Devin Suy
017001983

Artificial Intelligence – Assignment 2 Maze Search

Graph Representation:

For this assignment each segment of the maze is represented by a "Cell". Each of which is assigned a number that can be used to locate it, along with a corresponding X and Y coordinate that is utilized in the Manhattan Distance Heuristic approximation.

In the first assignment, relationships between each of our Node Cells was represented using an **adjacency matrix**. A drawback of such implementation lies in the usage of memory that scales quadratically, an issue when dealing with nontrivially large maze representations. Once constructed however, checking any given relationship can be done in constant time.

An alternative approach to representing our graph, utilized in this assignment, is the **adjacency list**. In this method, each Node maintains a list of each Node it has a relationship or "edge" to. Advantages are being able to iterate over edges directly as well as not being as memory intensive, particularly on a sparsely populated graph such as our maze where any given node has at most four edges. This approach is implemented in this assignment using Python Data Dictionary where the assigned cell number is mapped to a reference to the corresponding Cell object in order to reduce the cost of searching for any adjacent Node to an average of constant time.

Implementation

Handling of repeated states is done through the use of a Boolean visited attribute as part of the **Cell** (Vertex) class to avoid the need to maintain an updated visited list in searching. All .lay output writing is done by a generalized write_solution function within the **GameBoard** class that any of the search algorithms can utilize simply by passing a list of **Cells** that comprise the solution path. Each search algorithm is implemented as a separate class that manages its own bookkeeping.

GameBoard class also provides a few utility methods such as maintaining **Cell** attributes like cartesian coordinates and using these values to calculate and assign each cell a Manhattan Distance from the goal cell. This is used as the heuristic for the **A* Search** which maintains cells in a fringe set, choosing among this the cell with the minimum g(n) + h(n) distance on each pass.

Breadth First Search is implemented using a FIFO queue of lists, this list representing a fringe of paths. Each time a descendent node is located that is not in the current path, the path is copied and branched off on each descendent node. Eventually some paths locate the goal node and are retained in a two dimensional list of all successful pathways.

Depth First Search is implemented using a LIFO stack in which descendent nodes marked as unvisited are pushed to the top of the stack continuously until we are unable to locate any more. At this point we pop from the stack and backtrack, eventually locating pathways that reach the goal node which are similarly all stored in a two dimensional list of pathways.

Output – Depth First Search

smallMaze.lay:

```
Solving:  smallMaze.lay
Start @ # 77    Goal @ # 177

DFS First Solution Found!
-------------------------
[77, 78, 79, 101, 123, 124, 125, 126, 148, 149, 150, 128, 129, 130, 152, 174, 196, 195, 194, 193, 192, 191, 190, 189, 167, 166, 165, 164, 186, 185, 184, 183, 182, 181, 180, 179, 178, 177]
Solution Length:  38

(DFS First Solution) Number of Expanded Nodes:  92
(DFS First Solution) Max Fringe Size:  45


DFS Solution # 1
----------------
Solution Length:  38

DFS Solution # 2
----------------
Solution Length:  20

DFS Solution # 3
----------------
Solution Length:  30

DFS Solution # 4
----------------
Solution Length:  50

(DFS Exhaustive) Number of Expanded Nodes:  220
(DFS Exhaustive) Max Fringe Size:  49


All solutions written to : output\DFS\DFS_All_small_Solutions

Optimal DFS solution located, length:  20
Optimal solution written to:  output\DFS \ small_DFS_Optimal_Solution.lay

C:\Users\Devin\Dropbox\Summer '20\CS 451\Assignment\2_Search_Maze>
```

mediumMaze.lay:

```
Solving:  mediumMaze.lay
Start @ # 70    Goal @ # 577

DFS First Solution Found!
-------------------------
[70, 106, 142, 141, 140, 139, 138, 174, 210, 211, 212, 213, 214, 250, 286, 285, 284, 283, 282,
4, 603, 567, 531, 495, 459, 423, 387, 351, 315, 314, 313, 312, 311, 310, 309, 308, 272, 236, 2
, 43, 42, 41, 40, 39, 38, 37, 73, 109, 145, 181, 217, 253, 289, 325, 361, 362, 363, 364, 328,
, 264, 228, 192, 156, 120, 121, 122, 123, 124, 125, 161, 197, 196, 195, 194, 230, 266, 267, 26
510, 511, 512, 476, 477, 478, 479, 480, 481, 445, 409, 410, 411, 412, 413, 414, 415, 379, 380,
1, 580, 579, 578, 577]
Solution Length:  247

(DFS First Solution) Number of Expanded Nodes:  268
(DFS First Solution) Max Fringe Size:  246
```

```
DFS Solution # 52
-----------------
Solution Length:  131

(DFS Exhaustive) Number of Expanded Nodes:  648
(DFS Exhaustive) Max Fringe Size:  248


All solutions written to : output\DFS\DFS_All_medium_Solutions

Optimal DFS solution located, length:  69
Optimal solution written to:  output\DFS \ medium_DFS_Optimal_Solution.lay
```

