

# Aufgabe 1: Schiebeparkplatz

Team-ID: 00833

Team-Name: Gewinner bwinf

Bearbeiter/-innen dieser Aufgabe:  
Lukas Richter

22. November 2021

## Inhaltsverzeichnis

<b>1</b>	<b>Lösungsidee</b>	<b>1</b>
<b>2</b>	<b>Umsetzung</b>	<b>2</b>
<b>3</b>	<b>Beispiele</b>	<b>6</b>
<b>4</b>	<b>Quellcode</b>	<b>8</b>

## 1 Lösungsidee

Zuerst muss festgestellt werden, welche der normal stehenden Autos blockiert werden und durch welches quer stehende Auto diese Blockade verursacht wird. Außerdem muss ermittelt werden, welchen Bewegungsspielraum die quer stehenden Autos haben. Dabei ist mit dem Bewegungsspielraum die Anzahl der Felder gemeint, um die ein quer stehendes Auto von seinem Ausgangsstandpunkt maximal nach links beziehungsweise rechts bewegt werden kann, ohne noch ein weiteres Auto zu bewegen. Hierbei wird angenommen, dass es genau so viele Parkplätze in der Länge gibt, wie in der Länge normal stehende Autos stehen und dass diese Länge auch den Spielraum der quer stehenden Autos beschränkt.

Ist nun bekannt, durch welches quer stehende Auto ein normal stehendes Auto blockiert wird, so ist wichtig, zu wissen, wie weit das blockierende Auto nach rechts beziehungsweise links verschoben werden muss, damit das blockierte normal stehende Auto herausfahren kann. Da ein Auto nur 2 Felder lang ist, muss es nie mehr als 2 Felder in eine Richtung bewegt werden, um das blockierte Auto zu befreien. Außerdem gilt, dass wenn das blockierende Auto beispielsweise 2 Felder nach links fahren müsste, um das normal stehende Auto zu befreien, es dementsprechend nur 1 Feld nach rechts fahren müsste. Selbst dann, wenn das quer stehende Auto beispielsweise nur 1 Feld nach rechts, aber 2 Felder nach links verschoben werden müsste, müssen beide Fälle betrachtet werden. Der Grund hierfür ist, dass möglicherweise weniger Autos insgesamt verschoben werden, wenn man die Verschiebung nach links wählt, als wenn man jene nach rechts nutzt.

Als weiteres Axiom gilt, dass für die Befreiung eines Autos niemals sowohl eine Verschiebung nach rechts als auch eine nach links in derselben Lösung mit den wenigsten verschobenen Autos auftreten werden. Es werden innerhalb derselben Lösung entweder alle zu verschiebenden Autos nach links oder alle zu verschiebenden Autos nach rechts verschoben.

Um die optimale Lösung zu finden, wird prinzipiell folgendermaßen vorgegangen:

Falls der Spielraum des blockierenden Autos bereits die Verschiebung des blockierenden Autos um die jeweils nötige Anzahl an Feldern nach entweder rechts oder links zulässt, wird diese entsprechende Lösung gewählt. Ist sowohl der Spielraum nach rechts als auch nach links jeweils genug, wird die Richtung gewählt, in der die Anzahl an Feldern, um die das Auto verschoben werden muss, geringer ist.

Reicht hingegen keiner der Spielräume des blockierenden Autos allein aus, wird die Lösung, bei der das

blockierende Auto nach links verschoben wird und welche die geringste Anzahl insgesamt nach links zu verschiebender Autos hat, ermittelt. Gibt es eine solche Lösung nicht, wird dies während dieses Prozesses festgestellt. Hierzu werden solange von rechts nach links die Spielräume der quer stehenden Autos (deren Spielräume nach links), welche links vom blockierenden Auto stehen (dabei wird das blockierende Auto selber mit eingeschlossen und bei diesem begonnen) addiert, bis diese Summe von Spielräumen nach links größer oder gleich der Anzahl an Feldern ist, um welche das blockierende Auto nach links verschoben werden muss, um das blockierte Auto zu befreien.

