Kapitel 2

Selbstorganisierende Datenstrukturen

2.1 Lineare Listen

In diesem Abschnitt werden wir lineare Listen zur Lösung des Wörterbuchproblems verwenden (Abbildung 2.1). Dabei gehen wir davon aus, daß jedes Element höchstens einmal vorkommt.

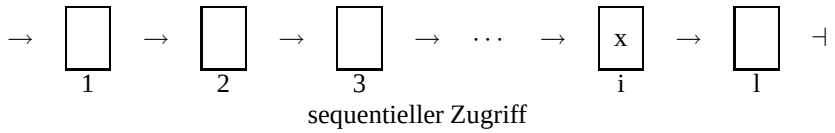

Abbildung 2.1: Modell: Lineare Liste

Statisch:

ACCESS(x): Kosten:
i , falls x an Position i steht
$l+1$, falls x nicht in der Liste vorkommt
Man darf die Liste umorganisieren (um sie für künftige Anfragen effizienter zu machen:

erlaubt:

1. Nach erfolgreichem Zugriff auf x darf x um eine beliebige Anzahl von Positionen nach vorn gebracht werden.

   Kosten $= 0$. (Argument: schlimmstenfalls muss man einfach nochmal den Weg zurück nach vorn laufen, den man eben von vorne zum x hin gelaufen ist $\Rightarrow$ Faktor 2).

   „Kostenfreie Vertauschung“.

2. Man darf jederzeit und überall zwei benachbarte Listenelemente vertauschen.

   Kosten $= 1$. (Die Kosten, um erstmal zu den zu vertauschenden Elementen zu kommen, werden nicht gezählt).

   „Kostenpflichtige Vertauschungen“.

Anmerkung: Die Kosten für diese Tauschoperationen sind relativ willkürlich definiert.
Unsere Algorithmen werden (2) nicht anwenden, aber der optimale Algorithmus, also unser Gegner, darf es.

Dynamisch:

Zusätzliche Operationen:

INSERT(x): Kosten $l + 1$ (eingefügt wird hinten wegen Test auf doppeltes Vorkommen $\Rightarrow$ ganze Liste wird einmal durchlaufen).

DELETE(x): Kosten i bzw. $l + 1$ (wie bei ACCESS(x)).

Situation:

Liste der Länge l gegeben. Es kommt eine Folge σ von n Zugriffen auf die Liste.
OPT kennt σ im Voraus und kann die Liste entsprechend organisieren.
Wie? NP-vollständig (Ambühl, 2000).

Bemerkung: Es gibt Fälle, in denen OPT kostenpflichtige Vertauschungen benötigt, um optimal zu bleiben (Übungsaufgabe).

ALG erhält jeweils nur die nächste Anforderung in der Folge $\sigma = \sigma_1\sigma_2 \ldots \sigma_i\sigma_{i+1}$ und muß sofort reagieren.

Mögliche Kandidaten für ALG:

TRANS (Transpose): Bringe x nach erfolgreichem ACCESS(x) (oder INSERT(x)) um **eine** Position weiter nach vorn.

(vorsichtig)

MTF (Move to front): Bringe x nach erfolgreichem ACCESS(x) (oder INSERT(x)) an den Listenanfang.

(energisch)

FC (Frequency Count): Führt Buch über die erfolgreichen Zugriffe; ordnet Liste entsprechend an.

(gewissenhaft, aber aufwendig zu implementieren)

Theorem 2.1.1 (Sleator, Tarjan '85: erstes Paper zu Online-Analyse):
MTF ist $\underbrace{\left(2 - \frac{1}{l+1}\right)}_{\leq 1}$ - kompetitiv bei maximaler Listenlänge l.

Beweis (Theorem 2.1.1): Unser Beweis benutzt zwei Techniken:

- Betrachtung der amortisierten Kosten
- Darstellung der amortisierten Kosten durch Potentialfunktion

Idee: Man investiert am Anfang etwas, was sich später auszahlt.
MTF und OPT starten mit derselben Liste der Länge l, müssen dieselbe Folge $\sigma = \sigma_1\sigma_2 \ldots$ bedienen, aber auf unterschiedliche Art.
Sei LOPT_i die Liste von OPT nach Bearbeitung von σ_i
Sei LMTF_i die Liste von MFT nach Bearbeitung von σ_i

Def: Amortisierte Kosten von MTF bei Bearbeitung von σ_i:

$$a_i := t_i + \phi_i - \phi_{i-1},$$

wobei $t_i =$ „echte“ Kosten (= reine Suchkosten) von MTF bei Bearbeitung von σ_i.
Potentialfunktion ϕ_i: Mißt die „Ähnlichkeit“ von LOPT_i und LMTF_i, genauer:

$$\phi_i = \text{Anzahl } \mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_i) \text{ der Inversionen der beiden Listen, d.h.:}$$

$$\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_i) := \{(x, y), x \text{ steht in } \mathrm{LMTF}_i \text{ vor } y, \text{ in } \mathrm{LOPT}_i \text{ hinter } y\}$$

Beispiel:

LMTF_i :	3	15	8	4	7	1	
LOPT_i :	8	1	4	7	15	3	

$\Rightarrow$

Eintrag	Anzahl Inversionen
3	5
15	4
4	1
7	1

$\Rightarrow$ insgesamt 11 Inversionen
$s_i :=$ reine Suchkosten von OPT bei Bearbeitung von σ_i.
$P_i :=$ Anzahl kostenpflichtiger Vertauschungen von OPT bei Bearbeitung von σ_i.
$F_i :=$ Anzahl kostenfreier Vertauschungen von OPT bei Bearbeitung von σ_i.
Um weiterzukommen, benötigen wir folgendes

Lemma 2.1.2:

$$a_i \leq (2s_i - 1) + P_i - F_i$$

Beweis (Lemma 2.1.2):

Methode:

$$a_i = \overbrace{\underbrace{t_i}_{=k} + \mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_{i-1}) - \mathrm{Inv}(\mathrm{LMTF}_{i-1}, \mathrm{LOPT}_{i-1})}^{\leq 2j-1=2S_i-1}$$
$$+ \underbrace{\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_i) - \mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_{i-1})}_{\leq P_i - F_i}$$

Fall 1: $\sigma_i = \mathrm{ACCESS}(x_j)$, erfolgreich.

Bemerkung: Daß x_j an Postition j steht, stellt keine Einschränkung dar.

Sei $\nu :=$ Anzahl der Elemente, die in LMTF_{i-1} **vor** x_j stehen und in LOPT_{i-1} **dahinter**. (Abbildung 2.2)
$\Rightarrow$ Von den $k-1$ Elementen, die in LMTF_{i-1} vor x_j stehen, stehen auch in LOPT_{i-1} $k-1-\nu$ vor x_j.
$\Rightarrow k - 1 - \nu \leq j - 1$
$\Rightarrow \boxed{k - \nu \leq j}$ (= Suchkosten von OPT)

Vorstellung: Erst bearbeitet MTF die Anforderung σ_i, dann OPT.

