**DECKBLATT IN DRUCKSCHRIFT AUSFÜLLEN!**

**Name:** **Vorname**: **Matrikelnr.**:
**Tutor**: □ Johanna □ Lena □ Max □ Michael

---

## Klausur zur Vorlesung

## Informatik B

( Dr. Frank Hoffmann)

**Sommersemester 2006** **19. Juli 2006**

---

Beginn: $8^{\underline{30}}$ Ende: $10^{\underline{00}}$ (90 min)

| 1. | 2. | 3. | 4. | Σ |
|---|---|---|---|---|
| /6 | /6 | /6 | /6 | /24 |

**Ich bin einverstanden, dass mein Punktergebnis zusammen mit der Matrikelnummer auf einer FU–internen Webseite erscheint:** □ **Ja** □ **Nein**

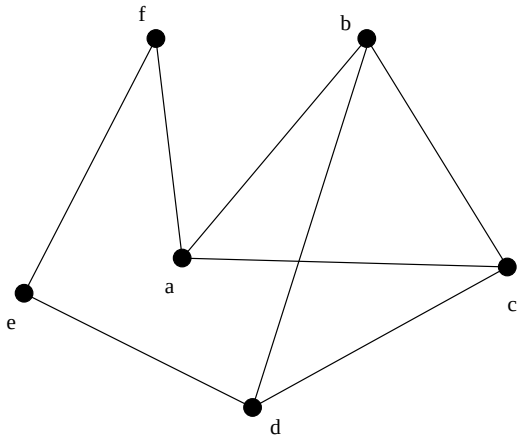
**Außer Schreibutensilien und einer handgeschriebenen DIN A4 Seite (mit abgeben!!) sind keine Hilfsmittel erlaubt!**

Auf diesem Klausurbogen ist genügend Platz, um die Lösungen der Aufgaben aufzuschreiben. Auch die Rückseiten der Blätter können verwendet werden (bitte auf der Vorderseite anmerken). Bitte geheftet lassen! **Zusätzliche lose Blätter** müssen mit der Matrikelnummer, Namen und Aufgabennummer versehen werden. Auf einem Zusatzblatt jeweils nur eine Aufgabe bearbeiten. Nicht mit Bleistift und nicht mit Rot schreiben.*Der Klausurbogen ist auf jeden Fall abzugeben!*

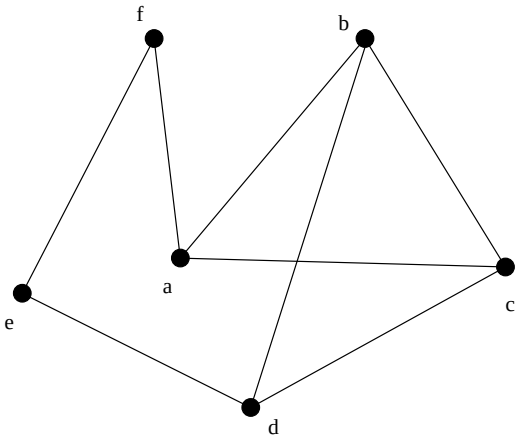
Viel Erfolg!

**Aufgabe 1** Graphen /4+2

(a) Bestimmen Sie für den unten dargestellten Graphen eine Darstellung in Adjazenzlistenform und markieren Sie bezüglich dieser von Ihnen gewählten Darstellung den Breiten- und Tiefensuchbaum startend in Knoten $a$.




DFS-Baum




BFS-Baum

Adjazenzlisten:

(b) Wann heißen zwei gerichtete Graphen ohne Mehrfachkanten isomorph? Beschreiben Sie sehr kurz eine brute-force-Methode, um zwei solche Graphen über der Knotenmenge $\{1, \ldots, n\}$ auf Isomorphie zu testen. Schätzen Sie deren Laufzeit nach oben ab!

