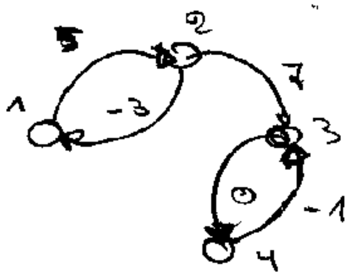
8. Kürzeste Wege

Hier sind alle Graphen gerichtet und gewichtet, d.h. wir haben eine Kostenfunktion $K: E \to \mathbb{R}$ dabei.

Also etwa



$K(1,2)=5$, $K(2,1)=-3$,

$K(2,3)=7$, $K(3,4)=0$

Ist $W=(v_0, v_1, \dots, v_k)$ irgendein Weg im Graphen, so

$$K(W) = K(v_0,v_1) + K(v_1,v_2) + \dots + K(v_{k-1},v_k)$$

die Kosten von W. $K(v_0)=0$.

Betrachten uns die Knoten 3 und 4, so gilt

$K(3,4) = 0, \quad K(4,3) = -1$

und

$K(3,4,3,4) = -1, \quad K(3,4,3,4,3,4) = -2, \dots$

Negative Kreise erlauben beliebig kurze Wege. Wir beschränken uns auf einfache Wege, d.h. noch einmal, daß alle Knoten verschieden sind.

<u>Definition</u> (Distanz) $\quad Dist(v,v) = 0$

Für $u,v \in V$ ist

$$Dist(u,v) = \min \{ K(w) \mid w = (v_0 = u, v_1, \dots, v_k = v) \text{ ist einfacher Weg von } u \text{ nach } v \},$$

sofern es einen Weg $u \to v$ gibt.

$Dist(u,v) = \infty$, wenn es keinen Weg $u \to v$ gibt. $\square$

Nun geht es jetzt darum, kürzeste Wege zu finden. Dazu machen wir zunächst folgende Beobachtung:

Ist

$$u=v_0 \to v_1 \to v_2 \to \dots \to v_k=v$$

ein k. W von u nach v, so sind alle Wege oben

$$v_0 \to v_i$$

auch kürzeste Wege. Deshalb ist es sinnvoll das single source shortest path Problem zu betrachten, also alle k. W's von einem Ausgangspunkt zu suchen. Ist s unser Ausgangspunkt so bietet es sich an eine Kante minimaler Kosten von s aus zu betrachten:

$s \longrightarrow v$

Gibt uns diese Kante einen k. W. von s nach v? Im allgemeinen nicht! Nur unter der Einschränkung, daß die Kostenfunktion

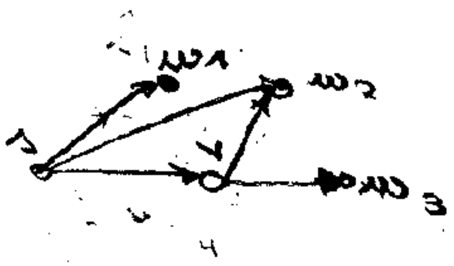
$$c : E \to \mathbb{R}^{\geq 0}$$

ist, also keine Kosten < 0 erlaubt. Diese Bedingung treffen wir zunächst einmal bis auf weiteres.

Also, warum ist der Weg $s \longrightarrow v$ oben ein k. W.? Jeder andere Weg von s aus hat doch eine erste Kante $s \longrightarrow w$ mindestens die Kosten $c(s,v)$. Der greedy Ansatz tut es am Anfang.

Das gilt bei negativen Kosten so nicht!

Wie bekommen wir einen weiteren k.W. von s aus?

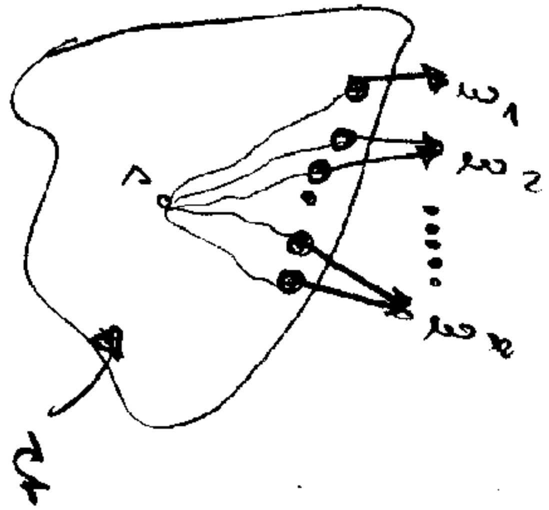


Wir schauen uns die zu s, v adjazenten Knoten an. Oben w_1, w_2, w_3. Wir ermitteln

- für w_1 $\ell(s,w_1)$
- für w_2 $\operatorname{Min}\{\ell(s,w_2);\ \ell(s,v)+\ell(v,w_2)\}$
- für w_3 $\ell(s,v)+\ell(v,w_3)$.

