**Documentation BWINF 2021 Round 1 Task 1: Sliding Parking Lot**

Problem....................................................................................................................................2

Solution idea.............................................................................................................................4

Implementation.........................................................................................................................6

Examples..................................................................................................................................7

Code.........................................................................................................................................8

**Problem**

There is a Parking Lot with cars in every slot and cars which are blocking the Parking Lot in some positions. One car has the length 2, so one blocking car can block two cars in the Parking Lot. The code should get as input:

* In the first line the character of the first and the last car in the Parking Lot. For example: A, D => A, B, C, D
* In the second line the number *n*, which is the number of blocking cars in front
* In the following lines are first the character from a blocking car and his left position

For example:

1 A G

2 2

3 H 2

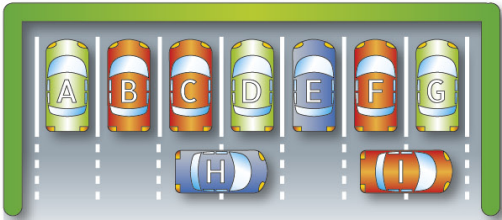
4 I 5

Line 1 -> cars in slot = {“*A*”, “*B*”, “*C*”, “*D*”, “*E*”, “*F*”, “*G*”}

Line 2 -> number of blocking cars *n* = 2

Line 3 -> blocking car one = H; positions: {2, 3}

Line 4 -> blocking car two = I; positions: {5, 6}



The code should give as output for every car in the slots, which blocking cars need to move how many steps and the direction, that the car can go out.

For example:

1 A: E 1 right

2 B: E 1 left

3 C:

4 D:

**Solution idea**

First I need to get the data from the file and separate the data in the constants CARS\_IN\_PAKRSLOT and BLOCKING\_CARS.

CARS\_IN\_PARKSLOT is a list of all cars in the Parking Lot and BLOCKING\_CARS is a dictionary of all blocking cars in front of the Parking Lot with their character as key and their positions as their values.

Then I create a new constants list CARS\_NEED\_MOVES. I iterate through CARS\_IN\_PARKSLOT and when one car is blocked it`s add to the list.

I iterate through CARS\_IN\_PARKSLOT and when the car needs no move I add it directly to the result list as:

*f*“*{*car*}*: ”

But if the car is in CARS\_NEED\_MOVES I need to collect the moves, which are needed for this car to get out of his slot. First I need to check if only one move is needed or multiple moves. For that I check if on one side of the directly blocking car is space, so that the car, can move out of the way, with one move. If this is possible I also get the direction the directly blocking car needs to move. After that I have to find the directly blocking car and add it to the result list as:

*f*“*{*car*}*: *{*blocking\_car*}* 1 *{*direction*}*”

This only works if the cars are in this positions:

*—B—* *or* *-B—*

*—CC—* *CC—*

-> B: C 1 right -> B: C 1 left

But when one move is not possible and multiple moves are needed, I need to do more.

I get the moves needed per side for the directly blocking car to move out of the way.

For example: [1, 2]

-> This means that I need 1 free space on the left or 2 free spaces on

the right side of the directly blocking car

Then I go 1 more position to the left and right and check if there is space. If there is space, I change the moves needed per side. When one side is 0 I return the range where I need to move the cars and their direction.

After that I collect the blocking cars in the range and look for their moves. The first car, which needs to move, moves always as far as he can (up to 2).

Finally I add everyone of them with their moves to the result and print the result list line after line.

**Implementation**

I separated my project in four files:

* *main.py*
* *get\_data.py*
* *solve\_sliding\_parking\_lot.py*
* *test\_code.py*

1) main.py:

This is my main file, which needs to be start to run the project.

2) get\_data.py:

This file get the data from the selected file (all examples in folder files) and returns CARS\_IN\_PARKSLOTS and BLOCKING\_CARS.

3) solve\_sliding\_parking\_lot.py:

This file solves the problem and has the logic in it.

4) test\_code.py:

Tests the code with 30 tests.

——————————————————————————————————————

Main file gets CARS\_IN\_PARKSLOTS and BLOCKING\_CARS from the static method get\_data\_from\_file\_return\_necessary\_data from the class GetData. This static method gets as parameter the file\_path by the main file. The data in the file is saved by the variable data. CARS\_IN\_PARKSLOTS and BLOCKING\_CARS were returned by this method and created by data.

