Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

July 13, 2023

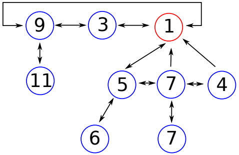
Lehrstuhl Informatik 2
Fakultät für Informatik

Fibonacciheaps

Worum geht es?

- der Dijkstra-Algorithmus hat quadratische Laufzeit
- Flaschenhals ist die Berechnung des Knotens mit minimalen Abstand
- mit dem Fibonacciheap lernen wir die effizienteste Datenstruktur für die Operation kennen
- wir wenden das Prinzip der amortisierten Laufzeitanalyse an

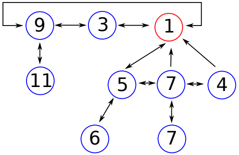
Fibonacciheaps



Aufbau

- der Fibonacci-Heap besteht aus gewurzelten Bäumen
- die Bäume sind ungeordnet (anders als im binomial heap)
- Gewicht eines Elternknotens $\leq$ Gewicht jedes Kindes
- die Liste der Wurzeln ist doppelt verkettet
- für jede Wurzel ist die Liste der Kinder doppelt verkettet
- Elemente löschen oder Listen kombinieren in Zeit $O(1)$

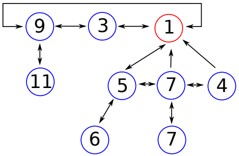
Fibonacciheaps



Aufbau

- der Heap hat eine Referenz auf das minimale Element
- die Bäume sind nicht notwendigerweise Binomialbäume
- mit $D(n)$ bezeichnen wir den maximalen Grad eines Fibonacci-Heaps mit Größe n
- wir werden zeigen, daß $D(n) = O(\log n)$
- jeder Knoten v hat ferner eine Markierung $\mu(v) \in \{0, 1\}$

Fibonacciheaps



Operationen

einfügen: neues Element mit gegebenen Gewicht hinzufügen

Minimum: auffinden des Elements mit minimalen Gewicht

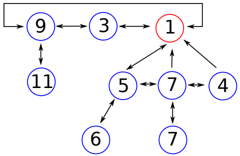
Minimum entnehmen: auffinden und entfernen

Vereinigung: zwei Heaps zu einem vereinigen

verringern: das Gewicht eines Elements verringern

löschen: ein Element aus der Datenstruktur entfernen

Fibonacciheaps

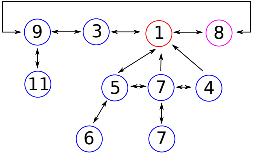


Potentialfunktion

- $T(\mathcal{F})$ bezeichet die Zahl der Bäume im Fibonacci-Heap $\mathcal{F}$
- $N(\mathcal{F})$ bezeichet die Zahl aller Knoten in $\mathcal{F}$
- $M(\mathcal{F})$ bezeichet die Zahl markierter Knoten v, d.h. $\mu(v) = 1$
- das Potential ist definiert als

$$\Phi(\mathcal{F}) = T(\mathcal{F}) + 2M(\mathcal{F})$$

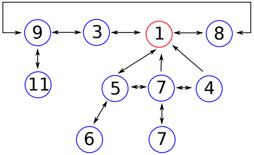
Fibonacciheaps



Einfügen

- wir fügen den neuen Eintrag einfach als neuen gewurzelten Baum, bestehend aus einem Knoten, in $\mathcal{F}$ ein
- ggf. wird das Minimum aktualisiert
- Laufzeit $O(1)$
- Potentialänderung $O(1)$

Fibonacciheaps



Minimum finden

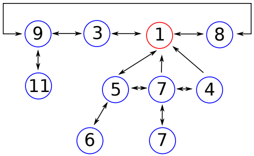
- der Fibonacci-Heap besitzt eine Referenz auf das minimale Element
- Laufzeit $O(1)$; Potentialänderung 0

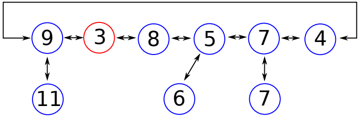
Fibonacciheaps

Vereinigung

- wir fügen einfach die Listen der Wurzeln zusammen
- weil sie doppelt verlinkt sind, geht das in Zeit $O(1)$
- außerdem wird das Minimum aktualisiert
- das Potential ändert sich nicht