Ein minimales dieser Werte gibt uns einen weiteren k.W. Ist etwa $\ell(s,v)+\ell(v,w_2)$ minimal, dann gibt es keinen kürzeren Weg $\hat{0} \longrightarrow v_{02}$. Ein solcher Weg müßte ja die Menge 1,4 irgendwann verlassen. Dazu müssen aber die eingezeichneten Wege genommen werden und der Weg zu v_2 wird höchstens länger (Wieder wichtig: Kosten ≥ 0).

So geht es allgemein weiter:
Ist S eine Menge von Knoten, zu denen ein k. W. von s aus gefunden ist, so betrachten wir alle Knoten v: adjazent zu S, aber nicht in S. . .



Für jeden Knoten w_i berechnen wir die minimalen Kosten eines Weges der Art

$$\underbrace{s = s_0 \to s_1 \to s_2 \to \dots \to s_k}_{\subseteq Z} \to \underbrace{w_i}_{\notin Z}$$

Für ein w_i werden diese Kosten minimal und wir haben einen k. W. $s \to w_i$. Wie vorher sieht man, daß es überhaupt keinen kürzeren Weg $s \to w_i$ gibt.

Das Prinzip: Nimm immer den nächstkürzesten Weg von s ausgehend; implementiert →

Algorithmus (Dijkstra 1959)

Eingabe: $G=(V,E)$, $w: E \to \mathbb{R}^{\geq 0}$ (!),
$|V| = n$, $s \in V$

Ausgabe: array $D[1,\dots,n]$ of real mit
$D[v] = Dist(s,v)$ für $v \in V$.

Datenstrukturen

S = Mge. der Knoten, zu denen kz. W. gefunden ist

$Q = V \setminus S$

1. $D[s] := 0$; $S = \{s\}$, $Q = V \setminus \{s\}$
2. for $i = 1$ to $n-1$ { // Suchen noch $n-1$ kz.W.'s
3. $H := \{\{v,w\} \mid v \in S, w \in Q\}$
4. for each $w \in Q$ adjazent zu S {
5. $D[w] = \min\{D[v] + w(v,w) \mid (v,w) \in H\}$ }
6. $w :=$ ein $w \in Q$ mit $D[w]$ minimal;
7. $S := S \dot\cup \{w\}$, $Q = Q \setminus \{w\}$ }

(8.2)

Korrektheit mit Invariante: Für alle $u \in S_e$ ist $D_e[u]$ = Kosten eines k.W.
Das Argument geht wie oben bereits vorgeführt.

Zwecks Merken der Wege das array $\pi[1..n]$ of $\{1,\dots,n\}$ mit

$$\pi[v] = u \iff \text{Ein k.W. } 1 \to v \text{ ist } (1, \dots, u, v).$$

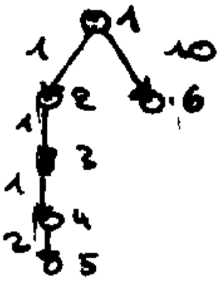
π kann leicht in G. mitgeführt werden.
Es ist das Vaterarray des kürzesten Wege Baumes mit Wurzel 1.

Diese Seite mußte aus rechtlichen Gründen entfernt werden!

8M

S	W	D[1]	D[2]	D[3]	D[4]	D[5]
1	/	0	10	∞	30	100
{1,2}	2	0	10	60	30	100
{1,2,4}	4	0	10	50	30	90
{1,2,4,3}	3	0	10	50	30	60
{1,2,4,3,5}	5	0	10	50	30	60

DIE WEGE WERDEN DER LÄNGE NACH GEFUNDEN:



KÜRZESTER WEGE BAUM:

$\pi[6]=1$, $\pi[2]=1$,
$\pi[3]=2$, $\pi[4]=3$
$\pi[5]=4$.

S

{1}
{1,2}
{1,2,3}
{1,2,3,4}
{1,2,3,4,5}
{1,2,3,4,5,6}.

Laufzeit bei direkter Implementierung etwa $O(|V|\cdot|E|)$, da $O(|V|)$ Schleifendurchläufe, jedesmal die Kanten durchsuchen, ob sie zu W gehören, Q und S als boolsche arrays mit $Q[v] = \text{true} \Leftrightarrow v \in Q$ implementieren.

Die Datenstruktur des dictionary (Wörterbuch) speichert eine Menge von Elementen und unterstützt die Operationen

- Find(u) = Finden des Elements u innerhalb der Struktur
- Insert(u) = Einfügen
- Delete(u) = Löschen.

Ist die Grundmenge nicht als Indexmenge geeignet: Hashing oder Suchbäume

Ist Q als dictionary implementiert, jede Operation $O(1)$.

Ziel: Bessere Implementierung.

