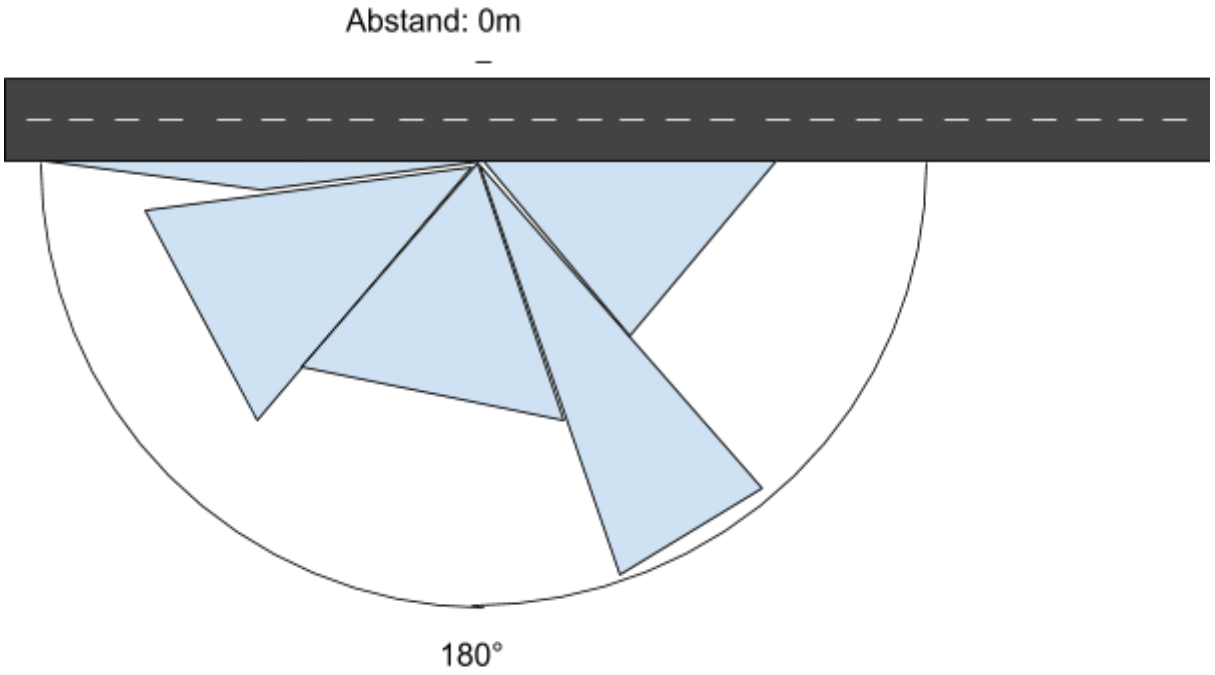
Aufgabe 2: “Dreiecksbeziehungen”

Inhaltsverzeichnis

Grundidee

Es ist am besten die Dreiecke in einem Halbkreis anzuordnen, so sind alle “Türen direkt beieinander ”, der gesamte Abstand beträgt 0.

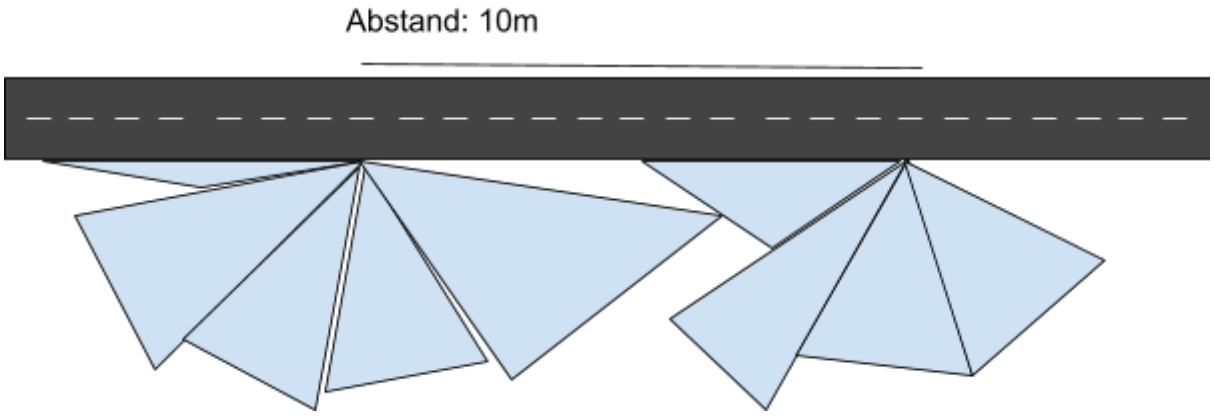
Man muss also die Dreiecke in mehreren Halbkreisen anordnen ,da man so wenig Platz verbraucht.



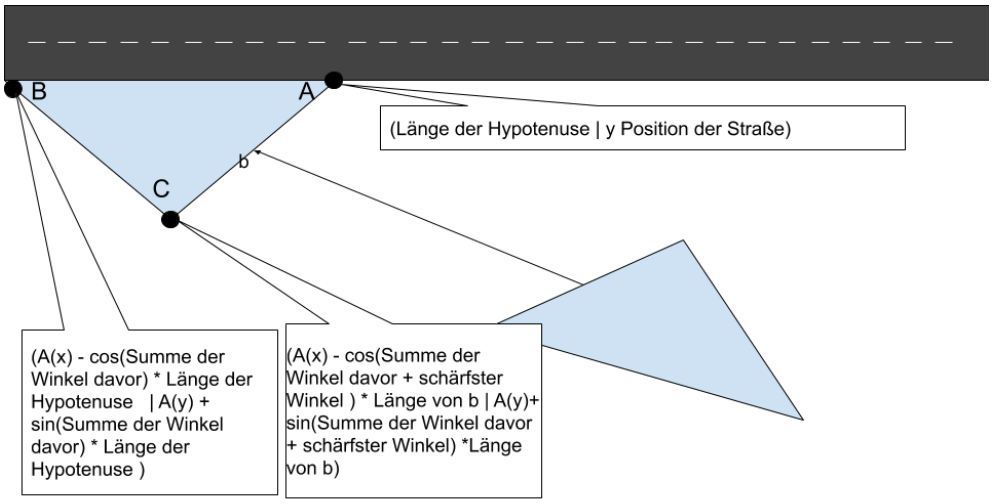
Wenn die Dreiecke zusammen weniger als **180°** haben hat man sogar einen Abstand von 0 Metern.

Lösungsidee

Zuerst werde ich alle Dreiecke so in Gruppen aufteilen, dass die Summe ihrer kleinsten Winkel nah an 180 Grad ist, damit die Halbkreise voll ausgefüllt sind.



Wie werden die Dreiecke an die Straße platziert



Zur Erklärung der Zeichnung: Die “Summe der Winkel davor” sind alle schärfsten Winkel von Dreiecken in diesem Halbkreis zusammen gerechnet. Die “Summe der Winkel davor” ist bei diesem Dreieck 0, da es das erste Dreieck in diesem Halbkreis ist.
So werden alle Dreiecke in einem Halbkreis angeordnet.

