Matrikelnummer:			
DHBW Duale Hochschule Baden-Württemberg Stuttgart **ÜBUNGSKLAUSUR**		Fakultät Studiengang: Jahrgang / Kurs : Studienhalbjahr:	**Technik** **Informatik** **22 A/B/C/D/IN** **2. Semester**
Datum:	**Juli 2023**	Bearbeitungszeit:	**90 Minuten**
Modul: Unit:	**T3INF1003.1** **Algorithmen**	Dozenten:	**Schulz, Dietzsch, Kochanowski, Gaugel, Diaz**
Hilfsmittel:	**Bis zu 2 read-only Hilfsmittel**		

Aufgabe	Hinweis zum Inhalt	erreichbar	erreicht
1	Sortieralgorithmen	9	
2	Heaps	14	
3	Komplexität	6	
4	Funktionen analysieren	8	
5	Pizza aufteilen	10	
6	Graphen	15	
7	Hashing und AVL	15	
Summe	77		

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 (7+1+1 Punkte)
Eine *lexikographische Ordnung* $<_{lex}$ erweitert eine Ordnung auf Zeichen zu einer Ordnung auf Worten. Dabei werden Worte Zeichen für Zeichen verglichen. Wenn eines der Worte ein echtes Anfangswort eines anderen ist, dann ist das kürzere Wort kleiner. Wir verwenden die *umgekehrte* alphabetische Ordnung als Grundlage, d.h. $a > b, b > c, \ldots$. Damit ist $<_{lex}$ im wesentlichen die Umkehrung der normale Telefonbuchordnung. Wir zählen *Schlüsselvergleiche*, d.h. der Vergleich von Jun und Jim ist für uns ein einzelner Vergleich (und $Jun < Jin$).

Betrachten Sie die Folge $S = (Mo, Al, Tim, Li, Jo, Bo, Jil, Cal)$.

a) Sortieren Sie die Folge S aufsteigend gemäß der Ordnung $<_{lex}$. Verwenden Sie das in der Vorlesung gezeigte *Selection-Sort*-Verfahren (Sortieren durch Auswahl). Geben Sie den Zustand von S nach jedem Durchlauf der äußersten Schleife an.

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

c) Wie viele *Vergleiche* von Elementen der Folge S benötigt der Algorithmus?

Lösung

a) Ein Punkt pro korrekter Iteration

Mo	Al	Tim	Li	Jo	Bo	Jil	Cal
Tim	Al	Mo	Li	Jo	Bo	Jil	Cal
Tim	Mo	Al	Li	Jo	Bo	Jil	Cal
Tim	Mo	Li	Al	Jo	Bo	Jil	Cal
Tim	Mo	Li	Jo	Al	Bo	Jil	Cal
Tim	Mo	Li	Jo	Jil	Bo	Al	Cal
Tim	Mo	Li	Jo	Jil	Cal	Al	Bo
Tim	Mo	Li	Jo	Jil	Cal	Bo	Al

b) 7 $(n-1)$

c) 28 $((n^2-n)/2)$

Aufgabe 2 (7+7 Punkte)

a) Gegeben sei die Folge

$$S = (12, 6, 24, 3, 10, 16, 25, 5, 28, 2, 23, 29, 18)$$

Betrachten Sie die Folge als potentiellen Max-Heap in einem Array und führen Sie die Operation `heapify()` durch.

- Geben Sie eine graphische Darstellung des Baums nach jedem *bubble-down* eines Wertes an.
- Geben Sie den endgültigen Zustand als Array/Zahlenfolge an.

b) Betrachten Sie nun den Heap

$$T = (27, 26, 17, 22, 21, 9, 1, 14, 19, 13, 20, 7, 8)$$

in Array-Darstellung. Sortieren Sie T mit Heapsort. Brechen Sie das Verfahren ab, sobald die 3 größten Zahlen an ihrem Platz sind (und die Heap-Eigenschaft wieder hergestellt ist). Stellen Sie nach jedem Schritt den Teil des Arrays, der den Heap repräsentiert, als Baum dar. Stellen Sie am Ende (also nach 3 abgeschlossenen Schritten) den dann aktuellen Zustand des gesamten Arrays als Zahlenfolge da.

