Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich für Algorithmen und Datenstrukturen

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

2. Übungstest SS 2013

6. Juni 2013

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Anzahl abgegebener Zusatzblätter:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie können die Lösungen entweder direkt auf die Angabeblätter oder auf Zusatzblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

Die Verwendung von Taschenrechnern, Mobiltelefonen, PDAs, Digitalkameras, Skripten, Büchern, Mitschriften, Ausarbeitungen oder vergleichbaren Hilfsmitteln ist unzulässig.

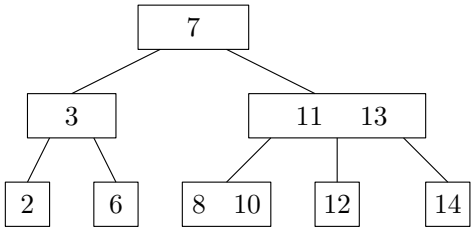
	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	16	16	18	50
Erreichte Punkte:				

Viel Erfolg!

Aufgabe 1.A: Suchbäume **(16 Punkte)**

a) (4 Punkte)

Fügen Sie den Schlüssel 9 in den folgenden **B-Baum mit Ordnung 3** ein und zeichnen Sie ihn neu.



b) (12 Punkte)

Gegeben ist ein **binärer Suchbaum** in dem jedes Element x die folgenden Einträge enthält:

$x.key$	Schlüssel von x
$x.left$	linkes Kind von x
$x.right$	rechtes Kind von x

Der Zugriff auf den Baum erfolgt über seinen Wurzelknoten *root*.

Schreiben Sie in detailliertem Pseudocode eine Methode `checkBalance(root)`, die für einen Baum mit Wurzelknoten `root` überprüft, ob es sich um einen **gültigen AVL-Baum** handelt.

Beachten Sie folgende Punkte:

- Die Laufzeit Ihres Algorithmus für einen Baum mit n Elementen muss in $\Theta(n)$ liegen.
- Die Höhe eines (Teil-)baums bzw. seine Balance sind von vornherein nicht bekannt und können über keine Funktion abgefragt werden.
- Sie können davon ausgehen, dass es sich um einen gültigen binären Suchbaum handelt.

Aufgabe 2.A: Hashtabellen **(16 Punkte)**

Fügen Sie die folgenden Zahlen in die jeweiligen Hashtabellen ein, indem Sie die angegebenen Hashfunktionen und Strategien für die Kollisionsbehandlung benutzen.

a) (4 Punkte)

Einzufügende Zahl: 15
Kollisionsbehandlung: Quadratisches Sondieren mit $c_1 = c_2 = 1$

Hashfunktion:

$$h'(k) = k \bmod 7$$

Hashtabelle:

0	1	2	3	4	5	6
	22		10	4		

b) (4 Punkte)

Einzufügende Zahl: 15
Kollisionsbehandlung: Double Hashing **ohne** der Verbesserung nach Brent

Hashfunktionen:

$$h_1(k) = k \bmod 7$$
$$h_2(k) = (k \bmod 6) + 1$$

Hashtabelle:

0	1	2	3	4	5	6
	22	16		4	19	

c) (4 Punkte)

Einzufügende Zahl: 15
Kollisionsbehandlung: Double Hashing **mit der Verbesserung nach Brent**
Wird ein bereits vorhandenes Element verschoben, so muss die neue Position dieses Elementes eindeutig gekennzeichnet werden.

Hashfunktionen:

$$h_1(k) = k \bmod 7$$
$$h_2(k) = (k \bmod 6) + 1$$

Hashtabelle:

0	1	2	3	4	5	6
	22	16		4	19	

d) (4 Punkte)

Gegeben sei eine Hashtabelle H. Zur Kollisionsbehandlung wird Double Hashing verwendet. Geben Sie zu jeder der beiden Konfigurationen an, ob sie gut funktionieren würde, oder ob es zu Problemen kommen könnte. Begründen Sie ihre Antworten.

