

Online - Algorithmen

Literatur

- Allan Borodin, Ran El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.
- Amos Fiat, Gerhard J. Woeginger (Eds.), Online Algorithms: The State of the Art, Springer, LNCS 1442, 1998.
- Sandy Irani, Anna R. Karlin, Online Computation, in Dorit S. Hochbaum (ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, 1997.
- Heiko Röglin, Online - Algorithmen, Vorlesungsskript, 2016.

0. Motivation

Offline - Problem $\leftrightarrow$ Online - Problem

Bsp.:

Offline - Probleme:

- Sortieren einer gegebenen Datenmenge.
- Netzwerkflussprobleme
- Lösen eines gegebenen linearen Programms
- Stringmatching - Probleme

Online - Probleme:

- Bewegungsplanung bei unvollständiger Information
- Spekulation mit Aktien
- Währungsaustausch
- Routing im Internet

Online - Algorithmus $\leftrightarrow$ Offline - Algorithmus

Güte eines Online - Algorithmus:

- Qualität der Ausgabe
- benötigte Gesamtzeit.

Analyse:

- average case Analyse:

Problem: Was sind akkurate Annahmen bzgl. der Verteilung der Eingaben?

- competitive (konkurrierende) Analyse:

Die Ausgabe des Online - Algorithmus wird zur Ausgabe eines optimalen Offline - Algorithmus in Relation gesetzt.

Nachteil: äußerst pessimistische Annahme, dass stets die für den Algorithmus ungünstigste Eingabe vorliegt.

Ziel:

Erarbeiten von grundlegenden Analysetechniken anhand von Beispielen.

1. Selbstorganisierende Datenstrukturen1.1. Unsortierte lineare ListenAufgabenstellung:

Verwaltung einer Menge S von Items unter folgenden Operationen:

- $ACCESS(x)$:
Lokaliere das Item x in S .
- $INSERT(x)$
Füge das Item x in S ein.
- $DELETE(x)$
Streiche das Item x aus S .

Datenstruktur : unsortierte lineare Listen.

Implementierung der Operationen:

$ACCESS(x)$: Vom Anfang der Liste an werden die Items nacheinander betrachtet bis x gefunden ist.

$INSERT(x)$:

(1) Verifikation, dass x nicht bereits in

④

S ist (d.h., die gesamte Liste wird betrachtet.)

(2) Falls $x \notin S$, dann wird x an das Listende gehängt.

DELETE(x): ACCESS(x) gefolgt von Entfernen des Items x .

Kosten der Operationen:

ACCESS(x): i , wobei x in der i -ten Position der Liste steht.

INSERT(x): $n+1$, wobei n die Länge der Liste ist.

DELETE(x): $i+1$, wobei i die Kosten von ACCESS(x) sind.

Modifikation der Anordnung der Liste:

- freier Positionswechsel:

Direkt nach Durchführung der Operation ACCESS(x) oder INSERT(x) kann x ohne Extrakosten an eine beliebige Position näher zum Anfang der Liste platziert werden.

gewünschter Effekt: Verringerung der Kosten für nachfolgende Operationen.

- bezahlter Positionswechsel

Zu jedem Zeitpunkt können zwei aufeinanderfolgende Items ihre Positionen tauschen.

Kosten hierfür betragen 1.

- bisher: dynamisches Listenverwaltungsproblem

Einschränkung:

gegeben: Liste der Länge n .

Operationen: nur ACCESS-Operationen.



statisches Listenverwaltungsproblem

Positionswechselregeln:

- MTF (Move-to-front):

Plaziere nach ACCESS(x) oder INSERT(x) das Item x an den Anfang der Liste

- TRANS (Transpose):

Vertausche nach ACCESS(x) oder INSERT(x) die Positionen von x und dem unmittelbar davor stehenden Item.

- FC (Frequency-count):

Verwalte für jedes Item x einen Zähler $Z(x)$. Dieser hat zu Beginn den Wert 0 und wird nach jeder ACCESS(x) - bzw. INSERT(x) -

Operation um 1 erhöht. Organisiere die Liste derart, dass die Zählerwerte der Items in der Reihenfolge der Listenordnung monoton fallen.

Ziel:

Durchführung von kompetitiven Analysen für das Listenverwaltungsproblem.

Notationen bzgl. gegebenen Algorithmus ALG:

- $ALG_p(\sigma)$ Anzahl der bezahlten Positionswechsel, die ALG bei der Operationenfolge σ durchführt.
- $ALG_f(\sigma)$ Anzahl der freien Positionswechsel, die ALG bei der Operationenfolge σ durchführt.
- $ALG_c(\sigma)$ Kosten aller Operationen $\neq$ bezahlte Positionswechsel, die ALG bei der Operationenfolge σ durchführt.
- $ALG(\sigma)$ Gesamtkosten von ALG bei der Operationenfolge σ .

Beobachtung:

$$ALG \in \{MTF, TRANS, FC\} \Rightarrow ALG_p(\sigma) = 0$$

$$\Rightarrow ALG(\sigma) = ALG_c(\sigma).$$

Nachfolgend berechnet OPT stets einen optimalen Offline - Algorithmus.

Satz 1.1

Sei σ eine Operationsfolge der Länge n . Nehmen wir an, dass MTF und OPT mit derselben Listenkonfiguration starten. Dann gilt

$$MTF(\sigma) \leq 2 \cdot OPT(\sigma) - n.$$

Beweis:

Seien

x_j das Item in Position j der aktuellen Liste
OPT des Algorithmus OPT und

die Position von x_j in der aktuellen Liste MTF des Algorithmus MTF.



* bezeichnet ein Item, das in der Liste OPT nach x_j , in der Liste MTF jedoch vor x_j steht. Solch ein Item heißt Inversion bzgl. x_j .

Bezeichnung

u die Anzahl der Inversionen bzgl. x_j .

Dann gilt

$$k - u \leq j.$$

Ferner gilt

$$\text{Kosten}_{\text{OPT}}(\text{ACCESS}(x_i, i)) = j + \# \text{ bez. P.w.} \geq j$$

und

$$\text{Kosten}_{\text{MTF}}(\text{ACCESS}(x_i)) = k \leq j + u.$$

$\Rightarrow$

$$\text{Kosten}_{\text{MTF}}(\text{ACCESS}(x_i)) \leq \text{Kosten}_{\text{OPT}}(\text{ACCESS}(x_i)) + \# \text{ Inversionen bzgl. } x_i$$

Effekt der Operation $\text{ACCESS}(x_i)$ auf $\#$ Inversionen bzgl. x_i und bzgl. der anderen Items in der Liste:

- MTF platziert x_i an den Listenanfang

$\Rightarrow$

- u Inversionen bzgl. x_i werden beseitigt und
- $\leq k - u - 1$ Inversionen bzgl. anderer Items werden kreiert.

Ziel:

Kosten und Änderungen bzgl. $\#$ der Inversionen zueinander in Beziehung setzen.

Bezeichnungen:

- ϕ_i Anzahl der Inversionen in Liste MTF nach der Durchführung der i -ten Operation
- t_i Kosten der i -ten Operation für MTF
- $a_i := t_i + \phi_i - \phi_{i-1}$ amortisierten Kosten der i -ten Operation

Falls $\text{ACCESS}(x_j)$ die i -te Operation ist, dann gilt mit obigen Definitionen

$$\begin{aligned} a_i &\leq k + (k-1-u) - u \\ &= 2(k-u) - 1 \\ &\leq 2j - 1. \end{aligned}$$

D.h., die amortisierten Kosten sind höchstens doppelt so hoch wie die Kosten bezüglich OPT reduziert um 1.

Falls $\text{DELETE}(x_j)$ die i -te Operation ist, dann gilt obige Beobachtung auch.

Annahme:

i te Operation ist $\text{INSERT}(x)$, $\text{DELETE}(x)$ oder $\text{ACCESS}(x)$, wobei x nicht in der Liste ist.

Bezeichne l die Länge der Liste unmittelbar vor der i -ten Operation. Dann gilt

$$a_i \leq \ell + 1 + \ell \\ = 2(\ell + 1) - 1.$$

Da die Kosten von OPT für die i -te Operation $\geq \ell + 1$ sind, gilt obige Beobachtung auch für diese Operationen. Also gilt insgesamt

$$\sum_{i=1}^n a_i \leq 2 \cdot \text{OPT}(\sigma) - n.$$

Ziel:

Wir möchten $\text{MTF}(\sigma) = \sum_{i=1}^n t_i$ und $\sum_{i=1}^n a_i$ zueinander in Beziehung setzen.

Es gilt:

$$\begin{aligned} \sum_{i=1}^n a_i &= \sum_{i=1}^n (t_i + \phi_i - \phi_{i-1}) \\ &= \phi_n - \phi_0 + \sum_{i=1}^n t_i \end{aligned}$$

$\Leftrightarrow$

$$\begin{aligned} \sum_{i=1}^n t_i &= \underbrace{\phi_0 - \phi_n}_{\leq 0} + \sum_{i=1}^n a_i \\ &\quad (\text{da } \phi_0 = 0 \text{ und } \phi_n \geq 0) \end{aligned}$$

$\Rightarrow$

$$\text{MTF}(\sigma) = \sum_{i=1}^n t_i \leq \sum_{i=1}^n a_i \leq 2 \text{OPT}(\sigma) - n,$$

womit der Satz bewiesen ist.

Die Funktion ϕ_i ist eine Potentialfunktion. Der Name kommt daher, dass mittels der Potentialfunktion "potentielle Energie" gezählt wird, die dann für die Bezahlung von zukünftigen Operationen verwendet wird.

Ein Online-Algorithmus ALG heißt c-competitive, falls eine Konstante α existiert, so dass für alle endlichen Eingabefolgen I gilt

$$\text{ALG}(I) \leq c \cdot \text{OPT}(I) + \alpha.$$

Falls $\alpha = 0$, dann heißt ALG stark c-competitive.

Korollar 1.1

Der Algorithmus MTF ist stark 2-competitive.

Korollar 1.2

MTF ist bezüglich des statischen Listenverwaltungsproblems $(2 - \frac{1}{\ell})$ -competitive, wobei ℓ die Länge der statischen Liste ist.

Beweis:

Sei n die Länge der Operationsfolge σ . Dann gilt

$$\bullet \text{OPT}(\sigma) \leq n \cdot \ell \Leftrightarrow \frac{1}{\ell} \cdot \text{OPT}(\sigma) \leq n.$$

$$\bullet \text{MTF}(\sigma) \leq 2 \cdot \text{OPT}(\sigma) - n.$$

$\Rightarrow$

$$2 \text{OPT}(\sigma) - n \leq (2 - \frac{1}{\ell}) \text{OPT}(\sigma).$$



Frage:

Wie gut ist MTF im Vergleich zu anderen Online-Algorithmen?

Satz 1.2

Sei ALG ein c -competitiver deterministischer Online-Algorithmus für das statische Listenverwaltungsproblem einer Liste mit ℓ Items. Dann gilt $c \geq 2 - \frac{2}{\ell+1}$.

Beweis: (mittels Gegenspieler)

Der Gegenspieler stellt stets eine Anfrage bzgl. des letzten Elementes der aktuellen Liste.

$\Rightarrow$

Die Gesamtkosten für den Online-Algorithmus bei einer Anfragefolge der Länge n betragen $n \cdot \ell$.

Ziel:

Beweis, dass ein derart guter Offline-Algorithmus existiert, so dass obige untere Schranke folgt.

Sei π eine feste Permutation der ℓ Items in der Liste. Wir definieren bzgl. π folgenden Offline-

Algorithmus A_π :

Algorithmus A_π :

Eingabe: Startliste

Methode:

- (1) Ordne die Startliste derart um, dass die Items gemäß der Permutation π in der Liste gespeichert sind.
- (2) Führe die Anfragefolge ohne weitere Umordnungen durch.

Kosten für (1): Konstante $b \leq l^2$, die nur von l abhängt.

Übung:

Entwickeln Sie ein Verfahren, das aus der Startliste die Liste gemäß der Permutation π mit Kosten $O(l \cdot \log l)$ konstruiert.

Es gibt $l!$ verschiedene Algorithmen A_π .

Betrachten wir eine Anfrage $\text{ACCESS}(x)$.

Ziel:

Ermittlung der Kosten $SK(\text{ACCESS}(x))$, summiert über alle $l!$ Algorithmen A_π .

Für jede der möglichen l Positionen für x in der Liste gibt es $(l-1)!$ Permutationen der

übrigen Items.

$$\Rightarrow SK(\text{ACCESS}(x)) = \sum_{i=1}^{\ell} i(\ell-1)! \\ = (\ell-1)! \cdot \frac{\ell(\ell+1)}{2}$$

Also gilt für die Gesamtkosten GK für die Anfragefolge σ des Gegenspielers summiert über alle $\ell!$ Algorithmen A_{π} :

$$GK \leq \ell! \cdot b + n(\ell-1)! \cdot \frac{\ell(\ell+1)}{2}.$$

$\Rightarrow$

Für die mittlere Kosten MK über alle Algorithmen A_{π} gilt:

$$MK \leq \frac{1}{2} \cdot n(\ell+1) + b.$$

Es muss ein Algorithmus A_{π} existieren, dessen Kosten bzgl. der Anfragefolge σ nicht größer sind, als obige mittleren Kosten.

Betrachte den Quotienten

$$\frac{n \cdot \ell}{\frac{1}{2} n(\ell+1) + b}.$$

Wenn wir die Länge n der Anfragefolge beliebig wachsen lassen, dann konvergiert dieser Quotient gegen $\frac{2\ell}{\ell+1} = 2 - \frac{2}{\ell+1}$.

Ziel: Ermittlung von unteren Schranken für TRANS und FC.

Untere Schranke für TRANS:

- statisches Listenverwaltungsproblem.

Gegenspieler: Wählt immer das letzte Element der Liste.

⇒

In einer beliebig langen Anfragefolge werden nur zwei Items befragt.

Kosten für jedes Paar von Anfragen: $2 \cdot l$, wobei l die Länge der Liste ist.

Offline-Algorithmus σ_1 :

- Bewegt zunächst beide Items auf die erste und zweite Position der Liste.
- Danach betragen die Kosten für jedes Anfragepaar 3.

Kosten für eine Anfragefolge σ der Länge n , wobei n gerade ist:

TRANS : $n \cdot l$

Offline-Alg. σ_1 : $2 \cdot l + \left(\frac{n}{2} - 1\right) \cdot 3$

$$\Rightarrow \frac{\text{TRANS}(\sigma)}{\sigma_1(\sigma)} = \frac{n \cdot l}{\left(\frac{n}{2} - 1\right) \cdot 3 + 2l}$$

$$\xrightarrow{n \rightarrow \infty} \frac{2}{3} \cdot l$$

- Diskutiere das Ergebnis unter Einbeziehung des dynamischen Listenverwaltungsproblems.

Untere Schranke für FC:

Seien $k > l$ und $x_1, x_2, \dots, x_e$ die Items in der Startliste und zwar in dieser Ordnung.

Gegenspieler

for $i = 1$ to e

do

führe $(k+1-i)$ -mal $\text{ACCESS}(x_i)$ aus

od.

Beobachtung:

- keine Positionswechsel
- Es gilt

$$FC(\sigma) = \sum_{i=1}^e i(k+1-i)$$

$$= \frac{(k+1) \cdot e \cdot (e+1)}{2} - \frac{e(e+1)(2e+1)}{6}$$

$$= \frac{k \cdot e \cdot (e+1)}{2} + \frac{3e(e+1)}{6} - \frac{e(e+1)}{6} - \frac{2e^2(e+1)}{6}$$

$$= \frac{k e (e+1)}{2} + \frac{e(e+1)}{3} - \frac{e^2(e+1)}{3}$$

$$= \frac{k e (e+1)}{2} - \frac{(e-1) e (e+1)}{3}$$

Obere Schranke für $OPT(\sigma)$:

Betrachte $MTF(\sigma)$. Es gilt

- $MTF(\sigma) \geq OPT(\sigma)$
- $MTF(\sigma) = \sum_{i=1}^{\ell} [i + (k-i)] = k \cdot \ell.$

$\Rightarrow$

$$\frac{FC(\sigma)}{OPT(\sigma)} \geq \frac{FC(\sigma)}{MTF(\sigma)}$$

$$= \frac{\ell+1}{2} - \frac{(\ell-1)(\ell+1)}{3k}.$$

Wegen $k > \ell$ gilt:

$$\frac{\ell+1}{2} - \frac{(\ell-1)(\ell+1)}{3k} \geq \frac{\ell+1}{2} - \frac{\ell-1}{3} = \frac{\ell+5}{6}.$$

erner gilt

$$\frac{\ell+1}{2} - \frac{(\ell-1)(\ell+1)}{3k} \xrightarrow{k \rightarrow \infty} \frac{\ell+1}{2}.$$

weitere Positionswechselregel:

- TS (Timestamp):

Nach $ACCESS(x)$ oder $INSERT(x)$ füge x unmittelbar vor das erste y vor x , das seit der letzten Anfrage bzgl. x höchstens einmal angefragt wurde. Fall y nicht existiert, dann führe keinen Positionswechsel durch.

Übung:

- a) Beweisen Sie für MTF explizit eine $2 - \frac{2}{e+1}$ untere Schranke für den competitiveness Faktor bzgl. dem statischen Listenverwaltungsproblems.
- b) Tun Sie für TS dasselbe.

Ziel: Beweis, dass TS 2-competitive ist.

Bemerkung:

Es ist kein Beweis bekannt, der die Potential-funktionenmethode verwendet. Wir werden eine neue Technik, mit deren Hilfe wir obiges Ziel erreichen, kennenlernen.

Die Listenfaktorisierungstechnik

Idee:

Reduktion der competitiveness Analyse von Listenverwaltungsalgorithmen auf Listen der Länge zwei.

Vorüberlegungen:

- 1) Die Operation $\text{ACCESS}(x)$ hat Kosten i , wobei i die Position von x in der aktuellen Liste ist. Dabei gilt:
 - $(i-1)$ Vergleiche gehen negativ aus.
 - Der letzte Vergleich geht positiv aus.
- 2) Die Anzahl der positiven Vergleiche ist in allen

(19)

Algorithmen gleich. Daher können die positiven Vergleiche bei der Analyse herausgenommen werden.



partiellles Kostenmodell: positive Vergleiche werden nicht gezählt.

volles Kostenmodell: alle Vergleiche werden gezählt

Eigenschaften:

- Falls keine bezahlte Positionswechsel durchgeführt werden, dann ergeben sich im partiellen Kostenmodell die Gesamtkosten aus den negativen Vergleichen.

Mögliche Zuordnung der Kosten $(i-1)$ für die negativen Vergleiche bei der Operation $\text{ACCESS}(x)$:

- 1) dem i -ten Item x oder
- 2) denjenigen Items, bzgl. denen die negativen Vergleiche erfolgt sind.
genauer: Ein negativer Vergleich verursacht die Kosten 1 für das betroffene Item.

Gegeben einen Algorithmus ALG und eine Anfragefolge $\sigma = r_1, r_2, \dots, r_n$ definieren wir

$$\text{ALG}_1(x, j) := \begin{cases} 1 & \text{falls } x \text{ bei Anfrage } \tau_j \text{ vor} \\ & \tau_j \text{ in der aktuellen Liste steht} \\ 0 & \text{sonst (einschließlich } x = \tau_j) \end{cases}$$

Interpretation:

$\text{ALG}_1(x, j)$ sind die Kosten, die dem Item x bei der Anfrage τ_j zugeordnet werden.

Seien

• $\text{ALG}^*(\sigma)$ die von σ verursachten Kosten beim partiellen Kostenmodell.

L

Liste

Dann gilt:

$$\begin{aligned} \text{ALG}^*(\sigma) &= \sum_{j=1}^n \sum_{x \in L} \text{ALG}_1(x, j) \\ &= \sum_{x \in L} \sum_{j=1}^n \text{ALG}_1(x, j) \\ &= \sum_{x \in L} \sum_{y \in L} \sum_{j: \tau_j = y} \text{ALG}_1(x, j) \\ &= \sum_{\substack{x, y \in L \\ x \neq y}} \sum_{j: \tau_j = y} \text{ALG}_1(x, j) \\ &= \sum_{\substack{\{x, y\} \subseteq L \\ x \neq y}} \sum_{j: \tau_j = y} \text{ALG}_1(x, j) \end{aligned}$$

$$(*) = \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} \sum_{j: r_j \in \{x,y\}} [ALG(x,j) + ALG(y,j)] \quad (21)$$

Für alle $\{x,y\} \subseteq L$ für alle Anfragefolgen σ sei

$$ALG_{xy}^*(\sigma) := \sum_{j: r_j \in \{x,y\}} [ALG(x,j) + ALG(y,j)]$$

Eingesetzt in $(*)$ erhalten wir

$$(**) \quad ALG^*(\sigma) = \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} ALG_{xy}^*(\sigma).$$

Sei σ_{xy} diejenige Anfragefolge, die wir aus σ erhalten, indem wir alle Anfrage bzgl. Items $\notin \{x,y\}$ herausnehmen.

Sei L_{xy} diejenige Liste, die wir aus der Liste L erhalten, indem wir Items $\notin \{x,y\}$ ignorieren.

Dann bezeichnet $ALG^*(\sigma_{xy})$ diejenige Kosten, die der Algorithmus ALG bei der Anfragefolge σ_{xy} bzgl. Liste L_{xy} bezahlt.

Beobachtung:

Die relativen Positionen von x und y sind in L_{xy} und in L stets dieselben.

Bezeichnung:

ALG erfüllt die paarweise Eigenschaft pE , falls $\forall \sigma \forall L \forall \{x, y\} \subseteq L$ gilt:

$$(***) \quad \text{ALG}^*(\sigma_{xy}) = \text{ALG}_{xy}^*(\sigma).$$

Aus obiger Definition folgt direkt folgende Charakterisierung der Online-Algorithmen, die pE erfüllen.

Lemma 1.1

Ein Online-Algorithmus ALG erfüllt genau dann pE , wenn $\forall \sigma \forall L$ für jedes Paar $\{x, y\} \subseteq L$ ihre relative Ordnung in L , wenn ALG σ durchführt und ihre relative Ordnung in L_{xy} , wenn ALG σ_{xy} durchführt, dieselben sind.

Beispiel:

- MTF erfüllt pE .
- FC erfüllt pE .
- TRANS erfüllt nicht pE .

Idee der Listenfaktorisierungstechnik:

Annahme:

- 1) Der Online-Algorithmus ALG verwendet keine bezahlte Positionswechsel und erfüllt pE .

2) Der optimale Offline-Algorithmus OPT verwendet keine bezahlte Positionswechsel, erfüllt pE und erfüllt (**).

3) Wir haben bewiesen, dass ALG stark c-competitive bzgl. allen Listenprojektionen der Länge 2 ist. D.h.,

$$ALG^*(\sigma_{xy}) \leq c \cdot OPT^*(\sigma_{xy}) \quad \forall \{x, y\} \in L \quad \forall \sigma.$$

Dann gilt:

pE bzgl. ALG und OPT $\Rightarrow$

$$ALG_{xy}^*(\sigma) \leq c \cdot OPT_{xy}^*(\sigma) \quad \forall x, y \in L \quad \forall \sigma$$

(**) bzgl. ALG und OPT $\Rightarrow$

$$ALG^*(\sigma) \leq c \cdot OPT^*(\sigma) \quad \forall \sigma$$

Also ist ALG stark c-competitive im partiellen Kostenmodell.

Problem:

1) Im allgemeinen kann OPT keine bezahlte Positionswechsel vermeiden.

Bsp.: Betrachte Startliste $L := [x_1, x_2, x_3]$

$$\sigma := x_3 x_2 x_2 x_3$$

Ohne bezahlte Positionswechsel betragen die Kosten im partiellen Kostenmodell mindestens 5.

Besser ist folgende Strategie:

- Konstruiere vor der Behandlung von σ mittels zwei bezahlte Positionswechsel die Liste $L' = [x_2 x_3 x_1]$.
- Führe dann σ ohne Positionswechsel durch.

Gesamtkosten: 4 im partiellen Kostenmodell.

Übung:

Beweisen Sie, dass OPT auch bzgl. dem vollen Kostenmodell bezahlte Positionswechsel nicht vermeiden kann.

2) Die Gleichungen, die zu (*) und (**) führen, setzen voraus, dass keine bezahlte Positionswechsel verwendet werden. Also impliziert 1), dass sie nicht notwendigerweise für OPT gelten.

3) OPT erfüllt im allgemeinen nicht pE.

Idee:

Beweise für OPT schwächere Eigenschaften, die auch ausreichen.

Schwächere Eigenschaften bzgl. OPT:

Es gilt:

$$(+) \text{OPT}^*(\sigma_{xy}) \leq \text{OPT}_{xy}^*(\sigma) + \text{OPT}_{p:xy}^*(\sigma),$$

wobei $\text{OPT}_{p:xy}^*(\sigma)$ die Gesamtkosten von OPT für bezahlte Positionswechsel zwischen x und y sind.

Bew.:

- Die linke Seite der Ungleichung sind die Kosten eines optimalen Offline - Algorithmus für Itemlisten der Länge zwei bzgl. der Bedienung der Anfragefolge σ_{xy} .
- Die rechte Seite der Ungleichung sind die Gesamtkosten eines (nicht notwendigerweise optimalen) Offline - Algorithmus, der auf L_{xy} operiert und die Anfragefolge σ_{xy} bzgl. der relativen Ordnung von x und y in L , die OPT bei der Durchführung von σ aufrecht hält, bedient.

$\Rightarrow$

Linke Seite kann nicht größer als die rechte Seite sein.

□

Ferner gilt

$$(++) \text{OPT}^*(\sigma) = \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} [\text{OPT}_{xy}^*(\sigma) + \text{OPT}_{p:xy}^*(\sigma)]$$

Übung:

Beweisen Sie die Gleichung (**).

Annahme:

$\forall \{x, y\} \in L \quad \forall \sigma$ gilt:

$$\text{ALG}^*(\sigma_{xy}) \leq c \cdot \text{OPT}^*(\sigma_{xy})$$

Dann folgt aus pE und (+) für alle $\{x, y\} \in L$:

$$\text{ALG}_{xy}^*(\sigma) \leq c \cdot [\text{OPT}_{xy}^*(\sigma) + \text{OPT}_{p:xy}^*(\sigma)]$$

$\Rightarrow$
(**)(++)

ALG ist stark c -competitive.

Somit haben wir folgendes Lemma bewiesen:

Lemma 1.2 (Faktorisierungslemma)

Sei ALG ein Online-Algorithmus für das Listenverwaltungsproblem, der keine bezahlte Positionswechsel verwendet und pE erfüllt. Sei L eine Liste. Falls für alle $\{x, y\} \in L \quad \forall \sigma$

$\text{ALG}^*(\sigma_{xy}) \leq c \cdot \text{OPT}^*(\sigma_{xy})$, dann ist ALG stark c -competitive für L im partiellen Kostenmodell.

Frage:

Gilt ein ähnliches Lemma bzgl. dem vollen Kostenmodell?

Bezeichnung:

Ein Algorithmus heißt kostenunabhängig, falls seine Entscheidungen unabhängig von den zu bezahlenden Kosten sind. D.h., ein kostenunabhängiger Algorithmus operiert unter allen Kostenmodellen gleich.

Lemma 1.3

Sei ALG ein kostenunabhängiger, (stark) c -competitiver Online-Algorithmus bzgl. dem partiellen Kostenmodell. Dann ist ALG auch (stark) c -competitive bzgl. dem vollen Kostenmodell.

Beweis:

Bezeichne OPT^* bzw. OPT einen optimalen Offline-Algorithmus bzgl. dem partiellen bzw. vollen Kostenmodell. Sei σ eine beliebige Anfragefolge.

Seien $ALG^*(\sigma)$ bzw. $ALG(\sigma)$ die Kosten, die ALG für σ bzgl. dem partiellen bzw. vollen Kostenmodell bezahlt.

Die Anzahl der positiven Vergleiche sind jeweils $|\sigma|$.

$\Rightarrow$

$$OPT(\sigma) = B + |\sigma|,$$

wobei B die Gesamtkosten für negative Vergleiche und bezahlte Positionswechsel sind.

$\Rightarrow$

$$OPT^*(\sigma) \leq B.$$

Ferner gilt

$$\begin{aligned}
 \text{ALG}(\sigma) &= \text{ALG}^*(\sigma) + |\sigma| \\
 &\stackrel{(st) \text{ } c\text{-comp}}{\leq} c \cdot \text{OPT}^*(\sigma) + |\sigma| (+\alpha) \\
 &\leq c \cdot B + |\sigma| (+\alpha) \\
 &\leq c \cdot \text{OPT}(\sigma) (+\alpha),
 \end{aligned}$$

womit das Lemma bewiesen ist. ■

Anwendungen:

- 1) Alternativer Beweis, dass MTF stark 2-competitive für das statische Listenverwaltungsproblem ist.

Beweis:

Sei L die Startliste und σ eine beliebige Anfragefolge.

MTF verwendet keine bezahlte Positionswechsel und erfüllt auch die pE.

Le. 1.2, 1.3 $\Rightarrow$

Es genügt bzgl. dem partiellen Kostenmodell folgende Behauptung zu beweisen:

Beh.:

$$\forall \{x, y\} \subseteq L \text{ gilt: } \text{MTF}^*(\sigma_{xy}) \leq 2 \cdot \text{OPT}^*(\sigma_{xy}).$$

Bew.:

Seien i und j , $i < j$ zwei aufeinanderfolgende Indizes von Anfragen in $\sigma_{x,y}$, für die MTF (operierend auf der Zweielementliste) Kosten 1 bezahlt.

O.B.d.A. sei unmittelbar vor der Durchführung der i -ten Anfrage x vor y in der von MTF manipulierten Liste.

 $\Rightarrow$

i -te Anfrage ist $\text{ACCESS}(y)$.

 $\Rightarrow$

j -te Anfrage ist $\text{ACCESS}(x)$.

Ferner gilt:

Für mindestens eine der beiden Anfragen oder eine zwischen diesen Anfragen bezahlt $\text{OPT} \geq 1$.

 $\Rightarrow$

Wenn MTF 2 bezahlt, dann bezahlt OPT mindestens 1.

 $\Rightarrow$

$$\text{MTF}^*(\sigma_{x,y}) \leq 2 \cdot \text{OPT}^*(\sigma_{x,y}) \quad \forall \{x,y\} \subseteq L, \forall \sigma.$$

■

2) Beweis, dass TS 2-competitive für das statische Listenverwaltungsproblem ist.

Offensichtlich verwendet TS keine bezahlte

Positionswechsel. Wir werden zunächst beweisen, dass TS auch pE erfüllt.

Beh.:

TS erfüllt pE.

Bew.:

Le 1.1 $\Rightarrow$

Es genügt zu zeigen, dass $\forall \{x, y\} \in L$ die relative Ordnung von x, y in L und in L_{xy} die = selbe ist.

Betrachte hierzu $x, y \in L, x \neq y$ beliebig, aber fest. Sei τ_i die erste Anfrage in $\sigma := \tau_1, \tau_2, \dots, \tau_n$, nach der die obige Behauptung verletzt ist.

$\Rightarrow$

$\tau_i = \text{ACCESS}(x)$ oder $\tau_i = \text{ACCESS}(y)$

(da $\text{ACCESS}(z)$ mit $z \neq y, z \neq x$ die relative Ordnung von x und y weder in L noch in L_{xy} ändert.)

O.B.d.A. sei $\tau_i = \text{ACCESS}(x)$.

Jenachdem, wie die relative Ordnung von x und y vor $\text{ACCESS}(x)$ ist, unterscheiden wir zwei Fälle.

1. Fall: x steht vor y in L und in L_{xy}

$\Rightarrow$

Die relative Ordnung ändert sich in beiden Listen nicht.

Widerspruch zur Wahl von r_i .

2. Fall: y steht vor x in L und in L_{xy}

a) Seit der letzten Anfrage bzgl. x wurde y höchstens einmal angefragt.

$\Rightarrow$

Sowohl in L , als auch in L_{xy} kommt x vor y

Widerspruch.

b) Seit der letzten Anfrage bzgl. x wurde y mindestens zweimal angefragt.

$\Rightarrow$

In L_{xy} bleibt y vor x .

Frage:

Was geschieht mit der relativen Ordnung bzgl. L ?

Falls vor y alle Items mindestens zweimal seit dem letzten ACCESS (x) in τ angefragt wurden, dann bleibt y vor x .

$\Rightarrow$

$\exists$ vor y ein Item z , das seit dem letzten ACCESS (x) höchstens einmal angefragt worden ist.

Beachte:

Die letzte Anfrage bzgl. x erfolgte vor dem letzten beiden Anfragen bzgl. y .

