Aufgabe 1: Lisa Rennt

Teilnahme-Id: 48933

Bearbeiter/-in dieser Aufgabe:
Oliver Schirmer

26. April 2019

Inhaltsverzeichnis

Lösungsidee

Zu Beginn werden die zu lösenden Daten entsprechend den Formatdetails auf der Materialwebsite eingelesen. Dabei werden erst nur die Koordinaten der Polygone bzw. Lisas Haus gespeichert.

Zur tatsächlichen Lösung und Wegfindung wird grundlegend der Dijkstra-Algorithmus verwendet. Dazu müssen jedoch erst alle möglichen Knoten und Kanten des dafür nötigen Grafen initialisiert werden. Die Knoten werden von allen Polygon-Eckpunkte und Lisas Haus gebildet. Für die Kanten werden nur gerade Strecken verwendet, die entweder Seite eines Polygons oder direkte Verbindung zwischen 2 Polygon-Eckpunkten sind, also keine anderen Polygone schneiden. Der Dijkstra-Algorithmus, ein typischer GreedyAlgorithmus funktioniert nun auf folgende Art und Weise:
Es wird ein Startknoten eingegeben und die Distanz zu jedem Knoten initialisiert. Diese ist für den Startknoten selbst 0 und für alle anderen zu Beginn unendlich. Dann wird der Knoten mit der kleinsten Distanz gesucht, was zu Beginn der Startknoten ist. Dessen benachbarten Knoten werden besucht und die neue Distanz eingetragen (das Kantengewicht bis zu diesem Knoten). Daraufhin wird wieder ein bisher unbesuchter Knoten mit der kleinsten Distanz zum Startknoten gesucht und von dort aus alle benachbarten Knoten geupdated, d.h. falls die Distanz über den aktuellen Knoten kleiner ist als die bisherige wird die neue Distanz eingetragen.
Damit sich der kürzeste Pfad zu einem Knoten im Nachhinein wieder rekonstruieren lässt wird zusätzlich bei jedem Knoten der Vorgängerknoten gespeichert.

Nachdem zu jedem Knoten der kürzeste Pfad gefunden wurde werden die Zielknoten gesammelt. Das sind alle die eine direkte Verbindung zur Straße über optimales Querfeldeinlaufen haben. Optimal lässt sich mathematisch mit einer linearen Funktion errechnen, also wird die Standardgleichung $y = m * x + n$ angesetzt.
Grundsätzlich ist das Ziel, dass die Laufzeit für Lisa und die Fahrzeit des Busses gleich lang sind. Da beides gleichförmige Bewegungen sind, lässt sich die Formel $v = \frac{s}{t}$ ansetzen und zu t umstellen, woraus sich $t = \frac{s}{v}$ ergibt. Da der Weg den der Bus zurücklegt gleich dem y-Achsenabschnitt, also n, ist, lässt sich die Zeit dafür folgendermaßen berechnen: $t_1 = \frac{n}{30 \frac{km}{h}}$

Um Lisas Weg zu berechnen wird die x- und y-Koordinate des Punktes benötigt, mit denen man durch den Satz des Pythagoras die Zeit folgendermaßen ausrechnen kann: $t_2 = \frac{\sqrt{x^2 + (n-y)^2}}{15\,\frac{km}{h}}$

Beim Gleichsetzen von t_1 und t_2 lassen sich die beiden Geschwindigkeiten kürzen, wodurch nur noch eine 2 bei t_1 unter dem Bruchstrich bleibt. Stellt man die Formel jetzt zu n um erhält man aufgrund der Wurzel 2 Gleichungen. Die 2. ist jedoch negiert, berechnet also den niedrigeren n-Wert, der in jedem Fall niedriger als der y-Wert des Punktes gelegen ist. Da wir jedoch den am weitesten oben liegenden n-Wert erreichen wollen, ist nur folgende Formel relevant: $n = \frac{2(\sqrt{y^2 - 3x^2} + 2y)}{3}$

Es wird nun über alle Knoten iteriert und der entsprechende n-Wert berechnet. Ist dieser niedriger als der y-Wert des Punktes, was aufgrund der Geschwindigkeits- und Wegverhältnisse passieren kann, wird stattdessen der y-Wert des Punktes genommen. Zwischen Knoten und entsprechendem Punkt auf der Straße wird eine Linie gezogen und geprüft, ob sich diese Linie mit einer Kante eines Polygons schneidet. Ist dies nicht der Fall wird der Knoten als möglicher Zielknoten gespeichert.

