Binomial Queues

Prof. Dr. S. Albers

Vorrangswarteschlangen: Operationen

(Vorrangswarte)schlange (queue) *Q*

Struktur zur Speicherung von Elementen, für die eine Prioritätsordnung definiert ist, und für die folgende Operationen ausführbar sind:

Operationen:

Q.initialize(): erstellt die leere Schlange *Q*

Q.isEmpty(): liefert true gdw. *Q* ist leer

Q.insert(e): fügt Eintrag *e* in *Q* ein und gibt einen Zeiger auf den Knoten, der Eintrag *e* enthält, zurück

Q.deletemin(): liefert den Eintrag aus *Q* mit minimalen Schlüssel und entfernt ihn

Q.min(): liefert den Eintrag aus *Q* mit minimalen Schlüssel

Q.decreasekey(v,k): verringert den Schlüssel von Knoten *v* auf *k*

Vorrangswarteschlangen:Operationen

Zusätzliche Operationen:

Q.delete(v): entfernt Knoten *v* mit Eintrag aus *Q* (ohne *v* zu suchen)

Q.meld(Q´): vereinigt *Q* und *Q´* (concatenable queue)

Q.search(k): sucht den Eintrag mit Schlüssel *k* in *Q* (searchable queue)

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

Vorrangswarteschlangen Implementationen

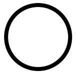
	Liste	Heap	Bin. – Q.	Fib.-Hp.
insert	O(1)	O(log n)	O(log n)	O(1)
min	O(n)	O(1)	O(log n)	O(1)
delete-min	O(n)	O(log n)	O(log n)	O(log n)*
meld ($m \leq n$)	O(1)	O(n) od. O(m log n)	O(log n)	O(1)
decr.-key	O(1)	O(log n)	O(log n)	O(1)*

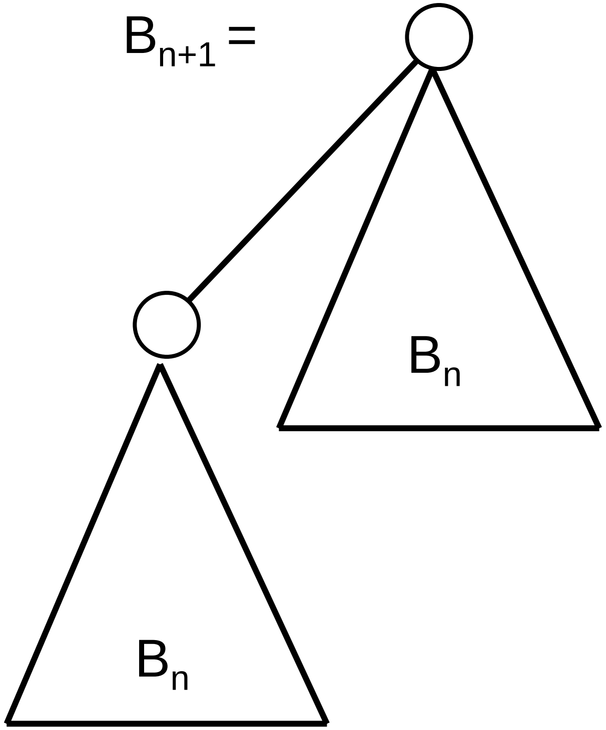
* = amortisierte Kosten

delete(e,Q) = decreasekey(e, ∞, Q) + deletemin(Q)

