Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

July 5, 2022

Lehrstuhl Informatik 2
Fakultät für Informatik

Binomial heaps

Worum geht es?

- der Dijkstra-Algorithmus hat quadratische Laufzeit
- Flaschenhals ist die Berechnung des Knotens mit minimalen Abstand
- mit dem binomial heap lernen wir eine erste Datenstruktur kennen, die diese Operation beschleunigt

Binomial heaps

Binomial heap: Überblick

einfügen: neues Element mit gegebenen Gewicht hinzufügen

Minimum: auffinden des Elements mit minimalen Gewicht

Minimum entnehmen: auffinden und entfernen

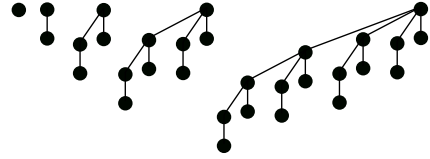
Vereinigung: zwei Heaps zu einem vereinigen

verringern: das Gewicht eines Elements verringern

löschen: ein Element aus der Datenstruktur entfernen

Alle Operationen haben Laufzeit $O(\log n)$

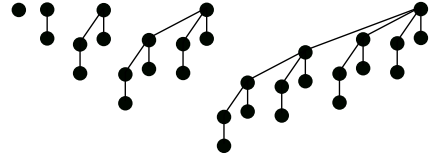
Binomial heaps



Binomialbäume

- ein Binomialbaum ist ein geordneter, gewurzelter Baum
- d.h. ein Knoten ist als Wurzel ausgezeichnet
- die Kinder jedes Knotens sind geordnet

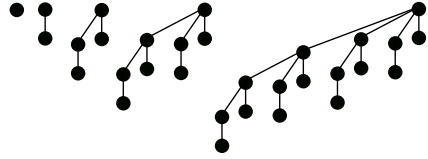
Binomial heaps



Binomialbäume

- zu jeder Ordnung $k \geq 0$ gibt es genau einen Binomialbaum, bis auf Isomorphie
- der Binomialbaum der Ordnung 0 besteht nur aus einem Knoten
- der Baum der Ordnung $k \geq 1$ entsteht aus zwei Bäumen der Ordnung $k - 1$
- der erste Baum wird links an die Wurzel des zweiten Baums angehängt wird

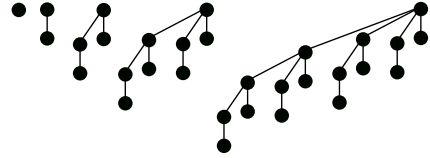
Binomial heaps



Binomialbäume

- im Baum der Ordnung k hängt an der Wurzel also jeweils ein Baum der Ordnung $0, 1, \ldots, k-1$

Binomial heaps



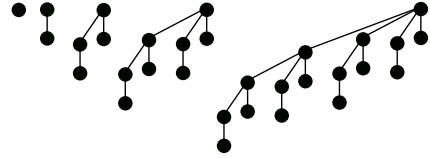
Lemma

Der Binomialbaum B_k der Ordnung $k \geq 0$ hat folgende Eigenschaften

- die Zahl der Knoten ist 2^k
- die Höhe des Baums ist k
- es gibt genau $\binom{k}{h}$ Knoten auf Tiefe h für $h = 0, \ldots, k$
- die Wurzel hat Grad $k = \Delta(B_k)$

Die Tiefe eines Knotens ist definiert als der Abstand von der Wurzel.

Binomial heaps

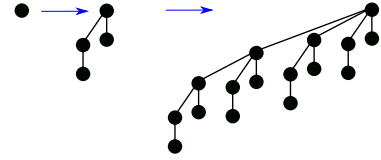


Beweis

- Induktion nach k
- im Fall $k = 0$ gelten die gewünschten Eigenschaften
- für $k > 0$ folgen die Behauptungen zu Knotenzahl, Höhe und Grad der Wurzel unmittelbar aus der Induktionsvoraussetzung
- die Zahl der Knoten auf Tiefe $0 \leq h \leq k$ berechnet sich nach Induktion zu

$$\binom{k}{h} + \binom{k}{h-1} = \binom{k+1}{h}$$

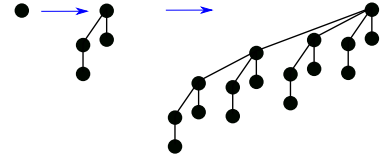
Binomial heaps



Binomial heap: Konstruktion

- ein binomial heap ist eine verkettete Liste von Binomialbäumen
- genauer sei $n \geq 1$ eine ganze Zahl mit Binärdarstellung $n = \sum_{j=0}^{\ell} b_j 2^j$ mit $b_j \in \{0, 1\},\ b_\ell = 1$
- dann enthält der binomial heap der Ordnung n genau b_j Binomialbäume der Ordnung j für $j = 0, \ldots, \ell$

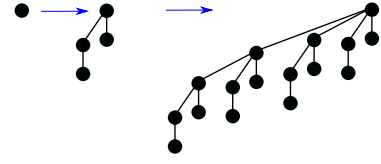
Binomial heaps



Binomial heap: Konstruktion

- diese Binomialbäume bilde eine der Ordnung nach aufsteigend geordnete verkettete Liste
- jeder einzelne Binomialbaum besteht aus einem Zeiger auf die Wurzel
- jeder Knoten ausser der Wurzel verfügt über einen Zeiger auf seinen Elternknoten
- ferner besitzt jeder Knoten einen Zeiger auf den nächsten Geschwisterknoten

