

Dokumentation zur Aufgabe „Dreiecksbeziehung“

Die Lösungsidee und Umsetzung:

Bei dieser Aufgabe musste ich meine Lösungsidee mehrfach verändern. Im folgenden werde ich meine verschiedenen Lösungsideen erklären und verdeutlichen, warum ich mich letztlich für die letzte meiner Lösungsideen entschieden habe.

Lösungsidee 1:

Meine erste Idee war eigentlich recht simpel: Alle Dreiecke in allen möglichen Anordnungen in allen möglichen Drehungen (einschließlich gespiegelten Möglichkeiten) möglichst dicht aneinander schieben und die Anordnung mit der kürzesten Distanz auswählen und ausgeben. Das Problem dabei: Alleine alle verschiedenen möglichen Reihenfolgen, in denen die Dreiecke stehen können, bedeuten eine so große Anzahl an Möglichkeiten, dass selbst wenn die Mitglieder des Clubs der Trianguläre reich genug wären um sich Unsterblichkeit verschaffen zu können, sie wahrscheinlich nicht mehr miterleben würden, wie die Häuser stehen müssten. Für das letzte Beispiel auf der Website würde sich alleine für die verschiedenen Sortierungsmöglichkeiten eine Anzahl von $37! = 13.763.753.091.226.345.046.315.979.581.580.902.400.000.000$ Möglichkeiten ergeben. Und das ohne auch nur ein Dreieck dabei bisher gedreht zu haben.

Lösungsidee 2:

Also überlegte ich mir zunächst, wie ich schonmal das Drehen der Dreiecke umgehen kann. Das im Kreis Drehen aller Dreiecke würde nämlich nicht nur unglaublich viel Zeit kosten — genauso wie das Umsortieren — sondern auch keine sonderlich optimale Lösung hervorbringen, weil die Dreiecke stets kleine Lücken zwischen einander hätten, völlig egal wie kleinschrittig die Drehungen der Dreiecke vollzogen werden würden.

Meine Lösung: die Dreiecke immer direkt Kante an Kante aneinander legen.

So bleibt nur noch das Problem „ $37!$ “.

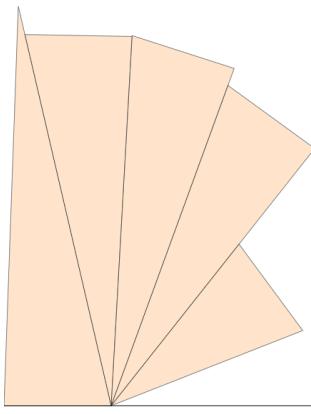
Lösungsidee 3:

Statt also alle Dreiecke in allen möglichen Reihenfolgen aneinander zu legen, überlegte ich mir, dass es am sinnvollsten wäre die Dreiecke nicht immer wieder umzusortieren und erneut aneinander zu schieben, sondern immer ein Dreieck zu nehmen und direkt eine möglichst gute Anordnung der übrigen Dreiecke an dieses anzulegen. Dabei soll immer um den am weitesten rechts liegenden Punkt, welcher zugleich auf dem Boden liegt, gedreht werden. Es sollen also Dreiecke im Kreis aneinander gelegt werden. Sobald dies aufgrund eines zu kleinen Winkels nicht mehr möglich ist, soll das nächste Dreieck einfach nur möglichst nah an die vorherigen angeschoben werden und alles wiederholt sich wieder, bis keine Dreiecke mehr übrig sind.

Beim letzten Beispiel bleiben folglich $37 (* 2$ mit gespiegelten Dreiecken) verschiedene Dreiecke für das erste Dreieck in Frage und dieses könnte beispielsweise in 1° Schritten gedreht werden. Alle anderen Dreiecke müssen nur möglichst gut aneinander geschoben werden nach den genannten Kriterien. Somit bleiben nur noch $37 * 360 = 13.320$ Gesamtmöglichkeiten für das letzte Beispiel (26.640 Möglichkeiten mit den gespiegelten Dreiecken zusammen).

Lösungsidee 4:

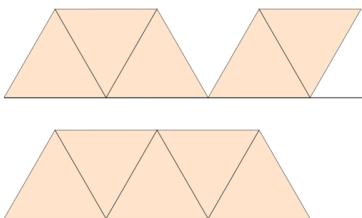
Statt das erste Dreieck in z.B. 1° Schritten zu drehen, wird in der Endvariante des Programs das jeweils erste Dreieck in jedem Durchlauf nun stattdessen mit den drei Seiten nach einander auf den Boden gelegt, da so bessere Lösungen in noch einmal deutlich schneller Zeit errechnet werden konnten (sowohl die dritte wie auch die vierte Lösungsidee sind jedoch zeitlich ausführbar).



Dieses Bild zeigt meine Vorstellung der letzten Lösungsidee. Es handelt sich um das zweite Beispiel. Die gefundene Lösung: 0 Meter.

Verbesserungen, die noch implementiert werden könnten:

Wenn kein Dreieck mehr im Kreis gelegt werden kann, könnte versucht werden, ganz kleine Dreiecke noch unter die vorherigen Dreiecke drunter zu schieben. Zudem könnte das nächste Dreieck statt auf den Boden gestellt und anschließend möglichst dicht an die anderen Dreiecke geschoben zu werden, möglichst weit mit einer Ecke in die vorherigen Dreiecke geschoben werden. Für die Umsetzung dieser Verbesserungen fehlt mir jedoch die nötige Zeit.



Auf diesem Weg könnte das oben zu sehende Problem bei dem ersten Beispiel behoben werden. Für nicht ähnliche Dreiecke, die sich nicht so anordnen lassen, wie es oben zu sehen ist funktioniert das Programm dafür aber ziemlich gut.

Quelltext und Umsetzung

Die Umsetzung der Lösungsidee erfolgt in meiner Bearbeitung in der Programmiersprache Ruby. Das Programm beziehungsweise Script („dreiecksbeziehungen.rb“) liegt im Ordner „Aufgabe 2\Programm“ und stellt zeitgleich Quelltext und ausführbare Datei dar. Ich habe das Script mit der Ruby Version 2.5.3-1-x64 in der CMD von Windows 10 getestet.

Die nachfolgenden Methoden sind teilweise nur dazu da die Lösungen einzelner Gleichungen zu erhalten. Aus diesem Grund findet sich bei diesen Methoden nur der Methoden Name statt die gesamte Methode.

Die Variablen Namen in dem Programm entsprechen denen, die bei uns im Mathematik Unterricht verwendet wurden:

x: Wert auf der x-Achse im Koordinatensystem
y: Wert auf der y-Achse im Koordinatensystem
m: Steigung einer Gerade
n: y-Achsenabschnitt einer Geraden

Es gilt also $y=m*x+n$.

def gleichsetzenVonGeraden(m1, n1, m2, n2)

Diese Methode findet den x Punkt an dem zwei Geraden sich schneiden. Sollten die zwei übergebenen Geraden die selbe Steigung haben wird entweder ∞ oder nil zurückgegeben, je nachdem ob die Geraden den selben y-Achsenabschnitt haben oder nicht.

Nimmt man jeweils die rechten Hälften von $y=m1*x+n1$ und $y=m2*x+n2$ und setzt diese gleich, sodass die Formel $m1*x+n1=m2*x+n2$ entsteht, kann man diese zu $x = (n1-n2)/(m2-m1)$ umformen. In diese werden letztlich die übergebenen Zahlen eingesetzt, wodurch der x Wert des Schnittpunktes der zwei Geraden bestimmt werden kann.

def xDurchYBestimmen(y, m, n)

Es wird der x Wert einer Geraden anhand der Werte von y, m und n berechnet. Dies geschieht durch die Formel $x=(y-n)/m$, welche aus $y=m*x+n$ entstanden ist.

def yDurchXBestimmen(x, m, n)

Es wird der y Wert einer Geraden anhand der Werte von x, m und n berechnet. Dies geschieht durch einsetzen der übergebenen Werte in die Formel $y=m*x+n$.

def xDurchWinkelBestimmen(w, r)

Es wird die Verschiebung auf der x-Achse eines neuen Punktes zu einem Startpunkt zurückgegeben, welche aus dem übergebenen Winkel und Radius (vom Startpunkt aus) folgt.

def yDurchWinkelBestimmen(w, r)

Es wird die Verschiebung auf der x-Achse eines neuen Punktes zu einem Startpunkt zurückgegeben, welche aus dem übergebenen Winkel und Radius (vom Startpunkt aus) folgt.

def steigungDurchZweiPunkteBestimmen(x1, y1, x2, y2)

Es werden zwei Punkte übergeben. Die Rückgabe ist die Steigung der Geraden, welche durch diese beiden Punkte verläuft. Sind die Werte x1 und x2 identisch ist die Steigung nicht bestimmbar und es wird nil zurückgegeben. Ansonsten ist die Rückgabe das Ergebnis des Einsetzens der Werte in die Formel $m=(y2-y1)/(x2-x1)$.

```
def steigungDurchWinkelBestimmen(w)
```

Es wird die Steigung m durch den Tangens des übergebenen Winkels berechnet.

```
def winkelDurchSteigungBestimmen(m)
```

Es wird durch den Arkustangens der Winkel bestimmt, der der übergebenen Steigung m entspricht.

```
def yAchsenabschnittBestimmen(x, y, m)
```

Diese Methode gibt den zu den übergebenen Werten gehörenden y-Achsenabschnitt der Gleichung zurück. Dies geschieht durch die aus der Formel $y=m*x+n$ entstandenen Formel $n=y-m*x$.

