

# Fibonacci-Heaps

VO Algorithm Engineering

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

26. VO

06./07.02.2006

## Literatur für diese VO

- T. Ottmann und P. Widmayer: Algorithmen und Datenstrukturen, Spektrum Akademischer Verlag, 2002
- S. Krumke: Algorithmen und Datenstrukturen, Skript, TU Berlin, WS 2002/03
- Folienteil: G. Klau, TU Wien, SS 2004 (jetzt: FU Berlin)

# Überblick

Einführung Fibonacci

Einführung Fibonacci-Heaps

Realisierung

Amortisierte Analyse

## Leonardo von Pisa

- Fibonacci = „Sohn des Bonacci“
- ca. 1170 – 1240, Pisa
- Mathematiker, Gleichungslehre
- Liber Abaci 1202



$$F(n+2)=F(n)+F(n+1)$$

$$F(1) = 1$$

$$F(2) = 1$$

- Berühmte Kaninchenaufgabe:

*„A certain man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair which from the second month on becomes productive?“*

n	1	2	3	4	5	6	7	8	9	10	11	12
F(n)	1	1	2	3	5	8	13	21	34	55	89	144
F(n)/ F(n-1)		1,0	2,0	1,5	1,667	1,60	1,625	1,615	1,619	1,618	1,618	1,618
F(n-1)/ F(n)		1,0	0,5	0,667	0,600	0,625	0,615	0,619	0,618	0,618	0,618	0,618

## Leonardo von Pisa

- Fibonacci = „Sohn des Bonacci“
- ca. 1170 – 1240, Pisa
- Mathematiker, Gleichungslehre
- Liber Abaci 1202



$$F(n+2) = F(n) + F(n+1)$$

$$F(1) = 1$$

$$F(2) = 1$$

- $\Phi = (1 + \sqrt{5})/2 = 1,6180339887\dots$
- $\underline{\Phi} = (1 - \sqrt{5})/2 = -0,6180339887\dots$
- $\Phi^{-1} = 2/(1 + \sqrt{5}) = -\underline{\Phi} = (\sqrt{5} - 1)/2 = 0,6180339887\dots$
- $\Phi^{-1}$ : „Der Goldene Schnitt“

<b>n</b>	1	2	3	4	5	6	7	8	9	10	11	12
<b>F(n)</b>	1	1	2	3	5	8	13	21	34	55	89	144
<b>F(n)/ F(n-1)</b>		1,0	2,0	1,5	1,667	1,60	1,625	1,615	1,619	1,618	1,618	1,618
<b>F(n-1)/ F(n)</b>		1,0	0,5	0,667	0,600	0,625	0,615	0,619	0,618	0,618	0,618	0,618

## Prioritätswarteschlangen

- Ausgangsfrage: Wie kann man den Datentyp Prioritätswarteschlange realisieren?
  - Anwendungen, z.B.
    - kürzeste Wege mit Dijkstra
    - MST mit Prim
    - Finden eines minimalen Schnittes



**Operationen:**

Insert

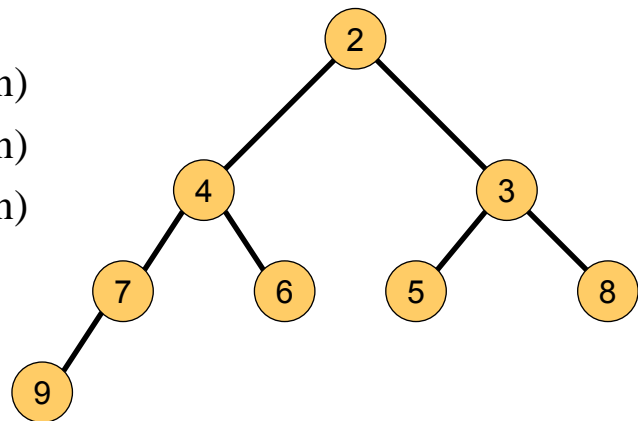
ExtractMin

DecreaseKey

# Binäre Heaps

Heapeigenschaft: Elter ist nicht größer als seine Kinder

- **Insert:**  $O(\log n)$
- **ExtractMin:**  $O(\log n)$
- **DecreaseKey:**  $O(\log n)$