Wie werden die Dreiecke in Gruppen aufgeteilt ?

Es wird versucht die Dreiecke in Gruppen zu platzieren, deren kleinste Winkel zusammen 180 ° haben. Dafür werden die Dreiecke zusammen kombiniert und die kürzesten WInkel addiert. DIeser Wert wird gespeichert, dann werden andere Kombinationen ausprobiert und geschaut, ob diese besser sind. Wenn diese besser sind, wird die neue beste Kombination und die beste Winkelsumme als neue beste gespeichert.

Es werden ersten alle Kombinationen durchgegangen mit nur einem Dreieck dann mit zwei dann 3 ,usw.
Insgesamt werden so viele Kombinationen miteinander verglichen :

$$\sum_{i=1}^{n} n! \,/\, i! \,/\, (n - i)!$$

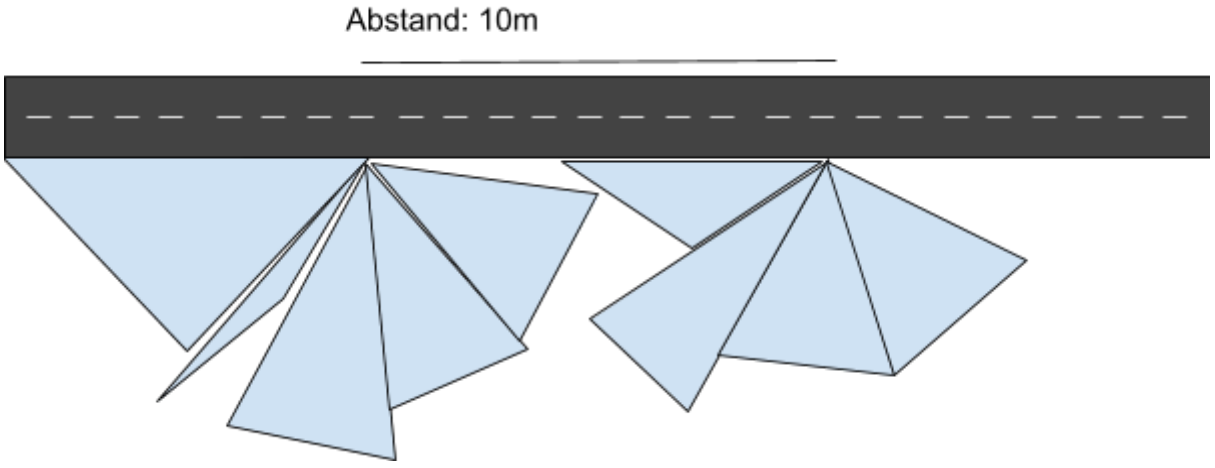
n ist die **Anzahl an Dreiecken**

Der Nachteil dieser Lösung ist die Rechendauer, aufgrund der enorm vielen Kombinationsmöglichkeiten!

`Umsetzung im Code :`

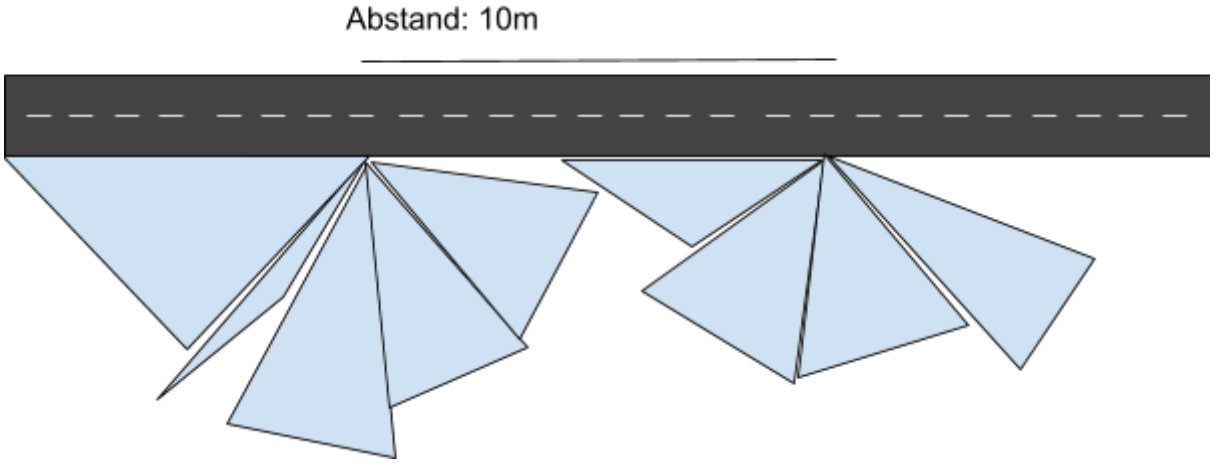
Optimierung für ein besseres Ergebnis

Die Dreiecke des ersten Halbkreis sortieren wir nach der Länge der Hypotenusen von den Dreiecken. Dies macht Sinn, da so weniger Platz zwischen dem ersten und dem zweiten Halbkreis ist: Die Halbkreise werden also näher aneinander gerückt.



Dreiecke des ersten Halbkreis sind jetzt nach der Hypotenusen-Länge sortiert und haben dadurch einen geringeren Abstand zueinander.

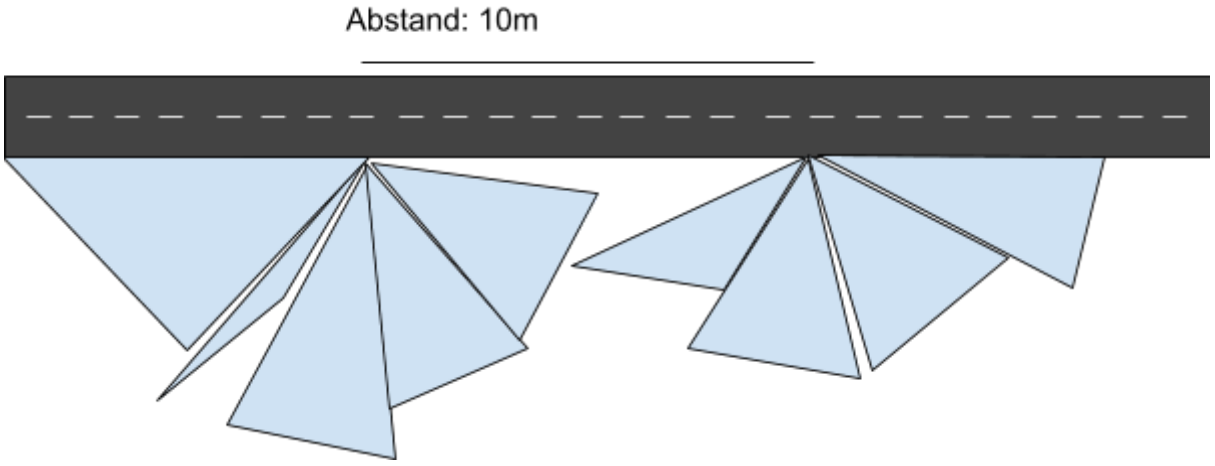
Anstatt es beim 2. (in diesem Fall letzten Halbkreis) genauso zu machen, wird anders herum sortiert, zuerst die kurzen Dreiecke, dann die Langen.



