0. Motivation und Inhalt der Vorlesungen A. u. B. I und II

Bisher:

- Aufbau von
  - Programmiersprachen
  - Computer

  in den Grundzügen.

- Lösen von einfachen Aufgaben mit Hilfe des Rechners.

Zielsetzung:

- Erarbeiten von grundlegenden Datenstrukturen und Methoden zur Lösung von in der Praxis auftretenden Fragestellungen.

- Lösen elementarer Probleme, die häufig selbst als Teilprobleme bei der Lösung von Fragestellungen in der Praxis auftreten.

- Befassen mit der Frage, welche Funktionen überhaupt berechenbar sind nebst formale Charakterisierung der "im intuitiven Sinn" berechenbaren Funktionen.

- Charakterisierung der "praktisch" berechenbaren Funktionen sowie einer wichtigen Klasse

von in der Praxis häufig auftretenden "bisher nicht praktisch" berechenbaren Funktionen (NP-vollständig) nebst deren Behandlung in der Praxis.

Inhalt:

1. Datenstrukturen zur Lösung von Mengenverwaltungsproblemen.
2. Durchmusterung von Graphen
3. Entwurf von Algorithmen
4. Automatentheorie und formale Sprachen Teil 2
   - Die Syntaxanalyse (kontextfreie Grammatiken, Kellerautomat)
5. Theoretische Berechenbarkeit
   5.1 Turingmaschine
   5.2 primitiv rekursive und μ-rekursive Funktionen
   5.3 Entscheidbarkeit und Aufzählbarkeit
6. Praktische Berechenbarkeit
   6.1 Random Access Maschine
   6.2 Die Klassen P und NP
   6.3 NP-vollständige Probleme
7. Entwurf von Algorithmen (Fortführung)
8. Algorithmen auf Graphen
9. Lineare Programmierung

Lernziele:

- Kenntnis der grundlegenden Datenstrukturen, Methoden und Konzepte
- Fähigkeit, ein gegebenes Problem dahin gehend zu analysieren, dass
  - seine Schwierigkeit eingeordnet wird und
  - Eigenschaften herausgearbeitet werden, anhand derer geeignete Datenstrukturen und Methoden ausgewählt werden.
- Entwurf eines Algorithmus zur Lösung des Problems, wobei die gewählten Datenstrukturen und Methoden ihre Anwendung finden.
- Analyse des entwickelten Algorithmus.

Literatur

- N. Blum, Algorithmen und Datenstrukturen, 2. Auflage, Oldenbourg Verlag, 2013. (AuD)
- N. Blum, Einführung in Formale Sprachen, Berechenbarkeit, Informations- und Lerntheorie, Oldenbourg Verlag, 2007. (FSuB)
- A. V. Aho, J. E. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, MIT Press, 2001

J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2001.

1. Datenstrukturen zur Lösung von Mengenverwaltungsproblemen

· Computerprogramme verwalten und manipulieren Daten.

Beispiel: Datenbank einer Bank

$\rightsquigarrow$

Wie strukturiert man die Daten, so dass die Operationen auf diesen effizient durchgeführt werden können?

Universum U Menge der möglichen Daten

$S \subseteq U$ zu verwaltende Menge

$\uparrow$

besteht aus Objekten, die sich aus

- Schlüssel (dient der Mengenmanipulation)
- Information

zusammensetzen.

$\rightsquigarrow$

U besteht aus der Menge der möglichen Schlüssel.

$(U, \leq)$ lineare Ordnung auf U.

<u>Typische einfache Operationen</u>:

Name	Effekt
Zugriff(a,S)	if $a \in S$ then Ausgabe := unter dem Schlüssel a gespeicherte Information else Ausgabe := $a \notin S$ fi.
Einfügen(a,S)	$S := S \cup \{a\}$
Streichen(a,S)	$S := S \setminus \{a\}$

<u>Ziel</u>:

Entwicklung von Datenstrukturen zur effizienten Lösung von Mengenverwaltungsproblemen.

1.1 Einfache Datenstrukturen

- Einfachste Strukturierungsart: Feld (array)
  - Durchmusterungsarten von Feldern
  - binäre Suche

Datenstrukturen, die leicht mittels eines Feldes realisiert werden können:

- Keller
- Schlange

- Listen
  - Realisierung durch
    a) Felder
    b) verkettete Speicherung
  - sortierte Liste

    Zugriff, Einfügen, Streichen.
    binäre Suche

- gelernt im 1. Studienjahr (ADIP)

  steht auch in

  AuD, S. 4-9.

06.10.

1.2 Bäume

Nachteil von sortierten Listen:

Einfüge-, Streiche- oder Zugriffsoperationen können Länge der Liste Zeiteinheiten benötigen.

Ziel: Vermeidung dieses Problems.

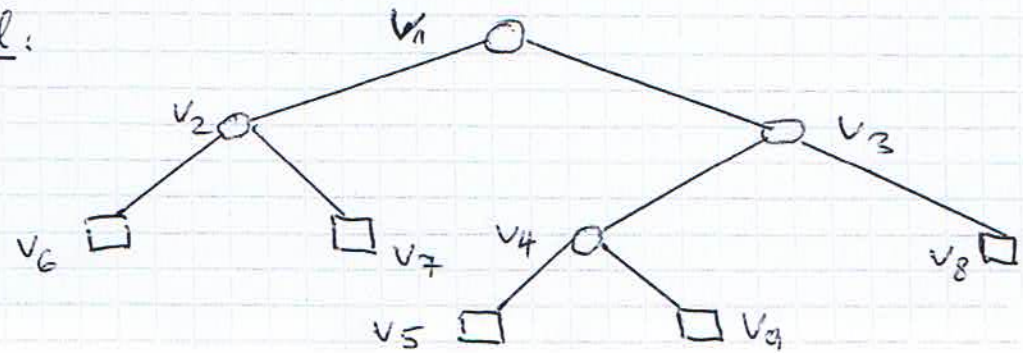
Sei $V = \{v_1, v_2, \dots\}$ eine unendliche Menge von Knoten.

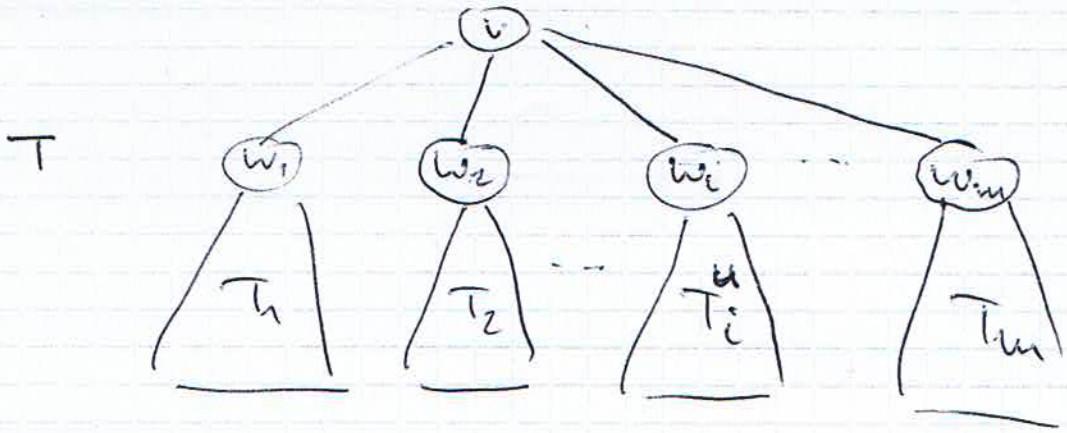
Induktive Definition der Menge der (geordneten) Bäume:
(etwas anders als in ADP)

1. Jeder Knoten $v_i \in V$ ist ein Baum. v_i ist die Wurzel des Baumes.

2. Wenn $T_1, T_2, \dots, T_m$ $(m \geq 1)$ Bäume mit paarweise disjunkten Knotenmengen sind und v ein neuer Knoten ist, dann ist das $(m+1)$-Tupel $T = [v, T_1, T_2, \dots, T_m]$ ein Baum. Der neue Knoten v heißt Wurzel des Baumes T. Der Baum T_i heißt i-ter direkter Unterbaum (Teilbaum) von T. Die Zahl m ist der Grad des Knotens v in T.

Beispiel:





<u>Schreib- und Sprechweisen:</u>

- w_i = Wurzel (T_i)
- w_i heißt i-ter <u>Sohn</u> von v
  v heißt <u>Vater</u> von w_i
- <u>Nachfolger</u> (<u>Vorgänger</u>)
  reflexive, transitive Hülle der Sohn (Vater)-Relation.
- w_j, $j \neq i$ heißt <u>Bruder</u> von w_i.
- Knoten von Grad 0 (>0) heißt <u>Blatt</u> (<u>innerer Knoten</u>) des Baumes T.
- <u>Tiefe von u bzgl. T</u> Tiefe (u, T)

$$\text{Tiefe}(u,T) = \begin{cases} 0 & \text{falls } u = \text{Wurzel}(T) \\ 1 + \text{Tiefe}(u, T_i) & \text{sonst,} \end{cases}$$

wobei T_i derjenige direkte Unterbaum von T ist, der u enthält.

- Höhe eines Baumes T Höhe(T)

$$\text{Höhe}(T) = \max\{\text{Tiefe}(b,T) \mid b \text{ ist Blatt von } T\}$$

Ein Baum T heißt Binärbaum, wenn jeder innere Knoten von T den Grad 2 hat.

1. (2.) Unterbaum ≙ linker (rechter) UB.

Speicherung von Information in einem Baum:

- Blattorientierte Speicherung:

  Speicherung von Daten nur in den Blättern

- Knotenorientierte Speicherung:

  Speicherung von Daten nur in den inneren Knoten.

