# Path finding algorithms

DFS

Uses a stack (last in first out)

Concept of how it works:

* It explores the nodes by going through to each node until no other nodes can be reached
* Once this happens it backtracks until there is a node that hasn’t been visited and then the process starts again

However, in the concept of our pathfinder, we would need to find all of the possibilities to reach the designated node so that from then we can find out which path is the shortest path

Pros:

* Space efficiency
* Easy to implement

Cons:

* Needs to find all possible routes to govern the shortest path
* Could reach an infinite loop

BFS

Uses a queue

Concept of how it works

* This explores all of the surface nodes before going to the next layer of nodes
* The queue works as nodes are added as they are discovered and then removed once we reach the end of the traversed path

In the concept of the pathfinder, the first path found from the starting node to the end node will always be the shortest path based off the nature of the bfs algorithm

Pros:

* Finds the shortest path
* Easy to implement
* Possibly the fastest runtime to find the shortest path as we don’t have to worry about weights

Cons:

* Space inefficiency
* Could be slow for large graphs (wont have to worry about that for ours)

Dijkstra

This helps finds the shortest path on a **weighted** graph

Concept:

* This can use a queue, priority queue, or a heap queue
  + This is to help find the next node with the smallest known distance
* Need to be aware of the previous node in case the distance increases and is no longer the shortest path

Its kinda similar to bfs

Pros:

* Will guarantee the shortest path whether it’s a weighted graph or unweighted group

Cons:

* Space inefficiency as distances and paths needs to be stored
* Can have slow runtimes for larger mazes (don’t have to worry about ours)
* Cant handle negative weights (don’t have to worry about that)

Astar

Combines the benefits of Dijkstra and greedy best-first search

Concept:

* Has a heuristic function (h)
  + This estimates the cost from a starting node to an end node
  + The heuristic function would include the Manhattan distance and Euclidean distance
  + Uses cost functions
    - G(n): actual cost from the start node to the next node
    - H(n): estimated cost from the next node to the end node (uses the heuristic function)
    - F(n): total estimated cost of the cheapest solution through the next node (calculated as f(n) = g(n) + h(n))
* Uses a priority queue or heap queue

Pros:

* Optimal (heuristic is admissible, never overestimates the cost to reach the goal) and is consistent
* Efficient as it would generally faster than Dijkstra due to the heuristic function

Cons:

* Performance is dependent on the quality of the Heuristic function
* Uses more memory than dfs or bfs
* Hard and complex to implement

Overal 2 choices:

BFS

* Easy to implement
* The first path to find the end point will always be the shortest path
* Don’t need to worry about runtime as the maze is relatively small
* No need to worry about there being any weights on the graph
* Arguably the fastest algorithm

(you can choose)