1 Graphen

Definition 1.1 (Ungerichteter Graph)**.** Ein **ungerichteter Graph** G ist ein Tripel (V, E, Ψ), für das

1. V und E endliche Mengen sind und
   - V ist die Knotenmenge
   - E ist die Kantenmenge
2. Ψ eine Funktion mit

$$\Psi : E \to \{X \subseteq V | 1 \leq |X| \leq 2\}$$

ist.

- Zwei Kanten e, e' sind **parallel**, wenn $\Psi(e) = \Psi(e')$.
- Eine Kante e ist eine **Schleife**, falls $|\Psi(e)| = 1$.
- Ein Graph ohne parallele Kanten und Schleifen heißt **einfach**. Man schreibt dann $G = (V, E)$ mit $E(G)$ der Kantenmenge von G.
  ↪ In einem einfachen Graphen kann man $\{v_i, v_j\}$ für eine Kante $e_{i,j}$ zwischen v_i und v_j schreiben.
- $|E|$ ist die Zahl der Kanten und $|V|$ die Zahl der Knoten von G.
  ↪ Oft verwendet man n für $|V|$ und m für $|E|$.

(a) parallele Kanten

(b) Schleife

Definition 1.2 (Nachbarschaft in Graphen)**.** Eine Kante $e = \{v, w\}$ in einem Graphen verbindet zwei Knoten v und w. Der Knoten v ist dann Nachbar von w, d.h. v und w sind **adjazent** ("benachbart"). Außerdem ist sowohl v als auch w **inzident** ("zusammentreffend mit") zu e.

Definition 1.3 (Knotengrad)**.** Der **Grad** eines Knotens v ist die Anzahl der inzidenten Kanten und wird bezeichnet mit $deg(v)$ (auch $\delta(v)$).

Ein Knoten v heißt **isoliert**, wenn $deg(v) = 0$.

Satz 1.4. Sei $G = (V, E)$ ein Graph. Dann gilt

$$\sum_{v \in V} deg(v) = 2|E|.$$

D.h. die Summe über alle Knotengrade ist gleich 2-mal die Anzahl der Kanten.

Lemma 1.5 (Handshake-Lemma)**.** In jedem Graphen ist die Anzahl der Knoten mit ungeradem Grad gerade.

Definition 1.6 (Teilgraph)**.** Ein Teilgraph $H = (V(H), E(H))$ eines Graphen $G = (V(G), E(G))$ ist ein Graph mit

$$\begin{aligned} V(H) &\subseteq V(G), \\ E(H) &\subseteq E(G). \end{aligned}$$

H ist **aufspannend**, wenn $V(H) = V(G)$ gilt.

Definition 1.7 (Isomorph)**.** Seien $G = (V(G), E(G))$ und $H = (V(H), E(H))$ zwei Graphen. Wenn es eine bijektive Abbildung $f : V(G) \to V(H)$ gibt, sodass $\{v, w\} \in E(G)$ gilt, genau dann, wenn $\{f(v), f(w)\} \in E(H)$ gilt, dann nennen wir die Graphen **äquivalent** (oder **isomorph**).

1.1 Datenstrukturen für Graphen

Definition 1.8 (Adjazenzmatrix)**.** Eine **Adjazenzmatrix** $A = (a_{vw})$ eines Graphen $G = (V, E)$ beschreibt, welche Knoten v_i zu welchen Knoten v_j adjazent sind ($v_i, v_j \in V$). Die Matrix hat die Dimension $n \times n$, wobei $n = |V|$.
Es gilt $A \in \{0, 1\}^{n \times n}$ mit

$$a_{vw} := \begin{cases} 1 \text{ für } \{v, w\} \in E \\ 0 \text{ sonst} \end{cases}$$

Eine Adjazenzmatrix ist quadratisch und symmetrisch. Für einfache Graphen enthält die Diagonale ausschließlich Nullen.

Definition 1.9 (Inzidenzmatrix)**.** Eine **Inzidenzmatrix** $A = (a_{ve})$ eines Graphen $G = (V, E)$ beschreibt, welche Knoten $v_i \in V$ zu welchen Kanten $e_j \in E$ inzident sind. Die Matrix hat die Dimension $n \times m$, wobei $n = |V|$ und $m = |E|$.
Es gilt $A \in \{0, 1\}^{n \times m}$ mit

$$a_{ve} := \begin{cases} 1 \text{ für } v \in e \\ 0 \text{ sonst} \end{cases}$$

Für große Graphen enthält die Matrix tendenziell viele Nullen.

Definition 1.10 (Kantenliste)**.** Eine **Kantenliste** eines Graphen $G = (V, E)$ ist eine Liste aus zweielementigen Knotenmengen $\{v_i, v_j\} \in E$, die eine Kante von v_i zu v_j beschreiben $(v_i, v_j \in V)$.
Die Kantenliste benötigt $\Theta(m\ log(n))$ Speicherplatz und ist damit sparsamer als die Inzidenzmatrix (für $n \geq 8$).

Definition 1.11 (Adjazenzliste)**.** Eine **Adjazenzliste** eines Graphen $G = (V, E)$ gibt zu jedem Knoten v_i die benachbarten Knoten v_j an $(v_i, v_j \in V)$.
Das ist etwas praktischer als die Kantenliste, wenn man für Graphenalgorithmen direkten Zugriff auf die Nachbarn eines Knotens benötigt. Man muss nicht die Nachbarn erst mühsam aus einer Liste heraussuchen.
Die Adjazenzliste benötigt $\Theta(n\ log(n) + m\ log(n))$ Speicherplatz. Im Allgemeinen sind Graphen mit vielen isolierten Knoten (ohne Kanten) uninteressant, d.h z.B. $m \geq \frac{n}{2}$ oder $m \geq n$ o.ä. Also wieder $\Theta(m\ log(n))$.

1.2 Wege und Pfade

Definition 1.12 (Kantenfolge)**.** Eine **Kantenfolge** W in einem Graphen $G = (V, E)$ ist eine Folge $v_1, e_1, v_2, e_2, v_3, ..., e_k, v_{k+1}$ mit $k \geq 0,\ e_i = \{v_i, v_{i+1}\} \in E$

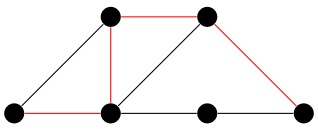


Abbildung 1.2: Kantenfolge in G

Definition 1.13 (Weg)**.** Wiederholt sich keine Kante in einer Kantenfolge, dann spricht man von einem **Weg**. Ein **geschlossener Weg** (auch Tour) kehrt am Ende zum Startknoten zurück. Ein **Eulerweg** benutzt alle Kanten eines Graphen. Eine **Eulertour** kehrt außerdem zum Startknoten zurück.

Definition 1.14 (Pfad)**.** Wiederholt sich kein Knoten in einer Kantenfolge, dann sprichtman von einem **Pfad**. Ein **Kreis** ist ein geschlossener Pfad, d.h. der Pfad kehrt zum Startknoten zurück. Ein **Hamiltonpfad** besucht alle Knoten eines Graphen. Ein **Hamiltonkreis** (auch Hamiltontour) kehrt außerdem zum Startknoten zurück.

Satz 1.15. Sei G ein Graph mit Adjazenzmatrix A. Der Koeffizient $a_{vw}^{(m)}$ von A^m gibt die Anzahl der Kantenfolgen der Länge m von Knoten v zu Knoten w an. Insbesondere gilt $a_{vv}^{(2)} = deg(v)$.

Definition 1.16 (Zusammenhängend)**.** Ein Graph G heißt **zusammenhängend**, wenn es zwischen je zwei Knoten aus G einen Weg gibt.

Ein maximaler zusammenhängender Teilgraph von G heißt eine **(Zusammenhangs-) Komponente** von G.

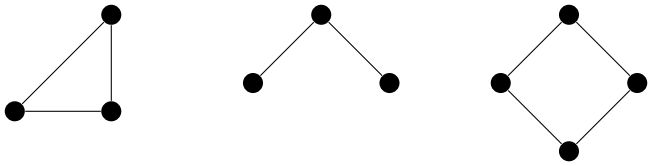


Abbildung 1.3: Graph mit 3 Zusammenhangskomponenten

Satz 1.17 (notwendiges Kriterium)**.** Ein zusammenhängender Graph G mit n Knoten muss zumindest $n-1$ Kanten haben.

Satz 1.18 (hinreichendes Kriterium)**.** Ein Graph G mit mehr als $\frac{(n-1)(n-2)}{2}$ Kanten ist zusammenhängend.

Problem 1.19 (Zusammenhang)**.**

Gegeben: Ein Graph $G = (V, E)$.

Gesucht: Ist G zusammenhängend?

Eingabe : Graph G
Ausgabe: Ist G zusammenhängend

1 Markiere einen beliebigen Startknoten.
2 Für jeden im voherigen Schritt markierten Knoten: Markiere alle noch nicht markierten adjazente Knoten. Wurde kein neuer Knoten markiert, dann 3. Sonst wiederhole 2.
3 Sind noch unmarkierte Knoten übrig **return** "G ist nicht zusammenhängend". Sonst "G ist zusammenhängend".

