14. Hashing

Hashtabellen, Pre-Hashing, Hashing, Kollisionsauflösung durch Verketten, Einfaches gleichmässiges Hashing, Gebräuchliche Hashfunktionen, Tabellenvergrösserung, offene Addressierung: Sondieren, Gleichmässiges Hashing, Universelles hashing, Perfektes Hashing [Ottman/Widmayer, Kap. 4.1-4.3.2, 4.3.4, Cormen et al, Kap. 11-11.4]

Motivierendes Beispiel

Ziel: Effiziente Verwaltung einer Tabelle aller n ETH-Studenten

Mögliche Anforderung: Schneller Zugriff (Einfügen, Löschen, Finden) von Datensätzen nach Name.

Wörterbuch (Dictionary)

Abstrakter Datentyp (ADT) D zur Verwaltung einer Menge von Einträgen[18] i mit Schlüsseln $k \in \mathcal{K}$. Operationen

- **`D.insert`**(i): Hinzufügen oder Überschreiben von i im Wörterbuch D.
- **`D.delete`**(i): Löschen von i aus dem Wörterbuch D. Nicht vorhanden $\Rightarrow$ Fehlermeldung.
- **`D.search`**(k): Liefert Eintrag mit Schlüssel k, wenn er existiert.

[18] Schlüssel-Wert Paare (k, v), im Folgenden betrachten wir hauptsächlich die Schlüssel.

Wörterbuch in C++

Assoziativer Container `std::unordered_map<>`

```
// Create an unordered_map of strings that map to strings
std::unordered_map<std::string, std::string> u = {
  {"RED","#FF0000"}, {"GREEN","#00FF00"}
};

u["BLUE"] = "#0000FF"; // Add

std::cout << "The HEX of color RED is: " << u["RED"] << "\n";

for( const auto& n : u ) // iterate over key-value pairs
  std::cout << n.first << ":" << n.second << "\n";
```

Motivation / Verwendung

Wahrscheinlich **die** gängigste Datenstruktur

- Unterstützt in vielen Programmiersprachen (C++, Java, Python, Ruby, Javascript, C# ...)
- Offensichtliche Verwendung
    - Datenbanken / Tabellenkalkulation
    - Symboltabellen in Compilern und Interpretern
- Weniger offensichtlich
    - Substring Suche (Google, grep)
    - Ähnlichkeit von Texten (Dokumentenvergleich, DNA)
    - Dateisynchronisation
    - Kryptographie: Filetransfer / Identifikation

1. Idee: Direkter Zugriff (Array)

Index	Eintrag
0	–
1	–
2	–
3	[3,wert(3)]
4	–
5	–
⋮	⋮
k	[k,wert(k)]
⋮	⋮

Probleme

1. Idee: Direkter Zugriff (Array)

Index	Eintrag
0	–
1	–
2	–
3	[3,wert(3)]
4	–
5	–
⋮	⋮
k	[k,wert(k)]
⋮	⋮

Probleme

1. Schlüssel müssen nichtnegative ganze Zahlen sein

1. Idee: Direkter Zugriff (Array)

Index	Eintrag
0	–
1	–
2	–
3	[3,wert(3)]
4	–
5	–
$\vdots$	$\vdots$
k	[k,wert(k)]
$\vdots$	$\vdots$

Probleme

1. Schlüssel müssen nichtnegative ganze Zahlen sein
2. Grosser Schlüsselbereich $\Rightarrow$ grosses Array

Lösung zum ersten Problem: Pre-hashing

Prehashing: Bilde Schlüssel ab auf positive Ganzzahlen mit einer Funktion $ph : \mathcal{K} \to \mathbb{N}$

- Theoretisch immer möglich, denn jeder Schlüssel ist als Bitsequenz im Computer gespeichert
- Theoretisch auch: $x = y \Leftrightarrow ph(x) = ph(y)$
- In der Praxis: APIs bieten Funktionen zum pre-hashing an. (Java: **`object.hashCode()`**, C++: **`std::hash<>`**, Python: **`hash(object)`**)
- APIs bilden einen Schlüssel aus der Schlüsselmenge ab auf eine Ganzzahl mit beschränkter Grösse.[19]

[19]Somit gilt die Implikation $ph(x) = ph(y) \Rightarrow x = y$ **nicht** mehr für alle x,y.

Prehashing Beispiel: String

Zuordnung Name $s = s_1 s_2 \dots s_{l_s}$ zu Schlüssel

$$ph(s) = \left(\sum_{i=0}^{l_s - 1} s_{l_s - i} \cdot b^i \right) \bmod 2^w$$

b so, dass verschiedene Namen möglichst verschiedene Schlüssel erhalten.

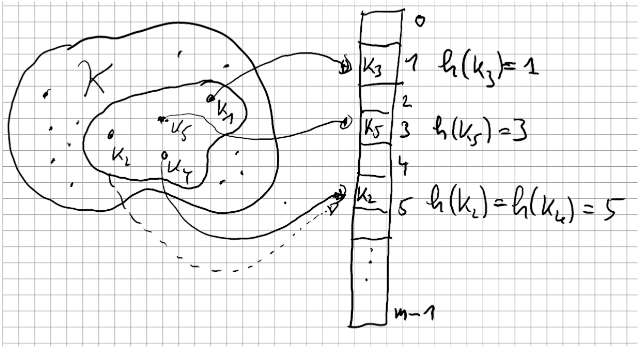
w Wortgrösse des Systems (z.B. 32 oder 64).

Beispiel (Java), mit $b = 31$, $w = 32$ Ascii-Werte s_i.

