Universität Karlsruhe
Fakultät für Informatik
ITI Wagner

Algorithmen für Routenplanung
Sommersemester 2009
Daniel Delling, Thomas Pajor

Drittes Übungsblatt - Musterlösung

Aufgabe 1: Contraction Hierarchies

Gegeben sei ein Graph $G = (V, E, \text{len})$.

(a) Um die 1-Hop-Zeugensuche für die Knotenreduktion eines Knotens v effizient berechnen zu können, reicht es aus alle Kombinationen aus eingehenden und ausgehenden Kanten zu betrachten. Für jede Kante $(u, v) \in E$ und $(v, w) \in E$ wird überprüft ob die Kante (u, w) bereits im Graphen enthalten ist.

- Fall I: Die Kante (u, w) ist nicht im Graph: Füge (u, w) in E ein und setze $\text{len}(u, w) := \text{len}(u, v) + \text{len}(v, w)$.
- Fall II: Die Kante (u, w) ist bereits im Graph: Falls $\text{len}(u, w) > \text{len}(u, v) + \text{len}(v, w)$, setze $\text{len}(u, w) := \text{len}(u, v) + \text{len}(v, w)$.

(b) Vermöge wieder die Knotenreduktion eines Knotes v. Seien dazu weiterhin $(u, v) \in E$ und $(v, w) \in E$ gegeben. Bei der 2-Hop-Zeugensuche wird überprüft ob es in G einen kürzesten u-v-Weg $P_{u,v}$ gibt, der maximal zwei Hops (Kanten) enthält.

Analog zu Aufgabe (a) wird der Shortcut (u, w) nur dann in E eingefügt, falls $\text{len}(P) > \text{len}(u, v) + \text{len}(v, w)$ gilt.

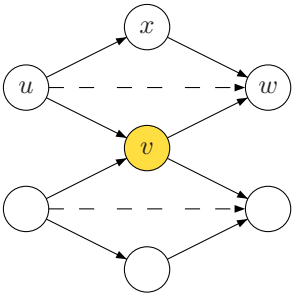
Zur Bestimmung von $\text{len}(P)$ bedienen wir uns folgendem effizienten zweiphasigen Verfahren, das ohne einer lokalen DIJKSTRA-Suche auskommt:

1. Sei $N_{\text{out}}(v)$ die Menge Nachbarn von v, zu denen es eine ausgehende Kante von v gibt, also

$$N_{\text{out}}(v) := \{w \in V \mid \exists (v, w) \in E\}.$$

Zu einem beliebigen Knoten v sei analog $N_{\text{in}}(v)$ die Menge Nachbarn von v, zu denen es eine eingehende Kante zu v gibt, also

$$N_{\text{in}}(v) := \{u \in V \mid \exists (u, v) \in E\}.$$



Sei nun $v \in V$ der Knoten, für den die Knoten-Reduktion durchgeführt werden soll. Für jeden Knoten $w \in N_{\text{out}}(v)$ betrachte alle Knoten $x \in N_{\text{in}}(w)$. Der Abstand $\text{len}(x, w)$ wird zusammen mit w in einer Tabelle an x abgespeichert. Somit ist an x die Information verfügbar, welchen Abstand von x alle Knoten aus $N_{\text{out}}(v)$ haben.

2. Für jeden Knoten $u \in N_{\text{in}}$ werden alle Knoten $x \in N_{\text{out}}(u)$ betrachtet. Enthält x eine Tabelle, so wird für alle eingetragenen Knoten w aus der Tabelle an x überprüft ob $\text{len}(u,x)+\text{len}(x,w) > \text{len}(u,w)$ gilt. Ist dies der Fall, so kann der Shortcut (u,w) eingefügt werden, andernfalls existiert bereits ein kürzerer u-w-Weg in G, der über zwei Hops führt.

Aufgabe 2: Transit-Node Routing

Gegeben sei ein Graph $G = (V, E, \text{len})$ und $L+1 \in \mathbb{N}$ Level von Mengen von Transit-Nodes $T_L \subseteq \cdots \subseteq T_1 \subseteq T_0$, wobei $T_0 = V$.

(a) Sind die Mengen von Transit-Nodes T_l gegeben, so können die Access-Nodes in drei Stufen berechnet effizient werden.

- **Lokale Suchen**

  Für jeden Level $l < L$ wird ausschließlich von *Transit-Knoten* $v \in T_l$ eine lokale Dijkstra-Suche gestartet. Dabei benutzen wir Aggressive-Stalling an den Transit-Knoten des nächst höheren Levels. Wird also während der Suche ein Transit-Knoten $t \in T_{l+1}$ abgearbeitet, so werden für diesen Knoten keine ausgehenden Kanten relaxiert.