Realisierung eines Binärbaumes mit knotenorientierter Speicherung durch drei Felder:

	INHALT	LSOHN	RSOHN
1	Schlüssel, Information zu v_3	6	3
2	Schlüssel, Information zu v_1	4	1
3	Schlüssel, Information zu v_8	0	0
4	Schlüssel, Information zu v_2	5	9
5	Schlüssel, Information zu v_6	0	0
6	Schlüssel, Information zu v_4	8	7
7	Schlüssel, Information zu v_9	0	0
8	Schlüssel, Information zu v_5	0	0
9	Schlüssel, Information zu v_7	0	0

Wurzel: $\boxed{2}$

Abbildung 3.4: Realisierung des Binärbaumes aus obigen Beispiel

Durchmusterungsmethoden für Binärbäume

Sei T ein Binärbaum mit Wurzel w, linken Unterbaum L und rechten Unterbaum R.

Präordnung:
Besuche die Wurzel, dann den linken Unterbaum und dann den rechten Unterbaum:
w L R

Postordnung:
Durchmustere den linken Unterbaum, dann den rechten Unterbaum und dann die Wurzel:
L R w

(ADIP: Inorder)
Symmetrische (lexikographische) Ordnung:
Durchmustere den linken Unterbaum, dann die Wurzel und dann den rechten Unterbaum
L w R.

1.2.1 Beliebige Suchbäume

Sei auf die Menge aller Schlüssel eine lineare Ordnung definiert. T heißt genau dann sortiert oder Suchbaum, wenn

1. L und R sortiert sind,
2. Schlüssel (v) ≤ Schlüssel (w) für alle Knoten v in L und
3. Schlüssel (w) < Schlüssel (v) für alle Knoten v in R.

Zugriff (x, T)

<u>gegeben:</u> Schlüssel x

<u>Aufgabe:</u> Falls Knoten v in T mit Schlüssel(v) = x existiert, dann gib die korrespondierende Information aus.

Starte in Wurzel w und verfahre wie folgt:

- <u>x = Schlüssel(w)</u>:

  Information gefunden — falls knotenorientierte Speicherung

  Suche in L weiter — sonst

- <u>x < Schlüssel(w)</u>:

  Suche in L weiter.

- <u>x > Schlüssel(w)</u>:

  Suche in R weiter.

Falls der betrachtete Knoten ein Blatt ist und nicht den Schlüssel x enthält, dann existiert in T kein Knoten v mit Schlüssel(v) = x.

- Einfügen und Streichen eines Knotens v mit Schlüssel(v) = x anhand eines Beispiels. (knotenorientiert)

09.10.

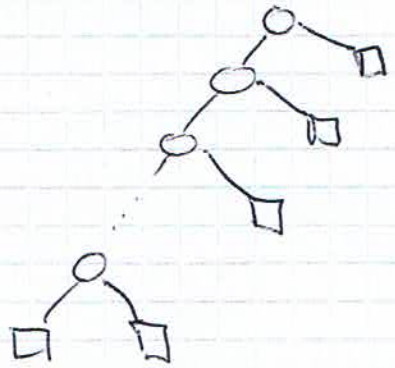
- <u>maximale Suchzeit:</u>

  beschränkt durch Höhe(T).

Nachteil:

Baum mit n Knoten kann Höhe $\frac{n}{2}$ haben.

Bsp.: Liste



Frage: Was ist die kleinstmögliche Suchzeit?

gegeben: Suchbaum T in dem $\{x_1, x_2, \dots, x_n\}$ knotenorientiert gespeichert sind

Bezeichnungen:

- b_i Tiefe des Knotens in T, der x_i enthält
- $P := \frac{1}{n} \cdot \sum_{i=1}^{n} (b_i + 1)$ mittlere Weglänge von T

Satz 1.1:

Sei T ein binärer Suchbaum für die Menge $S = \{x_1, x_2, \dots, x_n\}$ bei Knotenorientierter Speicherung. Dann gilt

a) Höhe (T) $\geq \lceil \log(n+1) \rceil$

b) $P \geq \lfloor \log(n+1) \rfloor - 1$

Beweis:

a) z.z.: Ein beliebiger Binärbaum mit n inneren Knoten hat Höhe

$$h \geq \lceil \log(n+1) \rceil$$

Beobachtung:

innere Knoten befinden sich nur in Tiefen $0, 1, 2, \ldots, h-1$.

Sei n_i die Anzahl der inneren Knoten der Tiefe i. Dann kann leicht mittels Induktion über i

$$n_i \leq 2^i$$

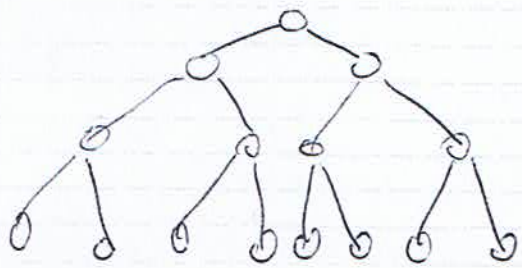
bewiesen werden. Also gilt:

$$n = \sum_{i=0}^{h-1} n_i \leq \sum_{i=0}^{h-1} 2^i = 2^h - 1$$

$$\Leftrightarrow \quad n+1 \leq 2^h \quad \Leftrightarrow \quad \log(n+1) \leq h$$

$$\underset{h \text{ ganzzahlig}}{\Longrightarrow} \quad \lceil \log(n+1) \rceil \leq h$$

b) Es gilt, P ist minimal, falls T



1 Knoten der Tiefe 0
2 Knoten der Tiefe 1
4 Knoten der Tiefe 2
⋮
2^k Knoten der Tiefe k

und $n - \sum_{i=0}^{k} 2^i = n - (2^{k+1} - 1)$ Knoten der Tiefe $k+1$

für $k = \lfloor \log(n+1) \rfloor - 1$ besitzt.

Also gilt.

$$P \geq \frac{1}{n} \cdot \left(\sum_{i=0}^{k} (i+1) 2^i + (k+2)(n - 2^{k+1} + 1) \right)$$

Da $\sum_{i=0}^{k} i 2^i = (k-1) 2^{k+1} + 2$ für $k \geq 1$ erhalten wir

Beweis durch vollst. Ind.

$$= \frac{1}{n} \left((k-1) 2^{k+1} + 2 + (2^{k+1} - 1) + (k+2)(n - 2^{k+1} + 1) \right)$$

$$= \frac{1}{n} \left(k \cdot 2^{k+1} + 1 + (k+2)(n - 2^{k+1} + 1) \right)$$

$$= \frac{1}{n} \left(k(n+1) + 1 + 2 \underbrace{(n - 2^{k+1} + 1)}_{\geq 0} \right)$$

$$\geq k$$

$$= \lfloor \log(n+1) \rfloor - 1$$

$\blacksquare$

$\rightsquigarrow$

<u>vollständig ausgeglichener</u> binärer Suchbaum

<u>Vorteil:</u> Zugriffszeit optimal

<u>Nachteil:</u> Eine Einfüge- oder Streichoperation kann den Umbau des gesamten Baumes nach sich ziehen.

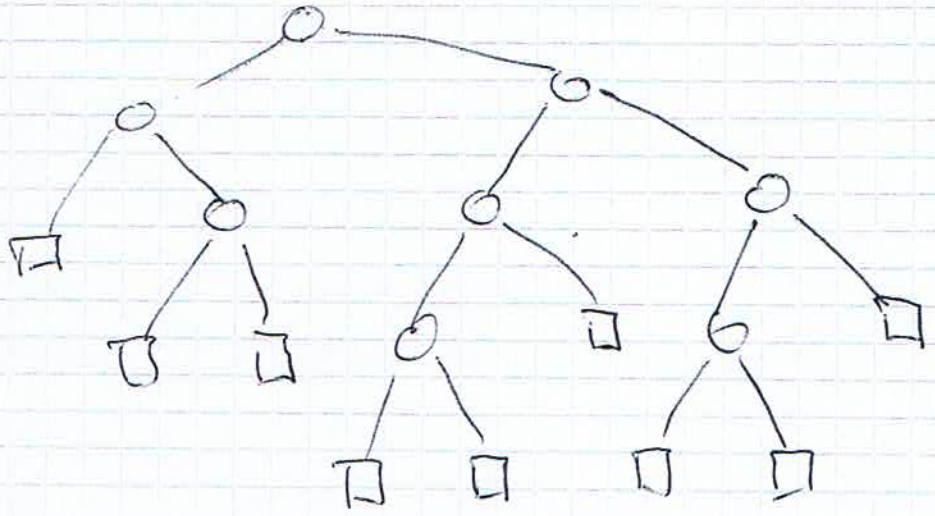
Ziel:

Konstruktion von Suchbäumen, die sowohl die Operation Zugriff, als auch die Operationen Einfügen und Streichen nahezu optimal (d.h. in logarithmischer Zeit) lösen.
(balancierte Suchbäume)

1.2.2 AVL-Bäume

Ein binärer Suchbaum T heißt AVL-Baum, falls für jeden Knoten v in T die Höhen der linken und rechten Teilbäumen sich höchstens um 1 unterscheiden.

Bsp.:



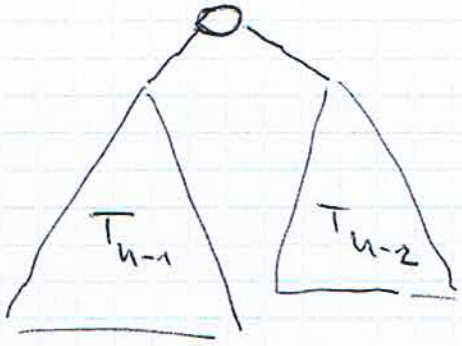
Satz 1.2

Sei T ein AVL-Baum mit n Blättern. Dann gilt:

$$\text{Höhe}(T) \leq 1{,}4405 \cdot \log(n+1).$$

Beweis:

Sei n_h die minimale Anzahl von Blättern, die ein AVL-Baum der Höhe h haben kann. Sei T_h ein AVL-Baum der Höhe h mit n_h Blättern. Dann sieht T_h wie folgt aus



oder umgekehrt.

