Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Stefan Werner (Übungen)

sowie viele Tutoren

Danksagung

Die nachfolgenden Präsentationen wurden ausdrücklicher Erlaubnis des Autors mit einigen Änderungen übernommen aus:

- „Effiziente Algorithmen und Datenstrukturen“ (Kapitel 6: Verschiedenes) gehalten von Christian Scheideler an der TUM
  http://www14.in.tum.de/lehre/2008WS/ea/index.html.de

Union-Find Datenstruktur

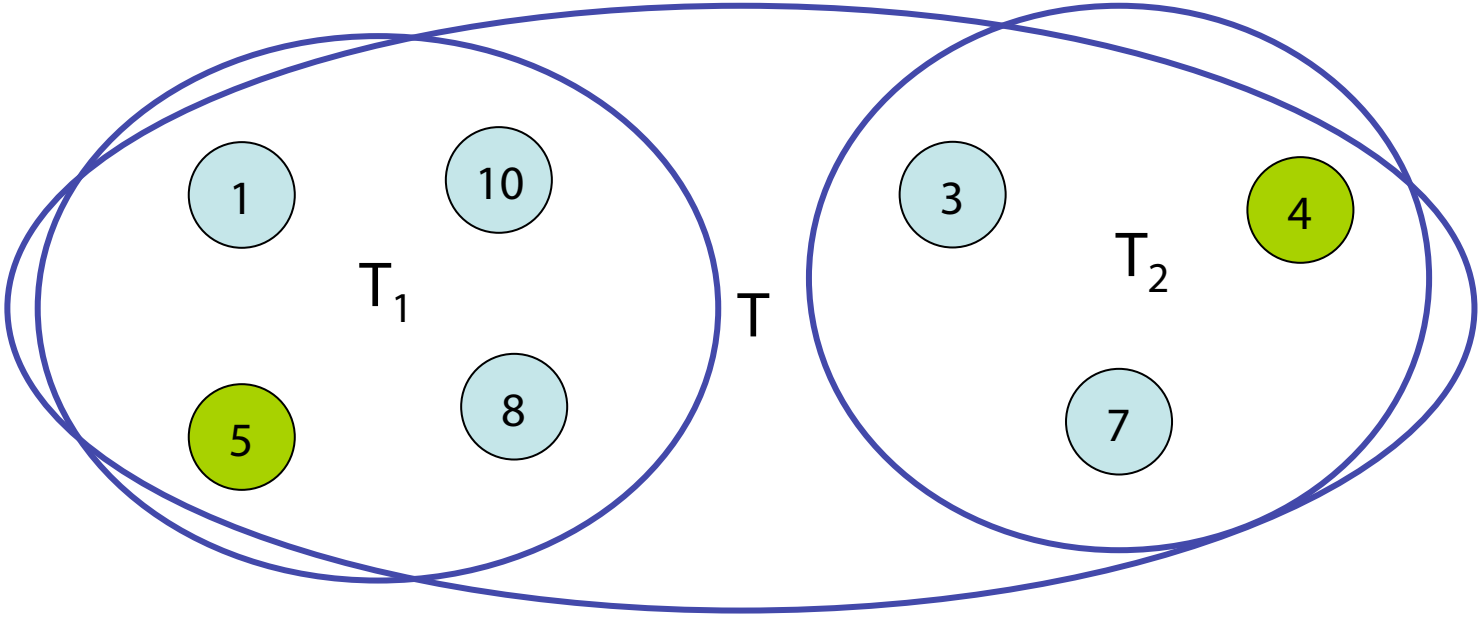
Gegeben: Menge von n Teilmengen $T_1,\ldots,T_n$ die jeweils ein Element enthalten.

Operationen:

- Union(T_1,T_2): vereinigt Elemente in T_1 und T_2 zu $T=T_1 \cup T_2$
- Find(x): gibt (eindeutigen) Repräsentanten der Teilmenge aus, zu der x gehört

Union-Find Datenstruktur

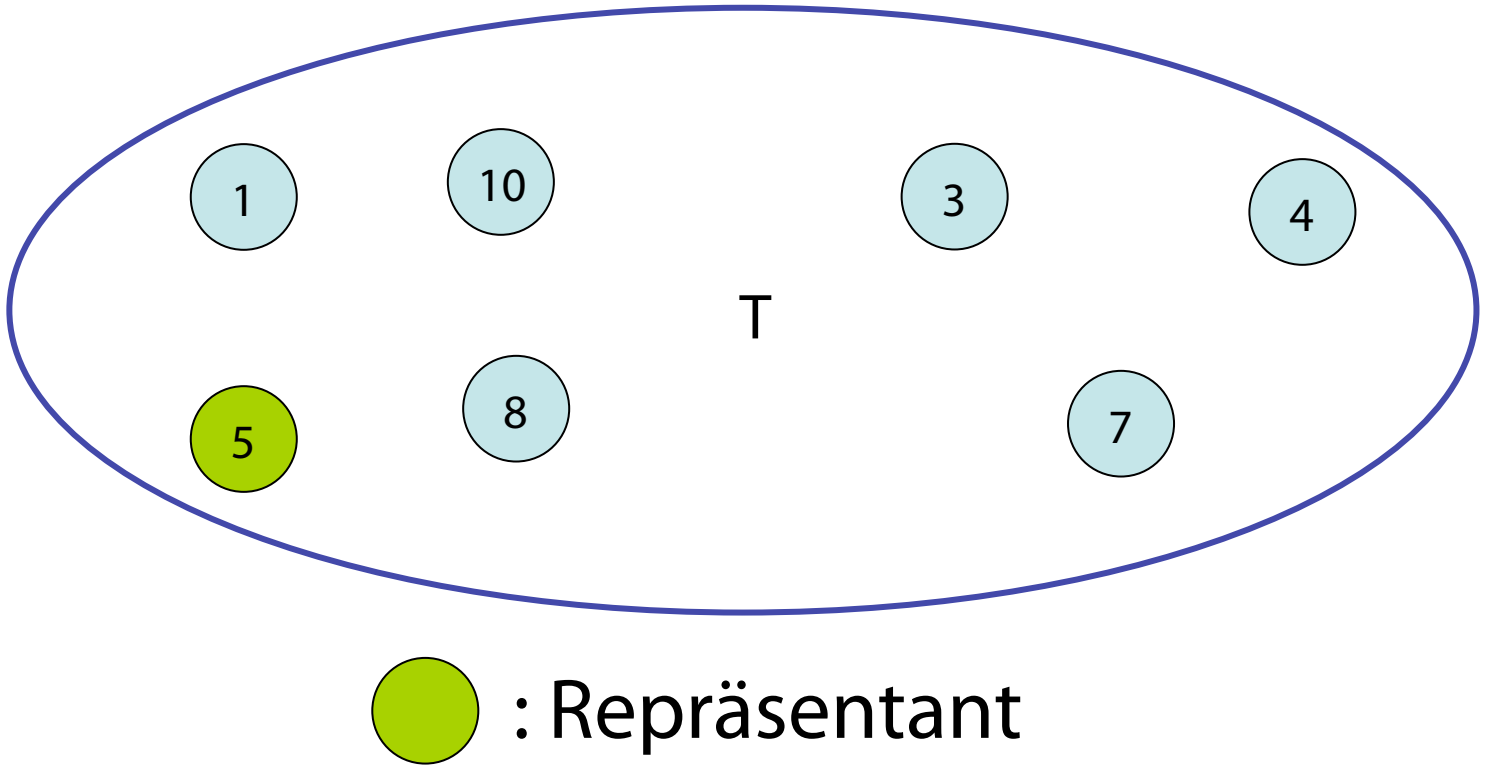
Union(T_1,T_2):



