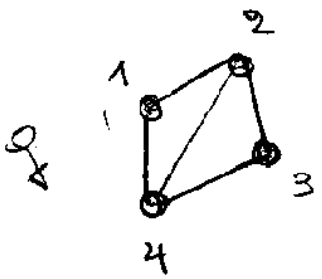
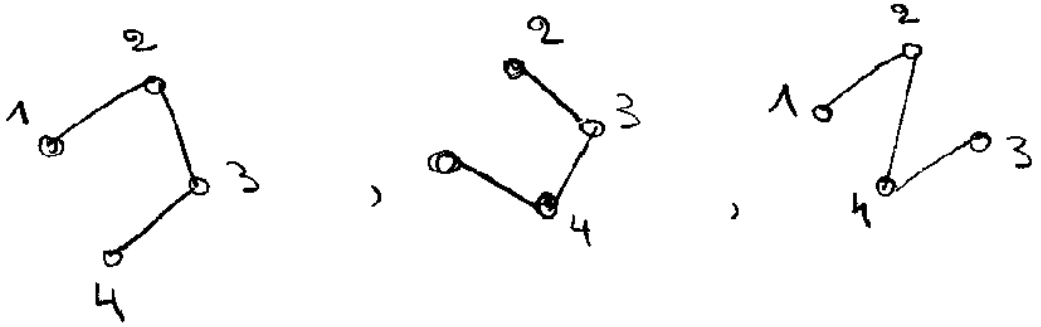
# 7. Minimaler Spannbaum und Datenstrukturen

Hier betrachten wir als Ausgangspunkt stets zusammenhängende, ungerichtete Graphen.




so sind




Spannbäume (aufspannende Bäume) von G. Beachte immer 3 Kanten, bei 4 Knoten.

## Definition (Spannbaum)

Ist $G = (V, E)$ zshg., so ist ein Teilgraph $H = (V, F)$ ein Spannbaum von $G$

gdw.

$H$ ist zusammenhängend und für alle $e \in F$ ist $H \setminus \{e\} = (V, F \setminus \{e\})$ nicht mehr zusammenhängend.

($H$ ist minimal zusammenhängend). □

## Folgerung

Sei $H$ zshg., Teilgraph von $G$.

$H$ Spannbaum von $G$

$\Leftrightarrow$

$H$ ohne Kreis.

## Beweis:

„⇒" Hat $H$ Kreis, dann gibt es Kante $f$, so daß $H \setminus \{f\}$ zshg.

$A \Rightarrow B$ äqu $\neg B \Rightarrow \neg A$

„⇐" Sei $H$ kein Spannbaum, dann gibt es Kante $f$, so daß $H \setminus \{f\}$ zshg.

Dann haben wir einen Kreis in $H$, der $f$ enthält.

$B \Rightarrow A$ äqu $\neg A \Rightarrow \neg B$

□

Damit hat jeder Spannbaum genau $n-1$ Kanten, wenn $|V| = n$ ist. Vergleiche S. 2.20 der Handschrift. Die Baumkanten eines Suchbaums ergeben einen solchen Spannbaum. Die einzelnen Spannbäume minimalen Kosten.

## Definition

Seien $G = (V, E)$ und $k: E \to \mathbb{R}$ (eine Kostenfunktion) gegeben.

Dann ist $H = (V, F)$ ein minimaler Spannbaum falls.

- $H$ ist Spannbaum von $G$
- $k(H) = \sum_{f \in F} k(f)$ ist minimal unter den Kosten aller Spannbäume.

($k(H)$ = Kosten von $H$). □

Wie finde ich einen minimalen Spannbaum? Systematisches Durchprobieren. Verbesserung durch branch-and-bound.

(Y.5)

Eingabe $G = (V, E)$, $w : E \to \mathbb{R}_+$

$E = \{e_1, \dots, e_m\}$ (also $m$ Kanten)

Systematisches Durchgehen aller Mengen von $n-1$ Kanten (alle Bitvektoren $b_1 \dots b_m$ mit genau $n-1$ Einsen), z.B. lexikogr.

Testen ob kein Kreis.

Kosten ermitteln, Den mit kleinstem Kosten ausgeben.

Zeit: Mindestens

$$\binom{m}{n-1} = \frac{m!}{(n-1)!\,(m-n+1)!}$$

für $n-1 \le m$; $(m-n+1)! \ge 0$

Worst case ist das $\frac{m}{2}$ $m = 2n$, dann

$$\frac{(2n)!}{(n-1)!\,(n+1)!}$$

$$= \frac{2n\,(2n-1)\cdot(2n-2)\cdots n}{(n+1)\,n\,(n-1)\cdots 1}$$

$$\geq \frac{2n-2}{\underbrace{n-1}_{\geq 2}} \cdot \frac{2n-3}{\underbrace{n-2}_{\geq 2}} \cdots \frac{2}{\underbrace{1}_{\geq 2}}$$

$$\geq \underbrace{2 \cdot 2 \cdots 2}_{n-1 \text{ - mal}} = 2^{n-1}$$

Also $\in \Omega(2^n)$

Regel: $c \geq 0$

$\frac{a}{b} \leq \frac{a-c}{b-c}$

$\Leftrightarrow b \leq a.$

Das $c$ im Nenner hat mehr gewicht.

