Algorithmen und Datenstrukturen (Th. Ottmann und P. Widmayer)

Folien: Binomial Queues

Autor: Sven Schuierer

Institut für Informatik
Georges-Köhler-Allee
Albert-Ludwigs-Universität Freiburg

1 Vorrangswarteschlangen

Operationen:

$Q.initialize()$: erstelle die leere Menge Q

$Q.isEmpty()$: true gdw. Q ist leer

$Q.insert(e)$: füge Eintrag e in Q ein und gebe einen Zeiger auf den Knoten, der Eintrag e enthält, zurück

$Q.deletemin()$: liefere den Eintrag aus Q mit minimalen Schlüssel und entferne ihn

$Q.min()$: liefere den Eintrag aus Q mit minimalen Schlüssel

$Q.decreasekey(v, k)$: verringere den Schlüssel von Knoten v auf k

Operationen

Zusätzliche Operationen:

$Q.delete(v)$: entferne Knoten v mit Eintrag aus Q (ohne v zu suchen)

$Q.meld(Q')$: vereinige Q und Q' (concatenable queue)

$Q.search(k)$: suche den Eintrag mit Schlüssel k in Q (searchable queue)

u.v.a., z.B. $predecessor$, $successor$, max, $deletemax$

Implementationen

	Liste	Heap	Bin.-Q.	Fib.-Hp.
insert	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
delete-min	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
meld $(m \leq n)$	$O(1)$	$O(n)$ od. $O(m \log n)$	$O(\log n)$	$O(1)$
decr.-key	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)^*$