Anschließend wird die Lösung, bei der das blockierende Auto nach rechts verschoben wird und welche die geringste Anzahl insgesamt nach rechts zu verschiebender Autos hat, ermittelt. Gibt es eine solche Lösung nicht, wird dies während dieses Prozesses festgestellt. Hierzu werden solange von links nach rechts die Spielräume der quer stehenden Autos (deren Spielräume nach rechts), welche rechts vom blockierenden Auto stehen (dabei wird das blockierende Auto selber mit eingeschlossen und bei diesem begonnen) addiert, bis diese Summe von Spielräumen nach rechts größer oder gleich der Anzahl an Feldern ist, um welche das blockierende Auto nach rechts verschoben werden muss, um das blockierte Auto zu befreien. Gibt es nun sowohl mit der Verschiebung nach links als auch mit der Verschiebung nach rechts jeweils eine Lösung, wird verglichen, bei welcher der Lösungen insgesamt weniger Autos verschoben werden müssen (ungeachtet der Anzahl der Felder, um die sie verschoben werden). Die Lösung mit der geringeren Anzahl zu verschiebender Autos wird gewählt. Ist diese Anzahl identisch, wird die Verschiebung nach links gewählt. Dass dies explizit links ist, ist rein willkürlich gewählt und hat keinen tieferen Sinn, als sich für eine der Lösungen zu entscheiden.

## 2 Umsetzung

```

1 parkplatz = open('parkplatz4.txt', "r").readlines()
2
3 erstes_auto = parkplatz[0][0]
4 letztes_auto = parkplatz[0][2]
5 anzahl_autos_normal = ord(letztes_auto) - ord(erstes_auto) + 1
6 anzahl_quer = int(parkplatz[1])
7
8
9 Quer = []
10 for i in range(anzahl_quer):
11     auto = parkplatz[2+i][0]
12     position = int(parkplatz[2+i][2:len(parkplatz[2+i])-1])
13     Quer += [[chr(i+anzahl_autos_normal+65)]+[position]+[position+1]]
14 # Quer: Liste aus Listen, welche jeweils den Namen eines querstehenden
15 # Autos und die 2 durch dieses Auto besetzten Felder beinhalten
16
17
18 blockierte_Autos = []
19 Blockierungen = []
20
21 for i in range(anzahl_autos_normal):
22     for k in Quer:
23         for j in k:
24             if j == i:
25                 blockierte_Autos += [chr(i+65)]
26                 Blockierungen += [[chr(i+65), "wird blockiert durch", k[0]]]
27             else:
28                 pass
29 # blockierte_Autos: Liste mit Namen aller
30 # blockierten normal stehenden Autos
31
32 # Blockierungen: Liste aller blockierten normalen Autos in Kombination
33 # mit dem Namen des Autos, durch welches sie blockiert werden

```

Abb. 1: Codezeilen 1 bis 33

Die Zeilen 1 bis 6 sind selbsterklärend. In Zeile 5 wird anhand der den Buchstaben der Namen der Autos im ASCII-System zugewiesenen Zahlen die Anzahl der normal stehenden Autos ausgerechnet.

Der in Zeile 9 als leer deklarierte Liste *Quer* wird in der darauf folgenden Schleife nun Listen als Elemente hinzugefügt, welche jeweils die Namen eines der querstehenden Autos sowie die Nummern der 2 Felder enthalten, die dieses Auto besetzt. Das erste (linke) der Felder wird dabei aus der entsprechenden Eingabedatei entnommen, während die Nummer des zweiten (des rechten) der Felder um 1 größer ist, als die des linken.

Nun wird ermittelt, welche der normalen Autos blockiert werden und durch welche der querstehenden Autos dies der Fall ist. Hierzu werden nacheinander die Zahlen 0 bis  $anzahl\_autos\_normal - 1$  ( $-1$ , da die Nummerierung der Felder bei 0 und nicht bei 1 beginnt) mit den Feldern der querstehenden Autos abgeglichen. Gleicht eine dieser Zahlen der Nummer eines der Felder eines der querstehenden Autos, so wird das Auto, dessen Buchstabe im ASCII-System mit  $(Nummer\ des\ Feldes) + 65$  nummeriert ist (65 ist die Nummerierung des großen A im ASCII-System, direkt danach folgenden alle weiteren Großbuchstaben des Alphabets), durch das entsprechende in der Schleife als  $k[0]$  bezeichnete Auto blockiert.

