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

1. Oktober 2012

Datenstrukturen (WS 12/13)
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 19.10.2012 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 23.10.2012, 11-13 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	8	14	11	11	15	100	10	110
erreichte Punkte												

Note:

Aufgabe 1: **(9 Punkte)**

Es seien die folgenden beiden Hashfunktionen gegeben:

$$f(x) = x \bmod 7 \quad \text{und} \quad g(x) = 5 - (x \bmod 5)$$

Fügen Sie die Zahlen

35, 48, 7, 32, 20, 28

in dieser Reihenfolge in eine anfangs leere Hashtabelle der Größe 7 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

(b) Benutzen Sie lineares Austesten mit $h_i(x) = (f(x) + i) \bmod 7$, wobei $i = 0, 1, \ldots$ (3 Pkte)

0	1	2	3	4	5	6

(c) Benutzen Sie doppeltes Hashing mit $h_i(x) = (f(x) + i \cdot g(x)) \bmod 7$, wobei $i = 0, 1, \ldots$ (3 Pkte)

0	1	2	3	4	5	6

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 $\mathcal{O}(n)$ und gleichzeitig eine Laufzeit von $\Omega(n)$ hat, so hat A auch eine Laufzeit von $\Theta(n)$. ☐ wahr ☐ falsch
- Wenn Algorithmus A eine Laufzeit von $\Theta(1)$ hat, so hat A auf allen Eingaben die gleiche Laufzeit. ☐ 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 muss es mindestens eine Eingabe geben, auf der A schneller ist als B. ☐ wahr ☐ falsch

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

$$
\begin{aligned}
f_1(n) &= \frac{2n}{3} \\
f_2(n) &= 3^{(\log_3 27) \cdot n/3} \\
f_3(n) &= 2^{(\log_2 4) \cdot n/2} \\
f_4(n) &= (\log_3 n)^2 \\
f_5(n) &= \log_2(n^3) \\
f_6(n) &= \sqrt[3]{n^2}
\end{aligned}
$$

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 4 und $n \geq 4$ ist.

(a) $T(0) = 2, \quad T(n) = T(n-4) + 8, \quad$ für $n \geq 1$ (5 Pkte)

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

(b) $T(1) = 2468, \quad T(n) = 2 \cdot T\left(\frac{n}{4}\right) + 6 \cdot \log_2 n + 8, \quad$ 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)

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

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

(b) (5 Pkte)

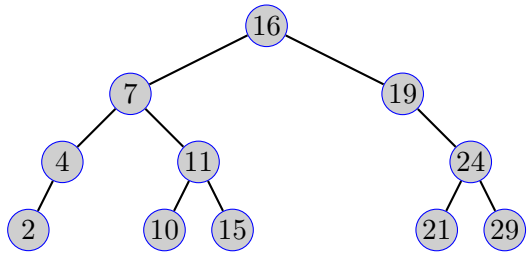
```
int result=n;
for(i=1; i <= n*n; i++) {
  int value=i;
  for(j=1; j <= i; j++)
    result=result/2;
}
for(k=1; k <= n*n*n; k++)
  result=result*4;
```

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

Aufgabe 5: **(8 Punkte)**

(a) Es sei der folgende binäre Suchbaum B_1 gegeben:

B_1 :



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

Reihenfolge Preorder-Durchlauf:

(ii) Aus B_1 wird durch `remove(7)` der Knoten mit dem Schlüssel 7 entfernt. Stellen Sie die Situation in B_1 nach Beendigung dieser Operation dar. (4 Pkte)

B_1 nach Beendigung von `remove(7)`:

(b) Sei B_0 ein leerer binärer Suchbaum (und *kein* AVL-Baum). Durch die `insert`-Operation werden in B_0 die Schlüssel (2 Pkte)

$$0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots, -n, n$$

in dieser Reihenfolge eingefügt, wobei $n \in \mathbb{N}_{>0}$. Welche asymptotische Laufzeit in Abhängigkeit von n ergibt sich insgesamt für diese $2n + 1$ Operationen?

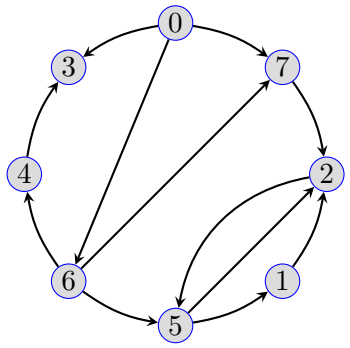
Gesamtlaufzeit der $2n + 1$ `insert`-Operationen mit obiger Reihenfolge:

$\Theta($ $)$

Aufgabe 6: **(14 Punkte)**

(a) Es sei der folgende gerichtete Graph $G_1 = (V, E)$ gegeben: (10 Pkte)

G_1 :



Geben Sie für G_1 den Wald der Tiefensuche an, der entsteht, wenn Tiefensuche in Knoten 0 startet und die unbesuchten Nachfolger jedes Knotens in aufsteigender Reihenfolge besucht werden. Geben Sie an, welche Kanten von G_1 jeweils als Vorwärts-, Quer- und Rückwärtskanten klassifiziert werden.

