Algorithmen und Datenstrukturen (EI)

ADS Zentralübung

Stefan Schmid

4. Februar 2009

TECHNISCHE
UNIVERSITÄT
MÜNCHEN

Einturnen...

Ein heutiger Computer aus dem *Saturn* ist im Prinzip eine Turing Maschine?

Nein. Zum Beispiel Sprache $L = \{0^i 1^i, i>0\}$: kein bel. grosser Speicher! „Im Prinzip“ gibt‘s nur endliche Automaten.

11 mod 3 = 2 (z.T. auch geschrieben als: **11 = 2 (mod 3)**)

Ein NDPA kann in einen DPDA umgewandelt werden?

Nein, NPDAs sind mächtiger!

Eine Typ 0 Grammatik kann eine Typ 3 Sprache beschreiben.

Ja.

Einturnen...

Achtung: Im Prinzip ist jede Typ i Sprache automatisch auch vom Typ i-1 für i $\in$ {1,2,3}. Zum Teil (in der Vorlesung!) interessieren wir uns aber nur für das minimale i, das der Sprache entspricht!
Eine Typ 3 Sprache ist dann nie vom Typ 2 zum Beispiel. In der Prüfung werden wir angeben, welche Definition wir betrachten wollen!

Einturnen...

Wenn L r.a. (= semi-entscheidbar) ist und nicht L auch, dann ist L rekursiv (= entscheidbar).

Ja: TM 1 sagt „ja“ wenn $w \in L$, TM 2 sagt „ja“ wenn $w \in$ nicht L, gibt immer eine Antwort!

$3n \in O(n^2)$? Ja.

Ein LBA kann sortieren? Ja.

Ist $a^i b^i$ ein regulärer Ausdruck? Nein, nur Vereinigung, Konkatenation, und Stern erlaubt. Sprache gar nicht regulär! (Zum Teil auch „+“: mind. 1x)

Einturnen...

Ist jede nutzlose Variable nullierbar?

Definition 101
$A \in V$ heißt nutzlos, falls es keine Ableitung

$$S \rightarrow^* w, \quad w \in \Sigma^*$$

gibt, in der A vorkommt.

Definition 52
Ein $A \in V$ mit $A \rightarrow^* \epsilon$ heißt nullierbar.

Nein:

S => a

T => b (nutzlos, aber nicht nullierbar)

Einturnen...

Deterministisch		Nichtdeterministisch	
DFA	=	NFA	(reguläre Sp.)
DPDA	<	(N)PDA	(NPDA = kontextf.)
DLBA	<	(N)LBA	(NLBA = kontexts.)
DTM	=	NTM	

Einturnen...

Gilt: x => y ⬄ ¬ y => ¬ x ?

Ja. Wir prüfen / beweisen?

Zum Beispiel durch Umformen:

x => y ⬄ ¬ x v y (wichtige Formel!)

¬ y => ¬ x ⬄ y v ¬ x

ist das gleiche!

Wahrheitstabelle...

Einturnen...

Betrachte $\langle\{b, c\}, \{\bullet\}\rangle$ mit

$\bullet$	b	c
b	b	b
c	c	c

Was ist also b OP c? **b**

Einturnen...

CFG für 0^i12^i, $i \geq 1$

S => N S 1 Z
N => 0
Z => 2

Nein! Werde S gar nicht los...

S => N S Z | 1
N => 0N
Z => 2Z

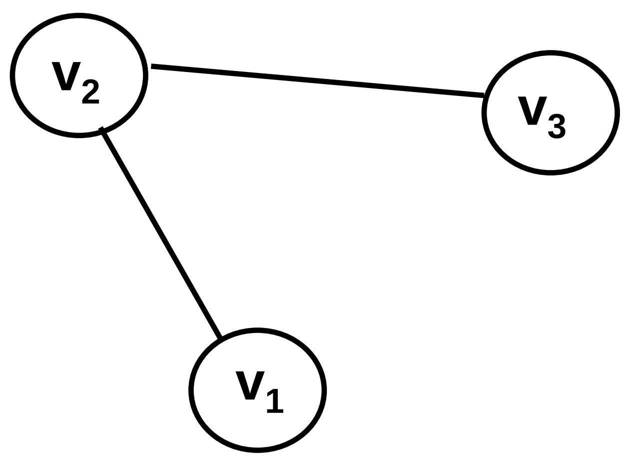
Nein! Ev. mehr 0 als 2!

S => N S Z | 1
N => 0
Z => 2

OK

Einturnen...

$G = (V, E)$ wobei $V = \{v_1,v_2,v_3\}$ und $E = \{\{v_1,v_2\}, \{v_2,v_3\}\}$



Einturnen...

Wie viel mal muss man n durch 2 teilen bis < 2?

n, n/2, n/4, n/8.... => O(log n)

Wie viel mal muss man Wurzel ziehen bis < 2?

n, sqrt(n), sqrt(sqrt n) ... => O(log log n)

Wie viel mal muss man Log ziehen bis < 2?

n, log(n), log(log n) ... => O(log* n)

Einturnen...

Was bedeutet |vx|?

Das ist die Länge des Strings v konkateniert mit x!

Falls v=0011 und x=01, dann |vx| = |001101| = 6

Wie baut man aus einer linkslinearen Typ 3 Grammatik einen EA?

Idee: Mit rechtslinearer Grammatik die „Rückwärtssprache“ bauen, dann Pfeile im EA umdrehen.

Beispiel zu Isomorphismus

Beispiel 17

$\langle \mathbb{R}^+, \cdot \rangle$ und $\langle \mathbb{R}, + \rangle$ $(\mathbb{R}^+ = \{x \in \mathbb{R}; x > 0\})$

$$h : \mathbb{R}^+ \ni x \mapsto \log x \in \mathbb{R}$$

ist ein Isomorphismus (der sog. Rechenschieberisomorphismus)

Definition 15

Eine Abbildung

$$h : S \to \tilde{S}$$

heißt ein Isomorphismus von A nach $\tilde{A}$, falls

- h bijektiv ist und
- h mit den in Φ und $\tilde{\Phi}$ einander entsprechenden Operator vertauschbar ist (kommutatives Diagramm):

$$\begin{tikzcd} S^m \arrow[r, "\circ"] \arrow[d, "(h,\ldots,h)"'] & S \arrow[d, "h"] \\ \tilde{S}^m \arrow[r, "\tilde{\circ}"] & \tilde{S} \end{tikzcd}$$

Sei log der Zweierlogarithmus.

Welt 1 <R+, *>: 8 * 4 = 32

h↓ h↓ h↓

Welt 2 <R, +>: 3 + 2 = 5

Weitere Literatur und Übungen

Aufgaben zu Algorithmen und Datenstrukturen siehe auch
Vorlesung Grundlagen Algorithmen und Datenstrukturen (GAD),
SS 2008, Prof. Christian Scheideler
=> Uebung => auch alte Prüfungen mit Lösungen!

a) $o(n \log n) \subset O(n)$. ☐ Richtig ☒ Falsch

b) $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$. ☒ Richtig ☐ Falsch

c) $f'(n) \in O(g'(n)) \Leftrightarrow f(n) \in O(g(n))$ ☐ Richtig ☒ Falsch

d) Falls $T(1) = 1$ und $T(n) = 4 \cdot T(n/2) + 2n$ für alle $n > 1$, dann ist $T(n) \in O(n^2)$.
☒ Richtig ☐ Falsch

e) Jeder zusammenhängende ungerichtete Graph kann durch eine Tiefensuche vollständig exploriert werden.
☒ Richtig ☐ Falsch

f) Bei der amortisierten Analyse wird die durchschnittliche Laufzeit einer Operation im Worst-Case berechnet.
☒ Richtig ☐ Falsch

g) Insertion Sort hat eine Laufzeit von $O(n \log n)$.
☐ Richtig ☒ Falsch

Aufgabe 3 [10 Punkte] **Suchbäume**

a) [2 Punkte] Geben Sie *irgendeinen* binären Suchbaum für die folgende Elementmenge an (nur Endergebnis, aber mit ∞-Element und Liste unten):

$$\{16, 15, 14, 13, 30, 29, 18, 17, 23, 5, 19\}$$

b) [4 Punkte] Geben Sie für einen anfangs leeren Binärbaum die binären Suchbäume (wie in der Vorlesung behandelt mit ∞-Element) aus, die sich aus folgenden Operationen ergeben (Bild von Baum nach *jeder* einzelnen Operation zeichnen!):

$$insert(2), insert(1), insert(4), insert(7), insert(5),$$

$$insert(8), insert(3), remove(4), insert(6), remove(2).$$

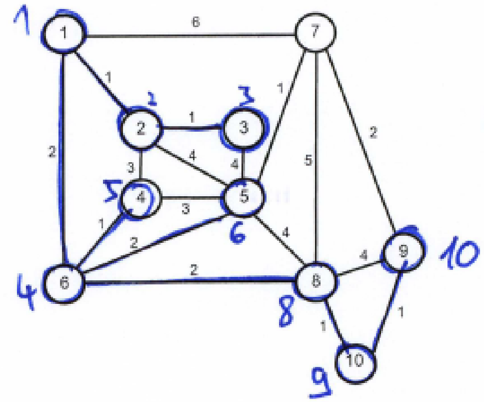
c) [4 Punkte] Geben Sie für einen anfangs leeren $(2,4)$-Baum die $(2,4)$-Bäume für die folgenden Operationen (wie in der Vorlesung behandelt mit ∞-Element) aus (nach *jeder* Operation Baum angeben!).

$$insert(3), insert(9), insert(23), insert(5), insert(4), insert(1),$$

$$insert(7), insert(6), insert(8), remove(4), remove(3), remove(1).$$

Aufgabe 5 [8 Punkte] **Kürzeste Wege und minimale Spannbäume**

Betrachten Sie den folgenden ungerichteten Graphen G.

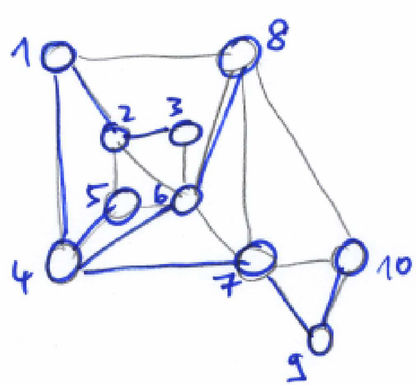


weight =12

Abbildung 5: Graph G.

a) [4 Punkte] Berechnen Sie mithilfe des Jarnik-Prim Algorithmus den minimalen Spannbaum des Graphen G ausgehend vom Knoten 1. Markieren Sie dazu durch Zahlen an den Knoten (direkt oben im Bild!), in welcher Reihenfolge die Knoten vom Jarnik-Prim Algorithmus aus der Priority Queue entnommen werden, und zeichnen Sie den resultierenden minimalen Spannbaum.

b) [4 Punkte] Berechnen Sie mittels Dijktras Algorithmus die kürzesten Wege vom Knoten 1 zu jedem anderen Knoten im Graphen G. Protokollieren Sie den Ablauf ausführlich (z.B. mittels Tabelle, oder markieren Sie durch Zahlen an den Knoten, in welcher Reihenfolge die Knoten vom Dijkstra Algorithmus aus der Priority Queue entnommen werden). Zeichnen Sie auch den resultierenden Baum der kürzesten Wege.



Weitere Literatur und Übungen

Logik, Sprachen, Algorithmen...:

http://www.ita.inf.ethz.ch/theoInf08/index.html

(mit Lösungen)

http://www.crypto.ethz.ch/teaching/lectures/DM08/

(mit Lösungen)

http://dcg.ethz.ch/lectures/ws0506/eventsystems/index.html

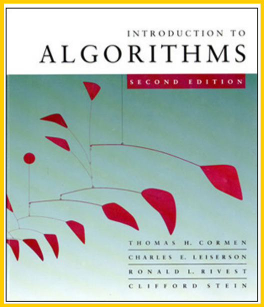
(mit Lösungen)

http://www.cse.yorku.ca/course_archive/2006-07/F/2001/

(gut! Und mit Lösungen)

MIT Open Class zu Algorithmen (mit Videos):

http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/index.htm



Weitere Literatur und Übungen

Computer Science Cheat Sheet

http://www.tug.org/texshowcase/cheat.pdf

Das meiste aber hier weniger relevant...!

Theoretical Computer Science Cheat Sheet

Definitions

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0$ $\forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a \rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n}{4}$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\Big[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\big((i+1)^{m+1} - i^{m+1} - (m$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c},$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2},$$

Harmonic series:

Heute

Viele, aber nicht alle Fragen.
(Auswahl von einfachen und fortgeschrittenen...)

Aber viele Slides..

