Aufgabe 2: Vollgeladen

Team-ID: ?????

Team: Jason

Bearbeiter dieser Aufgabe:
Jason Mund

21. November 2021

Inhaltsverzeichnis

Aufgabenstellung

Zusammenfassung:

Mit Information der Anzahl der Hotels, wie die Ziel Entfernung soll berechnet werden, welche Hotels in 5 Tagen erreicht werden kann. Ziel ist es eine Liste von Hotels auszugeben, womit in 6 Stunden am Tag und in 5 Tagen das Ziel erreicht werden kann und das schlecht Bewertest Hotel best möglich bewertet ist.

Ausführlich: https://bwinf.de/fileadmin/bundeswettbewerb/40/Bundeswettbewerb-Aufgabenblatt.pdf (Aufgabe 2: „Vollgeladen“, Seite 1)

Lösungsidee

Um das Problem zu lösen, muss ich die Hotels in Tagen einteilen, und aus diesen Tagen die Hotels nehmen und untersuchen ob bei diesem Hotel am nächsten Tag unter in 6 Std erreichbar ist und ob die niedrigste Bewertung so hoch wie möglich ist.

Der Weg mit der Hotelkompilation mit dem höchsten niedrigsten Bewertung ist der **gesuchte Weg**.

Umsetzung

Die Umsetzung erfolgt in Python.

Wir sortieren die Hotels in verschiedenen Tagen, in denn Tagen wo man hin muss um in 5Tagen am Ziel zu sein.

Danach nehmen wir ein Hotel von einem Tag und schauen ob man in 6 Std am nächsten Tag ein Hotel erreicht, wenn ja schreiben wir uns dieses Hotel auf, wir machen es solange bis wir beim 5 Tag angekommen sind. Und schauen ob die niedrigste Bewertung die höchstmögliche ist.

Dieses Prozedur machen wir für alle Kombinationen.

Am Ende haben wir die Kombination wo das Hotel mit dem niedrigsten Hotel am höchsten ist.

Zusammenfassung

Um die Lösungsidee in Python umsetzen zu können müssen einige Vorbereitungen getroffen werden.

1. Der Input muss gelesen werden können. Zeitgleich werden die doppelten Hotels aussortiert.
2. Die Hotels werden in Tagen aufgeteilt.
3. Hotel Distance von ein Tag wird verglichen und ob die Bewertung höher als das bislangste schächste Hotel ist und ob ein Hotel am nächsten Tag erreichbar ist.
4. Es muss eine Ausgabe erstellt werden.

1. Input einlesen:

Der Nutzer wird mit der Standardmethode input() aufgefordert den (vollständigen) Pfad zur Input-Textdatei anzugeben.

```
Geben Sie den Pfad an, wo ihre Datei sich befindet: hotels1.txt
```

Dann wird ein String aus der Text-Datei mit dem angegebenen Pfad abgespeichert. Mit der Funktion „open“ wird jede Zeile in einer Liste eingespeichert. Um die Information zu nutzen wird eine for each schleife genutzt welche an der 2 Stelle gestartet wird. Dieser String wird mit split(„ “) aufgeteilt. Es wird geschaut ob es 2 Elemente entstehen, wenn ja wird das erste Element wie auch das 2 in einem HotelObject eingespeichert, dieses wird in der Hotelsiste eingespeichert.

Wenn es nur 1 Element ist wird dieses Element in der variable „goalDistance“ gespeichert

Script:

```python
#die Informationen von der Text Datei werden eingelesen und eingeordnet
def getInformation(path: str):
    global hotels
    global goalDistance
    print(path)
    with open(path, "r") as rawInformation:
        contens = rawInformation.readlines()
        print(type(contens), contens)

    for i in contens[1:]:
        information = i.split(" ")
        if len(information) > 1:
            hotels.append(Hotel(information[0], information[1]))
            removeDuplication(hotels)
        else:
            checkPossibleDistance(int(information[0])) #checkt ob überhaupt die ZielDistance generell in denn 5 Tagen erreichbar ist.
            goalDistance = int(information[0])
```

2. Hotels werden in Tagen eingeteilt:

Die Hotelliste wird vertauscht. Der Defaultwert von der Variable „dayDistance“ wird mit der goalDistance erstellt. Als nächstes wird eine for Schleife ausgeführt. Es wird eine neue Liste erstellt. Die ZielDistance wird mit 360 subtrahiert. Eine weitere For each schleife wird ausgeführt welche die Stellen von der Hotelliste benutzen. Es wird geschaut ob die Distance von dem Hotel größer oder gleich der aktuellen TagesDistance. Wenn es größer ist wird es in der Tagesliste eingespeichert, wenn nicht wird diese Schleife abgebrochen. Die Hotels welche in der Tagesliste eingespeichert werden, werden in der Hotelliste gelöscht. Am Ende wird die Tagesliste in einer Liste gespeichert.

