Effiziente Algorithmen 1 - Zusammenfassung

Patrick Dammann

21.05.2017

1 Probleme und Algorithmen

Lineares kombinatorisches Optimierungsproblem
Gegeben sind eine endliche Menge E, ein System von Teilmengen $\mathcal{I} \subseteq 2^E$ (zulässige Lösungen) und eine Funktion $c : E \to \mathbb{R}$. Es ist eine Menge $I^* \in \mathcal{I}$ zu bestimmen, so dass $c(I^*) = \sum_{e \in I^*} c(e)$ minimal bzw. maximal ist.

Euklidisches Traveling-Salesman-Problem
Gegeben sind n Punkte in der Euklidischen Ebene. Zu bestimmen ist eine geschlossene Tour, die jeden Punkt genau einmal besucht und möglichst kurz ist.
E = Menge der Kanten
$\mathcal{I}$ = Alle Mengen von Kanten, die eine Tour bilden

Euklidisches Matching-Problem
Gegeben sind n Punkte in der Euklidischen Ebene (n gerade). Zu bestimmen sind $\frac{n}{2}$ Linien, so dass jeder Punkt Endpunkt genau einer Linie ist und die Summe der Linienlängen so klein wie möglich ist.
E = Menge der Kanten
$\mathcal{I}$ = Alle Mengen von Kanten mit der Eigenschaft, dass jeder Knoten zu genau einer der Kanten gehört.

Einheitskosten-Modell Es werden nur die Schritte des Algorithmus gezählt, die Zahlengrößen bleiben unberücksichtigt.

Bit-Modell Die Laufzeit für eine arithmetische Operation ist M, wobei M die größte Kodierungslänge einer an dieser Operation beteiligten Zahl ist.

Definition 1.1. Die Laufzeitfunktion $f_A : \mathbb{N} \to \mathbb{N}$ ist in $\mathcal{O}(g)$ für eine Funktion $g : \mathbb{N} \to \mathbb{N}$ falls es eine Konstante $c > 0$ und $n_o \in \mathbb{N}$ gibt, so dass $f_A \leq c \cdot g(n)$ für alle $n \geq n_o$.

Definition 1.2. Ein Algorithmus heißt **effizient** bzw. **polynomialer Algorithmus**, wenn seine Laufzeit in $\mathcal{O}\left(n^k\right)$ liegt.
Ein Problem, das mit einem polynomialen Algorithmus gelöst werden kann, heißt **polynomiales Problem**.

Definition. Ein **Graph** G ist ein Tupel $G = (V, E)$[1] bestehend aus einer nicht-leeren Knotenmenge V und einer Kantenmenge E.

- Ein Graph heißt **endlich**, wenn V und E endlich sind.
- Wenn $e = \{u, v\} \in E$ und $u, v \in V$, dann sind u und v **Nachbarn** bzw. **adjazen**t, sind **Endknoten** von e und werden von e **verbunden**.
- Eine Kante $e = \{u, u\} \in E$ heißt **Schleife**.
- Kanten mit $E \ni e = \{u, v\} = f \in E$ heißen **parallel** oder **mehrfach**.
- Ein Graph ohne Mehrfachkanten heißt **einfach**.
- Für $W \subseteq V$ bekommt die Menge aller Knoten in $V \setminus W$ mit Nachbarn in W die Bezeichnung $\Gamma(W)$.
- Kurzform von $\Gamma(\{v\})$ ist $\Gamma(v)$.
- Die Menge $\delta(W)$ aller Kanten mit je einem Endknoten in W und $V \setminus W$ heißt **Schnitt**.
- Kurzform von $\delta(\{v\})$ ist $\delta(v)$.
- Der **Grad** eines Knoten v ist die Anzahl seiner Nachbarn, bzw. $|\delta(v)|$.
- Ein **(s,t)-Schnitt** ist ein Schnitt $\delta(V)$ mit $s \in W$ und $t \in V \setminus W$ und gleichzeitig ein (t,s)-Schnitt.
- Mit $W \subseteq V$ ist $E(W)$ die Menge aller Kanten mit beiden Endknoten in W.
- Mit $F \subseteq E$ ist $V(F)$ die Menge aller Knoten, die Endknoten von mind. einer Kante in F sind.
- Sind $G = (V, E)$ und $H = (W, F)$ Graphen und $W \subseteq V$ und $F \subseteq E$, so heißt H **Untergraph** von G.
- Mit $W \subseteq V$ ist $G - W$ der Graph G ohne die Knoten in W und ohne alle Kanten an W.
- $G[W] = G - (V \setminus W)$ ist der **von W induzierte Untergraph**.
- Mit $F \subseteq E$ ist $G - F = (V, E \setminus F)$.
- Kurzform von $G - \{x\}$ ist $G - x$ für $x \in E$ oder $x \in V$.
- Ein einfacher Graph heißt **vollständig**, wenn jede mögliche Kante zwischen seinen Knoten existiert.
- Der vollstängige Graph mit n Knoten wird mit $K_n = (V_n, E_n)$ bezeichnet.
- Das **Komplement** des Graphen $G = (V, E)$ ist $\bar{G} = (V, E_n \setminus E)$.
- Ein Graph heißt **bipartit**, wenn er sich in zwei disjunkte Teilmengen V_1, V_2 mit $V_1 \cup V_2 = V$ teilen lässt, ohne dass es Kanten $\{u, v\}$ mit $u, v \in V_1 \vee u, v \in V_2$ gibt.