Ferner gilt:

Unmittelbar nach der Durchführung des letzten $\text{ACCESS}(y)$ gilt für alle Items vor y :

- Die Anzahl der Anfragen zwischen den beiden letzten Anfragen von y ist mindestens zwei.

Sei r_j das letzte $\text{ACCESS}(y)$ vor r_i .

Beh.:

Für alle $j \leq l < i$ gilt für alle Items u , die unmittelbar nach Durchführung der Anfrage r_l vor dem Item y in der Liste L sind

- Anzahl $\text{ACCESS}(u)$ seit dem letzten $\text{ACCESS}(x) \geq 2$.

Bew. d. Beh.:

Wir beweisen die Behauptung mittels vollständiger Induktion.

$l=j$: haben wir uns bereits oben überlegt.

Induktionsannahme:

Für alle l mit $j \leq l \leq k < i-1$, $k \geq j$ gilt die Behauptung.

$k \leadsto k+1$:

Sei $\Gamma_{k+1} = \text{ACCESS}(u)$.

1. Fall: Die relative Position von u und y ändert sich durch Γ_{k+1} nicht.

Dann gilt auch nach Γ_{k+1} die Behauptung.

2. Fall: Die relative Position von u und y ändert sich durch Γ_{k+1} .

Dann steht unmittelbar vor Γ_{k+1} das Item u hinter y und nach Γ_{k+1} vor y auf der Liste.

Definition von TS $\Rightarrow$

$\exists$ Item u' auf der Liste, so dass

- u' steht unmittelbar vor Γ_{k+1} vor dem Item y auf der Liste.
- Seit dem letzten $\text{ACCESS}(u)$ ist $\text{ACCESS}(u')$ höchstens einmal erfolgt.

Induktionsvoraussetzung $\Rightarrow$

Nach dem letzten $\text{ACCESS}(x)$ ist $\text{ACCESS}(u')$ mindestens zweimal erfolgt.

$\Rightarrow$

Nach dem letzten $\text{ACCESS}(x)$ muss bereits mindestens einmal $\text{ACCESS}(u)$ erfolgt sein.

Also gilt nach τ_{k+1} :

Die Anzahl der ACCESS (u) seit dem letzten ACCESS (x) beträgt mindestens zwei.

□

$\Rightarrow$

Item z existiert nicht

$\Rightarrow$

in L bleibt y vor x .

□

Satz 1.3

TS ist 2-competitive für das statische Listenverwaltungsproblem.

Beweis:

Sei L die von TS manipulierte Liste. Sei σ eine beliebige Anfragefolge.

TS verwendet keine bewährte Positionswechsel und erfüllt ρE .

Faktorisierungslemma $\Rightarrow$

Es genügt zu zeigen, dass $\forall \{x, y\} \in L$

$$TS^*(\sigma_{xy}) \leq 2 \cdot OPT^*(\sigma_{xy}).$$

Betrachte $\{x, y\} \in L$, $x \neq y$ beliebig, aber fest.

Idee:

- Unterteilung von σ_{xy} in Phasen, denen, je nachdem, ob x oder y erstes Element von L_{xy} ist, ein Typ zugeordnet wird.
- Analyse der Phasen bzgl. den Kosten.

Unterteilung in Phasen:

Typ 1 (x in Front)

a) $x^i yy$

b) $x^i (yx)^k yy$

c) $x^i (yx)^k x$

Typ 2 (y in Front)

$y^i xx$

$y^i (xy)^k xx$

$y^i (xy)^k y$,

wobei $i \geq 0$ und $k \geq 1$.

O.B.d. A. können wir annehmen, dass zu Beginn x das erste Element der Liste L_{xy} ist.

$\Rightarrow$

Die 1. Phase ist vom Typ 1.

Je nachdem, wie die Phase endet hat die nachfolgende Phase Typ 1 oder Typ 2.

a) 1. Phase endet mit xx (Form c)

$\Rightarrow$

2. Phase ist wiederum vom Typ 1.

b) 1. Phase endet mit y (Form a) oder b)

$\Rightarrow$

2. Phase ist vom Typ 2

u.s.w.

$\Rightarrow$

Wir erhalten eine eindeutige Partition von $\bar{D}_{xy}$ in Phasen. Möglicherweise ist die letzte Phase nicht vollständig.

Übung:

Beweisen Sie, dass obige Unterteilung stets eine eindeutige Partition in Phasen ergibt.

Analyse der Kosten

	TS^*	OPT^*	max. Verhältnis $\frac{TS^*}{OPT^*}$
a) $x^i y y$	2	1	2
b) $x^i (y x)^k y y$	$2k$	$k+1$	< 2
c) $x^i (y x)^k x$	$2k-1$	k	< 2

Falls die letzte Phase eine unvollständige Phase vom Typ 1 ist, dann wird das Verhältnis höchstens kleiner (überzeugen Sie sich).

Analyse bzgl. Typ 2: analog.

Beobachtung:

Falls ALG und OPT mit identischen Listen starten, dann gilt stets zu Beginn und nach Beendigung einer Phase, dass die Listen gleich sind.

Übung:

Arbeiten Sie den Beweis von Satz 1.3 unter Einbeziehung der Sonderfälle aus.

Frage:

Falls der Online-Algorithmus "würfeln" darf, kann dann ein besseres competitiveness Verhältnis erzielt werden als bei deterministischen Online-Algorithmen?

Ziel: Beantwortung obiger Frage.

Modell:

- 1) ALG ist ein randomisierter Online-Algorithmus.
- 2) Der vergessliche Gegenspieler (oblivious adversary) kennt den Algorithmus ALG inklusive Wahrscheinlichkeitsverteilung, gewirft der die Zufallsentscheidungen getroffen werden. Er wählt die Anfragefolge σ in voraus und zwar ohne Kenntnis, wie die Zufallsentscheidungen ausgehen, aus.

ALG heißt c-competitive, falls für alle Anfragefolgen σ

$$E(\text{ALG}(\sigma)) \leq c \cdot \text{OPT}(\sigma) + \alpha.$$

Dabei sind α eine Konstante, $E(\cdot)$ der Erwartungswert und $\text{OPT}(\sigma)$ die Kosten eines optimalen Offline-Algorithmus.

$\bar{R}(\text{ALG}) := c$ heißt erwartetes competitiveness Verhältnis von ALG gegen einen vergesslichen Gegenspieler.

Positionswechselregel (statisches Listenverwaltungsproblem):

BIT

Für jedes x in der Liste L initialisiert BIT zufällig und uniform ein Bit $b(x) \in \{0, 1\}$. Nach jedem $\text{ACCESS}(x)$ wird zunächst $b(x)$ komplementiert. Falls dann $b(x) = 1$, dann platziere x an den Anfang der Liste. Andernfalls führe keinen Positionswechsel durch.

Effekt:

- $\ell := |L|$ Zufallsbits.
- Mit gleicher Wahrscheinlichkeit wird einer von 2^ℓ deterministischen Algorithmen gewählt.

Satz 1.4

Sei σ eine Anfragefolge der Länge n . Dann gilt

$$E(\text{BIT}(\sigma)) \leq \frac{7}{4} \text{OPT}(\sigma) - \frac{3n}{4}.$$

Beweis:

Da jedes Bit $b(x)$ zu Beginn uniform und unabhängig aus $\{0,1\}$ gewählt wird und bei $\text{ACCESS}(x)$ das Bit $b(x)$ stets komplementiert wird, gilt zu jedem Zeitpunkt für jedes x :

$b(x)$ ist uniform und unabhängig aus $\{0,1\}$ gewählt.

Der Gegenspieler kennt die Zufallswerte $b(x)$, $x \in L$ nicht.

Wir fixieren eine beliebige Anfragefolge σ und lassen die Algorithmen BIT und OPT ver= schränkt bei dieser Anfragefolge σ laufen.

D.h., BIT und OPT bedienen jede Anfrage in σ in einer festen Ordnung. (z.B. zuerst BIT, dann OPT).

$\Rightarrow$

Diese Ordnung definiert eine Folge von Ereignissen, die durch die Operationen, die beide Algorithmen auf ihren Listen durchführen, spezifiziert sind.

Arten von Ereignissen:

- von OPT durchgeführte bezahlte Positionswechsel (erfolgen zwischen Anfrageoperationen) ..
- alle anderen Operationen, die entweder BIT oder OPT durchführen (korrespondieren zu einer Anfrageoperation).

Wir nummerieren die Ereignisse durch.

Idee: Anwendung der Potentialfunktionenmethode.

Potentialfunktion ϕ :

ϕ_i : Wert der Potentialfunktion unmittelbar nach dem i -ten Ereignis.

BIT_i : Kosten, die von BIT während dem Ereignis i verursacht werden.

unartisierten Kosten a_i des i -ten Ereignis:

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

Also gilt:

$$\text{BIT}(\sigma) = \sum_i \text{BIT}_i = \phi_0 - \phi_{\text{last}} + \sum_i a_i,$$

wobei ϕ_{last} den Wert von ϕ nach dem letzten Ereignis bezeichnet.

BIT randomisiert $\Rightarrow a_i$ ist eine Zufallsvariable

Zum Beweis des Satzes genügt es folgendes zu beweisen:

$$1) \quad E(a_i) \leq \begin{cases} 7/4 \text{ OPT}_i & \forall i: \text{Ereignis } i \text{ bez. Pw.} \\ 7/4 \text{ OPT}_i - 3/4 & \forall i: \text{Ereignis } i \equiv \text{ACCESS} \end{cases}$$

wobei OPT_i die von OPT während des Ereignisses i bezahlten Kosten bezeichnet.

2) Φ_{east} ist von unten begrenzt.

Interpretation der Potentialfunktion:

Die Potentialfunktion zählt "gewichtet" die Anzahl der Inversionen zwischen den von BIT und OPT manipulierten Listen. Dabei wird die Abhängigkeit des Wertes der Potentialfunktion von den Zufallsentscheidungen über die Gewichte eingebracht.

Erinnerung:

Inversion $\hat{=}$ geordnetes Paar $[x, y]$ von Items mit:

x vor y in Liste von BIT, aber
 y vor x in List von OPT.

Das Gewicht $w[x, y]$ der Inversion $[x, y]$ ist die Anzahl der ACCESS(y), bevor y das Item

x in der Liste von BIT passiert.

$\Rightarrow$
Def. von $b(y)$

$$w[x, y] = b(y) + 1.$$

Sei

$$\phi := \sum_{\text{Inversionen } [x, y]} w[x, y].$$

$\Rightarrow$

Es gilt stets $\phi \geq 0$ und somit $\phi_{\text{last}} \geq 0$.

Falls BIT und OPT mit derselben Startliste beginnen, dann gilt: $\phi_0 = 0$.

Ereignis i:

1) bezahlter Positionswechsel von OPT:

$$\Rightarrow \begin{cases} \text{OPT}_i = 1 \\ \text{BIT}_i = 0 \end{cases}$$

Worst case: Positionswechsel kreiert neue Inversion.

Beobachtung:

Das Gewicht dieser Inversion ist 1 oder 2, jeweiles mit Wahrscheinlichkeit $1/2$.

$\Rightarrow$

- B (negativer) Beitrag von eliminierten alten Inversionen.
- C Beitrag von alten gebliebenen Inversionen, deren Gewicht sich verändert hat.

Analyse:

- B und C

Je nach Wert von $b(y)$ vor $\text{ACCESS}(y)$ unterscheiden wir zwei Fälle.

1. Fall $b(y) = 1$ vor $\text{ACCESS}(y)$

$\Rightarrow$

keine Positionswechsel in der Liste BIT, aber $b(y) = 0$ nach $\text{ACCESS}(y)$.

$\Rightarrow$

Jede Inversion $[x, y]$ ändert ihr Gewicht von 2 auf 1.

Da BIT keine Positionswechsel durchführt, gilt:

- BIT eliminiert keine alten Inversionen
- OPT's freie Positionswechsel eliminieren keine von I gezählte Inversionen
- OPT's freie Positionswechsel eliminieren möglicherweise Inversionen der Form $[y, z]$.

$\Rightarrow$

$$B \leq 0 \quad \text{und} \quad C = -I.$$

2. Fall $b(y) = 0$ vor $\text{ACCESS}(y)$

(45)

$\Rightarrow$

y wird an den Listenanfang von BIT platziert

$\Rightarrow$

Alle Inversionen $[x, y]$ werden eliminiert.

Diese hatten vor $\text{ACCESS}(y)$ das Gewicht 1.
Keine alte Inversionen ändern ihr Gewicht.

$\Rightarrow$

$$B = -I \quad \text{und} \quad C = 0.$$

1. Fall und 2. Fall $\Rightarrow$

$$\text{Es gilt stets } B + C \leq -I.$$

• A

Schwierigkeit:

Da sowohl OPT als auch BIT mittels freien Positionswechsel y nach vorne bewegen können, hängt A sowohl von BIT als auch von OPT ab.

Seien $x_1, x_2, \dots, x_{\ell-1}$ die Items in Liste OPT in Positionen $1, 2, \dots, \ell-1$ unmittelbar vor $\text{ACCESS}(y)$.

Beobachtung:

Eine neue Inversion kann nur auf folgende Art

und Weise kreiert werden:

- x_e ist vor y in Liste BIT und auch in Liste OPT und genau eines von BIT und OPT platziert y vor x_e .
(überzeugen sie sich)

Annahme:

OPT bewegt y nach vorne auf Position $k' \leq k$.

Sei für $1 \leq l \leq k-1$

X_l die Zufallsvariable bzgl. dem Beitrag der Inversion $[y, x_e]$ oder $[x_e, y]$, falls diese kreiert wurde.

Wir unterscheiden zwei Fälle:

a) $b(y) = 0$ unmittelbar vor ACCESS(y)

$\Rightarrow$

$b(y)$ wird 1 und y an den Listenanfang von BIT platziert.

$\Rightarrow$

Für $1 \leq l < k'$ wird die neue Inversion $[y, x_e]$ vom Gewicht $b(x_e) + 1$ kreiert.

$\Rightarrow$

$$X_l = \begin{cases} b(x_e) + 1 & \text{für } 1 \leq l < k' \\ 0 & \text{für } k' \leq l \leq k-1. \end{cases}$$

b) $b(y) = 1$ unmittelbar ~~for~~ ACCESS(y)

$\Rightarrow$

$b(y)$ wird 0 und y ändert seine Platzierung in BIT nicht.

$\Rightarrow$

$$X_\ell = \begin{cases} 0 & \text{für } 1 \leq \ell \leq k'-1 \\ b(y)+1 = 1 & \text{für } k' \leq \ell \leq k-1. \end{cases}$$

Insgesamt erhalten wir somit

$$\begin{aligned} E(A) &= \sum_{\ell=1}^{k-1} E(X_\ell) \\ &\leq \sum_{\ell=1}^{k'-1} \frac{1}{2} \left(\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 \right) + \sum_{\ell=k'}^{k-1} \frac{1}{2} \cdot 1 \\ &\leq \frac{3}{4} (k-1). \end{aligned}$$

Also gilt

$$\begin{aligned} E(a_i) &= E(B\pi_i + A + B + C) \\ &\leq E(k + I + A - I) \\ &= E(k + A) \\ &\leq k + \frac{3}{4} (k-1) \\ &= \frac{7}{4} k - \frac{3}{4}. \end{aligned}$$

Positionswechselregel

RMTF (Random-move-to-front)

Bei jeder Anfrage bzgl. x platziere x mit Wahrscheinlichkeit $\frac{1}{2}$ an den Anfang der Liste.

Ziel:

Beweis, dass $\bar{R}(\text{RMTF}) > 2 - \varepsilon \quad \forall \varepsilon > 0$.

Gegeben: $\varepsilon > 0$ und Liste L mit $|L| = \ell = \ell(\varepsilon)$.

D.h., $L = \langle x_1, x_2, \dots, x_\ell \rangle$
 $\uparrow$ Listenanfang

Sei $k \in \mathbb{N}$. Betrachte folgende Anfragefolge σ :

$$\sigma := (x_\ell)^k, (x_{\ell-1})^k, \dots, (x_1)^k$$

k groß genug $\Rightarrow$

RMTF platziert mit großer Wahrscheinlichkeit während der Abarbeitung der Anfragen $(x_i)^k$ das Item x_i an den Listenanfang.

Im Mittel wird x_i während der 2. Anfrage an den Listenanfang platziert.

$\Rightarrow$

Die erwarteten Kosten von RMTF sind

$$\geq 2 \cdot \ell^2 + \ell(k-2).$$

Ferner gilt: $\text{MTF}(\sigma) = \ell(\ell + k - 1)$.

Betrachte $\delta > 0$ fest. Wähle ε und $\ell \gg \frac{1}{\varepsilon}$ groß genug, so dass

$$\Pr\left(\frac{R_{MTF}(\sigma)}{MTF(\sigma)} > 2 - \varepsilon\right) > 1 - \delta.$$

Übung:

Warum führt obige Anfragefolge σ nicht zu $\bar{R}(\text{BIT}) > 2 - \varepsilon$?

Bemerkung:

- Der bisher beste randomisierte Online - Algorithmus ist 1,6 - competitive. Die beste untere Schranke beträgt 1,5.
- Bzgl. Offline - Algorithmen für das statische Listenverwaltungsproblem ist folgendes bekannt:
 - Beste obere Schranke für die Laufzeit:

$$O(2^n (n-1)! \cdot m),$$

wobei n die Listenlänge und m die Anzahl der Anfragen sind.

Nick Reingold, Jeffery Westbrook, Offline algorithms for the list update problem, IPL 60, 75-80 (1996).

- Problem ist NP - vollständig
Christoph Ambühl, Offline list update

is NP-hard, 8th ESA 2000, LNCS 1878, 42-51. ⁽⁵⁰⁾

- Offenes Problem:

$(2-\varepsilon)$ Approximationsalgorithmus, wobei $\varepsilon > 0$.

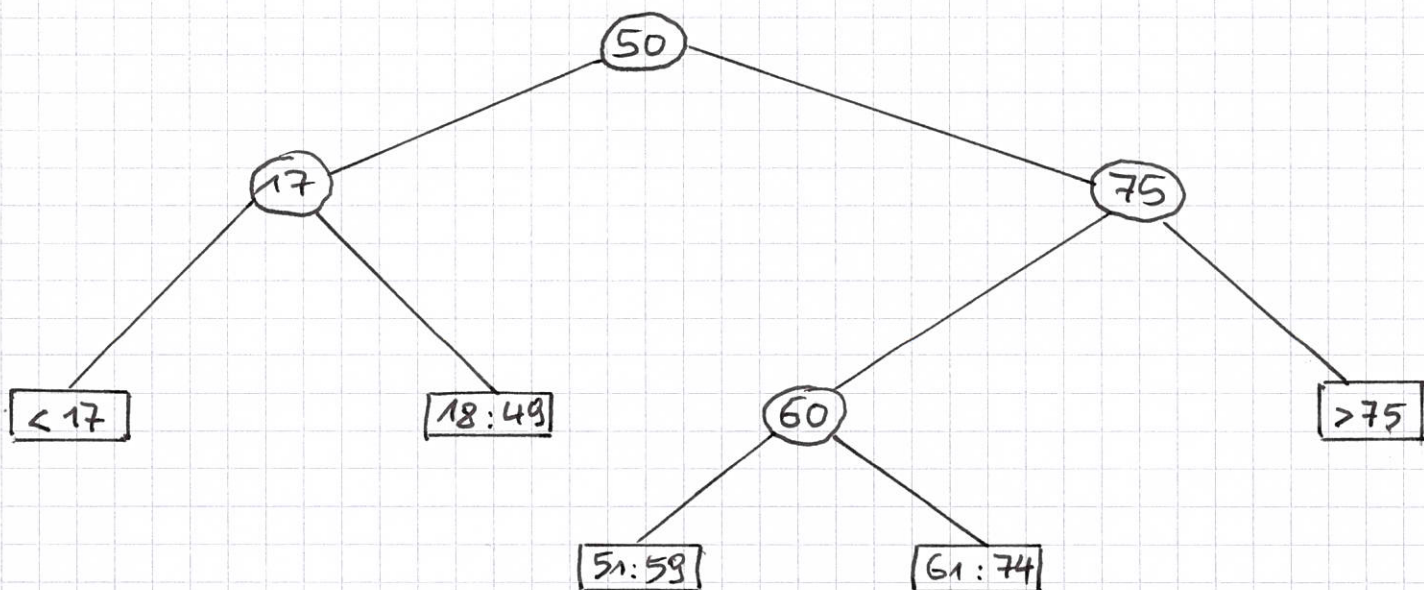
1.2 Binäre Suchbäume

Wiederholung:

geordnete Bäume, Binärbaum,
knotenorientierte Speicherung, symmetrische Ordnung,
Suchbaum, balancierter Suchbaum

Beispiel:

binärer Suchbaum T zur Speicherung einer Menge
 $S = \{17, 50, 60, 75\}$



Operationen auf T :

SUCCESS(x): Lokalisieren des Element $x \in S$.

UNSucACCESS(x): Entscheide, ob $x \in S$.

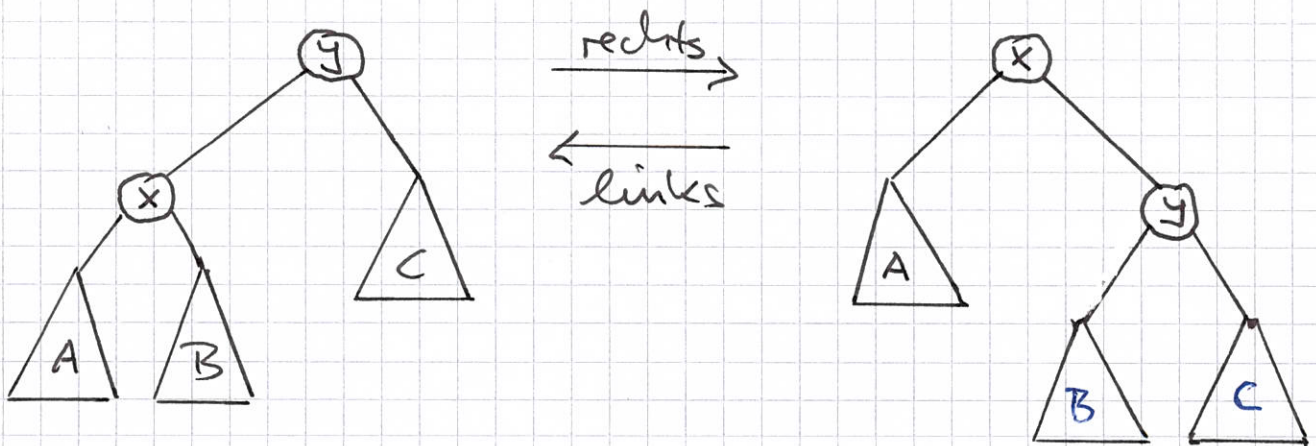
INSERT(x): Füge das neue Element x zu S hinzu. Der resultierende Baum erhält einen neuen Knoten, der x enthält, so dass dieser wieder ein Suchbaum ist.

DELETE(x): Streiche x aus S . Der resultierende Baum hat einen Knoten weniger.

SPLIT(x): Teile S in folgende zwei Mengen:
 $S_1 := \{y \in S \mid y \leq x\}$ und
 $S_2 := \{y \in S \mid y > x\}$. S_1 und S_2 sind in zwei Suchbäumen T_1 und T_2 gespeichert.

MELD(S_1, S_2): Inverse zu Split. Dabei gilt
 $\forall x \in S_1, \forall y \in S_2 : x < y$

Rotation:



Standard - Suchbaum - Modell:

SucACCESS(x):

Durchlaufen des Suchpfades von der Wurzel bis zum Knoten, der x enthält.

Kosten: Länge des Suchpfades + 1.

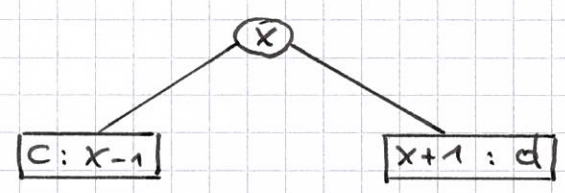
UNSucACCESS(x):

Durchlaufen des Suchpfades von der Wurzel bis zum Blatt $b(x)$, dessen Bereich x enthält.

Kosten: Länge des Suchpfades + 1

INSERT(x):

Ausführung von UNSucACCESS(x) nebst Ersetzung des Blattes $b(x)$ $[c:d]$ durch den Baum



Kosten: Länge des Suchpfades + 1

DELETE(x):

1. Phase: SucACCESS(x)

Kosten: Länge des Suchpfades + 1

2. Phase: Durchführung einer Anzahl von Rotationen

Kosten: 1 pro Rotation

3. Phase: Sei S_x der Teilbaum mit Wurzel x nach Beendigung der 2. Phase.

Seien

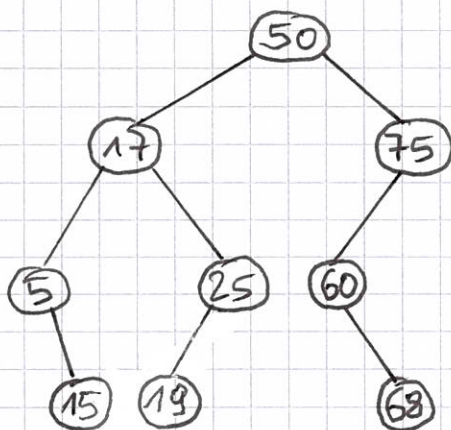
$$p := \begin{cases} \text{größtes Element } < x \\ \text{in } S_x & \text{falls solches existiert} \\ x & \text{sonst} \end{cases}$$

$$s := \begin{cases} \text{kleinstes Element } > x \\ \text{in } S_x & \text{falls solches existiert} \\ x & \text{sonst} \end{cases}$$

Der Algorithmus wählt $z := p$ oder $z := s$, vertauscht die Knoteninhalt der Knoten, die x und z enthalten und entfernt denjenigen Knoten, der nun x enthält.

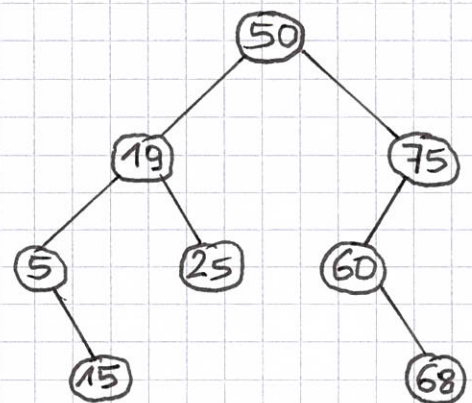
Kosten: Länge des Suchpfades von x nach z + 1 + 1 (für das Streichen).

Beispiel:



DELETE(17)

z.B.



Zwischen den Operationen kann der Algorithmus an beliebigen Knoten des Baumes eine beliebige Anzahl von Rotationen durchführen.

Kosten: 1 pro Rotation

Für einen Algorithmus ALG bezeichnet $ALG(\sigma, T_0)$ die Gesamtkosten für Zugriffe und Rotationen, die ALG bei Operationsfolge σ und Startbaum T_0 durchführt.

Wir betrachten zunächst das statische Datenverwaltungsproblem. Gegeben eine Menge S mit $|S| = n$ starten wir mit einem Anfangsbaum T_0 für S und führen eine Folge σ von ACCESS-Operationen durch. O.B.d.A. sei $S = \{1, 2, \dots, n\}$.

ALG heißt $f(n)$ -competitive für das statische Datenverwaltungsproblem, falls für alle ACCESS-Folgen σ und allen Startbäumen T_0

$$ALG(\sigma, T_0) \leq f(n) \cdot OPT(\sigma, T_0) + O(n).$$

Bemerkung:

- Für das dynamische Datenverwaltungsproblem sind balancierte Bäume $O(\log n)$ -competitive.
- Für das statische Datenverwaltungsproblem ist der vollständig ausgeglichene Baum $\log n$ -competitive.

Selbstorganisierende Suchbäume

- MTR (Move-to-root)

Plaziere nach $\text{ACCESS}(x)$ oder $\text{INSERT}(x)$ x mittels Rotationen in die Wurzel.

- SE (Simple exchange)

Nach $\text{ACCESS}(x)$ oder $\text{INSERT}(x)$ bringe mittels einer Rotation x um eine Position näher zur Wurzel.

Ziel: Analyse von MTR

Literatur:

Brian Allen, Ian Munro, Self-Organizing Binary Search Trees, JACM 25, 526 - 535 (1978).

Annahme:

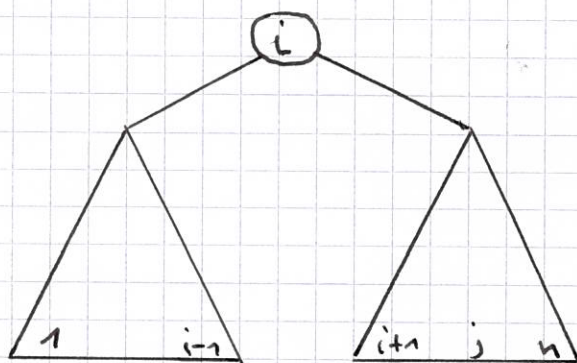
Die Operationen $\text{ACCESS}(i)$ und $\text{ACCESS}(j)$ sind in einem Baum T , der gemäß MTR modifiziert wird, erfolgt.

Lemma 1.4

Sei $i < j$. Dann gilt:

- i ist genau dann ein Vorgänger von j in T , wenn das letzte $\text{ACCESS}(i)$ nach dem letzten $\text{ACCESS}(k)$, $k \in \{i+1, i+2, \dots, j\}$ erfolgt ist.
- j ist genau dann ein Vorgänger von i in T , wenn das letzte $\text{ACCESS}(j)$ nach dem letzten $\text{ACCESS}(k)$, $k \in \{i, i+1, \dots, j-1\}$ erfolgt ist.

- a) Nach dem letzten $\text{ACCESS}(i)$ sieht der Suchbaum T wie folgt aus:



Es gilt: i ist Vorgänger von j .

Beh. (*):

- i) Falls i Vorgänger von j in T ist, dann ist i auch nach $\text{ACCESS}(k)$, wobei $k \in \{1, 2, \dots, i-1\} \cup \{j+1, j+2, \dots, n\}$ Vorgänger von j in T' , wobei T' der Suchbaum nach Durchführung von $\text{ACCESS}(k)$ ist.
- ii) Falls i Vorgänger von j in T ist, dann gilt nach $\text{ACCESS}(k)$, wobei $k \in \{i+1, i+2, \dots, j-1\}$, dass i weder Vorgänger noch Nachfolger von j in T' ist.
- iii) Falls i weder Vorgänger noch Nachfolger von j in T ist, dann gilt nach $\text{ACCESS}(k)$, wobei $k \in \{1, 2, \dots, n\} \setminus \{i, j\}$, dass i weder Vorgänger noch Nachfolger von j in T' ist.

Bew. d. Beh.:

Die Regel MTR bringt nach $\text{ACCESS}(k)$ mittels Rotationen auf dem Pfad vom Knoten mit Inhalt k zur Wurzel k in die Wurzel.

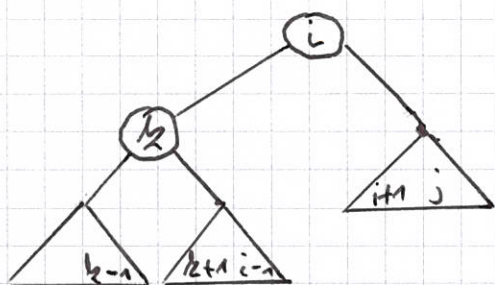
$\Rightarrow$

Teilbäume, die keinen Knoten mit P gemeinsam haben, werden durch Anwendung der Regel MTR nicht modifiziert.

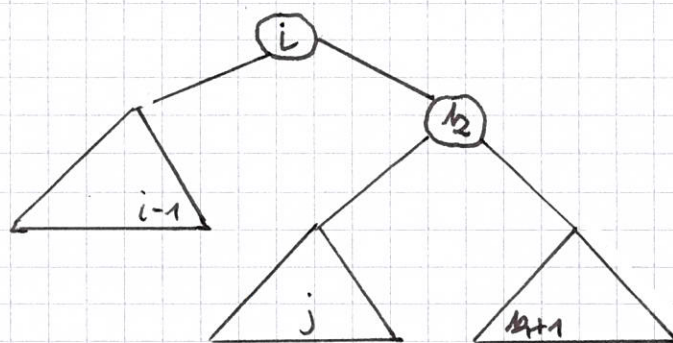
- i) Falls k nicht im Teilbaum von T mit Wurzel i enthalten ist, dann wird dieser Teilbaum nicht modifiziert, so dass die Behauptung direkt folgt.

Annahme: k ist im Unterbaum mit Wurzel i enthalten.

