COSC 1285/2123 Algorithms and Analysis
Semester 1, 2019

Assignment 2: Path Finding

Due date: 11:59pm Friday 31st May 2019
Weight: 15%
Pair (Group of 2) Assignment

1 Objectives

There are three key objectives for this assignment:

- Consider real problems and how algorithms can be used to solve them.
- Study path finding and develop algorithms and data structures to solve path finding for different realistic scenarios.
- Research and extend to practice developing algorithmic solutions beyond just applying them.

2 Background

The standard path finding involves finding the (shortest) path from an origin to a destination, typically on a map. This is an important problem and has many applications, including robotics, games and simulations, and maze solving. In this assignment, we'll implement and develop approaches for path finding and extend these to interesting but realistic scenarios.

In the following, we first provide more details about path finding, then provide some ideas on how to represent a map and finally describe existing approaches to find these. In lectures we will also go through some of these in more details, so make sure you attend or if you can't make it, as least tune in and look at the notes that will be put up.

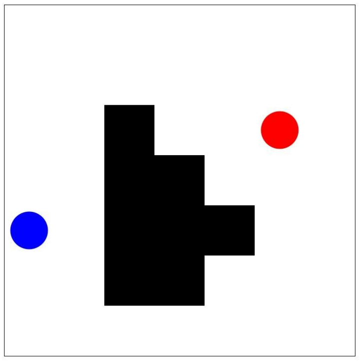
2.1 Path finding

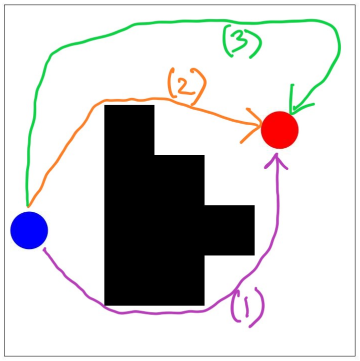
Path finding involves finding a path from A to B. Typically we want the path to have certain properties, such as being the shortest or to avoid going through certain obstacles. As the main aim is to think about path finding, we focus on the common task of finding shortest paths from A to B - shortest here means the path that has less total costs (more on this later, but given if we can travel at a constant speed over all parts of the map, then this equates to a path that is of least distance). The path is typically on a map (e.g., game) or a surface (e.g., robot).

Consider the map below (see Figure 1a), where we want to find a shortest path from origin point A to destination point B. There are obstacles and impassable locations (e.g., consider them walls in a maze, or a building one cannot enter in a game etc) that we cannot traverse (these are the block areas in the figure). If we use Euclidean distance as the cost (so the further one travels, the more costly), then the orange path, labelled (2), in Figure 1b is the shortest one[1]. The other two are possible paths but not the shortest.

To find (shortest) paths, we need to represent this map on a computer. There are several approaches to do so, which we will describe in the following subsection.

[1]Assume Jeff's hand drawings are straight.


(a) Original map.



(b) Map with a number of paths.

Figure 1: Example map and paths. The blue dot is the origin point, and red is the destination point. The black areas are obstacles and impassable locations that cannot be traversed.

2.2 Representing the surface or map

To represent the map, we can divide it into a number of coordinates. For this assignment, we assume a flat map/world so all coordinates are 2D, e.g., (row,column) or (r,c). To represent these coordinates and the map, there are generally two approaches:

- Grid representation: Dividing the map into a number of regular sized cells and considering the map as a grid of cells (see Figure 2a for an example with rectangular cells). Each cell represents a coordinate.
- Graph representation: Representing the map as a graph, where each node represents a coordinate and edges represent the possibility of moving from one coordinate to an adjacent one (think of the problem in tutes about mazes and how we can use a graph to represent the maze) (see Figure 2b). Edge weights represent the cost to travel between the coordinates.

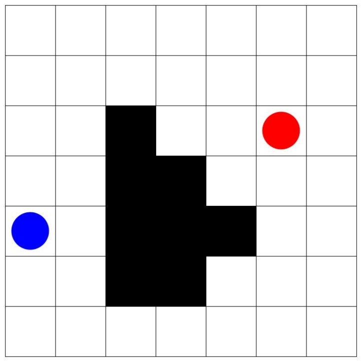
Because of the one to one mapping between cell, coordinate and node, each cell maps to a node, hence in essence both convey the same information about a map.

