Ersetzungsalgorithmen

Emilio Pielsticker

Kachelmengen

- Sei $K \subseteq \mathbb{Z}_n$ eine **Kachelmenge**
- Jede Kachel $k \in K$ besitzt einen eineindeutigen Index $\mathbf{Ind}(k)$
- Der aktuelle Inhalt einer TLB-Zeile oder eines Caches kann durch so eine Kachelmenge beschrieben werden

Ersetzungsalgorithmen

Problem: Anzahl der Kacheln im Cache bzw. einer TLB-Zeile darf eine bestimmte Kapazitätsgrenze $\hat{k}$ nicht überschreiten

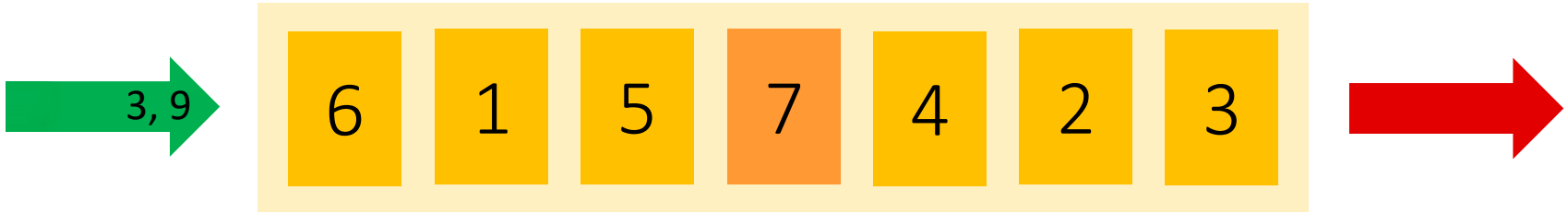
- Bei $|K| = \hat{k}$ müssen beim Hinzufügen neuer Kacheln andere Kacheln aus K wieder entfernt werden

→ $\textsc{ersetzungsproblem}$

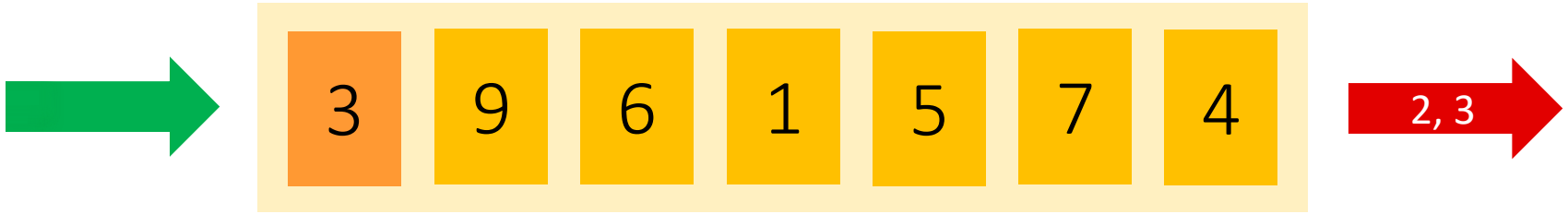
Lösung: Bestimmung einer zu entfernenden Kachel $k \in K$

→ Benötigt wird ein **Ersetzungsalgorithmus**

Windhund-Algorithmus



Windhund-Algorithmus



Nachteil: Bei der Verwendung des Algorithmus kann es unter anderem häufig vorkommen, dass häufig benötigte Einträge ausgelagert werden, um wenig später wieder eingelagert zu werden.

Präferenzrelation

- $\preccurlyeq \subset [|K|] \times [|K|]$ ist die **Präferenzrelation**:
- Das unter $\preccurlyeq$ vorzüglichste Element aus K heißt $\max_{\preccurlyeq} K$
- Gute Ersetzungsalgorithmen suchen nach $\max_{\preccurlyeq} K$

'Älter als'-Relation

Eine für Ersetzungsalgorithmen sinnvolle Relation ist die Folgende:

- i ist älter als j oder gleich alt: $j \preccurlyeq i \equiv (j, i) \in \preccurlyeq$ ('Älter als'-Relation)
- i ist älter als j: $j \prec i \equiv j \prec i \wedge i \neq j \equiv \neg(i \preccurlyeq j)$

Eine weitere sinnvolle Präferenzrelation könnte z.B.
eine 'Wird am seltensten verwendet'-Relation sein