In diesem Fall hat sich jetzt nicht viel verändert, aber allgemein verringert es den Abstand.

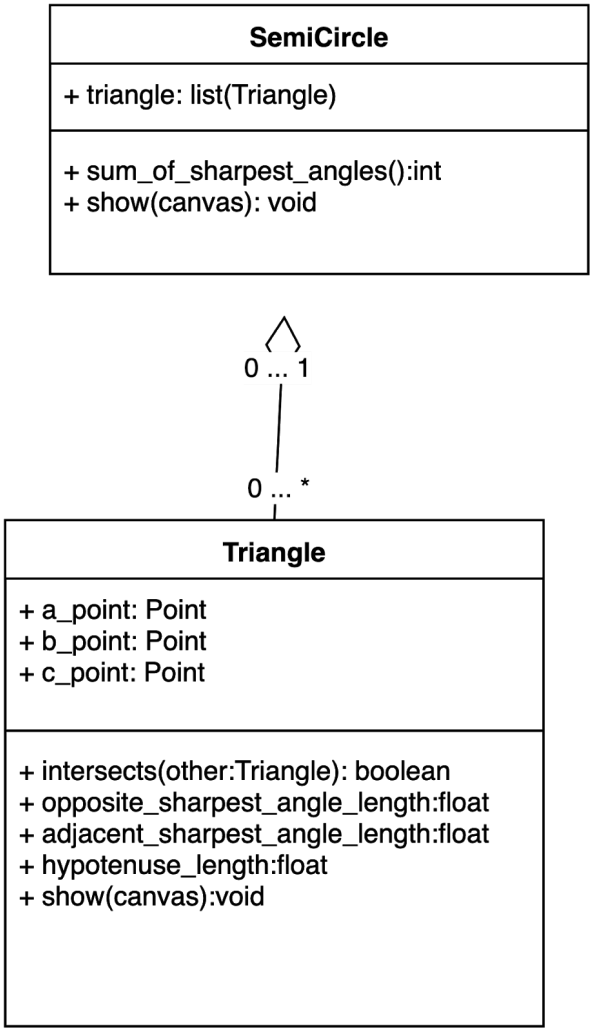
Die Halbkreise zwischen dem ersten und dem letzten Halbkreis werden so sortiert, dass die längsten Dreiecke in der Mitte sind. An unserem Beispiel lässt sich dies nicht zeigen, da es nur zwei Halbkreise gibt.

Um einen noch geringeren Abstand zu bekommen, sind die Dreiecke des letzten Halbkreises so anzuordnen ,dass die Lücke nicht hinten ist, sondern vorne.Dies lässt sich besser an einer Zeichnung zeigen:



Das UML Diagramm dieser Aufgabe

Es gibt nur zwei Klassen, nämlich Triangle und SemiCircle:

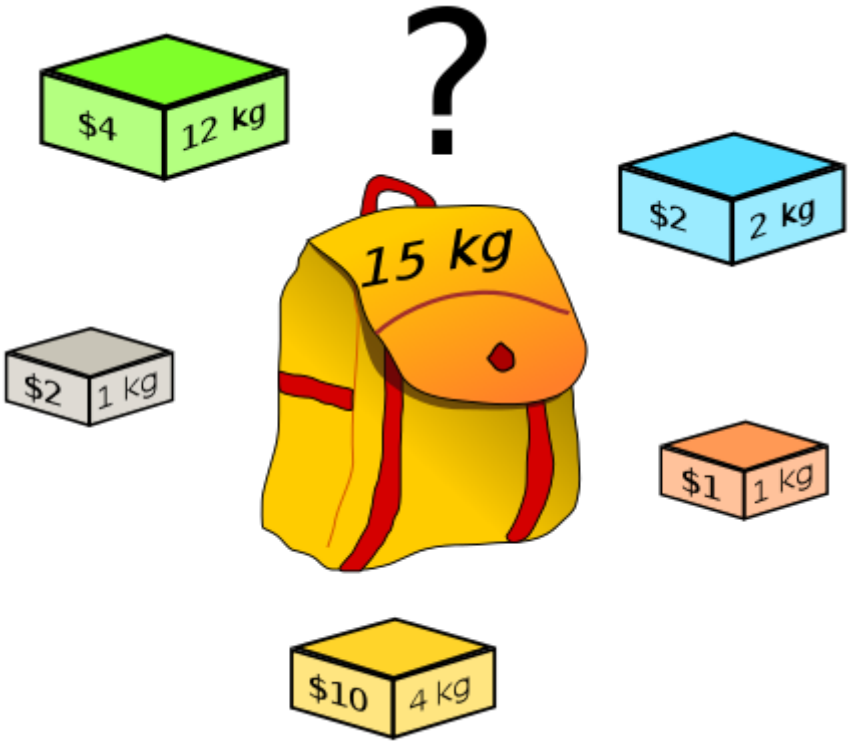


Es gibt im Moment noch das Problem, dass es viel zu lange dauert die Dreiecke so zu sortieren, dass die Summe der kürzesten Winkel möglichst nah an 180 ran kommt, da alle Kombinationen durchgegangen werden. Es wäre also schlau einen Algorithmus zu schreiben, welcher schneller die Dreiecke findet, deren schärfste Winkel zusammen 180 Grad ergeben.

Mit meiner jetzigen Lösung habe ich noch eine Zeitkomplexität von **O(n!),** was sehr schlecht ist**.**

Optimierung unter Anwendung des Rucksackproblems

Nach einer Internet-Recherche bin ich auf das **Rucksackproblem (engl. knapsack)** gestoßen, welches dieses Problem löst. Beim Rucksackproblem möchte man Sachen mit den größten Werten mitnehmen, aber das maximale Gewicht des Rucksacks darf nicht überschritten werden. Jeder Wert ist einem Gewicht zugeordnet.



https://de.wikipedia.org/wiki/Rucksackproblem

In unserem Fall ist das “maximal Gewicht” 180 °, die “Gewichte” sind die kürzesten Winkel der jeweiligen Dreiecke, die Werte sind auch die kürzesten Winkel der Dreiecke.

Algorithmus für das Rucksackproblem:

Zur Vereinfachung gehen wir davon aus, dass unser **maximales Gewicht für den Rucksack 7 kg** ist, außerdem entsprechen bei uns die Gewichte den Werten.
Die Gewichte sind 1 kg, 3 kg, 5 kg.

Nun müssen wir zuerst ein zweidimensionales Array erzeugen: Es muss so viele Zeilen haben wie Gewichte und so viele Spalten wie groß das Maximalgewicht plus 1 ist.

Also 8 * 3. Die Spalten sind das maximale Gewicht, das der Rucksack haben darf. Die Zeilen sind die vorhandenen Gewichte.

	0	1	2	3	4	5	6	7
1								
3								
5								

Nun füllen wir die Tabelle mit Werten auf. In jeder Zelle soll immer das größtmögliche Rucksackgewicht stehen, wenn nur dieses Gewicht der Zeile und kleinere Gewichte beachtet werden.
In die erste Spalte kommen nur Nullen, da wir bei einem Maximalgewicht von 0, nie einen Wert > 0 haben.