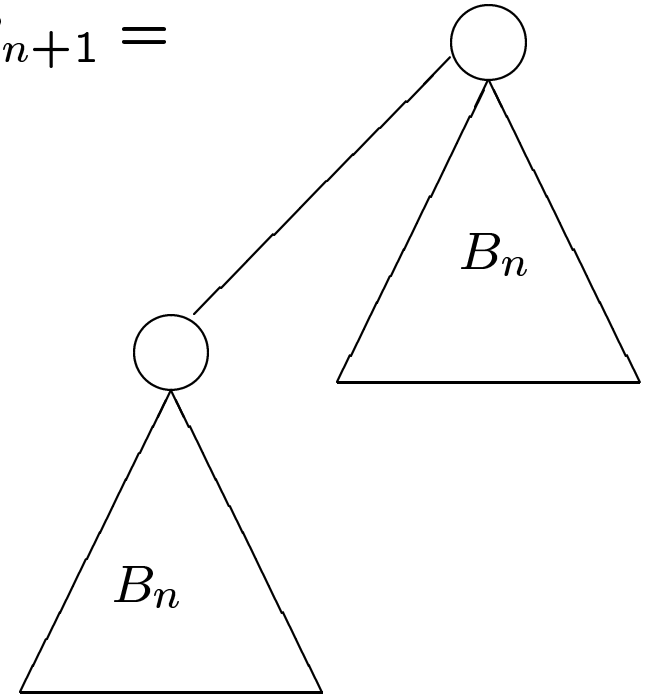
$*$ = amortisierte Zeit

$delete(e, Q) =$
$decreasekey(e, -\infty, Q) + deletemin(Q)$

2 Binomialbäume und -queues

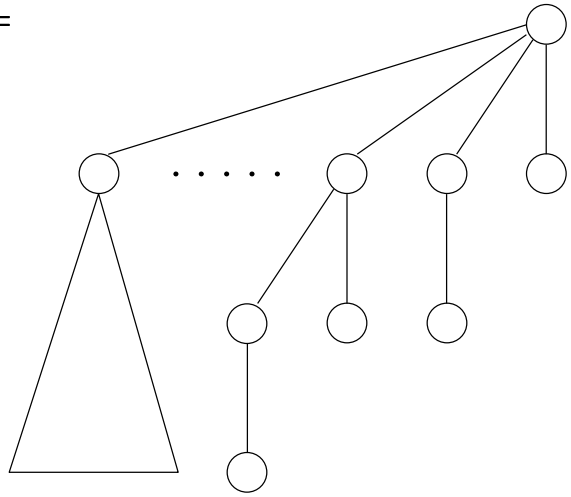
Definition Binomialbaum B_n, $n \geq 0$

$B_0 =$ 

$B_{n+1} =$ 

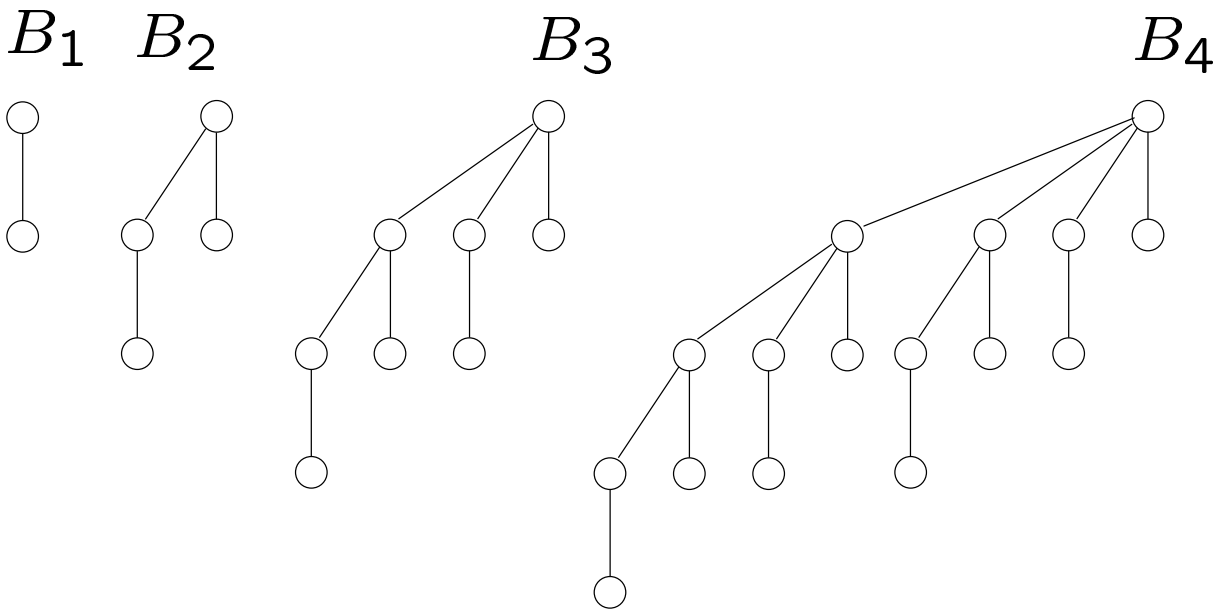
Folgerungen

1. B_n hat 2^n Knoten
2. B_n hat Höhe n
3. Wurzel hat Grad n (= Ordnung)
4. $B_n =$



5. Es gibt genau $\binom{n}{i}$ Knoten mit Tiefe i in B_n.

Binomial Bäume



3 Binomialqueues

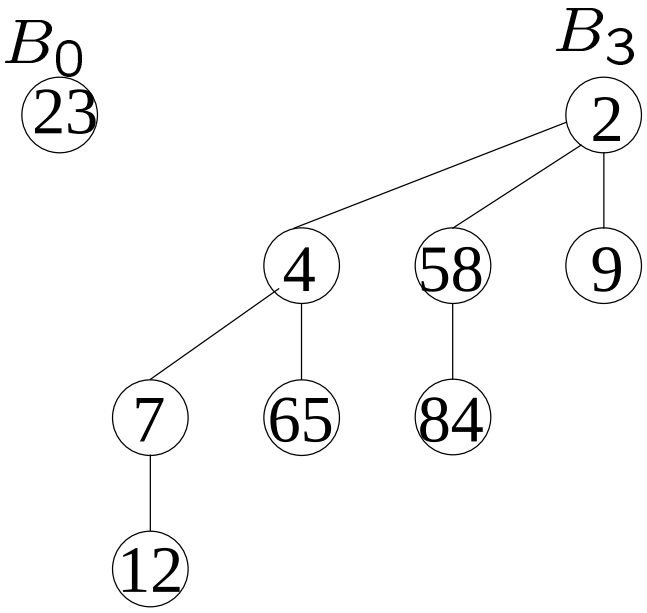
Binomialqueue Q: Vereinigung heapgeordneter Binomialbäume verschiedener Ordnung

n Schlüssel:

$B_i \in Q \Leftrightarrow i$-tes Bit in $(n)_2 = 1$

9 Schlüssel: $\{2, 4, 7, 9, 12, 23, 58, 65, 84\}$

$9 = (1001)_2$.



