Matrikelnummer:			
DHBW Duale Hochschule Baden-Württemberg Stuttgart **ÜBUNGSKLAUSUR**		Fakultät Studiengang: Jahrgang / Kurs : Studienhalbjahr:	**Technik** **Informatik** **23C** **2. Semester**
Datum:	**18.7.2024**	Bearbeitungszeit:	**90 Minuten**
Modul: Unit:	**T3INF1003.1** **Algorithmen**	Dozenten:	**Schulz**
Hilfsmittel:	**Zwei beliebige Papierwerke, Skript auch auf Tablet im Flugmodus**		

Aufgabe	Hinweis zum Inhalt	erreichbar	erreicht
1	Sortieren	11	
2	Komplexität	7	
3	Kartenfolgen	12	
4	Funktionen analysieren	6	
5	Heaps	12	
6	Graphen	11	
7	Hashing	12	
8	Bäume	9	
Summe		80	

1. Sind Sie gesund und prüfungsfähig? Wer krank ist, kann jetzt die Prüfung noch verlassen. (Antrag auf Prüfungsrücktritt und ärztliches Attest!) Wer bleibt, hat damit erklärt, dass er sich gesund und prüfungsfähig fühlt.
2. Ein Täuschungsversuch und/oder die Benutzung nicht zugelassener Hilfsmittel führen zum Nichtbestehen der Klausur (5,0).
3. Wer den ordnungsgemäßen Ablauf der Prüfung stört oder sich nicht an die besonderen Regeln im Rahmen des Corona-Infektionsschutzes hält, kann von der weiteren Prüfung ausgeschlossen werden.
4. Auch außerhalb des Klausurraumes dürfen keine unerlaubten Hilfsmittel oder Unterlagen für die Klausur deponiert, bzw. verwendet werden.
5. Es dürfen während der Klausur keine Unterlagen und Informationen ausgetauscht oder weitergegeben werden.
6. Fragen an die Aufsicht sind nur hinsichtlich der Aufgabenstellung erlaubt.
7. Die Matrikel-Nr. wird auf das Deckblatt und alle weiteren Blätter der Klausur geschrieben. Der Name der/des Studierenden darf nicht auf der Klausur erscheinen.

Aufgabe 1 (5+1+1+4 Punkte)
Betrachten Sie die folgende Folge:
$S = (1, 21, 2, 34, 7, 9, 12, 8, 17, 4)$.

a) Sortieren Sie die Folge S gemäß der Ordnung $>$ (d.h. *absteigend*) auf natürlichen Zahlen. Verwenden Sie das in der Vorlesung gezeigte *Selectionsort*-Verfahren (*Sortieren durch Auswählen*). Geben Sie den Zustand von S für jeden Durchlauf der äußeren Schleife an, sowie die Anzahl der Vergleiche und Vertauschungen.

b) Wie viele *Vertauschungen* von Elementen der Folge S werden insgesamt benötigt? Vertauschungen mit sich selbst werden mitgezählt.

c) Wie viele *Vergleiche* werden insgesamt benötigt?

d) Nehmen wir an, wir würde dieselbe Folge S mit dem *Quicksort*-Verfahren *absteigend* sortieren wie in der Vorlesung gezeigt. Partitionieren Sie die Folge *einmal* mit dem LL-Algorithmus (nach Nico Lomuto). Wählen Sie das mittlere Element als Pivot.

Lösung

a) 0,5 Punkte für alle Zeilen, 2 Punkte Abzug wenn konsequent falsch herum sortiert wird, 0,5 Punkte für falsches Sortierverfahren (?), dann aber ggfs. bei b) und c) Folgefehler (also noch 2 Punkte möglich)

Gegeben:	1	21	2	34	7	9	12	8	17	4		
Iteration 1:	34	21	2	1	7	9	12	8	17	4	Cmps: 9	Swps: 1
Iteration 2:	34	21	2	1	7	9	12	8	17	4	Cmps: 8	Swps: 1
Iteration 3:	34	21	17	1	7	9	12	8	2	4	Cmps: 7	Swps: 1
Iteration 4:	34	21	17	12	7	9	1	8	2	4	Cmps: 6	Swps: 1
Iteration 5:	34	21	17	12	9	7	1	8	2	4	Cmps: 5	Swps: 1
Iteration 6:	34	21	17	12	9	8	1	7	2	4	Cmps: 4	Swps: 1
Iteration 7:	34	21	17	12	9	8	7	1	2	4	Cmps: 3	Swps: 1
Iteration 8:	34	21	17	12	9	8	7	4	2	1	Cmps: 2	Swps: 1
Iteration 9:	34	21	17	12	9	8	7	4	2	1	Cmps: 1	Swps: 1

b) 9

c) $\Sigma_{i=1}^{n-1} i = \frac{n \cdot (n-1)}{2} = 45$

d) Pivot ist die 7. Ergebnis s.u. 0,5 Punkte pro Schritt oder alle 4 Punkte bei richtigem Ergebnis.

