Programming Abstractions

CS106B

Cynthia Bailey Lee
Julie Zelenski

End of Quarter Details

- Week 10 assignment schedule:
  - Assignment 7 (Huffman Coding) is out now and **due Wednesday 12/7**

- Week 10 lecture schedule:
  - Today/Monday: quarter wrap-up lecture
  - Wednesday: “Ask Me Anything” with the CS106B teaching team
  - Friday: no class

Final Exam Info

Final Exam Details

- The exam is **Monday December 12 8:30am-11:30am**.
- Locations by first letter of your surname/family/last name.
  - Last names **Aalami-Hekmat**, in **Cemex Auditorium**
  - Last names **Heng-Zu**, in **Dinkelspiel Auditorium**
  - Students with special circumstances (SCPD, OAE) will receive an email from Head TA Neel with your arrangements.
- You will write your answers directly on the paper exam.
- The exam is **closed-book** and **closed-device**.
  - Like the midterm, we'll provide a reference sheet for Stanford library collections and other common functions, and you may bring 1-page/2-sides of notes (8.5" x 11")

Final Exam Topics

The emphasis of topics is heavier on topics that came after the midterm, since we haven't tested those yet.

- ADTs
- Recursion
- Backtracking
- Big-O analysis
- Pointers, new & delete, linked lists, memory diagrams, heap vs stack
- Classes and objects
- Binary heaps
- Binary search trees (BSTs)
- Tree traversals
- Searching and sorting algorithms, Graphs, Hashing
- Anything on any assignment

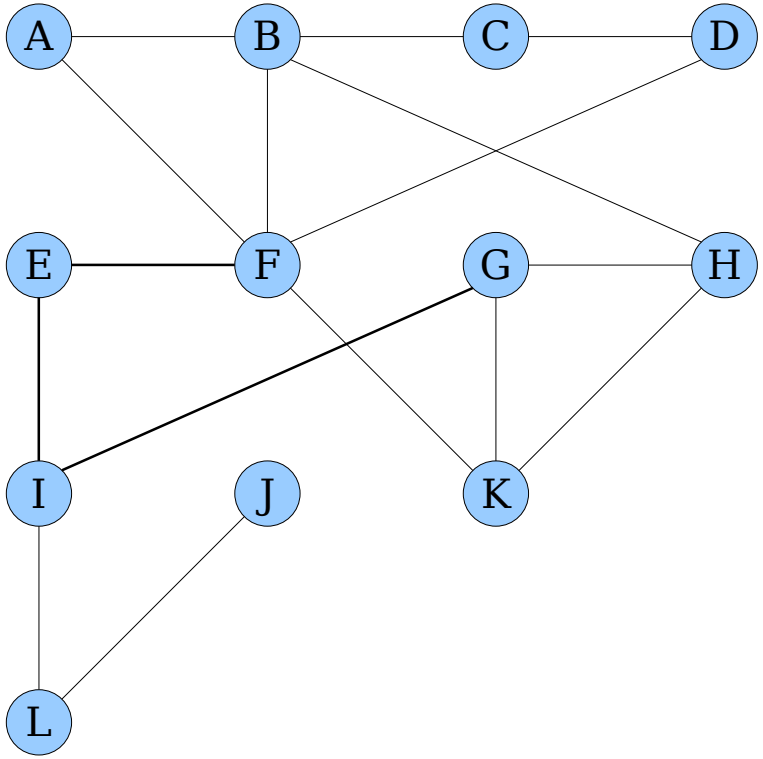
Final Exam Study Strategy

- Don't memorize things—either write it in notes, or learn the concept
  - If you've got flash cards, you're approaching this with the wrong mindset
  - No big multiple choice/true-false section where memorized facts would be tested
- Read the book and re-watch lectures (but **only** in a targeted way)
  - Computer science is about creating things, so do some practice problems
  - Re-do questions from lecture, do old section problems, do the practice exam
  - Practice problems from lecture and section, and CodeStepByStep.com are great to work through as well.
  - If you identify an area for improvement in doing the above, then re-watch lecture videos or read the book **as needed**
- Do the practice exam, preferably on paper
  - As with the midterm, the practice exam gives you practice not only with topics but with the logistics of writing an exam on paper

Breadth-First Search in a Graph

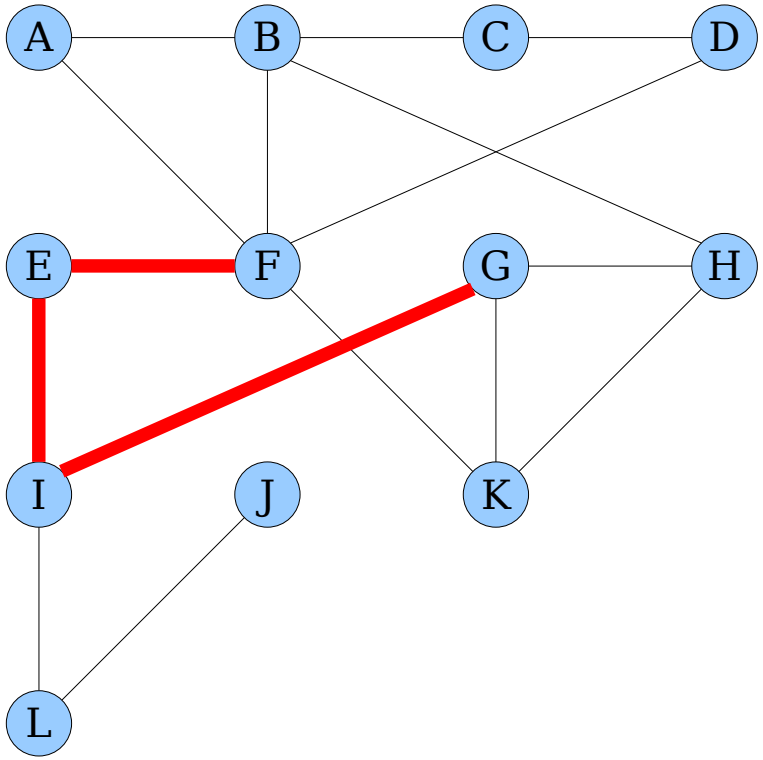
GRAPH ALGORITHMS

Breadth-First Search



BFS is useful for finding the <u>shortest path</u> between two nodes (in an unweighted, or equally-weighted graph).

Breadth-First Search

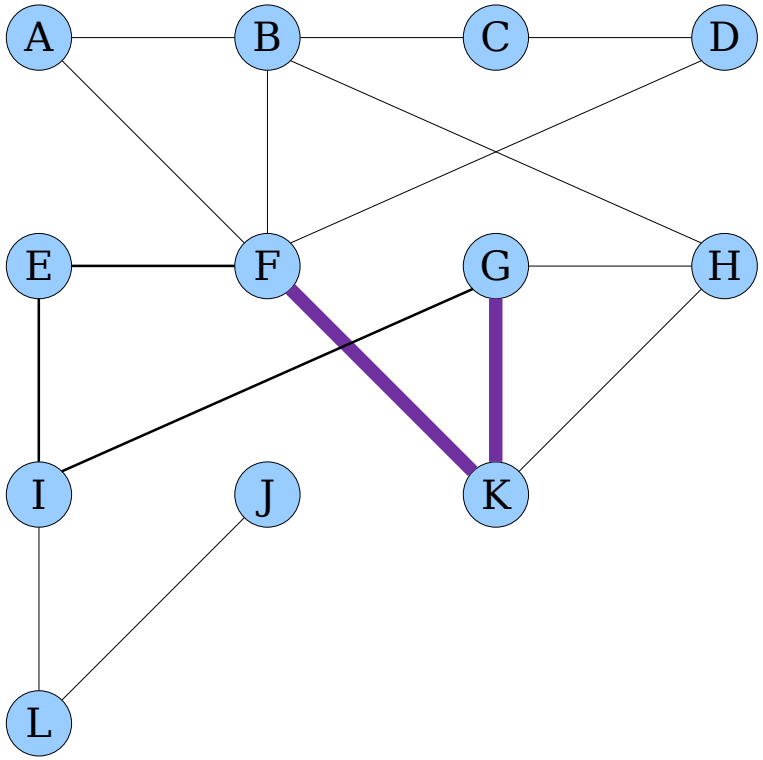


BFS is useful for finding the <u>shortest path</u> between two nodes (in an unweighted, or equally-weighted graph).

Example: What is the shortest way to go from F to G?

One way (not the shortest):
F → E → I → G **3 edges**

Breadth-First Search

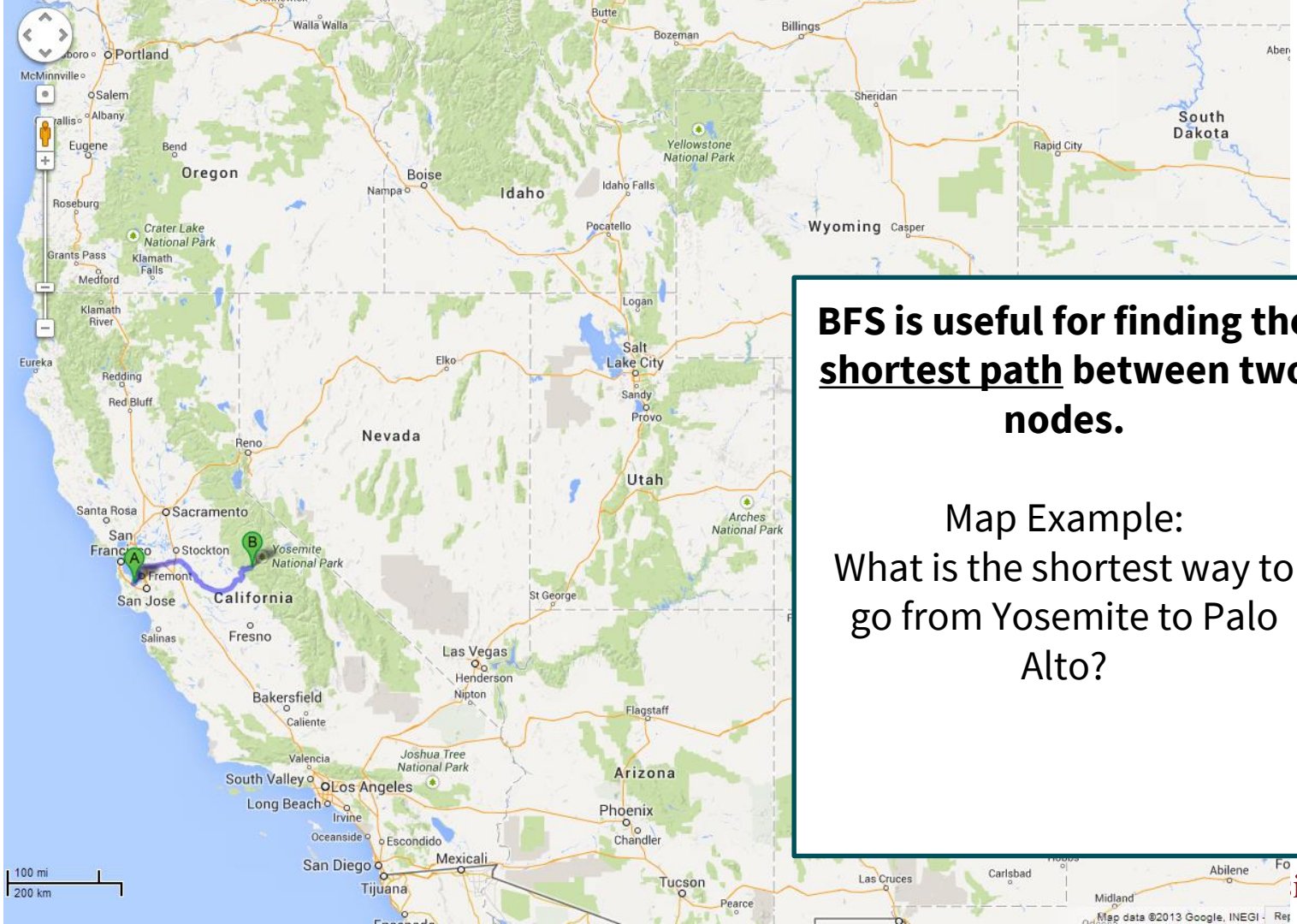


BFS is useful for finding the <u>shortest path</u> between two nodes (in an unweighted, or equally-weighted graph).

