CSCI 200: Foundational Programming Concepts & Design Lecture 40

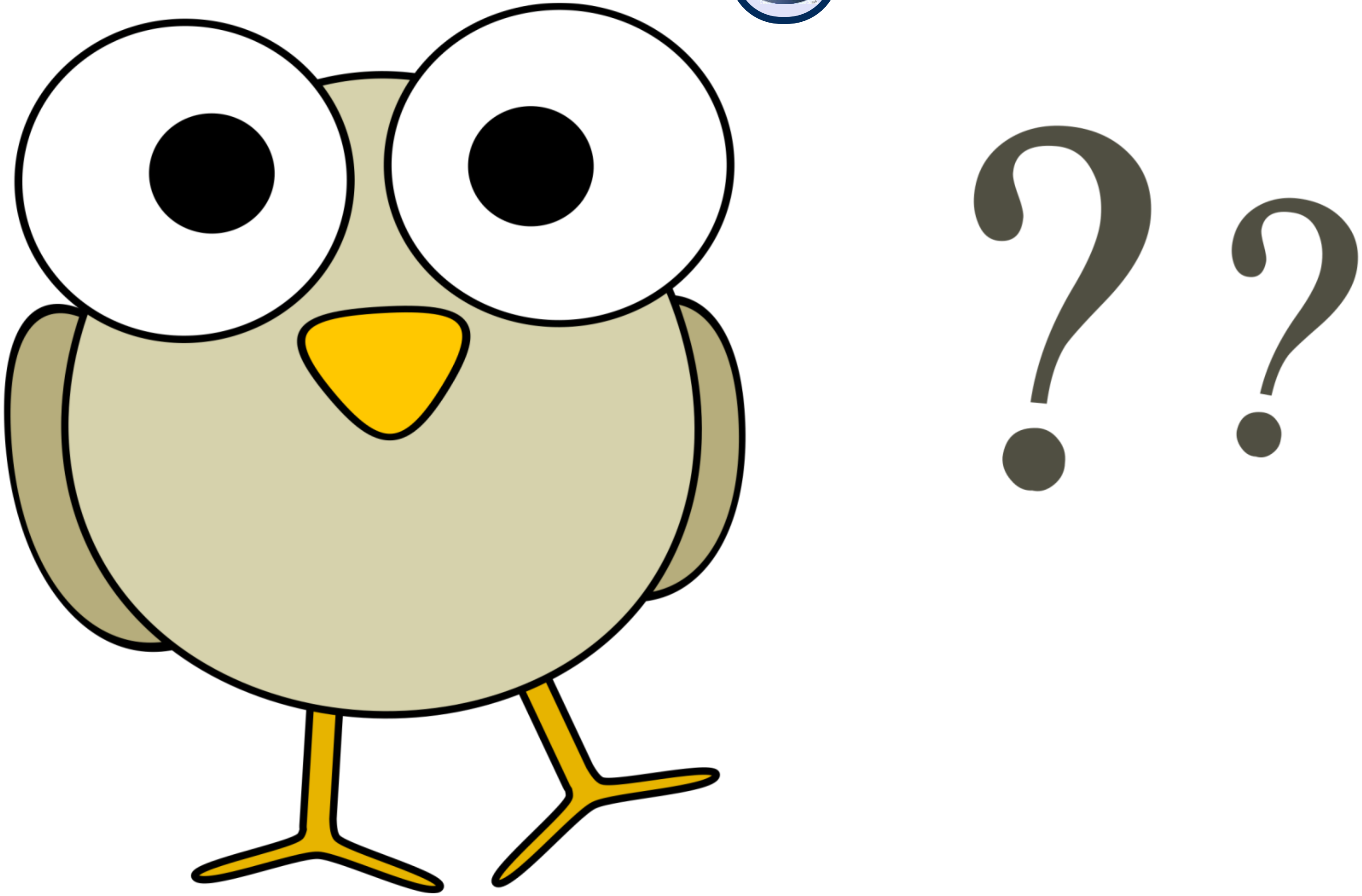
Multidimensional Lists

Stack & Queue

Previously in CSCI 200

- Multidimensional Lists
  - List of Lists of (Lists of Lists of …) object type

Questions?



