Union-Find-Strukturen

Union-Find Strukturen

Problem:

Verwalte eine Familie von paarweise disjunkten Mengen unter den Operationen:

e.make-set(): Erzeugt eine neue Menge mit Element *e*.

e.find-set(): Liefert diejenige Menge M_i, die das Element e enthält.

union(M_i, M_j)*:* Vereinigt die Mengen M_i und M_j zu einer neuen Menge.

Union-Find Strukturen

Repräsentation der Mengen M_i :

M_i wird durch ein repräsentatives (kanonisches) Element aus M_i identifiziert.

Union-Find Strukturen

Operationen mit repräsentativen Elementen:

e.make-set():

Erzeugt eine neue Menge mit einzigem Element *e*. Die Menge wird durch *e* repräsentiert.

e.find-set():

Liefert den Namen des Repräsentanten derjenigen Menge, die das Element *e* enthält.

e.union(f):

Vereinigt die Mengen M_e und M_f , die die Elemente *e* und *f* enthalten zu einer neuen Menge *M* und liefert ein Element aus $M_e \cup M_f$ als Repräsentanten von *M*. M_e und M_f werden zerstört.

Folgerungen

- Ist *n* die Anzahl der *e.make-set()* Operationen und werden insgesamt *m* Operationen *e.make-set(), e.find-set(), e.union(f)* ausgeführt, so ist
  - *m >= n*
  - nach höchstens *(n – 1)* union – Operationen besteht die Kollektion von Mengen nur noch aus einer einzigen Menge

