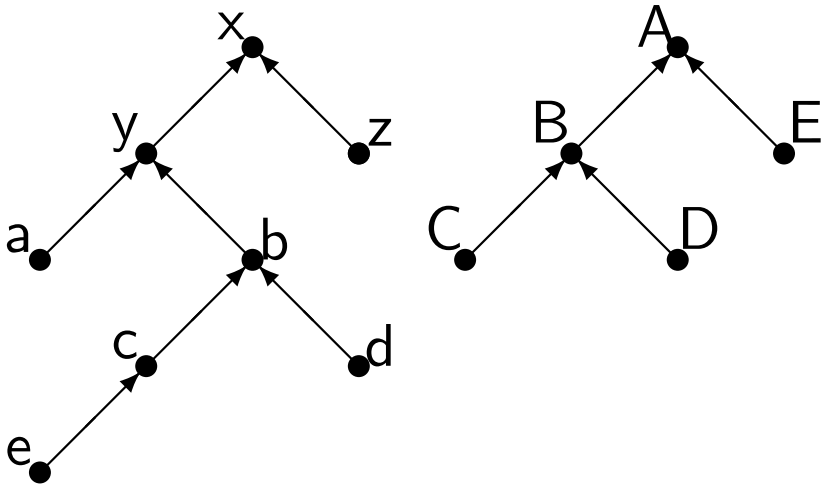
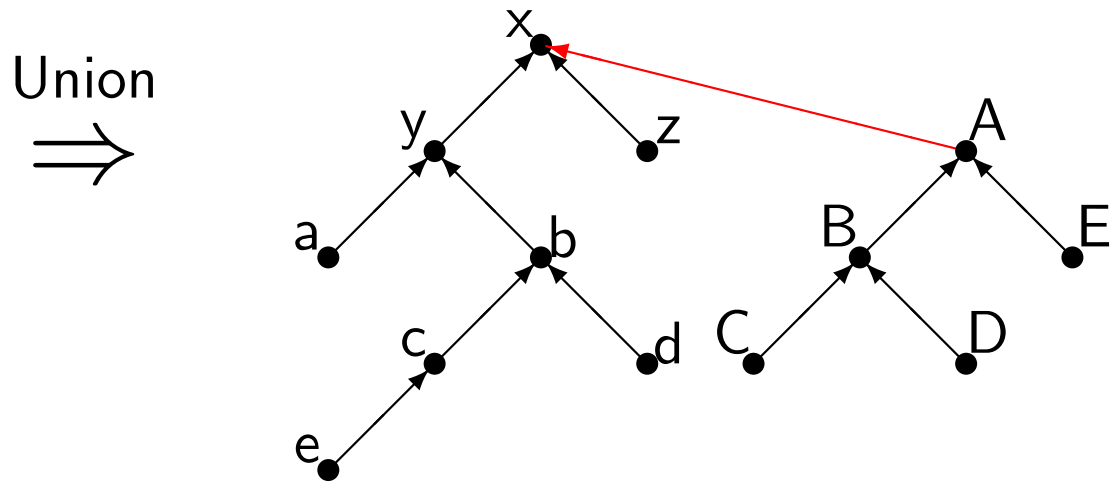
8.2.3 Pfad-Kompression mit gewichteter Union (zweite Verbesserung)

