Prof. G. Zachmann
Dipl.-Inf. D. Mohr

TU Clausthal
Institut für Informatik

Klausur zu „Informatik II“

Sommersemester 2011

IV/X/MMXI, $14^{00} - -15^{00}$

Name: ..... **Matrikelnummer:** .....

Vorname: .....

Studiengang: .....

Semesterzahl: .....

E-Mail: .....

Bitte in Druckschrift ausfüllen.

Hinweise: (GENAU DURCHLESEN!)

- Neben Papier und Schreibutensilien sind keine weiteren Hilfsmittel erlaubt. Verwenden Sie keine roten Stifte und keine Bleistifte.
- Vergessen Sie nicht, Ihren Namen und die Matrikelnummer auf *jedes* Blatt zu schreiben. Blätter ohne diese Angaben werden nicht gewertet.
- Schreiben Sie Ihre Lösungen auf die Aufgabenblätter möglichst in die dafür vorgesehenen Felder. Sie können auch die Rückseiten verwenden. Weiteres Schreibpapier kann von den Betreuern angefordert werden. Benutzen Sie kein mitgebrachtes Papier.
- Bei Multiple-Choice-Fragen gibt es für jedes richtig gesetzte Kreuz einen Punkt. Für jedes falsch gesetzte Kreuz wird ein halber Punkt abgezogen! Nicht beantwortete Fragen werden nicht gewertet. Die Gesamtpunktzahl beträgt mindestens 0.
- Bitte schreiben Sie in Ihrem eigenen Interesse deutlich. Für nicht lesbare Lösungen können wir keine Punkte vergeben.
- Klausurblätter dürfen nicht voneinander getrennt werden.
- Werden mehrere unterschiedliche Lösungen für eine Aufgabe abgegeben, so wird die Aufgabe nicht gewertet.
- Im Fall von Täuschungsversuchen wird die Klausur sofort mit 0 Punkten bewertet. Eine Vorwarnung erfolgt *nicht*.

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	$\sum$
max. Punkte	6	4	8	14	9	8	9	1	59
erreicht									

Aufgabe		**6 Punkte**
	Name:	(6 x 1)
1	Matrikel-Nr.:	Σ: .....

Datenstruktur: Hash-Tabellen

Für welche der folgenden Probleme / Aufgaben können Hash-Tabellen zu einem Verfahren mit einem effizienten durchschnittlichen Aufwand beitragen? (Hier soll "*effizient*" eine Laufzeit bedeuten, die in $O(n \log n)$ oder kleiner ist.)

	Problem / Aufgabe	**Ja**	**Nein**
a)	Anzahl verschiedener Werte bestimmen: gegeben eine Menge von n (Key,Value)-Paaren; wieviele verschiedene Values kommen darin vor?	☑	☐
b)	Dynamisches Dictionary, d.h., eine Datenstruktur, die die Funktionen Einfügen, Löschen und Suchen (bzgl. eines Keys) unterstützt	☑	☐
c)	Bereichsuche: gegeben seien zwei Werte a und b; gesucht werden sollen alle Keys, die zwischen a und b (einschließlich) liegen	☐	☑
d)	Symbol table lookup: gegeben sei ein Variablenname (der z.B. in einem arithmetischen Ausdruck in einem Programm steht); finde dessen Typ und Wert	☑	☐
e)	Schnittmenge: gegeben seien zwei Mengen von Keys, A und B; gesucht ist $A \cap B$ (wieder eine Menge von Keys)	☑	☐
f)	Sortieren (bzgl. der Keys)	☐	☑

Aufgabe		**4 Punkte**
2	Name:	(4)
	Matrikel-Nr.:	Σ: .....

Sortieralgorithmen

a) Welcher der folgenden Algorithmen hat im schlechtesten Fall eine Laufzeit von $O(n^2)$, aber im durchschnittlichen Fall eine Laufzeit von $\Theta(n \log n)$?

