HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Prof. Dr. W. Rülling
SS 2020

Klausur

Algorithmen und Datenstrukturen

(AIN2)

Name	Vorname	Matrikelnummer	Semester	Tisch

Nr	Thema	erreichbare Punkte	erreichte Punkte
1	`merge`-Funktion in Java	15	
2	Karatsuba-Multiplikation	15	
3	Balancierte Bäume	11	
4	Obere und untere Schranken	15	
5	Flussproblem	10	
6	Hashing mit offener Adressierung	10	
7	Allgemeines	13	
8	Dijkstra-Algorithmus	10	
9	Java-Programmierung	8	
	Summe	107	

1. Überprüfen Sie die Aufgabenblätter anhand obiger Liste auf Vollständigkeit.
2. Tragen Sie bitte oben auf diesem Deckblatt Ihren Namen, Ihr Semester und die Matrikelnummer ein. Nach der Klausur geben Sie bitte das Deckblatt zusammen mit Ihren Lösungsblättern ab.
3. Versehen Sie jedes abgegebene Lösungsblatt oben links mit Ihrem Namen.

Aufgabe 1: `merge`-Funktion in Java

a) Ein wesentlicher Bestandteil des *Merge-Sort*-Algorithmus, ist eine `merge`-Funktion, die zwei sortierte Listen durch eine neue sortierte Ergebnisliste ersetzt.
Implementieren Sie dazu in Java die folgende Methode

```
LinkedList<String> merge(LinkedList<String> left,
                         LinkedList<String> right)
```

Dabei sollen die beiden Argumentlisten `left` und `right` auch unterschiedlich lang sein dürfen und bei Ihrer Implementierung sollten Sie aus Effizienzgründen keine Indexzugriffe verwenden, sondern Methoden wie:
`getFirst`, `addFirst`, `removeFirst`, `getLast`, `addLast`, `removeLast`

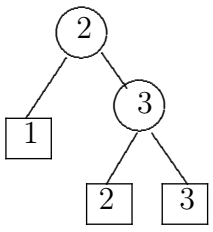
b) Erklären Sie kurz, warum Indexzugriffe bei einer `LinkedList` aufwändiger sind. als bei einer `ArrayList`.

Aufgabe 2: Karatsuba-Multiplikation

a) Berechnen Sie nach der Karatsuba-Methode das Produkt der beiden Dezimalzahlen $a = 1034$ und $b = 2103$, indem Sie die Multiplikation der 4-stelligen Dezimalzahlen auf 3 Multiplikationen von 2-stelligen Dezimalzahlen zurückführen. Geben Sie alle dabei entstehenden Zwischenergebnisse an.
(Die kleineren Multiplikationen müssen nicht weiter zerlegt werden.)

b) Um n-stellige Binärzahlen nach der Karatsuba-Methode zu multiplizieren, werden sie rekursiv auf Produkte immer kleinerer Binärzahlen zurückgeführt, bis z.B. nur noch "elementare Multiplikationen" von 4-Bit langen Zahlen benötigt werden.
Geben Sie an, wieviele solcher elementaren Multiplikationen für eine Karatsuba-Multiplikation von 128-stelligen Binärzahlen benötigt werden.

Aufgabe 3: Balancierte Bäume



Fügen Sie in den gegebenen Baum von beschränkter Balance $\alpha = \frac{1}{3}$ nacheinander die Werte 7, 6, 5 und 4 in dieser Reihenfolge ein und führen Sie dabei jeweils eine Rebalancierung durch, wenn der Baum nicht mehr von beschränkter Balance $\frac{1}{3}$ ist.
Skizzieren Sie dabei nach jedem Einfügen den entstandenen Baum und kennzeichnen Sie die jeweils aus der Balance $\alpha = \frac{1}{3}$ geratenen Knoten.

Aufgabe 4: Obere und untere Schranken

a) Geben Sie für die folgenden Funktionen jeweils möglichst gute größenordnungsmäßige obere Schranken (O-Notation) und untere Schranken (Ω-Notation) an.

$$
\begin{aligned}
f(n) &= (3n+1) \cdot 2\log_2 n^3 \\
g(n) &= 4n^2 + \log n^4 \\
h(n) &= \begin{cases} 3n^2+4 & \text{für ungerade } n > 10 \\ 3n+74 & \text{für gerade } n > 10 \\ \frac{n^3}{10}+1 & \text{für } n \leq 10 \end{cases} \\
k(n) &= 3n + 2\sqrt{n}
\end{aligned}
$$

b) Bestimmen Sie mit dem gegebenen rekursiven Ansatz die größenordnungsmäßige Laufzeit $T_z(n)$ (obere Schranke) einer Prozedur z.

