Here is an algorithm for finding a path from the entrance to the exit of such a maze. This algorithm, like many others, is based on the idea of a *worklist* that holds individual steps towards the solution. For our problem the worklist holds squares of the maze that we know how to reach from the entrance.

We'll actually code two solutions -- one where the worklist is a stack and one were it is a queue. But the algorithm is the same regardless of the structure used for the worklist.

We will start the worklist with the entrance square for the maze as its only element.

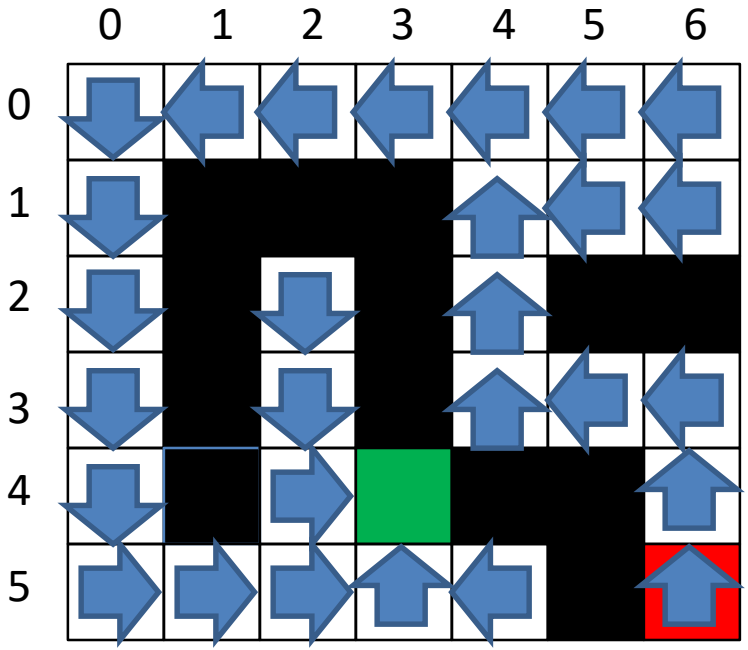
We will also use a *marking* system, marking squares that we have already reached. Each time we take the next element from the worklist, we look to see if it is the maze's exit; if so we are done. If it is not the exit we add all of its unmarked open neighbors (i.e. white squares) to the worklist, marking them as we do so.

You should see that this process will eventually mark every square that can be reached from the start square.

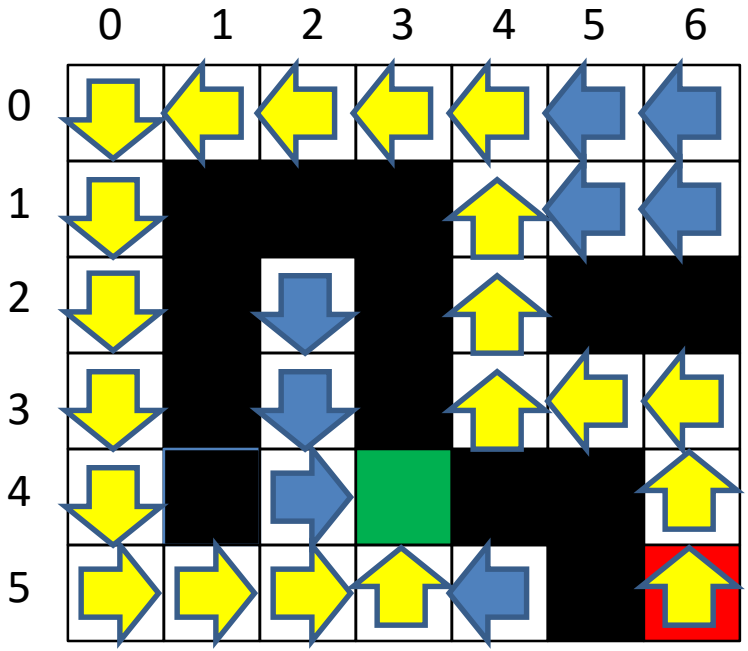
This ends in one of two ways -- either we reach the maze's exit, in which case we have solved the maze, or else we reach a stage where there is nothing left in the worklist, in which case the maze has no solution.