Im letzten Abschnitt wird über alle möglichen Zielknoten iteriert und der Gesamtweg berechnet, d.h. Weg zum Knoten von Lisas Haus plus Weg zur Straße per Querfeldeinlaufen. Zudem wird für diesen speziellen Weg die Startzeit für Lisa berechnet, da diese das Ziel ist und nicht nur abhängig von der Länge des Weges, sondern auch des Weges, den der Bus zurücklegen muss, und somit längere Wege eine spätere Startzeit haben können. Es wird der Zielknoten mit der spätesten Startzeit ausgewählt und als Ergebnis festgelegt.
Daraufhin werden Start-, Lauf-, Ankunftszeit und Pfad ausgegeben und eine grafische Darstellung des Pfads erstellt.

Umsetzung

Für das Einlesen der Daten wird ein BufferedReader genutzt, welcher über jede Zeile iteriert und prüft um welche Zeile es sich handelt, um dann entsprechend die Informationen zu speichern. Zur Speicherung der Polygone wird ein Adapter der AWT Polygon-Klasse genutzt, der die Eckpunkte zusätzlich als PointNodes und die Kanten als Line2D abspeichert. Hauptvorteil dieser beiden Klassen ist, dass sie Koordinaten auch als Double verarbeiten können und somit genauer sind. Wichtig ist dies jedoch nur beim auf der Straße liegenden Punkt, da dieser berechnet wird und somit meist keine Ganzzahlen, wie die Polygone, aufweist. Außerdem können durch den Adapter und die Verwendung von PointNode-Objekten die Namen der Polygone und Namen der Eckpunkte gespeichert werden, um diese später genau identifizieren zu können.

Für den Dijkstra-Algorithmus wird die PointNode-Klasse als Repräsentant für die Knoten und die Edge-Klasse als Repräsentant für die Kanten genutzt. Letztere speichert nur Start- und Endknoten und die Länge der Strecke, welche auch mit dem Satz des Pythagoras berechnet wird. In der PointNode-Klasse wird wie bereits gesagt der vorherige Knoten des kürzesten Pfads, die Distanz und ob der Knoten schon besucht wurde gespeichert, aber auch alle Kanten zu benachbarten Knoten-
Zu Beginn des Algorithmus wird eine Liste mit Knoten und Kanten an die Graph-Klasse übergeben. Es wird über alle Kanten iteriert und dem Start- und Endknoten der aktuellen Kante diese hinzugefügt. Damit ist es später möglich alle Nachbarn von Knoten zu finden, indem einfach über die Liste der Kanten eines Knotens iteriert wird. Während der Ausführung des Algorithmus sind keine Abweichungen vom Standardalgorithmus zu finden, da für die Ermittlung des optimalen Wegs mehrere möglichen Zielknoten untersucht werden müssen und somit nicht nach dem Erreichen eines bestimmten Knotens gestoppt werden kann.

Auch die Ermittlung der Zielknoten hat keine großen programmtechnischen Abweichungen von dem in der Lösungsidee erläuterten Ansatz. Es wird über alle Knoten iteriert, der optimale Straßenpunkt bezüglich des

Querfeldeinlaufens ermittelt und geprüft, ob sich die Linie zwischen beiden Punkten mit einem Polygon schneidet, indem über alle Polygone und nochmals über alle Kanten der Polygone iteriert wird und geprüft wird, ob sich die Kante des Polygons mit der Linie schneidet. Ist dies nicht der Fall wird der Knoten als möglicher Zielknoten gespeichert. Es wird für jeden Zielknoten im gleichen Zusammenhang die nötige Startzeit berechnet und geprüft ob diese später als die des aktuell besten Zielknotens ist. Ist das der Fall, wird der aktuelle Zielknoten als bester Zielknoten eingespeichert.