n Knoten

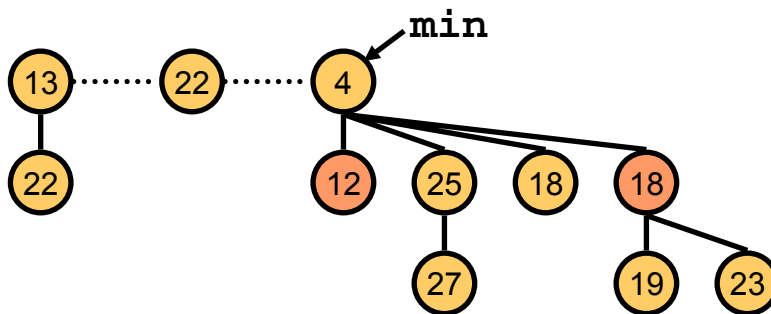
## Fibonacci-Heaps (Fredman & Tarjan 1987)

Operation	Bin. Heap	Fibonacci-Heap
<b>Insert</b>	$O(\log n)$	$O(1)$
<b>ExtractMin</b>	$O(\log n)$	$O(\log n)^*$
<b>DecreaseKey</b>	$O(\log n)$	$O(1)^*$

\* Nicht ganz fair, weil amortisierte Analyse

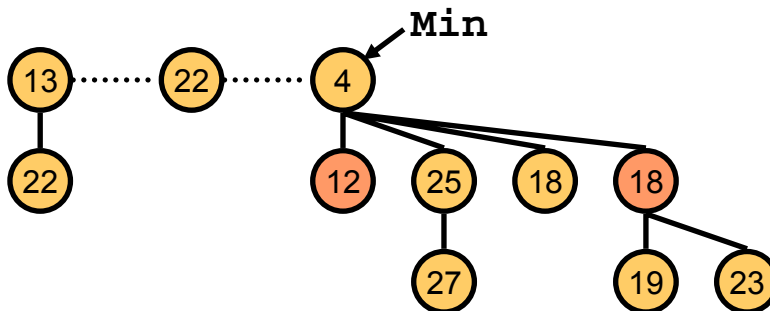
## Fibonacci-Heaps: Features

- Menge von Bäumen, die alle die Heapeigenschaft haben
- Globaler **Min**-Pointer
- Bäume nicht zwangsläufig binär
- Jeder Knoten ist entweder markiert oder nicht



## Fibonacci-Heaps: Realisierung

- Alle Wurzeln durch doppelt-verkettete, zyklisch geschlossene Liste verbunden: Wurzelliste
- Jeder Knoten hat Zeiger auf seinen Elter und auf einen seiner Kinder (**Elter**, **Kind**)
- Alle Kinder sind untereinander doppelt zyklisch verkettet
- Jeder Knoten hat Felder
  - **Schlüssel**, **Mark**, **Rang** (Anzahl der Kinder)



## Verschmelze ( $h_1, h_2$ )

- Verschmelzen zweier F-Heaps  $h_1$  und  $h_2$ :
  - Hänge beide Wurzellisten aneinander
  - Der **min**-Pointer zeigt zum kleineren der beiden Min-Zeiger
- Aufwand:  $O(1)$  Zeit.

