Algorithmen und Datenstrukturen 2

Lerneinheit 2: Hashing

Prof. Dr. Christoph Karg

Studiengang Informatik
Hochschule Aalen

Sommersemester 2016

Einleitung

Das Thema dieser Lerneinheit ist Hashing, einer Technik zur Speicherung von Datensätzen mit Zugriffsschlüsseln.

Die Lerneinheit gliedert sich in folgende Teile:

- Begriffsdefinitionen
- Hashing mit direkter Adressierung
- Hashing mit Chaining
- Hashing mit Open Addressing

Welche Daten werden verarbeitet?

Die zu verarbeitenden Elemente bestehen aus:

- Schlüssel/Key: Identifikation des Datensatzes
- Satellitendaten: Weitere zum Datensatz gehörende Informationen

Für ein Element x bezeichnet $key(x)$ den zugehörigen Schlüssel

Beispiel:

- Online Katalog $\rightsquigarrow$ Schlüssel: Bestellnummer
- Personaldatenbank $\rightsquigarrow$ Schlüssel: Personalnummer
- KFZ Datenbank $\rightsquigarrow$ Schlüssel: Autokennzeichen
- Cache im Webbrowser $\rightsquigarrow$ Schlüssel: URL

Bereitzustellende Operationen

Modifikationen:

- INSERT(T,x): Füge dem Datensatz x mit Schlüssel $key(x)$ in die Hashing Datenstruktur T ein
- DELETE(T, x): Entferne den Datensatz x mit Schlüssel $key(x)$ aus der Hashing Datenstruktur T

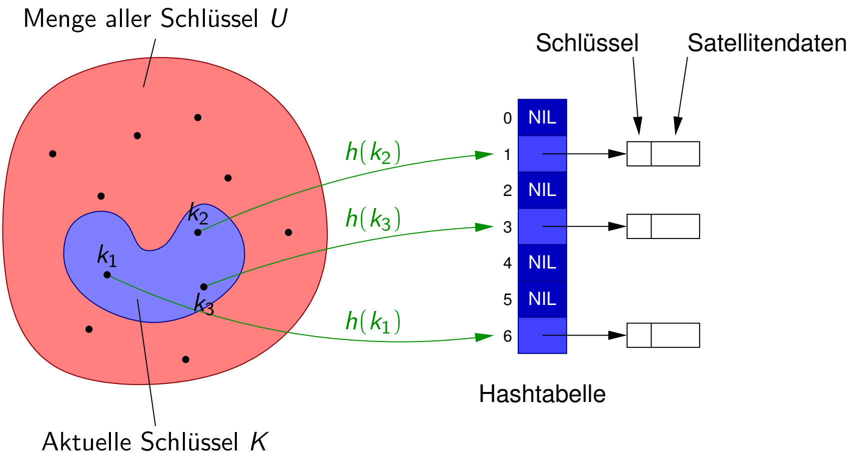
Anfrage:

- SEARCH(T, k): Suche in der Hashing Datenstruktur T nach einem Datensatz mit dem Schlüssel k

Hashtabellen

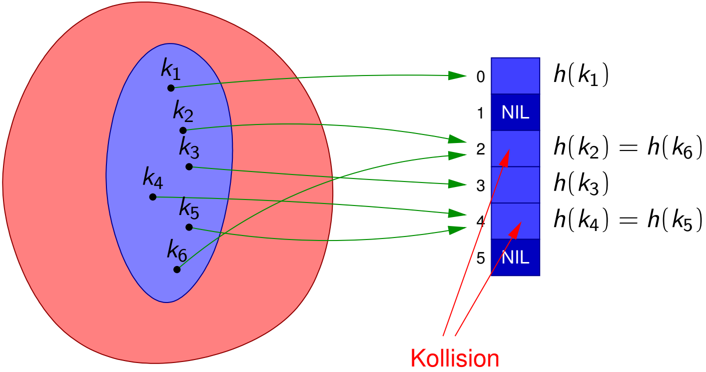
- Verallgemeinerung von Arrays
- Idee: Berechne Index im Array anhand des Schlüssels
- Ideal, falls die Anzahl der zu speichernden Schlüssel vergleichsweise klein zur Anzahl der möglichen Schlüssel
- Laufzeit
  - ▷ Schlechtes Worst Case Verhalten: $O(n)$
  - ▷ Sehr gutes Average Case Verhalten: $O(1)$
- Hashing Verfahren:
  - ▷ Direkte Adressierung
  - ▷ Hashing mit Chaining
  - ▷ Hashing mit Open Addressing

Idee hinter Hashing



Hashing

Idee: Speichere die Schlüssel in einer Tabelle, die deutlich kleiner als $\|U\|$ ist



Problem: Kollisionen

Definitionen zum Thema Hashing

- Das Universum U ist die Menge aller möglichen Schlüssel.
- Die Menge der aktuell gespeicherten Schlüssel ist K
- Eine Hashtabelle ist ein Array $T[0..m-1]$ der Dimension $m > 0$. Die Elemente des Arrays nennt man Slots
- Eine Hashfunktion ist eine Abbildung

$$h : U \mapsto \{0, 1, \ldots, m-1\}$$

- $h(key(x))$ ist der Hashwert des Datensatzes x
- Kurzschreibweise: $h(x)$ ist synonym mit $h(key(x))$
- Eine Kollision tritt auf, wenn zwei verschiedene Elemente x und y denselben Hashwert haben, d.h., wenn $h(x) = h(y)$ gilt

Direkte Adressierung

- Idee: Benutze die Schlüssel $\{0, 1, \dots, m-1\}$ direkt als Index in der Hashtabelle T
- Direkt mit einem Array realisierbar
- Voraussetzungen:
  - Universum U ist vergleichsweise klein
  - Alle Elemente haben paarweise verschiedene Schlüssel
- Hashfunktion $h : U \mapsto U$ mit $h(k) = k$
- Vorteil: Die Laufzeit der Operationen Insert, Delete und Search haben eine Worst Case Laufzeit von $O(1)$.
- Nachteil: Die Größe der Hashtabelle ist linear in $\|U\|$.

