Aufgabe 1: Schiebeparkplatz

Team-ID: 00105

Team: Lennart Enns

Bearbeiter dieser Aufgabe:

Lennart Enns

Inhaltsverzeichnis

Lösungsidee

Zuerst soll der Benutzer eine der Beispieldateien auswählen können, die geöffnet wird. Die Daten aus dieser Datei sollen anschließend passend gespeichert werden. Dann werden die eingeparkten Autos (A, B, C…) nacheinander ausgegeben und, falls sie blockiert sind, jeweils eine möglichst effiziente Lösung.

Um herauszufinden, welche Autos um wie viele Felder in welche Richtung verschoben werden müssen soll das Programm erst einmal „nachsehen“, ob links oder rechts genug Felder frei sind, um das blockierende Auto, nachdem ggf. noch andere Autos passend verschoben wurden, in diese Richtung verschieben zu können. Wenn beide Richtungen möglich wären, wird geprüft, in welcher weniger Autos verschoben werden müssen und falls auch die Anzahl zu verschiebender Autos in beiden Richtungen dieselbe ist, wird die Richtung ausgewählt, in die das blockierende Auto am Ende um nur ein Feld verschoben werden muss, um freie Ausfahrt zu gewährleisten.

Nachdem also eine Seite ausgewählt wurde, wird eine Kette aus allen betreffenden Autos erstellt, die zwischen dem blockierenden Auto und dem nächsten bzw. dem übernächsten freien Feld in der Richtung liegen, also verschoben werden müssen. Diese werden dann um jeweils ein oder zwei Felder nach links bzw. rechts verschoben, so wie am Ende auch das Auto, das vor dem auszuparkenden Auto steht. Schließlich wird die so ermittelte Lösung ausgegeben.

Umsetzung

Ich habe zur Umsetzung auch hier die Programmiersprache Python angewendet. Zuerst importiere ich aus dem Modul string das Alphabet in Großbuchstaben als String und das Modul os zur Ermittlung der Anzahl von vorhandenen Beispieldateien.

```
from string import ascii_uppercase as alphabet
import os
```

Der String mit dem Alphabet wird in eine Liste umgewandelt und in der Variable n_Dateien wird die Länge der Liste der Beispieldateien gespeichert, also die Anzahl der Dateien.

```
alphabet = list(alphabet)
n_Dateien = len(os.listdir("Beispieldateien"))
```

Als nächstes wird, wie in Junioraufgabe 1, nach einer Nummer für eine Beispieldatei gefragt, und zwar so lange, bis die Eingabe in einen Integer umgewandelt werden kann und dieser Wert zwischen 0 und der Nummer der letzten Beispieldatei liegt.

```
while True:
    try:
        datei_nummer = int(input("Welche Beispieldatei soll verwendet werden? "))
        if not datei_nummer in range(0, n_Dateien):
            raise Exception
        break
    except:
        print("Bitte gebe eine Zahl von 0 bis "+str(n_Dateien-1)+" ein.\n")
```