## Insert (v)

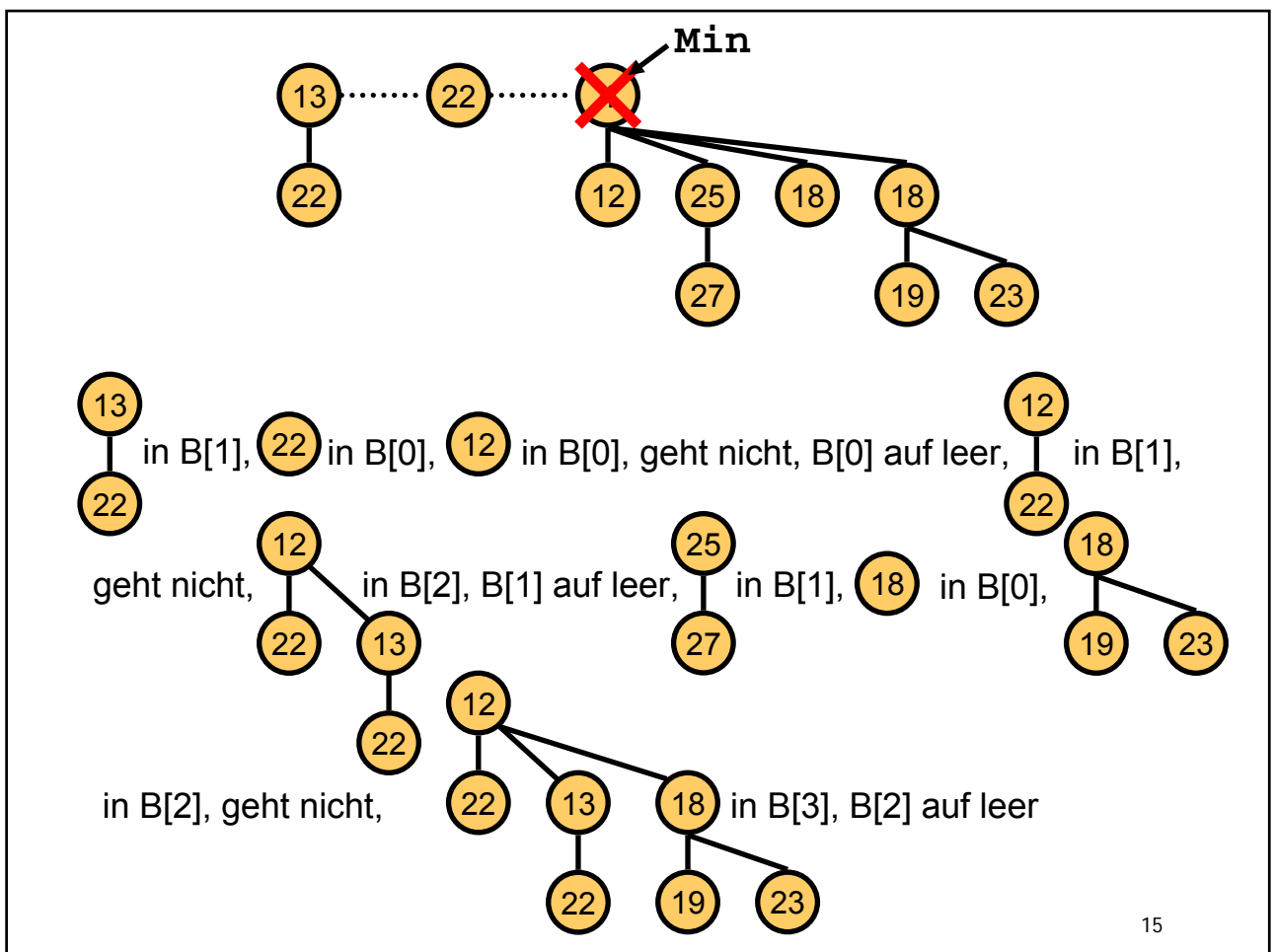
- Insert(v) in einen F-Heap h:
  - Bilde einen F-Heap h' aus einem einzigen Knoten v (**Mark=0, Rang=0**)
  - Verschmelze h und h' zu einem neuen F-Heap
- Aufwand:  $O(1)$  Zeit. Die Faulheit rächt sich aber bei **ExtractMin...**

