Assignment #3
02/23/2020

Prepared
for
Prof. Hammoud

Prepared
by
Abedin Sherifi

Summary:

In this coding assignment, PyCharm was used throughout. The following Python libraries were utilized: random, time, os, and matplotlib. Two python files were used to complete this coding assignment. The Random_Maze.py file generates a random maze with obstacles accounting for 30% of the total maze size. This newly generated maze is then saved as a .txt file which then gets opened and read in the Grid_Maze_Search.py file. The Grid_Maze_Search.py file solves the maze by first reading the maze from a text file from a directory and then using Breadth-First-Search (BFS) solving the maze. Cost function was also computed for each maze size by dividing the number of moves it took to solve the maze by the solve time. The start point on the maze is marked with “S” at the top left corner of the maze. The goal point of the maze is marked with “G” at the bottom right corner of the maze. Randomly generated obstacles are marked with “O” on the maze. The point robot is not allowed to visit an obstacle cell. The point robot is supposed to traverse the maze horizontally.

Results:

Figure 1 below summarizes the cost function for each grid(maze) problem. The BFS algorithm per the data below indicates that the cost function decrease as maze size increases. It can be observed that the maze size of 500x500 has the lowest cost function from the rest of the maze sizes.

Grid	Cost (Moves)	Time (Sec)	Cost vs. Time (Moves/Sec)
10x10	28	0.000555276870728	50425.2949763847
50x50	148	0.01541018486023	9604.03793610273
100x100	298	0.060060262680054	4961.68326115176
500x500	1498	2.027174949646	738.959407653293

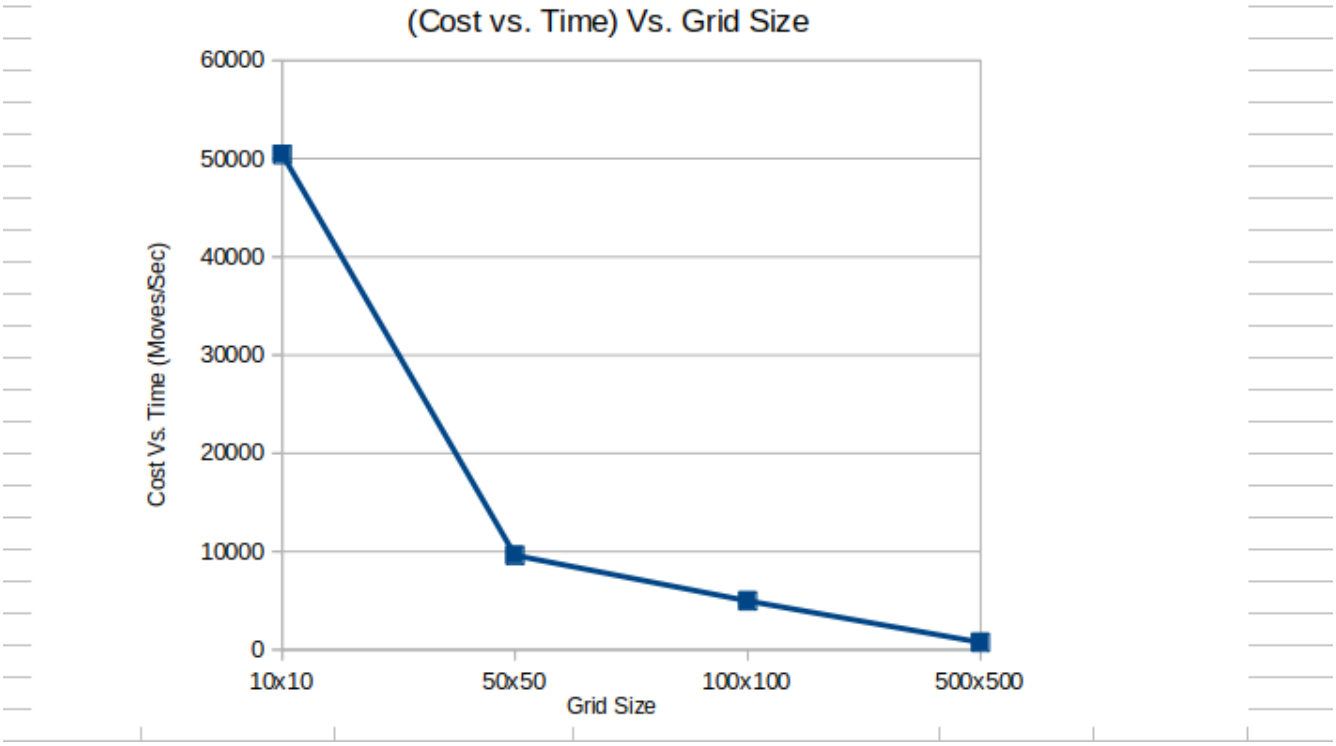

Figure 1. Cost Function

Maze = 10x10

The solved path provided by the algorithm is provided in figure 2 below as an array. The “S” start position is always (0,0). The path values are also plotted as shown below.

