Fakultät IV
Elektrotechnik/Informatik

Einführung in die Informatik - Vertiefung Probeklausur

Sommersemester 2015

Hinweis: Diese Probeklausur enthält eine kleine Sammlung an Aufgaben, deren Schwierigkeitsgrad etwa dem der schriftlichen Prüfung des Moduls Einführung in die Informatik Vertiefung entspricht. Die Aufgaben decken nicht alle behandelten Themenbereiche ab und der Umfang der Probeklausur entspricht nicht dem der echten Prüfung.

Aufgabe 1 Allgemeine Fragen.

1. **Teilaufgabe:** Womit kann man einen Booleschen Ausdruck im Allgemeinen nicht in seine minimale Form umwandeln?

   ☐ Ablesen aus Wahrheitstabellen.
   ☐ Verfahren von Quine und McCluskey.
   ☐ KV-Diagramme.
   ☐ Anwendung Boolescher Axiome.

2. **Teilaufgabe:** Welche der folgenden Komplexitätsklassen ist so groß, dass die anderen drei angegebenen Klassen darin enthalten sind?

   ☐ $O(n^2)$
   ☐ $O(1^n)$
   ☐ $O(n \log(n))$
   ☐ $O(n)$

3. **Teilaufgabe:** Wonach werden Objekte von allgemeinen Datentypen üblicherweise sortiert?

   ☐ Objekte allgemeiner Datentypen können nicht sortiert werden.
   ☐ Nach der Reihenfolge des Operators <=, welcher auch für allgemeine Datentypen definiert ist.
   ☐ Nach einem zu definierenden Schlüssel.
   ☐ Nach dem ersten Ganzzahl- oder Fließkomma-Attribut.

4. **Teilaufgabe:** Was ist eine generische Klasse?

   ☐ Eine Klasse, von der nicht geerbt werden kann.
   ☐ Eine Klasse, die einen Typen als Parameter besitzt, der zur Laufzeit verändert werden kann.
   ☐ Eine Klasse, die einen Typen als Parameter besitzt, der bei der Instanziierung dieser Klasse festgelegt wird.
   ☐ Ein abstrakter Datentyp.

5. **Teilaufgabe:** Welche Interface(s) braucht man, um eine nicht-abstrakte iterierbare Klasse zu implementieren?

   ☐ `Iterator<E>`, `Iterable<T>`
   ☐ `Comparable<T>`, `Iterator<E>`
   ☐ `Iterable<T>`
   ☐ `Iterator<E>`

6. **Teilaufgabe:** Worin unterscheidet sich eine doppelt verkettete Liste im Vergleich zur einfach verketteten Liste?
   n
   - ☐ Jeder Knoten zeigt auf seine zwei nachfolgenden Knoten.
   - ☐ Jeder Knoten speichert zusätzlich zu seinem eigenen Wert auch den Wert des Vorgängers.
   - ☐ Sie unterscheidet sich lediglich in einer Tail-Referenz, die auf das letzte Element der Liste zeigt.
   - ☐ Jeder Knoten zeigt auf seinen Vorgänger und Nachfolger.

7. **Teilaufgabe:** Wo steht in einem Max-Heap das größte Element?
   - ☐ Im linken äußersten Blatt des Baumes.
   - ☐ Im rechten äußersten Blatt des Baumes.
   - ☐ Das ist in einem Heap nicht genau definiert.
   - ☐ In der Wurzel des Baumes.

8. **Teilaufgabe:** Wenn x der linke Nachfolger von y in einem binären Suchbaum ist, dann gilt:
   - ☐ key(x) $\leq$ key(y)
   - ☐ key(x) $<$ key(y)
   - ☐ key(x) $\geq$ key(y)
   - ☐ key(x) $>$ key(y)

9. **Teilaufgabe:** Welche der folgenden Datenstrukturen ist linear?
   - ☐ Graph.
   - ☐ Queue.
   - ☐ AVL-Baum.
   - ☐ Heap.

10. **Teilaufgabe:** Was gilt für AVL-Bäume?
    - ☐ Es werden immer Rotationen beim Einfügen oder Löschen von Elementen benötigt.
    - ☐ Sie sind linksvoll.
    - ☐ Sie verhindern eine Degeneration zu einer Liste.
    - ☐ Sie verhalten sich wie Listen.

11. **Teilaufgabe:** Wie viele Einsen werden für den Ausdruck $\overline{x} \cdot y \cdot z$ in eine KV-Tafel mit 5 Eingangsvariablen eingetragen?
    - ☐ 8
    - ☐ 1
    - ☐ 4
    - ☐ 2

12. **Teilaufgabe:** Welche Aussage zu Heapsort ist richtig?
    - ☐ Heapsort hat im Average-Case die Komplexität O(n).
    - ☐ Bei Heapsort werden Sift-Down und Sift-Up benötigt.
    - ☐ Heapsort führt abwechselnd nacheinander Swaps und Heapifys aus.
    - ☐ Heapsort kann manche Arrays nicht sortieren, da diese als Heap vorliegen müssen.