## Fibonacci-Heaps: ExtractMin

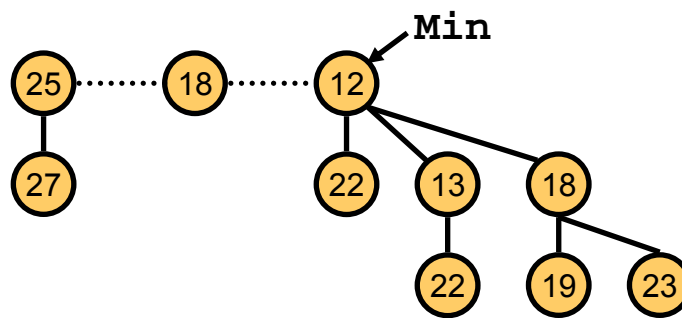
- Es ist einfach, das Minimum zu finden (**Min-Pointer**)
- Entferne den Minimalknoten aus der Wurzelliste von  $h$
- Hänge alle Kinder von  $Min$  in die Wurzelliste ein
- Durchlaufe alle Wurzeln, um neues Minimum zu finden
- **Idee:** Wenn man das schon machen muss, kann man dabei auch aufräumen
  - Wir sorgen dafür, dass keine zwei Wurzeln den gleichen Grad haben
  - Dazu: Feld  $B[0..n-1]$  mit der Interpretation  $B[i] = v \leftrightarrow \text{deg}(v) = i$

## Fibonacci-Heaps: ExtractMin

- Für alle Wurzeln  $v$ :
  - Wiederhole
    - $d = \text{deg}(v)$ ;
    - falls  $B[d]$  leer setze  $B[d] = v$ ;
    - sonst: mache größere Wurzel zum Kind der kleineren;
      - $v = \text{kleinere Wurzel}$ ;
      - // der Grad von  $v$  ist jetzt  $d + 1$
      - $B[d] = \text{leer}$ ; // wieder frei
  - bis ein leeres Feld gefunden wurde



## Fibonacci-Heaps: ExtractMin



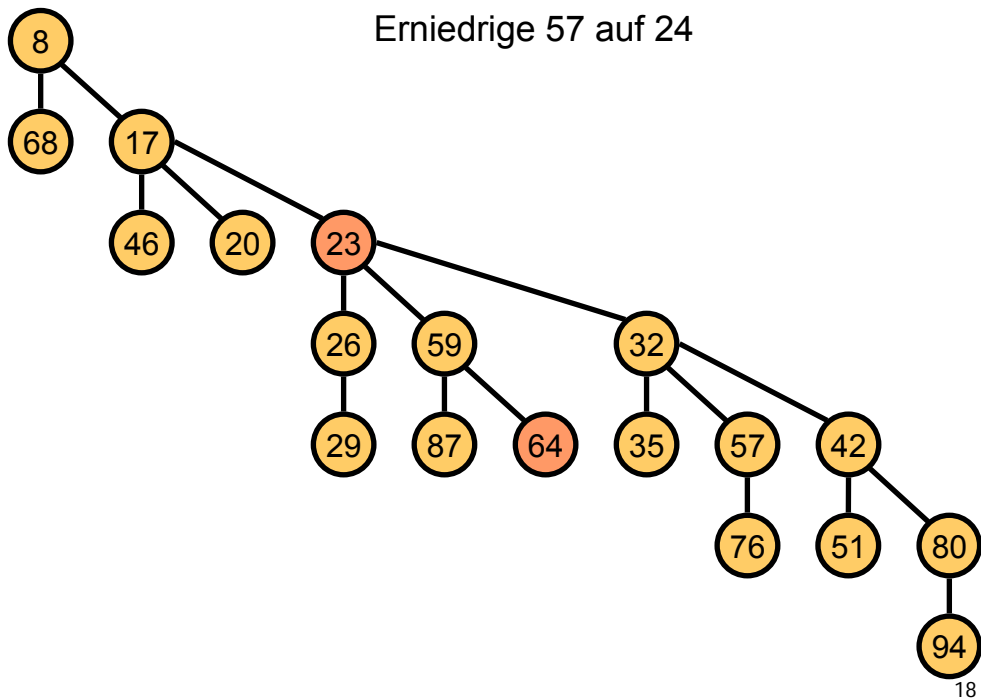
- Laufzeit:  $O(d + m)$ ,
  - wobei  $d = \text{deg}(\text{gelöschte Wurzel})$
  - und  $m = \#\text{Wurzeln vor dem Löschen}$
- genauere Analyse: später

