Aufgabe 3: Zauberschule

Teilnahme-ID: 68772

Team-ID: 00083

Bearbeiter/-in dieser Aufgabe:
Linus Schumann

19. November 2023

Inhaltsverzeichnis

1 Lösungsidee

1.1 Feststellung der Aufgabe

Grundsätzlich besteht die Aufgabe darin, innerhalb des Hauses den kürzesten Weg von Punkt A zu Punkt B zu finden. Im Haus existieren dabei zwei Etagen, die jeweils als Matrix dargestellt werden können. Dabei haben beide Matrizen die Größe $n * m$. Auf diesen Matrizen kann sich frei in alle 4 Richtungen (oben, unten, rechts, links) bewegt werden. Wände sind dabei auf der Matrix als eigenes Feld dargestellt. Eine normale Bewegung kostet einen Schritt, das Durchqueren des Bodens bzw. der Decke kostet 3 Schritte.

1.2 Lösungsansatz

Um den kürzesten Weg zwischen A und B zu finden, wird der Dijkstra-Algorithmus verwendet. Eine weitere Erläuterung des Dijkstra-Algorithmus lässt sich in Abschnitt 1.3 Finden.
Dieser Dijkstra-Algorithmus kann dann auch im Nachhinein so erweitert werden, dass der Weg rekonstruiert werden kann. Dafür muss nur für jede Zelle gespeichert werden, aus welcher Zelle sie mit dem geringsten Aufwand erreicht wurde.

1.3 Dijkstra-Algorithmus

Der Dijkstra-Algorithmus ist ein Algorithmus zur Bestimmung des kürzesten Weges von einem Startknoten zu allen anderen Knoten in einem Graphen. Der Algorithmus arbeitet dabei mit einer Prioritätswarteschlange. In dieser Warteschlange werden die Knoten gespeichert, die noch nicht abgearbeitet wurden. Dabei wird der Knoten mit dem geringsten Aufwand immer zuerst abgearbeitet.
Der Algorithmus arbeitet dabei wie folgt:

1. Initialisiere alle Knoten mit einem Aufwand von ∞
2. Setze den Aufwand des Startknotens auf 0 und füge ihn der Warteschlange hinzu
3. Solange die Warteschlange nicht leer ist:
4. Entferne den Knoten mit dem geringsten Aufwand aus der Warteschlange
5. Berechne den Aufwand zu diesem Knoten und ändere den Aufwand des Knotens, falls dieser geringer ist als der alte:
6. Speichere den Vorgänger des Knotens, bei dem der Aufwand geändert wurde
7. Für alle Nachbarn des Knotens:
8. Füge den Nachbarn der Warteschlange hinzu

Der Algorithmus terminiert, wenn die Warteschlange leer ist.

2 Umsetzung

2.1 Allgemeines

Im Folgenden wird die Umsetzung, der in Abschnitt 1 beschriebene Lösungsidee, näher erläutert. Grundsätzlich wurde diese Idee dabei in C++, genauer gesagt in der Datei "Aufgabe_3.cpp" implementiert. Diese Datei befindet sich im Verzeichnis "./source/".

Um das implementierte Programm zu starten, kann das Batch Skript "Aufgabe_3.bat" im Verzeichnis "./executables/" genutzt werden. Dieses startet das von mir für Windows kompilierte Programm ("Aufgabe_3.exe"). Für andere Betriebssysteme müsste die Source-Datei erneut auf dem entsprechenden Rechner kompiliert werden.

Im Verzeichnis "./beispieleingaben/" befinden sich alle in dieser Dokumentation aufgeführten Beispiele und im Verzeichnis "./beispielausgaben/" befinden sich dementsprechend die gesicherten Ausgaben. Damit Letztere besser von den Beispieldaten unterschieden werden können, werden diese mit der Dateiendung "_out.txt" gespeichert.

