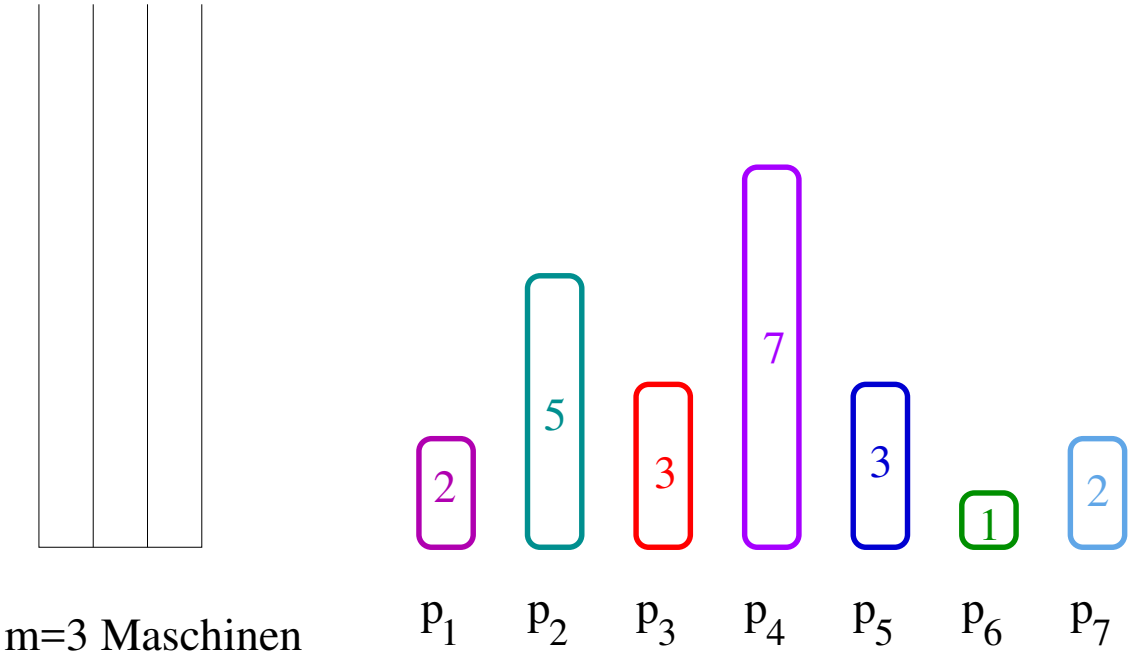
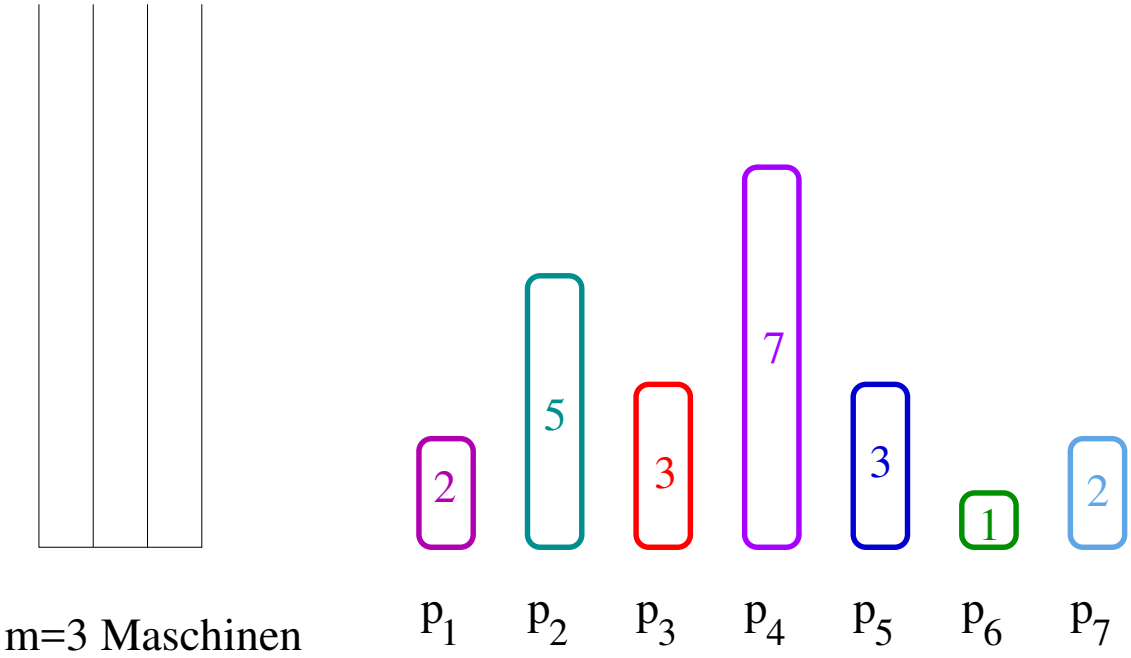
Makespan Scheduling auf identischen Maschinen



