Vollgeladen

Team-ID: 00940

Team: Lovely_Infestation

Bearbeiter/-innen dieser Aufgabe: Lars Noack

18. November 2020

Inhaltsverzeichnis

Lösungsidee

- man hat eine Liste von Hotels
- man muss vom Start zum Zielpunkt kommen
- Bedingungen:
    - man darf maximal 5 Tage fahren
    - jeden Tag darf man maximal 6 Stunden bzw. 360 min fahren
    - man muss jede Nacht in einem Hotel übernachten
- gesucht wird die Route bei der das Hotel mit der geringsten Bewertung, die beste Bewertung hat.

- man startet am Start
- man füllt alle Punkte bis zu einer Route auf.
    - man bekommt für den nächsten Stopp auf der Route alle Hotels, die man von dort aus erreichen kann, von denen man aber immer noch ans Ziel kommen kann.
    - man sortiert die Hotels absteigend nach der Bewertung
    - dies Speichert man ab
    - man macht dies, bis die Route voll ist

- dann berechnet man die minimale Bewertung dieses Pfades, und vergleicht, ob diese größer ist, als die vorherige (bei dem ersten Durchlauf ist die vorherige 0)
- dann löscht man das letzte Hotel, das ganz oben im letzten Element steht
    - ist im letzten Element kein Hotel mehr, löscht man auch das oberste Hotel aus dem letzten Element und immer so weiter.

- dies alles wiederholt man bis kein Pfad mehr übrig ist.

man kann alle Hotels ignorieren, bei denen die Bewertung schlechter ist, als bei einer Route, die schon gefunden wurde, das schlechteste Hotel bewertet ist.

Umsetzung

Speichern der Werte

- Hotels
  - Um die Hotels zu speichern, hab ich eine simple Datenklasse geschrieben.
  - diese Klasse enthält die Werte:
    - Index (das wievielte Hotel es ist)
    - Bewertung
    - Fahrzeit zu dem Hotel vom Start aus

- Route
  - eine Route ist als eine Liste aller Hotels auf der Route gespeichert

- Routenoptionen
  - Die Routenoptionen sind in einer Liste gespeichert.
  - in dem Ersten ist eine Liste, mit dem Element start, was in dem Fall eine Instanz der Hotelklasse ist mit den Attributen:
    - index: -1
    - rating: 99999.0 (dass die minimale Bewertung nicht heruntergezogen werden kann)
    - travel time: 0
  - die nächsten 4 Elemente bestehen aus Listen, die aus erreichbaren Hotels bestehen.
    - diese sind jeweils absteigend nach der Bewertung geordnet.

Wichtigste Funktionen

"auffüllen der Routenoptionen"

- findet das erste Element in den Routenoptionen, das leer ist
- dann startet es eine while Schleife, die aufhört, wenn die Routenoptionen voll sind.
  - alle Hotels, die von dem ersten Hotel (das mit der besten Bewertung) der letzten vollen Optionen erreichbar ist.
  - wenn
    - von dort aus kein Hotel erreichbar ist, wird das letzte Element mit einer rekursiven Funktion auf die ich später eingehen werde entfernt.
      - wenn mehr als ein Element entfernt werden musste, wird zurückgesprungen
      - wenn Elemente bis zum start entfernt wurden, wird die while Schleife beendet, dann wurden alle Routen gefunden.
    - wenn von dort aus Hotels erreicht werden können, werden sie du den Optionen hinzugefügt, und es wird zu den nächsten Optionen gesprungen.