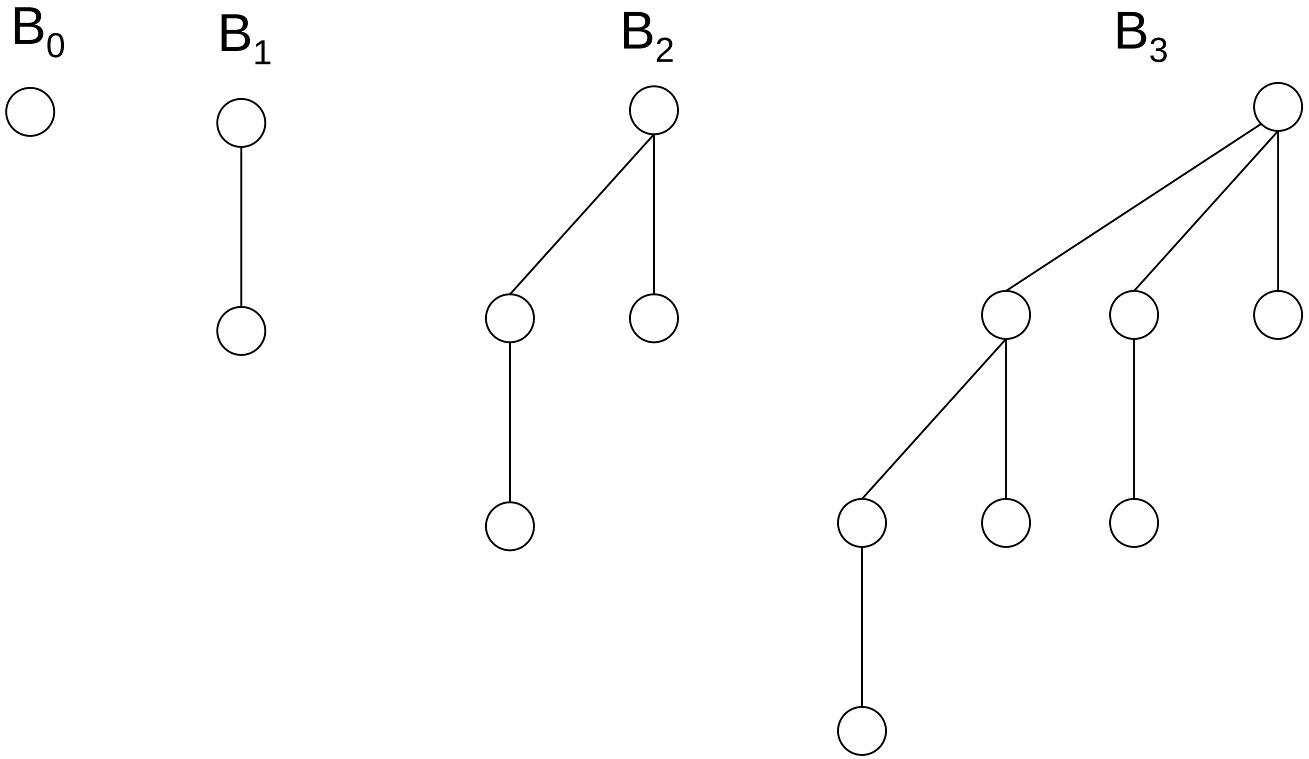
Definition

n-ter Binomialbaum B_n, $n \geq 0$

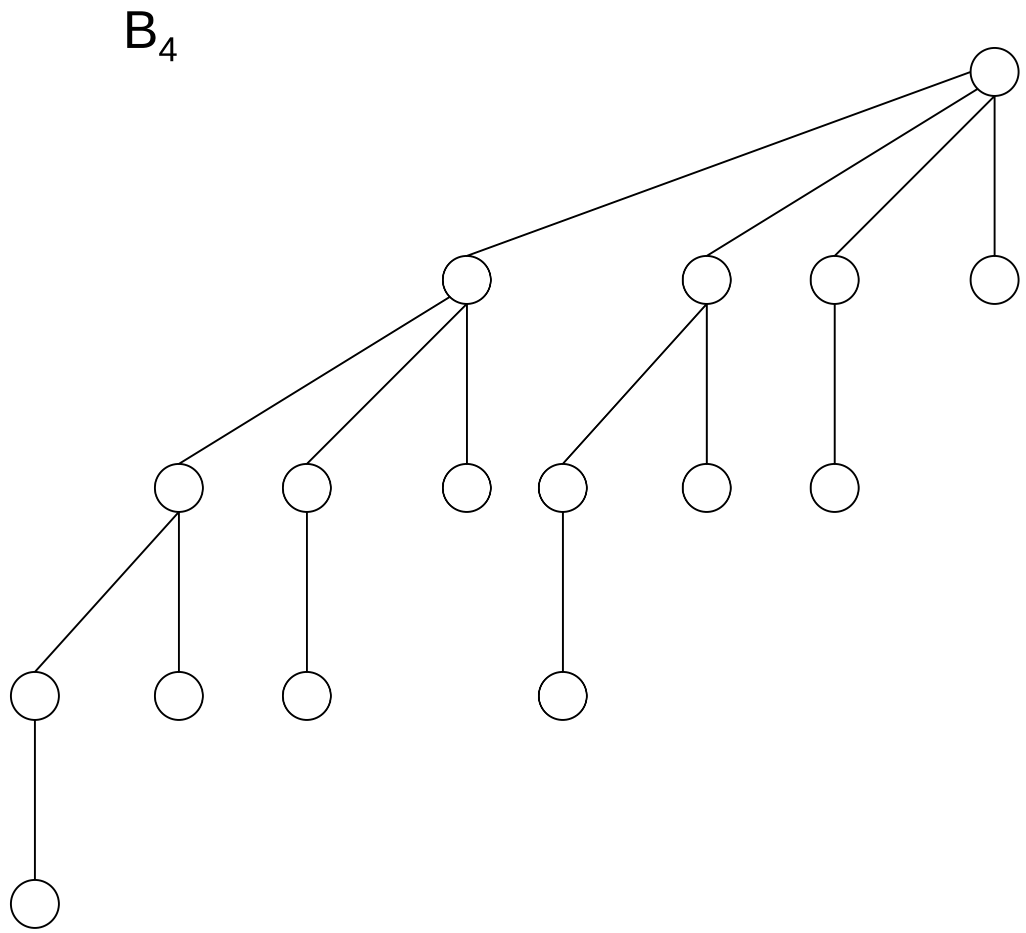
$B_0 =$ 

$B_{n+1} =$ 

Binomial Bäume

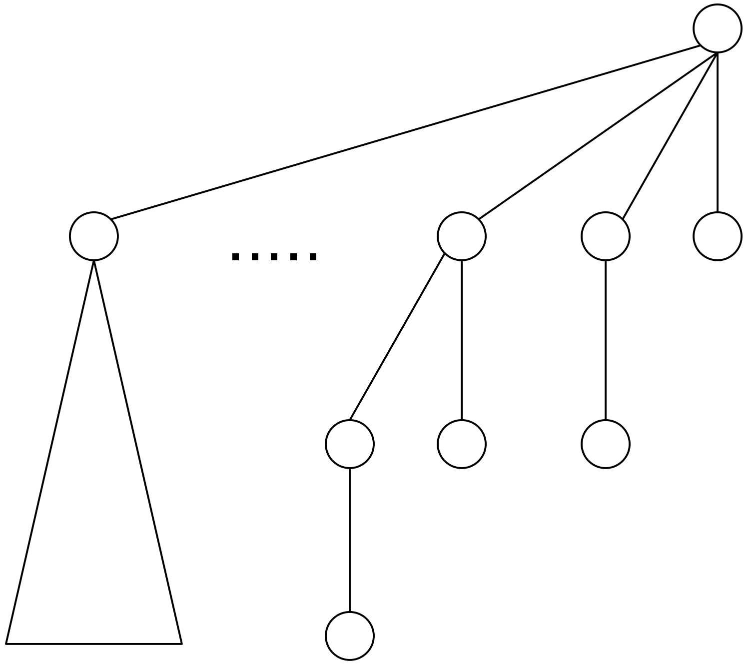


Binomial Bäume



Folgerung

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



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

Exkurs: Binomialkoeffizienten

$\binom{n}{i}$ = # Möglichkeiten, i aus n Objekten zu wählen

Pascal‘sches Dreieck:

				1				
			1		1			
		1		2		1		
	1		3		3		1	
1		4		6		4		1

