Algorithmen und Datenstrukturen 1

SoSe 2021 · Prof. Dr. H. Dell
6. Oktober 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**.
- Merken oder notieren Sie sich Ihre Klausurnummer **0000**.
- Zusatzpapier abgeben? Vermerken Sie darauf unbedingt die Klausurnummer **0000**. Außerdem: Verweisen Sie unbedingt von der Aufgabe auf das Zusatzblatt!
- Nur **dokumentenechte Stifte** in den Farben blau und schwarz! Füller, Tipp-Ex, Tintenlöscher sind untersagt.
- Zugelassenes Hilfsmittel: 1 DIN A4 Blatt mit handschriftlichen Notizen (beidseitig).
- Alle elektronischen Geräte sind untersagt! Insbesondere also Handys, Smartwatches, Taschenrechner. **Ausschalten und wegpacken!** Nichtbeachtung stellt einen Täuschungsversuch dar und führt zum Nichtbestehen der Klausur.
- Zwei oder mehr Lösungen angegeben? Die Aufgabe gilt dann 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 mindestens **50%** der Höchstpunktzahl erreicht wird. Die Klausur dauert 180 Minuten.

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	7
Erreichbar	16	10	12	10	12	9	11
Erreicht							

Klausur	Bonifikation	Summe	Note

a) Sei n eine natürliche Zahl. Wie viele Sterne (*) gibt der folgende Code aus?

```
for i = 1, 2, ..., 2n do
    if i ist gerade then
        print "**"
    else
        print "***"
```

Es werden genau $5n$ Sterne ausgegeben. (Gesucht ist die genaue Anzahl in Abhängigkeit von n, nicht die asymptotische Anzahl in O- oder Θ-Notation.)

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

b) Geben Sie für die folgenden Algorithmen jeweils die Laufzeit in Θ-Notation abhängig von n an.

```
for i = 1, 2, ..., n do
    if i == n then
        for j = 1, 2, ..., i
        do
            print "*"
```

$\Theta($ n $)$

```
s = 1
while s ≤ n do
    s = 5s
    for i = 1, 2, ..., s do
        print "*"
```

$\Theta($ n $)$

```
s = 1
while s ≤ 5^√n do
    s = 3s
```

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

⇑⇑⇑ _____ / 6 Punkte ⇑⇑⇑

c) Sei $f(n) = 100n\log n + n^2\log n + \frac{1}{2}n + n^3/\log\log n$. Welches asymptotische Wachstum ist für diese Funktion richtig? Kreuzen Sie genau eine Box an.

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

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

d) Sei $f(n) = \sqrt{n}\log n + n^{2/3} - 99\sqrt{n} - \log\log n$. Welches asymptotische Wachstum ist für diese Funktion richtig? Kreuzen Sie genau eine Box an.

☐ $\Theta(n^2)$ ☑ $\Theta(n^{2/3} - \sqrt{n}\log^{100} n)$ ☐ $o(n^{2/3})$ ☐ $\Theta(\sqrt{n})$ ☐ $\Theta(\log\log n)$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

e) Sei $f(n) = (n^3 + n^{0.1}) \cdot (\sqrt{n} + \log n)$. Welches asymptotische Wachstum ist für diese Funktion richtig? Kreuzen Sie genau eine Box an.

☐ $O(n^{3.1})$ ☐ $\Theta(n^{3.1})$ ☑ $\omega(n^{3.1})$ ☐ $o(n^{3.1})$ ☐ $\Omega(n^9)$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

f) Sei $f(n) = \dfrac{\log^{99} n}{n^{0.1}}$. Welches asymptotische Wachstum ist für diese Funktion richtig? Kreuzen Sie genau eine Box an.

☐ $\Theta(\log^{99} n)$ ☐ $\Omega(\log^{99} n)$ ☐ $\Omega(1)$ ☑ $O(1)$ ☐ $\omega(\log^{99} n)$

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

a) Betrachten Sie die folgende Funktion FOO.

```
function FOO(n)
    if n ≤ 0 then
        print "**"
    else
        FOO(n − 1)
        print "*"
        FOO(n − 1)
        print "**"
```

Zum Beispiel gibt FOO(0) zwei Sterne (∗) aus. Wie viele Sterne gibt FOO(1) aus?