Anwendungs-Beispiel: Zusammenhangstest

Input: Graph $G = (V,E)$

Output: die Familie der Zusammenhangskomponenten von G

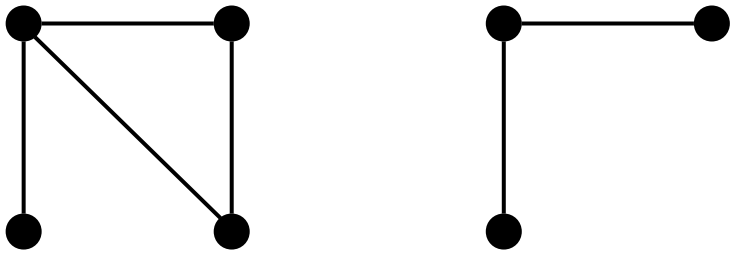
Algorithmus: Connected-Components

for all *v* **in** *V* **do** *v.makeset()*

for all *(u,v)* **in** *E* **do**

  if *u.findset()* $\neq$ *v.findset()*

    then *u.union(v)*

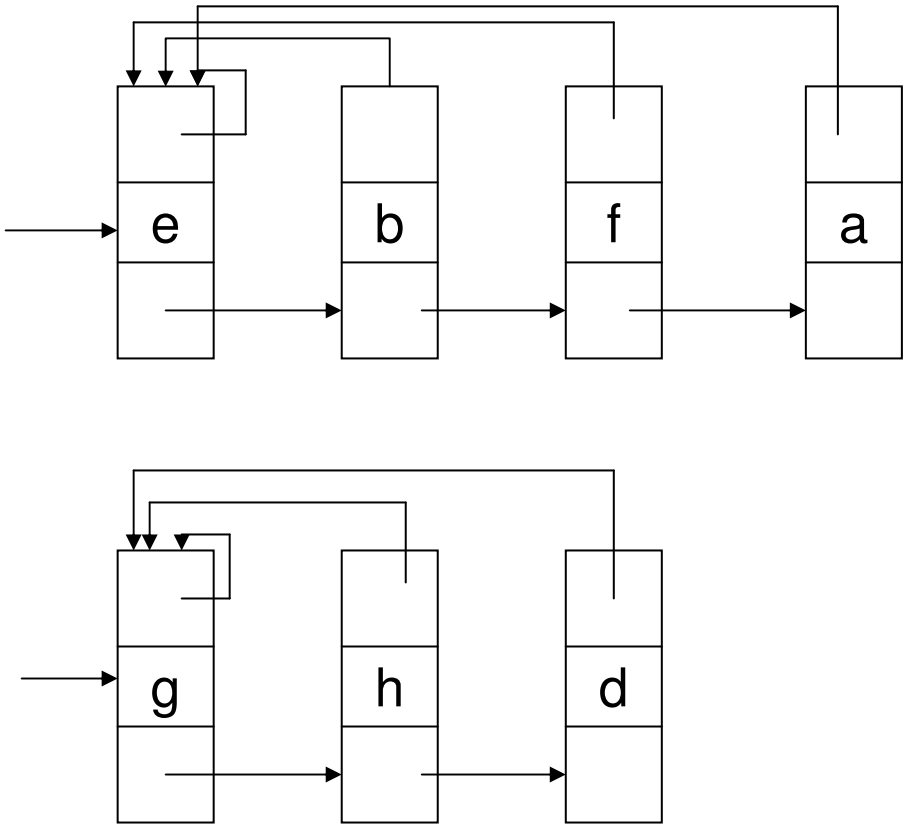


Same-Component *(u,v):*

  if *u.findset() = v.findset()*

  then return *true*

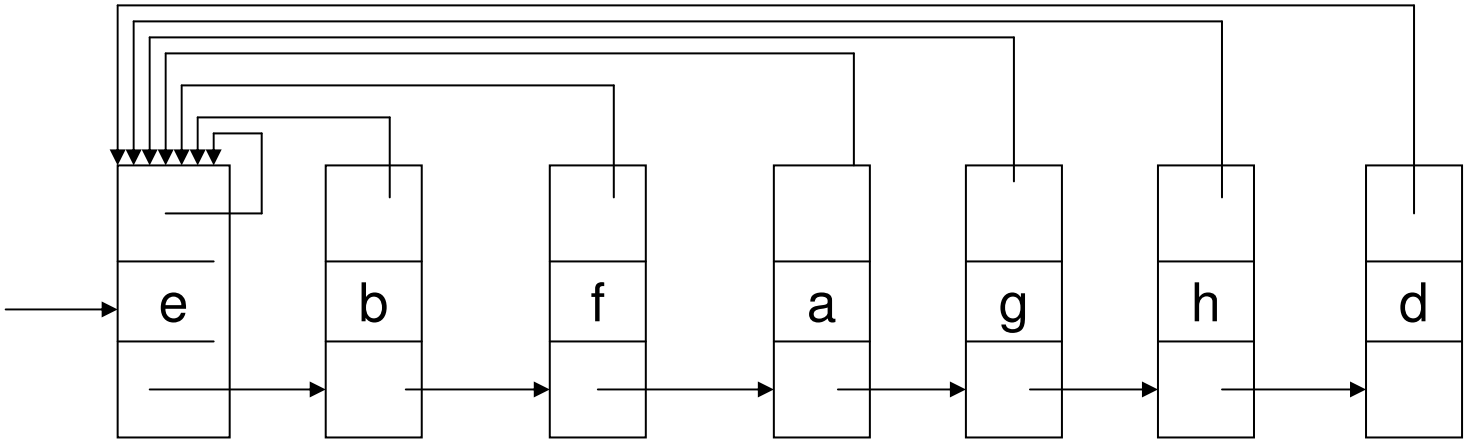
Repräsentation durch verkettete Listen



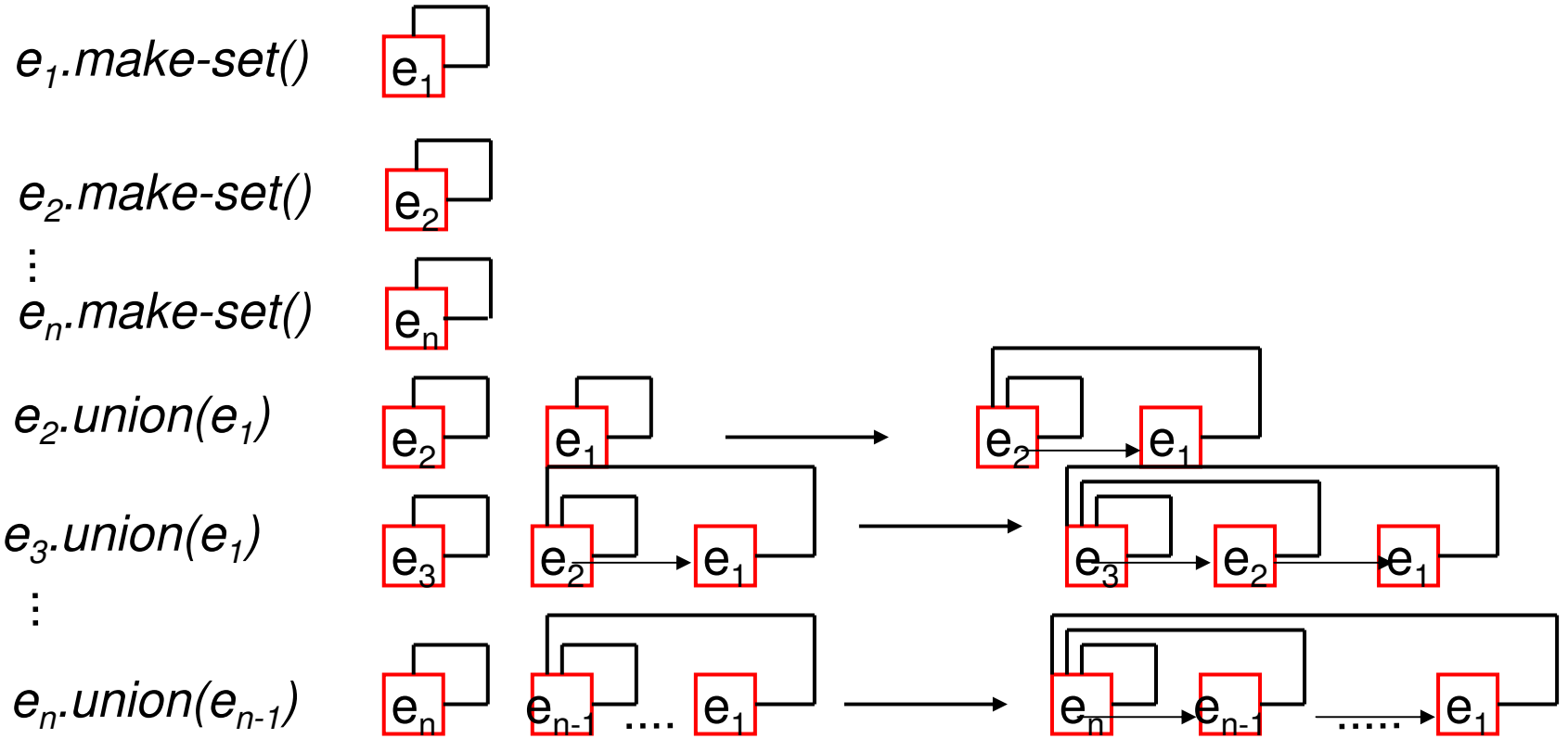
- *x.make-set()*
- *x.find-set()*
- *x.union(y)*

Repräsentation durch verkettete Listen

b.union(d)



Teure Operations-Folge



Längere Liste wird stets an kurze angehängt!

Zeigerzuweisungen im *i*-ten Schritt e_i.union(e_{i-1}):

2n Operationen kosten Zeit:

Verbesserung

Gewichtete Union-Heuristik

Hänge stets die kürzere an die längere Liste.
(Führe Listenlänge als Parameter mit).

Satz

Bei Verwendung der gewichteten Union-Heuristik kann jede Folge von *m make-set(), find-set(),* und *union()*-Operationen, die höchstens *n make-set()* Operationen enthält, in Zeit $O(m + n \log n)$ ausgeführt werden.

