Aufgabe A3: Zauberschule

Team-ID: 00250

pip install BWINF_42

Bearbeiter dieser Aufgabe: Imanuel Fehse (Teilnahme-ID: 69274)

06.11.2023

Inhaltsverzeichnis

1. Aufgabe

Hilf Ron und schreibe ein Programm, das für jeweils zwei Felder A und B in der Zauberschule einen Weg bestimmen kann, auf dem Ron so schnell wie möglich von A nach B gelangt.
Das Programm soll dazu auch den Plan von Bugwarts – und jeden anderen Plan der gleichen Art, in dem nur die Wände anders stehen – einlesen können. So kann Ron das Programm auch Hermine zur Verfügung stellen, die in der Zweigstelle Fameglitch Nachhilfeunterricht gibt.

2. Lösungsidee

Um den kürzesten Weg vom Starpunkt A, bis zum Endpunkt B zu finden, wird ein gewichteter, ungerichteter Graph[1] aus der Eingabedatei erstellt. Hierfür werden für jedes Feld als ein Knoten angesehen, der Verbindungen mit einem bestimmten Kantengewicht zu anderen Knoten hat. Das Kantengewicht einer Verbindung entspricht der Zeit, die benötigt wird, um das

[1] Präsentation der Uni Kassel über Graphen, die als Grundlage verwendet wurde: hier

entsprechende Feld von dem aktuellen Feld aus zu erreichen (auf selben Stockwerk 1 Sekunde = Kantengewicht 1; Stockwerkwechsel 3 Sekunden = Kantengewicht 3). Anschließend wird vom Starpunkt aus der kürzeste Weg und die benötigte Zeit zu jedem Knoten ermittelt. Als letzter Schritt wird in dem Ergebnis der kürzeste Weg ausgewählt, der vom Starpunkt A zum Endpunkt B geht und als Lösung ausgegeben.

3. Umsetzung

3.1. Algorithmus zur Erstellung des gewichteten Graphen

Als Grundlage für das Programm wird ein ungerichteter, gewichteter Graph (allgemein nur „gewichteter Graph“) genutzt, der aus der Ausgabedatei erstellt wird. Hierfür wird zunächst die Eingabedatei eingelesen. Anschließend werden dieser Datei die Dimensionen des Gebäudes entnommen, welche benötigt werden, um den gewichteten Graphen zu erstellen.

Zunächst wird die Eingabedatei in eine dreidimensionale Liste konvertiert, die das Gebäude repräsentiert (Implementierung). Beim erstellen dieser Liste ermittelt das Programm zugleich die Start- und Endposition. (Implementierung[2])

Als nächster Schritt, wird aus dieser Liste der gewichtete Graph erstellt. Dafür wird durch jedes Feld als ein Knoten betrachtet, der Verbindungen mit einem bestimmten Kantengewicht zu anderen Knoten hat. Der Algorithmus (Implementierung[3]) wird durch den folgenden Pseudocode dargestellt:

```
FUNCTION get_weighted_graph():
    Create empty graph
    FOR z in height of building:
        FOR y in depth of building:
            FOR x in width of building:
                IF field is not a wall:
                    Create list that stores all connections to current node
                    IF field in y+ direction is not a wall:
                        Create connection to this node with weight of 1
                    IF field in x+ direction is not a wall:
                        Create connection to this node with weight of 1
                    IF field in y- direction is not a wall:
                        Create connection to this node with weight of 1
                    IF field in x- direction is not a wall:
                        Create connection to this node with weight of 1
                    IF field on the other story is not a wall:
                        Create connection to this node with weight of 3
                    Add current node with all connections to graph
    RETURN weighted graph
```

[2] Algorithmus zur Erstellung der Liste, die die Konstruktion des Gebäudes repräsentiert: hier
[3] Algorithmus zur Erstellung des gewichteten Graphen: hier

Der Algorithmus durchläuft jedes Feld im Gebäude und überprüft, ob die Felder um das aktuelle Feld herum (nicht diagonal versetzte Felder), sowie das Feld auf dem jeweils anderen Stockwerk, Wände sind. Ist eines dieser Felder keine Wand, so wird eine Verbindung vom aktuellen Knoten zum entsprechenden Knoten erstellt. Jede Verbindung hat ein festgelegtes Kantengewicht. Ist der Knoten, zu dem die Verbindung erstellt wird, auf demselben Stockwerk wie der aktuelle Knoten, so wird das Kantengewicht der Verbindung auf 1 (1 Sekunde) gesetzt. Ist der Knoten auf dem anderen Stockwerk, so wird das Kantengewicht auf 3 (3 Sekunden) gesetzt. Wurden alle Verbindungen für den aktuellen Knoten ermittelt, so wird dieser mit allen Verbindungen und entsprechenden Kantengewichten zum gewichteten Graphen hinzugefügt.

