Aufgabe 3: Zauberschule

`ggf(filename)`

Lösungsidee

Wir konnen den kürzesten Weg von A nach B durch eine parallele Suche bestimmen. Das heißt, wir finden jeweils die kürzesten Wege zu den zu A benachbarten Felder, dann die benachbarten Felder von den Felder (Wobei wir die besuchten Felder ignorieren, da der Weg nicht kuerzer sein kann, weil wir diese ansonsten vorher besucht haetten), und so weiter, bis wir irgendwann am B ankommen. Wir müssen nur aufpassen, dass die zuruckgelegten Kosten berücksichtigt werden, und wir dann entsprechend z.B. nach einer Teleportation nicht direkt in der anderen Etage weitersuchen, sondern entsprechende 3 Zeitabschnitte (in unseren Fall Sekunden) warten und dann weitermachen.

Der davorgenannte Algorithmus aehnelt Algorithmen wie Dijkstra, ist aber in der Implementation den Vorgaben der Aufgabe angepasst.

Implementierung

Die Loesungsidee wurde in Python umgesetzt. Die Klassen `PathfindTile` und `PathfindGrid` sind für beliebig-dimensionale Objekte nutzbar, diese Funktion wird aber in der Aufgabe nicht benoetiegt.

Folgende Klassen wurden implementiert:

`PathfindTile(location: TileLocation, path: list[TileLocation], path_cost: int, maze_shape: tuple[int])`

Klasse zur Representation von einem Labyrinth-Feld mit spezifischen Informationen. Implementiert Funktionen, welche für das Arbeiten mit `set()`'s gebraucht wird, und eine Funktion `PathfindTile.adjacent()`, welche die zu den Feld naechste Felder generiert, mit der Berueecksichtigung von Labyrinth-groesse.

`PathfindGrid(self, grid: list, cost: tuple[int], shape: tuple[int])`

Klasse, welche das Labyrinth repraesentiert und in diesen auch die Wegsuche durchfuehrt.

Folgende Methoden wurden implementiert:

`importMaze(filename: str) -> PathfindGrid`

Importiert ein Labyrinth aus einer Datei und gibt das entsprechende PathfindGrid-Objekt zurueck.

`visualizeMazePath(pfg: PathfindGrid)`

Gibt den Labyrinth mit den eingezeichneten Weg mithilfe von Pfeil-aussehenden ASCII-Charactern (`^`, `<`, `>`, `v`) fuer Richtung und Ausrufezeichen (`!`) fuer Teleportation

zwischen den beiden Etagen. Gibt auch den Path Cost, bzw. wie viel Sekunden man braucht, um durch das Labyrinth mit dem Weg zu kommen.

Beispiele:

Ausgabe von zauberschule0.txt	Ausgabe von zauberschule1.txt
Path Cost: 8 ############# #.....#.....# #.###.#.###.# #...#.#...#.# ###.#.###.#.# #...#.....#.# #.#########.# #.....#.....# #####.#.###.# #....A#B..#.# #.#########.# #...........# ############# ############# #.......#...# #...#.#.#...# #...#.#.....# #.###.#.#.### #.....#.#...# #####.###...# #.....#.....# #.#########.# #...#>>!....# #.#.#.#.###.# #.#...#...#.# #############	Path Cost: 4 #################### #...#.....#...#.....# #.#.#.###.#.#.#.###.# #.#.#...#.#.#...#...# ###.###.#.#.#####.### #.#.#...#.#B....#...# #.#.#.###.#^###.##### #.#...#.#.#^<A#.....# #.#####.#.#########.# #.................# #################### #################### #.......#.....#.....# #.###.#.#.###.#.###.# #.....#.#.#.....#.#.# #######.#.#######.#.# #.....#.#.....#...#.# #.###.#.#.###.###.#.# #.#.#...#.#...#...#.# #.#.#######.###.###.# #...........#.......# ####################

Ausgabe von zauberschule2.txt

```
Path Cost: 14
###############################################
#...#.....#..........#.#..........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#A>!#v#.....#.#.#...#...#
###.###.#.#.##############v#.#.###.#.###.#.###
#.#.#...#.#..............>>B#.#...#.#...#.#.#
#.#.#.###.############.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
###############################################

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

Ausgabe von zauberschule4.txt

```
Path Cost: 28
################################
#...#.....#..........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.###########.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.#######.###.#.#####
#...#.#...#...#.#...#...#.#...#
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.##############.#.#######
#.#.#.#..............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
###.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#>>B#.#...#.#
#.#########.#######^#.###.###.#
#..A#>>>>v#.#>>>>>>^#...#.#.#.#
#.#v#^###v#.#^#########.#.#.#.#
#.#>>^..#>>>>^#...........#...#
################################

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

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

Fuer Beispiele 4 und 5 wurden jeweils die Dauer gefragt:

```
Path Cost (zauberschule4.txt): 84
Path Cost (zauberschule5.txt): 124
```

