



Klausur Informatik II

6. August 2008

Vorname:

Name:

Matrikelnummer:

Zur Klausur sind keine Hilfsmittel zugelassen. Die Bearbeitungszeit beträgt 60 Minuten. Bitte tragen Sie Ihren Namen und Ihre Matrikelnummer auf dieser und allen folgenden Seiten ein.

Schreiben Sie Ihre Lösungen in den jeweils für die entsprechende Aufgabe vorgesehenen Kasten. Die Größe des Kastens steht nicht zwangsweise in Zusammenhang mit dem Umfang der für volle Punktzahl erforderlichen Antwort. Sollte der Platz im Kasten nicht ausreichen, können Sie die Rückseite des jeweiligen Blattes verwenden. Vermerken Sie im Kasten, wo der Rest Ihrer Antwort zu finden ist.

Für alle Programmieraufgaben gilt, dass Sie Variablenamen, Kommentare, etc. wahlweise in deutscher oder in englischer Sprache angeben dürfen. Ansonsten gelten die aus der Vorlesung bekannten Programmierrichtlinien.

Die Klausur ist komplett und geheftet abzugeben. Sie dürfen die Rückseiten der Aufgabenblätter als Konzeptpapier benutzen. Verwenden Sie kein eigenes Papier. Sollten Sie zusätzliches Papier benötigen, so fordern Sie dieses bei der Klausuraufsicht an.

Die unten stehende Tabelle sowie das Summenfeld unten auf jeder Seite werden von uns bei der Korrektur ausgefüllt.

Aufgabe	1	2	3	4	5	6	7	Summe
Maximal	7	7	10	10	9	9	8	60
Erreicht								

Note:

1 Verständnis- und Wissensfragen (7 Punkte)

a) Kreuzen Sie jeweils an, ob die Aussage wahr oder falsch ist. (4 Punkte)

Hinweis: Korrekte Antworten werden mit 0,5 Punkten, falsche Antworten mit einem Abzug von 0,5 Punkten und nicht beantwortete Fragen mit 0 Punkten bewertet. Sollte daraus in der Summe ein negatives Ergebnis für die Teilaufgabe a) resultieren, wird die Teilaufgabe mit 0 Punkten bewertet.

Aussage	wahr	falsch
Es gibt 3 verschiedene AVL-Bäume, die die Menge $\{3,5,7\}$ speichern.		X
Das Interface <code>java.util.Iterator</code> deklariert eine Methode <code>next()</code> .	X	
Für die Höhe h eines AVL-Baumes mit n Schlüsseln gilt $h = O(\log n)$.	X	
Bei einem Zweigüberdeckungstest wird jede Anweisung des getesteten Programms mindestens einmal ausgeführt.	X	
Ein ungeordnetes Array mit n Elementen lässt sich in $O(n)$ Schritten in einen Max-Heap überführen.	X	
Das Einfügen eines Schlüssels in einen B-Baum mit n Schlüsseln hat eine asymptotische Laufzeit von $\Omega(\sqrt{n})$.		X
Das Interface <code>java.util.Map</code> lässt sich mittels einer Skip-Liste implementieren.	X	
<code>wp("i:=7", 1+1=2)=false</code>		X

b) Wie lässt sich die Fehlerwahrscheinlichkeit eines beliebigen Monte-Carlo-Algorithmus verringern? (1 Punkt)

Durch mehrfaches, unabhängiges Wiederholen des Algorithmus. Dies wird auch als *Wahrscheinlichkeitsverstärkung* bezeichnet.

c) Wie viele binomiale Bäume mit genau 4 Knoten befinden sich in einem binomialen Heap, der 64 Schlüssel speichert? Begründen Sie Ihre Antwort kurz. (2 Punkte)

0. 64 ist eine Zweierpotenz, d.h. der binomiale Heap besteht aus genau einem binomialen Baum vom Grad 6 mit 64 Knoten.

Hinweis: Abhängig von der Begründung wurden hier auch andere Interpretationen akzeptiert.