We will also be adding *previous edges* from nodes in the worklist back to the nodes that put them there. Eventually these previous edges will form a path from the goal back to the entrance. When we reverse it this path will be our solution to the maze.

Here is what those previous edges look like: edges from each square back to the square that caused it to be added to the worklist:



Now start at the exit square and follow the path edges back to the start square:



That gives us a path from the exit to the start; if we reverse it we have a path from the start to the exit: a solution to the maze.

Here is the full algorithm:

- Start the worklist with the entrance square
- At each step:
  - If the worklist is empty there is no solution
  - If the worklist isn't empty take node p from it.
  - If p is the exit square stop; you have solved the maze.
  - Otherwise let L be the list of p's neighbors.
  - For each q in L if q is unmarked and not a wall then mark q and add q to the worklist with a previous edge from q back to p.

For example, consider the following maze, where E indicates the entrance and G (for Goal) indicates the exit:

	0	**1**	**2**	**3**	**4**
0			E		
1					
2					
3					G

We will add neighbors to the worklist in the order North, East, South West. We will color the current node Blue, marked nodes Yellow.

First, let's use a queue for the worklist. We start

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (0, 2)

Worklist: empty

Unmarked neighbors of the current node:

(1, 2)

Worklist after unmarked neighbors are added:

(1, 2)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (1, 2)

Worklist: empty

Unmarked neighbors of the current node:

(1, 3), (2,2), (1,1)

Worklist after unmarked neighbors are added:

(1,3), (2,2), (1,1)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (1, 3)

Worklist: (2,2), (1,1)

Unmarked neighbors of the current node:

(1, 4)

Worklist after unmarked neighbors are added:

(2,2), (1,1), (1,4)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (2, 2)

Worklist: (1,1), (1,4)

Unmarked neighbors of the current node:

(3, 2), (2, 1)

Worklist after unmarked neighbors are added:

(1,1), (1,4), (3, 2), (2, 1)

The worklist as a queue:

Current node: (1, 1)

Worklist: (1,4), (3, 2), (2, 1)

Unmarked neighbors of the current node:

(1, 0)

Worklist after unmarked neighbors are added:

(1,4), (3, 2), (2, 1), (1, 0)

The worklist as a queue:

Current node: (1, 4)

Worklist: (3, 2), (2, 1), (1, 0)

Unmarked neighbors of the current node:

(0, 4), (2, 4)

Worklist after unmarked neighbors are added:

(3, 2), (2, 1), (1, 0), (0, 4), (2, 4)

The worklist as a queue:

Current node: (3, 2)

Worklist: (2, 1), (1, 0), (0, 4), (2, 4)

Unmarked neighbors of the current node:

(3, 1)

Worklist after unmarked neighbors are added:

(2, 1), (1, 0), (0, 4), (2, 4), (3, 1)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (2, 1)

Worklist: (1, 0), (0, 4), (2, 4), (3, 1)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(1, 0), (0, 4), (2, 4), (3, 1)

The worklist as a queue:

Current node: (1, 0)

Worklist: (0, 4), (2, 4), (3, 1)

Unmarked neighbors of the current node:

(0, 0)

Worklist after unmarked neighbors are added:

(0, 4), (2, 4), (3, 1), (0, 0)

The worklist as a queue:

Current node: (0, 4)

Worklist: (2, 4), (3, 1), (0, 0)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(2, 4), (3, 1), (0, 0)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (2, 4)

Worklist: (3, 1), (0, 0)

Unmarked neighbors of the current node:

(3, 4)

Worklist after unmarked neighbors are added:

(3, 1), (0, 0), (3, 4)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (3, 1)

Worklist: (0, 0), (3, 4)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(0, 0), (3, 4)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (0, 0)

Worklist: (3, 4)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(3, 4)

