Aufgabe 3: Zauberschule

Team-ID: 00128

Team-Name: E29C8CF09F8E89

Bearbeiter/-innen dieser Aufgabe:
Leonhard Masche

1. September 2023

Inhaltsverzeichnis

1 Lösungsidee

Zur Lösung dieses Shortest-Path-Problems gibt es einige bekannte Algorithmen. Die möglichen Wege in der Zauberschule können als gewichteter Graph dargestellt werden, wobei Bewegungen in die vier Richtungen (links, oben, rechts, unten) ein Gewicht

von 1, und Stockwerkwechsel ein Gewicht von 3 haben.[1] Um nun einen kürzesten Pfad zu finden, wird Dijkstra's Algorithmus verwendet: Für jeden besuchten Knoten (Feld) wird dessen Vorgänger gespeichert, sodass letztendlich der Pfad selbst zurückverfolgt werden kann. Entsprechend Dijktra's Algorithmus ist dies der kürzest mögliche Pfad.[2]

2 Umsetzung

Das Programm (`program.py`) ist in Python umgesetzt und mit einer Umgebung ab der Version 3.8 ausführbar. Zum Umgang mit Matrizen wird die externe Bibliothek `numpy` verwendet.

Beim Ausführen der Datei wird zuerst nach der Zahl des Beispiels gefragt. Dieses wird nun aus der Datei `input/zauberschule{n}.txt` geladen und bearbeitet. Das Ergebnis wird, zusammen mit einigen Werten, ausgegeben. Das resultierende Zauberschule-Gitter wird zusätzlich in eine Datei geschrieben.

3 Beispiele

Hier wird das Programm auf die 6 Beispiele von der Website angewendet. Zusätzlich wird ein eigenes Beispiel (`zauberschule6.txt`) bearbeitet, welches eine unlösbare Aufgabe darstellt:

- **zauberschule0.txt**

```
Oben:          Unten:
############# #############
#.....#.....# #.......#...#
#.###.#.###.# #...#.#.#...#
#...#.#...#.# #...#.#.....#
###.#.###.#.# #.###.#.#.###
#...#.....#.# #.....#.#...#
#.#########.# #####.###...#
```

[1] Das Programm wurde entsprechend der originalen Aufgabenstellung vom 1. Sept. geschrieben. Das bedeutet, Distanzen werden wie im Aufgabenblatt dargestellt berechnet.

[2] Vgl. E. W. Dijkstra. „A Note on Two Problems in Connexion with Graphs". In: *Numerical Mathematics* 1.1 (1. Dez. 1959), S. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390.

```
#.....#.....# #.....#.....#
#####.#.###.# #.#########.#
#....A#B..#.# #...#!>!....#
#.#########.# #.#.#.#.###.#
#..........# #.#...#...#.#
############# #############

Ausgabe gespeichert in "output/zauberschule0.txt"

Weg mit Länge 7s in 71 Iterationen (0.61ms) gefunden.
```

- **zauberschule1.txt**

```
Oben:                  Unten:
###################### ######################
#...#.....#...#.....# #.......#.....#.....#
#.#.#.###.#.#.#.###.# #.###.#.#.###.#.###.#
#.#.#...#.#.#...#...# #.....#.#.#.....#.#.#
###.###.#.#.#####.### #######.#.#######.#.#
#.#.#...#.#B....#...# #.....#.#.....#...#.#
#.#.#.###.#^###.##### #.###.#.#.###.###.#.#
#.#...#.#.#^<A#.....# #.#.#...#.#...#...#.#
#.#####.#.#########.# #.#.#######.###.###.#
#...................# #..........#.......#
###################### ######################

Ausgabe gespeichert in "output/zauberschule1.txt"

Weg mit Länge 2s in 4 Iterationen (0.16ms) gefunden.
```

- **zauberschule2.txt**

```
Oben:
###############################################
#...#.....#...........#.#...........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#A>!#!#.....#.#.#...#...#
###.###.#.#.#############v#.#.###.#.###.#.###
```

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

Unten:
###############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#!>!#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
###############################################

Ausgabe gespeichert in "output/zauberschule2.txt"

Weg mit Länge 10s in 93 Iterationen (0.79ms) gefunden.
```

- **zauberschule3.txt**