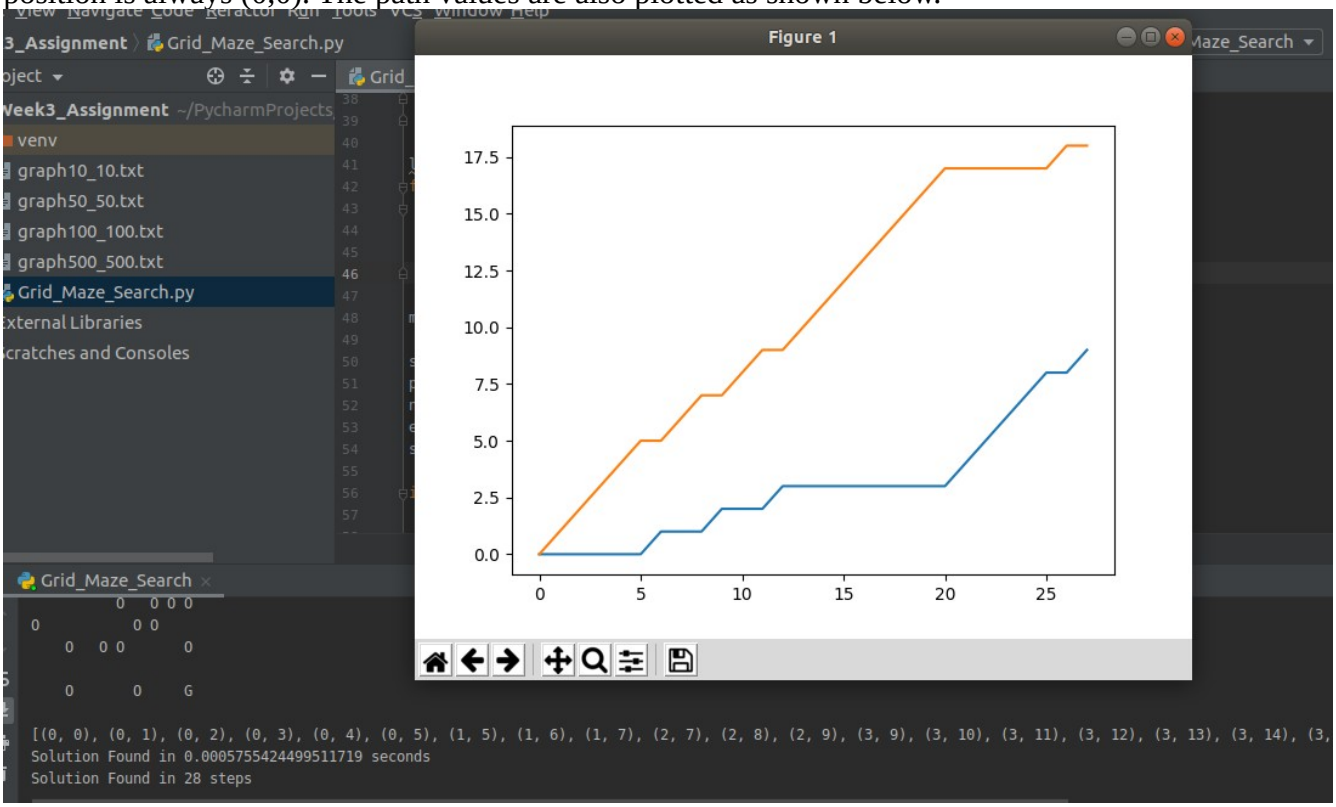

Figure 2. Maze 10x10 Result

Maze 50x50

The solved path provided by the algorithm is provided in figure 3 below as an array. The “S” start position is always (0,0). The path values are also plotted as shown below.

