Algorithmen und Datenstrukturen

Prioritätswarteschlangen mit Fibonacci-Heaps
Amortisierte Analyse

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Danksagung

Die nachfolgenden Präsentationen wurden mit einigen Änderungen übernommen aus der Vorlesung „Effiziente Algorithmen und Datenstrukturen“ (Kapitel 2: Priority Queues) gehalten von Christian Scheideler an der TUM

http://www14.in.tum.de/lehre/2008WS/ea/index.html.de

Amortisierte Analyse

- S: Zustandsraum einer Datenstruktur
- F: beliebige Folge von Operationen $<Op_1, Op_2, Op_3, \ldots, Op_n>$
- s_0: Anfangszustand der Datenstruktur

$$s_0 \xrightarrow{Op_1} s_1 \xrightarrow{Op_2} s_2 \xrightarrow{Op_3} \ldots \xrightarrow{Op_n} s_n$$

- Zeitaufwand $T(F) = \sum_{i=1}^{n} T_{Op_i}(s_{i-1})$

Amortisierte Analyse

Zeitaufwand $T(F) = \Sigma_{i=1}^{n} T_{Op_i}(s_{i-1})$

Für den Zeitbedarf definieren wir eine Menge von Abschätzungsfunktionen $A_{Op}(s)$, eine pro Operation Op

Die Menge $\{ A_X(s) \mid X \in \{Op_1, Op_2, Op_3, \ldots, Op_n\}$ heißt Familie amortisierter Zeitschranken falls für jede Sequenz F von Operationen gilt

$$T(F) = \Sigma_{i=1}^{n} T_{Op_i}(s_{i-1}) \leq c + \Sigma_{i=1}^{n} A_{Op_i}(s_{i-1})$$

für eine Konstante c unabhängig von F

Amortisierte Analyse: Potentialmethode

Behauptung: Sei S der Zustandsraum einer Datenstruktur, sei s_0 der Anfangszustand und sei ϕ:S $\rightarrow R_{\geq 0}$ eine nichtnegative Funktion.

$\phi : S \rightarrow R_{\geq 0}$ wird auch Potential genannt.

Für eine Operation X und einen Zustand s mit $s \xrightarrow{X} s'$ definiere $A_X(s)$ über die Potentialdifferenz:

$$A_X(s) := \phi(s') - \phi(s) + T_X(s) := \Delta\phi(s) + T_X(s)$$

Dann sind die Funktionen $A_X(s)$ eine Familie amortisierter Zeitschranken.

Amortisierte Analyse: Potentialmethode

Zu zeigen: $T(F) \leq c + \Sigma_{i=1}^{n} A_{Op_i}(s_{i-1})$

Beweis:

$$\Sigma_{i=1}^{n} A_{Op_i}(s_{i-1}) = \Sigma_{i=1}^{n} [\phi(s_i) - \phi(s_{i-1}) + T_{Op_i}(s_{i-1})]$$