The grid representation is generally easier to understand, and we have a basic implementation in the skeleton code to provide map and path visualisation. However, it has weaknesses, including difficulty in representing "non-standard" adjacencies and not able to directly use the multiple graph algorithms available to solve the path finding related problems. In comparison, the graph representation, although conceptually more difficult to understand initially, once it is understood has the same representation power as the grid one, but is generally more flexible and has many graph-based algorithms that can solve different path finding scenarios. In this assignment, you can use either, but conceptually we suggest it is easier to think about it as a graph, and in lectures we will discuss in terms of a graph representation.

2.3 Path finding Approaches

Once we represent a map as a graph, we can use the full set of graph algorithmic tools. A path in the original map is also a path[2] in the graph, which means we can use algorithms such as BFS to find a

[2]Technically can be a (graph) walk, but if you don't know what this is, don't worry, not important just a curious side note.


(a) Grid representation, with rectangular cells.

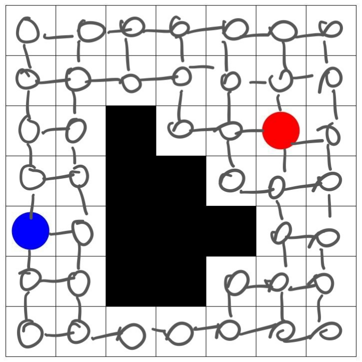

(b) Graph representation. Each node represents a coordinate and each edge represents adjacency and ability to traverse.

Figure 2: Example of the two map representations.

path from a origin point/coordinate (which corresponds to a starting node in the graph representation) to a destination point/coordinate (another node in the graph).

The cost of a path on a graph representation is then the sum of all the individual travel cost between nodes it traversed, or the sum of edge weights on the path. E.g., in Figure 3 the path is the orange line, it traverses from the origin (blue), over 8 nodes between the origin (blue) and destination (red), then to the destination. The travel cost is the sum of all the edge weights between the the traversed nodes. In this example, it is an unweighted graph so we can assume all edges have weight of 1 or unit (travel) cost).

To find a shortest path, we can either use BFS for unweighted graphs, or Dijsktra's for weighted ones, see Figure 3 again for an example.

But one can imagine (and in this assignment, see task B) where the travel cost is not uniform, e.g., travelling up slope/gradients, on different surfaces etc, and we don't have uniform travel cost between adjacent coordinates. Then the shortest path may not be the orange one anymore.

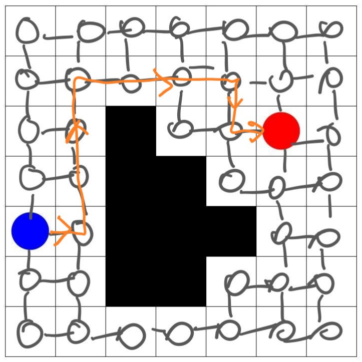


Figure 3: Shortest path found with Dijsktra's algorithm. Note that while the shortest path may not be unique, the total length/cost is.

Before we finish this section, note that the graph representation used in the examples are undirected, which assumes travelling from two adjacent coordinates, say A to B has the same cost as B to A, hence we don't need direction as they convey the same information. But if the cost aren't the same from A to B and B to A, then we need to use a directed (and weighted) graph representation.

2.4 Quick note about the coordinates system used

In this assignment, specifications and skeleton code implementation, the coordinates are based on (row, column) dimensions. Hence, the left most, bottom most coordinate on a map is row = 0, column = 0 or (0,0). In the example for either of the representations (Figure 2), the origin (blue) is row = 2, column = 0 or (2,0) and destination is row = 4, column = 5 or (4,5).

This concludes the background. In the following, we will describe the tasks of this assignment.

3 Tasks

The assignment is broken up into a number of tasks. Apart from Task A that should be completed initially, all other tasks can be completed in an order you are more comfortable with, although completion of task C is likely to help with task D and we consider task D as a distinction level one.

Task A: Implement Dijkstra's Algorithm for Path finding (4 marks)

To develop an initial approach to path finding, you should implement Dijkstra's algorithm. For this task initially, if not specified, assume all locations and coordinates have unit cost to travel to the adjacent one, e.g., (2,1) to (2,2) has a unit cost/cost of 1 to travel between.

Given input, your implementation should return a (shortest) path between origin and destination. Note there may be more than one shortest path, hence we ask for a shortest path. A path should include the origin cooridinate, all coordinates between, and the destination coordinate.

Hint: For a graph representation, we can use Dijkstra's algorithm as is. For a grid representation, you'll need to consider how to implement Dijkstra's on it. For both, we suggest to first figure it out conceptually, then translate into code.

Task B: Handling different Terrain (3 marks)

In maps, we frequently have locations that are of different surfaces, e.g., sand, gravel, mud etc. This can affect travelling speeds.