Für welche $v \in Q$ kann sich $D[v]$ im 5. nur ändern? Nur für diejenigen, die adjazent zu dem w des vorherigen Laufs sind. Dann sieht es etwa so aus:

Algorithmus (Dijkstra ohne mehrfache Berechnung desselben $D[w]$)

1. $D[s] := 0$, $D[v] := \infty$ für $v \in V \setminus \{s\}$,
   $Q := V \setminus \{s\}$, $S = \{s\}$.
2. for $i = 1$ to $n-1$
3. $\quad w :=$ ein $w \in Q$ mit $D[w]$ minimal,
4. $\quad S := S \cup \{w\}$; $Q = Q \setminus \{w\}$;
5. $\quad$ for each $v \in Adj[w]$ { // $D[v]$ anpassen
   $\quad$ // für v adjazent zu w
6. $\quad\quad$ if $D[w] + K(w,v) < D[v]$ {
   $\quad\quad\quad D[v] = D[w] + K(w,v)$ } }

Korrektheit mit zusätzlicher Invariante:

Für $w \in Q_\ell$ ist

$D_\ell[w]$ = minimale Kosten eines Weges der Art

$$\underbrace{s=s_0 \to s_1 \to s_2 \to \dots \to s_k}_{\subseteq S} \to \underbrace{w}_{\in Q}$$

Man kann auch leicht zeigen:

Für alle $v \in S_\ell$ ist $D_\ell[v] \leq D_\ell[w_\ell]$,

w_ℓ = das Minimum der ℓ'ten Runde.

Damit ändert 6. nichts mehr an $D.[v]$

für $v \in S_\ell$.

Laufzeit:

3. insgesamt $n-1$-mal Minimum finden.
4. " " $n-1$-mal " löschen.
5.,6. insgesamt $\Theta(|E|)$, da dort jede Adjazenzliste nur einmal durchlaufen wird.

7. " " "

Mit Q als boolsches array:

3. $O(n^2)$ insgesamt. Das Finden eines neuen Minimums nach Löschen dauert $O(n)$.

4. $O(n)$ insgesamt. 1. Löschen $O(1)$.

5.,6. Bleibt bei $O(|E|)$.

Also alles in allem $O(n^2)$.

Also auf Q benötigen nur die klassischen Operationen der priority queue: Also Q als heap, Schlüsselwert aus D.

3. $O(n)$ insgesamt.

4. $O(n \cdot \log n)$ insgesamt

5.,6. $|E|$-mal heap anpassen. mit Decreasekey(v, k) (vgl. Prim) $O(|E| \cdot \log n)$. Zum Finden Index mitführen!

Also: Q als boolsches array: $O(n^2)$.

Zeit fällt beim Finden des Minima an.

Q als heap mit Index: $O(|E| \log n)$

Zeit fällt beim DecreaseKey(v,t) an.

Ist $|E| \log n \geq n^2$, also Graph sehr dicht, dann array besser. (Vgl. Prim).

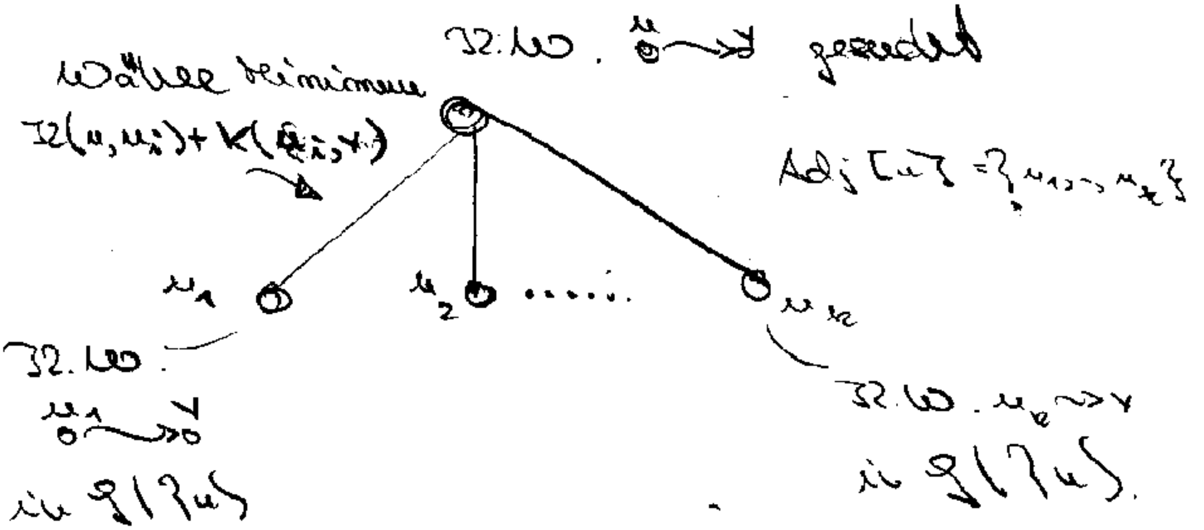
Ab jetzt betrachten wir wieder eine beliebige Kostenfunktion

$\ell: E \to \mathbb{R}$.

