# Programming Abstractions

CS106B

Cynthia Bailey
Chris Gregg

pollev.com/cs106b

# Today's Topics

Previously:

- Graphs
  - Terminology
  - Basics
  - A couple samples of classic Graph Theory problems (Hamiltonian Path and the Good Will Hunting movie problem)

Today:

- Another classic graph problem: shortest paths
  - BFS
  - Dijkstra's algorithm

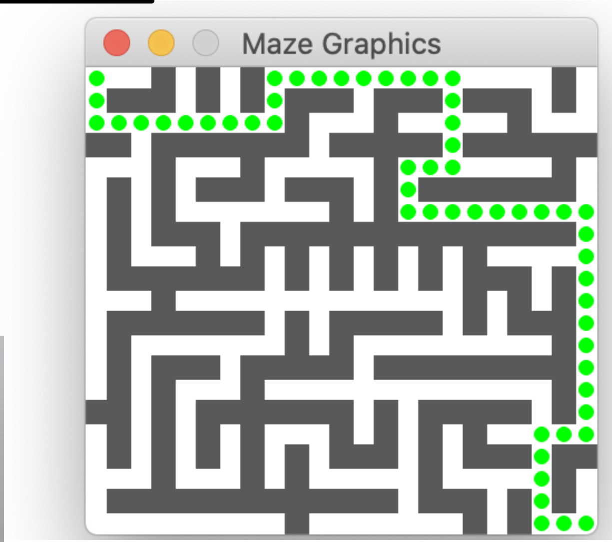
- **For important announcements, be sure to see the weekly announcements post on the Ed Q&A board! [https://edstem.org](https://edstem.org)**
- **Also on Ed: live lecture Q&A with Chris & Jonathan**

# Breadth-First Search

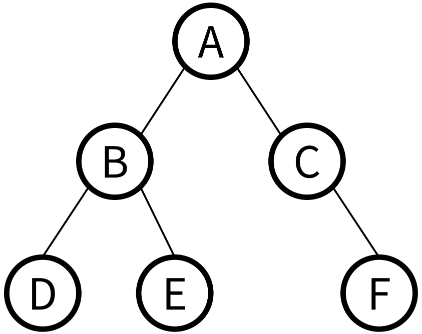
WE'VE SEEN BFS BEFORE THIS QUARTER!

# BFS in this class so far

**Assignments**




**Slime Mold**




**Trees**

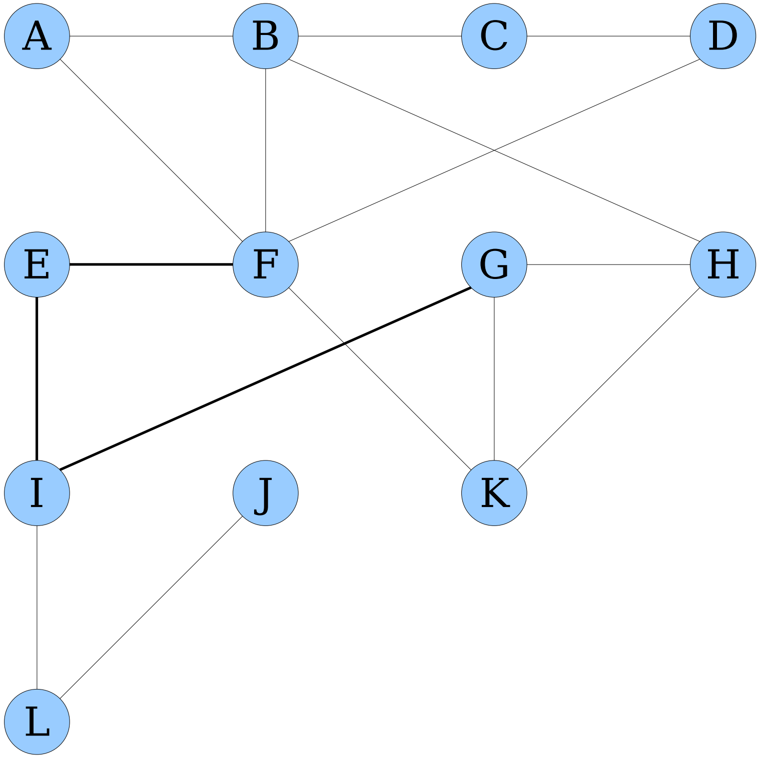
# Generic BFS algorithm pseudocode

1. Make an empty queue to store places we want to visit in the future
2. Enqueue the starting location
3. While the queue is not empty (and/or until you reach a desired destination):
   - Dequeue a location
   - Mark that location as visited
   - Enqueue all the neighbors of that location

