Aufgabe 2: Vollgeladen

Team-ID: 00023

Team-Name: Kallisto

Bearbeiter/-innen dieser Aufgabe: Alexander Lu

22.11.2021

Inhaltsverzeichnis:

Lösungsidee:

Es soll ein Programm geschrieben werden, welches einen optimierten Weg für die Familie sucht. Der optimierte Weg zeichnet sich durch Hotel besuche aus, wo die schlechteste Bewertung der Hotels möglichst hoch ist. Außerdem muss der Fahrtdauer nicht mehr als 5 Tage dauern, sprich 4 Hotel besuche und der Restfahrt zum Ziel, und auch nicht mehr als 6 Stunden am Stück fahren, da dann der Handy Akku der beiden Kinder ausgeht. Die Lösungsidee besteht darin einen Algorithmus zu finden, der zum Teil alle Wege sich anschaut und den besten ausgibt. Ein simpler Bruteforce Algorithmus mit Backtracking und anderen Optimierungen sollte sich bestens dafür eignen. Dafür geht das Programm alle Wege durch, und falls es einen Weg gefunden hat, müssen alle neuen durchsuchten Wege mindestens eine bessere Bewertung haben, als die schlechteste Bewertung des gefundenen Weges. Dieser Wert wird mit jedem neuem gefunden Weg, falls die schlechteste Bewertung besser ist, ersetzt, somit am Ende, der Weg übrigbleibt, der die höchste schlechteste Bewertung hat.

Umsetzung:

Die Implementierung erfolgt in der Skripting Sprache Python, weil es schnell gehen soll. Die Klasse Aufgabe wird implementiert und enthält Hilfsmethoden für die Eingabe und Eingabeformatierung. Die Einstiegsmethode oder auch „Main“ Methode *main* beginnt den Lösungsvorgang indem sie eine Eingabedatei, als Argument, mithilfe der Methode *readInput* einliest und die Eingabe in das geeignete Format formatiert. So entsteht die Instanz variable HOTELS welche in einer Matrix Liste jeweils die Distanz des Hotels vom Anfang im 0ten Index speichert, und die Bewertung des Hotels im ersten Index. Danach wird der Lösungsvorgang initialisiert, mit der Methode *loesen*. Die Methode *loesen* erhält als ersten Parameter eine Liste des aktuellen Hotels und den zweiten Parameter mit der bereits gewählten Strecke. Als Anfangswert wird die Strecke mit einer leeren Liste initialisiert und das Starthotel kann imaginär am Punkt 0 mit der Bewertung von 0 gesetzt werden. Die Methode *loesen* beginnt mit einer bedingten Anweisung, die überprüft ob die Strecke schon über 5 Hotels beinhaltet, und somit das Ziel nicht in 5 Tagen erreicht werden kann oder ob die restliche Strecke nicht mehr in dem restlichen Zeitraum überquerbar ist (restliche Strecke > übrige Tage * 6

Stunden Fahrt pro Tag). Ist dies der Fall wird der Funktionsablauf abgebrochen mit dem *return*. Als nächstes, falls die Strecke und Zeit noch alles im Rahmen ist, werden die nächsten möglichen Hotels mithilfe der Hilfsmethode *getMoeglicheStrecken* in einer Liste zurückgegeben. Die Hilfsmethode funktioniert so, indem sie die aktuelle Position des Autos als Argument nimmt, und alle Hotels zurückgibt, die sich innerhalb des 6 Stunden Fahrt Radius befinden und sich auf dem Weg befinden und keine Rückfahrt nötig ist. Dann erfolgt eine bedingte Anweisung, die überprüft ob das Ziel erreicht wurde und vergleicht die niedrigste Bewertung der getroffenen Hotels mit der Instanz Variable *bewertungMinimum*. Falls die niedrigste Bewertung höher als die Instanz variable ist, dann wird diese überschrieben und die Instanzvariable wird der Wert der Strecke hinzugewiesen. Im nächsten Schritt erfolgt die Rekursion mit dem Backtracking. Es wird durch jede mögliche Strecke iteriert und rekursiv die Methode *loesen* aufgerufen, mit der aktuellen Strecke in der Iteration und dem Lösungsweg als Argument. Vor dem rekursiven Aufruf wird der Lösungsweg noch um die Strecke erweitert und nach dem Aufruf wieder um die Strecke verkürzt. Dass ermöglicht, dass alle möglichen Strecken ausprobiert werden, durch den rekursiven Aufruf. Der rekursive Aufruf setzt dann die nächsten möglichen Strecken an, bis eine Lösung gefunden wurde bzw eine Abbruchbedingung einsetzt. In diesem Fall wird dann in der nächsten Iteration das nächste Hotel bzw Strecke verwendet und diese Strecke wird wieder bis zur Abbruchbedingung ausgeführt. Im letzten Teil wird in der Methode *main* eine lesbare Ausgabe generiert, die die Instanzvariable *loesung* ausliest.