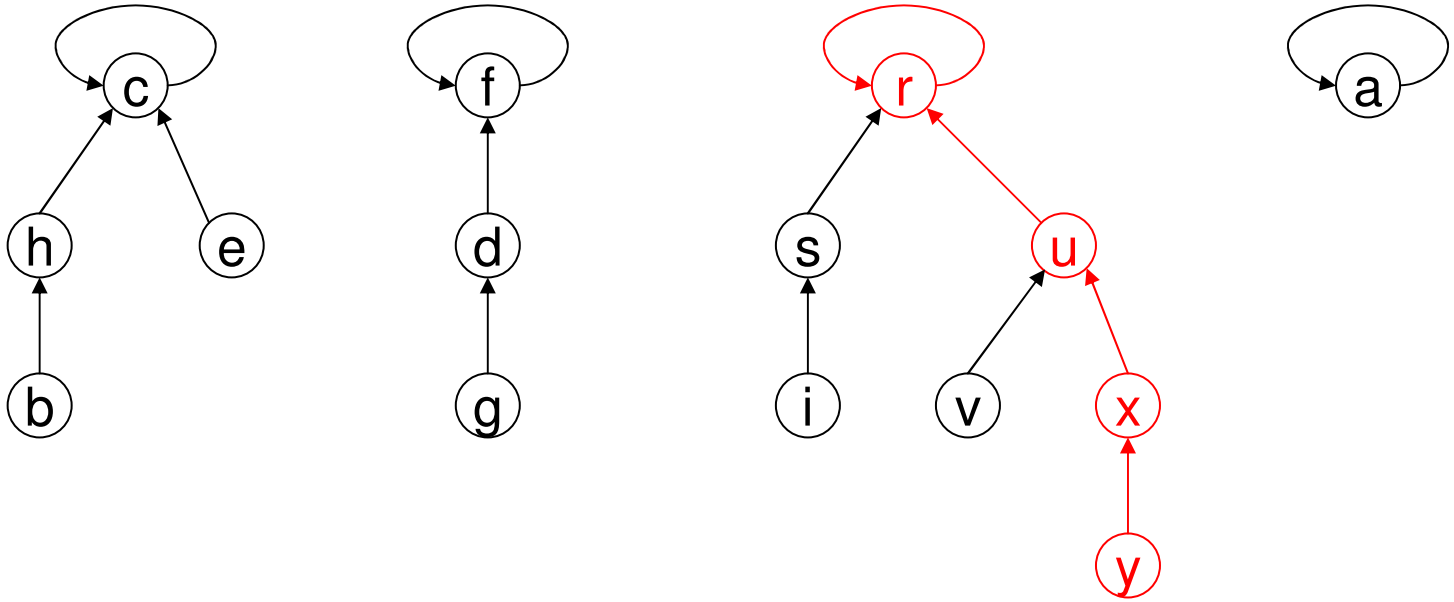
Beweis

Betrachte Element *e*

Anzahl der Änderungen des Zeigers von *e* auf das repräsentative Element:

log *n*

Repräsentation durch Wald von Bäumen



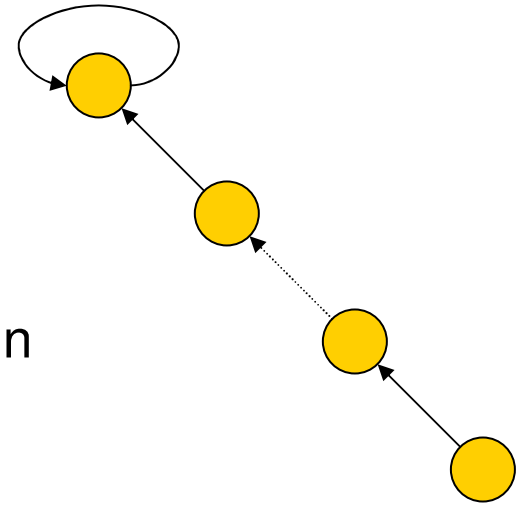
- a.make-set()
- y.find-set()
- d.union(e): mache den Repräsentanten der einen Menge (z.B. *f*) zum direkten Vater des Repräsentanten der anderen Menge.