$$= T(F) + \Sigma_{i=1}^{n} [\phi(s_i) - \phi(s_{i-1})]$$

$$= T(F) + \phi(s_n) - \phi(s_0)$$

$$T(F) = \phi(s_0) - \phi(s_n) + \Sigma_{i=1}^{n} A_{Op_i}(s_{i-1})$$

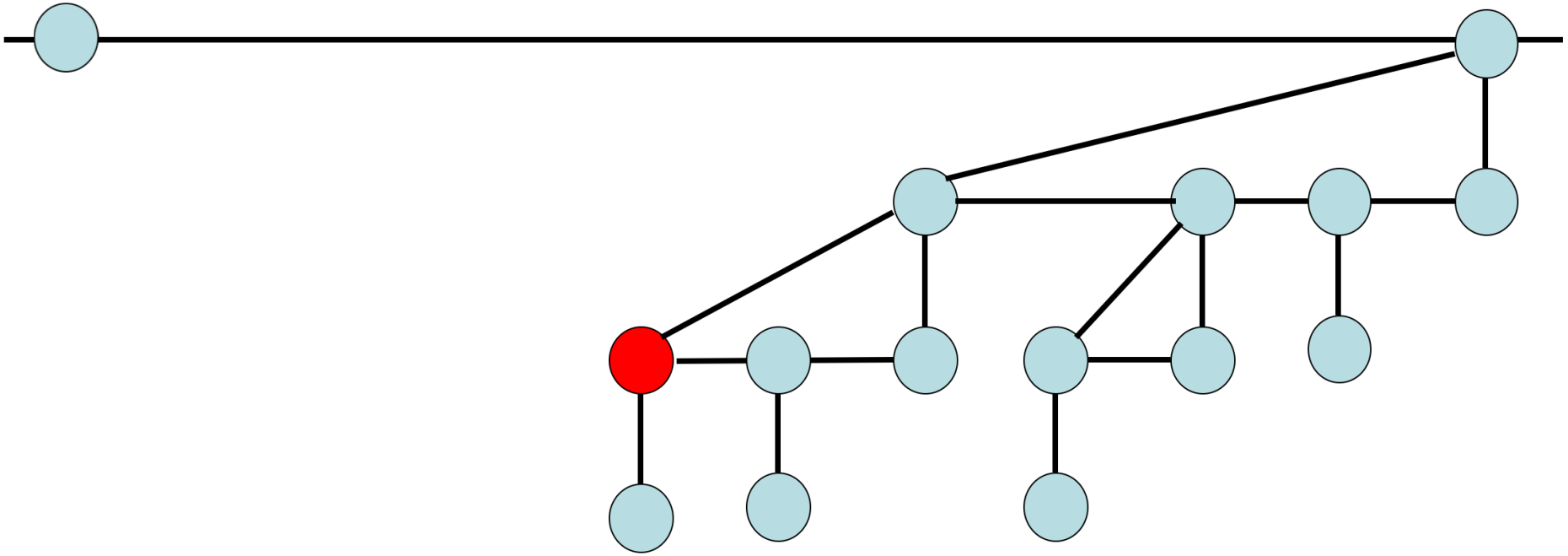
$$\leq \phi(s_0) + \Sigma_{i=1}^{n} A_{Op_i}(s_{i-1})$$

konstant

Amortisierte Analyse: Potentialmethode

Für Fibonacci-Heaps verwenden wir für das Potential den Begriff Balance (bal)

bal(s):= #Bäume + 2·#markierte Knoten im Zustand s



Fibonacci-Heap: insert, merge, min

- t_i: Zeit für Operation i ausgeführt im Zustand s
- a_i: amortisierter Aufwand für Operation i
  $a_i = t_i + \Delta bal_i$ mit $\Delta bal_i = bal(s')-bal(s)$, falls $i:s \rightarrow s'$ im aktuellen Zustand s ausgeführt wird

Amortisierte Kosten der Operationen:
$bal(s) = \#Bäume(s) + 2 \cdot \#markierte\ Knoten(s)$

- insert: $t=O(1)$ und $\Delta bal_{insert} = +1$, also $a_{insert}=O(1)$
- merge: $t=O(1)$ und $\Delta bal_{merge} = 0$, also $a_{merge}=O(1)$
- min: $t=O(1)$ und $\Delta bal_{min} = 0$, also $a_{min}=O(1)$
- deleteMin: ???

Fibonacci-Heap

- Inv:
- Ein Fibonacci-Heap aus n Elementen hat
  Bäume vom Rang maximal O(log n)
  (wie beim Binomial-Heap) auch, wenn durch delete einige Knoten entfernt wurden

Fibonacci-Heap: Eigenschaften

Invariante 1: Sei x ein Knoten im Fibonacci-Heap mit Rang(x)=k. Seien die Kinder von x sortiert in der Reihenfolge ihres Anfügens an x. Dann ist der Rang des i-ten Kindes ≥i-2.

Beweis der Gültigkeit:

- Beim Einfügen des i-ten Kindes ist Rang(x)=i-1.
- Das i-te Kind hat zu dieser Zeit auch Rang i-1.
- Danach verliert das i-te Kind höchstens eines seiner Kinder[1], d.h. sein Rang ist ≥i-2.

[1] Bei einem schon markierten Vater eines gelöschten Knotens wird konsolidiert

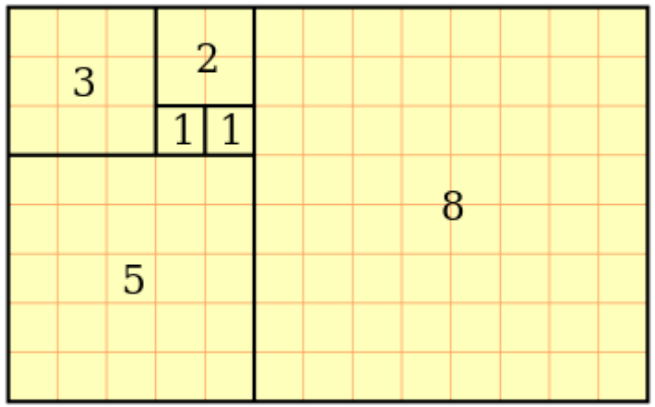
Fibonacci-Heap: Eigenschaften

Invariante 2: Sei x ein Knoten im Fibonacci-Heap mit Rang(x)=k. Dann enthält der Baum mit Wurzel x mindestens F_{k+2} Elemente, wobei F_{k+2} die (k+2)-te Fibonacci-Zahl ist.