2 Objektorientierter Entwurf mit UML (7 Punkte)

Gegeben sei folgender Javacode:

```
public interface IC {
    public void blup();
}

public class D implements IC {
    public void blup() {
        System.out.println("Blup");
    }
}

public class B {
    public void bla(IC einIC) {
        einIC.blup();
    }
}

public class A {
    private B b;

    public B getB() {
        return b;
    }

    public void setB(B einB) {
        this.b = einB;
    }

    public void bingo(IC einIC, int i) {
        if (i % 2 == 0) {
            B b = this.getB();
            b.bla(einIC);
        }
    }

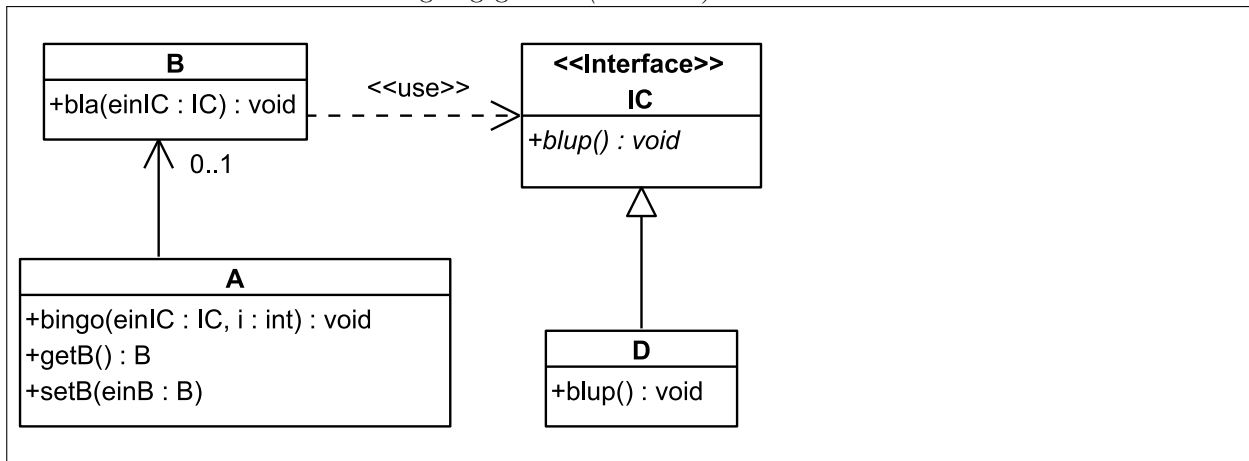
    public static void main(String[] args) {
        A einA = new A();
        einA.setB(new B());
        IC ic = new D();
        einA.bingo(ic, 3);
        System.out.println("Ende");
    }
}
```

a) Was ist die Ausgabe des Programms nach Aufruf von `A.main()`? (1 Punkt)

Ausgabe: "Ende"



- b) Modellieren Sie die im Javacode angegebenen Klassen inklusive ihrer Methoden und Assoziationen als UML-Klassendiagramm. Geben Sie Navigationsrichtungen und Multiplizitäten an. Berücksichtigen Sie die Parameter von Methoden und deren Typen sowie Rückgabetypen. Private und statische Methoden und Attribute, sowie Klassen der Java Klassenbibliothek müssen *nicht* berücksichtigt werden. *Hinweis:* Auf dieser und der folgenden Seite ist unten der gesamte Javacode von Seite 3 bis auf die `main()`-Methode in einer verkleinerten Darstellung angegeben. (3 Punkte)



```

public interface IC {
    public void blup();
}

public class D implements IC {
    public void blup() {
        System.out.println("Blup");
    }
}

public class B {
    public void bla(IC einIC) {
        einIC.blup();
    }
}
  
