Aufgabe 1: Schiebeparkplatz

Team-ID: 00195

Team-Name: WLR

Bearbeiter/-innen dieser Aufgabe:
Baran Peters

21. November 2021

Inhaltsverzeichnis

1 Lösungsidee

Jedes geparkte Auto kann einer von drei Gruppen zugeordnet werden:

- Freistehend, also, dass keines der beweglichen Autos dieses blockiert und daher ohne jegliche Umstellung herausfahren kann.
- Blockiert durch ein querstehendes Auto, aber kann durch dessen Umstellung herausfahren.
- Blockiert durch ein querstehendes Auto, aber kann durch dessen Umstellung nicht herausfahren.

Bei der Zuordnung eines Autos in diese Gruppen kann man vorerst überprüfen, ob es überhaupt blockiert wird. Falls nicht, kann mit dem nächsten geparkten Auto verfahren. Falls ja, muss man nun eine (möglichst effiziente) Kombination der beweglichen Autos bestimmen, wodurch das blockierte Feld frei wird und das Auto herausfahren kann. Ein Weg, wie man dieses Problem lösen kann, werde ich im Folgenden erklären:

Das Ziel ist es, das blockierende Auto wegzubewegen, sodass das geparkte Auto herausfahren kann. Dies kann nun erfolgen, indem sich das blockierende Auto entweder nach links oder rechts bewegt. Die Anzahl der Felder, die sich das Auto verschieben muss, und Anzahl der Autos, die sich verschieben müssen, variieren je nachdem, in welche Richtung das Auto sich bewegt.

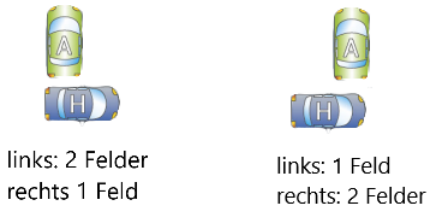

Abbildung 1: Visualisierung der unterschiedlichen Anzahl von zu bewegenden Feldern

Egal wie das querstehende Auto steht, wird es, wie in Abbildung 1 dargestellt, immer eine Bewegungsrichtung geben, mit welcher man immer zwei Felder im Gegensatz zu einem Feld zurücklegen muss, da ein Auto zwei Felder einnimmt. Nun lassen sich theoretisch zwei mögliche Kombinationen von Verschiebungen mehrerer/eines Autos finden, einmal wenn das blockierende Auto nach rechts beziehungsweise links hin Platz macht. Daher sind in diesen Kombinationen auch nur Autos zu finden, die sich vom blockierenden Auto aus in der Bewegungsrichtung befinden. Bei der Bestimmung dieser Kombinationen muss man ausgehend vom blockierenden Auto immer das äußerste (bewegliche) Auto bestimmen, welches sich mit der jeweiligen Felderanzahl nach außen hin bewegen kann, sodass die restlichen Autos inklusive des blockierenden Autos aufrücken können. Allerdings ist es nicht sicher immer zwei Kombinationen zu finden, da es sein kann, dass in eine Richtung hin die Autos aufgrund von räumlicher Begrenzung des Parkplatzes sich nicht mehr bewegen können, oder das blockierende Auto schon am Rand des Parkplatzes steht.

Falls man nun zwei verschiedene Möglichkeiten hat, das blockierte Feld zu öffnen, sollte man hier ausgehend von der Aufgabenstellung die Kombination nehmen, an welcher die kleinste Anzahl an Autos beteiligt sind, da dies in einer realen Anwendung auch die schnellere sowie effizientere Variante wäre. Falls beide Kombinationen die gleiche Anzahl an Autos bewegen würden, wird die Variante, mit der kleinsten Anzahl an genutzten Feldern, ausgewählt.

2 Umsetzung

Die Lösungsidee wird in einem Programm der Sprache JavaScript umgesetzt. Voraussetzung ist hierbei die Nutzung der neusten Node.JS Laufzeitumgebung. Nachdem die Daten über die Kommandozeile eingelesen werden, beginnt der hauptsächliche Teil des Programms.