Eingabe: n Jobs mit Laufzeiten $p_1, p_2, \ldots, p_n > 0$ und m Maschinen

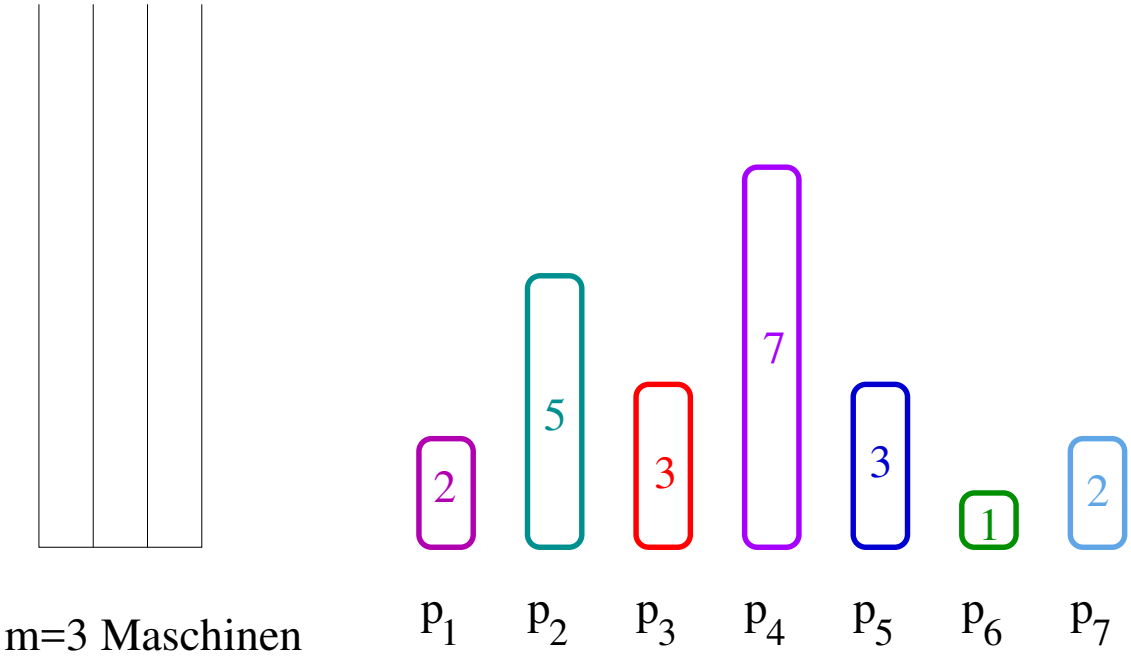
Makespan Scheduling auf identischen Maschinen