Algorithmus 1: Algorithmus zum Entscheiden ob ein Graph G zusammenhängend ist. Laufzeit: $O(|E|)$

Satz 1.20. Ein Graph G mit Adjazenzmatrix A ist zusammenhängend, genau dann, wenn

$$Z = I_n + A + A^2 + ... + A^{n-1} \text{ mit } z_{vw} \neq 0 \ \forall v, w \in \{1, ..., n\}$$

existiert.

1.3 Eulerwege und Hamiltonpfade

Problem 1.21 (Eulerweg)**.**

Gegeben: Ein Graph $G = (V, E)$.

Gesucht: Ein Eulerweg W in G - oder ein Argument, dass kein Eulerweg existiert.

Satz 1.22. Ein Graph $G = (V, E)$ besitzt genau dann einen Eulerweg, wenn es höchstens zwei Knoten mit ungeradem Grad gibt.

Eingabe : Graph G
Ausgabe: Ein Weg in G

Starte in einem Knoten v_0
(wenn einer mit ungeradem Grad existiert, dort, sonst beliebig)
$i \leftarrow 0$
while *es gibt eine zu* v_i *inzidente Kante* **do**
  wähle eine zu v_i inzidente, unbenutze Kante $\{v_i, v_j\}$
  laufe zum Nachbarknoten v_j
  lösche $\{v_i, v_j\}$ aus der Menge der unbenutzen Kanten
  $v_{i+1} \leftarrow v_j$
  $i \leftarrow i + 1$
end

Algorithmus 2: Algorithmus zum Finden eines Weges in einem Graphen

Satz 1.23. Wenn der Algorithmus 2 stoppt, bleibt ein eulerscher Graph zurück, d.h. ein Graph mit lauter Knoten geraden Grades.

Korollar 1.24.

1. Zwei geschlossene Wege mit einem gemeinsamen Knoten kann man in einen geschlossenen Weg verwandeln.

2. Man kann aus allen Wegen einen Weg machen, wenn der Graph zusammenhängend ist, indem man immer wieder 1. anwendet.

Eingabe : Zusammenhängender Graph G mit höchstens 2 ungeraden Knoten
Ausgabe: Ein Eulerweg, bzw. eine Eulertour in G

Starte in einem Knoten v
(wenn einer mit ungeradem Grad existiert, dort, sonst beliebig)
verwende Algorithmus 2, um einen Weg W von v aus zu bestimmen
while *es existieren unbenutzte Kanten* **do**
  wähle einen Knoten w aus W mit positivem Grad im Restgraphen
  verwende Algorithmus 2, um einen Weg W' von w aus zu bestimmen
  verschmelze W und W'
end

Algorithmus 3: Algorithmus zum Finden eines Eulerweges oder einer Eulertour

Eingabe : Zusammenhängender Graph G mit höchstens 2 ungeraden Knoten
Ausgabe: Ein Eulerweg, bzw. eine Eulertour in G

Starte in einem Knoten v_0
(wenn einer mit ungeradem Grad existiert, dort, sonst beliebig)
$i \leftarrow 0$
while *es gibt eine zu* v_i *inzidente, unbenutzte Kante* **do**
    wähle eine dieser Kanten $\{v_i, v_j\}$, die den Restgraph zshgd. lässt
    laufe zum Nachbarknoten v_j
    markiere $\{v_i, v_j\}$ als benutzt
    $v_{i+1} \leftarrow v_j$
    $i \leftarrow i + 1$
end

Algorithmus 4: Algorithmus von Fleury zum Finden eines Eulerweges oder einer Eulertour

Satz 1.25 (hinreichendes Kriterium)**.** Wenn ein Graph mit n Knoten mindestens $\frac{1}{2}(n-1)(n-2)+2$ Kanten hat, dann besitzt er einen Hamiltonkreis.

2 Minimale aufspannende Bäume

Problem 2.1 (minimaler aufspannender Baum / kürzestes zsgd.es Netzwerk)**.**

Gegeben: Ein Graph $G = (V, E)$, Gewichtfunktion

$$\begin{aligned} c : E &\to \mathbb{R}_+ \\ e &\mapsto c_e \end{aligned}$$

Gesucht: Eine Kantenmenge $F \subseteq E$ mit möglichst geringen Gesamtgewicht $\sum_{e \in F} c_e$, so dass in $T = (V, F)$ alle Knoten verbunden sind ODER entscheide, dass G nicht zshgd. ist.

1. *Welche Eigenschaften haben Lösungen?*
   $\hookrightarrow$*Struktur*
2. *Wie findet man optimale Lösungen?*
3. *Können wir algorithmische Grundideen verwenden, die auch in anderen Situationen funktionieren?*
4. *Effizienz*

Beobachtung. Ein optimale Lösung für Problem 2.1 kann kein Kreis enthalten. (Beweisskizze: Entfernt man aus einem Kreis eine Kante bleibt der Rest immer noch verbunden).

Satz 2.2. Eine optimale Lösung für Problem 2.1 ist zusammenhängend und kreisfrei, also ein Baum.

Beweis. Klar. □

Wir betrachten Eigenschaften von Bäumen. Wie viele Kanten hat ein Baum mit n Knoten? Mit jeder eingefügten Kante nimmt die Zahl der Zhk. um 1 ab (mit jeder gelöschten um 1 zu).

Beobachtung. Ein Baum mit n Knoten hat $n-1$ Kanten. ABER: Nicht jeder Graph mit $n-1$ Kanten und n Knoten ist ein Baum.

Definition 2.3. Für einen Graphen G und $X, Y \subseteq V(G)$ ist

$$\begin{aligned} E(X,Y) &= \{\{x,y\} \in E(G) | x \in X \setminus Y,\ y \in Y \setminus X\} \\ E^+(X,Y) &= \{(x,y) \in E(G) | x \in X \setminus Y,\ y \in Y \setminus X\} \end{aligned}$$

Für einen ungerichteten Graphen G und $X \subseteq V(G)$

$$\delta(X) = E(X, V(G) \setminus X)$$

Für einen gerichteten Graphen G und $X \subseteq V(G)$

$$\begin{aligned} \delta^+(X) &= E^+(X, V(G) \setminus X) \\ \delta^-(X) &= \delta^+(V(G) \setminus X) \\ \delta(X) &= \delta^+(X) \cup \delta^-(X) \end{aligned}$$

Lemma 2.4.

a) Ein ungerichteter Graph G ist zusammenhängend $\Leftrightarrow \delta(X) \neq \emptyset \; \forall \; \emptyset \neq X \subset V(G)$.

b) Sei G ein gerichteter Graph und $r \in V(G)$. Dann gibt es einen Pfad von r nach v für jeden $v \in V(G) \Leftrightarrow \delta^+(X) \neq \emptyset \; \forall \; X \subset V(G)$ mit $r \in X$.

Beweis.

a) Gibt es eine Menge $X \subset V(G)$ mit $r \in X,\ v \in V(G) \setminus X$ und $\delta(X) = \emptyset$ dann kann es keinen r-v-Pfad geben, damit ist G nicht zusammenhängend. Wenn G nicht zusammenhängend ist, dann gibt es für irgendwelche Knoten r und v keinen r-v-Pfad. Sei R die Menge der Knoten, die von r aus erreichbar sind. Es gilt $r \in R,\ v \notin R$ und $\delta(R) = \emptyset$.

b) H.A.

□

Satz 2.5 (Eigenschaften von Bäumen)**.** Sei $T = (V, F)$ ein Graph mit n Knoten. Dann sind äquivalent

a) T ist ein Baum (zshgd., kreisfrei).

b) T hat $n - 1$ Kanten und ist kreisfrei.

c) T hat $n - 1$ Kanten und ist zshgd..

d) T ist ein minimaler zusammenhängender Graph (D.h. keine Kante kann entfernt werden ohne dass der Zusammenhang verloren geht).

e) T ist ein minimaler Graph, für den es für jede Knotenmenge $\emptyset \neq X \subseteq V$ eine nach außen verbindende Kante gibt $\delta(X) \neq \emptyset$.

f) T ist ein maximaler kreisfreier Graph (D.h. keine Kante kann hinzugefügt werden ohne dass ein Kreis entsteht).

g) T enthält einen eindeutigen Pfad zwischen je zwei Knoten.