Der greedy Ansatz wie bei Dijkstra geht nicht mehr. Z.B. $u \rightsquigarrow v$ durch Durchprobieren. Zeit $(n-2)! \geq 2^{\Omega(n \log n)} \gg 2^n$!

Es werden i.a. viele Permutationen generiert, die gar keinen Weg ergeben. -

Das vermeidet backtracking:



Korrekt, da Teilwege kürzester Wege wieder kürzeste Wege sind und wir mit kürzesten Wegen immer nur einfache Wege meinen.

8.18

Das ganze leicht durch Rekursion zu machen.

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

SZW(W; u, v) { // $u, v \in W$,
// W = noch b... // betrachtete Knotenmenge

1. if $u=v$
return 0;

2. $l := \infty$;

3. for each $w \in Adj[u] \cap W$: {

$l' := SZW(W \setminus \{u\}, w, v)$;

$l' := w(u,w) + l'$;

if $l' < l$:

$l := l'$; // Hier $\pi[w] := u$
// gibt k. Wege selbst.

}

4. return l // Ausgabe ∞, wenn
// $Adj[u] \cap W = \emptyset$.

Aufruf: SZW(V, a, b). Korrektheit: Induktion über $|W|$.

Etwas einfacher ist es, alle einfachen Wege $\overset{u}{\circ} \rightsquigarrow \overset{v}{\circ}$ systematisch zu erzeugen und den Längenvergleich nur am Ende zu machen.

Datenstruktur: L

L = Liste von Knoten als Keller implementiert

Intention: L enthält den Weg vom Startknoten a bis zum aktuellen Knoten.

g = Liste von Knoten. Der aktuelle k. W. $\overset{a}{\circ} \rightsquigarrow \overset{b}{\circ}$

L, g globale arrays.

```
SZW(W, u, v); {
 1. if u = v {
        if SZ(L) < SZ(g); g := L   // SZ(L), SZ(g) =
    }                              // Kosten von L, g
 2. for each w ∈ Adj[u] ∩ W; {
        w auf Keller L tun;
        SZW(W\{u}, w, v);
        w vom Keller L löschen
 }} Aufruf mit SZ(L) = ∞, K(g) = ∞, L = (a), SZW(V, a, b).
```

Korrektheit mit der Aussage: Beim Aufruf von $RW(W, u, v)$ ist

$L = w \ldots a$.

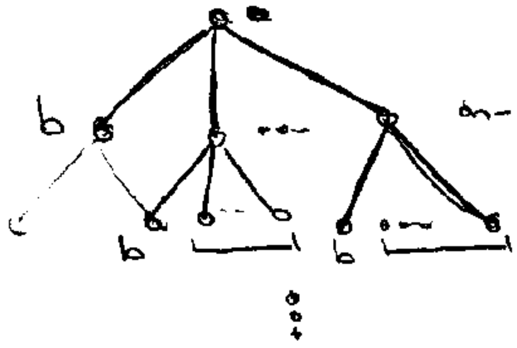
Am Ende des Aufrufs $RW(W, u, v)$ enthält G einen k. W. der Art

$G = b \ldots L$ ← Das L von oben.

Das ganze induktiv über $|W|$.

Laufzeit des rekursiven Verfahrens?

Enthält der Graph alle $n(n-1)$ Kanten, so Aufrufbaum



$n-1$ Kinder von a

Je $n-2$ Kinder $\neq b$

Also $\geq (n-2)(n-3) \cdots 3 \cdot 2 \cdot 1 = 2^{\Omega(n \cdot \log n)}$.

Wieviele verschiedene Aufrufe haben wir höchstens? Also Aufrufe $sew(w, c, d)$?

- $w \in Y$ $\leq 2^n$ Möglichkeiten.
- c, d $\leq n$ " ,

da Endpunkt immer gleich.

Also $n \cdot 2^n = 2^{\log n} \cdot 2^n = 2^{O(n)}$.

Also bei $2^{\Omega(n \cdot \log n)}$ Aufrufen wie oben viele **doppelt**. Es ist ja

$$\frac{2^{O(n)}}{2^{\Omega(n \cdot \log n)}} = \frac{1}{2^{\Omega(\log n)}} = \frac{1}{\underset{\underset{n^{\Omega(1)}}{\Uparrow}}{n^{\varepsilon}}} \to 0$$

für ein $\varepsilon > 0$, konstant.

Also: Vermeiden der doppelten Berechnung durch Merken (Tabellieren).

Dann das

$$\text{array } T[\underbrace{1 \ldots 2^n}_{\in \{0,1\}^n}][1 \ldots n] \text{ of int}$$

mit der Interpretation: Für $W \in Y$ mit $b, v \in W$ ist

$T[W, v]$ = Länge eines k. W. von v nach b nur durch W.