```

```

public class A {
    private B b;

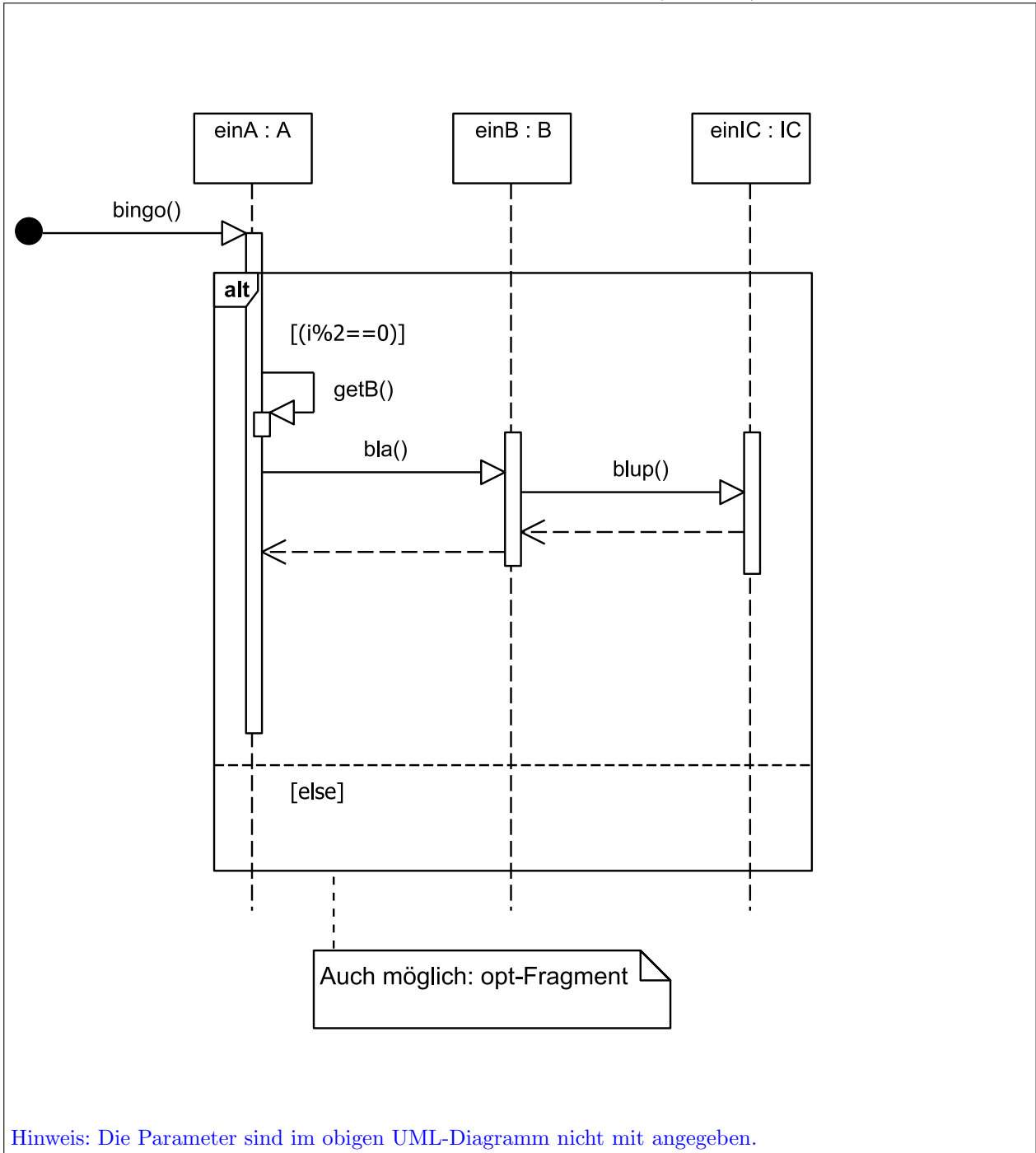
    public B getB() {
        return b;
    }

    public void setB(B einB) {
        this.b = einB;
    }

    public void bingo(IC einIC, int i) {
        if (i % 2 == 0) {
            B b = this.getB();
            b.bla(einIC);
        }
    }
    // ...
}
  