Antwort: 7.

Geben Sie außerdem eine Rekursionsgleichung für die genaue Anzahl $A(n)$ von Sternen an, die ein Aufruf von FOO(n) insgesamt ausgibt. Der Basisfall ist $A(0) = 2$.

Für alle $n \geq 1$ gilt: $A(n) =$ $2 \cdot A(n-1) + 3$

⇑⇑⇑ _____ / 4 Punkte ⇑⇑⇑

b) Geben Sie für folgende Rekursionsgleichungen eine geschlossene Form an.

Sie können bei B und C davon ausgehen, dass $n = 2^k$ für eine natürliche Zahl $k > 0$ gilt, und bei D, dass $n = 2k$ für eine natürliche Zahl $k > 0$ gilt. Geben Sie das Ergebnis in Θ-Notation abhängig von n an.

- $B(n) = B(\frac{n}{2}) + n^2, \quad B(1) = 1.$

  $B(n) = \Theta($ n^2 $)$

- $C(n) = 8 \cdot C(\frac{n}{2}) + n^2, \quad C(1) = 1.$

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

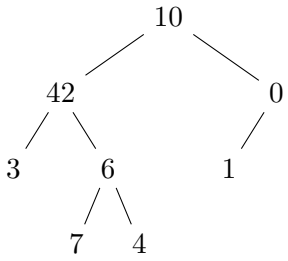
- $D(n) = D(n-2) + n^3, \quad D(0) = 1.$

  $D(n) = \Theta($ n^4 $)$

⇑⇑⇑ _____ / 6 Punkte ⇑⇑⇑

Sei T ein Binärbaum. Jeder Knoten x von T hat die Eigenschaften $x.\texttt{parent}$, $x.\texttt{left}$ und $x.\texttt{right}$, welche auf den Elternknoten sowie auf das linke und rechte Kind von x verweisen. Wenn der Knoten kein Kind oder keinen Elternknoten hat, ist der jeweilige Wert auf `null` gesetzt (zum Beispiel $x.\texttt{right} = \texttt{null}$). Des Weiteren hat jeder Knoten x eine Eigenschaft $x.\texttt{number}$, die eine einzelne Zahl speichert.

In diesem Beispielbaum B ist jeder Knoten mit $x.\texttt{number}$ markiert:



Die Wurzel von B ist der Knoten v mit $v.\texttt{number} = 10$.

a) Betrachten Sie die folgende Funktion:

```
function FuncA(x)
    if x ≠ null then
        return FuncA(x.left) + FuncA(x.right)
    else
        return 1
```

Welche Zahl liefert FuncA(v) über die **return** Anweisung als Rückgabewert, wenn v die Wurzel des Beispielbaums B ist?

Antwort: 9

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

b) Betrachten Sie die folgende Funktion:

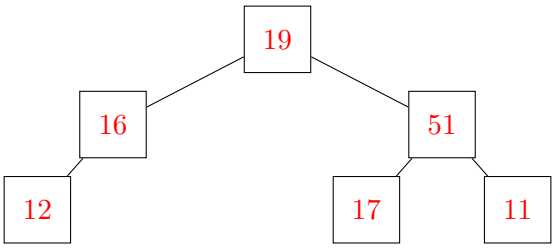
```
function FuncB(x, i)
    if x ≠ null und i == 0 then
        print x.number
        FuncB(x.left, 1)
        FuncB(x.right, 1)
    else if x ≠ null und i == 1 then
        FuncB(x.left, 0)
        print x.number
```

Welche Zahlen gibt FuncB($v, 1$) über die **print** Anweisungen aus, wenn v die Wurzel des Beispielbaums B ist? Geben Sie die Zahlen in derselben Reihenfolge an, in der sie ausgegeben werden.

