What’s the Difference Here?

- How does find-a-track work? Fast forward?

Contrast LinkedList and ArrayList

- See ISimpleList, SimpleLinkedList, SimpleArrayList
  - Meant to illustrate concepts, not industrial-strength
  - Very similar to industrial-strength, however
- ArrayList --- why is access $O(1)$ or constant time?
  - Storage in memory is contiguous, all elements same size
  - Where is the 1st element? 40th? 360th?
  - Doesn’t matter what’s in the ArrayList, everything is a pointer or a reference (what about null?)

What about LinkedList?

- Why is access of Nth element linear time?
- Why is adding to front constant-time $O(1)$?

ArrayLists and linked lists as ADTs

- As an ADT (abstract data type) ArrayLists support
  - Constant-time or $O(1)$ access to the k-th element
  - Amortized linear or $O(n)$ storage/time with add
    - Total storage used in n-element vector is approx. 2n, spread over all accesses/additions (why?)
    - Adding a new value in the middle of an ArrayList is expensive, linear or $O(n)$ because shifting required
- Linked lists as ADT
  - Constant-time or $O(1)$ insertion/deletion anywhere, but...
  - Linear or $O(n)$ time to find where, sequential search
- Good for sparse structures: when data are scarce, allocate exactly as many list elements as needed, no wasted space/copying (e.g., what happens when vector grows?)
Linked list applications

- Remove element from middle of a collection, maintain order, no shifting. Add an element in the middle, no shifting
  - What’s the problem with a vector (array)?
  - Emacs visits several files, internally keeps a linked-list of buffers
  - Naively keep characters in a linked list, but in practice too much storage, need more esoteric data structures

- What’s \((3x^3 + 2x^2 + x + 5) + (2x^4 + 5x^3 + 2x^2 + 4x)\)?
  - As a vector \((3, 0, 2, 0, 1, 5)\) and \((0, 2, 5, 1, 4, 0)\)
  - As a list \((3, 5), (2, 3), (1, 1), (5, 0)\) and _______?
  - Most polynomial operations sequentially visit terms, don’t need random access, do need “splicing”
- What about \((3x^{100} + 5)\)?

Linked lists, CDT and ADT

- As an ADT
  - A list is empty, or contains an element and a list
  - ( ) or (x, (y, ( ) ) )

- As a picture

- As a CDT (concrete data type)

```
public class Node {
    String value;
    Node next;
    Node p = new Node();
    p.value = "hello";
    p.next = null;
}
```

Building linked lists

- Add words to the front of a list (draw a picture)
  - Create new node with next pointing to list, reset start of list

```
public class Node {
    String value;
    Node next;
    Node(String s, Node link){
        value = s;
        next = link;
    }
}
```

// ... declarations here
Node list = null;
while (scanner.hasNext()) {
    list = new Node(scanner.next(), list);
}

- What about adding to the end of the list?

Linked list applications continued

- If programming in C, there are no “growable-arrays”, so typically linked lists used when # elements in a collection varies, isn’t known, can’t be fixed at compile time
  - Could grow array, potentially expensive/wasteful especially if # elements is small.
  - Also need # elements in array, requires extra parameter
  - With linked list, one pointer used to access all the elements in a collection

- Simulation/modeling of DNA gene-splicing
  - Given list of millions of CGTA... for DNA strand, find locations where new DNA/gene can be spliced in
    - Remove target sequence, insert new sequence
Dissection of add-to-front

- List initially empty
- First node has first word

```
list = new Node(word, list);
```

- Each new word causes new node to be created
  - New node added to front
- Rhs of operator = completely evaluated before assignment

Building linked lists continued

- What about adding a node to the end of the list?
  - Can we search and find the end?
  - If we do this every time, what's complexity of building an N-node list? Why?

- Alternatively, keep pointers to first and last nodes of list
  - If we add node to end, which pointer changes?
  - What about initially empty list: values of pointers?
    - Will lead to consideration of header node to avoid special cases in writing code

- What about keeping list in order, adding nodes by splicing into list? Issues in writing code? When do we stop searching?

Standard list processing (iterative)

- Visit all nodes once, e.g., count them or process them

```
public int size(Node list){
    int count = 0;
    while (list != null) {
        count++;
        list = list.next;
    }
    return count;
}
```

- What changes in code above if we change what “process” means?
  - Print nodes?
  - Append “s” to all strings in list?

Standard list processing (recursive)

- Visit all nodes once, e.g., count them

```
public int recsize(Node list) {
    if (list == null) return 0;
    return 1 + recsize(list.next);
}
```

- Base case is almost always empty list: null pointer
  - Must return correct value, perform correct action
  - Recursive calls use this value/state to anchor recursion
  - Sometimes one node list also used, two “base” cases
- Recursive calls make progress towards base case
  - Almost always using list.next as argument
Recursion with pictures

- Counting recursively

```java
int recsize(Node list){
  if (list == null)
    return 0;
  return 1+ recsize(list.next);
}
```

Recursion and linked lists

- Print nodes in reverse order
  - Print all but first node and...
  - Print first node before or after other printing?

```java
public void print(Node list) {
  if (list != null) {
    System.out.println(list.info);
    print(list.next);
  }
}
```

Complexity Practice

- What is complexity of Build? (what does it do?)

```java
public Node build(int n) {
  if (null == n) return null;
  Node first = new Node(n, build(n-1));
  for(int k = 0; k < n-1; k++) {
    first = new Node(n,first);
  }
  return first;
}
```

- Write an expression for T(n) and for T(0), solve.
  - Let T(n) be time for build to execute with n-node list
  - T(n) = T(n-1) + O(n)

Changing a linked list recursively

- Pass list to method, return altered list, assign to list
  - Idiom for changing value parameters

```java
list = change(list, "apple");
public Node change(Node list, String key) {
  if (list != null) {
    list.next = change(list.next, key);
    if (list.info.equals(key)) return list.next;
    return list;
  }
  return null;
}
```

- What does this code do? How can we reason about it?
  - Empty list, one-node list, two-node list, n-node list
  - Similar to proof by induction