Beweis. Wir zeigen a) ⇒ g) ⇒ e) ⇒ d) ⇒ f) ⇒ b) ⇒ c) ⇒ a) (Ringschluss).
a) ⇒ g): Weil T zusammenhängend ist, muss es je einen Pfad zwischen zwei Knoten geben. Gebe es zwischen zwei Knoten mehr als einen Pfad, so enthielte die Vereinigung der Pfade einen Kreis ↯ Kreisfreiheit.
g) ⇒ e) ⇒ d): Folgt aus Lemma 2.4. a).

d) ⇒ f): Da alle Knoten bereits verbunden sind, liefert jede eingefügte Kante einen Kreis.
f) ⇒ b) ⇒ c): Ist klar, wenn man Lemma 2.6 zeigt.
c) ⇒ a): Sei T zusammenhängend mit $n-1$ Kanten. Solange Kreise in T existieren löschen wir eine der Kanten des Kreises und entfernen somit den Kreis. Angenommen wir haben k Kanten entfernt. Der entstehende Graph T' ist immer noch zusammenhängend und kreisfrei. Wir haben $p = 1$ Komponenten und $m = (n-1)-k$ Kanten $\overset{2.6}{\Rightarrow} n = m + p = n - 1 - k + 1 \Rightarrow k = 0$. □

Lemma 2.6. *W sei ein Wald mit n Knoten, m Kanten und p ZhK.. Dann gilt $n = m + p$.*

Beweis. H.A. □

2.1 Berechnung optimaler Bäume

Kruskals Algorithmus

Eingabe : Zshgd.er, ungerichteter Graph G
Kantengewichte $c : E(G) \to \mathbb{R}_+$
Ausgabe: Aufspannender Baum T minimalen Gewichts

```
1 Sortiere Kanten nach Gewicht
    c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m)
2 Setze T := (V(G), ∅)
3 for i=1 to m do
4 |   if T + e_i enthält keinen Kreis then
5 |   |   T := T + e_i
6 end
```

Algorithmus 5: Kruskals Algorithmus

Laufzeit

1. *Sortieren der Kanten* $O(m\ log(m)) = \underbrace{O(m\ log(n))}_{Da\ m \in O(n^2)}$.
2. *Initialisierung des Baumes* $O(1)$.
3. *Test auf Kreisfreiheit* $O(m\ log(n))$.

Problem 2.7.

Gegeben: G' kreisfrei ($\leq n-1$ Kanten), Kante e.

Frage: Ist $G' + e$ kreisfrei?

DFS oder BFS $\to O(n)$.
$\hookrightarrow$ *Teste das m-fach* $\to O(n \cdot m)$
***Aber** mit anderer Implementierung geht es besser.*

Satz 2.8. Kruskals Algorithmus kann so implementiert werden, dass sich eine Laufzeit von $O(m\ log(n))$ ergibt.

Beweis. Wir löschen parallele Kanten (nur die Billigste bleibt, die Anderen sind redundant). Für 3. Datenstruktur, die die ZhKs von T verwaltet. Test in 3. ob $T+e_i = \{v, w\}$ ergibt Kreis $\Leftrightarrow$ v, w liegen in derselben ZhK (Kante verbindet zwei Knoten in derselben ZhK). Für jede Komponente merken wir uns einen gerichteten Baum mit einer eindeutigen Wurzel, wobei jeder Knoten einen eindeutigen Vorgänger hat. Wenn wir $e_i = \{v, w\}$ in 3. prüfen, finden wir die Wurzel r_v und r_w zu den ZhKs von B, die v bzw. w enthalten. Zeit? Proportional zu Länge r_v-v-Pfad in B + r_w-w-Pfad in B. $\hookrightarrow O(log(n))$ (= Höhe der Bäume) (noch zu zeigen). Teste $r_v = r_w$.

- falls ja: überprüfe nächste Kante.
- falls nein: wir fügen e_i zu T hinzu und wir müssen eine Kante zu B hinzufügen.

Sei $h(r)$ die maximale Länge eines Pfades von r in B.

- falls $h(r_v) \geq h(r_w)$: füge Kante (r_v, r_w) zu B hinzu.
- sonst füge (r_w, r_v) zu B hinzu.

Änderung von $h(r_v)$?

- $h(r_v) = h(r_w) \Rightarrow$ erhöht $h(r_v)$ um eins.
- sonst: neue Wurzel hat gleichen h-Wert wie vorher.

Behauptung: Ein gerichteter Teilbaum von B mit Wurzel r enthält mindestens $2^{h(r)}$ Knoten. Beweis durch Induktion:
I:A.: $B = (V(G), \emptyset),\ h(r) = 0$. Beh. gilt. Zu zeigen: Eigenschaft bleibt erhalten, wenn wir eine Kante (x, y) zu B hinzufügen.

- wenn $h(r)$ sich nicht ändert.
- sonst: vorher hatten wir $h(r_x) = h(r_y)$. Beide ZhKs von B hatten mindestens $2^{h(r_x)}$ Knoten $\Rightarrow$ neue ZhK, verwurzelt in r_x, hat mind. $2 \cdot 2^{h(r_x)} = 2^{h(r_x)+1}$ Knoten.

$\Rightarrow h(r) \leq log(n) \Rightarrow$ log. Höhe $\Rightarrow$ Laufzeit $O(m\ log(n))$. □

Korrektheit

Wir schauen uns zwei Optimalitätsbedingungen an.

Satz 2.9. Sei (G, c) eine Instanz des Minimum Spanning Tree Problems (MST), sei T ein Spannbaum in G. Dann gilt

a) T ist optimal.

b) Für jede Kante $e = \{x, y\} \in E(G) \setminus E(T)$ hat keine Kante auf dem x-y-Pfad in T höhere Kosten als e.

c) Für jede Kante $e \in E(T)$ ist e eine billigste Kante in $\delta(V(C))$, wobei C eine ZhK von $T - e$ ist.

Beweis.
a)⇒ b): Wir nehmen an b) ist verletzt. Sei $e = \{x, y\} \in E(G) \setminus E(T)$ und f eine Kante aus dem x-y-Pfad in T mit $c(f) > c(e)$. Dann ist $(T - f) + e$ ein Spannbaum mit niedrigeren Kosten als T ↯ T optimal.
b)⇒ c): Wir nehmen an c) ist verletzt. Sei $e \in E(T)$, C eine ZhK von $T - e$ und $f = \{x, y\} \in \delta(V(C))$ mit $c(f) < c(e)$. Der x-y-Pfad in T muss eine Kante in $\delta(V(C))$ enthalten, aber die einzige Kante, die dies erfüllt ist e.
c) ⇒ a): Sei T ein Baum, der c) erfüllt. Sei T^* ein optimaler MST mit $E(T^*) \cap E(T)$ maximal. Wir zeigen $T = T^*$. Angenommen, es gibt eine Kante $e = \{x, y\} \in E(T) \setminus E(T^*)$. Sei C eine ZhK von $T-e$. T^*+e enthält einen Kreis D. Da $e \in E(D) \cap \delta(V(C))$ muss mindestens noch eine weitere Kante $f(\neq e)$ aus D zu $\delta(V(C))$ gehören. $(T^*+e)-f$ ist ein Spannbaum. T^* optimal ⇒ $c(e) \geq c(f)$. c) gilt für T ⇒ $c(f) \geq c(e)$ ⇒ $c(f) = c(e)$. ⇒ $(T^* + e) - f$ ist ein anderer optimaler MST ↯ (Widerspruch $E(T^*) \cap E(T)$ maximal (hat eine Kante mehr gemeinsam mit T)). □

Satz 2.10. Kruskals Algorithmus ist korrekt.

Beweis. Der Algorithmus konstruiert einen Spannbaum (einen maximalen kreisfreien Teilgraphen $\overset{2.5\ f)}{\Rightarrow}$ Spannbaum). Zudem wird Bedingung b) aus Satz 2.9 erfüllt ⇒ T ist optimal. □

Kruskal nutzt Optimalitätskriterium 2.9 b) - können wir auch c) nutzen?
↪ *Ja!* → *Prims Algorithmus*

Prims Algorithmus

Eingabe : Zshgd.er, ungerichteter Graph G
Kantengewichte $c : E(G) \to \mathbb{R}$
Ausgabe: Spannbaum T minimalen Gewichts

```
1 Wähle v ∈ V(G)
2 Setze T := ({v}, ∅)
3 while V(T) ≠ V(G) do
4 |   Wähle eine Kante e ∈ δ_G(V(T)) mit minimalen Gewicht
5 |   Setze T = T + e
6 end
```

Algorithmus 6: Prims Algorithmus

Satz 2.11. Prims Algorithmus ist korrekt. Die Laufzeit ist $O(n^2)$.

Beweis. Korrektheit: folgt, da Bedingung c) aus 2.9 erfüllt ist.
Laufzeit: Nicht in 3. je alle Kanten druchsehen $\rightarrow O(n^2)$. Sondern: für jeden knoten $v \in V(T)$ merken wir uns die billigste Kante ("Kandidatenkante") $e \in E(V(T), \{v\})$.

- Initialisierung der Kandidatenkanten $O(m)$
- Auswahl der billigsten Kante unter den Kandidaten $O(n)$
- Update der Kandidaten: überprüfe alle Kanten die zum zu $V(T)$ neu hinzugefügten Knoten inzident sind $\rightarrow O(n)$

WHILE-Schleife in 3. hat $n-1$ Iterationen $\Rightarrow O(n^2)$. □

Bemerkung.

