# Kapitel 12

# Hashing: Einführung (Teil 1)

Effiziente Algorithmen, SS 2018

Professor Dr. Petra Mutzel

VO 21/22 am 3./5. Juli 2018

# Übersicht

I. Effiziente Graphalgorithmen

II. Approximationsalgorithmen

III. Wörterbuch- und Wörtersuchprobleme

11. Skiplisten
12. Hashing und Perfektes Hashing
13. String Matching

# Wörterbuch- und Wörtersuchprobleme

Es gibt verschiedene Ansätze zur Lösung des Wörterbuchproblems:

11. Strukturierung der aktuellen Schlüsselmenge: z.B. Listen (Kap. 11: Skiplisten), Bäume, Graphen,...
12. Aufteilung des gesamten Schlüssel-Universums: Hashing (Kap. 12: Hashing und Perfektes Hashing)

String Matching ist die Suche von Wörtern in Texten und gehört zu den Wörtersuchproblemen.

13. String Matching (Kap. 13)

# Literatur

- Cormen, Leiserson, Rivest, Stein: Algorithmen – Eine Einführung, z.B. 4. Auflage, 2011, Kap. 11 Hashtabellen

# 12.1 Einführung

**Wörterbuchproblem** Verwalte eine Menge von Schlüsseln (mit Daten).
Operationen: `Insert, Search, Delete`

**Erinnerung** Schlüssel können geordnet und ungeordnet sein

**Erinnerung** „Lösungen“ z. B.

- lineare Listen (einfach, langsam, deterministisch)
- AVL-Bäume (schnell, kompliziert, deterministisch, nur für geordnete Schlüsseluniversen)
- Skiplisten (schnell, eher unkompliziert, randomisiert, nur für geordnete Schlüsseluniversen)
- Hashing (schnell im Average Case, langsam im Worst Case, einfach, randomisiert)

# Hashing

Idee von Hashing:

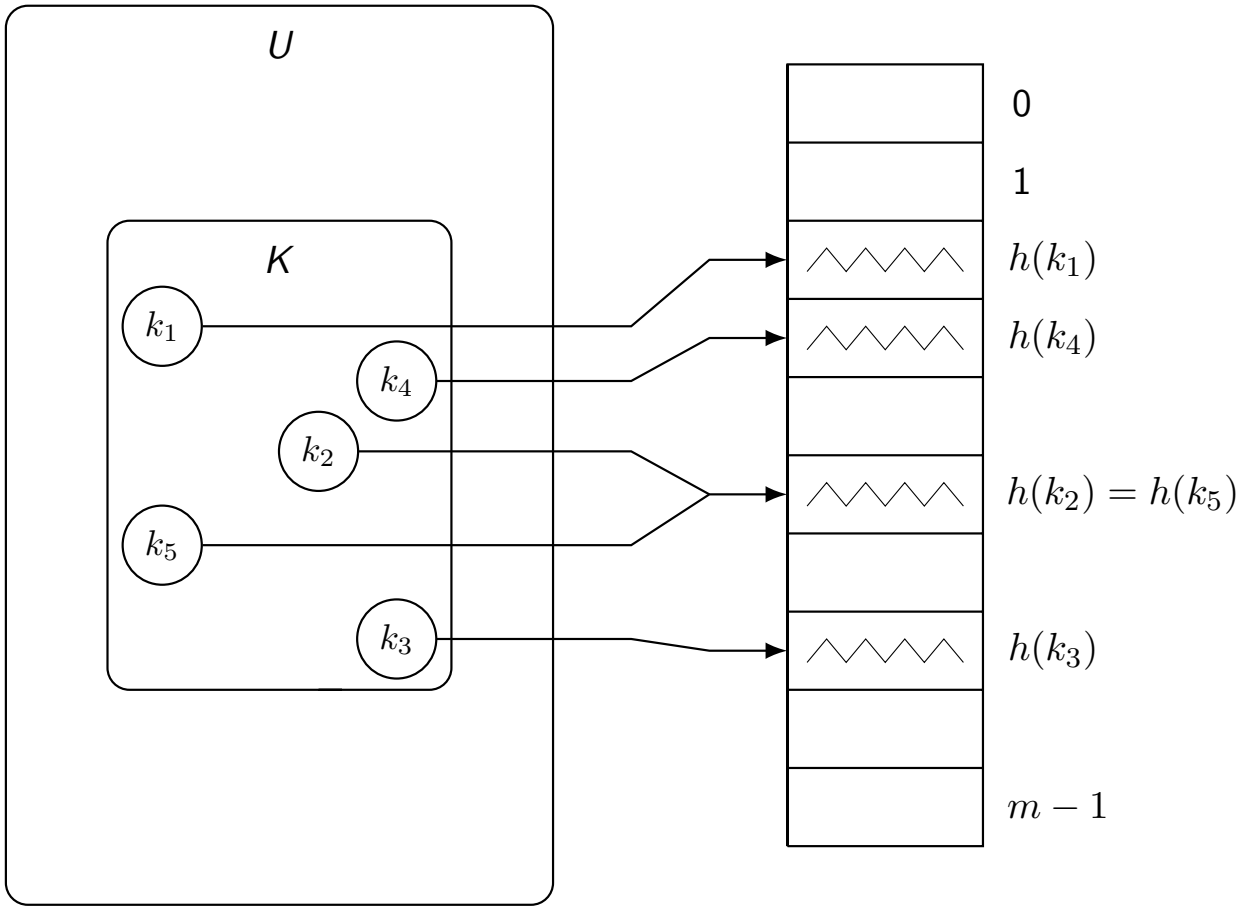
- Ermittle die Position eines Schlüssels durch eine arithmetische Berechnung statt durch Schlüsselvergleiche
- Dazu: Datenstruktur Hashtabelle: speichert $M$ Schlüssel mit Hashadressen $0, ..., M-1$

