Algorithmen und Datenstrukturen - Übungsblatt 4 Lösungen

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

Sommersemester 2024

Aufgabe 1: Hashing mit offener Adressierung

a) Lineares Sondieren unter der Benutzung von $h1$ (2 Punkte)

- **23:** $23 \mod 13 = 10$
  - Position 10
- **12:** $12 \mod 13 = 12$
  - Position 12
- **75:** $75 \mod 13 = 10$
  - Kollision, nächste freie Position (linear probing): 11
- **945:** $945 \mod 13 = 9$
  - Position 9
- **30:** $30 \mod 13 = 4$
  - Position 4
- **99:** $99 \mod 13 = 8$
  - Position 8
- **345:** $345 \mod 13 = 7$
  - Position 7

Zustand der Hashtabelle:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	–	–	–	–	30	–	–	345	99	945	23	75	12

b) Doppel-Hashing unter Benutzung von $h2$ und $h3$

- **23:**

  $h2(23) = 3 \cdot 23 \mod 13 = 69 \mod 13 = 4h3(23) = 23 + 1 \mod 13 = 24 \mod 13 = 11$- Position 4

- **12:**

  $h2(12) = 3 \cdot 12 \mod 13 = 36 \mod 13 = 10h3(12) = 12 + 1 \mod 13 = 13 \mod 13 = 0$- Position 10

- **75:**

  $h2(75) = 3 \cdot 75 \mod 13 = 225 \mod 13 = 4h3(75) = 75 + 1 \mod 13 = 76 \mod 13 = 11$- Kollision, nächste Position (

- **945:**

  $h2(945) = 3 \cdot 945 \mod 13 = 2835 \mod 13 = 1 h3(945) = 945+1 \mod 13 = 946 \mod 13 = 9$- Position 1

- **30:**

  $h2(30) = 3 \cdot 30 \mod 13 = 90 \mod 13 = 12 h3(30) = 30+1 \mod 13 = 31 \mod 13 = 5$- Position 12

- **99:**

  $h2(99) = 3 \cdot 99 \mod 13 = 297 \mod 13 = 11 h3(99) = 99+1 \mod 13 = 100 \mod 13 = 9$- Position 11

- **345:**

  $h2(345) = 3 \cdot 345 \mod 13 = 1035 \mod 13 = 9 h3(345) = 345+1 \mod 13 = 346 \mod 13 = 8$- Position 9

Zustand der Hashtabelle:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	–	945	75	–	23	–	–	–	–	345	12	99	30

Aufgabe 2: Hashing mit Chaining

a) Beweis, dass S eine Teilmenge Y besitzt

Gegeben:

- Hash-Table der Größe m
- Hashfunktion $h : S \to \{0, ..., m-1\}$
- $|S| \geq y \cdot m$

Beweis:

- Nach dem Schubfachprinzip (Pigeonhole Principle) müssen bei $|S| \geq y \cdot m$ mindestens y Elemente in mindestens einem der m Schubfächer (Buckets) landen.
- Daraus folgt, dass es mindestens eine Teilmenge Y mit $|Y| \geq y$ gibt, für die $h(x1) = h(x2)$ für alle $x1, x2 \in Y$.

b) Worst-Case Laufzeit von find" in einer Hashtabelle mit Chaining

Ergebnis:

- Im Worst-Case müssen wir alle Elemente in einem Bucket durchsuchen.
- Wenn $|Y| \geq y$, dann ist die Worst-Case Laufzeit von find" $O(y)$, weil wir im schlimmsten Fall alle y Elemente in diesem Bucket durchsuchen müssen.

Aufgabe 3: Anwendung von Hashtabellen)

a) Beschreibung und asymptotische Laufzeit des Algorithmus

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
```

Beschreibung:

- Der Algorithmus überprüft, ob es zwei Indizes i und j gibt, so dass die Differenz $|A[i] - A[j]|$ einem Wert in A entspricht.

Laufzeitanalyse:

- Die drei verschachtelten Schleifen haben Laufzeiten $O(n)$, $O(n)$ und $O(n)$.
- Daher ist die Gesamtlaufzeit $O(n^3)$.

b) Alternativer Algorithmus B mit Hashing

Beschreibung:

- Nutzen einer Hashtabelle, um die Differenzwerte effizient zu speichern und zu finden.

Algorithmus:

1. Initialisiere eine leere Hashtabelle H.
2. Für jedes Paar (i, j):
   - Berechne die Differenz $d = |A[i] - A[j]|$.
   - Prüfe, ob d in H ist.
   - Falls ja, return true.
   - Falls nein, füge d zu H hinzu.
3. Wenn keine Differenz gefunden wurde, return false.

Laufzeit:

- Die äußeren beiden Schleifen haben Laufzeit $O(n^2)$.
- Einfügen und Finden in der Hashtabelle haben amortisierte Laufzeit $O(1)$.
- Daher ist die Gesamtlaufzeit $O(n^2)$.

c) Algorithmus ohne Hashing mit Laufzeit $O(n^2 \log n)$

Beschreibung:

- Sortieren des Arrays und dann binäre Suche für die Differenz.

Algorithmus:

1. Sortiere das Array A (Laufzeit $O(n \log n)$).
2. Für jedes Paar (i, j):
   - Berechne die Differenz $d = |A[i] - A[j]|$.
   - Suche d im sortierten Array mittels binärer Suche (Laufzeit $O(\log n)$).

  - Falls d gefunden, return true.

3. Wenn keine Differenz gefunden wurde, return false.

Laufzeit:

- Sortieren hat Laufzeit $O(n \log n)$.
- Die äußeren beiden Schleifen haben Laufzeit $O(n^2)$.
- Binäre Suche hat Laufzeit $O(\log n)$.
- Daher ist die Gesamtlaufzeit $O(n^2 \log n)$.