Fibonacciheaps

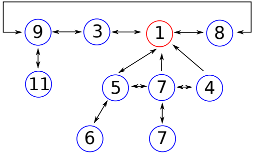


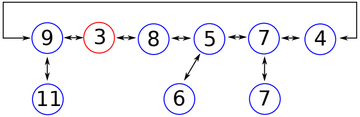


Minimum extrahieren

- das Element minmalen Gewichts wird gelöscht
- aus jedem Kind des min. Elements wird eine neue Wurzel
- anschließend wird der Heap konsolidiert
- dabei wird auch das Minimum aktualisiert

Fibonacciheaps

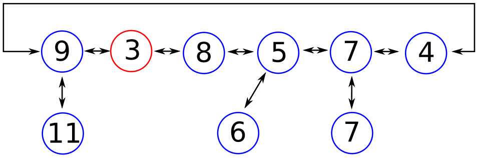




Konsolidieren

- zum Konsolidieren iterieren wir über die Wurzelliste
- wir suchen ein Paar von Wurzeln gleichen Grades
- dazu verwenden wir ein Array $A[0 \ldots D(N(\mathcal{F}))]$
- $A[i]$ verweist auf eine Wurzel vom Grad i (oder $\emptyset$)

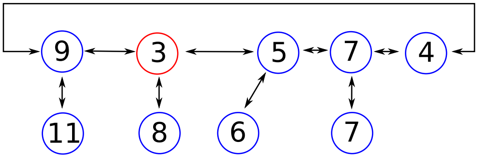
Fibonacciheaps



Konsolidieren

- wenn zwei Wurzeln gleichen Grades gefunden sind, wird die Wurzel mit größerem Gewicht wird zum Kind der anderen
- das Feld $\mu(v)$ der Wurzel v, die zum Kind der anderen wird, wird auf 0 gesetzt
- der Grad der anderen Wurzel wird um 1 erhöht
- wir prüfen, ob es eine andere Wurzel dieses neuen Grades schon gibt und wiederholen

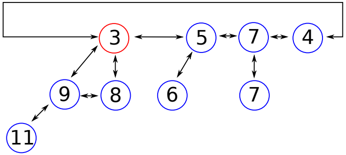
Fibonacciheaps



Konsolidieren

- wenn zwei Wurzeln gleichen Grades gefunden sind, wird die Wurzel mit größerem Gewicht wird zum Kind der anderen
- das Feld $\mu(v)$ der Wurzel v, die zum Kind der anderen wird, wird auf 0 gesetzt
- der Grad der anderen Wurzel wird um 1 erhöht
- wir prüfen, ob es eine andere Wurzel dieses neuen Grades schon gibt und wiederholen

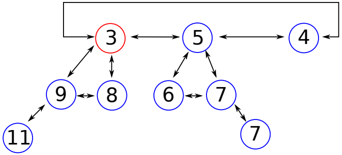
Fibonacciheaps



Konsolidieren

- wenn zwei Wurzeln gleichen Grades gefunden sind, wird die Wurzel mit größerem Gewicht wird zum Kind der anderen
- das Feld $\mu(v)$ der Wurzel v, die zum Kind der anderen wird, wird auf 0 gesetzt
- der Grad der anderen Wurzel wird um 1 erhöht
- wir prüfen, ob es eine andere Wurzel dieses neuen Grades schon gibt und wiederholen

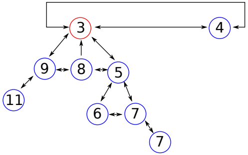
Fibonacciheaps



Konsolidieren

- wenn zwei Wurzeln gleichen Grades gefunden sind, wird die Wurzel mit größerem Gewicht wird zum Kind der anderen
- das Feld $\mu(v)$ der Wurzel v, die zum Kind der anderen wird, wird auf 0 gesetzt
- der Grad der anderen Wurzel wird um 1 erhöht
- wir prüfen, ob es eine andere Wurzel dieses neuen Grades schon gibt und wiederholen

Fibonacciheaps



Konsolidieren

- wenn zwei Wurzeln gleichen Grades gefunden sind, wird die Wurzel mit größerem Gewicht wird zum Kind der anderen
- das Feld $\mu(v)$ der Wurzel v, die zum Kind der anderen wird, wird auf 0 gesetzt
- der Grad der anderen Wurzel wird um 1 erhöht
- wir prüfen, ob es eine andere Wurzel dieses neuen Grades schon gibt und wiederholen

Fibonacciheaps

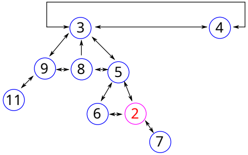
Lemma

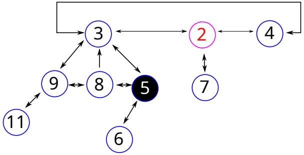
Die amortisierten Kosten zum Extrahieren des Minimums sind $O(D(N(\mathcal{F})))$.

