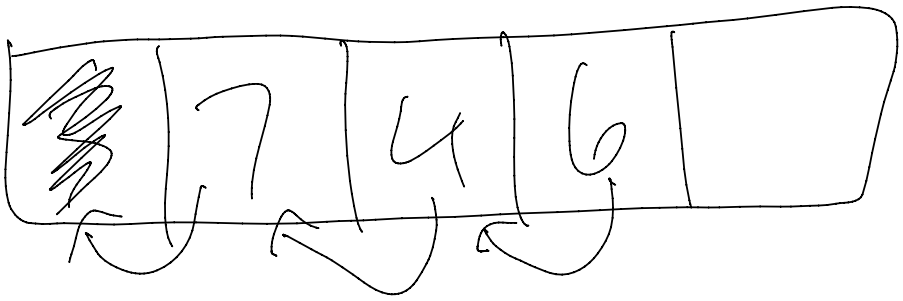
6/1/2020 Lecture - Queues

Monday, June 1, 2020 4:22 PM



enqueue(3)
enqueue(7)
enqueue(4)
enqueue(6)
x = dequeue()

↳ having everyone step up is slow in code - O(n)

= # people in line
steps.

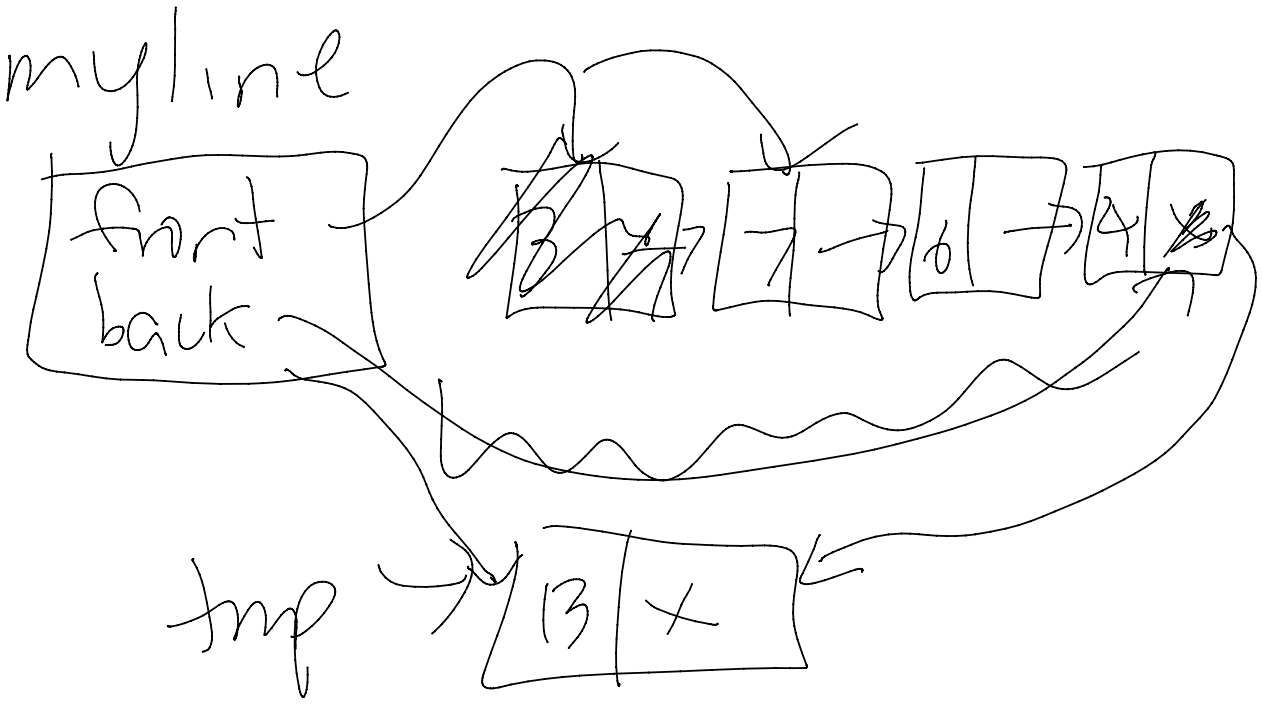
dequeue front = O(1)

enqueue back = O(n)

```
struct node {
    int data; // or whatever
    struct node* next;
};
```

```
struct queue {
    struct node* front;
    struct node* back;
};
```

```
queue myline = NULL;
// Enequeue people…
```



By storing a pointer to both the front of the linked list and the back, we can get O(1) run times for both enqueue and dequeue.

