Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

June 10, 2022

Lehrstuhl Informatik 2
Fakultät für Informatik

Hashing

Die Problemstellung

- Arrays sind zusammenhängende Speicherblöcke, die direkt addressiert werden
- häufig werden aber Datenstrukturen benötigt, deren Elemente durch besondere Schlüssel adressiert werden können
- Beispiele: Matrikelnummern, Wörterbücher, Telefonverzeichnisse
- wir nehmen an, daß die verschiedenen Elemente verschiedene Schlüssel haben
- Hashtabellen stellen eine entsprechende Datenstruktur bereit

Hashing

Einfach verkettete Listen

- wir wiederholen *verkettete Listen*
- jeder Listeneintrag enthält einen Zeiger auf den nächsten Eintrag
- die Abwesenheit eines Nachfolgers wird durch einen besonderen Wert `NULL` gekennzeichnet
- neue Elemente können in Zeit $O(1)$ eingefügt werden
- Elemente können in Zeit $O(1)$ gelöscht werden
- Durchsuchen der Liste kostet Zeit $\Theta(n)$
- zwei Listen können in Zeit $O(1)$ vereinigt werden

Hashing

Doppelt verkettete Listen

- jeder Listeneintrag enthält zusätzlich einen Zeiger auf den Vorgänger
- eine Liste kann in Zeit $O(1)$ an einer bestimmten Stelle aufgeteilt werden
- unser “Wörterbuchproblem” kann mit einer verketten Liste gelöst werden
- allerdings ist $\Theta(n)$ Zeit notwendig, um Einträge aufzufinden

Hashing

Direkte Adressierung

- sei $\mathcal{S} = \{s_1, \ldots, s_k\}$ die Menge der Schlüssel
- wenn $\mathcal{S}$ "klein" ist, können wir einfach ein Array anlegen, das für jeden möglichen Schlüssel einen Eintrag bereitstellt
- in dieser Speicherstelle legen wir den entsprechenden Eintrag ab, wenn einer vorhanden ist; sonst `NULL`
- Speicherbedarf: $\Theta(k)$; ok für "kleine" k
- Zugriffszeit: $O(1)$ (speichern/abrufen/löschen)

Hashing

Hashtabellen

- für große Schlüsselmengen kommt direkte Adressierung nicht infrage
- stattdessen verwenden wir Hashing
- eine Hashfunktion

$$\mathcal{H} : \mathcal{S} \to [m] = \{1, \ldots, m\}$$

  bildet die Schlüssel auf eine (kleiner) Menge $[m]$ ab
- die Zahlen $1, \ldots, m$ verwenden wir, um ein Array zu adressieren

Hashing

Hashtabellen

- wenn $m < |\mathcal{S}|$, kann es zu *Kollisionen* kommen
- deshalb speichern wir die Einträge nicht einfach in einem Array ab
- stattdessen legen wir m einfach verkettete Listen $\boldsymbol{L} = (L_1, \ldots, L_m)$ an
- Liste L_i speichert Elemente mit Hash i
- die Datenstruktur erlaubt Operationen `Insert`, `Search` und `Delete`

Hashing

Hashtabellen

`Insert` Um ein neues Element e mit Schlüssel s in die Hashtabelle einzufügen, berechnen wir zunächst den Hash $\mathcal{H}(s)$. Dann wird das Element in die Liste $L_{\mathcal{H}(s)}$ eingefügt. Diese Operationen kann in Laufzeit $O(1)$ durchgeführt werden (wenn wir die Berechnung der Hashfunktion als eine Operation zählen).

`Search` Um ein Element mit einem gegeben Schlüssel s zu finden, bestimmen wir den Hash $\mathcal{H}(s)$. Anschließend durchsuchen wir die Liste $L_{\mathcal{H}(s)}$ nach einem Element mit Schlüssel s. Die Laufzeit für diese Operation ist die Länge der Liste $L_{\mathcal{H}(s)}$.

`Delete` Auch diese Operation kann in Zeit $L_{\mathcal{H}(s)}$ ausgeführt werden, wenn s der Schlüssel des zu löschenden Elements ist.