1. Laufzeit $O(n^2)$ ist bestmöglich für dichte Graphen ($m \in \Omega(n^2)$) da wir jede Kante ansehen müssen
2. mit Fibonacci heaps kann $O(m + n\ log(n))$ erreicht werden [für $m \in O(n)$ noch nicht besser]
3. bester det. Algorithmus von Chazelle (2000) $O(m \cdot \underbrace{\alpha(m,n)}_{\text{inverse Ackermannfunktion}})$
4. Es existiert noch kein det. Algorithmus mit Laufzeit $O(m)$
5. Bekannt:
   - Randomisierte Algorithmen mit erwarteter Laufzeit $O(m)$
   - Planare Graphen in $O(m)$
   - n Punkte in der Ebene $O(n\ log(n))$

3 Kürzeste Wege

Problem 3.1.

Gegeben: Gerichteter Graph G, Gewichte $c : A(G) \to \mathbb{R}$, zwei ausgezeichnete Knoten s, t.

Gesucht: s-t-Pfad minimalen Gesamtgewichts.

Verschiedene Schwierigkeitsstufen

- *alle Kanten Einheitslänge $c(a) = 1$ $\forall a \in A(G)$.*
  $\hookrightarrow$ (BFS) Suche nach Pfad mit minimaler Anzahl an Kanten.
- *nicht negative Kantenlängen.*
  $\hookrightarrow$ Dijkstra.
- *ohne Kreise negativen Gesamtgewichts.*
- *allgemeine Kantengewichte.*

Definition 3.2 (Konservativ)**.** Sei G ein Graph mit Gewichten $c : E(G) \to \mathbb{R}$. Diese Kantengewichtsfunktion heißt **konservativ** wenn sie keine Kreise negativen Gesamtgewichts enthält.

3.1 Kürzeste Wege von einer Quelle

Algorithmen beruhen auf einer Beobachtung "Bellmans Prinzip".

Lemma 3.3. Sei G ein Digraph mit konservativen Kantengewichten $c : A(G) \to \mathbb{R}$, seinen s und w zwei Knoten. Ist $e = (v, w)$ die letzte Kante eines kürzesten Pfades P von s nach w, dann ist $P_{[s,v]}$ (P ohne die Kante e) ein kürzester Pfad von s nach v.

Beweis. Angenommen Q ist ein kürzerer s-v-Pfad als $P_{[s,v]} \Rightarrow c(Q) + c(e) < c(P)$.

- Falls Q w nicht enthält $\Rightarrow Q + e$ ist kürzerer s-w-Pfad als P.
- Sonst gilt für $Q_{[s,w]}$:
  $c(Q_{[s,w]}) = c(Q) + c(e) - c(Q_{[w,v]} + e) < c(P) - \underbrace{c(Q_{[w,v]} + e)}_{\text{ist ein Kreis und } c \text{ ist konservativ}}) \leq c(P)$
  $\lightning$ P ist kürzester s-w-Pfad.

□

Folgerungen:

- *Algorithmen bauen Wege schrittweise auf.*
- *Grund, dass meiste Algorithmen den kürzesten Weg von s zu allen Knoten berechnen: Kürzesten s-t-Pfad berechnet $\Rightarrow$ auch kürzesten s-v-Pfad $\forall v \in P$ + wir wissen vorher nicht welche Knoten zu P gehören.*

Wir betrachten zunächst nicht-negative Kantengewichte. Sind diese auch ganzzahlig könnten wir auch BFS nutzen, indem wir Knoten durch Einheitskanten ersetzen. Nachteil: Bläst Eingabe um exponentiellen Faktor auf. Statt

$$\Theta(\underbrace{n\ log(m)}_{inzidente\ Kanten} + \underbrace{m\ log(n)}_{Kanten} + \sum_{e\in E(G)} log(c(e))) \text{ bekommen wir}$$

$$\Theta(n'\ log(m') + m'\ log(n'))\ mit\ m' = \sum_{e\in E(G)} c(e),\ n' = n + \sum_{e\in E(G)} c(e) - 1$$

Besser: Dijkstra.

Definition 3.4. Sei G ein Digraph mit $s, v \in V(G)$. Wir definieren

- $L(v) :=$ Länge eines kürzesten s-v-Pfades
- $p(v) :=$ Vorgänger von v in einem kürzesten s-v-Pfad

(v nicht erreichbar: $L(v) = \infty,\ p(v) = \text{NIL}$).

Eingabe : Digraph G, Gewichte $c : A(G) \to \mathbb{R}_+$, Knoten $s \in V(G)$
Ausgabe: Kürzeste Wege von s zu allen $v \in V(G)$ und ihre Länge, genauer $L(v),\ p(v)$

1 Setze $L(s) = 0$
2 $L(v) = \infty\ \forall v \in V(G) \setminus \{s\}$
3 $R = \emptyset$
4 Finde einen Knoten $v \in V(G) \setminus R$ mit $L(v) = \min_{w\in V(G)\setminus R} L(w)$
5 Setze $R = R \cup \{v\}$
6 **for** $\forall w \in V(G) \setminus R$ *mit* $(v, w) \in A(G)$ **do**
7 | **if** $L(w) > L(v) + c((v, w))$ **then**
8 | | Setze $L(w) = L(v) + c((v, w))$
9 | | $p(w) = v$
10 | **end**
11 **end**
12 **if** $R \neq V(G)$ **then**
13 | **goto** 4
14 **end**

Algorithmus 7: Dijkstras Algorithmus

Satz 3.5. Dijkstras Algorithmus ist korrekt. Die Laufzeit ist $O(n^2)$.

Beweis. Wir zeigen, dass die folgenden Aussagen nach jeder Ausführung von 4. gelten

a) $\forall v \in R$ und $w \in V(G) \setminus R : L(v) \leq L(w)$.

b) $\forall v \in R : L(v)$ ist die Länge eines kürzesten s-v-Pfades in G. Wenn $L(v) < \infty$, dann existiert ein s-v-Pfad der Länge $L(v)$, dessen letzte Kante $(p(v), v)$ ist und dessen Knoten alle zu R gehören.

c) $\forall w \in V(G) \setminus R : L(w)$ ist die Länge eines kürzesten s-w-Pfades in $G[R \cup \{w\}]$. Wenn $L(w) \neq \infty$, dann $p(w) \in R$ und $L(w) = L(p(w)) + c((p(w), w))$.

Die Aussagen gelten nach 1-3. Wir müssen zeigen, dass 5 und 6 die Gültigkeit von a), b) und c) aufrecht erhalten. Sei v ein in 4 ausgewählter Knoten.
zu a): Für jedes $x \in R,\ y \in V(G) \setminus R$ gilt nach a) und der Wahl von v in 4: $L(x) \leq L(v) \leq L(y) \Rightarrow$ a) gilt weiter nach 5 und 6.
zu b): Zu zeigen: b) gilt nach 5 (dazu reicht: betrachte Knoten v). c) galt vor $5 \rightarrow$ reicht zu zeigen, dass kein s-v-Pfad in G, der irgendein Knoten in $V(G) \setminus R$ enthält, kürzer ist als $L(v)$. Wir nehmen an es gebe einen s-v-Pfad P in G, der einen knoten $w \in V(G) \setminus R$ enthält, der kürzer als $L(v)$ ist. Sei w der erste Knoten außerhalb von R, wenn wir P von s nach v durchlaufen. c) galt vor $5 \Rightarrow L(w) \leq c(P_{[s,w]})$. Kantengewichte nicht-negativ $\Rightarrow c(P_{[s,w]}) \leq c(P) < L(v) \Rightarrow L(w) < L(v)$ $\lightning$ Wahl von v in 4.
zu c): Wir zeigen, dass 5 und 6 c) aufrecht erhalten $\rightarrow$ wenn für w (wir setzen $p(w) = v$ und $L(w) = L(v) + c((v, w))$) in 6 $\rightarrow$ existiert ein s-v-Pfad in $G[R \cup \{w\}]$ der Länge $L(v) + c((v, w))$ mit letzter Kante (v, w) + wir wissen c) galt für v. Wir nehmen an, dass nach 5 und 6 ein s-v-Pfad P in $G[R \cup \{w\}]$ existiert, der kürzer als $L(w)$ für ein $w \in V(G) \setminus R$ ist. P muss v enthalten - nur w wurde hinzugefügt und sonst wäre c) schon vor 5 und 6 verletzt gewesen (wir wissen $L(w)$ wächst nicht). Sei x der Nachbar von w in P: $x \in R \overset{a)}{\Rightarrow} L(x) \leq L(v) \overset{6}{\Rightarrow} L(w) \leq L(x) + c((x, w)) \Rightarrow L(w) \leq L(x) + c((x, w)) \leq L(v) + c((x, w)) \leq c(P)$ $\lightning$ b) $\rightarrow L(v)$ ist Länge eines kürzesten s-v-Pfades und P enthält

- einen s-v-Pfad
- Kante (x, w)

Wir haben gezeigt:

- a), b) und c) gelten je in 4
- b) gilt insbesondere, wenn der Algorithmus terminiert
  $\hookrightarrow$ Ausgabe korrekt.

