Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2012
Martin Nöllenburg/Andreas Gemsa

Übungsblatt 1 – Konvexe Hüllen

1 Einstieg

In einigen Verfahren zur Berechnung der konvexen Hülle muss bestimmt werden ob ein Punkt $p \in \mathbb{R}^2$ links oder rechts von einer gegebenen Linie liegt. In der Vorlesung wurde erwähnt, dass man das mit Hilfe der Berechnung der Determinante einer bestimmten Matrix durchführen kann. Im Folgenden soll diese Behauptung bewiesen werden.

Seien die Punkte $p = (p_x, p_y)$, $q = (q_x, q_y)$ und $r = (r_x, r_y)$ gegeben. Die gerichtete Gerade g verlaufe durch die Punkte p und q. Wir nehmen an, dass r nicht auf g liegt.

a) Beweise, dass das Vorzeichen der Determinante $\det(A)$ der Matrix

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

anzeigt ob r sich links oder rechts von g befindet.

b) Da sich die drei Punkte p, q und r in allgemeiner Lage befinden induzieren sie ein Dreieck. Zeige, dass $|\det(A)|$ der doppelte Flächeninhalt dieses Dreiecks ist.

Lösung:

Zu a): Man kann das Kreuzprodukt im 2dimensionalen analog zum 3dimensionalen definieren. Das Kreuzprodukt aus $(r - p) \times n_{pq}$, wobei n_{pq} ein Normalenvektor des Vektors $(p - q)$ ist. Das Produkt ist genau dann 0, wenn der Punkt auf der Geraden g liegt. Ansonsten ist das Vorzeichen positiv bzw. negativ je nachdem ob der Punkt r links oder rechts von g liegt.

Zu b): Erneute Anwendung liefert das gewünschte Resultat. Für eine Illustration s. Folien der ersten Übung.

2 Algorithmus zur Berechnung der konvexen Hülle

Wir betrachten nun eine alternative zu dem in der Vorlesung vorgestellten Algorithmus zu Berechnung der konvexen Hülle von P Punkten in der Ebene.

Die grundlegende Idee des Algorithmus' ist wie folgt: Bestimme den rechtesten Punkt der Eingabemenge. Nenne diese Punkt p_1. Lege eine vertikale Gerade durch p_1 und rotiere sie im Uhrzeigersinn um p_1 bis die Gerade einen zweiten Punkt, p_2, trifft. Rotiere die Gerade weiter, aber jetzt um p_2 bis die Gerade einen dritten Punkt p_3 trifft. Wiederhole das Verfahren so lange bis die Gerade wieder p_1 trifft.

a) Gebe für den Algorithmus eine Beschreibung in Pseudocode an.

b) Welche degenerierten Fälle können auftreten? Wie kann man mit diesen Fällen umgehen?

c) Zeige, dass dieser Algorithmus die konvexe Hülle *korrekt* berechnet.

d) Zeige, dass dieser Algorithmus in $O(n \cdot h)$ implementiert werden kann, wobei h die Anzahl der Punkte auf der konvexen Hülle ist.

Lösung:
Zu a):

Algorithm 1: GiftWrapping

```
   Data: Liste P von Punkten
   Result: CH(P)
 1 p[1] = rechtester Punkt aus P
 2 i = 1
 3 repeat
 4 |   i++
 5 |   NEXT = P[1]
 6 |   for j = 2 to n do
 7 |   |   if P[j] links von der gerichteten Linie durch p[i] und NEXT then
 8 |   |   |   NEXT = P[j]
 9 |   p[i] = NEXT
10 until p[i] == p[1]
```

Zu b):

1. Der rechteste Punkt ist nicht eindeutig.
   Lösung:
   Wähle den obersten Punkt.

2. In Zeile 7 des Algorithmus' (s. Algorithm 1) liegen mehrere Punkte auf der Linie durch $p[i]$ und NEXT.
   Lösung:
   Wähle aus diesen Punkt den Punkt mit größter Distanz.

Zu c):

Beweis. Sei T die Ausgabe des Algorithmus.