**Lösung**: Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ heißen isomorph, wenn es eine Bijektion $\phi : V_1 \to V_2$ gibt, so dass für jedes Knotenpaar $(v, w) \in V_1 \times V_1$ gilt: $(v, w) \in E_1 \Leftrightarrow (\phi(v), \phi(w) \in E_2$
Ein brute-force-Algorithmus: Für $V_1 = V_2 = \{1, \ldots, n\}$ ist jede Bijektion eine Permutation der Elemente. Davon gibt es $n!$ viele. Das Testen, ob eine solche Permutation genau die Kanten wieder auf Kanten abbildet, geht in $O(n^2)$, wenn die Graphen durch Adjazenzmatrizen gegeben sind. Das ergibt eine Gesamtlaufzeit von $O(n! \cdot n^2)$.

**Aufgabe 2** Heaps und Suchbäume /2+2+2

(a) Es sei $T_1$ ein Heap mit $n = 2^h - 1$ Einträgen und $T_2$ ein Heap mit $2^h + 1$ Einträgen. Beschreiben Sie eine effiziente Methode, diese zwei Heaps zu einem Heap zu vereinigen und analysieren Sie die Laufzeit (sie sollte möglichst klein sein)!

**Lösung**:Nehmen wir an, die Heaps sind mit Binärbäumen realisiert. In $T_1$ sind die $h$ obersten Level mit Einträgen voll belegt, in $T_2$ gibt es neben den vollen $h$ obersten Level noch 2 Einträge auf Level $h + 1$. Wir bilden einen neuen Heap $T$: Als erstes Zugriff in $O(1)$ auf den letzten Eintrag (sei dies $x$) von $T_2$. Wir bilden einen neuen Binärbaum mit Wurzel $x$, darunter als linker Teilbaum $T_2 \setminus \{x\}$, als rechten Teilbaum nehmen wir $T_1$. Dies geht in $O(1)$ einschließlich des Verweises auf den neuen letzten Eintrag. Dieser Baum hat schon die äußere Gestalt eines Heaps. Nun muss nur noch $x$ ggf. versickert werden, um die Heapeigenschaft herzustellen. Dies kostet im worst case $O(h)$. Also Gesamtaufwand $O(h) = O(\log n)$.

(b) Sei $T$ ein Binärbaum mit Schlüsseleinträgen an den inneren Knoten, der gleichzeitig ein Min–Heap und auch ein AVL–Baum ist.
Wieviele Einträge kann er haben, wenn die Schlüssel alle verschieden sind und wieviele, wenn Schlüssel mehrfach auftreten können. Kurze Begründung!

**Lösung:** Wenn die Einträge verschieden sind, so kann es höchstens einen geben. Wären es zwei oder mehr, so müsste wegen der äußeren Gestalt eines Heaps ein Eintrag $x$ links unter der Wurzel liegen. Wegen der Suchbaumeigenschaft eines AVL–Baumes ist $x$ kleiner als der Wurzeleintrag. Widerspruch zum Min–Heap. Wenn die Einträge alle gleich sind, so gibt es keine Größenbeschränkung! Ein vollständiger balancierter Binärbaum, bei dem in allen inneren Knoten derselbe Eintrag steht, ist offensichtlich sowohl AVL–Baum wie auch Min–Heap.

(c) Sei $k$ ein Schlüssel aus einem (total geordneten) Universum und sei `nextKey(k)` eine Methode, die für ein Wörterbuch den nächstgrößeren Schlüssel zu $k$ liefert, der im Wörterbuch abgespeichert ist, falls vorhanden. Die gespeicherten Schlüssel seien alle verschieden. Diskutieren Sie, wie gut sich AVL-Bäume für die Implementierung eignen!

**Lösung:** Sei $T$ der AVL–Baum. Wir suchen zunächst nach dem Eintrag $k$. Dies geht in $O(\log |T|)$.
Fall 1: $k$ ist nicht enthalten. Sei $l$ das Blatt, das dabei als Einfügeposition gefunden wurde. Ist $l$ linkes Kind, so ist der Eintrag beim Vaterknoten der gesuchte `nextKey(k)`. Ist $l$ rechtes Kind, so suchen wir den tiefsten Vorfahren, der linkes Kind seines Vaters $v$ ist. Wenn es einen solchen gibt, so ist er Nachfolger bei der Inorder–Traversierung und hat den Eintrag `nextKey(k)`. Ansonsten gibt es `nextKey(k)` nicht im Baum. Diese Suche ist in $O(\log |T|)$ zu bewerkstelligen.
Fall 2: $k$ ist enthalten. Sei $v$ der entsprechende innere Knoten. Der Nachfolger

