Zeichenketten / Parser / Grammatiken

Vortrag
Philipp Kirchhofer / Benedikt Schneyer
05/2007

Übersicht

- **Parser**
  - Einführung
  - Design
- **Cocke-Younger-Kasami**
- **Loglan**
- **Trie**
  - PATRICIA - Trie
- **Hashing**
  - Hashfunktion
  - Kollisionen
  - Hashtabelle
- **scanf & Scanner**
- **Quellen**

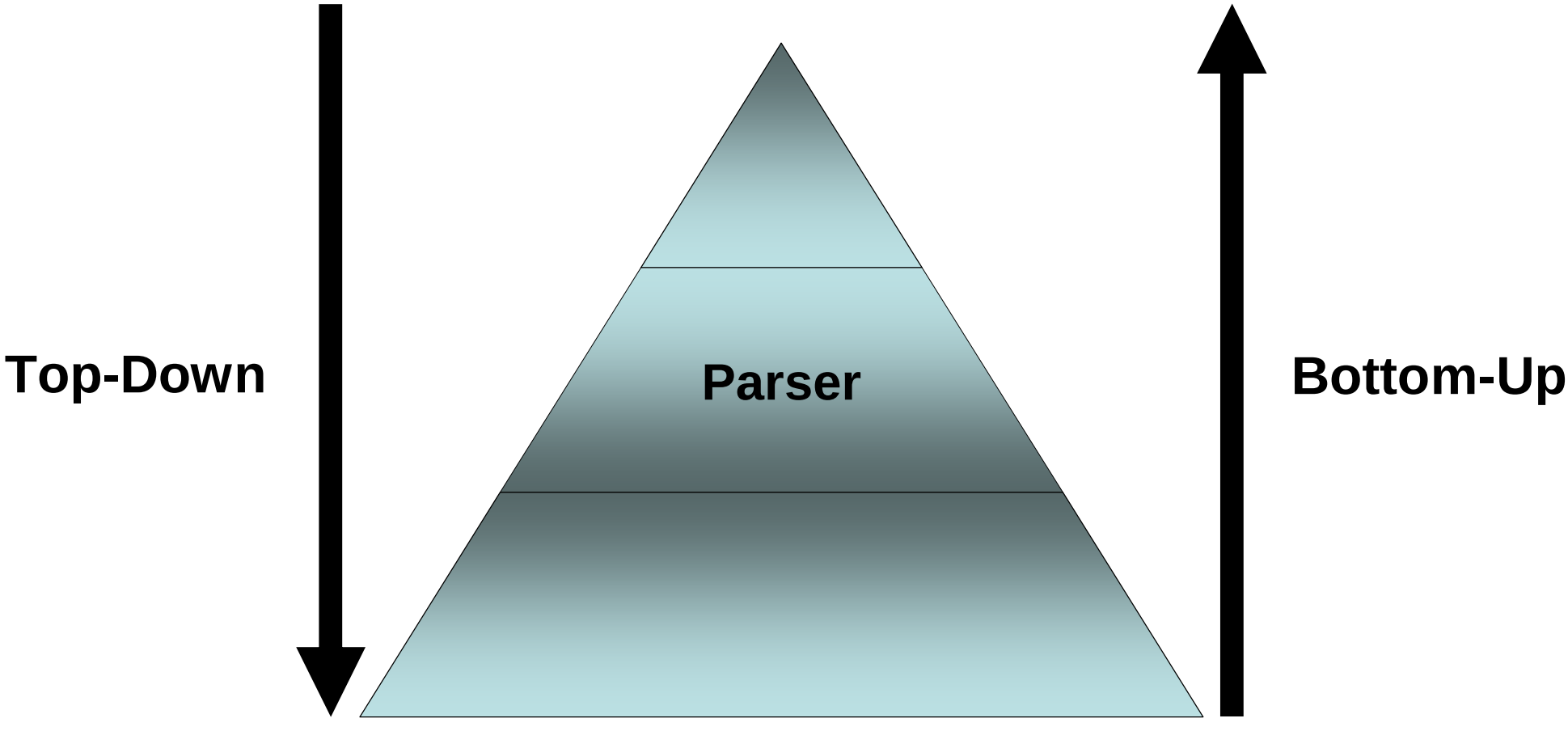
Parser - Einführung

Parser: Zerlegt Eingabedaten in für die Weiterverarbeitung brauchbares Format

- Verwendet **Lexer** für die Zerlegung der Eingabe in **Token**
- **Token** dienen als Eingabe für den eigentlichen Parser
- Prüft anschließend die Syntax der Eingabe