# Breadth-First Search in a Graph

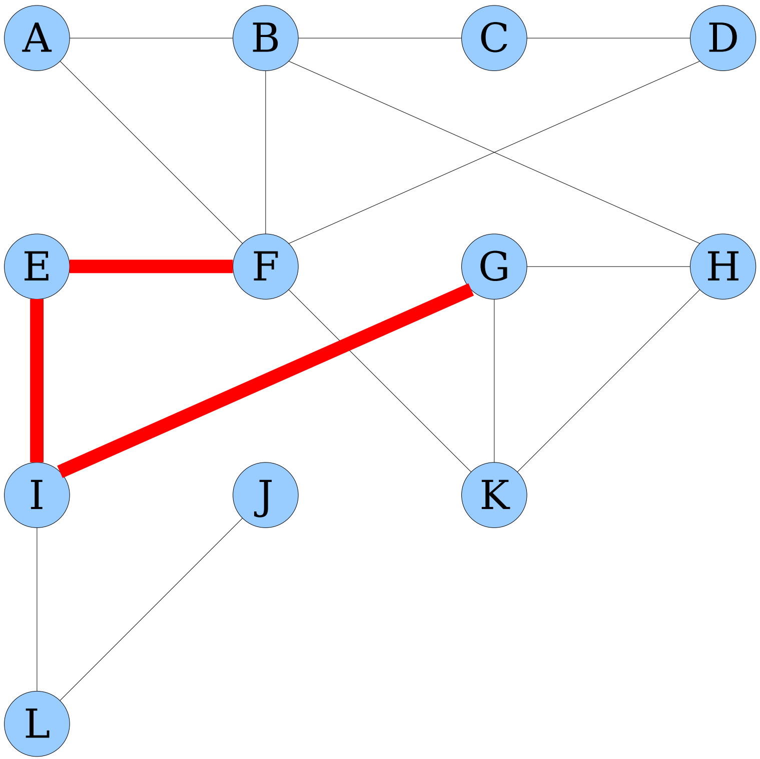
GRAPH ALGORITHMS

# 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

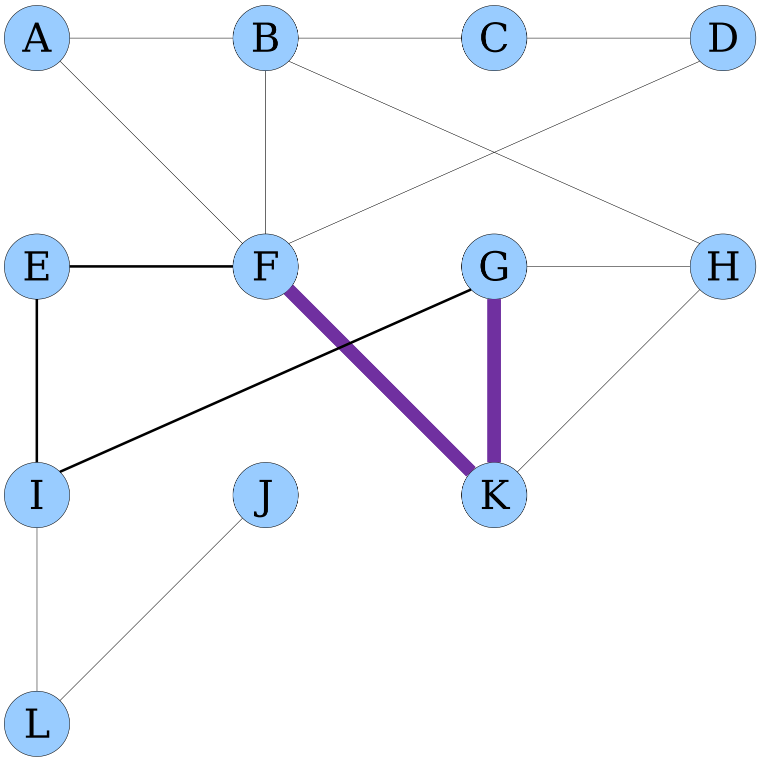



**BFS is useful for finding the <u>shortest path</u> between two nodes (in an unweighted, or equally-weighted graph).**

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