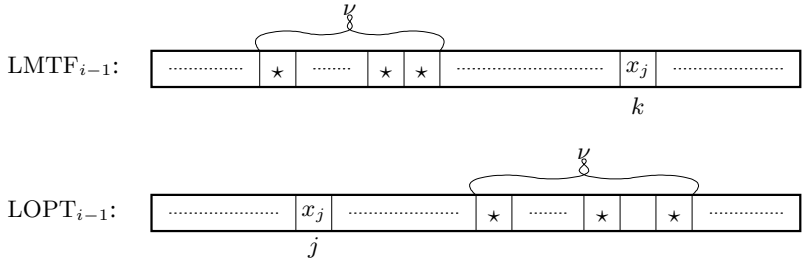

Abbildung 2.2: Situation vor Ausführung von σ_i:

MTF:

- Suchkosten $t_i = k$
- Dadurch, daß x_j ganz nach vorn kommt:
  ν alte Inversionen verschwinden
  $k - 1 - \nu$ neue Inversionen entstehen

$$\Rightarrow \#\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_{i-1}) - \#\mathrm{Inv}(LMTF_{i-1}, LOPT_{i-1}) = k - 1 - 2\nu$$
$$\Rightarrow t_i + k - 1 - 2\nu = 2(k - \nu) - 1 \leq 2j - 1 \quad \text{Zwischenbilanz}$$

OPT:

- Suchkosten $s_i = j$
- Jede freie Vertauschung durch OPT beseitigt eine Inversion (da x_j in LMTF_i ganz vorn steht).
- Jede kostenpflichtige Vertauschung durch OPT schafft ≤ 1 neue Inversion.

$$\Rightarrow \#\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_i) \leq \#\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_{i-1}) + P_i - F_i$$

$$\left.\begin{aligned} \Rightarrow a_i &= t_i + \phi_i - \phi_{i-1} \\ &= t_i + \#\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_i) - \#\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_{i-1}) \\ &\quad + \#\mathrm{Inv}(\mathrm{LMTF}_i, \mathrm{LOPT}_{i-1}) - \#\mathrm{Inv}(\mathrm{LMTF}_{i-1}, \mathrm{LOPT}_{i-1}) \\ &\leq 2j - 1 + P_i - F_i \\ &= 2s_i - 1 + P_i - F_i \end{aligned}\right\}$$

Fall 2: $\sigma_i = ACCESS(x_j)$ nicht erfolgreich
$\Rightarrow$ weder MTF noch OPT machen freie Vertauschungen
Müssen zeigen:

$$a_i = t_i + \underbrace{\phi_i - \phi_i - 1}_{\leq P_i} \leq \underbrace{2s_i - 1}_{=2l+1} + P_i - \underbrace{F_i}_{=0} \surd$$

Fall 3: $\sigma_i = DELETE(x_j)$ erfolgreich
$\Rightarrow$ keine freien Vertauschungen

$$\underbrace{t_i}_{=k} + \underbrace{Inv(LMTF_i, LOPT_i - 1) - Inv(LMTF_i - 1, LOPT_i - 1)}_{=\nu}$$

$$\left(= j \leq 2j - 1 \overset{(j \geq 1)}{=} 2s_i - 1, \text{Rest analog} \right)$$

Fall 4: $\sigma_i = DELETE(x_j)$ erfolglos: wie erfolgloses $\mathrm{ACCESS}(x_j)$

Fall 5: $\sigma_i = \mathrm{INSERT}(x_j)$: wie $\mathrm{ACCESS}(x_j)$mit $k = j = l + 1, \nu = 0$ □

Theorem 2.1.3: Sei σ eine Folge von n Zugriffen (ACCESS, INSERT oder DELETE). Dann gilt bei gleicher Anfangsliste:

$$\mathrm{MTF}(\sigma) \leq 2 \cdot \mathrm{OPT}_s(\sigma) + P - F - n$$

mit $\mathrm{OPT}_s(\sigma) =$ reine Suchkosten von OPT
$P =$ kostenpflichtige Vertauschungen von OPT
$F =$ kostenfreie Vertauschungen von OPT.

Beweis (Theorem 2.1.3):

$$\begin{aligned}
\mathrm{MTF}(\sigma) &= \underbrace{\sum_{i=1}^{n} t_i}_{\text{wollen}} \\
&\leq \sum_{i=1}^{n} t_i + \underbrace{\phi_n}_{\geq 0} - \underbrace{\phi_0}_{=0} \\
&= \sum_{i=1}^{n} (t_i + \phi_i - \phi_{i-1}) \\
&= \underbrace{\sum_{i=1}^{n} a_i}_{\text{können}} \\
&\overset{\text{Lemma 2.1.2}}{\leq} \sum_{i=1}^{n} ((2s_i - 1) + P_i - F_i) \\
&= 2 \cdot \mathrm{OPT}_s(\sigma) + P - F - n
\end{aligned}$$

□

Damit: Weiter im Beweis von Theorem 2.1.1:

$$\mathrm{MTF}(\sigma) \overset{\text{Thm. 2.1.3}}{\leq} 2 \cdot \mathrm{OPT}_s(\sigma) + P - F - n$$

mit $l =$ maximale Listenlänge,
$2 \cdot \mathrm{OPT}_s(\sigma) + P \leq 2 \cdot \mathrm{OPT}(\sigma)$ wegen $\mathrm{OPT}(\sigma) = \mathrm{OPT}_s(\sigma) + P$.

$$\Rightarrow \mathrm{OPT}(\sigma) \leq n \cdot (l + 1)$$

(denn OPT ist sicher besser als der bequeme Algorithmus, der bei jeder Operation bis zum Listenende läuft.)

$$\begin{aligned}
&\Rightarrow n \geq \frac{\mathrm{OPT}(\sigma)}{l+1} \\
&\Rightarrow \mathrm{MTF}(\sigma) \leq \left(2 - \frac{1}{l+1}\right) \mathrm{OPT}(\sigma)
\end{aligned}$$

□

Fragen:

- Ist das gut?
- Ist MTF überreagierend?
- Ist TRANS besser?

Proposition 2.1.4: TRANS ist **nicht** kompetitiv (falls die Listenlänge beliebig ist).

Beweis (Proposition 2.1.4): Sei eine Liste mit l Elementen gegeben.
Der böse Gegenspieler (adversary) ärgert TRANS und fordert stets ACCESS-Operationen für das letzte Element in der Liste LTRANS
$\Rightarrow$ TRANS vertauscht stets die beiden letzten Listenelemente: Abbildung 2.3.

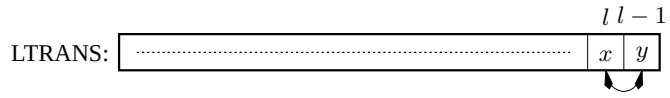

Abbildung 2.3: Liste von TRANS (LTRANS)

$\Rightarrow \mathrm{TRANS}(\sigma) = 2 \cdot n \cdot l$ bei einer Folge von $2n$ Zugriffen.
Der Gegenspieler OPT bringt zunächst x, y nach vorn: Abbildung 2.4.

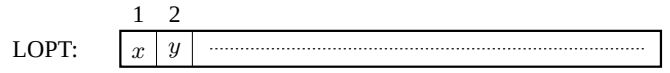

Abbildung 2.4: Liste von OPT (LOPT)

Kosten: $2 \cdot (l-2)$
Danach hat OPT Kosten 3 für je zwei Zugriffe (auf x, y).

$$\Rightarrow \frac{\mathrm{TRANS}(\sigma)}{\mathrm{OPT}(\sigma)} = \frac{2nl}{2(l-2)+3n} \overset{n\to\infty}{\longrightarrow} \frac{2}{3} \cdot l$$

Für großes l ist TRANS also beliebig schlecht. □

Frage: Gibt es andere Algorithmen, die besser sind als MTF?