Lösung

a) (Ein Punkt pro Baum, ein Punkt für das Array)

```
Original array
              12
       6              24
   3      10      16      25
 5  28   2  23  29  18

Bubbling down  16
              12
       6              24
   3      10      29      25
 5  28   2  23  16  18

Bubbling down  10
              12
       6              24
   3      23      29      25
 5  28   2  10  16  18

Bubbling down  3
              12
       6              24
  28      23      29      25
 5   3   2  10  16  18

Bubbling down  24
              12
       6              29
  28      23      24      25
 5   3   2  10  16  18

Bubbling down  6
              12
      28              29
   6      23      24      25
 5   3   2  10  16  18

Bubbling down  12
              29
      28              25
   6      23      24      12
 5   3   2  10  16  18

Final Array [29, 28, 25, 6, 23, 24, 12, 5, 3, 2, 10, 16, 18]
```

Lösung

b) (2 Punkte pro Schritt, 1 Punkt für das Array)

```
Original heap:
              27
      26              17
  22      21       9       1
14  19  13  20   7   8

Selected element 27 for position 12

              26
      22              17
  19      21       9       1
14   8  13  20   7  27

[26, 22, 17, 19, 21, 9, 1, 14, 8, 13, 20, 7, 27]

Selected element 26 for position 11

              22
      21              17
  19      20       9       1
14   8  13   7  26  27

[22, 21, 17, 19, 20, 9, 1, 14, 8, 13, 7, 26, 27]

Selected element 22 for position 10

              21
      20              17
  19      13       9       1
14   8   7  22  26  27

[21, 20, 17, 19, 13, 9, 1, 14, 8, 7, 22, 26, 27]
```

Array nach 3 Schritten: $(21, 20, 17, 19, 13, 9, 1, 14, 8, 7, 22, 26, 27)$

Fortsetzung

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

a) Seien $f : \mathbb{R} \to \mathbb{R}$ und $g : \mathbb{R} \to \mathbb{R}$ definiert als die Funktionen $f(x) = \sqrt[3]{x}, g(x) = log_2(x)$.

   a1) Bestimmem Sie ein beliebiges $c \in \mathbb{R}$ und $k \in \mathbb{N}$ für die gilt $c \cdot f(k) > g(k)$ und begründen Sie ihre Aussage.

   a2) Gilt allgemein $g \in \mathcal{O}(f)$? Beweisen Sie ihre Aussage.

b) Zeigen oder widerlegen Sie $(x - x \cdot 2^{-x}) \in \mathcal{O}(x)$

Lösung

a1) c beliebig mit $c \in \mathbb{R}, c > 0$ und $k = 1$, denn $c \cdot \sqrt[3]{1} = c > log_2(1) = 0$

bzw. allgemein:

$\forall i \in \mathbb{N} : k = 2^{3i}, c > \frac{3i}{2^i}$, denn

$$\begin{aligned} c \cdot f(k) &> g(k) \\ c \cdot \sqrt[3]{2^{3i}} &> log_2(2^{3i}) \\ c \cdot 2^i &> 3i \\ c &> \frac{3i}{2^i} \end{aligned}$$

oder

Es gilt $10^3 = 1000$, also $\sqrt[3]{1000} = 10$. Es gilt $2^{10} = 1024$, also $\log_2 1024 = 10$. Beide Funktionen sind für positive Eingaben streng monoton steigend.

Wähle $k = 1000, c = 1$. Dann ist $c \cdot f(x) = 10 = g(1024) > g(1000)$.

a2) $g \in \mathcal{O}(f)$ gilt. Beweis über das Grenzwert-Kriterium:

$$\begin{aligned} & \lim_{x\to\infty} \frac{\log_2(x)}{\sqrt[3]{x}} \\ = \; & \lim_{x\to\infty} \frac{\frac{1}{x \ln 2}}{\frac{1}{3\sqrt[3]{x^2}}} && \text{(l'Hopital)} \\ = \; & \lim_{x\to\infty} \frac{3\sqrt[3]{x^2}}{x \ln 2} && \text{(Kürzen)} \\ = \; & \lim_{x\to\infty} \frac{3}{\sqrt[3]{x} \ln 2} \\ = \; & 0 \in \mathbb{R} \end{aligned}$$