Antwort: 42, 3, 7, 6, 10

⇑⇑⇑ _____ / 3 Punkte ⇑⇑⇑

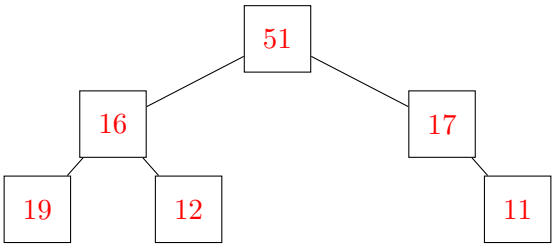
c) Hier ist die Struktur eines Baumes, die Zahlen x.`number` sind nicht dargestellt:



Die Präorder-Traversierung (*preorder traversal*) dieses Baums gibt die Zahlen 19, 16, 12, 51, 17, 11 in dieser Reihenfolge aus. Füllen Sie die Zahlen richtig in die Knoten ein.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

d) Hier ist die Struktur eines Baumes, die Zahlen x.`number` sind nicht dargestellt:



Die Inorder-Traversierung (*inorder traversal*) dieses Baums gibt die Zahlen 19, 16, 12, 51, 17, 11 in dieser Reihenfolge aus. Füllen Sie die Zahlen richtig in die Knoten ein.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

e) Wir erinnern uns an die Funktion Size aus der Vorlesung, die bei Aufruf von Size(x) die Zahl aller Knoten im Unterbaum mit Wurzel x zurückliefert.

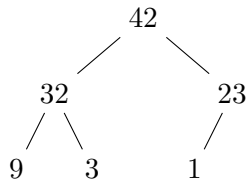
Wir wollen jetzt eine Funktion Blubb schreiben, die die Anzahl aller Knoten zurückliefert, die einen Vorfahren y mit y.`number` == 42 haben. Zum Beispiel: Wenn v die Wurzel des Beispielbaums B ist, dann soll Blubb(v) die Zahl 5 zurückliefern (der Knoten 42 zählt also mit).

Vervollständigen Sie in der folgenden Funktion die drei **return**-Anweisungen. Sie dürfen die Funktion Size benutzen, aber keine weiteren Zeilen einfügen!

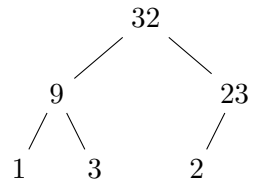
```
function Blubb(x)
    if x == null then
        return 0
    else if x.number == 42 then
        return Size(x)
    else
        return Blubb(x.left) + Blubb(x.right)
```

⇑⇑⇑ _____ / 3 Punkte ⇑⇑⇑

a) Der **Max-Heap** ist eine konkrete Datenstruktur, die eine Prioritätswarteschlange implementiert. Hier ist ein **Max-Heap** mit sechs Elementen abgebildet:



Musterlösung:



Wir rufen jetzt ExtractMax() und Insert(2) in dieser Reihenfolge auf. Zeichnen Sie im selben Stil den Max-Heap, der dadurch am Ende entsteht.

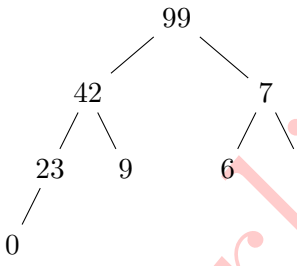
⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

b) Hier ist ein Max-Heap als Feld dargestellt:

0	1	2	3	4	5	6	7	8
–	99	42	7	23	9	6	1	0

Zeichnen Sie den Max-Heap als Baum im selben Stil wie in Aufgabenteil a).

Musterlösung:



⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

c) **Hashing mit linearem Sondieren** ist eine konkrete Datenstruktur, die eine dynamische Menge von Elementen verwaltet. Als Hashfunktion benutzen wir die Funktion $h\colon \mathbb{N} \to \{0, 1, \ldots, 12\}$ mit

$$h(x) = (2x) \bmod 13\,.$$

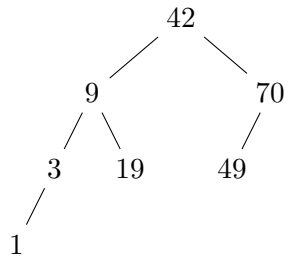
Die Hash-Tabelle hat Platz für dreizehn Einträge und ist derzeit wie folgt gefüllt:

0	1	2	3	4	5	6	7	8	9	10	11	12
19	33	7				42		30	4			6

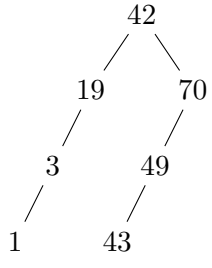
Fügen Sie nacheinander 4, 7, 6, 19 in dieser Reihenfolge in die Hashtabelle ein.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

d) Ein **binärer Suchbaum** ist eine konkrete Datenstruktur, die eine Menge von sortierten Daten verwaltet. Hier ist ein binärer Suchbaum mit sieben Elementen abgebildet:



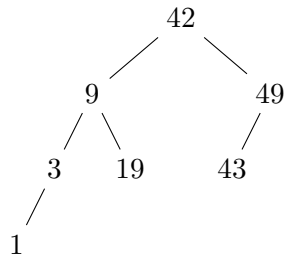
Musterlösung:



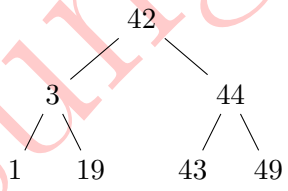
Wir rufen jetzt DELETE(9) und INSERT(43) in dieser Reihenfolge auf. Zeichnen Sie im selben Stil den binären Suchbaum, der dadurch am Ende entsteht.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

e) Ein **AVL-Baum** ist eine konkrete Datenstruktur, die eine Menge von sortierten Daten verwaltet. Hier ist ein AVL-Baum mit sieben Elementen abgebildet:



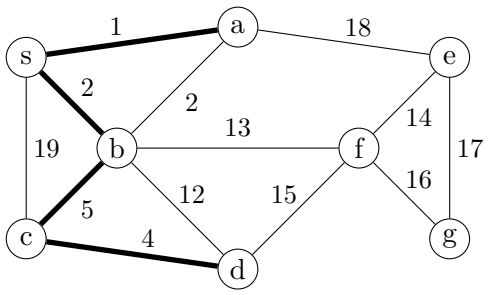
Musterlösung:



Wir rufen jetzt DELETE(9) und INSERT(44) in dieser Reihenfolge auf. Zeichnen Sie im selben Stil den AVL-Baum, der dadurch am Ende entsteht.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

Wir lassen Prims, Kruskals und Dijkstras Algorithmus auf diesem ungerichteten, gewichteten Graphen laufen:



a) Prims Algorithmus wurde mit Wurzel s gestartet. Die Abbildung zeigt den Zustand von Prims Algorithmus, nachdem vier Kanten in den Baum eingefügt wurden (fett gezeichnet). Welche Kante $\{u, v\}$ wird Prims Algorithmus als Nächstes einfügen?

Antwort: *bf* (gesucht ist die Kante, nicht ihr Gewicht)

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

b) Dieselbe Abbildung zeigt den Zustand von Kruskals Algorithmus, nachdem vier Kanten in den Wald eingefügt wurden (fett gezeichnet). Welche Kante $\{u, v\}$ wird Kruskals Algorithmus als Nächstes einfügen?

Antwort: *bf* (gesucht ist die Kante, nicht ihr Gewicht)

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

c) Dijkstras Algorithmus wurde mit Wurzel s gestartet. Dieselbe Abbildung zeigt jetzt den Zustand von Dijkstras Algorithmus, nachdem vier Kanten in den Baum eingefügt wurden (fett gezeichnet). Welche Kante $\{u, v\}$ wird Dijkstras Algorithmus als Nächstes einfügen?

Antwort: *bf* (gesucht ist die Kante, nicht ihr Gewicht)

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

d) Welches Gewicht hat der minimale Spannbaum?

Antwort: 55

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

e) Gibt es einen minimalen Spannbaum, der die Kante $\{a, e\}$ enthält? Schreiben Sie "Ja" oder "Nein" und begründen Sie Ihre Antwort.

Musterlösung: Nein: Auf dem Kreis a, e, f, b, a ist ae die eindeutige schwerste Kante. Aus der Kreiseigenschaft folgt, dass ae in keinem MST enthalten ist.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

f) Gibt es einen minimalen Spannbaum, der die Kante $\{b, c\}$ nicht enthält? Schreiben Sie "Ja" oder "Nein" und begründen Sie Ihre Antwort.

Musterlösung: Nein: Die Partition $\{c, d\}$ und $\{s, a, b, e, f, g\}$ bildet einen Schnitt, für den $\{b, c\}$ die eindeutige leichteste kreuzende Kante ist. Aufgrund der Schnitteigenschaft muss $\{b, c\}$ in jedem MST enthalten sein.