Example: What is the shortest way to go from F to G?

One way (not the shortest):
F → E → I → G **3 edges**

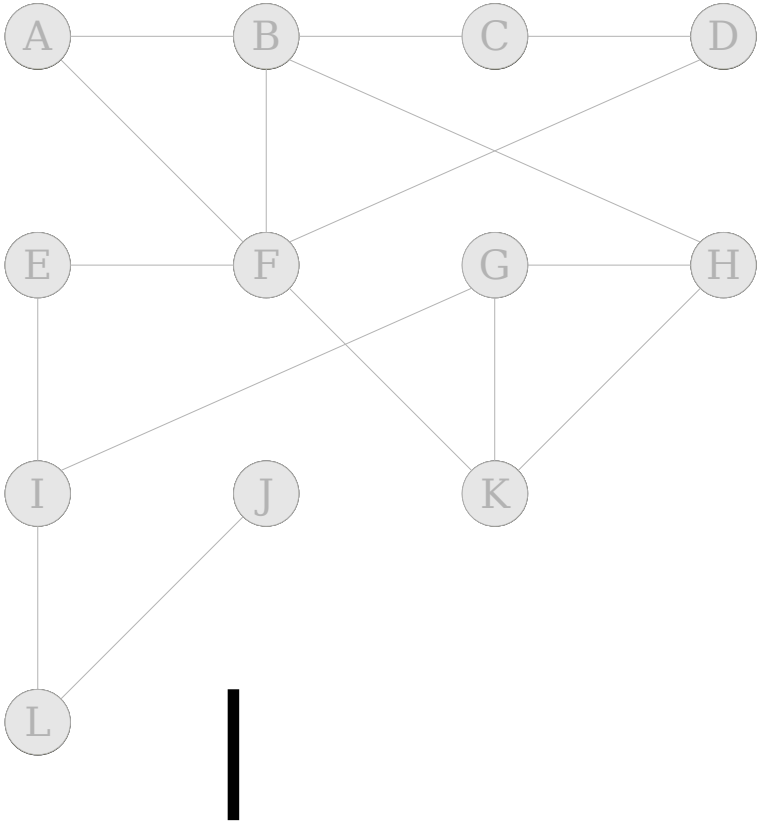
Shortest way:
F → K → G **2 edges**



BFS is useful for finding the <u>shortest path</u> between two nodes.

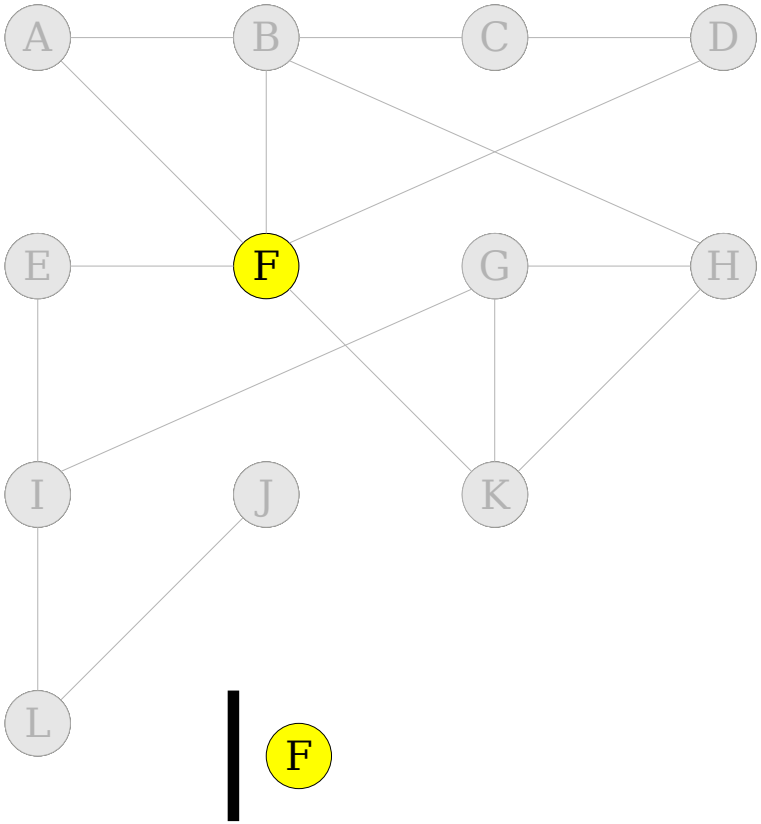
Map Example:
What is the shortest way to go from Yosemite to Palo Alto?

A BFS algorithm for graphs with a special property…



TO START:
(1)Color all nodes GREY to mean UNVISITED
(2)Queue is empty

A BFS algorithm for graphs with a special property…



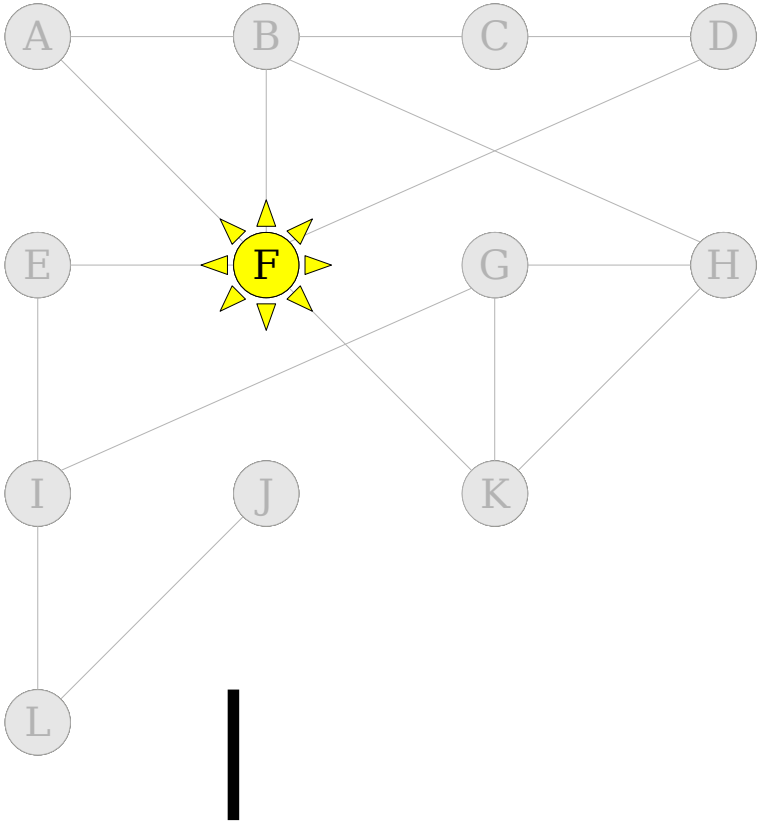
TO START:

(1) Color all nodes GREY to mean UNVISITED

(2) Queue is empty

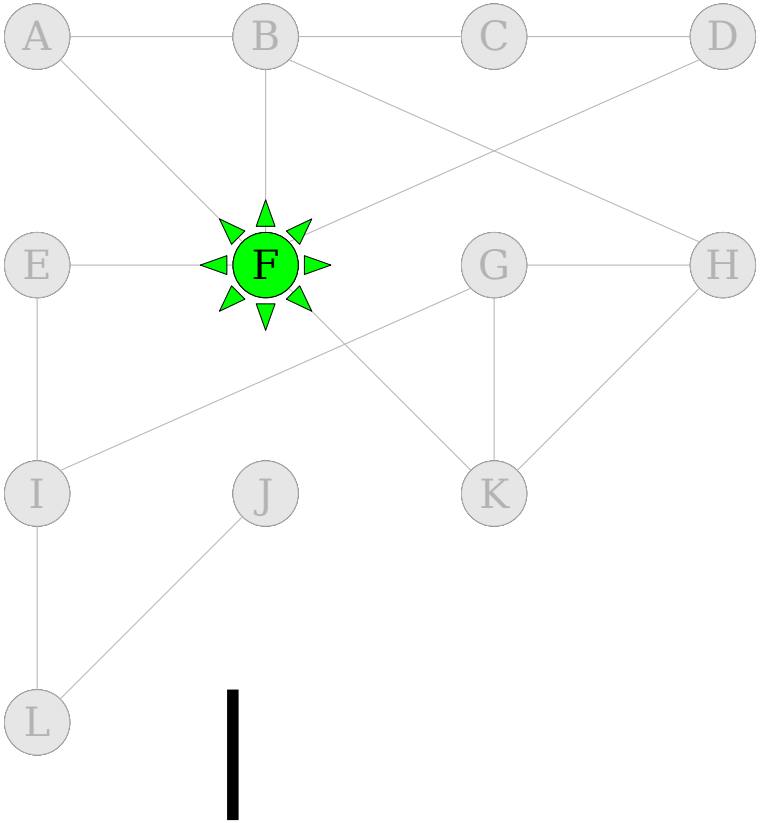
(3) Enqueue the desired **start** node, change its color to mark it VISITED