Füllen $T[W, v]$ für $|W| = 1, 2, \ldots$

1. $|W| = 1$, dann $v = b \in W$

$T[\{b\}, b] = 0$

2. $|W| = 2$, dann

$T[W, b] = 0$

$T[W, v] = D(v, b)$, wenn $v \neq b$

z. $|W| = 3$.

$T[W, b] = 0$

$T[W, v]$ = ergibt sich aus

$l := \infty$, $W' := W \setminus \{v\}$

for each $u \in Adj[v] \cap W$ {

$l' := w(v, u) + T[W', u]$

if $l' \leq l$;

$l := l'$

}

$T[W, v] := l$

Also das heißt

$$T[W, v] := \min \{ w(v,u) + T[W', u] \mid W' = W \setminus \{v\}, u \in W \}$$

$T[W, v] = \infty$, wenn keine Kante

$v \rightarrow u$ mit $u \in W$.

Weg mitführen durch

$P[W,v] = u$, u vom Minimum.

ermöglicht leichtere Ermittlung des Weges.

Also der Eintrag $T[W,v]$ wird so gesetzt:

```
setze(W,v) {          // stets b,v ∈ W

 1. if v = b; {
          T[W,v] = 0; P[W,v] = b; return
    }
    T[W,v] := ∞; W' = W\{v}   // Hier |W| ≥ 2
 2. for each u ∈ Adj[v] ∩ W {
          l := c(v,u) + T(W',b);
          if l < T[W,v]; {
               T[W,v] := l;
               P[W,v] := u
    }
```

Dann insgesamt

$SZW(V, a, b)$ {

1. for $i = 1$ to n {
2. for each $W \subseteq V$, $b \in W$, $|W| = i$ {
3. for each $v \in W$ {
4. Setze(W, v)

}

} } }

Dann $T[V, a]$ enthält das Ergebnis

Weg ist

$a_0 = a$, $a_1 = P[V, a]$, $a_2 = P[V \setminus \{a\}, a_1]$

$\ldots$ $a_i = P[V \setminus \{a_0, \ldots, a_{i-2}\}, a_{i-1}]$, $\ldots$

Invariante für 1. Nach l'tem Lauf ist $T_l[W, v]$ korrekt für alle W mit $|W| \le l$. Laufzeit: $O(n^2 \cdot 2^n)$

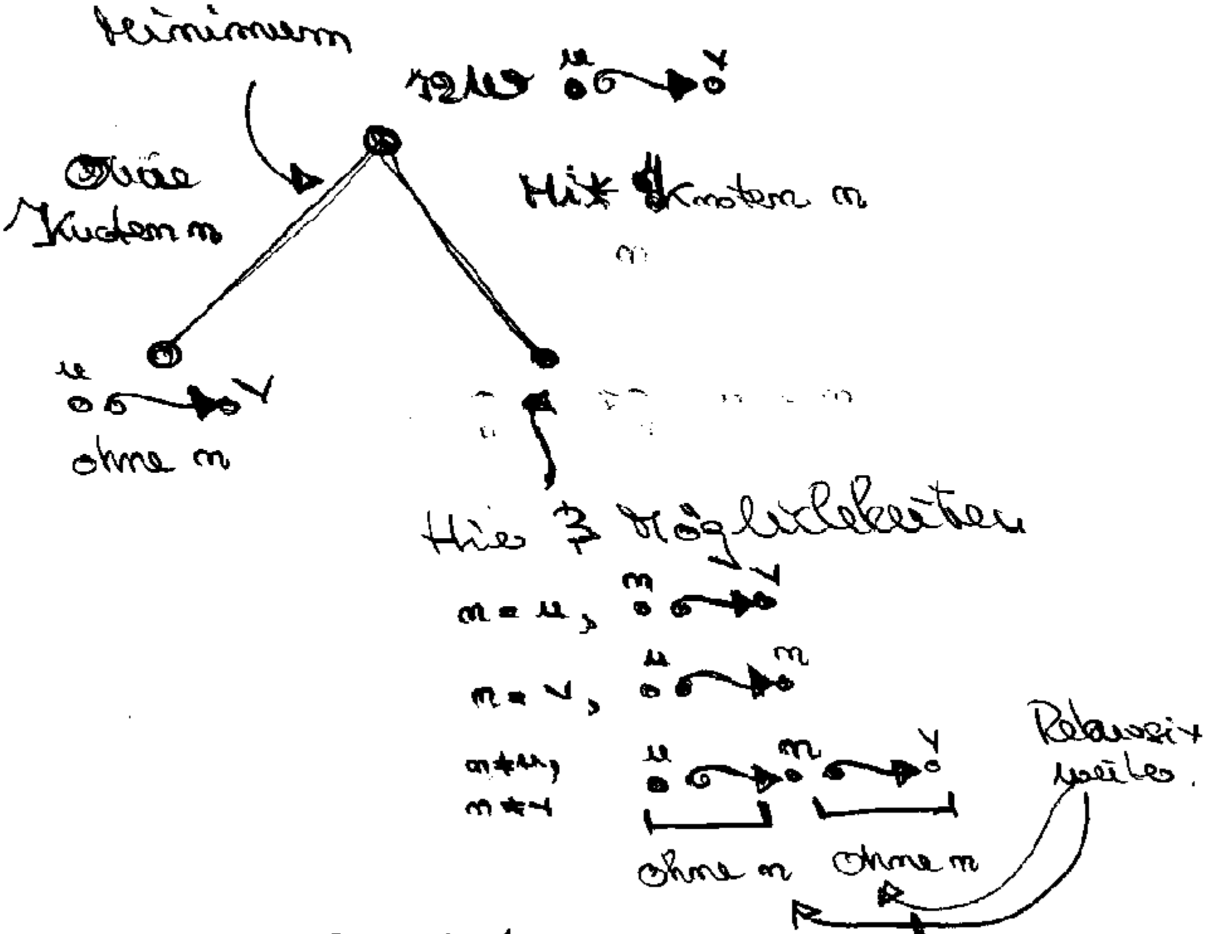
da ein Lauf von Setze(w,v) in $O(n)$ möglich ist.