The worklist as a queue:

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (3, 4)

This is the goal so we have found a solution

If we trace back through our steps, we see that

(3, 4) was added by (2, 4)

(2, 4) was added by (1, 4)

(1, 4) was added by (1, 3)

(1, 3) was added by (1, 2)

(1, 2) was added by (0, 2)

(0, 2) was the entrance

	0	1	2	3	4
0			E		
1					
2					
3					G

Do it again, with a stack for the worklist. We start

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (0, 2)

Worklist: empty

Unmarked neighbors of the current node:

(1, 2)

Worklist after unmarked neighbors are added:

(1, 2)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (1, 2)

Worklist: empty

Unmarked neighbors of the current node:

(1, 3), (2, 2), (1, 1)

Worklist after unmarked neighbors are added:

(1, 3), (2, 2), (1, 1)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (1, 1)

Worklist: (1, 3), (2, 2)

Unmarked neighbors of the current node:

(2, 1), (1, 0)

Worklist after unmarked neighbors are added:

(1, 3), (2, 2), (2, 1), (1, 0)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (1, 0)

Worklist: (1, 3), (2, 2), (2, 1)

Unmarked neighbors of the current node:

(0, 0)

Worklist after unmarked neighbors are added:

(1, 3), (2, 2), (2, 1), (0, 0)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (0, 0)

Worklist: (1, 3), (2, 2), (2, 1)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(1, 3), (2, 2), (2, 1)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (2, 1)

Worklist: (1, 3), (2, 2)

Unmarked neighbors of the current node:

(3, 1)

Worklist after unmarked neighbors are added:

(1, 3), (2, 2), (3, 1)

Stack for the worklist (growing to the right).

Current node: (3, 1)

Worklist: (1, 3), (2, 2)

Unmarked neighbors of the current node:

(3, 2)

Worklist after unmarked neighbors are added:

(1, 3), (2, 2), (3, 2)

Stack for the worklist (growing to the right).

Current node: (3, 2)

Worklist: (1, 3), (2, 2)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(1, 3), (2, 2)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (2, 2)

Worklist: (1, 3)

Unmarked neighbors of the current node:

none

Worklist after unmarked neighbors are added:

(1, 3)

Stack for the worklist (growing to the right).

Current node: (1, 3)

Worklist: empty

Unmarked neighbors of the current node:

(1, 4)

Worklist after unmarked neighbors are added:

(1,4)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (1, 4)

Worklist: empty

Unmarked neighbors of the current node:

(0, 4), (2, 4)

Worklist after unmarked neighbors are added:

(0, 4), (2, 4)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (2, 4)

Worklist: (0, 4)

Unmarked neighbors of the current node:

(3, 4)

Worklist after unmarked neighbors are added:

(0, 4), (3, 4)

Stack for the worklist (growing to the right).

	0	1	2	3	4
0			E		
1					
2					
3					G

Current node: (3, 4)

Again, that is our goal.

Again, we find the actual path by tracing back from the goal to the entrance:

(3, 4) was added by (2, 4)
(2, 4) was added by (1, 4)
(1, 4) was added by (1, 3)
(1, 3) was added by (1, 2)
(1, 2) was added by (0, 2) which is the entrance

	0	1	2	3	4
0			E		
1					
2					
3					G

Following the path edges we get the path from the exit to the entrance:

(3, 4)-> (2, 4)-> (1, 4)-> (1, 3)-> (1, 2)-> (0, 2)

(0, 2)-> (1, 2)-> (1, 3)-> (1, 4)-> (2, 4)-> (3, 4)

Reversing this gives a path from the entrance to the exit, which is the solution to the maze that we seek.

Note that a solution to the maze is a path -- a sequence of squares from the entrance to the exit where each square is a neighbor of the previous one. Our algorithm will generate such a path if there is one, regardless of whether we use a stack or a queue for the worklist. Changing the data structure changes the order in which we add nodes to the worklist, but either structure will eventually get us a path if there is one.