```
Oben:                                Unten:
################################## ##################################
#...#.....#..........#.......# #.......#.....#...#...#.......#
#.#.#.###.#.###.#######.###.#.# #.###.#.#.#.#.#.###.#.#.#.#####
#.#.#...#.#.#.#.......#...#.# #.....#.#.#.#.....#.#.#.....#
###.###.#.#.#.#.#######.###.# #######.#.#.#.#####.#.#####.#
#.#.#...#.#...#.....#.....#.#.# #.....#.#.#.#.#...#...#.....#
#.#.#.###.############.#####.#.# #.###.#.###.#.###.#.#.###.#####
#.#...#.#.#.........#...#.....# #...#.#.#...#.....#.#.#.#.....#
#.#####.#.#.#######.###.#.##### #.#.###.#.#########.#.#.#####.#
#...#.#...#...#.#...#...#.#...# #.#.....#.#.......#.#.....#...#
#.#.#.#####.#.#.###.###.#.#.# #.#######.#######.#.#.#####.#.#
#.#.#.#.......#.#.#...#.#...#.# #...#...#.....#...#.#.#...#.#.#
#.#.#.#########.#.#.###.#####.# ###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.......#.#.....#.#.#.....#.# #.#.#.#...#...#.....#.#.#.#.#.#
#.#######.#.#.#####.#.#.###.#.# #.#.#.#####.###.#######.#.#.#.#
#.#.....#...#.#.#...#.#.#.#...# #.#.#.....#.#.....#.....#.#.#.#
#.#.###.#####.#.#.#.#.#.#.##### #.#.#######.#####.###.###.#.#.#
#.#...#.....#.#.#.#.#...#.....# #.#.....#...#...#.....#.#...#.#
#.###.#####.#.#.#.#.###.#####.# #.#####.#.###.#.#######.#####.#
#...#.#...#...#...#...#.......# #.#...#.#.#...#.......#.#.....#
#.#.#.#.############.#.####### #.#.#.#.#.#.#########.#.#.#####
#.#.#.#.............#.#.....# #...#.#.#.#...#...#.#.#.#.....#
#.###.#.#######.#########.###.# #####.#.#.###.#.#.#.#.#.#####.#
#...#.#.#...#...#.........#...# #.....#.#.......#.#.#.#...#...#
###.#.###.#.#.###.#####.###.#.# #.###.#.#########.#.#.#.#.#.###
#...#.....#.#.....#>>B#.#...#.# #...#.#.....#.#.....#.#.#.#.#.#
#.#########.#######^#.###.###.# #.#.#######.#.#.#####.#.#.#.#.#
#..A#>>>>v#.#>>>>>>^#...#.#.#.# #.#.......#.#...#.....#.#...#.#
#.#v#^###v#.#^#########.#.#.#.# #.###.#####.#####.#####.#####.#
#.#>>^..#>>>>^#..........#...# #...#...........#.........#
################################## ##################################

Ausgabe gespeichert in "output/zauberschule3.txt"
```

```
Weg mit Länge 14s in 218 Iterationen (1.78ms) gefunden.
```

- **zauberschule4.txt**

```
Ausgabe gespeichert in "output/zauberschule4.txt"

Weg mit Länge 51s in 7043 Iterationen (45.75ms) gefunden.
```

- **zauberschule5.txt**

```
Ausgabe gespeichert in "output/zauberschule5.txt"

Weg mit Länge 75s in 10791 Iterationen (65.23ms) gefunden.
```

- **zauberschule6.txt**

```
ValueError: Es wurde kein Pfad gefunden!
```

4 Quellcode

program.py

```python
import dataclasses
import heapq
import os
import time
from typing import Iterable, List, Tuple

import numpy as np


@dataclasses.dataclass
class Char:
    _FIELD = "."
    _WALL = "#"
    _START = "A"
    _END = "B"
    BL = "<"
    BT = "^"
    BR = ">"
    LT = "^"
```

```
    LR = ">"
    LB = "v"
    TR = ">"
    TB = "v"
    TL = "<"
    RB = "v"
    RL = "<"
    RT = "^"
    VU = "!"
    VD = "!"