Hashing

Hashfunktionen

- wir konstruieren wir gute Hashfunktionen?
- für jede *deterministische* Hashfunktion kann die Hashtabelle schlimmstenfalls zu einer verketteten Liste degenerieren
- dennoch werden oft einfache deterministische Hashfunktionen eingesetzt, weil sie schnell zu berechnen sind
- zufällige Hashfunktionen sind eine attraktive, beweisbar gute Alternative

Hashing

Die Multiplikationsmethode

- Annahme: die Schlüssel s_i sind natürliche Zahlen
- wir wählen eine reelle Zahl $0 < \alpha < 1$, z.B.

$$\alpha = \frac{\sqrt{5}-1}{2}$$

und definieren

$$\mathcal{H}(s) = \lceil m \cdot (s\alpha - \lfloor s\alpha \rfloor) \rceil \in \{1, \ldots, m\}$$

- Vorsicht: diese Methode ist “heuristisch”

Hashing

Definition **[Universelle Hashfunktionen]**

Eine Folge $\mathfrak{H} = (\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ von Hashfunktionen

$$\mathcal{H}_i : \mathcal{S} \to [m]$$

heißt universell, falls für je zwei Schlüssel $s, s' \in \mathcal{S}$, $s \neq s'$, gilt

$$|\{i \in [\ell] : \mathcal{H}_i(s) = \mathcal{H}_i(s')\}| \leq \frac{\ell}{m} \tag{1}$$

Hashing

Satz

- Angenommen n Elemente werden in einer Hashtabelle gespeichert.
- Angenommen $\mathfrak{H}$ ist eine universelle Folge von Hashfunktionen.
- Wenn $\mathcal{H}$ ist ein zufälliges Element von $\mathfrak{H}$ ist, hat für jeden Schlüssel $k \in \mathcal{S}$ die Liste $L_{h(k)}$ erwartete Länge $\frac{n}{m} + O(1)$.

Die Erwartung bezieht sich nur auf die Wahl von $\mathcal{H}$.

Hashing

Division mit Rest

- für $a, b \in \mathbb{Z}$, $b \neq 0$, existieren ganze Zahlen q und $0 \leq r < |b|$, so daß

$$a = q \cdot b + r.$$

- wir nennen r den *Rest* von a bei Division durch b und schreiben

$$r = a \mod b \qquad \text{[“modulo”]}$$

- b *teilt* a, falls $r = 0$. *Schreibweise:* $b|a$.

Hashing

Der größte gemeinsame Teiler

- für $a, b \in \mathbb{Z}$ ist $\mathrm{ggT}(a, b)$ die größte Zahl $c \in \mathbb{N}$, so daß $c \mid a$ und $c \mid b$
- jede Zahl teilt die Null
- falls $a = b = 0$ definieren wir daher $\mathrm{ggT}(a, b) = \infty$
- für $a, b \in \mathbb{N}$ bestimmt der Algorithmus `Euclid` den größten gemeinsamen Teiler

Hashing

$\texttt{Euclid}(a, b)$

1. falls $a < b$, vertausche a und b
2. setze $a_0 = a$, $a_1 = b$, $i = 1$.
3. solange $a_i > 0$
4. berechne $q_i \in \mathbb{Z}$, $a_{i+1} \in \{0, 1, \ldots, a_i\}$, so daß $a_{i-1} = q_i a_i + a_{i+1}$.
5. erhöhe i um 1
6. gib a_{i-1} aus

Hashing

Satz

Für $a, b \in \mathbb{N}$ gibt `Euclid` den $\mathrm{ggT}(a, b)$ aus und hat Laufzeit $O(\log(a+b))$.

- der Algorithmus ist effizient
- denn die Länge der Eingabe ist $\Theta(\log(a+b))$

Hashing

Korollar

Für je zwei Zahlen $a, b \in \mathbb{N}$ gibt es Zahlen $u, v \in \mathbb{Z}$, so daß $\mathrm{ggT}(a, b) = au + bv$.

Hashing

Konstruktion universeller Hashfunktionen

- sei $m > 1$ eine natürliche Zahl und $p > m$ eine Primzahl
- für ganze Zahlen $1 \leq a < p$ und $0 \leq b < p$ definiere

$$\mathcal{H}_{a,b} : \{0, \ldots, p-1\} \to \{0, \ldots, m-1\}, \qquad k \mapsto ((a \cdot k + b) \mod p) \mod m.$$

- sei $\mathfrak{H}_{p,m} = (\mathcal{H}_{a,b})_{a,b}$.

Hashing

Satz

Die Menge $\mathfrak{H}_{p,m}$ von Hashfunktionen $\{0, 1, \ldots, p-1\} \to \{0, 1, \ldots, m-1\}$ ist universell.

Hashing

Zusammenfassung

- das Hashingproblem tritt in der Praxis häufig auf
- die Multiplikationsmethode bietet eine einfache heuristische Lösung
- mit Hilfe der Zahlentheorie haben wir universelle Hashfunktionen konstruiert