Datenstrukturen und Algorithmen, SS 2019
Prof. Dr. P. Rossmanith
S. Dollase, J. Dreier, H. Lotze

Datum: 9.5.2019

Übungsblatt mit Lösungen 05

Aufgabe T16

In der Vorlesung wurde für (a, b)-Bäume verlangt, dass $b \geq 2a - 1$ gilt. Daher gibt es $(2, 3)$-Bäume, aber keine $(3, 4)$-Bäume.
Warum gilt diese Einschränkung? Was für Probleme könnte es beispielsweise mit $(3, 4)$-Bäumen geben?

Lösungsvorschlag

Aus der Definition von (a, b)-Bäumen wissen wir, dass die Wurzel zwischen 2 und b Kinder - hier also zwischen 2 und 4 - haben muss. Gleichzeitig verlangt die Definition, dass jeder innere Knoten außer der Wurzel zwischen a und b Knoten als Kinder hat - hier also zwischen 3 und 4.
Wir betrachten nun den Fall, dass wir genau fünf Schlüssel einfügen möchten - dies sollte in jedem (a, b)-Baum möglich sein. Da die Wurzel zwischen 2 und b Kinder hat, muss mindestens eines dieser Kinder ein innerer Knoten sein - wir können nicht alle 5 Schlüssel als Blätter an die Wurzel anhängen. Da alle Blätter den gleichen Abstand zur Wurzel haben müssen, folgt daraus, dass jedes Kind der Wurzel ein innerer Knoten sein muss. Da wiederum jeder innere Knoten zwischen 3 und 4 Kinder haben muss, wir also mindestens 6 Kinder bräuchten, jedoch nur 5 Elemente zur Verfügung haben, ist es damit nicht möglich, genau 5 Knoten in einen $(3, 4)$-Baum einzufügen.

Aufgabe T17

Wir betrachten die Hashfunktion $h\colon \Sigma^* \to \{0, \ldots, m-1\},\ c_1 c_2 \ldots c_k \mapsto \left(\sum_{i=1}^{k} c_i\right) \bmod m$, welche die Menge Σ^* aller Wörter (im ASCII-Alphabet) auf eine Zahl zwischen 0 und $m - 1$ abbildet.

a) Ist h ein gute Hashfunktion? Überlegen Sie sich realistische Anwendungsbeispiele, für welche h sehr viele Kollisionen erzeugt.

b) Wie könnte eine vernünftige Hashfunktion für Zeichenketten aussehen, für welche Sie unter normalen Umständen ein gutes Verhalten erwarten würden? Wie gehen Sie mit dem Fall um, dass m sehr groß ist?

Lösungsvorschlag

a) Wenn wir Wörter hashen, die alle Permutationen voneinander sind, dann werden *alle* auf denselben Wert abgebildet – schlechter geht es wirklich nicht. Außerdem werden alle kurzen Wörter auf relativ kleine Zahlen abgebildet, so dass nur ein verschwindend kleiner Teil einer sehr großen Hashtabelle verwendet würde. Dort gäbe es dann sehr viele Kollisionen.

b) Eine einfache Möglichkeit ist eine Linearkombination der c_i mit geeigneten Koeffizienten, die mit steigendem i auch größer werden.

Aufgabe T18

Wir verwenden jetzt folgende universelle Familie von Hashfunktionen:

$$\{ h_{a,b} \mid 1 \leq a < 5,\ 0 \leq b < 5 \}$$

mit $h_{a,b}(x) = ((ax + b) \bmod 5) \bmod 4$.

Berechnen Sie die Wahrscheinlichkeit, dass $h(2) = h(3)$ ist, falls wir h zufällig aus obiger Familie von Funktionen wählen. Was müsste herauskommen, wenn man bedenkt, dass es sich um eine universelle Familie von Hashfunktionen handelt?

Da es sich um viele Funktionen handelt, sollte die Arbeit unter vielen Personen verteilt werden, damit es schneller geht. Wie lange brauchen Sie gemeinsam, um diese Wahrscheinlichkeit ohne Taschenrechner oder ähnlichem zu berechnen?

