Aufgabe 3: Abbiegen?

Teilnahme-Id: 00041

Bearbeiter/-in dieser Aufgabe:
Sebastian Brunnert

16. April 2020

Inhaltsverzeichnis

Lösungsidee

Zum Lösen dieser Aufgabe, betrachte ich den Stadtplan als Koordinatensystem und die verschiedenen Graphen bzw. die zwei Vektoren als Straßen. Nachdem der Stadtplan eingelesen wurde, wird vom Startpunkt ausgehend jede direkt befahrbare Straße gespeichert und dazu auch die Kosten, die aufgebracht werden müssen, um diese Straße zu befahren. Die Liste wird nun wieder erweitert, indem von allen jetzigen potenziellen Standpunkten wieder mögliche weiter erreichbare Straßen und dazu addierte Kosten gespeichert werden. Dies geschieht immer wieder, bis die Liste leer ist. Eine Route wird aus der Liste geschmissen, wenn ein Straße mehrmals befahren wurde, da diese Route dann definitiv nicht mehr die kürzest mögliche ist mit dieser Anzahl an Kurven ist oder wenn die Route am Zielpunkt angekommen ist. Auch wird ein Pfad irgnoriert, wenn die benötigte Strecke zu dem jetzigen Standpunkt größer ist, als die kürzeste Strecke zu diesem Punkt in den vorherigen Durchläufen des Algorithmus unter Berücksichtigung des Abbiegezählers. Wenn die Route am Ziel angelangt ist, wird sie separat in einer anderen Liste gespeichert und schlussendlich bei Bedarf wird aus dieser Liste gesucht. Bei der Lösung dieser Aufgabe wird sich der Vorgang der Rekursion zu Nutze gemacht. Alles in Allem kann gesagt werden, dass der Algorithmus alle mögliche und rationale Wege organisiert durchprobiert werden. Nichtdestotrotz kann man nicht von einem Brute-Force sprechen, da nur gezielt ausgewählte Pfade ausprobiert werden.

Umsetzung

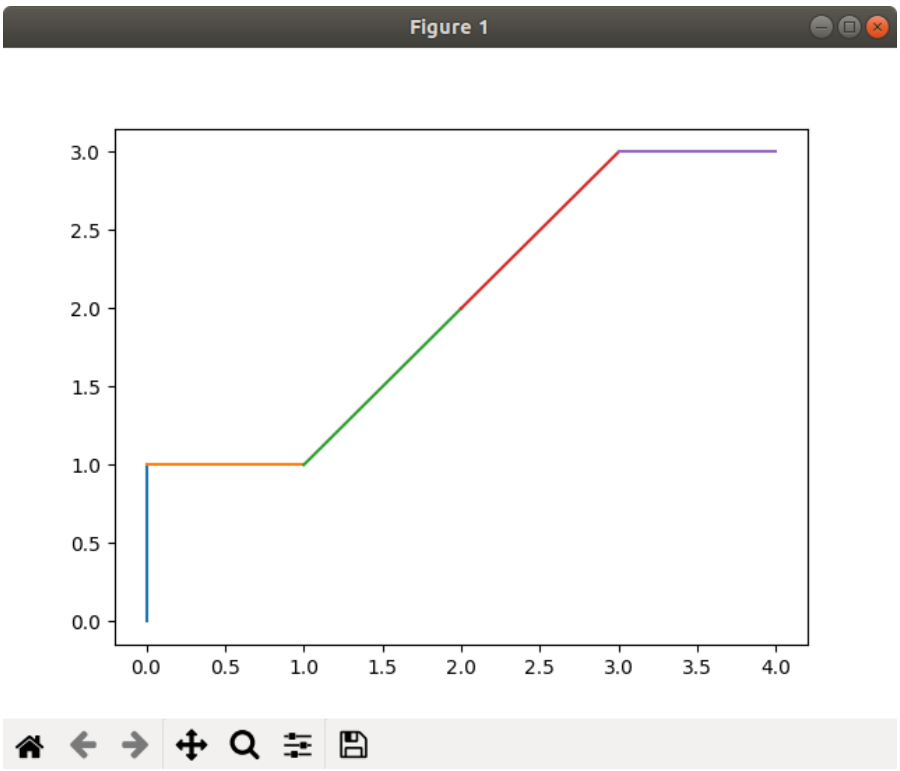
Die Umsetzung dieser Aufgabe geschieht in der Programmiersprache Python (3). Zu Beginn des Programmes werden alle nötigen Daten – wie die Vektoren der Straßen und Start- bzw. Endpunkt - aus der Kartendatei ausgelesen. Darüber hinaus wird für den Algorithmus das Rekursions-Limit des Programmes auf eine extremst hohe Zahl gesetzt. Dies ist notwendig, da der Algorithmus sich selber ausführt, um den nächsten Schritt zu simulieren.

