Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithmen und Datenstrukturen
Sommersemester 2024
Korrekturanweisung Übungsblatt 4

Abgabe: Dienstag, 14. Mai, 2024, 10:00 Uhr

Aufgabe 1: Hashing mit offener Adressierung *(5 Punkte)*

Sei $\mathcal{H}$ eine Hashtabelle der Größe $m = 13$ und seinen $h_1, h_2, h_3 : \mathbb{N}_0 \mapsto \{0, ..., m-1\}$ Hashfunktionen definiert wie folgt[1]:

- $h_1(x) := \overline{x} \mod m$
- $h_2(x) := 3 \cdot x \mod m$
- $h_3(x) := x + 1 \mod m$

Fügen Sie die Schlüssel $23, 12, 75, 945, 30, 99, 345$ (in dieser Reihenfolge) in die initial leere Hashtabelle $\mathcal{H}$ ein. Lösen Sie Konflikte wie folgt:

a) Lineares Sondieren unter der Benutzung von h_1. *(2 Punkte)*

b) Doppel-Hashing unter Benutzung von h_2 und h_3.[2] *(3 Punkte)*

Geben Sie den Zustand der Hashtabelle in jedem Schritt an!

Aufgabe 2: Hashing mit Chaining *(5 Punkte)*

Gegeben sei eine Hash-Table der Größe m und eine beliebige Hashfunktion $h : S \mapsto \{0, ..., m-1\}$. Die Menge S habe mindestens $y \cdot m$ Elemente, also $|S| \geq y \cdot m$.

a) Zeigen Sie, dass S mindestens eine Teilmenge Y, bestehend aus mindestens y Elementen (also $|Y| \geq y$), besitzt, so dass $h(x_1) = h(x_2)$ für alle $x_1, x_2 \in Y$. *(4 Punkte)*

b) Was sagt uns das Resultat in *a)* über die Worst-Case Laufzeit von *"find"* in einer Hashtabelle mit Chaining aus (wenn unsere Hashtabelle genau die Elemente aus S speichert bevor *"find"* aufgerufen wird)? *(1 Punkt)*

Aufgabe 3: Anwendung von Hashtabellen *(10 Punkte)*

Gegeben ist folgender Algorithmus:

Algorithm 1 `algorithm` ▷ Input: Array A of length n with integer entries

```
for i = 1 to n − 1 do
    for j = 0 to i − 1 do
        for k = 0 to n − 1 do
            if |A[i] − A[j]| = A[k] then
                return true
return false
```

[1] Wir definieren $\overline{x}$ als die Quersumme von x.
[2] Es soll also $(h_2(x) + i \cdot h_3(x)) \mod m$ als Hashfunktion verwendet werden.

1) • lineares sondieren mit $h_1(x) = \bar{x} \bmod m$

• bei Konflikten: $h_2(x) + i \cdot h_3(x)$

mit $h_2(x) = 3 \cdot x \bmod m$ und $h_3(x) = x + 1 \bmod m$

• m sei 13

x = 23

1	2	3	4	5	6	7	8	9	10	11	12	13
				23								

x = 12

1	2	3	4	5	6	7	8	9	10	11	12	13
		12		23								

x = 75

1	2	3	4	5	6	7	8	9	10	11	12	13
		12		23							75	

x = 945

1	2	3	4	5	6	7	8	9	10	11	12	13
945		12		23							75	

x = 30

1	2	3	4	5	6	7	8	9	10	11	12	13
945		12	30	23							75	

x = 99

1	2	3	4	5	6	7	8	9	10	11	12	13
945		12	30	23						99	75	

x = 345

1	2	3	4	5	6	7	8	9	10	11	12	13
945		12	30	23		345				99	75	

2) $|S| \geq y \cdot m$ $\quad |Y| \geq y$

zz.: $h(x_1) = h(x_2)$ für $\forall x_1, x_2 \in Y$

angenommen h ist eine zufällige Hashfunktion so ist die Wahrscheinlichkeit für die Berechnung eines Schlüssels $\frac{1}{m}$.
Multipliziert mit der Anzahl an Elementen $|S| \geq y \cdot m$ erhält man die Anzahl der Elemente pro Schlüssel $|Y| \geq y$

$$\to \quad y \cdot m \cdot \frac{1}{m} = y$$

$\Rightarrow$ wenn $x_1, x_2 \in Y$ so gilt, dass die Hashfunktion für beide Werte den selben Schlüssel errechnet. Somit gilt: $h(x_1) = h(x_2)$

b) für die Laufzeit gilt $T(n) \in O(1 + \alpha)$

mit $\alpha = \frac{n}{m}$ und $n = y \cdot m$

$$\alpha = \frac{y \cdot m}{m} = y$$

$$\Rightarrow T(n) \in O(1 + y)$$

3)a) Der Algorithmus berechnet den Betrag der Differenz einer Stelle eines Arrays mit den bereits durchlaufenen Stellen des Arrays und gibt true aus, wenn der Betrag dieser Differenz gleich groß ist wie ein Eintrag in dem gesamten Array

Laufzeit: $(n-1) \cdot n \cdot n = n^3 - n^2$
$\leq n^3 \leq c \cdot n^3$

$T(n) \in O(n^3)$

b) Alternativer Algorithmus: H: Hashtabelle

```
for i = 1 to n-1 do:
    for j = 0 to i-1 do:
        insert in H = |A[i] - A[j]|

for k = 0 to n-1 do:
    if A[k] in H:
        return true
return false
```

(a) Beschreiben Sie, was `algorithm` berechnet und analysieren Sie die asymptotische Laufzeit. *(3 Punkte)*

Hinweis: Die Differenz $|A[i] - A[j]|$ kann beliebig große Werte annehmen.

(b) Beschreiben Sie einen auf *hashing* basierenden alternativen Algorithmus $\mathcal{B}$ für dieses Problem (d.h. $\mathcal{B}(A) = \texttt{algorithm}(A)$ für jede Eingabe A) mit einer Laufzeit von $\mathcal{O}(n^2)$ (mit Begründung). *(3 Punkte)*

Hinweis: Sie dürfen annehmen, dass das Einfügen und Finden von Schlüsseln in einer Hashtabelle $\mathcal{O}(1)$ Zeitschritte benötigt, wenn $\alpha = \mathcal{O}(1)$ (α ist der Load der Hashtabelle).

(c) Beschreiben Sie einen weiteren Algorithmus für dieses Problem ohne Verwendung von Hashing mit einer Laufzeit von $\mathcal{O}(n^2 \log n)$ (mit Begründung). *(4 Punkte)*