Universität Leipzig
Institut für Informatik
Abteilung Datenbanken
Prof. Dr. E. Rahm
T. Böhme, Dr. P. Richter

Ausgabe: 27.05.2002
Abgabe: 10.06.2002

Algorithmen und Datenstrukturen 2
SS 2002 - Übungsblatt 4

Hinweis: Alle *Listings* müssen in der ersten Zeile nach der Klassendefinition den *Namen des Programmautors* als Kommentar enthalten. Ohne diesen Namen werden die Listings **nicht gewertet**. Ebenfalls führt das Fehlen eines geforderten Ausdrucks der Programmausführung zur Nichtbewertung der Aufgabe.

1. Aufgabe (Hash-Verfahren) (10 Punkte)

Gegeben sei eine Hash-Tabelle mit 11 Plätzen. Die abzuspeichernden Schlüssel seien natürliche Zahlen und die Hash-Funktion h sei definiert durch:

$$h(x) = x \text{ MOD } 11$$

Fügen Sie nacheinander folgende Schlüssel in die Hashtabelle ein:
24, 34, 56, 74, 18, 76, 30, 40, 98, 62

Löschen Sie nun die Einträge:
30, 18

Fügen Sie wieder ein:
18, 30, 29

Und löschen Sie nochmals:
24, 34, 56

Lösen Sie die Kollisionen durch

a) lineares Sondieren und

b) quadratisches Sondieren mittels *(geändert!)*

$$h_i(K_p) = \left(h_0(K_p) - \left(\left\lceil \frac{i}{2} \right\rceil\right)^2 (-1)^i\right) \bmod 11 \qquad 1 \le i \le m-1$$

Achtung: wenn h_i negativ ist, dann berechnet sich die Hash-Tabellenadresse aus $11-h_i$
(der Modulo-Operator arbeitet auf einem zyklischen Bereich von 0..11)

Hinweise:
- Führen Sie das Löschen mit Auffüllen der Lücken analog zu Folie 2-20 durch

2. Aufgabe (Hashing) (4 Punkte)

In eine Hash-Tabelle sollen Schlüssel aus dem Bereich 0..9999, alle Schlüssel gleichwahrscheinlich, eingeordnet werden.

1. h(k) = k MOD 1000
2. h(k) = (k MOD 100 + [k/100]) MOD 100
3. h(k) = [k²/10] MOD 100
4. h(k) = k MOD 1019

Diskutieren Sie die Eigenschaften der genannten Hashfunktionen.

Hinweis: [x] = "größte ganze Zahl ≤ x "

3. Aufgabe (Suchaufwand in Hash-Tabellen) (10 Punkte)

Untersuchen Sie für offene Hash-Verfahren den Suchaufwand zum Auffinden eines Schlüssels unter Berücksichtigung verschiedener Hash-Funktionen, Sondierverfahren sowie unterschiedlichem Füllgrad der Hash-Tabelle.

Als Ausgangspunkt ist Ihnen die Klasse *OpenHashTableLin* gegeben, die eine Hash-Tabelle mit linearem Sondieren implementiert. Ändern Sie darin die Methode *get* derart, dass als Rückgabewert ein Integer-Objekt mit der Anzahl der Schlüsselvergleiche (Anzahl von *table[pos].key.equals(key)*) geliefert wird.

Leiten Sie dann von dieser Klasse noch die folgenden Unterklassen ab:

1. **OpenHashTableLin2**: anstatt der Standard-Hash-Funktion soll die Funktion *h(x)=[x²/100] MOD TABELLENGRÖßE* verwendet werden (Überladen der Methode *protected int h(Object key)* aus der Klasse *HashTable*)
   Hinweis: *x* bezeichnet den Wert, der bei *key.hashCode()* geliefert wird
2. **OpenHashTableQuad**: anstatt dem linearen soll quadratisches Sondieren mit der Sondierfunktion $s_i(x) = (h_0(x) + i^2)$ *MOD TABELLENGRÖßE* verwendet werden (Überladen der Methode *protected int s(int startPos, int lastPos, int i)*)
3. **OpenHashTableQuad2**: wie *OpenHashTableQuad*, jedoch mit der Hash-Funktion aus *OpenHashTableLin2* (Überladen von *h* uns *s*)

Ermitteln Sie durch Ausführen des Programms Ueb4Test den Aufwand für das Auffinden von Schlüsseln in den verschiedenen Hash-Tabellenvarianten. Das Programm wird dafür jeweils Hash-Tabellen der Größe 1019 erzeugen und nacheinander mit 500, 800, 900 und 1000 gleichverteilten Integer-Werten füllen. Für jede Konfiguration werden 50 Werte aufgesucht und das Mittel der benötigten Schlüsselvergleiche zurückgeliefert.

Diskutieren Sie die gewonnenen Ergebnisse.

Für das Übersetzen und Ausführen des Programms benötigen Sie die auf der Web-Übungsseite bereitgestellten Dateien *HTEntry.java*, *HashTable.java*, *OpenHashTableLin.java* und *Ueb4Test.java*.

Abgabe:

1. dokumentiertes Listing der geänderten Methoden der jeweiligen Klassen
2. Ausgabe eines Programmlaufs von *Ueb4Test*
3. Diskussion der Ergebnisse

4. Aufgabe (Erweiterbares Hashing) (8 Punkte)

Die Sätze mit den Schlüsseln C0, E8, E1, A3, D1, A0, C1, B8 sollen in der angegebenen Reihenfolge mit dem Verfahren des Erweiterbaren Hashing in Buckets abgespeichert werden, die jeweils 2 Sätze aufnehmen können. Den Pseudoschlüssel (8 Bit) des jeweiligen Satzes erhält man durch zeichenweises Verknüpfung der in ASCII codierten Zeichen mittels EXOR. (exklusiv-ODER). Siehe dazu das nachfolgende Beispiel. Zeichnen Sie die Belegung der Buckets und das Directory nach jeder Erhöhung der globalen Tiefe sowie nach Einfügen aller Schlüssel und geben Sie jeweils globale und lokale Tiefe an.

Achtung: Verarbeiten Sie die Bits der Pseudoschlüssel vom niederwertigsten zum höchstwertigsten (also von rechts nach links), d.h. bei Pseudoschlüssel 0010 0101 ist zuerst Bit 0 = 1, dann Bit 1 = 0, dann Bit 2 = 1 usw. zu betrachten.

Beispiel zur Berechnung des Pseudoschlüssels für den Schlüssel A3:

Zeichen	Hex	Bin
A	41	0100 0001
3	33	0011 0011
	EXOR	0111 0010