein. Da $e_0 \neq e$ und $e_0 \notin T$, ändert sich keins der Gewichte auf diesem Kreis von w zu w'. Somit kann e nach der Kreiseigenschaft auch in keinem minimalen Spannbaum von (G, w') enthalten sein.

⇑⇑⇑ _____ / 15 Punkte ⇑⇑⇑

Aufgabe 6: Median-Datenstruktur — 20 Punkte

Entwerfen Sie eine Datenstruktur, die eine endliche Menge $S \subseteq \mathbb{Z}$ der Größe n speichert und dabei die folgenden Operationen unterstützt:

- `insert`(x): Fügt die Zahl x in die Menge S ein; falls $x \in S$ gilt, so passiert nichts. Diese Operation darf nur Zeit $O(\log n)$ brauchen.
- `median`(): Falls n ungerade ist, liefert diese Operation den aktuellen Median der Menge zurück (also die Zahl $m \in S$, für die gilt: $|\{x \in S \mid x < m\}| = |\{x \in S \mid x > m\}|$). Falls n gerade ist, wird `None` zurückgeliefert. Diese Operation darf nur **konstante Laufzeit** haben.

Beschreiben Sie die Datenstruktur und die Operationen in Worten oder Pseudocode und begründen Sie, warum Ihre Operationen die Anforderungen an die Laufzeiten einhalten.

Design and describe a data structure that stores a set $S \subseteq \mathbb{Z}$ of size n and supports the following operations:

- *`insert`(x): Inserts the number x into the set S; if $x \in S$, nothing happens. This operation must take time $O(\log n)$.*
- *`median`(): If n is odd, this operation returns the current median of the set (i.e., the number $m \in S$ with $|\{x \in S \mid x < m\}| = |\{x \in S \mid x > m\}|$). If n is even, `None` is returned. This operation must have **constant running time**.*

Describe the data structure and the operations in words or pseudocode and justify why your operations meet the requirements on the running time.

Musterlösung: Die Datenstruktur ist ein balancierter Suchbaum (z.B. ein AVL-Baum), der an jedem Knoten x zusätzlich die Anzahl der Knoten im Teilbaum unter x speichert. Die `insert`(x)-Operation wird wie üblich für balancierte Suchbäume implementiert, wobei ein bereits existierendes Element nicht doppelt eingefügt wird, und die zusätzliche Information über die Anzahl der Knoten im Teilbaum wird bei allen Operationen (Rotationen, etc.) entsprechend aktualisiert.
Nach jedem `insert`(x)-Aufruf wird der neue Median in Zeit $O(\log n)$ berechnet und in einer zusätzlichen globalen Variablen gespeichert. Die `median`()-Operation gibt dann einfach den Wert dieser Variablen zurück.
Der Median wird in $O(\log n)$-Zeit wie folgt berechnet: Falls n gerade ist, so setzen wir die globale Variable auf `None` und beenden die Operation. Falls n ungerade ist, so suchen wir den Eintrag $m \in S$, sodass es genau $R = \frac{n}{2} - 1$ Elemente kleiner als m in S gibt. Wir tun das mit einer rekursiven Funktion `FindRank`(x, R), wobei x anfangs die Wurzel ist:

- Falls der linke Unterbaum von x genau R Elemente enthält, so ist x der gesuchte Eintrag und wir beenden die Suche.
- Falls der linke Unterbaum von x mehr als R Elemente enthält, so gehe in den linken Unterbaum und suche dort rekursiv nach dem Eintrag mit dem Rang R: `FindRank`$(x$`.left`$, R)$
- Ansonsten enthält der linke Unterbaum z Elemente mit $z < R$, also muss der gesuchte Eintrag im rechten Unterbaum von x sein und dort Rang $R - z - 1$ haben. Wir machen also den rekursiven Aufruf `FindRank`$(x$`.right`$, R - z - 1)$.

Die Funktion `FindRank` braucht Zeit $O(\log n)$, da der Baum balanciert ist und also nur Höhe $O(\log n)$ hat.
Alternative Lösung: Zwei Heaps. Es ist auch möglich, einen Max-Heap und einen Min-Heap zu benutzen, sodass der Median immer das Maximum des Max-Heaps ist (siehe `https://stackoverflow.com/a/11385422`). Hierbei muss man jedoch zusätzlich beachten, dass `insert`(x) den Eintrag x nicht doppelt einfügen darf, falls x bereits in der Menge enthalten ist. Daher müssen wir uns zusätzlich mit Hilfe einer dynamischen Menge merken, welche Elemente bereits in den Heaps enthalten sind. Für die dynamische Menge können wir keine Hash-Tabelle nutzen, da diese nur erwartete Konstante Laufzeit per Operation hat; stattdessen würde man die dynamische Menge mit einem AVL-Baum speichern.