Algorithmen und Datenstrukturen

Prioritätswarteschlangen mit Fibonacci-Heaps

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Zusammenfassung bisheriger Ergebnisse

Laufzeit	Binärer-Heap	Binomial-Heap
insert	O(log n)	O(log n)
min	O(1)	O(1)
deleteMin	O(log n)	O(log n)
delete	O(log n)	O(log n)
decreaseKey	O(log n)	O(log n)
merge	O(n)	O(log n)

Wunschliste: Weitere Verbesserungen

Laufzeit	Binärer-Heap	Binomial-Heap
insert	O(log n)	O(log n)
min	O(1)	O(1)
deleteMin	O(log n)	O(log n)
delete	O(log n)	O(log n)
decreaseKey	O(log n)	O(log n)
merge	O(n)	O(log n)

Datenstruktur Fibonacci-Heap

- Baut auf Binomial-Bäumen auf,
  aber erlaubt lazy merge und lazy delete.

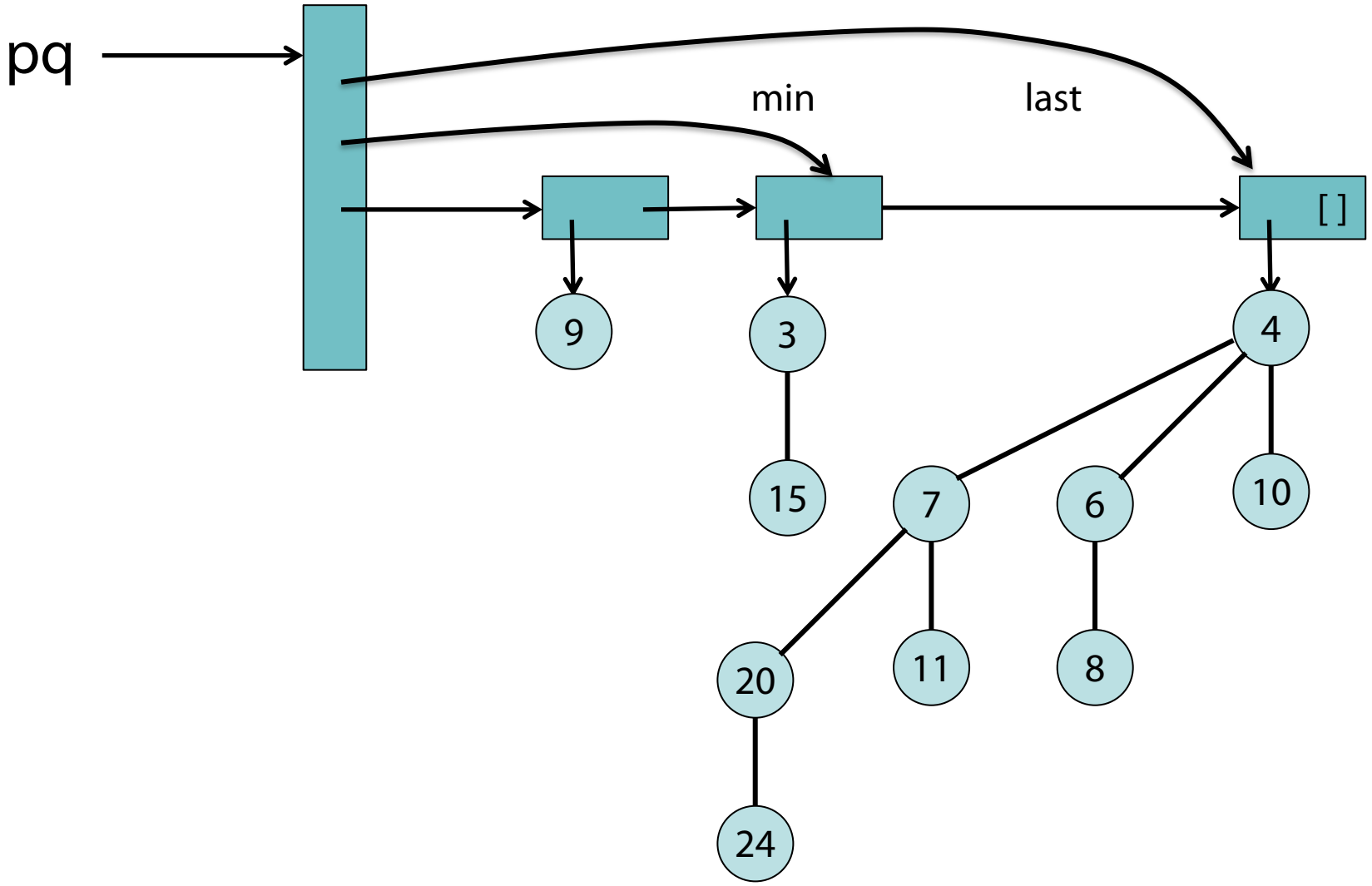
- Lazy merge: keine Verschmelzung von Binomial-Bäumen gleichen Ranges bei merge, nur Verkettung der Wurzellisten