Zuletzt werden die Weglänge, an welcher Stelle Lisa auf den Bus auf der y-Achse trifft, auf Meter genau und die Start-, Lauf- und Ankunftszeit auf Sekunden genau ausgegeben. Der Pfad wird ausgegeben, indem von jedem Knoten, beginnend beim Zielknoten, rekursiv der Vorgänger zu einer Liste hinzugefügt wird. Diese wird zuletzt invertiert, damit der Zielknoten den letzten Punkt bildet (auf den nur noch der Straßenpunkt folgt).

Beispiel

Eigenes Beispiel 1:

An diesem Beispiel lässt sich erkennen, dass tatsächlich quer gelaufen wird, wenn der x-Wert des Punkts kleiner oder gleich der Hälfte des y-Werts ist, was das Geschwindigkeitsverhältnis von 2:1 für den Bus repräsentiert.

```
---INFOS---
L: 50.0|100.0
---ALGORITHMUS---
L 0.0 null
---ERGEBNIS---
Lisa's Weg: 83.33m
Bei y: 166.66m
Startzeit: 07:29:59
Laufzeit: 00:00:20
Ankunft: 07:30:19
Pfad: L S
```

Eigenes Beispiel 2:

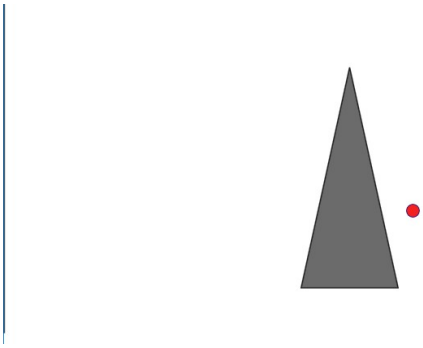
Dieses Beispiel ist das direkte Gegenbeispiel zum vorherigen Beispiel, dass bei einem höheren Verhältnis des x- zum y-Wert von Lisas Haus (höher als 2:1) nicht quer gelaufen wird, also der Punkt auf der Straße immer mindestens genauso hoch wie Lisas Haus liegt.

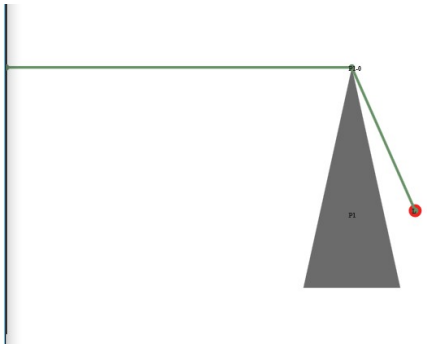
```
---INFOS---
L: 100.0|50.0
---ALGORITHMUS---
L 0.0 null
---ERGEBNIS---
Lisa's Weg: 100.0m
Bei y: 50.0m
Startzeit: 07:29:42
Laufzeit: 00:00:24
Ankunft: 07:30:06
Pfad: L S
```

Beispiel 1:

Im folgenden werden die Ergebnisse der Beispiele der Materialwebsite dargestellt. Die Infos werden ab dem 2. Beispiel weggelassen, da der Nachweis, dass das Einlesen funktioniert bereits in diesem Beispiel erbracht wurde. Trotz dessen wird die Arbeitsweise des Algorithmus dargestellt

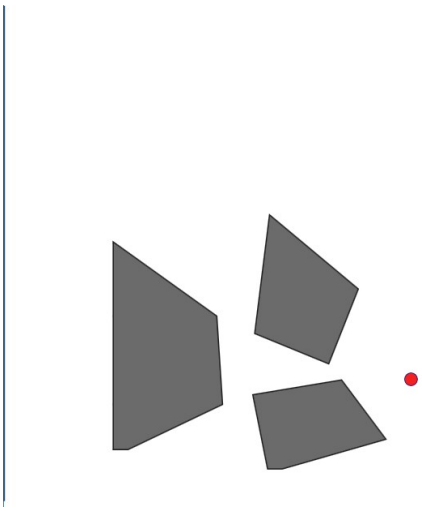
---ALGORITHMUS---
L 0.0 null
P1-0 241.75 L
P1-1 121.2 L
P1-2 271.2 P1-1
---ERGEBNIS---
Lisa's Weg: 776.75m
Bei y: 410.0m
Startzeit: 07:27:42
Laufzeit: 00:03:07
Ankunft: 07:30:49
Pfad: L P1-0 S