Anna $\mapsto 2045632$
Jacqueline $\mapsto 42042089953442505 \bmod 2^{32} = 507919049$

Lösung zum zweiten Problem: Hashing

Reduziere des Schlüsseluniversum: Abbildung (Hash-Funktion) $h : \mathcal{K} \rightarrow \{0, ..., m-1\}$ ($m \approx n =$ Anzahl Einträge in der Tabelle)



Kollision: $h(k_i) = h(k_j)$.

Nomenklatur

Hashfunktion h: Abbildung aus der Menge der Schlüssel $\mathcal{K}$ auf die Indexmenge $\{0, 1, \ldots, m-1\}$ eines Arrays (**Hashtabelle**).

$$h : \mathcal{K} \to \{0, 1, \ldots, m-1\}.$$

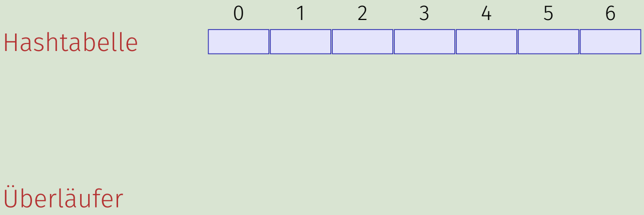
Meist $|\mathcal{K}| \gg m$. Es gibt also $k_1, k_2 \in \mathcal{K}$ mit $h(k_1) = h(k_2)$ (**Kollision**).
Eine Hashfunktion sollte die Menge der Schlüssel möglichst gleichmässig auf die Positionen der Hashtabelle verteilen.

Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12

Direkte Verkettung der Überläufer

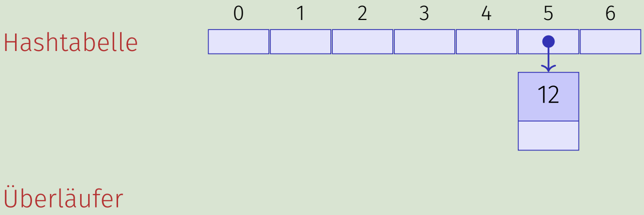


Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55

Direkte Verkettung der Überläufer

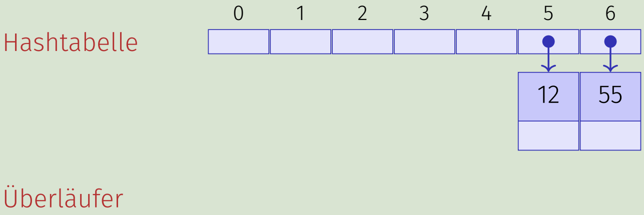


Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5

Direkte Verkettung der Überläufer



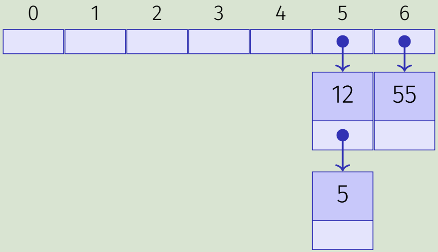
Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15

Direkte Verkettung der Überläufer

Hashtabelle



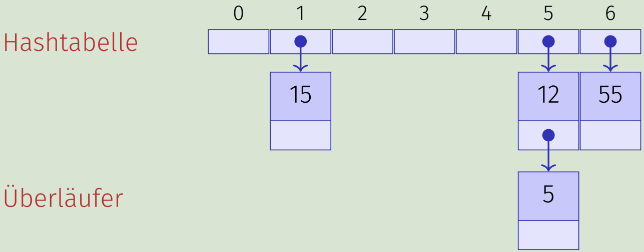
Überläufer

Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2

Direkte Verkettung der Überläufer

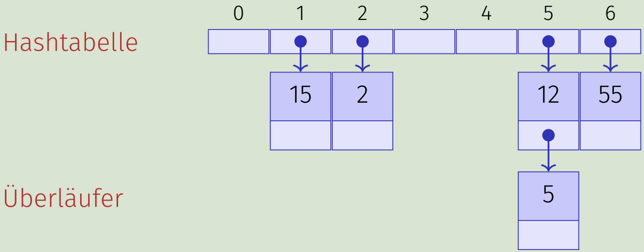


Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

Direkte Verkettung der Überläufer

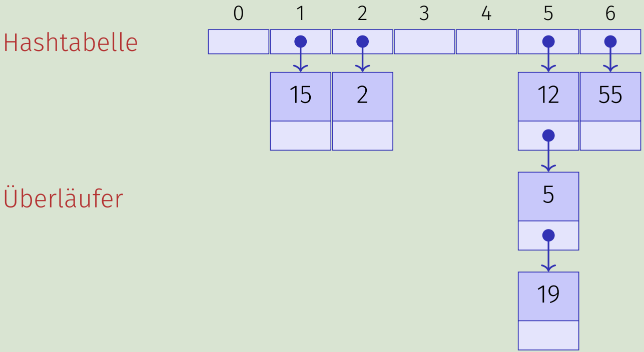


Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19 , 43

Direkte Verkettung der Überläufer

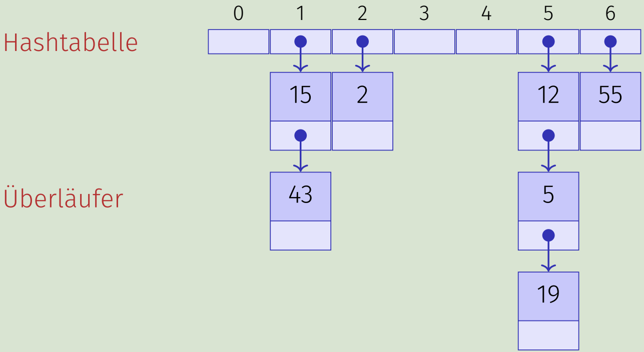


Behandlung von Kollisionen: Verkettung

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19 , 43

Direkte Verkettung der Überläufer



Algorithmen zum Hashing mit Verkettung

- $\texttt{insert}(i)$ Prüfe ob Schlüssel k vom Eintrag i in Liste an Position $h(k)$. Falls nein, füge i am Ende der Liste ein; andernfalls ersetze das Element durch i.
- $\texttt{find}(k)$ Prüfe ob Schlüssel k in Liste an Position $h(k)$. Falls ja, gib die Daten zum Schlüssel k zurück. Andernfalls Rückgabe eines leeren Elements **`null`**.
- $\texttt{delete}(k)$ Durchsuche die Liste an Position $h(k)$ nach k. Wenn Suche erfolgreich, entferne das entsprechende Listenelement.