- Lazy delete: erzeugt unvollständige Binomial-Bäume

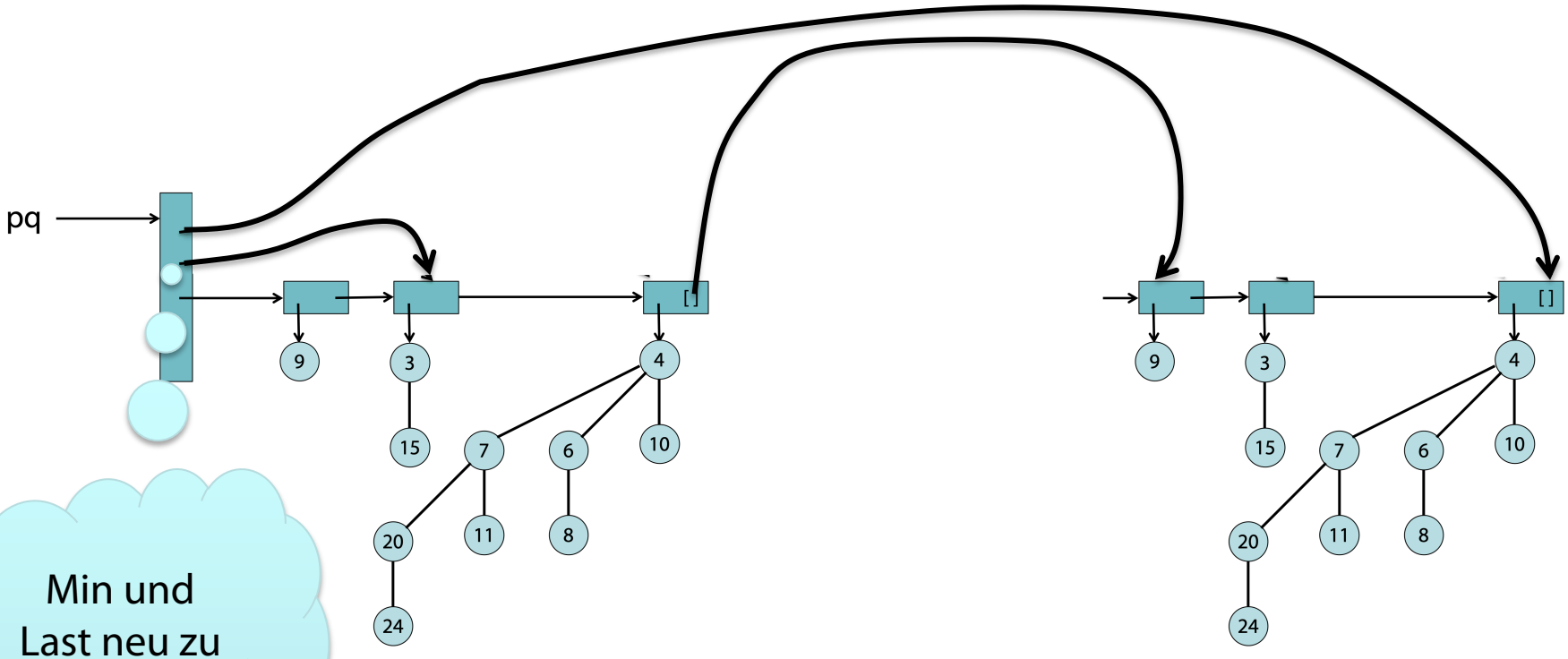
Der Name rührt von der Analyse der Datenstruktur her, bei der Fibonacci-Zahlen eine große Rolle spielen (wird nachher deutlich)

Michael L. Fredman, Robert E. Tarjan: Fibonacci heaps and their uses in improved network optimization algorithms. In: Journal of the ACM. 34, Nr. 3, S. 596–615, **1987**

