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

## Folien: Fibonacci-Heaps

**Autor: Sven Schuierer**

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

# 1 Fibonacci-Heaps

Lazy-meld-Version von Binomialqueues
$\Rightarrow meld$, $decreasekey$ werden billiger

Ein Fibonacci-Heap $Q$ ist eine Kollektion heapgeordneter Binomial-Bäume

$Q.min$: Wurzel des Baumes mit kleinstem Schlüssel

$Q.rootlist$: zirkuläre, doppeltverkette und ungeordnete Liste der Wurzeln der Bäume

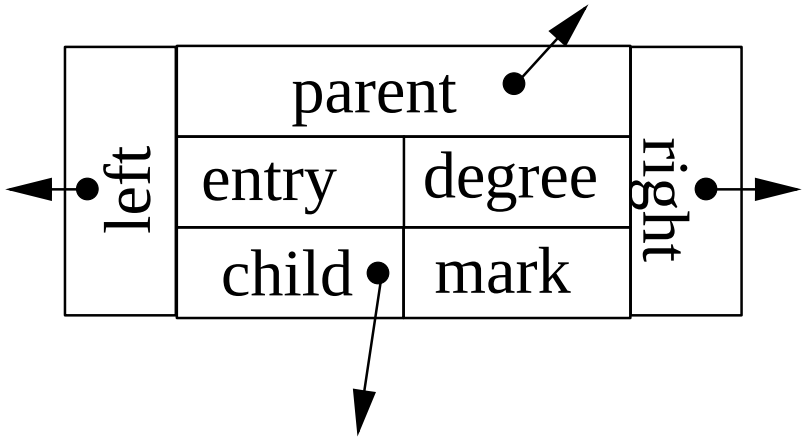
$Q.size$: Anzahl der Knoten in $Q$

# 2 Bäume

Sei $B$ heap-geordneter Baum in $Q.rootlist$:

$B.childlist$: zirkuläre, doppeltverkette und ungeordnete Liste der Söhne von $B$

Knotenformat:




Vorteile zirkulärer, doppeltverketter Listen:

1. Entfernen eines Elements in konstanter Zeit
2. Vereinigen zweier Listen in konstanter Zeit

# 3 Operationen

$F$-Heap mit Operationen

$initialize$,

$insert$,

$meld$,

$min$,

$deletemin$

= “lazy” Binomialqueue (d.h. Liste von Binomialbäumen)

Zwei Unterschiede zu Binomialqueue:

1. Söhne der Binomialbäume sind ungeordnet
2. Mehr als ein Binomialbaum des Ranges $i$ in $Q.rootlist$ möglich

# Meld

$Q.initialize()$: $Q.rootlist = Q.min = null$

$Q.meld(Q')$:
1. verkette $Q.rootlist$ und $Q'.rootlist$
2. update $Q.min$

$Q.insert(e)$:
1. Bilde einen Knoten für einen neuen Schlüssel $\Rightarrow Q'$
2. $Q.meld(Q')$

$Q.min()$:
Gebe $Q.min.entry$ zurück

# Delete-Min

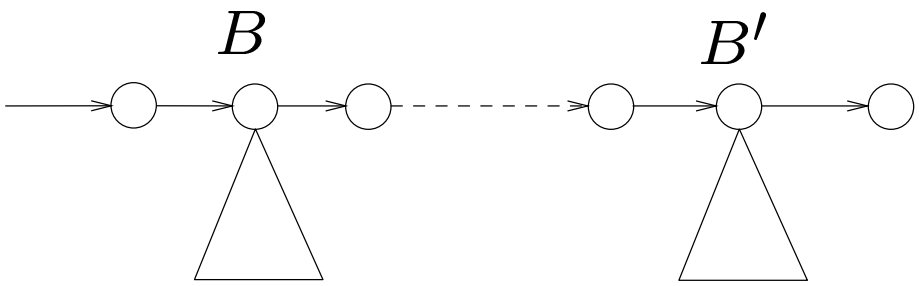
$Q.deletemin()$:

1. Entferne $Q.min$ aus $Q.rootlist$ und hänge $Q.min.childlist$ in $Q.rootlist$ ein

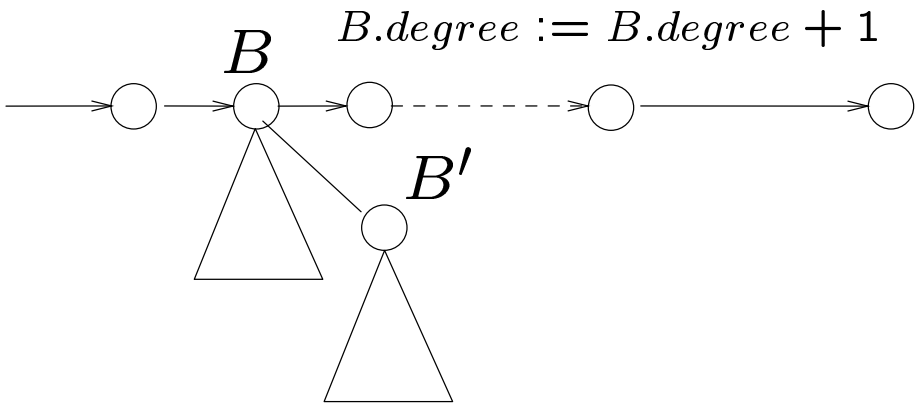
2. Konsolidiere Wurzelliste durch Verschmelzen von Wurze

