ONLINE ALGORITHMEN I.

DER WETTBEWERBSFAKTOR, PAGING

Beispiel: min-Makespan-SCHEDULING

Eingabe: Aufgaben mit Laufzeiten

$t_1, t_2, t_3 \ldots, t_i, \ldots t_n$

und m, die Anzahl der (identischen) Maschinen

Aufgabe: Weise jede Aufgabe einer der Maschinen zu, so dass der Makespan, also die maximale Fertigstellungszeit minimal ist (bezüglich aller Aufgaben)

(Wenn I_j die Indexmenge von Aufgaben ist, die auf Maschine j ausgeführt werden, dann ist der

$$\text{Makespan} = \max_j \left(\sum_{i \in I_j} t_i \right)$$

max über alle Maschinen — Fertigstellungszeit der Maschine j)

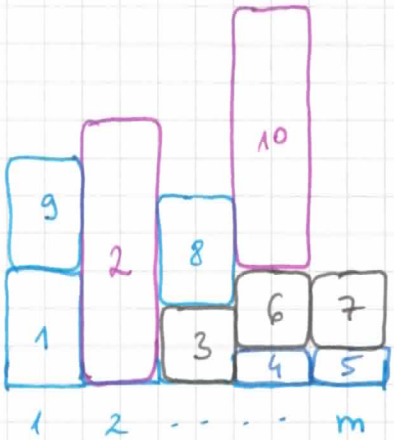
Der LIST-Scheduling Algorithmus:

→ Weise die Aufgaben in der Reihenfolge der Eingabe $t_1 t_2 t_3 \ldots t_i \ldots$ jeweils der/einer Maschine zu, die aktuell kleinste Last hat.

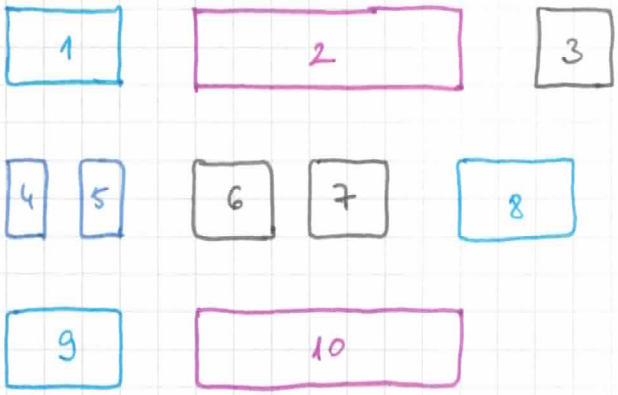
(Was ist hier der Unterschied zum Bin-Packing Problem?)

P2.

Zuteilung



Jobs in der Eingabe:



In der ONLINE Form des Scheduling Problems erhält der Algorithmus die Aufgaben einzeln, eine nach der anderen, und sobald eine neue Aufgabe i mit Laufzeit t_i angekommen ist, muss sie einer Maschine zugeteilt werden, ohne die zukünftigen Aufgaben, oder sogar die Anzahl n der Aufgaben zu kennen. LIST-Scheduling funktioniert auch als online Algorithmus: t_i wird zugeteilt unabhängig von t_{i+1}, t_{i+2}, usw.

((Es ist jedoch irrelevant, ob die Aufgaben „sofort" oder später „ausgeführt" werden. Die Berechnung von LIST vermischt sich nicht mit der Ausführung der Aufgaben in der Eingabe, oder mit dem Makespan.))

Online Algorithmen Allgemein:

Definition: Sei A ein Algorithmus, der für eine Eingabe $\sigma = (\sigma_1, \sigma_2 \dots \sigma_n)$ eine Ausgabe $\tau = (\tau_1, \tau_2 \dots \tau_n)$ berechnet.

→ A ist ein **offline Algorithmus**, falls die Ausgabe berechnet wird, nachdem die gesamte Eingabe betrachtet wurde.

→ A ist ein **online Algorithmus**, falls für jedes i nur der Teil der Eingabe $(\sigma_1, \sigma_2 \dots \sigma_i)$ während der Berechnung von τ_i bekannt ist.

Notation: $(\tau_1, \tau_2 \dots \tau_i) = A(\sigma_1, \sigma_2 \dots \sigma_i)$

manchmal auch: $\tau_i = A(\sigma_1, \sigma_2 \dots \sigma_i)$

(Im Scheduling Beispiel könnte $\sigma_i =$ "t_i" sein und $\tau_i =$ "j_i" die der t_i zugewiesene Maschine.)

Der Wettbewerbsfaktor (competitive ratio)

Sei $f(\sigma, \tau)$ irgendeine Zielfunktion deren Wert wir minimieren (oder maximieren) wollen.

(Im Beispiel ist $f(\sigma, \tau)$ der Makespan, aber es gibt viele andere mögliche Zielfunktionen; z.B: „maximiere die minimale Fertigstellungszeit der Maschinen." Allgemein, z.B. Speicherplatzbedarf, „Anzahl Strafpunkte", generell beliebige Kostenfunktion, bzw. Profitfunktion)

→ ein online Algorithmus hat den Nachteil, dass er die zukünftigen Eingaben nicht vorhersieht

→ Vergleichen wir:

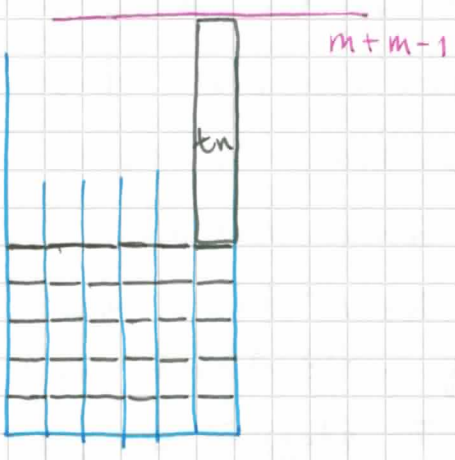
LIST (online)

