Lösungsidee:

Als Information sind Strecken im zweidimensionalen Raum gegeben. Daher besitzen die Punkte nur eine Abszisse und eine Ordinate. Die Aufgabe wird so vereinfacht, dass keine zwei Strecken auf derselben Gerade liegen und dass sich nie mehr als zwei Strecken im gleichen Punkt schneiden. Nach meinem Verständnis müsste man die Kanten der jeweiligen Dreiecke auch auf einem Blatt Papier ohne Verlängerung einer Strecke in einem Zug nachfahren können, damit sie auch als Dreiecke bezeichnet werden können.

Zuerst zeichnet man die Strecken in einem Koordinatensystem. Sowohl die Punkte der Strecken als auch die Schnittpunkte zweier oder mehrerer Strecken werden markiert. Jetzt bildet man alle möglichen Kombinationen aus drei Punkten, wobei die Reihenfolge der Punkte vernachlässigt wird (d.h. man streicht Mehrfachnennungen durch). Man geht die Liste der Kombinationen dann der Reihe nach durch:

Es existiert ein Dreieck, wenn man von einem Punkt zu den anderen zwei Punkten gelangen kann über die gegebenen Strecken, ohne dass die drei Punkte auf einer Strecke liegen. Folglich erhöht sich die Zahl der Dreiecke um eins.

Im zweidimensionalen Raum können folgende Lagen zweier Geraden existieren:

1. Die Geraden sind parallel zueinander.

2. Die Geraden schneiden sich in genau einem Punkt.

Mathematische Herleitung der benötigten Formeln:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Um den Schnittpunkt zu bekommen, werden die Werte jeweils in die entsprechenden Geraden eingesetzt. Da die Aufgabe keine Geraden, sondern Strecken enthält, müssen t und s >= 0 und <=1 sein (zwischen Start- und Endpunkt der beiden Strecken).

Bei den restlichen Fällen kann auf analoger Weise gezeigt werden, dass die Richtungsvektoren stets linear unabhängig sind, die Geraden sind damit nicht parallel und müssen sich in einem Punkt schneiden.

Mathematisch betrachtet (Mir ist diese Idee erst nach der Programmierung eingefallen, daher sieht die Implementierung anders aus.):

Man betrachtet das Ganze als ein Graph mit Knoten, die durch die Punkte symbolisiert werden, und mit Kanten, die durch die Strecken symbolisiert werden. Als erstes markiert man die Schnittpunkte der Kanten als neue Knoten. Dann beginnt man mit einem der Knoten wie folgt:

Man sucht nach Zyklen mit der Weglänge drei. Die Kanten, die in diesem Weg enthalten sind, müssen sich paarweise unterscheiden. Wenn der Knoten einen Knotengrad kleiner als 2 besitzt, dann kann es keinen solchen Zyklus geben, also wird der übersprungen. Sonst betrachtet man immer paarweise zwei Kanten, die von dem betrachteten Knoten ausgehen. Gibt es eine Kante, die die beiden Endknoten verbindet, so erhöht sich die Anzahl der Dreiecke um eins.

Ergibt sich derselbe Zyklus bei einem anderem Knoten, so wird dieser einfach übersprungen.

Mir gefällt diese Idee sehr, da sie viel systematischer an die Aufgabe rangeht. Wahrscheinlich würde man so Laufzeit sparen, da man mit meiner Methode viel zu „brute-force“-mäßig arbeitet.

Umsetzung:

Es existieren drei Klassen:

Die Klasse Punkt beschreibt einen Punkt in einem zweidimensionalen Raum. Ein Punkt wird hier beschrieben durch seine Abszisse (private double x) und seine Ordinate (private double y). Die Methode public boolean equals(Punkt p) überprüft zwei Punkte auf Gleichheit, indem die beiden Abszissen und Ordinaten miteinander vergleicht werden. Der Rest der Klasse besteht aus Getter- und Setter-Methoden.

