Theoretical Analysis

Adjacency list

method	Best Case	Worst case
UpdateWall()	O(1) This function isn't affected by the size of the maze and will perform the same	O(1) Nothing affects the speed negatively
Neighbours()	O(1) Due to the way adjacency lists work, the neighbours list is already made, just return it	O(1) Nothing affects the speed of the list

Adjacency matrix

method	Best Case	Worst case
UpdateWall()	O(1) This function isn't affected by the size of the maze and will perform the same	O(1) Nothing affects the speed negatively
Neighbours()	O(height x width) As we need to loop through every node in the maze to see if they have an edge with the node being checked	O(height x width) Nothing changes the amount of iterations we need to make so the worst case is the same as the best case

Empirical Analysis

For the empirical analysis I am using 6 differently sized maze configurations to get a large number of plots on the graph to better visualise the different execution times of each of the implementations. Each of the mazes will be generated 3 times and the average of all will be used for the data plot. The data is generated using the existing sample configuration files and can all be found in the dataGen folder of the project

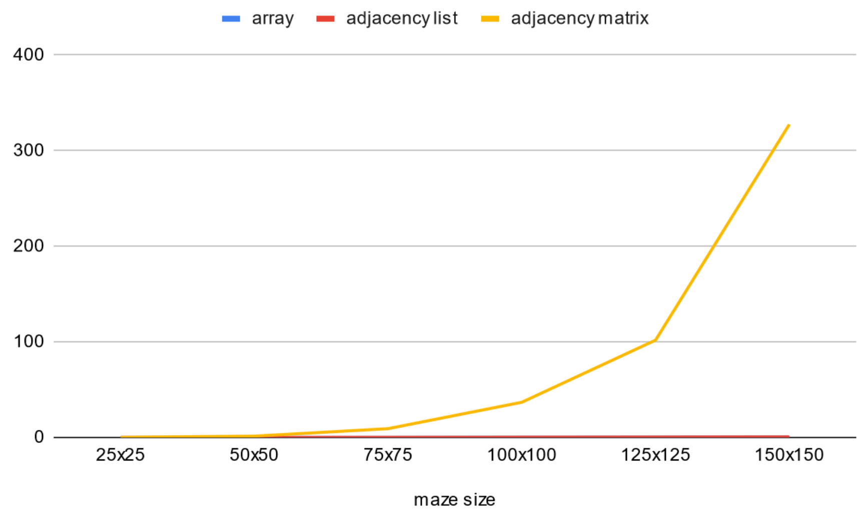
Maze sizes:
25x25
50x50
75x75
100x100
125x125
150x150

Execution results:

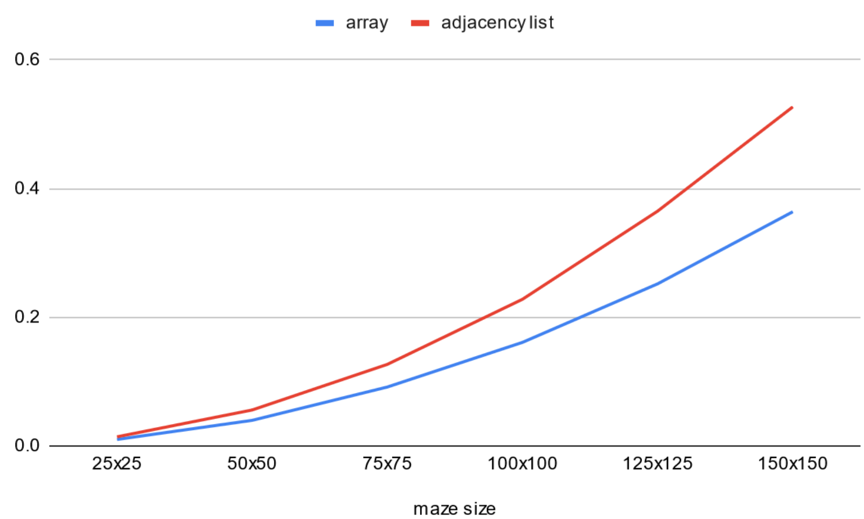
Maze size	arraymaze	Adjacency list	Adjacency matrix
25x25	1. 0.0100 2. 0.0101 3. 0.0101 Avg: 0.0101	1. 0.0145 2. 0.0142 3. 0.0140 Avg 0.0142	1. 0.1000 2. 0.1016 3. 0.1089 Avg 0.1035
50x50	1. 0.0390 2. 0.0402 3. 0.0403 Avg 0.0398	1. 0.0551 2. 0.0569 3. 0.0556 Avg 0.0559	1. 1.3823 2. 1.3738 3. 1.3543 Avg 1.3701
75x75	1. 0.0932 2. 0.0903 3. 0.0914 Avg 0.0916	1. 0.1251 2. 0.1271 3. 0.1283 Avg 0.1268	1. 9.3979 2. 8.8915 3. 9.2540 Avg 9.1811
100x100 (only to 3 decimal places now)	1. 0.162 2. 0.161 3. 0.161 Avg 0.161	1. 0.232 2. 0.228 3. 0.223 Avg 0.228	1. 35.889 2. 37.298 3. 37.278 Avg 36.8217
125x125	1. 0.250 2. 0.254 3. 0.252 Avg 0.252	1. 0.363 2. 0.366 3. 0.365 Avg 0.365	1. 99.400 2. 100.43 3. 103.656 Avg 101.829
150x150	1. 0.363 2. 0.366 3. 0.364 Avg 0.364	1. 0.525 2. 0.532 3. 0.525 Avg 0.527	1. 320.110 2. 341.756 3. 321.789 Avg 327.551

Charts

All data structures included:



Adjacency matrix excluded



Analysis of varying size

So from the results we can clearly see that the adjacency matrix's time increase exponentially and is far slower than the other 2 structure types, this is due to the fact that methods such as addvertex() and neighbours() have a time complexity of O(n) whilst the adjacency list and array have a much faster O(1) for the same task.

Comparing the other 2 data structures we can see that the others also increase exponentially but at a much slower rate than the adjacency matrix. We can also see that array implementation is slightly faster than the adjacency list. But as the maze continues to expand the gap between the 2 is also going to increase exponentially. This is due to the O(n) complexity of the addVerticies of the adjacency list implementation, something that the array implementation does not have.

Summary

After comparing all the data structures for this range of maze sizes I would recommend using either an array or adjacency list implementation as they were significantly faster than the adjacency matrix implementation. Between the array and adjacency list the performance difference is minimal but since the array implementation was faster i would recommend it over the adjacency list