Aufgabe 3: Zauberschule

Team-ID: 00265

Teilnahme-ID: 69306

Team-Name: Die Debugger

Bearbeiter dieser Aufgabe:
Abulfasl Ahmadi

19. November 2023

Inhaltsverzeichnis

1 Lösungsidee und Umsetzung

Vorweg sei gesagt, dass die Umsetzung in Java Script ES6 erfolgte. Die Lösungsidee ist zusätzich in Pseudocode beschrieben. Der Quellcode ist im Anhang zu finden.

Am anfang erstellte ich ein 3 dimensionalen Raum mit 2 Etagen und der länge + breite von der txt datei. Die einzelnen Felder waren mit den Indexen von verschachtelten Arrays ansteuerbar. Nach dem Laden der txt Datei wurden die Mauern mit einem Hashtag und freie Felder mit einem Punkt gekennzeichnet.

Dannach habe ich mich von Dijkstras Algorithmus inspirieren lassen, um den schnellsten weg von A nach B zu finden. Denn ein Wechsel der Etage dauert 3 Sekunden und oben/unten/links/rechts nur eine Sekunde. A habe ich also auf 0 gesetzt und B auf Infinity. Der Algorithmus hat dann für jeden seinen nachbarn (links, rechts, oben, unten, hoch, runter) geschaut, ob

1. dieses Feld existiert (z.B. wenn man in der oberen Ebene ist, gibt es zwar eine untere Ebene aber keine weitere obere Ebene. Der Algorithmus soll also nicht auf einen nicht definierten Wert zugreifen)
2. keine Mauern sind, da er nicht mehr zaubern kann ;) und
3. Ob der nachbar einen niedrigeren Wert hat, als der aktuelle Wert + 1

Wenn alle 3 Konditionen gegeben sind, wird der Wert des Nachbarns auf den aktuellen Wert + 1 gesetzt. Bei einem wechel der Ebene wurde der Aktuelle Wert + 3 gesetzt. Die gesammte prozedur wird für jeden neu erstellten nachbar berechnet, bis B erreicht wurde.

Um anschließed den schnellsten weg zu finden, wird von B aus die nachbarn überprüft, welcher nachbar den niedrigsten Wert hat. Die Richtung wird mit < > v ^ ! angegeben, um den Weg nachzuvollziehen.

Am ende wird auch nochmal die Zeit des schnellsten Weges ausgegeben.

2 Psuedocode:

Note: Der Pseudocode ist nicht vollständig, da er nur die Lösungsidee beschreibt.

Mauern und frie Felder in 3 dimensionalen Raum eintragen (verschachtelte Arrays)

dijkstras algorithmus:

A = 0

B = Infinity

für jeden nachbarn vom Aktuellen Feld:

wenn nachbar existiert:

wenn nachbar keine Mauer:

wenn nachbar < Wert vom aktuellen Feld + 1:

nachbar = Wert vom aktuellen Feld + 1

Ebene wechsel:

nachbar = Wert vom aktuellen Feld + 3

kürzester weg:

für jeden nachbarn von B:

wenn nachbar existiert:

wenn nachbar < B:

B = nachbar

richtung wird angegeben mit < > v ^ !

3 Beispiele

3.1 Beispiel 0

```
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
###.#.###.#.#
#...#.....#.#
```

```
#.#########.#
#.....#.....#
#####.#.###.#
#....!#B..#.#
#.#########.#
#...........#
#############

#############
#.......#...#
#...#.#.#...#
#...#.#.....#
#.###.#.#.###
#.....#.#...#
#####.###...#
#.....#.....#
#.#########.#
#...#>>!....#
#.#.#.#.###.#
#.#...#...#.#
#############
```

Ron braucht 8s, um von A nach B zu kommen

3.2 Beispiel 1

```
######################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
#.#...#.#.#^<<#.....#
#.#####.#.#########.#
#...................#
######################

######################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#...........#.......#
######################
```

Ron braucht 4s, um von A nach B zu kommen

3.3 Beispiel 2

```
###########################################
#...#.....#..........#.#..........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#>>!#v#.....#.#.#...#...#
###.###.#.#.#############v#.#.###.#.###.#.###
```

```
#.#.#...#.#.............>>B#.#...#.#...#.#.#
#.#.#.###.###########.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
###############################################

###############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#
###.#.###.#.#.#############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
###############################################

Ron braucht 14s, um von A nach B zu kommen
```