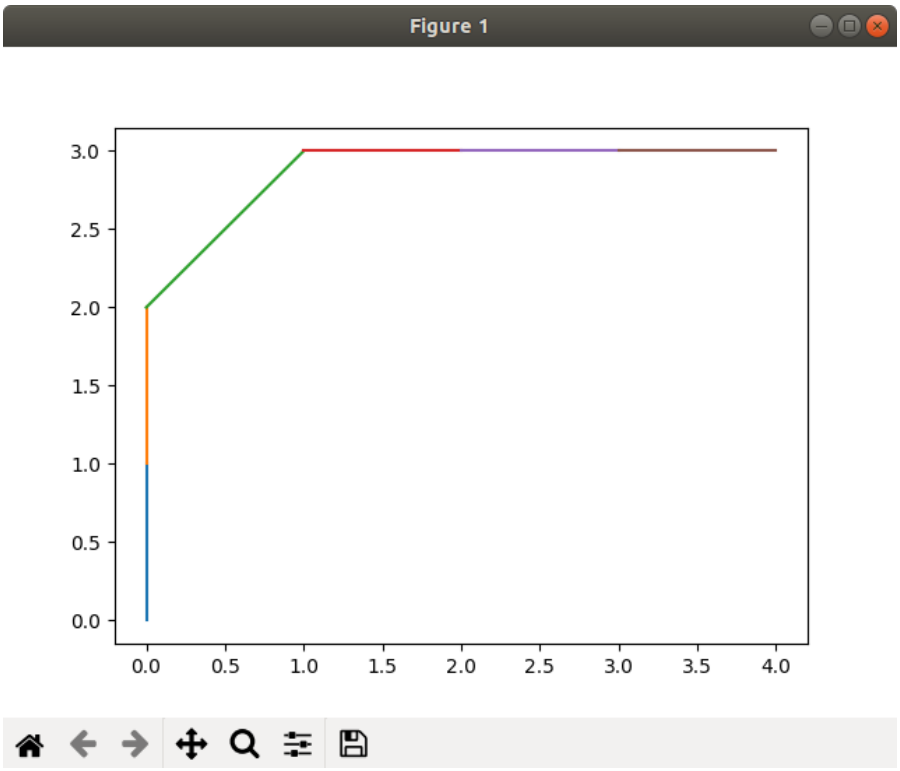
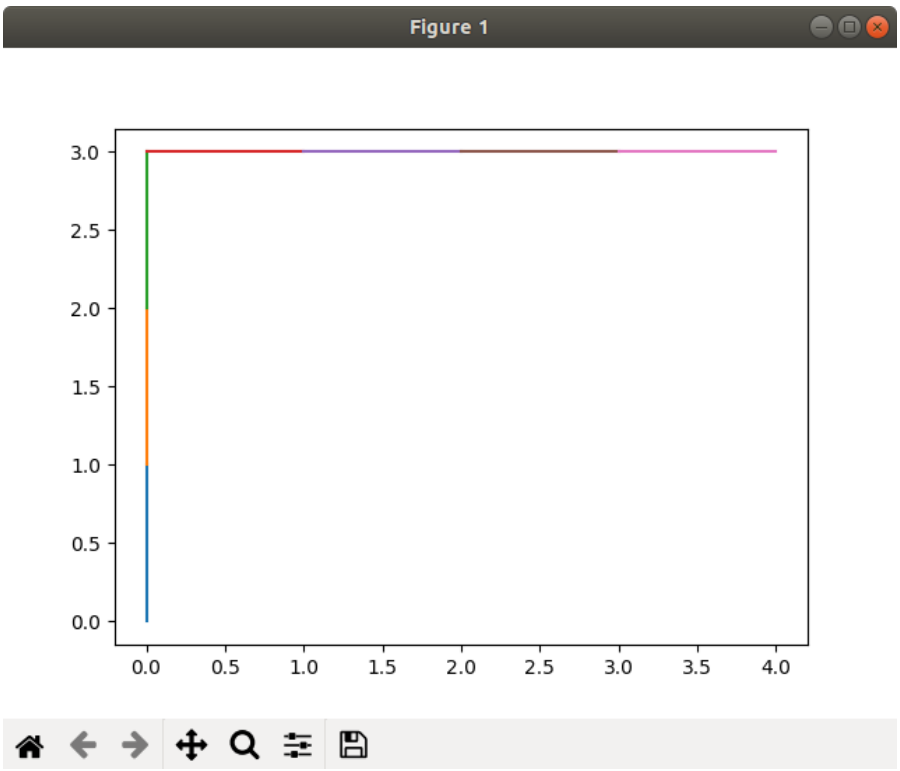
Anschließend werden zwei Hilfsfunktionen aufgestellt, die für den Algorithmus benötigt werden. Zum einen ist das die Funktion moegliche Strassen(akutelleKoord). Diese gibt als Liste aus, welche Straßen an einem bestimmten Punkt aktuelleKoord genommen werden können. Dabei werden alle Straßen durchgegangen und geprüft, ob eine Koordinate der Straße gleich aktuelleKoord ist. Ist dies der Fall wird diese Straße mit allen nötigen Daten zwischengespeichert und zum Schluss ausgegeben – auch die nötige Strecke wird errechnet mithilfe des Satz des Pythagoras. Die zweite Hilfsfunktion ist die Funktion mussAbgebogenWerden(von,zu).