Script:

```python
def declarationDays():
    global hotels
    global goalDistance
    days: list = []
    dayDistance = goalDistance
    hotels.reverse()
    for i in range(4):
        oneDay: list = []
        dayDistance -= 360 #für jeden schleifendurchlauf(ein durchlauf = ein Tag) wird die mindeste erreichbare Tagesdistance berechnet
        for y in hotels:
            if int(y.distance) >= dayDistance: #Untersuchung ob das Hotel größer ist als die mindest erreichbate Distance ist
                oneDay.append(y)
            else:
                break
        #Hotels welche in einem Tag drinne sind werden von der Hotelsliste entfernt
        for z in oneDay:
            hotels.remove(z)
        days.append(oneDay)
        """for z in oneDay:
            print(z, " ", end="")
        print()"""
    declarationHotelDay(days)
```

3. Tageshotel wird verglichen und ausgewertet:

Die Tagesliste wird wieder vertauscht.

Es wird eine Schleife ausgeführt wo eine Liste in der Tagesliste genommen wird und die Hotels genommen wird. Es wird geschaut ob die Hotel Distance kleiner/gleich der maximalen erreichbaren Tagesdistance(beim ersten Durchlauf ist es 360) ist und ob die Bewertung diesen Hotels größer ist als das schlechtbewerteste Hotel(Am Anfang 0) in der Result Klasse.

Wenn nicht wird die Funktion abgebrochen mit dem Rückgabe Wert der Result Klasse,

Wenn alles wahr ist wird geschaut ob es der letzte Tag ist, wenn ja wird das Hotel in einer seperaten Liste gespeichert, dann wird die seperate Liste zur referenzliste in der Result Klasse, Es wird geschaut welche Bewertung das schlecht bewerteste Hotel besitzt, dies wird auch in der Result Klasse gespeichert. Am Ende wird dieses aktuelle Hotel von der seperatenListe entfernt.

Wenn irgendein Tag ist, wird das Hotel in der seperaten Liste gespeichert, und ein neuer Tag wird eingeleitet so, das die Funktion erneut aufgerufen wird, wobei der Tag und die TagesDistance erhöht wird.

Am Ende ist die Result Klasse das Ergebnis.

Script:

```python
#durchsuchung von denn Hotels und ob der niedrigste Rank von Result(ausgewählte Hotel von Tagen zusammengefasst) am höchsten ist.
def declarationday1to42(days: list, result: Result, goalDistance: int = 360, possibelWay: list=[], day= 0):
    #schleifen durchlauf von einen Tag wo bei jedem durchlauf ein Hotel genommen wird von dem Tag, beginn von nahsten Hotel bis entferntesten Hotel
    for hotel in days[day]:
        #überprüfung ob die Hotel Distance kleiner odr = der maximalen erreichbaren Distance diesen Tages ist und ob überhaupt die bewertung höher ist
        #Wenn nicht wird ein neuer Durchlauf stattfinden.
        if hotel.distance <= goalDistance and result.weakestHotel <= hotel.rank:
                #es wird geschaut ob es der letzte Tag ist, wenn ja wird das Hotel in der seperaten referenzliste gespeichert und diese Liste wird als
                #und das schlecht Bewerteste Hotel wird herausgefunden, am ende wird geschaut für die weiteren Hotels diesen Tages ob sie eine bessere
                if day == 3:
                    possibelWay.append(hotel)
                    result.hotels = possibelWay[:]
                    result.weakestHotel = min(possibelWay).rank
                    possibelWay.pop(day)
                #Wenn es irgendein Hotel ist werden sie eingefügt in der seperatenreferenzliste, und die funktion wird erneut aufgerufen mit dem nächst
                else:
                    possibelWay.append(hotel)
                    declarationday1to42(days, result, hotel.distance + 360, possibelWay, day + 1)
                    possibelWay.pop(-1)
    return result
```

3.Ausgabe:

Die Resultklasse wird die referenzliste genommen und für jeden Tag wird die Stelle genommen und so die Distance wie auch die Bewertung ausgegeben.