Worst-case Analyse

Schlechtester Fall: alle Schlüssel werden auf den gleichen Index abgebildet.

$\Rightarrow \Theta(n)$ pro Operation im schlechtesten Fall. ☹

Einfaches Gleichmässiges Hashing

Starke Annahmen: Jeder beliebige Schlüssel wird

- mit gleicher Wahrscheinlichkeit (Uniformität)
- und unabhängig von den anderen Schlüsseln (Unabhängigkeit)

auf einen der m verfügbaren Slots abgebildet.

Einfaches Gleichmässiges Hashing

Unter der Voraussetzung von einfachem gleichmässigen Hashing:
Erwartete Länge einer Kette, wenn n Elemente in eine Hashtabelle mit m Elementen eingefügt werden

$$\mathbb{E}(\text{Länge Kette j}) = \mathbb{E}\left(\sum_{i=0}^{n-1} \mathbb{1}(k_i = j)\right) = \sum_{i=0}^{n-1} \mathbb{P}(k_i = j)$$
$$= \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m}$$

$\alpha = n/m$ heisst **Belegungsfaktor** oder **Füllgrad** der Hashtabelle.

Einfaches Gleichmässiges Hashing

Theorem 16

Sei eine Hashtabelle Verkettung gefüllt mit Füllgrad $\alpha = \frac{n}{m} < 1$. Unter der Annahme vom einfachen gleichmässigen Hashing hat die nächste Operation erwartete Laufzeitkosten von $\leq 1 + \alpha$.

Folgerung: ist die Anzahl der Slots m der Hashtabelle immer mindestens proportional zur Anzahl Elemente n in der Hashtabelle, $n \in \mathcal{O}(m) \Rightarrow$ Erwartete Laufzeit der Operationen Suchen, Einfügen und Löschen ist $\mathcal{O}(1)$.

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche.

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche. Durchschnittliche Listenlänge ist $\alpha = \frac{n}{m}$. Liste muss ganz durchlaufen werden.

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche. Durchschnittliche Listenlänge ist $\alpha = \frac{n}{m}$. Liste muss ganz durchlaufen werden.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge

$$C'_n = \alpha.$$

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche. Durchschnittliche Listenlänge ist $\alpha = \frac{n}{m}$. Liste muss ganz durchlaufen werden.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge
   $$C'_n = \alpha.$$
2. Erfolgreiche Suche. Betrachten die Einfügehistorie: Schlüssel j sieht durchschnittliche Listenlänge $(j-1)/m$.

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche. Durchschnittliche Listenlänge ist $\alpha = \frac{n}{m}$. Liste muss ganz durchlaufen werden.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge
   $$C'_n = \alpha.$$
2. Erfolgreiche Suche. Betrachten die Einfügehistorie: Schlüssel j sieht durchschnittliche Listenlänge $(j-1)/m$.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge
   $$C_n = \frac{1}{n}\sum_{j=1}^{n}(1 + (j-1)/m)) \qquad .$$

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche. Durchschnittliche Listenlänge ist $\alpha = \frac{n}{m}$. Liste muss ganz durchlaufen werden.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge

$$C'_n = \alpha.$$

2. Erfolgreiche Suche. Betrachten die Einfügehistorie: Schlüssel j sieht durchschnittliche Listenlänge $(j-1)/m$.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge

$$C_n = \frac{1}{n}\sum_{j=1}^{n}(1+(j-1)/m) = 1 + \frac{1}{n}\frac{n(n-1)}{2m} \qquad .$$

Weitere Analyse (direkt verkettete Liste)

1. Erfolglose Suche. Durchschnittliche Listenlänge ist $\alpha = \frac{n}{m}$. Liste muss ganz durchlaufen werden.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge
   $$C'_n = \alpha.$$

2. Erfolgreiche Suche. Betrachten die Einfügehistorie: Schlüssel j sieht durchschnittliche Listenlänge $(j-1)/m$.
   $\Rightarrow$ Durchschnittliche Anzahl betrachteter Einträge
   $$C_n = \frac{1}{n}\sum_{j=1}^{n}(1 + (j-1)/m)) = 1 + \frac{1}{n}\frac{n(n-1)}{2m} \approx 1 + \frac{\alpha}{2}.$$

Vor und Nachteile der Verkettung

Vorteile der Strategie:

- Belegungsfaktoren $\alpha > 1$ möglich
- Entfernen von Schlüsseln einfach

Nachteile

- Speicherverbrauch der Verkettung

Beispiele gebräuchlicher Hashfunktionen

$$h(k) = k \bmod m$$

Ideal: m Primzahl, nicht zu nahe bei Potenzen von 2 oder 10
Aber oft: $m = 2^k - 1$ $(k \in \mathbb{N})$

