Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2012
Martin Nöllenburg/Andreas Gemsa

Übungsblatt 7 - Voronoi Diagramme

1 Voronoi-Zellen

Sei $\mathrm{Vor}(P)$ ein Voronoi Diagramm für die Menge P bestehend aus n Punkten. Eine einzelne Voronoi-Zelle in $\mathrm{Vor}(P)$ besteht aus maximal $n-1$ Kanten. Naiv könnte man jetzt annehmen, dass die Komplexität von Voronoi Diagrammen quadratisch ist. Allerdings kann man beweisen, dass die Komplexität nur linear ist. Zeige dazu, dass die durchschnittliche Anzahl an Kanten aller Voronoi-Zellen in $\mathrm{Vor}(P)$ nicht größer als 6 sein kann.

Lösung:
Aus der Vorlesung ist folgender Satz bekannt:

Lemma 1. *Sei $P \subset \mathbb{R}^2$ Menge von n Punkten.* $\mathrm{Vor}(P)$ *besteht aus höchstens $2n-5$ Voronoi-Knoten und $3n-6$ Voronoi-Kanten.*

Wenn es maximal $3n-6$ Voronoi-Kanten gibt und jede Voronoi-Kante zwei Voronoi-Zellen begrenzt, dann gibt es pro Punkt (= Anzahl Voronoi-Zellen) durchschnittlich höchstens $(6n-12)/n = 6-12/n$ begrenzende Voronoi-Kanten.

Beweise also Lemma 1. Dann folgt die Aussage.

1. Verwende Euler'sche Formel: $n_v - e + n = 2$
2. Passe Voronoi-Diagramm an damit man die Euler'sche Formel nutzen kann. Dazu: Füge Knoten v_∞ hinzu. Verbinde alle unendlich langen Kanten mit v_∞. Dann ist das Voronoi-Diagramm ein Graph im herkömmlichen Sinn.
3. $(n_v + 1) - e + n = 2$.
4. Jede Kante im Graph ist begrenzt durch 2 Knoten. D.h. die Summe der Grade aller Knoten gibt uns die doppelte Anzahl der Kanten. Da jeder Knoten wenigstens Grad 3 hat, folgt: $2e \geq 3(n_v + 1)$.
5. Einsetzen von $2e \geq 3(n_v + 1)$ liefert Aussage von Lemma 1.

2 Beach-Line

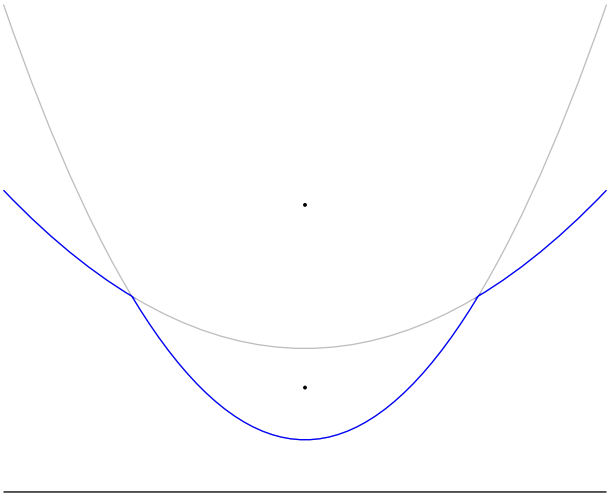
In der Vorlesung wurde erklärt, dass man ein Voronoi-Diagramm mit Hilfe einer *Beach-Line* konstruieren kann.

a) Gib ein Beispiel an, bei dem eine Parabel (festgelegt durch einen Punkt p_i) mehr als einen Kreisbogen zur Beach-Line beisteuert.

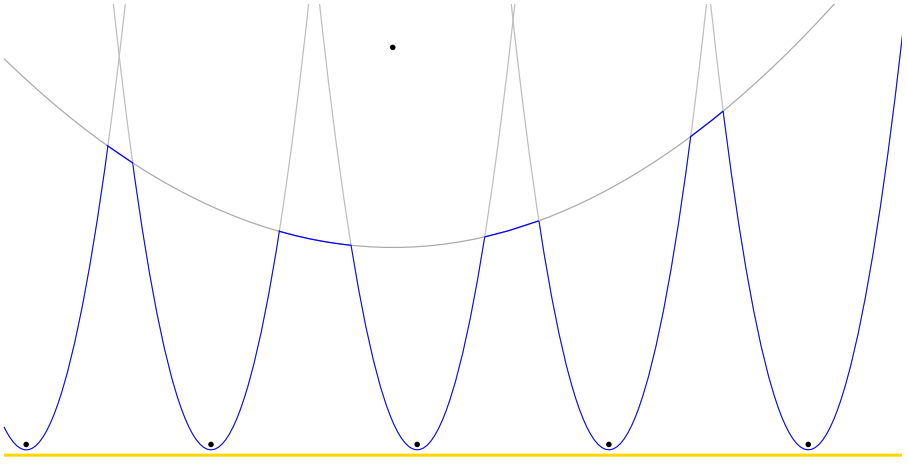
b) Ist es möglich, dass eine einzelne Parabel eine lineare Anzahl an Kreisbögen zur Beach-Line beisteuert? Erläutere deine Antwort.