Up to this task, traversal between adjacent locations/coordinates are assumed to have a cost of 1. But in this task, now the costs can be greater than 1, and will be specified in a terrain parameter file (see Details of Files section for more details).

In this task, you'll need to modify your data structures and algorithms to handle terrain cost greater than 1. This might mean a path that was shortest for uniform unit cost map will now be very costly, e.g., it has to go through many coordinates of high terrain cost such as mud. As an example, see Figure 4, where the original orange path of Figure 2b is no longer the shortest as it has to go through coordinates with high costs, e.g., coordinate (5,2) with cost 12. Instead the purple coloured path is the shortest.

Hint: you'll need to consider how to translate this into the graph representation if you using that. There are at least two ways you can model this. One is to carefully consider what is the meaning of an edge weight in the graph representation, then whether you need a directed graph representation.

Task C: Handling multiple origins and destinations (3 marks)

Sometimes we can have multiple origin and destination points, e.g., we have a number of ambulances waiting at a number of hospitals (these are our origin locations) and we have a number of emergencies

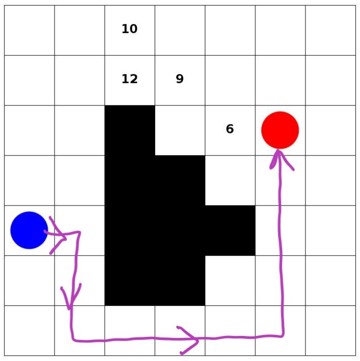


Figure 4: Map with terrain cost greater than 1 (coordinates/cells with numbers). Corresponding shortest path is rediverted in the other direction because of the terrain costs.

to attend to (these are the destination locations). We want to send an ambulance to one of the cases asap, hence it doesn't matter which hospital (origin) or to which emergency (destination), as long as the path is shortest.

In this task, you'll modify your implementation to cater for multiple origin and/or destinations. Note that we just need to find a shortest path between any of the origins to any of the destinations.

As an example, see Figure 5. There are two origins and two destinations, hence four possible paths between the origins and destinations. Path labelled (1) and coloured orange is the shortest one among the four.

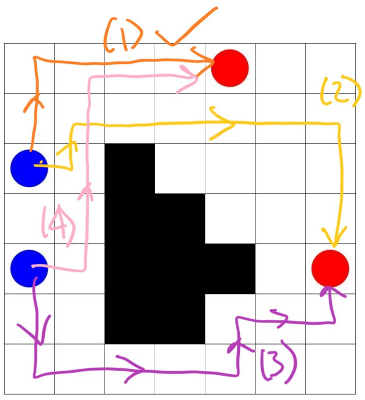


Figure 5: Map with multi-origin (blue) and multi-destinations (red). There are 4 possible paths, but the shortest one is the orange one, labelled (1).

Hint: Consider how Dijsktra's searches for its shortest path.

Task D: Waypoints (3 marks)

Note this task is probably the hardest of the four and we consider it as an distinction level task, but you are welcome to tackle these in the order you find easiest or desire. In path finding, it is typical that we must visit a number of waypoints when traversing from origin to destination, e.g., in a game, the player sets some waypoints so their units can avoid certain enemies etc.

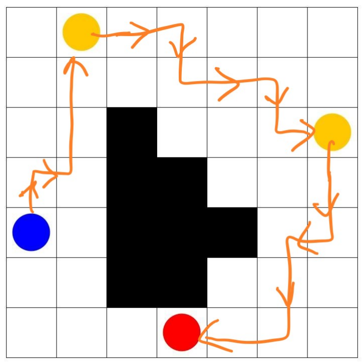


Figure 6: Map with multiple waypoints (yellow circles). Corresponding shortest path has to go through the waypoints instead of direct shortest path from origin to destination.

In this task, you are given a number of waypoints, and your path needs to traverse through all of them and be the shortest one that does.

Hint: There a number of solutions for this. But as a hint, consider task C and whether it has some similarities to task D.

4 Details for all tasks

To help you get started and to provide a framework for testing, you are provided with skeleton code that implements some of the mechanics of the path finding program. The main class (PathFinderTester) implements functionality of path finding, parsing parameters and read in the details of the map/surface that we are seeking a (shortest) path on. The list of main java files provided are listed in Table 1.

file	**description**
`PathFinderTester.java`	Class implementing basic framework of the data service. *Do not modify useless have to.*
`pathFinder/PathFinder.java`	Abstract class for path finding algorithms (in case you wanted to experiment with others). *Can add to, but don't modify existing methods.*
`pathFinder/DijkstraPathFinder.java`	Class implementing the Dijsktra's path finding algorithm *You need to implement this, minimum for task A.*
`map/Coordinate.java`	Class implementing some coordinate functionality *Can add to, but don't modify existing methods, as visualisation relies on them.*
`map/PathMap.java`	Class implementing a basic grid representation. *Can add to, but don't modify existing methods, as visualisation relies on them.*
`map/StdDraw.java`	Class implementing the visualisation. *Do not modify.*

Table 1: Table of supplied Java files.