Eingabe: n Jobs mit Laufzeiten $p_1, p_2, \ldots, p_n > 0$ und m Maschinen

Ausgabe: Weise jeden Job genau einer Maschine zu

Makespan Scheduling auf identischen Maschinen

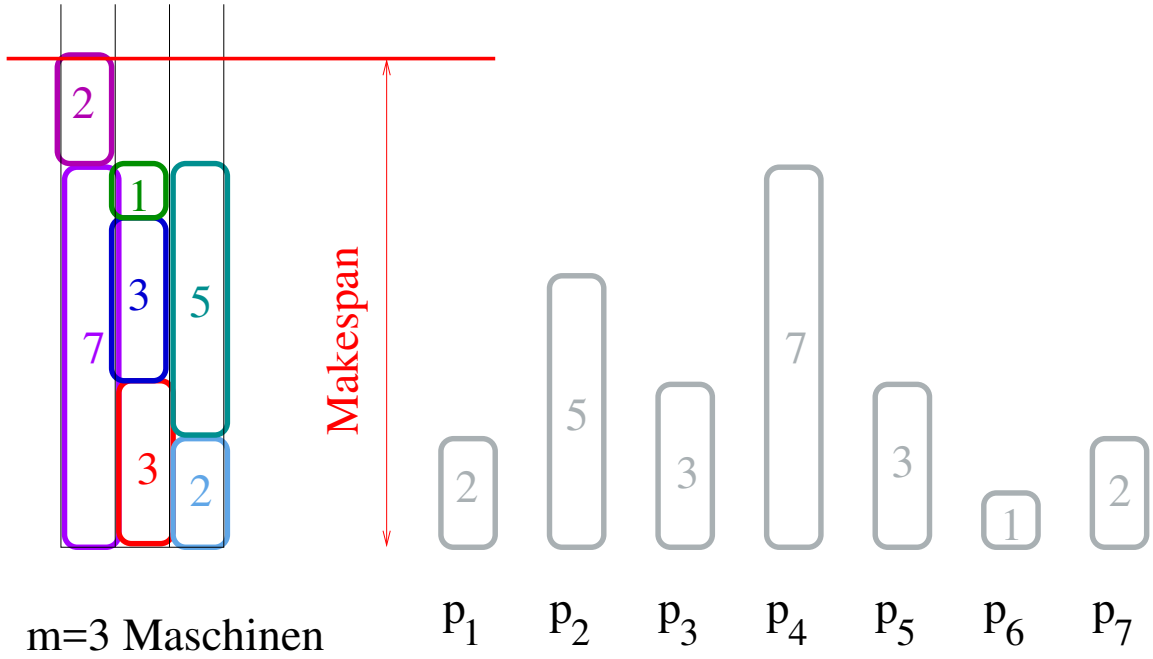


Eingabe: n Jobs mit Laufzeiten $p_1, p_2, \ldots, p_n > 0$ und m Maschinen

Ausgabe: Weise jeden Job genau einer Maschine zu

Ziel: Der Makespan soll minimiert werden

Makespan Scheduling auf identischen Maschinen



Eingabe: n Jobs mit Laufzeiten $p_1, p_2, \ldots, p_n > 0$ und m Maschinen

Ausgabe: Weise jeden Job genau einer Maschine zu

Ziel: Der Makespan soll minimiert werden

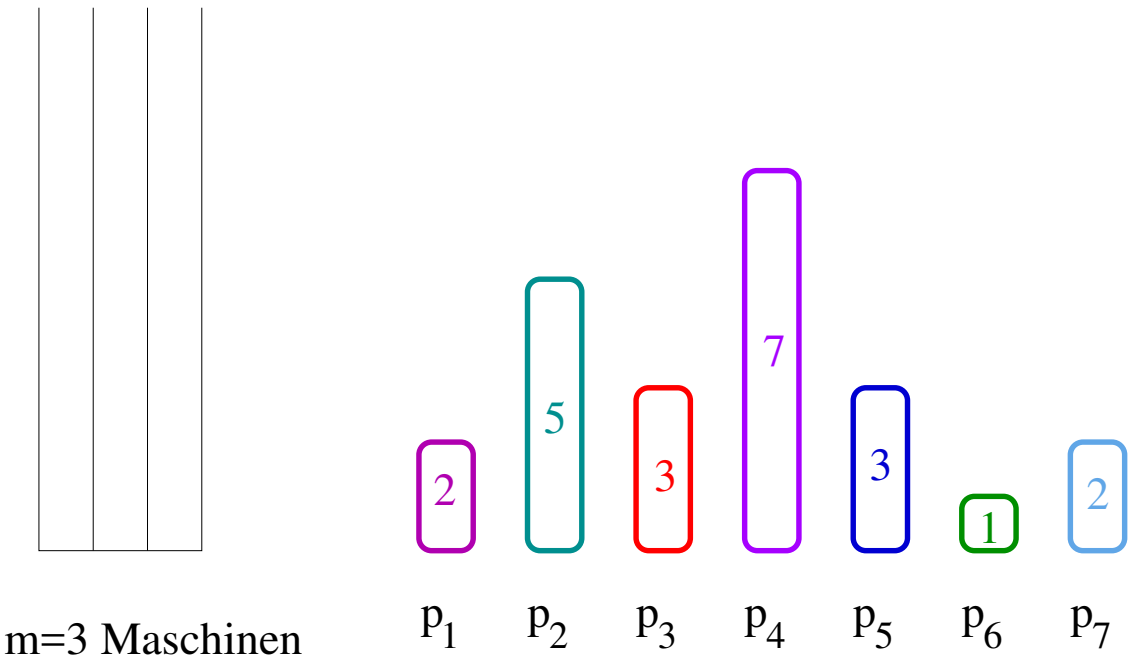
(Makespan = maximale Fertigstellungszeit)

Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit

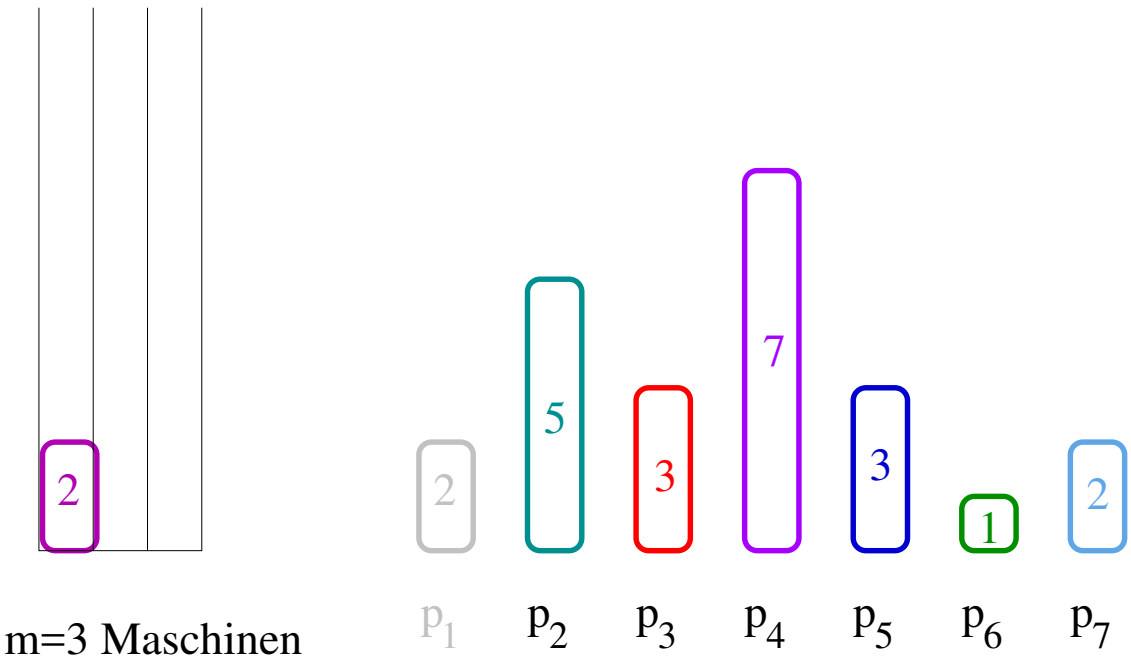
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



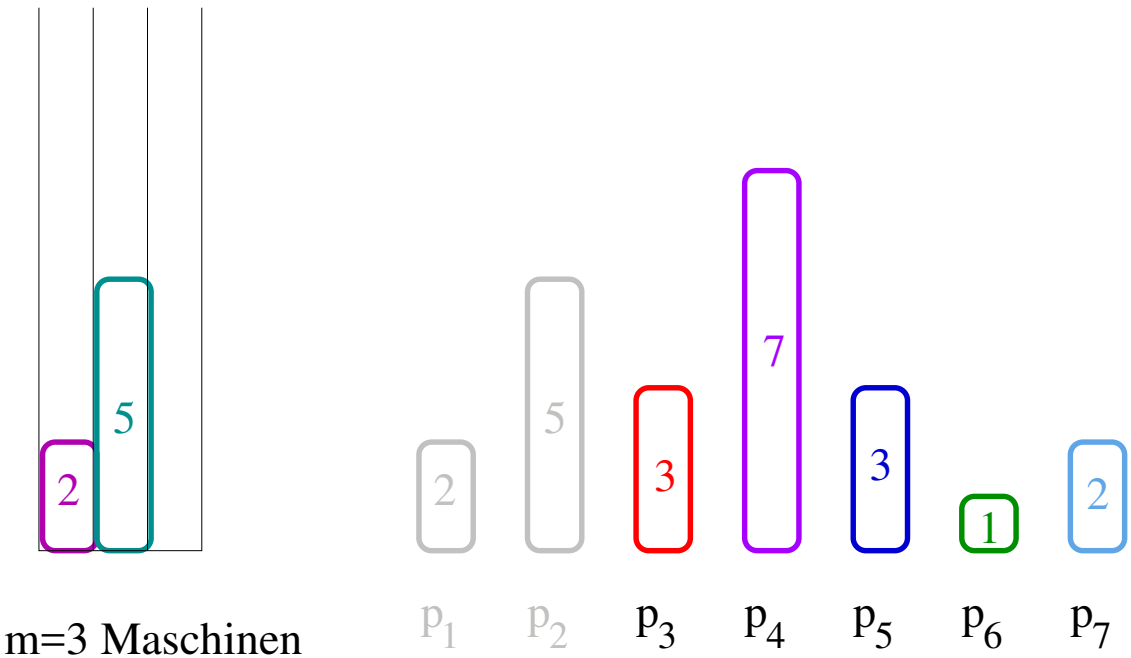
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