OPT ein optimaler (offline) Algorithmus

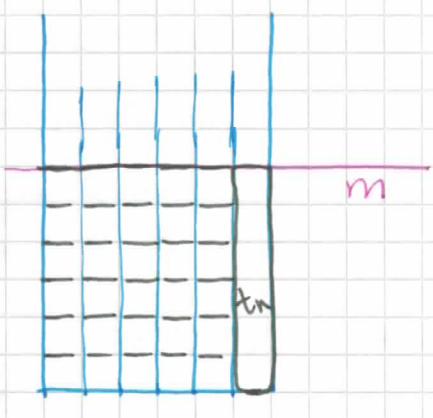
(OPT ist ein allwissender Algorithmus der die Zukunft sieht (ohne Laufzeitbeschränkung))

→ Um wieviel (welchen Faktor) schlechteren Makespan kann LIST ausgeben als ein optimaler offline Algorithmus?

Beispiel: Bestehe die Eingabe aus $m(m-1)$ Jobs der Größe $t_i = 1$, gefolgt von einem großen Job der Größe $t_k = m$ $\quad \sigma = \{\underbrace{1, 1, 1, 1 \ldots, 1}_{m(m-1) \text{ Stück}}, m\}$



$LIST_\sigma = m + m - 1$



$OPT_\sigma = m$

Für diese σ ist LIST PS.

→ fast 2-mal so schlecht wie OPT, weil

$$\frac{LIST_\sigma}{OPT_\sigma} = \frac{2m-1}{m} = 2 - \frac{1}{m}$$

Der Faktor $\frac{LIST_\sigma}{OPT_\sigma}$ ist beliebig nah an 2

(er kann kann für jede $\varepsilon > 0$ höher sein als $2-\varepsilon$.

d.h. für geeignete σ $\frac{LIST_\sigma}{OPT_\sigma} > 2-\varepsilon$)

→ Es kann aber auch gezeigt werden, dass

$$\frac{LIST_\sigma}{OPT_\sigma} < 2$$ für jede Eingabe σ gilt.

→ Man sagt: Der Algorithmus LIST hat Wettbewerbsfaktor 2

(er kann mindestens um Faktor $2(-\varepsilon)$ und auch höchstens um Faktor 2 schlechteren Zielwert erreichen als ein optimaler offline Algorithmus.)

Der Wettbewerbsfaktor eines online Algorithmus

→ Der Wettbewerbsfaktor eines online Algorithmus ALG zeigt, um welchen Faktor schlechter (die Ausgabe von) ALG sein kann als (die Ausgabe von) ein optimaler offline Algorithmus.

Wir hätten gerne Algorithmen mit möglichst kleinem Wettbewerbsfaktor.

- Sei eine Zielfunktion $f(\sigma, \tau)$ festgelegt (z.B. der Makespan)
- Für fixierte Eingabe σ seien

ALG_σ: Zielwert (der Ausgabe) von ALG für σ

OPT_σ: Zielwert eines optimalen offline Algorithmus für σ

Definition: Ein online Algorithmus ALG hat den Wettbewerbsfaktor höchstens α, wenn es eine Konstante c gibt, so dass

$$ALG_\sigma \leq \alpha \cdot OPT_\sigma + c$$

für jede Eingabefolge σ gilt (bei Minimierungsproblem)

$$\left(ALG_\sigma \geq \frac{1}{\alpha} \cdot OPT_\sigma - c \quad \text{bei Maximierungsproblem} \right)$$

Bemerkung: Der Wettbewerbsfaktor ist ein ähnliches Konzept wie der Approximationsfaktor, und kann für konkrete Algorithmen, wie für LIST den gleichen Wert haben. Jedoch, bei Approximationsalgorithmen werden effiziente (d.h. polynomialzeit) Algorithmen mit einem optimalen Algorithmus verglichen, während bei online Algorithmen werden online Algorithmen mit einem optimalen (offline) Algorithmus verglichen, unabhängig von Laufzeit.
Der Unterschied wird bei den folgenden allgemeinen unteren Schranken deutlich:

Beachte, dass die folgenden Beispiele für beliebigen online Algorithmus für ein bestimmtes Problem eine untere Schranke beweisen.

Allgemeine untere Schranken für den Wettbewerbsfaktor

Beispiel 1: Kein online Algorithmus für min-Makespan-SCHEDULING kann Wettbewerbsfaktor $\frac{3}{2} - \varepsilon$ besitzen.

Warum?

Sei ALG ein online Algorithmus.
Sei $m = 2$, und $t_1 = t_2 = t$, d.h. die ersten beiden Jobs gleichgroß.

→ FALL 1: ALG weist t_1 und t_2 derselben Maschine zu.

