

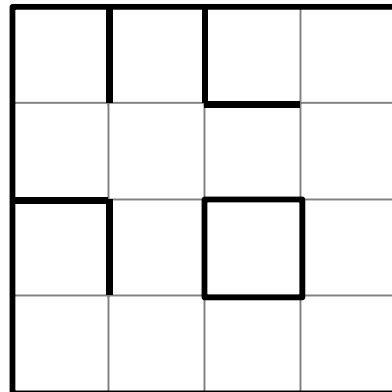
- Problem Statement
- Basic Graph Theory
- Depth First Search
- Breadth First Search
- Building the Map
- Traversing the Map
- Shortest Path in the Map

- Develop a robot that can navigate and learn a maze
 - Use only the parts from the Mindstorms Kit (plus micromotors and rotation sensors)
 - Building a maze around a small RCX-based robot would be difficult
 - Navigating the robot is subject to small incremental errors which sum to large positional errors
 - Using the Mindstoms software to optimize the path through the maze is almost impossible
-
- Use pbForth to get access to more memory in the RCX
 - Eliminate the incremental errors with wheel based robots
 - Develop a gantry system with the light sensor as the robot and a sheet of paper as the maze
 - Use fixed racks to move the “robot”
 - Use rubber tracks to move the “maze”
 - Only allow the maze walls to be drawn on predetermined lines.

- Robert Sedgewick's *Algorithms in C* is a classic reference work
- Each square in the maze is called a node
- A node may have 4 paths to its neighbors
- The path from one node to the next is a vertex
- The MazeWalker has three things it needs to do
 1. Find all of the paths from a node to its neighbours
 2. Optimize the path between arbitrary nodes
 3. Traverse the path between nodes
- Depending on the algorithm used, you can get around the maze in different ways

- To implement a non-recursive depth first search, we need a “stack” to keep track of nodes we have visited but not explored
- We also need an array to help keep track of the order in which nodes have been visited
- Here is the procedure for doing a depth first traversal of a maze
 - Push the starting node on the stack
 - While the stack is not empty
 - Pop a node off the stack
 - Mark it as visited
 - Push all unvisited nodes attached to this node on the stack
- When the stack is empty, each node will have been visited
- Try it yourself on the worksheet on the next page...

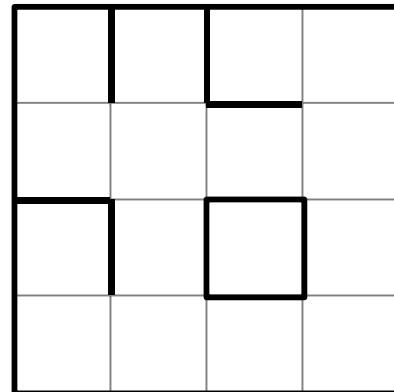
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



- The maze on the left has the nodes numbered. The maze on the right is for you to fill in the index of the square as they are visited.
- Check the connections of each square and push the visited nodes in the order Left, Up, Down, Right

- To implement a breadth first search, we need a “queue” to keep track of nodes we have visited but not explored
- We also need an array to help keep track of the order in which nodes have been visited
- Here is the procedure for doing a breadth first traversal of a maze
 - Put the starting node in the queue
 - While the queue is not empty
 - Get a node from the queue
 - Mark it as visited
 - Put all unvisited nodes attached to this node in the queue
- When the queue is empty, each node will have been visited
- Try it yourself on the worksheet on the next page...

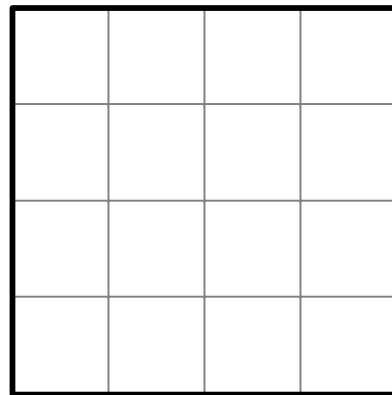
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



- The maze on the left has the nodes numbered. The maze on the right is for you to fill in the index of the square as they are visited.
- Check the connections of each square and push the visited nodes in the order Left, Up, Down, Right

- Given a map of the maze as above, it's easy to figure out a path through it
- It is only a little more difficult to figure out how to actually map the maze given some information about the boundaries
- The work area below represents the map as the MazeWalker finds it at the beginning
- Using the known map on the left, fill in the boundaries as you discover them in a depth-first traversal of the maze

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



- The last problem is how to figure out the shortest path between two nodes in the map
- Given that the map has been made, we can follow a simple addition to the breadth first search mechanism

Put the ending node in the queue

Mark its distance as 1

While the queue is not empty

 Get a node from the queue

 Mark it as visited

 Put all unvisited nodes attached to this node in the queue

 Set the distance of these nodes to current distance plus 1

- Try it yourself with some random node pairs in the pictures below

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

