Aufgabe 3: Dreiecke zählen

Aufgabe

Schreibe ein Programm, das die Dreiecke in einer Rätsel-Zeichnung zählt. Eine Zeichnung besteht aus einigen Strecken. Du kannst davon ausgehen, dass keine zwei Strecken auf derselben Geraden liegen und dass sich nie mehr als zwei Strecken im gleichen Punkt schneiden.
Wende dein Programm auf die Beispiele an, die du auf den BwInf-Webseiten findest.

Lösungsidee

Nachdem man die Schnittpunkte der gegebenen Strecken berechnet hat, kann man einen ungerichteten Graphen erstellen, mit den Anfangs-, End- und Schnittpunkten als Knoten und Strecken als Kanten.
Auf diesem Graph kann man eine modifizierte Tiefensuche durchführen, über die man eine Liste aller Wege der Länge 3 im Graphen erhält.
Aus dieser Liste werden dann diejenigen Wege entfernt, die kein Dreieck beschreiben. Wenn beispielsweise der Anfangspunkt nicht gleich der Endpunkt ist, kann der Weg kein Dreieck sein.
Nach der Bereinigung der Liste ist die Anzahl der Dreiecke gleich die Anzahl der Elemente in der Liste.

Umsetzung

Die Lösungsidee wird in Python 3.6 implementiert.

Zu Beginn wird die Datei eingelesen und in drei Strukturen konvertiert:

1) ein Dictionary, welches jedem Punkt alle Geraden zuordnet, die diesen berühren
2) ein Dictionary, welches jeder Geraden alle Punkte zuordnet, die sie berührt
3) ein Set, welches alle Punkte enthält

Daraufhin werden alle Schnittpunkte im Graphen G bestimmt. Dafür muss man mit zwei verschachtelten Schleifen für jede Zweierkombination an Strecken in G deren Schnittpunkt berechnen, falls dieser existiert.

Den Schnittpunkt zweier Strecken berechnet man wie folgt:

$P_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ und $P_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$ sind Start- und Endpunkte der Strecke s_1

$P_3 = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}$ und $P_4 = \begin{pmatrix} x_4 \\ y_4 \end{pmatrix}$ sind Start- und Endpunkte der Strecke s_2

Zuerst berechnet man den Schnittpunkt[1] der Geraden g_1 und g_2,
wobei g_1 auf den Punkten P_1 und P_2
und g_2 auf den Punkten P_3 und P_4 liegt.

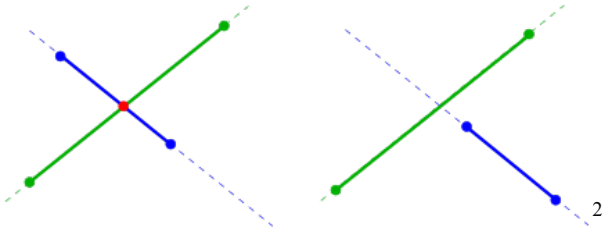
Der Schnittpunkt $S = \begin{pmatrix} x_s \\ y_s \end{pmatrix}$ ergibt sich aus

$$x_s = \frac{(x_4 - x_3)(x_2 y_1 - x_1 y_2) - (x_2 - x_1)(x_4 y_3 - x_3 y_4)}{(y_4 - y_3)(x_2 - x_1) - (y_2 - y_1)(x_4 - x_3)}$$

und

$$y_s = \frac{(y_1 - y_2)(x_4 y_3 - x_3 y_4) - (y_3 - y_4)(x_2 y_1 - x_1 y_2)}{(y_4 - y_3)(x_2 - x_1) - (y_2 - y_1)(x_4 - x_3)}$$

Liegt x_s innerhalb $[x_1, x_2]$ und $[x_3, x_4]$ und y_s innerhalb $[y_1, y_2]$ und $[y_3, y_4]$ schneiden sich $s_1$und s_2.



s_1: blaue Strecke, s_2: grüne Strecke

Die beiden Geraden auf der rechten Seite schneiden sich nicht, da x_s nicht in $[x_1, x_2]$ und y_s nicht in $[y_1, y_2]$ liegen.

[1] https://de.wikipedia.org/wiki/Schnittpunkt#Schnittpunkt_zweier_Geraden
[2] Von Ag2gaeh - Eigenes Werk, CC-BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44999745

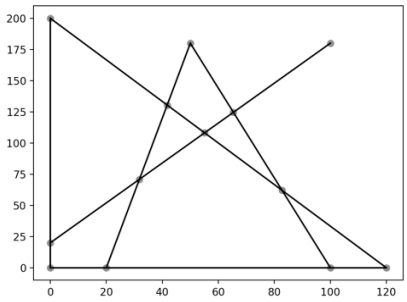
Wenn sich beide Strecken schneiden, werden die beiden Dictionaries und das Set angepasst.

