Goethe-Universität Frankfurt am Main
Institut für Informatik
Theorie komplexer Systeme
Dr. Mariano Zelke

27. Juli 2012

Datenstrukturen (SoSe 12)
Klausur (Modulabschlussprüfung)

Name: _____ Vorname: _____

Matrikelnummer: _____ Studiengang: _____

⇓ BITTE GENAU LESEN ⇓

- Außer einem dokumentenechten Schreibstift und einem handschriftlich beschriebenem DIN-A4-Blatt sind zu dieser Klausur keine weiteren Hilfsmittel erlaubt. Bitte beachten Sie, dass die Verwendung nicht zugelassener Hilfsmittel eine Täuschung darstellt und zwangsläufig zum Nichtbestehen der Klausur führt.

  Insbesondere müssen Sie Ihre Handys vor Beginn der Klausur ausschalten.
- Bitte legen Sie Ihre Goethe-Card bzw. Ihren Studierendenausweis und einen gültigen Lichtbildausweis deutlich sichtbar an Ihren Platz, damit wir im Laufe der Klausur die Identität überprüfen können.
- Zur Bearbeitung der Aufgaben stehen Ihnen 100 Minuten zur Verfügung.
- Überprüfen Sie, ob Ihr Exemplar der Klausur alle von 2 bis 14 durchnummerierten Seiten enthält.
- Bitte schreiben Sie Ihre Lösungen direkt an die dafür vorgesehene Stelle. Gegebenenfalls können Sie auch die beigefügten Zusatzblätter benutzen. Weitere Blätter sind auf Nachfrage erhältlich.
- Begründungen sind nur dann notwendig, wenn die Aufgabenformulierung dies verlangt.
- Jedes Blatt der abgegebenen Lösung muss mit Namen, Vornamen und Matrikelnummer gekennzeichnet sein; andernfalls werden diese Blätter nicht gewertet.
- Schreiben Sie ausschließlich mit einem dokumentenechten blauen oder schwarzen Stift – alle mit einem anderen Stift angefertigten Lösungen werden nicht gewertet.
- Werden zu einer Aufgabe zwei oder mehr Lösungen angegeben, so gilt die Aufgabe als nicht gelöst. Entscheiden Sie sich also immer für **eine** Lösung.
- In der Klausur können maximal 100 Punkte erreicht werden. Die in den Übungsaufgaben im SoSe 2012 erworbenen Bonuspunkte werden zu der in der Klausur erreichten Punktzahl hinzuaddiert. Werden insgesamt mindestens 50 Punkte erreicht, so ist die Prüfung bestanden.
- Die Ergebnisse der Klausur werden spätestens am 17.08.2012 um 18:00 Uhr auf der folgenden zur Vorlesung gehörenden Internetseite bekannt gegeben:

  `http://www.tks.informatik.uni-frankfurt.de/teaching/ss12/ds`
- Die Möglichkeit zur Klausureinsicht besteht am 27.08.2012, 14-17 Uhr im Raum 117, RM 11-15.

Aufgabe	1	2	3	4	5	6	7	8	9	Klausur	Bonus	Gesamt
maximale Punkte	9	12	10	10	11	13	11	9	15	100	10	110
erreichte Punkte												

Note:

Aufgabe 1: **(9 Punkte)**

Es seien die folgenden beiden Hashfunktionen gegeben:

$$f(x) = x \bmod 11 \quad \text{und} \quad g(x) = 5 - (x \bmod 5)$$

Fügen Sie die Zahlen

32, 25, 21, 4, 14, 38

in dieser Reihenfolge in eine anfangs leere Hashtabelle der Größe 11 ein. Stellen Sie jeweils die Hashtabelle am Ende aller Einfügungen dar.

