**Aufgabe 1: Schiebeparkplatz**

Team-ID: 00450 Team: Die Lamas

Bearbeiter/-innen dieser Aufgabe:

Philipp Tiede

1. November 2021

**Inhaltsverzeichnis**

[Lösungsidee 1](#_Toc88418357)

[Umsetzung 1](#_Toc88418358)

[Beispiele 1](#_Toc88418359)

[Quellcode 3](#_Toc88418360)

# Lösungsidee

Damit in dieser Aufgabe die Autos richtig verschoben werden. Teste ich für jedes Auto, das blockiert ist wie das Auto, das das blockierte Auto blockiert aus dem Weg verschoben werden kann.

Dafür werden alle Bewegungen ausprobiert, die dieses blockierend Auto ausführen, um das blockierte Auto zu befreien. Jedes Auto, mit dem es „kollidiert“ wird auch versucht aus dem Weg zu fahren. Dies wird sowohl fürs nach links und rechts ausweichen in eine Lösung gespeichert und die kürzere der beiden Lösungen wird zurückgegeben.

# Umsetzung

Um dies zu schaffen, gucke ich für jedes Auto erst, ob es von der linken oder rechten Seite des anderen Autos blockiert wird. Wenn das Auto dann direkt nach links oder rechts ausweichen kann, ohne mit anderen Autos zu kollidieren, dann weicht er in die Richtung aus, die kürzer ist.

Ansonsten, wenn das Auto in beide Richtungen von anderen Autos blockiert wird, versucht das Programm das Auto nach links und nach rechts zu bewegen und dabei andere Autos zu verschieben. Dies wird in einer rekursiven Funktion getan. Diese ruft sich immer selbst auf, um andere Autos zu verschieben, die Autos den weg versperren. Wenn das Auto die Wand trifft gibt das Programm einen Fehler in die Lösung, da diese Seite komplett blockiert ist.

Es wird dann zuletzt Überprüft, welche der Lösungen für beide Richtungen kürzer ist und dann diese für das blockierte Auto ausgegeben.

# Beispiele

> python main.py parkplatz1.txt

A:

B:

C: H 1 rechts

D: H 1 links

E:

F: H 1 links; I 2 links

G: I 1 links

> python main.py parkplatz2.txt

A:

B:

C: O 1 rechts

D: O 1 links

E:

F: O 2 links; P 2 links

G: P 1 links

H: R 1 rechts; Q 1 rechts

I: P 1 links; Q 1 links

J: R 1 rechts

K: P 1 links; Q 1 links; R 1 links

L:

M: O 2 links; P 2 links; Q 2 links; R 2 links; S 2 links

N: S 1 links

> python main.py parkplatz3.txt

A:

B: O 1 rechts

C: O 1 links

D:

E: P 1 rechts

F: P 1 links

G:

H:

I: Q 2 links

J: Q 1 links

K: Q 2 links; R 2 links

L: Q 1 links; R 1 links

M: Q 2 links; R 2 links; S 2 links

N: Q 1 links; R 1 links; S 1 links

> python main.py parkplatz4.txt

A: R 1 rechts; Q 1 rechts

B: R 2 rechts; Q 2 rechts

C: R 1 rechts

D: R 2 rechts

E:

F:

G: S 1 rechts

H: S 1 links

I:

J:

K: T 1 rechts

L: T 1 links

M:

N: U 1 rechts

O: U 1 links

P:

> python main.py parkplatz5.txt

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:

# Quellcode

**class** **carDiagonal**:

*"""*

*Klasse für die quer stehenden Autos*

*"""*

name:int

blocking:List[int]

@blocking.setter

**def** blocking(self, value:int) -> None:

self.\_blocking = [value, value+1]

**class** **carStraight**:

*"""*

*Klasse für die gerade stehenden Autos*

*"""*

name:int

blockedBy:carDiagonal

**def** createParkingLot(data) -> Tuple[List[carStraight], List[carDiagonal]]:

*"""*

*Erstellt den Parkplatzt, sodass er vom*

*Program verstanden werden kann.*

*"""*

parkingRow:List[carStraight] = []

blockingRow:List[carDiagonal] = []

i:int = ord(data[0])

*# Parkreihe füllen*

**while** i <= ord(data[2]):

parkingRow.append(carStraight(i - ord("A"), None))

i += 1

fillBlock(data, blockingRow)