Also gilt:

$$n_h = n_{h-1} + n_{h-2}$$

Betrachte Zahlenfolge F_n mit

$$F_0 = 0 \,, \quad F_1 = 1 \quad \text{und}$$

$$F_{n+2} = F_{n+1} + F_n \quad \text{für } n \geq 0$$

Fibonacci-Folge

Wegen $n_0 = 1$, $n_1 = 2$ und $n_h = n_{h-1} + n_{h-2}$, $h \geq 2$ gilt:

$$n_h = F_{h+2}.$$

Lemma 1.1

$$F_i = \frac{\alpha^i - \beta^i}{\sqrt{5}}, \quad \text{wobei}$$

$$\alpha = \frac{1+\sqrt{5}}{2} \approx 1{,}618 \quad \text{und}$$

$$\beta = 1 - \alpha = \frac{1-\sqrt{5}}{2} \approx -0{,}618.$$

Beweis:

- einfache Induktion über i.
- Siehe Knuth, The Art of Computer Programming Vol 1, (3. Aufl.), p 78 ff. bzw. p.79 ff.

$\blacksquare$

Lemma 1.1 impliziert

$$n_h = \frac{\alpha^{h+2} - \beta^{h+2}}{\sqrt{5}}$$

$$\geq \frac{1}{\sqrt{5}} \left(\alpha^{h+2} - 1 \right)$$

$$\Leftrightarrow \quad \alpha^{h+2} \leq \sqrt{5}\, n_h + 1$$

$$\Rightarrow \quad h \leq \frac{\log \sqrt{5}}{\log \alpha} + \frac{\log(n_h + 1)}{\log \alpha} - 2$$

$$\leq 1{,}4405 \log(n_h + 1).$$

$\blacksquare$

Einfügungen und Streichen in AVL-Bäumen

- Führe die Operationen Einfügen (a, S) und Streichen (a, S) wie bei binären Suchbäumen durch.
- Sorge dafür, daß der resultierende Baum wieder ein AVL-Baum wird.

Bezeichnung: $v \in T$, L (R) linker (rechter) Unterbaum von v

$$\rho(v) = \text{Höhe}(R) - \text{Höhe}(L)$$

Balance des Knotens v.

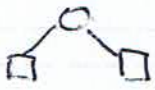
In AVL-Bäumen gilt stets: $\rho(v) \in \{-1, 0, 1\}$

Nach Durchführung der Einfüge- bzw. Streichoperation kann

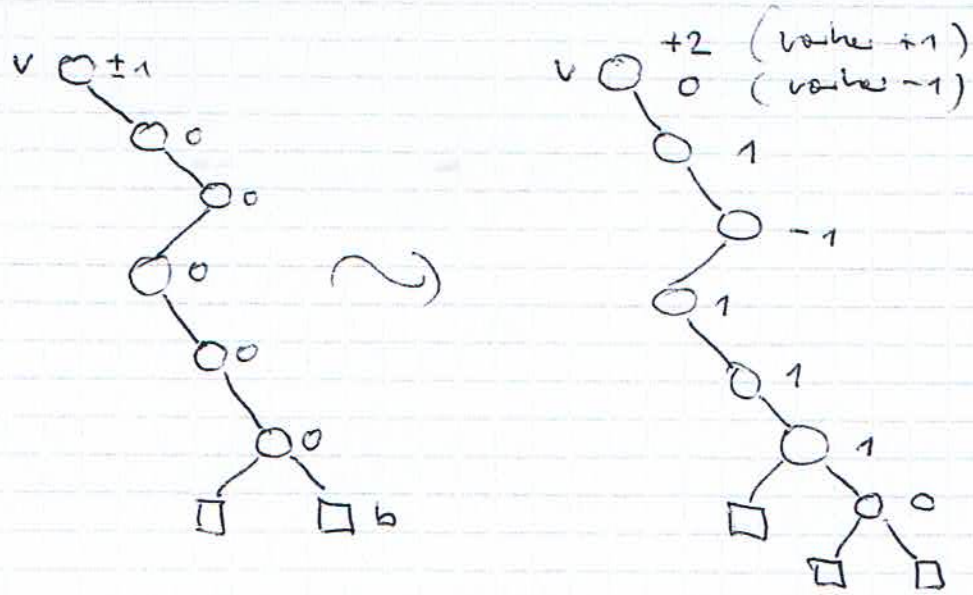
$$\rho(v) = +2 \quad \text{oder} \quad \rho(v) = -2$$

sein. D.h., $\rho(v) \in \{-2, -1, 0, 1, 2\}$ $\forall v$ auf dem Suchpfad

Einfügen (a, S):

Ein Blatt b $\square$ wird durch  ersetzt.

Betrachte den Pfad P von b bis zum ersten Vorgänger von b in T mit $\rho(v) \neq 0$ vor Durchführung von Einfügen (a, S).



Sei $\rho'(v)$ die Balance von v nach Durchführung der Einfüge/Streiche-Operation.

<u>1. Fall:</u> $\rho'(v) = 0$

Die Höhe des Unterbaumes mit Wurzel v hat sich nicht geändert.

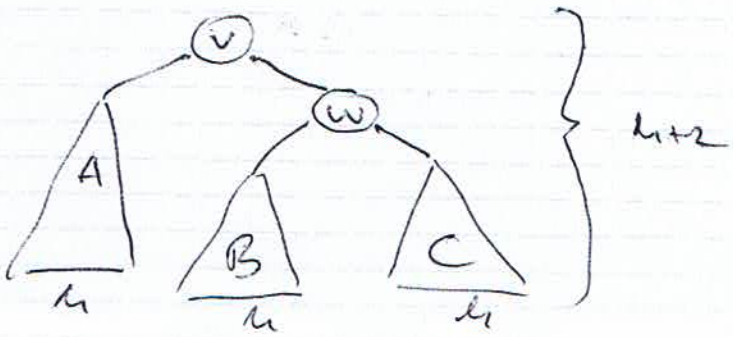
$\Longrightarrow$

Oberhalb von v ändern sich auf dem Suchpfad keine Balancen

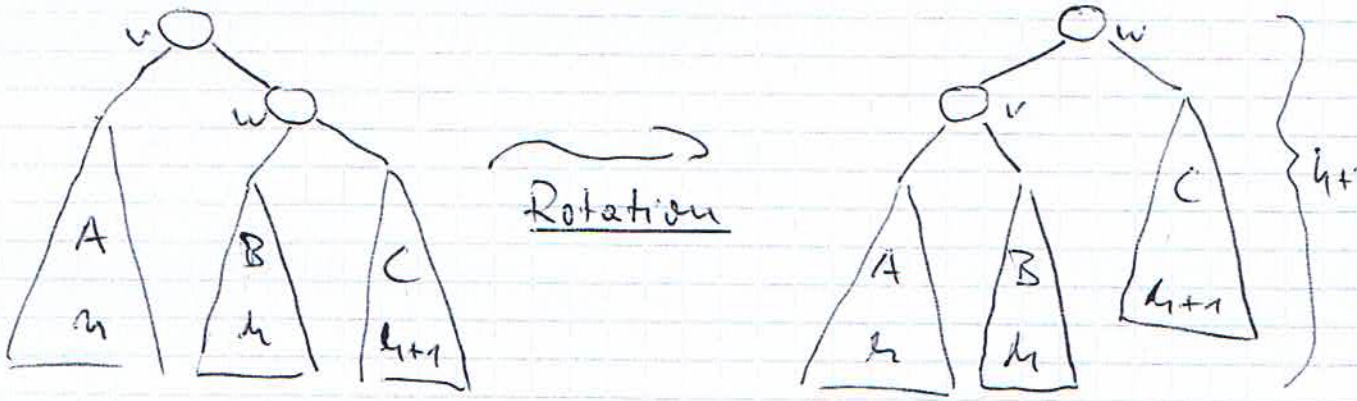
$\rightsquigarrow$

Der resultierende Baum ist ein AVL-Baum.

<u>2. Fall:</u> $\rho'(v) = +2$.



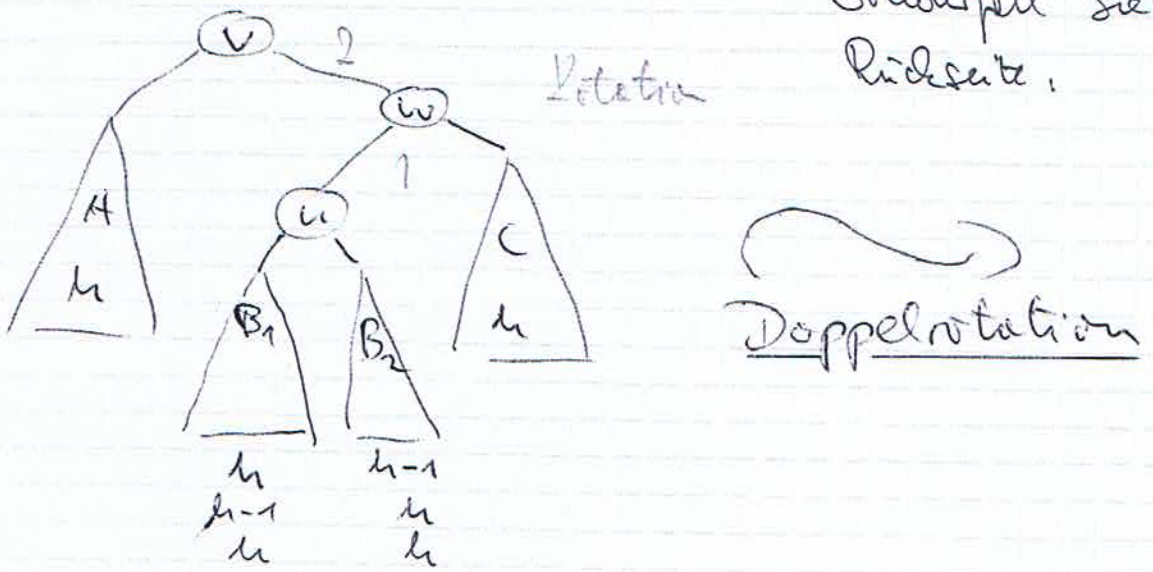
2.1 a wird in C eingefügt.