Lösung:

Zu a): ja!



Zu b): ja!



3 Nächster Nachbar

Sei P eine Menge von n Punkten in der Ebene. Geben Sie einen Algorithmus an, der in $O(n \log n)$ Zeit zu jedem Punkt p in P einen anderen Punkt $a(p)$ in P bestimmt, der p am nächsten liegt.

Lösung:
Beweis Idee:

1. Zeige, dass $a(p)$ in einer benachbarten Voronoi-Zelle von p liegt.
2. Zeige, dass man für alle Punkte alle Nachbarn in $\mathcal{O}(n)$ prüfen kann.

Zu 1): Es ist folgendes bekannt: Der Bisektor zwischen zwei Knoten $p, q \in P$ gehört zu einer Voronoi-Kante des Voronoi-Diagramms Vor(P) genau dann, wenn es einen auf dem Bisektor einen Punkt $r \in \mathbb{R}^2$ gibt und der größte leere Kreis mit Mittelpunkt r hat nur Knoten p und q auf seinem Rand.

Betrachte zwei Knoten $p, q \in P$ und sei q der zu p nächste Knoten. Sei $r \in \mathbb{R}^2$ der Punkt, der mittig auf der Verbindungslinie von p und q liegt. Wir zeigen nun, dass der Kreis K mit Mittelpunkt r und Durchmesser $|pq|$ keinen Punkt aus $q' \in P$ im Inneren hat und auch keinen Punkt aus P auf seinem Kreisrand hat. Angenommen, es gäbe einen Punkt q' der der im Inneren von K oder auf dem Rand von K liegt. Von allen Punkten auf dem Kreisrand von K oder im Inneren von K hat q den maximalen euklidischen Abstand zu p. Jeder andere Punkt hat einen Abstand $d' < |pq|$. Da aber q der Punkt mit minimalem Abstand zu p ist, kann es keinen Punkt $q'' \neq q$ auf dem Kreisrand oder im Inneren von K geben.

Abbildung 1 dient als Skizze für diesen Teil des Beweises.

Angenommen, es ex. Knoten v der minimale Distanz zu p hat, aber nicht in einer benachbarten Voronoi-Zelle liegt.

1. Laut Aufgabe 1 ist jede Voronoi-Zelle im Schnitt durch weniger als 6 Kanten begrenzt.
2. Das heißt: jede Voronoi-Zelle hat im Schnitt $\mathcal{O}(1)$ Nachbarn.
3. In doppelt-verketteter Kantenliste können wir von einer Facette auf Nachbarfacetten in $\mathcal{O}(1)$ zugreifen.
4. n Voronoi-Zellen $\cdot$ $\mathcal{O}(1)$ Operationen $\Rightarrow$ $\mathcal{O}(n)$ Laufzeit.

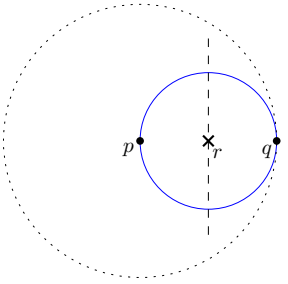


Abbildung 1: Skizze für Beweisteil 1.

4 Atom-Kraftwerke

Angenommen, du bist ein Atomkraftgegner, der möglichst weit von den dir bekannten Atomkraftwerken entfernt leben möchte. Gleichzeitig willst du aber innerhalb eines von dir bevorzugten Gebietes wohnen. Diese Situation kann man wie folgt formalisieren: Die Menge der Atomkraftwerke werde durch eine Menge S von Punkten in der Ebene repräsentiert. Das von Dir bevorzugte Wohngebiet sei der Einfachheit halber durch ein Rechteck R modelliert. Gesucht ist nun ein Punkt $p \in R$ (dein Wohnort), dessen Abstand $\min_{s \in S} d(p, s)$ zum nächstgelegenen AKW maximal ist.

a) Zeige, dass jeder optimale Wohnort entweder eine Ecke des Voronoi-Diagramms $\mathrm{Vor}(S)$, eine Ecke von R, oder ein Schnittpunkt des Randes von R mit einer Kante von $\mathrm{Vor}(S)$ ist.