: Repräsentant

Union-Find Datenstruktur

Find(10) liefert 5



Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

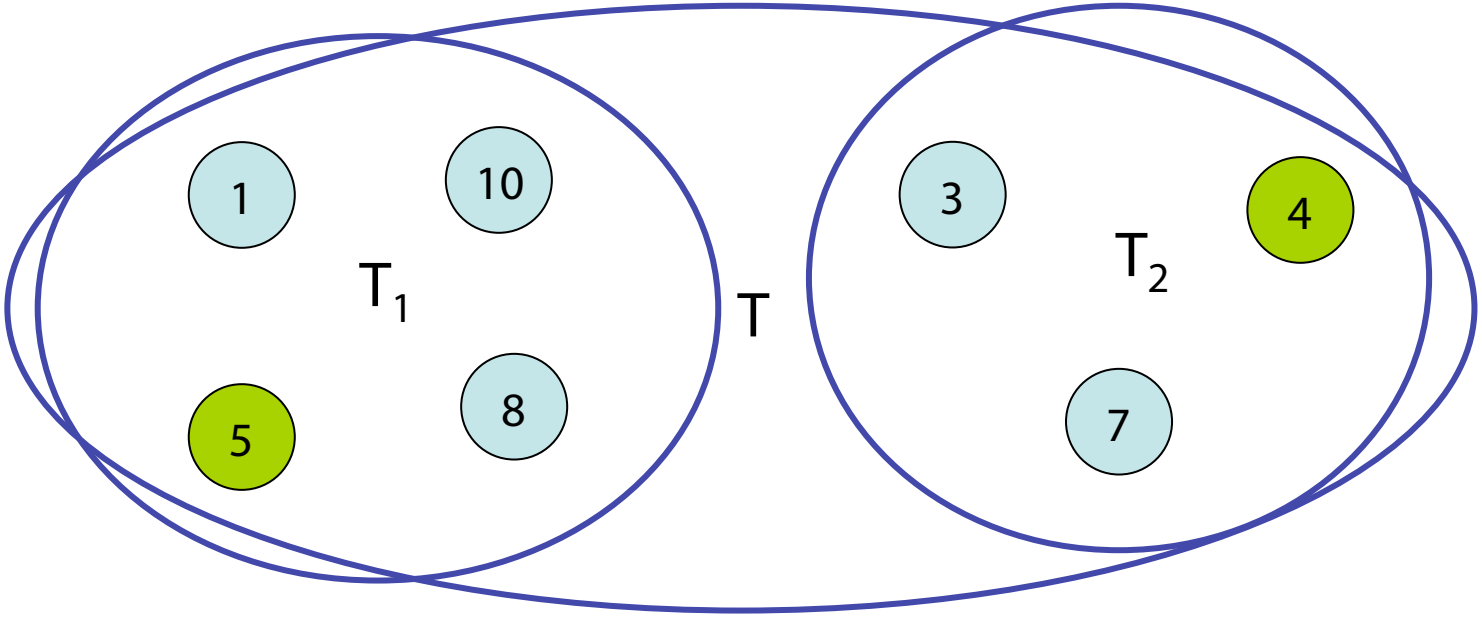
Institut für Informationssysteme

Stefan Werner (Übungen)

