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

München, 22.07.2021

Algorithmen und Datenstrukturen

SS 2021

Übungsblatt 10: Hashing

Tutorien: 28.07.-02.08.2021

Aufgabe 10-1 *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?

Aufgabe 10-2 *Geschlossenes und doppeltes Hashing*

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

(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) Nutzen 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.

Aufgabe 10-3 *Bonus: Blockchain*

Wir haben in Discord einen Textchannel **#smart-tale** für diese Aufgabe angelegt. Dieser Channel soll für die Daten dieser Aufgabe reserviert bleiben, daher bitte dort keine sonstigen Nachrichten posten, damit die Übersicht erhalten bleibt.
Eine BlockChain ist eine einfach verkettete Liste wie Sie sie bereits aus der Vorlesung kennen. Jedes Listenelement besteht aus drei Bestandteilen:

- Hash: Der Hash referenziert den vorherigen Block. Wir nutzen für unser Beispiel die weit verbreitete Funktion SHA256. In allen gängigen Programmiersprachen wird diese in Libraries zur Verfügung gestellt (Python: hashlib.sha256, Java: MessageDigest.getInstance(SSHA-256")). Wie SHA256 arbeitet, werden wir hier nicht erläutern. Wir könnten genauso gut eine andere Hashfunktion verwenden, müssen uns aber im Blockchain-Netzwerk auf einen Standard festlegen.
- Data: Die bekannteste Anwendung auf Blockchains sind sicherlich Bitcoin Transactions. Blockchains lassen sich aber auch für andere Daten verwenden. Wir benutzen als Datensatz eines einzigen Blocks stets einen String, der einen einzigen Satz beinhaltet.
- Nonce: Ein String, der zum Abschließen des Blocks ermittelt werden muss. Die Nonce ist genau dann gültig, wenn die String-Konkatenation Hash+Data+Nonce mittels SHA256 auf einen Wert gehasht wird, der mit $k = 7$ Nullen beginnt. Je kleiner k gewählt wird, desto schneller kann eine passende Nonce gefunden werden. Größere k erhöhen den Aufwand. Das Finden einer Nonce nennt man Proof-of-Work und ist der wesentliche Bestandteil beim Bitcoin Mining.

Eine Blockchain beginnt stets mit einem Genesisblock G. Dies ist der einzige Block, der in unserem Fall mit einem leeren Hashwert beginnt. Unser Genesisblock lautet:

Es war einmal in einem fernen
Land.77c638b18e8413931d98e94cfce023c96e5fffdf505aaeb04ef0c7f3d2ef8ba2

Dieser Block ist gültig, denn SHA256(G) beginnt mit 7 Nullen:

$$SHA256(G) = 0000000d0f746b7b0f4404cce1b61eec7dc4c65622d80647e46548e1404283d6$$

Der zweite Block hat dann die Form:

$$0000000d0f746b7b0f4404cce1b61eec7dc4c65622d80647e46548e1404283d6 \langle DATA \rangle \langle NONCE \rangle$$

Die eigentliche Aufgabe besteht aus mehreren asynchronen Teilen:

1. Führen Sie die Geschichte beliebig (aber juristisch rechtskonform) weiter. Wählen Sie dazu einen beliebigen möglichst passenden Satz. Leisten Sie anschließend den Proof-Of-Work. Sobald Sie einen PoW ermittelt haben, fügen Sie den Block als String in smart-tale hinzu. Seien Sie dabei schnell, denn falls in der Zwischenzeit die Blockchain fortgesetzt wurde, müssen Sie den PoW neu bestimmen.

2. Wurde ein neuer Block hinzugefügt, sollte er vom Netzwerk (das sind Sie und alle anderen Studierenden) akzeptiert oder abgelehnt werden. Nutzen Sie dafür die Reaktionsfunktion in Discord, um möglichst eindeutig einen Block zu bestätigen oder ihn als ungültig zu markieren.

Tipps:

- Wir empfehlen, ein kleines Skript zu programmieren, das in kurzer Zeit viele Nonces ausprobiert und die erstbeste gültige Nonce zurückwirft.
- In der Praxis schließen sich einzelne Netzwerkteilnehmer oft zusammen, um den Suchraum aufzuteilen und sich die Belohnung zu teilen (hier den Ruhm, einen Satz unterbringen zu dürfen).
- Für das Validieren eines Blocks gibt es bereits viele Onlinetools, die zu einem String den SHA256 Hashwert bestimmen. Hier ist Selbstprogrammieren nicht unbedingt nötig. In Python aber auch kein komplizierter Code: def block2hash(string): return sha256(string.encode('utf-8')).hexdigest()

Was sind die Vorteile dieser Datenstruktur gegenüber einer simplen Liste (siehe Vorlesung, Kapitel 1)?
Was fällt auf beim Vergleich der beiden Tasks Nonce-Mining vs. Blockvalidierung?
Welche Faktoren sollten in die Wahl des Parameters k eingehen?