Jetzt wird die ausgewählte Datei geöffnet und mit der Funktion readlines() eine Liste mit den einzelnen Zeilen erstellt.

```
with open("Beispieldateien\\"+os.listdir("Beispieldateien")[datei_nummer]) as file:
```

```
    Zeilen = file.readlines()
```
Der Bereich, in dem die Buchstaben von den eingeparkten Autos liegen, wird ermittelt, indem von der ersten Zeile erst das Zeilenende-Zeichen entfernt wird und dann mit split() eine Liste der beiden Buchstaben der Zeile mit einem Leerzeichen als Separator erstellt wird. Mithilfe dieser Liste wird eine weitere Liste erstellt, die den Ausschnitt des Alphabets enthält, in dem die Namen der Autos liegen.

```
    bereich = Zeilen[0].strip().split(" ")
    Auto_namen = alphabet[alphabet.index(bereich[0]):alphabet.index(bereich[1])+1]
```
Aus der Länge dieser Liste wird die Anzahl der normal eingeparkten Autos ermittelt. Jetzt wird die Liste „Autos“, in der später jedes quer geparktes Auto repräsentiert ist, deklariert. Sie wird zu Beginn mit so vielen -1 gefüllt, wie es normal eingeparkte Autos gibt. Eine -1 steht für ein freies Feld.

```
    n_normal = len(Auto_namen)
    Autos = n_normal*[-1]
```
Nachdem nun mit dem gleichen Vorgehen wie bisher aus der zweiten Zeile die Anzahl der quer stehenden Autos ermittelt und in einen Integer-Wert umgewandelt wurde, werden zur einfacheren Iteration in der bevorstehenden For-Schleife die ersten beiden Zeilen aus der Liste mit den Zeilen gelöscht und eine Liste erstellt, die später die Referenzen zu den Instanzen der Klasse Quer_auto enthält, deren Aufbau später erklärt wird.

```
    n_quer = int(Zeilen[1].strip())
    del Zeilen[:2]
    quer_obj = []
```
In einer For-Schleife, die n_quer mal durchlaufen wird, wird bei jedem Durchlauf zuerst der Name des Autos aus dem ersten Zeichen der entsprechenden Zeile ermittelt. Die jeweilige Position wird bestimmt, indem in der Zeile alles ab dem Beginn der Zahl, also ab dem Index 2, um das Zeilenende-Zeichen gekürzt und in eine Ganzzahl konvertiert wird. Mit dem Index des Objektes in der Liste quer_obj und den eben ausgemachten Daten als Parameter für den Konstruktor, die später für

die Methoden wichtig sein werden, initialisiert das Programm ein Objekt der Klasse Quer_auto.

```
    for n in range(n_quer):
        name = Zeilen[n][0]
        pos = int(Zeilen[n][2:].strip())
        quer_obj.append(Quer_auto(len(quer_obj), name, pos))
```

Im Folgenden erkläre ich die Klasse Quer_auto, die unter Anderem die Methode zur eigentlichen Lösung eines Blockadeproblems beinhaltet. Der Konstruktor übernimmt den nötigen self-Parameter und dazu die drei Argumente nummer, name und pos, welche für den Index des Objektes in der Liste, den Namen als Buchstabe und die Position in der Liste Autos stehen. Diese werden in den Attributen self.nummer, etc. gespeichert. In die Liste Autos wird beim Index der Position des Autos und eine Stelle danach, da das Auto zwei Felder belegt, die eigene Nummer eingetragen.

```
    def __init__(self, nummer, name, pos):
        self.nummer = nummer
        self.name = name
        self.pos = pos
        Autos[pos],Autos[pos+1]=2*[nummer]
```

Die Methode abfolge, welche als Parameter die Position des Autos übernimmt, das vom durch das Objekt dargestellten Auto blockiert wird, bestimmt, wie welche Autos in welcher Reihenfolge verschoben werden sollen, damit möglichst wenige Autos bewegt werden müssen.

Zuerst wird die Differenz aus den Positionen des blockierenden und der des blockierten Autos ermittelt, da sie für spätere Berechnungen nützlich sein wird. Der Boolean-Wert richtung, der angibt, in welche Richtung die quer stehenden Autos alle verschoben werden müssen, wird zuerst auf False gesetzt, was für rechts steht. True steht dementsprechend für links. Außerdem werden die leeren Stellen in der Reihe aus allen quer stehenden Autos ausgemacht, indem die Liste Autos in einer For-Schleife durchgegangen wird und der Index von jedem Wert, der -1, also Leere, ist, in der Liste leere_stellen gespeichert wird.

```
    def abfolge(self, posAuto):
        diff=posAuto-self.pos
        richtung = False
        leere_stellen = [i for i, stelle in enumerate(Autos) if stelle == -1]
```

Wenn nun das Ausparken des blockierten Autos durch eine Verschiebung nach links erreicht werden kann, wenn also in der Liste Autos links vom blockierenden Auto mindestens 2-diff leere Stellen vorkommen, wird die Variable richtung auf True, also links, gesetzt. In einer Liste werden alle leeren Stellen gespeichert, die sich links vom Auto befinden, deren Index also kleiner ist als seine Position.

```
        if Autos[:self.pos].count(-1) >= 2-diff:
            richtung = True
            leere_stellen_l = [i for i in leere_stellen if i < self.pos]
```

