Mitschrift zu Graphentheoretische Konzepte und Algorithmen

Christoph Häberlein

January 13, 2009

Contents

1 Kapitel 1: Einleitung

1.1 Beispiele

Routenplanung

- Problem mit Stadtplan, Einbahnstraßen, usw.

Postbote/Müllabfuhr

- alle Straßen mindestens einmal durchlaufen
- zusätzliche Umwege vermeiden (Eulersche Graphen)

Handelsreisender (Traveling Salesman Problem)

- alle Kunden mindestens einmal anfahren
- möglichst kleine Gesamtlänge

Standortplanung

- Eröffnung mehrerer Filialen in einer Region

Verkehrsplanung/Stau

- Fluss, Verkehrsstrom

Präzedenzgraphen

- kreisfrei! sonst tritt ein Deadlock auf.

Verträglichkeitsgraphen

- Bsp: Frequenzplanung bei Funksystemen/Mobilfunk
- Benachbarte Sender brauchen unterschiedliche Frequenz (Knotenfärbung)
- gesucht: Minimalzahl der benötigten Farben

Weitere Relationen

n Personen, $aRb := a\,kennt\,b \wedge a\,kennt\,b$

- symmetrisch $aRb \Leftrightarrow bRa$
- nicht transitiv $aRb \wedge bRc \not\Rightarrow aRc$

Bsp: $n = 5$. Gibt es eine Teilmenge von 3 Leuten, die sich parrweise kennen oder sich paarweise nicht kennen?
(Komplementgraph: hat genau dort eine Kante, wo der ursprüngliche Graph keine Kante hatte!)
Was ist die kleinste Zahl von Personen, die diese Bedingung erfüllen?

Behauptung:

In jeder Gruppe von 6 Personen gibt es eine Teilgruppe, die die Bedingung erfüllen.

Beweis:

Sei Graph mit 6 Personen gegeben. Betrachte eine beliebige Person v.

- 1. Fall: v kennt mindestens drei andere Personen
  - Betrachte beliebige 3 der mit v bekannten Personen
    - a) diese drei sind paarweise nicht miteinander bekannt. Bedingung erfüllt.
    - b) mindestens ein Paar kennt sich schon. Zusammen mit v bilden diese eine 3-Clique. Bedingung erfüllt.
- 2. Fall: v kennt höchstens 2 andere Personen, d.h. v kennt mindestens drei andere Personen nicht. Die Behauptung folgt aus Fall 1, wenn wir zum Komplementgraphen übergehen.

□.
Ramsey-Zahl $r(3,3) = 6$.

1.2 Grundbegriffe

1.2.1 Definition: gerichteter Graph

Ein gerichteter Graph ist gegeben durch $G = (V, R)$ mit
$V \neq \emptyset$: Menge der Knoten/Ecken (Vertex, Node)
und
$R \subseteq \{(u,\, v)\,|u, v \in V, , u \neq v\}$. Ein Pfeil $r = (u, v)$ hat die Anfangsecke $\alpha(r) = u$ und die Endecke $\omega(r) = v$.

Bemerkung: parallele Pfeile und Schlingen (Selbstverweis) sind nach unserer Definition nicht erlaubt.

Bsp: $V = \mathbb{N}_0$**.** $R = \{(i,\, 2i)\,|i > 0\}$ ist ein unendlicher Graph.

1.2.2 Vereinbarung:

Alle Graphen sind endlich, d.h. V ist eine endliche Menge.
$v \in V$, $r \in R$ heißen inzident, falls $v = \alpha(r) \vee v = \omega(r)$.
r_1, r_2 heißen inzident, falls $\exists v \in V$ inzident mit beiden r_1, r_2.
$u, v \in V$ heißen adjazent (benachbart), falls $\exists r \in R$ inzident mit beiden u, v.
$\delta^+(v) = \{r \in R|\overline{\alpha(r) = v}\}$ der von v ausgehende Pfeilbüschel.
$\delta^-(v) = \{r \in R|\omega(r) = v\}$ der in v einmündende Pfeilbüschel.
$N^+(v) := \{u \in V|\exists(v, u) \in R\}$ Menge der Nachfolger von v.
$N^-(v) := \{u \in V|\exists(u, v) \in R\}$ Menge der Vorgänger von v.
$g^+(v) := |N^+(v)|$ Außengrad (out degree).
$g^-(v) := |N^-(v)|$ Innengrad (in degree).
$g(v) = g^+(v) + g^-(v)$ Grad der Ecke v.

1.2.3 Definition: Teilgraph/Obergraph

Sei $G = (V, R)$ ein Graph. Für alle $V' \subseteq V$ und $R' \subseteq R$, für die (V', R') ein Graph ist (d.h. keine inder Luft hängenden Pfeile), heißt $G' = (V', R')$ Teilgraph von G und G heißt Obergraph von G'.

1.2.4 Definition: Induzierter Subgraph

Sei $G = (V, R)$ ein Graph. Sei $V' \subseteq V$. Setze $R' = \{r \in R|\alpha(r) \in V' \wedge \omega(r) \in V'\}$, dann heißt (V', R') der durch V induzierte Subgraph. Notiert $G\,[V']$. Ferner $G - v := [V \backslash \{v\}]$.

1.2.5 Definition: ungerichteter Graph

$G = (V, E)$ mit
$V \neq \emptyset$ Ecken (endlich)
und
$E \subseteq \{\{u, v\} | u, v \in V \wedge u \neq v\}$ Menge der Kanten (edges). Notieren für Kanten dennoch (u, v) statt $\{u, v\}$.

1.2.6 Lemma

Sei $G = (V, E)$ ein ungerichteter Graph. Die Anzahl der Ecken mit einem ungeraden Grad ist selbst eine gerade Zahl.

Beweis: partitioniere $V = V_g \cup V_u$, sodass $V_g = \{v \in V | g(v)\, gerade\}$ und $V_u O \{v \in V | g(v)\, ungerade\}$.

$$\begin{aligned} 2|E| &= \sum_{v \in V} g(v) = \\ &= \sum_{v \in V_g} g(v) + \sum_{v \in V_u} g(v) = \\ &= \underbrace{\sum_{v \in V_g} g(v)}_{gerade} + \underbrace{\sum_{v \in V_u} g(v - 1 + |V_2|}_{gerade} \\ \Rightarrow & \quad |V_u|\, ist\, gerade \end{aligned}$$

1.2.7 Definition: invers

Sei $G = (V, R)$. Für $r = (u, v)$ heißt $r^{-1} = (v, u)$ der inverse Pfeil. Setze $R^{-1} := \left\{r^{-1} | r \in R\right\}$, dann heißt $G^{-1} = \left(V, R^{-1}\right)$ der inverse Graph und $G^{sym} := \left(V, R \cup R^{-1}\right)$ die symmetrische Hülle mit $E := \left\{\{u, v\} | (u, v) \in R \cup R^{-1}\right\}$ ist $H := (V, E)$ der zu G gehöroge ungerichtete Graph. Falls $R \cap R^{-1} = \emptyset$(d.h. G enthält ein Paar von Antiparallelen), dann heißt G eine Orientierung von H.

1.2.8 Definition: vollständiger Graph

Für $n \in \mathbb{N}$ sei $V_n := \{v_1, ..., v_n\}$, $E_n := \{\{v_i, v_j\} | i \neq j\}$ heißt $K_n := (V_n, E_n)$ der vollständige Graph der Ordnung n. Für ein $G = (V_n, E)$ heißt $\overline{G} := (V_n, E_n \backslash E)$ der Komplementgraph zu G.
beobachtung: G' und $\overline{G'}$ sind "gleich" $\Rightarrow$ Isomorphie

1.2.9 Definition: Isomorphie

Zwei Graphen $G = (V, R)$ und $G' = (V', R')$ heißen isomorph, wenn es bijektive Abbildungen $\sigma : V \to V'$ und $\tau : R \to R'$ gibt, sodass die Inzidenzen erhalten bleiben, d.h.

$$\begin{aligned} \alpha'(\tau(r)) &= \sigma(\alpha(r)) \\ \omega'(\tau(r)) &= \sigma(\omega(r)) \end{aligned}$$

für alle $r \in R$.
Im Beispiel: G' und $\overline{G'}$ von oben sind isomorph.
Notation: $G \cong G'$ isomorph

1.3 Komplexität

$n := |V|$. $m := |E|$ oder $m := |R|$.

Speicherung von Graphen

Adjazenzmatrix: $n \times n$-Matrix. Der Eintrag

$$\alpha_{ij} := \begin{cases} 1, \text{ falls } (i, j) \in R \\ 0, \text{ sonst} \end{cases}$$

. Der Speicherbedarf für eine Adjazenzmatrix ist $\Theta\left(n^2\right)$. Problematisch für "dünn" besetzte Graphen.

Adjazenzliste: Eine Liste $(a_1, .., a_n)$. Jeder Eintrag a_i enthält eine weitere Liste des Pfeilbündels $\delta^+(i)$. Der Speicherbedarf hierfür ist $O(m+n)$. Die Liste kann verschieden realisiert sein (Hash, doppelt verkettet,...).
Die Laufzeiten werden bei Graphalgorithmen als Funktion $f(n)$ oder $f(n, m)$ angegeben.

2 Wege, Kreise, Zusammenhang

2.1 Wege

2.1.1 Definition: Weg

Eine Folge $p = (r_1, ..., r_k)$ (mit $k \geq 0$) von paarweise verschiedenen Pfeilen r_i heißt Weg (trail), wenn $\omega(r_i) = \alpha(r_{i+1})$ für alle $i = 1, ..., k-1$ gilt.
Anfang: $\alpha(p) := \alpha(r_1)$
Ende: $\omega(p) := \omega(r_k)$
Länge: k (Anzahl der Pfeile)
Der leere Weg $p = ()$ hat die Länge $k = 0$. Dem genügt $\alpha(p) = \omega(p)$.

2.1.2 Definition: Kreis

Ein Weg p mit $\alpha(p) = \omega(p)$ heißt Kreis.

2.1.3 Definition: Spur

Die Folge $(\alpha(r_1), \alpha(r_2), ..., \alpha(r_k), \omega(r_k))$ heißt Spur der besuchten Ecken.

2.1.4 Definition: elementar

Sind alle Ecken der Spur paarweise verschieden (mit der möglichen Ausnahme $\alpha(p) = \omega(p)$), so heißt der Weg p elementar (path).

2.1.5 Definition: Komposition

Für Wege p, q mit $\omega(p) = \alpha(q)$ heißt $p \circ q$ die Komposition von p und q (Pfeilfolgen aneinanderhängen).

Interessant:

Kürzeste Wege, d.h. bei vorgegebenem $s, t \in V$ ein s-t-Weg minimaler Pfeilzahl.

2.1.6 Lemma:

Seien $s, t \in V$ und p ein kürzester s-t-Weg. Dann ist jeder Teilweg p' von p selbst ein kürzester Weg (von $\alpha(p')$ nach $\omega(p')$).

