Programming and Mathematics for AI Coursework

Task 1

By Muhammad Faaiz Ansari

Table of Contents

-task 1 description:

The aim of task 1 was to create a grid game based on a grid of random numbers between 0 and 9. Starting from the top left, the aim was to find a path to the bottom right. First find the length of a path using a basic heuristic algorithm that is better than random movement. Secondly, to find the length of the shortest path based on Dijkstra's shortest path algorithm.

-game implementation

First of all, I must note that I had issues with timing due to the fact my partner decided to split up and do the coursework separately due to his own personal issues too late to partner with anyone else. I also had personal issues a month before the due date for the coursework which reduced the time I had to work on this course work. I also needed to fit in my 3 other units coursework projects.

I had to compromise on quality of some of my work, so I was only able to implement a basic grid game with only the first game mode where cell number is time spent. I used a 2d numpy array in python, with its cells randomized and a set 4 x 4 grid size.

```
# Convert 2d list to numpy array
gridNpA = np.array(gridList)
print(gridNpA)
```

```
[[3 4 1 6]
 [7 2 1 1]
 [0 6 8 4]
 [0 3 8 8]]
```

A way to build this better would be to make a game class with 2 different modes to select. Unfortunately, due to timing issues I faced, I couldn't implement this.

-basic heuristic algorithm

As the basic heuristic algorithm was of smaller weightage, I decided to make a simple algorithm which first moved to the right most cell, and then the bottom most cell. This would be better than a random movement as required in the question. The path wouldn't have any backtracking as the movement is only right and down with no left and right. Of course, this wouldn't give the shortest path, as there may be better paths available, however, this performs much better than just random movement. With the grid illustrated above, the output would be

```
#run the basic algorithm on the grid
basic = BasicAlg(gridNpA)
print(basic.alg())
```

```
24
```

-Dijkstra algorithm

For the Dijkstra algorithm, I had more focus more time as it would be more complicated. I decided to an approach based on transforming the grid to a node structure using convertGrid class. Then I built a Dijkstra algorithm using some code from an external reference (added in references below) as a base. I adapted it to fit my approach as It was built more for nodes specifically. I then used this code to find the shortest path according to Dijkstra's shortest path algorithm using loops, if statements, dictionaries, and list slices amongst others. In the above illustrated grid, the shortest path was of size 19, with the current and end strings multiplied to overcome a value error as shown below:

```
#find dijsktras shortest path output
print (dijkstraAlg(graph,str(C),str(E)))
```

```
(19, ['0', '1', '2', '102', '103', '203', '303'])
```

Overall, I had some struggles with different ways to write the algorithm and errors I faced and had to overcome, but I did find a satisfactory solution which shows the size of the shortest path according to Dijkstra's algorithm.

-Analysis: to find length of shortest path based on grid size and cell numbers generated?

The shortest path size increases overall if the dimensions of the grid increase. This is because the more cells there are overall, the more cells there will be between the start and end cells, and so the path will add up in size as more cells need to be crossed. Similarly, if the size of each cell numbers generated was greater, the sum of all the cell numbers in the shortest path would be greater exponentially.

This problem would effect both game modes, however, as the absolute difference between cells might increase(with a greater range of cell numbers), the increase in difference would be smaller in proportion to the overall increase in the sum of cell numbers in the 1st simple game mode. If I had built both game modes I could illustrate through samples of runs that the greater effect is on the time spent = cell number game mode.

-reference external sources

-I based a large part of my Dijkstra algorithm code on the ideas learned from the following website. I wrote my own code but also used some code for some small parts up to 20% of my Dijkstra algorithm code to overcome errors. Some with changes and adapting to fit my task approach: https://www.udacity.com/blog/2021/10/implementing-dijkstras-algorithm-in-python.html

-I also learned more about using python classes, functions, etc. I used in my code from the following site: https://www.programiz.com/python-programming/class

-I used some information from the networkx package to base what I needed to build for my own code without using the package itself to convert my nunpy matrix grid to a network of nodes: https://networkx.org/documentation/stable/reference/convert.html#module-networkx.convert_matrix

I also used a couple of videos and the lectures about Dijkstra's algorithm to learn more about building my own Dijkstra algorithm code.