**Definition** ($\Omega$-Notation)

Seien $f, g : \mathbb{N} \to \mathbb{R}$, so $f(n)$ ist $\Omega(g(n))$

gdw. es gibt eine Konstante $\zeta > 0$

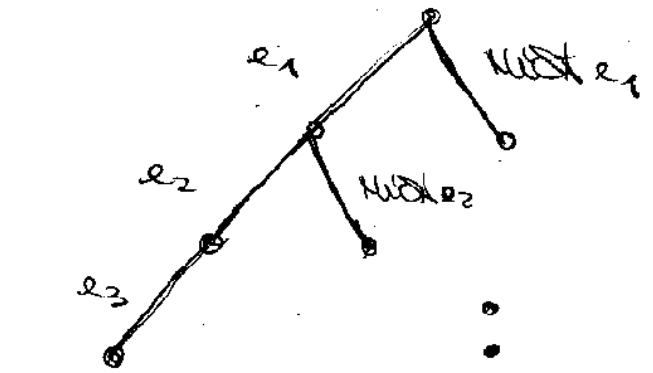
mit $|f(n)| \geq \zeta \cdot g(n)$ für alle hinreichend großen $n$

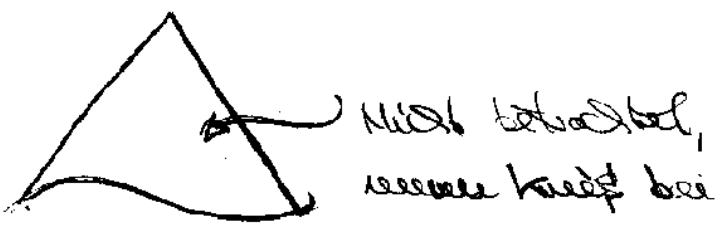
Vgl. O-Notation, S. 3.10, ff. □

O - Notation: Obere Schranke

Ω - Notation: Untere Schranke.

Verbesserung: Backtracking, frühzeitiges Erkennen von Knoten: Aufbau eines Baumes (Prozedurenaufrufbaum):

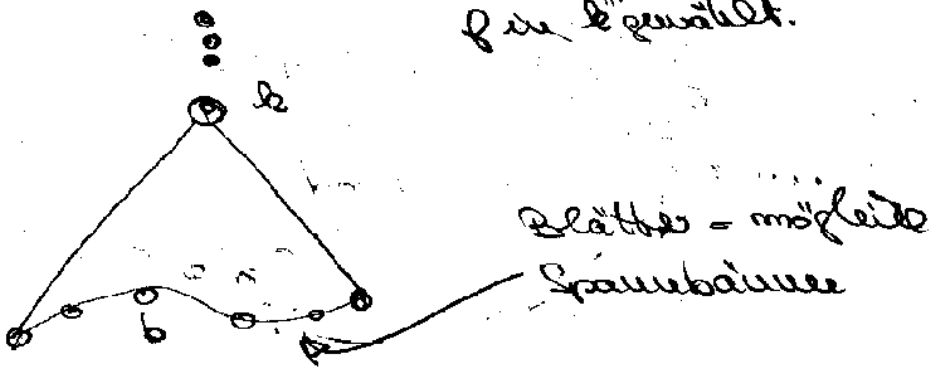



Alle Teilwege
mit $e_1, e_2, e_3$
sind in einem
Schritt erledigt, wenn
kein vorliegt.

Einbau einer Kostenschranke in den Backtracking Algorithmus: Branch-and-bound.

Kosten der Kante $k = \sum_{f} k(f)$.
$f$ ist $k$ zugeteilt.




Es sei für Blätter wie b unter k

Kosten von $k \leq$ Kosten von b.

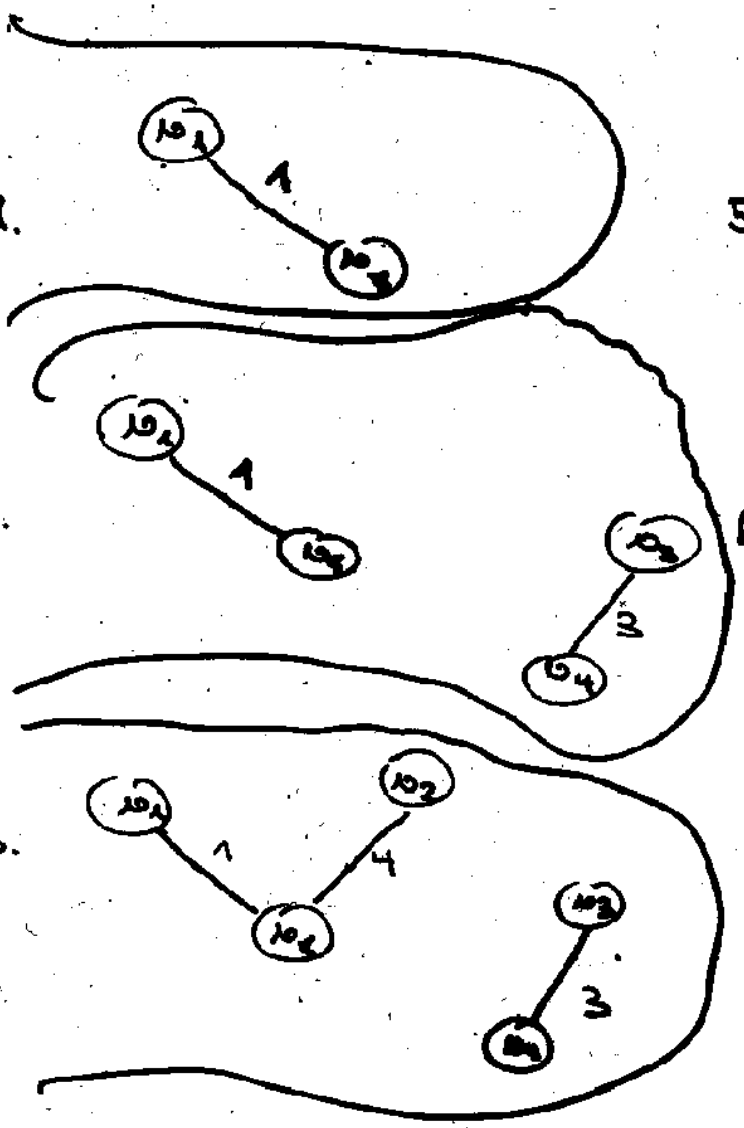
↑
Untere Schranke an die Spannbäume unter k.

Also weiteres Ausschneiden möglich, wenn Kosten von $k \geq$ Kosten eines bereits gefundenen Baumes. Im allgemeinen ohne vorheriges Kennen besserer Lösung nachinvestiert. Besser: Interaktives Aufbauen eines min. Sb.

Diese Seite mußte aus rechtlichen Gründen entfernt werden!

