Aufgabe 3: Zauberschule

Team-ID: 00988

Team-Name: BitShifter

Bearbeiter dieser Aufgabe:
Filip Zelinskyi

21. November 2023

Inhaltsverzeichnis

1 Abstract

Die Lösung basiert auf einem Algorithmus A*, der für die Suche nach dem kürzesten Weg in einem dreidimensionalen Labyrinth angepasst wurde.

In diesem Dokument wird der Lösungsansatz beschrieben, die Umsetzung auf C++ erläutert und die Lösungen für die Beispiele vorgestellt.

2 Lösungsidee

Das Problem lässt sich zur Vereinfachung zunächst auf die bekannten Algorithmen für das Finden des kürzesten Weges in einem zweidimensionalen Labyrinth reduzieren.

Ein Labyrinth kann als gewichteter Graph dargestellt werden, wobei jedes freie Feld einem Knoten entspricht und jede Verbindung zwischen benachbarten Feldern (außer Wänden) einer Kante entspricht. In einem zweidimensionalen Labyrinth beträgt das Gewicht jeder Kante 1.

Um einen Weg durch das Labyrinth zu finden, können wir verschiedene Algorithmen verwenden. Der Breadth-First Search (BFS) Algorithmus ist eine Möglichkeit, aber er ist nicht sehr effizient für die gezielte Pfadsuche und berücksichtigt keine Gewichte.

Eine bessere Option könnte der Dijkstra-Algorithmus sein, der die Kosten für jedes Feld berechnet und Wege mit geringeren Kosten bevorzugt. Allerdings breitet sich dieser Algorithmus in alle Richtungen aus und verschwendet Zeit für unwahrscheinliche Lösungen.

Wenn wir nur einen Zielpunkt haben, können wir den A*-Algorithmus verwenden, der eine heuristische, gezielte Suche durchführt und effizienter ist als der Dijkstra-Algorithmus.

Wenn wir das Problem auf ein dreidimensionales Labyrinth erweitern wollen, können wir eine zusätzliche Koordinate z einführen, die die Ebene des Knotens darstellt. Das Gewicht der Kante zwischen einem Punkt (x, y, z) und einem anderen Punkt (x, y, z+1) oder (x, y, z-1) wird als 3 Sekunden festgelegt.

3 Umsetzung

Zunächst möchte ich einige neue Code-Strukturen definieren, die oft bei der Implementierung verwendet werden. Dies umfasst die Struktur eines 3D-Punktes, die Darstellung eines Punkteintrags in der Warteschlange mit einer zugehörigen Priorität sowie die Repräsentation des Labyrinths in Form einer dreidimensionalen Matrix. Gemäß den Anforderungen der Aufgabenstellung bleibt die Tiefe ($DEPTH$) stets konstant bei 2.

Require: DEPTH: 2, ROW: int, COL: int ▷*Größe des Feldes aus der Eingabe*
type Point $\leftarrow (x : int, y : int, z : int)$
type PriorityPoint $\leftarrow (point : Point, priority : int)$
type LabyrinthMatrix $\leftarrow array_{DEPTH,ROW,COL}$

Der Suchalgorithmus benötigt fortlaufend Informationen über die möglichen Auswege zu naheliegenden Feldern im Labyrinth, beziehungsweise die ausgehenden Verbindungen zu anderen Knoten. Hierzu wird für alle sechs möglichen Fälle zu den aktuellen Koordinaten der entsprechende Offset hinzugefügt, um die Koordinaten des benachbarten Punktes zu erhalten. Anschließend wird der benachbarte Punkt daraufhin überprüft, ob er sich innerhalb der Grenzen der Karte befindet und ob es sich nicht um eine Wand handelt.

Algorithm 1 Finde alle Nachbarn von dem Punkt im Matrix

```
offsetX ← {0, 0, −1, 1, 0, 0}                ▷Sechs mögliche Fälle von Nachbarn
offsetY ← {−1, 1, 0, 0, 0, 0}
offsetZ ← {0, 0, 0, 0, 1, −1}
function IsValid(x, y, z)
    return (y ≥ 0) and y < ROW and (x ≥ 0) and x < COL and (z ≥ 0) and (z < DEPTH)
end function
function GetNeighbors(mat : LabyrinthMatrix, point : Point)
    result ← array(Point)
    for alle sechs mögliche Seiten do
        x ← point.x + offsetX[i]
        y ← point.y + offsetY[i]
        z ← point.z + offsetZ[i]
        if IsValid(x, y, z) and mat[z][y][x] ≠ # then   ▷Innerhalb Grenzen der Karte und nicht eine Wand
            result.push({x, y, z}
        end if
    end for
    return result
end function
```

Die Funktion *Heuristik* gibt an, wie nahe der Punkt A dem Punkt B ist. Diese Information ist entscheidend für die gezielte Suche nach dem Zielpunkt (Finish-Knoten).

