## Analyse von Double-Hashing

- Satz: Wenn Kollisionen mit Double-Hashing aufgelöst werden, dann ist die durchschnittliche Anzahl von Sondierungsschritten in einer Tabelle der Größe $m$ mit $n=\alpha m$ vielen Elementen

$$C_n = \frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right) \quad \text{bzw.} \quad C'_n = \frac{1}{1-\alpha}$$

  für die erfolgreiche bzw. erfolglose Suche.
- Beweis:
  - sehr kompliziert [Guibas & Szemeredi]
  - Idee: Zeige, daß Double-Hashing fast äquivalent zu dem (aufwendigeren) *Random-Hashing* ist
- Random-Hashing: $s(j,k)$ ist eine Folge von Pseudo-Zufallszahlen, die von $k$ abhängt und jeden Slot der Hash-Tabelle gleich wahrscheinlich trifft (mehrfache Hits sind aber möglich)



## Analyse der mittleren Kosten für *Random-Hashing*

- Definiere ZV $X$ = Anzahl der Sondierungen bei erfolgloser Suche
- Sei $P[X \geq i]$ := Wahrscheinlichkeit, daß eine Suche $i$ oder mehr Sondierungsschritte machen muß, $i = 1, 2, \ldots$
- Klar: $P[X \geq 1] = 1$
- $P[X \geq 2]$ = W'keit, daß erster zufällig untersuchter Slot belegt ist
$$= \frac{n}{m} = \alpha$$
- $P[X \geq 3]$ = W'keit, daß erster, zufällig gewählter Slot belegt ist *und* zweiter, zufällig gewählter Slot besetzt ist
$$= \frac{n}{m} \cdot \frac{n}{m} = \alpha^2$$

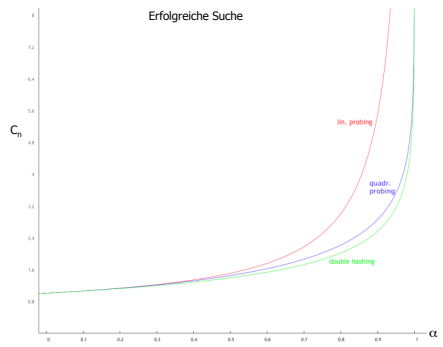


$$\begin{aligned}
E[X] &= \sum_{i=1}^{\infty} i \cdot P[X = i] \\
&= 1 \cdot P[X = 1] + 2 \cdot P[X = 2] + \cdots \\
&= P[X = 1] + \\
&\quad P[X = 2] + P[X = 2] + \\
&\quad P[X = 3] + P[X = 3] + P[X = 3] + \\
&\quad \vdots \qquad \rightarrow \text{Additivität, da Ereignisse unabhängig} \\
&= P[X = 1 \vee X = 2 \vee \ldots] + \\
&\qquad P[X = 2 \vee X = 3 \vee \ldots] + \\
&\qquad\quad P[X = 3 \vee X = 4 \vee \ldots] + \cdots \\
&= P[X \geq 1] + P[X \geq 2] + P[X \geq 3] + \cdots \\
&= 1 + \alpha + \alpha^2 + \cdots = \frac{1}{1-\alpha}
\end{aligned}$$

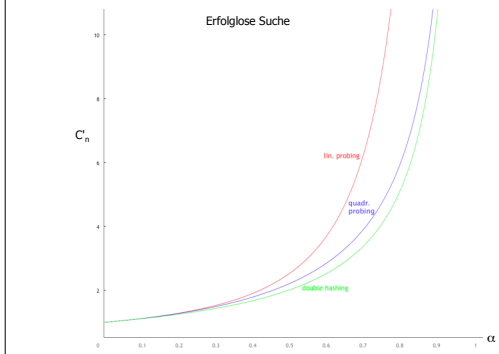


- Fazit:
  - Double-Hashing ist genauso effizient wie randomisiertes Sondieren
  - Double-Hashing ist schneller, um konstanten Faktor (Pseudo-Zufallszahlen sind rel. teuer)

## Vergleich








## Verbesserung der erfolgreichen Suche

- Allgemein: Kollision beim Einfügen bedeutet
  - $k$ trifft in $T[i]$ auf $k_{alt}$, d.h. $i = h(k) - s(j,k) = h(k_{alt}) - s(j', k_{alt})$
- Problem:
  - Sondierungsfolge $h(k) - s(1,k), h(k) - s(2,k), \ldots$ könnte lang werden
  - dieselbe Folge muß man später bei der Suche nach $k$ wieder durchlaufen
  - evtl. wäre die Folge $h(k_{alt}) - s(j'+1, k_{alt}), h(k_{alt}) - s(j'+2, k_{alt}), \ldots$ viel schneller auf einen leeren Slot gestoßen
- Idee: suche freien Platz für $k$ oder $k_{alt}$
- 2 Möglichkeiten:

  (M1) $k_{alt}$ bleibt in $T[i]$: betrachte neue Position $h(k) - s(j+1, k)$ für $k$

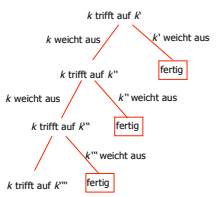
  (M2) $k$ verdrängt $k_{alt}$: betrachte neue Position $h(k_{alt}) - s(j'+1, k_{alt})$ für $k_{alt}$
  - solange (M1) und (M2) auf belegten Slot stoßen, verfolge (M1) oder (M2) weiter



## Brent's Verfahren

- Verfolge (M2) = Verdrängung nur 1x, sonst (M1)
- Annahme: Double-Hashing, d.h., wenn man in Sondierungsfolge bei Slot $i$ steht, ist der nächste Slot (in der Sondierungsfolge) bei $i$ - $h'(k)$ mod $m$

```
i = h( k )
while T[i] belegt:
    k_alt = T[i]
    i1 = (i-h'(k)) % m
    i2 = (i-h'(k_alt)) % m
    if T[b1] belegt and T[b2] frei:
        # verdränge k_alt
        T[i] = k
        k = k_alt
        i = i2
    else:
        # leave k_alt in its slot
        i = i1
T[i] = k
```



- Analyse (o.Bew.):
  - Kosten für erfolglose Suche bleiben unverändert:

$$C'_n \approx \frac{1}{1-\alpha}$$

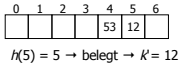
  - Kosten für erfolgreiche Suche im Schnitt:

$$C_n \approx 1 + \frac{\alpha}{2} + \frac{\alpha^3}{4} + \frac{\alpha^4}{15} + \cdots < 2.5$$



Beispiel

- Hash-Funktionen: $h(k) = k \bmod 7$
  $h'(k) = 1 + (k \bmod 5)$
- Schlüsselfolge: 12, 53, 5, 15, 2, 15
- Betrachte:
  5 - $h'(k)$ = 5 - $h'$(5) = 4 belegt
  und 5 - $h'(k')$ = 5 - $h'$(12) = 3 frei
  → 5 verdrängt 12 von seinem Platz




$h$(5) = 5 → belegt → $k'$ = 12