sowie viele Tutoren

Union-Find Datenstruktur

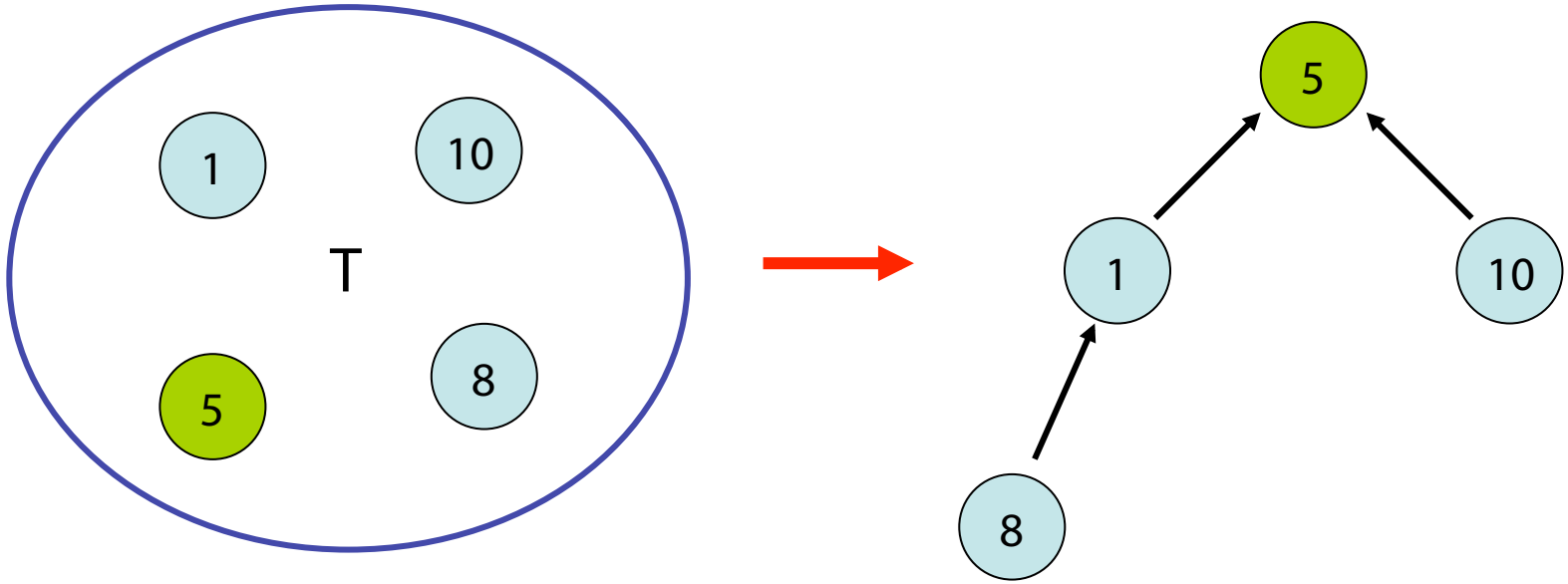
Union(T_1,T_2):



: Repräsentant

Union-Find Datenstruktur: Repräsentant

Idee: repräsentiere jede Menge T als gerichteten Baum mit Wurzel als Repräsentant

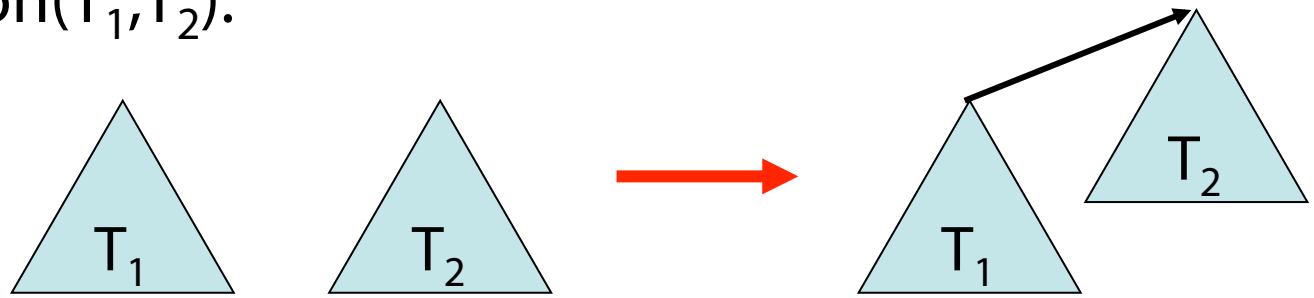


Galler, Bernard A.; Fischer, Michael J., An improved equivalence algorithm, Communications of the ACM 7 (5): 301–303, **1964**

Union-Find Datenstruktur

Realisierung der Operationen:

- Union(T_1,T_2):



- Find(x): Suche Wurzel des Baumes, in dem sich x befindet

Union-Find Datenstruktur

Naïve Implementierung:

- Tiefe des Baums kann bis zu n (bei n Elementen) sein
- Zeit für Find: $\Theta(n)$ im worst case
- Zeit für Union: $O(1)$

Union-Find Datenstruktur

Gewichtete Union-Operation: Mache die Wurzel des flacheren Baums zum Kind der Wurzel des tieferen Baums.

Beh.: Die Tiefe eines Baums ist höchstens $O(\log n)$

Beweis:

- Die Tiefe von $T=T_1 \cup T_2$ erhöht sich nur dann, wenn $Tiefe(T_1)=Tiefe(T_2)$ ist
- $N(t)$: min. Anzahl Elemente in Baum der Tiefe t
- Es gilt $N(t)=2 \cdot N(t-1) = 2^t$ mit $N(0)=1$
- Also ist $N(\log n) = 2^{\log n} = n$

Union-Find Datenstruktur

Mit gewichteter Union-Operation:

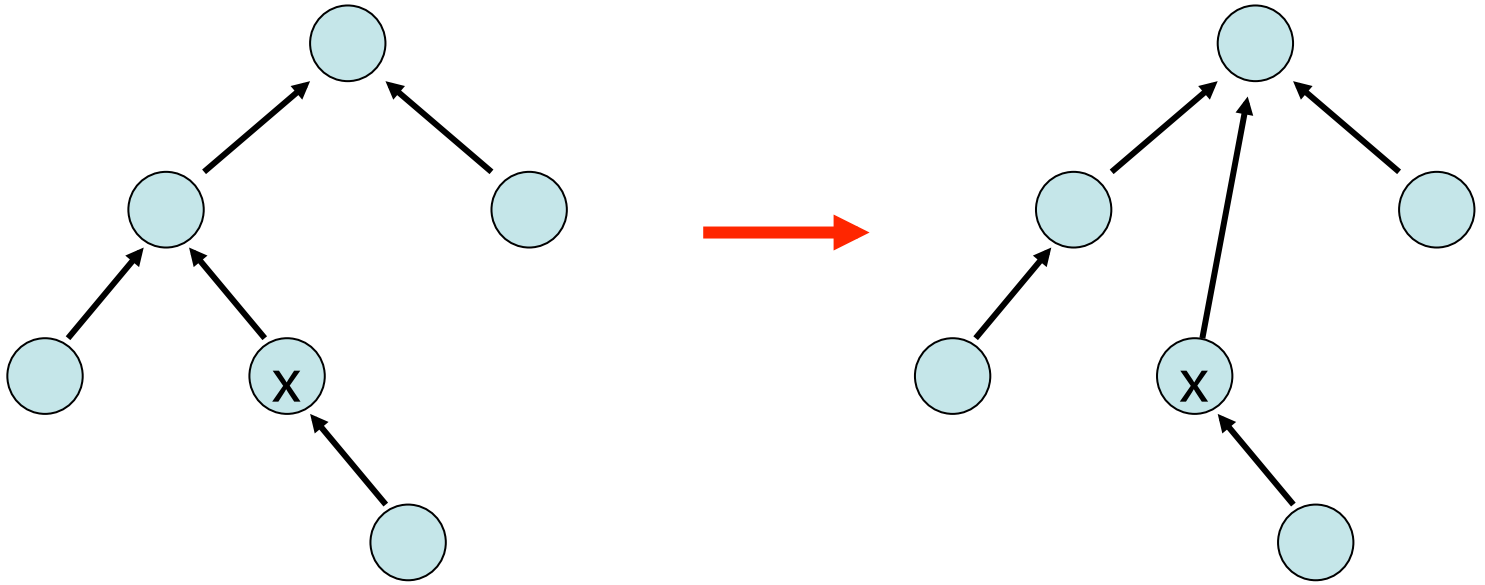
- Zeit für Find: $O(\log n)$
- Zeit für Union: $O(1)$

Es gilt auch: Tiefe eines Baums im worst-case $\Omega(\log n)$ (verwende Strategie, die Formel im vorigen Beweis folgt)

Union-Find Datenstruktur

Besser: gewichtetes Union mit Pfadkompression

- Pfadkompression bei **jedem Find**(x): alle Knoten von x zur Wurzel zeigen direkt auf Wurzel



Union-Find Datenstruktur

Bemerkung: $\log^* n$ ist definiert als

$$\log^* 0 = \log^* 1 = 0 \text{ für } n \leq 1$$

$$\log^* n = \min\{ i > 0 \mid \underbrace{\log \log \ \ldots \log}_{\text{i-mal}} n \leq 1\} \text{ sonst}$$

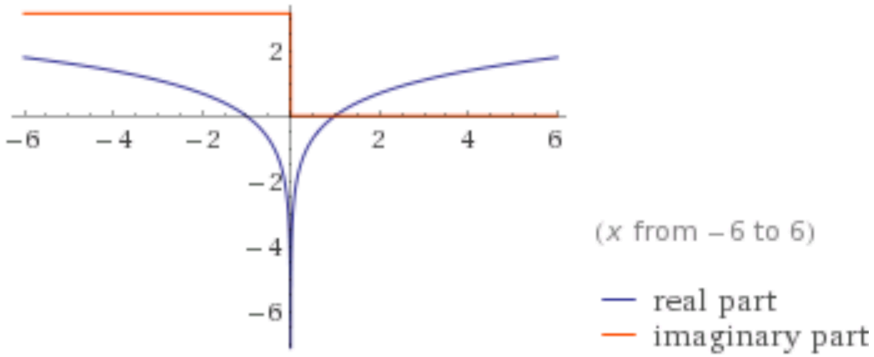
Iterierter Logarithmus

Beipiele:

- $\log^* 2 = 1$
- $\log^* 4 = 2$
- $\log^* 16 = 3$
- $\log^* 2^{65536} = 5$

Kurvenverläufe

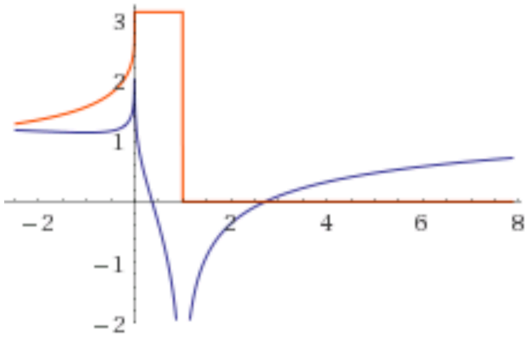
- log x



(*x* from −6 to 6)

real part
imaginary part

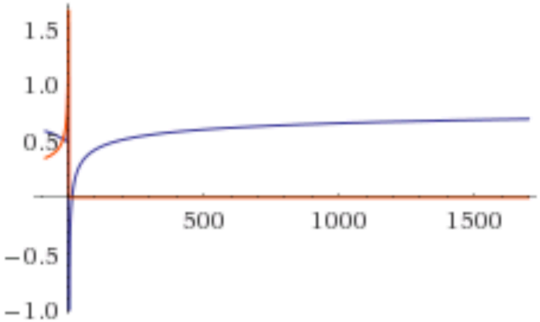
- log log x



(*x* from −2.5 to 7.9)

real part
imaginary part

- log log log x



(*x* from −83 to 1699)

real part
imaginary part

Union-Find Datenstruktur: Amortisierte Analyse

Theorem: Bei gewichtetem Union mit Pfadkompression ist die amortisierte Zeit für **Find** $O(\log^* n)$.

