Plan for Week

- **Linear structures: Stack, Queue**
  - Applications: LIFO, FIFO, other
  - Canonical examples and view to more
  - Java specifics of how to use these

- **Friday: More examples of recursion and alternatives**
  - Blob counting and applications of neighbor-search

- **Guide to using practice-it for APT-like problems**
What is a linear structure?

- Structure elements have predecessors, successors
- Arrays, Linked-lists, ArrayLists
  - ADT: insert, delete, iterate/traverse, grow, ...
  - Advantages and trade-offs include ...

- Some linear structures are used to solve problems and are typically studied in courses like 201
  - We won't look in detail at implementations
  - We will mention the implementations
  - Viewed via APIs and problems solved
Stacks Queues and Deques, Oh My!

- Linear structures used in problem domains and algorithms: Stack, Queue, Dequeue.
  - Similar abstractions with different semantics
https://www.google.com/search?q=stack&source=lnms&tbm=isch
Why don't we just use arrays?

- **Stacks** used in implementing recursion, postscript language, Java language, graph algorithms
  - Stacks implemented using array/ArrayList

- **Queues** used in simulation, graph algorithms, scheduling
  - Queues implemented using array/LinkedList

- **Deque (dequeue)** double ended queue, Gradescope
  - Implemented using LinkedList or other
Stacks: Java and Otherwise

https://git.cs.duke.edu/201fall16/linearstructures/tree/master/src

● See StackDemo.java
  ➢ Note that Stack is in java.util
  ➢ Stack is generic, e.g., Stack<String>

● https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html
  ➢ See API documentation for details on methods

● In code example: push, pop, create, size
  ➢ Also noted isEmpty and peek
Simple stack example

- **Stack** is part of java.util.Collections hierarchy
  - Typically avoid `.peek()`, but can be useful

```java
Stack<String> s = new Stack<>();

s.push("panda");
s.push("grizzly");
s.push("brown");

System.out.println("size = "+s.size());
System.out.println(s.peek());
String str = s.pop();
System.out.println(s.peek());
System.out.println(s.pop());
```
Two applications of stacks

- Manage runtime of method invocations, e.g., in implementing compiler. Note that Java Virtual machine is a stack-based machine.

- Postfix notation for arithmetic expressions
  - Can be used in evaluating expressions
  - What is $3 + 5 * 7$? This is NOT 56
  - Without precedence, need parentheses, e.g., we'd write $(3 + 5) * 7$

- Postfix is sometimes called Reverse Polish Notation or RPN
Postfix, prefix, and infix notation

- **Postfix notation used in some HP calculators**
  - No parentheses needed, precedence rules work
  - `3 5 + 4 2 * 7 + 3 - 9 7 + *`
  - **Read expression**
    - For number/operand: push onto stack
    - For operator: pop, pop, operate, push

- **To evaluate prefix: (3 + 5) * 7 we'd use**
  - `3 5 + 7 *`
  - What is `3 5 7 * +`
Postfix.java

- To evaluate a postfix expression:
  - If number, push. If operator: pop, pop, op, push
- [https://git.cs.duke.edu/201fall16/linearstructures/blob/master/src/Postfix.java](https://git.cs.duke.edu/201fall16/linearstructures/blob/master/src/Postfix.java)

- Things to look at in program: What exceptions are caught? When do these occur?
  - Number format and Stack exceptions
- How does StringTokenizer work:
  - Like .split(), but delivers tokens or chunks one at a time. Delimiters or separators are tokens too
Modifying Postfix.java

- How can we add exponentiation to the program
  - 2 3 ^ would be 8 and 3 2 ^ would be 9

- Must add new delimiter
  - What happens before we add ^ when ^ entered?
  - What happens after we add ^ when ^ entered?

- Must add new line in method operator
  - How do we return appropriate value? Test?
WOTO


- Look at source code for Postfix.java to understand both RPN and parsing using tokens in Java
Barbara Liskov

Turing Award Winner in 2008 for
For contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.