3.4 Beispiel 3

```
################################
#...#.....#..........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.###########.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.#######.###.#.#####
#...#.#...#...#.#...#...#.#...#
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.#############.#.#######
#.#.#.#.............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
###.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#>>B#.#...#.#
#.#########.#######^#.###.###.#
```

```
#..v#>>>>v#.#>>>>>>^#...#.#.#.#
#.#v#^###v#.#^########.#.#.#.#
#.#>>^..#>>>>^#..........#...#
###############################

###############################
#.......#.....#...#...#.......#
#.###.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.....#.#.#.....#
#######.#.#.#.#####.#.#####.#
#.....#.#.#.#.#...#...#.....#
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#.#.#.#.#.#
#.#.#######.#.#.#####.#.#.#.#.#
#.#.......#.#...#.....#.#...#.#
#.###.#####.#####.#####.#####.#
#...#.............#..........#
###############################

Ron braucht 28s, um von A nach B zu kommen
```

3.5 Beispiel 4

Abbildung 1: Erste Ebene

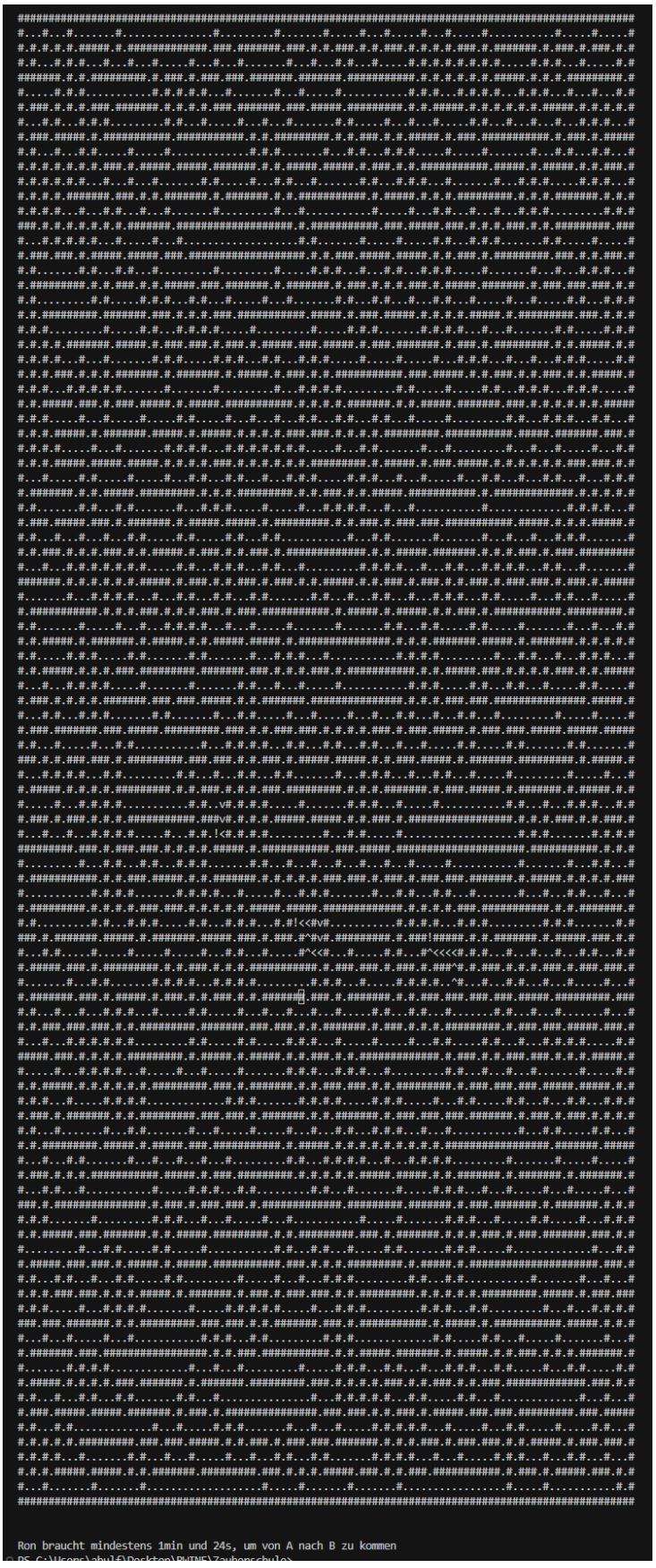

Abbildung 2: Zweite Ebene

3.6 Beispiel 5

Da es sich um ein sehr großes Beispiel handelt, ist es in dem Ordner als txt-Datei zu finden.