Learning Outcomes For Today

- Explain the uses of list, stack, and queue data structures. Implement each.
- Implement BFS and DFS. Explain the uses of a queue and stack in each process.
- Explain the uses of list, stack, and queue data structures. Implement each.

On Tap For Today

- Searching A Grid
  - BFS or DFS
  - Data Structures: List, Stack, Queue
- Practice

On Tap For Today

- Searching A Grid
  - BFS or DFS
  - Data Structures: List, Stack, Queue
- Practice

Multidimensional Searches

- Breadth-First Search (BFS)
  - Search all neighbors first, then search neighbors of neighbors, and so forth
- Depth-First Search (DFS)
  - Search one direction first, then backtrack and search a different direction, and so forth

Example BFS Search Ordering

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

Example BFS Search Ordering

0	1	3	6	10
2	4	7	11	14
5	8	12	15	17
9	13	16	18	19

Example DFS Search Ordering

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

Example DFS Search Ordering

0	17	16	11	10
1	18	15	12	9
2	19	14	13	8
3	4	5	6	7

Multidimensional Search Pseudocode

```
create list of positions to check

initial list is start node position
mark start node as visited

while there are still nodes to check
        get current node to check
        check if current node is target
        if yes, found!
        if no,
                for each neighbor
                        if neighbor exists and is unvisited
                                add neighbor to list to check
                                mark neighbor as visited
```

Two Questions

1. How to mark node as visited?
2. How to store and process nodes to visit?

1. Tracking Visited Nodes

- Create a second multidimensional list of Booleans

  if **`booleanTable[i][j]`** == **`true`**

  then **`dataTable[i][j]`** has been visited

How To Determine Next Node?