Das hier verwendete Prinzip:
Mehr rekursive Aufrufe als
Möglichkeiten ⇒ Tabellieren
der Aufrufe heißt:

Dynamisches Programmieren.

Aufstellen einer
Tabelle 1950'er

Wir beginnen einmal mit einer anderen Fallunterscheidung beim backtracking:



Korrektheit: Ist $u, v \neq m$ und ist eine k.W. $u \rightsquigarrow v$ ohne m, dann wird dieser nach Ind.-Vor. links gefunden. Enthalten jetzt jedes k.W. $\overset{u}{\circ} \rightsquigarrow \overset{v}{\circ}$ den

Knoten m. Wird ein solcher

unbedingt rechts gefunden?

Nur dann wenn die k.W.'s

$$\overset{u}{\circ} \rightsquigarrow \overset{m}{\circ} \quad \overset{m}{\circ} \rightsquigarrow \overset{v}{\circ}$$

keinen weiteren gemeinsamen

Knoten haben. Wenn sie einen

gemeinsamen Knoten w haben, so

$$\overset{u}{\circ} \rightsquigarrow \overset{w}{\circ} \rightsquigarrow \overset{m}{\circ} \rightsquigarrow \overset{w}{\circ} \rightsquigarrow \overset{v}{\circ}$$

Da dieser Weg kürzer ist als

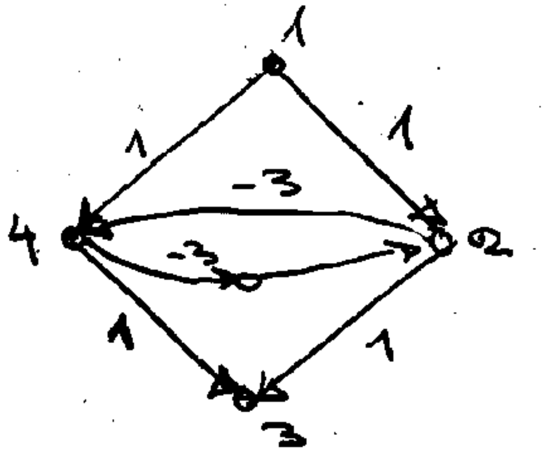
jeder Weg durch m, muß

$$\overset{w}{\circ} \rightsquigarrow \overset{m}{\circ} \rightsquigarrow \overset{w}{\circ}$$

ein Kreis der Länge $\lneq 0$ sein.

Note 4 erkennbar Betrachten nur also jetzt $\ell: E \to \mathbb{R}$,

aber so, daß keine Kreise < 0 existieren.



KZ. W. $\overset{1}{\circ} \longrightarrow \overset{3}{\circ}$ OHNE 4

KOSTEN 2

KZ. W. $\overset{1}{\circ} \longrightarrow \overset{2}{\circ} \longrightarrow \overset{4}{\circ}$ KOSTEN -2

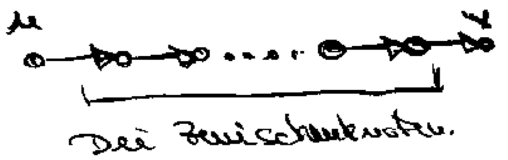
KZ. W. $\overset{4}{\circ} \longrightarrow \overset{2}{\circ} \longrightarrow \overset{3}{\circ}$ " -2