Definition. Ein **Digraph** G ist ein Tupel $D = (V, A)$ bestehend aus einer nicht-leeren Knotenmenge V und einer Kantenmenge A.

- Wenn $a = (u, v) \in A$ und $u, v \in V$, dann ist u **Anfangsknoten** und v **Endknoten** von a. Hier heißt u **Vorgänger** von v und v **Nachfolger** von u.
- Die Kanten (u, v) und (v, u) heißen **antiparallel**.
- Mit $W \subseteq V$ ist $A(W)$ die Menge aller Kanten mit Anfangs- und Endknoten in W.
- Mit $B \subseteq A$ ist $V(A)$ die Menge aller Knoten, die Anfangs- oder Endknoten von mind. einer Kante in B sind.

[1] In der Vorlesung werden primär endliche, einfache, schleifenfreie Graphen behandelt, die der Einfachheit halber eine Notation ohne Inzendenzfunktion nutzen können. In diesem Skript wird (sofern nicht anders angegeben) von solchen Graphen ausgegangen.

- $G = (V, E)$ ist der unterliegende Graph von $D = (V, A)$, wenn E genau die Kanten $\{u, v\}$ enthält, für die (u, v) oder (v, u) in A liegen.
- Ein einfacher Digraph heißt **vollständig**, wenn jede mögliche Kante (in beide Richtungen) zwischen seinen Knoten existiert.
- Der vollstängige Digraph mit n Knoten wird mit $D_n = (V_n, A_n)$ bezeichnet.
- Für $W \subseteq V, W \neq V \neq \emptyset$ enthält $\delta^+(W) = \{(i, j) \in A \mid i \in W, j \notin W\}$ alle Kanten, die W verlassen, $\delta^-(W) = \delta^+(V \setminus W)$ alle Kanten, die in W hinein führen und $\delta(W) = \delta^+(W) \cup \delta^-(W)$ beide. Diese Mengen heißen **Schnitt**. Es gelten die Kurzformen für einzelne Knoten.
- Für $s \in W$ und $t \notin W$ heißt $\delta^+(W)$ auch (s, t)**-Schnitt**.
- Die Kardinalitäten der Schnitte heißen **Außengrad** ($|\delta^+(v)|$), **Innengrad** ($|\delta^-(v)|$) und **Grad** ($|\delta(v)|$).
- $\delta^+(W)$ heißt **gerichteter Schnitt**, wenn $\delta^-(W) = \emptyset$.

Definition. Eine endliche Folge $W = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$ heißt **Kette** oder $[v_0, v_k]$-Kette der Länge k, wenn jede Kante e_i die Knoten v_{i-1} und v_i in einem (Di-)Graphen indiziert und **gerichtete Kette** oder (v_0, v_k)-Kette, wenn alle Kanten in der Form $e_i = (v_{i-1}, v_i)$ sind. v_0 und v_k heißen **Anfangs- und Endknoten**.

