Aufgabe 1: Schiebeparkplatz

Team-ID: 00155

Team: Opensourcehacker

Bearbeiter/-innen dieser Aufgabe:
Ron Jost

21. November 2021

Inhaltsverzeichnis

Lösungsidee

Die Lösungsidee, die ich zur Lösung der Aufgabe verwendet habe, war sehr intuitiv. Jedes der Querparkenden Autos hat prinzipiell zwei Möglichkeiten, eine normal, längst, parkendes Auto herauszulassen:

1. Es kann nach rechts verschoben werden, bis die Parkposition frei ist
2. Es kann nach links verschoben werden, bis die Parkposition

Um zu entscheiden, ob keine, nur eine der beiden Möglichkeiten oder beide Möglichkeiten in Betracht gezogen werden, betrachtet man zunächst die Querparkenden Autos und jeweils die Position des längst parkenden Autos. Wichtig ist, dass die Möglichkeit ein Querparkendes Auto nach links und dann ein anderes nach rechts zu verschieben, keinerlei Sinn hat und somit in einer Berechnung immer nur die Verschiebung nach rechts oder nach links betrachtet wird. Zur besseren Vorstellung, das Beispiel aus der Aufgabenstellung:

Normal parkende Autos: [A , B , C , D , E , F , G]

Querparkende Autos: [0 , 0 , H , H , 0 , I , I]

Die Autos A, B und E bedürfen keiner Maßnahmen, um auszuparken. Jedoch benötigen C, D, F und G Verschiebungen der Querparkenden Autos. Zu erst überprüft man, ob man das Querparkende Auto nach rechts verschieben kann, indem man überprüft, ob rechts von der jeweiligen Position

eine, respektive zwei Nullen, also nicht belegte Querparkpositionen befinden. Ob man nun eine freie Parklücke oder nicht braucht, lässt sich leicht feststellen und wird auch im nächsten Schritt gemacht. Sollte die Querparkende Position links von der freizumachenden Querparkposition vom selben Auto belegt sein, benötigt es zwei frei Parkplätze auf der rechten Seite, da das Auto zwei mal verschoben werden muss. Sollte die Querparkende Position links von der freizumachenden Querparkposition nicht vom selben Auto belegt sein, wird nur ein freier Parkplatz rechts benötigt. Die selbe Überprüfung wird für jedes auszuparkende Auto, bei dem ein Querparkendes Auto verschoben werden muss, auch für die Verschiebung nach links durchgeführt. Sollte nur die Verschiebung nach rechts möglich sein, werden die nötigen Verschiebungen berechnet und ausgegeben. Sollte nur die Verschiebung nach links möglich sein, werden die nötigen Verschiebungen berechnet und ausgegeben. Sollten beide Richtungen möglich sein, werden die Verschiebungen in beide Richtungen berechnet und diejenige, welche weniger Verschiebungen von Querparkenden Autos benötigt, wird ausgegeben. Doch wie werden diese Verschiebungen nach links und rechts eigentlich berechnet?

Das Verfahren zur Berechnung der benötigten Verschiebungen nach rechts und nach links sind prinzipiell identisch. Sollte nur eine Verschiebung nötig seien, wird in die jeweilige Richtung, links oder rechts, nach einer 0, also einem freien Parkplatz für Querparkende Autos, gesucht. Dann werden alle Querparkenden Autos, die zwischen der Position x des auszuparkenden Autos und der Position y des freien Querparkenden Platzes, eingeschlossen dem Querparkenden Auto, einmal in die jeweilig zu berechnende Richtung, also links oder rechts verschoben. Diese Verschiebungen werden für den finalen Schritt gespeichert.
Für den Fall, dass zwei Verschiebungen, des Querparkenden blockierenden Autos, nötig sein, wird zunächst in die jeweilige Richtung nach einem freien Querparkparkplatz gesucht. Nachdem einer gefunden wurde, wird nach einem zweiten gesucht. Sollten zwei freie gefunden werden, nutzt man den zweiten zur Berechnung. Es wird nun rückwärts gearbeitet. Alle Querparkenden Autos zwischen der Position x und y_2 werden nun so verschoben, dass zwei Nullen, also freie Parkplätze, nach rechts, respektive links, nebeneinander verfügbar sind und direkt neben der auszuparkenden Position sind. Dann wird das Querparkende Auto, welches die Poisition x blockiert, zwei mal verschoben und die Verschiebungen werden für den finalen Schritt gespeichert.