Direkte Adressierung (Forts.)

$\textsc{DirectAddressSearch}(T, k)$

return $T[k]$

$\textsc{DirectAddressInsert}(T, x)$

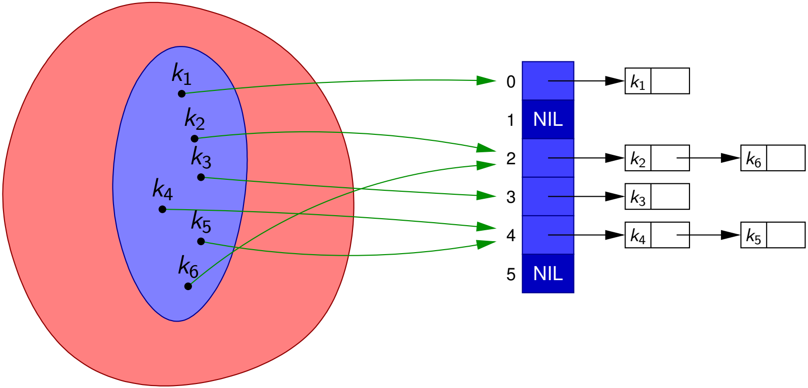
$T[key(x)] := x$

$\textsc{DirectAddressDelete}(T, x)$

$T[key(x)] := \texttt{NIL}$

Kollisionsbehandlung mittels Verkettung

Idee: Elemente mit demselben Schlüssel werden in einer verketteten Liste gespeichert ⇝ Hashing mit Verkettung (Chaining)



Algorithmen für Verkettung

$\textsc{ChainedHashSearch}(T, x)$

1 *Suche nach einem Element mit Schlüssel* $k = key(x)$ *in Liste* $T[h(k)]$

$\textsc{ChainedHashInsert}(T, x)$

1 *Füge* x *mit Schlüssel* $k = key(x)$ *am Anfang der Liste* $T[h(k)]$ *ein*

$\textsc{ChainedHashDelete}(T, x)$

1 *Lösche* x *aus der Liste* $T[h(k)]$

Beispiel für Verkettung

Hashfunktion: $h(x) = x \bmod 7$

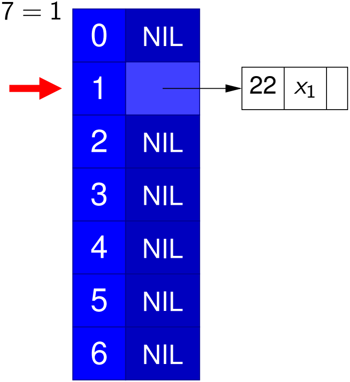
Hashtabelle mit 7 Slots:

0	NIL
1	NIL
2	NIL
3	NIL
4	NIL
5	NIL
6	NIL

Beispiel für Verkettung (Forts.)

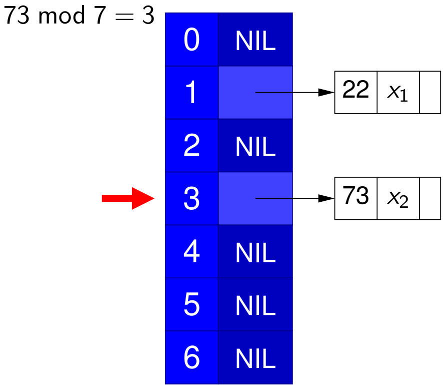
Einfügen des Elements x_1 mit $key(x_1) = 22$

$22 \bmod 7 = 1$



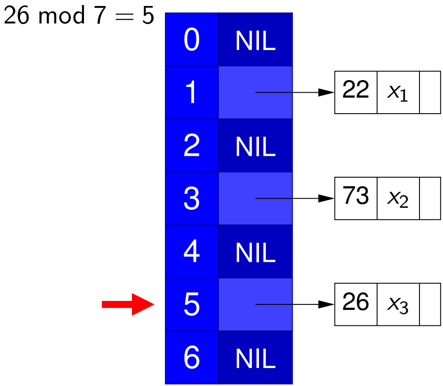
Beispiel für Verkettung (Forts.)

Einfügen des Elements x_2 mit $key(x_2) = 73$



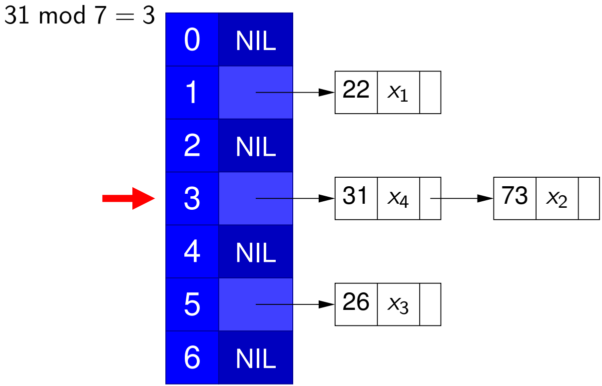
Beispiel für Verkettung (Forts.)

Einfügen des Elements x_3 mit $key(x_3) = 26$



Beispiel für Verkettung (Forts.)