```



- c) Vervollständigen Sie das folgende UML-Sequenzdiagramm. Modellieren Sie detailliert. Berücksichtigen Sie Rückantworten, Parameter und Kontrollstrukturen. Berücksichtigen Sie außerdem die Typen der beteiligten Interaktionspartner. Auch hier müssen private und statische Methoden und Attribute, sowie Klassen der Java Klassenbibliothek *nicht* berücksichtigt werden. (3 Punkte)



```
public interface IC {
    public void blup();
}

public class D implements IC {
    public void blup() {
        System.out.println("Blup");
    }
}

public class B {
    public void bla(IC einIC) {
        einIC.blup();
    }
}
```

```
public class A {
    private B b;

    public B getB() {
        return b;
    }

    public void setB(B einB) {
        this.b = einB;
    }

    public void bingo(IC einIC, int i) {
        if (i % 2 == 0) {
            B b = this.getB();
            b.bla(einIC);
        }
    }
    //...
}
```



3 Asymptotische Notation (10 Punkte)

a) Beweisen Sie $n! = \Omega(n^2)$ mit Hilfe der Definition der Ω -Notation. (3 Punkte)

Um dies zu beweisen, müssen positive Konstanten n_0 und c bestimmt werden, sodass $0 \leq cn^2 \leq n!$ für alle $n \geq n_0$ gilt.

Es gilt

$$\begin{aligned} n! &\geq n \cdot (n-1) \cdot (n-2) && (\text{für } n \geq 3) \\ &= (n^2 - n) \cdot (n-2) \\ &= n^3 - 2n^2 - n^2 + 2n \\ &= n^3 - 3n^2 + 2n \end{aligned}$$

Noch zu zeigen

$cn^2 \leq n^3 - 3n^2 + 2n$. Division durch n^2 liefert $c \leq n - 3 + 2/n$. Diese Ungleichung lässt sich z.B. erfüllen durch $n \geq 3$ und $c \leq 2/3$.

b) Geben Sie für die folgenden Rekursionsgleichungen eine asymptotische obere und untere Schranke für $T(n)$ an. Gehen Sie davon aus, dass $T(n)$ für hinreichend kleine n konstant ist. Geben Sie möglichst scharfe Schranken an und begründen Sie Ihre Antworten. (7 Punkte)

$T(n) = 27T(n/3) + n^2$ Lösung mit Hilfe des Mastertheorems.

Es gilt: $a = 27$, $b = 3$ und $f(n) = n^2$ und somit

$f(n) = O(n^{\log_3 27 - \epsilon}) = O(n^{3 - \epsilon})$, z.B. für $\epsilon = 1$.

Es lässt sich also Fall 1 des Mastertheorems anwenden und es gilt

$T(n) = \Theta(n^3)$

$T(n) = 3T(n/9) + \sqrt{n} + 1729$

Lösung mit Hilfe des Mastertheorems.

Es gilt $a = 3$, $b = 9$ und $f(n) = \sqrt{n} + 1729$. Daher

$f(n) = \Theta(n^{\log_9 3}) = \Theta(n^{1/2})$.

Es lässt sich also Fall 2 des Mastertheorems anwenden und es gilt

$T(n) = \Theta(\sqrt{n} \log n)$

$T(n) = 2T(\lfloor n/2 \rfloor) + n!$ Lösung mit Hilfe des Mastertheorems.

Es gilt $a = 2$, $b = 2$ und $f(n) = n!$. Daher

$f(n) = \Omega(n^{\log_2 2 + \epsilon})$, z.B. für $\epsilon = 1$ (siehe Teilaufgabe a)).

d.h. es gilt Fall 3 des Mastertheorems falls wir zeigen können, dass die Regularitätsbedingung gilt:

$$\begin{aligned} a \cdot f(n/b) &\leq c \cdot f(n) \\ 2(\lfloor n/2 \rfloor)! &\leq c \cdot n! \\ \frac{2(\lfloor n/2 \rfloor)!}{n!} &\leq c \end{aligned}$$

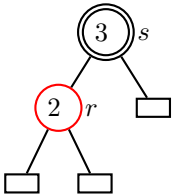
Da $\frac{2(\lfloor n/2 \rfloor)!}{n!} \leq 1$ für $n \geq 3$ ist die Regularitätsbedingung für ein $c < 1$ erfüllt. $\Rightarrow T(n) = \Theta(n!)$

Hinweis: Für eine reelle Zahl x bezeichnet $\lfloor x \rfloor$ (sprich: "floor von x ") die größte ganze Zahl, die kleiner oder gleich x ist.