## Fibonacci-Heaps: DecreaseKey

- **DecreaseKey**( $v$ , newKey) :
  - Reduziere Schlüssel von  $v$  auf newKey;
  - Wiederhole:
    - Trenne  $v$  von  $p=\mathbf{Elter}(v)$  ab;
    - Füge  $v$  in Wurzelliste ein; setze  $\mathbf{Mark}(v)=0$ ;
    - Reduziere  $\mathbf{Rang}(p)$
    - Setze  $v=p$ ;
  - solange  $\mathbf{Mark}(p)=1$ ; // trenne solange Elter( $v$ ) ab,  
// bis unmarkiert
  - Setze  $\mathbf{Mark}(p)=1$ ;

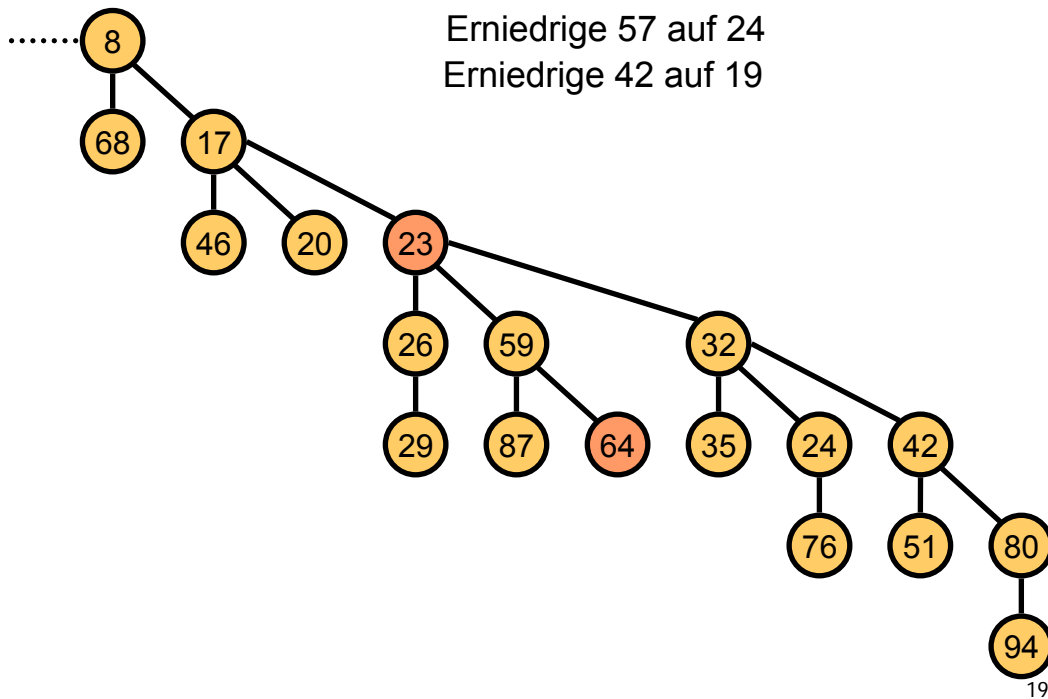
Markierung verhindert, dass F-Heaps zu dünn werden