	0	1	2	3	4	5	6	7
1	0							
3	0							
5	0							

In die Zelle (1,1) kommt der Wert 1, da man bei einem Maximalgewicht von 1 und wenn man nur das Gewicht 1 verwenden darf, als besten Wert 1 hat.

	0	1	2	3	4	5	6	7
1	0	1						
3	0							
5	0							

Wenn wir nur das Gewicht 1 verwenden können, haben wir bei höheren Maximalgewichten, 1 als bestes Gewicht.

	0	1	2	3	4	5	6	7
1	0	1	1	1	1	1	1	1
3	0							
5	0							

Nächste Zeile:
In der zweiten Zeile der Tabelle können wir die Gewichte 3 und 1 verwenden. Da wir in den Spalten mit den maximal Gewichten 1 und 2 das Gewicht 3 nicht verwenden können, müssen wir dort 1 eintragen. Das heißt, auch wenn der Rucksack mit 2 kg beladen werden kann, können wir bestenfalls nur 1 kg mitnehmen.

	0	1	2	3	4	5	6	7
1	0	1	1	1	1	1	1	1
3	0	1	1					
5	0							

In der Zelle (3,3) wird nun das Gewicht 3 eingetragen, da wir, nachdem wir das Gewicht 3 verwendet haben, kein weiteres Gewicht mehr verwenden können.

	0	1	2	3	4	5	6	7
1	0	1	1	1	1	1	1	1
3	0	1	1	3				
5	0							

Ins nächste kommt dann 4, da wir, nachdem wir das 3kg-Gewicht in den Rucksack getan haben, immer noch 1 kg Gewicht frei haben und wir dann noch das 1 kg Gewicht hineinlegen können. Dieser Prozess wird im gleichen Prinzip für das ganze zweidimensionale Array ausgeführt.

Am Ende schaut es dann so aus:

	0	1	2	3	4	5	6	7
1	0	1	1	1	1	1	1	1
3	0	1	1	3	4	4	4	4
5	0	1	1	3	4	5	6	6

Das beste Ergebnis ist also 6. Nun wollen wir noch herausfinden, welche Gewichte verwendet wurden, um auf die 6 kg zu kommen.

Da die 6 nicht in der Zelle eine Zeile oben drüber ist, wissen wir, dass 5kg verwendet wurden. Die 6 kg wurden also erst erreicht, nachdem wir das 5 kg-Gewicht eingepackt haben.Wir können nun noch 1 kg in den Rucksack packen, also müssen wir das Element in der Zeile eins oben drüber anschauen. Es ist 1, dieses Element wurde von der Zelle oben drüber übernommen. Also müssen wir uns dieses Element anschauen, dieses wurde nicht

von der Zeile oben drüber übernommen, da es die erste Zeile ist, also sind wir fertig. Es wurden 5kg und 1 kg verwendet.

```
def knapsack(allowed_weight, items):
    k = [
        [0 for x in range(allowed_weight + 1)]
        for x in range(len(items) + 1)
    ]

    for next_idx, (item, weights) in enumerate(zip(items, k), 1):
        for w, current_weight in enumerate(weights[1:], 1):
            if item.weight <= w:
                k[next_idx][w] = max(
                    item.weight + weights[w - item.weight],
                    current_weight
                )
            else:
                k[next_idx][w] = current_weight

    return k[-1][-1], list(fetch_items(k, allowed_weight, items))

# find which items are picked


def fetch_items(k, allowed_weight, items):
    for item, weights_p, weights_n in zip(items[::-1], k[-2::-1],
k[::-1]):
        if weights_n[allowed_weight] != weights_p[allowed_weight]:
            yield item
            allowed_weight -= item.weight
```

Dieser Algorithmus braucht nur noch **O(nW)** Kombinationen. Dabei ist **n** die Anzahl an Dreiecken und **W** ist das maximale Gewicht, bzw in unserem Fall 180 Grad, deswegen haben wir O(180n). Der Zeitbedarf für den Algorithmus steigt linear zu der Anzahl an Dreiecken, also nur **O(n)**. Diesen Algorithmus müssen wir aber für jeden Halbkreis wiederholen. Also steigt auch die Anzahl an Wiederholungen, wenn wir mehr Dreiecke haben. Da wir nicht genau wissen, wie viele Halbkreise wir bei wie vielen Dreiecken brauchen, ist es also schwer darüber eine Aussage zu treffen. Bei jeder Wiederholung müssen wir aber weniger Elemente durchgehen. Also haben wir **O(n log n)**.

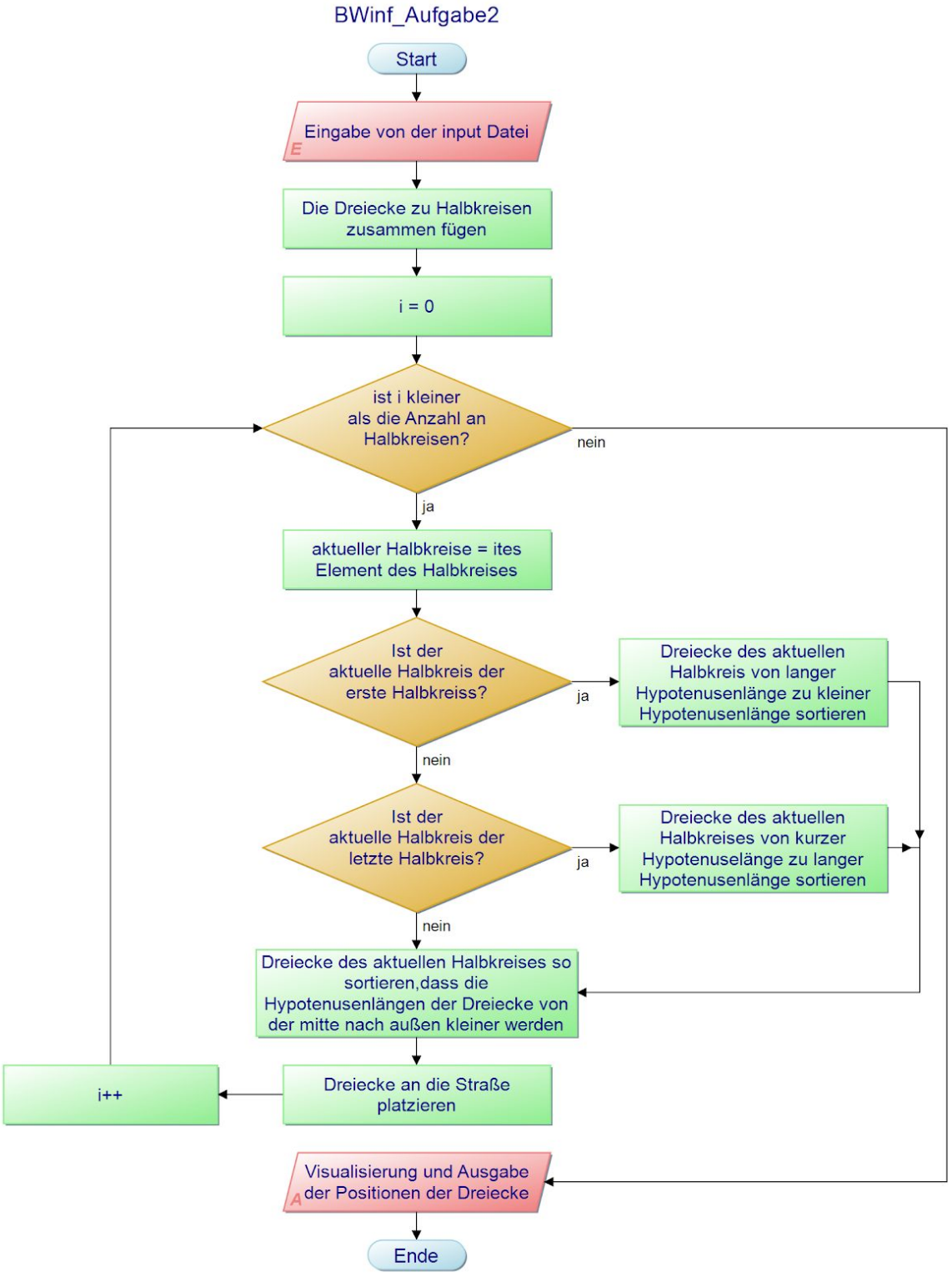
Um die Dreiecke herauszusuchen, die zusammen möglichst nah an 180° sind, habe ich die Funktion **make_semi_cicrle_list** programmiert. Diese ruft die “knapsack”-Funktion auf:

```
def make_semi_circle_list(triangles):
    triangles_copy = triangles[:]
    semi_circles = []
    while len(triangles_copy) > 0:
        items = [Item(triangle, ceil(triangle.sharpest_angle)) for
triangle in triangles_copy]
        max_value, picked_list =
knapsack(MAXIMUM_ANGLE_SEMI_CIRCLE, items)
        best_triangles = [picked.triangle for picked in
picked_list]
        semi_circles.append(SemiCircle(best_triangles))
        triangles_copy = [triangle for triangle in triangles_copy
if (triangle not in best_triangles)]
    return semi_circles
```