Die Liste *blockierte\_Autos* wäre nicht unbedingt nötig, sie dient lediglich der Vereinfachung eines späteren Teils des Codes. Der String, welcher am ersten Index jeder Liste in *Blockierungen* enthalten ist, diene lediglich der vereinfachten Lesbarkeit einiger Ausgaben bei der Programmierung. Mit Hilfe der Liste *Blockierungen* kann später nachvollzogen werden, durch welches querstehende Auto ein blockiertes normales Auto blockiert wird.

```

37 Spielraum = []
38
39 for i in range(anzahl_quer):
40     if i < len(Quer) - 1:
41         rechts = abs(Quer[i][2] - Quer[i+1][1]) - 1
42     else:
43         rechts = anzahl_autos_normal - 1 - Quer[i][2]
44     if i > 0:
45         links = Quer[i][1] - Quer[i-1][2] - 1
46     else:
47         links = Quer[i][1]
48     Spielraum += [[Quer[i][0], links, rechts]]
49 # Spielraum: Liste aus Listen, welche jeweils den Namen eines
50 # querstehenden Autos sowie dessen Spielraum nach rechts und
51 # links beinhalten

```

Abb. 2: Codezeilen 37 bis 51

Da nun die Namen und Felder der querstehenden Autos in den Listen in der Liste *Quer* enthalten sind, ist es möglich, die Spielräume der querstehenden Autos festzustellen. Hierbei hat das Programm die Limitierung, dass die querstehenden Autos in der Liste *Quer* in der Reihenfolge vorkommen müssen, in welcher sie auch von links nach rechts auf dem Parkplatz stehen.

Hierbei wird der Spielraum eines Autos nach links ermittelt, indem von der Nummer von dessen linken Feld die Nummer des rechten Feldes des vorherigen Autos subtrahiert wird. Zudem muss noch 1 subtrahiert werden, da bei Autos, die ohne Freiraum nebeneinander stehen, die Differenz 1 wäre, obwohl gar kein Spielraum besteht. Um dies zu verhindern, wird also 1 subtrahiert. Ein Sonderfall ist das von links nach rechts erste Auto, da dieses keinen Vorgänger hat. Dessen Spielraum nach links ist daher stattdessen die Nummer von dessen linken Feld.

Der Spielraum eines Autos nach rechts hingegen wird ermittelt, indem von der Nummer des rechten Feldes von dessen Vorgänger die Nummer des linken Feldes des betrachteten Autos subtrahiert wird. Zudem muss noch 1 subtrahiert werden, da bei Autos, die ohne Freiraum nebeneinander stehen, die Differenz 1 wäre, obwohl gar kein Spielraum besteht. Um dies zu verhindern, wird also 1 subtrahiert. Ein Sonderfall ist das von links nach rechts letzte Auto, da dieses keinen Vorgänger hat. Dessen Spielraum nach rechts ist daher stattdessen die Differenz von der Anzahl der normalen Autos und der Nummer von dessen rechten Feld. Außerdem muss hiervon wieder 1 subtrahiert werden, da die Anzahl der normalen Autos zwar  $x$  sein mag, die Nummerierung jedoch bei 0 beginnt und das äußerste rechte Feld dementsprechend mit  $x - 1$  nummeriert ist.

Sind nun die Spielräume eines Autos nach links sowie rechts bekannt, werden diese zusammen mit der Bezeichnung des entsprechenden Autos in einer Liste zusammengefasst und diese Liste wiederum der Liste *Spielraum* hinzugefügt.

```

55 index = -1
56 for i in range(anzahl_autos_normal):
57     if chr(i+65) in blockierte_Autos:
135
136     else:
137         print(chr(i+65) + ":")
138         #Ausgabe der nicht blockierten Autos