# (Veränderung der Feld-Koordinaten (plan, seen), zu überprüfende Wand-Koordinate (room), Wegkosten)
STEPS = (
    (lambda i, j, k: ((i, j, k - 1), (i, 2 * j + 1, 2 * k)), 1),  # Schritt nach links
    (lambda i, j, k: ((i, j, k + 1), (i, 2 * j + 1, 2 * k + 2)), 1),  # Schritt nach rechts
    (lambda i, j, k: ((i, j - 1, k), (i, 2 * j, 2 * k + 1)), 1),  # Schritt nach oben
    (lambda i, j, k: ((i, j + 1, k), (i, 2 * j + 2, 2 * k + 1)), 1),  # Schritt nach unten
    (lambda i, j, k: ((i - 1, j, k), None), 3),  # Stockwerk-wechsel nach unten
    (lambda i, j, k: ((i + 1, j, k), None), 3),  # Stockwerk-wechsel nach oben
)

# Helfer-Funktionen und Konstanten für die Ausgabe
STEP_CHARS = {(0, 0, -1): Char.RL, (0, 0, 1): Char.LR, (0, -1, 0): Char.BT, (0, 1, 0): Char.TB}
PRETTY_KERNELS = (
    lambda x: x[0, 1] == Char.TB and x[1, 2] == Char.LR and Char.TR,
    lambda x: x[0, 1] == Char.TB and x[2, 1] == Char.TB and Char.TB,
    lambda x: x[0, 1] == Char.TB and x[1, 0] == Char.RL and Char.TL,
    lambda x: x[1, 2] == Char.RL and x[0, 1] == Char.BT and Char.RT,
    lambda x: x[1, 2] == Char.RL and x[1, 0] == Char.RL and Char.RL,
    lambda x: x[1, 2] == Char.RL and x[2, 1] == Char.TB and Char.RB,
    lambda x: x[2, 1] == Char.BT and x[1, 0] == Char.RL and Char.BL,
    lambda x: x[2, 1] == Char.BT and x[0, 1] == Char.BT and Char.BT,
    lambda x: x[2, 1] == Char.BT and x[1, 2] == Char.LR and Char.BR,
    lambda x: x[1, 0] == Char.LR and x[0, 1] == Char.BT and Char.LT,
    lambda x: x[1, 0] == Char.LR and x[1, 2] == Char.LR and Char.LR,
    lambda x: x[1, 0] == Char.LR and x[2, 1] == Char.TB and Char.LB,
```

```
57  )
58
59
60  def load_zauberschule(
61      path: str,
62  ) -> Tuple[np.ndarray, Tuple[int, int, int], Tuple[int, int, int]]:
63      """
64      Öffne ein Beispiel und gebe den Raumplan, sowie die Start- und Endposition
    ↪   zurück.
65
66      Parameters
67      ----------
68      path : str
69          Der Dateipfad der Beispieldatei relativ zur `program.py`-Datei.
70
71      Returns
72      -------
73      Tuple[np.ndarray, Tuple[int, int, int], Tuple[int, int, int]]
74          Ein Tuple bestehend aus:
75            - Raumplan: 3-dimensionales numpy.ndarray, das die einzelnen Charaktere
    ↪   enthält.
76            - Der Startposition: Koordinate im Raumplan.
77            - Der Endposition: Koordinate im Raumplan.
78      """
79      with open(os.path.join(os.path.dirname(__file__), path), "r", encoding="utf8") as
        ↪   f:
80          # Dimensionen einer Ebene einlesen
81          n, m = map(int, f.readline().split())
82
83          # Variablen für Start- und Endkoordinaten und den Raumplan
84          start, end = None, None
85          room = np.empty((2, n, m), dtype=str)
86
87          def load(lv: int):
88              nonlocal start, end
89              for ni in range(n):
90                  for mi, c in enumerate(f.readline()[:m]):
91                      room[lv][ni][mi] = c  # Charakter in `room` speichern
92                      if c == Char._START:
93                          start = (lv, ni, mi)  # Startposition speichern
94                      if c == Char._END:
95                          end = (lv, ni, mi)  # Endposition speichern
```

