# Aufgabe 3: Zauberschule

Team-ID: 00099

Team-Name: Cryptellum

Bearbeiter/-innen dieser Aufgabe:
Selahaddin Inan

3. November 2023

## Inhaltsverzeichnis



## 1 Lösungsidee

Diese Aufgabe können wir mithilfe von einer erweiterten Breitensuche, nämlich dem Djikstra-Algorithmus[1] lösen, was uns dabei helfen wird, den kürzesten Weg von einem gewählten Punkt A nach allen anderen Punkten in unserem 3-dimensionalen Array (Bugwarts) zu finden. Hierbei stellen wir uns jeden einzelnen Punkt im Array als ein Knote/Punkt in einem Graphen (auch Vertex genannt) vor. Dabei suchen wir uns dann den kürzesten Weg für die gegebene Punkt B.

## 2 Umsetzung

### 2.1 Nachbarn im Array finden

Nachdem wir die Eingabe gelesen, und die Koordinaten der Start-, bzw. Endpunkte gefunden haben, definieren wir eine Funktion namens Nachbarn_Finden(array, knoten), was uns später bei der Algorithmus-Umsetzung helfen wird. Diese Funktion findet alle „gültigen“ Nachbarn vom gegebenen Knoten. Als gültig gelten alle Nachbarn, die kein „#“ sind, oder außerhalb des Arrays liegen (sonst würden wir ja einen IndexError bekommen). Falls wir einen gültigen Nachbar gefunden haben, fügen wir diese in eine Liste, und speichern gleichzeitig den Kosten (also 3 sek. für einen Ebenen Wechsel und 1sek. für ein Schritt in derselben Ebene) mit ein.

[1] https://de.wikipedia.org/wiki/Dijkstra-Algorithmus

### 2.2 Djikstra-Algorithmus Implementation

Zu Beginn erstellen wir drei Datenstrukturen: distanzen, vorgänger und queue. Im distanzen-Dictionary speichern wir die kumulativen Distanzen von einem Startknoten zu allen anderen Knoten im Graphen. Die Datenstruktur vorgänger hält den Vorgängerknoten jeden Knotens, was wir dann bei der Rekonstruktion des kürzesten Pfades brauchen. Die Datenstruktur queue ist eine Prioritätswarteschlange, die die Knoten und ihre (vorläufigen) Distanzwerte verwaltet. Der Dijkstra-Algorithmus wird in einer while-Schleife durchgeführt, die solange läuft, wie es noch unbesuchte Knoten in der Warteschlange gibt. In jedem Schleifendurchlauf wird der Knoten mit der kürzesten vorläufigen Distanz aus der Warteschlange entfernt. Für jeden Nachbarknoten des aktuellen Knotens wird die vorläufige Distanz berechnet und mit der bereits im distanzen-Dictionary gespeicherten Distanz verglichen. Wenn die neue Distanz kürzer ist, wird sie aktualisiert, und der Vorgängerknoten wird festgelegt. Sobald der Zielpunkt erreicht ist, wird der kürzeste Pfad durch Zurückverfolgen der Vorgängerknoten rekonstruiert und in "pfad"gespeichert. Die insgesamt benötigte Zeit für die Reise wird ebenfalls ermittelt und zurückgegeben.