## Fibonacci-Heaps: DecreaseKey

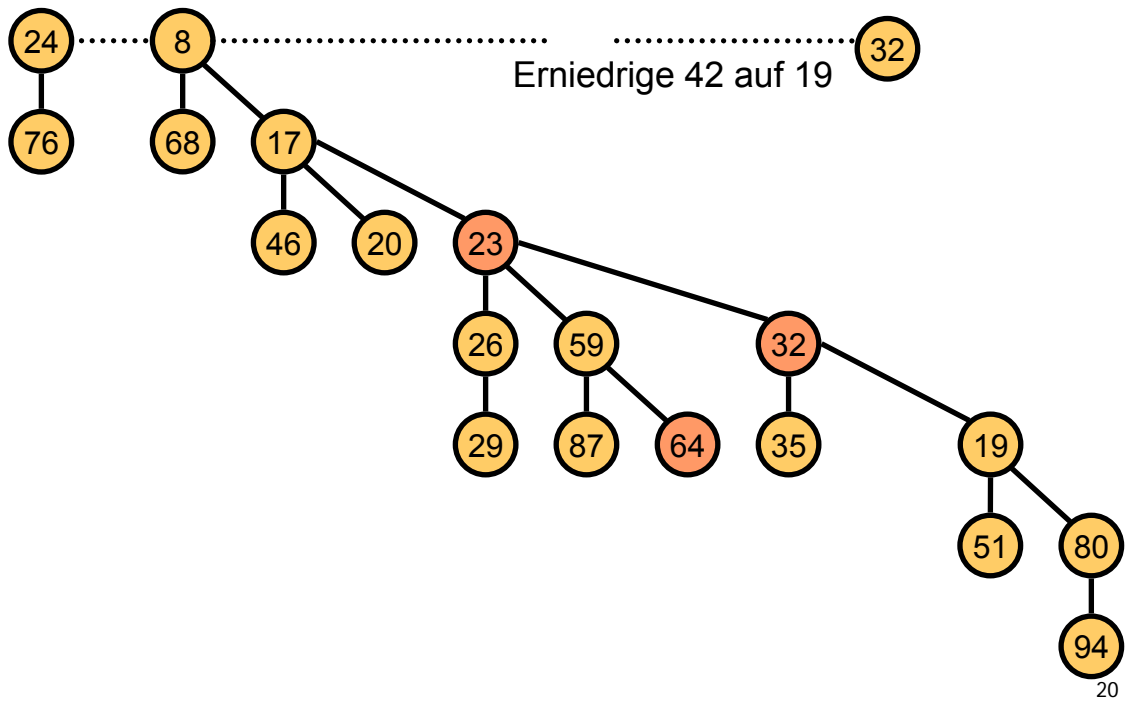


18

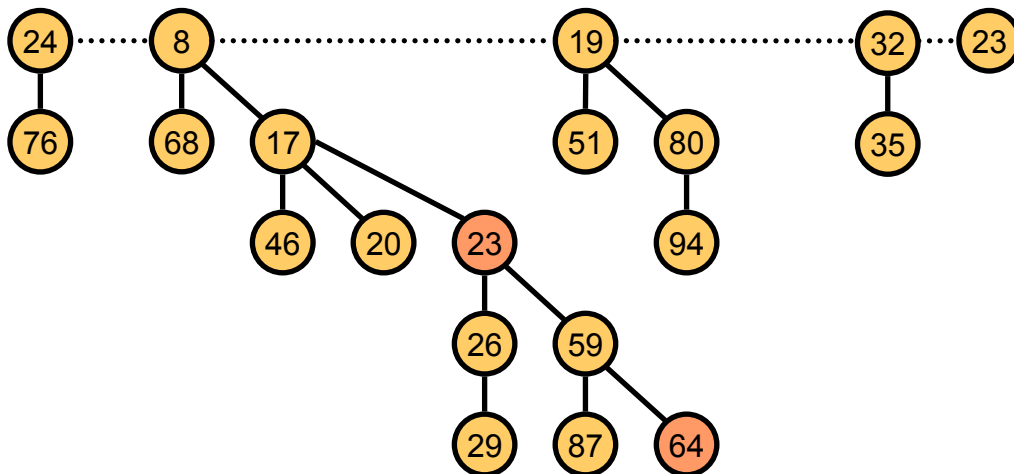
## Fibonacci-Heaps: DecreaseKey



## Fibonacci-Heaps: DecreaseKey



## Fibonacci-Heaps: DecreaseKey



Erniedrige 42 auf 19

## Fibonacci-Heaps: Analyse

Wir wollen jetzt zeigen, dass die Knotengrade im F-Heap durch  $O(\log n)$  beschränkt sind (deswegen auch die Markierung).