Asymptotische Notation

Intuitiv: Zwei Funktionen $f(n)$ und $g(n)$ haben dasselbe Wachstumsverhalten, falls es Konstanten c und d gibt mit $c<f(n)/g(n)<d$ und $c<g(n)/f(n)<d$ für alle genügend große n.

Beispiel: n^2, $5n^2-7n$ und $n^2/10+100n$ haben dasselbe Wachstumsverhalten, da z.B.

$1/5<(5n^2-7n)/n^2<5$ und $1/5<n^2/(5n^2-7n)<5$

für alle $n\geq 2$.

Asymptotische Notation

Warum reichen genügend große n?

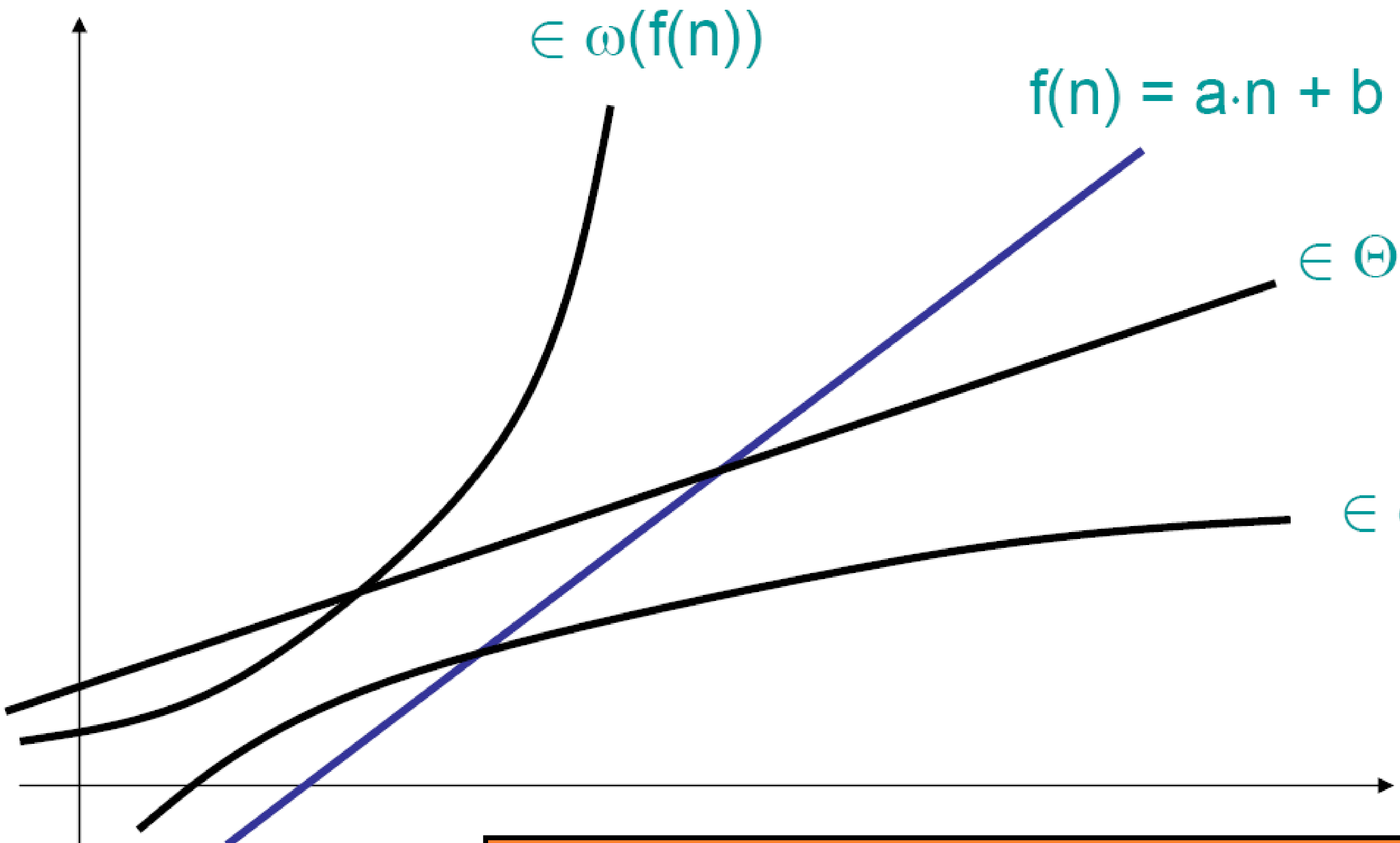
Ziel: Verfahren, die auch für große Instanzen noch effizient sind (d.h. sie skalieren gut).

Folgende Mengen formalisieren asymptotisches Verhalten:

- $O(f(n))=\{ g(n) \mid \exists\, c>0\ \exists\, n_0>0\ \forall\, n>n_0\colon g(n) \leq c\cdot f(n) \}$ **Asympt. höchstens**
- $\Omega(f(n))=\{ g(n) \mid \exists\, c>0\ \exists\, n_0>0\ \forall\, n>n_0\colon g(n) \geq c\cdot f(n) \}$ **Asympt. mindestens**
- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$ **Asympt. genau**
- $o(f(n))=\{ g(n) \mid \forall\, c>0\ \exists\, n_0>0\ \forall\, n>n_0\colon g(n) \leq c\cdot f(n) \}$ **Asympt. echt kleiner**
- $\omega(f(n))=\{ g(n) \mid \forall\, c>0\ \exists\, n_0>0\ \forall\, n>n_0\colon g(n) \geq c\cdot f(n) \}$ **Asympt. echt grösser**

Nur Funktionen f(n) (bzw. g(n)) mit $\exists\, N>0\ \forall\, n>N\colon f(n) > 0$!

Asymptotische Notation



Beispiele:

- n^2, $5n^2$-7n, $n^2/10 + 100n \in O(n^2)$
- $n \log n \in \Omega(n)$, $n^3 \in \Omega(n^2)$
- $\log n \in o(n)$, $n^3 \in o(2^n)$
- $n^5 \in \omega(n^3)$, $2^{2n} \in \omega(2^n)$

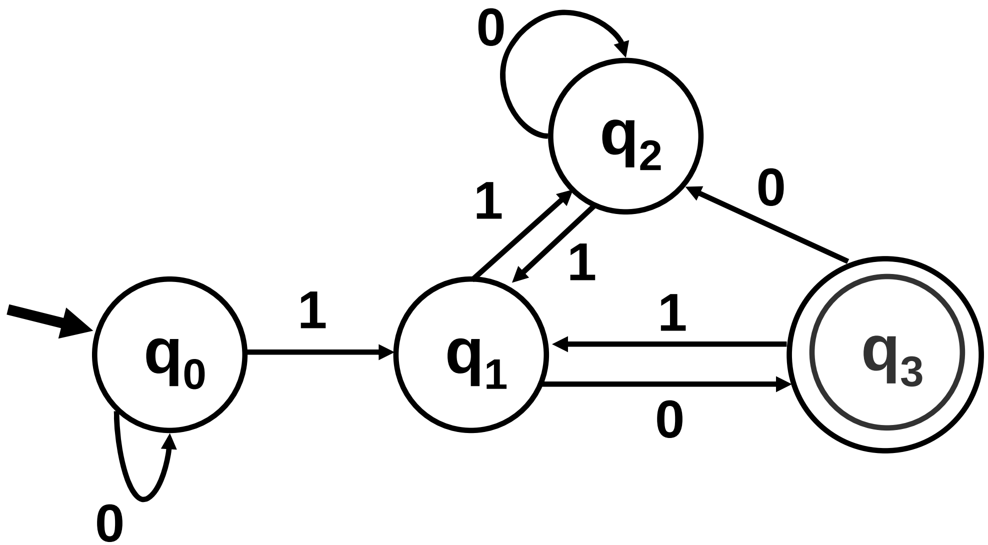
O-Notation

Aufgabe 1 [5 Punkte]

Kreuzen Sie pro Teilaufgabe höchstens ein Kästchen an. Für ein falsches Kreuz gibt es einen halben Minuspunkt, für ein richtiges einen halben Pluspunkt. Wenn Sie kein Kreuz setzen, bekommen Sie auch keine Punkte. Eine negative Gesamtpunktzahl dieser Aufgabe wird zu 0 aufgerundet. Maximieren Sie Ihre Punktzahl!

a) $o(n) \subset \mathcal{O}(n^2)$	☒ Richtig	☐ Falsch
b) $f(n) \in \Omega(g(n)) \Rightarrow f(n) \in \Theta(g(n))$	☐ Richtig	☒ Falsch
c) $\mathcal{O}(g(n)) \cap \omega(g(n)) = \emptyset$	☒ Richtig	☐ Falsch
d) Für jede positive Funktion $f(n)$ gilt: $g(n) + f(n) \in \mathcal{O}(g(n))$	☐ Richtig	☒ Falsch

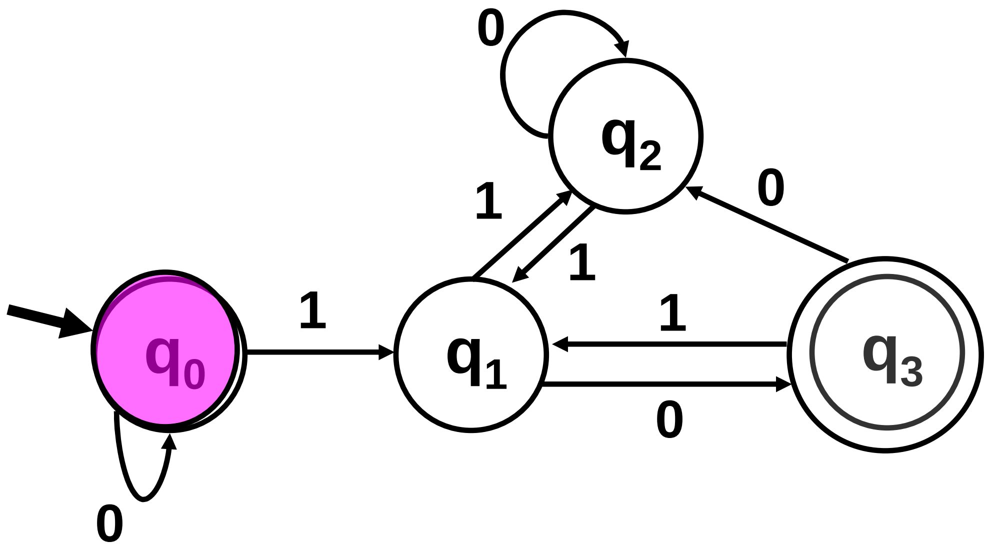
Pumping Lemma: Idee



Beispiel: Wort w = 11010 $\in$ L

Weg:

Pumping Lemma: Idee

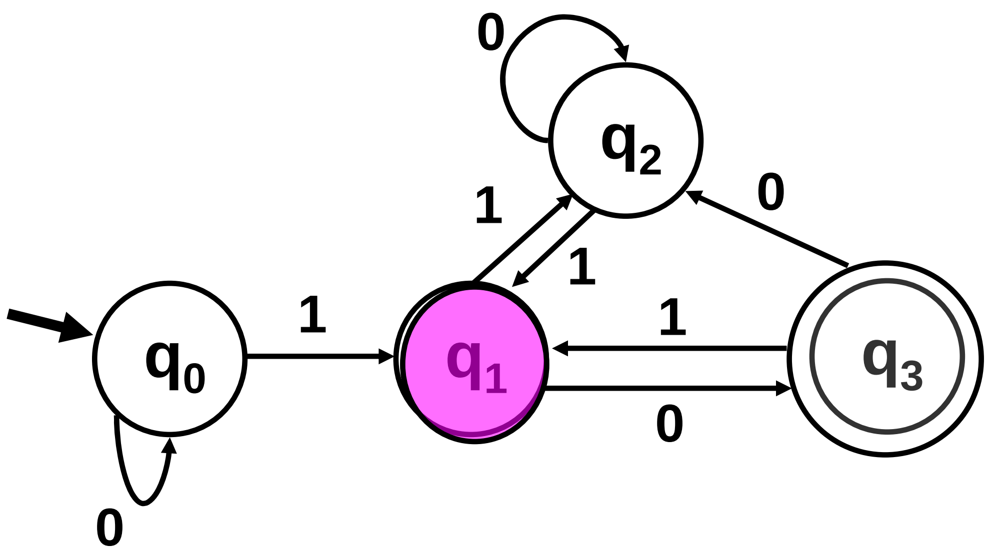


Beispiel: Wort w = 11010 $\in$ L

Weg: $\mathbf{q^{(0)}}$

$\mathbf{q_0}$

Pumping Lemma: Idee

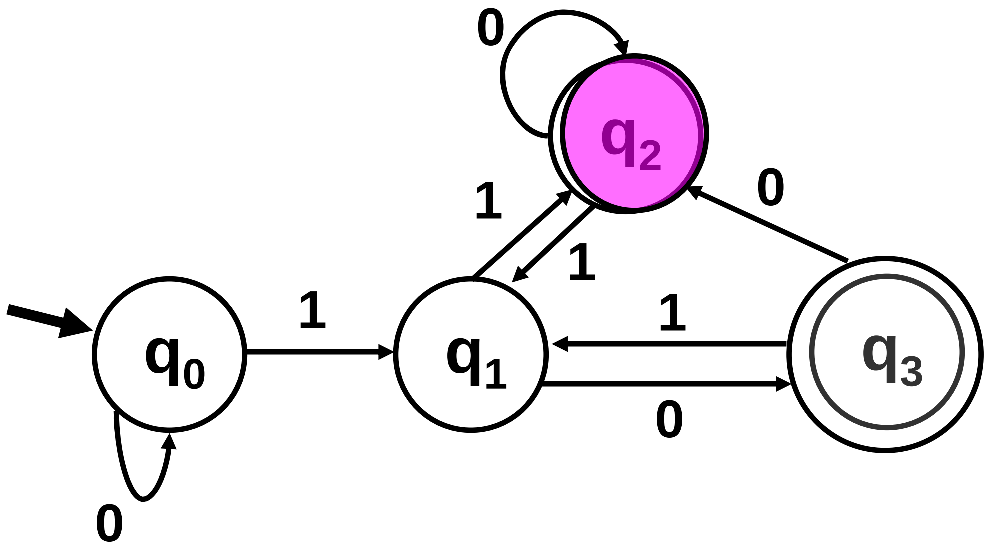


Beispiel: Wort w = **1**1010 $\in$ L

Weg: $q^{(0)}$ $\mathbf{q^{(1)}}$

q_0 $\mathbf{q_1}$

Pumping Lemma: Idee

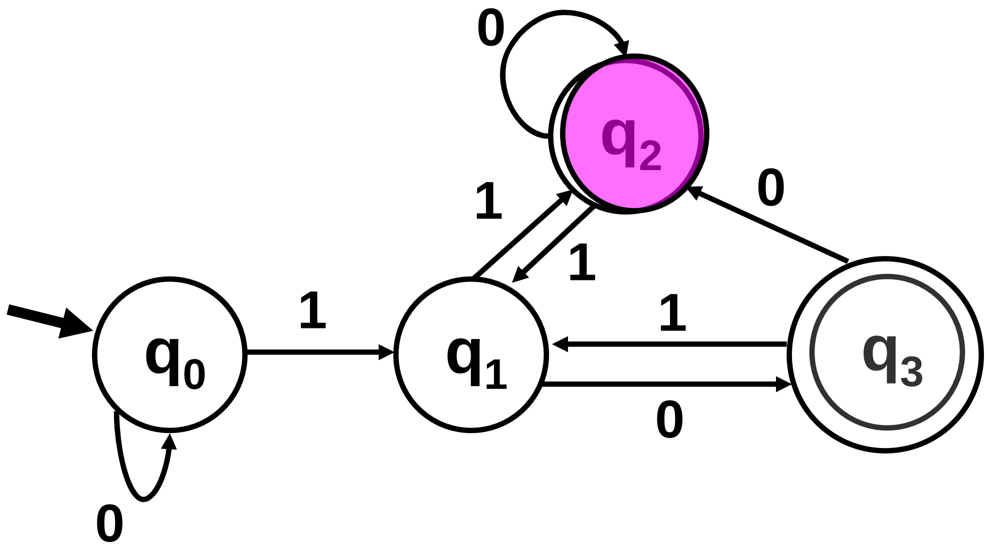


Beispiel: Wort w = 1**1**010 $\in$ L

Weg: $q^{(0)}\ q^{(1)}\ \mathbf{q^{(2)}}$

$q_0\ \ q_1\ \ \mathbf{q_2}$

Pumping Lemma: Idee

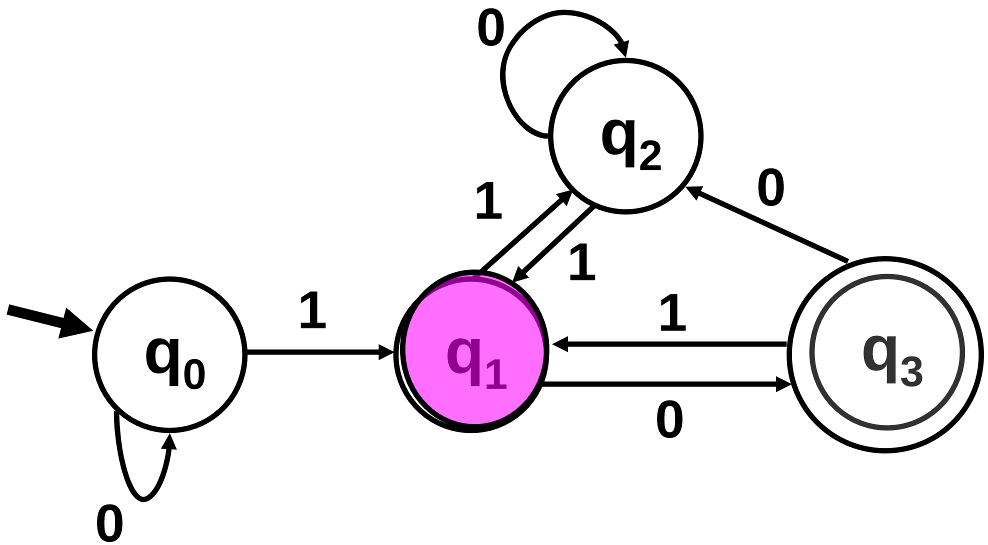


Beispiel: Wort w = 11**0**10 $\in$ L

Weg: $q^{(0)}$ $q^{(1)}$ $q^{(2)}$ $\mathbf{q^{(3)}}$

q_0 q_1 q_2 $\mathbf{q_2}$

Pumping Lemma: Idee

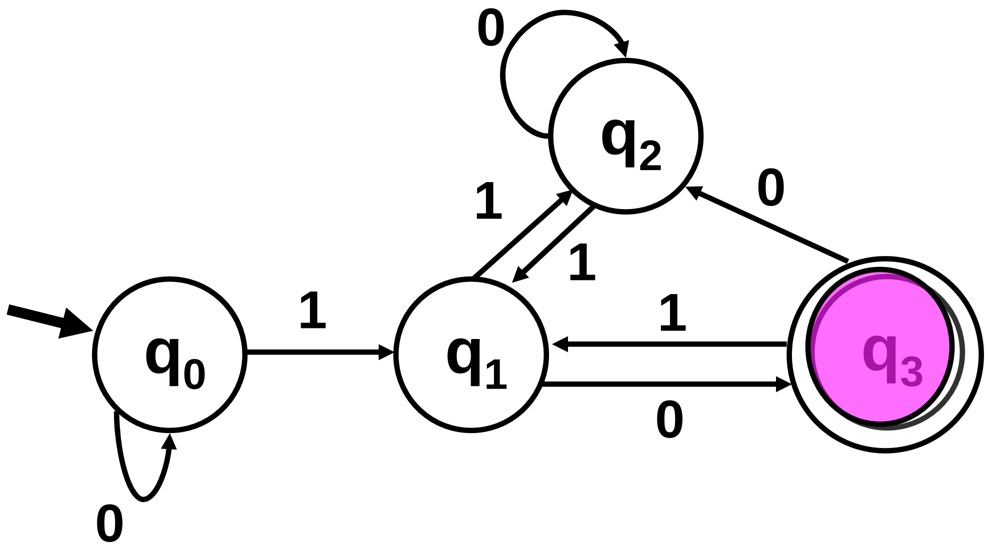


Beispiel: Wort w = 110**1**0 $\in$ L

Weg: $q^{(0)}$ $q^{(1)}$ $q^{(2)}$ $q^{(3)}$ $\mathbf{q^{(4)}}$

q_0 q_1 q_2 q_2 $\mathbf{q_1}$

Pumping Lemma: Idee

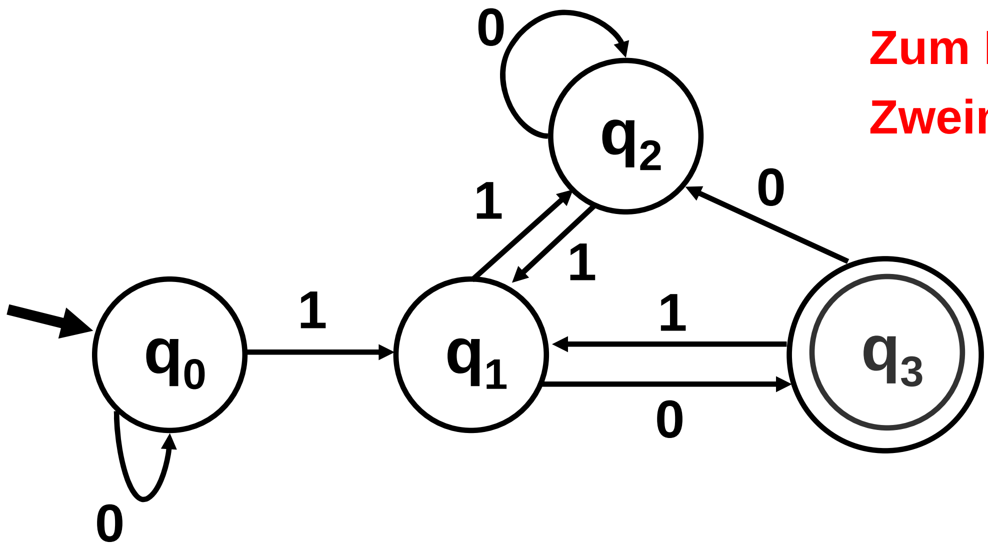


Beispiel: Wort w = 1101**0** $\in$ L

Weg: $q^{(0)}\ q^{(1)}\ q^{(2)}\ q^{(3)}\ q^{(4)}\ \mathbf{q^{(5)}}$

$q_0\ \ q_1\ \ q_2\ \ q_2\ \ q_1\ \ \mathbf{q_3}$

Pumping Lemma: Idee



Zum Beispiel hier:

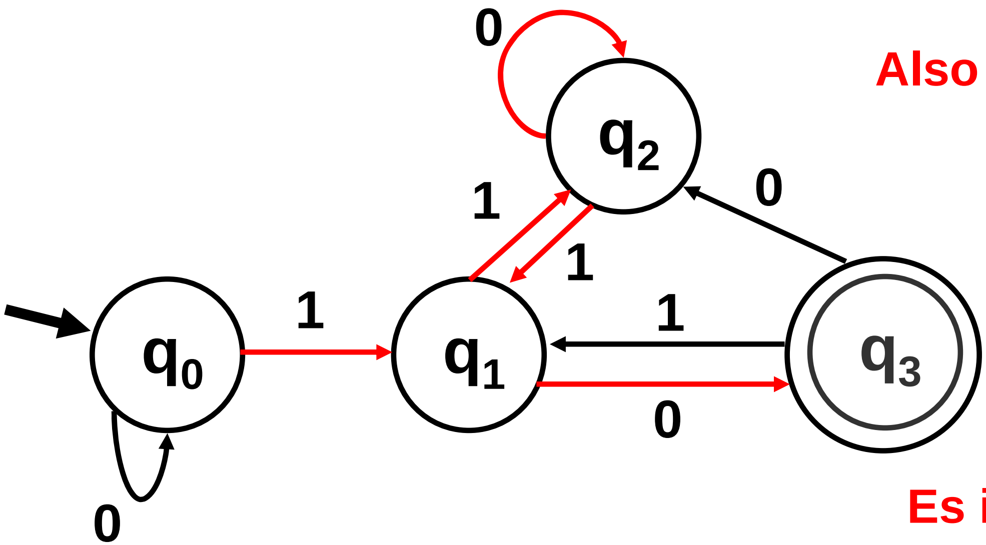
Zweimal im gleichen Zustand!

Beispiel: Wort w = 11010 $\in$ L

Weg: $q^{(0)}$ $q^{(1)}$ $q^{(2)}$ $q^{(3)}$ $q^{(4)}$ $q^{(5)}$

q_0 $\mathbf{q_1}$ q_2 q_2 $\mathbf{q_1}$ q_3

Pumping Lemma: Idee



Also wäre das eine Zerlegung:

$$w = \underbrace{1}_{x}\underbrace{101}_{y}\underbrace{0}_{z}$$

Es ist also auch in der Sprache:

xz = 10

xyyz = 11011010

Etc.!

Beispiel: Wort w = 1**101**0 $\in$ L

Weg: $q^{(0)}\ q^{(1)}\ q^{(2)}\ q^{(3)}\ q^{(4)}\ q^{(5)}$

$q_0\ \ \mathbf{q_1}\ \ q_2\ \ q_2\ \ \mathbf{q_1}\ \ q_3$

Keine Prüfungsaufgabe!

