Andreas Mühlmann

Andi.muelmann@arcor.de, 5.11.2021

Programmiersprache: Python

Dokumentation Schiebeparkplatz

Aufgabe 1

Ansatz:

Meine Idee war es, um die Autos im normalen Parkplatz nacheinander zu befreien, zu berechnen wie weit ein Auto bewegt werden muss, um einen Platz frei zu machen. Das kann durch die Seite, die blockiert und die Richtung berechnet werden. Die Idee wird mithilfe von Rekursion vervollständigt (das Programm kann dadurch auch mit mehreren Autos umgehen), indem immer das blockierende Auto in eine Richtung weg bewegt wird. Wenn dies von einem anderen Auto blockiert wird ist das, das Blockierende (die Richtung bleibt gleich). Beide Richtungen werden probiert und dann verglichen.

Für F:

[ A B C D E F G ]

[ 0 0 H H 0 I I ]

Hier muss „I“ hin.

Deshalb muss sich „H“ wegbewegen.

Weil I“ nach links fahren muss und mit der linken Seite blockiert, muss „I“ um zwei nach links bewegt werden.

„I“ kann nicht nach rechts bewegt werden.

* „I“ muss nach links bewegt werden.

„I“ blockiert mit der linken Seite.

Für „H“ und alle folgenden (hier gibt es keine) kann jetzt das selbe Schema angewendet werde.

Beschreibung:

Funktion main:

* Die Test werden nacheinander durchgegangen.
* Eine Datei wird geöffnet.
* Die Aufstellung der Autos wird von der Website gelesen und strukturiert (Funktion get\_parkinglot\_from\_website).
* Wenn ein normal parkendes Auto nicht blockiert wird, wird es ohne Aktion in die geöffnete Datei geschrieben.
* Sonst werden die zu machenden Aktionen nach links und nach rechts gesucht und zurückgegeben (Funktion car\_mover).
* Dann wird die kürzeste Variante in die geöffnete Datei geschrieben (Funktion write\_shortest\_method\_to\_file).

Funktion get\_parking\_lot\_from\_website:

* Die Website wird von der requests library angesprochen und es wird eine Antwort gegeben.
* Der Inhalt dieser Antwort wird nach dem „utf-8“ decodiert.
* Dieser wird formatiert (Funktion give\_normal\_parkinglot, Funktion give\_obstructing\_parkinglot).
* Dann formatiert zurückgegeben.

Funktion give\_normal\_parkinglot:

* Um die Länge des normalen Parkplatzes zu finden, wird das Alphabet kreiert.
* Dann wird bis zum letzten Auto/ Buchstaben iteriert.
* Die Buchstaben, über die iteriert wurde, werden dann in einer Liste gespeichert und zurückgegeben.

Funktion give\_obstructing\_parkinglot:

* Erst wird ein leerer Parkplatz in der Länge des normalen Parkplatzes kreiert.
* Dann werden die blockierenden Autos an den richtigen Stellen eingefügt.
* Diese Liste wird zurückgegeben.

Funktion car\_mover:

* Die Seite die blockiert wird gegeben (Funktion give\_side\_obstructing).
* Daraus wird berechnet wie weit das blockierende Auto fahren muss (Funktion give\_amount).
* Der Platz an den das Auto müsste, um das andere zu befreien wird berechnet (Funktion give\_spot\_to\_check).
* Wenn dieser nicht innerhalb des Parkplatzes ist es nicht möglich.
* Falls an dem berechneten Platz schon ein Auto ist, wird car\_mover für den Platz aufgerufen.
* Die benötigten Aktionen werden zurückgegeben.

Funktion give\_side\_obstructing:

* Schaut auf welcher Seite der andere Teil des Autos ist.
* Darauf basierend wird eine Seite zurückgegeben.

Funktion give\_amount:

* Gibt basierend auf der Seite des Autos, das blockiert wird, wie weit dieses fahren muss.

Funktion give\_spot\_to\_check:

* Gibt basierend auf der Seite und wie weit das blockierende Auto bewegt werden muss zurück, welcher Platz als nächstes überprüft werden muss.

Funktion write\_shortest\_method\_to\_file:

* Wenn es nur eine Variante gibt, die funktioniert, wird diese in die geöffnete Datei geschrieben (Funktion write\_actions\_to\_file).
* Wenn zwei Varianten möglich sind, wird überprüft, welche kürzer (Funktion count\_amount) ist und dann die kürzere in die Datei geschrieben (Funktion write\_actions\_to\_file).

Funktion count\_amount:

* Für jede Aktion wird die Strecke zusammengerechnet, die gefahren werden muss.
* Diese wird zurückgegeben.

Funktion write\_actions\_to\_file:

* Alle Aktionen für eine Parklücke werden nacheinander in die geöffnete Datei geschrieben.

**Lösungen:**

-----------------------------------------------------------------------------------------------------------------

Test 0

A:

B:

C: H right 1

D: H left 1

E:

F: H left 1, I left 2

G: I left 1

-----------------------------------------------------------------------------------------------------------------

Test 1

A:

B: P right 1, O right 1

C: O left 1

D: P right 1

E: O left 1, P left 1

F:

G: Q right 1

H: Q left 1

I:

J:

K: R right 1

L: R left 1

M:

N:

-----------------------------------------------------------------------------------------------------------------

Test 2

A:

B:

C: O right 1

D: O left 1

E:

F: O left 1, P left 2

G: P left 1

H: R right 1, Q right 1

I: P left 1, Q left 1

J: R right 1

K: P left 1, Q left 1, R left 1

L:

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

N: S left 1

-----------------------------------------------------------------------------------------------------------------

Test 3

A:

B: O right 1

C: O left 1

D:

E: P right 1

F: P left 1

G:

H:

I: Q left 2

J: Q left 1

K: Q left 2, R left 2

L: Q left 1, R left 1

M: Q left 2, R left 2, S left 2

N: Q left 1, R left 1, S left 1

-----------------------------------------------------------------------------------------------------------------

Test 4

A: R right 1, Q right 1

B: R right 2, Q right 2

C: R right 1

D: R right 2

E:

F:

G: S right 1

H: S left 1

I:

J:

K: T right 1

L: T left 1

M:

N: U right 1

O: U left 1

P:

-----------------------------------------------------------------------------------------------------------------

Test 5

A:

B:

C: P left 2

D: P left 1

E: Q right 1

F: Q right 2

G:

H:

I: R right 1

J: R left 1

K:

L:

M: S right 1

N: S left 1

O: