RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 5
Emmes—Langer—Reidl—Rossmanith 09.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T16

Angenommen wir versuchen uns das Leben zu erleichtern, indem wir Splaybäume nicht mit der Zig-Zig-Operation versehen. Stattdessen benutzen wir einfach zwei aufeinanderfolgende Zigs. Finden Sie ein Beispiel, bei welchem dieser Zig-Zig-lose Splaybaum für $O(n)$ Operationen $\Theta(n^2)$ Schritte benötigt. Wie verhält sich der reguläre Splaybaum auf eben diesem Beispiel?

Aufgabe T17

Beschreiben Sie einen Algorithmus, der mit Hilfe dynamischer Programmierung berechnet, wie viele verschiedene binäre Suchbäume, welche die Schlüssel $1, \ldots, n$ enthalten, existieren.

Aufgabe T18

Verwenden Sie LuFGTI-Schnipsel, um zwei Zufallszahlen a, b mit $1 \leq a \leq 6$ und $0 \leq b \leq 6$ zu bestimmen. Verwenden Sie diese persönlichen Glückszahlen, um die untenstehenden Zahlen mit der Funktion $h(x) = (ax + b) \bmod 7$ zu hashen.

x	$a \cdot x$	$a \cdot x + b$	$(a \cdot x + b) \bmod 7$
2			
4			
5			
7			
10			
12			
100			
1000			

Tragen Sie die Zahlen anschließend in eine Hashtabelle der Größe $m = 7$ ein. Verwenden Sie dabei Listen, um Kollisionen zu behandeln. Ermitteln Sie, wie viele Listenvergleiche eine (erfolglose) Suche nach dem Element 3 benötigt. Was ist der Durchschnitt in der Übungsgruppe? Was ist der Erwartungswert?

Aufgabe H12 (10 Punkte)

Implementieren Sie einen Algorithmus, der für natürliche Zahlen n, h mit $h \leq n$ bestimmt, wie viele binäre Suchbäume der Höhe h mit den Schlüsseln $1, \ldots, n$ existieren. Bestimmen Sie für $0 \leq n \leq 20$ jeweils die *durchschnittliche* Höhe und plotten Sie das Ergebnis in einer geeigneten Form oder fertigen Sie eine Tabelle an. Stellen Sie eine Vermutung über den Verlauf der so dargestellten Funktion auf.

In der Vorlesung wurde bewiesen, daß eine zufällige Einfügereihenfolge von n Schlüsseln einen Suchbaum mit erwarteter Höhe von $O(\log n)$ ergibt. Wie verträgt sich dieser Sachverhalt mit Ihrer Beobachtung?

Aufgabe H13 (10 Punkte)
Die Größe einer Hashtabelle wird gemäß folgender Strategie angepaßt:

- Falls sie mehr als 10 Elemente enthält und der Lastfaktor mindestens drei ist, wird die Hashtabelle durch eine Tabelle dreifacher Größe ersetzt.
- Falls sie mehr als 10 Elemente enthält und der Lastfaktor höchstens 1/3 ist, wird die Hashtabelle durch eine Tabelle ersetzt, deren Größe nur 1/3 der aktuellen Tabellengröße entspricht.
- Bei höchstens 10 Elementen wird eine Tabelle der Größe 30 verwendet.

Finden Sie eine geeignete Potentialfunktion, bezüglich welcher die amortisierten Kosten des Einfügens und des Löschens konstant sind. Beweisen Sie dies.

Aufgabe H14 (10 Punkte)
Aus seiner Abneigung gegenüber Listen mit doppelten Einträgen heraus hat Felix beschlossen, einen Webservice anzubieten, um eben solche zu erkennen. Der Kernalgorithmus ist (verkürzt) auf der rechten Seite abgebildet: mit Hilfe einer Hashtabelle wird nach doppelten Einträgen gesucht.

Leider hat er sich keine Gedanken über die Sicherheit des LuFGTI-Webservers gemacht. Beweisen Sie ihm, daß sein Algorithmus bereits bei einer Liste mit höchstens 1000 Elementen mehr als 400000 Vergleiche benötigen kann und damit verwundbar für eine Denial-of-Service-Attacke ist!

Beschreiben Sie ihr Vorgehen und geben Sie eventuell benutzte Programme an.

```
public static boolean hasDoublet(int n, int[] list) {
    int m = 2 * n;
    int c = (int) (Math.random() * 1000);
    ArrayList<ArrayList<Integer>> buckets =
            new ArrayList<ArrayList<Integer>>();
    for (int i = 0; i < m; ++i)
        buckets.add(new ArrayList<Integer>());
    for (int i = 0; i < n; ++i) {
        int key = hash(list[i], m, c);
        for (Integer element : buckets.get(key)) {
            if (element == list[i])
                return true;
        }
        buckets.get(key).add(list[i]);
    }
    return false;
}

public static int hash(int x, int m, int c) {
    int r = x % 23;
    r = r * r * r * r;
    return ((x % m) * (x % m) + 3 * x + r + c) % m;
}
```

Auf `http://tcs.rwth-aachen.de/lehre/DA/SS2011/` finden Sie das Programm sowohl in Java als auch in C implementiert. Es gibt praktischerweise die Anzahl der benötigten Vergleiche direkt aus.