EDM, Algorithmen und Graphenspeicherung

1 Graphenspeicherung

Gespeichert werden soll ein Graph $G = (V, E)$ bzw. Digraph $D = (V, A)$. Man beachte: $E \in \binom{V}{2}$ bzw. $E \subseteq V^2$

1.1 Adjazenzmatrix

- Graph G: $A = (a_{vw})_{v,w \in V}$ mit $a_{vw} = \begin{cases} 1 & \{v, w\} \in E \\ 0 & \text{sonst.} \end{cases}$
- Digraph D: $A = (a_{vw})_{v,w \in V}$ mit $a_{vw} = \begin{cases} 1 & (v, w) \in A \\ -1 & (w, v) \in E \\ 0 & \text{sonst.} \end{cases}$

1.2 Inzidenzmatrix:

- Graph G: $B = (b_{ve})_{v \in V, e \in E}$ mit $b_{ve} = \begin{cases} 1 & v \in e \\ 0 & \text{sonst.} \end{cases}$
- Digraph D: $B = (b_{va})_{v \in V, a \in A}$ mit $b_{va} = \begin{cases} 1 & a_1 = v \\ -1 & a_2 = v \\ 0 & \text{sonst.} \end{cases}$

1.3 Adjazenzliste:

- Graph G: $L = (L_v)_{v \in V}$ mit $L_v = N_G(v) = \{w \in V | \{v, w\} \in E\}$
- Digraph D:
  - Vorgängerliste: $L = (L_v)_{v \in V}$ mit $L_v = N_G^+(v) = \{w \in V | (w, v) \in E\}$
  - Nachfolgerliste: $L = (L_v)_{v \in V}$ mit $L_v = N_G^-(v) = \{w \in V | (v, w) \in E\}$

2 Baumsuche

- Eingabe: Zusammenhängender Graph $G = (V, E)$.
- Ausgabe: Maximaler kreisfreier Untergraph durch gerichtete Adjazenzliste L.
- Hilfsdaten:
  - U: Menge unbearbeiteter Knoten
  - I: Menge der Knoten in Bearbeitung
  - v: Aktueller Knoten
  - w: unbearbeiteter Nachbar
- Ablauf:
  1. Initialisierung:
     - Setze $L_v := \emptyset$ für alle $v \in V$,
     - wähle $I \in \binom{V}{1}$ und
     - setze $U := V \setminus I$.
  2. Solange noch Knoten in I existieren,
     (a) wähle $v \in I$
     (b) Falls $N_G(v) \cap U$ eine Knoten enthält, wähle $w \in N_G(v) \cap U$ und setze

$$U := U \setminus \{w\},$$
$$I := I \cup \{w\} \text{ und}$$
$$L_v := L_v \cup \{w\} \text{ Nachfolgerliste oder } L_w := \{v\} \text{ Vorgängerliste}$$

     Ansonsten setze $I := I \setminus \{v\}$.
  3. Ausgabe.

2.1 Spezialfälle und Bemerkungen

- BFS (Breitensuchbaum): I als Warteschlange (first in first out) realisiert.
- DFS (Tiefensuchbaum): I als Stapel (last in first out) realisiert.
- Auch auf gerichteten Graphen, findet alle vom ersten Knoten aus erreichbaren.
- Laufzeit: $O(|E|)$.

Ist U am Ende nicht leer, so enthält L den Suchbaum einer Komponente. Man kann den Algorithmus in eine Schleife packen (Initialisierung von L bleibt draußen) und am Schleifenende $V = U$ setzen, solange $U \neq \emptyset$ gilt - ansonsten wird die Schleife beendet. Dann erhält man einen Algorithmus, der zu jedem Graphen einen maximalen kreisfreien Untergraphen (Gerüst) sucht.

3 Greedy-Algorithmen

3.1 Best-in-Greedy

- Eingabe: Unabhängigkeitssystem $(E, \mathcal{F})$ mittels Unabhängigkeitsorakel, Gewichte $c : E \mapsto \mathbb{R}_+$.
- Ausgabe: Menge $F \in \mathcal{F}$
- Hilfsdaten: e_i: Elemente von E
- Ablauf:
  1. Initialisierung:
     - Sortiere $E = \{e_1, \ldots, e_n\}$ so, dass $c(e_{|E|}) \leq c(e_{|E|-1}) \leq \ldots \leq c(e_1)$
     - Setze $F := 0$.
  2. Für $i = 1$ bis $|E|$
     (a) Überprüfe $F \cup \{e_i\}$ auf Unabhängigkeit und
     (b) setze gegebenenfalls $F := F \cup \{e_i\}$.