Dann stoppen wir (als bösartiger Gegner) die Job-Folge, also setzen

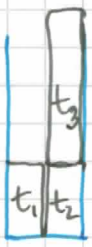
$$\sigma = (t_1, t_2)$$

für diese Eingabe hat ALG offensichtlich Wettbewerbsfaktor 2

→ FALL 2: ALG weist t_1 und t_2 unterschiedlichen Maschinen zu

Dann sei noch $t_3 = 2t$

und $\sigma = (t_1, t_2, t_3) = (t, t, 2t)$



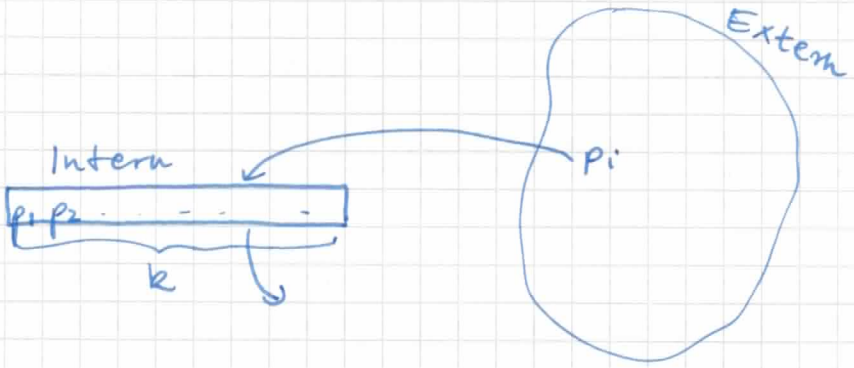
für diese Eingabe hat ALG Wettbewerbsfaktor $\frac{3}{2}$ weil

$$ALG_\sigma = 3t \text{ und } OPT_\sigma = 2t$$

(Die $+c$ in der Definition des Wettbewerbsfaktors hilft hier nicht, weil $3t > \left(\frac{3}{2} - \varepsilon\right) \cdot 2t + c$ für t groß genug.)

Das Paging Problem

ein abstraktes Modell für den folgenden Problem-Typ:



→ es gibt einen schnellen internen Speicher (RAM), (Cache) wo genau k Seiten gespeichert werden können

→ es gibt einen langsamen externen Speicher (Disk, Internet) wo alle Seiten gespeichert sind

→ die <u>Eingabe</u> ist eine Folge von Seiten

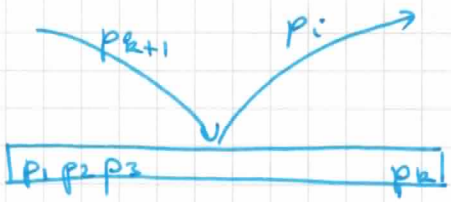
$\sigma = \sigma_1 \sigma_2 \sigma_3 \ldots\ldots$ die online angefordert werden

→ die ersten k verschiedenen angeforderten Seiten werden intern gespeichert; wenn eine Seite angefordert wird, die sich <u>nicht</u> im internen Speicher befindet ('page-fault') und der Cache voll ist, muss der Paging-Algorithmus eine andere Seite auslagern um der neuen Seite Platz zu machen. Welche Seite soll ausgelagert werden?

→ Für jede angeforderte aber nicht vorhandene Seite erhält der Paging-Algorithmus einen Strafpunkt (page-fault)

(Der Einfachheit halber werden in den nachfolgenden Analysen die ersten k page-faults nicht mitgezählt.)
bis der Cache gefüllt wird

P 8.



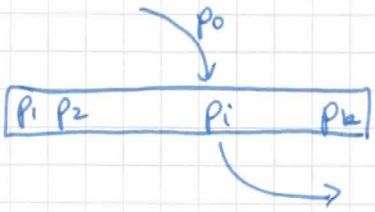
Der online Paging-Algorithmus soll die Anzahl der erhaltenen Strafpunkte minimieren.

I. Deterministische Paging-Strategien

Offensichtlich hat ein online Algorithmus keine guten Chancen: eine bösartige Eingabe kann z.B. immer gerade diejenige Seite anfordern, die eben ausgelagert wurde. (Es gibt trotzdem bessere und schlechtere Algorithmen, was den Wettbewerbsfaktor betrifft: schwache Algorithmen erhalten sogar für einfache Eingaben (mit OPT = 1) in jedem Schritt einen Strafpunkt. Bei besseren Algorithmen ist die bösartige Eingabe selbst für OPT schwierig.)

1.) Wie wäre eine bestmögliche OFFLINE Strategie? ein all-wissender Algorithmus?

Definition: Die LFD (Longest-Forward-Distance) Strategie lagert immer die Seite aus, die am spätesten in der Zukunft neu angefordert wird, (unter allen aktuell gespeicherten Seiten).



Eingabe: p_0 .... p_k ... p_3 ... p_{i-1} .. p_2 .. p_1 .. p_{k-1} .. p_1 .... p_i

↑ aktuelle Anfrage

jede andere mindestens 1mal

Theorem: LFD ist ein optimaler offline Algorithmus.
(ohne Beweis)

Für die Feststellung des Wettbewerbsfaktors werden wir die Online Strategien mit LFD vergleichen.

ONLINE STRATEGIEN

2.) Def: LFU (Least-Frequently-Used): Lagere die Seite aus, die seit ihrer letzten Einlagerung am seltensten angefordert wurde.

Behauptung: Die LFU-Strategie hat Wettbewerbsfaktor ∞.

Beweis: Betrachte die Eingabefolge:

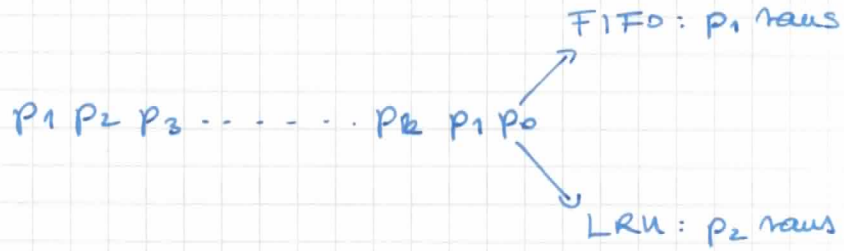
$p_1\ p_2\ p_3 \ldots p_k \mid p_2\ p_3 \ldots p_k \mid p_0 \mid p_1 \mid p_0 \mid p_1 \mid p_0 \mid p_1 \mid p_0 \mid p_1$