Gegeben:	1	21	2	34	7	9	12	8	17	4
Pivot ans Ende:	1	21	2	34	4	9	12	8	17	7
Tausche 1/21:	21	1	2	34	4	9	12	8	17	7
Tausche 1/34:	21	34	2	1	4	9	12	8	17	7
Tausche 2/9:	21	34	9	1	4	2	12	8	17	7
Tausche 1/12:	21	34	9	12	4	2	1	8	17	7
Tausche 4/8:	21	34	9	12	8	2	1	4	17	7
Tausche 2/17:	21	34	9	12	8	17	1	4	2	7
Pivot zurück:	21	34	9	12	8	17	7	4	2	1

Fortsetzung

Aufgabe 2 (2+2+3 Punkte)
Für die folgenden Funktionen gilt $x \in \mathbb{R}$

a) Gegeben seien eine konstante Zahl $k \in \mathbb{R}, k > 0$ und die folgenden beiden Funktionen $f : \mathbb{R} \to \mathbb{R}$ und $g : \mathbb{R} \to \mathbb{R}$ mit $f(x) = x^{2k}, g(x) = kx^2$.

- Für welche Wertebereiche von k gilt $f \in \mathcal{O}(g)$?
- Für welche Wertebereiche von k gilt $g \in \mathcal{O}(f)$?
- Für welche Wertebereiche von k gilt $f \sim g$?

b) Gegeben seien eine konstante Zahl $k \in \mathbb{R}, k > 0$ und die folgenden beiden Funktionen $f : \mathbb{R} \to \mathbb{R}$ und $g : \mathbb{R} \to \mathbb{R}$ mit $f(x) = \ln 2^{kx}, g(x) = \ln k^{2x}$.

- Für welche Wertebereiche von k gilt $f \in \mathcal{O}(g)$?
- Für welche Wertebereiche von k gilt $g \in \mathcal{O}(f)$?
- Für welche Wertebereiche von k gilt $f \sim g$?

c) Betrachten Sie nun $f : \mathbb{R} \to \mathbb{R}$ und $g : \mathbb{R} \to \mathbb{R}$ mit $f(x) = x \cdot \ln(x \cdot x), g(x) = x \cdot \sqrt{4 \cdot x}$ Gilt $f \in \mathcal{O}(g)$? Begründen Sie!

Lösung

Jeweils 0,5 Punkte für jeden Wertebereich und 0,5 Punkte, falls nirgendwo < / <= oder > / >= verwechselt

a)
- Für welche Wertebereiche von k gilt $f \in \mathcal{O}(g)$? $k <= 1$
- Für welche Wertebereiche von k gilt $g \in \mathcal{O}(f)$? $k >= 1$
- Für welche Wertebereiche von k gilt $f \sim g$? $k = 1$

b) $f(x) = \ln 2^{kx} = kx \ln 2, g(x) = 2x \ln k$

- Für welche Wertebereiche von k gilt $f \in \mathcal{O}(g)$? $k > 0$ (alle definierten Werte)
- Für welche Wertebereiche von k gilt $g \in \mathcal{O}(f)$? $k > 0$ (alle definierten Werte)
- Für welche Wertebereiche von k gilt $f \sim g$? $k = 2$

c) Behauptung $f \in \mathcal{O}(g)$, zunächst Vereinfachung beider Funktionen:
$f(x) = x \cdot \ln(x \cdot x) = x \cdot \ln(x^2) = 2x \cdot \ln(x)$
$g(x) = x \cdot \sqrt{4 \cdot x} = x \cdot \sqrt{4} \cdot \sqrt{x} = 2x \cdot \sqrt{x}$
Anwendung des Grenzwertkriteriums:

$$\begin{aligned}
\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{2x \cdot \ln(x)}{2x \cdot \sqrt{x}} \\
&= \lim_{x\to\infty} \frac{\ln(x)}{\sqrt{x}} \qquad \text{(Kürzen)} \\
&= \lim_{x\to\infty} \frac{\frac{1}{x}}{\frac{1}{2\sqrt{(x)}}} \qquad \text{(l'Hospital)} \\
&= \lim_{x\to\infty} \frac{2\sqrt{x}}{x} \qquad \text{(Kehrwert)} \\
&= \lim_{x\to\infty} \frac{2}{\sqrt{x}} \qquad \text{(Kürzen)} \\
&= 0
\end{aligned}$$

Fortsetzung

Aufgabe 3 (1+5+1+2+1+2 Punkte)
Es gibt auf der Welt viele verschiedene Kartenspiele. Die Karten werden in der Vorbereitung oft gemischt, wobei das Mischen zufällig erfolgen soll. In vielen Kartenspielen gibt es zwei Sorten von Karten. Beispiele sind Trumpfkarten und Nicht-Trumpfkarten, Länder und Zauberer, rote und schwarze Karten.

In diesem Beispiel können Sie o.B.d.A. von einem Kartensatz (genannt Deck) aus 52 Karten mit französischem Blatt ausgehen, davon 26 Karten rot, die anderen 26 schwarz. Für alle Aufgaben starten Sie mit einem zufällig gemischten Deck, d.h. alle Permutationen des Decks sind gleich wahrscheinlich.