Angenommen es existiert ein Punkt $p[\ell]$ in T, der nicht zur konvexen Hülle $CH(P)$ gehört. Da er zu T aufgenommen wurde gibt es keinen Punkt links von der gerichteten Linie durch $p[\ell-1]$ und $p[\ell]$. Wenn $p[\ell-1]$ Teil der konvexen Hülle $CH(P)$ sein, dann muss auch $p[\ell]$ Teil der konvexen Hülle sein, da sonst dieser Punkt nicht im Inneren der konvexen Hülle wäre. Wir wissen, dass der rechteste Punkt in $CH(P)$ ist und dieser ist $p[1]$. Also kann es keinen Punkt $p[\ell]$ in T geben der nicht Teil von $CH(P)$ ist.

Angenommen es existiert ein Punkt der in $CH(P)$ aber nicht in T enthalten ist. Wir wissen, dass der rechteste Punkt aus P sowohl in $CH(P)$ als auch in T enthalten ist. Wir nehmen an, dass sowohl T als auch $CH(P)$ im Uhrzeigersinn sortiert sind. Sei dann p der erste Punkt welcher in $CH(P)$ aber nicht in T enthalten ist. Wir bezeichnen den letzten gemeinsamen Punkt von T und $CH(P)$ vor p als $p[k]$. Der Punkt p kann nicht links von der gerichteten Linie durch $p[k]$ und $p[k+1]$ liegen, da sonst der Algorithmus den Punkt zu T hinzugefügt hätte. Also muss der Punkt rechts von dieser Linie liegen. Nun müssen zwei Fälle betrachtet werden:

- Wenn $CH(P)$ den Punkt $p[k+1]$ nicht enthält, dann ist $CH(P)$ nicht die konvexe Hülle, da der Punkt $p[k+1]$ außerhalb der konvexen Hülle liegen würde. Widerspruch!
- Wenn $CH(P)$ den Punkt $p[k+1]$ enthält, dann hat $CH(P)$ einen Rechtsknick, da der Punkt p rechts von der gerichteten Linie durch $p[k]$ und $p[k+1]$ liegt. Widerspruch!

Insgesamt folgt die Korrektheit des Verfahrens, d.h., $CH(P) = T$. □

Zu d): Folgt aus dem Pseudocode. Die äußere Schleife wird $O(h)$ mal durchlaufen. Die innere Schleife wird $O(n)$ mal durchlaufen. Damit folgt die Gesamtlaufzeit.

3 Optimalität!

Von einem Algorithmus, der die konvexe Hülle einer gegebenen Punktmenge berechnet, fordern wir, dass er die Punkte als (im Uhrzeigersinn) sortierte Liste ausgibt.

a) Zeige, dass jeder Algorithmus zur Berechnung der konvexen Hülle von n Punkten eine Laufzeit von $\Omega(n \log n)$ benötigt, was bedeutet, dass der Algorithmus aus der Vorlesung optimal im Sinne der asymptotischen Laufzeit ist.

   Hinweis: Benutze, dass die *Sortierung* von n Schlüsseln (in gewissen Rechnermodellen) eine Laufzeit von $\Omega(n \log n)$ benötigt.

b) Gegeben sei ein einfaches, nicht notwendigerweise konvexes Polygon in der üblichen Listenrepräsentation. Gebe einen Algorithmus an, der die konvexe Hülle der Eckenmenge dieses Polygons in $\mathcal{O}(n)$ Zeit berechnet. Erläutere, warum dies keinen Widerspruch zum Ergebnis aus Teilaufgabe a) darstellt.

Lösung:

Zu a): Wandle die Eingabe Zahlen $E = \{e_1, \ldots, e_n\}$ in Punkte mit Koordinaten (e_1, e_1^2) um. Berechne von diesen Punkten die konvexe Hülle. Damit ist die Eingabe sortiert.

Zu b): Nutze die 'while'-Schleife aus dem Algorithmus UpperConvexHull aus der VL. Diese Lösung stellt keinen Widerspruch zu dem Ergebnis aus Teilaufgabe a) dar, da wir bereits ein Polygon, bei dem die Knoten im Uhrzeigersinn sortiert sind, gegeben haben.