Diese Seite mußte aus rechtlichen Gründen entfernt werden!

# AUFBAU EINES MINIMALEN SPANNBAUMS



# IMPLEMENTIERUNG DURCH PARTITION VON V

| | KEINE KANTE |
|---|---|
| $\{v_1\}, \{v_2\}, \ldots, \{v_7\}$ | |
| $\{v_1, v_7\}, v_2, v_3, v_4, v_5$ | $\{v_1, v_7\}$ |
| $\{v_1, v_7\}\ \{v_3, v_4\}, \ldots$ | $\{v_1, v_7\}\ \{v_3, v_4\}$ |
| $\{v_1, v_7, v_2\}\ \{v_3, v_4\}, \ldots$ | $\{v_1, v_7\}, \{v_3, v_4\}, \{v_7, v_2\}$ |
| $\{v_1, v_7, v_2, v_3, v_4\}, \ldots$ | $\{v_1, v_4\}, \{v_3, v_4\}, \{v_7, v_2\}, \{v_7, v_3\}$ |
| $v_5$ HINZU | |
| $v_6$ HINZU | |

PROBLEM: WIE PARTITION DARSTELLEN?

# Algorithmus (Minimale Spannbaum, Kruskal 1956)

Eingabe: $G = (V, E)$ zshg., $V = \{1, \dots, n\}$,
Kostenfunktion $k : E \to \mathbb{R}$

Ausgabe: $F$ = Menge von Kanten eines minimalen Spannbaums.

1. $F = \emptyset$, $P = \{\{1\}, \dots, \{n\}\}$ // $P$ die Partition
2. $E$ nach Kosten sortieren // geht in
   // $O(E \log|E|) = O(|E| \log M)$
   // $|E| \leq |V|^2$, $\log|E| = O(\log M)$
3. while $|P| \geq 1$ { // solange $P \neq \{\{1, \dots, n\}\}$
4. $\{v, w\} :=$ kürzeste (= erste) Kante von $E$
   $\{v, w\}$ aus $E$ löschen
5. Testen ob $F \cup \{v, w\}$ kreis hat.
6. if $\{v, w\}$ induziert keinen Kreis {
   $U_v :=$ die Menge mit $v$ aus $P$;
   $U_w :=$ " " $w$ " ;
7. $U_v$ und $U_w$ in $P$ vereinigen
   $F := F \cup \{\{v, w\}\}$ } }

□

Der Algorithmus arbeitet nach dem Greedy Prinzip (Greedy = gierig):
Es wird an jedem Zeitpunkt die günstigste Wahl getroffen
(= Kante mit den kleinsten Kosten, die möglich ist) und diese genommen.
Eine einmal getroffene Wahl bleibt bestehen. Die lokal günstige Wahl führt zum globalen Optimum.

Zur Korrektheit des Algorithmus:
Sei

$F_e$ = der Inhalt von F nach dem $e$'ten Lauf der Schleife.

$P_e$ = " P "

Wie verwendet die Invariante:

$F_\ell$ läßt sich zu einem minimalen Spannbaum fortsetzen. (D.h. es gibt $F \supseteq F_\ell$, so daß $F$ ein minimaler Spannbaum ist.)

$P_\ell$ stellt die Zusammenhangskomponenten von $F_\ell$ dar.

Invariante gilt für $\ell = 0$.

Gelte sie für $\ell$ und finde ein $\ell+1^{\text{te}}$ Lauf statt.

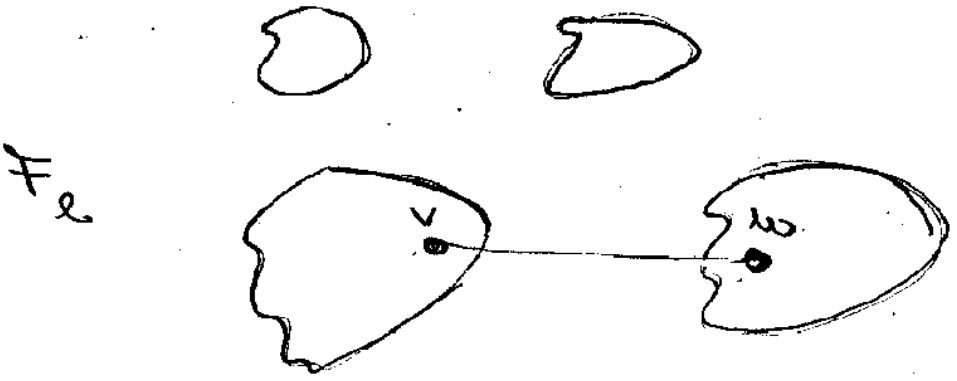
1. Fall, die Kante $\{x, w\}$ wird nicht genommen.

Alles unverändert. Invariante gilt für $\ell+1$.

2. Fall, $\{v,w\}$ wird genommen.

$\{v,w\}$ = Kante von minimalen Kosten, die 2 Zshg'skomp. von $F_e$ verbindet.

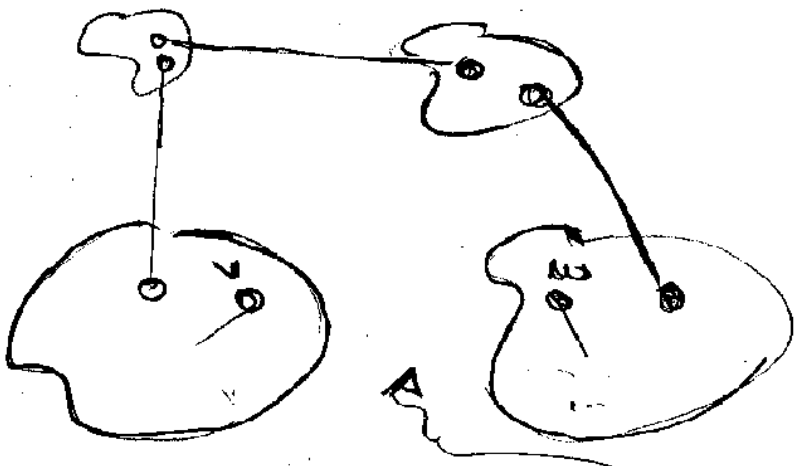
Also liegt folgende Situation vor:




Zu zeigen: Es gibt minimalen Sb. der $F_{e+1} = F_e \cup \{v,w\}$ enthält.