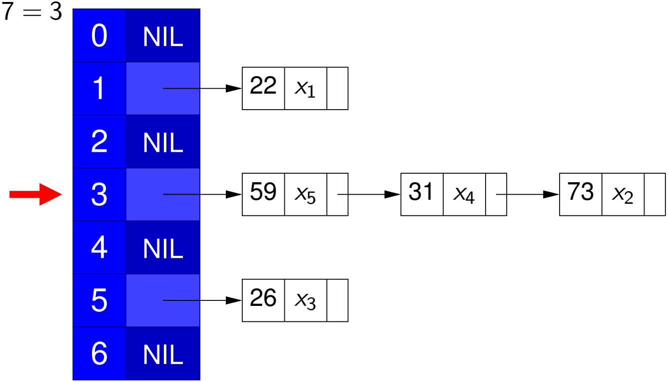
Einfügen des Elements x_4 mit $key(x_4) = 31$



Beispiel für Verkettung (Forts.)

Einfügen des Elements x_5 mit $key(x_5) = 59$

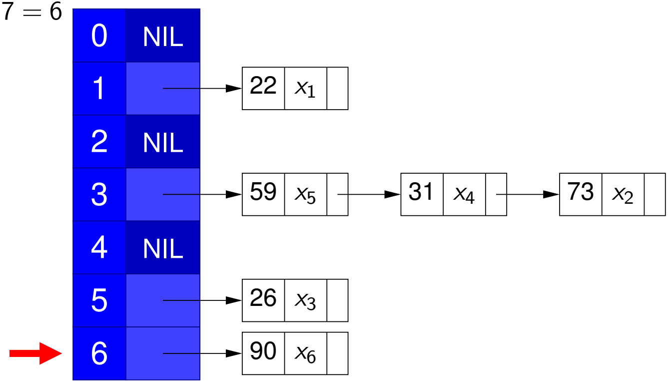
$59 \bmod 7 = 3$



Beispiel für Verkettung (Forts.)

Einfügen des Elements x_6 mit $key(x_6) = 90$

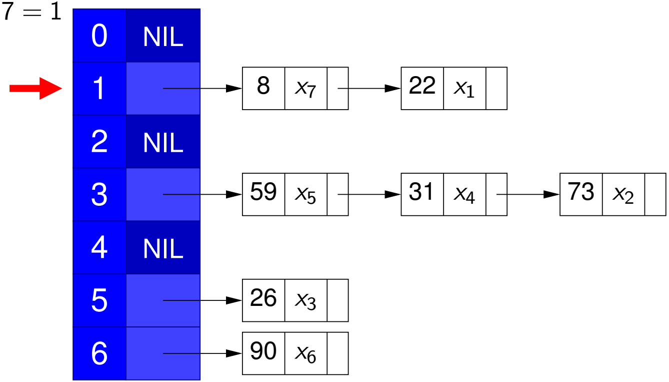
$90 \bmod 7 = 6$



Beispiel für Verkettung (Forts.)

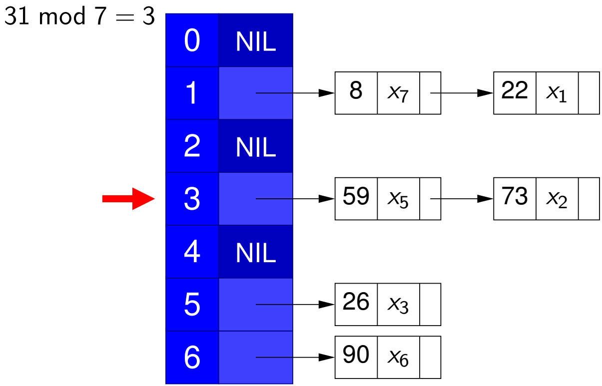
Einfügen des Elements x_7 mit $key(x_7) = 8$

$8 \bmod 7 = 1$



Beispiel für Verkettung (Forts.)

Löschen des Elements x_4 mit $key(x_4) = 31$



Worst Case Analyse

Annahmen:

- Die Tabelle T enthält n Elemente.
- Die Berechnung von h und der Tabellenzugriff erfolgen in Zeit $O(1)$.

Schlimmster Fall: Alle Elemente wurden in einen Slot gehasht $\rightsquigarrow$ verkettete Liste mit n Elementen

- $\textsc{Search}(T, x)$: Das gesuchte Element x ist am Ende der Liste. Laufzeit: $O(n)$.
- $\textsc{Insert}(\mathrm{T}, x)$: Die Laufzeit ist $O(1)$, da x am Anfang der Liste $T[h(key(x))]$ eingefügt wird.
- $\textsc{Delete}(T, x)$: Das Element wird zuerst in der Tabelle gesucht und dann aus der Liste gelöscht. Laufzeit: $O(n)$.

Belegungsfaktor einer Hashtabelle

Gegeben ist eine Hashtabelle T mit m Slots.

Der Belegungsfaktor α von T ist die mittlere Anzahl von Elementen pro Slot.

Fakt: Falls T n Elemente enthält, dann ist

$$\alpha = \frac{n}{m}$$

Simple Uniform Hashing

Unter Simple Uniform Hashing versteht man die folgende Annahme:

> *Ein Element x wird mit Wahrscheinlichkeit $\frac{1}{m}$ in den Slot j gehasht, unabhängig von der Platzierung der anderen Elemente.*

Mit n_j wird die Anzahl der Elemente in Slot j bezeichnet, wobei $j = 0, 1, \ldots, m-1$. Es gilt:

$$n = n_0 + n_1 + \ldots + n_{m-1}.$$

Erwartungswert von n_j:

$$E[n_j] = \frac{n}{m} = \alpha$$

Analyse einer nicht erfolgreichen Suche

Satz. Bei Hashing mit Verkettung ist die durchschnittliche Laufzeit einer nicht erfolgreichen Suche gleich $\Theta(1+\alpha)$, falls die Simple Uniform Hashing Annahme gilt.

Beweis. Betrachte eine Hashtabelle T mit n Einträgen und ein Element x mit einem Schlüssel $k = key(x)$, der nicht in T enthalten ist.

