Implementing queues

Prof. Ramin Zabih

http://cs100r.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 3 is out, due this+next Friday
- Prelim on 10/11
  - Quiz 4 this Tuesday, 10/2
- Wednesday evening 10/3, RPC205:
  - 7PM: Bobby Kleinberg on graphs (optional!)
  - 8PM: RDZ prelim review
- Gurmeet will run a review session 10/10
  - Timing and details to be arranged

A note on reductions

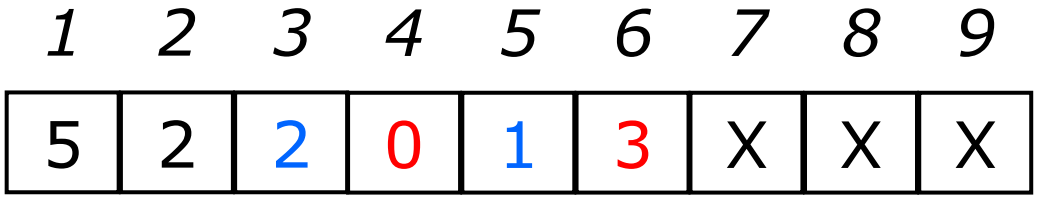
- The neurotic high school Q3 question actually asks you to do a **reduction**
  - Turn the problem you are given (rivalries) into one you already know about (graph coloring)
  - This is one classic use of reduction
    - Reduce an arbitrary instance of problem A to an instance of problem B
  - ♦♦ If you can do this, what else do you know?
    - Major topic in CS381/CS482
    - You'll see a simple example or two in CS100R

Linked lists with headers

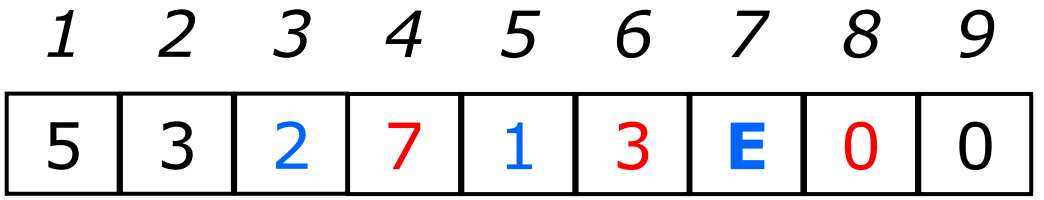
- To insert an element, we create a new cell and splice it into the list
  - At the beginning or end
    - Actually, could add it anywhere
  - Need to update header info, and the cell (if any) before the new one
- To delete an element we simply update the header and change some pointers
  - Basically, need to "skip over" deleted element

Linked list with header

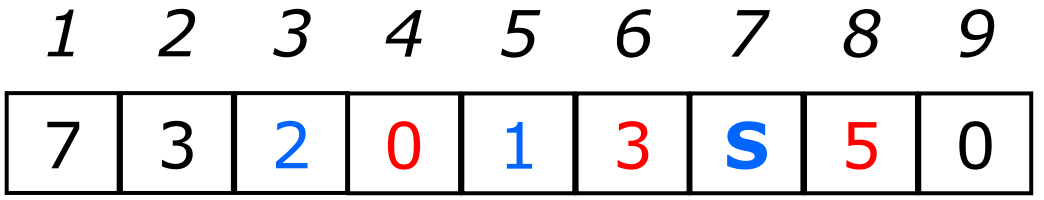
- Initial list:



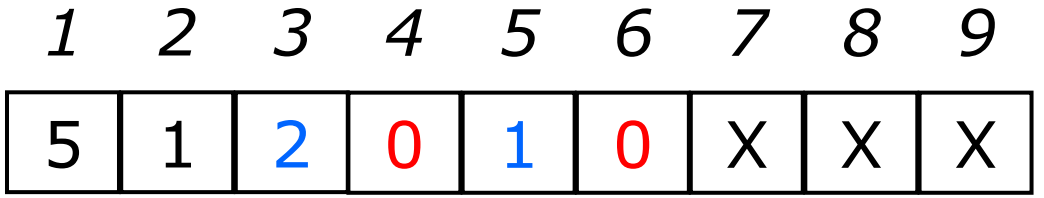
- Insert (end):