Bei der Anfrage p_0

- der optimale LFD lagert z.B. p_2 aus, und erhält keine weiteren Strafpunkte → nur der erste p_0 verursacht page-fault
- LFU lagert p_1 und p_0 abwechselnd aus
(die waren ja seit ihrer Einlagerung nicht angefordert, während $p_2 \ldots p_k$ wurden noch einmal angefordert → jede $p_0\ p_1\ p_0\ p_1\ p_0\ p_1$ verursacht page-fault

3.) Def: FIFO (First-In-First-Out): Lagere die Seite aus, die am längsten gespeichert wurde

4.) Def: LRU (Least-Recently-Used): Lagere die Seite aus, deren letzte Anforderung am weitesten zurückliegt.



Theorem: FIFO und LRU haben Wettbewerbsfaktor k.

Beweis: Im Folgenden bedeutet ○ einen Strafpunkt

Behauptung 1: In einer beliebigen Teilfolge σ' mit mindestens $k+1$ verschiedenen Seiten hat jeder Algorithmus (auch ein offline Algorithmus oder der optimale) mindestens einen Strafpunkt.

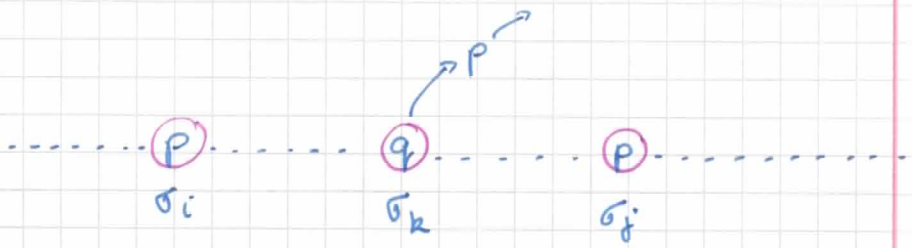
Warum?

Am Anfang der Teilfolge σ' sind gerade maximal k Seiten im Cache. Es muss also mindestens eine Seite in σ' geben, die am Anfang nicht im Cache ist. Spätestens als diese Seite angefordert wird, gibt es ein Page-Fault.

Behauptung 2: In FIFO und LRU gilt:

Zwischen zwei Anforderungen an die selbe Seite p, die beide einen Strafpunkt verursachen, gibt es mindestens k verschiedene Seiten in der Eingabefolge. Mit p zusammen gibt es $\geq k+1$ verschiedene Seiten.

Warum?



- die Seite p erhält beim zweiten Mal einen Strafpunkt, also sie wurde zwischen σ_i und σ_j wieder ausgelagert
- Bezeichne q die Seite bei deren Anforderung $\sigma_k = q$, die Seite p ausgelagert wurde
- da p von LRU ausgelagert wurde, erfolgte eine Anfrage seit der letzten Anfrage (σ_i oder später) an die Seite p, für alle anderen $k-1$ Seiten (die im Speicher sind zur Zeit der Anfrage σ_k). Diese sind also $k-1$ Anfragen, plus die Anfrage an q sind (seien) die k Anfragen nach verschiedenen Seiten echt zwischen σ_i und σ_j.
- für FIFO gilt dasselbe Argument.

Sei jetzt $\sigma = \sigma_1 \sigma_2 \sigma_3 \ldots$ eine Worst-Case Eingabefolge für LRU (bzw. für FIFO).

Wir teilen diese Eingabefolge in Teilfolgen

$$\sigma^0 \sigma^1 \sigma^2 \ldots \sigma^j \ldots$$ auf wie folgt:

- σ^0 endet mit dem ersten Strafpunkt für LRU (und auch für OPT); d.h. mit der $(k+1)$-ten neuen Seite
- σ^1 endet mit dem $(k+1)$-ten Strafpunkt für LRU
- ⋮
- σ^j endet mit dem $1+j \cdot k$-ten Strafpunkt für LRU
- ⋮

Also, LRU erhält in jeder Teilfolge genau k Strafpunkte.

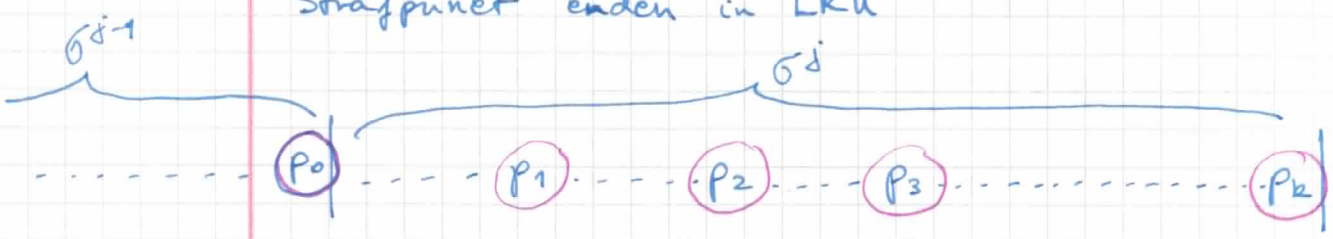
Wir zeigen, dass Opt in jeder Teilfolge mindestens 1 Strafpunkt erhält. Damit beweisen wir für die Anzahl aller Strafpunkte von Opt bzw. LRU das Verhältnis

$$LRU_\sigma \leq k \cdot OPT_\sigma$$

Da σ eine Worst-Case (oder, beliebige) Eingabe ist, beweist das den Wettbewerbsfaktor höchstens k für LRU.