Situation:

- Schlüssel $S$ aus Universum $\mathcal{U}$: $S \subseteq \mathcal{U}$, $|S| = n$
- Hashfunktion $h: \mathcal{U} \to \{0, 1, \ldots, M-1\}$ $(|\mathcal{U}| \gg M)$
- Speicherung von Schlüssel $x$ an Position $T[h(x)]$ in Hashtabelle
- Kollision $x \neq y \in \mathcal{U}$ mit $h(x) = h(y)$
- Geburtstagsparadoxon: $M = 365, n = 23$ Kollisionen unvermeidlich

# Illustration von Hashing

- Schlüssel $S$ aus Universum $\mathcal{U}$: $S \subseteq \mathcal{U}$, $|S| = n$
- Hashfunktion $h\colon \mathcal{U} \to \{0, 1, \ldots, M-1\}$ $(|\mathcal{U}| \gg M)$
- Speicherung von Schlüssel $x$ an Position $T[h(x)]$



# Offene Fragen

- Was sind gute Hashfunktionen?
- Wie geht man mit Kollisionen um?

# 12.2 Wahl der Hashfunktion

- Ziel: Hashadressen sollen gleichverteilt in $\{0, \ldots, M-1\}$ sein
- Hashfunktionen sollen Häufungen fast gleicher Schlüssel möglichst gleichmäßig auf den Adressbereich streuen

- Generalannahme: Schlüssel sind nicht-negative ganze Zahlen
- Z.B.: Lösung für Character-Strings: deren ASCII-Code ist Nummer in [0,. . . ,127], z.B.

$$\texttt{p} \mathrel{\hat{=}} 112, \quad \texttt{t} \mathrel{\hat{=}} 116 \quad \Longrightarrow \quad \texttt{pt} \mathrel{\hat{=}} 112 \times 128 + 116 = 14452$$

# Divisions-Rest-Methode

## Definition (Divisions-Rest-Methode)

Die Hashfunktion der Divisions-Rest-Methode ist gegeben durch

$$h(x) = x \bmod M$$

**Eigenschaften**

(1) $h(x)$ kann schnell berechnet werden.

(2) Die richtige Wahl von $M$ (Tabellengröße) ist sehr wichtig.

# Gute Wahl von $M$

- **Gute Wahl:** Primzahl $M$, die kein $r^i \pm j$ teilt.
- Vermeide z.B.
  - $M = 2^i$: Alle bis auf die letzten Binärziffern werden ignoriert.
  - $M = 10^i$ oder $M = r^i$: Analog bei Dezimal- bzw. $r$-adischen
  - $M = r^i \pm j$ für kleines $j$, z.B. $M = 2^7 - 1 = 127$:

    $$\texttt{pt} \mathrel{\hat{=}} (112 \cdot 128 + 116) \bmod 127 = 14452 \bmod 127 = 101$$
    $$\texttt{tp} \mathrel{\hat{=}} (116 \cdot 128 + 112) \bmod 127 = 14960 \bmod 127 = 101$$

    Dasselbe passiert, wenn in einem längeren String zwei Buchstaben vertauscht werden.
- Praktisch bewährt!
- **Beispiel:** Die Hashtabelle soll circa 700 Einträge aufnehmen, die Schlüssel sind Dualzahlen.

  Gute Wahl: $M = 701$, da $2^9 = 512$ und $2^{10} = 1024$.

# Multiplikationsmethode

## Definition (Multiplikationsmethode)

Die Hashfunktion der Multiplikationsmethode ist gegeben durch

$$h(x) = \lfloor M(x \cdot A \bmod 1) \rfloor = \lfloor M \underbrace{(x \cdot A - \lfloor x \cdot A \rfloor)}_{\in [0,1)} \rfloor$$

mit $0 < A < 1$.

**Eigenschaften**

(1) Die Wahl von $M$ ist unkritisch.

(2) Wir erhalten eine gleichmäßige Verteilung für $U = \{1, 2, \ldots, n\}$ bei einer guten Wahl von $A$.