Final werden die Verschiebungen betrachtet und wie im Vorigen erläutert, gegebenenfalls die Menge an Verschiebungen zwischen links und rechts verglichen. Für jedes Auto, welches zum ausparken eine Verschiebung eines Querparkenden Autos benötigt, werden die idealen Verschiebungen ausgegeben.

Umsetzung

Das Programm wird in Python implementiert.

Dateneinlese und Verarbeitung

Zu Beginn wird die zu lösende Parkplatzdatei per Kommandozeilenparameter übergeben und aus der Datei entstehen zwei Array. Der erste Array beinhaltet die Normal parkenden Autos, der zweite die Querparkenden Autos. Die Arrays haben die selben Länge und sind parallel, das bedeutet, dass Array2 [3] den Querparkplatz vor Array1 [3] beschreibt.

Bestimmung der Verschiebungsrichtung

Nun wird für jede Position(x) in Array1, respektive dem Array *normal_park_liste_Buchstaben*, überprüft, ob eine Verschiebung eines Querparkenden Autos nötig ist. Dies wird überprüft, indem man den Wert y in dem Array, in der Position x, der Querparkenden Autos, *querpark_liste*, überprüft. Steht dort ein Buchstabe, ist eine Verschiebung zum ausparken notwendig, ist dort eine 0, ist keine Verschiebung notwendig. Sollte jedoch eine Verschiebing notwendig sein, wird nun die Richtung gecheckt. Sollte in *querpark_liste[x:]* keine 0 vorkommen, muss nach links verschoben werden. Sollte eine 0 vorkommen, wird eine Verschiebung nach rechts gecheckt. Sollte in *querpark_liste[:x]* keine 0 vorkommen, muss nach rechts verschoben. Kommt jedoch in beide Richtungen eine 0 vor, wird die Verschiebung sowohl nach rechts, als auch nach links überprüft. Verschiebungen nach links werden durch die Funktion *testing_nach_links(querpark_liste, position x)* berechnet und Verschiebungen nach rechts werden mithilfe der Funktion *testing_nach_rechts(querpark_liste, position x).*

Verschiebung nach links: testing_nach_links

Zu Beginn wird bestimmt, ob das Querparkende Auto auf der Position x ein oder zwei mal nach links bewegt werden muss. Sollte *querpark_liste[x+1] = querpark_liste[x]* sein, werden zwei Verschiebungen des Querparkenden Autos auf der Position x benötigt, andernfalls nur eine.
Sollte nur eine benötigt werden, wird mit Hilfe einer *while* Schleife solange die *querpark_liste* nach links durchsucht, bis eine freie Position, eine 0 gefunden wird. Von der Position x ausgehend wird ausgehend von *x = x -1*, als neue Startposition nun immer die Verschiebung *querpark_liste[x] = querpark_liste[x-1] und x = x- 2* durchgeführt, solange x> y. Währenddessen wird jede Verschiebung im Array *verschiebung* gespeichert.
Sollten zwei Verschiebungen des Querparkenden Autos auf der Position X nötig sein, ist das Verfahren ähnlich. Es wird nun der Array nach links durchsucht, bis zwei freie Positionen, zwei Nullen gefunden werden. Ausgehend davon, dass zwei Nullen gefunden werden, werden solange $x > y_2$ die Verschiebung durch *querpark_liste[x] = querpark_liste[x +1] und x = x -2* berechnet. Jedoch

würde nach der Logik auch einmal eine freie Stelle verschoben, diese wird jedoch übersprungen. Die Verschiebungen werden im selben Format im Array *verschiebung* gespeichert.

Verschiebung nach rechts: testing_nach_rechts

Die Verschiebung nach rechts erfolgt nach dem selben Prinzip, wie des im *testing_nach_links* beschrieben und verwendeten. Offensichtlich, wird jedoch dort der Array nach rechts durchsucht. Die Verschiebungen, falls nur eine Verschiebung notwendig ist ausgehend von $x = x+1$ und dann solange *x<y: querpark_liste[x] = querpark_liste[x-1]* und dann $x = x + 2$ berechnet. Falls zwei Verschiebungen nötig sind wird der selbe Algortihmus verwendet jedoch ist hier $y = y_2$ und man startet von x und nicht von x +1. Auch hier wird die freie Stelle y_1 übersprungen und die Verschiebungen werden im selben Format, wie bei *testing_nach_links* gespeichert.

