ETH Zürich
Institute of Theoretical Computer Science
Prof. Rasmus Kyng
Prof. Angelika Steger

FS 2024

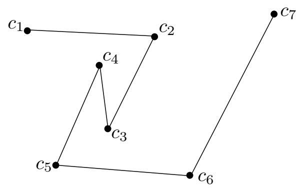
Algorithmen und Wahrscheinlichkeit
Theorie-Aufgaben 5

Lösung zu Aufgabe 1

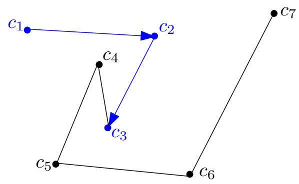
Wir erinnern uns an den Algorithmus LocalRepair aus der Vorlesung. Die Punkte werden in LocalRepair zuerst sortiert, und dann werden die unteren und oberen konvexen Hüllen berechnet. Die Laufzeit von LocalRepair ist $O(n \log n)$, inklusive dem Sortieren. In dieser Übung sehen wir Beispiele von Punktemengen, für die wir die konvexe Hülle in Zeit $O(n)$ berechnen können.

(a) Wir bekommen Punkte $s_1, s_2, \ldots s_n$, sortiert nach ihrer x-Koordinate. Wir können den Teil des Algorithmus LocalRepair nach dem sortieren direkt auf $s_1, s_2, \ldots s_n$ anwenden um die konvexe Hülle zu bekommen. Die Korrektheit wurde in der Vorlesung gezeigt. Die Laufzeit ist nur $O(n)$, da wir die Punkte nicht sortieren müssen.

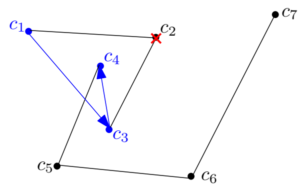
(b) Die entscheidende Annahme die wir hier ausnutzen ist, dass der Streckenzug, der von den Punkten $c_1, c_2, \ldots, c_n$ gebildet wird, sich nicht selbst schneidet. Wir bemerken zuerst, dass der Algorithmus aus (a) nicht mehr funktioniert, da er neue Überschneidungen produzieren kann, diese jedoch nicht entfernt:



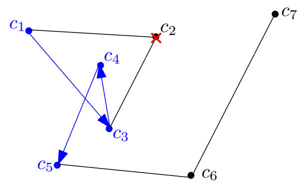
Point set C



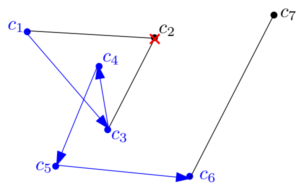
Iteration 1: Point c_2 is discarded



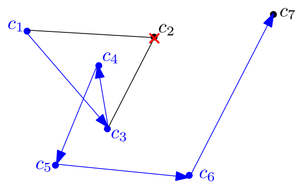
Iteration 2: Point c_4 is added, no point discarded



Iteration 3: Point c_5 is added, no point discarded



Iteration 3: Point c_6 is added, no point discarded



Iteration 3: Point c_7 is added, no point discarded

The final lower convex hull is $c_1, c_3, c_4, c_5, c_6, c_7$ which intersects itself, and is a wrong solution.

Lösung: Wir betrachten das Liniensegment c_n, c_1 und nehmen zunächst an, dass alle anderen Punkte sich links davon befinden. Wir finden nun der Reihe nach für alle i zwischen 1 und $n-1$ jeweils eine Teilfolge H_i der Punkte $c_n, c_1, c_2, \ldots, c_i$, so dass H_i ein konvexes Polygon beschreibt (jeder Punkt von H_i liegt also links des Liniensegments, welches von den beiden vorherigen Punkten gebildet wird), und alle Punkte $c_n, c_1, \ldots, c_i$ in/auf einem der Polygone H_i oder $H_i, c_{i+1}, c_{i+2}, \ldots, c_{n-1}$ liegen. Für c_n, c_1 wählen wir H_1 einfach als c_n, c_1, und sowohl c_n wie auch c_1 werden in jedem Schritt Teil von H_i bleiben, da sie extremale x-Koordinaten haben.