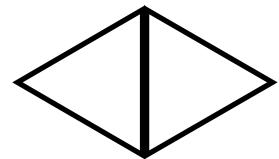
Die nachfolgende Methode erschafft die gespiegelte Variante des Dreiecks dessen drei Eckkoordinaten übergeben werden.

```
def spiegelDreieck(x1, y1, x2, y2, x3, y3)
```

```
    if x1 == x2
```

```
        newX3 = x1-x3+x1
```

```
        return x1, y1, x2, y2, newX3, y3
```

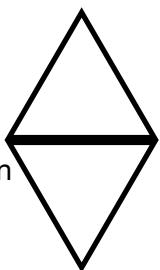


Wenn die Koordinaten vom ersten und zweiten übergebenen Eckpunkt auf dem selben x Wert liegen, wird die Distanz (auf x-Achse) der dritten Ecke zur ersten Ecke mit dem x Wert der ersten Ecke addiert. So wird entlang der Geraden zwischen dem ersten und zweiten Punkt gespiegelt.

```
    elsif y1 == y2
```

```
        newY3 = y1-y3+y1
```

```
        return x1, y1, x2, y2, x3, newY3
```



Wenn die Koordinaten vom ersten und zweiten übergebenen Eckpunkt auf dem selben y Wert liegen, wird die Distanz (auf y-Achse) der dritten Ecke zur ersten Ecke mit dem y Wert der ersten Ecke addiert. So wird entlang der Geraden zwischen dem ersten und zweiten Punkt gespiegelt.

```
    else
```

```
        oldM = steigungDurchZweiPunkteBestimmen(x1, y1, x2, y2)
```

```
        oldN = yAchsenabschnittBestimmen(x1, y1, oldM)
```

```
        m = steigungDurchWinkelBestimmen(winkelDurchSteigungBestimmen(oldM)+90)
```

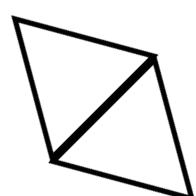
```
        n = yAchsenabschnittBestimmen(x3, y3, m)
```

```
        xSpiegel = gleichsetzenVonGeraden(oldM, oldN, m, n)
```

```
        ySpiegel = yDurchXBestimmen(xSpiegel, m, n)
```

```
        newX3 = xSpiegel - x3 + xSpiegel
```

```
        newY3 = ySpiegel - y3 + ySpiegel
```



```
        return x1, y1, x2, y2, newX3, newY3
```

```
    end
```

```
end
```

Ansonsten wird die Steigung zwischen dem ersten und zweiten Punkt berechnet, um 90° vergrößert und zusammen mit der Koordinate der dritten Ecke wird der y-Achsenabschnitt bestimmt. Anschließend wird bestimmt, auf welcher Koordinate die neue entstandene Gerade sich mit der Gerade zwischen den zwei ersten Eckpunkten schneidet. Zuletzt wird die Distanz (auf x- und y-Achse) der dritten Ecke zu dieser Koordinate mit der x und y Koordinate dieser Koordinate addiert. So wird entlang der Geraden zwischen dem ersten und zweiten Punkt gespiegelt.

```

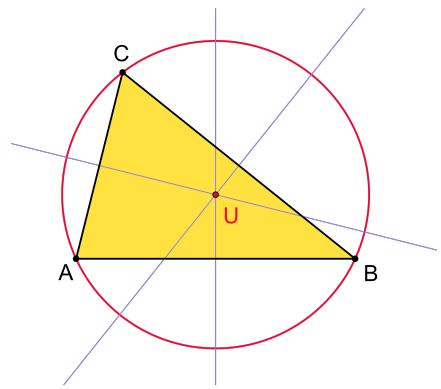
def mittelpunktVonUmkreisBestimmen(x1, y1, x2, y2, x3, y3)
    x = 0.0
    y = 0.0

    m1 = 0.0
    m2 = 0.0

    n1 = 0.0
    n2 = 0.0

    gotX = false
    gotY = false

```



Diese Methode dient dazu, den Umkreismittelpunkt eines Dreiecks zu bestimmen. Dazu werden zu zwei Seiten des übergebenen Dreiecks Mittelsenkrechten erschaffen. Der Schnittpunkt der zwei Mittelsenkrechten stellt den Mittelpunkt des Umkreises des Dreiecks dar. Wenn der Umkreismittelpunkt eines Dreiecks bekannt ist kann dieses an diesem Punkt gedreht werden. Es werden im Folgenden zu den Geraden des ersten und zweiten sowie des zweiten und dritten Punktes die Mittelsenkrechten aufgestellt, sofern dies notwendig sein sollte.

```

if x1 == x2
    gotY = true
    y = y1+(y2-y1)/2
elsif y1 == y2
    gotX = true
    x = x1+(x2-x1)/2

```

Wenn die ersten beiden Punkte genau übereinander oder nebeneinander liegen, steht zwangsweise eine Koordinate des Umkreismittelpunktes fest, da die zu bildende Mittelsenkrechte entweder mit 90° oder 0° verlaufen würde. Folglich ist entweder der y oder der x Wert an jeder Stelle der Mittelsenkrechten gleich und entspricht somit auch dem entsprechenden Wert des Umkreismittelpunktes.

```

else
    m1 =
steigungDurchWinkelBestimmen(winkelDurchSteigungBestimmen(steigungDurchZweiPunkteBesti
mmen(x1, y1, x2, y2))+90)
    n1 = yAchsenabschnittBestimmen(x1+(x2-x1)/2, y1+(y2-y1)/2, m1)
end

```

Konnte weder der x noch der y Wert, wie oben beschrieben, bestimmt werden, wird stattdessen die Steigung und der y-Achsenabschnitt der Mittelsenkrechten zwischen den ersten beiden Punkten berechnet.

```

if x2 == x3
    gotY = true
    y = y2+(y3-y2)/2
elsif y2 == y3
    gotX = true
    x = x2+(x3-x2)/2
else
    if gotX or gotY
        m1 =
steigungDurchWinkelBestimmen(winkelDurchSteigungBestimmen(steigungDurchZweiPunkteBesti
mmen(x2, y2, x3, y3))+90)
        n1 = yAchsenabschnittBestimmen(x2+(x3-x2)/2, y2+(y3-y2)/2, m1)

```

Anschließend wiederholt sich der Vorgang, wobei dieses Mal der zweite und dritte Punkt verwendet werden.

```

elsif not gotX and not gotY
    m2 =
steigungDurchWinkelBestimmen(winkelDurchSteigungBestimmen(steigungDurchZweiPunkteBesti
mmen(x2, y2, x3, y3)+90)
    n2 = yAchsenabschnittBestimmen(x2+(x3-x2)/2, y2+(y3-y2)/2, m2)
end
end

```

Nun können bereits beide Umkreismittelpunkt Werte (x und y), nur einer der beiden Werte und eine Mittelsenkrechte oder aber zwei Mittelsenkrechte bestimmt sein.

```

if gotX and gotY
    return x, y

```

Sind bereits beide Werte bestimmt, so werden diese zurückgegeben.

```

elsif gotX ^ gotY
    if gotX
        y = yDurchXBestimmen(x, m1, n1)
    else
        x = xDurchYBestimmen(y, m1, n1)
    end

    return x, y

```

Wenn nur ein Wert bisher bestimmt ist, so wird dieser in die Gleichung der Mittelsenkrechten eingesetzt und so der andere Wert bestimmt. Anschließend werden beide Werte zurückgegeben.

```

else
    x = gleichsetzenVonGeraden(m1, n1, m2, n2)
    y = yDurchXBestimmen(x, m1, n1)

    return x, y
end

```

Sollten nur die zwei Mittelsenkrechten bekannt sein, werden diese gleichgesetzt, um so den x Wert des Umkreismittelpunktes zu bestimmen. Dieser wird dann wiederum in die Gleichung der ersten Mittelsenkrechten eingesetzt, wodurch der y Wert des Umkreismittelpunktes bestimmt wird. Anschließend werden beide Werte zurückgegeben.

def winkelVonZweiPunktenBestimmen(x1, y1, x2, y2)

Mithilfe des Arkustangens wird der Winkel der Geraden, die durch die Punkte x1|y1 und x2|y2 verläuft, berechnet.

def distanzVonZweiPunktenBestimmen(x1, y1, x2, y2)

Dem Satz des Pythagoras folgend wird bei dieser Methode der Abstand der übergebenen Punkte x1|y1 und x2|y2 errechnet und zurückgegeben.

def innenwinkelBestimmen(a, b, c)

Mithilfe des Arkuskosinus werden aus den drei übergebenen Seitenlängen eines Dreiecks die entsprechenden Innenwinkel des Dreieckes errechnet und zurückgegeben.

def koordinatenUmsortieren(x1, y1, x2, y2, x3, y3)

Diese Methode sortiert die drei übergebenen Koordinaten entsprechend ihrer y Werte aufsteigend um und gibt die umsortierten Koordinaten wieder zurück.

```
def SVGPolygonAusKoordinaten(id, istGespiegelt, x1, y1, x2, y2, x3, y3)
```

Diese Methode erhält die Informationen eines Dreiecks und wandelt diese in SVG Code um, welcher als String zurückgegeben wird.

