

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**.
- Diese Klausur ist für die Variante **Algo1 (vollständig)**.
- Die Klausur besteht aus den Aufgaben **1 – 9**; Rückseiten werden verwendet.
- 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 und Tipp-Ex** 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**.

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

Aufgabe	1	2	3	4	5	6	7	8	9
Erreichbar	10	23	12	22	14	20	19	24	16
Erreicht									

Klausur	Bonifikation

Für $n, k \in \mathbb{N}_{>0}$ sei $Z_{k,n}$ die Menge aller Wörter in $\{0, \dots, k-1\}^n$, deren Ziffern von links nach rechts aufsteigend sortiert sind:

$$Z_{k,n} := \{ d_1 d_2 \cdots d_n \mid d_1, \dots, d_n \in \{0, 1, \dots, k-1\}, \text{ mit } d_i \leq d_{i+1} \forall 1 \leq i < n \}$$

Beispiele:

$$\{000, 001, 011, 111\} = Z_{2,3} \subseteq Z_{3,3} \subseteq Z_{4,3}, \quad 0112 \in Z_{3,4}$$

Gegenbeispiele:

$$\begin{aligned} 100 &\notin Z_{2,3} && \text{da nicht aufsteigend sortiert (0 kommt nach 1)} \\ 002 &\notin Z_{2,3} && \text{da Ziffer 2 nicht zur Sprache gehört} \\ 0000 &\notin Z_{2,3} && \text{da das Wort Länge 4 statt 3 hat} \end{aligned}$$

Im Folgenden möchten wir die **Anzahl solcher Worte** $N_{k,n} = |Z_{k,n}|$ mit Hilfe eines **dynamischen Programms** berechnen. Modellieren Sie hierfür zunächst geeignete Teilprobleme $T[\dots]$, die sich effizient rekursiv berechnen lassen. Nennen Sie dann eine geeignete Rekursionsgleichung und minimal notwendige Basisfälle. Nennen Sie wie sich das Endergebnis $N_{k,n}$ aus Ihrer Rekursion ergibt.

Natürlichsprachliche Beschreibung der Teilprobleme $T[\dots]$:

Rekursion: $T[\underline{\hspace{2cm}}] = \underline{\hspace{10cm}}$

Rekursionsbasis: $\underline{\hspace{10cm}}$

Endergebnis: $N_{k,n} = \underline{\hspace{10cm}}$



In der Vorlesung präsentierten wir die Union-Find Datenstruktur sinngemäß wie folgt:

```
struct UnionFind {
    int[] parent; // Elternarray
    int[] helper; // siehe Aufgabenteil b
};
```

a) Betrachten Sie folgende langsame Implementierung:

<pre>// Initialisiere UnionFind function create(int n) { UnionFind uf; // allokieren Array[0, ..., n-1]: uf.parent = new int[n]; for(int i = 0; i < n; ++i) { uf.parent[i] = i; } return uf; }</pre>	<pre>// Verschmelze Bäume von u und v. // uf wird als Referenz übergeben // und kann verändert werden function union(UnionFind uf, int u, int v) { while (uf.parent[u] != u) { u = uf.parent[u]; } while (uf.parent[v] != v) { v = uf.parent[v]; } uf.parent[v] = u; }</pre>
--	---

Wir suchen eine Sequenz von $\Theta(n)$ sukzessiven `union` Aufrufen, deren **Gesamtlaufzeit** **möglichst groß** ist. Die Sequenz soll unmittelbar nach einem `create(n)` starten.

Beschreiben Sie eine solche Sequenz für alle $n > 1$, nennen Sie ihre Gesamtlaufzeit asymptotisch exakt¹, und begründen Sie wie sich diese Laufzeit ergibt.

Gesamtlaufzeit: _____

↑↑↑ _____ / 8 Punkt(e) ↑↑↑

¹Übereinstimmende obere und untere Schranken.