Nach Invariante für $F_\ell$ gibt es min. 2b $F \supseteq F_\ell$. Halten wir ein solches $F$ fest. Dieses $F$ kann, muß aber nicht $\{v,w\}$ enthalten.

$F_\ell \subseteq F$




Ein Beispiel für $F$ { keine Kante dazwischen.

Falls $F$ $\{v,w\}$ enthält gilt die Invariante für $\ell+1$ (sofern die Operation auf $P$ richtig implementiert ist).

Falls aber $F$ $\{v,w\}$ nicht enthält augmentieren wir so:

$F \cup \{v,w\}$ enthält einen
kreis. (sonst $F$ nicht zshg.)

Dieser Kreis muß mindestens
weitere Kante haben, die
2 verschiedene Komponenten von
$F_e$ verbindet, also nicht zu
$F_e$ gehört. Diese Kante
gehört zu der Liste von Kanten
$E_{e'}$, hat also Kosten,
die höchstens größer als
die von $\{v,w\}$ sind.

Dazu evtl. noch weitere Invariante

Tauschen wir in $F$ diese Kante
und $\{v,w\}$ aus haben wir einen
minimalen Spannbaum mit $\{v,w\}$.

Also gilt die Invariante

für $F_{e+1} = F_e \cup \{s, ss\}$ und

$P_{e+1}$ (also Operationen richtig

implementiert sind.)

Konsistenz: Am Ende ist also hat

Fe nur noch eine Komponente,

da $|P_e| = 1$. Also minimales

Spannbaum, wegen Invariante

Terminierung: Entweder wird

die Liste $E'_e$ kleiner oder $P_e$ wird

kleiner.

Laufzeit:

1. $O(m)$
2. Kanten sortieren:

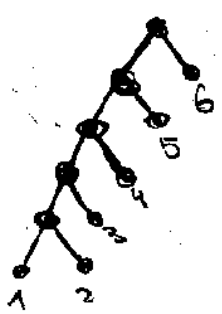
   $O(|E| \log |E|)$ (wird später behandelt.)

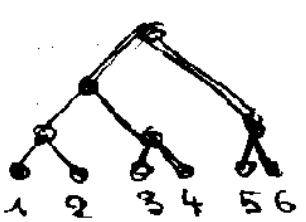
   also $O(|E| \log |M|)$ !

3. Die Schleife

   • $n-1 \leq$ # Läufe $\leq |E|$

   Mächtigkeit $\{\{1\}, \ldots, \{m\}\}$ zu $\{\{1, \ldots, m\}\}$, machen. Jedesmal eine Menge weniger, da 2 vereinigt. Also $(n-1)$ Vereinigungen. Egal wie vereinigt wird:



Binäres Baum mit $n$ Blättern
hat immer genau $n-1$ Nicht-Blätter.

- Teste $\{v, w\}$: suche ob es
  Kante in $F$ indiziert, d.h. ob $v, w$
  innerhalb eines Baums von $P$ adjazent.

  Bis zu $|E|$-mal

- Wird $UD_v$ mit $UD_w$ in $P$ vereinigt

  genau $n-1$-mal.

Zeit hängt an der Darstellung von $P$.

Einfache Darstellung: array $P[1,\dots,n]$
wobei eine Partition von $\{1,\dots,n\}$,
die Indexmenge dargestellt wird durch:

$u, v$ in gleicher Menge von $P \Leftrightarrow P[u] = P[v]$
A A — Elemente der Grundmenge = Indices
R

Die Kante $\{u,v\}$ induziert kreis

$\Leftrightarrow$ $u,v$ in derselben Zusammenhangskomp. vor T

$\Leftrightarrow$ $P[u] = P[v]$

$W_u$ und $W_v$ vereinigen:

1. $a = P[v]$ ; $b = P[w]$ // $a \neq b$, wenn // $W_u \neq W_v$
2. for $i = 1, \ldots, n$ {
   if $P[i] = a$ {
   $P[i] = b$
   }
   }

Durch Laufzeit der Schleife:

1. insgesamt $n \cdot O(n) = O(n^2)$

(Es wird $n-1$ - mal vereinigt.)

5. insgesamt $O(|E|)$

3. $O(n)$ insgesamt, wenn $|P|$ mitgeführt.

Also $O(n^2)$, wenn E bereits sortiert ist.

Darstellung von P durch eine Union-Find Struktur:

- Darstellung einer Partition P einer Grundmenge (hier klein, d.h. als Indexmenge verwendbar)
- Für $A, B \in P$

  union(A, B)

  soll $A, B \in P$ durch $A \cup B \in P$ ersetzen
- Für $a$ aus der Grundmenge

  Find(a)

  soll Name der Menge $A \in P$ mit $a \in A$ geben. (Also es gibt auch eine Funktion von a!)

Für Kruskal

$\leq 2|E|$ - mal Find(u) + $|V|-1$ - mal Union(u,v)

$\Rightarrow$

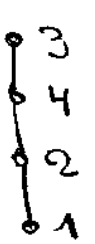
Zeit $\Omega(|E|+|V|)$, sogar falls $|E|$ bereits sortiert ist.

## Datenstrukturen (Union-Find Strukturen)

(a) Jede Menge von P als ein Baum mit Wurzel.

$P = \{\{1,2,3,4\},\{5,6\}\}$, dann etwa Wurzel = Name der Menge.


oder auch





$\rightarrow \cdots$ Struktur der Bäume egal!

4.95

(b) Find (u)

1. u in den Bäumen finden
   // kein Problem, solange
   // Knotenmenge Indexmenge
   // sein kann.

2. Gehe von u aus hoch bis zur Wurzel.

3. (Name der) Wurzel ausgeben.

Union(u,v) // u,v sind Namen von
// Mengen, der Wurzeln

1. u,v in den Bäumen finden.