Beweis:

Sei p' Teilweg von p, d.h. $p = p_1 \circ p' \circ p_2$. Wäre p' nicht kürzester Weg, dann wäre p'' kürzer und damit $p_1 \circ p'' \circ p_2$ ein Weg von s nach t, der kürzer als p wäre. Dies ist ein Widerspruch. □

2.1.7 Lemma:

Sei $G = (V, R)$. Wenn $g^-(v) > 0$ (oder $g^+(v) > 0$) für alle $v \in V$, dann besitzt G einen Kreis.

Beweis:

1. Wähle eine beliebige Ecke und setze $p := ()$.
2. Iteriere: wegen $g^-(v) > 0$ gibt es einen Pfeil r mit $\omega(r) = v$. Ersetze $p := r \circ p$ und setze $v := \alpha(r)$, sofern $\alpha(r)$ noch nicht in der Spur von p auftritt. Wiederhole die Iteration.
3. Da G endlich ist, bricht die Iteration ab, der gewünschte Kreis kann als Teilweg von p entnommen werden.

2.1.8 Definition: Weg im ungerichteten Graphen

vgl. Übung.

2.2 Kreisfreie Graphen

2.2.1 Definition: kreisfrei

Ein Graph heißt kreisfrei, wenn es keinen elementaren Weg der Länge > 0 enthält.

Beispiel: Präzedenzgraph

(Bild)

2.2.2 Definition: topologische Sortierung

Sei $G = (V, R)$. Eine bijektive Abbildung $\sigma : V \to \{1, ..., n\}$ heißt topologische Sortierung, falls $\sigma(\alpha(r) < \sigma(\omega(r))$ für alle $r \in R$.

2.2.3 Satz

Ein gerichteter Graph ist genau dann kreisfrei, wenn er eine topologische Sortierung zulässt.

Beweis:

" $\Rightarrow$ "
Induktion nach n.
I.A.: $n = 1$: ist klar.
Sei also G mit n Ecken gegeben. Es gibt ein $v \in G$ mit $g^+(v) = 0$(sonst Widerspruch zu obigen Lemmata). $G' := G - v$ ist nach I.V. topologisch sortierbar mit den Werten $\{1, ..., n-1\}$.
Setze diese topologische Sortierung fort mit $\sigma(v) := n$.

" $\Leftarrow$ "
Sei σ topologische Sortierung. Hätte G einen Kreis p, so betrachte die Pfeile $p = (r_1, ..., r_k)$. Es ist $\sigma(\alpha(r_1)) < ... < \sigma(\alpha(r_k))$. Widerspruch zu $\alpha(r_1 = \omega(r_k)$. $\square$

Bemerkung:

Der Algorithmus zur Bestimmung einer topologischen Sortierung ist in $O(m + n)$ möglich.

2.3 Zusammenhang

2.3.1 Definition: Erreichbarkeit

Eine Ecke w heißt erreichbar von v, falls es einen $v - w - Weg$ gibt.
Notation: $E(v) := \{w \in V | \text{w ist von v erreichbar}\}$. Insbesondere gilt $v \in E(v)$.

Algorithmus:

1. Setze $L = \{v\}$.
2. while $L \neq \emptyset$: entferne erstes Element $w \in L$ (und markiere es). Füge alle (unmarkierten) $w' \in \delta^+(w)$ ans Ende von L.

Die Laufzeit liegt in $O(m + n)$.
Korrektheit für "jede Ecke aus $E(v)$ wird auch markiert" mit vollständiger Induktion nach der Länge kürzester Wege.

2.3.2 Definition: stark zusammenhängend

Ist $w \in E(v) \wedge v \in E(w)$, so heißen v, w stark zusammenhängend, in Zeichen $v \sim w$.
Gilt $v \sim w$ für alle Paare $v, w \in V$, so heißt der Graph stark zusammenhängend.

2.3.3 Definition: Zusammenhangskomponente

$ZK(v) := \{w \in V | w \sim v\}$.

Bemerkung:

Für $v \neq w$ gilt $ZK(v) = ZK(w) \vee ZK(v) \cap ZK(w) = \emptyset$. Die ZK bilden eine Partition der Eckenmenge V.

2.3.4 Definition: schwach zusammenhängend

Ein gerichteter Graph heißt schwach zusammenhängend, wenn seine symmetrische Hülle stark zusammenhängend ist (Wenn man die Pfeilrichtung nicht betrachtet, und alle Punkte zusammenhängen).
Für ungerichtete Graphen spricht man nur noch von zusammenhängenden Graphen.

Algorithmus:

Eingabe: $G = (V, E)$
Ausgabe: alle Zusammenhangskomponenten

- setze $k \leftarrow 0$ (Nummer der Zusammenhangskomponente)
- if (v_i noch unmarkiert)
- setze $k \leftarrow k + 1$
- berechne ($E(v_i)$) und markiere dabei alle besuchten Ecken mit k

Laufzeit: $O(m + n)$
Bestimmung von starken Zusammenhangskomponenten in gerichteten Graphen mit Tiefensuche (später!).

2.3.5 Definition: Schnitt

Ein Schnitt ist eine Partition der Eckenmenge in zwei nicht leere Teile. $V = S \cup (V - S)$ mit $S \neq \emptyset, V$
Wir notieren:
$\delta^+(S) := \{r | \alpha(r) \in S, \omega(r) \notin S\}$ der Vorwärtsteil des Schnittes.
$\delta^-(S) = \delta^+(V - S)$ der Rückwärtsteil des Schnittes.

2.3.6 Satz:

Ein gerichteter Graph ist stark zusammenhängend genau dann, wenn $\delta^+(S) \neq \emptyset$ für alle $S \subset V$, $S \neq \emptyset, V$.

Beweis:

" $\Rightarrow$ ":
Wäre $\delta^+(S) = \emptyset$ für ein $S \neq \emptyset, V$, so wären die Ecken in $V - S$ nicht mehr erreichbar. Dies ist ein Widerspruch.
" $\Leftarrow$ ":
Zu zeigen: $w \in E(v)$ für alle $v, w \in V$
Konstruktiv: beginne mit $S := \{v\}$.
Invariante: $S \subseteq E(v)$.
Iteriere: Wähle $r \in \delta^+(S)$ (existiert nach Vorraussetzung) und füge $\omega(r)$ zu S hinzu.
Abbruch bei $S = V$. Damit ist $w \in E(v) = V$ nachgewiesen. $\square$

Schnitt bei ungerichteten Graphen: $\delta(S)$ Kantenmenge. Analog: Ein ungerichteter Graph ist genau dann zusammenhängend, wenn $\delta(S) \neq \emptyset$ für alle $S \subset V$, $S \neq \emptyset, V$.

2.3.7 Satz:

Ist $G = (V, E)$ zusammenhängend, so gilt $|E| \geq |V| - 1$.

Beweis:

- Setze $S := \{v_1\}$ und $E' := \emptyset$.
- Invariante: $|E'| = |S| - 1$ und alle Kanten in E' haben beide Endpunkte in S.
- Iteriere: solange $S \neq V$
  - da $\delta(S) \neq \emptyset$ wähle eine Kante e aus $\delta(S)$.
  - wegen $e \notin E'$ füge e zu E' hinzu und füge den neuen Endpunkt zu S hinzu.
- Bei Abbruch ist $S = V$ und damit $|E'| = |V| - 1$. Wegen $E' \subseteq E$ ist $|E| \geq |V| - 1$ gezeigt. □

2.3.8 Definition: Mehrfacher Zusammenhang

Ein ungerichteter Graph heißt k-fach zusammenhängend, wenn er nach Wegnehmen von $k-1$ vielen beliebig ausgewählten Kanten immer noch zusammenhängend ist.

2.3.9 Definition: allgemeiner Weg (walk)

$\omega(r_i) = \alpha(r_{i+1})$

2.3.10 Definition: einfacher Weg (trail)

Genauso wie allgemeine Definition von Weg, nur dass zusätzlich die Pfeile paarweise verschieden sind.

2.3.11 Definition: elementarer Weg (path)

Genauso wie einfacher Weg, nur dass zusätzlich alle Ecken paarweise verschieden sind. Die Ausnahme $\omega = \alpha$ ist zugelassen.

2.4 Bipartite und k-partite Graphen

2.4.1 Definition: k-partite Graphen

Ein Graph $G = (V, E)$ heißt k-partit, wenn es eine Partition $V = V_1 \cup ... \cup V_k$ gibt, so dass die induzierten Graphen $G[V_1], ..., G[V_k]$ keine Kanten enthalten (mit anderen Worten: die Endpunkte jeder Kante liegen in verschiedenen Teilen der Partition). Für $k = 2$ sagen wir bipartit.

Bemerkung:

ein $k-$partiter Graph ist $k-$färbbar.

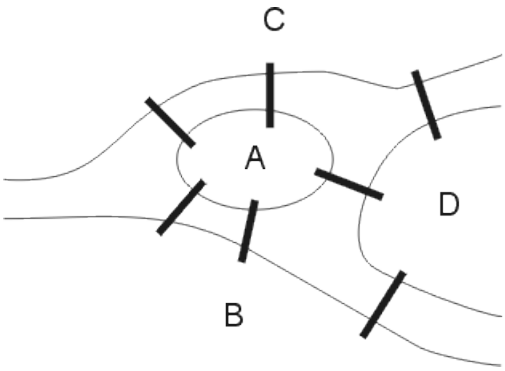
$K_{r,s}$ ist der vollständig bipartite Graph mit Partition $V = V_r \cup V_s$ mit $|V_r| = r$ und $|V_s| = s$ und Kantenmenge $V_r \times V_s$.

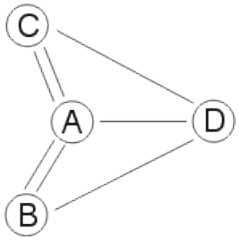
Beispiel:

Für $r = s = \frac{n}{s}$ ergeben sich $\frac{n^2}{2}$ viele Kanten.

2.5 Eulersche Wege

Das Königsberger Brückenproblem





Gibt es einen Kreis im Graphen, der alle Kanten genau einmal benutzt?

2.5.1 Definition: Eulersch

Ein Weg heißt Eulersch, wenn er alle Pfeile (Kanten) eines Graphen genau einmal benutzt. Ein Graph heißt Eulersch, wenn er einen Euerlschen Kreis zulässt.

2.5.2 Definition: Gradbalance

$gb(v) = g^+(v) - g^-(v)$.

2.5.3 Satz von Euler:

Ein schwach zusammenhängender Graph $G = (V, R)$ ist Eulersch genau dann, wenn $gb(v) = 0$ für alle Ecken $v \in V$.

Beweis:

" $\Rightarrow$ ":

- durchlaufe den Kreis. Offenbar wird jede erreichte Ecke auch wieder verlassen, also gilt $g^+(v) = g^-(v)$.