- Es ist $h(k) = j$ mit Wahrscheinlichkeit $\frac{1}{m}$
- Die Liste in $T[j]$ wird während der Suche komplett durchlaufen, da k nicht in ihr enthalten ist
- Die Länge der Liste $T[j]$ ist im Mittel $n_j = n/m = \alpha$
- Die Laufzeit setzt sich zusammen aus der Berechnung von h und der Zeit zum Durchlaufen der Liste

Somit: Laufzeit $\Theta(1+\alpha)$

Wahrscheinlichkeit einer Kollision

Satz. Falls die Simple Uniform Hashing Annahme gilt, dann ist die Wahrscheinlichkeit einer Kollision zweier Elemente in der Hashtabelle gleich $\frac{1}{m}$.

Beweis: Betrachte zwei Elemente x und y. Es gilt:

$$\begin{aligned}
&\text{Prob}[h(key(x)) = h(key(y))] \\
&= \sum_{j=0}^{m-1} \text{Prob}[h(key(x)) = j \text{ und } h(key(y)) = j] \\
&= \sum_{j=0}^{m-1} \frac{1}{m} \cdot \frac{1}{m} \\
&= \frac{m}{m^2} = \frac{1}{m}
\end{aligned}$$

Analyse einer erfolgreichen Suche

Satz. Bei Hashing mit Verkettung ist die durchschnittliche Laufzeit einer erfolgreichen Suche gleich $\Theta(1 + \alpha/2)$, falls die Simple Uniform Hashing Annahme gilt.

Beweis. Annahme: das zu suchende Element x wird unter Gleichverteilung ausgewählt.

- Um x zu finden, müssen alle Elemente untersucht werden, die in der Liste $T[h(key(x))]$ vor x sind sowie das Element x selbst.
- Die durchschnittliche Laufzeit ist linear in der Anzahl der Elemente, die nach x in die Liste $T[h(key(x))]$ eingefügt wurden.

Ziel: Berechnung der zu erwartenden Anzahl dieser Elemente

Analyse einer erfolgreichen Suche (Forts.)

Sei x_i das i-te Element, das in die Hashtabelle eingefügt wird, wobei $i = 1, 2, \ldots, n$. Sei $k_i = key(x_i)$.

Für k_i und k_j ist die Zufallsvariable X_{ij} wie folgt definiert:

$$X_{ij} = \begin{cases} 1, & \text{falls } h(k_i) = h(k_j), \\ 0, & \text{sonst.} \end{cases}$$

Es gilt:

$$\begin{aligned} E[X_{ij}] &= 1 \cdot \text{Prob}[X_{ij} = 1] + 0 \cdot \text{Prob}[X_{ij} = 0] \\ &= \text{Prob}[X_{ij} = 1] \\ &= \frac{1}{m} \end{aligned}$$

Analyse einer erfolgreichen Suche (Forts.)

Die Elemente, die nach x_i in die Hashtabelle eingefügt wurden, sind $x_{i+1}, \ldots, x_n$.

Die zu erwartende Anzahl der zu untersuchenden Elemente bei einer Suche nach x_i ist deshalb:

$$
\begin{aligned}
E\left[1 + \sum_{j=i+1}^{n} X_{ij}\right] &= 1 + \sum_{j=i+1}^{n} E[X_{ij}] \\
&= 1 + \sum_{j=i+1}^{n} \frac{1}{m}
\end{aligned}
$$

Analyse einer erfolgreichen Suche (Forts.)

Aufwand für ein zufällig unter Gleichverteilung gezogenes Element x_i:

$$
\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right) &= \frac{1}{n}\left(\sum_{i=1}^{n}1+\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{m}\right)\\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n}1\\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)\\
&= 1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right)
\end{aligned}
$$

Analyse einer erfolgreichen Suche (Forts.)

$$
\begin{aligned}
1 + \frac{1}{nm}\left(\sum_{i=1}^{n} n - \sum_{i=1}^{n} i\right) &= 1 + \frac{1}{nm}\left(n^2 - \frac{n(n+1)}{2}\right) \\
&= 1 + \frac{1}{nm} \cdot \frac{n^2 - n}{2} \\
&= 1 + \frac{n-1}{2m} \\
&= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}
\end{aligned}
$$

Somit: Average Case Laufzeit einer erfolgreichen Suche:

$$\Theta(1 + \alpha/2 - \alpha/2n) = \Theta(1 + \alpha/2)$$

Auswahl der Hashfunktion

- Wünschenswert: Hashfunktion erfüllt die Simple Uniform Assumption
- Problem: Verteilung der Eingaben ist nicht bekannt
- Lösung: Einsatz von Heuristiken $\rightsquigarrow$ Wähle eine Hashfunktion, die sich in der Praxis bewährt hat
- Konstruktion von Hashfunktionen mittels
  - ▷ Divisionsmethode
  - ▷ Multiplikationsmethode

Divisionsmethode

- Universum $U = \mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$
- Größe der Hashtabelle $m > 0$
- Hashfunktion $h(k) = k \bmod m$
- Modul m muss "geschickt" gewählt werden:
  - ▷ Schlecht: Zweierpotenz $m = 2^p$, $p \in \mathbb{N}$
  - ▷ Gut: Primzahl, die "nicht zu nahe an" einer Zweierpotenz liegt

Multiplikationsmethode

Verfahren: Berechnung des Hashwerts $h(k)$ in zwei Schritten:

1. Multipliziere k mit einer Konstante A, wobei $0 < A < 1$
2. Multipliziere die Nachkommastellen von $k \cdot A$ mit m, d.h., berechne $\lfloor (k \cdot A - \lfloor k \cdot A \rfloor) m \rfloor$

Bemerkungen:

- die Wahl der Tabellengröße m ist unkritisch
- Die Wahl von A ist ebenfalls unkritisch. Es gibt jedoch Werte die besser funktionieren als andere

Empfehlung von Knuth: Goldener Schnitt

$$A = (\sqrt{5} - 1)/2 = 0.6180339887\ldots$$

Beispiel Multiplikationsmethode

Wähle $m = 1024$ und $A = \frac{\sqrt{5}-1}{2}$.