Anzahl Knoten mit Tiefe i in B_n

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

Binomial Queues

Binomialqueue *Q*:

Vereinigung heapgeordneter Binomialbäume verschiedener Ordnung zur Speicherung von Schlüsseln

***n* Schlüssel:**

$B_i \in Q \iff i\text{-tes Bit in } (n)_2 = 1$

9 Schlüssel:

{2, 4, 7, 9,12, 23, 58, 65, 85}

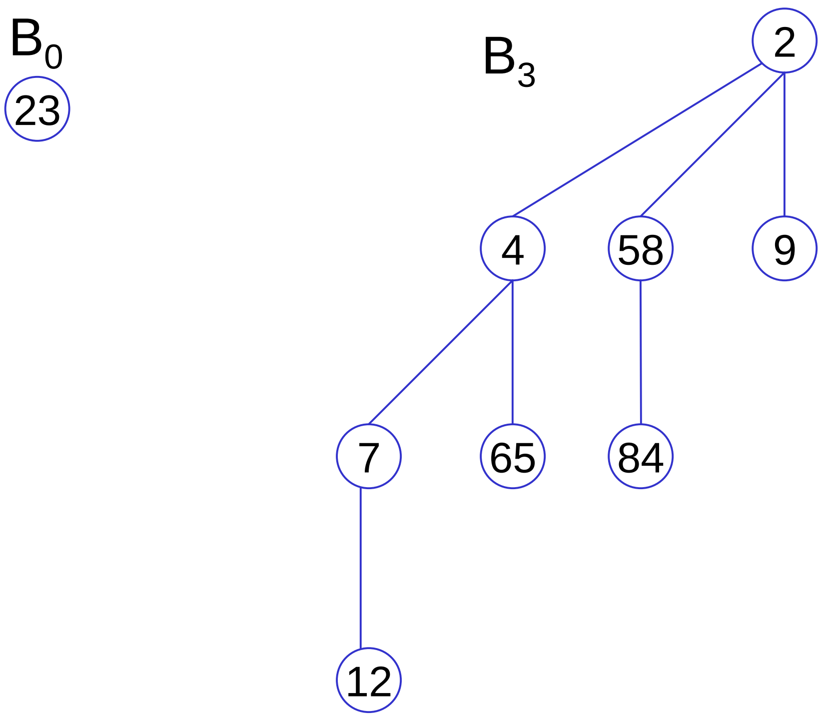
$9 = (1001)_2$

Binomial Queues: Beispiel 1

9 Schlüssel:

{2, 4, 7, 9,12, 23, 58, 65, 85}

$9 = (1001)_2$



Min bestimmen in Zeit: O(log *n*)