☐ Bubblesort ☐ Mergesort ☐ Heapsort ☑ Quicksort

b) Welcher der folgenden Sortieralgorithmen arbeitet *in situ* (*in place*)?

☐ Mergesort ☑ Heapsort ☐ LSB-Radix-Sort

c) Welche der folgenden Aussagen zu Radix-Sort ist richtig?

☐ Alle zu sortierenden Zahlen müssen in eine 32-Bit-Darstellung passen.
☑ Man kann damit auch Zahlen in IEEE-754-Darstellung (für Floats) sortieren.
☑ Man kann damit auch Strings sortieren, vorausgesetzt, alle Strings haben eine einheitliche Länge.

Aufgabe		8 Punkte
3	Name:	(6+2)
	Matrikel-Nr.:	Σ: .....

Komplexität von Berechnungsproblemen, Rekursionsgleichungen

Geben Sie die worst-case Komplexität der folgenden Berechnungsprobleme in Groß-$\mathcal{O}$-Notation und in Worten an (z.B. "$O(2^n)$ = exponentielle Laufzeit"). Gehen Sie dabei davon aus, daß ein effizienter Algorithmus für die Aufgabenstellung verwendet wird. Benutzen Sie n, um die Größe der Eingabe zu bezeichnen.

a) Auffinden eines Elements mit einem bestimmten Wert in einem sortierten Array.

Lösung:
$O(log\, n)$ = Logarithmische Zeit (Binäre Suche)

b) Ausgeben des mittleren Elements eines Arrays bekannter Größe.

Lösung:
$O(1)$ = Konstante Zeit

c) Ausgeben des mittleren Elements einer einfach verketteten Liste bekannter Größe.

Lösung:
$O(n)$ = Lineare Zeit

d) Sortieren eines Arrays mit Quicksort.

Lösung:
$O(n^2)$ = Quadratische Zeit. (Worst-Case Szenario!)

e) Sortieren einer einfach verketteten Liste.

Lösung:
$O(n\, log\, n)$ = Log-linear time. (letzteres ist nicht notwendig, diesen Begriff hatten wir so nicht)
Mergesort: beim ersten Pass wird die Liste in $O(n)$ geteilt = gleiche Laufzeit wie beim Array-basierten Mergesort; Zusammenfügen geht in $O(1)$.

$T(n)$ sei definiert durch $T(1) = 7$ und $T(n+1) = 3n + T(n)$ für alle $n \in \mathbb{N} \setminus \{0\}$.
Geben Sie das Wachstum von $T(n)$ in asymptotischer Θ-Schreibweise an.

Aufgabe		14 Punkte
	Name:	(10 + 4)
4	Matrikel-Nr.:	Σ: .....

Komplexitätsanalyse, Schleifeninvariante

Wir betrachten folgenden Programmcode:

```
x = 2
i = 1
while x < n:
  x = x * x
  i += 1
```

Der Wert von n (ein Integer) sei vorgegeben und $n \geq 2$.

a) Geben Sie die Schleifeninvariante an, die am Beginn des Rumpfes der `while`-Schleife gilt.

Lösung:
Erste Werte von $x = 2^1$ $(i = 1)$, 2^2 $(i = 2)$, 2^4 $(i = 3)$, 2^8 $(i = 4)$, 2^{16} $(i = 5), \ldots$
Schleifeninvariante: $x = 2^{2^{i-1}}$

b) Welchen Wert hat i nach der Ausführung? (Falls Sie die Rundungszeichen in die falsche Richtung angeben, gibt es *keinen* Punktabzug.)

Lösung:
Also: $i = 1 + \lceil \log_2(\log_2(n)) \rceil$

c) Welche Laufzeit hat der Algorithmus in Groß-$\mathcal{O}$-Notation?

Lösung:
$O(\log \log n)$

Aufgabe		**9 Punkte**
	Name:	(9 x 1)
5	Matrikel-Nr.:	Σ: .....

Quickies

Kreuzen Sie für jede Aussage an, ob sie wahr oder falsch ist.