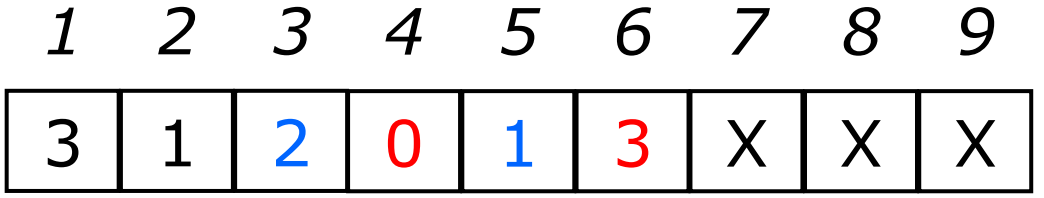
- Insert (start):



- Delete (end):



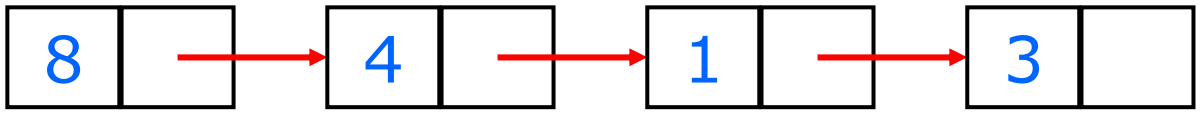
- Delete (start):

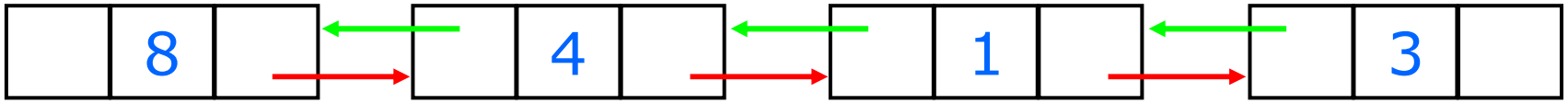


Linked lists and deques

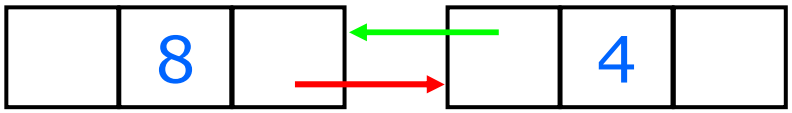
- Linked lists support insertion/deletion at beginning or end
  - What is the complexity of these 4 operations?
  - Is this a good way to implement queues?
- How do we modify linked lists so that we can delete from the end in constant time?
  - We clearly need a pointer to the last element in the header
    - But this is not enough!

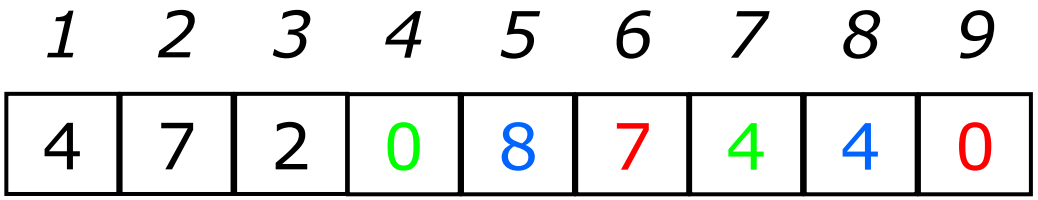
Doubly linked lists





A doubly-linked list in memory





Notes on doubly-linked lists

- Inserting and deleting 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
- Notice that when we delete items from a linked list we simply changed pointers so that the items were 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")
- Suppose your linked list was corrupted
  - ♦Why do computers crash when we read/write an arbitrary location? Must they??