(a) Benutzen Sie Hashing mit Verkettung, wobei die Hashfunktion $f(x)$ anzuwenden ist. (3 Pkte)

0	1	2	3	4	5	6	7	8	9	10

(b) Benutzen Sie lineares Austesten mit $h_i(x) = (f(x) + i) \bmod 11$, wobei $i = 0, 1, \ldots$ (3 Pkte)

0	1	2	3	4	5	6	7	8	9	10

(c) Benutzen Sie doppeltes Hashing mit $h_i(x) = (f(x) + i \cdot g(x)) \bmod 11$, wobei $i = 0, 1, \ldots$ (3 Pkte)

0	1	2	3	4	5	6	7	8	9	10

Aufgabe 2: **(12 Punkte)**

(a) Welche der folgenden Aussagen sind wahr, welche falsch? (6 Pkte)

Kreuzen Sie alle richtigen Antworten an. Für jedes korrekte Kreuz bekommen Sie zwei Punkte, für jedes **falsche Kreuz** werden **zwei Punkte abgezogen**. Setzen Sie kein Kreuz, so wird dies mit null Punkten bewertet. Die Gesamtpunktzahl in dieser Teilaufgabe ist aber mindestens 0.

Sei $n \in \mathbb{N}$ die Länge der Eingabe für einen Algorithmus.

- Wenn Algorithmus A eine Laufzeit von $\Theta(n^2)$ hat, so hat A auch eine Laufzeit von $\mathcal{O}(n^3)$ und von $o(n^3)$. ☐ wahr ☐ falsch
- Wenn Algorithmus A eine Laufzeit von $\mathcal{O}(n)$ hat, so hat auch eine 10-fache Wiederholung von A eine Laufzeit von $\mathcal{O}(n)$. ☐ wahr ☐ falsch
- Wenn Algorithmus A eine Laufzeit von $\mathcal{O}(n^2 \cdot \log_2 n)$ hat und Algorithmus B eine Laufzeit von $\mathcal{O}(n^3)$ hat, so ist A auf jeder Eingabe schneller als B. ☐ wahr ☐ falsch

(b) Es seien die folgenden Funktionen von $\mathbb{N}_{>0}$ nach $\mathbb{R}$ gegeben: (6 Pkte)

$$
\begin{aligned}
f_1(n) &= \frac{3^n}{9} \\
f_2(n) &= 9n \cdot \log_3 n \\
f_3(n) &= 3^{\log_9 n} \\
f_4(n) &= 9^{\log_3 n} \\
f_5(n) &= \frac{9n}{3} \\
f_6(n) &= \frac{(n+1)!}{n+1}
\end{aligned}
$$

Ordnen Sie die Funktionen nach ihrem asymptotischen Wachstum beginnend mit der am langsamsten wachsenden Funktion.

Aufgabe 3: **(10 Punkte)**

Bestimmen Sie das asymptotische Wachstum der folgenden Rekursionsgleichungen als Θ-Notation in Abhängigkeit von n. Sie können davon ausgehen, dass n eine Potenz von 2 ist.

(a) $T(1) = 17, \quad T(n) = 8 \cdot T\left(\frac{n}{2}\right) + 5n^3, \quad \text{für } n > 1$ (5 Pkte)

$T(n) = \Theta($ $)$

(b) $T(1) = 1, \quad T(n) = T(n-1) + 2n, \quad \text{für } n > 1$ (5 Pkte)

$T(n) = \Theta($ $)$

Aufgabe 4: **(10 Punkte)**

Bestimmen Sie die asymptotische Laufzeit der folgenden Pseudocode-Fragmente als Θ-Notation in Abhängigkeit von n. Sie können davon ausgehen, dass n eine Potenz von 2 ist.

(a) (5 Pkte)

```
int factor=0;
for(i=1; i <= n/8; i++) {
   int x=i;
   factor=factor*x;
   for(j=1; j <= n/4; j++)
      for(k=1; k < 97; k++)
         factor=factor/2;
}
```

Die Laufzeit in Abhängigkeit von n ist: $\Theta($ $)$

(b) (5 Pkte)

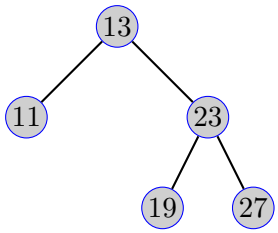
```
for(i=0; i < 2n; i++) {
   int j=n;
   while (j >= 1)
      j=j/2;
}
```