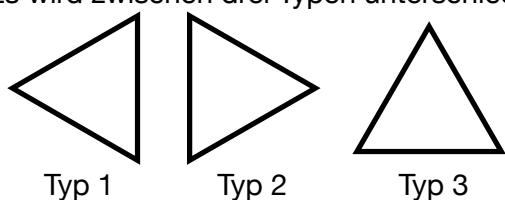
```
def typBestimmen(x1, y1, x2, y2, x3, y3)
    typ = 0
    x1, y1, x2, y2, x3, y3 = koordinatenUmsortieren(x1, y1, x2, y2, x3, y3)

    if y1 == y2
        typ = 3
    elsif y2 == y3
        typ = 1
    else
        if x1 == x3
            if x2 < x1
                typ = 1
            else
                typ = 2
            end
        else
            m = steigungDurchZweiPunkteBestimmen(x1, y1, x3, y3)
            n = yAchsenabschnittBestimmen(x1, y1, m)

            if y2 > yDurchXBestimmen(x2, m, n)
                if m > 0
                    typ = 1
                else
                    typ = 2
                end
            else
                if m > 0
                    typ = 2
                else
                    typ = 1
                end
            end
        end
    end
return typ
end
```

Diese Methode bestimmt den Typen eines Dreiecks, was beim aneinander legen später von Bedeutung ist, da man nur dann Dreiecke aneinander legen kann, wenn man weiß, an welche Seite des vorherigen Dreiecks das nächste Dreieck angelegt werden soll/muss.

Es wird zwischen drei Typen unterschieden:



Typ 1 bedeutet, dass die Ecke mit zweithöchstem y-Wert links von der Geraden zwischen der untersten und obersten Ecke liegt.

Typ 2 bedeutet, dass die Ecke mit zweithöchstem y-Wert rechts von der Geraden zwischen der untersten und obersten Ecke liegt.

Typ 3 bedeutet, dass das Dreieck mit zwei Ecken auf dem Boden steht.

```
def zweitenPunkteAufWinkelDrehen(x1, y1, r, w)
```

Es wird ein Punkt übergeben, von dem aus im Winkel w eine weiterer Punkt erschaffen wird mit einem Abstand von r zum übergebenen Punkt. Der neue Punkt wird zurückgegeben.

Die nachfolgende Methode dient dazu mithilfe der vorherigen Methode und der Rekonstruktion von Dreiecken durch ihre Innenwinkel, ein Dreieck perfekt an ein anderes anzulehnen. Dabei wird das vorherige Dreieck in Form von 3 Koordinaten, die Art, wie bzw. wo das nächste Dreieck angelehnt werden soll sowie das nächste Dreieck, wobei hier nur dessen Innenwinkel und Seitenlängen von Bedeutung sind, übergeben.

```
def dreieckeAneinanderLegen(x1, y1, x2, y2, x3, y3, num, dreieck)
```

```
a = dreieck[6]
```

```
b = dreieck[7]
```

```
c = dreieck[8]
```

```
alpha = dreieck[9]
```

```
beta = dreieck[10]
```

```
gamma = dreieck[11]
```

```
case num
```

```
when 0
```

```
w = winkelVonZweiPunktenBestimmen(x1, y1, x3, y3)
```

```
x4 = x1
```

```
y4 = y1
```

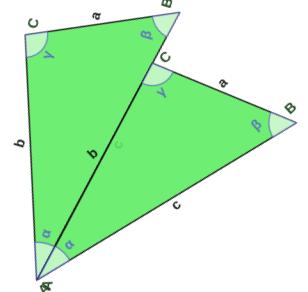
```
x5, y5 = zweitenPunkteAufWinkelDrehen(x4, y4, b, w)
```

```
winkelVonC = w - alpha
```

```
x6, y6 = zweitenPunkteAufWinkelDrehen(x4, y4, c, winkelVonC)
```

```
[...]
```

Wenn num 0, 1 oder 2 ist, wird davon ausgegangen, dass beim vorherigen Dreieck die mittlere Ecke links von der Geraden zwischen dem höchsten und tiefsten Punkt des Dreiecks liegt (siehe erstes Dreieck). Je nach dem, welche Zahl num ist, wird entweder die Seite a, b oder c des zweiten Dreiecks an das erste angelegt.



```
when 3
```

```
w = winkelVonZweiPunktenBestimmen(x1, y1, x2, y2)
```

```
x4 = x1
```

```
y4 = y1
```

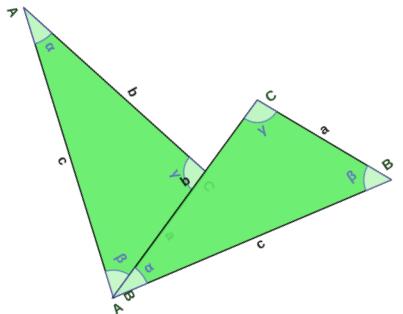
```
x5, y5 = zweitenPunkteAufWinkelDrehen(x4, y4, b, w)
```

```
winkelVonC = w - alpha
```

```
x6, y6 = zweitenPunkteAufWinkelDrehen(x4, y4, c, winkelVonC)
```

```
[...]
```

Wenn num 3, 4 oder 5 ist, wird davon ausgegangen, dass beim vorherigen Dreieck die mittlere Ecke rechts von der Geraden zwischen dem höchsten und tiefsten Punkt des Dreiecks liegt (siehe erstes Dreieck). Je nach dem, welche Zahl num ist, wird entweder die Seite a, b oder c des zweiten Dreiecks an das erste an der weiter unten liegenden Kante angelegt.



```
when 6
```

```
w = winkelVonZweiPunktenBestimmen(x2, y2, x3, y3)
```

```
x4 = x2
```

```
y4 = y2
```

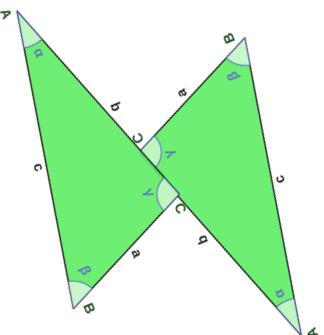
```
x5, y5 = zweitenPunkteAufWinkelDrehen(x4, y4, b, w)
```

```
winkelVonC = w - alpha
```

```
x6, y6 = zweitenPunkteAufWinkelDrehen(x4, y4, c, winkelVonC)
```

```
[...]
```

Wenn num 6, 7 oder 8 ist, wird davon ausgegangen, dass beim vorherigen Dreieck die mittlere Ecke rechts von der Geraden zwischen dem höchsten und tiefsten Punkt des Dreiecks liegt (siehe erstes Dreieck). Je nach dem, welche Zahl num ist, wird entweder die Seite a, b oder c des zweiten Dreiecks an das erste an der weiter oben liegenden Kante angelegt.



Die Fälle 6 bis 8 werden im Programm nicht verwendet, könnten aber für andere Algorithmen zum aneinander Schieben benutzt werden.

```

end

yVerschiebung = [y4, y5, y6].min
y4 -= yVerschiebung
y5 -= yVerschiebung
y6 -= yVerschiebung

m = steigungDurchWinkelBestimmen(w)
xVerschiebung = yVerschiebung/m
x4 -= xVerschiebung
x5 -= xVerschiebung
x6 -= xVerschiebung

return x4, y4, x5, y5, x6, y6
end

```

Zum Schluss wird das neu entstandene Dreieck bis auf $y = 0$ verschoben, wobei es zugleich auf der x-Achse entlang der verwendeten Kante zum anlehnen (vom ersten Dreieck) verschoben wird, sodass es noch immer möglichst optimal liegt.
(Da die verschiedenen Fälle für num alle recht ähnlich ablaufen habe ich es auf drei Beispiele beschränkt und nicht den kompletten Code eingefügt.)

