Algorithmen und Datenstrukturen 1

SoSe 2022 · Prof. Dr. M. Hoefer
8. August 2022 (Erstklausur)
Klausurnummer: **0001**

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nachname (Druckschrift)**:** _____

Vorname (Druckschrift)**:** _____

Matrikelnummer: _____

Studiengang: _____

Die folgenden Hinweise sind zu beachten:

- Schreiben Sie Ihren Namen **nur auf dieses Titelblatt**. Sollten Sie Zusatzpapier bekommen, vermerken Sie darauf unbedingt Ihre Klausurnummer **0001**.
- Merken oder notieren Sie sich Ihre Klausurnummer **0001**, da nur unter dieser Nummer die Ergebnisse veröffentlicht werden.
- Diese Klausur ist für die Prüfungsvariante **Algo-1 (beide Teile)** und besteht aus den Aufgaben **1 – 10**.
- Legen Sie Ihre **Goethe-Card** deutlich sichtbar an Ihren Platz, damit wir während der Klausur Ihre Identität überprüfen können.
- Überprüfen Sie bitte, ob die Klausur aus **?** durchnummerierten **Vorder- und Rückseiten** besteht. Beachten Sie, dass die Aufgabenstellungen sowohl auf den Vorder- als auch auf den Rückseiten stehen.
- Es dürfen nur **dokumentenechte Stifte** in den Farben blau und schwarz verwendet werden. Insbesondere ist die Nutzung von Tintenlöschern und Tipp-Ex untersagt. Zugelassene Hilfsmittel:
  1 Blatt DIN A4 mit handschriftlichen Notizen (beidseitig).
- Das Mitbringen nicht zugelassener Hilfsmittel stellt eine Täuschung dar und führt zum Nichtbestehen der Klausur. **Schalten Sie bitte deshalb alle elektronischen Geräte, insbesondere Handys und Smartwatches, vor Beginn der Klausur aus.**
- Sollte der Platz unter einer Aufgabe nicht ausreichen, **nutzen Sie bitte die Zusatzblätter am Ende**. Weitere Blätter sind bei Bedarf erhältlich.
- Wenn sich Ihre Lösung zu einer Aufgabe teilweise oder ganz auf Zusatzblättern befindet, vermerken Sie dies entsprechend bei der Aufgabe und auf dem Zusatzblatt.
- 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 explizit verlangt.
- Die Klausur ist mit Sicherheit bestanden, wenn mindestens **50%** der Höchstpunktzahl (ohne Bonifikation aus den Übungen) erreicht wird. Die Bearbeitungszeit beträgt **180 Minuten**.

Viel Erfolg!

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

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	22	18	14	16	17	13	22	15	14	9
Erreicht										

Klausur	Bonifikation

1 0032211 00015

Aufgabe 1: Asymptotische Notation und Codeanalyse — $6 + 6 + 6 + 4 = 22$ Punkt(e)

a) Geben Sie jeweils eine Funktion $f : \mathbb{N} \mapsto \mathbb{R}$ an, welche die angegebenen Wachstumseinschränkungen einhält.

$f \in o(n)$ $\quad$ $f \in \omega(\log n)$ $\quad$ $f(n) =$ $\sqrt{n}$

$f \in \mathcal{O}(n^2)$ $\quad$ $f \in \Omega(n \log n)$ $\quad$ $f(n) =$ $n^{1.5}$

$f \in \Theta(f^2)$ $\quad$ $f(n) =$ 1

Lösungsskizze: Siehe Textfelder.

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

b) Lösen Sie die folgenden Rekursionsgleichungen auf. Für alle Rekursionsgleichungen gilt $T(1) = 1$. Nehmen Sie vereinfachend an, dass n so gewählt ist, so dass sämtliche Divisionen restfrei aufgehen.

$T(n) = 2 \cdot T(\frac{n}{2}) + n$ $\quad$ $T(n) = \Theta($ $n \log n$ $)$

$T(n) = 9 \cdot T(\frac{n}{3}) + 2n + 1$ $\quad$ $T(n) = \Theta($ n^2 $)$

$T(n) = T(\frac{3}{6}n) + \sqrt{n}$ $\quad$ $T(n) = \Theta($ $\sqrt{n}$ $)$

Lösungsskizze: Siehe Textfelder.

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

c) Geben Sie für die drei folgenden Algorithmen in Pseudo-Code jeweils die Laufzeit in Θ-Notation an.

```
for i = 0...n do
    j = i;
    while j > 0 do
        j = j - 1;
```

$\Theta($ n^2 $)$

```
if n > 2022 then
    i = n * n;
    while i > 2 do
        i = ⌊i/2⌋;
```

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

```
i = 1; j = n;
while i < j do
    i = i + 1;
    while 2j ≥ n do
        j = j - 1;
```

$\Theta($ n $)$

Lösungsskizze: Siehe Textfelder.