Algorithm 2 Berechne die Heuristik für zwei Knoten

```
function Heuristic(a : Point, b : Point)
    return abs(a.x − b.x) + abs(a.y − b.y) + abs(a.z − b.z)
end function
```

Die Funktion *GetCost* ermittelt das Gewicht der Kante zwischen Punkt A und Punkt B. Wenn sich die Punkte auf derselben Ebene befinden, beträgt das Gewicht **eine Sekunde**, andernfalls **drei Sekunden**.

Der Hauptalgorithmus zur Pfadsuche ist eine modifizierte Implementierung des A*-Algorithmus, der auf 3D erweitert wurde und für die vereinfachte Darstellung des Labyrinths in Form einer Matrix angepasst ist. Dabei werden die noch zu prüfenden Knoten in einer priorisierten Schlange gespeichert. Dies gewährleistet, dass zunächst die wahrscheinlichsten Knoten in Richtung des Ziel-Knotens überprüft werden, wodurch

Algorithm 3 Berechne die Kosten für den Übergang vom Punkt A zum Punkt B

```
function GetCost(a : Point, b : Point)
    if a.z = b.z then
        return 1
    else
        return 3
    end if
end function
```

eine schnellere Lösung erreicht werden kann. Die bereits besuchten Knoten und die Kosten für jeden überprüften Knoten werden in einer Hashmap gespeichert.

Der Algorithmus arbeitet, bis er vom Startpunkt A zum Zielpunkt B gelangt. Alle Nachbarn des ersten Elements in der Schlange werden überprüft. Falls sie noch nicht geprüft wurden, werden sie mit einer entsprechenden Priorität bezüglich der Distanz zum Zielpunkt in die Schlange eingefügt.

Nachdem die Schleife den Zielpunkt erreicht hat, wird die beste Zeit aus der Hashmap **costSoFar** ausgelesen, und die Reihenfolge der Koordinaten für die Lösung wird in einem Array gespeichert. Dieses Array dient dann als Grundlage für die Erstellung der Ausgabe.

Algorithm 4 Suche den Pfad

```
Require: mat: LabyrinthMatrix, start: Point, finish: Point
    frontier ← PriorityQueue(PriorityPoint)      ▷Gebiet von Knoten, die überprüft werden müssen
    cameFrom ← HashMap(Point, Point)                                  ▷Zurückgelegte Knoten
    costSoFar ← HashMap(Point, int)                                   ▷Kosten für jeden Knoten
    frontier.Enqueue(new PriorityPoint(start, 0))                ▷Festlegen von Standardwerten für den
    Start-Knoten
 5: cameFrom[start] ← {}
    costSoFar[start] ← 0
    while frontier nicht leer do                                  ▷Im schlimmsten Falle alle Knoten durchgehen
        currentPoint ← frontier.Dequeue().point                   ▷Der wahrscheinlichste nächste Knoten
        if current = finish then                                  ▷Frühzeitiger Stopp, wenn Finish gefunden wurde
10:         break
        end if
        for next of GetNeighbors(mat, current) do                                ▷Überprüfe alle Nachbarn
            newCost ← costSoFar[current] + GetCost(current, next)
            if next nicht besucht  or newCost < costSoFar[next] then
15:             costSoFar[next] ← newCost
                frontier.Enqueue(new PriorityPoint(next, (newCost + Heuristic(finish, next))))
                cameFrom[next] ← current
            end if
        end for
20: end while
    bestCost ← costSoFar[finish]
    solution ← GetSolutionPath(cameFrom, start, finish)    ▷Liste von Punkten, die vom Start zum
    Finish führen
```

4 Beispiele

Für die Art und Weise der Ausgabe des gefundenen Pfades wurde das Beispiel von der BwInf-Webseite übernommen. Die Pfade für die Beispiele 4 und 5 wurden im Ordner "results"gespeichert.

Zauberschule 0

```
cat examples/zauberschule0.txt | ./a.out
Beste Zeit: 8
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
##.#.#.###.#.#
#...#.....#.#
#.#########.#
#.....#.....#
####.#.###.#
#....!#B..#.#
#.#########.#
#...........#
#############
#############
#.......#...#
#...#.#.#...#
#...#.#.....#
#.###.#.#.###
#.....#.#...#
#####.###...#
#.....#.....#
#.#########.#
#...#>>!....#
#.#.#.#.###.#
#.#...#...#.#
#############
```

Zauberschule 1

```
cat examples/zauberschule1.txt | ./a.out
Beste Zeit: 4
#####################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
##.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
#.#...#.#.#^<<#.....#
#.#####.#.#########.#
#...................#
#####################
#####################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#...........#.......#
#####################
```