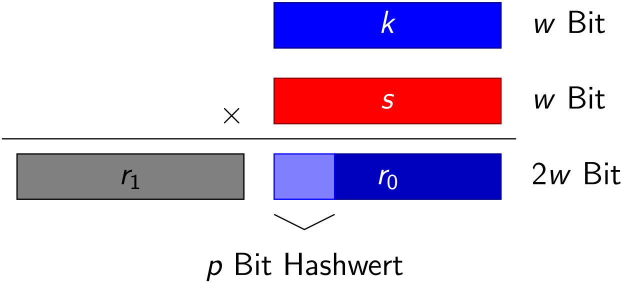
Der Schlüssel $k = 15341$ wird gehasht auf:

$$\begin{aligned} h(15341) &= \left\lfloor \left(15341 \cdot \frac{\sqrt{5}-1}{2} - \left\lfloor 15341 \cdot \frac{\sqrt{5}-1}{2} \right\rfloor \right) \cdot 1024 \right\rfloor \\ &\approx \lfloor (9481.259421 - 9481) \cdot 1024 \rfloor \\ &= 265 \end{aligned}$$

Berechnung mit Ganzzahlarithmetik

Annahme: Tabellengröße $m = 2^p$, Wortlänge w, d.h., Schlüssel k ist eine w-Bit Zahl

Berechnung von $h(k)$: Wähle ganze Zahl $0 < s < 2^w$ und definiere $A = \frac{s}{2^w}$



Berechnung mit Ganzzahlarithmetik (Forts.)

Führe zur Berechnung von $h(k)$ folgende Schritte aus:

- Berechne $r = k \cdot s$ (dies ist eine $2w$-Bit Zahl)
- Zerlege r in zwei w-Bit Wörter r_0, r_1 derart, dass $r = r_1 \cdot 2^w + r_0$
- Wähle die p höchstwertigsten Bits von r_0 als Hashwert $h(k)$

Bemerkung: Aufgrund von Rundungsfehlern weicht das Ergebnis in der Regel von der exakten Berechnung ab

Zur Illustration ein Beispiel

Gegeben: Wortlänge 16 Bit, $m = 2^{10} = 1024$

Wähle s so, dass $|s/2^{16} - (\sqrt{5} - 1)/2|$ minimal:

$$s = \lceil 2^{16} \cdot (\sqrt{5} - 1)/2 \rceil = 40503$$

Somit ist $A = \frac{40503}{65536} = 0.6180267333\ldots$

Berechnung des Hashwerts von $k = 15341$:

$$r = 40503 \cdot 15341 = 621356523 = 9481 \cdot 2^{16} + 9707$$

Binärdarstellung von $r_0 = 9707$: $\underbrace{0010010111}_{=h(k)}\ 101011$

Ergebnis: $h(15341) = 151$

Konvertierung von Zeichenketten

Gegeben: Alphabet $\Sigma = \{a_0, a_1, \ldots, a_{b-1}\}$

Aufgabe: Konvertiere ein Wort $x = x_1 \ldots x_n \in \Sigma^n$ in eine natürliche Zahl

Idee: Interpretiere x als Zahl $1x$ zur Basis $b = \|\Sigma\|$ und konvertiere diese ins Dezimalsystem

Formel:

$$d = b^n + \sum_{i=1}^{n} val(x_i) \cdot b^{n-i}$$

wobei $val : \Sigma \mapsto \{0, 1, \ldots, b-1\}$ eine bijektive Abbildung ist

Beispiel zur Konvertierung von Zeichenketten

- $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$
- Bijektive Abbildung:

x	a	b	c	d
$val(x)$	0	1	2	3

- Konvertierung von $x = \texttt{cbad}$:

$$
\begin{aligned}
4^4 + \sum_{i=1}^{4} b^{4-i} val(x_i) = & \\
= \quad & 4^4 + 4^3 \cdot val(\texttt{c}) + 4^2 \cdot val(\texttt{b}) + 4^1 \cdot val(\texttt{a}) + 4^0 \cdot val(\texttt{d}) \\
= \quad & 256 + 64 \cdot 2 + 16 \cdot 1 + 4 \cdot 0 + 1 \cdot 3 \\
= \quad & 403
\end{aligned}
$$

Open Addressing

- Idee: Alle Elemente werden direkt in der Hashtabelle gespeichert
- Vorteil: es ist kein zusätzlicher Speicherplatz für verkettete Listen notwendig.
- Nachteil: Kapazität der Tabelle ist begrenzt. D.h., der Belegungsfaktor α immer ≤ 1
- Die Suche nach einem freien Slot erfolgt durch sogenanntes Probing. Es werden die Slots der Reihe nach sondiert, bis ein freier Slot gefunden wird
- Die Sondierungsreihenfolge hängt vom einzufügenden Schlüssel k ab

Berechnen der Sondierungssequenz

- Hashfunktion bei Tabellengröße m:

$$h : U \times \{0, 1, \ldots, m-1\} \mapsto \{0, 1, \ldots, m-1\}$$

- Sondierungssequenz (probe sequence) für Schlüssel k:

$$\langle h(k, 0), h(k, 1), \ldots, h(k, m-1) \rangle$$

- Bedingung: Sondierungssequenz muss eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$ sein.

