CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2021
Webpage: `http://www.cs.clemson.edu/~bcdean/` — MWF 9:05-9:55
Handout 17: Homework #4 — Powers 208

1 Optimal Path Finding

This problem will allow us to practice solving for shortest paths in a graph with a somewhat interesting state space of nodes. Files for the assignment are available here:

`/group/course/cpsc212/f21/hw04/`

The main file is `railway.ppm`, an image file of a "railway maze" puzzle created by the famous mathematician / physicist Roger Penrose:

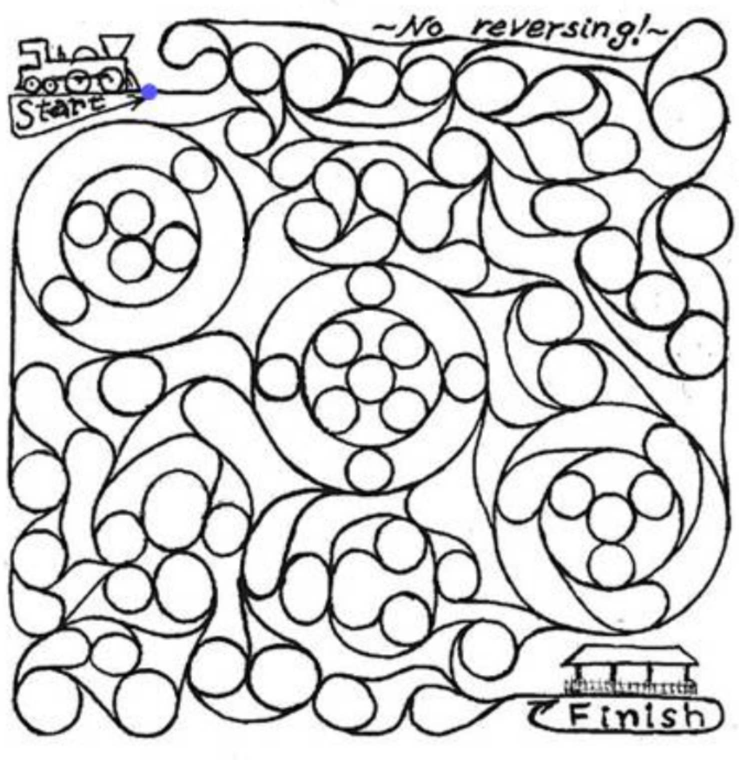

Figure 1: A Railway Maze

To solve the railway maze, one must find a shortest path from start to finish, where movement models that of a train: reversing direction is not allowed, and only gradual turns can be made.

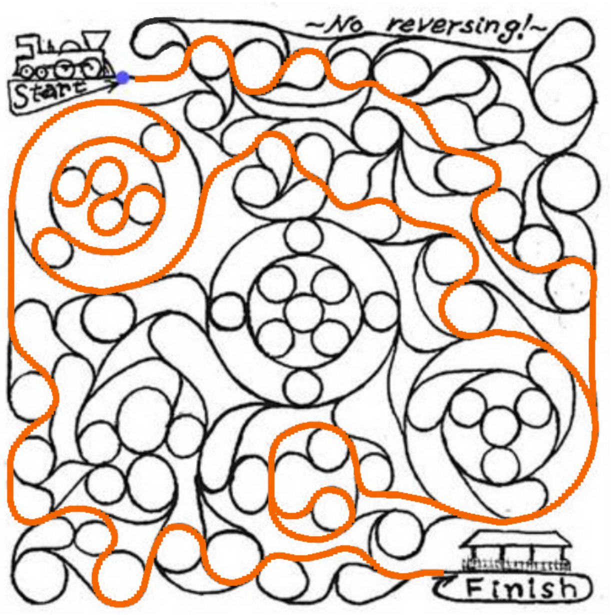

Figure 2: Solution to the Railway Maze (Ignoring Passenger Constraints)

2 Problem Setup

All of the constraints and input parameters for the problem are provided in a configuration file — an example is given to you in the file `config.txt`. Your program should accept the name of this configuration file on the command line, then read the configuration file to figure out all the details of the problem it needs to solve.

The lines of the configuration file tell the following information:

- The name of the PPM image file[1] describing the maze to be solve.
- (x, y) coordinates of the starting point pixel (x is horizontal and y vertical, with $(0, 0)$ being the upper-left corner).
- (x, y) coordinates of the goal point pixel.
- The next four lines respectively give (x, y) coordinates of Prof. Dean, TA Zhihong Lin, TA John Perez, and TA John Reeder.
- The threshold T governing which pixels are valid to choose as part of your solution path — you can include a pixel as part of your path if the sum of its red, green, and blue color values is at most T.
- The minimum and maximum Euclidean distance each step of your path can take. Your solution path consists of a series of pixels, and consecutive pixels along the path must respect these constraints so each step you take is not too short or not too long.

[1]See the code posted on Canvas for the exercise we solved in class coloring in the tiger paw image for a good reference on reading and writing PPM files.

- The maximum number of degrees M you can turn left or right in any step of the path relative to the direction of the previous step. This constraint is what forces you to only be able to turn gradually.
- A distance threshold D specifying the minimum distance you must be from one of the four passengers above before you can pick them up. For example, to pick up Prof. Dean, your path must include a pixel within distance at most D of Prof. Dean's location. This distance threshold also governs the end of the path — you are allowed to end the path at any pixel within at most D units of distance of the goal pixel.

In terms of passengers, your optimal path must be designed so it picks up Prof. Dean, Zhihong, and *at least one* of the two Johns. Hence, you probably want the state space represented by the nodes in your graph to include not just your present pixel location, but also information on the set of passengers you are currently carrying (a bitwise representation stored in a single int might be convenient here). You probably also need to incorporate information on the last pixel location visited into your state space, since this allows you to enforce the constraint that the vector v from your current pixel to the next pixel can turn at most M degrees from the vector u from the previous pixel to the current pixel[2].

For grading, your program may be run on several different configuration files, possibly involving different PPM railway maze images.

3 Output

Your program should write two output files:

- `path.ppm`, an image file that is the original PPM image but with the path taken from start to finish highlighted (i.e., with a colored dot or small box drawn at each pixel visited by the path). You don't have to highlight the path in Clemson orange but that would be a nice touch.
- `path.txt`, a text file describing the coordinates of each individual pixel making up your optimal path. Each line of the file should contain two space-separated integers giving the x and y coordinates of a pixel along the path, starting with the source pixel and ending with a pixel that is at most D units of distance away from the goal pixel. The course staff will provide a validation program that can read this file and check that all necessary constraints are being satisfied (e.g., that the path isn't turning too quickly).

Your program can also print things to the terminal (please keep this brief — there shouldn't be pages of output). However, its primary, graded output should be the two files above.

[2]Mathematically, this is easy to check, since if you take two vectors u and v each of unit length (which you can ensure by dividing them by their lengths), then their dot product $u \cdot v$ gives the cosine of the angle between them. For example, if both are pointing in exactly the same direction, the dot product is 1, and if they are at right angles the dot product is 0.

4 Submission Details

Solutions should run in less than 10 seconds, when compiled with the full optimization `-O9` flag, for full credit. Substantial partial credit will be awarded for slightly slower solutions.

Other scenarios that can earn partial credit are: finding a shortest path that neglects to pick up the required passengers, or finding a shortest path that ignores the constraint on turning.

Final submissions are due by 11:59pm on the evening of Friday, December 3. No late submissions will be accepted.