Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Kartografie
Sommersemester 2013
Martin Nöllenburg/Benjamin Niedermann

Übungsblatt 1

Ausgabe: Dienstag, 23. April 2013
Abgabe: Bis spätestens Dienstag, 30. April 2013 um 12:00 Uhr.

Hinweis: Abgabe ist sowohl in den Vorlesungen und Übungen als auch im Raum 322 des Informatik-Hauptgebäudes möglich.

1 Einstieg

In der Vorlesung wurde folgende Beobachtung gemacht: *Für eine Gerade ℓ und Punktmenge P liegt der von ℓ weitest entfernte Punkt auf einer zu ℓ parallelen Tangente an die konvexe Hülle $CH(P)$.* Beweisen Sie diese Beobachtung formal.

Lösung: Eine Gerade t berührt $CH(P)$ tangential, falls $CH(P)$ die Gerade t schneidet und $CH(P)$ sich auf t und einer Seite von t befindet. Des weiteren gilt, dass die Punkt mit gleichem Abstand zu ℓ auf einer parallelen Gerade zu ℓ liegen. Betrachte nun für einen Punkt $p \in CH(P)$ mit maximalen Abstand zu ℓ die parallele Gerade ℓ' zu ℓ, die durch p führt. Von ℓ aus gesehen kann kein Punkt von $CH(P)$ auf der gegenüberliegenden Seite von ℓ' liegen. Damit ist ℓ' eine Tangente von $CH(P)$.

2 Kürzeste Wege

Gegeben sei ein azyklischer, gerichteter Graph $G = (V, E)$ mit Gewichtsfunktion $w\colon E \to \mathbb{R}$. Ein gerichteter s,t-Pfad $(s = v_0, \ldots, v_\ell = t)$ in G heißt *k-minimal*, falls er nicht mehr als k Kanten enthält und für alle anderen s,t-Pfade $(s = v'_0, \ldots, v'_{\ell'} = t)$ mit höchsten k Kanten gilt:

$$\sum_{j=1}^{\ell} w((v_{j-1}, v_j)) \leq \sum_{j=1}^{\ell'} w((v'_{j-1}, v'_j))$$

1. Geben Sie einen Algorithmus an, der einen k-minimalen Pfad $(s = v_0, \ldots, v_n = t)$ zwischen zwei gegebenen Knoten $s, t \in V$ berechnet.
2. Welche Laufzeit und welchen Speicherplatzbedarf hat der von Ihnen vorgeschlagene Algorithmus?
3. Begründen Sie die Korrektheit Ihres Algorithmus.

Lösung: Das Problem kann mithilfe dynamischer Programmierung gelöst werden. Sei hierzu für jeden Knoten $u \in V$ und jede Konstante h mit $0 \leq h \leq k$

$$\text{dist}(u,h) := \begin{cases} \text{Minimale Kosten eines Pfads mit Länge } h \text{ von } s \text{ nach } u \text{ mit } h \text{ Kanten.} \\ \infty \text{ falls es einen solchen Pfad nicht gibt.} \end{cases}$$

Die Lösung entspricht dann $\min_{1 \leq h \leq k} \text{dist}(t,h)$. Hierzu werden zuerst alle Werte $\text{dist}(u,h)$ mit $u \in V$ und $0 \leq h \leq k$ berechnet. Setze hierzu initial $\text{dist}(v,h) := \infty$ für alle Knoten $v \in V$ und h mit $0 \leq h \leq k$. Anschließend setze $\text{dist}(s,0) = 0$, da s von s aus mit null Sprüngen kostenfrei erreicht werden kann.

Betrachte nun eine beliebige topologische Ordnung $v_1, \ldots, v_n$ von V und iteriere durch diese Ordnung. Sei v_i der aktuelle Knoten. Falls $s = v_i$ tue nichts, ansonsten berechne für jedes h mit $1 \leq h \leq k$:

$$\text{dist}(v_i,h) := \min_{(u,v_i) \in E} \{\text{dist}(u,h-1) + w(u,v_i)\}$$

Korrektheit: Offensichtlich gilt $\text{dist}(v_i,h) \neq \infty$ mit $0 \leq h \leq k$ nur für Knoten v_i, die von s aus erreichbar sind. Folglich können wir o.B.d.A. davon ausgehen, dass alle Knoten $v_1, \ldots, v_n$ von s aus erreichbar sind und dass $s = v_1$.

IA: $i = 1$ und somit $v_1 = s$: Offensichtlich ist $\text{dist}(s,h)$ korrekt für alle h mit $0 \leq h \leq k$.

IV: Für alle v_j mit $j < i$ und alle h mit $0 \leq h \leq k$ wurde $\text{dist}(v_j,h)$ korrekt berechnet.

IS: Da $v_1, \ldots, v_n$ topologisch sortiert sind, ist v_i von s aus nur über Knoten v_j mit $1 \leq j < i$ erreichbar. Die Minimumsberechnung im Ausdruck

$$\text{dist}(v_i,h) := \min_{(u,v_i) \in E} \{\text{dist}(u,h-1) + w(u,v_i)\}$$

betrachtet also nur Knoten u für die bereits der Wert $\text{dist}(u,h-1)$ nach Induktionsvoraussetzung korrekt berechnet wurde. Offensichtlich, wenn u in $h-1$ Schritten und Kosten $\text{dist}(u,h-1)$ von s aus erreichbar ist, dann ist v_i über u mit h Schritten und Kosten $\text{dist}(u,h-1) + w(u,v_i)$ erreichbar. Es ist also folgerichtig, dass über alle eingehenden Kanten die Kante verwendet wird, die minimale Kosten verspricht.

Laufzeit: dist kann als zwei dimensionale Tabelle mit $n \times (k+1)$ Einträgen betrachtet werden. Die Berechnung eines Eintrags von dist benötigt aufgrund der Minimumsberechnung $O(n)$ Zeit. Die Tabelle kann also in $O(n^2 \cdot k)$ Zeit berechnet werden. Die abschließende Minimumsberechnung $\min_{1 \leq h \leq k} \text{dist}(t,h)$ benötigt $O(k)$ Laufzeit. Es ergibt sich also eine Laufzeit von $O(n^2 \cdot k)$ und ein Speicherplatzverbrauch von $O(n \cdot k)$.

3 Algorithmus von Hershberger und Snoeyink

Der in der Vorlesung vorgestellte Algorithmus von Hershberger und Snoeyink funktioniert nur auf einfachen Polygonzügen.

1. Geben Sie einen nicht einfachen Polygonzug an, für den das Verfahren nicht funktioniert.
2. An welcher Stelle müsste der Algorithmus abgeändert werden, damit er auch auf nicht einfachen Polygonzügen funktioniert?
3. Weist der Algorithmus für jeden nicht einfachen Polygonzug ein Fehlverhalten auf?

Lösung:

1. Siehe Folien der Übung.
2. Die Konstruktion der konvexen Hülle müsste so angepasst werden, dass auch nicht einfache Polygonzüge zu einer gültigen Lösung führen. Allerdings müsste dieses Verfahren auch weiterhin iterativ sein, da eine entscheidende Grundidee des Algorithmus auf der Wiederverwendung von bereits berechneten konvexen Hüllen aufbaut.
3. Siehe Folien der Übung.

4 Algorithmus von de Berg et al.

Betrachten Sie für diese Aufgabe den Algorithmus zur Vereinfachung von Unterteilungen, wie er in der Vorlesung vorgestellt wurde:

1. In der Vorlesung wurde behauptet, dass die Methode *Punktzuweisung* die Laufzeit $O((n+m)\log n)$ besitzt. Beweisen Sie dies und gehen Sie insbesondere darauf ein, wie die Sortierung von $P' \cup \{v_{i+1}, \ldots, v_n\}$ effizient implementiert werden kann.
2. Überlegen Sie sich basierend auf geometrischen Beobachtungen ein Beschleunigungsverfahren, mit dessen Hilfe die Punktmenge P eingeschränkt werden kann.
3. Überlegen Sie sich ein Beschleunigungsverfahren, mit dessen Hilfe die Konsistenzberechnung von shortcuts beschleunigt werden kann.

Hinweis zu 2. und 3.: Es ist nicht nötig die Laufzeit für den schlimmsten anzunehmenden Fall zu verbessern, sondern es reicht aus, sich Beschleunigungstechniken zu überlegen, die in der Praxis gut funktionieren könnten.

Lösung:

1. Entscheidend für die Laufzeit ist die im ersten Schritt stattfindende Sortierung der Punkte $P' \cup \{v_{i+1}, \ldots, v_n\}$. Anstatt alle Punkte aus dieser Menge zu sortieren, werden nur die Punkte $\{v_{i+1}, \ldots, v_n\}$ nach dem Winkel bzgl. v_i sortiert – was $O(n \log n)$ Zeit benötigt.

Basiererend auf dieser Sortierung induzieren die Punkte $\{v_{i+1}, \ldots, v_n\}$ *Buckets*, in die dann die Punkte P' in $O(m \log n)$ Zeit einsortiert werden (siehe Folien der Übung). Die Sortierung der Punkte P' innerhalb der Buckets ist zwar beliebig, die Korrektheit des Algorithmus bleibt aber davon unberührt.

2. Da die Vereinfachung von Polygonzügen C innerhalb der konvexen Hülle von C stattfindet, können all die Punkte aus P entfernt werden, die nicht in der konvexen Hülle von C liegen. Für die Vereinfachung von Landkarten wird dies im Regelfall eine gute Beschleunigung liefern.

3. Sei C der zu vereinfachende Polygonzug. Angenommen wir kennen für jeden Eckpunkt p von C den Eckpunkt von C, der am weitesten von p entfernt liegt und noch einen konsistenten Shortcut zu lässt, dann reicht es aus, die erlaubten Shortcuts bis zu diesem Punkt zu berechnen.