
Algorithmen & Komplexität

Angelika Steger
Institut für Theoretische Informatik

steger@inf.ethz.ch

Kapitel 4:

Datenstrukturen

Kapitel 4.2:

Union-Find-Strukturen

Union-Find-Strukturen

Gegeben:

Datensätze partitioniert in (paarweise disjunkte) Mengen, wobei jede Menge durch einen in ihr enthaltenen Datensatz repräsentiert wird.

Gesucht:

Datenstruktur, die folgende Operationen unterstützt.

- MakeNewSet(x)** → Füge neuen Datensatz x ein; dieser bildet eine neue (einelementige) Menge.
- Find(x)** → Gebe Repräsentanten derjenigen Menge aus, die x enthält.
- Union(x,y,r)** → Vereinige die beiden Mengen zu denen x und y gehören; Repräsentant der neuen Menge sei r.

Anwendungsbeispiel: Algorithmus von Kruskal

Algorithmus 2.9 Algorithmus von Kruskal

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$

Ausgabe: Ein Spannbaum $T = (V, F)$ in N

$\forall v \in V:$

MakeNewSet(v)

{ Initialisierung }

Sortiere die Kanten, so dass $\ell(e_1) \leq \dots \leq \ell(e_m)$

Setze $F := \emptyset$.

{ Aufbau des Baums }

for $i := 1$ to m do

 if $(V, F \cup \{e_i\})$ ist kreisfrei then $F := F \cup \{e_i\}$.

Let $e_i = \{x, y\}$:

Union(x, y)

Let $e_i = \{x, y\}$:

if Find(x) \neq Find(y) then ...

Gegeben:

Datensätze partitioniert in (paarweise disjunkte) Mengen, wobei jede Menge durch einen in ihr enthaltenen Datensatz repräsentiert werde.

Idee:

Jede Menge wird durch einen zur Wurzel hin gerichteten Baum dargestellt (engl. *intree*). An jeder Wurzel ist gespeichert:

- Repräsentant der Menge
- Höhe des Baumes

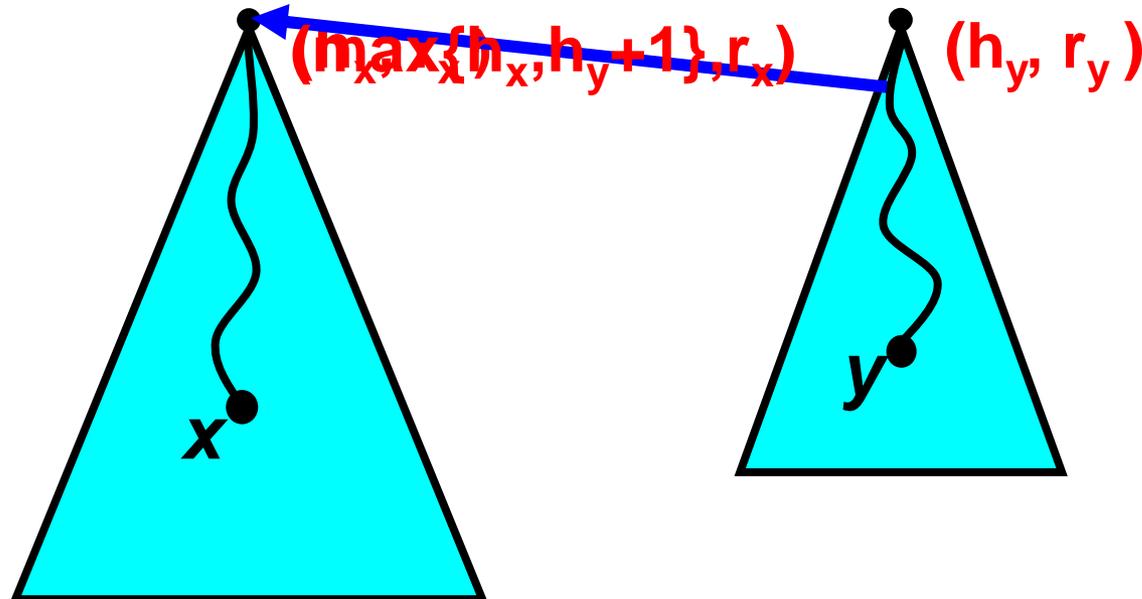
MakeNewSet(x) → x bildet einen Baum, der nur aus der Wurzel besteht;
Höhe := 0, Repräsentant := x