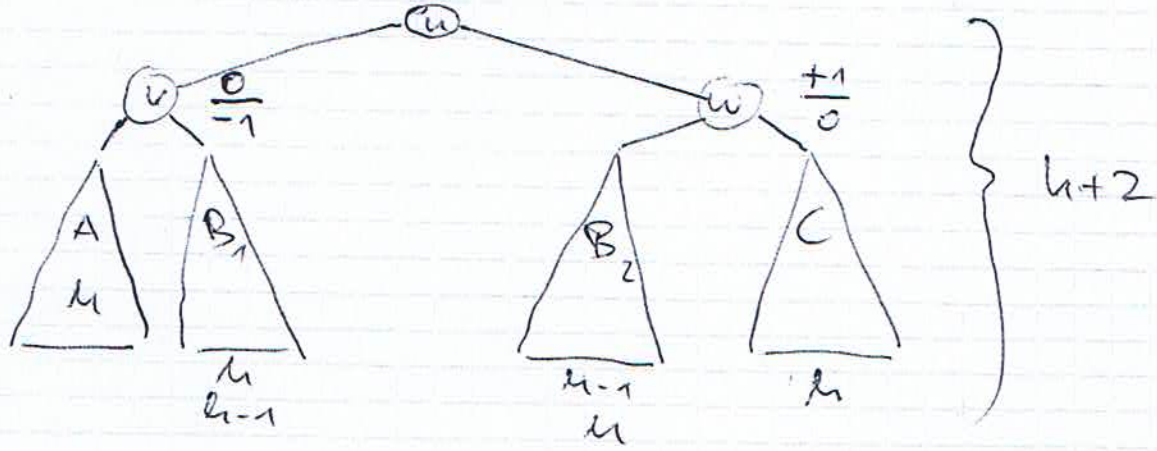


Höhe des gesamten Unterbaumes ändert sich gegenüber vorher nicht.

2.2 a wird in B eingefügt.

Sonderfall siehe Rückseite.





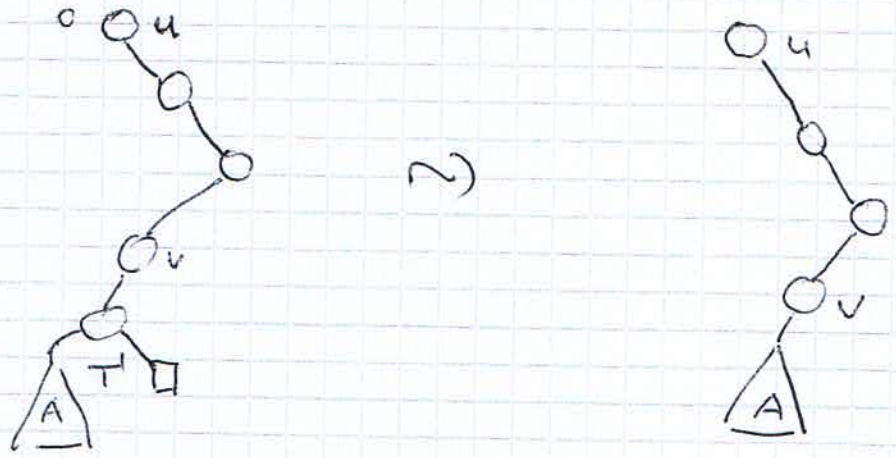
Höhe des gesamten Unterbaumes ändert sich gegenüber vorher nicht.

<u>Streichen (a, S)</u>:

<u>Effekt</u>:

Ein Teilbaum T' wird durch seinen linken oder seinen rechten Unterbaum A ersetzt. Der andere Unterbaum von T' ist ein Blatt.

Sei v der direkte Vorgänger von Wurzel(T'). Sei P der Pfad von v zu seinem letzten Vorgänger u mit $\rho(u) = 0$ <u>vor</u> Durchführung von Streichen(a, S)



<u>1. Fall</u>: $\rho(v) = 0$ (d.h. $v = u$)

Dann gilt: $\rho'(v) = +1$ und die Höhe des Unterbaumes mit Wurzel v hat sich nicht geändert. ⇝ Balancen von Vorgängern unterschiedlich sich nicht.

<u>2. Fall</u>: $\rho(v) = -1$

Dann gilt: $\rho'(v) = 0$

<u>aber</u>

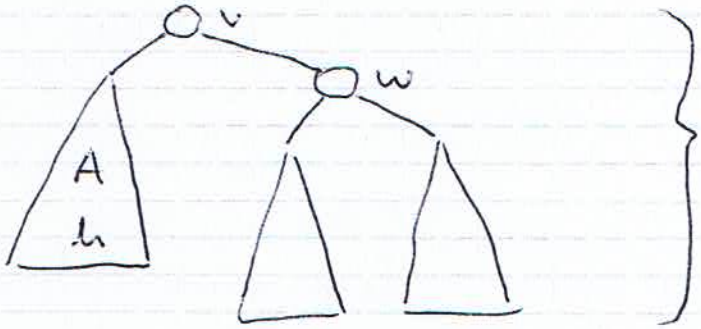
Die Höhe des Unterbaumes mit Wurzel v hat sich um 1 verringert.

$\Rightarrow$ Vater(v) muß noch betrachtet werden.

<u>3. Fall</u>: $\rho(v) = +1$

Dann gilt: $\rho'(v) = +2$

$\Rightarrow$ Baum muß umstrukturiert werden.

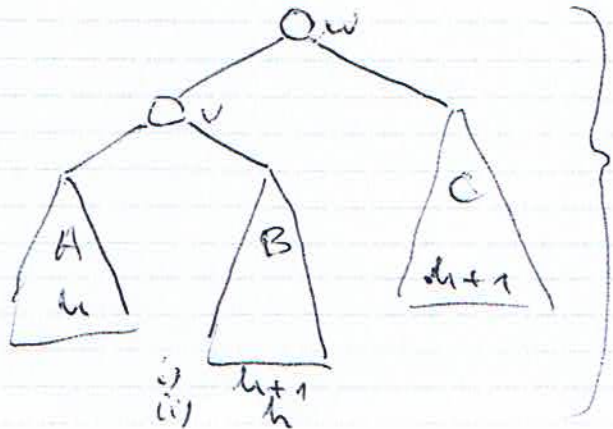


nach Speichern (v, S)

Fälle			
	i)	h+1	h+1
	ii)	h	h+1
	iii)	h+1	h

<u>i) und ii)</u>

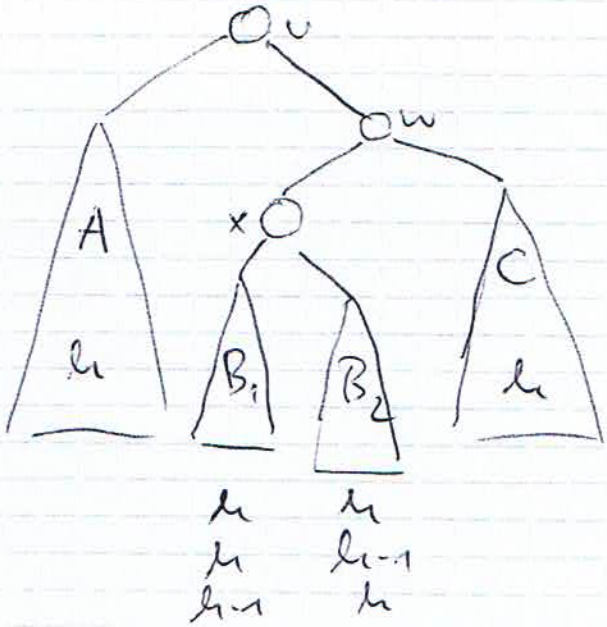
Rotation



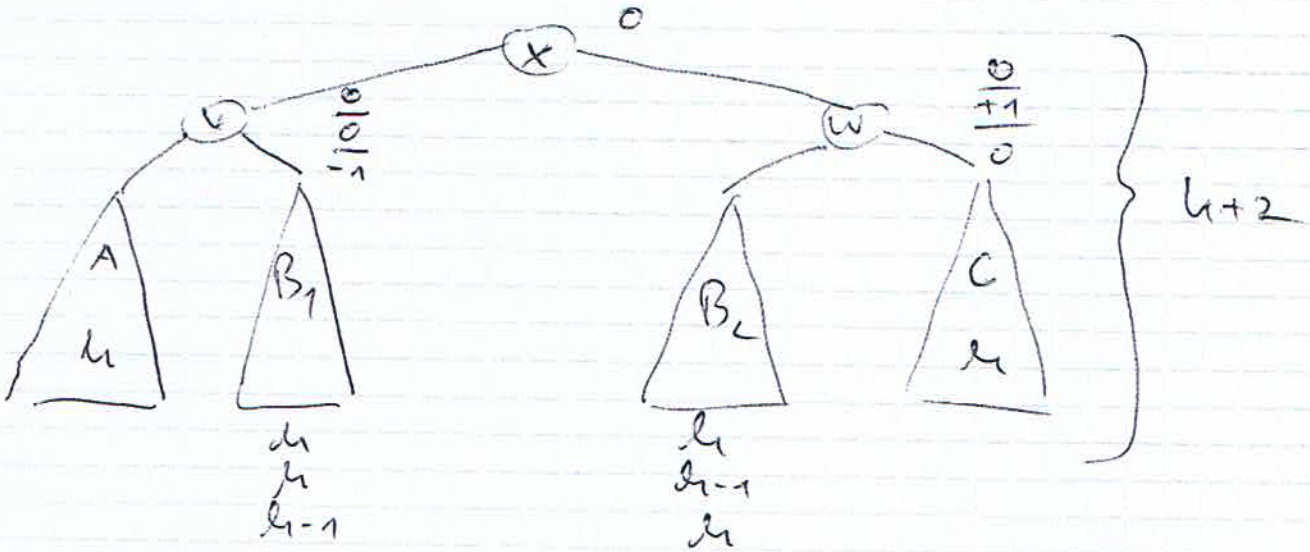
Höhe des Teilbaumes hat sich im Fall i) nicht verändert. ⇒ Vorgängerknoten müssen nicht betrachtet werden.

Im Fall ii) verringert sich die Höhe des Teilbaumes um 1. ⇒ Vater(w) muß noch betrachtet werden.

iii)



⇝ Doppelrotation



Die Höhe des Teilbaumes verringert sich um 1, so daß Vater(x) noch betrachtet werden muß.

- Rebalancierungen enden spätestens beim Knoten u bzw. bei der Wurzel des Baums.
- Rebalancierungen werden nur auf dem Suchpfad durchgeführt.

$\rightsquigarrow$

<u>Satz 1.3</u>

Die Operationen Zugriff, Einfügen und Streichen können in AVL-Bäumen mit n Blättern in $O(\log n)$ Zeit ausgeführt werden.

<u>1.2.3 B-Bäume</u>

- <u>Eigenschaften von vollständig ausgeglichenen binären Suchbäumen:</u>

  1. Alle inneren Knoten haben zwei Söhne.
  2. Die Tiefen zweier Blätter unterscheiden sich maximal um 1.

  $\rightsquigarrow$

  größere Freiheitsgrade um effizientes Einfügen und Streichen zu ermöglichen.

- <u>AVL-Bäume</u>

  Bedingung bzgl der Tiefen der Blätter aufgehoben. (Satz 1.2 $\Rightarrow$ Differenz $\leq 1{,}4405 \log(n+1)$)

balancierter Baum, der Eigenschaft 2. aufrecht hält, jedoch Eigenschaft 1. aufhebt:

Ein Baum T ist ein <u>B-Baum der Ordnung k</u>, $k \in \mathbb{N}$, $k \geq 2$, falls gilt:

1. Alle Blätter haben dieselbe Tiefe.
2. Die Wurzel hat mindestens zwei und jeder andere innere Knoten mindestens k Söhne.
3. Jeder innere Knoten hat höchstens $2k-1$ Söhne.

<u>Satz 1.4</u>

Sei T ein B-Baum der Ordnung k mit Höhe h und n Blättern. Dann gilt:

$$2 \cdot k^{h-1} \leq n \leq (2k-1)^h.$$

<u>Beweis:</u>

Anzahl der Blätter in T ist minimal, wenn die Wurzel genau 2 und jeder andere innere Knoten genau k Söhne hat. Also gilt

$$2 \cdot k^{h-1} \leq n.$$

Anzahl der Blätter in T ist maximal, wenn jeder innere Knoten genau $2k-1$ Söhne besitzt. Also gilt

$$n \leq (2k-1)^h$$

∎

Korollar 1.1

Sei T ein B-Baum der Ordnung k mit n Blättern. Dann gilt für seine Höhe h

$$\log_{(2k-1)} n \leq h \leq 1 + \log_k \frac{n}{2}$$

$\blacksquare$

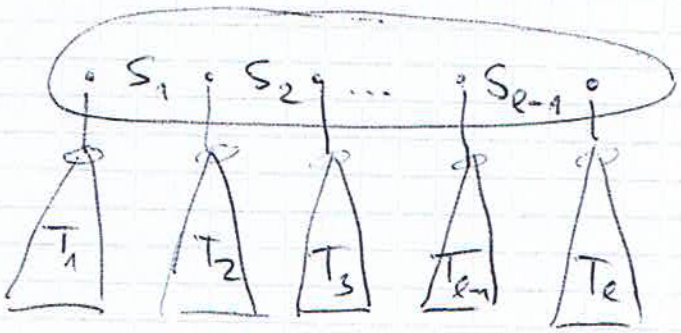
Vorteil von B-Bäumen:

- gut geeignet zur Speicherung großer Mengen, die nicht insgesamt in den Hauptspeicher abgelegt werden können.
  - Ordnung k wird häufig derart gewählt, daß ein Knoten eine ganze Seite füllt.

  $\Rightarrow$

  in Anwendungen ist die Höhe meistens gering.

Anordnung der Schlüssel in sortierten B-Bäumen:



innerer Knoten mit ℓ Söhnen.

enthält

$\ell - 1$ Schlüssel

ℓ Zeiger auf Söhne

Stets gilt $\forall$ v in T_i $\forall$ Schlüssel s in v:

$$\begin{cases} s \leq s_i & \text{falls } i = 1 \\ s_{i-1} < s \leq s_i & \text{falls } 1 < i < \ell \\ s_{\ell-1} < s & \text{falls } i = \ell \end{cases}$$

<u>Zugriff (a, S)</u>:

Vergleiche a mit den Schlüsseln in der Wurzel w. Sei r die Anzahl der Söhne von w. Sei s_j der kleinste Schlüssel in w mit $a \leq s_j$, falls solcher existiert. Dann gilt:

$a = s_j$: Bei knotenorientierter Speicherung ist die Information gefunden. Bei blattorientierter Speicherung suchen wir in T_j weiter.

$a < s_j$: Wir suchen in T_j weiter.

$a > s_r$: D.h., s_j existiert nicht. Wir suchen in T_r weiter.

Falls der betrachtete Knoten ein Blatt ist und nicht den Schlüssel enthält, dann ist a nicht im Suchbaum gespeichert.

Wir behandeln nun die Operationen Einfügen und Streichen in B-Bäumen mit knotenorientierter Speicherung

<u>Einfügen (a, S)</u>:

- Führe Zugriff (a, S) aus.
  - Falls $a \in S$, dann wird der Schlüssel a gefunden und muss nicht noch einmal eingefügt werden.
  - Andernfalls endet die Suche in einem Blatt b. Seien

- $v = \text{Vater}(b)$,
- ℓ die Anzahl der Söhne von v und
- s_j kleinster Schlüssel in v mit $a < s_j$, falls solcher existiert.

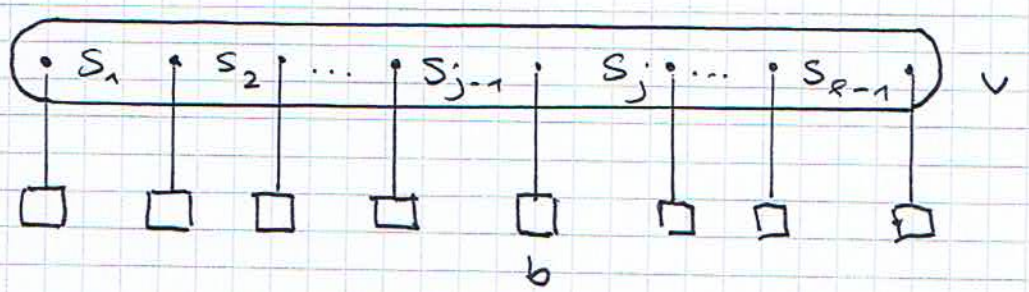
$\rightsquigarrow$

v erhält neuen Schlüssel a und ein zu a korrespondierendes neues Blatt b', wobei a

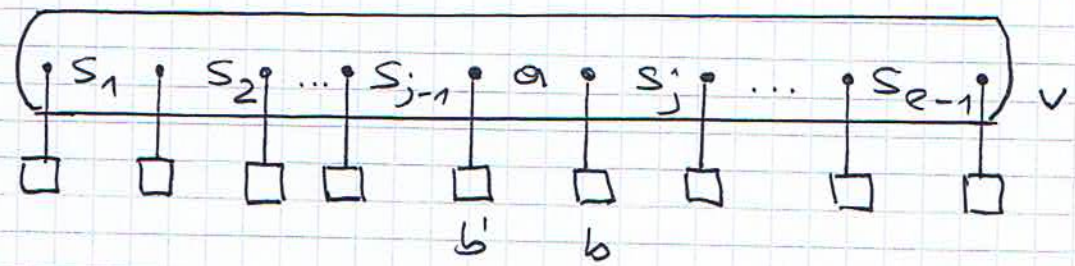
$$\begin{cases} \text{links von } s_1 & \text{falls } j = 1 \\ \text{zwischen } s_{j-1} \text{ und } s_j & \text{falls } 1 < j \leq \ell - 1 \\ \text{rechts von } s_{\ell-1} & \text{falls } s_j \text{ nicht existiert} \end{cases}$$

plaziert wird.

$\rightsquigarrow$ Fall $1 < j \leq \ell - 1$



$\rightsquigarrow$



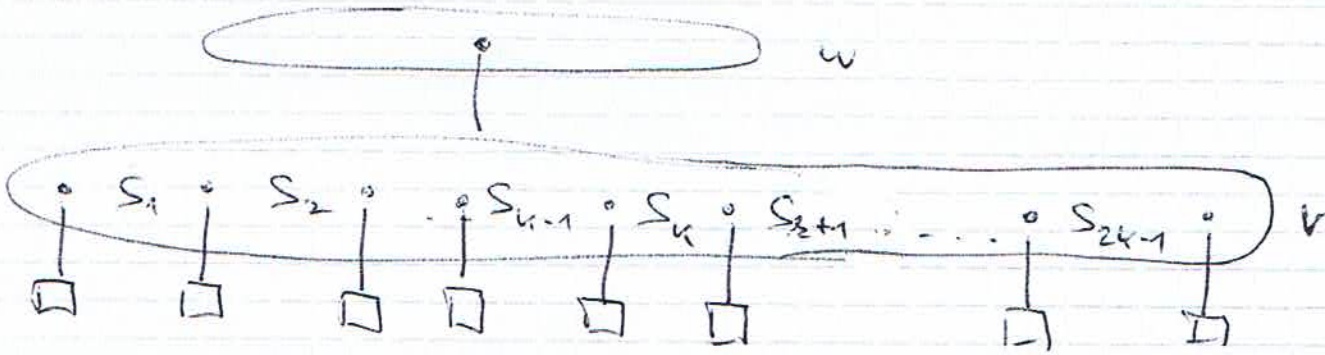
<u>$\ell < 2k-1$</u>: Dann hat v nach Einfügen(a, S) maximal $2k-1$ Söhne. ✓

<u>$\ell = 2k-1$</u>: Dann hat v nach Einfügen(a, S) exakt $2k$ Söhne.