Wald der Tiefensuche:

Vorwärtskanten:

Querkanten:

Rückwärtskanten:

(b) Der folgende Pseudocode beschreibt das schon aus der Vorlesung bekannte Verfahren Tiefensuche, das einen gerichteten Graphen $G = (V, E)$ durchsucht. Es wird ein Array `besucht` benutzt, das für jeden der Knoten $0, 1, \ldots, |V| - 1$ des Graphen G einen Eintrag besitzt. (4 Pkte)

Tiefensuche()

```
1 for k = 0, 1, ..., |V| − 1 do
2     besucht[k] = false;
3 for k = 0, 1, ..., |V| − 1 do
4     if besucht[k] = false then
5         tsuche(k);
```

tsuche(Knoten v)

```
1 besucht[v] = true;
2 for jeden Nachfolger p von v in aufsteigender Reihenfolge do
3     if besucht[p] = false then
4         tsuche(p);
```

Welche asymptotische Gesamtlaufzeit ergibt sich für Tiefensuche, wenn sie auf einem Graphen aufgerufen wird, der in Form einer *Adjazenzmatrix* gegeben ist? Begründen Sie Ihre Antwort.

Gesamtlaufzeit Tiefensuche auf Graph $G = (V, E)$ gegeben als Adjazenzmatrix:

$\Theta($ $)$

Begründung:

Aufgabe 7: **(11 Punkte)**

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

0	37	31	17	23	29	13	3	19	2	5	7	11

(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) Geben Sie zu folgender Problemstellung einen Algorithmus in Pseudocode an, der das Problem löst. Ihr Algorithmus soll die vorgegebene worst-case Laufzeit erreichen, die Verwendung von Sortieralgorithmen ist nicht erlaubt. (4 Pkte)

Eingabe: Array H mit Einträgen $H[0], H[1], H[2], \ldots, H[n]$; H ist ein Heap für einen Baum T_H mit n Knoten, die paarweise verschiedene Prioritäten haben und es gilt $n \geq 7$.

Ausgabe: Die zweitgrößte Priorität in T_H

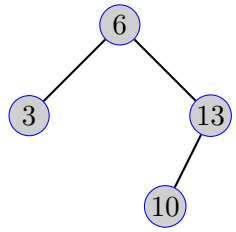
Laufzeit: $\mathcal{O}(1)$

Algorithmus in Pseudocode:

Aufgabe 8: **(11 Punkte)**

Es sei der folgende AVL-Baum T_1 gegeben:

T_1 :



(a) Geben Sie den Knoten/die Knoten mit dem größten Balance-Grad in T_1 an. (1 Pkt)

Knoten mit dem größten Balance-Grad in T_1:

(b) Sei $\mathbb{Z}$ das Universum aller möglichen Schlüssel. Geben Sie alle Schlüssel aus $\mathbb{Z}$ an, deren alleinige Einfügung (d. h. ohne das Einfügen weiterer Schlüssel davor oder danach) in T_1 jeweils zu mindestens einer Rotation führt. (3 Pkte)

Schlüssel aus $\mathbb{Z}$, deren alleinige Einfügung in T_1 jeweils zu einer Rotation führt:

(c) Fügen Sie in T_1 einen Knoten mit dem Schlüssel 12 ein. Stellen Sie alle erforderlichen AVL-Baum-Rotationen dar. (7 Pkte)

Aufgabe 9: **(15 Punkte)**

Ziel dieser Aufgabe ist der Entwurf einer Datenstruktur, die einen *gerichteten* Graphen $G = (V, E)$ repräsentiert. Die Knoten des Graphen seien von 0 bis $n - 1$ durchnummeriert, es gelte also $V = \{0, 1, \ldots, n-1\}$, 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 gerichtete Kante von Knoten i zu Knoten 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 in G eine Kante von Knoten i zu Knoten j existiert.
  Laufzeit: $\mathcal{O}(1)$
- `löscheKanteZuKleinstemNachfolger(`i`)`: Löscht in G die Kante von Knoten i zum kleinsten Nachfolger von i. (Sie können bei Aufruf dieser Operation davon ausgehen, dass i mindestens einen Nachfolger in G besitzt.)
  Laufzeit: $\mathcal{O}(\log_2 n)$

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.