b) Behauptung $(x - x2^{-x}) \in \mathcal{O}(x) \to$ Beweis über das Grenzwertkriterium:

$$\begin{aligned} & \lim_{x\to\infty} \frac{x - x2^{-x}}{x} \\ = \; & \lim_{x\to\infty} (1 - 2^{-x}) && \text{(Kürzen)} \\ = \; & \lim_{x\to\infty} (1 - \tfrac{1}{2^x}) \\ = \; & \lim_{x\to\infty} 1 - \lim_{x\to\infty} \tfrac{1}{2^x} \\ = \; & 1 - 0 = 1 \in \mathbb{R} \end{aligned}$$

Fortsetzung

Aufgabe 4 (2+4+2 Punkte)

a) Betrachten Sie folgende Rekurrenzrelation: $R(n) = 4R(\frac{n}{10}) + 3n(n^2 + 4)$. Schätzen Sie (im Sinne $\mathcal{O}$ bzw. Θ) die Funktion möglichst gut ab.

b) Die unten stehende C-Funktion `hashrek()` berechnet aus einem Array-Auschnitt von `n` Elementen ab dem Index `bot` einen einzelnen Integer-Wert.

b1) Geben Sie eine Rekurrenz-Relation an, die die Laufzeit der Funktion in Abhängigeit von `n` für den Normalfall (`n` relativ groß) beschreibt. Sie können konstante Aufwände/Faktoren mit Konstanten (z.B. c_1, c_2, c_3) abschätzen.

b2) Lösen Sie die Rekurrenzrelation und bestimmen Sie so die Laufzeitkomplexität der Funktion (im Sinne $\mathcal{O}$ bzw. Θ).

```
int hashrek(int array[], int bot, int n)
{
   int top, mid, hash1, hash2, res, i, j;

   if(n == 0)
   {
      return 0;
   }
   if(n == 1)
   {
      return array[bot];
   }
   top = bot+n;
   mid = (bot+top)/2;
   hash1 = hashrek(array, bot, mid-bot);
   hash2 = hashrek(array, mid, top-mid);
   res = 0;
   for(i=bot; i<bot+n; i++)
   {
      res++;
      res += hash1*(i+1)+hash2*i;
   }
   return res;
}
```

Lösung

a) Master-Theorem: $a = 4, b = 10, d = 3$, also $b^d = 1000 > 4 = a$. Fall 1 und damit $R \in \Theta(n^3)$.

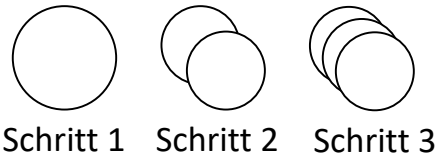
b1) $S(n) = 2S(\frac{n}{2}) + c_1 n + c_2$

b2) Master-Theorem: $a = 2, b = 2, d = 1$, also $a = b^d \Longrightarrow$ Fall 2 und $S \in \Theta(n^d \log_2 n) = \Theta(n \log n)$.

Fortsetzung

Aufgabe 5 (3+3+3+1 Punkte)
Algorithmen können auch im Alltag durchgeführt werden. Bei einer Party stellt sich Ihnen folgendes Problem: Sie möchten Pizza so aufteilen, dass jeder der g Gäste *genau* eines von s Stücken Pizza erhält. Dazu sollen Sie mehrere Algorithmen untersuchen.

Für DozentInnen und ProfessorInnen gibt es einen einfachen *Beispiel-Algorithmus*:

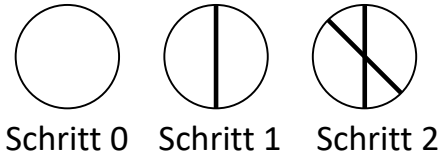


- Beginne mit 0 Pizzen.
- In jedem Schritt: Bestelle eine Pizza.
- Beende den Algorithmus, sobald es genau so viele Pizzastücke s wie Gäste g gibt. (Natürlich ist eine ganze Pizza ein Stück).