2.2 Implementierung der Lösungsidee

Nun werden die einzelnen Bestandteile der Implementierung näher erläutert.

2.2.1 Main-Funktion

Die Main-Funktion ist der Einstiegspunkt des Programms. Sie liest zuerst den Namen der Input-Datei ein und öffnet diese. Dann wird diese Input-Datei mit der Funktion "readData()" eingelesen und der eigentliche Algorithmus kann starten. Dabei wird zuerst "findPath()", dann "buildPath()", dann "printPath()" und schließlich "generateFileOutput()" aufgerufen.

2.2.2 findPath-Funktion

Die findPath-Funktion ist die Funktion, in der der eigentlichen Dijkstra-Algorithmus implementiert wurde. Dieser wird wie in Abschnitt 1.3 beschrieben mit Hilfe einer Prioritätswarteschlange umgesetzt. Dazu wird die C++ Klasse "priority_queue" genutzt, die in der Standardbibliothek von C++ enthalten ist.

2.2.3 buildPath-Funktion

Die buildPath-Funktion ist die Funktion, die den kürzesten Weg rekonstruiert. Dazu wird bei B gestartet und der Weg wird Schritt für Schritt zurückverfolgt. Dabei wird der Weg in einem Vektor gespeichert und anschließend noch umgedreht, damit er in der richtigen Reihenfolge ausgegeben wird.

2.2.4 printPath-Funktion

Hier wird einfach nur der Pfad für den Benutzer ausgegeben. Dabei werden alle Punkte des Pfades mit ihren x, y, z Koordinaten angegeben. Der erste Punkt stellt natürlich dabei A und der letzte Punkt B dar.

2.2.5 generateFileOutput-Funktion

In dieser Funktion wird die Ausgabedatei generiert. In der Datei wird ein ASCII-Artbild des Hauses ausgegeben. Es wird also die Eingabedatei kopiert und der Pfad wird ergänzt. Dabei werden die Richtungen mit den Zeichen ">", "<", "∧" und "v" dargestellt. Dabei steht ">" für rechts, "<" für links, "∧" für oben und "v" für unten. Ein Wechsel der Etage wird mit einem "!" dargestellt. Dieses wird dabei auf beiden Etagen ausgegeben.

2.3 Laufzeitanalyse

Die Laufzeit des Programms ist hauptsächlich von der Laufzeit des Dijkstra-Algorithmus abhängig. In meiner Implementierung des Dijkstra Algorithmus liegt diese Laufzeit in der Klasse $O(n^2)$. Die Laufzeit der Funktionen "buildPath" und "printPath" ist nur linear abhängig von der Anzahl der Punkte im Pfad. Als Laufzeit würde sich also für diese beiden Funktionen die Komplexitätsklasse $O(|Pfad|)$ (bzw. $O(2 * |Pfad|)$) ergeben. Dabei ist $|Pfad|$ die Anzahl der Punkte im Pfad. Die Laufzeit der Funktion generateFileOutput ist aufgrund der Ausgabe von allen Punkten im Haus abhängig von der Anzahl der Punkte im Haus $(n * m)$. Die Gesamtlaufzeit des Programms liegt also in der Komplexitätsklasse $O(n^2)$ bzw. $O(n * m)$.

3 Beispiele

Im Folgenden werden alle Ausgaben der Beispiele aufgeführt. Bei den Beispielen 0 bis 3 wird dabei die komplette Ausgabe aufgeführt. Bei den Beispielen 4 und 5 wiederum wird nur die Länge des kürzesten Pfades ausgegeben. Die vollständigen Ausgaben für Beispiel 4 und 5 lassen sich natürlich im Verzeichnis "./beispielausgaben/" finden.

