Name, Vorname, Matrikelnummer Studiengang

Klausur
Algorithmen und Datenstrukturen
SS 2018 – 16.08.2018

Hinweise:

- Die Bearbeitungszeit beträgt 120 Minuten.
- Zum Bestehen der Klausur sind 50 Punkte erforderlich.
- Schreiben Sie auf die ersten beiden Blätter Ihren Namen, Matrikelnummer und Studiengang.
- Erlaubte Hilfsmittel: keine
- Lösen Sie nicht die Klammerung der Klausur!
- Tragen Sie Ihre Lösungsvorschläge in die Klausurvorlage (evtl. auf den Rückseiten) ein. Weiteres Papier können Sie bei der Klausuraufsicht anfordern.
- Alle vorgegebenen und zu erstellenden Programmtexte beziehen sich auf die Programmiersprache Java.
- Bitte schreiben Sie deutlich.

Viel Erfolg !

Aufgabe	**Maximalpunkte**	**Erreichte Punkte**
1 (Multiple Choice, Wissen)	13	
2 (Graphen)	18	
3 (Listen)	18	
4 (Komplexität)	13	
5 (Graphen und Bäume)	20	
6 (Sortieren)	16	
7 (B-Bäume)	12	
Summe	110	

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 1 (Multiple Choice, Wissen) *13 Punkte*

a) Kreuzen Sie in den folgenden Teilaufgaben an, welche Aussagen richtig und welche falsch sind. Für jede korrekt markierte Aussage erhalten Sie 0,5 Punkte, für jede falsch markierte werden 0,5 Punkte abgezogen. Pro Teilaufgabe erhalten Sie mindestens 0 Punkte.

Bewerten Sie folgende Aussagen über Sortieralgorithmen	**Richtig**	**Falsch**
Die schnellsten in der Vorlesung behandelten Sortierverfahren benötigen im Mittel Zeitaufwand O(n) zur Sortierung von n Schlüsseln		
Quicksort benötigt im Mittel O(n log n) Vergleiche um n Schlüssel zu sortieren		
Die in der Vorlesung behandelten Algorithmen kopieren im schlechtesten Fall (worst-case) jeweils mindestens O(n log n) der n Schlüssel		
Merge-Sort benötigt bei der günstigsten Eingabe maximal O(n) Vergleiche zum Sortieren von n Schlüsseln		

Welche der folgenden Aussagen sind korrekt?	**Richtig**	**Falsch**
$n \log_3 n \in O(n \log_2 n)$		
$n^2 \in O(2^{n \log n})$		
$n \log_3 n^n \in O(n^3)$		
$15\ n + 2^n \in O(2^n)$		

Welche Aussagen über Suchverfahren sind korrekt?	**Richtig**	**Falsch**
Für die binäre Suche benötigen wir numerische Schlüssel, also Schlüssel die Zahlen sind		
In Hash-Tabellen werden im schlimmsten Fall (worst-case) O(log n) Schritte benötigt, um in n Schlüsseln einen Schlüssel zu finden		
In Hash-Tabellen werden im schlimmsten Fall (worst-case) O(n) Schritte benötigt, um in n Schlüsseln einen Schlüssel zu finden		
Das Suchen in einem AVL-Baum mit n Knoten benötigt im schlimmsten Fall O(log n) Schritte		

Welche der folgenden Aussagen sind korrekt?	**Richtig**	**Falsch**
Falls P=NP gilt, so ist jedes Problem berechenbar.		
Wird ein binärer Suchbaum in Pre-Order durchlaufen, so erhält man die Knoten in sortierter Reihenfolge.		
Das Traversieren eines Baumes liegt in der Klasse NP.		
Das Topologische Sortieren liegt in der Klasse P.		

b) Was versteht man unter „quadratischem Sondieren“?

c) Erläutern Sie das Problem beim Löschen eines Elements aus einer Hash-Tabelle, wenn quadratisches Sondieren verwendet wird.

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 2 (Graphen) *18 Punkte*

Folgender Graph ist durch seine Adjazenzmatrix gegeben:

		Nach				
		A	**B**	**C**	**D**	**E**
Von	**A**		2	5		4
	B			2		1
	C	1			3	
	D		1			
	E				7	

a) Ist der Graph zusammenhängend (Begründung)?

b) Zeichnen Sie den gegebenen Graphen.

c) Führen Sie auf dem obigen Graphen den Algorithmus von Dijkstra zur Bestimmung minimaler Wege mit dem Startknoten A aus. Geben Sie nach jeder Runde die Menge der markierten Knoten, sowie die bis dahin errechneten Abstände und den jeweiligen Vorgänger in Tabellenform an.

d) Bestimmen Sie auf Basis der Tabelle aus Aufgabenteil c) die kürzesten Wege von Knoten A zu den Knoten D und E. Erläutern Sie kurz, wie Sie dabei vorgehen.