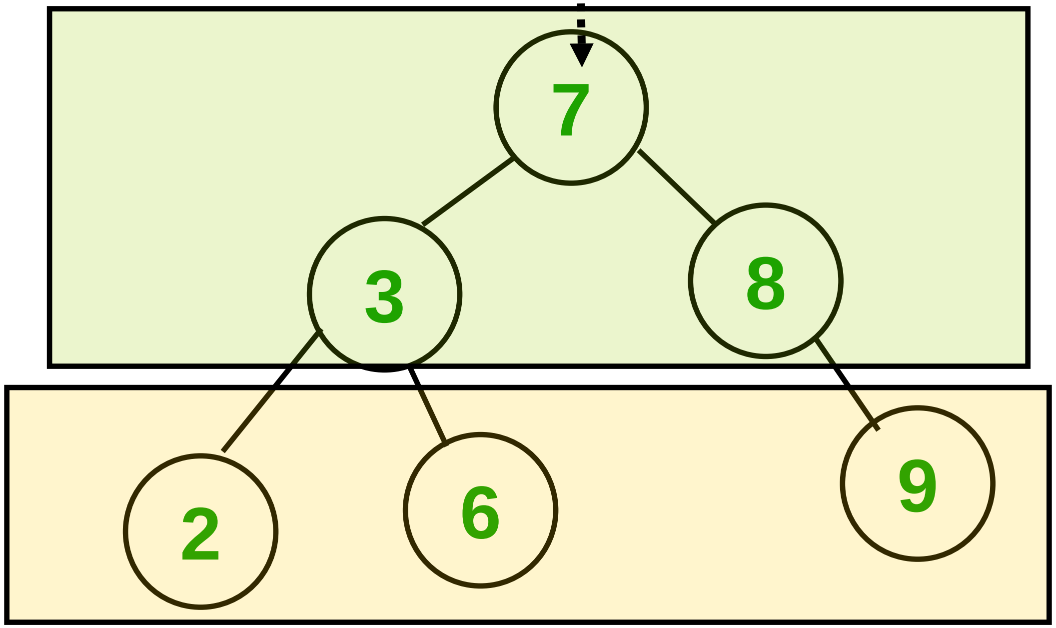
Zeige dass $0^i 1^j$ mit $i>j$ nicht regulär!

Idee: Wort $0^{n+1}1^n$ führt zum Widerspruch!

Bäume

Wurzel

(* es gibt auch Bäume ohne Wurzel natürlich)

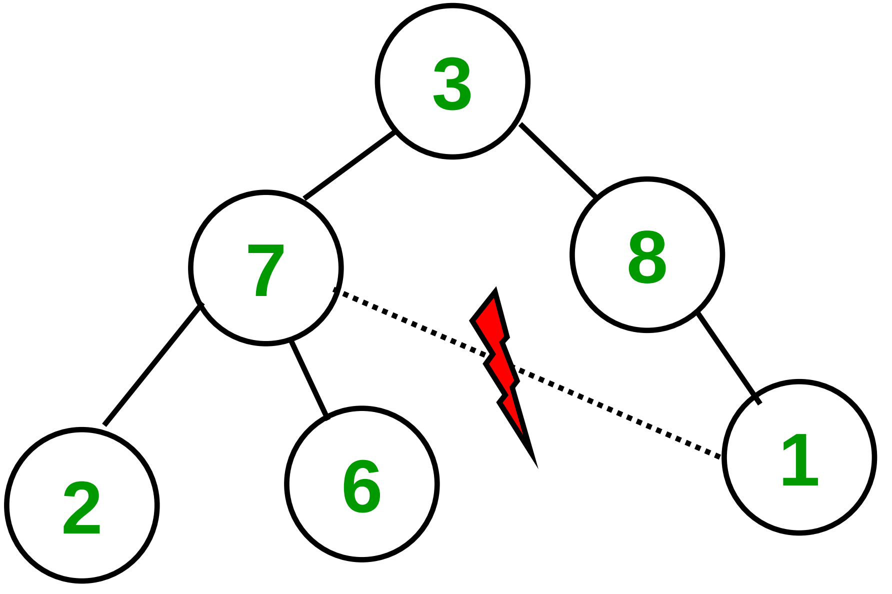


Innere Knoten

Blätter

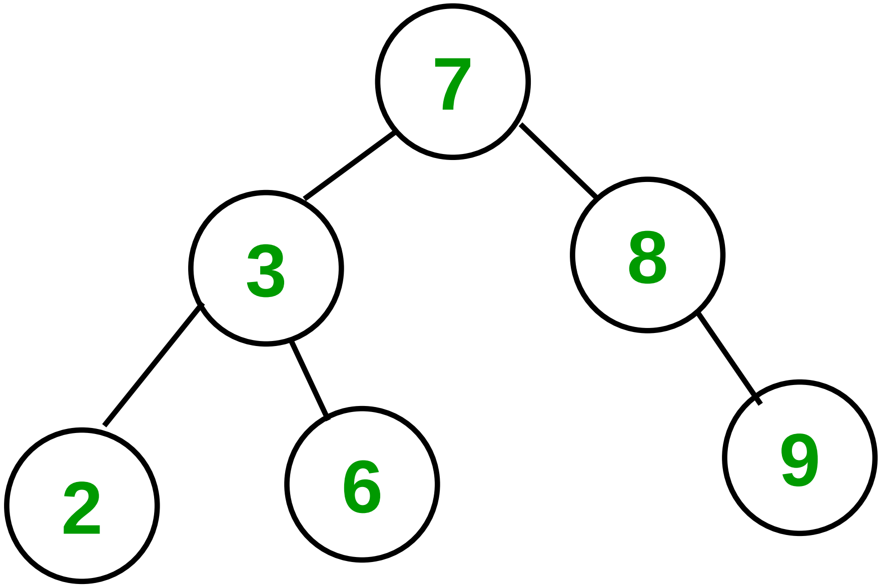
Baum = Graph ohne Zyklen

(ev. mit Wurzel, ev. gerichtet, etc.)



Kein Suchbaum!

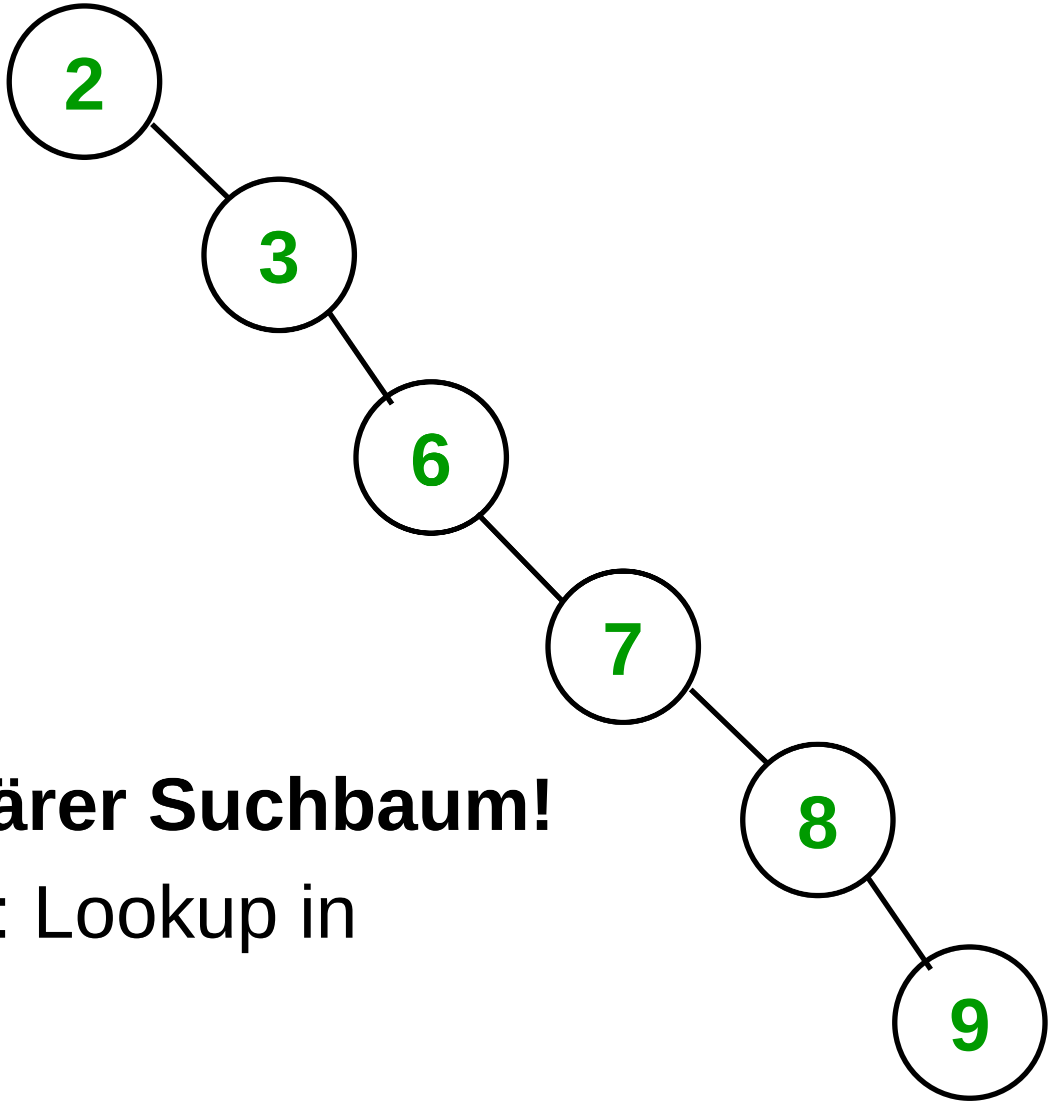
(Nicht sortiert!)



Binärer *Such*baum!

(„balanciert“: Lookup in logarithmischer Zeit)

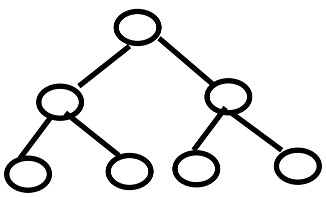
Bäume



Auch ein binärer Suchbaum!

(„degeneriert“: Lookup in linearer Zeit)

Baum mit n Knoten:
Wie hoch ist Binärbaum mindestens?



Tiefe 0 => max 1 Knote

Tiefe 1 => max 3 Knoten

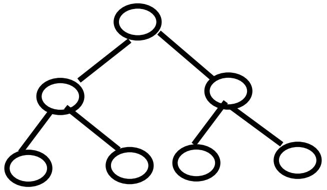
Tiefe 2 => max 7 Knoten

Höhe h hat maximal

$n \leq 1+2+4+...+2^h = 2^{h+1}-1$ Knoten.

Also $h \approx \log n$.

Perfekt balancierter Binärbaum mit n Blättern. Wie viele innere Knoten?



Tiefe 0 => max 1 Knote

Tiefe 2 => max 7 Knoten

Tiefe 1 => max 3 Knoten

Antwort: n-1

Grund: Induktion. In Tiefe i kommen 2^i innere Knoten *dazu* und *total* hat's $2^{\mathbf{i+1}}$ Blätter.

CYK Algorithmus

Beispiel [Bearbeiten]

Die Fragestellung lautet, ob sich das Wort w, $w = bbabaa$ durch die Grammatik $G = (\{S,A,B,C\},\{a,b\},P,S)$ erzeugen lässt. Die Produktionen P der Grammatik seien:

$$S \rightarrow AB \mid BC$$
$$A \rightarrow BA \mid a$$
$$B \rightarrow CC \mid b$$
$$C \rightarrow AB \mid a$$

Den Algorithmus kann man mittels einer Tabelle durchführen. Dabei ist in der i-ten Zeile und j-ten Spalte $V_{i,j}$ gespeichert, also die Menge der Nichtterminalsymbole aus denen sich das Teilwort $w_i \ldots w_j$ ableiten lässt.

$V_{i,j}$	1	2	3	4	5	6
1	{B}	{}	{A}	{S,C}	{B}	{A,S}
2		{B}	{S,A}	{S,C}	{B}	{A,S}
3			{A,C}	{S,C}	{B}	{S,A}
4				{B}	{A,S}	{}
5					{A,C}	{B}
6						{A,C}

...

2.

1.