```

Abb. 3: Zeilen 55 bis 138, Zeilen 58 bis 134 ausgeblendet

Da nun die Spielräume der einzelnen querstehenden Autos bekannt sind, kann begonnen werden, die Schritte zur Befreiung der normal stehenden Autos auszugeben. Hierzu werden diese gemäß der Nummerierung ihrer Felder darauf überprüft, ob sie in der Liste *blockierte\_Autos* enthalten sind. Wenn sie nicht darin enthalten sind, sind sie nicht blockiert und können einfach herausfahren. Sind sie dort doch enthalten, so muss ein Weg gefunden werden, sie zu befreien.

```

55 index = -1
56 for i in range(anzahl_autos_normal):
57     if chr(i+65) in blockierte_Autos:
58         index += 1
59         blockiert = Blockierungen[index][0]
60         position = ord(blockiert) - 65
61
62         blockierend = Blockierungen[index][2]
63
64         for k in Quer:
65             if blockierend in k:
66                 if k[1] == position:
67                     links = 2
68                     rechts = 1
69                 else:
70                     links = 1
71                     rechts = 2
72                 blockierend = [blockierend, links, rechts]
73
74         for k in Spielraum:
75             if blockierend[0] in k:
76                 if blockierend[1] < blockierend[2]:
77                     if k[1] >= blockierend[1]:
78                         print(chr(i+65) + ": " + blockierend[0] + " " + str(blockierend[1]) + " links")
79                     elif k[2] >= blockierend[2]:
80                         print(chr(i+65) + ": " + blockierend[0] + " " + str(blockierend[2]) + " rechts")
81
82                 elif blockierend[1] > blockierend[2]:
83                     if k[2] >= blockierend[2]:
84                         print(chr(i+65) + ": " + blockierend[0] + " " + str(blockierend[2]) + " rechts")
85                     elif k[1] >= blockierend[1]:
86                         print(chr(i+65) + ": " + blockierend[0] + " " + str(blockierend[1]) + " links")
87
88                 if blockierend[1] > k[1] and blockierend[2] > k[2]:
132
133             else:
134                 pass
135
136     else:
137         print(chr(i+65) + ":")
138         #Ausgabe der nicht blockierten Autos

```

Abb. 4: Zeilen 55 bis 138, Zeilen 89 bis 131 ausgeblendet

Dieser Weg kann gefunden werden, indem durch die Variable *index* der Index Listeneintrags des Autos, welches in *blockierte\_Autos* vorkommt, bekannt ist. Da zu jedem Auto in *blockierte\_Autos* zum selben Index, welchen es in *blockierte\_Autos* hat, auch das zugehörige Listenelement in *Blockierungen* vorkommt, kann das blockierende Auto abgefragt und mit der Variable *blockierend* festgehalten werden. Auf die Position des blockierten Autos lässt sich durch dessen Bezeichnung schließen. Anschließend kann die zum blockierenden Auto gehörige Liste in *Quer* abgefragt werden, welche die 2 vom blockierenden Auto belegten Felder enthält. Sind diese Felder bekannt, ergibt sich, in welcher Richtung das blockierende Auto um nur 1 Feld verschoben werden müsste, um das blockierte Auto zu befreien und in welcher Richtung es dafür um 2 Felder verschoben werden müsste (dies passiert in den Zeilen 64 bis 72). Diese Informationen werden zusammen mit dem Namen des blockierenden Autos in einer Liste zusammengefasst, welche der Variablen *blockierend* zugewiesen wird.

Befindet sich mindestens eine dieser minimalen Verschiebungen innerhalb des Spielraums des blockierenden Autos, wird in der Ausgabe die Verschiebung des blockierenden Autos um die minimale nötige Anzahl an Feldern mit der entsprechenden Richtung ausgegeben (Zeilen 74 bis 134)

```

88     if blockierend[1] > k[1] and blockierend[2] > k[2]:
89         nummer_quer = ord(blockierend[0]) - 65 - anzahl_autos_normal
90         links = [0, ""]
91         rechts = [0, ""]
92         beste_loesung = ""
93
94         zu_bewegen = 0
95         for j in range(nummer_quer+1):
96             if links[0] < blockierend[1]:
97                 if nummer_quer-j < 0:
98                     pass
99                 else:
100                     zu_bewegen += 1
101                     string_links = chr(ord(blockierend[0])-j) + " " + str(blockierend[1]-links[0]) + " links, " + links[1]
102                     links_neu = int(links[0] + Spielraum[nummer_quer-j][1])
103                     links = [links_neu, string_links, zu_bewegen]
104
105         zu_bewegen = 0
106         for j in range(nummer_quer, anzahl_quer):
107             if rechts[0] < blockierend[2]:
108                 if nummer_quer+j > anzahl_quer-1:
109                     pass
110                 else:
111                     zu_bewegen += 1
112                     string_rechts = chr(ord(blockierend[0])+j) + " " + str(blockierend[2]-rechts[0]) + " rechts, " + rechts[1]
113                     rechts_neu = rechts[0] + Spielraum[nummer_quer+j][2]
114                     rechts = [rechts_neu, string_rechts, zu_bewegen]
115

