Aufgabe 1: Schiebeparkplatz

Team-ID: ?????

Team: Teamname (oder dein Name)

Bearbeiter/-innen dieser Aufgabe:  
Benedikt Biedermann, Kevin Bah, Leonie-Sophie Ilyuk, Nils Kettler, Ole Kettler, Dominik Pfeiffer, Shalina Rohdenburg, Jule Schlifke, Rebekka Schmidt

16. November 2021

[Lösungsidee 1](#_Toc88422163)

[Umsetzung 2](#_Toc88422164)

[Beispiele 4](#_Toc88422165)

[Quellcode 4](#_Toc88422166)

Lösungsidee

Mithilfe der Beispieldaten wird ein Parkplatz erstellt, der aus zwei Reihen besteht. In der oberen Reihe stehen die Autos, die ausgeparkt werden sollen, in der unteren befinden sich die Autos, die verschoben werden können, um den Weg freizugeben.

Zu Beginn wird sich jedes Auto nacheinander angeschaut und geprüft, ob dies bereits ausparken kann, oder ob eines der Schiebeautos verschoben werden muss. Je nachdem, ob das Auto von dem vorderen oder hinteren Teil des Schiebeautos blockiert wird, wird entschieden, in welche Richtung (rechts oder links) das Schiebeauto zunächst verschoben werden soll, um den geringsten Aufwand zu haben.

Bevor das Schiebeauto tatsächlich bewegt wird, wird geprüft, ob es überhaupt in beide Richtungen genug Platz dafür gibt. Für den Fall, dass es sich um ein Schiebeauto handelt, das am Rand des Parkplatzes steht, gibt es dann nur noch die Möglichkeit, es in eine Richtung zu verschieben. Steht ein anderes Schiebeauto daneben, das den Weg blockiert, wird dies zuerst verschoben und danach folgt das Schiebeauto, das den Weg für das ausparkende Auto blockiert. Dieser Vorgang geschieht so lange, bis der Weg für das Auto freigemacht wurde. Falls neben dem Schiebeauto bereits genügend Platz ist, wird es auf kürzestem Wege verschoben, sodass das Auto direkt ausparken kann.

Für den Fall, dass während dem Schiebevorgang eine Grenze erreicht wird, wird die Schieberichtung verändert und es wird versucht, die Autos nun auf der anderen Seite zu bewegen.

Sobald ein Auto frei ist, entsteht eine Ausgabe, die angibt, welche Autos wie weit und in welche Richtung verschoben werden müssen, um den Weg für das ausparkende Auto freizugeben.

Umsetzung

**Modellierung des Parkplatzes**

Um einen Parkplatz den vorgegebenen Beispieldaten entsprechend zu modellieren, wird jedes auszuparkende Auto einer zweidimensionalen Liste hinzugefügt. Jede Parklücke enthält den Buchstaben des Autos, das dort parkt und einem weiteren Wert, der angibt, ob das Auto blockiert wird und wenn ja, von welchem Schiebeauto es blockiert wird.

**(***Abbildung des Parkplatzes)*

**Status-Abfrage der Autos**

Mithilfe eines for-Loops wird jede Parklücke einzeln angeschaut und geprüft, ob das dort stehende Auto bereits ausparken kann oder nicht. Falls ja, wird direkt ausgegeben, dass das Auto frei ist und der Loop geht weiter mit dem nächsten Auto. Wenn dies von einem Schiebeauto blockiert wird, folgt eine Überprüfung, ob es sich um ein Randauto oder ein Auto in der Mitte handelt. Daraus kann darauf geschlossen werden, ob die linke oder die rechte Seite des Autos im Weg steht. Bei den Autos in der Mitte des Parkplatzes wird dies mithilfe einer weiteren if/elif-Abfrage herausgefunden. Als Fixpunkt der Schiebeautos wird immer die linke Seite festgelegt. Von diesem Punkt aus werden die Autos entweder nach rechts oder links verschoben.

