ADTs?

Depth-first search : application of Stack

List

Stack

Queue : ADT

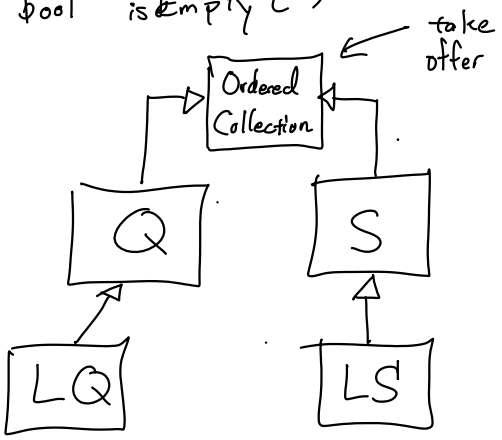
T peek() —— who's next?

T dequeue() —— remove least recent item

void enqueue (T element) — add item

int getSize()

bool isEmpty()




```
class LinkedQueue : public Queue<T> {
  .
  .
  .
private
  LinkedList<T> list;
```

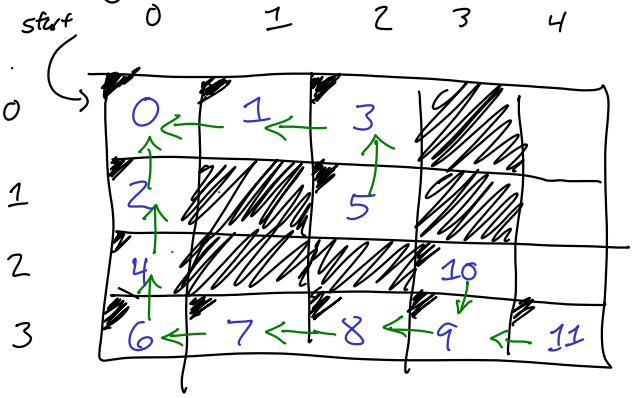
Queues

	LL w/o tail (front ← back)	LL w/o tail (→)	LL w/ tail (←)	LL w/ tail (→)	AL (← amort.)	CircularArrayList (← amort.)
enqueue	iAT: O(n)	iAH: O(1)	iAT: O(1)	iAH: O(1)	iAT: O(1)	iAT: O(1)
dequeue	rFH: O(1)	rFT: O(n)	rFH: O(1)	rFT: O(n)	rFH: O(n)	rFH: O(1)

Search

Maintain frontier
Remember every place I've seen
For each location: remember how you got there (previous location)

```
Function Search():
    frontier ← new Queue
    frontier.enqueue(start)
    Mark start as seen
    While frontier is not empty:
        pos ← frontier.dequeue()
        If pos is my goal:
            cursor ← pos
            sol ← new List
            While cursor ≠ start
                sol.insertAtHead(cursor)
                cursor ← cursor.prev
            sol.insertAtHead(start)
            return sol
        For each neighbor of pos:
            If neighbor not been seen:
                Mark neighbor as seen
                frontier.enqueue(neighbor)
                neighbor.previous ← pos
```



Queue Frontier

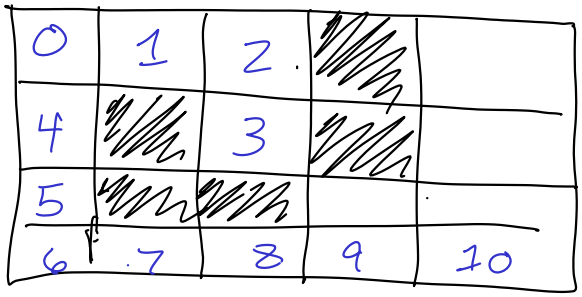
✓

4, 2

Breadth-First Search

Always explore closer things first
Always gives the shortest solution

Stack Frontier



Depth-First Search

Tends to use less memory