CS 1114: Implementing Search

Prof. Graeme Bailey

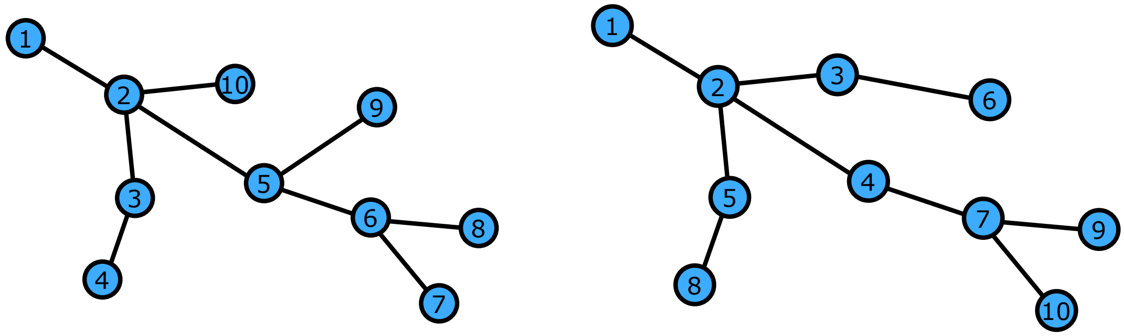
http://cs1114.cs.cornell.edu

(notes modified from Noah Snavely, Spring 2009)

Cornell University Computer Science

Last time

- Graph traversal



- Two types of *todo* lists:
  - Stacks → Depth-first search
  - Queues → Breadth-first search

Basic algorithms

BREADTH-FIRST SEARCH (Graph G)

- While there is an uncoloured node **r**
  - Choose a new colour
  - Create an empty *queue* **Q**
  - Let **r** be the root node, colour 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 coloured, colour it and add it to **Q**

Basic algorithms

DEPTH-FIRST SEARCH (Graph G)

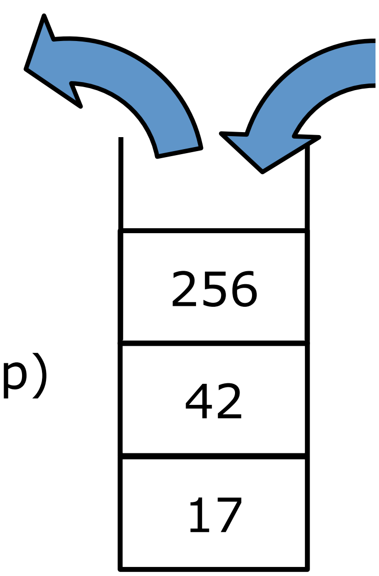
- While there is an uncoloured node **r**
  - Choose a new colour
  - Create an empty *stack* **S**
  - Let **r** be the root node, colour it, and push it on **S**
  - While **S** is not empty
    - Pop a node **v** from **S**
    - For each of **v**'s neighbors **u**
      - If **u** is not coloured, colour 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
- This is a *really important* technique!!!!

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

```
function e = IsEmpty(S)
    e = (length(S) == 0);
```

- Push (add an element to the top)

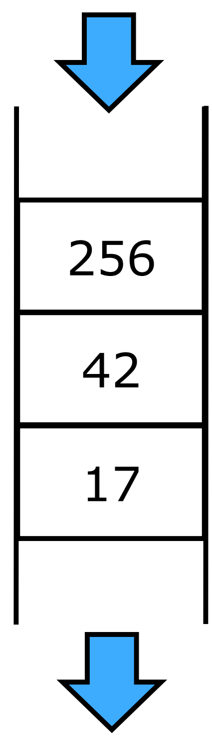
```
function S = push(S, x)
    S = [ S  x ]    % appends x to the end of the array S
```

- Pop (remove an element from the top)

```
function [S, x] = pop(S)
    n = length(S); x = S(n); S = S(1:n-1); % abbreviates S
    % but what happens if n = 0?
```


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…



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

```
function e = IsEmpty(Q)
   e = (length(S) == 0);
```

- Enqueue (add an element)

```
function Q = enqueue(Q,x)
   Q = [ Q  x ];
```

- Dequeue (remove an element)

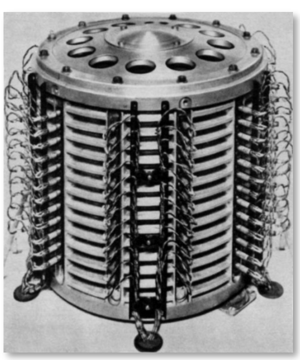
```
function [Q, x] = dequeue(Q)
   n = length(Q); x = Q(1);
   for i = 1:n-1
      Q(i) = Q(i+1); % everyone steps forward one step
```

But now imagine them all sitting in chairs in the queue!

What is the running time?