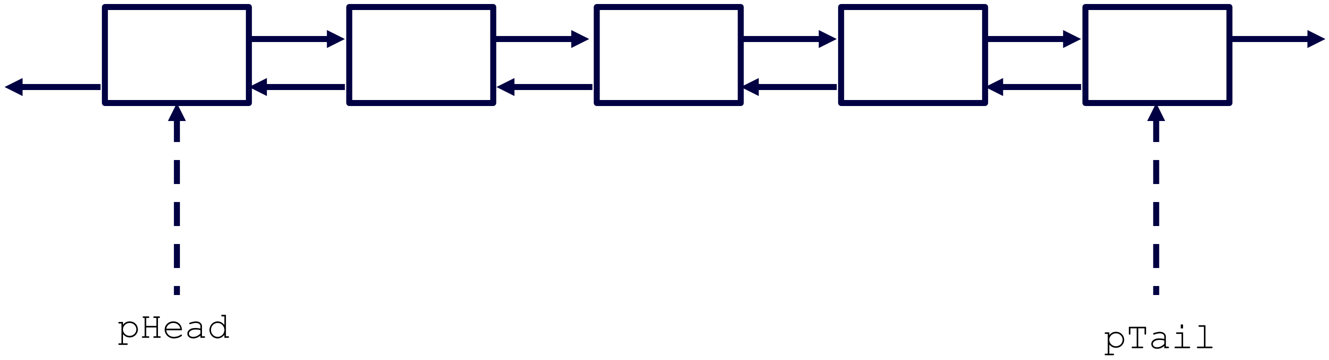
On Tap For Today

- Searching A Grid
  - BFS or DFS
  - Data Structures: List, Stack, Queue

- Practice

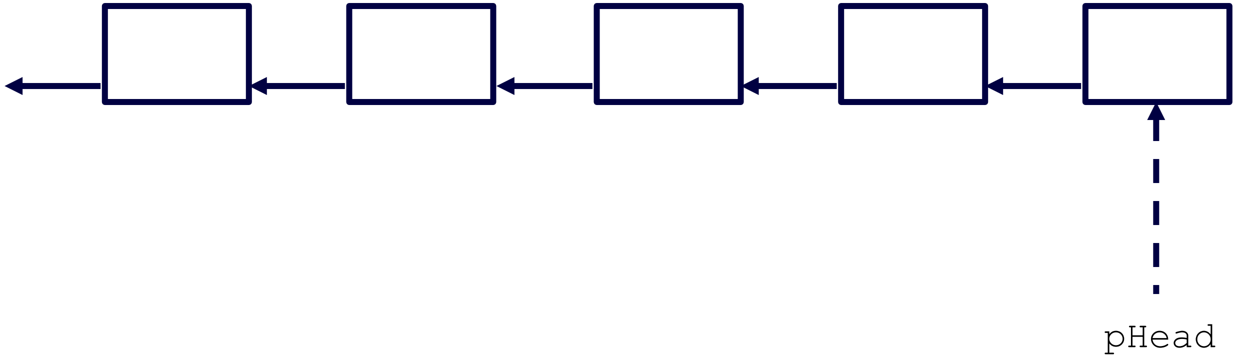
List Operations

Data Structure	push Front()	pop Front()	push Back( )	pop Back( )	insert Middle()	remove Middle()	traverse Forward( )	traverse Backward( )
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓



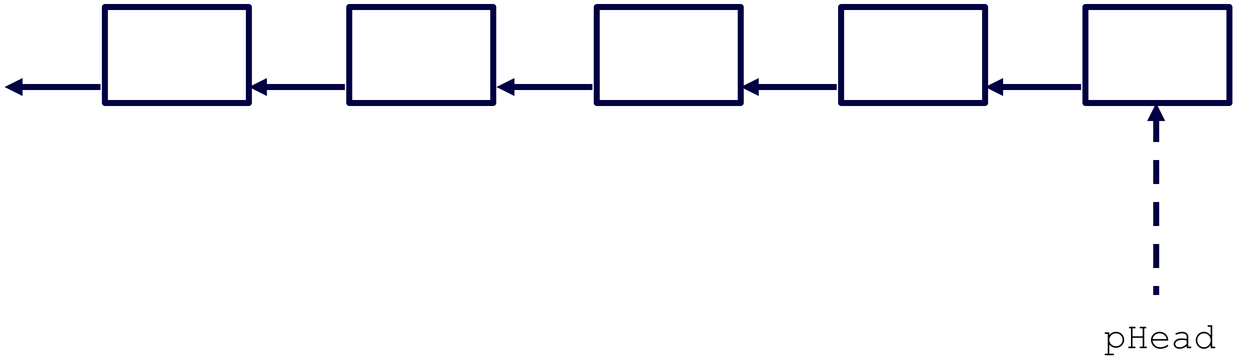
List Operations

Data Structure	push Front()	pop Front()	push Back( )	pop Back( )	insert Middle()	remove Middle()	traverse Forward( )	traverse Backward( )
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)								



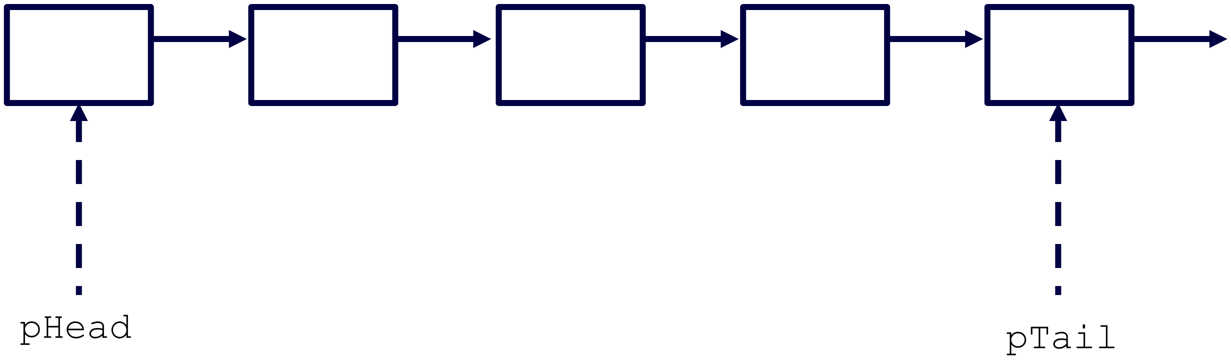
List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)			✓	✓				✓