Zweite-Chance-Algorithmus

Sobald Eintrag zu entfernen ist, wird ein Zeiger auf älteste Kachel gesetzt

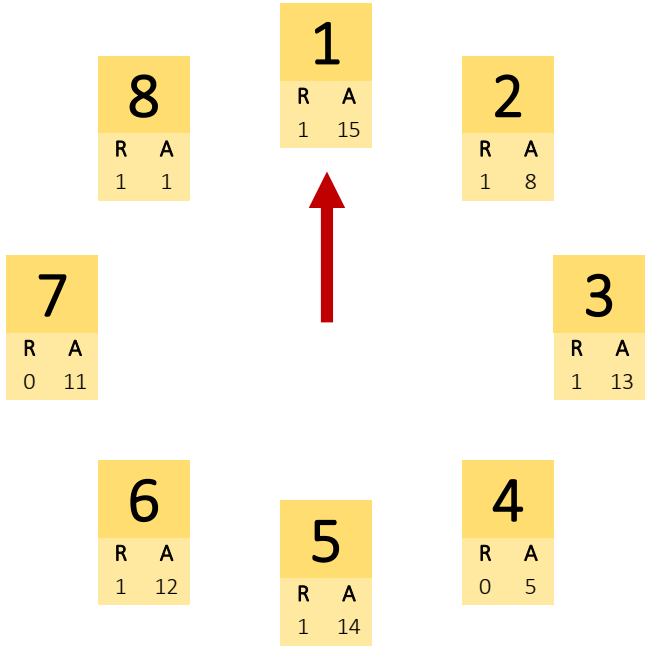
Wiederhole bis eine zu entfernende $(R = 0)$ **Kachel gefunden wurde**:

- R-Bit entfernt: Kachel erhält eine zweite Chance
- Zeiger auf die nächste Kachel setzen

Rückgabewert: Index der zu entfernenden Kachel

Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb:

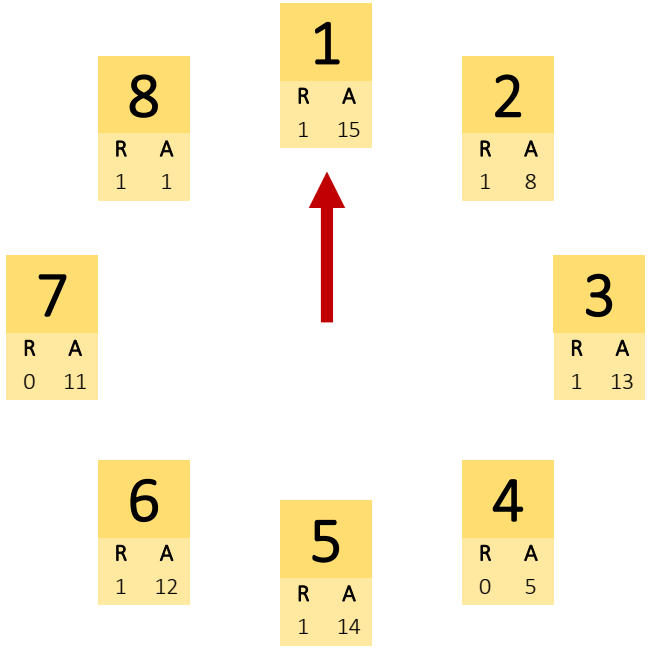


Anfrage (Beispiel):

- Zu entfernende Kachel finden

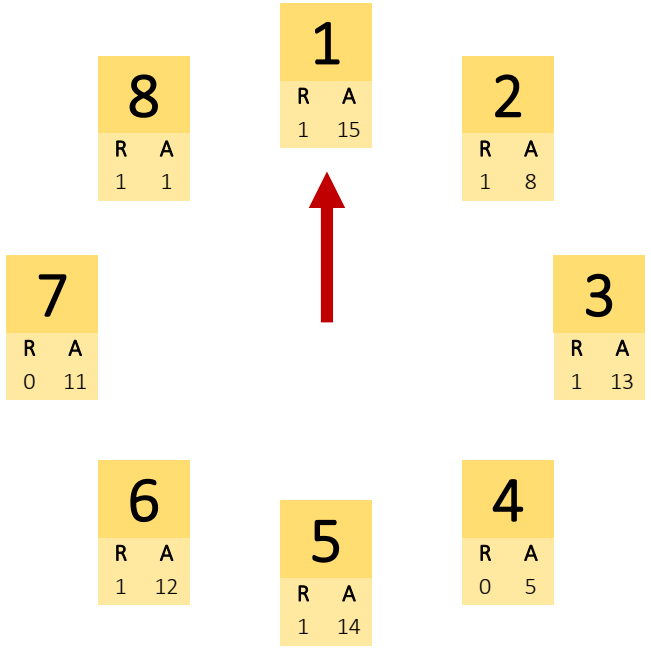
Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb: Zeiger auf die älteste Kachel setzen



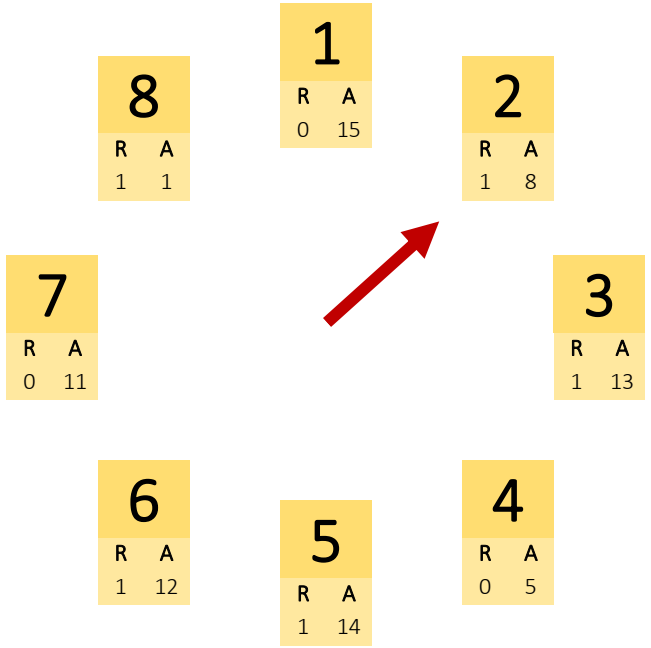
Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb: Entfernen der ersten Kachel



