Aufgabe 3: Zauberschule

Teilnahme-ID: 69408

Bearbeiter dieser Aufgabe:
Tim Krome

31. Oktober 2023

Lösungsidee 1
Finden des kürzesten Wegs 1
Optimierung 2
Umsetzung 3
Einlesen der Eingabedatei 3
Definieren einer Klasse zum Speichern von Positionen 4
Durchführen der Breitensuche 5
Ausgabe des Wegs 5
Beispiele 5
Beispiel 0 6
Beispiel 1 6
Beispiel 2 7
Beispiel 3 8
Beispiel 4 9
Beispiel 5 10
Quellcode 10

Lösungsidee

Finden des kürzesten Wegs

Jedes Feld in der Zauberschule lässt sich über die drei Koordinaten x (Spalte, in der das Feld positioniert ist), y (Zeile, in der das Feld positioniert ist) und Stockwerk (oben / unten) lokalisieren. Jede Position kann somit als Tripel (Stockwerk, y, x) dargestellt werden. Zu jeder Position, die Teil eines Wegs ist, lässt sich auch ein Zeitstempel angeben, der beschreibt, wie lange man auf dem Weg zum Erreichen der Position brauchte.

Zum Finden des kürzesten Wegs von A nach B kann eine Breitensuche verwendet werden:

1. Eine Warteschlange und eine Menge für bereits besuchte Felder werden initialisiert, beide Mengen sind zunächst leer.
2. Die Suche beginnt bei Feld A. Position A wird mit dem Zeitstempel t = 0s versehen und zur Warteschlange hinzugefügt.
3. Die folgenden Schritte werden wiederholt, bis die Warteschlange leer ist oder man bei Feld B angelangt ist:
    a. Die Position mit dem niedrigsten Zeitstempel (die Position wird im Folgenden als *node* bezeichnet) wird aus der Warteschlange entnommen und aus der Warteschlange entfernt.
    b. Es wird überprüft, ob *node* gleich Feld B ist. Wenn ja, wird die Suche abgeschlossen, da man Feld B erreicht hat.

c. Es wird überprüft, ob *node* zu einem anderen Zeitpunkt bereits besucht wurde. Wenn *node* in der Menge der bereits besuchten Felder ist, werden d. und e. übersprungen und die Position *node* wird nicht weiter betrachtet (auf die Art wird verhindert, dass Felder doppelt besucht werden)[1]. Wenn *node* noch nicht in der Menge der bereits besuchten Felder ist, dann wird *node* zu ebendieser Menge hinzugefügt und es wird mit d. und e. fortgefahren.
d. Es wird überprüft, ob das Feld im Stockwerk über bzw. unter *node* frei ist. Wenn ja, dann wird es mit dem Zeitstempel t = [Zeitstempel von *node* + 3 Sekunden] versehen zur Warteschlange hinzugefügt. (Um das Stockwerk zu wechseln, braucht man nämlich 3 Sekunden)
e. Es wird überprüft, ob die Nachbarfelder von *node* frei sind (also keine Mauer enthalten), werden mit dem Zeitstempel t = [Zeitstempel von *node* + 1 Sekunde] versehen zur Warteschlange hinzugefügt. (Um auf dem gleichen Stockwerk von einem Feld zum nächsten zu gelangen, braucht man nämlich 1 Sekunde)

Da die Breitensuche in jeder Iteration die Position mit dem niedrigsten Zeitstempel aus der Warteschlange entnimmt, wird sie immer den kürzesten Weg finden.
Wenn die Warteschlange leer ist, bevor die Breitensuche Feld B erreicht hat, dann gibt es keinen Weg, der von Feld A zu Feld B führt. Da Felder nicht doppelt betrachtet werden und eine Zauberschule nur endlich viele Felder hat, terminiert die Breitensuche auf jeden Fall.

Auf die Implementierung einer Heuristik, die die Evaluierungsdauer der Breitensuche verbessern könnte, wird verzichtet. Eine Implementierung einer A*-Heuristik wurde ausprobiert, hat aber bei den meisten Tests die Evaluierungsdauer kaum gesenkt und in einigen Fällen sogar verschlechtert. Außerdem hat das Programm, in dem die Lösungsidee umgesetzt ist, auch ohne Heuristik bereits eine akzeptable Evaluierungsdauer von weniger als einer Sekunde bei allen Beispielen (siehe das entsprechende Kapitel).