Aufgabe 2: Union-Find (Fortsetzung)

- b) In der Vorlesung und Übung diskutierten wir unterschiedliche Ansätze, die Laufzeit mit Hilfe eines weiteren Arrays (im Folgenden **helper** genannt) zu verbessern. Wählen und beschreiben Sie eine dieser Möglichkeiten natürlichsprachlich. Nennen Sie die verbesserte Laufzeit von **union** und **find** und skizzieren Sie kurz die Beweisidee.

Worst-Case Laufzeit von **union**: _____

Worst-Case Laufzeit von **find**: _____

↑↑↑ _____ / 8 Punkt(e) ↑↑↑

- c) Implementieren Sie **create** und **union** (Semantik siehe a)) einer möglichst effizienten Union-Find Datenstruktur indem Sie den folgenden **Pseudocode vervollständigen**.

```
function create(int n) {  
    UnionFind uf;  
    uf.parent = new int[n]; // allokiere Array[0, ..., n-1]  
    for(int i = 0; i < n; ++i) { uf.parent[i] = i; }  
  
    return uf;  
}
```

```
function union(UnionFind uf, int u, int v) {  
    while(uf.parent[u] != u) { u = uf.parent[u]; }  
    while(uf.parent[v] != v) { v = uf.parent[v]; }  
  
}
```

}

↑↑↑ _____ / 7 Punkt(e) ↑↑↑



Geben Sie zu jedem Zeichen das Codewort eines Huffman-Codes an, der zu folgender Häufigkeitstabelle gehört.

a	b	c	d	e	f	g
1	21	36	10	12	14	45

Immer wenn zwei Teilbäume vereint werden, können Sie diese beliebig anordnen. Sollten Sie zwischen zwei gleichwertigen Knoten auswählen müssen, können Sie beliebig entscheiden.

Der Huffman-Baum muss nicht angegeben werden und wird auch nicht bewertet.

a $\hat{=}$ _____

b $\hat{=}$ _____

c $\hat{=}$ _____

d $\hat{=}$ _____

e $\hat{=}$ _____

f $\hat{=}$ _____

g $\hat{=}$ _____

↑↑↑ _____ / 12 Punkt(e) ↑↑↑



- a) Geben Sie jeweils eine Funktion f an, welche die angegebenen Wachstumseinschränkungen einhält.

$f \in \omega(\frac{1}{n})$ $f \in o(\log n)$ $f(n) =$ _____

$f \in \omega(\log n)$ $f \in o(n)$ $f(n) =$ _____

$\log(f) \in \omega(n)$ $f(n) =$ _____

$f \in \Theta(n)$ $2^f \in \omega(2^n)$ $f(n) =$ _____

↑↑↑ _____ / 8 Punkt(e) ↑↑↑

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

$T(n) = 4 \cdot T(n - \frac{n}{2}) + 3n$ $T(n) = \Theta(\text{_____})$

$T(n) = 2 \cdot T(n - 1) + 3$ $T(n) = \Theta(\text{_____})$

$T(n) = 2 \cdot T(n - \frac{3n}{4}) + 3$ $T(n) = \Theta(\text{_____})$

$T(n) = 3 \cdot T(n - \frac{2n}{3}) + 2n$ $T(n) = \Theta(\text{_____})$

↑↑↑ _____ / 8 Punkt(e) ↑↑↑

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

Algorithmus loop1(n)
 $s = 0$
 $i = 1$
while $i * i \leq n$
 for $j = 1$ **to** i
 $s = s + 1$
 $i = i + 1$

Algorithmus loop2(n)
 $i = 1$
while $i \leq n * n$
 $i = 3 * i$

Algorithmus loop3(n)
 $i = 1$
while $i \leq n$
 $j = 0$
 while $j \leq i$
 $j = j + 1$
 $i = 2 * i$

loop1: $\Theta(\text{_____})$

loop2: $\Theta(\text{_____})$

loop3: $\Theta(\text{_____})$

↑↑↑ _____ / 6 Punkt(e) ↑↑↑



Bei der Modellierung eines dynamisches Programms ergab sich folgende Rekursion:

$$X[a, b] = \begin{cases} 1 & \text{falls } b = 0 \vee a = 0 \\ a \cdot X[a - 1, b] + b \cdot X[a - 1, b - 1] & \text{falls } a > 0 \wedge b > 0 \end{cases}$$

a) Betrachten Sie die folgende naive Implementierung der Gleichung:

```
def f(a,b):
    if a == 0 or b == 0:
        return 1
    return a*f(a-1, b) + b*f(a-1,b-1)
```

Ermitteln Sie die Laufzeit dieses Algorithmus für den Fall $a = b$ asymptotisch exakt. Begründen Sie Ihre Antwort kurz.