bigMaze.lay:

```
Solving:  bigMaze.lay
Start @ # 1330    Goal @ # 1296

DFS First Solution Found!
------------------------
[1330, 1293, 1256, 1255, 1254, 1253, 1252, 1215, 1178, 1177, 1176, 1213, 1250, 1249, 1
 866, 867, 868, 869, 870, 871, 872, 909, 946, 947, 948, 911, 874, 875, 876, 877, 878,
29, 366, 365, 364, 363, 362, 399, 436, 473, 510, 547, 584, 583, 582, 619, 656, 693, 73
, 353, 354, 317, 280, 243, 206, 169, 132, 95, 58, 57, 56, 55, 54, 53, 52, 89, 126, 125
 706, 707, 708, 745, 782, 819, 856, 855, 854, 891, 928, 965, 1002, 1003, 1004, 1041, 1
Solution Length:  211

(DFS First Solution) Number of Expanded Nodes:  465
(DFS First Solution) Max Fringe Size:  211


DFS Solution # 1
----------------
Solution Length:  211

(DFS Exhaustive) Number of Expanded Nodes:  1369
(DFS Exhaustive) Max Fringe Size:  222


All solutions written to : output\DFS\DFS_All_big_Solutions

Optimal DFS solution located, length:  211
Optimal solution written to:  output\DFS \ big_DFS_Optimal_Solution.lay
```

Output – Breadth First Search

smallMaze.lay:

```
Solving:  smallMaze.lay
Start @ # 77    Goal @ # 177

BFS First Solution
------------------
Path:  [77, 78, 79, 101, 123, 122, 144, 166, 165, 164, 186, 185, 184, 183, 182, 181, 180, 179, 178, 177]
Solution Length:  20

(BFS First Solution) Number of Expanded Nodes:  95
(BFS First Solution) Max Fringe Size:  8

(BFS Exhaustive) Solution # 1  length:  50
(BFS Exhaustive) Solution # 2  length:  30
(BFS Exhaustive) Solution # 3  length:  20
(BFS Exhaustive) Solution # 4  length:  38

(BFS Exhaustive) Number of Expanded Nodes:  220
(BFS Exhaustive) Max Fringe Size:  7

All solutions written to : output\DFS\DFS_All_small_Solutions

Optimal BFS solution located, length:  20
Optimal solution written to:  output\BFS \ small_BFS_Optimal_Solution.lay
```

mediumMaze.lay:

```
Solving:  mediumMaze.lay
Start @ # 70    Goal @ # 577

BFS First Solution
------------------
Path:  [70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 97, 133, 134, 135, 171, 207, 243, 242, 241, 240, 204, 203
560, 559, 558, 557, 556, 555, 554, 553, 552, 551, 550, 586, 585, 584, 583, 582, 581, 580, 579, 578, 577]
Solution Length:  69

(BFS First Solution) Number of Expanded Nodes:  276
(BFS First Solution) Max Fringe Size:  8

(BFS Exhaustive) Solution # 1  length:  131
(BFS Exhaustive) Solution # 2  length:  131
(BFS Exhaustive) Solution # 3  length:  153
(BFS Exhaustive) Solution # 4  length:  165
(BFS Exhaustive) Solution # 5  length:  163
(BFS Exhaustive) Solution # 6  length:  175
(BFS Exhaustive) Solution # 7  length:  135
(BFS Exhaustive) Solution # 8  length:  135
(BFS Exhaustive) Solution # 9  length:  103
(BFS Exhaustive) Solution # 10  length:  115
(BFS Exhaustive) Solution # 11  length:  111
```

```
(BFS Exhaustive) Solution # 43  length:  237
(BFS Exhaustive) Solution # 44  length:  249
(BFS Exhaustive) Solution # 45  length:  145
(BFS Exhaustive) Solution # 46  length:  157
(BFS Exhaustive) Solution # 47  length:  153
(BFS Exhaustive) Solution # 48  length:  165
(BFS Exhaustive) Solution # 49  length:  225
(BFS Exhaustive) Solution # 50  length:  237
(BFS Exhaustive) Solution # 51  length:  235
(BFS Exhaustive) Solution # 52  length:  247

(BFS Exhaustive) Number of Expanded Nodes:  648
(BFS Exhaustive) Max Fringe Size:  11

All solutions written to : output\DFS\DFS_All_medium_Solutions

Optimal BFS solution located, length:  69
Optimal solution written to:  output\BFS \ medium_BFS_Optimal_Solution.lay
```