Laufzeit:

- n Iterationen
- je $O(n)$

$\rightarrow O(n^2)$

□

Bemerkung.

- $O(n^2)$ wieder bestmöglich für dichte Graphen.
- für "dünne" Graphen: Friedman und Tarjan (1987) haben mittels Fibonacci heaps die Laufzeit auf $O(n\ log(n) + m)$ verbessert. Beste Laufzeit für nicht-negative Gewichte.

Jetzt: konservative Gewichte.

Eingabe : Digraph G, konservative Gewichte $c : A(G) \to \mathbb{R}$, Knoten $s \in V(G)$
Ausgabe: Kürzeste Wege von s zu allen $v \in V(G)$ und ihre Länge

```
Setze L(s) = 0
     L(v) = ∞ ∀v ∈ V(G) \ {s}
for i = 1 to n − 1 do
    for jede Kante (v, w) ∈ A(G) do
        if L(w) > L(v) + c((v, w)) then
            Setze L(w) = L(v) + c((v, w))
            p(w) = v
        end
    end
end
```

Algorithmus 8: Moore-Bellman-Ford Algorithmus

Satz 3.6. Der Moore-Bellman-Ford Algorithmus ist korrekt und hat eine Laufzeit von $O(m \cdot n)$.

Beweis. Sei zu beliebigen Zeitpunkt $R := \{v \in V(G) \mid L(v) < \infty\}$, $F := \{(x, y) \in A(G) \mid x = p(y)\}$. Wir behaupten:

a) $L(y) \geq L(x) + c((x, y))\ \forall (x, y) \in F$.

b) Wenn F einen Kreis C enthält hat dieser negatives Gesamtgewicht.

c) Wenn C konservativ ist, ist (R, F) eine in s verwurzelte Arboreszenz.

zu a): $L(y) = L(x) + c((x, y))$ wenn $p(y) = x$ gesetzt wurde und $L(x)$ wird nie erhöht.
zu b): Wir nehmen an, es wird ein Kreis C in F erzeugt indem wir $p(y) = x$ setzen $\to$ vor Einfügen: $L(y) > L(x) + c((x, y)) \Rightarrow L(y) - L(x) > c((x, y))$ und wegen a) $L(w) \geq L(v) + c((v, w))\ \forall (v, w) \in A(G) \setminus \{(x, y)\} \Rightarrow L(w) - L(v) \geq c((v, w))$. Aufsummieren $\to$ Gesamtgewicht von C negativ $c((x, y)) + c((y, v_1)) + ... + c((v_l, x)) < L(y) - L(x) + L(v_1) - L(y) + ... + L(x) - L(v_l) = 0$.
zu c): C konservativ $\overset{b)}{\Rightarrow}$ F azyklisch. $x \in R \setminus \{s\} \Rightarrow p(x) \in R \Rightarrow (R, F)$ Arboreszenz die in s verwurzelt.
$L(x)$ ist die länge des s-x-Pfades in (R, F) für jeden $x \in R$ (zu bel. Zeitpunkt im Algorithmus). Wir behaupten nach k Iterationen, $L(x)$ ist höchstens so lang wie ein

kürzester s-x-Pfad mit k Kanten. Beweis durch Induktion:
Sei P ein kürzester s-x-Pfad mit höchstens k Kanten und sei (w, x) die letzte Kante von $P \Rightarrow P_{[s,w]}$ ist kürzester s-w-Pfad mit höchstens $k-1$ Kanten $\overset{I.V.}{\Rightarrow} L(w) \leq c(P_{[s,w]})$ nach $k-1$ Iterationen. In k-ter Iteration wird auch Kante (w, x) betrachtet, danach $L(x) \leq L(w) + c((w, x)) \leq c(P)$. Da kein Weg mehr als $n-1$ Kanten hat folgt aus der Behauptung die Korrektheit vom Algorithmus.
Laufzeit: offensichtlich. □

Bemerkung. Dieser Algorithmus ist der Schnellste für konservative Gewichte.

4 Netzwerkflüsse

Flüsse in Graphen. Beispiele:

- *Wasser in Leitungssystemen*
- *Verkehr in Straßensystemen*
- *Passagiere in Transportsystemen*

Eigenschaften:

- *pro Kante 2 Kenngrößen*
  - *mögliche Flusskapazität*
  - *tatsächlicher Fluss*
- *"Flusserhaltung": Bilanzierung der Flüsse an jedem Knoten ("fliest nicht mehr raus, als rein")*

Definition 4.1. Gegeben: Digraph G mit Kapazität $u : A(G) \to \mathbb{R}_+$.

1. Ein **Fluss** ist eine Funktion $f : A(G) \to \mathbb{R}_+$ mit $f(e) \leq u(e)\ \forall e \in A(G)$.
2. An einem Knoten v gilt **Flusserhaltung**, wenn
$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e).$$
3. Eine **Zirkulation** ist ein Fluss für den an jedem Knoten Flusserhaltung gilt.
4. Für ein Netzwerk (G, u, s, t) ist ein Fluss ein s-t-Fluss, wenn Flusserhaltung an allen Knoten außer s und t gilt
$$Wert(f) := \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e).$$

Problem 4.2 (Maximaler Fluss)**.**

Gegeben: Netzwerk (G, u, s, t).

Gesucht: Ein s-t-Fluss mit maximalem Wert.

Formulierung als LP (Lineares Programm):

$$\begin{aligned} max\ F\ sodass \quad & \sum_{e \in \delta^+(s)} f_e - \sum_{e \in \delta^-(s)} f_e = F, \\ & \sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 0\ \forall v \in V(G) \setminus \{s, t\}, \\ & 0 \leq f_e \leq u_e\ \forall e \in A(G). \end{aligned}$$

Also: Können wir effizient lösen ($LP \in P$).

Ziel hier: "kombinatorische"Algorithmen.

Beobachtung 4.3. Das Max Flow Problem hat immer eine optimale Lösung.

Beweis.

1. LP ist beschränkt.
2. Fluss mit $f \equiv 0$ ist immer zulässig.

□

Definition 4.4. In einem Digraph G heißt eine Kantenmenge $\delta^+(X)$ mit $s \in X$ und $t \notin X$ ein **s-t-Schnitt**. Wert des Schnittes := Summe der Kantenkapazitäten der ausgehenden Kanten.

Lemma 4.5 (Schranken durch Schnitte)**.** Für bel. $A \subseteq V(G)$ mit $s \in A$, $t \notin A$ und jedem s-t-Fluss f gilt

a) $Wert(f) = \sum_{e \in \delta^-(A)} f(e) - \sum_{e \in \delta^+(A)} f(e)$

b) $Wert(f) \leq \sum_{e \in \delta^+(A)} f(e)$

Beweis. zu a): Flusserhaltung gilt für $v \in A \setminus \{s\}$, daraus folgt

$$\begin{aligned} Wert(f) &= \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) \\ &= \sum_{v \in A} \Big(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \Big) \\ &= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e). \end{aligned}$$

zu b): Folgt mit a), wenn wir $0 \leq f(e) \leq u(e)\ \forall e \in A(G)$ nutzen. □

Was sagt Lemma 4.5:

Der Wert eines Max Flow kann die Kapazität eines minimalen s-t-Schnitts nicht überschreiben - tatsächlich gilt Gleichheit, zeigen wir später.

Grundkonzept für Algorithmen: Flüsse erhöhen oder verringern. Interpretation: Fluss verringern $\widehat{=}$ Fluss umleiten.

Definition 4.6.

1. Für ein Digraph G ist $\overleftrightarrow{G} := (V(G), A(G) \dot\cup \{\overleftarrow{e} | e \in A(G)\})$ mit $\overleftarrow{e} = (w, v)$ für $e = (v, w)$ "Rückwärtskante" von e der doppelt gewichtete Graph für G.
2. Für einen Fluss f in einem Digraph G mit Kapazität $z : A(G) \to \mathbb{R}_+$ definieren wir die **Residualkapazitäten** u_f

$$u_f(e) = \begin{cases} u(e) - f(e) & \text{für Vorwärtskante,} \\ f(e) & \text{für Rückwärtskante.} \end{cases}$$

3. Der **Residualgraph** G_t ist der Graph $(V(G), \{e \in A(\overleftrightarrow{G}) \mid u_f(e) > 0\})$.
4. Interpretation des Residualgraphen:
   - u_f auf Vorwärtskante: um wie viel wir f noch erhöhen können.
   - u_f auf Rückwärtskante: um wie viel wir f noch verringern können.
5. Gegeben ein Fluss f und ein Pfad (oder Kreis) P in G_f. Um f entlang P um γ zu augmentieren muss man
   - den Fluss für jede Vorwärtskante um γ erhöhen ($e \in A(P)$: wenn $e \in A(G)$, erhöhe $f(e)$ um γ)
   - den Fluss für jede Rückwärtskante um γ reduzieren ($e \in A(P)$: wenn $e = \overleftarrow{e_0}$ für $e_0 \in A(G)$, reduziere $f(e_0)$ um γ)
6. Ein **f-augmentierender** Pfad in einem Netzwerk (G, u, s, t) mit einem Fluss f ist ein s-t-Pfad im Residualgraph G_f.

