Übungsblatt 5

Julius Auer, Alexa Schlegel

Aufgabe 1 (Suchen in ebenen Unterteilungen):

Sei $G = (V, E, F)$ die ebene Unterteilung mit Knoten, Kanten und Facetten, wobei die Anzahl der Kanten und Facetten asymtotisch durch die Anzahl Knoten begrenzt ist, also $|E|, |F| \in O(n), n = |V|$.

Idee ist, die Ebene vertikal entlang der y-Koordinaten der Punkte und horizontal entlang der Geraden, welche die vertikalen Trennlinien schneiden, aufzuteilen. Hierzu werden in einem Array A (auf dem später eine Binärsuche ausgeführt werden kann) Dupel (z_i, X_i) gespeichert, wobei z_i die y-Koordinate einer Trennlinie ist und $X_i = \{(e_1, f_1), ..., (e_m, f_m)\}$ ein Array (für spätere Binärsuche) in dem alle Strecken gespeichert werden, die zwischen z_i und z_{i+1} entlang einer Kante $e \in E$ verlaufen. Es kann nur $O(n)$ solche Strecken geben und aufgrund der Planarität von G werden sich auch diese Strecken nie schneiden. Dadurch teilen die Strecken die "Spalte" zwischen z_i und z_{i+1} in $O(n)$ disjunkte Teilräume, für welche die jeweils zugehörige Facette $f_j \in F$ bestimmt werden kann:

Algorithm 1 $INIT(V = \{(x_1, y_1), ..., (x_n, y_n)\}, E, A = \{(z_1, X_1), ..., (z_n, X_n)\})$

```
1: sort V by y-coordinate
2: for each y_i ∈ {y_1, ..., y_n} do
3:    z_i := y_i
4: end for
5: for each z_i ∈ {z_1, ..., z_n} do
6:    for each e ∈ E do
7:       if e intersects vertical line through z_i then
8:          f := find face directly below e and between z_i and z_{i+1}
9:          X_i ← (e, f)
10:      end if
11:   end for
12:   sort X_i by y-coordinate of left intersection
13:   copy X_i to array
14: end for
```

Speicherkomplexität ist offensichtlich $O(n^2)$ für das Array.

Zeitkomplexität ist insgesamt $O(n^3)$, gegeben durch:

- $O(n \cdot \log n)$ für Sortieren mit geeignetem Algorithmus (1-4)
- n Ausführungen der äußeren Schleife (5)
- $O(n)$ Ausführungen der inneren Schleife (6)
- in den Schleifen:
  - konstante Zeit für die Operationen (7, 9-11)
  - $O(n)$ für das Finden der passenden Facette (8) wenn naiv alle Facetten ausprobiert werden

– $O(n \cdot \log n)$ für Sortieren mit geeignetem Algorithmus (12)

– $O(n)$ zur Umwandlung einer Liste in ein Array (13)

Nach dieser Vorverarbeitung kann für einen gegebenen Punkt $p = (x, y)$ mittels Binärsuche auf $\{z_1, ..., z_n\}$ die passende "Spalte" z_i gefunden werden, und mit einer zweiten Binärsuche auf $X_i = \{(e_1, f_1), ..., (e_m, f_m)\}$ die passende Facette gefunden werden, indem on-the-fly geprüft wird ob p ober- oder unterhalb einer Strecke e_j liegt. 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).

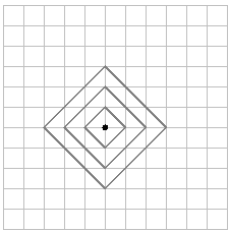

Abbildung 1: Punkte mit Abständen 1,2,3 von einem Punkt liegen auf den Ränder dieser Quadrate

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.