- Gibt es in einem (Di-)Graphen keine parallelen Kanten, ist eine (gerichtete) Kette durch ihre Knoten eindeutig identifiziert.
- Ein **(gerichteter) Weg** oder **(gerichteter) Pfad** ist eine (gerichtete) Kette, in der alle Knoten verschieden sind.
- Die Notation $u \underset{D}{\to} v$ bedeutet, dass es einen (u, v)-Weg in D gibt.
- Man spricht auch von (u, v)-Wegen bzw. (u, v)-Pfaden.
- Die Knoten s und t eines Graphen heißen **zusammenhängend**, wenn ein (s, t)-Weg existiert.
- Ein **zusammenhängender Graph** enthält nur Knoten, die paarweise zusammenhängend sind.
- Ein Digraph heißt **stark zusammenhängend**, wenn es zu jedem Knotenpaar s, t einen (s, t)-Weg und einen (t, s)-Weg gibt.
- **Kompontenten** eines Graphen sind die (bezüglich Kanteninklusion) maximalen zusammenhängenden Untergraphen.
- **Starke Kompontenten** eines Diraphen sind die (bezüglich Kanteninklusion) maximalen stark zusammenhängenden Unterdigraphen.
- Ein Graph heißt k**-fach zusammenhängend**, wenn jedes Paar Knoten s, t durch mindestens k (s, t)-Wege verbunden ist, die keine inneren Knoten gemeinsam haben.
- Ein Digraph heißt k**-fach stark zusammenhängend**, wenn jedes Paar Knoten s, t durch mindestens k (s, t)-Wege und (t, s)-Wege verbunden ist, die keine inneren Knoten gemeinsam haben.
- Eine **geschlossene Kette** hat mehr als 0 Kanten und den gleichen Anfangs- und Endknoten.
- Ein **Kreis** ist eine geschlossene Kette mit paarweise verschiedenen inneren Knoten. Seine **Länge** ist die Anzahl seiner Kanten.
- Ein **Eulerpfad** ist eine Kette, die jede Kante eines (Di-)Graphen einmal enthält.
- Eine **Eulertour** ist ein geschlossener Eulerpfad.
- Ein **Eulergraph** ist ein Graph, der eine Eulertour enthält.

- Ein **Hamiltonkreis** (oder **Hamiltontour**) ist ein Kreis der Länge $|V|$.
- Ein **hamiltonischer** Graph enthält einen Hamiltonkreis.
- Ein **Hamiltonweg** ist ein (gerichteter) Weg der Länge $|V| - 1$.
- Ein **Wald** ist eine Kantenmenge in einem Graphen, die keinen Kreis enthält.
- Ein **Baum** ist ein zusammenhängender Wald.
- Ein **aufspannender** Baum enthält alle Knoten des Graphen.
- Ein **azyklischer** Digraph enthält keine gerichteten Kreise.
- Ein **Branching** in einem Digraphen ist eine azyklische Kantenmenge, sodass jeder Knoten maximal eine Eingangskante besitzt.
- Eine **Arboreszenz** ist ein zusammenhängendes Branching.
- Eine **aufspannende Arboreszenz** enthält alle Knoten ihres Digraphen.
- **Gewichte** (auch "Kosten", "Distanzen", "Kapazitäten", usw.) werden durch Funktionen der Form $c : E \to \mathbb{R}$ bzw. $c : A \to \mathbb{R}$ mit Kanten assoziiert.

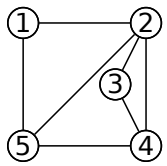
2 Grundlegende Graphenalgorithmen

2.1 Repräsentationen von Graphen

Adjazenzliste Jeder Knoten hat eine Liste seiner Nachbarn gespeichert. Speichersparend für dünne Graphen. Hinzufügen und Entfernen von Knoten und Kanten sehr einfach. Existenz von Kanten prüfen teuer (einen Knoten durchlaufen). Speicheraufwand: $\mathcal{O}(V + E)$

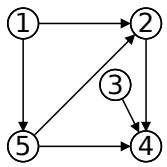
Adjazenzmatrix $|V| \times |V|$-Matrix, die an der Stelle (u, v) eine 1 hat, wenn die Kante (u, v) bzw. $\{u, v\}$ existiert. Existenz von Kanten prüfen in $\mathcal{O}(1)$. Nachbarn durchlaufen unanhängig von ihrer Anzahl in $\mathcal{O}(V)$. Speicheraufwand: $\mathcal{O}\left(V^2\right)$