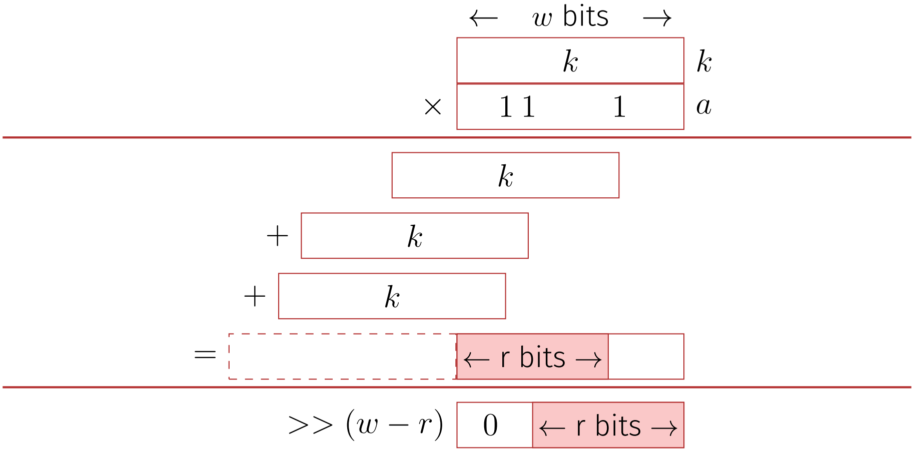
Beispiele gebräuchlicher Hashfunktionen

Multiplikationsmethode

$$h(k) = \left\lfloor (a \cdot k \bmod 2^w) / 2^{w-r} \right\rfloor \bmod m$$

- $m = 2^r$, $w =$ Grösse des Maschinenworts in Bits.
- Multiplikation addiert k entlang aller Bits von a, Ganzzahldivision mit 2^{w-r} und $\mathrm{mod} m$ extrahiert die oberen r Bits.
- Als Code geschrieben: `a * k >> (w-r)`
- Guter Wert für a: $\left\lfloor \frac{\sqrt{5}-1}{2} \cdot 2^w \right\rfloor$: Integer, der die ersten w Bits des gebrochenen Teils der irrationalen Zahl darstellt.

Illustration



Tabellenvergrösserung

- Wissen nicht a priori, wie gross n sein wird.
- Benötigen $m = \Theta(n)$ zu jeder Zeit.

Grösse der Tabelle muss angepasst werden. Hash-Funktion ändert sich $\Rightarrow$ **Rehashing**

- Alloziere Array A' mit Grösse $m' > m$
- Füge jeden Eintrag von A erneut in A' ein (mit erneutem Hashing)
- Setze $A \leftarrow A'$.
- Kosten: $\mathcal{O}(n + m + m')$.

Wie wählt man m'?

Tabellenvergrösserung

- 1.Idee $n = m \Rightarrow m' \leftarrow m + 1$
  Bei jedem Einfügen vergrössern. Kosten $\Theta(1 + 2 + 3 + \cdots + n) = \Theta(n^2)$ ☹
- 2.Idee $n = m \Rightarrow m' \leftarrow 2m$ Vergrössern nur wenn $m = 2^i$:
  $\Theta(1 + 2 + 4 + 8 + \cdots + n) = \Theta(n)$
  Einige Einfügeoperationen kosten lineare Zeit, aber im Durchschnitt kosten sie $\Theta(1)$ ☺

Jede Operation vom Hashing mit Verketten hat erwartet amortisierte Kosten $\Theta(1)$.
($\Rightarrow$ Amortisierte Analyse)

Offene Addressierung

Speichere die Überläufer direkt in der Hashtabelle mit einer **Sondierungsfunktion** $s : \mathcal{K} \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$
Tabellenposition des Schlüssels entlang der **Sondierungsfolge**

$$S(k) := (s(k,0), s(k,1), \ldots, s(k, m-1)) \mod m$$

Sondierungsfolge muss für jedes $k \in \mathcal{K}$ eine Permutation sein von $\{0, 1, \ldots, m-1\}$

Begriffsklärung: Dieses Verfahren nutzt **offene Addressierung** (Positionen in der Hashtabelle nicht fixiert), ist aber **ein geschlossenes Hashverfahren** (Einträge bleiben in der Hashtabelle)

Algorithmen zur offenen Addressierung

- **`insert`**(i) Suche Schlüssel k von i in der Tabelle gemäss Sondierungssequenz $S(k)$. Ist k nicht vorhanden, füge k an die erste freie Position in der Sondierungsfolge ein. Andernfalls Fehlermeldung.
- **`find`**(k) Durchlaufe Tabelleneinträge gemäss $S(k)$. Wird k gefunden, gib die zu k gehörenden Daten zurück. Andernfalls Rückgabe eines leeres Elements **`null`**.
- **`delete`**(k) Suche k in der Tabelle gemäss $S(k)$. Wenn k gefunden, ersetze k durch den speziellen Schlüssel **`removed`**.

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$$

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12

0	1	2	3	4	5	6

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \dots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55

0	1	2	3	4	5	6
					12	

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \dots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15

0	1	2	3	4	5	6
5					12	55

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
5	15				12	55

Lineares Sondieren

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2			12	55

Lineares Sondieren

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \ldots, h(k) + m - 1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \mod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12	55

Diskussion

Diskussion

Beispiel $\alpha = 0.95$

Erfolglose Suche betrachtet im Durchschnitt 200 Tabelleneinträge! (hier ohne Herleitung).

Diskussion

Beispiel $\alpha = 0.95$

Erfolglose Suche betrachtet im Durchschnitt 200 Tabelleneinträge! (hier ohne Herleitung).

Grund für die schlechte Performance?

Diskussion

Beispiel $\alpha = 0.95$

Erfolglose Suche betrachtet im Durchschnitt 200 Tabelleneinträge! (hier ohne Herleitung).

Grund für die schlechte Performance?