Für viele Spiele ist es von Vor- oder Nachteil, wenn Karten der gleichen Farbe, z. B. rote Karten, in einer Permutation aufeinander folgen. Daher ist es interessant, die Länge der längsten Folge von Karten einer Farbe zu zählen. Hier folgt ein Beispiel, bei dem jeder Spieler hintereinander 9 Karten aus dem eigenen Deck zieht:

Gezogene Karte	1	2	3	4	5	6	7	8	9
Spieler 1	Rot	Schwarz	Schwarz	Schwarz	Rot	Schwarz	Rot	Rot	Schwarz
Spieler 2	Schwarz	Rot	Rot	Rot	Rot	Schwarz	Schwarz	Rot	Schwarz

a1) Was ist die Länge der längsten Folge *roter* Karten von Spieler 1? Von Spieler 2?

a2) Entwerfen Sie in *Pseudocode* oder einer geeigneten Programmiersprache ihrer Wahl eine Funktion, die zu einer Folge von Karten die *längste rote Folge* berechnet und zurückgibt. Sie können davon ausgehen, dass die Eingabe in Form eines Arrays `sample` mit `n` Elementen übergeben wird, das von 0 bis $\texttt{n}-1$ addressiert wird. Der Inhalt ist vom Typ `int`, und zwar 0 für schwarz, 1 für rot.

a3) Welche Laufzeitkomplexität (im Sinne von $\mathcal{O}$ bzw. Θ) hat ihre Lösung zu Teil (a1)? Begründen Sie ihre Aussage!

b1) Skizzieren Sie in *Stichworten* bzw. halbformellen Text einen Algorithmus für die Zählung aller roten Folgen einer bestimmten Länge `l` in einem Deck basierend auf Ihrer (oder einer fiktiven) Lösung der Aufgabe a2! Achtung: Gesucht ist die genaue Länge, d. h. eine Folge der Länge drei wird nicht der Länge zwei zugerechnet.

b2) Welche Laufzeitkomplexität (im Sinne von $\mathcal{O}$ bzw. Θ) hat ihre Lösung zu Teil (b1)? Begründen Sie ihre Aussage!

b3) Wenn Sie die Decks der beiden Spieler ansehen, welches scheint im Sinne der roten Folgen *besser* verteilt zu sein? Skizzieren Sie (in Worten) eine Lösung, wie man die *Rot-Verteilung* von zwei Decks vergleichen könnte.

Lösung

a1) Spieler 1: 2, Spieler 2: 4

a2)
```
int findmaxredseries (int[] sample, int n) {
      int maxred = 0;
      int countred = 0;
      int i = 0;
      for (i = 0; i < n; i++) {
        if (sample[i] == 0)
           counter = 0;
        else {
           counter++;
           if (counter > maxred)
              maxred = counter;
       }
       return maxred;
}
```

Wenn die Indices durcheinander geraten, 1 Punkt Abzug.

a3) Die Funktion hat nur einen Schleife, die von 0 bis $n-1$ läuft. Alle anderen Operationen haben konstante Zeit. Die Komplexität ist also $\Theta(n)$.

b1)
- Erweitere den Funktionsheader um die gesuchte Folgenlänge
- Füge eine Variable hinzu um die Folgenanzahl zu zählen (redseries) statt maxred, das kann weg

- Wenn eine schwarze Karte gefunden wird, dann prüfen, ob countred der gesuchten Länge entspricht
- Wenn ja, redseries erhöhen
- return redseries (statt maxred)

b2) So wie es hier steht $\Theta(n)$, da sich nur konstante Operationen innerhalb der Schleife ändern.

b3) Zähle die Länge aller 2-er Folgen, aller 3-er Folgen, etc., gewichte die Folgen (z. B. mit der Anzahl an aufeinanderfolgenden Karten), zähle das zusammen.

Fortsetzung

Aufgabe 4 (3+3 Punkte)

a) Betrachten Sie folgende Rekurrenzrelation: $R(n) = 2 \cdot R(\frac{n}{4}) + 4\sqrt{8n} + 16$. Schätzen Sie (im Sinne von $\mathcal{O}$ oder θ) die Funktion möglichst gut ab.

b) Die unten stehende C-Funktion `fun()` gibt mehrfach den Text *Hello World!* aus. Schätzen Sie (möglichst genau) die Laufzeitkomplexität der Funktion (im Sinne von $\mathcal{O}$) ab. Begründen Sie Ihre Antwort.

```
int fun(int n)
{
  int i,k;

  for (i=0; i<n; i++) {
    int j = i;
    while(j < i * i) {
      j = j + 1;
      if (j % 3 == 0) {
        for (k=0; k<j; k++) {
            printf("Hello World!");
        }
      }
    }
  }
  return 0;
}
```

Lösung

a) Master-Theorem: $a = 2, b = 4, d = 1$, also $b^d = 4 > 2 = a$. Fall 1 und damit $R \in \mathcal{O}(n)$ aber in dem Fall eben nicht $\in \theta(..)$.

Ein Punkt Abzug, falls Theta benutzt.

b) Äußere for-Schleife n-mal.

innere while-Schleife läuft von i bis i^2 (also max. n^2).

if-Bedingung springt bei vielfachen von 3 an, also Faktor 1/3 (konstant).

Innerste Schleife läuft zum aktuellen wert von j (also max. n^2).

Ergibt gesamt $fun \in O(n^5)$.

Fortsetzung

Aufgabe 5 (7+5 Punkte)

a) Betrachten Sie die Folge

$$S = (b, d, i, c, g, e, a, h, f)$$

als fast vollständigen Binärbaum (potentiellen Max-Heap) in Array-Darstellung. Wir verwenden die normale alphabetische Ordnung, d.h. a is das kleinste Element, i das größte Element.