Sei σ^j eine Teilfolge, und seien

- p_0 die letzte Anfrage in σ^{j-1}
- p_1 p_2 ... p_k die k Anfragen, die in σ^j mit einem Strafpunkt enden in LRU



FALL 1: es gibt gleiche Seiten unter $p_0 p_1 \dots p_k$

Dann, laut Behauptung 2 gibt es mindestens $k+1$ verschiedene Seiten in σ^i, und laut Behauptung 1 erhält sogar Opt mindestens 1 Strafpunkt in σ^i.

FALL 2: $p_0 p_1 p_2 \dots p_k$ sind alle unterschiedlich.

Im Opt, p_0 ist im Cache als σ^i beginnt (p_0 war eben die letzte Anfrage).
Da es weitere $\geq k$ unterschiedliche Seiten folgen in σ^i, mindestens eine von ihnen ist nicht im Cache von Opt am Anfang von σ^i, und verursacht somit einen Strafpunkt.
(gleiches Argument wie bei Behauptung 1.)

(Für FIFO geht der Beweis analog.)

Theorem: Jeder deterministische ONLINE Paging-Algorithmus hat Wettbewerbsfaktor mindestens k.

Somit sind LRU und FIFO optimale Paging-Strategien, was den Wettbewerbsfaktor angeht.

Beweis: Sei A ein deterministischer online Paging Algorithmus fixiert. Wir definieren die (früher schon erwähnte) bösartige Eingabe σ mit $k+1$ verschiedenen Seiten: $p_0 p_1, p_2, \dots p_k$
Wir vergleichen das Verhalten von A und von der optimalen LFD Strategie für diese Eingabe.

Die worst-case Eingabe σ fängt so an:

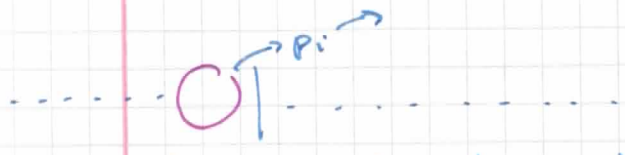
$p_1 p_2 p_3 \dots p_k (p_0)$

- auf p_0 erhalten A und LFD den ersten Strafpunkt. Ab hier wird σ so definiert, dass A bei jeder Anfrage einen Strafpunkt (I.) erhält (immer die Seite wird gefragt, die eben ausgelagert wurde).

  (Beachte, dass A ein deterministischer Algorithmus ist, es ist also in jedem Schritt i der rekursiven Definition eindeutig bestimmt, dass σ_{i+1} die Seite sei, die nach der Folge $\sigma_1, \sigma_2 \dots \sigma_i$ von A gerade ausgelagert ist. Die worst-case Eingabe σ entsteht also nicht „online".)

- LFD hingegen erhält höchstens einen Strafpunkt pro k Anfragen. (II.) (gilt nur für $k+1$ verschiedene Seiten!)

(Warum? Angenommen, dass bei einem Strafpunkt LFD lagert die Seite p_i aus.



p_i wird ausgelagert von LFD, weil die anderen $k-1$ Seiten im Speicher alle früher als p_i gefragt werden in der Zukunft. Solange p_i nicht wieder angefordert wird, frühestens die k-te Anfrage von jetzt, erhält LFD keinen Strafpunkt.) Von (I.) und (II.) erhalten wir

$$LFD_\sigma \geq \frac{A_\sigma}{k}$$

□

II. Eine randomisierte online Paging-Strategie

5.) Die Marking-Strategie

Idee:
- die (neu) angeforderten Seiten werden sofort markiert
- immer nur unmarkierte Seiten werden ausgelagert; aus diesen wird immer zufällig gewählt
- wenn alle Seiten im Speicher markiert sind, werden die Markierungen gelöscht

Bemerkung:
- die unmarkierten Seiten waren „lange" nicht mehr angefordert (im Vergleich zu den Markierten), ähnlich zum Least Recently Used, deshalb wird eine aus gerade diesen Seiten ausgelagert;
- die zufällige Wahl einer solchen Seite macht es schwieriger, Worst-Case Eingaben zu konstruieren: jede Eingabe σ verursacht wenige Strafpunkte für die meisten zufälligen Folgen von auszulagernden Seiten

Beispiel:

$p_1^* \; p_2^* \; p_2 \; p_3^* \; p_1 \; p_3 \; | \; p_4^* \; p_2^* \; p_4 \; p_2 \; p_3^* \; p_4 \; p_4 \; p_2 \; p_3 \; p_2 \; | \; p_5^*$

Markierung löschen — Markierung löschen

$p_1^* \; p_2^* \; p_3^*$

$p_1 \; p_2 \; p_4^*$

$p_1 \; p_2^* \; p_4^*$

$p_3^* \; p_2^* \; p_4^*$

Marking - Algorithmus:

→ die Seite p wird angefordert

→ falls p nicht gespeichert ist, wähle zufällig gleichverteilt eine nicht-markierte Seite im Speicher, und lagere sie aus
- falls es keine nicht-markierte Seite gibt, lösche alle Markierungen, und lagere eine zufällige Seite aus;

→ markiere die Seite p

Die Analyse von Marking

Zur Erinnerung: die k-te Harmonische Zahl ist

$$H_k = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots\ldots + \frac{1}{k} = \sum_{i=1}^{k} \frac{1}{i}$$