Anschließend wird der Bereich aus der Liste Autos ermittelt, in dem sich die Autos befinden, die verschoben werden müssen. Dafür wird der Ausschnitt vom Index der von rechts nach links mindestens benötigten leeren Stelle in leere_stellen_l bis ausschließlich zur Position des Objekts gezogen. Falls auch rechts vom Auto genug Stellen zum Verschieben frei sind, wird dasselbe für die leeren Stellen rechts vom Auto mit entsprechend angepasster Formel wiederholt.

```
            bereich_l = Autos[leere_stellen_l[-2+diff]:self.pos]
                if Autos[self.pos:].count(-1) >= 1+diff:
                    leere_stellen_r = [i for i in leere_stellen if i > self.pos]
                    bereich_r = Autos[self.pos+2:leere_stellen_r[diff]+1]
```

Wenn also sowohl durch das Verschieben nach links als auch nach rechts eine Lösung erreicht werden kann, muss entschieden werden, was effizienter ist. Dazu werden zwei Variablen mit 0 initialisiert, die angeben sollen, wie viele zu verschiebende Autos, außer dem die Methode ausführenden Objekt, es in der jeweiligen Richtung gibt. In zwei For-Schleifen wird für beide Seiten der relevante Bereich durchgegangen. Wenn der aktuelle Wert nicht -1 ist und nicht das rechte Feld eines Autos ist, also noch nicht berücksichtigt wurde, wird die jeweilige Zählervariable um 1 erhöht. Die Funktionalität wird nicht dadurch beeinträchtigt, dass der aktuell betrachtete Index minus 1 in der Liste abgefragt wird, denn wenn der aktuelle Index 0 ist und damit

der abgefragte Index -1, bedeutet das in Python der letzte Wert in der Liste, welcher unmöglich gleich dem ersten sein kann, da sonst ein Auto „zerteilt“ wäre.

```
                    n_Autos_l, n_Autos_r = 0, 0
                    for i in range(len(bereich_l)):
                        if bereich_l[i-1] != bereich_l[i] != -1:
                            n_Autos_l += 1
                    for i in range(len(bereich_r)):
                        if bereich_r[i-1] != bereich_r[i] != -1:
                            n_Autos_r += 1
```

Wenn rechts weniger Autos verschoben werden müssten als links, wird die Richtung dementsprechend auf rechts gesetzt.

```
                if n_Autos_r < n_Autos_l:
                    richtung=False
```

Wenn allerdings links und rechts gleich viele zu verschiebende Autos sind, wird die Richtung ausgewählt, in die das ausführende Objekt am Ende um nur ein statt zwei Felder verschoben werden muss, um das Auto ausparken zu lassen. Wenn die Differenz zwischen den Positionen des eingeparkten und des davor stehenden Autos also 1 beträgt, wird die Richtung auf links (True) gesetzt und umgekehrt. Die Formel dafür ist also die Differenz als Wahrheitswert.

```
                elif n_Autos_r == n_Autos_l:
                    richtung = bool(diff)
```

Falls die Lösung nur mit einer Verschiebung nach rechts zu erreichen ist, werden die leeren Stellen und der relevante Bereich entsprechend nur für die rechte Seite bestimmt und die Richtung auf False, also rechts, gesetzt.

```
        elif Autos[self.pos:].count(-1) >= 1+diff:
            leere_stellen_r = [i for i in leere_stellen if i > self.pos]
            bereich_r = Autos[self.pos+2:leere_stellen_r[diff]+1]
            richtung = False
```

Wenn in keiner Richtung eine Lösung erreicht werden kann, wird die Methode beendet und gibt False aus.

```
        else:
            return False
```

Wenn die Methode noch nicht beendet wurde und damit eine Lösung existiert, initialisiert die Methode eine Liste namens kette, in der nachher die Nummern der zu verschiebenden Autos in der Reihenfolge stehen, in der sie verschoben werden müssen. Anschließend wird je nach Richtung zuerst die Anzahl von leeren Feldern im zu verändernden Bereich links bzw. rechts ausgemacht und dann in einer For-Schleife der jeweilige Bereich von vorwärts bzw. rückwärts durchgegangen und jede Autonummer, die nicht -1 und nicht bereits in der Liste ist, hinzugefügt.

```
        kette=[]
        if richtung:
            for i in bereich_l:
                if i != -1 and not i in kette:
                    kette.append(i)
        else:
            for i in bereich_r[::-1]:
                if i != -1 and not i in kette:
                    kette.append(i)
```