- Stellen Sie das Array als fast vollständigen Binärbaum dar.
- Führen Sie auf dem Baum die Operation *Heapify* aus, um ihn in einen Max-Heap zu überführen. Schreiben Sie für jeden Schritt, welches Element sie bewegen, und zeichnen Sie das Ergebnis als Baum.
- Schreiben Sie das Ergebnis (den tatsächlichen Max-Heap) als Folge der Elemente.

b) Betrachten Sie jetzt den Max-Heap

$$T = (29, 23, 28, 16, 23, 27, 1, 12, 9, 17, 10, 8, 5)$$

in Array-Darstellung. Entfernen Sie aus T nacheinander die 3 größten Werte. Sie müssen diese *nicht* wie beim Heapsort unten im Array wieder einfügen!

- Stellen Sie auch hier den Ausgangsheap graphisch dar und geben Sie eine graphische Darstellung des Heaps nach jedem entfernten Wert an.
- Notieren Sie den endgültigen Zustand des Heaps als Array/Zahlenfolge.

Lösung

a) (1 Punkt für den ersten Baum und das endgülitge Array, 1 Punkt für jeden anderen Baum)

```
Original array in tree form

               b
       d               i
   c       g       e       a
 h   f

Bubbling down  c
               b
       d               i
   h       g       e       a
 c   f

Bubbling down  i
               b
       d               i
   h       g       e       a
 c   f

Bubbling down  d
               b
       h               i
   f       g       e       a
 c   d

Bubbling down  b
               i
       h               e
   f       g       b       a
 c   d

Heap
               i
       h               e
   f       g       b       a
 c   d

Final Array['i', 'h', 'e', 'f', 'g', 'b', 'a', 'c', 'd']
```

Fortsetzung

Lösung

b) (Ein Punkt pro Baum, ein Punkt für das endgültige Array)

```
Original Heap ============
              29
      23                28
  16      23      27        1
12   9  17  10   8   5

Removed element 29

              28
      23                27
  16      23       8        1
12   9  17  10   5 (29)

Removed element 28

              27
      23                8
  16      23       5        1
12   9  17  10 (28)(29)

Removed element 27

              23
      23                8
  16      17       5        1
12   9  10 (27)(28) (29)

[23, 23, 8, 16, 17, 5, 1, 12, 9, 10]
bzw. als Anfang von HeapSort:
[23, 23, 8, 16, 17, 5, 1, 12, 9, 10, 27, 28, 29]
```

Fortsetzung

Aufgabe 6 (5+6 Punkte)

a) Betrachten Sie den folgenden Graphen G_a. Verwenden Sie den Algorithmus von Prim, um einen minimalen Spannbaum, ausgehend vom Knoten **e**, zu bestimmen. Tragen Sie dazu in die folgende Tabelle die jeweils ausgewählten Knoten, das Gewicht der neu hinzugefügten Kante, und das Gesamtgewicht des verbundenen Teilbaums in der Reihenfolge ein, die der Algorithmus vorgibt. Hinweis: Bei diesem Graphen gibt es nur eine richtige Lösung. Auf der übernächsten Seite finden Sie Backup-Kopien der Graphen, wenn ihre Notizen zu unübersichtlich werden.

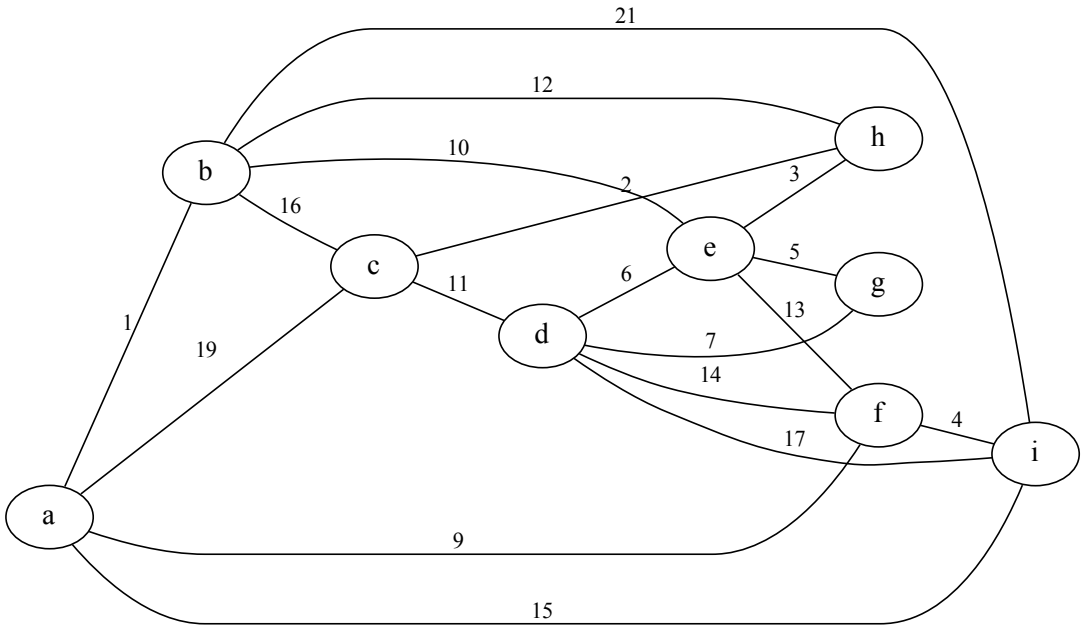


Abbildung 1: Graph G_a für Aufgabe 6a

Neuer Knoten	Neues K-Gewicht	Gesamt-gewicht
e	-	(0)

(Fortsetzung nächste Seite)

b) Betrachten Sie nun den folgenden Graphen G_b. Verwenden Sie den Algorithmus von Dijkstra, um die minimale Entfernung aller Knoten in G_b vom Knoten **a** zu bestimmen. Geben Sie die Knoten in der Reihenfolge an, in der sie im Algorithmus durchlaufen werden (d.h. in der ihre endgültige Entfernung bestimmt wird). Es kann durchaus vorkommen, dass mehr als eine Reihenfolge möglich ist - in diesem Fall wählen Sie bitte den Knoten mit dem alphabetisch kleineren Namen zuerst (wenn z.B. a und g möglich wären, expandieren Sie a zuerst).