" $\Leftarrow$ ":

- Unterroutine zur Ermittlung eines Kreises im Graphen:
  - Wähle $v_0 \in V$ beliebig mit $g^+(v_0) > 0$.
  - setze $v \leftarrow v_0$
  - repeat:
    - wähle $r \in \delta^+(v)$
    - lösche r aus dem Graphen
    - setze $v \leftarrow \omega(r)$
    - until $v = v_0$
- Korrektheit:
  - nach dem ersten Schritt gilt $gb(v_0) = -1$, $gb(v) = +1$ und $gb = 0$ sonst. Ebenso nach jedem weiteren Schritt. Bei Abbruch gilt $gb = 0$ für alle Knoten.
  - die Folge der gelöschten Pfeile ist ein Weg.
  - wenn eine Ecke $v \neq v_0$ besucht wird, gilt offensichtlich $g^-(v) > 0$ und damit auch $g^+(v) > 0$, also kann sie auch wieder verlassen werden.
  - da Pfeile gelöscht werden, muss das Verfahren terminieren, was nur in v_0 möglich ist. Damit schließt sich der Kreis.
- Falls nach der Unterroutine noch Pfeile unbenutzt sind:

- Wähle v_0 auf dem aktuellen Kreis C_0 mit $g^+(v_0) > 0$
- ermittle neuen Kreis C von $v_0$aus
- füge C in C_0 ein.
- Wiederhole, bis alle Pfeile tatsächlich benutzt sind.

- $\Rightarrow$ es entsteht ein Euler-Kreis. $\square$

Laufzeit eines Algorithmus:

Adjazenzlistendarstellung. Ersetze die Operation "Löschen eines Pfeils" durch "Markieren eines Pfeils". Wähle die Pfeile von $\delta^+(v)$ in der Reihenfolge der Adjazenzliste. Daher ist die Laufzeit $O(m+n)$ möglich.

2.5.4 Korollar:

Ein schwach zusammenhängender Graph mit $gb(v) = 0$ für alle Ecken $v \in V$ ist stark zusammenhängend. $\square$

2.5.5 Satz:

Falls es im Graphen Ecken $s \neq t$ gibt mit $gb(s) = +1$, $gb(t) = -1$ und $gb(v) = 0$ sonst, dann hat der Graph einen eulerschen Weg, der in s startet und in t endet.

Satz von Euler für ungerichtete Graphen:

Ein ungerichteter Graph heißt eulersch genau dann, wenn alle Ecken geraden Grad haben.
Ein ungerichteter Graph hat einen eulerschen Weg, der kein Kreis ist, wenn Grad an Ecken $s \neq t$ ungerade ist und an allen anderen Ecken gerade. (Beispiel: Haus vom Nikolaus).

2.6 Hamiltonsche Wege

2.6.1 Definition: hamiltonsch

Ein Weg heißt hamiltonsch, wenn er jede Ecke des Graphen genau einmal besucht. (Anfang = Ende ist zugelassen, dann heißt er Hamiltonscher Kreis. Ein Graph heißt hamiltonsch, wenn er einen Hamiltonschen Kreis zulässt.

Entscheidungsproblem (Hamiltonian)

Gegeben sei ein gerichteter/ungerichteter Graph. Die Frage ist, existiert ein Hamiltonscher Kreis?

2.6.2 Satz: Hamiltonian

Das Hamiltonian-Problem ist NP-vollständig.

Beobachtung

Ein Hamiltonscher Kreis ist ein elementarer Weg der Länge n.

zugehöriges Optimierungsproblem (Longest Path)

Gegeben sei ein gerichteter/ungerichteter Graph. Die Frage ist, wie groß ist die Länge eines längsten elementaren Weges? (Antwort: $1 \leq k \leq n$).

2.6.3 Korollar

Das Longest-Path-Problem ist NP hart (d.h. mindestens so schwierig wie ein NP-vollständiges Problem).

Beweis

Der Beweis ist eine triviale Reduktion des Hamiltonian. Die Antwort auf Hamiltonian ist "ja" $\Leftrightarrow$ Die Antwort auf Longest Path ist n.

2.7 Komplexität

Beobachtung

Wir betrachten ein Paar aus einem Entscheidungsproblem und einem Optimierungsproblem. Ist ersteres NP-vollständig, so ist letzteres NP-hart, d.h. mindestens so schwierig.

2.7.1 Definition: Die Klasse P

Die Klasse P enthält alle Entscheidungsprobleme, die von einer deterministischen Touringmaschine in polynomieller Laufzeit gelöst werden können (auf einem Graphen also in $O(polynom(n))$).

2.7.2 Definition: Die Klasse NP

Die Klasse NP enthält alle Entscheidungsprobleme, die von einer nicht-deterministischen Touringmaschine in polynomieller Laufzeit gelöst werden können.
Alternative Definition:
NP enthält alle Entscheidungsprobleme, die in polynomieller Zeit verifiziert werden können.

Beispiel: Hamiltonian

Falls ein Pfad gegeben ist, kann sehr leicht nachgeprüft werden, ob er Hamiltonsch ist.

2.7.3 Definition: NP-vollständig

Die Klasse der NP-vollständigen Probleme enthält die schwierigsten Probleme aus NP, d.h. sie sind aufeinander reduzierbar.

Reduktion:

Reduktion ist ein Verfahren zur Abschätzung der Komplexität eines Problems Π, wenn die Komplexität eines Problems Π' schon bekannt ist. Deduktion von Π' auf Π.

$Instanz\,von\,\Pi'$	$\overset{"schwierig"}{\rightarrow}$	$Loesung\,von\,\Pi'$
$\downarrow Konstruieren\,einer\,neuen\,Instanz : leicht$		$\uparrow (Ablesen\,des\,Ergebnisses : ja \Leftrightarrow Laenge = n$
$Instanz\,von\,\Pi$	$\overset{?}{\rightarrow}$	Lösung von Π
(Longest Path)	Folge: NP-vollständig	(Länge des Weges)

2.7.4 Definition: Optimierungsproblem

- Menge der zulässigen Lösungen
- Zielfunktion, die jeder zulässigen Lösung einen numerischen Wert zuordnet
- Angabe, ob Zielfunktion minimiert/maximiert werden soll

2.7.5 Beispiel: Longest Path

- Menge: alle elementaren Pfade
- Zielfunktion: Länge des Weges

Bemerkung

Entscheidungsproblem NP-vollständig $\Leftrightarrow$ zugehöriges Optimierungsproblem NP-hart. Möglicherweise sind NP-harte Optimierungsprobleme nicht in Polynomialzeit lösbar.

Approximation

Gegeben sei ein Minimierungsproblem Π. Zu jeder Instanz x sei $\Pi(x)$ der optimale Lösungswert. Ist A ein Polynomialzeit-Algorithmus mit Ergebnis $A(x)$ und gilt $A(x) \leq r \cdot \Pi(x)$ für alle Instanzen x, so heißt A ein r-Approximationsalgorithmus.

3 Nachbarschaften

In diesem Kapitel sind alle Graphen ungerichtet.

3.1 Cliquen, unabhängige Mengen, Färbungen

3.1.1 Definition: unabhängige Mengen

Eine Teilmenge $U \subseteq V$ heißt unabhängig, falls $G[U]$ keine Kanten enthält (in anderen Worten: keine zwei Ecken aus U sind adjazent).

3.1.2 Definition: Clique

Eine Menge $C \subseteq V$ heißt Clique, falls $G[U]$ ein vollständiger Graph ist.

Beobachtung:

U ist unabhängig in $G \Leftrightarrow U$ ist Clique in $\overline{G}$

3.1.3 Definition: Färbung

Eine Funktion $f : V \to \{1, ..., k\}$ heißt $k-$Färbung, wenn $f^{-1}(i)$ unabhängige Mengen sind für alle $i = 1, ..., k$.

3.1.4 Definition: Cliquenzerlegung

Eine Partition $V = V_1 \cup ... \cup V_k$ heißt Cliquenzerlegung, falls alle V_i jeweils eine Clique sind.

Beobachtung:

Eine Partition ist Cliquenzerlegung in G $\Leftrightarrow$ sie bildet Farbklassen in $\overline{G}$.

Extremale Graphentheorie

Eine Menge heißt (bzgl. Inklusion) maximal für eine Eigenschaft, wenn jede echte Obermenge diese Eigenschaft nicht mehr hat. Eine Menge heißt Maximum-Menge, falls sie im Graphen größte Kardinalität hat unter allen Mengen im Graphen mit dieser Eigenschaft.
Eine Menge heißt (bzgl. Inklusion) minimal für eine Eigenschaft, wenn jede echte Teilmenge diese Eigenschaft nicht mehr hat. Eine Menge heißt Minimum-Menge, falls sie im Graphen kleinste Kardinalität hat unter allen Mengen im Graphen mit dieser Eigenschaft.

3.1.5 Definition: Extremum-Zahlen für einen Graphen G

Unabhängigkeitszahl:$\alpha(G) := max\{|U| \,|\text{U ist unabhängig}\}$
Cliquenzahl: $\omega(G) := max\{|C| \,|\text{C ist Clique}\}$
chromatische Zahl $\chi(G) := min\{k|\text{es gibt eine k-Färbung}\}$
Cliquenzerlegungszahl: $\overline{\chi}(G) := min\{k|\text{es gibt k-Cliquenzerlegungen}\}$

3.1.6 Satz

Es gelten:

- $\alpha(G) = \omega(\overline{G})$
- $\overline{\chi}(G) = \chi(\overline{G})$
- $\alpha(G)\} \leq \overline{\chi}(G)$
- $\omega(\overline{G}) \leq \chi(\overline{G})$

Beweis

1. Ist U unabhängig in G $\Leftrightarrow$ U ist Clique in $\overline{G}$
2. $\overline{\chi}(G) \geq \chi(\overline{G})$, die k-Cliquenzerlegung in G induziert k unabhängige Mengen in $\overline{G}$, jede kann eine Farbe bekommen.
   $\overline{\chi}(G) \leq \chi(\overline{G})$, jede Farbklasse in $\overline{G}$ ist Clique in G. k Farben $\Rightarrow$ k Cliquen in G.
3. z.Z: $\omega(G) \leq \chi(G)$. Jede Ecke einer Clique muss eine eigene Farbe tragen.
4. folgt aus 1-3.

3.1.7 Satz: chromatische Zahl

1. $\chi(G) \geq \frac{n}{\alpha(G)}$
   Beweis: sei f eine optimale Färbung, sie induziert Farbklassen.
   $$n = \sum_i |\left(f^{-1}(i)\right)| \leq \sum_i \alpha(G) = \chi(G) \cdot \alpha(G)$$
2. $\chi(G) \leq \Delta(G) + 1$
3. $\chi(G) \leq \Delta(G)$, falls G nicht ein K_n oder ein Kreis ungerader Länge ist. (Satz von Brooks)

Algorithmus für Graphfärbung

Sequential Coloring:

- Wähle eine Folge $v_1, ..., v_n$
- Für $i = 1, ..., n$: Färbe die Ecke v_i mit der kleinsten Farbnummer, die keinen Konflikt mit $v_1, ..., v_{i-1}$ erzeugt.

