Assignment 4

COS 212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Deadline: 02/07/2021 at 18:00

General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of 0. Only the output of your program will be considered for marks, but your code may be inspected.
- **All submissions will be checked for plagiarism.**
- Read the entire assignment before you start coding.
- You will be afforded three upload opportunities per task.

Plagiarism:

The Department of Computer Science regards plagiarism as a serious offence. Your code will be subject to plagiarism checks and appropriate action will be taken against offending parties. You may refer to the the Library's website at `www.library.up.ac.za/plagiarism/index.htm` for more information.

After completing this assignment:

Upon successful completion of this assignment, you will have implemented a graph data structure to represent a two-dimensional (2D) dungeon. You will also have implemented an algorithm to find the shortest path to unlock and retrieve the treasure from the dungeon.

Your task

Your task will be to find the shortest path through a 2D dungeon to unlock and retrieve a treasure. This will involve creating a graph data structure from the text representation of a dungeon. Once the dungeon is converted to a graph, graph algorithms can be applied and the problem becomes more familiar.

This assignment is divided into a number of tasks which build on each other, and should therefore be completed in the given order.

Task 1: Creating a Graph from the Dungeon File

To enable the search for the shortest path, a graph structure must be created from the given text representation of a 2D dungeon.

Dungeon Representation

The dungeon will be read from a text file. Each file will contain a single dungeon only. The basic dungeon is represented by the following character tile types:

- `E` – **Entrance**. There will only be one entrance to a dungeon.
- `#` – **Wall**. It is not possible to "walk" through a wall. These tiles should not be included in the graph.
- `.` (period) – **Empty tile**. These tiles can be walked over and may form part of a path. When creating the graph, consider tiles that are up, down, left and right of an empty tile to create the necessary edges.
- `K` – **Key**. The key unlocks the treasure, and must be collected (i.e. visited) before the treasure can be retrieved. The key can be walked over, just like an empty tile.
- `T` – **Treasure**. The treasure can be collected by visiting this tile after the key has been retrieved. The treasure can be walked over, just like an empty tile.

Consider the following example dungeon, further referred to as **Dungeon 1** in this document:

```
######
E....#
#.##T#
#.K..#
######
```

The dungeon can be entered and exited via the 'E' tile. To get the treasure, you must navigate to the 'K' tile first, and then make your way to the 'T' tile. Once you have visited both 'K' and 'T', you can exit the dungeon by returning to the 'E' tile. Navigating the dungeon can be done in 4 directions: left, right, up, and down. It is not possible to navigate diagonally.

The supplied dungeon file may contain white space before or after the dungeon. You can assume the dungeon map to always be rectangular.

Graph Representation

To convert a dungeon to an undirected graph, you must read the dungeon file, and create a separate vertex for each walkable tile (E, K, T, .). An edge between any two vertices will be used to indicate that the two tiles are adjacent, and will have the default weight of 1.

Vertices in the graph will be uniquely identified by their coordinates in the dungeon, assuming that the top left tile has coordinates (0, 0). You are provided with skeletal `Coordinates` and `Vertex` classes which you may extend as necessary.

Consider the example Dungeon 1. The tile at position (1, 0) will be the `'E'` tile. From this tile you can go to (1, 1). From (1, 1), you can either go right to (1, 2), or go down to (2, 1). Figure 1 illustrates the graph created from the example dungeon. As you can see, wall tiles do not make a part of the graph. Entrance is shown in green, key in red, and treasure in yellow for illustration purpose.

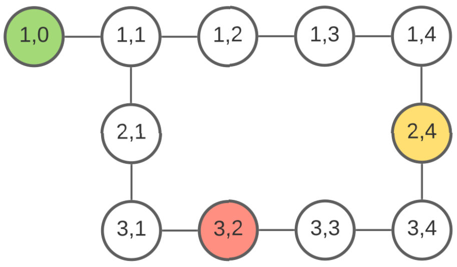

Figure 1: Visual representation of the graph created from the example Dungeon 1.

The specific graph implementation (i.e. adjacency matrix or adjacency list) of the graph data structure is left up to you. Use the `DungeonGraph` class to implement your solution.

Functions to Implement for Task 1

You are required to implement the following functions in `DungeonGraph.java`:

- `void createGraph(String filename)` – Use the information provided above to create a graph from the given dungeon file.
- `Vertex getVertex(Integer row, Integer col)` – Return the vertex with the given coordinates (row, col). If the vertex does not exist, return null. If the coordinates are out of bounds, return null.
- `String toString()` – Return a string representing a depth-first traversal of the graph. The traversal must start from the Entrance vertex. For each tile, visit the adjacent vertices in the following order: left, up, right, down. For each vertex, output its coordinates. For the graph in Figure 1, the following string must be returned:
  `(1,0),(1,1),(1,2),(1,3),(1,4),(2,4),(3,4),(3,3),(3,2),(3,1),(2,1)`
- `Vertex[] getAdjacentVertices(Vertex vertex)` – Return the vertices adjacent to the given vertex. The vertices in the returned array must be sorted in the following order: left, top, right, bottom. Return an empty array if there are no adjacent vertices.
- `Vertex getDoor()` – return the vertex corresponding to the dungeon entrance.
- `Vertex getKey()` – return the vertex corresponding to the key tile.
- `Vertex getTreasure()` – return the vertex corresponding to the treasure tile.