Einschub: Definition der Fibonacci-Zahlen:

- $F_0 = 0$ und $F_1 = 1$
- $F_i = F_{i-2} + F_{i-1}$ für alle $i > 1$

Daraus folgt, dass $F_{i+2} = 1 + \Sigma_{j=0}^{i} F_j$



[Wikipedia]

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, usw.

$$F_{i+2} = 1 + \Sigma_{j=0}^{i} F_j \qquad F_i = F_{i-2} + F_{i-1}$$

$$F_{i+2} = F_i + F_{i+1}$$

$$F_{i+2} = F_i + F_{i-1} + F_i$$

$$F_{i+2} = F_i + F_{i-1} + F_{i-2} + F_{i-1}$$

$$F_{i+2} = F_i + F_{i-1} + F_{i-2} + F_{i-3} + F_{i-2}$$

$F_0 = 0$ und $F_1 = 1$

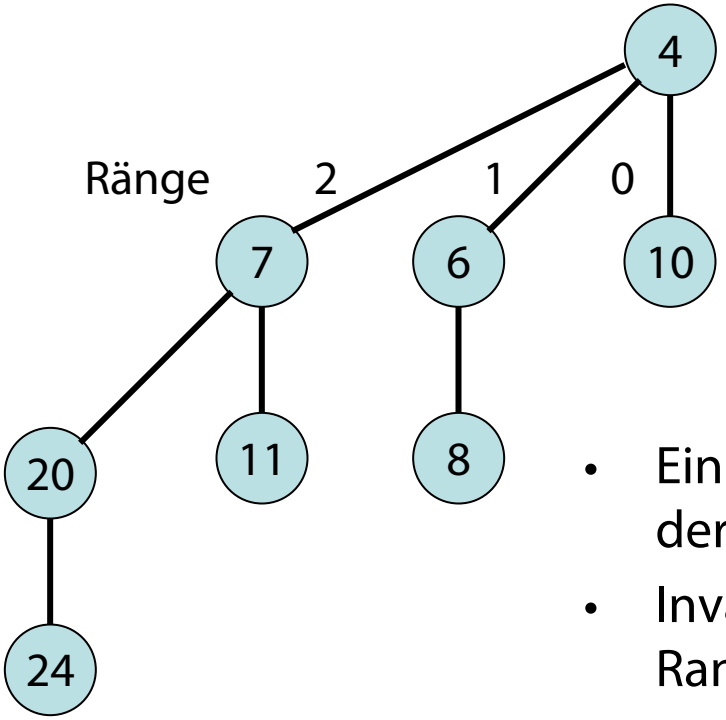
....

$$F_{i+2} = F_i + F_{i-1} + F_{i-2} + F_{i-3} + F_{i-2} + \ldots\ldots\ldots + F_2 + F_1 + F_2$$

$$F_{i+2} = \underbrace{F_i + F_{i-1} + F_{i-2} + F_{i-3} + F_{i-2} + \ldots\ldots\ldots + F_2 + F_1 + F_0}_{\Sigma_{j=0}^{i} F_j} + \underbrace{F_1}_{1}$$

$$F_{i+2} = 1 + \Sigma_{j=0}^{i} F_j$$

Rang eines Knotens im Binomialbaum



Rang 3

- Ein Binomialbaum vom Rang k hat k Kinder, deren Ränge sind k-1, k-2, …. 1, 0
- Invariante 1:
  Rang des i-ten Kindes im Fib. Heap ≥i-2
- Sei f_k die Mindestanzahl von Knoten eines Baumes vom Rang k
- $f_{k-1} + f_{k-2} + f_{k-3} + \ldots + f_2 + f_1 + f_0 + 1 \leq f_k$

