Linked lists

Prof. Noah Snavely
CS1114
http://www.cs.cornell.edu/courses/cs1114

Administrivia

- Assignment 3 due next Friday, 3/9

- Prelim 1! This Thursday in class
  - Topics through today (including running time, sorting, selection, graphs, linked lists)
  - Closed book / closed notes

- Review session Wednesday evening, 7pm, Upson 111
**Linked Lists -- Example**

![Linked Lists Example Diagram]

**Inserting an element – linked lists**

- Create a new cell and splice it into the list
  
  ![Inserting an Element Diagram]

- Splicing depends on where the cell goes:
  - How do we insert:
    - At the end?
    - In the middle?
    - At the beginning?
Adding a header

- We can represent the linked list just by the initial cell, but this is problematic
  - Problem with inserting at the beginning

- Instead, we add a header – a few entries that are not cells, but hold information about the list
  1. A pointer to the first element
  2. A count of the number of elements

### Linked list insertion

<table>
<thead>
<tr>
<th>Initial list</th>
<th>First element starts at 5</th>
<th>Size of list is 2</th>
<th>Insert a 5 at end</th>
<th>Insert an 8 after the 1</th>
<th>Insert a 6 at the start</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 2 2 0 1 3 X X X X X X X</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
<td></td>
<td>5 3 2 7 1 3 5 0 X X X X X</td>
<td>5 4 2 7 1 9 5 0 8 3 X X X</td>
<td>11 5 2 7 1 9 5 0 8 3 6 5 X</td>
</tr>
</tbody>
</table>
Linked list deletion

- We can also delete cells

- Simply update the header and change one pointers (to skip over the deleted element)

- Deleting things is the source of many bugs in computer programs
  - You need to make sure you delete something once, and only once

---

Initial list

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Delete the last cell

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Delete the 8

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Delete the first cell

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Linked lists – running time

- We can insert an item (at the front) in constant (O(1)) time
  - Just manipulating the pointers
  - As long as we know where to allocate the cell

- We can delete an element (at the front) in constant time

Linked lists – running time

- What about inserting / deleting from the end of the list?

- How can we fix this?
Doubly linked lists

A doubly-linked list in memory
Notes on doubly-linked lists

- Inserting and deleting at both ends is fast, but the code is very easy to get wrong
  - Try it on all cases, especially trivial ones
  - Look for invariants: statements that must be true of any valid list
  - Debug your code by checking invariants
    - In C/C++, this is done via assert
    - Most languages have a facility like this built in
      - But if not, you can just write your own!

Memory allocation

- So far we just assumed that the hardware supplied us with a huge array M
  - When we need more storage, we just grab locations at the end
    - Keep track of next free memory location
  - What can go wrong?
    - Consider repeatedly adding, deleting an item
- When we delete items from a linked list we change pointers so that the items are inaccessible
  - But they still waste space!
Storage reclamation

- Someone has to figure out that certain locations can be re-used (“garbage”)
  - If this is too conservative, your program will run slower and slower (“memory leak”)
  - If it’s too aggressive, your program will crash (“blue screen of death”)

Questions?