Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2012
Martin Nöllenburg/Andreas Gemsa

Übungsblatt 2

1 Einfaches Polygon?

In der Vorlesung wurde ein Verfahren vorgestellt mit dem man Schnitte zwischen beliebigen Segmenten in $\mathcal{O}((n+k)\log n)$ bestimmen kann. Wir betrachten hier ein verwandtes Problem. Gegeben sei ein Polygon $\mathcal{P}$. Gebe einen Algorithmus an, der überprüft ob $\mathcal{P}$ ein einfaches (d.h. schnittfreies) Polygon ist. Der Algorithmus soll eine worst-case Zeitkomplexität von $\mathcal{O}(n \log n)$ haben.

Lösung:
Modifikation des Algorithmus aus der Vorlesung: Beachte ein Polygon hat genau n Schnittpunkte (Segmente schneiden sich an Knoten des Polygons). Führe den Algorithmus aus und breche ab sobald der Algorithmus mehr als n Schnittpunkte erkannt hat und gebe aus, dass das Polygon *nicht* einfach ist. In diesem Fall ist $k = n + 1$. Damit ist die Laufzeit in $O((n + n + 1)\log n) = O(2n \log n) = O(n \log n)$.

2 Weniger Speicherplatzverbrauch

Der Algorithmus zur Bestimmung von Schnitten zweier Segmente aus der Vorlesung benötigt $\mathcal{O}(n+k)$ Speicher. Modifziere den Algorithmus so, dass der Speicherplatzbedarf auf $\mathcal{O}(n)$ reduziert wird.

Lösung:
Speichere nur die Schnittpunkte von benachbarten Kanten (bzgl. der aktuellen Position der Sweep Line). Es gibt nur $\mathcal{O}(n)$ viele solcher Punkte und der Speicherplatzbedarf sinkt auf $\mathcal{O}(n)$. Um das zu erreichen füge nur Schnittpunkte von benachbarten Kanten ein und entferne Schnittpunkte von Kanten die im Verlauf des Algorithmus keine direkt Nachbarn mehr sind. Diese Schnittpunkte werden erst dann wieder eingefügt wenn beide Kanten erneut Nachbarn werden.

3 Sweepen

Gegeben sei eine endliche Menge P von Punkten in der Ebene. Der *größte rechte obere Bereich* eines Punkes $p \in P$ ist die Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke berühren und keinen Punkt aus P in ihrem Inneren enthalten.

a) Zeige, dass der größte rechte obere Bereich eines Punktes entweder ein Quadrat oder der Schnitt zweier offener Halbebenen ist.

b) Überlege für einen Punkt $p \in P$ und jeden der beiden zugehörigen Oktanten (vgl. Abbildung 1), welche Punkte aus P im jeweiligen Oktant den größten rechten oberen Bereich von p am stärksten einschränken. Reicht die Kenntnis dieser Punkte aus den beiden Oktanten aus, um den größten rechten oberen Bereich von p zu bestimmen?

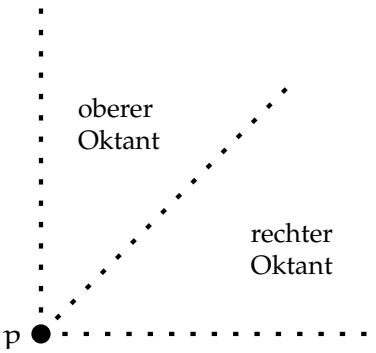

Abbildung 1: Oberer und rechter Oktant zum Punkt p

c) Gegeben sei eine Menge P von n Punkten. Berechne für jeden Punkt aus P den größten rechten oberen Bereich mit einer Gesamtlaufzeit von $\mathcal{O}(n \log n)$.

Hinweis: Sweepe die Ebene zweimal und ermittle dabei die Punkte aus Teilaufgabe b).

Lösung:
Zu a):
Wenn kein Punkt im oberen oder rechten Oktanten von p ist, dann ist der größte rechte obere Bereich (groB) offensichtlich der Schnitt zweier offener Halbebenen ist.