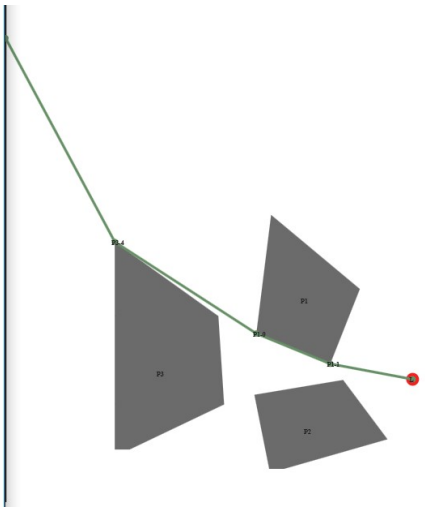




Beispiel 2:

---ALGORITHMUS---
L 0.0 null
P1-0 254.46 P1-1
P1-1 130.23 L
P1-2 162.24 L
P1-3 341.88 P1-2
P2-0 291.28 P2-1
P2-1 268.28 P2-2
P2-2 100.84 L
P2-3 108.0 L
P2-4 247.9 P2-3
P3-0 482.71 P3-1
P3-1 459.71 P2-4
P3-2 297.24 P2-4
P3-3 319.34 P1-0
P3-4 516.31 P1-0
---ERGEBNIS---
Lisa's Weg: 875.54m
Bei y: 718.45m
Startzeit: 07:27:55
Laufzeit: 00:03:31
Ankunft: 07:31:26
Pfad: L P1-1 P1-0 P3-4 S

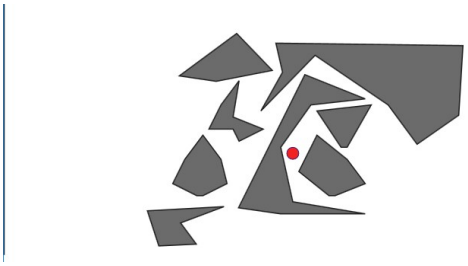


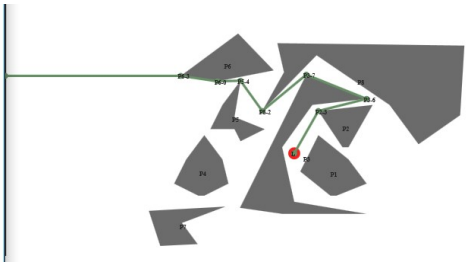


Beispiel 3:

---ALGORITHMUS---
L 0.0 null
P1-0 95.65 P1-5
P1-1 105.65 P1-0
P1-2 159.5 P1-1
P1-3 114.03 P1-4
P1-4 50.0 L
P1-5 31.62 L
P2-0 94.72 P1-4
P2-1 104.72 P2-0
P2-2 171.17 P2-3
P2-3 80.62 L
P3-0 373.44 P3-1
P3-1 302.73 P3-2
P3-2 162.73 P1-0
P3-3 80.0 L
P3-4 22.36 L
P3-5 85.44 L
P3-6 163.08 P2-3
P3-7 270.78 P3-6
P4-0 445.28 P3-0
P4-1 435.74 P3-0
P4-2 417.72 P3-0
P4-3 458.54 P3-0
P4-4 489.74 P5-1
P4-5 534.72 P4-6
P4-6 490.0 P4-0
P5-0 451.27 P5-1
P5-1 428.91 P5-2
P5-2 384.19 P3-7
P5-3 412.35 P8-2
P5-4 426.89 P8-2
P5-5 476.89 P5-4
P5-6 491.27 P5-0
P6-0 464.9 P5-4
P6-1 434.92 P8-2
P6-2 519.78 P6-1
P6-3 526.56 P6-0
P7-0 447.16 P3-2
P7-1 469.1 P3-2
P7-2 397.49 P3-1
P7-3 524.69 P7-2

