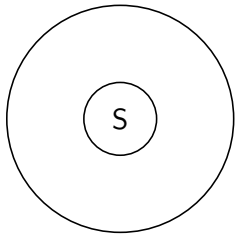
Paralles-Hashing

Sequential and Parallel Algorithms and Data Structures von Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, Roman Dementiev

Marco Bellmann

27. Juni 2023

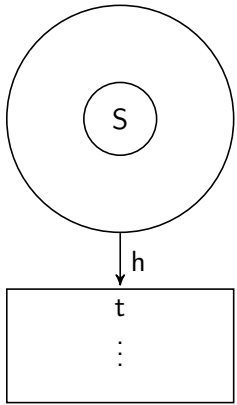
Lineares Hashing



Schlüsseluniversum mit Teilmenge S

Abbildung: MIT 6.006 Introduction to Algorithms, Fall 2011

Lineares Hashing



Schlüsseluniversum mit Teilmenge S

Hashfunktion

Hashtabelle

Abbildung: MIT 6.006 Introduction to Algorithms, Fall 2011

Datenstruktur

Hashtabelle

Datenstruktur

Hashtabelle

Funktionen	Sequentiel	
	Bedingung	Aktion
find(x)	$x = key(e)$	return e or return$\perp$
insert(e)	$e \notin t$	-
remove(x)	$e \in t$	-

Datenstruktur

Hashtabelle

Funktionen	Sequentiel Bedingung	Aktion
find(x)	$x = key(e)$	return e or return$\perp$
insert(e)	$e \notin t$	-
remove(x)	$e \in t$	-

$$Element\ e := (Key\ x, Value\ v)$$

Arten von Hashing

Offenes Hashing	Geschlossenes Hashing
Linear Probing	Verkettetes Hahing

Arten von Hashing

Offenes Hashing Linear Probing	Geschlossenes Hashing Verkettetes Hahing
Elemente können an einer anderen Adresse liegen	fixer Adressraum Erweiterung mit Liste

Arten von Hashing

Offenes Hashing Linear Probing	Geschlossenes Hashing Verkettetes Hahing
Elemente können an einer anderen Adresse liegen	fixer Adressraum Erweiterung mit Liste
Suche in ganzer Hashtabelle (Grafik)	Suche in Liste (Grafik)

Arten von Hashing

Offenes Hashing Linear Probing	Geschlossenes Hashing Verkettetes Hahing
Elemente können an einer anderen Adresse liegen	fixer Adressraum Erweiterung mit Liste
Suche in ganzer Hashtabelle (Grafik)	Suche in Liste (Grafik)

Suche: $O(n)$

Parallel vs Sequentiell

Funktionen	Sequentiel		Parralell	
	Bedingung	Aktion	Bedingung	Aktion
find(x)	$x = key(e)$	return e or $\perp$	$x = key(e)$	return e.copy() or $\perp$
insert(e)	$e \notin t$	-	$e \notin t$	-
remove(x)	$e \in t$	-	$e \in t$	-

Parallel vs Sequentiell

Funktionen	Sequentiel		Parralell	
	Bedingung	Aktion	Bedingung	Aktion
find(x)	$x = key(e)$	return e or $\perp$	$x = key(e)$	return e.copy() or $\perp$
insert(e)	$e \notin t$	-	$e \notin t$	-
remove(x)	$e \in t$	-	$e \in t$	-
update(x,v,f)	-	-	$(x, v') \in t$	$f(v', v)$
insertOrUpdate(x,v,f)	-	-	$e \notin t$	$insert(x)$ or $f(v', v)$

Element $e := (Key\ x, Value\ v)$

Parallel vs Sequentiell

Funktionen	Sequentiel		Parralell	
	Bedingung	Aktion	Bedingung	Aktion
find(x)	$x = key(e)$	return e or $\perp$	$x = key(e)$	return e.copy() or $\perp$
insert(e)	$e \notin t$	-	$e \notin t$	-
remove(x)	$e \in t$	-	$e \in t$	-
update(x,v,f)	-	-	$(x, v') \in t$	$f(v', v)$
insertOrUpdate(x,v,f)	-	-	$e \notin t$	$insert(x)$ or $f(v', v)$

Element $e := (Key\ x, Value\ v)$

Achtung:
remove und insert nur einmal

Distributed-Memory Hashing

Simuliere mehrere Tabellen

Distributed-Memory Hashing

Simuliere mehrere Tabellen

PEs Adressraum:

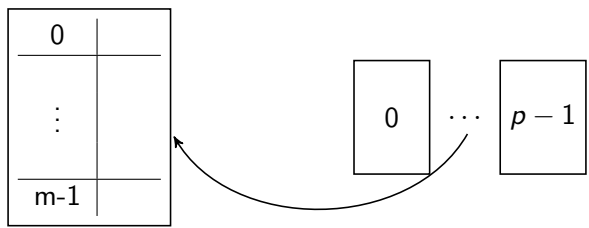
für ein $i \in 0 \ldots p-1$

Distributed-Memory Hashing

Simuliere mehrere Tabellen

PEs Adressraum:

für ein $i \in 0 \ldots p-1$

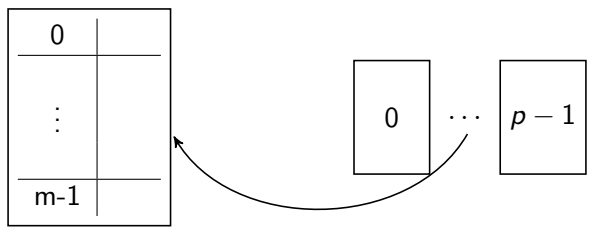


Distributed-Memory Hashing

Simuliere mehrere Tabellen

PEs Adressraum:

für ein $i \in 0 \ldots p-1$



$i * m/p \ldots (i+1) * m/p$

Beispiel

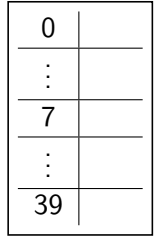
$$i * m/p \ldots (i+1) * m/p$$
$$m = 40,\ p = 5$$

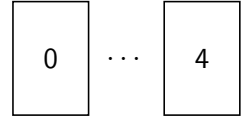
Beispiel

$$i * m/p \ldots (i+1) * m/p$$
$$m = 40,\ p = 5$$
$$\Rightarrow \text{für } i = 0, \text{ Adressraum von 0 bis 8}$$

Beispiel

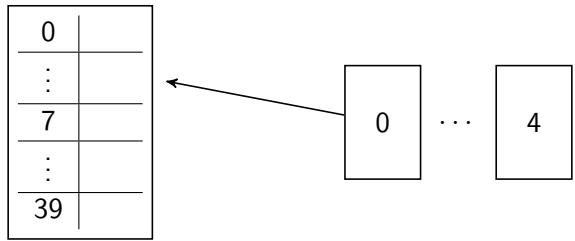
$$i * m/p \ldots (i+1) * m/p$$
$$m = 40,\ p = 5$$
$$\Rightarrow \text{für } i = 0, \text{ Adressraum von 0 bis 8}$$





Beispiel

$$i * m/p \ldots (i+1) * m/p$$
$$m = 40,\ p = 5$$
$$\Rightarrow \text{für } i = 0, \text{ Adressraum von 0 bis 8}$$



Distributed-Memory Hashing

Funktionen	Spezifizierung

Distributed-Memory Hashing

Funktionen	Spezifizierung
find()	

Distributed-Memory Hashing

Funktionen	Spezifizierung
find()	
insert()	

Distributed-Memory Hashing

Funktionen	Spezifizierung
find()	
insert()	
remove()	

Distributed-Memory Hashing

Funktionen	Spezifizierung
find()	
insert()	
remove()	
insertOrUpdate()	

Distributed-Memory Hashing

Performance:

- Sei $o = \Omega(p \log p)$ und o *truly random*

Distributed-Memory Hashing

Performance:

- Sei $o = \Omega(p \log p)$ und o *truly random*
- Sei $T_{all \to all}(x)$ *all-to-all* Datenaustausch und x maximale Nachrichtengöße

Distributed-Memory Hashing

Performance:

- Sei $o = \Omega(p \log p)$ und o *truly random*
- Sei $T_{all \to all}(x)$ *all-to-all* Datenaustausch und x maximale Nachrichtengöße
- (Grafik)

Distributed-Memory Hashing

Performance:

- Sei $o = \Omega(p \log p)$ und o *truly random*
- Sei $T_{all \to all}(x)$ *all-to-all* Datenaustausch und x maximale Nachrichtengöße
- (Grafik)
- Zeige: max Nachrichtengröße $= O(o/p)$

Distributed-Memory Hashing

Performance:

- Sei $o = \Omega(p \log p)$ und o *truly random*
- Sei $T_{all \to all}(x)$ *all-to-all* Datenaustausch und x maximale Nachrichtengöße
- (Grafik)
- Zeige: max Nachrichtengröße $= O(o/p)$
- $O(T_{all \to all}(\log p)) = O(T_{all \to all}(o/p))$

Shared-Memory Hashing

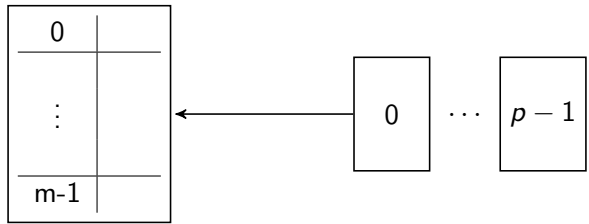
- Linear probing

Shared-Memory Hashing

- Linear probing
- Offenenes Hashing

Shared-Memory Hashing

- Linear probing
- Offenenes Hashing



Leichte Änderungen im Vergleich zum Distributet-Hashing

Leichte Änderungen im Vergleich zum Distributet-Hashing

Funktionen	Veränderung

Leichte Änderungen im Vergleich zum Distributet-Hashing

Funktionen	Veränderung
find()	gesamter Adressraum

Leichte Änderungen im Vergleich zum Distributet-Hashing

Funktionen	Veränderung
find()	gesamter Adressraum
insert()	nur elementare Operationen, nächsten freien Platz automatisch finden

Leichte Änderungen im Vergleich zum Distributet-Hashing

Funktionen	Veränderung
find()	gesamter Adressraum
insert()	nur elementare Operationen, nächsten freien Platz automatisch finden
remove()	markiere zu löschendes Element scanne die gesamte Tabelle um echt zu löschen

Leichte Änderungen im Vergleich zum Distributet-Hashing

Funktionen	Veränderung
find()	gesamter Adressraum
insert()	nur elementare Operationen, nächsten freien Platz automatisch finden
remove()	markiere zu löschendes Element scanne die gesamte Tabelle um echt zu löschen
insertOrUpdate()	nur elementare Operationen

Leichte Änderungen im Vergleich zum Distributet-Hashing

Funktionen	Veränderung
find()	gesamter Adressraum
insert()	nur elementare Operationen, nächsten freien Platz automatisch finden
remove()	markiere zu löschendes Element scanne die gesamte Tabelle um echt zu löschen
insertOrUpdate()	nur elementare Operationen

Skalierbarkeit der Tabelle und Löschen nicht effizeint

Fazit

- linear gut für kleine Mengen

Fazit

- linear gut für kleine Mengen
- distributed meisten schlechter

Fazit

- linear gut für kleine Mengen
- distributed meisten schlechter
- distributed schneller bei vielen Zugriffen auf einen Key

Fazit

- linear gut für kleine Mengen
- distributed meisten schlechter
- distributed schneller bei vielen Zugriffen auf einen Key
- häufigster Zugriff: **update**

Fazit

- linear gut für kleine Mengen
- distributed meisten schlechter
- distributed schneller bei vielen Zugriffen auf einen Key
- häufigster Zugriff: **update**
- shared schnell, da Operationen zusammen gefasst werden