42. Bundeswettbewerb Informatik

Runde 1

Aufgabe 3: Zauberschule

Team-ID: 00380

Team-Name: CTRL-C / CTRL-V

Bearbeiter dieser Aufgabe:

Nicolas Beninde

20. November 2023

Inhaltsverzeichnis

1. Lösungsidee

Die in dem Problem gegebenen dreidimensionalen Layouts können sich als Graphen vorgestellt werden, dessen Knoten die einzelnen Punkte im Layout und dessen Kanten die Verbindungen zwischen diesen darstellen. Die Kanten tragen den Aufwand der Bewegung als Gewicht. Die unterschiedliche Gewichtung der Kanten stellt ein Problem für die Optimierung eines einfachen Suchalgorithmus wie BFS oder DFS dar. Daher wird für dieses Problem der Dijkstra-Algorithmus verwendet. Dieser hat für das Finden und Entfernen des kleinsten Elements eine Laufzeit von O(n) und für das Aktualisieren der Daten eine Laufzeit von (6*n). Da in diesem Beispiel keine Priority-Queue verwendet wird, die das Finden des kleinsten Elements erleichtern würde, ist die Laufzeit quadratisch $O(n^2)$.

2. Umsetzung

Das Programm wird in Python implementiert.

```
#returns wether a position is in bounds of the layout and on a path or not
def isValidPosition(layout, dimX, dimY, x, y, z):

    #checks if the position is in bounds
    if(0 <= x < dimX and 0 <= y < dimY and 0 <= z < 2):

        #returns if a wall is in the position
        return layout[x][y][z] != "#"

    #if the position is out of bounds False is returned
    return False
```

Aufgrund des quaderförmigen Aufbaus des Layouts bietet sich ein dreidimensionales Array als Datenstruktur an auf dem die Berechnung stattfinden kann. Dadurch entsteht aber die Situation, dass Nachbarn nicht in gültig und ungültig unterscheidet werden und, da manche Positionen unbegehbar sind, ergibt sich das Problem, dass Positionen erst auf ihre Gültigkeit überprüft werden müssen. Dies lässt sich mit der Funktion isValidPosition() machen, die prüft, ob sich eine Position im Array befinden würde und ob sich an dieser Position eine Wand befindet.

```
#returns the collected path information, the shortest distance from the start
#to the goal and the coordinates of the goal
def findShortestPathWithDijkstra(layout, dimX, dimY, startX, startY, startZ):

    #coordinates to save current position and direction tuple are initialized
    x, y, z = startX, startY, startZ
    movement = [(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0,
0, -1)]

    #library that saves distance to start, predecessor and visit status for
coordinates is initialized and data of starting point is entered
    data = {}
    data[startX, startY, startZ] = [0, None, True]
```

Der Hauptalgorithmus ist nicht weiter in Unterfunktionen aufgeteilt. Für die Implementierung von Dijkstra sind zwei Instanzen von Datenstrukturen hier besonders wichtig. Einmal der Tuple movement, der aus 6 Sub-Tuples besteht, die die 6 Richtungen darstellen, in die man sich bewegen kann, und das Bewegen von Position zu Position ermöglicht. Ein Tuple bietet sich hier an, da feste sich nicht verändernde Werte gespeichert werden. Zum anderen die Library data, auf die man mit

einem Schlüssel, bestehend aus den x, y und z-Koordinaten, zugreifen kann und die für jede Position die bisher kürzeste Entfernung, den Vorgänger und den Besucht-Status speichert. Eine Library bietet sich hier an, da sie einen unkomplizierten Zugriff auf eingabespezifische Werte ermöglicht.

```
#iterates through different positions till the goal is found or all positions
#where checked
while True:

    #current position is marked in library
    data[x, y, z][2] = True

    #if the current position is the goal, the collected path information,
    the shortest distance from the start to the goal and the coordinates
    of the goal is returned
    if layout[x][y][z] == "B":
        return data, shortestDistance, x, y, z

    #iteration through all six directions from current position
    for dX, dY, dZ in movement:

        #if the new positon is valid, the distance to the current position and
        the distance to the new position, if there is one, is extracted from
        the library and saved
        #if there is no entry for the new position in the library yet, the
        distance is saved as infinite
        if isValidPosition(layout, dimX, dimY, x + dX, y + dY, z + dZ):
        distanceToCurrentPosition = data[(x, y, z)][0]
        distanceToNewPosition = data.get((x + dX, y + dY, z + dZ),
        [float('inf')])[0]
```

```
#if the new position is on the same layer as the current, it is checked if the
#distance to the current position + 1 is smaller than currently saved distance
#if the new position is on a different layer as the current, it is checked if
the distance to the current position + 3 is smaller than currently saved
distance
#if that is the case the library entry to the new position is overwritten with
the new distance, the current position as the predecessor and the previous
visit status
#if there was no previous entry, a new one is created with the calculated
distance, the current position as the predecessor and the visit status as
False
if dZ == 0 and (distanceToNewPosition > distanceToCurrentPosition + 1):
data[x + dX, y + dY, z + dZ] = [distanceToCurrentPosition + 1, (x, y, z),
data.get((x + dX, y + dY, z + dZ), [None, None, False])[2]]
if dZ != 0 and (distanceToNewPosition > distanceToCurrentPosition + 3):
data[x + dX, y + dY, z + dZ] = [distanceToCurrentPosition + 3, (x, y, z),
data.get((x + dX, y + dY, z + dZ), [None, None, False])[2]]
```