Optimierung

Eine Zauberschule sieht eigentlich so aus wie auf dem Aufgabenblatt, d.h. sie ist ein Raster aus Feldern, zwischen denen sich Trennwände befinden können. In den Eingabedateien von der BWinf-Webseite sind die Zauberschulen in einer digitalisierten Textdarstellung dargestellt, in der die Wände eine Dicke von 1 haben. Hieraus folgt, dass die Verteilung von Feldern und Wändern in den Eingabedateien von der BWinf-Webseite immer dasselbe Muster aufweist. Siehe hierzu folgende Beispiel-Textdarstellung einer Zauberschulen-Etage:

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

[1] Auf die Art wird verhindert, dass Felder doppelt besucht werden: Es soll nämlich die kürzeste Route gefunden werden, zu Feld B zu gelangen. Dies bedeutet, dass für alle Felder auf dem Weg von A nach B die kürzeste Verbindung von A zum jeweiligen Feld genommen werden muss. Da eine Breitensuche verwendet wird, wird die kürzeste Verbindung von A zum jeweiligen Feld als erstes gefunden.

```
#############
```

Die in der obigen Textdarstellung gelb markierten Felder müssen in jeder Zauberschulen-Etage frei sein, da sie den Feldern entsprechen, die in der Bilddarstellung tatsächlich existieren. Vergleicht man die Textdarstellung mit der Bilddarstellung, dann können die Wände als bloße Trenneinheiten zwischen den oben gelb markierten Feldern verstanden werden. Zwischen zwei gelb markierten Feldern befindet sich in der Textdarstellung demnach entweder eine Wand, die dafür sorgt, dass die beiden Felder voneinander getrennt sind, oder ein „Durchgangsfeld“, das angibt, dass die beiden Felder nicht durch eine Wand voneinander getrennt sind.

Basierend auf dieser Erkenntnis kann die im Vorkapitel beschriebene Tiefensuche optimiert werden: Beim Suchen eines Wegs kann direkt zwischen zwei gelb markierten Feldern „gesprungen“ werden, sofern sich zwischen ihnen ein Durchgangsfeld bzw. keine Wand befindet. Es müssen somit nur die Felder berücksichtigt werden, die auch in der Bilddarstellung existieren. Wände zwischen solchen Feldern werden dabei als bloße Trenneinheiten verstanden, die dafür sorgen, dass von einem Feld nicht auf das andere gewechselt werden kann. Durchgangsfelder werden als bloße Informationsträger, die die Abwesenheit einer Wand angeben, verstanden. Diese Optimierung führt zu einer verbesserten Worst-Case-Szenario-Laufzeit, da die Breitensuche jetzt nur noch die Felder betrachten muss, die auch tatsächlich in der Bilddarstellung existieren. **Wichtig:** Durchgangsfelder werden bei der Berechnung der Routendauer trotzdem berücksichtigt, da die Berechnung der Routendauer auf der Textdarstellung der Zauberschule aufbauen soll.

Umsetzung

Die Lösungsidee wird in Python 3.10 implementiert.

Einlesen der Eingabedatei

Der Dateipfad der .txt-Eingabedatei wird vom Nutzer als erstes Kommandozeilenargument angegeben. Das Programm liest die Textdatei als String ein und splittet sie in eine Liste namens *input_lines* auf, die die Zeilen der Textdatei enthält. Die Seitenlängen der Stockwerke werden in den Variablen *y_size* und *x_size* gespeichert.

```
# Textdatei einlesen
# sys.argv[1] ist das erste Kommandozeilenargument
with open(sys.argv[1]) as f:
    input_lines = f.read().split("\n")

(...)

# Dimensionen der Stockwerke speichern
y_size, x_size = input_lines.pop(0).split(" ")
y_size, x_size = int(y_size), int(x_size)
```

Anschließend wird in *floor1* wird in Form von einer verschachteln Liste der Aufbau des oberen Stockwerks gespeichert, in *floor0* wird der Aufbau des unteren Stockwerks gespeichert. Dabei wird die Darstellung eines Felds aus der Eingabedatei beibehalten: Felder, auf denen eine Wand ist, werden als „#“ gespeichert, freie Felder werden als „.“ Gespeichert.
Parallel dazu wird nach den Positionen von Feld A und Feld B gesucht, die Koordinaten der beiden