**Grenzen und blockierende Autos**

Bevor das Auto tatsächlich verschoben wird, muss eine Funktion aufgerufen werden, die mithilfe einer if/else-Abfrage prüft, ob der geplante Zug durch eine Grenze behindert wird. Falls keine Grenze auftaucht, wird zudem überprüft, ob ein anderes Auto in die Quere kommt. Die abgebildete Funktion gibt schließlich zwei bool-Werte aus, die dazu genutzt werden, auf jede Situation passend zu reagieren.

def is\_crash\_left(parking\_lot\_copy, moving\_car):

if moving\_car - 1 >= 0:  
 if parking\_lot\_copy[moving\_car - 1][1] != 0:  
 crash = True  
 blocking\_car = True  
 else:  
 crash = False  
 blocking\_car = False  
 return crash, blocking\_car  
 else:  
 crash = True  
 blocking\_car = False  
 return crash, blocking\_car

**Autos verschieben**

Für den Fall, dass es weder einen Zusammenstoß mit der Grenze noch mit einem anderen Auto gibt, wird das Schiebeauto direkt verschoben. Da all dies innerhalb eines while-Loops geschieht, wird das aktuelle Schiebeauto bei jedem Durchlauf nur um eine Stelle verschoben, bis der Loop zu False wird und die schlussendliche Print-Ausgabe gemacht wird.

Taucht jedoch im Verlauf eine Grenze auf, die verhindert, dass die Schiebeautos weiter verschoben werden können, wird automatisch die Schieberichtung geändert. Das Schiebeauto wird anschließend so weit wie nötig verschoben, um das Auto ausparken zu lassen.

**Sonderfall „mehrere Schiebeautos nacheinander verschieben“: Lösung mit Rekursion**

Da es vorkommen kann, dass ein anderes Schiebeauto das Weiterschieben behindert, wurde eine Rekursion in die Funktion eingebaut. Diese prüft jedes im Weg stehende Auto und verschiebt dann das letzte dieser Reihe, sodass alle anderen nachrücken können. Auch dies geschieht, bis das ausparkende Auto frei ist.

def move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction):

crash, blocking\_car = is\_crash\_left(parking\_lot\_copy, moving\_car)  
 if not crash:  
 direction = "links"  
 if not cars.get(parking\_lot\_copy[moving\_car][1]):  
 cars[parking\_lot\_copy[moving\_car][1]] = 0  
 cars[parking\_lot\_copy[moving\_car][1]] += 1  
 moving\_car = move\_car\_one\_left(parking\_lot\_copy, moving\_car)  
 elif crash and not blocking\_car:  
 direction = "rechts"  
 else:  
 direction = "links"  
 moving\_car -= 2  
 move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 moving\_car += 2  
 if not cars.get(parking\_lot\_copy[moving\_car][1]):  
 cars[parking\_lot\_copy[moving\_car][1]] = 0  
 cars[parking\_lot\_copy[moving\_car][1]] += 1  
 moving\_car = move\_car\_one\_left(parking\_lot\_copy, moving\_car)

return moving\_car, direction

**Ausgabe**

Während des gesamten Vorgangs werden die Autos, die verschoben werden müssen, zusammen mit der Anzahl an Schritten, die sie verschoben werden müssen, in einem Dictionary gespeichert. Zusätzlich wird die Schieberichtung zurückgegeben. Um die Ausgabe richtig zu formatieren, wird ein for-Loop verwendet, mit dem aus den gespeicherten Verschiebungen eine Liste erstellt wird. Diese Liste enthält nun alle nötigen Daten pro Schiebeauto, das bewegt werden muss: Buchstabe, Schrittanzahl und Richtung. Anschließend wird die Liste sortiert, ggf. rückwärts gedreht und mithilfe der Methode .join(list) zu einem String zusammengefasst. Daraus folgt eine formatierte Print-Ausgabe.

Beispiele

Genügend Beispiele einbinden! Die Beispiele von den BWINF-Webseiten sollten hier diskutiert werden, aber auch eigene Beispiele sind sehr gut – besonders wenn sie Spezialfälle abdecken. Bitte jedoch nicht 30 Seiten Programmausgabe hier einfügen!