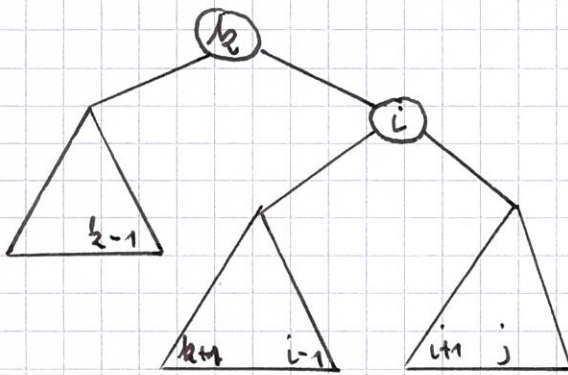
Dann macht die Regel MTR zunächst k zum Sohn von i . D.h., der Unterbaum mit Wurzel i sieht dann folgendermaßen aus:



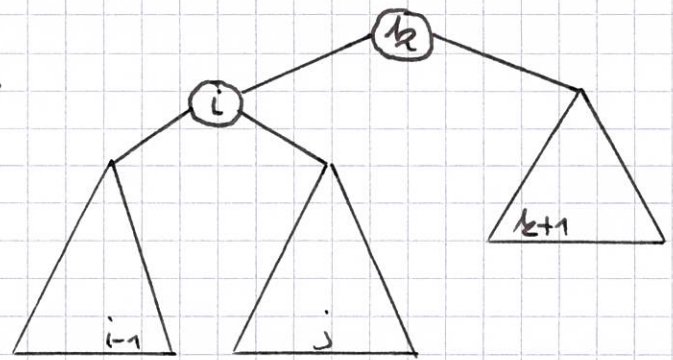
bzw.



Nach der nächsten Rotation erhalten wir:

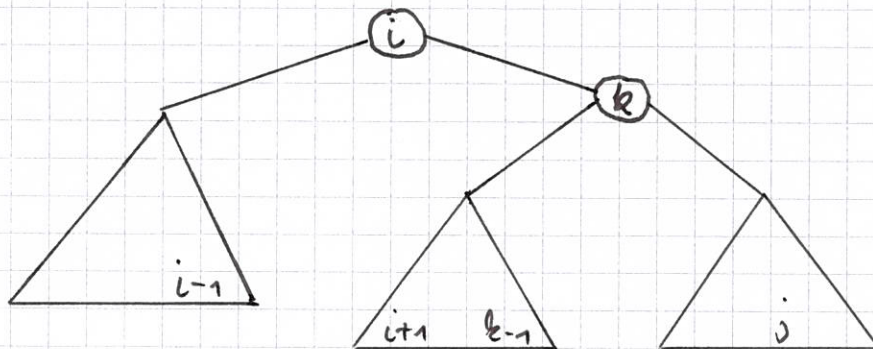


bzw.

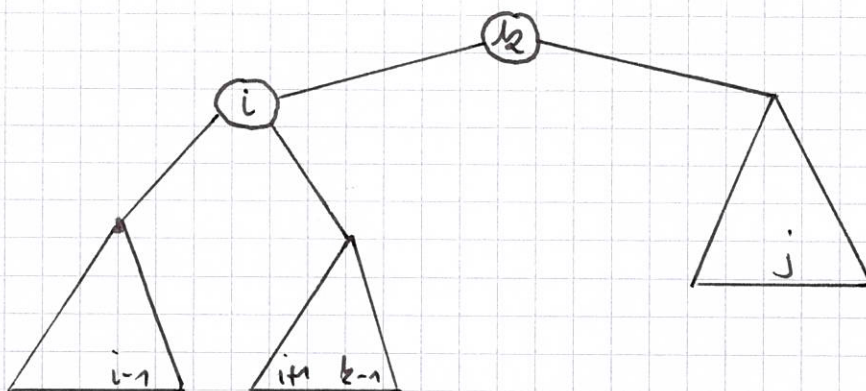


In beiden Fällen bleibt i Vorgänger von j .

- ii) Da i Vorgänger von j ist und $k \in \{i+1, \dots, j-1\}$ ist k im Unterbaum mit Wurzel i enthalten. MTR macht zunächst k zum Sohn von i . Der Unterbaum mit Wurzel i sieht dann wie folgt aus:



Nach der nächsten Rotation erhalten wir:



i ist dann weder Vorgänger noch Nachfolger von j .

iii) In der Vorlesung diskutieren.

□

Falls nach dem letzten $\text{ACCESS}(i)$ die Operation $\text{ACCESS}(j)$ nicht erfolgt, dann folgt die Behauptung α des Lemmas direkt aus obiger Beh. (*).

Es fehlt noch die Betrachtung, dass nach $\text{ACCESS}(i)$ die Operation $\text{ACCESS}(j)$ erfolgt. Diese ist jedoch implizit im Beweis von β drin.

b)

Übung

■

Lemma 1.5

Seien T und T' zwei beliebige binäre Suchbäume zur Speicherung einer Menge S mit $|S| = n$. Dann kann bei Verwendung von MTR der Suchbaum T aus T' mittels einer beliebigen Folge σ von n ACCESS -Operationen, die folgende Eigenschaften besitzt, konstruiert werden:

- i) Alle Elemente in S werden angefragt.
- ii) Wenn $\text{ACCESS}(x)$, $x \in S$ erfolgt, dann ist für jeden Nachfolger y von x in T die Operation $\text{ACCESS}(y)$ bereits durchgeführt.

Beweis:

O.B.d.A. sei $S = \{1, 2, \dots, n\}$. Betrachte $i < j$. Je nachdem, wie die relative Lage von i und j zueinander in T ist, unterscheiden wir drei Fälle.

a) i ist Vorgänger von j in T

Dann gilt für alle k mit $i < k \leq j$

- k ist im Unterbaum von T mit Wurzel i .

$\Rightarrow$

$\text{ACCESS}(i)$ erfolgt nach $\text{ACCESS}(k)$, $i < k \leq j$.

b) j ist Vorgänger von i in T .

Dann gilt für alle k mit $i \leq k < j$

- k ist im Unterbaum von T mit Wurzel j .

$\Rightarrow$

$\text{ACCESS}(j)$ erfolgt nach $\text{ACCESS}(k)$, $i \leq k < j$.

c) $\neg a) \wedge \neg b)$.

Dann existiert genau ein l mit $i < l < j$, i ist im linken Unterbaum von l und j ist im rechten Unterbaum von l in T .

$\Rightarrow$

$\text{ACCESS}(l)$ erfolgt nach $\text{ACCESS}(i)$ und nach $\text{ACCESS}(j)$.

Lemma 1.4 $\Rightarrow$ Beh. (Ausarbeitung als Übung.)

Ziel:

Konstruktion einer Anfragefolge σ der Länge n , deren mittlere Zugriffszeit bei einem beliebigen Startbaum und Verwendung von MTR $\geq n/8$ beträgt.

Bemerkung:

Falls die Menge S in einem balancierten Suchbaum (z.B. AVL-Baum) gespeichert ist, dann ist die worst case Zugriffszeit $O(\log n)$.

Idee:

Sorge dafür, dass der Suchbaum eine "Hälfte" der Länge $n/2$ enthält und arbeite diese ab.

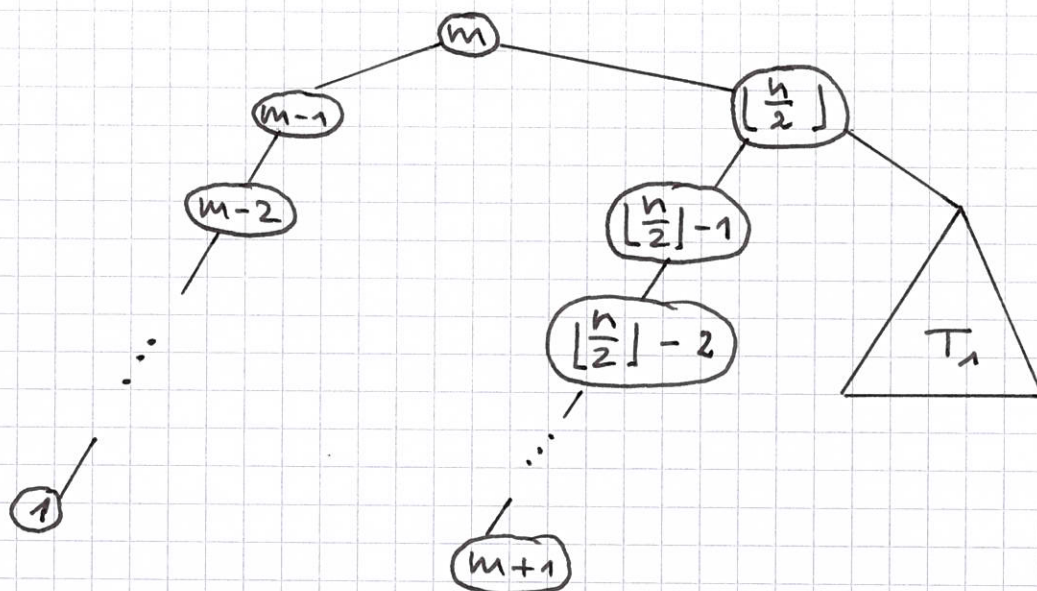
Satz 1.5

Sei T' ein beliebiger Suchbaum für die Menge $S = \{1, 2, \dots, n\}$. Sei

$$\sigma = \text{ACCESS}(1), \text{ACCESS}(2), \dots, \text{ACCESS}(\lfloor \frac{n}{2} \rfloor), \\ \text{ACCESS}(1), \text{ACCESS}(2), \dots, \text{ACCESS}(\lfloor \frac{n}{2} \rfloor).$$

Dann beträgt die mittlere Zugriffszeit mindestens $n/8$.

Sei $0 \leq m < \lfloor \frac{n}{2} \rfloor$. Dann folgt aus Lemma 1.4, dass nach genau $\lfloor \frac{n}{2} \rfloor + m$ ACCESS-Operationen der Suchbaum die folgende Struktur hat:



Also gilt für $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$:

- Die Zugriffszeit der $(\lfloor \frac{n}{2} \rfloor + i)$ -ten ACCESS-Operation $\text{ACCESS}(i)$ beträgt $\geq \lfloor \frac{n}{2} \rfloor - i + 1$.

Also gilt für die Gesamtzugriffszeit GZ :

$$\begin{aligned}
 GZ &\geq 2\lfloor \frac{n}{2} \rfloor - 1 + \lfloor \frac{n}{2} \rfloor + \sum_{i=2}^{\lfloor \frac{n}{2} \rfloor} i \\
 &\geq 3\frac{(n-1)}{2} - 1 + \frac{1}{2}\frac{(n-1)}{2}\frac{(n+1)}{2} - 1 \\
 &= \frac{n^2 + 12n - 29}{8} \\
 &> \frac{n^2}{8} \quad \text{für } n \geq 3.
 \end{aligned}$$

Mit Hilfe des voll ausgeglichenen binären Suchbaumes kann obige Operationsfolge mit worst case Zugriffszeit $\log n$ durchgeführt werden. Also gilt folgendes Korollar:

Korollar 1.1

Für das statische Datenverwaltungsproblem hat MTR ein competitives Verhältnis von mindestens $\frac{n}{2 \cdot \log n}$.

Bemerkung:

MTR bringt zwar das angefragte Element in die Wurzel, verbessert ansonsten aber die Struktur des Baumes nicht. So bleiben bei MTR lange Links- oder Rechtspfade auch lange Links- oder Rechtspfade.

Bsp.:



Splay - Bäume

Literatur:

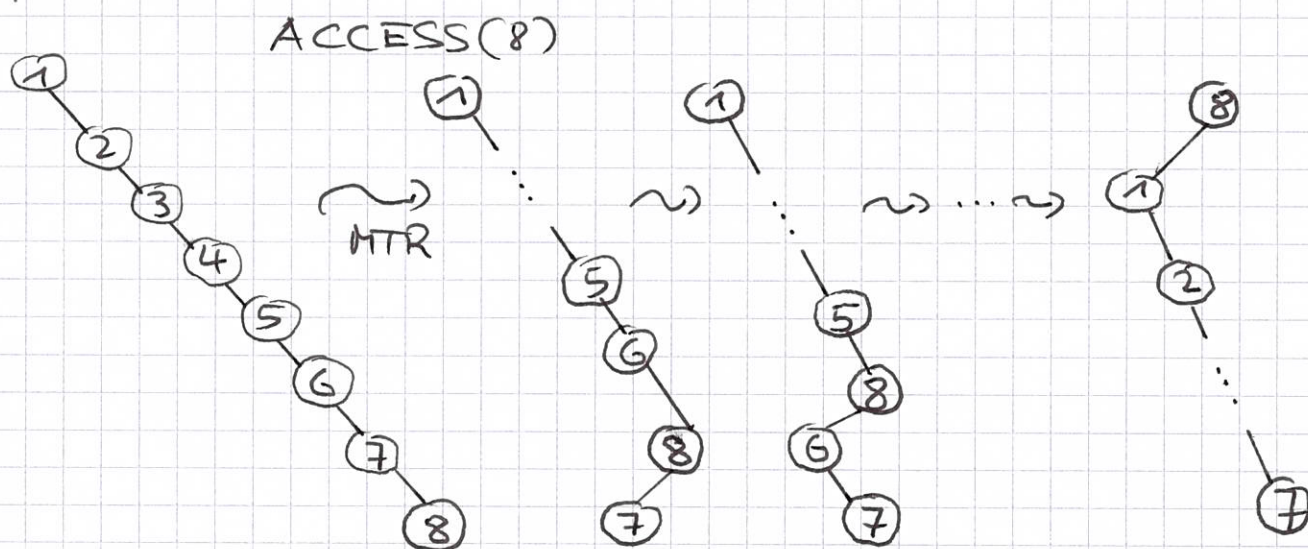
- D.D. Sleator, R.E. Tarjan, Self-Adjusting Binary Search Trees, JACM 22, 652 - 686 (1985).

- R.E. Tarjan, Data Structures and Network Algorithms, SIAM 1983, S. 53-57.
- M.T. Goodrich, R. Tamassia, Algorithm Design: Foundations, Analysis and Internet Examples, Wiley 2002, S. 185-194.

Idee:

- Verfeinerung von MTR, so dass lange Links- oder Rechtspfade nicht mehr in einen langen Links- oder Rechtspfad transformiert werden.

Betrachte hierzu noch einmal obiges Beispiel für $n = 8$.



Die Ursache dafür, dass MTR lange Links- bzw. Rechtspfade wieder in solche transformiert liegt darin, dass die Ordnung der Elemente auf solchen langen Pfaden erhalten bleibt.

Frage:

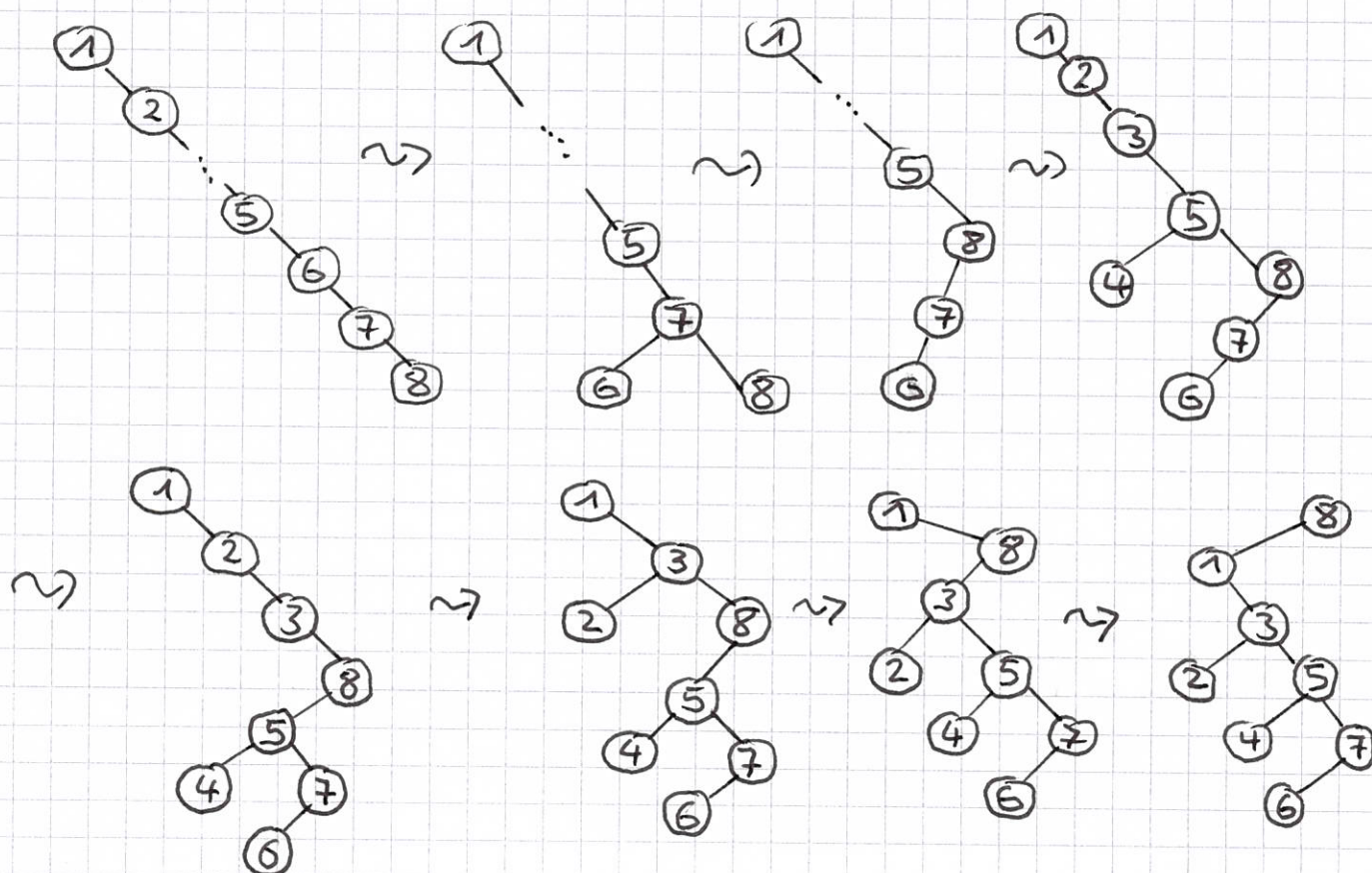
Können wir mittels Rotationen ein Element x über einen langen Links- oder Rechtspfad bewegen, ohne dass die Ordnung der Elemente auf diesem langen Pfad zu sehr erhalten bleibt?

Beobachtung:

MTR rotiert immer am Vater des Knotens, der nach oben gebracht werden soll.

Idee:

Rotiere zunächst am Großvater und anschließend am Vater des Knotens, der nach oben gebracht werden soll.

Bsp.:

Effekt:

Die Länge des Links- bzw. Rechtspades wird halbiert.

↪

Regel (x soll zur Wurzel gebracht werden):

(1) Betrachte x , Vater(x) und Großvater(x)

- Falls Großvater(x) existiert, dann
 - falls x , Vater(x) und Großvater(x) auf einem Links- oder Rechtspfad (d.h., $\text{Großvater}(x) > \text{Vater}(x) > x$ oder $\text{Großvater}(x) < \text{Vater}(x) < x$)

dann

rotiere am Großvater(x);
rotiere am Vater(x).

- andernfalls

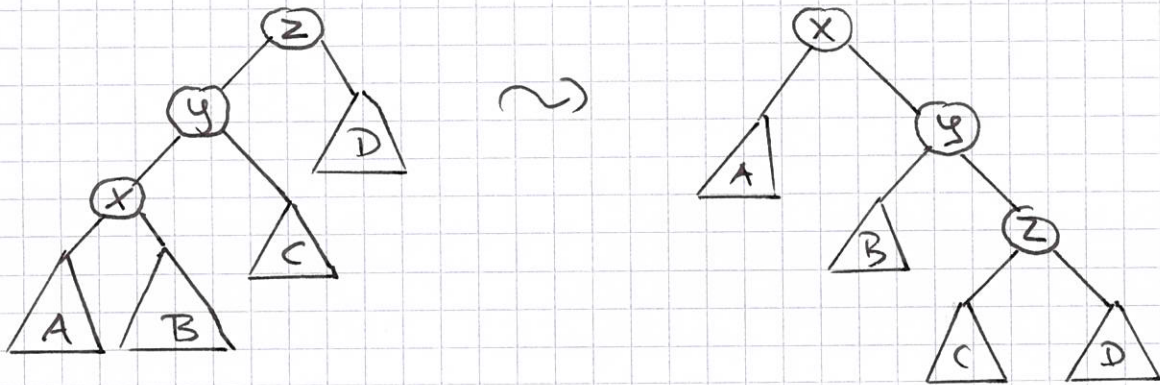
rotiere am Vater(x);
rotiere am Vater(x).

- Falls Großvater(x) nicht existiert, dann

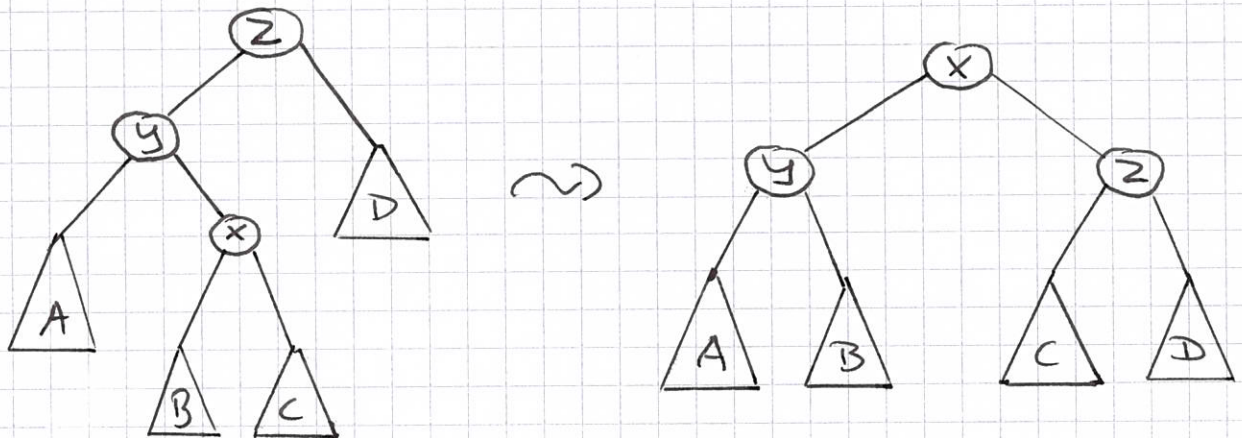
rotiere am Vater(x).

Die in obiger Regel vorkommenden Transformationen heißen Splay-Schritte. Diese können wie folgt graphisch dargestellt werden:

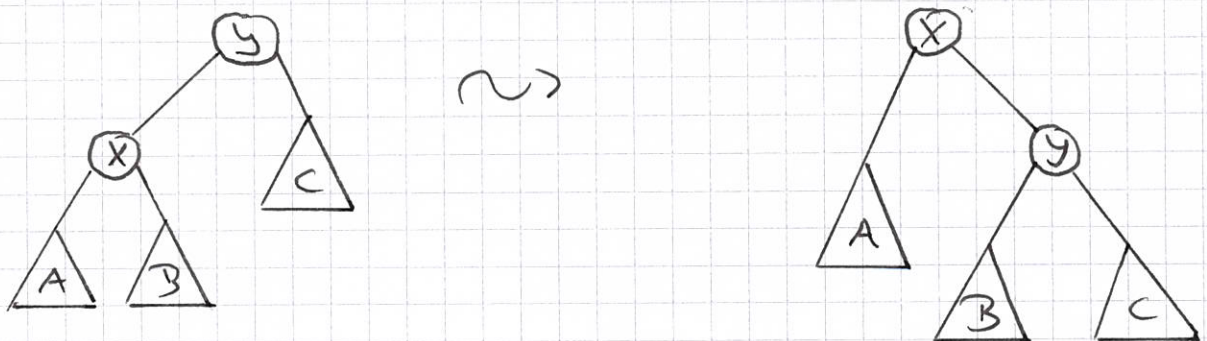
i) zig-zig:



ii) zig-zag



iii) zig



Hinzu kommen die symmetrischen Varianten.

Ziel:

Analyse von Splay-Bäumen mit Hilfe der Potentialfunktionmethode.

Hierzu ordnen wir jedem Item i ein Gewicht $w(i) > 0$ beliebig, aber fest zu. (68)

Sei für jeden Knoten x im Baum T die Größe $s(x)$ definiert durch

$$s(x) := \sum_{i \in T_x} w(i),$$

wobei T_x den Unterbaum von T mit Wurzel x bezeichnet.

Sei der Rang $r(x)$ definiert durch

$$r(x) := \log s(x).$$

Ferner sei

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

das Potential des Baumes T .

Wir definieren die Splay - Operation wie folgt.

Splay (x, T) ist das Bringen von x zur Wurzel in T unter Anwendung obiger Regel.

Die Kosten von $\text{Splay}(x, T)$ sind

$$\begin{cases} \text{Anzahl der Rotationen} \\ 1 \end{cases}$$

falls solche durchgeführt werden
sonst

Lemma 1.6 (ACCESS - Lemma):

Sei T ein Splay-Baum mit Wurzel t und x ein Knoten in T . Dann sind die amortisierten Kosten für $\text{Splay}(x, T)$

$$\leq 3(r(t) - r(x)) + 1 = O\left(\log \frac{s(t)}{s(x)}\right).$$

Beweis:

Falls keine Rotation durchgeführt wird, dann betragen die Kosten für $\text{Splay}(x, T)$ eins.

Da der Baum T dann nicht modifiziert wird, ändert sich auch sein Potential nicht.

Erinnerung:

Die amortisierten Kosten einer Operation ergeben sich aus der Summe der Kosten der Operation und der Potentialänderung.

$\Rightarrow$

Die amortisierten Kosten von $\text{Splay}(x, T)$ sind 1.

Da x die Wurzel von T ist, gilt

$$3(r(t) - r(x)) + 1 = 1.$$

Also gilt die Behauptung.

Annahme:

Mindestens eine Rotation wird durchgeführt.

Betrachten wir einen beliebigen Splay-Schritt.
Seien s und s' bzw. r und r' die Größe-
bzw. die Rangfunktionen vor und nach
dem Splay-Schritt.

Ziel:

Beweis, dass die amortisierten Kosten für
diesen Schritt

$$\leq \begin{cases} 3(r'(x) - r(x)) + 1 & \text{falls zig} \\ 3(r'(x) - r(x)) & \text{falls zigzig oder zigzag} \end{cases}$$

sind.

Seien vor dem Splay-Schritt $y = \text{Vater}(x)$
und $z = \text{Vater}(y)$, falls $\text{Vater}(y)$ existiert.

Wir analysieren nun die drei Fälle nacheinander.

zig:

Es wird eine Rotation durchgeführt. Da
nur x und y ihren Rang ändern können
gilt für die amortisierten Kosten für diesen
Schritt

$$\begin{aligned} &= 1 + r'(x) + r'(y) - r(x) - r(y) \\ &\leq 1 + r'(x) - r(x) \\ &\text{wegen } r(y) \geq r'(y) \\ &\leq 1 + 3(r'(x) - r(x)) \\ &\text{wegen } r'(x) \geq r(x) \end{aligned}$$

Zigzag:

Es werden zwei Rotationen durchgeführt. Nur x , y und z können ihren Rang ändern. Also gilt für die amortisierten Kosten dieses Schrittes

$$= 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

$$= 2 + r'(y) + r'(z) - r(x) - r(y)$$

da $r'(x) = r(z)$

$$\leq 2 + r'(x) + r'(z) - 2r(x)$$

da $r'(x) \geq r'(y)$
und $r(y) \geq r(x)$

Also genügt es

$$2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$$

zu zeigen. Dies ist äquivalent zu

$$2r'(x) - r(x) - r'(z) \geq 2$$

$$\Leftrightarrow r(x) + r'(z) - 2r'(x) \leq -2$$

Nach Definition gilt

$$r(x) + r'(z) - 2r'(x) = \log s(x) + \log s'(z) - 2 \log s'(x)$$

$$= \log \frac{s(x)}{s'(x)} + \log \frac{s'(z)}{s'(x)}$$

Wegen $s'(x) > s(x)$ und $s'(x) > s'(z)$ gilt

$$0 < \frac{s(x)}{s'(x)}, \frac{s'(z)}{s'(x)} < 1.$$

Wegen $s(x) + s'(z) \leq s'(x)$ gilt

$$\frac{s(x) + s'(z)}{s'(x)} \leq 1.$$

Betrachten wir die Funktion

$$f(x, y) = \log x + \log y$$

auf dem Bereich $x, y > 0$, $x + y \leq 1$.

Da die Logarithmusfunktion streng monoton wächst kann die Funktion $f(x, y)$ ihr Maximum nur auf dem Geradensegment

$$x + y = 1, \quad x, y > 0$$

annehmen. Also ergibt sich das Maximum der Funktion $f(x, y)$ aus dem Maximum der Funktion

$$g(x) = \log x + \log(1-x).$$

Lösung

Zeigen Sie, dass die Funktion $g(x) = \log x + \log(1-x)$ ihr Maximum an $x = \frac{1}{2}$ annimmt.

Also nimmt $\log x + \log y$ ihr Maximum für $x = y = \frac{1}{2}$ an. Es gilt

$$\log \frac{1}{2} + \log \frac{1}{2} = -2$$

Also gilt

$$\log \frac{s(x)}{s'(x)} + \log \frac{s'(z)}{s'(x)} \leq -2.$$

zig zag:

Es werden zwei Rotationen durchgeführt. Nur x, y und z können ihren Rang ändern. Somit gilt für die amortisierten Kosten dieses Schrittes

$$= 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

$$\leq 2 + r'(y) + r'(z) - 2r(x)$$

da $r'(x) = r(z)$
und $r(x) \leq r(y)$

Also genügt es

$$2 + r'(y) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$$

zu zeigen. Dies ist äquivalent zu

$$3r'(x) - r(x) - r'(y) - r'(z) \geq 2.$$

Wegen $r'(x) \geq r(x)$ genügt es

$$2r'(x) - r'(y) - r'(z) \geq 2$$

zu beweisen. (Übung)

Die amortisierten Kosten für $\text{Splay}(x, T)$ ergeben sich aus der Summe der amortisierten Kosten aller Splay -Schritte, die $\text{Splay}(x, T)$ durchführt. Dies ist eine sogenannte "Teleskopsumme", so dass nur die beiden äußersten Summanden übrig bleiben. Somit gilt für die amortisierten Kosten für $\text{Splay}(x, T)$

$$\leq 3(r'(x) - r(x)) + 1 = 3(r(t) - r(x)) + 1,$$

wobei r bzw. r' die Rangfunktionen vor bzw.

nach der gesamten Splay-Operation sind.

Bemerkung:

Die Gewichte der Items sind Parameter der Analyse. Der Algorithmus hängt nicht von den Gewichten ab. D.h., Lemma 1.6 gilt für jede beliebige Wahl von positiven Gewichten für die Items.

Ziel: Anwendungen des ACCESS-Lemmas.

Betrachten wir eine Folge von m Zugriffsoperationen auf einen Splay-Baum T mit n Knoten. Sei

$$W := \sum_{i=1}^n w(i).$$

Falls die Gewichte der Items fest bleiben, dann gilt stets für die Größe $s(x_i)$ des Knotens, der das Item i enthält

$$w(i) \leq s(x_i) \leq W.$$

Also reduziert sich das Potential des Baumes T durch die Folge von m Zugriffsoperationen höchstens um

$$\begin{aligned} & \sum_{i=1}^n (\log W - \log w(i)) \\ &= \sum_{i=1}^n \log \frac{W}{w(i)}. \end{aligned}$$

Satz 1.6 (Balance Theorem)

Sei T ein beliebiges Splay-Baum mit n Knoten.
 Dann betragen die Gesamtkosten für m
 ACCESS-Operationen $O((m+n) \log n + m)$.

Beweis:

Wähle für jedes Item das Gewicht $\frac{1}{n}$. Dann gilt

$$W = 1.$$

ACCESS-Lemma $\Rightarrow$

Die amortisierte Kosten sind für jede ACCESS-Operation

$$\begin{aligned} &\leq 3 \left(\log 1 - \log \frac{1}{n} \right) + 1 \\ &= 3 \cdot \log n + 1 \end{aligned}$$

Die Gesamtreduktion des Potentials ist

$$\leq \sum_{i=1}^n \log \frac{W}{w(i)} = \sum_{i=1}^n \log n = n \cdot \log n.$$

Also gilt für die Gesamtkosten

$$\leq m(1 + 3 \cdot \log n) + n \cdot \log n$$

Interpretation

Bei einer genügend langen Folge von Zugriffsoperationen sind Splay-Bäume genauso effizient wie ein balancierter Baum.