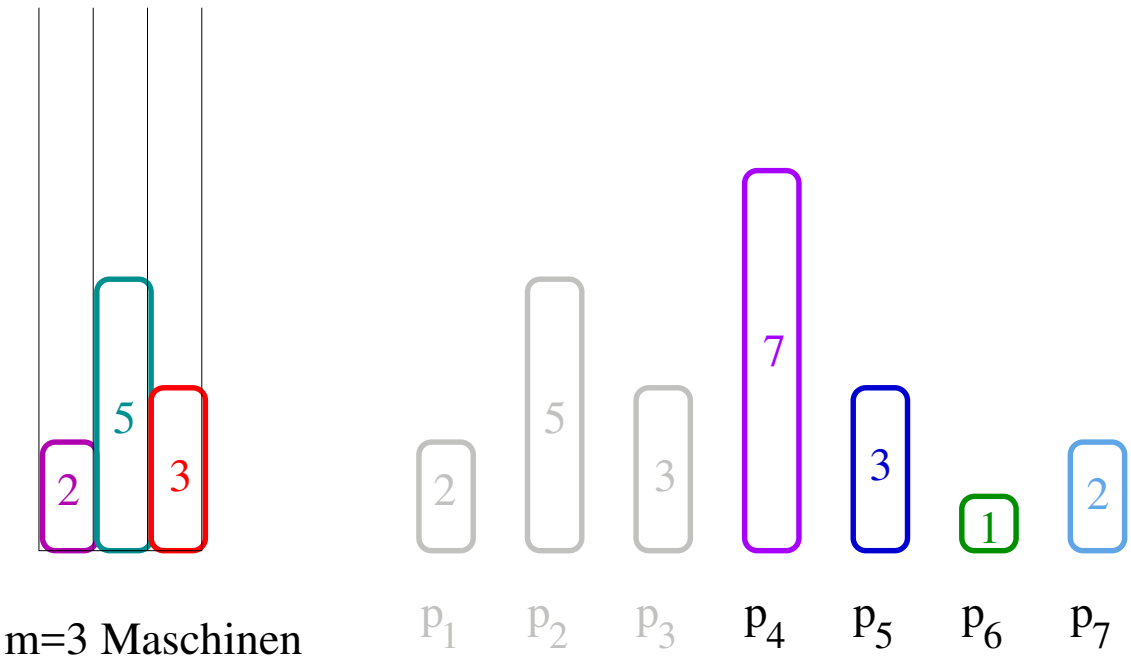
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



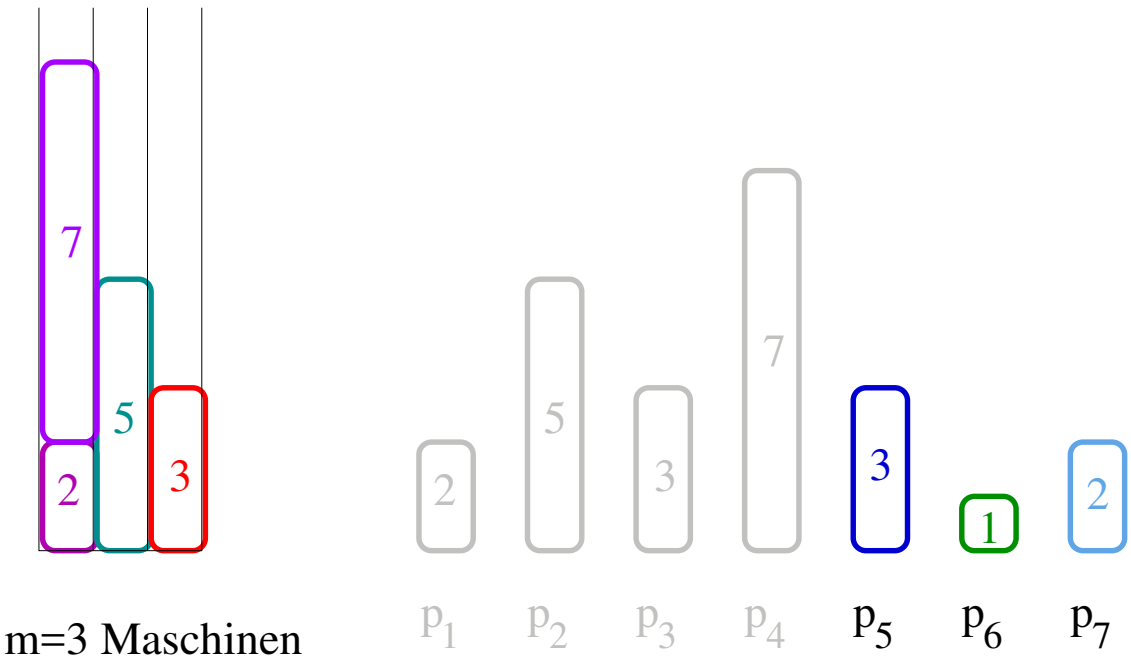
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