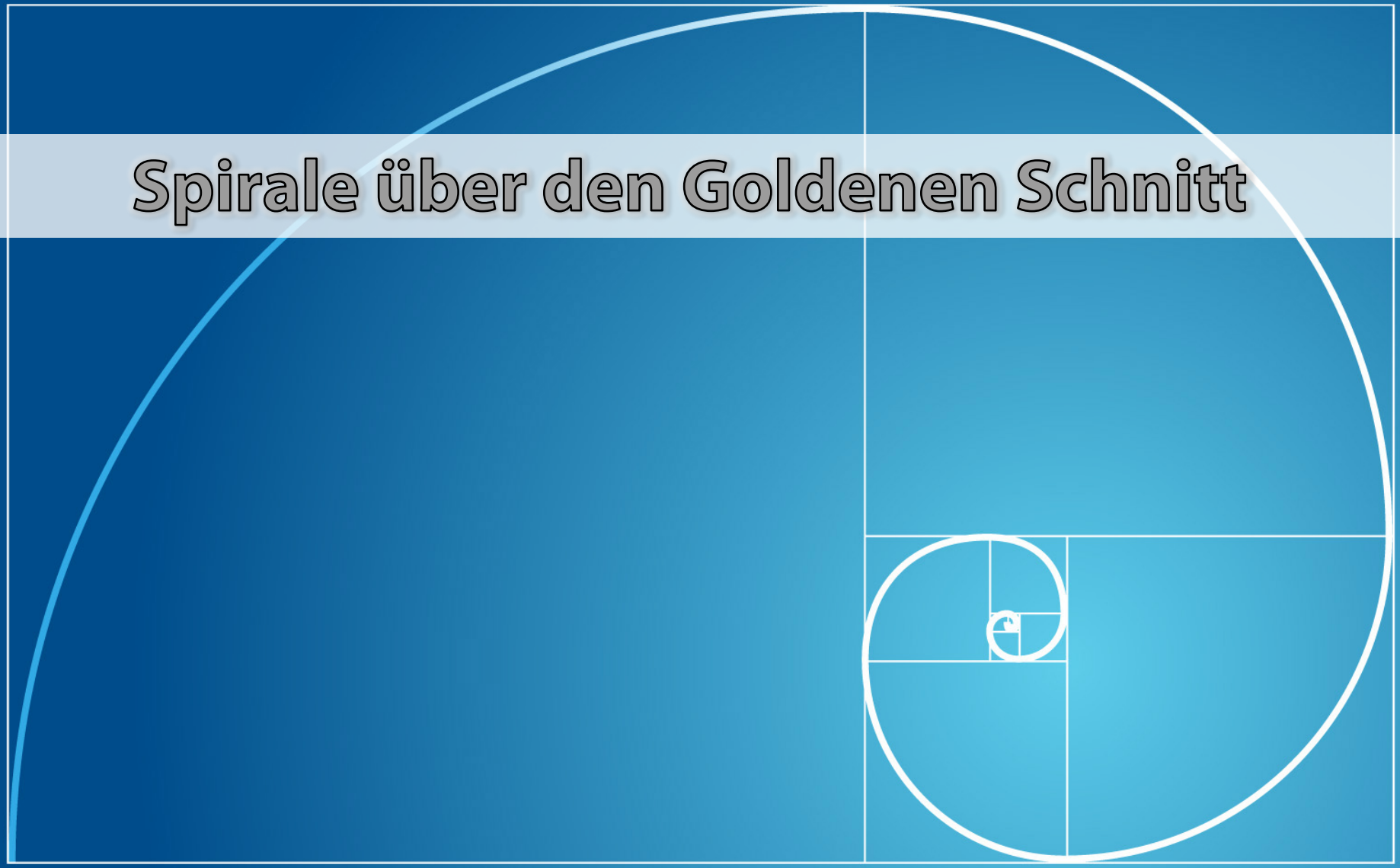
Fibonacci-Heap

Beweis der Gültigkeit von Invariante 2:

- Sei f_k die Mindestanzahl von Knoten eines Baumes vom Rang k
- $f_k \geq f_{k-1} + f_{k-2} + f_{k-3} + \ldots + f_2 + f_1 + f_0 + 1$
- Weiterhin ist $f_0=1$ und $f_1=2$
- $f_k \geq f_{k-1} + f_{k-2} + f_{k-3} + \ldots + f_2 + 2 + 1 + 1$
- $f_k \geq f_{k-1} + f_{k-2} + f_{k-3} + \ldots + f_2 + 2 + 1 + 0 + 1$
- $\geq F_k + F_{k-1} + F_{k-2} + \ldots + F_3 + F_2 + F_1 + F_0 + 1$
- $= 1 + \Sigma_{j=0}^{k} F_j = F_{k+2}$
- Also folgt nach den Fibonacci-Zahlen:

$$f_k \geq F_{k+2}$$

Spirale über den Goldenen Schnitt



$$\frac{a+b}{a} = \frac{a}{b} = \varphi \approx 1{,}61803$$

aus: [Wikipedia]

