Institut für Informatik
Ludwig-Maximilians-Universität München

Prof. Dr. Hans-Peter Kriegel
Thomas Bernecker, Tobias Emrich

SS 2008
26.05.2008
Übungsblatt 7

Übungen zur Vorlesung

Effiziente Algorithmen

Aufgabe 7.1: *Lineares Hashing* **(6 + 2 Punkte)**

Gegeben sei eine Hashtabelle mit Primärseiten der Größe 2 und Überlaufseiten der Größe 1, die anfangs aus einer leeren Primärseite besteht. Die Folge der verwendeten Hashfunktionen sei h_0, h_1, . . . mit $h_i(x) = x$ **MOD** 2^i. Eine Expansion der Hashtabelle soll immer dann erfolgen, wenn der Belegungsfaktor 0.8 erreicht.

a) Verwenden Sie lineares Hashing und fügen Sie in die Hashtabelle die Schlüssel 27, 23, 22, 21, 19, 16, 15, 12, 11, 7, 3 und 2 ein. Stellen Sie den Zustand der Hashtabelle (Primärseiten, Überlaufseiten, Expansionszeiger) nach jeder Expansion und nach der letzten Einfügung graphisch dar.

b) Wie gut werden die Schlüssel auf den Adressenraum verteilt? Was ist die Ursache dieses Phänomens?

Aufgabe 7.2: *Lineares Hashing mit partiellen Erweiterungen* **(2 + 2 + 4 Punkte)**

Betrachten Sie lineares Hashing mit $n_0 = 2$ partiellen Erweiterungen. Gegeben seien

- 2N: anfängliche Größe der Hashtabelle (konstant)
- $n \in [1..n_0]$: aktuelle partielle Expansion, anfangs n = 1
- $p \in [0..N-1]$: Expansionszeiger, d. h. nächste zu expandierende Seite, anfangs p = 0
- L: Level, d. h. Anzahl der Verdoppelungen der Tabelle seit Beginn, anfangs L = 0
- $h_L(n, k)$: Folge von Hashfunktionen. $h_0(1, k) = k$ **MOD** 2N, $h_0(2, k) = k$ **MOD** 3N, $h_1(1, k) = k$ **MOD** 4N, $h_1(2, k) = k$ **MOD** 6N, $h_2(1, k) = k$ **MOD** 8N, . . .
  Beachte: Hier ist $n \in [1.. n_0 + 1]$ und $h_L(n_0+1, k) = h_{L+1}(1, k)$, z. B. $h_0(3, k) = h_1(1, k)$.

a) Geben Sie die nächsten 5 Hashfunktionen obiger Folge an! Wie ist das allgemeine Bildungsgesetz dieser Hashfunktionenfolge (d.h. geben Sie die Formel für $h_L(n, k)$ an)?

b) Geben Sie in Java-Notation einen Algorithmus (kein vollständig lauffähiges Programm) für die Suche nach einem Schlüssel k an und erläutern Sie Ihre Lösungsidee! Stützen Sie sich dabei auf das unten angegebene Klassengerüst!

c) Geben Sie in Java-Notation einen Algorithmus (kein vollständig lauffähiges Programm) für die Expansion der Hashtabelle um eine Primärseite an und erläutern Sie Ihre Lösungsidee! Stützen Sie sich dabei auf das unten angegebene Klassengerüst!

```
class HashTable {

 static final int startGroesse = 4;       // entspricht 2N
 static final int anzahlPE = 2;           // entspricht n0
 static final double schwellenwert = 0.8; // wird dieser Wert überschritten,
                                          // muß gesplittet werden

 int aktPartielleExpansion; // entspricht n
 int expansionsZeiger;      // entspricht p
 int expansionsLevel;       // entspricht L

 Vector primaerSeiten;      // Primärseiten der Hashtabelle
 ...
```

```
 // Methoden

 // Inititalisierung einer neuen Hashtabelle
 void HashTable() {/*Implementierung*/};
 // berechnet Adresse für key bei Level level
 int getHashValue(int key, int level, int aktPartExp) {/*Implementierung*/};
 // liefert den aktuellen Füllungsgrad zurück
 double getFuellungsgrad() {/*Implementierung*/};
 ...
}

class HashSeite {

 static final int bucketSize = 100;
 int[] primaerEintraege = new int[bucketSize]; // Einträge Primärseite
 SekundaerSeite sekundaerEintraege;            // erste Sekundärseite
 ... // z.B. Verwaltung der Einträge, etc.

 // Methoden

 // Erstellt eine neue Hashseite
 void HashSeite() {/*Implementierung*/};
 // gibt Anzahl Schlüssel in Primärseite + Sekundärseiten zurück
 int anzahlSchluessel() {/*Implementierung*/};
 // gibt an, ob key in Primärseite bzw. Sekundärseiten enthalten ist
 boolean durchsuche (int key) {/*Implementierung*/};
 // Löscht den Schlüssel key aus Primärseite oder Sekundärseiten
 void loesche(int key) {/*Implementierung*/};
 // Fügt den Schlüssel key in die Primärseite oder die Sekundärseiten ein
 void fuegeEin(int key) {/*Implementierung*/};
 // Liefert alle Schlüssel der Primärseite bzw. der Sekundärseiten zurück
 int[] alleSchluessel() {/*Implementierung*/};
 ... // z.B. Verwaltung der Einträge, etc.
}

class SekundaerSeite {
 int[] key;
 SekundaerSeite Nachfolger;
}
```

Aufgabe 7.3: *Perfekte Hashfunktionen* **(3 Punkte)**

Gegeben seien $n \in$ IN (n >= 1) Schlüssel und eine Tabelle mit $m \in$ IN (m >= n) Adressen. Wieviele Hashfunktionen gibt es insgesamt und wieviele davon sind perfekt?