Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. Fabian Kuhn
P. Schneider, P. Bamberger, Y. Maus

Informatik 2 - Sommersemester 2018
Übungsblatt 4

Abgabe: Montag, 28. Mai, 14:00 Uhr

Aufgabe 1: Fixpunktsuche *(7 Punkte)*

Gegeben sei ein *aufsteigend sortiertes Array* A der Länge n gefüllt mit *paarweise verschiedenen* ganzen Zahlen. Unser Ziel ist die Existenz eines Fixpunktes nachzuweisen oder zu widerlegen. Ein Fixpunkt ist ein Index $i \in \{0, \dots, n-1\}$ für den $A[i] = i$ gilt. Wir schlagen den folgenden Algorithmus vor, welcher den Index eines Fixpunktes zurück gibt sofern dieser existiert und sonst n.

Algorithm 1: `FindFixedPoint`(A)

```
ℓ ← 0; r ← n−1
while ℓ ⩽ r do
    p ← ⌊(ℓ+r)/2⌋
    if A[p] < p then
        ℓ ← p+1
    else if A[p] > p then
        r ← p−1
    else
        return p
return n
```

(a) Geben Sie die Worst-Case Laufzeit $T(n)$ als Rekursions(un)gleichung an. Sie dürfen annehmen dass jeder Schleifendurchlauf eine einzige Zeiteinheit benötigt. Geben Sie außerdem die asymptotische Laufzeit an. *(2 Punkte)*

(b) Formulieren Sie Vorbedingung und Nachbedingung der Schleife, sowie die Schleifeninvariante bezüglich der Existenz eines Fixpunktes im Suchbereich $\{\ell, \dots, r\}$. Beweisen Sie die Schleifeninvariante formal induktiv. Leiten Sie aus der Schleifeninvariante die Korrektheit her. *(5 Punkte)*

Aufgabe 2: Schlechte Hashfunktionen *(5 Punkte)*

Sei m die Größe einer Hashtabelle und $n \gg m$ der größte mögliche Schlüssel der Datenelemente die wir in dieser Hashtabelle speichern möchten. Die folgenden "Hashfunktionen" sind aus unterschiedlichen Gründen schlecht gewählt. Erläutern sie für jede Hashfunktion kurz warum das so ist.

(a) $h_1 : k \mapsto k$. *(1 Punkt)*

(b) $h_2 : k \mapsto \lfloor \frac{k}{n} \cdot (m-1) \rfloor$. *(1 Punkt)*

(c) $h_3 : k \mapsto 2 \cdot \big(k \bmod \lfloor \frac{m-1}{2} \rfloor\big)$. *(1 Punkt)*

(d) $h_4 : k \mapsto$ `random`(m), $\big($`random`(m) ist eine gleichverteilt zufällige Zahl aus $\{0, \dots, m-1\}\big)$. *(1 Punkt)*

(e) $h_5 : k \mapsto$ `chaos`(m), $\big($`chaos`(m) berechnet einen guten Hashwert aus $\{0, \dots, m-1\}$ deterministisch in $\mathcal{O}(\log n)$ vielen Zeitschritten$\big)$. *(1 Punkt)*

Aufgabe 3: Hashing mit Chaining *(8 Punkte)*

Implementieren Sie eine Hashtabelle zum Abspeichern positiver, ganzzahliger Schlüssel. Sie dürfen annehmen dass keine Duplikate abgespeichert werden. Benutzen Sie zur Kollisionsbehandlung *Hashing mit Chaining* und nutzen Sie dazu die `DoublyLinkedList` von Aufgabenblatt 3. Sie dürfen die Funktion $k \mapsto (k \bmod m)$ als (triviale) Hashfunktion benutzen, oder anspruchsvollere aus der Vorlesung.

Hinweise: Im public*-Ordner finden Sie eine vorgefertigte Strukturdatei* `HashTable.py`*, die spezifiziert welche Operationen Ihre Hashtabelle anbieten soll. Sie* können *die Strukturdatei herunterladen und anpassen, bzw. mit Programmcode füllen. Falls Sie* `DoublyLinkedList` *aus Aufgabenblatt 3 nicht (korrekt) implementiert haben, können Sie die Musterlösung aus dem* public*-Ordner im SVN verwenden.*