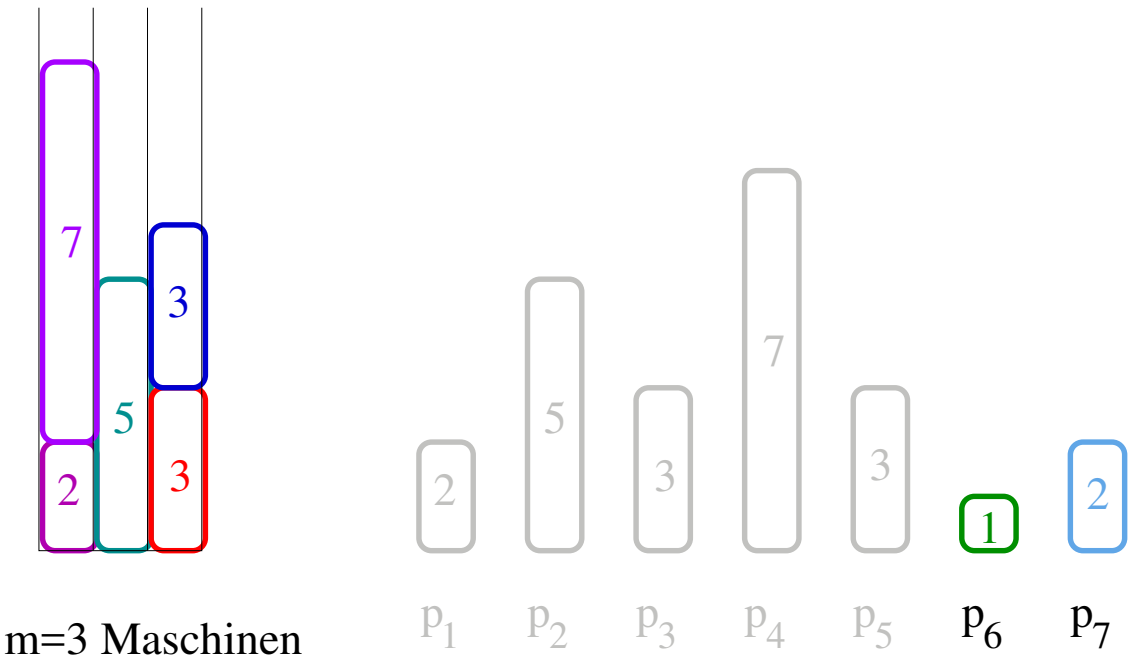
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



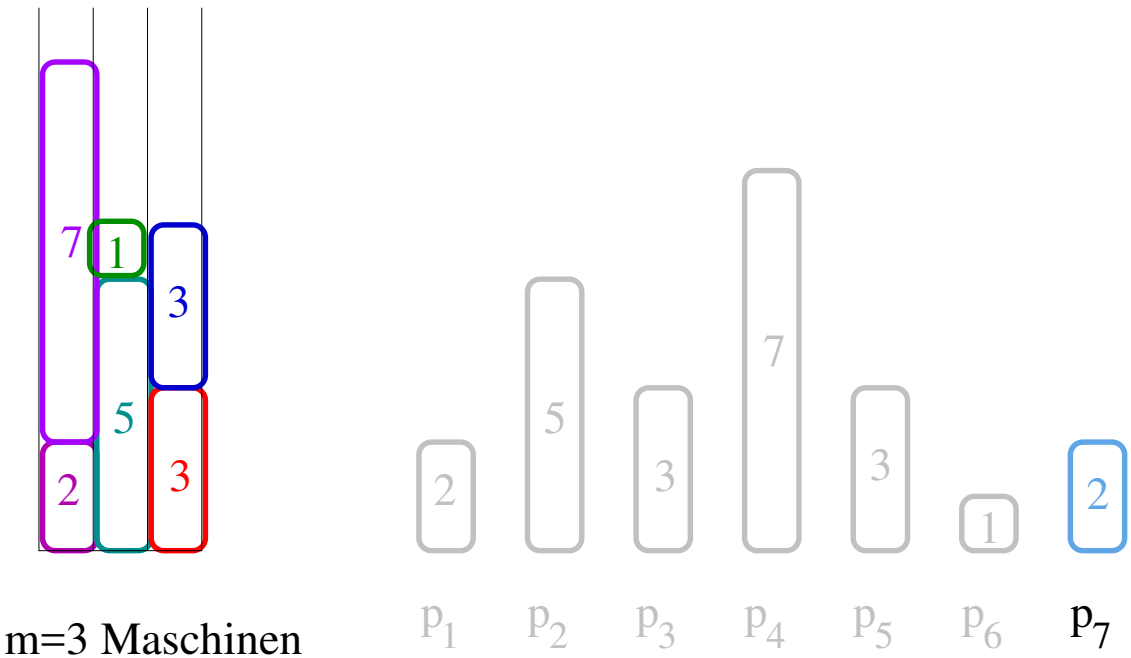
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



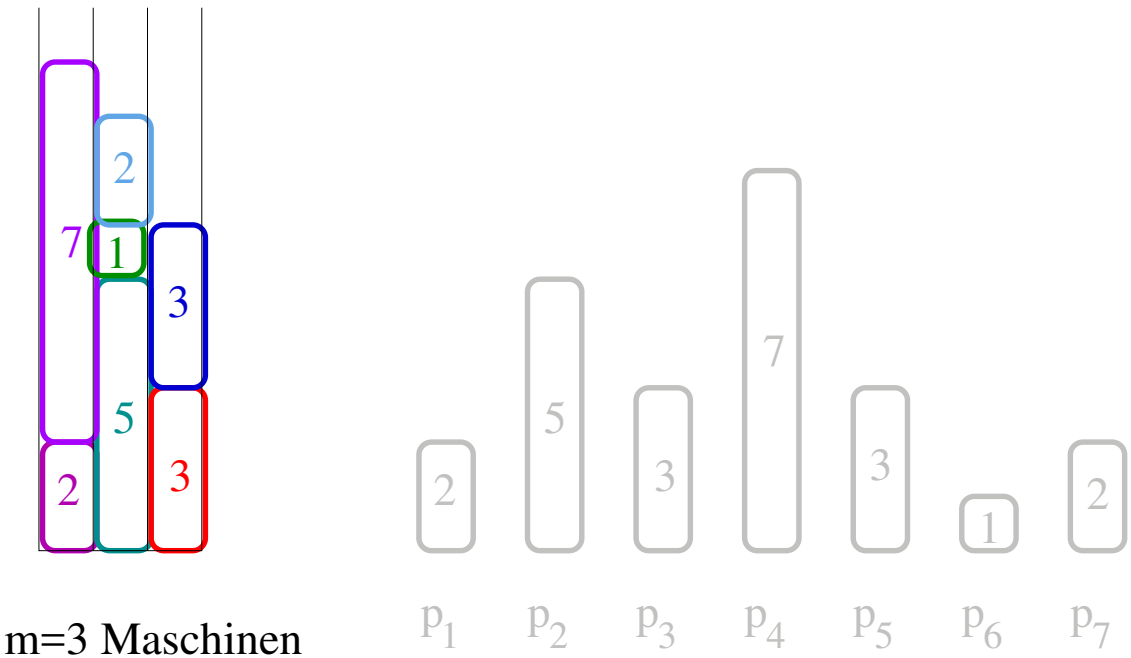
Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



Greedy LIST-Scheduling Algorithmus

- Weise die Jobs nacheinander den Maschinen zu:
- Setze p_i auf eine Maschine mit bisher kleinster Gesamtlaufzeit



(LIST kann auch als ein *offline* Algorithmus angesehen werden.)