  Als Ergebnis erhalten wir alle Abstände von $v \in T_l$ zu allen Transit-Knoten $t \in T_{l+1}$ auf dem nächst höheren Level $l+1$. Natürlich sind nicht alle Transit-Knoten $t \in T_{l+1}$ Access-Nodes für den Knoten v bezüglich Level l. Daher wird in einem zweiten Schritt die Menge relevanter Access-Nodes ausgedünnt.

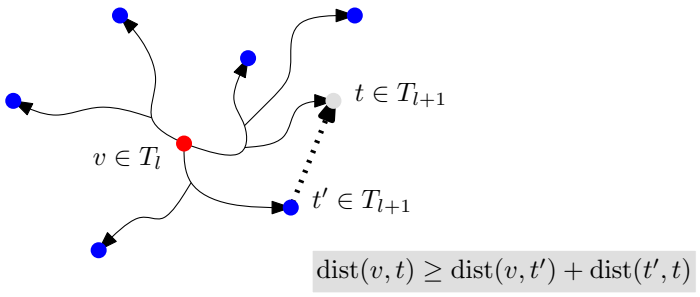
- **Ausdünnung**

  Seien die Distanztabellen D_l für alle Level $l \leq L$ bereits gegeben (siehe Aufgabe b). Da für jeden Transit-Knoten $t \in T_{l+1}$ der Abstand $\text{dist}(v,t)$ mit der lokalen Suche im vorangegangenen Schritt berechnet wurde, können wir schnell ermitteln ob der Knoten t als Access-Node relevant ist indem wir überprüfen ob

  $$\exists t' \in T_{l+1} : \text{dist}(v,t) \geq \text{dist}(v,t') + \text{dist}(t',t) \tag{1}$$

  erfüllt wird. Dabei ist $\text{dist}(v,t')$ ebenfalls durch die lokale Suche bereits berechnet, und $\text{dist}(t',t)$ lässt sich durch die Distanztabelle D_{l+1} nachschlagen.

  Wird Gleichung (1) erfüllt, so ist der Transit-Knoten $t \in T_{l+1}$ überflüssig, da er schneller über einen anderen Transit-Knoten $t' \in T_{l+1}$ erreicht werden kann.



  Die Menge von Access-Nodes $\overrightarrow{A}_{l+1}(v)$ setzt sich schließlich aus den Transit-Knoten zusammen die nicht dominiert werden, also insgesamt

  $$\overrightarrow{A}_{l+1}(v) = \{t \in T_{l+1} \mid \nexists t' \in T_{l+1} : \text{dist}(v,t) \geq \text{dist}(v,t') + \text{dist}(t',t)\}.$$

- **Top-Down-Durchreichen**

  In den beiden vorangegangenen Schritten haben wir Access-Nodes $\overrightarrow{A}_{l+1}(v)$ lediglich für Transit-Knoten $v \in T_l$ berechnet. Um nun für beliebige Knoten (insbesondere auf dem untersten Level) Access-Nodes für *alle* höheren Level zur Verfügung zu haben, können wir Access-Nodes höherer Level nach unten weiterpropagieren.

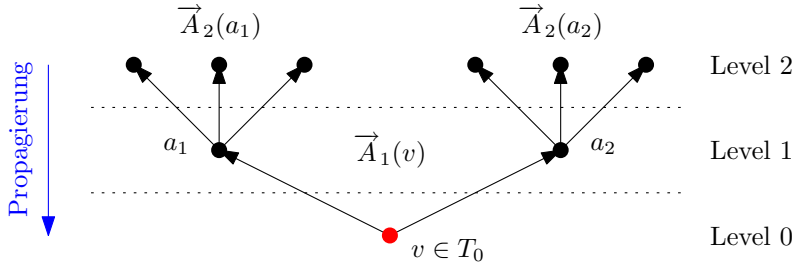
  Sei also $v \in T_0 \stackrel{!}{=} V$ ein beliebiger Knoten auf dem untersten Level. Die Access-Nodes auf Level L können auf Level $L-1$ propagiert werden, indem für jeden Knoten $a \in \overrightarrow{A}_{L-1}(v)$ deren Access-Nodes $x \in \overrightarrow{A}_L(a)$ aufgesammelt werden. Also insgesamt

  $$\overrightarrow{A}_L(v) := \bigcup_{a \in \overrightarrow{A}_{L-1}(v)} \overrightarrow{A}_L(a).$$