4 Rot/Schwarz-Bäume (10 Punkte)

Hinweis: Im Folgenden sind einige Rot/Schwarz-Bäume dargestellt. Schwarze Knoten sind dabei ausgefüllt, rote Knoten sind nicht ausgefüllt. Knoten die einen Schlüssel speichern sind als Kreise mit dem entsprechenden Schlüssel dargestellt. Null-Knoten (Blattknoten) sind als Vierecke dargestellt. Verwenden Sie für Ihre eigenen Zeichnungen zur Darstellung *schwarzer* innerer Knoten bitte *doppelt umrundete* Kreise. Sofern Sie mehrfarbige Stifte besitzen, können Sie auch stattdessen diese benutzen. Falls Sie verbessern müssen, schreiben Sie bitte noch *r* und *s*, für rot bzw. schwarz, rechts neben die entsprechenden Knoten.

Beispiel für Ihre eigene Zeichnung:



- a) Fügen Sie den Schlüssel 7 in den folgenden Rot/Schwarz-Baum ein und führen Sie die eventuell nötigen Operationen zur Wiederherstellung der Eigenschaften des Rot/Schwarz-Baumes durch. Zeichnen Sie den Baum nach dem Einfügen. Fertigen Sie außerdem eine Zeichnung des Baumes nach jeder erfolgten Rotation von Knoten an inklusive der erfolgten Umfärbungen. Zeichnen Sie den Rot/Schwarz-Baum nach der Wiederherstellung der Eigenschaften. (4 Punkte)

Fall 2: Durch Rechtsrotation in Fall 3 überführen.

Einfügen:

Fall 3: Rot färben des Knotens mit Schlüssel 5. Schwarz färben des Knoten mit Schlüssel 7. Linksrotation über 5.



b) Löschen Sie den Schlüssel 3 aus dem folgenden Rot/Schwarz-Baum und führen Sie die eventuell nötigen Operationen zur Wiederherstellung der Eigenschaften des Rot/Schwarz-Baumes durch. Zeichnen Sie den Baum nach dem Löschen. Fertigen Sie außerdem eine Zeichnung des Baumes nach jeder erfolgten Rotation von Knoten an inklusive der erfolgten Umfärbungen. Zeichnen Sie den Rot/Schwarz-Baum nach der Wiederherstellung der Eigenschaften. (3 Punkte)

Ausgeschnittener Knoten schwarz. Fall 2: Bruder w ist schwarz und hat 2 schwarze Kinder.

Färbe 5 rot und 4 nach Abschluss der Schleife der fixup-Prozedur schwarz.

c) Was ist die maximale Anzahl von roten Knoten in einem Rot/Schwarz-Baum, der 31 Schlüssel speichert? Begründen Sie Ihre Antwort kurz. (3 Punkte)



Name:

Matrikelnummer:

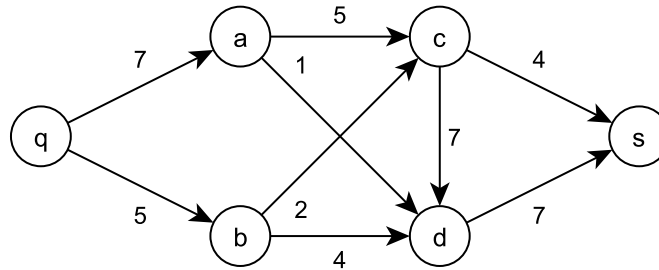
Hinweis: Die Null-Knoten werden im Folgenden nicht beachtet.

Die Anzahl der roten Knoten lässt sich maximieren, indem beginnend mit der Blattebene jede zweite Ebene möglichst vollständig mit roten Knoten besetzt wird. Aus 31 Schlüsseln ($= 2^5 - 1$) lässt sich ein voll besetzter R/S-Baum mit Höhe 4 erstellen, bei dem jede zweite Ebene, angefangen in Tiefe 4 vollständig aus roten Knoten besteht. Die max. Anzahl der roten Knoten beträgt $2^4 + 2^2 = 16 + 4 = 20$.



