Algorithmen zur Berechnung der konvexen Hülle

Wolfgang Mulzer

1 Graham Scan

Die Eingabe besteht aus einer Menge P von n Punkten in der Ebene. Wir nehmen an, dass alle x-Koordinaten in P verschieden sind und dass keine drei Punkte in P auf einer Geraden liegen.

Wir berechnen nur die obere konvexe Hülle $\mathrm{UH}(P)$ von P. Der Algorithmus für die untere Hülle ist analog (ersetze `isLeftOf` durch `isRightOf`)

```
sort P by x-coordinate
Let p[1], p[2], ..., p[n] be the sorted points
S <- new Stack
S.push(p[1])
S.push(p[2])
for i := 3 to n do
    while S.size >= 2 and p[i] is left of the line through
          the topmost two points of S do
        S.pop()
    S.push(p[i])
```

Danach enthält S die obere konvexe Hülle von P, sortiert von rechts nach links. Der Sortiervorgang benötigt $O(n \log n)$ Zeit. Die übrigen Schritte laufen in Zeit $O(n)$, da für jeden Punkt eine `push`-Operation und höchstens eine `pop`-Operation durchgeführt wird und da der `isLeftOf`-Test konstante Zeit braucht.

2 Jarvis March

Wieder besteht die Eingabe aus einer Menge P von n Punkten in der Ebene, so dass alle x-Koordinaten in P verschieden sind und so dass keine drei Punkte in P auf einer Geraden liegen.

```
p, p0 <- leftmost point in P
do
    q <- arbitrary point in P \ p
    for all points r in P \ {p, q} do
      if r is to the left of the directed line through p and q then
          q <- r
      add the line segment (p,q) to the convex hull
    p <- q
while p != p0
```

Jeder Durchlauf der `do...while` Schleife benötigt $O(n)$ Zeit. In jedem Durchlauf wird eine neue Kante auf der konvexen Hülle gefunden. Also ist die Laufzeit O(nh), wobei h die Anzahl der Punkte (also auch die Anzahl der Kanten) auf der konvexen Hülle bezeichnet. Somit ist Jarivs March ein ausgabesensitiver Algorithmus.

3 Chans Algorithmus

Gegeben sei wieder eine Menge P von n Punkten in der Ebene, so dass alle x-Koordinaten in P verschieden sind und so dass keine drei Punkte in P auf einer Geraden liegen. Außerdem nehmen wir an, dass die Anzahl h der Punkte auf CH(P) bekannt ist. Der Algorithmus besteht aus zwei Phasen. In der ersten Phase wird P beliebig in Gruppen der Größe h aufgeteilt, und für jede Gruppe wird die konvexe Hülle berechnet.

```
 1. subdivide P into n/h groups of size h
 2. for each group P[i], find CH(P[i]) in time
    O(h log h) using Graham's scan
```

Die Gesamtlaufzeit für die erste Phase ist $O(n/h \cdot h \log h) = O(n \log h)$. In der zweiten Phase werden die einzelnen Gruppen kombiniert. Dies geschieht analog zu Jarvis's march.

```
 3. For each group P[i], let s[i] be the leftmost point.
 4. p0, p <- leftmost point of P
 5. i <- index of the group containing p
 6. do
 7.     s[i], q <- the successor of p on CH(P[i])
 8.     for j := 1 to n/h except for i do
 9.         while the successor of s[j] on CH(P[j]) is left of ps[j] do
10.            s[j] <- successor of s[j]
11.        if s[j] is to the left of pq then
12.            q <- s[j]
13.    add the line segment (p, q) to the convex hull
14.    p <- q
15.    i <- index of the group containing p
16. while p != p0
```

Jeder Durchlauf der äußeren `do...while`-Schleife hat $O(n/h + \#$ Durchläufe der inneren `while`-Schleife) Schritte. Man kann sich überlegen, dass jeder Punkt nur einmal als Nachfolger in der inneren `while`-Schleife vorkommt (d.h., jeder Punkt `s[j]` wandert nur einmal um die jeweilige Hülle herum). Wie bei Jarvis's march gibt es nur h Durchläufe der äußeren `do...while` Schleife, da in jedem Durchlauf ein neuer Punkt auf CH(P) entdeckt wird. Folglich ist die Gesamtzeit $O(n)$. Insgesamt hat Chans Algorithmus also Laufzeit $O(n \log h)$. Um h zu finden, führt man eine superexponentielle Suche durch. Dies wird in einer Übung behandelt.