Matthias Krause
Alexander Moch

Mannheim, den 26.10.2020

CS 307 Algorithmen und Datenstrukturen, Herbstsemester 2020

Lösungsskizze zum Übungsblatt 5

AUFGABE 5.1:

Das Schlüsseluniversum $\mathcal{U}$ bestehe aus Zahlen 720·i, wobei $1 \leq i \leq 10^9$. Die Schlüssel werden mit Gleichverteilung gezogen.
Man gebe Zahlen m_1, m_2 aus dem Bereich $1000 \ldots 2000$ an, so dass

a) die Hashfunktion $h_1(k) = k \mod m_1$ die Bedingung des Simple Uniform Hashing (fast) erfüllt.

Laut Skript (Folie 109) empfiehlt es sich, m_1 als Primzahl zu wählen. Daher wählen wir $m_1 = 1439$. Wir sehen:

i	1	2	3	4	5	6	7	8	9	10	$\cdots$
$k = 720 \cdot i$	720	1440	2160	2880	3600	4320	5040	5760	6480	7200	$\cdots$
$h(k)$	720	1	721	2	722	3	723	4	724	5	$\cdots$

$\cdots$	1435	1436	1437	1438	1439	1440	1441	$\cdots$
$\cdots$	1033200	1033720	1034640	1035360	1036080	1036800	1037520	$\cdots$
$\cdots$	1437	718	1438	719	0	720	1	$\cdots$

b) die Hashfunktion $h_2(k) = k \mod m_2$ die Bedingung des Simple Uniform Hashing nicht erfüllt.

Alle Zahlen im Schlüsseluniversum $\mathcal{U}$ sind Vielfache von 720. Wir wählen daher $m_2 = 2 \cdot 720 = 1440$. Es ist leicht zu sehen, dass $k \mod 1440 \in \{0, 720\}$ liegt. Insbesondere gilt für alle $x \in [1439] \setminus \{0, 720\}$:

$$\Pr[k \mod 1440 = x] = 0.$$

AUFGABE 5.2$^{\mathbf{K}}$:

Folgende Schlüssel sollen in eine Hash-Tabelle eingefügt werden: 12, 23, 13, 56, 26, 45, 24, 94, 42.

a) Benutzen Sie zunächst eine Hash-Tabelle mit 13 Behältern und *chained hashing*. Als Hashfunktion soll $h_1(x) := (3x + 1) \mod 13$ benutzt werden. Wie sieht die Hash-Tabelle nach den Einfügeoperationen aus?

$(0 : 56), (1 : 26 \to 13), (5 : 23), (6 : 45), (8 : 24), (10 : 42 \to 94), (11 : 12)$

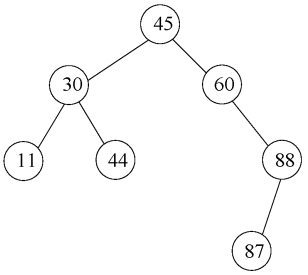
b) Wie sähe die Hash-Tabelle aus, wenn offene Adressierung mit Doppelhashing und der zusätzlichen Hashfunktion $h_2(x) := 1 + (x \mod 12)$ benutzt würde?

$(0 : 56), (1 : 13), (4 : 26), (5 : 23), (6 : 45), (8 : 24), (10 : 94), (11 : 12), (12 : 42)$

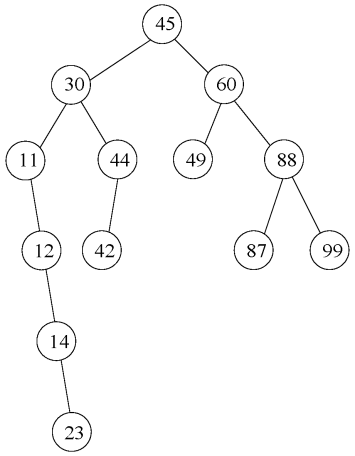
c) Wieviele Schlüsselvergleiche werden in jedem der beiden Fälle benötigt, wenn der Schlüssel 42 gesucht wird? (Zählen Sie nur die Vergleiche mit 42.)

Bei chained hashing einer, bei offener Adressierung fünf.

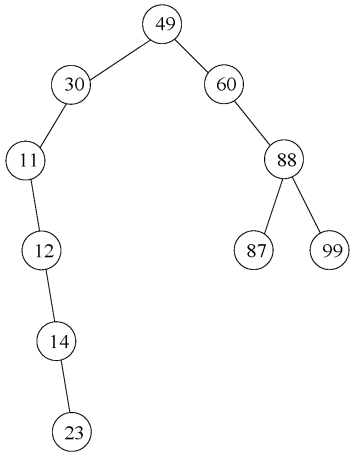
AUFGABE 5.3: Betrachten Sie den abgebildeten binären Suchbaum.



a) Fügen Sie in den Baum die Elemente mit den Schlüsseln 42, 12, 49, 14, 99, 23 (in dieser Reihenfolge) ein. Wie sieht der Baum nun aus?

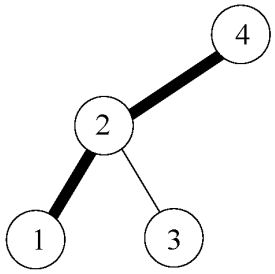


b) Löschen Sie aus dem Baum aus Aufgabenteil (a) die Elemente mit den Schlüsseln 44, 42, 45. Wie sieht der Baum nun aus?