Inzidenzmatrix $|V| \times |E|$-Matrix, die an der Stelle (v, e) eine 1 hat, wenn der Knoten v Endknoten der Kante e ist. Erlaubt Untersuchung des Graphen mit diversen, algebraischen Methoden. *[Kein Beispiel]*



$$\begin{bmatrix} 1 & \to 2 & \to 5 & \\ 2 & \to 1 & \to 3 & \to 4 & \to 5 \\ 3 & \to 2 & \to 4 & \\ 4 & \to 2 & \to 3 & \to 5 & \\ 5 & \to 1 & \to 2 & \to 4 & \end{bmatrix}$$

$$\left[\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}\right]$$



$$\begin{bmatrix} 1 & \to 2 & \to 5 \\ 2 & \to 4 & \\ 3 & \to 4 & \\ 4 & & \\ 5 & \to 2 & \to 4 \end{bmatrix}$$

$$\left[\begin{array}{c|ccccc} \nearrow & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 1 & 0 & 1 & 0 \end{array}\right]$$

2.2 Durchsuchen von Graphen

Das "färben" von Knoten ist eine Kurzschreibweise für folgende Sachverhalte:

weiß	Der Knoten ist noch nicht erreicht. (Im Normalfall Grundzustand)
grau	Der Knoten wurde erreicht, seine Nachbarn jedoch noch nicht abgearbeitet.
schwarz	Der Knoten wurde komplett bearbeitet.

2.2.1 Breitensuche (BFS)

Start bei Knoten s. $\pi[u] = v$ heißt v ist direkter Vorgänger von u. Q ist eine Queue.

BFS(G, s):

(1) Färbe s grau, setze $d[s] = 0$, $\pi[s] = \perp$ und initialisiere Q mit s.

(2) Färbe alle Knoten $v \in V \setminus \{s\}$ weiß und setze $d[v] = +\infty$ und $\pi[v] = \perp$.

(3) Falls Q leer, *Stop*("BFS beendet"), sonst sei u erster Knoten in Q.

(4) Für alle v aus Adjazenzliste von u:

(4.1) Falls v weiß ist, färbe v grau, setze $d[v] = d[u] + 1$, $\pi[v] = u$ und füge v ans Ende von Q ein.

(5) Entferne u aus Q, färbe u schwarz und gehe zu (3).

Laufzeit: linear in Bezug auf Adjazenzstruktur. Das heißt $\mathcal{O}(V + E)$ bei Adjazenzlisten und $\mathcal{O}(V^2)$ bei Adjazenzmatrizen.

Definition. Für $v \in V$ sei $\delta(s, u)$ die Zahl der Kanten des kürzesten (s, u)-Weges, bzw. ∞ wenn kein solcher existiert.

Lemma 2.1. *Sei $G = (V, E)$ ein Graph und $s \in V$.*

- *Für jede Kante $uv \in E$ gilt: $\delta(s, v) \leq \delta(s, u) + 1$.*
- *Nach Terminierung von BFS(G, s) gilt: $\forall v \in V : d[v] \geq \delta(s, v)$*
- *Enthält Q während BFS(G, s) $v_1, v_2, \ldots, v_r$ gilt: $d[v_r] \leq d[v_1] + 1$ und $d[v_i] \leq d[v_{i+1}], 1 \leq i < r$*

Satz 2.2. *Sei $G = (V, E)$, $s \in V$ und BFS(G, s) ausgeführt. Dann ist jeder Knoten, der von s aus erreichbar ist, schwarz gefärbt und es gilt $d[v] = \delta(s, v)$.*

2.2.2 Tiefensuche (DFS)

Kein spezieller Startknoten. Die Zeit der Grau-Färbung wird in $d[v]$ gespeichtert, die Zeit der Schwarz-Färbung (Terminierungszeit) in $t[v]$.

DFS(D):

(1) Färbe alle Knoten $u \in V$ weiß und setze $\pi[u] = \perp$.

(2) Setze globale Zeit $t = 0$.

(3) Für jeden Knoten $u \in V$:

(3.1) Falls u weiß gefärbt ist, dann DFSVisit(u).

DFSVisit(u):

(1) Färbe u grau, setze $t = t + 1$ und $d[u] = t$.

(2) Für alle v aus Adjazenzliste von u:

(2.1) Falls v weiß ist, dann setze $\pi[v] = u$ und vollziehe DFSVisit(v).

(3) Färbe u schwarz, setze $t = t + 1$ und $f[u] = t$.

Laufzeit: linear in Bezug auf Adjazenzstruktur. Das heißt $\mathcal{O}(V + A)$ bei Adjazenzlisten und $\mathcal{O}(V^2)$ bei Adjazenzmatrizen.