Prioritätswarteschlangen als Binomial-Heaps



Verkettung der Liste zweier PQs

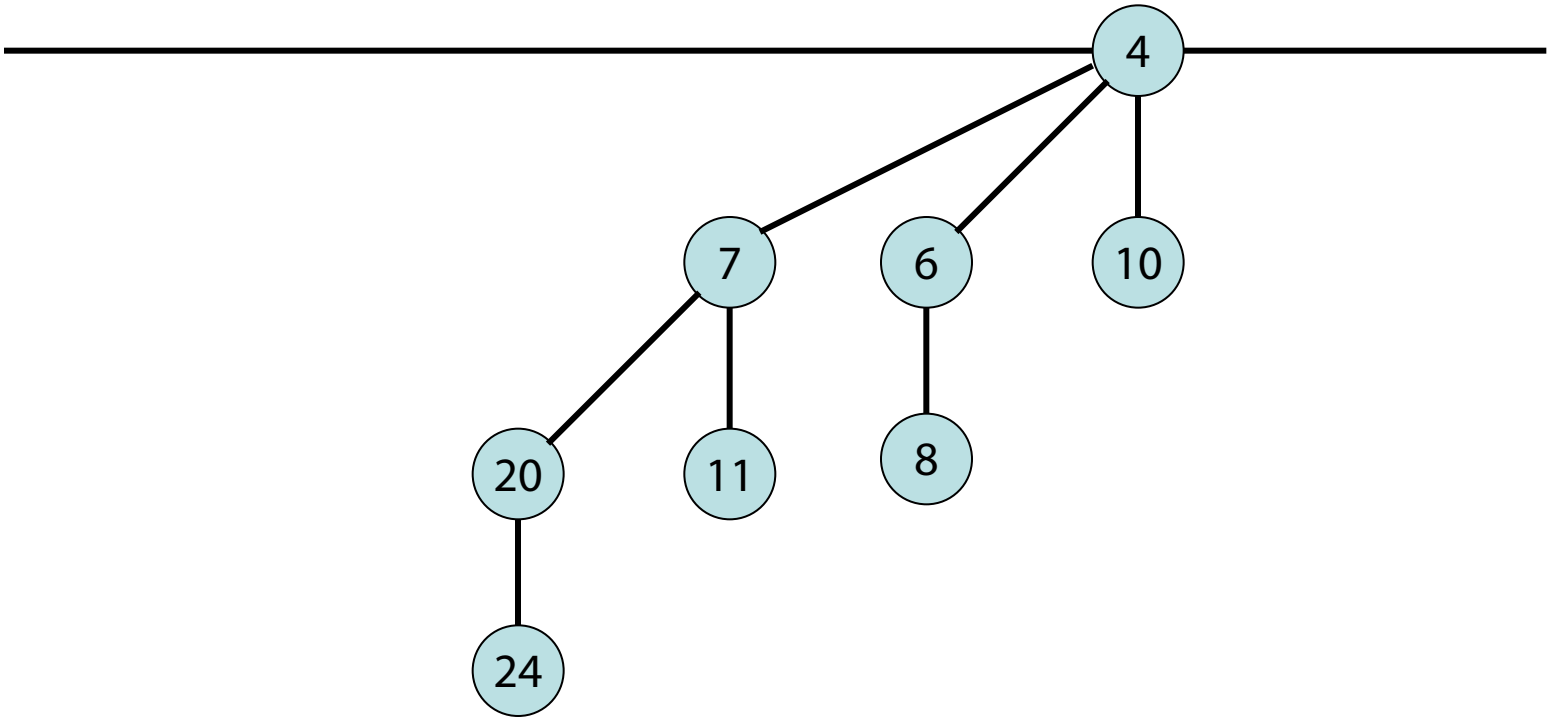


Min und Last neu zu bestimmen

Fibonacci-Heap

Baum in Binomial-Heap: Zentrale Invariante

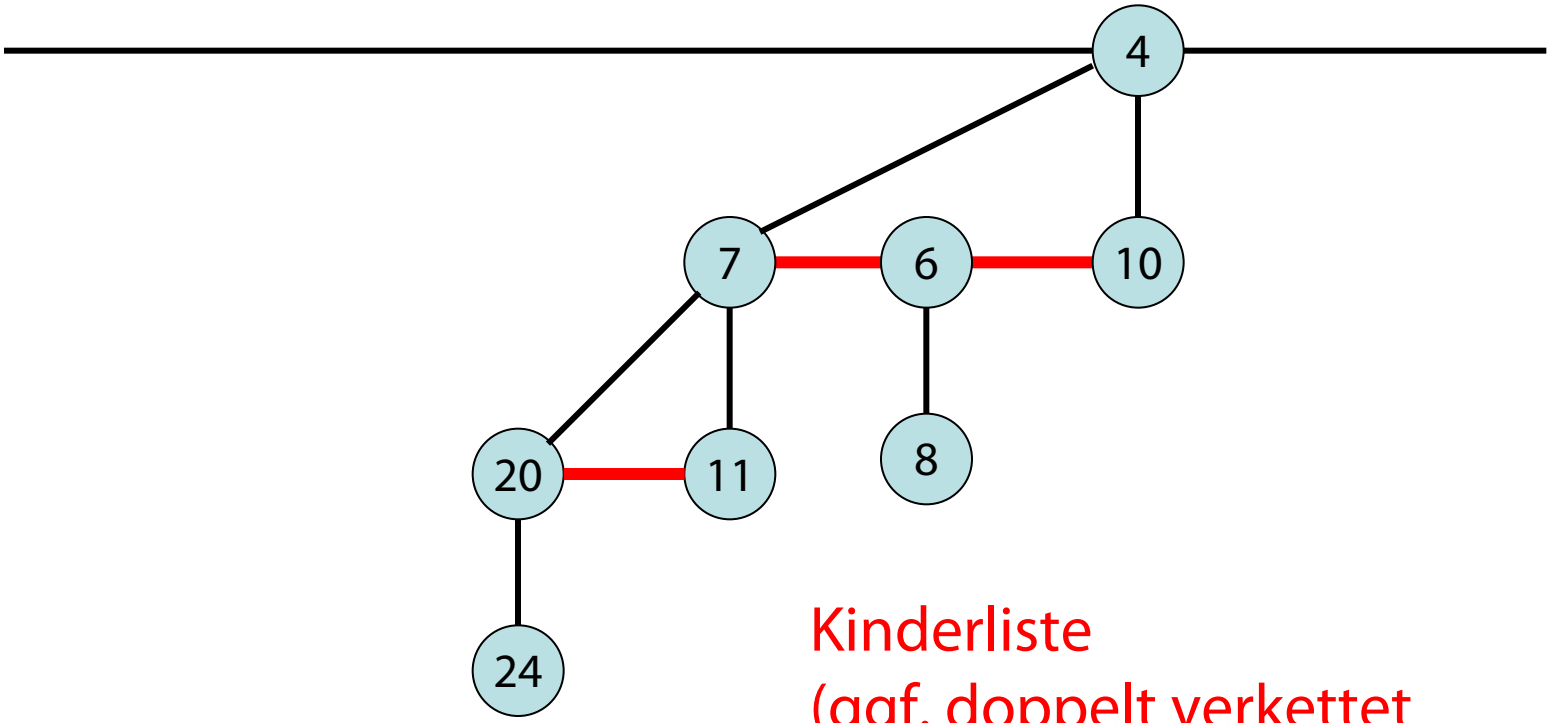
Rang = Anzahl der Kinder des Wurzelknotens



Fibonacci-Heap: Anpassung der Struktur

Baum in Fibonacci-Heap: Zentrale Invariante

Rang = Anzahl der Kinder des Wurzelknotens



Kinderliste
(ggf. doppelt verkettet
dito für Wurzelliste)