Vorerst wird über alle stehenden Autos iteriert, mit dem Hintergedanken, dass, falls das hinter dem Auto stehende Feld blockiert sein sollte, die benötigten Verschiebungen der querstehenden Autos mithilfe von `findUnblockCombo()` bestimmt werden können und im Nachhinein dem Nutzer über die Konsole ausgegeben werden können. Die Reihenfolge der ausgegebenen benötigten Verschiebungen entspricht der Reihenfolge der Verschiebungen in der realen Anwendung, also, falls vorhanden, von dem äuSSersten Auto zum eigentlich blockierenden Auto. In `findUnblockCombo()` werden für ein Ausweichen nach links sowie rechts die benötigten Verschiebungen bestimmt. Es wird erstmal überprüft, ob eine Bewegung außerhalb des Parkplatzes enden würde. Falls nicht, wird nun die Kette der Autos bestimmt, die sich bewegen muss. Das geschieht, indem man ausgehend von dem blockierenden Auto schaut, ob ein weiter außen stehendes Auto dessen Verschiebung blockieren würde. Falls ja, wird überprüft, ob man noch ein Feld zwischen diesem Auto und dem derzeitigen Ende der Kette existiert, wodurch die Autos nach diesem in der Kette weniger Raum benötigen, da sie sich ein Feld weniger bewegen müssen. Falls kein blockierendes Auto gefunden wird, überprüft man, ob durch die Verschiebung das Ende des Parkplatzes überschritten werden würde, wodurch auch zumindest diese Kombinationen für links oder rechts nicht vollständig wäre und daher ungültig ist. Weiterhin werde dann, ausgehend von dem Feld des äußersten Autos bis hin zum blockierenden Auto, alle Autos, die sich auch bewegen müssen, mit ihrer jeweiligen Felderanzahl, die sie sich bewegen müssen, in `moveCombinations` gespeichert.

Zuletzt wird dann die bessere Kombination aus `moveCombinations` anhand der in der Lösungsidee genannten Kriterien bestimmt und von der Funktion zurückgegeben.

3 Beispiele

- Die Ausgabe besteht, wie im Beispiel angegeben, aus dem jeweiligen Buchstaben des ausparkenden Autos, dann dem Buchstaben des blockierenden Autos, der Anzahl der Felder, die es sich bewegen muss und sowie die Richtung.
- Darüber hinaus wurden hier zwei weitere Beispiele angeführt. parkplatz6.txt stellt einen komplett zugeparkten Raum überhalb der regulär geparkten Autos dar, wodurch kein Auto mehr ausparken kann. parkplatz7.txt dahingehen den Fall, dass nur ein Feld offen ist, aber dadurch auch bestimmte Autos auch durch eine Verschiebung nicht herausfahren können. Beide Fälle sollten in einer Realanwendung vermieden werden.

3.1 Beispieldaten Nr. 1

Eingabe: parkplatz0.txt aus den Beispiel-Eingaben
Ausgabe:

```
A: Kein Bewegen nötig
B: Kein Bewegen nötig
C: H 1 Rechts
D: H 1 Links
E: Kein Bewegen nötig
F: H 1 Links, I 2 Links
G: I 1 Links
```

3.2 Beispieldaten Nr. 2

Eingabe: parkplatz1.txt aus den Beispiel-Eingaben
Ausgabe:

```
A: Kein Bewegen nötig
B: P 1 Rechts, O 1 Rechts
C: O 1 Links
D: P 1 Rechts
E: O 1 Links, P 1 Links
F: Kein Bewegen nötig
G: Q 1 Rechts
H: Q 1 Links
I: Kein Bewegen nötig
J: Kein Bewegen nötig
K: R 1 Rechts
L: R 1 Links
M: Kein Bewegen nötig
N: Kein Bewegen nötig
```

3.3 Beispieldaten Nr. 3

Eingabe: parkplatz2.txt aus den Beispiel-Eingaben
Ausgabe:

```
A: Kein Bewegen nötig
B: Kein Bewegen nötig
C: O 1 Rechts
D: O 1 Links
E: Kein Bewegen nötig
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: Kein Bewegen nötig
M: P 1 Links, Q 1 Links, R 1 Links, S 2 Links
N: S 1 Links
```

3.4 Beispieldaten Nr. 4

Eingabe: parkplatz3.txt aus den Beispiel-Eingaben
Ausgabe:

```
A: Kein Bewegen nötig
B: O 1 Rechts
C: O 1 Links
D: Kein Bewegen nötig
E: P 1 Rechts
F: P 1 Links
G: Kein Bewegen nötig
H: Kein Bewegen nötig
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
```

3.5 Beispieldaten Nr. 5

Eingabe: parkplatz4.txt aus den Beispiel-Eingaben
Ausgabe:

```
A: R 1 Rechts, Q 1 Rechts
B: R 2 Rechts, Q 2 Rechts
C: R 1 Rechts
D: R 2 Rechts
E: Kein Bewegen nötig
F: Kein Bewegen nötig
G: S 1 Rechts
H: S 1 Links
I: Kein Bewegen nötig
J: Kein Bewegen nötig
K: T 1 Rechts
L: T 1 Links
M: Kein Bewegen nötig
N: U 1 Rechts
O: U 1 Links
P: Kein Bewegen nötig
```