bigMaze.lay:

```
Solving:  bigMaze.lay
Start @ # 1330    Goal @ # 1296

BFS First Solution
------------------
Path:  [1330, 1293, 1256, 1255, 1254, 1253, 1252, 1215, 1178, 1177, 1176, 1213, 1250, 1249, 1248, 1247, 12
0, 903, 866, 867, 868, 869, 870, 871, 872, 909, 946, 947, 948, 911, 874, 875, 876, 877, 878, 841, 804, 805
 292, 329, 366, 365, 364, 363, 362, 399, 436, 473, 510, 547, 584, 583, 582, 619, 656, 693, 730, 729, 728,
51, 352, 353, 354, 317, 280, 243, 206, 169, 132, 95, 58, 57, 56, 55, 54, 53, 52, 89, 126, 125, 124, 123, 1
2, 669, 706, 707, 708, 745, 782, 819, 856, 855, 854, 891, 928, 965, 1002, 1003, 1004, 1041, 1078, 1077, 10
Solution Length:  211

(BFS First Solution) Number of Expanded Nodes:  621
(BFS First Solution) Max Fringe Size:  7

(BFS Exhaustive) Solution # 1  length:  211

(BFS Exhaustive) Number of Expanded Nodes:  1369
(BFS Exhaustive) Max Fringe Size:  38

All solutions written to : output\DFS\DFS_All_big_Solutions

Optimal BFS solution located, length:  211
Optimal solution written to:  output\BFS \ big_BFS_Optimal_Solution.lay
```

Output – A* Search

smallMaze.lay:

```
Solving:  smallMaze.lay
Start @ # 77    Goal @ # 177

A* Solution Found!
------------------
Path:  [77, 78, 79, 101, 123, 122, 144, 166, 165, 164, 186, 185, 184, 183, 182, 181, 180, 179, 178, 177]

Solution Length:  20
Number of Expanded Nodes:  40
Max Fringe Size:  6
```

mediumMaze.lay:

```
Solving:  mediumMaze.lay
Start @ # 70    Goal @ # 577

A* Solution Found!
------------------
Path:  [70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 97, 133, 134, 135, 171, 207, 243, 242, 241, 240,
560, 559, 558, 557, 556, 555, 554, 553, 552, 551, 550, 586, 585, 584, 583, 582, 581, 580, 579, 5

Solution Length:  69
Number of Expanded Nodes:  218
Max Fringe Size:  7
```

bigMaze.lay:

```
Solving:  bigMaze.lay
Start @ # 1330    Goal @ # 1296

A* Solution Found!
------------------
Path:  [1330, 1293, 1256, 1255, 1254, 1253, 1252, 1215, 1178, 1177, 1176, 1213,
0, 903, 866, 867, 868, 869, 870, 871, 872, 909, 946, 947, 948, 911, 874, 875, 87
 292, 329, 366, 365, 364, 363, 362, 399, 436, 473, 510, 547, 584, 583, 582, 619,
51, 352, 353, 354, 317, 280, 243, 206, 169, 132, 95, 58, 57, 56, 55, 54, 53, 52,
2, 669, 706, 707, 708, 745, 782, 819, 856, 855, 854, 891, 928, 965, 1002, 1003,

Solution Length:  211
Number of Expanded Nodes:  541
Max Fringe Size:  11
```

Execution

Program is run from main.py that utilizes Python pathlib Path to point directories to the respective input and output folders, the base Path is joined with further directories as necessary.

The input folder simply contains all input .lay maze files read to generate each maze. Each search algorithm outputs the solved .lay maze files into its respective folder within the output folder.

Analysis

Results between the search algorithms can vary depending on the structure of the input maze as well as the placement of the start and end cells, particularly when we observe the first solution returned by each algorithm. This is due to the nature of the traversal order of cells each one explores, DFS effectively continuing along a pathway until eventually reaching a dead end state, only then backtracking to explore other areas. Compare this to BFS which may more slowly move in a given direction, but is iteratively exploring all of the surrounding region as it traverses. A* takes a completely different approach, while its heuristic is only an estimate, it is able to prune the search space by simply disregarding nodes on the fringe that will not move it towards the direction of the goal node.

Unlike A* though, DFS and BFS are exhaustive and are eventually able to retrieve all solution pathways. Among the two though, the first solution returned by DFS is generally further from the root node (starting cell) than the optimal solution or even the first solution reached by BFS due to the nature of traversal order discussed. Given an extremely large or infinite maze to traverse, DFS could potentially traverse downward forever in which case IDDFS might be a better search variant to use, however on a more shallow graph where distance from the root is not too important DFS can yield higher performance and memory on reaching a solution before BFS.