Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

2.Übungstest SS 2016

2. Juni 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname:		Vorname:	
Matrikelnummer:		Unterschrift:	

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

Sie dürfen die Lösungen nur auf die Angabeblätter 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, Tablets, Digitalkameras, Skripten, Büchern, Mitschriften, Ausarbeitungen oder vergleichbaren Hilfsmitteln ist unzulässig.

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	16	22	12	50
Erreichte Punkte:				

Viel Erfolg!

Aufgabe A1: 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: 26
Kollisionsbehandlung: Double Hashing **ohne** die Verbesserung nach Brent

Hashfunktionen:

$h_1(k) = k \bmod 11$
$h_2(k) = (k \bmod 8) + 1$

Hashtabelle:

0	1	2	3	4	5	6	7	8	9	10
33				37		17	84	30		

b) (4 Punkte)

Einzufügende Zahl: 26
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:

$h_1(k) = k \bmod 11$
$h_2(k) = (k \bmod 5) + 1$

Hashtabelle:

0	1	2	3	4	5	6	7	8	9	10
33				37		17	84	30		

c) (8 Punkte)

Gegeben sind im Folgenden mehrere Hashverfahren. Geben Sie zu jedem an, ob es gut funktionieren würde, oder ob es zu Problemen kommen könnte. Begründen Sie ihre Antworten.

I) (2 Punkte)

Multiplikationsmethode
Tabellengröße $m = 2^{10}$
Faktor $A = \frac{\pi}{2}$

II) (2 Punkte)

Multiplikationsmethode
Tabellengröße $m = 13$
Faktor $A = 2$

III) (2 Punkte)

Divisions-Rest-Methode
Tabellengröße $m = 2^5$
$h(k) = (k + 1) \mod m$

IV) (2 Punkte)

Divisions-Rest-Methode mit Double Hashing
Tabellengröße $m = 27$
$h_1(k) = k \bmod 23$
$h_2(k) = (k \bmod 6) + 1$

Aufgabe A2: Bäume **(22 Punkte)**

a) (12 Punkte)

Gegeben sei der Wurzelknoten `root` eines (vermeintlichen) B-Baumes. Schreiben Sie eine rekursive Funktion `int checkBTree(node, leftBound, rightBound)` in detailliertem Pseudocode, die überprüft ob der Baum tatsächlich ein gültiger B-Baum ist. Die Funktion soll die Höhe des Baumes zurückgeben oder -1 falls es sich um keinen gültigen B-Baum handelt.

Sie können davon ausgehen, dass jeder Knoten für sich genommen ein korrekter Knoten ist, d.h. die Anzahl der Schlüssel/Kinder, sowie die Ordnung der Schlüssel innerhalb eines Knotens ist korrekt.

Die Parameter der Funktion sind wie folgt definiert:

- `node` ist ein Knoten des B-Baumes,
- `leftBound` und `rightBound` geben das geschlossene Intervall an, in dem sich die Schlüssel des übergebenen Knotens befinden dürfen.

Die Datenstruktur eines B-Baum-Knotens r ist wie folgt definiert:

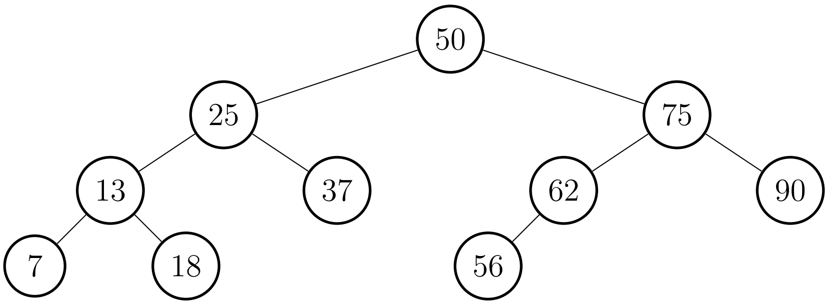
- `r.key[x]`, $1 \leq x$ — Ein Array das die Schlüssel des Knotens speichert.
- `r.child[x]`, $0 \leq x$ — Ein Array das Verweise auf die Kindknoten speichert.
- Blätter sind Knoten mit `r.key.size() == r.child.size() == 0`.