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

d) Sei `A[1...n]` ein Array mit Zahlen. Die Rekursion `rek` wird mit folgendem Algorithmus in Pseudo-Code berechnet:

```
int rek(int a, int b) {
    if(a == b) return A[a];
    int m = ⌊ (a+b)/2 ⌋;
    int p = rek(a, m);
    int q = rek(m+1, b);
    if(p >= q) return p;
    return q;
}
```

i) **(2 Punkte)** Welchen Wert berechnet der Aufruf `rek(1,n)`?

`rek(1,n)` berechnet $\max_{i=1\ldots n} A[i]$.

ii) **(2 Punkte)** Welche der folgenden Rekursionsgleichungen beschreibt die Laufzeit des Aufrufs `rek(1,n)`? Kreuzen Sie genau eine Antwortmöglichkeit an.

- ☐ $T(n) = 2 \cdot T(n/2) + O(n)$
- ☐ $T(n) = T(n/2) + O(n)$
- ☐ $T(n) = 2 \cdot T(n/2) + O(1)$ ✓
- ☐ $T(n) = T(n-1) + O(1)$

Lösungsskizze: $T(n) = 2 \cdot T(n/2) + O(1)$.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

a) Gegeben sei das Array `H[1...n]` eines *Max-Heaps* mit $n = 10$ paarweise verschiedenen Zahlen.

i) **(4 Punkte)** Geben Sie die Menge aller Indizes aus $\{1, \ldots, 10\}$ an, an denen in `H` die *zweitgrößSSte* Zahl stehen kann.

Lösungsskizze: 2, 3

ii) **(5 Punkte)** Geben Sie die Menge aller Indizes aus $\{1, \ldots, 10\}$ an, an denen in `H` die *kleinste* Zahl stehen kann.

Lösungsskizze: 10, 9, 8, 7, 6

⇑⇑⇑ _____ / 9 Punkt(e) ⇑⇑⇑

b) Erweitern Sie die *Max-Heap*-Datenstruktur mit den Operationen `insert` und `delete_max` um eine Operation `decrease_all(Δ)`, die alle Prioritäten im Heap um den Wert $\Delta \geq 0$ verringert. Die Laufzeit der Operation `decrease_all` soll $\mathcal{O}(1)$ betragen. Die asymptotische worst-case Laufzeit der Operationen `insert` und `delete_max` soll durch Ihre Änderungen nicht schlechter werden.

Beschreiben Sie alle Änderungen an der Max-Heap-Datenstruktur. Beschreiben Sie außerdem den (geänderten) Ablauf der Operationen `insert` und `delete_max` sowie den Ablauf der neuen Operation `decrease_all`. Begründen Sie auch, warum die Laufzeitschranken von allen Operationen eingehalten werden.

Lösungsskizze:
Zusätzlich zum Heap-Array wird noch ein Offset δ gespeichert, welcher zu Beginn mit $\delta = 0$ initialisiert wird. Beim Aufruf von `decrease_all(Δ)` wird einfach $\delta = \delta - \Delta$ ausgeführt.

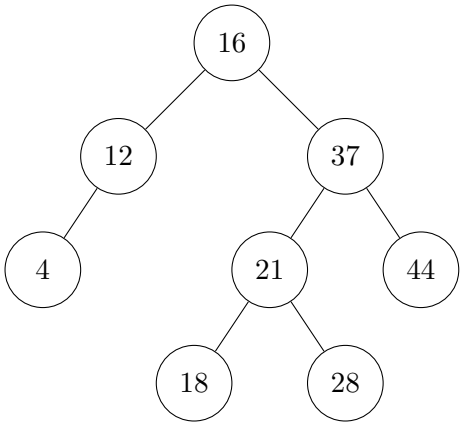
Beim Aufruf von `insert(x,p)` wird zunächst p um δ erhöht und anschließend wie gehabt vorgegangen.

Beim Aufruf von `delete_max()` wird wie gehabt vorgegangen und am Ende der Rückgabewert um δ verringert.

Die Operationen `insert` und `delete_max` wurden um Operationen konstanter Laufzeit erweitert, somit ändert sich deren Laufzeit nicht. Die Operation `decrease_all` läuft gänzlich in Konstantzeit.

⇑⇑⇑ _____ / 9 Punkt(e) ⇑⇑⇑

Gegeben sei der folgende AVL-Baum T mit dem Knoten mit Schlüssel 16 als Wurzel:



a) Geben Sie die Reihenfolge der besuchten Knoten eines *Inorder*-Durchlaufs von T an.