```
96
97          load(1)  # Oberes Stockwerk laden
98          f.readline()  # eine Leerzeile "verbrauchen"
99          load(0)  # Unteres Stockwerk einlesen
100
101         assert None not in (
102             start,
103             end,
104         ), "Ungültiges Beispiel! (Punkt A oder B konnten nicht gefunden werden)"
105
106         return room, start, end
107
108
109 @dataclasses.dataclass(order=True)
110 class DijkstraItem:
111     """
112     Ein Eintrag in der Dijkstra-Queue.
113
114     Attributes
115     ----------
116     distance : int
117         Die Distanz von Punkt A zur Koordinate `coord`.
118     coord : Tuple[int, int, int]
119         Die Koordinate des Felds.
120     prev_coord : Tuple[int, int, int]
121         Die Koordinate des Feldes bei dessen Besichtigung dieses Item
122         in die queue hinzugefügt wurde. D. h. Diese Koordinate ist das nächste
123         Feld auf dem kürzesten Weg in Richtung Startpunkt.
124     """
125
126     distance: int
127     coord: Tuple[int, int, int] = dataclasses.field(compare=False)
128     prev_coord: Tuple[int, int, int] = dataclasses.field(compare=False)
129
130
131 def in_bounds(coord: Iterable[int], max_: Iterable[int], min_: Iterable[int] = None)
    ↪  -> bool:
132     """
133     Überprüfe, ob eine Koordinate innerhalb der Grenzen eines n-dimensionalen Arrays
    ↪  liegt,
134         die durch `min_` und `max_` angegeben werden.
135
```

```
136         Parameters
137         ----------
138         coord : Iterable[int]
139             Die Koordinate (eg. "(1, 2, 3)").
140         max_ : Iterable[int]
141             Die maximalen Werte für die einzelnen Dimensionen (exklusiv) (eg. "(4, 5,
    ↪ 5)").
142         min_ : Iterable[int], optional
143             Die minimalen Werte für die einzelnen Dimensionen (eg. "(0, 0, 0)").
144 
145         Returns
146         -------
147         bool
148             Ein Wahrheitswert, der besagt, ob `coord` innerhalb der gegebenen Grenzen
    ↪ liegt.
149         """
150         if min_ is None:
151             min_ = (0,) * len(max_)
152 
153         for i, n in enumerate(coord):
154             if n < min_[i] or n >= max_[i]:
155                 return False
156         return True
157 
158 
159 def to_plan_coord(x: Tuple[int, int, int]) -> Tuple[int, int, int]:
160     """
161     Rechne eine `room`-Kordinate in eine `plan`-Koordinate um.
162 
163     Parameters
164     ----------
165     x : Tuple[int, int, int]
166         Die `room`-Koordinate.
167 
168     Returns
169     -------
170     Tuple[int, int, int]
171         Die `plan`-Koordinate.
172     """
173     return x[0], int((x[1] - 1) / 2), int((x[2] - 1) / 2)
174 
175 
```

```python
def to_room_coord(x: Tuple[int, int, int]) -> Tuple[int, int, int]:
    """
    Rechne eine `plan`-Kordinate in eine `room`-Koordinate um.

    Parameters
    ----------
    x : Tuple[int, int, int]
        Die `plan`-Koordinate.

    Returns
    -------
    Tuple[int, int, int]
        Die `room`-Koordinate.
    """
    return x[0], x[1] * 2 + 1, x[2] * 2 + 1


def dijkstra(
    room: np.ndarray,
    plan: np.ndarray,
    n_p: int,
    m_p: int,
    start_p: Tuple[int, int, int],
    end_p: Tuple[int, int, int],
):
    # Die einzelnen Felder für den Dijkstra-Algorithmus
    # und das Zugehörige Array zum markieren besuchter Felder
    seen = np.zeros((2, n_p, m_p), bool)
    # heap als Schlange für den Algorithmus
    queue: List[DijkstraItem] = []
    # Erstes Item (Startpunkt) in die Schlange
    heapq.heappush(queue, DijkstraItem(0, start_p, (0, 0, 0)))

    # Dijkstra-Algorithmus
    n_steps = 0
    while len(queue) != 0:
        n_steps += 1
        item = heapq.heappop(queue)
        distance = item.distance
        coord = item.coord
        # Wenn dieses Feld schon besucht wurde, weiter iterieren
        if seen[coord]:
```

```
218             continue
219         # Aktuelles Feld als besucht markieren
220         seen[coord] = True
221         # Vorgänger-Feld festhalten
222         plan[coord] = item.prev_coord
223 
224         # Wenn der Endpunkt gefunden wurde, ist der Algorithmus fertig
225         if coord == end_p:
226             return n_steps, distance
227 
228         # Iterieren der verschiedenen Schritt-Möglichkeiten
229         for fn, cost in STEPS:
230             next_coord, to_check = fn(*coord)
231             next_distance = distance + cost
232             # wenn der Schritt außerhalb des Raums liegt oder eine Wand im Weg steht,
                ↪ überpringen
233             if not in_bounds(next_coord, np.shape(plan)) or (
234                 to_check and room[to_check] == Char._WALL
235             ):
236                 continue
237             # mögliches Feld in die Schlange hinzufügen
238             heapq.heappush(queue, DijkstraItem(next_distance, next_coord, coord))
239     else:
240         raise ValueError("Es wurde kein Pfad gefunden!")
241 
242 
243 # Zurückverfolgen des Pfades und einfügen der Wegmarkierungen
244 def trace(room, start_p, end_p, plan):
245     prev_p = end_p
246     while True:
247         current_p = tuple(plan[prev_p])
248         current_r = to_room_coord(current_p)
249 
250         step = tuple(np.subtract(prev_p, current_p, dtype="i8"))
251         if step[0] == 0:
252             room[tuple(np.add(current_r, step))] = STEP_CHARS[step]
253         else:
254             if step[0] == 1:
255                 if room[0, current_r[1], current_r[2]] == Char._FIELD:
256                     room[0, current_r[1], current_r[2]] = Char.VU
257                 if room[1, current_r[1], current_r[2]] == Char._FIELD:
258                     room[1, current_r[1], current_r[2]] = Char.VU
```

```
            elif step[0] == -1:
                if room[0, current_r[1], current_r[2]] == Char._FIELD:
                    room[0, current_r[1], current_r[2]] = Char.VD
                if room[1, current_r[1], current_r[2]] == Char._FIELD:
                    room[1, current_r[1], current_r[2]] = Char.VD

        if current_p == start_p:
            break
        prev_p = current_p


# einfügen von Markierungen zwischen den Feldern
def pretty(room):
    for lv in range(2):
        for n_ci in range(1, np.shape(room)[1], 2):
            for m_ci in range(1, np.shape(room)[2], 2):
                space = room[lv, (n_ci - 1) : (n_ci + 2), (m_ci - 1) : (m_ci + 2)]
                for kernel in PRETTY_KERNELS:
                    if char := kernel(space):
                        room[lv, n_ci, m_ci] = char
                        break


# Speichern des Pfades in eine Datei
def export(room: np.ndarray, path: str) -> str:
    p = os.path.join(os.path.dirname(__file__), path)
    with open(p, "w", encoding="utf8") as f:
        f.write(" ".join(map(str, np.shape(room)[1:])))
        f.write("\n")
        for li in room[::-1]:
            for ni in li:
                for mi in ni:
                    f.write(mi)
                f.write("\n")
            f.write("\n")
    return p


# Haupt-Loop
def main(room, start, end, n_bsp):
    t_s = time.time()  # Zeitmessung start

```

```
    # Werte für die kleinere Matrix umrechen
    n_p, m_p = map(lambda x: int((x - 1) / 2), np.shape(room)[1:])
    start_p, end_p = map(to_plan_coord, (start, end))
    plan = np.empty((2, n_p, m_p), "3uint16")
    n_steps, distance = dijkstra(room, plan, n_p, m_p, start_p, end_p)

    t_e = time.time()  # Zeitmessung ende

    # Verfolgen des berechneten Pfades und einfügen der Wegmarkierungen
    trace(room, start_p, end_p, plan)

    pretty(room)

    # Raum als ASCII-Art ausgeben
    if n_bsp <= 3:
        shape = np.shape(room)
        print(f'Oben:{" "*(shape[2]-4)}Unten:')
        for ni in range(shape[1]):
            for li in range(shape[0] - 1, -1, -1):
                for mi in range(shape[2]):
                    print(room[li, ni, mi], end="")
                print(" ", end="")
            print()

        print()

    # Exportieren nach Datei
    export(room, f"output/zauberschule{n_bsp}.txt")
    print(f'Ausgabe gespeichert in "output/zauberschule{n_bsp}.txt"')
    print()

    # Lösungswerte ausgeben
    print(
        f"Weg mit Länge {distance}s in {n_steps} Iterationen
        ↪ ({((t_e-t_s)*1000):.2f}ms) gefunden."
    )


while True:
    try:
        n_bsp = int(input("Bitte Nummer des Beispiels eingeben:\n> "))
        room, start, end = load_zauberschule(f"input/zauberschule{n_bsp}.txt")
```

```
342             print()
343             main(room, start, end, n_bsp)
344         except Exception as e:
345             print(f"{e.__class__.__name__}: {e}")
346         print()
```