Binomial Queues: Beispiel 2

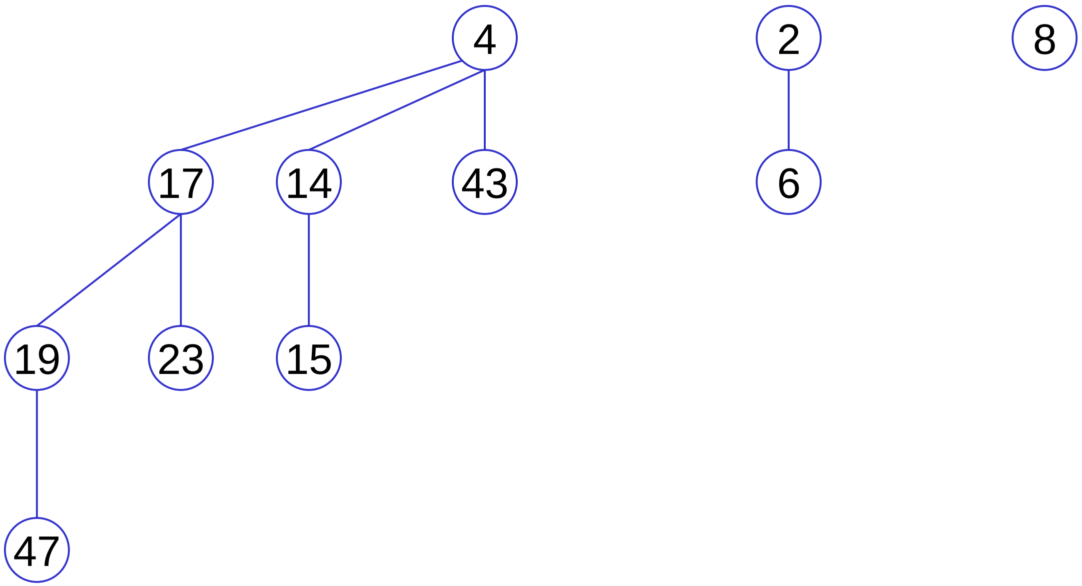
11 Schlüssel:

{2, 4, 6, 8, 14, 15, 17, 19, 23, 43, 47}

11 = $(1011)_2$ → 3 Binomialbäume

B_3, B_1, und B_0

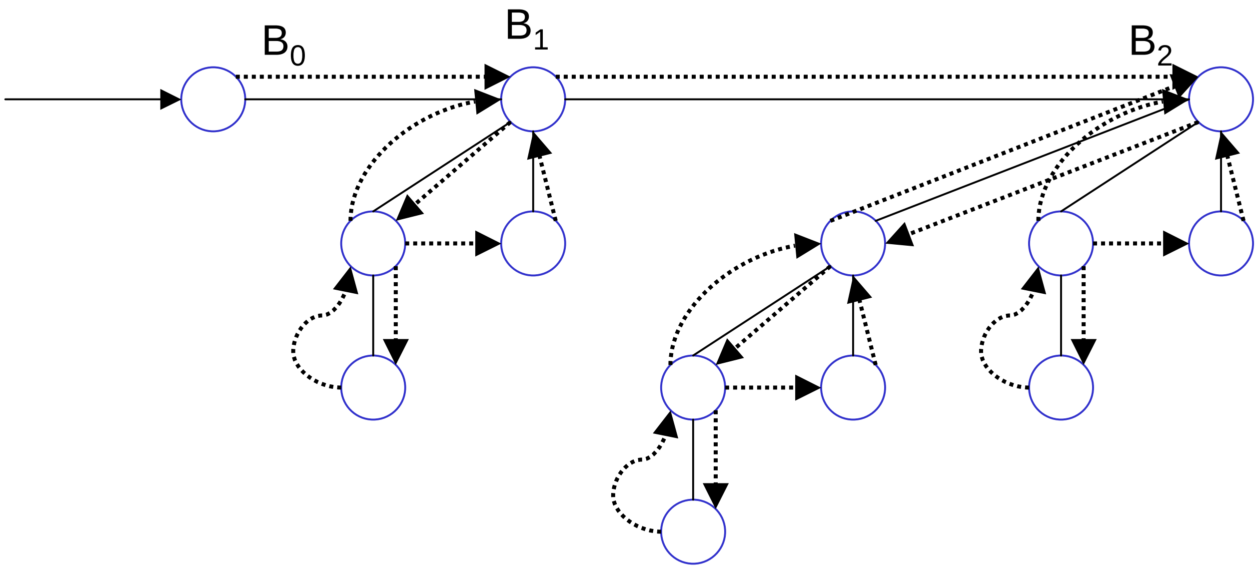
Q_{11}:



Child - Sibling Darstellung

Knotenformat:

parent	
entry	degree
child	sibling



Binomialbäume: Vereinigung (Link)

Vereinigung zweier Binomialbäume *B, B´* von gleicher Ordnung

$$B_n + B_n \rightarrow B_{n+1}$$

Link-Operation:

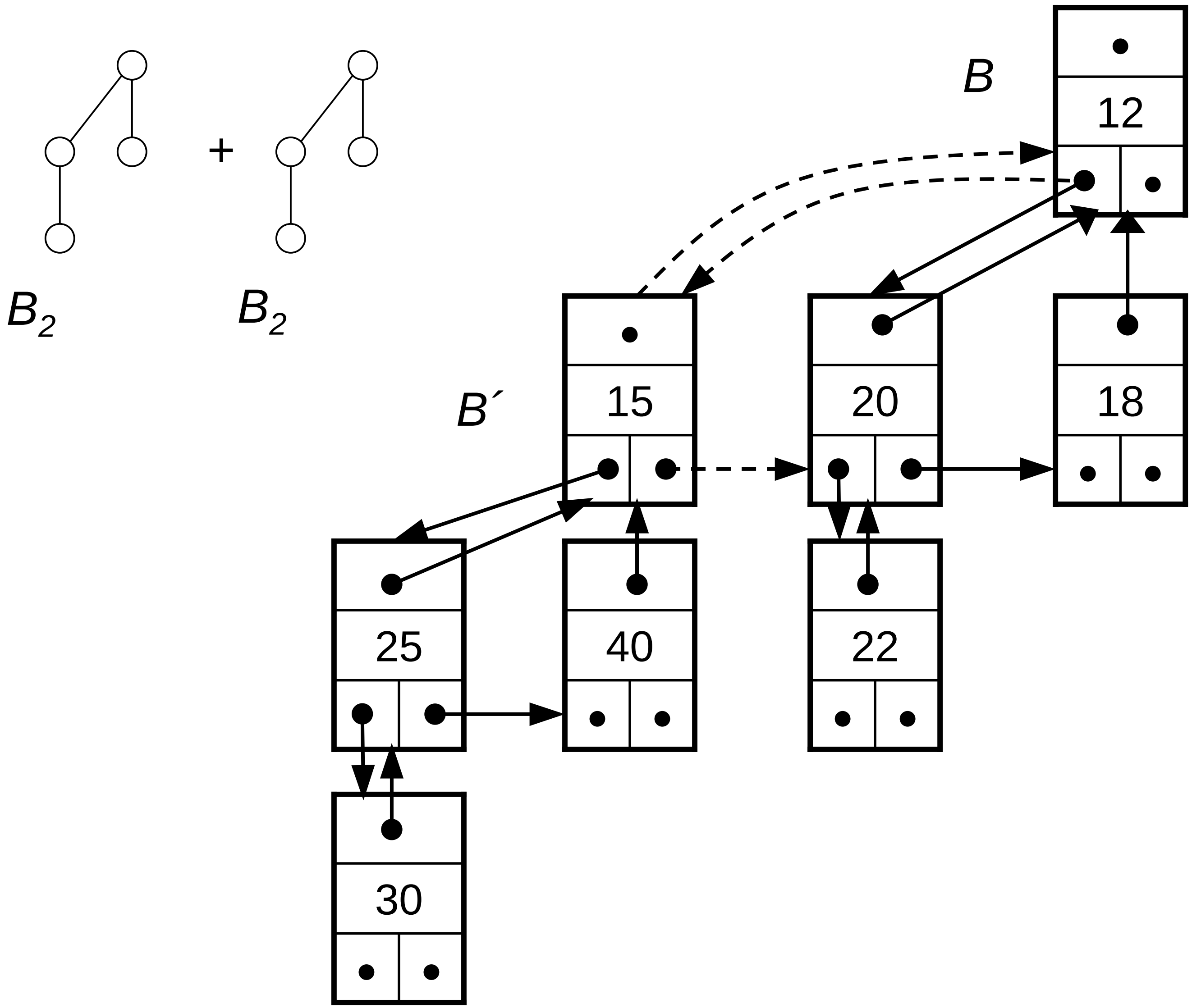
B.Link(B´)

/*Mache Wurzel des Baumes mit größerem Schlüssel zum Sohn des Baumes mit kleinerem Schlüssel */