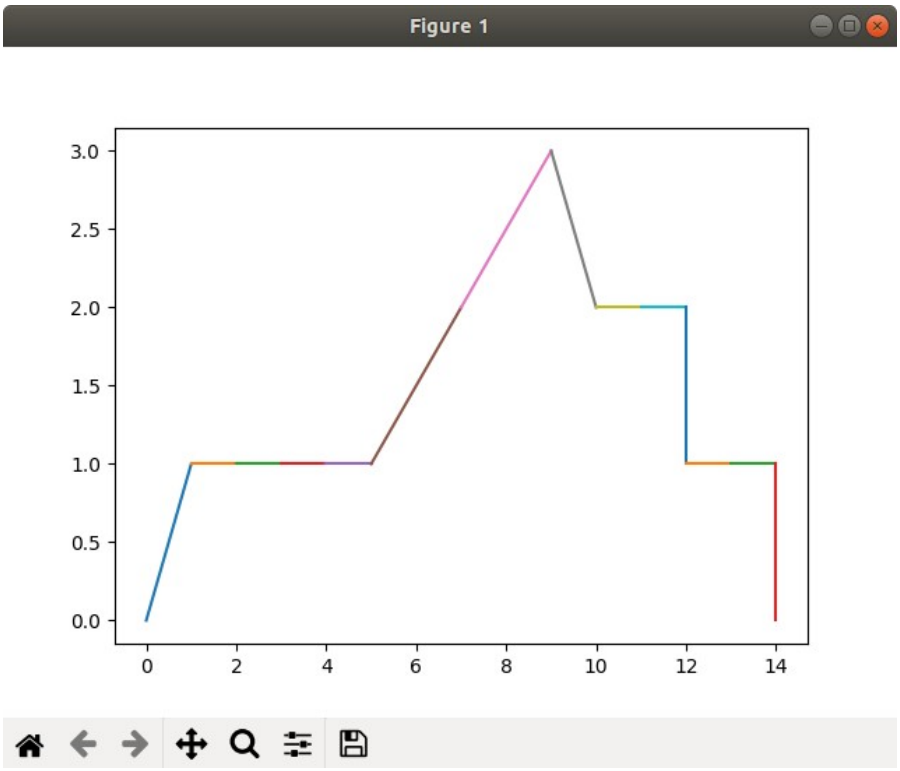
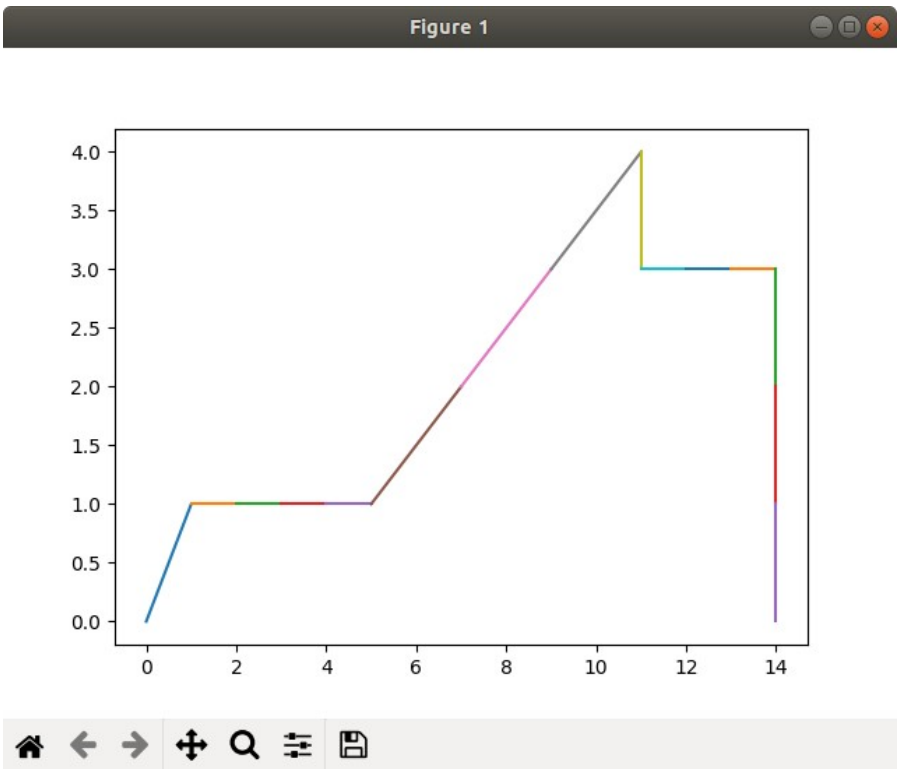
Diese Funktion gibt die Möglichkeit, zu ermitteln, ob abgebogen werden muss, um von Straße „von“ zu Straße „zu“ zu gelangen. Mathematisch bedeutet Abbiegen, dass die Steigung der Graphen sich verändert. Es wird also die Steigung beider Graphen aufgestellt und miteinader verglichen. Da senkrechte Straßen bzw. Graphen keine Steigung haben, wird bie diesen nur geprüft, ob alle x-Koordinaten der Start- und Endpunkte beider Graphen gleich sind.