# Multiplikationsmethode: Wahl von $A$

Irrationale Zahlen sind eine gute Wahl, denn:

**Satz:** Sei $\xi$ eine irrationale Zahl. Plaziert man die Punkte

$$\xi - \lfloor \xi \rfloor,\ 2\xi - \lfloor 2\xi \rfloor,\ \ldots\ ,\ n\xi - \lfloor n\xi \rfloor$$

in das Intervall $[0, 1]$, dann haben die $n + 1$ Intervallteile höchstens drei verschiedene Längen.
Außerdem fällt der nächste Punkt

$$(n+1)\xi - \lfloor (n+1)\xi \rfloor$$

in einen der größten Intervallteile.
(Beweis: Vera Turán Sos [1957])

# Multiplikationsmethode: Wahl von $A$

Beste Wahl für $A$ nach Knuth [1973]:

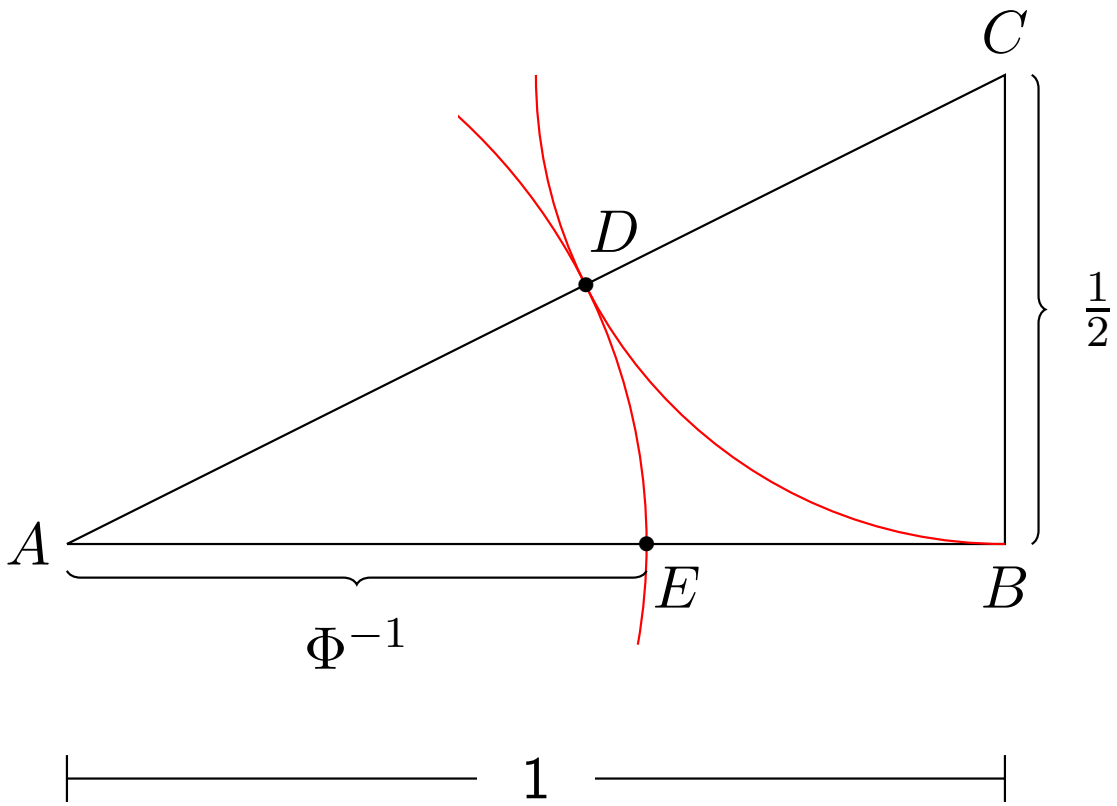
$$A = \Phi^{-1} = \frac{2}{1+\sqrt{5}} = -\hat{\Phi} = \frac{\sqrt{5}-1}{2} = 0.6180339887\ldots$$

$\Phi^{-1}$ ist bekannt als der „goldene Schnitt“.

**Beispiel:** (Dezimalrechnung) $x = 123456,\ M = 10000,\ A = -\hat{\Phi}$

$$\begin{aligned} h(x) &= \lfloor 10000 \cdot (123456 \cdot 0.61803\ldots \bmod 1) \rfloor \\ &= \lfloor 10000 \cdot (76300.0041151\ldots \bmod 1) \rfloor \\ &= \lfloor 10000 \cdot 0.0041151\ldots \rfloor \\ &= \lfloor 41.151\ldots \rfloor \\ &= 41 \end{aligned}$$

# Der goldene Schnitt



# Der goldene Schnitt

- ist behandelt in Euklids *Elemente*,
- wird von Kepler als die „göttliche Teilung“ bezeichnet,
- findet sich in vielfacher Weise in der Natur,
  - z.B. Durchmesserverhältnis benachbarter Nautilus-Spiralkammern,
  - z.B. Verhältnis von Sonnenblumenkernspiralen,
  - z.B. beim Menschen (Leonardo da Vinci)
    - Verhältnis Scheitel/Sohle zu Nabel/Sohle,
    - Verhältnis Hüfte/Sohle zu Knie/Sohle,
    - Verhältnis Schulter/Fingerspitzen zu Ellbogen/Fingerspitzen,
- bestimmt Maßverhältnisse in Kunstwerken,
  - z.B. von Michelangelo, Dürer, da Vinci,
- erscheint in kompositorischen Strukturen,
  - z.B. von Mozart, Beethoven, Bartók, Debussy, Schubert,
- bestimmt Maßverhältnisse in der Architektur,
  - z.B. ägyptischer Pyramiden,
  - z.B. antiker Gebäude,
  - z.B. des UN-Hauptquartiers in New York.