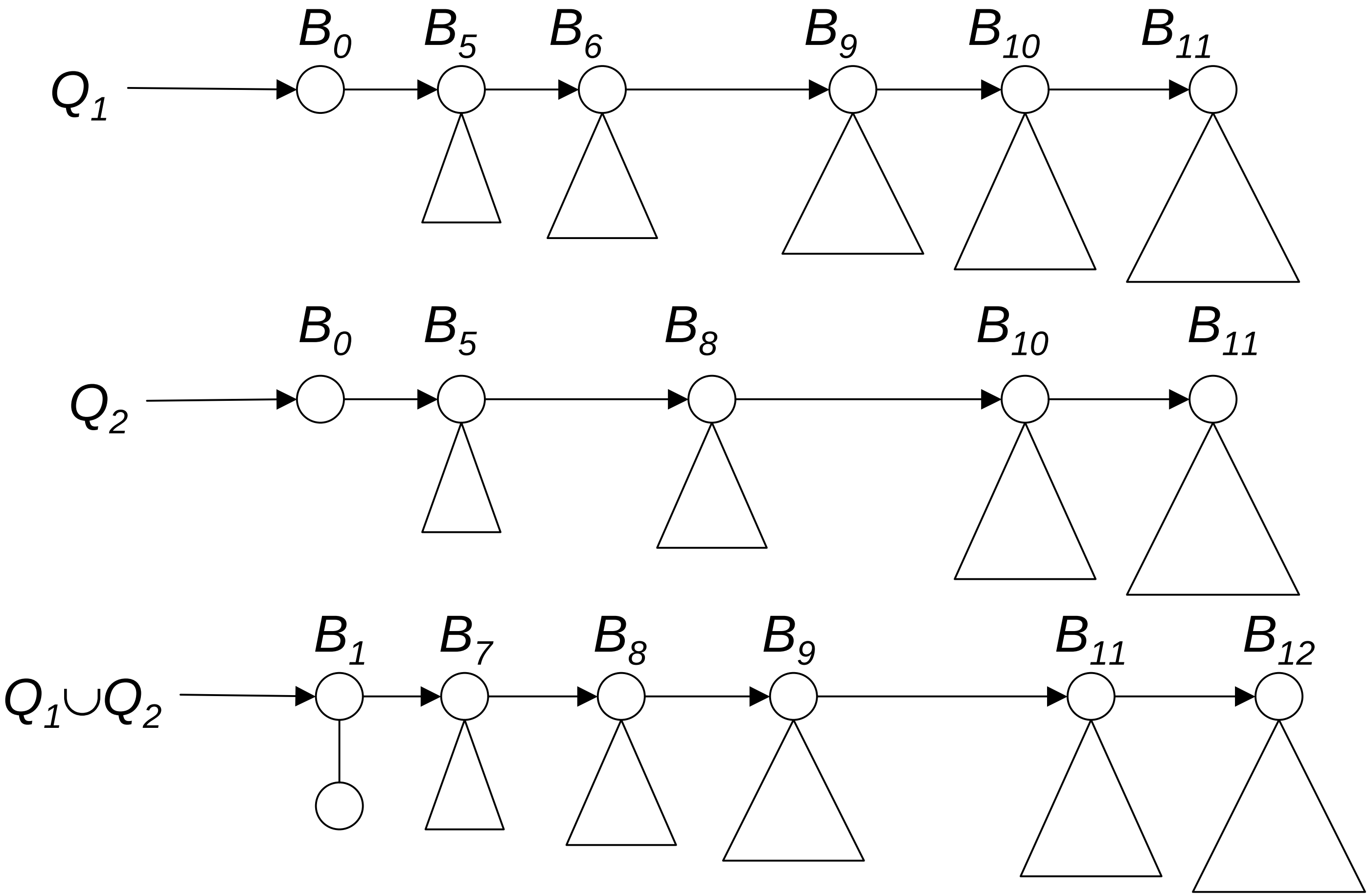
```
if B.key >  B´.key
   then B´.Link(B)
   return
  /* B.key ≤ B´.key*/
B´.parent = B
B´.sibling = B.child
B.child = B´
B.degree = B.degree +1
```

konstante Zeit

Beispiel zur Link-Operation



Binomial Queues: Vereinigung (Meld)



Zeit: O (log *n*)

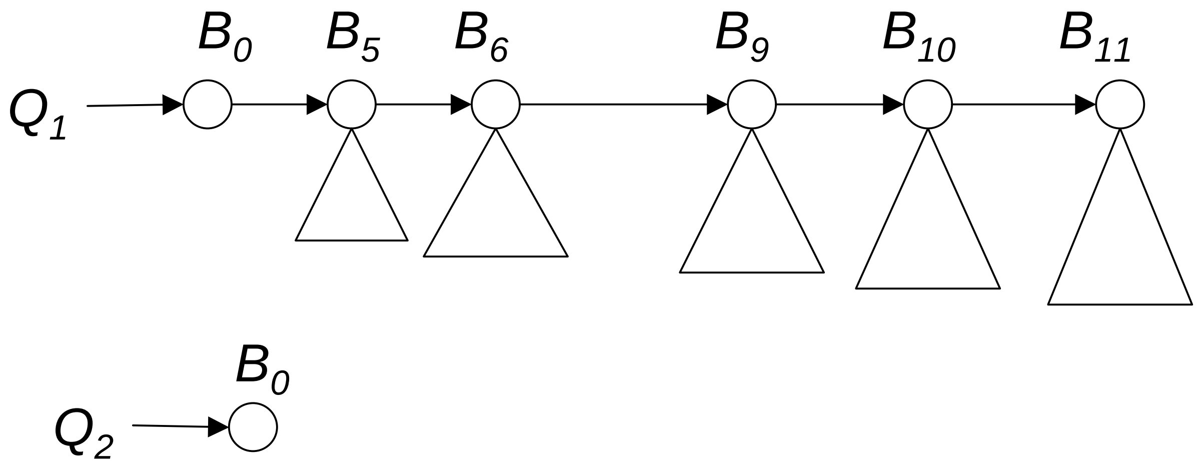
Binomial Queues: Operationen

Q.initialize:

Q.root = null

Q.insert(e):

new B_0
B_0.*entry* = e
Q.*meld*(B_0)



Zeit = O(log *n*)

