Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Thomas Seidl
Janina Sontheim, Florian Richter

München, 26.06.2019

Algorithmen und Datenstrukturen

SS 2019

Übungsblatt 9: Suchen II

Aufgabe 9-1 *Bitvektoren*

Hinweis: Man kann Bitvektoren auf zwei gegensätzliche Arten definieren. Dabei gibt es keine richtige Definition. Um es etwas intuitiver und Übung sowie Vorlesung konform zu machen, definieren wir: Ein Element ist genau dann in der Menge enthalten, wenn das entsprechende Bit den Wert 1 hat. Wir haben dies entsprechend auf den Vorlesungsfolien geändert.

Gegeben ist wieder die Sammlung von Informatik-Fachbüchern:

Nummer	Titel	Autor	Jahr	Kürzel
0	Design Patterns	E. Gamma, et al.	1994	DP
1	Clean Code	Robert C. Martin	2008	CC
2	Make Your Own Neural Network	Tariq Rashid	2016	MNN
3	Agile Software Development	Robert C. Martin	2002	AgSD
4	Introduction to Algorithms	T. H. Cormen, et al.	1989	IA
5	Functional Thinking: Paradigm Over Syntax	Neal Ford	2014	FuT
6	Extreme Programming Explained: Embrace Change	Kent Beck	1999	ExPE
7	Algorithms for Reinforcement Learning	Csaba Szepesvari	2010	AlRL
8	The Software Craftsman	Robert C. Martin	2014	TheSC
9	Test Driven Development: By Example	Kent Beck	2002	TDD
10	Programming Pearls	Jon Bentley	1986	PP
11	Building Your Own Compiler with C++	Jim Holmes	1994	BYOC

Das Universum U soll im folgendem alle obigen Bücher enthalten. Ein Eintrag Bit[i] eines Bitvektors steht dafür, ob ein Buch mit Nummer i in einer gegebenen Menge enthalten ist.
Hinweis: Sie dürfen die Vektoren auch als Zeilenvektoren schreiben.

(a) Geben Sie die Größe des Universums N an.

(b) Geben Sie den Bitvektor an, der das Universum repräsentiert. Wie viele Elemente (Bücher) enthält die Menge, die er repräsentiert.

(c) Welcher Bitvektor repräsentiert folgende Menge an Büchern $M_c = \{$AgSD, TDD, BYOC, IA, TheSC$\}$?

(d) Welcher Menge repräsentiert der Bitvektor $Bit(M_d) = (1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0)^T$?

(e) Wie könnte ein effizienter Algorithmus aussehen, der mittels zweier Bitvektoren $Bit(M_1)$ und $Bit(M_2)$ die Schnittmenge und Vereinigung der Mengen, die sie repräsentieren (M_1 und M_2), berechnet und diese wieder als Bitvektor $Bit(M_{result})$ ausgibt? Geben Sie diesen Algorithmus möglichst formal korrekt in Pseudocode an. Wie groß ist die Laufzeit in O-Notation?

Lösungsvorschlag:

(a) $N = 12$

(b) $Bit(U) = (1,1,1,1,1,1,1,1,1,1,1,1)^T$. Die Menge enthält natürlich genau $N = 12$ Elemente (Bücher).

(c) $Bit(M_c) = (0,0,0,1,1,0,0,0,1,1,0,1)^T$

(d) $M_d = \{$DP, AgSD, IA, FuT, TDD$\}$

(e) Schnittmenge:

```
Algorithmus schnitt(Bit(M_1), Bit(M_2)) :
        Bit(M_result) = Initialize new Bitvektor
        for i = 0 bis N-1
                Bit(M_result)[i] = Bit(M_1)[i] AND Bit(M_2)[i]
        return Bit(M_result)
```

$\Rightarrow$ Es gilt also $Bit(M_1 \cap M_2) = schnitt(Bit(M_1), Bit(M_2))$

Vereinigung:

```
Algorithmus vereinigung(Bit(M_1), Bit(M_2)) :
        Bit(M_result) = Initialize new Bitvektor
        for i = 0 bis N-1
                Bit(M_result)[i] = Bit(M_1)[i] OR Bit(M_2)[i]
        return Bit(M_result)
```

