Linked lists

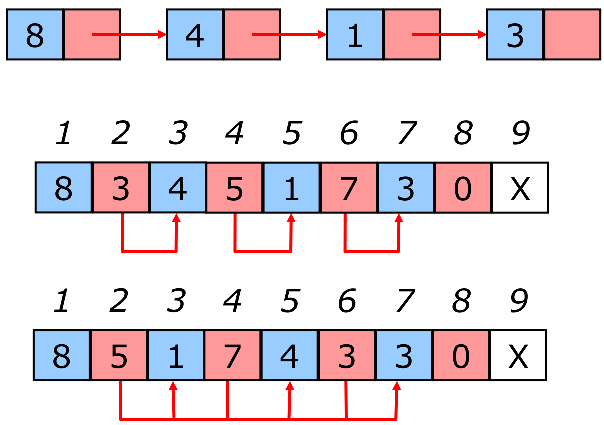
Prof. Noah Snavely
CS1114
http://www.cs.cornell.edu/courses/cs1114

Cornell University
Computer Science

Administrivia

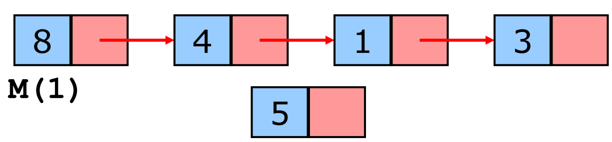
- Assignment 3 due next Friday, 3/9
- Prelim 1! This Thursday in class
  - Topics through today (including running time, sorting, selection, graphs, linked lists)
  - Closed book / closed notes
- Review session Wednesday evening, 7pm, Upson 111

Linked Lists -- Example



Inserting an element – linked lists

- Create a new cell and splice it into the list



- 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

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
Initial list	5	2	2	0	1	3	X	X	X	X	X	X	X

First element starts at 5

Size of list is 2

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
Insert a 5 at end	5	**3**	2	**7**	1	3	**5**	**0**	X	X	X	X	X

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
Insert an 8 after the 1	5	**4**	2	7	1	**9**	5	0	**8**	**3**	X	X	X

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
Insert a 6 at the start	**11**	**5**	2	7	1	9	5	0	8	3	**6**	**5**	X

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

Linked list deletion

	1	2	3	4	5	6	7	8	9	10	11	12	13
Initial list	5	4	2	7	1	9	5	0	8	3	X	X	X
Delete the last cell	5	**3**	2	**0**	1	9	5	0	8	3	X	X	X
Delete the 8	5	**2**	2	0	1	**3**	5	0	8	3	X	X	X
Delete the first cell	**3**	**1**	2	0	1	3	5	0	8	3	X	X	X

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

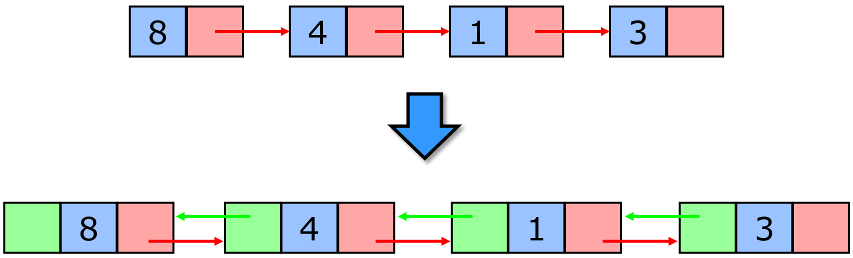
- We can delete an element (at the front) in constant time

Linked lists – running time

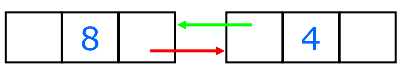
- What about inserting / deleting from the end of the list?

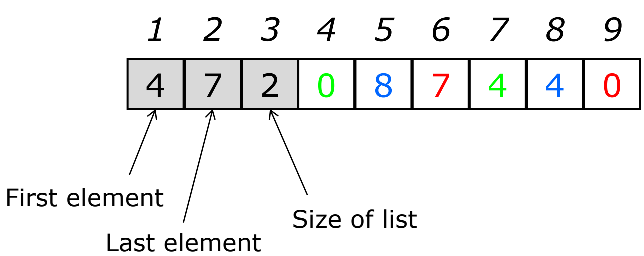
- How can we fix this?

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?