Task 2: Finding the Shortest Path to the Treasure

For this task, you must implement an algorithm to find the shortest path from the entrance of the dungeon to the key, from the key to the treasure, and from the treasure back to the entrance. Assume that each dungeon has only one entrance, key, and treasure, and that every edge has a weight of 1.

If multiple paths of the same length (i.e. total weight) exist, assume that the following directional preference applies: left, up, right, down. I.e., if two paths starting from a tile have the same length, then the path that begins with a left step should be preferred to a path that begins with an up step, up should be preferred to right, and right should be preferred to down. HINT: since each node in the graph is identified by the coordinates, a comparison between the current node's coordinates and the coordinates of the adjacent nodes will help you to easily determine the relative location of the adjacent nodes (left, right, up, down) without storing any additional information.

Consider the following example dungeon (Dungeon 2):

```
######
E....#
#...T#
#.K..#
######
```

The shortest path from 'E' to 'K' that should be reported is "right, right, down, down", i.e. (1,0),(1,1),(1,2),(2,2),(3,2).

Functions to Implement for Task 2

You are required to implement the following functions in `DungeonGraph.java`:

- `Vertex[] getShortestPath(Coordinates start, Coordinates end)` – Return the vertices along the shortest path from the start vertex to the end vertex, as identified by the given coordinates. The start and the end vertex must be included. If no path exists, return an empty array.
- `Vertex[] getShortestPath()` – Return an array of vertices that make up the shortest path from the entrance to the key to the treasure and back to the entrance, in order from start to end. The starting and the ending vertex (entrance) should be included in the path. If there is no path, return an empty array.
- `Integer getShortestPathLength(Coordinates start, Coordinates end)` – Return the length of the shortest path from the given starting vertex coordinates, to the end vertex coordinates. The start and end vertices should be part of the path. The length can be calculated by summing the weights of edges in the path. For Task 2, each edge has an implicit weight of 1, so the path length is the number of edges in the path. If no path exists, return null.
- `String getShortestPathString(Coordinates start, Coordinates end)` – Return the string representing the shortest path from start vertex to end vertex by indicating the succession of steps (left, right, up, down) that need to be taken. Eg., refer to the example dungeon on Page 2. The shortest path from 'E' to 'K' can be represented as `right, down, down, right.` The words must be comma-separated, with a space after each comma, and a full stop at the end. Left-up-right-down movement preference applies.
- `String getShortestPathString()` – This method has the same functionality as the one above, but should return the text representation of the shortest path from entrance to key to treasure and back to the entrance.

Task 3: Traps and Teleports

For this task, you will expand the dungeon implementation to allow for two new tile types: traps and teleports.

Traps

Traps are walkable tiles indicated by the 'X' character. To cross the trap unharmed, extra care must be taken. Thus, crossing the trap tile takes twice as much time as crossing any other walkable tile. Therefore, all edges incident with a trap tile will have a weight of 2 rather than 1. A dungeon may contain any number of trap tiles.

Teleports

Teleports, as opposed to the traps, potentially make the path through the dungeon shorter. Teleports are indicated using the '!' character on the map. There will be exactly two teleport tiles in a dungeon. By stepping on a teleport tile, the player has the option to activate teleport and move directly to the matching teleport tile elsewhere in the dungeon. Thus, the two teleport tiles must be connected by an edge with a weight of 1 in your graph.

Consider the following example dungeon (Dungeon 3):

```
######
E..X.#
#!##T#
#.K.!#
######
```

What is the shortest path from 'T' to 'E'? If you go up, you'll have to go through an 'X' tile (trap), thus the total path length will be 7. If you go down, you can use the teleport, thus the total length of the path will be 4. The `getShortestPathString` function should indicate teleportation by adding "teleport" to the string when teleportation takes place. For the Dungeon 3 example, the following string must be returned (door to key to treasure):

```
right, down, down, right, right, right, up, down, teleport, up, left.
```

If two paths of the same length are encountered, where one path uses teleportation and the other one does not, the path without teleportation should be preferred.

Submission instructions

It should be possible to compile your code with the following command:

```
javac *.java
```

A makefile will be provided, and thus does not need to be uploaded. Once you are satisfied that everything is working, you must tar all of your Java code, including any additional files which you've created, into one archive called sXXX.tar.gz, where XXX is your student/staff number. Make sure there are no folders or subfolders in the archive. Submit your code for marking under the appropriate link (Assignment 4) before the deadline.