Reihenfolge: 4, 12, 16, 18, 21, 28, 37, 44.

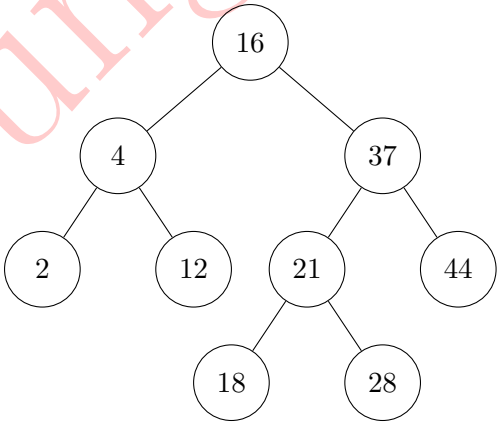
⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Der Schlüssel 2 wird in T eingefügt.

Geben Sie den daraus resultierenden Baum graphisch an (nachdem alle Reparaturschritte durchgeführt wurden).

Nennen Sie auSSerdem die von Ihnen durchgeführten Reparaturschritte (Links- und Rechtsrotationen) in chronologischer Reihenfolge und in welchen Knoten diese ausgeführt wurden.

Lösungsskizze: Zack-Zack-Fall. Resultat:



Reparaturschritte: Rechtsrotation in Schlüssel 12.

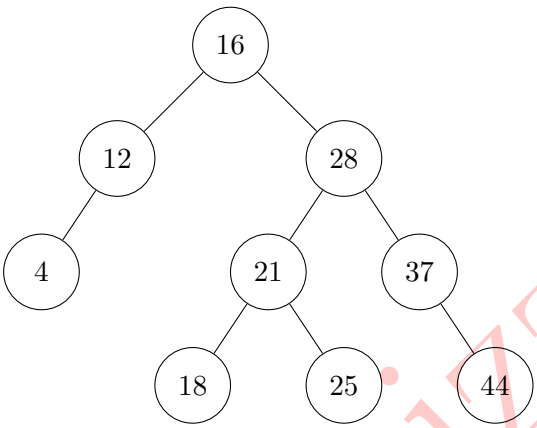
⇑⇑⇑ _____ / 5 Punkt(e) ⇑⇑⇑

c) Es sei wieder der *ursprüngliche* Baum T aus der obigen Abbildung gegeben. Der Schlüssel 25 wird in T eingefügt.

Geben Sie den daraus resultierenden Baum graphisch an (nachdem alle Reparaturschritte durchgeführt wurden).

Nennen Sie auSSerdem die von Ihnen durchgeführten Reparaturschritte (Links- und Rechtsrotationen) in chronologischer Reihenfolge und in welchen Knoten diese ausgeführt wurden.

Lösungsskizze: Zack-Zick-Fall. Resultat:



Reparaturschritte: Linksrotation in Schlüssel 21, Rechtsrotation in Schlüssel 37.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

a) Gegeben sei die folgende Hashtabelle der GröSSe $m = 11$.

0	1	2	3	4	5	6	7	8	9	10
11	34		24	14		17	18	19	31	40

Es wird Hashing mit linearem Austesten, $h_i(x) = (f(x)+i) \mod 11$ (für $i = 0, 1, ..., 10$), mit der Hashfunktion mit $f(x) = x \mod 11$ verwendet.

Fügen Sie die folgenden Schlüssel in der gegebenen Reihenfolge in die obige Hashtabelle ein:

$$19, 40, 14, 34\,.$$

Lösungsskizze: Siehe Hashtabelle.

⇑⇑⇑ _____ / 5 Punkt(e) ⇑⇑⇑

b) Betrachten Sie nun eine Tabelle der GröSSe $m = 5$ und Hashing mit Verkettung für die Hashfunktion $f(x) = x \mod 5$. Es seien bereits die Schlüssel aus einer Menge M in der Hashtabelle gespeichert. Nun wird der Schlüssel 13 eingefügt.

i) **(3 Punkte)** Für welche der folgenden Mengen M ist die Einfügezeit von Schlüssel 13 am gröSSten? Kreuzen Sie genau eine Antwortmöglichkeit an.

☐ $M := \{3, 5, 7\}$

☐ $M := \{1, 6, 9\}$

☐ $M := \{0, 2, 4\}$

☐ $M := \{1, 3, 8\}$ ✓

Lösungsskizze: $M := \{1, 3, 8\}$ (Option 4).

ii) **(2 Punkte)** Wie groSS ist der Auslastungsfaktor der Tabelle *nach* der Einfügung von Schlüssel 13?