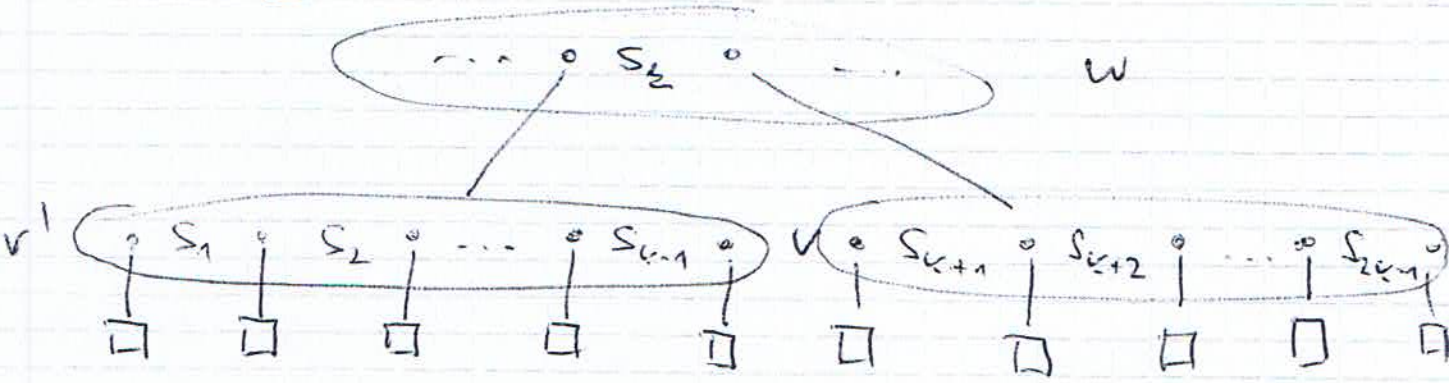
$\Longrightarrow$

Umstrukturierung ist notwendig.

<u>Idee</u>: Aufteilen von v in der Mitte.



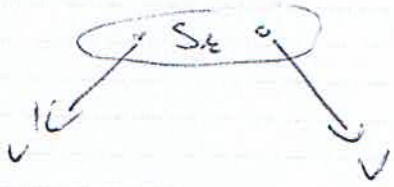
$\rightsquigarrow$



<u>Effekt</u>:

- v und v' haben k Söhne ✓
- w hat einen Sohn mehr als vorher

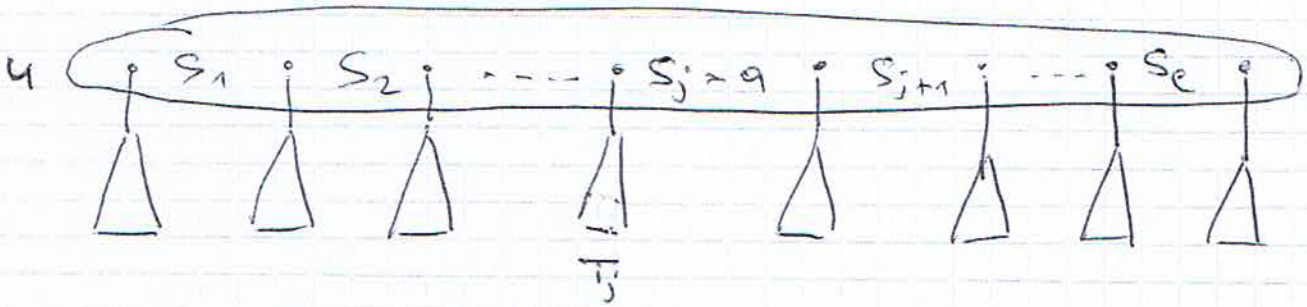
  $\rightsquigarrow$
  analoge Betrachtung für w
- Falls v keinen Vater in T besitzt, also die Wurzel ist, dann kreieren wir eine neue Wurzel w



✓

Streichen(a, S):

- Führe Zugriff (a, S) aus.
  - Falls $a \notin S$, dann endet die Zugriffsoperation erfolglos und die Streichoperation ist beendet.
  - Andernfalls endet die Suche in einem Knoten u, der den Schlüssel $s_j = a$ enthält.



Sei

- v Vater des am weitesten rechts stehenden Blattes in T_j

<u>v = u</u>: D.h. T_j besteht nur aus einem Blatt:

- entferne T_j und s_j ersatzlos

$\rightsquigarrow$ Anzahl der Söhne von v verringert sich um 1.

<u>v ≠ u</u>:

Dann ist der letzte Schlüssel s in v oder

größte Schlüssel in T_j.

- Ersetze in u s_j durch s
- Streiche in v s ersatzlos und entferne das Blatt b.

$\rightsquigarrow$

Anzahl der Söhne von v verringert sich um 1. Oberhalb von v ändert sich die Struktur des Baumes nicht.

Sei ℓ die Anzahl der Söhne von v nach Durchführung der Streichoperation.

- $\ell \geq k$ oder $\ell \geq 2$ und v Wurzel des resultierenden Baumes T'. ✓
- $\ell = k-1$ und $v \neq$ Wurzel(T')

$\Longrightarrow$

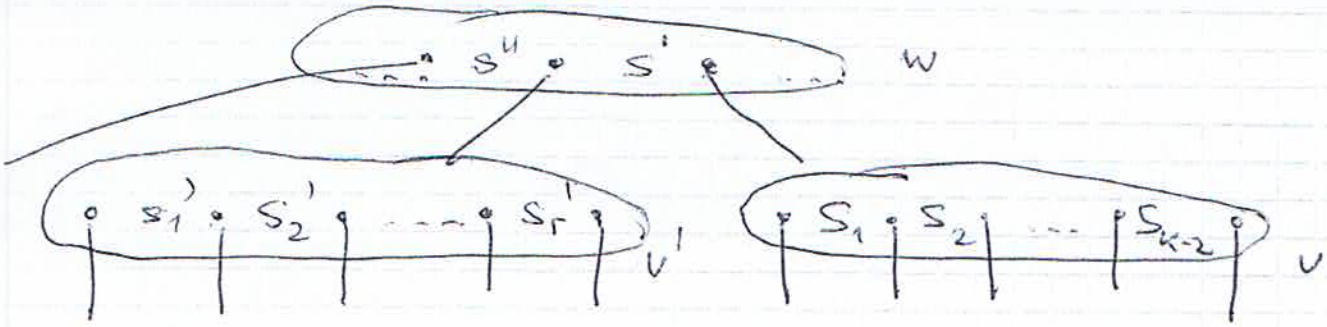
Umstrukturierung ist notwendig

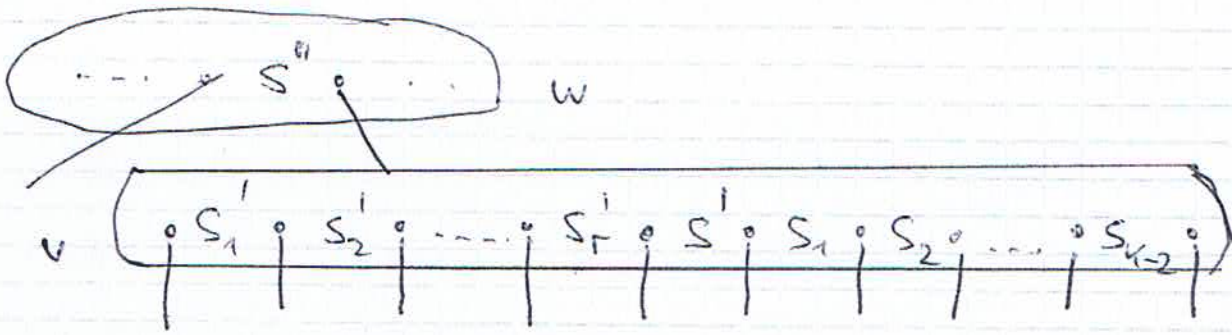
Dann gilt:

v besitzt mindestens einen Bruder v', der unmittelbar neben v in T' steht.

O.B.d.A. sei v' links von v in T'

<u>Idee</u>: Verschmelzung von v und v' zu einem Knoten.





<u>Effekt:</u>

- w hat einen Sohn weniger
- $2k-1 \leq \underbrace{(r'+1)+(k-1)}_{m} \leq 3k-2$

<u>$m = 2k-1$:</u>

$\rightsquigarrow$ v in Ordnung. Analoge Betrachtung für w.

<u>$m > 2k-1$:</u>

$\rightsquigarrow$

Aufteilung von v in der Mitte.

$\rightsquigarrow$

- resultierende Knoten in Ordnung
- w hat wieder einen Sohn weniger ✓

- $\ell = 1$

Dann ist v die Wurzel des resultierenden Baumes T' und enthält keinen Schlüssel. Wir streichen nun v. Der einzige Sohn von v ist die Wurzel des Baumes und der Streichvorgang ist beendet.

O-Schreibweise einführen. (auch Ω und Θ) bereits in ADIP passiert!

Aufwand:

- pro inneren Knoten: $O(k)$
- Anzahl der betrachteten inneren Knoten: $O(\log_k n)$

Satz 1.5

Die Operationen Zugriff, Einfügen und Streichen können in B-Bäumen der Ordnung k in Zeit $O(k \cdot \log_k n)$, wobei n die Anzahl der im Baum gespeicherten Schlüssel ist, durchgeführt werden.

1.2.4 Tries

Universum $U = \Sigma^{\leq k}$, $|\Sigma| = k$

Ein Trie bezüglich dem Alphabet Σ ist ein Baum $T = (V, E)$, für den gilt:

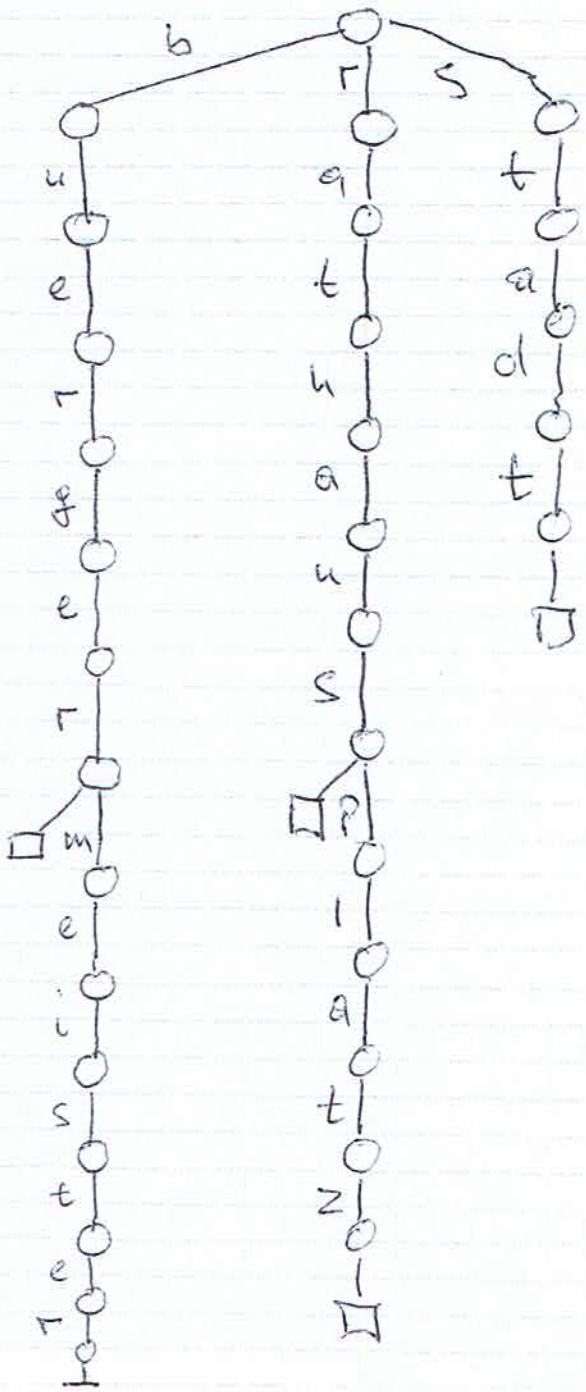
1. Jeder innere Knoten hat Ausgangsgrad $\leq k$.

