7.3 Minimale Spannbäume

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph mit einer Gewichtsfunktion $w : E \to \mathbb{Q}^+$. Das Gewicht eines Teilgraphen $G' = (V', E')$ von G ist dann $w(G') = \sum_{e \in E'} w(e)$.

Definition 213

Ein minimaler Spannbaum von $G = (V, E)$ ist ein Spannbaum von G (also ein *Baum* (V, E') mit $E' \subseteq E$) mit minimalem Gewicht unter allen Spannbäumen von G.

Anwendungen: Verdrahtungs- und Routingprobleme, Zuverlässigkeit von Netzwerken

Lemma 214 (rote Regel)

Sei $G = (V, E)$, $e \in E$ schwerste Kante auf einem Kreis C in G. Dann gibt es einen minimalen Spannbaum von G, der e nicht enthält (d.h. dann ist jeder minimale Spannbaum von $G - e$ auch minimaler Spannbaum von G). Ist e die einzige schwerste Kante auf einem Kreis C in G, dann ist e in keinem minimalen Spannbaum von G enthalten.

Beweis:

Sei e schwerste Kante in C, und sei M ein minimaler Spannbaum von G, der e enthält. Durch Entfernen von e zerfällt M in zwei Teilbäume, die, da C ein Kreis ist, durch eine geeignete Kante $f \neq e$ aus C wieder verbunden werden können. Der dadurch entstehende Spannbaum M' von G enthält e nicht und hat das Gewicht

$$w(M') = w(M) - w(e) + w(f)\,.$$

Falls e einzige schwerste Kante in C ist, wäre $w(M') < w(M)$ im Widerspruch zur Tatsache, dass M minimaler Spannbaum von G ist. □

Der Algorithmus $\textsc{Kruskal}(V, E, w)$:

sortiere E aufsteigend: $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$
initialisiere Union-Find-Struktur für V
$T := \emptyset$
for $i = 1$ **to** m **do**
  bestimme die Endpunkte v und w von e_i
  if Find$(v) \neq$ Find(w) **then**
    $T := T \cup \{e_i\}$
    Union(Find(v),Find(w))
return T

Satz 215

Kruskal's Algorithmus bestimmt einen minimalen Spannbaum des zusammenhängenden Graphen $G = (V, E)$ in Zeit $O(m \log n)$.

Beweis:

Find$(v) =$ Find(w) gdw die Kante (v, w) schwerste Kante auf einem Kreis in G ist. Die Korrektheit des Algorithmus folgt damit aus dem vorigen Lemma.

Die Laufzeit für das Sortieren beträgt $O(m \log n)$ ($m \geq n - 1$, da G zusammenhängend). Bei geeigneter Implementierung (gewichtete Vereinigung) ist die Zeitkomplexität jeder Find-Operation $O(\log n)$. □

Lemma 216 (blaue Regel)

Sei $G = (V, E)$, $(W, V \setminus W)$ ein Schnitt von G (d.h. $\emptyset \neq W \neq V$). Sei ferner e eine leichteste Kante $\in W \times (V \setminus W) \cap E$. Dann gibt es einen minimalen Spannbaum von G, der e enthält. Ist e die einzige leichteste Kante in dem Schnitt, dann ist e in jedem minimalen Spannbaum von G enthalten.

Beweis:

Sei $(W, V \setminus W)$ ein Schnitt in G und e leichteste Kante in diesem Schnitt. Sei ferner M ein minimaler Spannbaum von G, der e nicht enthält. Durch Hinzufügen von e zu M entsteht (genau) ein Kreis, der, da M ein Spannbaum ist, mindestens eine weitere Kante f aus dem Schnitt $(W, V \setminus W)$ enthält. Entfernt man nun f, so entsteht wieder ein Spannbaum M', mit dem Gewicht

$$w(M') = w(M) - w(f) + w(e)\,.$$

Falls e einzige leichteste Kante im Schnitt ist, wäre $w(M') < w(M)$ im Widerspruch zur Tatsache, dass M minimaler Spannbaum von G ist. □

Der Algorithmus $\textsc{Prim}(V, E, w)$:

$W := \{s\}$ für ein beliebiges $s \in V$; $T := \emptyset$

initialisiere Priority-Queue-Struktur R für V, Schlüssel $\rho(v)$ von v gleich

$$\rho(v) := \begin{cases} 0 & \text{falls } v = s \\ d(s,v) & \text{falls } v \in \Gamma(s) \\ \infty & \text{sonst} \end{cases}$$

$$\text{pred}[v] := \begin{cases} s & \text{falls } v \in \Gamma(s) \\ \text{nil} & \text{sonst} \end{cases}$$

while $W \neq V$ **do**

  $x :=$ ExtractMin(R)

  $W := W \cup \{x\}$; $T := T \cup \{x, \text{pred}[x]\}$

  for all $v \in \Gamma(x) \cap (V \setminus W)$ **do**

    if $\rho(v) > w(x,v)$ **then**

      DecreaseKey$(R, v, w(x,v))$; pred$[v] := x$

return T

Satz 217

Prim's Algorithmus bestimmt einen minimalen Spannbaum des zusammenhängenden Graphen $G = (V, E)$ in Zeit $O(m + n \log n)$ (bei Verwendung von Fibonacci-Heaps).

Beweis:

Betrachte in jeder Iteration den Schnitt $(W, V \setminus W)$. Die Korrektheit des Algorithmus folgt damit aus dem vorigen Lemma.

Der Algorithmus benötigt i.W. $\leq m$ DecreaseKey-Operationen und $n - 1$ ExtractMin-Operationen einer Priority-Queue. Damit ergibt sich die behauptete Zeitkomplexität. □

Kapitel IV Komplexitätstheorie

1. Definitionen

Definition 218

Sei M eine deterministische Turingmaschine und $\Sigma = \{0, 1\}$.

1. $\text{TIME}_M(x) :=$ Anzahl der Schritte, die M bei Eingabe $x \in \Sigma^*$ durchführt
2. $\text{DTIME}(f) :=$ Menge aller Sprachen, für die es eine deterministische Mehrband-Turingmaschine M gibt mit $\text{TIME}_M(x) \leq f(|x|)$ für alle $x \in \Sigma^*$
3. $$\mathcal{P} = \bigcup_{p \text{ Polynom}} \text{DTIME}(p)$$

$\mathcal{P}$ ist die Menge der polynomiell lösbaren Probleme, es wird allgemein mit der Klasse der effizient lösbaren Probleme gleichgesetzt.