Lösungsskizze: $\lambda = \frac{4}{5}$

⇑⇑⇑ _____ / 5 Punkt(e) ⇑⇑⇑

c) Es ist das Universum $U = \{0, 1, \ldots, 11\}$ gegeben. Betrachten Sie die Klasse H von Hashfunktionen

$$H = \{h_i \mid 0 \leq i < m \ , \ h_i(x) = (x + i) \mod m\}.$$

Ist H 2-universell für eine TabellengröSSe von $m = 11$? Beweisen Sie Ihre Antwort.

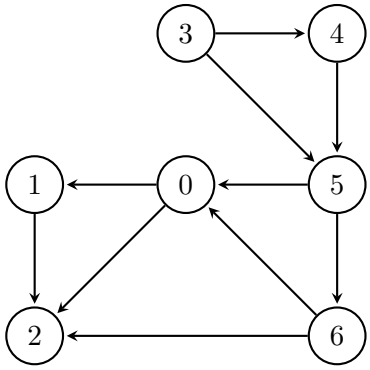
Lösungsskizze:
Nein. Wenn H 2-universell wäre, müsste für alle $x, y \in U$ mit $x \neq y$ gelten:

$$\frac{|\{h \in H \mid h(x) = h(y)\}|}{|H|} \leq \frac{2}{m}$$

Die Schlüssel 0 und 11 werden aber von allen Hashfunktionen in H auf die gleiche Zelle abgebildet, denn f.a. i gilt $h_i(0) = (0 + i) \mod 11 = i \mod 11 = (11 + i) \mod 11 = h_i(11)$. Somit ist $\frac{|\{h \in H \mid h(0) = h(11)\}|}{|H|} = 1$ und obige Bedingung nicht erfüllt für $m = 11$.

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

Betrachten Sie den folgenden gerichteten Graphen $G = (V, E)$:



a) i) **(2 Punkte)** Geben Sie eine topologische Sortierung der Knoten in G an.

Topologische Sortierung: 3, 4, 5, 6, 0, 1, 2

Lösungsskizze: Siehe Textfeld.

ii) **(2 Punkte)** Existiert mehr als eine topologische Sortierung für G? Begründen Sie Ihre Antwort kurz.

Lösungsskizze: Nein, da es in jedem Auswahlschritt nur genau eine Quelle gibt.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Angenommen, es wird eine Breitensuche auf G ausgeführt, die im Knoten 4 beginnt. Dabei werden die Knoten in aufsteigender Reihenfolge besucht, falls ein Knoten mehr als einen Nachfolger hat. Geben Sie an, an welcher Stelle in der Besuchsreihenfolge die Knoten 0, 1, 5 sowie 6 besucht werden.

i) **(1 Punkt) Knoten 0 :** An 3 ter Stelle.

ii) **(1 Punkt) Knoten 1 :** An 5 ter Stelle.

iii) **(1 Punkt) Knoten 5 :** An 2 ter Stelle.

iv) **(1 Punkt) Knoten 6 :** An 4 ter Stelle.

Lösungsskizze: Siehe Textfelder.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

c) i) **(6 Punkte)** Betrachten Sie nun einen Aufruf der Funktion `Tiefensuche()` aus der Vorlesung auf dem obigen Graphen G. Nehmen Sie an, dass alle Nachbarn eines Knotens stets in aufsteigender Reihenfolge in dessen Adjazenzliste stehen.

Geben Sie für jede Kante $e \in E$ von G an, ob es sich bei dieser um eine Baum- (B), Vorwärts- (V), Rückwärts- (R) oder Querkante (Q) handelt.

$e := (0,1)$: *B*	$e := (0,2)$: *V*	$e := (1,2)$: *B*
$e := (3,4)$: *B*	$e := (3,5)$: *V*	$e := (4,5)$: *B*
$e := (5,0)$: *Q*	$e := (5,6)$: *B*	$e := (6,0)$: *Q*
$e := (6,2)$: *Q*		

Lösungsskizze: Siehe Textfelder.

ii) **(3 Punkte)** Kommen alle vier genannten Kantentypen in G vor? Geben Sie eine kurze Begründung anhand der Struktur von G.

Lösungsskizze: Nein. Es kommen keine Rückwärtskanten vor, da G kreisfrei ist.

⇑⇑⇑ _____ / 9 Punkt(e) ⇑⇑⇑

Aufgabe 6: Haustiere verteilen — $4 + 9 = 13$ Punkt(e)

Alice und Bob möchten die Haustiere in der Menge $V = \{0, 1, \ldots, n-1\}$ untereinander aufteilen. Ein *Konfliktgraph* $G = (V, E)$ beschreibt, welche Tiere nicht zusammen gehalten werden können, d.h., zwischen Tier u und Tier v existiert genau dann eine Kante $\{u, v\} \in E$, wenn Tier u und Tier v sich nicht vertragen.