Die Klasse Strecke beschreibt eine Strecke in einem zweidimensionalen Raum. Eine Strecke wird beschrieben durch seinen Start- und Endpunkt im Array private Punkt[] punkte. Die Methode public boolean equals(Strecke s) überprüft zwei Strecken auf Gleichheit. Dabei wird geschaut, ob entweder die beiden Start- und Endpunkte jeweils einander gleichen oder der eine Start- und der eine Endpunkt jeweils einander gleichen. Dies ist sinnvoll, da sich durch Vertauschen des Starts- mit dem Endpunkt keine neue Strecke ergeben soll. Der Rest der Klasse enthält Setter- und Getter-Methoden.

In der Klasse Koordinatensystem wird das Problem gelöst. Ein Koordinatensystem wird beschrieben durch Punkte (private ArrayList<Punkt> punkte), Schnittpunkte der Strecken (private ArrayList<Punkt> schnittpunkte), Punkte auf Strecken (private ArrayList<ArrayList<Punkt>> punkteAufStrecken), Strecken (private ArrayList<Strecke> strecken) und durch Dreiecke (private ArrayList<ArrayList<Punkt>> dreiecke). Die wichtigsten Methoden werden im Folgenden näher erläutert:

- public int anzahlDreiecke()

Diese Methode gibt die Anzahl der Dreiecke zurück und fügt die Eckpunkte der Dreiecke der Dreiecksliste zu. In drei aufeinanderfolgenden for-Schleifen werden alle Kombinationen aus drei Punkten des Koordinatensystems kreiert. Der Zähler anzahlDreiecke enthält die Anzahl der bisher gefundenen Dreiecke. Punktekonstellationen, bei welchen zwei Punkte einander gleichen, werden in einer if-Abfrage abgefangen und dann übersprungen. In einer letzten if-Abfrage wird überprüft, ob die drei Punkte durch genau drei Strecken miteinander verbunden sind. Dieses Dreieck wird auch nur dann hinzugefügt, wenn es nicht bereits in der Dreiecksliste existiert. Innerhalb dieser if-Abfrage erhöht sich auch anzahlDreiecke um eins. Nach dem Durchlaufen der drei for-Schleifen wird anzahlDreiecke zurückgegeben.

- public void setPunkteAufStrecken()

In dieser Methode werden die Punkte, die auf einer Strecke liegen, in einer Liste gesammelt. In einer for-Schleife werden alle Strecken durchlaufen. In dieser wird die Gerade g mit Ortsvektor (p1, p2) und Richtungsvektor (u1, u2) gebildet. P1 und p2 stellen jeweils die Abszisse und Ordinate des Startpunkts der aktuellen Strecke. U1 und u2 stellen jeweils die Differenz von der Abszisse/Ordinate des Endpunkts der Strecke und p1/p2. In einer weiteren, inneren for-Schleife werden alle Punkte durchlaufen. Drei if-Abfragen in dieser Schleife dienen zum Unterscheiden der verschiedenen Fälle der Punktprobe, wie sie in der Lösungsidee beschrieben wurde. In der Umsetzung wurde das Verfahren insofern modifiziert, dass die Werte der Parameter um 0,001 von 0 bzw. 1 abweichen dürfen. Dies muss garantiert werden, weil mit gerundeten Werten gerechnet wird. Je nachdem werden die Punkte in den Fällen, in welchen sie auf der Strecke liegen, der entsprechenden Liste hinzugefügt.

- public void setStrecken(String datei)

Mithilfe dieser Methode werden die Strecken aus der Textdatei eingelesen. Zuerst werden die einzelnen Textzeilen in einer while-Schleife so lange eingelesen, bis das Dateiende erreicht wurde. Darin werden in einer for-Schleife alle Zeichen durchlaufen. Die Variable temp des Typs String erhöht sich so lange um ein Zeichen der Textzeile, bis ein Leerzeichen vorkommt. Das Leerzeichen signalisiert das Ende der Zahl, die alternierend erst der entsprechenden Abszisse, dann der Ordinate des Punkts zug