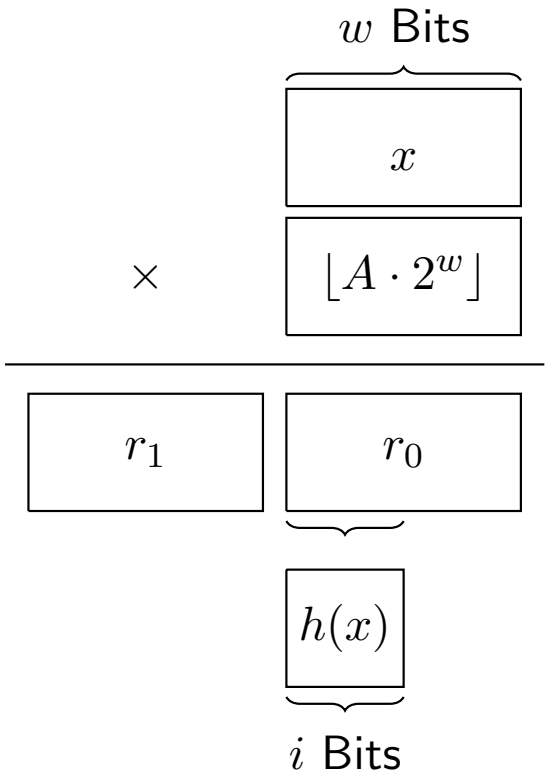
# Multiplikationsmethode: Wahl von $M$

Eine gute Wahl ist $M = 2^i$.

Ist dann $A$ von der Form $0.\underbrace{\cdots\cdots\cdots}_{\leq w \text{ Bits}}$

(eine praktisch vernünftige Annahme!),

so kann $h(x)$ nach folgendem Schema effizient berechnet werden:



# Multiplikationsmethode: Beispiel zur Berechnung

**Beispiel:** $i = 5,\ w = 8,\ A = 0.101,\ x = 101111$

```
x mal A = 101111 x 0.101
          --------------
                 101.111
              10111.1
          --------------
              11101.011
            - 11101
          --------------
                    0.011 x 2^5 = 1100 = h(x)
```

# Multiplikationsmethode: Beispiel zur Berechnung ff

**Beispiel:** $i = 5,\ w = 8,\ A = 0.101,\ x = 101111$

```
        00101111
      x 10100000
--------------
    10111100000
 1011110000000 Ueberlauf
-------------- ignorieren!
 11101011000000
      h(x)
```

Übungsaufgabe: Erklären Sie, warum das funktioniert.

# Divisions-Restmethode vs. Multiplikationsmethode

Empirische Untersuchungen zeigen:

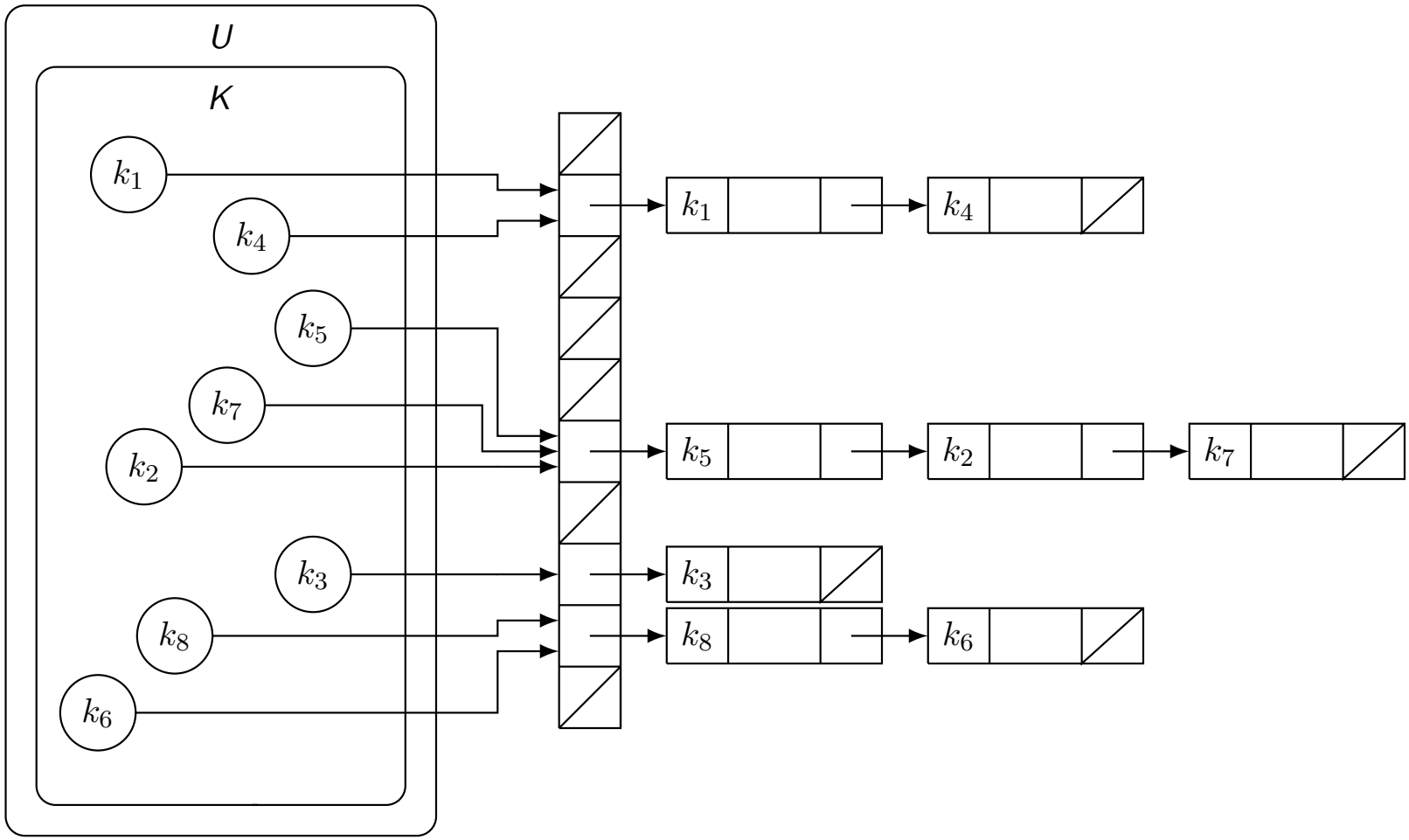
Divisions-Rest-Methode ist besser

Jetzt: Umgang mit Kollisionen

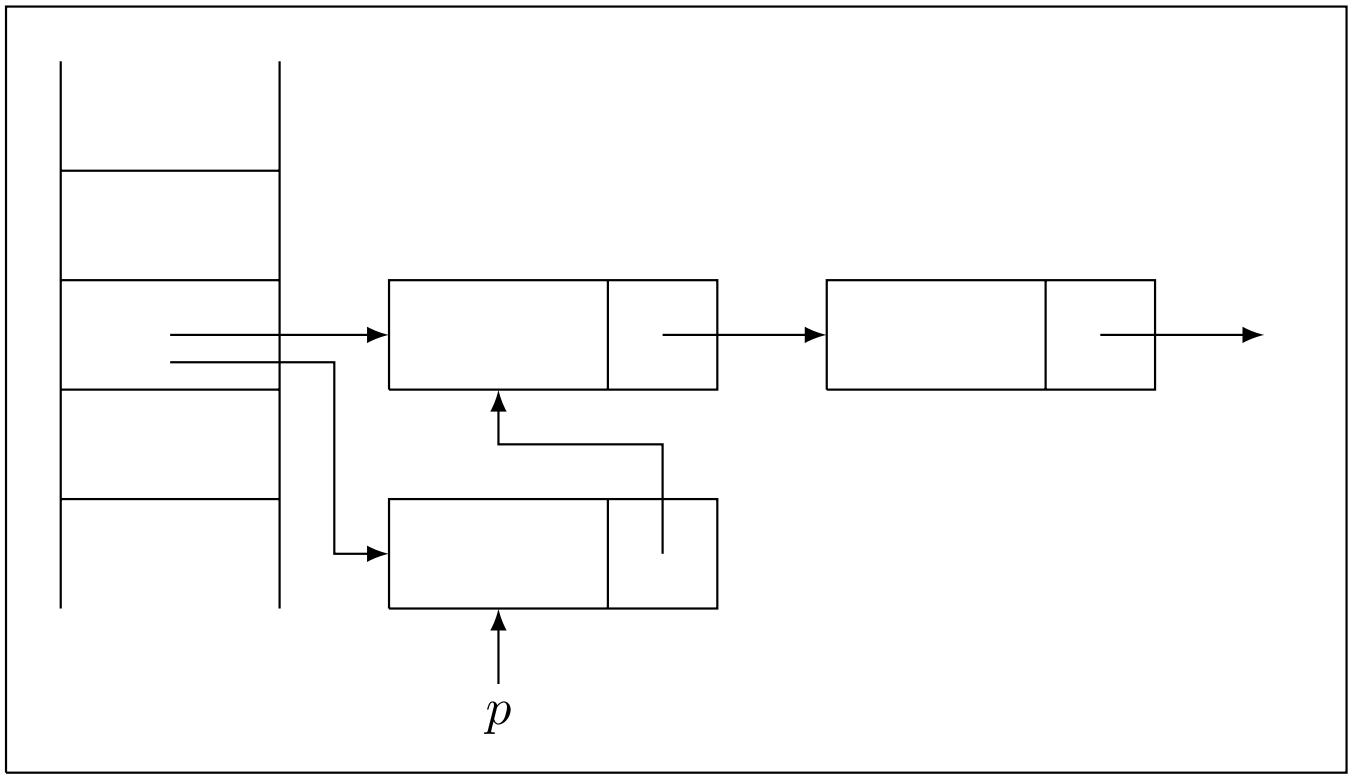
- Hashing mit Verkettung (Offenes Hashing)
- Hashing mit offener Adressierung (Geschlossenes Hashing)

# 12.3 Hashing mit Verkettung

Methode: Jedes Element der Hashtabelle ist eine Referenz auf eine Überlaufkette, die als verkettete lineare Liste implementiert ist.