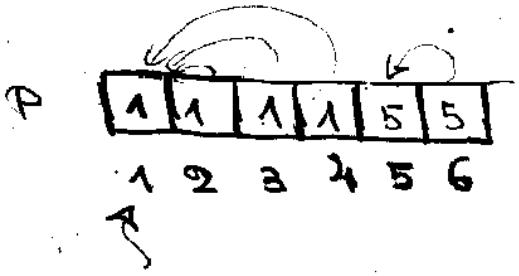
2. Hänge u unter v (oder umgekehrt)


(c) Bäume in array (Vaterarray):

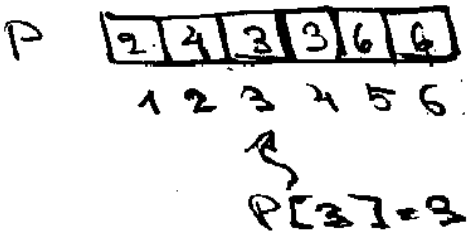
P[1,-,n] ∈ 1-n

P[v] = Vater von v.

Idee (a) weiter:




1 Wurzel, deshalb P[1] = 1.




Wieder wichtig:
Index = Element.
Vorsorge für beliebige Bäume.

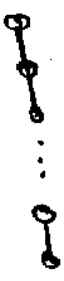
Union (u,v) // u,v Wurzeln

P[u] := v //  O(1)

Find (v)

while P[v] ≠ v {
    v := P[v]
}
Ausgabe von v.

// O(n) im // worst case

// 

1. ① ② ③ ④ ... ⓝ

P = [1|2|3| ..... |N]
1 2 3 N

↓ UNION(1,2) $O(1)$

2. ② ③ ... ⓝ

P = [2|2|3|...|N]
1 2 3 N

↓ UNION(2,3) $O(1)$

3. ③

P = [2|3|3|]
1 2 2

↓ UNION(3,4)

4. ④

DAS ENTSTEHEN LANGER DÜNNER BÄUME.

⋮

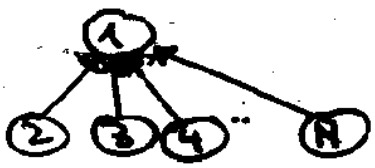
N. ⓝ

P = [2|3|3|3|...|N]
1 2 2 1 N

FIND(x) IN $O(\log n)$

① ② ③ ④ ... (N-1) (N)

UNION(2,1); UNION(3,1); UNION(4,1); ..., UNION(N,1)




VERTAUSCHEN DER ARGUMENTE.

FIND(x) IN O(1).

⇒ UNION BY SIZE: KLEINERE ANZAHL NACH UNTEN.

⇒ MAXIMALE TIEFE $\log_2 N$.

LÄNGSTER WEG VON WURZEL ZU BLATT.

BEACHTE:

$\log_2 N$ ZU $N = 2^{\log_2 N}$.

WIE

$N$ ZU $2^N$

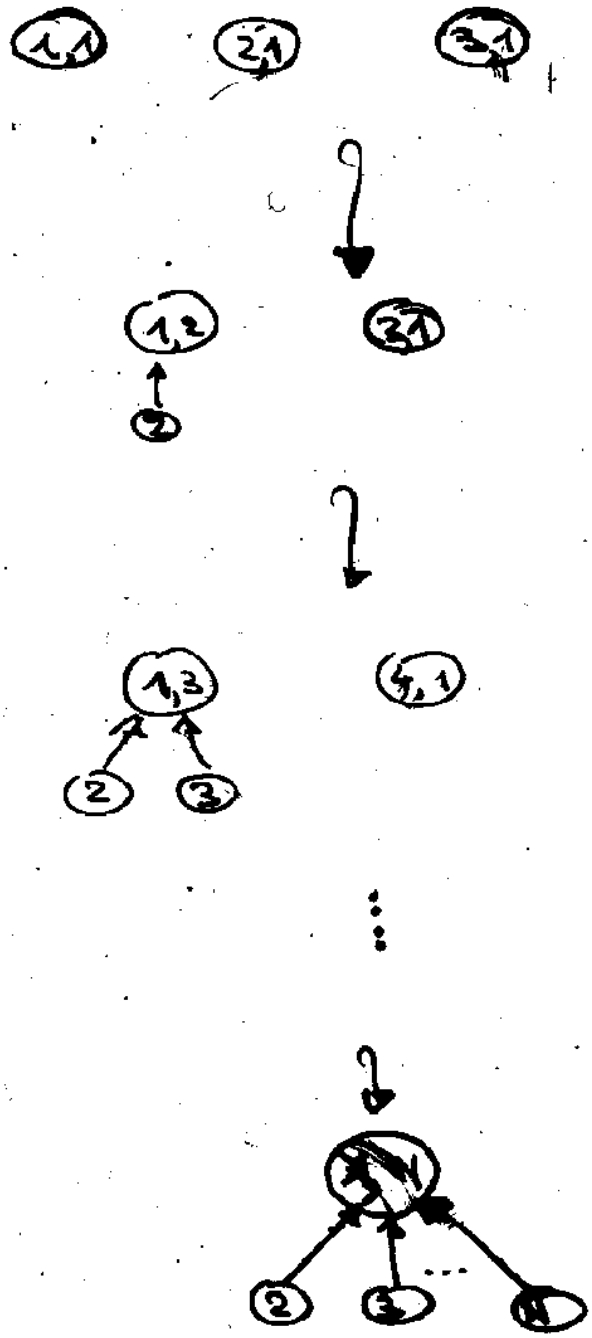
Algorithmus (union-by-size)

union ( (v,i), (w,j) ) // i,j ≠ Elemente in
// den Mengen, die
// vereinigt werden.