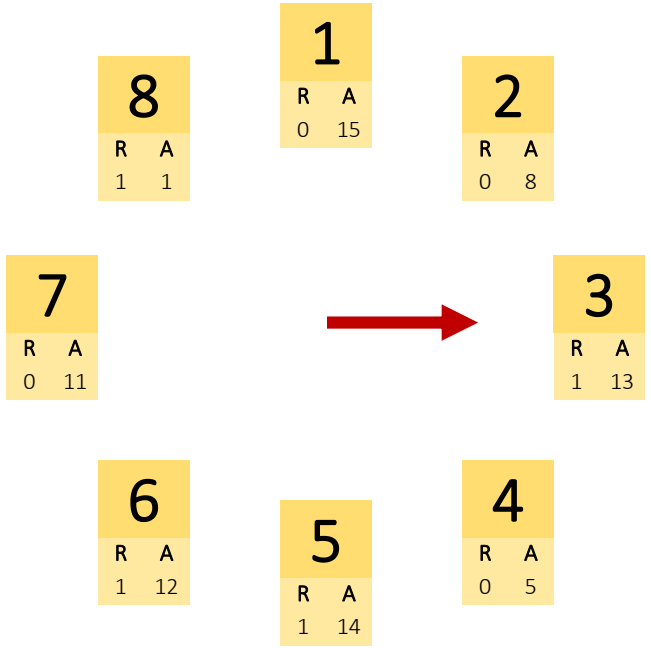
Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb: Entfernen der ersten Kachel



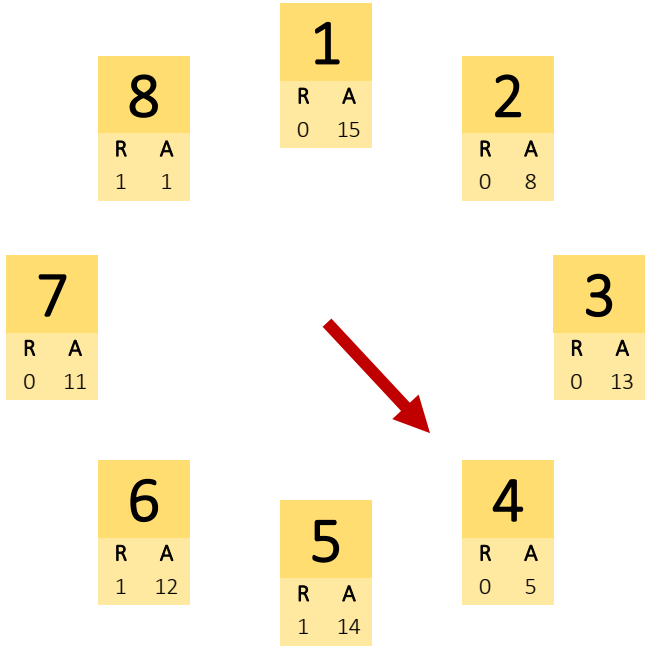
Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb: Entfernen der ersten Kachel



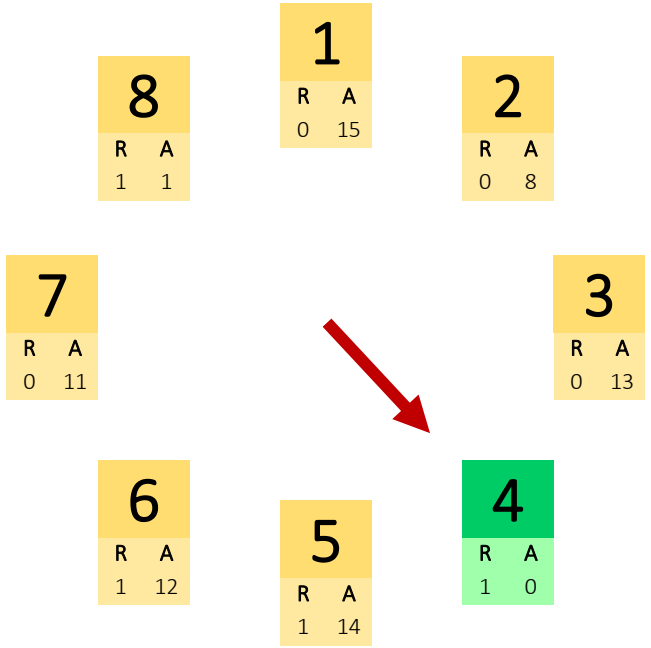
Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb: Entfernen der ersten Kachel