Quellcode

from pathlib import Path  
from copy import deepcopy  
  
  
def read\_input(filename='parkplatz0.txt'):  
 *""" Beispieldatei einlesen  
 Die Zeilen in Integer und List umwandeln und Zeilenumbrüche mit .strip() entfernen.  
 Default ist das Aufgabenbeispiel parkplatz0.txt.  
 """* file = Path('sampledata', filename)  
 with open(file, 'r') as file\_in:  
 parked\_cars = file\_in.readline().split()  
 moving\_cars\_total = file\_in.readline().strip()  
 moving\_cars = [line.strip() for line in file\_in.readlines()]  
  
 print(f"Parkende Autos: {parked\_cars}")  
 print(f"Anzahl an Schiebeautos: {moving\_cars\_total}")  
 print(f"Position der Schiebeautos: {moving\_cars}")  
  
 return parked\_cars, moving\_cars  
  
  
def make\_parkinglot(parked\_cars, moving\_cars):  
 *"""  
 Erstellt aus parked\_cars und alphabet die Liste parkinglot mit allen Autos, die ausgeparkt werden sollen und dem  
 dazugehörigen Wert, der angibt, ob die Stelle blockiert wird oder frei ist.  
 """* alphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T",  
 "U", "V", "W", "X", "Y", "Z"]  
 parkinglot = []  
 end = parked\_cars[1]  
  
 occupiedlot = {}  
  
 for element in moving\_cars:  
 letter, number = element.split()  
 number = int(number)  
 occupiedlot[letter] = number  
 occupiedlot[letter.lower()] = number + 1  
  
 count = 0  
 for letter in alphabet:  
 spot = [letter, 0]  
 for key, value in occupiedlot.items():  
 if value == count:  
 spot.append(key.upper())  
 spot.remove(0)  
 parkinglot.append(spot)  
 if letter == end:  
 break  
 count += 1  
  
 print(f"Layout Parkplatz: {parkinglot}\n")  
 return parkinglot  
  
  
def move\_cars(parkinglot):  
 *"""  
 Geht durch den gesamten Parkplatz und entscheidet, was getan werden muss, damit die Autos ausparken können.  
 """* moving\_car = None  
 for current\_car in range(0, len(parkinglot)):  
 parking\_lot\_copy = deepcopy(parkinglot)  
 cars = {}  
 count = 0  
 direction = None  
 if parking\_lot\_copy[current\_car][1] != 0:  
 while parking\_lot\_copy[current\_car][1] != 0:  
 count += 1  
  
 if count == 1:  
 if current\_car == len(parkinglot) - 1:  
 moving\_car = current\_car - 1  
 moving\_car, direction = move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 continue  
  
 if current\_car == 0:  
 moving\_car = current\_car  
 moving\_car, direction = move\_right\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 continue  
  
 if 0 < current\_car < len(parkinglot) - 1:  
 if parking\_lot\_copy[current\_car + 1][1] == parking\_lot\_copy[current\_car][1]:  
 moving\_car = current\_car  
 moving\_car, direction = move\_right\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 continue  
  
 elif parking\_lot\_copy[current\_car - 1][1] == parking\_lot\_copy[current\_car][1]:  
 moving\_car = current\_car - 1  
 moving\_car, direction = move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 continue  
 else:  
 if direction == "links":  
 moving\_car, direction = move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 continue  
 elif direction == "rechts":  
 moving\_car, direction = move\_right\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 continue  
  
 print\_result(parking\_lot\_copy, current\_car, cars, direction)  
  
 else:  
 print(f"{parking\_lot\_copy[current\_car][0]}: ")  
  
  
def move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction):  
 *"""  
 Prüft, inwiefern Autos nach links verschoben werden können und verschiebt sie, sobald genügend Platz vorhanden ist.  
 """* crash, blocking\_car = is\_crash\_left(parking\_lot\_copy, moving\_car)  
 if not crash:  
 direction = "links"  
 if not cars.get(parking\_lot\_copy[moving\_car][1]):  
 cars[parking\_lot\_copy[moving\_car][1]] = 0  
 cars[parking\_lot\_copy[moving\_car][1]] += 1  
 moving\_car = move\_car\_one\_left(parking\_lot\_copy, moving\_car)  
 elif crash and not blocking\_car:  
 direction = "rechts"  
 else:  
 direction = "links"  
 moving\_car -= 2  
 move\_left\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 moving\_car += 2  
 if not cars.get(parking\_lot\_copy[moving\_car][1]):  
 cars[parking\_lot\_copy[moving\_car][1]] = 0  
 cars[parking\_lot\_copy[moving\_car][1]] += 1  
 moving\_car = move\_car\_one\_left(parking\_lot\_copy, moving\_car)  
  