Lösungsvorschlag

Es gibt insgesamt vier Möglichkeiten für a und fünf Möglichkeiten für b, so dass wir 20 verschiedene $h_{a,b}$ in unserer Familie finden. Für jede dieser Funktionen müssen wir nun herausfinden, ob $h_{a,b}(2) = h_{a,b}(3)$ gilt. Wir berechnen also einfache beide benötigte Funktionswerte für alle 20 Funktionen:

a	b	$h_{a,b}(2)$	$h_{a,b}(3)$
1	0	2	3
1	1	3	0
1	2	0	0
1	3	0	1
1	4	1	2
2	0	0	1
2	1	0	2
2	2	1	3
2	3	2	0
2	4	3	0
3	0	1	0
3	1	2	0
3	2	3	1
3	3	0	2
3	4	0	3
4	0	3	2
4	1	0	3
4	2	0	0
4	3	1	0
4	4	2	1

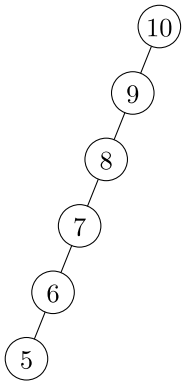
Wir sehen, dass es genau zwei Kollisionen gibt. Da es 20 verschiedene Funktionen sind, ist die Wahrscheinlichkeit einer Kollision also genau 1/10. Aus der Theorie der universellen Hashfunktionen erwarten wir, dass die Kollisionswahrscheinlichkeit höchstens $1/m = 1/4$ beträgt. Mit 1/10 wird dieser garantierte Wert sogar noch unterboten.

Aufgabe T19

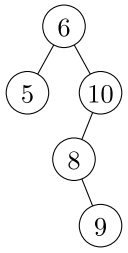
Fügen Sie in einen leeren Splaybaum die Schlüssel 5,. . . ,10 in dieser Reihenfolge ein. Löschen Sie dann die 7 und anschließend die 8. Fügen Sie danach die 8 wieder ein.

Lösungsvorschlag

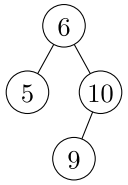
Nachdem ein Wert eingefügt wird, wird er in die Wurzel gesplayt. Somit ergibt sich der folgende Baum.



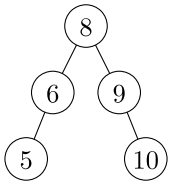
Um die sieben zu löschen gehen wird wie folgt vor: Zunächst wird 7 in die Wurzel gesplayt, dann wird das größte linke Kind, die 6, nach oben gesplayt. Die 7 hat nun nur ein einziges Kind und wird einfach gelöscht. Dies ergibt diesen Baum.



Das Löschen der 8 funktioniert ähnlich: Die 8 wird in die Wurzel gesplayt. Das größte linke Kind, die 6, nach oben gesplayt. Die 8 wird gelöscht.



Die 8 wird zunächst als linkes Kind der 9 eingefügt und dann hochgesplayt.



Aufgabe H13 (5+1+3 Punkte)

a) Erstellen Sie mithilfe eines Programms eine Tabelle, welche die Wahrscheinlichkeiten von $h(x) = h(y)$ für alle $0 \leq x < 5, 0 \leq y < 5$ enthält, falls h wieder zufällig aus der Familie von T18 gezogen wird.

b) Ist das Ergebnis das, was Sie erwarten? Warum?

c) Wiederholen Sie das Experiment für $0 \leq x, y < 6$, $1 \leq a < 6$, $0 \leq b < 6$ und $h_{a,b}(x) = ((ax + b) \bmod 6) \bmod 4$. Kommentieren Sie das Ergebnis. Nehmen Sie insbesondere dazu Stellung, ob es sich auch jetzt um eine universelle Familie von Hashfunktionen handelt.

```
#include ⟨stdio.h⟩

int p;
int h(int a, int b, int x) {
  return((a * x + b)%p)%4;
}
int countcollisions(int x, int y) {
  int a, b, count = 0;
  for(a = 1; a < p; a++) {
    for(b = 0; b < p; b++) {
      if(h(a, b, x) == h(a, b, y)) {
        count++;
      }
    }
  }
  return count;
}
void printtable() {
  int x, y;
  for(x = 0; x < p; x++) {
    for(y = 0; y < p; y++) {
      printf("%d ", countcollisions(x, y));
    }
    printf("\n");
  }
}
int main() {
  p = 5;
  printf("H13a\n");
  printtable();
  p = 6;
  printf("H13b\n");
  printtable();
}
```

Abbildung 1: Programm für Aufgabe H13

Lösungsvorschlag

Das Programm in Abbildung 1 erledigt die Arbeit für $p = 5$ und $p = 6$, wobei es nicht die Wahrscheinlichkeiten, sondern die Gesamtzahl der Kollisionen ausgibt. Um Wahrscheinlichkeiten zu erhalten, muss man noch durch $p(p-1)$ teilen, was aber hässliche Brüche ergibt.