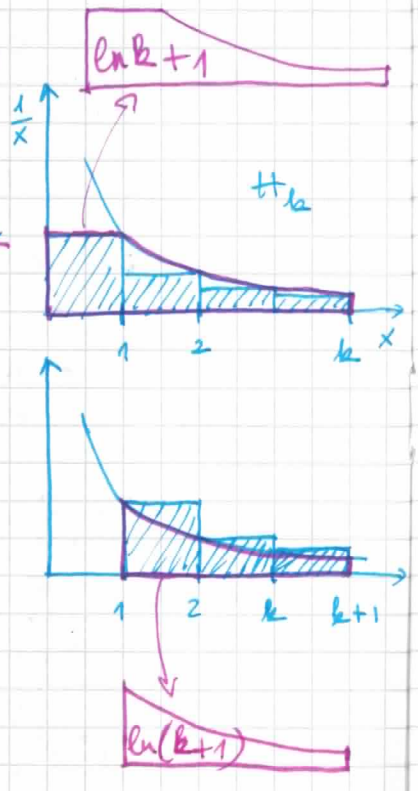
asymptotisch gilt: $\boxed{H_k \sim \ln k}$

weil

$$H_k = \sum_{i=1}^{k} \frac{1}{i} \leq 1 + \int_1^k \frac{1}{x}\,dx = 1 + \left[\ln x\right]_1^k = \underline{1 + \ln k}$$

und

$$H_k = \sum_{i=1}^{k} \frac{1}{i} \geq \int_1^{k+1} \frac{1}{x}\,dx = \left[\ln x\right]_1^{k+1} = \ln(k+1)$$



also

$$\boxed{\ln(k+1) \leq H_k \leq \ln k + 1}$$

Theorem: Die Marking-Strategie hat höchstens den Wettbewerbsfaktor $2H_k \leq 2 \ln k + 2$ wobei k die Anzahl der Seiten im Cache bezeichnet.

Bemerkung: Beim Wettbewerbsfaktor eines randomisierten Algorithmus wie Marking, rechnet man mit dem Erwartungswert der Zielfunktion $f(\sigma, T)$; in diesem Fall mit der erwarteten Anzahl von Strafpunkten.

! Der Wettbewerbsfaktor mindestens k für deterministische Strategien konnte durch Randomisierung deutlich verbessert werden. !

Beweis:
Im verbleibenden Teil des Abschnitts über Paging, beweisen wir das Theorem in drei Teilen:

1. Vorbereitung: Zerlegung der Eingabe in Teilfolgen
2. die Anzahl der Strafpunkte von Opt
3. die erwartete Anzahl der Strafpunkte von Marking

1. Vorbereitung:
Sei $\sigma = \sigma_1 \sigma_2 \sigma_3 \ldots$ eine beliebige fixierte Eingabe; wir zerlegen σ (iterativ) in Teilfolgen $\sigma^0 \sigma^1 \sigma^2 \ldots \sigma^i \ldots \sigma^m$ so dass:

- σ^0 ist längstmöglich wo genau k verschiedene Seiten angefragt werden;
- $\vdots$
- σ^i ist ein längstes, auf σ^{i-1} folgendes Eingabeintervall, in dem genau k verschiedene Seiten angefragt werden.

Beispiel:

$\underbrace{p_1\ p_2\ p_2\ p_3\ p_1\ p_3}_{\sigma^0}\ \underbrace{p_4\ p_2\ p_4\ p_2\ p_3\ p_4\ p_4\ p_2\ p_3\ p_2}_{\sigma^1}\ \underbrace{p_5 \ldots}_{}$

P18.

σ^0 σ^1 $S_1 = \{p_1 p_2 p_3\}$ $S_2 = \{p_2 p_3 p_4\}$

Beispiel: $p_1 p_2 p_2 p_3 p_1 p_3 \mid p_4 p_2 p_4 p_2 p_3 p_4 p_1 p_2 p_3 p_2 \mid p_5 \ldots$

Löschen Löschen

Sei S_i die Menge der in σ^{i-1} gefragten Seiten.

Beachte: Die Definition der Zerlegung $\sigma^0 \sigma^1 \ldots \sigma^i \ldots$ hängt nur von der Eingabefolge σ ab; das gleiche gilt für die Menge S_i. Sie wurden unabhängig von der Paging-Strategie definiert.

Im Folgenden betrachten wir das Vorgehen von Marking auf Eingabe σ:

→ $|S_i| = k$ S_i sind die Seiten im Speicher bevor σ^i beginnt

→ σ^i (für $i = 1, 2, 3, \ldots$) beginnt mit einem Strafpunkt und mit dem Löschen aller Markierungen; danach werden die k verschiedenen Seiten (wenn nötig eingeladen und) markiert; am Ende von σ^i sind alle Seiten im Speicher markiert; dies ist die Seitenmenge S_{i+1}

→ die k verschiedenen Seiten von σ^i werden nicht alle einen Strafpunkt verursachen! Sie könnten ja schon im Speicher liegen bei ihrer ersten Anforderung in σ^i

→ jede der k Seiten von σ^i (von S_{i+1}) verursacht höchstens 1 Strafpunkt während σ^i (danach bleibt sie im Speicher während σ^i weil sie ja markiert wurde)

Definition: Für eine gegebene Teilfolge σ^i nennen wir eine Anforderung **neu** wenn die Seite nicht in S_i liegt und **alt** wenn die Seite in S_i liegt.

n_i sei die Anzahl der neuen Anforderungen während σ^i

Beachte, dass n_i auch unabhängig vom Paging-Algorithmus definiert wurde!

Beispiel: In unserer Beispiel-Eingabe in der Teilfolge σ^1

p_4 ist neue Seite, p_2 p_3 sind alte Seiten, weil $p_2, p_3 \in S_1$

$n_1 = 1$

Welche von den neuen oder alten Seiten in σ^i verursachen einen Strafpunkt?