Eingabe : Netzwerk (G, u, s, t)
Ausgabe: Ein s-t-Fluss f mit maximalem Wert

1 Setze $f(e) = 0\ \forall e \in A(G)$
2 Finde einen f-augmentierenden Pfad P
3 Falls keiner existiert: **stop**
4 Berechne $\gamma := \min_{e \in P} u_f(e)$
5 Augmentiere f entlang P um γ und **goto** 2

Algorithmus 9: Ford-Fulkerson Algorithmus

Satz 4.7. Ein s-t-Fluss ist optimal $\Leftrightarrow$ Es gibt keinen f-augmentierenden Pfad.

Also: stoppt der Algorithmus, dann ist f tatsächlich ein optimaler Fluss.

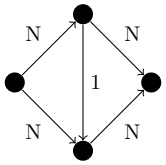
Beweis.
$\Rightarrow$: Wenn es einen f-augmentierenden Pfad gibt $\rightarrow$ 4 berechnet Fluss mit größerem Wert $\Rightarrow$ f war nicht maximal.
$\Leftarrow$: Wenn es keinen augmentierenden Pfad gibt $\rightarrow$ t ist in G_f von s nicht erreichbar. $R :=$ {Knoten in G_f , die von s erreichbar sind}, Definition von G_f:

$$f(e) = \begin{cases} u(e) & \forall e \in \delta_G^+(R), \\ 0 & \forall e \in \delta_G^-(R). \end{cases}$$

$\overset{4.5\ a)}{\Rightarrow}$ $Wert(f) = \sum_{e \in \delta^+(R)} u(e)$ $\overset{4.5\ b)}{\Rightarrow}$ f ist optimal. □

4.1 Schwierigkeit mit Ford-Fulkerson

Beispiel 4.8.



Satz 4.9 (MaxFlow-MinCut)**.** In einem Netzwerk (G, u, s, t) ist der maximale Wert eines s-t-Flusses gleich der minimalen Kapazität eines s-t-Schnittes.

Beweis. Satz 4.7 $\Rightarrow$ für jeden maximalen s-t-Fluss f gibt es einen s-t-Schnitt dessen Kapazität gleich dem Flusswert ist $\Rightarrow$ $max\, Wert(f) \geq min$ Wert eines Schnittes $\overset{4.5\ b)}{\Rightarrow}$ MaxFlow = MinCut. $Wert(f) \leq \sum_{e \in \delta^+(A)} u(e)$ □

Beobachtung 4.10. Wollen wir den Wert eines maximalen Flusses erhöhen, müssen wir am minimalen Cut die Kapazitäten verändern.

Korollar 4.11. Wenn die Kapazitäten in einem Netzwerk ganzzahlig sind, dann existiert ein optimaler ganzzahliger Fluss.

Satz 4.12. Sei (G, u, s, t) ein Netzwerk und f ein s-t-Fluss in G. Dann gibt es

- eine Familie P von s-t-Pfaden
- eine Familie C von Kreisen

so dass

$$f(e) = \sum_{p \in P \cup C : e \in E(p)} w(p),\ \forall\ e \in E(G)$$

und $|P| + |C| \leq |E(G)|$. $\sum_{p \in P} w(p) = Wert(f)$.

Außerdem: Wenn f ganzzahlig ist kann w ganzzahlig gewählt werden.

Beweis. Wir konstruieren P, C und w durch Induktion über die Anzahl der Kanten mit positivem Fluss. Sei $e = (v_0, w_0)$ eine Kante mit $f(e) > 0$. Falls $w_0 \neq t$ gibt es eine Kante (v_0, w_1) mit positivem Fluss (Flusserhaltung). Wir machen immer weiter (mit $i = 1$ startend) und finden ein w_{i+1}:
Wenn $w_i \in \{t, v_0, w_0, ..., w_{i-1}\}$ stoppen wir (Muss passieren, da endliche Knotenzahl).
Sonst finden wir Kante (w_i, w_{i+1}) mit positivem Fluss und setzen $i = i + 1$.
Das Gleiche machen wir in der anderen Richtung. Wir suchen sukzessive Vorgängerkanten:
Falls $v_0 \neq s$, gibt es eine Kante (v_1, v_0) mit positivem Fluss ... . Ende:

1) $w_i \in \{v_0, w_0, ..., w_{i-1}\}$ oder $v_j \in \{w_0, v_0, ..., v_{j-1}\}$ dann haben wir einen Kreis.
2) $w_i = t$ und $v_j = s$, dann haben wir einen s-t-Pfad.

Wir haben eine Kreis oder einen s-t-Pfad in G gefunden. UND: wir haben nur Kanten mit positivem Fluss verwendet. Sei p dieser Kreis oder Pfad. Sei $w(p) = \min_{e \in E(p)} f(e)$.

Setze

$$f'(e) := \begin{cases} f(e) - w(p) & \text{für } e \in E(p), \\ f(e) & \text{für } e \notin E(p). \end{cases}$$

Das heißt wir entfernen einen Fluss in Höhe von $w(p)$ auf p. Dann können wir für f' die Induktionsannahme verwenden (weniger Kanten mit positivem Fluss). □

Eingabe : Netzwerk (G, u, s, t)
Ausgabe: Ein s-t-Fluss f mit maximalem Wert

1 Setze $f(e) = 0\ \forall e \in A(G)$
2 Finde einen kürzesten f-augmentierenden Pfad P
3 Falls keiner existiert: **stop**
4 Berechne $\gamma := \min\limits_{e \in P} u_f(e)$
5 Augmentiere f entlang P um γ und **goto** 2

Algorithmus 10: Edmonds-Karp Algorithmus (1972)

Also: Implementiere 2. von Ford-Fulkerson als BFS in G_f

Lemma 4.13. Sei $f_1, f_2, ...$ eine Folge von Flüssen, wobei f_{i+1} aus f_i durch Augmentierung entlang eines kürzesten f-augmentierenden Pfades P_i entsteht. Dann gilt:

a) $|E(P_k)| \leq |E(P_{k+1})|\ \forall\ k$

b) $|E(P_k)| + 2 \leq |E(P_l)|\ \forall\ k < l$, so dass $P_k \cup P_l$ ein Paar entgegengesetzer Kanten enthält.

Beweis.
zu a): Betrachte den Graphen G_1, den man aus $P_k \cup P_{k+1}$ durch entfernen von entgegengesetzen Kanten erhält (Kanten, die in P_k und P_{k+1} auftreten, werden zweifach aufgenommen). Jede Kante aus $E(G_{f_{k+1}}) \setminus E(G_{f_k})$ muss eine Umkehrkante einer Kante in P_k sein. $\Rightarrow E(G_1) \subseteq E(G_{f_k})$. Sei H_1 der Graph bestehend aus zwei Kopien von $(t, s) \Rightarrow G_1 + H_1$ ist eulersch $\Rightarrow$ Jeder Knoten wird genauso oft betreten wie verlassen $\Rightarrow G_1 + H_1$ lässt sich in Menge von Kantendisjunkte Kreise zerlegen. $\Rightarrow$ für jede Kante in H_1 gibt es einen Kantendisjunkten s-t-Pfad Q_1, Q_2. $E(G_1) \subseteq E(G_{f_k}) \Rightarrow Q_1, Q_2$ sind f_k-augmentierende Pfade. P_k kürzester f_k-augmentierender Pfad.
$\Rightarrow |E(P_k)| \leq |E(Q_1)|, |E(P_k)| \leq |E(Q_2)|$.
$\Rightarrow 2|E(P_k)| \leq |E(Q_1)| + |E(Q_2)| \leq |E(G_1)| \leq |E(P_k)| + |E(P_{k+1})|$.
$\Rightarrow |E(P_k)| \leq |E(P_{k+1})|$.
zu b): Wegen a) gilt auch $k < i \leq l : |E(P_k)| \leq |E(P_i)| \leq |E(P_l)|$.
Spezieller: Wir betrachten ein Paar k, l mit $k < i \leq l$ mit $P_i \cup P_l$ hat kein Paar entgegengesetzter Kanten. Wie oben: Sei G_1 der Graph, der aus $P_k \cup P_l$ durch Entfernen entgegengesetzter Kantenpaare entsteht. Wieder $E(G_1) \subseteq E(G_{f_k})$, $E(P_k) \subseteq E(G_{f_k})$, $E(P_l) \subseteq E(G_{f_k})$. Jede Kante in $E(G_{f_l}), E(G_{f_k})$ muss eine Umkehrung einer Kante in $P_k, ..., P_{l-1}$ sein. Aber (Wahl von k und l) nur P_k enthält Umkehrung von Kanten in P_l. Wieder:

- H_1 zwei Kopien von (t, s)
  $\hookrightarrow$ 2 kantendisjunkte s-t-Pfade Q_1 und Q_2
- Q_1, Q_2 beide f_k-augmentierende Pfade

P_k war kürzester f_k-augmentierender Pfad.
$\Rightarrow |E(P_k)| \leq |E(Q_1)|, |E(P_k)| \leq |E(Q_2)|$.