Theorem 2.1.5: Jeder deterministische Online-Algorithmus für das statische (nur ACCESS) Listenproblem hat einen kompetitiven Faktor $\geq 2 - \frac{2}{l+1}$

Beweis (Theorem 2.1.5): Sei ALG ein solcher Algorithmus.
Wenn der Gegenspieler n mal das letzte Element verlangt:

$$\mathrm{ALG}(\sigma) \geq n \cdot l$$

Frage: Wie gut kann OPT die Folge σ bedienen?
Trick: Sei π eine beliebige Permutation der l Elemente.
Definiere Algorithmus A_π wie folgt:

- Stelle zunächst Permutation π her: Kosten $b \cdot l^2$.
- Beantworte dann alle ACCESS-Operationen **ohne** weitere Umstrukturierung.

Betrachte eine einzelne Anforderung ACCESS(x):
Es gibt $(l-1)!$ Permutationen π, bei denen x an Stelle i steht.

$$\begin{aligned}
\sum_i A_\pi(\mathrm{ACCESS}(x)) &= \sum_{i=1}^{l} i \cdot (l-1)! \\
&= (l-1)! \frac{l(l+1)}{2} \\
\Rightarrow \qquad \sum_\pi A_\pi(\sigma) &= n \cdot (l-1)! \cdot \frac{l(l+1)}{2} + l! \cdot bl^2 \\
\text{Mittelwert:} \qquad \frac{1}{l!} \sum_\pi A_\pi(\sigma) &= \frac{n(l+1)}{2} + bl^2
\end{aligned}$$

Trick: Es muß mindestens ein π_0 geben mit $A_{\pi_0}(\sigma) \leq$ Mittelwert.

$$\begin{aligned}
\Rightarrow \frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} &\geq \frac{nl}{\mathrm{OPT}(\sigma)} \\
&\geq \frac{nl}{A_{\pi_0}(\sigma)} \\
&\geq \frac{nl}{\frac{n(l+1)}{2} + bl^2} \\
&\overset{n\to\infty}{\longrightarrow} \frac{2l}{l+1} \\
&= 2\frac{2}{l+1}
\end{aligned}$$

□

Rekapitulation:

- Selbst im statischen Fall ist die Berechnung von OPT NP-vollständig (ohne Beweis).
- *MoveToFront* ist 2-kompetitiv in dynamischen selbstorganisierenden Listen.
- Eine bessere *deterministische* Online-Lösung gibt es nicht.
- Auch in der Offline-Situation ist keine bessere Approximation von OPT (in polynomieller Laufzeit) als 2 bekannt.
- *Transpose* ist nicht kompetitiv.

Frage: Hilft Randomisierung? Folgendes Modell:

Online-Algorithmus ("Spieler"): Trifft zur Laufzeit zufällige Entscheidungen nach einer bestimmten Wahrscheinlichkeitsverteilung (siehe Abbildung 2.5).

Vergeßlicher Gegenspieler (oblivious adversary): Trifft seine Entscheidungen vor dem Start

- in Kenntnis des Algorithmus inklusive der Wahrscheinlichkeitsverteilungen
- ohne Kenntnis der konkreten Entscheidungen das Spielers

Abrechnung: Zu erwartende Kosten des Online-Algorithmus gegen Kosten einer optimalen Offline-Lösung.

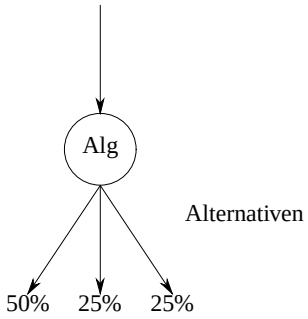

Abbildung 2.5: Zufallsgesteurte Entscheidung

Definition 2.1.1: Eine randomisierter Online-Algorithmus ALG heißt c-kompetitiv gegen vergeßliche Gegenspieler, falls gilt: $\exists A : [\forall P \in \Pi : [\underbrace{E(\mathrm{ALG}(P))}_{\text{Erwartungswert}} \leq c \cdot \mathrm{OPT}(P) + A]]$

Jetzt: Dynamische Listenorganisation mit randomisiertem Algorithmus BIT.

Statischer Fall: Nur Zugriffe $\mathrm{ACCESS}(x)$:

- Rate anfangs für jedes Listenelement x ein Zufallsbit $b(x)$. i.i.d, d.h.:
  - gleichwahrscheinlich 0 oder 1
  - unabhängig
- Zur Laufzeit (deterministisch) bei $\mathrm{ACCESS}(x)$:
  - komplementiere $b(x)$ (d.h.: $b(x) := 1 -^{\cdot} b(x)$)
  - falls $b(x)$ jetzt $= 1$: bringe x an Listenanfang (Kostenfreie Vertauschungen; es bleibt dabei $b(x) = 1$)

Theorem 2.1.6 (Reingold, Westbrook '90): Für jede Zugriffsfolge σ der Länge n gilt:

$$E(\mathrm{BIT}(\sigma)) \leq \underbrace{\frac{7}{4}}_{<2} \cdot \mathrm{OPT}(\sigma) - \frac{3}{4}n$$

Beweis (Theorem 2.1.6): Sei σ eine feste Folge von n ACCESS-Operationen. Während BIT die Folge σ bearbeitet, sind auch die aktuellen Werte $b(x)$ i.i.d.

Betrachte zwei Typen von Events:

1. OPT führt kostenpflichtige Vertauschung aus
2. BIT und OPT beantworten $\mathrm{ACCESS}(y)$ mit kostenfreien Vertauschungen

Wie vorher: Potentialfunktion ϕ_i, mißt das Verhältnis der Listen LBIT_i, LOPT_i nach Bearbeitung des i-ten Events.

Intuitiv klar: ϕ muß auch die Bits $b(x)$ berücksichtigen!

Definition 2.1.2:

1. Sei (x, y) eine Inversion von LBIT_i bzgl. LOPT_i (d.h.: in LBIT_i steht x vor y, aber in LOPT_i steht x hinter y). Dann heißt

$$w(x, y) := b(y) + 1 = \#\mathrm{ACCESS}(y), \text{ bevor } y \text{ nach vorn kommt}$$

   das Gewicht von (x, y). Das Gewicht hängt nur von y ab!

2. $$\phi := \sum_{(x,y)\in \mathrm{Inv}(\mathrm{LBIT},\mathrm{LOPT})} w(x, y)$$

3. $$a_i := \mathrm{BIT}_i + \phi_i - \phi_{i-1}$$

   mit $\mathrm{BIT}_i =$ reale Kosten von BIT

Trick: Berechnung der amortisierten Kosten von BIT bei der i-ten Operation ACCESS

Damit (wie im deterministischen Fall):

$$\mathrm{BIT}(\sigma) = \sum_i \mathrm{BIT}_i \underbrace{=}_{3} \sum_i a_i + \underbrace{\phi_0}_{=0,\text{ da } \mathrm{LBIT}_0=\mathrm{LOPT}_0} - \underbrace{\phi_{\text{last}}}_{\geq 0} \leq \sum_i a_i \tag{2.1}$$

Zu zeigen: Bei Event vom Typ

1. $E(a_i) \leq \frac{7}{4}\mathrm{OPT}_i$
2. $E(a_i) \leq \frac{7}{4}\mathrm{OPT}_i - \frac{3}{4}$ (n mal)

$\Rightarrow$ Theorem.