```

Abb. 5: Codezeilen 88 bis 115

Reicht der Spielraum des blockierenden allein Autos nicht aus, wird in den Zeilen 88 bis 115 weiter nach

der optimalen Möglichkeit zur Befreiung des blockierten Autos gesucht. Hierzu wird zu Beginn in Zeile 89 die Nummer des blockierenden Autos, welche auch dessen Index in der Liste *Quer* und der Index von dessen zugehöriger Liste in *Spielraum* ermittelt (die Nummerierung der quer stehenden Autos beginnt also bei 0 und verläuft von links nach rechts). Anschließend werden 2 Listen *links* und *rechts* angelegt, welche dazu dienen, die nach den einzelnen Iterationen der folgenden Schleifen erreichten Beträge der Bewegungen in der entsprechenden Richtung sowie den String mit den Anweisungen für diese Bewegungen zu beinhalten. Der String *beste\_Lsung* wird erst später relevant. Mit der Variable *zu\_bewegen* wird die Anzahl der Autos gezählt, welche insgesamt bewegt werden müssen, um die Bewegung des blockierenden Autos nach links beziehungsweise rechts um den nötigen Betrag zu erreichen.

In den Zeilen 95 bis 103 wird nun nach einer Lösung gesucht, bei der das blockierende Auto nach links verschoben wird. Hierzu werden, das blockierende Auto selbst einschließend und bei diesem beginnend, nun der Reihenfolge vom blockierenden Auto aus nach links bis zum äußersten linken queren Auto hin die Spielräume aller queren Autos, welche diese nach links haben, addiert. Dies stoppt, wenn entweder die Summe dieser Spielräume nach links größer oder gleich der Anzahl an Feldern ist, um die das blockierende Auto nach links verschoben werden muss, um das blockierte Auto zu befreien oder wenn das äußerste linke quere Auto erreicht ist. Außerdem wird jedes Mal, wenn ein weiteres Auto hinzu kommt, was hierzu bewegt werden muss, die Variable *zu\_bewegen* um 1 erhöht. Darüber hinaus wird der String *string\_links* erzeugt, indem an den String, welcher bisher das zweite Element der Liste *links* war, vorn die Anweisung des entsprechenden zu bewegendes Autos angehängt wird (wenn es denn nötig ist, dieses Auto zu bewegen). Wurde die neue Summe *links\_neu* der Spielräume nach links ausgerechnet und der String *string\_links* erzeugt, so werden *links\_neu*, *string\_links* und *zu\_bewegen* in dieser Reihenfolge die bisherigen Elemente der Liste *links* ersetzen. Zu Beginn der nächsten Iteration der in Zeile 95 beginnenden Schleife wird überprüft, ob entweder die Summe der Spielräume schon groß genug oder bereits das äußerste linke Auto erreicht ist. Ist mindestens eines davon der Fall, wird die Liste *links* nicht weiter verändert. Anderenfalls wird das beschriebene Vorgehen wiederholt.

Analog hierzu lässt sich das Vorgehen in den Zeilen 105 bis 114 zur Ermittlung der optimalen Lösung, bei welcher das blockierende Auto nach rechts verschoben wird, erklären. Lediglich wird hier von links nach rechts, das blockierende Auto einschließend und bei diesem beginnend, vorgegangen:

Hierzu werden, das blockierende Auto selbst einschließend und bei diesem beginnend, nun der Reihenfolge vom blockierenden Auto aus nach rechts bis zum äußersten rechten queren Auto hin die Spielräume aller queren Autos, welche diese nach rechts haben, addiert. Dies stoppt, wenn entweder die Summe dieser Spielräume nach rechts größer oder gleich der Anzahl an Feldern ist, um die das blockierende Auto nach rechts verschoben werden muss, um das blockierte Auto zu befreien oder wenn das äußerste rechte quere Auto erreicht ist. Außerdem wird jedes Mal, wenn ein weiteres Auto hinzu kommt, was hierzu bewegt werden muss, die Variable *zu\_bewegen* um 1 erhöht. Darüber hinaus wird der String *string\_rechts* erzeugt, indem an den String, welcher bisher das zweite Element der Liste *rechts* war, vorn die Anweisung des entsprechenden zu bewegendes Autos angehängt wird (wenn es denn nötig ist, dieses Auto zu bewegen). Wurde die neue Summe *rechts\_neu* der Spielräume nach links ausgerechnet und der String *string\_rechts* erzeugt, so werden *rechts\_neu*, *string\_rechts* und *zu\_bewegen* in dieser Reihenfolge die bisherigen Elemente der Liste *rechts* ersetzen. Zu Beginn der nächsten Iteration der in Zeile 106 beginnenden Schleife wird überprüft, ob entweder die Summe der Spielräume schon groß genug oder bereits das äußerste rechte Auto erreicht ist. Ist mindestens eines davon der Fall, wird die Liste *rechts* nicht weiter verändert. Anderenfalls wird das beschriebene Vorgehen wiederholt.

```

116         if links[0] >= blockierend[1] and rechts[0] >= blockierend[2]:
117             if links[2] <= rechts[2]:
118                 beste_lösung = links[1][0:len(links[1])-2]
119             elif links[2] > rechts[2]:
120                 beste_lösung = rechts[1][0:len(rechts[1])-2]
121
122         elif links[0] >= blockierend[1]:
123             beste_lösung = links[1][0:len(links[1])-2]
124
125         elif rechts[0] >= blockierend[2]:
126             beste_lösung = rechts[1][0:len(rechts[1])-2]
127
128         elif links[0] < blockierend[1] and rechts[0] < blockierend[2]:
129             beste_lösung = "Kann nicht ausgeparkt werden"
130
131         print(chr(i+65) + ": " + beste_lösung)

```

Abb. 6: Zeilen 116 bis 131

Wurde nun sowohl die in Zeile 95 als auch die in Zeile 106 beginnende Schleife beendet, kann die optimale Lösung ermittelt werden. Ist das erste Element der Liste *links* größer oder gleich der Anzahl der zur Befreiung des blockierten Autos minimal nötigen Verschiebung des blockierenden Autos nach links, gibt es

eine Lösung mit Verschiebung des blockierenden Autos nach links. Ist das erste Element der Liste *rechte* größer oder gleich der Anzahl der zur Befreiung des blockierten Autos minimal nötigen Verschiebung des blockierenden Autos nach rechts, gibt es eine Lösung mit Verschiebung des blockierenden Autos nach rechts.

Gibt es sowohl eine Lösung mit Verschiebung des blockierenden Autos nach links als auch eine mit Verschiebung des blockierenden Autos nach rechts, wird die Lösung gewählt, für welche insgesamt weniger Autos verschoben werden müssen. Müssen bei beiden Lösungen gleich viel Autos verschoben werden, wird die Lösung mit Verschiebung des blockierenden Autos nach links gewählt. Es wäre auch möglich, in diesem Fall in nächster Instanz die Gesamtzahl der Felder zu vergleichen, um die man bei den Lösungen jeweils insgesamt Autos verschieben muss, jedoch wird dies in der Aufgabenstellung nicht gefordert.

Gibt es nur für eine der Richtungen eine Lösung, so ist dies auch die optimale Lösung. Gibt es für keine der Richtungen eine Lösungen, so wird ausgegeben, dass es keine Möglichkeit gibt, das blockierte Auto auszuparken. Zum Schluss wird die ermittelte Lösung (oder auch die Aussage, dass es keine Lösung gibt), ausgegeben.