Approximationsgüte? Schlecht:
Betrachte einen vollständig bipartiten Graphen $\{v_1, ..., v_n, u_1, ..., u_n\}$, aus dem die Kanten (v_i, u_i) entfernt wurden.

Sequential Coloring in Reihenfolge $v_1, u_1, v_2, u_2, ..., v_n, u_n$.
Ergebnis: n viele Farben sind vergeben (bei $2n$ Knoten).
Die Approximationsgüte ist also schlechter als $\frac{\frac{n}{2}}{2} = \frac{n}{4}$.

Bemerkung: Extremumprobleme

- Maximum Independent Set ($\alpha(G)$)
- Maximum Clique ($\omega(G)$)
- Minimum Graph Coloring ($\chi(G)$)

sind alle NP-hart.

3.1.8 Definition: Matching

Eine Kantenmenge $M \subseteq E$ heißt Matching, falls keine zwei Kanten aus M inzidieren ("unabhängige Kantenmenge").

3.2 Überdeckungen

3.2.1 Definition

Eine Eckenmenge $V' \subseteq V$ heißt Eckenüberdeckung (vertex cover), wenn sie inzident zu jeder Kante ist. Sie heißt dominierende Menge, wenn sie adjazent zu jeder Ecke ist.

Bemerkung:

Die entsprechenden Optimierungsprobleme

- Maximum Independent Set
- Maximum Clique
- Minimum Vertex Cover
- Minimum Dominating Set

sind alle NP-hart.

Lemma

Jede maximale unabhängige Menge ist dominierend.
Beweis: siehe Übungsblatt 3.

Lemma

Sei M ein beliebiges Matching und C eine beliebige Eckenüberdeckung, dann gilt $|M| \leq |C|$.

Beweis:

Um jede Kante des Matchings zu überdecken, ist pro Kante eine eigene Ecke in C nötig.

Lemma

Sei M ein maximales Matching. Dann gibt es eine Eckenüberdeckung mit $|C| \leq 2 \cdot |M|$.

Beweis:

Wähle C als Menge aller Endecken der Kanten bis aus M. C ist eine Eckenüberdeckung, andernfalls gäbe es eine Kante (u, v), die noch nicht von C überdeckt wäre. Diese könnte zu M hinzugefügt werden. Widerspruch zu maximales Matching.

3.2.2 Satz (Minimal Vertex Cover)

das Problem Minimal Vertex Cover kann in der Zeit $O(m + n)$ auf Faktor 2 approximiert werden.

Beweis:

Der Greedy-Algorithmus bestimmt ein verändertes Matching in $O(m + n)$. Wähle C wie oben. C ist Vertex Cover. Sei C^* eine optimale Lösung. Dann gilt:

$$\begin{aligned} |C| &\leq 2 \cdot |M| \\ &\leq 2 \cdot |C^*| \end{aligned}$$

4 Bäume, Wälder und Matroide

In diesem Kapitel sind alle Graphen ungerichtet.

4.1 Bäume

4.1.1 Definition: Baum, Wald

Ein Graph heißt Wald, wenn er kreisfrei ist. Ein zusammenhängender Wald heißt Baum.

4.1.2 Definition: aufspannend

Ein Teilgraph H spannt eine Zusammenhangskomponente auf, wenn er die gleiche Eckenmenge wie die Zusammenhangskomponente besitzt und selbst zusammenhängend ist. Ein Teilgraph spannt einen Graphen auf, wenn er jede Zusammenhangskomponente des Graphen aufspannt.
→ aufspannender Baum, aufspannender Wald.

Erinnerung an Satz 2.14 (??):

Ist G zusammenhängend mit n Ecken, so besitzt G mindestens $n-1$ Kanten.

4.1.3 Korollar:

Ein aufspannender Wald mit k Bäumen besitzt mindestens $n-k$ Kanten.

Beweis:

Sei E die Kantenmenge des Waldes und $V = V_1 \cup ... \cup V_k$ die Partition der Ecken in die Bäume.

$$|E_i| \quad \geq \quad |V_i| - 1$$

mitsummieren:

$$\begin{aligned} |E| = \quad & \textstyle\sum_i |E_i| \quad && \geq \sum_i (|V_i| - 1) \\ = \quad & (\textstyle\sum_i |V_i|) - k \\ = \quad & |V| - k \end{aligned}$$

4.1.4 Lemma

Seien $s, t \in V$ verschiedene Ecken. Wenn in G zwei verschiedene elementare Wege von s nach t bestehen, dann hat G einen elementaren Kreis.

Beweis:

Sei p ein $s-t-Weg$ mit der Spur $(v_0 = s, v_1, v_2, ..., v_l = t)$ und q ein $s-t-Weg$ mit der Spur $(u_0 = s, u_1, u_2, ..., u_l = t)$.
Abschneiden der gemeinsamen Anfangs- und Endstücke:
Wähle i maximal, so dass $v_{i'} = u_{i'}$ für alle $i' \leq i$, aber $v_{i'+1} \neq u_{i'+1}$.
Wähle j maximal, sodass $v_{l-j'} = u_{k-j'}$ für alle $j' \leq j$, aber $v_{i-j-1} \neq u_{k-j-1}$.
Die beiden Teilwege von p, q zwischen v_i und v_{l-j} bilden zusammen einen Kreis der Länge ≥ 3.
Falls dieser nicht elementar ist, dann wähle einen geeigneten Teilkreis. □

4.1.5 Satz: Baum

Sei $G = (V, E)$. Dann sind äquivalent:

1. G ist ein Baum.
2. G ist kreisfrei, aber jeder echte Obergraph mit derselben Eckenmenge ist nicht kreisfrei (maximal kreisfrei).
3. Für jedes Paar $u, v \in V$ existiert in G genau ein elementarer Weg zwischen u und v.

4. G ist zusammenhängend, aber für jede Kante $e \in E$ ist $G - e$ nicht zusammenhängend (minimal zusammenhängend).
5. G ist zusammenhängend, und $|E| = |V| - 1$.
6. G ist kreisfrei und $|E| = |V| - 1$.

Beweis:

$1 \to 2$: Sei G ein Baum, also zusammenhängend und kreisfrei. Sei $e \notin E$ eine neue Kante mit $e = (u, v)$ .Falls $u = v$, bildet diese Schlinge einen Kreis. Sei also $u \neq v$. Da G zusammenhängend ist, existiert ein Weg p von u nach v. Dann ist $p \circ e$ ein Kreis.
$2 \to 3$: Sei G kreisfrei, aber jeder Obergraph nicht. Zu zeigen: Für alle u, v gibt es genau einen elementaren $u - v - Weg$.

- Für $u = v$ gilt: da G kreisfrei und damit schlingenfrei ist, existiert nur der leere Weg. Dieser ist eindeutig.
- Für $u \neq v$ gilt:
  - Existenz: Bilde $G' := (V, E')$ mit $E' := E \cup e := (u, v)$. Da G' einen Kreis enthält, muss dieser die neue Kante benutzen. Sei C der Kreis, zerlege $C = p \circ e$. Dann ist p ein $u - v - Weg$ in G.
  - Eindeutigkeit: vergleiche Lemma von oben: 2 verschiedene elementare Wege in G würden die Kreisfreiheit verletzen.

$3 \to 4$: Für jedes Paar u, v existiert genau ein elementarer Weg in G. Zu zeigen: G ist zusammenhängend, aber für alle $e \in E$ ist $G - e$ nicht zusammenhängend.

- Zusammenhang: klar.
- Wähle eine Kante $e \in E$. e ist keine Schlinge. Sei $e = (u, v)$. Wäre $G - e$ zusammenhängend, dann enthielte er einen Weg p von u nach v. p wäre ein zweiter $u - v - Weg$ in G. Widerspruch zur Voraussetzung.

$4 \to 5$: Sei G zusammenhängend, aber $G - e$ nicht für alle $e \in E$. Zu zeigen: G ist zusammenhängend mit $|E| = |V| - 1$. Da G zusammenhängend ist, ist $|E| \geq |V| - 1$ bereits gezeigt. Es bleibt zu zeigen: $|E| \leq |V| - 1$.

- Mit vollständiger Induktion nach der Zahl der Ecken.
- $n = 1$: es muss $|V| = 1$ und $E = \emptyset$ sein. $\surd$
- $n \to n + 1$: Sei die Annahme bereits für alle kleineren Graphen bereits gezeigt.
  - Betrachte G, wähle $e \in E$, entferne e: $G' := G - e$.
  - G' zerfällt in Zusammenhangskomponenten $G_1, ..., G_p$ mit $p > 1$.
  - Für jede Komponente G_i gilt nach Induktionsvoraussetzung: $|E_i| \leq |V_i| - 1$.

$$\begin{aligned} |E| = & \quad 1 + \textstyle\sum_{i=1}^{p} (|V_i| - 1) & = \\ = & \quad 1 + (\textstyle\sum_i |V_i|) - p & = \\ = & \quad 1 + |V| - p & \leq |V|(-1) \end{aligned}$$

$5 \to 6$: Sei G zusammenhängend mit $|E| = |V| - 1$. Zu zeigen: G ist kreisfrei und $|E| = |V| - 1$. Wäre G nicht kreisfrei, so wähle Kreis C aus, und darin eine Kante e. Dann ist $G - e$ immer noch zusammenhängend, mit $|E| = |V| - 2$. Widerspruch zu Satz 2.14.
$6 \to 1$: Sei G kreisfrei mit $|E| = |V| - 1$. Zu zeigen: G ist ein Baum (d.h. kreisfrei und zusammenhängend).

- G ist ein Wald. Seien $G_1, ..., G_p$ die Zusammenhangskomponenten, also Bäume. Wir sind fertig, wenn $p = 1$ nachgewiesen ist. Für Bäume ist wegen $1 \to 2 \to 3 \to 4 \to 5 \to 6$ bereits gezeigt, dass $|E_i| = |V_i| - 1$.

$$\begin{aligned} |V| - 1 = & \quad |E| = & \sum_i |E_i| = \\ = & \quad \textstyle\sum_i (|V_i| - 1) & = \\ = & \quad (\textstyle\sum_i |V_i|) - p & = \\ = & \quad |V| - p \end{aligned}$$

  Also ist $p = 1$.

4.2 Der Minimum-aufspannende Baum

4.2.1 Algorithmus von Kruskal

Problem MST (Minimum Spending Tree):
Gegeben ist ein zusammenhängender Graph $G = (V, E)$ und eine Kostenfunktion $c : E \to \mathbb{R}_0^+$. Gesucht ist ein aufspannender Baum T in G mit den minimalen Kosten $c(T) := \sum_{e \in T} c(e)$.
Analog: Problem MSF (Minimum Spending Forest), falls G nicht zusammenhängend ist: Suche aufspannenden Wald F kleinsten Gewichtes $c(F)$.

