Question 1.1: Describe how you would frame the maze solver as a search problem.

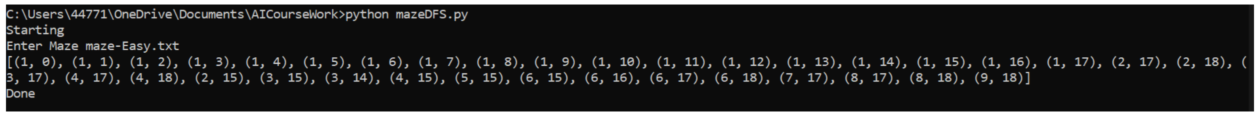
You can treat the solver as a search problem if you model the tree as a graph so that each space in the tree and adjacent spaces next to it are the nodes connected to it. With this in mind it is possible to then search the maze using a graph searching algorithm.

Question 1.2: Solve the maze using depth-first search.
This question is further divided into parts. This is a guide to help your in the coding process. You do not need
to submitted different versions of the code, but just the last updated version.
1. Briefly outline the depth-first algorithm.

Depth first search is an algorithm for searching through trees or graphs. The algorithm starts at a root node, which in the case of a maze is the entrance, and explores as far down as possible between each branch until it reaches the end of the branch. It then backtracks until it reaches a node which branches off to other branches. It then goes down as deep as it can down that branch. This process is repeated until the end goal is found or the entire graph is explored.

2. Implement depth-first to solve the maze. The solution of the maze should print the locations in the map
that the algorithm will take to reach from the start to the goal. This question can be solved using the file
'maze-Easy.txt".

Code found in the mazeDFS.py file

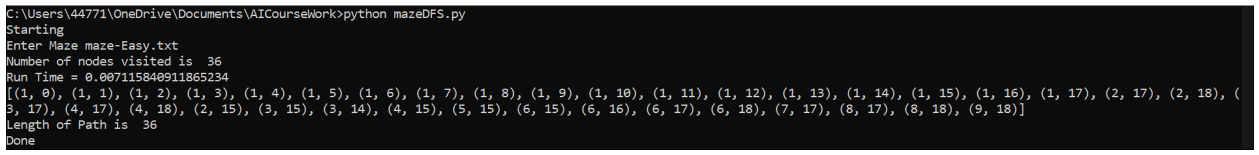


```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeDFS.py
Starting
Enter Maze maze-Easy.txt
[(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (1, 11), (1, 12), (1, 13), (1, 14), (1, 15), (1, 16), (1, 17), (2, 17), (2, 18), (
3, 17), (4, 17), (4, 18), (2, 15), (3, 15), (3, 14), (4, 15), (5, 15), (6, 15), (6, 16), (6, 17), (6, 18), (7, 17), (8, 17), (8, 18), (9, 18)]
Done
```

The numbers represent the coordinates in the maze the algorithm traversed.

3. Update your algorithm to calculate some statistics about its performance. The number of nodes explored
to find the path, the time of the execution, and the number of steps in the resulting path.

Code found in the mazeDFS.py file



```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeDFS.py
Starting
Enter Maze maze-Easy.txt
Number of nodes visited is  36
Run Time = 0.007115840911865234
[(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (1, 11), (1, 12), (1, 13), (1, 14), (1, 15), (1, 16), (1, 17), (2, 17), (2, 18), (
3, 17), (4, 17), (4, 18), (2, 15), (3, 15), (3, 14), (4, 15), (5, 15), (6, 15), (6, 16), (6, 17), (6, 18), (7, 17), (8, 17), (8, 18), (9, 18)]
Length of Path is  36
Done
```

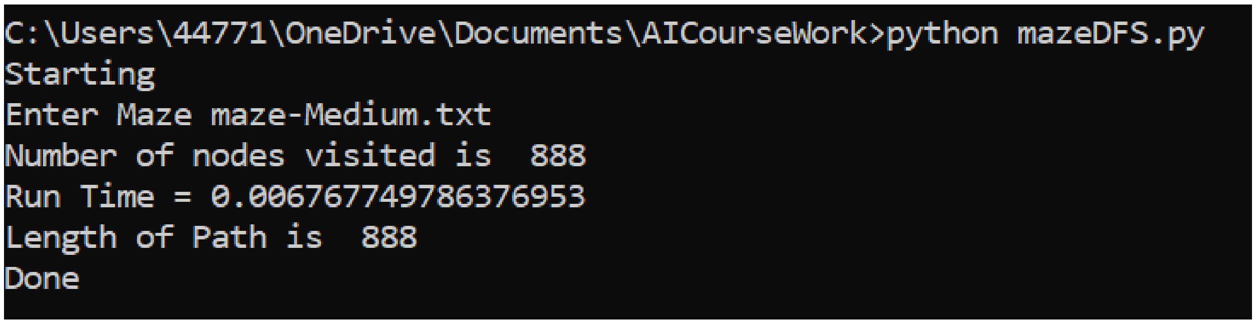
As you can see the path through the maze and the number of nodes the algorithm visited are the same. This is because DFS does not find an optimal path through a maze but instead searches through the maze until the end is found. Due to this path the algorithm produces is often inefficient.