	Aussage	**Richtig**	**Falsch**
a)	Der Aufwand zur Suche nach einem Element in einer doppelt verketteten, sortierten Liste beträgt im worst-case $O(\log n)$.	☐	☑
b)	Ein Stack arbeitet nach dem *Last-in-First-Out*-Prinzip (LIFO).	☑	☐
c)	Ein Algorithmus A habe die Laufzeit $\Theta(n^2)$, und B die Laufzeit $\Theta(n \log n)$. Dann kann es gewisse (kleine) Werte für n geben, wo A schneller als B ist.	☑	☐
d)	Die Algorithmentechnik *Divide-and-Conquer* eignet sich gut, um Optimierungsprobleme zu lösen	☐	☑
e)	Backtracking kann man oftmals einsetzen, wenn der Algorithmus eine zulässige Belegung für eine Menge von Variablen bestimmen muss, zwischen denen Nebenbedingungen eingehalten werden müssen.	☑	☐
f)	In B-Bäumen können die Blätter verschiedene Tiefe haben.	☐	☑
g)	AVL-Bäume sind eine spezielle Variante von binären Suchbäumen.	☑	☐
h)	Van-Emde-Boas-Trees ermöglichen, den Nachfolger zu einem gegebenen Key k (also das kleinste x im VEB größer k) in Zeit $O(\log \log n)$ zu finden, wobei n in der selben Größenordnung wie die Größe des Universums sei.	☑	☐

Aufgabe		8 Punkte
6	Name:	(2 + 2 + 4)
	Matrikel-Nr.:	Σ: .....

Datenstrukturen: Unterschiede zwischen Min-Heap und binärem Suchbaum

a) Wo findet man jeweils das *kleinste* Element?

Lösung:
Min-Heap: An der Wurzel
Suchbaum: Am äußeren linken Blatt

b) Welche Ordnungsrelation besteht zwischen einem inneren Knoten und seinen Kindern?

Lösung:
Min-Heap: node $<$ left, und node $<$ right
Suchbaum: left $<$ node $<$ right

c) Geben Sie jeweils einen Algorithmus an, um das *größte* Element in einem gegebenen Min-Heap (`H`) bzw. in einem binärem Suchbaum (`S`) zu finden. Dabei sei `H` die Wurzel des Min-Heaps, `H.key` dessen Schlüssel, und `H.left` und `H.right` ein Zeiger auf dessen linkes bzw. rechtes Kind. Analog seien `S.key`, `S.left` und `S.right` definiert. Sie dürfen von der Verfügbarkeit einer Funktion $\max(x, y)$ ausgehen. Sie dürfen Ihre Lösung in Pseudo-Code oder in Python oder einem Mix angeben.

Für Min-Heap:

```
def get_max( H ):
```

Lösung:

```
if H.left == None and H.right == None:
  return H.key
if H.left == None:
  return get_max(H.right)
if H.right == None:
  return get_max(H.left)
return max( get_max(H.left), get_max(H.right) )
```

Für Binären Suchbaum:

```
def get_max( S ):
```

Lösung:

```
while S.right != None:
  S = S.right
return S.key
```

Aufgabe		9 Punkte
7	Name:	(3 + 6)
	Matrikel-Nr.:	Σ: .....

Algorithmentechnik: Backtracking

Sie sollen in dieser Aufgabe einen Algorithmus entwickeln, der die Lösung für ein sogenanntes "magisches Quadrat" berechnet. Dieses besteht aus 3×3 Feldern, die mit den Ziffern von 1 bis 9 so belegt werden, daß:

- jede Ziffer nur einmal vorkommt,
- die Summen aller Zeilen, aller Spalten und der beiden Diagonalen den selben Wert ergeben,
- alle Felder belegt sind.