Quelltext

Die Kommentare wurden wegen Gewohnheit in Englisch geschrieben. Alles, was für selbstbeschreibenden Code gehalten wurde, wurde optional kommentiert.

```python
import itertools

# types purely for signature readability
TileLocation = tuple[int]

class PathfindTile:

    def __init__(self, location: TileLocation, path: list[TileLocation], path_cost: int, maze_shape: tuple[int]):
        self.location = location
        self.path = path
        self.path_cost = path_cost
        self.maze_shape = maze_shape

    def adjacent(self):
        # generates tiles that are adjacent to the current tile, by iterating over every dimension and getting the
tile coordinates that are offset by +1/-1
        # doesn't go out of bounds using the maze shape
        for inx in range(len(self.location)):
            if self.location[inx] + 1 < self.maze_shape[inx]:
                adj = list(self.location)
```

```
                adj[inx] += 1
                yield tuple(adj), inx
            if self.location[inx] - 1 >= 0:
                adj = list(self.location)
                adj[inx] -= 1
                yield tuple(adj), inx

    def __hash__(self):
        # hash function to work properly with sets
        return self.location.__hash__()

    def __eq__(self, other):
        # equality function to work properly with sets
        if isinstance(other, PathfindTile):
            return self.location == other.location
        return self.location == other

class PathfindGrid:

    WALL = "#"

    def __init__(self, grid: list, cost: tuple[int], shape: tuple[int]):
        self.grid = grid
        self.movement_cost = cost
        self.shape = shape

    def pathfind(self, begin_tile: TileLocation, end_tile: TileLocation):
        # pathfinds from begin_tile to end_tile. returns the path cost and the path itself.

        edges: list[PathfindTile] = [
            PathfindTile(location=begin_tile,
                         path=[begin_tile],
                         path_cost=0,
                         maze_shape=self.shape
            )
        ]
        visited: set[TileLocation] = set()

        path_cost = 0

        while True:
            next_edges: list[PathfindTile] = list()
            for tile in edges:
                if path_cost < tile.path_cost:
                    next_edges.append(tile)
                    continue
                if tile == end_tile:
                    return tile.path, tile.path_cost

                for adjacent_tile, direction in tile.adjacent():
```

```
                    tile_value = self.getTileValue(adjacent_tile)

                    if tile_value == self.WALL or adjacent_tile in visited:
                        continue

                    path = tile.path + [adjacent_tile]
                    cost = tile.path_cost + self.movement_cost[direction]

                    next_tile = PathfindTile(location=adjacent_tile, path=path, path_cost=cost,
maze_shape=self.shape)
                    next_edges.append(next_tile)

            edges = next_edges.copy()

            visited = visited.union((next_tile.location for next_tile in next_edges))

            path_cost += 1

    def getTileValue(self, loc: list[TileLocation]) -> str:
        # aqcuires a tile value from any-dimensional grids
        item = self.grid
        for _loc in loc:
            item = item[_loc]
        return item


def importMaze(filename: str) -> PathfindGrid:
    with open(filename, "r") as f:
        data = f.read().split("\n")
    height, width = map(int, data[0].split())
    grid = [list(k) for k in data[1:height+1]], [list(k) for k in data[height+2:2*height+2]]
    return PathfindGrid(
        grid = grid,
        cost = (3, 1, 1),
        shape = (2, height, width)
    )

def visualizeMazePath(pfg: PathfindGrid):
    def searchTileValue(value, grid: PathfindGrid):
        # itertools.product generates the cartesian product, equivalent to a nested for-loop
        # (gives us all the possible coordinates that we have to go through in a singular for-loop
        # any amount of dimensions)
        for loc in itertools.product(*[range(k) for k in grid.shape]):
            if grid.getTileValue(loc) == value:
                return loc
    begin_tile = searchTileValue("A", pfg)
    end_tile = searchTileValue("B", pfg)

    path, cost = pfg.pathfind(begin_tile, end_tile)
    grid = pfg.grid
```

```
    for inx, current_tile in enumerate(path[:-1]):

        next_tile = path[inx+1]

        # in the 2 cases below, due to implementation, each value represents the folllowing:
        #   - x/dx describes the top/bottom level,
        #   - y/dy the row, and
        #   - z/dz the column
        dx, dy, dz = tuple([nxt - cur for nxt, cur in zip(next_tile, current_tile)])

        x, y, z = current_tile

        grid[x][y][z] = {
            (1, 0, 0) : "!",
            (-1, 0, 0) : "!",
            (0, 1, 0) : "v",
            (0, -1, 0) : "^",
            (0, 0, 1) : ">",
            (0, 0, -1) : "<"
        }[dx, dy, dz]

    grid[begin_tile[0]][begin_tile[1]][begin_tile[2]] = "A"

    # (an unholy) way to string up the maze in a singular row instead of looping 3 times.
    string = "\n\n".join("\n".join("".join(row) for row in level) for level in grid)

    return f"Path Cost: {cost}\n" + string


if __name__ == "__main__":
    for i in range(5):
        res = visualizeMazePath(importMaze(f"files/zauberschule{i}.txt"))
        with open(f"output/zauberschule{i}.txt","w") as f:
            f.write(res)
```