13. **Teilaufgabe:** Wozu dient die Methode iterator() des Interfaces Iterable?

    ☐ Sie ruft die for-each Schleife auf und sorgt somit dafür, dass die Iteration einmal vollständig durchgeführt wird.

    ☐ Sie erzeugt ein neues Iteratorobjekt und gibt dessen Referenz zurück.

    ☐ Sie gibt true zurück, falls ein Iterator noch Elemente enthält.

    ☐ Sie überprüft, welche der implementierten Iteratorklassen verwendet werden soll und gibt einen entsprechenden String zurück.

14. **Teilaufgabe:** Welche Interface(s) braucht man, um eine nicht-abstrakte iterierbare Klasse zu implementieren?

    ☐ Iterable<T>

    ☐ Iterator<E>

    ☐ Comparable<T>, Iterator<E>

    ☐ Iterator<E>, Iterable<T>.

15. **Teilaufgabe:** Welche Wege findet der Dijkstra-Algorithmus ?

    ☐ die kürzesten Wege

    ☐ die schönsten Wege

    ☐ die längsten Wege

    ☐ einen optimalen Weg über alle Knoten im Graph

Aufgabe 2 Boolesche Algebra.

1. **Teilaufgabe:** Sind die folgenden Booleschen Ausdrücke äquivalent?

   $$\begin{aligned} f(x,y) &= (x \Rightarrow y) + (y \Rightarrow x) \\ g(x,y) &= (x + y) \Rightarrow (x \cdot y). \end{aligned}$$

   Beweisen oder widerlegen Sie die Behauptung mit der Wahrheitstafelmethode. Die Zwischenschritte müssen erkennbar sein.

   Hinweis: Es gilt $x \Rightarrow y := \overline{x} + y$.

2. **Teilaufgabe:** Wandeln Sie den Booleschen Ausdruck der Funktion:

   $$f(x,y,z) = \overline{(x \equiv y)} + z$$

   mit Hilfe der algebraischen Umformung in eine ausgezeichnete konjunktive Normalform um. Die Zwischenschritte müssen erkennbar sein.

   Hinweis: Es gilt $x \equiv y := (x \cdot y) + (\overline{x} \cdot \overline{y})$.

3. **Teilaufgabe:** Wandeln Sie den Booleschen Ausdruck der Funktion:

   $$f(x,y,z,w) = (((x \cdot y) + x + (z + w)) \cdot (w + z)$$

   mit Hilfe einer KV-Tafel in eine ausgezeichnete disjunktive Normalform um. Die Zwischenschritte müssen erkennbar sein.

Aufgabe 3 Komplexität.

- Bestimmen Sie eine Formel fär den Aufwand $T_g(n)$ der folgenden Methode $g(n)$. Dabei soll für die Berechnung des Zeitaufwands nur in Zeile 7 der Funktionsaufruf `fun(i)` berücksichtigt werden. Die Funktion `fun(i)` weise hierbei einen Aufwand von $T_{\text{fun}}(i) = 1$ für alle i auf.

```
1  public void g(int n) {
2    int i = 0;
3    int j = 0;
4    while (i < n) {
5      j = i;
6      while (j == i ) {
7        fun(i);
8        j++;
9      }
10     i++;
11   }
12 }
```

- Nennen Sie die Komplexitätsklasse, in welcher sich $T_g(n)$ befindet und markieren Sie in der folgenden Tabelle die Komplexitätsklassen zu denen $T_g(n)$ gehört.
  Hinweis: Ein Beweis ist nicht gefordert.

Ordnung	**fällt in diese Klasse**
$\mathcal{O}(1)$	
$\mathcal{O}(\log(n))$	
$\mathcal{O}(n)$	
$\mathcal{O}(n \log(n))$	
$\mathcal{O}(n^2)$	
$\mathcal{O}(n^2 \log(n))$	
$\mathcal{O}(n^3)$	
$\mathcal{O}(n^p)$ mit $p > 3$	
$\mathcal{O}(p^n)$	

Aufgabe 4 Heapsort.

1. **Teilaufgabe:** Gegeben sei die Zahlenfolge

   $$F_1 = 47, 21, 35, 34, 59, 22, 36, 58, 60, 23$$

   Geben sie einen Binärbaum an, der die Elemente der Folge F_1 enthält und die (Max-)Heap Eigenschaft erfüllt. Geben Sie zusätzlich dazu den zum Heap gehörigen Array an.