Für $i = 2, \ldots, n-1$ wollen wir H_{i-1} jeweils um einen weiteren Punkt c_i erweitern. Falls der neue Punkt c_i nicht im Inneren von H_{i-1} ist, dann machen wir c_i Teil von H_i. Sei p der zweitletzte Punkt von H_{i-1} und q der letzte Punkt von H_{i-1}. Da sich der Streckenzug $c_n, c_1, c_2, \ldots, c_i$ nicht selbst schneidet, liegt c_i in diesem Fall entweder rechts von p, q oder rechts von q, c_n. Daher reicht es aus, diese beiden Liniensegmente zu überprüfen: Ist c_i links von beiden davon, so liegt c_i bereits im Inneren von H_{i-1}, dann haben wir einfach $H_i = H_{i-1}$. Andernfalls entfernen wir so lange Punkte am Ende von H_{i-1}, bis c_i links des Liniensegmentes vom zweitletzten zum letzten Punkt der erhaltenen Folge H'_{i-1} ist. Wir können dann c_i am Ende von H'_{i-1} einfügen. Die so erhaltene Folge ist H_i. Alle Punkte im Polygon $H_{i-1}, c_i, c_{i+1}, \ldots, c_{n-1}$ liegen auch im Polygon $H_i, c_{i+1}, \ldots, c_{n-1}$, da sich der Streckenzug $c_n, c_1, c_2, \ldots, c_i$ nicht selbst schneidet. Es kann allerdings Punkte in $c_2, c_3, \ldots, c_{i-1}$ geben, die in H_{i-1} sind aber nicht in H_i, jedoch werden diese von $H_i, c_{i+1}, \ldots, c_{n-1}$ noch einmal umschlossen. Daher bleibt unsere Invariante erhalten.

Nach einigen Iterationen erhalten wir so H_{n-1}, ein konvexes Polygon das alle gegebenen Punkte enthält und dessen Ecken eine Teilfolge von $c_n, c_1, c_2, \ldots, c_{n-1}$ sind. Daher ist H_{n-1} die konvexe Hülle dieser Punkte.

Falls es nun aber doch Punkte gibt, die rechts des Liniensegments c_n, c_1 liegen, so werden diese von diesem Algorithmus einfach verworfen. In diesem Fall berechnet er also nur die konvexe Hülle der Punktemenge geschnitten mit der Halbebene links des Liniensegments c_n, c_1. Wir können den Algorithmus also einfach zweimal anwenden, einmal auf die Folge $c_n, c_1, c_2, \ldots, c_{n-1}$ und einmal auf die Folge $c_1, c_n, c_{n-1}, \ldots, c_2$ und die beiden Resultate entlang c_1, c_n aneinanderfügen, indem wir jeweils den ersten Punkt der beiden konvexen Hüllen entfernen und die Resultate konkatenieren.

Mithilfe von Stacks lässt sich dieser Algorithmus in Zeit $O(n)$ implementieren, ähnlich wie wir das bei LocalRepair bereits gesehen haben.

(c) **Algorithmus:** Wir iterieren über die Punkte $p_1, p_2, \ldots, p_n$ und finden die Punkte p_ℓ und p_r mit der kleinsten und grössten x-Koordinate. Wir verschieben die Punkte $p_1, p_2, \ldots, p_n$ dann zirkulär, um die Folge $p_\ell, p_{\ell+1}, \ldots, p_r, p_{r+1} \ldots, p_{\ell-1}$ zu erhalten. Wir lassen den entsprechenden Teil des Algorithmus von (b) mit Eingabe $p_\ell, p_{\ell+1}, \ldots, p_r$ laufen, um die untere konvexe Hülle von P zu berechnen, sowie mit $p_r, p_{r+1} \ldots, p_\ell$ um die obere konvexe Hülle von P zu berechnen. Wir fügen die beiden Teile dann zusammen. (Gibt es mehrere Punkte mit minimalen bzw. maximalen x-Koordinaten, so können wir für die untere konvexe Hülle die x-extremalen Punkte mit minimalen y-Koordinaten wählen und für die obere konvexe Hülle die mit maximalen y-Koordinaten.)

Laufzeit: Die Berechnung der Punkte p_ℓ und p_r benötigt Zeit $O(n)$. Wir lassen dann den Algorithmus von (b) zweimal laufen, was Zeit $O(n)$ braucht. Insgesamt ist die Laufzeit also $O(n)$.

Korrektheit: Wir zerteilen das Polygon in zwei Streckenzüge und berechnen die untere und obere konvexe Hülle separat. Die Korrektheit unseres Algorithmus folgt aus der Korrektheit des Algorithmus aus (b), sowie daraus, dass sich P nicht selbst überschneidet und daher der untere Streckenzug sich nie komplett über dem oberen Streckenzug befinden kann.