Find(x) → Laufe von x zur Wurzel; gebe den an der Wurzel gespeicherten Repräsentanten aus,

Union(x,y,r) → Vereinige die Bäume ...

Union(x,y)

Informationen an der
Wurzel müssen
angepasst werden.



$h_y \leq h_x$: Hänge Wurzel von y an die Wurzel von x an ...

Lemma:

Für jeden in der Union-Find-Struktur enthaltenen Baum T gilt

$$\text{size}(T) \geq 2^{\text{height}(T)}$$

wobei $\text{size}(T)$ die Anzahl der Knoten in T und $\text{height}(T)$ die Höhe von T sei.

Korollar:

Für eine Union-Find Struktur mit n Elementen haben $\text{Find}(x)$ und $\text{Union}(x,y,r)$ Laufzeit $O(\log(n))$.

Kapitel 4:

Datenstrukturen

Kapitel 4.4:

Vorrangwarteschlangen

Algorithmus von Dijkstra

Algorithmus 2.5 Algorithmus von Dijkstra

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein kürzester s - t -Pfad in N

{ *Initialisierung* }

$W := \emptyset$; $\rho[s] := 0$; $\text{pred}[s] := \text{nil}$;

for all $v \in V \setminus \{s\}$ **do** ~~$\rho[v] := \infty$~~ ; **Insert**(v, ∞)

{ *Traversieren und Markieren* }

while $t \notin W$ **do**

 { *Traversieren* }

~~Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$~~ ; **$x_0 = \text{ExtractMin}$**

$W := W \cup \{x_0\}$;

 { *Markieren* }

for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ **do**

~~$\rho[v] := \rho[x_0] + \ell(x_0, v)$~~ ; $\text{pred}[v] := x_0$; **DecreaseKey**($v, \rho[x_0] + \ell(x_0, v)$)

Laufzeit:

$O(n \cdot \text{Laufzeit von Insert} + n \cdot \text{Laufzeit von ExtractMin} +$

$m \cdot \text{Laufzeit von DecreaseKey})$

Algorithmus von ~~Dijkstra~~ **Prim**

Algorithmus 2.5 Algorithmus von ~~Dijkstra~~ **Prim**

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein ~~kürzester $s-t$ Pfad in N~~ **minimaler Spannbaum**

{ *Initialisierung* } **Wähle $s \in V$ beliebig**

$W := \emptyset$; $\rho[s] := 0$; $\text{pred}[s] := \text{nil}$;

for all $v \in V \setminus \{s\}$ **do** $\rho[v] := \infty$;

{ *Traversieren und Markieren* }

while ~~$t \notin W$~~ **do** **$W \neq V$**

 { *Traversieren* }

 Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$;

$W := W \cup \{x_0\}$;

 { *Markieren* }

for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ **do**

$\rho[v] := \rho[x_0] + \ell(x_0, v)$; $\text{pred}[v] := x_0$;



Laufzeit:

$O(n \cdot \text{Laufzeit von Insert} + n \cdot \text{Laufzeit von ExtractMin} + m \cdot \text{Laufzeit von DecreaseKey})$

Vorrangwarteschlangen - Realisierungen

	Feld oder Liste (sortiert)	Feld oder Liste (unsortiert)	Suchbaum
Insert Extract-Min Decrease-Key			

Fibonacci-Heaps:

Insert: $O(1)$ *
DecreaseKey: $O(1)$ *
ExtractMin: $O(\log n)$ *

*) amortisierte Kosten

Fibonacci-Heaps: Eigenschaften

Satz:

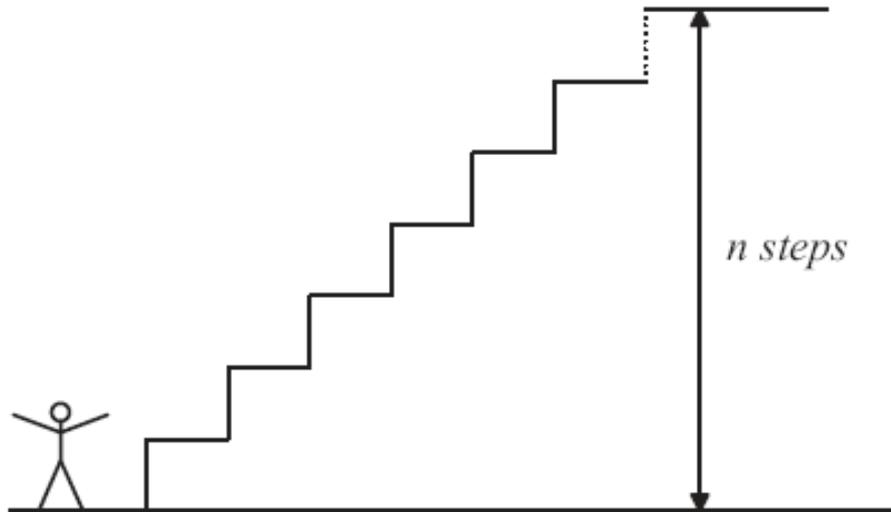
Beginnend mit einem leeren Heap können k Operationen **Insert**, **Decrease-Key** und **Extract-Min** in Zeit $\mathcal{O}(k + \ell \cdot \log n)$ ausgeführt werden, wobei ℓ die Anzahl von **Extract-Min** Operationen ist und n die maximale Anzahl von Elementen bezeichnet, die zu irgend einem Zeitpunkt im Heap enthalten sind.

Für Algorithmus von Dijkstra/Prim heisst das:

n **Insert**,
 n **Extract-Min** und
 m **Decrease-Key**

können in Zeit $\mathcal{O}(n + m \log n)$ ausgeführt werden

Amortisierte Analyse - Beispiel



Annahmen:

- pro Stufe eine Sekunde, unabh. von Richtung
- wir beginnen unten

Zwei Arten von Operationen:

Rauf: Steige eine Stufe hinauf, falls noch nicht ganz oben.

Runter: Gehe ganz nach unten

Frage:

Wie lange dauert eine Operation höchstens?

Wie lange dauert eine Folge von k Operationen höchstens?

Fibonacci-Heaps: Eigenschaften

Satz:

Beginnend mit einem leeren Heap können k Operationen **Insert**, **Decrease-Key** und **Extract-Min** in Zeit $\mathcal{O}(k + \ell \cdot \log n)$ ausgeführt werden, wobei ℓ die Anzahl von **Extract-Min** Operationen ist und n die maximale Anzahl von Elementen bezeichnet, die zu irgend einem Zeitpunkt im Heap enthalten sind.

Für Algorithmus von Dijkstra/Prim heisst das:

n **Insert**,
 n **Extract-Min** und
 m **Decrease-Key**

können in Zeit $\mathcal{O}(m + n \log n)$ ausgeführt werden

Fibonacci-Heap (F-Heap):

- Menge von gewurzelten Bäumen.
- Jeder Baum hat die Eigenschaft, dass jeder Knoten eine Menge von Nachfolgern oder *Kindern* hat, so dass der Schlüssel jedes Knotens kleiner oder gleich dem Schlüssel jedes seiner Kinder ist.
- Die Wurzeln dieser Bäume sind in einer (doppelt verketteten) zyklischen Liste gespeichert, der *Wurzelliste*.
- Zu jedem Zeitpunkt zeigt der Zeiger $\text{min}[H]$ eines F-Heaps auf den Knoten mit dem kleinsten Schlüssel in der Wurzelliste (und damit im gesamten Heap).

Fibonacci-Heap