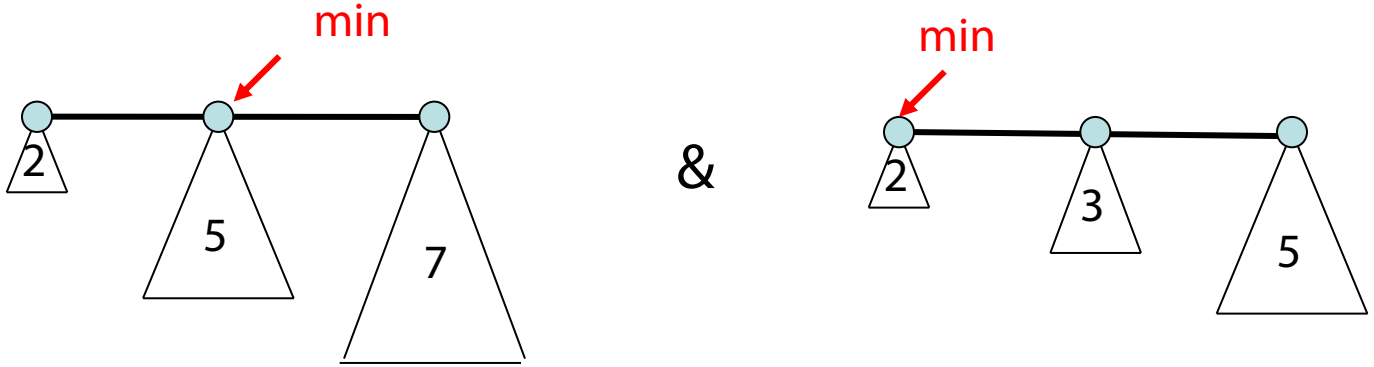
Fibonacci-Heap

Für jeden Knoten wird gespeichert:

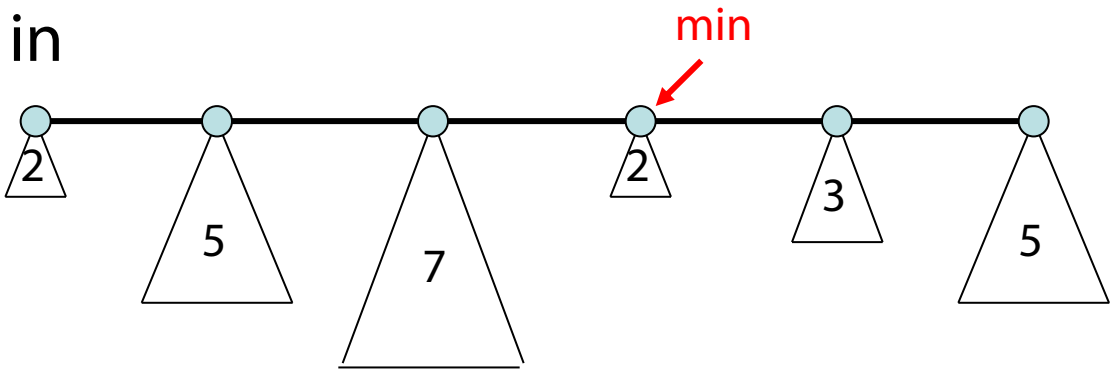
- im Knoten gespeichertes Element
- Vater
- Liste der Kinder (mit Start- und Endpunkt)
- Rang

Fibonacci-Heap: Lazy-Merge

Lazy merge von



resultiert in



Wurzellisten hintereinander-hängen: O(1)

Problem:

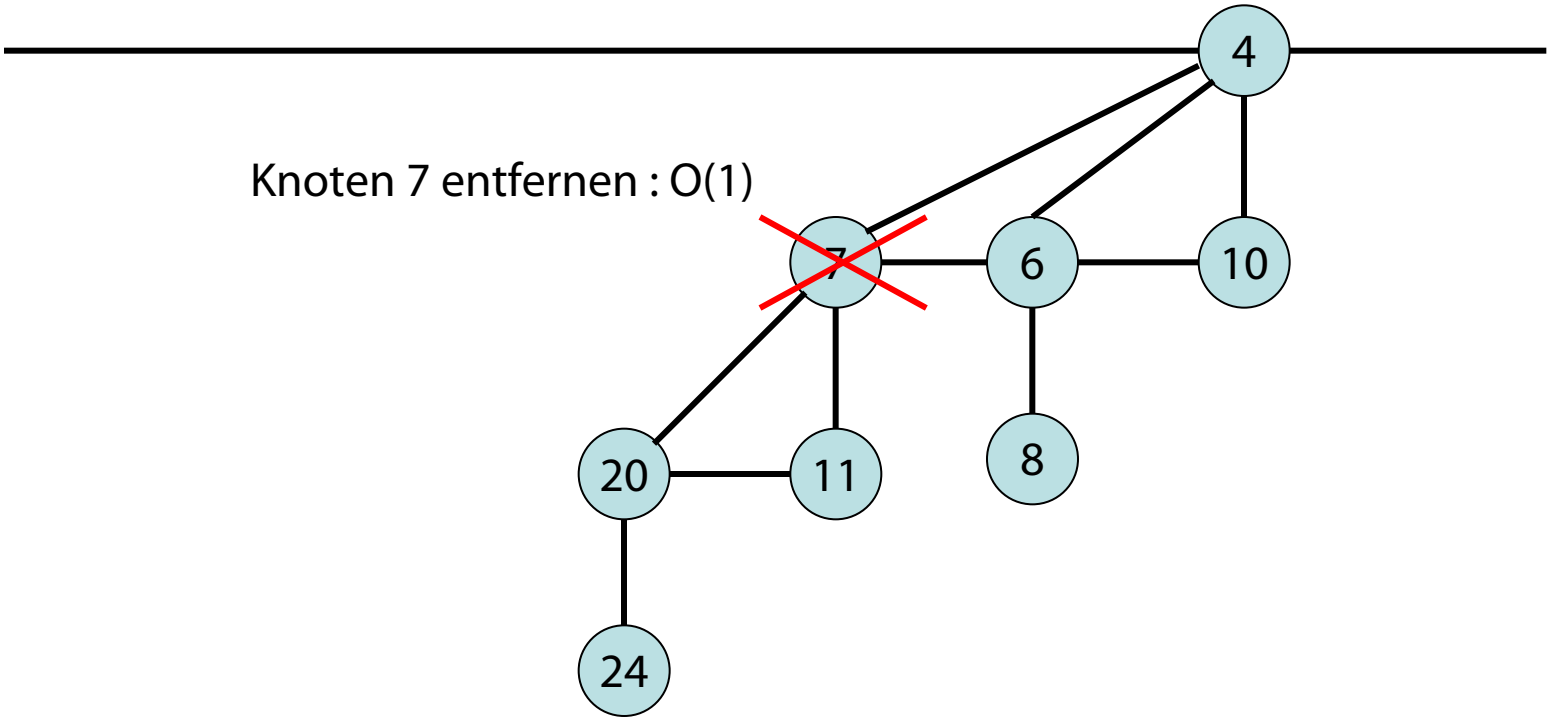
- Bäume gleichen Ranges treten in der Wurzelliste auf
- Binomial-Heap-Eigenschaft kann verletzt sein