A BFS algorithm for graphs with a special property…

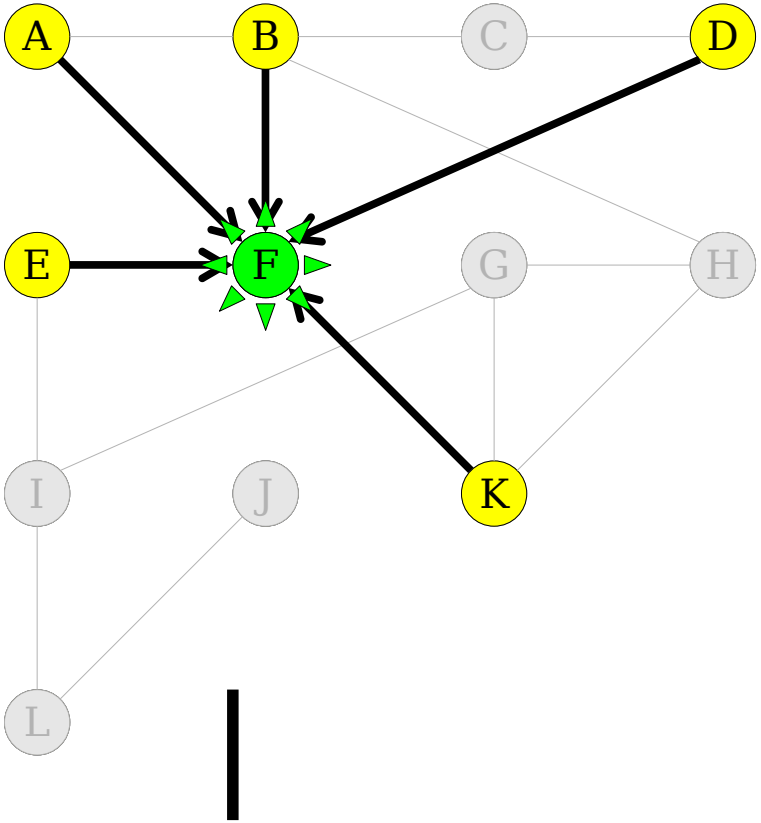


LOOP PROCEDURE:
(1)Dequeue a node
(2)Set current node's UNVISITED neighbors' parent pointers to current node, then enqueue them (and mark them visited when we enqueue)