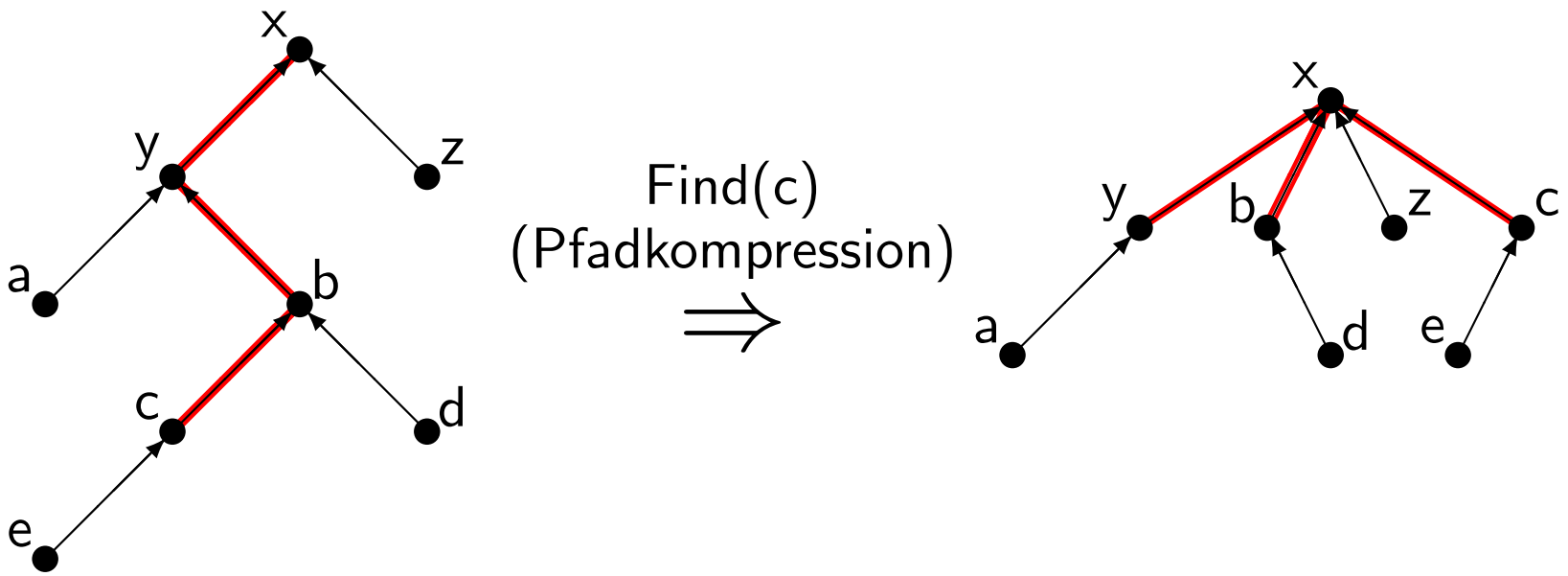
Wir betrachten eine Folge von k *Find*- und *Union*-Operationen auf einer Menge mit n Elementen, darunter $n-1$ *Union*.

Implementierung: Gewichtete *Union* für Pfad-Kompression:





Implementierung: *Find* für Pfad-Kompression:



Bemerkung:
Nach Definition ist

$$\log^* n = \min\{i \geq 0; \underbrace{\log\log\log\ldots\log n}_{i \text{ log 's}} \leq 1\}$$

Beispiel 72

$$\log^* 0 = \log^* 1 = 0$$

$$\log^* 2 = 1$$

$$\log^* 3 = 2$$

$$\log^* 16 = 3 \qquad \text{da } 16 = 2^{2^2}$$

$$\log^* 2^{65536} = 5 \qquad \text{da } 65536 = 2^{2^{2^{2^2}}}$$

Satz 73

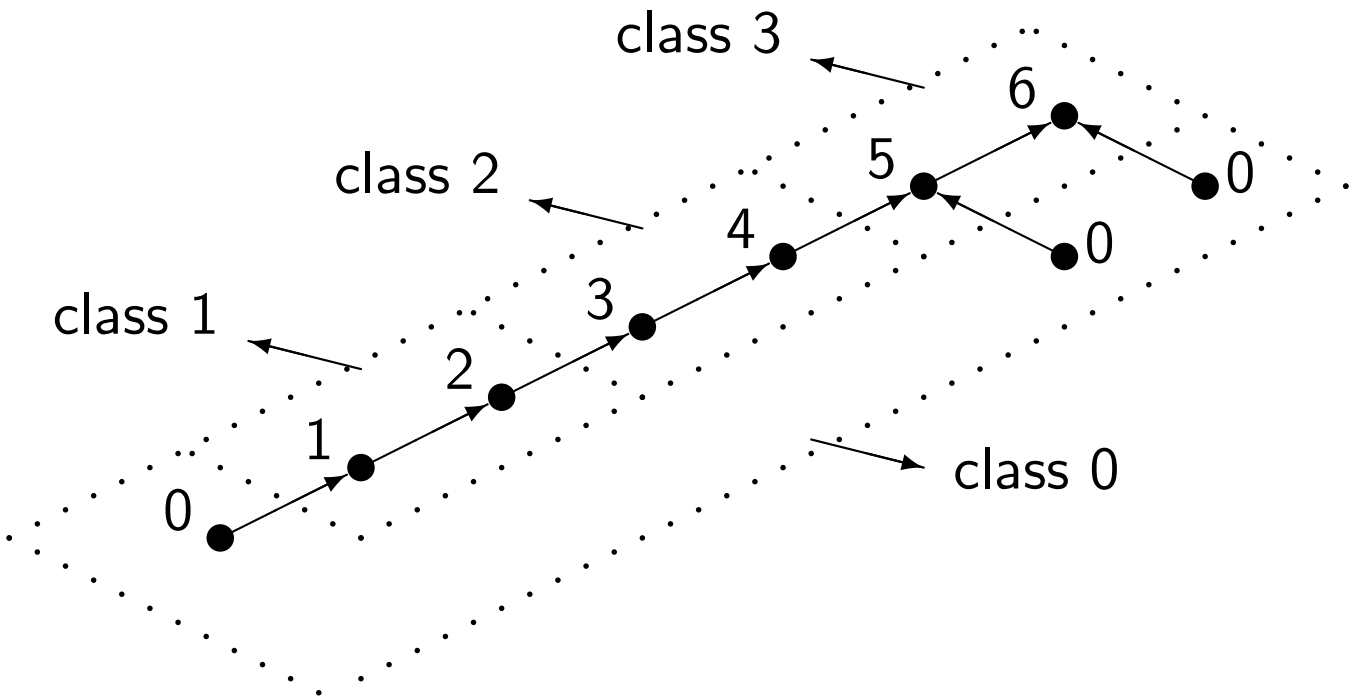
Bei der obigen Implementierung ergibt sich eine amortisierte Komplexität von $\mathcal{O}(\log^* n)$ pro Operation.