$\Rightarrow 2|E(P_k)| \leq |E(Q_1)| + |E(Q_2)| \leq |G_1| - 2 \leq |E(P_k)| + |E(P_l)| - 2$ da wir mindestens 2 Kanten entfernt haben. □

Satz 4.14. Unabhängig von den Kantenkapazitäten stoppt Algorithmus 10 (Edmonds-Karp) nach höchstens $\frac{m \cdot n}{2}$ Augmentierungen.

Beweis.

i) Jede Augmentierung hat eine "Engstelle", eine "Flaschenhalskante", die den Wert γ, und damit den Wert um den der Fluss erhöht wird, begrenzt.

ii) Eine Kante in $\overleftrightarrow{G}$ kann nur dann zweimal als Flaschenhalskante auftauchen, wenn zwischendurch die Gegenkante in einem augmentierenden Pfad vorgekommen ist (dann wird die Kante wieder in den Residualgraphen aufgenommen). Für eine Kante e sei $P_{i_1}, P_{i_2}, ...$ die Folge von augmentierenden Pfaden, die e als Flaschenhalskante enthalten $\rightarrow$ zwischen P_{ij} und P_{ij+1} gibt es einen augmentierenden Pfad P_k $(ij < k \leq ij + 1)$ der $\overleftarrow{e}$ enthält. $\overset{4.11\ b)}{\Rightarrow}$ $|E(P_{ij})| + 4 \leq |E(P_k)| + 2 \leq |E(P_{ij+1})|$ $\forall j$. Wegen $1 \leq |E(P_{ij})| \leq n - 1$ gilt $j < \frac{n}{4}$ also höchstens $\frac{n}{4}$ augmentierende Pfade enthalten e als Flaschenhalskante. Jeder augmentierende Pfad, muss mindestens eine Kante aus $\overleftrightarrow{G}$ als Flaschenhalskante haben. $\Rightarrow$ Höchstens $\underbrace{|E(\overleftrightarrow{G})|}_{2m} \cdot \frac{n}{4} = \frac{n \cdot m}{2}$ augmentierende Pfade.

□

Korollar 4.15. Der Edmonds-Karp Algorithmus löst MaxFlow in $O(m^2 \cdot n)$.

Beweis. Satz 4.12 $\rightarrow \leq \frac{m \cdot n}{2}$ Augmentierungen, je Augmentierung BFS $\rightarrow O(m)$. □

5 Matching

Definition 5.1 (Matching)**.**

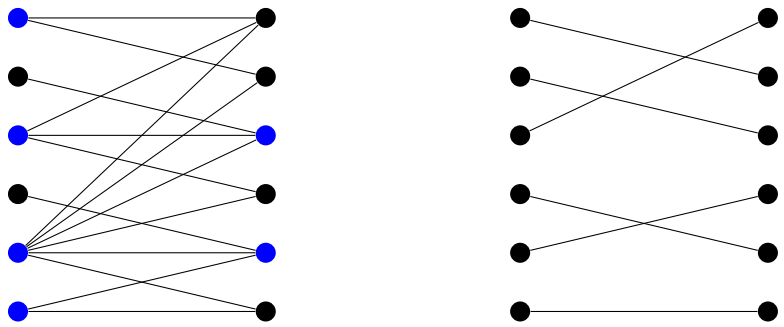
i) Ein **Matching** in einem Grap $G = (V, E)$ ist eine Menge paarweise disjunkter Kanten.

ii) Ein Matching heißt **perfekt**, wenn jeder Knoten in $V(G)$ zu einer Matchingkante gehört.

iii) Ein **Vertex Cover** ist eine Kantenüberdeckende Knotenmenge.

iv) **Maximales Matching** in $G : \mathcal{V}(G)$.

v) **Minimales Vertex Cover** in $G : \mathcal{T}(G)$.



(a) Vertex Cover (b) Matching

Abbildung 5.1: Vertex Cover und Matching im bipartiten Graph

Problem 5.2 (Kardinalitätsmatching/Max Matching)**.**

Gegeben: Ungerichteter Graph $G = (V, E)$.

Gesucht: Ein Matching M größtmöglicher Kardinalität.

Problem 5.3 (Minimales Vertex Cover)**.**

Gegeben: Ungerichteter Graph $G = (V, E)$.

Gesucht: Ein Vertex Cover C kleinstmöglicher Kardinalität.

Satz 5.4. Sei G ein ungerichteter Graph. Es gilt max Matching $\leq$ min Vertex Cover.

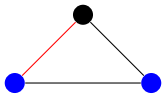
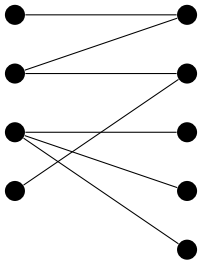


Abbildung 5.2: max Matching $\leq$ min Vertex Cover

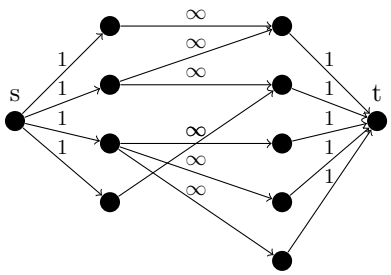
5.1 Bipartites Matching

Definition 5.5 (Bipartite Graphen)**.** Ein Graph heißt **bipartit**, wenn sich die Knotenmenge so in zwei Mengen A, B mit $V(G) = A \dot{\cup} B$ zerlegen lässt, dass jede Kante genau einen Knoten in A und einen Knoten in B enthält.

Damit: Bipartite Graphen enthalten keine Kreise ungerader Länge. Für bipartite Graphen gibt es eine Beziehung zu Flussproblemen:



- *füge 2 neue Knoten s, t ein: Kanten von s nach $v \in A$ und von $v \in B$ zu t.*
- *Kanten von A nach B richten.*
- *$u(e) = 1$ falls $s \in e$ oder $t \in e$, $u(e) = \infty$ sonst.*



Aus gegebenen bipartiten Graph G wird (G', u, s, t).

Satz 5.6. Ein bipartites Matching maximaler Kardinalität in G entspricht einem maximalen Fluss in (G', u, s, t) und umgekehrt.

Beweis.

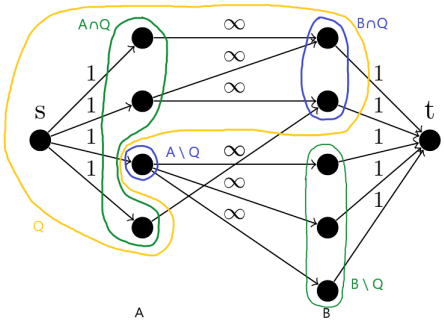
– Jedes Matching in G lässt sich direkt auf einen Fluss in G' abbilden, indem die Matchingkanten einen Fluss von jeweisl 1 erhalten und entsprechende Flüsse von s und t gewählt werden.

– Umgekehrt lässt sich jeder ganzzahliger Fluss (der auf jeder Kante den Wert 0 oder 1 hat) auf ein Matching in G abbilden.

Da es immer ein ganzzahliges Optimum für das Flussproblem gibt, sind also insbesondere die Optimalwerte gleich. □

Satz 5.7. In bipartiten Graphen gilt $\mathcal{V}(G) = \mathcal{T}(G)$.

Beweis. $\mathcal{V}(G) = Max\ Flow(G) = Min\ Cut(G) \stackrel{!}{=} \mathcal{T}(G)$.



Betrachte Min Cut $\delta^+(\{s\} \cup Q), Q \subseteq V$. Dieser Cut hat endliche Kapazität, d.h. es gibt keine Kante von $A \cap Q$ zu $B \setminus Q$.

$$\begin{array}{lll} A \cap Q & \text{zu} & B \cap Q \\ A \cap Q & \text{zu} & B \setminus Q \ \lightning \\ A \setminus Q & \text{zu} & B \cap Q \\ A \setminus Q & \text{zu} & B \setminus Q \end{array}$$

Also ist jede Kante in G inzident zu einem Knoten aus $C = (A \setminus Q) \cup (B \cap Q)$. D.h. C ist ein Cover der Größe $|C| = |A \setminus Q| + |B \cap Q|$. Die Kapazität des Cuts ist ebenfalls $|C| = |A \setminus Q| + |B \cap Q|$ (nach Konstruktion) d.h. Min Cut = min Vertex Cover. □

Satz 5.8 (Satz von Hall)**.** Sei G ein bipartiter Graph mit $V(G) = A \dot\cup B$. Dann hat G ein A überdeckendes Matching $\Leftrightarrow |\underbrace{T(X)}_{\text{Menge der Nachbarn von } X}| \geq |X|\ \forall X \subseteq A$.