Script:

```python
#Ausgabe
def evaluation(result: Result):
    for i in range(4):
        print(f"Tag {i+1}: {result.hotels[i].distance} mit einem Rank von {result.hotels[i].rank}")
```

Beispiele

BWINF hotels1.txt (https://bwinf.de/fileadmin/user_upload/hotels1.txt)

```
Tag 1: 360 mit einem Rank von 1.3
Tag 2: 717 mit einem Rank von 0.3
Tag 3: 1076 mit einem Rank von 3.8
Tag 4: 1433 mit einem Rank von 1.7
```

BWINF hotels2.txt (https://bwinf.de/fileadmin/user_upload/hotels2.txt)

```
Tag 1: 341 mit einem Rank von 2.3
Tag 2: 700 mit einem Rank von 3.0
Tag 3: 1053 mit einem Rank von 4.8
Tag 4: 1380 mit einem Rank von 5.0
```

BWINF hotels3.txt (https://bwinf.de/fileadmin/user_upload/hotels3.txt)

```
Tag 1: 360 mit einem Rank von 1.3
Tag 2: 717 mit einem Rank von 0.3
Tag 3: 1076 mit einem Rank von 3.8
Tag 4: 1433 mit einem Rank von 1.7
```

BWINF hotels4.txt (https://bwinf.de/fileadmin/user_upload/hotels4.txt)

```
Tag 1: 340 mit einem Rank von 4.6
Tag 2: 676 mit einem Rank von 4.6
Tag 3: 1032 mit einem Rank von 4.9
Tag 4: 1316 mit einem Rank von 4.9
```

BWINF hotels5.txt (https://bwinf.de/fileadmin/user_upload/hotels5.txt)

```
Tag 1: 317 mit einem Rank von 5.0
Tag 2: 636 mit einem Rank von 5.0
Tag 3: 987 mit einem Rank von 5.0
Tag 4: 1286 mit einem Rank von 5.0
```