# Illustration für `Insert`



# Analyse von Search

## Definition (Belegungsfaktor einer Hashtabelle)

Wir nennen die durchschnittliche Anzahl von Elementen in einer Überlaufkette

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

den Belegungsfaktor (Auslastungsfaktor) der Hashtabelle.

Im **worst case** (alle Elemente hashen auf denselben Platz) haben wir offensichtlich $\Theta(n)$ Suchaufwand.

# Average Case Analyse von `Search`

Annahmen:

(1) Ein gegebenes Element hasht auf jeden der $M$ Plätze mit gleicher Wahrscheinlichkeit $\frac{1}{M}$, unabhängig von den anderen Elementen.

(2) Jeder der $n$ gespeicherten Schlüssel ist mit gleicher Wahrscheinlichkeit der Gesuchte.

(3) `Insert` fügt neue Elemente am Anfang der Liste ein.

(4) Die Berechnung von $h(x)$ benötigt konstante Zeit und bleibt außen vor.

# Average Case Analyse

## Theorem (Analyse Hashing mit Verkettung)

*Die erwartete Anzahl der betrachteten Einträge bei Hashing mit Verkettung bei einem Belegungswert* $0 < \alpha = \frac{n}{M} < 1$ *beträgt für*

- *erfolglose Suche:* $n/M = \alpha$
- *erfolgreiche Suche:* $1 + \frac{\alpha}{2} - \frac{1}{2M} = \theta(1 + \alpha)$

Beweis: siehe später in Kapitel: Perfektes Hashing

# Diskussion: Hashing mit Verkettung

Vorteile:

- Belegungsfaktor von mehr als 1 ist möglich
- Echte Entfernungen von Einträgen sind möglich
- eignet sich für Externspeichereinsatz (Hashtabelle im Internspeicher, Listen extern)

Nachteile:

- Zu den Nutzdaten kommt der Speicherplatzbedarf für die Zeiger
- Speicherplatz der Tabelle wird nicht genutzt, dies ist umso schlimmer, wenn in der Tabelle viele Plätze leer bleiben

# 12.4 Hashing mit offener Adressierung

- Methode: Alle Elemente werden in der Hashtabelle gespeichert. Wenn ein Platz belegt ist, so werden in einer bestimmten Reihenfolge weitere Plätze ausprobiert $\rightarrow$ Sondierungsreihenfolge.
- Hierzu: Erweiterung der Hashfunktion auf zwei Argumente:

$$h : U \times \{0, 1, \ldots, M-1\} \rightarrow \{0, 1, \ldots, M-1\}.$$

- Die Sondierungsreihenfolge ergibt sich dann durch

$$\langle h(x, 0), h(x, 1), \ldots, h(x, M-1) \rangle$$

- Diese ist idealerweise eine Permutation von $\langle 0, 1, \ldots, M-1 \rangle$.

# Hashing mit offener Adressierung

- Annahme: Die Hashtabelle enthält immer wenigstens einen unbelegten Platz.
- **Eigenschaften:**
  (1) Stößt man bei der Suche auf einen unbelegten Platz, so kann die Suche erfolglos abgebrochen werden.
  (2) Wegen (1) wird bei `Delete` nicht wirklich entfernt, sondern nur als „entfernt“ markiert. Beim Einfügen wird ein solches Element als nicht vorhanden, beim Suchen als „entfernt vorhanden“ betrachtet.
- **Nachteil:** Die Suchzeit ist nicht mehr proportional zu $(1 + \alpha)$. Wenn `Delete` benötigt wird, ist „Hashing mit Verkettung“ die bessere Wahl. Deshalb nehmen wir jetzt an, dass `Delete` nicht benötigt wird.

# Lineares Sondieren

## Definition (Lineares Sondieren)

Gegeben sei eine normale Hashfunktion $h' : U \to \{0, 1, \ldots, M-1\}$. Für Lineares Sondieren definieren wir

$$h(x, i) = (h'(x) + i) \bmod M$$

für $i = 0, 1, \ldots, M-1$.

**Kritik:** Es gibt nur $M$ verschiedene Sondierungsreihenfolgen, da die erste Position die gesamte Sequenz festlegt:

$$h'(x), h'(x)+1, \ldots, M-1, 0, 1, \ldots, h'(x)-1.$$

# Lineares Sondieren

**Beispiel:** $M = 8$, $h'(x) = x \bmod M$

Schlüssel

| $x$ | 10 | 19 | 31 | 22 | 14 | 16 |
|---|---|---|---|---|---|---|
| $h'(x)$ | 2 | 3 | 7 | 6 | 6 | 0 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 14 | 16 | 10 | 19 | | | 22 | 31 |

Die durchschnittliche Zeit für eine erfolgreiche Suche

| $x$ | 10 | | 19 | | 31 | | 22 | | 14 | | 16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | + | 1 | + | 1 | + | 1 | + | 3 | + | 2 | = | 9 |

ist $\frac{9}{6} = 1.5$.