⇑⇑⇑ _____ / 2 Punkte ⇑⇑⇑

a) Sei $A[1..n]$ ein Feld der Länge $n \geq 3$, welches rationale Zahlen enthält. Und seien i, j ganze Zahlen mit $1 \leq i \leq j \leq n$. Hier ist ein rekursiver Algorithmus:

```
function RecFunc(A, i, j)
    if i == n then
        return A[j]
    else if j == i then
        return A[i]
    else
        return max{ A[i] · RecFunc(A, i + 1, j),  (A[j])^2 · RecFunc(A, i, j − 1) }
```

Benutzen Sie dynamische Programmierung, um diesen rekursiven Algorithmus iterativ zu machen. Füllen Sie hierzu die folgende Funktion aus. Sie dürfen nur die unterstrichenen Teile vervollständigen und keine neuen Zeilen hinzufügen!

```
function IterFunc(A)
    Initialisiere ein zweidimensionales Feld T[1..n][1..n].
    for i from  n  to  1  do
        for j from  i  to  n  do
            if  i == n  then
                T[i][j] = A[j]
            else if  j == i  then
                T[i][j] = A[i]
            else
                T[i][j] = max{A[i] · T[i + 1][j],  (A[j])^2 · T[i][j − 1]}
    return T[1][n]
```

⇑⇑⇑ _____ / 4 Punkte ⇑⇑⇑

b) Sei $S[1..n]$ eine Zeichenkette (*string*) der Länge n. Jedes Zeichen ist entweder `a` oder `b`. Geben Sie einen **rekursiven** Algorithmus Count an, der `True` oder `False` zurückgibt. Hierbei soll Count(i, j, k) für $1 \leq i \leq j \leq n$ genau dann `True` zurückliefern, wenn der Teilstring $S[i..j]$ genau k mal das Zeichen `a` enthält. Abgesehen von rekursiven Aufrufen muss die Laufzeit **konstant** sein. Geben Sie Ihren Algorithmus in **Pseudocode** an:

Musterlösung:

```
function Count(i, j, k)
    if i == j then
        return (S[i] == 'a' and k == 1) or ((S[i] == 'b' and k == 0)
    else
        return (S[i] == 'a' and Count(i + 1, j, k − 1)) or (S[i] == 'b' and
        Count(i + 1, j, k))
```

⇑⇑⇑ _____ / 5 Punkte ⇑⇑⇑

Aufgabe 7: Joggen 🏃‍♀️🏃 — $5 + 6 = 11$ Punkte

Um wieder in Form zu kommen, haben Sie sich entschieden, im nun anstehenden Präsenzsemester jeden Tag von Zuhause zur Uni zu joggen. Es gibt viele Hügel in der Stadt. Sie wollen eine *ausgeglichene Joggingroute* finden, die erst anstrengend und dann lockerer ist, damit Sie gegen Ende abkühlen können. Das heißt, Ihre Joggingroute darf zuerst nur bergauf führen und dann nur bergab.

Sie haben eine detaillierte Straßenkarte mit insgesamt $\boldsymbol{m}$ **Straßenabschnitten** und $\boldsymbol{n}$ **Kreuzungen**. Jeder Straßenabschnitt verbindet zwei Kreuzungen, hat eine **positive, bekannte Länge** und hat eine feste Steigung. Jede Kreuzung hat eine **bekannte Höhe**. Ihre Joggingroute startet an Kreuzung s (Zuhause) und endet an Kreuzung t (Uni), wobei $s \neq t$. Eine Joggingroute ist **ausgeglichen**, wenn es für die Sequenz der Höhen $h_1, h_2, \ldots, h_\ell$ entlang der besuchten Kreuzungen einen Index i gibt mit

$$h_1 \leq h_2 \leq \cdots \leq h_{i-1} \leq h_i \geq h_{i+1} \geq h_{i+2} \geq \cdots \geq h_\ell\,.$$

a) Beschreiben Sie **kurz, präzise und in Stichpunkten** einen Algorithmus, der die kürzeste ausgeglichene Laufstrecke berechnet. Der Algorithmus soll die Antwort in Zeit $\boldsymbol{O(m + n \log n)}$ berechnen; begründen Sie, warum Ihr Algorithmus die Zeit einhält.