b) Gebe einen Algorithmus an, der in $O(n)$ Zeit einen optimalen Wohnort findet, wenn das Voronoi-Diagramm $\mathrm{Vor}(S)$ bereits bekannt ist. Hierbei ist n die Anzahl der Punkte in S.

c) Angenommen der bevorzugt Wohnort soll sich innerhalb eines gegebenen konvexen Polygons P mit m Ecken befinden. Gib einen Algorithmus mit Laufzeit $O(m+n)$ an, der unter dieser Voraussetzung einen optimalen Wohnort findet. Wir nehmen erneut an, dass das Voronoi-Diagramm $\mathrm{Vor}(S)$ bereits bekannt ist.

Lösung:
Zu a): Angenommen es ex. optimaler Punkt r der nicht den Kriterien aus a) genügt.

- r im Inneren einer Voronoi-Zelle. Nach außen 'schieben' liefert bessere Lösung ↯
- r auf einer Voronoi-Kante. In eine der beiden Richtungen schieben liefert bessere Lösung ↯
- r auf Kante von R und nicht Ecke oder Schnittpunkt mit Voronoi-Kante. Schieben in Richtung von Schnittpunkt Voronoi-Kante und R oder schieben in Richtung Eckpunkt von R liefert besseres Ergebnis ↯

Zu b): Alle Kandidaten testen (nur linear viele)

Zu c): Analoge Argumentation zu a) liefert, dass wir alle Eckpunkte des Polygons, alle Schnittpunkte des Polygons mit Voronoi-Kanten und alle Voronoi-Knoten betrachten müssen. Naive Lösung hat Laufzeit $\mathcal{O}(mn)$.

Zwei Probleme: i) Bestimmen der Schnittpunkt von P und den Voronoi-Kanten, und ii) Bestimmen aller Voronoi-Knoten im Inneren von P.

Zu i):

1. Berechne für eine Ecke p_i die Voronoi-Zelle C in der sie liegt $\mathcal{O}(n)$
2. Folge im Uhrzeigersinn dem Polygon P. Bestimme dazu Schnittpunkt der Geraden durch p_i und p_{i+1}. Dazu muss man alle Kanten die C begrenzen durchprobieren (w-c $\mathcal{O}(n)$, aber im Schnitt nur $\mathcal{O}(1)$).
3. Liegt der Schnittpunkt auf der Strecke $p_i p_{i+1}$ wiederholt sich das Verfahren (Schritt 2) mit $p_{i+1} p_{i+2}$.

4. Liegt der Schnittpunkt nicht auf $p_i p_{i+1}$. Merke die Voronoi-Kante k auf der der Schnittpunkt liegt. Wiederhole Schritt 2 mit $p_i p_{i+1}$ aber suche nach Schnittpunkt beginnend ab Kante k im Uhrzeigersinn. Dadurch wird selbst im worst-case jede Voronoi-Kante höchstens zweimal betrachtet.

Zu ii): Wir merken uns im Schritt i) alle besuchten Voronoi-Zellen und markieren diese. In jeder Voronoi-Zelle bestimmen wir die Voronoi-Knoten die wir betrachten müssen und ebenso die benachbarten Voronoi-Zellen (wir müssen hier nur die Kanten zwischen die Schnittpunkten testen). Führe dann eine Tiefensuche durch.

Da es nur linear viele Voronoi-Kanten gibt ist die Gesamtausführungszeit für Schritt 2 $\mathcal{O}(n)$. Da P $\mathcal{O}(m)$ Kanten hat ergibt sich die Gesamtlaufzeit von $\mathcal{O}(m+n)$.

5 Rückblick – kd-Trees

Die in der Vorlesung vorgestellten Datenstruktur *kd-tree* für Bereichsabfragen ist ohne Modifikation nur für feste Eingabemengen geeignet. Würde die Eingabemenge verändert (Punkte werden hinzugefügt oder entfernt) werden müsste man den kd-Tree vollständig neu berechnen.

Überlege dir wie man kd-Trees dynamisieren kann. Dabei kannst du das rebalancieren des Baums ignorieren.

Lösung:

Illustrationen für beide Operationen finden sich in den Übungsfolien der fünften Übung.

Einfügen: Die Koordinaten des einzufügenden Punktes p als Anfragerechteck verwenden um im kd-Tree bis zu einem Blatt b herabzusteigen. Füge dann einen neuen Knoten b' ein der im kd-Tree Vater von b und p wird.

Löschen: Steige im kd-Tree herab bis zum zu löschenden Punkt p. Entferne p aus dem kd-Tree und vereinige den Vater p' von p mit dem verbleibenden Kind von p'.