Spirits of Linear Structures

- Past, Present, Future, ...
- Linked-lists and arrays and ArrayLists
  - ADT: insert, delete, iterate/traverse, grow, ...
  - Advantages and trade-offs include ...

- Goal: structures that support *very* efficient insertion and lookup, lists can't do better than $O(n)$ for one of these: consider binary search and insert for arrays, or insert and lookup for linked lists. One is very good, the other not-so-good.
  - What’s the hybrid approach?

Stacks Queues and Deques, Oh My!

- Linear structures used in problem domains and algorithms: Stack, Queue, Dequeue.
  - Similar abstractions with different semantics
  - What?

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, wiki...
  - Implemented using LinkedList or other

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
Simple stack example

- Stack is part of java.util.Collections hierarchy
  - As an ADT it’s a LIFO data structure
  - In Java you can randomly access it: OO abomination
  - what does pop do? What does push do?

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

Postfix, prefix, and infix notation

- Postfix notation used in some HP calculators
  - No parentheses needed, precedence rules still respected
  - Read expression
    - For number/operand: push
    - For operator: pop, pop, operate, push

```plaintext
3 5 + 4 2 * 7 + 3 - 9 7 + *
```

- See Postfix.java for example code, key ideas:
  - Use StringTokenizer, handy tool for parsing
  - Note: Exceptions thrown, what are these?

- What about prefix and infix notations, advantages?

Interlude: Exceptions

- Exceptions are raised or thrown in exceptional cases
  - Bad indexes, null pointers, illegal arguments, ...
  - File not found, URL malformed, ...

- Runtime exceptions aren't handled or caught
  - Bad index in array, don’t try to handle this in code
  - Null pointer stops your program, don’t code that way!

- Some exceptions are caught or rethrown
  - FileNotFoundException and IOException

- RuntimeException extends Exception
  - catch not required

Danny Hillis

The third culture consists of those scientists and other thinkers in the empirical world who, through their work and expository writing, are taking the place of the traditional intellectual in rendering visible the deeper meanings of our lives, redefining who and what we are.

This comes back to the view of the Web as the first form of life to grow on the Internet. It’s the slime mold of the Internet. I don’t want to disparage it, but it’s primitive.

When computers pass around Web pages, they don’t know what they’re talking about. They’re just passing around bits, and they can be meaningless bits as far as they’re concerned. They’re just acting like a big telephone system. (1996, edge.org)
Queue: another linear ADT

- **FIFO**: first in, first out, used in many applications
  - Scheduling jobs/processes on a computer
  - Tenting policy?
  - Computer simulations
- Common operations: add (back), remove (front), peek ??
  - `java.util.Queue` is an interface (jdk5)
    - `offer(E)`, `remove()`, `peek()`, `size()`
  - `java.util.LinkedList` implements the interface
    - `add()`, `addLast()`, `getFirst()`, `removeFirst()`
- Downside of using `LinkedList` as queue
  - Can access middle elements, remove last, etc. why?

Stack and Queue implementations

- Different implementations of queue (and stack) not interesting from an algorithmic standpoint
  - Complexity is the same, performance may change (why?)
  - Use `ArrayList`, growable array, `Vector`, `linked list`, ...
- As we'll see `java.util.LinkedList` is good basis for all
  - In Java 5+, `LinkedList` implements the Queue interface, low-level linked lists/nodes facilitate (circular list!)
- `ArrayList` for queue is tricky, *ring buffer* implementation, add but wrap-around if possible before growing
  - Tricky to get right (exercise left to reader)

Implementation is very simple

- Extends Vector, so simply wraps Vector/`ArrayList` methods in better names
  - `push==add`, `pop==remove` (also peek and empty)
  - Note: code below for `ArrayList`, `Vector` is used
    - Stack is generic, so Object replaced by generic reference (see next slide)

```java
public Object push(Object o) {
    add(o);
    return o;
}
public Object pop() {
    return remove(size() - 1);
}
```

Implementation is very simple

- Extends Vector, so simply wraps Vector/`ArrayList` methods in better names
  - What does generic look like?

```java
public class <E> Stack extends ArrayList<E> {
    public E push(E o) {
        add(o);
        return o;
    }
    public E pop(Object o) {
        return remove(size() - 1);
    }
}
```
Uses or “has-a” rather than “is-a”

- Suppose there's a private ArrayList myStorage
  - Doesn't extend Vector, simply uses Vector/ArrayList
  - Disadvantages of this approach?
    - Synchronization issues

```
public class Stack<E> {
    private ArrayList<E> myStorage;
    public Stack() {
        myStorage = new ArrayList<E>();
    }
    public E push(E o) {
        myStorage.add(o);
        return o;
    }
    public E pop() {  
        return myStorage.remove(size() - 1);
    }
}
```

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 flood-fill to find path to some neighbor or in word-ladder
  - Get from “white” to “house” one-letter at a time?
    - white, while, whale, shale, shake, ...

```
public boolean ladderExists(String[] words, String from, String to) {
    Queue<String> q = new LinkedList<String>();
    Set<String> used = new TreeSet<String>();
    for (String s : words) {
        if (oneAway(from, s)) {
            q.add(s);
            used.add(s);
        }
    }
    while (q.size() != 0) {
        String current = q.remove();
        if (oneAway(current, to)) return true;
        // add code here, what?
    }
    return false;
}
```

Maria Klawe

Chair of Computer Science at UBC, Dean of Engineering at Princeton, President of Harvey Mudd College, ACM Fellow...

Klawe’s personal interests include painting, long distance running, hiking, kayaking, juggling and playing electric guitar. She describes herself as “crazy about mathematics” and enjoys playing video games.

“I personally believe that the most important thing we have to do today is use technology to address societal problems, especially in developing regions”

Queue for shortest path (see APT)

```
public boolean ladderExists(String[] words, String from, String to) {
    Queue<String> q = new LinkedList<String>();
    Set<String> used = new TreeSet<String>();
    for (String s : words) {
        if (oneAway(from, s)) {
            q.add(s);
            used.add(s);
        }
    }
    while (q.size() != 0) {
        String current = q.remove();
        if (oneAway(current, to)) return true;
        // add code here, what?
    }
    return false;
}
```
Shortest Path reprised

- How does Queue ensure we find shortest path?
  - Where are words one away from start?
  - Where are words two away from start?

- Why do we need to avoid revisiting a word, when?
  - Why do we use a set for this? Why a TreeSet?
  - Alternatives?

- If we want the ladder, not just whether it exists
  - What’s path from white to house? We know there is one.
  - Ideas? Options?

Shortest path proof

- All words one away from start on queue before loop
  - Obvious from code

- 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 $k \geq 1$!