alle möglichen Hotels ausgehend eines vorherigen Hotels bekommen

- die Zeit, die das nächste Hotel mindestens haben muss, um das Ziel noch erreichen zu können wird berechnet.
  - `t - ((5 - s) * 360)`
  - 5 ist die Anzahl der Fahrten
  - s ist die Fahrt, bei der man am Hotel ankommt
  - 360 ist die Minutenanzahl, die eine Fahrt maximal brauchen darf
  - t ist die gesamte Fahrzeit in Minuten, die die Strecke braucht
- die Zeit, die das Hotel maximal brauchen darf, wird berechnet
  - `x + 360`
  - x ist die Zeit des Ausgangshotels
- eine for schleife, die durch die Hotels läuft startend bei einem Hotel nach dem Ausgangshotel wird gestartet
  - alle Hotels deren Bewertung schlechter ist als die größte minimale Bewertung einer schon gefundenen Route können ignoriert werden. Dies verbessert die Geschwindigkeit meines Algorithms sehr
  - wenn die minimale Zeit < Hotel < maximale Zeit, dann wird das Hotel zu einer Liste mit allen Hotels hinzugefügt.
  - wenn die Zeit des Hotels die maximale Zeit überschreitet wird die Schleife beendet.
- die Hotels werden absteigend nach ihrer Bewertung mithilfe des Insertionsorts sortiert.

ein Element der Optionen löschen

- diese Funktion ist rekursiv. Das heißt, sie ruft sich bis sie fertig ist immer wieder selber auf.
- von den Optionen unter dem gegebenen Index, wird das erste Element entfernt.
- wenn diese Option jetzt 0 ist
  - schaue ob es die 2te Option war
  - wenn nicht, rufe die Funktion nochmal auf, und gebe ihr den gleichen Index um eins verringert.
- sonst gebe den jetzigen Index zurück

Beispiele

hotels1.txt

- 2.7

```
stop: 0; hotel: 0002; time: 0347; rating: 2.7
stop: 1; hotel: 0006; time: 0687; rating: 4.4
stop: 2; hotel: 0007; time: 1007; rating: 2.8
stop: 3; hotel: 0010; time: 1360; rating: 2.8
```

hotels2.txt

- 2.3

```
Es gibt 2 Wege, die gleich gut sind.

Route 1:
```

```
stop: 0; hotel: 0002; time: 0341; rating: 2.3
stop: 1; hotel: 0009; time: 0700; rating: 3.0
stop: 2; hotel: 0014; time: 1053; rating: 4.8
stop: 3; hotel: 0024; time: 1380; rating: 5.0

Route 2:
stop: 0; hotel: 0002; time: 0341; rating: 2.3
stop: 1; hotel: 0009; time: 0700; rating: 3.0
stop: 2; hotel: 0013; time: 1051; rating: 2.3
stop: 3; hotel: 0024; time: 1380; rating: 5.0
```

hotels3.txt

```
Es wurde kein valider Pfad gefunden.
```

hotels4.txt

- 4.6

```
Es gibt 6 Wege, die gleich gut sind.

Route 1:
stop: 0; hotel: 0096; time: 0340; rating: 4.6
stop: 1; hotel: 0206; time: 0658; rating: 4.6
stop: 2; hotel: 0309; time: 0979; rating: 4.7
stop: 3; hotel: 0425; time: 1301; rating: 5.0

Route 2:
stop: 0; hotel: 0096; time: 0340; rating: 4.6
stop: 1; hotel: 0206; time: 0658; rating: 4.6
stop: 2; hotel: 0309; time: 0979; rating: 4.7
stop: 3; hotel: 0433; time: 1316; rating: 4.9

Route 3:
stop: 0; hotel: 0096; time: 0340; rating: 4.6
stop: 1; hotel: 0211; time: 0676; rating: 4.6
stop: 2; hotel: 0331; time: 1032; rating: 4.9
stop: 3; hotel: 0425; time: 1301; rating: 5.0

Route 4:
stop: 0; hotel: 0096; time: 0340; rating: 4.6
stop: 1; hotel: 0211; time: 0676; rating: 4.6
stop: 2; hotel: 0331; time: 1032; rating: 4.9
stop: 3; hotel: 0433; time: 1316; rating: 4.9

Route 5:
stop: 0; hotel: 0096; time: 0340; rating: 4.6
stop: 1; hotel: 0211; time: 0676; rating: 4.6
stop: 2; hotel: 0309; time: 0979; rating: 4.7
stop: 3; hotel: 0425; time: 1301; rating: 5.0

Route 6:
stop: 0; hotel: 0096; time: 0340; rating: 4.6
stop: 1; hotel: 0211; time: 0676; rating: 4.6
stop: 2; hotel: 0309; time: 0979; rating: 4.7
stop: 3; hotel: 0433; time: 1316; rating: 4.9
```