Die nachfolgende Methode dient dazu, herauszufinden, um wie weit auf der x-Achse eine Linie zwischen zwei Punkten ($x3|y3$ und $x4|y4$) verschoben werden muss, damit diese die Gerade zwischen zwei anderen Punkten ($x1|y1$ und $x2|y2$) berührt.

```

def linienZusammenSchieben(x1, y1, x2, y2, x3, y3, x4, y4)
    m1 = steigungDurchZweiPunkteBestimmen(x1, y1, x2, y2)
    m2 = steigungDurchZweiPunkteBestimmen(x3, y3, x4, y4)

```

```

    if y2 < y3 or y1 > y4
        return Float::INFINITY
    end

```

Sollten die beiden übergebenen Linien nirgends auf der selben Höhe (y-Achse) liegen, so wird unendlich zurückgegeben. Es ist schließlich nicht möglich die zweite Linie an die erste zu schieben.

```

    if m1 == nil and m2 == nil
        return x3 - x1
    end

```

Wenn beide Linien gerade nach oben zeigen berechnet sich die zurückgegebene Verschiebung aus der x Koordinate der zweiten Linie minus der x Koordinate der ersten Linie.

```

    elsif m1 == nil and not m2 == nil
        n2 = yAchsenabschnittBestimmen(x3, y3, m2)
        if m2 > 0
            if y1 <= y3
                return x3 - x1
            else
                return xDurchYBestimmen(y1, m2, n2) - x1
            end
        else
            if y2 >= y4
                return x4 - x2
            else
                return xDurchYBestimmen(y2, m2, n2) - x1
            end
        end
    elsif not m1 == nil and m2 == nil
        n1 = yAchsenabschnittBestimmen(x1, y1, m1)

```

```

if m1 > 0
    if y2 <= y4
        return x4 - x2
    else
        return x4 - xDurchYBestimmen(y4, m1, n1)
    end
else
    if y1 >= y3
        return x3 - x1
    else
        return x3 - xDurchYBestimmen(y3, m1, n1)
    end
end
else
    n1 = yAchsenabschnittBestimmen(x1, y1, m1)
    n2 = yAchsenabschnittBestimmen(x3, y3, m2)

    if m1 == m2
        return x3 - xDurchYBestimmen(y3, m1, n1)
    elsif (m1 > 0 and m2 > 0) or (m1 < 0 and m2 < 0)
        if m1 > m2
            if y1 >= y3
                return xDurchYBestimmen(y1, m2, n2) - x1
            else
                return x3 - xDurchYBestimmen(y3, m1, n1)
            end
        else
            if y2 <= y4
                return xDurchYBestimmen(y2, m2, n2) - x2
            else
                return x4 - xDurchYBestimmen(y4, m1, n1)
            end
        end
    elsif m1 < 0 and m2 > 0
        if y1 >= y3
            return xDurchYBestimmen(y1, m2, n2) - x1
        else
            return x3 - xDurchYBestimmen(y3, m1, n1)
        end
    elsif m1 > 0 and m2 < 0
        if y2 <= y4
            return xDurchYBestimmen(y2, m2, n2) - x2
        else
            return x4 - xDurchYBestimmen(y4, m1, n1)
        end
    end
end

```

Um es etwas kürzer zu halten: es wird geguckt, wie weit die zweite Linie von der Position entfernt liegt, wo sie am nächsten an der ersten Linie auf der rechten Seite von dieser anliegen kann.

Um nun zwei Dreiecke statt nur zwei Linien möglichst dicht aneinander schieben zu können, müssen alle Linien des ersten Dreiecks mit je allen Linien des zweiten Dreiecks durch die vorherige Methode auf ihre Verschiebung überprüft werden. Der kleinste Wert stellt dann die Lösung dar, wie weit das zweite Dreieck auf der x-Achse verschoben werden muss, um das erste Dreieck zu berühren.

```
def verschiebungBestimmen(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6)
    x1, y1, x2, y2, x3, y3 = koordinatenUmsortieren(x1, y1, x2, y2, x3, y3)
    x4, y4, x5, y5, x6, y6 = koordinatenUmsortieren(x4, y4, x5, y5, x6, y6)
    verschiebung = [linienZusammenSchieben(x1, y1, x2, y2, x4, y4, x5, y5),
linienZusammenSchieben(x1, y1, x2, y2, x5, y5, x6, y6), linienZusammenSchieben(x1, y1, x2, y2, y4, x6, y6), linienZusammenSchieben(x2, y2, x3, y3, x4, y4, x5, y5),
linienZusammenSchieben(x2, y2, x3, y3, x5, y5, x6, y6), linienZusammenSchieben(x2, y2, x3, y3, x4, y4, x6, y6), linienZusammenSchieben(x1, y1, x3, y3, x4, y4, x5, y5),
linienZusammenSchieben(x1, y1, x3, y3, x5, y5, x6, y6), linienZusammenSchieben(x1, y1, x3, y3, x4, y4, x6, y6)]
    verschiebung -= [nil]
    verschiebung.reject! &:nan?
    return verschiebung.min
end
```

```
def naechstesDreieckBestimmen(uebrigeDreiecke, dreiecke, derzeitigerWinkel)
    nochPassendeDreiecke = Array.new
    i = 0
    while i < uebrigeDreiecke.length
        minWinkel = dreiecke[uebrigeDreiecke[i]][9 + dreiecke[uebrigeDreiecke[i]][12]]
        maxKante = dreiecke[uebrigeDreiecke[i]][6 + dreiecke[uebrigeDreiecke[i]][13]]
```

```
        if minWinkel <= derzeitigerWinkel
            nochPassendeDreiecke.push([maxKante, uebrigeDreiecke[i]])
        end
        i += 1
    end
```

nochPassendeDreiecke.sort!

Es wird für jedes noch übrige Dreieck kontrolliert, ob der kleinste Winkel des jeweiligen Dreiecks <= dem derzeitigen Winkel (übergebenen Variable derzeitigerWinkel) ist. Alle Dreiecke mit noch passendem Winkel werden anschließend der Länge der längsten Kante entsprechend sortiert.

Sollte es kein Dreieck mit passendem Winkel mehr geben, wird nil zurückgegeben.

```
if nochPassendeDreiecke.length == 0
    return nil
end

if derzeitigerWinkel >= 45
    return nochPassendeDreiecke[-1][1]
else
    return nochPassendeDreiecke[0][1]
end
end
```

Sollte der derzeitige Winkel >= 45 sein, so wird das Dreieck mit längster Kante zurückgegeben, ansonsten das Dreieck mit der kleinsten längsten Kante. So wird sichergestellt, dass lange Dreiecke möglichst nach oben zeigen und nicht zur Seite, da dies den Gesamtabstand unnötig vergrößern würde.

```

def dreieckAufBodenStellen(num, dreieck)
    a = dreieck[6]
    b = dreieck[7]
    c = dreieck[8]
    alpha = dreieck[9]
    beta = dreieck[10]
    gamma = dreieck[11]

    case num
    when 0
        x1 = 0.0
        y1 = 0.0
        x2 = a
        y2 = 0.0
        x3, y3 = zweitenPunkteAufWinkelDrehen(x1, y1, c, beta)
    when 1
        x1 = 0.0
        y1 = 0.0
        x2 = b
        y2 = 0.0
        x3, y3 = zweitenPunkteAufWinkelDrehen(x1, y1, a, gamma)
    when 2
        x1 = 0.0
        y1 = 0.0
        x2 = c
        y2 = 0.0
        x3, y3 = zweitenPunkteAufWinkelDrehen(x1, y1, b, alpha)
    end
    return x1, y1, x2, y2, x3, y3
end

```

Diese Methode erhält ein Dreieck sowie die Information auf welche Seite dieses gelegt werden soll (num). Anschließend wird das übergebene Dreieck mithilfe eines entsprechenden Innenwinkels auf dem Boden aufliegend rekonstruiert.

```

def naechstesDreieckBestimmen2(x1, y1, x2, y2, x3, y3, uebrigeDreiecke, dreiecke,
spiegelDreiecke)
    maxKanten = Array.new
    i = 0
    while i < uebrigeDreiecke.length
        maxKanten.push([dreiecke[uebrigeDreiecke[i]][6 + dreiecke[uebrigeDreiecke[i]][12]],
uebrigeDreiecke[i]])
        i += 1
    end
    maxKanten.sort!
    dreieck = dreiecke[maxKanten[-1][1]]
    spiegelDreieck = spiegelDreiecke[maxKanten[-1][1]]

    x4, y4, x5, y5, x6, y6 = dreieckAufBodenStellen(dreieck[12], dreieck)
    spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6 =
dreieckAufBodenStellen(spiegelDreieck[12], spiegelDreieck)

    verschiebung = Array.new
    spiegelVerschiebung = Array.new
    i = 0
    while i < x1.length
        verschiebung.push(verschiebungBestimmen(x1[i], y1[i], x2[i], y2[i], x3[i], y3[i], x4, y4, x5,
y5, x6, y6))
        spiegelVerschiebung.push(verschiebungBestimmen(x1[i], y1[i], x2[i], y2[i], x3[i], y3[i],
spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6))
        i += 1
    end

    min = verschiebung.min
    x4 -= min
    x5 -= min
    x6 -= min
    spiegelMin = spiegelVerschiebung.min
    spiegelX4 -= spiegelMin
    spiegelX5 -= spiegelMin
    spiegelX6 -= spiegelMin

    if [x4, x5, x6].max > [spiegelX4, spiegelX5, spiegelX6].max
        return x4, y4, x5, y5, x6, y6, false, maxKanten[-1][1]
    else
        return spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6, true,
maxKanten[-1][1]
    end
end