## 3 Beispiele

```
zauberschule1.txt
######################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
#.#...#.#.#^<<#.....#
#.#####.#.#########.#
#...................#
######################
######################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#...........#.......#
######################
Gesamtkosten (in Sekunden): 4

zauberschule2.txt
################################################
#...#.....#...........#.#...........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#>>!#v#.....#.#.#...#...#
###.###.#.#.##############v#.#.###.#.###.#.###
#.#.#...#.#..............>>B#.#...#.#...#.#.#
#.#.#.###.###########.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
```

```
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
###############################################
###############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
###############################################
Gesamtkosten (in Sekunden): 14

zauberschule3.txt
#################################
#...#.....#...........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.###########.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.#######.###.#.#####
#...#.#...#...#.#...#...#.#...#
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.##############.#.#######
#.#.#.#...............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
###.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#>>B#.#...#.#
#.#########.#######^#.###.###.#
#..v#>>>>v#.#>>>>>>>^#...#.#.#.#
#.#v#^###v#.#^#########.#.#.#.#
#.#>>^..#>>>>^#..........#...#
#################################
#################################
#.......#.....#...#...#.......#
#.###.#.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.#.....#.#.#.....#
#######.#.#.#.#.#####.#.#####.#
#.....#.#.#.#.#.#...#...#.....#
```

```
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.#########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#.#.#.#.#.#
#.#.#######.#.#.#####.#.#.#.#.#
#.#.......#.#...#.....#.#...#.#
#.###.#####.#####.#####.#####.#
#...#.............#..........#
################################
Gesamtkosten (in Sekunden): 28

zauberschule4.txt
...
Gesamtkosten (in Sekunden): 84

zauberschule5.txt
...
Gesamtkosten (in Sekunden): 124

zauberschule6.txt (eigenes Beispiel)
#.###
#.#B#
#.#^#
#...#
#.#.#
#####
#.###
#.#!#
#>>^#
#^#.#
Gesamtkosten (in Sekunden): 8

zauberschule7.txt (eigenes Beispiel)
########
#v..#..#
#v#.#.#.
#v#.#..#
#v#.####
#!....##
##.###.#
########
########
```

```
#.#....#
#.#.##.#
#...#..#
##.#.#.#
#v..#..#
#v.#.#.#
#B######
Gesamtkosten (in Sekunden): 9
```

zauberschule8.txt (eigenes Beispiel)

```
############
#>>>>>>>>v#
#........v#
#........v#
#........v#
#........v#
#........v#
#........v#
#........!#
############
############
#.........#
#.........#
#.........#
#.........#
#.........#
#.........#
#.........#
#........B#
############
Gesamtkosten (in Sekunden): 18
```

zauberschule9.txt (Beispiel von der README Datei)

```
##############
#............#
#######B###.#
#....!#^..#.#
#.##########.#
#............#
##############
##############
#.....#.....#
#.#.#######.#
#...#>>!....#
#.#.#.#.###.#
#.#...#...#.#
##############
Gesamtkosten (in Sekunden): 9
```

# 4 Quellcode

```python
def nachbarn_finden(array, eckpunkt):
    nachbarn = []

    richtungen = [(1, 0, 0), (-1, 0, 0), (0, 1, 0),
                  (0, -1, 0), (0, 0, 1), (0, 0, -1)]

    # ueberprueft, ob die Nachbarn innerhalb des Arrays liegen und ob sie nicht "#"
    sind, wenn nicht, werden sie der Liste "nachbarn" hinzugefuegt
    for dx, dy, dz in richtungen:
        neu_x, neu_y, neu_z = eckpunkt[0] + dx, eckpunkt[1] + dy, eckpunkt[2] + dz

        if (
            0 <= neu_x < len(array) and
            0 <= neu_y < len(array[0]) and
            0 <= neu_z < len(array[0][0])
        ):
            if array[neu_x][neu_y][neu_z] != "#":
                if neu_x == eckpunkt[0]:
                    gewichtung = 1
                    nachbarn.append(((neu_x, neu_y, neu_z), gewichtung))

                elif neu_x != eckpunkt[0]:
                    gewichtung = 3
                    nachbarn.append(((neu_x, neu_y, neu_z), gewichtung))

    return nachbarn
```

Nachbarn eines Eckpunkts finden

```python
def dijkstra(array, start, end):
    distanzen = {}
    vorgaenger = {} # speichert den Vorgaenger eines Knotens
    queue = [(0, start)]

    distanzen[start] = 0

    while queue:
        derzeitige_distanz, eckpunkt = hq.heappop(queue)

        if eckpunkt == end:
            pfad = []
            while eckpunkt in vorgaenger:
                pfad.insert(0, eckpunkt)
                eckpunkt = vorgaenger[eckpunkt]
            pfad.insert(0, start)

            sekunden = distanzen[end]
            return pfad, sekunden
        if derzeitige_distanz > distanzen[eckpunkt]:
            continue

        for nachbar, gewichtung in nachbarn_finden(array, eckpunkt):
            distanz = derzeitige_distanz + gewichtung

            if nachbar not in distanzen or distanz < distanzen[nachbar]:
                distanzen[nachbar] = distanz
                vorgaenger[nachbar] = eckpunkt
                hq.heappush(queue, (distanz, nachbar))
```

Djikstra-Algorithmus Umsetzung