**Lemma:** Sei  $p$  ein Knoten vom Grad  $d$  und seien  $v_1, \dots, v_d$  die Kinder von  $p$  in der zeitlichen Reihenfolge, in der sie an  $p$  (durch Verschmelzen) angehängt wurden, dann gilt: Das  $i$ -te Kind von  $p$  hat mindestens Grad  $i-2$ .

**Beweis:** Tafel.

## Fibonacci-Heaps: Analyse

**Lemma:** Jeder Knoten  $p$  vom Grad  $k$  eines F-Heaps ist Wurzel eines Teilbaumes mit mindestens  $F_{k+2}$  Knoten.

$F_{k+2}$  ist die  $k+2$ -te Fibonaccizahl. Deswegen haben Fredman und Tarjan den Namen Fibonacci-Heap eingeführt.

**Lemma:** Jeder Knoten eines F-Heaps mit insgesamt  $N$  Knoten hat einen Grad von höchstens  $k \leq \log_{\phi} N$ .

**Folgerung:** Bei ExtractMin wird die Wurzelliste um höchstens  $O(\log N)$  Wurzeln verlängert.

## Fibonacci-Heaps: Analyse

- Oops:  $n$  mal `insert`, 1 `ExtractMin`  $\rightarrow \Theta(n)$ 
  - Worst case. Schlechter als binärer Heap!
  - Aber selten so schlecht.
  - Intuition: `Insert` baut Spannung/Potenzial auf, das `ExtractMin` verlangsamt, sich aber dabei abbaut. Nachfolgende `ExtractMin` haben es dafür leichter...
  - Potenzial baut sich nur langsam auf (über  $n$  Einfügeoperationen)  $\rightarrow$  durchschnittliche Kosten über alle Operationen  $\mathbf{O(1)}$ .
- Wir brauchen eine amortisierte Analyse.
  - Nichts für eine Realzeitumgebung.

## Fibonacci-Heaps: Amortisierte Analyse

- Führt man eine beliebige Folge von Operationen aus, dann ist die dafür insgesamt benötigte Zeit beschränkt durch die gesamte amortisierte Zeit;
- die amortisierte Zeit einer einzelnen **ExtractMin** Operation ist  $O(\log N)$ , die amortisierte Zeit aller anderen Operationen ist  $O(1)$ .

## Amortisierte Analyse

- Dazu Potenzial mathematisch beschreiben:
  - Funktion  $\Phi$ , die jeden Heap auf eine reelle Zahl abbildet
- Amortisierte Kosten einer Operation sind
  - tatsächliche Kosten +  $\Delta\Phi$
  - also auch Belastung durch zukünftige Kosten (**Insert**) oder Entschärfung von teuren Operationen ( $\Delta\Phi$  negativ).
- Sei  $\mathbf{o}_1, \dots, \mathbf{o}_n$  eine Folge von  $n$  Operationen
  - $\mathbf{a}_i$  amortisierte Kosten für Operation  $\mathbf{o}_i$
  - $\mathbf{t}_i$  tatsächliche Kosten für Operation  $\mathbf{o}_i$
  - $\Phi_i$  Potenzial direkt nach Operation  $\mathbf{o}_i$
  - $\Phi_0$  Potenzial vor  $\mathbf{o}_1$

## Amortisierte Analyse

- Dann ist die Summe der amortisierten Kosten

$$\sum_{i=1}^n a_i = \sum_{i=1}^n (t_i + \Phi_i - \Phi_{i-1}) = \Phi_n - \Phi_0 + \sum_{i=1}^n t_i$$