Quellcode

```
import os, time

#Hotel Klasse, speichert die Attrituben von einem Hotel: Attribute = distance wie auch die Bewertung
class Hotel:
    def __init__(self, distance, rank):
        self.distance: int = int(distance)
        self.rank: float = float(rank)

    def __lt__(self, other):
        return self.rank < other

    def __gt__(self, other):
        return self.rank > other

#speichert eine Referenzliste, wie auch von der Referenzliste das Hotel mit der niedrigsten Bewertung.
class Result:
    def __init__(self,):
        self.hotels: list = []
        self.weakestHotel: float = float(0.0)

    def __lt__(self, other):
        return self.weakestHotel < other

    def __gt__(self, other):
        return self.weakestHotel > other

#liest denn vorgegebenen Input aus
# -> bevor die Information aus dem Input rausgenommen werden wird mit der funktion checkPathExist überprüft ob der
Pfad exestiert.
def getInput():
    path = checkPathExist(input("Geben Sie den Pfad an, wo ihre Datei sich befindet: "))
    #path = "hotels7.txt"
    getInformation(checkPathExist(path))



def checkPathExist(path: str):
    if os.path.exists(path):
        return path
    else:
        print("Dieser Pfad existiert nicht, bitte geben sie ein gültigen Pfad ein.")
        getInput()
```

```
#die Informationen von der Text Datei werden eingelesen und eingeordnet
def getInformation(path: str):
    global hotels
    global goalDistance
    print(path)
    with open(path, "r") as rawInformation:
        contens = rawInformation.readlines()
        print(type(contens), contens)

    for i in contens[1:]:
        information = i.split(" ")
        if len(information) > 1:
            hotels.append(Hotel(information[0], information[1]))
            removeDuplication(hotels)
        else:
            checkPossibleDistance(int(information[0])) #checkt ob überhaupt die ZielDistance generell in denn 5 Tagen
erreichbar ist.
            goalDistance = int(information[0])
    print("Hotelliste: ", hotels)
#Idenfizieren ob das letzte und das vorletzte Hotel die gleiche Distance haben, wenn ja wird das Hotel mit der besten
Bewertung genommen.
def removeDuplication(hotels):
    try:
        if hotels[-1].distance == hotels[-2].distance:
            if hotels[-1].rank <= hotels[-2].rank:
                hotels.pop(-1)
            else:
                hotels.pop(-2)
    except IndexError:
        pass



def checkPossibleDistance(goalDistance: int):
    if goalDistance <= 1800:
        return
    else:
        print("Dieser Weg, ist mit den vorgegebenen Zielen, nicht erreichbar!")
        getInput()

#in der Hotelliste wird von dem entfernsteten Hotel bis zum nahsten Hotel geschaut und in Tagen eingeteilt.
#Es für jeden durchlauf ein neue Tagesliste erstellt, Es wird geschaut ob die Distance von Hotel kleiner als die minimale
erreichbare Distanz ist,
#Wenn ja wird sie in der Tagesliste eingeführt am Ende werden die Tageslisten in einer zusammengefasste Tagesliste
eingepeichert
def declarationDays():
    global hotels
    global goalDistance
    days: list = []
    dayDistance = goalDistance
    hotels.reverse()
    for i in range(4):
        oneDay: list = []
        dayDistance -= 360 #für jeden schleifendurchlauf(ein durchlauf = ein Tag) wird die mindeste erreichbare
Tagesdistance berechnet
        for y in hotels:
```

```
        if int(y.distance) >= dayDistance: #Untersuchung ob das Hotel größer ist als die mindest erreichbate Distance ist
            oneDay.append(y)
        else:
            break
    #Hotels welche in einem Tag drinne sind werden von der Hotelsliste entfernt
    for z in oneDay:
        hotels.remove(z)
    days.append(oneDay)
    """for z in oneDay:
        print(z, " ", end="")
    print()"""
  declarationHotelDay(days)

#verschachtete Tagesliste werden vertauscht, es fängt von Tag 1(index = 0) bis maximal Tag4(index = 3) am Ende an.
def declarationHotelDay(days: []):
  global goalDistance
  days.reverse()
  for i in days:
    i.reverse()
  result: Result = declarationday1to42(days, Result())
  evaluation(result)



#durchsuchung von denn Hotels und ob der niedrigste Rank von Result(ausgewählte Hotel von Tagen zusammengefasst) am höchsten ist.
def declarationday1to42(days: list, result: Result, goalDistance: int = 360, possibelWay: list=[], day= 0):
  #schleifen durchlauf von einen Tag wo bei jedem durchlauf ein Hotel genommen wird von dem Tag, beginn von nahsten Hotel bis entferntesten Hotel
  for hotel in days[day]:
    #überprüfung ob die Hotel Distance kleiner odr = der maximalen erreichbaren Distance diesen Tages ist und ob überhaupt die bewertung höher ist als von dem schwächsten Hotel, von der Referenzliste.
    #Wenn nicht wird ein neuer Durchlauf stattfinden.
    if hotel.distance <= goalDistance and result.weakestHotel <= hotel.rank:
        #es wird geschaut ob es der letzte Tag ist, wenn ja wird das Hotel in der seperaten referenzliste gespeichert und diese Liste wird als Referenzliste gespeichert
        #und das schlecht Bewerteste Hotel wird herausgefunden, am ende wird geschaut für die weiteren Hotels diesen Tages ob sie eine bessere Bewertung besitzen.
        if day == 3:
            possibelWay.append(hotel)
            result.hotels = possibelWay[:]
            result.weakestHotel = min(possibelWay).rank
            possibelWay.pop(day)
        #Wenn es irgendein Hotel ist werden sie eingefügt in der seperatenreferenzliste, und die funktion wird erneut aufgerufen mit dem nächsten Tag.
        else:
            possibelWay.append(hotel)
            declarationday1to42(days, result, hotel.distance + 360, possibelWay, day + 1)
            possibelWay.pop(-1)
  return result

#Ausgabe
def evaluation(result: Result):
  for i in range(4):
    print(f"Tag {i+1}: {result.hotels[i].distance} mit einem Rank von {result.hotels[i].rank}")

if __name__ == "__main__":
  hotels: list = []
```

```
goalDistance: int = 0

starttime = time.time()
getInput()

declarationDays()
endtime = time.time() - starttime
print(endtime)
```