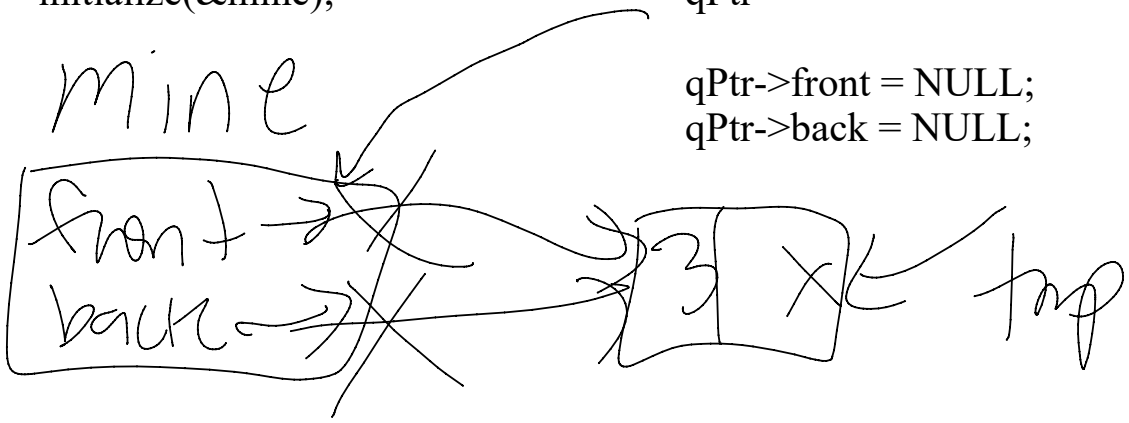
Other potential operations for a queue:

1) size (what is the size of the queue)
2) empty (tells us if the queue is empty or not)
3) front (returns the front of the queue without removing it)

main
struct queue mine;
initialize(&mine);

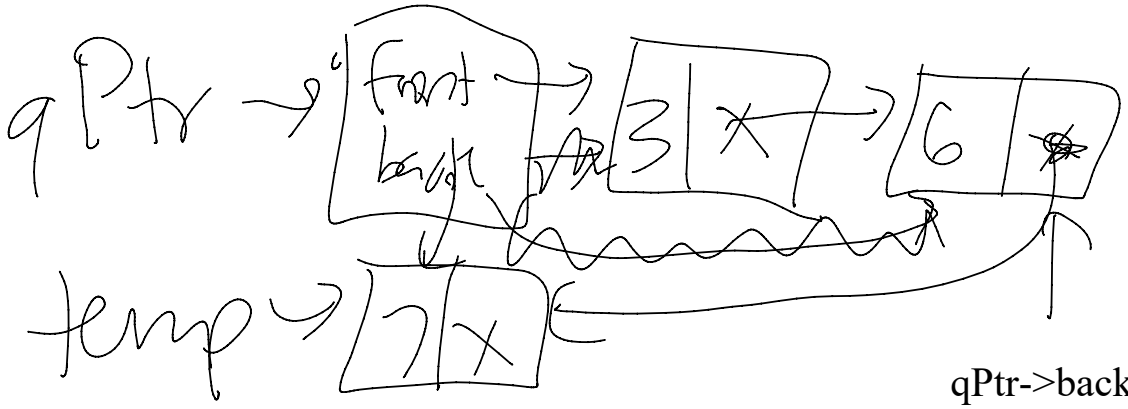
init
---
qPtr

qPtr->front = NULL;
qPtr->back = NULL;



