Algorithmische Geometrie

November 3, 2016

1 Konvexe Hüllen

1.1 Konvexe Hülle von Punktmengen

<u>Definition</u> Sei $S \subseteq \mathbb{R}^2$ eine Punktmenge in der Ebene. S heißt konvexe Hülle, genau dann, wenn $\forall p, q \in S$: $\overline{pq} \subseteq S$ wobei $\overline{pq}$ eine Gerade von p nach q ist.
Die Konvexe Hülle CH(s) einer Menge $S \subseteq \mathbb{R}^2$ ist die kleinste (in Benzug auf Inklusion) konvexe Menge die S enthält.

Eingabe: n Punkte, $S = \{q_1, .., q_n\}$
Ausgabe: Ecken $p_1, .., p_k$ der Konvexen Hülle CH(s). Wir wissen, dass $p_i \in S$ für $i = 1, .., k$. Ausgabe als Folge gegen den Uhrzeigersinn entlang des Randes von $CH(S) \Rightarrow \overline{p_i p_{i+1}}$ Randsegmente.
<u>Komplexität des Problems</u>: Satz: Die Berechnung der Konvexen Hülle von n Punkten im $\mathbb{R}^2$ ist mindestens so schwer wie das Sortieren von n reelen Zahlen.
Beweis: Reduktion des Sortierens auf CH. Sei CONVEX_HULL(S) ein ALgorithmus für CH. Zeige, wie man diesen Algorithmus verwenden kann, um n reele Zahlen $x_1, .., x_n$ aufsteigend zu sortieren.
Betrachte die Punktmenge $S = \{(x_i, x_i^2) | i = 1, ..., n\}$. CH(S) liefer alle Punkte in S gegen den Uhrzeigersinn zyklisch sortiert. Wandle die zyklisch sortierte Folge in Linearzeit in eine von "rechts" sortierte Folge um (X-Koordinate aufsteigend sortiert).
Folgerung: Die Komplexität des konvexe Hülle Problems ist Ω(n log n) (untere Schranke).

1.1.1 Gift-Wrapping

In $\mathbb{R}^3$ oder $\mathbb{R}^2$
Idee: Extrempunkt suchen, Strahl anlegen und drehen bis er einen weiteren extremen Punkt erreicht.
Lexikographische Ordnung von Punkten p und q $p = (p_x, p_y), p <_{xy} q \Leftrightarrow p_x < q_x \vee (p_x = q_x \wedge p_y < q_y)$.
Beobachtung: Der min/max Punkt in der (xy) oder (yx)-Ordnung ist eine Ecke der konvexen Hülle.
Idee für Algorithmus: $S = \{q_1, .., q_n\}$ mit Ecken $p_1, .., p_n$.
Startpunkt $p_1 \leftarrow min_{xy}(S)$ (unten links).
Wie finet man p_2: 1. Betrachte horizontalen Strahl nach rechts, der in p_1 startet. 2. Drehe diesen gegen den Uhrzeigersinn bis er auf einen Punkt von S trifft. Wiederhole vom so gefundenen p_2 aus.
Schritt 2 benötigt $\mathcal{O}(n)$.
Worst-case: h=n (Anzahl Ecken): $\mathcal{O}(n^2)$, best-case: h konstant.
Details der Implementierung:
1. Vergleiche in der linearen Suche: Winkelvergleich, bei Gleichheit Entfernung. Besser: Statt Winkel verwenden wir Orientierung.
Definition: Orientation-Prädikat:
Gegeben sind drei Punkte $a, b, c \in \mathbb{R}^2$.
orientation(a,b,c)=-1 $\Rightarrow$ c liegt rechts der Gerade a,b
orientation(a,b,c)=0 $\Rightarrow$ a,b,c liegen auf einer Gerade.
orientation(a,b,c)=1 $\Rightarrow$ c liegt links der Gerade a,b

1.1.2 Graham-Scan

Trick: Lege am Anfang p_n und p_1 auf den Stack. Dann haben wir immer ≥ 2 Punkte auf dem Stack.
Der komplette Algorithmus:
Vorbedingung: 1. $p_2, ..., p_n$ sind aufsteigend nach xy-Ordnung sortiert. 2. p_i liegt oberhalb bzw. auf der Geraden durch p_2 und p_n

```
UPPER_HULL(p_1, ..., p_n)
        Stack S
        s.push(p_n)
        s.push(p_1)
        s.push(p_2)
        for i=3 to n-1 do
                a <- S.top()
                b <- S.top_pred()
                while (orientation(b,a,p_i) ≥ 0 do
                        S.pop()
                        a <- b
                        b <- S.top_pred()
                od
                S.push(p_i)
        od
        return S
```

Laufzeitanalyse:
Beobachtung: Jeder Punkt wird genau einmal auf den Stack gepusht. Jeder Punkt wird höchstens einmal vom Stack entfernt.
$\Rightarrow$ DIe innere Schleife wird insgesamt höchstens n mal ausgeführt.
Laufzeit: $\mathcal{O}(n)$
Analog dazu LOWER_HULL mit orientation ≤ 0.
Vorbereitung:
1. Sortiere die Gesamtmenge S
2. Filtere S in S' und S" (Upper/Lower)
3. Berechne UPPER_HULL(S'), LOWER_HULL(S")
4. Konstruiere CH(S) aus den beiden Resultaten.
Gesamtlaufzeit inklusive Sortieren: $\mathcal{O}(nlog(n))$ plus der Rest $\mathcal{O}(n)$
Satz (Graham):
1. Die konvexe Hülle von n Punkten in $\mathbb{R}^2$ kann in Zeit $\mathcal{O}(nlog(n))$ berechnet werden.
2. Die Laufzeit reduziert sich auf $\mathcal{O}(n)$, falls die Punkte sortiert sind.
Bemerkungen:
1. Varianten: Gleichzeitig obere und untere Hälfte berechnen (2 Stacks). Oder gesamte Hülle in einem zirkulären Scan.
2. UPPER_HULL und LOWER_HULL sind auch für sich alleine interessant.
3. Graham's Scan ist optimal, da die untere Schranke $nlog(n)$ erreicht wird.

1.1.3 Inkrementeller Algorithmus

Idee:
1. Sortiere nach xy-Ordnung
2. Betrachte die Punkte nacheinander. Finde obere und untere Tangente von neuem Punkt zu bisheriger Konvexer Hülle, ersetze die damit übersprungenen Kanten mit der jeweiligen Tangente.
Finde Berührungspunkte o und u der oberen und unteren Tangente von p_i an das Polygon CH$(p_1, ..., p_{i-1})$. Entferne alle Ecken zwischen o und u. Füge p_i nach u (vor o) ein.
Details: Darstellung der CH: Zirkuläre doppelt verkettete Liste gegen den Uhrzeigersinn, pred-Verweise im Uhrzeigersinn.
Initialisierung: CH ¡- Dreieck (p_1, p_2, p_3) unter Beobachtung der Orientierung entweder (p_1, p_2, p_3) oder (p_1, p_3, p_2). Ein Orientation-Test notwendig.

Tangenten und Berührpunkte für p_i mit $i > 3$:
Verwende Orientation um zu prüfen, ob es Knoten "oberhalb" der Geraden p_i, p_{i-1}
Pseudocode für oberen Teil:

```
p <- p_{i-1}
while !let_turn(p_i, p, CH.succ(p)) do
        p <- CH.succ(p)
od
```

Laufzeit: Beobachtung: Berechnung der Tangenten für einen Punkt p_i hat Laufzeit $\mathcal{O}(1 + \#entfernterEcken)$.
Gesamtaufwand der Tangentenberechnung linear. $\mathcal{O}(n)$