Zauberschule 2

```
cat examples/zauberschule2.txt | ./a.out
Beste Zeit: 14
###############################################
#...#.....#..........#.#..........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#>>!#v#.....#.#.#...#...#
##.###.#.#.#############v#.#.###.#.###.#.###
#.#.#...#.#.............>>B#.#...#.#...#.#.#
#.#.#.###.###########.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
###############################################
###############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.########.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
###############################################
```

Zauberschule 3

```
cat examples/zauberschule3.txt | ./a.out
Beste Zeit: 28
################################
#...#.....#..........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
##.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.############.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.#######.###.#.#####
#...#.#...#...#.#...#...#.#...#
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.##############.#.#######
#.#.#.#..............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
##.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#>>B#.#...#.#
#.#########.#######^#.###.###.#
#..v#>>>>v#.#>>>>>>^#...#.#.#.#
#.#v#^###v#.#^#########.#.#.#.#
#.#>>^..#>>>>^#...........#...#
################################
################################
#.......#.....#...#...#.......#
#.###.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.#.....#.#.#.....#
######.#.#.#.#.#####.#.#####.#
#.....#.#.#.#.#.#...#...#.....#
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.#########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
##.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#.#.#.#.#.#
#.#.########.#.#.#####.#.#.#.#.#
#.#.......#.#...#.....#.#...#.#
#.###.#####.#####.#####.#####.#
#...#............#..........#
################################
```

Zauberschule 4

```
cat examples/zauberschule4.txt | ./a.out
Beste Zeit: 84
...
```

Zauberschule 5

```
cat examples/zauberschule5.txt | ./a.out
Beste Zeit: 124
...
```

5 Quellcode

In diesem Kontext wird die C++20-Version verwendet, um das Verhalten von Vergleichsoperatoren für eigene Strukturen zu definieren.

```
int ROW, COL, DEPTH = 2;
const int yNum[] = {-1, 1, 0, 0, 0, 0};
const int xNum[] = {0, 0, -1, 1, 0, 0};
const int zNum[] = {0, 0, 0, 0, 1, -1};

struct Point {
  int x;
  int y;
  int z;

  bool operator==(const Point& other) const {
    return x == other.x && y == other.y && z == other.z;
  }
};

struct PriorityPoint {
  Point point;
  int priority;

  bool operator<(const PriorityPoint& other) const {
    return priority > other.priority;
  }
};

bool is_valid(int x, int y, int z) {
  return (y >= 0) && (y < ROW) && (x >= 0) && (x < COL) && (z >= 0) &&
         (z < DEPTH);
}

vector<Point> get_neighbors(vector<vector<vector<char>>> mat, Point pt) {
  vector<Point> result;
  for (int i = 0; i < 6; i++) {
    int y = pt.y + yNum[i];
    int x = pt.x + xNum[i];
    int z = pt.z + zNum[i];
    if (!is_valid(x, y, z) || mat[z][y][x] == '#') continue;
    result.push_back({x, y, z});
  }
  return result;
}

void print_maze(vector<vector<vector<char>>> mat) {
  for (int z = 0; z < DEPTH; z++) {
    for (int y = 0; y < mat[z].size(); y++) {
      for (int x = 0; x < mat[z][y].size(); x++) {
        cout << mat[z][y][x];
      }
      cout << "\n";
    }
```

```
  }
}

int get_cost(Point a, Point b) { return a.z == b.z ? 1 : 3; }

int heuristic(Point a, Point b) {
  return abs(a.x - b.x) + abs(a.y - b.y) + abs(a.z - b.z);
}

vector<Point> get_solution_path(unordered_map<Point, Point> came_from,
                                Point start, Point finish) {
  vector<Point> path;
  Point current = finish;
  while (current != start) {
    path.push_back(current);
    current = came_from[current];
  }
  path.push_back(start);
  return path;
}

pair<int, vector<Point>> find_path(vector<vector<vector<char>>> mat,
                                   Point start, Point finish) {
  priority_queue<PriorityPoint> frontier;
  frontier.push({start, 0});

  unordered_map<Point, Point> came_from;
  came_from[start] = {};

  unordered_map<Point, int> cost_so_far;
  cost_so_far[start] = 0;

  while (!frontier.empty()) {
    Point current = frontier.top().point;
    frontier.pop();
    if (current == finish) break;
    for (Point next : get_neighbors(mat, current)) {
      int new_cost = cost_so_far[current] + get_cost(current, next);
      if (came_from.count(next) == 0 || new_cost < cost_so_far[next]) {
        cost_so_far[next] = new_cost;
        frontier.push({next, new_cost + heuristic(finish, next)});
        came_from[next] = current;
      }
    }
  }

  int best_cost = cost_so_far[finish];
  vector<Point> solution_path = get_solution_path(came_from, start, finish);

  return {best_cost, solution_path};
}
```