Warum habe ich den Knapsack Algorithmus verwendet ?

Das “Bin packing”-Problem handelt davon,dass man versucht Gewichte in möglichst wenige Behälter zu verteilen. Ich habe auch den “Bin packing first in decreasing”-Algorithmus implementiert und verwendet, aber dieser hat schlechtere Ergebnisse erzielt als der Knapsack-Algorithmus, deswegen bin ich beim Knapsack-Algorithmus geblieben.

PAP:


BWinf_Aufgabe2
Start
Eingabe von der input Datei
E
Die Dreiecke zu Halbkreisen zusammen fügen
i = 0
ist i kleiner als die Anzahl an Halbkreisen?
nein
ja
aktueller Halbkreise = ites Element des Halbkreises
Ist der aktuelle Halbkreis der erste Halbkreiss?
ja
Dreiecke des aktuellen Halbkreis von langer Hypotenusenlänge zu kleiner Hypotenusenlänge sortieren
nein
Ist der aktuelle Halbkreis der letzte Halbkreis?
ja
Dreiecke des aktuellen Halbkreises von kurzer Hypotenuselänge zu langer Hypotenusenlänge sortieren
nein
Dreiecke des aktuellen Halbkreises so sortieren,dass die Hypotenusenlängen der Dreiecke von der mitte nach außen kleiner werden
Dreiecke an die Straße platzieren
i++
Visualisierung und Ausgabe der Positionen der Dreiecke
A
Ende

Beispielaufgaben:

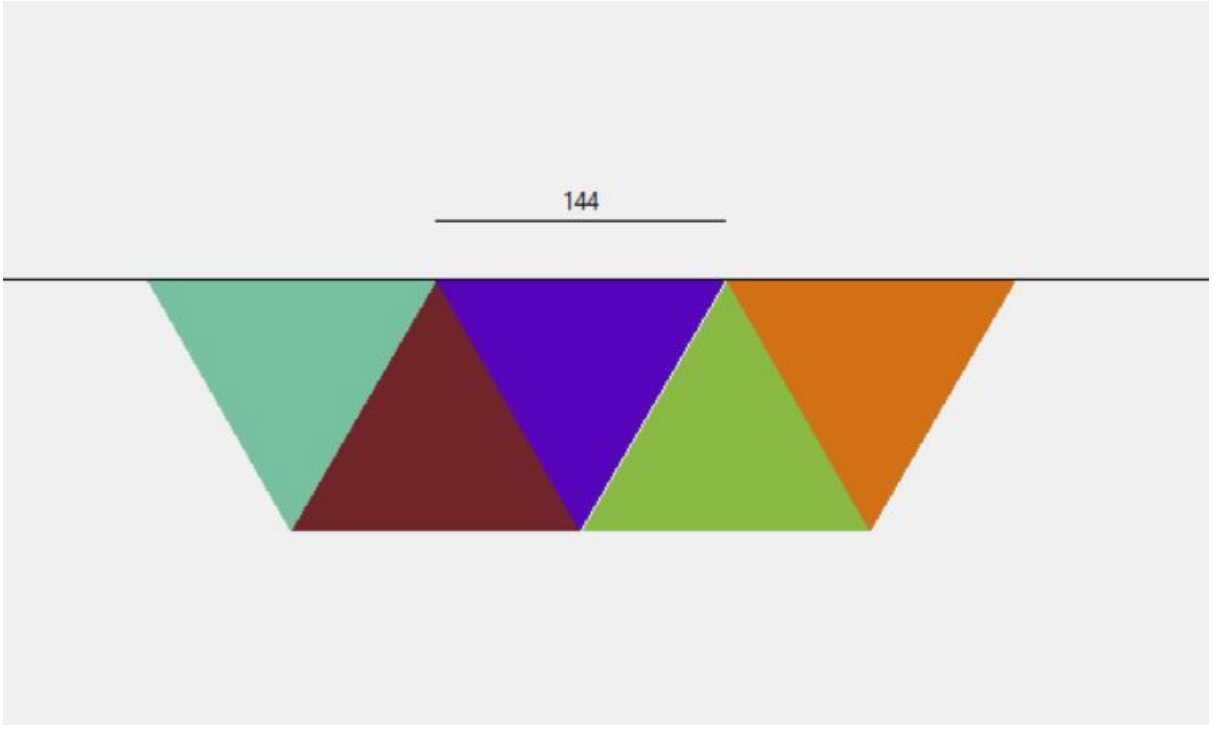
dreiecke1.txt

Ausgabe:

Distanz; Dreiecke(Id und Ecken)

257, [['D3: (143|400), (0|400), (71|524)', 'D2: (143|400), (71|524), (214|524)', 'D1: (143|400), (214|524), (286|400)'], ['D5: (287|400), (215|524), (358|524)', 'D4: (287|400), (358|524), (430|400)']]

Visualisierung:

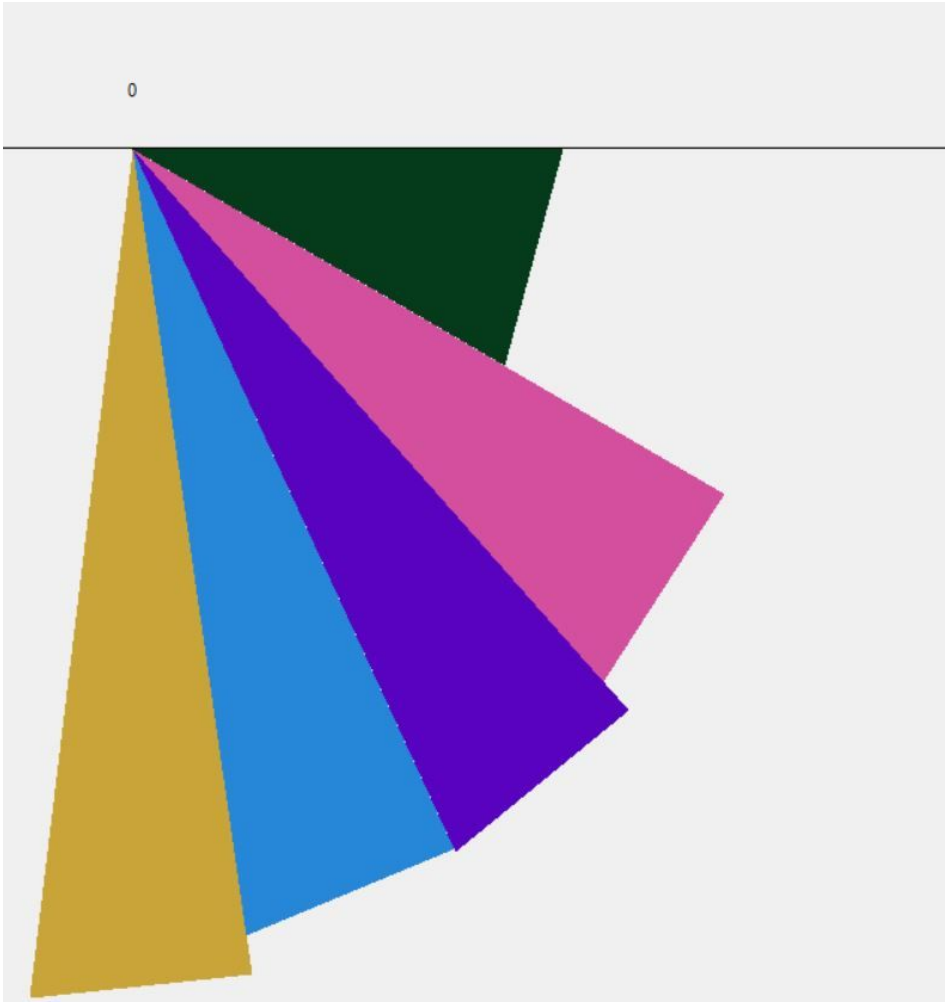


dreiecke2.txt

Ausgabe:

0, [['D2: (100|200), (31|769), (180|753)', 'D4: (100|200), (176|727), (315|669)', 'D5: (100|200), (316|671), (432|576)', 'D1: (100|200), (415|557), (496|431)', 'D3: (100|200), (349|345), (388|200)']]

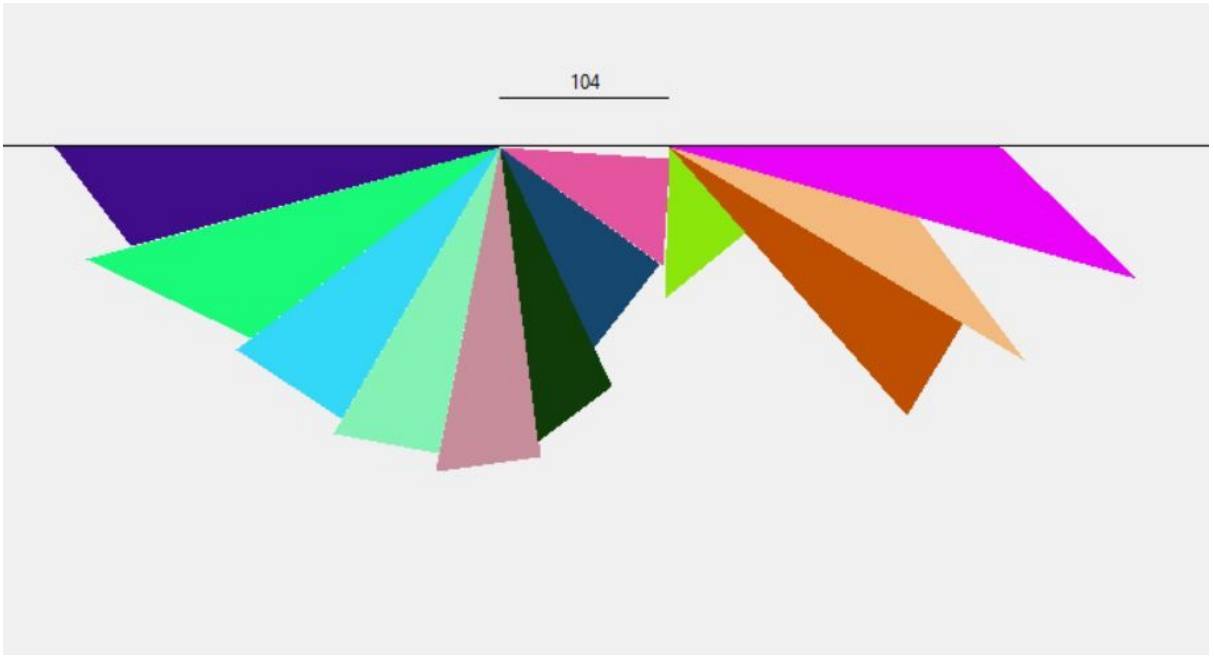
Visualisierung:



dreiecke3.txt

Ausgabe:
104, [['D4: (370|200), (100|200), (146|260)', 'D2: (370|200), (119|268), (219|316)', 'D6: (370|200), (210|323), (274|365)', 'D5: (370|200), (269|374), (333|386)', 'D1: (370|200), (331|397), (395|388)', 'D3: (370|200), (393|379), (438|345)', 'D7: (370|200), (427|322), (467|271)', 'D10: (370|200), (469|272), (473|207)'], ['D8: (473|200), (470|292), (519|252)', 'D11: (473|200), (617|363), (651|307)', 'D9: (473|200), (689|330), (623|242)', 'D12: (473|200), (757|280), (674|200)']]

Visualisierung:

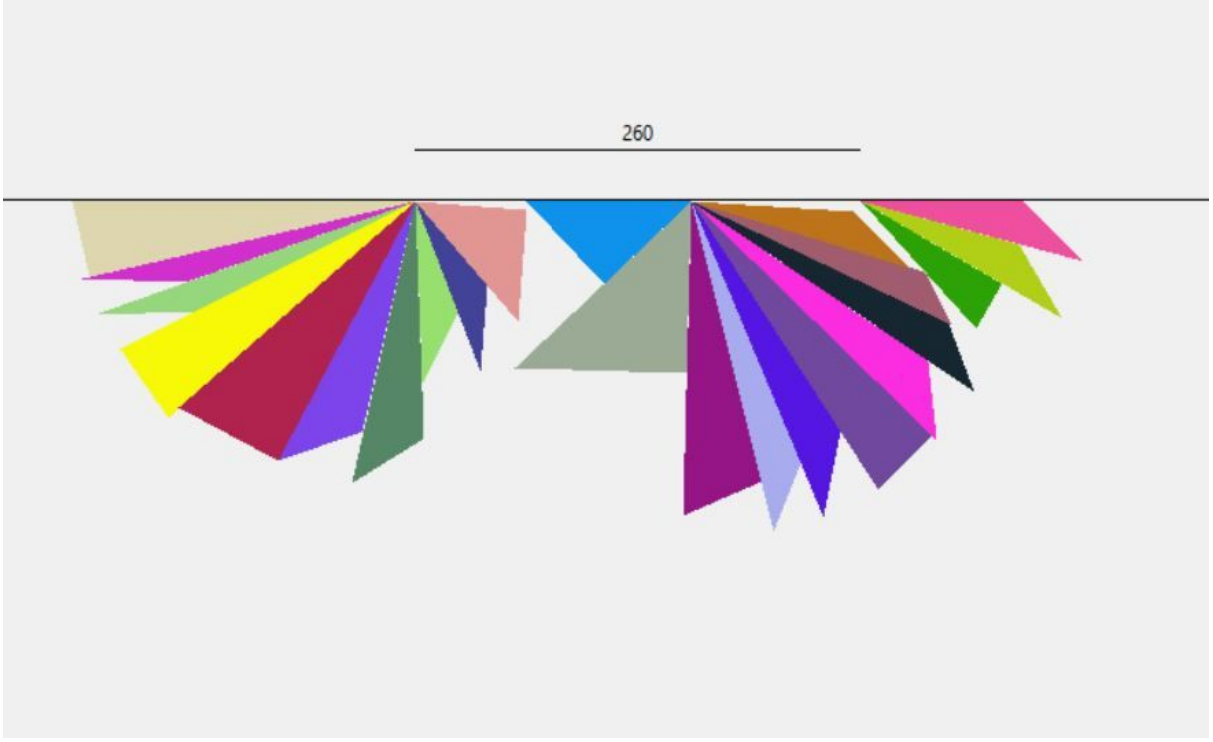


dreiecke4.txt

Ausgabe:

260, [['D12: (300|200), (100|200), (110|244)', 'D9: (300|200), (106|245), (166|247)', 'D11: (300|200), (113|266), (172|264)', 'D10: (300|200), (128|286), (156|326)', 'D8: (300|200), (162|320), (220|351)', 'D4: (300|200), (220|351), (269|334)', 'D3: (300|200), (263|364), (305|338)', 'D2: (300|200), (304|306), (325|265)', 'D1: (300|200), (338|299), (342|249)', 'D5: (300|200), (360|270), (365|205)'], ['D23: (461|200), (365|200), (410|248)', 'D16: (461|200), (358|297), (458|300)', 'D7: (461|200), (456|383), (502|363)', 'D14: (461|200), (509|392), (525|352)', 'D17: (461|200), (538|384), (548|334)', 'D13: (461|200), (570|368), (601|336)', 'D18: (461|200), (604|339), (599|293)', 'D15: (461|200), (626|311), (611|271)', 'D6: (461|200), (613|272), (597|241)', 'D19: (461|200), (589|239), (556|206)'], ['D20: (560|200), (627|274), (642|248)', 'D22: (560|200), (677|268), (652|225)', 'D21: (560|200), (690|235), (655|200)']]

Visualisierung:

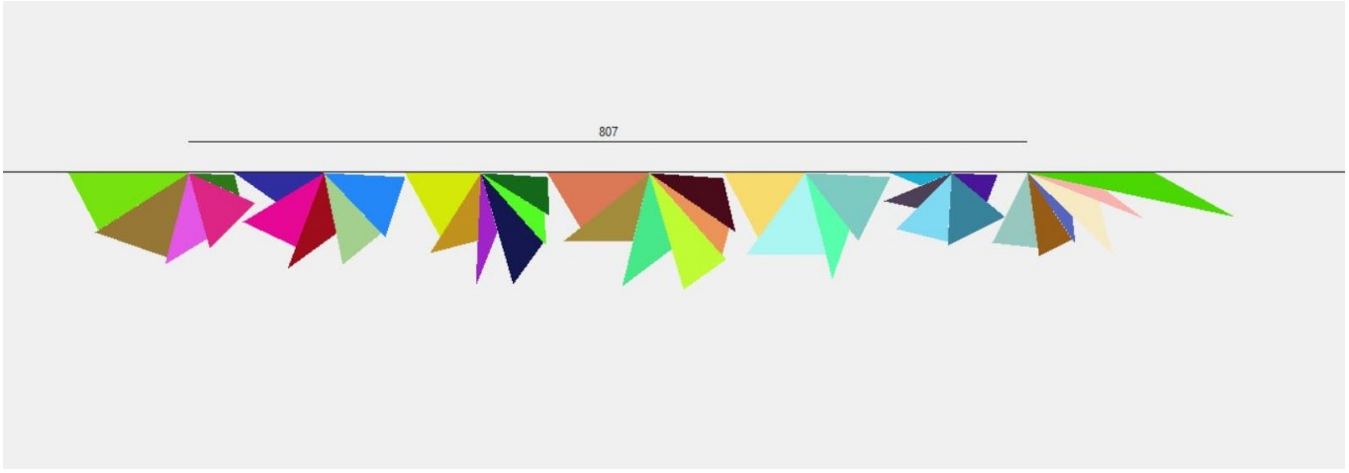


dreiecke5.txt

Ausgabe:

807, [['D4: (216|200), (100|200), (129|255)', 'D2: (216|200), (126|257), (195|281)', 'D5: (216|200), (193|288), (234|263)', 'D9: (216|200), (236|272), (280|229)', 'D8: (216|200), (265|222), (260|202)'], ['D11: (346|200), (260|200), (300|228)', 'D12: (346|200), (268|247), (319|271)', 'D6: (346|200), (311|292), (358|258)', 'D3: (346|200), (364|288), (401|258)', 'D13: (346|200), (405|262), (424|204)'], ['D14: (497|200), (424|200), (458|262)', 'D19: (497|200), (448|277), (494|265)', 'D1: (497|200), (492|309), (513|256)', 'D7: (497|200), (528|307), (557|267)', 'D17: (497|200), (560|270), (558|238)', 'D15: (497|200), (563|241), (561|204)'], ['D20: (659|200), (561|200), (593|253)', 'D23: (659|200), (577|265), (643|266)', 'D21: (659|200), (633|309), (681|273)', 'D24: (659|200), (692|312), (733|283)', 'D16: (659|200), (729|279), (736|253)', 'D18: (659|200), (742|257), (730|205)'], ['D32: (809|200), (731|200), (764|263)', 'D26: (809|200), (752|279), (829|279)', 'D10: (809|200), (835|302), (852|254)', 'D22: (809|200), (860|265), (891|204)'], ['D28: (950|200), (891|200), (920|212)', 'D30: (950|200), (885|227), (917|234)', 'D35: (950|200), (897|254), (946|266)', 'D33: (950|200), (946|270), (1001|242)', 'D27: (950|200), (984|228), (994|202)'], ['D36: (1023|200), (989|267), (1033|272)', 'D31: (1023|200), (1034|280), (1066|264)', 'D25: (1023|200), (1069|268), (1066|241)', 'D29: (1023|200), (1104|277), (1092|227)', 'D37: (1023|200), (1135|244), (1096|215)', 'D34: (1023|200), (1222|242), (1146|200)']]