3.6 Beispieldaten Nr. 6

Eingabe: parkplatz5.txt aus den Beispiel-Eingaben
Ausgabe:

```
A: Kein Bewegen nötig
B: Kein Bewegen nötig
C: P 2 Links
D: P 1 Links
E: Q 1 Rechts
F: Q 2 Rechts
G: Kein Bewegen nötig
H: Kein Bewegen nötig
I: R 1 Rechts
J: R 1 Links
K: Kein Bewegen nötig
L: Kein Bewegen nötig
M: S 1 Rechts
N: S 1 Links
O: Kein Bewegen nötig
```

3.7 Beispieldaten Nr. 7

Eingabe: parkplatz6.txt (im Aufgabenordner)
Ausgabe:

```
A: Keine Ausfahrt möglich
B: Keine Ausfahrt möglich
C: Keine Ausfahrt möglich
D: Keine Ausfahrt möglich
E: Keine Ausfahrt möglich
F: Keine Ausfahrt möglich
G: Keine Ausfahrt möglich
H: Keine Ausfahrt möglich
I: Keine Ausfahrt möglich
J: Keine Ausfahrt möglich
K: Keine Ausfahrt möglich
L: Keine Ausfahrt möglich
M: Keine Ausfahrt möglich
N: Keine Ausfahrt möglich
```

3.8 Beispieldaten Nr. 8

Eingabe: parkplatz7.txt (im Aufgabenordner)
Ausgabe:

```
A: Kein Bewegen nötig
B: Keine Ausfahrt möglich
C: O 1 Links
D: Keine Ausfahrt möglich
E: O 1 Links, P 1 Links
F: Keine Ausfahrt möglich
G: O 1 Links, P 1 Links, Q 1 Links
H: Keine Ausfahrt möglich
I: O 1 Links, P 1 Links, Q 1 Links, U 1 Links
J: Keine Ausfahrt möglich
K: O 1 Links, P 1 Links, Q 1 Links, U 1 Links, R 1 Links
L: Keine Ausfahrt möglich
M: O 1 Links, P 1 Links, Q 1 Links, U 1 Links, R 1 Links, V 1 Links
N: Keine Ausfahrt möglich
O: O 1 Links, P 1 Links, Q 1 Links, U 1 Links, R 1 Links, V 1 Links, S 1 Links
```