Seien nun der rechte und obere Oktant von p nicht leer. Die Menge der Punkte im rechten und oberen Oktanten von p sei R. Dann gibt es ein Quadrat Q im groB welches durch die Punkte p und $q \in R$ induziert wird (untere linke Ecke ist fest und vertikaler bzw. horizontaler Abstand von p und q legt die Seitenlänge fest). Es kann kein größeres Quadrat im groB geben, da so ein Quadrat den Punkt q enthalten würde. Da alle Quadrate die im groB enthalten sind kleiner als Q sind und sich mit Q die linke untere Ecke teilen muss der groB in diesem Fall ein Quadrat sein.

Zu b):
Die Menge der Punkte im rechten und oberen Oktanten von p sei R. Entscheidend ist hier der Punkt $p_r \in R$ im rechten Oktant, der den geringsten horizontalen Abstand zu p hat und analog der Punkt $p_o \in R$ im oberen Oktant, der den geringsten vertikalen Abstand zu p hat. Das ist leicht ersichtlich wenn wir die Punkte aus R, die im rechten Oktanten liegen vertikal auf die

Begrenzungslinie zwischen dem oberen und rechten Oktanten projizieren. Jedes Quadrat, dass nicht durch p_r begrenzt wird enthält wenigstens p_r und somit würde dieses Quadrat nicht den groB begrenzen (analoge Argumentation für p_o und den oberen Oktanten).

Zu c):
Lösung benötigt zwei Sweeps (s. Hinweis). Hier wird nur der Sweep für die Bestimmung der Punkte im obere Oktant beschrieben. Der zweite Sweep-Vorgang läuft analog.

Für den Algorithmus nutzen wir einen balancierten binären Suchbaum $\mathcal{B}$ in dem Knoten gespeichert werden deren groB noch nicht von einem Punkt im oberen Oktanten begrentzt ist. In diesem Suchbaum sind alle Knoten nach ihrer jeweiligen x-Koordinate sortiert. Wir sweepen nun von unten nach oben.

In jedem Schritt prüfen wir suchen wir den Punkt p' in $\mathcal{B}$ der direkt links vom aktuell betrachteten Puntk p liegt. Wenn es einen solchen Punkt p' gibt wird geprüft ob p im oberen Oktanten von p' liegt. ollte das der Fall sein haben wir die Begrenzung des oberen Oktantens von p' gefunden und entfernen p' aus $\mathcal{B}$. Wir suchen nun den nächsnten Knoten in $\mathcal{B}$ der direkt links von p liegt und wiederholen die beschriebenen Schritte. Sollte es keinen Punkt mehr links von p in $\mathcal{B}$ geben, oder sollte p nicht im oberen Oktanten von dem gefunden Punkt liegen brechen wir ab, fügen p zu $\mathcal{B}$ hinzu, und führen den sweep fort.

Laufzeit: Jeder Knoten wird genau einmal zum Baum $\mathcal{B}$ hinzugefügt und höchstens einmal wieder entfernt. Ingesamt benötigen wir für die Einfüge- und Lösch-Operationen $O(n \log n)$ Zeit. Damit wird sweepen können müssen wir auch die Punkte nach ihren y-Koordinaten sortieren was ebenfalls $O(n \log n)$ Zeit benötigt. In jedem Schritt des Sweeps müssen wir mindestens eine Suchoperation in $\mathcal{B}$ durchführen. Eventuell müssen wir weitere Suchoperationen durchführen, aber in jedem Schritt werden genau $(k+1)$ solche Operationen durchgeführt, wobei k die Anzahl der Punkte ist, die aus $\mathcal{B}$ entfernt werden. Wir schlagen die Kosten für die k Knoten den Kosten für das Einfügen/Löschen der Knoten auf. Übrig bleibt eine einzige Suchoperation pro Schritt. Diese benötigt $O(\log n)$ Zeit, da $\mathcal{B}$ ein balancierter binärer Suchbaum ist. Insgesamt folgt eine Laufzeit von $O(n \log n)$.