CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL
Institut für Informatik, Arbeitsgruppe Theorie der Parallelität
Prof. Dr. K. Jansen, M Kaluza, K.-M. Klein, M. Maack, M. Rau

11. Juni 2015

Präsenzaufgaben zur Vorlesung »Algorithmen und Datenstrukturen«

Blatt 10

Präsenzaufgabe 10.1 (Hashing)
Konstruieren Sie die Hashtabelle mit direkter Verkettung der Größe $m = 23$, die durch Einfügen der Elemente

```
47 17 24 70 22 01 40 45 36 59
```

mit der

1. Divisionsmethode,
2. Multiplikationsmethode mit $c = 1/2$ und
3. erweiterter Divisionsmethode mit $a = 5$ und $b = 3$

entsteht.

Präsenzaufgabe 10.2 (Hashing)
Gegeben sei eine Hashfunktion h mit der n verschiedene Schlüssel in ein Array T der Länge m eingefügt werden. Was ist die erwartete Anzahl der Kollisionen (bei einfachem universellen Hashing)? Oder genauer, was ist die erwartete Mächtigkeit von $\{\{k,l\}|k \neq l \wedge h(k) = h(l)\}$?

Präsenzaufgabe 10.3 (Hashing)
In Präsenzübung und Hausaufgabe soll eine Hash-Tabelle für Hashing mit offener Adressierung und linearer Sondierung in Pseudocode bzw Java implementiert werden. Verwenden Sie als Hash-Funktion die erweiterte Divisionsmethode. Um ein Element wieder zu entfernen, verwenden Sie eine spezielle Markierung *DELETED* um zukünftiges Suchen nicht zu behindern.

Implementieren Sie die beschriebenen Operationen *locate*, *lookup* und *delete*.

```
/**
   HashTable for keys that are strictly positive integers.
*/
public class HashTable {

    static final int DEFAULT_M=101;
    private static final int EMPTY=0;
    private static final int DELETED=-1;

    private int[] t;
    private int m;
    //Definiert die Parameter der Hashfunktion
```

```
    private final int A=13, B=3;


    //Konstruktoren
    public HashTable(){
        this(DEFAULT_M);
    }


    public HashTable(int size){
        m=size;
        t = new int[m];
        // Vorbelegung der Elemente ist 0 = EMPTY.
    }


   // Gibt den Wert der Hash-Funktion zur\"uck
    private int hash(int x){}


   //Gibt die Position eines abgespeicherten Elementes
      zur\"uck
    private int locate(int x){}


   // \"Uberpr\"ufe, ob x in der Hashtabelle vorhanden ist.
    public boolean lookup(int x){}


   // Sucht einen freien Platz f"ur ein Element x mittels
      linearer Sondierung
    private int maybelocate(int x){}


    // F\"ugt ein Element x in die Tabelle ein. Wirft eine
       Exception, wenn x bereits vorhanden ist oder kein
       freier Platz vorhanden ist.
    public void insert(int x) throws HashTableException {}


    //L\"oscht das Element x aus der Hashtabelle. Wirft eine
       Exception, wenn das Element nicht vorhanden ist.
    public void delete(int x) throws HashTableException {}
}
```