### 3 Beispiele

parkplatz0:

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

parkplatz1:

```
A:
B: P 1 rechts, O 1 rechts
C: O 1 links
D: P 1 rechts
E: Q 2 rechts
F:
G: Q 1 rechts
H: Q 1 links
I:
J:
K: R 1 rechts
L: R 1 links
M:
N:
```

parkplatz2:

```
A:
B:
C: O 1 rechts
D: O 1 links
E:
F: O 1 links, P 2 links
G: P 1 links
H: O 1 links, P 2 links, Q 2 links
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:

```
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:

```
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:
```

parkplatz5:

```
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:
```

Ein Sonderfall wäre es, wenn querstehende Autos "ineinander" stehen, also beispielsweise in der Eingabedatei als Position für ein querstehendes Auto 5 und für das nächste 6 angegeben wird, wodurch beide Autos die Position 6 besetzen würden. Zwar wäre ein solches "Ineinanderstehen" ein Fehler der Eingabedatei, da dies ganz offensichtlich nicht möglich ist, es könnte jedoch als Schwäche des Programms bezeichnet werden, dass dennoch eine (völlig abwegige) Ausgabe erzeugt wird, anstatt den Fehler zu erkennen und auf diesen hinzuweisen.

**Beispiel hierfür:**

Eingabe:	Ausgabe:
<pre> 1  A G 2  2 3  H 5 4  I 5 5 </pre>	<pre> A: B: C: D: E: F: H 2 links G: H 4 links, I 2 links </pre>

Ein anderer Sonderfall wäre es, wenn es  $x$  normal stehende Autos und demnach auch  $x$  Felder gibt, von den  $x$  Feldern für querstehende Autos aber  $x - 1$  oder gar alle  $x$  Felder mit querstehenden Autos besetzt sind. Im ersten dieser Fälle können nur die normalen Autos auf Feldern mit geraden Nummern (wozu hierbei auch das 0. Feld gezählt wird) ausgeparkt werden, im letzten Fall kann gar kein normal stehendes Auto ausgeparkt werden:

**Beispiel für  $x$  normal stehende Autos und  $x - 1$  von querstehenden Autos besetzte Felder**  
 $(x - 1 = 2 \cdot \frac{\text{Felder}}{\text{querstehendes Auto}} \cdot 3 \text{ querstehende Autos} = 6 \text{ von insgesamt } x = 7 \text{ Feldern besetzt}):$

Eingabe:	Ausgabe:
<pre> 1  A G 2  3 3  H 0 4  I 3 5  J 5 6 </pre>	<pre> A: H 1 rechts B: Kann nicht ausgeparkt werden C: D: Kann nicht ausgeparkt werden E: I 1 links F: Kann nicht ausgeparkt werden G: I 1 links, J 1 links </pre>

**Beispiel für  $x$  normal stehende Autos und  $x$  von querstehenden Autos besetzte Felder**  
 $(6 = x = 2 \cdot \frac{\text{Felder}}{\text{querstehendes Auto}} \cdot 3 \text{ querstehende Autos} = 6 \text{ von } x = 6 \text{ Feldern besetzt}):$

Eingabe:	Ausgabe:
<pre> 1  A F 2  3 3  G 0 4  H 2 5  I 4 6 </pre>	<pre> A: Kann nicht ausgeparkt werden B: Kann nicht ausgeparkt werden C: Kann nicht ausgeparkt werden D: Kann nicht ausgeparkt werden E: Kann nicht ausgeparkt werden F: Kann nicht ausgeparkt werden </pre>

Natürlich gibt es noch diverse weitere Sonderfälle durch schlichtweg fehlerhafte Eingabedateien, jedoch liegt das dementsprechend an der Eingabedatei und wird hier daher nicht näher betrachtet.