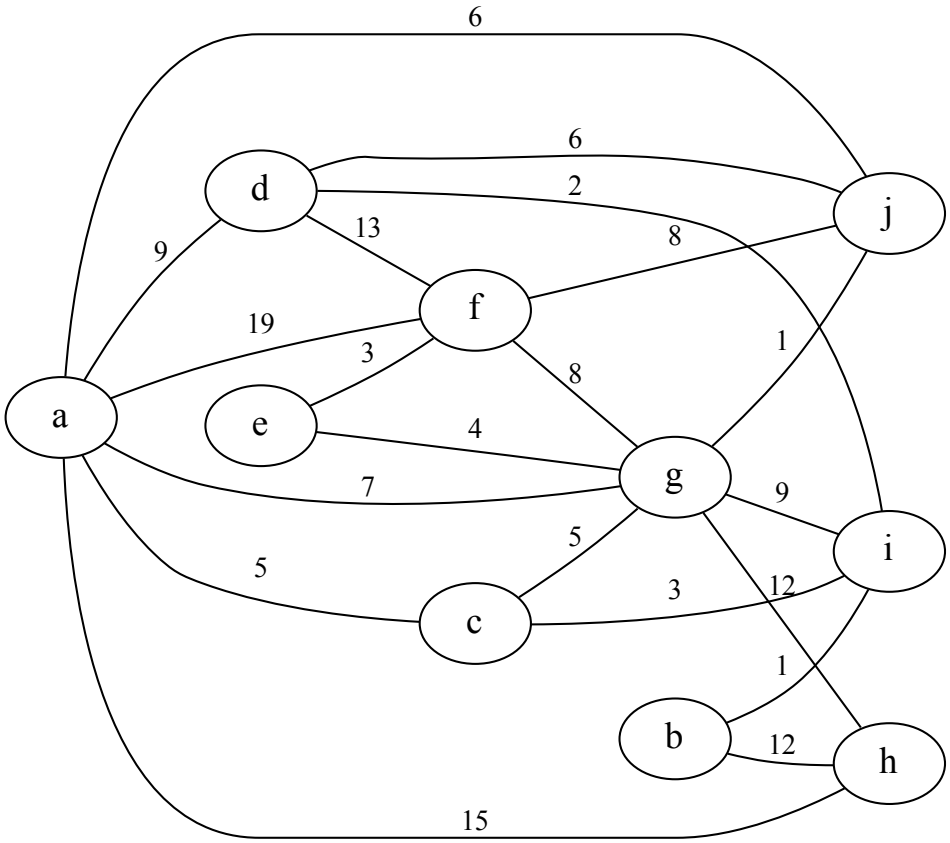
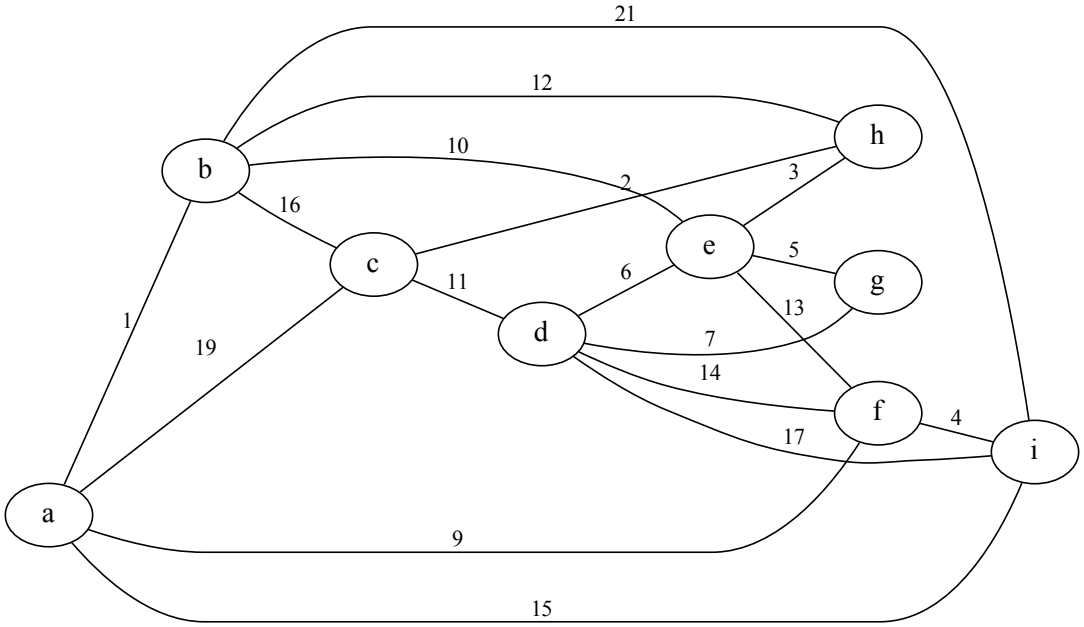