Für i = 1 ... n
  Für jede Produktion $(l \rightarrow r) \in P$
    Falls r = w_i
      Setze $V_{i,i} := V_{i,i} \cup \{l\}$
Für j = 2 ... n
  Für i = j-1 ... 1
    Für k = i ... j - 1
      Für jede Produktion $(l \rightarrow BC) \in P$
        Falls $B \in V_{i,k}$ und $C \in V_{k+1,j}$
          Setze $V_{i,j} := V_{i,j} \cup \{l\}$
Falls $S \in V_{1,n}$, stoppe und gib **w wird von G erzeugt** aus
Stoppe und gib **w wird nicht von G erzeugt** aus

Da $S \in V_{1,6}$, lässt sich das gegebene Wort w = bbabaa unter der Grammatik G aus S ableiten. Also ist w ein Wort der Sprache $L(G)$.

CYK Algorithmus

Beispiel [Bearbeiten]

Die Fragestellung lautet, ob sich das Wort w, $w = bbabaa$ durch die Grammatik $G = (\{S,A,B,C\},\{a,b\},P,S)$ erzeugen lässt. Die Produktionen P der Grammatik seien:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Den Algorithmus kann man mittels einer Tabelle durchführen. Dabei ist in der i-ten Zeile und j-ten Spalte $V_{i,j}$ gespeichert, also die Menge der Nichtterminalsymbole aus denen sich das Teilwort $w_i \ldots w_j$ ableiten lässt.

$V_{i,j}$	1	2	3	4	5	6
1	{B}	{}	{A}	{S,C}	{B}	{A,S}
2		{B}	{S,A}	{S,C}	{B}	{A,S}
3			{A,C}	{S,C}	{B}	{S,A}
4				{B}	{A,S}	{}
5					{A,C}	{B}
6						{A,C}

Wie „a“?

Da $S \in V_{1,6}$, lässt sich das gegebene Wort w = bbabaa unter der Grammatik G aus S ableiten. Also ist w ein Wort der Sprache $L(G)$.

CYK Algorithmus

Beispiel [Bearbeiten]

Die Fragestellung lautet, ob sich das Wort w, $w = bbabaa$ durch die Grammatik $G = (\{S,A,B,C\},\{a,b\},P,S)$ erzeugen lässt. Die Produktionen P der Grammatik seien:

$$
\begin{aligned}
S &\rightarrow AB \mid BC \\
A &\rightarrow BA \mid a \\
B &\rightarrow CC \mid b \\
C &\rightarrow AB \mid a
\end{aligned}
$$

Den Algorithmus kann man mittels einer Tabelle durchführen. Dabei ist in der i-ten Zeile und j-ten Spalte $V_{i,j}$ gespeichert, also die Menge der Nichtterminalsymbole aus denen sich das Teilwort $w_i \ldots w_j$ ableiten lässt.

$V_{i,j}$	1	2	3	4	5	6
1	{B}	{}	{A}	{S,C}	{B}	{A,S}
2		{B}	{S,A}	{S,C}	{B}	{A,S}
3			{A,C}	{S,C}	{B}	{S,A}
4				{B}	{A,S}	{}
5					{A,C}	{B}
6						{A,C}

Wie „aa“? Durch
AA, AC, CA, CC,
also durch „B->CC“
Konkret: B ist die einzige Variable mit
X => $V_{5,5}$ $V_{6,6}$
für ein Element von
$V_{5,5}$ resp. $V_{6,6}$

Da $S \in V_{1,6}$, lässt sich das gegebene Wort w = bbabaa unter der Grammatik G aus S ableiten. Also ist w ein Wort der Sprache $L(G)$.

CYK Algorithmus

Beispiel [Bearbeiten]

Die Fragestellung lautet, ob sich das Wort w, $w = bbabaa$ durch die Grammatik $G = (\{S,A,B,C\},\{a,b\},P,S)$ erzeugen lässt. Die Produktionen P der Grammatik seien:

$$S \rightarrow AB \mid BC$$
$$A \rightarrow BA \mid a$$
$$B \rightarrow CC \mid b$$
$$C \rightarrow AB \mid a$$

Den Algorithmus kann man mittels einer Tabelle durchführen. Dabei ist in der i-ten Zeile und j-ten Spalte $V_{i,j}$ gespeichert, also die Menge der Nichtterminalsymbole aus denen sich das Teilwort $w_i \ldots w_j$ ableiten lässt.

$V_{i,j}$	1	2	3	4	5	6
1	{B}	{}	{A}	{S,C}	{B}	{A,S}
2		{B}	{S,A}	{S,C}	{B}	{A,S}
3			{A,C}	{S,C}	{B}	{S,A}
4				{B}	{A,S}	{}
5					{A,C}	{B}
6						{A,C}

Wie „babaa" ($V_{2,6}$)? Möglichkeiten:

$X \Rightarrow V_{2,2}\ V_{3,6}$ Geht! A => BA

$X \Rightarrow V_{2,3}\ V_{4,6}$ Geht nicht!

$X \Rightarrow V_{2,4}\ V_{5,6}$ Geht nicht!

$X \Rightarrow V_{2,5}\ V_{6,6}$ Geht! S => BC

Da $S \in V_{1,6}$, lässt sich das gegebene Wort w = bbabaa unter der Grammatik G aus S ableiten. Also ist w ein Wort der Sprache $L(G)$.

Eine alte Prüfungsaufgabe

Aufgabe 3 (8 Punkte)

Gegeben sei die Sprache $L \subseteq \{a, b, c\}^*$ mit

$$L = \{a^i b^j c^k \,;\, i, j, k \in \mathbb{N}_0,\ j < i \text{ und } j < k\}\,.$$

1. Stellen Sie L dar als Durchschnitt kontextfreier Sprachen L_1 und L_2. Zeigen Sie die Kontextfreiheit für die von Ihnen gewählten Sprachen L_1 und L_2.

Durchschnitt zweier Sprachen?

$$L_1 = \{a^i b^j c^k \,;\, i, j, k \in \mathbb{N}_0, j < i\} \quad \text{und} \quad L_2 = \{a^i b^j c^k \,;\, i, j, k \in \mathbb{N}_0, j < k\}$$

Alle mit weniger b‘s als a‘s UND weniger b‘s als c‘s!

Eine alte Prüfungsaufgabe

Wieso kontextfrei?

$$L_1 = \{a^i b^j c^k \,;\, i,j,k \in \mathbb{N}_0, j < i\} \quad \text{und} \quad L_2 = \{a^i b^j c^k \,;\, i,j,k \in \mathbb{N}_0, j < k\}$$

$$\begin{aligned} S &\rightarrow XTY \,|\, XY \,|\, XT \,|\, X, \\ T &\rightarrow aTb \,|\, ab\,, \\ X &\rightarrow aX \mid a\,, \\ Y &\rightarrow cY \mid c\,. \end{aligned} \qquad \text{(1 P.)}$$

$$\begin{aligned} S &\rightarrow XTY \,|\, XY \,|\, TY \,|\, Y\,, \\ T &\rightarrow bTc \,|\, bc\,, \\ X &\rightarrow aX \mid a\,, \\ Y &\rightarrow cY \mid c\,. \end{aligned} \qquad \text{(1 P.)}$$

Beweis durch kontextfreie Grammatiken!
L_1: T erzeugt gleich viele a‘s wie b‘s, X erzeugt a‘s, Y am Schluss die c‘s. Wegen S Produktion mind. Ein a mehr als b‘s. L_2: analog.

Auch grad eine Repetition von Pumping allgemein...

Satz 98 (Ogdens Lemma)

Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so dass für jedes Wort $z \in L$ mit $|z| \geq n$ die folgende Aussage gilt: Werden in z mindestens n (beliebige) Buchstaben markiert, so lässt sich z zerlegen in

$$z = uvwxy,$$

so dass

1. *in vx mindestens ein Buchstabe und*
2. *in vwx höchstens n Buchstaben markiert sind und*
3. $(\forall i \in \mathbb{N}_0)[uv^i wx^i y \in L]$.

Zeigen Sie mit Hilfe des Lemmas von Ogden, dass die Sprache

$$L = \{a^n b^n c^i \,;\, i, n \in \mathbb{N}, i \neq n\}$$

nicht kontextfrei ist.

Zeigen Sie mit Hilfe des Lemmas von Ogden, dass die Sprache

$$L = \{a^n b^n c^i\ ;\ i, n \in \mathbb{N}, i \neq n\}$$

nicht kontextfrei ist.

Satz 98 (Ogdens Lemma)

Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so dass für jedes Wort $z \in L$ mit $|z| \geq n$ die folgende Aussage gilt: Werden in z mindestens n (beliebige) Buchstaben markiert, so lässt sich z zerlegen in

$$z = uvwxy,$$

so dass

1. *in vx mindestens ein Buchstabe und*
2. *in vwx höchstens n Buchstaben markiert sind und*
3. $(\forall i \in \mathbb{N}_0)[uv^iwx^iy \in L]$.

1. Wählen wir ein Wort aus der Sprache: **$z = a^n\ b^n\ c^{n+n!}$**, wobei n die Pumping Konstante sei

2. Das Wort ist genug lang, und wir **markieren alle a‘s** in z

3. Laut Lemma also: es gibt Zerlegung z = uvwxy, sodass in **vx** mindestens ein a, in **vwx** höchstens n a‘s, und **$uv^iwx^iy \in L$** für alle $i \geq 0$

Zeigen Sie mit Hilfe des Lemmas von Ogden, dass die Sprache

$$L = \{a^n b^n c^i \,;\, i, n \in \mathbb{N}, i \neq n\}$$

nicht kontextfrei ist.

Satz 98 (Ogdens Lemma)

Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so dass für jedes Wort $z \in L$ mit $|z| \geq n$ die folgende Aussage gilt: Werden in z mindestens n (beliebige) Buchstaben markiert, so lässt sich z zerlegen in

$$z = uvwxy,$$

so dass

1. *in vx mindestens ein Buchstabe und*
2. *in vwx höchstens n Buchstaben markiert sind und*
3. $(\forall i \in \mathbb{N}_0)[uv^i wx^i y \in L]$.

z = a^n b^n $c^{n+n!}$, markieren alle a‘s, z = uvwxy, vx mindestens ein a, in vwx höchstens n a‘s, und $uv^iwx^iy \in L$ für alle $i \geq 0$

Wir zeigen nun Widerspruch zu Lemma

1. Falls v oder x verschiedene Buchstaben enthalten, wird Reihenfolge nach Pumpen nicht mehr stimmen => Widerspruch, Wort nicht in Sprache

2. Falls v oder x „rein“ sind: Da vx mindestens ein a hat, muss entweder v oder x nur aus a‘s bestehen. (Beide gibt auch kein Wort in L!)

3. Es bleibt folgende Fallunterscheidung!

Wir haben total nur n a‘s, und
v besteht nur aus a‘s.

Jede Zahl {1,...,n} teilt
n! = n (n-1) (n-2) ... 1

1. Fall: $v \in a^+$ und $x \in b^+$.
Sei $p := |v|$. Wegen $1 \leq p \leq n$ ist p ein Teiler von $n!$. Sei q mit $pq = n!$. Nach Ogden ist $z' = uv^{q+1}wx^{q+1}y \in L$. Das Wort uwy enthält genau $(n-p)$-mal a. Daher enthält z' genau

einmal v weglassen!
(egal was w ist)

$$n - p + p(q+1) = n + n!$$

a's. Also ist in z' die Anzahl der a's gleich der Anzahl der c's, Widerspruch zu $z' \in L$.

Wir pumpen nur a‘s und b‘s, und haben
weiterhin n+n! viele c‘s => Wort nicht in Sprache

2. Fall: $(v \in a^+$ und $x \notin b^+)$ oder $v \notin a^+$.
Dann enthält uv^2wx^2y unterschiedlich viele a's und b's, Widerspruch zu $uv^2wx^2y \in L$.

Kontextfreie Grammatiken

Zeigen Sie, dass die Sprache der Palindrome (ohne ε) über {0,1} kontextfrei ist!

Palindrom?? Wort vorwärts gelesen = rückwärts gelesen.
Z.B.: „sugus“

Wie beweisen? NFA zeichnen, Grammatik angeben, ...
(„Abschlusseigenschaften“)

Lösung als Grammatik?

S => 0 ¦ 1
S => 0S0
S => 1S1

Eine alte Prüfungsaufgabe

Aufgabe 4 (12 Punkte)

Sei $G = (\{Z, A, B, X, Y\}, \{a, b\}, P, Z)$ mit den Produktionen

$$\begin{array}{rcl@{\qquad\qquad}rcl} Z & \to & A \mid ZB\,, & & & \\ A & \to & aX \mid a\,, & X & \to & BY\,, \\ B & \to & bB \mid b\,, & Y & \to & AZ\,. \end{array}$$

1. Konstruieren Sie eine Grammatik G_1 in Greibach-Normalform, die $L(G)$ erzeugt.

Greibach Normalform: Was zuerst machen? Chomsky Normalform!

Wozu Chomsky Normalform gut? Ausgangslage für viele Algos und Beweise, z.B. CYK Algo für Wortproblem, nutzlose Variablen finden, etc.