Ein Mengenpaar (A, B) heiSSt *Aufteilung*, wenn $A \cap B = \emptyset$ und $A \cup B = V$ gilt. Eine Aufteilung (A, B) heiSSt *gültig*, wenn jede Kante des Graphen einen Endpunkt in A und einen Endpunkt in B hat. Eine gültige Aufteilung existiert genau dann, wenn G bipartit ist. Dabei müssen die Mengen A und B nicht gleich groSS sein.

Zwei Aufteilungen (A, B) und (A', B') sind verschieden genau dann, wenn $A \neq A'$ oder $B \neq B'$ gilt. Insbesondere sind (A, B) und (B, A) verschieden.

a) Nehmen Sie für diesen Aufgabenteil an, dass eine gültige Aufteilung existiert. Sei t die Anzahl der Zusammenhangskomponenten von G. Geben Sie die Anzahl gültiger Aufteilungen in Abhängigkeit von t an.

Lösungsskizze:

2^t

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Entwerfen Sie einen Algorithmus, der die Anzahl der verschiedenen gültigen Aufteilungen der Tiere berechnet. Der Konfliktgraph ist in Adjazenzlistendarstellung gegeben. Es ist nicht bekannt, ob überhaupt eine gültige Aufteilung existiert. Ihr Algorithmus soll Laufzeit $\mathcal{O}(|V| + |E|)$ haben.

Beschreiben Sie zuerst Ihre Idee und geben Sie dann den Algorithmus in Pseudo-Code. Begründen Sie auch, warum die Laufzeitschranke eingehalten wird.

Lösungsskizze:

Idee: Wir führen eine Tiefensuche auf G aus und färben die Knoten entlang der gelaufenen Kanten alternierend mit zwei Farben entsprechend der beiden Mengen A und B. Ist eine konfliktfreie Färbung an irgendeiner Stelle nicht möglich, geben wir 0 aus und terminieren. Ansonsten endet die Tiefensuche und wir geben 2^t aus, wobei t die Anzahl der Zusammenhangskomponenten ist, die wir ebenfalls während der Tiefensuche zählen.

Pseudo-Code:

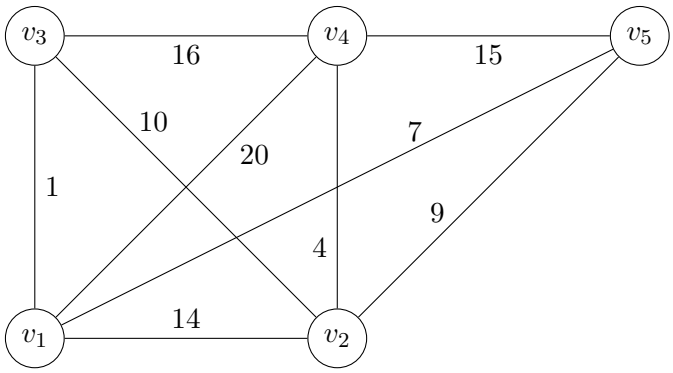
```
void tsuche(int v, int farbe) {    // farbe ist 1 oder 2
    besucht[v] = farbe;
    for(Knoten *p = A[v]; p!=0; p = p->next) {
        if(besucht[p->name] == 0) {   // falls Nachbar ungefärbt
            tsuche(p->name, 3-farbe);    // Nachbarn mit invertierter Farbe färben
        }
        else if(besucht[p->name] == farbe) {    // falls Nachbar gleiche Farbe
            Gebe 0 aus und terminiere.    // Konflikt
        }
    }
}

for(int k=0; k<n; k++) besucht[k] = 0;
int num = 1;
for(int k=0; k<n; k++) {
    if(besucht[k] == 0) {    // neue Zusammenhangskomponente gefunden
        num = 2*num;
        tsuche(k, 1);
    }
}
Gebe num aus.
```

Laufzeit: Die Operationen der Tiefensuche wurden nur um konstante Operationen ergänzt, also bleibt die Laufzeit $\mathcal{O}(|V| + |E|)$.