Event vom Typ 1: schafft ≤ 1 neue Inversion ($= (x, y)$); deren Wert ($1 + b(y)$): 1 oder 2, mit derselben Wahrscheinlichkeit. $\Rightarrow$

$$E(a_i) = E(\underbrace{\mathrm{BIT}_i}_{=0}) + \underbrace{E(\phi_i - \phi_{i-1})}_{\frac{1}{2}(1)+\frac{1}{2}(2)} \leq \frac{3}{2} < \frac{7}{4} = \frac{7}{4} \cdot 1 = \frac{7}{4} \cdot \mathrm{OPT}_i$$

mit $\mathrm{OPT}_i =$ Kosten von OPT bei Bearbeitung des Events i

Event vom Typ 2: Situation wie in Abbildung 2.6. Sei

$$\begin{aligned} I &:= \#x : x \text{ steht in LBIT vor } y \text{ und in LOPT hinter } y \\ &= \#x : (x, y) \in \mathrm{Inv}(\mathrm{LBIT}, \mathrm{LOPT}) \end{aligned}$$

Wie früher: Von den $m - 1$ Vorgängern von y in LBIT müssen $m - 1 - I$ auch Vorgänger von y in LOPT sein $\Rightarrow m - 1 - I \leq k - 1 \Rightarrow$

$$\mathrm{BIT}_i = m \leq k + I \tag{2.2}$$

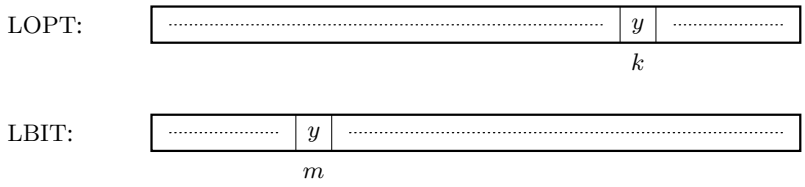

Abbildung 2.6: Event vom Typ 2

Trick: Schreibe $\phi_i - \phi_{i-1} = A + B + C$, mit

- $A =$ Gesamtgewicht aller neuen Inversionen
- $B =$ Gesamtgewicht der entfernten alten Inversionen ≤ 0
- $C =$ Gewichtsänderung der überlebenden alten Inversionen

Betrachte zunächst B und C: Falls $b(y) = 1 \Rightarrow$ BIT bewegt y nicht, $b(y) := 0$. OPT kann y weiter nach vorn bringen und dadurch Inversionen (y, z) entfernen $\Rightarrow B \leq 0$.

- Falls überlebende Inversion ihr Gewicht verändert $\Rightarrow$ sie enthält y an zweiter Stelle, hat also die Gestalt $(x, y) \Rightarrow$ Gewicht wird um 1 kleiner $\Rightarrow C = -I$.
- Falls $b(y) = 0$: BIT bringt y an Listenanfang, $b(y) := 1 \Rightarrow$ alle alten Inversionen (x, y) verschwinden, hatten vorher Gewicht $1 \Rightarrow B = -I$, $C = 0$, denn keine Inversion, deren Gewicht sich geändert hätte, kann überleben.

In den oben aufgeführten Fällen gilt stets

$$B + C \leq -I \tag{2.3}$$

$$\Rightarrow \qquad E(a_i) = E(\mathrm{BIT}_i + A + B + C) \leq E(A) + E(\underbrace{k + I}_{\text{nach (2.2)}} - \underbrace{I}_{\text{nach (2.3)}}) = \underbrace{E(A)}_{?} + k \tag{2.4}$$

Betrachte jetzt $E(A)$ (Siehe auch Abbildung 2.7): OPT bringt y (kostenfrei) an Stelle $k' \leq k$. Zwei Arten neuer Inversionen:

$b(y) = 0$**:** y wird von BIT am Listenanfang gebracht $\Rightarrow$ genau die $(y, x_1), \ldots, (y, x_{k'-1})$ sind neu, mit Gewicht

$$(y, x_j) = \underbrace{b(x_j)}_{\text{0 oder 1}} + 1 \tag{2.5}$$

$b(y) = 1$**:** y von BIT nicht verschoben $\Rightarrow$ höchstens $(x_{k'+1}, y), \ldots, (x_{k-1}, y)$ sind neu und haben Gewicht 1.

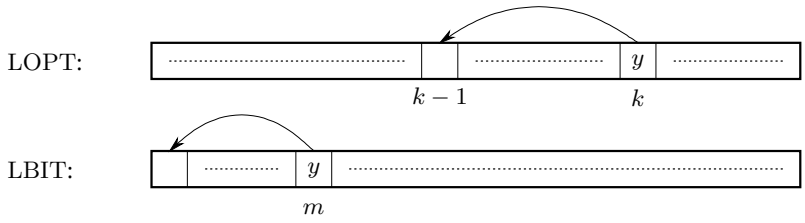

Abbildung 2.7: Situation: OPT bringt y weiter nach vorne

Beide Fälle sind gleichwahrscheinlich ⇒

$$E(A) \leq \frac{1}{2}\left(\sum_{j=1}^{k'-1}\frac{1}{2}(1+2)\right) + \frac{1}{2}\left(\sum_{j=k'+1}^{k-1} 1\right) \leq \frac{3}{4}(k-1) \tag{2.6}$$

⇒

$$E(a_i) \underbrace{\leq}_{\text{nach (2.4)}} k + E(A) \leq \frac{7}{4}k - \frac{3}{4} = \frac{7}{4}\mathrm{OPT}_i - \frac{3}{4} \tag{2.7}$$

□

Rekapitulation:

- Move To Front: deterministisch, 2-kompetitiv für dynamische Listen.
- Besser geht es nicht determnistisch.
- MTF zu BIT modifizieren: randomisiert, $\frac{7}{4}$-kompetitiv.
- Man kann auf $\frac{8}{5}$ herunterkommen mit $\frac{4}{5} \cdot \mathrm{BIT} + \frac{1}{5} \cdot \mathrm{TIMESTAMP}$
  (S. Albers '95, determ. 2-kompetitiv;
  nach $\mathrm{ACCESS}(x)$:
  bringe x vor das vorderste y, das nach dem letzten Zugriff auf x erst einmal dran war
  falls solch ein y existiert. Sonst: lasse x stehen.)
- Geht es noch besser?
  Offen!

2.2 Selbstorganisierende Bäume

Idee 1:

Statt Move To Front (nach $\mathrm{ACCESS}(x)$): Move To Root! (Abbildung 2.8). Wir müssen dabei die Such-

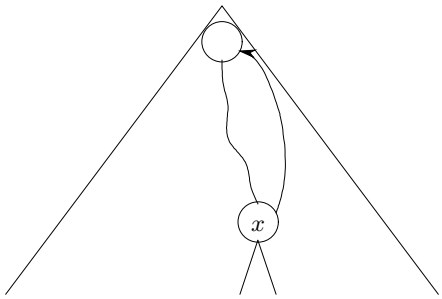

Abbildung 2.8: Move To Root

baumstruktur erhalten! Wie?