Wozu Greibach Normalform gut? NPDA konstruieren

Aufgabe 4 (12 Punkte)

Sei $G = (\{Z, A, B, X, Y\}, \{a, b\}, P, Z)$ mit den Produktionen

$$\begin{array}{rclcrcl} Z & \rightarrow & A \mid ZB\,, & & & & \\ A & \rightarrow & aX \mid a\,, & & X & \rightarrow & BY\,, \\ B & \rightarrow & bB \mid b\,, & & Y & \rightarrow & AZ\,. \end{array}$$

1. Konstruieren Sie eine Grammatik G_1 in Greibach-Normalform, die $L(G)$ erzeugt.

Also zuerst Chomsky:

Sei $G = (V, \Sigma, P, S)$ eine kontexfreie Grammatik.

Definition 94

Eine kontextfreie Grammatik G ist in Chomsky-Normalform, falls alle Produktionen eine der Formen

$$\begin{array}{ll} A \rightarrow a & A \in V, a \in \Sigma, \\ A \rightarrow BC & A, B, C \in V, \text{ oder} \\ S \rightarrow \epsilon & \end{array}$$

haben.

Ist im Prinzip schon fast Chomsky: ersetze Terminale durch Nichtterminale, ersetzte einzelnes Terminal durch die beiden Möglichkeiten.

Aufgabe 4 (12 Punkte)

Sei $G = (\{Z, A, B, X, Y\}, \{a, b\}, P, Z)$ mit den Produktionen

$$
\begin{array}{rclcrcl}
Z & \rightarrow & A \mid ZB\,, & & & & \\
A & \rightarrow & aX \mid a\,, & & X & \rightarrow & BY\,, \\
B & \rightarrow & bB \mid b\,, & & Y & \rightarrow & AZ\,.
\end{array}
$$

1. Konstruieren Sie eine Grammatik G_1 in Greibach-Normalform, die $L(G)$ erzeugt.

Also Greibach Normalform:

Definition 108

Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. G ist in Greibach-Normalform (benannt nach Sheila Greibach (UCLA)), falls jede Produktion $\neq S \rightarrow \epsilon$ von der Form

$$A \rightarrow a\alpha \text{ mit } a \in \Sigma,\ \alpha \in V^*$$

ist.

Aufgabe 4 (12 Punkte)

Sei $G = (\{Z, A, B, X, Y\}, \{a, b\}, P, Z)$ mit den Produktionen

$$\begin{aligned} Z &\rightarrow A \mid ZB\,, & & \\ A &\rightarrow aX \mid a\,, & X &\rightarrow BY\,, \\ B &\rightarrow bB \mid b\,, & Y &\rightarrow AZ\,. \end{aligned}$$

1. Konstruieren Sie eine Grammatik G_1 in Greibach-Normalform, di

Definition 108

Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. G ist in Greibach-Normalform (benannt nach Sheila Greibach (UCLA)), falls jede Produktion $\neq S \rightarrow \epsilon$ von der Form

$$A \rightarrow a\alpha \text{ mit } a \in \Sigma,\ \alpha \in V^*$$

ist.

Benutze:

Lemma 109

Sei $G = (V, \Sigma, P, S)$ kontextfrei, $(A \rightarrow \alpha_1 B \alpha_2) \in P$, und sei $B \rightarrow \beta_1|\beta_2|\ldots|\beta_r$ die Menge der B-Produktionen (also die Menge der Produktionen mit B auf der linken Seite). Ersetzt man $A \rightarrow \alpha_1 B \alpha_2$ durch $A \rightarrow \alpha_1\beta_1\alpha_2|\alpha_1\beta_2\alpha_2|\ldots|\alpha_1\beta_r\alpha_2$, so ändert sich die von der Grammatik erzeugte Sprache nicht.

Lemma 110

Sei $G = (V, \Sigma, P, S)$ kontextfrei, sei $A \rightarrow A\alpha_1|A\alpha_2|\ldots|A\alpha_r$ die Menge der linksrekursiven A-Produktionen (alle $\alpha_i \neq \epsilon$, die Produktion $A \rightarrow A$ kommt o.B.d.A. nicht vor), und seien $A \rightarrow \beta_1|\beta_2|\ldots|\beta_s$ die restlichen A-Produktionen (ebenfalls alle $\beta_i \neq \epsilon$).

Ersetzen wir alle A-Produktionen durch

$$\begin{aligned} A &\rightarrow \beta_1|\ldots|\beta_s|\beta_1 A'|\ldots|\beta_s A' \\ A' &\rightarrow \alpha_1|\ldots|\alpha_r|\alpha_1 A'|\ldots|\alpha_r A'\,, \end{aligned}$$

wobei A' ein neues Nichtterminal ist, so ändert sich die Sprache nicht, und die neue Grammatik enthält keine linksrekursive A-Produktion mehr.

Aufgabe 4 (12 Punkte)

Sei $G = (\{Z, A, B, X, Y\}, \{a, b\}, P, Z)$ mit den Produktionen

$$\begin{aligned} Z &\rightarrow A \mid ZB\,, & & \\ A &\rightarrow aX \mid a\,, & X &\rightarrow BY\,, \\ B &\rightarrow bB \mid b\,, & Y &\rightarrow AZ\,. \end{aligned}$$

1. Konstruieren Sie eine Grammatik G_1 in Greibach-Normalform, die $L(G)$ erzeugt.

Lemma 110

Sei $G = (V, \Sigma, P, S)$ kontextfrei, sei $A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_r$ die Menge der linksrekursiven A-Produktionen (alle $\alpha_i \neq \epsilon$, die Produktion $A \rightarrow A$ kommt o.B.d.A. nicht vor), und seien $A \rightarrow \beta_1 | \beta_2 | \ldots | \beta_s$ die restlichen A-Produktionen (ebenfalls alle $\beta_i \neq \epsilon$).
Ersetzen wir alle A-Produktionen durch

$$\begin{aligned} A &\rightarrow \beta_1 | \ldots | \beta_s | \beta_1 A' | \ldots | \beta_s A' \\ A' &\rightarrow \alpha_1 | \ldots | \alpha_r | \alpha_1 A' | \ldots | \alpha_r A'\,, \end{aligned}$$

wobei A' ein neues Nichtterminal ist, so ändert sich die Sprache nicht, und die neue Grammatik enthält keine linksrekursive A-Produktion mehr.

Reihenfolge der Variablen: $Z, X, Y, A, B.$

Welche Produktion ist linksrekursiv? Z -> ZB, also mit Lemma 110 ersetzen!

Ersetzung der linksrekursiven Produktion $Z \rightarrow ZB$:
Sei Z' eine neue Variable. Die neuen Z-Produktionen sind dann

$$Z \rightarrow A \mid AZ'\,, \qquad Z' \rightarrow B \mid BZ'\,.$$

Aufgabe 4 (12 Punkte)

Sei $G = (\{Z, A, B, X, Y\}, \{a, b\}, P, Z)$ mit den Produktionen

$$\begin{aligned} Z &\rightarrow A \mid ZB, & & \\ A &\rightarrow aX \mid a, & X &\rightarrow BY, \\ B &\rightarrow bB \mid b, & Y &\rightarrow AZ. \end{aligned}$$

1. Konstruieren Sie eine Grammatik G_1 in Greibach-Normalform, die $L(G)$ erzeugt.

$$Z \rightarrow A \mid AZ', \qquad Z' \rightarrow B \mid BZ'.$$

Lemma 109

Sei $G = (V, \Sigma, P, S)$ kontextfrei, $(A \rightarrow \alpha_1 B \alpha_2) \in P$, und sei $B \rightarrow \beta_1|\beta_2|\ldots|\beta_r$ die Menge der B-Produktionen (also die Menge der Produktionen mit B auf der linken Seite). Ersetzt man $A \rightarrow \alpha_1 B \alpha_2$ durch $A \rightarrow \alpha_1\beta_1\alpha_2|\alpha_1\beta_2\alpha_2|\ldots|\alpha_1\beta_r\alpha_2$, so ändert sich die von der Grammatik erzeugte Sprache nicht.

Setze nun A und B in anderen Produktionen direkt ein mit Lemma 109:

Ersetzung der übrigen Produktionen:

$$\begin{aligned} Y &\rightarrow aZ \mid aXZ, \\ X &\rightarrow bY \mid bBY, \\ Z' &\rightarrow bB \mid b \mid bBZ' \mid bZ', \\ Z &\rightarrow a \mid aX \mid aZ' \mid aXZ', \end{aligned}$$

In G_1 ist A nutzlos und kann samt A-Produktionen gestrichen werden.

CFG -> NPDA

LEMMA 2.13

If a language is context free, then some pushdown automaton recognizes it.

PROOF IDEA Let A be a CFL. From the definition we know that A has a CFG, G, generating it. We show how to convert G into an equivalent PDA, which we call P.

The PDA P that we now describe will work by accepting its input w, if G generates that input, by determining whether there is a derivation for w. Recall that a derivation is simply the sequence of substitutions made as a grammar generates a string. Each step of the derivation yields an ***intermediate string*** of variables and terminals. We design P to determine whether some series of substitutions using the rules of G can lead from the start variable to w.

One of the difficulties in testing whether there is a derivation for w is in figuring out which substitutions to make. The PDA's nondeterminism allows it to guess the sequence of correct substitutions. At each step of the derivation one of the rules for a particular variable is selected nondeterministically and used to substitute for that variable.

The PDA P begins by writing the start variable on its stack. It goes through a series of intermediate strings, making one substitution after another. Eventually it may arrive at a string that contains only terminal symbols, meaning that it has derived a string using the grammar. Then P accepts if this string is identical to the string it has received as input.

Implementing this strategy on a PDA requires one additional idea. We need to see how the PDA stores the intermediate strings as it goes from one to another. Simply using the stack for storing each intermediate string is tempting. However, that doesn't quite work because the PDA needs to find the variables in the intermediate string and make substitutions. The PDA can access only the top symbol on the stack and that may be a terminal symbol instead of a variable. The way around this problem is to keep only *part* of the intermediate string on the stack: the symbols starting with the first variable in the intermediate string. Any terminal symbols appearing before the first variable are matched immediately with symbols in the input string. The following figure shows the PDA P.

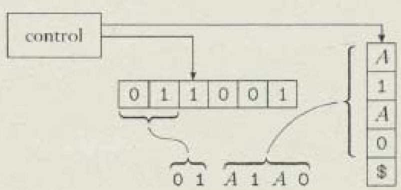


FIGURE 2.9
P representing the intermediate string $01A1A0$

CFG -> NPDA

The following is an informal description of P.

1. Place the marker symbol \$ and the start variable on the stack.
2. Repeat the following steps forever.
   a. If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
   c. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

CFG -> NPDA

EXAMPLE 2.14

We use the procedure developed in Lemma 2.13 to construct a PDA P_1 from the following CFG G.

$$S \to \mathrm{a}T\mathrm{b} \mid \mathrm{b}$$
$$T \to T\mathrm{a} \mid \varepsilon$$

The transition function is shown in the following diagram.