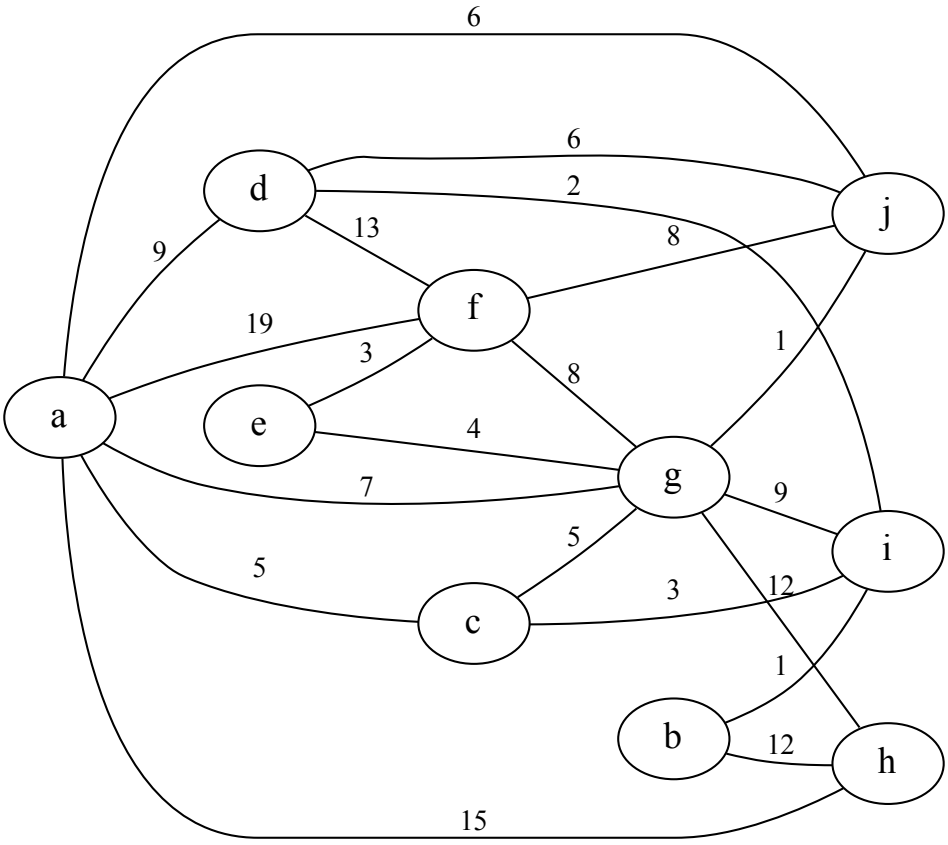
Abbildung 2: Graph G_b für Aufgabe 6b

Knoten	Abstand
a	0

Backup-Kopien der Graphen



Graph G_a für Aufgabe 6a

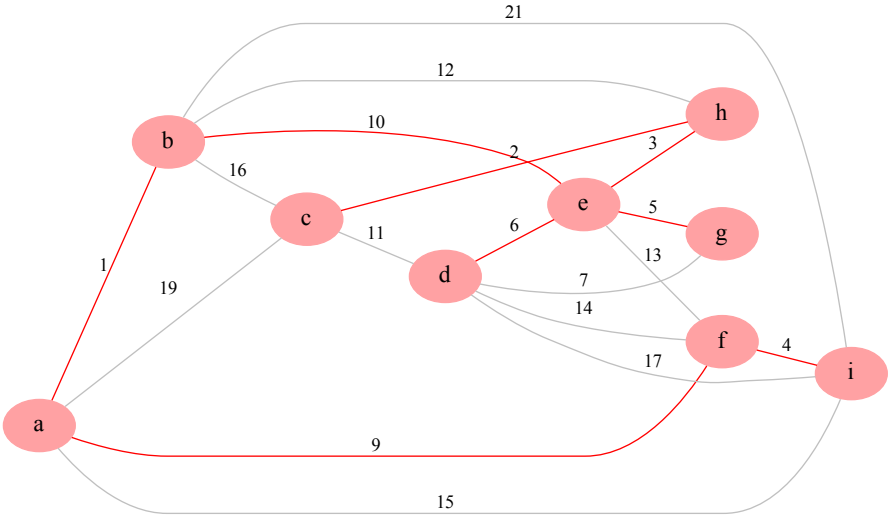


Graph G_b für Aufgabe 6b

Fortsetzung

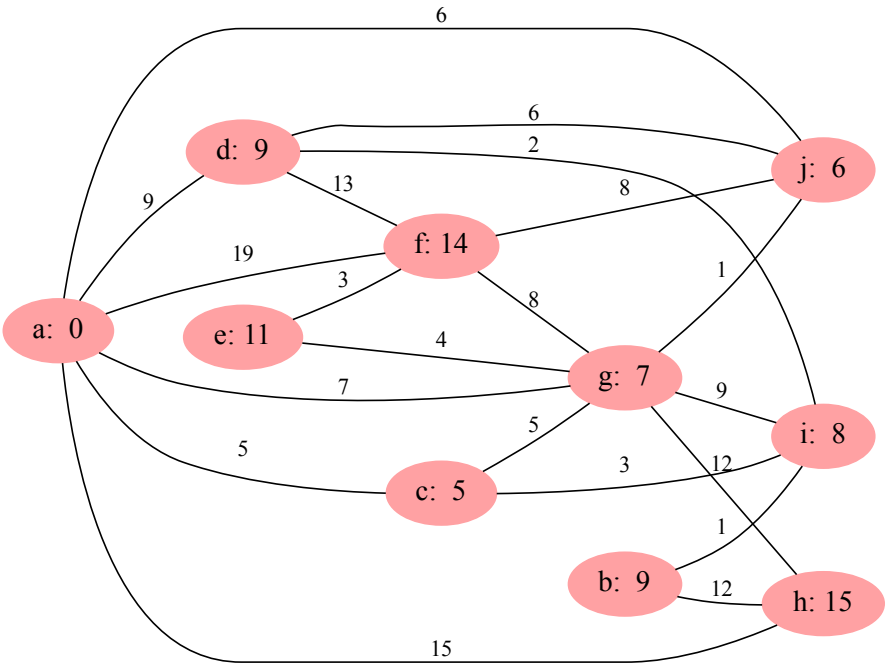
Lösung

a) je 1/2P für h-a, je 1P für die restlichen.



Neuer Knoten	Neues K-Gewicht	Gesamt-gewicht
e	-	(0)
h	3	3
c	2	5
g	5	10
d	6	16
b	10	26
a	1	27
f	9	36
i	4	40

b) 2/3P pro korrektem Knoten, Gesamtergebnis auf 0.5 runden.



Knoten	Abstand
a	0
c	5
j	6
g	7
i	8
b	9
d	9
e	11
f	14
h	15

Aufgabe 7 (3+3+6 Punkte)
Gegeben sei die Schlüsselfolge

$$S = (21, 2, 8, 17, 34, 9)$$

und die Hash-Funktion $h(x) = ((|x| * 2) \mod 13)$ sowie die Rehash-Funktion $r(x) = (|x| \mod 5) + 1$.

a) Geben Sie zunächst das Ergebnis der Hash-Funktion für jeden Schlüsselwert in S an. Verwenden Sie dafür die erste der Tabellen unten.

b) Geben Sie nun das Ergebnis der Rehash-Funktion für jeden Schlüsselwert in S an. Verwenden Sie dafür die erste der Tabellen unten.