Schlussendlich erfolgt die Programmausgabe, es wird die beste Verschiebung, also die Verschiebungen der Richtung, welche am wenigsten Verschiebung benötigt, ausgegeben.

Beispiele

$ python3 Aufgabe1.py parkplatz0.txt
A: Keine Verschiebung nötig
B: Keine Verschiebung nötig
C: H 1 rechts
D: H 1 links
E: Keine Verschiebung nötig
F: I 2 links
H 1 links
G: I 1 links

$ python3 Aufgabe1.py parkplatz1.txt
A: Keine Verschiebung nötig
B: O 1 rechts P 1 rechts
C: O 1 links
D: P 1 rechts
E: P 1 links O 1 links
F: Keine Verschiebung nötig
G: Q 1 rechts
H: Q 1 links
I: Keine Verschiebung nötig

J: Keine Verschiebung nötig
K: R 1 rechts
L: R 1 links
M: Keine Verschiebung nötig
N: Keine Verschiebung nötig

$ python3 Aufgabe1.py parkplatz2.txt
A: Keine Verschiebung nötig
B: Keine Verschiebung nötig
C: O 1 rechts
D: O 1 links
E: Keine Verschiebung nötig
F: P 1 rechts Q 1 rechts R 1 rechts
G: P 1 links
H: Q 1 rechts R 1 rechts
I: Q 1 links P 1 links
J: R 1 rechts
K: R 1 links Q 1 links P 1 links
L: Keine Verschiebung nötig
M: S 2 links R 1 links Q 1 links P 1 links
N: S 1 links

$ python3 Aufgabe1.py parkplatz3.txt
A: Keine Verschiebung nötig
B: O 1 rechts
C: O 1 links
D: Keine Verschiebung nötig
E: P 1 rechts
F: P 1 links
G: Keine Verschiebung nötig
H: Keine Verschiebung nötig
I: Q 2 links
J: Q 1 links
K: R 2 links Q 2 links
L: R 1 links Q 1 links
M: S 2 links R 2 links Q 2 links
N: S 1 links R 1 links Q 1 links

$ python3 Aufgabe1.py parkplatz4.txt
A: Q 1 rechts R 1 rechts
B: Q 2 rechts R 2 rechts
C: R 1 rechts
D: R 2 rechts
E: Keine Verschiebung nötig

F: Keine Verschiebung nötig
G: S 1 rechts
H: S 1 links
I: Keine Verschiebung nötig
J: Keine Verschiebung nötig
K: T 1 rechts
L: T 1 links
M: Keine Verschiebung nötig
N: U 1 rechts
O: U 1 links
P: Keine Verschiebung nötig

$ python3 Aufgabe1.py parkplatz5.txt

A: Keine Verschiebung nötig
B: Keine Verschiebung nötig
C: P 1 rechts Q 1 rechts
D: P 1 links
E: Q 1 rechts
F: Q 2 rechts
G: Keine Verschiebung nötig
H: Keine Verschiebung nötig
I: R 1 rechts
J: R 1 links
K: Keine Verschiebung nötig
L: Keine Verschiebung nötig
M: S 1 rechts
N: S 1 links
O: Keine Verschiebung nötig