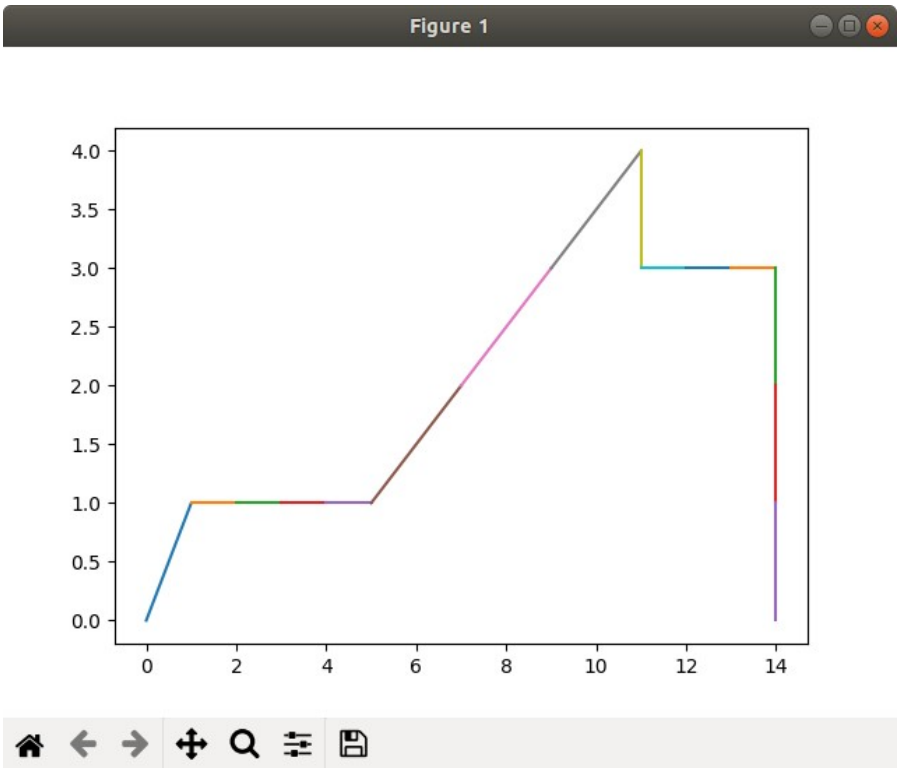
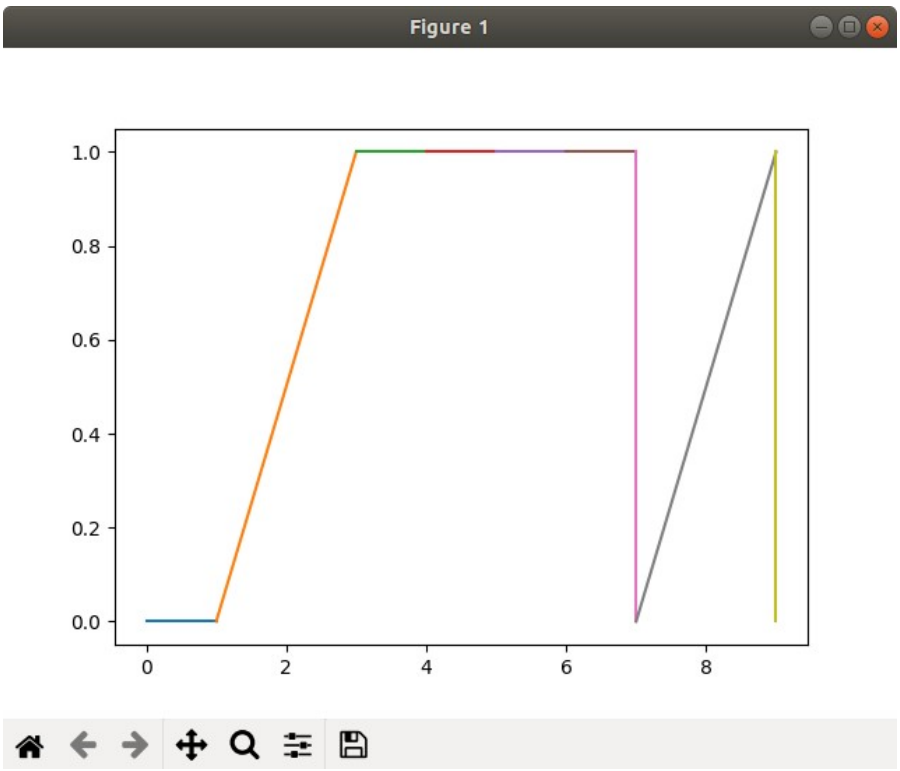
Wenn nun der Algirthmus startet, wird erst geprüft, ob bereits Pfade aufgestellt wurden. Da dies nicht der Fall ist werden mithilfe des Aufrufens der Funktion moeglicheStrassen(startPunkt) alle möglichen Pfade vom Startpunkt ausgehend ermittelt. Sollte bereits ein Pfad am Ziel angelangt sein, speichert der Algorithmus diesen Pfad und stoppt der Algorithmus. Falls nicht, werden alle Pfade gespeichert samt Abbiegezähler 0 (da davon ausgegangen wird, dass der erste Schritt kein Abbiegen darstellt), aktuellen Koordinaten und zurückgelegter Strecke. Extern wird ebenfalls gespeichert wieviel Strecke zurückgelegt werden musste, um zu den aktuellen Koordinaten zu gelangen zusammen mit dem Abbiegezähler, der also gleich 0 ist. Nun führt der Algorithmus sich wieder selber aus und es wurden bereits Pfade aufgestellt. Also werden diese kopiert und geleert, damit die Liste für den eventuellen nächsten Durchlauf wieder leer ist. Nun werden alle Pfade weiter simuliert. Ist das Ende angelangt wird dieser erfolgreiche Pfad gespeichert. Falls nicht wird erst geprüft, ob dieser Pfad bereits die aktuellen Koordinaten durchlaufen ist. Wenn ja, ist dieser Weg sicher nicht der kürzeste Weg dieser oder niedriger Länge mit dieserm Abbiegezähler und der Pfad wird vernachlässigt. Auch wird der Pfad vernachlässigt, wenn die Strecke zu diesem spezifischen Punkt länger ist als alle anderen vorherigen mit dem selben Abbiegezähler. Nun wird der mögliche Pfad wieder samt wichtiger Daten gespeichert. Darunter auch, ob bzw. wie oft abgebogen werden muss, was durch die zuvor aufgestellt Hilfsmethode ermittelt wird. Nun wird der Algorithmus so oft wiederholt, bis die Liste an möglichen Pfaden leer ist. Anschließend wird der kürzest mögliche Weg ermittelt und der Nutzer wird nach der maximalen Verlängerung gefragt. Es wird nun also aus der Liste an erfolgreichen Pfaden der optimale Weg gesucht und mithilfe der Libary matplotlib im Koordinatensystem dargestellt.