- Wir wählen eine Potenzialfunktion, die
  - nichtnegativ und
  - am Anfang gleich Null ist
- Dann ist  $\Phi_n - \Phi_0 \geq 0$  und damit  $\sum t_i \leq \sum a_i$

## Amortisierte Analyse

- Konkrete Wahl von  $\Phi$ ?
  - Was kann später große Kosten verursachen?
  - Sicher Anzahl der Wurzeln  $\mathbf{W}$ !
  - Provisorisch:  $\Phi = \alpha \mathbf{W}$

## Amortisierte Analyse: ExtractMin

- fängt mit  $W_1$  Wurzeln an, hört mit  $W_2$  auf, entfernter Knoten hatte Grad  $d$
- tatsächliche Kosten  $t_i = c(d + W_1)$ ,  $c$  Konstante
- $\Delta\Phi = \alpha(W_2 - W_1)$ , also amortisierte Kosten:  
 $a_i = c(d + W_1) + \alpha(W_2 - W_1) = (c - \alpha)W_1 + cd + \alpha W_2$
- Wir wählen  $\alpha = c$ , also  $a_i = cd + \alpha W_2$ 
  - aber  $d = O(\log n)$  und  $W_2 = O(\log n)$ , denn alle Wurzeln haben unterschiedlichen Grad
- also  $a_i = O(\log n)$

## Amortisierte Analyse: DecreaseKey

- schafft  $k$  neue Wurzeln
  - tatsächliche Kosten  $t_i = c'(k + 1)$ ,  $c'$  Konstante
- Problem: teuer und erhöht Potenzial...
- Zweite Quelle für  $\Phi$ : markierte Knoten  $A$ 
  - Werden leicht zu Wurzeln und verursachen dabei noch Arbeit, also  $\Phi = \alpha W + \beta A$
- $\Delta\Phi \leq \alpha k + \beta(2 - k)$ , denn alle neuen Wurzeln waren markiert (bis auf vielleicht eine), höchstens ein Knoten wird neu markiert
- amortisierte Kosten
$$a_i = c'(k + 1) + \alpha k + \beta(2 - k) = (c' + \alpha - \beta)k + c' + 2\beta$$
- Wähle  $\beta = c' + \alpha$ , es folgt  $a_i = 3c' + 2\alpha = O(1)$

## Amortisierte Analyse: Insert

- Müssen wir **ExtractMin** neu abschätzen?
  - Nein, denn dort verlieren höchstens Knoten ihre Markierung
- **Insert**:
  - Erhöht die Anzahl der Wurzeln um 1
  - $a_i = t_i + \alpha = O(1 + \alpha) = O(1)$
- Klar, dass  $\Phi$  nichtnegativ und am Anfang **0**

## Experimenteller Vergleich

- f\_heap: Fibonacci-Heaps
  - p\_heap: Pairing Heaps
  - k\_heap: k-ary Heaps
  - bin\_heap: Binary Heaps
  - list\_pq: lineare unsortierte Liste
  - r\_heap: Redistributive Heaps (AMOT90)\*
  - m\_heap: Monotone Heaps\*
- 
- \*: nur monotone Heaps
  - die letzten drei Varianten nur Integer in best. Bereich

## Experimenteller Vergleich

- Mehlhorn & Näher: LEDA Buch 2000
- Vergleich innerhalb von Dijkstra
- Je 1 Graph pro Datenpunkt (!)
- Knotenzahl mit je 2000 (s), 20.000 (m) und 200.000 (l)
- je 4 verschiedene Instanzen für s, m und l
- Kantengewichte random (r) aus  $[0..M-1]$  für M=100 (S) und M=1000 (L)
- Kantengewichte (w) mit  $c=0$  (S) und  $c=10.000$  (L)
  
- Gute Erfolge werden mit Bucket-Heaps und HOT-Queues erzielt (s. experimentelle Studie Übung)