Note, you may modify the PathFinderTester class, but we strongly suggest you not to, as this contains the code to load input/output and call relevent methods in your implementations and you do not want to break this. You may add java files and methods, but it should be within the structure of the skeleton code, i.e., keep the same directory structure. Similar to assignment 1, this is to minimise compiling and running issues. However, you should implement all the missing functionality in *.java files. However, ensure your structure compiles and runs on the **core teaching servers**. Note that the onus is on you to ensure correct compilation and behaviour on the core teaching servers before submission.

As a friendly reminder, remember how packages work and IDE like Eclipse will automatically add the package qualifiers to files created in their environments. This is a large source of compile errors, so remove these package qualifiers when testing on the core teaching servers - and this was the case for assignment 1 so please avoid this.

Compiling and Executing

To compile the files, run the following command from the root directory (the directory that PathFinderTester.java is in):

```
javac -cp .:jopt-simple-5.0.2.jar PathFinderTester.java map/*.java pathFinder/*.java
```

Note that for Windows machine, remember to replace ':' with ';' in the classpath specifier (-cp part).

To run the framework:

```
java -cp .:jopt-simple.5.0.2.jar PathFinderTester [-v] [-o <path output file>] [-t
          <terrain file>] [-w <waypoints file>] <main parameter file>
```

where

- optional [-v]: visualise map and (if implemented) discovered path.
- optional [-o <output file>]: for our testing purposes, outputs the discovered path to "output file".
- optional [-t <terrain file>]: specify the loading of terrain information from "terrain file".
- optional [-w <waypoints file>]: specify the loading of waypoint information from "waypoints file".
- main parameter file: name of the file that specifies the map, origin and destination and impassable coordinates information.

We next describe the contents of the parameter files.

4.1 Details of Files

Main parameter file

This specifies the main configuration of the tasks and assignment, including:

- number of rows and columns in the map
- list of origin coordinates in (row, column) format
- list of destination coordinates in (row, column) format

- list of impassable coordinates in (row, column) format

The exact format is as follows:

```
[number of rows in map] [number of columns in map]
[List of row,col coordinates of origin points]
[List of row,col coordinates of destination points]
[row column (coordinates) of impassible points]
```

Using the example from Figure 2a, the parameter file corresponding to this example would be:

```
7 7
2 0
4 5
2 2
3 2
4 2
1 2
1 3
2 3
3 3
2 4
```

First line, “7 7” is the number of rows and columns. Second line, “2 0” is the (row, column) coordinate of the origin. Third line, “4 5” is the (row, column) coordinate of the destination. Fourth to end (starting at “2 2”) are the (row, column) coordinates of each impassable coordinate. So in this case, there are 8 impassable ones.

The skeleton code will parse and load these. We explain the format here so you can develop your own test cases. Examine the skeleton code, the origin and destination coordinates are loaded as List of coordinates, while the terrain information is loaded into the PathMap class itself.

Terrain parameter file

Coordinates that have terrain cost greater than 1 are listed in the terrain input files (all coordinates not mentioned in the terrain file are assumed to have unit cost (cost = 1)), which has the following format:

```
[row column (coordinates) terrain cost]
```

Using the example from Figure 4, the terrain parameter file is as follows:

```
5 2 12
6 2 10
5 3 9
4 4 6
```

We have added code in the provided skeleton that loads this information from the terrain parameter files and stores them as a Map (Coordinate, Terrain cost) structure that you can use to construct your approach to handle different terrain.

Waypoint parameter file

This file specifies the optional waypoints. The waypoints are stored in the following format:

```
[row column (coordinates)]
```

Using the example from Figure 6, the waypoint parameter file is as follows:

```
6 1
4 6
```

Again we load the waypoints for you in the skeleton code. They are available as a List of Coordinates. Note that the waypoints are not provided in the order they are visited, you'll have to work out the optimal order for this.

4.2 Clarification to Specifications

Please periodically check the assignment FAQ for further clarifications about specifications. In addition, the lecturer will go through different aspects of the assignment each week, so even if you cannot make it to the lectures, be sure to check the course material page on Canvas to see if there are additional notes posted.