Als Studentin oder Student kommt dieser Algorithmus für Sie allerdings nicht in Frage. Sie können nur *eine* Pizza bestellen, die Sie für alle Gäste aufteilen müssen. Untersuchen Sie die folgenden drei Algorithmen und beantworten Sie jeweils die folgenden Fragen:

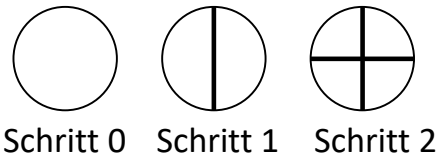
1. Berechnen Sie die Anzahl der Stücke s für die Schritte 1 bis 5. (Für den Dozenten-Algorithmus: 1, 2, 3, 4, 5)
2. Geben Sie eine Formel für die Stücke s abhängig vom Schritt i an. (Im Dozenten-Beispiel: $s(i) = i$)
3. Geben Sie eine Bedingung an, für welche Anzahl von Gästen g der Algorithmus aufgeht, so dass jeder Gast genau ein Stück bekommt und keine Stücke übrigbleiben. (Im Beispiel: $g \in \mathbf{N}$)

a) Untersuchen Sie *Pizza-Algorithmus 1* und beantworten Sie die drei o.g. Fragen.



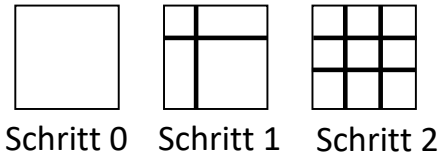
- Beginne mit einer runden Pizza.
- In jedem Schritt: Schneide die komplette Pizza einmal durch den Mittelpunkt.
- Beende den Algorithmus, sobald es gleich viele oder mehr Pizzastücke s als Gäste g gibt.

b) Untersuchen Sie *Pizza-Algorithmus 2* und beantworten Sie die drei o.g. Fragen.



- Beginne mit einer runden Pizza.
- In jedem Schritt: Schneide alle bestehenden Stücke in der Mitte durch.
- Beende den Algorithmus, sobald es gleich viele oder mehr Pizzastücke s als Gäste g gibt.

c) Untersuchen Sie *Pizza-Algorithmus 3* und beantworten Sie die drei o.g. Fragen.



- Beginne mit einer rechteckigen Party-Pizza.
- In jedem Schritt: Mache je einen horizontalen und einen vertikalen Schnitt durch die komplette Pizza.
- Beende den Algorithmus, sobald es gleich viele oder mehr Pizzastücke s als Gäste g gibt.

d) Können Sie für jede mögliche Anzahl von Gästen $g \in \mathbf{N}$ mittels mindestens einem der Algorithmen 1, 2 oder 3 (NICHT dem Beispiel-Algorithmus) eine Pizza so zerschneiden, dass jeder Gast genau ein Stück bekommt, ohne dass Stücke übrig bleiben? Begründen Sie ihre Antwort.

Lösung

a) In jedem Schritt werden zwei bestehende Stücke gleichzeitig durchgeschnitten (im ersten Schritt nur eines).

Anzahl der Stücke: 2, 4, 6, 8, 10

Formel: $s(i) = 2 * i$

Bedingung für Anzahl der Gäste: $g\%2 = 0$

b) In jedem Schritt werden aus jedem bestehenden Stück zwei Stücke mit der halben Größe gemacht.

Anzahl der Stücke: 2, 4, 8, 16, 32

Formel: $s(i) = 2^i$

Bedingung für Anzahl der Gäste: $log_2(g) \in \mathbf{N}$ oder $\exists k \in \mathbf{N} : 2^k = g$

c) In jedem Schritt wird eine horizontale und vertikale Unterteilung hinzugefügt. Es entsteht also eine Quadratzahl an Stücken:

Anzahl der Stücke: 4, 9, 16, 25, 36

Formel: $s(i) = (i + 1)^2$

Bedingung für Anzahl der Gäste: $\sqrt{g} \in \mathbf{N}$ oder $\exists k \in \mathbf{N} : k^2 = g$

d) Nein, für manche Zahlen ist es mit keinem der drei Algorithmen möglich. Gegenbeispiel: 3 Gäste.