Satz 1.7 (Statistisches Optimalitätstheorem)

(76)

Sei $q(i)$, $i \in \{1, 2, \dots, n\}$ die Anzahl der Zugriffe auf das Item i . Falls auf jedes Item mindestens einmal zugegriffen wird, dann betragen die Gesamtkosten der m ACCESS-Operationen $O(m + \sum_{i=1}^n q(i) \log \frac{m}{q(i)})$.

Beweis:

Wähle für jedes Item i das Gewicht

$$w(i) := \frac{q(i)}{m}.$$

Dann gilt $W = 1$.

ACCESS-Lemma $\Rightarrow$

Die amortisierte Kosten pro Zugriff sind für jedes Item i

$$\begin{aligned} &\leq 3 \left(\log 1 - \log \frac{q(i)}{m} \right) + 1 \\ &= 3 \cdot \log \frac{m}{q(i)} + 1. \end{aligned}$$

Die Gesamtreduktion des Potentials ist

$$\leq \sum_{i=1}^n \log \frac{W}{w(i)} = \sum_{i=1}^n \log \frac{m}{q(i)}.$$

Also gilt für die Gesamtkosten

$$\begin{aligned} &\leq m + \sum_{i=1}^n q(i) \cdot 3 \log \frac{m}{q(i)} + \sum_{i=1}^n \log \frac{m}{q(i)} \\ &= O\left(m + \sum_{i=1}^n q(i) \log \frac{m}{q(i)}\right). \end{aligned}$$



Bemerkung:

Mit Hilfe von Methoden der Informationstheorie kann gezeigt werden, dass in einem optimalen statischen Suchbaum (d.h., dass dieses nicht verändert wird), die Tiefe eines Items i , auf das mit der relativen Häufigkeit $\frac{q(i)}{m}$ zugegriffen wird, $\Omega\left(\log \frac{m}{q(i)}\right)$ ist. Somit besagt das statische Optimalitätstheorem, dass Splay-Bäume, bis auf einen konstanten Faktor, genauso effizient sind, wie optimale statische Suchbäume.

(siehe

Kurt Mehlhorn, Datenstrukturen und effiziente Algorithmen, Band 1: Sortieren und Suchen, Teubner 1988, S. 161 - 164.)

Seien $S = \{1, 2, \dots, n\}$ und $\sigma = \text{ACCESS}(i_1), \text{ACCESS}(i_2), \dots, \text{ACCESS}(i_m)$ die Folge der m Zugriffsoperationen.

Satz 1.8 (Statisches Fingertheorem)

Sei f ein beliebiges festes Item. Dann betragen die Gesamtkosten der m Zugriffsoperationen $O\left(n \log n + m + \sum_{j=1}^m \log(|i_j - f| + 1)\right)$.

Beweis:

Wähle für jedes Item i das Gewicht

$$w(i) := \frac{1}{(|i - f| + 1)^2}.$$

Dann gilt

$$\begin{aligned} W &\leq 2 \cdot \sum_{k=1}^n \frac{1}{k^2} \\ &\leq 2 \cdot \sum_{k=1}^{\infty} \frac{1}{k^2} \\ &= 2 \cdot \frac{\pi^2}{6} = \frac{\pi^2}{3} < 4 \end{aligned}$$

Für die amortisierten Kosten der j -ten Zugriffsoperation gilt

$$\leq 3(2 + 2 \cdot \log(|i_j - f| + 1)) + 1$$

Die Gesamtreduktion des Potentials ist

$$\begin{aligned} &\leq \sum_{i=1}^n \log \frac{W}{w(i)} \\ &= O(n \cdot \log n) \quad \left(\text{da } \forall i \text{ gilt } w(i) \geq \frac{1}{n^2} \right) \end{aligned}$$

Also gilt für die Gesamtkosten

$$\leq O(n \log n + m + \sum_{j=1}^m \log(|i_j - f| + 1))$$

Etwas zu Fingerbäume sagen.

Das statische Fingertheorem besagt, dass Splay-Bäume, bis auf einen konstanten Faktor, genauso effizient wie statische Fingerbäume sind.

Die letzten beiden Sätze zeigen, dass Splay-Bäume höchstens um einen konstanten Faktor schlechter als optimale statische binäre Suchbäume sind. Bezüglich des Vergleichs mit dynamischen Suchbäumen haben Sleator und Tarjan folgende Vermutungen aufgestellt:

Vermutung 1: (Dynamische Optimalitätsvermutung)
Betrachte eine beliebige Folge von erfolgreichen Zugriffoperationen in einem binären Suchbaum mit n Knoten. Sei A ein beliebiger Algorithmus, der jedes $\text{ACCESS}(i)$ mittels Betrachtung des Suchpfades von der Wurzel bis zum Knoten x , der i enthält, mit Kosten Länge des Suchpfades $+ 1$ durchführt. Zwischen Zugriffoperationen kann A eine beliebige Anzahl von Rotationen an beliebigen Knoten des Suchbaumes mit Kosten 1 pro Rotation durchführen. Dann ist die Gesamtzeit, die ein Splay-Baum für diese Folge von Zugriffoperationen benötigt $O(n + T(A))$, wobei $T(A)$ die von Algorithmus A benötigte Gesamtzeit ist.

Vermutung 2: (Dynamische Fingervermutung).
Die Gesamtzeit um eine Folge von m erfolgreichen Zugriffoperationen in einem beliebigen n -Knoten Splay-Baum durchzuführen ist

$$O(m + n + \sum_{j=1}^m \log(|i_{j+1} - i_j| + 1)),$$

(80)

wobei für $1 \leq j \leq m$ die j -te Zugriffsoperation $\text{ACCESS}(i_j)$ und $|i_{j+1} - i_j|$ der Abstand von i_{j+1} und i_j bzgl. der symmetrischen Ordnung sind.

Bemerkung:

Die dynamische Fingervermutung ist mittlerweile bewiesen:

R. Cole, B. Mishra, J. Schmidt, A. Siegel, On the Dynamic Finger Conjecture for Splay Trees. Part I: Splay Sorting $\log n$ - Block Sequences, SIAM J. Comput. 30(2000), 1-43.

R. Cole, On the Dynamic Finger Conjecture for Splay Trees. Part II: The Proof, SIAM J. Comput. 30(2000), 44-85.

2. Paging

weitere Literatur:

R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995, Chapter 13.

Modell: (virtueller Speicher)

- Kernspeicher K der Größe k
- Hintergrundspeicher H der Größe N .

81
Eine Anfrage bzgl. Seite p verlangt, dass $p \in K$. D.h.,

- falls $p \in K$ ✓
- falls $p \notin K$ und noch hinreichend Platz in K , dann wird p geladen.
- falls $p \notin K$ und kein Platz in K , dann muss eine Seite $q \in K$ bestimmt werden, Diese wird dann in den Hintergrund Speicher abgelegt um Platz für p zu schaffen. Danach wird p geladen.

Kosten hierfür: 1

Frage:

Wie soll q gewählt werden?

Zur Beantwortung dieser Frage werden wir sowohl deterministische als auch randomisierte Methoden kennenlernen.

2.1 Deterministische Algorithmen

Heuristiken:

- LRU (least-recently-used)

Ersetze diejenige Seite, die am längsten nicht mehr angefragt worden ist.

- FIFO (first-in-first-out)

Ersetze diejenige Seite, die am längsten im Kernspeicher ist.

- LIFO (last-in-first-out)

Ersetze diejenige Seite, die am kürzesten im Kernspeicher ist.

- LFU (least-frequently-used)

Ersetze diejenige Seite, die am wenigsten angefragt wurde, seitdem sie im Kernspeicher ist.

- LFD (longest-forward-distance)

Ersetze diejenige Seite, deren nächste Anfrage am spätesten erfolgen wird

Bemerkung:

LFD ist eine offline-Heuristik. Alle anderen Heuristiken sind online.

Lemma 2.1

Sei ALG ein Pagingalgorithmus (online oder offline). Dann können die Seitenaustausche in ALG derart verschoben werden, dass die Gesamtkosten sich nicht vergrößern und eine Seite erst dann in den Hauptspeicher geladen wird, wenn sie auch angefragt wird.

Beweis:

Hierzu verschiebt man einen Seitenaustausch unmittelbar vor die nächste Anfrage derjenigen Seite, die gerade geladen werden soll. Man kann sich leicht überlegen, dass sich dadurch die Kosten höchstens verringern können.

Übung:

Arbeiten Sie den Beweis von Lemma 2.1 aus.

Eine LFD-Folge ist eine Folge von Seitenaustauschen, wobei die aus dem Hauptspeicher zu verdrängende Seite stets diejenige Seite ist, die auch mittels der LFD-Heuristik ermittelt worden wäre.

Wir können jeden Pagingalgorithmus derart interpretieren, dass dieser zunächst mit einer (möglicherweise leeren) maximalen LFD-Folge startet. Maximal bedeutet, dass der nachfolgende Seitenaustausch eine andere Seite zur Verdrängung aus dem Hauptspeicher auswählt, als diejenige, die aufgrund der LFD-Heuristik ausgewählt werden würde.

Satz 2.1

LFD ist ein optimaler Offline-Algorithmus.

Beweis:

Sei σ eine beliebige Anfragefolge und sei ALG ein optimaler Pagingalgorithmus bzgl. σ , für den gilt:

1. ALG erfüllt Lemma 2.1.
2. ALG startet mit einer längsten LFD-Folge unter allen optimalen Paging-Algorithmen.

Falls die Startfolge alle Seitenaustausche enthält, dann gilt $\text{ALG} = \text{LFD}$ bzgl. σ , womit nichts mehr zu beweisen wäre.

Annahme: $\text{ALG} \neq \text{LFD}$ bzgl. σ .

Mit $p \leftrightarrow q$ bezeichnen wir den Seitenaustausch, der die Seite q aus dem Kernspeicher verdrängt und die Seite p in den Kernspeicher lädt.

Seien

$x \leftrightarrow v$ derjenige Seitenaustausch, der auf die Startfolge in ALG folgt und

$x \leftrightarrow u$ derjenige Seitenaustausch, den statt dessen LFD durchführen würde.

Idee:

Konstruiere aus ALG einen optimalen Pagingalgorithmus ALG^* mit längerer LFD-Startfolge, was ein Widerspruch zur Eigenschaft 2 von ALG sein würde.

Durchführung:

ALG^* führt zunächst die Startfolge von ALG aus. Dann wird anstatt dem von ALG durchgeführten Seiten austausch $x \leftrightarrow v$ der von LFD durchgeführte Seiten austausch $x \leftrightarrow u$ ausgeführt. Im Anschluss daran verfährt ALG^* wie ALG bis in ALG ein Seiten austausch $y \leftrightarrow u$ oder $v \leftrightarrow z$ erfolgt. ALG^* ersetzt $y \leftrightarrow u$ durch $y \leftrightarrow v$ bzw. $v \leftrightarrow z$ durch $u \leftrightarrow z$. Danach verfährt ALG^* genauso wie ALG .

Bemerkung:

Falls $v \leftrightarrow z$ durch $u \leftrightarrow z$ ersetzt wird, dann erfüllt ALG^* nicht mehr Lemma 2.1. Wir schieben dann den Seiten austausch $u \leftrightarrow z$ vor die nächste Anfrage bzgl. u , so dass ALG^* danach Lemma 2.1 erfüllt.

Konstruktion $\Rightarrow$

Anfrage auf u erfolgt nach Anfrage auf v und somit nach dem Seiten austausch $v \leftrightarrow z$.

$\Rightarrow$

ALG^* löst das durch die Anfragefolge σ definierte Pagingproblem korrekt.

Ferner gilt: $ALG^*(\sigma) = ALG(\sigma)$.

Aber: ALG^* startet mit einer strikt längeren LFD -Folge und erfüllt

Widerspruch!

Also ist die Annahme $ALG \neq LFD$ bzgl. σ falsch.

■

(h, k) -Pagingproblem: $h \leq k$

Online-Algorithmus mit Kernspeicher der Größe k .
competitive Analyse mit

Offline-Algorithmus mit Kernspeicher der Größe h .

Markierungsalgorithmen:

Kernspeichergroße k , Anfragefolge σ beliebig, aber fest.

Idee:

Unterteilung von σ in Phasen, die sogenannte
 k -Phasenunterteilung.

Phase i ist wie folgt definiert:

$i = 0$: die leere Anfragefolge.

$i > 0$: die maximale auf Phase $i-1$ folgende
Anfragefolge σ' , für die gilt:

σ' enthält $\leq k$ verschiedene Seiten =
anfragen.

Bemerkung:

Die k -Phasenunterteilung für σ ist eindeutig bestimmt.

Markierungsalgorithmen arbeiten nach folgenden Spielregeln:

1. Jede Seite erhält ein Markierungsbit.
2. Vor Beginn einer Phase werden alle Markierungen entfernt.
3. Wenn während einer Phase eine Seite angefragt wird, dann wird diese markiert.
4. Es wird niemals während einer Phase eine Seite in den Kernspeicher geladen, die auch nicht während derselben Phase angefragt wird.
5. Es wird niemals eine markierte Seite aus dem Kernspeicher verdrängt.

Satz 2.2

Sei ALG ein beliebiger Markierungsalgorithmus bzgl. eines Kernspeichers der Größe k . Dann ist $ALG \frac{k}{k-h+1}$ -competitive bzgl. Offline-Algorithmen, die einen Kernspeicher der Größe $h \leq k$ verwenden.

Beweis:

Fixiere eine beliebige Anfragefolge σ und betrachte die korrespondierende k -Phasenunterteilung.

Beh.:

Für jede Phase $i \geq 1$ verursacht ALG $\leq k$ Seitenwechsel.

Bew. d. Beh.:

Bis auf die letzte Phase gibt es in jeder Phase $i > 0$ k verschiedene Seitenanfragen. Sobald eine solche erfolgt, wird die betreffende Seite markiert und kann während der Phase nicht verdrängt werden. Jede Seite, die während einer Phase in den Kernspeicher geladen wird, wird auch in derselben Phase angefragt. Also folgt die Beh. □

Sei OPT ein optimaler Offline-Algorithmus bzgl. der Anfragefolge σ mit Kernspeichergröße k . O.B.d.A. erfülle OPT Lemma 2.1.

Idee:

Ordne, bis auf die letzte, jeder Phase $i \geq 1$ mindestens $k - (k-1) = k - k + 1$ der von OPT durchgeführten Seitenwechsel zu.

Durchführung:

Betrachte beliebige Phase $i \geq 1$ ungleich der letzten Phase. Sei q die erste in Phase i angefragte Seite.

Betrachte die Anfragefolge σ , die mit der zweiten Seitenanfrage der Phase i beginnt und mit der ersten Seitenanfrage der Phase $(i+1)$ endet.

Konstruktion $\Rightarrow$

- $\mathcal{I}$ enthält k unterschiedliche Seitenanfragen, die alle eine Markierung der betreffenden Seiten bedingen.
- Da zu Beginn von Phase $(i+1)$ die Seite q sich im Kernspeicher befindet und die erste Seitenanfrage der Phase $(i+1)$ einen Seitenwechsel bedingt, befindet sich unter den k unterschiedlichen Seitenanfragen von $\mathcal{I}$ die Seite q nicht.
- Zu Beginn von $\mathcal{I}$ befindet sich die Seite q auch im Kernspeicher von OPT.

$\Rightarrow$

OPT führt während der Anfragefolge $\mathcal{I}$ $\geq k - (k-1)$ Seitenwechsel durch.

Diese ordnen wir der Phase i zu.

Die Nichtüberlappung der verschiedenen Phasen zugeordneten Anfragefolgen $\Rightarrow$

Ein Seitenwechsel des Algorithmus OPT wird maximal einer Phase zugeordnet.

Also gilt insgesamt

$$\text{ALG}(\sigma) \leq \frac{k}{k-k+1} \text{OPT}(\sigma) + \alpha,$$

wobei $\alpha \leq k$ die maximale Anzahl der von ALG während der letzten Phase durchgeführten Seitenwechsel ist.



Satz 2.3

LRU ist ein Markierungsalgorithmus.

Beweis:

Fixiere eine beliebige Anfragefolge σ und betrachte ihre k -Phasenunterteilung.

Annahme:

LRU ist kein Markierungsalgorithmus bzgl. σ .

$\Rightarrow$

LRU verdrängt während einer Phase eine markierte Seite x .

Betrachte die erste Anfrage bzgl. x während dieser Phase.

$\Rightarrow$

- x wird markiert.
- x ist unmittelbar nach dieser Anfrage diejenige Seite im Hauptspeicher, die als letzte angefragt worden ist.

$\Rightarrow$

Damit x aus dem Kernspeicher verdrängt werden kann, müssen in derselben Phase mindestens k weitere unterschiedliche Seiten angefragt werden.

$\Rightarrow$

Die betrachtete Phase enthält Anfragen bzgl.
 $\geq k+1$ verschiedenen Seiten.

Dies ist ein Widerspruch zur Definition der k -Phasenunterteilung.

$\Rightarrow$

Die Annahme war falsch.



Satz 2.4

FIFO ist kein Markierungsalgorithmus.

Beweis:

Übung



Ziel:

Beweis, dass FIFO $\frac{k}{k-n+1}$ -competitive ist.

Ein Pagingalgorithmus ALG heißt konservativ, falls er folgende Eigenschaft besitzt:

Während einer beliebigen Anfragefolge mit $\leq k$ verschiedenen Seitenanfragen hat ALG $\leq k$ Seitenfehler (d.h., die angefragte Seite befindet sich nicht im Kernspeicher).

Satz 2.5

Sei ALG ein beliebiger konservativer Online-Pagingalgorithmus bzgl. eines Kernspeichers der Größe k . Ferner sei OPT ein beliebiger optimaler Offline-Algorithmus bzgl. eines Kernspeichers der Größe $h \leq k$, welcher zu Beginn eine Teilmenge des Kernspeicherinhaltes von ALG enthält. Dann gilt für alle Anfragefolgen σ

$$\text{ALG}(\sigma) \leq \frac{k}{k-h+1} \text{OPT}(\sigma).$$

Beweis:

Analog zum Beweis von Satz 2.2. Dort haben wir auch nur verwendet, dass die Anzahl der Seitenwechsel bei k unterschiedlichen Seitenanfragen durch k beschränkt ist.

Übung:

- Arbeiten Sie den Beweis von Satz 2.5 aus.
- Zeigen Sie, dass LRU und FIFO konservativ sind.

Ziel: Beweis einer unteren Schranke.

Hierzu beweisen wir zunächst eine obere Schranke

für den optimalen Offline-Algorithmus LFD.

Lemma 2.2

Sei k die Größe des Kernspeichers. Dann gilt für jede endliche Anfragefolge σ , die stets eine Seite aus einer Menge von $k+1$ Seiten anfragt

$$\text{LFD}(\sigma) \leq \frac{|\sigma|}{k}.$$

Beweis:

Jedes Mal, wenn ein Seitenwechsel notwendig ist, verdrängt LFD diejenige Seite p mit dem größten Abstand zur nächsten Anfrage.

$\Rightarrow$

Vor der nächsten Anfrage bzgl. p erfolgen bzgl. den verbleibenden $k-1$ Seiten jeweils mindestens eine Anfrage.

$\Rightarrow$

Pro Anfrage mit Seitenwechsel können wir mindestens $k-1$ Anfragen ohne Seitenwechsel zählen. ■

Satz 2.6

Sei k die Größe des Kernspeichers und sei ALG ein beliebiger Online-Pagingalgorithmus. Dann gibt es eine Anfragefolge σ beliebiger Länge, so dass $\text{ALG}(\sigma) \geq k \cdot \text{LFD}(\sigma)$.

Beweis:

Lemma 2.2 $\Rightarrow$

Es genügt bzgl. $k+1$ Seiten $p_1, p_2, \dots, p_{k+1}$ eine beliebig lange Anfragefolge σ mit $ALG(\sigma) = |\sigma|$ zu konstruieren

Zu jedem Zeitpunkt ist exakt eine Seite nicht im Hauptspeicher. Der Gegenspieler fragt immer diese Seite an.

$\Rightarrow$

Jede Anfrage in σ verursacht für ALG einen Seitenwechsel.

$\Rightarrow$

$$ALG(\sigma) = |\sigma|.$$

Übung:

Beweisen Sie, dass sowohl für LIFO als auch für LFU keine Konstante c existiert, so dass der Online-Algorithmus c -competitiv ist.

Bemerkung:

In

Barodin, El-Yaniv: Online Computation and Competitive Analysis, S. 40-41

wird der Einfluss des Kostenmodells diskutiert.

2.2 Randomisierte Algorithmen

Die Analyse von deterministische Pagingalgorithmen erfolgte immer im Vergleich zu der für den Algorithmus ungünstigste Anfragefolge. Auch bei randomisierte Pagingalgorithmen gehen wir davon aus, dass ein Gegenspieler existiert, der für den Algorithmus eine möglichst ungünstige Anfragefolge konstruiert. Dabei unterscheiden wir zwischen unterschiedlichen Gegenspielertypen.

Gegenspielmodell:

Der Gegenspieler kennt stets den Online-Algorithmus inklusive der Wahrscheinlichkeitsverteilung, die der Algorithmus verwendet.

Hierbei unterscheiden wir adaptive und vergessliche Gegenspieler.

adaptive Gegenspieler:

Kennt zu jedem Zeitpunkt alles, was der Online-Algorithmus bisher getan hat und verwendet dieses Wissen bei der Auswahl der nächsten Anfrage.

vergessliche (oblivious) Gegenspieler:

Muss die gesamte Anfragefolge im Voraus ohne obige Kenntnis wählen.

adaptive Gegenspielerkosten:

a) adaptiv-offline (ADOF)

Kosten Online-Algorithmus $\longleftrightarrow$
optimale Offlinekosten (bzgl. Anfrage=
folge, die Gegenspieler online kreiert).

b) adaptiv-online (ADON)

Der Gegenspieler muss seine kreierte
Anfrage beobachten, bevor der Online-Al-
gorithmus dies tut.

vergessliche Gegenspielerkosten: (OBL)

Kosten Online-Algorithmus $\longleftrightarrow$
optimale Offlinekosten.

Sei ADV ein Gegenspieler vom Typ aus $\{OBL, ADOF, ADON\}$. Wir sagen, dass ein Online-Algorithmus ALG c-competitive gegen ADV ist, falls eine Konstante c existiert, so dass $\forall \sigma$

$$(*) \quad E[ALG(\sigma) - c \cdot ADV(\sigma)] \leq \alpha.$$

Bemerkung

Es gilt: $OBL(\sigma) = OPT(\sigma)$ und $ADOF(\sigma) = OPT(\sigma)$

Unterschied:

- $OBL(\sigma)$ ist eine feste Größe.

- $ADOFF(\sigma)$ ist eine Zufallsvariable, da die Wahl von σ von den von ALG getroffenen Zufallsentscheidungen abhängt. D.h., σ ist eine Zufallsvariable und somit $ADOFF(\sigma) = OPT(\sigma)$ auch.
- $ADON(\sigma)$ ist aus demselben Grund auch eine Zufallsvariable. Jedoch haben wir keine genaue Charakterisierung des Wertes $ADON(\sigma)$.

Also kann (*) folgendermaßen geschrieben werden:

- $ADV = OBL$:

$$E[ALG(\sigma)] - c \cdot OPT(\sigma) \leq \alpha$$

- $ADV = ADOF$:

$$E[ALG(\sigma) - c \cdot OPT(\sigma)] \leq \alpha$$

Sei $\bar{R}_{ADV}(ALG)$ das Infimum über alle c , so dass ALG c -competitive gegen Gegenspieler vom Typ ADV ist. Dann heißt $\bar{R}_{ADV}(ALG)$ das competitive Verhältnis von ALG gegen Gegenspieler vom Typ ADV .

Folgende Heuristik für die zufällige Auswahl der zu verdrängenden Seite scheint sinnvoll zu sein:

RAND (Random):

Wenn immer ein Seitenwechsel durchzuführen ist, wähle die zu verdrängende Seite zufällig, wobei jede Seite in k mit derselben Wahrscheinlichkeit verdrängt wird.

Folgender Satz zeigt, dass das erwartete Verhalten von RAND gegen einen adaptiven Online-Gegenspieler genauso gut ist, als das von deterministischen Online-Algorithmen LRU oder FIFO.

Satz 2.7

Der Algorithmus RAND ist $\frac{k}{k-h+1}$ -competitive gegen einen adaptiven Online-Gegenspieler für das (h, k) -Pageingproblem.

Beweis:

siehe Borodin, El-Yaniv S. 47-48.

Für vergessliche Gegenspieler können wir beweisen, dass es nicht besser geht. Hierfür ist folgendes Lemma nützlich:

Lemma 2.3

Sei W eine Zufallsvariable für die Wartezeit auf einen Erfolg in einer Folge von Bernoulli-Versuchen mit Erfolgswahrscheinlichkeit p . Für $j \in \mathbb{N}$ definiere wie folgt die Zufallsvariable W_j :

$$W_j := \begin{cases} W & \text{falls } W \leq j \\ j & \text{sonst} \end{cases}$$

99

Dann gilt $E[W_j] = \frac{1}{p} (1 - (1-p)^j)$.

Beweis:

Es gilt:

$$\begin{aligned} E[W_j] &= j \cdot P[W > j] + \sum_{i=1}^j i \cdot P[1. \text{ Erfolg in Runde } i] \\ &= j(1-p)^j + p \cdot \sum_{i=0}^{j-1} (i+1)(1-p)^i \end{aligned}$$

Ziel:

Entwicklung einer allgemeinen Formel für $\sum_{i=0}^{j-1} (i+1)(1-p)^i$.

Bemerkung:

- Wenn wir die Formel wüssten, dann könnten wir diese mittels Induktion beweisen.
- Wie man auf die Formel kommt, kann man in

R.L. Graham, D.E. Knuth, O. Patashnik:

Concrete Mathematics: A Foundation for Computer Science, 2nd. Ed., Addison-Wesley 1994, Kapitel 2.2 - 2.3, insbesondere S. 33.

nachlesen.

Wir erhalten

$$\begin{aligned} &= j(1-p)^j + p \left[\frac{1 - jp(1-p)^j - (1-p)^j}{p^2} \right] \\ &= \frac{1}{p} [1 - (1-p)^j]. \end{aligned}$$

Satz 2.8

Das competitive Verhältnis von RAND gegen einen vergesslichen Gegenspieler für das (n, k) -Pagingproblem ist $\geq \frac{1}{k-n+1}$.

Beweis:

Sei $\varepsilon > 0$ vorgegeben. Seien $b_1, b_2, \dots$ Seiten, die zu Beginn nicht in U_{RAND} sind.

Zunächst definieren wir eine Anfragefolge σ , für die wir dann die untere Schranke beweisen werden. Sei

$$\sigma = (b_1, a_2, a_3, \dots, a_n)^{\ell}, (b_2, a_2, a_3, \dots, a_n)^{\ell}, \dots,$$

wobei $\ell \in \mathbb{N}$ später festgelegt wird.

Die Unterfolge $(b_i, a_2, a_3, \dots, a_n)^{\ell}$ $i=1, 2, \dots$ heißt i -ter Block der Anfragefolge.

Zu Beginn des i -ten Blockes gilt:

- $b_i \notin U_{\text{RAND}}$

$\Rightarrow$ Zu Beginn des i -ten Blockes gilt:

$$|U_{\text{RAND}} \cap \{b_i, a_2, \dots, a_n\}| \leq n-1.$$

Betrachte RAND während der Bearbeitung eines Segmentes $b_i, a_2, a_3, \dots, a_n$ des i -ten Blockes

Annahme.

Unmittelbar vor dem Abarbeiten des Segmentes gilt $|K_{\text{RAND}} \cap \{b_i, a_2, a_3, \dots, a_n\}| \leq h-1$.

Betrachte folgenden Bernoulliversuch:

- Erfolg während des betrachteten Segmentes:
 $\Leftrightarrow$
Während des letzten Seitenfehlers (der wegen obiger Annahme existieren muss) gilt:
 $|K_{\text{RAND}} \cap \{b_i, a_2, a_3, \dots, a_n\}| = h-1$
und RAND legt Seite $q \notin \{b_i, a_2, a_3, \dots, a_n\}$ aus.

Konstruktion $\Rightarrow$

Solange die Bernoulliversuche erfolglos enden, enthält jedes Segment mindestens einen Seitenfehler.

Auswahlregel von RAND $\Rightarrow$

$$Pr[\text{Erfolg}] \leq \frac{k - (h-1)}{k}$$

Mittels Anwendung des Lemmas 2.3 mit $p = \frac{k-h+1}{k}$ und $j = \ell$ erhalten wir

E [Anzahl der Seitenfehler von RAND während Block i]

$$\geq \frac{k}{k-h+1} \left(1 - \left(\frac{h-1}{k} \right)^l \right)$$

$\xrightarrow{l \rightarrow \infty} 0$

$$\geq \frac{k}{k-h+1} - \varepsilon \quad \text{für } l \text{ groß genug.}$$

Übung:

Modifizieren Sie den Beweis von Satz 2.8, so dass nur $(k+1)$ Seiten des Hintergrundspeichers benötigt werden.

Somit bringt die nahe liegende Heuristik RAND für die zufällige Auswahl der zu verdrängenden Seite keine Verbesserung für das competitive Verhältnis gegenüber den besten bekannten deterministischen Online-Algorithmen für das (k, k) -Pagingproblem.

Idee

Verknüpfe RAND und Markierungsalgorithmen.

Obige Idee führt zu folgender Heuristik:

MARK (Markierungsalgorithmus):

- Zu Beginn sind alle Seiten in $\mathcal{U}_{\text{MARK}}$ unmarkiert.

• Aufgabe p

- Falls $p \in K_{\text{MARK}}$ und unmarkiert, dann markiere p.

- Falls $p \notin K_{\text{MARK}}$, dann

Seitenaustausch $p \leftrightarrow q$ und markiere p

wobei q unter allen nicht markierten Seiten in K_{MARK} zufällig gewählt wird. Falls dies nicht möglich ist, da alle Seiten markiert sind, dann entferne zunächst alle Markierungen.

Ziel:

Analyse von MARK gegen einen vergesslichen Gegenspieler.

Berechnung:

Sei $k \in \mathbb{N}$. $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k}$ heißt k-te harmonische Zahl.

Übung:

Beweisen Sie $\ln k \leq H_k \leq 1 + \ln k \quad \forall k \geq 1$

Satz 2.8

$$\bar{R}_{\text{OBL}}(\text{MARK}) \leq 2 \cdot H_k.$$

Beweis:

Annahme:

Zu Beginn gilt $K_{\text{MARK}} = K_{\text{OBL}}$.

Sei σ eine beliebige Anfragefolge.

Idee:

- Herleitung einer
 - oberen Schranke für die erwartete Anzahl der Seitenfehler von MARK.
 - unteren Schranke für die Anzahl der Seitenfehler in $\text{OPT}(\sigma)$.
- Eine obere Schranke für das kompetitive Verhältnis von Mark ergibt sich dann aus dem Quotienten der beiden Schranken.

Durchführung:

Betrachte die zu σ korrespondierende k -Phasenunterteilung.

Bezeichnungen (bzgl. Phase i):

- Seiten zu Beginn in K_{MARK} heißen alt.
- Jede in Phase i angefragte nichtalte Seite heißt neu.

Es gilt:

Jede während einer Phase angefragte Seite verläßt den Kassenpeicher während dieser

Phase nicht mehr.

Sei m_i die Anzahl der neuen Seiten während Phase i . Die schlechteste Anordnung der Seitenanfragen für den online-Algorithmus während Phase i sieht wie folgt aus:

- Frage zunächst alle m_i neue Seiten an und
- dann führe alle Anfragen bzgl. alte Seiten durch.

Wenn dies so ist, dann verursachen die ersten m_i Anfragen für MARK einen Seitenfehler, also

kosten m_i .

Ziel:

Analyse der erwarteten Anzahl von Seitenfehlern bzgl. den restlichen $k - m_i$ (ersten) Anfragen bzgl. alten Seiten.

Beobachtung:

Pr [j -te alte Seite befindet sich bei Anfrage in K_{MARK}]

$$= \frac{k - m_i - (j-1)}{k - (j-1)}$$

← # alte unmarkierte Seiten in K_{MARK}

← # alte unmarkierte Seiten insgesamt

Also gilt:

$\Pr[\text{Anfrage auf } j\text{-te alte Seite bedingt Seitenfehler}]$

$$= 1 - \frac{k - m_i - (j-1)}{k - (j-1)}$$

$$= \frac{m_i}{k - j + 1}$$

Also gilt

$E[\text{Anzahl der Seitenfehler in Phase } i]$

$$= m_i + \sum_{j=1}^{k-m_i} \frac{m_i}{k-j+1}$$

$$= m_i + m_i \sum_{j=m_i+1}^k \frac{1}{j}$$

$$= m_i + m_i (H_k - H_{m_i})$$

$$= m_i (H_k - H_{m_i} + 1)$$

$$\leq m_i H_k$$

Also gilt

$$\text{MARK}(\sigma) \leq H_k \cdot \left(\sum_i m_i \right)$$

Ziel:

Herleitung einer unteren Schranke für die Anzahl der Seitenfehler in $\text{OPT}(\sigma)$.

Definition der k -Phasenunterteilung und m_i



Während den Phasen $(i-1)$ und i werden $\geq m_i + k$ verschiedene Seiten angefragt.

⇒

∀ i > 1 gilt:

Die Anzahl der Seitenfehler, die OPT während den Phasen (i-1) und i macht, ist

$$\geq m_i.$$

Wegen $K_{OPT} = K_{MARK}$ zu Beginn der Phase 1 gilt:

Die Anzahl der Seitenfehler während der Phase 1 ist

$$\geq m_1.$$

Also gilt:

$$OPT(\sigma) \geq \frac{1}{2} \cdot \sum_i m_i.$$

Also gilt:

$$\frac{MARK(\sigma)}{OPT(\sigma)} \leq \frac{H_2(\sum_i m_i)}{\frac{1}{2}(\sum_i m_i)} = 2 \cdot H_2$$

Übung:

Exercise 4.5 in Borodin / El Yoniv

Exercise 4.6 in Borodin / El Yoniv.

Ziel:

Herleitung einer unteren Schranke für randomisierte Pagingalgorithmen.

Satz 2.10

Sei ALG ein randomisierter Pagingalgorithmus mit Kernspeichergroße k . Sei die Gesamtanzahl der Seiten $N \geq k+1$. Dann gilt

$$\bar{R}_{\text{OBL}}(\text{ALG}) \geq H_k.$$

Beweis:

Annahme: $N = k+1$

Ziel:

Konstruktion einer Anfragefolge σ durch einen vorguesslichen Gegenspieler, so dass

$$E[\text{ALG}(\sigma)] \geq H_k \cdot \text{OPT}(\sigma).$$

Gegenspieler:

- Für $1 \leq j \leq N$ speichert sich der Gegenspieler zu jedem Zeitpunkt die aktuelle Wahrscheinlichkeit p_j , dass Seite $j \notin K_{\text{ALG}}$.

Dies ist möglich, da der Gegenspieler die von ALG verwendete Wahrscheinlichkeitsverteilung kennt.

D.h.,

N -Tupel $(p_1, p_2, \dots, p_N)$ mit der Eigenschaft, dass nach jeder Anfrage

$$\sum_{j=1}^N p_j = 1$$

gilt.

103

Somit ist $\text{skts } (p_1, p_2, \dots, p_N)$ eine Wahrscheinlichkeitsverteilung.

$\Rightarrow$

Erwartete Kosten für eine Anfrage bzgl. i -te Seite für ALG betragen p_i .

Struktur von $\mathcal{T}$:

$\mathcal{T}$ besteht aus einer beliebig großen Anzahl von k -Phasen. Diese werden mit Hilfe eines Markierungsalgorithmus konstruiert. Dabei merkt sich der Gegenspieler stets die aktuell markierten Seiten.

Ziel:

Konstruktion einer k -Phasenunterteilung nebst dem Beweis, dass die erwartete Anzahl von Seitenfehlern von ALG in jeder k -Phase $\geq H_k$ ist.

Bemerkung:

OPT wird bzgl. jeder k -Phase höchstens einen Seitenfehler haben, woraus dann unmittelbar die untere Schranke folgt.

k -Phase:

Wir unterteilen die k -Phase in k Unterphasen, wobei

- i -te Unterphase startet, wenn genau $k - i + 1$ Seiten unmarkiert sind.
- Am Ende der k -ten Unterphase bleibt die zuletzt angefragte Seite markiert, während von allen anderen Seiten die Markierungen entfernt werden. Dann beginnt eine neue k -Phase.

Bemerkung:

Beachten Sie, dass wir zum Beweis der unteren Schranke einen speziellen Markierungsalgorithmus konstruieren, dessen Markierungsregeln sich von den des früheren Markierungsalgorithmus unterscheiden.

Ziel:

Konstruktion der i -ten Unterphase, so dass die erwarteten Kosten von ALG $\geq \frac{1}{k - i + 1}$ sind.

Bemerkung (*):

$$\text{Es gilt: } \sum_{i=1}^k \frac{1}{k - i + 1} = \sum_{j=1}^k \frac{1}{j} = H_k.$$

Struktur der Unterphasen:

- Folge von ≥ 0 Anfragen bzgl. markierten Seiten gefolgt von
- einer Anfrage bzgl. einer unmarkierten Seite, die dann markiert wird.

j-te Unterphase

M Menge der zu Beginn der j-ten Unterphase markierten Seiten.

D.h., $|M| = j$ und
 $r := k+1-j$ Anzahl der unmarkierten Seiten.

Sei

$$\gamma := \sum_{i \in M} p_i.$$

Wir unterscheiden zwei Fälle.

1. Fall: $\gamma = 0$

Dann existiert eine unmarkierte Seite a mit

$$p_a \geq \frac{1}{r}$$

$\leadsto$

Als nächstes erfolgt eine Anfrage bzgl. Seite a .

2. Fall: $\gamma > 0$

Dann existiert eine Seite $m \in M$ mit

$$p_m = \varepsilon > 0.$$

Der Gegenspieler ergänzt σ um folgende Anfragefolge:

- (1) Anfrage bzgl. Seite m
- (2) while erwartete Kosten von ALG für ~~j -te~~ Unterphase $< \frac{1}{r}$ und $\gamma > \varepsilon$

do

Anfrage bzgl. Seite $l \in M$ mit

$$p_l = \max_{i \in M} p_i.$$

od.

Beobachtung:

Jede Iteration der while-Schleife erhöht die erwartete Kosten um $\geq \frac{\varepsilon}{|M|}$.

$\Rightarrow$

Die Anzahl der Durchläufe des Blocks der while-Schleife ist maximal $\frac{|M|}{r \cdot \varepsilon}$

$\Rightarrow$

Verfahren terminiert.

Betrachte die Situation unmittelbar nach der Terminierung. Wir unterscheiden zwei Fälle:

2.1 erwartete Kosten $\geq \frac{1}{r}$

- Beende die Unterphase mit einer Anfrage bzgl. einer beliebigen unmarkierten Seite.

2.2 erwartete Kosten $< \frac{1}{r}$

$\Rightarrow$

Die Schleife terminiert mit $\gamma \leq \varepsilon$.

- Beende die Unterphase mit einer Anfrage bzgl. einer unmarkierten Seite b mit p_b maximal.

$\Rightarrow$

$$p_b \geq \frac{1-\alpha}{r}$$

Also gilt für die erwarteten Kosten von ALG in der j -ten Unterphase

$$\begin{aligned} &\geq p_m + p_b \geq \varepsilon + \frac{1-\alpha}{r} \\ &\geq \varepsilon + \frac{1-\varepsilon}{r} \\ &= \frac{1}{r} + \underbrace{\left(\varepsilon - \frac{\varepsilon}{r} \right)}_{\geq 0} \\ &\geq \frac{1}{r} \end{aligned}$$

Somit haben wir unser Ziel erreicht. Aus Bemerkung (*) und der Tatsache, dass OPT bzgl. jeder k -Phase höchstens einen Seitenfehler hat folgt direkt

$$E[ALG(\sigma)] \geq H_k \cdot OPT(\sigma)$$

und somit

$$\bar{R}_{OBL}(ALG) \geq H_k$$

Wiederholung:

Ein metrischer Raum M ist ein Paar (S, d) , wobei S eine Menge von Punkten und $d: S \times S \rightarrow \mathbb{R}^+$ eine Distanzfunktion sind. Dabei gilt:

- | | | |
|---------------------------------------|--|--------------------------|
| i) $d(i, j) > 0$ | $\forall i \in S, j \in S \setminus \{i\}$ | (Positivität) |
| ii) $d(i, i) = 0$ | $\forall i \in S$ | (Reflexivität) |
| iii) $d(i, j) + d(j, k) \geq d(i, k)$ | $\forall i, j, k \in S$ | (Δ -Ungleichung) |
| iv) $d(i, j) = d(j, i)$ | $\forall i, j \in S$ | (Symmetrie) |

3. Das k -Server-Problem

Seien $k > 1$ ganzzahlig, $M = (S, d)$ ein metrischer Raum mit $|S| > k$.

Problemstellung:

- k mobile Server befinden sich auf Punkten in S .
Anfragefolge $\sigma := r_1 r_2 \dots r_n$, wobei $r_i \in S, 1 \leq i \leq n$.
- Um die Anfrage r_i zu bedienen muss sich ein Server auf r_i befinden
- Ein Algorithmus arbeitet σ ab, indem er die Server bewegt um nacheinander alle Anfragen zu bedienen.

$ALG(\sigma) :=$ Gesamtdistanz, die die Server, bewegt durch ALG , zurücklegen.

Das k-Server-Problem ist die Frage nach c-competitiven Online-Algorithmen für beliebige oder spezielle metrische Räume.

Beispiel:

Paging, modelliert als k-Server-Problem:

$M := (S, d)$, wobei

S Menge aller Seiten

$d: S \times S \rightarrow \mathbb{R}^+$ ist definiert durch

$$d(i, j) := \begin{cases} 1 & \text{falls } i \neq j \\ 0 & \text{falls } i = j \end{cases}$$

k Server repräsentieren den Hauptspeicher der Größe k.

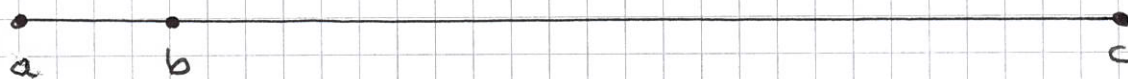
Übung:

Gegeben sei eine Anfragefolge $\sigma := \tau_1 \tau_2 \dots \tau_n$ für das k-Server-Problem. Entwickeln Sie unter Verwendung von dynamischer Programmierung einen Algorithmus, der einen optimalen Schedule für diese Anfragefolge berechnet. Analysieren Sie die Komplexität Ihres Algorithmus.

Folgende Heuristik sieht plausibel aus:

- Bewege stets einen Server, der sich am nächsten zum angefragten Punkt sich befindet.

Folgendes einfache Beispiel zeigt, dass diese Heuristik beliebig schlecht sein kann:



2 Server, zu Beginn auf a und b.

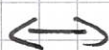
$$\sigma := c b a b a b a \dots$$

Im obigen Beispiel führt die Minimierung lokaler Kosten zu unbegrenzten Kosten, während

$$\text{OPT}(\sigma) = 2 \cdot d(b, c).$$

Ähnlich wie beim Paging können wir auch hier das (h, k) -Server-Problem definieren. D.h.,

Online
 k Server



Offline
 k Server.

Dementspforte implizieren untere Schranken für das (h, k) -Pagingproblem untere Schranken für gewisse (h, k) -Serverprobleme. Folgender Satz gibt uns eine allgemeine untere Schranke:

Satz 3.1

Sei $M = (S, d)$ ein metrischer Raum mit $|S| \geq k+1$.

Seien $1 \leq h \leq k$ und ALG ein beliebiger Online-Algorithmus für das (h, k) -Server-Problem auf M .

Dann ist das competitive Verhältnis von

$$\text{ALG} \geq \frac{k}{k-h+1}.$$

Beweis:

Idee:

Konstruktion einer beliebig langen Anfragefolge σ , so dass

$$\text{ALG}(\sigma) \geq \frac{k}{k-n+1} \text{OPT}(\sigma).$$

O.B.d.A. können wir annehmen, dass ALG stets die k Server auf k verschiedene Punkte in S platziert.

Wir wählen beliebige $k+1$ Punkte in S und nummerieren diese mit 0 beginnend durh.

Sei $[1, 2, \dots, k]$ die Startkonfiguration von ALG. D.h., die k Server befinden sich zu Beginn auf den Punkten $1, 2, \dots, k$.

Bezeichne $\sigma := \tau_1 \tau_2 \dots \tau_n$ die zu konstruierende Anfragefolge. Dann definieren wir für $1 \leq i \leq n$

$\tau_i := x \in \{0, 1, 2, \dots, k\}$ mit kein Server befindet sich auf Punkt x .

$\Rightarrow$

ALG bedient τ_i mit dem Server auf Punkt τ_{i+1} .

$\Rightarrow$

Die Kosten hierfür sind $d(\tau_{i+1}, \tau_i)$.

(113)

Sei $n \geq 2$. Dann gilt:

$$\begin{aligned} \text{ALG}(\sigma) &\geq \sum_{i=1}^{n-1} d(r_{i+1}, r_i) \\ &= \sum_{i=1}^{n-1} d(r_i, r_{i+1}). \end{aligned}$$

Ziel:

Beweis der Existenz eines Offline- k -Server-Algorithmus, der für σ Kosten

$$\leq \frac{k-1}{k} \text{ALG}(\sigma)$$

hat.

Idee:

Konstruktion einer Menge $\mathcal{B}$ von Offline-Algorithmen nebst Abschätzung der Gesamtkosten von $\mathcal{B}$ für σ . Dann existiert in $\mathcal{B}$ ein Algorithmus, dessen Kosten nicht höher, als die durchschnittlichen Kosten sind.

Durchführung:

Sei $S' := \{0, 1, 2, \dots, k\}$. Wir betrachten $R \subseteq S'$ mit $|R| = k$ und $r_1 \in R$ beliebig, aber fest. Wir definieren bzgl. R folgenden Algorithmus A_R :

Algorithmus A_R :

- Startkonfiguration ist R
- bedient nicht besetzte Position r_i mit Server auf r_{i-1} ($i = 2, 3, \dots, n$)

Gemäß Konstruktion gilt $r_1 \in R$. D.h., r_1 befindet sich in der Startkonfiguration.

$\Rightarrow$

A_R ist wohldefiniert.

Sei

$$\mathcal{B} := \{ A_R \mid R \in S', |R| = k \text{ und } r_1 \in R \}.$$

Lemma 3.1

Während der Bearbeitung von σ erreichen niemals zwei Algorithmen A_{R_1} und A_{R_2} mit $A_{R_1}, A_{R_2} \in \mathcal{B}$ und $R_1 \neq R_2$ dieselbe Konfiguration.

Bevor wir das Lemma beweisen, führen wir den Beweis von Satz 3.1 zu Ende.

Lemma 3.1 $\Rightarrow$

Alle $\binom{k}{n-1}$ Algorithmen in $\mathcal{B}$ sind stets in paarweise unterschiedlichen Konfigurationen.

Also gilt für $i > 1$:

Exakt $\binom{k-1}{n-1}$ der Algorithmen haben bei der Anfrage r_i keinen Server auf r_i und bewegen den Server von r_{i-1} nach r_i .

$\Rightarrow$

Für die Gesamtkosten $B(\sigma)$ gilt:

$$\begin{aligned} B(\sigma) &= \binom{k-1}{n-1} \sum_{i=2}^n d(r_{i-1}, r_i) \\ &= \binom{k-1}{n-1} \sum_{i=1}^{n-1} d(r_i, r_{i+1}). \end{aligned}$$

Also betragen die mittleren Kosten der Algorithmen in B :

$$\frac{\binom{k-1}{n-1}}{\binom{k}{n-1}} \cdot \sum_{i=1}^{n-1} d(r_i, r_{i+1})$$

$$\leq \frac{k - n + 1}{k} \text{ALG}(\sigma).$$

Somit verbleibt nur noch der Beweis von Lemma 3.1.

Bew. v. Lem. 3.1: (indirekt)

Betrachten wir die erste Situation, in der das Lemma verletzt wird.

Sei dies bedingt durch Abarbeiten von r_i , $i > 1$.

D.h., $\exists A_{R_1}, A_{R_2} \in B$, $R_1 \neq R_2$ mit:

- Vor Abarbeitung von r_i befinden sich A_{e_1} und A_{e_2} in unterschiedlichen Konfigurationen.
- Danach sind A_{e_1} und A_{e_2} in derselben Konfiguration.

Seien $\tilde{R}_1$ und $\tilde{R}_2$ die Konfigurationen von A_{e_1} und A_{e_2} unmittelbar vor der Bearbeitung von r_i .

Wir unterscheiden drei Fälle:

1. Fall: $r_i \in \tilde{R}_1 \cap \tilde{R}_2$

Dann ändern beide Algorithmen ihre Konfiguration nicht. ✓

2. Fall: r_i ist in genau einer der Konfigurationen $\tilde{R}_1$ und $\tilde{R}_2$ enthalten.

O.B.d.A. sei $r_i \in \tilde{R}_1$.

$\Rightarrow$

A_{e_1} ändert seine Konfiguration nicht. D.h. insbesondere, dass r_{i-1} seinen Server behält.

Aber:

A_{e_2} bewegt den Server von r_{i-1} nach r_i . D.h., auf r_{i-1} befindet sich anschließend kein Server mehr. ✓

3. Fall: $r_i \notin \tilde{R}_1$ und $r_i \notin \tilde{R}_2$.

Beide Algorithmen bewegen den Server von r_{i-1} nach r_i . ✓

3.1 Spezielle metrische Räume

Zunächst betrachten wir die reelle Linie mit der euklidischen Metrik.

Algorithmus DC

- Anfrage jenseits vom äußersten Server
 $\leadsto$
 Bewege einen nächsten äußersten Server auf den angefragten Punkt.
- Anfrage zwischen zwei benachbarten Server
 $\leadsto$
 Bewege jeweils einen der benachbarten Server mit derselben Geschwindigkeit zum angefragten Punkt bis mindestens einer diesen erreicht hat. HALT.

Beobachtung:

DC stellt nicht sicher, dass sich alle k Server stets auf unterschiedlichen Punkten befinden.

Satz 3.2

DC ist k -competitive.

Beweis:

Bezeichne $M_{\min}$ die Kosten einer perfekten Matchings minimaler Kosten zwischen den

Servern von OPT und von DC. Seien

- s_i , $1 \leq i \leq k$ Positionen der Server von DC
- $\sum_{DC} := \sum_{i < j} d(s_i, s_j)$
- Potentialfunktion $\Phi := k \cdot M_{\min} + \sum_{DC}$.

Beobachtung: $\Phi \geq 0$.

Falls die Startkonfigurationen für OPT und DC gleich sind, dann gilt zu Beginn

$$\Phi_{\text{start}} = \sum_{DC},$$

was eine Konstante ist.

Für die Durchführung der Potentialfunktionsmethode nehmen wir an, dass OPT stets vor DC seine Serverbewegung durchführt.

Da auch nach der Terminierung $\Phi \geq 0$ gilt, genügt es folgendes zu zeigen:

- Wenn OPT seine Server eine Distanz d bewegt, dann wächst der Wert der Potentialfunktion höchstens um $k \cdot d$.
- Wenn DC seine Server eine Distanz d bewegt, dann vermindert sich der Wert der Potentialfunktion mindestens um d .

i) und ii) implizieren dann

$$DC(\sigma) \leq k \cdot OPT(\sigma) + \phi_{\text{start}}.$$

i) folgt aus folgenden Beobachtungen:

a) Bewegungen von Servern durch OPT verändern $\sum_{DC}$ nicht.

b) Eine Bewegung eines Servers durch OPT um d erhöht $M_{\min}$ höchstens um d .

Bew. von ii):

Wir unterscheiden zwei Fälle:

1. Fall: DC bewegt einen einzelnen Server um die Distanz d .

Definition von DC $\Rightarrow$

Äußerster Server wird bewegt. D.h., dieser Server wird von jedem anderen Server um die Distanz d weg bewegt.

$\Rightarrow$

$\sum_{DC}$ wächst um $(k-1) \cdot d$.

Seien s der Server, den DC auf den Auftragspunkt bewegt und t der Server von OPT, der sich auf dem Auftragspunkt befindet.

(12)

Sei M ein Matching minimaler Kosten unmittelbar bevor DC den Server s bewegt. Zunächst werden wir uns davon überzeugen, dass ein Matching minimaler Kosten existiert, in dem s und t miteinander gematcht sind. Falls M bereits diese Eigenschaft besitzt, dann ist diesbezüglich nichts mehr zu zeigen.

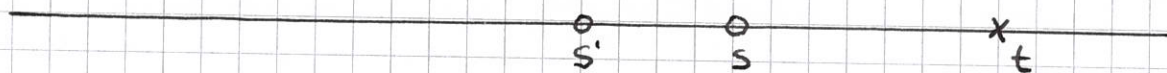
Annahme:

s und t sind in M nicht miteinander gematcht.

Seien s' derjenige Server von DC, der in M mit t und t' derjenige Server von OPT, der in M mit s gematcht ist.

Falls s und s' sich auf demselben Punkt der Linie befinden, dann können wir beide im Matching vertauschen, ohne die Kosten des Matchings zu verändern. Im resultierenden Matching minimaler Kosten sind s und t miteinander gematcht.

Falls sich s und s' auf verschiedenen Punkten befinden, dann liegt, da s äußerster Server ist, folgende Situation vor:



Egal, wo t' auf der Linie sich befindet, führt das Vertauschen von s und s' zu einem Matching, dessen Kosten nicht größer als die Kosten von M sind. (überzeugen Sie sich.)

Insgesamt haben wir gezeigt, dass ein Matching minimaler Kosten, in dem s und t miteinander gematcht sind, existiert.

$\Rightarrow$

$M_{\min}$ vermindert sich mindestens um d .

$\Rightarrow$

Der Wert der Potentialfunktion vermindert sich mindestens um

$$k \cdot d - (k-1)d = d.$$

2. Fall: DC bewegt zwei Server s_1 und s_2 von zwei gegenüberliegenden Punkten auf den Anfragepunkt zu.

Es existiert ein Matching M minimaler Kosten, in dem einer der Server, die bewegt werden, mit dem Server von OPT, der sich auf dem Anfragepunkt befindet, gematcht ist.

Übung:

Beweisen Sie obige Aussage.

$\Rightarrow$

- Die Kosten der betreffenden Zuordnung vermindert sich in M um $\frac{d}{2}$.

- Der andere Server kann sich von seinem zugeordneten Server maximal um $\frac{d}{2}$ entfernen.

$\Rightarrow$

Gesamtveränderung von $M_{\min} \leq 0$.

Änderung von Σ_{DC} :

- Die Summe der Änderungen von s_1 und s_2 zu den anderen Servern ist 0.
- $d(s_1, s_2)$ vermindert sich um $2 \cdot \frac{d}{2} = d$.

$\Rightarrow$

Σ_{DC} vermindert sich um d .

$\Rightarrow$

Der Wert der Potentialfunktion vermindert sich mindestens um d .

Übung: (Verallgemeinerung von DC auf reelle Bäume)

Algorithmus DC-Baum

- Bewege alle zum Anziehungspunkt benachbarten Server mit gleicher Geschwindigkeit in Richtung Anziehungspunkt, bis mindestens einer diesen

erreicht hat. HALT

Beachten Sie, dass zwischendurch ein Server die Eigenschaft 'benachbart' verlieren kann, da ein anderer Server sich auf den Pfad zwischen diesem und dem Anfragepunkt platziert hat. In dem Moment bleibt dieser Server stehen. Falls mehrere Server sich in derselben Position befinden und zum Anfragepunkt benachbart sind, wird nur einer von diesen ausgewählt und bewegt.

Zeigen Sie, dass DC-Baum k -competitive ist.

3.2 Balancierungsalgorithmen

Idee:

Verteile die Last auf den k Servern gleichmäßig.

Algorithmus BAL

- Verwalte für jeden Server s_i , $1 \leq i \leq k$ stets die bisher zurückgelegte Gesamtdistanz D_i .
- Anfrage r
 - Falls sich ein Server auf r befindet, dann tue nichts.
 - Andernfalls wähle Server s_i , der $D_i + d(s_i, r)$ minimiert und platziere gewählten Server

auf r.

Ziel:

Beweis, dass BAL für metrische Räume $M = (S, d)$ mit $|S| = k+1$ k -competitive ist.

Satz 3.3

Sei $M = (S, d)$ ein beliebiger metrischer Raum mit $|S| = k+1$. Dann ist BAL k -competitive bzgl. des k -Server-Problems auf M .

Beweis:

• aus

S. Irani, A.R. Karlin, Online Computation, Chapter 13 in D. Hochbaum (ed.) "Approximation Algorithms for NP-Hard Problems", PWS Publishing Company (1997).

anderer Beweis in Borodin/El Yaniv.

Zu jedem Zeitpunkt existiert genau ein Punkt in S , auf dem sich keiner von BAL's Server befindet. Diesen Punkt nennen wir Lücke.

Annahme:

Alle Anfragen beziehen sich stets auf die Lücke von BAL.

Eine Anfrage bzgl. eines anderen Punktes kostet BAL nichts und kann lediglich die Kosten von OPT erhöhen. Also können wir obiges o.B.d.A. annehmen.

Sei

$$S := \{1, 2, \dots, k+1\}.$$

Bezeichne

- k_i diejenige Konfiguration, in der jeder Punkt bis auf den Punkt i einen Server besitzt.
- $\text{opt}_t^+(k_i)$ optimale Kosten, die ersten t Anfragen zu bedienen und dabei in Konfiguration k_i zu enden.

Beobachtung:

Für alle $i, j \in S$ betragen die Kosten um von i nach j zu gelangen $d(i, j)$

$\Rightarrow$

$$\text{opt}_t^+(k_i) \leq \text{opt}_t^+(k_j) + d(i, j) \quad \forall i, j \in S.$$

Bezeichne

- σ^t die Folge der ersten t Anfragen.
- h^t die Lücke von BAL nach σ^t .

$$D_i^t := \begin{cases} \text{Distanz, die BAL's Server} & \text{falls } i \neq h^t \\ \text{auf } i \text{ nach } \sigma^t \text{ bisher ins-} & \\ \text{gesamt zur\u00fcckgelegt hat} & \\ \text{undefiniert} & \text{falls } i = h^t \end{cases}$$

Annahme $\Rightarrow$

In der $(t+1)$ -ten Anfrage wird der Punkt h^t angefragt.

Beh.: F\u00fcr $i \neq h^t$ gilt $D_i^t \leq \text{opt}_t(k_i)$

Bevor wir die Behauptung beweisen, f\u00fchren wir den Beweis des Satzes zu Ende.

Es gilt:

$$\begin{aligned} \text{BAL}(\sigma^t) &= \sum_{i \neq h^t} D_i^t \\ &\stackrel{\text{Beh.}}{\leq} \sum_{i \neq h^t} \text{opt}_t(k_i) \\ &\leq k \cdot \left[\min_{1 \leq j \leq k+1} \{ \text{opt}_t(k_j) \} + \max_{i \neq j} \{ d(i, j) \} \right] \\ &\leq k \cdot \text{OPT}(\sigma^t) + b, \end{aligned}$$

wobei $b := k \cdot \max_{i \neq j} \{ d(i, j) \}$ eine Konstante ist.

Da dies f\u00fcr jedes t gilt, beweist dies den Satz. Somit verbleibt nur noch der Beweis der Behauptung.

Bew. d. Beh.: (vollständige Induktion)

Da zu Beginn keiner der Server irgendeine Distanz zurückgelegt hat, ist die Behauptung für $t=0$ trivialerweise erfüllt.

Annahme: Beh. ist erfüllt für alle $l \leq t$.

$t \leadsto t+1$:

Es gilt: $r_{t+1} = h^t$.

Sei $\min$ derjenige Punkt, der

$$\min_{i \neq h^t} \{ D_i^t + d(i, h^t) \}$$

minimiert und von BAL gewählt wird.

Dann schickt BAL den Server von $\min$ nach h^t .

$\Rightarrow$

- $h^{t+1} = \min$ und
- $D_{h^t}^{t+1} = D_{\min}^t + d(\min, h^t)$.

Ferner gilt:

$$\text{opt}_{t+1}(k_{h^t}) = \min_{i \neq h^t} \{ \text{opt}_t(k_i) + d(h^t, i) \}$$

Beachten Sie, dass zur Bedienung von r_{t+1} sich ein Server auf h^t befinden muss. Dieser muss dann noch weggeschickt werden, damit die Konfiguration k_{h^t} erreicht wird.

Wegen

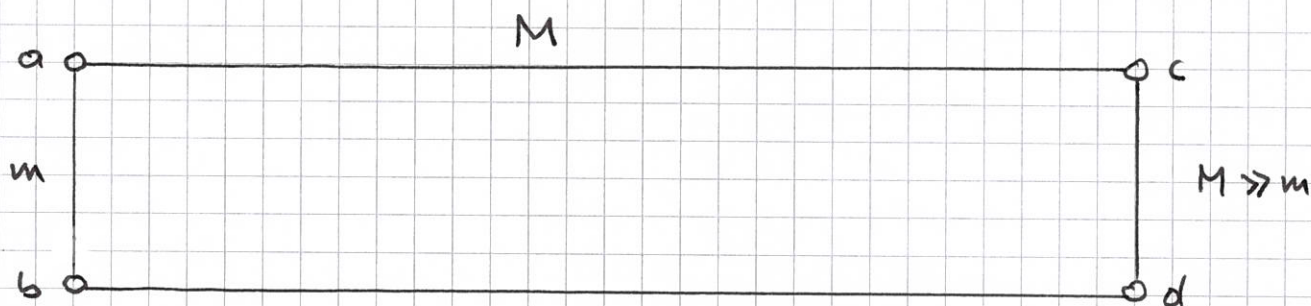
$$\text{opt}_t(\ell_i) \geq D_i^t \quad \text{für } i \neq \ell_t$$

gilt gemäß der Auswahlregel von BAL

$$\text{opt}_{t+1}(\ell_{t+1}) \geq D_{\ell_t}^{t+1}$$

Wegen $D_i^{t+1} = D_i^t$ und $\text{opt}_{t+1}(\ell_i) \geq \text{opt}_t(\ell_i)$ für $i \neq \ell_t$ und $i \neq \min$ folgt die Behauptung für diese i direkt aus der Induktionsannahme. ■

Falls $|S| > \ell+1$, dann ist BAL nicht mehr competitive. Betrachten wir hierzu folgendes Beispiel:



$|S| = 4$, $\ell = 2$, Startkonfiguration $X_0 := \{c, d\}$
 Anfragefolge $\sigma = a b c d a b c d \dots$

BAL bewegt die Server stets entlang den horizontalen Linien. Also sind die Kosten für BAL für jede Anfrage M . OPT schickt zur Bearbeitung der ersten Anfrage den auf c befindlichen Server nach a . Diese Anfrage kostet somit M . Jede weitere Anfrage wird

mittels Bewegung entlang einer vertikalen Linie bedient. Somit zahlt OPT einmal M und ansonsten m .

Bemerkung:

Obiges Beispiel würde durch unsere erste Online-Heuristik, die stets die lokalen Kosten optimiert, optimal gelöst werden.

Bisher kennen wir keinen kompetitiven Online-Algorithmus für allgemeine metrische Räume. Unser Ziel ist es, solchen zu finden.

3.3 Der Arbeitsfunktionalalgorithmus

Zunächst werden wir einige Bezeichnungen einführen und einige grundlegende Eigenschaften beweisen. Seien

- $M := (S, d)$ ein metrischer Raum,
- $S^k := \{X \subseteq S \mid |X| = k\}$ die Menge der möglichen Konfigurationen,
- $D(X, Y)$, $X, Y \in S^k$ die minimalen Kosten, um von X nach Y zu gelangen. D.h., $D(X, Y)$ sind die Kosten eines minimalen Matchings zwischen den Punkten in X und den Punkten in Y .
- $\sigma := \tau_1 \tau_2 \dots \tau_i \dots \tau_n$ Anfragefolge. D.h., $\tau_i \in S$ für $1 \leq i \leq n$.

- $\sigma^t := \tau_1 \tau_2 \dots \tau_t$ die Folge der ersten t Anfragen von σ .
- $X_0 \in S^k$ Startkonfiguration des Online- und des optimalen Offline-Algorithmus.
- $\text{opt}_t(\sigma, X)$, $X \in S^k$ minimale Kosten die Anfragefolge σ^t mit Startkonfiguration X_0 und Endkonfiguration X zu bedienen.

Bemerkung:

- $\text{opt}_t(\sigma, X)$ kann mit Hilfe dynamischer Programmierung wie folgt berechnet werden:

$$- \text{opt}_0(\sigma, X) := D(X_0, X)$$

$$- \text{opt}_t(\sigma, X) := \begin{cases} \text{opt}_{t-1}(\sigma, X) & \text{falls } \tau_t \in X \\ \min_{Y: \tau_t \in Y} \{ \text{opt}_{t-1}(\sigma, Y) + D(Y, X) \} & \text{falls } \tau_t \notin X \end{cases}$$

Wegen $\text{opt}_{t-1}(\sigma, X) = \text{opt}_{t-1}(\sigma, X) + D(X, X)$ können beide Fälle wie folgt zusammengefasst werden:

$$\text{opt}_t(\sigma, X) := \min_{Y: \tau_t \in Y} \{ \text{opt}_{t-1}(\sigma, Y) + D(Y, X) \}.$$

- Die Funktion $\text{opt}_t(\sigma, \cdot)$ wird üblicherweise Arbeitsfunktion genannt und häufig mit w_t bezeichnet.

