Aufgabe 1: Schiebeparkplatz

20. November 2021

Team-ID: 00821
Team: IaC 0.9524
Bearbeiter dieser Aufgabe: Paul Buda

Lösungsidee:

Zuerst muss geprüft werden, ob vor dem normal parkendem Auto ein querstehendes Auto steht, wenn dass der Fall ist wird zuerst geprüft ob das Auto nach rechts und danach nach links weggeschoben werden kann. Dabei muss aber beachtet werden das beim Querschieben dann andere Autos im Weg stehen können, welche dann auch weggeschoben werden müssen, dies ist jedoch nur so lange möglich, wie die Wand rechts bzw. links nicht im Weg ist.
Um die Ausfahrtzeit optimieren zu können, kann man mitzählen wie viele Autos wie weit bewegt werden müssen um später zu entscheiden, ob es sinnvoller ist die querstehenden Autos nach rechts oder links wegzuschieben.

Umsetzung:

Mittels zwei For-Schleifen werden zuerst alle Autos aus der Datei ausgelesen und dann für jedes Auto überprüft ob ein Auto quer davor steht und wenn ja, ob dieses querstehende Auto ein, oder zwei Parkplätze nach rechts bzw. links verschoben werden muss.
Wenn ein querstehendes Auto verschoben werden muss, wird jetzt für dieses geprüft, wie weit es nach rechts geschoben werden kann, bevor ein anderes querstehendes Auto im Weg steht, wenn das Auto nicht weit genug nach rechts fahren kann wird versucht das nächste Auto quer zu verschieben. Dies wird so lange wiederholt bis das Auto welches das andere Auto zuparkt weit genug weggeschoben werden kann, oder die rechte Wand im Weg ist.
Dieser Vorgang wird dann auch mit der linken Seite wiederholt. Es wird jeweils mit gezählt wie viele Auto wie weit in welche Richtung fahren können um dann im nächsten Schritt mittels einer Fallunterscheidung entscheiden zu können, wie man die Autos am besten verschieben muss um die Ausfahrtzeit zu optimieren. Dabei wird zuerst nach der Anzahlt der zu verschiebenden Autos und dann nach dem nach der Anzahl der Parkplätze welche die Autos insgesamt wegschoben werden müssen entscheiden, falls man die Autos sowohl nach rechts, als auch nach links wegschieben könnte.

Beispiele:

Bei Beispielparkplatz 0 erzeugt das Programm folgende Ausgabe:

```
A:
B:
C: H 1 rechts
D: H 1 links
E:
F: I 2 links, H 1 links
G: I 1 links
```

Die Autos A, B und E können somit einfach herausfahren, um die restlichen Autos ausparken zu können müssen die querstehenden Autos wie angegeben verschoben werden.
Bei Auto C sieht man, dass H eins nach rechts und nicht zwei nach links verschoben werden soll, da immer die schnellste Möglichkeit ausgegeben wird.
Bei Beispielparkplatz 1 erzeugt das Programm folgende Ausgabe:

```
A:
B: O 1 rechts, P 1 rechts
C: O 1 links
D: P 1 rechts
E: P 1 links, O 1 links
F:
G: Q 1 rechts
H: Q 1 links
I:
J:
```

Die Autos A, F , I und J können somit einfach herausfahren, um die restlichen Autos ausparken zu können müssen die querstehenden Autos wie angegeben verschoben werden.
Bei Beispielparkplatz 5 erzeugt das Programm folgende Ausgabe:

```
A:
B:
C: P 2 links
D: P 1 links
E: Q 1 rechts
F: Q 2 rechts
G:
H:
I: R 1 rechts
J: R 1 links
K:
L:
M: S 1 rechts
N: S 1 links
O:
```

Die Autos A, B , G, H, K, L und O können somit einfach herausfahren, um die restlichen Autos ausparken zu können müssen die querstehenden Autos wie angegeben verschoben werden.