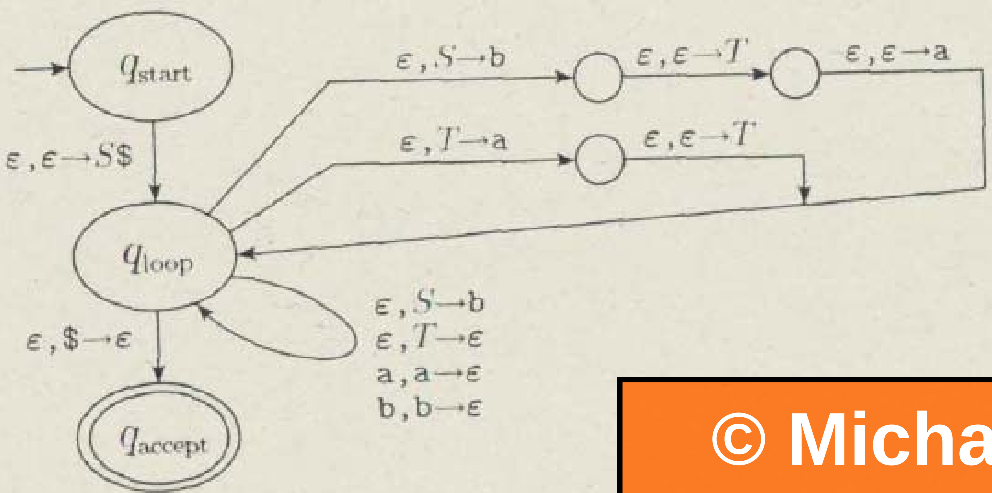


FIGURE 2.12
State diagram of P_1

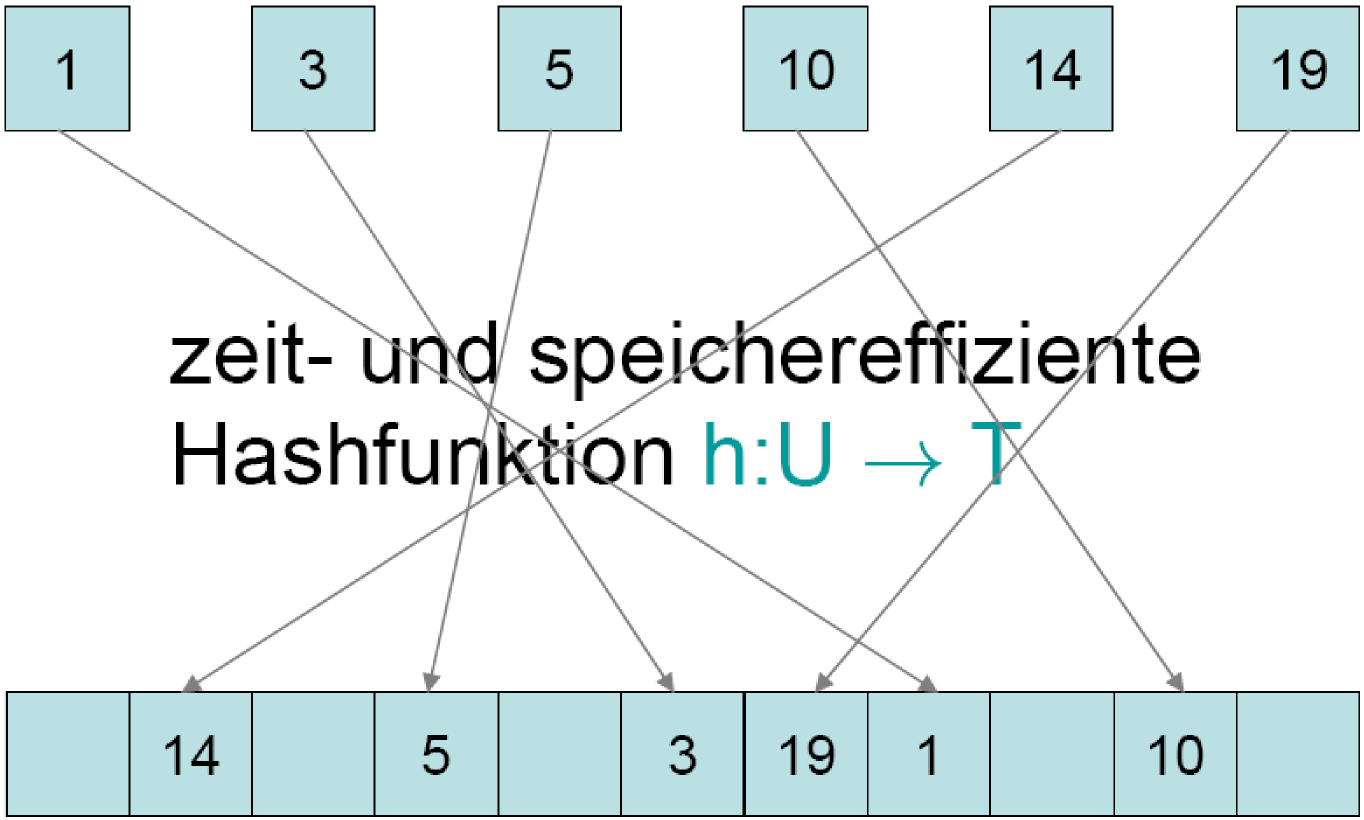
Wozu Hashing?

Alternative zu Suchbäumen! Kann Daten einfügen, löschen und suchen! Suche geht in konstanter Zeit (Binärbaum: logarithmische)

Nachteil von Hashing?

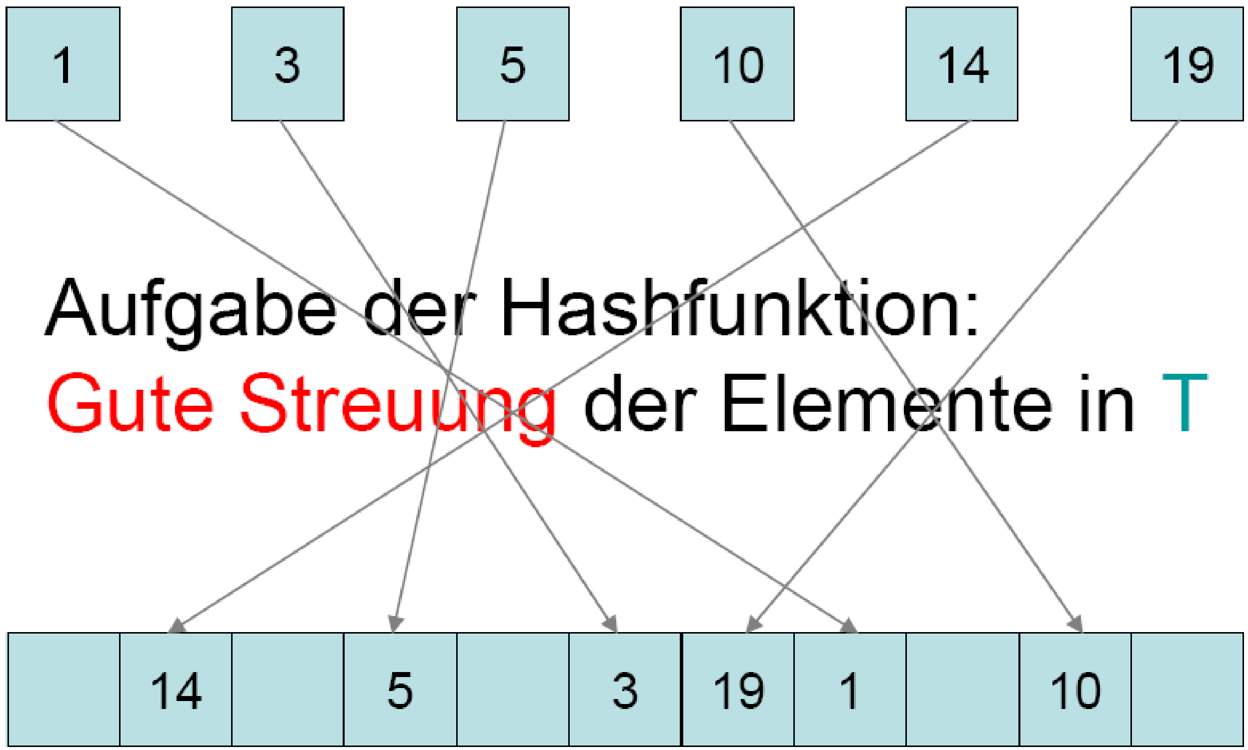
Suchstruktur geht kaputt! Z.B. im Binärbaum kann man besser *alle* Elemente zwischen x und y ausgeben als beim Hashing!