Idee 2:

Durch Rotation: Siehe Abbildung 2.9

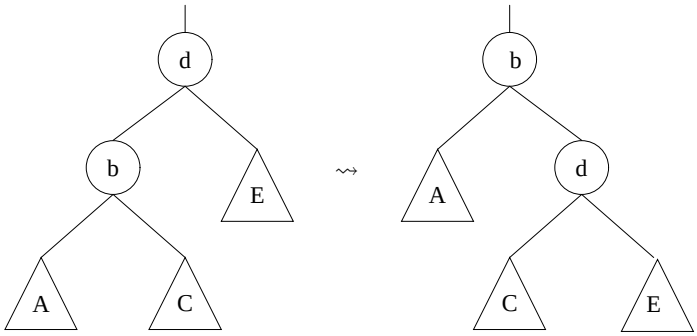

Abbildung 2.9: Rotation

Idee 3:

Wiederhole Rotation von Vater, bis x oben steht.

Kostenmodell:

(Wie bei den Listen)

- Nach $\mathrm{ACCESS}(x)$ sind Aufwärtsrotationen von x kostenfrei.
- Ansonsten kostet jede Rotation 1 Einheit.

Theorem 2.2.1 (Allen, Munro '78): Sei T Suchbaum mit Schlüsselmenge $S = \{1, 2, \ldots, 2m\}$. Dann sind bei der Zugriffsfolge $(\mathrm{ACCESS}(1), \mathrm{ACCESS}(2), \ldots, \mathrm{ACCESS}(m))^2$ die mittleren Kosten pro Zugriff in $\Omega(m)$ bei Verwendung von MTR. (Dagegen wäre beim AVL-Baum soger der Worst Case pro Zugriff in $O(\log m)$.)

Beweis (Theorem 2.2.1): entfällt. Hier nur Skizze:
Egal, mit welchem T man startet: Nach $\underbrace{\mathrm{ACCESS}(1), \ldots, \mathrm{ACCESS}(m),}_{\text{„ganzer Durchlauf"}}, \mathrm{ACCESS}(1), \ldots, \mathrm{ACCESS}(i-1)$ hat der Baum die Gestalt wie in Abbildung 2.10 $\Rightarrow \mathrm{ACCESS}(i)$ kostet $m - i + 1$ Einheiten. Wie kommt das? Grund: Lange Ketten bleiben unter MTR erhalten: Abbildungen 2.11 und 2.12 allgemein: (nach dem 3. Teilbild) (Gerte mit Knick) ... Endergebnis $\square$

Abhilfe (Sleator, Tarjan '85):

Rotiere erst am Großvater, dann am Vater: Abbildungen 2.13, 2.14 und 2.15 $\rightsquigarrow$ Splay Trees (Splay = ausgebreitet, gespreizt).
MTR*: falls Vater(x) existiert
falls Großvater(x) existiert
falls Großvater(x), Vater(x), x auf Rechts- oder Linkspfad:
rotiere Großvater(x), Vater(x)
sonst rotiere Vater(x), [Groß-]Vater(x)
sonst rotiere Vater(x).

Literatur: Ottmann/Widmayer: Algorithmen und Datenstrukturen.

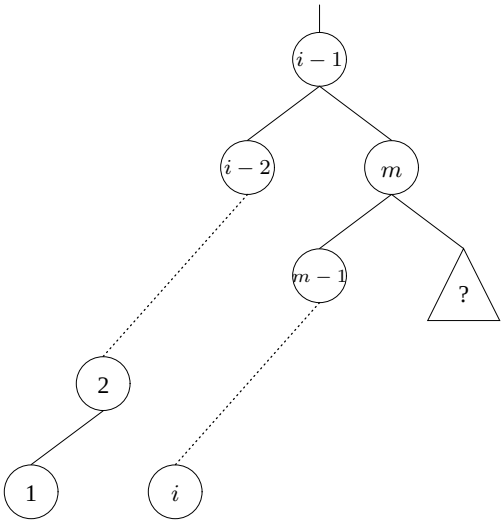

Abbildung 2.10: Baum nach Teildurchlauf

3 Fälle:

Zig-Zig: Abbildung 2.16

Zig-Zag: Abbildung 2.17

Zig: Abbildung 2.18

Achtung: Auch die Zig-Zig-Fälle können die Höhe reduzieren.

Analyse

Jedes gespeicherte Element i habe Gewicht $w(i) > 0$. (Die $w(i)$ können später nach Bedarf festgelegt werden).

Definition 2.2.1: x Knoten von T:

$$s(x) := \sum_{i \text{ in Teilbaum } T_x} w(i) \text{ (siehe Abbildung 2.19)}$$

$$r(x) := \log_2 s(x)$$

$$\phi(T) := \sum_{x \text{ in } T} r(x)$$

$$\text{Splay}(x, T) := \text{echte Kosten des Aufrufs MTF} * (x) \text{ in } T$$

$$A(x, T) := \text{Splay}(x, T) + \phi(\tilde{T}) - \phi(T), \text{ amortisierte Kosten vom Zugriff auf } x;$$

$$\tilde{T} = \text{der aus } T \text{ durch MTF} * (x) \text{ entstehende Baum.}$$

Brauchen ein technisches

Lemma 2.2.2: Seien T, T' zwei beliebige (Splay-Trees) über $\{1, \ldots, n\}$, und sei $W := \sum_{i=1}^{n} w(i)$. Dann gilt:

$$|\phi(T) - \phi(T')| \leq \sum_{i=1}^{n} \log \frac{w}{w(i)}$$

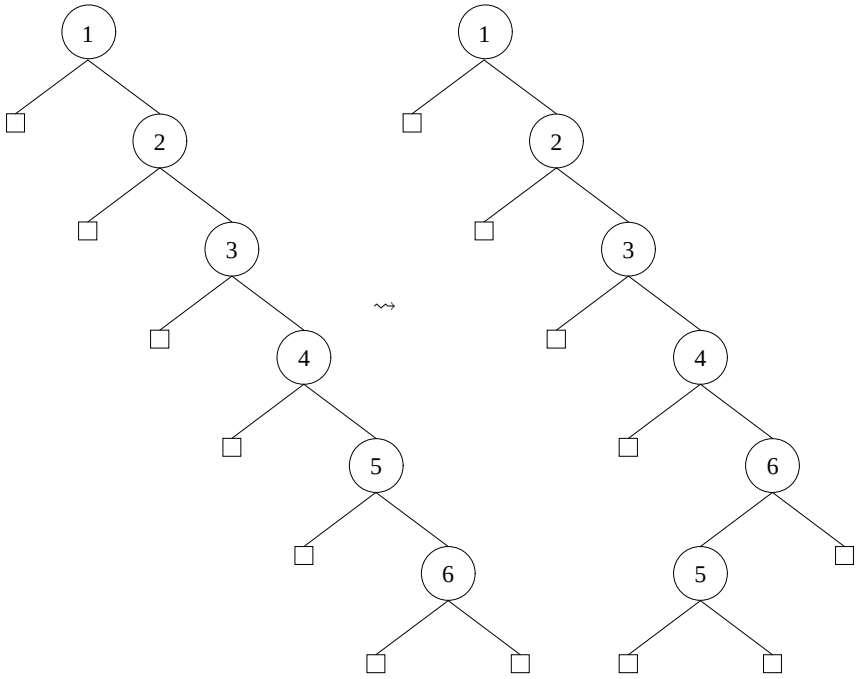

Abbildung 2.11: Lange Ketten unter MTR (a)

