Aufgabe 8

Gegeben seien zwei nichtleere Mengen R und B von roten bzw. blauen Punkten auf der x-Achse. Gesucht ist der minimale euklidische Abstand $d(r, b)$ über alle Punktepaare (r, b) mit $r \in R$ und $b \in B$. Hier ist eine Beispielinstanz:

Die Eingabe wird in einem Feld A übergeben. Jeder Punkt $A[i]$ mit $1 \leq i \leq n$ hat eine x-Koordinate $A[i].x$ und eine Farbe $A[i].color \in \{\text{rot}, \text{blau}\}$. Das Feld A ist nach x-Koordinate sortiert, d. h. es gilt $A[1].x < A[2].x < \cdots < A[n].x$, wobei $n = |R| + |B|$.

(a) Geben Sie in Worten einen Algorithmus an, der den gesuchten Abstand in $\mathcal{O}(n)$ Zeit berechnet.

Pseudo-Code

- Iteriere über die Indizes des Punkte-Arrays P bis zum vorletzten Index ($P[n-1]$)
- Falls $P[n].color = P[n+1].color$, dann gehe zum nächsten Durchlauf mit $P[n+1]$
- Andernfalls berechne $d = P[n+1].x - P[n].x$
- Vergleiche d mit d_{min} (ist anfangs als $d_{min} = Integer.MAX\_VALUE$ definiert). Falls $d < d_{min}$ definiere $d_{min} = d$

Java

```
30  public double findMinimalDistance() {
31    double distanceMin = Double.MAX_VALUE;
32    for (int i = 0; i < latestIndex - 1; i++) {
33      if (points[i].color != points[i + 1].color) {
34        double distance = points[i + 1].x - points[i].x;
35        if (distance < distanceMin) {
36          distanceMin = distance;
37        }
38      }
39    }
40    return distanceMin;
41  }
```

github: raw

(b) Begründen Sie kurz die Laufzeit Ihres Algorithmus.

Da das Array der Länge n nur einmal durchlaufen wird, ist die Laufzeit $\mathcal{O}(n)$ sichergestellt.

(c) Begründen Sie die Korrektheit Ihres Algorithmus.

In d_{min} steht am Ende der gesuchte Wert (sofern nicht $d_{min} = Integer.MAX\_VALUE$ geblieben ist)

(d) Wir betrachten nun den Spezialfall, dass alle blauen Punkte links von allen roten Punkten liegen. Beschreiben Sie in Worten, wie man in dieser Situation den gesuchten Abstand in $o(n)$ Zeit berechnen kann. (Ihr Algorithmus darf also insbesondere nicht Laufzeit $\Theta(n)$ haben.)

> Zuerst muss man den letzten blauen Punkt finden. Das ist mit einer binären Suche möglich. (Man beginnt mit dem ganzen Array als Suchbereich und betrachtet den mittleren Punkt. Wenn er blau ist, wiederholt man die Suche in der zweiten Hälfte des Suchbereichs, sonst in der ersten, bis man einen blauen Punkt gefolgt von einem roten Punkt gefunden hat.) Der gesuchte minimale Abstand ist dann der Abstand zwischen dem gefundenen blauen und dem nachfolgenden roten Punkt. Die Binärsuche hat eine Worst-case-Laufzeit in $\mathcal{O}(\log n)$ und damit auch in $o(n)$.

```
3   enum Color {
4     RED, BLUE
5   }
6
7   class Point {
8     double x;
9     Color color;
10
11    public Point(double x, Color color) {
12      this.x = x;
13      this.color = color;
14    }
15  }
16
17  public class RedBluePairCollection {
18    Point[] points;
19    int latestIndex;
20
21    public RedBluePairCollection(int size) {
22      points = new Point[size];
23    }
24
25    public void addPoint(double x, Color color) {
26      points[latestIndex] = new Point(x, color);
27      latestIndex++;
28    }
29
30    public double findMinimalDistance() {
31      double distanceMin = Double.MAX_VALUE;
32      for (int i = 0; i < latestIndex - 1; i++) {
33        if (points[i].color != points[i + 1].color) {
34          double distance = points[i + 1].x - points[i].x;
35          if (distance < distanceMin) {
36            distanceMin = distance;
37          }
38        }
39      }
40      return distanceMin;
41    }
42
43    public static void main(String[] args) {
44      RedBluePairCollection pairs = new RedBluePairCollection(11);
45
46      pairs.addPoint(0, Color.RED);
```

```
47        pairs.addPoint(0.2, Color.RED);
48        pairs.addPoint(1.5, Color.BLUE);
49        pairs.addPoint(3.1, Color.RED);
50        pairs.addPoint(4.0, Color.BLUE);
51        pairs.addPoint(4.2, Color.BLUE);
52        pairs.addPoint(5.1, Color.RED);
53        pairs.addPoint(6, Color.BLUE);
54        pairs.addPoint(6.1, Color.BLUE);
55        pairs.addPoint(6.2, Color.BLUE);
56        pairs.addPoint(7.2, Color.RED);
57
58        System.out.println(pairs.findMinimalDistance());
59    }
60 }
```

github: raw