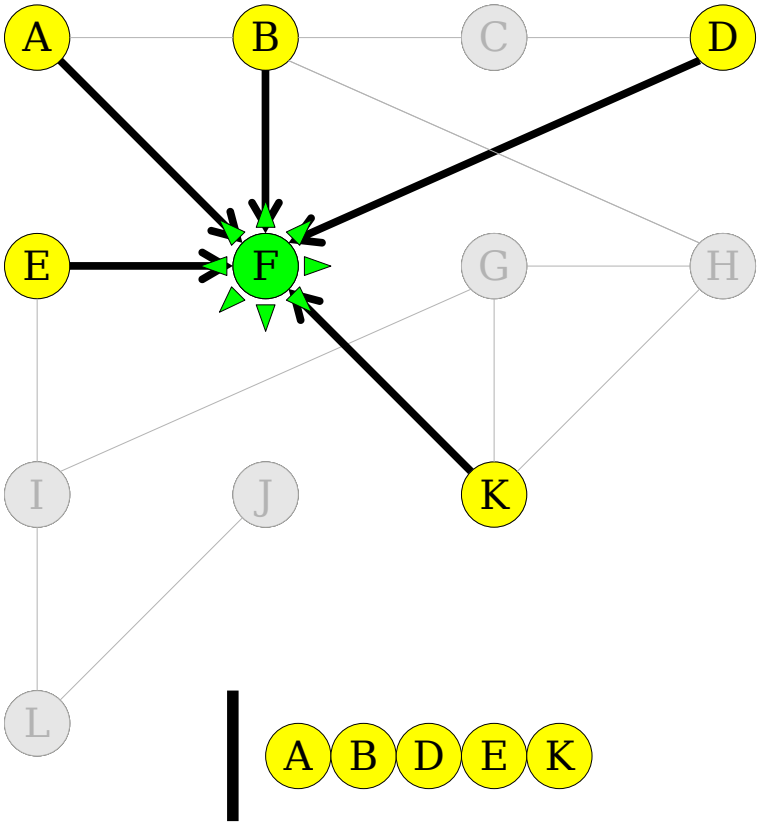
Breadth-First Search



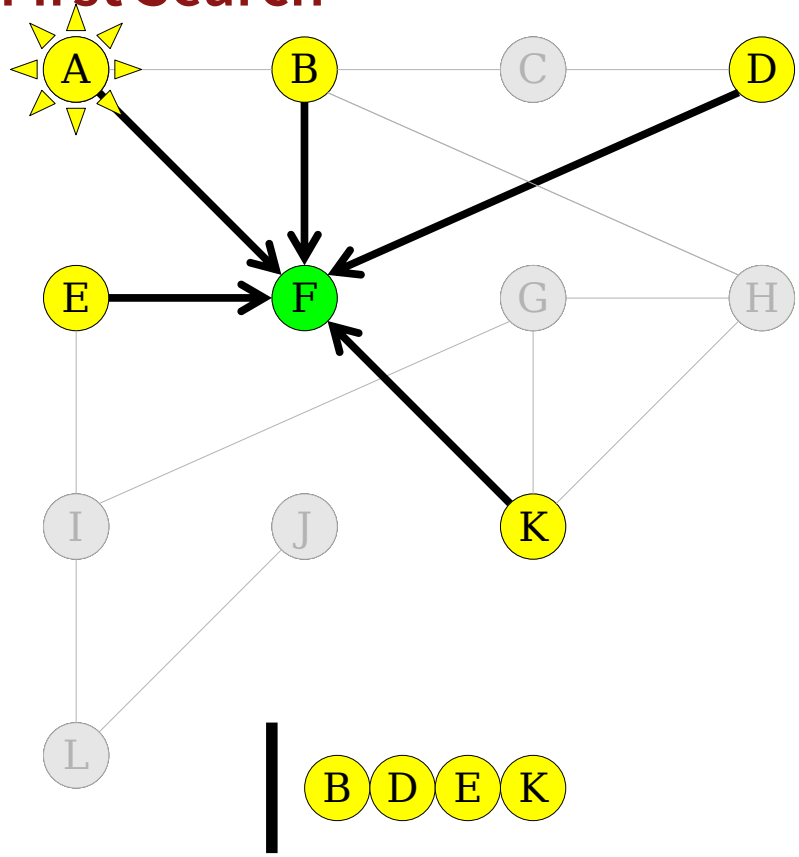
Breadth-First Search



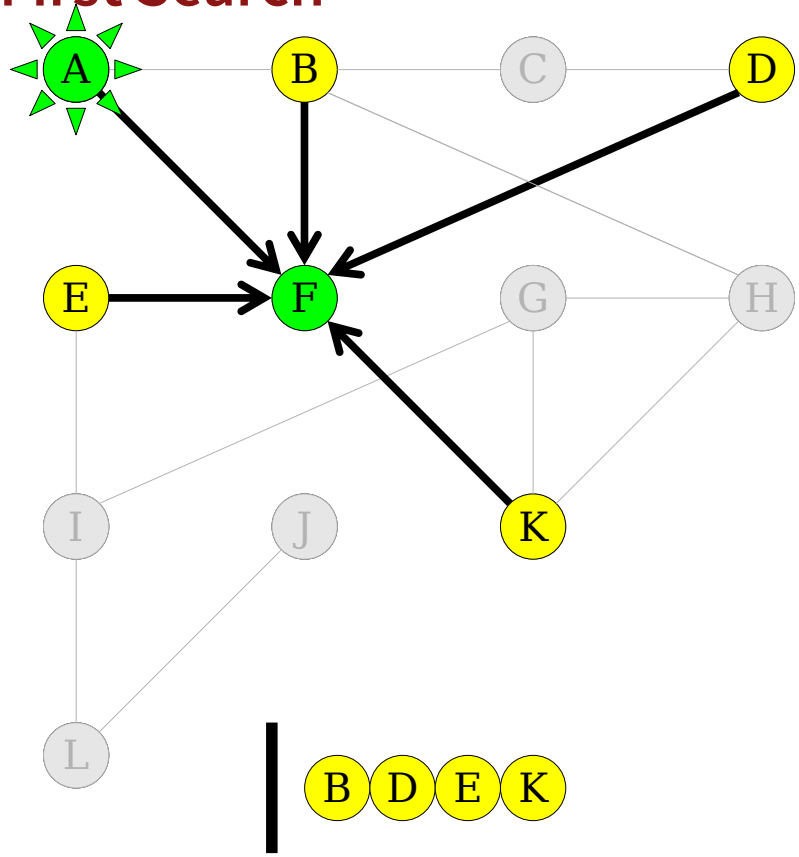
Breadth-First Search



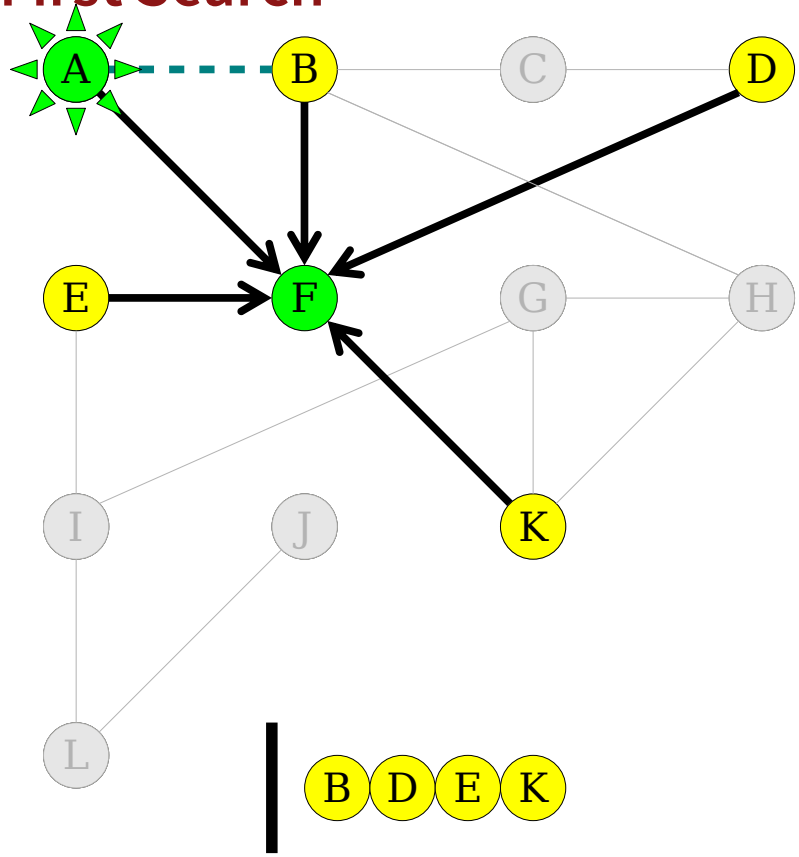
Breadth-First Search



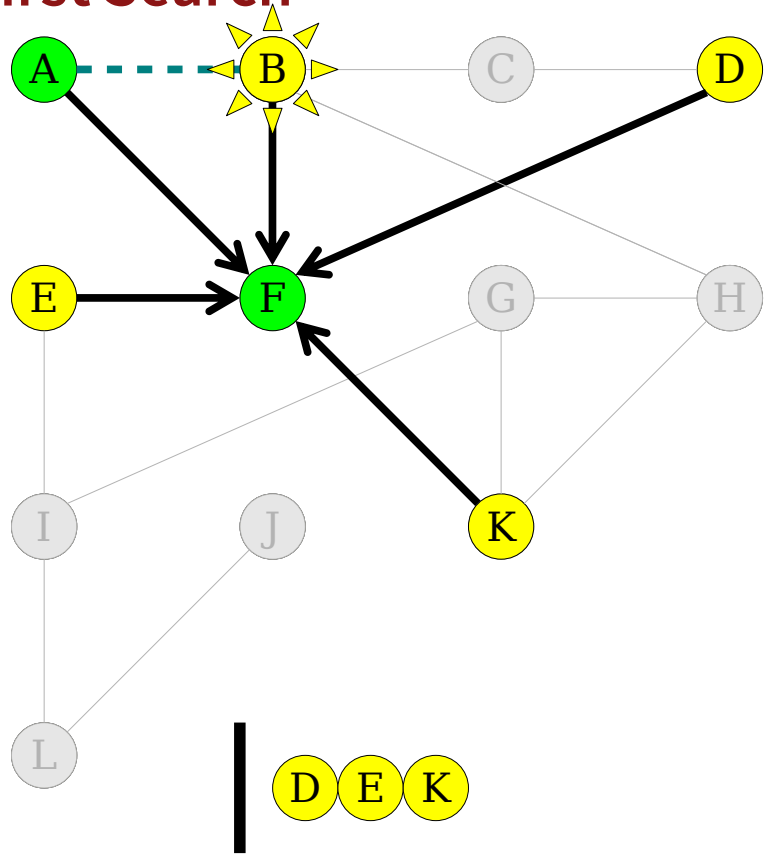
Breadth-First Search



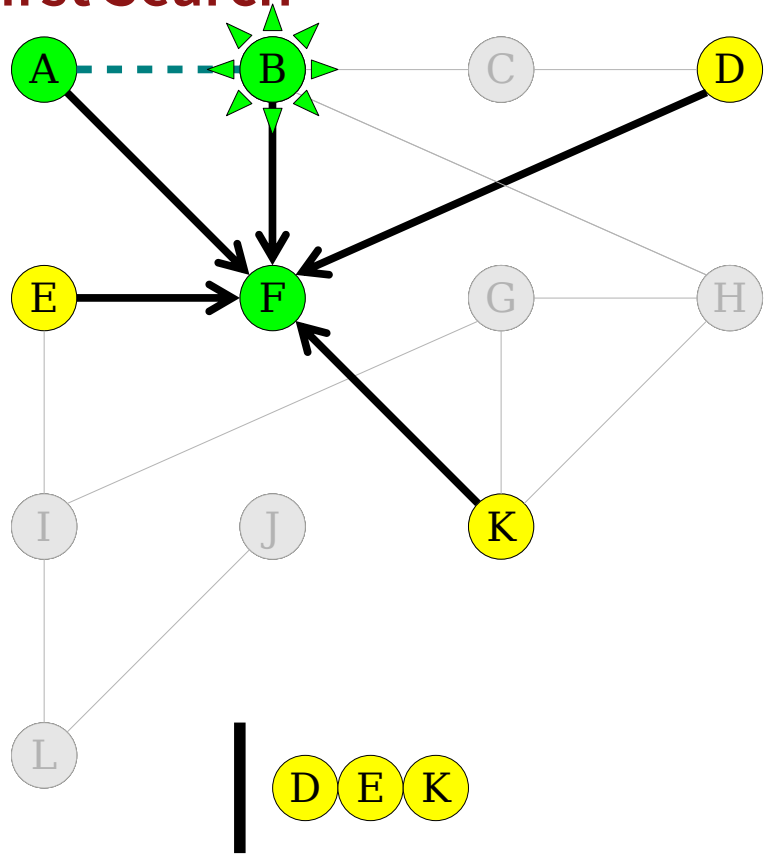
Breadth-First Search



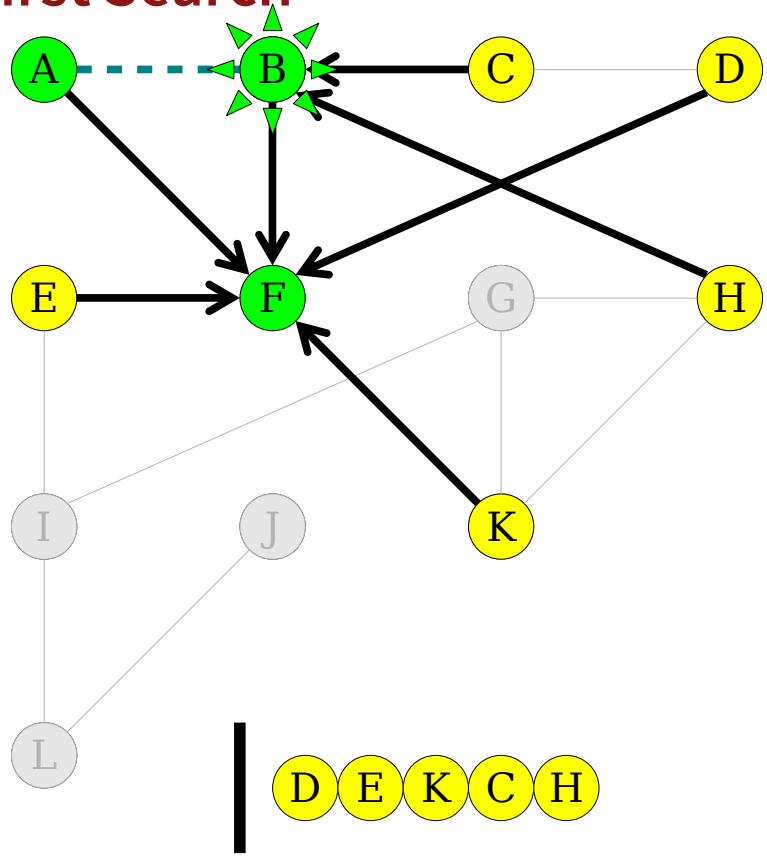
Breadth-First Search



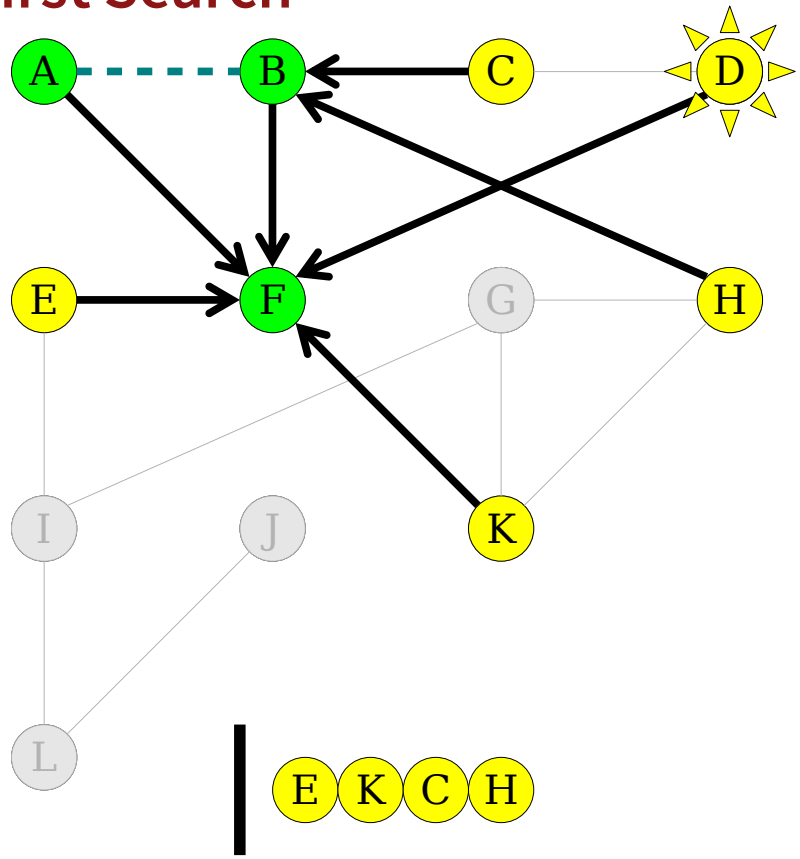
Breadth-First Search



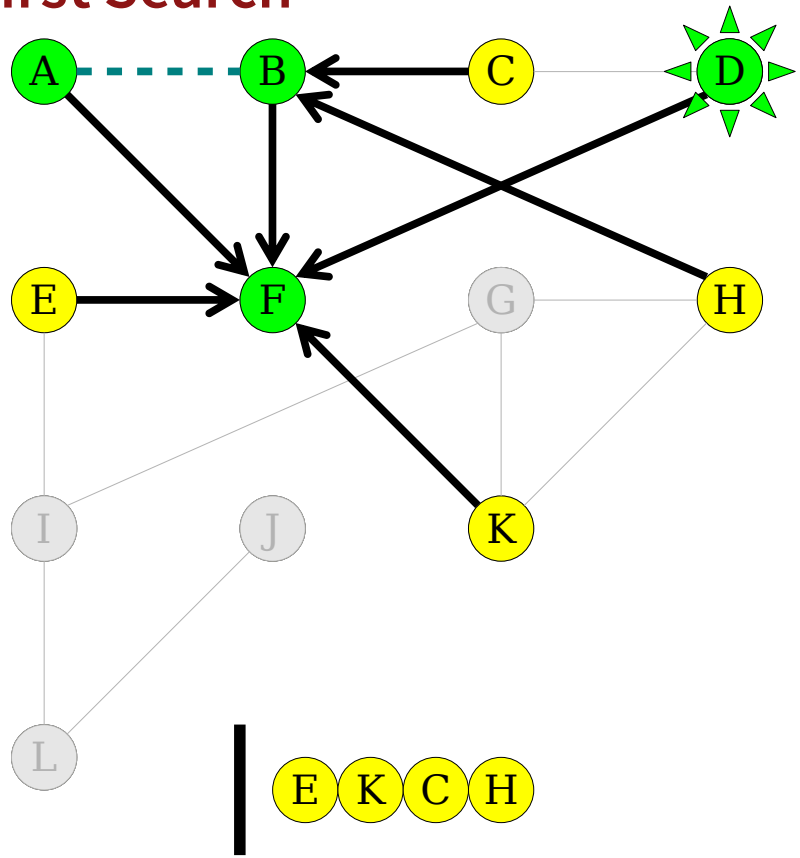
Breadth-First Search



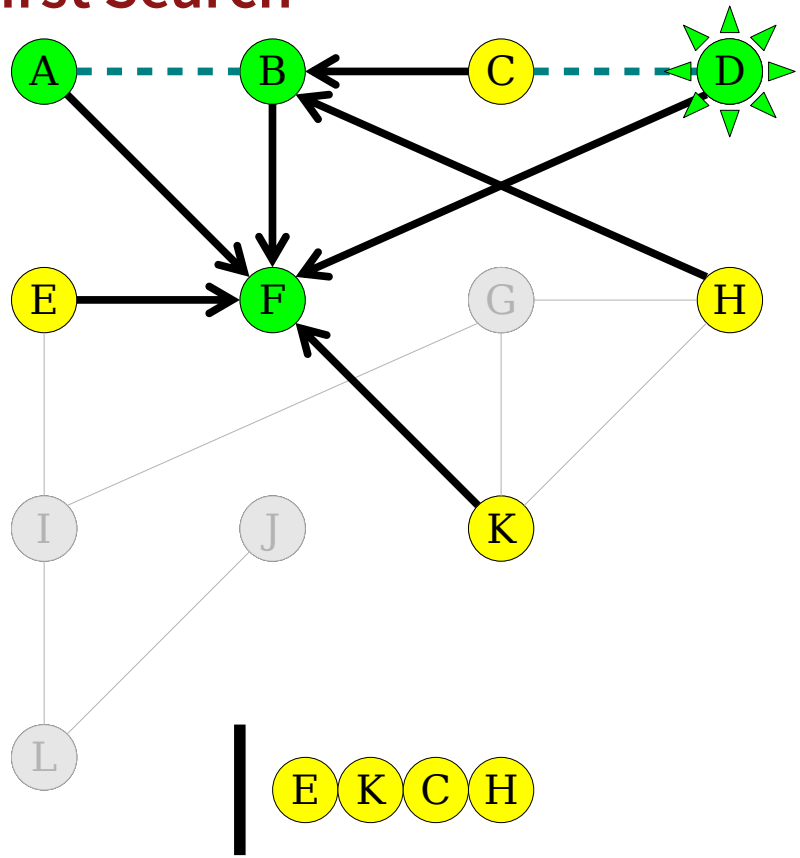
Breadth-First Search



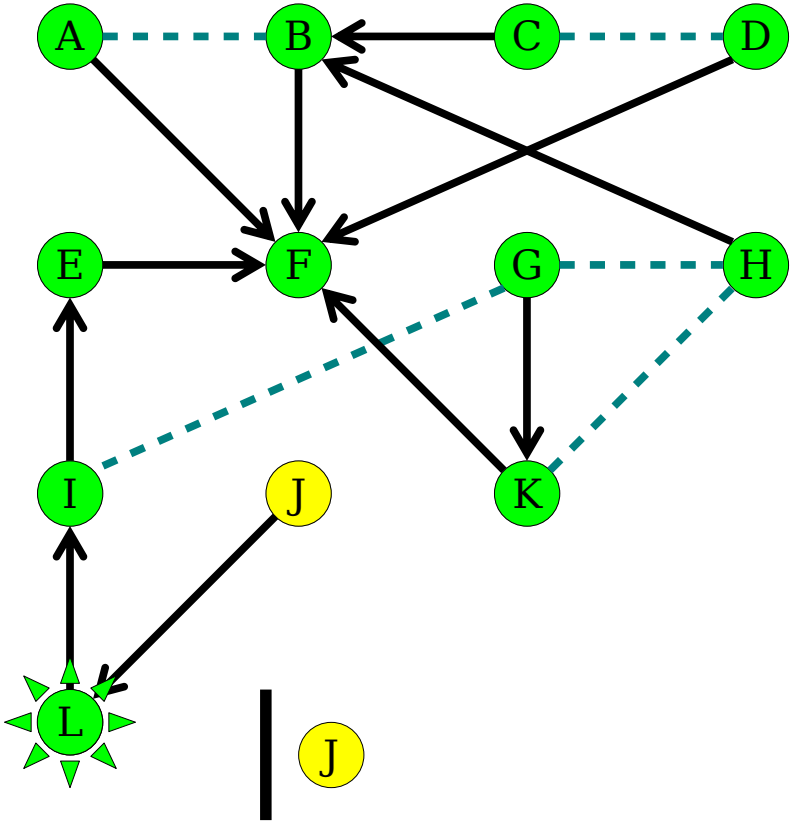
