# Informatik

Hashing

Zum Abspeichern und Wiederfinden von Objekten wäre folgende Funktion hilfreich:

f: Objekte $\rightarrow \mathbb{N}$

Dann könnte Objekt x unter Adresse f(x) gespeichert werden.

f heißt Hashfunktion.

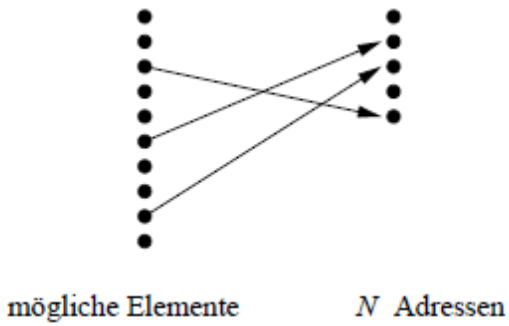
Zum Abspeichern und Wiederfinden von Objekten wäre folgende Funktion hilfreich:

f: Objekte $\rightarrow \mathbb{N}$

Dann könnte Objekt x unter Adresse f(x) gespeichert werden.

f heißt Hashfunktion.

Problem: Anzahl Objekte ist möglicherweise größer als die Anzahl der Adressen.




Kollision: wenn $f(x) = f(y)$ für $x \neq y$

Beispiel einer einfachen Hashfunktion.

Gegeben seien die Adressen von 0 bis $N - 1$.

f: Menge der Strings $\rightarrow \{0, ...., N - 1\}$

Beispiel einer einfachen Hashfunktion.

Gegeben seien die Adressen von 0 bis $N - 1$.

f: Menge der Strings $\rightarrow \{0, ...., N - 1\}$

f(s) = (Summe aller ASCII-Codes der Zeichen von s ) % N

Beispiel einer einfachen Hashfunktion.

Gegeben seien die Adressen von 0 bis $N - 1$.

f: Menge der Strings $\rightarrow \{0, ...., N - 1\}$

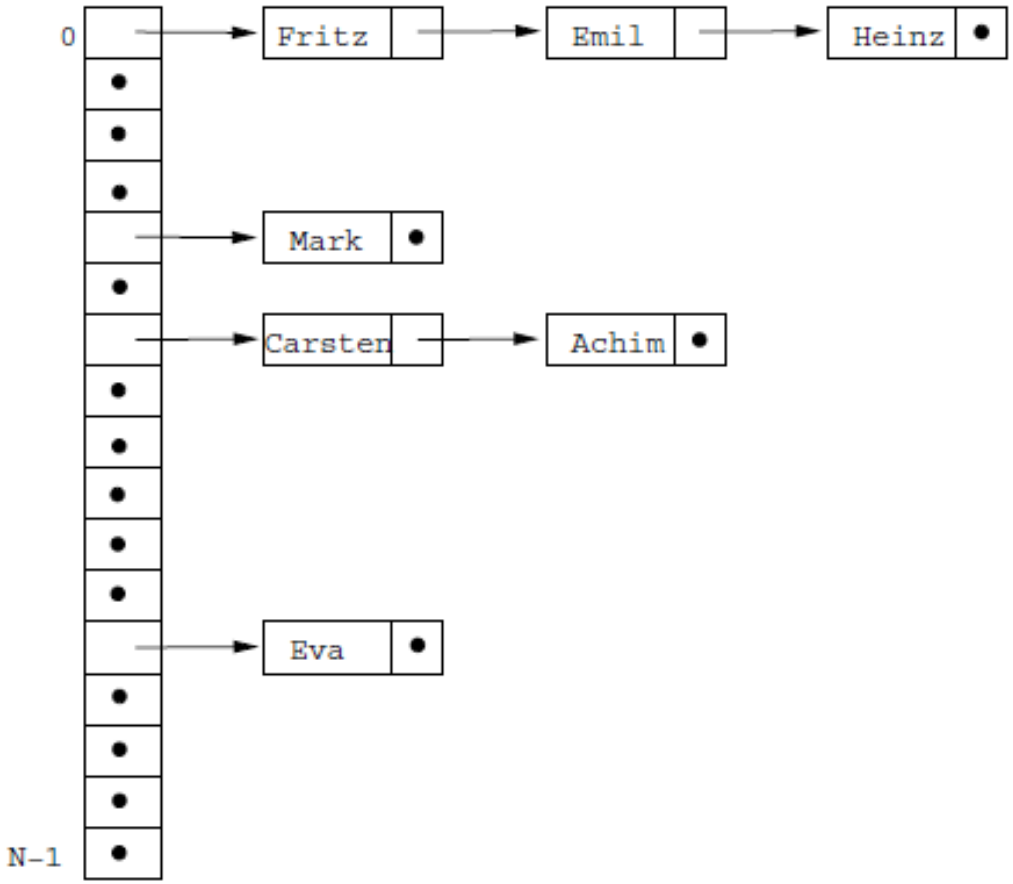
f(s) = (Summe aller ASCII-Codes der Zeichen von s ) % N