Zweite-Chance-Algorithmus

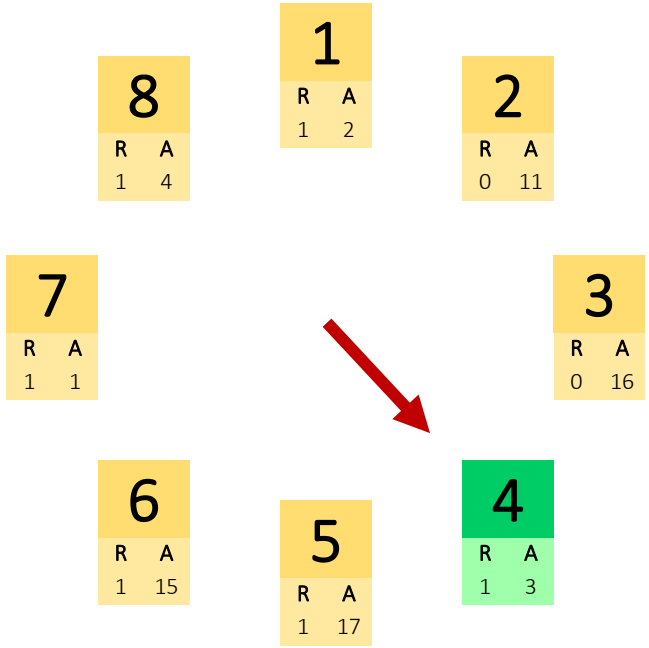
Zweite-Chance-Algorithmus im Betrieb: Entfernen der ersten Kachel



Kachel 4 wird ersetzt.

Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb:

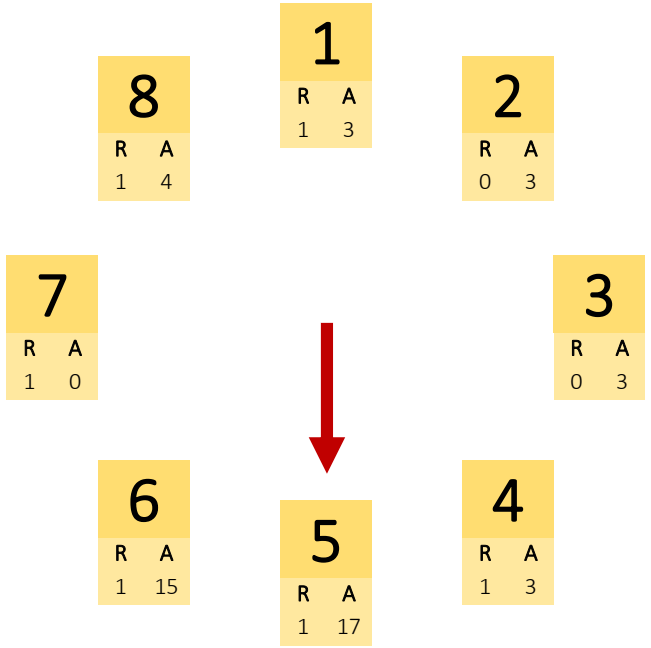


Anfragen (Beispiele):

- Nach 2 Zeitschritten:
  - Anfrage von 1
- Nach weiterem Zeitschritt:
  - Anfrage von 7
  - Zu entfernende Kachel finden

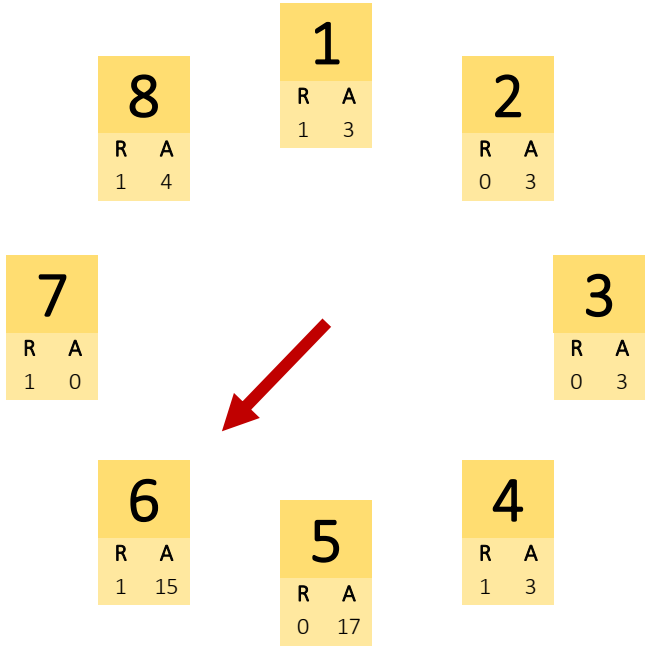
Zweite-Chance-Algorithmus

Zweite-Chance-Algorithmus im Betrieb: Zeiger auf älteste Kachel setzen



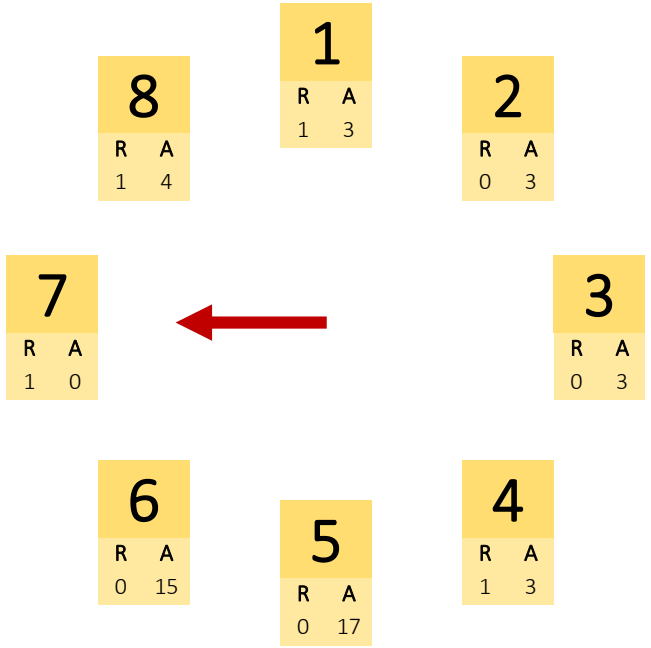
Zweite-Chance-Algorithmus

Suche nach zwei zu entfernenden Kacheln: Entfernen der zweiten Kachel



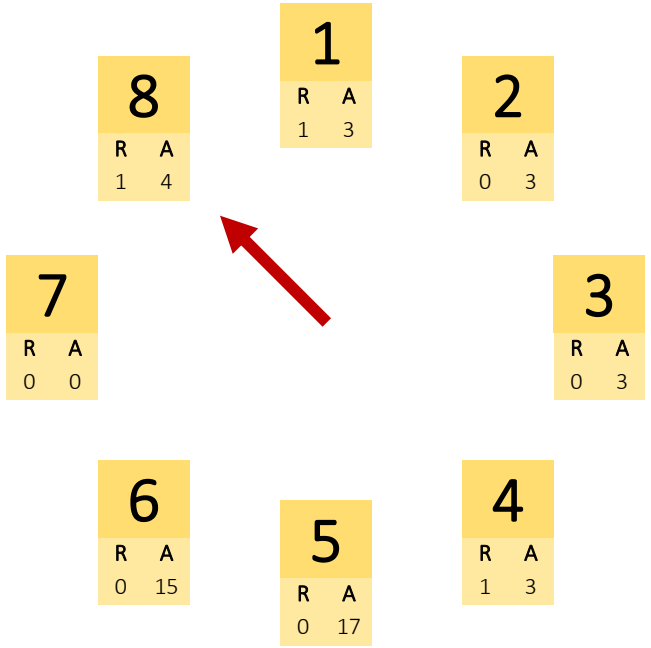
Zweite-Chance-Algorithmus

Suche nach zwei zu entfernenden Kacheln: Entfernen der zweiten Kachel



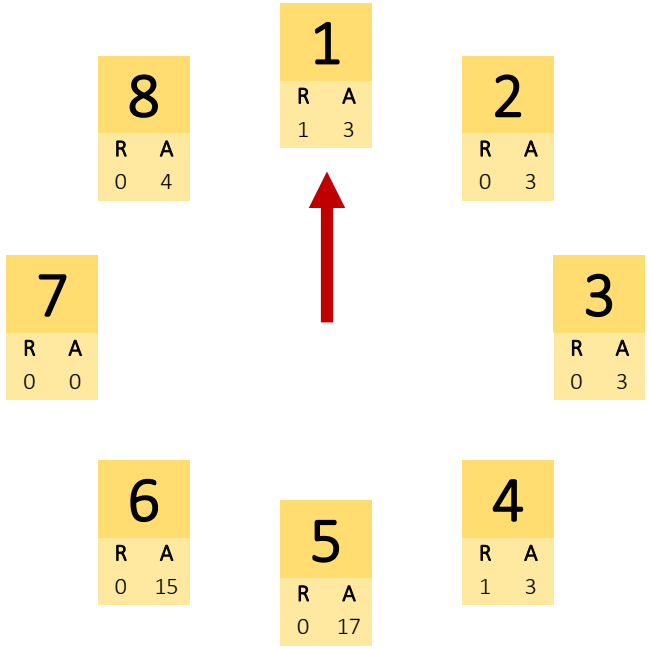
Zweite-Chance-Algorithmus

Suche nach zwei zu entfernenden Kacheln: Entfernen der zweiten Kachel



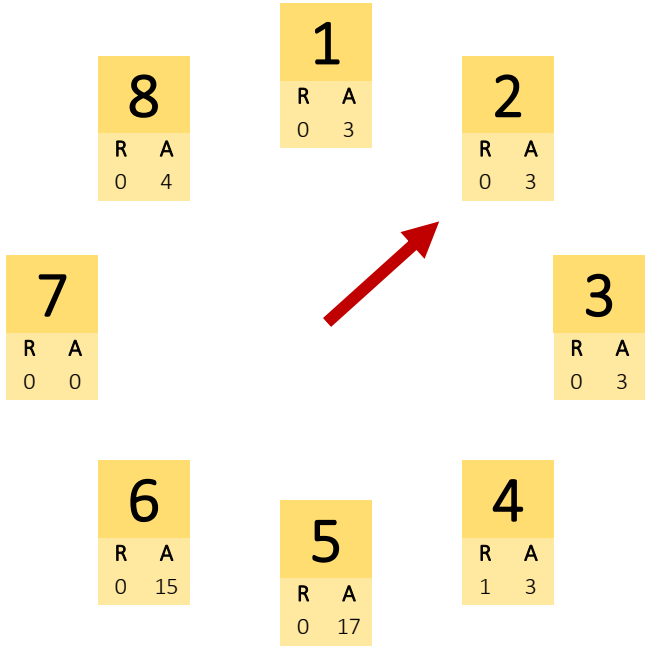
Zweite-Chance-Algorithmus

Suche nach zwei zu entfernenden Kacheln: Entfernen der zweiten Kachel



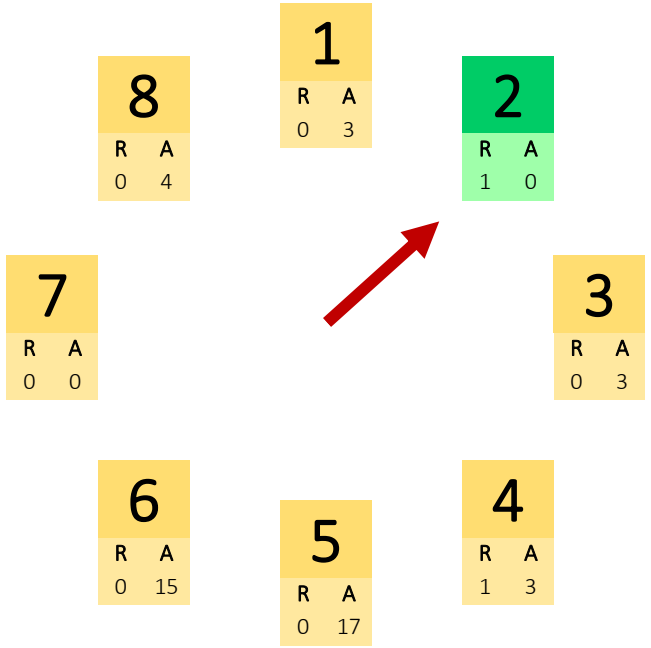
Zweite-Chance-Algorithmus

Suche nach zwei zu entfernenden Kacheln: Entfernen der zweiten Kachel



Zweite-Chance-Algorithmus

Suche nach zwei zu entfernenden Kacheln: Entfernen der zweiten Kachel



