Data Structures – Stacks and Queus

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

Cornell University
Computer Science

Administrivia

- Assignment 2, Part 2, due tomorrow
  - Please sign up for a Friday slot
- Assignment 3 will be out Friday
- Prelim 1! Next Thursday, 3/1, in class
  - There will be a review session Wednesday evening, 7pm, Upson 315

Finding blobs

1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Finding blobs

1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Blobs are connected components!

Finding components

1. Pick a 1 to start with, where you don't know which component it is in
   - When there aren't any, you're done
2. Give it a new component color
3. Assign the same component color to each pixel that is part of the same component
   - Basic strategy: color any neighboring 1's, have them color their neighbors, and so on

From Section

- For each vertex we visit, we color its neighbors and remember that we need to visit them at some point
  - Need to keep track of the vertices we still need to visit in a todo list
  - After we visit a vertex, we'll pick one of the vertices in the todo list to visit next
- This is also called *graph traversal*

Stacks and queues

- Two ways of representing a "todo list"

- Stack: Last In First Out (LIFO)
  - (Think cafeteria trays)
  - The newest task is the one you'll do next

- Queue: First In First Out (FIFO)
  - (Think a line of people at the cafeteria)
  - The oldest task is the one you'll do next

Stacks

- Two operations:

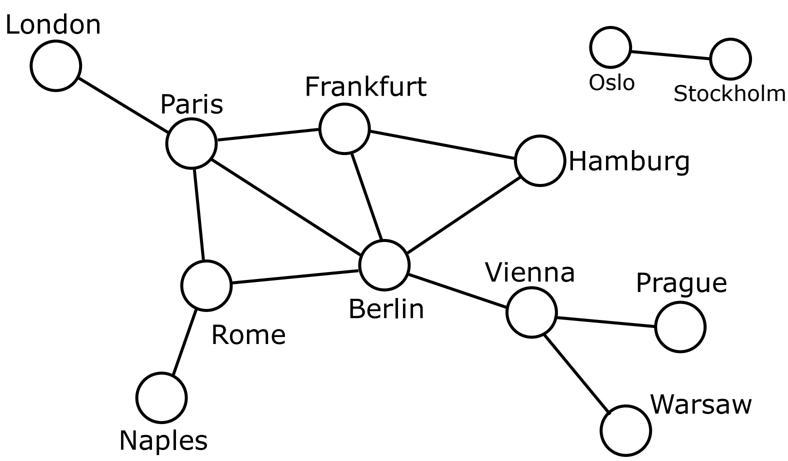
- Push: add something to the top of the stack

- Pop: remove the thing on top of the stack

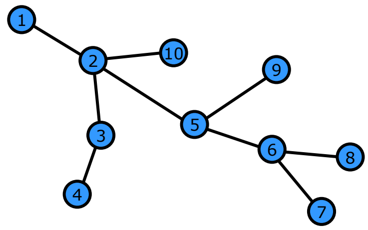
Queue

- Two operations:
- Enqueue: add something to the end of the queue
- Dequeue: remove something from the front of the queue

Graph traversal



Depth-first search (DFS)



- Call the starting node the *root*
- We traverse paths all the way until we get to a dead-end, then backtrack (until we find an unexplored path)
- Corresponds to using a *stack*

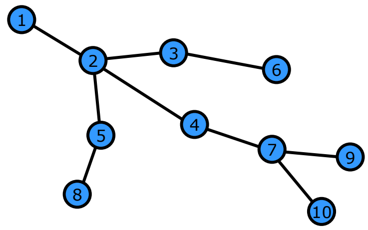
Another strategy

1. Explore all the cities that are one hop away from the root
2. Explore all cities that are two hops away from the root
3. Explore all cities that are three hops away from the root

   …

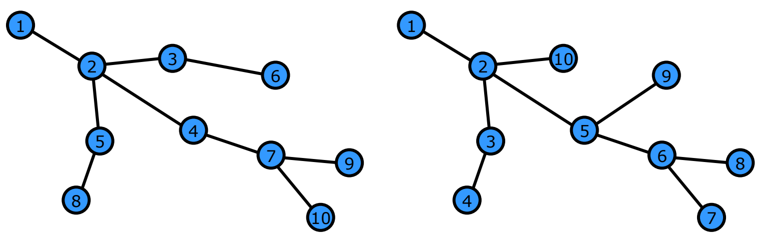
- This corresponds to using a *queue*

Breadth-first search (BFS)