c) Fügen Sie nun die Elemente von S in der angegebenen Reihenfolge in die *teilweise gefüllte* Hash-Tabelle der Größe 13 (Indizes 0-12) ein. Verwenden Sie *Rehashing*, um Konflikte zu lösen. Geben Sie den vollständigen Inhalt der Hash-Tabelle nach jeder Einfüge-Operation an und markieren Sie Kollisionen in der Tabelle. Wie viele Kollisionen treten auf? Bedenken Sie, dass es für jeden Wert mehrere Kollisionen geben kann.

Schlüsseltabelle (a)

Key x	$h(x)$	$r(x)$
21		
2		
8		
17		
34		
9		

Rehashing mit $r(x)$

	-	21	2	8	17	34	9
0	13						
1	7						
2							
3							
4	28						
5							
6	16						
7							
8							
9							
10							
11							
12							

Anzahl der Kollisionen:

Fortsetzung

Lösung

a) Je 1/2P pro Wert, 1P pro Schlüssel + Anzahl Kollisionen, 1/2P Abzug, falls falsche Kollisionsanzahl.

Key x	$h(x)$	$r(x)$
21	3	2
2	4	3
8	3	4
17	8	3
34	3	5
9	5	5

Rehashing

	-	21	2	8	17	34	9
0	13	13	13	13	13	13	13
1	7	7	7	7	7	7	7
2							
3		21	21	21	21	21	21
4	28	28	28	28	28	28	28
5						34	34
6	16	16	16	16	16	16	16
7			2	2	2	2	2
8					17	17	17
9							
10							9
11				8	8	8	8
12							

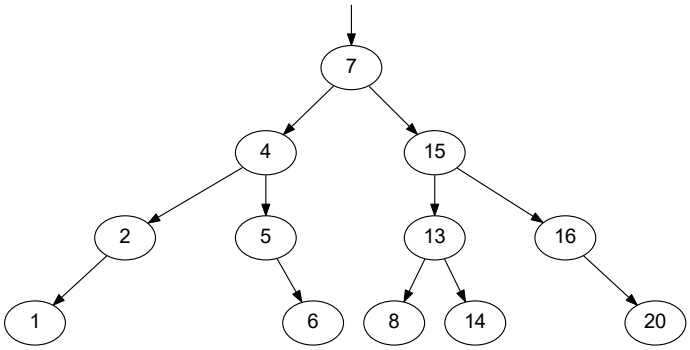
2 : 1 collisions
8 : 2 collisions
34 : 3 collisions
9 : 1 collisions
Anzahl der Kollisionen: 7 (s. o.)