Enqueue
---

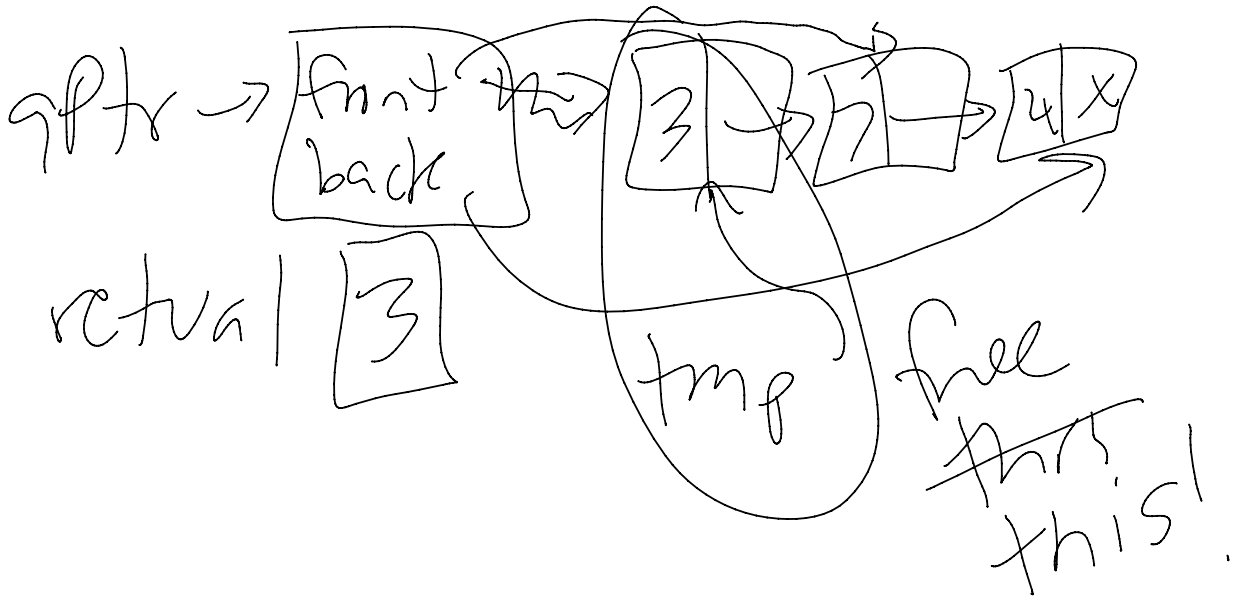
1. Create a new node with the number enqueue, return 0 if malloc failed.
2. Two cases:
   a. queue is empty
      i. set front to new node
      ii. set back to new node
   b. queue is NOT empty
      i. attach back to new node
      ii. reset back to the new back of the list



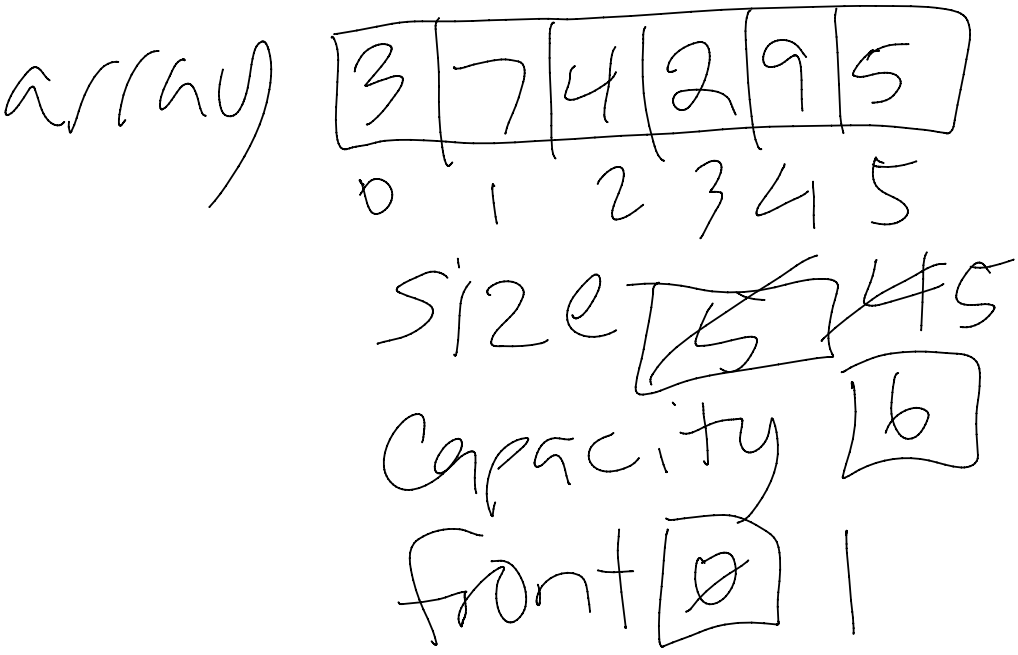
qPtr->back->next = temp

Dequeue
---

1. If empty, I return -1 to signify an empty queue.
2. Save a ptr to the second node.
3. Save the value in the first node.
4. Free the first node.
5. Return the appropriate value.
6. If this empties the list, set both front and back to NULL.