3.2. Dijkstra Algorithmus

Sobald das Programm über den gewichteten Graphen verfügt, kann der kürzeste Weg vom Start- zum Endpunkt ermittelt werden. Hierfür wird der Dijkstra-Algorithmus verwendet (Quelle[4]). Dieser Algorithmus wird dazu verwendet, um den kürzesten Weg zwischen einem Startpunkt und jedem andern Punkt in einem gewichteten Graphen zu ermitteln, der entweder gerichtet oder ungerichtet ist. Der folgende Pseudocode repräsentiert die Funktionsweise des Dijkstra-Algorithmus:

```
1  FUNCTION Dijkstra (graph, startpoint):
2      Distance to startpoint = 0
3      Distance to every other point = infinity
4      Create a list with every node that hasn't been visited already
5      WHILE there are nodes that have not been visited so far:
6          Get the current node with the smallest distance
7          IF current distance to the current node > distance calculated with the current
8              path:
9              Pass
10         ELSE:
11             FOR every node that has a connection with the current node:
12                 New distance = current distance to the current node + distance to
13                     the current neighbor
14             IF new distance < distance to the current neighbor:
15                 Update the distance to the current neighbor
16                 Update the path to the current neighbor (previous node)
```

Der Dijkstra-Algorithmus ermittelt vom Starpunkt aus die Distanz zu jedem andern Knoten, indem er jeden möglichen Pfad ermittelt und berechnet. Jeder Knoten erhält die Information, wie er am schnellsten erreicht wird und wie lange der Weg ist. Stößt der Algorithmus beim Berechnen auf einen Knoten, dessen Distanz schon einmal berechnet wurde, der aktuelle Weg aber ein kürzerer Weg zu dem entsprechenden Knoten ist, so wird dieser Knoten mit dem

[4] Informationen und Erklärung des Dijkstra-Algorithmus der Technische Universität München: hier

aktuellen Weg überschrieben. Auf diese Weise wird garantiert, das für jeden Knoten, der kürzeste Weg vom Startpunkt aus ermittelt wird.

Nun wird das Ergebnis dieses Algorithmus genutzt um den kürzesten Weg zum Endpunkt zu erhalten. Hierfür wird einfach der mit dem Dijkstra-Algorithmus ermittelte Weg zum Endpunkt ausgewählt, welcher die Lösung ist. Der Algorithmus gibt diesen Weg, genauso wie die Länge dieses Weges als Ergebnis zurück (Implementierung[5]).

3.3. Speichern des Ergebnisses

Nachdem der kürzeste Weg vom Starpunkt A zum Endpunkt B gefunden wurde, so wird dieser Weg und seine Länge abgespeichert. Um den Plan zu erstellen, der als Lösung ausgegeben wird, werden die ursprünglichen Daten der Eingabedatei verwendet. Für jeden Schritt im Lösungsweg wird ermittelt, in welche Richtung der Schritt führt. Ist die Richtung des Schritts ermittelt, so wird ein entsprechendes Zeichen an dieser Stelle in der Grafik eingefügt (Implementierung[6]). Folgende Zeichen werden für die entsprechenden Bewegungen eingefügt:

Bewegung	Zeichen
X+	>
X-	<
Y+	^
Y-	V
Stockwerkwechsel	!

Wurde jeder Schritt in die Grafik eingefügt, so wird diese Grafik als Textdatei abgespeichert. (Implementierung[7])

[5] Dijkstra Algorithmus mit Auswahl des Weges zum Endpunkt: hier
[6] Implementierung der Erstellung der Grafik: hier
[7] Implementierung des Speicherns des Ergebnisses als Textdatei: hier

4. Beispiele

Zauberschule0.txt

```
distance from A to B: 8
building with path:
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
###.#.###.#.#
#...#.....#.#
#.#########.#
#.....#.....#
#####.#.###.#
#....!#B..#.#
#.#########.#
#...........#
#############

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

Zauberschule1.txt

```
distance from A to B: 4
building with path:
######################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
#.#...#.#.#^<<#.....#
#.#####.#.#########.#
#...................#
######################

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

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

Zauberschule2.txt

```
distance from A to B: 14
building with path:
###############################################
#...#.....#...........#.#...........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#>>!#v#.....#.#.#...#...#
###.###.#.#.#############v#.#.###.#.###.#.###
#.#.#...#.#.............>>B#.#...#.#...#.#.#
#.#.#.###.###########.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
###############################################

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

Zauberschule3.txt

```
distance from A to B: 28
building with path:
################################
#...#.....#...........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
```

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

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

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

Zauberschule4.txt

Aufgrund der Länge der Ausgabedatei, ist nur die Länge des kürzesten Pfades angegeben, der den Startpunkt mit dem Endpunkt verbindet. Der gesamte Pfad kann in der Ausgabedatei im Verzeichnis gefunden werden.

```
distance from A to B: 84
```

Zauberschule5.txt

Aufgrund der Länge der Ausgabedatei, ist nur die Länge des kürzesten Pfades angegeben, der den Startpunkt mit dem Endpunkt verbindet. Der gesamte Pfad kann in der Ausgabedatei im Verzeichnis gefunden werden.

```
distance from A to B: 124
```

5. Wichtige Teile des Codes

```
def convert_from_file(self):
    def get_floor(y_start, y_end, floor):
        # print("get_floor", y_start, y_end, floor)
        c_floor = []
        for y in range(y_start, y_end):
            c_line = []
            for x in range(self.size_x):
                # if field is a free
                if self.data[y][x] == '.':
                    c_line.append([x, y - y_start, floor])
                # if start position
                elif self.data[y][x] == 'A':
                    c_line.append([x, y - y_start, floor])
                    self.start_pos = [x, y - y_start, floor]
                # if end position
                elif self.data[y][x] == 'B':
                    c_line.append([x, y - y_start, floor])
                    self.end_pos = [x, y - y_start, floor]
                # if field is a wall
                else:
                    c_line.append([])
            # add current line to current floor
            c_floor.append(c_line)
        return c_floor

    # first floor
    self.building.append(get_floor(1, self.size_y+1, 0))
    # second floor
    self.building.append(get_floor(self.size_y+2, self.size_y*2+2, 1))
```

[C1] Algorithmus zur Erstellung der Liste, die die Konstruktion des Gebäudes repräsentiert

```
def get_weighted_graph(self):
    for z in range(len(self.building)):
        for y in range(len(self.building[z])):
            for x in range(len(self.building[z][y])):
                # if field is not a wall
                if self.building[z][y][x]:
                    c_connections = []
                    # do not need to check is field is at the end of the
building, because outer ring of building
                    # is a wall
                    # check for surrounding fields
                    # check for field in y+ direction
                    if self.building[z][y + 1][x]:
                        c_connections.append([[x, y+1, z], 1])
                    # check for field in x+ direction
                    if self.building[z][y][x + 1]:
                        c_connections.append([[x+1, y, z], 1])
                    # check for field in y- direction
                    if self.building[z][y - 1][x]:
```

```
                    c_connections.append([[x, y-1, z], 1])
                # check for field in x- direction
                if self.building[z][y][x - 1]:
                    c_connections.append([[x-1, y, z], 1])
                # check for field in other story
                if z == 0:
                    if self.building[z+1][y][x]:
                        c_connections.append([[x, y, z+1], 3])
                else:
                    if self.building[z-1][y][x]:
                        c_connections.append([[x, y, z-1], 3])
                # add connections to connections dictionary
                self.weighted_graph[str([x, y, z])] = c_connections
```

[C2] Algorithmus zur Erstellung des gewichteten Graphen

```
def dijkstra(self, start, end):
    # use dijkstra's algorithm to find the shortest path and keep track of
the path