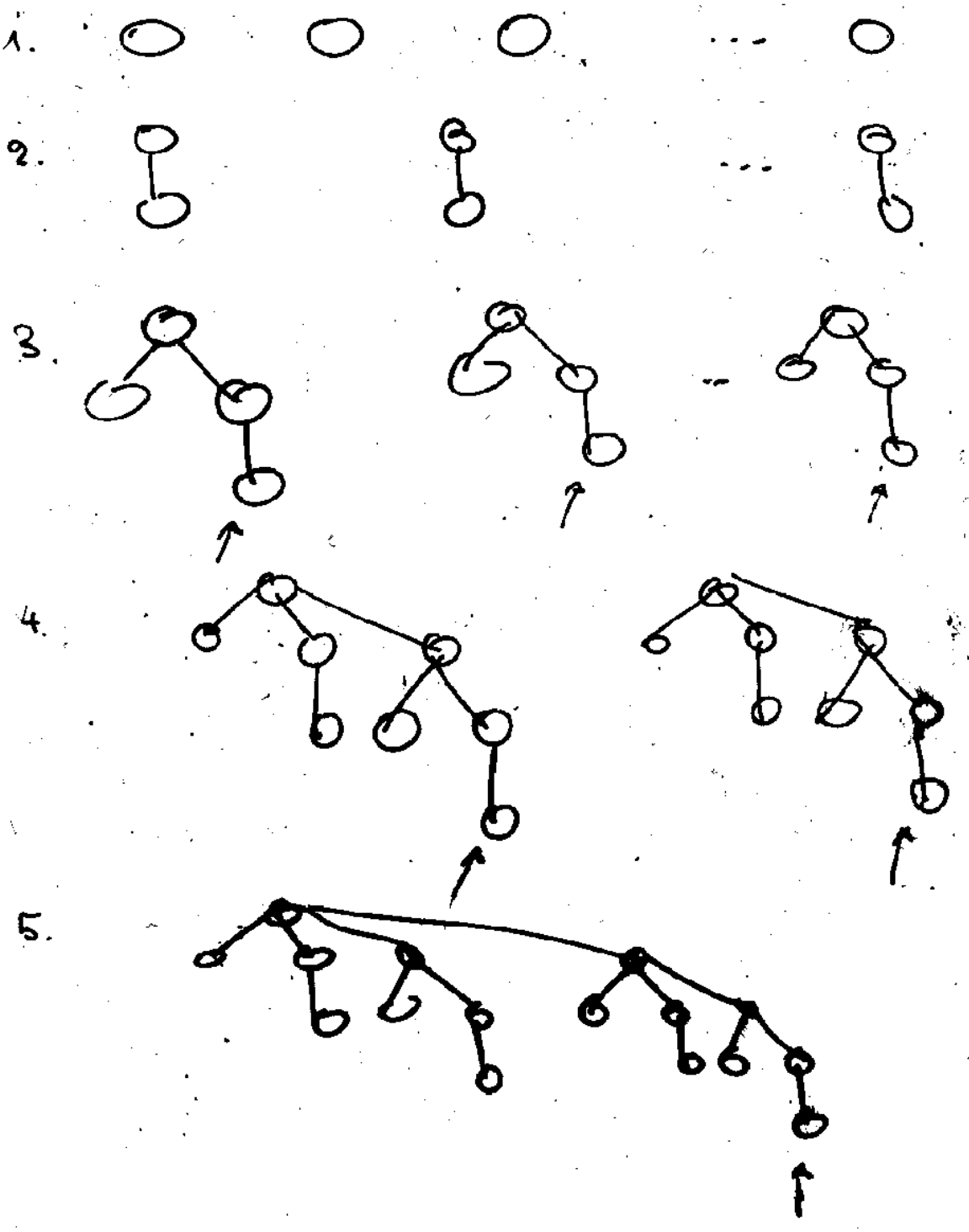
{ if i ≤ j {


} else { „umgekehrt" } }

UNION BY SIZE:



# TIEFE BÄUME DURCH UNION-BY-SIZE




VERMUTUNG: N ELEMENTE

$\Rightarrow$ TIEFE $\leq \log_2 N$.

Satz

Sei $n$ mit $P = \{\{1\}, \{2\}, \ldots, \{n\}\}$ gilt, daß mit Union-by-size für jeden entstehenden Baum $T$ gilt

$$\text{Tiefe}(T) \leq \log_2 |T|$$

Beweis $|T| = \#\text{Elemente von } T - \#\text{Knoten von } T.$

Induktion über die # ausgeführten Operationen Union$(u,v)$ um $T$ zu bekommen.

Induktionsanfang, kein Union$(u,v)$ ✓.

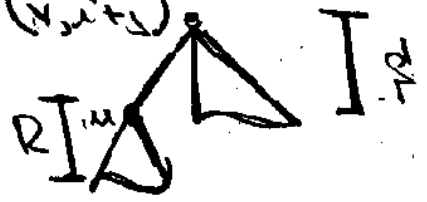
$$\text{Tiefe}(\bullet^u) = 0 = \log_2 1 \quad (2^0 = 1)$$

Induktionsschluß

$T$ durch Union$(u,v)$ sei

$R = \overset{(u,i)}{\wedge}$ $\quad S = \overset{(v,j)}{\wedge}$

Sei $i \leq j$, dann

$T =$ 


1. Fall, keine größere Tiefe als vorher.

Die Behauptung gilt nach Ind.-Vor, da T mehr Ebenen hat als R.

2. Fall, Tiefe vergrößert sich echt

Dann aber

$$\text{Tiefe}(T) = \text{Tiefe}(R) + 1 \quad \text{(Ind.-Vor)}$$
$$\leq (\log_2 |R|) + 1$$
$$= \log_2(2|R|)$$
$$\leq |T|$$

$2^x = |R| \Rightarrow 2 \cdot 2^x = 2^{x+1} = 2|R|$

Induktion über Größe. Tiefe wächst nur um 1, wenn Größe mindestens Verdopplung der # Ebenen.