Visualisierung:



100 Zufällige Dreiecke:

Ausgabe:

1277, [['D6: (100|200), (20|200), (51|250)', 'D1: (100|200), (49|251), (82|264)', 'D7: (100|200), (86|250), (120|237)', 'D3: (100|200), (119|236), (118|217)', 'D5: (100|200), (123|222), (129|201)'], ['D11: (154|200), (129|200), (139|216)', 'D2: (156|200), (118|240), (139|251)', 'D13: (156|200), (132|274), (159|265)', 'D14: (156|200), (162|310), (189|260)', 'D15: (156|200), (188|259), (190|232)', 'D4: (156|200), (196|237), (202|202)'], ['D16: (221|200), (202|200), (210|214)', 'D12: (225|200), (196|236), (236|243)', 'D22: (225|200), (244|278), (288|235)', 'D8: (225|200), (257|218), (259|202)'], ['D9: (290|200), (259|200), (266|218)', 'D24: (327|200), (283|232), (297|239)', 'D18: (327|200), (273|270), (325|266)', 'D23: (327|200), (324|306), (336|264)', 'D17: (327|200), (337|271), (361|261)', 'D10: (327|200), (343|230), (359|202)'], ['D20: (385|200), (359|200), (365|215)', 'D28: (390|200), (346|233), (368|246)', 'D27: (390|200), (360|264), (376|244)', 'D25: (390|200), (366|275), (422|253)', 'D19: (390|200), (420|250), (423|241)', 'D31: (390|200), (407|221), (415|201)'], ['D33: (468|200), (415|200), (435|238)', 'D26: (468|200), (419|257), (463|267)', 'D35: (468|200), (462|279), (516|246)', 'D32: (468|200), (510|241), (523|216)', 'D21: (468|200), (511|212), (505|201)'], ['D37: (522|200), (505|200), (511|206)', 'D40: (563|200), (520|224), (527|234)', 'D29: (565|200), (501|261), (539|250)', 'D39: (565|200), (522|282), (567|272)', 'D36: (565|200), (567|287), (625|251)', 'D38: (565|200), (591|222), (598|202)'], ['D44: (625|200), (598|200), (602|213)', 'D30: (662|200), (605|233), (623|245)', 'D41: (670|200), (616|262), (654|251)', 'D34: (670|200), (643|284), (660|250)', 'D42: (670|200), (656|271), (683|264)', 'D45: (670|200), (676|231), (700|211)', 'D43: (670|200), (694|209), (690|201)'], ['D51: (749|200), (695|200), (713|239)', 'D46: (749|200), (701|253), (744|267)', 'D47: (749|200), (743|281), (794|256)', 'D49: (749|200), (787|248), (803|223)', 'D52: (749|200), (795|220), (789|202)'], ['D59: (843|200), (791|200), (794|216)', 'D58: (845|200), (794|217), (821|220)', 'D50: (845|200), (800|238), (823|252)', 'D48: (845|200), (802|302), (868|270)', 'D57: (845|200), (862|252), (884|226)', 'D54: (845|200), (889|230), (896|215)', 'D55: (845|200), (884|211), (871|201)'], ['D66: (950|200), (884|200), (902|223)', 'D67: (951|200), (883|233), (899|245)', 'D53: (951|200), (874|267), (927|255)', 'D61: (951|200), (907|303), (966|277)', 'D62: (951|200), (969|296), (984|251)', 'D64: (951|200), (990|260), (979|231)', 'D65: (951|200), (988|240), (993|203)'], ['D70: (1010|200), (994|200), (1001|206)', 'D71: (1021|200), (991|219), (1004|227)', 'D69: (1021|200), (989|251), (1010|257)', 'D56: (1021|200), (1006|276), (1019|247)', 'D60: (1021|200), (1017|281), (1025|256)', 'D68: (1021|200), (1026|263), (1052|242)', 'D63: (1021|200), (1043|230), (1045|221)', 'D74: (1021|200), (1042|218), (1047|202)'], ['D80: (1111|200), (1047|200), (1058|218)', 'D76: (1113|200), (1045|223), (1094|265)', 'D75: (1113|200), (1092|272), (1145|250)', 'D78: (1113|200), (1151|259), (1171|230)', 'D79: (1113|200), (1162|225), (1167|202)'], ['D81: (1196|200), (1167|200), (1175|206)', 'D82: (1214|200), (1165|214), (1174|226)', 'D73: (1220|200), (1168|234), (1200|230)', 'D77: (1220|200), (1179|261), (1201|253)', 'D87: (1220|200), (1194|271), (1244|258)', 'D83: (1220|200), (1247|265), (1261|253)', 'D84: (1220|200), (1254|245), (1256|235)', 'D72: (1220|200), (1248|227), (1254|203)'], ['D92: (1284|200), (1254|200), (1260|216)', 'D85: (1284|200), (1257|218), (1271|218)', 'D93: (1284|200), (1263|229), (1291|232)', 'D88: (1284|200), (1298|264), (1308|253)', 'D89: (1284|200), (1298|232), (1304|222)', 'D91: (1284|200), (1305|223), (1307|213)', 'D86: (1284|200), (1302|211), (1299|201)'], ['D99: (1320|200), (1299|200), (1305|212)', 'D97: (1336|200), (1302|227), (1311|228)', 'D98: (1346|200), (1305|245), (1318|245)', 'D90:

(1346|200), (1311|255), (1359|261)', 'D95: (1346|200), (1358|259), (1367|230)', 'D96: (1346|200), (1367|231), (1377|203)'], ['D94: (1377|200), (1434|235), (1440|206)', 'D100: (1377|200), (1483|210), (1441|200)']]

Visualisierung:

Informationen zum Programm:

Ich habe die Programmiersprache **Python** (Version 3.7) verwendet. Zusätzlich habe ich die Bibliothek **sympy** verwendet, um geometrische Operationen durchzuführen z.B. Schnittpunkten von mehreren Dreiecken zu erkennen. Es kann sein, dass Sie diese Bibliothek installieren müssen, um das Programm zu starten. Wenn Sie pip haben: einfach **pip install sympy** eingeben.

Das Haupt-Programm befindet sich in der main.py Datei. Nach dem Starten des Programms werden Sie aufgefordert den Namen der Input-Datei einzugeben.

Quellen:

https://en.wikipedia.org/wiki/Knapsack_problem
https://en.wikipedia.org/wiki/Bin_packing_problem
https://en.wikipedia.org/wiki/Big_O_notation
https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/
https://wiki.python.org/moin/TkInter
https://stackoverflow.com/
https://en.wikipedia.org/wiki/Class_diagram
https://de.wikipedia.org/wiki/Programmablaufplan