P7-4 509.23 P7-0
P8-0 199.17 P2-2
P8-1 272.96 P3-6
P8-2 365.28 P3-7
P8-3 438.71 P8-2
P8-4 480.32 P6-1
P8-5 568.1 P8-6
P8-6 420.09 P8-7
P8-7 304.88 P8-8
P8-8 220.82 P2-1
---ERGEBNIS---
Lisa's Weg: 817.56m
Bei y: 296.0m
Startzeit: 07:27:18
Laufzeit: 00:03:17
Ankunft: 07:30:35
Pfad: L P2-3 P3-6 P3-7 P8-2 P5-4 P6-0 P6-3 S

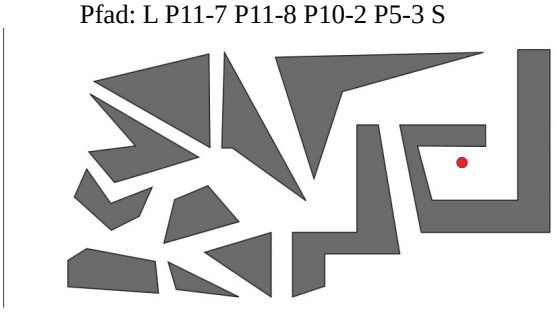


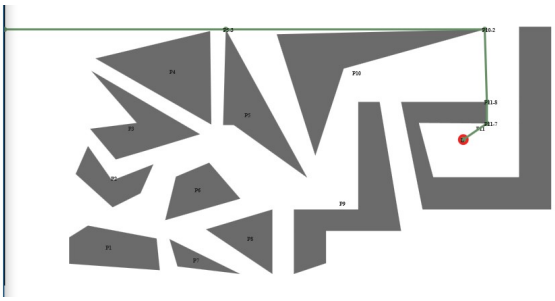


Beispiel 4:

---ALGORITHMUS---
L 0.0 null
P1-0 1014.5 P1-4
P1-1 845.15 P7-2
P1-2 806.14 P5-2
P1-3 924.16 P6-0
P1-4 965.5 P1-3
P2-0 973.47 P2-1
P2-1 880.71 P6-0
P2-2 828.55 P6-3
P2-3 792.44 P6-2
P2-4 875.08 P2-3
P2-5 920.2 P3-5
P3-0 937.02 P3-5
P3-1 1024.71 P3-0
P3-2 952.72 P5-2
P3-3 719.05 P5-2
P3-4 797.27 P5-2
P3-5 862.5 P6-2
P4-0 956.39 P4-1
P4-1 707.82 P5-2
P4-2 740.46 P5-3
P5-0 691.71 P5-1
P5-1 671.71 P5-2
P5-2 503.27 P9-5
P5-3 711.31 P10-2
P6-0 779.81 P6-1
P6-1 634.21 P5-2
P6-2 688.4 P5-2
P6-3 755.63 P6-2
P7-0 837.58 P7-2
P7-1 753.84 P8-0
P7-2 784.42 P5-2

P8-0 693.84 P9-7
P8-1 591.05 P5-2
P8-2 715.25 P5-2
P9-0 687.34 P9-7
P9-1 693.25 P9-2
P9-2 633.25 P9-3
P9-3 493.25 P11-9
P9-4 293.25 P11-9
P9-5 333.25 P9-4
P9-6 533.25 P9-5
P9-7 567.34 P5-2
P10-0 461.31 P9-5
P10-1 367.35 P11-8
P10-2 228.31 P11-8
P10-3 616.41 P10-2
P11-0 457.21 P11-9
P11-1 692.5 P11-2
P11-2 352.5 P11-3
P11-3 292.5 P10-2
P11-4 125.36 L
P11-5 89.64 L
P11-6 88.6 L
P11-7 53.25 L
P11-8 93.25 P11-7
P11-9 253.25 P11-8
---ERGEBNIS---
Lisa's Weg: 1124.31m
Bei y: 475.0m
Startzeit: 07:26:26
Laufzeit: 00:04:30
Ankunft: 07:30:56
Pfad: L P11-7 P11-8 P10-2 P5-3 S