Zum Schluss wird noch die Nummer der ausführenden Instanz zu kette hinzugefügt, weil dieses Auto unabhängig von der Richtung als letztes verschoben wird, nachdem die anderen Autos den Weg frei gemacht haben. Es werden die Richtung und die Kette aus den zu verschiebenden Autos in der richtigen Reihenfolge ausgegeben.

```
        kette.append(self.nummer)
        return richtung, kette, n_felder
```

Die Klasse Quer_auto beinhaltet auch die Methode verschieben, die mit den von abfolge() ausgegebenen Parametern arbeitet. Sie simuliert das Verschieben eines Autos in der Kette und gibt einen Satz wie z.B. „H 2 links“ aus. Zuerst wird die Anzahl der Felder ermittelt, um die das jeweilige Auto verschoben werden muss. Dafür werden die leeren Felder in Autos zwischen der Nummer der ausführenden Instanz und

der des letzten Autos in der Kette mit count() gezählt und das Ergebnis von 2 abgezogen. Jedes Auto muss nämlich maximal um 2 Felder verschoben werden, damit das eingeparkte Auto am Ende ausparken kann, und wenn das nächste benötigte leere Feld bei einem anderen Auto ist, muss das Aktuelle nur um ein Feld verschoben werden.

```
    def verschieben(self, richtung, kette):
        i1=Autos.index(self.nummer)
        i2=Autos.index(kette[-1])
        n_felder=2-Autos[i1:i2].count(-1)
```

Als nächstes wird in der Methode ein String ausgegeben, der aus dem eigenen Buchstaben, der Anzahl der Felder und der Richtung zusammengesetzt ist.

```
        if richtung:
            return (self.name+" "+str(n_felder)+" links")
        else:
            return (self.name+" "+str(n_felder)+" rechts")
```

Nachdem die Klasse Quer_auto definiert, die Dateinummer abgefragt und die Datei ausgelesen wurde, wird mit einer For-Schleife von 0 bis ausschließlich zur Anzahl von eingeparkten Autos gezählt. Bei jedem Durchgang werden ein Zeilenumbruch und dann der jeweilige Buchstabe mit Doppelpunkt in der Konsole ausgegeben, wobei durch den Parameter end=““ beim nächsten print-Befehl in derselben Zeile weitergeschrieben wird. Wenn die Stelle n in Autos kein leeres Feld, also blockiert, ist, wird vom blockierenden Auto, dessen Objekt anhand seiner Nummer in quer_obj ermittelt wird, die Methode abfolge(n) abgefragt und die Ausgabe in der Variable ausgabe gespeichert.

```
for n in range(n_normal):
    print("\n"+Auto_namen[n]+": ",end="")
    if Autos[n] != -1:
        blockierend=quer_obj[Autos[n]]
        ausgabe=blockierend.abfolge(n)
```

Wenn die Ausgabe nicht False ist, es also eine Lösung gibt, wird zuerst die Ausgabe der Methode verschieben vom ersten Objekt, dessen Nummer in kette steht, in der Konsole ausgegeben und dann die Ausgaben von den restlichen Objekten, jeweils mit einem Komma

davor und auch in derselben Zeile. Wenn es keine Lösung gibt, wird man darüber informiert.

```
if ausgabe:
            richtung, kette=ausgabe
            print(quer_obj[kette[0]].verschieben(richtung, kette),end="")
            for i in kette[1:]:
                print(", "+quer_obj[i].verschieben(richtung, kette),end="")
        else:
            print("Dieses Auto ist unmöglich auszuparken!",end="")
```