    # create a dictionary to keep track of the path
    path = {}
    # create a dictionary to keep track of the distance
    distance = {knot: float('inf') for knot in self.weighted_graph}
    distance[str(start)] = 0
    queue = [(0, str(start))]
    while queue:
        # get the current distance and the current knot
        current_distance, current_knot = heapq.heappop(queue)
        # check if the current distance is smaller than the distance in the
dictionary
        if current_distance > distance[current_knot]:
            continue

        for neighbour, weight in self.weighted_graph[current_knot]:
            # calculate the new to the neighbour
            distance_to_neighbour = current_distance + weight
            # check if the new distance is smaller than the distance in the
dictionary
            if distance_to_neighbour < distance[str(neighbour)]:
                # update the distance in the dictionary
                distance[str(neighbour)] = distance_to_neighbour
                # update the path in the dictionary
                path[str(neighbour)] = current_knot
                # add the new distance and the new knot to the queue
                heapq.heappush(queue, (distance_to_neighbour,
str(neighbour)))
    # create a list to keep track of the path
    shortest_path = []
    # get the current knot
    current_knot = str(end)
    # add the current knot to the list
    shortest_path.append(current_knot)
```

```
    # loop until the current knot is the start knot
    while current_knot != str(start):
        # update the current knot
        current_knot = path[current_knot]
        # add the current knot to the list
        shortest_path.append(current_knot)
    shortest_path.append(str(start))
    # reverse the list
    shortest_path.reverse()
    # return the list and distance to reach the endpoint
    return [shortest_path, distance[str(end)]]
```

[C3] Dijkstra Algorithmus mit Auswahl des Weges zum Endpunkt

```
def create_path(self):
    # create graph of building
    # for every floor (z)
    for z in range(len(self.building)):
        c_floor = []
        # for every line (y)
        for y in range(len(self.building[z])):
            c_line = []
            # for every field (x)
            for x in range(len(self.building[z][y])):
                # if field is not a wall
                if self.building[z][y][x]:
                    c_line.append('.')
                # if field is a wall
                else:
                    c_line.append('#')
            c_floor.append(c_line)
        self.res_graph.append(c_floor)
    # add path to res_graph
    # for every field in path
    for i in range(len(self.path)):
        c_field = []
        for item in self.path[i].strip("[]").split(","):
            c_field.append(int(item))

        # if current field is not the end field
        if i != len(self.path)-1:
            # if path moves in x+ direction
            if c_field[0] < int(self.path[i+1].strip("[]").split(",")[0]):
                self.res_graph[c_field[2]][c_field[1]][c_field[0]] = '>'
            # if path moves in x- direction
            elif c_field[0] >
int(self.path[i+1].strip("[]").split(",")[0]):
                self.res_graph[c_field[2]][c_field[1]][c_field[0]] = '<'
            # if path moves in y+ direction
            elif c_field[1] <
int(self.path[i+1].strip("[]").split(",")[1]):
                self.res_graph[c_field[2]][c_field[1]][c_field[0]] = 'v'
            # if path moves in y- direction
```

```
            elif c_field[1] >
int(self.path[i+1].strip("[]").split(",")[1]):
                self.res_graph[c_field[2]][c_field[1]][c_field[0]] = '^'
            # if path moves in z+ direction
            elif c_field[2] <
int(self.path[i+1].strip("[]").split(",")[2]):
                self.res_graph[c_field[2]][c_field[1]][c_field[0]] = '!'
            # if path moves in z- direction
            elif c_field[2] >
int(self.path[i+1].strip("[]").split(",")[2]):
                self.res_graph[c_field[2]][c_field[1]][c_field[0]] = '!'
        # if current field is the end field
        else:
            self.res_graph[c_field[2]][c_field[1]][c_field[0]] = 'B'
```

[C4] Einfügen des Lösungsweges in die Grafik

```
def save_result(self):
    print("save result")
    f = open('result_a3_' + self.filename + '.txt', 'w')
    f.write("distance from A to B: " + str(self.distance) + '\n')
    f.write("building with path:\n")
    for z in range(len(self.res_graph)):
        for y in range(len(self.res_graph[z])):
            line = ""
            for x in range(len(self.res_graph[z][y])):
                line += str(self.res_graph[z][y][x])
            f.write(line + '\n')
        f.write('\n')
    f.close()
```

[C5] Speichern der Grafik als Textdatei

6. Quellen

- **[Q1]** Uni Kassel: Graphen (genutzter Teil: ungerichtete, gewichtete Graphen) https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ss2009/algorithmen/folien/9_Graphen_Handout.pdf
  05.11.2023 | 20:53 Uhr
- **[Q2]** Technische Universität München: Dijkstra-Algorithmus
  https://algorithms.discrete.ma.tum.de/graph-algorithms/spp-dijkstra/index_de.html
  06.11.2023 | 21:04 Uhr