- We visit all the vertices at the same level (same distance to the root) before moving on to the next level

BFS vs. DFS



Breadth-first (queue) Depth-first (stack)

Basic algorithms

BREADTH-FIRST SEARCH (Graph G)

- While there is an uncolored node **r**
  - Choose a new color
  - Create an empty queue **Q**
  - Let **r** be the root node, color it, and add it to **Q**
  - While **Q** is not empty
    - Dequeue a node **v** from **Q**
    - For each of **v**'s neighbors **u**
      - If **u** is not colored, color it and add it to **Q**

Basic algorithms

DEPTH-FIRST SEARCH (Graph G)

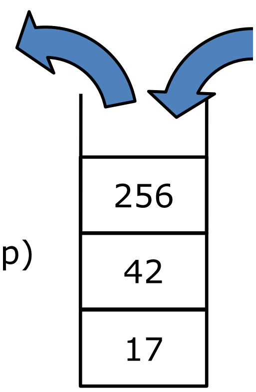
- While there is an uncolored node **r**
  - Choose a new color
  - Create an empty stack **S**
  - Let **r** be the root node, color it, and push it on **S**
  - While **S** is not empty
    - Pop a node **v** from **S**
    - For each of **v**'s neighbors **u**
      - If **u** is not colored, color it and push it onto **S**

Queues and Stacks

- Examples of Abstract Data Types (ADTs)
- ADTs fulfill a contract:
  - The contract tells you what the ADT can do, and what the behavior is
  - For instance, with a stack:
    - We can push and pop
    - If we push X onto S and then pop S, we get back X, and S is as before
- Doesn't tell you *how* it fulfills the contract

Implementing DFS

- How can we implement a stack?
  - Needs to support several operations:
  - Push (add an element to the top)
  - Pop (remove the element from the top)
  - IsEmpty

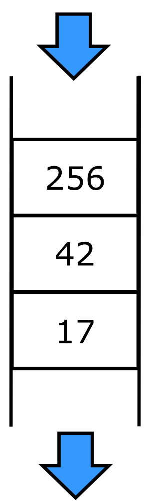


Implementing a stack

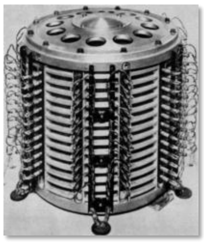
- IsEmpty
- Push (add an element to the top)
- Pop (remove an element from the top)

Implementing BFS

- How can we implement a queue?
  - Needs to support several operations:
  - Enqueue (add an element to back)
  - Dequeue (remove an element from front)
  - IsEmpty
- Not quite as easy as a stack…



Efficiency



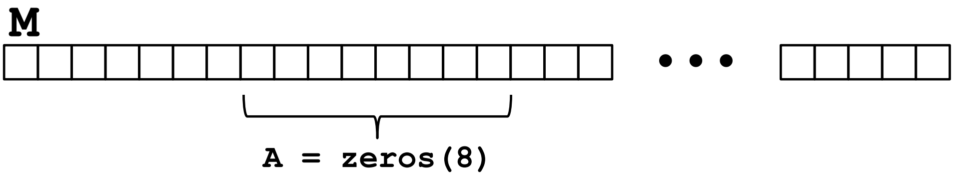
- Ideally, all of the operations (push, pop, enqueue, dequeue, IsEmpty) run in constant (O(1)) time
- To figure out running time, we need a model of how the computer's memory works

Computers and arrays

- Computer memory is a large array
  - We will call it M
- In constant time, a computer can:
  - Read any element of M (random access)
  - 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

Computers and arrays

- Arrays in Matlab are consecutive subsequences of M



Memory manipulation

- How long does it take to:
  - Read A(8)?
  - Set A(7) = A(8)?
  - Copy all the elements of an array (of size *n*) A to a new part of M?
  - Shift all the elements of A one cell to the left?