Einfügen eines Elements

HashInsert(T, k)

```
i := 0
repeat
    j := h(k, i)
    if T[j] = NIL then
        T[j] := k
        return j
    else
        i := i + 1
until i = m
print "Hash table overflow"
```

Suche nach einem Element

$\textsc{HashSearch}(T, k)$

$i := 0$
repeat
  $j := h(k, i)$
  if $T[j] = k$ **then**
    return j
  $i := i + 1$
until $T[j] = \texttt{NIL}$ **or** $i = m$
return $\texttt{NIL}$

Löschen eines Elements

Vorsicht: Löschen von Slot j durch die Zuweisung $T[j] = \texttt{NIL}$ verhindert das Finden von Elementen, bei denen Slot j als belegt sondiert wurde.

Lösung:

- Markiere einen gelöschten Slot mit DELETE anstatt mit $\texttt{NIL}$.
- Passe die Prozedur HashInsert so an, dass Elemente auch in mit DELETE markierte Felder eingefügt werden.

Bemerkung: Bei Einsatz obiger Variante hängt die Laufzeit von HashSearch nicht mehr von Belegungsfaktor α ab.

Linear Probing

Ziel: Berechnung einer geeigneten Sondierungssequenz

Idee: Setze eine zusätzliche Hashfunktion $h' : U \mapsto \{0, 1, \ldots, m-1\}$ ein

Linear Probing: Für $i = 0, 1, \ldots, m-1$ ist

$$h(k, i) = (h'(k) + i) \bmod m$$

Bemerkungen:

- Linear Probing ist einfach zu implementieren.
- Das Primary Clustering Problem erhöht die mittlere Suchzeit.

Quadratic Probing

Einsatz einer Hashfunktion der Bauart

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

wobei c_1 und c_2 positive Konstanten sind

Bemerkungen:

- Die Sondierungswerte haben einen quadratischen Offset in Abhängigkeit von i
- Quadratic Probing ist besser als Linear Probing
- Nicht jede Kombination von m, c_1 und c_2 erfüllt die Permutationsbedingung
- Das Secondary Clustering Problem erhöht die mittlere Suchzeit

Bestimmen der Parameter c_1 und c_2

Ausgangspunkt:

$\mathrm{A}(T,\ k)$

Input: Hashtabelle T der Größe m, Hashfunktion h, Schlüssel k

Output: Freier Slot j, falls es einen solchen gibt

```
j := h(k) mod m; i := 0
while T[j] ≠ NIL and i < m do
    i := i + 1
    j := (i + j) mod m
if (i < m) then
    return j
else
    return NIL
```

Bestimmen der Parameter c_1 und c_2 (Forts.)

Satz. Angenommen, $m = 2^n$. Dann ist die von dem Algorithmus A durchlaufene Sondierungssequenz identisch zu Quadratic Probing mit den Parametern $c_1 = \frac{1}{2}$ und $c_2 = \frac{1}{2}$.

Zu beweisen:

1. Der Algorithmus ist identisch zu Quadratic Probing mit obigen Parametern
2. Mit den obigen Parametern erzeugt Quadratic Probing für beliebiges k eine korrekte Sondierungssequenz

Beweis von Punkt 1

Schleifeninvariante: Angenommen, $m = 2^n$. Dann gilt nach dem j-ten Durchlauf der while-Schleife des Algorithmus A, dass

$$j = h(k) + \frac{1}{2} \cdot i + \frac{1}{2} \cdot i^2 \bmod m$$

Beweis. Induktion über die Anzahl der Schleifendurchläufe

Initialisierung: Vor der ersten Ausführung von Zeile 3 des Algorithmus A ist $j = h(k)$ und $i = 0$. Somit:

$$\begin{aligned} j &= h(k) \\ &= h(k) + \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0^2 \bmod m \\ & h(k) + \frac{1}{2} \cdot i + \frac{1}{2} \cdot i^2 \bmod m \end{aligned}$$

Beweis von Punkt 1 (Forts.)

Aufrechterhaltung: Angenommen, die Schleife wurde j-mal durchlaufen und es gilt:

$$j = \left(h(k) + \frac{1}{2} \cdot i + \frac{1}{2} \cdot i^2 \right) \bmod m$$

Im nächsten Durchlauf sind die neuen Werte der Variablen i und j gleich

$$\begin{aligned} i^* &= i + 1 \\ j^* &= i^* + j \bmod m \\ &= i + 1 + j \bmod m \end{aligned}$$

Beweis von Punkt 1 (Forts.)

Hieraus folgt:

$$\begin{aligned}
j^* &\equiv j + i + 1 \pmod{m} \\
&\equiv h(k) + \frac{1}{2} \cdot i + \frac{1}{2} \cdot i^2 + i + 1 \pmod{m} \\
&\equiv h(k) + \frac{1}{2} \cdot i + \frac{1}{2} + \frac{1}{2} \cdot i^2 + 2 \cdot \frac{1}{2} \cdot i + \frac{1}{2} \pmod{m} \\
&\equiv h(k) + \frac{1}{2} \cdot (i+1) + \frac{1}{2} \cdot (i^2 + 2i + 1) \pmod{m} \\
&\equiv h(k) + \frac{1}{2} \cdot (i+1) + \frac{1}{2} \cdot (i+1)^2 \pmod{m}
\end{aligned}$$

Also gilt am Ende des Durchlaufs:

$$j^* = h(k) + \frac{1}{2} \cdot (i+1) + \frac{1}{2} \cdot (i+1)^2 \bmod m$$

Beweis von Punkt 2

Wir definieren:

$$\begin{aligned} h(k,i) &= h(k) + \frac{1}{2} \cdot i + \frac{1}{2} \cdot i^2 \bmod m \\ &= h(k) + \frac{i \cdot (i+1)}{2} \bmod m \\ &= h(k) + S_i \bmod m \end{aligned}$$

wobei $S_i = \sum_{\ell=1}^{i} \ell$.