Aufgabe 8 (3+2+4 Punkte)

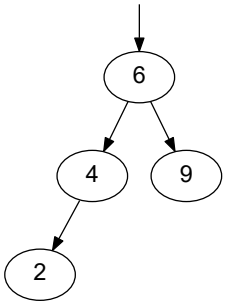
a) Betrachten Sie den **oberen** Baum. Geben Sie an, welche Knoten höhenbalanciert sind und welche Knoten mengen- bzw. größenbalanciert sind. Die Blätter müssen Sie nicht betrachten.

b) Zeichen sie je einen AVL-Baum mit minimaler und maximaler Höhe, der die Zahlen $1, 2, 3, 4, 5, 6, 7$ enthält.

c) Fügen Sie in den **unteren** AVL-Baum die angegebenen Zahlen in genau dieser Reihenfolge ein: 3, 1, 5. Geben Sie an, welche Rotationen notwendig sind. Sie finden den Baum nochmal auf der nächsten Seite

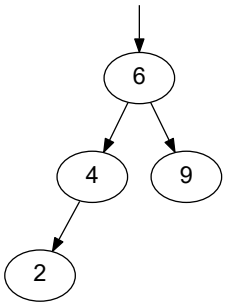


Baum für Aufgabe 8a



Baum für Aufgabe 8c

Backup-Kopie des Baums

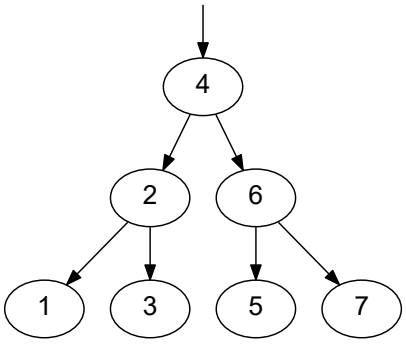


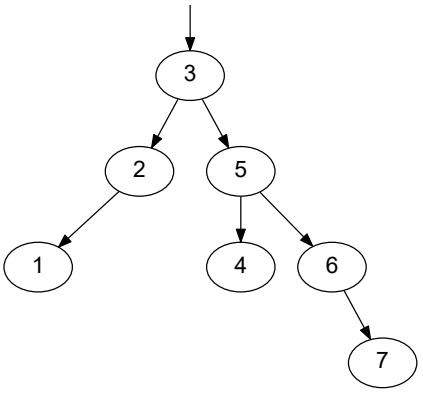
Baum für Aufgabe 8c - einzufügen: 3, 1, 5

Lösung

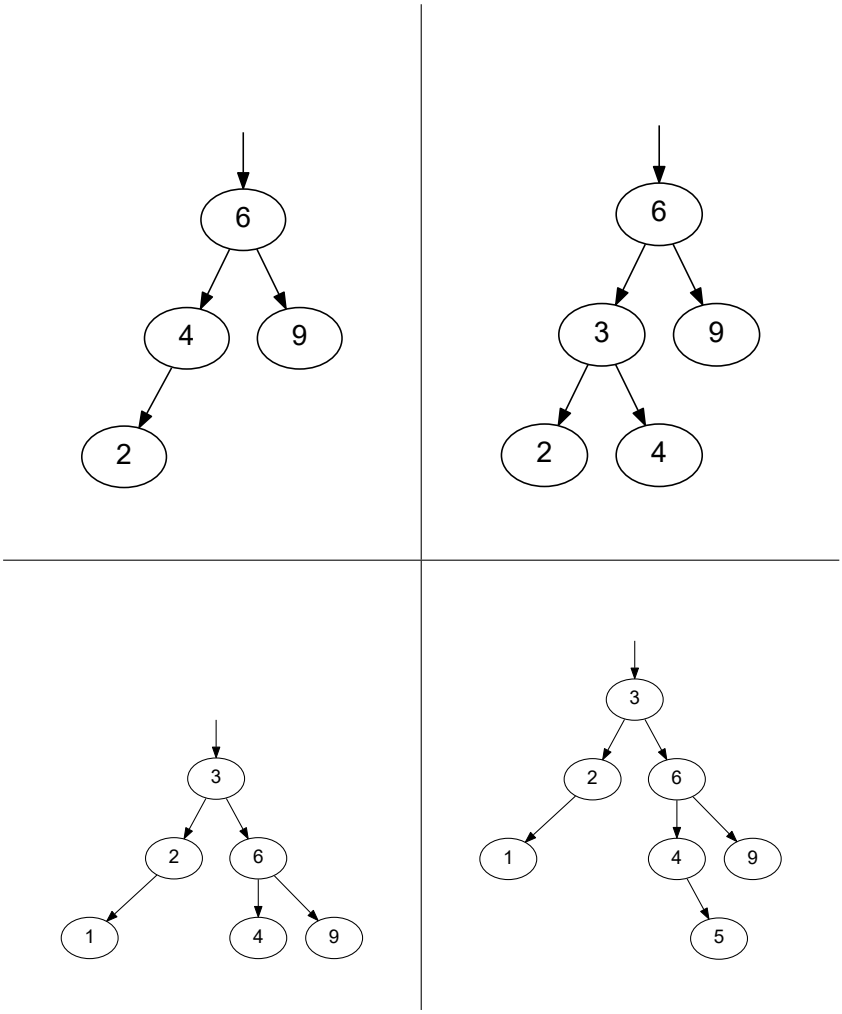
a) Mengenbalanciert: 0.5 für 4, 13. Höhenbalanciert: 0.5 für 7, 4, 15, 13.

b) Minimaler und maximaler AVL-Baum:





c) 1 Punkt pro Einfügen, 0.5 pro korrekt benannter Rotation (3: LR-Rotation um 2,4, 1: R-Rotation um 3, 6, 5: keine Rotation).


6
4
9
2
6
3
9
2
4
3
2
6
1
4
9
3
2
6
1
4
9
5

– Ende der Klausur –