Homework 5

MacMillan, Kyle

November 26, 2018

Contents

List of Figures

1 Chapter 11

1.1 Problem 3

To accomplish this problem I created a class to generate a random map of any size you want up to 99. It generates a random number of obstacles of random size. Play around with it, it's fun. You can change the random numbers or just run the file multiple times. Code for this problem can be found here.

Demonstration is shown in Figure 1. This is an application of the WaveFront BFS algorithm. From there it goes on to find the shortest path as seen in Figure 2.

The seed for that particular example is: `7635686187880284248`

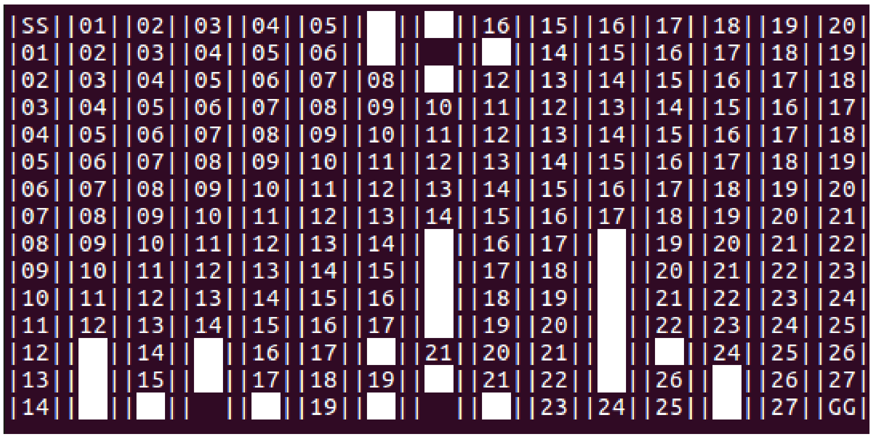

Figure 1: WaveFront Distance Evaluation

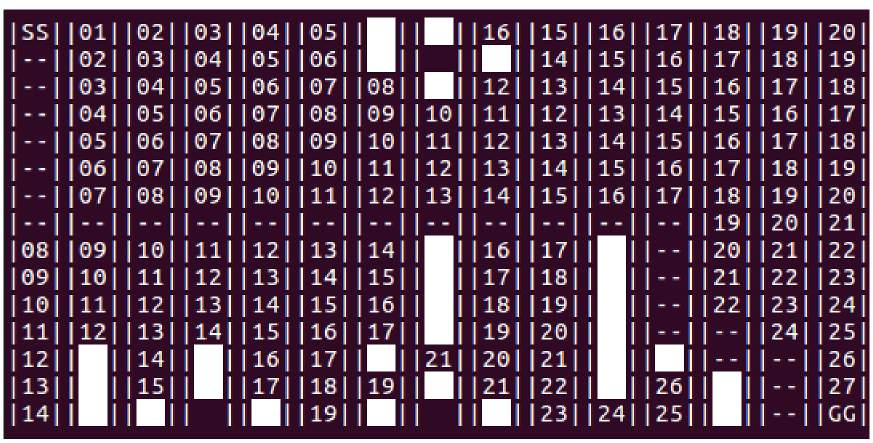

Figure 2: WaveFront Shortest Path

The book constantly refers to this as a start to goal process but that is not how flood-fill works because you can not guarantee shortest path; you can not have it both ways. The book calls for "minimal path" but then uses a non-optimal solver. I wrote it as the book asked the first time around then changed it to actually give the optimal answer because that's what we want.

To clarify: Starting from the start is fine but you have to traverse it backwards from goal to start. The reason is because you do not know which of the "equal" numbers to choose when you're walking from start to goal. You **know** which path is the shortest if you walk from goal to start. If two numbers are the same then either path is optimal/minimal.

2 Chapter 16

2.1 Problem 1

asdf

2.2 Problem 2

Given 40 measurements at 2 meters we can calculate the mean:

A 2.23521735

B 1.86443904

C 2.33806001

That is how far off each senor averages from 2 meters. Each new sensor reading then has that mean subtracted from it to yield:

A 2.22255225

B 2.03233526

C 1.79719609

We can then apply the formula provided in the sensor fusion portion of the book for an expected distance of: `2.057449909256987` meters. The code for this can be found here and in the code snippet 2.2.

```
# Calculate mean and standard deviation
mean = np.mean(dist_sens, dtype=np.float64, axis=0)
std = np.std(dist_sens, dtype=np.float64, axis=0)

# Reshape because it doesn't need to be 2D
np.reshape(mean, (1, 3))
np.reshape(std, (1, 3))

# Input for this step
new_sens = np.array([2.4577696, 1.8967743, 2.1352561]) + (2.0 - mean)

# Calculate variance
var = std * std

# Sensor fusion to obtain x_hat
top = 0.0
bot = 0.0
for i in range(3):
    top += (new_sens[i] / var[i])
    bot += 1 / var[i]

# Estimated distance
x_hat = top / bot
```

3 Chapter 17

3.1 Problem 2

asdf

3.2 Problem 3

asdf

3.3 Problem 4

3.3.1 Problem 4a

asdf

3.3.2 Problem 4b

asdf

3.3.3 Problem 4c

asdf

3.3.4 Problem 4d

asdf

3.3.5 Problem 4e

asdf

4 Chapter 18

4.1 Problem 1

4.1.1 Problem 1.1

asdf

4.1.2 Problem 1.2

asdf

4.1.3 Problem 1.3

asdf