Behauptung. Angenommen. $m = 2^n$. Dann ist die Folge

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

für alle k eine Sondierungsequenz.

Beweis von Punkt 2 (Forts.)

Annahme: Es gibt Werte k, i, j mit $h(k, i) = h(k, j)$, wobei $0 \leq i < j < 2^n - 1$.

Also ist:

$$S_i \equiv S_j \pmod{2^n}$$

Somit:

$$\begin{aligned} S_i &= a \cdot 2^n + r \\ S_j &= b \cdot 2^n + r \end{aligned}$$

wobei $a < b$

Beweis von Punkt 2 (Forts.)

Hieraus folgt:

$$S_j - S_i = \sum_{\ell=i+1}^{j} \ell = \underbrace{(b-a)}_{=c} \cdot 2^n$$

Wegen

$$\sum_{\ell=i+1}^{j} \ell = \frac{(j-i)\cdot(j+i+1)}{2}$$

folgt: $(j-i)\cdot(j+i+1) = c \cdot 2^{n+1}$

Also: $2^{n+1} \mid (j-i)\cdot(j+i+1)$

Beweis von Punkt 2 (Forts.)

Bekannt: Für alle natürlichen Zahlen a, b, d gilt: Falls $d \mid ab$ und $\gcd(a, d) = 1$, dann $d \mid b$.

Fallunterscheidung:

- i gerade, j gerade: Also ist $j + i + 1$ ungerade und deshalb $\gcd(j + i + 1, 2^{n+1}) = 1$. Also muss $2^{n+1} \mid (j - i)$ gelten. Widerspruch, da $j - i < m < 2^{n+1}$.
- i ungerade, j ungerade: analog zum vorigen Punkt

Beweis von Punkt 2 (Forts.)

- i ungerade, j gerade: Also ist $j - i$ ungerade und deshalb $\gcd(j - i, 2^{n+1}) = 1$. Also muss $2^{n+1} \mid (j + i + 1)$ gelten. Widerspruch, da $j + i + 1 \leq 2m - 1 < 2^{n+1}$.
- i gerade, j ungerade: analog zum vorigen Punkt

Konsequenz: Jede Wahl von i und j führen zu einem Widerspruch. Folglich ist obige Annahme falsch (und die Behauptung korrekt).

Bemerkung: Bei Quadratic Probing verwendet man den Algorithmus A, um die Hashtabelle nach einem freien Slot zu durchsuchen

Double Hashing

Kombination zweier Hashfunktion h_1 und h_2:

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

Bedingung: $h_2(k)$ und m müssen für alle k teilerfremd sein.

Bemerkungen:

- Double Hashing ist eine der besten Methoden für Open Addressing.
- Die erzeugten Sondierungssequenzen ähneln Zufallspermutationen.

Korrektheit von Double Hashing

Satz. Falls $\gcd(h_2(k), m) = 1$, dann ist

$$\langle h(k, 0), \ldots, h(k, m-1) \rangle$$

eine Sondierungssequenz.

Beweis. Angenommen, $\gcd(h_2(k), m) = 1$ und es gibt zwei Zahlen $0 \leq i < j \leq m-1$, so dass $h(k, i) = h(k, j)$.

Hieraus folgt: $i \cdot h_2(k) \equiv j \cdot h_2(k) \pmod{m}$

Da $\gcd(h_2(k), m) = 1$, besitzt $h_2(k)$ ein multiplikatives Inverses modulo m.

Somit ist $i \equiv j \pmod{m}$. Widerspruch!

Funktionen für Double Hashing

Methode 1: Wähle $m = 2^k$, wobei $k \in \mathbb{N}$ und erstelle h_2 derart, dass die Funktion immer eine ungerade Zahl liefert

Beispiel: $m = 2048$ $k = 123456$:

$$\begin{aligned} h_1(k) &= k \bmod 2048 \\ h_2(k) &= (2k+1) \bmod 2048 \end{aligned}$$

Somit $h(k,0) = 576$, $h(k,i) = (576 + i \cdot 1153) \bmod 2048$ für $i = 1, 2, \ldots, m-1$.

Funktionen für Double Hashing (Forts.)

Methode 2: Wähle eine Primzahl m und

$$\begin{aligned} h_1(k) &= k \bmod m \\ h_2(k) &= 1 + (k \bmod m') \end{aligned}$$

wobei m' "etwas kleiner" als m ist, z.B. $m' = m - 1$.

Beispiel: $m = 701$, $m' = 700$, $k = 123456$:

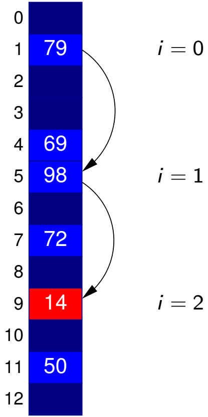
$$\begin{aligned} h_1(k) &= 80 \\ h_2(k) &= 257 \end{aligned}$$

Somit $h(k, 0) = 80$, $h(k, i) = (80 + i \cdot 257) \bmod 701$ für $i = 1, 2, \ldots, m - 1$.

Beispiel Double Hashing

$m = 13,\ h_1(k) = k \bmod 13,\ h_2(k) = 1 + (k \bmod 11)$
$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod 13$

Einfügen des Schlüssels 14:



Uniform Hashing Annahme

Uniform Hashing Annahme: Jeder Schlüssel k wählt eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$ mit Wahrscheinlichkeit $\frac{1}{m!}$ als Sondierungssequenz aus.

Bemerkung: Jedem Schlüssel wird eine Sondierungsequenz zufällig unter Gleichverteilung zugewiesen

Open Addressing: nicht erfolgreiche Suche

Satz. Unter der Uniform Hashing Annahme gilt für jede Hashtabelle mit Belegungsfaktor $\alpha = \frac{n}{m} < 1$: die durchschnittliche Anzahl der Sondierungsversuche bei einer nicht erfolgreichen Suche ist höchstens $\frac{1}{1-\alpha}$.

