Lists (a first look)

CSE 373
Data Structures
Readings

• Reading
  › Chapter 3
  › You can start peeking at Chapter 6
We will cover

- List ADT (a first look)
- List implementation
  - Array
  - Linked list
  - Doubly linked list
- An example application (long integers)
- Circular list
List ADT

- What is a List?
  - Ordered sequence of elements $A_1, A_2, \ldots, A_N$
- Elements may be of arbitrary type, but all are of the same type
- Elements have values
- Elements have positions (first, kth, last etc..)
Common operations on lists

- Constructor for an empty list
- Queries: size(); isEmpty();
- Insert and delete
  - Must indicate where: first, last, kth, after some element etc…
- Find, set, replace
  - With a given value, find previous etc…
- Will look at a “list interface” in the Java sense later
Simple Examples of List Use

• Polynomials
  › $25 + 4x^2 + 75x^{85}$
  › An element is a term whose value must indicate the power and the coefficient

• Unbounded Integers
  › $457680909938365839018745764949494578$
  › Do not fit within a single computer word
  › An element has for value a single digit
List Implementations

• Two types of implementation:
  › Array-Based
  › Linked list (pointer based)
List: Array Implementation

- Basic Idea:
  - Pre-allocate a big array of size MAX_SIZE
  - Keep track of current size using a variable `count`
  - Shift elements when you have to insert or delete (except of course for insertlast and deletelast)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>count-1</th>
<th>MAX_SIZE-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>...</td>
<td>A_N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## List: Array Implementation

Insert Z in kth position

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>MAX_SIZE-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>MAX_SIZE-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>Z</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Array Insert\(_{kth}\) Running Time

- Running time for \(N\) elements?
- On average, must move half the elements to make room – assuming insertions at positions are equally likely
- This is \(O(N)\) running time.
- Worst case is insert at position 0. Must move all \(N\) items one position before the insert. Still \(O(N)\)
Linked Implementation

• Basic Idea:
  › Allocate little blocks of memory (nodes) as elements are added to the list
  › Keep track of list by linking the nodes together
  › Change links (pointers) when you want to insert or delete
Linked list: Insert_after

Insert the value E after P
Insertion After

InsertAfter(p : node, e : thing): {
  x : node; // declares the type of x
  x := new node;
  x.value := v;
  x.next := p.next; // be sure to do in right order
  p.next := x;
}
**Linked List with Header Node**

Advantage: “insert after” and “delete after” can be easily done at the beginning of the list (insert_first and delete_first)
Linked list Implementation

Caveats

• Whenever you break a list, your code should fix the list up as soon as possible
  › **Draw pictures of the list** to visualize what needs to be done

• Pay special attention to **boundary conditions**:
  › Empty list
  › Single item – same item is both first and last
  › Two items – first, last, but no middle items
Linked List Insert Running Time

- Running time for N elements?
- "Insert_after" takes constant time (O(1))
- Does not depend on input size
- Compare to array based list Insert_kth which is O(N)
- However, how about Insert_last?
To delete the node pointed to by P, need a pointer to the previous node. Thus we need to traverse the list to find the previous node. So we might want to use ...
Doubly Linked Lists

- In singly linked lists, findPrevious (and hence Delete) is slow \([O(N)]\) because we cannot go directly to previous node
- Solution: Keep a "previous" pointer at each node

![Doubly Linked List Diagram]
Double Link Pros and Cons

• Advantage
  › Delete (not DeleteAfter) and FindPrev are faster

• Disadvantages:
  › More space used up (double the number of pointers at each node)
  › More book-keeping for updating the two pointers at each node (pretty negligible overhead)
Unbounded Integers Base 10

- **-4572**
  - Node pointers: X
  - Base 10 representation: $10^3 \rightarrow 4 \rightarrow 10^2 \rightarrow 5 \rightarrow 10^1 \rightarrow 7 \rightarrow 10^0 \rightarrow 2 \rightarrow -1$

- **348**
  - Node pointers: Y
  - Base 10 representation: $10^2 \rightarrow 3 \rightarrow 10^1 \rightarrow 4 \rightarrow 10^0 \rightarrow 8 \rightarrow 1$
Zero

null → -1

null → 1
Recursive Addition

- Positive numbers (or negative numbers)
Recursive Addition

• Mixed numbers

\[
\begin{align*}
3427 - 898 & \rightarrow 7 - 8 \rightarrow 342 - 89 \\
-898 & \rightarrow -8 \rightarrow -10 \rightarrow -1
\end{align*}
\]