1. Corn mazes: real-life setup where humans can make choices at different junctures.
   1. AbcNEWS: 9/11 calls from people at corn mazes.
   2. Akin to solving problems with algorithms and searching in search spaces.
2. **Breadth first search (BFS)**
   1. Setup
      1. Road map
      2. Starting location
      3. Desired destination
      4. DX/DY array
   2. Goal: travel to each reachable location from starting location, marking where we have been.
3. Key issues
   1. How to determine if we have been somewhere before?
      1. Array marking as true
      2. Queue
   2. How to determine which roads to travel from?
4. **Queue**: abstract linear data structure.
   1. Rules
      1. First in, first out
   2. Supported operations
      1. **Enqueue**: when you get to the line, you go to the back
      2. **Dequeue**: the person to check out is the one in front
   3. Role in a BFS
      1. Keep track of locations from which we explore.
         1. Ordered list (queue) of places visited
      2. The problem can be floodfilled to mark reachable spots, but…
         1. Floodfill marks all possible locations
            1. Overflows with a large enough grid
         2. Queues also give the shortest distance to the location.
      3. Shortest Distances
         1. Steps
            1. A counter of steps starts at 0.
            2. Nodes visited in order.
            3. Each node enqueues adjacent spots that are valid and have not yet been visited.
            4. The node is dequeued and marked as visited by the number of steps to get to it.
   4. 2 Implementations
      1. Linked List: linked list with pointer to front and pointer to back.
         1. Code
            1. struct queue {

struct node\* front;

struct node\* back;

* + - * 1. };
      1. Enqueue (O(*n*)): insert to back for *n* element
         1. myQ->back->front is the front node
         2. Code

myQ->back->next = tmp;

myQ->back = tmp;

* + - 1. Dequeue (O(1)):
         1. Non-null case

int retval = myQ->front->data;

node\* freenode = myQ->front;

myQ->front = myQ->front->next;

free(freenode);

return reval;

* + 1. Array
       1. Why not realloc()?
          1. Takes time to rearrange memory.

Better to set up statically to save time.

* + - * 1. More complicated.

Half the size of the queue when less than ¼ full.

Still constant runtime.

* + - 1. Enqueue (O(1)): in order
      2. Dequeue (O(*n*)): everyone skips forward
         1. Code

//Move pointers forward and back

back = (back + 1) % size;

front = (back + 1) % size;

* + - 1. 2 solutions to problems
         1. Empty queue

If empty, set front and back = -1 as sentinel values.

Store front and size, not front and back.

Enqueue goes to index (front + size) % arraysize.

* + - * 1. Full queue

realloc() or temporary malloc()

Move front to 0

Copy accordingly

size++;