Karlsruher Institut für Technologie — Prof. Dr. Jörn Müller-Quade
Institut für Theoretische Informatik — Björn Kaidel, Sebastian Schlag, Sascha Witt

11. Tutorenblatt zu Algorithmen I im SoSe 2017

http://crypto.iti.kit.edu/index.php?id=799
{bjoern.kaidel,sascha.witt}@kit.edu

In Vorlesung wurden Bellman-Ford, azyklische Graphen, Routing, minimale Spannbäume, der Jarnik-Prim-Algorithmus und Kruskal Algorithmus behandelt.

Aufgabe 1 (*Minimale Spannbäume*)

Es sollte kurz wiederholt werden was ein minimaler Spannbaum ist und grundlegende Eigenschaften wie die Schnitteigenschaft und die Kreiseigenschaft. Eine weitere mögliche Aufgabe ist es einen minimalen Spannbaum in einem ungerichteten zusammenhängenden gewichteten Graph G zu finden (durch scharfes hinsehen/ausprobieren).

Aufgabe 2 (*Algorithmen zur Berechnung minimaler Spannbäume*)

Gegeben sei ein ungerichteter zusammenhängender gewichteter Graph G. Führe den Jarnik-Prim-Algorithmus und Kruskals-Algorithmus auf G aus. (Dies dient auch as Vorbereitung für die Kreativaufgabe.)

Aufgabe 3 (*Kreativaufgabe*)

Streaming MST: Gegeben sei ein zusammenhängender Graph G mit n Knoten und m Kanten, dessen Knoten lokal gespeichert sind, und dessen Kanten über eine Netzwerkverbindung o.ä. gestreamt werden. Man hat nicht genug Speicherplatz um alle Kanten lokal zu Speichern, da lokal nur $O(n)$ Platz verfügbar ist. Die Kanten kommen in einer beliebigen Reihenfolge an, sie sind insbesondere nicht sortiert. Die Kanten werden aber einzeln angefordert, wir haben hier kein Echtzeitproblem.
Stufe 1: Gib einen Algorithmus an, der einen MST von G unter diesen Einschränkungen bestimmt.
Stufe 2: Verbessere diesen Algorithmus so, dass er nur $O(m \log n)$ Rechenzeit benötigt.

Musterlösung:
Stufe 1: Verwende eine Union-Find Datenstruktur wie bei Kruskal. Wenn die neue Kante zwei verschiedene Teilbäume verbindet nimm sie hinzu. Wenn die neue Kante zwei Knoten eines Teilbaumes verbindet, bestimme den Weg zwischen diesen beiden Knoten im Teilbaum und verwende die Kreiseigenschaft, d.h. laufe den Weg zwischen den beiden Knoten im Teilbaum ab und bestimme die schwerste Kante darauf, dann entscheide ob diese oder die neue Kante weggeworfen wird. Dieser Algorithmus hat Laufzeit $O(mn)$.
Stufe 2: Man besorgt sich Kanten-Pakete mit jeweils n Kanten. Mit dem ersten Paket bestimmt man einen minimalen spannenden Wald von G. Für die weiteren Pakete geht man folgendermaßen vor:

a) Vereinige die Menge der n Kanten mit der Kantenmenge des minimalen spannenden Waldes aus dem Vorgängerschritt.

b) Bestimme mit z.B. Kruskal für diese Menge von maximal $2n - 1$ Kanten einen minimalen spannenden Wald für G.

Jeder Schritt benötigt Zeit in $O(n \log n)$. Die $\frac{m}{n}$ Schritte haben damit die gewünschte Laufzeit.

Aufgabe 4 (*Minimum Spanning Tree (MST)*)

Wir betrachten beliebige zusammenhängende ungerichtete Graphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$, $n \in \mathbb{N}$, und Kantengewichten aus $\{1, 3\}$. Sei G in Form eines Adjazenzfeldes gegeben.

a) Geben Sie einen Algorithmus an, der in Zeit $\mathcal{O}(|E|)$ einen MST von G berechnet.

b) Argumentieren Sie kurz, warum Ihr Algorithmus aus Teilaufgabe a) das gewünschte Laufzeitverhalten aufweist.

Musterlösung:

a) Sei $c : E \to \{1, 3\}$ die Gewichtsfunktion von G. Man modifiziere den Jarník-Prim-Algorithmus nun wie folgt:

- Man ersetze die Priority-Queue (d.h. den binären Heap) Q durch zwei FIFO-Queues Q_1 und Q_3. Diese sollen als doppelt verkettete Listen realisiert sein, sodass Elemente in Zeit $\mathcal{O}(1)$ nicht nur am Anfang und Ende der FIFO-Queue entfernt werden können.
- Man lässt $Q.\mathit{insert}(u)$ weg, stattdessen durchläuft man alle Kanten $(u, v) \in E$. Für $c(u, v) = 1$ führe $Q_1.\mathit{pushBack}(v)$ aus, ansonsten führe $Q_3.\mathit{pushBack}(v)$ aus.
- Statt $Q.\mathit{deleteMin}$ führt man $Q_1.\mathit{popFront}$ aus, sofern Q_1 nicht leer ist. Ansonsten führt man $Q_3.\mathit{popFront}$ aus. Wenn aber auch dieses leer ist, wird der Algorithmus beendet.
- Beim Relaxieren einer Kante $(u, v) \in E$ werden Knoten v mit $v \notin Q_1$ und $v \notin Q_3$ in $Q_{c(u,v)}$ eingefügt, beim Verringern eines Gewichtes (nur von 3 nach 1 möglich) entfernt man v aus Q_3 und führt $Q_1.\mathit{pushBack}(v)$ aus. Allerdings benötigt man hierzu auch ein Array $A[1..n]$, das zu jedem $v \in V$, welches sich in Q_3 befindet, an der Stelle $A[v]$ einen Handle auf das zugehörige Listenelement speichert.

b) Jede Kante in E wird genau einmal betrachtet. Dabei wird jeweils höchstens ein *pushBack* ausgeführt und höchstens ein *remove* auf einer doppelt verketteten Liste. Außerdem wird für jeden Knoten höchstens einmal *popFront* ausgeführt und es gilt $|V| = \mathcal{O}(|E|)$. All die Operationen *pushBack*, *popFront* und *remove* benötigen jeweils $\mathcal{O}(1)$ Zeit. Initialisieren von A benötigt $\mathcal{O}(|V|) = \mathcal{O}(|E|)$ Zeit. Insgesamt braucht man also $\mathcal{O}(|E|)$ Zeit.