```

Diese Methode funktioniert recht ähnlich wie die Methode naechstesDreieckBestimmen: Es wird in dieser Methode das Dreieck mit der längsten der längsten Kanten aller übriger Dreiecke ausgewählt. Dieses, sowie das dazugehörige gespiegelte Dreieck werden dann möglichst dicht an alle bereits gelegten Dreiecke verschoben. Das Dreieck mit dem höheren höchsten x-Wert wird anschließend zurückgegeben. Grundsätzlich hätte ich gedacht, dass es logischer wäre genau das andere Dreieck in diesem Fall zurückzugeben, aber eine Reihe von Versuchen hat mich eines besseren belehrt. Meine Vermutung für dieses Phänomen: auf diese Art haben die neuen Dreiecke in der Regel einen noch größeren übrigen Winkel und es können mehr Dreiecke anschließend im Kreis aneinander gelegt werden.

```

def dreieckeLegen(dreiecke, spiegelDreiecke)
    alleDreiecke = dreiecke + spiegelDreiecke
    l = dreiecke.length
    i = 0
    alleKombinationen = Array.new

```

```

        while i < alleDreiecke.length
            erstesDreieck = alleDreiecke[i]

```

Durch die erste Schleife wird jedes Dreieck einmal als erstes Dreieck verwendet (dies umschließt auch die gespiegelten Varianten der Dreiecke).

```

        num = 0
        while num < 3
            idListe = Array.new
            uebrigeDreiecke = Array.new(l) {|a| a}
            if i > l
                idListe.push([i.dup-l, true])
                uebrigeDreiecke.delete_at(i-l)
            else
                idListe.push([i.dup, false])
                uebrigeDreiecke.delete_at(i)
            end

```

In der zweiten Schleife wird das jeweilige Dreieck nacheinander mit jeder der drei Seiten auf den Boden gestellt. Das derzeitige Dreieck wird zudem aus der Liste der übrigen Dreiecke entfernt.

```

        x1 = Array.new
        y1 = Array.new
        x2 = Array.new
        y2 = Array.new
        x3 = Array.new
        y3 = Array.new
        erstesX1, erstesY1, erstesX2, erstesY2, erstesX3, erstesY3 =
        dreieckAufBodenStellen(num, erstesDreieck)

```

$x1[0], y1[0], x2[0], y2[0], x3[0], y3[0] = \text{koordinatenUmsortieren}(erstesX1, erstesY1, erstesX2, erstesY2, erstesX3, erstesY3)$

Des weiteren wird für jeden x- und jeden y-Wert, den ein Dreieck haben kann (also drei x- und drei y-Werte) ein Array erschaffen. An die erste Stelle in diesen sechs Arrays wird das derzeitige Dreieck eingefügt.

Die nachfolgende Schleife läuft solange, bis keine Dreiecke mehr übrig sind.

```

        while uebrigeDreiecke.length > 0
            if y1[-1] == y2[-1]
                dw = winkelVonZweiPunktenBestimmen(x2[-1], y2[-1], x3[-1], y3[-1])
            else
                dw = [winkelVonZweiPunktenBestimmen(x1[-1], y1[-1], x2[-1], y2[-1]),
                winkelVonZweiPunktenBestimmen(x1[-1], y1[-1], x3[-1], y3[-1])].min
            end

```

Als erstes wird der derzeitige Winkel bestimmt, in welchem das nächste Dreieck an das vorherige angelegt werden soll.

```

            naechstesDreieck = naechstesDreieckBestimmen(uebrigeDreiecke, dreiecke,
dw)

```

Anschließend wird mit dieser Information das nächste Dreieck bestimmt.

```

if naechstesDreieck == nil
    x4, y4, x5, y5, x6, y6, istGespiegelt, id =
naechstesDreieckBestimmen2(x1, y1, x2, y2, x3, y3, uebrigeDreiecke, dreiecke, spiegelDreiecke)
    idListe.push([id, istGespiegelt])
    uebrigeDreiecke.delete_at(uebrigeDreiecke.index(id))

```

Sollte dabei nil zurückgegeben werden kann kein Dreieck mehr im Kreis angelegt werden. In diesem Fall wird das nächste Dreieck stattdessen durch die Methode naechstesDreieckBestimmen2 bestimmt. Das dabei bestimmte Dreieck wird aus der Liste der übrigen Dreiecke entfernt.

```

else
    dreieck = dreiecke[naechstesDreieck]
    spiegelDreieck = spiegelDreiecke[naechstesDreieck]

    derzeitigerTyp = typBestimmen(x1[-1], y1[-1], x2[-1], y2[-1], x3[-1], y3[-1])
    case derzeitigerTyp
        when 1
            x4, y4, x5, y5, x6, y6 = dreieckeAneinanderLegen(x1[-1], y1[-1],
x2[-1], y2[-1], x3[-1], y3[-1], dreieck[12], dreieck)
            spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6 =
dreieckeAneinanderLegen(x1[-1], y1[-1], x2[-1], y2[-1], x3[-1], y3[-1], spiegelDreieck[12],
spiegelDreieck)
        when 2
            x4, y4, x5, y5, x6, y6 = dreieckeAneinanderLegen(x1[-1], y1[-1],
x2[-1], y2[-1], x3[-1], y3[-1], dreieck[12] + 3, dreieck)
            spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6 =
dreieckeAneinanderLegen(x1[-1], y1[-1], x2[-1], y2[-1], x3[-1], y3[-1], spiegelDreieck[12] + 3,
spiegelDreieck)
        when 3
            x4, y4, x5, y5, x6, y6 = dreieckeAneinanderLegen(x2[-1], y2[-1],
x1[-1], y1[-1], x3[-1], y3[-1], dreieck[12], dreieck)
            spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6 =
dreieckeAneinanderLegen(x2[-1], y2[-1], x1[-1], y1[-1], x3[-1], y3[-1], spiegelDreieck[12],
spiegelDreieck)
    end

```

Abhängig vom Typen (siehe typBestimmen) des vorherigen Dreiecks wird das derzeitig ausgewählte Dreiecke sowie dessen gespiegelte Version an das vorherige Dreieck angelegt.

```

verschiebung = Array.new
spiegelVerschiebung = Array.new
z = 0
while z < x1.length
    verschiebung.push(verschiebungBestimmen(x1[z], y1[z], x2[z], y2[z],
x3[z], y3[z], x4, y4, x5, y5, x6, y6))
    spiegelVerschiebung.push(verschiebungBestimmen(x1[z], y1[z],
x2[z], y2[z], x3[z], y3[z], spiegelX4, spiegelY4, spiegelX5, spiegelY5, spiegelX6, spiegelY6))
    z += 1
end
min = verschiebung.min
spiegelMin = spiegelVerschiebung.min
x4 -= min
x5 -= min
x6 -= min
spiegelMin = spiegelVerschiebung.min
spiegelX4 -= spiegelMin
spiegelX5 -= spiegelMin
spiegelX6 -= spiegelMin

```

Sowohl das ursprüngliche wie auch das gespiegelte Dreieck werden anschließend soweit verschoben, dass sie in kein vorheriges Dreieck schneiden, sondern diese höchstens berühren.

```

if [x4, x5, x6].max > [spiegelX4, spiegelX5, spiegelX6].max
    x4, y4, x5, y5, x6, y6 = spiegelX4, spiegelY4, spiegelX5, spiegelY5,
spiegelX6, spiegelY6
    idListe.push([naechstesDreieck, true])
else
    idListe.push([naechstesDreieck, false])
end

```

uebrigeDreiecke.delete_at(uebrigeDreiecke.index(naechstesDreieck))
Anschließend wird das Dreieck zwischen dem ursprünglichen und dem gespiegelten ausgewählt, welches weniger weit nach rechts reicht. Dieses zeigt schließlich stattdessen zwangsweise weiter nach oben und ist somit als vorteilhafter zu betrachten.

```

end

x1.push(x4)
y1.push(y4)
x2.push(x5)
y2.push(y5)
x3.push(x6)
y3.push(y6)
end