Aufgabe 3 (Listen) *18 Punkte*

Betrachten Sie die folgende Implementierung einer Liste von Personen:

```
public class Person
{
   public String name;
   public String vorname;
   public int alter;

   public Person(String name, String vorname, int alter)
   {
      this.name = name;
      this.vorname = vorname;
      this.alter = alter;
   }
}

public class Link
{
   public Person daten;
   public Link naechster;

   public Link(Person daten, Link naechster)
   {
      this.daten = daten;
      this.naechster = naechster;
   }
}

public class Liste
{
   private Link anfang;

   public void fuegeEin(Person p)
   {
      anfang = new Link(p, anfang);
   }

   public Person loesche()
   {
      // Aufgabenteil a)
   }

   public int getAnzahl()
   {
      // Aufgabenteil b)
   }
}
```

a) Ergänzen Sie die Methode `loesche()`, die das erste Element aus der Liste löscht und zurückgibt.

```
public Person loesche()
{









}
```

b) Ergänzen Sie die Methode `getAnzahl()`, die die Anzahl der in der Liste vorhandenen Elemente zurückgibt.

```
public int getAnzahl()
{









}
```

c) Ergänzen Sie die folgende Klasse `ListenSortierer` um die Methode `sortiere`, die die Elemente einer übergebene Liste nach dem Alter sortiert. Die übergebene Liste soll also so umgeändert werden, dass die Elemente nach Aufruf dieser Methode sortiert sind. Die Sortierreihenfolge (aufsteigend oder absteigend) spielt hierbei keine Rolle. **Es soll kein eigener Sortieralgorithmus implementiert werden und nicht auf die Link-Elemente der Liste zugegriffen werden!**

Hinweis: *Speichern Sie die Elemente der Liste mit Hilfe der `loesche` –Methode des Listen-Objekts in einer geeigneten Datenstruktur der Java-Collections oder einem Feld, sortieren Sie dann die Elemente mit der entsprechenden `sort`-Methode, und fügen Sie anschließend die Elemente mittels der `fuegeEin` - Methode wieder in die Liste ein. Sie können die Personen-Klasse entsprechend ergänzen.*

```
public class ListenSortierer
{
   public static void sortiere(Liste liste)
   {




   }
}
```

```
public class Person
{
   public String name;
   public String vorname;
   public int alter

   public Person(String name, String vorname, int alter)
   {
      this.name = name;
      this.vorname = vorname;
      this.alter = alter;
   }

}
```

Aufgabe 4 (Komplexität) *13 Punkte*

In einem Feld seien n Strings jeweils mit der Länge n gespeichert. Es sollen nun n Strings in diesem Feld gesucht werden, wobei der Vergleich zweier Strings der Länge n O(n) Schritte benötigt. Uns interessiert nur, ob der jeweils gesuchte String enthalten ist. Um das Problem zu lösen kann man z.B. einen der folgenden Algorithmen nutzen:

i) Lineares Durchsuchen des Feldes für jeden der n zu suchenden Strings.

ii) Sortieren des Feldes mit Selectionsort und anschließende binäre Suche für jeden der n zu suchenden Strings.

iii) Anlegen einer Hash-Tabelle, in der zunächst alle Strings eingetragen werden. Überläufer werden in konstanter Zeit separat verkettet. Anschließend werden die n zu suchenden Strings jeweils in der Hash-Tabelle gesucht. Die Berechnung eines Hash-Wertes benötigt jeweils O(n) Schritte.

iv) Die Strings werden zunächst in einen AVL-Baum eingetragen und dann jeder der n zu suchenden Strings in dem AVL-Baum gesucht.

Hinweis: Beachten Sie, dass der Vergleich jeweils O(n) Schritte, nicht wie in der Vorlesung angenommen, O(1) Schritte, benötigt.

a) Bestimmen Sie zu den Möglichkeiten i) – iv) die asymptotischen Laufzeitkomplexitäten (worst case):

b) Welche dieser Lösungen würden Sie auf Grund Ihrer Ergebnisse in Teil a) wählen?