Definition. Der **DFS-Wald** des (Di-)Graphen $G = (V, E)$ ist ein Digraph der Form $(V, \{\{\pi[v], v\} | v \in V, \pi[v] \neq \perp\})$. Er symbolisiert also den Weg durch den Graphen während einer DFS.
Alle Kanten $\{u, v\}$ aus E, die nicht zum Wald gehören sind:

Vorwärtskanten Es gibt einen (u, v)-Weg im DFS-Wald

Rückwärtskanten Es gibt einen (v, u)-Weg im DFS-Wald

Kreuzungskanten Es gibt keinen der Wege im DFS-Wald

Für ungerichtete Graphen gibt es nur Rückwärtskanten.

Satz 2.3. *Ein Knoten v ist Nachfolger eines Knotens u im DFS-Wald genau dann, wenn gilt: Zu dem Zeitpunkt, zu dem u grau gefärbt wird, ist v von u aus auf einem Weg erreichbar, der nur aus weißen Knoten besteht.*

2.3 Topologisches Sortieren

Die topologische Sortierung eines Digraphen ist eine Knotenreihenfolge, in der alle Kanten nur zu später auftretenden Knoten führen.

Topsort(D):

(1) Initialisiere leere Liste $L = \emptyset$.

(2) Führe DFS(D) aus, mit Zusatz: Wenn ein Knoten v schwarz gefärbt wird, füge ihn am Anfang der Liste L ein.

(3) L liefert topologische Sortierung.

Laufzeit: siehe DFS(D)

Lemma 2.4. *Ein Digraph D ist genau dann kreisfrei, wenn DFS(D) keine Ruckwärtskanten liefert.*

Satz 2.5. *Ein azyklischer Digraph D wird durch Topsort(D) topologisch sortiert.*

2.4 Starke Zusammenhangskomponenten

StrongComponents(D):

(1) Führe DFS(D) aus, merke Terminierungszeiten $f[u]$ für alle $u \in V$.

(2) Generiere $D^\top$.

(3) Führe DFS$\left(D^\top\right)$ aus, wobei im Schritt (3) die Knoten nach absteigenden Werten von $f[u]$ sortiert werden.

(4) Die Knoten jedes Baumes im DFS-Wald von $D^\top$ bestimmen eine starke Zusammenhangskomponente.

Lemma 2.6. *Jeder Weg, der zwei Knoten aus derselben Komponente verbindet, enthält nur Knoten aus dieser Komponente.*

Lemma 2.7. *Alle Knoten einer Komponente sind im gleichen DFS-Baum enthalten.*

Definition. Für einen Knoten u bezeichne $\phi(u)$ den Knoten v mit der größten Terminierungszeit bei DFS(D), der von u aus in D erreichbar ist, das heißt:
$\phi(u) = \text{argmax}_v \left\{ f[v] | u \underset{D}{\to} v \right\}$

Satz 2.8. *Für $D = (V, A)$ sei DFS(D) ausgeführt. Dann ist für jeden Knoten $\phi(u)$ Vorgänger von u im DFS-Wald.*

Satz 2.9. *Nach DFS(D) liegen zwei Knoten u und v genau dann in der gleichen Komponente, wenn $\phi(u) = \phi(v)$.*

Satz 2.10. *Der Algorithmus StrongComponents identifiziert die starken Zusammenhangskomponenten eines Digraphen.*

3 Optimale Bäume und Branchings

3.1 Minimale aufspannende Bäume

Die Probleme, einen minimalen, aufspannenden Baum oder einen maximalen, aufspannenden Wald zu finden lassen sich einfach ineinander transformieren. Darum beschränken wir uns auf ersteres.

Minimaler-aufspannender-Baum-Problem

Gegeben ist ein Graph $G = (V, E)$ mit Kantengewichten c_e für $e \in E$. Zu bestimmen ist ein aufspannender Baum für G, dessen Gesamtgewicht möglichst klein ist.
(Wir nehmen an, dass G zusammenhängend ist, da das Problem sonst einzeln pro Komponente lösbar wäre.)

3.1.1 Ein allgemeiner MST-Algorithmus

Dieser Algorithmus färbt Kanten nach festen Regeln blau und rot. Es ist stets die Invariante *„Es gibt einen minimalen, aufspannenden Baum, der alle blauen und keine rote Kante enthält."* erfüllt.

Regel B Wähle einen Schnitt, der keine blaue Kante enthält. Färbe eine seiner kürzesten Kanten blau.

Regel R Wähle einen Kreis, der keine rote Kante enthält. Färbe eine seiner längsten Kanten rot.