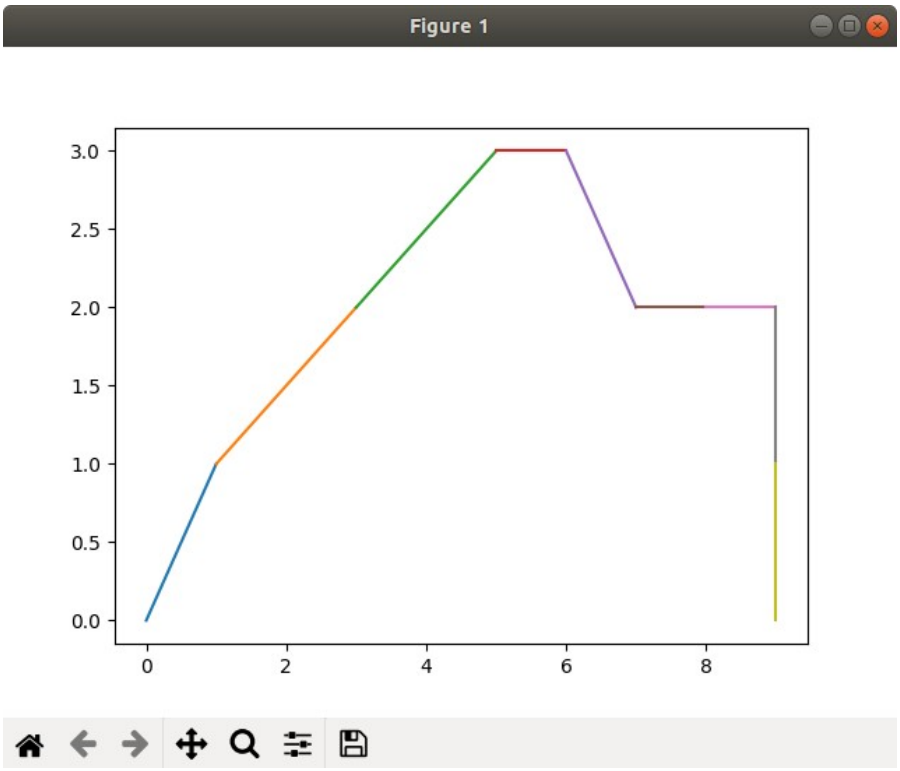
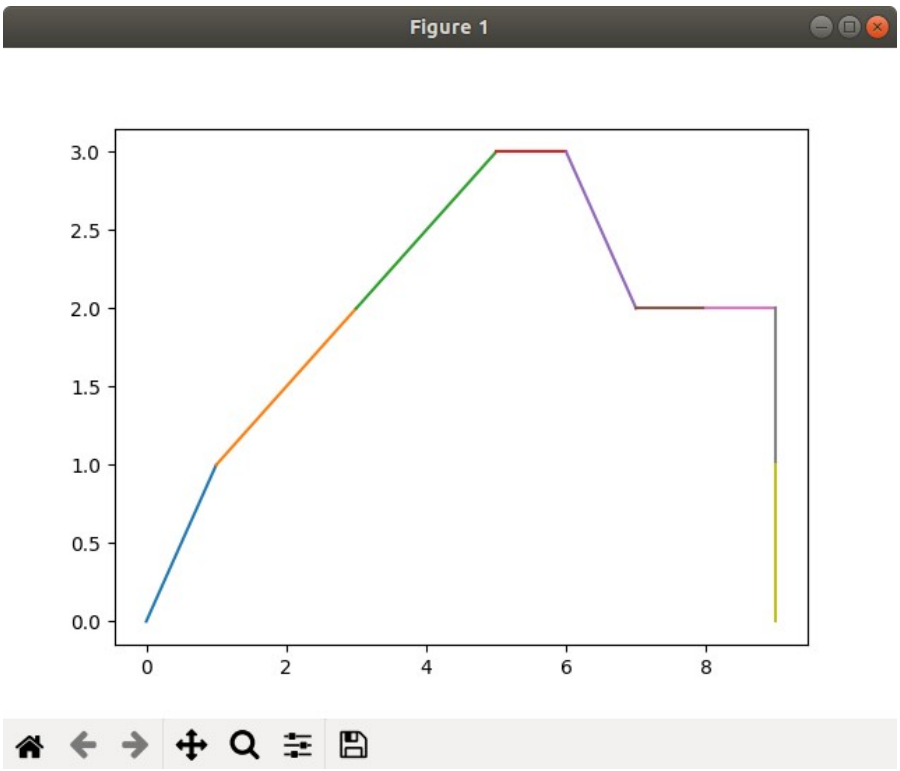
Beispiele