Beweis:

Sei T' der (endgültige) In-Baum, der durch die Folge der *Union*'s, ohne die *Find*'s, entstehen würde (also keine Pfad-Kompression).
Ordne jedem Element x drei Werte zu:

- rank(x):= Höhe des Unterbaums in T' mit Wurzel x
- class(x):= $\begin{cases} i \geq 1 & \text{falls } a_{i-1} < \text{ rank}(x) \leq a_i \text{ ist } (i \geq 1) \\ 0 & \text{falls rank}(x) = 0 \end{cases}$

  Dabei gilt: $a_0 = 0, a_i = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{i \text{ 2'en}}$ für $i \geq 1$.
  Setze zusätzlich $a_{-1} := -1$.


class 3
class 2
class 1
class 0
0
1
2
3
4
5
6
0
0

Beweis (Forts.):

- dist(x) ist die Distanz von x zu einem Vorfahr y im momentanen Union/Find-Baum (mit Pfad-Kompression), so dass class(y) $>$ class(x) bzw. y die Wurzel des Baumes ist.

Definiere die Potenzialfunktion

$$\text{Potenzial} := c \sum_x \text{dist}(x), \quad c \text{ eine geeignete Konstante } > 0$$

Beweis (Forts.):

Beobachtungen:

i) Sei T ein Baum in der aktuellen Union/Find-Stuktur (mit Pfad-Kompression), seien x, y Knoten in T, y Vater von x. Dann ist class$(x) \leq$ class(y).

ii) Aufeinander folgende *Find*(x) durchlaufen (bis auf eine) verschiedene Kanten. Diese Kanten sind (im wesentlichen) eine Teilfolge der Kanten in T' auf dem Pfad von x zur Wurzel.

Beweis (Forts.):

Amortisierte Kosten Find(x):

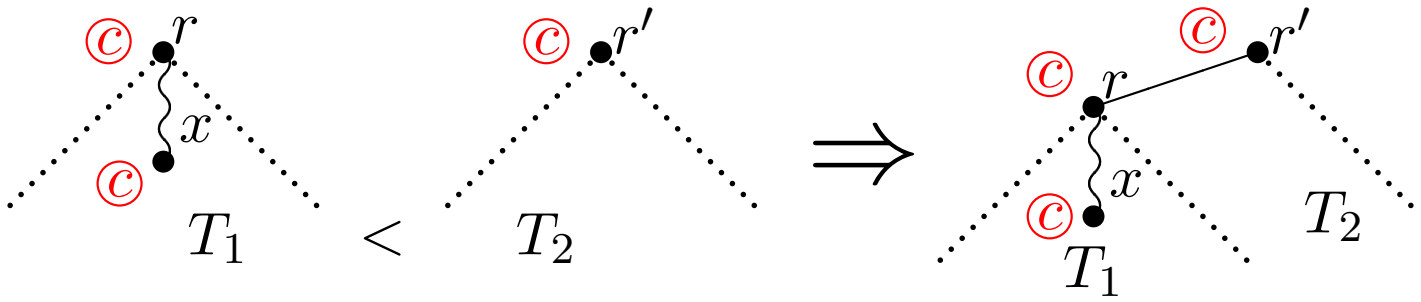
Sei $x_0 \to x_1 \to x_2 \ldots x_k = r$ der Pfad von x_0 zur Wurzel. Es gibt höchstens $\log^* n$-Kanten (x_{i-1}, x_i) mit class$(x_{i-1}) <$ class(x_i). Ist class$(x_{i-1}) =$ class(x_i) und $i < k$ (also $x_i \neq r$), dann ist dist(x_{i-1}) vor der *Find(x)*-Operation ≥ 2, nachher gleich 1.