Löst das Maximierungsproblem für Unabhängigkeitssysteme, sofern $(E, \mathcal{F})$ ein Matroid ist, exakt mit Laufzeit $|E| * \text{Laufzeit(Orakel)} + O(n \log n)$.
Beispiel: Minimum Spanning tree, $c(e) := \max\{w_e | e \in E(G)\} - w_e$, $(E, \mathcal{F})$ graphisches Matroid.

3.2 Worst-out-Greedy

- Eingabe: Unabhängigkeitssystem $(E, \mathcal{F})$ mittels Basis-Obermengen-Orakel, Gewichte $c : E \mapsto \mathbb{R}_+$.
- Ausgabe: Basis F von $(E, \mathcal{F})$
- Hilfsdaten: e_i: Elemente von E
- Ablauf:
  1. Initialisierung:
     - Sortiere $E = \{e_1, \ldots, e_n\}$ so, dass $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_{|E|})$
     - Setze $F := E$.
  2. Für $i = 1$ bis $|E|$
     (a) Überprüfe $F \setminus \{e_i\}$ darauf, ob es eine Basis von $(E, \mathcal{F})$ enthält und
     (b) setze gegebenenfalls $F := F \cup \{e_i\}$.

Löst das Minimierungsproblem für Unabhängigkeitssysteme, sofern $(E, \mathcal{F})$ ein Matroid ist, exakt mit Laufzeit $|E| * \text{Laufzeit(Orakel)} + O(n \log n)$.
Beispiel: Minimum Spanning tree, $c(e) := w_e$, $(E, \mathcal{F})$ graphisches Matroid.

4 Edmonds Matroid-Schnitt Algorithmus

- Eingabe: Matroide $(E, \mathcal{F})$ und $(E, \mathcal{F}_2)$ mittels Unabh.-Orakel.
- Ausgabe: Menge $X \in \mathcal{F}_1 \cap \mathcal{F}_2$ maximaler Kardinalität.
- Hilfsdaten: $C_1(y), C_2(y), A_y^{(1)}, A_y^{(2)}, S, T, P$
- Ablauf:

  1. Initialisierung: Setze $X = 0$
  2. Für jedes $y \in E \setminus X$ und jedes $i \in \{1, 2\}$ setze

$$C_i(y) := \begin{cases} \emptyset & X \cup \{y\} \in \mathcal{F} \\ \{x \in X \cup \{y\} | (X \cup \{y\}) \setminus \{x\} \in \mathcal{F}_i\} \end{cases}$$

  3. Setze

$$S := \{y \in E \setminus X | X \cup \{y\} \in \mathcal{F}_1\},$$
$$T := \{y \in E \setminus X | X \cup \{y\} \in \mathcal{F}_2\}.$$

     Für jedes $x \in X$ setze

$$A_x^{(1)} := \{(x, y) | y \in E \setminus X \wedge x \in C_1(X, y) \setminus \{y\}\},$$
$$A_x^{(2)} := \{(y, x) | y \in E \setminus X \wedge x \in C_2(X, y) \setminus \{y\}\}.$$

  4. Suche kürzesten Weg P in $(E, A_x^{(1)} \cup A_x^{(2)})$ von S_x nach B_x über BFS.
     (Hinweis: in Initialisierung BFS kann abweichend $I = S$ gesetzt werden. Das entspricht der Einführung eines gemeinsamen Vorgängerknotens s aller Knoten aus S und Start mit $I = \{s\}$. Abbruch bei erreichen eines Knotens aus T.)
     Gibt es keinen solchen Weg P: Beende Algorithmus.
  5. Setze $X := X \triangle V(P) = (X \setminus V(P)) \cup (V(P) \setminus X)$ und gehe zu 2.

Laufzeit: $O(|E|^3)$

5 Kürzeste Wege Algorithmen

5.1 Dijkstra-Algorithmus

- Eingabe: Digraph $D = (V, A)$ mit $c : A \mapsto \mathbb{R}_+$, $s \in V$ $(t \in V)$
- Ausgabe: Kürzester-Wege-Baum von s zu $v \in V$ (bzw. (s,t)-Weg) durch Vorgängerliste $p : V \mapsto V$ und entspr. Längen $d : V \mapsto \mathbb{R}$.
- Hilfsdaten für jeden Knoten $v \in V$:
  - $d(v)$: aktuell beste bekannte Distanz zu s,
  - $p(v)$: Vorgänger (predecessor) von v im aktuell kürzesten s-v-Weg,
  - $b(v)$: besucht (wahr oder falsch)
- Ablauf:
  1. Initialisierung: Für alle Knoten $v \in V$ setze
  $$d(v) := \begin{cases} 0 & v = s \\ \infty & \text{sonst,} \end{cases},$$
  $$p(v) := 0, \text{ und}$$
  $$b(v) := \text{falsch (kein Knoten zählt als besucht).}$$
  2. Solange nicht alle Knoten aus v besucht sind,
     (a) wähle $u \in \text{Argmin}\{d(v) | v \in V, b(v) = \text{falsch}\}$ und
     (b) setze $b(u) = \text{wahr}$ (u zählt nun als besucht).
     (c) Für alle $v \in V$ mit $b(v) =$falsch, $(u,v) \in A$ und $d(u) + c(u,v) < d(v)$ setze
     $$d(v) := d(u) + c(u,v) \text{ und}$$
     $$p(v) := u.$$