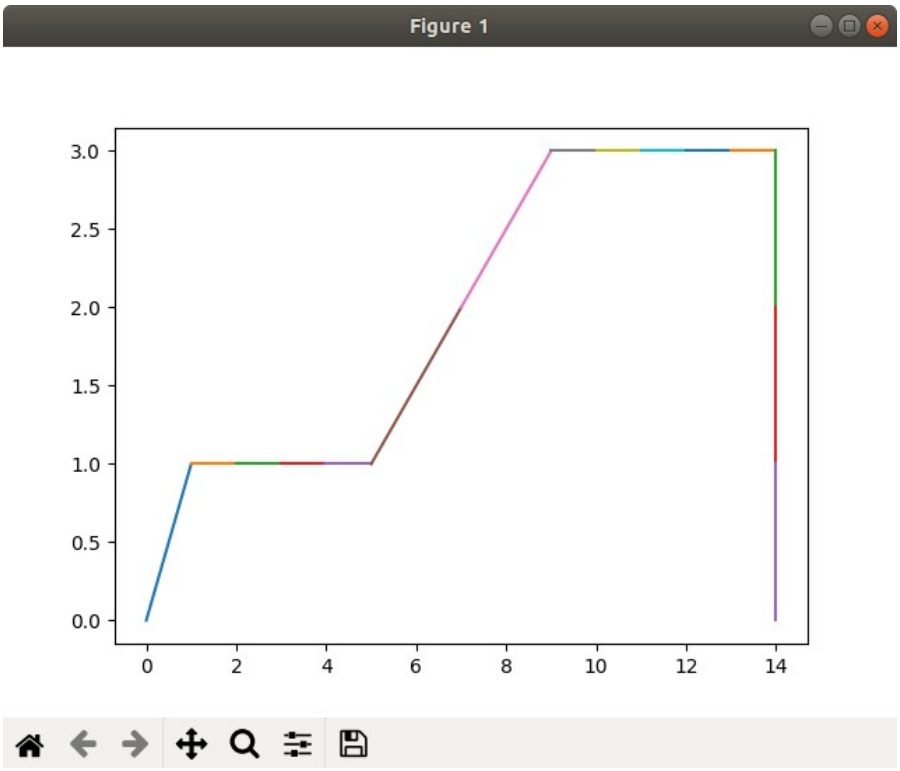
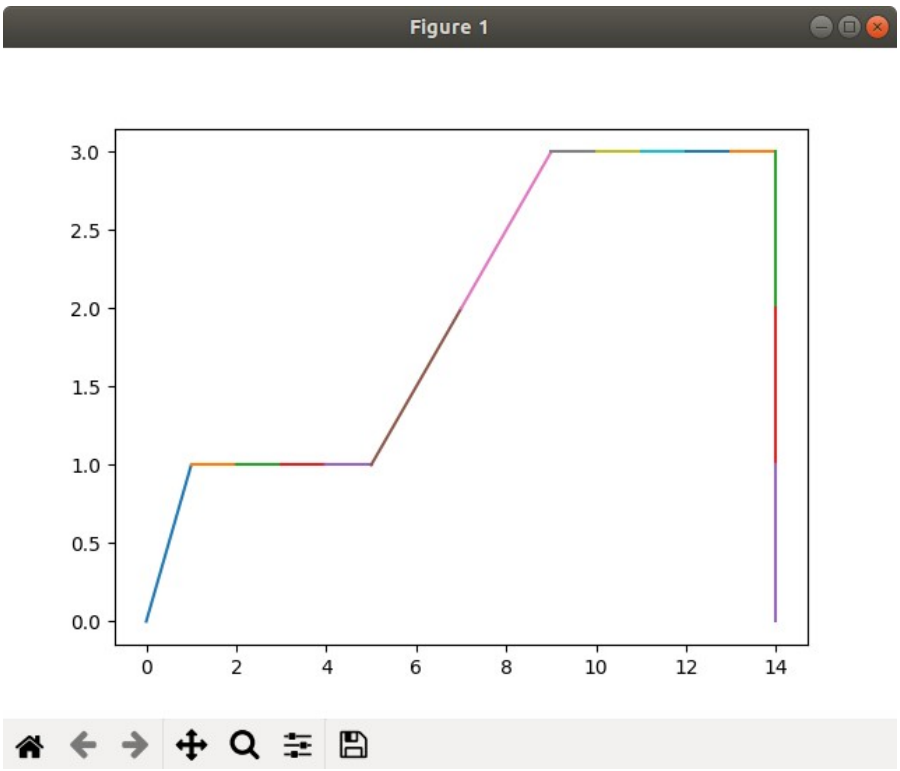
python3 abbiegen.py abbiegen0.txt 10% Verlängerung	Ausgabe Sofort: 