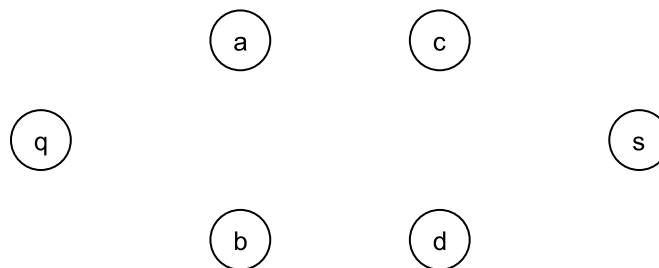
5 Netzwerkflüsse (9 Punkte)

- a) Wenden Sie den Ford-Fulkerson Algorithmus auf das unten angegebene Netzwerk an und berechnen Sie den maximalen Fluss von der Quelle q zur Senke s . Wählen Sie dabei als nächsten Pfad jeweils den Pfad mit der höchsten Kapazität. Sollte es mehrere solcher Pfade geben, wählen Sie den kürzesten. Geben Sie den berechneten maximalen Fluss sowie alle gewählten Pfade mitsamt ihrer Kapazität in der Reihenfolge an, in der Sie sie ausgewählt haben. (5 Punkte)



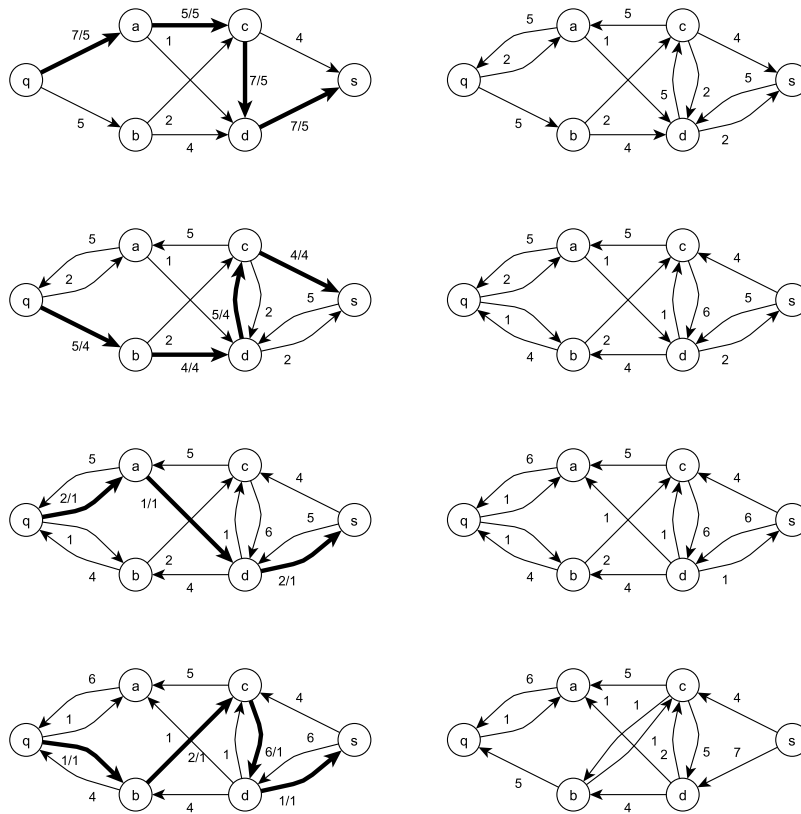
$q \rightarrow a \rightarrow c \rightarrow d \rightarrow s : 5$
 $q \rightarrow b \rightarrow d \rightarrow c \rightarrow s : 4$
 $q \rightarrow a \rightarrow d \rightarrow s : 1$
 $q \rightarrow b \rightarrow c \rightarrow d \rightarrow s : 1$
 Fluss (Summer aller Wege): 11

- b) Zeichnen Sie das Restkapazitätsnetzwerk nach Anwendung des Algorithmus. Das Restkapazitätsnetzwerk enthält überall dort Kanten mit Restkapazitätsangaben, wo noch (theoretische) Flussmöglichkeiten bestehen. Beträgt eine Restkapazität 0, kann die Kante entfallen. (3 Punkte)