Quellcode

```
# BWINF: Nr. 40
# Runde: 1
# Aufgabe: 1
# Autor: Ron Jost
# Team: Opensourcehacker (00155)

# Import von benötigten Bibliotheken
import sys
import copy

# Dateeineinlese:
```

```
dateiname = sys.argv[1]
datei = open(dateiname, 'r')
lines = datei.readlines()
lines_bearbeitet = []
for line in lines:
   lines_bearbeitet.append(line.replace('\n', ''))

# Verarbeitung
start_stop = lines_bearbeitet[0]
anz_quer = lines_bearbeitet[1]
quer_fahrezuge = lines_bearbeitet[2:]
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']

stop = Alphabet.index(start_stop[2]) + 1
normal_park_liste = list(range(stop))
normal_park_liste_buchstaben = []
for element in normal_park_liste:
   normal_park_liste_buchstaben.append(Alphabet[element])
querpark_liste = []
for i in normal_park_liste:
   querpark_liste.append(0)
quer_fahrezuge = lines_bearbeitet[2:]
for auto in quer_fahrezuge:
   pos = auto[2:]
   querpark_liste[int(pos)] = auto[0]
   querpark_liste[int(pos) + 1] = auto[0]



# Das blockierende Auto nach rechts wegbewegen
def testing_nach_rechts(array2, position):
   try:
      array_on = copy.deepcopy(array2)
      # Check, ob das querparkende Auto 2x mal nach rechts bewegt werden muss, um die position
frei zu machen.
      if array_on[position + 1] == array_on[position]:
         need_2 = False
      else:
         need_2 = True

      # Falls zwei Bewegungen nach rechts gesucht werden.
```

```
    if need_2:
        new_pos = position + 1
        verschiebe_ar = []

        # Es werden zwei freie Parkplätze rechts von der Position gesucht
        while new_pos <= len(array_on):
            if array_on[new_pos] == 0:
                if len(verschiebe_ar) == 1:
                    verschiebe_ar.append((array_on[new_pos - 1], new_pos))
                    break
                else:
                    verschiebe_ar.append((array_on[new_pos - 1], new_pos))
                    new_pos = new_pos + 1
            else:
                new_pos = new_pos + 1
        # Falls es keine 2 gibt, wird abgebrochen
        if len(verschiebe_ar) != 2:
            return (0, False)

        else:
            # Die nötigen Verschiebungen nach rechts werden berechnet
            verschiebung = []
            for element in verschiebe_ar:
                new_position = position
                backstart = element[1]
                while new_position < backstart:
                    if array_on[new_position - 1] == 0:
                        new_position = new_position + 1
                    else:
                        array_on[new_position] = array_on[new_position - 1]
                        verschiebung.append((array_on[new_position], '1 rechts'))
                        new_position = new_position +    2

    # Falls nur ein freier Parkplatz nach rechts benötigt wird: Auto steht mit der hinteren Hälfte auf der Position
    if need_2 == False:
        new_pos = position + 2
        verschiebe_ar = []
        # Der freie Parkplatz wird gesucht.
        while new_pos <= len(array_on):
            if array_on[new_pos] == 0:
                verschiebe_ar.append((array_on[new_pos - 1], new_pos))
```

```
                break
            else:
                new_pos = new_pos + 1
        back_start = verschiebe_ar[0][1]
        verschiebung = []
        new_position = position + 1
        # Die einzelnen Verschiebungen werden berechnet:
        while new_position < back_start:
            array_on[new_position] = array_on[new_position -1]
            verschiebung.append((array_on[new_position], '1 rechts'))
            new_position = new_position + 2
    return (len(verschiebung), verschiebung)
  except:
    return (0, False)

# Das blockierende Auto nach links wegbewegen
def testing_nach_links(array2, position):
  verschiebung = []
  array_on = copy.deepcopy(array2)
  # Check ob 2 Verschiebungen nach links nötig sind oder 1 reicht
  if position + 1 < len(array_on):
    if array_on[position + 1] == array_on[position]:
      need_2 = True
    else:
      need_2 = False
  else:
    need_2 = False

  # Falls 1 Verschiebung nach links langt
  if need_2 == False:
    new_pos = position - 2
    verschiebe_ar = []
    while new_pos >= 0:
      if array_on[new_pos] == 0:
        verschiebe_ar.append((array_on[new_pos + 1], new_pos))
        break
      else:
        new_pos = new_pos -1

    # Berechnung der Verschiebung nach Entdecken der freien Parklücke
    back_start = verschiebe_ar[0][1]
    verschiebung = []
```

```
        new_position = position - 1
        while new_position > back_start:
            array_on[new_position] = array_on[new_position +1]
            verschiebung.append((array_on[new_position], '1 links'))
            new_position = new_position -2

    # Falls zwei Parklücken benötigt werden
    if need_2:
        new_pos = position - 1
        verschiebe_ar = []
        # 2 freie Parklücken nach links werden gesucht
        while new_pos >= 0:
            if array_on[new_pos] == 0:
                if len(verschiebe_ar) == 1:
                    verschiebe_ar.append((array_on[new_pos + 1], new_pos))
                    break
                else:
                    verschiebe_ar.append((array_on[new_pos + 1], new_pos))
                    new_pos = new_pos - 1
            else:
                new_pos = new_pos - 1

        # Falls keine 2 freien Parkplätze gefunden wurden
        if len(verschiebe_ar) != 2:
            return (0, False)

        else:
            # Berechnung der benötigten Verschiebungen
            array_on = copy.deepcopy(array2)
