Algorithmen und Datenstrukturen
Klausur SS 2015

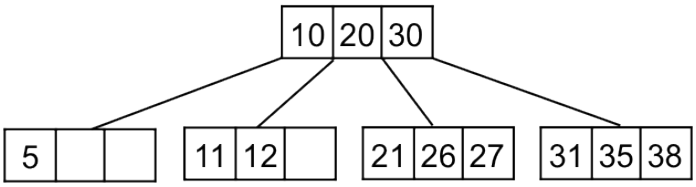
Angewandte Informatik Bachelor

Name	
Matrikelnummer	

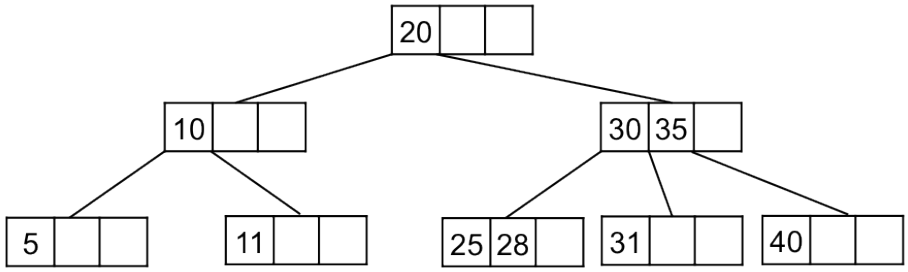
Aufgabe 1	B-Baum	15	
Aufgabe 2	Algorithmus von Dijkstra	13	
Aufgabe 3	Minimal aufspannender Baum mit Algorithmus von Kruskal	14	
Aufgabe 4	Tiefensuche und Zyklenprüfung	18	
Summe		**60**	

Aufgabe 1 B-Bäume (15 Punkte)

a) Fügen Sie in folgendem B-Baum (der Ordnung 4) die Schlüssel 32 und dann 28 ein.

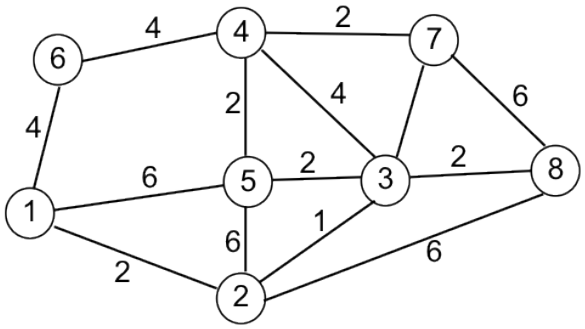


b) Löschen Sie in folgendem B-Baum (der Ordnung 4) die Schlüssel 30, 10 und dann 28.



Aufgabe 2 Algorithmus von Dijkstra (13 Punkte)

Gegeben ist ein ungerichteter Graph mit Zeitaufwand als Gewichte. Bestimmen Sie mit dem Algorithmus von Dijkstra vom Startknoten s = 6 zu allen anderen Knoten jeweils einen schnellsten Weg.



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 nach v
- den Vorgängerknoten p[v] für den günstigsten Weg von Startknoten s 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]		p[1]	p[2]	p[3]	p[4]	p[5]	p[6]	p[7]	p[8]

b) Geben Sie den gefundenen günstigsten Weg von 6 nach 8 an.

c) Welche Kosten hat der günstigste Weg von 6 nach 8?

Aufgabe 3 Minimal aufspannende Bäume (14 Punkte)

Ein gewichteter, ungerichteter Graph mit der Knotenmenge V = {1, 2, 3, 4, 5, 6, 7, 8, 9} und 18 Kanten ist durch folgende Adjazenzmatrix gegeben. Einfachheitshalber sind die Gewichte nur für eine Kantenrichtung eingetragen.

	1	2	3	4	5	6	7	8	9
1		9	3			10	4		
2				12	5	11			6
3				12	10		2		10
4								6	
5						6			4
6									5
7								13	
8									13
9									

a) Bestimmen Sie einen minimal aufspannenden Baum mit dem Algorithmus von Kruskal. Geben Sie dazu für jeden Schritt die ausgewählte Kante und das Gewicht an und skizzieren Sie die dazugehörende Union-Find-Struktur für die Knotenmenge V. Verwenden Sie den Union-By-Height-Algorithmus.

Schritt	Kante	Gewicht	Union-Find-Struktur
1			
2			
3			
4			
5			
6			
7			
8			

b) Geben Sie die Datenstruktur (Elternfeld p) für die im letzten Schritt erhaltene Union-Find-Struktur an.

Knoten	1	2	3	4	5	6	7	8	9
p									

Aufgabe 4 Tiefensuche und Zyklenerkennung (18 Punkte)

Gegeben ist ein rekursiver Tiefensuchalgorithmus, der alle Knoten in einem gerichteten Graphen besucht. Ergänzt wurde der Algorithmus um eine Datenstruktur `besuchBeendet` (im Algorithmus unterstrichen), in der jeder Knoten v gespeichert wird, dessen Besuch abgeschlossen wurde (alle Nachfolger von v wurden besucht).

a) Illustrieren Sie die Tiefensuche für den gegebenen gerichteten Graphen durch den sogenannten Tiefensuchwald (TSW). Geben Sie dazu einen Graphen an, in dem immer dann eine gerichtete Kante (v,w) eingetragen wird, wenn Knoten w von Knoten v aus besucht wird. Gehen Sie davon aus, dass in den for-Schleifen über die Knoten in numerischer Reihenfolge iteriert wird. Tragen Sie außerdem im TSW für jeden Knoten den Besuchszeitpunkt ein (1, 2, 3,...) ein.

b) Für den Fall, dass ein Knoten v zu einem bereits besuchten Knoten w führt, soll der TWS aus a) durch eine weitere Kante (v,w) ergänzt werden. Unterscheiden Sie drei Fälle:

- Vorwärtskante (v,w): es gibt im TWS bereits einen Weg von v nach w. Zeichnen Sie Vorwärtskanten in blau ein (alternativ Pfeil mit Beschriftung V)
- Rückwärtskante (v,w): es gibt im TWS bereits umgekehrt einen Weg von w nach v. Zeichnen Sie Rückwärtskanten in rot ein (alternativ Pfeil mit Beschriftung R).
- Querkante (v,w): Kanten, die weder Vorwärts- noch Rückwärtskanten sind, sind Querkanten und sollen grün eingezeichnet werden (alternativ Pfeil mit Beschriftung Q)

c) Wo enthält der Graph einen Zyklus und durch welche Kante aus b) kann er erkannt werden?

d) Wie kann man im Algorithmus mit Hilfe der Datenstruktur `besuchBeendet` einen Zyklus erkennen?

```
Set<Vertex> besucht ;
Set<Vertex> besuchBeendet;

void visitAllNodes(DirectedGraph g) {
        besucht = ∅;
        besuchBeendet = ∅;
        for (jeden Knoten v)
                if (! besucht.contains(v) )
                        visit(v, g, besucht);
}

void visit(Vertex v, DirectedGraph g) {
        besucht.add(v);    // besuche v
        for ( jeden Nachfolger w von v )
                if ( ! besucht.contains(w) ) // w nicht besucht
                        visit(w, g);
        besuchtBeendet.add(v);
}
```