Sie sollen nun einen Algorithmus beschreiben, der **alle** Lösungen durch *Backtracking* findet. Zweckmäßigerweise definiert man dazu eine *Teillösung* als ein teilweise von oben links belegtes Quadrat, wie z.B.:

4	9	2
3	5	

Eine Teillösung definieren wir als *nützlich*, wenn:

- jede Ziffer in den belegten Feldern höchstens einmal vorkommt,
- die Summen aller vollständig belegten Zeilen, Spalten und Diagonalen den selben Wert ergeben.

a) Welche (nützliche) Teillösung ist die Wurzel des Suchbaums? Welches sind die Kinder einer Teillösung im Suchbaum? Wann ist eine Teillösung eine Gesamtlösung?

Lösung:
Wurzel = leeres Quadrat.
Nachfolger = nächstes leeres Feld belegt, wobei nur Zahlen dort vorkommen, die noch nicht "verbraucht" sind (kann man leicht mit einem Bit-Array machen), und die Summen-Bedingung erhalten bleibt.
Gesamtlösung = Blätter des Suchbaums, da nur nützliche Teillösungen vorkommen.

b) Geben Sie einen Algorithmus an, der die Aufgabe mit Backtracking löst. Das magische Quadrat soll dabei in dem Array Q[3][3] gespeichert sein. Sie dürfen Ihren Algorithmus umgangssprachlich, in Pseudo-Code, in Python, oder einem Mix beschreiben. Einzige Bedingung ist, daß wir daraus die Vollständigkeit und Korrektheit ersehen können müssen. Tip: schreiben Sie auf der Rückseite Ihren Algorithmus im Konzept vor. Sie dürfen folgende Funktion als gegeben verwenden:

```
def sums_ok( Q, field ):
# Checkt die Summen-Bedingung fuer eine Teilloesung in Q
# field (0 ... 8) = Anzahl besetzte Felder - 1 = Nr. des rechten unteren Feldes
```

Lösung:

```
# Q = 3x3 int-Array, mit 0 vorinitialisiert
# unused = bool-Array der Länge 10, vorinitialisiert mit true
# unused[i] = true <-> Zahl i kommt noch nicht in Q vor

def mag_quadrat( Q, next_field )
# next_field = 0, ..., 9 = naechstes zu besetzendes Feld;
# if next_field==9 -> finish

   if next_field == 9:
      Q ausgeben
      return

   i = next_field div 3
   j = next_field mod 3
```

```
   for n = 1 ... 9:                    # alle Zahlen durchprobieren
      if unused[n]:                    # n ist noch nicht "verbraucht"
         Q[i][j] = n
         if sums_ok( Q, next_field ):
            unused[n] = false               # Zahl "sperren"
            mag_quadrat( Q, next_field+1 )
            unused[n] = true                # Zahl wieder freigeben
        # end if
    #end for
#end def

# Evtl. diese Funktion als gegeben voraussetzen
def sums_ok( Q, field ):
# field (0 ... 8) = Anzahl besetzte Felder - 1 = Nr des rechten unteren Feldes
   if field >= 6:           # letzte Zeile ist (teilweise) besetzt
      for j = 0 ... field-6:
         Summe der Spalte j berechnen
         Summe der ersten Spalte speichern, die anderen damit vergleichen
         bei Mismatch: return false
   # end if
   if field mod 3 == 2:       # letzte Spalte ist (teilweise) besetzt
      for i = 0 ... (field mod 3):
         Summe der Zeile i berechnen, mit bisherigen Summe vergleichen
         bei Mismatch: return false
   # end if
   if field >= 6:
      Summe der Diagonalen RO nach LU überprüfen
   if field == 8:
      Summe der Diagonalen LO nach RU überprüfen
   # alles gut
   return true
#end def

# Start
mag_quadrat( Q, 0 )
```

Aufgabe		**1 Punkt**
	Name:	(1)
8	Matrikel-Nr.:	Σ: .....

Informatik II

Welche der hier abgebildeten Personen ist der Dozent der Vorlesung “Informatik II” im Sommersemester 2011?


☐


☐


☐


☐


☑


☐


☐