Laufzeit: _____

↑↑↑ _____ / 4 Punkt(e) ↑↑↑

b) Beschreiben Sie einen **iterativen** Algorithmus in **Pseudo-Code**, der für die Eingabe $a, b \in \mathbb{N}$ den Wert $X[a, b]$ in Zeit $\mathcal{O}(a \cdot b)$ berechnet. Nutzen Sie **return** um das Endergebnis $X[a, b]$ zurückzugeben.

return _____

↑↑↑ _____ / 6 Punkt(e) ↑↑↑



Aufgabe 5: Dynamische Programmierung (Fortsetzung)

- c) Beschreiben Sie einen Implementierungsansatz **natürlichsprachlich**, der den Speicher-
verbrauch auf $\mathcal{O}(b)$ Speicherzellen reduziert und in Zeit $\mathcal{O}(a \cdot b)$ arbeitet. **Begründen**
Sie kurz die Korrektheit Ihrer Antwort.

Hinweis: Sollte Ihre Lösung in b) nur $\mathcal{O}(b)$ Speicherzellen benötigen, zeigen Sie dies.

↑↑↑ _____ / 4 Punkt(e) ↑↑↑



- a) Gegeben ist folgende Hashtabelle der Größe 13. Es wird doppeltes Hashing mit den folgenden Funktionen verwendet:

$$f_i(k) = (h_1(k) + i \cdot h_2(k)) \bmod 13$$

$$h_1(k) = 5k \bmod 13$$

$$h_2(k) = 6 - (k \bmod 6)$$

0	1	2	3	4	5	6	7	8	9	10	11	12
30		11		41			20		57		8	

Fügen Sie die folgenden Elemente in der gegebenen Reihenfolge in die Hashtabelle ein.

1, 2, 3, 4, 5

Rechenhilfe:

k	1	2	3	4	5
$h_1(k)$	5	10	2	7	12

↑↑↑ _____ / 10 Punkt(e) ↑↑↑

- b) Gegeben ist folgende Hashtabelle der Größe 13. Es wird Hashing mit linearem Austesten mit den Hashfunktionen $h_i(k) = (8k + i) \bmod 13$ verwendet.

0	1	2	3	4	5	6	7	8	9	10	11	12
34	41		57	8		22		11			63	78

Fügen Sie die folgenden Elemente in der gegebenen Reihenfolge in die Hashtabelle ein.

1, 2, 3, 4, 5

Rechenhilfe:

k	1	2	3	4	5
$h_0(k)$	8	3	11	6	1

↑↑↑ _____ / 10 Punkt(e) ↑↑↑



Gegeben sei ein ungerichteter und zusammenhängender Graph $G = (V, E)$ mit $V = \{1, \dots, n\}$ und nicht-negativen Kantengewichten $w: E \rightarrow \mathbb{R}_{\geq 0}$ sowie ein fixierter Startknoten $s \in V$. Gegeben ist zudem ein Distanzarray D , das an i -ter Stelle die **Länge** eines kürzesten Weges von s nach Knoten i speichert (z.B. mittels Dijkstra berechnet).

Wir möchten nun einen gerichteten Kürzeste-Wege-Baum (KWB) $T = (V, E_T)$ berechnen, in dem s die Wurzel ist und alle Kanten in Richtung der Wurzel zeigen. Der KWB erlaubt es von jedem Knoten aus den kürzesten Weg zu s zu finden: wir müssen nur den gerichteten Kanten bis zur Wurzel folgen. Zur Berechnung des KWB modifizieren wir im Folgenden die Tiefensuche.