4 Quellcode

```
const alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
var stationaryCars;

function main() {
    // Geparkten Autos
    stationaryCars = readline().split("␣")

    // Verschiebbaren Autos, die waagerecht zu den anderen Autos stehen
    var moveableCars = []

    // Anzahl der verschiebbaren Autos
    let numMCars = +readline();

    for(let i = 0; i < numMCars; i++){
        let car = readline().split("␣")

        car[1] = +car[1]

        moveableCars.push(car)
    }

    for(let i = 0; i < alphabet.indexOf(stationaryCars[1]) - alphabet.indexOf(
    stationaryCars[0]) + 1; i++){
```

```
        // Auto, welches hinter dem ausfahrenden Auto möglicherweise steht
        let blockingCar = moveableCars.find(car => car[1] == i || car[1] + 1 == i)
        if(blockingCar){
            let result = findUnblockCombo(i, blockingCar, JSON.parse(JSON.stringify(moveableCars)))
            let output = ""
            if(result){
                // Umkehren des Arrays, damit die einzeln benötigten Verschiebung der Autos in der Reihenfolge sind, in welcher sie durchgeführt werden müssten
                for(let steps of result.reverse()){
                    output += `${steps[0][0]} ${steps[1]} ${steps[2] == "L" ? "Links" : "Rechts"}, `
                }

                console.log(`${alphabet[alphabet.indexOf(stationaryCars[0]) + i]}: ${output.slice(0, -2)}`)
            } else {
                console.log(`${alphabet[alphabet.indexOf(stationaryCars[0]) + i]}: Keine Ausfahrt möglich`)
            }

        } else {
            console.log(`${alphabet[alphabet.indexOf(stationaryCars[0]) + i]}: Kein Bewegen nötig`)
        }
    }
}

// Versucht aus beiden Möglichkeiten das blockierende Auto zu bewegen (links oder rechts), die beste Kombination zu finden
function findUnblockCombo(carPosition, blockingCar, moveableCars){
    let parkingLotEnd = alphabet.indexOf(stationaryCars[1]) - alphabet.indexOf(stationaryCars[0])

    // Array, in denen die Kombinationen gespeichert werden und aus welchen am Ende die besser gewählt wird
    let moveCombinations = [[],[]]

    // Anzahl der Felder, die man nach links bzw. rechts gehen müsste, damit das Auto frei wird.
    let movesNeeded = [1 + blockingCar[1] - carPosition + 1, 1 + carPosition - blockingCar[1]]

    // Bestimmung der Kombination bei einer Bewegung nach rechts
    if(blockingCar[1] + movesNeeded[1] <= parkingLotEnd){
        // Äußerstes bewegliche Auto
        let furthestNearbyCar = blockingCar
        // Anzahl der Menge der Felder, die die Autos sich wegbewegen müssen
        // (nur wichtig für die Zusammenstellung der Verschiebungen)
        let moveCount = []

        // Felder, die die einzelnen Autos sich bewegen müssen
        let moveCombo = []

        // Verbleibenden Felder, die die weiteren Autos der Kette sich bewegen müssen
        let movesRemaining = movesNeeded[1];

        while(movesRemaining != 0){
            // Bestimmen eines weiteren störenden Autos; Müsste weiter außen sein als das jetzige Auto und auf den zu überquerenden Feldern stehen
            let anotherNearbyCar = moveableCars.find(car => car[1] <= furthestNearbyCar[1] + movesRemaining + 1 && car[1] > furthestNearbyCar[1])
            if(anotherNearbyCar){
                if(anotherNearbyCar[1] - furthestNearbyCar[1] - 1 == 2 && movesRemaining == 2){
                    moveCount.push(movesRemaining)
                    movesRemaining = 1
                } else {
                    moveCount.push(movesRemaining)
                }

                furthestNearbyCar = anotherNearbyCar
            } else if(furthestNearbyCar[1] + movesRemaining + 1 > parkingLotEnd){
```

```
            // Falls bei keinem störenden Auto der Raum des Parkplatz überschritten würde, kann das blockierende Auto in diese Richtung nicht weggebewegt werden
            furthestNearbyCar = 0;
            break;
        } else {
            moveCount.push(movesRemaining)
            break;
        }
    }

    if(furthestNearbyCar != 0){
        for(let i = blockingCar[1]; i <= furthestNearbyCar[1]; i++){
            let car = moveableCars.find(car => car[1] == i)
            if(car){
                moveCombo.push([car, moveCount.shift(), "R"])
            }
        }

        moveCombinations[1] = moveCombo
    }
}

 // Bestimmung der Kombination bei einer Bewegung nach links
if(blockingCar[1] - movesNeeded[0] >= 0){
    // Äußerstes bewegliche Auto
    let furthestNearbyCar = blockingCar

    // Anzahl der Menge der Felder, die die Autos sich wegbewegen müssen
    // (nur wichtig für die Zusammenstellung der Verschiebungen)
    let moveCount = []

    // Felder, die die einzelnen Autos sich bewegen müssen
    let moveCombo = []

    // Verbleibenden Felder, die die weiteren Autos der Kette sich bewegen müssen
    let movesRemaining = movesNeeded[0];

    while(movesRemaining != 0){
        // Bestimmen eines weiteren störenden Autos; Müsste weiter außen sein als das jetzige Auto und auf den zu überquerenden Feldern stehen
        let anotherNearbyCar = moveableCars.find(car => car[1] == furthestNearbyCar[1] - movesRemaining || car[1] + 1 == furthestNearbyCar[1] - movesRemaining && car[1] < furthestNearbyCar[1])
        if(anotherNearbyCar){
            if(furthestNearbyCar[1] - anotherNearbyCar[1] - 1 == 2 && movesRemaining == 2){
                moveCount.push(movesRemaining)
                movesRemaining = 1
            } else {
                moveCount.push(movesRemaining)
            }

            furthestNearbyCar = anotherNearbyCar
        } else if(furthestNearbyCar[1] - movesRemaining < 0){
            // Falls bei keinem störenden Auto der Raum des Parkplatz überschritten würde, kann das blockierende Auto in diese Richtung nicht weggebewegt werden
            furthestNearbyCar = 0;
            break;
        } else {
            moveCount.push(movesRemaining)
            break;
        }
    }

    if(furthestNearbyCar != 0){
        for(let i =  blockingCar[1]; i >= furthestNearbyCar[1]; i--){
            let car = moveableCars.find(car => car[1] == i)
            if(car){
                moveCombo.push([car,moveCount.shift(), "L"])
            }
        }

        moveCombinations[0] = moveCombo
```

```
        }
    }

    // Einzelnen Summen der Schritte & Summe der Autos die bewegt werden müssen.
    if(!moveCombinations.some(comb => comb.length == 0)){
        let scores = []
        for(let comb of moveCombinations){
            let stepSum = 0
            let carsToBeMoved = 0
            for(let step of comb){
                carsToBeMoved++
                stepSum += step[1]
            }
            scores.push([stepSum, carsToBeMoved])
        }

        if(scores[0][1] == scores[1][1]){
            return scores[0][0] < scores[1][0] ? moveCombinations[0]: moveCombinations[1]
        } else {
            return scores[0][1] < scores[1][1] ? moveCombinations[0]: moveCombinations[1]
        }
    } else {
        return moveCombinations.find(comb => comb.length != 0)
    }
}
```