Verwandtschaft mit dem Goldenen Schnitt [Bearbeiten]

Wie von Johannes Kepler festgestellt wurde, nähert sich der Quotient zweier aufeinander folgender Fibonacci-Zahlen dem Goldenen Schnitt Φ an. Dies folgt unmittelbar aus der Näherungsformel für große *n*:

$$\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \lim_{n\to\infty} \frac{\Phi^{n+1}}{\Phi^n} = \Phi \approx 1{,}618\ldots$$

Diese Quotienten zweier aufeinander folgender Fibonacci-Zahlen haben eine bemerkenswerte Kettenbruchdarstellung

$$\frac{1}{1} = 1 \qquad \frac{2}{1} = 1+\frac{1}{1} \qquad \frac{3}{2} = 1+\frac{1}{1+\frac{1}{1}} \qquad \frac{5}{3} = 1+\frac{1}{1+\frac{1}{1+\frac{1}{1}}} \qquad \frac{8}{5} = 1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1}}}}$$

Da diese Quotienten im Grenzwert gegen den goldenen Schnitt konvergieren, lässt sich dieser als der unendliche Kettenbruch

$$\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}}$$

darstellen.

Die Zahl Φ ist irrational. Das bedeutet, dass sie sich nicht durch ein Verhältnis zweier ganzer Zahlen darstellen lässt, ein Umstand, der wesentlich zu ihrer Bedeutung in Kunst und Natur beiträgt. Am besten lässt sich Φ durch Quotienten zweier aufeinander folgender Fibonacci-Zahlen darstellen. Dies gilt auch für verallgemeinerte Fibonaccifolgen, bei denen f_0 und f_1 beliebige natürliche Zahlen annehmen.

Fibonacci-Heap

- Man hat herausgefunden, dass $F_k > \Phi^k$ ist für

$$\Phi=(1+\sqrt{5})/2 \approx 1{,}618034$$

- $f_k \geq F_{k+2} \geq \Phi^{k+2}$
- D.h. ein Baum mit Rang k im Fibonacci-Heap hat mindestens $1{,}61^{k+2}$ Knoten
- Sei $n = 1{,}61^{k+2}$ dann gilt: $n/1{,}61^2 = 1{,}61^k$
- Also $k \in O(\log n)$

NB: $\log(n)$ bezeichnet den Zweierlogarithmus ($\log(64) = 6$). Bei O-Notation spielt die Basis des Logarithmus keine Rolle, da alle Logarithmen proportional sind. Z.B.: $\log(n) = \ln(n)/\ln(2)$.

Fibonacci-Heap

- Also $k \in O(\log n)$
- Ein Fibonacci-Heap aus n Elementen hat also Bäume vom Rang maximal $O(\log n)$ (wie beim Binomial-Heap)

Fibonacci-Heap: deleteMin

Behauptung:
Die amortisierten Kosten von deleteMin() sind in O(log n).

Beweis:

- Einfügen der Kinder von x in Wurzelliste (#Kinder(x) = Rang(x)):
  $\Delta bal_1 =$ Rang(x) – 1 (-1 weil x entfernt wird)
- Jeder Merge-Schritt verkleinert #Bäume um 1:
  $\Delta bal_2 =$ -(#Merge-Schritte)
- Wegen Inv (Rang der Bäume max. O(log n))
  gilt: #Merge-Schritte = #Bäume – O(log n)
- Insgesamt: $\Delta bal_{deleteMin}$ = Rang(x) – 1 - #Bäume + O(log n)
- Laufzeit (in geeigneten Zeiteinheiten):
  $t_{deleteMin}$ = #Bäume[1] + O(log n)
- Amortisierte Laufzeit:
  $a_{deleteMin} = t_{deleteMin} + \Delta bal_{deleteMin} \in O(\log n)$

[1] Realisierung der Eimerkette

Fibonacci-Heap: delete

Behauptung: Die amortisierten Kosten von delete(x) sind O(log n).