CARS\_IN\_PARKSLOTS is created by a list comprehension. It goes through the alphabet and add the char up to the last car char and stops then. This is accomplished by *takewhile* from the build in package *itertools*:

CARS\_IN\_PARKSLOTS = [car for car in

takewhile(lambda car: car != alphabet[last\_car\_index], alphabet)]

BLOCKING\_CARS is a dictionary with blocking\_cars char as keys and their positions as values in a list:

BLOCKING\_CARS = {cars[0]: [int(cars[1:]), int(cars[1:])+1]

for cars in data[2:]}

After collecting these two constants the main file generates a object of the class SolveSlidingParkingLot with CARS\_IN\_PARKSLOTS and BLOCKING\_CARS as parameters of the constructor.

In the constructor the attributes self.CARS\_IN\_PARKSLOTS and self.BLOCKING\_CARS are getting their values by the two parameters. In \_\_init\_\_ multiple attributes are getting also their values:

* self.BLOCKING\_CARS\_POSITIONS is a list with all blocked positions by the self.BLOCKING\_CARS.
* self.CARS\_NEED\_MOVES is a list created by list comprehension and is a collection of all self.CARS\_IN\_PARKSLOTS, which are blocked by self.BLOCKING\_CARS.
* self.result is unfilled list, for the final result
* self.car\_position is the position of the current car, which the algorithm is trying to find moves so the car can go out of the slot.
* self.parkslot\_border is a list with the positions of the end of the park slot

After that the main function will call the method assign\_moves\_to\_cars.

Here a for loop iterate through self.CARS\_IN\_PARKSLOTS and checks if the current car is in self.CARS\_NEED\_MOVES. If not it gets directly added to the result as: f"{car}:"

But if the current car is blocked the position of the current car gets saved in the variable self.car\_position.

The method check\_if\_car\_needs\_one\_or\_multiple\_moves gets called and checks if one or multiple moves are needed, that the current car can go out of the slot.

If the directly right slot is blocked and two right slots is free and it´s not out of self.parkslot\_border, then it returns "right".

Else if the directly left slot is blocked and two left slots is free and it´s not out of self.parkslot\_border, then it returns "left".

Else it returns "multiple".

If the return value of the method is not "multiple", the directly blocking car is determined by the method one\_move\_get\_blocking\_car. The method iterate through all self.BLOCKING\_CARS and checks if the self.car\_position is in the positions of the blocking car, if this is true the method returns the directly blocking car. After that it gets added to the result as:

f"{car}: {blocking\_car} 1 {move\_status}".

But if multiple moves are needed the multiple\_moves method gets called. This method coordinated other methods and their return values to get the final result for the current car to get out the slot.

First it calls get\_moves\_needed\_per\_side which return the moves needed per side for the directly blocking car to get out of the way of the slot.

It returns either [1, 2] or [2, 1]. The first digit is the moves for the left and the second the moves for the right side to get out of the way. This is accomplished by checking if the directly blocking car starts at the position of the current blocked car or if it ends their. Also the method returns the result twice. The second one is the backup, which is needed in the next method. This next method is find\_range\_of\_blocking\_cars\_and\_side.

In an infinite while loop gets every loop the position\_changer value increased by one. Every loop self.car\_position gets increased and decrease by the position\_changer to check if their is a free space on the left or right side. If there is a free space and it´s inside of the self.parkslot\_border then moves\_needed\_per\_side the return of the method before gets decreased on the side, with the free space. If one side is zero, so all free spaces are enough to get the directly blocking car out of the way, the loop breaks by returning the range, where all blocking cars need to move, the direction they need to move and the moves the directly blocking car needs to do. The last value is return by the second return of the method before, this was the backup for this return.

The last step is to get the blocking cars and their moves. Now the method get\_multiple\_cars\_need\_to\_move gets called. Now the directly\_blocking\_car

is created. And the blocking\_cars is filled with all cars which need to move. This list is created by a list comprehension which iterate through the blocking cars and checks if the positions of the car are in range\_of\_blocking\_cars\_and\_side[0] and that the car is one the right side of the car and is not the directly blocking car. After that the list gets cleaned, by removing the double cars.