Felder werden als Tripel (Stockwerk, y-Koordinate, x-Koordinate) in den Variablen namens *startfeld* (Position von A) und *endfeld* (Position von B) gespeichert.

```
floor1 = []
floor0 = []
for i in range(y_size):
    line = input_lines.pop(0)
    if "A" in line:
        startfeld = (1, i, line.index("A"))
    if "B" in line:
        endfeld = (1, i, line.index("B"))
    floor1.append(list(line))
input_lines.pop(0)
for i in range(y_size):
    line = input_lines.pop(0)
    if "A" in line:
        startfeld = (0, i, line.index("A"))
    if "B" in line:
        endfeld = (1, i, line.index("B"))
    floor0.append(list(line))
floors = [floor0, floor1]
```

Definieren einer Klasse zum Speichern von Positionen

Damit eine Position auf dem Weg mit Zeitstempel und mit einem Verweis auf die vorherige Position gespeichert werden kann, wird einer Klasse namens *Node* erstellt.

Die Klasse hat diese Attribute:
- *Node.position : tuple* – Speichert die Koordinaten der aktuellen Position als Tripel der Form (Stockwerk, y-Koordinate, x-Koordinate)
- *Node.timestamp : int* – Speichert, nach wie vielen Sekunden die aktuelle Position erreicht wurde
- *Node.previous_node : Node* – Speichert die vorherige Position, was später das Rekonstruieren des Wegs erleichert
- *Node.operation : str* – Speichert die Operation („<“,“>“,“^“,“v“ oder „!“, die verwendet wurde, um zur Position zu gelangen. was später das Rekonstruieren des Wegs erleichert.

Die Klasse hat eine *successors()* Funktion, die eine Liste mit allen Positionen (als *Node*-Objekte) zurückgibt, die von der aktuellen Position aus erreicht werden können. Hierbei wird darauf geachtet, dass die zurückgegebenen Positionen alle innerhalb der Zauberschule liegen (für den Fall, dass die Zauberschule nicht von einer Mauer umgeben ist, der in der Aufgabenstellung nicht ausgeschlossen wird).
Es wird außerdem eine *__lt__(obj2)* Funktion implementiert, sodass zwei *Node*-Objekte mit dem „<“-Vergleichsoperator verglichen werden kann[2]. Die Funktion ist so programmiert, dass das Objekt mit dem kleineren Zeitstempel dabei das kleinere Objekt ist. Dies ermöglicht es, mit *min(list<Node>)* auf das Node-Objekt mit dem kleinsten Zeitstempel zuzugreifen.

[2] Beim Vergleichen zweier Objekte mit den „<“-Vergleichsoperator oder beim Anwenden von *min()* greift Python auf die *__lt__*-Funktion der zu vergleichenden Objekte zu.

Durchführen der Breitensuche

Zunächst werden eine Warteschlange (als Liste namens *to_visit*) und eine Menge für bereits besuchte Felder (als Set namens *visited_fields*) initialisiert. Die Warteschlange enthält hierbei zunächst nur das Startfeld A (als *Node*-Objekt mit dem Zeitstempel t = 0s):

```
visited_fields = set()
to_visit = [Node(startfeld, None, 0, None)]
```

Danach wird in einer while-Schleife die Position mit dem niedrigsten Zeitstempel aus *to_visit* entnommen (solange *to_visit* Positionen noch enthält):

```
while len(to_visit) > 0:
    node = min(to_visit)
    to_visit.remove(node)

    ...
```

Es wird überprüft, ob die Position *node* bereits besucht wurde und danach, ob die Position das Endfeld ist (wenn ja, wird die Schleife abgebrochen und der Weg ausgegeben):

```
    if node.position in visited_fields:
        continue # Bereits besuchte Felder nicht erneut besuchen
    if node.position == endfeld:
        print_path(node)
        print(f"\nDer Weg dauert {node.timestamp} Sekunden")
        break
    visited_fields.add(node.position)
    to_visit = to_visit + node.successors()
else:
    # Wenn die Warteschlange leer ist, konnte kein Weg gefunden werden
    print("Konnte keinen Weg finden.")
```