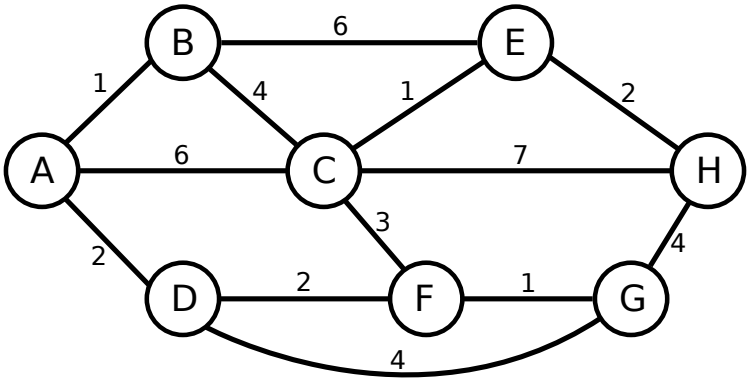
I) Tabellengröße $m = 13$
$h_1(k) = k \bmod 7$
$h_2(k) = (k \bmod 6) + 1$

II) Tabellengröße $m = 10$
$h_1(k) = k \bmod 10$
$h_2(k) = (k \bmod 7) + 1$

Aufgabe 3.A: Graphen **(18 Punkte)**

Wenden Sie den Algorithmus von *Dijkstra* auf den folgenden Graphen an um den kürzesten Weg vom Startknoten A zum Zielknoten H zu finden, Die Kantenbeschriftungen entsprechen den Distanzen zwischen den verbundenen Knoten.

- (14 Punkte) Geben Sie in jedem Schritt die Inhalte der Arrays `pred` (Vorgänger) und `dist` (Distanz) für jeden Knoten in der untenstehenden Tabelle an. Füllen Sie jede Spalte der Tabelle vollständig aus.
- (4 Punkte) Geben Sie anschließend den kürzesten Weg (die einzelnen Kanten sowie die Gesamtlänge) vom Startknoten A zum Zielknoten H an.



Schritt		A	B	C	D	E	F	G	H
0	`dist`	∞	∞	∞	∞	∞	∞	∞	∞
	`pred`	*undef*	*undef*	*undef*	*undef*	*undef*	*undef*	*undef*	*undef*
1	`dist`								
	`pred`								
2	`dist`								
	`pred`								
3	`dist`								
	`pred`								
4	`dist`								
	`pred`								
5	`dist`								
	`pred`								
6	`dist`								
	`pred`								
7	`dist`								
	`pred`								

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich für Algorithmen und Datenstrukturen

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

2. Übungstest SS 2013

6. Juni 2013

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Anzahl abgegebener Zusatzblätter:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie können die Lösungen entweder direkt auf die Angabeblätter oder auf Zusatzblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

Die Verwendung von Taschenrechnern, Mobiltelefonen, PDAs, Digitalkameras, Skripten, Büchern, Mitschriften, Ausarbeitungen oder vergleichbaren Hilfsmitteln ist unzulässig.

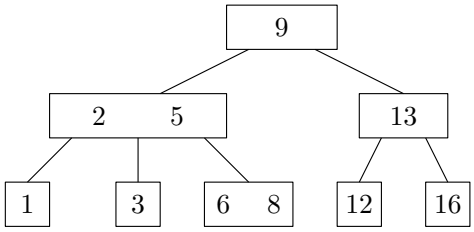
	B1:	B2:	B3:	Summe:
Erreichbare Punkte:	16	16	18	50
Erreichte Punkte:				

Viel Glück!

Aufgabe 1.B: Suchbäume **(16 Punkte)**

a) (4 Punkte)

Fügen Sie den Schlüssel 7 in den folgenden **B-Baum mit Ordnung 3** ein und zeichnen Sie ihn neu.



b) (12 Punkte)

Gegeben ist ein **binärer Suchbaum** in dem jedes Element x die folgenden Einträge enthält:

$x.key$	Schlüssel von x
$x.left$	linkes Kind von x
$x.right$	rechtes Kind von x

Der Zugriff auf den Baum erfolgt über seinen Wurzelknoten *root*.

Schreiben Sie in detailliertem Pseudocode eine Methode `checkBalance(root)`, die für einen Baum mit Wurzelknoten `root` überprüft, ob es sich um einen **gültigen AVL-Baum** handelt.

Beachten Sie folgende Punkte:

- Die Laufzeit Ihres Algorithmus für einen Baum mit n Elementen muss in $\Theta(n)$ liegen.
- Die Höhe eines (Teil-)baums bzw. seine Balance sind von vornherein nicht bekannt und können über keine Funktion abgefragt werden.
- Sie können davon ausgehen, dass es sich um einen gültigen binären Suchbaum handelt.