Beweis (Teil 1):

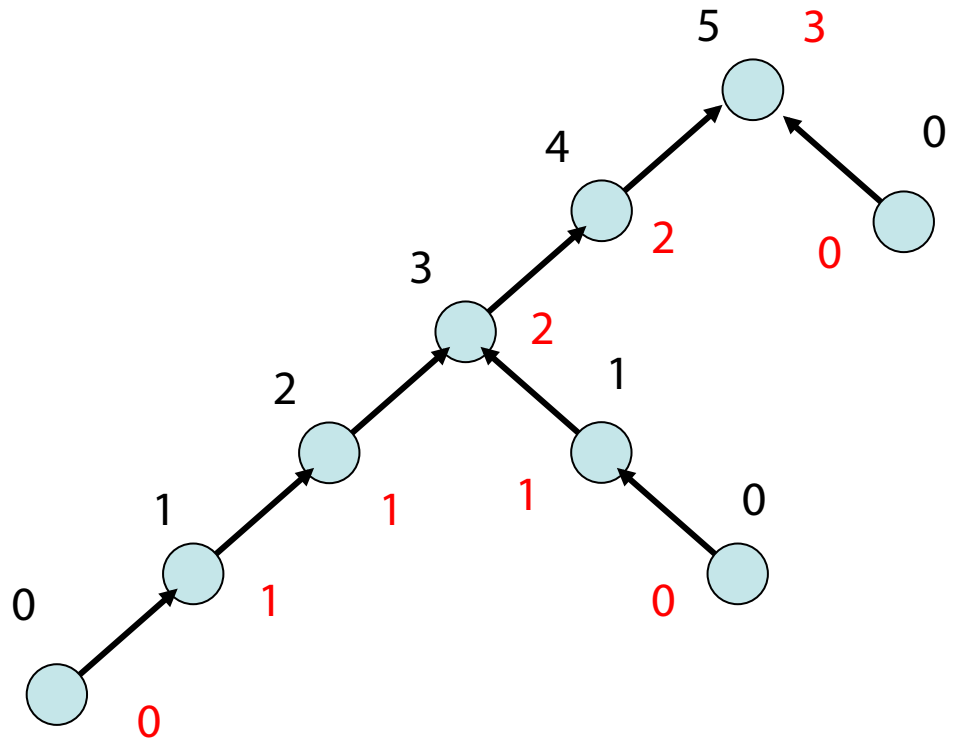
- T': endgültiger Baum, der durch die Folge der Unions ohne die Finds entstehen würde (also ohne Pfadkompression).

Ordne jedem Element x zwei Werte zu:

- rank(x) = Höhe des Unterbaums mit Wurzel x
- class(x) = i für das i mit $a_{i-1} < rank(x) \leq a_i$ wobei $a_{-1}=-1$, $a_0=0$ und $a_i = 2^{a_{i-1}}$ für alle $i>0$

Union-Find Datenstruktur

Beispiel 1:

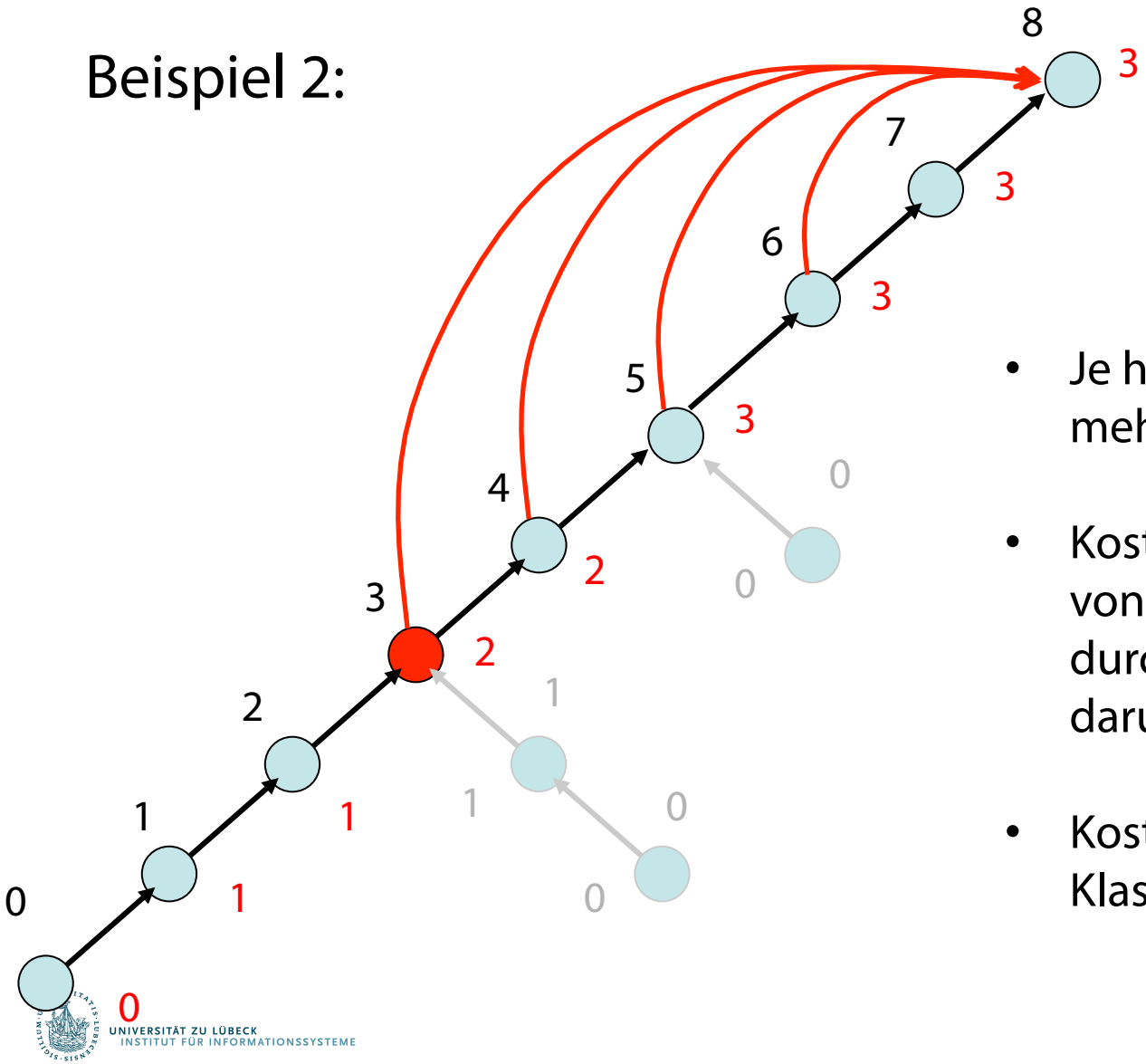


x: rank

x: class

Union-Find Datenstruktur

Beispiel 2:



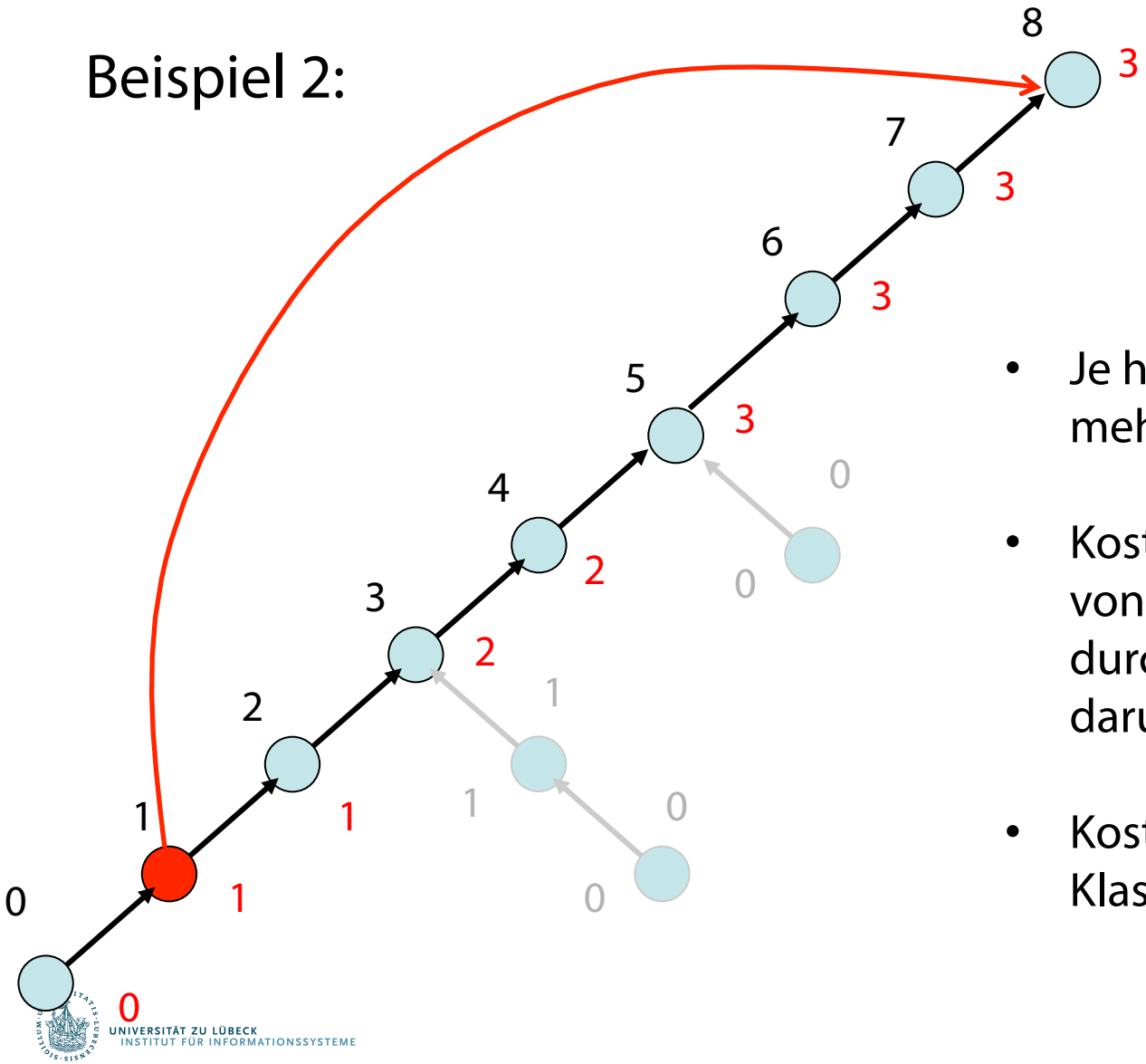
x: rank

x: class

- Je höher class(x), desto mehr Knoten darunter
- Kosten für "Überschreitung" von Kanten gleicher Klasse durch Gewinn für alle Knoten darunter "wettgemacht"
- Kosten treten bei Klassenwechsel auf, hier: **1**

Union-Find Datenstruktur

Beispiel 2:



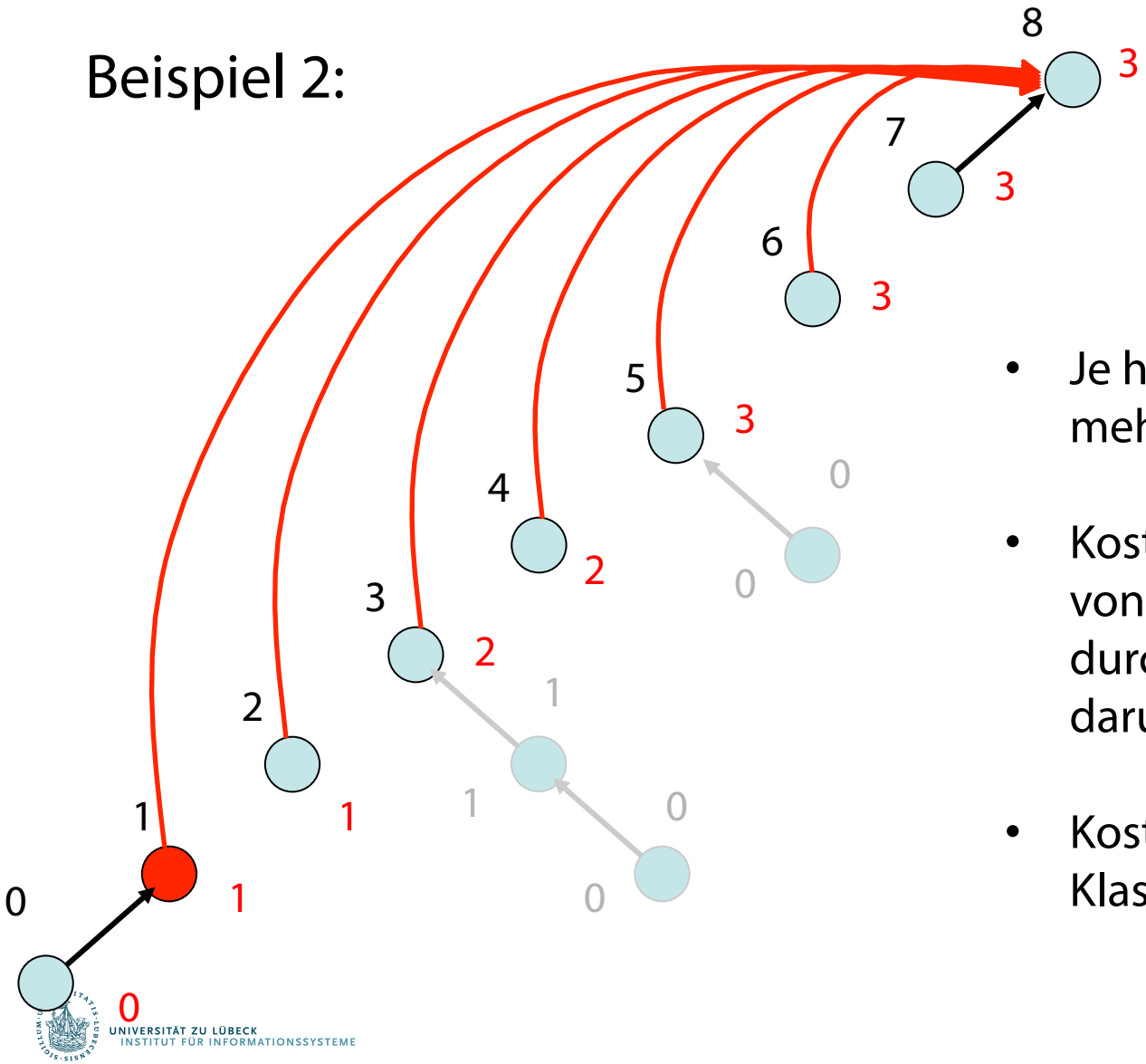
x: rank

x: class

- Je höher class(x), desto mehr Knoten darunter
- Kosten für "Überschreitung" von Kanten gleicher Klasse durch Gewinn für alle Knoten darunter "wettgemacht"
- Kosten treten bei Klassenwechsel auf: **2**

Union-Find Datenstruktur

Beispiel 2:



x: rank

x: class

- Je höher class(x), desto mehr Knoten darunter
- Kosten für "Überschreitung" von Kanten gleicher Klasse durch Gewinn für alle Knoten darunter "wettgemacht"
- Kosten treten bei Klassenwechsel auf: **2**

Union-Find Datenstruktur

Beweis (Fortsetzung):

- dist(x): Min. Distanz von x zu einem Vorfahr y im tatsächlichen Union-Find-Baum T (mit Pfadkompression), so dass class(y)>class(x) ist, bzw. zur Wurzel y