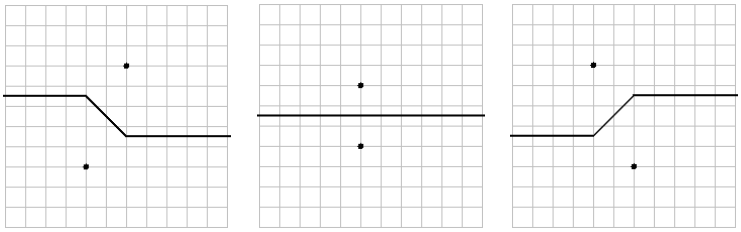

Abbildung 2: Fall 1: $\left|\frac{x_1 - x_2}{y_1 - y_2}\right| < 1$

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.

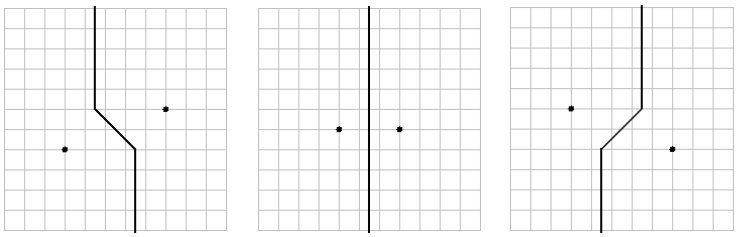

Abbildung 3: Fall 2: $\left|\frac{x_1-x_2}{y_1-y_2}\right| > 1$

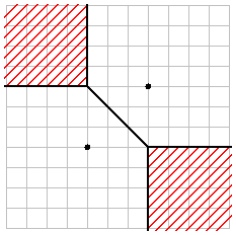

Abbildung 4: Fall 3: $\left|\frac{x_1-x_2}{y_1-y_2}\right| = 1$

Aufgabe 3 (Suche in ebenen Unterteilungen - Verallgemeinerung):

Die Lokalisierungsdatenstruktur (LDS) wurde in der Vorlesung als gerichteter azyklischer Graph (DAG) beschrieben und funktioniert für ebene Unterteilungen, wo jede Facette ein Dreieck ist, auch die Außenfacette.

Wir werden nun im ersten Schritt aus der beliebigen Unterteilung der Ebene eine Triangulierung erzeugen - mit einem Dreieck als Außenfacette. Im zweiten Schritt wenden wir darauf den in der Vorlsung vorgestellten Algorithmus zur Konstruktion der LDS an.

a) Um die vorhandene Unterteilung U ein großes Dreieck D legen, sodass alle Kanten im Inneren des Dreicks liegen.

b) Schnittpunkte von D mit Strahlen (Verlängerungen der Kanten) der Unterteilung U als Knoten in Uaufnehmen.

c) Eckpunkte des Dreiecks auch in U aufnehmen.

d) Inneres von U Triangulieren $\rightarrow G$.

e) Algorithmus zur Konstruktion der LDS(G) anwenden.

Folgendes wurde in der Vorlesung definiert:

- eingebetteter Graph G entspricht S_1
- S_h ist das äußere, umschließende Dreieck
- $h = O(\log n)$
- I ist die unabhängige Knotenmenge

Algorithm 2 Algorithmus zur Konstruktion der LDS(G)

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 von I
6: Entferne I aus S_i
7: Trianguliere $S_i \setminus I \rightarrow S_i + 1$
8: für jedes neue $\triangle$ t in $S_i + 1$ erzeuge einen Knoten K(t)in LDS(G) und 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: **if** $q \notin S_h$ **then**
2: **return** q liegt in Außenfacette
3: **end if**
4: v=Wurzel(LDS(G))
5: **while** v hat Nachfolger **do**
6: **for all** Nachfolger u von v **do**
7: **if** q ist in Dreieck von u **then**
8: $v = u$
9: **end if**
10: **end for**
11: **end while**
12: **return** q liegt im Dreieck von v

Vorverarbeitungszeit (a) dauert $O(n)$, um das min-x, max-x zu finden. (b) benötigt ebenfalls $O(n)$ wenn naiv alle Kanten betrachtet werden.
$\rightarrow O(n)$

Speicherbedarf Im schlimmsten Fall können $O(n)$ zusätzliche Knoten und genauso viele neue Kanten entstehen, die zusätzlichen Speicher benötigen.
$\rightarrow O(n)$

Anfragezeit unverändert.
$\rightarrow O(\log n)$