Musterlösung: 1) Konstruiere einen gerichteten, gewichteten Graphen $\boldsymbol{D_{\text{hoch}}}$ wie folgt: Jede Kreuzung ist ein Knoten. Jeder Straßenabschnitt von einer Kreuzung $\boldsymbol{u}$ zu einer Kreuzung $\boldsymbol{v}$, deren Höhen $\boldsymbol{h_u}$ und $\boldsymbol{h_v}$ die Ungleichung $\boldsymbol{h_u \leq h_v}$ erfüllen, wird zu einer Kante $\boldsymbol{uv}$ in $\boldsymbol{D_{\text{hoch}}}$. Das Gewicht $\boldsymbol{w(uv)}$ ist dann definiert als die Länge des Straßenabschnitts $\boldsymbol{uv}$.
2) Lass Dijkstra auf $\boldsymbol{D_{\text{hoch}}}$ von $\boldsymbol{s}$ ausgehend laufen, und sei $\boldsymbol{d}$ die Ausgabe von Dijkstras Algorithmus, also $\boldsymbol{d[1..n]}$ ist ein Feld, wo $\boldsymbol{d[v]}$ die Länge des kürzesten Wegs von $\boldsymbol{s}$ nach $\boldsymbol{v}$ für alle $\boldsymbol{v}$ ist.
2') Lass Dijkstra auf $\boldsymbol{D_{\text{hoch}}}$ von $\boldsymbol{t}$ ausgehend laufen, und sei $\boldsymbol{d'}$ die Ausgabe von Dijkstras Algorithmus, also $\boldsymbol{d'[1..n]}$ ist ein Feld, wo $\boldsymbol{d'[v]}$ die Länge des kürzesten Wegs von $\boldsymbol{t}$ nach $\boldsymbol{v}$ für alle $\boldsymbol{v}$ ist.
3) Finde einen Knoten $\boldsymbol{v}$, der die Summe $\boldsymbol{d(v) + d'(v)}$ minimiert. Die kürzeste Laufstrecke entspricht dem kürzesten Weg von $\boldsymbol{s}$ nach $\boldsymbol{v}$ in $\boldsymbol{D_{\text{hoch}}}$ konkateniert mit dem umgedrehten kürzesten Weg von $\boldsymbol{t}$ nach $\boldsymbol{v}$ in $\boldsymbol{D_{\text{hoch}}}$. Die Laufstrecke selbst erhalten wir aus den Feldern $\boldsymbol{d}$ und $\boldsymbol{d'}$ mithilfe von Backtracking.
Laufzeit: 1) und 3) laufen in Linearzeit $\boldsymbol{O(n + m)}$. Schritt 2) und 2') laufen in Zeit $\boldsymbol{O(m + n \log n)}$, da dies die Laufzeit von Dijkstras Algorithmus ist.

⇑⇑⇑ _____ / 5 Punkte ⇑⇑⇑

b) Wir nehmen an, dass alle Kreuzungen unterschiedliche Höhen haben. Beschreiben Sie **kurz, präzise und in Stichpunkten** einen Algorithmus, der die kürzeste ausgeglichene Laufstrecke berechnet. Der Algorithmus soll die Antwort in Zeit $\boldsymbol{O(m + n)}$ berechnen; begründen Sie, warum Ihr Algorithmus die Zeit einhält.

Musterlösung: Wie a), nur sind jetzt alle Höhen unterschiedlich. Aus diesem Grund ist $\boldsymbol{D_{\text{hoch}}}$ ein azyklischer gerichteter Graph. In der Vorlesung haben wir gesehen, dass man kürzeste Wege in azyklischen gerichteten Graphen in Zeit $\boldsymbol{O(n + m)}$ berechnen kann, indem man den Graphen zunächst topologisch sortiert und dann alle Kanten von links nach rechts relaxiert. Wir ersetzen also 2) und 2') durch diesen Algorithmus.

⇑⇑⇑ _____ / 6 Punkte ⇑⇑⇑

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Verweis zu Seite 13 platziert wurde.

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Verweis zu Seite 14 platziert wurde.