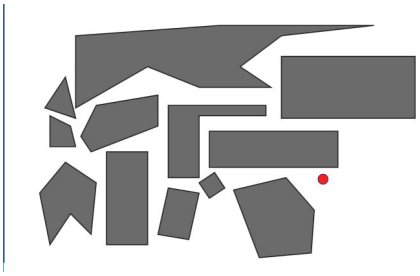


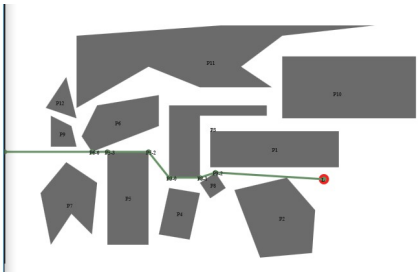


Beispiel 5:

---ALGORITHMUS---
L 0.0 null
P1-0 222.19 L
P1-1 37.01 L
P1-2 107.01 P1-1
P1-3 292.19 P1-0
P2-0 246.23 P2-1
P2-1 144.83 L
P2-2 62.36 L
P2-3 72.06 L
P2-4 176.84 P2-3
P3-0 303.02 P3-1
P3-1 243.02 P8-2
P3-2 324.17 P1-0
P3-3 250.19 P1-2
P3-4 258.86 P10-0
P3-5 348.86 P3-4

P3-6 443.02 P3-0
P4-0 367.23 P4-1
P4-1 306.4 P2-4
P4-2 248.85 P8-0
P4-3 306.26 P3-1
P5-0 467.02 P5-1
P5-1 387.02 P4-1
P5-2 367.05 P3-0
P5-3 447.05 P5-2
P6-0 477.05 P5-3
P6-1 405.0 P3-0
P6-2 464.26 P3-0
P6-3 580.19 P6-4
P6-4 513.1 P6-0
P7-0 577.02 P5-0
P7-1 559.65 P7-2
P7-2 503.08 P5-0
P7-3 510.29 P5-3
P7-4 529.51 P5-3
P7-5 607.61 P7-4
P8-0 226.49 P2-4
P8-1 192.73 P2-3
P8-2 211.4 L
P8-3 247.45 P8-2
P9-0 557.67 P6-0
P9-1 508.67 P6-0
P9-2 541.08 P6-0
P9-3 585.8 P9-2
P10-0 219.81 P1-2
P10-1 218.84 P1-2
P10-2 214.39 L
P10-3 334.39 P10-2
P10-4 339.81 P10-0
P11-0 393.49 P3-4
P11-1 392.53 P3-4
P11-2 283.06 P10-0
P11-3 355.17 P11-2
P11-4 379.81 P10-4
P11-5 434.39 P10-3
P11-6 734.39 P11-5
P11-7 709.01 P11-8
P11-8 569.01 P6-4
P11-9 501.2 P11-0
P12-0 602.11 P9-2
P12-1 601.3 P9-2
P12-2 549.51 P6-4
P12-3 631.95 P6-4
---ERGEBNIS---
Lisa's Weg: 647.05m
Bei y: 215.0m
Startzeit: 07:27:49
Laufzeit: 00:02:36
Ankunft: 07:30:25
Pfad: L P8-2 P3-1 P3-0 P5-2 P5-3 P6-0 S





