TMI - Cheat Sheet Complexiteiten

Matthias Kovacic

18 januari 2022

1 Les 1 - Convex Omhullende

In de eerste les zijn de volgende algoritmes besproken om de convex omhullende te berekenen:

1. Brute Force
2. (Andrew's) Incremental Algorithm
3. Graham Scan
4. Jarvis March
5. Divide-&-Conquer

1.1 Brute Force

1. Tijdscomplexiteit: $O(n^3)$ - Voor de n^2 paren moeten er nog $n-1$ punten gecontroleerd worden.

1.2 Incremental Algorithm

1. Eerst worden alle punten gesorteerd volgens x-coordinaat. Dit vraagt $O(nlog(n))$ tijd.
2. Daarna worden de punten n keer overlopen, wat $O(n)$ tijd vraagt.
3. Tijdscomplexiteit: $O(nlog(n)) + O(n) = O(nlog(n))$

1.3 Graham Scan

1. Zoek het punt met laagste y-coordinaat. Dit vraagt $O(n)$ tijd.
2. Sorteer alle punten volgens poolhoek t.o.v. het punt gevonden in stap 1. Dit vraagt $O(nlog(n))$ tijd.
3. Daarna wordt elk punt overlopen (a la Incremental Style). Dit vraagt $O(n)$.
4. Tijdscomplexiteit: $O(n) + O(nlog(n)) + O(n) = O(nlog(n))$

1.4 Jarvis March

1. Zoek het punt met laagste y-coordinaat. Dit vraagt $O(n)$ tijd.
2. Overloop n punten en bereken hun poolhoek t.o.v. het eerste punt. Zoek het punt met kleinste poolhoek. Dit vraagt opnieuw $O(n)$ tijd. Beschouw het gevonden punt nu als referentiepunt.
3. Doe dit tot je terug in het oorspronkelijke punt komt met laagste y-coordinaat. Dit vraagt $O(k)$ tijd waarbij k het aantal punten op de convex omhullende is. Elke iteratie worden er dus n punten gecontroleerd.
4. Tijdscomplexiteit: $O(n) + O(kn) = O(kn)$
5. Opmerking 1. De tijdscomplexiteit is hier *output-afhankelijk*!
6. Opmerking 2. k is hier geen constante. De $O(kn)$ term mag dus niet zomaar vereenvoudigd worden naar $O(n)$!

1.5 Divide-&-Conquer

1. Verdeel de punten in 2 helften, dit vraagt $O(n)$ tijd.
2. Bepaal de convex omhullende voor de twee helften. Dit kan met verdere opdeling of een algoritme dat eerder gezien is. Dit vraagt ongeveer $O(nlog(n))$ tijd.
3. De *merge*-stap (e.g. het zoeken van onder/bovenbruggen) kan gebeuren in $O(n)$ tijd.
4. Tijdscomplexiteit: $O(n) + O(nlog(n)) + O(n) = O(nlog(n))$

2 Les 2 - Intersecties van lijnstukken/DCEL

In de tweede les zijn de volgende algoritmes besproken:

1. Brute Force Line Intersection
2. Bentley-Ottman Algorithm
3. DCEL Overlay

2.1 Brute Force Line Intersection

1. Tijdscomplexiteit: $O(n^2)$, alle paren van lijnstukken moeten worden gecontroleerd op intersectie.

2.2 Bentley-Ottman Algorithm

1. Opmerking 1. Dit is een sweep-line algoritme. Er moet een event-queue en status bijgehouden worden. We veronderstellen deze om BST (Binary Search Trees) te zijn, wat altijd $O(log(n))$ operaties garandeert.

2. Opmerking 2. De events zijn de start/eindpunten van de lijnsegmenten. Bij n segmenten zijn er dus n start-events en n eind-events.

3. Een start-event kan worden behandelt in $O(log(n))$ tijd. Het komt n keer voor, dus alle start-events behandelen vraagt $O(nlog(n))$ tijd.

4. Eind-events zijn analoog aan start-events: $O(nlog(n))$.

5. Het andere soort events zijn de intersectie-events. Zo zijn er k (ook weer onbekend, Cfr. Jarvis March). Alle intersectie-events behandelen vraagt dus $O(klogn)$ tijd.

6. Tijdscomplexiteit: $O(nlog(n)) + O(nlog(n)) + O(klog(n)) = O((n+k)log(n))$

7. Opmerking 3. De tijdscomplexiteit is hier *output-afhankelijk*!

2.3 DCEL Overlay

1. Opmerking 1. Het algoritme zelf staat volledig uitgeschreven in het boek. Het maakt gebruik van een sweepline en van supergrafen.

2. Opmerking 2. De complexiteit van een subdivisie (e.g. DCEL) is gegeven door: $c = v + e + f$ waarbij v het aantal vertices, e het aantal edges en f het aantal faces is.

3. Tijdscomplexiteit: $O(nlog(n)+klog(n))$ waarbij $n = n_1+n_2$ met n_1 de complexiteit van de eerste subdivisie en n_2 die van de tweede. k is de complexiteit van de geconstrueerde overlay.

4. Opmerking 3. De tijdscomplexiteit is hier *output-afhankelijk*!

3 Les 3 - Veelhoek-triangulatie

In de derde les zijn de volgende algoritmes besproken:

1. Y-monotone polygon

2. Polygon triangulation

3.1 Y-monotone polygon

1. Opmerking 1. Dit is opnieuw een sweepline-algoritme. Er wordt nergens het "bewijs"van complexiteit geleverd.

2. Tijdscomplexiteit: $O(nlog(n))$ met n het aantal vertices in de polygon.

3.2 Polygon Triangulation

1. Alle vertices overlopen kost $O(n)$ tijd. Het kan bewezen worden (zie boek) dat ook de verbindingen maken maximaal lineaire tijd kost.

2. Tijdscomplexiteit: $O(n)$