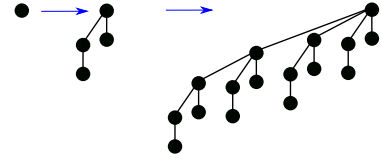
Binomial heaps



Binomial heap: Konstruktion

- der binomial heap besteht also aus $O(\log n)$ Bäumen
- die maximale Tiefe dieser Bäume ist $O(\log n)$

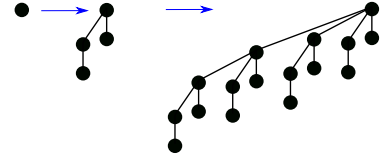
Binomial heaps



Binomial heap: Konstruktion

- innerhalb jedes Baumes ist das Gewicht des Elternknotens immer kleiner oder gleich dem Gewicht jedes Kindknotens
- insbesondere hat der Wurzelknoten das geringste Gewicht
- die Gewichte der Wurzelknoten im binomial heap sind nicht notwendigerweise geordnet

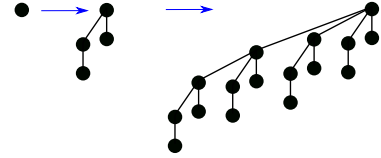
Binomial heaps



Binomial heap: Minimum finden

- das Element minimalen Gewichts finden wir, indem wir einfach die Wurzelknoten der Bäume des binomial heap durchsuchen
- das funktioniert, weil in jedem einzelen binomial tree die Wurzel das Element minimalen Gewichts ist
- Zeitbedarf: $O(\log n)$

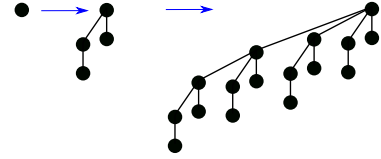
Binomial heaps



Binomial heap: Vereinigung

- um zwei binomial heaps B_1, B_2 zu vereinigen, fügen wir zunächst die Listen der Binomialbäume zusammen
- beim Zusammenfügen wird die Ordnung beibehalten
- es treten also zu jeder Ordnung j höchsts zwei Bäume der Ordnung j auf
- beim Vereinigungsprozess kann ein dritter Baum einer gegebenen Ordnung entstehen

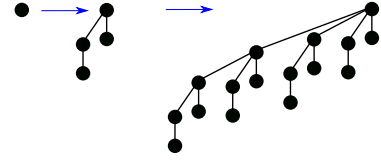
Binomial heaps



Binomial heap: Vereinigung

- anschließend gehen wir die vereinige Liste der Bäume durch
- wenn von einer Ordnung genau zwei Bäume vorkommen, vereinigen wir sie
- wenn von einer Ordnung drei Bäume vorkommen, behalten wir den ersten bei und vereinigen die beiden anderen

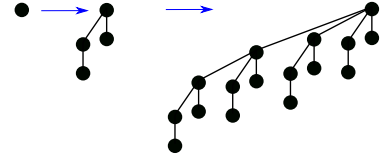
Binomial heaps



Binomial heap: Vereinigung

- dabei stellen wir sicher, daß die Wurzel das Element geringsten Gewichtes bleibt
- die Laufzeit für den Vereinigungsprozess liegt bei $O(\log n)$

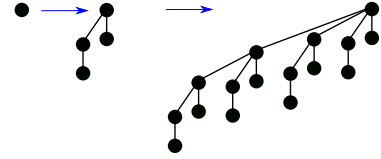
Binomial heaps



Binomial heap: Element einfügen

- zum Einfügen eines neuen Elementes in eine binomial heap H legen wir einen binomial heap H' der Ordnung 0 an, der *nur* das neue Element enthält
- dann bilden wir die Vereinigung von H und H'
- Laufzeit: $O(\log n)$

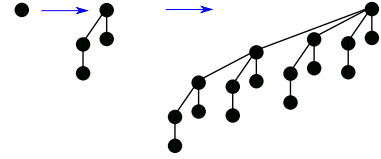
Binomial heaps



Binomial heap: Minimum entnehmen

- finde das Element x minimalen Gewichts in H
- dieses Element ist Wurzel eines Binomialbaums T
- bilde einen neuen binomial heap H', dessen Elemente die Kinder von T (in umgekehrter Reihenfolge) sind
- vereinige H' mit dem binomial heap $H'' = H - T$
- Laufzeit: $O(\log n)$

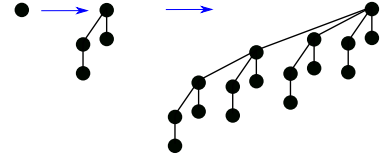
Binomial heaps



Binomial heap: Gewicht verringern

- um das Gewicht eines Elements x zu verringern, müssen nur die Elemente des Baums, dem x angehört, neu angeordnet werden
- genauer lassen wir x in Richtung Wurzel aufsteigen, bis alle Kinder von x mindestens das Gewicht von x haben
- Laufzeit $O(\log n)$, weil die Tiefe des Baums $O(\log n)$ ist

Binomial heaps



Binomial heap: löschen

- setze das Gewicht des zu löschenden Elements auf $-\infty$
- anschließend entfere das Minimum
- Laufzeit: $O(\log n)$

Binomial heaps

Anwendung auf Dijkstra

Kürzeste Pfade in einem Graphen $G = (V, E)$

- naiver Dijkstra hatte Laufzeit $O(|V|^2)$
- mit binomial heaps ist die Laufzeit $O(|E| \log |V|)$
- besser als naiver Dijkstra in “dünnen” Graphen

Binomial heaps

Zusammenfassung

- binomial heaps sind geeignet, um gewichtete Elemente zu speichern, wenn man an dem Element minimalen Gewichts interessiert ist
- Laufzeit aller Operationen ist $O(\log n)$
- *Anwendung:* Dijkstra