Quellcode

```
public class LisaRennt {
    public static List<PolygonAdapter> polygons = new
ArrayList<>();
    public static PointNode lisaPoint;
    public static List<PointNode> resultPoints;
    public static List<Edge> edges;
    public static List<Edge> dijkstraWay = new ArrayList<>();

    public static void main(String[] args) {
        System.out.println("---INFOS---");
        // read information from file
        readFileData(coordsFileName);
        System.out.println(lisaPoint.toString());
        for (PolygonAdapter polygon : polygons) {
            System.out.println(polygon.toString());
        }
        // execute algorithm
        System.out.println("---ALGORITHMUS---");
        executeAlgorithm();
        SVGGenerator.generateSVG(coordsFileName.substring(0,
coordsFileName.length() - 4) + ".svg");
    }

    private static void executeAlgorithm() {
        // intialize all nodes
        List<PointNode> nodes = new ArrayList<>();
        nodes.add(lisaPoint);
        polygons.forEach(polygon ->
nodes.addAll(polygon.getCorners()));

        // initialize all edges
        edges = new ArrayList<>();
        Set<PolygonAdapter> unfinished = new HashSet<>(polygons);
        // iterate over all polygons
        for (PolygonAdapter polygon : polygons) {
            unfinished.remove(polygon);
            List<PointNode> corners = polygon.getCorners();
            // iterate over all corners of a polygon
            for (PointNode node : corners) {
                // add all connections to corners of other
polygons
                for (PolygonAdapter targetPolygon : unfinished) {
                    for (PointNode target :
targetPolygon.getCorners()) {
                        if (hasDirectConnection(node, target)) {
                            edges.add(new Edge(node, target));
                        }
                    }
                }
                // add connection to Lisas home, if one exists
                if (hasDirectConnection(node, lisaPoint)) {
                    edges.add(new Edge(node, lisaPoint));
                }
            }

            // add all connections between the polygon nodes
            int nPoints = polygon.npoints;
            for (int i = 0; i < nPoints - 1; i++) {
```

```
                edges.add(new Edge(corners.get(i), corners.get(i +
1)));
            }
            edges.add(new Edge(corners.get(nPoints - 1),
corners.get(0)));
        }

        Graph graph = new Graph(nodes, edges);
        graph.execute(lisaPoint);

        for (PointNode node : nodes) {
            PointNode predecessor = node.getPredecessor();
            String name = predecessor == null ? null :
predecessor.getName();
            System.out.println(
                    node.getName() + " " +
Utils.round(node.getDistanceFromSource(), 2) + " " + name);
        }

        // get point with street connection (optimal not with an
angle of 0°) and latest starting time
        PointNode streetConnectionPoint = null;
        PointNode streetPoint = null;
        double bestStreetPointWay = 0;
        double bestStartTime = 0;
        for (PointNode node : nodes) {
            PointNode potentialStreetPoint = new PointNode("S", 0,
getStreetY(node));
            if (hasDirectConnection(node, potentialStreetPoint)) {
                // check if point is reachable by Lisa (Graph from
Dijkstra)
                if (!node.getName().equals("L") &&
node.getPredecessor() == null) {
                    continue;
                }
                double streePointWay =
                        node.getDistanceFromSource() +
node.distance(potentialStreetPoint.x, potentialStreetPoint.y);
                double startTime = calcStartSec(streePointWay,
potentialStreetPoint.y);
                if (startTime > bestStartTime) {
                    streetConnectionPoint = node;
                    streetPoint = potentialStreetPoint;
                    bestStreetPointWay = streePointWay;
                    bestStartTime = startTime;
                }
            }
        }

        try {
            // create walking path
            resultPoints =
lisaPoint.getPath(streetConnectionPoint);
            resultPoints.add(streetPoint);

            // bus (30 km/h) starts at 7:30
            // Lisa walks with 15 km/h -> calculate time when she
has to leave home and still catches the bus
            // round up -> may walk slower
            int lisaSec = (int) Math.ceil((bestStreetPointWay /
(15.0 / 3.6)));
```

```
            // round down -> may drive faster
            int busSec = (int) Math.floor((streetPoint.getY() /
(30.0 / 3.6)));
            int timeshift = (int) Math.floor(busSec - lisaSec);