Fibonacci-Heap: Lazy-Delete

Lazy delete:

Kinderliste in Wurzelliste integrieren: O(1)

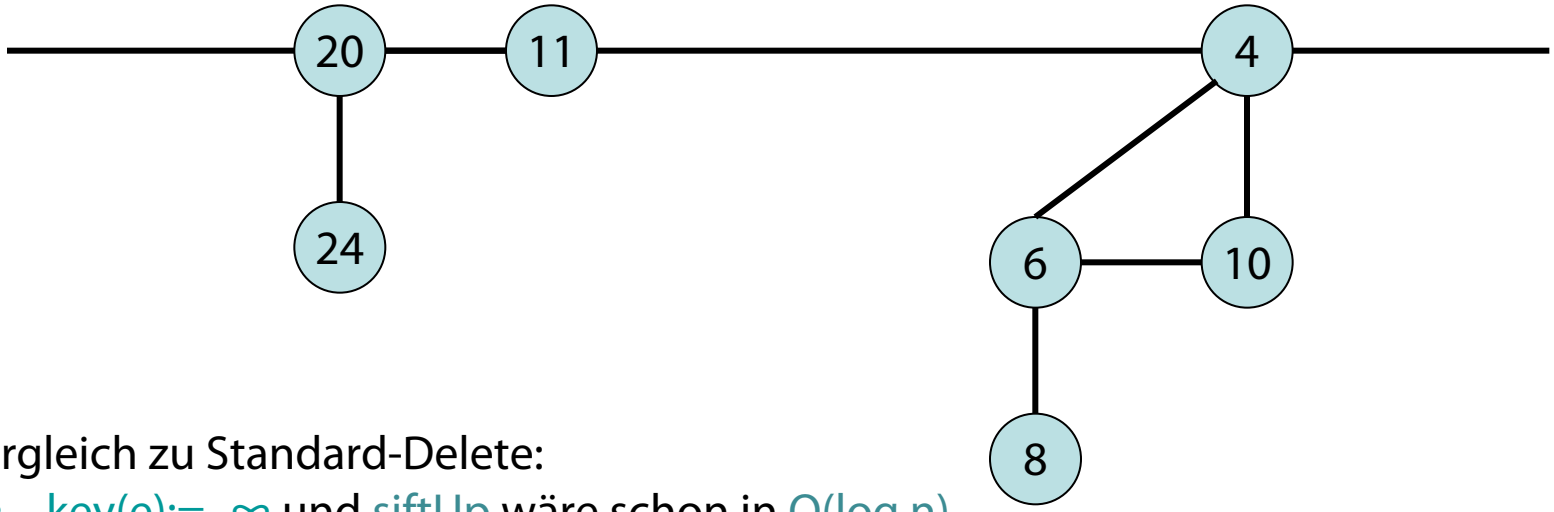
Knoten 7 entfernen : O(1)



Fibonacci-Heap: Lazy-Delete

Lazy delete:

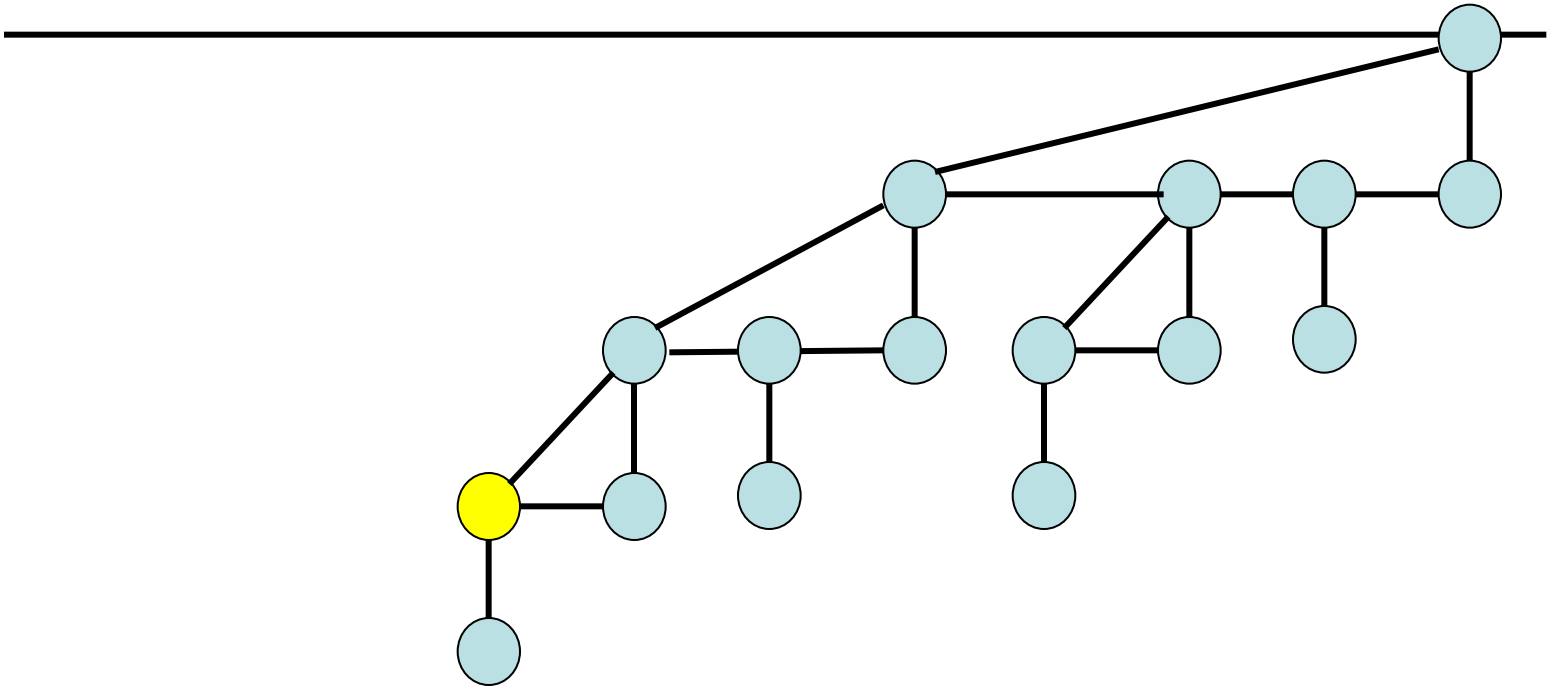
Annahme: Knoten 7 ist im direkten Zugriff
Aufwand: O(1), da gegebener Knoten 7 in
O(1) Zeit entfernbar und Kinderliste
von 7 in O(1) Zeit in Wurzelliste integrierbar



- Vergleich zu Standard-Delete:
  - key(e):= -∞ und siftUp wäre schon in O(log n)
- Also kein siftUp
- Problem:
  - Bäume gleichen Ranges treten in der Wurzelliste auf
  - Binomial-Heap-Eigenschaft kann verletzt sein

Fibonacci-Heap

Beispiel für delete(x)



Fibonacci-Heap: Übersicht

Operationen:

- merge: Verbinde Wurzellisten, aktualisiere min-Pointer: Zeit O(1)
- insert(x, pq): Füge B_0 (mit x) in Wurzelliste ein, aktualisiere min-Pointer. Zeit O(1)
- min(pq): Gib Element, auf das der min-Pointer zeigt, zurück. Zeit O(1)
- deleteMin(pq), delete(x, pq), decreaseKey(x, pq, Δ): noch zu bestimmen…

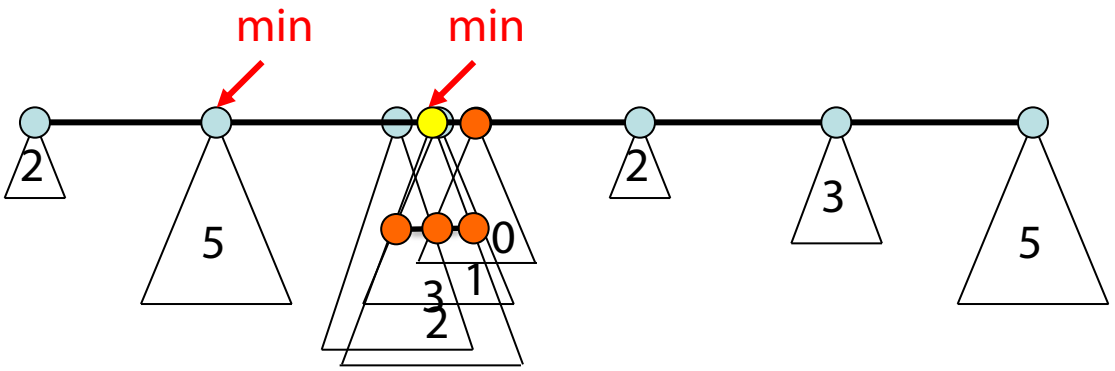
Fibonacci-Heap

deleteMin(pq): Diese Operation hat Aufräumfunktion.
Der min-Pointer zeige auf x.

procedure deleteMin()
entferne x aus Wurzelliste
konkateniere Kinderliste von x mit Wurzelliste
while *≥2 Bäume mit gleichem Rang i* **do**
merge Bäume zu Baum mit Rang i+1 // (wie bei zwei Binomial-Bäumen)
aktualisiere den min-Pointer

- Durch die Integration der Kinderliste in die Wurzelliste können dort Bäume gleichen Ranges auftreten, die Struktur wird jedoch danach konsolidiert
- Die schon durch delete auftretenden Heaps gleichen Ranges werden gleich mit behandelt!