 return moving\_car, direction  
  
  
def move\_right\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction):  
 *"""  
 Prüft, inwiefern Autos nach rechts verschoben werden können und verschiebt sie, sobald genügend Platz vorhanden ist.  
 """* crash, blocking\_car = is\_crash\_right(parking\_lot\_copy, moving\_car)  
 if not crash:  
 direction = "rechts"  
 if not cars.get(parking\_lot\_copy[moving\_car][1]):  
 cars[parking\_lot\_copy[moving\_car][1]] = 0  
 cars[parking\_lot\_copy[moving\_car][1]] += 1  
 moving\_car = move\_car\_one\_right(parking\_lot\_copy, moving\_car)  
 elif crash and not blocking\_car:  
 direction = "links"  
 else:  
 direction = "rechts"  
 moving\_car += 2  
 move\_right\_recursion(parking\_lot\_copy, moving\_car, current\_car, cars, direction)  
 moving\_car -= 2  
 if not cars.get(parking\_lot\_copy[moving\_car][1]):  
 cars[parking\_lot\_copy[moving\_car][1]] = 0  
 cars[parking\_lot\_copy[moving\_car][1]] += 1  
 moving\_car = move\_car\_one\_right(parking\_lot\_copy, moving\_car)  
  
 return moving\_car, direction  
  
  
def move\_car\_one\_left(parking\_lot\_copy, moving\_car):  
 *"""  
 Verschiebt das Auto um eine Stelle nach links.  
 """* parking\_lot\_copy[moving\_car - 1][1] = parking\_lot\_copy[moving\_car][1]  
 parking\_lot\_copy[moving\_car + 1][1] = 0  
 moving\_car -= 1  
 return moving\_car  
  
  
def move\_car\_one\_right(parking\_lot\_copy, moving\_car):  
 *"""  
 Verschiebt das Auto um eine Stelle nach rechts.  
 """* parking\_lot\_copy[moving\_car + 2][1] = parking\_lot\_copy[moving\_car][1]  
 parking\_lot\_copy[moving\_car][1] = 0  
 moving\_car += 1  
 return moving\_car  
  
  
def is\_crash\_left(parking\_lot\_copy, moving\_car):  
 *"""  
 Prüft, ob das Auto auf der linken Seite durch eine Grenze blockiert wird oder nicht.  
 Falls daneben keine Grenze liegt, wird überprüft, ob es durch ein anderes Auto blockiert wird.  
 """* if moving\_car - 1 >= 0:  
 if parking\_lot\_copy[moving\_car - 1][1] != 0:  
 crash = True  
 blocking\_car = True  
 else:  
 crash = False  
 blocking\_car = False  
 return crash, blocking\_car  
 else:  
 crash = True  
 blocking\_car = False  
 return crash, blocking\_car  
  
  
def is\_crash\_right(parking\_lot\_copy, moving\_car):  
 *"""  
 Prüft, ob das Auto auf der rechten Seite durch eine Grenze blockiert wird oder nicht.  
 Falls daneben keine Grenze liegt, wird überprüft, ob es durch ein anderes Auto blockiert wird.  
 """* if moving\_car + 2 <= len(parkinglot) - 1:  
 if parking\_lot\_copy[moving\_car + 2][1] != 0:  
 crash = True  
 blocking\_car = True  
 else:  
 crash = False  
 blocking\_car = False  
 return crash, blocking\_car  
 else:  
 crash = True  
 blocking\_car = False  
 return crash, blocking\_car  
  
  
def print\_result(parking\_lot\_copy, current\_car, cars, direction):  
 *"""  
 Macht das dictionary zu einer Liste und formatiert die Ausgabe so, dass die Autos in der richtigen Reihenfolge  
 genannt werden.  
 """* solution = []  
 for key, value in cars.items():  
 move = f"{key} {value} {direction}"  
 solution.append(move)  
 solution.sort()  
 if direction == "rechts":  
 solution.reverse()  
 result = ", ".join(solution)  
 print(f"{parking\_lot\_copy[current\_car][0]}: {result}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 parked\_cars, moving\_cars = read\_input()  
 parkinglot = make\_parkinglot(parked\_cars, moving\_cars)  
 move\_cars(parkinglot)