Nun werden in einer while-Schleife alle Knoten durchlaufen bis der Zielknoten gefunden wurde oder, wie im nächsten Codeblock zu sehen, alle verfügbaren Knoten durchlaufen wurden. Der aktuelle Knoten wird sich über die drei Koordinaten gemerkt. In jeder Iteration wird der aktuelle Knoten zu

Beginn markiert. Danach folgt die Analyse neuer Daten. Dazu wird die neue Position auf ihre Gültigkeit überprüft und eine eventuelle bisherige Distanz des neuen Knoten ausgelesen. Sollte kein Eintrag vorhanden sein, wird die bekannte Distanz auf ‘inf‘ gesetzt. Nach der Unterscheidung ob Ebenenwechsel stattfindet oder nicht, wird beim Finden eines kürzeren oder ersten Weges, die Daten in die Library geschrieben. Der Ausdruck [None, None, False] dient einer Fehlervorbeugung, da sich der Besucht-Status im [2] Element befindet und im Fall, dass noch kein Eintrag vorhanden ist wird der Index 2 aus dieser Rückgabe also False extrahiert.

```
#to find a new position a tuple saving the new position and an integer saving
#the current shortst distance are initialized
shortestDistance = float('inf')
newNode = ()

#iteration through the whole library
for values in data.items():

    #if the distance of the current element is smaller than the
    currentlysaved distance and the element was not visited yet,
    #the new shortest distance and the coordinates of the element are saved
    distanceOfCurrentElement, visitStatus, coordinatesOfCurrentElement =
    values[1][0], values[1][2], values[0]
    if distanceOfCurrentElement < shortestDistance and visitStatus == False:
        shortestDistance = distanceOfCurrentElement
        newNode = coordinatesOfCurrentElement

#if all positions where visited an error message is returned
if newNode == (): return "no valid path from start to goal"

#the new position is set for the next iteration
x, y, z = newNode
```

Zuletzt wird der Knoten für die nächste Iteration bestimmt. Dazu werden alle Werte aus der Library extrahiert und die kürzeste Distanz mithilfe der Variable shortestDistance während einer Iteration durch alle Werte bestimmt. Bedingung für einen gültigen Knoten ist zusätzlich zu einer kürzeren Distanz als das momentane Minimum, dass der Knoten bisher noch nicht besucht wurde. Nach der Iteration wird, wie oben erwähnt überprüft ob alle Knoten besichtigt wurden bzw. kein Knoten unmarkiert war und die Iteration, sollte ein Knoten gefunden worden sein, fortgesetzt.

3. Beispiele

Das Programm gibt zu einem gültigen Layout eines Gebäudes die optimale Route vom Start- zum Zielpunkt, eingezeichnet in den Gebäudeplan, aus. Der Startknoten A ist wie in der Beispielausgabe auf der Website durch ein Richtungszeichen ersetzt. Zusätzlich werden die Dimensionen des Layouts, der Aufwand sowie die Koordinaten des Start- und Zielpunktes ausgegeben. Des Weiteren berechnet das Programm mit der Methode perf_counter() der time-Bibliothek die Zeit, die das Programm benötigt hat. zaubeschule4.txt und zauberschule5.txt sind hier nur ohne Route dokumentiert. Die Pläne stehen aufgrund ihrer Größe, wie durch die Organisation erlaubt, in einem zusätzlichen Order output.