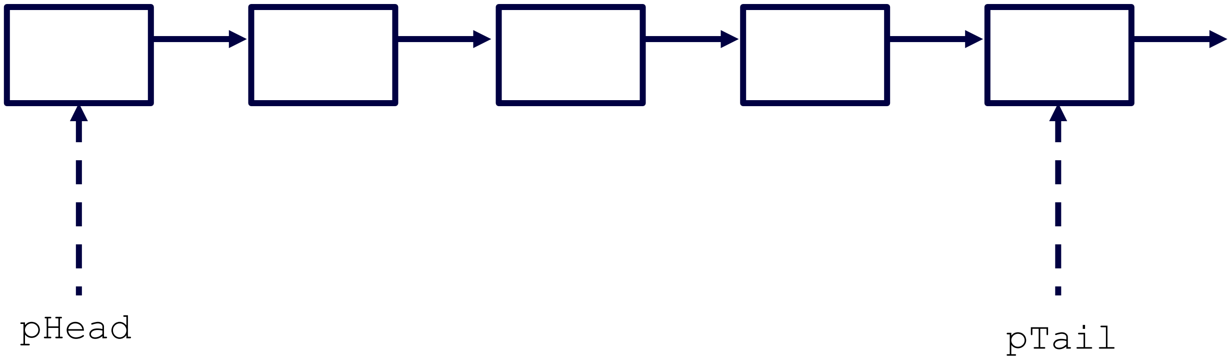
List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)			✓	✓				✓
Queue (FIFO)								



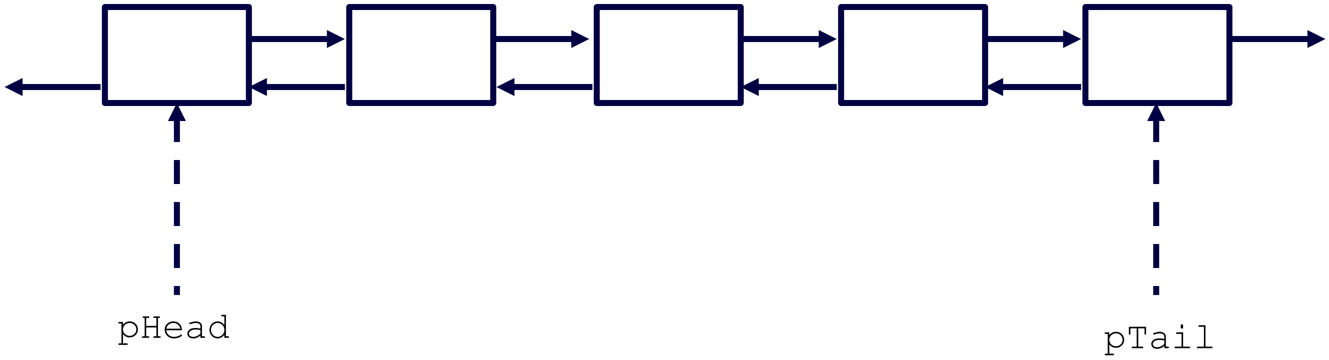
List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)			✓	✓				✓
Queue (FIFO)		✓	✓				✓	