Beweis anhand der Bedingungen:

Algorithmus 1: $3\%2 = 1 \neq 0$

Algorithmus 2: $log_2(3) \approx 1.58 \notin \mathbf{N}$

Algorithmus 3: $\sqrt{3} \approx 1.73 \notin \mathbf{N}$

Es ist allerdings möglich, so zu schneiden, dass es aufgeht, wenn mindestens ein Gast mehrere Stücke bekommen darf. (Das war aber nicht gefragt)

Fortsetzung

Aufgabe 6 (5+2+1+7 Punkte)

a) Betrachten Sie den folgenden Graph G_a:

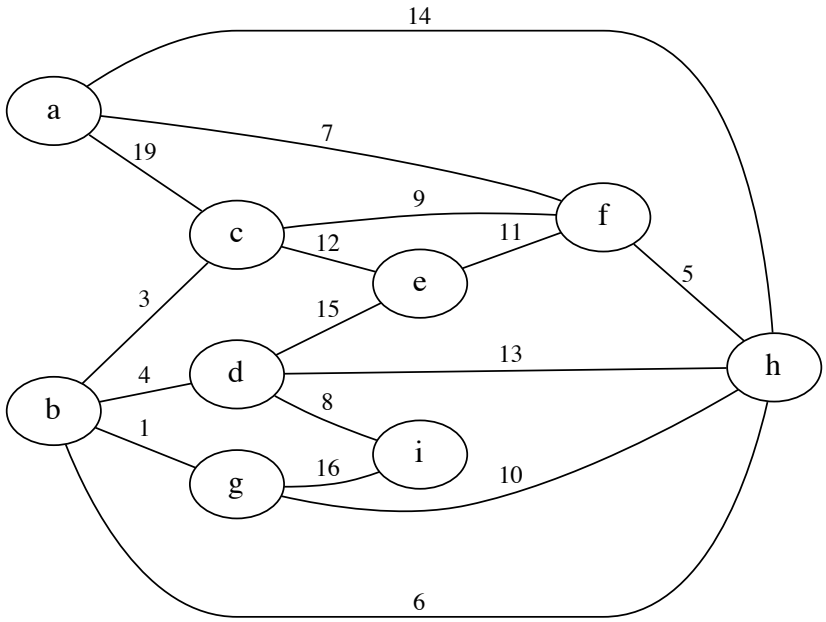


Abbildung 1: Aufgabe 6a: Graph G_a (Prim, Ausgangsknoten **d**)

Bestimmen Sie für G_a ausgehend vom Knoten **d** einen minimalen Spannbaum mit Hilfe des Prim-Algorithmus. Sie können hierzu eine Liste der verwendeten Kanten angeben. Geben Sie die Reihenfolge an, in der Sie die Knoten dem Spannbaum hinzufügen. Wie hoch ist das Gesamtgewicht der Kanten des minimalen Spannbaums?

b) Sei $G_b = (V, E)$ ein ungerichteter vollständig verbundener Graph (d.h. jeder Knoten ist mit jedem anderen Knoten direkt verbunden) mit $|V| = n$ und einer Kantengewichtsfunktion $e : E \to \mathbb{R}$, die jeder Kante das Gewicht 3 zuordnet.

b1) Wie hoch ist das Gewicht eines minimalen Spannbaums (MST) von G_b? Begründen Sie Ihr Ergebnis.

b2) Es seien $a, b \in V$ und $a \neq b$. Wie lang ist die kürzeste Route von a nach b, die Dijkstra's Algorithmus findet?

c) Siehe nächste Seite!

c) Betrachten Sie den folgenden Graph G_c:

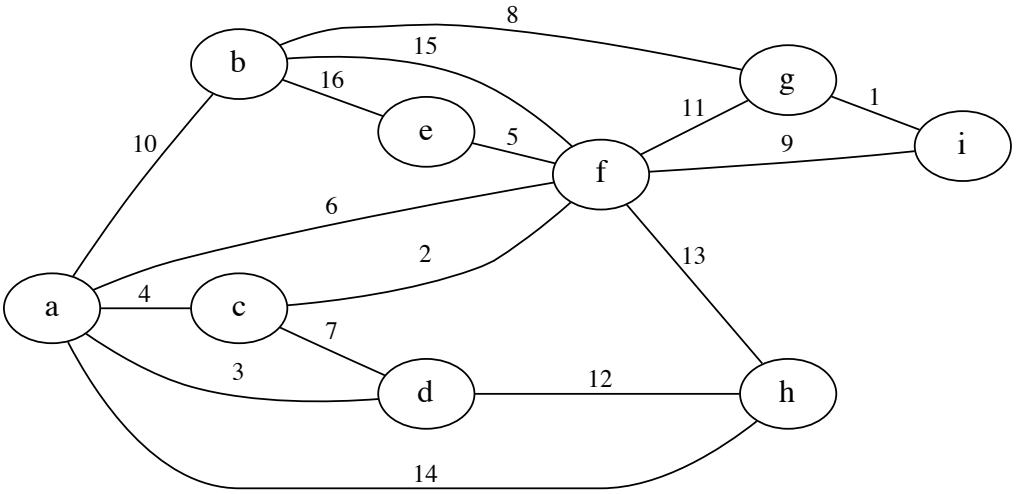
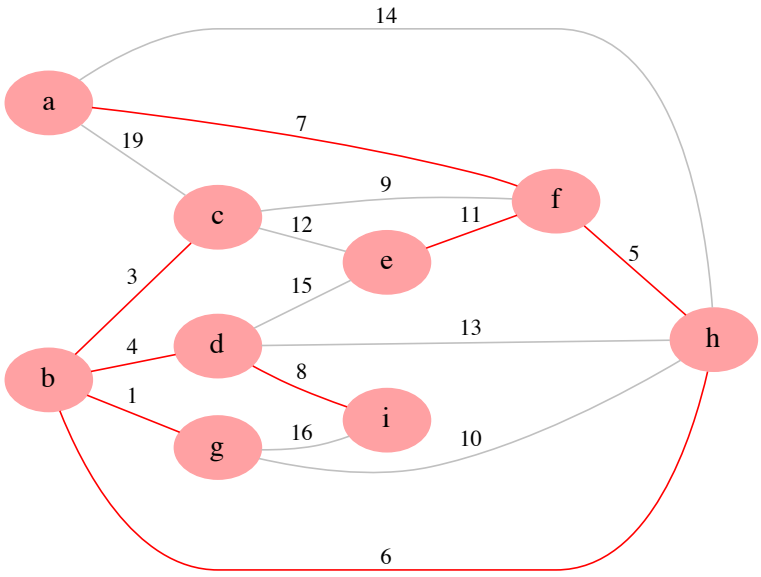


Abbildung 2: Aufgabe 6c: Graph G_c (Dijkstra, Ausgangsknoten **a**)

Verwenden Sie den Algorithmus von Dijkstra, um die minimale Entfernung aller Knoten in G_c 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).

Lösung

a) Gesamtgewicht ist 45.



Reihenfolge: (d,) b, g, c, h,f, a, i, e

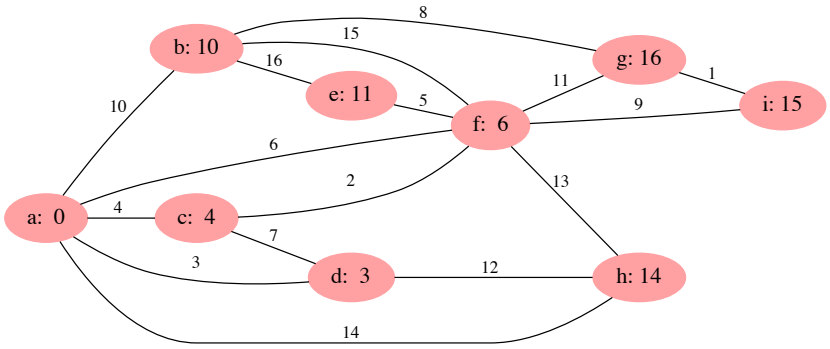
b1) Jede Kante genau mit einer anderen verbunden, Gewicht dann: $(n-1)*3$.

b2) Da alle Knoten direkt verbunden sind, besteht der Weg aus einer Kante und hat als Länge 3.