Implementing a queue: Take 1

- First approach: use an array
- Add (enqueue) new elements to the end of the array
- When removing an element (dequeue), shift the entire array left one unit

```
Q = [];
```


Implementing a queue: Take 1

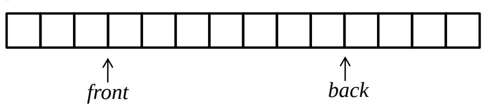
- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)

What is the running time?

- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)

Implementing a queue: Take 2

- Second approach: use an array AND
- Keep two pointers for the front and back of the queue



- Add new elements to the back of the array
- Take old elements off the front of the array

```
Q = zeros(1000000);
front = 1; back = 1;
```

Implementing a queue: Take 2

- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)

What is the running time?

- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)

Implementing a queue: Take 2

- What problems can occur?

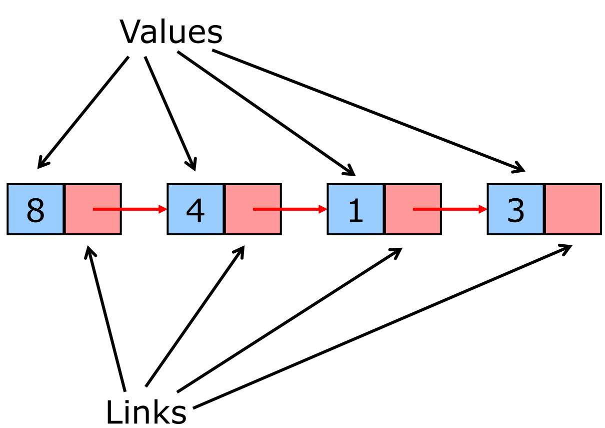
Questions?

Linked lists

- Alternative to an array

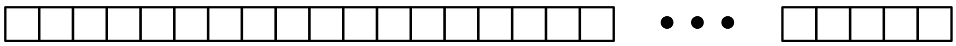
- Every element (cell) has two parts:
  1. A value (as in an array)
  2. A link to the next cell

Linked lists



Linked lists as memory arrays

M

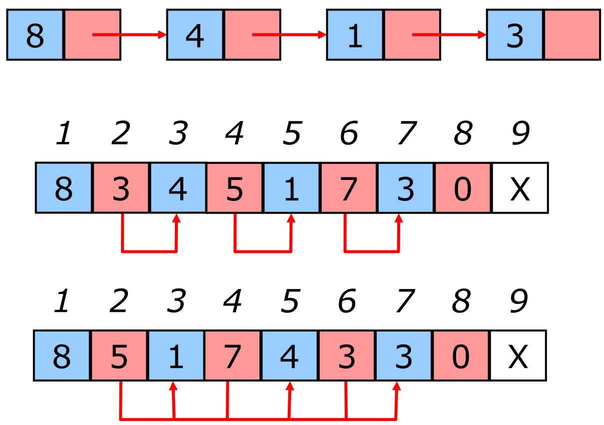


- We'll implement linked lists using M
- A cell will be represented by a pair of adjacent array entries

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 links
    - 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



Traversing a linked list

- Start at the first cell, **`[M(1),M(2)]`**
- Access the first value, **`M(1)`**
- The next cell is at location **`c = M(2)`**
- If **`c = 0`**, we're done
- Otherwise, access the next value, **`M(c)`**
- The next cell is at location **`c = M(c+1)`**
- Keep going until **`c = 0`**

Inserting an element – arrays

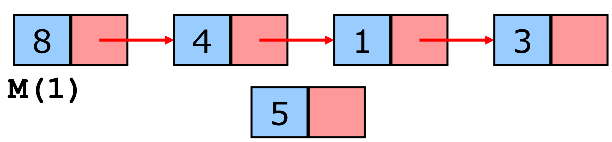
- How can we insert an element **x** into an array **A**?
- Depends where it needs to go:
  - End of the array:

    ```
    A = [A x];
    ```

  - Middle of the array (say, between elements A(5) and A(6))?
  - Beginning of the array?

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

Questions?