2. **Teilaufgabe:** Gegeben sei die Zahlenfolge

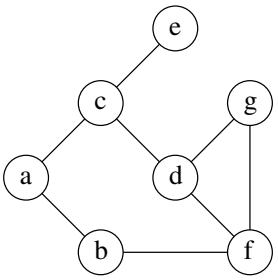
   $$F_2 = 10, 9, 6, 8, 7, 2, 5, 1, 4, 3.$$

   Sortieren Sie die Folge F_2 mit Heapsort. Stellen Sie nach jeder Iteration den Restheap als Baum und die gesamte Zahlenfolge als Array dar.

   Hinweis: Die Zahlenfolge F_2 ist ein Heap.

Aufgabe 5 Tiefensuche.

Betrachten Sie den folgenden Graphen:



1. **Teilaufgabe:** Welchen abstrakten Datentyp verwendet die Tiefensuche? Wie lautet das Speicherprinzip dieses Datentyps?

2. **Teilaufgabe:** Traversieren Sie den Graphen G mit Tiefensuche. Führen Sie dazu eine Handsimulation mit Hilfe der untenstehenden Tabelle durch. Dabei bezeichne *Schritt* die Nummer des aktuellen Schleifendurchlaufs und AK den aktuellen Knoten. Beachten Sie bei der Handsimulation Folgendes:

   - Startknoten ist der Knoten mit Bezeichner a, welcher sich nach Initialisierung (Schritt 0) im Stack befindet.
   - Geben Sie für $Schritt > 0$ den Inhalt des Stacks jeweils am Ende des aktuellen Schleifendurchlaufs an.
   - Fügen Sie pro Schleifendurchlauf jeweils alle weißen Nachfolger von AK stets in *alphabetisch aufsteigender* Reihenfolge in den Stack ein.
   - Die schwarze Liste enthält alle Knoten, die schon abgearbeitet worden sind. Fügen Sie einen Knoten in dem selben Schleifendurchlauf in die Schwarze Liste ein, in welchem alle seine Nachfolger-Knoten in den Stack eingefügt wurden.

Schritt	AK	Stack	schwarze Liste
0	-	a	-

t	AK	Stack

Aufgabe 6 Mergesort.

Implementieren Sie eine Java-Methode `public void mergesort(int[] arr)` die das Sortierverfahren Mergesort für ganze Zahlen realisiert und gegebenenfalls benötigte Hilfsmethoden.

Aufgabe 7 Generische Klassen.

1. **Teilaufgabe:** Es seien $\mathcal{X}$ und $\mathcal{Y}$ zwei Mengen beliebigen Typs. Implementieren Sie ein generisches Interface mit dem Namen `Function`, das eine Funktion $f : \mathcal{X} \to \mathcal{Y}$ repräsentiert. Dabei sollen die Mengen $\mathcal{X}$ und $\mathcal{Y}$ generisch verschieden Datentypen angehören können. Ausserdem fordert das Interface eine Methode `apply`, die bei Eingabe eines Elementes $x \in \mathcal{X}$ den Funktionswert $y = f(x) \in \mathcal{Y}$ zurück liefert.

   Hinweis: mit der Schreibweise `<T1,T2, ..., Tn>` kann man ein generischer Datentyp mit n Typvariablen `T1, T2, ..., Tn` definiert werden.

2. **Teilaufgabe:** Implementieren Sie eine Klasse `Length`, die eine Funktion f : `String` $\to$ `Integer` repräsentiert. Die Klasse implementiert das Interface `Function`. Die Methode mit dem Name `apply` erhält als Eingabe ein Objekt vom Typ String und liefert die Länge des Strings als Objekt der Klasse `Integer` zurück.

   Hinweis: Für ein Objekt `str` der Klasse `String` liefert die Methode `int length()` die Länge (Anzahl der Zeichen) von `str` zurück.

3. **Teilaufgabe:** Implementieren Sie eine Klasse `Test`. Diese Klasse besitzt eine `main()`-Methode, die folgenden Ablauf realisiert:

   - Es wird ein Objekt der Klasse `Length` erzeugt.
   - Das Objekt ruft die Methode `apply()` für die Eingabe des Strings `"Inftech"` auf.
   - Der resultierende Funktionswert wird auf der Konsole ausgegeben.

Aufgabe 8 Listen.

Betrachten Sie das folgende unvollständige Java-Programm für die doppelt verkettete Liste:

```
1  public class DoppeltVerketteteListe<T> {
2
3      private class ListElem {
4
5          T data;
6
7          ListElem(T data) {
8              this.data = data;
9          }
10     }
11
12 }
```

1. **Teilaufgabe:**

   Ergänzen Sie die Klasse `DoppeltVerketteteListe` um Referenzen auf das erste und letzte Element der Liste (head und tail). Ergänzen Sie weiter die innere Klasse ListElem um die benötigten Referenzen auf Vorgänger- und Nachfolge-Elemente.