Knoten	Abstand
a	0

Lösung

c)



Knoten	Abstand
a	0
d	3
c	4
f	6
b	10
e	11
h	14
i	15
g	16

Aufgabe 7 (3+4+4+2+2 Punkte)

a) Gegeben sei die Schlüsselfolge

$$S = (21, 8, 2, 17, 13, 9)$$

und die Hash-Funktion $h(x) = (x + 3) \mod 13$.

a1) Geben Sie zunächst das Ergebnis der Hash-Funktion für jeden Schlüsselwert in S an. Verwenden Sie dafür eine Tabelle wie unten gezeigt.

a2) Fügen Sie nun die Elemente von S in der angegebenen Reihenfolge in die bereits z.T. gefüllte Hash-Tabelle der Größe 13 (Indizes 0-12) ein, die unten abgebildet ist. Verwenden Sie *linear Probing* mit Abstand 1, um Konflikte zu lösen. Geben Sie den vollständigen Inhalt der Hash-Tabelle nach jeder Einfüge-Operation an. Wie viele Kollisionen treten auf? Bedenken Sie, dass es für jeden Wert mehrere Kollisionen geben kann.

b) s. nächste Seite

c) s. nächste Seite

Schlüsseltabelle (a1)

Key x	$h(x)$
21	
8	
2	
17	
13	
9	

Linear Probing (a2)

	-	21	8	2	17	13	9
0							
1							
2	12						
3	26						
4	27						
5							
6	29						
7							
8							
9							
10							
11							
12							

Anzahl der Kollisionen:

Lösung

a) Je 1/2P pro Wert, 1/2P pro Schlüssel, 1P für Zahl der Kollisionen

a1)

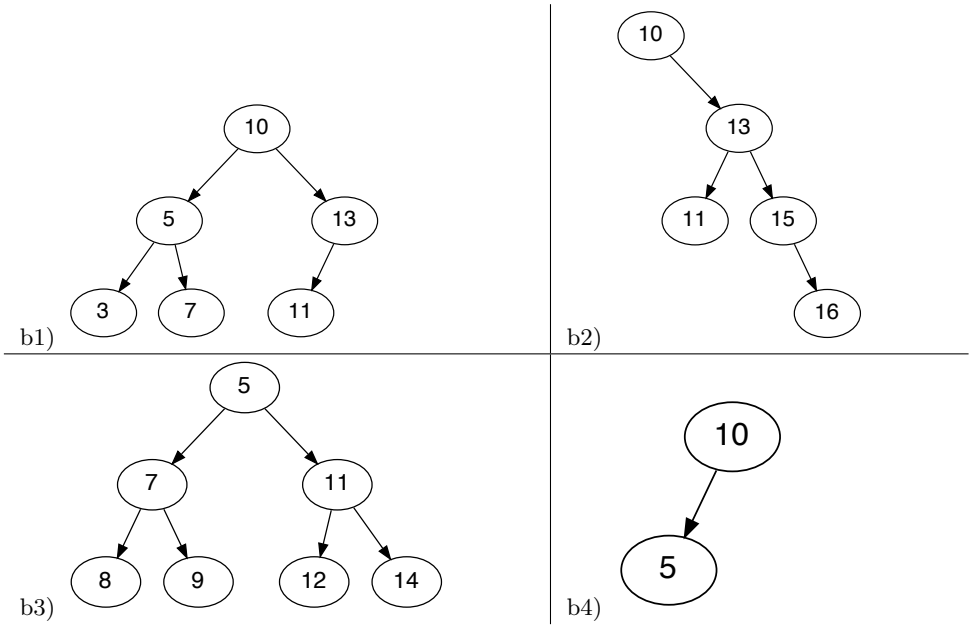
Key x	$h(x)$
21	11
8	11
2	5
17	7
13	3
9	12

Linear Probing (a2)

	-	21	8	2	17	13	9
0							9
1							
2	12	12	12	12	12	12	12
3	26	26	26	26	26	26	26
4	27	27	27	27	27	27	27
5				2	2	2	2
6	29	29	29	29	29	29	29
7					17	17	17
8						13	13
9							
10							
11		21	21	21	21	21	21
12			8	8	8	8	8

Anzahl der Kollisionen: 8*1, 13*5, 9*1=7

b) Betrachten Sie die folgenden Binärbäume. Entscheiden Sie für jeden Baum, ob er ein Max-Heap, Min-Heap, Binärer Suchbaum oder AVL-Baum ist. Als richtig gewertet werden Bäume, bei denen alle 4 Eigenschaften korrekt (mit je einem Kreuz für *ja* und einem Kreis für *nein*) markiert sind.