Primäre Häufung: Ähnliche Hashaddressen haben ähnliche Sondierungsfolgen $\Rightarrow$ lange zusammenhängende belegte Bereiche.

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Quadratisches Sondieren

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12

0	1	2	3	4	5	6

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7, \mathcal{K} = \{0, \dots, 500\}, h(k) = k \bmod m.$

Schlüssel 12 , 55

0	1	2	3	4	5	6
					12	

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7, \mathcal{K} = \{0, \dots, 500\}, h(k) = k \bmod m.$

Schlüssel 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15

0	1	2	3	4	5	6
				5	12	55

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
	15			5	12	55

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
	15	2		5	12	55

Quadratisches Sondieren

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2		5	12	55

Diskussion

Beispiel $\alpha = 0.95$

Erfolglose Suche betrachtet im Durchschnitt 22 Tabelleneinträge (hier ohne Herleitung)

Diskussion

Beispiel $\alpha = 0.95$

Erfolglose Suche betrachtet im Durchschnitt 22 Tabelleneinträge (hier ohne Herleitung)

Grund für die schlechte Performance?

Diskussion

Beispiel $\alpha = 0.95$

Erfolglose Suche betrachtet im Durchschnitt 22 Tabelleneinträge (hier ohne Herleitung)

Grund für die schlechte Performance?

Sekundäre Häufung: Synonyme k und k' (mit $h(k) = h(k')$) durchlaufen dieselbe Sondierungsfolge.

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k)+h'(k), h(k)+2h'(k), \dots, h(k)+(m-1)h'(k)) \mod m$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Schlüssel 12

0	1	2	3	4	5	6

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m-1)h'(k)) \mod m$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Schlüssel 12 , 55

0	1	2	3	4	5	6
					12	

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$$S(k) = (h(k), h(k)+h'(k), h(k)+2h'(k), \ldots, h(k)+(m-1)h'(k)) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Schlüssel 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \mod 7$, $h'(k) = 1 + k \mod 5$.

Schlüssel 12 , 55 , 5 , 15

0	1	2	3	4	5	6
5					12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Schlüssel 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
5	15				12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2			12	55

Double Hashing

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Schlüssel 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12	55

Double Hashing

- Sondierungsreihenfolge muss Permutation aller Hashadressen bilden. Also $h'(k) \neq 0$ und $h'(k)$ darf m nicht teilen, z.B. garantiert mit m prim.
- h' sollte möglichst unabhängig von h sein (Vermeidung sekundärer Häufung).

Unabhängigkeit:

$$\mathbb{P}((h(k) = h(k')) \ \wedge \ (h'(k) = h'(k'))) = \mathbb{P}(h(k) = h(k')) \cdot \mathbb{P}(h'(k) = h'(k')).$$

Unabhängigkeit weitgehend erfüllt von $h(k) = k \bmod m$ und $h'(k) = 1 + k \bmod (m-2)$ (m prim).

Gleichmässiges Hashing

Starke Annahme: Die Sondierungssequenz $S(k)$ eines Schlüssels k ist mit gleicher Wahrscheinlichkeit eine der $m!$ vielen Permutationssequenzen von $\{0, 1, \ldots, m-1\}$.

(Double Hashing kommt dem am ehesten nahe)

Analyse gleichmässiges Hashing mit offener Addressierung

Theorem 17

Sei eine Hashtabelle mit offener Addressierung gefüllt mit Füllgrad $\alpha = \frac{n}{m} < 1$*. Unter der Annahme vom gleichmässigen Hashing hat die nächste Operation erwartete Laufzeitkosten von* $\leq \frac{1}{1-\alpha}$*.*

Analyse gleichmässiges Hashing mit offener Addressierung

Beweis des Theorems: Zufallsvariable X: Anzahl Sondierungen bei einer erfolglosen Suche.

$$\mathbb{P}(X \geq i) \stackrel{*}{=} \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2}$$
$$\stackrel{**}{\leq} \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}. \qquad (1 \leq i \leq m)$$

*: A_j:Slot beim j-ten Schritt belegt.
$\mathbb{P}(A_1 \cap \cdots \cap A_{i-1}) = \mathbb{P}(A_1) \cdot \mathbb{P}(A_2|A_1) \cdot \ldots \cdot \mathbb{P}(A_{i-1}|A_1 \cap \cdots \cap A_{i-2})$,
**: $\frac{n-1}{m-1} < \frac{n}{m}$ da[20] $n < m$.
Ausserdem $\mathbb{P}(x \geq i) = 0$ für $i \geq m$. Also

$$\mathbb{E}(X) \stackrel{\text{Anhang}}{=} \sum_{i=1}^{\infty} \mathbb{P}(X \geq i) \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}.$$

[20] $\frac{n-1}{m-1} < \frac{n}{m} \Leftrightarrow \frac{n-1}{n} < \frac{m-1}{m} \Leftrightarrow 1 - \frac{1}{n} < 1 - \frac{1}{m} \Leftrightarrow n < m \; (n > 0, m > 0)$

Übersicht

	$\alpha = 0.50$		$\alpha = 0.90$		$\alpha = 0.95$	
	C_n	C'_n	C_n	C'_n	C_n	C'_n
(Direkte) Verkettung	1.25	0.50	1.45	0.90	1.48	0.95
Lineares Sondieren	1.50	2.50	5.50	50.50	10.50	200.50
Quadratisches Sondieren	1.44	2.19	2.85	11.40	3.52	22.05
Gleichmässiges Hashing	1.39	2.00	2.56	10.00	3.15	20.00

: C_n: Anzahl Schritte erfolgreiche Suche, C'_n: Anzahl Schritte erfolglose Suche, Belegungsgrad α.