Aufgabe 2.B: Hashtabellen **(16 Punkte)**

Fügen Sie die folgenden Zahlen in die jeweiligen Hashtabellen ein, indem Sie die angegebenen Hashfunktionen und Strategien für die Kollisionsbehandlung benutzen.

a) (4 Punkte)

Einzufügende Zahl: 16
Kollisionsbehandlung: Quadratisches Sondieren mit $c_1 = c_2 = 1$

Hashfunktion:

$$h'(k) = k \bmod 7$$

Hashtabelle:

0	1	2	3	4	5	6
21		9		4		

b) (4 Punkte)

Einzufügende Zahl: 14
Kollisionsbehandlung: Double Hashing **ohne** der Verbesserung nach Brent

Hashfunktionen:

$$h_1(k) = k \bmod 7$$
$$h_2(k) = (k \bmod 5) + 1$$

Hashtabelle:

0	1	2	3	4	5	6
21			3	10	18	

c) (4 Punkte)

Einzufügende Zahl: 14
Kollisionsbehandlung: Double Hashing **mit der Verbesserung nach Brent**
Wird ein bereits vorhandenes Element verschoben, so muss die neue Position dieses Elementes eindeutig gekennzeichnet werden.

Hashfunktionen:

$$h_1(k) = k \bmod 7$$
$$h_2(k) = (k \bmod 5) + 1$$

Hashtabelle:

0	1	2	3	4	5	6
21			3	10	18	

d) (4 Punkte)

Gegeben sei eine Hashtabelle H. Zur Kollisionsbehandlung wird Double Hashing verwendet. Geben Sie zu jeder der beiden Konfigurationen an, ob sie gut funktionieren würde, oder ob es zu Problemen kommen könnte. Begründen Sie ihre Antworten.

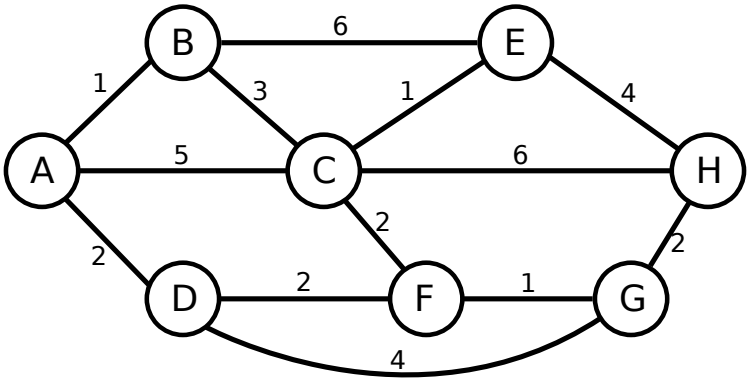
I) Tabellengröße $m = 12$
$h_1(k) = k \bmod 12$
$h_2(k) = (k \bmod 7) + 1$

II) Tabellengröße $m = 13$
$h_1(k) = k \bmod 13$
$h_2(k) = (k + 1) \bmod 6$

Aufgabe 3.B: Graphen **(18 Punkte)**

Wenden Sie den Algorithmus von *Dijkstra* auf den folgenden Graphen an um den kürzesten Weg vom Startknoten A zum Zielknoten H zu finden, Die Kantenbeschriftungen entsprechen den Distanzen zwischen den verbundenen Knoten.

- (14 Punkte) Geben Sie in jedem Schritt die Inhalte der Arrays `pred` (Vorgänger) und `dist` (Distanz) für jeden Knoten in der untenstehenden Tabelle an. Füllen Sie jede Spalte der Tabelle vollständig aus.
- (4 Punkte) Geben Sie anschließend den kürzesten Weg (die einzelnen Kanten sowie die Gesamtlänge) vom Startknoten A zum Zielknoten H an.



Schritt		A	B	C	D	E	F	G	H
0	`dist`	∞	∞	∞	∞	∞	∞	∞	∞
	`pred`	*undef*	*undef*	*undef*	*undef*	*undef*	*undef*	*undef*	*undef*
1	`dist`								
	`pred`								
2	`dist`								
	`pred`								
3	`dist`								
	`pred`								
4	`dist`								
	`pred`								
5	`dist`								
	`pred`								
6	`dist`								
	`pred`								
7	`dist`								
	`pred`								