Breadth-First Search



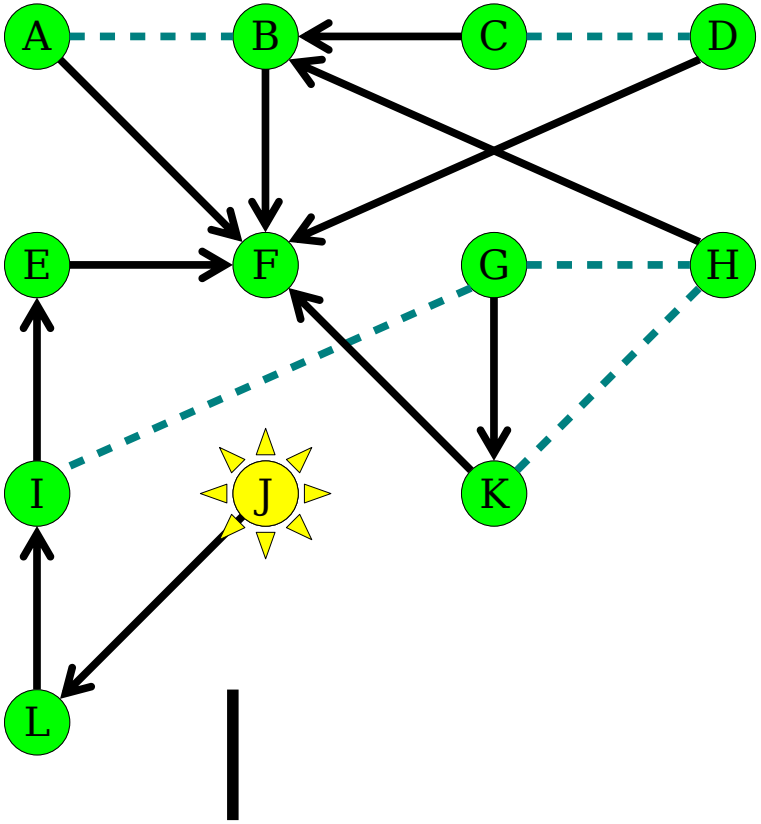
Breadth-First Search



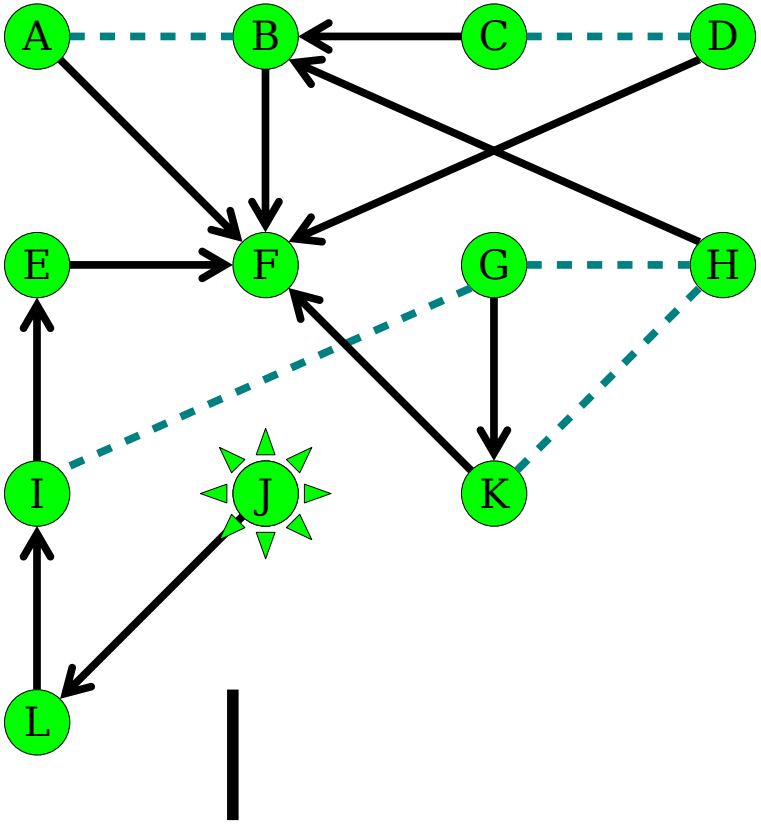
Breadth-First Search



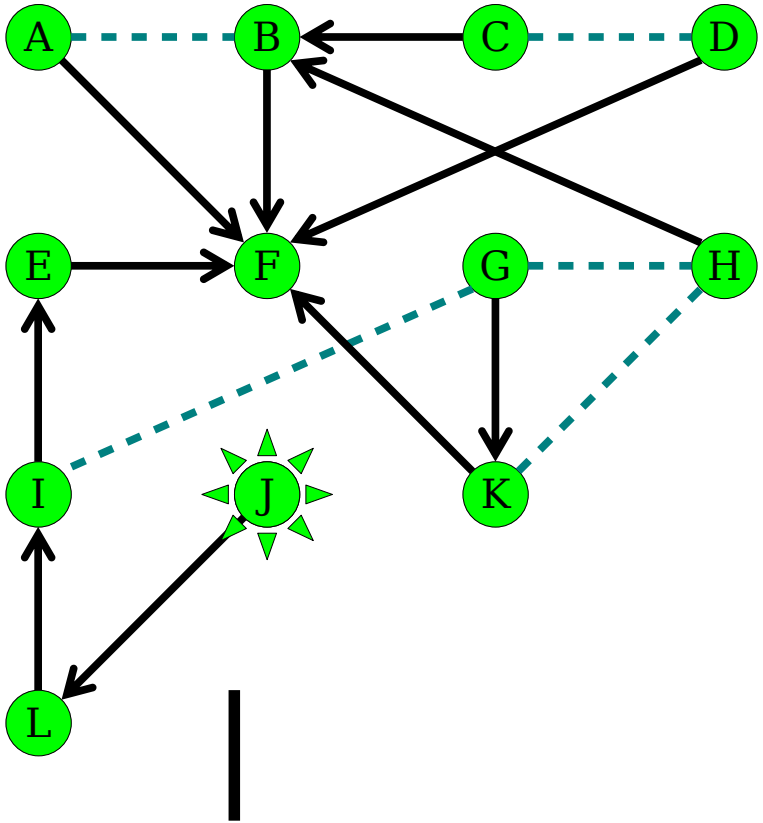
Breadth-First Search



Breadth-First Search



Breadth-First Search



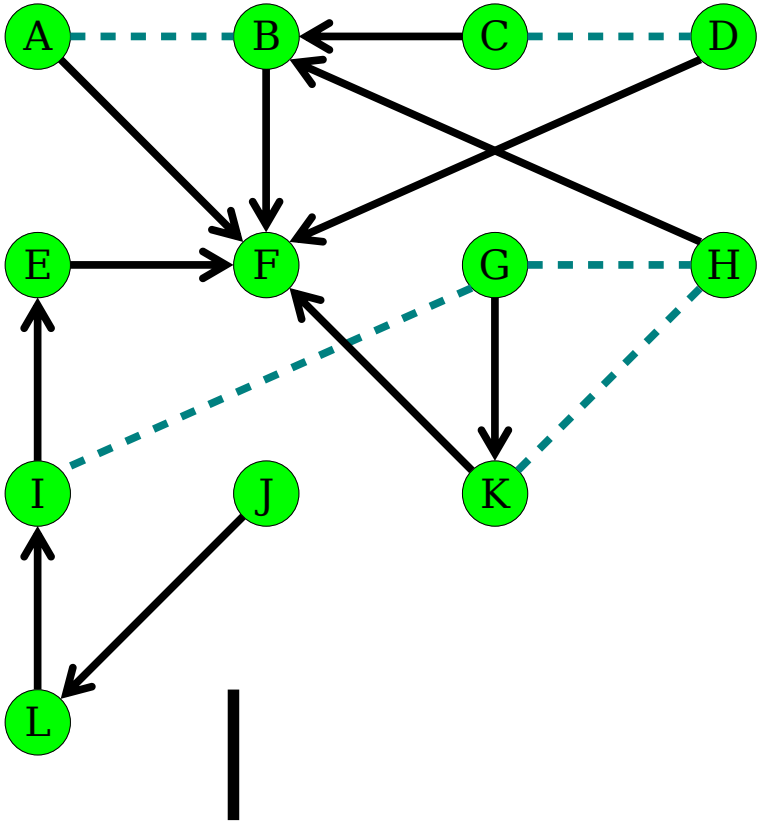
Done!

Now we know that to go from Yoesmite (F) to Palo Alto (J), we should go:

F->E->I->L->J
(4 edges)

(note we follow the parent pointers backwards)

Breadth-First Search



THINGS TO NOTICE:

(1) We used a queue

(2) What's left is a kind of subset of the edges, in the form of 'parent' pointers

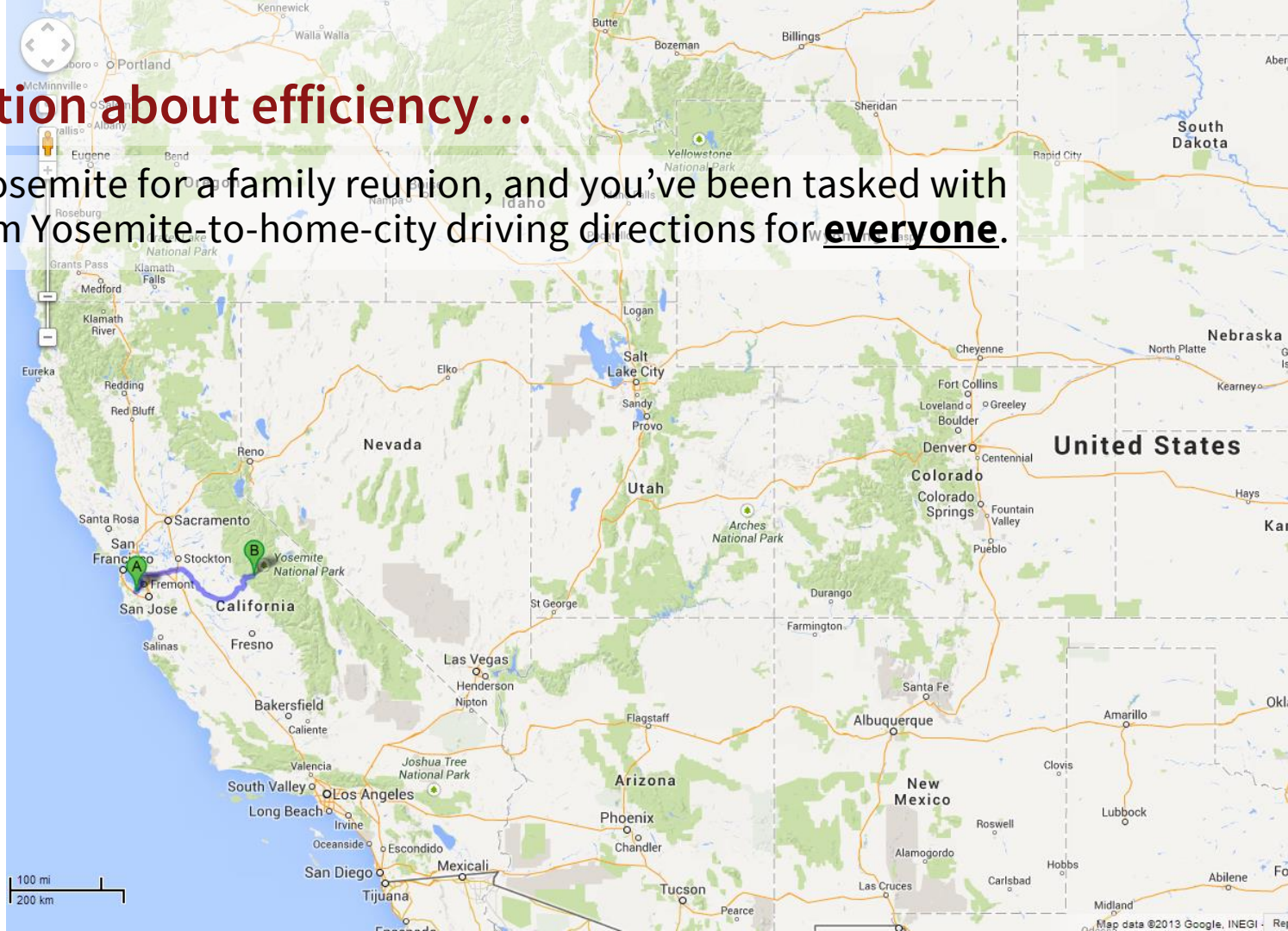