Universelles Hashing

- $|\mathcal{K}| > m \Rightarrow$ Menge "ähnlicher Schlüssel" kann immer so gewählt sein, so dass überdurchschnittlich viele Kollisionen entstehen.
- Unmöglich, einzelne für alle Fälle "beste" Hashfunktion auszuwählen.
- Jedoch möglich[21]: randomisieren!

Universelle Hashklasse $\mathcal{H} \subseteq \{h : \mathcal{K} \to \{0, 1, \ldots, m-1\}\}$ ist eine Familie von Hashfunktionen, so dass

$$\forall\, k_1 \neq k_2 \in \mathcal{K} \text{ gilt } |\{h \in \mathcal{H} \text{ mit } h(k_1) = h(k_2)\}| \leq \frac{|\mathcal{H}|}{m}.$$

[21] Ähnlich wie beim Quicksort

Universelles Hashing

Theorem 19

Eine aus einer universellen Klasse $\mathcal{H}$ von Hashfunktionen zufällig gewählte Funktion $h \in \mathcal{H}$ verteilt im Erwartungswert eine beliebige Folge von Schlüsseln aus $\mathcal{K}$ so gleichmässig wie nur möglich auf die verfügbaren Plätze.

Beim Hashing mit Verketten ist die erwartete Kettenlänge für ein nicht enthaltenes Element $\leq \alpha = n/m$. Die erwartete Kettenlänge für ein enthaltenes Element ist $\leq 1 + \alpha$.

Universelles Hashing

Vorbemerkung zum Beweis des Theorems.

Definiere mit $x, y \in \mathcal{K}$, $h \in \mathcal{H}$, $Y \subseteq \mathcal{K}$:

$$\delta(h, x, y) = \begin{cases} 1, & \text{falls } h(x) = h(y) \\ 0, & \text{sonst,} \end{cases}$$

ist $h(x) = h(y)$ (0 oder 1)?

$$\delta(h, x, Y) = \sum_{y \in Y} \delta(x, y, h),$$

für viele $y \in Y$ ist $h(x) = h(y)$?

$$\delta(\mathcal{H}, x, y) = \sum_{h \in \mathcal{H}} \delta(x, y, h)$$

für wie viele $h \in \mathcal{H}$ ist $h(x) = h(y)$?.

$\mathcal{H}$ ist universell, wenn für alle $x, y \in \mathcal{K}$, $x \neq y : \delta(\mathcal{H}, x, y) \leq |\mathcal{H}|/m$.

Universelles Hashing

Beweis des Theorems

$S \subseteq \mathcal{K}$: bereits gespeicherte Schlüssel. x wird hinzugefügt: ($x \notin S$)

Erwartete Anzahl Kollisionen von x mit S

$$\begin{aligned}
\mathbb{E}_{\mathcal{H}}(\delta(h,x,S)) &= \sum_{h\in\mathcal{H}} \delta(h,x,S)/|\mathcal{H}| \\
&= \frac{1}{|\mathcal{H}|}\sum_{h\in\mathcal{H}}\sum_{y\in S}\delta(h,x,y) = \frac{1}{|\mathcal{H}|}\sum_{y\in S}\sum_{h\in\mathcal{H}}\delta(h,x,y) \\
&= \frac{1}{|\mathcal{H}|}\sum_{y\in S}\delta(\mathcal{H},x,y) \\
&\leq \frac{1}{|\mathcal{H}|}\sum_{y\in S}\frac{|\mathcal{H}|}{m} = \frac{|S|}{m} = \alpha.
\end{aligned}$$

■

Universelles Hashing

$S \subseteq \mathcal{K}$: bereits gespeicherte Schlüssel, nun $x \in S$.

Erwartete Anzahl Kollisionen von x mit S

$$
\begin{aligned}
\mathbb{E}_{\mathcal{H}}(\delta(x,S,h)) &= \sum_{h\in\mathcal{H}} \delta(x,S,h)/|\mathcal{H}| \\
&= \frac{1}{|\mathcal{H}|}\sum_{h\in\mathcal{H}}\sum_{y\in S}\delta(h,x,y) = \frac{1}{|\mathcal{H}|}\sum_{y\in S}\sum_{h\in\mathcal{H}}\delta(h,x,y) \\
&= \frac{1}{|\mathcal{H}|}\left(\delta(\mathcal{H},x,x) + \sum_{y\in S-\{x\}}\delta(\mathcal{H},x,y)\right) \\
&\le \frac{1}{|\mathcal{H}|}\left(|\mathcal{H}| + \sum_{y\in S-\{x\}}|\mathcal{H}|/m\right) = 1 + \frac{|S|-1}{m} = 1 + \frac{n-1}{m} \le 1+\alpha.
\end{aligned}
$$

$\blacksquare$

Konstruktion Universelle Hashklasse

Sei Schlüsselmenge $\mathcal{K} = \{0, \dots, u-1\}$ und $p \geq u$ Primzahl.. Mit $a \in \mathcal{K} \setminus \{0\}$, $b \in \mathcal{K}$ definiere

$$h_{ab} : \mathcal{K} \to \{0, \dots, m-1\}, h_{ab}(x) = ((ax + b) \bmod p) \bmod m.$$

Dann gilt

Theorem 20

Die Klasse $\mathcal{H} = \{h_{ab} | a, b \in \mathcal{K}, a \neq 0\}$ ist eine universelle Klasse von Hashfunktionen.

(Hier ohne Beweis. Siehe z.B. Cormen et al, Kap. 11.3.3)

Perfektes Hashing

Ist im Vorhinein die Menge der verwendeten Schlüssel bekannt? Dann kann die Hashfunktion perfekt, also kollisionsfrei, gewählt werden.

