AuW-u05-bf

Aufgabe 1 – *Bewachen eines nicht-konvexen Reiches*

Alice ist die Königin eines polygonalen Reiches, das heisst, die Grenzen ihres Reiches bilden ein Polygon P (nicht unbedingt konvex), mit n Knoten. Sie will ihr Reich mit neu gebauten Wachtürmen beschützen. Es erscheint ihr ausreichend, Wachtürme bei allen Knoten von P zu bauen, welche auf dem Rand der konvexen Hülle von P liegen. Daher will sie die konvexe Hülle ihres Reiches berechnen, und sie würde das gerne in Zeit $O(n)$ erreichen. Für diese Aufgabe nehmen wir an, dass die Punkte in $\mathbb{R}^2$ liegen und sich in allgemeiner Lage befinden, das heisst, dass keine drei Punkte auf einer gemeinsamen Linie liegen. Für einen Punkt p schreiben wir $p(x)$ für die x-Koordinate von p und $p(y)$ für die y-Koordinate von p.

(a)

Sei $S = \{s_1, s_2, \ldots, s_n\}$ eine Menge von Punkten, die eine x-monotone Kurve bilden, das heisst, $s_i(x) < s_{i+1}(x)$, für $1 \leq i < n$. Berechnen Sie die konvexe Hülle von S in Zeit $O(n)$.

We use the LocalRepair algorithm with input S which, per definition, is sorted by x coordinates.

LocalRepair$(p_1, p_2, \ldots, p_n)$

$\triangleright$ setzt $(p_1, p_2, \ldots, p_n)$, $n \geq 3$, nach x-Koordinate sortiert, voraus

$q_0 \leftarrow p_1$
$h \leftarrow 0$
for $i \leftarrow 2$ **to** n **do** $\triangleright$ untere konvexe Hülle, links nach rechts
    while $h > 0$ und q_h links von $q_{h-1}p_i$ **do**
        $h \leftarrow h - 1$
    $h \leftarrow h + 1$
    $q_h \leftarrow p_i$ $\triangleright$ $(q_0, \ldots, q_h)$ untere konvexe Hülle von $\{p_1, \ldots, p_i\}$
$h' \leftarrow h$
for $i \leftarrow n - 1$ **downto** 1 **do** $\triangleright$ obere konvexe Hülle, rechts nach links
    while $h > h'$ und q_h links von $q_{h-1}p_i$ **do**
        $h \leftarrow h - 1$
    $h \leftarrow h + 1$
    $q_h \leftarrow p_i$
return $(q_0, q_1, \ldots, q_{h-1})$ $\triangleright$ Ecken der konvexen Hülle, gg. Uhrzeigersinn

---

Satz 3.42. Gegeben eine Folge $p_1, p_2, \ldots, p_n$ nach x-Koordinate sortierter Punkte in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus LocalRepair die konvexe Hülle von $\{p_1, p_2, \ldots, p_n\}$ in Zeit $O(n)$.

Thus we have computed $conv(S)$ in $O(n)$.

$\square$

(b)

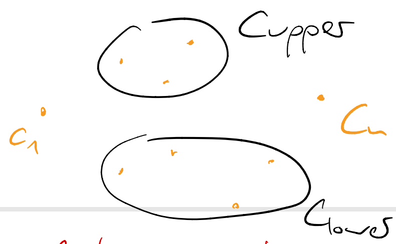
Sei $C = \{c_1, c_2, \ldots, c_n\}$ eine Menge von Punkten, die eine nicht-überschneidende Kurve bilden, das heisst, das Segment (c_i, c_{i+1}) schneidet maximal zwei weitere Segmente: Segment (c_{i-1}, c_i) bei c_i (falls $i \geq 2$), und Segment (c_{i+1}, c_{i+2}) bei c_{i+1} (falls $i \leq n-2$). Wir nehmen ausserdem an, dass $c_1(x) < c_i(x)$ für $1 < i \leq n$ und $c_i(x) < c_n(x)$ für $1 \leq i < n$. Berechnen Sie die konvexe Hülle von C in Zeit $O(n)$.

$$p_1 \prec_q p_2 \quad :\Leftrightarrow p_1 \text{ rechts von } qp_2$$

We split the set C into two subsets, $C_{upper} = \{c_u \in C \mid c_u \prec_{c_n} c_1\}$ and $C_{lower} = \{c_l \in C \mid c_l \prec_{c_1} c_n\}$. Since the set is non crossing, we now have two basically sorted subsets. We run LocalRepair on them and merge the two resulting convex hulls.

Thus, we have computed $conv(C)$ in $O(n)$.

$\square$



⇒ Splitting the set into an upper & lower set is a solid approach (also part of the Musterlösung) but this doesn't sort the subsets. This is why you can't simply run LocalRepair.

(c)

Sei P ein (möglicherweise) nicht-konvexes einfaches[1] Polygon. Wir nehmen an, dass die Punkte $p_1, p_2, \ldots, p_n$ rund um den Rand des Polygons P der Reihe nach entgegen dem Uhrzeigersinn gegeben sind. Berechnen Sie die konvexe Hülle von P in Zeit $O(n)$.

*Since the polygon isn't centered around the origin, we don't get an order based on the angle

Since the points p in P are sorted anticlockwise and P is a simple polygon (no crossing edges), we can turn every point's x, y coordinates into their polar system equivalent (r, θ). This takes $O(n)$ in total. We definitely can, but I don't quite get why we should*

We then start building a stack, where the top element is always the last point considered a potential candidates for the convex hull's edge points.

```
function ccw(p, q, r)
        return (q.x - p.x)(r.y - p.y) - (q.y - p.y)(r.x - p.x)
end

P' = {}
for p in P do
        p' = polar(p)
        P' += p'
end

Q = {}
for p' in P' do
        while Q.size() > 1 and ccw(Q.peaktwice(), Q.peak(), p') <= 0
                Q.pop
        end
        Q.push(p')
end

return Q
```

what do these functions do?

??

While it may seem that the time complexity of the loop is $O(n^2)$, because for each point it goes back to check if any of the previous points make a "right turn", it is actually $O(n)$, because each point is considered at most twice in some sense. Each point can appear only once as a point (x_2, y_2) in a "left turn" (because the algorithm advances to the next point (x_3, y_3) after that), and as a point (x_2, y_2) in a "right turn" (because the point (x_2, y_2) is removed).

Thus we have computed $conv(P)$ in $O(n)$.

□