(3) If you follow the parent pointers from the desired end point, you will get back to the start point, and it will be the shortest way to do that

Quick question about efficiency…

Let's say that you have an extended family with somebody living in every major city in the western U.S.

Quick question about efficiency…

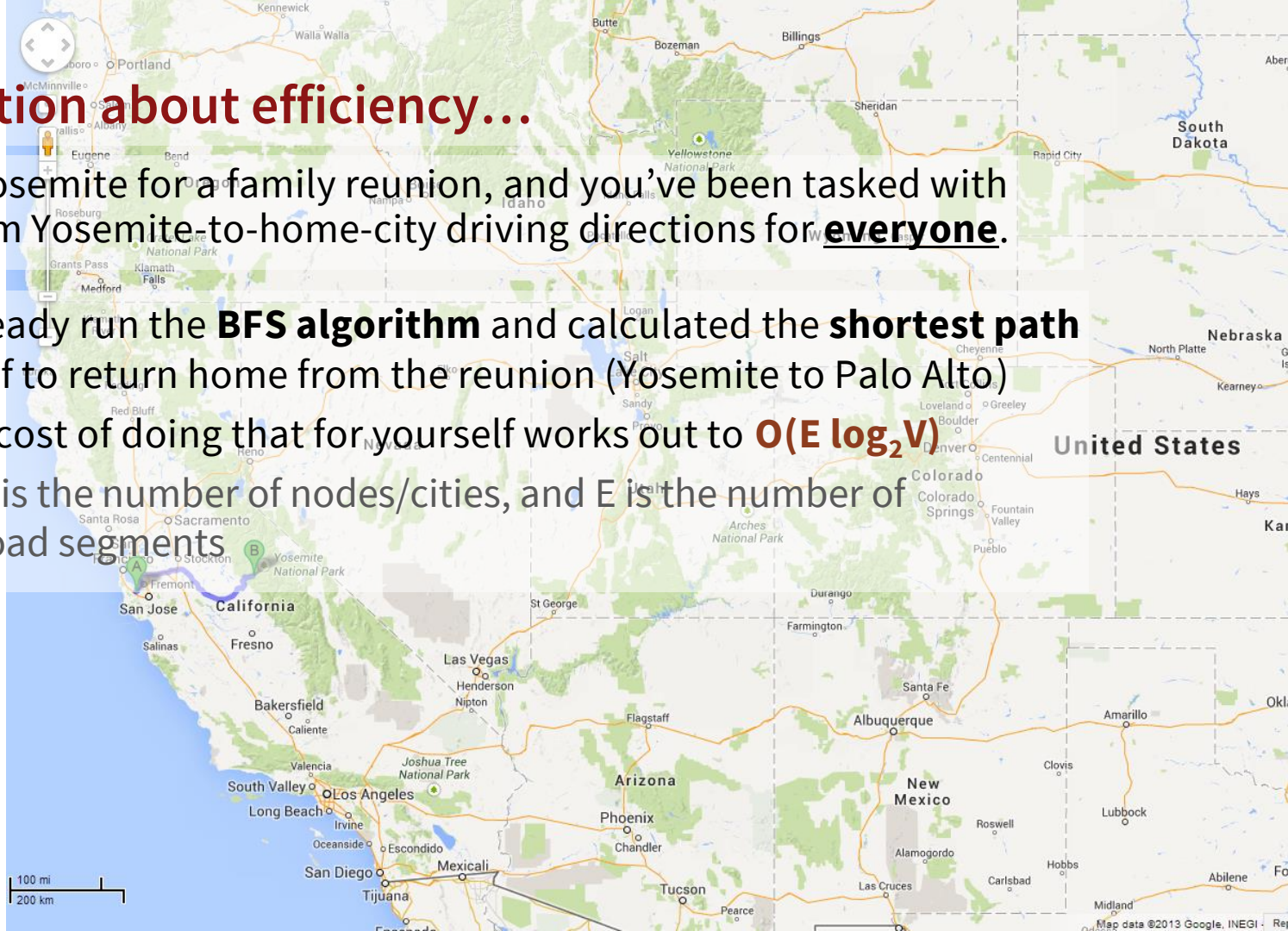
You're all in Yosemite for a family reunion, and you've been tasked with making custom Yosemite-to-home-city driving directions for **everyone**.



Quick question about efficiency…

You're all in Yosemite for a family reunion, and you've been tasked with making custom Yosemite-to-home-city driving directions for **everyone**.