5 Assessment

The assignment will be marked out of 15.

The assessment in this assignment will be broken down into a number of components. The following criteria will be considered when allocating marks. All evaluation will be done on the core teaching servers.

Task A (4/15):
For this task, we will evaluate your implementation and algorithm on whether:

1. Implementation and Approach: It implements Dijkstra's algorithm and adapted to solve path finding.
2. Correctness: Whether if finds a shortest path for given input maps.

Task B (3/15):
For this task, we will evaluate your implementation and algorithm on whether:

1. Implementation and Approach: It takes a reasonable approach and can incorporate terrain information when computing shortest paths.
2. Correctness: Whether if finds a shortest path for given input maps (which can include terrain information).

Task C (3/15):
For this task, we will evaluate your implementation and algorithm on whether:

1. Implementation and Approach: It takes a reasonable approach and can incorporate multiple origins and destinations when computing shortest paths.
2. Correctness: Whether if finds a shortest path for given input maps (which can include multiple origins and destinations).

Task D (3/15):
For this task, we will evaluate your implementation and algorithm on whether:

1. Implementation and Approach: It takes a reasonable approach and can incorporate waypoints when computing shortest paths.
2. Correctness: Whether if finds a shortest path for given input maps (which can include waypoints).

Coding style and Commenting (1/15):

You will be evaluated on your level of commenting, readability and modularity. This should be at least at the level expected of a second year undergraduate/first year masters student who has done some programming courses.

Description of Approach (1/15):

In addition to the code, you will be assessed on a brief description of your approach, which will help us to understand your approach (this will help us with assessing the Implementation and Approach of all tasks). Include your representation of the map and briefly your implementation approach to Dijsktra's algorithm (task A), how you modelled and implemented the terrain (task B), your approach to handling multiple origins and destinations (task C) and how you implemented waypoints (task D). If you didn't implement one or more of the tasks, then no need to describe it. This should be no longer than one page and submitted as a pdf as part of your submission. You will be assessed on its clarity and whether it reflects your code.

5.1 Late Submissions

Late submissions will incur a *deduction of 1.5 marks per day or part of day late.* Please ensure your submission is correct (all files are there, compiles etc), resubmissions after the due date and time will be considered as late submissions. The core teaching servers and Canvas can be slow, so please ensure you have your assignments are done and submitted a little before the submission deadline to avoid submitting late.

6 Team Structure

This assignment is designed to be done in *pairs* (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. If there are issues with work division and workload in your group, please contact your lecturer as soon as possible.

In addition, please submit what percentage each partner made to the assignment (a contribution sheet will be made available for you to fill in), and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution X%, and student B has contribution Y%, and $X > Y$. The group is given a group mark of M. Student A will get M for assignment 1, but student B will get $\frac{M}{\frac{X}{Y}}$.

7 Submission

The final submission will consist of:

- The Java source code of your implementations, including the ones we provided (please also include the provided jar file). Keep the same folder structure as provided in skeleton (otherwise the packages won't work). Maintaining the folder structure, ensure all the java source files are within the folder tree structure. Rename the root folder as `Assign2-<partner 1 student number>-<partner 2 student number>`. Specifically, if your student numbers are s12345 and s67890, then all the source code files should be within the root folder Assign2-s12345-s67890 and its children folders.
- All folder (and files within) should be zipped up and named as `Assign2-<partner 1 student number>-<partner 2 student number>.zip`. E.g., if your student numbers are s12345 and

s67890, then your submission file should be called `Assign2-s12345-s67890.zip`, and when we unzip that zip file, then all the submission files should be in the folder Assign2-s12345-s67890.

- Your report of your approach, called "assign2Report.pdf". Place this pdf within the Java source file root directory/folder, e.g., Assign2-s12345-s67890.
- Your group's **contribution sheet**. See the previous 'Team Structure' section for more details. This sheet should also be placed within your root source file folder.

Note: submission of the zip file will be done via Canvas.

8 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted assignment work in this course is to be the work of you and your partner. It should not be shared with other groups. **Multiple automated similarity checking software will be used to compare submissions**. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the students concerned. Plagiarism of any form may result in zero marks being given for this assessment and result in disciplinary action.

For more details, please see the policy at `http://www1.rmit.edu.au/students/academic-integrity`.

9 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Canvas for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Demonstrator. We will also be posting common questions on the assignment 2 Q&A section on Canvas and we encourage you to check and participate in the discussion forum on Canvas. However, please **refrain from posting solutions**, particularly as this assignment is focused on algorithmic and data structure design.