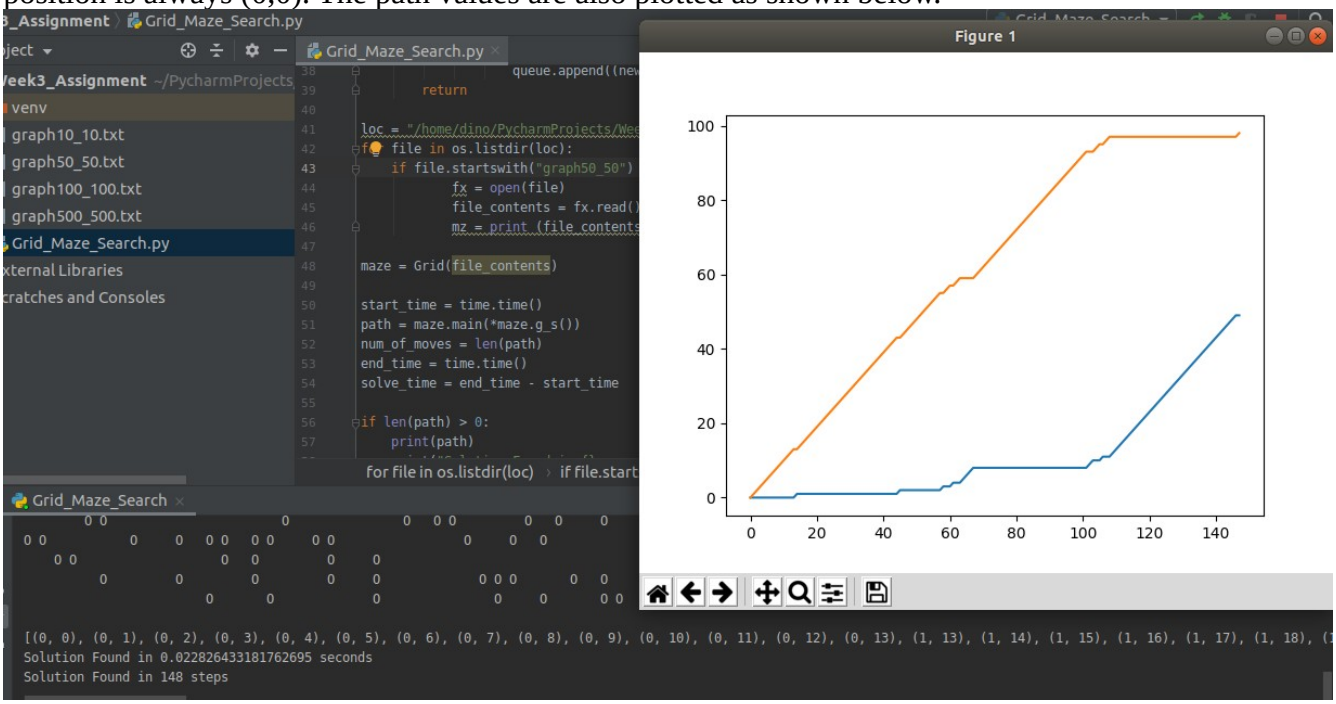

Figure 3. Maze 50x50

Maze 100x100

The solved path provided by the algorithm is provided in figure 4 below as an array. The “S” start position is always (0,0). The path values are also plotted as shown below.