Als nächstes werden wir einige Eigenschaften der Arbeitsfunktion herausarbeiten.

Lemma 3.2

Für alle $0 \leq t \leq n$, $X, Y \in S^k$ gilt:

$$a) \text{opt}_t(\sigma, X) \leq \text{opt}_t(\sigma, Y) + D(Y, X).$$

$$b) \text{opt}_t(\sigma, X) = \min_{x \in X} \{ \text{opt}_{t-1}(\sigma, X - x + r_t) + d(r_t, x) \}.$$

$$c) \text{opt}_t(\sigma, X) \geq \text{opt}_{t-1}(\sigma, X).$$

$$d) r_t \in X \Rightarrow \text{opt}_t(\sigma, X) = \text{opt}_{t-1}(\sigma, X).$$

$$e) \text{opt}_t(\sigma, X) = \min_{x \in X} \{ \text{opt}_t(\sigma, X - x + r_t) + d(r_t, x) \}.$$

Beweis:

a) Es gilt

$$\text{opt}_t(\sigma, Y) + D(Y, X)$$

$$= \min_{Z: r_t \in Z} \{ \text{opt}_{t-1}(\sigma, Z) + D(Z, Y) + D(Y, X) \}$$

$$\geq \min_{Z: r_t \in Z} \{ \text{opt}_{t-1}(\sigma, Z) + D(Z, X) \}$$

$$= \text{opt}_t(\sigma, X).$$

b) Es gilt:

$$(+)\text{opt}_t(\sigma, X) = \min_{Y: r_t \in Y} \{ \text{opt}_{t-1}(\sigma, Y) + D(Y, X) \}.$$

"≤"

Setze in (+) $Y := X - x + r_t$. Dann gilt
 $D(Y, X) = d(r_t, x)$.

"≥"

Sei Y_0 eine Konfiguration, die die rechte Seite von (+) minimiert.

Betrachten wir ein Matching M minimaler Kosten zwischen X und Y_0 . Sei $x \in X$ derjenige Punkt, der von M dem Punkt $r_t \in Y_0$ zugeordnet wird. D.h., $(x, r_t) \in M$.

Wir definieren dann

$$Y' := X - x + r_t.$$

Dann gilt da $M = (S, d)$ metrischer Raum und wegen a)

$$\begin{aligned} & \text{opt}_{t-1}^+(\sigma, Y_0) + D(Y_0, X) \\ &= \text{opt}_{t-1}^+(\sigma, Y_0) + D(Y_0, Y') + d(x, r_t) \\ &\geq \text{opt}_{t-1}^+(\sigma, Y') + d(x, r_t) \\ &\geq \min_{x \in X} \{ \text{opt}_{t-1}^+(\sigma, X - x + r_t) + d(x, r_t) \}. \end{aligned}$$

c) Es gilt:

$$\begin{aligned} \text{opt}_t^+(\sigma, X) &= \text{opt}_{t-1}^+(\sigma, Y) + D(Y, X) \quad \text{für ein } Y \in S^k \\ &\geq \text{opt}_{t-1}^+(\sigma, X). \end{aligned}$$

a)

d)

Teil b) mit $x = \tau_t$ impliziert

$$\text{opt}_t(\sigma, x) \leq \text{opt}_{t-1}(\sigma, x)$$

Wegen Teil c) gilt auch

$$\text{opt}_t(\sigma, x) \geq \text{opt}_{t-1}(\sigma, x)$$

Also gilt

$$\text{opt}_t(\sigma, x) = \text{opt}_{t-1}(\sigma, x).$$

e) Die Behauptung folgt direkt aus b) und d).



Idee des Arbeitsfunktionalalgorithmus:

Annahme:

Nach der Bearbeitung von σ^{t-1} befindet sich der Online-Algorithmus in der Konfiguration $X_{t-1} \in S^k$.

Um die nächste Anfrage τ_t zu bearbeiten, muss der Online-Algorithmus in eine Konfiguration $X \in S^k$ mit $\tau_t \in X$ übergehen.

- Falls $\tau_t \in X_{t-1}$, dann ist der Online-Algorithmus bereits in solcher Konfiguration und diese Anfrage kostet nichts.
- Falls $\tau_t \notin X_{t-1}$, dann stellt sich folgende Frage:

Wie wählt der Online-Algorithmus $X \in S^k$ mit $r_t \in X$ aus?

Eine naheliegende Idee ist es, in diejenige Konfiguration X_{opt} überzugehen, in die OPT bei der Bearbeitung von σ^t gelangen würde. Der Übergang von X_{t-1} nach X_{opt} kann sehr teuer sein. Nach Hinzukommen von wenigen weiteren Aufträgen könnte OPT die Konfiguration X_{opt} wieder verwerfen, während die Kosten dieses teuren Überganges die Kosten des Online-Algorithmus in die Höhe treiben würde. Demzufolge sind bei der Wahl von $X \in S^k$ mit $r_t \in X$ auch die Übergangskosten von X_{t-1} nach X zu berücksichtigen. Diese Überlegungen führen zu folgendem Algorithmus:

Arbeitsfunktionenalgorithmus (WFA):

- Gehe in eine Konfiguration $X \in S^k$ mit
 - a) $r_t \in X$ und
 - b) X minimiert

$$(*) \quad \text{opt}_t(\sigma, X) + D(X_{t-1}, X)$$

über.

25.11.

Ziel:

Gegenüberstellung der Algorithmen WFA und OPT.

Beide Algorithmen haben dieselbe Startkonfiguration x_0 und erhalten die Anfragefolge σ online.

Annahme:

OPT und WFA haben gerade die Anfrage τ_t erhalten.

Verhalten der Algorithmen:

- WFA darf seine Lösung für σ^{t-1} nicht modifizieren und ergänzt diese gemäß obiger Regel zu einer Lösung für σ^t .
- OPT erstellt für σ^t eine optimale Lösung. Dabei kann seine bisherige Lösung für σ^{t-1} modifiziert werden.

Annahme

OPT berechnet stets eine optimale Lösung für σ^t , die den längstmöglichen Präfix für σ^{t-1} enthält.

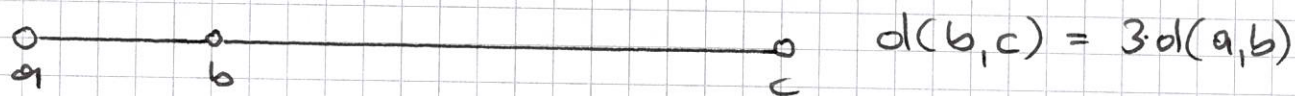
Wir bezeichnen mit OPT_{on} die oben beschriebene Online-Version von OPT.

Frage:

Wann wirkt sich eine Änderung der bisherigen Lösung für σ^{t-1} durch OPT_{on} auf das Verhalten von WFA aus?

Zur Beantwortung dieser Frage betrachten wir zu=nächst ein Beispiel.

Beispiel:



2 Server, Startkonfiguration $X_0 = \{a, b\}$

Wir betrachten zunächst die Anfragefolge

$$\sigma := c b a b a b a \dots$$

Wir diskutieren das Verhalten von OPT_{on} und von WFA für wachsende Präfixe der Anfragefolge σ . Dabei symbolisiert $x \rightarrow y$ die Serverbewegung vom Punkt x zum Punkt y .

$$\underline{\sigma^1 = c} :$$

OPT_{on} :

$$b \rightarrow c \rightsquigarrow \text{Konfiguration } \{a, c\}$$

WFA:

$$b \rightarrow c \rightsquigarrow \{a, c\}$$

$$\underline{\sigma^2 = cb} :$$

OPT_{on} :

$$a \rightarrow b \rightsquigarrow \{b, c\}$$

WFA:

$$a \rightarrow b \rightsquigarrow \{b, c\}$$

$$\underline{\sigma^3 = cba}$$

OPT_{ori}

$$b \rightarrow a \rightsquigarrow \{a, c\}$$

WFA:

$$b \rightarrow a \rightsquigarrow \{a, c\}$$

$$\underline{\sigma^4 = cba b}$$

OPT_{on}

$$a \rightarrow b \rightsquigarrow \{b, c\}$$

Bemerkung:

Zu einer Lösung mit denselben Kosten würde folgende Modifikation der bisherigen Lösung $b \rightarrow c, a \rightarrow b, b \rightarrow a$ führen:

$$b \rightarrow c, c \rightarrow b \rightsquigarrow \{a, b\}.$$

Alle nachfolgenden Anfragen $\tau_3 \tau_4 \dots$ bedient OPT_{on} ohne Serverbewegung. Gemäß unserer Präfixannahme wählt OPT_{on} jedoch obige Lösung.

WFA:

$$a \rightarrow b \rightsquigarrow \{b, c\}$$

$$\underline{\sigma^5 = cba b a}$$

OPT_{on}

Würde OPT_{on} seine bisherige Lösung

$b \rightarrow c, a \rightarrow b, b \rightarrow a, a \rightarrow b$ mit $b \rightarrow a$

fortsetzen, dann hätte die konstruierte Lösung die Kosten $7 \cdot d(a,b)$. Demgegenüber hat die in obiger Bemerkung beschriebene Lösung die Kosten $6 \cdot d(a,b)$.

$\Rightarrow$

OPT_{on} modifiziert seine Lösung wie in obiger Bemerkung beschrieben.

WFA:

Um das Verhalten von WFA zu bestimmen, benötigen wir für beide möglichen Konfigurationen $\{a,b\}$ und $\{a,c\}$ für X_t die optimalen Kosten bei festgelegter Zielkonfiguration zuzüglich den Übergangskosten $D(X_{t-1}, X_t)$.

$$\cdot \text{opt}_5(\Sigma, \{a,b\}) + D(\{b,c\}, \{a,b\})$$

$$= 6 \cdot d(a,b) + d(c,a)$$

$$= 10 \cdot d(a,b)$$

$$\cdot \text{opt}_5(\Sigma, \{a,c\}) + D(\{b,c\}, \{a,c\})$$

$$= 7 \cdot d(a,b) + d(b,a)$$

$$= 8 \cdot d(a,b)$$

$\Rightarrow$

$$b \rightarrow a \leadsto \{a,c\}.$$

$$\sigma^6 = c b a b a b$$

Da OPT_n ab jetzt seine Server nicht mehr bewegt, diskutieren wir ab jetzt nur noch WFA.

WFA:

$$\cdot \text{opt}_6(\sigma, \{a, b\}) + D(\{a, c\}, \{a, b\})$$

$$= 6 \cdot d(a, b) + d(c, b)$$

$$= 9 \cdot d(a, b)$$

$$\cdot \text{opt}_6(\sigma, \{b, c\}) + D(\{a, c\}, \{b, c\})$$

$$= 8 \cdot d(a, b) + d(a, b)$$

$$= 9 \cdot d(a, b)$$

Heuristik:

Im Fall von Kostengleichheit wählt WFA unter den in Frage kommenden Lösungen eine mit minimalen Übergangskosten aus.

$\Rightarrow$

$$a \rightarrow b \rightsquigarrow \{b, c\}$$

$$\sigma^7 = c b a b a b a$$

$$\cdot \text{opt}_7(\sigma, \{a, b\}) + D(\{b, c\}, \{a, b\})$$

$$= 6 d(a, b) + d(c, a)$$

$$= 10 \cdot d(a, b)$$

$$\begin{aligned}
 & \cdot \text{opt}_7(\sigma, \{a, c\}) + D(\{b, c\}, \{a, c\}) \\
 &= 9 \cdot d(a, b) + d(b, a) \\
 &= 10 \cdot d(a, b)
 \end{aligned}$$

$\Rightarrow$

$$b \rightarrow a \leadsto \{a, c\}$$

$$\underline{\sigma^8 = c b a b a b a b :}$$

WFA:

$$\begin{aligned}
 & \cdot \text{opt}_8(\sigma, \{a, b\}) + D(\{a, c\}, \{a, b\}) \\
 &= 6 \cdot d(a, b) + d(c, b) \\
 &= 9 \cdot d(a, b)
 \end{aligned}$$

$$\begin{aligned}
 & \cdot \text{opt}_8(\sigma, \{b, c\}) + D(\{a, c\}, \{b, c\}) \\
 &= 10 \cdot d(a, b) + d(a, b) \\
 &= 11 \cdot d(a, b)
 \end{aligned}$$

$\Rightarrow$

$$c \rightarrow b \leadsto \{a, b\}$$

Alle nachfolgenden Anfragen $\tau_9, \tau_{10} \dots$ bedient WFA ohne Serverbewegung.

Als nächstes modifizieren wir die Anfragefolge σ derart, dass OPT_{on} seine beim Präfix σ^5 vorgenommene Modifikation zurücknimmt und

analysieren den daraus resultierenden Effekt auf WFA.

Betrachten wir hierin die Anfragefolge $\bar{\sigma}$, die wir aus σ durch Ersetzung von $\tau_7 = a$ durch $\bar{\tau}_7 = c$ erhalten.

Beobachtung:

Auf dem Präfix $\bar{\sigma}^6$ verhalten sich OPT_{on} und WFA genauso, wie auf σ^6 , da beide Anfragefolgen identisch sind. D.h., die konstruierten Lösungen sind:

OPT_{on} :

$$b \rightarrow c, c \rightarrow b \sim \{a, b\}$$

WFA:

$$b \rightarrow c, a \rightarrow b, b \rightarrow a, a \rightarrow b, b \rightarrow a, a \rightarrow b \\ \sim \{b, c\}$$

Nach Erhalt von $\bar{\tau}_7 = c$ verhalten sich OPT und WFA wie folgt:

OPT_{on} :

Das Zurückschicken des Servers von b nach c ergäbe die Lösung $b \rightarrow c, c \rightarrow b, b \rightarrow c$, welche die Kosten 9 hat. Modifiziert OPT_{on} stattdessen seine Lösung für $\bar{\sigma}^6$ und verhält sich wie WFA, dann hat die konstruierte

Lösung Kosten 8.

$\Rightarrow$

$b \rightarrow c, a \rightarrow b, b \rightarrow a, a \rightarrow b, b \rightarrow a, a \rightarrow b$

$\leadsto$

$\{b, c\}$

WFA:

WFA hat alles richtig gemacht und $\bar{\sigma}^7$ genauso wie OPT_{on} gelöst.

Bemerkung:

Wenn die Anfrage bzgl. Punkt c früher erfolgt, dann ist der Effekt derselbe.

Wenn die Anfrage bzgl. Punkt c spät genug erfolgt, dann modifiziert OPT_{on} seine bisherige Lösung nicht. Auch dies möchten wir anhand des Beispiels illustrieren.

Betrachte hierzu die Anfragefolge $\bar{\sigma}$, die wir aus σ durch Ersetzung von $\tau_3 = a$ durch $\bar{\tau}_3 = c$ erhalten.

Auch hier verhalten sich OPT_{on} und WFA auf dem Präfix $\bar{\sigma}^8$ genauso wie auf σ^8 . Die konstruierten Lösungen sind:

OPT_{on} :

$b \rightarrow c, c \rightarrow b \leadsto \{a, b\}$

WFA:

$b \rightarrow c, a \rightarrow b, b \rightarrow a, a \rightarrow b, b \rightarrow a, a \rightarrow b, b \rightarrow a,$
 $c \rightarrow b$

$\leadsto \{a, b\}$

Kosten: $12 \cdot d(a, b)$

Nach Erhalt der Anfrage $\bar{r}_g = c$ verhalten sich OPT_{on} und WFA wie folgt:

 OPT_{on}

OPT_{on} schickt den Server auf b und c und erhält somit die Lösung

$b \rightarrow c, c \rightarrow b, b \rightarrow c \leadsto \{a, c\}$

Die Kosten dieser Lösung betragen 9.

WFA:

$$\cdot \text{opt}_g(\sigma, \{a, c\}) + D(\{a, b\}, \{a, c\})$$

$$= 9 d(a, b) + d(b, c)$$

$$= 12 d(a, b)$$

$$\cdot \text{opt}_g(\sigma, \{b, c\}) + D(\{a, b\}, \{b, c\})$$

$$= 10 d(a, b) + d(a, c)$$

$$= 14 d(a, b)$$

$\Rightarrow$

$b \rightarrow c \leadsto \{a, c\}$

Kosten: $15 d(a, b)$

Obiges Beispiel beschreibt Situationen, in denen sich OPT und WFA unterschiedlich verhalten. Wir werden diese nachfolgend kurz charakterisieren.

- 1) Der Server auf c war derart lange auf dem Punkt c , ohne dass dieses angeprezt worden ist, dass es, trotz größeren Bewegungskosten, rückblickend günstiger ist, diesen zu den angefragten Punkten zu bewegen um aufgrund des weiteren Servers dort Serverbewegungen einzusparen. Da die Bewegungskosten von c nach $\{a, b\}$ relativ groß sind, kann WFA der Modifikation von OPT_{on} nicht unmittelbar folgen. Erst dann, wenn die durch die Modifikation bedingte Ersparnis für OPT_{on} so groß wie die Bewegungskosten ist, bewegt WFA den Server von c nach $\{a, b\}$.
- 2) Falls der Punkt c , nachdem OPT_{on} den Server von c nach $\{a, b\}$ bewegt hat, derart früh angeprezt wird, dass es günstiger gewesen wäre, wenn OPT_{on} den Server auf c belassen hätte, dann macht OPT_{on} seine Modifikation wieder rückgängig. Da die durch diese Modifikation bedingte Ersparnis für OPT_{on} noch nicht so groß wie die Bewegungskosten war, ist WFA noch nicht dieser Modifikation gefolgt.

- 3) Die Bewegung des Servers von c nach b war für OPT_{on} bereits soviel nützlich, dass sich dieser Transfer auch bei einer Anfrage bzgl. c gelohnt hat und auch WFA dieser Modifikation gefolgt ist.

Zum Verständnis der Arbeitsweise von WFA in Relation zur Arbeitsweise von OPT_{on} ist es sinnvoll, weitere etwas größere Beispiele zu analysieren. Um WFA und somit auch die Analyse der Beispiele zu vereinfachen, liegt es nahe, zunächst folgende Frage zu beantworten:

Muss WFA zur Bestimmung einer Konfiguration $\bar{X} \in S^k$ mit $r_t \in \bar{X}$ und $\bar{X}$ minimiert bzgl. allen Konfigurationen $X \in S^k$ mit $r_t \in X$

$$(*) \quad \text{opt}_t(\sigma, X) + \mathbb{D}(X_{t-1}, X)$$

alle mögliche Konfigurationen betrachten?

Folgendes Lemma beantwortet diese Frage:

Lemma 3.3

Es existiert eine Konfiguration $\bar{X} \in S^k$, $r_t \in \bar{X}$, die $(*)$ minimiert und sich von der aktuellen Konfiguration X_{t-1} höchstens in der Position eines Servers unterscheidet.

Beweis:

Sei Y eine beliebige Konfiguration mit $r_t \in Y$, die $(*)$ minimiert. Sei M ein perfektes Matching minimaler Kosten zwischen X_{t-1} und Y . Sei $x \in X_{t-1}$ derjenige Punkt, der von M dem Punkt $r_t \in Y$ zugeordnet wird. D.h., $(x, r_t) \in M$. Wir definieren dann:

$$\bar{X} := X_{t-1} - x + r_t.$$

Dann gilt:

$$\text{opt}_t(\sigma, Y) + D(X_{t-1}, Y)$$

$$= \text{opt}_t(\sigma, Y) + D(\bar{X}, Y) + d(x, r_t)$$

$$\stackrel{\text{L. 3.2. a)}}{\geq} \text{opt}_t(\sigma, \bar{X}) + d(x, r_t)$$

$$\stackrel{\text{Def. v. } \bar{X}}{=} \text{opt}_t(\sigma, \bar{X}) + D(X_{t-1}, \bar{X}),$$

womit das Lemma bewiesen ist. ■

Der Beweis des obigen Lemmas kann wie folgt interpretiert werden:

Mit Kosten $D(Y, \bar{X}) = D(\bar{X}, Y)$ kann man von der Konfiguration Y in die Konfiguration $\bar{X}$ gelangen. Wegen $\text{opt}_t(\sigma, Y) + D(\bar{X}, Y) \geq \text{opt}_t(\sigma, \bar{X})$ bildet die Summe aus

- optimale Kosten, um σ^t zu bedienen und in der Konfiguration Y zu enden und

- die Kosten, um von der Konfiguration X_{t-1} in die Konfiguration Y zu gelangen

eine obere Schranke für die Summe aus

- optimale Kosten, um σ^t zu bedienen und in der Konfiguration $\bar{X}$ zu enden und
- die Kosten, um von der Konfiguration X_{t-1} in die Konfiguration $\bar{X}$ zu gelangen.

Also kann WFA die Anfrage r_t durch Bewegen eines Servers von x nach r_t , wobei x folgenden Ausdruck (+) minimiert, bedienen:

$$\begin{aligned}
 (+) \min_{x \in X_{t-1}} \{ \text{opt}_t^+(\sigma, X_{t-1} - x + r_t) + d(r_t, x) \} \\
 =_{\text{L3.2.e)}} \text{opt}_t^+(\sigma, X_{t-1})
 \end{aligned}$$

Danach gilt: $X_t = X_{t-1} - x + r_t$.

Um WFA in Relation zu OPT_{on} besser zu verstehen, werden wir nun ein weiteres Beispiel betrachten.

Beispiel:

Seien

$$M := (\{1, 2, \dots, k+1\}, d),$$

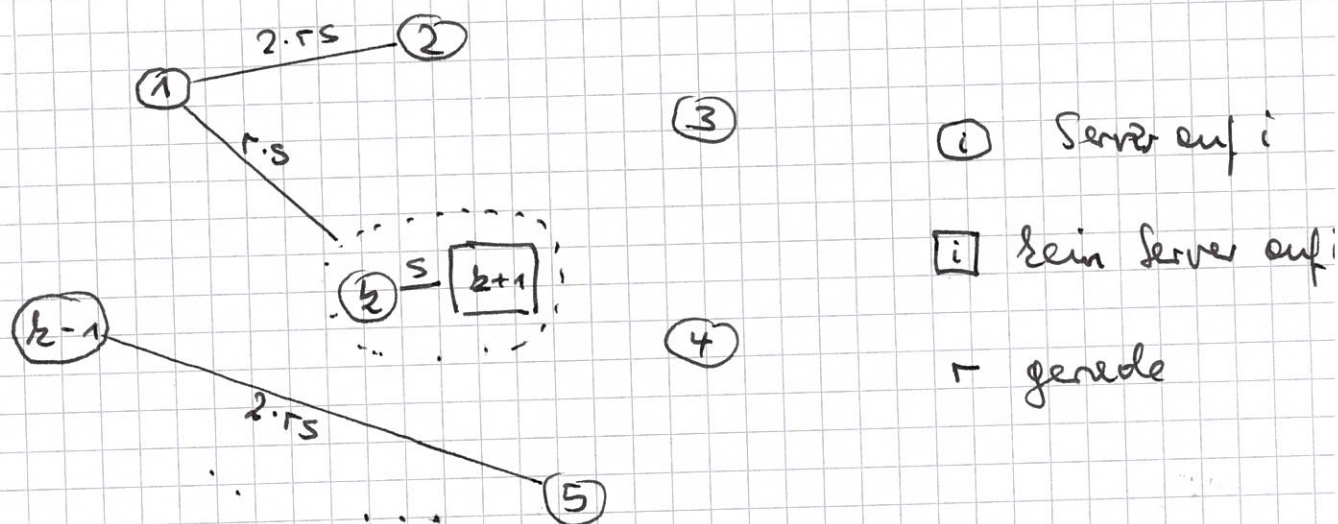
wobei

$$d(i, j) := \begin{cases} s & \text{falls } i = k \text{ und } j = k+1 \\ r.s & \text{falls } i \in \{1, 2, \dots, k-1\}, j \in \{k, k+1\} \\ 2.r.s & \text{falls } i, j \in \{1, 2, \dots, k-1\}, i \neq j \end{cases}$$

Startkonfiguration von WFA und OPT_{on} :

$$X_0 := \{1, 2, \dots, k\}.$$

Folgendes Bild beschreibt die Startkonfiguration.



Wir betrachten nun folgende Anfragefolge σ :

$$\sigma := \underbrace{(k+1)k(k+1)k \dots (k+1)k}_{(r+1)\text{-mal}} 1 2 3 \dots k-2 \dots$$

Verhalten von OPT_{on} :

Die ersten r Anfragen bedient OPT_{on} mittels Bewegen des einen Servers im Cluster $k, k+1$. Bei der $(r+1)$ -te Anfrage stellt OPT_{on} fest, dass es zur Bedienung der $r+1$ Anfragen günstiger ist, bereits bei der ersten Anfrage einen der

Serren, die sich auf den Punkten $1, 2, \dots, k-1$ befinden, nach $k+1$ zu bewegen.

Annahme:

Stets wenn OPT_{on} die Auswahl unter mehreren Serren hat, bewegt OPT_{on} von diesen Serren denjenigen, der sich auf dem Punkt mit der kleinsten Nummer befindet.

$\leadsto$

$1 \rightarrow k+1 \quad \leadsto \quad \{2, 3, \dots, k+1\}$

Die nächsten Anfragen bzgl. k und $k+1$ kann OPT_{on} ohne Serverbewegung bedienen.

Bei der Anfrage bzgl. dem Punkt 1 stellt OPT_{on} fest, dass es günstiger gewesen wäre $2 \rightarrow k+1$ anstatt $1 \rightarrow k+1$ durchzuführen und modifiziert seine Serverbewegung entsprechend. Bei der nächsten Anfrage stellt OPT_{on} fest, dass es günstiger gewesen wäre, $3 \rightarrow k+1$ anstatt $2 \rightarrow k+1$ durchzuführen und modifiziert seine Serverbewegung entsprechend. u.s.w

$\leadsto$

OPT_{on} bedient σ^{2r+k} , indem die erste Anfrage bzgl. $k+1$ mittels der Serverbewegung $k-1 \rightarrow k+1$ bedient wird. Danach sind keine Serverbewegungen mehr erforderlich.

$\leadsto$

$$k-1 \rightarrow k+1 \rightsquigarrow \{1, 2, \dots, k-2, k, k+1\}$$

Kosten: $r \cdot s$

Verhalten von WFA:

Die ersten $2r$ Anfragen bedient WFA mittels Bewegung des einen Servers im Cluster $k, k+1$. Bei der $(2r+1)$ -ten Anfrage gilt:

$$\text{opt}_t(\sigma, \{1, 2, \dots, k-1, k+1\}) + d(k+1, k) = (2r+1)s$$

Wegen

$$\begin{aligned} \text{opt}_t(\sigma, (\{1, 2, \dots, k-1\} \setminus \{i\}) \cup \{k, k+1\}) + d(k+1, i) \\ = 2rs \quad \forall i \in \{1, 2, \dots, k-1\} \end{aligned}$$

wird WFA einen der Server in $\{1, 2, \dots, k-1\}$ nach $k+1$ bewegen.

Annahme:

Stets wenn WFA die Auswahl unter mehreren Servern hat, bewegt WFA denjenigen Server, den auch OPT_n bewegt hat.

$\rightsquigarrow$

$$1 \rightarrow k+1 \rightsquigarrow \{2, 3, \dots, k+1\}$$

Die nächste Anfrage nach k bedient WFA ohne Serverbewegung. Betrachten wir die anschließende Anfrage bzgl. dem Punkt 1. Es gilt dann

$$\text{opt}_t(\sigma, \{1, 2, \dots, k-1, k'\}) + d(1, k'') = (3r+1)s$$

$$\text{für } (k' = k \text{ und } k'' = k+1) \text{ oder } (k' = k+1 \text{ und } k'' = k)$$

und

$$\begin{aligned} \text{opt}_t(\sigma, (\{1, 2, \dots, k-1\} \setminus \{i\}) \cup \{k, k+1\}) + d(1, i) \\ = 3rs \quad \forall i \in \{2, 3, \dots, k-1\} \end{aligned}$$

Also bewegt wegen obiger Annahme WFA den Server von 2 nach 1.

$\leadsto$

$$2 \rightarrow 1 \quad \leadsto \quad \{1, 3, 4, \dots, k-1, k, k+1\}$$

Genauso überlegt man sich, dass zur Bedienung der Anfrage 2 den Server 3 nach 2 bewegt.

u.s.w.

$\leadsto$

WFA bedient σ^{2r+k} , indem die ersten $2r$ Anfragen mittels Bewegen des einen Servers im Cluster $k, k+1$ bedient werden.

Kosten: $2rs$

Die $(2r+1)$ -te Anfrage fragt den Punkt $k+1$ an.
WFA bewegt den Server auf 1 nach $k+1$

Kosten: rs

Die nächsten $k-2$ Anfragen $1 \ 2 \ 3 \ \dots \ k-2$ bedient WFA, indem bei der Anfrage i , $i \in \{1, \dots, k-2\}$

der Server auf $i+1$ nach i bewegt wird.

Kosten pro Anfrage: $2rs$

Gesamtkosten: $(k-2)2rs$

Dennzufolge bedient WFA 5^{2r+k} mit Kosten

$$(2k-1) \cdot rs$$

Frage:

Dies sieht fast wie eine $2k-1$ untere Schranke für das competitive Verhältnis aus. Wom besteht der Denkfehler?

Antwort:

Obige Folge hat konstante Länge. Für eine untere Schranke benötigen wir eine Folge von beliebig Länge. Zur Konstruktion einer beliebig langen Folge müsste obige Konstruktion wiederholt werden. Hierin wäre eine Anfrage bezüglich Punkt $k-1$ notwendig, da = mit einer der Server auf k bzw. $k+1$ nach $k-1$ bewegt wird, so dass sowohl für OPT_{on} als auch für WFA die Ausgangssituation wieder gegeben ist. Das = nach gilt aber:

Kosten für OPT_{on} : $2rs$

Kosten für WFA : $k \cdot 2rs$

Dennnach wäre WFA eher k -competitive.

Ziel:

Beweis, dass eine Konstante c existiert, so dass für alle Anfragefolgen σ

$$\text{WFA}(\sigma) \leq (2k-1) \text{OPT}(\sigma) + c.$$

Annahme:

OPT und WFA starten in derselben Konfiguration und enden auch in derselben Konfiguration.

Falls WFA in einer anderen Konfiguration als OPT endet, dann könnten wir die Anfragefolge bezüglich Punkten in der Endkonfiguration von OPT verlängern, bis schließlich WFA auch in der Endkonfiguration von OPT gelangt ist.

Effekt:

WFA hat höchstens höhere Kosten während die Kosten von OPT unverändert bleiben.

Demzufolge bedeutet obige Annahme keine Einschränkung.

1. Ziel

Verteilung der optimalen offline-Kosten auf die einzelnen Anfragen.

Hierzu definieren wir wie folgt die offline Pseudokosten $\tilde{\text{opt}}_t$ der t -ten Anfrage:

$$\tilde{\text{opt}}_t := \text{opt}_t(\sigma, X_t) - \text{opt}_{t-1}(\sigma, X_{t-1})$$

Es gilt:

$$\begin{aligned} \sum_{t=1}^n \tilde{\text{opt}}_t &= \text{opt}_n(\sigma, X_n) - \text{opt}_0(\sigma, X_0) \\ &= \text{OPT}(\sigma). \end{aligned}$$

Bereichne WFA_t die Kosten von WFA für die t -te Anfrage.

2. Ziel:

Obere Schranke für $\text{WFA}_t + \tilde{\text{opt}}_t$.

Annahme:

WFA bewegt Server von x nach r_t um r_t zu bedienen.

Dann gilt:

$$\begin{aligned} \text{WFA}_t + \tilde{\text{opt}}_t &= d(x, r_t) + \text{opt}_t(\sigma, X_t) - \text{opt}_{t-1}(\sigma, X_{t-1}) \\ &\stackrel{(+)}{=} \text{opt}_t(\sigma, X_{t-1}) - \text{opt}_{t-1}(\sigma, X_{t-1}) \\ &\leq \max_x \{ \text{opt}_t(\sigma, x) - \text{opt}_{t-1}(\sigma, x) \} \end{aligned}$$

Bereichne

$$\text{EC}_t := \max_x \{ \text{opt}_t(\sigma, x) - \text{opt}_{t-1}(\sigma, x) \}$$

die erweiterten Kosten der t -ten Anfrage.