- a) Zeigen Sie anhand eines Beispiels, dass eine in s startende unmodifizierte Tiefensuche (DFS) Kanten wählen kann, die nicht im KWB T enthalten sein dürfen.

↑↑↑ _____ / 4 Punkt(e) ↑↑↑

- b) Beschreiben Sie **natürlich-sprachlich**, wie wir DFS durch eine Überprüfung von D modifizieren können, damit diese nur Kanten wählt, die im KWB vorhanden sein müssen. Die asymptotische Laufzeit der DFS soll hierbei unverändert bleiben.

↑↑↑ _____ / 7 Punkt(e) ↑↑↑



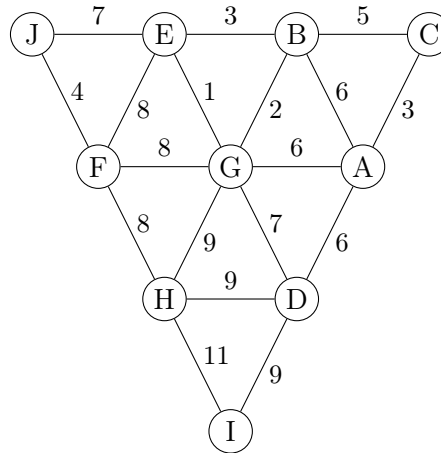
Aufgabe 7: Baum kürzester Wege (Fortsetzung)

- c) Implementieren Sie die modifizierte Tiefensuche **in Pseudocode**; diese soll alle Anforderungen aus b) erfüllen. Das Ergebnis soll als **Elternarray** ausgegeben werden. Reproduzieren Sie dabei die relevanten Teile des Algorithmus (d.h. **DFS darf nicht als Blackbox** verwendet werden), **definieren Sie alle verwendeten Symbole**, nennen Sie Datenstrukturen (inklusive der Eingabe!), und geben Sie die Rückgabe mittels **return** aus.

↑↑↑ _____ / 8 Punkt(e) ↑↑↑



- a) Auf folgendem Graph soll ein minimaler Spannbaum sowohl mit Prim's als auch Kruskal's Algorithmus berechnet werden.



Geben Sie die Kanten in der Reihenfolge an, in der sie vom jeweiligen Algorithmus in den Spannbaum eingefügt werden.

Hinweis: Sie können Kanten $\{A, B\}$ vereinfachend als AB schreiben..

Reihenfolge für Prim's Algorithmus bei Start in Knoten A:

Reihenfolge für Kruskal's Algorithmus:

↑↑↑ _____ / 16 Punkt(e) ↑↑↑



Aufgabe 8: Spannbäume (Fortsetzung)

- b) Sei $G = (V, E)$ ein gewichteter, ungerichteter, zusammenhängender Graph und eine Kantengewichtsfunktion $w: E \rightarrow \mathbb{R}_{\geq 1}$. Wir betrachten die folgende Transformation der Kantengewichte:

$$w'_a(e) = 2^{\log_a(w(e))}$$

Zeigen oder widerlegen Sie: Minimale Spannbäume bleiben unter der Transformation w'_a für alle $a \in \mathbb{N}_{>1}$ erhalten. *Hinweis:* Sie können annehmen, dass jedes Kantengewicht nur einmal vorkommt.

↑↑↑ _____ / 8 Punkt(e) ↑↑↑



a) Fügen Sie die Elemente

2, 10, 1, 6, 4, 7, 5, 9, 3, 8

in der gegebenen Reihenfolge in einen zu Beginn leeren Min-Heap ein und stellen Sie den resultierenden Heap als Array dar.

1	2	3	4	5	6	7	8	9	10

↑↑↑ _____ / 8 Punkt(e) ↑↑↑

b) Zeigen oder widerlegen Sie: Eine Inorder-Traversierung eines nicht leeren AVL-Baums liefert eine sortierte Sequenz.

↑↑↑ _____ / 8 Punkt(e) ↑↑↑



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

