Praktische Informatik 3 WS 12/13

Christoph Lüth
Marcus Ermler
Christian Maeder
Martin Ring
Diedrich Wolter

3. Übungsblatt

Ausgabe: 07.11.12
Abgabe: 16.11.12

3.1 *Ghostbusters!* *20 Punkte*

Wenn die Geisterjäger Geister jagen, dann benutzen sie dabei *proton packs*, die einen Strahl von Hochenergiepartikeln erzeugen, und damit außerweltliche Materialisationen (Geister) einfangen können. Dabei ist darauf zu achten, dass die Strahlen sich nicht kreuzen dürfen, weil dieses „katastrophale Konsequenzen“ haben kann.

Wenn also n Geisterjäger auf n Geister treffen, dann müssen die Geisterjäger eine Strategie entwickeln, wie sie die Geister einfangen, ohne in Gefahr zu laufen, dass sich die Strahlen kreuzen. Dabei wollen wir ihnen helfen.

Gegeben seien also zwei gleich mächtige, disjunkte Mengen $\mathscr{G}$ von Positionen von Geistern und $\mathscr{B}$ von Positionen von Geisterjägern. Die Strategie besteht darin, hieraus eine Menge von Paaren (B,G) mit $B \in \mathscr{B}, G \in \mathscr{G}$ zu bilden, so dass jedem Geisterjäger B genau ein Geist G zugeordent wird, und dass für keine zwei Paare (B,G) und (C,H) die Strahlen nicht kreuzen (also die Strecken $\overrightarrow{B\,G}$ und $\overrightarrow{C\,H}$ sich schneiden).

Der Algorithmus, der diese Zuordnung berechnet, ist rekursiv. Der Rekursionsschritt besteht darin, ein Paar (B,G) zu finden, so dass sich auf beiden Seiten der dadurch beschriebenen Geraden $\overline{B\,G}$ jeweils gleich viele Geister und Geisterjäger befinden, und diese dann rekursiv zuzuordnen. Das Paar (B,G) wird wie folgt gefunden: sei $\mathscr{P} = \mathscr{G} \cup \mathscr{B}$, dann nehmen wir den Punkt $P \in \mathscr{P}$, der sich am weitesten links unten befindet, und ordnen $\mathscr{P} \setminus \{P\}$ in ansteigendem Winkel zu P an. Nehmen wir (oBdA) an, P sei die Position eines Geisterjägers, dann nehmen wir aus der sortierten Menge so viele Punkte $\mathscr{Q} = \{Q_1, \ldots, Q_n\}$, bis die Anzahl Geister die der Geisterjäger um eins überschreitet. Jetzt können wir auf die Mengen $\{Q_1, \ldots Q_{n-1}\}$ und $\mathscr{P} \setminus (\{P\} \cup \mathscr{Q})$ den Algorithmus rekursiv anwenden, und die Ergebnisse zusammen mit (P,Q_n) zum Gesamtergebnis vereinigen.

Wir könnnen davon ausgehen, dass in der Eingabemenge gleich viele Geister und Geisterjäger sind, und dass keine drei Punkte kolinear (auf einer Geraden) sind.

Zur Implementierung der Lösung in Haskell gehen wir wie folgt vor:

1. Zuerst definieren wir einen algebraischen Datentypen Obj, der einen Geist oder einen Geisterjäger repräsentiert, und dessen Koordinaten (die Positon) enthält. Die oben beschriebenen Mengen $\mathscr{B}$ und $\mathscr{G}$ werden dann durch Listen von Obj repräsentiert.

2. Um zu berechnen, ob ein Punkt rechts oder links einer Geraden ist, nutzen wir die Determinantenformel, die für drei Punkte P,Q,R definiert ist als

$$\begin{aligned} D_{P,Q,R} &= (Q_y - P_y)(R_x - Q_x) - (R_y - Q_y)(Q_x - P_x) \\ &= Q_y R_x - P_y R_x + P_y Q_x - R_y Q_x + R_y P_x - Q_y P_x \end{aligned}$$

   Falls $D_{P,Q,R} < 0$, dann ist R links des Strahls $\overline{P\,Q}$, und fall $D_{P,Q,R} > 0$, dann ist R rechts des Strahls.

3. Damit können wir jetzt eine Funktion definieren, die eine Menge von Objekten $Q_1, \ldots, Q_n$ in ansteigendem Winkel bezüglich eines Objektes P sortiert: wir definieren, dass $Q_i \leq Q_j \Longleftrightarrow D_{P,Q_i,Q_j} > 0$, und sortieren bezüglich dieser Ordnung in aufsteigender Reihenfolge:

   ```
   sortByAngle :: Obj→ [Obj]→ [Obj]
   ```

4. Ferner benötigen wir eine Funktion

   ```
   leftmostBottom :: [Obj]→ (Obj, [Obj])
   ```

   die das Objekt am weitesten links unten aus der Liste sowie den Rest der Liste zurückliefert.

5. Die Hauptfunktion ist

   ```
   pairs :: [Obj]→ [(Obj, Obj)]
   ```

   Sie berechnet mittels leftmostBottom das Objekt links unten, sortiert den Rest der Objekte mit sortByAngle, teilt die Liste in zwei Teillisten, so dass in beiden Teillisten die gleiche Anzahl Geister und Geisterjäger zu finden sind, und ruft sich rekursiv für diese Teillisten auf.

Um sicherzugehen, dass keine katastrophalen Konsequenzen durch Programmierfehler auftreten, und sich doch Strahlen kreuzen, wollen wir das Ergebnis prüfen können. Ingesamt ist eine korrekte Lösung eine Liste von Paaren, so dass

- die Elemente der Paare und der Eingangsliste eine Permutation bilden, und
- sich keine zwei Strecken kreuzen.

Implementieren Sie folgende Funktionen

```
isPerm :: [Obj]→ [(Obj, Obj)]→ Bool
intersect :: (Obj, Obj)→ (Obj, Obj)→ Bool
noIntersect :: [(Obj, Obj)]→ Bool
```

die prüfen, ob diese zwei Bedingungen zutreffen; intersect ist hierbei eine Hilfsfunktion, die prüft, ob zwei Strecken sich schneiden, und benutzen Sie diese, um eine Testfunktion zu implementieren.

? *Verständnisfragen*

1. Was ist Polymorphie?
2. Welche zwei Arten der Polymorphie haben wir kennengelernt, und wie unterschieden sie sich?
3. Was ist der Unterschied zwischen Polymorphie in Haskell, und Polymorphie in Java?