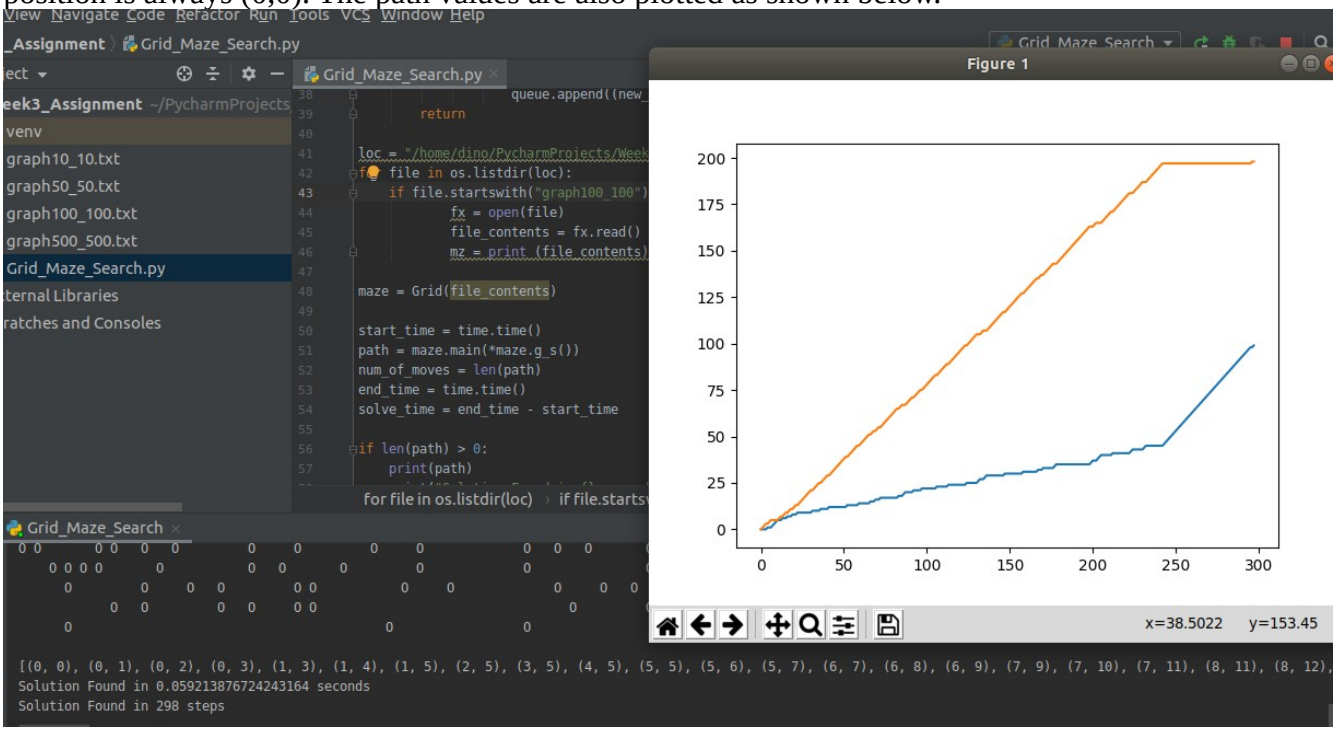

Figure 4. Maze 100x100

Maze 500x500

The solved path provided by the algorithm is provided in figure 5 below as an array. The “S” start position is always (0,0). The path values are also plotted as shown below.