Lange belegte Teilstücke tendieren zu schnellerem Wachstum als kurze: „Primäre Häufung“.

# Lineares Sondieren: Analyseergebnisse

**Theorem (Analyse Hashing mit Linearem Sondieren)**

*Die erwartete Anzahl der betrachteten Einträge (Sondierungen) bei Linearem Sondieren bei einem Belegungswert* $0 < \alpha = \frac{n}{M} < 1$ *beträgt für*

- *erfolglose Suche:* $\leq \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$
- *erfolgreiche Suche:* $\leq \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$

(Ohne Beweis)

# Quadratisches Sondieren

## Definition (Quadratisches Sondieren)

Gegeben sei eine normale Hashfunktion $h' : U \to \{0, 1, \ldots, M-1\}$. Für Quadratisches Sondieren definieren wir für $i = 0, 1, \ldots, M-1$:

$$h(x, i) = (h'(x) + c_1 i + c_2 i^2) \bmod M.$$

**Beispiel:** $M = 8$, $h'(x) = x \bmod M$, $c_1 = c_2 = \frac{1}{2}$, gleiche Schlüssel wie vorhin ($h(x, i)$ muss ganzzahlig sein!)

| $x$ | 10 | 19 | 31 | 22 | 14 | 16 |
|---|---|---|---|---|---|---|
| $h'(x)$ | 2 | 3 | 7 | 6 | 6 | 0 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | | 10 | 19 | | | 22 | 31 |

# Quadratisches Sondieren

$$14 \rightarrow 6 \rightarrow \left(6 + \tfrac{1}{2}(1 + 1^2)\right) \bmod 8 = 7$$
$$\rightarrow \left(6 + \tfrac{1}{2}(2 + 2^2)\right) \bmod 8 = 1$$

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 16 | 14 | 10 | 19 | | | 22 | 31 |

Die durchschnittliche Zeit für eine erfolgreiche Suche

| $x$ | 10 | | 19 | | 31 | | 22 | | 14 | | 16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | + | 1 | + | 1 | + | 1 | + | 3 | + | 1 | = | 8 |

ist $\frac{8}{6} = 1.33$.

# Quadratisches Sondieren

- **Kritik:** Wie beim linearen Sondieren gibt es nur $M$ verschiedene Sondierungsfolgen. $\rightarrow$ „Sekundäre Häufung“
- **Aber:** Bei richtiger Wahl von $M$, $c_1$, $c_2$ erhalten wir in der Praxis eine bessere Verteilung.

**Theorem (Analyse Hashing mit Quadratischem Sondieren)**

*Die erwartete Anzahl der betrachteten Einträge (Sondierungen) bei Quadratischem Sondieren bei einem Belegungswert* $0 < \alpha = \frac{n}{M} < 1$ *beträgt für*

- *erfolglose Suche:* $\leq \frac{1}{1-\alpha} - \alpha + \ln\left(\frac{1}{1-\alpha}\right)$
- *erfolgreiche Suche:* $\leq 1 + \ln\left(\frac{1}{1-\alpha}\right) - \frac{\alpha}{2}$

(Ohne Beweis)

# Uniformes Hashing

- **Ideal:** „uniform hashing“: Jeder Schlüssel erhält mit gleicher Wahrscheinlichkeit jede der $M!$ Permutationen von $\{0, 1, \ldots, M-1\}$ als Sondierungsreihenfolge zugeordnet.
- Das ist aber schwierig zu implementieren, später mehr darüber

**Theorem (Analyse Uniformes Hashing)**

*Die erwartete Anzahl der betrachteten Einträge (Sondierungen) bei Uniformem Hashing bei einem Belegungswert* $0 < \alpha = \frac{n}{M} < 1$ *beträgt für*

- *erfolglose Suche:* $\leq \frac{1}{1-\alpha}$
- *erfolgreiche Suche:* $\leq \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

(Ohne Beweis)

# Übersicht über durchschnittliche Anzahl an Sondierungen

| $\alpha$ | Verkettung | | mit offener Adressierung | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | lineares S. | | quadr. S. | | uniform | |
| | erfolgreich | erfolglos | er | el | er | el | er | el |
| 0.5 | 1.250 | 0.50 | 1.5 | 2.5 | 1.44 | 2.19 | 1.39 | 2 |
| 0.9 | 1.450 | 0.90 | 5.5 | 50.5 | 2.85 | 11.40 | 2.56 | 10 |
| 0.95 | 1.475 | 0.95 | 10.5 | 200.5 | 3.52 | 22.05 | 3.15 | 20 |
| 1.0 | 1.500 | 1.00 | - | - | - | - | - | - |

Quadratisches Sondieren ist schon sehr gut!

# Diskussion

- Hashing mit Verkettung hat deutlich weniger Schritte; jedoch Nachteil: hoher Speicherbedarf und evtl. Suchzeit trotzdem langsamer, wegen Zeigerverfolgung
- Quadratisches Hashing ist nahe an Uniform Hashing $\rightarrow$ empfehlenswert
- Lineare Sondierung fällt deutlich ab $\rightarrow$ nicht empfehlenswert
- Belegungsfaktor $\alpha$ sollte nicht höher als $0,9$ sein; besser: $0,7$ oder $0,8$.