hotels5.txt

- 5.0

```
Es gibt 4 Wege, die gleich gut sind.

Route 1:
stop: 0; hotel: 0241; time: 0280; rating: 5.0
stop: 1; hotel: 0580; time: 0636; rating: 5.0
stop: 2; hotel: 0912; time: 0987; rating: 5.0
stop: 3; hotel: 1167; time: 1271; rating: 5.0

Route 2:
stop: 0; hotel: 0241; time: 0280; rating: 5.0
stop: 1; hotel: 0580; time: 0636; rating: 5.0
stop: 2; hotel: 0912; time: 0987; rating: 5.0
stop: 3; hotel: 1177; time: 1286; rating: 5.0

Route 3:
stop: 0; hotel: 0284; time: 0317; rating: 5.0
stop: 1; hotel: 0580; time: 0636; rating: 5.0
stop: 2; hotel: 0912; time: 0987; rating: 5.0
stop: 3; hotel: 1167; time: 1271; rating: 5.0

Route 4:
stop: 0; hotel: 0284; time: 0317; rating: 5.0
stop: 1; hotel: 0580; time: 0636; rating: 5.0
stop: 2; hotel: 0912; time: 0987; rating: 5.0
stop: 3; hotel: 1177; time: 1286; rating: 5.0
```