4 Quellcode

```
const fs = require("fs")
let data = fs.readFileSync("./tests/zauberschule5.txt", {encoding: "utf-8"})

// speichern der Felder in verschatelte Arrays
let floors = [[] /* i=0 ist die oberer Ebene */, [] /* i=1 ist die untere Ebene
*/] /* Äußere Klammer beinhaltet alle ebenen */
let temp = []
temp = data.replace(/\r/g, "").split("\n") // jede zeile wird in temp gespeichert

for(let i=1; i<temp.indexOf(""); i++) {
       floors[0].push(temp[i].split("")) // untere ebene wird in floors[0]
gespeichert und in jede Spalte einer Zeile bekommt ein index
}

for(let i=temp.indexOf("")+1; i<temp.length; i++) {
   floors[1].push(temp[i].split("")) // obere ebene wird in floors[1] gespeichert
und in jede Spalte einer Zeile bekommt ein index
}

// Positionen vom sStart (A) und Ziel (B) finden und in indexofA bzw indexofB
speichern
let indexOfA = []
let indexOfB = []

for(i in floors) {
    for(j of floors[i]) {
        for(k of j) {
            if(k == "A") {
                indexOfA.push(parseInt(i), floors[i].indexOf(j), j.indexOf(k))
            }
            if(k == "B") {
                indexOfB.push(parseInt(i), floors[i].indexOf(j), j.indexOf(k))
            }
        }
    }
}

// Funktion zum finden des schnellsten Weges mit Dijkstra's Algorithmus
// s = start, g = goal(ziel)
function dijkstra(s, g) {
    let ebene = s[0]
    let zeile = s[1]
    let spalte = s[2]
    floors[ebene][zeile][spalte] = 0 // Anfangsknoten (A) mit 0 markieren
    floors[g[0]][g[1]][g[2]] = Infinity // Zielknoten (B) mit ∞ markieren

    let toSearch = [[ebene, zeile, spalte, 0]]

    function conditions() {
```

```
        let length = toSearch.length
        for(i of toSearch) {
            if((floors[i[0]][i[1]][i[2]-1] == ".") || (floors[i[0]][i[1]][i[2]
-1] > (i[3]+1))) { // nach links
                floors[i[0]][i[1]][i[2]-1] = i[3]+1 // entfernung zum Knoten
schreiben
              toSearch.push([i[0], i[1], i[2]-1, i[3]+1]) // Knoten in toSearch
schreiben um nach weiteren Knoten zu suchen
            }
           if((floors[i[0]][i[1]][i[2]+1] == ".") || (floors[i[0]][i[1]][i[2]+1]
> (i[3]+1))) { // nach rechts
                floors[i[0]][i[1]][i[2]+1] = i[3]+1 // entfernung zum Knoten
schreiben
              toSearch.push([i[0], i[1], i[2]+1, i[3]+1]) // Knoten in toSearch
schreiben um nach weiteren Knoten zu suchen
            }
             if((floors[i[0]][i[1]-1][i[2]] == ".") || (floors[i[0]][i[1]-1]
[i[2]] > (i[3]+1))) { // nach oben
                floors[i[0]][i[1]-1][i[2]] = i[3]+1 // entfernung zum Knoten
schreiben
              toSearch.push([i[0], i[1]-1, i[2], i[3]+1]) // Knoten in toSearch
schreiben um nach weiteren Knoten zu suchen
            }
             if((floors[i[0]][i[1]+1][i[2]] == ".") || (floors[i[0]][i[1]+1]
[i[2]] > (i[3]+1))) { // nach unten
                floors[i[0]][i[1]+1][i[2]] = i[3]+1 // entfernung zum Knoten
schreiben
              toSearch.push([i[0], i[1]+1, i[2], i[3]+1]) // Knoten in toSearch
schreiben um nach weiteren Knoten zu suchen
            }
            if(floors[i[0]+1]) { // ebene nach unten
              if((floors[i[0]+1][i[1]][i[2]] == ".") || (floors[i[0]+1][i[1]]
[i[2]] > (i[3]+3))) {
                floors[i[0]+1][i[1]][i[2]] = i[3]+3 // entfernung zum Knoten
schreiben
                  toSearch.push([i[0]+1, i[1], i[2], i[3]+3]) // Knoten in
toSearch schreiben um nach weiteren Knoten zu suchen
               }
            }
