Schiebeparkplatz

A1, Team-ID: 00353, Team-Name: U+1F947, Leonhard Masche, 24.09.2021

Inhaltsverzeichnis

Lösungsidee

Aus den beiden Buchstaben in Zeile 1 der Test-Datei wird eine Liste mit allen Großbuchstaben, die sich dazwischen befinden, erstellt. Für die blockierenden (querstehenden) Autos wird in einem Dictionary ihr Buchstabe mit ihrer Position assoziiert.

Als erster Schritt wird berechnet, wieviel Platz zum Bewegen nach links und rechts verfügbar ist. Damit wird entschieden, welche Richtungen überhaupt beim Verschieben infrage kommen.

Nun werden für beide Richtungen (wenn möglich), berechnet, wieviele 'Züge' gemacht werden müssen, um aus dem Stellpatz ausfahren zu können. Auch Kollisionen werden über einen Check in der `squash_left` und `squash_right` Funktion, und ihr rekursives Verhalten bedacht und diese Autos passend weiter verschoben.

Zuletzt werden die beiden Richtungen auf Basis der benötigten 'Züge' verglichen. Die Richtung mit der geringsten Anzahl an Zügen wird formatiert und ausgegeben.

Diese Schritte wiederholen sich für jeden Stellplatz aus der Liste.

Funktionen

def get_blocking(i)

> Gibt den Buchstaben des querstehenden Autos zurück, das die Position `i` blockiert.

def check_free(i)

> Gibt einen Wahrheitswert zurück, ob die Position `i` blockiert ist.

def get_movement(car, i)

> Gibt die Anzahl Felder zurück, die das Auto `car` in eine bestimmte Richtung bewegt werden müsste, um das Feld `i` freizumachen.

def squash_left(i, actions)

> Berechnet rekursiv alle Züge, um Position `i` durch Verschiebung nach links freizumachen. Diese Berechnung schließt Kollisionen mit anderen Autos ein.

def squash_right(i, actions)

> *wie `squash_left`, nur für rechts*

Umsetzung

Das Programm ist in der Sprache Python umgesetzt. Der Aufgabenordner enthält neben dieser Dokumentation eine ausführbare Python-Datei. Diese Datei ist mit einer Python-Umgebung ab der Version `3.6` ausführbar.

Wird das Programm gestartet, wird zuerst eine Eingabe in Form einer einstelligen Zahl erwartet, um ein bestimmtes Beispiel auszuwählen. *(Das heißt: `0` für Beispiel `parkplatz0.txt`)*

Nun wird die Logik des Programms angewandt und die Ausgabe erscheint in der Kommandozeile.

Beispiele

Hier wird das Programm auf die fünf Beispiele aus dem Git-Repo angewendet:

`parkplatz0.txt`

```
A G
2
H 2
I 5
```

Ausgabe zu `parkplatz0.txt`

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

`parkplatz1.txt`

```
A N
4
O 1
P 3
Q 6
R 10
```

Ausgabe zu parkplatz1.txt

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

---

parkplatz2.txt

```
A N
5
O 2
P 5
Q 7
R 9
S 12
```

Ausgabe zu parkplatz2.txt

```
A:
B:
C: O 1 rechts
D: O 1 links
E:
F: O 1 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: P 1 links, Q 1 links, R 1 links, S 2 links
N: S 1 links
```

---

parkplatz3.txt

```
A N
5
O 1
P 4
Q 8
R 10
S 12
```

Ausgabe zu `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
```

`parkplatz4.txt`

```
A P
5
Q 0
R 2
S 6
T 10
U 13
```

Ausgabe zu `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:
```

Quellcode

```
# pylama:ignore=C901,E501
import string
from os import path


# absoluter Pfad des ausgewählten Beispiels
path = path.join(
    path.dirname(path.abspath(__file__)),
    f'beispieldaten/parkplatz{input("Nummer des Beispiels eingeben: ")}.txt')

with open(path, 'r') as f:
    lines = f.read().split('\n')


# ausgewählte Test-Datei einlesen
spaces = lines[0].split(' ')
letters = list(string.ascii_uppercase)
bounds = (letters.index(spaces[0]), letters.index(spaces[1])+1)
parked = letters[bounds[0]:bounds[1]]


# dictionary aus blockierenden (querstehenden) Autos erstellen
blocking = dict()
for i in range(int(lines[1])):
    line = lines[2+i]
    blocking[line.split(' ')[0]] = int(line.split(' ')[1])


def get_blocking(i):
    # gibt das Auto, das die Position mit dem index
    # `i` blockiert, zurück. Sonst `None`
    for x in blocking.items():
        if i-1 in x:
            return x[0]
    for x in blocking.items():
        if i in x:
            return x[0]


def check_free(i):
```

```
    # gibt `True` zurück, wenn die Position `i`
    # frei ist, sonst `False`
    if (i - 1 in blocking.values() or
            i in blocking.values()):
        return False
    return True


def get_movement(car, i):
    # gibt die Bewegung, die ein Auto `car` machen müsste,
    # um Position `i` freizugeben, zurück
    left = 1
    if(blocking[car] == i):
        left = 2
    right = 1
    if(blocking[car] == i-1):
        right = 2
    return left, right


def squash_left(i, actions=None):
    # Zeichnet rekursiv alle Bewegungen auf, die gemacht werden
    # müssten, um Position `i` nach links freizumachen
    if not actions:
        actions = [0, []]
    car = get_blocking(i)
    left, _ = get_movement(car, i)
    actions[0] += left
    actions[1].append(f'{car} {left} links')
    if not check_free(blocking[car]-left):
        actions = squash_left(blocking[car]-left, actions)
    return actions


def squash_right(i, actions=None):
    # Zeichnet rekursiv alle Bewegungen auf, die gemacht werden
    # müssten, um Position `i` nach rechts freizumachen
    if not actions:
        actions = [0, []]
    car = get_blocking(i)
    _, right = get_movement(car, i)
    actions[0] += right
    actions[1].append(f'{car} {right} rechts')
    if not check_free(blocking[car]+right+1):
        actions = squash_right(blocking[car]+right+1, actions)
    return actions


for space, car in enumerate(parked):
    bound_left, bound_right = bounds
    output = f'{car}: '

    # berechnen, wie viel Platz in jede Richtung frei ist
    buffer_left = 0
```

```
    for i in range(bound_left, space):
        if check_free(i):
            buffer_left += 1
    buffer_right = 0
    for i in range(space+1, bound_right):
        if check_free(i):
            buffer_right += 1

    options = []
    car = get_blocking(space)
    # wenn dieser Parkplatz blockiert ist,
    if car is not None:
        left, right = get_movement(car, space)
        # dann vergleiche das Verschieben nach
        if buffer_left >= left:
            # links
            options.append(squash_left(space))
        # mit dem Verschieben nach
        if buffer_right >= right:
            # rechts
            options.append(squash_right(space))
        # wähle die seite mit den wenigsten Bewegungen aus
        options.sort(key=lambda x: x[0])
        # drehe die Liste der Aktionen um, um Kollisionen zu vermeiden
        output += ', '.join(options[0][1][::-1])

    print(output)
```