Algorithmische Geometrie

February 9, 2017

1 Konvexe Hüllen

Berechnung der Konvexen Hülle in $\mathbb{R}^2$ ist genau so teuer wie sortieren ($\mathcal{O}(nlogn)$).

1.1 Orientation(p,q,r)

Definiere Funktion Orientation(p,q,r):= $sign\begin{vmatrix} px & py & 1 \\ qx & qy & 1 \\ rx & ry & 1 \end{vmatrix}$

Durch das Aufspannen eines Parallelogramms zwsichen $\overrightarrow{pq}$ und $\overrightarrow{pr}$ ergibt sich die Matrix $A = \begin{vmatrix} qx-px & qy-py \\ rx-px & ry-py \end{vmatrix}$.
Der Flächeninhalt ist die Determinante, wobei gilt:
$detA = (qx-px)*(ry-py)-(rx-px)*(qy-py)$
$sign(detA) > 0 \Rightarrow$ r liegt links der Halbgerade von p nach q
$sign(detA) < 0 \Rightarrow$ r liegt rechts der Halbgerade von p nach q
$sign(detA) = 0 \Rightarrow$ r liegt auf der Halbgerade von p nach q
Damit gilt auch für orientation(p,q,r):
$= sign((qx-px)*(ry-py)-(rx-px)*(qy-py))$
$orientation(p,q,r) > 0 \Rightarrow$ r liegt links der Halbgerade von p nach q
$orientation(p,q,r) < 0 \Rightarrow$ r liegt rechts der Halbgerade von p nach q
$orientation(p,q,r) = 0 \Rightarrow$ r liegt auf der Halbgerade von p nach q

1.1.1 Schneidende Geraden mithilfe von orientation

$if(orientation(a,b,c) \neq orientation(a,b,d) \wedge orientation(c,d,a) \neq orientation(c,d,b))$

2 Gift-Wrapping

```
q1 = minxy(S) //lineare Suche O(n)
j=1
repeat
        a = bel. aus S \ {q_j}
        forall p ∈ S \ {q_j} do
                orient = orientation(q_j, p, q)
                if orient < 0 || (orient = 0 && cmp-dist(q_j, p, q) < 0) then
                        q = p
                fi
        of
        j=j+1
        q_j=q
until a_j = a_1
l=j-1
```

Laufzeit: $\mathcal{O}(l*n)$
Worst-Case:$\mathcal{O}(n^2)$

3 Grahams Scan

```
CONVEX_HULL(S)
S.sortxy()
a = S.min
b = S.max
forall p ∈ S \ {a,b} do
        if(orientation(a,b,p)>0) S1.append(p)
        if(orientation(a,b,p)<0) S2.append(p)
od
S1.push(a)
S1.append(b)
S2.push(a)
S2.append(b)
H1 = UPPER_HULL(S1)
H2 = LOWER_HULL(S2)
H1.reverse()
Concat(H1,H2)
LoescheDuplikate(H1H2)

UPPER_HULL(S):
Stack<Point> S
S.push(qn) //letztes Element der xy-sortierten Liste
S.push(q1)
S.push(q2)
for s=3 to n-1 do
        x = S.top()
        y = S.top_pred() //2. Auf dem Stack
        while orientation(y,x,qs) ≥ 0 do
                S.pop()
                x = y
                y = S.top_pred()
        od
        S.push(qs)
od
return S
```

Laufzeit: $\mathcal{O}(nlogn)$

4 Plane-Sweep

Definiere Events: Endpunkte x_1, x_2, y der horizontalen Segmente und x, y_1, y_2 der vertikalen Segmente.
X-Struktur: Eine nach x-Koordinaten sortierte Liste von Events
Y-Struktur: Blattorientierter balancierter Suchbaum der horizontale Segmente nach deren y-Koordinate speichert.
Operationen auf X und Y dauern höchsten $\mathcal{O}(logn)$
Initialisierung: $\mathcal{O}(nlogn)$
Hauptschleife wird 2n+s mal ausgeführt (n=# Segmente, s Schnittpunkte) und kostet $\mathcal{O}(logn)$ pro Durchlauf
Gesamtlaufzeit: $\mathcal{O}((n+s)*logn)$

```
SWEEP(S)
X=∅
Y=∅
```

```
double xpos=∞
forall s ∈ S do
        X.insert(s.left)
        X.insert(s.right)
od
while(not X.empty() )
        p = X.findmin
        X.delete(p)
        xpos = p.x
        switch(p)
                case: p linker Endpunkt von s
                        Y.insert(s)
                        s1=Y.succ(s)
                        s2=Y.pred(s)
                        X.delete(s1 ∩ s2)
                        X.insert(s1 ∩ s)
                        X.insert(s2 ∩ s)
                case: p rechter Endpunkt von s
                        s2=Y.succ(s)
                        s1=Y.pred(s)
                        Y.delete(s)
                        X.insert(s1 ∩ s2)
                case: p Schnittpunkt von s' und s"
                        s1=Y.succ(s')
                        s2=Y.pred(s")
                        Y.swap(s',s")
                        X.delete(s1 ∩ s')
                        X.delete(s2 ∩ s'')
                        X.insert(s1 ∩ s'')
                        X.insert(s2 ∩ s')
                        Ausgabe "p=s' ∩ s''"
        end switch
od
```

4.1 Plane-Sweep nur horizontal und vertikal

Verwende Plane-Sweep mit folgender Event-Behandlung:

```
Horizontales Segment: s=(x1,x2,y)
x1:Y.insert(s)
x2:Y.delete(s)
Vertikales Segment: s=(x,y1,y2)
s'=y.locate(y1)
while s' * y ≤ y2 do
        Ausgabe s'
        s' ← y.succ(s')
od
```

5 Voronoi-Diagramme

5.1 Unbeschränkt = Kante

1. Sei VR(x) unbeschränkt, z.z. x ist eine Ecke von CH(S)
Annahme, x ist keine Ecke

Wenn VR(x) unbeschränkt und konvex, dann gilt: $\exists$ Strahl s, der in x startet und komplett in VR(x) liegt.
Nach Annahme: x liegt innerhalb von CH(S), dann gilt, Strahl s schneidet den Rand von CH(S) in einer Kante(y,z), $\exists p \in s$ der näher zu y oder z ist als zu x.
2. Sei x eine Ecke von CH(S)
Betrachte den Kegel K zwischen den Senkrechten auf den zu x benachbarten Kanten.
Alle Punktein K liegen näher zu x als zu allen anderen Orten -¿ VR(x) Unbeschränkt

6 Triangulierung

Algorithmus zur Berechnung einer Triangulierung von CH(S)

```
i=1
S_i=G
forall Dreiecke t in G do
        erzeuge einen Knoten n(t)
od
while S_i > 3 do
        berechne unabh. Knotenmenge I in S_i mit mindestens ⌈1/2(n/2 * 3)⌉ Knoten,
                die am Rand liegen und maximal Grad 2 haben.
        Entferne die Knoten und ihre angrenzenden Kanten aus S_i
        Trianguliere den entstandenen Graphen neu in S_{i+1}
        forall Dreiecke t in S_{i+1} not in S_i do
                erzeuge Knoten n(t)
                forall n(t') mit t' ∈ S_i und t' schneidet t  do
                        erzeuge pointer n(t) → n(t')
                od
        od
        i++
od
```

6.1 Beweis: Triangulierung Kanten/Knoten

Für eine Triangulierung einer Menge S mit n Knoten und einer CH(S) mit h Ecken gilt:
$e = 3n - 3 - h$ Die Triangulierung hat e-Kanten
$f = 2n - 2 - h$ Die Triangulierung hat f-Dreiecke

7 Point Locatoin

7.1 Streifenmethode

Zerlege S in vertikale Streifen, einen auf jedem Knoten.
Speichere die Streifen in einem Feld X[0,...,n]
Jedes Element enthält ein Feld Y mit allen Kanten, die innerhalb dieses Streifens liegen
X ist nach x-Koordinate sortiert, Y nach y-koordinate
1. Binäre Suche auf X
2. Binäre Suche mithilfe von orientation auf Y
Mit G planaraen Graphen mit n Kanten gilt $\mathcal{O}(logn)$

7.2 Triangulierung

Gesucht Punkt q, Datenstruktur D (Triangulierung)
D: Baum der die einzelnen S_i der Generierung repräsentiert.

```
if q außerhalb der Wurzel von D then
        q liegt außerhalb von G
else
        v = Wurzel
        while v kein Blatt do
                forall Knoten u mit einem Pointer v → u do
                        if q innerhalb Dreieck u then
                                v = u
                        fi
                od
        od
        q liegt im Dreieck von v
fi
```

8 Schnitt von Halbebenen (Streifenmethode)

Sei S eine Menge von Halbebenen
Laufzeit $\mathcal{O}(nlogn)$

```
INTERSECT(S):
if |S| = 1 then
        return S
else
        teile S in zwei gleich große Teile S_1, S_2
        P = INTERSECT (S_1)
        Q = INTERSECT (S_2)
        return P ∩ Q
fi
```