Die Laufzeit in Abhängigkeit von n ist: $\Theta($ $)$

Aufgabe 5: **(11 Punkte)**

Es sei der folgende AVL-Baum T_1 gegeben:

T_1 :



(a) Geben Sie die Summe der Balance-Grade aller Knoten in T_1 an. (1 Pkt)

Summe der Balance-Grade:

(b) Sei T_0 ein *leerer* AVL-Baum, also ohne gespeicherte Schlüssel, und sei S die Menge der in T_1 gespeicherten Schlüssel, also $S = \{11, 13, 19, 23, 27\}$. T_1 kann aus T_0 erzeugt werden, indem in T_0 nacheinander die Schlüssel aus S eingefügt werden. (3 Pkte)

Geben Sie eine Reihenfolge an, in der die Schlüssel aus S in T_0 eingefügt werden können, so dass T_1 entsteht und *ohne* dass dabei eine AVL-Baum-Rotation stattfindet.

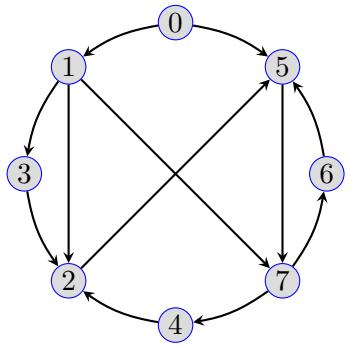
Reihenfolge von Schlüsseleinfügungen, um aus T_0 *ohne* Rotationen T_1 zu erzeugen:

(c) Fügen Sie in T_1 einen Knoten mit dem Schlüssel 17 ein. Stellen Sie alle erforderlichen AVL-Baum-Rotationen dar. (7 Pkte)

Aufgabe 6: **(13 Punkte)**

(a) Es sei der folgende gerichtete Graph $G = (V, E)$ gegeben: (10 Pkte)

G :



Geben Sie für G den Wald der Tiefensuche an, der entsteht, wenn Tiefensuche in Knoten 0 startet und die unbesuchten Nachbarn jedes Knotens in aufsteigender Reihenfolge besucht werden. Geben Sie an, welche Kanten jeweils als Vorwärts-, Quer- und Rückwärtskanten klassifiziert werden.

Wald der Tiefensuche:

Vorwärtskanten:

Querkanten:

Rückwärtskanten:

(b) Geben Sie einen *ungerichteten, zusammenhängenden* Graphen G' mit der Knotenmenge $\{0, 1, 2, 3, 4\}$ an, so dass Folgendes gilt: (3 Pkte)

Tiefensuche auf G' markiert die Knoten von G' in der *gleichen Reihenfolge* als besucht wie Breitensuche auf G'.

Gehen Sie dabei wie üblich davon aus, dass Tiefensuche und Breitensuche auf Knoten 0 starten und die unbesuchten Nachbarn eines Knotens in aufsteigender Reihenfolge abarbeiten.

Es reicht aus, wenn Sie G' in graphischer Darstellung angeben.

G' mit Knotenmenge $\{0, 1, 2, 3, 4\}$ in graphischer Darstellung:

Aufgabe 7: **(11 Punkte)**

(a) Das Array H_1 sei als Heap mit den folgenden Einträgen gegeben:

0	11	10	3	6	9	1	2	5	4	8	7

(i) Geben Sie den binären Baum T_{H_1} an, der durch H_1 repräsentiert wird. (2 Pkte)

repräsentierter binärer Baum T_{H_1}:

(ii) Auf dem durch T_{H_1} repräsentierten Heap wird nun `delete_max()` ausgeführt. Stellen Sie die Situation in T_{H_1} dar, nachdem `delete_max()` und alle notwendigen Reparaturen beendet wurden. (5 Pkte)

T_{H_1} nach `delete_max()` und allen notwendigen Reparaturen:

(b) Der folgende Pseudocode beschreibt den schon aus der Vorlesung bekannten Algorithmus Heapsort, der ein gegebenes Array A mit n Einträgen $A[1], A[2], \ldots, A[n]$ aufsteigend sortiert. Dazu wird ein Heap h benutzt, der n Prioritäten aufnehmen kann und zu Beginn leer ist. (4 Pkte)

```
Heapsort( Array A )
for i = 1, 2, ..., n do
    h.insert(A[i]);
for i = n, n − 1, ..., 1 do
    A[i] = h.delete_max();
```

Welche asymptotische Gesamtlaufzeit ergibt sich für Heapsort, *in dem speziellen Fall*, dass alle Einträge des Arrays A gleich groß sind, d. h. wenn gilt $A[1] = A[2] = \cdots = A[n]$? Begründen Sie Ihre Antwort.

Gesamtlaufzeit Heapsort auf Array der Länge n mit identischen Einträgen:

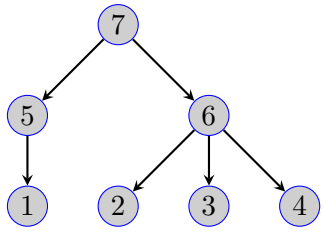
$\Theta($ $)$

Begründung:

Aufgabe 8: **(9 Punkte)**

(a) Es sei der folgende *geordnete* Baum B_1 gegeben:

B_1 :



(i) Geben Sie die Reihenfolge an, in der ein Postorder-Durchlauf die Knoten von B_1 besucht. (2 Pkte)

Reihenfolge Postorder-Durchlauf:

(ii) Geben Sie die Darstellung von B_1 als Vater-Array an. Benutzen Sie dabei die Konvention, dass für die Wurzel w eines Baumes gelten soll `vater[`w`]=`w. (2 Pkte)

Darstellung Vater-Array:

(b) Geben Sie zu folgender Problemstellung einen Algorithmus in Pseudocode an, der das Problem löst. Ihr Algorithmus soll die vorgegebene worst-case Laufzeit erreichen. (5 Pkte)

Eingabe: Array A mit Einträgen $A[1], A[2], \ldots, A[n]$; die Einträge von A entsprechen der Vater-Darstellung eines Baums mit n Knoten. Dabei gilt $A[i] = i$ genau dann, wenn i die Wurzel des Baumes ist.

Ausgabe: Alle Kinder der Wurzel

Laufzeit: $\mathcal{O}(n)$

Algorithmus in Pseudocode:

Aufgabe 9: **(15 Punkte)**

Ziel dieser Aufgabe ist der Entwurf einer Datenstruktur, die einen *ungerichteten* Graphen $G = (V, E)$ repräsentiert. Die Knoten des Graphen seien von 1 bis n durchnummeriert, es gilt also $V = \{1, 2, \ldots, n\}$, und zu Beginn enthalte der Graph keine Kanten, d. h. anfangs gelte $E = \emptyset$. Es sollen folgende Operationen mit den angegebenen Laufzeiten unterstützt werden:

- `add(`i`,`j`)`: Fügt eine ungerichtete Kante zwischen den Knoten i und j in G ein. (Sie können davon ausgehen, dass keine Kante mehr als einmal eingefügt wird.)
  Laufzeit: $\mathcal{O}(\log_2 n)$
- `lookup(`i`,`j`)`: Gibt aus, ob zwischen i und j eine Kante in G existiert.
  Laufzeit: $\mathcal{O}(\log_2 n)$
- `printNeighbors(`i`)`: Gibt alle Nachbarn von i in G in *aufsteigend sortierter* Reihenfolge aus.
  Laufzeit: $\mathcal{O}(\text{grad}(i))$, wobei $\text{grad}(i)$ die Anzahl der Nachbarn von i in G ist.

Beschreiben Sie kurz, wie Ihre Datenstruktur aufgebaut ist, wie die drei Operationen darauf arbeiten und wie die jeweils verlangte Laufzeit erreicht wird. Dabei ist *kein* Code gefordert.

Analysieren Sie auch kurz den Speicherbedarf Ihrer Datenstruktur.