Für beliebige Objekte x könnte f für die String-Repräsentation `x.__str()__` einen Wert berechnen.

Diese Hashfunktion funktioniert in vielen Fällen schon ganz gut, wenn N eine Primzahl ist, die nicht nahe bei Potenzen von 2 oder 10 liegt.

# Offenes Hashing

# Offenes Hashing




Datenstruktur für die Implementierung: ein Liste von Listen (ein Array von verketteten Listen)

# Geschlossenes Hashing

# Geschlossenes Hashing

| | | |
|---|---|---|
| 0 | B | Fritz |
| | B | Emil |
| | L | |
| | L | |
| | B | Mark |
| | L | |
| | B | Carsten |
| | G | |
| | L | |
| | B | Heinz |
| | G | |
| | L | |
| | B | Eva |
| | L | |
| | L | |
| | B | Achim |
| N-1 | L | |

L = LEER
B = BELEGT
G = GELOESCHT

Datenstruktur für die Implementierung: eine Liste für die Objekte und eine für die Zustände.

Sondierung: wenn $y = f(x)$ schon belegt, so suche für x einen Alternativplatz

- lineares Sonderieren: $y + 1, y + 2, y + 3, y + 4....$
- quadratisches Sondieren:

Sondierung: wenn $y = f(x)$ schon belegt, so suche für x einen Alternativplatz

- lineares Sonderieren: $y + 1, y + 2, y + 3, y + 4....$
- quadratisches Sondieren: $y + 1, y + 4, y + 9, y + 16, ....$
- double Hashing:

Sondierung: wenn $y = f(x)$ schon belegt, so suche für x einen Alternativplatz

- lineares Sonderieren: $y + 1, y + 2, y + 3, y + 4....$
- quadratisches Sondieren: $y + 1, y + 4, y + 9, y + 16, ....$
- double Hashing: $y + f_2(x), y + 2 \cdot f_2(x), ...$ die Schrittweite wird durch eine 2. Hashfunktion bestimmt.

Sondierung: wenn $y = f(x)$ schon belegt, so suche für x einen Alternativplatz

- lineares Sonderieren: $y + 1, y + 2, y + 3, y + 4....$
- quadratisches Sondieren: $y + 1, y + 4, y + 9, y + 16, ....$
- double Hashing: $y + f_2(x), y + 2 \cdot f_2(x), ...$ die Schrittweite wird durch eine 2. Hashfunktion bestimmt.

Alle Berechnungen werden jeweils mod(N) durchgeführt. Beim quadratischen Sondieren werden ggf. nicht alle Buckets besucht.

Eine *perfekte* Hashfunktion verursacht keine Kollisionen.

Laufzeit bei geschlossenem Hashing

N = Anzahl der möglichen Speicherpositionen, n = Anzahl der gespeicherten Objekte

$\alpha = \frac{n}{N}$ heißt Belegungsgrad oder Auslastungsfaktor.