Arrays erlauben Ihnen ausschließlich die folgenden Operationen:

- `a[x]` — Indexzugriff auf das Element an Position x des Arrays a.
- `a.size()` — Gibt die Anzahl an gespeicherten Elementen im Array a zurück.

Sie können davon ausgehen, dass der B-Baum **paarweise verschiedene** ganzzahlige Schlüssel aus $[0, 100]$ speichert. Die Funktion wird mit `checkBTree(root, 0, 100)` aufgerufen.

b) (10 Punkte) Gegeben sei folgender AVL-Baum:



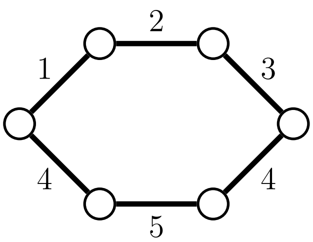
Teilen Sie alle noch nicht vorhandenen ganzen Zahlen aus dem Intervall $[0, 100]$ in Teilintervalle auf, sodass jede Zahl aus demselben Teilintervall an der gleichen Position in dem gegebenen Baum eingefügt werden würde. Tragen Sie nun diese Intervalle zusammen mit der entsprechenden Einfügestelle sowie die Art der Reorganisation, welche nach dem Einfügen einer Zahl in einem solchen Intervall ausgeführt werden muss, in die folgende Tabelle ein. Die erste Zeile in der Tabelle ist bereits exemplarisch ausgefüllt.

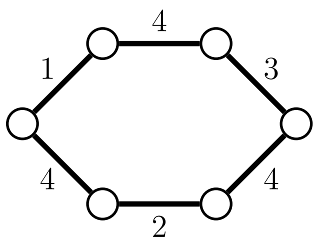
Intervall	Einfügestelle	Art der Reorganisation		
		keine	einfache Rotation	doppelte Rotation
0–6	links von 7		✓	

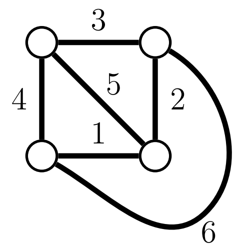
Aufgabe A3: Greedy Algorithmen **(12 Punkte)**

a) (6 Punkte)

Geben Sie zu jedem der folgenden drei gewichteten Graphen an, wieviele minimale Spannbäume der Graph hat.







Anzahl:

Anzahl:

Anzahl:

b) (6 Punkte)

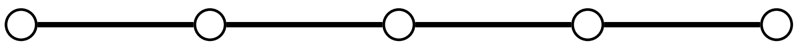
Betrachten Sie das sogenannte k-Minimum Spanning Tree (k-MST) Problem, welches wie folgt definiert ist.

Eine Instanz des k-MST Problems ist gegeben durch einen kantengewichteten zusammenhängenden Graph $G = (V, E)$ mit Kantengewichten $c(e) : E \to \mathbb{R}$ und eine natürliche Zahl k mit $k > 2$. Gesucht ist ein zusammenhängender Teilbaum von G mit genau k Knoten und minimalem Kantengewicht.

Was würde passieren, wenn Sie die bekannten Algorithmen von Prim bzw. Kruskal auf das Problem anwenden und beide Algorithmen nach $k-1$ hinzugefügten Kanten abbrechen? Zeigen Sie, dass beide Algorithmen das falsche bzw. kein optimales Ergebnis liefern, indem Sie die folgenden Gegenbeispiele vervollständigen.

I) (3 Punkte)

Geben Sie für den folgenden Graphen Kantengewichte an, sodass der modifizierte Algorithmus von Kruskal für $k = 4$ ein falsches Ergebnis liefert.



II) (3 Punkte) Geben Sie für den folgenden Graphen Kantengewichte und einen Startknoten an, sodass der modifizierte Algorithmus von Prim für $k = 4$ beginnend mit dem angegebenen Startknoten ein falsches Ergebnis liefert.