Kachel 2 wird ersetzt.

LFU-Algorithmus

Der LFU-Algorithmus entfernt genau die Kachel, der am wenigsten verwendet wurde. Statt des Alters wird die **Aufrufhäufigkeit** gespeichert.

Registertabelle

Zugriff:	1	5	3	3	5	4	4	2	7	4	4	4	5	6
Kachel 0:	1	1	1	1	1	1	1	2	7	7	7	7	7	6
Kachel 2:	-	5	5	5	5	5	5	5	5	5	5	5	5	5
Kachel 2:	-	-	3	3	3	3	3	3	3	3	3	3	3	3
Kachel 3:	-	-	-	-	-	4	4	4	4	4	4	4	4	4

Zugriffstabelle (Aufrufhäufigkeit)

Kachel 0:	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Kachel 2:	0	1	1	1	2	2	2	2	2	2	2	2	3	3
Kachel 2:	0	0	1	2	2	2	2	2	2	2	2	2	2	2
Kachel 3:	0	0	0	0	0	1	2	2	2	3	4	5	5	5

LFU = Least Frequently Used, dt. am seltensten verwendet

Nachteil: LFU kann nur mit spezieller Hardware implementiert werden und ist für Caches ungeeignet → M-Bit

LRU-Algorithmus

Der LRU-Algorithmus entfernt genau die Kachel, der am längsten nicht mehr verwendet wurde. LRU besitzt eine bessere Performance als LFU.