Dann gilt:

$$\text{WFA}_t + \tilde{\text{opt}}_t \leq \text{EC}_t,$$

womit das 2. Ziel erreicht ist.

3. Ziel:

Obere Schranke für EC_t .

Idee (Koutsoupias):

Konstruktion einer Potentialfunktion ϕ mit:

- a) $EC_t \leq \phi_t - \phi_{t-1}$ und
- b) $\phi_n - \phi_0 \leq (k+1) \text{OPT}(\sigma) + c$.

Dann gilt:

$$\text{WFA}(\sigma) + \text{OPT}(\sigma) = \sum_t (\text{WFA}_t + \tilde{\text{opt}}_t)$$

$$\begin{aligned} &\leq \sum_t EC_t \\ &\leq \sum_t (\phi_t - \phi_{t-1}) \\ &= \phi_n - \phi_0 \\ &\leq (k+1) \text{OPT}(\sigma) + c \end{aligned}$$

$$\Leftrightarrow \text{WFA}(\sigma) \leq k \text{OPT}(\sigma) + c.$$

$\Rightarrow$ WFA ist k -competitive.

Spezialfall: $|S| = k+1$.

$$\text{Wähle } \phi_t := \sum_x \text{opt}_t(\sigma, x).$$

Dann gilt:

$$\begin{aligned}\Phi_t - \Phi_{t-1} &= \sum_x (\text{opt}_t(\sigma, x) - \text{opt}_{t-1}(\sigma, x)) \\ &\geq EC_t\end{aligned}$$

und

$$\begin{aligned}\Phi_n - \Phi_0 &\leq \sum_x \text{opt}_n(\sigma, x) \\ &\leq \sum_x [\text{opt}_n(\sigma, x_n) + D(x_n, x)] \\ &\stackrel{|S| = k+1}{\leq} (k+1) [\text{opt}_n(\sigma, x_n) + \max_{i,j} d(i,j)] \\ &= (k+1) \text{OPT}(\sigma) + \underbrace{(k+1) \max_{i,j} d(i,j)}_{:= C}\end{aligned}$$

□

4. Ziel:

Charakterisierung der Konfigurationen, in denen die erweiterten Kosten angenommen werden.

Intuition:

Falls eine Konfiguration X die erweiterten Kosten annimmt, dann besitzt X folgende Eigenschaften:

- 1) Alle Punkte in X müssen genügend weit vom Anfragepunkt τ_t entfernt sein.
(Ansonsten ist $\text{opt}_t(\sigma, x)$ nicht signifikant größer als $\text{opt}_{t-1}(\sigma, x)$.)
- 2) $\text{opt}_{t-1}(\sigma, x)$ darf nicht zu groß sein.

Begründung:

Sei Z eine Konfiguration mit $r_t \in Z$. Dann gilt:

$$\text{opt}_{t-1}(\sigma, x) \leq \text{opt}_{t-1}(\sigma, Z) + D(Z, x)$$

Beh.:

$$\text{opt}_{t-1}(\sigma, x) < \text{opt}_{t-1}(\sigma, Z) + D(Z, x)$$

Obige Behauptung besagt, dass für alle Konfigurationen Z mit $r_t \in Z$ $\text{opt}_{t-1}(\sigma, x)$ strikt kleiner ist, als $\text{opt}_{t-1}(\sigma, Z) + D(Z, x)$.

Bew. d. Beh.:

Annahme: $\text{opt}_{t-1}(\sigma, x) = \text{opt}_{t-1}(\sigma, Z) + D(Z, x)$

$\Rightarrow$

$$\text{opt}_t(\sigma, x) = \min_{Y | r_t \in Y} \{ \text{opt}_{t-1}(\sigma, Y) + D(Y, x) \}$$

$$\leq \text{opt}_{t-1}(\sigma, Z) + D(Z, x)$$

$$= \text{opt}_{t-1}(\sigma, x).$$

Da Arbeitsfunktionen monoton wachsend sind, folgt somit

$$\text{opt}_t(\sigma, x) = \text{opt}_{t-1}(\sigma, x).$$

Dies ist ein Widerspruch dazu, dass die erweiterten Kosten bezüglich der Konfiguration x angenommen werden.

□

Obige Intuition motiviert folgende Definition:

Eine Konfiguration A heißt Minimierer für r_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$, falls A folgenden Ausdruck minimiert:

$$\text{opt}_{t-1}(\sigma, x) - \sum_{x \in X} d(r_t, x)$$

↖ Eigenschaft 2
↖ Eigenschaft 1

Frage:

Warum nicht

$$\text{opt}_{t-1}(\sigma, x) - \min \{ d(r_t, x) \mid x \in X \} ?$$

Eine Antwort auf diese Frage erhalten wir, indem wir mit diesem statt obigen Ausdruck die nachfolgende Analyse durchführen.

Lemma 3.4 (Dualitätslemma)

Sei die Konfiguration A ein Minimierer für r_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$. Dann gilt:

a) $EC_t = \text{opt}_t(\sigma, A) - \text{opt}_{t-1}(\sigma, A)$

(D.h., A ist auch ein Maximierer für r_t bzgl. EC_t)

b) A ist auch ein Minimierer für r_t bzgl. $\text{opt}_t(\sigma, \cdot)$

Bevor wir das Dualitätslemma beweisen, führen wir die Analyse des Arbeitsfunktionenalgorithmus zu Ende.

5. Ziel:

Definition der Potentialfunktion.

Idee:

Verwende das Dualitätslemma zur Definition einer Potentialfunktion Φ , so dass

$$a) \quad EC_t \leq \Phi_t - \Phi_{t-1} \quad \text{und}$$

b) Ferner soll

$$\Phi_n - \Phi_0 \leq f(k) \text{OPT}(\sigma) + c$$

für eine Konstante c und möglichst kleinem $f(k)$ sein. (Wir bekommen $f(k) = 2 \cdot k$ hin).

Durchführung:

• Seien

$$U := (u_1, u_2, \dots, u_k) \quad \text{und}$$

$$B_i := (b_{i1}, b_{i2}, \dots, b_{ik}) \quad 1 \leq i \leq k$$

$k+1$ Konfigurationen.

• Sei ψ eine Funktion, definiert durch

$$\psi(\text{opt}_t, U, B_1, B_2, \dots, B_k)$$

$$:= k \cdot \text{opt}_t(\sigma, U) + \sum_{i=1}^k (\text{opt}_t(\sigma, B_i) - \sum_{j=1}^k d(u_j, b_{ji}))$$

• Φ_t ist dann definiert als minimaler Wert von ψ über alle möglichen Konfigurationen $U, B_1, B_2, \dots, B_k$
D.h.,

$$\phi_t := \min_{u, B_1, \dots, B_k} \psi(\text{opt}_t, u, B_1, B_2, \dots, B_k).$$

6. Ziel:

Charakterisierung der Konfigurationen $u, B_1, B_2, \dots, B_k$, die ϕ_t definieren.

Lemma 3.5

Sei ϕ_t durch $\tilde{u}, \tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k$ definiert. Sei A ein Minimierer für r_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$.

Dann existiert ein $i \in \{1, 2, \dots, k\}$, so dass

$$\phi_t \geq \psi_t := \psi(\text{opt}_t, \tilde{u} - u_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k)$$

Interpretation:

Wir können annehmen, dass $\tilde{u}$ den letzten angefragten Punkt r_t enthält und eine der Konfigurationen $\tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k$ ein Minimierer bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$ ist.

Beweis:

Gemäß Definition gilt:

$$\begin{aligned} \phi_t &= \psi(\text{opt}_t, \tilde{u}, \tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k) \\ (*) \quad &= k \cdot \text{opt}_t(\sigma, \tilde{u}) + \sum_{\ell=1}^k \left(\text{opt}_t(\sigma, \tilde{B}_\ell) - \sum_{j=1}^k d(\tilde{u}_\ell, \tilde{B}_{\ell j}) \right) \end{aligned}$$

Lemma 3.2.e) $\Rightarrow$

$$\text{opt}_t(\sigma, \tilde{u}) = \min_{\tilde{u} \in \tilde{U}} \left\{ \text{opt}_t(\sigma, \tilde{u} - \tilde{u} + r_t) + d(r_t, \tilde{u}) \right\}.$$

Sei $\tilde{u}_i \in \tilde{U}$ ein Punkt, bzgl. dem das Minimum angenommen wird. D.h.,

$$\text{opt}_t(\sigma, \tilde{u}) = \text{opt}_t(\sigma, \tilde{u} - \tilde{u}_i + r_t) + d(r_t, \tilde{u}_i)$$

Eingesetzt in (*) ergibt dies:

$$\begin{aligned} \Phi_t &= \frac{1}{2} \left[\text{opt}_t(\sigma, \tilde{u} - \tilde{u}_i + r_t) + d(r_t, \tilde{u}_i) \right] \\ &\quad + \sum_{e=1}^k \left(\text{opt}_t(\sigma, \tilde{B}_e) - \sum_{j=1}^k d(\tilde{u}_e, \tilde{b}_{ej}) \right). \end{aligned}$$

$$\begin{aligned} &= \frac{1}{2} \cdot \text{opt}_t(\sigma, \tilde{u} - \tilde{u}_i + r_t) \\ &\quad + \sum_{\substack{e=1 \\ e \neq i}}^k \left(\text{opt}_t(\sigma, \tilde{B}_e) - \sum_{j=1}^k d(\tilde{u}_e, \tilde{b}_{ej}) \right) \\ &\quad + \text{opt}_t(\sigma, \tilde{B}_i) - \underbrace{\sum_{j=1}^k \left(d(\tilde{u}_i, \tilde{b}_{ij}) - d(r_t, \tilde{u}_i) \right)}_{\substack{\geq \\ \Delta\text{-Ungl. +} \\ \text{Symmetrie}}} - \sum_{j=1}^k d(r_t, \tilde{b}_{ij}) \end{aligned}$$

(Beachten wir $\Delta\text{-Ungl.} \Rightarrow d(\tilde{u}_i, r_t) + d(r_t, \tilde{b}_{ij}) \geq d(\tilde{u}_i, \tilde{b}_{ij})$)

$$\geq \Psi(\text{opt}_t, \tilde{u} - \tilde{u}_i + r_t, \tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k).$$

Lemma 3.4.5) $\Rightarrow$

A ist Minimierer für r_t bzgl. $\text{opt}_t(\sigma, \cdot)$.

$$\Rightarrow \text{opt}_t(\sigma, \tilde{B}_i) - \sum_{j=1}^k d(r_t, \tilde{b}_{ij})$$

ist minimal für $\tilde{B}_i = A$.

Also gilt

$$\Phi_t \geq \psi(\text{opt}_t, \tilde{u} - \tilde{u}_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k)$$

Als nächstes zeigen wir, dass $\Phi_t - \Phi_{t-1}$ eine obere Schranke für die erweiterten Kosten EC_t der t -ten Anfrage ist.

Lemma 3.6

Für alle t gilt: $EC_t \leq \Phi_t - \Phi_{t-1}$.

Beweis:

Annahme:

Φ_t wird durch die Konfiguration $\tilde{u}, \tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k$ definiert.

Sei $A := \{\tilde{a}_1, \tilde{a}_2, \dots, \tilde{a}_k\}$ ein Minimierer für r_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$.

$$\text{Lemma 3.5} \Rightarrow \Phi_t \geq \Psi_t$$

$$\Rightarrow \Phi_t - \Phi_{t-1} \geq \Psi_t - \Psi_{t-1},$$

wobei

$$\bar{\Psi}_{t-1} := \psi(\text{opt}_{t-1}, \tilde{u} - \tilde{u}_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k)$$

Beachten wir, dass

$$\bar{\Psi}_{t-1} \geq \Phi_{t-1},$$

da Φ_{t-1} als Minimum über allen möglichen $k+1$ Konfigurationen definiert ist. In $\bar{\Psi}_{t-1}$ haben wir $k+1$ spezielle Konfigurationen gewählt, was den Wert höchstens vergrößert.

Es gilt:

$$\begin{aligned} \Psi_t - \bar{\Psi}_{t-1} &= \psi(\text{opt}_t, \tilde{u} - \tilde{u}_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k) \\ &\quad - \psi(\text{opt}_{t-1}, \tilde{u} - \tilde{u}_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k) \\ &= k \cdot [\text{opt}_t(\sigma, \tilde{u} - \tilde{u}_i + r_t) - \text{opt}_{t-1}(\sigma, \tilde{u} - \tilde{u}_i + r_t)] \\ &\quad + \sum_{\substack{e=1 \\ e \neq i}}^k [\text{opt}_t(\sigma, \tilde{B}_e) - \text{opt}_{t-1}(\sigma, \tilde{B}_e)] \\ &\quad + \text{opt}_t(\sigma, A) - \text{opt}_{t-1}(\sigma, A) \\ &\stackrel{\text{Le. 3.2.c)}}{\geq} \text{opt}_t(\sigma, A) - \text{opt}_{t-1}(\sigma, A) \\ &\stackrel{\text{Le. 3.4.a)}}{=} EC_t \end{aligned}$$

Nun können wir beweisen, dass WFA $(2k-1)$ -competitive ist.

Satz 3.4

WFA ist $(2k-1)$ -competitive.

Beweis:

Es gilt:

$$\begin{aligned}
 \text{WFA}(\sigma) + \text{OPT}(\sigma) &= \sum_{t=1}^n (\text{WFA}_t + \widetilde{\text{opt}}_t) \\
 &\leq \sum_{t=1}^n \text{EC}_t \\
 &\stackrel{\text{L. 3.6}}{\leq} \sum_{t=1}^n (\phi_t - \phi_{t-1}) \\
 &= \phi_n - \phi_0.
 \end{aligned}$$

Sei $X_n = \{x'_1, x'_2, \dots, x'_k\}$. Dann gilt:

$$\begin{aligned}
 \phi_n &\leq \psi(\text{opt}_n, X_n, X_n, \dots, X_n) \\
 &= k \cdot \text{opt}_n(\sigma, X_n) + \sum_{i=1}^k \left[\text{opt}_n(\sigma, X_n) - \sum_{j=1}^k d(x'_i, x'_j) \right] \\
 &= 2k \cdot \text{opt}_n(\sigma, X_n) - \sum_{i=1}^k \sum_{j=1}^k d(x'_i, x'_j) \\
 &\leq 2k \cdot \text{opt}_n(\sigma, X_n) \\
 &= 2k \cdot \text{OPT}(\sigma).
 \end{aligned}$$

Also gilt

$$\begin{aligned}
 \phi_n - \phi_0 &\leq 2k \cdot \text{OPT}(\sigma) - \phi_0 \\
 \Rightarrow \text{WFA}(\sigma) &\leq (2k-1) \text{OPT}(\sigma) - \phi_0
 \end{aligned}$$

Beh.: $\phi_0 \geq -c$ für eine Konstante c .

Somit folgt aus obiger Behauptung direkt, dass WFA $(2k-1)$ -competitive ist.

Bew. d. Beh.:

Annahme: $\phi_0 = \psi(\text{opt}_0, X_0, X_0, \dots, X_0)$

Dann gilt:

$$\begin{aligned}\phi_0 &= \psi(\text{opt}_0, X_0, X_0, \dots, X_0) \\ &= \sum_{e=1}^k \text{opt}_0(\sigma, X_0) + \sum_{e=1}^k \left(\text{opt}_0(\sigma, X_0) - \sum_{j=1}^k d(x_e, x_j) \right),\end{aligned}$$

wobei $X_0 := (x_1, x_2, \dots, x_k)$.

$$= - \sum_{e=1}^k \sum_{j=1}^k d(x_e, x_j),$$

was eine Konstante ist, die nur von der Startkonfiguration und nicht von der Anfragefolge σ abhängt.

Es verbleibt noch der Beweis der obigen Annahme.

Übung.

Es fehlt noch der Beweis des Dualitätslemmas

Lemma 3.4 (Dualitätslemma)

Sei die Konfiguration A ein Minimierer für σ_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$. Dann gilt

a) $EC_t = \text{opt}_t(\sigma, A) - \text{opt}_{t-1}(\sigma, A)$

b) A ist auch ein Minimierer für σ_t bzgl. $\text{opt}_t(\sigma, \cdot)$.

Beweis: (Michel X. Goemans course notes)
über Homepage beziehbar.

Folgendes Lemma wird für den Beweis benötigt:

Lemma 3.7

$$\forall t \forall A, B \forall a \in A \exists b \in B \text{ mit} \\ \text{opt}_t(\sigma, A) + \text{opt}_t(\sigma, B) \geq \text{opt}_t(\sigma, A - a + b) + \text{opt}_t(\sigma, B - b + a).$$

Bevor wir Lemma 3.7 beweisen, beweisen wir unter Verwendung dieses Lemmas das Dualitätslemma.

a) Zu zeigen: $\forall B$ gilt

$$\text{opt}_t(\sigma, A) - \text{opt}_{t-1}(\sigma, A) \geq \text{opt}_t(\sigma, B) - \text{opt}_{t-1}(\sigma, B)$$

$$\Leftrightarrow \text{opt}_t(\sigma, A) + \text{opt}_{t-1}(\sigma, B) \geq \text{opt}_t(\sigma, B) + \text{opt}_{t-1}(\sigma, A)$$

$\Leftrightarrow \forall a \in A$ gilt:

Le 3.2.b)

$$\text{opt}_{t-1}(\sigma, A - a + r_t) + d(r_t, a) + \text{opt}_{t-1}(\sigma, B)$$

$$\geq \min_{b \in B} \{ \text{opt}_{t-1}(\sigma, B - b + r_t) + d(r_t, b) + \text{opt}_{t-1}(\sigma, A) \}$$

Da A ein Minimierer für r_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$ ist, gilt:

$$\text{opt}_{t-1}(\sigma, A) - \sum_{x \in A} d(r_t, x) \leq \text{opt}_{t-1}(\sigma, A - a + b) - \sum_{y \in A - a + b} d(r_t, y).$$

$$\Leftrightarrow \text{opt}_{t-1}(\sigma, A) + d(r_t, b) \leq \text{opt}_{t-1}(\sigma, A - a + b) + d(r_t, a)$$

$$\Leftrightarrow d(r_t, a) \geq \text{opt}_{t-1}(\sigma, A) + d(r_t, b) - \text{opt}_{t-1}(\sigma, A - a + b)$$

Also genügt es folgendes zu beweisen:

$$\text{opt}_{t-1}(\sigma, A - a + r_t) + \text{opt}_{t-1}(\sigma, B)$$

$$\geq \min_{b \in B} \{ \text{opt}_{t-1}(\sigma, B - b + r_t) + \text{opt}_{t-1}(\sigma, A - a + b) \}$$

Wegen $r_t \in A - a + r_t$ und Lemma 3.7 gilt:

$\exists b \in B$ mit

$$\text{opt}_{t-1}(\sigma, A - a + r_t) + \text{opt}_{t-1}(\sigma, B)$$

$$\geq \text{opt}_{t-1}(\sigma, A - a + r_t - r_t + b) + \text{opt}_{t-1}(\sigma, B - b + r_t)$$

$$\Leftrightarrow \text{opt}_{t-1}(\sigma, A - a + r_t) + \text{opt}_{t-1}(\sigma, B)$$

$$\geq \text{opt}_{t-1}(\sigma, B - b + r_t) + \text{opt}_{t-1}(\sigma, A - a + b)$$

q.e.d.

b) zu zeigen: $\forall B$ gilt:

$$\text{opt}_t(\sigma, A) - \sum_{a \in A} d(r_t, a) \leq \text{opt}_t(\sigma, B) - \sum_{b \in B} d(r_t, b)$$

$\Leftrightarrow$
Le 3.2.5) $\forall y \in B$ gilt:

$$\min_{x \in A} \{ \text{opt}_{t-1}(\sigma, A - x + r_t) + d(r_t, x) - \sum_{a \in A} d(r_t, a) \}$$

$$\leq \text{opt}_{t-1}(\sigma, B - y + r_t) + d(r_t, y) - \sum_{b \in B} d(r_t, b)$$

Betrachten wir $y \in B$ beliebig, aber fest.

Da A ein Minimierer für r_t bzgl. $\text{opt}_{t-1}(\sigma, \cdot)$ gilt:

$$\text{opt}_{t-1}(\sigma, A) - \sum_{a \in A} d(r_t, a) \leq \text{opt}_{t-1}(\sigma, B - y + x) - \sum_{b \in B - y + x} d(r_t, b).$$

Nach Einsetzung von

$$\sum_{b \in B} d(r_t, b) - d(r_t, y) + d(r_t, x) \text{ für } \sum_{b \in B - y + x} d(r_t, b)$$

erhalten wir:

$$\begin{aligned} \text{opt}_{t-1}(\sigma, A) - \sum_{a \in A} d(r_t, a) \\ \leq \text{opt}_{t-1}(\sigma, B - y + x) - \left(\sum_{b \in B} d(r_t, b) - d(r_t, y) + d(r_t, x) \right) \end{aligned}$$

$\Leftrightarrow$

$$\begin{aligned} (*) \quad -\text{opt}_{t-1}(\sigma, B - y + x) + \text{opt}_{t-1}(\sigma, A) + d(r_t, x) - \sum_{a \in A} d(r_t, a) \\ \leq d(r_t, y) - \sum_{b \in B} d(r_t, b) \end{aligned}$$

Wegen $r_t \in B - y + r_t$ impliziert Lemma 2.7:

$\exists x \in A$ mit

$$\begin{aligned} (**) \quad \text{opt}_{t-1}(\sigma, B - y + x) + \text{opt}_{t-1}(\sigma, A - x + r_t) \\ \leq \text{opt}_{t-1}(\sigma, B - y + r_t) + \text{opt}_{t-1}(\sigma, A) \end{aligned}$$

Addition von (*) und (**) gefolgt von Subtraktion von $\text{opt}_{t-1}(\sigma, A)$ auf beiden Seiten ergibt:

$$\begin{aligned} \text{opt}_{t-1}(\sigma, A - x + r_t) + d(r_t, x) - \sum_{a \in A} d(r_t, a) \\ \leq \text{opt}_{t-1}(\sigma, B - y + r_t) + d(r_t, y) - \sum_{b \in B} d(r_t, b) \end{aligned}$$

Beweis von Lemma 3.7: (durch Induktion über t)

$t = 0$:

Sei X_0 die Startkonfiguration. Dann gilt:

$$\text{opt}_0(\sigma, x) = D(X_0, x) \quad \forall \text{ Konfigurationen } x.$$

Seien M_A bzw. M_B zwei Matchings minimaler Kosten zwischen A und X_0 bzw. B und X_0 .

Für $a \in A$ beliebig, aber fest betrachten wir

$$\begin{aligned} a_0 \in X_0 \text{ mit } (a, a_0) \in M_A \text{ und} \\ b \in B \text{ mit } (b, a_0) \in M_B. \end{aligned}$$

Dann gilt:

$$\text{opt}_0(\sigma, A) + \text{opt}_0(\sigma, B)$$

$$= D(X_0, A) + D(X_0, B)$$

$$\stackrel{\substack{\rightarrow = \\ \text{wahl von } a_0 \\ \text{und von } b}}{=} D(X_0 - a_0, A - a) + D(X_0 - a_0, B - b) + d(a_0, a) + d(a_0, b)$$

$$\geq D(X_0, A - a + b) + D(X_0, B - b + a)$$

$$= \text{opt}_0(\sigma, A - a + b) + \text{opt}_0(\sigma, B - b + a)$$

$t > 0$:

Annahme: Die Behauptung gilt für $t-1$

$t-1 \leadsto t$:

Sei $a \in A$ beliebig, aber fest.

Lemma 3.2.6 $\Rightarrow$

$$(+)\begin{cases} \exists x \in A : \text{opt}_t(\sigma, A) = \text{opt}_{t-1}(\sigma, A - x + r_t) + d(r_t, x) \\ \exists y \in B : \text{opt}_t(\sigma, B) = \text{opt}_{t-1}(\sigma, B - y + r_t) + d(r_t, y) \end{cases}$$

Wir unterscheiden zwei Fälle:

1. Fall: $x = a$

Setze $b := y$.

Definition von $\text{opt}_t \Rightarrow$

$$\text{opt}_t(\sigma, A - a + b) \leq \text{opt}_{t-1}(\sigma, A - a + r_t) + d(r_t, b)$$

und

$$\text{opt}_t(\sigma, B - b + a) \leq \text{opt}_{t-1}(\sigma, B - b + r_t) + d(r_t, a).$$

Die Addition der beiden Ungleichungen unter Berücksichtigung von (+) ergibt:

$$\begin{aligned} \text{opt}_t(\sigma, A - a + b) + \text{opt}_t(\sigma, B - b + a) \\ \leq \text{opt}_t(\sigma, A) + \text{opt}_t(\sigma, B) \end{aligned}$$

q.e.d.

2. Fall: $x \neq a$

Dann gilt $a \in A - x$.

Die Anwendung der Induktionshypothese auf $A - x + r_t$ und $B - y + r_t$ ergibt:

$\exists b \in B - y + r_t$ mit

$$\begin{aligned} & \text{opt}_{t-1}(\sigma, A - x + r_t) + \text{opt}_{t-1}(\sigma, B - y + r_t) \\ (+) & \geq \text{opt}_{t-1}(\sigma, A - x + r_t - a + b) + \text{opt}_{t-1}(\sigma, B - y + r_t - b + a) \end{aligned}$$

Ferner gilt wegen Lemma 3.2. b):

$$(+++) \begin{cases} \text{opt}_t(\sigma, A - a + b) \leq \text{opt}_{t-1}(\sigma, A - a + b - x + r_t) + d(r_t, x) \\ \text{opt}_t(\sigma, B - b + a) \leq \text{opt}_{t-1}(\sigma, B - b + a - y + r_t) + d(r_t, y) \end{cases}$$

Insgesamt haben wir gezeigt:

$$\text{opt}_t(\sigma, A) + \text{opt}_t(\sigma, B)$$

$$\stackrel{(+)}{=} \text{opt}_{t-1}(\sigma, A - x + r_t) + d(r_t, x) + \text{opt}_{t-1}(\sigma, B - y + r_t) + d(r_t, y)$$

$$\begin{aligned} & \geq \text{opt}_{t-1}(\sigma, A - x + r_t - a + b) + d(r_t, x) \\ (+) & \quad + \text{opt}_{t-1}(\sigma, B - y + r_t - b + a) + d(r_t, y) \end{aligned}$$

$$\stackrel{(+++)}{\geq} \text{opt}_t(\sigma, A - a + b) + \text{opt}_t(\sigma, B - b + a)$$

q.e.d.



4. Metrische Aufgabensysteme (Metrical Task Systems)

Eiscremeproblem:

Maschine M kann zwei Sorten Eis produzieren.

Sorte	Kosten
Vanille	1
Schokolade	2

- nur eine Sorte gleichzeitig
- Kosten für Sortenwechsel betragen 1.

Ohne Maschine kann Eis mit folgenden Kosten produziert werden:

Sorte	Kosten
Vanille	2
Schokolade	4

Bestellung einer Einheit Eis (Vanille oder Schokolade)



- Verwendung von M (eventuell mit Sortenwechsel)
- ohne Verwendung von M.

Frage:

Wie soll bei einer Folge von Bestellungen das Eis produziert werden?

Ziel dabei ist, die Kosten zu minimieren.

Das Eiscremeproblem ist ein Beispiel für ein metrisches Aufgabensystem. Formel ist ein metrisches Aufgabensystem wie folgt definiert:

Sei $M = (S, d)$ ein metrischer Raum. Einer Aufgabe τ ist eine Abbildung $\tau: S \mapsto \mathbb{R}^+$ zugeordnet. Dabei bezeichnet $\tau(x)$, $x \in S$ die Kosten für die Erledigung der Aufgabe τ im Zustand x .

Sei T eine endliche Menge von Aufgaben. Dann heißt das Paar (M, T) metrisches Aufgabensystem (MTS).

Seien

- $x_0 \in S$ der Startzustand
- $\bar{\tau} = \tau_1, \tau_2, \dots, \tau_m$ eine Aufgabenfolge und
- $\bar{x} = x_1, x_2, \dots, x_m$, $x_i \in S$ ein (Service)schedule

Definiere

$$\text{cost}(x_0, \bar{\tau}, \bar{x}) = \sum_{i=1}^m [d(x_{i-1}, x_i) + \tau_i(x_i)]$$

und

$$\text{opt}(x_0, \bar{\tau}) = \min_{\bar{x}} \{ \text{cost}(x_0, \bar{\tau}, \bar{x}) \}$$

Ziel:

Entwicklung von Online-Strategien für MTS.

Sei $ALG: S \times T^* \rightarrow S$ eine Abbildung.

Die Abbildung ALG berechnet, gegeben einen Startzustand x_0 , eine Aufgabenfolge $\bar{J}$ einen Zustand $ALG(x_0, \bar{J}) \in S$.

Seien $\bar{J} = J_1 J_2 \dots J_m$ und $x_i = ALG(x_0, J_1 \dots J_i)$ für $1 \leq i \leq m$. Die Folge $x_0, x_1, x_2, \dots, x_m$ heißt der von ALG produzierte Schedule.

Die Kosten $cost_{ALG}(x_0, \bar{J})$ von ALG auf $x_0, \bar{J}$ sind definiert durch

$$cost_{ALG}(x_0, \bar{J}) := cost(x_0, \bar{J}, \bar{x}),$$

wobei $\bar{x}$ der von ALG produzierte Schedule ist.

ALG heißt c-competitive mit Startfunktion $\alpha: S \rightarrow \mathbb{R}$, falls $\forall x_0 \in S \ \forall \bar{J} \in T^*$ gilt:

$$cost_{ALG}(x_0, \bar{J}) \leq c \cdot opt(x_0, \bar{J}) + \alpha(x_0)$$

Die kleinste Konstante c mit ALG ist c-competitive heißt competitives Verhältnis von ALG .

Ein metrisches Dienstsystem (MSS) ist ein Paar (M, R) , wobei $M = (S, d)$ ein endlicher metrischer Raum und R eine Familie von nichtleeren Teilmengen von S sind. Ein Element von R heißt Aufgabe. Falls $g \in R$ angefragt wird, dann wird ein Server

178

zu einem Punkt in $\mathcal{P}$ bewegt, wo er dann die Anfrage bearbeitet. $(M, \mathcal{R})$ heißt unbeschränkt, falls $\mathcal{R}$ alle nichtleeren Teilmengen von S enthält.

Übung:

Zeigen Sie, dass MSS eine Verallgemeinerung des k -Server-Problems ist.

Analog zum k -Server-Problem können wir auch bei metrischen Aufgabensystemen Arbeitsfunktionen definieren um dann mit Hilfe dieser Online-Algorithmen zu entwickeln.

~~09.04.~~

Seien (M, T) ein metrisches Aufgabensystem, x_0 ein Startzustand und $\bar{T} = T_1 T_2 \dots T_m$ eine Aufgabenfolge. Die zu x_0 und $\bar{T}$ assoziierte Arbeitsfunktion $w: S \rightarrow \mathbb{R}$ ist wie folgt definiert:

$$w(x) := \min_{\bar{x}} \{ \text{cost}(x_0, \bar{T}, \bar{x}) + d(x_m, x) \},$$

wobei $\bar{x} = x_0 x_1 \dots x_m$. D.h., $w(x)$ sind die minimalen Kosten zur Erledigung von $\bar{T}$, gestartet in x_0 und beendet in x .

Bemerkung:

Ursprünglich wurden Arbeitsfunktionen für metrische Aufgabensysteme erfunden und fanden danach ihre Anwendung beim k -Server-Problem.

Beobachtung:

Die optimalen Kosten $\text{opt}(x_0, \bar{S})$ zur Erledigung von $\bar{S}$ gestartet in x_0 ergeben sich aus

$$\text{opt}(x_0, \bar{S}) = \min \{ w(x) \mid x \in S \}.$$

Zu Aufgabenfolgen assoziierte Arbeitsfunktionen heißen erreichbar.

Bemerkung:

Beim k -Server-Problem haben wir gesehen, dass die Arbeitsfunktion sämtliche Information enthält, die der Arbeitsfunktionalgorithmus benötigt, um den nächsten Schritt zu entscheiden. Auch hier werden wir einen Arbeitsfunktionalgorithmus entwickeln, für den dann auch entsprechendes erfüllt sein wird.

Ziel:

Entwicklung von Werkzeugen zum schrittweisen Update der Arbeitsfunktion.

Zum Erreichen des Zieles ist zunächst nützlich, Eigenschaften von erreichbaren Arbeitsfunktionen herauszuarbeiten.

Eigenschaften:

$$\bullet w(x) - w(y) \leq d(x, y) \quad \forall x, y \in S$$

- Berechnungen:

w wird gestützt von $K \subseteq S$, falls $\forall x \in S$
 $\exists y \in K$ mit $w(x) = w(y) + d(y, x)$.