2. Die ausgehenden Kanten eines inneren Knotens sind mit paarweise verschiedenen Elementen von Σ beschriftet.

<u>Beispiel:</u>

$\Sigma = \{a, b, c, \ldots, z\}$

$S = \{$ buerger, buergermeister, rathaus, rathausplatz, stadt $\}$



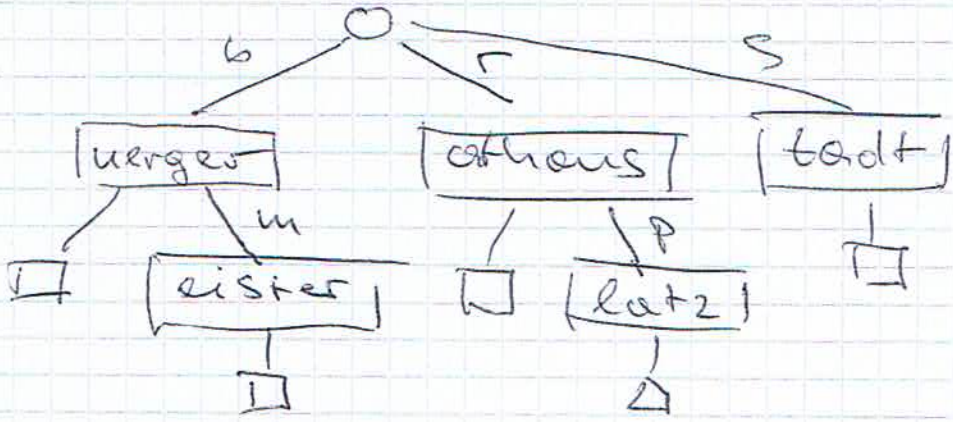
Menge der Pfade von der Wurzel zu einem Blatt

$\overset{1}{=}$

Menge der Schlüssel in S.

Kantenbeschriftungen dieser Pfade korrespondieren zu den Schlüsseln.

- präfixfrei (denn können Kanten zu Blättern Markierung erhalten)
- Komprimierung.

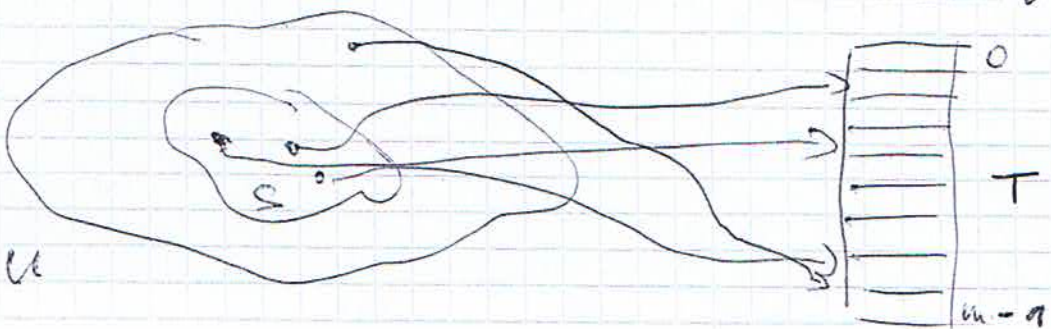


1.3 Hashing

- erfordert nicht, daß Menge geordnet ist.

<u>Hashing</u> benötigt

- ein Feld $T[0:m-1]$, die <u>Hashtafel</u>
- eine Funktion $h: U \longmapsto \{0,1,\dots,m-1\}$



Sei $S \subseteq U$ die zu verwaltende Menge

<u>Ziel:</u>

Jedes Element $x \in S$ in $T[h(x)]$ zu speichern.

<u>Beispiel:</u>

$m = 5$, $S = \{3, 15, 22, 24\}$

$h(x) = x \bmod 5$

0	15
1	
2	22
3	3
4	24

Kollision, falls 22 durch 20 ersetzt wird.

□

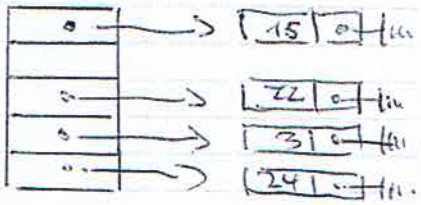
<u>Problem:</u>

Was tun, falls $h(x) = h(y)$ für $x, y \in S$, $x \neq y$?

! Dieses Ereignis heißt <u>Kollision</u>

1.3.1 <u>Kollisionsbehandlung mittels verketteter Listen</u>

Hashtafel ist ein Feld von linearen Listen



~~<u>Modifikation:</u>~~

~~1. Feldkomponente wird in Tafel gespeichert.~~

<u>Zugriff (x, S):</u>

(1) Berechne $h(x)$;

(2) Besuche das Element x in der Liste $T[h(x)]$.

<u>Einfügen (x, S):</u>

(1) Berechne $h(x)$;

(2) Füge x in die Liste $T[h(x)]$ ein.

<u>Streichen (x, S):</u>

(1) Berechne $h(x)$.

(2) Streiche x aus der Liste $T[h(x)]$.

Aufwand hierfür:

1.) Kosten zur Berechnung des Funktionswertes $h(x)$

2) Zeit für die Durchmusterung der Liste $T[h(x)]$

<u>Annahme:</u>

h ist derart einfach, daß $h(x)$ in konstanter Zeit berechnet werden kann.

$1 + \delta_h(x, S)$ Kosten für eine Operation bzgl. Schlüssel x,

wobei

$$\delta_h(x, S) = \sum_{y \in S} \delta_h(x, y) \quad \text{mit}$$

$$\delta_h(x, y) = \begin{cases} 1 & \text{falls } h(x) = h(y) \text{ und } x \neq y \\ 0 & \text{sonst} \end{cases}$$

<u>ungünstigster Fall:</u>

$h(x) = i_0 \quad \forall x \in S$ für eine Konstante i_0

<u>Satz 1.6</u>

Die Operationen Zugriff(a,S) und Streichen(a,S) benötigen im worst case $\Omega(|S|)$ Zeit.

<u>Ziel:</u>

Analyse der erwarteten benötigten Zeit einer Folge von n Einfüge-, Streiche- und Zugriffsoperationen bei anfangs leerer Hashtafel unter folgenden Annahmen:

1. Die Hashfunktion $h: U \to [0..m-1]$ streut das Universum gleichmäßig über das Intervall $[0..m-1]$. D.h.,

   $\forall i,j \in [0..m-1]$ gilt: $\big|\, |h^{-1}(i)| - |h^{-1}(j)| \,\big| \leq 1.$

2. Sämtliche Elemente des Universums sind mit gleicher Wahrscheinlichkeit Argument der nächsten Operation. D.h., für alle $x \in U$ und $k \geq 1$ ist x mit Wahrscheinlichkeit $\frac{1}{|U|}$ Argument der k-ten Operation.

Sei

x_k das Argument der k-ten Operation.

Dann folgt aus obigen Annahmen

$$\text{prob}(h(x_z) = i) = \frac{1}{m} \quad \forall\, i \in \{0, 1, \dots, m-1\}$$

Wahrscheinlichkeit für $h(x_z) = i$.

<u>Satz 1.7</u>

Unter obigen Annahmen ist der Erwartungswert für die von einer Folge von n Einfüge-, Streiche- und Zugriffsoperationen benötigte Zeit $T(n,m) \leq (1 + \beta/2) \cdot n$, wobei $\beta := n/m$ der maximal mögliche <u>Belegungsfaktor</u> der Hashtafel ist.

<u>Beweis</u>:

<u>Beobachtung</u>:
Falls wir jeder Liste i, $0 \leq i \leq m$ einen Zähler $\ell(i)$ zuordnen, so dass stets der Wert des Zählers $\ell(i)$ größer gleich der Länge der Liste i ist und wir, falls eine Operation die i-te Liste betrifft, als Aufwand $1 + \ell(i)$ zählen, dann ist der gezählte Wert stets mindestens so groß wie die benötigte Zeit.

<u>Start</u>: $\ell(i) := 0 \quad \forall\, i \in \{0, 1, \dots, m-1\}$

<u>Regel</u>:
Immer wenn eine Operation die i-te Liste betrifft, addieren wir unabhängig vom Typ der Operation eine Einheit auf den Zähler $\ell(i)$.

$\Rightarrow$

$\ell(i)$ ist stets eine obere Schranke für die Länge der i-ten Liste.

Zunächst berechnen wir den erwarteten gezählten Wert EC_{k+1} der $(k+1)$-ten Operation.

<u>Annahme</u>: $h(x_{k+1}) = i$.