- You've already run the **BFS algorithm** and calculated the **shortest path** for yourself to return home from the reunion (Yosemite to Palo Alto)
- The Big-O cost of doing that for yourself works out to **$O(E \log_2 V)$**
  - Where V is the number of nodes/cities, and E is the number of edges/road segments



Quick question about efficiency…

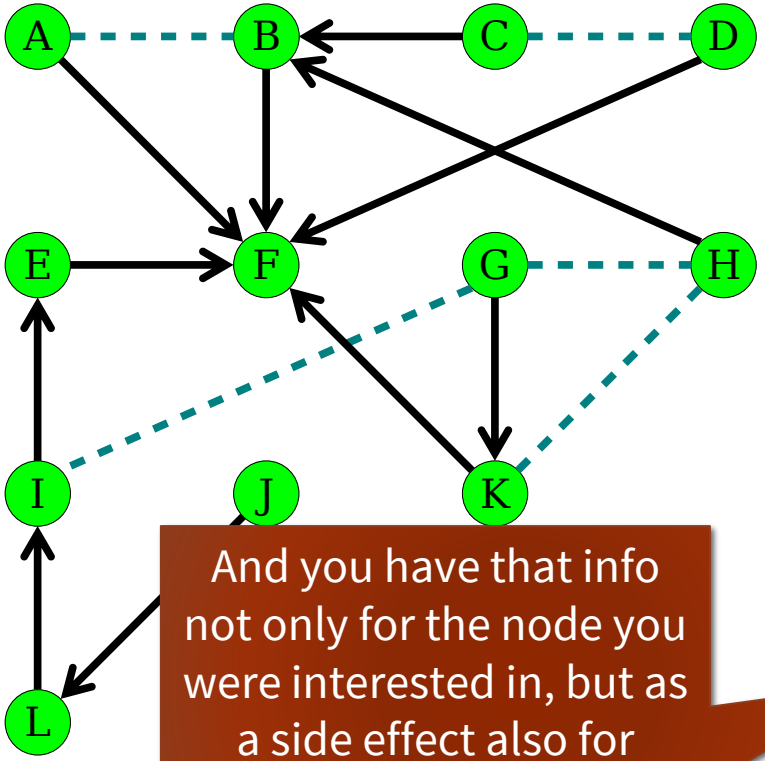
You're all in Yosemite for a family reunion, and you've been tasked with making custom Yosemite-to-home-city driving directions for **everyone**.

- You've already run the **BFS algorithm** and calculated the **shortest path** for yourself to return home from the reunion (Yosemite to Palo Alto)
- **$O(E \log_2 V)$** was the Big-O cost of doing that for yourself
  - Where **V** is the number of nodes/cities, and **E** is the number of edges/road segments

Your Turn: How long will it take you, in total, to calculate the shortest paths for you and all of your relatives?

A. $O(\mathbf{V}E \log_2 V)$
B. $O(E \log_2 \mathbf{V^2})$
C. $O(\mathbf{V} \log_2 \mathbf{E})$
D. $O(E \log_2 V)$
E. Something else

Breadth-First Search



And you have that info not only for the node you were interested in, but as a side effect also for every node in the graph!

THINGS TO NOTICE:

(1) We used a queue

(2) What's left is a kind of subset of the edges, in the form of 'parent' pointers

(3) If you follow the parent pointers from the desired end point, you will get back to the start point, and it will be the shortest way to do that

Quick question about efficiency…

You're all in Yosemite for a family reunion, and you've been tasked with making custom Yosemite-to-home-city driving directions for **everyone**.

- You've already run the **BFS algorithm** and calculated the **shortest path** for yourself to return home from the reunion (Yosemite to Palo Alto)
- **$O(E \log_2 V)$** was the Big-O cost of doing that for yourself
  - Where **V** is the number of nodes/cities, and **E** is the number of edges/road segments

Your Turn: How long will it take you, in total, to use BFS to calculate the shortest paths for you and all of your relatives?

A. $O(\mathbf{V}E \log_2 V)$
B. $O(E \log_2 \mathbf{V^2})$
C. $O(\mathbf{V} \log_2 \mathbf{E})$
D. $O(E \log_2 V)$
E. Something else

No additional work for BFS to determine the shortest paths for all your relatives, vs just for yourself!

Big O Quick Reference (see also http://bigocheatsheet.com/)

What	Cost
• Hash table average case (good design)	O(1)
• Balanced trees • Heap, BST with balancing such as Red-Black • Binary search on sorted array	O(logn)
• Linked list find • Inserting into beginning of array/Vector • Hash table worst case • Unbalanced tree (e.g. unbalanced BST) worst case	O(n)
• Good sorting • Mergesort, Heapsort, Quicksort (expected)	O(nlogn)
• Bad sorting • Insertion, Bubble, Selection, Quicksort (worst case)	$O(n^2)$

Quarter Wrap-Up

WHAT DID WE SET OUT TO DO IN THE BEGINNING?

WHERE ARE WE NOW?

Goals for this Course

- **Learn how to model and solve complex problems with computers.**
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Bring added rigor to your understanding of algorithmic performance, so you can quantitatively compare approaches for solving problems.

From here on out, there are no obvious answers to any problem worth your hourly rate. ☺

- Programming is all about exploring new ways to **model** and **solve** problems.
- There are **choices** and **tradeoffs** in how we model these and how we implement them!
- Skilled computer scientists recognize that any problem worth tackling has *many* possible models and *many* possible solutions, often none of which is clearly better than the others in all dimensions
  - Array or linked list? BST or hash table?
  - **Tradeoffs!**

That's a lot of material to cover in 10 weeks

You are part of a very challenging course,
in the best CS department *in the world*,
and you are so, so close to completing this course!

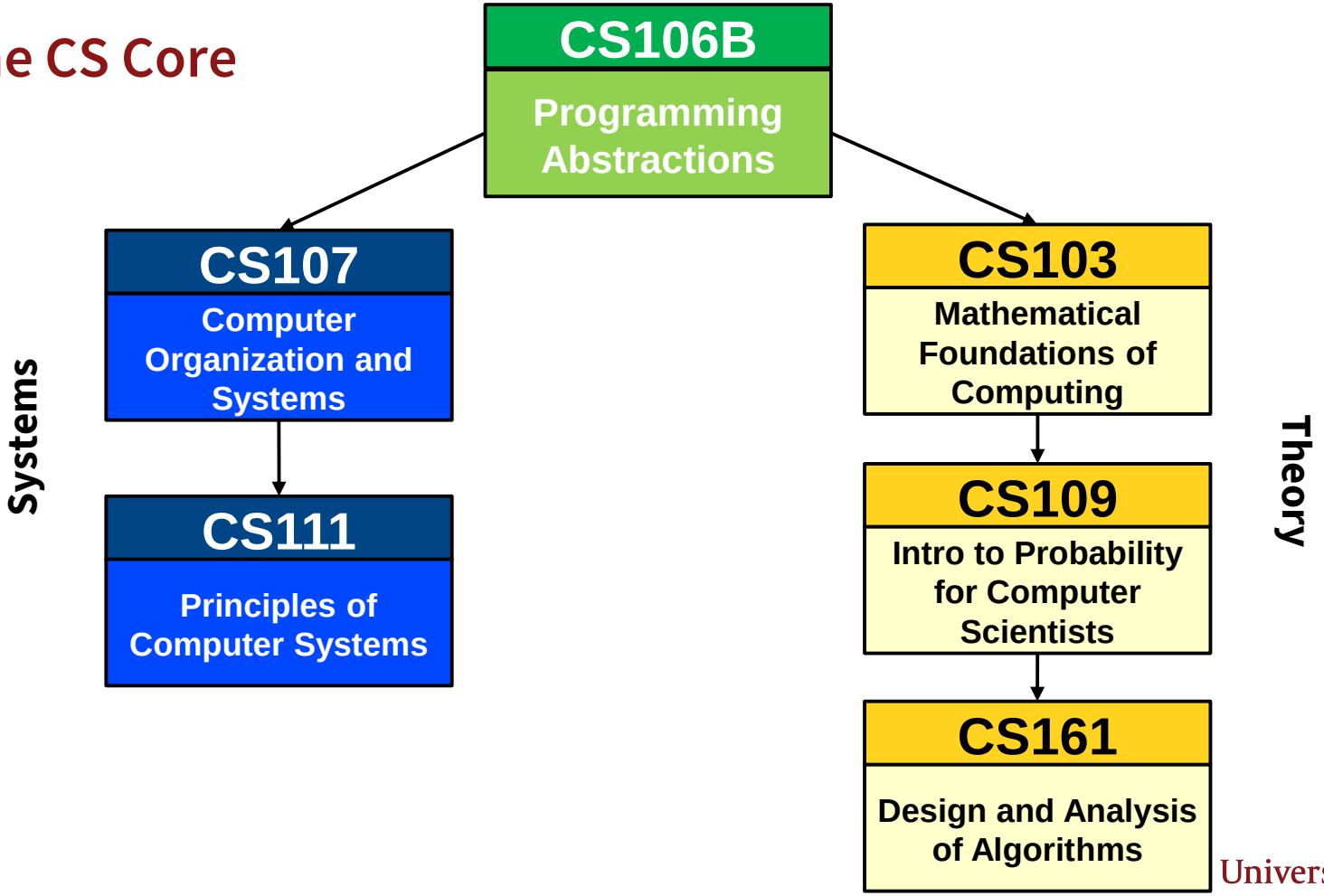
Congratulations!!
You've almost made it through CS106B!

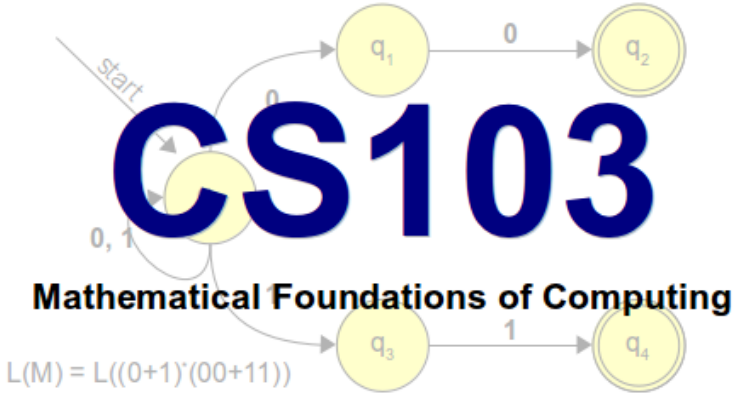
- *So…what next?*

What next?

CS OPTIONS AFTER CS106B

The CS Core


CS106B
Programming Abstractions
CS107
Computer Organization and Systems
CS111
Principles of Computer Systems
Systems
CS103
Mathematical Foundations of Computing
CS109
Intro to Probability for Computer Scientists
CS161
Design and Analysis of Algorithms
Theory



CS103

Mathematical Foundations of Computing

Can computers solve all mathematical problems?

Spoiler: no!

Why are some problems harder than others?