```

Zuletzt wird das neue Dreieck in die sechs Arrays vom Anfang eingefügt und das ganze wiederholt sich bis keine Dreiecke mehr übrig sind.

```

distanz = x1[-1] - x2[0]
alleKombinationen.push([distanz, x1, y1, x2, y2, x3, y3, idListe])

```

Ist dies der Fall werden die Dreiecke sowie die berechnete Distanz in den Array alleKombinationen gespeichert.

```

num += 1
end

i += 1
end

```

All dies wiederholt sich solange, bis jedes Dreieck — einschließlich der gespiegelten Varianten — einmal mit allen drei Kanten als erstes Dreieck auf dem Boden gelegen hat.

```

return alleKombinationen.min
end

```

Zuletzt wird die Dreieck Aufstellung zurückgegeben, die den kleinsten berechneten Gesamtabstand besitzt.

```

def dreieckeEinlesen(datei)
    dreiecke = Array.new
    spiegelDreiecke = Array.new

    liste = File.open(datei).read.force_encoding("UTF-8").split("\n")

    i = 1
    while i < liste.length
        zeile = liste[i].split(" ")
        x1 = zeile[1].to_f
        y1 = zeile[2].to_f
        x2 = zeile[3].to_f
        y2 = zeile[4].to_f
        x3 = zeile[5].to_f
        y3 = zeile[6].to_f

```

Als erstes wird jeder x- und jeder y-Wert aus der übergebenen Datei ausgelesen.

spiegelX1, spiegelY1, spiegelX2, spiegelY2, spiegelX3, spiegelY3 = spiegelDreieck(x1, y1, x2, y2, x3, y3)

Anschließend werden mit diesen drei Koordinaten die drei entsprechenden Koordinaten der gespiegelten Variante des Dreiecks bestimmt.

xMitte, yMitte = mittelpunktVonUmkreisBestimmen(x1, y1, x2, y2, x3, y3)

spiegelXMitte, spiegelYMitte = mittelpunktVonUmkreisBestimmen(spiegelX1, spiegelY1, spiegelX2, spiegelY2, spiegelX3, spiegelY3)

Darauf folgend werden die Umkreismittelpunkte des ursprünglichen und des gespiegelten Dreiecks bestimmt.

w1 = winkelVonZweiPunktenBestimmen(xMitte, yMitte, x1, y1)
w2 = winkelVonZweiPunktenBestimmen(xMitte, yMitte, x2, y2)
w3 = winkelVonZweiPunktenBestimmen(xMitte, yMitte, x3, y3)

spiegelW1 = winkelVonZweiPunktenBestimmen(spiegelXMitte, spiegelYMitte, spiegelX1, spiegelY1)

spiegelW2 = winkelVonZweiPunktenBestimmen(spiegelXMitte, spiegelYMitte, spiegelX2, spiegelY2)

spiegelW3 = winkelVonZweiPunktenBestimmen(spiegelXMitte, spiegelYMitte, spiegelX3, spiegelY3)

Von den entsprechenden Umkreismittelpunkten werden anschließend die Winkel zu den jeweiligen Ecken der Dreiecke bestimmt.

r = distanzVonZweiPunktenBestimmen(xMitte, yMitte, x1, y1)

Zudem wird der Radius der Umkreise bestimmt.

a = distanzVonZweiPunktenBestimmen(x2, y2, x3, y3)
b = distanzVonZweiPunktenBestimmen(x3, y3, x1, y1)
c = distanzVonZweiPunktenBestimmen(x1, y1, x2, y2)
spiegelA = distanzVonZweiPunktenBestimmen(spiegelX1, spiegelY1, spiegelX3, spiegelY3)
spiegelB = distanzVonZweiPunktenBestimmen(spiegelX3, spiegelY3, spiegelX2, spiegelY2)
spiegelC = distanzVonZweiPunktenBestimmen(spiegelX2, spiegelY2, spiegelX1, spiegelY1)

Des Weiteren werden die drei Seitenlängen beider Dreiecke bestimmt.

```
alpha, beta, gamma = innenwinkelBestimmen(a, b, c)
spiegelAlpha, spiegelBeta, spiegelGamma = innenwinkelBestimmen(spiegelA, spiegelB,
spiegelC)
Anhand der drei Seitenlängen werden dann die drei jeweils den Seiten gegenüberliegenden
Innenwinkel der Dreiecke bestimmt.
```

```
min = [alpha, beta, gamma].index([alpha, beta, gamma].min)
max = [alpha, beta, gamma].index([alpha, beta, gamma].max)
spiegelMin = [spiegelAlpha, spiegelBeta, spiegelGamma].index([spiegelAlpha,
spiegelBeta, spiegelGamma].min)
spiegelMax = [spiegelAlpha, spiegelBeta, spiegelGamma].index([spiegelAlpha,
spiegelBeta, spiegelGamma].max)
Zuletzt wird noch bestimmt, welche Innenwinkel jeweils die größten und welche die kleinste der
jeweiligen Dreiecke sind.
```

```
dreiecke.push([xMitte, yMitte, w1, w2, w3, r, a, b, c, alpha, beta, gamma, min, max])
```

```
spiegelDreiecke.push([spiegelXMitte, spiegelYMitte, spiegelW1, spiegelW2, spiegelW3,
r, spiegelA, spiegelB, spiegelC, spiegelAlpha, spiegelBeta, spiegelGamma, spiegelMin,
spiegelMax])
```

Alle entstandenen Informationen werden als Array gespeichert. Sowohl der jeweilige Umkreismittelpunkt, sowie der Radius des jeweiligen Umkreises und die bestimmten Winkel zu den Ecken werden der letzten Lösungsidee folgend nicht verwendet. Ich habe sie jedoch nicht entfernt, da diese Informationen für Verbesserungen des Algorithmus später noch nützlich sein könnten und die Geschwindigkeit des Programms nicht bemerkbar beeinträchtigen.

```
i += 1
```

```
end
```

Der gesamte Prozess wiederholt sich solange, bis keine Dreiecke mehr in der eingelesenen Datei übrig sind.

```
return dreiecke, spiegelDreiecke
```

```
end
```

Zuletzt werden alle ursprünglichen und gespiegelten Dreiecke zurückgegeben.

```

puts "Gebe den Namen der einzulesenen .txt Datei ein:"
dreiecke, spiegelDreiecke = dreieckeEinlesen(gets.chomp)
t = Time.now
ergebnis = dreieckeLegen(dreiecke, spiegelDreiecke)
svg = ""
puts "Es wurde der folgende minimale Gesamtabstand berechnet: #{ergebnis[0]} Meter\nEs
handelt sich bei dieser Lösung nicht zwangsweise um die optimalste Lösung, sondern
ausschließlich um eine Annäherung an diese."
x1 = ergebnis[1].min
x2 = ergebnis[3].min
x3 = ergebnis[5].min
verschiebung = [x1, x2, x3].min
i = 0
puts "ID |ist Gespiegelt | x1|y1 | x2|y2 | x3|y3\n"
while i < ergebnis[1].length
    svg += SVGPolygonAusKoordinaten(ergebnis[7][i][0], ergebnis[7][i][1], ergebnis[1][i] -
verschiebung, ergebnis[2][i], ergebnis[3][i] - verschiebung, ergebnis[4][i], ergebnis[5][i] -
verschiebung, ergebnis[6][i])
    puts "D#{ergebnis[7][i][0]+1} |#{ergebnis[7][i][1]} | #{ergebnis[1][i] - verschiebung}
#{ergebnis[2][i]} | #{ergebnis[3][i] - verschiebung}|#{ergebnis[4][i]} | #{ergebnis[5][i] - verschiebung}
#{ergebnis[6][i]}"
    i += 1
end

newSVGName = "#{Time.now.to_s[0..-7].gsub!(":", "-")}.svg"
newSVG = File.new(newSVGName, "w+")
newSVG.write("<svg version='1.1' viewBox='0 0 1800 620' xmlns='http://www.w3.org/2000/
svg'><g transform='scale(1 -1)'><g transform='translate(0 -600)'><line xmlns='http://
www.w3.org/2000/svg' id='y' x1='0' x2='1800' y1='0' y2='0' fill='none' stroke='#000000' stroke-
width='1' />#{svg}</g></g></svg>")
newSVG.close
puts "Die Lösung wurde in folgender Datei visuell gespeichert: #{newSVGName}"
t2 = Time.now
puts "Die Lösung wurde in folgender Zeit berechnet:\nStartzeit: #{t}\nEndzeit: #{t2}\nDifferenz:
#{t2-t}"

```

Zum Schluss wird dann alles nur noch gestartet und die entsprechenden Informationen, die in der Aufgabenstellung gefordert wurden, ausgegeben.