Im Folgenden ist G zusammenhängend, sonst betrachte jede Zusammenhangskomponente einzeln.
Algorithmus von Kruskal:
Eingabe: $G = (V, E)$, $c : E \to \mathbb{R}_0^+$.
Ausgabe: MST (oder MSF)

1. sortiere die Kanten nach ihren Kosten: $c(e_1) \leq c(e_2) \leq ... \leq c(e_m)$
2. $F \leftarrow \emptyset$
3. for $i = 1\, to\, m$
   (a) if $(V, F \cup \{e_i\})$ kreisfrei
   (b) setze $F \leftarrow F \cup \{e_i\}$
4. return (V, F).

Korrektheit:

Eine Teilmenge $F \subseteq E$ heißt fehlerfrei, wenn es einen MST $T_0 = (V, E_0)$ gilt, sodass $E_0 \supseteq F$. Ist F fehlerfrei, und $e \in E \backslash F$, und wird $F \cup \{e\}$ ebenfalls fehlerfrei, so heißt e sichere Kante für F.

Lemma:

Sei $F \subseteq E$ fehlerfrei. Sei A ein Schnitt (d.h. $\emptyset \neq A \subsetneq V$) mit $\delta(A) \cap F = \emptyset$. Wähle $e \in \delta(A)$ mit unterstem Gewicht, d.h. $e = arg\, min\, \{c(e) | e \in \delta(A)\}$. Dann ist e sicher für F.

Beweis:

Sei $T_0 = (V, E_0)$ ein MST mit $E_0 \supseteq F$. Sei $e \in \delta(A)$, aber $e \notin E_0$. $T_0 + e$ enthält dann einen Kreis, dieser enthält (mindestens eine) zweite Kante $e' \neq e$ und $e' \in \delta(A)$. Es ist $c\,(e') \geq c(e)$. Sei $T' := T_0 + e - e'$.
Dann gilt:

1. T' ist zusammenhängend und hat $|V| - 1$ Kanten $\Rightarrow T'$ ist Baum.
2. Seie Kantenmenge umfasst $F \cup \{e\}$
3. $c\,(T') \leq c\,(T_0)$, also ist T' ein MST.

4.2.2 Satz

Der Algorithmus von Kruskal liefert einen MSF bzw. MST.

Beweis:

Sei G zusammenhängend (sonst betrachte die Zusammenhangskomponenten).
Sei $T = (V, F)$ bei Abbruch.

1. T ist Baum: T ist kreisfrei nach Konstruktion. T ist auch zusammenhängend, andernfalls betrachte die Zusammenhangskomponenten $T_1, .., T_k$, $k \geq 1$. Weil G zusammenhängend ist, gibt es eine Kante $e \in \delta(T_1)$. Beim Test wäre sie zu F hinzugenommen worden. Widerspruch!
2. Jede hinzugenommene Kante e_i ist eine sichere Kante: Sei F_i die Situation zu Beginn des Schrittes i. Sei $e_i = (u, v)$. Sei U die Zusammenhangskomponente von F_i, in der u liegt. U ist Schnitt und $\delta(U) \cap F_i = \emptyset$. $\delta(U)$ enthält keine Kante, die leichet als e_i ist, denn diese wäre früher betrachtet und auch hinzugenommen worden $\Rightarrow e_i$ ist sicher für F_i.

3. per Induktion: F ist fehlerfrei und muss damit ein MST sein.

Laufzeit:

1. Sortierung der Kanten: $O(m \log m)$
2. Test auf Kreise kann $O(n)$ pro Test sein, also $O(m \cdot n)$ für Phase 2.

Falls der Graph dicht ist, also $m \in \Theta\left(n^2\right)$, ergibt sich eine Laufzeit von $o\left(n^3\right)$.

Beobachtung:

$e = (u, v)$ schließt einen Kreis in (V, F_i) genau dann, wenn u, v in F_i zusammenhängend sind.

4.2.3 Union-Find-Datenstruktur

Drei Operationen:

1. MAKESET(x) erzeugt eine Menge, die nur x enthält. Diese hat x als Repräsentant.
2. FINDSET(x) liefert einen Zeiger auf den Repräsentanten der Menge, in der x liegt.
3. UNION(x,y) vereinigt die dusjunkten Mengen, in denen x und y liegen und wählt einen neuen Repräsentanten aus der Vereinigung.

Idee: Graph mit Knotenmenge = Menge der Elemente. Jedes Element hat einen gerichteten Weg zum Repräsentanten.
Laufzeit: UNIONFIND ist sehr effizient implementierbar:
k Operationen, von denen n MAKESET() waren, benötigen Zeit $O(k \cdot \alpha(n))$. Hierbei ist $\alpha()$ die Inverse der Ackermann-Funktion (es ist z.B. $\alpha(n) \leq 4$ für $n \leq 10^{684}$).

4.2.4 Satz

Der Algorithmus von Kruskal kann in Zeit $O(m \cdot \alpha(n))$ plus Laufzeit für die Sortierung (i.d.R $O(m \log m)$) implementiert werden.

4.3 Der Algorithmus von Prim

Problem bei Kruskal: Sortierung vieler Elemente.
Idee bei Prim: Wachsen eines Baumes.

1. Setze $S \leftarrow \{s\}$mit $s \in V$ beliebig.
2. Setze $F \leftarrow \emptyset$
3. while $S \neq V$
   (a) wähle $e \in \delta(S)$ mit minimalem Gewicht
   (b) Sei $e = (u, v)$ mit $u \in S$
   (c) Füge v zu S hinzu.

Implementierung:

- INSERT(x,i)
- EXTRACT_MIN()
- DECREASE_KEY(x,i')

4.3.1 Satz

Der Algorithmus von Prim liefert einen MST in der Zeit $O(m + n \log n)$.

4.4 Unabhängigkeitssystem und Matroid

4.4.1 Definition (Unabhängigkeitssystem)

Sei S eine endliche Menge. $\mathcal{F} \subseteq 2^S$ sei eine Familie von Teilmengen von S. Falls

1. $A \in \mathcal{F} \wedge B \subseteq A \Rightarrow B \in \mathcal{F}$, so heißt $(S, \mathcal{F})$ein Unabhängigkeitssystem. Die Mengen in $\mathcal{F}$ heißen unabhängige Mengen. Gilt zusätzlich
2. $A, B \in \mathcal{F} \wedge |B| < |A| \Rightarrow \exists x \in A \backslash B$, sodass $B \cup \{x\} \in \mathcal{F}$, so heißt $(S, \mathcal{F})$ Matroid.

Beispiele

$S = \{1, 2, ..., 10\}$. $\mathcal{F} = \{F \subseteq S | \sum F \leq 10\}$. $\left(\mathcal{F} = \left\{\{1\}, \underbrace{\{1,2,3,4\}}_{A}, ..., \underbrace{\{4,6\}}_{B}, ...\right\}\right)$. Dieses $\mathcal{F}$ ist ein Unabhängigkeitssystem. Ist es ein Matroid? Nein, wegen den Mengen A, B.

Sei G gegeben. Wähle S als Menge aller Kanten. Wähle $\mathcal{F}$ als Menge aller Matchings auf dem Graphen. Dieses $\mathcal{F}$ ist ein Unabhängigkeitssystem, aber kein Matroid.

Sei G gegeben. Sei S die Menge aller Kanten. $\mathcal{F}$ sei die Menge aller Wälder (kreisfreien Teilgraphen). Dieses $\mathcal{F}$ ist ein Unabhängigkeitssystem und auch ein Matroid. (Wird unten gezeigt).

4.4.2 Definition

Sei $(S, \mathcal{F})$ ein Unabhängigkeitssystem. Sei $M \in \mathcal{F}$ eine unabhängige Menge und sei $A \in 2^S$ beliebig mit $A \supseteq M$. Wenn für jedes $M' \in \mathcal{F}$ mit $M \subseteq M' \subseteq A$ gilt, dass $M' = M$ ist, dann heißt M maximal bzgl. A. Die bzgl. S maximalen Mengen heißen maximale unabhängige Mengen.

4.4.3 Satz (Matroid)

Sei $(S, \mathcal{F})$ ein Unabhängigkeitssystem. Dann sind äquivalent:

1. $(S, \mathcal{F})$ ist Matroid.
2. Für jede Teilmenge $A \in 2^S$gilt: Sind $M_1, M_2 \in \mathcal{F}$ maximal bzgl. A, so ist $|M_1| = |M_2|$.

4.4.4 Satz (Graphisches Matroid)

Sei $G = (V, E)$ und $\mathcal{F} := \{F \subseteq E | (V, F)\, ist\, kreisfrei\}$. Dann ist $(E, \mathcal{F})$ ein Matroid, und die maximalen unabhängigend Mengen sind die aufspannenden Wälder von G.

Beweis

Sei $A \subseteq E$ beliebig gewählt. Es genügt zu zeigen: Jede bezüglich A maximale unabhängige Menge hat die gleiche Kardinalität.
Sei $F \in \mathcal{F}$ maximal bezüglich A. Seien $A_1, ..., A_k$ die Zusammenhangskomponenten von (V, A). Sei $T_1, ..., T_k$ die durch $A_1, ..., A_k$ induzierten Teilgraphen von (V, F).
T_i ist zusammenhängend, sonst wäre (V, F) nicht maximal. T_i ist kreisfrei. Also ist T_i ein Bau und besitzt $|A_i| - 1$ viele Kanten.
Aufsummieren: $|F| = \sum_i (|A_i| - 1) = |V| - k$

Greedy-Algorithmus für Matroide

Eingabe: Matroid $(S, \mathcal{F})$, Gewichtsfunktion $c : S \to \mathbb{R}$.
Ausgabe: maximale unabhängige Menge minimalen Gewichts.

1. sortiere S, sodass $c(s_1) \leq ... \leq c(s_n)$
2. $M \leftarrow \emptyset$
3. for $i = 1\, to\, m$

(a) if $M + s_i \in \mathcal{F}$

i. $M \leftarrow M + s_i$

4. return M

4.4.5 Satz (Edmonds)

Der Greedy-Algorithmus auf einem Matroid findet stets eine maximale unabhängige Menge minimalen Gewichts.

Beweisidee:

Ist M_k die Menge nach dem $k-ten$ Schritt, so gilt:

$$c(M_k) = min\{c(I)|I \in \mathcal{F} \wedge |I| = k\}$$

Bemerkung: Es gilt auch die Umkehrung: Wenn auf einem Unabhängigkeitssystem U der Greedy-Algorithmus für jede Gewichtsfunktion c stets eine maximale unabhängige Menge minimalen Gewichts liefert, dann ist U ein Matroid.
Bemerkung: Folgt (erneut) die Korrektheit des Algorithmus von Kruskal (dieser ist ein Sonderfall).