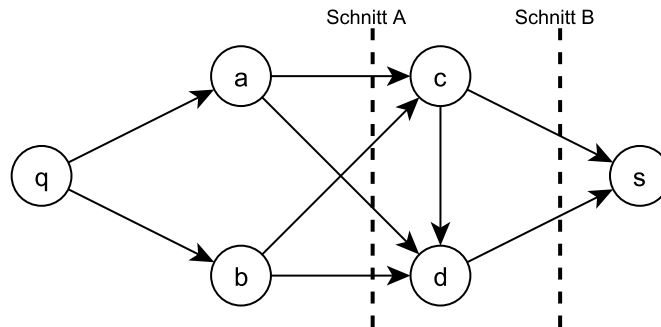


Genauer Lösungsweg, gefordert ist nur der Graph ganz unten rechts (links ist jeweils der Fluss vom aktuellen Pfad, rechts das Restkapazitätsnetzwerk):





c) In dem im Folgenden dargestellten Netzwerk mit Quelle q und Senke s sind zwei Schnitte eingezeichnet. Nehmen Sie an, dass die Summe aller Flüsse an Schnitt B den Wert 9 aufweist. Geben sie möglichst präzise einen Wert oder Wertebereich für die Summe aller Flüsse an Schnitt A an. (1 Punkte)



Laut Vorlesung gilt: "Netto Fluß ist an allen Schnitten identisch". Folglich ist die Summe von Schnitt A gleich der Summe von Schnitt B, also 9.



6 Scan-line Prinzip (9 Punkte)

In Abbildung 1 ist ein in der Vorlesung behandelter Pseudocode eines Scanline-Algorithmus mit Lücken abgebildet. Dieser berechnet die Schnittpunkte einer Menge n iso-orientierter Strecken unter der Voraussetzung, dass Anfangs- und Endpunkte horizontaler Strecken und alle vertikalen Segmente paarweise verschiedene x -Koordinaten haben.

```

1   $Q :=$  Menge der  $x$ -Koordinaten der Anfangs- und Endpunkte horizontaler
2     Strecken und von vertikalen Strecken in aufsteigender  $x$ -Reihenfolge;
3
4   $L :=$  leer; //Menge der jeweils aktiven horizontalen Strecken in
5     aufsteigender  $y$ -Reihenfolge
6
7  while .....  $Q$  ist nicht leer do
8     begin
9          $p :=$  nächster (Halte)-Punkt von .....  $Q$  ;
10        if  $p$  ist ..... linker Endpunkt (oder „Anfangspunkt“)
        einer horizontalen Strecke  $s$  then
11            füge  $s$  in .....  $L$  ein;
12        else
13            if  $p$  ist ..... rechter Endpunkt (oder „End-
        punkt“) einer horizontalen Strecke  $s$  then
14                entferne  $s$  aus .....  $L$  ;
15            else
16                //  $p$  ist  $x$ -Wert einer vertikalen Strecke  $s$ 
17                // mit unterem Endpunkt  $(p; y_u)$  und oberem Endpunkt  $(p; y_o)$ 
18                bestimme alle horizontalen Strecken  $t$  aus  $L$ ,
19                deren  $y$ -Koordinate  $y(t)$  im Bereich  $y_u \leq y(t) \leq y_o$  liegt,
20                und gib  $(s,t)$  als Paar sich schneidender Strecken aus;
21    end while

```

Abbildung 1: Algorithmus *Schnitte*

- a) Der Pseudocode ist gegenüber dem aus der Vorlesung bekannten unverändert, abgesehen von 6 Lücken (Zeilen 7, 9, 10, 11, 13, 14). Füllen Sie diese Lücken so aus, dass der Algorithmus das Gewünschte (wie oben beschrieben) leistet. (2 Punkte)

s. Pseudocode

- b) Welchen Aufwand hat dieser Algorithmus laut Vorlesung (ohne Begründung)? Geben Sie eine möglichst scharfe obere Schranke an. (1 Punkte)

$O(n \log n + k)$ hat für k Schnittpunkte



- c) Der Aufwand des Algorithmus liegt nicht in $O(n \log n)$. Ist es möglich, den Algorithmus so zu optimieren, dass er in $O(n \log n)$ liegt? Bitte begründen Sie ihre Antwort kurz! (2 Punkte)

Nein, dies ist nicht möglich: Im schlimmsten Fall existieren quadratisch viele Schnittpunkte. Somit kann im Allgemeinen keine Lösung besser sein als $O(n^2)$.

