Übungsblatt 5

Julius Auer, Alexa Schlegel

Aufgabe 1 (Suchen in ebenen Unterteilungen):

Für eine *einfache* Datenstruktur kann durch jeden Knoten eine vertikale *und* eine horizontale Linie gezogen werden. Es entstehen dabei zwar n^2 viele Gitterzellen, da bei einer Anfrage allerdings zunächst nur entlang einer Dimension (n Reihen bzw. Spalten) und danach erst entlang der anderen Dimension gesucht wird, ist eine Zeitkomplexität von schlimmstenfalls nur $2 \cdot \log n$ zu erwarten.

Es seien im Folgenden $G = (V, E, F)$ mit $|V| = n$ eine Unterteilung der Ebene mit Knoten, Kanten und Facetten, und A ein 2-dimensionales Array der Größe $n \times n$ mit Tupeln $(x_{i,j}, y_{i,j}, f_{i,j})$ und der Eigenschaft, dass für alle $(x, y) \in \mathbb{R}^2$ gilt $(x, y) \in f_{i,j} \Leftrightarrow x_{i,j} \leq x \wedge y_{i,j} \leq y \wedge x < x_{i,j+1} \wedge y < y_{i+1,j}$. Es wird beim Erstellen der Datenstruktur also jedem Rechteck des Gitters durch alle Punkte die Facette der Unterteilung zugeordnet, die dieses Rechteck vollständig enthält.

Algorithm 1 $INIT(V, F, A)$

```
1: sort V along x-coordinate
2: i := 0
3: for each v ∈ V do
4:     x_{0,i} := v.x
5:     i+ = 1
6: end for
7: sort V along y-coordinate
8: j := 0
9: for each v ∈ V do
10:    y_{j,0} := v.y
11:    j+ = 1
12: end for
13: for each (x_{j,i}, y_{j,i}) ∈ A do
14:    f_{j,i} := f ∈ F : (x_{j,i}, y_{j,i}) ∈ f
15: end for
```

Zeitkomplexität:

- Sortieren nach x und "benennen" der Spalten (1-5) benötigt mit geeignetem Sortier-Algorithmus $O(n \cdot \log n)$
- Sortieren nach y und "benennen" der Zeilen (6-8) benötigt mit geeignetem Sortier-Algorithmus $O(n \cdot \log n)$
- Um jedem Eintrag (naiv!) eine Facette zuzuordnen (11-13), kann jeweils über die gesamte Menge der Facetten iteriert werden und in konstanter Zeit festgestellt werden, ob ein Punkt der Facette im Rechteck liegt, das von $(x_{j,i}, y_{j,i}), (x_{j+1,i+1}, y_{j+1,i+1})$ aufgespannt wird. Die Anzahl Facetten ist asymptotisch begrenzt durch die Anzahl der Knoten. Insgesamt kostet diese Zuordnung somit $O(n^3)$ Zeit. Dieser Aufwand dominiert die zuvor erwähnten Kosten.

Speicherkomplexität:

$O(n^2)$ für das Array, das den Rechtecken des Gitters jeweils eine Facette zuordnet.

Suchen wird nun durch eine Binärsuche entlang der x-Koordinate und eine zweite entlang der y-Koordinate realisiert. Im Array findet sich die zugehörige Facette. Der Algorithmus ist trivial und die Zeitkomplexität offensichtlich $O(\log n)$.

Aufgabe 2 (L_1-Voronoi-Diagramme):

In L_1-Metrik beschreiben die Punkte mit Abstand d von einem Punkt ein Quadrat mit Seitenlänge $2 \cdot d$ (Abb. 1).

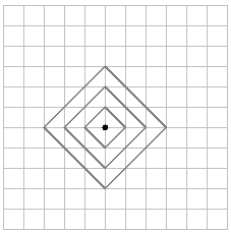

Abbildung 1: Punkte mit Abständen 1,2,3 von einem Punkt liegen auf den Ränder dieser Quadrate

Eine zwei Voronoi-Regionen trennende Kante hat nun stets eine von drei möglichen Formen, welche vom Verhältnis zwischen der X-Differenz und der Y-Differenz der Punkte bestimmt wird. Dominiert die X-Differenz, wird die Kante senkrecht "aussehen". Dominiert die Y-Differenz, wird die Kante waagerecht "aussehen".

Es sind im Folgenden die drei Fälle abgebildet, dass bei zwei Punkten die Y-Differenz (Abb. 2), die X-Differenz (Abb. 3) oder keine der beiden (Abb. 4) dominiert.

Jede Abbildung zeigt die drei untergeordneten Fälle, bei denen der in der dominanten Dimension größere Punkt in der rezesiven Dimension kleiner/gleich/größer dem anderen Punkt ist. Andere Fälle als die gezeigten sieben kann es nicht geben.