            if(floors[i[0]-1]) { // ebene nach oben
              if((floors[i[0]-1][i[1]][i[2]] == ".") || (floors[i[0]-1][i[1]]
[i[2]] > (i[3]+3))) {
                floors[i[0]-1][i[1]][i[2]] = i[3]+3 // entfernung zum Knoten
schreiben
                  toSearch.push([i[0]-1, i[1], i[2], i[3]+3]) // Knoten in
toSearch schreiben um nach weiteren Knoten zu suchen
               }
            }
        }
         toSearch.splice(0, length) // bereits durchsuchte Knoten aus toSearch
löschen

          if(toSearch.length > 0) { // wenn toSearch nicht leer ist, es also
weitere zu untersuchende Pukte gibt, werden die fknt rekursiv aufgerufen um die
nächsten Punkte zu untersuchen
            conditions()
        }
    }
```

```
    conditions();
}
dijkstra(indexOfA, indexOfB);

// Funktion, die den (von der Dijkstra Funktion gegebenen) schnellsten Weg visuell darstellt mit <; >; ^; v; !
let time = 0
function showPath(s) {
    let ebene = s[0]
    let zeile = s[1]
    let spalte = s[2]
      time = floors[ebene][zeile][spalte] // Die von der djikstra Funktion berechnete Zeit wird gespeichtert
    floors[ebene][zeile][spalte] = "B" // Zeit wird wieder mit B überschrieben
    let temp = time

    function reverseSearching() {
        if(floors[ebene][zeile][spalte+1] === (temp-1)) { // nach rechts
            temp = floors[ebene][zeile][spalte+1]
            floors[ebene][zeile][spalte+1] = "<"
            spalte++
        }

        if(floors[ebene][zeile][spalte-1] === (temp-1)) { // nach links
            temp = floors[ebene][zeile][spalte-1]
            floors[ebene][zeile][spalte-1] = ">"
            spalte--
        }
        if(floors[ebene][zeile+1][spalte] === (temp-1)) { // nach unten
            temp = floors[ebene][zeile+1][spalte]
            floors[ebene][zeile+1][spalte] = "^"
            zeile++
        }
        if(floors[ebene][zeile-1][spalte] === (temp-1)) { // nach oben
            temp = floors[ebene][zeile-1][spalte]
            floors[ebene][zeile-1][spalte] = "v"
            zeile--
        }
        if(floors[ebene+1]) {
            if(floors[ebene+1][zeile][spalte] === (temp-3)) { // nach oben
                temp = floors[ebene+1][zeile][spalte]
                floors[ebene+1][zeile][spalte] = "!"
                ebene++
            }
        }
        if(floors[ebene-1]) {
            if(floors[ebene-1][zeile][spalte] === (temp-3)) { // nach oben
                temp = floors[ebene-1][zeile][spalte]
                floors[ebene-1][zeile][spalte] = "!"
                ebene--
            }
        }
        if(temp > 0) {
            reverseSearching()
        }
    }
    reverseSearching()
```

```
    // Berechnete distanzen wieder mit "." überschreiben
    for(i of floors) {
        for(j of i) {
            for(k of i) {
                for(l of k) {
                    if(typeof(l) == "number") {
                        k[k.indexOf(l)] = "."
                    }
                }
            }
        }
    }
}

showPath(indexOfB);

// Ausgabe der Felder
for(i of floors) {
    for(j of i) {
        let tempText = "";
        for(k of j) {
            tempText += k;
        }
        console.log(tempText)
    }
    console.log("")
}

// Ausgabe der Zeit
if(time > 60) {
    console.log(`Ron braucht mindestens ${Math.floor(time/60)}min und ${time%60}
s, um von A nach B zu kommen`)
} else {
    console.log(`Ron braucht ${time}s, um von A nach B zu kommen`)
}
```