Methoden, die ich bereits programmiert habe, aber letztlich nicht verwendet wurden, da sich die Lösungsidee geändert hat:

Die hier aufgeführten Methoden befinden sich getrennt vom Rest in der ausführbaren Datei „alle_reihenfolgen.rb“

def fakultaet(int)

Diese Methode errechnet die Fakultät einer Zahl. Die Methode wird im Programm nicht benutzt, da sie zum Umsortieren der Dreiecke verwendet wurde, was aber wie in der Lösungsidee beschrieben ist zu zeitaufwändig war.

def naechsteZahl(benutzt, derzeitigeZahl, ende)

Diese Methode entscheidet anhand der bereits benutzten Zahlen, der derzeitigen Zahl und der höchstmöglichen Zahl, welche Zahl als nächstes beim Umsortieren dran ist. Wie auch die vorherige Methode wird diese Methode nicht benutzt, da die Dreiecke letztlich nicht mehr umsortiert werden.

def vertauschen(dreiecke)

```
i = 0
a = Array.new(dreiecke.length)
c = Array.new(a.length)
while i < a.length
    a[i] = i
    c[c.length - i - 1] = fakultaet(i + 1) / (i + 1)
    i += 1
end
```

```
zaehler = Array.new(a.length, 0)
z = 0
while z < fakultaet(a.length)
    neueDreiecke = Array.new
    q = 0
    while q < a.length
        neueDreiecke.push(dreiecke[a[q]])
        q += 1
    end
```

In dieser Lücke wäre der jeweilige Array mit allen Dreiecken in seiner derzeitigen Reihenfolge an eine Methode, die alle Dreiecke, die diese erhält, dreht, übergeben worden. Nach jedem kompletten Drehen (jedes Dreieck in jeder Drehkombination -> 360° Anzahl an Dreiecken Möglichkeiten) wäre die Gesamtlänge der Dreiecksaufstellung gespeichert worden und am Ende wäre die beste ausgegeben worden. Die Methode zum Drehen habe ich jedoch nicht programmiert, da zuvor bereits klar wurde, dass es zeitlich, durch willkürliches Drehen und Vertauschen der Dreiecke, unmöglich ist die Aufgabe zu lösen.

```
i = 0
b = Array.new
while i < a.length
    if zaehler[i] == c[i] - 1
        a[i] = naechsteZahl(b, a[i], a.length - 1)
        zaehler[i] = 0
    else
        zaehler[i] += 1
    end
    b.push(a[i])
    i += 1
end
z += 1
end
end
```

Beispiele

```
C:\Windows\system32\cmd.exe
I:\BWINF.2\Dreiecksbeziehungen>dreiecksbeziehungen.rb
Geben den Namen der einzulesenen .txt Datei ein:
dreiecke1.txt
Es wurde der folgende minimale Gesamtabstand berechnet: 285.6326078599832 Meter
Es handelt sich bei dieser Lösung nicht zwangsweise um die optimalste Lösung, sondern ausschließlich um eine Annäherung an diese.
ID |ist Gespiegelt | x1|y1 | x2|y2 | x3|y3

D4 |false | 0.0|0.0 | 142.87407042567241|0.0 | 71.3985397742754|123.71034119305148
D5 |false | 142.87407042567241|0.0 | 71.39853977427542|123.71034119305146 | 214.23409401726204|123.77700526756891
D3 |true | 142.87407042567241|0.0 | 285.7480786077912|0.1333640724572798 | 214.23409401726204|123.77700526756891
D2 |true | 285.671108485973|0.0 | 428.5066782856556|0.0 | 357.08889338581434|123.74368670764584
D1 |false | 428.5066782856556|0.0 | 499.9244631854969|123.74368670764581 | 357.08889338581434|123.74368670764584
Die Lösung wurde in folgender Datei visuell gespeichert: 2019-04-27 15-01-55.svg
Die Lösung wurde in folgender Zeit berechnet:
Startzeit: 2019-04-27 15:01:55 +0200
Endzeit: 2019-04-27 15:01:56 +0200
Differenz: 0.725089

I:\BWINF.2\Dreiecksbeziehungen>dreiecksbeziehungen.rb
Geben den Namen der einzulesenen .txt Datei ein:
dreiecke2.txt
Es wurde der folgende minimale Gesamtabstand berechnet: 0.0 Meter
Es handelt sich bei dieser Lösung nicht zwangsweise um die optimalste Lösung, sondern ausschließlich um eine Annäherung an diese.
ID |ist Gespiegelt | x1|y1 | x2|y2 | x3|y3

D2 |true | 0.0|0.0 | 149.99999999999991|0.0 | 20.00000000000345|558.0
D4 |false | 149.99999999999991|0.0 | 179.23776823496098|515.559068302206 | 29.261824729861885|518.2453984672102
D5 |false | 149.99999999999991|0.0 | 322.13161414222714|470.92643524524226 | 179.30594339665888|516.7612230824134
D1 |true | 149.99999999999991|0.0 | 434.91252163713455|359.15018448327345 | 313.443785592318|447.1578341426886
D3 |false | 149.99999999999991|0.0 | 418.1258666382091|104.964373191618 | 328.95062195101804|225.5785337822369
Die Lösung wurde in folgender Datei visuell gespeichert: 2019-04-27 15-02-13.svg
Die Lösung wurde in folgender Zeit berechnet:
Startzeit: 2019-04-27 15:02:13 +0200
Endzeit: 2019-04-27 15:02:14 +0200
Differenz: 0.967293

I:\BWINF.2\Dreiecksbeziehungen>dreiecksbeziehungen.rb
Geben den Namen der einzulesenen .txt Datei ein:
dreiecke3.txt
Es wurde der folgende minimale Gesamtabstand berechnet: 204.8567159776318 Meter
Es handelt sich bei dieser Lösung nicht zwangsweise um die optimalste Lösung, sondern ausschließlich um eine Annäherung an diese.
ID |ist Gespiegelt | x1|y1 | x2|y2 | x3|y3

D7 |false | 123.97999337440271|0.0 | 258.2185869308014|0.0 | 153.0625388374538|57.56913712562979
D12 |false | 258.2185869308014|0.0 | 114.95990104610286|140.2602898848754 | 0.0|141.36534710975022
D4 |false | 258.2185869308014|0.0 | 65.44898369616078|188.7349466017336 | 140.28773554057346|200.0557779479912
D2 |false | 258.2185869308014|0.0 | 161.6988450910651|163.73435630676494 | 222.1637493072834|257.18485313863596
D9 |false | 258.2185869308014|0.0 | 278.12747215696083|153.83314431243875 | 223.2680142150224|249.3079570868909
D11 |false | 258.2185869308014|0.0 | 344.9863554093132|187.93976256572043 | 286.1844510832768|215.4701338479876
D6 |true | 258.21858693080145|0.0 | 398.07530290843306|129.81948619507074 | 342.5631132422666|182.69099835868917
D10 |true | 258.2185869308015|0.0 | 359.9415353762878|19.50491629197413 | 347.997240759102|83.33542512242703
D5 |false | 398.07530290843306|0.0 | 463.07530290843306|0.0 | 398.07530290843306|190.0
D1 |false | 463.07530290843306|0.0 | 398.1793539520571|189.69585079556023 | 462.1786412592353|189.9978841931848
D3 |false | 463.0753029084332|0.0 | 510.9382097164216|152.6765933333915 | 462.2216493762761|180.88469055076783
D8 |true | 463.0753029084332|0.0 | 523.5260312055138|33.74476919986945 | 490.72083586389044|88.18573868607469
Die Lösung wurde in folgender Datei visuell gespeichert: 2019-04-27 15-02-22.svg
Die Lösung wurde in folgender Zeit berechnet:
Startzeit: 2019-04-27 15:02:22 +0200
Endzeit: 2019-04-27 15:02:23 +0200
Differenz: 1.100612
```



