

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.

- Verwenden Sie zunächst die Hashfunktion: $g(x) = 5x \mod m$
- 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 $w = \lfloor \log_2 m \rfloor = \lfloor 3.45 \dots \rfloor = 3$ gilt und der Ausdruck $a \cdot x'$ ein Skalarprodukt bezeichnet.

Aufgabe 9.1 (a)

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

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

Aufgabe 9.1 (a)

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

$k(e)$	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	
				3							

2.) insert(11)

0	1	2	3	4	5	6	7	8	9	10	
11				3							

3.) insert(9)

0	1	2	3	4	5	6	7	8	9	10	
11	9			3							

4.) insert(7)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3							

5.) insert(14)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3			14				

6.) insert(56)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3	56						

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	
11	9	7		3	56			4			

8.) insert(12)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3	56			4			

9.) insert(15)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7	?	3	56			4	15		

10.) insert(8)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3	56		8		4	15	

11.) insert(1)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3	56		8		4	15	

12.) delete(56)

0	1	2	3	4	5	6	7	8	9	10	
11	9	7		3	12		8		4	15	

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
11	9	7		3	12		8		4	

Red annotations:

- Row 1: Values 11, 9, 7, 14, 1, 25, 8, 15.
- Row 2: Values 11, 9, 7, 14, 1, 25, 8, 15.

Aufgabe 9.1 (b)

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

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

$$\omega = \lfloor \log_2 m \rfloor$$

$$= \lfloor 3,45\dots \rfloor$$

$$= 3$$

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

- 1.) $x' = \dots$
- 2.) $a \cdot x'$
- 3.) $\bmod m$

$$k_c=1 : \quad x' = (0, 1) \rightarrow 7 \cdot 0 + 5 \cdot 1 = 5 \rightarrow \underline{\underline{5}}$$

$$k=3 : \quad x' = (0, 3) \rightarrow 7 \cdot 0 + 5 \cdot 3 = 15 \rightarrow \underline{\underline{4}}$$

:

- Gleiche Hashwerte wie in (a)

Aufgabe 9.2

B-Baum Analyse

Aufgabe 9.2

Der Aufwand nach dem Einfügen 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 $k,2k$ -Baum, bei einem B-Baum ist also die Obergrenze für die Anzahl der Kinder eines Knoten genau das Doppelte.

$$3 - \text{B} \xrightarrow{\text{m}} 2 = 3 \quad b = 2 \cdot 3 = 6$$

- 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.
- 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?
- 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

- Größtmögliche Tiefe \Rightarrow möglichst wenige Elemente pro Knoten $(k-1)$

- k -Baum: $\log_k(n)$ Ebenen

- Pro Knoten: $\log_2(k)$ Vergleiche \Rightarrow Mit binärer Suche im Knoten

$$\hookrightarrow \log_k(n) \cdot \log_2(k) = \frac{\log_2(n)}{\cancel{\log_2(k)}} \cdot \cancel{\log_2(k)} = \underline{\underline{\log_2(n)}}$$

Anmerkung: Dies sind Abschätzungen nach oben (z.B. $\log_2(k)$ statt $\log_2(k-1)$)

Aufgabe 9.2 (b)

Gesucht: Möglichst viele Vergleiche im Verhältnis zur Elementzahl

- $\underbrace{k, 2k}$ - Baum
- Möglichst Elemente pro Knoten
- $\leq \log_k(n)$ Ebenen (Abschätzung durch Ergebnis in (a) für bessere Vergleichbarkeit)
- $\log_2(2k)$ Vergleiche pro Knoten

$$\hookrightarrow \log_k(n) \cdot \log_2(2k) = \frac{\log_2(n)}{\log_2(k)} \cdot \log_2(2k) = \underbrace{\frac{\log_2(2k)}{\log_2(k)}} \cdot \log_2(n)$$

Aufgabe 9.2 (c)

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

$$l_c = 2 : 2 \cdot \log_2(n)$$

$$l_c = 4 : 1,5 \cdot \dots$$

$$l_c = 8 : 1,33 \cdot \dots$$

$$\vdots \\ l_c = 512 : 1,11 \cdot \log_2(n)$$

- Mit steigendem k nähert sich die Vergleichzahl $\log_2(n)$ an
- Dies entspricht binärer Suche
- Intuitiv: Wenn alle Elemente in einem Knoten sind, wird nur dieser mit bin. Suche durchsucht.
Steigendes k nähert diesen Fall an.

Aus (b) :

$$\underbrace{\frac{\log_2(2k)}{\log_2(k)}}_{\text{L'Hospital}} \cdot \log_2(n)$$

$$\lim_{k \rightarrow \infty} \left(\frac{\log_2(2k)}{\log_2(k)} \right) = \lim_{k \rightarrow \infty} \left(\frac{\frac{\ln(2k)}{\ln(2)}}{\frac{\ln(k)}{\ln(2)}} \right) = \lim_{k \rightarrow \infty} \left(\frac{\ln(2k)}{\ln(k)} \right) \xrightarrow{\text{L'Hospital}} \lim_{k \rightarrow \infty} \left(\frac{\frac{1}{2k} \cdot 2}{\frac{1}{k}} \right) = \lim_{k \rightarrow \infty} (1) = \underline{\underline{1}}$$

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?

