3. Aufgabe aus Computational Geometry

Jakob Schöttl, Markus Stampfl

Implementierung

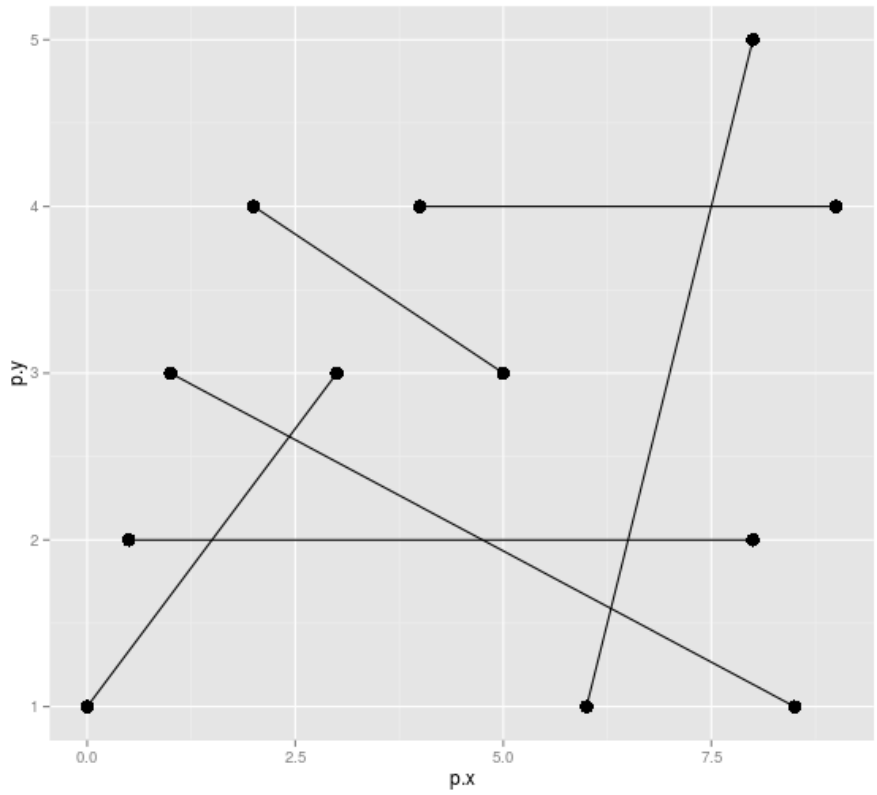
Das Programm liegt im Ordner `haskell/`. Der Algorithmus ist im Modul `CG.CG3` implementiert. Der Programmaufruf ist wie bei der 1. Aufgabe:

```
haskell/intersect < data/test.dat
```

Die Ausgabe ist die Anzahl der Schnittpunkte in der Eingabedatei.

Ergebnisse

Zum Test wurde die Datei `data/test.dat` verwendet. Diesen Testfall haben wir auf dem Papier konstruiert. Hier liefert unser Programm das richtige Ergebnis, nämlich 6.



Unsere Testdatei mit sechs Schnittpunkten.

Ergebnisse von unserem Line Sweep (losgelassen auf die Originaldaten):

```
s_100000_1.dat -> 28349
s_10000_1.dat  -> 649
s_1000_1.dat   -> 4
s_1000_10.dat  -> 774
```

Ergebnisse von 1. Aufgabe:

```
s_100000_1.dat -> 77126
s_10000_1.dat  -> 732
s_1000_1.dat   -> 11
s_1000_10.dat  -> 796
```

Was schonmal gut ist: Line Sweep findet wie erwartet in keiner Testdatei mehr Schnittpunkte als der einfache Algorithmus aus der 1. Aufgabe.

Dass der Line Sweep weniger Schnittpunkte findet ist auch plausibel, da er – in einfacher Ausführung – ja bekanntlich folgende Voraussetzungen hat:

1. x-Koordinaten der Schnitt- und Endpunkte sind paarweise
2. verschieden
3. Länge der Segmente > 0
4. nur echte Schnittpunkte
5. keine Linien parallel zur y-Achse
6. keine Mehrfachschnittpunkte
7. keine überlappenden Segmente

(vgl. Folien 4-3)

Die Einschränkung 1. verstehen wir so, dass auch die x-Koordinaten der Start- und Endpunkte in den Eingabedaten paarweise verschieden sind. Diese Einschränkung ist notwendig, da unsere Event Queue-Datenstruktur eine Tree Map ist, die x-Werte auf Ereignisse abbildet. Somit darf es keine zwei gleichen x-Werte geben. Falls doch, verdrängt ein neues Ereignis ein altes Ereignis, das den selben x-Wert hat.

Bei keinem der Datensätze ist die 1. Voraussetzung erfüllt. Schon beim Einlesen der Daten zählt also jeweils nur das letzte Ereignis an einer Stelle x.

Hiermit überprüfen wir, ob die Daten die 1. Vorraussetzung erfüllen:

```
cat data/xxx.dat | awk '{print $1 RS $3}' | wc -l
cat data/xxx.dat | awk '{print $1 RS $3}' | sort -u | wc -l
```

Bei der neuen Datei `s_1000_10.dat` unterscheiden sich die Zahlen nur durch 6, also zum Beispiel 6 Paare mit gleichen x-Koordinaten. Bei den anderen Dateien ist der Unterschied noch größer.

Optimierung

Eine nette Optimierung wäre noch, dass bei Änderungen in der y-Struktur, nur in den geänderten Nachbarschaften auf neue Schnittpunkte getestet würde. Dazu würde `insertIntersections` nur noch mit einer Teilmenge der `yo`-Liste (y-Struktur) aufgerufen werden. `insertLineInY`, `removeLineFromY` und `swapLinesInY` müssten dazu aber nicht nur die neue y-Struktur zurückgeben, sondern eben auch die Teilmenge, in der sich Nachbarschaften geändert haben. Ganz trivial ist das aber auch nicht, weil es Spezialfälle gibt, z. B. eine Linie am Anfang oder am Ende der `yo`-Liste entfernen oder hinzufügen ...