Baum	b1	b2	b3	b4
Max-Heap				
Min-Heap				
Binärer Suchbaum				
AVL-Baum				

Lösung

b) Je 1 Punkt pro vollständig richtiger Klassifikation

b1)	Ja	Nein
Max-Heap		X
Min-Heap		X
Binärer Suchbaum	X	
AVL-Baum	X	

b2)	Ja	Nein
Max-Heap		X
Min-Heap		X
Binärer Suchbaum	X	
AVL-Baum		X

b3)	Ja	Nein
Max-Heap		X
Min-Heap	X	
Binärer Suchbaum		X
AVL-Baum		X

b4)	Ja	Nein
Max-Heap	X	
Min-Heap		X
Binärer Suchbaum	X	
AVL-Baum	X	

c) Betrachten Sie den AVL-Baum B.

c1) Fügen Sie den Schlüssel **4** in den Baum B ein und stellen Sie die AVL-Eigenschaft gemäß dem AVL-Algorithmus wieder her. Zeichnen Sie das Ergebnis.

c2) Löschen Sie den Schlüssel **12** aus dem Baum B (nicht aus dem Ergebnis von Aufgabenteil c1) und stellen Sie die AVL-Eigenschaft gemäß dem AVL-Algorithmus wieder her. Zeichnen Sie das Ergebnis.

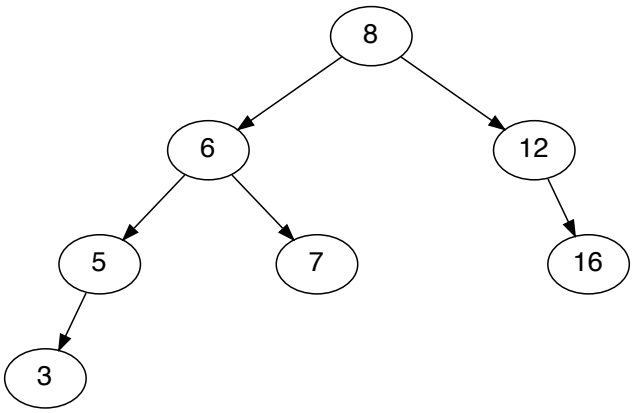
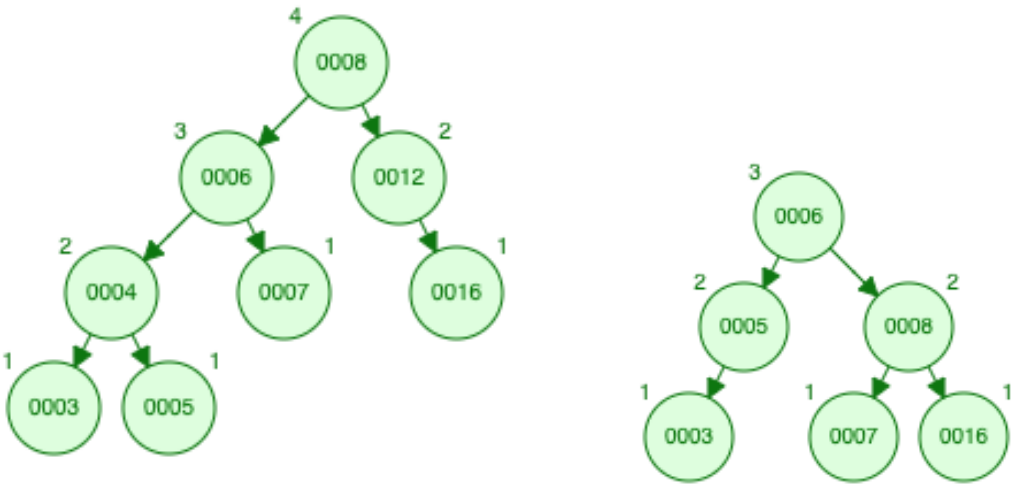


Abbildung 3: AVL-Baum B

Lösung



c1)+c2)

Fortsetzung

Fortsetzung

– Ende der Klausur –