Als durchschnittliche Anzahl der Sondierungsschritte bei Double-Hashing ergibt sich bei

- erfolgloser Suche: $\approx \frac{1}{1-\alpha} = 5$ für $\alpha = 0.8$
- erfolgreicher Suche: $\approx \frac{\ln(1-\alpha)}{\alpha} = 2.01$ für $\alpha = 0.8$

Laufzeit bei geschlossenem Hashing

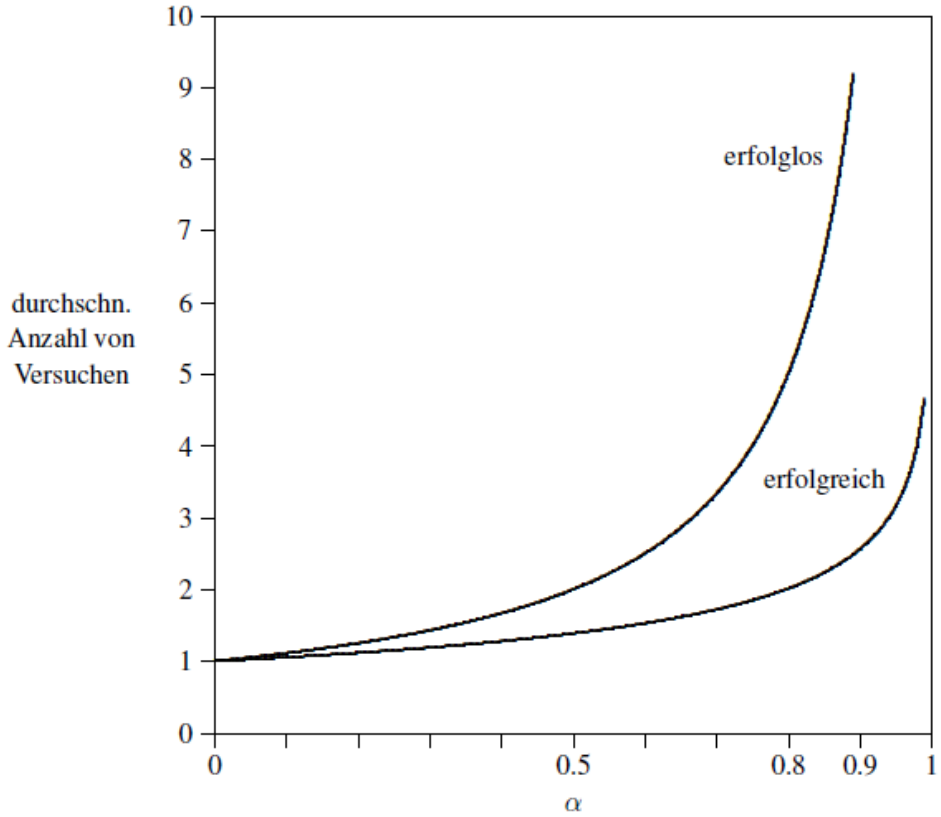
N = Anzahl der möglichen Speicherpositionen, n = Anzahl der gespeicherten Objekte

$\alpha = \frac{n}{N}$ heißt Belegungsgrad oder Auslastungsfaktor.

Als durchschnittliche Anzahl der Sondierungsschritte bei Double-Hashing ergibt sich bei

- erfolgloser Suche: $\approx \frac{1}{1-\alpha} = 5$ für $\alpha = 0.8$
- erfolgreicher Suche: $\approx \frac{\ln(1-\alpha)}{\alpha} = 2.01$ für $\alpha = 0.8$

d.h. in 2 Schritten wird von 1.000.000 Elementen aus einer 1.250.000 großen Tabelle das richtige gefunden. Ein ausgeglichener Suchbaum benötigt dafür etwa 20 Vergleiche.




| | ausgeglichener Suchbaum | geschlossenes Hashing |
|---|---|---|
| Laufzeit | logarithmisch | konstant |
| Speicherbedarf | dynamisch wachsend | in Sprüngen wachsend |
| Sortierung | durch Traversierung | - |