## 4 Quellcode

```

1  parkplatz = open('parkplatz5.txt', "r").readlines()
2
3  erstes_auto = parkplatz[0][0]
4  letztes_auto = parkplatz[0][2]
5  anzahl_autos_normal = ord(letztes_auto) - ord(erstes_auto) + 1
6  anzahl_quer = int(parkplatz[1])
7
8
9  Quer = []
10 for i in range(anzahl_quer):
11     auto = parkplatz[2+i][0]
12     position = int(parkplatz[2+i][2:len(parkplatz[2+i])-1])
13     Quer += [[chr(i+anzahl_autos_normal+65)]+[position]+[position+1]]
14 # Quer: Liste aus Listen, welche jeweils den Namen eines querstehenden
15 # Autos und die 2 durch dieses Auto besetzten Felder beinhalten
16
17
18 blockierte_Autos = []
19 Blockierungen = []
20
21 for i in range(anzahl_autos_normal):
22     for k in Quer:
23         for j in k:
24             if j == i:
25                 blockierte_Autos += [chr(i+65)]
26                 Blockierungen += [[chr(i+65), "wird blockiert durch", k[0]]]
27             else:
28                 pass
29 # blockierte_Autos: Liste mit Namen aller
30 # blockierten normal stehenden Autos
31
32 # Blockierungen: Liste aller blockierten normalen Autos in Kombination
33 # mit dem Namen des Autos, durch welches sie blockiert werden
34
35
36
37 Spielraum = []
38
39 for i in range(anzahl_quer):
40     if i < len(Quer) - 1:
41         rechts = abs(Quer[i][2] - Quer[i+1][1]) - 1
42     else:
43         rechts = anzahl_autos_normal - 1 - Quer[i][2]
44     if i > 0:
45         links = Quer[i][1] - Quer[i-1][2] - 1
46     else:
47         links = Quer[i][1]
48     Spielraum += [[Quer[i][0], links, rechts]]
49 # Spielraum: Liste aus Listen, welche jeweils den Namen eines
50 # querstehenden Autos sowie dessen Spielraum nach rechts und
51 # links beinhalten
52
53
54
55 index = -1
56 for i in range(anzahl_autos_normal):
57     if chr(i+65) in blockierte_Autos:
58         index += 1
59         blockiert = Blockierungen[index][0]
60         position = ord(blockiert) - 65
61
62         blockierend = Blockierungen[index][2]
63

```



```

64     for k in Quer:
65         if blockierend in k:
66             if k[1] == position:
67                 links = 2
68                 rechts = 1
69             else:
70                 links = 1
71                 rechts = 2
72             blockierend = [blockierend, links, rechts]
73
74     for k in Spielraum:
75         if blockierend[0] in k:
76             if blockierend[1] < blockierend[2]:
77
78                 elif blockierend[1] > blockierend[2]:
79
80             if blockierend[1] > k[1] and blockierend[2] > k[2]:
81                 nummer_quer = ord(blockierend[0])-65-anzahl_autos_normal
82                 links = [0, ""]
83                 rechts = [0, ""]
84                 beste_lösung = ""
85
86                 zu_bewegen = 0
87                 for j in range(nummer_quer+1):
88                     if links[0] < blockierend[1]:
89                         if nummer_quer-j < 0:
90                             pass
91                         else:
92                             zu_bewegen += 1
93                             string_links = chr(ord(blockierend[0])-j) + " " + str(blockierend[1]-links[0]) + " links, " + links[1]
94                             links_neu = int(links[0] + Spielraum[nummer_quer-j][1])
95                             links = [links_neu, string_links, zu_bewegen]
96
97                 zu_bewegen = 0
98                 for j in range(nummer_quer, anzahl_quer):
99                     if rechts[0] < blockierend[2]:
100                         if nummer_quer+j > anzahl_quer-1:
101                             pass
102                         else:
103                             zu_bewegen += 1
104                             string_rechts = chr(ord(blockierend[0])+j) + " " + str(blockierend[2]-rechts[0]) + " rechts, " + rechts[1]
105                             rechts_neu = rechts[0] + Spielraum[nummer_quer+j][2]
106                             rechts = [rechts_neu, string_rechts, zu_bewegen]
107
108                 if links[0] >= blockierend[1] and rechts[0] >= blockierend[2]:
109                     if links[2] <= rechts[2]:
110                         beste_lösung = links[1][0:len(links[1])-2]
111                     elif links[2] > rechts[2]:
112                         beste_lösung = rechts[1][0:len(rechts[1])-2]
113
114                 elif links[0] >= blockierend[1]:
115                     beste_lösung = links[1][0:len(links[1])-2]
116
117                 elif rechts[0] >= blockierend[2]:
118                     beste_lösung = rechts[1][0:len(rechts[1])-2]
119
120                 elif links[0] < blockierend[1] and rechts[0] < blockierend[2]:
121                     beste_lösung = "Kann nicht ausgeparkt werden"
122
123                 print(chr(i+65) + ": " + beste_lösung)
124
125     else:
126         pass
127
128 else:
129     print(chr(i+65) + ":")
130     #Ausgabe der nicht blockierten Autos
131
132
133
134
135
136
137
138

```