Aufgabe 2.0 & 2.2

T. Adam, M. ben Ahmed

Universität Osnabrück

Æ

December 6, 2020

Aufgabe 2.0 Lemma C

Lemma C

- Zu Zeigen: Nach N Operationen werden maximal $L \leq \log_\alpha(\frac{N}{B})$ Level benutzt.
- Im schlimmsten fall N insert Operationen
- Anzahl an Elemente bis Level j entspricht: $\sum_{i=1}^{j} |\mathcal{L}_i|$
- $N \leq \sum_{i=1}^{j} |\mathcal{L}_i| \leq \ell_{j+1} = B \cdot \alpha^{j+1}$

$$N \leq B \cdot \alpha^{j+1} \Rightarrow \frac{N}{B} \leq \alpha^{j+1} \Rightarrow \log_\alpha(\frac{N}{B}) \leq j + 1$$

- $N \leq \sum_{i=1}^{j} |\mathcal{L}_i|$ ist somit erfüllt $\forall j \geq \log_\alpha(\frac{N}{B})$
- $L \leq \log_\alpha(\frac{N}{B})$

Aufgabe 2.0 Lemma D

Lemma D

- $\texttt{store}(\texttt{i}, \texttt{S})$, $\texttt{compact}(\texttt{i})$ und $\texttt{merge}(\texttt{i} - 1, \texttt{S}, \texttt{S}')$ benötigen maximal $\frac{3\ell_i}{B}$ I/Os

store(i,S)

- $\texttt{store}(\texttt{i}, \texttt{S})$: Speichere S in einem freien Slot von $\mathcal{L}_i$ und verschiebe B Elemente nach H_2
- Über S iterieren (lesen und schreiben): $2\frac{\ell_i}{B}$ I/Os ($|S| < \ell_i$)
- B Elemente in H_2 speichern: 2 I/Os
- Mit $\ell_i > \ell_1 = cM > 3B$ gilt:
- $2\frac{\ell_i}{B} + 2 < 3\frac{\ell_i}{B}$

Aufgabe 2.0 Lemma D

compact(i)

- `compact(i)`: Verschmelzen von 2 kleinen Slots zu einem großen Slot auf Level i
- Mergen der kleinen Slots maximal: $\frac{\ell_i}{2B} + \frac{\ell_i}{2B} + \frac{\ell_i}{B} = 2\frac{\ell_i}{B}$ I/Os
- Blöcke in H_2 verschmelzen: 3 I/Os
- Mit $\ell_i > \ell_1 = cM > 3B$ gilt:
- $2\frac{\ell_i}{B} + 3 < 3\frac{\ell_i}{B}$

Aufgabe 2.0 Lemma D

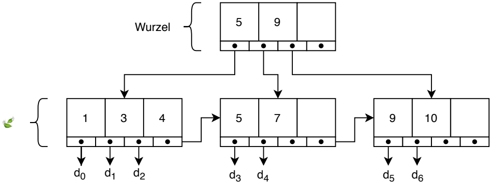
merge(i-1,S,S')

- $\texttt{merge}(\texttt{i}-1, \texttt{S}, \texttt{S}')$: Verschmelzen alle Slots aus $\mathcal{L}_{i-1}$ und S zu einer Folge S'
- μ Slots in $\mathcal{L}_{i-1}$ mit jeweils Größe l_{i-1} und $|S| \leq \ell_{i-1}$
- Jeder Block aus $\mathcal{L}_{i-1}$ und S wird einmal gelesen und geschrieben
- $2\mu\frac{\ell_{i-1}}{B} + 2\frac{\ell_{i-1}}{B} = 2\frac{\ell_i}{B} + 2\frac{\ell_{i-1}}{B} < 3\frac{\ell_i}{B}$ I/Os

Aufgabe 2.2 - Buffer Tree

Grundlagen

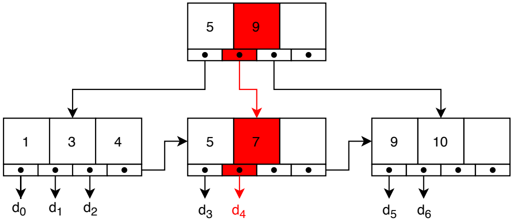
- basiert auf (a,b)-Baum
- $b > 2a$
- speichert Key-Data Paare
- Knoten u. Wurzel speichern Pivots
- Blätter enthalten die Daten
- Wurzel: $[2, b] \in \mathbb{N}$ Kinder
- Knoten: $[a, b] \in \mathbb{N}$ Kinder



Aufgabe 2.2 - Buffer Tree

Operationen

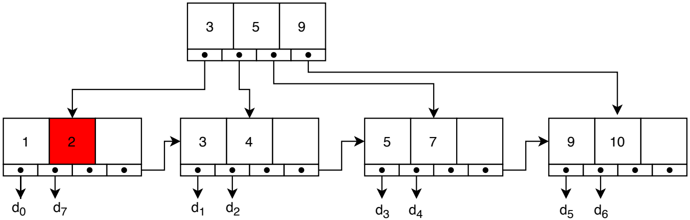
1. **search(7)**
2. insert(x)
3. delete(x)



Aufgabe 2.2 - Buffer Tree

Operationen

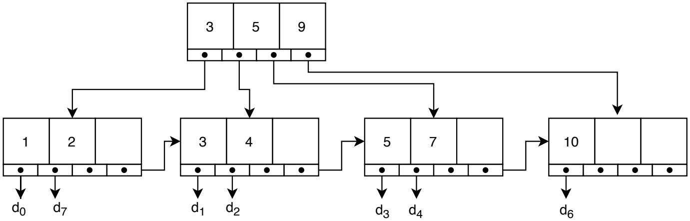
1. search(x)
2. **insert(2)**
3. delete(x)



Aufgabe 2.2 - Buffer Tree

Operationen

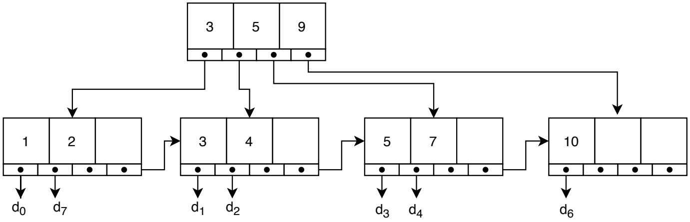
1. search(x)
2. insert(x)
3. **delete(9)**



Aufgabe 2.2 - Buffer Tree

Operationen

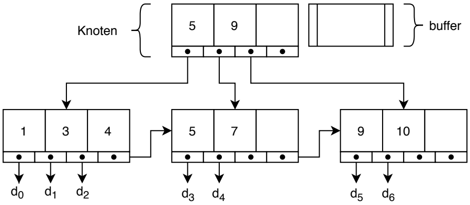
1. search(x)
2. insert(x)
3. **delete(9)**



Aufgabe 2.2 - Buffer Tree

Buffer Tree

- (a,b)-Baum mit $a = m/4$ und $b = m$
- $m :=$ größe des Buffers
- Tiefe $\mathcal{O}(\log_m(n))$
- n Blätter je $\Theta(B)$
- Speicher $\mathcal{O}(n)$
- jeder Knoten bekommt Buffer



Aufgabe 2.2 - Buffer Tree

Buffer Tree

- Operationen werden nicht sofort ausgeführt
- erzeugen Tupel [Element, Operation, Zeitstempel]
- wenn Anzahl Tupel $\geq B \rightarrow$ Tupel in Buffer der Wurzel
- wenn Anzahl Tupel in Buffer $> m \rightarrow$ starte Buffer-Entleerungsprozess

Aufgabe 2.2 - Buffer Tree

Buffer-Entleerungsprozess

- interne Knoten := Knoten die keine Blätter als Kinder haben
- Blattknoten := Knoten deren Kinder Blätter sind

Buffer-Entleerungsprozess: interne Knoten

- Lade jeweils M Tupel aus dem Buffer in den Hauptspeicher
- sortieren und mergen
- entferne Operationen die sich aufheben $\rightarrow$ insert(x), delete(x)
- verschiebe Tupel entsprechend ihrer Elemente in Buffer der Kinder
- starte Entleerungsprozess rekursiv für tiefere Buffer mit mehr als m Elementen

Aufgabe 2.2 - Buffer Tree

Buffer-Entleerungsprozess: Blattknoten

- Lade Tupel aus Blattknoten mit zugehörigen Blättern in den Hauptspeicher
- Führe Operationen gemäß der Tupel aus
- schreibe Änderungen in den (a,b)-Baum zurück

Aufgabe 2.2 - Buffer Tree

I/O Analyse

- Sequenz von N Operationen $\rightarrow \mathcal{O}(n \cdot \log_m n)$ I/Os
- Kosten Operation $\rightarrow \mathcal{O}((\log_m n)/B)$ I/Os amortisiert
- Buffer Tree benötigt $\mathcal{O}(n)$ Speicher
- ein Buffer Tree kann I/O optimal sortieren $\rightarrow \mathcal{O}(n \cdot \log_m n)$ I/Os