Beweis. Ablauf einer nicht erfolgreichen Suche nach dem Schlüssel k:

- Alle bis auf den letzten untersuchten Slot enthalten einen Schlüssel, der ungleich k ist
- Der letzte untersuchte Slot ist leer

Open Addressing: nicht erfolgreiche Suche (Forts.)

Ereignis A_i: Es gibt einen i-ten Sondierungsversuch und der Zugriff erfolgt auf einen belegten Slot, wobei $i = 1, \ldots, n$

Zufallsvariable X: Anzahl der sondierten Slots bei einer nicht erfolgreichen Suche

Für $i \geq 2$ gilt:

$$Prob\left[X \geq i\right] = Prob\left[A_1 \cap \ldots \cap A_{i-1}\right]$$

Rechenregel:

$$\begin{aligned} & Prob\left[A_1 \cap \ldots \cap A_{i-1}\right] \\ = \;& Prob\left[A_1\right] \cdot Prob\left[A_2 \mid A_1\right] \cdot Prob\left[A_3 \mid A_1 \cap A_2\right] \\ & \cdots Prob\left[A_{i-1} \mid A_1 \cap \ldots \cap A_{i-2}\right] \end{aligned}$$

Open Addressing: nicht erfolgreiche Suche (Forts.)

Uniform Hashing Annahme:

$$\begin{aligned} Prob\left[A_1\right] &= \frac{n}{m} \\ Prob\left[A_j \mid A_1 \cap \ldots \cap A_{j-1}\right] &= \frac{n-(j-1)}{m-(j-1)} \end{aligned}$$

Hierauf folgt:

$$\begin{aligned} Prob\left[X \geq i\right] &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\ &\leq \left(\frac{n}{m}\right)^{i-1} \\ &= \alpha^{i-1} \end{aligned}$$

Open Addressing: nicht erfolgreiche Suche (Forts.)

Erwartungswert von X:

$$
\begin{aligned}
Exp[X] &= \sum_{i=1}^{\infty} Prob[X \geq i] \\
&\leq \sum_{i=1}^{\infty} \alpha^{i-1} \\
&= \sum_{i=0}^{\infty} \alpha^{i} \\
&= \frac{1}{1-\alpha}
\end{aligned}
$$

Open Addressing: Einfügen eines Elements

Konsequenz: Unter Uniform Hashing gilt: Um ein Element in eine Hashtabelle mit Belegungsfaktor α einzufügen, benötigt man im Durchschnitt höchstens $\frac{1}{1-\alpha}$ Sondierungsschritte

Zahlenbeispiele:

α	$1/(1-\alpha)$
10%	1.11
20%	1.25
50%	2.00
75%	4.00
90%	10.00
95%	20.00

Open Addressing: erfolgreiche Suche

Satz. Gegeben ist eine Open Addressing Hashtabelle mit einem Belegungsfaktor $\alpha < 1$.

Angenommen, es gilt Uniform Hashing und der zu suchende Schlüssel wird zufällig unter Gleichverteilung gezogen.

Dann ist die durchschnittliche Anzahl der Sondierungsversuche bei einer erfolgreichen Suche höchstens $\frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$.

Open Addressing: erfolgreiche Suche (Forts.)

Beweis. Da die Suche nach k erfolgreich verläuft, muss k in der Tabelle enthalten sein.

Beobachtung: Bei der Suche nach k wird dieselbe Sondierungssequenz durchlaufen wie beim Einfügen des Schlüssels.

Annahme: k war der $(i+1)$-te Schlüssel, der in die Tabelle eingefügt wurde.
Konsequenz: Mittlere Anzahl Sondierungsversuche, um k zu finden, ist gleich

$$\frac{1}{1-\left(\frac{i}{m}\right)} = \frac{m}{m-i}$$

Open Addressing: erfolgreiche Suche (Forts.)

Erwartungswert über alle n Schlüssel in der Tabelle:

$$\begin{aligned}\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} \\ &= \frac{1}{\alpha}(H_m - H_{m-n}) \\ &= \frac{1}{\alpha}\sum_{i=m-n+1}^{m}\frac{1}{i}\end{aligned}$$

Hierbei steht H_m für die harmonische Reihe, d.h.

$$H_m = \sum_{i=1}^{m}\frac{1}{i}.$$

Open Addressing: erfolgreiche Suche (Forts.)

Fakt: Für jede monoton abnehmende Funktion $f(x)$ gilt:

$$\sum_{i=m}^{n} f(i) \leq \int_{m-1}^{n} f(x)\ dx$$

Hieraus folgt:

$$\begin{aligned} \frac{1}{\alpha} \sum_{i=m-n+1}^{m} \frac{1}{i} &\leq \frac{1}{\alpha} \int_{m-n}^{m} \frac{1}{x} dx \\ &= \frac{1}{\alpha} \ln x \Big|_{m-n}^{m} \end{aligned}$$

Open Addressing: erfolgreiche Suche (Forts.)

$$
\begin{aligned}
&= \frac{1}{\alpha}(\ln m - \ln(m-n)) \\
&= \frac{1}{\alpha}\ln\left(\frac{m}{m-n}\right) \\
&= \frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)
\end{aligned}
$$

Zusammenfassung

- Hashing ist eine Verallgemeinerung von Arrays
- In einer Hashtabelle werden Daten mit Zugriffsschlüsseln gespeichert
- Die Zugriffszeit auf die Hashtabelle ist im Durchschnitt konstant
- Kollisionen werden aufgelöst durch
  - Chaining
  - Open Addressing