Beweis: (x ist kein min-Element – sonst wie oben)

- Einfügen der Kinder von x in Wurzelliste:
  $\Delta bal_1 \leq$ Rang(x)
- Jeder kaskadierende Schritt (Entfernung eines markierten Knotens) erhöht die Anzahl Bäume um 1:
  $\Delta bal_2 =$ #kaskadierende Schritte
- Jeder kaskadierende Schritt entfernt eine Markierung:
  $\Delta bal_3 = -2 \cdot$#kaskadierende Schritte
- Der letzte Schritt von delete erzeugt evtl. eine Markierung:
  $\Delta bal_4 \in \{0,2\}$

Fibonacci-Heap: delete (Forts.)

Behauptung: Die amortisierten Kosten von delete(x) sind O(log n).

Beweis (Fortsetzung):

- Insgesamt:
  Δbal_{delete} = Rang(x) - #kaskadierende Schritte + O(1)
  = O(log n) - #kaskadierende Schritte
- Laufzeit (in geeigneten Zeiteinheiten):
  t_{delete} = O(1) + #kaskadierende Schritte
- Amortisierte Laufzeit:
  $a_{delete} = t_{delete} + \Delta bal_{delete} \in O(\log n)$

Fibonacci-Heap: decreaseKey

Behauptung: Die amortisierten Kosten von decreaseKey(x,Δ) sind O(1).

Beweis:

- Jeder kask. Schritt erhöht die Anzahl Bäume um 1:
  Δbal_1 = #kaskadierende Schritte
- Jeder kask. Schritt entfernt eine Markierung (bis auf x):
  $\Delta bal_2 \leq -2 \cdot$(#kaskadierende Schritte-1)
- Der letzte Schritt erzeugt evtl. eine Markierung:
  $\Delta bal_3 \in \{0,2\}$
- Insgesamt: $\Delta bal_{decreaseKey} = -$ #kask. Schritte + O(1)
- Laufzeit: $t_{decreaseKey}$ = #kask. Schritte + O(1)
- Amortisierte Laufzeit:
  $a_{decreaseKey} = t_{decreaseKey} + \Delta bal_{decreaseKey} = O(1)$

Zusammenfassung: Laufzeitvergleich

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

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**

Zusammenfassung: Laufzeitvergleich

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

Weitere Entwicklung unter Ausnutzung von Dateneigenschaften: Radix-Heap

Radix-Heap (nur Analyse)

Voraussetzungen [Bearbeiten]

1. alle Schlüssel sind aus den natürlichen Zahlen
2. max. Schlüssel - min. Schlüssel $\leq$ C für ein festes C
3. Monotonie von *extractMin*, d.h. die von aufeinander folgenden *extractMin*-Aufrufen zurückgegebenen Werte sind monoton steigend

Laufzeit	Radix-Heap	erw. Radix-Heap
insert	O(log C) amor.	O(log C) amor.
min	O(1)	O(1)
deleteMin	O(1) amor.	O(1) amor.
delete	O(1)	O(1) amor.
merge	n/a	O(log C) amor.
decreaseKey	O(1)	O(log C) amor.

Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E., Faster algorithms for the shortest path problem, Journal of the Association for Computing Machinery 37 (2): 213–223, **1990**

Es geht auch ohne amortisierte Analyse …

operation	linked list	binary heap	binomial heap	pairing heap †	Fibonacci heap †	Brodal queue
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
IS-EMPTY	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$
DELETE-MIN	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
DECREASE-KEY	$O(1)$	$O(\log n)$	$O(\log n)$	$2^{\sqrt{O(\log \log n)}}$	$O(1)$	$O(1)$
DELETE	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
MERGE	$O(1)$	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$
MIN	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$

† amortized

Fredman, Michael L.; Sedgewick, Robert; Sleator, Daniel D.; Tarjan, Robert E. The pairing heap: a new form of self-adjusting heap. Algorithmica. 1 (1): 111–129, **1986**.

Gerth Stølting Brodal, Worst-case efficient priority queues. Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, pp. 52–58, **1996**

Gerth Stølting Brodal, George Lagogiannis, and Robert E. Tarjan, STRICT FIBONACCI HEAPS, In Proc. 44th Annual ACM Symposium on Theory of Computing, pages 1177-1184, **2012**.

Informatik als Wissenschaft

70 Jahre Entwicklung und Analyse von Algorithmen und Datenstrukturen