Recognizer: Gibt zurück, ob die Eingabe den Regeln der Grammatik entspricht

Parser-Design



Cocke-Younger-Kasami-Algorithmus

- **Was:**
  - Lösen des Wortproblems für kontextfreie Sprachen
- **Wann:**
  - Wenn die Grammatik in Chomsky-Normalform vorliegt
- **Wie:**
  - Ursprungssymbole für immer größere Teilworte suchen
- **Warum:**
  - Weil er mit einem Aufwand von $O(n^3)$ und einem Speicherbedarf von $O(n^2)$ sonst keine Arbeit findet

CYK – Wie:

- **Wie:**
  - Prämisse suchen aus denen sich die kleinsten Teilworte bilden lassen, beginnend bei n = 1
  - Unter Nutzung der Vorangegangenen Schritte wiederholen
    für n = 2, 3, etc.
  - Ergebnisse des letzten Schrittes auf das Startsymbol untersuchen

CYK – Wie:

Beispiel:

w=	b	a	a	b	a
j \ i	1	2	3	4	5
1					
2					
3					
4					
5					

Grammatik:

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK – Wie:

Beispiel:

w=	b	a	a	b	a
j \ i	1	2	3	4	5
1	B	A, C	A, C	B	A, C
2					
3					
4					
5					

Grammatik:

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

CYK – Wie:

Beispiel:

w=	b	a	a	b	a
j \ i	1	2	3	4	5
1	B	A, C	A, C	B	A, C
2	A, S	B	S, C	A, S	
3	—	B	B		
4	—	S, C A			
5	S, C A				

Grammatik:

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

$S \in V_{1,5}$

also: $w \in L(G)$

Loglan

- künstliche menschliche systematische Sprache
- erfunden in den späten 50ern
- zur Lösung aller zukünftigen Probleme wie:
  - Kommunikation Mensch – Maschine
  - Speicherung von Wissen
  - Sapir-Whorf-Hypothese

Loglan Grammatik:

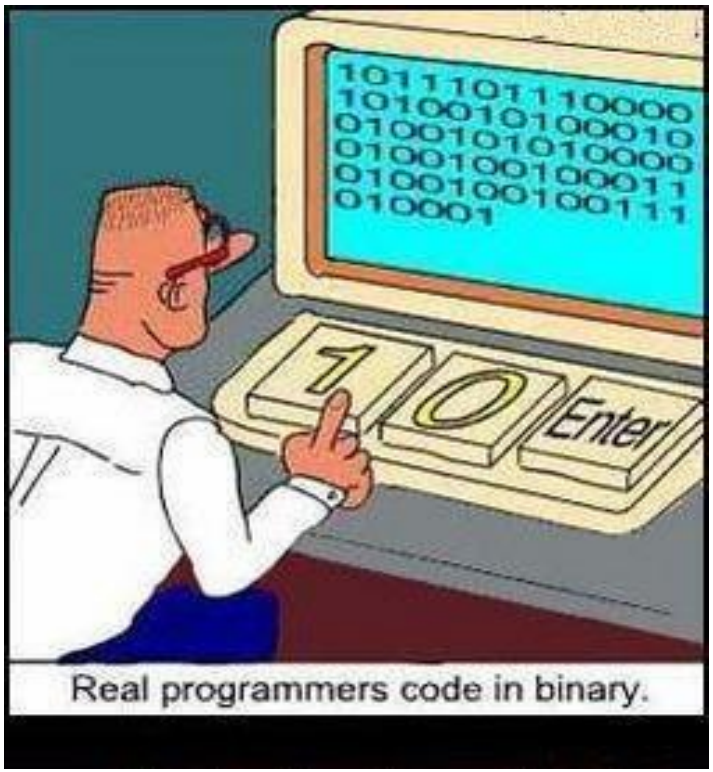
- Drei Arten von Wörtern:
  - « kleine Wörter »
  - Prädikate von der Gestalt: CCVCV oder CVCCV
  - Namen (der Rest)