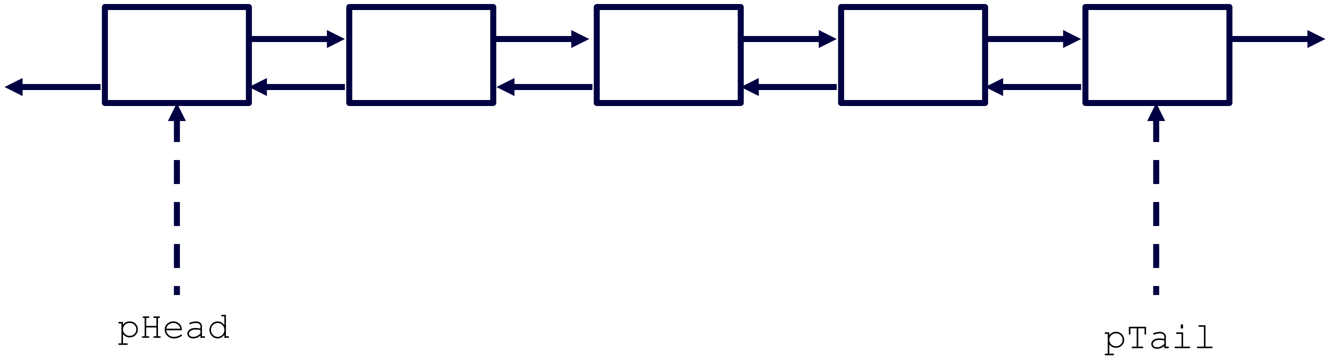
List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)			✓	✓				✓
Queue (FIFO)		✓	✓				✓	
Deque								



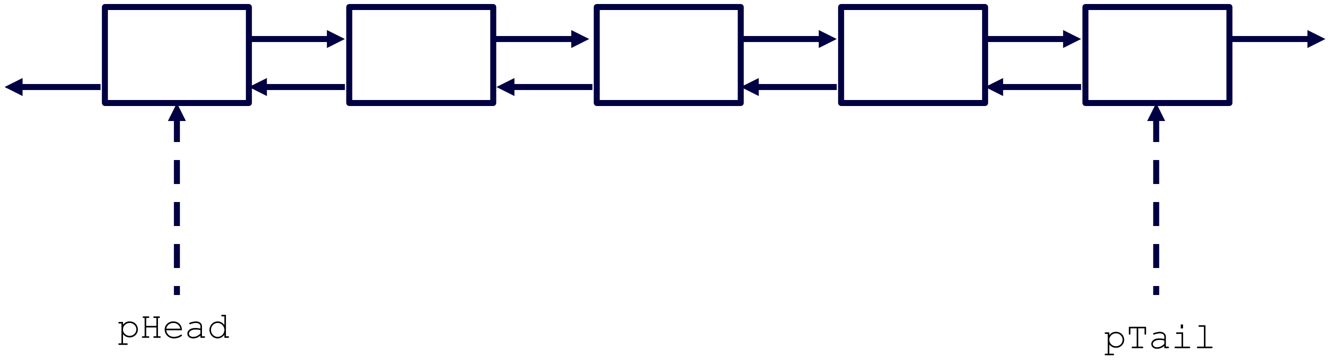
List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)			✓	✓				✓
Queue (FIFO)		✓	✓				✓	
Deque	✓	✓	✓	✓			✓	✓



