Algorithmische Geometrie
Wintersemester 2020/2021
Thomas Bläsius & Marcus Wilhelm

Karlsruher Institut für Technologie

Musterlösung zum Übungsblatt 1

Erstellt von Stefan Meintrup und Jean-Pierre von der Heydt

Aufgabe 1: Noch mehr Meteoriten

5 Punkte

Wie viele Pfosten sollten die Forscher für den schlimmsten Fall mitnehmen?
Um zu bestimmen, wie viele Pfosten die Forscher für das kombinierte Polygon im schlimmsten Fall benötigen, betrachten wir zunächst die maximale Anzahl an Knoten, welche das kombinierte Polygon haben kann.
Sei $K(m,n)$ die maximal benötigte Anzahl an Knoten des kombinierten Polygons und sei x die maximale Anzahl an Schnittpunkten der beiden Polygone.
Offensichtlich gilt: $K(m,n) \leq m + n + x$.
Um x zu bestimmen, nutzen wir den Fakt, dass eine Kante e ein konvexes Polygon p nur in zwei Punkten schneiden kann. Würde die Kante das Polygon öfter schneiden, dann wäre ein Teil der Kante außerhalb des Polygons. Somit wäre die Strecke zwischen zwei Punkten des Polygons nicht immer innerhalb des Polygons, was der Konvexität des Polygons widerspricht.
Also gilt: $x \leq 2 \cdot \min\{m,n\}$.
Somit ist unsere neue obere Schranke für die maximale Anzahl der Knoten:
$K(m,n) \leq m + n + 2 \cdot \min\{m,n\}$.
Offensichtlich können wir nicht mehr Pfosten benötigen, als Knoten vorhanden sind, denn sonst könnte man einfach auf jeden Knoten einen Pfosten stellen und hätte die Anzahl der Pfosten reduziert.
Nun zeigen wir noch, dass $K(m,n)$ eine scharfe Schranke für die maximal benötigte Anzahl der Pfosten ist. Für $m = 3$ und $n = 4$ gibt es zwei Polygone p_1 und p_2, sodass:
$\#BenoetigterPfosten = K(m,n)$ (Siehe Abbildung: 1). Somit gilt, dass maximal und manchmal genau $m + n + 2 \cdot \min\{m,n\}$ Pfosten benötigt werden.

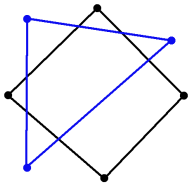

Abbildung 1: p_1 geschnitten p_2

Wie kommt man drauf?

Man kann sich leicht vorstellen, dass die max. Anzahl an benötigten Pfosten höchstens gleich der Anzahl der Knoten des kombinierten Polygons sein kann. Dann haben wir einfach nach einem Beispiel gesucht, welches zeigt, dass die Schranke scharf ist.

Aufgabe 2: Map-Overlay **8 Punkte**

Teilaufgabe a) Der Algorithmus geht grob in folgenden Schritten vor

- Bestimme alle Schnittpunkte der Kanten von G_1 und G_2
- für jeden gefundenen Schnittpunkt:
  - Lösche die Schnitt-Kanten aus dem neuen Graphen
  - Füge neue Kante und neuen Knoten in den neuen Graphen ein

Wir gehen davon aus, dass die Knoten von G_1 und G_2 in allgemeiner Lage sind, keine Kante parallel zur x-Achse verläuft und sich nicht mehr als 2 Kanten in einem Punkt Schneiden. Die Schnittpunkt der Kanten können in $\mathcal{O}((n+k)\log n)$ bestimmt werden, wobei n die Zahl aller Knoten und k die Zahl der Schnittpunkte ist. Zu jedem Schnittpunkt speichern wir uns die zugehörigen Kanten und umgekehrt. Die Schnittpunkte der Kanten werden nach absteigender y Koordinate abgearbeitet.

Um die Kantenliste L des neuen Graphen zu erhalten werden zunächst die Kantenlisten der alten Graphen konkateniert. L wird nun für jeden Schnittpunkt geupdated.

Betrachten wir nun einen konkreten Schnittpunkt. In der Abbildung 2 ist der Schnittpunkt p durch die Kante e_1 von G_1 und e_2 von G_2 entstanden. Um L zu updaten, gehen wir wie folgt vor: update e_1, e_2 (und die Twins) aus L und füge zwei neue Kanten a,b (und ihre Twins) zu L hinzu. Die geupdateten Kanten werden mit e_1^* und e_2^* dargestellt. Der Übersicht halber ist nur der Twin a' von a eingezeichnet. Wir bestimmen beispielhaft die Pointer für die neue a Kante:

- `orig(a) = orig(e1)`
- `twin(a) = a'`
- `next(a) = twin(b)`
- `prev(a) = prev(e1)`

Die anderen 7 Kanten folgen ähnlich. Zusätzlich muss noch vom ehemaligen Vorgänger von e_1 der `next` Pointer auf a gesetzt werden. Analoges für e_2.

Dies updated alle Pointer in L korrekt. Wir müssen nur noch sicherstellen, dass nun noch die Pointer der Schnittpunkte zu den entsprechenden Kanten richtig geupdated werden. Betrachte dafür folgendes Beispiel in Abbildung 3. Wir wollen verhindern, dass beim Bearbeiten des Schnittpunktes alle Referenzen zu kanten der anderen Schnittpunkte der Kante e_1 oder der Kante e_2 geupdatet

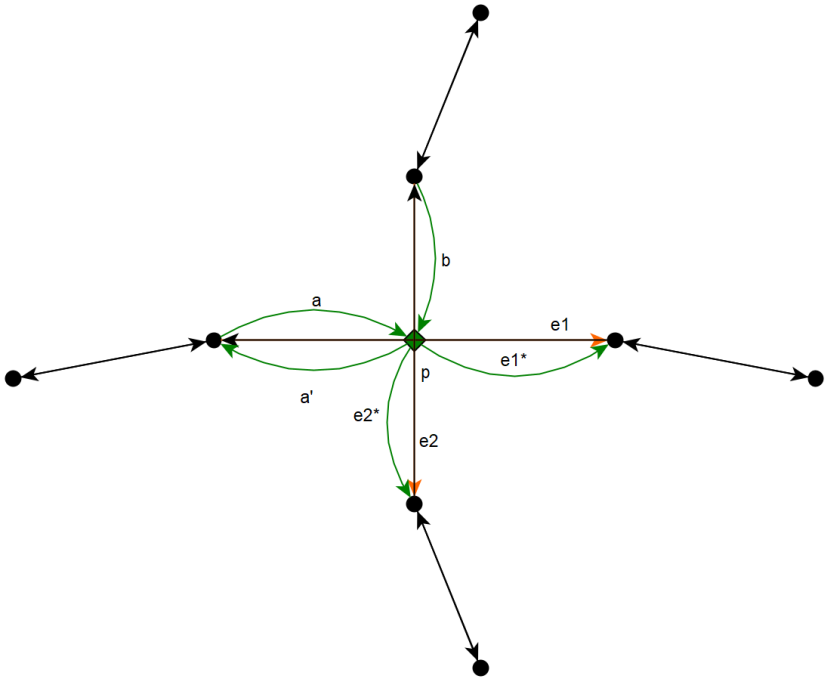

Abbildung 2: Übersicht eines Schnittpunktes p der Kanten e_1 und e_2

werden müssen. Dies würde zu einer in k quadratischen Laufzeit führen. Um dies zu umgehen, werden die Kante e_1 und e_2 nicht gelöscht sondern nur geupdated und nur zwei statt vier neue Kanten hinzugefügt. Hierbei ist wichtig, dass die Schnittpunkte der Kanten von oben nach unten abgearbeitet werden und nur der ober teil der ursprünglichen e_1, e_2 Kante durch eine neue ersetzt wird. Somit müssen keine Pointer der unbearbeiteten Schnittpunkte umgeschrieben werden.

Da das Abarbeiten jedes Schnittpunkte in $\mathcal{O}(1)$ geschehen kann, ist die Laufzeit ist durch das Bestimmen und sortieren der Schnittpuntke dominiert. Der Algorithmus läuft also in $\mathcal{O}((m + k) \log m)$, wobei m die Anzahl an Kanten der beiden Graphen ist.

Werde die Punkte nicht in allgemeiner Lage angenommen, können eine Vielzahl anderer Randfälle auftreten. Beispielsweise können beide Graphen einen Knoten auf der Selben Koordinate haben oder zwei Kanten aus verschiedenen Graphen schneiden sich in mehr als einem Punkt.

Teilaufgabe b) Wir bestimmen zuerst die Facette zu jeder Kante. Wir wählen eine beliebige unmarkierte Kante e aus der Kantenliste. Wir laufen solange `next(e)` ab, bis wir wieder bei e ankommen. Alle abgelaufenen Kanten bekommen die neue Facette f zugewiesen und werden markiert. Dies hat Laufzeit $\mathcal{O}(m)$.

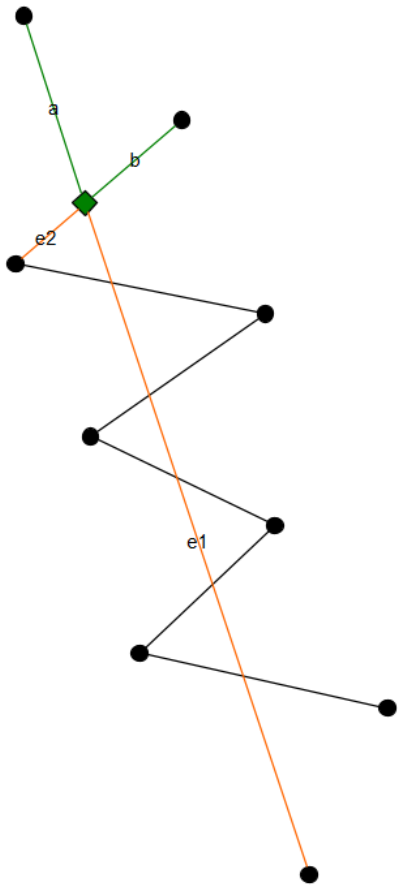

Abbildung 3: Updaten der Schnittpunkt Pointer

Um die Kinder und Eltern zu bestimmten, bestimmen wir zunächst mit einer Breitensuchen alle zusammenhängenden Teile des Graphen. Für jede Zusammenhangskomponente merken wir uns die äußere Facette (diese kann gefunden werden, indem wir uns den Knoten mit der größten y Koordinate angucken). Zu diesen müssen wir die Eltern bestimmen. Dafür schießen wir von dem Punkt mit der größten y Koordinate einen Strahl nach oben und schneiden diesen mit allen Kanten. Eine Vorberechnung kann bestimmen welche Kanten einer Zusammenhangskomponente zur äußeren und welche zur inneren Facette gehören. Betrachten wir die Schnittpunkte entlang des Strahls, so wollen wir den ersten Schnittpunkt auswählen, der die innere Facette eines Graphen schneidet und vorher keine äußere Facette geschnitten hat. Die erste innere Facette (also die mit dem geringsten Abstand zum Schnittpunkt), in der wir liegen ist der Parent unsere äußeren Facette. Wir müssen `parent` und `children` entsprechend updaten. Wird kein solcher Schnittpunkt gefunden, liegt die Facette in der Äußeren Facette.

Die Laufzeit ist wieder durch das finden und Sortieren der Schnittpunkte bestimmt. Die Anzahl an Strahlen und Kanten kann durch $\mathcal{O}(n)$ beschränkt werden. Damit ist die Laufzeit in $\mathcal{O}((n+k)\log(n))$.

Teilaufgabe c) Um die Labels für den Graphen G zu bestimmen, Müssen wir zunächst für jeden Knoten von G_1 (und umgekehrt G_2) bestimmen, in welcher Facette von G_2 er sich befindet. Dafür verwenden wir einen Sweepline Algorithmus der von oben nach unten über die Graphen scannt. Die Events sind die Knoten der beiden Graphen. Der Zustand der Sweepline wird durch eine Sortierte Folge der Kanten, welche durch diese geschnitten wird bestimmt. Zusätzlich steht für jede Kante in der Sweepline, welche Facette links beziehungsweise rechts von ihr liegt. Beim Bearbeiten eines Knotens von G_2 wird der Zustand der Sweepline entsprechend geupdated. Beim Bearbeiten eines Knotens von G_1 wird in dem Zustand der Sweepline nachgeschaut, in welcher Facette der Knoten liegt.

In G können wir dann zwischen zwei Arten von Knoten unterscheiden: Knoten welche Knoten aus G_1 oder G_2 entsprechen und Knoten die durch einen Schnittpunkt entstanden sind. Jede Kante in G entspricht einer Kante aus G_1 oder G_2. Wir merken uns an Kanten aus G an welchen Facetten/Labels sie urprünglich anlagen. Um für eine Facette f aus G zu entscheiden, welche Labels zu ihr gehören, wählen wir eine beliebige Kante e auf der Facette. Das eine Label von f erhalten wir aus dem entsprechenden Label von e. Das andere Label erhalten wir, indem wir uns einen beliebigen Endpunkt von e ansehen.

- Ist dieser Endpunkt ein Knoten, welcher einem Knoten aus G_1 oder G_2 entspricht, können wir durch die Vorberechnung bestimmen, in welcher Facette f' dieser liegt. Das Label von f' ist das gesucht Zweite Label.
- Ist der Endpunkt ein Knoten, welcher durch einen Schnitt entstanden ist, können wir über das navigiern mittels `prev(e)` (oder `next(e)`) das Label der anliegenden Kante bestimmen. Dies ist das gesuchte zweite Label.

Das Berechnen der Facettenlabel für eine Facette geschieht in $\mathcal{O}(1)$. Die Laufzeit ist also durch die Vorberechnung des Sweepline Algorithmus bestimmt. Die Laufzeit beträgt also $\mathcal{O}((n+m)\log(n))$.

Wie kommt man drauf?

Es ist sehr schnell ersichtlich, dass die Schnittpunkte von G_1 und G_2 eine besondere Bedeutung für den Graphen G haben. Wenn man sich einen Schnittpunkt aufzeichnet wird auch klar, wie man diesen behandeln muss. Es ist nur wichtig nicht den Überblick bei den ganzen Kanten und Pointern zu verlieren.

Der Kniff bei Teilaufgabe b) besteht darin die erste Facette zu finden in der man enthalten ist und nicht irgendeine. Aber auch dies ist durch schießen von genug Strahlen und Sortieren möglich.

Teilaufgabe c) wird dadurch erschwert, dass Graphen nicht zusammenhängend sein müssen und einige Facetten keine Schnittkanten mit den anderen Graphen besitzen. Hätte jede Facette eine

Schnittkante, könnten wir uns die Informationen der beiden Labels an dieser extrahieren. Deswegen greifen wir hier etwas tiefer in die Trickkisten und berechnen uns mit einem Sweeplinealgorithmus die fehlenden Informationen vor.

Aufgabe 3: Boolsche Operationen auf Polygonen **2 Punkte**

Da Polygone sich nicht als Ausgabe eignen, ist die Ausgabe unseres Algorithmus ein geometrischer Graph. Die Eingabe interpretieren wir ebenfalls als geometrischen Graphen. Wir verwenden das Ergebnis der Berechnung von Aufgabe 2, Teilaufgabe c). Dabei weisen wir den Facetten das Label äußenöder ïnnenßu.

- Schnitt: Füge alle Facetten zur Ausgabe hinzu, die in zwei inneren Facetten enthalten sind
- Vereinigung: Füge alle Facetten zur Ausgabe hinzu, die in mindestens einer inneren Facette enthalten sind
- Symmetrische Differenz: Füge alle Facetten zur Ausgabe hinzu, die in genau einer inneren Facette enthalten ist.

Es sei angemerkt, dass die ausgegebenen Facetten nur aus einer inneren, aber keiner äußeren Facette bestehen. Diese muss noch ergänzt werden. Außerdem ist es beim Bilden der Vereinigung möglich, dass geometrische Graphen entstehen, die innere Kanten besitzen, wie in Abbildung 4.

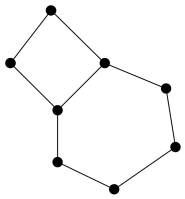

Abbildung 4: Geometrischer Graph der aus einer Vereinigung aus Viereck und Sechseck entstehen kann

Wie kommt man drauf?

Diese Aufgabe zu lösen kann auf den ersten Blick erschlagend wirken. Die Punktzahl verrät aber, dass sie nicht allzu schwer sein kann. Tatsächlich haben wir schon einen Großteil der Arbeit in Aufgabe 2 Teilaufgabe c) geleistet. Wenn man etwas drüber nachdenkt kommt man auch recht schnell darauf, dass durch das richtige Setzen von Labels Aufgabe 3 leicht lösbar ist.

Aufgabe 4: Gefährliche Wände **5 Punkte**

Wir verwenden einen Sweepline Algorithmus, welcher die Start- und Endpunkte der Geraden als Events verwendet. Die Sweepline rotiert als Strahl um den Punkt p. Der Zustand der Sweepline wird durch die Reihenfolge der Strecken, welche von ihr geschnitten werden, beschrieben.

1. Bestimme einen Strahl s von p nach oben.
2. Sei p_{ij} einer der beiden Punkte welche Wand i bestimmen, mit $i \in \{1,2,...,n\}$ und $j \in \{1,2\}$. Bestimme $\forall p_{ij}$ den Winkel ω zwischen den beiden Strahlen s und dem Strahl s', welcher auch in p anfängt, aber in Richtung p_{ij} verläuft. Speichere das Paar p_{ij} und ω ab.
3. Lege eine Liste an, welche alle p_{ij} enthält und sortiere diese nach ω.
4. Starte beim ersten Punkt p_{ij} der Liste und erzeuge eine Strecke von p nach p_{ij}. Schneide diese Strecke mit allen Wänden und füge alle geschnittenen Wände in eine, nach dem Abstand zu p sortierte, Folge ein.
5. Markiere die erste Wand in der sortierten Folge als beleuchtet.
6. Entferne so lange den nächsten Punkt p_{ij} aus der Liste der sortierten Punkte, bis diese leer ist.
    - Erster Fall: Die zu p_{ij} gehörige Wand W startet an diesem Punkt. Speichere W in der sortierten Folge der Wände. Wenn W ganz vorne in der sortierten Folge ist, dann markiere W als beleuchtet.
    - Zweiter Fall: Die zu p_{ij} gehörige Wand W endet an diesem Punkt. Entferne W aus der sortierten Folge. Wenn W die vorderste Wand war, dann markiere die nun vorderste Wand als beleuchtet (falls diese existiert).
7. Gebe alle nicht beleuchteten Wände aus

Die Laufzeit wird durch das Einfügen und Löschen der Wände und das Sortieren der Punkte nach dem Winkel dominiert. Laufzeit: $\mathcal{O}(n \cdot \log(n))$.

Wie kommt man drauf?

Es ist klar, dass die Wand, welche am nächsten zu zu p ist, beleuchtet sein muss. Durch eine rotierende Sweepline kann man eine Punktlichtquelle sehr natürlich modellieren. Wichtig ist allerdings, dass man den Sweepline Algorithmus am Anfang richtig initialisiert.