3.1 Beispiel 0

```
Length of shortest path: 8
##############    ##############
#.....#.....#    #.......#...#
#.###.#.###.#    #...#.#.#...#
#...#.#...#.#    #...#.#.....#
###.#.###.#.#    #.###.#.#.###
#...#.....#.#    #.....#.#...#
#.##########.#    #####.###...#
#.....#.....#    #.....#.....#
#####.#.###.#    #.##########.#
#....!#B..#.#    #...#>>!....#
#.##########.#    #.#.#.#.###.#
#..........#    #.#...#...#.#
##############    ##############
```

Listing 1: Ausgabe Beispiel 0

3.2 Beispiel 1

```
Length of shortest path: 4
######################    ######################
#...#.....#...#.....#    #.......#.....#.....#
#.#.#.###.#.#.#.###.#    #.###.#.#.###.#.###.#
#.#.#...#.#.#...#...#    #.....#.#.#.....#.#.#
###.###.#.#.######.###    ########.#.########.#.#
#.#.#...#.#B....#...#    #.....#.#.....#...#.#
#.#.#.###.#^###.#####    #.###.#.#.###.###.#.#
#.#...#.#.#^<<#.....#    #.#.#...#.#...#...#.#
#.#####.#.##########.#    #.#.########.###.###.#
#.................#    #...........#.......#
######################    ######################
```

Listing 2: Ausgabe Beispiel 1

3.3 Beispiel 2

```
Length of shortest path: 14
################################################    ################################################
#...#.....#.....#.#.....#.#.....#.....#.....#    #...#.....#.....#.....#...#...#.....#.....#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#    #.#.#.#####.###.#.###.#.#.#.#.#.###.###.###
#.#.#...#.#.........#>>!#v#.....#.#.#...#...#    #.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#
###.###.#.#.###############v#.#.###.#.###.#.###    ###.#.###.#.#.###############.#.#####.###.###.#
#.#.#...#.#..............>>B#.#...#.#...#.#.#    #.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#.###.###########.#########.###.#.###.#.#    #.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#    #.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#    #.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#    #...#.#.#...#...#...#...#.#...#.#.....#.....#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#    #.###.#.#.#######.#####.#####.#.#.#.#######.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#    #...#...#...#...#.....#.......#.#.#.#.....#.#
#.###.##########.#.###.#.#.#.#.#.#.###.###.#    #.#.#######.#.#.#####.#.#######.#.###.###.#.#
#...#.................#...#.....#...#.......#    #.#...........#.......#.........#.......#...#
################################################    ################################################
```

Listing 3: Ausgabe Beispiel 2

3.4 Beispiel 3

```
Length of shortest path: 28
################################    ################################
#...#.....#...........#.......#    #.......#.....#...#...#.......#
#.#.#.###.#.###.#######.###.#.#    #.###.#.#.#.#.#.###.#.#.#.#####
#.#.#...#.#.#.#.#.......#...#.#    #.....#.#.#.#.#.....#.#.#.....#
###.###.#.#.#.#.#.#######.###.#    #######.#.#.#.#.#####.#.#####.#
#.#.#...#.#...#.....#.....#.#.#    #.....#.#.#.#.#.#...#...#.....#
#.#.#.###.###########.#####.#.#    #.###.#.###.#.###.#.#.###.#####
#.#...#.#.#.........#...#.....#    #...#.#.#...#.....#.#.#.#.....#
#.#####.#.#.#######.###.#.#####    #.#.###.#.#########.#.#.#####.#
#...#.#...#...#.#...#...#.#...#    #.#.....#.#.......#.#.....#...#
#.#.#.#.#####.#.#.###.###.#.#.#    #.#######.#######.#.#.#####.#.#
#.#.#.#.......#.#.#...#.#...#.#    #...#...#.....#...#.#.#...#.#.#
#.#.#.#########.#.#.###.#####.#    ###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.......#.#.....#.#.#.....#.#    #.#.#.#...#.....#.#.#.#.#.#.#
#.#######.#.#.#####.#.#.###.#.#    #.#.#.#####.###.#######.#.#.#.#
#.#.....#...#.#.#...#.#.#.#...#    #.#.#.....#.#.....#.....#.#.#.#
#.#.###.#####.#.#.#.#.#.#.#####    #.#.#######.#####.###.###.#.#.#
#.#...#.....#.#.#.#.#...#.....#    #.#.....#...#...#.....#.#...#.#
#.###.#####.#.#.#.#.###.#####.#    #.#####.#.###.#.#######.#####.#
#...#.#...#...#...#...#.......#    #.#...#.#.#...#.......#.#.....#
#.#.#.#.##############.#.#######    #.#.#.#.#.#.#########.#.#.#####
#.#.#.#..............#.#.....#    #...#.#.#.#...#...#.#.#.#.....#
#.###.#.#######.#########.###.#    #####.#.#.###.#.#.#.#.#.#####.#
#...#.#.#...#...#.........#...#    #.....#.#.......#.#.#.#...#...#
###.#.###.#.#.###.#####.###.#.#    #.###.#.#########.#.#.#.#.#.###
#...#.....#.#.....#>>B#.#...#.#    #...#.#.....#.#.....#.#.#.#.#.#
#.##########.#######^#.###.###.#    #.#.#######.#.#.#####.#.#.#.#.#
#..v#>>>>v#.#>>>>>>>^#...#.#.#.#    #.#.......#.#...#.....#.#...#.#
#.#v#^###v#.#^##########.#.#.#.#    #.###.#####.#####.#####.#####.#
#.#>>^..#>>>>^#...........#...#    #...#.............#...........#
################################    ################################
```

Listing 4: Ausgabe Beispiel 3

3.5 Beispiel 4

```
Length of shortest path: 84
```

Listing 5: Ausgabe Beispiel 4

3.6 Beispiel 5

```
Length of shortest path: 124
```

Listing 6: Ausgabe Beispiel 5

4 Quellcode

Im Folgenden wird der Quellcode der Datei ”Aufgabe_3.cpp” aufgeführt. Zur näheren Erläuterung des Quellcodes wurden Kommentare hinzugefügt, die mit ”//” und in grüner Schrift gekennzeichnet sind.

```
#include <bits/stdc++.h>

#define ve vector
#define pos pair<pair<int,int>, int> // position with (x,y) and z
#define pospos pair<pos, pos> // pair of two positions
#define graphNode pair<bool, pair<pos, int>> // node in grid with blocked? and (previous,
    ↪ min-distance)
#define everyN(var) for(int var = 0; var < n; var++) // for every row
#define everyM(var) for(int var = 0; var < m; var++) // for every column
#define INF_INT numeric_limits<int>::max()/2 // infinity for int
#define f first // first element of pair
```

```
#define s second // second element of pair

using namespace std; // use std namespace
ifstream inputFile; // input file stream
int n, m, length; // n = rows, m = columns, length = length of shortest path
pos start, stop; // start and stop position of path
ve<ve<graphNode>> g1; // grid 1 (z = 0)
ve<ve<graphNode>> g2; // grid 2 (z = 1)
ve<pos> path; // path from start to stop
ve<pair<pos, int>> allpos = { // all possible moves with (((dx, dy), changeZ?), move-costs)
    make_pair(make_pair(make_pair(0,1), 0), 1),
    make_pair(make_pair(make_pair(1,0), 0), 1),
    make_pair(make_pair(make_pair(0,-1), 0), 1),
    make_pair(make_pair(make_pair(-1,0), 0), 1),
    make_pair(make_pair(make_pair(0,0), 1), 3)
};

/**
 * @brief Get the filename from user
 *
 * @return string (filename)
 */
string getFilename(){
    string filename;
    cout << "Please enter filename without file extension" << endl; // print message to user
    cout << "Files must be located in '/beispieleingaben/'" << endl;
    cout << "-> ";
    cin >> filename; // get input from user
    return filename;
}

/**
 * @brief Read data from input file
 *
 */
void readData(){
    inputFile >> n >> m; // read n and m from input file
    everyN(i){
        g1.push_back(ve<graphNode>()); // add new row to matrix
        g2.push_back(ve<graphNode>()); // add new row to matrix
        everyM(j){
            g1[i].push_back(make_pair(false, make_pair(make_pair(make_pair(-1,-1), -1), INF_INT))); // add new column to first matrix
            g2[i].push_back(make_pair(false, make_pair(make_pair(make_pair(-1,-1), -1), INF_INT))); // add new column to second matrix
        }
    }
    for(int z = 0; z < 2; z++){ // for both grids
        everyN(i){ // for every row
            everyM(j){ // for every column
                char input; // get input from file
                inputFile >> input;
                if(input == '#') // if input is '#', set blocked to true
                    if(z == 0)
                        g1[i][j].f = true;
                    else
                        g2[i][j].f = true;
                else if(input == 'A') // if input is 'A', set start position
                    start = make_pair(make_pair(i,j), z);
                else if(input == 'B') // if input is 'B', set stop position
                    stop = make_pair(make_pair(i,j), z);
            }
        }
    }
    inputFile.close(); // close input file
}

/**
 * @brief perform djikstra algorithm to find shortest path starting from start
 *
 */
```

```
void findPath(){
    long long starttime = clock(); // start timer
    priority_queue<pair<int, pospos>, ve<pair<int, pospos>>, greater<pair<int, pospos>>>
    ↪ q; // priority queue with (distance, (current-position, previous-position))
    q.push(make_pair(0, make_pair(start, make_pair(make_pair(-1,-1), -1)))); // push
    ↪ start position with distance 0 and no previous position
    while(q.size()){ // while queue is not empty
        int d, z, x, y, newX, newY, newZ;
        pos p, pre; // p = current position, pre = previous position
        pospos pp; // pp = pair of p and pre
        tie(d, pp) = q.top(); // get first element of queue
        tie(p, pre) = pp;
        tie(x, y) = p.f; // get current x and y
        z = p.s; // get current z
        q.pop(); // remove first element from queue
        if((z == 0 ? g1 : g2)[x][y].s.s <= d) continue; // if there is already a shorter
    ↪ path to this position, continue with next element
        (z == 0 ? g1 : g2)[x][y].s.s = d; // set distance to current position
        (z == 0 ? g1 : g2)[x][y].s.f.f = pre.f; // set previous position x and y
        (z == 0 ? g1 : g2)[x][y].s.f.s = pre.s; // set previous position z
        for(auto newpos : allpos){ // for every possible move
            newX = newpos.f.f.f + x; // get new x
            newY = newpos.f.f.s + y; // get new y
            newZ = newpos.f.s; // get new z
            if(newZ == 1 && (z == 0 ? g2 : g1)[newX][newY].f) continue; // if z changes
    ↪ and new position is blocked, continue with next element