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 5 (Graphen und Bäume) *20 Punkte*

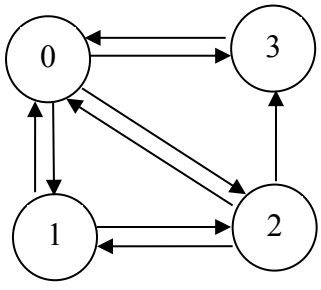
a) Ein gerichteter Graph sei in Adjazenzlistendarstellung gegeben. Wie kann man unter der Annahme, dass der Graph ein Baum ist, die Wurzel des Baums finden? Beschreiben Sie informell einen Algorithmus für dieses Problem.

b) Setzen Sie Ihren Algorithmus aus Aufgabenteil a) in eine Java-Methode um. Die Adjazenzliste des Graphen ist hierbei als zweidimensionales Feld gegeben, in der die i-te Zeile die Nachfolger des Knotens i enthält. Insbesondere können die verschiedenen Zeilen unterschiedliche Länge besitzen.

Beispiel:

1	2	3
0	2	
0	1	3
0		

Feld



zugehöriger Graph

Sie können hierbei in den Aufgabenteilen b) und c) voraussetzen, dass es sich um einen Baum handelt (das Beispiel kann also nicht vorkommen).

Die folgende Methode soll die Wurzel des Baumes bestimmen:

```
public static int bestimmeWurzel(int[][] adjazenzListe)
{

}
```

c) Die folgende Methode soll die Höhe des Baumes bestimmen, der wie in Aufgabenteil b) beschrieben durch das zweidimensionale Feld `adjazenzListe` gegeben ist:

```
public static int bestimmeHoehe(int[][] adjazenzListe)
{
   int wurzel = bestimmeWurzel(adjazenzListe);
   return tiefe(adjazenzListe, wurzel);
}
```

Ergänzen Sie nun die notwendige Methode

```
tiefe(int[][] adjazenzListe, int i).
```

Hinweis: Nutzen Sie Tiefensuche ohne Markierung der Knoten.

```
public static int tiefe(int[][] adjazenzListe, int i)
{

}
```

Name, Vorname, Matrikelnummer Studiengang

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 6 (Sortieren) *16 Punkte*

a) In der folgenden Tabelle ist die Implementierungen eines Sortieralgorithmus angegeben. Ergänzen Sie die Tabelle um den Namen des Verfahrens, asymptotische Anzahl an Vergleichen im besten (best case) und im schlechtesten Fall (worst case), sowie die Sortierreihenfolge (aufsteigend/absteigend).

Implementierung	**Name**	**Anz. Vergleiche** *O-Notation*		**Reihen-folge**
		Best case	**Worst Case**	

`public static int[] sortiere(int[] feld)`
`{`
`  for (int i=0; i<feld.length-1; i++)`
`  {`
`  int sup=i;`
`  for (int j=i+1; j<feld.length; j++)`
`  { if (feld[j]>feld[sup]) sup=j;`
`  }`
`  int tmp=feld[i];`
`  feld[i]=feld[sup];`
`  feld[sup]=tmp;`
`  }`
`  return feld;`
`}`				

b) Erläutern Sie den Unterschied zwischen Bottom-Up und normalem Heapsort.

c) Sortieren Sie die folgenden Werte mit Quicksort aufsteigend. Schreiben Sie nach jeder Runde (Teilen des Feldes) die neue Reihenfolge auf. Markieren Sie das jeweilige Pivotelement durch Einkreisen. Das Pivotelement soll jeweils als mittleres Element des Teilfeldes gewählt werden. Gibt es zwei mittlere Elemente, so soll das linke der beiden Elemente gewählt werden. Die Tabelle enthält mehr Zeilen als für eine vollständige Lösung erforderlich sind.

4	8	5	6	3	8	7

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 7 (B-Bäume) *12 Punkte*

a) Zeichnen Sie den B-Baum der Ordnung 2, der entsteht, wenn man in einen anfangs leeren Baum nacheinander die Schlüssel `5`, `10`, `15` einfügt.

b) Zeichnen Sie den B-Baum, nachdem zusätzlich die Schlüssel `23`, `2`, `24`, `34`, `11`, `25` und `21` (in dieser Reihenfolge) eingefügt wurden.

c) Zeichnen Sie nun den B-Baum, nachdem die Schlüssel `23` und `11` gelöscht wurden.

d) Zeichnen Sie nun den B-Baum, nachdem der Schlüssel `2` gelöscht wurde.