Min bestimmen = $O(\log n)$

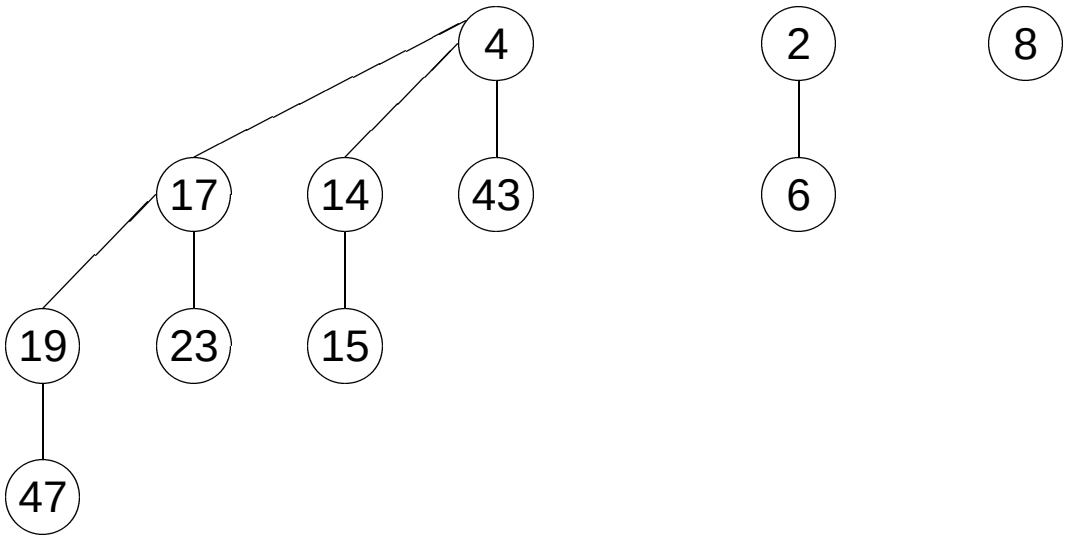
Beispiel

11 Schlüssel: $\{2, 4, 6, 8, 14, 15, 17, 19, 23, 43, 47\}$
(entspricht nicht der Einfügereihenfolge).

$11 = (1011)_2 \Rightarrow$ 3 Binomialbäume

B_3, B_1 und B_0

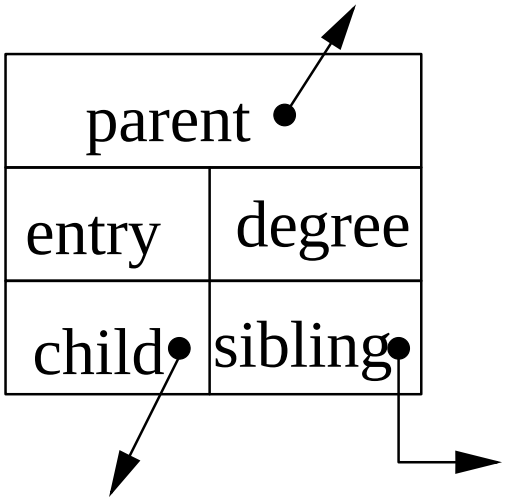
F_{11} :