Gefundene Lösung für Beispiel 5.

```
C:\Windows\system32\cmd.exe
I:\BWINF.2\Dreiecksbeziehungen>dreiecksbeziehungen.rb
Geben den Namen der einzulesenen .txt Datei ein:
dreiecke4.txt
Es wurde der folgende minimale Gesamtabstand berechnet: 296.5277924636313 Meter
Es handelt sich bei dieser Lösung nicht zwangsweise um die optimalste Lösung, sondern ausschließlich um eine Annäherung an diese.
ID | ist Gespiegelt | x1|y1 | x2|y2 | x3|y3

D19 | true | 61.58335935873686|0.0 | 195.00000000000002|0.0 | 102.4328364090115|21.361653136381577
D13 | false | 195.00000000000002|0.0 | 0.0|44.9999999999915 | 19.719101123595323|85.44943820224643
D12 | false | 195.00000000000002|0.0 | 19.71910112359533|85.44943820224643 | 39.438202247190596|125.89887640449363
D18 | false | 195.00000000000002|0.0 | 65.96432997239529|104.43081853804973 | 64.66317664711596|150.5101740025539
D17 | false | 195.00000000000002|0.0 | 90.25945876509643|120.95213525117869 | 93.1511165734385|171.07835907782916
D9 | false | 195.00000000000002|0.0 | 121.9922733926272|122.63307811447548 | 113.03841474827146|181.25754754774655
D11 | true | 195.00000000000002|0.0 | 136.1967670907885|130.04299211962572 | 135.8501169700769|189.581885573349
D15 | false | 195.00000000000002|0.0 | 145.55828520540686|158.46613782814956 | 165.26149418737492|196.03984613346455
D14 | true | 195.00000000000002|0.0 | 195.22011294079647|164.99985318264163 | 165.26149418737492|196.03984613346455
D10 | true | 195.00000000000002|0.0 | 243.6855138632419|184.74230901412946 | 195.25639878084405|192.20024521229203
D8 | false | 195.00000000000002|0.0 | 294.4853969795071|139.0814717632436 | 241.61817087067178|176.89755833440117
D7 | true | 195.00000000000002|0.0 | 326.10081361603375|105.05987183132297 | 301.43025017479042|148.79045461410692
D20 | false | 195.00000000000002|0.0 | 283.5815687665346|34.32645735958983 | 272.74163486350625|62.29958433849544
D1 | true | 195.00000000000002|0.0 | 259.96456724683867|145.92693981801575 | 293.7899889506232|38.282346886467096
D16 | false | 319.7867633291238|0.0 | 419.7867633291238|0.0 | 319.7867633291238|100.0
D4 | true | 421.68463037486055|0.0 | 324.2551687115628|97.42946166329774 | 336.56150787776722|148.3072958971631
D6 | false | 421.68463037486055|0.0 | 351.2856395520126|122.65391184599281 | 360.430250731063|156.4381698130399
D3 | false | 421.68463037486055|0.0 | 405.89928964911894|136.87886256896002 | 360.430250731063|156.43816981303993
D22 | true | 421.68463037486055|0.0 | 435.7101303101895|93.95895567514619 | 406.26096240604636|133.74270247900546
D21 | false | 421.68463037486055|0.0 | 459.6547102019473|87.08199031903588 | 441.5608484417555|133.15380563602844
D2 | true | 421.68463037486055|0.0 | 469.2890143088816|51.32078166067147 | 464.03050230741206|97.11759433939835
D23 | false | 421.68463037486055|0.0 | 491.52779246363144|14|683237068703252 | 487.1121689417112|70.5353613238301
D5 | false | 491.52779246363144|0.0 | 556.5277924636314|0.0 | 491.52779246363144|65.0

Die Lösung wurde in folgender Datei visuell gespeichert: 2019-04-27 15-02-33.svg
Die Lösung wurde in folgender Zeit berechnet:
Startzeit: 2019-04-27 15:02:32 +0200
Endzeit: 2019-04-27 15:02:34 +0200
Differenz: 1.986493

I:\BWINF.2\Dreiecksbeziehungen>dreiecksbeziehungen.rb
Geben den Namen der einzulesenen .txt Datei ein:
dreiecke5.txt
Es wurde der folgende minimale Gesamtabstand berechnet: 783.6788945422909 Meter
Es handelt sich bei dieser Lösung nicht zwangsweise um die optimalste Lösung, sondern ausschließlich um eine Annäherung an diese.
ID | ist Gespiegelt | x1|y1 | x2|y2 | x3|y3

D26 | false | 14.357244495246164|0.0 | 95.76977424343664|0.0 | 33.44521541085992|74.73051161556893
D34 | false | 95.76977424343664|0.0 | 16.77983827708138|94.71319874247251 | 0.0|179.89205191269897
D37 | false | 95.76977424343664|0.0 | 72.17966297621057|71.15832102010204 | 39.3763926167114456|105.9282139408691
D4 | true | 95.76977424343664|0.0 | 120.39210765486949|99.96869858799418 | 59.15375196047029|110.45029158935841
D24 | true | 95.76977424343664|0.0 | 165.99655672111328|86.2449941899979 | 123.5702458349301|112.87220108468887
D21 | true | 95.76977424343664|0.0 | 166.83265844029052|27.01974258989943 | 166.87107535277113|87.31898407883386
D25 | true | 95.76977424343664|0.0 | 154.25548279825674|8.324957803107349 | 172.6042036224227|29.21421677891763
D2 | true | 170.00783435318556|0.0 | 243.06900218361687|0.0 | 177.45315432316585|83.76491216227795
D29 | false | 243.06900218361687|0.0 | 197.1494181896232|58.62074552424728 | 212.22465824465496|107.66905984067579
D7 | true | 243.06900218361687|0.0 | 257.91160759747095|88.65493254483161 | 212.3997364802073|107.05791022252245
D1 | false | 243.0690021836169|0.0 | 270.62120400268645|51.096733505394035 | 261.073338443071|107.5399640199332
D10 | false | 243.0690021836169|0.0 | 297.20953593443016|41.92615655730566 | 293.0862965259812|92.7592058324633
D27 | false | 243.0690021836169|0.0 | 286.6533085528846|1.1866921714625374 | 277.8304889880154|26.91912025211131
D32 | false | 286.07492888591463|0.0 | 356.955109472592|0.0 | 319.87845450010235|69.35648243712303
D16 | true | 356.955109472592|0.0 | 312.9488403321783|82.31918534787929 | 330.67078459379184|101.88785141353056
D23 | false | 356.955109472592|0.0 | 384.214282048095|62.42545562908417 | 330.7076065415651|101.74511580358897
D18 | false | 356.955109472592|0.0 | 414.9757639647007|41.45604482228551 | 397.02494215658464|91.76278389781359
D8 | false | 356.955109472592|0.0 | 399.00790201061255|10.225587501664906 | 400.6958434425028|31.252970927153417
D14 | false | 393.3270200652432|0.0 | 464.03769818389793|0.0 | 428.68235912457055|63.63961030678929
D20 | false | 464.037698183898|0.0 | 477.7111197308156|83.91684898279173 | 416.0965983153742|86.29397976334286
D6 | true | 464.037698183898|0.0 | 503.46516019865504|43.99403640352734 | 479.8119294257356|96.8099872658176
D17 | false | 464.037698183898|0.0 | 525.125273387122|37.85905814834915 | 527.0000746699882|70.25481583747242
D28 | true | 464.03769818389804|0.0 | 496.67198853630265|5.568042133007868 | 514.2523470056711|31.120556609835855
D22 | true | 507.9731638348826|0.0 | 576.3836893408309|0.0 | 541.8276033801338|74.72534324495892
D12 | false | 576.3836893408309|0.0 | 593.9314550210876|74.00726937018284 | 538.0015794050743|82.99887732300655
D5 | true | 576.383689340831|0.0 | 621.3475847442131|47.3523838705721 | 597.4405025974636|88.80659105874577
D19 | true | 576.383689340831|0.0 | 638.9118204109978|19.29336281525615 | 639.2108527216708|66.1645489784164
D33 | true | 630.5577387207624|0.0 | 692.5416075900093|0.0 | 656.677437757543|60.322146200445495
D3 | false | 692.5416075900093|0.0 | 646.7839686788892|76.96257844744774 | 693.9095952583257|80.59236074057722
D11 | false | 692.5416075900093|0.0 | 721.1507845820432|45.51389886367464 | 694.0032458027509|86.10960233176701
D13 | true | 692.5416075900093|0.0 | 768.1947901962416|19.431828569336496 | 738.2593339690693|72.7312650958247
D9 | true | 760.8105986881724|0.0 | 822.1049703710602|0.0 | 785.7394780950003|65.60145555946023
D31 | false | 822.1049703710602|0.0 | 783.0167151451129|50.5131782250417 | 817.55512282248454|76.92398122359505
D15 | false | 822.1049703710602|0.0 | 849.3208897023061|57.873082991621544 | 817.4930982460835|77.97262747349394
D35 | false | 822.1049703710602|0.0 | 882.8168305704966|27.09372678544095 | 854.3381968710877|68.54209735190426
D30 | true | 822.1049703710602|0.0 | 869.9045131188543|0.451345878231681 | 886.9995007843534|28.96031633871016
D36 | true | 879.4486687857275|0.0 | 923.8333508280717|0.0 | 897.788329284423|70.33958240416187

Die Lösung wurde in folgender Datei visuell gespeichert: 2019-04-27 15-02-45.svg
Die Lösung wurde in folgender Zeit berechnet:
Startzeit: 2019-04-27 15:02:40 +0200
Endzeit: 2019-04-27 15:02:46 +0200
Differenz: 5.282262
```

Im Unterordner „Aufgabe 2\Beispiele“ befindet sich die oben zu sehende Ausgabe aus der Konsole für alle fünf Beispiele auf der Webseite sowie die fünf dazugehörigen SVG Dateien.