Laufzeit: $\max\{O(|V|^2), O(|E|log|V|)\}$

5.2 Moore-Bellman Algorithmus

- Eingabe: Digraph $D = (V, A)$ mit $c : A \mapsto \mathbb{R}$, $s \in V$ $(t \in V)$ ohne negativen Kreis
- Ausgabe: Kürzester-Wege-Baum von s zu $v \in V$ (bzw. (s,t)-Weg) durch Vorgängerliste $p : V \mapsto V$ und entspr. Längen $d : V \mapsto \mathbb{R}$
- Ablauf: Solange Suche nach $u \in V$ und $(u,v) \in A$ mit $d(u)+c(u,v) < d(v)$ erfolgreich, setze
$$d(v) := d(u) + c(u,v) \text{ und}$$
$$p(v) := u$$

5.3 Floyd-Algorithmus

- Eingabe:
  - Digraph $D = (V, A)$, $V = \{1, \ldots, n\}$ sowie
  - Kantenlängen $c : A \mapsto \mathbb{R}$ ohne negativen Kreis in D
- Ausgabe:
  - Matrix $W = (w_{ij})_{i,j \in V}$ wobei w_{ij} Länge des kürzesten gerichteten i-j-Weges (bzw. für $i = j$ des kürzesten i enthaltenden Kreises) ist, sowie
  - Matrix $P = (p_{ij})_{i,j \in V}$ wobei p_{ij} vorletzter Knoten eines kürzesten (i, j)-Weges ist.
- Ablauf:
  1. Initialisierung: Für alle $i, j \in V$ setze

$$w_{ij} := \begin{cases} c(i,j) & (i,j) \in A \\ \infty & \text{sonst,} \end{cases}$$

$$p_{ij} := \begin{cases} i & (i,j) \in A \\ 0 & \text{sonst.} \end{cases}$$

  2. Für jedes $k \in V$ tue folgendes:
     - Für jedes $i \in V$ tue folgendes:
       - Für jedes $j \in V$ tue folgedes:
         - Falls $w_{ij} > w_{ik} + w_{kj}$ setze

$$w_{ij} := w_{ik} + w_{kj} \text{ und}$$
$$p_{ij} := p_{kj}.$$

(Falls $i = j$ und $w_{ij} < 0$ Fehler: negativer Kreis!)

Laufzeit: $O(|V|^3)$

6 Flüsse in Netzwerken

6.1 Ford-Fulkerson-Algorithmus

- Eingabe:
  - Digraph $D = (V, A)$, $V = \{1, \ldots, n\}$ sowie
  - untere Kantenkapazitäten $\underline{c} : A \mapsto \mathbb{R}$
  - obere Kantenkapazitäten $\overline{c} : A \mapsto \mathbb{R}$, wobei $\underline{c}(a) \leq \overline{c}(a)$ für alle $a \in A$ gelten möge.
  - Zulässiger Fluss $x : A \mapsto \mathbb{R}$, wobei zulässig bedeutet:

$$\forall v \in V \setminus \{s, t\} : \sum_{(u,v)\in A} x(u, v) = \sum_{(v,u)\in A} x(v, u) \text{ und}$$
$$\forall a \in A : \quad \underline{c}(a) \leq x(a) \leq \overline{c}(a).$$

- Ausgabe:
  - Zulässiger Fluss x maximaler Kapazität $\sum_{(s,v)\in A} x(s, v) - \sum_{(v,s)\in A} x(v, s)$.
  - Menge $W \subseteq V$ mit $s \in W$ und $t \in V \setminus W$ sodass

$$\sum_{(u,v)\in A\cap(W\times(V\setminus W))} x(u, v) - \sum_{(u,v)\in A\cap((V\setminus W)\times W)} x(u, v)$$

    minimal ist (minimaler Schnitt).

- Hilfsdaten wie bei Baumsuche
  - U: Menge von Knoten außerhalb des bisher abgesuchten Baumes
  - I: Menge der Knoten in Bearbeitung
  - v: Aktueller Knoten
  - w: unbearbeiteter Nachbar
  - p_v: Vorgängerknoten auf Verbesserungsweg
  - ε_v: Bestmögliche Verbesserung auf letzter Kante (negativ: letzte Kante ging von v aus)

- Ablauf:

  Wiederhole

  1. Baue gerichteten Baum für von s mit augmentierenden Wegen erreichbare Knoten auf (Baumsuche)
     (a) Initialisierung Baumsuche:
     - Setze $p_v := 0$ für alle $v \in V$,
     - wähle $I = \{s\}$ und
     - setze $U := V \setminus I$ sowie $\varepsilon_s := \infty$.