The advice I give people in general is that you should figure out what you like to do, and what you can do well—and the two are not all that dissimilar, because you don’t typically like doing something if you don’t do it well. … So you should instead watch—be aware of what you’re doing, and what the opportunities are, and step into what seems right, and see where it takes you.
Queue: another linear ADT

- **FIFO**: first in, first out, used in many applications
  - Scheduling jobs/processes on a computer
  - Computer simulations, lines at stores or banks: one line or multiple lines?

- **Common operations**: add (back), remove (front), peek ??
  - `java.util.Queue` is an interface
  - `java.util.LinkedList` implements the interface
    - `add()`, `remove()`, `offer()`, `poll()`
Stacks: Java and Otherwise

https://git.cs.duke.edu/201fall16/linearstructures/tree/master/src

● See QueueDemo.java
  ➢ Note that Queue is in java.util and is an Interface! Typically use LinkedList to create concrete object

● https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html
  ➢ See API documentation for details on methods and on many implementing classes

● Why use LinkedList? O(1) add back, remove front
Simple Queue example

- Queue is part of java.util.Collections hierarchy
  - Typically avoid .peek(), but can be useful

```java
Queue<String> q = new LinkedList<>();
q.add("panda");
q.add("grizzly");
q.add("brown");
System.out.println("size = "+q.size());
System.out.println(q.peek());
String str = q.remove();
System.out.println(q.peek());
System.out.println(q.remove());
```
Queue applications

- **Simulation, discrete-event simulation**
  - How many toll-booths do we need? How many express lanes or self-checkout at grocery store? Runway access at airport?
  - Queues facilitate simulation with mathematical distributions governing events, e.g., Poisson distribution for arrival times

- **Shortest path, e.g., in word-ladder or Flood-Fill algorithms we'll see on Friday**
  - Get from "white" to "house" one-letter at a time?
    - white, while, whale, shale, shake, ...?
Wordladder Story

- Ladder from ‘white’ to ‘house’
  - White, while, whale, shale, ...
- I can do that... optimally
  - My brother was an English major
  - My ladder is 16, his is 15, how?
- There's a ladder that's 14 words!
  - The key is ‘sough’

- Guarantee optimality!
  - QUEUE
Queue for shortest path – see gitlab

public boolean findLadder(String[] words,
                             String first, String last){
    Queue<String> qu = new LinkedList<>();
    Set<String> set = new HashSet<>();

    qu.add(first);
    while (qu.size() > 0){
        String current = qu.remove();
        if (oneAway(current,last)) return true;

        for(String s : words){
            if (!set.contains(s) && oneAway(from,s)){
                qu.add(s);
                set.add(s);
            }
        }
    }
    return false;
}
Shortest Path reprised

● How does Queue ensure we find shortest path?
  ➢ Where are words one away from first?
  ➢ Where are words two away from first?

● Why do we need to avoid revisiting a word, when?
  ➢ Why do we use a set for this? Why a HashSet?
  ➢ Alternatives?

● If we want the ladder, not just whether it exists
  ➢ What's path from white to house? We know there is one.
Shortest path proof

- All words one away from start on queue first iteration
  - What is first value of current when loop entered?
- All one-away words dequeued before two-away
  - See previous assertion, property of queues
  - Two-away before 3-away, ...
- Each 2-away word is one away from a 1-away word
  - So all enqueued after one-away, before three-away
    - How do we find three-away word?

- Any $w$ seen/dequeued that's $n$-away is:
  - Seen before every $n+k$-away word for $k \geq 1$!
Keeping track of ladder

- **Find w, a one-away word from current**
  - Enqueue w if not seen
  - Call `map.put(w, current)`

- **Remember keys are unique!**
  - Put word on queue once!
  - `map.put("lot", "hot")`
  - `map.put("dot", "hot")`
  - `map.put("hat", "hot")`
Reconstructing Word Ladder

- Run WordLaddersFull
  - https://git.cs.duke.edu/201fall16/linearstructures/blob/master/src/WordLaddersFull.java

- Notice map and call to `map.put(word, current)`
  - What about when returning the ladder, why is the returned ladder in reverse order?
  - What do we know about code when statement adding `(key, value)` to `map` runs?
WOTO


What's your favorite word ladder?
Could you write code to find the longest word ladder?