Auf dem vervollständigten Graphen kann man nun eine modifizierte Tiefensuche anwenden.
Mithilfe dreier verschachtelter while-Schleifen werden alle Wege, die drei Kanten lang sind betrachtet.
Mithilfe von Struktur (1) und (2) werden die vom gerade betrachteten Punkt p_1 erreichbaren Punkte bestimmt und in der Liste e_1 gespeichert. In der zweiten Schleife wird über e_1 iteriert und für jeden p_2 eine Liste an erreichbaren Punkten e_2 berechnet. Die dritte Schleife iteriert über e_2 und bestimmt die von p_2 erreichbaren Punkte e_3. Befindet sich der Startpunkt p_1 in e_3 ist p_1 gleichzeitig der Endpunkt des Weges. Ein das 3-Tupel (p_1, p_2, p_3) wird daraufhin sortiert in einem Set gespeichert, damit keine Dreiecke mehrfach gezählt werden.
Die Länge des Sets ist somit auch die Anzahl der Dreiecke.

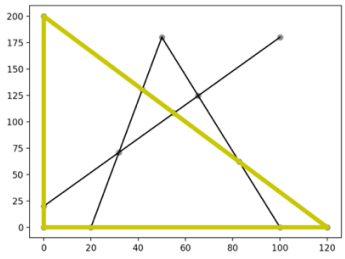
Das Script wird wie folgt ausgeführt:
`dreiecke.py FILE [--visualize|-v]`
Wählt man durch *–visualize* oder *–v* die Visualisierung durch pyplot aus, muss man dieses Modul bereits installiert haben. Solange nicht alle Dreiecke gezeigt wurden, öffnet sich ein neues Fenster, welches das nächste Dreieck hervorhebt, wenn man das alte Fenster schließt.

Beispiele

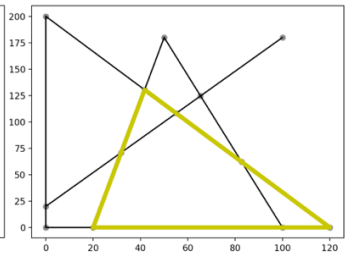
```
./dreiecke.py txt/dreiecke1.txt –v
```