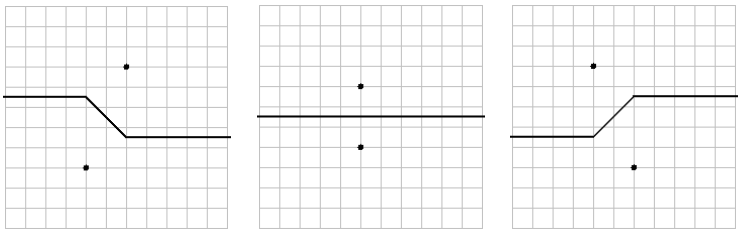

Abbildung 2: Fall 1: $\left|\frac{x_1-x_2}{y_1-y_2}\right| < 1$

Auffälig ist hier der letzte, entartete Fall, bei dem eine Menge von Punkten (rot) zu beiden Punkten equidistant ist und keine Kante sondern eine Fläche beschreibt. Wie dieser Fall im Kontext von Voronoi-Diagrammen behandelt werden sollte ist nicht aus deren Definition abzuleiten. In der Praxis sollte dieser Fall jedoch ohnehin nur relevant sein, wenn mit Ganzzahligen Koordinaten gerechnet wird.

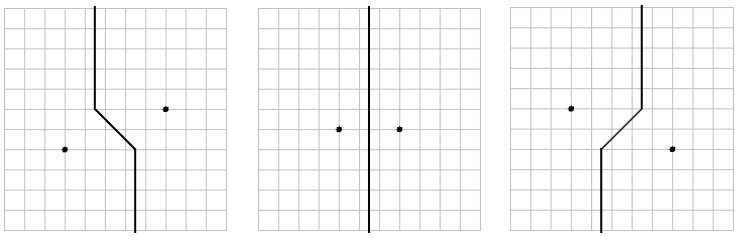

Abbildung 3: Fall 2: $\left|\frac{x_1-x_2}{y_1-y_2}\right| > 1$

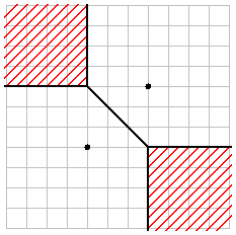

Abbildung 4: Fall 3: $\left|\frac{x_1-x_2}{y_1-y_2}\right| = 1$

Aufgabe 3 (Suche in ebenen Unterteilungen - Verallgemeinerung):

Die Lokalisierungsdatenstruktur (LDS) wurde in der Vorlesung als gerichteter azyklischer Graph (ADG) beschrieben und funktioniert für Triangulierung von G mit Dreieck als Außenfacette.

S_1 entspricht G, also der ebenen Unterteilung (eingebetteter Graph) S_n ist das äußere (umschließende) Dreieck

Ich habe es so verstanden dass in jedem Schritt die unabhänige Knotenmenge bestimmt und entfert wird, dass sind dann die S_i's' die zwischen 1 und n konstruiert werden.

Eine Knotenmenge I von G (Teilmenge) heißt unabhängig, wenn keine zwei Knoten aus I in G durch eine Kante verbungen sind. Es geht wohl in linearer Zeit zu berechnen, weil warum auch immer, wurde in VL bewiesen.

1. vorher das Ding Triangulieren
2. großes Dreieck drumrum bauen
3. wenn ein Strahl das äußerer Dreieck schneidet, dann dort einen neuen Knoten einfügen
4. Eckpunkte vom Dreieck mit Punkten verbinden oder einfach nochmal Triangulieren? oder voher schon Dreicke rumbasteln

die beiden algorithmen wurden in der VL vorgestellt, diese dann noch ein bischen erweitern.

h ... log(n)

* Erweiterung von LDS
* Algorithmen zur Suche und Konstruktion anpassen
* alle Unterteilungen den Ebene sollen unterstützt werden (mehrere unbeschränkte Facetten)
* Einzelheiten der Algorithmen beschreiben
*Vorverarbeitungszeit, Speicherbedarf, Anfragezeit, Anhängig von Anzahl der Knoten

Algorithm 2 Algorithmus zur Konstruktion

1: Sei $S_1 = G$
2: Für jedes $\triangle$ in G erzeuge einen Knoten in LDS(G)
3: i = 1
4: **while** $|S_i| > 3$ **do**
5: Berechne I
6: Entferne I aus S_i
7: Trianguliere $S_i ohne I$, das ist dann $S_i + 1$
8: für jedes neue $\triangle$ t in $S_i + 1$ erzeuge einen Knoten K(t)in LDS(G), füge eine Kante von K(t) zu allen Knoten von Dreiecken von S_i die von t geschnitten werden
9: i++
10: **end while**

Algorithm 3 Lokalisiere (q, LDS(G))

1: teste ob $q \in S_h$ falls nein gib aussenfassette zurück
2: v=Wurzel(LDS(G))
3: **while** v hat Nachfolger **do**
4: für jeden Nachfolger von u von v teste ob q im Dreiech u liegt.
5: Falls ja, setze $v = u$ und wiederhole
6: **return** q liegt im Dreieck von v
7: **end while**