Am Ende wird durch print(„“) nochmal für einen Zeilenumbruch gesorgt, damit auch die letzte Konsolenausgabe zu sehen ist. Durch eine unendliche Schleife schließt sich das Konsolenfenster nicht, sodass man die Ausgabe in Ruhe lesen kann.

```
print("")
while True:
    pass
```

Beispiele

Hier sind die Programmausgaben für jede Parkplatzdatei. Ich habe zusätzlich noch die Dateien „parkplatz6.txt“ bis „parkplatz9.txt“ erstellt, die verschiedene Sonderfälle darstellen. Bei Parkplatz 6 gibt es nur ein leeres Feld, bei Parkplatz 7 können alle Autos unmöglich ausparken, bei Nummer 8 können alle direkt ausparken und Parkplatz 9 ist besonders groß.

Parkplatz 0:

A:

B:

C: H 2 rechts

D: H 2 links

E:

F: H 1 links, I 2 links

G: I 2 links

Parkplatz 1:

A:

B: P 2 rechts, O 2 rechts

C: O 2 links

D: P 2 rechts

E: O 2 links, P 2 links

F:

G: Q 2 rechts

H: Q 2 links

I:

J:

K: R 2 rechts

L: R 2 links

M:

N:

Parkplatz 2:

A:

B:

C: O 2 rechts

D: O 2 links

E:

F: O 1 links, P 2 links

G: P 2 links

H: R 2 rechts, Q 2 rechts

I: P 2 links, Q 2 links

J: R 2 rechts

K: P 2 links, Q 2 links, R 2 links

L:

M: P 1 links, Q 1 links, R 1 links, S 2 links

N: S 2 links

Parkplatz 3:

A:

B: O 2 rechts

C: O 2 links

D:

E: P 2 rechts

F: P 2 links

G:

H:

I: Q 2 links

J: Q 2 links

K: Q 2 links, R 2 links

L: Q 2 links, R 2 links

M: Q 2 links, R 2 links, S 2 links

N: Q 2 links, R 2 links, S 2 links

Parkplatz 4:

A: R 2 rechts, Q 2 rechts

B: R 2 rechts, Q 2 rechts

C: R 2 rechts

D: R 2 rechts

E:

F:

G: S 2 rechts

H: S 2 links

I:

J:

K: T 2 rechts

L: T 2 links

M:

N: U 2 rechts

O: U 2 links

P:

Parkplatz 5:

A:

B:

C: P 2 links

D: P 2 links

E: Q 2 rechts

F: Q 2 rechts

G:

H:

I: R 2 rechts

J: R 2 links

K:

L:

M: S 2 rechts

N: S 2 links

O:

Parkplatz 6:

A: K 2 rechts, J 2 rechts

B: Dieses Auto ist unmöglich auszuparken!

C: K 2 rechts

D: Dieses Auto ist unmöglich auszuparken!

E:

F: Dieses Auto ist unmöglich auszuparken!

G: L 2 links

H: Dieses Auto ist unmöglich auszuparken!

I: L 2 links, M 2 links

Parkplatz 7:

A: Dieses Auto ist unmöglich auszuparken!

B: Dieses Auto ist unmöglich auszuparken!

C: Dieses Auto ist unmöglich auszuparken!

D: Dieses Auto ist unmöglich auszuparken!

E: Dieses Auto ist unmöglich auszuparken!

F: Dieses Auto ist unmöglich auszuparken!

G: Dieses Auto ist unmöglich auszuparken!

H: Dieses Auto ist unmöglich auszuparken!

I: Dieses Auto ist unmöglich auszuparken!

J: Dieses Auto ist unmöglich auszuparken!

K: Dieses Auto ist unmöglich auszuparken!

L: Dieses Auto ist unmöglich auszuparken!