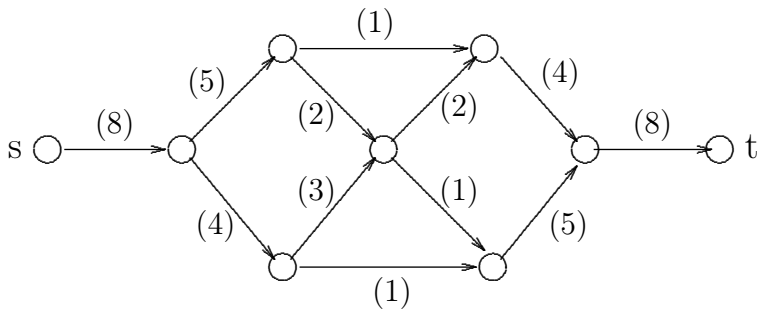
$$
T_z(n) = \begin{cases} 10 & \text{für } n = 1 \\ 4 \cdot T_z(\frac{n}{2}) + 3 & \text{für } n > 1 \end{cases}
$$

c) Bestimmen Sie für die folgenden beiden Java-Funktionen jeweils die größenordnungsmäßigen Laufzeiten $T_f(n)$ bzw. $T_g(n)$ in O-Notation.

```
int f(int n) {
   int y=0;
   while (n>1) {
      if (n%2) == 1)
        ++y;
      n/=2;
   }
   return y;
}
```

```
int g(int n) {
   int y=0;
   for (int i=1; i*i<n; ++i)
        y=i;
   return y;
}
```

Aufgabe 5: Flussproblem



a) Skizzieren Sie für das angegebene Flussproblem einen maximalen Fluss von s nach t. (Es ist egal, wie Sie diesen Fluss finden)
b) Konstruieren Sie mit dem Fluss aus a) den Hilfsgraphen mit dem mögliche Flussverbesserungen konstruiert werden können.
c) Markieren Sie im Hilfsgraphen alle von s aus erreichbaren Knoten.
d) Tragen Sie in Ihre Skizze aus a) den minimalen Schnitt für Flüsse von s nach t ein.

Aufgabe 6: Hashing mit offener Adressierung

Zur Speicherung von Zahlen in einer Hashtabelle der Größe 11 soll folgende Hashfunktion verwendet werden.

$$h_i(x) = (x \bmod 11 + i \cdot (1 + x \bmod 5)) \bmod 11$$

a) Tragen Sie die folgenden Werte in der angegebenen Reihenfolge in die anfangs leere Hashtabelle ein und geben Sie für jeden Wert an, wieviele Hashadressen für den Eintrag ausprobiert werden müssen.

10 20 30 40 15 17 3 8

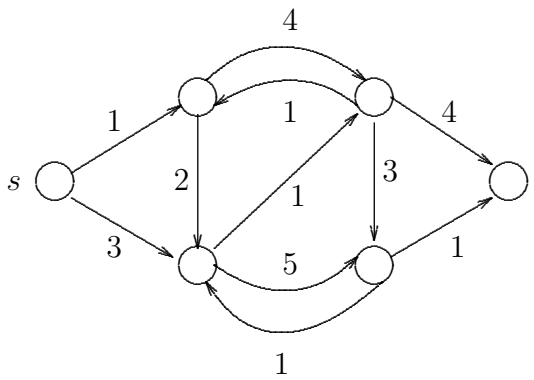
b) Geben Sie **für dieses Zahlenbeispiel** an, wieviele Zugriffe man im Durchschnitt für die **erfolgreiche Suche** nach eingetragenen Werten benötigt.

c) Geben Sie an, wieviele Zugriffe man für die erfolglose Suche nach dem Wert 7 benötigt.

Aufgabe 7: Allgemeines

a) Geben Sie die wesentliche Idee der *Divide-and-Conquer*-Technik an.
b) Welche größenordnungsmäßige Laufzeit braucht man zum Sortieren von n Strings der Länge n?
c) Was versteht man unter einer statischen Methode?
d) Welche Vorteile hat die Verwendung von typisierten Klassen?
e) Wozu benutzt man in Java das Schlüsselwort `throws`?
f) Warum darf man in einer `.java`-Datei nicht mehrere öffentliche Klassen definieren?
g) Welche Art von Daten wird in Java in `Referenzvariablen` gespeichert?
h) Wozu verwendet man in Java das Schlüsselwort `super`?

Aufgabe 8: Dijkstra-Algorithmus



a) Führen Sie für den angegebenen Graphen beim Knoten A beginnend den Dijkstra-Algorithmus aus und tragen Sie sämtliche während der Abarbeitung entstehenden Knotenbeschriftungen in eine Skizze ein.

b) Skizzieren Sie den berechneten Kürzeste-Wege-Baum. Ist dieser Baum eindeutig bestimmt? Begründen Sie Ihre Antwort.

Aufgabe 9: Java-Programmierung
Programmieren Sie eine statische Java-Methode
`public static int gemeinsam(List<String> l1, List<String> l2)`
die ermittelt, wieviele Einträge die beiden Listen `l1` und `l2` gemeinsam haben.

Z.B. soll für `l1` = `("rot", "gelb", "gelb", "blau", "gruen")`
und `l2` = `("blau", "gelb", "rot", "rot", "lila", "gelb")`
der Wert 3 geliefert werden, weil nur die 3 Strings `"rot"`, `"gelb"` und `"blau"` in beiden Listen vorkommen.