Eine Stützmenge für w ist eine kleinste Teilmenge $K \subseteq S$, die w stützt.

Eine Arbeitsfunktion, die von einem einzelnen Punkt in S gestützt wird, heißt Kegel.

X_x bezeichnet den von $\{x\}$ gestützten Kegel mit $w(x) = 0$.

Sei x_0 der Startzustand. Dann ist X_{x_0} die Startarbeitsfunktion.

Sei $X \subseteq S$. X_x bezeichnet den verallgemeinerten Kegel auf X . D.h., diejenige Arbeitsfunktion w , für die gilt:

$$X \text{ stützt } w \text{ und } w(x) = 0 \quad \forall x \in X.$$

- Update der Arbeitsfunktion (Startzustand x_0)

- Startarbeitsfunktion X_{x_0}

- Arbeitsfunktion w , $\tau \in T$. Dann bezeichnet $w \wedge \tau$ folgende Arbeitsfunktion:

$$w \wedge \tau (x) := \min_{y \in S} \{ w(y) + \tau(y) + d(y, x) \}$$

$\uparrow$ Updateoperator $\forall x \in S$.

Sei $\bar{J} = J_1 J_2 \dots J_m$ eine Aufgabenfolge. Dann schreiben wir

$$w \wedge \bar{J} := w \wedge J_1 \wedge J_2 \wedge \dots \wedge J_m.$$

Lemma 4.1

$$w \wedge J(x) = w(y) + J(y) + d(y, x)$$

$\Rightarrow$

$$w \wedge J(y) = w(y) + J(y).$$

Beweis:

$\forall z \in S$ gilt wegen $w \wedge J(x) \leq w(z) + J(z) + d(z, x)$

$$w(y) + J(y) + d(y, x) \leq w(z) + J(z) + d(z, x)$$

$$\Leftrightarrow w(y) + J(y) \leq w(z) + J(z) + d(z, x) - d(y, x)$$

$$\leq w(z) + J(z) + d(z, y)$$

Δ -Ungl.

Also folgt aus der Definition von $w \wedge J(y)$

$$w \wedge J(y) = w(y) + J(y).$$



Mit Hilfe des Lemmas 4.1 lässt sich unmittelbar folgendes Korollar beweisen:

Korollar 4.1

Falls x Element der Stützmenge von $w \wedge J$ ist, dann gilt:

$$w \wedge J(x) = w(x) + J(x).$$

Beweis:

Annahme: $w \wedge T(x) \neq w(x) + T(x)$

$\Rightarrow$

$\exists y \in S \setminus \{x\} : w \wedge T(x) = w(y) + T(y) + d(y, x)$

Lemma 4.1 $\Rightarrow$

$$w \wedge T(y) = w(y) + T(y)$$

Also gilt

$$(*) \quad w \wedge T(x) = w \wedge T(y) + d(y, x)$$

Somit gilt für alle $z \in S$:

$$w \wedge T(x) + d(x, z) = w \wedge T(y) + \underbrace{d(y, x) + d(x, z)}_{\geq d(y, z)}$$

$$\geq w \wedge T(y) + d(y, z)$$

$$= w(y) + T(y) + d(y, z)$$

$$\stackrel{\text{Def.}}{\geq} w \wedge T(z)$$

Also gilt für alle $z \in S$ mit

$$w \wedge T(z) = w \wedge T(x) + d(x, z)$$

auch

$$(**) \quad w \wedge T(z) = w \wedge T(y) + d(y, z)$$

Wegen (*) gilt (Beachte $y \neq x$)

$$w \wedge \tau(y) \neq w \wedge \tau(x) + d(x, y).$$

$\Rightarrow$ in der Stützmenge K existiert $u \neq x$ mit

$$w \wedge \tau(y) = w \wedge \tau(u) + d(u, y)$$

In (**) eingesetzt ergibt dies

$$w \wedge \tau(z) = w \wedge \tau(u) + \underbrace{d(u, y) + d(y, z)}_{\geq d(u, z)}$$

$$\geq w \wedge \tau(u) + d(u, z)$$

$$= w(u') + \tau(u') + d(u', u) + d(u, z)$$

für ein $u' \in S$

$$\geq w(u') + \tau(u') + d(u', z)$$

$$\stackrel{\text{Def.}}{\geq} w \wedge \tau(z)$$

Also gilt

$$w \wedge \tau(z) = w \wedge \tau(u) + d(u, z)$$

Also gilt

$$K \setminus \{x\} \text{ stützt } w \wedge \tau.$$

Dies ist ein Widerspruch dazu, dass K eine Stützmenge von $w \wedge \tau$ ist.

Eine auf Arbeitsfunktionen basierende Strategie sieht wie folgt aus:

gegeben: $x \in S$, aktuelle Arbeitsfunktion w und eine neue Aufgabe $J \in T$.

Abbildung $A: S \times \Omega \times T \rightarrow S$,
wobei

- $\Omega = \{ w \mid w \text{ erreichbare Arbeitsfunktion} \}$
- $A(x, w, J) = y$ ist derjenige Zustand, in den das System geht um J zu erledigen.

D.h., die Abbildung A definiert einen auf Arbeitsfunktionen basierenden Online-Algorithmus.

Ziel:

Beweis, dass für ein beliebiges MTS ein optimaler auf Arbeitsfunktionen basierender Online-Algorithmus existiert.

Satz 4.1

Sei ALG ein beliebiger Online-Algorithmus für ein MTS (M, T) , der c -competitive mit Startfunktion $\alpha: S \rightarrow \mathbb{R}^+$ ist. Dann existiert ein auf Arbeitsfunktionen basierender

Online-Algorithmus ALG' , der auch c -competitive mit Startfunktion α ist.

Beweis:

Ziel:

Konstruktion der Abbildung ALG' . D.h.,
Definition der Werte $ALG'(x, w, \bar{J})$.

Für alle $x_0 \in S$ und alle $\bar{J} \in T^*$ definiere

$$L(x_0, \bar{J}) := \text{cost}_{ALG}(x_0, \bar{J}) - c \cdot \text{opt}(x_0, \bar{J})$$

Da ALG c -competitive mit Startfunktion α ist,
folgt aus der Definition

$$L(x_0, \bar{J}) \leq \alpha(x_0).$$

Wir wollen nun erreichbare Paare (x, w) charakterisieren.

Sei $\forall x \in S \forall$ Arbeitsfunktionen w

$$F(x, w) := \left\{ (x_0, \bar{J}) \mid ALG(x_0, \bar{J}) = x \text{ und } \chi_{x_0} \wedge \bar{J} = w \right\}$$

Falls $F(x, w) \neq \emptyset$, dann heißt das Paar (x, w) ALG -erreichbar.

Für ALG -erreichbare (x, w) definieren wir

$$\phi(x, w) = \inf_{\substack{(x_0, \bar{J}) \in \\ F(x, w)}} \sup_{\bar{J} \in T^*} \left\{ L(x_0, \bar{J} \bar{J}) - L(x_0, \bar{J}) \right\}$$

Beobachtung:

- $\phi(x, w) \geq 0$, da $\bar{\sigma} = \varepsilon$ gewählt werden kann.
- $\phi(x_0, \chi_{x_0}) \leq \alpha(x_0)$, da $\bar{\sigma} = \varepsilon$ gewählt werden kann.

Beh.:

Falls (x, w) ALG-erreichbar, dann existiert für alle $\sigma \in T$ ein y in S , so dass

$$d(x, y) + \sigma(y) + \phi(y, w \wedge \sigma) - \phi(x, w) \leq C \cdot \left[\min_z w \wedge \sigma(z) - \min_z w(z) \right]$$

□

Bevor wir die Behauptung beweisen, führen wir den Beweis des Satzes zu Ende.

Sei (x, w) ALG-erreichbar und $\sigma \in T$. Wir definieren dann

$$\text{ALG}'(x, w, \sigma) := y,$$

wobei $y \in S$ mit $d(x, y) + \sigma(y) + \phi(y, w \wedge \sigma)$ minimal.

Für $x_0 \in S$ und $\bar{\sigma} = \sigma_1, \sigma_2, \dots, \sigma_m$ erhalten wir dann

$$\text{cost}_{\text{ALG}'}(x_0, \bar{S})$$

$$= \sum_{i=1}^m (d(x_{i-1}, x_i) + \bar{S}_i(x_i))$$

$$= \sum_{i=1}^m \left[d(x_{i-1}, x_i) + \bar{S}_i(x_i) + (\phi(x_i, \chi_{x_0} \wedge \bar{S}_1 \dots \bar{S}_i) - \phi(x_{i-1}, \chi_{x_0} \wedge \bar{S}_1 \dots \bar{S}_{i-1})) + \underbrace{\phi(x_0, \chi_{x_0}) - \phi(x_m, \chi_{x_0} \wedge \bar{S})}_{\geq 0} \right]$$

$$\stackrel{\text{Beh.}}{\leq} \sum_{i=1}^m c \cdot \left[\min_{z \in S} \chi_{x_0} \wedge \bar{S}_1 \wedge \dots \wedge \bar{S}_i(z) - \min_{z \in S} \chi_{x_0} \wedge \bar{S}_1 \wedge \dots \wedge \bar{S}_{i-1}(z) \right] + \phi(x_0, \chi_{x_0})$$

$$\leq c \cdot \min_{z \in S} \chi_{x_0} \wedge \bar{S}(z) + \phi(x_0, \chi_{x_0})$$

$$\leq c \cdot \text{opt}(x_0, \bar{S}) + \alpha(x_0)$$

Es verbleibt noch der Beweis der Behauptung.

Bew. d. Beh.:

Wähle $(x_0, \bar{S}) \in F(x, w)$ mit

$$\phi(x, w) = \sup_{\bar{S} \in T^*} \{ L(x_0, \bar{S} \bar{S}) - L(x_0, \bar{S}) \}$$

Dann gilt:

$$(*) \quad \phi(x, w) \geq \sup_{\bar{\sigma}' \in T^*} \{L(x_0, \bar{\sigma} \bar{\sigma}') - L(x_0, \bar{\sigma})\},$$

da die Festlegung der ersten Aufgabe in $\bar{\sigma}$ die Auswahl höchstens einschränkt.

Sei

$$y = \text{ALG}(x_0, \bar{\sigma} \bar{\sigma}).$$

Konstruktion $\Rightarrow$

$$(**) \quad \phi(y, w \wedge \bar{\sigma}) \leq \sup_{\bar{\sigma}' \in T^*} \{L(x_0, \bar{\sigma} \bar{\sigma}') - L(x_0, \bar{\sigma} \bar{\sigma})\}$$

$\Rightarrow$

$$d(x, y) + \bar{\sigma}(y) + \phi(y, w \wedge \bar{\sigma}) - \phi(x, w)$$

$$\stackrel{(*) \text{ u. } (**)}{\leq} d(x, y) + \bar{\sigma}(y) + \sup_{\bar{\sigma}' \in T^*} \{L(x_0, \bar{\sigma} \bar{\sigma}') - L(x_0, \bar{\sigma} \bar{\sigma})\}$$

$$- \sup_{\bar{\sigma}' \in T^*} \{L(x_0, \bar{\sigma} \bar{\sigma}') - L(x_0, \bar{\sigma})\}$$

$$= d(x, y) + \bar{\sigma}(y) - (L(x_0, \bar{\sigma} \bar{\sigma}) - L(x_0, \bar{\sigma}))$$

$$= d(x, y) + \bar{\sigma}(y)$$

$$- [\text{cost}_{\text{ALG}}(x_0, \bar{\sigma} \bar{\sigma}) - c \cdot \text{opt}(x_0, \bar{\sigma} \bar{\sigma})]$$

$$- (\text{cost}_{\text{ALG}}(x_0, \bar{\sigma}) - c \cdot \text{opt}(x_0, \bar{\sigma}))]$$

$$\stackrel{\text{Wahl von } y}{=} c \cdot \left(\min_z w \wedge \bar{\sigma}(z) - \min_z w(z) \right)$$

4.1 Durchschnittseigenschaft von metrischen Dienstsystemen

Sei (M, R) ein metrisches Dienstsystem. Ein Online-Algorithmus für (M, R) heißt träge (lazy), falls er niemals seinen Zustand ändert, wenn der aktuelle Zustand Element der Nachfragemenge ist.

Lemma 4.2

Sei ALG ein Online-Algorithmus für ein MSS (M, R) . Dann existiert ein träger Online-Algorithmus ALG' für (M, R) , dessen Kosten niemals größer als die von ALG sind.

Beweis:

Wir beweisen das Lemma, indem wir mit Hilfe von ALG den trägen Online-Algorithmus ALG' konstruieren. Sei

$$ALG'(x_0, \varepsilon) = x_0$$

$$ALG'(x_0, \bar{p}_S) = \begin{cases} ALG'(x_0, \bar{p}) & \text{falls } ALG'(x_0, \bar{p}) \in S \\ ALG(x_0, \bar{p}_S) & \text{sonst.} \end{cases}$$

ALG' arbeitet wie ALG mit der Ausnahme, dass ALG' den Zustand nicht ändert, wenn dieser Element der Anfrage ist. Beim nächsten Mal, wenn

(19)

Dies der Fall ist, geht ALG' in denselben Zustand wie ALG . Die Behauptung folgt nun aus der Δ -Ungleichung.

Folgender Satz ist gerade die Durchschnittseigenschaft von metrischen Dienstsystemen.

Satz 4.2

Sei (M, R) ein MSS. Sei

$$R' = \{ p_1' \cap p_2' \mid p_1', p_2' \in R \text{ und } p_1' \cap p_2' \neq \emptyset \}.$$

Dann hat (M, R) genau dann einen c -competitiven Online-Algorithmus, wenn (M, R') solchen besitzt.

Beweis:

Folgende Behauptung impliziert den Satz:

Beh. 1:

Sei (M, R) ein MSS. Seien $p_1', p_2' \in R$ mit $\emptyset \neq p_1' \cap p_2' \notin R$. Sei $R' = R \cup \{p_1' \cap p_2'\}$. Dann existiert genau dann ein c -competitiver Online-Algorithmus für (M, R) , wenn es einen solchen für (M, R') gibt.

Bew. d. Beh.:

" $\Leftarrow$ "

trivial

" $\Rightarrow$ "

Sei ALG ein träger Online-Algorithmus für (M, R) .

Ziel:

Konstruktion eines Online-Algorithmus ALG' für (M, R') , dessen competitiveness Verhältnis nicht schlechter ist, als das von ALG für (M, R) .

Idee:

Gegeben eine beliebige Anfragefolge $\bar{g}$ für (M, R') definieren wir gleichzeitig das Verhalten von ALG' auf $\bar{g}$ und eine Anfragefolge $\bar{f}$ für (M, R) und zeigen, dass das competitive Verhältnis von ALG' bzgl. $\bar{g}$ nicht schlechter ist, als das von ALG bzgl. $\bar{f}$.

Sei $\bar{g} = g_1 g_2 \dots g_m$. Für $1 \leq i \leq m$ definieren wir in Abhängigkeit von g_i eine Anfragefolge $\bar{f}_i$ für (M, R) und das Verhalten von ALG' bzgl. g_i ; d.h., den Zustand x'_i , in dem ALG' die Anfrage g_i erledigt.

Wir unterscheiden zwei Fälle:

1. Fall: $g_i \in R$

Wir definieren dann:

- $\bar{g}_i := g_i$
- Sei x_i der Zustand, in dem ALG die Anfrage g_i erledigt. Setze $x'_i := x_i$.

2. Fall $g_i \notin R \Rightarrow g_i = g'_1 \cap g'_2$

Wir definieren dann

- $\bar{g}_i := (g'_1 g'_2)^N$, wobei

$$N \geq \frac{D}{d_{\min}} + 1 \quad \text{ganzzahlig mit}$$

D ist der Diameter bzgl. $M = (S, d)$ und $d_{\min}$ ist die kleinste Distanz in M .

- Sei x_i der Zustand, in dem ALG₁ sich nach Abarbeitung der Anfragefolge $\bar{g}_i$ befindet.

Falls $x_i \in g'_1 \cap g'_2$, dann setze

$$x'_i := x_i.$$

Andernfalls setze

$$x'_i \in g'_1 \cap g'_2 \text{ beliebig.}$$

□

Als nächstes werden wir die Kosten der beiden Algorithmen analysieren.

Seien

cost_i' die Kosten von ALG' nach $f_1, f_2, \dots, f_i$

cost_i die Kosten von ALG nach $\bar{f}_1, \bar{f}_2, \dots, \bar{f}_i$.

Aus folgender Behauptung folgt direkt die Beh. 1:

Beh. 2:

i) $\text{opt}(x_0, \bar{f}) \leq \text{opt}(x_0, \bar{g})$

ii) $\text{cost}_i' \leq \text{cost}_i - d(x_i, x_i') \quad 1 \leq i \leq m$

Bew. d. Beh. 2:

i) folgt direkt, da eine beliebige träge Lösung für $\bar{g}$ auch eine träge Lösung für $\bar{f}$ ist.

Wir beweisen ii) durch Induktion über i .

Für $i=0$, d.h., keine Anfrage ist bisher erfolgt, gilt die Behauptung offensichtlich.

Sei $i > 0$.

Annahme:

Die Behauptung gilt für $i-1$.

$i-1 \leadsto i$:

Wir unterscheiden zwei Fälle:

1. Fall: $x_i = x_i'$

Dann gilt:

$$\begin{aligned} \text{cost}_i' &= \text{cost}_{i-1}' + d(x_{i-1}', x_i') \\ &\leq \text{cost}_{i-1} - d(x_{i-1}, x_{i-1}') + d(x_{i-1}', x_i') \\ &\leq \text{cost}_{i-1} + d(x_{i-1}, x_i') \\ &= \text{cost}_i \\ &= \text{cost}_i - d(x_i, x_i') \end{aligned}$$

"0"

2. Fall: $x_i \neq x_i'$

Dann gilt: $\mathcal{S}_i = \mathcal{S}_1' \cap \mathcal{S}_2'$.

Seien $v_1, v_2, \dots, v_{2N} = x_i$ diejenige Zustände, die ALG zur Abarbeitung von $\vec{\mathcal{S}}_i = (\mathcal{S}_1' \mathcal{S}_2')^N$ in dieser Reihenfolge annimmt.

Falls

$$v_\ell \in \mathcal{S}_i \quad \text{für ein } \ell \in \{1, 2, \dots, 2N\},$$

dann impliziert die Trägheit von ALG

$$v_\ell = v_{\ell+1} = \dots = v_{2N} = x_i.$$

Dies ist ein Widerspruch zu $x_i \neq x_i'$.

Also gilt $v_\ell \notin \mathcal{S}_i \quad \forall \ell \in \{1, 2, \dots, 2N\}$.

$\Rightarrow$

$$v_i \in \begin{cases} g_1' \setminus g_2' & \text{falls } l \text{ ungerade} \\ g_2' \setminus g_1' & \text{falls } l \text{ gerade} \end{cases}$$

$\Rightarrow$

ALG behält für $\bar{J}_i$:

$$> 2N \cdot d_{\min} > 2D.$$

Falls $g_{i-1} = g_i$, dann behält ALG' für g_i nichts und die Behauptung folgt offensichtlich aus der Induktionsvoraussetzung.

Somit verbleibt noch der Fall

$$g_{i-1} \neq g_i = g_1' \cap g_2'.$$

Konstruktion $\Rightarrow$

$$x_{i-1}' = x_{i-1}.$$

$\Rightarrow$

$$\text{cost}_i' = \text{cost}_{i-1}' + d(x_{i-1}', x_i')$$

$$\leq \text{cost}_{i-1}' + D$$

$$\leq \text{cost}_{i-1} - \underbrace{d(x_{i-1}, x_{i-1}')}_0 + D$$

$$\leq \text{cost}_i - D$$

$$\leq \text{cost}_i - d(x_i, x_i')$$

4.2 Der Arbeitsfunktionalalgorithmus

Satz 4.1 zeigt, dass für ein beliebiges MTS stets ein auf Arbeitsfunktionen basierender optimaler Online-Algorithmus existiert. Jedoch ist der Beweis des Satzes 4.1 nicht konstruktiv, so dass sich direkt folgende Frage aufdrängt:

Frage:

Ist es möglich, für ein beliebiges MTS einen auf Arbeitsfunktionen basierenden optimalen Online-Algorithmus explizit anzugeben?

Idee:

Verfahre lokal am kostengünstigsten.

Folgendes Algorithmus implementiert obige Idee:

Algorithmus WFA

Online-Regel:

Annahme:

WFA ist im Zustand s , w ist die aktuelle Arbeitsfunktion und J die zu erledigende Aufgabe.

neuer Zustand:

$WFA(s, w, J) = x'$, wobei

i) $x' \in \text{Stützmenge von } w \cap J$ und

ii) $d(s, x') + w \cap J(x') = \min_{x \in S} \{d(s, x) + w \cap J(x)\}$

Falls mehrere solche Zustände existieren, dann kann unter diesen ein beliebiger Zustand gewählt werden.

Bemerkung:

Lemma 4.1 $\Rightarrow$

Für das gewählte x' gilt:

$$w \wedge \gamma(x') = w(x') + \gamma(x').$$

Frage:

Ist $w \wedge \gamma$ wohldefiniert? D.h., gibt es stets ein Zustand x' in der Stützmenge von $w \wedge \gamma$ mit $d(s, x') + w \wedge \gamma(x') = \min_{x \in S} \{d(s, x) + w \wedge \gamma(x)\}$?

Folgendes Lemma bejaht diese Frage.

Lemma 4.3

Für alle s, w, γ existiert in der Stützmenge von $w \wedge \gamma$ ein Zustand x' mit

$$d(s, x') + w \wedge \gamma(x') = \min_{x \in S} \{d(s, x) + w \wedge \gamma(x)\}.$$

Beweis:

Betrachte $y \in S$ mit

$$d(s, y) + w \wedge \gamma(y) = \min_{x \in S} \{d(s, x) + w \wedge \gamma(x)\}.$$

Wähle x' aus der Stützmenge von $w \wedge \bar{s}$ mit

$$w \wedge \bar{s}(y) = w \wedge \bar{s}(x') + d(x', y)$$

Dann gilt:

$$\begin{aligned} d(s, x') + w \wedge \bar{s}(x') &= d(s, x') - d(x', y) + w \wedge \bar{s}(y) \\ &= d(s, x') - \underbrace{d(y, x') - d(s, y)}_{\leq -d(s, x')} + d(s, y) + w \wedge \bar{s}(y) \\ &\leq d(s, y) + w \wedge \bar{s}(y). \end{aligned}$$

$\Rightarrow$

$$d(s, x') + w \wedge \bar{s}(x') = \min_{x \in S} \{ d(s, x) + w \wedge \bar{s}(x) \}.$$

Ziel: Analyse von WFA.

Seien

s_0

Startzustand

$\bar{s} = s_1, s_2, \dots, s_m$

Aufgabenfolge

$\bar{s} = s_1, s_2, \dots, s_m$

Zustände, in denen WFA die Aufgaben erledigt.

$$w_0 = \chi_{s_0}$$

$$w_i = w_0 \wedge s_1, s_2, \dots, s_i.$$

Dann gilt:

$$(*) \text{ cost}_{\text{WFA}}(s_0, \bar{s}) = \sum_{i=1}^m d(s_{i-1}, s_i) + \bar{s}_i(s_i).$$

Ferner gilt:

$$\begin{aligned}
 w_i(s_i) + d(s_{i-1}, s_i) &= \min_{x \in S} \{w_i(x) + d(s_{i-1}, x)\} \\
 &\leq w_i(s_{i-1}) + \underbrace{d(s_{i-1}, s_{i-1})}_0 \\
 &= w_i(s_{i-1})
 \end{aligned}$$

$$\Leftrightarrow d(s_{i-1}, s_i) \leq w_i(s_{i-1}) - w_i(s_i).$$

In (*) eingesetzt erhalten wir:

$$(**) \text{ cost}_{\text{WFA}}(s_0, \bar{s}) \leq \sum_{i=1}^m w_i(s_{i-1}) - w_i(s_i) + J_i(s_i)$$

Korollar 4.1 $\Rightarrow$

$$w_i(s_i) = w_{i-1}(s_i) + J_i(s_i)$$

$$\Leftrightarrow -w_{i-1}(s_i) = -w_i(s_i) + J_i(s_i)$$

In (**) eingesetzt erhalten wir:

$$\begin{aligned}
 \text{cost}_{\text{WFA}}(s_0, \bar{s}) &\leq \sum_{i=1}^m w_i(s_{i-1}) - w_{i-1}(s_i) \\
 &= \sum_{i=1}^m [w_i(s_{i-1}) - w_{i-1}(s_{i-1}) + w_i(s_i) - w_{i-1}(s_i)] \\
 &\quad + w_0(s_0) - w_m(s_m).
 \end{aligned}$$

Seien

$$\nabla(w, \bar{J}) := \max_{x \in S} \{w \wedge \bar{J}(x) - w(x)\}$$

und für $\bar{J} = J_1 J_2 \dots J_m$

$$\nabla(w, \bar{J}) := \sum_{i=1}^m \max_{x \in S} \{w \wedge J_1 J_2 \dots J_i(x) - w \wedge J_1 \dots J_{i-1}(x)\}$$

Dann gilt

$$\text{Cost}_{\text{WFA}}(s_0, \bar{J})$$

$$\leq 2 \nabla(\chi_{s_0}, \bar{J}) - \text{opt}(s_0, \bar{J}).$$

Also haben wir folgendes Lemma bewiesen:

Lemma 4.4

Seien s_0 der Startzustand eines MTS und $\bar{J}$ eine Aufgabenfolge. Dann gilt:

$$\text{Cost}_{\text{WFA}}(s_0, \bar{J}) \leq 2 \nabla(\chi_{s_0}, \bar{J}) - \text{opt}(s_0, \bar{J}).$$

Ziel:

Herleitung einer oberen Schranke für $\nabla(\chi_{s_0}, \bar{J})$.

Betrachte hierzu folgende Potentialfunktion:

$$\phi(w) = - \sum_x w(x).$$

$\Rightarrow$

$$-ND \leq \phi(\chi_{s_0}) \leq 0 \quad \text{und}$$

$$\phi(w) \geq -N(\min\{w(x) \mid x \in S\} + D),$$

wobei N die Anzahl der Zustände in S ist.

Ferner gilt:

$$\nabla(w, \bar{S}) + \phi(w \wedge \bar{S}) - \phi(w)$$

$$= \max_{x \in S} \{w \wedge \bar{S}(x) - w(x)\} - \sum_{y \in S} [w \wedge \bar{S}(y) - w(y)]$$

$$\leq 0.$$

Also gilt:

$$\nabla(\chi_{s_0}, \bar{S}) + \phi(\chi_{s_0} \wedge \bar{S}) - \phi(\chi_{s_0}) \leq 0$$

$$\Leftrightarrow \nabla(\chi_{s_0}, \bar{S}) \leq \underbrace{\phi(\chi_{s_0})}_{\leq 0} - \underbrace{\phi(\chi_{s_0} \wedge \bar{S})}_{\geq -N(\min_{x \in S} \{\chi_{s_0} \wedge \bar{S}(x)\} + D)}$$

$$\leq N(\min_{x \in S} \{\chi_{s_0} \wedge \bar{S}(x)\} + D)$$

$$= N \cdot \min_{x \in S} \{\chi_{s_0} \wedge \bar{S}(x)\} + N \cdot D$$

$$= N \cdot \text{opt}(s_0, \bar{S}) + ND$$

↑ Konstante

Zusammen mit Lemma 4.4 haben wir $s_0 =$ mit folgenden Satz bewiesen:

Satz 4.3

Für jedes MTS mit N Zuständen ist WFA $(2N-1)$ -competitive.

Übung:

Zeigen Sie, dass WFA für jedes MTS mit N Zuständen $(N-1)$ -competitive ist.

Frage:

Wie gut ist obige obere Schranke?

Ziel:

Beweis einer unteren Schranke.

Nehmen sei ALG ein deterministischer Online-Algorithmus für ein MTS mit metrischen Raum $M = (S, d)$. Sei $\varepsilon > 0$ eine positive reelle Zahl.

Dann bezeichnet ADV_ε denjenigen Gegenspieler, der folgende Aufgabenfolge $\bar{J} = J_1 J_2 \dots$ konstruiert:

$$\forall s \in S$$

$$J_i(s) = \begin{cases} \varepsilon & \text{falls } \text{ALG}(J_1 J_2 \dots J_{i-1}) = s \\ 0 & \text{sonst} \end{cases}$$

D.h., ALG bezahlt zur Erledigung einer Aufgabe ε , falls ALG im vorangegangenen Zu-

Stand bleibt und 0 sonst.

Satz 4.4

Sei ALG ein beliebiger deterministischer Online-Algorithmus für ein MTS mit metrischen Raum $M = (S, d)$, wobei $|S| = N$. Sei $\varepsilon > 0$ eine positive reelle Zahl. Sei $\bar{J}_m = J_1 J_2 \dots J_m$ die Aufgabenfolge der Länge m , die ADV_ε konstruiert. Dann gilt

$$\limsup_{m \rightarrow \infty} \frac{ALG(\bar{J}_m)}{OPT(\bar{J}_m)} \geq \frac{2N-1}{1+2\varepsilon}.$$

Beweis:

Annahme:

ALG ändert seinen Zustand k -mal ($k \leq m$).

Seien $s_0, s_1, \dots, s_k$ die Zustände von ALG in dieser Ordnung. Dann gilt:

Kosten von ALG bzgl. $\bar{J}_m$:

- $(m-k) \cdot \varepsilon$ plus
- $\sum_{i=1}^k d(s_{i-1}, s_i)$

Ziel:

Herleitung einer oberen Schranke für $OPT(\bar{J}_m)$.

Idee:

- Betrachte eine endliche Menge B von Offline-Algorithmus.

• Berechne $\sum_{\sigma \in B} \text{cost}(\sigma(\bar{J}_m)) = B(\bar{J}_m)$

$\Rightarrow \text{OPT}(\bar{J}_m) \leq \frac{B(\bar{J}_m)}{|B|}$

Durchführung:

Seien

q der Zustand von ALG während $[i, i+1)$

$q_1, q_2, \dots, q_{N-1}$ andere Zustände in S .

B enthält genau $2N-1$ Offline-Algorithmen, welche für $i=1, 2, \dots, m-1$ folgende Invariante erfüllen:

Invariante:

Während $[i, i+1)$ sind exakt zwei Offline-Algorithmen in Zustand q_j für $j=1, 2, \dots, N-1$. Genau ein Algorithmus ist in Zustand q .

$\Rightarrow$

B enthält $2N-1$ Algorithmen.

Realisierung:

ALG ändert seinen Zustand von q nach q'

$\Rightarrow$

Genau einer der beiden Algorithmen in Zustand q' ändert seinen Zustand nach q .

Berechnung von $B(\bar{J}_m)$.

- a) Jedesmal, wenn ALG seinen Zustand ändert, führt exakt ein Offline-Algorithmus den symmetrischen Übergang durch.
- b) Genau ein Offline-Algorithmus in $\mathcal{B}$ ist stets im selben Zustand wie ALG.

b) $\Rightarrow$

Gesamtkosten, wenn ALG ε bezahlt $(m-k)\varepsilon$

Wenn ALG nicht ε bezahlt, dann bezahlen genau zwei Offline-Algorithmus ε . Also betragen die Gesamtkosten hierfür

$$2k \cdot \varepsilon$$

Insgesamt erhalten wir:

$$B(\bar{J}_m) = \underbrace{(m-k)\varepsilon + \sum_{i=1}^k d(s_{i-1}, s_i)}_{\text{ALG}(\bar{J}_m)} + 2k\varepsilon$$

$$\Rightarrow \text{OPT}(\bar{J}_m) \leq \frac{2k\varepsilon + \text{ALG}(\bar{J}_m)}{2N-1}$$

Wegen

$$\text{ALG}(\bar{J}_m) \geq \sum_{i=1}^k d(s_{i-1}, s_i)$$

$$\geq \frac{1}{2} \cdot d_{\min} \quad \text{skizieren, so dass } d_{\min} \geq 1.$$

$$\geq \frac{1}{2}$$

folgt

$$2k\varepsilon \leq 2\varepsilon \cdot \text{ALG}(\bar{J}_m).$$

$\Rightarrow$

$$\begin{aligned} (2N-1) \text{OPT}(\bar{J}_m) &\leq 2\varepsilon \text{ALG}(\bar{J}_m) + \text{ALG}(\bar{J}_m) \\ &= (1+2\varepsilon) \text{ALG}(\bar{J}_m) \end{aligned}$$

$\Rightarrow$

$$\limsup_{m \rightarrow \infty} \frac{\text{ALG}(\bar{J}_m)}{\text{OPT}(\bar{J}_m)} \geq \frac{2N-1}{1+2\varepsilon}.$$