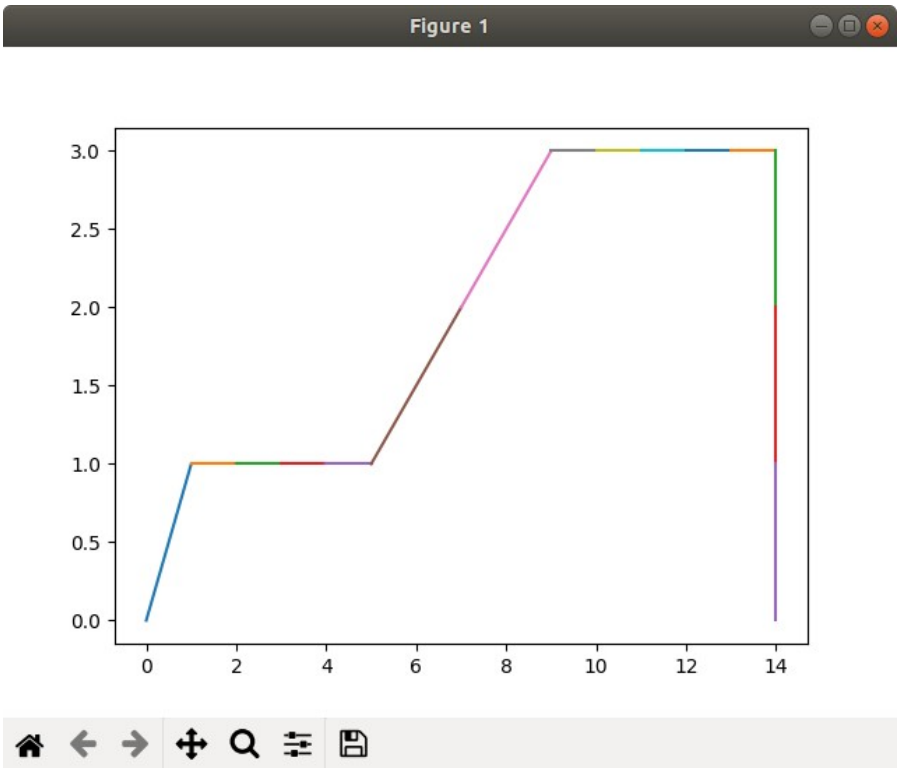
python3 abbiegen.py abbiegen0.txt 15% Verlängerung	Ausgabe Sofort: 
python3 abbiegen.py abbiegen0.txt 30% Verlängerung	Ausgabe Sofort: 

python3 abbiegen.py abbiegen1.txt 10% Verlängerung	Ausgabe Sofort: 
python3 abbiegen.py abbiegen1.txt 15% Verlängerung	Ausgabe Sofort: 

python3 abbiegen.py abbiegen1.txt 30% Verlängerung	Ausgabe Sofort: 
python3 abbiegen.py abbiegen2.txt 10% Verlängerung	Ausgabe Sofort: 

python3 abbiegen.py abbiegen2.txt 15% Verlängerung	Ausgabe Sofort: 
python3 abbiegen.py abbiegen2.txt 30% Verlängerung	Ausgabe Sofort: 

python3 abbiegen.py abbiegen3.txt 10% Verlängerung	Ausgabe Sofort: 
python3 abbiegen.py abbiegen3.txt 15% Verlängerung	Ausgabe Sofort: 

python3 abbiegen.py abbiegen3.txt 30% Verlängerung	Ausgabe Sofort: 

* Zeitangaben: Getestet wurde das Programm aus einen System mit folgenden Spezifikationen:

CPU: AMD Ryzen 5 2600

RAM: 16 Gigabyte 2666 MHz

Quellcode

Der vollständige Quellcode, ist in der Datei abbiegen.py zu finden.

```
# Array der alle Wege die mithilfe des Algorithmus berechnet werden in sich trägt
#   - ["koords"]["x"]: Int - aktuelle x-Koordinate
#   - ["koords"]["y"]: Int - aktuelle y-Koordinate
#   - ["strecke"]: Int - Zurückgelegte Strecke
#   - ["abbiegen"]: Int - Gibt an wie oft abgebogen werden musste
abzweigungen = []

# Dieser Array gibt an, welche Wege erfolgreich am Endpunkt angelangt sind
erfolgreicheWege = []

# Dieser Boolean gibt an, ob die Zahl der Wege, die in der Berechnung sind bereits einmal null war. Dies dient dazu, dass nicht alle Wege wieder berechnet werden und es zu einer Rekursion kommt.
ende = False

# Dieses Dictionary gibt dem Algorithmus an wie kurz die kürzeste Strecke war, um zu einen spezifischen Punkt zu gelangen (in Relation zum Abbiegezähler). Das ist insofern wichtig, dass somit die definitiv kürzeren Strecken von vornherein ausgeschlossen werden.
kuerzesteStreckeZuPunkt = {}

def naechsterSchritt():
    global abzweigungen, ende
    # Prüfe, ob noch kein Weg in der Berechnung war
    if(abzweigungen == []):
        # Wurde dieser Durchlauf bereits einmal getätigt?
        if(ende):
            # Beende die Funktion
            return
        # Gehe alle Straßen durch, die vom Startpunkt aus erreicht werden können.
        for moeglicheStrasse in moeglicheStrassen(start):
            # Prüfe, ob bereits jetzt das Ziel erreicht wurde
```

```
        if(moeglicheStrasse["koords"] == ziel):
            # Wenn ja, soll dies natürlich im nächsten Durchlauf nicht noch einmal wiederholt werden und dieser Pfad wird nicht mit in den Array für die Berechnungen des nächsten Durchlauf gegeben.
            erfolgreicheWege.append({
                "strecke": moeglicheStrasse["strecke"],
                "gefahrenerWeg": [moeglicheStrasse["index"]],
                "abbiegen": 0
            })
        else:
            # Prüfe, ob ein Pfad bereits über diesen Punkt führte
            punktVereinfacht = str(moeglicheStrasse["koords"]["x"]) + "-" + str(moeglicheStrasse["koords"]["y"]) + "-0"
            if(not punktVereinfacht in kuerzesteStreckeZuPunkt):
                # Wenn nicht, setze die nötige Strecke auf die gegebene Strecke
                kuerzesteStreckeZuPunkt[punktVereinfacht] = moeglicheStrasse["strecke"]
            else:
                # Wenn nicht, prüfe, ob die jetzig erreichte Strecke kürzer als die vorherige Strecke ist
                if(kuerzesteStreckeZuPunkt[punktVereinfacht] > moeglicheStrasse["strecke"]):
                    # Wenn ja, wird die kürzest benötigte Strecke neu gesetzt
                    kuerzesteStreckeZuPunkt[punktVereinfacht] = moeglicheStrasse["strecke"]
                else:
                    # Wenn nicht, wird dieser Pfad ignoriert
                    continue
            # Gebe diese Straßen in dein Array
            abzweigungen.append({
                "strecke": moeglicheStrasse["strecke"],
                "koords": moeglicheStrasse["koords"],
                "gefahrenerWeg": [moeglicheStrasse["index"]],
                "abbiegen": 0
            })
    # Setze ende auf wahr. Das bedeutet, dass dieser Durchlauf bereits einmal getätigt wurde,
    ende = True
    # Führe diese Funktion wieder aus (es wird die zweite Verzweigung ausgeführt)
    naechsterSchritt()
```

```
    else:
        # Kopiere den Array, damit die angefangenen Wege nicht mit in der Berechnung berücksichtigt werden und im nächsten Durchlauf wieder das selbe berechnet wird. Es wird also die Rekursion verhindert.
        abzweigungenDuplikat = abzweigungen.copy()
        # Leere den Array
        abzweigungen = []
        # Gehe alle nötigen Berechnungen durch
        for i in range(0,len(abzweigungenDuplikat)):
            # Gehe alle möglichen Abzweigungen durch
            abzweigung = abzweigungenDuplikat[i]
            for moeglicheStrasse in moeglicheStrassen(abzweigung["koords"]):
                # Prüfe, ob das Ziel erreicht wurde
                if(moeglicheStrasse["koords"] == ziel):
                    # Wenn ja, soll dies natürlich im nächsten Durchlauf nicht noch einmal wiederholt werden und dieser Pfad wird nicht mit in den Array für die Berechnungen des nächsten Durchlauf gegeben.
                    erfolgreicheWege.append({
                        "strecke": abzweigung["strecke"] + moeglicheStrasse["strecke"],
                        "gefahrenerWeg": abzweigung["gefahrenerWeg"] + [moeglicheStrasse["index"]],
                        "abbiegen": abzweigung["abbiegen"] + mussAbgebogenWerden(abzweigung["gefahrenerWeg"][-1],moeglicheStrasse["index"])
                    })
                # Prüfe, ob dieser Pfad diese Stelle schon durchlaufen ist. Wenn ja ist dieser Weg ganz sicher nicht der kürzeste mit der Anzahl an Abbiegen und der Weg wird vernachlässigt.
                elif(abzweigung["gefahrenerWeg"].count(moeglicheStrasse["index"]) < 1):
                    # Prüfe, ob ein Pfad bereits über diesen Punkt führte (in Relation zum Abbiegezähler)
                    punktVereinfacht = str(moeglicheStrasse["koords"]["x"]) + "-" + str(moeglicheStrasse["koords"]["y"]) + "-" + str(abzweigung["abbiegen"] + mussAbgebogenWerden(abzweigung["gefahrenerWeg"][-1],moeglicheStrasse["index"]))
                    if(not punktVereinfacht in kuerzesteStreckeZuPunkt):
                        # Wenn nicht, setze die nötige Strecke auf die gegebene Strecke
                        kuerzesteStreckeZuPunkt[punktVereinfacht] = moeglicheStrasse["strecke"]
                    else:
                        # Wenn nicht, prüfe, ob die jetzig erreichte Strecke kürzer als die vorherige Strecke ist
                        if(kuerzesteStreckeZuPunkt[punktVereinfacht] > moeglicheStrasse["strecke"]):
                            # Wenn ja, wird die kürzest benötigte Strecke neu gesetzt
                            kuerzesteStreckeZuPunkt[punktVereinfacht] = moeglicheStrasse["strecke"]
```

```
                else:
                    # Wenn nicht, wird dieser Pfad ignoriert
                    continue
            # Ansonsten wird der Pfad in den Array für den nächsten Berechnungsdurchlauf gegeben.
            abzweigungen.append({
                "strecke": abzweigung["strecke"] + moeglicheStrasse["strecke"],
                "koords": moeglicheStrasse["koords"],
                "gefahrenerWeg": abzweigung["gefahrenerWeg"] + [moeglicheStrasse["index"]],
                "abbiegen": abzweigung["abbiegen"] + mussAbgebogenWerden(abzweigung["gefahrenerWeg"]
[-1],moeglicheStrasse["index"])
            })

    naechsterSchritt()

# Führe den Algorithmus aus. Dieser wird sich selber so oft wie nötig ausführen (Rekursion)
naechsterSchritt()
```