A	⇒	a | e | i | o | u
MOD	⇒	ga | ge | gi | go | gu
BA	⇒	ba | be | bi | bo | bu
DA	⇒	da | de | di | do | du
LA	⇒	la | le | li | lo | lu
NAM	⇒	{all names}
PREDA	⇒	{all predicates}
<sentence>	⇒	<statement> | <predclaim>
<predclaim>	⇒	<predname> BA <preds> | DA <preds>
<preds>	⇒	<predstring> | <preds> A <predstring>
<predname>	⇒	LA <predstring> | NAM
<predstring>	⇒	PREDA | <predstring> PREDA
<statement>	⇒	<predname> <verbpred> <predname> | <predname> <verbpred>
<verbpred>	⇒	MOD <predstring>

Loglan - Recognizer

- **Aufgabestellung:**
  - Schreibe ein Programm das erkennt ob ein Satz der Loglan Grammatik entspricht
- **Erst implementieren, dann denken:**
  - Cocke-Younger-Kasami-Algorithmus
- **Erst denken, dann implementieren:**
  - Grammatik hat Baumstruktur,

    also einfache Rückwärtsersetzung



Loglan - Recognizer

- **1.Schritt:**
  - Sätze einlesen
    - übliche Einlese-Problematik
    - Lösung mit Scannerklasse
- **2.Schritt:**
  - auf „Worte“ durch Bezeichner ersetzen
- **3.Schritt:**
  - Nichtterminale immer weiter ersetzen
- **4.Schritt:**
  - letztes Symbol überprüfen

- **Beispiel:**

  dJan ga VEDma

  le NEGRO keTpi.