We can do find in an unsorted array in O(N), and we can sort an unsorted array in O(NlogN). Is sorting just inherently a harder problem, or are there better O(N) sorting algorithms yet to be discovered?

How can we be certain about this?

CS107
Computer Organization and Systems

How do we encode text, numbers, programs, etc. using just 0s and 1s?

Where does memory come from? How is it managed?

How do compilers, debuggers, etc. work?

CS107 in the news: Heartbleed

- In April 2014, security experts warned that users of thousands of websites, including many crucial apps such as banking, shopping, needed to change their passwords due to potential exposure caused by the “Heartbleed” vulnerability

- Heartbleed exploited a **buffer overrun** bug in OpenSSL
  - SSL is the layer that secures web interactions, i.e., it’s what makes the “s” in “https://” mean something

CS107 in the news: Heartbleed

- The protocol allows you to send "heartbeat" messages, which basically say:
  - *Are you still there? If you are, repeat this message back to me:*
    - *"hello"*
  - *The length of the message I want you to send back to me is this many bytes:*
    - *5 bytes*
- (Recall each char is one byte)

CS107 in the news: Heartbleed

- The protocol allows you to send "heartbeat" messages, which basically say:
  - *Are you still there? If you are, repeat this message back to me:*
    - *"hello"*
  - *The length of the message I want you to send back to me is this many bytes:*
    - *5 bytes*
- (Recall each char is one byte)

- The server code has a loop that uses the requested number of bytes to send the requested echo message back, something like this (not the actual code, but the same concept):

```
char *reply = getReplyMsg(); // "hello"
int n_bytes = getReplyLen(); // 5
string newMsg = "Yes alive here's proof: ";
for (int i = 0; i < n_bytes; i++) {
    newMsg += reply[i];
}
sendReply(newMsg);
```

CS107 in the news: Heartbleed

- The protocol allows you to send "heartbeat" messages, which basically say:
  - *Are you still there? If you are, repeat this message back to me:*
    - *"hello"*
  - *The length of the message I want you to send back to me is this many bytes:*
    - *5 bytes*
- (Recall each char is one byte)

Your Turn: why would this code potentially compromise the server's security and data?

- The server code has a loop that uses the requested number of bytes to send the requested echo message back, something like this (not the actual code, but the same concept):

```
char *reply = getReplyMsg(); // "hello"
int n_bytes = getReplyLen(); // 5
string newMsg = "Yes alive here's proof: ";
for (int i = 0; i < n_bytes; i++) {
    newMsg += reply[i];
}
sendReply(newMsg);
```

Think about this loop condition

CS107 in the news: Chrome

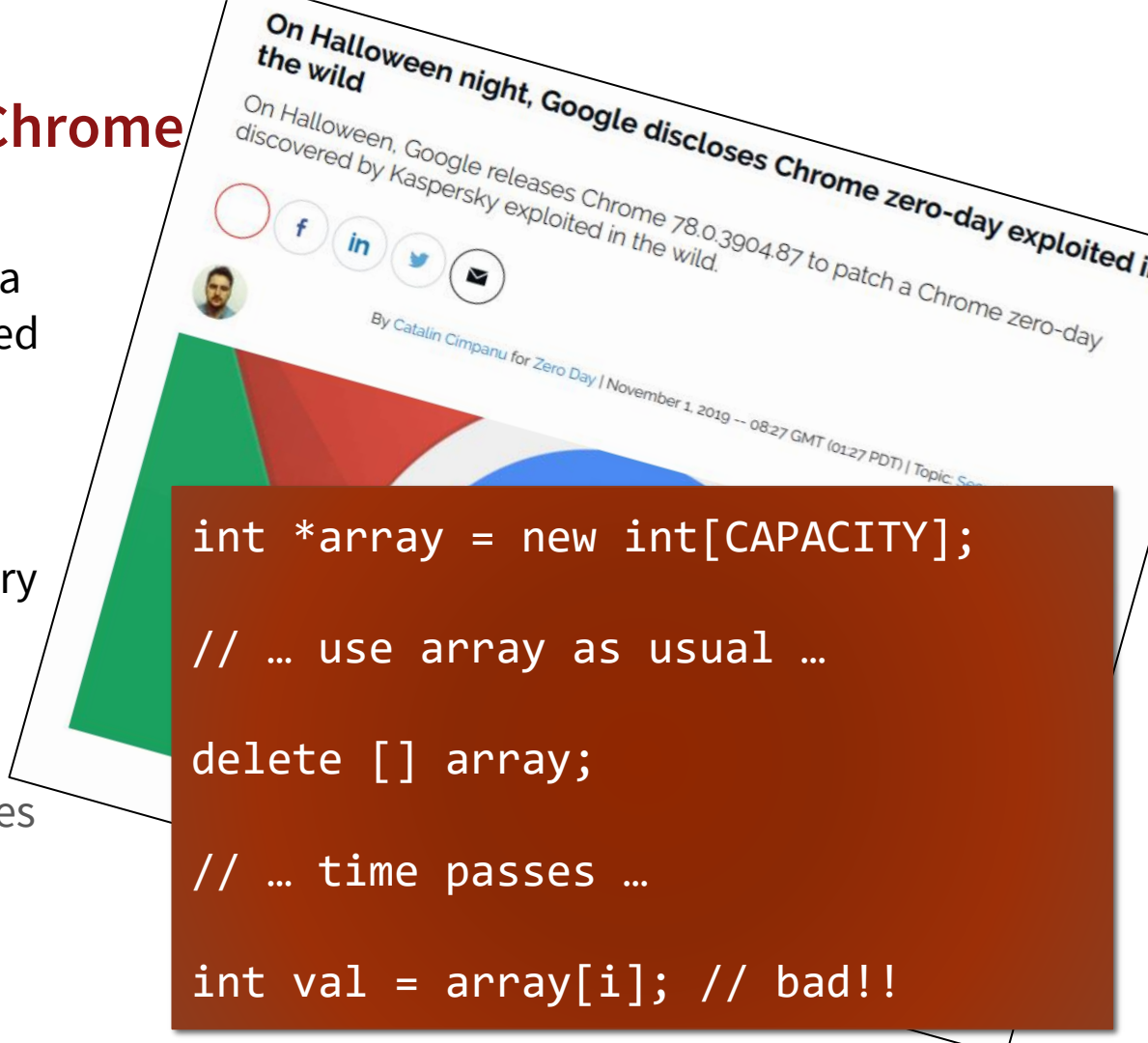
- On Oct 31, 2019, Google disclosed that there was a bug in Chrome that caused a security breach

- The bug was that the program accesses memory after it has already been freed/deleted
  - Usually works, but incorrect and sometimes causes the bug

CS107 in the news: Chrome

- On Oct 31, 2019, Google disclosed that there was a bug in Chrome that caused a security breach

- The bug was that the program accesses memory after it has already been freed/deleted
  - Usually works, but incorrect and sometimes causes the bug



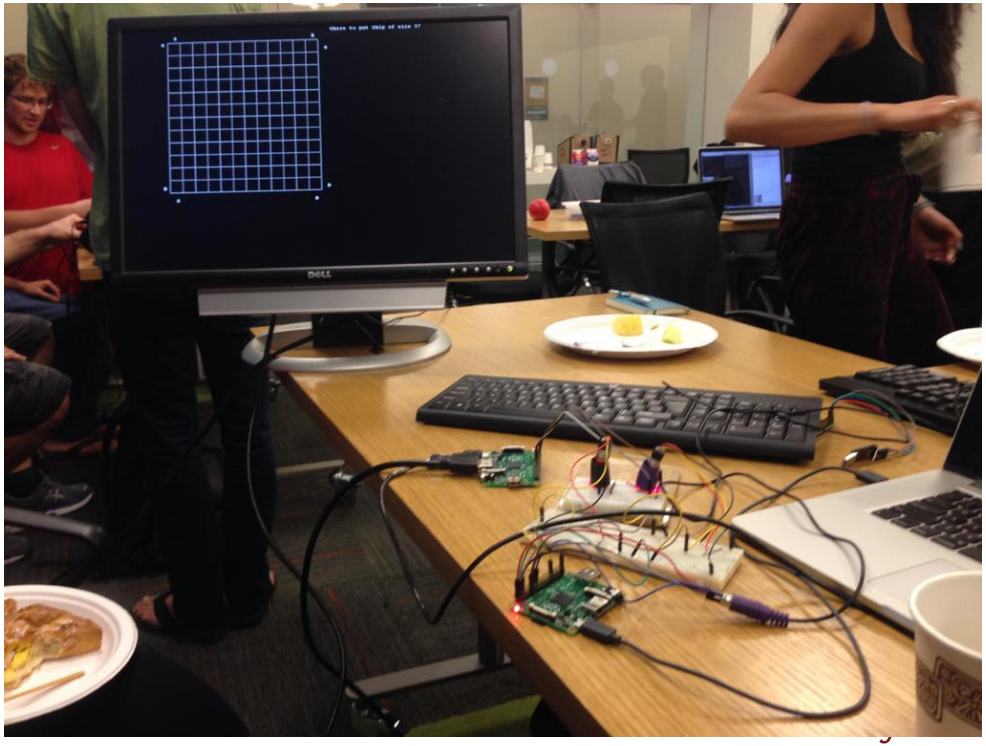

```
int *array = new int[CAPACITY];

// … use array as usual …

delete [] array;

// … time passes …

int val = array[i]; // bad!!
```

What CS103 and CS017 Aren't

- CS107-style systems programming is ***one*** kind of programming.
  - One that I really love! But one of many, and it's a matter of personal taste.
- Neither CS107 nor CS103 is a litmus test for whether you can be a computer scientist.
- Neither CS107 nor CS103 is indicative of what being a computer scientist is "really like."
  - CS107 does a lot of low-level programming. You don't have to do low-level programming to be a good computer scientist.
  - CS103 does a lot of proof-writing. You don't have to do proof-writing to be a good computer scientist.

CS107E
Computer Organization and Systems—*Embedded*

- **Counts for prerequisites etc. the same as original CS107**, but covers the topics with a twist: embedded work on Raspberry Pi



Other CS Courses

CS106L
Learning the Standard Template Library (STL)

- In CS106B, we learn the Stanford Library containers
- ***Now learn the industrial-strength ones!***

CS182
Computers, Ethics, and Public Policy

- Did you love the thought-provoking issues raised in the embedded ethics segments in this class?
- Want to dive deeper into that topic and more, and be prepared to participate in urgent, ongoing conversations in media, government, non-profits, and the tech industry?

We have the power to control and create technology, but how should we use it? Who should get a say in how it's used?

Options besides CS Major

CS Minor: only 5 more classes!

- 103, 107, 109, two your choice—fun!

CS Coterminal MS degree

- Earn an MS in CS while you are here earning your BS
- Possible for CS majors **and** other majors
  - ex: Psych major, CS co-term

See you Wednesday!