Registertabelle

Zugriff:	**1**	**5**	**3**	**3**	**5**	**4**	**4**	**2**	**7**	**4**	**9**	**1**	**4**	**6**
Kachel 0:	1	1	1	1	1	1	1	2	2	2	2	1	1	1
Kachel 2:	-	5	5	5	5	5	5	5	5	5	9	9	9	9
Kachel 2:	-	-	3	3	3	3	3	3	7	7	7	7	7	6
Kachel 3:	-	-	-	-	-	4	4	4	4	4	4	4	4	4

Zugriffstabelle (Zahl der Zyklen seit des letzten Aufrufs)

Kachel 0:	0	1	2	3	4	5	6	0	1	2	3	0	1	2
Kachel 2:	INT_MAX	0	1	2	0	1	2	3	4	5	0	1	2	3
Kachel 2:	INT_MAX	INT_MAX	0	0	1	2	3	4	0	1	2	3	4	0
Kachel 3:	INT_MAX	INT_MAX	INT_MAX	INT_MAX	INT_MAX	0	0	1	2	0	1	2	0	1

LRU = Least Recently Used, dt. vor längster Zeit zuletzt verwendet …

Nachteil: LRU kann nur mit spezieller Hardware implementiert werden und ist für Caches ungeeignet → M-Bit

Adjazenztafel der 'Älter als'-Relation

- $\preccurlyeq$ kann als Adjazenzmatrix $\mathbf{Adj}_{\preccurlyeq}$ dargestellt werden
- $j \preccurlyeq i \equiv \mathbf{Adj}_{\preccurlyeq}[i,j] = 1$, ansonsten gilt $\mathbf{Adj}_{\preccurlyeq}[i,j] = 0$
- Offensichtlich muss $\preccurlyeq$ reflexiv sein: Für die Diagonale $d_0 \cdots d_{n-1} = 1$
- Außerdem ist $\preccurlyeq$ antisymmetrisch: $\mathbf{Adj}_{\preccurlyeq}[i,j] = 0 \Leftrightarrow \mathbf{Adj}_{\preccurlyeq}[j,i] = 1$
- Statt $\forall j \in \{k \in K | j > k\}$ wird für ein festes k hier $\forall j > k$ geschrieben

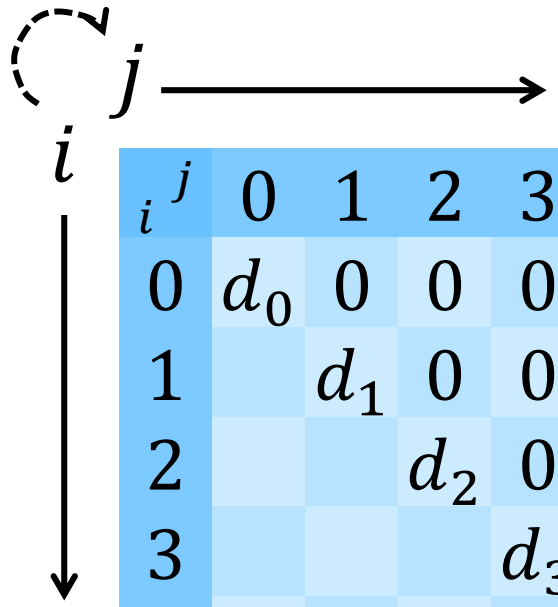
Adjazenztafel der 'Älter als'-Relation

- $\preccurlyeq$ kann als Adjazenzmatrix $\mathrm{Adj}_{\preccurlyeq}$ dargestellt werden
- $j \preccurlyeq i \equiv \mathrm{Adj}_{\preccurlyeq}[i,j] = 1$, ansonsten gilt $\mathrm{Adj}_{\preccurlyeq}[i,j] = 0$
- Offensichtlich muss $\preccurlyeq$ reflexiv sein: Für die Diagonale $d_0 \cdots d_{n-1} = 1$
- Außerdem ist $\preccurlyeq$ antisymmetrisch: $\mathrm{Adj}_{\preccurlyeq}[i,j] = 0 \Leftrightarrow \mathrm{Adj}_{\preccurlyeq}[j,i] = 1$
- Statt $\forall j \in \{k \in K | j > k\}$ wird für ein festes k hier $\forall j > k$ geschrieben

Verwendung im LRU-Algorithmus (Ähnliches Vorgehen bei LFU):

- $\preccurlyeq$ kann sich nach jedem Kachelaufruf verändern
- $\preccurlyeq_t$ ist die 'Älter als'-Relation nach dem Zeitschritt t

Adjazenztafel der 'Älter als'-Relation (Beispiel)



i \ j	0	1	2	3	4	5	6	7
0	d_0	0	0	0	0	0	0	0
1		d_1	0	0	0	0	1	0
2			d_2	0	1	1	1	0
3				d_3	1	1	1	0
4					d_4	0	1	0
5						d_5	1	0
6							d_6	0
7								d_7

LRU-Algorithmus mit Adjazenzmatrix (Aufruf)

- Wurde unmittelbar vor dem t-ten Zeitschritt die Kachel $k \in K$ aufgerufen, so muss k unter $\preccurlyeq_t$ als $\preccurlyeq$ bzw. $\prec$ die folgenden Eigenschaften erfüllen:
  - $\forall j > k.\ \mathrm{Adj}_{\preccurlyeq}[k, j] = 0$ bzw. $\forall j > k.\ k \prec j$: Zeile auf Null setzen
  - $\forall i < k.\ \mathrm{Adj}_{\preccurlyeq}[i, k] = 1$ bzw. $\forall i < k.\ k \preccurlyeq i$: Spalte auf Eins setzen
- Der Älteste Eintrag zum Zeitpunkt t wird $x := \max_{\preccurlyeq_t} K$ genannt:

LRU-Algorithmus mit Adjazenzmatrix (Aufruf)

- Wurde unmittelbar vor dem t-ten Zeitschritt die Kachel $k \in K$ aufgerufen, so muss k unter $\preccurlyeq_t$ als $\preccurlyeq$ bzw. $\prec$ die folgenden Eigenschaften erfüllen:
  - $\forall j > k.\ \mathrm{Adj}_{\preccurlyeq}[k, j] = 0$ bzw. $\forall j > k.\ k \prec j$: Zeile auf Null setzen
  - $\forall i < k.\ \mathrm{Adj}_{\preccurlyeq}[i, k] = 1$ bzw. $\forall i < k.\ k \preccurlyeq i$: Spalte auf Eins setzen
- Der Älteste Eintrag zum Zeitpunkt t wird $x := \max_{\preccurlyeq_t} K$ genannt:
  - $\forall i < x.\ \mathrm{Adj}_{\preccurlyeq}[i, x] = 0$ bzw. $\forall i < x.\ i \prec x$: Spalte ist Null
  - $\forall j > x.\ \mathrm{Adj}_{\preccurlyeq}[x, j] = 1$ bzw. $\forall j > x.\ j \preccurlyeq x$: Zeile ist Eins

  } $\forall j \in K.\ j \preccurlyeq x$

LRU-Algorithmus mit Adjazenzmatrix (Beispiel)

Prüfen, welcher Eintrag der älteste ist:

i \ j	0	1	2	**3**	4	5	6
0	d_0	0	0	0	0	0	0
1		d_1	0	0	0	0	1
2			d_2	0	1	1	1
3				$\boldsymbol{d_3}$	1	1	1
4					d_4	0	1
5						d_5	1
6							d_6

$k = 3$

Ältesten Eintrag ersetzen:

i \ j	0	1	2	**3**	4	5	6
0	d_0	0	0	1	0	0	0
1		d_1	0	1	0	0	1
2			d_2	1	1	1	1
3				$\boldsymbol{d_3}$	0	0	0
4					d_4	0	1
5						d_5	1
6							d_6

Weitere Beispiele zu Kachelaufrufen

i \ j	0	1	2	3	4	5	6	7
0	d_0	0	0	0	0	0	1	0
1		d_1	0	0	0	0	1	0
2			d_2	0	1	1	1	0
3				d_3	1	1	1	0
4					d_4	0	1	0
5						d_5	1	0
6							d_6	0
7								d_7

$k = 3$

i \ j	0	1	2	3	4	5	6	7
0	d_0	0	0	1	0	0	1	0
1		d_1	0	1	0	0	1	0
2			d_2	1	1	1	1	0
3				$\boldsymbol{d_3}$	0	0	0	0
4					d_4	0	1	0
5						d_5	1	0
6							d_6	0
7								d_7

$k = 4$

i \ j	0	1	2	3	4	5	6	7
0	d_0	0	0	1	1	0	1	0
1		d_1	0	1	1	0	1	0
2			d_2	1	1	1	1	0
3				d_3	1	0	0	0
4					$\boldsymbol{d_4}$	0	0	0
5						d_5	1	0
6							d_6	0
7								d_7

Weitere Beispiele zur Berechnung von $\max_{\preccurlyeq t} K$

i \ j	0	1	2	3	4	5	6	7
0	d_0	1	1	1	1	1	1	1
1		d_1	1	1	0	0	1	0
2			d_2	0	0	0	1	0
3				d_3	0	0	1	0
4					d_4	1	1	1
5						d_5	1	1
6							d_6	0
7								d_7

i \ j	0	1	2	3	4	5	6	7
0	d_0	1	1	1	0	1	0	1
1		d_1	0	1	0	0	0	0
2			d_2	1	0	0	0	1
3				d_3	0	0	0	0
4					d_4	1	0	1
5						d_5	0	1
6							d_6	1
7								d_7

Zusammenfassung

Algorithmus	Eigenschaften
Windhund-Algorithmus	Schneidet im Vergleich zu den anderen Algorithmen mit am schlechtesten ab.
Zweite-Chance-Algorithmus	Deutliche Verbesserung gegenüber dem Windhund-Algorithmus, ist aber komplizierter zu implementieren.
Clock-Algorithmus	Sehr ähnliche Funktionsweise wie die des Zweite-Chance-Algorithmus mit einfacher Implementierung.
WS-Clock-Algorithmus	Bester Ersetzungsalgorithmus für Caches.
LFU-Algorithmus	Liefert gute Heuristik für das ERSETZUNGSPROBLEM, schneidet aber im **Realbetrieb** schlechter ab als der LRU-Algorithmus.
LRU-Algorithmus	Bester Ersetzungsalgorithmus für den TLB.