Lists

CSE 373
Data Structures

Readings

• Reading
  › Goodrich and Tamassia, Chapter 5

List ADT

• What is a List?
  › Ordered sequence of elements A1, A2, ..., AN
• Elements may be of arbitrary type, but all are of the same type
• Common List operations are:
  › Insert, Find, Delete, IsEmpty, IsLast, FindPrevious, First, Kth, Last, Print, etc.

Simple Examples of List Use

• Polynomials
  › 25 + 4x^2 + 75x^65
• Unbounded Integers
  › 457609099383658390187457649494578
• Text
  › “This is an example of text”

List Implementations

• Two types of implementation:
  › Array-Based
  › 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

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & \text{count} & \text{MAX\_SIZE} \\
A_0 & A_1 & A_2 & A_3 & A_4 & A_5 \\
\end{array}
\]
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 List Insert Running Time

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

Review Big Oh Notation

- T(N) = O(f(N)) if there are positive constants c and n₀ such that:
  \[ T(N) < c \cdot f(N) \text{ when } N > n₀ \]
- T(N) = O(N) linear

List: Pointer 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 when you want to insert or delete

Pointer-Based Linked List

Pointer-based Insert (after p)

Insert the value E after P
Insertion After

InsertAfter(p : node pointer, v : thing): {
  x := new node;
  x.value := v;
  x.next := p.next;
  p.next := x;
}

Linked List with Header Node

Advantage: “insert after” and “delete after” can be done at the beginning of the list.

Linked List Delete

To delete the node pointed to by P, need a pointer to the previous node; See book for findPrevious method

Pointer Implementation Issues

• 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
  › Three or more items – first, last, and middle items

Pointer List Insert Running Time

• Running time for N elements?
• Insert takes constant time (O(1))
• Does not depend on input size
• Compare to array based list which is O(N)

Doubly 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
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

- \(-4572\)
- \(348\)

Recursive Addition

- Positive numbers (or negative numbers)

Example

- Mixed numbers