2. **Teilaufgabe:**

   Implementieren Sie eine Methode `public void get(int i)`, die das Datenobjekt `data` des Listelements an der i-ten Stelle zurückgibt.

   Hinweise:
   1. Es sei n die Anzahl der Listenelemente. Der Kopf der Liste befindet sich an der 0-ten Stelle. Das letzte Listenelement befindet sich an der (n-1)-ten Stelle.

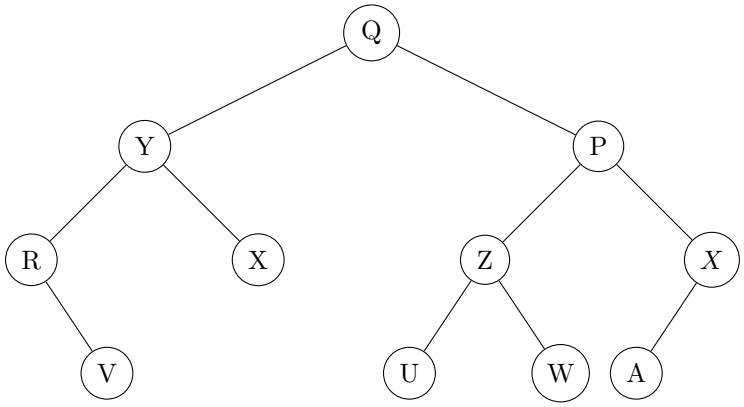
   2. Gehen Sie davon aus, dass stets $0 \leq i < n$ gilt.

3. **Teilaufgabe:**

   Implementieren Sie eine Methode `public void addFirst(T data)`, die an der ersten Stelle der Liste das übergebene Datenobjekt einfügt.

Aufgabe 9 Traversierung von Bäumen.

1. **Teilaufgabe:** Geben Sie die entstehende Buchstabenfolge aus, wenn Sie den folgenden Binärbaum in postorder-Reihenfolge traversieren.



2. **Teilaufgabe:** Implementieren Sie eine Java-Klasse `Tree` mit einer Unterklasse `Node` für einen Baum und für Baumknoten an. Dabei sollen folgende Bedingungen erfüllt sein:

   - Die in den Knoten abgespeicherten Nutzdaten sind Elemente eines generischen Datentyps.
   - Die Klasse `Node` ist außerhalb der Klasse `Tree` unsichtbar.
   - Jeder Knoten des Baums hat beliebig viele Nachfolger.

   Es sollen nur die Attribute (und keine Methoden) der Klassen angegeben werden.

3. **Teilaufgabe:** Implementieren Sie für die Klasse `Tree` aus Aufgabenteil 2 eine *rekursive* Methode `int countNodes()`, welche die Anzahl der Knoten dieses Baumes zurückgibt.

```
int countNodes() {
```

Aufgabe 10 AVL-Bäume.

Gegeben sei eine Klasse `AVLBaum`, die die spezifischen Unterklassen für innere Knoten (Fork) und Blätter (Leaf) sowie eine Referenz auf das Wurzelelement enthält.

```
1  public class AVLBaum<T>{
2
3      private abstract class Node{
4          public int key;
5          public int hoehe;
6
7          Node(int key){
8              this.key = key;
9              hoehe = 0;
10         }
11
12         public abstract boolean checkAVLCondition();
13
14     }
15
16     private class Fork extends Node{
17         Node links;
18         Node rechts;
19
20         Fork(int key, Node links, Node rechts){
21             super(key);
22             this.links = links;
23             this.rechts = rechts;
24             hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
25         }
26
27         public void setRechts(Node rechts){
28             this.rechts = rechts;
29             hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
30         }
31
32         public void setLinks(Node links){
33             this.links = links;
34             hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
35         }
36
37     }
38
39     private class Leaf extends Node{
40         T daten;
41
42         Leaf(int schluessel, T daten){
43             super(schluessel);
44             this.daten = daten;
45         }
46
47     }
48
49     // Wurzel des AVL-Baums
50     private Node root;
```

```
51
52 }
```

1. **Teilaufgabe:** Erweitern Sie die Klasse Fork um eine Methode `public Fork rotateLeft()`, die eine Linksrotation am aufrufenden Knoten durchführt.

2. **Teilaufgabe:** Erweitern Sie die Klasse AVLTree um eine Methode `public boolean checkAVLCondition()`, die die AVL-Eigenschaft des gesamten Baumes testet.
   Hinweis: Für Ihre Implementierung müssen Sie die Methode `public boolean checkAVLCondition()` innerhalb der Fork- und Leaf-Klasse implementieren.