Beispiel: Tabelle der Schlüsselwörter in einem Compiler.

Beobachtung (Geburtstagsparadoxon umgekehrt)

- h zufällig gewählt aus universeller Hashfamilie $\mathcal{H}$.
- n Schlüssel $S \subset \mathcal{K}$
- Zufallsvariable X : Anzahl Kollisionen der n Schlüssel aus S

$\Rightarrow$

$$\begin{aligned}\mathbb{E}(X) &= \mathbb{E}\left(\sum_{i \neq j} \mathbb{1}(h(k_i) = h(k_j)\right) = \sum_{i \neq j} \mathbb{E}(\mathbb{1}(h(k_i) = h(k_j)) \\ &\stackrel{*}{=} \binom{n}{2}\frac{1}{m} \leq \frac{n^2}{2m}\end{aligned}$$

* # Ungeordnete Paare
$\sum_{i \neq j} 1 = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} (n-1-i) = n(n-1) - n(n-1)/2 = n(n-1)/2$

Perfektes Hashing mit $\Theta(n^2)$ Speicherbedarf

Wenn $m = n^2 \Rightarrow \mathbb{E}(X) \leq \frac{1}{2}$.

Markov-Ungleichung[22] $\mathbb{P}(X \geq 1) \leq \frac{\mathbb{E}(X)}{1} \leq \frac{1}{2}$

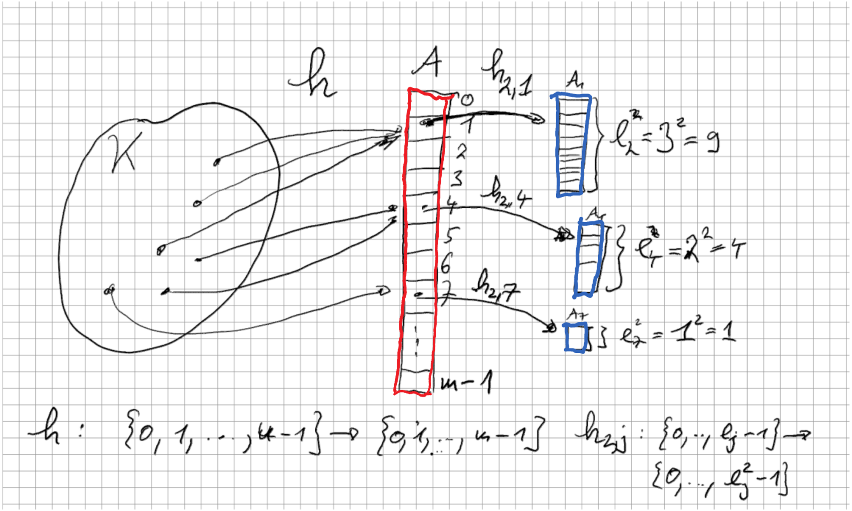
Also

$$\mathbb{P}(X < 1) = \mathbb{P}(\text{keine Kollision}) \geq \frac{1}{2}.$$

Folgerung: in erwartet $2 \cdot n$ Schritten kann man zu n Schlüsseln eine kollisionsfreie Hashtabelle der Grösse $m = n^2$ durch zufällige Wahl aus einer universellen Hashfamilie konstruieren.

[22] Appendix

Perfect Hashing Idee



Perfektes Hashing mit $\Theta(n)$ Speicherbedarf

2-Stufiges Verfahren

1. Wähle $m = n$ und $h : \{0, 1, \ldots, u-1\} \to \{0, 1, \ldots, m-1\}$ aus einer universellen Hashfamilie. Füge alle n Schlüssel in die Hashtabelle mit Verketten ein. Sei l_i die Länge der Kette am Index i.
   Wenn $\sum_{i=0}^{m-1} l_i^2 > 4n$, dann wiederhole diesen Schritt 1.
2. Für jeden Index $i = 1, \ldots, m-1$ mit $l_i > 0$ erzeuge so lange Hashtabellen für die enthaltenen l_i Schlüssel der Länge l_i^2 mit universellem Hashing (Hashfunktion $h_{2,i}$), bis keine Kollisionen auftreten.

Speicherbedarf $\Theta(n)$.

Erwartete Laufzeiten

- Für Schritt 1: Hashtabelle der Grösse $m = n$.
  Wir zeigen auf der nächsten Seite, dass $\mathbb{E}\left(\sum_{j=0}^{m-1} l_j^2\right) \leq 2n$. Dann folgt (Markov): $\mathbb{P}\left(\sum_{j=0}^{m-1} l_j^2 \geq 4n\right) \leq \frac{2n}{4n} = \frac{1}{2}$.
  $\Rightarrow$ Erwartete zwei Wiederholungen vom Schritt 1.
- Für Schritt 2: $\sum l_i^2 \leq 4n$. Für jedes i erwartet zwei Versuche mit Laufzeit l_i^2.
  Insgesamt $\mathcal{O}(n)$

$\Rightarrow$ Die perfekte Hashtabelle kann in erwartet $\mathcal{O}(n)$ Schritten erstellt werden.