Another thing of note from this is how fast the run time is, which is due to the algorithm not needing to consider finding the optimal path when running through the maze.

4. Update your algorithm to be generalised to read any maze in the same format. Then calculate the paths and statistics for all ‘maze-Medium.txt”/ ‘maze-Large.txt” / ‘maze-VLarge.txt”

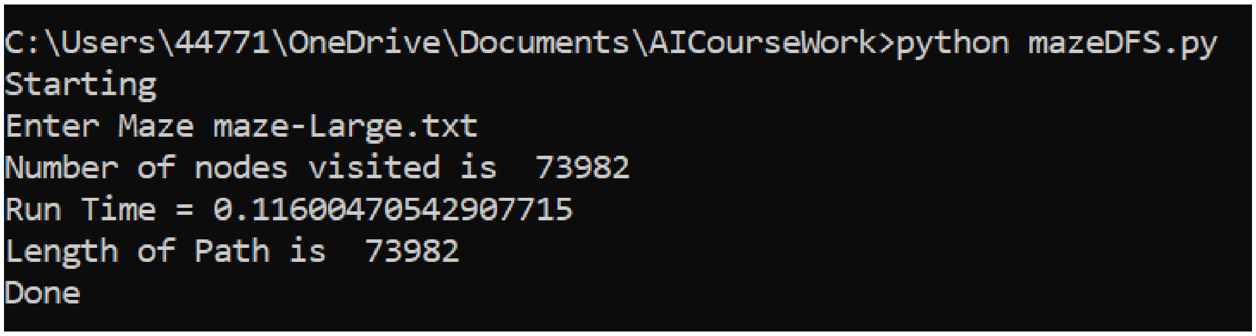
Code found in the mazeDFS.py file
Note: full path of made not shown as its too large, to see the full path uncomment the print statement on line 109 of mazeDFS.py and run the program

Medium:



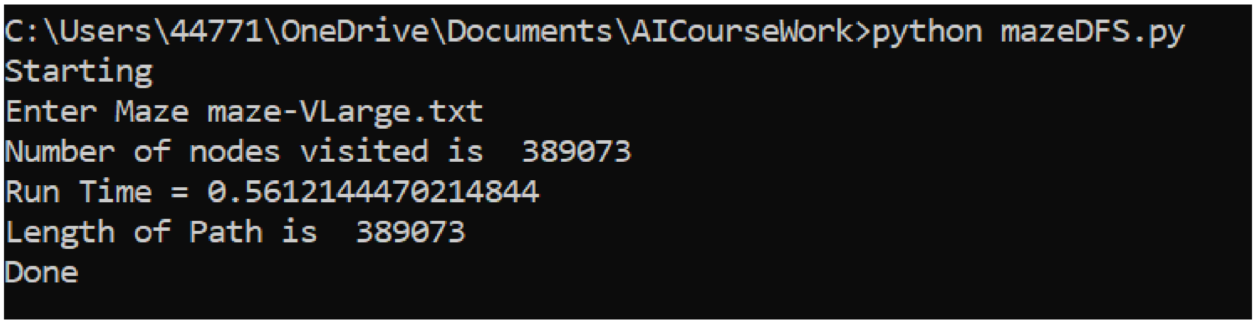
```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeDFS.py
Starting
Enter Maze maze-Medium.txt
Number of nodes visited is  888
Run Time = 0.006767749786376953
Length of Path is  888
Done
```

Large:



```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeDFS.py
Starting
Enter Maze maze-Large.txt
Number of nodes visited is  73982
Run Time = 0.11600470542907715
Length of Path is  73982
Done
```

VLarge:



```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeDFS.py
Starting
Enter Maze maze-VLarge.txt
Number of nodes visited is  389073
Run Time = 0.5612144470214844
Length of Path is  389073
Done
```

As seen above the larger the maze is the longer the runtime and path is. It is worth noting that generally the run time is quite fast but the path gets increasingly less efficient as the size increases. It is also worth noting that with depth first search the order you visit the nodes can affect the efficiency, currently it searches in the order left, right up down however a different order can be used. The order you choose could potentially make the program more or less efficient as the path taken will change.

Question 1.3: Suggest an improved algorithm for this problem.
This question is further divided into parts. This is a guide to help your in the coding process. You do not need
to submitted different versions of the code, but just the last updated version.
1. Briefly outline another algorithm to solve the maze. That decision should be justified by what you expect
to improve in the performance over depth-first search.