4.5 Wurzelbäume

In diesem Abschnitt sind alle Graphen gerichtet mit $G = (V, R)$ und haben Pfeilkosten $c : R \to \mathbb{R}$.

Motivation

zum Beispiel in der Informationsübertragung von einer Quelle zu allen Empfängern.

4.5.1 Definition (Baum)

Ein gerichteter Graph heißt Baum, wenn der zugehörige ungerichtete Graph ebenfalls ein Baum ist.

4.5.2 Definition (Wurzel)

Eine Ecke $s \in V$ heißt Wurzel, falls $E(s) = V$. Ein s-Wurzelbaum ist ein Baum mit Wurzel s.

4.5.3 Satz (Wurzelbaum)

Sei $G = (V, R)$, $s \in V$, dann sind äquivalent:

1. G ist s-Wurzelbaum
2. G ist ein Baum und $g^-(v) = \begin{cases} 1 & v \neq s \\ 0 & v = s \end{cases}$
3. $E(s) = V$ und $g^-(v) \begin{cases} \leq 1 & v \neq s \\ = 0 & v = s \end{cases}$

Beweis:

$1 \to 2$: Da $E(s) = V$, muss $g^-(v) \geq 1$ für alle $v \neq s$ gelten.

$$|V| - 1 = |R| = \sum_{v \in V} g^-(v) = g^-(s) + \sum_{v \neq s} g^-(v) \geq g^-(s) + |V| - 1 \geq |V| - 1$$

d.h. überall gilt Gleichheit und die Behauptung folgt.
$2 \to 3$: Sei $v \neq s$. Betrachte den Weg von s nach v $(e_1, ..., e_k)$ im ungerichteten Graphen. ei $(s = v_0, v_1, ..., v_k = v)$ seine Spur. Wegen $g^-(s) = 0$ muss der zu e_1 gehörige Pfeil die Richtung (s, v_1) haben. Induktiv: Wegen $g^-(v_i) = 1$ muss der Pfeil r_{i+1} die Richtung (v_i, v_{i+1}) haben. $\Rightarrow$ es gibt einen s-v-Weg im gerichteten Graphen.

$3 \to 1$: Sei$E(s) = V$ und sei $g^-(v) \leq 1$ für $v \neq s$ und $g^-(s) = 0$. Wegen $E(s) = V$ ist G zusammenhängend, also gilt $|R| \geq |V| - 1$

$$|R| = \sum_{v \in V} g^-(v) \leq |V| - 1$$

also ist $|R| = |V| - 1$ und G ist ein Baum. wegen der Erreichbarkeit $E(s) = V$ ist G sogar ein s-Wurzelbaum. □

4.5.4 Lemma (Existenz)

G besitzt einen s-Wurzelbaum genau dann, wenn $E(s) = V$ gilt. Konstruktion in linearer Zeit.

1. Starte bei s
2. Setze $S \leftarrow \{s\}$
3. Betrachte $\delta^+(S)$ und erweitere $S \leftarrow S + \delta^+(S)$
4. Breche ab, wenn $\delta^+(S) = \emptyset$.

4.6 Kostenminimaler Wurzelbaum

Problemstellung:

gegeben: $G = (V, R)$, $s \in V$, $c : R \to \mathbb{R}$.
gesucht: Teilgraph T mit

- T ist aufspannender s-Wurzelbaum
- $c(T) = \sum_{r \in T} c(r)$ ist minimal

Idee für einen Algorithmus: Wegen $g^-(v) = 1(v \neq s)$ wähle aus dem Büschel $\delta^-(v)$ jeweils einen billigsten Pfeil.

4.6.1 Definition (Reduzierte Kosten)

Setze $z(v) := min\{c(r) | r \in \delta^-(v)\}$ für alle $v \in V$. $c'(u, v) := c(u, v) - z(v)$ für alle $(u, v) \in R$. Folge: $c' \geq 0$. Für jedes $v \in V$ existiert mindestens ein eingehender Pfeil mit Kosten $c' = 0$.

4.6.2 Lemma

Sei T ein s-Wurzelbaum. Dann gilt: T ist c-minimal genau dann, wenn T c'-minimal ist.

Beweis:

Sei $T = (V, R_T)$. Es ist $|R_T \cap \delta^-(v)| = \begin{cases} 1 & v \neq s \\ 0 & v = s \end{cases}$.

$$c'(T) = \sum_{r \in R_T} c'(r) = \sum_{v \in V} \sum_{r \in \delta^-(V)} \underbrace{c'(r)}_{=c(r)-z(v)} = c(T) - \underbrace{\sum_{v \in V} z(v)}_{\text{unabhängig von T}}$$

⇒Behauptung. □

4.6.3 Algorithmus von Edmonds

Betrachte den Graphen $G_0 = (V, \{r \in R | c'(r) = 0\})$.

1. Falls $E_{G_0}(s) = V$:
   (a) Bestimme einen s-Wurzelbaum T_0 in G_0 (lineare Zeit)
   (b) T_0 ist c'-minimal (wegen $c' \geq 0$)
   (c) T_0 ist c-minimal (wegen Lemma)
2. Andernfalls: es ist $E_{G_0} \neq V$.

(a) Es gibt mindestens eine starke Zusammenhangskomponente $K \subset V$, die nicht erreichbar ist, d.h. $s \notin K$ und $|\delta^-_{G_0}(K)| = 0$ bzw. $c'(r) > 0$ für alle $r \in \delta^-(K)$.

(b) Setze erneut: $z_K := min\left\{c'(r) | r \in \delta^-_{G_0}(K)\right\}$. $c'' := \begin{cases} c'(r) - z_K & r \in \delta^-(K) \\ c'(r) & sonst \end{cases}$

(c) Folge:

i. $c' \geq c'' \geq 0$

ii. für K gibt es mindestens einem Pfeil $r \in \delta^-(K)$ mit $c''(r) = 0$

(d) Bestimme rekursiv einen c''-minimalen Wurzelbaum T'.

Für jeden nach K einlaufenden Pfeil werden c'-Kosten $z_K > 0$ fällig, also sorge dafür, dass wir nur einen solchen haben.
Idee: ersetze die eingehenden Pfeile durch jeweils einen passenden inneren Pfeil, der $c'' = c' = 0$ aufweist.
Folge: der umgebaute Baum enthält nur noch einen Pfeil aus $\delta^-(K)$ und ist damit c'-minimal und damit auch c-minimal.

4.6.4 Satz

Der Algorithmus von Edmonds berechnet einen s-Wurzelbaum minimalen Gewichts in Zeit $O(m \cdot n)$.

5 Suchstrategien

5.1 Tiefensuche

Exploration eines gerichteten Graphen "in die Tiefe". Beim Erforschen eines Pfeiles (u, v) wird zunächst rekursiv weiter von v aus exploriert, bevor die Kontrolle wieder zu u zurückkehrt.

5.1.1 Hilfsmittel für die Darstellung des Algorithmus und der Beweise

- Eckenfarben $F(v)$
- eindeutige Zeitstempel $\in \mathbb{N}$
  - $d(v)$ discovery time
  - $f(v)$ finishing time

Lebenszyklus einer Ecke:

$$\begin{array}{ccccc} \text{weiß} & \rightarrow & grau & \rightarrow & schwarz \\ unberuehrt & d(v) & entdeckt & f(v) & abgearbeitet \end{array}$$

$I(v) := [d(v), f(v)]$
Vorgängerzeiger: Wenn beim Erforschen von $\delta^+(u)$ eine Ecke v neu entdeckt wird, so merke die Vorgängerschaft:

$$\pi(v) := u$$

Vorgängergraph:

$$G_\pi := (V, \{(\pi(v), v) \,|v \in V, \pi(v) \neq \perp\})$$

5.1.2 Algorithmus Tiefensuche

Hauptroutine DFS(G)

- for each $v \in V$:
  - initialisiere Farbe $F(v) \leftarrow$ weiß und Vorgänger $\pi(v) \leftarrow \perp$
  - setze Zeit $t \leftarrow 0$
- for each $u \in V$:
  - if $F(u) =$ weiß then DFS_VISIT(u)

Unterroutine DFS_VISIT(u)

- Setze Farbe $F(u) \leftarrow grau$
- Setze Zeitstempel $d(u) \leftarrow t++$
- for each $(u, v) \in \delta^+(u)$:
  - if $F(v) =$ weiß
    - setze Vorgänger $\pi(v) := u$
    - DFS_VISIT(v)
- Setze Farbe $F(u) \leftarrow schwarz$
- Setze Zeitstempel $f(u) \leftarrow t++$

5.1.3 Satz

Der Vorgängergraph G_π ist ein Wald. Jede schwache Zusammenhangskomponente von G_π ist ein Wurzelbaum.

Beweis

- Kreisfreiheit : Für jeden Pfeil $r \in G_\pi$ gilt: er wird von einer grauen zu einer weißen Ecke gezogen, d.h. $(*)d(\alpha(r)) < d(\omega(r))$. Außerdem werden die Vorgänger nur höchstens einmal gesetzt. Ein Kreis in G_π könnte die Bedingung $(*)$ nicht an allen Stellen gewährleisten.
- Wurzelbaum: Per Induktion ist $g^-(v) \leq 1$. Außerdem sind alle Ecken eines Teilgraphen von der Startecke aus erreichbar, da Pfeile nur gemäß ihrer Richtung durchlaufen werden. $\rightarrow$ Behauptung.

5.1.4 Satz (Intervallsatz)

Für $u \neq v$ mit $d(u) < d(v)$ gilt genau eine der Aussagen:

1. $I(u) \cap I(v) = \emptyset$
2. $I(v) \subset I(u)$ und v ist Nachfahre von u in G_π

Beweis

Für $f(u)$ haben wir zwei Fälle

- $d(u) < f(u) < d(v)(< f(v)) \rightarrow$ die Intervalle liegen disjunkt.
- $d(u) < d(v) < f(u)$ Zum Zeitpunkt $d(v)$ ist u grau und damit wird v Nachfahre von u in G_π. Die Kontrolle kehrt von v erst dann zurück, wenn v abgearbeitet und schwarz gefärbt ist, also $f(v) < f(u)$, d.h. $I(v) \subset I(u)$. $\square$

5.1.5 Satz (vom weißen Weg)

Eine Ecke v wird Nachfahre von u im DFS-Wald G_π genau dann, wenn zum Zeitpunkt $d(u)$ die Ecke v über einen Weg erreicht werden kann, der nur weiße Ecken benutzt.

Beweis: siehe Übung

Induktion

5.1.6 Satz (vom grauen Weg)

Zum Zeitpunkt $d(v)$ bildet die Menge aller grauen Ecken einen Weg. Dieser beginnt an der Wurzel, mit der im DFS der aktuelle Baum gestartet wurde, und er endet in v.

Beweis: siehe Übung

(Induktion, Widerspruchsbeweis)

5.1.7 Korollar

Alle Ecken einer starken Zusammenhangskomponente erscheinen im selben Baum von G_π.