Ausgabe des Wegs

Eine *print_path* Funktion wird definiert, die einen Weg basierend auf einem *Node*-Objekt rekonstruiert. Hierfür greift es über das *previous_node*-Attribut auf das *Node*-Objekt zu, das den vorherigen Wegpunkt beschreibt, und wiederholt dies so lange, bis der gesamte Weg rekonstruiert ist. Die Funktion gibt beide Stockwerke (zuerst das obere, dann das untere) mit eingezeichnetem Weg in der Konsole aus. Die Stockwerke werden im gleichen Format wie in der Eingabe ausgegebenen, der Pfad wird mit ‚<' ‚>' ‚^' ‚v' für die Richtungen und ‚!' für den Wechsel in das jeweilige Stockwerk eingezeichnet. Die Programmausgabe folgt also dem in der README.md vorgeschlagenen Ausgabeformat.
Das Programm gibt außerdem aus, wie viele Sekunden man für den Weg braucht. **Bei der Berechnung der benötigten Zeit werden auch „Durchgangsfelder" berücksichtigt**. Für die Zauberschule vom Aufgabenblatt gibt das Programm folglich eine Routendauer von 8 Sekunden aus.

Beispiele

Die in Beispiel 0 - 5 verwendeten Eingabedateien stammen von der BWinf-Webseite. Zu Beispiel *x* gehört stets die Eingabedatei zauberschule*x*.txt.

Beispiel 0

Eingabedatei:
zauberschule0.txt

Ausgabe:
```
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
###.#.###.#.#
#...#.....#.#
#.#########.#
#.....#.....#
#####.#.###.#
#....!#B..#.#
#.#########.#
#...........#
#############

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

Der Weg dauert 8 Sekunden
```

Evalierungsdauer:
0,185 Sekunden

Beispiel 1

Eingabedatei:
zauberschule1.txt

Ausgabe:
```
####################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
```

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

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

```
Der Weg dauert 4 Sekunden
```

Evalierungsdauer:
0,193 Sekunden

Beispiel 2

Eingabedatei:
zauberschule2.txt

Ausgabe:

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

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

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

```
Der Weg dauert 14 Sekunden
```

Evalierungsdauer:
0,178 Sekunden

Beispiel 3

Eingabedatei:
zauberschule3.txt

Ausgabe:

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

################################
```

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

```
Der Weg dauert 28 Sekunden
```

Evalierungsdauer:
0,17 Sekunden

Beispiel 4

Eingabedatei:
zauberschule4.txt

Anmerkung:
Die in der Dokumentation angegebene Ausgabe wurde gekürzt, da die Ausgabe mehrere Seiten lang ist; die vollständige Ausgabe ist in der Datei */A3_Zauberschule/beispiel4_ausgabe.txt* zu finden.

Ausgabe:

```
(...)

Der Weg dauert 84 Sekunden
```

Evalierungsdauer:
0,212 Sekunden

Beispiel 5

Eingabedatei:
wundertuete5.txt

Anmerkung:
Die in der Dokumentation angegebene Ausgabe wurde gekürzt, da die Ausgabe mehrere Seiten lang ist; die vollständige Ausgabe ist in der Datei */A3_Zauberschule/beispiel5_ausgabe.txt* zu finden.

Ausgabe:
```
(...)

Der Weg dauert 124 Sekunden
```

Evalierungsdauer:
0,214 Sekunden

Quellcode

```
import sys

# Textdatei einlesen
# sys.argv[1] ist das erste Kommandozeilenargument
with open(sys.argv[1]) as f:
    input_lines = f.read().split("\n")

class Node:
    """
    Dient zum Speichern einer Position auf dem Weg
    """
    def __init__(self, position : tuple, previous_node, timestamp : int, operation :
str):
        self.position = position # Tripel, das die Position angibt und wie folgt
aufgebaut ist: (Stockwerk, y-Koordinate, x-Koordinate)
        self.previous_node = previous_node # Die vorherige Position, von der man zur
aktuellen Position gelangt ist
        self.timestamp = timestamp # Zeit [s], die man gebraucht hat, um zur aktuellen
Position zu gelangen
        self.operation = operation # Operation, die angewendet wurde, um zur aktuellen
Position zu gelangen

    def __lt__(self, obj2):
        return self.timestamp < obj2.timestamp

    def successors(self):
        """
        Returns:
            list<Node>: Liste mit allen Positionen, die von der aktuellen Position aus
erreicht werden könenn
        """
        floor, y, x = self.position
        successors = []

        if floors[1 if floor == 0 else 0][y][x] != "#":
            new_position = (1 if floor == 0 else 0, y, x)
            if new_position not in visited_fields:
```

