Klausur Algorithmen

(Prof. Kaufmann/Bekos/Schneck, Sommersemester 2018)

- Die Bearbeitungszeit beträgt **90 Minuten**, elektronische Hilfsmittel sind verboten, nur ein handbeschriebenes DIN-A4-Blatt mit Notizen ist erlaubt.
- Von den 6 Aufgaben mit jeweils 6 möglichen Punkten werden 5 bewertet. **Markieren Sie die Aufgabe, die nicht bewertet werden soll.**

Viel Erfolg!

Name: .....
Matr.-Nr.: .....
Studiengang: .....
Angestr. Abschluss: .....

	Punkte	max
1		6
2		6
3		6
4		6
5		6
6		6
Σ		30

Aufgabe 1: (Rekursion) 6 Punkte

Betrachten Sie folgende Rekursionsgleichung: Für eine Zweierpotenz n sei

$$T(n) = \frac{n}{2} \cdot T(n/2), \text{ für } n > 1$$
$$T(1) = 1$$

Geben Sie eine geschlossene Form für $T(n)$ and und benutzen Sie dabei **nicht** das Mastertheorem. Beweisen Sie die Korrektheit Ihrer Antwort.

Aufgabe 2 (Orientierung von Graphen) 6 Punkte

Sei $G = (V, E)$ ein **ungerichteter** Graph **in Adjazenzlistendarstellung** mit $|V| = n$ und $|E| = m$. Eine *Orientierung* der Kanten bedeutet für jede ungerichtete Kante eine gerichtete Kante einzuführen, so dass aus G ein gerichteter Graph G' wird.

Ist es für einen gegebenen ungerichteten Graph immer möglich eine Orientierung zu finden, so dass der entstehende Graph zykelfrei ist? Wenn ja, entwerfen Sie einen entsprechenden Algorithmus. Wenn nein, geben Sie ein möglichst kleines Gegenbeispiel an.

Aufgabe 3 (Erkennung von Zykelgraphen) 6 Punkte

Sei $G = (V, E)$ ein **gerichteter** Graph **in Adjazenzlistendarstellung** mit $|V| = n$ und $|E| = m$. Geben Sie einen Algorithmus in Pseudocode an, der feststellt, ob G ein Zykel der Länge n ist. Benutzen Sie dabei keinerlei Hilfsfunktionen wie MST, DFS, kürzeste Wege, TopSort oder dergleichen.

Aufgabe 4 (Dynamische Programmierung) 6 Punkte

Ein String $w = w_1 \cdots w_k$ ist ein *Substring* eines Strings $s = s_1 \cdots s_n$, wenn es eine streng monoton wachsende Folge $\langle i_1, \ldots, i_k \rangle$ gibt, so dass $s_{i_j} = w_j$ für alle $j = 1, \ldots, k$ (d.h. w entsteht aus s durch Streichen von Buchstaben). Gegeben seien zwei Strings $s = s_1 \cdots s_n$ und $t = t_1 \cdots t_m$. Ein *längster gemeinsamer Substring* $w = w_1 \cdots w_k$ von s und t ist wie folgt definiert:

- w ist sowohl Substring von s als auch von t und
- jeder andere gemeinsame Substring von s und t hat eine Länge von höchstens k.

Zum Beispiel sind 'OIT', 'LIT' und 'OME' längste gemeinsame Substrings von s = ALGORITHMEN und t = KOMPLEXITÄT.

a) Geben Sie einen Algorithmus in Pseudocode an, der mit Hilfe von **Dynamischer Programmierung** einen längsten gemeinsamen Substring von s und t berechnet.

b) Begründen Sie kurz die Korrektheit und die Laufzeit Ihres Algorithmus.

Aufgabe 5 ((2,4)-Bäume) 6 Punkte

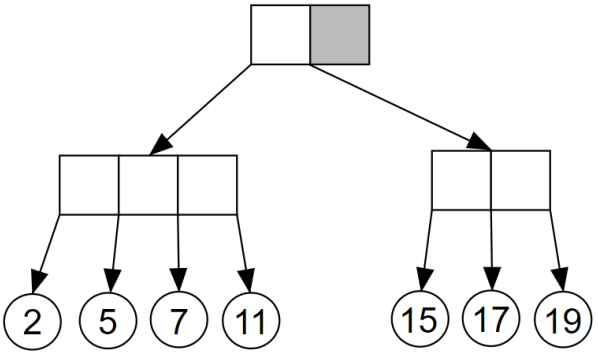
Betrachten Sie den untenstehenden (2,4)-Baum.

a) Fügen Sie die fehlenden Labels ein.

b) Fügen Sie die Schlüssel 20 und 12 in dieser Reihenfolge in den (2,4)-Baum ein und zeichnen Sie die resultierenden Bäume. Erläutern Sie Ihr Vorgehen kurz.

c) Sei T der Baum nach dem Einfügen von 20 und 12. Löschen Sie den Schlüssel 15 aus T und zeichnen Sie den resultierenden Baum. Erläutern Sie Ihr Vorgehen kurz.

d) Beschreiben Sie die Unterschiede zwischen (2,4)-Bäumen und B-Bäumen der Ordnung 2.



Aufgabe 6 (Hashing und Amortisierte Analyse) 6 Punkte

Betrachen Sie ein Hashing-Verfahren, das folgendermaßen funktioniert: Wir verwenden eine Folge von Hashtafeln $T_0, T_1, T_2, \ldots$. Die Mindestgröße von jeder Hashtafel T_i ist 4 und T_0 hat die Größe 4 und ist leer. Der *Belegungsfaktor* β_i von T_i ist

$$\beta_i = \frac{\text{Anzahl der Elemente in } T_i}{\text{Größe } |T_i| \text{ von } T_i}$$

Wenn nach **Einfügen** eines Elementes $\beta_i = 1$ gilt, dann wird eine neue Hashtafel T_{i+1} erzeugt, die die doppelte Größe von T_i hat. Andererseits wird eine Tafel T_{i+1} der halben Größe von T_i erzeugt, wenn nach **Löschen** eines Elementes $b_i \leq \frac{1}{4}$ und $|T_i| > 4$ gilt. Bei jeder **Verdopplung** und **Halbierung** werden alle Elemente aus T_i in die Tafel T_{i+1} eingefügt unter Verwendung der entsprechenden Hashfunktion für T_{i+1}.
Es wird Hashing mit Verkettung verwendet. Die Hashfunktion für Tabelle T_i ist definiert durch:

$$h_i : \mathbb{N} \to \{0, \ldots, |T_i| - 1\}, \quad h_i(x) = x \mod |T_i|.$$

a) Fügen Sie die Elemente 6, 1, 2, 8, 24, 7, 11, 19 in eine leere Hashtafel T_0 ein.

b) Löschen Sie die Elemente 8, 1, 19, 24 aus der aktuellen Hashtafel.

c) Nun sollen im Allgemeinen n Einfüge- bzw. Löschoperationen in einer zu Beginn leeren Hashtafel vorgenommen werden. Dabei sollen wieder die obigen Regeln beachtet werden. Nehmen Sie an, dass es keine Kollisionen gibt (d.h. die verketteten Listen haben höchstens die Größe 1.) Zeigen Sie dass die n Einfüge- bzw. Löschoperationen amortisierte Zeit $\mathcal{O}(n)$ brauchen.