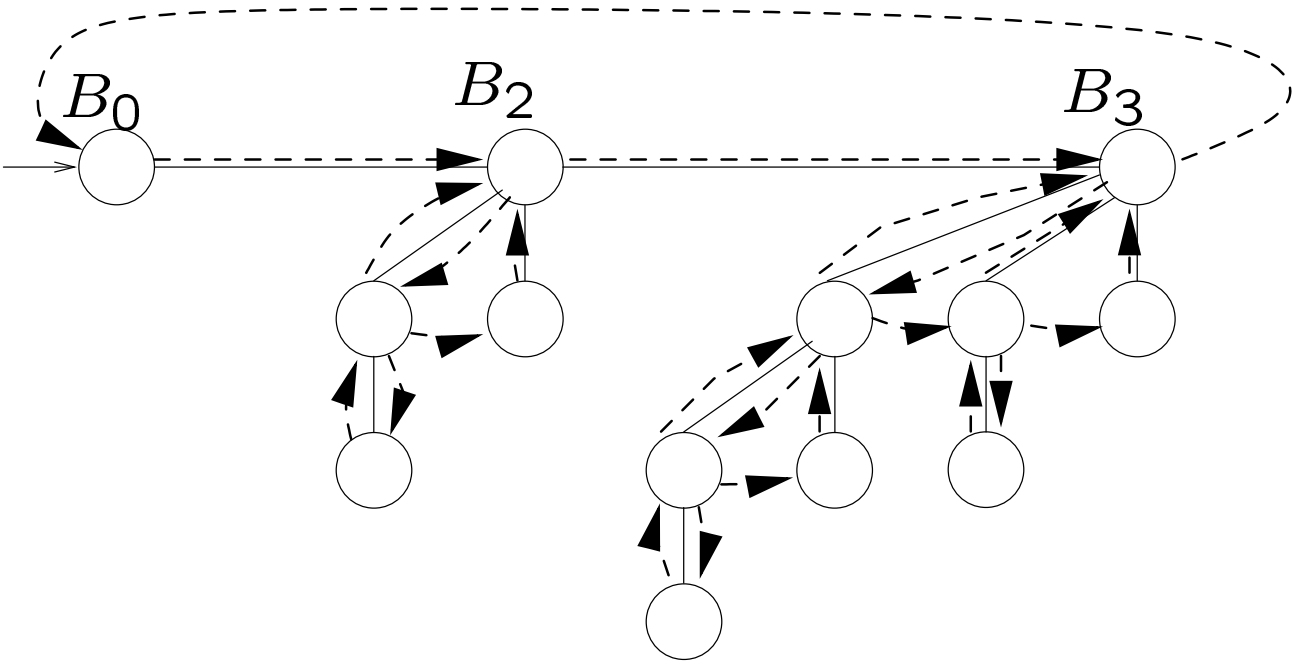


Implementation

Child-sibling Darstellung

Knotenformat:





Vereinigung

Vereinigung zweier Binomialbäume B, B' von gleicher Ordnung

Link-Operation:

B.Link(B')