Klassifikation von Pfeilen aus G

- tree edge : Pfeil aus G_π
- back edge : Pfeil (u, v) mit: u ist Nachfahre von v in G_π
- forward edge: Pfeil (u, v) mit: v ist Nachfahre von u in G_π
- cross edge : alle übrigen

Während des Algorithmus klassifizieren: $r = (u, v)$ zum Zeitpunkt $d(u)$

- Farbe von v ist weiß: $\Leftrightarrow$ r ist tree edge
- Farbe von v ist grau: $\Leftrightarrow$ r ist back edge
- Farbe von v ist schwarz: $\Leftrightarrow$ r ist cross edge oder forward edge (vgl. Übung)

5.2 Anwendungen der Tiefensuche

5.2.1 Test auf Kreise

Satz DFS produziert back edge $\Leftrightarrow$ der Graph enthält Kreise

Beweis " $\Rightarrow$ " trivial
" $\Leftarrow$ " Sei $(v_0, ..., v_k, v_0)$ die Spur des Kreises. o.E. sei v_0 die zuerst entdeckte Ecke. Nach dem Satz vom weißen Weg werden alle v_i Nachfahren von v_0 in $G_\pi \rightarrow (v_k, v_0)$ ist back edge.

Korollar Test auf Kreise und Ausgabe eines solchen Kreises sind mit DFS in Zeit $O(m+n)$ möglich.

5.2.2 Topologische Sortierung

Satz Ist G kreisfrei, so gilt $f(\alpha(r)) > f(\omega(r))$ für alle $r \in R$.

Beweis Betrachte die Erforschung von $r = (u, v)$, Zeitpunkt $d(u)$. Farbe von v:

- grau: dann ist r back edge, damit ist G nicht kreisfrei. Widerspruch!
- weiß: Nach dem Satz vom weißen Weg wird v Nachfahre von u $\rightarrow$ Intervallsatz: $I(v) \subset I(u)$, also $f(v) < f(u)$.
- schwarz: da u noch grau ist, gilt $f(u) > f(v)$ sowieso.

Korollar DFS konstituiert eine topologische Sortierung, falls eine existiert.

Beweis setze $\forall v \in V : \sigma(v) := n - f(v)$

5.2.3 Bestimmung von starken Zusammenhangskomponenten

Algorithmus:

1. rufe DFS(G) auf. Merke den Zeitstempel $f()$
2. rufe DFS(G^{-1}) auf. Dabei sollen in der Hauptroutine DFS die Ecken gemäß absteigenden Werten $f()$ probiert werden.
3. Rückgabe: die Bäume aus Phase 2 bilden die starken Zusammenhangskomponenten.

cross edges laufen nur von rechts nach links. Für starken Zusammenhang: $v \in E(v_p)$, $v_p \in E(v) \Leftrightarrow E_{G^{-1}}(v_p)$. Cross edge in G^{-1}: Laufen von links nach rechts, d.h. $E_{G^{-1}}(v_p)$ liegt ganz in T_p. $\rightarrow$ die starke Zusammenhangskomponente von v_p ist ein Teilbaum von T_p.

Satz Der Algorithmus findet in der Zeit $O(m+n)$ die starken Zusammenhangskomponenten.

Beweis Seien $T_1, ..., T_p$ die Bäume aus Phase 1, $v_1, ..., v_p$ ihre Wurzeln und sortiert, so dass $f(v_1) < ... < f(v_p)$. Induktion nach Zahl der Zusammenhangskomponenten:

1. Der Graph ist stark zusammenhängend, dann gibt es als Ergebnis jeweils nur einen Baum.
2. Der Graph hat mehr als eine starke Zusammenhangskomponente. Es gibt für $i < j$ keine cross edge von T_i nach T_j. In Phase 2 wird der erste Baum mit v_p aufgerufen. Sei v im ersten Baum: $v \in E_{G^{-1}}(v_p)$, also $v_p \in E_G(v)$. Da es keine Pfeile nach rechts gibt, folgt $v \in T_p$, also $v \in E_G(v_p)$. Somit $v \sim v_p$.
   Sei v nicht im ersten Baum. Also $v \notin E_{G^{-1}}(v_p)$ und $v \not\sim v_p$.
   Betrachte $G' := G - ZK(v_p)$, der eine Zusammenhangskomponente weniger hat. Der Baum T_p zerfällt durch $T_p - ZK(v_p)$ in Teilbäume $U_1, ..., U_k$ mit Wurzeln $u_1, ..., u_k$, die wieder sortiert werden können, sodass $f(v_1) < ... < f(v_{p-1}) < f(u_1) < ... < f(u_k)$. Damit ist die Induktionsvoraussetzung anwendbar. $\square$

6 Weglängen

$G = (V, R), G = (V, E)$. $c : E \to \mathbb{R}$ Länge der Kante.

6.1 Metrischer Graph

6.1.1 Definition (Länge eines Weges)

Die Länge des Weges $p = (e_1, ..., e_k)$ is $c(p) = \sum_i c(e_i)$.

6.1.2 Definition (Distanz)

Die Distanz zweier Knoten wir beschrieben als $dist(u, v) := inf\,\{c(p)|\text{p ist ein u-v-Weg}\}$.

Bemerkung: falls $c > 0$ gilt:

- $dist \geq 0$
- $dist(u, v) = 0 \Leftrightarrow u = v$
- $v \notin E(a) \Leftrightarrow dist(u, v) = +\infty$
- Dreiecksungleichung: $dist(u, w) \leq dist(u, v) + dist(v, w)$ für alle $u, v, w \in V$
- Symmetrie: $dist(u, v) = dist(v, u)$ für alle $u, v \in V$

dist erfüllt also die Axiome einer Metrik.

Metrischer Graph:

ausgehend von $G = (V, E)$ mit $c > 0$ bilde die transitive Hülle G^* und setze dort $\widehat{c}(u, v) := dist(u, v)$.
Gerichteter Graph: gleiche Konstruktion nützlich, aber ggf. gilt die Symmetrie nicht mehr.
negative Kantenlängen: c beliebig. Es ist $dist(u, v) = -\infty \Leftrightarrow$ auf dem Weg von u nach v liegt eine Ecke, die Teil eines negativen Kreises ist.

6.2 Baum kürzester Wege

Ab jetzt: gerichteter Graph.

6.2.1 Definition (Baum kürzester Wege bzgl. s)

Sei $G = (V, R)$, $c : R \to \mathbb{R}$, $s \in V$. Ein Baum kürzester Wege bzgl. s (kurz: s-SPT, shortest path tree) ist

- ein s-Wurzelbaum mit
- $dist_T(s, v) = dist_G(s, v)$ für alle $v \in V$.

Im Allgemeinen ist der s-SPT nicht eindeutig.

6.2.2 Satz (Existenz eines s-SPT)

Ein s-SPT existiert genau dann, wenn von s aus kein negativer Kreis erreichbar ist.

Beweis:

1. Sei kein negativer Kreis erreichbar.

- für alle $v \in V$ ermittle einen kürzesten s-v-Weg. Die Vereinigung aller dieser Wege ergibt einen Teilgraphen T, der $dist_T(s, v) = dist_G(s, v)$.
- Solange ein $v \in V$ existiert mit $g^-(v) > 1$: entferne aus $\delta^-(v)$ alle bis auf einen Pfeil. Distanzen bleiben erhalten, weil Teilweg eines kürzesten Weges wieder ein kürzester Weg ist.
- ebenso falls $g^-(s) > 0$ (hier ist nur Kreis der Länge 0 möglich)
- Resultat: s-Wurzelbaum, dieser ist auch ein s-SPT

2. es sei ein negativer Kreis erreichbar. Dann gibt es Ecken v mit $dist(s,v) = -\infty$. Dies kann im Baum (kreisfrei!) nicht dargestellt werden. □

6.2.3 Lemma

Es gilt $dist(s,v) \leq dist(s,u) + c(u,v)$ für alle Pfeile $(u,v) \in R$.

Ziel: Eckenbewertung $d : V \to \mathbb{R}$ mit $d(v) = dist(s,v)$.

Hilfsroutine Initialisierung:

- $d(v) \leftarrow +\infty \; v \neq s$
- $d(s) \leftarrow 0$
- ferner $\Pi(v) \leftarrow \perp \; v \in V$ (Vorgängergraph G_π,vgl. DFS)

RELAX(u,v) (aufgerufen für Pfeile (u,v) aus R)

- if $(d(v) > d(u) + c(u,v)$
  - $d(v) \leftarrow d(u) + c(u,v)$
  - $\pi(v) \leftarrow u$

6.2.4 Satz (Eigenschaften von Relax)

Nach der Initialisierung werde RELAX(u,v) für beliebig viele Pfeile aufgerufen. Dann gilt stets:

1. $d(v) \geq d(u) + c(u,v)$ für alle $(u,v) \in G_\pi$
2. $d(v) \geq dist(s,v)$ für alle $v \in V$
3. alle Kreise in G_π haben negative Länge.
4. wenn von s aus kein negativer Kreis erreichbar ist, dann gilt:
   (a) G_π ist ein s-Wurzelbaum und
   (b) $dist_{G_\pi}(s,v) \leq d(v)$ für alle $v \in V$.

Beweis:

1. Induktion nach Zahl der Ausführungen von RELAX:
   (a) IA: Klar.
   (b) Betrachte RELAX(u,v): $d(v)$ erniedrigt sich, alle anderen Labels bleiben unverändert. Die Ungleichung $d(v') \geq d(u') + c(u',v')$ kann überhaupt nur verletzt werden, also für Pfeile mit $\omega(r) = v$. Davon gibt es nur einen in G_π , nämlich den aktuellen (u,v). Für diesen wird die Ungleichung nach Konstruktion mit Gleichheit erfüllt.
2. Wenn $d(v)$ neu gesetzt wird, dann gilt $d(v) = d(u) + c(u,v) \geq dist(s,u) + c(u,v) = dist(s,v)$.
3. Sei $C = (c_1, ..., c_k)$ ein Kreis in G_π. Ohne Einschränkung sei $r_k = (u,v)$ der Pfeil, der C geschlossen hat. Dann galt vorher $d(v) > d(u) + c(u,v)$ und wegen 1. gilt $d(v_i) \geq d(u_i) + c(u_i, v_i)$ für alle $r_i = (u_i, v_i)$. Damit ist $c(C) = \sum_i c(r_i) < \sum_i (d(\omega(r_i)) - (\alpha(r_i))) = 0$.
4. zu Zeigen: falls kein negativer Kreis erreichbar ist, dann ist G_π ein s-Wurzelbaum und $dist_{G_\pi}(s,v) \leq d(v)$. Wegen 3. gitl: G_π ist kreisfrei. Nach Konstruktion ist $g^-(v) \leq 1$ und außerdem ist $g^-(s) = 0$, da s nur über Kreis angefahren werden könnte. $d(v) \overbrace{=}^{RELAX} d(u) + c(u,v) \overbrace{=}^{IV} dist_{G_\pi}(s,u) + \underbrace{c(u,v)}_{\in G_\pi} = dist_{G_\pi}(s,v)$.