□

Mit Bäumen und Union-by-size

Laufzeit der Schleife:

$\leq 2|E|$-mal Find(u): $O(|E| \underbrace{\log |V|}_{\text{Tiefe des Baums}})$

$n-1$-mal Union(u,v): $O(n)$

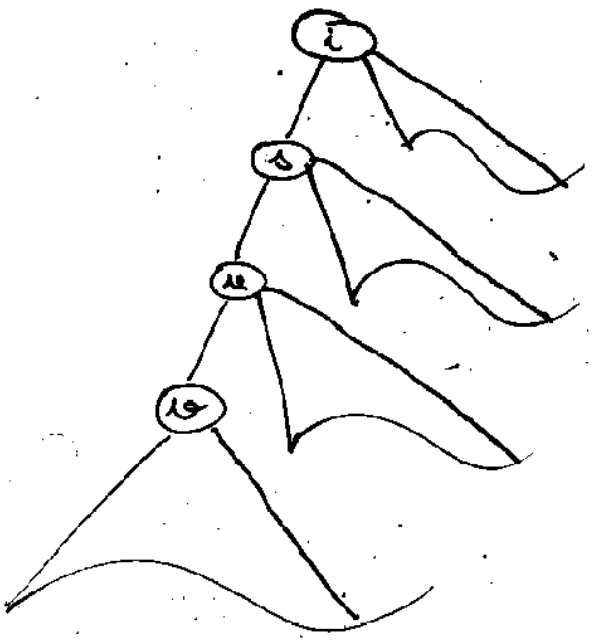
Real $O(|E|)$. Also insgesamt

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

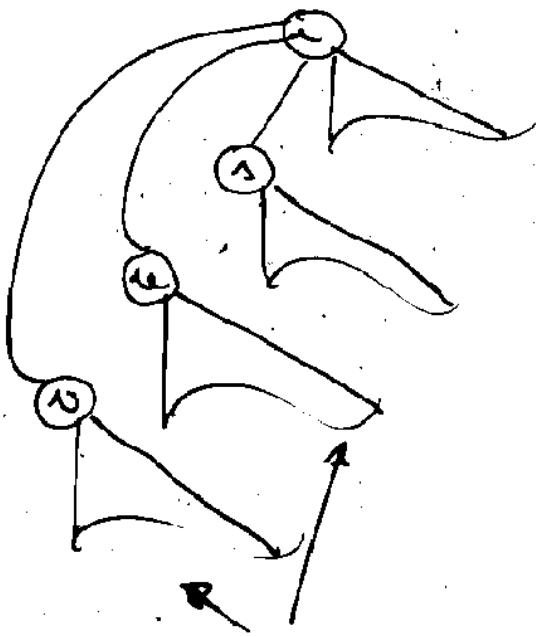
Vorne: $O(|V|^2)$. Für dichte Graphen, $|E| = \Omega\left(\frac{|V|^2}{\log |V|}\right)$

so keine Verbesserung erlaubt.

# BEISPIEL WEGKOMPRESSION



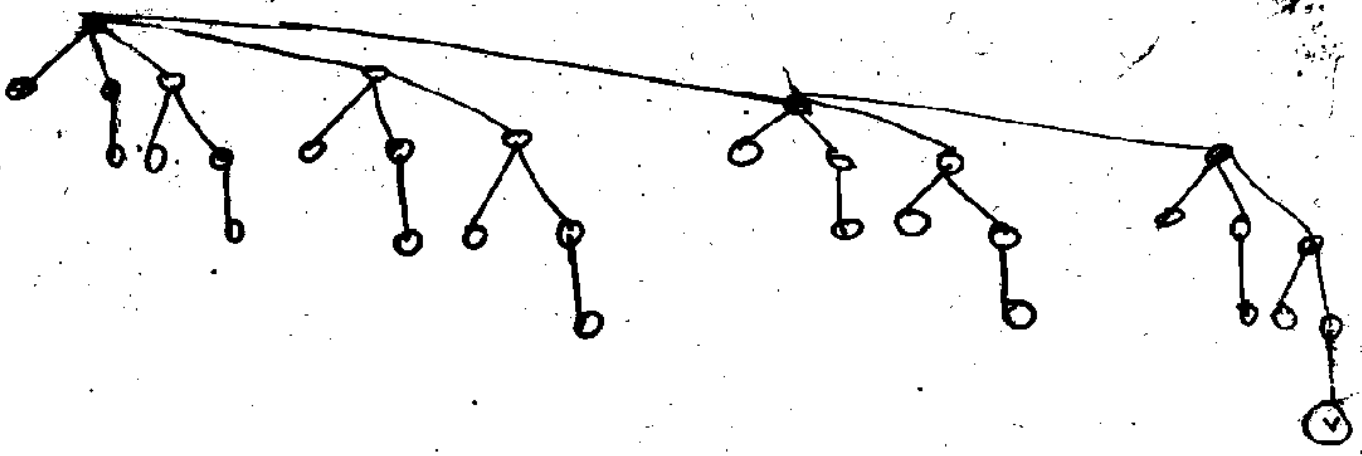

FIND(v)