⇑⇑⇑ _____ / 9 Punkt(e) ⇑⇑⇑

a) Gegeben sei der folgende ungerichtete Graph mit Kantengewichten:



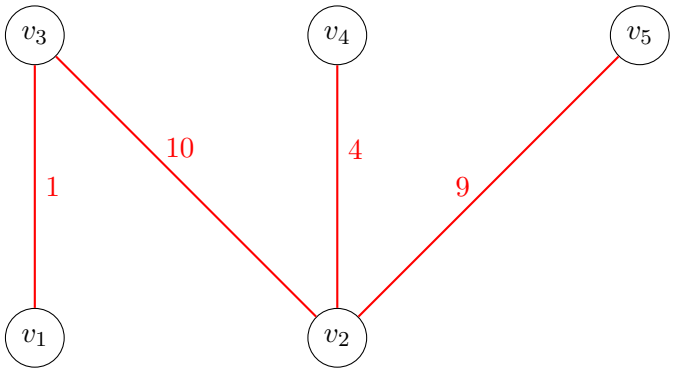
i) **(15 Punkte)** Bestimmen Sie mit Hilfe des Algorithmus von Dijkstra die kürzesten Wege von Knoten $\mathbf{v_4}$ zu allen anderen Knoten. Füllen Sie hierzu die nachfolgende Tabelle aus, wobei jede Zeile einem Schritt des Algorithmus entspricht.

Die Spalte **d** soll den aktuellen Distanzwert des Knoten enthalten, die Spalte **p** den aktuellen Vorgänger des Knoten im Elternarray.

Menge S	v_1		v_2		v_3		v_4		v_5	
	d	**p**	**d**	**p**	**d**	**p**	**d**	**p**	**d**	**p**
$\{v_4\}$	20	v_4	4	v_4	16	v_4	0	–	15	v_4
$\{v_4, v_2\}$	18	v_2	”	”	14	v_2	”	”	13	v_2
$\{v_4, v_2, v_5\}$	18	v_2	”	”	14	v_2	”	”	”	”
$\{v_4, v_2, v_5, v_3\}$	15	v_3	”	”	”	”	”	”	”	”
$\{v_4, v_2, v_5, v_3, v_1\}$	”	”	”	”	”	”	”	”	”	”

Lösungsskizze: Siehe Tabelle.

ii) **(4 Punkte)** Zeichnen Sie den resultierenden Baum kürzester Wege:



⇑⇑⇑ _____ / 19 Punkt(e) ⇑⇑⇑

b) Sei $G = (V, E)$ nun ein beliebiger gewichteter ungerichteter Graph. Wir transformieren die Kantengewichte von G: Das Gewicht $w(e)$ einer Kante $e \in E$ ändert sich zu $w'(e)$.

In welchem der folgenden Fälle werden sich die kürzesten Wege in der Transformation niemals ändern? Kreuzen Sie genau eine Antwortmöglichkeit an.

☐ $w'(e) = 2 \cdot w(e) + 1$

☐ $w'(e) = w(e) \cdot w(e)$

☐ $w'(e) = \log_2(|V| + 2) \cdot w(e)$ ✓

☐ $w'(e) = \log_2(w(e))$

Lösungsskizze: $w'(e) = \log_2(|V| + 2) \cdot w(e)$

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

a) Gegeben seien $n > 0$ Zeichen $z_1, z_2, \ldots, z_n$, wobei Zeichen z_i die absolute Häufigkeit $H(z_i) = 2^{i-1}$ hat, für $i \in \{1, ..., n\}$.

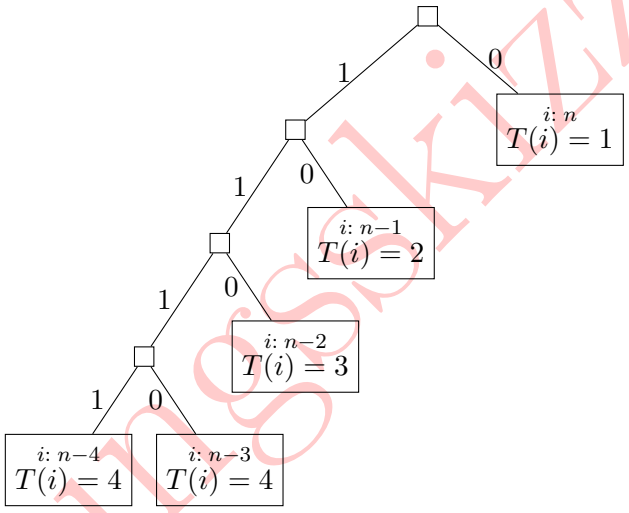
Geben Sie die Tiefe des zugehörigen Huffman-Baums asymptotisch exakt an. Begründen Sie Ihre Antwort kurz.

Tiefe: $\Theta(n)$

Kurze Begründung:

Lösungsskizze: Für alle i gilt, dass $\sum_{k=1}^{i-1} 2^{k-1} = 2^{i-1} - 1 < 2^{i-1}$, d.h., die absolute Häufigkeit von i übertrifft jegliche kumulierten Häufigkeiten seiner Vorgänger in der Sortierung. Der Algorithmus verschmilzt also die Zeichen sukzessiv von vorne nach hinten gemäSS der Sortierung. Dadurch wächst seine Tiefe linear in n.

