Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Matthias Schubert
Dr. Andreas Züfle, Dr. Erich Schubert

München, 27.05.2014

Algorithmen und Datenstrukturen

SS 2014

Übungsblatt 7: Hashtabellen

Besprechung: 12.6.–17.6.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 11.6.

Aufgabe 7-1 Hashfunktionen

(a) Gegeben seien 2^8 Buckets und 4-stellige Zahlen als Schlüssel. Es sollen die Schlüssel 2040 bis 2050 einsortiert werden. Vervollständigen Sie den auf der Vorlesungswebsite verfügbaren Java-Code (`MiddleSquare.java`) mit der Hashfunktion nach der Middle-Square-Methode. Geben Sie außerdem an, welche Werte diese Hashfunktion für die gegebenen Schlüssel liefert.

```
public class MiddleSquare {
  public int hash(int key) {
    // TODO: zu implementieren
  }

  public static void main(String[] args) {
    MiddleSquare ms = new MiddleSquare();
    for(int i = 2040; i <= 2050; i++) {
      int hash = ms.hash(i);
      System.out.println("key: " + i + " - hash: " + hash);
    }
  }
}
```

(b) Vervollständigen Sie den zweiten verfügbaren Java-Code (`Division.java`) mit der Hashfunktion nach der Divisionsmethode. Verwenden Sie für die Division eine möglichst gut geeignete Primzahl (unabhängig von den gegebenen Schlüsseln!). Begründen Sie Ihre Wahl! Welche Werte liefert diese Hashfunktion für die gegebenen Schlüssel?

```
public class Division {
  int m;

  public Division(int m) {
    this.m = m;
  }

  public int hash(int key) {
    // TODO: zu implementieren
  }

  public static void main(String[] args) {
```

```
        Division d = new Division( ); // TODO: geeignete Primzahl einsetzen

        for(int i = 2040; i <= 2050; i++) {
          int hash = d.hash(i);
          System.out.println("key: " + i + " - hash: " + hash);
        }
      }
    }
```

Aufgabe 7-2 Lineares Hashing mit partiellen Erweiterungen 4+7 Punkte

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

- $2N$: anfängliche Größe der Hashtabelle (konstant)
- $n \in \{1, \ldots, n_0\}$: aktuelle partielle Expansion, anfangs $n = 1$
- $p \in \{0, \ldots, N-1\}$: 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
  $h_0(1, k) = k \mod 2N, h_0(2, k) = k \mod 3N,$
  $h_1(1, k) = k \mod 4N, h_1(2, k) = k \mod 6N,$
  $h_2(1, k) = k \mod 8N, \ldots$

  Beachte: Hier ist $n \in \{1, \ldots, n_0 + 1\}$ und $h_L(n_0 + 1, k) = h_{L+1}(1, k)$, z.B. $h_0(3, k) = h_1(1, k)$.
- Expansionsregel / Kontrollfunktion

(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, für $n_0 = 2$)?

(b) Fügen Sie in die Hashtabelle ($N = 1$, Primärseiten Größe 2, Überlaufseiten Größe 1, Expansionsregel Speicherplatzausnutzung $\alpha = 0.7$) die Schlüssel 72, 88, 91, 34, 89, 40, 65, 25, 93, 87, 85 und 29 ein. Stellen Sie den Zustand der Hashtabelle (Primärseiten, Überlaufseiten) nach jeder Expansion und nach der letzten Einfügung graphisch dar.
(Hinweis: Verwenden Sie die Speicherplatzausnutzung, nicht den Belegungsfaktor – expandieren Sie, falls die Kontrollfunktion verletzt werden würde.)