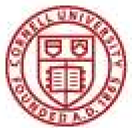
Implementing stacks

Prof. Ramin Zabih

http://cs100r.cs.cornell.edu



Cornell University
Computer Science

Administrivia

- Assignment 3 is out, due this+next Friday
- Prelim!
  - Options: Thursday 10/4 or 10/11
    - The other date will have Quiz 4
- First optional evening lecture:
  - Bobby Kleinberg on graphs
  - Wednesday evening 10/3 (exact time TBD)
  - RPC Auditorium (RPC205)
- A final note about big-O notation
  - Linear vs quadratic algorithms

Depth versus breadth

- In DFS, you only need to keep track of the vertices "above" you
  - I.e., on the path to the root
- In BFS, you need to store all the vertices at the same level
  - I.e., same distance to the root
- In a binary tree with 100 levels, this is the difference between storing 100 vertices and storing 2^{100} vertices!
  - So, why ever do BFS??

Computers and arrays

- The notion of “constant time” operations relies on a model of the hardware
- Computer memory is a large array
  - We will call it M
- In constant time, a computer can:
  - Read any element of M
  - Change any element of M to another element
  - Perform any simple arithmetic operation
- This is more or less what the hardware manual for an x86 describes

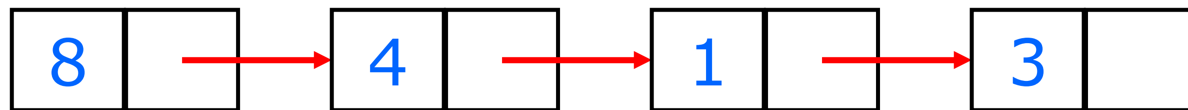
Data structures in memory

- This week we will focus on implementing various data structures using the array M
- We have a contract we wish to fulfill
  - Example: stack contract
    - If we push X onto S and then pop S, we get back X, and S is as before
- We want to fulfill this contract using the memory array M
- We'll look at a related data structure that is even more useful than a stack

Linked lists

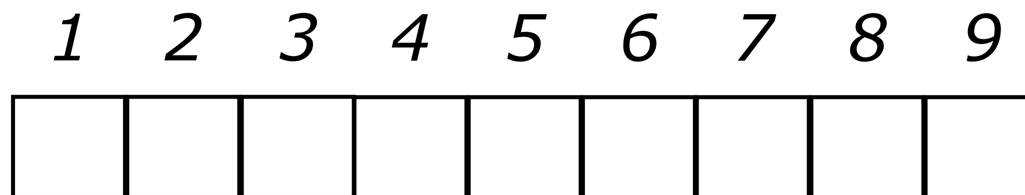
- You can add items to it, and you can process the items you added
- Unlike a stack, you don't need to pop it in order to get at all the items
- Adding items takes constant time
- Getting each item takes linear time
  - Example: searching the list for an element
- You can also delete an element
  - This is **always** the hard part of any data structure, and the source of most bugs

Linked lists



Linked lists as memory arrays

- We'll implement linked lists using M
  - This is very close to what the hardware does
    - More details in CS316

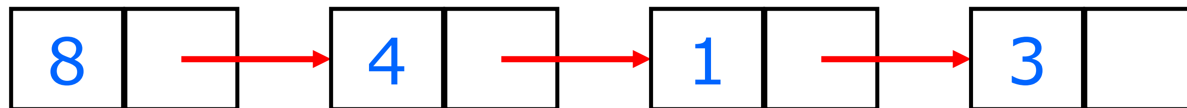


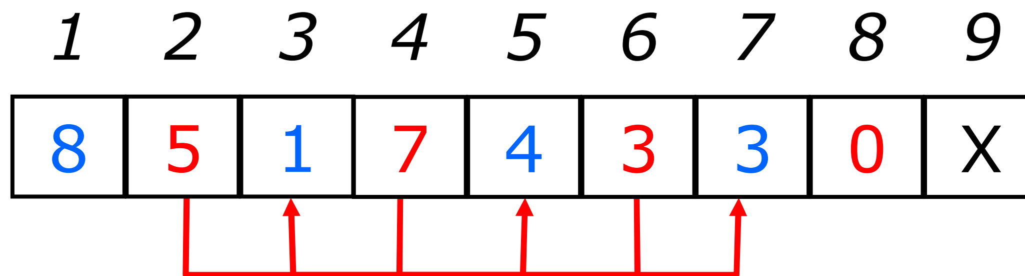
- A linked list contains "cells"
- A value, and where the next cell is
  - We will represent cells by a pair of adjacent array entries

A few details

- I will draw odd numbered entries in blue and even ones in red
  - Odd entries are values
    - Number interpreted as list elements
  - Even ones are "cells"
    - Number interpreted as index of the next cell
    - AKA *location*, *address*, or ***pointer***
- The first cell is M(1) and M(2), for now
- The last cell has 0, i.e. pointer to M(0)
  - Also called a "null pointer"

Example





Access every item in a list

- Start at the first cell, M(1) and M(2)
- Access the first value, M(1)
- The next cell is at location v=M(2)
- If v = 0, we are done
- Otherwise, access the next value, M(v)
- The next cell is at location v’ = M(v+1)
- Keep on going until the next cell is at location k, such that M(k+1) = 0

Adding a header

- We can represent the linked list just by the initial cell, but this is problematic
  - Think about deleting the last cell!
  - More subtly, it's also a problem for insertion
- Instead, we will have a few entries in M that are not cells, but instead hold some information about the list
  - Specifically, a pointer to the first element
  - Having a counter is also useful at times

Insert and delete

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

Linked list with header

- Initial list:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
5	2	2	0	1	3	X	X	X

- Insert (end):

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
5	3	2	7	1	3	**E**	0	0

- Insert (start):

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
7	3	2	0	1	3	**S**	5	0

- Delete (end):

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
5	1	2	0	1	0	X	X	X

- Delete (start):

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
3	1	2	0	1	3	X	X	X