     (b) Solange noch Knoten in I existieren und $t \notin I$,
     i. wähle $v \in I$
     ii. Falls U einen Knoten w enthält mit
     - $(v, w) \in A$ und $\varepsilon := \overline{c}(v, w) - x(v, w) > 0$
       (ausgehende Kante zu w mit Platz nach oben) oder
     - $(w, v) \in A$ und $\varepsilon := \underline{c}(w, v) - x(w, v) < 0$
       (von w eingehende Kante mit Platz nach unten),

     dann wähle einen solchen Knoten w und setze

     $$\begin{aligned} U &:= U \setminus \{w\}, \\ I &:= I \cup \{w\}, \\ p_w &:= v \text{ und} \\ \varepsilon_w &:= \text{signum}(\varepsilon) \min\{|\varepsilon_v|, |\varepsilon|\} \end{aligned}$$

     Ansonsten setze $I := I \setminus v$.
  2. Ist $t \notin U$ so verbessere den Fluß entlang des im Baum gefundenen s-t-Weges:
     (a) Setze $\varepsilon := \varepsilon(t)$
     (b) Setze $v := t$
     (c) Wiederhole
     i. Setze $u := p_v$
     ii. Falls $\varepsilon_v > 0$ setze $x(u, v) := x(u, v) + \varepsilon$
     ansonsten setze $x(v, u) := x(v, u) - \varepsilon$
     iii. Setze $v = u$.
     bis $u = s$.

  bis $t \in U$, also bis im erzeugten Baum kein Weg mehr nach t führt.

- Laufzeit: $O(|E|^2|V|)$, wenn I als Warteschlange organisiert.

6.2 Erzeugung eines zulässigen Flusses

- Eingabe:
  - Digraph $D = (V, A)$, $V = \{1, \ldots, n\}$ sowie
  - untere Kantenkapazitäten $\underline{c} : A \mapsto \mathbb{R}$
  - obere Kantenkapazitäten $\overline{c} : A \mapsto \mathbb{R}$, mit $\underline{c}(a) \leq \overline{c}(a)$ für alle $a \in A$.
- Ausgabe: Zulässiger Fluss $x : A \mapsto \mathbb{R}$
- Hilfsdaten: Hilfsgraph (V^*, A^*), Balance (Verbrauch) $b : V \mapsto \mathbb{R}$
- Ablauf:
  1. Erweitere Graphen um Zusatzbogen für unbalancierte Knoten bzgl. Fluss $\underline{c}$ zu Ersatzgraph (V^*, A^*):
     (a) Setze $V^* := V \dot{\cup} \{s^*, t^*\}$ und $A^* := A \cup \{(t, s)\}$.
     (b) Für jeden Knoten $v \in V$
     - setze $b(v) := \sum_{(u,v)\in A} \underline{c}(u, v) - \sum_{(v,u)\in A} \underline{c}(v, u)$
     - falls $b(v) \neq 0$ setze

$$A^* := A^* \cup \begin{cases} \{(v, s^*)\} & \text{falls } b(v) > 0 \\ & \text{(Senken vorwärts mit Quelle verbinden),} \\ \{t^*, v\} & \text{falls } b(v) < 0 \\ & \text{(Quellen rückwärts mit Senke verbinden).} \end{cases}$$

  2. Führe den Ford-Fulkerson-Algorithmus für Ersatzgraph (V^*, A^*) mit oberen Kantenkapazitäten $\overline{c}^*(u, v) = \begin{cases} \overline{c}(u, v) & (u, v) \in A, \\ \infty & \text{sonst,} \end{cases}$

unteren Kantenkapazitäten $\underline{c}^*(u, v) = \begin{cases} \underline{c}(u, v) & (u, v) \in A, \\ -\infty & (u, v) = (t, s), \\ 0 & \text{sonst,} \end{cases}$

zulässigem Fluss $x(u, v) = \begin{cases} \underline{c}^*(u, v) & \text{falls } (u, v) \in A \setminus \{(t, s)\}, \\ b(u) & \text{falls } v = s^*, \\ -b(v) & \text{falls } u = t^*, \\ 0 & \text{sonst, d.h. falls } (u, v) = (t, s), \end{cases}$

Quelle s^* und Senke t^* aus.

  3. Falls der Wert des gefundenen Flusses gleich 0 ist, gib ihn eingeschränkt auf G aus, ansonsten gib aus, dass kein zuläsiger Fluss existiert.