*# Für parkende Autos einstellen, wer sie blockiert*

**for** car **in** blockingRow:

**for** block **in** car.blocking:

blockCar = parkingRow[block]

blockCar.blockedBy = car

*# Prüft, ob die Datei richtig gelesen wurde*

**assert** len(blockingRow) == int(data.splitlines()[1])

**return** parkingRow, blockingRow

**def** fillBlock(data, blockingRow):

*"""Blockierende Reihe füllen"""*

i = 2 *# beginnt bei der 3. Reihe*

**while** i < len(data.splitlines()):

line = data.splitlines()[i]

position = int(line[2:len(line)])

blocking=position

blockingRow.append(carDiagonal(ord(line[0]) - ord("A"), blocking))

i += 1

**def** canMove(blockingRow:List[carDiagonal], car:carDiagonal, size:int, amount:int):

*"""*

*Gibt aus, ob das Blockierende Auto die Angegebene Menge an Plätzen*

*bewegt werden kann. (Positive Menge nach rechts, negativ nach links)*

*"""*

**if** amount == 0: *# Das Auto kann immer stehen bleiben*

**return** True

**if** **not** canMove(blockingRow, car, size, amount - (amount/abs(amount))):

*# Gucken, ob das Auto zu der Position einen*

*# näher an der Startposition gehen kann*

**return** canMove(blockingRow, car, size, amount - (amount/abs(amount)))

**if** car.blocking[0] + amount < 0 **or** \

car.blocking[1] + amount >= size: *# Fährt das Auto über den Rand?*

**return** False

**for** cars **in** blockingRow: *# Es wird probiert ob mit einem der Autos kollidiert*

**if** ((cars.blocking[0] == car.blocking[0] + amount **and** cars != car) **or** \

(cars.blocking[1] == car.blocking[0] + amount **and** cars != car) **or** \

(cars.blocking[0] == car.blocking[1] + amount **and** cars != car) **or** \

(cars.blocking[1] == car.blocking[1] + amount **and** cars != car)):

**return** cars

**return** True

**def** move(car, dir, blockingRow, size):

*"""*

*Führt die Bewegung aus und gibt das Ergebnis als String zurück. \n*

*Diese Funktion läuft rekursiv.*

*"""*

movement = canMove(blockingRow, car, size, dir[0])

**if** movement == True:

**return** ""

**elif** movement == False:

**return** "Error"

**else**:

**return** str(move(movement, [dir[0], dir[1]], blockingRow, size)) + toChr(movement.name) + " " + str(abs(dir[0])) + " " + dir[1] + "; "

**def** main() -> None:

*"""*

*main*

*"""*

dataFileLocation = sys.argv[1] *#*

dataFile = open(dataFileLocation, "r") *# Datei einlesen und in data als String*

data = dataFile.read() *# speichern.*

parkingRow, blockingRow = createParkingLot(data) *# Aus den daten den Parkplatz "erstellen"*

solution = ""

**for** car **in** parkingRow:

solution += toChr(car.name) + ": "

**if** car.blockedBy:

*#solution += toChr(car.blockedBy.name)*

**if** car.blockedBy.blocking[0] == car.name:

dir = [[1, "rechts"], [-2, "links"]]

**else**:

dir = [[-1, "links"], [2, "rechts"]]

movement0 = canMove(blockingRow, car.blockedBy, len(parkingRow), dir[0][0])

movement1 = canMove(blockingRow, car.blockedBy, len(parkingRow), dir[1][0])

**if** movement0 == True:

solution += toChr(car.blockedBy.name) + " 1 " + dir[0][1]

**elif** movement1 == True:

solution += toChr(car.blockedBy.name) + " 2 " + dir[1][1]

**elif** movement0 != False **and** movement1 != False:

solution1 = str(move(car.blockedBy, dir[0], blockingRow, len(parkingRow))) + toChr(car.blockedBy.name) + " 1 " + dir[0][1]

solution2 = str(move(car.blockedBy, dir[1], blockingRow, len(parkingRow))) + toChr(car.blockedBy.name) + " 2 " + dir[1][1]

solution += solution1 **if** (len(solution1) <= len(solution2) **and** **not** "Error" **in** solution1) **or** "Error" **in** solution2 **else** solution2

**elif** movement0 != False:

solution += str(move(car.blockedBy, dir[0], blockingRow, len(parkingRow))) + toChr(car.blockedBy.name) + " 1 " + dir[0][1]

**elif** movement1 != False:

solution += str(move(car.blockedBy, dir[1], blockingRow, len(parkingRow))) + toChr(car.blockedBy.name) + " 2 " + dir[1][1]

**else**:

solution += toChr(car.blockedBy.name) + " Der Wagen kann nicht so bewegt werden, dass das Auto befreit werden kann"

solution += "**\n**"

**print**(solution)