Beispiel: deleteMin

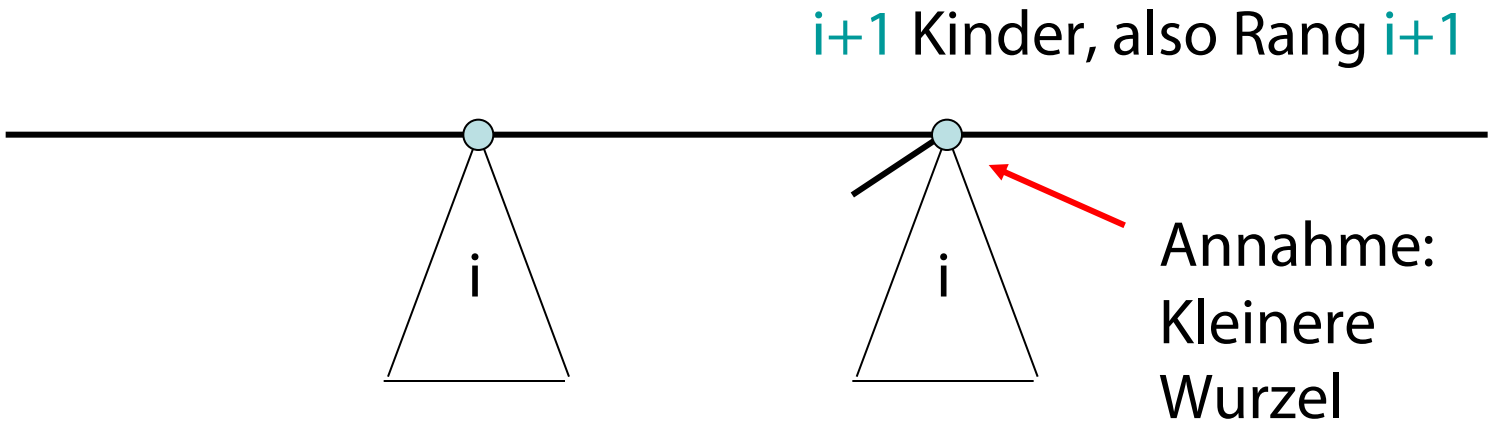


vorher

Binomial-Heap-Eigenschaft gilt auch

Verschmelzung zweier Bäume mit Rang i

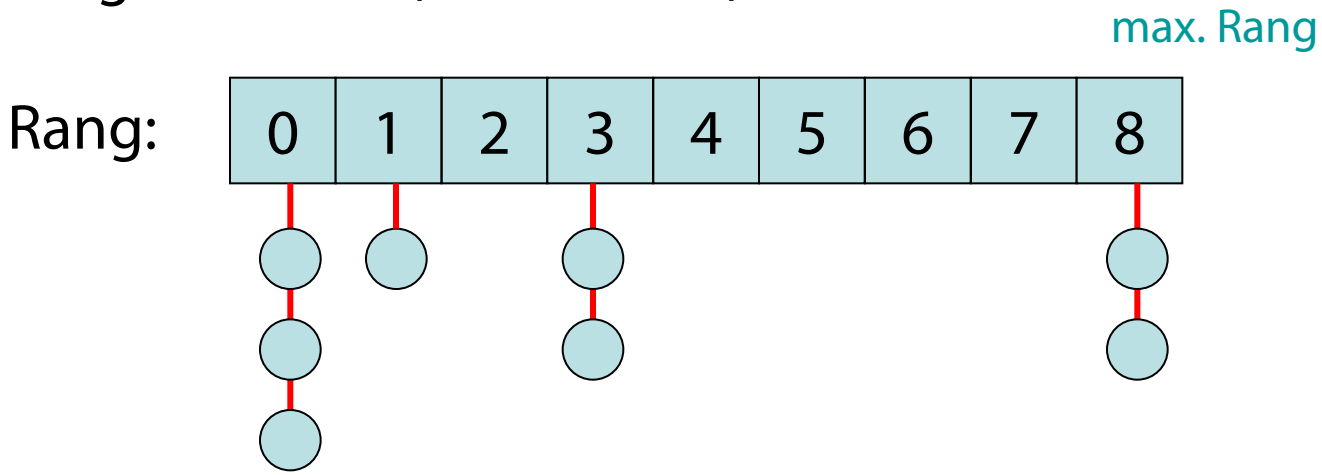
(d.h. Wurzelknoten haben i Kinder):



Fibonacci-Heap

Effiziente Findung von Wurzeln mit gleichem Rang:

- Scanne vor while-Schleife alle Wurzeln und speichere diese nach Rängen in Feld (Eimerkette!):