Hashing



Hashtabelle T

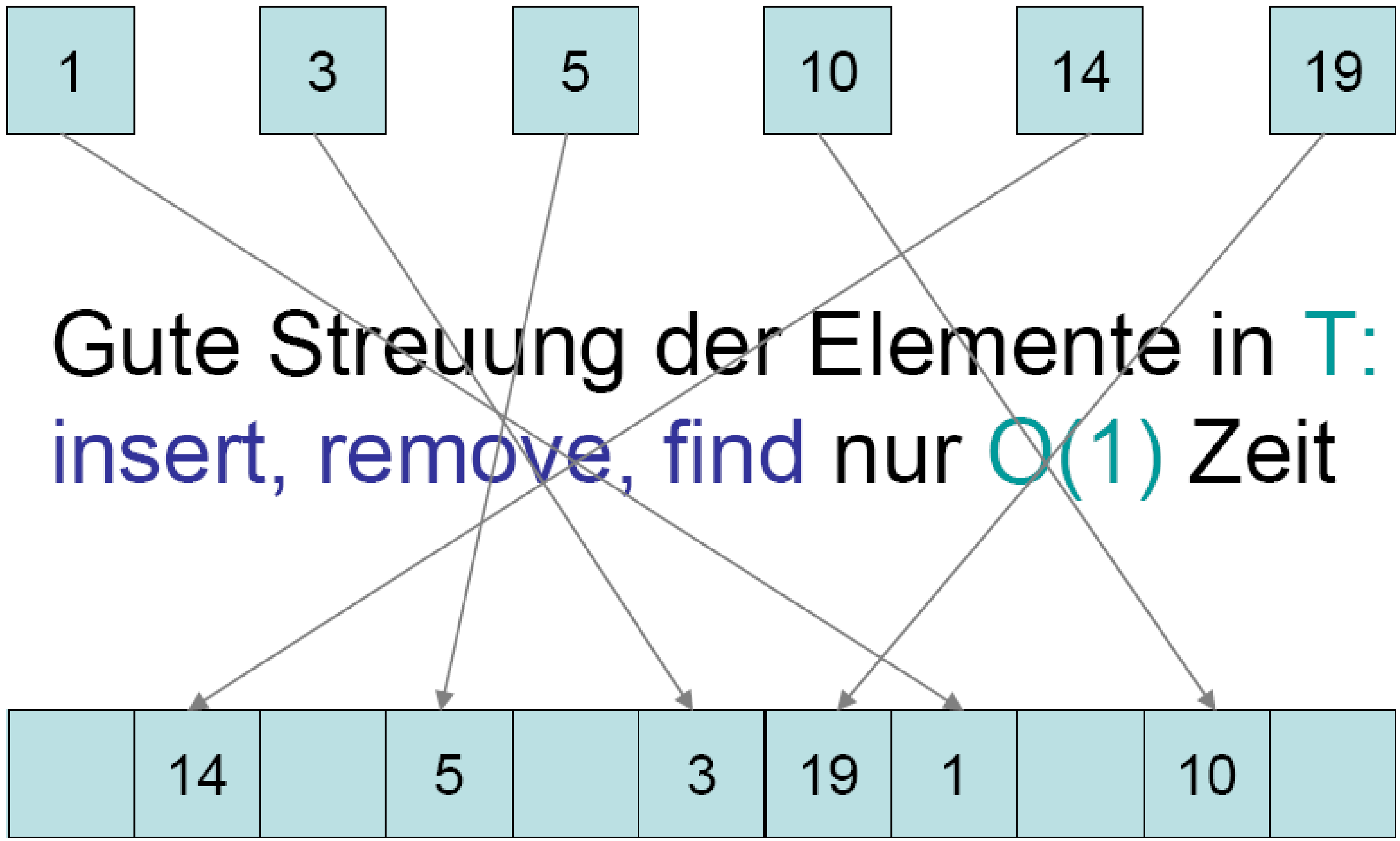
Hashing



Aufgabe der Hashfunktion:
Gute Streuung der Elemente in T

Hashtabelle T

Hashing

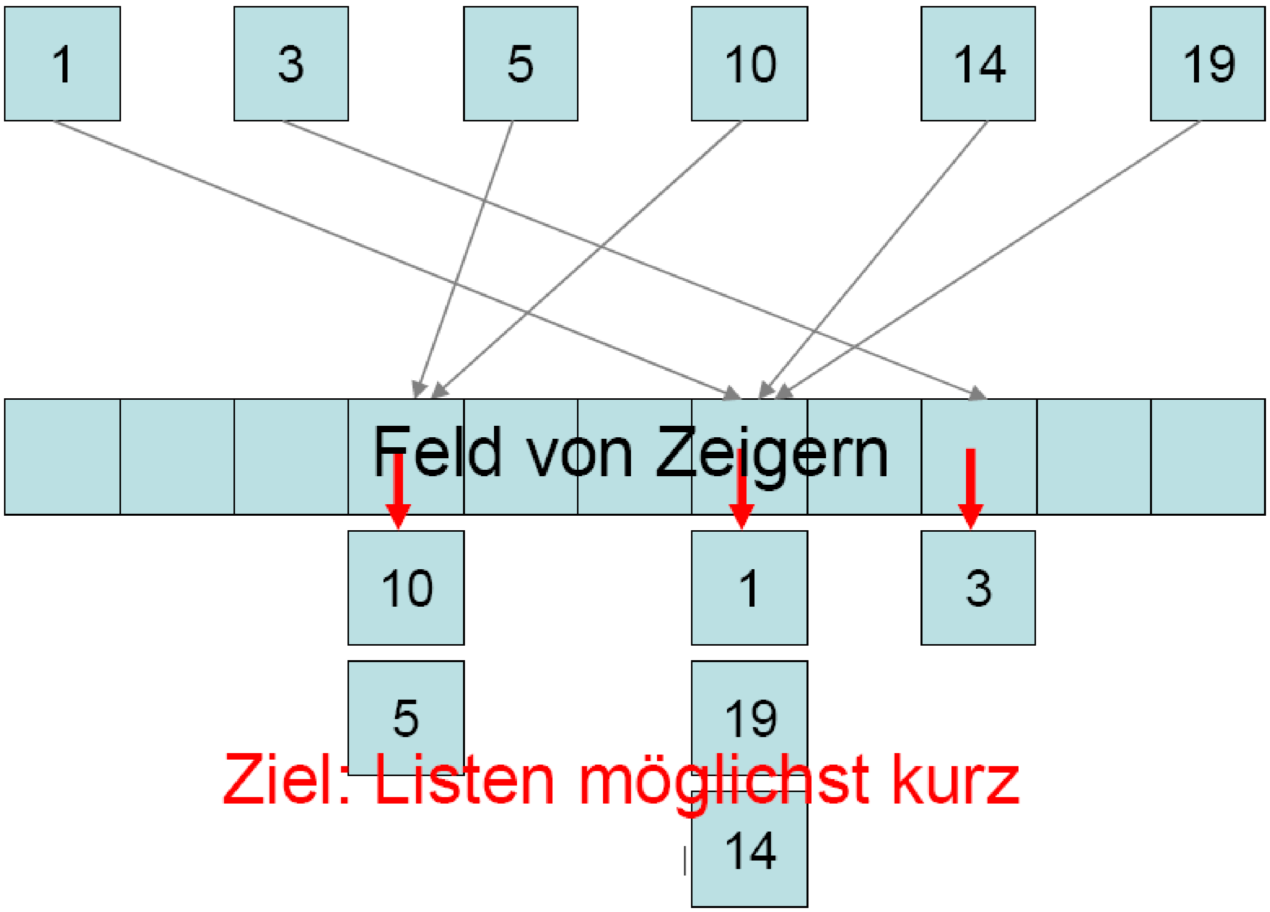


Gute Streuung der Elemente in T:
insert, remove, find nur O(1) Zeit

Hashtabelle T

Dynamisches Wörterbuch

Hashing with Chaining:

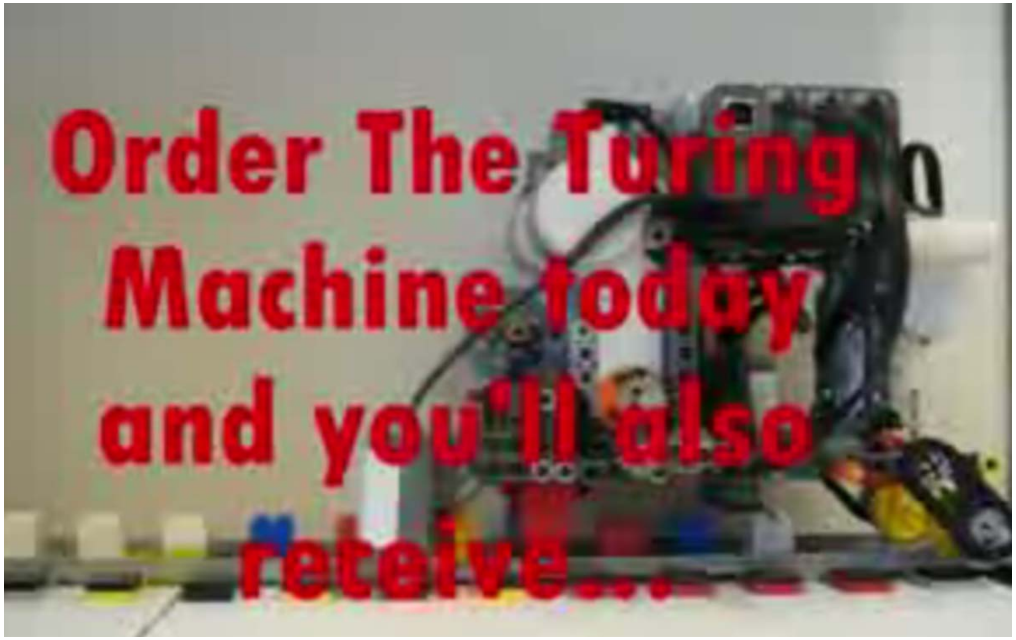


Hashing with Linear Probing:

Ziel: Folgen besetzter Feldpositionen in Hashtabelle T nicht zu lang

Die LEGO Turing Maschine:

http://www.youtube.com/watch?v=cYw2ewoO6c4



Tschüss und viel Erfolg an der Klausur!

Bonus Material

$$
\begin{aligned}
S &\rightarrow Z \mid \epsilon\,, \\
X &\rightarrow aXZ \mid aX \mid a\,, \\
Y &\rightarrow ZbZ \mid Zb \mid bZ \mid X \mid b \mid bb\,, \\
Z &\rightarrow XYZ \mid XY\,.
\end{aligned}
$$

Chomsky Normalform?

Lösungsvorschlag

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ist in Chomsky-Normalform, falls alle Produktionen aus P eine der drei Formen $A \to BC$, $A \to a$ oder $S \to \epsilon$ besitzen, wobei $A, B, C \in V$, $a \in \Sigma$.
Die Konstruktion einer zu G äquivalenten Grammatik G' erfolgt in drei aufeinanderfolgenden Schritten gemäß Vorlesung vom 11.5.07., in denen jeweils zu G äquivalente Grammatiken G_1, G_2, G_3 konstruiert werden. Die gesuchte Grammatik ist dann $G' = G_3$.

- Konstruktion einer separierten kontextfreien Grammatik G_1.
- Darstellung aller Produktionen von G_1, deren rechte Seiten länger als 2 ist, durch eine Folge von Produktionen, deren rechte Seiten die Länge 2 besitzen. Ergebnis G_2.
- Entfernung der Kettenproduktionen der Form $A \to B$ aus G_2.

Wir übernehmen die separierte Grammatik aus der Vorbereitungsaufgabe 2.

$$\begin{array}{rcl}
S & \to & Z \mid \epsilon\,, \\
X & \to & AXZ \mid AX \mid A\,, \\
Y & \to & ZBZ \mid ZB \mid BZ \mid X \mid B \mid BB\,, \\
Z & \to & XYZ \mid XY\,, \\
A & \to & a\,, \\
B & \to & b\,.
\end{array}$$

Im zweiten Schritt werden lange Produktionen durch eine Folge von kurzen Produktionen ersetzt.
Dazu führen wir neue Variable C_1, C_2, C_3 ein.

$$\begin{aligned}
S &\rightarrow Z \mid \epsilon\,, \\
X &\rightarrow AC_1 \mid AX \mid A\,, \\
C_1 &\rightarrow XZ\,, \\
Y &\rightarrow ZC_2 \mid ZB \mid BZ \mid X \mid B \mid BB\,, \\
C_2 &\rightarrow BZ\,, \\
Z &\rightarrow XC_3 \mid XY\,, \\
C_3 &\rightarrow YZ\,, \\
A &\rightarrow a\,, \\
B &\rightarrow b\,.
\end{aligned}$$

Die Entfernung von Kettenproduktionen $C \rightarrow D$ mit $C, D \in V$ geschieht auf der Basis der bisher erhaltenen Grammatik G_2, und zwar wiederum in 3 aufeinanderfolgenden Schritten. Grundlage der Operationen ist die Bestimmung der Relation $C {\rightarrow_G}^+ D$. d. h. der Relation, die darstellt, welche Variablen aus welchen anderen herleitbar sind. Für kontextfreie Grammatiken ist das natürlich eine Relation über der Menge der Variablen, denn es kann keine Umwege über längere Wörter geben.
Die vollständige Liste der Paare der transitiven Relation ${\rightarrow_G}^+$ über V enthält sicherlich $S {\rightarrow_G}^+ Z$, $X {\rightarrow_G}^+ A$, $Y {\rightarrow_G}^+ X$ und $Y {\rightarrow_G}^+ B$.
Die Bildung der transitiven Hülle liefert noch $Y {\rightarrow_G}^+ A$.

1. Schritt:
Gemäß Verfahren in der Vorlesung fügen wir Produktionen hinzu, und zwar die Produktion $A \to DB$ bzw. $A \to BD$, falls $A \to CB$ oder $A \to BC$ und $C {\to_G}^+ D$ gilt.
Für X an der Stelle von C, also $X {\to_G}^+ A$:

$$\begin{aligned} X &\to AA\,, \\ C_1 &\to AZ\,, \\ Z &\to AC_3 \mid AY\,. \end{aligned}$$

Für Y an der Stelle von C, also $Y {\to_G}^+ X$, $Y {\to_G}^+ B$ und $Y {\to_G}^+ A$:

$$\begin{aligned} Z &\to XX \mid XB \mid XA\,, \\ C_3 &\to XZ \mid BZ \mid AZ\,. \end{aligned}$$

2. Schritt:
Wichtig ist, dass wir nun von der bereits erweiterten Produktionenmenge ausgehen, die bereits weitgehend ohne Anwendung von Kettenproduktionen auskommt.
Allerdings sind Ableitungen, die bei S mit Kettenproduktionen beginnen, bisher nicht berücksichtigt worden. Deshalb fügen wir in diesem Schritt alle Produktionen der Form $S \to UV$ hinzu, falls $S {\to_G}^+ D$ und $D \to UV$ gilt. Schließlich fügen wir gfs. auch Produktionen der Form $S \to a$ mit $a \in \Sigma$ hinzu, falls $S {\to_G}^+ D$ und $D \to a$ gilt.
Da $S {\to_G}^+ Z$ das einzige Tupel der Relation ist, fügen wir die folgenden Produktionen hinzu.

$$S \to XC_3 \mid XY \mid AC_3 \mid AY \mid XX \mid XB \mid XA\,.$$

3. Schritt:
Entfernung aller Kettenproduktionen.

Wir erhalten als Ergebnis die folgende Grammatik G' mit den Produktionen

$$
\begin{array}{rcl}
S & \rightarrow & XC_3 \mid XY \mid AC_3 \mid AY \mid XX \mid XB \mid XA \mid \epsilon\,, \\
X & \rightarrow & AC_1 \mid AX \mid AA\,, \\
C_1 & \rightarrow & XZ \mid AZ\,, \\
Y & \rightarrow & ZC_2 \mid ZB \mid BZ \mid BB\,, \\
C_2 & \rightarrow & BZ\,, \\
Z & \rightarrow & XC_3 \mid XY \mid AC_3 \mid AY \mid XX \mid XB \mid XA\,, \\
C_3 & \rightarrow & YZ \mid XZ \mid BZ \mid AZ\,, \\
A & \rightarrow & a\,, \\
B & \rightarrow & b\,.
\end{array}
$$

Hashing Formulas

3.4.1 Kollisionsauflösung

Hashing durch Verkettung

In jedem Feldelement der Hashtabelle wird eine lineare Liste der Schlüssel abgespeichert, die durch die Hashfunktion auf dieses Feldelement abgebildet werden. Die Implementierung der Operationen is_member, insert und delete ist offensichtlich.

Sei $\alpha = \frac{n}{m}$ der Füllgrad der Hashtabelle. Dann beträgt, bei Gleichverteilung der Schlüssel, die Länge einer jeden der linearen Listen im Erwartungswert α und der Zeitaufwand für die Suche nach einem Schlüssel im

Hashfkt. zu Array

halbe Liste durchlaufen

$$\text{erfolgreichen Fall } 1 + \frac{\alpha}{2}$$

$$\text{erfolglosen Fall } 1 + \alpha$$

Für die Operationen insert und delete ist darüber hinaus lediglich ein Zeitaufwand von $O(1)$ erforderlich.

Hashing with Linear Probing

Lauf:

	14		5	12	3	19	1		10	

Lauf der Länge 5

Theorem 4.1: Falls n Elemente in einer Hashtabelle T der Größe $m>2en$ mittels einer zufälligen Hashfunktion h gespeichert werden, dann ist für jedes $T[i]$ die erwartete Länge eines Laufes in T, der $T[i]$ enthält, $O(1)$. (e: Eulersche Zahl)

Hashing with Linear Probing

Beweis:

n: Anzahl Elemente, $m>2n$: Größe der Tabelle

Anzahl Möglichkeiten zur Wahl von k Elementen: $\binom{n}{k}$

Wahrscheinlichkeit, dass Hashwerte in Lauf: $k \cdot (k/m)^k$

$\Pr[T[i]$ in Lauf der Länge $k] < (en/k)^k\, k(k/m)^k < k(1/2)^k$

Hashing with Linear Probing

- p_k := Pr[T[i] in Lauf der Länge k] < $k\,(1/2)^k$
- E[Länge des Laufs über T[i]]
  $= \sum_{k>=0} k \cdot p_k < \sum_{k>=0} k^2\,(1/2)^k = O(1)$

Also erwartet konstanter Aufwand für

Operationen insert, remove und find.