MinimumSpanningTree(G, c) :

(1) Zu Beginn seien alle Kanten ungefärbt.

(2) Wende Regeln B und R an, bis alle Kanten gefärbt sind.

(3) Die blauen Kanten bilden einen minimalen aufspannenden Baum.

Satz 3.1. *Der Algorithmus färbt alle Kanten eines zusammenhängenden Graphen und erhält dabei die Invarianz-Bedingung, d.h. er berechnet einen minimalen aufspannenden Baum.*

3.1.2 Der Algorithmus von Boruvka

Boruvka(G, c):

(1) Initialisiere n blaue Bäume bestehend aus je einem Knoten.

(2) Solange mehr als ein blauer Baum vorhanden ist, wähle gleichzeitig zu jedem blauen Baum die minimale inzidente Kante und färbe alle ausgewählten Kanten blau.

(3) Die blauen Kanten bilden einen minimalen Baum.

Der Algorithmus kann blaue Kreise erzeugen, wenn es im Graphen Kanten gleichen Gewichts gibt.

3.1.3 Der Algorithmus von Kruskal

Kruskal(G, c):

(1) Initialisiere n blaue Bäume bestehend aus je einem Knoten.

(2) Sortiere die Kanten von G nach nichtabsteigenden Gewichten, so dass $c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_m}$.

(3) Für $i = 1, 2, \ldots, m$:

(3.1) Sind die Endknoten von e_i im gleichen blauen Baum, färbe e_i rot, andernfalls blau.

(4) Die blauen Kanten bilden einen minimalen Baum.

Laufzeit: $\mathcal{O}(E \log V)$, wenn die Bäume per fast-union-find verwaltet werden. Ein Heap ermöglicht Optimierung bezöglich der Sortierung, da abgebrochen werden kann, wenn es $n - 1$ blaue Kanten gibt.

3.1.4 Der Algorithmus von Prim

Der Algorithmus verwendet nur Regel R und benötigt einen Startknoten s.

Prim(G, c, s):

(1) Initialisiere n blaue Bäume bestehend aus je einem Knoten.

(2) Solange noch blaue Bäume aus einem Knoten existieren, färbe minimale Kante im Schnitt induziert durch den Baum, der s enthält, blau.

(3) Die blauen Kanten bilden einen minimalen Baum.

Es folgt eine einfach Implementierung. T ist der angehende Baum:

Prim1(G, c, s):

(1) Setze $V_T = \{s\}$, $T = \emptyset$ und $i = 0$.

(2) Falls $i = n - 1$, *Stop*(„T ist MST“).

(3) Bestimme $u \in V_T$ und $v \in V \setminus T_T$ mit $c_{uv} = \min\{c_{xy} | x \in V_T, y \in V \setminus V_T\}$.

(4) Setze $V_T = V_T \cup \{v\}$ und $T = T \cup \{uv\}$ und $i = i + 1$. Gehe zu (2).

Laufzeit: $\mathcal{O}(VE)$

Es gibt eine bessere Implementierung.$t[v]$ speichert den nächsten Baumknoten, $d[v]$ die zugehörige Distanz.

Prim2(G, c, s):

(1) Setze $V_T = \{s\}$, $T = \emptyset$ und $i = 0$. Setze $d[v] = c_{sv}$ und $t[v] = s$, falls $sv \in E$, setze $d[v] = \infty$ und $t[v] = \perp$, falls $sv \notin E$.

(2) Falls $i = n - 1$, *Stop*(„T ist MST“).

(3) Bestimme $v \in V \setminus T_T$ mit $d[v] = \min\{d[u] | u \in V \setminus V_T\}$.

(4) Setze $V_T = V_T \cup \{v\}$ und $T = T \cup \{t[v]\, v\}$.

(5) Für alle w adjazent zu v:

(5.1) Falls $w \in V \setminus V_T$ und $c_{vw} < d[w]$, dann setze $d[w] = c_{vw}$ und $t[w] = v$.

(6) Setze $i = i + 1$ und gehe zu (2).

Laufzeit (naiv):	$\mathcal{O}(V^2)$
Laufzeit (Heap):	$\mathcal{O}(E \log V)$ (besser für dünnbesetzte Graphen)
Laufzeit (Fib-Heap):	$\mathcal{O}(E + V \log V)$

3.1.5 Der Round-Robin-Algorithmus

RoundRobin(G, c):

(1) Initialisiere n blaue Bäume bestehend aus je einem Knoten.