Quellcode

```
# Datenklasse für die Hotels
class Hotel:
    def __init__(self, index: int, rating: float, travel_time: int):
        self.index = index
        self.rating = rating
        self.travel_time = travel_time


# Klasse um die Hotellisten einzulesen und zu bearbeiten
class Vacation:
    def __init__(self, file_path: str):
        self.name = file_path
        with open(file_path, 'r') as hotel_file:
            raw_data = hotel_file.read().split("\n")

            # Gesamtfahrzeit
            self.total_time = int(raw_data[1])

            # liste von allen hotels (jeweils in der Datenklasse gespeichert)
            hotels_str = raw_data[2:-1]
            self.hotels = []
            for i, hotel_str in enumerate(hotels_str):
                time, rating = hotel_str.split(' ')
```

```
            self.hotels.append(Hotel(i, float(rating), int(time)))

        # Der start (zuhause)
        root = Hotel(-1, 99999, 0)

        # der Pfad
        # hier werden alle mögliche nächsten Hotels gespeichert von höchster Bewertung zu niedrigs
        # die Anzahl der Listen entspricht der Anzahl der Stoppe
        self.path = [
            [root],
            [],
            [],
            [],
            []
        ]

        self.STOPS = len(self.path)
        # 6h = 360min
        self.MAX_TIME_BEFORE_HOTEL = 360
        # die beste Bewertung eines Pfades
        # alle Hotels mit einer Bewertung unter diesem Wert werden ignoriert
        self.highest_rating = 0
        self.winning_paths = []

        # mit dem eigentlichen Algorithmus starten

        success = 0
        while success != -1:
            success = self.fill_path()
            while len(self.path[-1]) != 0:
                success = self.set_max_minimum()

        self.print_results()
        print("```")

    def print_results(self):
        # guard clauses um das Ergebnis richtig auszugeben

        print(f"**{self.name}**")

        if len(self.winning_paths) <= 0:
            # print(f"____________________{self.name}____________________")
            print("\n```\nEs wurde kein valider Pfad gefunden.")
            return

        # print(f"_____________{self.name} min:{self.highest_rating}____________________")
        print(f"- {self.highest_rating}\n```")

        if len(self.winning_paths) == 1:
            if len(self.winning_paths) > 0:
                self.print_path(self.winning_paths[0])
            return

        if len(self.winning_paths) > 1:
            print(f"Es gibt {len(self.winning_paths)} Wege, die gleich gut sind.")
            for i, path in enumerate(self.winning_paths):
                print(f"\nRoute {i + 1}:")
                self.print_path(path)


    @staticmethod
    def print_path(path: list):
```

```
    for stop, hotel in enumerate(path[1:]):
        print(f'stop: {stop}; hotel: {str(hotel.index).zfill(4)}; time: {str(hotel.travel_time

def shorten_solutions(self):
    solutions = {}

    for solution in self.winning_paths:
        solutions[solution[-2]] = solution

    self.winning_paths = solutions.values()

def fill_path(self):
    # füllt den pfad mit neuen optionen auf
    # finde das erste Element mit keinen Optionen
    for i, hotel_options in enumerate(self.path):
        if len(hotel_options) == 0:
            break

    if i == 0:
        return -1

    # wiederhole so lange bis alles aufgefüllt wurde
    while i < len(self.path):
        # fülle das nächste Element auf
        self.path[i] = self.get_all_possible_hotels(self.path[i-1][0], i-1)


        # wenn keine möglichen auffüllungen gefunden worden sind, gehe nicht zum nächsten elem
        # und lösche der erste eintrag bei dem Referenzelement
        if len(self.path[i]) == 0:
            i = self.remove_one_elem_from_path(i-1) + 1
            if i == 0:
                return -1
        else:
            i += 1

    return 1

def remove_one_elem_from_path(self, i: int):
    # entfernt ein element von den Pfadoptionen
    # wenn dies dann leer ist entfernt die Funktion rekursiv weitere Elemente,
    # bis das vorherige Element nicht leer ist.
    self.path[i] = self.path[i][1:]
    if len(self.path[i]) == 0:
        if i == 1:
            return -1

        return self.remove_one_elem_from_path(i-1)

    return i

def set_max_minimum(self):
    # speicher das neue minimum
    minimum_of_path = 99999
    current_path = []

    for element in self.path:
        current_path.append(element[0])
        if element[0].rating < minimum_of_path:
            minimum_of_path = element[0].rating

    # kontrolliert, ob man von dem letzten Hotel in 360 minuten zu dem Ziel kommt
```

```
            # (lediglich falls ich ein bug habe)
            path_works = True
            latest_stop = current_path[-1].travel_time
            if self.total_time - latest_stop > self.MAX_TIME_BEFORE_HOTEL:
                success = self.remove_one_elem_from_path(len(self.path) - 1)
                path_works = False

            if minimum_of_path > self.highest_rating:
                self.highest_rating = minimum_of_path
                self.winning_paths = [current_path]
            elif minimum_of_path == self.highest_rating:
                self.winning_paths.append(current_path)

            success = self.remove_one_elem_from_path(len(self.path)-1)
            if success == -1:
                return -1
            return 1

        def get_all_possible_hotels(self, root_hotel: Hotel, step: int):
            # diese Funktion gibt eine Liste alle Hotels zurück, die von einem Starthotel aus erreichb
            possible_hotels = []

            # berechnet den kleinsten Zeitschritt, dass die Zeit noch reichen kann.
            min_time = self.total_time - ((self.STOPS - (step + 1)) * self.MAX_TIME_BEFORE_HOTEL)
            max_time = root_hotel.travel_time + self.MAX_TIME_BEFORE_HOTEL


            # fügt jedes hotel, dessen Reisezeit innerhalb min_time und max_time ist
            # und dessen Bewertung höher gleich die beste pfadbewertung ist.
            for hotel in self.hotels[root_hotel.index + 1:]:
                if hotel.travel_time > max_time:
                    break
                if min_time < hotel.travel_time and hotel.rating >= self.highest_rating:
                    possible_hotels.append(hotel)

            # sortiere die Hotels nach der Bewertung
            possible_hotels = Vacation.sort_hotels(possible_hotels)

            return possible_hotels

        @staticmethod
        def sort_hotels(hotels: list):
            # sortiert eine Liste von Hotels absteigend nach ihrer Bewertung
            # nutzt insertion sort

            for hotel_index in range(1, len(hotels)):
                current_hotel = hotels[hotel_index]
                i = hotel_index - 1

                while i >= 0 and current_hotel.rating > hotels[i].rating:
                    hotels[i + 1] = hotels[i]
                    i -= 1

                hotels[i + 1] = current_hotel

            return hotels


    # Die Schleife lässt das Programm auf allen 5 Beispielen laufen
    for i in range(1, 6, 1):
        vacation = Vacation(f"hotels{i}.txt")
        print("\n")
```