Bezeichne $\text{prob}(\ell_k(i) = j)$ die Wahrscheinlichkeit, dass der Zähler $\ell(i)$ nach der k-ten Operation den Wert j hat. Dann gilt

$$EC_{k+1} = 1 + \sum_{j \geq 0} \text{prob}(\ell_k(i) = j) \cdot j.$$

Für $j \geq 1$ gilt

$$\text{prob}(\ell_k(i) = j) = \binom{k}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{k-j}.$$

Also gilt

$$\begin{aligned} EC_{k+1} &= 1 + \sum_{j=0}^{k} \binom{k}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{k-j} \cdot j \\ &= 1 + \frac{k}{m} \sum_{j=1}^{k} \binom{k-1}{j-1} \left(\frac{1}{m}\right)^{j-1} \left(1 - \frac{1}{m}\right)^{(k-1)-(j-1)} \\ &= 1 + \frac{k}{m} \left(\frac{1}{m} + \left(1 - \frac{1}{m}\right)\right)^{k-1} \quad \text{(Binomischer Lehrsatz)} \\ &= 1 + \frac{k}{m}. \end{aligned}$$

Insgesamt ergibt sich somit für den erwarteten gezählten Wert $EC(n)$ von n Einfüge-, Streiche- und Zugriffsoperationen folgende Abschätzung:

$$
\begin{aligned}
EC(n) &= \sum_{k=1}^{n} EC_k \\
&= \sum_{k=0}^{n-1} \left(1 + \frac{k}{m}\right) \\
&= n + \frac{1}{m} \cdot \sum_{k=0}^{n-1} k \\
&= n + \frac{1}{m} \cdot \frac{n(n-1)}{2} \\
&= \left(1 + \frac{n-1}{2m}\right) \cdot n \\
&< \left(1 + \frac{1}{2} \cdot \frac{n}{m}\right) \cdot n \\
&= \left(1 + \mathcal{F}/2\right) \cdot n .
\end{aligned}
$$

$\blacksquare$

<u>Frage:</u>

Wie realistisch sind die Annahmen 1 und 2?

Annahme 1:

Kann leicht durch Verwendung der Divisionsmethode sicher gestellt werden:

$h: U \mapsto [0..m-1]$, wobei
$h(x) = x \bmod m$.

Annahme 2:

kritisch.

Bemerkung:

- erwartetes Verhalten hängt vom Belegungsfaktor $\beta = \frac{n}{m}$ ab.

  β durch Konstante beschränkt $\Rightarrow$ erwartetes Verhalten ist konstant.

Frage: Kann dies sicher gestellt werden?

Antwort: Mittels Anpassung von m.

- Folge $T_0, T_1, T_2, \ldots, T_i, \ldots$ von Hashtafeln der Größe $m, 2m, \ldots, 2^i \cdot m, \ldots$
- Für jedes i Hashfunktion $h_i : U \mapsto [0..2^i m - 1]$.

Ziel:

Verwendung der Hashtafeln, so daß stets $\beta \in (\frac{1}{4}, 1)$.

<u>Annahme:</u> T_i aktuelle Hashtafel.

- β wird zu 1:

  Umspeichern $T_i \to T_{i+1}$.

  <u>Zeit</u>: $O(2^i m)$

  <u>Belegungsfaktor</u> von T_{i+1}: $1/2$

  $\Rightarrow$

  $\geq 1/4 \cdot 2^{i+1} m$ Operationen vor nächster Umspeicherung.

- β wird zu $1/4$:

  Umspeichern $T_i \to T_{i-1}$ (falls $i > 0$).

  <u>Zeit</u>: $O(\frac{1}{4} \cdot 2^i m)$

  <u>Belegungsfaktor</u> von T_{i-1}: $1/2$

  $\Rightarrow$

  $\geq \frac{1}{4} \cdot 2^{i-1} m$ Operationen vor nächster Umspeicherung.

<u>Beobachtung:</u>

- In beiden Fällen gilt:

  Kosten Umspeicherung $\leq O(2 \times$ # Operationen seit der letzten US$)$.

$\Rightarrow$ Verteilung der Umspeicherungskosten auf einzelne Operationen impliziert $O(1)$ erwartete mittleren Kosten.

- Falls $i = 0$, dann kann $\beta \leq 1/4$ sein.

1.3.2 Kollisionsbehandlung mittels offener Adressierung

Idee:

Alle Elemente werden in der Hashtafel gespeichert. Hierzu definiert jedes Element $x \in U$ eine Folge $h(x,i)$, $i = 0, 1, 2, \ldots$ von Tafelpositionen. Durchmusterung dieser Folge, wenn eine Operation bzgl. des Schlüssels x durchgeführt wird.

Probleme:

- Organisation der Operation Zugriff(a, S)

  Frage: Wann terminiert diese Operation im Fall der erfolglosen Suche?

  - Nach Beendigung der gesamten Probefolge

    $\leadsto$ benötigte Zeit ist stets $\Omega(m)$

  - Nach Betrachtung einer leeren Tafelposition.

    $\leadsto$ Durchführung von Streiche(a, S) ist schwierig.

mögliche Lösung:

- anstatt a aus T zu entfernen, wird betreffende Position mit "gestrichen" über-

schreiben. Eine Einfügeoperation behandelt solche Tafelposition als wäre sie leer.

Methoden zur Konstruktion der Probefolge

Sei $h' : U \to \{0, 1, \ldots, m-1\}$ eine Hashfunktion.

1) Lineares Sondieren

Probefolge für $x \in U$:

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

Vorteil: leicht zu implementieren

Nachteil: Lange Cluster haben stärkere Tendenz zu wachsen, als kurze Cluster.
(primäre Häufung)

2) Quadratisches Sondieren

Probefolge für $x \in U$:

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

wobei c_1 und $c_2 \neq 0$ Konstanten sind.

Nachteil:
Für $x, y \in U$ mit $h(x,0) = h(y,0)$ sind die Probefolgen identisch.
(sekundäre Häufung)

3) <u>Double Hashing</u>

$h_1 : U \to [0..m-1]$, $h_2 : U \to [0..m-2]$
Hashfunktionen.

Probefolge für $x \in U$:

$$h(x,i) = (h_1(x) + i\, h_2(x)) \bmod m.$$

<u>Vorteil</u>:
Probefolge hängt auf zwei Arten und Weisen vom Schlüssel x ab.

<u>aber</u>:

- m und $h_2(x)$ müssen zueinander prim sein, damit die Probefolge die gesamte Hashtafel besucht.

  <u>einfache Realisierung</u>:

  - Wähle m als Potenz von 2 und $h_2(x)$ ungerade.

- Da verschiedene $(h_1(x), h_2(x))$-Paare zu verschiedenen Probefolgen führen, ist die Anzahl der verwendeten Probefolgen quadratisch in m. (im Gegensatz zu linearen und quadratischen Sondierung, wo diese Anzahl gleich m ist.)

1.4 Datenstrukturen für disjunkte Mengen

gegeben: n paarweise disjunkte einelementige Mengen $S_1, S_2, \ldots, S_n$.

O.B.d.A sei $S_i = \{i\}$, $1 \leq i \leq n$.

Operationen:

FIND (x): Bestimme den Namen derjenigen Menge, die das Element x enthält.

UNION(A,B,C): Vereinige die Mengen A und B und gib der neuen Menge den Namen C.

Aufgabe:

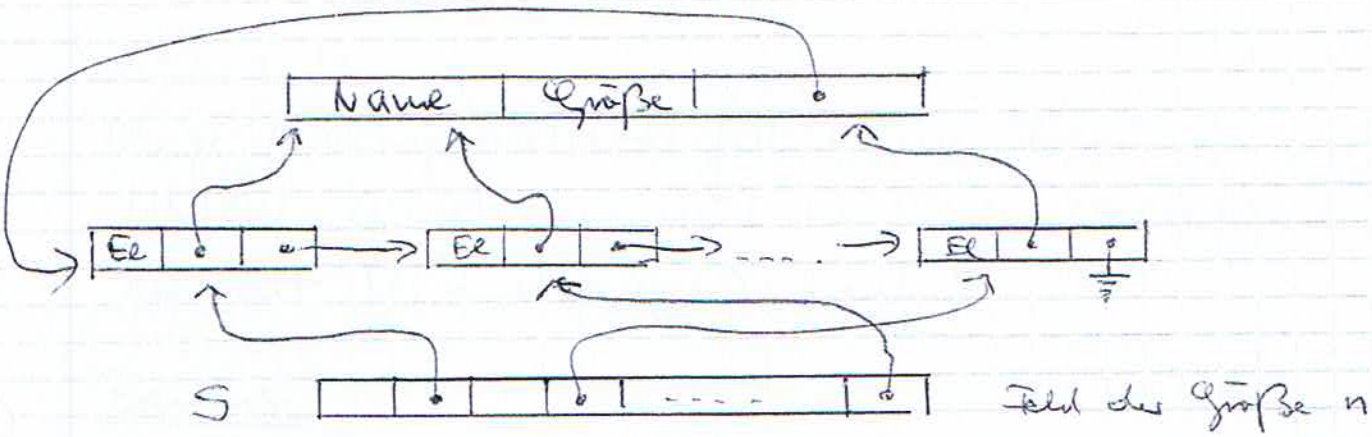
Durchführung einer Folge von UNION- und FIND-Operationen.

Mögliche Randbedingungen:

1) Die Operationen werden on-line durchgeführt. D.h., jede Operation muss vollständig durchgeführt sein, bevor die nächste Operation bekannt gegeben wird.

2. Die UNION-Operationen sind im voraus bekannt.

<u>Datenstruktur:</u>



- S[i] Pointer auf Record, der i enthält.

<u>FIND(i)</u>: Greife über S[i] auf denjenigen Record zu, der i enthält. Über diesen Record bestimme denjenigen Record, der den Mengennamen enthält.

UNION(A,B,C): <u>Annahme</u>: Records, die die Mengennamen A und B enthalten, sind bekannt. O.B.d.A. sei $|A| \geq |B|$.

(1) Ändere jeden Zeiger auf den Namensrecord der Menge B in einen Zeiger auf den Namensrecord der Menge A.

(2) Hänge die verkettete Liste der Menge A an das Ende der verketteten Liste der Menge B. Addiere die Größe der Menge A und die Größe der Menge B und ersetze die Größen kom-