(2) Solange weniger als $n - 1$ blaue Kanten vorhanden sind, wähle einen blauen Baum, bestimme die kürzeste Kante im durch diesen Baum induzierten Schnitt und färbe sie blau.

(3) Die blauen Kanten bilden einen minimalen Baum.

Eine $\mathcal{O}(E \log\log V)$-Implementation ist möglich, wenn immer der kleinste Baum gewählt wird.

3.1.6 Eine Anwendung aufspannender Bäume

Sogenannte 1-Bäume sind eine untere Schranke für eine optimale TSM-Tour.

OneTree(G, c):

(1) Bestimme für die Knoten $\{2, 3, \ldots, n\}$einen MST T. Sei c_T die Länge des MST.

(2) Seien e_1 und e_2 die zwei kürzesten Kanten an Knoten 1.

(3) $T \cup \{e_1, e_2\}$ist optimaler 1-Baum mit Wert $c_T + c_{e_1} + c_{e_2}$.

3.2 Maximale Branchings

Neue Notationen: $s : A \to V$, $t : A \to V$ und $c : A \to \mathbb{R}$ sind Start- und Endknoten sowie Gewichte von Kanten.

Maximales-Branching-Problem
Gegeben ist ein Digraph $D = (V, A)$ mit Kantengewichten c_e, für $e \in A$. Zu bestimmen ist ein Branching für D, dessen Gesamtgewicht möglichst groß ist.

3.2.1 Der Branching-Algorithmus von Edmonds

Definition 3.2. Sei $D = (V, A)$ ein Digraph mit Kantengewichten c_e, für $e \in A$.

a) Eine Kante $e \in A$ heißt **kritisch**, falls $c(e) > 0$ und falls $c(e') \leq c(e)$, für alle $e' \in A$ mit $t(e') = t(e)$.

b) Ein Subgraph $H \subseteq A$ heißt **kritisch**, falls er nur aus kritischen Kanten besteht, jeder Knoten Endknoten von höchstens einer dieser Kanten ist, und falls er inklusionsmaximal bezüglich dieser Eigenschaft ist.

Lemma 3.3. *Ein azyklischer kritischer Graph H ist ein maximaler Branching.*

Lemma 3.4. *Sei H ein kritischer Graph. Dann ist jeder Knoten von H in maximal einem Kreis enthalten.*

Lemma 3.5. *Seien B ein Branching und u, v, w drei Knoten. Falls $u \underset{B}{\to} v$ und $w \underset{B}{\to} v$, dann gilt entweder $u \underset{B}{\to} w$ oder $w \underset{B}{\to} u$.*

Definition. Sei B ein Branching. Eine Kante $e \notin B$ heißt **zulässig** (relativ zu B), falls die Menge $B' = B \cup \{e\} \setminus \{f | f \in B \wedge t(f) = t(e)\}$ ebenfalls ein Branching ist.

Lemma 3.6. *Sei B ein Branching und $e \in A \setminus B$. Dann ist e genau dann zulässig relativ zu B, wenn kein Weg von $t(e)$ nach $s(e)$ in B existiert.*

Lemma 3.7. *Sei B ein Branching und C ein Kreis mit der Eigenschaft, dass keine Kante aus $C \setminus B$ zulässig relativ zu B ist. Dann gilt $|C \setminus B| = 1$.*

Satz 3.8. *Sei H ein kritischer Graph. Dann existiert ein Branching$B$ mit maximalem Gewicht, so dass für jeden Kreis $C \subseteq H$ gilt $|C \setminus B| = 1$.*

Im Folgenden bezeichne $V_i = V(C_i)$ und a_i die kürzeste Kante in C_i.

Korollar 3.9. *Sei $D = (V, A)$ ein Digraph mit Kantengewichten und H ein kritischer Graph mit Kreisen $C_i, i = 1, 2, \ldots, k$. Es existiert ein gewichtsmaximales Branching B mit den Eigenschaften*

a) $|C_i \setminus B| = 1$, *für* $i = 1, 2, \ldots, k$

b) *Falls für jede Kante $e \in B \setminus C_i$ gilt, dass $t(e) \notin V_i$,dann folgt $C_i \setminus B = \{a_i\}$.*

Schrumpfen von kritischen Graphen:
Sei $\widetilde{V} = V \setminus \bigcup_{i=1}^{k} V_i$. Für $v \in V_i$ sei $\widetilde{e}(v)$ die Kante aus C_i mit $t(\widetilde{e}(v)) = v$, d.h. die Kreiskante mit Endknoten v.

