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.