Online Algorithmen

Definition:

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

Online Algorithmen

Definition:

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

A ist ein...

- ...**offline Algorithmus**, falls die Ausgabe berechnet wird, nachdem die gesamte Eingabe betrachtet wurde;

Online Algorithmen

Definition:

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

A ist ein...

- ...**offline Algorithmus**, falls die Ausgabe berechnet wird, nachdem die gesamte Eingabe betrachtet wurde;
- ...**online Algorithmus**, falls (für jedes i) während der Berechnung von τ_i nur der Teil $(\sigma_1, \sigma_2, \ldots, \sigma_i)$ der Eingabe bekannt ist.

Der Wettbewerbsfaktor

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Der Wettbewerbsfaktor

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Für einen Algorithmus A und eine Zielfunktion f bezeichne $A_\sigma = f(\sigma, A(\sigma)))$.

Der Wettbewerbsfaktor

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Für einen Algorithmus A und eine Zielfunktion f bezeichne $A_\sigma = f(\sigma, A(\sigma)))$.

Definition:

Ein offline Algorithmus OPT ist **optimal für die Zielfunktion** f, falls

$$OPT_\sigma \leq A_\sigma$$

für jeden Algorithmus A und jede Eingabe σ gilt.

Der Wettbewerbsfaktor

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Für einen Algorithmus A und eine Zielfunktion f bezeichne $A_\sigma = f(\sigma, A(\sigma)))$.

Definition:

Ein offline Algorithmus OPT ist **optimal für die Zielfunktion** f, falls

$$OPT_\sigma \leq A_\sigma$$

für jeden Algorithmus A und jede Eingabe σ gilt.

Definition:

Ein online Algorithmus A hat einen **Wettbewerbsfaktor** von höchstens α, wenn es eine Konstante c gibt, so dass für jede Eingabe σ gilt

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

Wettbewerbsfaktor von LIST-Scheduling

Theorem:

Für eine beliebige Eingabe-Instanz σ seien $LIST_\sigma$ und OPT_σ der Makespan von LIST bzw. der optimale Makespan. Dann gilt:

$$LIST_\sigma \leq \left(2 - \frac{1}{m}\right) \cdot OPT_\sigma.$$

Der **Wettbewerbsfaktor** von LIST ist somit **höchstens** $2 - \frac{1}{m}$.

Wettbewerbsfaktor von LIST-Scheduling

Theorem:

Für eine beliebige Eingabe-Instanz σ seien $LIST_\sigma$ und OPT_σ der Makespan von LIST bzw. der optimale Makespan. Dann gilt:

$$LIST_\sigma \leq \left(2 - \frac{1}{m}\right) \cdot OPT_\sigma.$$

Der **Wettbewerbsfaktor** von LIST ist somit **höchstens** $2 - \frac{1}{m}$.

(Bemerkung: Er ist auch mindestens (also genau) $2 - \frac{1}{m}$.)

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Es gibt einen langsamen externen Speicher (Disk, Internet..) auf dem alle Seiten gespeichert sind.

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Es gibt einen langsamen externen Speicher (Disk, Internet..) auf dem alle Seiten gespeichert sind.

Eingabe: Eine Folge von Seiten $\sigma_1, \sigma_2, \ldots$, werden online angefordert.

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Es gibt einen langsamen externen Speicher (Disk, Internet..) auf dem alle Seiten gespeichert sind.

Eingabe: Eine Folge von Seiten $\sigma_1, \sigma_2, \ldots$, werden online angefordert.

Wenn der Cache voll ist und die angeforderte Seite nicht im Cache, dann soll eine Seite **ausgelagert** werden.

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Es gibt einen langsamen externen Speicher (Disk, Internet..) auf dem alle Seiten gespeichert sind.

Eingabe: Eine Folge von Seiten $\sigma_1, \sigma_2, \ldots$, werden online angefordert.

Wenn der Cache voll ist und die angeforderte Seite nicht im Cache, dann soll eine Seite **ausgelagert** werden.

Welche Seite ausgelagert wird entscheidet (online) Paging Algorithmus.

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Es gibt einen langsamen externen Speicher (Disk, Internet..) auf dem alle Seiten gespeichert sind.

Eingabe: Eine Folge von Seiten $\sigma_1, \sigma_2, \ldots$, werden online angefordert.

Wenn der Cache voll ist und die angeforderte Seite nicht im Cache, dann soll eine Seite **ausgelagert** werden.

Welche Seite ausgelagert wird entscheidet (online) Paging Algorithmus.

Für jede angeforderte Seite, die nicht im Cache vorhanden ist, gibt es einen **Page-Fault** (Strafpunkt).

Das Paging Problem

Es gibt einen schnellen internen Speicher (Cache) in den k Seiten passen.

Es gibt einen langsamen externen Speicher (Disk, Internet..) auf dem alle Seiten gespeichert sind.

Eingabe: Eine Folge von Seiten $\sigma_1, \sigma_2, \ldots$, werden online angefordert.

Wenn der Cache voll ist und die angeforderte Seite nicht im Cache, dann soll eine Seite **ausgelagert** werden.

Welche Seite ausgelagert wird entscheidet (online) Paging Algorithmus.

Für jede angeforderte Seite, die nicht im Cache vorhanden ist, gibt es einen **Page-Fault** (Strafpunkt).

Zielfunktion: Minimiere die Anzahl der Page-Faults.