3.1 zauberschule0.txt

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

#############
#.......#...#
#...#.#.#...#
#...#.#.....#
#.###.#.#.###
#.....#.#...#
#####.###...#
#.....#.....#
#.#########.#
#...#>>!....#
#.#.#.#.###.#
#.#...#...#.#
#############
Dimensionen: 13 13
Startpunkt: 5 9 0       Zielpunkt: 7 9 0
Aufwand: 8              Laufzeit: 0.006
```

3.2 zauberschule1.txt

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

```
####################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#...........#.......#
####################
Dimensionen: 21 11
Startpunkt: 13 7 0      Zielpunkt: 11 5 0
Aufwand: 4              Laufzeit: 0.005
```

3.3 zauberschule2.txt

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

#############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
#############################################
Dimensionen: 45 15
Startpunkt: 21 3 0      Zielpunkt: 27 5 0
Aufwand: 14             Laufzeit: 0.009
```

3.4 zauberschule3.txt

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

```
################################
#.......#.....#...#...#.......#
#.###.#.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.#.....#.#.#.....#
#######.#.#.#.#.#####.#.#####.#
#.....#.#.#.#.#.#...#...#.....#
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.#########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#!#.#.#.#.#
#.#.#######.#.#.#####^#.#.#.#.#
#.#.......#.#...#>>>>^#.#...#.#
#.###.#####.#####^#####.#####.#
#...#..>>>>>>>>>>^#..........#
################################
Dimensionen: 31 31
Startpunkt: 3 27 0      Zielpunkt: 21 25 0
Aufwand: 28             Laufzeit: 0.021
```

3.5 zauberschule4.txt

```
Dimensionen: 101 101
Startpunkt: 73 67 0     Zielpunkt: 31 55 0
Aufwand: 84             Laufzeit: 2.093
```

3.6 zauberschule5.txt

```
Dimensionen: 81 201
Startpunkt: 13 167 0    Zielpunkt: 71 155 0
Aufwand: 124            Laufzeit: 4.530
```

4. Quellcode

```
#returns wether a position is in bounds of the layout and on a path or not
def isValidPosition(layout, dimX, dimY, x, y, z):

    if(0 <= x < dimX and 0 <= y < dimY and 0 <= z < 2):
        return layout[x][y][z] != "#"

    return False


#returns the collected path information, the shortest distance from the start to the goal and the
#coordinates of the goal
def findShortestPathWithDijkstra(layout, dimX, dimY, startX, startY, startZ):

    x, y, z = startX, startY, startZ
    movement = [(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1)]

    #library that saves distance to start, predecessor and visit status for coordinates is initialized
    data = {}
    data[startX, startY, startZ] = [0, None, True]

     #iterates through different positions till the goal is found or all positions where checked
    while True:

        #current position is marked in library
        data[x, y, z][2] = True

        if layout[x][y][z] == "B":
            return data, shortestDistance, x, y, z

        #iteration through all six directions from current position
        for dX, dY, dZ in movement:

            #if the new positon is valid, the distance to the current position and the distance to the
            #new position, if there is one, is extracted from the library and saved
            #if there is no entry for the new position in the library yet, the distance is saved as
            #infinite
            if isValidPosition(layout, dimX, dimY, x + dX, y + dY, z + dZ):
                distanceToCurrentPosition = data[(x, y, z)][0]
                distanceToNewPosition = data.get((x + dX, y + dY, z + dZ), [float('inf')])[0]

                #if the new position is on the same / different layer, it is checked if the distance to
                #the current position + 1 / + 3 is smaller than currently saved distance
                #if that is the case the library entry to the new position is overwritten with the new
                #distance, the current position as the predecessor and the previous visit status
                #if there was no previous entry, the the visit status is set as False
                if dZ == 0 and (distanceToNewPosition > distanceToCurrentPosition + 1):
                    data[x + dX, y + dY, z + dZ] = [distanceToCurrentPosition + 1,
```

```
                (x, y, z), data.get((x + dX, y + dY, z + dZ), [False])[0]]
            if dZ != 0 and (distanceToNewPosition > distanceToCurrentPosition + 3):
                data[x + dX, y + dY, z + dZ] = [distanceToCurrentPosition + 3,
                (x, y, z), data.get((x + dX, y + dY, z + dZ), [False])[0]]

#to find a new position a tuple saving the new position and an integer saving the   current
#shortst distance are initialized
shortestDistance = float('inf')
newNode = ()

#iteration through the whole library
for values in data.items():

    distanceOfCurrentElement, visitStatus, coordinatesOfCurrentElement =
    values[1][0],  values[1][2], values[0]

    if distanceOfCurrentElement < shortestDistance and visitStatus == False:
        shortestDistance = distanceOfCurrentElement
        newNode = coordinatesOfCurrentElement

#if all positions where visited an error message is returned
if newNode == (): return "no valid path from start to goal"

#the new position is set for the next iteration
x, y, z = newNode
```