Binomial Queues: Deletemin

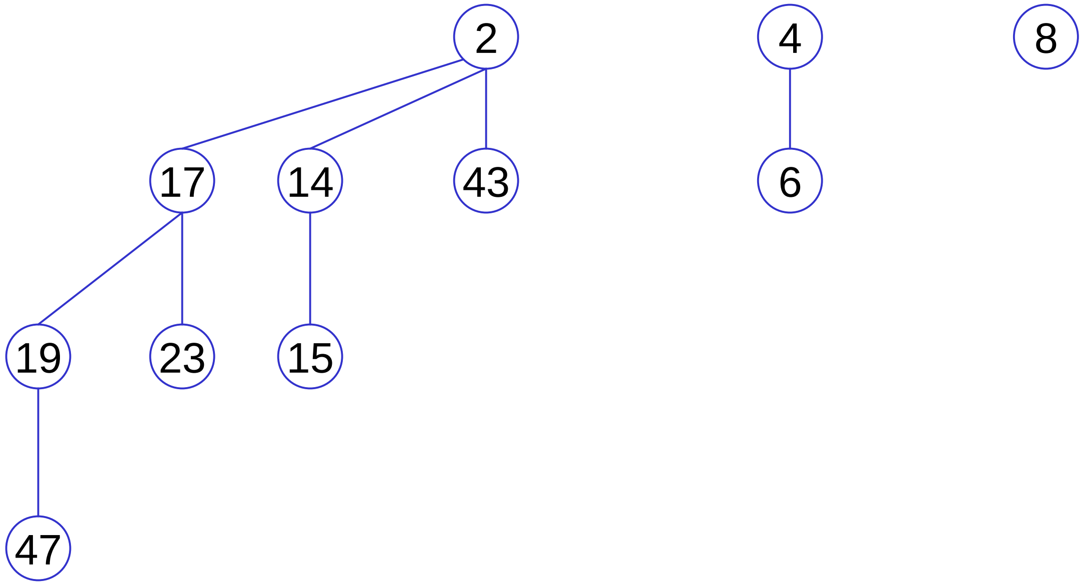
Q.deletemin():

1. Bestimme B_i mit minimalen Schlüssel in der Wurzelliste und entferne B_i aus Q (liefert Q´)
2. Drehe die Reihenfolge der Söhne von B_i um, also zu B_0 , B_1 , ... , B_{i-1} → *Q´´*
3. *Q´.meld(Q´´)*

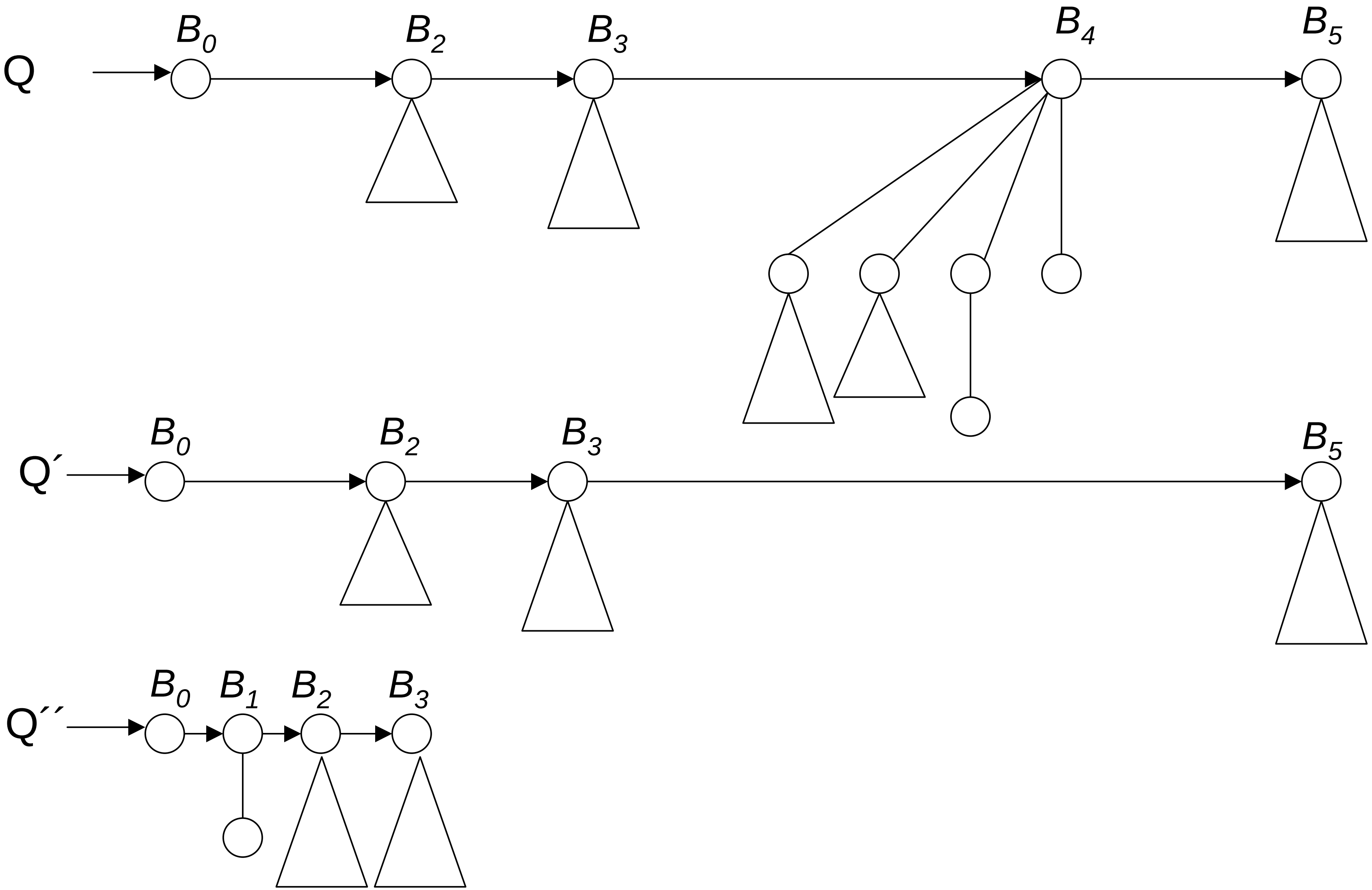
Zeit: O(log *n*)