Die Strategien FIFO und LRU

FIFO: (First-In-First-Out)

Lagere die Seite aus, die *am längsten gespeichert* wurde.

Die Strategien FIFO und LRU

FIFO: (First-In-First-Out)

Lagere die Seite aus, die *am längsten gespeichert* wurde.

LRU: (Least-Recently-Used)

Lagere die Seite aus, deren *letzte Anforderung* am weitesten zurückliegt.

Die Strategien FIFO und LRU

FIFO: (First-In-First-Out)

Lagere die Seite aus, die *am längsten gespeichert* wurde.

LRU: (Least-Recently-Used)

Lagere die Seite aus, deren *letzte Anforderung* am weitesten zurückliegt.

Theorem:

FIFO und LRU haben **Wettbewerbsfaktor** k**.**

Die Strategien FIFO und LRU

FIFO: (First-In-First-Out)

Lagere die Seite aus, die *am längsten gespeichert* wurde.

LRU: (Least-Recently-Used)

Lagere die Seite aus, deren *letzte Anforderung* am weitesten zurückliegt.

Theorem:

FIFO und LRU haben **Wettbewerbsfaktor** k**.**

Theorem:

Jeder deterministische online Paging-Algorithmus hat **Wettbewerbsfaktor mindestens** k.

FIFO und LRU (Analyse)

Behauptung 1:

In einer beliebigen Teilfolge σ' mit **mindestens** $k+1$ **verschiedenen Seiten** hat jeder Algorithmus **mindestens einen Strafpunkt**.

FIFO und LRU (Analyse)

Behauptung 1:

In einer beliebigen Teilfolge σ' mit **mindestens** $k+1$ **verschiedenen Seiten** hat jeder Algorithmus **mindestens einen Strafpunkt**.

Behauptung 2:

In FIFO und LRU gilt:

Zwischen zwei Anforderungen an **dieselbe Seite** p, die **beide einen Page-Fault** verursachen, gibt es **mindestens** k **verschiedene Seiten** in der Eingabefolge.

Mit p zusammen gibt es mindestens $k+1$ verschiedene Seiten.

Schwächere Online Strategien

LFU: (Least-Frequently Used)

Lagere die Seite aus, die *seit ihrer letzten Einlagerung am seltensten angefordert* wurde

Schwächere Online Strategien

LFU: (Least-Frequently Used)

Lagere die Seite aus, die *seit ihrer letzten Einlagerung am seltensten angefordert* wurde

FWF: (Flush-When-Full)

Wenn der Cache voll ist und Seitenfehler auftritt, *entleere den Cache* und lagere die Seite ein

Ein randomisierter Ansatz: Marking-Algorithmus

1. Die Seite p wird angefordert

Ein randomisierter Ansatz: Marking-Algorithmus

1. Die Seite p wird angefordert
2. **if** p im Cache gespeichert **then**
3. **if** p unmarkiert **then** markiere p.

Ein randomisierter Ansatz: Marking-Algorithmus

1. Die Seite p wird angefordert
2. **if** p im Cache gespeichert **then**
3. **if** p unmarkiert **then** markiere p.
4. **else** (p nicht im Cache)
5. **if** es gibt nicht-markierte Seiten im Cache **then**
6. Wähle uniform zufällig eine nicht-markierte Seite q im Cache aus

Ein randomisierter Ansatz: Marking-Algorithmus

1. Die Seite p wird angefordert
2. **if** p im Cache gespeichert **then**
3. **if** p unmarkiert **then** markiere p.
4. **else** (p nicht im Cache)
5. **if** es gibt nicht-markierte Seiten im Cache **then**
6. Wähle uniform zufällig eine nicht-markierte Seite q im Cache aus
7. **else** (alle Seiten im Cache markiert)
8. Lösche alle Markierungen
9. Wähle uniform zufällig eine Seite q im Cache aus

Ein randomisierter Ansatz: Marking-Algorithmus

1. Die Seite p wird angefordert
2. **if** p im Cache gespeichert **then**
3. **if** p unmarkiert **then** markiere p.
4. **else** (p nicht im Cache)
5. **if** es gibt nicht-markierte Seiten im Cache **then**
6. Wähle uniform zufällig eine nicht-markierte Seite q im Cache aus
7. **else** (alle Seiten im Cache markiert)
8. Lösche alle Markierungen
9. Wähle uniform zufällig eine Seite q im Cache aus
10. Lagere q aus und p ein. Markiere p.

Wettbewerbsfaktor (randomisiert)

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Wettbewerbsfaktor (randomisiert)

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Für einen **randomisierten** Algorithmus A und eine Zielfunktion f bezeichne

$$\mathbb{E}[A_\sigma] = \mathbb{E}[f(\sigma, A(\sigma)))]$$

den **erwarteten Zielwert** bei Eingabesequenz σ und randomisierter Ausgabe $A(\sigma)$ des Algorithmus.

Wettbewerbsfaktor (randomisiert)

Für ein gegebenes online Problem sei $f(\sigma, \tau)$ eine Zielfunktion deren Wert wir minimieren (oder maximieren) möchten.

Für einen **randomisierten** Algorithmus A und eine Zielfunktion f bezeichne

$$\mathbb{E}[A_\sigma] = \mathbb{E}[f(\sigma, A(\sigma))]$$

den **erwarteten Zielwert** bei Eingabesequenz σ und randomisierter Ausgabe $A(\sigma)$ des Algorithmus.

Definition:

Ein **randomisierter** online Algorithmus A hat einen **Wettbewerbsfaktor** von höchstens α, wenn es eine Konstante c gibt, so dass für jede Eingabe σ gilt

$$\mathbb{E}[A_\sigma] \leq \alpha \cdot OPT_\sigma + c$$

Analyse von Marking: Vorbereitung

Analyse von Marking: Vorbereitung

Sei σ eine Eingabefolge.

Analyse von Marking: Vorbereitung

Sei σ eine Eingabefolge.

Wir zerlegen σ iterativ in die Teilfolgen

$$\sigma^0 \quad \sigma^1 \quad \sigma^2 \quad \ldots \quad \sigma^i \quad \ldots$$

Analyse von Marking: Vorbereitung

Sei σ eine Eingabefolge.