Parkplatz 8:

A:

B:

C:

D:

E:

F:

G:

H:

I:

J:

K:

Parkplatz 9:

A:

B:

C: T 2 rechts

D: T 2 links

E:

F: V 2 rechts, U 2 rechts

G: U 2 links

H: V 2 rechts

I: U 2 links, V 2 links

J:

K: X 2 rechts, W 2 rechts

L: W 2 links

M: X 2 rechts

N: W 2 links, X 2 links

O:

P: W 1 links, X 1 links, Y 2 links

Q: Y 2 links

R: W 1 links, X 1 links, Y 2 links, Z 2 links

S: Y 2 links, Z 2 links

Quellcode

Das ist der unkommentierte Quellcode:

```
from string import ascii_uppercase as alphabet
import os

alphabet = list(alphabet)
n_Dateien = len(os.listdir("Beispieldateien"))

class Quer_auto:
    def __init__(self, nummer, name, pos):
        self.nummer = nummer
        self.name = name
        self.pos = pos
        Autos[pos],Autos[pos+1]=2*[nummer]

    def verschieben(self, richtung, kette):
        i1=Autos.index(self.nummer)
        i2=Autos.index(kette[-1])
        n_felder=2-Autos[i1:i2].count(-1)
        if richtung:
            return (self.name+" "+str(n_felder)+" links")
        else:
            return (self.name+" "+str(n_felder)+" rechts")

    def abfolge(self, posAuto):
        diff=posAuto-self.pos
        richtung = False
        leere_stellen = [i for i, stelle in enumerate(Autos) if stelle == -1]
        if Autos[:self.pos].count(-1) >= 2-diff:
            richtung = True
            leere_stellen_l = [i for i in leere_stellen if i < self.pos]
            bereich_l = Autos[leere_stellen_l[-2+diff]:self.pos]
            if Autos[self.pos:].count(-1) >= 1+diff:
                leere_stellen_r = [i for i in leere_stellen if i > self.pos]
                bereich_r = Autos[self.pos+2:leere_stellen_r[diff]+1]
                n_Autos_l, n_Autos_r = 0, 0
                for i in range(len(bereich_l)):
                    if bereich_l[i-1] != bereich_l[i] != -1:
                        n_Autos_l += 1
                for i in range(len(bereich_r)):
                    if bereich_r[i-1] != bereich_r[i] != -1:
                        n_Autos_r += 1
```

```
                if n_Autos_r < n_Autos_l:
                    richtung=False
                elif n_Autos_r == n_Autos_l:
                    richtung = bool(diff)
        elif Autos[self.pos:].count(-1) >= 1+diff:
            leere_stellen_r = [i for i in leere_stellen if i > self.pos]
            bereich_r = Autos[self.pos+2:leere_stellen_r[diff]+1]
            richtung = False
        else:
            return False
        kette=[]
        if richtung:
            for i in bereich_l:
                if i != -1 and not i in kette:
                    kette.append(i)
        else:
            for i in bereich_r[::-1]:
                if i != -1 and not i in kette:
                    kette.append(i)
        kette.append(self.nummer)
        return richtung, kette

while True:
    try:
        datei_nummer = int(input("Welche Beispieldatei soll verwendet werden? "))
        if not datei_nummer in range(0, n_Dateien):
            raise Exception
        break
    except:
        print("Bitte gebe eine Zahl von 0 bis "+str(n_Dateien-1)+" ein.\n")

with open("Beispieldateien\\"+os.listdir("Beispieldateien")[datei_nummer]) as file:
    Zeilen = file.readlines()
    bereich = Zeilen[0].strip().split(" ")
    Auto_namen = alphabet[alphabet.index(bereich[0]):alphabet.index(bereich[1])+1]
    n_normal = len(Auto_namen)
    Autos = n_normal*[-1]
    n_quer = int(Zeilen[1].strip())
    del Zeilen[:2]
    quer_obj = []
    for n in range(n_quer):
        name = Zeilen[n][0]
        pos = int(Zeilen[n][2:].strip())
        quer_obj.append(Quer_auto(len(quer_obj), name, pos))

for n in range(n_normal):
    print("\n"+Auto_namen[n]+": ",end="")
    if Autos[n] != -1:
        blockierend=quer_obj[Autos[n]]
        ausgabe=blockierend.abfolge(n)
        if ausgabe:
            richtung, kette=ausgabe
```

```
            print(quer_obj[kette[0]].verschieben(richtung, kette),end="")
            for i in kette[1:]:
                print(", "+quer_obj[i].verschieben(richtung, kette),end="")
        else:
            print("Dieses Auto ist unmöglich auszuparken!",end="")
print("")
while True:
    pass
```