Tutorium
Grundlagen: Algorithmen und Datenstrukturen

Übungsblatt Woche 9

Aufgabe 9.1
Hashing mit Chaining

Aufgabe 9.1

Veranschaulichen Sie Hashing mit Chaining. Die Größe m der Hash-Tabelle ist in den folgenden Beispielen jeweils die Primzahl 11. Die folgenden Operationen sollen nacheinander ausgeführt werden.

```
insert  3, 11, 9, 7, 14, 56, 4, 12, 15, 8, 1
delete  56
insert  25
```

Der Einfachheit halber sollen die Schlüssel der Elemente die Elemente selbst sein.

a) Verwenden Sie zunächst die Hashfunktion: $g(x) = 5x \mod m$

b) Berechnen Sie die Hashwerte unter Verwendung der Hashfunktion $h(x) = a \cdot x' \mod m$ nach dem aus der Vorlesung bekannten Verfahren für einfache universelle Hashfunktionen, wobei $a = (7, 5)$ und $x' = \left(\left\lfloor \frac{x}{2^w} \right\rfloor mod\ 2^w, x\ mod\ 2^w\right)$ für $w = \lfloor \log_2 m \rfloor = \lfloor 3.45 \ldots \rfloor = 3$ gilt und der Ausdruck $a \cdot x'$ ein Skalarprodukt bezeichnet.

Aufgabe 9.1 (a)

$$g(x) = 5x \mod m \qquad m = 11$$

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$												

Aufgabe 9.1 (a)

$$g(x) = 5x \mod m \qquad m = 11$$

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$	5	4	9	2	7	1	0	5	4	9	4	5

Aufgabe 9.1 (a)

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$	5	4	9	2	7	1	0	5	4	9	4	5

1.) insert(3)

0	1	2	3	4	5	6	7	8	9	10

2.) insert(11)

0	1	2	3	4	5	6	7	8	9	10

3.) insert(9)

0	1	2	3	4	5	6	7	8	9	10

4.) insert(7)

0	1	2	3	4	5	6	7	8	9	10

5.) insert(14)

0	1	2	3	4	5	6	7	8	9	10

6.) insert(56)

0	1	2	3	4	5	6	7	8	9	10

Aufgabe 9.1 (a)

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$	5	4	9	2	7	1	0	5	4	9	4	5

7.) insert(4)

0	1	2	3	4	5	6	7	8	9	10

8.) insert(12)

0	1	2	3	4	5	6	7	8	9	10

9.) insert(15)

0	1	2	3	4	5	6	7	8	9	10

10.) insert(8)

0	1	2	3	4	5	6	7	8	9	10

11.) insert(1)

0	1	2	3	4	5	6	7	8	9	10

12.) delete(56)

0	1	2	3	4	5	6	7	8	9	10

Aufgabe 9.1 (a)

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$	5	4	9	2	7	1	0	5	4	9	4	5

13.) insert(25)

0	1	2	3	4	5	6	7	8	9	10

Aufgabe 9.1 (b)

$$h(x) = a \cdot x' \mod m$$

$$m = 11 \qquad a = (7, 5) \qquad x' = \left(\left\lfloor \frac{x}{2^w} \right\rfloor \mod 2^w, x \mod 2^w\right)$$

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$												

Aufgabe 9.2
B-Baum Analyse

Aufgabe 9.2

Der Aufwand nach dem Einfugen oder Entfernen einen perfekt balancierten Binärbaum wiederherzustellen ist $\mathcal{O}(n)$ im Worstcase. Daher versuchen AVL-Bäume keinen perfekt balancierten Binärbaum zu erreichen sondern stellen lediglich einen möglichst gut balancierten Binärbaum sicher. Die Tiefe eines AVL-Baums ist dabei im Worstcase $1.44 * \log_2 n$ und bei einem Rot-Schwarz-Baum $2 * \log_2 n$. Diese Tiefe ist bei Binärbäumen auch die maximale Anzahl an notwendigen Vergleichen z.B. beim Suchen eines Werts. Wir wollen nun so eine Analyse für einen B-Baum machen. B-Bäume sind eng verwandt mit den AB-Bäumen. Ein *k*-Baum ist dasselbe wie ein *k*,2*k*-Baum, bei einem B-Baum ist also die Obergrenze für die Anzahl der Kinder eines Knoten genau das Doppelte.

a) Wie ist ein Baum aufgebaut, der die **größtmögliche Tiefe mit der kleinstmöglichen Anzahl an Elementen** erreicht? Schätzen Sie die notwendigen Vergleiche im Worstcase für eine Suche nach einem Wert in diesem Baum ab. **Hinweis**: Es müssen dabei nicht alle Werte aus einem Knoten betrachtet werden.

b) Wie sieht der Baum aus, der bei der Suche nach einem Wert im Worstcase **möglichst viele Vergleiche** im Verhältnis zu der Anzahl von Elementen im Baum benötigt? Wie viele Vergleiche sind nun im Worstcase schätzungsweise nötig?

c) Wie verhält sich die notwendige Anzahl an Vergleichen bei wachsenden *k*? Erkläre warum sich die Anzahl der Vergleiche so verhält.