UND

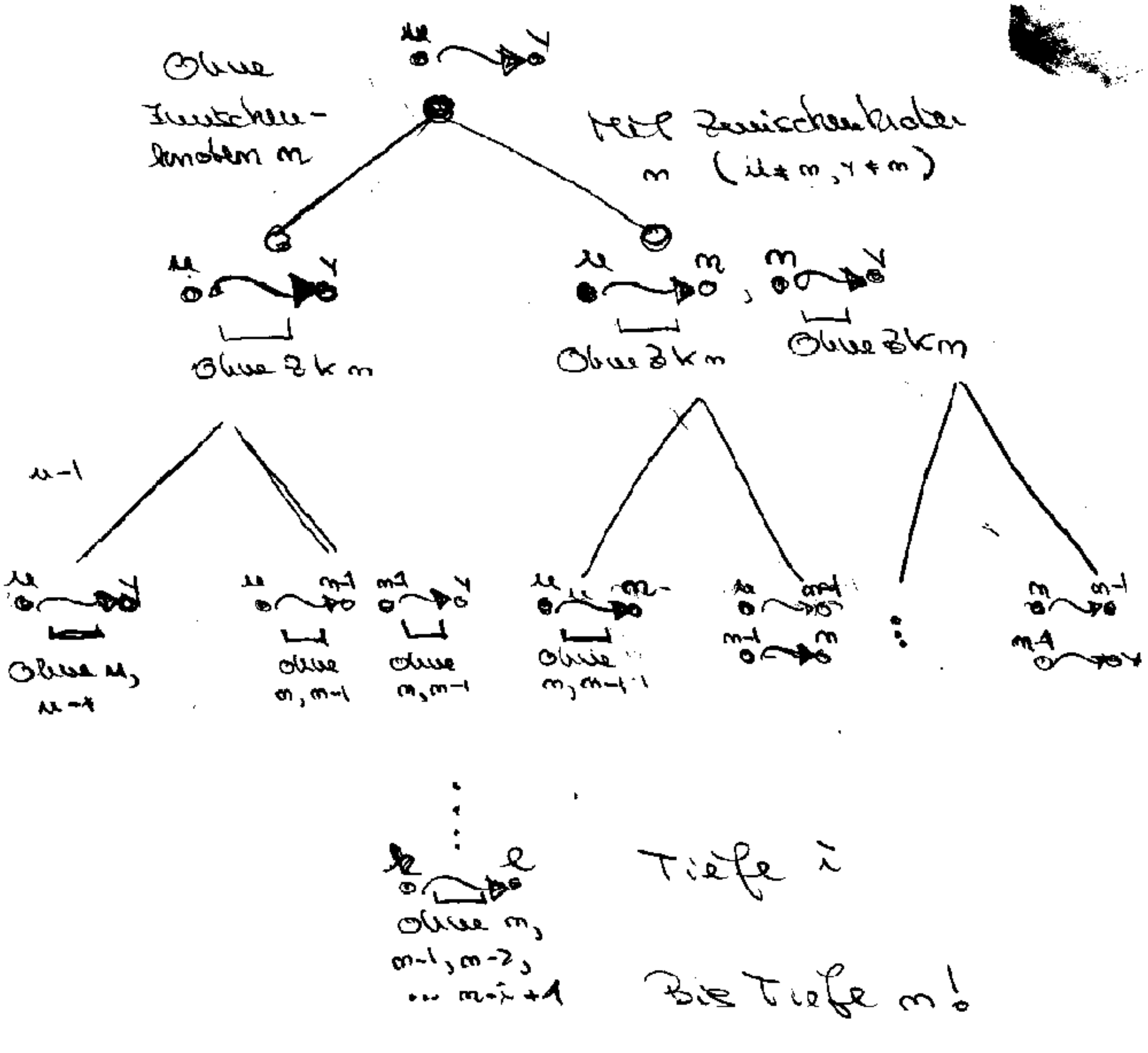
(2, 4, 0, 2)

KREIS DER KOSTEN -6.

Es ist günstiger, die Fallunterscheidung nach den ersten Zwischenknoten zu machen:



Also Backtracking



Also rekursive Formeln: $SPW(u,v,i)$

kü. k. W. $\overset{u}{\circ} \to \overset{v}{\circ}$ ohne Zwischenknoten

$n, n-1, n-2, \ldots, n-i+1$. Kürzester Weg

gibt sich durch $SPW(u,v,0)$. Rekursions-

anfang $SPW(u,v,n)$ gibt Kante

$\overset{u}{\circ} \longrightarrow \overset{v}{\circ}$. Wieviele verschiedene Aufrufe?

$O(n^3)$! Also dynamisches

Programmieren:

"$T[u,v,i]$ = k. W $\overset{u}{\circ} \longrightarrow \overset{v}{\circ}$ wobei

Zwischenknoten $\subseteq \{1,\ldots,i\}$.

also nicht unter

$i+1, \ldots, n$.

0. $T[u,v,0] = \Omega(u,v)$ für alle u,v!

1. $T[u,v,1] = \min\{T[u,1]+T[1,v], T[u,v,0]\}$

für alle u,v

$\vdots$

n. $T[u,v,n] = \min\{T[u,n,n-1]+T[n,v,$

8.32

m. $T[u,v,m] :=$

$$\min\{T[u,m,m-1] + T[m,v,m-1],\ T[u,v,m-1]\}$$

für alle u,v

Alle Paare shortest path.

