Algorithmen und Datenstrukturen
Klausur WS 2016/17

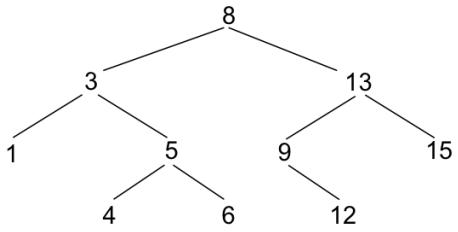
Angewandte Informatik Bachelor

Name	
Matrikelnummer	

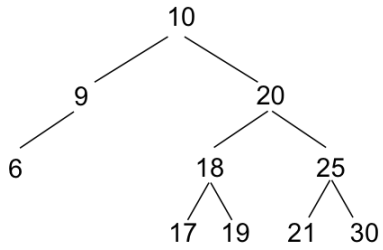
Aufgabe 1	AVL-Bäume	12	
Aufgabe 2	B-Bäume	14	
Aufgabe 3	Algorithmus von Dijkstra	16	
Aufgabe 4	Anwendung von Prioritätslisten	18	
Summe		**60**	

Aufgabe 1 AVL-Baum (12 Punkte)

a) Löschen Sie in folgendem AVL-Baum nacheinander die Zahlen 13, 8 und 9.
Hinweis: Hat der zu löschende Knoten zwei Kinder, dann ist er immer durch das Minimum im rechten Teilbaums zu ersetzen:

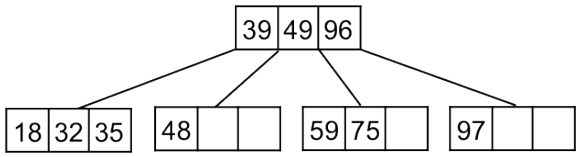


b) Fügen Sie in folgendem AVL-Baum die Zahlen 5 und 15 ein:

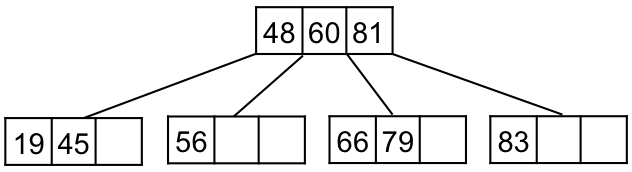


Aufgabe 2 B-Bäume (14 Punkte)

a) Fügen Sie in folgendem B-Baum (der Ordnung 4) die Schlüssel 55 und dann 95 ein.



b) Löschen Sie in folgendem B-Baum (der Ordnung 4) die Schlüssel 48, 83 und 60.



Aufgabe 3 Algorithmus von Dijkstra (16 Punkte)

Die folgende Adjazenzmatrix definiert einen gerichteten Graphen mit den Knoten 1, 2, 3, ..., 10 und Kosten als Gewichte. Bei leeren Einträgen ist keine Kante vorhanden. Bestimmen Sie mit dem Algorithmus von Dijkstra den günstigsten Weg von Startknoten 7 zu allen anderen Knoten.

	1	2	3	4	5	6	7	8	9	10
1						14		20		
2									1	
3		4								
4								2		
5								10		
6									8	
7	6		12	20						8
8										
9					1			6		
10						2				

a) Tragen Sie in folgende Tabelle nach jedem Besuchsschritt folgendes ein:
- der besuchte Knoten b
- die Kosten d[v] für den günstigsten Weg von Startknoten s = 7 nach v
- den Vorgängerknoten p[v] für den günstigsten Weg von Startknoten s = 7 nach v.

Hinweis: Es brauchen nur die p- und p-Werte eingetragen werden, die sich geändert haben. Die endgültigen p- und d-Werte können durch Umrandung besonders gekennzeichnet werden.

b	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]	d[8]	d[9]	d[10]		p[1]	p[2]	p[3]	p[4]	p[5]	p[6]	p[7]	p[8]	p[9]	p[10]

b) Geben Sie den gefundenen günstigsten Weg von 7 nach 5 an.

c) Welche Kosten hat der günstigste Weg von 7 nach 5?

Aufgabe 4 Anwendung von Prioritätslisten (18 Punkte)

MinPriorityQueue ist eine Prioritätsliste mit einer effizienten Operation delMin() bzw. getMin() zum Löschen bzw. Lesen des kleinsten Elements. Außerdem gibt es noch eine Operation add(x) zum Einfügen eines Elements x und eine Operation size() zur Ermittlung der Anzahl der Elemente in der Prioritätsliste. MinPriorityQueue bietet außerdem einen Iterator an.

Analog ist MaxPriorityQueue definiert, wobei es eine Operation delMax() bzw. getMax() zum Löschen bzw. Lesen des größten Elements gibt.

a) Schreiben Sie einen kleinen Algorithmus, der mit Hilfe einer Prioritätsliste aus einem beliebig langen Strom von Integer-Werten die N kleinsten Zahlen herausfiltert (N-Min-Problem). Ergänzen Sie dazu folgenden Pseudo-Code. Wählen Sie als Prioritätsliste entweder eine MinPriorityQueue oder eine MaxProprityQueue.

```
// Definition einer Prioritätsliste:


while (es kann eine Zahl x vom Datenstrom eingelesen werden){
     // verarbeite x:









     }
}

// Gib die N kleinsten Integerzahlen aus:
```

b) Geben Sie die Komplexität (O-Notation) Ihres Algorithmus in Abhängigkeit von der Anzahl der eingelesenen Zahlen n und der Problemgröße N an:

c) Begründen Sie kurz, warum auch mit einem AVL-Baum das N-Min-Problem genauso effizient gelöst werden kann.

d) Das N-Min-Problem soll auf einem Rechner mit beschränkten Ressourcen realisiert werden, bei dem nur statische Felder gestattet sind. Welche der beiden Datenstrukturen – AVL-Baum oder Prioritätsliste - ist geschickter? Begründen Sie Ihre Antwort.