Algorithmische Geometrie
Wintersemester 2020/2021
Thomas Bläsius & Marcus Wilhelm

Karlsruher Institut für Technologie

Musterlösung zum Übungsblatt 3

Erstellt von Carina Weber und Jakob Wagenblatt

Aufgabe 1: Pizza-Guillotine

15 Punkte

Teilaufgabe (a) Wir nehmen allgemeine Lage an, d.h. die Klinge fällt nicht genau auf eine Kante oder einen Eckpunkt.

Vorberechnung: Erstelle ein Array A, welches das Polygon repräsentiert, indem:

- der Punkt p_1 mit kleinster x-Koordinate als erstes Element in A eingefügt wird
- die weiteren $n - 1$ Elemente hinzugefügt werden, indem das Polygon im Uhrzeigersinn abgelaufen wird
- der Punkt p_1 erneut am Ende des Arrays eingefügt wird

Dies ist in $\mathcal{O}(n)$ möglich, da jeder Punkt nur einmal, beziehungsweise im Fall von p_2 zweimal, in das Array eingefügt wird.

Anfrage: Eine Anfrage kann nun in $\mathcal{O}(\log n)$ durch einen rekursiven Algorithmus beantwortet werden. Der verwendete Algorithmus ist eine modifizierte binäre Suche mit linken Rand l und rechtem Rand r, sowie einem Flag, ob bereits ein Schnittpunkt gefunden wurde. Initial wird $l = A[0]$ und $r = A[-1]$ gesetzt.

Setze $m = \lfloor \frac{l+r}{2} \rfloor$ und ziehe eine Linie von $A[l]$ zu $A[m]$. Betrachte zuerst den Fall, dass noch kein Schnittpunkt gefunden wurde:
Schneidet $\overline{A[l]A[m]}$ die Gerade: Setze das Flag und starte zwei Suchen in den Intervallen $[l, m]$ und $[m, r]$. Gebe die beiden gefundenen Schnittpunkte zurück.
Falls sie die Gerade nicht schneidet berechne den Schnittpunkt p der Geraden mit der Normalen von $\overline{A[l]A[m]}$ durch l. Fall p links von $\overline{A[l]A[m]}$ liegt, setze $r = m$. Ansonsten setze $l = m$.

Betrachte nun den Fall, dass bereits ein Schnittpunkt gefunden wurde.
Falls $\overline{A[l]A[m]}$ die Gerade schneidet:

- falls $l + 1 = m$: $\overline{A[l]A[m]}$ ist eine Kante des Polygons, gebe den Schnittpunkt zurück
- sonst: setze $r = m$

Falls die Linie die Gerade nicht schneidet setze $l = m$.

Der Algorithmus bricht ab, wenn $l + 1 = r$ gilt. Falls noch kein Schnittpunkt gefunden wurde, schneidet die Gerade das Polygon nicht. Falls bereits ein Schnittpunkt gefunden wurde, schneidet die Gerade die Kante $\overline{A[l]A[r]}$ des Polygons und der Schnittpunkt wird zurückgegeben.

Laufzeit: Da der Suchraum in jedem rekursivem Aufruf halbiert wird und die Suche nur einmal beide Suchräume betrachtet, liegt die Laufzeit in $\mathcal{O}(2 \log n) = \mathcal{O}(\log n)$

Wie kommt man drauf?

Die Laufzeitanforderung zusammen mit der Sortierung der Kanten nach Winkel in einem konvexen Polygon legt eine rekursive binäre Suche nahe.

Teilaufgabe (b) Wir nutzen im Folgenden aus, dass eine konvexe Hülle eine Sortierung der Kanten der Hülle nach Winkel zur y-Achse liefert.

Vorberechnung: Von innen nach außen gehend speichere für alle konvexen Hüllen ein Array aus allen Kanten sortiert nach Winkel zur y-Achse, beginnend mit der Kante mit dem kleinsten Winkel größer null zur y-Achse. Kaskadiere hierbei die Werte der inneren Hüllen nach außen. Dies ist in $\mathcal{O}(n)$ möglich, da das Mergen sortierter Folgen in Linearzeit möglich ist.

Anfrage: Suche für beide Orientierungen der Gerade die Kante mit geringster Winkelabweichung. Die Gerade schneidet das Polygon genau dann, wenn sie zwischen diesen Kanten verläuft.
Falls die Gerade das Polygon nicht schneidet sind zwei Fälle zu unterscheiden: Das gesamte Polygon liegt auf der Ofenseite, dann werden alle Zutaten zurückgegeben oder es liegt nicht auf der Ofenseite, dann wird die leere Menge zurückgegeben.
Falls die Gerade das Polygon schneidet, so liegt mindestens einer der Endpunkte der beiden Kanten auf der Ofenseite. Laufe dann von dieser Zutat aus in beide Richtungen solange den Rand der konvexen Hülle ab, bis man nicht auf der Ofenseite ist oder die gesamte Hülle abgelaufen ist und gebe alle gefundenen Zutaten aus. Finde anschließend mithilfe des fractional cascading in konstanter Zeit die Kante auf der folgenden Hülle, deren Winkel ebenfalls die geringste Abweichung aufweist. Verfahre hier analog.

Laufzeit: Die Kanten mit der geringsten Abweichung der beiden Orientierungen der Gerade können mit binärer Suche jeweils in $\mathcal{O}(\log n)$ gefunden werden. Das Finden der korrespondierenden Kanten auf den inneren konvexen Hüllen ist durch fractional cascading in konstanter Zeit möglich. Da nur solange der Rand der Hülle abgelaufen wird, bis eine Zutat nicht mehr auf der Ofenseite liegt ergibt sich eine Gesamtlaufzeit in $\mathcal{O}(\log n + k)$.

Wie kommt man drauf?

Auch hier gibt die Laufzeit Aufschluss auf das Lösungsverfahren (fractional cascading). Der wichtigste Schritt ist zu realisieren, dass die beiden zu der Geraden „parallelen“ Kanten betrachtet werden müssen, um das Problem effizient zu lösen (und man diese Information in konstanter Zeit nach innen propagieren kann). Danach nutzt man wie in a) nur die Sortiertheit einer konvexen Hülle nach Winkel aus.

Teilaufgabe (c) Wir nutzen auch hier die Eigenschaft aus, dass eine konvexe Hülle eine Sortierung der Kanten der Hülle nach Winkel zur y-Achse liefert.

Vorberechnung: Sortiere die Punkte, sodass die konvexe Hülle des Polygonzugs in Linearzeit berechnet werden kann. Dies ist in $\mathcal{O}(n \log n)$ möglich.
Berechne in Linearzeit die konvexe Hülle des gesamten Polygonzugs, wobei die Kanten der konvexen Hülle gerichtet gespeichert werden. Teile den Polygonzug in Pfade, welche sich einen Endpunkt teilen und berechne die konvexen Hüllen der „linken“ und „rechten“ Teilmenge. Teile so lange, bis die konvexe Hülle nur noch 2 Knoten und damit deren gemeinsame Kante umfasst. Baue aus diesen Hüllen einen Binärbaum, wobei die konvexe Hülle des gesamten Polygonzugs die Wurzel ist. Bestimme die Winkel der Hüllen-Kanten zur y-Achse (wie bei (b)) und kaskadiere die Werte nach oben. Dies ist ebenfalls in $\mathcal{O}(n)$ möglich.

Anfrage: Richte die Gerade beliebig und finde den Winkel der Geraden zur y-Achse. Starte dann an der konvexen Hülle des gesamten Polygonzugs und verfahre wie folgt:

Finde die Kante der aktuellen Hülle, welche parallel zur Geraden ist bzw. finde die Kanten welche die geringste abweichende Links-/ bzw. Rechtsdrehung von der Geraden haben. Betrachte einen beliebigen Endpunkt der parallelen Kante bzw. den gemeinsamen Eckpunkt der nur gering abweichenden Kanten. Richte die Gerade entgegengesetzt und verfahre analog.

Befinden sich die beiden Knoten auf der gleichen Seite der Geraden, so wird die konvexe Hülle, welche sie repräsentieren, nie geschnitten. Somit werden auch die linke und rechte Teilhülle und keine Kante innerhalb geschnitten.

Befinden sich die Knoten auf unterschiedlichen Seiten der Geraden, so existiert zumindest ein Schnitt mit einer Kante innerhalb dieser konvexen Hülle. Befinden sich nur noch 2 Knoten in der konvexen Hülle, so befindet sich nur noch die Kante zwischen ihnen in der konvexen Hülle. Diese wird geschnitten, die Anzahl der gefundenen Schnittpunkte kann um eins erhöht werden. In jedem anderen Fall bestimme durch fractional cascading die entsprechenden Kanten der linken/ rechten Teilhülle und verfahre analog.

Laufzeit: Finden der Kanten mit der geringsten Abweichung der beiden Orientierungen zur Gerade können mit binärer Suche jeweils in $\mathcal{O}(\log n)$ gefunden werden. Das Finden der korrespondierenden Kanten auf den inneren konvexen Hüllen ist durch fractional cascading in konstanter Zeit möglich.

Wir betrachten hier $k > 0$, falls $k = 0$ besteht die Anfrage nur aus der binären Suche, die in $\mathcal{O}(\log n)$ erfolgt. Der Binärbaum aus konvexen Hüllen hat insgesamt n Blätter, von denen k gesucht werden. Wir betrachten den Teilbaum aus diesen k Blättern und den $k - 1$ Verzweigungen, die zu ihnen führen. Im schlimmsten Fall sind diese Verzweigungen möglichst weit oben, sodass noch $k \log n - k \log k = k(\log n - \log k)$ zusätzliche Knoten im Teilbaum liegen, deren Unterbäume dann nicht weiter betrachtet werden müssen. Insgesamt hat dieser Teilbaum t damit eine Größe von $|t| \leq 2k - 1 + k \log n - k \log k$.
Die Laufzeit setzt sich dann aus der ersten binären Suche und dem Ablaufen aller Knoten des Teilbaums zusammen. Wir setzen $k = k + 1$ um eine gemeinsame obere Schranke für den Fall $k = 0$

und $k > 0$ angeben zu können.

$$\begin{aligned}\mathcal{O}(\log n + |t|) &= \mathcal{O}(\log n + 2(k+1) - 1 + (k+1)\log n - (k+1)\log(k+1)) \\ &= \mathcal{O}((k+1)\log n - (k+1)\log(k+1)) = \mathcal{O}((k+1)(\log n - \log(k+1))) \\ &= \mathcal{O}\left((k+1)\log\frac{n}{k+1}\right)\end{aligned}$$

Wie kommt man drauf?

In dieser Aufgabe ist das Wichtigste zu realisieren, dass man einen binären Suchbaum aus konvexen Hüllen aufbauen muss. Dazu muss man sich bewusst machen, dass, gegeben eine Sortierung der Punkte, eine konvexe Hülle eines Polygons in $\mathcal{O}(n)$ berechnet werden kann. Zudem sollte die Laufzeit der Anfrage Aufschluss darüber geben, dass auch hier wie bei b) „parallele " Geraden fractional cascading verwendet wird, und dass durch Ausnutzen der Baumstruktur, diese Laufzeit verringert werden kann.

Aufgabe 2: Halbe 2D-Bereichsanfragen **5 Punkte**

Wir verwenden einen Sweep-Line-Algorithmus, der über alle Punkte in x-Richtung iteriert und den geometrischen Graphen schrittweise von links nach rechts aufbaut. Der Sweep-Line-Status verwaltet die Punkte, deren vertikalen Kante noch nicht vollständig nach rechts von anderen Kanten verdeckt ist in einem Stack. Die Punkte im Stack sind hierbei absteigend nach y-Koordinate sortiert.

Sei p der aktuell betrachtete Punkt. Wir schießen von p einen Strahl vertikal nach oben. Solange die y-Koordinate des obersten Element q des Stacks größer als die y-Koordinate von p ist, entfernen wir dieses und fügen eine horizontale Kante von q zum Strahl ein. Dieser wird dabei in eine obere und untere Kante geteilt. Ist der Stack danach leer, fügen wir eine horizontale Kante von p ein, die nach links ins Unendliche geht. Sei ansonsten wiederum q das oberste Element des Stacks, das dann tiefer als p liegt. Wir fügen eine horizontale Kante von p zur vertikal nach oben ausgehenden Kante von q ein, die diese in zwei vertikale Teile aufteilt. Anschließend wird der Punkt p auf den Stack gelegt und der nächste Punkt wird abgearbeitet.

Nachdem alle Punkte abgearbeitet sind, wird von allen auf dem Stack verbliebenen Punkte eine horizontale Kante eingefügt, die nach rechts ins Unendliche geht.

Laufzeit: Es wird einmal über alle Punkte iteriert. Für jeden Punkt wird nur eine horizontale Kante nach links, eine vertikale Kante nach oben und, nachdem das Element vom Stack genommen wurde, eine horizontale Kante nach rechts eingefügt. Das Einfügen dieser Kanten ist in konstanter Zeit möglich. Jeder Knoten wird nur einmal auf den Stack gelegt und herunter genommen. Somit ist die Gesamtlaufzeit des Algorithmus in $\mathcal{O}(n)$.

Wie kommt man drauf?

Ein Sweep-Line-Algorithmus sollte hier der erste Gedanke sein, vor allem da die Punkte bereits nach x sortiert sind. Zusammen mit dem Kniff, die noch nicht abgearbeiteten Knoten auf einem Stack zu speichern, liefert das die Lösung.