Wir zerlegen σ iterativ in die Teilfolgen

$$\sigma^0 \quad \sigma^1 \quad \sigma^2 \quad \ldots \quad \sigma^i \quad \ldots$$

Dabei gilt:

- σ^0 ist die längstmögliche Anfangsfolge mit k verschiedenen Seiten

Analyse von Marking: Vorbereitung

Sei σ eine Eingabefolge.

Wir zerlegen σ iterativ in die Teilfolgen

$$\sigma^0 \quad \sigma^1 \quad \sigma^2 \quad \ldots \quad \sigma^i \quad \ldots$$

Dabei gilt:

- σ^0 ist die längstmögliche Anfangsfolge mit k verschiedenen Seiten
- σ^1 ist längste, auf σ^0 folgende Teilfolge mit k verschiedenen Seiten

  $\vdots$

- σ^i ist längste, auf σ^{i-1} folgende Teilfolge mit k verschiedenen Seiten

Analyse von Marking: Vorbereitung

Sei σ eine Eingabefolge.

Wir zerlegen σ iterativ in die Teilfolgen

$$\sigma^0 \quad \sigma^1 \quad \sigma^2 \quad \ldots \quad \sigma^i \quad \ldots$$

Dabei gilt:

- σ^0 ist die längstmögliche Anfangsfolge mit k verschiedenen Seiten
- σ^1 ist längste, auf σ^0 folgende Teilfolge mit k verschiedenen Seiten

  $\vdots$

- σ^i ist längste, auf σ^{i-1} folgende Teilfolge mit k verschiedenen Seiten

Die Definition der Teilfolgen σ^i hängt **nur von der Eingabefolge** σ ab, aber **nicht vom Algorithmus**.

Analyse von Marking: Definition

Analyse von Marking: Definition

- Wir nennen S_i die Menge der in σ^{i-1} gefragten k versch. Seiten

Analyse von Marking: Definition

- Wir nennen S_i die Menge der in σ^{i-1} gefragten k versch. Seiten
- Für σ^i nennen wir eine Seite *alt* wenn sie auch in σ^{i-1} gefragt wurde, und *neu* sonst

Analyse von Marking: Definition

- Wir nennen S_i die Menge der in σ^{i-1} gefragten k versch. Seiten
- Für σ^i nennen wir eine Seite *alt* wenn sie auch in σ^{i-1} gefragt wurde, und *neu* sonst
- Sei n_i die Anzahl der neuen Anforderungen in σ^i.
  n_i ist unabhängig vom Algorithmus!

Analyse von Marking: Definition

- Wir nennen S_i die Menge der in σ^{i-1} gefragten k versch. Seiten
- Für σ^i nennen wir eine Seite *alt* wenn sie auch in σ^{i-1} gefragt wurde, und *neu* sonst
- Sei n_i die Anzahl der neuen Anforderungen in σ^i.
  n_i ist unabhängig vom Algorithmus!

Die neuen Seiten verursachen auf jeden Fall Strafpunkte.

Wie viele Strafpunkte verursachen die alten Seiten?

Analyse von Marking: Analyse

Analyse von Marking: Analyse

Behauptung 1: Für Eingabe $\sigma = \sigma^0\sigma^1\sigma^2 \dots \sigma^m$ ist die Anzahl Strafpunkte für **jeden Paging-Algorithmus (auch OPT)** mindestens

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

Analyse von Marking: Analyse

Behauptung 1: Für Eingabe $\sigma = \sigma^0\sigma^1\sigma^2 \dots \sigma^m$ ist die Anzahl Strafpunkte für **jeden Paging-Algorithmus (auch OPT)** mindestens

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

Behauptung 2: Die **erwartete Anzahl der Strafpunkte von Marking** für die Teilfolge σ^i ist höchstens

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

insgesamt also höchstens

$$H_k \sum_{i=1}^{m} n_i.$$

Korollar: Der Wettbewerbsfaktor von MARKING ist höchstens $2H_k$.

Bemerkungen

- FIFO, LRU und FWF haben Wettbewerbsfaktor k ... Sind sie wirklich gleich gut?

Bemerkungen

- FIFO, LRU und FWF haben Wettbewerbsfaktor k ... Sind sie wirklich gleich gut?

- Praktische Erfahrungen und genauere Analysemethoden (z.B. via Locality-of-Reference) zeigen, dass LRU tatschlich oft die beste Strategie ist.

Bemerkungen

- FIFO, LRU und FWF haben Wettbewerbsfaktor k ... Sind sie wirklich gleich gut?
- Praktische Erfahrungen und genauere Analysemethoden (z.B. via Locality-of-Reference) zeigen, dass LRU tatschlich oft die beste Strategie ist.
- Der Wettbewerbsfaktor als Qualitätsmaß ist durchaus umstritten.

Bemerkungen

- FIFO, LRU und FWF haben Wettbewerbsfaktor k ... Sind sie wirklich gleich gut?
- Praktische Erfahrungen und genauere Analysemethoden (z.B. via Locality-of-Reference) zeigen, dass LRU tatschlich oft die beste Strategie ist.
- Der Wettbewerbsfaktor als Qualitätsmaß ist durchaus umstritten.
- Das gilt oftmals auch für die Worst-Case Analyse ohne "realitätsnahe" Einschränkungen...

Bemerkungen

- FIFO, LRU und FWF haben Wettbewerbsfaktor k ... Sind sie wirklich gleich gut?
- Praktische Erfahrungen und genauere Analysemethoden (z.B. via Locality-of-Reference) zeigen, dass LRU tatschlich oft die beste Strategie ist.
- Der Wettbewerbsfaktor als Qualitätsmaß ist durchaus umstritten.
- Das gilt oftmals auch für die Worst-Case Analyse ohne "realitätsnahe" Einschränkungen...
- Sogar für seine"eigene" Worst-Case Eingabe funktioniert LRU gut mit den allermeisten Cache-Größen $k' \neq k$.