- d) Der Pseudocode ist in den Zeilen 16 bis 20 wenig detailliert. Mit welcher Datenstruktur muss die Menge L implementiert sein, um den in der Vorlesung angegebenen Aufwand zu erreichen? Skizzieren Sie kurz das Vorgehen zur Bestimmung der Strecken, deren y-Koordinate im Bereich $[y_u, y_o]$ liegen? Hinweis: Der Aufwand braucht an dieser Stelle nicht diskutiert zu werden. (2 Punkte)

L ist als balancierter Suchbaum mit doppelter Verkettung benachbarter Blätter zu implementieren. In diesem Fall kann die betrachtete Bereichsanfrage (range query) wie folgt bearbeitet werden: Zunächst bestimmt man durch Suchoperationen das Blatt mit kleinstem Wert größer gleich y_u . Anschließend gibt man entlang der doppelt verketteten Liste solange Elemente aus, bis man auf eines echt größer y_o trifft. Sei r die Anzahl der Objekte im Bereich $[y_u, y_o]$, dann liegt dieses Vorgehen in $O(\log N + r)$.

- e) Kann man L auch als Hash-Tabelle implementieren, ohne dass der Aufwand des Algorithmus (laut Vorlesung) darunter leidet? Bitte begründen Sie Ihre Antwort kurz. Gehen Sie bei Ihrer Begründung davon aus, dass Einfügen und Löschen in eine Hash-Tabelle in diesem Fall in konstanter Zeit möglich sind. (2 Punkte)

Nein: Die wesentlichen Operationen am Haltepunkt sind Einfügen, Löschen und das Suchen der beiden Nachbarn in jedem Schritt. Bei perfekten Hashing (s. Annahme) sind Einfügen und Löschen in $O(1)$ möglich und somit unproblematisch. Das Suchen der Nachbarn allerdings ist mit einer Hash-Tabelle nicht machbar, ohne dass der Aufwand zu $O(n^2)$ degeneriert.



7 Entwurf eines Algorithmus in Pseudocode (8 Punkte)

Gegeben sei eine ganze Zahl k sowie ein ungeordnetes Feld $A[1..n]$, das n (paarweise verschiedene) ganze Zahlen enthält, mit $n \geq 2$.

- a) Beschreiben Sie einen Algorithmus in Pseudocode, der mit einer Laufzeit von $O(n)$ WAHR zurück gibt, falls die Summe der größten $\lfloor \log_2 n \rfloor$ Zahlen von A größer als k ist, und FALSCH sonst. *Hinweis:* Sie können alle in der Lehrveranstaltung Informatik II angegebenen Algorithmen als gegeben voraussetzen und die entsprechenden Prozeduren bei Bedarf aufrufen. (6 Punkte)

```
1 SUM-LOG-N-BIGGEST-NUMBERS(A,k)
2   BUILD-MAX-HEAP(A)
3   summe:=0
4   for i:=1 to  $\lfloor \log_2 n \rfloor$  do
5       summe:=summe+HEAP-EXTRACT-MAX(A)
6   if summe>k
7       then return WAHR
8       else return FALSCH
```

Weitere Lösungsmöglichkeit mit Hilfe des $O(n)$ Algorithmus zur Lösung des Auswahlproblems.

- b) Begründen Sie kurz, dass Ihr Algorithmus tatsächlich die geforderte asymptotische Laufzeit aufweist. (2 Punkte)

Aufwand BUILD-MAX-HEAP(A) = $O(n)$, HEAP-EXTRACT-MAX(A) = $O(\log n)$. Schleife wird $O(\log n)$ -Mal durchlaufen, hat also einen Aufwand von $O(\log^2 n)$. Der Gesamtaufwand beträgt daher $O(n) + O(\log^2 n) = O(n)$.