Beweis (Lemma 2.2.2): Seien x_i und x_i' die Knoten von T und T', die das Element i enthalten. Klar:

$$\begin{aligned} & w(i) \le s(x_i) \quad s(x_i') \le w \qquad \text{(sehr grob!)} \\ \Rightarrow \qquad & \log w(i) \le r(x_i) \quad r(x_i') \le \log w \\ \Rightarrow \qquad & \phi(T) - \phi(T') = \sum_{i=1}^{n} (r(x_i) - r(x_i')) \\ & \le \sum_{i=1}^{n} (\log w - \log(w(i))) \\ & = \sum_{i=1}^{n} \log \frac{w}{w(i)} \end{aligned}$$

und symmetrisch □

Lemma 2.2.3 (ACCESS-Lemma): $A(x,T) =$ sind die Kosten von einer elementaren Operation (zig-zig, zig-zag, zig) bei ACC
$\phi(T') - \phi(T)$

$$A(x,T) \le \begin{cases} 3(r'(x) - r(x)) & \text{falls zig-zig, zig-zag} \\ 3(r'(x) - r(x)) + 1 & \text{falls zig} \end{cases}$$

mit r' Rand-Funktion des Baumes T', der aus einem Zugriff entsteht.

Beweis (Lemma 2.2.3): Fallunterscheidung:

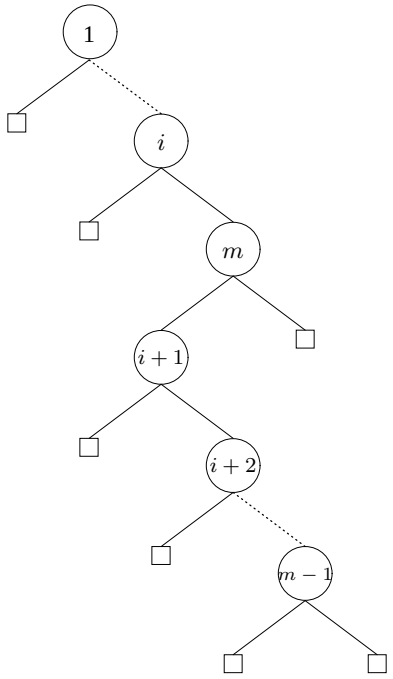

Abbildung 2.12: Lange Ketten unter MTR (b)

1. es finden (bei $\mathrm{ACCESS}(x)$) keine Umbauten statt $\Rightarrow x = t =$ Wurzel von $T \Rightarrow$

$$A(x,T) \leq^{!} 3(\underbrace{r(t) - r(x)}_{=0}) + 1$$
$$= \underbrace{\mathrm{Splay}(x,T)}_{=1,\text{ da } x \text{ Wurzel}} + \underbrace{\phi(\tilde{T}) + \phi(T)}_{=0,\text{ da } \tilde{T} = T}$$

   $\surd$

2. Es finden Umbauten statt. Zu zeigen: Sei $\sigma \in \{\mathrm{zig}, \mathrm{zig}-\mathrm{zig}, \mathrm{zig}-\mathrm{zag}\}$, T der Baum vor und T' der Baum nach der Ausführung von σ. Seien r und r' die Rangfunktionen von T und T'.

   Dann gilt:

   $\sigma = \mathrm{zig}\ \ 1 + \phi(T') - \phi(T) \leq 3(r'(x) - r(x)) + 1$
   $\sigma \in \{\mathrm{zig}-\mathrm{zig}, \mathrm{zig}-\mathrm{zag}\}\ \ 2 + \phi(T') - \phi(T) \leq 3(r'(x) - r(x)) + 0$

Daraus folgt Lemma 2.2.3, denn

- zig kommt höchstens $1\times$ vor (bei $\mathrm{ACCESS}(x)$), wenn nämlich Suchpfad ungerade Länge hat.
- Im letzten Baum, $\tilde{T}$ gilt (mit r/r' Rangfunktion von $T/\tilde{T}$): $r'(x) = r(t)$, denn beide Elemente x, t sind Wurzeln von Bäumen mit identischen Einträgen.

Fall 1 $\sigma = \mathrm{zig}$

Zu zeigen: $1 + \phi(T') - \phi(T) \leq 3(r'(x) - r(x)) + 1$. Siehe Abbildung 2.18. Nur $x, y = V(x)$ haben ihre r-Werte geändert:

$$1 + \phi(T') - \phi(T) = 1 + r'(x) + r'(y) - r(x) - r(y)$$
$$= \underbrace{r'(y) - r(y)}_{\leq 0} + \underbrace{r'(x) - r(x)}_{\geq 0} + 1 \qquad \leq 3(r'(x) - r(x)) + 1$$

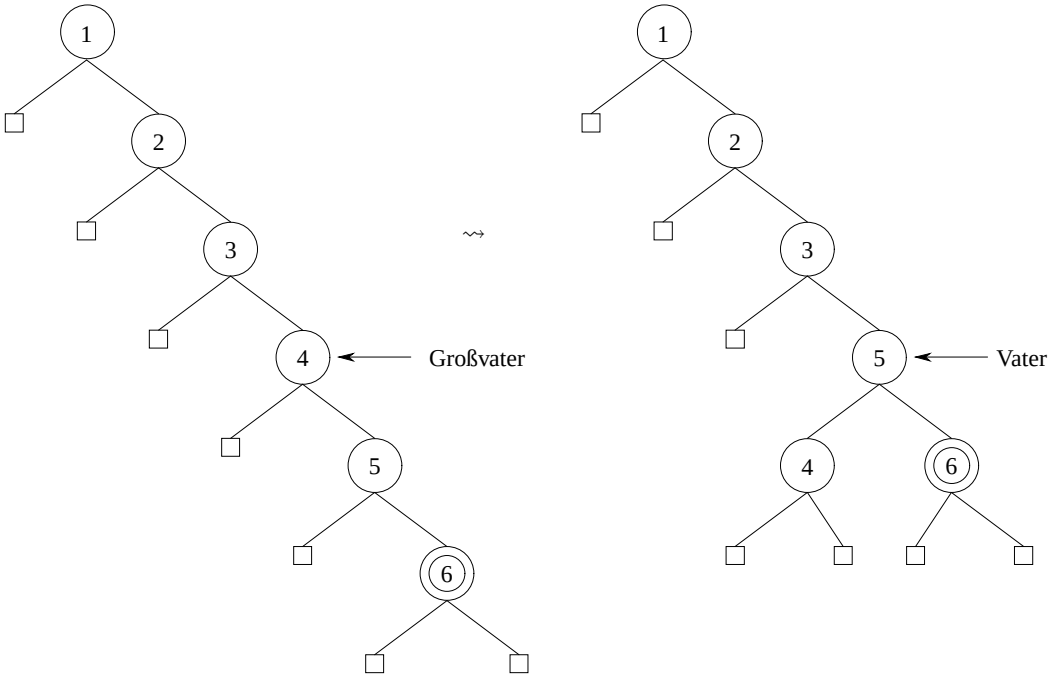

Abbildung 2.13: Veränderte Rotation (a)