Beweis

- sei $\mathcal{F}'$ der Fibonacci-Heap nach der Extraktion
- alle Wurzeln in $\mathcal{F}'$ haben verschiedene Grade
- also $\Phi(\mathcal{F}') \leq D(N(\mathcal{F})) + 1 + 2M(\mathcal{F})$
- folglich $\Phi(\mathcal{F}') - \Phi(\mathcal{F}) = O(D(N(\mathcal{F})))$

Fibonacciheaps

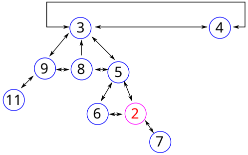


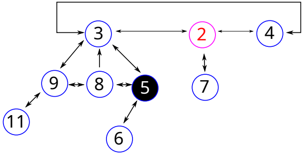


Gewicht verringern

- wir müssen darauf achten, daß die Ordnung erhalten bleibt,
- d.h. Kinder haben mindestens so großes Gewicht wie Eltern
- um den Grad zu begrenzen, trennen wir den Baum ggf. auf
- dazu verwenden wir die Markierungen
- $\mu(x) = 1$ zeigt an, daß x bereits ein Kind verloren hat

Fibonacciheaps

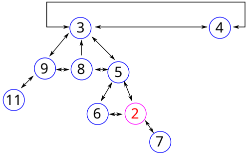


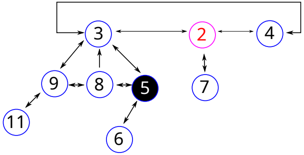


Gewicht verringern

- falls der Knoten x, dessen Gewicht reduziert wird, schon eine Wurzel ist, verringern wir einfach das Gewicht, passen ggf. das Minimum an und sind fertig
- wenn allgemeiner das Gewicht von x nicht kleiner wird als das Gewicht des Elternknotens y, gehen wir genauso vor
- andernfalls wende die folgende Trennoperation auf x an

Fibonacciheaps

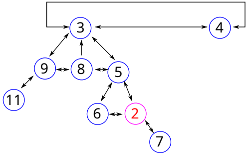


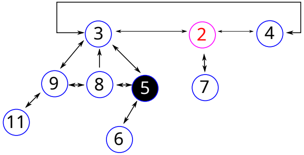


Trennen von x

- entferne x aus der Kinderliste von y
- füge x (und seine Nachkommen) als Wurzel ein
- aktualsiere das Minimum
- setze $\mu(x) = 0$
- wende folgende Operation an, um y zu aktualisieren

Fibonacciheaps

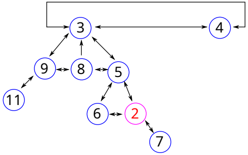


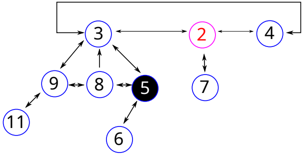


Aktualisieren von y

- setze z auf den Elternknoten von y
- falls y keinen Elternknoten hat, ist nichts zu tun
- sonst prüfe, ob $\mu(y) = 0$; dann setze $\mu(y) = 1$ und stoppe
- wenn $\mu(y) = 1$, wende Trennen auf y an (und iteriere)

Fibonacciheaps

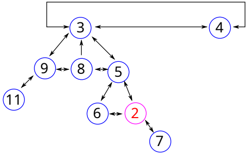


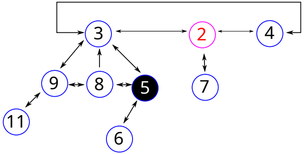


Amortisierte Laufzeit

- sei $\mathcal{F}'$ der Heap nach der Extraktion
- ein neuer Baum, der in die Wurzelliste eingefügt wird, stammt von einem Knoten u, dessen Markierung $\mu(u)$ die Trennoperation von 1 auf 0 setzt
- einzige Ausnahme ist ggf. der extrahierte Knoten

Fibonacciheaps





Amortisierte Laufzeit

- wenn t neue Bäume in die Wurzelliste eingefügt werden, ändert sich das Potential also um

$$\Phi(\mathcal{F}') - \Phi(\mathcal{F}) \leq t - 2t + 1 \leq 1 - t$$

- dem steht eine materielle Laufzeit von $O(t)$ gegenüber
- die amortisierte Laufzeit ist also $O(1)$

Fibonacciheaps

Entfernen eines Knotens

- verringere das Gewicht des zu entfernenden Knotens auf $-\infty$
- extrahiere anschließend den Knoten geringsten Gewichts
- amortisierte Laufzeit ist $O(D(N(\mathcal{F})))$

Fibonacciheaps

Proposition

Der maximale Grad eines Fibonacci-Heap $\mathcal{F}$ ist $O(\log N(\mathcal{F}))$.