  djan ga vedma le negro ketpi

  <predname> MOD <preds> LA <preds> <preds>

  <predname> MOD <preds> LA <preds>

  <predname> <verbpred> LA <preds>

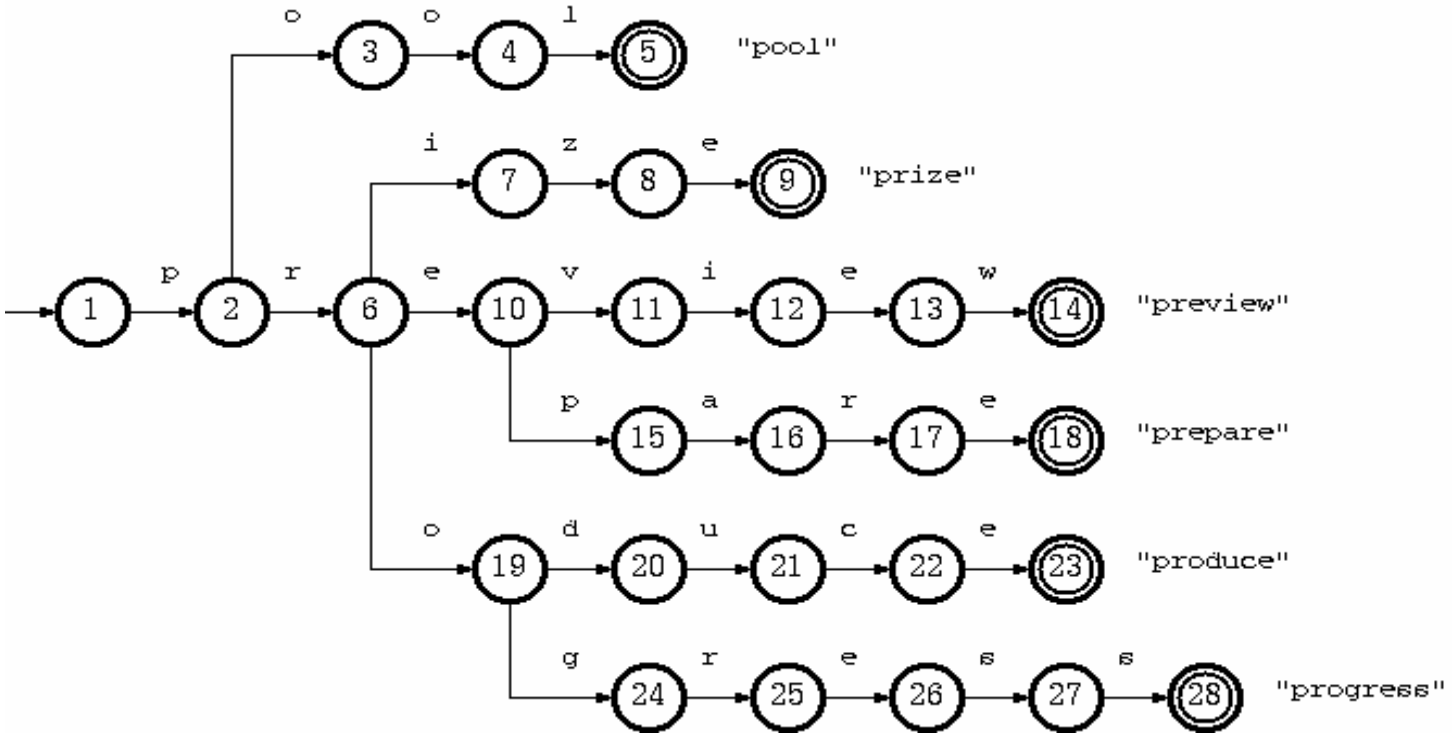
  <predname> <verbpred> <predname>

  <sentence>

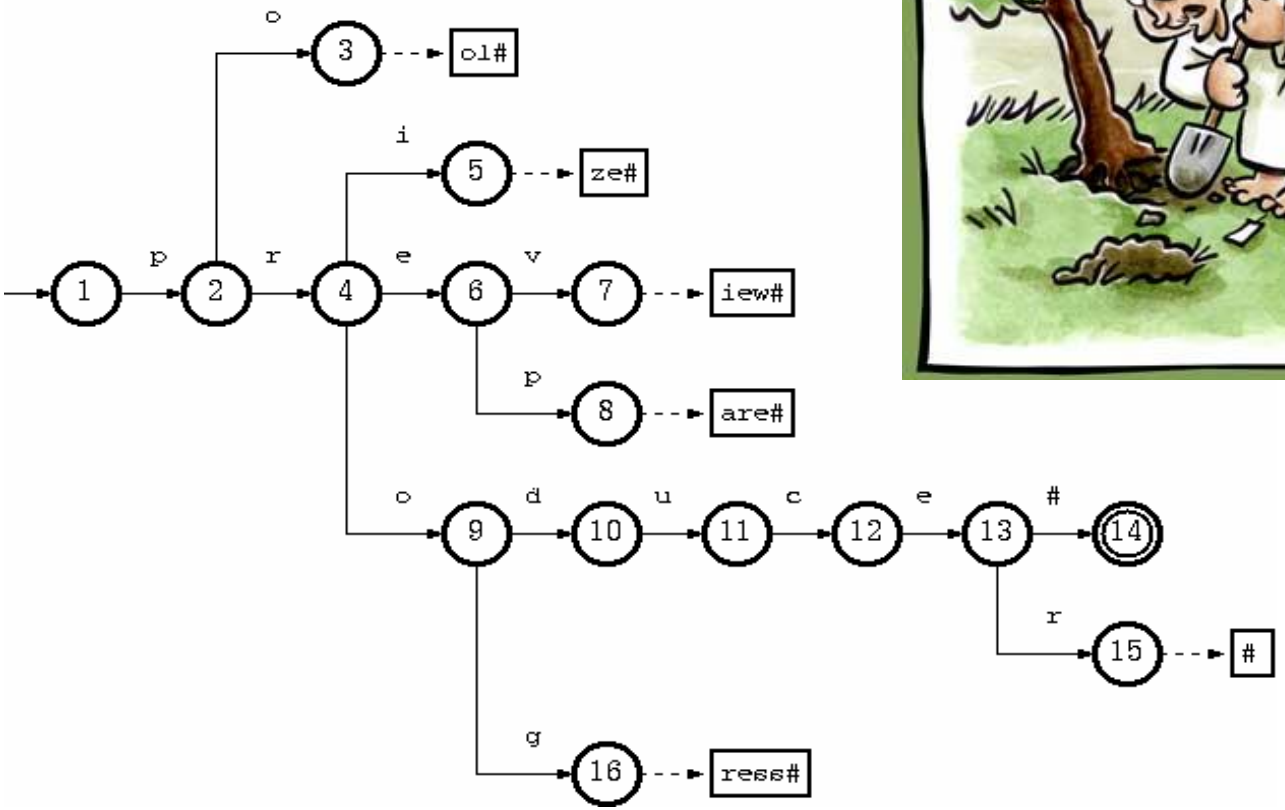
  Good

Trie

- **Datenstruktur:**
  - spezieller Suchbaum
    - ermöglicht effizientes Speichern ähnlicher Zeichenketten