Damit können die Kosten für alle Kanten (x_{i-1}, x_i) mit class(x_{i-1}) $=$ class(x_i) aus der Potenzialverringerung bezahlt werden. Es ergeben sich damit amortisierte Kosten

$$\boxed{\mathcal{O}(\log^* n)}$$

Beweis (Forts.):

Amortisierte Gesamtkosten aller $(n-1)$-Union's:



Die gesamte Potenzialerhöhung durch alle *Union*'s ist nach oben durch das Potenzial von T' beschränkt (Beobachtung ii).

Beweis (Forts.):

$$\begin{aligned}
\text{Potenzial}(T') &\leq \sum_{i=0}^{\log^* n} \sum_{\text{rank}(x)=j=a_{i-1}+1}^{a_i} \text{dist}(x) \\
&\leq \sum_{i=0}^{\log^* n} \sum_{\text{rank}(x)=j=a_{i-1}+1}^{a_i} \frac{n}{2^j} a_i \\
&\leq n \sum_{i=0}^{\log^* n} a_i \frac{1}{2^{a_{i-1}}} = n \sum_{i=0}^{\log^* n} 1 \\
&= \mathcal{O}(n \log^* n) \,.
\end{aligned}$$

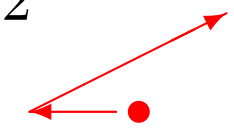
Die zweite Ungleichung ergibt sich, da alle Unterbäume, deren Wurzel x $\text{rank}(x) = j$ hat, disjunkt sind und jeweils $\geq 2^j$ Knoten enthalten. □

8.2.4 Erweiterungen

1) Bessere obere Schranke $\alpha(n,n), k \geq n$. Betrachte die (Variante der) Ackermannfunktion $A(m,n)$ mit:

$$A(0,n) = 2n; \qquad n \geq 0$$
$$A(m,0) = 2; \qquad m \geq 1$$
$$A(m+1,n+1) = A(m, A(m+1,n))$$

$m \downarrow$ / $n \rightarrow$	0	2	4	6	8
	2	4	8	16	32
	2	8	2^9		
	2				
	2				
	$\vdots$				



Die Ackermannfunktion $A(\cdot,\cdot)$ steigt asymptotisch schneller als jede primitiv-rekursive Funktion.

Definition 74

Die Klasse der primitiv-rekursiven Funktionen (auf den natürlichen Zahlen) ist induktiv wie folgt definiert:

1. Alle konstanten Funktionen sind primitiv-rekursiv.
2. Alle Projektionen sind primitiv-rekursiv.
3. Die Nachfolgerfunktion auf den natürlichen Zahlen ist primitiv-rekursiv.
4. Jede Funktion, die durch Komposition von primitiv-rekursiven Funktionen entsteht, ist primitiv-rekursiv.
5. Jede Funktion, die durch sog. primitive Rekursion aus primitiv-rekursiven Funktionen entsteht, ist primitiv-rekursiv. Primitive Rekursion bedeutet folgendes Schema für die Definition von f:

$$f(0, \cdots) = g(\cdots)$$
$$f(n+1, \cdots) = h(f(n, \cdots), \cdots)$$

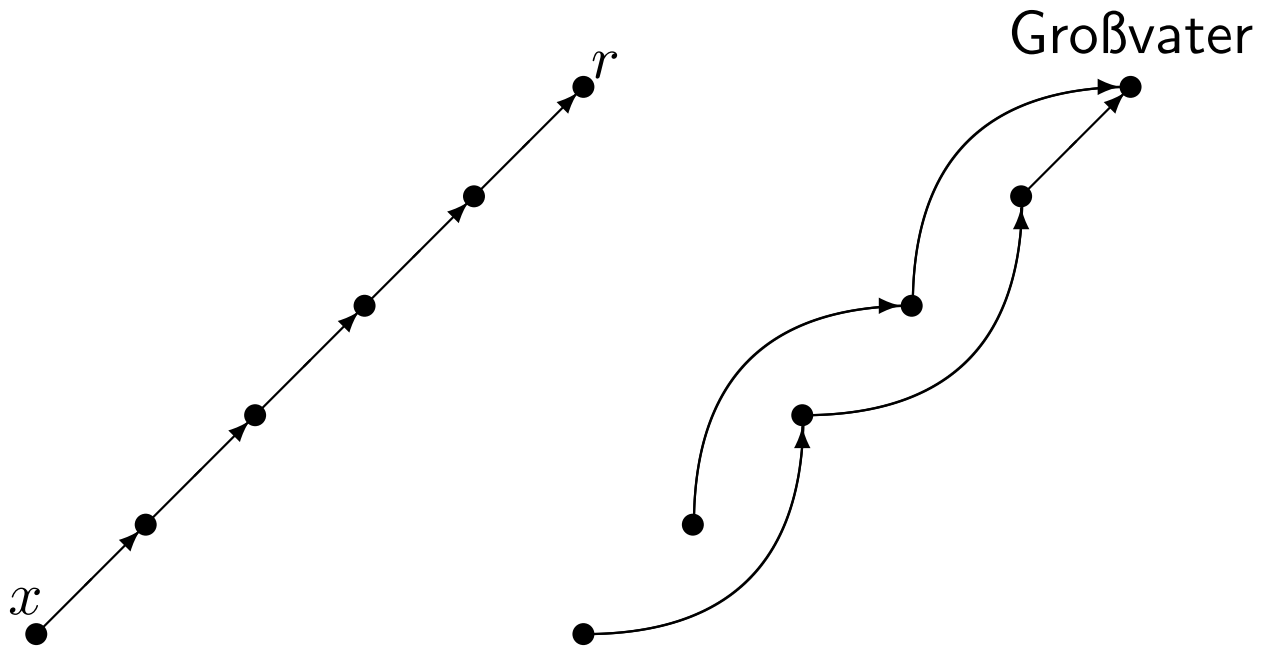
wobei g, h bereits primitiv-rekursive Funktionen sind.

Weiter wird gesetzt:

$$\alpha(n, n) = \min\{i; A(i, i) \geq n\}$$

Es gilt auch eine entsprechende untere Schranke.

2) Variante der Pfadkompression:



Kapitel III Selektieren und Sortieren

1. Einleitung

Gegeben: Menge S von n Elementen aus einem total geordneten Universum U, $i \in \mathbb{N}$, $1 \leq i \leq n$.
Gesucht: i-kleinstes Element in S.

Die Fälle $i = 1$ bzw. $i = n$ entsprechen der Suche nach dem Minimum bzw. Maximum.
Der Standardalgorithmus dafür benötigt $n - 1$ Vergleiche.

Satz 75

Die Bestimmung des Minimums/Maximums von n Elementen benötigt mindestens $n-1$ Vergleiche.

Beweis:

Interpretiere Algorithmus als Turnier. Ein Spiel wird jeweils vom kleineren Element gewonnen. Wir beobachten: Jedes Element außer dem Gesamtsieger muss mindestens ein Spiel verloren haben $\Rightarrow n-1$ Vergleiche notwendig. □

Bestimmung des Vize-Meisters bzw. des zweitkleinsten Elements

Satz 76

Das zweitkleinste von n Elementen kann mit

$$n + \lceil \log_2 n \rceil - 2$$

Vergleichen bestimmt werden.

Beweis:

Wir betrachten wiederum ein KO-Turnier: $(n-1)$ Vergleiche genügen zur Bestimmung des Siegers (Minimum).
Das zweitkleinste Element ist unter den „Verlierern“ gegen das Minimum zu suchen. Deren Anzahl ist $\leq \lceil \log_2 n \rceil$. Man bestimme nun unter diesen Elementen wiederum das Minimum und erhält damit das zweitkleinste Element in $\leq \lceil \log_2 n \rceil - 1$ weiteren Vergleichen. □


Lewis Carroll:
Lawn Tennis Tournaments
St. Jones Gazette (Aug. 1, 1883), pp. 5–6
Reprinted in The Complete Work of Lewis Carroll. Modern Library, New York (1947)


Vaughan R. Pratt, Frances F. Yao:
On lower bounds for computing the i-th largest element
Proc. 14th Ann. IEEE SWAT, pp. 70–81 (1973)


Donald E. Knuth:
The art of computer programming. Vol. 3: Sorting and searching,
3. Auflage, Addison-Wesley Publishing Company: Reading (MA), 1997