Fibonacciheaps

Lemma

Angenommen Knoten x des Fibonacci-Heaps $\mathcal{F}$ hat k Kinder $y_1, \ldots, y_k$, numeriert nach der Reihenfolge, in der sie an x angefügt worden sind. Dann gilt hat y_i mindestens $i - 2$ Kinder ($i = 2, \ldots, k$).

Beweis

- zu dem Zeitpunkt, als y_i an x angefügt wurde, hatte x bereits mindestens $i - 1$ Kinder
- ein Knoten wird nur dann als Kind an einen anderen angehängt, wenn beide den gleichen Grad haben
- seitdem y_i an x angehängt wurde, hat y_i höchstens ein Kind verloren (weil im Fall $\mu(y) = 1$ der Knoten y selbst zur Wurzel wird, wenn er noch ein Kind verliert)

Fibonacciheaps

Erinnerung: die Fibonacci-Zahlen

- $F_0 = 0$
- $F_1 = 1$
- $F_k = F_{k-1} + F_{k-2}$ für $k \geq 2$
- also erhalten wir für $\ell \geq 0$,

$$F_{\ell+2} = 1 + \sum_{i=0}^{\ell} F_i \geq \left(\frac{1+\sqrt{5}}{2}\right)^{\ell}$$

Fibonacciheaps

Lemma

Für einen Knoten x in einem Fibonacci-Heap $\mathcal{F}$ sei $N(x, \mathcal{F})$ die Zahl der Nachkommen von x, einschließlich x selbst. Wenn ℓ die Zahl der Kinder von x ist, gilt $N(x, \mathcal{F}) \geq F_{\ell+2}$.

Beweis

- Induktion nach ℓ
- sei s_ℓ die kleinstmögliche Zahl von Nachkommen eines Knotens mit ℓ Kindern
- wir sehen unmittelbar, daß $s_0 = 1$ und $s_1 = 2$
- seien nun allgemeien $y_1, \ldots, y_k$ die Kinder von x in $\mathcal{F}$
- Knoten y_i habe k_i Kinder
- dann gilt $k_i \geq i - 2$

Fibonacciheaps

Beweis (fortgesetzt)

- ferner gilt $s_{h+1} \geq s_h$
- also erhalten wir mit Induktion und dem vorherigen Lemma

$$\begin{aligned} s_\ell &\geq 2 + \sum_{i=2}^{\ell} s_{k_i} \geq 2 + \sum_{i=2}^{\ell} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{\ell} F_i > F_{\ell+2} \end{aligned}$$

- weil $N(x, \mathcal{F}) \geq s_\ell$, folgt die Behauptung

Fibonacciheaps

Beweis der Proposition

- das Lemma zeigt, daß für jeden Knoten x mit ℓ Kindern gilt

$$N(x,\mathcal{F}) \geq F_{\ell+2} \geq \left(\frac{1+\sqrt{5}}{2}\right)^{\ell}$$

- also folgt $D(\mathcal{F}) \leq O(\log N(\mathcal{F}))$

Fibonacciheaps

Amortisierte Laufzeiten

Aus der Proposition und unserer obigen Analyse ergeben sich die folgenden amortisierten Laufzeiten für den Fibonacci-Heap:

einfügen: $O(1)$

Minimum: $O(1)$

Minimum entnehmen: $O(\log n)$

Vereinigung: $O(1)$

verringern: $O(1)$

löschen: $O(\log n)$

Fibonacciheaps

Zusammenfassung

- Fibonacciheaps sind die effizienteste Datenstruktur für den Dijkstra-Algorithmus
- die Laufzeit von Dijkstra mit Fibonacciheaps beträgt

$$O(|E| + |V| \log |V|)$$

- Die Laufzeit wir mit Hilfe der amortisierten Analyse abgeschätzt