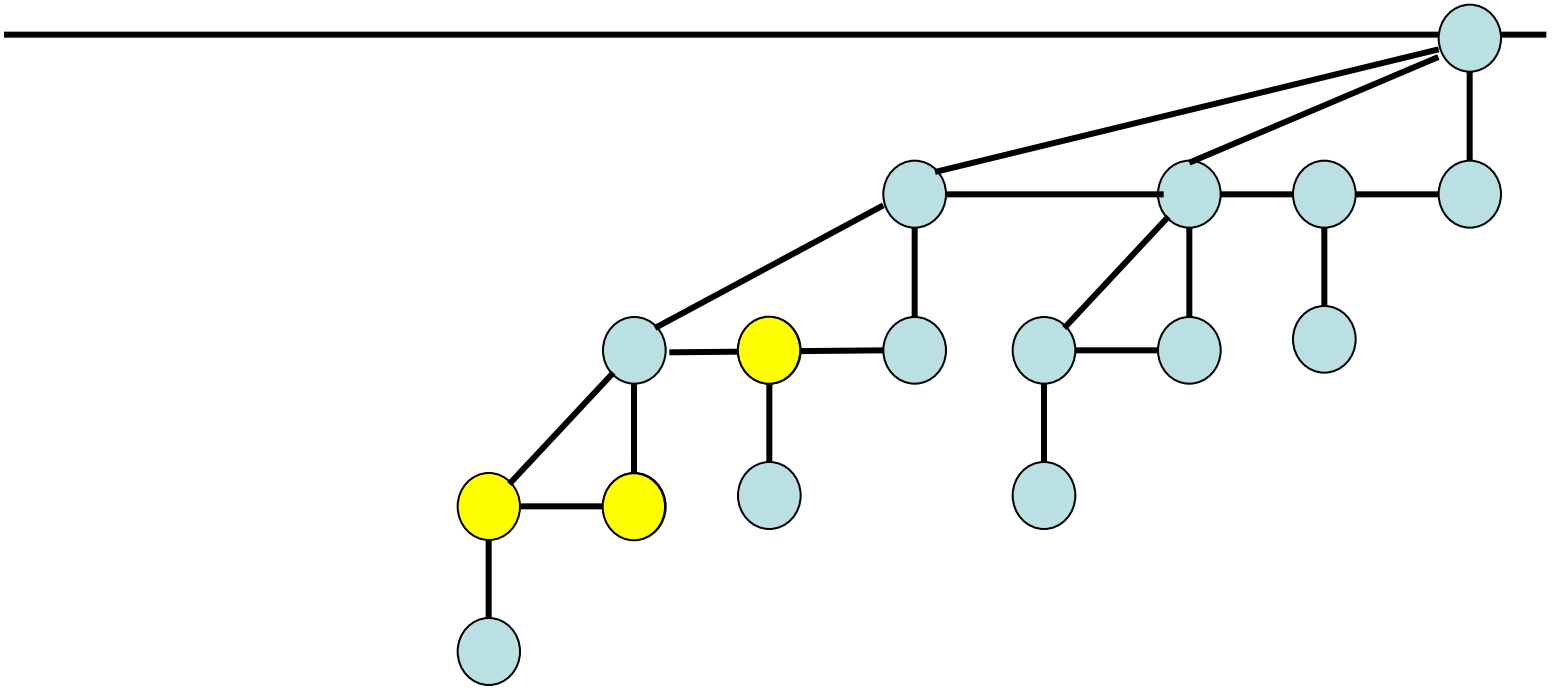
- Merge dann wie bei Binomialbäumen von Rang 0 an bis maximaler Rang erreicht

Operation delete

- Konsolidierung im Inneren der Heaps

Fibonacci-Heap

Beispiel für delete(x): (: Mark=1)



Fibonacci-Heap

Sei parent(x) Vater von Knoten x. Wenn x neu eingefügt wird, ist Mark(x)=0 (Mark(x) speichert, ob ein Kind entfernt wurde).

```
procedure delete(x):
    if x ist min-Wurzel then
       deleteMin()
    else
      hänge x aus, füge Kinder von x in Wurzelliste ein (entmarkiere entsprechend)
    if not parentExists(x) then
      exit   // x ist Wurzel
    while true do
       x:=parent(x)
       if not parentExists(x) then
           exit     // x ist Wurzel
       if Mark(x)=0 then
           Mark(x):=1
           exit
       else      // Mark(x)=1, also schon Kind weg
          hänge x samt Unterbaum in Wurzelliste
          Mark(x):=0  // Wurzeln benötigen kein Mark
```

Fibonacci-Heap

```
procedure decreaseKey(x,Δ):
    füge x samt Unterbaum in Wurzelliste ein (entmarkiere entsprechend)
    key(x):=key(x)-Δ
    aktualisiere min-Pointer
    if not parentExists(x) then
       exit // war x Wurzel?
    while true do
      x:=parent(x)
      if not parentExists(x) then
        exit // x ist Wurzel
      if Mark(x)=0 then
         Mark(x):=1
         exit
      else      // Mark(x)=1
         hänge x samt Unterbaum in Wurzelliste
         Mark(x):=0
```

Fibonacci-Heap mit markierten Fehlern

Zeitaufwand:

- deleteMin():
  O(max. Rang + #Baumverschmelzungen)
- delete(x), decreaseKey(x,Δ):
  O(1 + #kaskadierende Schritte)
  d.h. #umgehängter markierter Knoten

Wir werden sehen:

Zeitaufwand kann in beiden Fällen durch n gedeckelt werden (ist also in O(n)), ist aber richtig verteilt viel günstiger.

Strukturfehler: Verschiebung der Arbeit

- Statt bei jedem merge einen Aufwand von O(log n) zu leisten, ...
- ... wird die Arbeit bei einem deleteMin mit übernommen, ...
- ... mit der Idee, dass man die entsprechenden Strukturen dort sowieso anfassen muss
- Das Umverteilen kann sich über eine längere Sequenz von Operationen durchaus amortisieren
- Vgl. build für binäre Heaps

Zusammenfassung

- Wird uns unsere Intuition der Verschiebung der Arbeit im Fibonacci-Heap helfen?
- Werden wir für viele Zugriffe den erhofften Gewinn erzielen?