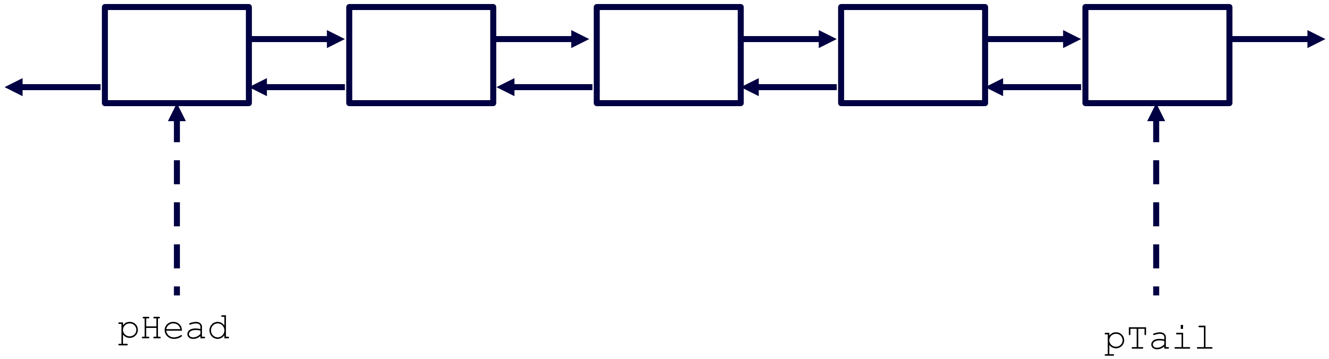
List Operation Costs O(?)

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	1	1	1	n	n	n	n	n^2
Doubly Linked List	1	1	1	1	n	n	n	n
Stack (LIFO)			1	1				n
Queue (FIFO)		1	1				n	
Deque	1	1	1	1			n	n



List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Stack (LIFO)			✓	✓				✓
Queue (FIFO)		✓	✓				✓	
Deque	✓	✓	✓	✓			✓	✓



On Tap For Today

- Searching A Grid
  - BFS or DFS
  - Data Structures: List, Stack, Queue

- Practice

2. Store/Process Nodes

- Add neighbors to a list
- Each time checking a node, remove a node from the list of nodes to check

- BFS - use a queue
- DFS - use a stack

Multidimensional Search Pseudocode

```
create list of positions to check

initial list is start node position
mark start node as visited

while there are still nodes to check
        get current node to check
        check if current node is target
        if yes, found!
        if no,
                for each neighbor
                        if neighbor exists and is unvisited
                                add neighbor to list to check
                                mark neighbor as visited
```

Multidimensional Search Pseudocode

```
struct Position { int r, c; }
List<Position> positionsToCheck;
positionsToCheck.push( Position(0,0) );

while( positionsToCheck.isNotEmpty() ) {
        Position currPos = positionsToCheck.pop();
        if(maze.at(currPos.r).at(currPos.c) == target)
          return currPos;
        else {
          // need to check if exists AND unvisited
          positionsToCheck.push(Position(currPos.r-1, currPos.c  ));
          positionsToCheck.push(Position(currPos.r,   currPos.c-1));
          positionsToCheck.push(Position(currPos.r+1, currPos.c  ));
          positionsToCheck.push(Position(currPos.r,   currPos.c+1));
       }
}
```

Algorithm Complexities

Algorithm	Worst Case	Best Case	Average Case
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Algorithm	Worst Case	Best Case	Average Case
Linear Search	$O(n)$	$O(1)$	$O(n)$
Binary Search	$O(\log n)$	$O(1)$	$O(\log n)$
Breadth-First Search[1]	$O(n^2)$	$O(1)$	$O(n^2)$
Depth-First Search[1]	$O(n^2)$	$O(1)$	$O(n^2)$

[1] BFS and DFS fall under “graph algorithms” so actual complexity is $O(|V| + |E|)$. For our case $|V| = n^2$ and $|E| = 2n^2$

On Tap For Today

- Searching A Grid
  - BFS or DFS
  - Data Structures: List, Stack, Queue

- Practice

To Do For Next Time

- Rest of semester
  - M 12/08: Trees & Graphs, L6B due, Quiz 6
  - W 12/10: Exam Review, L6C due, Exam XC due
  - R 12/11: A6, AXC, Final Project due
  - M 12/15 8am - 10am: Final Exam