When considering an improved algorithm it is first worth considering what DFS currently does. DFS gives us a path through the maze, although the path is not the shortest path, and it calculates this path relatively quickly. Therefore when suggesting I believe it is better to use an algorithm which gives us the most optimal path as that is the area where DFS is most lacking.

The algorithm I have chose for this is the a* algorithm. This is a modified version of dijkstra's algorithm which in of itself is a modified version of breadth first search. The difference the a* algorithm in comparison to dijkstra's is that instead of organizing the priority queue based on the distance from the start node it organizes it based on the distance from the start node plus a heuristic value. The heuristic value is often the distance to the end goal as. For the implementation I am using I'll be using the euclidean distance and the heuristic. The reason you want to pick a heuristic like this is so that the algorithm is generally working towards the end goal.

The reason why this algorithm is good is because breadth first search can be used to give the most optimal path through the maze however, breadth first search is incredibly slow and will struggle to go through the larger mazes. This algorithm on the other hand modifies breadth first search so that it will run quicker when searching a maze but in a way where the path found is still the most optimal path through the maze.

Although this algorithm gives a better path than DFS it will run slower. However I believe this trade off is justified as the run speed of a* is not so slow where it is no longer usable and the path it gives is substantially more efficient than that of DFS.

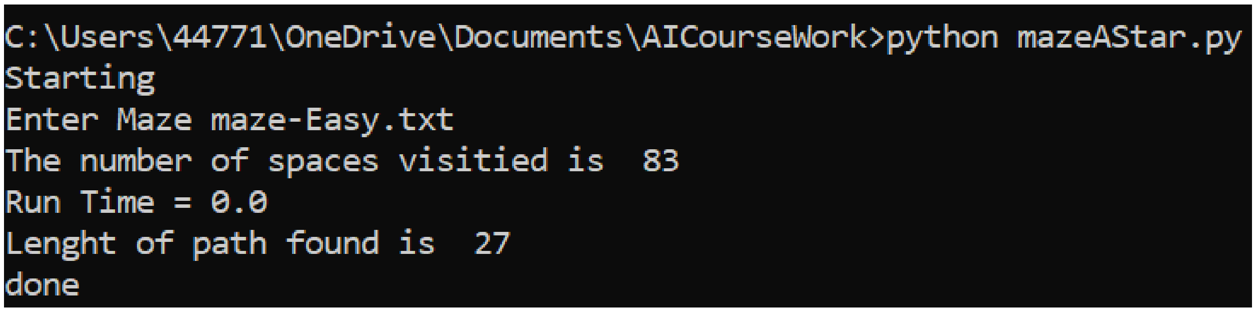
2. Implement the suggested algorithm to solve the maze. It should also calculate the same statistics about
its performance as the previous depth-first algorithm. The solution of the maze should print the locations
in the map that the algorithm will take to reach from the start to the goal.

Code found in the mazeAStar.py file

**3. Run your algorithm to calculate the paths and statistics for all given mazes which are 'maze-Easy.txt"/
'maze-Medium.txt"/ 'maze-Large.txt" / 'maze-VLarge.txt"**

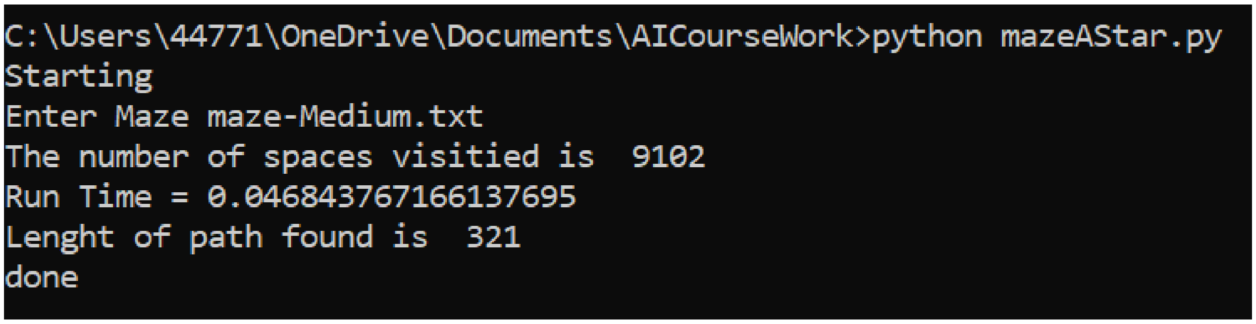
Code found in the mazeAStar.py file
Note: full path of made not shown as its too large, to see the full path uncomment the print statement on line 133 of mazeAStar.py and run the program

Easy:



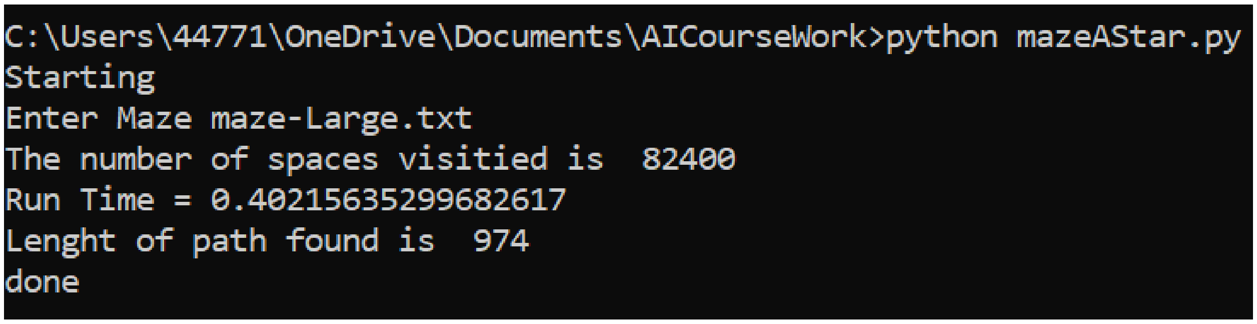
```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeAStar.py
Starting
Enter Maze maze-Easy.txt
The number of spaces visitied is  83
Run Time = 0.0
Lenght of path found is  27
done
```

Medium:



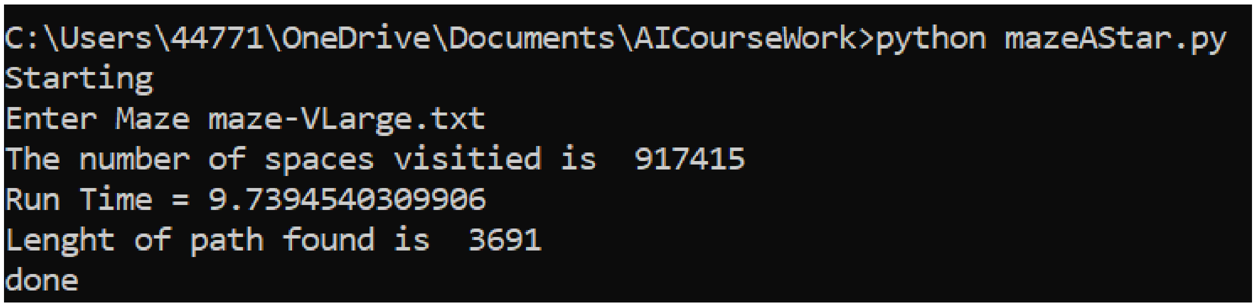
```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeAStar.py
Starting
Enter Maze maze-Medium.txt
The number of spaces visitied is  9102
Run Time = 0.046843767166137695
Lenght of path found is  321
done
```

Large:



```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeAStar.py
Starting
Enter Maze maze-Large.txt
The number of spaces visitied is  82400
Run Time = 0.40215635299682617
Lenght of path found is  974
done
```

VLarge:



```
C:\Users\44771\OneDrive\Documents\AICourseWork>python mazeAStar.py
Starting
Enter Maze maze-VLarge.txt
The number of spaces visitied is  917415
Run Time = 9.7394540309906
Lenght of path found is  3691
done
```

From the results above we can see that firstly the algorithm produces far shorter paths than DFS. This difference grows larger as the mazes grow larger, as such it is almost negligible with maze-Easy.txt but significant with maze-VLarge.txt. It is also worth noting that the algorithm was generally a lot slower than DFS however it was faster for maze-Easy.txt. The reason it was a lot slower was because despite finding the shortest path it visited more spaces in the process which led to the runtime being far greater. The difference wasn't too great for Easy.txt and Medium.txt however for Large.txt it was significant and for VLarge.txt the a* algorithm was far slower.

It should be noted that the run time of the a* algorithm is dependent on how good the heuristic used is. The heuristic I used does help the performance of the algorithm however, it is possible to incorporate a heuristic which is more optimized for the problem at hand.

4. Discuss based on your results, analysis how the suggested algorithm performed better than the depth-first search algorithm.

In comparison to depth first search the a* algorithm produced a better result as the path it produced was more optimal. This is important as most of them when using a searching algorithm it is desirable that the path found is as short as possible. This becomes more significant as the maze gets larger. I believe this to be a significant improvement as the drawback to DFS is the path it finds is very sub optimal.

However, the a* algorithm was noticeably slower than DFS. Despite this I still believe the a* algorithm to be an improvement as it was not so slow to the point of unusable and the decreased speed only became too great for the very large maze, and for mazes of this size the path produced from DFS will be too long to be practical.

Therefore, despite a* running slower than DFS its improvement on the path produced is significant enough that it is worth using over DFS. Also the improved speed of DFS is not noticeable for most sizes of the maze and is only significant for mazes so large that the path found from DFS is too large to be practical.