→ jede höchstens einen während σ^i

→ die neuen Seiten mit Sicherheit, weil sie am Anfang von σ^i nicht im Speicher sind wie p_4 in σ^1

→ die alten Seiten entweder ja, oder nein:

NEIN, falls sie im Speicher geblieben sind bis ihrer Anfrage wie p_2 in σ^1

JA, wenn sie wegen einer anderen Seite zwischendurch ausgelagert wurden wie p_3 in σ^1

Was ist eine ungünstigste Positionierung der alten und neuen Seiten, damit sie die meisten Strafpunkte verursachen in Erwartung?

(2.) Wir schätzen die Anzahl der Strafpunkte von Opt nach unten ab:

Behauptung 1: Jeder (offline oder online) Algorithmus erhält insgesamt mindestens

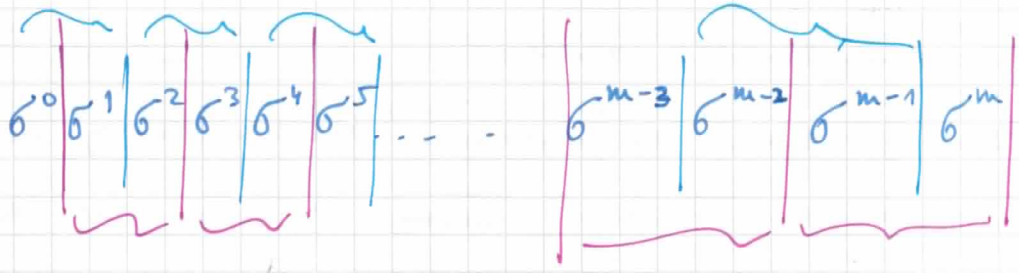
$$\frac{1}{2}\sum_{i=1}^{m'} n_i \quad \text{Strafpunkte.}$$

nach der Abarbeitung von $\sigma^0\sigma^1\sigma^2 \dots \sigma^m$.

Beweis: wir beweisen Behauptung 1 für gerade m.

Beobachtung: Sei σ' eine Teilfolge von Anforderungen. Falls σ' $k+n'$ verschiedene Seiten enthält, dann erhält jeder Algorithmus mindestens n' Strafpunkte während σ'.

(Warum? Wenn die Anforderung einen dieser Seiten keinen Strafpunkt verursacht, dann war die Seite am Anfang von σ' im Speicher. Solche Seiten gibt es aber höchstens k von den $k+n'$ verschiedenen Seiten; mindestens n' waren nicht im Speicher, und jede von dieser erhält mindestens 1 Strafpunkt.)



Die Teilfolgen $\sigma^0\sigma^1$
$\sigma^2\sigma^3$
$\sigma^4\sigma^5$
$\vdots$
$\sigma^{m-2}\sigma^{m-1}$

sind Teilfolgen mit jeweils $k+n_1$, $k+n_3$, ... $k+n_{m-1}$ verschiedenen Seiten, und jeder Algorithmus erhält insgesamt mindestens $n_1+n_3+n_5+\ldots+n_{m-1}$ Strafpunkte. Analog aufgeteilt auf die Teilfolgen $\sigma^1\sigma^2$, $\sigma^3\sigma^4 \ldots \sigma^{m-1}\sigma^m$, erhält auch jeder Algorithmus mindestens

$n_2+n_4+n_6+\ldots+n_m$ Strafpunkte.

Schließlich, ist mindestens eine dieser beiden Summen

$$\geq \frac{1}{2}(n_1+n_2+n_3+\ldots+n_m)$$

und somit haben wir Behauptung 1 bewiesen. □

(3.) Wir schätzen die erwartete Anzahl der Strafpunkte von Marking nach oben ab:

Behauptung 2: Die erwartete Strafpunktzahl von Marking während der Abarbeitung von σ^i ist höchstens

$$n_i \cdot \sum_{j=1}^{k} \frac{1}{j} = n_i H_k$$

Korollar 1. Die erwartete Strafpunktzahl von Marking ist insgesamt höchstens $H_k \cdot \sum_{i=1}^{m} n_i$

Korollar 2. Marking hat den Wettbewerbsfaktor höchstens

$$\frac{H_k \cdot \sum_{i=1}^{m} n_i}{\frac{1}{2}\sum_{i=1}^{m} n_i} = 2 \cdot H_k$$

(laut Behauptungen 1 und 2)

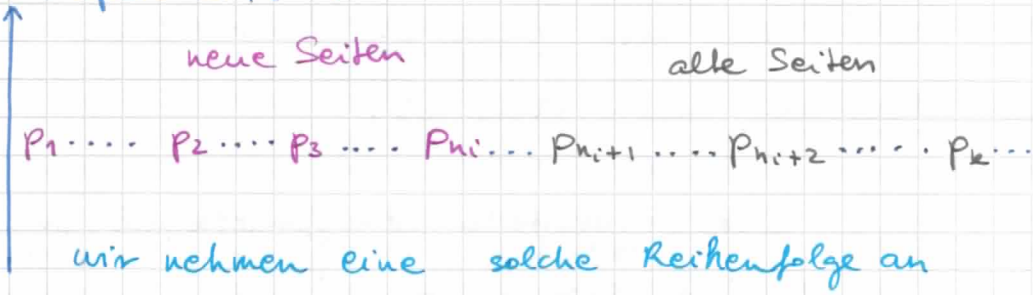
□

Beweis der Behauptung 2:

Sei die Teilfolge σ^i fixiert.
Die neuen Seiten erhalten mit Sicherheit je einen Strafpunkt.
Bei welcher Reihenfolge der alten bzw. neuen Seiten innerhalb von σ^i erhält eine alte Seite mit höchster Wahrscheinlichkeit einen Strafpunkt?