Die Ausgabe des Programms ist:

```
H13a
20 2 2 2 2
2 20 2 2 2
2 2 20 2 2
2 2 2 20 2
2 2 2 2 20
H13b
30 4 14 12 14 4
4 30 4 14 12 14
14 4 30 4 14 12
```

```
12 14 4 30 4 14
14 12 14 4 30 4
4 14 12 14 4 30
```

Natürlich sind die Kollisionswahrscheinlichkeiten stets 1, wenn $x = y$. Ansonsten sind sie für $p = 5$ stets $1/10$. Wir erwarten natürlich, dass sie höchstens $1/4$ sind. Für $p = 6$ sieht es bitterer aus, da hier Kollisionswahrscheinlichkeiten von bis zu $7/15$ auftreten, was deutlich höher als $1/m = 1/4$ ist. Allerdings ist das Theorem aus der Vorlesung dadurch nicht widerlegt, weil ja 6 keine Primzahl ist. Ganz im Gegenteil: Dieses Beispiel demonstriert, wie wichtig es ist, eine Primzahl zu wählen.

Aufgabe H14 (2+8+2 Punkte)
Am Ende dieser Aufgabe und auf der Webseite finden Sie ein Programm, das eine Datei lesen kann, welche je Zeile ein Wort enthalten soll. Das Programm zählt, wie viele verschiedene Wörter es in der Datei gibt, wobei es eine Hashtabelle verwendet.

a) Wenn Sie das Programm auf einer typischen Datei laufen lassen, wie lang ist die Laufzeit auf Ihrem Computer typischerweise für eine Datei mit n Bytes?

b) Erzeugen Sie eine Eingabedatei, deren Größe 1MB nicht überschreitet und deren Bearbeitung länger als 5 Sekunden dauert. Machen Sie die Datei Ihrem Tutor zugänglich. Erklären Sie, wie sie die Datei erstellt haben.

c) Könnten Sie solch eine furchtbare Datei auch dann erstellen, wenn im Programm h nicht mit x sondern mit y multipliziert würde? (Eine sehr kurze Antwort reicht hier.)

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;

public class WordCount {
    static int x = 37;
    static int y = Math.abs((new Random()).nextInt());

    public static int hash(String w) {
        int h = 0;
        for(int i = 0; i < w.length(); i++) {
            h = h * x + w.charAt(i);
        }
        return Math.abs(h);
    }

    public static void main(String args[]) throws IOException {
        String filename = args[0];
        List<String> words = Files.readAllLines(Paths.get(filename));
        int m = 100000;
        long count = 0;
        ArrayList<LinkedList<String>> slot = new ArrayList<>();
        for(int i = 0; i < m; i++) {
            slot.add(new LinkedList<String>());
        }
        for(String w : words) {
            LinkedList<String> list = slot.get(hash(w)%m);
            if(!list.contains(w)) {
                count = count + 1;
                list.add(w);
            }
        }
        System.out.println(count + " different words in " + filename + ".");
    }
}
```

Lösungsvorschlag

a) Falls wir das Programm auf zufällig generierten Eingaben laufen lassen, können wir beobachten, dass es sehr schnell ist: Wir haben uns dafür einige Listen verschiedener Längen mit zufällig generierten, 10 Zeichen langen Worten erstellt. Wir bermerken, dass wir etwa 0.1 s/Mb auf unserem Rechner benötigen. Offensichtlich benötigt das Programm lineare Laufzeit in der Anzahl der Wörter einer Datei, wir können jedoch feststellen, dass die Einfügeoperationen in etwa konstante Zeit benötigen. Wenn wir die Anzahl der eingefügten Wörter gegen unendlich laufen lassen, wird die Zeit für das Einfügen in die konstant große Hashtabelle aufgrund der vielen Kollisionen jedoch gegen $O(n^2)$ konvergieren.

b) Um eine lange Laufzeit zu erzwingen, wollen wir viele Wörter mit dem gleichen Hash einfügen, welche dann alle in einer einfachen Liste gespeichert werden. Dies führt zu einer quadratischen Laufzeit.

   Folgendes Programm zählt Dezimalzahlen auf und speichert alle, die einen Hashwert von 115 haben.

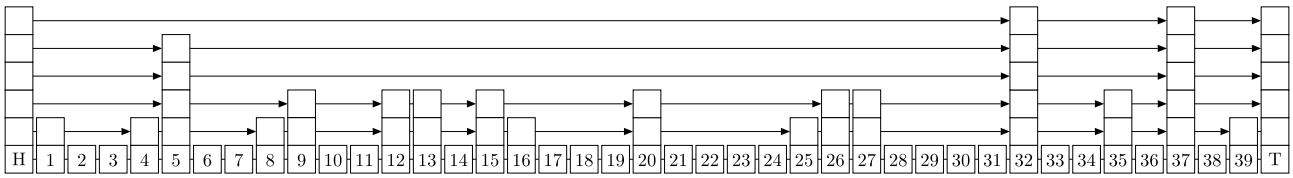
```
import java.util.*;