Binomial Queues: Deletemin Beispiel 1

Q_{11}:



Binomial Queues: Deletemin Beispiel 2

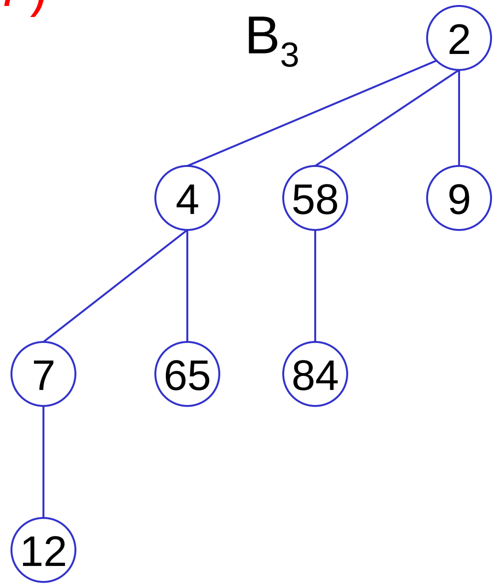


Binomialqueues: Decreasekey

Q.decreasekey(v, k):

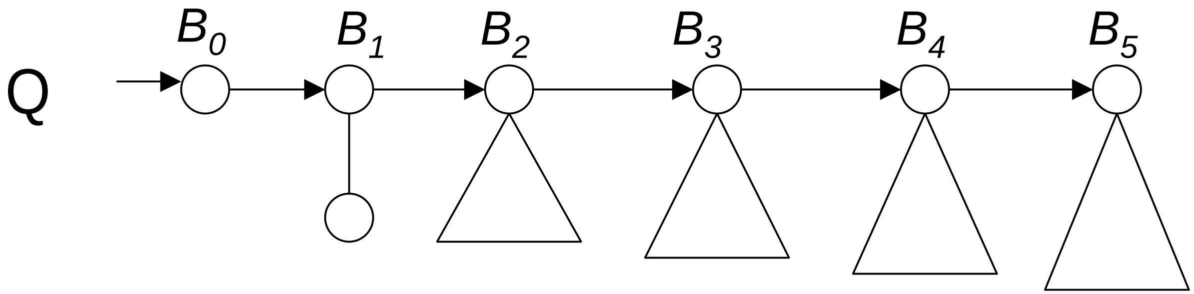
1. *v.entry.key := k*
2. *v.entry* nach oben steigen lassen in dem geg. Baum, bis die Heapbedingung erfüllt ist.

Zeit: O (log *n*)



Binomial Queues: Worst Case Folge von Operationen

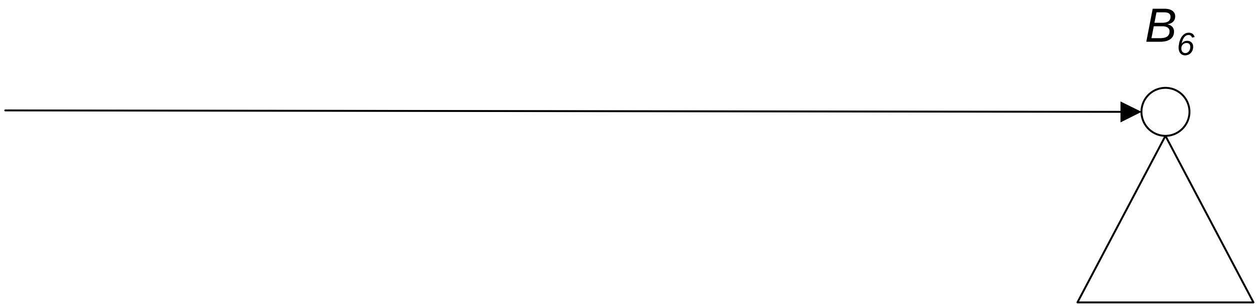
Q.deletemin():



Zeit: O(log n)

Binomialqueues Worst Case Folge von Operationen

insert(e, Q):



Zeit: O(log n)