Queue: Array Implementation (need O(1) enqueue, O(1) dequeue)



Dequeue
---
roughly, we just do
front++;
size--;

Enqueue
---
roughly we do
array[?] = newval
size++;

This idea is pretty good, but it looks like we could fall off the array!!!

Is there anything useful being stored in index 0? --- No, it's been dequeued.

So, instead of falling off of the array - where should we go?
BACK TO INDEX 0.

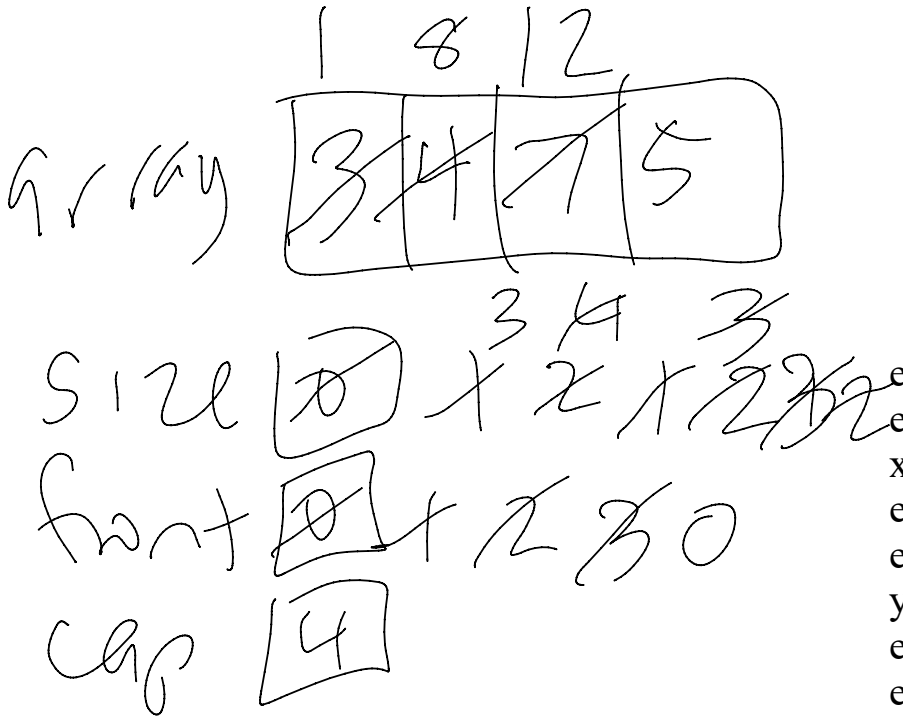
Mod lets me do this very easily!!!

Dequeue logic:

front = (front+1)%capacity;

Enqueue logic:

array[(front+size)%capacity] = enqueueValue
Note: size does NOT get modded.



enqueue(3)
enqueue(4)
x = dequeue()
enequeue(7)
enqueue(5)
y = dequeue()
enqueue(1)
enqueue(8)
z = dequeue()
enqueue(12)
w = dequeue()

Realloc case

Realloc case

front 2 q: 2, 6, 3, 7

size 4

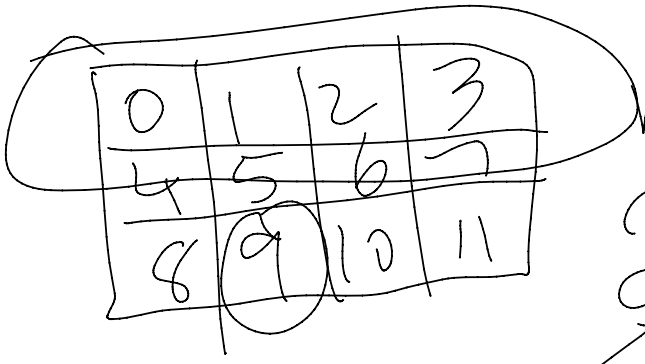
cap 8

enqueue 1

new q: 2, 6, 3, 7, 1

maze location storage idea:

row i, col j will be stored as numC*i + j



row 2 col 1

2 × 4 + 1

row # col size col #

9 / 4 = 2 row

9 % 4 = 1 col

4104=101