AUFGABE 5.4: Angenommen, die Suche nach einem Schlüssel in einem binären Suchbaum endet in einem Blatt. Betrachten Sie die folgenden drei Mengen: P, die Schlüssel auf dem Suchpfad; L, die Schlüssel links von dem Suchpfad; und R, die Schlüssel rechts von dem Suchpfad. Beweisen Sie oder finden Sie ein Gegenbeispiel: Für alle $p \in P$, $l \in L$ und $r \in R$ gilt: $l \leq p \leq r$.

Gegenbeispiel:



AUFGABE 5.5: Beantworten Sie die folgenden Teilaufgaben, in denen es um die Tiefe von Suchbäumen geht:

a) Man gebe für alle n eine Folge π von n untereinander verschiedener Schlüssel an, die einen binären Suchbaum $T(\Pi)$ der maximal möglichen Tiefe $n-1$ erzeugt. $T(\Pi)$ bezeichnet hier den eindeutig bestimmten Suchbaum, der dadurch entsteht, dass die Schlüssel in der Reihenfolge π eingegeben werden.

n, n-1, n-2, ..., 2, 1

b) Für $n = 15$ und $n = 31$ gebe man jeweils eine Folge von n Schlüsseln an, die binäre Suchbäume der jeweils minimal möglichen Tiefe 3 bzw. 4 erzeugen.

$n = 15$*:*
8, 4, 2, 1, 3, 6, 5, 7, 12, 10, 9, 11, 14, 13, 15

$n = 31$*:*
16, 8, 4, 2, 1, 3, 6, 5, 7, 12, 10, 9, 11, 14, 13, 15, 24, 20, 18, 17, 19, 22, 21, 23, 28, 26, 25, 27, 30, 29, 31

c) Für alle $r \geq 1$ gebe man eine (ggf. rekursiv definierte) Folge von $n = 2^r - 1$ Schlüsseln an, die einen binären Suchbaum der minimal möglichen Tiefe $r-1$ erzeugen.

Das untenstehende MAGMA-Programm nutzt die gesuchte (allgemeine) Rekursion, um die beiden Lösungen von Teilaufgabe (b) zu ermitteln. Zum Ausführen kann der kostenlose Magma Calculator (http://magma.maths.usyd.edu.au/calc/) per Browser verwendet werden. Kopiert man dort bspw. den folgenden Quelltext in das entsprechende obere Textfeld und drückt „Submit“, sollten daraufhin im Feld unmittelbar darunter die genannten Ergebnisse für $n = 15$ *und* $n = 31$ *angezeigt werden.*

```
forward A;

A := function(low, high)
  root  := (low + high) / 2;

  result := [ root ];
```

```
  if (low ne high) then
    left  := A(low, (root - 1));
    result := result cat left;
    right := A((root + 1), high);
    result := result cat right;
  end if;

  return result;
end function;


// Beispiel: Loesungen zu Teilaufgabe b)
A(1, 15);
A(1, 31);
```

AUFGABE 5.6$^{\mathbf{K}}$: Geben Sie eine möglichst genaue untere Schranke (in der O-Notation) für die worst-case Laufzeit eines beliebigen vergleichsbasierten Algorithmus an, der eine beliebige Menge von n Zahlen bekommt und einen binären Suchbaum mit diesen Zahlen als Schlüsseln aufbaut. Begründen Sie Ihre Antwort.

Es gilt, dass die Laufzeit jedes vergleichsbasierte Algorithmus, der eine beliebige Menge von n Zahlen bekommt und daraus einen binären Suchbaum aufbaut, in $\Omega(n \cdot \log(n))$ liegt.

Diese Schranke kann nicht unterboten werden, da man aus jedem Suchbaum mit n inneren Knoten die sortierte Folge seiner Schlüssel in Zeit $O(n)$ erzeugen kann (InOrderTreeWalk), und andererseits jeder vergleichsbasierte Sortieralgorithmus eine worst-case Laufzeit von $\Omega(n \cdot \log(n))$ hat. Somit (da das Sortieren durch den binären Suchbaum die Zeit zum Erzeugen plus die Zeit zum Sortieren beinhaltet) muss das Erzeugen des Baums mindestens Zeit $\Omega(n \cdot \log(n))$ benötigen.

Dass diese Schranke auch tatsächlich erreicht wird, kann man wie folgt sehen: Es gibt einen vergleichsbasierten Algorithmus, der eine beliebige Menge von n Zahlen bekommt und daraus einen binären Suchbaum in Zeit in $O(n \cdot \log(n))$ aufbaut. Dieser Algorithmus trägt die n Zahlen in ein Feld A ein (Zeit $O(n)$), sortiert A in Zeit $O(n \cdot \log(n))$, z.B. durch HeapSort, und wandelt A in Zeit $O(n)$ in die Datenstruktur "binärer Suchbaum" um (durch setzen der entsprechenden Pointer). Somit braucht dieser Algorithmus insgesamt Zeit $O(n \cdot \log(n))$ und die Schranke wurde erreicht.