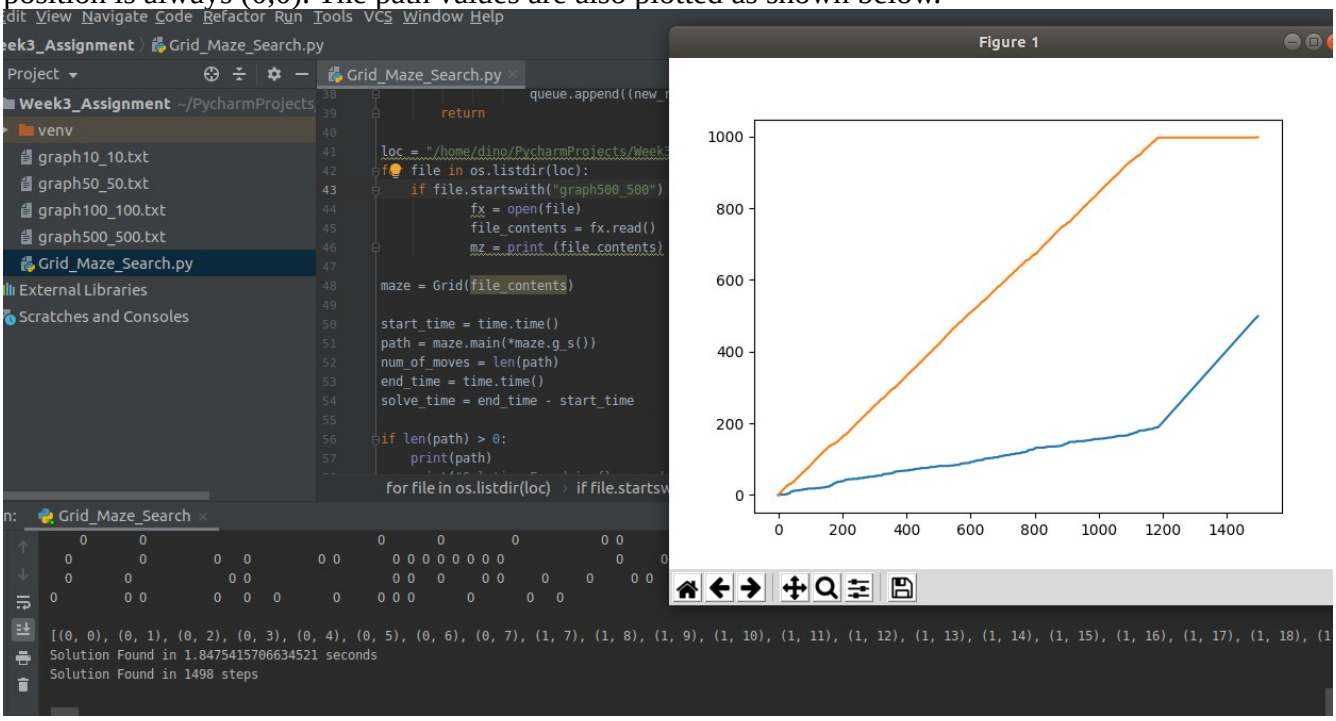

Figure 5. Maze 500x500

```
#Abedin Sherifi
#RBE 550, Assignment #3
#02/23/2020

import time
import os
import matplotlib.pyplot as plt
class Grid():
    def __init__(self, str):
        self.maze = str.splitlines()
    def g_s(self):
        x_r = next(i for i, line in enumerate(self.maze) if "S" in line)
        y_c = self.maze[x_r].index("S")
        return x_r, y_c
# This function solves the specified maze using BFS Algorithm. The usual BFS psuedo code is
# followed.
    def main(self, row, col):
        queue = []
        visited = {}
        visited[(row, col)] = (-1, -1)
        queue.append((row, col))
        while len(queue) > 0:
            row, col = queue.pop(0)
            if self.maze[row][col] == 'G':
                path = []
                while row != -1:
                    path.append((row, col))
                    row, col = visited[(row, col)]
                path.reverse()
                return path
```

```
            for dr, dc in ((-1, 0), (0, -1), (1, 0), (0, 1)):
                new_r = row + dc
                new_c = col + dr
                if (0 <= new_r < len(self.maze) and
                        0 <= new_c < len(self.maze[0]) and
                        not (new_r, new_c) in visited and
                        self.maze[new_r][new_c] != 'O'):
                    visited[(new_r, new_c)] = (row, col)
                    queue.append((new_r, new_c))
        return
# Open the .txt file that contains the specific maze and read the maze in the file.
loc = "/home/dino/PycharmProjects/Week3_Assignment"
for file in os.listdir(loc):
    if file.startswith("graph100_100") and file.endswith(".txt"):
            fx = open(file)
            file_contents = fx.read()
            mz = print (file_contents)
maze = Grid(file_contents)
start_time = time.time()
path = maze.main(*maze.g_s())
num_of_moves = len(path)
end_time = time.time()
solve_time = end_time – start_time
# Print number of moves and solve time for the specified maze.
if len(path) > 0:
    print(path)
    print("Solution Found in {} seconds".format(solve_time))
    print("Solution Found in {} steps".format(num_of_moves))
else:
    print("No Path Found")
```

```
# Plot path data
plt.plot(path)
plt.show()
```

```
#Abedin Sherifi

#RBE 550, Assignment #3

#02/23/2020

#This function generates a random maze with start position marked as “S” on the top left corner and the
#goal position marked as “G” at the bottom right corner. Obstacles are randomly generated and account
#for 30% of the space. All of the free spaces are set to spaces ‘ ’

import random

def Random_Maze(n_r, n_c):

#2-D Array

    row = n_r

    col = n_c

    maze = [[' ' for i in range(col)] for j in range(row)]

#Random obstacles added and account for 30% (0.3) of the total space.

    n_obs = round(row * col * .3)

    for i in range(n_obs):

        maze[random.randint(0, row - 1)][random.randint(0, col - 1)] = 'O'

    maze[0][0] = 'S'

    maze[row - 1][col - 1] = 'G'

    return maze


#Input for the generation of the maze (number of rows and number of columns).

if __name__ == '__main__':

    row = 100

    col = 100

    mz = Random_Maze(row, col)

    for row in mz:

        print(' '.join([str(elem) for elem in row]))
```