            // print out result
            System.out.println("---ERGEBNIS---");
            int startSec = 7 * 3600 + 30 * 60 + timeshift;
            System.out.println("Lisa's Weg: " +
Utils.round(bestStreetPointWay, 2) + "m");
            System.out.println("Bei y: " +
Utils.round(streetPoint.getY(), 2) + "m");
            System.out.println("Startzeit: " +
getTimeFromSec(startSec));
            System.out.println("Laufzeit: " +
getTimeFromSec(lisaSec));
            System.out.println("Ankunft: " +
getTimeFromSec(startSec + lisaSec));
            String path = "";
            for (int i = 0; i < resultPoints.size(); i++) {
                path += resultPoints.get(i).getName() + " ";
            }
            System.out.println("Pfad: " + path);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private static double calcStartSec(double lisaWay, double
busWay) {
        // bus (30 km/h) starts at 7:30
        // Lisa walks with 15 km/h -> calculate time when she has
to leave home and still catches the bus
        double lisaSec = (lisaWay / (15.0 / 3.6));
        double busSec = (busWay / (30.0 / 3.6));
        double timeshift = busSec - lisaSec;

        // print out result
        double startSec = 7 * 3600 + 30 * 60 + timeshift;
        return startSec;
    }

    private static double getStreetY(PointNode start) {
        double x = start.getX();
        double y = start.getY();
        double n = (2 * (Math.sqrt(Math.pow(y, 2) - 3 *
Math.pow(x, 2)) + 2 * y)) / 3;
        if (n > start.getY()) {
            return n;
        } else {
            return start.getY();
        }
    }

    private static boolean hasDirectConnection(PointNode start,
PointNode end) {
        Line2D line = new Line2D.Double(start, end);
        for (PolygonAdapter polygon : polygons) {
            for (Line2D polyLine : polygon.getLines()) {
                // check if line shares an end or starting point
with polygon-line
```

```
                if (line.getP1().equals(polyLine.getP1()) ||
line.getP1().equals(polyLine.getP2()) ||
                        line.getP2().equals(polyLine.getP1()) ||
line.getP2().equals(polyLine.getP2())) {
                    continue;
                }
                if (line.intersectsLine(polyLine)) {
                    return false;
                }
            }
        }
        return true;
    }
}

public class Graph {
    private List<PointNode> nodes;
    private List<Edge> edges;

    public Graph(List<PointNode> nodes, List<Edge> edges) {
        this.nodes = nodes;
        this.edges = edges;

        for (Edge edge : edges) {
            edge.getEnd().getEdges().add(edge);
            edge.getStart().getEdges().add(edge);
        }
    }

    public void execute(PointNode start) {
        start.setDistanceFromSource(0);
        PointNode currentNode = start;

        while (currentNode != null) {
            // look for neighbours, which arent visited
            for (Edge edge : currentNode.getEdges()) {
                PointNode neighbour =
edge.getNeighbourNode(currentNode);
                if (neighbour.isVisited()) {
                    continue;
                }
                double tentative =
currentNode.getDistanceFromSource() + edge.getLength();

                if (tentative < neighbour.getDistanceFromSource())
{
                    neighbour.setDistanceFromSource(tentative);
                    neighbour.setPredecessor(currentNode);
                }
            }

            // all neighbours checked so node visited
            currentNode.setVisited(true);

            // next node must be with shortest distance
            currentNode = getNodeShortestDistanced(currentNode);
        }
    }

    private PointNode getNodeShortestDistanced(PointNode current)
{
```

```
            PointNode nextNode = null;
            double nextNodeDist = Double.MAX_VALUE;

            for (PointNode node : nodes) {
                if (node.isVisited()) {
                    continue;
                }
                double distance = node.getDistanceFromSource();
                if (distance < nextNodeDist) {
                    nextNodeDist = distance;
                    nextNode = node;
                }
            }
            LisaRennt.dijkstraWay.add(new Edge(current, nextNode));
            return nextNode;
        }

        private List<PointNode> getNeighbours(PointNode node) {
            List<PointNode> neighbours = new ArrayList<>();
            for (Edge edge : node.getEdges()) {
                neighbours.add(edge.getNeighbourNode(node));
            }
            return neighbours;
        }
    }
```