Aufgabe 5: Das ägyptische Grabmal

Team-ID: **00078**

Team: Timon

Bearbeiter dieser Aufgabe:
Timon Retzlaff

Inhaltsverzeichnis

Lösungsidee

Das Programm verfolgt den Ansatz einer Breitensuche, bei der mögliche Zustände durch Bewegungen und Wartezeiten generiert werden. Die erreichbaren Zustände werden dabei nach der vergangenen Zeit sortiert. Dadurch ist der erste Zustand, welcher das Ziel erreicht, der der durch den schnellsten Pfad resultiert. Sobald das Ziel erreicht wird, rekonstruiert das Programm den besten Weg, indem es auf die gespeicherten Verweise der besuchten Zustände zurückgreift.

Umsetzung

Einlesen der Datei

Die Implementierung beginnt damit, dass die Eingabedaten aus einer Datei gelesen und in dem Array *gates* gespeichert werden.

FindPath-Methode

In der *findPath*-Methode wird die Breitensuche durchgeführt. In einer Liste (*toCheck*) werden alle Zustände gespeichert, die noch überprüft werden müssen. Und der Startzustand wird hinzugefügt.

State-Klasse

Die Zustände werden durch die Klasse *State* repräsentiert, in der die aktuelle Zeit und Position gespeichert wird.

Generierung von Folgezuständen

Die Methode *getAllChildStates* generiert alle möglichen Folgezustände für den vordersten Eintrag in *toCheck*. Sie prüft, ob das aktuelle Tor zur gegebenen Zeit offen ist. Andernfalls wird eine leere Liste zurückgegeben, da keine Bewegung möglich ist. Abhängig von der Position im Pfad wird ermittelt, ob sich der Zustand durch Warten oder Bewegung in benachbarte Abschnitte ändern lässt. Die Methode erzeugt Zustände für:

- Das Warten an der aktuellen Position, bis das Tor sich öffnet.
- Das Vorwärts- oder Rückwärtsbewegen, wenn das jeweilige Tor geöffnet ist.
- Zusätzliche Wartebewegungen, die erforderlich sein könnten, um auf das Öffnen eines benachbarten Tores zu warten.

FindPath-Methode

Dann wird jeder generierte Zustand an der passenden Stelle in *toCheck* hinzugefügt. Außerdem wird für jeden Zustand der vorherige Zustand in einer *HashMap* gespeichert. Dann wird das ganze wiederholt. Sobald ein Zustand beim Ziel ist wird die Methode *getPath* aufgerufen.

GetPath

Die Methode *getPath* verfolgt den aktuellen Zustand über die *HashMap* bis zum Anfang zurück und gibt den Pfad als *State*-Liste zurück.

Beispiele

Beispiel 1: Ein Tor

In einem Fall mit nur einem Tor prüft das Programm, ob das Tor zum gegebenen Zeitpunkt offen ist. Wenn es nicht offen ist, wird die Wartezeit berechnet, bis das Tor sich öffnet. Dann kann man zum Ziel.

Eingabe:

- 1
- 2

Ausgabe:

Warte für 2 Minuten; Laufe zum Grabmal

Found in 0.4242ms

Beispiel 2: Mehrere Tore

Hier ein Beispiel mit mehreren Toren.

Eingabe:

- 3
- 7
- 3
- 2

Ausgabe:

Warte für 7 Minuten; Laufe in den Abschnitt 1; Warte für 2 Minuten; Laufe in den Abschnitt 2; Warte für 1 Minute; Laufe zum Grabmal

Found in 0.7347ms

Beispiel 3: Viele Tore

Eingabe:

- <siehe gegebene Beispiel aus grabmal5.txt>

Ausgabe:

Warte für 43838148 Minuten; Laufe in den Abschnitt 1; Warte für 45 Minuten; Laufe in den Abschnitt 5; Warte für 4542 Minuten; Laufe in den Abschnitt 6; Warte für 6752 Minuten; Laufe in den Abschnitt 7; Warte für 1452 Minuten; Laufe in den Abschnitt 8; Warte für 3576 Minuten; Laufe in den Abschnitt 9; Warte für 281 Minuten; Laufe in den Abschnitt 14; Warte für 5969 Minuten; Laufe in den Abschnitt 17; Warte für 2246 Minuten; Laufe in den Abschnitt 18; Warte für 2629 Minuten; Laufe in den Abschnitt 19; Warte für 1120 Minuten; Laufe in den Abschnitt 21; Warte für 2408 Minuten; Laufe in den Abschnitt 22; Warte für 5247 Minuten; Laufe in den Abschnitt 23; Warte für 1620 Minuten; Laufe in den Abschnitt 24; Warte für 825 Minuten; Laufe in den Abschnitt 25; Warte für 3438 Minuten; Laufe in den Abschnitt 27; Warte für 1034 Minuten; Laufe in den Abschnitt 28; Warte für 349 Minuten; Laufe in den Abschnitt 32; Warte für 1694 Minuten; Laufe in den Abschnitt 34; Warte für 2894 Minuten; Laufe in den Abschnitt 35; Warte für 4867 Minuten; Laufe in den Abschnitt 36; Warte für 70 Minuten; Laufe in den Abschnitt 39; Warte für 1706 Minuten; Laufe in den Abschnitt 43; Warte für 204 Minuten; Laufe in den Abschnitt 45; Warte für 859 Minuten; Laufe in den Abschnitt 46; Warte für 195 Minuten; Laufe in den Abschnitt 47; Warte für 2040 Minuten; Laufe in den Abschnitt 48; Warte für 1751 Minuten; Laufe in den Abschnitt 49; Warte für 4079 Minuten; Laufe in den Abschnitt 50; Warte für 1257 Minuten; Laufe in den Abschnitt 51; Warte für 731 Minuten; Laufe in den Abschnitt 52; Warte für 1001 Minuten; Laufe in den Abschnitt 53; Warte für 4342 Minuten; Laufe in den Abschnitt 54; Warte für 1332 Minuten; Laufe in den Abschnitt 55; Warte für 4533 Minuten; Laufe in den Abschnitt 56; Warte für 3474 Minuten; Laufe in den Abschnitt 57; Warte für 113 Minuten; Laufe in den Abschnitt 59; Warte für 2935 Minuten; Laufe in den Abschnitt 61; Warte für 1112 Minuten; Laufe in den Abschnitt 62; Warte für 289 Minuten; Laufe in den Abschnitt 63; Warte für 3931 Minuten; Laufe in den Abschnitt 64; Warte für 704 Minuten; Laufe in den Abschnitt 68; Warte für 481 Minuten; Laufe in den Abschnitt 69; Warte für 4319 Minuten; Laufe in den Abschnitt 71; Warte für 5643 Minuten; Laufe in den Abschnitt 72; Warte für 3633 Minuten; Laufe in den Abschnitt 73; Warte für 413 Minuten; Laufe in den Abschnitt 74; Warte für 4375 Minuten; Laufe in den Abschnitt 73; Warte für 617 Minuten; Laufe in den Abschnitt 72; Warte für 1906 Minuten; Laufe in den Abschnitt 71; Warte für 7132 Minuten; Laufe in den Abschnitt 72; Warte für 3113 Minuten; Laufe in den Abschnitt 73; Warte für 651 Minuten; Laufe in den Abschnitt 74; Warte für 792 Minuten; Laufe in den Abschnitt 80; Warte für 2414 Minuten; Laufe in den Abschnitt 81; Warte für 1222 Minuten; Laufe in den Abschnitt 82; Warte für 1485 Minuten; Laufe zum Grabmal

Found in 138.2041ms

Beispiel 4

Eingabe:

- <siehe gegebene Beispiel aus grabmal4.txt>

Ausgabe:

Warte für 3242835 Minuten; Laufe in den Abschnitt 1; Warte für 60360 Minuten; Laufe in den Abschnitt 2; Warte für 360 Minuten; Laufe in den Abschnitt 5; Warte für 37695 Minuten; Laufe in

den Abschnitt 6; Warte für 51407 Minuten; Laufe in den Abschnitt 8; Warte für 36216 Minuten; Laufe in den Abschnitt 9; Warte für 4024 Minuten; Laufe zum Grabmal

Found in 23.2225ms

Beispiel 5

Eingabe:

- <siehe gegebene Beispiel aus grabmal3.txt>

Ausgabe:

Warte für 20 Minuten; Laufe in den Abschnitt 1; Warte für 2 Minuten; Laufe in den Abschnitt 6; Warte für 13 Minuten; Laufe zum Grabmal

Found in 0.9607ms

Beispiel 6

Eingabe:

- <siehe gegebene Beispiel aus grabmal2.txt>

Ausgabe:

Warte für 170000 Minuten; Laufe in den Abschnitt 2; Warte für 40009 Minuten; Laufe in den Abschnitt 3; Warte für 59991 Minuten; Laufe in den Abschnitt 4; Warte für 80015 Minuten; Laufe in den Abschnitt 3; Warte für 39985 Minuten; Laufe in den Abschnitt 2; Warte für 100021 Minuten; Laufe zum Grabmal

Found in 0.6085ms

Beispiel 7

Eingabe:

- <siehe gegebene Beispiel aus grabmal1.txt>

Ausgabe:

Warte für 17 Minuten; Laufe in den Abschnitt 2; Warte für 4 Minuten; Laufe in den Abschnitt 3; Warte für 6 Minuten; Laufe in den Abschnitt 4; Warte für 8 Minuten; Laufe in den Abschnitt 3; Warte für 4 Minuten; Laufe in den Abschnitt 2; Warte für 10 Minuten; Laufe zum Grabmal

Found in 0.6196ms

Beispiel 8

Eingabe:

- <siehe gegebene Beispiel aus grabmal0.txt>

Ausgabe:

Warte für 5 Minuten; Laufe in den Abschnitt 1; Warte für 3 Minuten; Laufe in den Abschnitt 2; Warte für 4 Minuten; Laufe zum Grabmal

Found in 0.4614ms

Quellcode

```
private static List<State> findPath(final Gate[] gates) throws Exception {
    Map<State, State> previous = new HashMap<>();
    List<State> toCheck = new ArrayList<>();
    State state = new State(0, -1);
    toCheck.add(state);
    do {
        state = toCheck.get(0);
        toCheck.remove(0);
        List<State> allChildStates = getAllChildStates(state, gates);
        for (State childState : allChildStates) {
            final int stateTime = state.time();
            final boolean isWaitingMove = stateTime != childState.time();
            if (isWaitingMove && childState.position() >= 0) {
                if ((childState.time() - stateTime)
                  >= getTimeUntil(gates[state.position()], false, stateTime)) {
                    // I died here!
                    continue;
                }
            }
            if (previous.containsKey(childState)) {
                final State state1 = previous.get(childState);
                if (stateTime < state1.time()) {
                    previous.put(childState, state);
                }
            } else {
                previous.put(childState, state);
                addState(toCheck, childState);
            }
            if (childState.position() == gates.length - 1) {
                return getPath(childState, previous);
            }
        }
    } while (state.position() < gates.length);
    throw new Exception("Couldn't find a path");
}


private static List<State> getAllChildStates(final State state, final Gate[]
gates) {
    final int stateTime = state.time();

    final int position = state.position();
    if (position > -1 && !gates[position].isOpen(stateTime)) {
```

```
        return Collections.emptyList();
    }
    List<State> result = new ArrayList<>();
    if (position == -1) {
        final Gate gate = gates[0];
        if (!gate.isOpen(stateTime)) {
            return Collections.singletonList(new State(stateTime +
getTimeUntil(gate, true, stateTime), position));
        } else {
            final int dt1 = getTimeUntil(gate, false, stateTime);
            final int time = stateTime + dt1;
            result.add(new State(time + getTimeUntil(gate, true, time),
position));
        }
    } else {
        addWaitingMovesFor(
            gates[position],
            gates[position + 1],
            stateTime,
            position,
            result
        );
        if (position > 0) {
            addWaitingMovesFor(
                  gates[position],
                  gates[position - 1],
                  stateTime,
                  position,
                  result
            );
        }
    }

    if (gates[position + 1].isOpen(stateTime)) {
        result.add(new State(stateTime, position + 1));
    }
    if (position > 0) {
        if (gates[position - 1].isOpen(stateTime)) {
            result.add(new State(stateTime, position - 1));
        }
    }
    return result;
}
```