Beispiel

m = Gesamtanzahl der Operationen (≥ 2n)

for *i* = 1 **to** *n* **do** e_i.*make-set( )*
for *i* = 2 **to** *n* **do** e_i.*union*(e_{i-1})
for *i* = *1* **to** *f* **do** e_1.*find-set( )*

n – ter Schritt



Kosten für f find-set Operationen: $O(f * n)$

Vereinigung nach Größe

zusätzliches Feld:

e.size = (#Knoten im Teilbaum von *e*)

e.make-set()

1 *e.parent = e*

2 *e.size* = 1

e.union(f)

1 *Link(e.find-set( ), f.find-set( ))*

Vereinigung nach Größe

Link(e,f)

```
1 if e.size ≥ f.size
2     then f.parent = e
3          e.size = e.size + f.size
4     else /* e.size < f.size */
5          e.parent = f
6          f.size = e.size + f.size
```

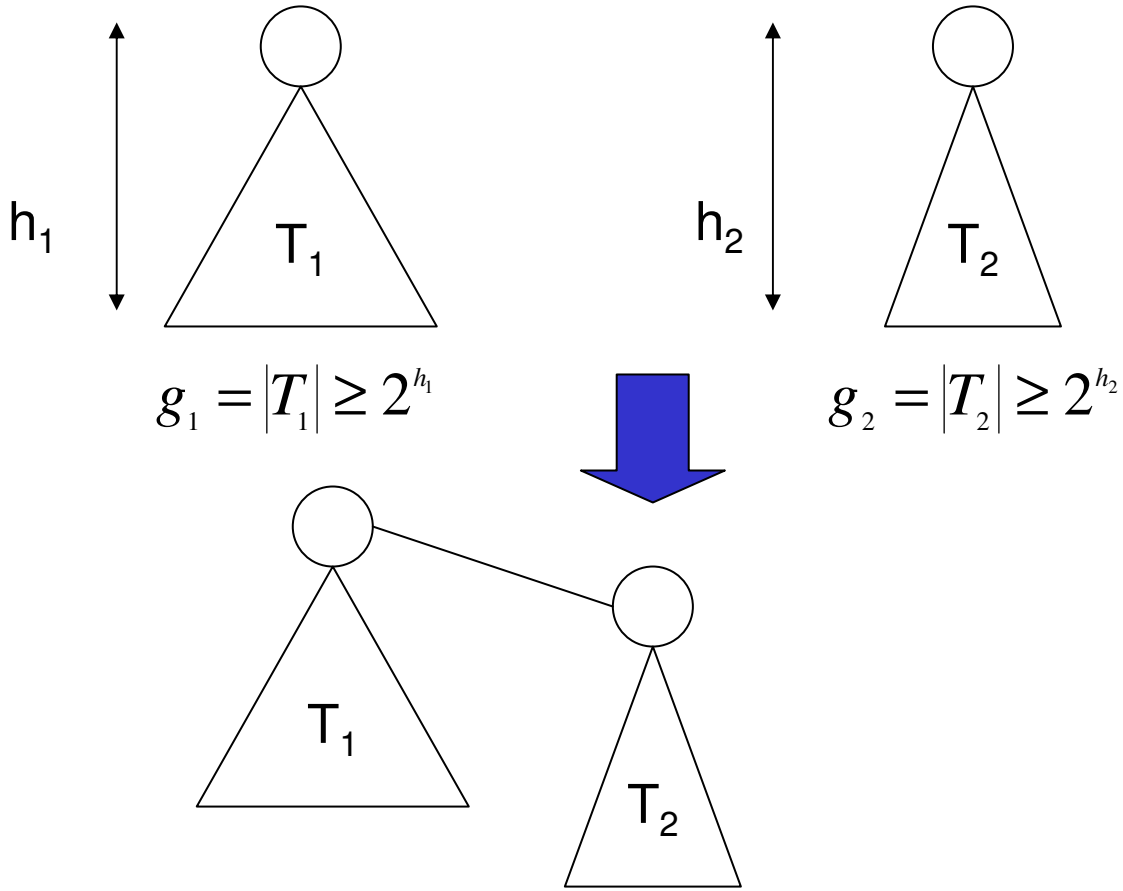
Vereinigung nach Größe

Satz

Das Verfahren Vereinigung nach Größe hat folgende Invariante:

Ein Baum mit Höhe *h* hat mindestens 2^h Knoten

Beweis



Vereinigung nach Größe

Fall 1: Der neue Baum ist genauso hoch wie T_1

$$g_1 + g_2 \geq g_1 \geq 2^{h_1}$$

Fall 2 : Der neue Baum ist gewachsen

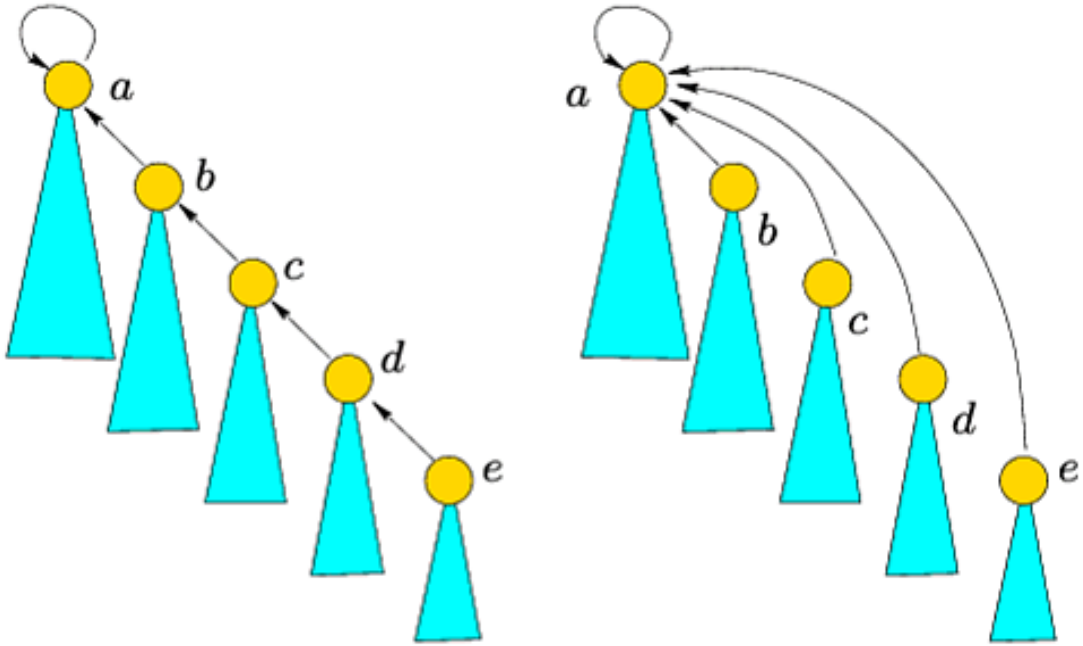
Höhe von T: $h_2 + 1$

$$g = g_1 + g_2 \geq 2^{h_2} + 2^{h_2} = 2^{h_2+1}$$

Konsequenz

Eine *find-set*-Operation kostet höchsten O(*log n*) viele Schritte, falls die Anzahl der *make-set*-Operationen gleich *n* ist.

Pfadverkürzung bei Find-Operation




```
e.find-set()
1   if e ≠ e.parent
2        then e.parent = e.parent.find-set( )
3   return e.parent
```

Analyse der Laufzeit

m Gesamtanzahl der Operationen, davon

f *find-Set*-Operation und

n *make-Set*-Operation.

→ höchstens $n - 1$ *union*-Operationen

Vereinigung nach Größe:

$O(n + f \log n)$

Find-Operation mit Pfadverkürzung:

Falls $f < n$, $\Theta(n + f \log n)$

Falls $f \geq n$, $\Theta(f \log_{1+f/n} n)$

Analyse der Laufzeit

Satz (Vereinigung nach Größe und Pfadverkürzung)

Bei der Verwendung der Strategie *Vereinigung nach Größe* und *Pfadverkürzung* benötigen m Union-Find-Operation über n Elementen $\Theta(m * \alpha(n))$ Schritte.

$\alpha(n)$ = Inverse der Ackermann-Funktion

Ackermann-Funktion und Inverse

Ackermann-Funktion

$$A(0,j) = j+1$$

$$A(k,j) = A^{(j+1)}(k-1,j) \qquad \text{für } k \geq 1$$

$$A^{(i+1)}(k-1,j) = A(k-1,A^{(i)}(k-1,j))$$

Inverse Ackermann-Funktion

$$\alpha(n) = \min\left\{ i \geq 1 \middle| A(i,1) > n \right\}$$