6.2.5 Korollar

Wenn wir (wie auch immer) erreichen, dass $d(v) \leq dist(s,v)$, dann gilt tatsächlich Gleichheit und G_π ist ein gerichteter s-SPT.

6.3 Algorithmus von Dijkstra

Berechnung eines Baums kürzester Wege bzgl. $s \in V$.
Voraussetzung:

1. $c \geq 0$
2. $E(s) = V$

Algorithmus: Eingabe: $G = (V, R)$, $c : R \to \mathbb{R}_0^+$, Wurzel $s \in V$.

1. initialisiere d, π
2. setze $S \leftarrow \emptyset$
3. while $S \neq V$
   (a) wähle $u \in V - S$ mit minimalem Wert $d(u)$.
   (b) Setze $S \leftarrow S \cup \{u\}$
   (c) for all $r = (u, v) \in \delta^+(u)$
      i. RELAX(u,v)
4. return G_π und d.

6.3.1 Satz

Falls $c \geq 0$, ermittelt der Algorithmus von Dijkstra einen $s - SPT$.

Beweis:

1. Behauptung: bei Terminierung gilt $d(v) \leq dist(s,v)$ für alle $v \in V$.
   (a) Induktion nach $|S|$.
   (b) IA: $S = \emptyset$.
   (c) $|S| \to |S| + 1$: betrachte Auswahl $u \in V - S$. Widerspruchsannahme: $d(u) > dist(s,u)$. Betrachte den kürzesten s-u-Weg mit der Spurt $(u_0 = s, u_1, ..., u_k = u)$. Wähle i minimal, so dass $u_i \notin S$. Es ist $i > 0$. Als u_{i-1} zu S hinzugefügt wurde, wurde unter anderem auch RELAX(u_{i-1}, u_i) aufgerufen. Anschließend gilt: $d(u_i) = d(u_{i-1}) + c(u_{i-1}, u_i) \overbrace{\leq}^{IV} dist(s, u_{i-1}) + c(u_{i-1}, u_i) \overbrace{=}^{Teilweg} dist(s, u_i) \overbrace{\leq}^{c \geq 0} d(u)$. Widerspruch zur Wahl von u mit minimalem d-Wert.
2. Da $E(s) = V$ terminiert der Algorithmus mit $S = V$.
3. Mit Korollar 1.2.5 folgt die Korrektheit.

Implementierung: mit einer Prioritätswarteschlange zur Verwaltung der Knoten in $V - S$:

- INSERT(v,d)
- EXTRACTMIN()
- DECREASEKEY(v,d')

geeignet ist hierfür zum Beispiel ein Fibonacci-Heap → Laufzeit $O(m + n, log\, n)$ für Dijkstra.
Negative Kantenlängen?

6.4 Breitensuche

ergibt sich mit $c \equiv 1$ als Spezialfall von Dijkstra (Distanz=Zahl der Pfeile).

Breitensuche setze Abstand $d(v) \leftarrow +\infty$ $v \in V$
$S \leftarrow \{s\}$ $d(s) \leftarrow 0$.
while $S \neq \emptyset$
$u \leftarrow DEQUEUE(S)$
for $v \in N^+(u)$
if $d(v) = \infty$
setze $d(v) \leftarrow d(u) + 1$
ENQUEUE(v)
return d.
Laufzeit $O(m + n)$.

Bemerkung:

Falls BFS (Breadth-First Search) nur zur Erforschung des Graphen eingesetzt wird, reicht anstelle der Verwaltung von $d()$ auch ein Bit pro Ecke ("Ecke ist bereits besucht").

6.5 Algorithmus von Bellman und Ford

Der Algorithmus von Bellman und Ford berechnet kürzeste Wege von $s \in V$ aus, ohne $c \geq 0$ vorauszusetzen.

Algorithmus benutzt:

- RELAX(u,v)
- if $d(v) > d(u) + c(u, v)$
- then $d(v) := d(u) + c(u, v)$
- $\pi(v) := u$

6.5.1 Algorithmus:

- initialisiere d, π
- for $k = 1, ..., n - 1$ (Phasenzähler)
  - for each $r = (u, v) \in R$
    - RELAX(u,v)
- for each $r = (u, v) \in R$
  - if $d(v) > d(u) + c(u, v)$
    - return "es ist ein negativer Kreis von s aus erreichbar."
- return G_π und $d()$ (SPT ist gefunden)

6.5.2 Lemma

Am Ende der Phase k ($k = 0, ..., n - 1$) gilt:

$$d(v) \leq inf\,\{c(p)|\text{p ist ein s-v-Weg mit höchstens k vielen Pfeilen.}\}$$

Beweis per vollständige Induktion (Übung)

6.5.3 Korollar

Der Algorithmus von Bellman und Ford berechnet in $O(m \cdot n)$ einen s-SPT oder gibt die Information, dass ein solcher nicht existiert.

Beweis:

1. Wenn kein negativer Kreis von s aus erreichbar ist, dann sind kürzeste s-v-Wege ohne Einschränkung elementar, hat also höchstens n-1 Pfeile und ist nach obigem Lemma nach Phase $n-1$ gefunden.

2. Sei $C = (r_1, ..., r_k)$ ein negativer Kreis. Betrachte die Eckenbewertung $d()$ am Ende der Phase $n-1$:

   (a)
   $$0 > c(C) = \sum_i c(r_i) = \sum_i (c(r_i) + d(\omega(r_i)) - d(\alpha(r_i)))$$

   Also muss für mindestens ein i gelten:

   $$c(r_i) < d(\omega(r_i)) - d(\alpha(r_i))$$

   und dies führt in der n-ten Phase zum Abbruch des Algorithmus.

Laufzeit:

- n Phasen
- in jeder Phase m Tests. □

Bemerkung: Falls ein negativer Kreis vorhanden ist, muss dieser bei Abbruch nicht notwendigerweise in G_π enthalten sein. (vlg. Übung)

7 Flüsse

Alle Graphen in diesem Kapitel sind gerichtet.

7.1 Einführung

7.1.1 Definition (Fluss)

Sei $G = (V, R)$. $u : R \mapsto \mathbb{R}_0^+$ ist die Kapazitätsfunktion, $s, t \in V$ sind zwei ausgezeichnete Ecken, Quelle s und Senke t und $x : R \mapsto \mathbb{R}_0^+$ ist der Flusswert auf einem Pfeil.
Für $v \in V$ definieren wir $\beta^-(v) := \sum_{r \in \delta^-(v)} x(r)$ als Zufluss sowie $\beta^+(v) := \sum_{r \in \delta^+(v)} x(r)$ als Abfluss. $\beta(v) := \beta^-(v) - \beta^+(v)$ ist der Überfluss.
x heißt s-t-Fluss, wenn $\beta(v) = 0$ für alle $v \in V \setminus \{s, t\}$ gilt.
Der Wert des Flusses ist $F(x) := \beta(t)$.
Der Fluss heißt zulässig, wenn $0 \leq x \leq u$ (gilt pfeilweise).

7.1.2 Lemma

Sei x der Fluss und $S \subseteq V$. Dann ist $\beta(S) = \sum_{v \in S} \beta(v)$.

Beweis:

$$\begin{aligned} \sum_{v \in S} \beta(v) = & \quad \sum_{v \in S} \left(\sum_{r \in \delta^-(v)} x(r) - \sum_{r \in \delta^+(v)} x(r) \right) & = \\ = & \quad \sum_{r \in \delta^-(S)} x(r) - \sum_{r \in \delta^+(S)} x(r) & = \\ = & \quad \beta^-(S) - \beta^+(S) & \square \end{aligned}$$

Beobachtung: Sei x ein Fluss, $S \subseteq V$, $\beta^+(S) \geq 0$, $\beta^-(S) \geq 0$. Dann ist $\beta(S) = -\beta(V - S)$.

Erinnerung: Eine Teilmenge $S \subseteq V$ mit $V \neq S \neq \emptyset$ heißt Schnitt. Ein Schnitt S heißt s-t-Schnitt, wenn er die Quelle enthält, aber nicht die Senke.

7.1.3 Lemma

Sei x ein s-t-Fluss und S ein beliebiger s-t-Schnitt. Dann gilt: $F(x) = -\beta(S)$.

Beweis:

$$F(x) = \beta(t) = \sum_{v \in V - S} \beta(v) = \beta(V - S) = -\beta(S)$$

Folge: $\beta(s) = -\beta(t)$

7.1.4 Problemstellung (Maximaler Fluss)

Gegeben: Graph G mit Kapazitäten u, Quelle s, Senke t.
Gesucht: zulässiger s-t-Fluss von maximalem Flusswert.

7.2 Residualgraph

7.2.1 Definition (Residualgraph, Restgraph)

Sei x ein zulässiger Fluss in $G = (V, R)$ mit Kapazitäten u. Der Residualgraph G_x hat die gleiche Eckenmenge V. Für jeden Pfeil $r \in R$ hat G_x

- einen gleichgerichteten Pfeil r^+ mit Kapazität $u_x(r^+) := u(r) - x(r)$, sofern dieser Wert positiv ist (sonst gibt es diesen Pfeil nicht!)
- einen entgegengesetzt gerichteten Pfeil r^- mit der Kapazität $u_x(r^-) := x(r)$, sofern dieser Wert > 0 ist.

Bemerkung: für $x \equiv 0$ ist $G_x = G$.

Bemerkung: falls G antiparallele Pfeile enthält, so kann der Residualgraph Parallelen enthalten.

7.2.2 Definition (flussvergrößernder Weg)

Ein s-t-Weg p in G_x heißt flussvergrößernder Weg. Seine Residualkapazität ist gegeben durch $min_{r \in p} u_x(r)$.

Bemerkung: Die Residualkapazität ist stets > 0.

Beobachtung: Wenn ein flussvergrößernder Weg in G_x besteht, dann ist x kein maximaler Fluss.

7.3 Das Max-Flow-Min-Cut-Theorem

7.3.1 Definition

Sei S ein s-t-Schnitt. Seine Kapazität ist gegeben durch $u(S) := u(\delta^+(S)) = \sum_{r \in \delta^+(S)} u(r)$.

7.3.2 Lemma

Sei S ein s-t-Schnitt und x ein zulässiger s-t-Fluss. Dann gilt: $F(x) \leq u(S)$.

Beweis:

$$u(S) = \sum_{r \in \delta^+(S)} u(r) \overbrace{\geq}^{zul.} \sum_{r \in \delta^+(S)} x(r) = \beta^+(S) \geq \beta^+(S) - \beta^-(S) = -\beta(S) = F(x)$$

7.3.3 Korollar

$max\,\{F(x)|\text{x ist zulässiger s-t-Fluss}\} \leq min\,\{u(S)|\text{S ist s-t-Schnitt}\}$.