$$\begin{aligned}
\overline{A} =& \{e \in A | \text{für kein } i \text{ gilt } s(e) \in V_i \text{ und } t(e) \in V_i\} \\
=& A \setminus \bigcup_{i=1}^{k} A(V(C_i)), \\
\overline{V} =& \widetilde{V} \cup \{w_1, w_2, \ldots, w_k\}, \text{ wobei die } w_i \text{ neue Symbole sind,} \\
\overline{s}(e) =& \begin{cases} s(e), & \text{falls } s(e) \in \widetilde{V}, \\ w_i & \text{falls } s(e) \in V_i, \end{cases} \\
\overline{t}(e) =& \begin{cases} t(e), & \text{falls } t(e) \in \widetilde{V}, \\ w_i & \text{falls } t(e) \in V_i, \end{cases} \\
\overline{c}(e) =& \begin{cases} c(e), & \text{falls } t(e) \in \widetilde{V}, \\ c(e) - c(\widetilde{e}(t(e))) + c(a_i) & \text{falls } t(e) \in V_i. \end{cases}
\end{aligned}$$

Es entsteht ein neues Problem $\overline{P}$ im Digraphen $\overline{D} = (\overline{V}, \overline{A})$ mit den neuen Inzidenzfunktionen $\overline{s}$, $\overline{t}$ und Gewichtsfunktion $\overline{c}$.

Satz 3.10. *Es gibt eine bijektive Abbildung zwischen $\mathcal{B}$ (der Menge der Branchings im Originalproblem) und der Menge der Branchings im Problem $\overline{P}$. Es korrespondiert das Branching $B \in \mathcal{B}$ mit dem Branching $\overline{B} = B \cap \overline{A}$ in $\overline{P}$ und es gilt:*

$$c(B) - \overline{c}(\overline{B}) = \sum_{i=1}^{k} c(C_i) - \sum_{i=1}^{k} c(a_i).$$

Branching(D, c):

(1) Bestimme einen kritischen Graphen H für D.

(2) Ist H azyklisch, dann *Stop*(„H ist optimales Branching für D“).

(3) Schrumpfe die Kreise von H, um $\overline{D}$ und $\overline{c}$ zu erhalten.

(4) Berechne durch den rekursiven Aufruf *Branching*$(\overline{D}, \overline{c})$ ein optimales Branching $\overline{B}$ für $\overline{D}$.

(5) Expandiere das Branching $\overline{B}$ zu einem optimalen Branching B für D.

Laufzeit: $\mathcal{O}(A \log V)$ oder $\mathcal{O}(V^2)$.

3.2.2 Arboreszenzen und das asymmetrische TSP

→analog zum symmetrischen Fall

4 Kürzeste Wege

Kürstester-(s,t)-Weg-Problem
Gegeben sind ein Digraph $D = (V, A)$ mit Kantengewichten c_e für $e \in A$, und zwei Knoten $s, t \in V$. Zu bestimmen ist ein (s,t)-Weg in D, dessen Gesamtlänge möglichst klein ist.

Definition. Im folgenden ist $s \in V$ der Startknoten, dessen kürzeste Wege berechnet werden sollen. $\delta(u,v)$ ist die Länge des kürzesten (u,v)-Weges, bzw. $+\infty$ (oder $-\infty$) wenn keiner existiert. $\pi[v]$ ist der Vorgängerknoten von v auf dem kürstesten (s,v)-Weg.

Lemma 4.1. *Sei $p = (v_1v_2, v_2v_3, \ldots, v_{k-1}v_k,)$ ein kürzester Weg von v_1 nach v_k. Dann ist für $1 \leq i \leq j \leq k$ der Teilweg $(v_iv_{i+1}, \ldots, v_{j-1}v_j,)$ ein kürzester Weg von v_i nach v_j.*

Lemma 4.2. *Für jede Kante (u, v) gilt $\delta(s, v) \leq \delta(s, u) + c_{uv}$.*

Die **Standardinitialisierung** für d ist $d[s] = 0$ und $d[v] = \infty, v \in V \setminus \{s\}$ und $\pi[v] = \perp, v \in V$.

Correct(u, v):
Falls $d[v] > d[u] + c_{uv}$, dann setze $d[v] = d[u] + c_{uv}$ und $\pi[v] = u$.