            if((z == 0 ? g1 : g2)[newX][newY].f) continue; // if new position is blocked,
    ↪  continue with next element
            q.push(make_pair(d + newpos.s, make_pair(make_pair(make_pair(newX, newY), (
    ↪ newZ == 1 ? (z == 0 ? 1 : 0) : z)), p))); // push (distance, (pos, prev)) to queue
        }
    }
    long long endtime = clock(); // end timer
    cout << "Calculated path in: " << (endtime - starttime) << " ms" << endl; // print
    ↪ time needed to calculate path
    length = (stop.s == 0 ? g1 : g2)[stop.f.f][stop.f.s].s.s; // get length of shortest
    ↪ path
    cout << "Length of shortest path: " << length << endl; // print length of shortest
    ↪ path
}

/**
 * @brief Print out calculated path to user
 *
 * @param path Calculated path
 */
void printPath(ve<pos> path){
    cout << "print path? (y/n)" << endl; // ask user if path should be printed
    char ans;
    cin >> ans;
    if(ans != 'y') return; // if user does not want to print path, return
    cout << "Path:" << endl << "x y z" << endl; // print header
    for(pos p : path) // for every position in path
        cout << p.f.f << " " << p.f.s << " " << p.s << endl; // print position
}

/**
 * @brief Generate content of output file and write to output file
 *
 * @param path Calculated path
 * @param filename filename of input file without file extension
 * @param length length of calculated path
 */
void generateFileOutput(ve<pos> path, string filename, int length){
    ofstream outputFile; // create output file stream
    outputFile.open("../beispielausgaben/"+filename+"_out.txt"); // open output file
    char output1[n][m], output2[n][m]; // create output matrix 1 and 2
    for(int z = 0; z < 2; z++){ // for both grids
        everyN(i){ // for every row
            everyM(j){ // for every column
                if((z == 0 ? g1 : g2)[i][j].f) // if position is blocked, set output to
    ↪ '#'
                    (z == 0 ? output1 : output2)[i][j] = '#';
```

```
143                 else // else set output to '.'
                        (z == 0 ? output1 : output2)[i][j] = '.';
145             }
            }
147     }
        (start.s == 0 ? output1 : output2)[start.f.f][start.f.s] = 'A'; // set start position
149     for(int i = 0; i < path.size()-1; i++){ // for every position in path
            int x = path[i].f.f;
151         int y = path[i].f.s;
            int z = path[i].s;
153         int nextX = path[i+1].f.f;
            int nextY = path[i+1].f.s;
155         if(nextX - x == 1) // check if direction is down
                (z == 0 ? output1 : output2)[x][y]='v';
157         else if(nextX - x == -1) // check if direction is up
                (z == 0 ? output1 : output2)[x][y]='^';
159         else if(nextY - y == 1) // check if direction is right
                (z == 0 ? output1 : output2)[x][y]='>';
161         else if(nextY - y == -1) // check if direction is left
                (z == 0 ? output1 : output2)[x][y]='<';
163         else { // else set output to '!'
                output1[x][y]='!';
165             output2[x][y]='!';
            }
167     }
        (stop.s == 0 ? output1 : output2)[stop.f.f][stop.f.s] = 'B'; // set stop position
169     outputFile << "Length␣of␣shortest␣path:␣" << length << endl; // write length of
        ↪ shortest path to output file
        everyN(i){ // for every row
171         everyM(j) // for every column
                outputFile << output1[i][j]; // write output maxtrix 1 to output file
173         outputFile << "␣␣␣";
            everyM(j) // for every column
175             outputFile << output2[i][j]; // write output maxtrix 2 to output file
            outputFile << endl;
177     }
        outputFile.close(); // close output file
179 }

181 /**
     * @brief Build path from start to stop after djikstra algorithm
183  *
     */
185 void buildPath(){
        pos p = stop; // start at stop position
187     path = {stop}; // add stop position to path
        while(p != start) // while current position is not start position
189         path.push_back(p = (p.s == 0 ? g1 : g2)[p.f.f][p.f.s].s.f); // add previous
        ↪ position to path
        reverse(path.begin(), path.end()); // reverse path, so it starts at start position
191 }

193 /**
     * @brief Main Function of program
195  *
     * @return int (exit code)
197  */
    int main(){
199     string filename = getFilename(); // get filename for in-/output
        inputFile.open("../beispieleingaben/"+filename+".txt"); // open input file
201     if(inputFile.is_open()) readData(); // if input file is open, read data from input
        ↪ file
        else {
203         cout << "File␣not␣found" << endl; // if input file is not open, print error
        ↪ message
            return 1;
205     }
        findPath(); // find shortest path
207     buildPath(); // build path from start to stop
        printPath(path); // print path to user
209     generateFileOutput(path, filename, length); // generate output file
        return 0;
211 }
```