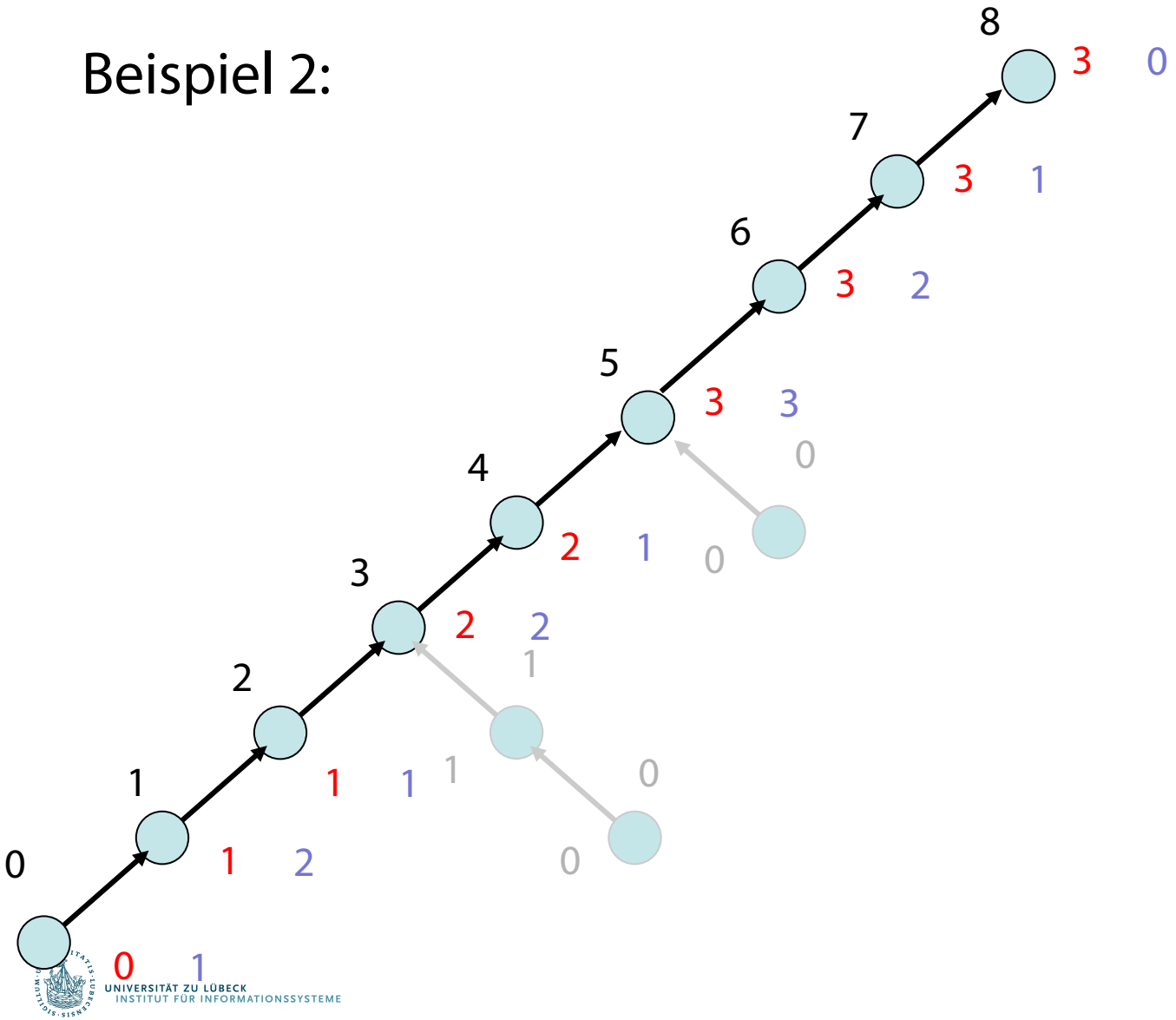
Potenzialfunktion:

$$\Phi(T) := c \sum_{x \in T} \text{dist}(x)$$

für eine geeignete Konstante c>0.

Union-Find Datenstruktur

Beispiel 2:



Union-Find Datenstruktur

Beobachtungen:

- Für den tatsächlichen Union-Find-Baum T seien x und y Knoten in T, y Vater von x. Dann ist class(x) ≤ class(y).
- Aufeinanderfolgende (rekursive) Find-Operationen durchlaufen (bis auf die letzte) verschiedene Kanten. Diese Kanten sind eine Teilfolge der Kanten in T' auf dem Pfad von x zur Wurzel.

Union-Find Datenstruktur

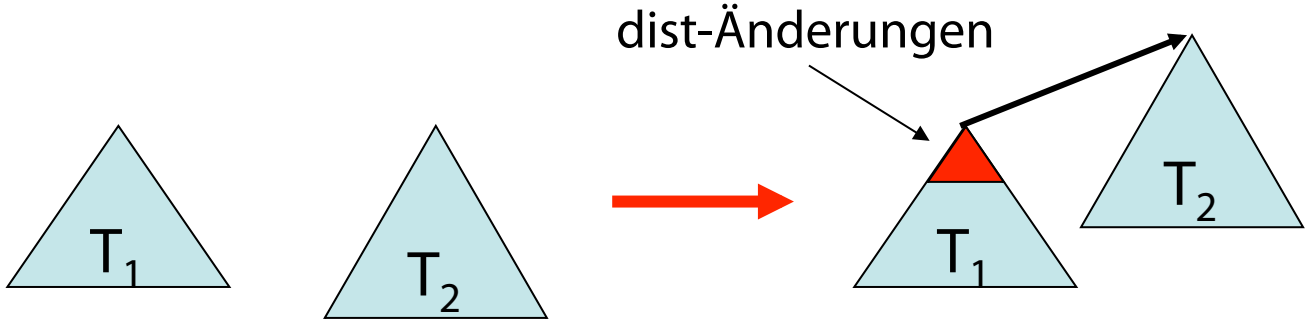
Amortisierte Kosten von **Find**:

- $x_0 \longrightarrow x_1 \longrightarrow x_2 \ldots x_k$: Pfad von x_0 zur Wurzel in T'
- 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 $i<k$, dann ist $dist(x_{i-1})$ vor der Find-Operation ≥ 2 und nachher $=1$.
- 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
- Amortisierte Kosten sind also $O(\log^* n)$
- NB: $\log^* n$ ist nicht asymptotisch eng (ist nur eine "lose" Abschätzung)

Union-Find Datenstruktur

Amortisierte Kosten von **Union**:

- Beobachtung: dist-Änderungen über alle Unions bzgl. T'



- Ohne weitere Analyse: Wenn Union-Operationen in der Folge der amortisierten Analyse sind, ändert sich an O(log* n) nichts

Zusammenfassung

- Find: O(log* n), Union: O(1)
- Können wir Find auf O(1) bringen?
  - Möglicherweise: Alle Knoten direkt mit dem Repräsentanten verbinden (und verbunden lassen)
  - Aber: Dann geht Union nicht mehr in O(1)
  - Die Find-Abschätzung kann tatsächlich noch deutlich verbessert werden[1]: $O(\alpha(n))$ amort.,
    wobei α die Umkehrfunktion der Ackermannfunktion ist, also SEHR SEHR langsam wächst
- Man kann nicht gleichzeitig Find und Union auf O(1) bringen[2]

[1] Tarjan, Robert E.; van Leeuwen, Worst-case analysis of set union algorithms, Journal of the ACM 31 (2), S. 245–281, **1984**

[2] M. Fredman, M. Saks, The cell probe complexity of dynamic data structures, In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing., S. 345–354, **1989**