Eigens erfundenes Beispiel:

```
A E
2
F 0
G 2
```

Ausgabe des Programms:

```
A: F 1 rechts, G 1 rechts
B: kann nicht ausparken
C: G 1 rechts
```

```
D: kann nicht ausparken
E:
```

Das Programm gibt nun aus, dass die Autos B und D nicht ausparken können.
Quelltext:

```
#liest den Parkplatz aus
parkingSpot = open('parkplatz.txt','r')
#zählt die normal parkenden Autos
numberCars = ord(parkingSpot.readline()[2]) - 64

#ließt alle queer parkende Autos ein, die linke Wand ist das Auto mit der Position
-2 und die rechts Wand das Auto mit der Position <numberCars>
acrossCars = [-2]
for r in range(int(parkingSpot.readline())):
    acrossCars.append(int(parkingSpot.readline()[2:]))
parkingSpot.close
acrossCars.append(numberCars)

#überprüft für jedes Auto, ob ein queerstehendes Auto davor steht, und wie weit
dieses Auto nach rechts bzw. links fahren muss
#und speicher die Nummer des Querstehenden Autos
for carNumber in range(numberCars):
    if(carNumber in acrossCars):
        toRight, toLeft = 1, 2
        acrossNumber = acrossCars.index(carNumber)
    elif(carNumber - 1 in acrossCars):
        toRight, toLeft = 2, 1
        acrossNumber = acrossCars.index(carNumber - 1)
    #wenn vor dem Auto kein Anderes quer steht, so wird ausgegben, dass kein Auto
bewegt werden muss
    else:
        print(chr(carNumber + 65) + ":")
        continue

    #gibt an, ob die Autos nach rechts bzw. links Platz machen können
    canRight = canLeft = True

    #zählt wie viele Autos nach rechts fahren müssen
    rightNeeded = 0

    #zählt wie weit alle Autos zusammen nach rechts fahren müssen
    rightSteps = 0

    #speichert alle benötigten Autos und wie weit diese fahren müssen
    movedCarsRight = []
    movedCarsLeft = []

    #überprüft ob die Autos nach rechts Platz machen können
    while(toRight > 0):
        if(acrossNumber > len(acrossCars) - 2):
            canRight = False
            break
        rightSteps += toRight
        rightNeeded += 1
```

```
        movedCarsRight.append([acrossNumber, toRight])
        toRight -= acrossCars[acrossNumber + 1] - acrossCars[acrossNumber] - 2
        acrossNumber += 1

    #zurücksetzten der acrossNumber
    acrossNumber -= rightNeeded

    #überprüft ob die Autos nach links Platz machen können
    while(toLeft > 0):
        if(acrossNumber - 1 < 0):
            canLeft = False
            break
        rightSteps -= toLeft
        rightNeeded -= 1
        movedCarsLeft.append([acrossNumber, toLeft])
        toLeft -= acrossCars[acrossNumber] - acrossCars[acrossNumber - 1] - 2
        acrossNumber -= 1

    #überprüft, welcher Weg schneller ist, rechts oder links
    #wenn die Autos weder nach links noch nach rechts Platz machen können wird
ausgegeben, dass das Auto nicht ausparken kann
    if(canRight + canLeft == 0):
        print(chr(carNumber + 65) + ": kann nicht ausparken")
        continue
    elif(canRight + canLeft == 1):
        driveRight = canRight
    else:
        if(rightNeeded != 0):
            driveRight = True if(rightNeeded < 0) else False
        else:
            driveRight = True if(rightSteps < 0) else False

    #gibt den schnellsten Weg aus
    if(driveRight):
        print(chr(carNumber + 65) + ": ", end = "")
        [print(chr(car[0] + 64 + numberCars) + " " + str(car[1]) + " rechts", end =
", " if(car != movedCarsRight[-1]) else "\n") for car in movedCarsRight]
    else:
        print(chr(carNumber + 65) + ": ", end = "")
        [print(chr(car[0] + 64 + numberCars) + " " + str(car[1]) + " links", end =
", " if(car != movedCarsLeft[-1]) else "\n") for car in movedCarsLeft]
```