Erwarteter Speicherverbrauch Hashtabellen 2.Stufe

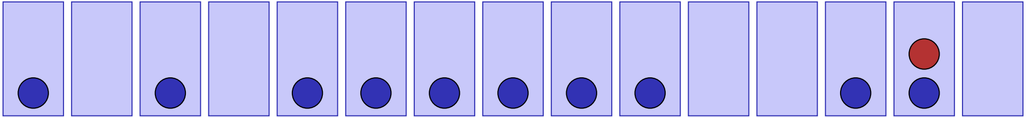
$$
\begin{aligned}
\mathbb{E}\left(\sum_{j=0}^{m-1} l_j^2\right) &= \mathbb{E}\left(\sum_{j=0}^{m-1}\sum_{i=0}^{n-1}\sum_{i'=0}^{n-1} \mathbb{1}(h(k_i) = h(k_{i'}) = j)\right) \\
&= \mathbb{E}\left(\sum_{i=0}^{n-1}\sum_{i'=0}^{n-1} \mathbb{1}(h(k_i) = h(k_{i'}))\right) \\
&= \mathbb{E}\left(\sum_{i=i'} \mathbb{1}(h(k_i) = h(k_{i'})) + 2 \cdot \sum_{i \neq i'} \mathbb{1}(h(k_i) = h(k_{i'}))\right) \\
&= n + 2 \cdot \sum_{i \neq i'} \mathbb{E}(\mathbb{1}(h(k_i) = h(k_{i'}))) \\
&= n + 2\binom{n}{2}\frac{1}{m} \overset{m=n}{=} 2n - 1 \leq 2n.
\end{aligned}
$$

14.9 Anhang

Mathematische Formeln

[Geburtstagsparadoxon]

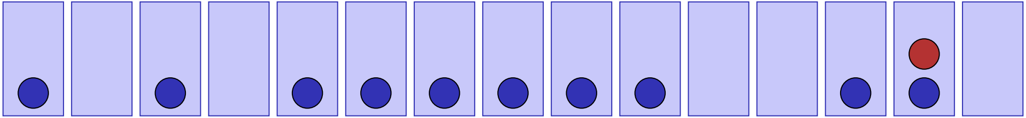
Annahme: m Urnen, n Kugeln (oBdA $n \leq m$).
n Kugeln werden gleichverteilt in Urnen gelegt.



Wie gross ist die Kollisionswahrscheinlichkeit?

[Geburtstagsparadoxon]

Annahme: m Urnen, n Kugeln (oBdA $n \le m$).
n Kugeln werden gleichverteilt in Urnen gelegt.



Wie gross ist die Kollisionswahrscheinlichkeit?

Geburtstagsparadoxon: Bei wie vielen Personen (n) ist die Wahrscheinlichkeit, dass zwei am selben Tag ($m = 365$) Geburtstag haben grösser als 50%?

[Geburtstagsparadoxon]

$\mathbb{P}(\text{keine Kollision}) = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-n+1}{m} = \frac{m!}{(m-n)! \cdot m^m}.$

[Geburtstagsparadoxon]

$\mathbb{P}$(keine Kollision) $= \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-n+1}{m} = \frac{m!}{(m-n)! \cdot m^m}$.

Sei $a \ll m$. Mit $e^x = 1 + x + \frac{x^2}{2!} + \ldots$ approximiere $1 - \frac{a}{m} \approx e^{-\frac{a}{m}}$. Damit:

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{m}\right) \approx e^{-\frac{1+\cdots+n-1}{m}} = e^{-\frac{n(n-1)}{2m}}.$$

[Geburtstagsparadoxon]

$\mathbb{P}(\text{keine Kollision}) = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \dots \cdot \frac{m-n+1}{m} = \frac{m!}{(m-n)! \cdot m^m}$.

Sei $a \ll m$. Mit $e^x = 1 + x + \frac{x^2}{2!} + \dots$ approximiere $1 - \frac{a}{m} \approx e^{-\frac{a}{m}}$. Damit:

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{m}\right) \approx e^{-\frac{1+\cdots+n-1}{m}} = e^{-\frac{n(n-1)}{2m}}.$$

Es ergibt sich

$$\mathbb{P}(\text{Kollision}) = 1 - e^{-\frac{n(n-1)}{2m}}.$$

[Geburtstagsparadoxon]

$\mathbb{P}(\text{keine Kollision}) = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-n+1}{m} = \frac{m!}{(m-n)! \cdot m^m}.$

Sei $a \ll m$. Mit $e^x = 1 + x + \frac{x^2}{2!} + \ldots$ approximiere $1 - \frac{a}{m} \approx e^{-\frac{a}{m}}$. Damit:

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{m}\right) \approx e^{-\frac{1+\cdots+n-1}{m}} = e^{-\frac{n(n-1)}{2m}}.$$

Es ergibt sich

$$\mathbb{P}(\text{Kollision}) = 1 - e^{-\frac{n(n-1)}{2m}}.$$

Auflösung zum Geburtstagsparadoxon: Bei 23 Leuten ist die Wahrscheinlichkeit für Geburstagskollision 50.7%. Zahl stammt von der leicht besseren Approximation via Stirling Formel. $n! \approx \sqrt{2\pi n} \cdot n^n \cdot e^{-n}$

[Erwartungswertformel]

$X \geq 0$ diskrete Zufallsvariable mit $\mathbb{E}(X) < \infty$

$$\begin{aligned}\mathbb{E}(X) &\overset{(def)}{=} \sum_{x=0}^{\infty} x\mathbb{P}(X = x) \\ &\overset{\text{Aufzählen}}{=} \sum_{x=1}^{\infty} \sum_{y=x}^{\infty} \mathbb{P}(X = y) \\ &= \sum_{x=0}^{\infty} \mathbb{P}(X > x)\end{aligned}$$

[Markov Ungleichung]

diskrete Version $X \geq 0, a > 0$:

$$\begin{aligned}
\mathbb{E}(X) &= \sum_{x=0}^{\infty} x\mathbb{P}(X = x) \\
&\geq \sum_{x=a}^{\infty} x\mathbb{P}(X = x) \\
&\geq a \sum_{x=a}^{\infty} \mathbb{P}(X = x) \\
&= a \cdot \mathbb{P}(X \geq a)
\end{aligned}$$

$\Rightarrow$

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}(X)}{a}$$