<u>Algorithmus</u> (Floyd Warshall)

G ohne Kreise < 0

$V = \{1, \dots, n\}$

1. $T[u,u] := 0$, $T[u,v] = w(u,v)$ für $(u,v) \in E$

   $T[u,v] := \infty$ für $u \neq v, (u,v) \notin E$.

2. for $i = 1, 2, \dots, n$ {

   for each $(u,v) \in V$ { // Alle // geordneten // Paare.

   $T[u,v] := \min\{T[u,v],\ T[u,i] + T[i,v]\}$

   }

S.33

Invariante: Nach l'tem Lauf der Schleife in 1. enthält

$T_l[u,v]$ = Länge eines k. W. $u \rightsquigarrow v$
mit Zwischenknoten $\subseteq \{1,\dots,l\}$

Wichtig: keine negativen Kreise,
denn in $l+1$'ter Runde

$$T_l[u,l+1] + T_l[l+1,v] < T_l[u,v]$$

dann ist der Weg hinter
$T_l[u,l+1] + T_l[l+1,v]$ ein einfacher!

$$u \longrightarrow l+1 \longrightarrow v$$

kürzer als k. W. $u \rightsquigarrow v$ ohne $l+1$,

dann ~~kann~~ nicht

$$u \rightsquigarrow w \rightsquigarrow l+1 \rightsquigarrow w \rightsquigarrow v,$$

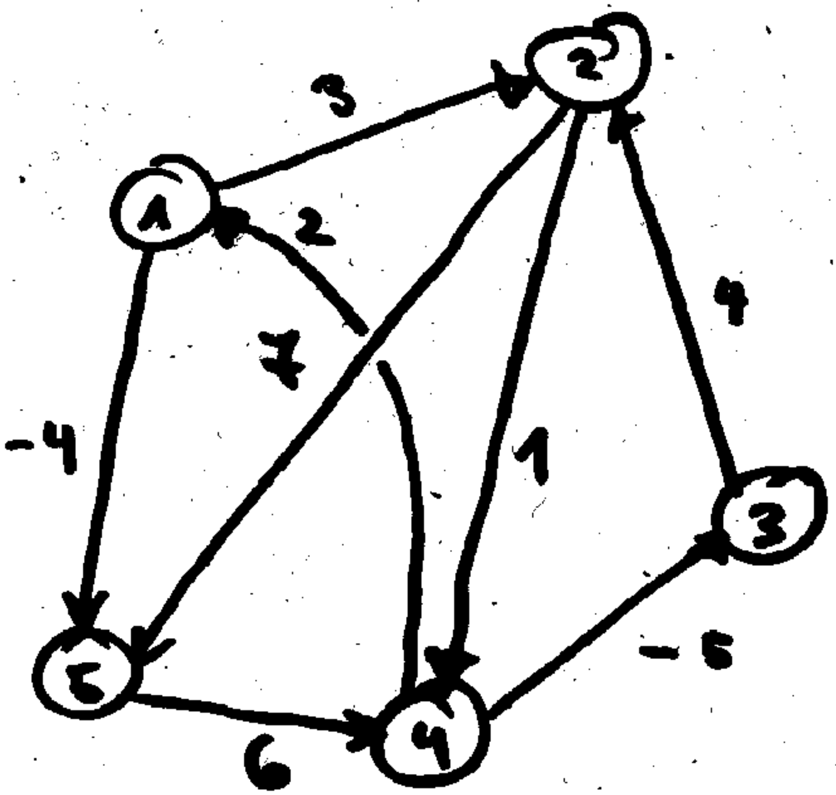
da sonst $l(w \rightsquigarrow l+1 \rightsquigarrow w) < 0$ sein müßte.

Erkennen negativer Kreise. Kann man sich, also bei beliebiger Kostenfunktion, daß Floyd Warshall die Kosten eines Weges von u nach v in $T[u,v]$ liefert. Dann

$$T[u,u] < 0 \iff G \text{ hat Kreis } \lneq 0.$$

Laufzeit: $O(n^3)$.

Vergleiche Dijkstra $O(|E| \log |V|)$ oder $O(|V|^2)$.



A

AM ANFANG

$$
\begin{matrix}
0 & 3 & \infty & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & \infty & -5 & 0 & \infty \\
\infty & \infty & \infty & 6 & 0
\end{matrix}
$$

A BEVOR k AUF 2 GEHT:

0	3	∞	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	∞
∞	∞	∞	6	0

ES IST $A[i,j] = \mathrm{MIN}\{A[i,j],$

$A[i,1] + A[1,j]\}$

DIREKTE WEGE.

WEGE MIT ZWISCHEN-

KNOTEN 1.

IN A BEVOR k AUF 3 GEHT:

DIREKTE WEGE.

\+ WEGE MIT ZW 1.

\+ WEGE MIT ZW 1, 2.

NICHT: MIT 2 ZWISCHEN-

KNOTEN!