public class BadWords {

  public static int hash(String w) {
    int h = 0;
    for(int i = 0; i < w.length(); i++) {
      h = h * 37 + w.charAt(i);
    }
    return Math.abs(h);
  }

  public static void main(String args[]) {
    Random gen = new Random();
    long l = 0;
    int len = 0;
    while(len < 900000) {
      String w = "" + l;
      int h = hash(w)%100000;
      l = l + 1;
      if(h ≠ 115) continue;
      System.out.println(w);
      len += w.length() + 1;
    }
  }
}
```

Henri Lotzes Computer braucht für die von obigem Programm erzeugte Datei immerhin 8.3 Sekunden (i7-6700K), während Peter Rossmaniths 64 Sekunden benötigt (Celeron N2807).

c) Nein können wir nicht, da wir dann keine Chance haben vorher zu wissen auf welchen Hash ein Wort abgebildet wird.

Aufgabe H15 (3+3+3 Punkte)

Geben sie jeden Schritt im Detail an, wenn wir in folgender Skiplist nach 22, 40 und 0 suchen.



Lösungsvorschlag
Suche nach 22:

- Starte an H in oberster Liste (Ebene 5)
- $22 < 32 \Rightarrow$ Wechsel an H zur nächst tieferen Liste (Ebene 4)
- $22 > 5 \Rightarrow$ Wechsel zur 5 (Ebene 4)
- $22 < 32 \Rightarrow$ Wechsel an 5 zur nächst tieferen Liste (Ebene 3)
- $22 < 32 \Rightarrow$ Wechsel an 5 zur nächst tieferen Liste (Ebene 2)
- $22 > 9 \Rightarrow$ Wechsel zur 9 (Ebene 2)
- $22 > 12 \Rightarrow$ Wechsel zur 12 (Ebene 2)
- $22 > 13 \Rightarrow$ Wechsel zur 13 (Ebene 2)
- $22 > 15 \Rightarrow$ Wechsel zur 15 (Ebene 2)
- $22 > 20 \Rightarrow$ Wechsel zur 20 (Ebene 2)
- $22 < 26 \Rightarrow$ Wechsel an 20 zur nächst tieferen Liste (Ebene 1)
- $22 < 25 \Rightarrow$ Wechsel an 20 zur nächst tieferen Liste (Ebene 0)
- $22 > 21 \Rightarrow$ Wechsel zur 21 (Ebene 0)
- $22 = 22 \Rightarrow$ 22 gefunden nach 13 Vergleichen

Suche nach 40:

- Starte an H in oberster Liste (Ebene 5)
- $40 > 32 \Rightarrow$ Wechsel zur 32 (Ebene 5)
- $40 > 37 \Rightarrow$ Wechsel zur 37 (Ebene 5)
- Nächstes Element ist $T \Rightarrow$ Wechsel an 37 zur nächst tieferen Liste (Ebene 4)
- Nächstes Element ist $T \Rightarrow$ Wechsel an 37 zur nächst tieferen Liste (Ebene 3)
- Nächstes Element ist $T \Rightarrow$ Wechsel an 37 zur nächst tieferen Liste (Ebene 2)
- Nächstes Element ist $T \Rightarrow$ Wechsel an 37 zur nächst tieferen Liste (Ebene 1)
- $40 > 39 \Rightarrow$ Wechsel zur 39 (Ebene 1)
- Nächstes Element ist $T \Rightarrow$ Wechsel an 39 zur nächst tieferen Liste (Ebene 0)
- Nächstes Element ist $T \Rightarrow$ Suche endet erfolglos nach 9 Vergleichen

Suche nach 0:

- Starte an H in oberster Liste (Ebene 5)
- $0 < 32 \Rightarrow$ Wechsel an H zur nächst tieferen Liste (Ebene 4)
- $0 < 5 \Rightarrow$ Wechsel an H zur nächst tieferen Liste (Ebene 3)

- $0 < 5 \Rightarrow$ Wechsel an H zur nächst tieferen Liste (Ebene 2)
- $0 < 5 \Rightarrow$ Wechsel an H zur nächst tieferen Liste (Ebene 1)
- $0 < 1 \Rightarrow$ Wechsel an H zur nächst tieferen Liste (Ebene 0)
- $0 < 1 \Rightarrow$ Suche endet erfolglos nach 6 Vergleichen