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

Efficiency



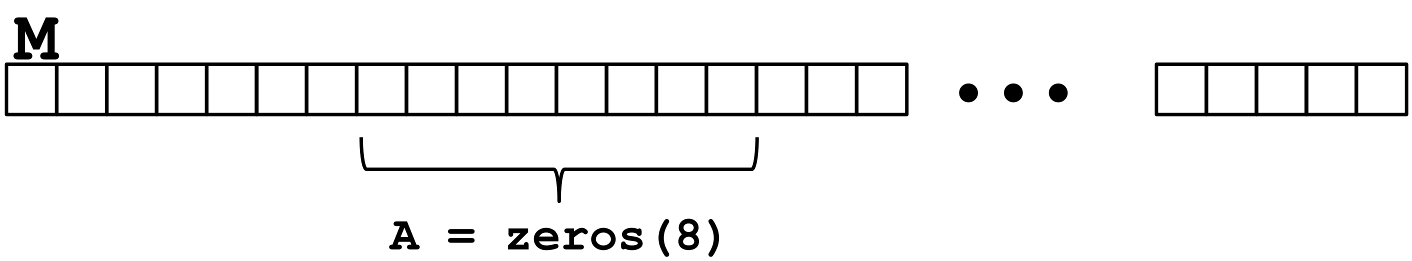
- Ideally, all of the operations (push, pop, enqueue, dequeue, IsEmpty) run in constant (O(1)) time
- To figure out running time, we need a model of how the computer's memory works

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

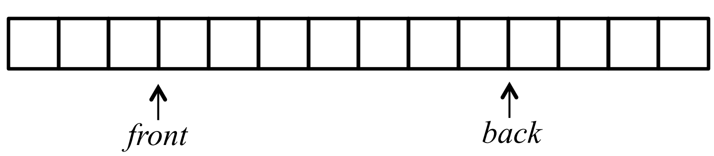


Memory manipulation

- 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 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

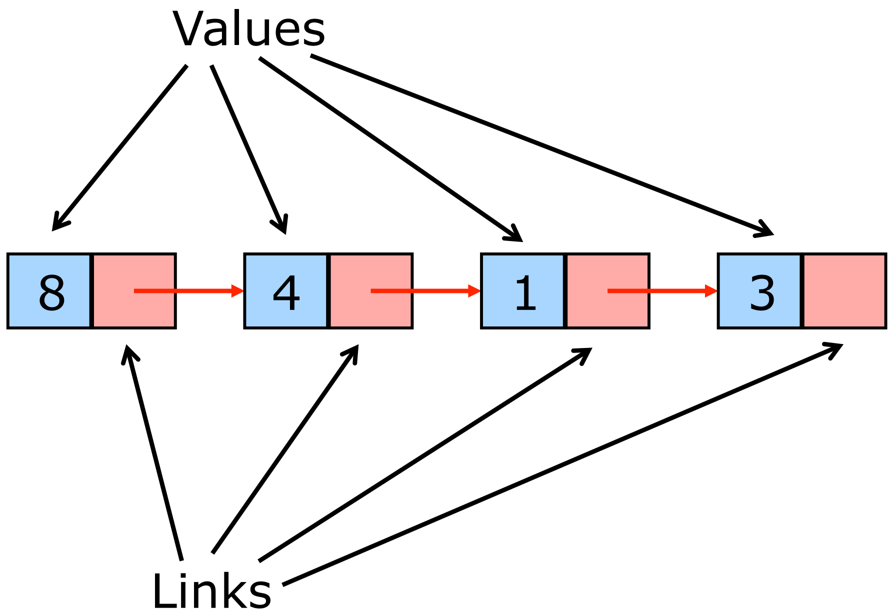
- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)