Hilfsskizze für $n = 5$:



⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Bei der *Union-Find*-Datenstruktur wird in jedem Union-Schritt stets die Wurzel des „kleineren“ Baums unter die Wurzel des „gröSSeren“ Baums gehängt.

Für welche der folgenden Definitionen von „GröSSe“ eines Baums B kann die Laufzeit einer `find`-Operation asymptotisch linear in der Anzahl der Elemente in der Union-Find-Datenstruktur werden? Kreuzen Sie genau eine Antwortmöglichkeit an.

Wenn die „GröSSe“ von B definiert wird als...

☐ ...die Knotenzahl von B.

☐ ...der durchschnittliche Knotengrad von B.

☐ ...der gröSSte in B vorkommende Knotenindex. ✓

☐ ...die Kantenanzahl von B.

Lösungsskizze: ...der gröSSte in B vorkommende Knotenindex.

c) Gegeben sei ein gewichteter gerichteter Graph $G = (V, E)$ mit Knotenmenge $V = \{1, 2 \ldots, n\}$ und Kantenmenge $E = \{(i, i+1) \mid 1 \leq i < n\}$. Geben Sie die exakte asymptotische Laufzeit folgender Algorithmen auf G in Abhängigkeit von n an.

i) **(2 Punkte)** Dijkstras Algorithmus mit Startknoten $s = 1$:

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

Lösungsskizze:

ii) **(2 Punkte)** Algorithmus von Floyd:

$\Theta($ n^3 $)$

Lösungsskizze:

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

d) Im Folgenden wird *Stabiles Matching* für zwei Mengen A und B mit $|A| = |B| = n$ betrachtet. Die Agenten in A sind aktiv und machen Anträge. Alle Agenten in einer Menge haben eine vollständige Präferenzordnung über die Agenten in der jeweils anderen Menge.

Geben Sie an, nach wie vielen Anträgen der Deferred-Acceptance Algorithmus bei den folgenden Präferenzordnungen terminiert.

i) **(2 Punkte)** Alle Agenten in A haben voneinander verschiedene Agenten aus B an der ersten Stelle ihrer Präferenzordnung, d.h., denselben Favoriten.

Lösungsskizze: Nach n Anträgen. (Jeder Agent in A macht Antrag an seinen Favoriten in B und wird unwiderruflich genommen)

ii) **(2 Punkte)** Alle Agenten in A haben eine identische Präferenzordnung.

Lösungsskizze: Nach exakt $(n^2 + n)/2$ Anträgen, asymptotisch $\Theta(n^2)$ (Alle Agenten in A machen Antrag an gemeinsamen Favoriten in B, der wiederum seinen Favoriten aus A wählt; danach machen $n - 1$ Agenten aus A Antrag an Zweitfavoriten, der wiederum seinen Favoriten aus A wählt usw. Daher

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2} = \frac{n^2 + n}{2}$$

für die Anzahl der gemachten Anträge).

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

Aufgabe 9: Dynamische Programmierung — $7 + 7 = 14$ Punkt(e)

Ein Gebiet ist in Planquadrate $(x, y) \in \{0, 1, \ldots, n\}^2$ aufgeteilt. Jedes Planquadrat (x, y) hat eine Höhe $h(x, y)$ in Metern über dem Meeresspiegel.

Ein Wanderer möchte von Planquadrat $(0, 0)$ zu (n, n) laufen und dabei möglichst wenige Höhenmeter überwinden. Die Wahl seines Weges unterliegt dabei folgenden Einschränkungen: Befindet er sich in einem Planquadrat (x, y), kann er entweder zum nächsten Planquadrat $(x+1, y)$ in x-Richtung oder zum nächsten Planquadrat $(x, y+1)$ in y-Richtung laufen, sofern er dabei das Gebiet nicht verlässt.

Die Höhendifferenz eines Weges $(x_0, y_0) \to (x_1, y_1) \to (x_2, y_2) \to \cdots \to (x_k, y_k)$ von Planquadrat (x_0, y_0) zu Planquadrat (x_k, y_k) ist gegeben durch $\sum_{\ell=1}^{k} |h(x_\ell, y_\ell) - h(x_{\ell-1}, y_{\ell-1})|$.

Um die minimale Höhendifferenz eines Weges von $(0, 0)$ nach (n, n) zu berechnen, werden Teilprobleme für $x, y \in \{0, 1 \ldots, n\}$ wie folgt definiert:

$$\text{opt}(x, y) = \text{minimale Höhendifferenz, die überwunden werden muss, um von } (0, 0) \text{ zu } (x, y) \text{ zu gelangen.}$$

a) Geben Sie eine Rekursionsgleichung sowie Basisfälle für obige Teilprobleme an.