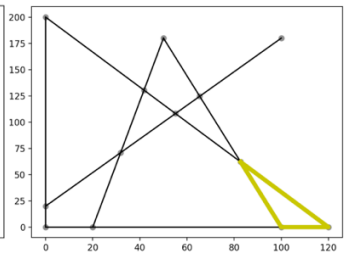
Visualisierung des Graphen



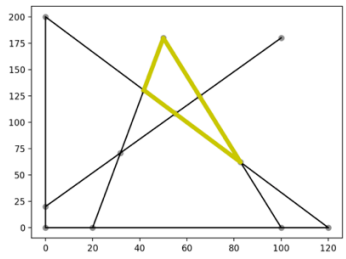
Dreieck 1



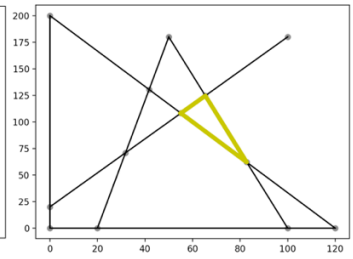
Dreieck 2



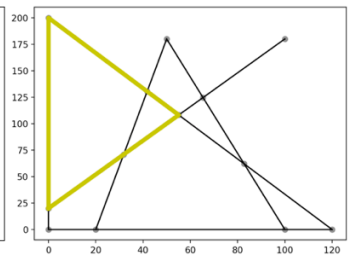
Dreieck 3



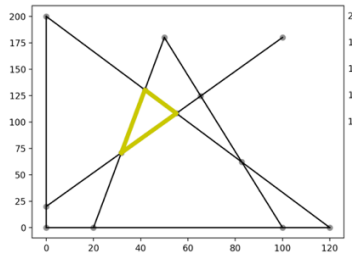
Dreieck 4



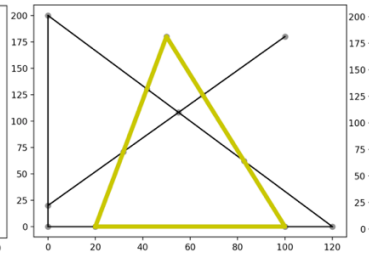
Dreieck 5



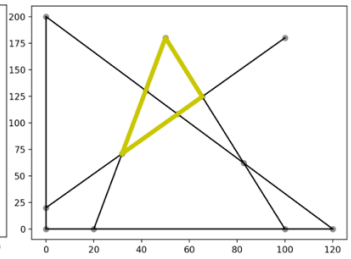
Dreieck 6



Dreieck 5



Dreieck 6

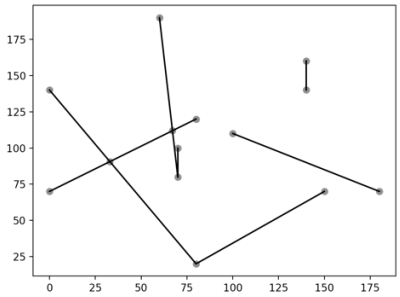


Dreieck 7

```
1 -> ((0.0, 0.0), (0.0, 200.0), (120.0, 0.0))
2 -> ((20.0, 0.0), (41.73913043478261, 130.43478260869566),
(120.0, 0.0))
3 -> ((82.75862068965517, 62.06896551724138), (100.0, 0.0),
(120.0, 0.0))
4 -> ((41.73913043478261, 130.43478260869566), (50.0, 180.0),
      (82.75862068965517, 62.06896551724138))
5 -> ((55.10204081632653, 108.16326530612245),
      (65.38461538461539,124.61538461538461),
      (82.75862068965517, 62.06896551724138))
```

```
6 -> ((0.0, 20.0), (0.0, 200.0), (55.10204081632653,
108.16326530612245))
7 -> ((31.818181818181817, 70.9090909090909),
      (41.73913043478261,130.43478260869566),
      (55.10204081632653, 108.16326530612245))
8 -> ((20.0, 0.0), (50.0, 180.0), (100.0, 0.0))
9 -> ((31.818181818181817, 70.9090909090909),
      (50.0, 180.0), (65.38461538461539,124.61538461538461))
Anzahl Dreiecke:  9
```

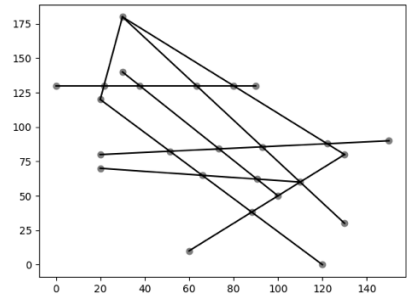
./dreiecke.py txt/dreiecke2.txt –v



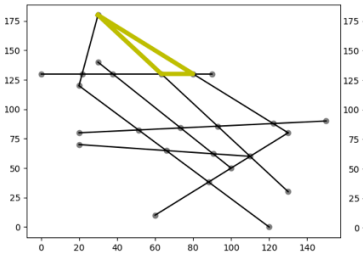
Visualisierung des Graphen

```
Anzahl Dreiecke:  0
```

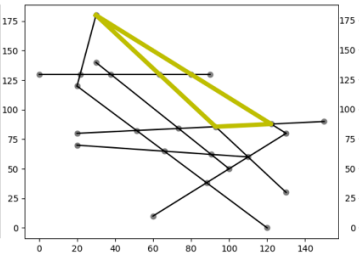
./dreiecke.py txt/dreiecke3.txt –v



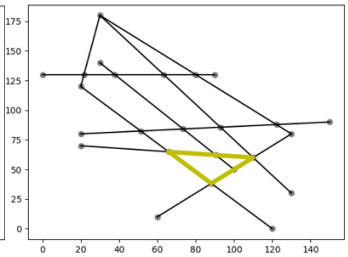
Visualisierung des Graphen



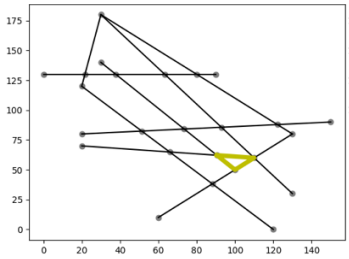
Dreieck 1



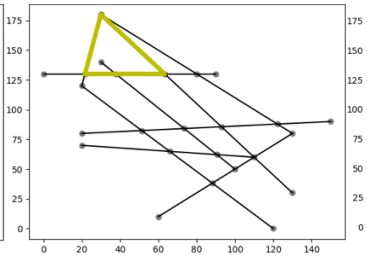
Dreieck 2



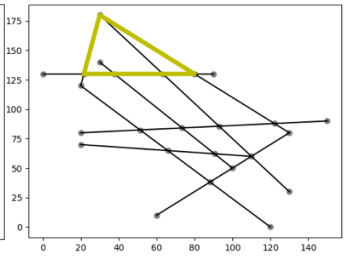
Dreieck 3



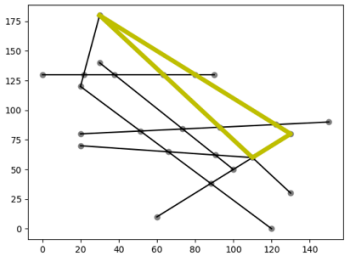
Dreieck 4



Dreieck 5



Dreieck 6



Dreieck 7

```
1 -> ((30.0, 180.0), (63.333333333333336, 130.0), (80.0, 130.0))
2 -> ((30.0, 180.0), (92.92682926829268, 85.60975609756098),
      (122.14285714285714, 87.85714285714286))
3 -> ((65.91836734693878, 64.89795918367346),
      (88.18181818181819,38.18181818181818), (110.0, 60.0))
4 -> ((90.54054054054055, 62.16216216216216), (100.0, 50.0),
(110.0, 60.0))
5 -> ((21.666666666666668, 130.0), (30.0, 180.0),
(63.333333333333336, 130.0))
6 -> ((21.666666666666668, 130.0), (30.0, 180.0), (80.0, 130.0))
7 -> ((30.0, 180.0), (110.0, 60.0), (130.0, 80.0))
Anzahl Dreiecke:  7
```