Beweis.

– Notwendigkeit ist klar.

– Um zu zeigen, dass auch hinreichend nehmen wir an G hat kein A überdeckendes Matching. $\mathcal{V}(G) \leq |A| \overset{5.7}{\Rightarrow} \mathcal{T}(G) < |A|$. Sei $A' \subseteq A, B' \subseteq B$, so dass $A' \cup B'$ alle Kanten überdeckt und $|A' \cup B'| < |A| \Leftrightarrow |A'| + |B'| < |A|$ (da A' und B' disjunkt) $\Leftrightarrow |B'| < |A| - |A'|$. Es gilt $T(A \setminus A') \subseteq B'$ (da B' alle Kanten überdeckt, die nicht von A' überdeckt werden) $\Rightarrow |T(A \setminus A')| \leq |B'| < |A| - |A'| = |A \setminus A'|$ ↯ Kontraposition der Hinrichtung gezeigt.

□

Korollar 5.9 (Heiratssatz von Frobenius)**.** Sei G ein bipartiter Graph mit $V(G) = A \dot{\cup} B$. Dann hat G ein perfektes Matching $\Leftrightarrow |A| = |B|$ und $|T(X)| \geq |X| \; \forall X \subseteq A$.

Aus dem Beweis von Satz 5.7 folgt

Korollar 5.10. Das Kardinalitätsmatching-Problem kann in bipartiten Graphen in $O(n \cdot m)$ gelöst werden.

Beweis. Konstruktion von oben. Betrachte Ford-Fulkerson für das äquivalente Flussproblem

– Eine Augmentierung benötigt $O(m)$.

– Um einen maximalen s-t-Fluss (und damit ein maximales Matching) zu finden brauchen wir höchstens n Augmentierungen $\Rightarrow (O(m \cdot n))$.

□

Wie sehen Augmentierungen aus?
↪ Verbesserung von Matchings

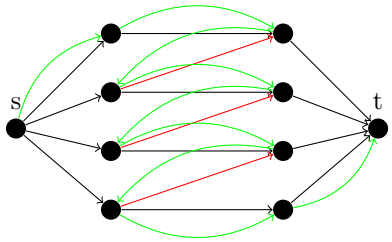


Abbildung 5.3: augmentierender Pfad in G'

Augmentierende Pfade in (G', u, s, t) entsprechen alternierenden Pfaden in G.

Definition 5.11. Sei G ein Graph (bipartit oder nicht), und sei M ein beliebiges Matching in G. Ein Pfad P ist ein **M-alternierender Pfad**, wenn $E(P) \setminus M$ ein Matching ist. Ein M-alternierender Pfad ist **M-augmentierend**, wenn seine Endpunkte nicht von M überdeckt werden (er zwei nicht gematchte Knoten verbindet).

Augmentierende Pfade müssen ungerade Länge haben.

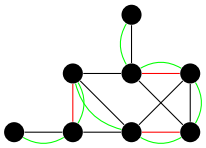


Abbildung 5.4: M-augmentierender Pfad

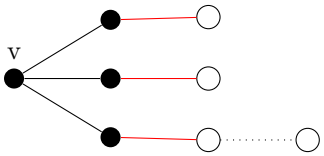
Satz 5.12 (Berg, 1957)**.** Sei G ein Graph mit einem beliebigen Matching M. M hat max Kardinalität $\Leftrightarrow$ es gibt keinen M-augmentierenden Pfad.

Beweis.
„$\Rightarrow$“: Sei P ein M-augmentierender Pfad $\Rightarrow M \Delta E(P)$ $(= (M \setminus E(P)) \cup (E(P) \setminus M))$ ist Matching größterer Kardinalität la M. $\Rightarrow M$ ist nicht maximal.
„$\Leftarrow$“: Sei M' ein Matching mit $|M'| > |M| \Rightarrow M \Delta M'$ ist kantendisjunkte Vereinigung alternierender Kreise und Pfade. Wegen $|M'| > |M|$ muss mindestens einer dieser Pfade M-augmentierend sein. □

Wie findet man einen M-augmentierende Pfad?

- *Matchingkanten (schwarz)*
- *Nicht-Matchingkanten (weiß)*
- *„suchende“ Knoten (weiß)*
- *„gleichgültige“ Knoten (schwarz)*

Starte bei einem ungematchten Knoten $v \rightarrow$ „weiß“.
Falls einer der Nachbarknoten ungematcht
↪ Füge Kante hinzu, liefert besseres Matching
Falls alle gematcht $\rightarrow$ alle „schwarz “; daran: „schwarze“ Kanten
↪ andere Endknoten



Falls einer der weißen Knoten anderweitig „versorgbar“ ist entsteht ein ungerader alternierender Pfad, also eine Verbesserung des Matchings.
Falls nicht: Färbe Endknoten wieder weiß und fahre fort.
$\Rightarrow$ Baumstruktur
↪ weiße\schwarze Knoten abwechselnd
↪ weiße\schwarze Kanten abwechselnd
$\Rightarrow$ Breitensuche von v aus
↪ Für jede ZhK von G wähle einen ungematchten Knoten r als Wurzel

↪ *Wir nennen einen Knoten „schwarz“, wenn er ungeraden Abstand von r hat; „weiß“, wenn er geraden Abstand von r hat*
Mit BFS bauen wir einen alternierenden Wald.

Eingabe : $G = (V, E)$ mit $V = V_1 \dot\cup V_2$
Ausgabe: Maximales Matching M

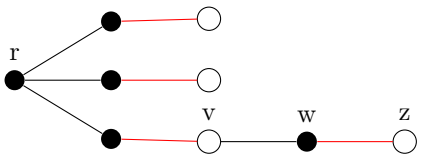
```
1  Setze M = ∅, R = ∅
2  while ∃r ∈ V_1 \ R ungematcht do
3  |  Wähle r ∈ V_1 \ R ungematcht
4  |  Setze T := ({r}, ∅)
5  |  R := R ∪ {r}
6  |  W(T) = {r}
7  |  while ∃{v, w} ∈ E mit v ∈ W(T) ∧ w ∉ V(T) do
8  |  |  if w ist ungematcht then
9  |  |  |  Benutze {v, w} um augmentierenden Pfad zu komplettieren
10 |  |  |    ↪ augmentiere M
11 |  |  |  if es gibt keinen ungematchten Knoten mehr then
12 |  |  |  |  return „Perfektes Matching“
13 |  |  |  end
14 |  |  |  goto 2
15 |  |  else
16 |  |  |  Benutze {v, w} um T zu erweitern
17 |  |  end
18 |  end
19 end
20 return Maximales Matching M
```

Algorithmus 11: Maximales Matching in bipartiten Graphen

Eingabe : Matching M in Graph G,
M-alternierender Baum T,
Kante $\{v, w\}$ von G mit $v \in W(T), w \notin V(T)$, w ist gematcht

```
1  Sei {w, z} die Matchingkante, die w überdeckt
2  Ersetze T durch E(T) = E(T) ∪ {{v, w}, {w, z}}, W(T) = W(T) ∪ {z}
```

Algorithmus 12: Benutze $\{v, w\}$ um T zu erweitern



Satz 5.13. Algorithmus 11 liefert ein korrektes Ergebnis.

Beweis. Satz 5.12 in Verbindung mit Bipartitheit von G. □

Beobachtung 5.14. Für perfekte Matchings brauchen wir eine gerade Anzahl an Knoten. Reicht aber nicht aus.

Satz 5.15 (Satz von Tutte, 1947)**.** Ein Graph $G = (V, E)$ hat ein perfektes Matching $\Leftrightarrow$ für jede Menge $A \subseteq V$ gilt $\underbrace{OC}_{\#\text{ ungerader Komponenten}}(G \setminus A) \leq |A|$.

Satz 5.16 (Tutte-Berg-Formel, 1958)**.** Für $G = (V, E)$ gilt:

$$max\{|M| \mid M \text{ ist Matching}\} = min\{\frac{1}{2}(|V| - OC(G \setminus A) + |A|) \mid A \subseteq V\}$$

Satz 5.17. Sei G bipartit, M ein Matching, T ein alternierender Baum für den keine Kante von G einen Knoten in $W(T)$ mit einem Knoten nicht in $V(T)$ verbindet. Dann hat G kein perfektes Matching.

Beweis. Sei $S(T) = V(T) \setminus W(T)$, die Menge der schwarzen Kanten in T. Nach Konstruktion gilt $|S(T)| = |W(T)| - 1$. G ist bipartit.
$\Rightarrow$ keine 2 weißen Knoten können benachbart sein.
$\Rightarrow$ jeder Knoten in $W(T)$ ungerade Komponente (Kardinalität 1) von $G \setminus S(T)$.
$\underset{|S(T)|<|W(T)|}{\overset{5.15}{\Rightarrow}}$ Behauptung. □