Basisfälle:

Lösungsskizze: $\text{opt}(0, 0) = 0$

Rekursionsgleichung:

Lösungsskizze:

$$\text{Für } x, y > 0: \quad opt(x, y) = \min \left\{ \begin{array}{l} |h(x, y) - h(x-1, y)| + opt(x-1, y), \\ |h(x, y) - h(x, y-1)| + opt(x, y-1) \end{array} \right\} \tag{1}$$

$$\text{Für } x > 0: \quad opt(x, 0) = |h(x, 0) - h(x-1, 0)| + opt(x-1, 0) \tag{2}$$

$$\text{Für } y > 0: \quad opt(0, y) = |h(0, y) - h(0, y-1)| + opt(0, y-1) \tag{3}$$

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

b) Entwickeln Sie aus Ihrer Rekursionsgleichung ein dynamisches Programm zur Berechnung der minimal nötigen Höhendifferenz. Ihr Algorithmus soll Laufzeit $\mathcal{O}(n^2)$ haben.

Geben Sie Ihren Algorithmus in Pseudo-Code an und analysieren Sie dessen Laufzeit.

Lösungsskizze:

$opt(0,0) = 0$;
for $x = 1 \ldots n$ **do**
  $opt(x,0) = |h(x,0) - h(x-1,0)| + opt(x-1,0)$;
for $y = 1 \ldots n$ **do**
  $opt(0,y) = |h(0,y) - h(0,y-1)| + opt(0,y-1)$;
for $x = 1 \ldots n$ **do**
  for $y = 1 \ldots n$ **do**
    $a = |h(x,y) - h(x-1,y)| + opt(x-1,y)$;
    $b = |h(x,y) - h(x,y-1)| + opt(x,y-1)$;
    if $a \geq b$ **then**
      $opt(x,y) = a$;
    else
      $opt(x,y) = b$;
Gebe $opt(n,n)$ aus;

Laufzeit: Die ersten beiden Schleifen werden je n mal durchlaufen. Die beiden geschachtelten Schleifen werden ebenfalls je n mal durchlaufen. Insgesamt ergibt sich also $\Theta(n^2)$ (die Angabe $\mathcal{O}(n^2)$ genügt).

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit ganzzahligen Kantengewichten aus der Menge $\{1, 2, \ldots, k\}$, wobei k eine *Konstante* unabhängig von der GröSSe des Graphen ist.

Entwerfen Sie einen Algorithmus, der in Zeit $\mathcal{O}(|V| + |E|)$ einen minimalen Spannbaum von G berechnet. G ist in Adjazenzlistendarstellung gegeben.

Beschreiben Sie Ihren Algorithmus in Worten und begründen Sie, warum die gegebene Laufzeitschranke eingehalten wird.

Lösungsskizze:

Änderungen am Algorithmus:

Der im DJP-Algorithmus verwendete Min-Heap wird durch eine Prioritätswarteschlange P ersetzt, die alle Heap-Operationen in Konstantzeit unterstützt. P besteht aus einem Array $A[1 \ldots k]$ von k Listen. Die Liste $A[i]$ enthält alle Elemente mit Priorität i.

Beim Einfügen eines Elements x der Priorität p (`insert`), wird das Element x an Liste $A[p]$ angehängt (Laufzeit $\mathcal{O}(1)$).

Beim Suchen und Entfernen des Elements mit kleinster Priorität (`delete_min`), wird die erste Liste $A[i]$, $i = 1 \ldots k$, gesucht, die nicht leer ist (Laufzeit $\mathcal{O}(k) = \mathcal{O}(1)$). In $A[i]$ wird dann das erste Element gelöscht und zurückgegeben (Laufzeit $\mathcal{O}(1)$).

Beim Verringern der Priorität p eines Elements x auf $q < p$, wird x zunächst aus der Liste $A[p]$ entfernt und anschlieSSend mittels `insert` mit neuer Priorität eingefügt (Laufzeit $\mathcal{O}(1)$). Das Entfernen aus der Liste $A[p]$ erfolgt mittels eines Zeiger-Arrays `Ptr[]`, in dem für jedes Element x die Adresse seines Vorgängers in der Liste steht.

Laufzeit: Die Laufzeit jeder einzelnen Heap-Operationen ist nun $\mathcal{O}(1)$. Insgesamt werden höchstens $\mathcal{O}(|V| + |E|)$ Heap-Operationen durchgeführt, also wird die geforderte Laufzeit eingehalten.

Bemerkung: Bei obigem Verfahren handelt es sich um Breitensuche mit veränderter Queue (wie generell bei Prim und Dijkstra).

Notizseite

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Hinweis auf Seite 24 platziert wurde.

Notizseite

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Hinweis auf Seite 25 platziert wurde.