Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Sommersemester 2021
23. 05. 2021

Prof. Dr. Bernhard Seeger
Nikolaus Glombiewski M.Sc.
Amir El-Shaikh

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Abgabe: 30. 05. 2021,
bis **spätestens** 23:59 Uhr
über die ILIAS Plattform

Übung 6

Aufgabe 6.1: Manuelles Hashing (1+2) (3 Punkte)

Fügen Sie im Folgenden die Datensätze mit den Schlüsseln: 55, 70, 125, 3, 8, 133, 16, 21, 172 in dieser Reihenfolge in eine Hashtabelle der Länge 13 unter Verwendung verschiedenener Kollisions-Strategien ein. Der Wert der jeweiligen Schlüssel ist die textuelle Repräsentation der Zahl in der englischen Sprache, d.h. für 55 wird das Key-Value Paar <55, fifty-five> eingefügt.

a) **Alternierendes quadratisches Sondieren** mit Hash-Funktion $h(k) = k \mod 13$

b) **Kuckucks-Hashing** mit den Hash-Funktionen $h_0(k) = k \mod 13$ und $h_1(k) = \lfloor \frac{k}{13} \rfloor \mod 13$

Aufgabe 6.2: Hashing Theorie (1+2) (3 Punkte)

a) Gegeben sei der Wertebereich W einer Hashtabelle H (direkte Verkettung) mit m Adressen. Es gilt $|W| > n \cdot m$. Zeigen Sie: Es gibt eine Menge $W' \subseteq W$ mit $|W'| = n$, sodass nach Einfügen aller Elemente aus W' in H die Worst-Case Laufzeit für die Suche $\Theta(n)$ beträgt.

b) Beim linearem Hashing gibt es einen Belegungsfaktor α, ab welchem die Hash-Tabelle erweitert wird. Neben linearem Hashing mit separater Verkettung, wie aus der Vorlesung bekannt, existieren auch Vorschläge lineares Hashing mit anderen Sondierungsverfahren umzusetzen. Erläutern Sie am Beispiel von linearem Sondieren, warum bei der Wahl von α ein Wert der gegen 1 geht ungünstig ist.

Aufgabe 6.3: Hashing Implementierung (5+4) (9 Punkte)

In dieser Aufgabe entwickeln Sie Implementierungen einer HashMap. Das zur Verfügung gestellte Interface `HashFunction` bietet eine Methode um Objekten beliebigen (generischen) Typs eine Integer-Zahl zuzuordnen. Das Interface Map stellt eine abgespeckte Version der `java.util.Map` dar, welches Sie implementieren sollen. `MapEntry<K, V>` soll als Struktur für Datensätze in Ihren Hash-Tabellen verwendet werden.

a) Vervollständigen Sie die Klasse `LinearHashingMap`, indem Sie die Methoden `getAddress`, `getAlpha`, `checkOverflow`, `split` und `get` implementieren. Die `LinearHashingMap` speichert Schlüssel-Wert-Paare in Buckets die jeweils mehrere Elemente in einer Liste verwalten. Sobald ein vorgegebenes Maximum für den Belegungsfaktor (`alphaMax`) erreicht ist, wird die Hash-Tabelle mittels der split-Methode schrittweise vergößert.

b) Vervollständigen Sie die Klasse `SecondChoiceMap`, indem Sie die Methoden `get`, `put` und `remove` implementieren.

Hinweis: Beachten Sie, dass `HashFunction` **Werte im Bereich** `[Integer.MIN_VALUE, Integer.MAX_VALUE]` **liefert und diese von Ihnen geeignet auf den jeweiligen Addressbereich der Hashtabelle abgebildet werden müssen.**

Aufgabe 6.4: Bloom Filter* (2+2+1) **(5 Punkte)**

Ein **BloomFilter** ist eine probalistische Datenstruktur, mit welcher effizient untersucht werden kann, ob ein Element *nicht* in der Datenstruktur liegt. Der **BloomFilter** besteht aus einem Bit-Array fester Größe und k Hashfunktionen, welche auf die Positionen des Bit-Arrays abbilden.

Wird ein Element in den **BloomFilter** eingefügt, wird jede der k Hash-Funktionen auf das Element angewandt. Die Ergebnisse dieser Operation sind k Positionen im Bit-Array. Die Bits dieser Positionen werden auf 1 gesetzt.

Wird ein Element im **BloomFilter** gesucht, werden die k Hashfunktionen auf dieses Element angewandt und die entsprechenden Positionen im Bit-Array untersucht. Ist mindestens ein Bit 0, so kann mit Sicherheit gesagt werden, dass das Element nicht in den **BloomFilter** eingefügt wurde. Sind alle Bits 1, kann aufgrund von möglichen Kollisionen nicht gesagt werden, ob das Element tatsächlich im BloomFilter liegt. Durch diese Eigenschaft, und die Tatsache, dass der **BloomFilter** die eigentlichen Elemente nicht abspeichert, wird der **BloomFilter** in Kombination mit anderen Datenstrukturen verwendet, auf welche man in diesen Fällen zurückgreift.

a) Implementieren Sie das Interface `BloomFilter` in einer Klasse `BloomFilterImpl`. Verwenden Sie zur Verwaltung der Hash-Funktionen das Interface `HashFunction`. Generieren Sie im Konstruktor k Hash Funktionen und ein Bit-Array der Größe m basierend auf Parametern im Konstruktor.

b) Stellen Sie für das Interface `list.List` eine Klasse `BloomFilterList` bereit, welche ein BloomFilter Objekt und eine Instanz der `list.List` Schnittstelle übergeben bekommt. Verwenden Sie an geeigneten Stellen den **BloomFilter**. Wenn nötig, delegieren Sie Operationen an die übergebene Liste weiter. Beschreiben Sie, in welchen Operationen der Einsatz des **BloomFilters** potentiell Schwierigkeiten bereitet.

c) Vervollständigen Sie die Methode `initBloomFilter` aus der Klasse `BloomFilterMain`, sodass die Methode ein Objekt der `BloomFilterList` Klasse zurückgibt. Der **BloomFilter** soll hierbei `HASHES` viele Hashfunktionen und ein Bit-Array der Größe `BUCKETS` verwenden. Führen Sie das Programm aus und erklären Sie Ihre Resultate.

Hinweis: Beachten Sie, dass `HashFunction` **Werte im Bereich** `[Integer.MIN_VALUE, Integer.MAX_VALUE]` **liefert und diese von Ihnen geeignet auf den jeweiligen Addressbereich der Hashtabelle abgebildet werden müssen.**

*** Aufgabe 6.4 ist für Lehramtsstudierende optional.**