Fall 2

$\sigma = \text{zig} - \text{zig}$ Abbildung 2.16 veranschaulicht die Rotationen. Klar: Nur x, y, z können ihre Ränge ändern. Es gilt:

$$\begin{aligned} 2 + \phi(T') - \phi(T) &= 2 + \underbrace{r'(x) - r(z)}_{=0} + \underbrace{r'(y)}_{\leq r'(x)} + r'(z) - r(x) \underbrace{-r(y)}_{-r(x)} \\ &\leq 2 + r'(x) + r'(z) - 2r(x) \\ &\overset{!}{\leq} 3(r'(x) - r(x)) \end{aligned}$$

d.h. zu zeigen:

$$\begin{aligned} \underbrace{r(x) + r'(z) - 2r'(x)}_{=\log s(x) + \log s'(z) - 2\log s'(x)} &\overset{!}{\leq} -2 \\ &= \log \underbrace{\frac{s(x)}{s'(x)}}_{\in(0,1)} + \log \underbrace{\frac{s'(z)}{s'(x)}}_{\in(0,1)} \end{aligned}$$

Es gilt

- $s(x) < s'(x)$, $s'(z) < s'(x)$
- $s(x) + s'(z) < s'(x)$

$$\Rightarrow \qquad \frac{s(x)}{s'(x)} + \frac{s'(z)}{s'(x)} < 1$$

Zum glück gilt: Die Funktion $f(v, w) := \log v + \log w$ hat für $0 < v, w < 1$, $v + w \leq 1$ ihr Maximum -2 (an der Stelle $(v, w) = (\frac{1}{2}, \frac{1}{2})$). Denn $\log(vw) = \log v + \log w \leq -2 \Leftrightarrow vw \leq \frac{1}{4}$. vw wird maximal für $v + w = 1$. $\surd$

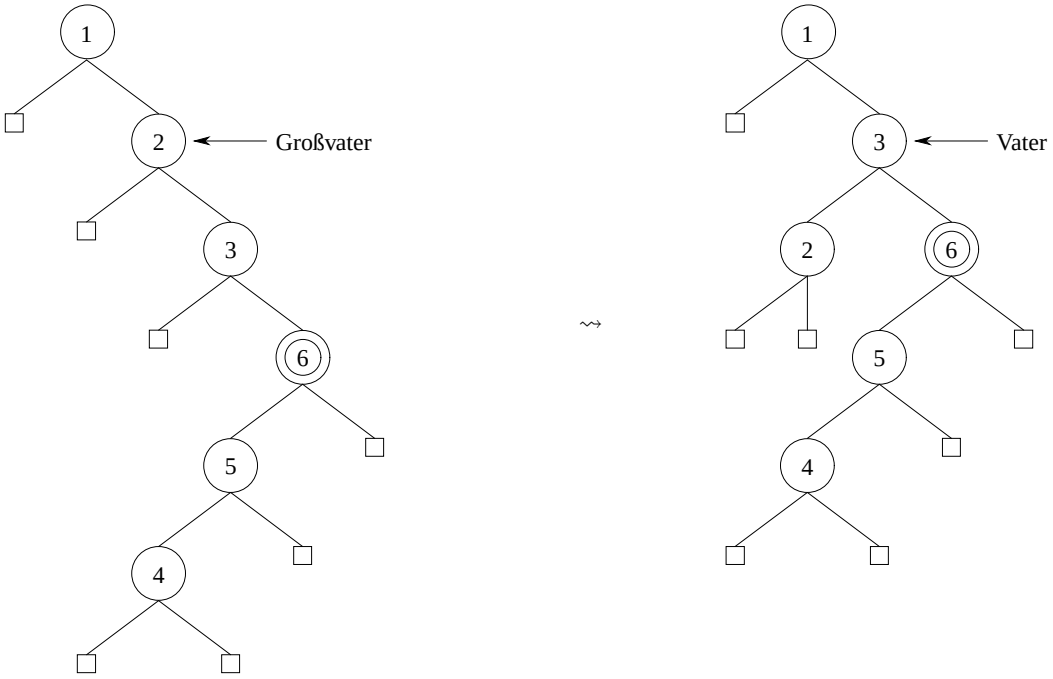

Abbildung 2.14: Veränderte Rotation (b)

$\sigma = \text{zig} - \text{zag}$ Abbildung 2.17 veranschaulicht die Rotationen. Klar: Nur x, y, z können Rang ändern.

$$\begin{aligned} 2 + \phi(T') - \phi(T) &= 2 + \underbrace{r'(x) - r(z)}_{=0} + r'(y) + r'(z) - r(x) \underbrace{-r(y)}_{\leq -r(x)} \\ &\leq 2 + r'(y) - r'(z) - 2r(x) \\ &\overset{!}{\leq} 2(\underbrace{r'(x) - r(x)}_{\geq 0}) \qquad \text{mehr als nötig!} \end{aligned}$$

Zu zeigen:

$$2 \overset{!}{\leq} 2r'(x) - r'(y) - r'(z)$$

Nun gilt: $s'(x) \geq s'(y) + s'(z)$, also (mit Eigenschaften des arithmetischen und geometrischen Mittel):

$$\begin{aligned} & \frac{s'(x)}{2} \geq \frac{s'(y) + s'(z)}{2} \geq \sqrt{s'(y)s'(z)} \\ \Rightarrow \quad & \frac{s'(x)^2}{4} \geq s'(y)s'(z) \\ \Rightarrow \quad & \frac{s'(x)^2}{s'(y)s'(z)} \geq 4 \\ \Rightarrow \quad & \underbrace{\log \frac{s'(x)}{s'(y)} + \log \frac{s'(x)}{s'(z)}}_{2r'(x) - r'(y) - r'(z)} = \log \frac{s'(x)^2}{s'(y)s'(z)} \geq \log 4 = 2 \end{aligned}$$

□

Theorem 2.2.4 (Balance Theorem): Sei T ein Splay-Tree mit n Elementen. Dann verursacht eine Folge

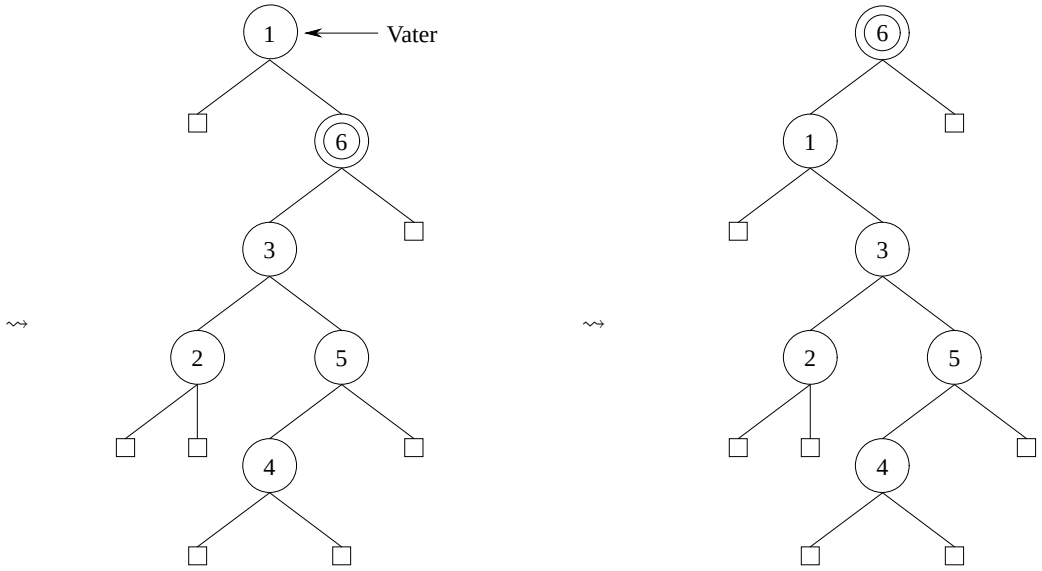

Abbildung 2.15: Veränderte Rotation (c)