von $v$ in der Inordertraversierung (falls es ihn gibt) hat den Eintrag `nextKey(k)`. Angenommen das rechte Kind $w$ von $v$ hat einen Eintrag. Dann suchen wir im Teilbaum mit Wurzel $w$ den kleinsten Eintrag. Dazu gehen wir von $w$ aus nach links unten, solange es dort Einträge gibt. Das geht wieder in $O(\log |T|)$. Falls aber $w$ keinen Eintrag hat, also ein Blatt ist, so suchen wir wie im Fall 1 den tiefsten Vorfahren, der linkes Kind seines Vaters $v$ ist. Dort bei $v$ steht dann `nextKey(k)`.

**Aufgabe 3** Java /2+4

(a) Was versteht man in Java unter dem Begriff overriding von Methoden und welche Regel kommt zur Anwendung?

**Lösung**: Methoden in einer abgeleiteten Klasse können public– und protected–Methoden der Oberklasse überschreiben, falls sie den gleichen Namen, diesselbe Parameterliste und denselben Rückgabetyp haben. Beim Aufruf der Methode entscheidet der Instanztyp, welche Methode ausgeführt wird.

(b) Schreiben Sie ein Java-Programm `PrimeTwins.java`, das die Primzahlzwillinge bis zur Größe 10000 berechnet. Dies sind Primzahlpaare $(p, q)$ mit $p+2 = q$, also zum Beispiel 11 und 13. Es ist vielleicht sinnvoll (allerdings nicht notwendig), geeignete Teilaufgaben zu definieren und zu lösen. Kommentare nicht vergessen!

**Aufgabe 4** Verschiedenes /2+2+2

(a) Nennen Sie die Methoden des ADT `priority queue`, und was sind deren Laufzeiten in O–Notation bei einer Realisierung dieses ADT mittels eines Heaps bzw. mittels einer ungeordneten einfach verketteten Liste.

**Lösung**:
Die wichtigsten Methoden sind insertItem(key k, Object o) und removeMin(), plus Abfragen nach Größe bzw Leer–Sein. Bei $n$ gespeicherten Elementen:
mit einem Heap: int size() und boolean isEmpty() in $O(1)$, insertItem(key k, Object o) und removeMin() in $O(\log n)$

mit Liste: int size(), boolean isEmpty() und insertItem(key k, Object o) in $O(1)$, removeMin() in $O(n)$

(b) Definieren Sie, was für zwei Funktionen $f, g$ heißt, dass $f(n) = \Theta(g(n))$ gilt. Benutzen Sie diese Definition um zu zeigen: $n \log_8 n = \Theta(n \log_2 n^2)$

**Lösung**:
Es gilt

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists c_1, c_2 > 0 \exists n_0 \forall n > n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Wegen $n \log_8 n = (n \log_2 n)/3$ und $n \log_2 n^2 = 2n \log_2 n$ gilt für $n_0 = 0$, $c_1 = 1/6$ und $c_2 = 1$:

$$\forall n > 0 : (1/6) \cdot n \log_2 n^2 \leq n \log_8 n \leq 1 \cdot (n \log_2 n^2)$$

(c) Geben Sie die bestmögliche Obere–Schranken–Beziehung in O–Notation (also O oder $\Theta$) für die Anzahl $|E|$ der Kanten in Abhängigkeit von einer Funktion in der Anzahl $n$ der Knoten von ungerichteten Graphen aus folgenden Graphklassen (alle Graphen ohne Doppelkanten) an:
mit Lösung:
(1) Klasse der Bäume: $|E| = \Theta(n)$
(2) der Wälder: $|E| = O(n)$
(3) der zusammenhängenden bipartiten Graphen: $|E| = O(n^2)$
(4) der zusammenhängenden Graphen mit höchstens zwei Kreisen: $|E| = \Theta(n)$
.