$\Rightarrow$ Es gilt also $Bit(M_1 \cup M_2) = vereinigung(Bit(M_1), Bit(M_2))$

Die Laufzeit ermittelt sich folgendermaßen:

- Die Initialisierung benötigt $O(N)$ (siehe Vorlesung)
- Die for-Schleife wird N mal durchlaufen $\Rightarrow O(N)$
- Innerhalb der for-Schleife finden zwei Suchen nach Elementen statt, diese sind in konstanter Zeit durchführbar (siehe Vorlesung), die logischen Operationen ($\wedge$ und $\vee$) bzw. ('AND' und 'OR') sind ebenfalls in konstanter Zeit durchführbar. Eine Iteration ist also in $O(1)$ durchführbar.

Insgesamt ergibt sich für die Laufzeit also: $O(N + N * 1) = O(N)$

Aufgabe 9-2 *Hashing*

Gegeben sei folgende Hashfunktion $h(k) = k \bmod 7$. Fügen sie nun nacheinander folgende Schlüssel in eine Hashtabelle ein. Verwenden Sie dabei *offenes Hashing*:

(a) 1, 7, 13, 15, 14, 5

(b) c, z, a, j, v, k, l *(überlegen Sie Möglichketen, wie Sie Buchstaben mit der Hashfunktion $h(k)$ in die Tabelle einfügen können)*

Lösungsvorschlag:

(a)

0	7	→ 14
1	1	→ 15
2		→
3		→
4		→
5	5	→
6	13	→

(b) Wir brauchen eine Funktion, die die Buchstaben in Zahlen umwandelt. Dafür eignet sich zum Beispiel die ORD-Funktion: $ORD(a) = 1, ORD(c) = 3$, und so weiter. Alternativ könnte man auch die ASCII-Kodierung der einzelnen Buchstaben verwenden; damit kann man auch zwischen groß-/kleinschreibung unterscheiden. In der Lösung nehmen wir aber die einfachere ORD-Funktion her.

0		→
1	a	→ v
2		→
3	c	→ j
4	k	→
5	z	→ l
6		→

Aufgabe 9-3 *Geschlossenes und doppeltes Hashing*

Gegeben sind die Zahlen {13, 7, 31, 19, 27, 42, 69, 96} welche in eine Hashtabelle der Größe 11 eingeordnet werden soll.

(a) Benutzen Sie geschlossenes Hashing mit $h(k) = k \bmod 11$ und ordnen Sie die Zahlen ein. Kennzeichnen Sie Kollisionen und lösen Sie diese durch lineares Sondieren mit $c_1 = 1$.

(b) Mutzen Sie nun Doppel-Hashing mit $h(k) = k \bmod 11$, der Sekundären Hashfunktion $h'(k) = k \bmod 9 + 1$ und der Sondierungsfunktion $h_j(k) = (h(k) + j * h'(k)) \bmod 11$. Dokumentieren Sie Kollisionen und deren Lösung.

Lösungsvorschlag:

geschlossenes Hashing mit linearem Sondieren und $c_1 = 1$:

$13 \to 2$
$7 \to 7$
$31 \to 9$
$19 \to 8$
$27 \to 5$
$42 \to 9 \to 10$
$69 \to 3$
$96 \to 8 \to 9 \to 10 \to 0$

doppeltes Hashing:

$13 \to 2$
$7 \to 7$
$31 \to 9$
$19 \to 8$
$27 \to 5$
$42 \to 9 : h_1(42) = (h(42) + 1 * h'(42)) mod 11 = 5 \to h_2(42) = (h(42) + 2 * h'(42)) mod 11 = 1$
$69 \to 3$
$96 \to 8 \to h_1(96) = (h(96) + 1 * h'(96)) mod 11 = 4$