1 if $B.key > B'.key$

2 then $B'.Link(B)$

$/* B.key \leq B'.key */$

3 $B'.parent := B$

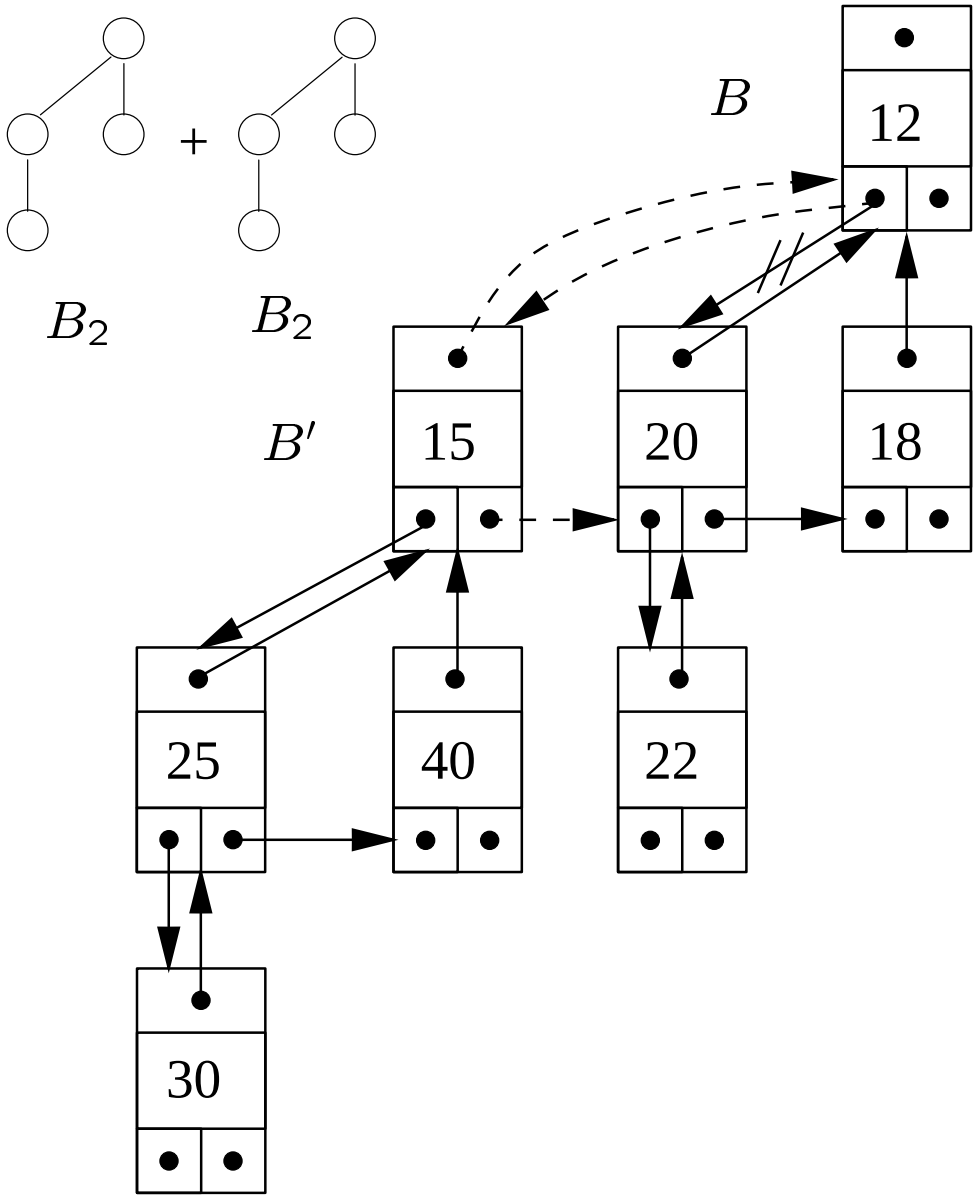
4 $B'.sibling := B.child$

5 $B.child := B'$

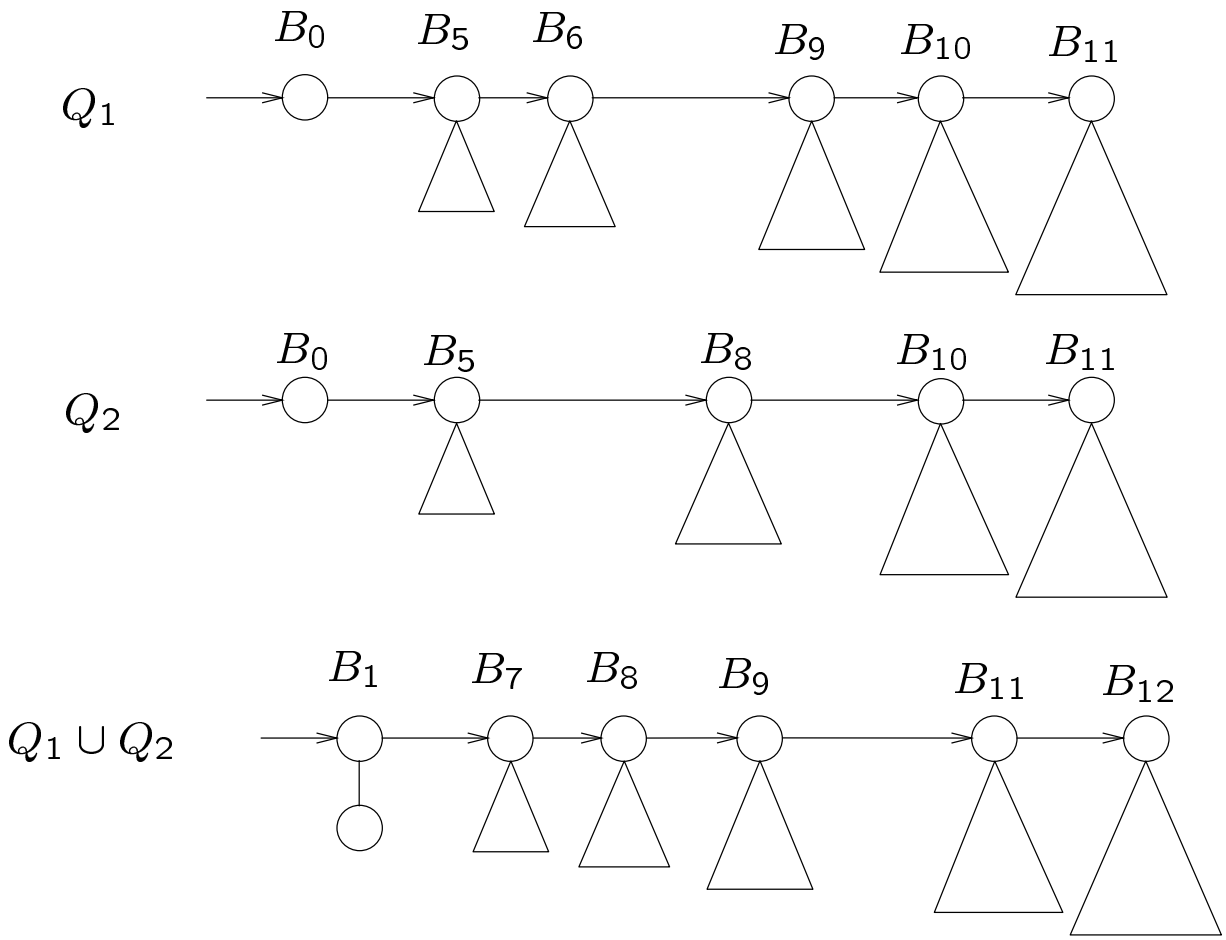
6 $B.degree := B.degree + 1$

Konstante Zeit

Link



Vereinigung (Meld)



Zeit: $O(\log n)$

(Wie Dualzahladdition: paralleler Scan durch Wurzellisten von Q und Q' für zwei B_i: Link, ggf. Übertrag beachten)

Operationen

$Q.initialize$: $Q.root = null$

$Q.insert(e)$: new B_0; $B_0.entry := e$; $Q.meld(B_0)$

Zeit: $O(\log n)$