- Ebenen einschränken: 1 Ebene \rightarrow nur Wurzel \rightarrow Maximal 4 Elemente (zu wenig)

3 Ebenen \rightarrow Wurzel 1 Element, 2 Kinder mit 2 Elementen, je 3 Kinder mit 2 Elementen darunter (kleinst möglicher 3-Ebenen-Baum)

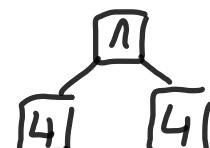
$$\Rightarrow 1 + 2 \cdot 2 + 6 \cdot 2 = 17 \text{ (zu viel)}$$

\Rightarrow 2 Ebenen

Wurzelelemente:

1

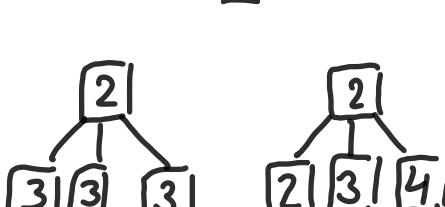
größter Baum:



$n=9$

X

2

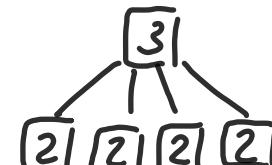


1x

$6 \times$
(Permutationen
der Ebene 2)

Tobias Eppacher

3

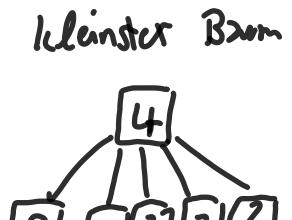


1x

$8 \times$ Möglichkeiten

keine andere
Möglichkeit
da Ebene 2
minimal

4



$n=14$

X

Aufgabe 9.3 (b)

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

- Betrachte Bäume ebenenweise von unten nach oben

1 Ebene: 2, 3, 4, 5 Elemente \Rightarrow 4 Möglichkeiten

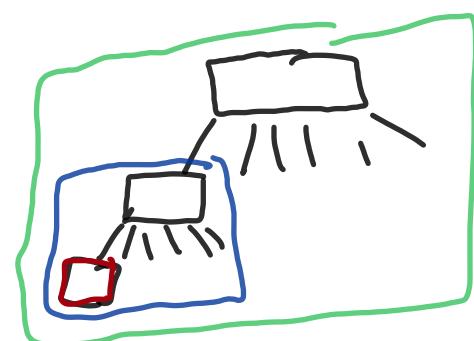


2 Ebenen: 3, 4, 5, 6 Kinder, je Kind 4 Möglichkeiten $\Rightarrow 4^3 + 4^4 + 4^5 + 4^6 = 5.440$ Möglichkeiten



3 Ebenen: Wurzelebene (auch nur 2 Kinder möglich), 2, 3, 4, 5, 6 Kinder, je Kind 5.440 Möglichkeiten

$$\hookrightarrow 5.440^2 + \dots + 5.440^6 \approx 2,6 \cdot 10^{22} \text{ Möglichkeiten}$$



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

Algorithmus? Quicksort

Laufzeit? $O(n^2)$ Worstcase | $O(n \log n)$ Avg. Case

Fehler und Auswirkungen?

Zweiter rek. Aufruf nimmt Pivot ernst zum sortieren mit. Kann bei Duplikaten zu Endlosschleife führen.

Bsp. Array [3,3,3]

Mögliche Korrektur?

i beim zweiten rek. Aufruf ausschließen

→ $\text{foo}(arr, i+1, r);$

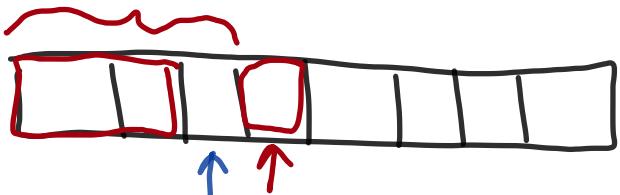
Vergleich mit Vorlesung?

Benötigt generell mehr swaps als Vorlesung simpl. mit zwei Pointern → langsamer

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 - 1) / 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? Bin. Search

Laufzeit? $O(\log_2 n)$

Fehler und Auswirkungen?

r wird falsch gesetzt. Rechte Grenze ist exclusive.

\Rightarrow Vergisst im unteren Teilaarray Elemente

Mögliche Korrektur?

$r = m$ statt $r = m - 1$

Vergleich mit Vorlesung?

gleich schnell

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* konkatениert 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? Quicksort

Laufzeit? $O(n^2)$ worst $O(n \log n)$ avg.

Fehler und Auswirkungen?

Sollte das Pivot Element mehrfach im Array vorkommen,
werden hier die Duplikate entfernt.

Im "merge"-Schritt wird der Pivot nur einmal eingefügt (*of(pivot)*)

Mögliche Korrektur?

Zusätzlicher Array *pivots* = *stream(arr).filter(x -> x == pivot).toArray()*
dann *stream(pivots)* statt *of(pivot)*

- Auch andere Lösungen möglich

Vergleich mit Vorlesung?

Längsspur als in Vorlesung
(Streams, Memory-Allocations für Arrays,...)