Linear Structures Revisited

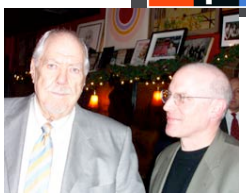
- Linked-lists and arrays and ArrayLists and …
  - Linear structures, operations include insert, delete, traverse, …
  - Advantages and trade-offs include …
- Goal: move toward structures that support very efficient insertion and lookup, lists can't do better than O(n) for one of these: consider binary search and insert for arrays, or insert and lookup for linked lists. One is 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?

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!



Stack: What problems does it solve?

- Stacks are used to avoid recursion, a stack can replace the implicit/actual stack of functions called recursively
- Stacks are used to evaluate arithmetic expressions, to implement compilers, to implement interpreters
  - Java Virtual Machine (JVM) is a stack-based machine
  - Postscript is a stack-based language
  - Stacks are used to evaluate arithmetic expressions in many languages
- Small set of operations: LIFO or last in is first out access
  - Operations: push, pop, top, create, clear, size
  - More in postscript, e.g., swap, dup, rotate, …

Simple stack example

- `Stack` is part of java.util.Collections hierarchy
  - It's an OO abomination, extends Vector (like ArrayList)
    - Should be implemented using Vector
    - Doesn't model "is-a" inheritance, does use "has-a"
  - what does pop do? What does push do?

```
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

    `3 5 +` `4 2 * 7 + 3 -` `9 7 + *`
  - Read expression
    - For number/operand: push
    - For operator: pop, pop, operate, push
- 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 meant to be 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!
- Other exceptions must be caught or rethrown
  - See `FileNotFoundException` and `IOException` in Scanner class implementation
- 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.

(Wired 1998) And now we are beginning to depend on computers to help us evolve new computers that let us produce things of much greater complexity. Yet we don't quite understand the process - it's getting ahead of us. We're now using programs to make much faster computers so the process can run much faster.

That's what's so confusing - technologies are feeding back on themselves; we're taking off. We're at that point analogous to when single-celled organisms were turning into multicelled organisms. We are amoebas and we can't figure out what the hell this thing is that we're creating.

Prefix and Postfix notation in action

- Scheme/LISP and other functional languages tend to use a prefix notation
- Postscript printers use a stack-based language

```
(define (square x) (* x x))
(define (expt b n)
 (if (= n 0)
   1
   (* b (expt b (- n 1)))))
```

```
%!
200 200 moveto
100 100 rlineto
200 300 moveto
100 -100 rlineto
stroke showpage
```


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) aren't really interesting from an algorithmic standpoint
  - Complexity is the same, performance may change (why?)
  - Use ArrayList, growable array, Vector, linked list, …
    - Any sequential structure
- 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)

```
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?

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


Using linear data structures

- We've studied arrays, stacks, queues, which to use?
  - It depends on the application
  - ArrayList is multipurpose, why not always use it?
    - Make it clear to programmer what's being done
    - Other reasons?
- Other linear ADTs exist
  - List: add-to-front, add-to-back, insert anywhere, iterate
    - Alternative: create, head, tail, Lisp or
    - Linked-list nodes are concrete implementation
  - Deque: add-to-front, add-to-back, random access
    - Why is this "better" than an ArrayList?
    - How to implement?

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 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 aiport?
  - 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
  - How do we get from "white" to "house" one-letter at a time?
    - white, while, whale, shale, shake, …?

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 use of 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?
- What 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

- Every word one away from start is on queue before loop
  - Obvious from code
- All one-away words dequeued before two-away..n-away
  - See previous assertion, property of queues
- Every two-away word is one away from a one-away word
  - So all enqueued after one-away, before three-away
    - How do we find three-away word?
- Word w put on queue is one-away from an n-away word
  - w is n+1(away), can't be earlier than that, why?