$Q.deletemin()$:

1. bestimme B_i mit minimalem Schlüssel in der Wurzelliste und entferne B_i aus Q
2. drehe die Reihenfolge der Söhne von B_i um, also zu $B_0, B_1, \ldots, B_{i-1} \Rightarrow Q'$
3. $Q.meld(Q')$

Zeit: $O(\log n)$

$Q.decreasekey(v, k)$:

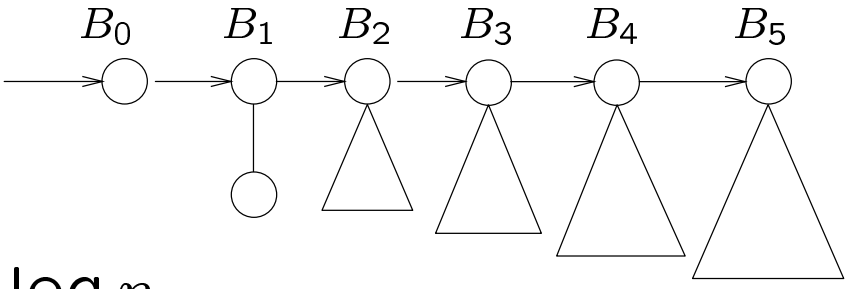
1. $v.entry.key := k$
2. $v.entry$ nach oben steigen lassen in dem geg. Baum, bis die Heapbedingung erfüllt ist

Zeit: $O(\log n)$

Worst-Case Folge von Operationen

Q:

$deletemin(Q)$:



Zeit: $\log n$

$insert(e, Q)$:

Zeit: $\log n$

$\Phi(Q)$ = # Bäume in Q