# Double Hashing

## Definition (Double Hashing)

Für zwei normale Hashfunktionen $h_1, h_2 : U \to \{0, 1, \ldots, M-1\}$ definieren wir

$$h(x, i) = (h_1(x) + ih_2(x)) \bmod M.$$

- Die Sondierungsreihenfolge hängt in zweifacher Weise vom Schlüssel ab:
  - erste Sondierungsposition (wie gehabt)
  - Schrittweite (neu)
- Es ergeben sich $\Theta(M^2)$ verschiedene Sondierungsreihenfolgen.

# Double Hashing

- Bedingung an $h_2$: Für alle Schlüssel $x$ muss $h_2(x)$ relativ prim zu $M$ sein:

$$\text{ggT}(h_2(x), M) = 1,$$

  sonst wird die Tabelle nicht vollständig durchsucht. (Falls $\text{ggT}(h_2(x), M) = d > 1$, so wird nur $\frac{1}{d}$-tel durchsucht.)
- **Zwei Vorschläge:**
  (1) $M = 2^p$ (schlecht für Divisionsmethode), $h_2(x)$ immer ungerade
  (2) $M$ Primzahl, $0 < h_2(x) < M$, z.B.

$$h_1(x) = x \bmod M \qquad h_2(x) = 1 + (x \bmod M')$$

  mit $M' = M - 1$ oder $M' = M - 2$.

# Double Hashing: Beispiel

**Beispiel für (2):** $M = 7$, $h_1(x) = x \bmod 7$, $h_2(x) = 1 + (x \bmod 5)$

| $x$ | 10 | 19 | 31 | 22 | 14 | 16 |
|---|---|---|---|---|---|---|
| $h_1(x)$ | 3 | 5 | 3 | 1 | 0 | 2 |
| $h_2(x)$ | 1 | 5 | 2 | 3 | 5 | 2 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | | | 10 | | 19 | |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 31 | 22 | | 10 | | 19 | |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 31 | 22 | 16 | 10 | | 19 | 14 |

Die durchschnittliche Zeit für eine erfolgreiche Suche

| $x$ | 10 | | 19 | | 31 | | 22 | | 14 | | 16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | + | 1 | + | 3 | + | 1 | + | 5 | + | 1 | = | 12 |

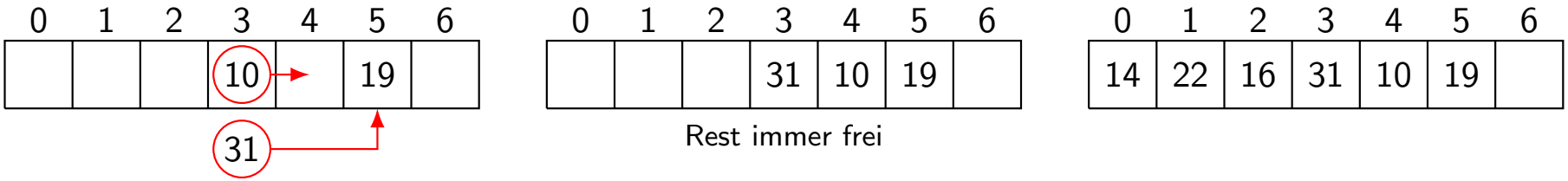
ist $\frac{12}{6} = 2.00$. (Untypisch schlechtes Beispiel für double hashing.)

Double Hashing ist eine gute Approximation an uniformes Hashing!

# Verbesserung von Brent [1973]

- **Idee:** Wenn beim Einfügen von `Item p` mit $x =$ `p.key` ein sondierter Platz $j$ belegt ist mit $x' =$ `T[j].key`, setze
  - $j_1 = (j + h_2(x)) \bmod M$
  - $j_2 = (j + h_2(x')) \bmod M$
- Ist $j_1$ frei oder $j_2$ belegt, so fahre fort wie im Original mit $j = j_1$. Sonst ($j_1$ belegt und $j_2$ frei) setze
  - `T[j2]` $=$ `T[j]`
  - `T[j]` $=$ `p`

# Verbesserung von Brent: Beispiel




Die durchschnittliche Zeit für eine erfolgreiche Suche

| $x$ | 10 | | 19 | | 31 | | 22 | | 14 | | 16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | + | 1 | + | 1 | + | 1 | + | 1 | + | 1 | = | 7 |

ist $\frac{7}{6} = 1.17$.

# Verbesserung von Brent: Analyse

**Theorem (Analyse Double Hashing nach Brent)**

*Die erwartete Anzahl der betrachteten Einträge bei Double Hashing nach Brent bei einem Belegungswert* $0 < \alpha = \frac{n}{M} < 1$ *beträgt für*

- *erfolglose Suche:* $\leq \frac{1}{1-\alpha}$ *(wie uniform)*
- *erfolgreiche Suche:* $\leq 2.5$ *(unabhängig von* $\alpha$*!)*

(Ohne Beweis)