Beispiele:

Input:
12
1680
12 4.3
326 4.8
347 2.7
359 2.6
553 3.6
590 0.8
687 4.4
1007 2.8
1008 2.6
1321 2.1
1360 2.8
1411 3.3
Output:

Die geeigneteste Strecke nach den Kriterien ist:
- Stop bei 347 für das Hotel mit der Bewertung 2.7
- Stop bei 687 für das Hotel mit der Bewertung 4.4
- Stop bei 1007 für das Hotel mit der Bewertung 2.8
- Stop bei 1360 für das Hotel mit der Bewertung 2.8
Die schlechteste Bewertung der Hotels ist 2.7

Input:
25
1737
340 1.6
341 2.2
341 2.3
342 2.1
360 1.9
361 4.4
362 3.1
442 5.0
567 4.9
700 3.0
710 2.9
718 1.4
987 4.6
1051 2.3
1053 4.8
1057 0.2
1199 5.0
1279 5.0
1367 4.5
1377 1.8
1377 1.6
1377 2.0
1378 2.1
1378 2.2
1380 5.0
Output:
Die geeigneteste Strecke nach den Kriterien ist:
- Stop bei 358 für das Hotel mit der Bewertung 2.5
- Stop bei 717 für das Hotel mit der Bewertung 0.3
- Stop bei 1075 für das Hotel mit der Bewertung 0.8
- Stop bei 1433 für das Hotel mit der Bewertung 1.7
Die schlechteste Bewertung der Hotels ist 0.3

Input:
<hotels5.txt> (Zu lang mit über 1500 Teilen, befindet sich im Assets Ordner der Aufgabe 2)
Output:
Die geeigneteste Strecke nach den Kriterien ist:
- Stop bei 280 für das Hotel mit der Bewertung 5.0
- Stop bei 636 für das Hotel mit der Bewertung 5.0
- Stop bei 987 für das Hotel mit der Bewertung 5.0
- Stop bei 1271 für das Hotel mit der Bewertung 5.0
Die schlechteste Bewertung der Hotels ist 5.0

Input (Selfmade):
4
500
12 4.3
326 4.8
347 2.7
359 2.6
Output:
Die geeigneteste Strecke nach den Kriterien ist:
- Stop bei 326 für das Hotel mit der Bewertung 4.8
Die schlechteste Bewertung der Hotels ist 4.8
(Test ob es auch nur ein Hotel sein kann)

Input (Selfmade):
12
1680
12 4.3
12 5.0
326 4.8
326 5.0
687 4.4
687 5.0
1007 2.8
1007 5.0
1321 2.1
1321 5.0
(Distanz zwischen 326 und 687 ist zu groß und fällt nicht unter der 6 Stunden Marke)
Output:
Das Ziel kann nicht erreicht werden, da die Distanz zwischen den Hotels zu groß ist.