  Da jedoch für $v \in T_0$ die Menge $\overrightarrow{A}_{L-1}(v)$ nicht bekannt ist (lediglich $\overrightarrow{A}_1(v)$ ist bekannt), müssen wir dieses Verfahren rekursiv durchführen indem Access-Nodes von Level $L-1$ auf Level $L-2$ herunterpropagiert werden, diese auf Level $L-3$, und so weiter. Insgesamt erhält man für einen Knoten $v \in T_0$ also alle Access-Nodes durch:

  $$\overrightarrow{A}_L(v) := \bigcup_{a_1 \in \overrightarrow{A}_1(u_0)} \bigcup_{a_2 \in \overrightarrow{A}_2(u_1)} \cdots \bigcup_{u_L \in \overrightarrow{A}_{L-1}(u_{L-2})} \overrightarrow{A}_L(u_{L-1}).$$

  Das folgende Bild illustriert diesen Zusammenhang anhand drei Levels.



$\overrightarrow{A}_2(v) := \overrightarrow{A}_2(a_1) \cup \overrightarrow{A}_2(a_2)$

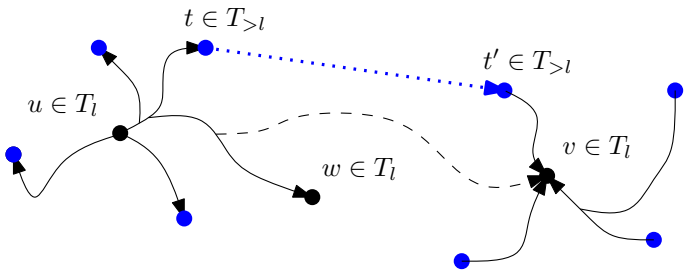
(b) Gegeben seien die Transit-Knoten $T_L \subseteq T_{L-1} \subseteq \cdots \subseteq T_0 = V$. Um die Distanztabelle zwischen Knoten $v \in T_l$ zu berechnen bedienen wir uns folgender zwei Maßnahmen.

1. Wir speichern in D_l für Knoten $u, v \in T_l$ nur die Distanz bezüglich Level l, wenn $\text{dist}_l(u,v) < \text{dist}_{\geq l}(u,v)$. Das heißt, der kürzeste Weg von u nach v führt nicht über zwei Transit-Knoten auf einem höheren Level – und kann somit *nicht* bereits durch Table-Lookups in höhere Level bestimmt werden. Sonst wird $D_l(u,v) := \infty$ gesetzt.
2. Um dies zu überprüfen führen wir die Berechnung der Distanztabellen D_l durch einen Top-Down-Ansatz durch.

   Die Distanzen zwischen Level-L-Transit-Knoten werden direkt über Many-to-Many Dijkstra-Suchen berechnet. Dies führt zu einer vollständigen Distanztabelle $D_L(u,v)$ für alle Knoten $u, v \in T_L$.

   Sind für ein beliebig aber festes $l < L$ die Distanztabellen $D_{>l}$ bereits berechnet, so berechnen wir D_l wie folgt. Wird bei der Vorwärts- bzw. Rückwärtssuchen ein Knoten $v \in T_{l'>l}$ abgearbeitet, so kann die Suche an diesem Knoten geprunt werden, da die Distanzen zwischen Transit-Knoten aus $T_{>l}$ bereits bekannt sind und durch Table-Lookups ermittelt werden können. Wird ein u-v-Weg gefunden, der nicht geprunt wurde, so folgt, dass dieser nicht über ein Paar $t, t' \in T_{>l}$ von Transit-Knoten aus höheren Leveln führt. Das heißt, $\text{dist}_l(u,t) < \text{dist}_{>l}(u,t)$, also setze $D_l(u,t) := \text{dist}_l(u,t)$.

Das folgende Bild illustriert nochmals die Vorgehensweise:



$$\text{dist}_l(u,v) \geq \text{dist}_l(u,t) + D_{>l}(t,t') + \text{dist}_l(t',v) \quad \Rightarrow \quad D_l(u,v) := \infty$$

Die Distanz $D_l(u,v)$ wird auf ∞ gesetzt, da sich der kürzeste Weg mit Hilfe von Transit-Knoten $t, t' \in T_{>l}$ eines höheren Levels berechnen lässt (Diese Transit-Knoten sind selbstverständlich Access-Nodes von u bzw. v). Andererseits benutzt der kürzeste Weg von u nach w keine zwei Transit-Nodes eines höheren Levels. Das heißt $\text{dist}_l(u,w) < \text{dist}_{>l}(u,t) + D_{>l}(t,t') + \text{dist}_l(t',w)$ für alle $t, t' \in T_{>l}$. Also wird $D_l(u,w) := \text{dist}_l(u,w)$ gesetzt.