            verschiebung = []
            for element in verschiebe_ar:
                new_position = position
                back_start = element[1]
                while new_position > back_start:
                    if array_on[new_position + 1] == 0:
                        new_position = new_position -1
                    else:
                        array_on[new_position] = array_on[new_position + 1]
                        verschiebung.append((array_on[new_position], '1 links'))
                        new_position = new_position -2
    return (len(verschiebung), verschiebung)
```

```
for x in range(0, len(querpark_liste)):
    score = False
    # Check ob man Querparkende Autos verschieben muss
    if querpark_liste[x] != 0:
        # Check ob man nach Rechts verschieben kann
        if x + 1 <= len(querpark_liste):
            # Check ob sich ein freier Parkplatz zum Verschieben nach rechts findet
            if 0 in querpark_liste[x:]:
                # Falls sich kein freier Parkplatz zum Verschieben nach links findet -> muss nach rechts
                if 0 not in querpark_liste[:x]:
                    rechts = testing_nach_rechts(querpark_liste, x)
                    rechts_string = str(normal_park_liste_buchstaben[x]) + ':'
                    rechts_dic = {}
                    for element in rechts[1]:
                        if element[0] in rechts_dic:
                            rechts_dic[element[0]] = rechts_dic[element[0]] + 1
                        else:
                            rechts_dic[element[0]] = 1
                    for element in rechts_dic:
                        rechts_string = rechts_string + ' ' + element + ' ' + str(rechts_dic[element]) + ' rechts'
                    print(rechts_string)
                # Falls man ebenso nach links verschieben kann
                if 0 in querpark_liste[:x]:
                    links = testing_nach_links(querpark_liste, x)
                    rechts = testing_nach_rechts(querpark_liste, x)
                    # Check ob nach links oder nach rechts mehr Verschiebungen benötigt:
                    # --> Das niedrigere wird ausgegeben
                    if type(rechts[1]) != bool:
                        if type(links[1]) != bool:
                            if rechts[0] <= links[0]:
                                rechts_string = str(normal_park_liste_buchstaben[x]) + ':'
                                rechts_dic = {}
                                for element in rechts[1]:
                                    if element[0] in rechts_dic:
                                        rechts_dic[element[0]] = rechts_dic[element[0]] + 1
                                    else:
                                        rechts_dic[element[0]] = 1
                                for element in rechts_dic:
                                    rechts_string = rechts_string + ' ' + element + ' ' + str(rechts_dic[element]) + '
rechts'
                                print(rechts_string)
                                score = True
```

```
                else:
                    rechts_string = str(normal_park_liste_buchstaben[x]) + ':'
                    rechts_dic = {}
                    for element in rechts[1]:
                        if element[0] in rechts_dic:
                            rechts_dic[element[0]] = rechts_dic[element[0]] + 1
                        else:
                            rechts_dic[element[0]] = 1
                    for element in rechts_dic:
                        rechts_string = rechts_string + ' ' + element + ' ' + str(
                            rechts_dic[element]) + ' rechts'
                    print(rechts_string)
                    score = True
            if score != True:
                links_string = str(normal_park_liste_buchstaben[x]) + ':'
                links_dic = {}
                for element in links[1]:
                    if element[0] in links_dic:
                        links_dic[element[0]] = links_dic[element[0]] + 1
                    else:
                        links_dic[element[0]] = 1
                for element in links_dic:
                    links_string = links_string + ' ' + element + ' ' + str(links_dic[element]) + ' links'
                print(links_string)
        # Falls ein Verschieben nach rechts nicht möglich ist, da man an der äußersten pPosition
angelangt ist
        else:
            links = testing_nach_links(querpark_liste, x)
            links_string = str(normal_park_liste_buchstaben[x]) + ':'
            links_dic = {}
            for element in links[1]:
                if element[0] in links_dic:
                    links_dic[element[0]] = links_dic[element[0]] + 1
                else:
                    links_dic[element[0]] = 1
            for element in links_dic:
                links_string = links_string + ' ' + element + ' ' + str(links_dic[element]) + ' links'
            print(links_string)
    if x + 1 > len(querpark_liste):
        links_string = str(normal_park_liste_buchstaben[x]) + ':'
        links_dic = {}
        for element in links[1]:
```

```
            if element[0] in links_dic:
                links_dic[element[0]] = links_dic[element[0]] + 1
            else:
                links_dic[element[0]] = 1
        for element in links_dic:
            links_string = links_string + ' ' + element + ' ' + str(links_dic[element]) + ' links'
        print(links_string)
# Falls keine Verschiebung nötig ist
else:
    print(normal_park_liste_buchstaben[x] + ': Keine Verschiebung nötig')
```