Implementing a queue: Take 3

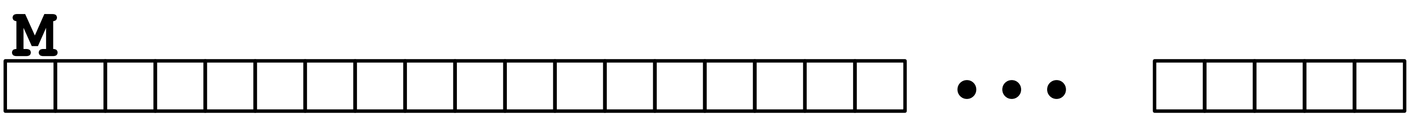
- Linked lists -

- 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

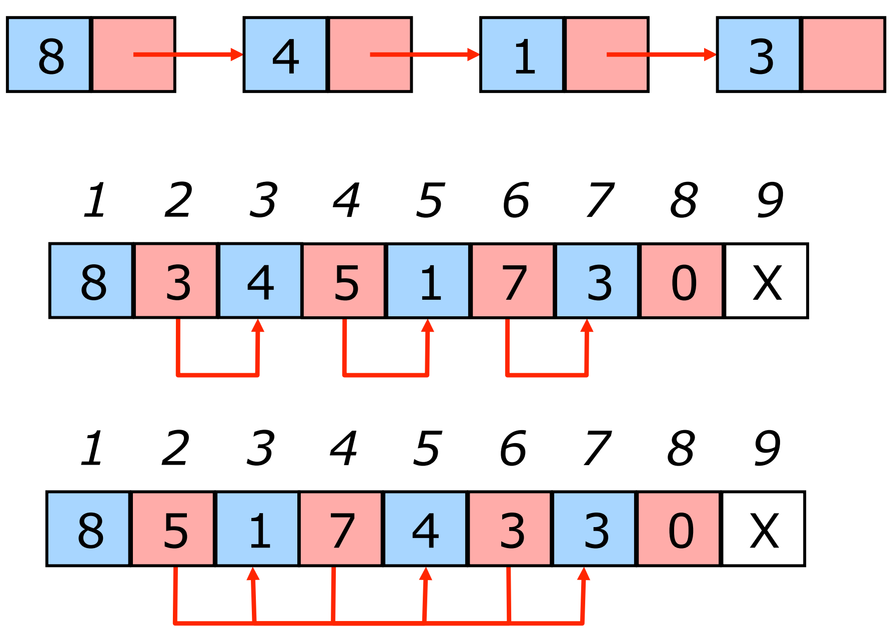


- We'll implement linked lists using M
- A cell will be represented by a pair of adjacent array entries

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

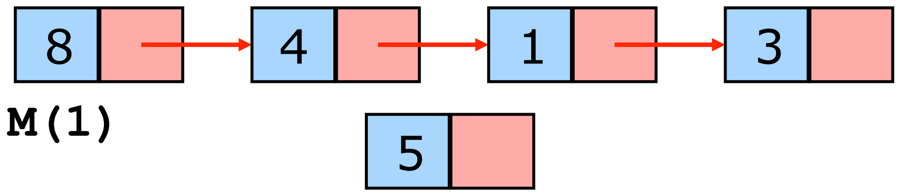
- How can we insert an element **`x`** into an array **`A`**?
- Depends where it needs to go:
  - End of the array:

```
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



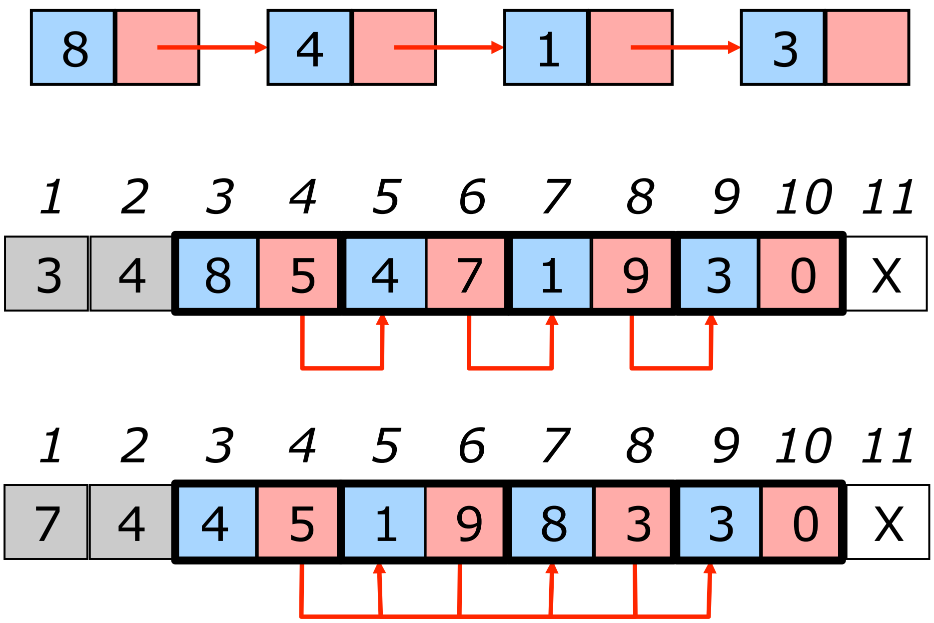
- Splicing depends on where the cell goes:
  - How do we insert:
    - At the end?
    - In the middle?
    - At the beginning?

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

Example



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 (→ cell 1); Size of list is 2 (→ cell 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
Insert an 8 after the 1	5	**4**	2	7	1	**9**	5	0	**8**	**3**	X	X	X
Insert a 6 at the start	**11**	**5**	2	0	1	9	5	0	8	3	**6**	**5**	X

Linked list deletion

- We can also delete cells
- Simply update the header and change one pointer (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
  - If we need to insert an item *inside* the list, then we must first *find* the place to put it.

- We can delete an element (at the front) in constant time
  - If the element isn't at the front, then we have to *find* it … how long does that take?

Linked lists – running time

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

- How long does it take to get there?