Bemerkung Jeder Rang ist höchstens 1-mal in $Q.rootlist$ vertreten

$$Link(B, B'): \{B.key \leq B'.key\}$$




wird zu:



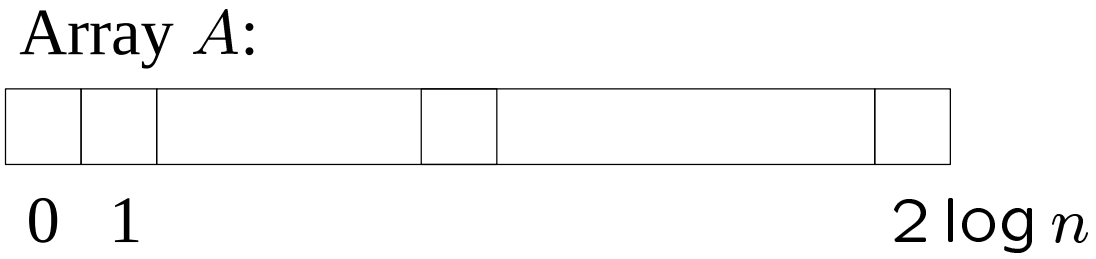
# 4 Rang

## Maximaler Grad

$Q.degree =$ maximaler Knotengrad eines Baumes in $Q$.

Annahme: $Q.degree \leq 2 \log n$, wenn $Q.size = n$.

Finde Wurzeln mit gleichem Rang:




$Q.consolidate()$

```
1 fibNode [] A = new fibNode [2 log n]
2 for i = 0 to 2 log n do
3     A[i] = null
4 while Q.rootlist ≠ ∅ do
5     B = Q.delete-first()
6     A.trage-ein(B)
  end while;
7 Q.rootlist = A.list()
8 bestimme Q.min
```

## Eintrag

```
A.trage-ein(B)
1 if A[B.degree] == null
2   then A[B.degree] = B
3   else B' = A[B.degree]
4        A[B.degree] = null
5        B = Link(B, B')
6        A.trage-ein(B)
```

# Analyse

Potentialmethode zur Analyse der Kosten der Operationen von $F$-Heaps

Potentialfunktion $\Phi_Q$:

$$\Phi_Q = r_Q + 2m_Q$$

mit: $r_Q =$ Anzahl der Knoten in $Q.rootlist$

$m_Q =$ Anzahl der markierten Knoten in $Q$

# 5 Dekrease-Key

$Q.decrease\text{-}key(v, k)$

1 if $k > v.key$ then return

2 $v.key = k$

3 update $Q.min$

4 if $v.parent == null$ or $k \geq v.parent.key$

5 then return

6 $v.mark = true$

7 while $v.parent \neq null$ and $v.mark$ do

8 $parent = v.parent$

/* trenne $v$ von seinem Vater */

9 $Q.cut(v)$

/* kaskadiere */

10 $v = parent$

end while;

11 if $v.parent \neq null$ then $v.mark = true$

Alle Knoten in $Q.rootlist$ sind nicht markiert.

# Invariante

Falls $v$ markiert ist und $v.parent \neq null$, dann muß $v$ von $v.parent$ getrennt werden.

$v$ verliert zum zweiten Mal einen Sohn
(d.h. $v.mark = true$)
$\Rightarrow$ $v$ wird auch abgetrennt

$Q.cut(v)$
1 if $v.parent \neq null$
2 then /* trenne $v$ von seinem Vater */
3 $v.parent.degree = v.parent.degree - 1$
4 $v.parent = null$
5 $v.mark = false$
6 entferne $v$ aus $v.parent.childlist$
7 füge $v$ in $Q.rootlist$ ein

## Historie

$v$ ist Wurzel $\longrightarrow$ $v.mark = false$
($deletemin$, $insert$: $mark = false$)

$\downarrow$

$v$ wird an einen Knoten angehängt $\longrightarrow$ $v.mark = false$

$\downarrow$

$v$ verliert einen Sohn $\longrightarrow$ $v.mark = true$

$\downarrow$

$v$ verliert zweiten Sohn $\longrightarrow$ abtrennen

# 6 Vorrangswarteschlangen

Zusammenfassung:

| | Liste | Heap | Bin.-Q. | Fib.-Hp. |
|---|---|---|---|---|
| initial. | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| insert | $O(1)$ | $O(\log n)$ | $O(\log n)$ | $O(1)$ |
| min | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| 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 | $O(1)$ | $O(\log n)$ | $O(\log n)$ | $O(1)^*$ |
| delete | $O(1)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)^*$ |

$*$ = amortisierte Zeit