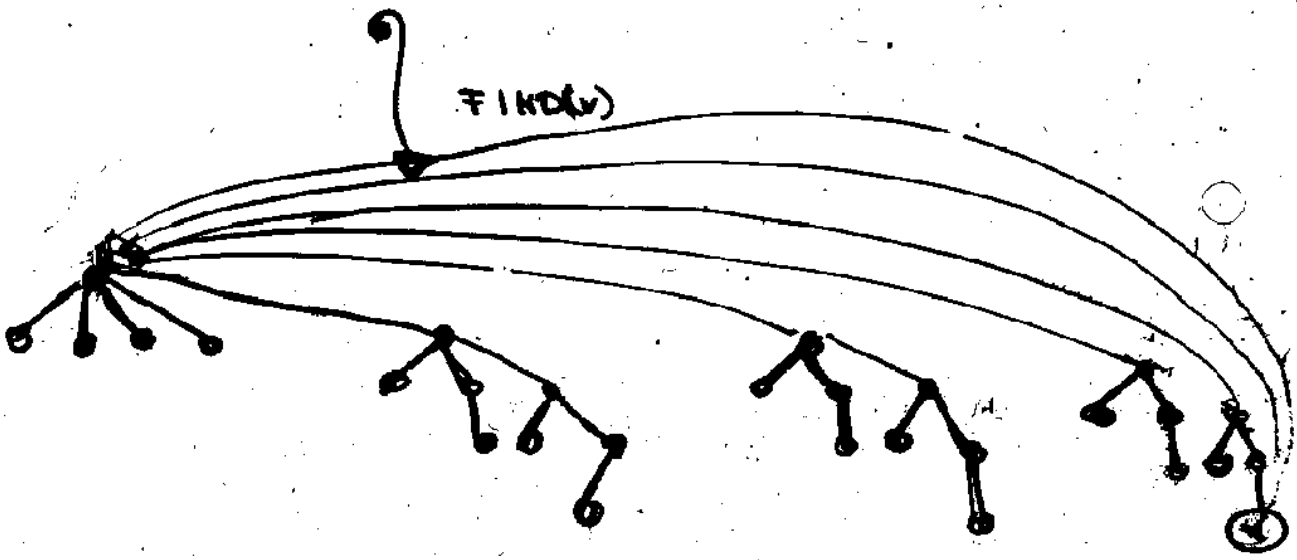



HIER NACHFOLGENDE FIND'S SCHNELLER.

# WEG KOMPRESSION






IM ALLGEMEINEN $\Omega(\log N)$ UND $O(\log N)$

VERGLEICHE BÄUME AUS S. 7.31

# Algorithmus (Wegkompression)

Find (v)

1. v auf Keller tun // Etwa so als
2. while P[v] ≠ v { // noch umhängen,
3. v := P[v]; // (mit Schlange, Kap. 2)
   v auf Keller tun
   }
4. v ausgeben;
5. for w auf dem Keller {
   P[w] := v // können auch
   } // Schlange oder
   // Liste oder sonstwas
   // nehmen, da Reihen-
   // folge egal!

Laufzeit $O(\log|V|)$ wie zuvor,
falls Union-by-size dabei.

Es gibt sogar (eines der frühen Höhepunkte der Theorie der Datenstrukturen):

m-1 Union's, m Find's

mit Union-by-size und Wegkompression in Zeit

$$O(m + (n+m) \cdot \log^*(n))$$

bei anfangs $P = \{\{1\}, \{2\}, \ldots, \{n\}\}$

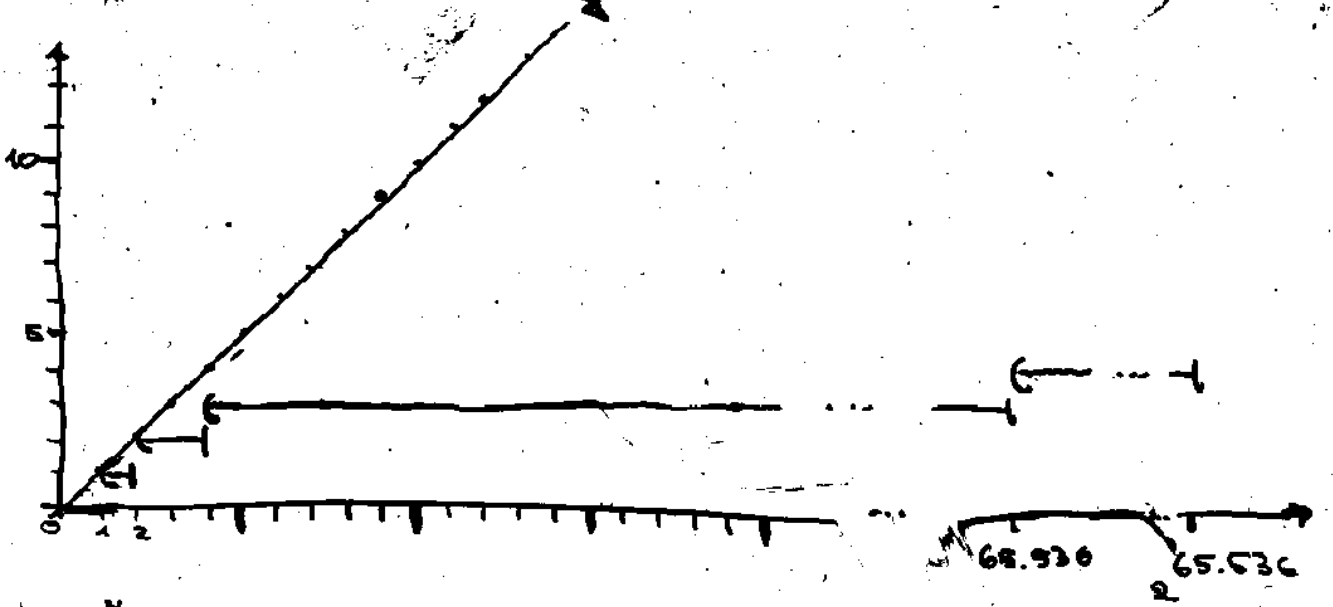
Falls $m \geq \Omega(n)$ ist das

$$O(m + m \cdot \log^*(n)).$$

(Beachten sie die Ablängigkeit der Konstanten: O-Konstante hängt von $\Omega$-Konstante ab!)

Wer ist $\log^*(n)$?

## 7.1 E FUNKTION $\log^*$




$\log^* 1 = 0$

$\log^* 2 = 1$ „FAST" KONSTANT.

$\log^* 3 = 2$

$\log^* 4 = 1 + \log^* 2 = 2$

$\log^* 5 = 3$

$\log^* 8 = \log^* 3 + 1 = 3$

$\log^* 16 = \log^* 4 + 1 = 3$

$\log^* 2^{16} = 4$

$\log^* (2^{16}) = 5$ $\quad 2^{16} = 65.536$

$\log^* n = \min\{s \mid \log_2^{(s)}(n) \leq 1\}$

$\log^* 2^{2^{2^{2^{(2^9)}}}} = 4$

$\log_2^{(s)}(n) = \log(\log(\ldots(\log(n))\ldots))$

s-MAL.

Kruskal bekommt dann eine Zeit von

$$O(|V| + |E| \cdot \log^* |V|)$$

Fast $O(|V| + |E|)$ bei vorsortierten Kanten natürlich nur. $\Omega(|V| + |E|)$ in jedem Falle.