Info:
Alle Input Dateien befindet sich auch im Aufgabe 2/assets Ordner

Quellcode:

```
class Aufgabe:
    def __init__(self):
        self.HOTELS = []
        self.bewertungMinimum = 0
        self.maximaleLaenge = 360
        self.strecke = 1510
        self.loesung = []

    def _getMoeglicheStrecken(self, letztesVerwendetesHotel):
        moeglicheStrecken = []

        """

            Bedingte Anweisung, die alle Hotels rausfiltert, die sich
            innerhalb 360 Längeneinheiten des letzten verwendeten Hotels
            befindet.
            Die erste Abfrage dient zum filtern des Bereichs auf innerhalb
            360 Längeneinheiten, der zweite Bereich sorgt dafür, dass man
            nicht zurückfährt oder zu einem Hotel der gleichen Entfernung
fährt.
            In dem Fall lohnt es sich nicht zurück zufahren um ein Hotel zu
besuchen
            dass hinter Einem liegt, weil man das Hotel, dass Hinter einem
liegt
            zuerst besuchen könnte aber auch, da die perfekte Strecke 5 * 260
1800 ergibt,
            und man mit einer Zurückfahrt oder das Bleiben im Hotel für
maximale Ideale Strecke von 1540 liefert,
            welche in 4/5 Beispielen übertroffen wird.
            Das Wechseln in ein Hotel mit der selben Entfernung, aber
unterschiedlicher
            Bewertung macht ebenfalls kein Sinn, da man nur das Hotel mit der
besseren Bewertung
            besuchen könnte und das andere Ignorieren.

        """
        for hotel in self.HOTELS:
            if hotel[0] - letztesVerwendetesHotel[0] <= self.maximaleLaenge
and \
               hotel[0] - letztesVerwendetesHotel[0] > 0:
                moeglicheStrecken.append(hotel)

        return moeglicheStrecken

    def _getSchlechtesteBewertung(self, strecke):
        """

            6 als Wert der nicht durch Bewertungen erreicht werden kann.
            Theoretisch kann jeder Wert über 5 für diese Variable
```

```
            verwendet werden.

        """

        schlechtesteBewertung = 6

        for hotel in strecke:
            if hotel[1] < schlechtesteBewertung:
                schlechtesteBewertung = hotel[1]

        return schlechtesteBewertung

    # def _sortiere_nach_bester():

    def _amEnde(self, letztesVerwendetesHotel):
        if (self.strecke - letztesVerwendetesHotel[0]) <= self.maximaleLaenge:
            return True

        return False

    def _loesen(self, letztesVerwendetesHotel, loesungsStrecke):
        """

            Optimierung: Falls eine Strecke über 4 Hotels enthält, kann
            mithilfe der Rückgabe der Vorgang abgebrochen werden, weil
            somit das Ziel nicht innerhalb 5 Tage erreicht wird.
            Außerdem wird abgebrochen, wenn es nicht mehr möglich ist
            die restliche Strecke, innerhalb der verbleibenden Tage und pro
            Tag 6 Fahrstunden, zu überqueren

        """

        if len(loesungsStrecke) > 4 or ( len(loesungsStrecke) != 0 and \
            self.strecke - loesungsStrecke[-1][0] > (5 - len(loesungsStrecke))
* self.maximaleLaenge):
            return

        """

            Die Methode _getMoeglicheStrecken gibt alle Strecken zurück
            die ab dem letzten verwendeten Hotels möglich sind, innerhalb
            der 6 Stunden Grenze

        """

        moeglicheStrecken =
self._getMoeglicheStrecken(letztesVerwendetesHotel)

        """
```

```
            Falls eine komplette Strecke gefunden wurde,
            vergleiche die schlechteste Bewertung dieser mit der
            bewertungMinimum Variable und ersetze diese, falls die
            schlechteste Bewertung besser ist

        """

        if len(moeglicheStrecken) == 0 or
self._amEnde(letztesVerwendetesHotel):
            schlechtesteBewertung =
self._getSchlechtesteBewertung(loesungsStrecke)

            if self.bewertungMinimum < schlechtesteBewertung:
                self.bewertungMinimum = schlechtesteBewertung
                #print(loesungsStrecke, self.bewertungMinimum)
                self.loesung = loesungsStrecke[:]
            return

        # Für optimierung: sorten nach bester bewertung

        for strecke in moeglicheStrecken:

            """

                Falls eine Strecke mit einem Hotel ausgewählt wird,
                wessen bewertung schon unter der mindes Bewertung einer
                anderen Strecke ausgewählt wird, wird diese Strecke
                übersprungen

            """
            if strecke[1] < self.bewertungMinimum:
                continue

            """

                Strecke wird hier rekursiv aufgebaut und nach jedem
                rekursivem Durchlauf wieder an den Anfangszustand gebracht,
                damit die nächste Strecke in der nächsten Iteration verwendet
                werden kann.

            """
            loesungsStrecke.append(strecke)
            self._loesen(strecke, loesungsStrecke)
            loesungsStrecke.pop()

    def _readInput(self, file):
        f = open("Aufgabe 2/assets/" + file, "r")
        content = f.read()
        f.close()
```

```
        data = content.splitlines()

        for i in range(0, len(data)):
            if(i == 0):
                continue
            elif(i == 1):
                self.strecke = int(data[1])
            else:
                [ strecke, bewertung ] = data[i].split(" ")
                self.HOTELS.append([ int(strecke), float(bewertung) ])

    def _formatierLoesung(self, loesungsweg):
        if self.strecke > loesungsweg[-1][0] + self.maximaleLaenge:
            print("Das Ziel kann nicht erreicht werden, da die Distanz
zwischen den Hotels zu groß ist.")
            return

        print("Die geeigneteste Strecke nach den Kriterien ist:")
        for [ stop, hotel ] in loesungsweg:
            print(f"- Stop bei {stop} für das Hotel mit der Bewertung
{hotel}")


        print(f"Die schlechteste Bewertung der Hotels ist
{self._getSchlechtesteBewertung(loesungsweg)}")


    def main(self, file):
        self._readInput(file)
        self._loesen( [0, 0], [] )
        self._formatierLoesung(self.loesung)



test = Aufgabe()
test.main("hotels7.txt")
```