- **PATRICIA Trie:**
  - verkürzter Trie
    - überflüssige Knoten entfernt



Hashing

Allgemein:

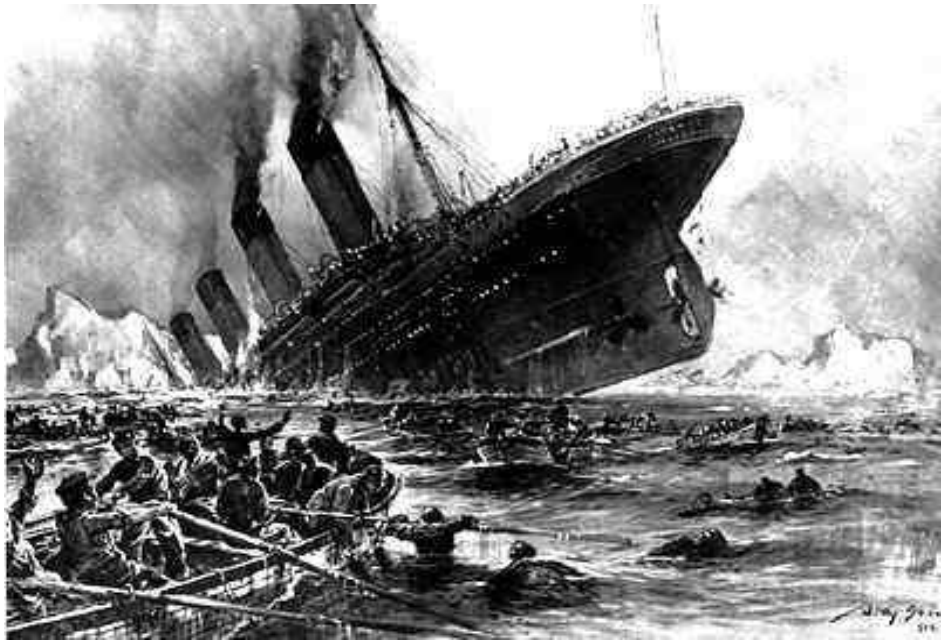
Eigenschaften: Datenreduzierend, Chaotisch, Surjektiv, Effizient

Beispiel:

„Franz jagt im komplett verwahrlosten Taxi quer durch Bayern“

MD5 → **a3cca2b2aa1e3b5b3b5aad99a8529074**

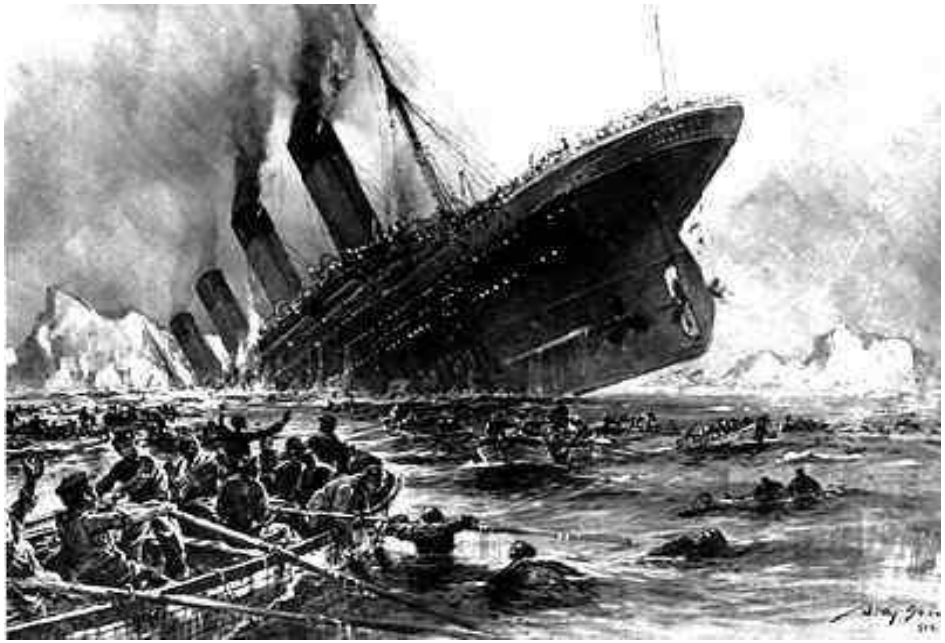
Hashing - Kollisionen



d1 31 dd 02 […] 2b 6f f7 2a 70 → **MD5** → **79054025255fb1a26e4bc422aef54eb4**

d1 31 dd 02 […] ab 6f f7 2a 70 → **MD5** → **79054025255fb1a26e4bc422aef54eb4**

Hashing - Kollisionen



d1 31 dd 02 […] 2b 6f f7 2a 70 —MD5→ **79054025255fb1a26e4bc422aef54eb4**

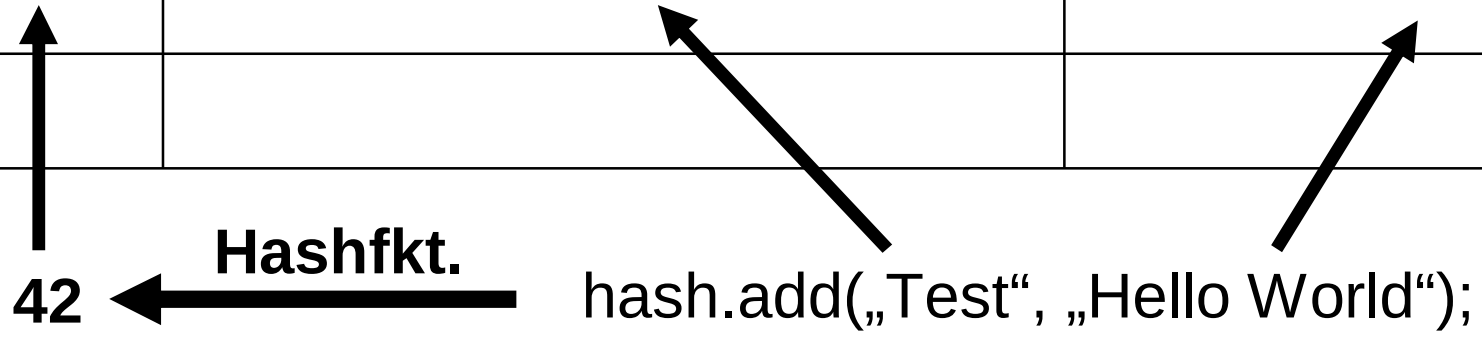
d1 31 dd 02 […] ab 6f f7 2a 70 —MD5→ **79054025255fb1a26e4bc422aef54eb4**

Kollision!

Hashtabelle

Effiziente Speicherung von Zeichenketten
Lookup: O(1)

Index	Schlüssel	Wert
0	FEE1DEAD	CAFEBABE
1	Hello World	Java ist toll!
…		



Bei Kollision: Speicherung in Verlinkter Liste o.ä.

scanf / sscanf

- **scanf:**
  - liest den input auf stdin vorformatiert
    - liefert Anzahl erfolgreich eingelesener Werte zurück
    - scanf(<format>, address)
    - formatspecifier: [=%[*][width][modifiers]type=]
    - sscanf(str, <format>, address
- **Beispiele:**
  - scanf("%f %x", &a, &b)
  - scanf("%s", name)
  - sscanf(input, "%2d", &a)
  - scanf("%*3c)

Scanner

Was ähnliches gibt es auch für Java:

```
Scanner scanner = new Scanner(<Stream>);
while (scanner.hasNext()) {
  String token = scanner.next();
  int number = scanner.nextInt();
  BigInteger big_number = scanner.nextBigInteger();
}
```

Quellen

- Wikipedia
- A Real MD5 Collision (http://www.x-ways.net/md5collision.html)
- http://www.pohlig.de/
- http://www1.informatik.unibw-muenchen.de/
- http://www.loglan.org/
- http://www.cplusplus.com/

Vielen Dank für Eure Aufmerksamkeit!