(ZB. wenn eine alte Seite als zweite in σ^i gefragt wird, erhält sie mit Prob. $\frac{k-1}{k}$ keinen Strafpunkt weil sie am Anfang im Cache liegt. Wenn aber diese alte Seite erst nach den n_i neuen Seiten angefragt wird, dann erhält sie nur mit Prob $\frac{k-n_i}{k}$ keinen Strafpunkt. (Warum?))

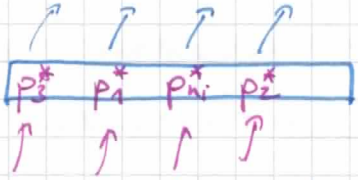
→ Die erwartete Strafpunktzahl ist maximal, wenn zuerst die n_i neuen Seiten angefragt werden.
S_i im Speicher, $|S_i| = k$



wir nehmen eine solche Reihenfolge an

→ wegen der n_i neuen Seiten erhält Marking n_i Strafpunkte

→ außerdem werden n_i zufällig gewählten Seiten aus S_i ausgelagert



* Markierung

→ Die Wahrscheinlichkeit, dass die erste alte Seite p_{n_i+1} ausgelagert wurde (und deshalb einen Strafpunkt erhält)?

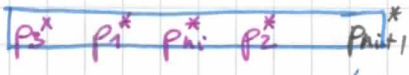
$\frac{n_i}{k}$ (wie oben schon gesagt)

($\frac{n_i}{k}$ ist die Wahrscheinlichkeit, dass p_{n_i+1} auf einer ungünstigen / zufällig ausgewählten / rosa Position war)

→ Weiter, betrachten wir die Wahrscheinlichkeit, dass die zweite alte Seite p_{n_i+2} ausgelagert wurde während der Anforderungen $p_1 .. p_2 .. p_3 .. p_{n_i}$ p_{n_i+1} und deshalb einen Strafpunkt erhält.

Es gibt 2 Möglichkeiten (2 Bedingungen, ein vollständiges Ereignissystem)

Fall 1. p_{n_i+1} blieb im Speicher, und wurde jetzt in σ' nur markiert



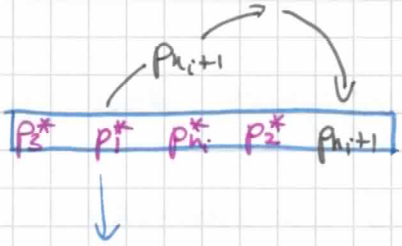
unmöglich, dass p_{n_i+2} hier war am Anfang von σ'

in diesem Fall gibt es $k-1$ mögliche Zellen für p_{n_i+2} im Cache, und n_i davon (die rosa) sind ungünstig. Wahrscheinlichkeit für Strafpunkt ist somit

$$\frac{n_i}{k-1}$$

Fall 2. p_{n_i+1} wurde während der neuen Seiten ausgelagert und dann erneut in den Speicher gebracht, in andere Zelle

P24.



unmöglich, dass p_{n_i+2} hier war am Anfang von σ' weil hier war p_{n_i+1}

in diesem Fall gibt es auch $k-1$ mögliche Zellen für p_{n_i+2} und n_i davon (n_i-1 rosa, und die neue Zelle von p_{n_i+1}) sind ungünstig.

Prob. für Strafpunkt für p_{n_i+2} ist somit auch im Fall 2

$$\frac{n_i}{k-1}$$

In beiden Fällen (unter beiden Bedingungen) ist die genaue Wahrscheinlichkeit, dass p_{n_i+2} einen Strafpunkt verursacht $\frac{n_i}{k-1}$

→ die erwartete Strafpunktzahl von p_{n_i+1}

$$= 0 \cdot \frac{k-n_i}{k} + 1 \cdot \frac{n_i}{k} = \frac{n_i}{k}$$

die erwartete Strafpunktzahl für p_{n_i+2}

$$= 0 \cdot \frac{k-1-n_i}{k-1} + 1 \cdot \frac{n_i}{k-1} = \frac{n_i}{k-1}$$

wenn wir weiter so argumentieren, erhalten wir die erwartete Strafpunktzahl für p_{n_i+j} (j-te alte Zelle)

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

(Erwartungswert von Indikatorvariablen)

$X_j = \begin{cases} 1 & \text{wenn } p_{n_i+j} \text{ Strafpunkt erzeugt} \\ 0 & \text{sonst} \end{cases}$

Die erwartete Strafpunktzahl für allen $k-n_i$ alten Seiten ist somit (Erwartungswert einer Summe von Indikatorvariablen)

$$\frac{n_i}{k} + \frac{n_i}{k-1} + \dots + \frac{n_i}{k-(k-n_i-1)} = n_i\left(\frac{1}{k} + \frac{1}{k-1} + \dots + \frac{1}{n_i+1}\right) \leq$$

$$\leq n_i\left(\frac{1}{k} + \frac{1}{k-1} + \dots + \frac{1}{2}\right) \quad \text{weil } n_i \geq 1$$

→ mit den n_i Strafpunkten für die neuen Seiten haben wir insgesamt in Erwartung höchstens

$$n_i + n_i\left(\frac{1}{k} + \frac{1}{k-1} + \dots + \frac{1}{2}\right) = n_i\left(\frac{1}{k} + \frac{1}{k-1} + \dots + \frac{1}{2} + 1\right) =$$

$$= n_i \cdot H_k$$

Strafpunkte in dem Fall wenn alle neuen Seiten vor allen alten Seiten kommen in σ! Diese obere Schranke muss also auch in den günstigeren Eingaben gelten, wenn neue und alte Seiten anders gemischt werden (ihre Reihenfolge).