Aufgabe 9.2 (a)

Gesucht: Größtmögliche Tiefe mit möglichst wenig Elementen

Aufgabe 9.2 (b)

Gesucht: Möglichst viele Vergleiche im Verhältnis zur Elementzahl

Aufgabe 9.2 (c)

Gesucht: Verhalten der notwendigen Vergleichsanzahl bei wachsendem *k.* Begründung?

Aufgabe 9.3 Klausurvorbereitung: Transfer

Aufgabe 9.3

Diese Aufgabe zeigt mögliche Aufgabenstellungen für die Klausur. Es geht darum zu zeigen, dass Sie die Theorie hinter dem Stoff verstanden haben und basierend darauf neue Problemstellungen lösen können.

a) Wie viele verschiedene Baumstrukturen sind mit 11 Elementen bei einem 3,5-Baum möglich?

b) Wie viele verschiedene Baumstrukturen sind bei einem 3-Baum (3,6-Baum) mit 3 Ebenen möglich?

Aufgabe 9.3 (a)

Wie viele verschiedene Baumstrukturen sind mit 11 Elementen bei einem 3,5-Baum möglich?

Aufgabe 9.3 (b)

Wie viele verschiedene Baumstrukturen sind bei einem 3-Baum (3,6-Baum) mit 3 Ebenen möglich?

Aufgabe 9.4
Klausurvorbereitung: Fehlersuche

Aufgabe 9.4

Diese Aufgabe zeigt eine mögliche Aufgabenstellung für die Klausur. Es geht darum zu zeigen, dass Sie bekannte Algorithmen verstanden haben und mit Code dieser Algorithmen umgehen können.
Schauen Sie sich folgende drei Methoden an, die einen Algorithmus aus der Vorlesung implementiert haben.

- Beschreiben Sie welcher Algorithmus implementiert wurde und was dessen Laufzeit sein sollte.
- Geben Sie zudem den oder die Fehler in den gegebenen Methoden an und beschreiben Sie welche Auswirkungen die Fehler auf die Laufzeit und/oder das Ergebnis der Methoden haben.
- Beschreiben sie zudem, wie der Code verändert werden muss, um den Fehler zu beheben.
- Sind diese Methoden genau so performant wie die Implementierung aus der Vorlesung?

Aufgabe 9.4

a) Methode 1:

```
1  public static void foo(int[] arr, int l, int r) {
2      if (l >= r - 1) {
3          return;
4      }
5
6      int p = arr[r - 1];
7      int i = l;
8      for (int j = l; j < r - 1; j++) {
9          if (arr[j] < p) {
10             int temp = arr[i];
11             arr[i] = arr[j];
12             arr[j] = temp;
13             i++;
14         }
15     }
16     int temp = arr[i];
17     arr[i] = arr[r - 1];
18     arr[r - 1] = temp;
19
20     foo(arr, l, i);
21     foo(arr, i, r);
22 }
```

Algorithmus?

Laufzeit?

Fehler und Auswirkungen?

Mögliche Korrektur?

Vergleich mit Vorlesung?

Aufgabe 9.4

b) Methode 2:

```
1  public static int bar(int[] arr, int v) {
2      int l = 0;
3      int r = arr.length;
4      while (l < r) {
5          int m = l + (r - l) / 2;
6          if (arr[m] - v < 0) {
7              l = m + 1;
8          } else if (arr[m] - v > 0) {
9              r = m - 1;
10         } else {
11             return m;
12         }
13     }
14     return -1;
15 }
```

Algorithmus?

Laufzeit?

Fehler und Auswirkungen?

Mögliche Korrektur?

Vergleich mit Vorlesung?

Aufgabe 9.4

Methode 3: *stream* erzeugt dabei einen Stream aus dem übergebenen Array, *of* gibt einen Stream zurück, der nur den angegebenen Wert enthält, und *concat* konkateniert zwei Streams zu einem längeren.

```
1  public static int[] baz(int[] arr) {
2      if (arr.length == 0) {
3          return arr;
4      }
5
6      int pivot = arr[arr.length - 1];
7      int[] left = stream(arr).filter(x -> x < pivot).toArray();
8      int[] right = stream(arr).filter(x -> x > pivot).toArray();
9      return concat(
10         concat(stream(baz(left)), of(pivot)),
11         stream(baz(right))
12     ).toArray();
13 }
```

Algorithmus?

Laufzeit?

Fehler und Auswirkungen?

Mögliche Korrektur?

Vergleich mit Vorlesung?