```
                successors.append(
                    Node(new_position, self, self.timestamp + 3, "!")
                )

        if floors[floor][y][x-1] != "#":
            new_position = (floor, y, x-2)
            if x-2 > 0 and new_position not in visited_fields:
                successors.append(Node(new_position, self, self.timestamp+2, "<"))
# Timestamp wird um 2 erhöht, da beim Ändern der Position ein Durchgangsfeld
"übersprungen" wird
        if floors[floor][y][x+1] != "#":
            new_position = (floor, y, x+2)
            if x+2 < x_size and new_position not in visited_fields:
                successors.append(Node(new_position, self, self.timestamp+2, ">"))
        if floors[floor][y-1][x] != "#":
            new_position = (floor, y-2, x)
            if y - 2 > 0 and new_position not in visited_fields:
                successors.append(Node(new_position, self, self.timestamp+2, "^"))
        if floors[floor][y+1][x] != "#":
            new_position = (floor, y+2, x)
            if y + 2 < y_size and new_position not in visited_fields:
                successors.append(Node(new_position, self, self.timestamp+2, "v"))

        return successors

def print_path(node):
    """
    Rekonstruiert den zurückgelegten Weg (von der Zielposition aus) und gibt ihn aus
    Args:
        node (Node): Die Zielposition
    """
    while node.previous_node is not None:
        previous_node = node.previous_node

floors[previous_node.position[0]][previous_node.position[1]][previous_node.position[2]]
= node.operation
        if node.operation == "<":

floors[previous_node.position[0]][previous_node.position[1]][previous_node.position[2]-
1] = "<"
        if node.operation == ">":

floors[previous_node.position[0]][previous_node.position[1]][previous_node.position[2]+
1] = ">"
        if node.operation == "^":
            floors[previous_node.position[0]][previous_node.position[1]-
1][previous_node.position[2]] = "^"
        if node.operation == "v":

floors[previous_node.position[0]][previous_node.position[1]+1][previous_node.position[2
]] = "v"
        node = previous_node
    for line in floors[1]:
        print("".join(line))
    print("")
    for line in floors[0]:
        print("".join(line))
```

```
# Dimensionen der Stockwerke speichern
y_size, x_size = input_lines.pop(0).split(" ")
y_size, x_size = int(y_size), int(x_size)

# Aufbau vom oberen und unteren Stockwerk in Liste floors speichern und dabei das
Startfeld und das Endfeld finden
floor1 = []
floor0 = []
for i in range(y_size):
    line = input_lines.pop(0)
    if "A" in line:
        startfeld = (1, i, line.index("A"))
    if "B" in line:
        endfeld = (1, i, line.index("B"))
    floor1.append(list(line))
input_lines.pop(0)
for i in range(y_size):
    line = input_lines.pop(0)
    if "A" in line:
        startfeld = (0, i, line.index("A"))
    if "B" in line:
        endfeld = (1, i, line.index("B"))
    floor0.append(list(line))
floors = [floor0, floor1]

# Set initialisieren, in dem alle besuchten Felder gespeichert werden sollen
visited_fields = set()

to_visit = [Node(startfeld, None, 0, None)]
while len(to_visit) > 0:
    node = min(to_visit)
    to_visit.remove(node)
    if node.position in visited_fields:
        continue # Bereits besuchte Felder nicht erneut besuchen
    if node.position == endfeld:
        print_path(node)
        print(f"\nDer Weg dauert {node.timestamp} Sekunden")
        break
    visited_fields.add(node.position)
    to_visit = to_visit + node.successors()
else:
    # Wenn die Warteschlange leer ist, konnte kein Weg gefunden werden
    print("Konnte keinen Weg finden.")
```