von m ACCESS-Operationen Kostetn in

$$O((m+n)\log n + m)$$

Beweis (Theorem 2.2.4): Wähle alle Gewichte $w(i) = \frac{1}{n}$

$$\Rightarrow \qquad W = \sum_i w(i) = 1$$

Amortisierte Kosten pro ACCESS-Operation im aktuellen Baum T (mit Lemma 2.2.3):

$$\begin{aligned} A(x,T) &\leq 3(\underbrace{r(t)}_{\log W = 0} - \underbrace{r(x)}_{\geq \log(\text{Gewicht von } x) = \log \frac{1}{n}}) + 1 \\ &\leq 3\log n + 1 \\ \Rightarrow \qquad \text{echte Gesamtkosten} &= \underbrace{\sum_x A(x,T)}_{3m\log n + m} + \underbrace{\phi(T) - \phi(T')}_{\leq \sum_{i=1}^n \log \frac{W}{w(i)} = n\log n} \end{aligned}$$

□

Was besagt Theorem 2.2.4?: Für lange Zugriffsfolgen, d.h. für große m, sind Splay-Trees fast so gut wie balancierte Bäume ($O(m\log n)$). Was soll's? → Siehe Theorem 2.2.5.

Theorem 2.2.5 (Statische Optimalitätstheorem): In einer Folge von m Zugriffen auf Elemente der Menge $\{1,2,\ldots,n\}$. Sei $q(i) \overset{?}{\geq} 1$ die Anzahl der Zugriffe auf Element i. Dann sind die Gesamtkosten aller Zugriffe in

$$O\left(m + \sum_{i=1}^n q(i)\log\frac{m}{q(i)}\right)$$

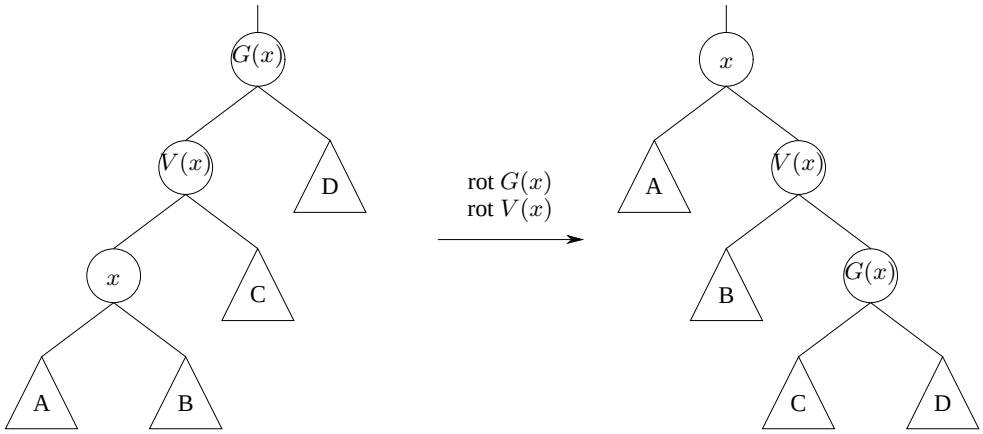

Abbildung 2.16: Zig-Zig

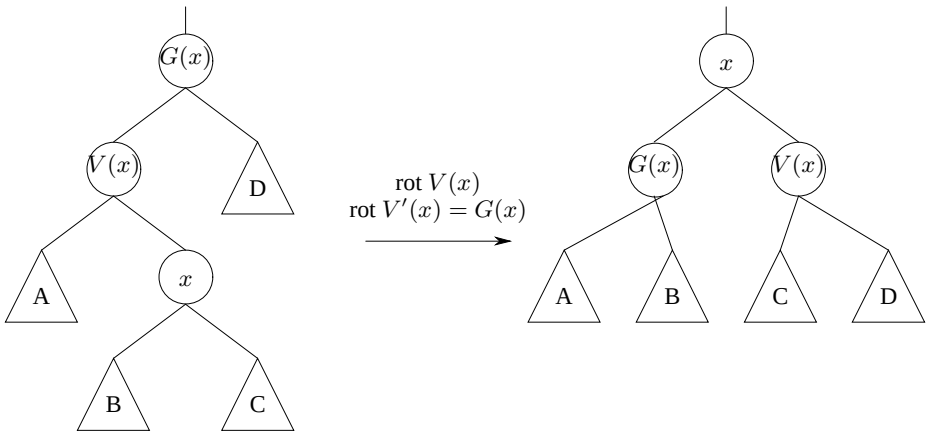

Abbildung 2.17: Zig-Zag

Beweis (Theorem 2.2.5): Diesmal setzten wir $w(i) := \frac{q(i)}{m} \Rightarrow W = \sum_i w(i) = 1$. Die amortisierten Kosten pro Zugriff auf Element i (mit Lemma 2.2.3):

$$\leq 3(\underbrace{r(t)}_{=0} - \underbrace{r(x_i)}_{\geq \log \frac{q(i)}{m}}) + 1 \geq 3 \log \frac{m}{q(i)} + 1$$

mit Lemma 2.2.2 folgt

$$\text{echte Gesamtkosten} \leq \sum_{i=1}^{n} q(i)(3 \log \frac{m}{q(i)} + 1) + \underbrace{\phi(T) - \phi(T')}_{\leq \sum_{i=1}^{n} \log \underbrace{\frac{W}{w(i)}}_{=\frac{m}{q(i)}} = n \log \frac{m}{q(i)}}$$

$$\leq c \sum_{i=1}^{n} q(i) \log \frac{m}{q(i)} + \sum_{i=1}^{n} + \sum_{i=1}^{n} q(i)$$

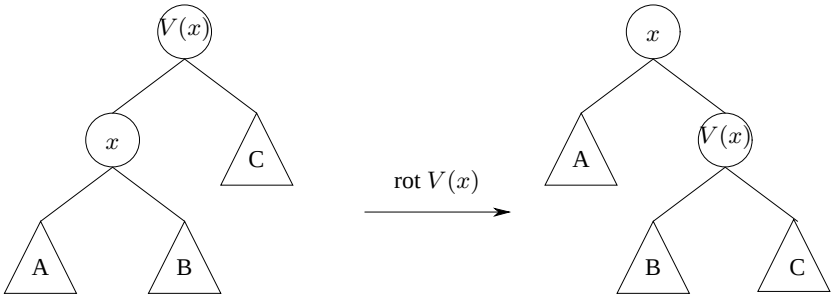

Abbildung 2.18: Zig

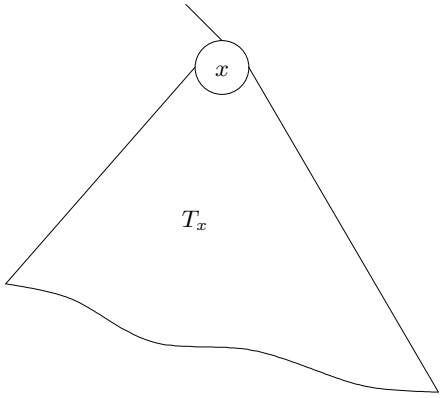

Abbildung 2.19: $s(x)$

□