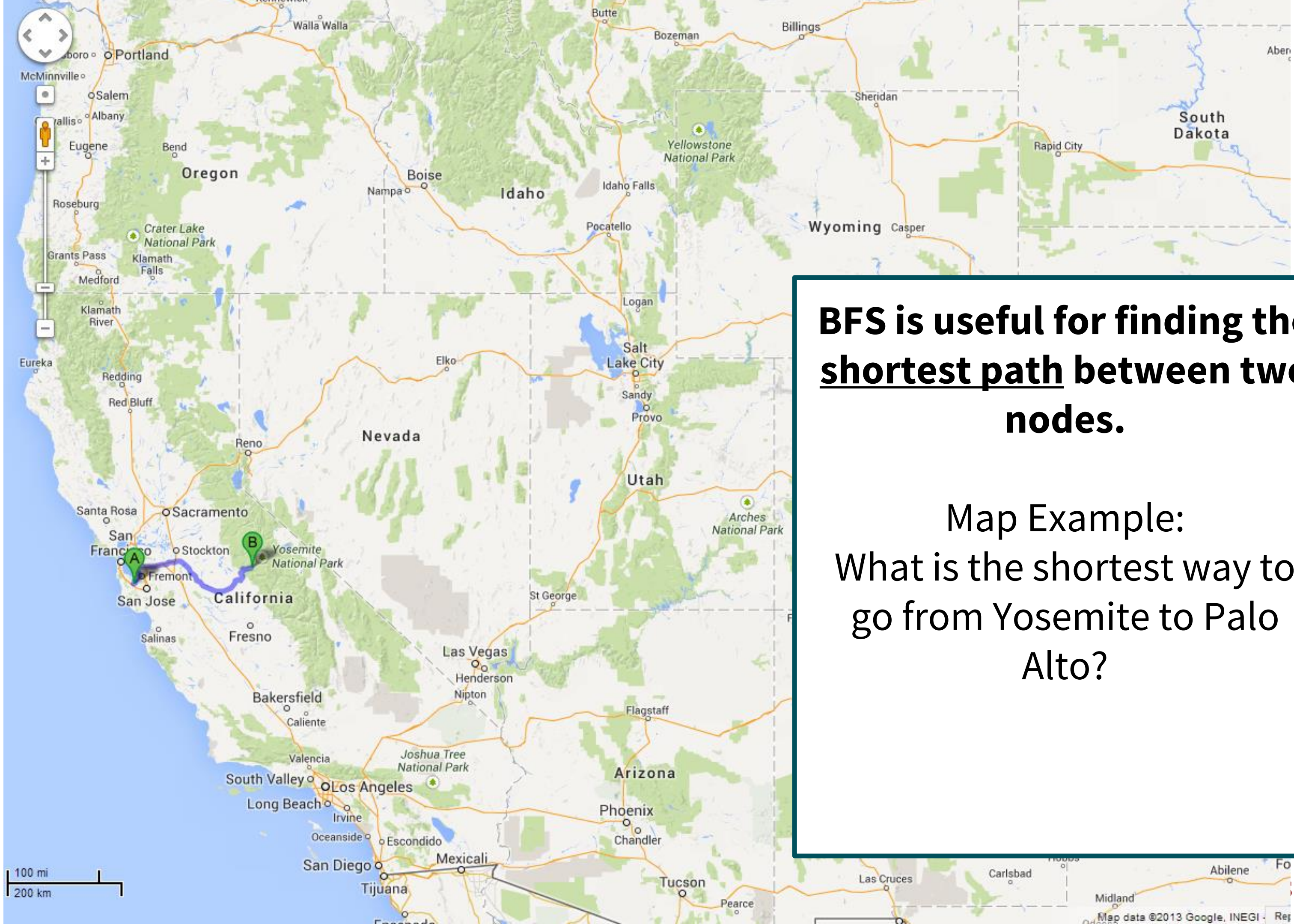



**BFS is useful for finding the <u>shortest path</u> between two nodes (in an unweighted, or equally-weighted graph).**

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

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

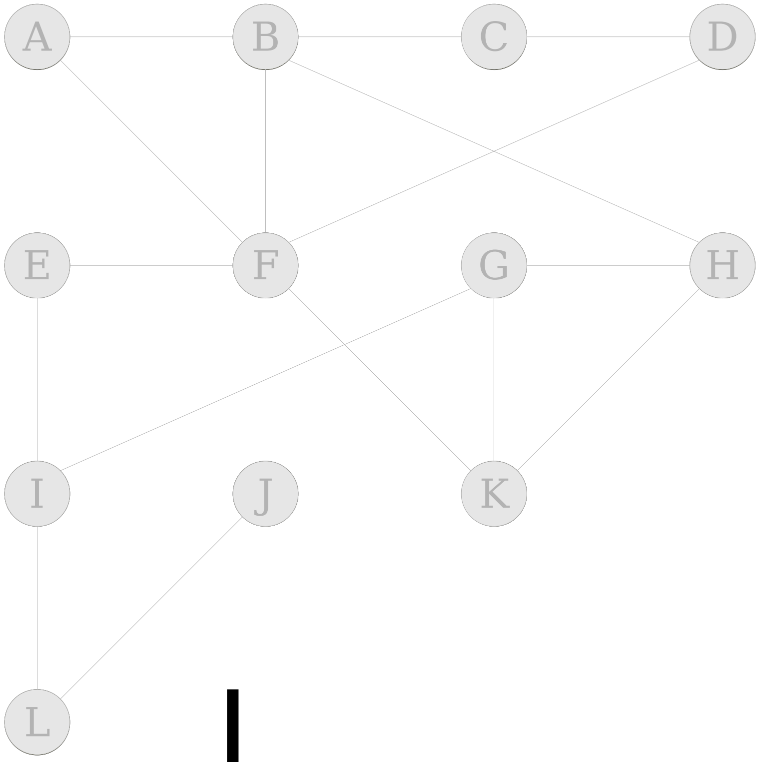
Shortest way:
F → K → G **2 edges**




**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…

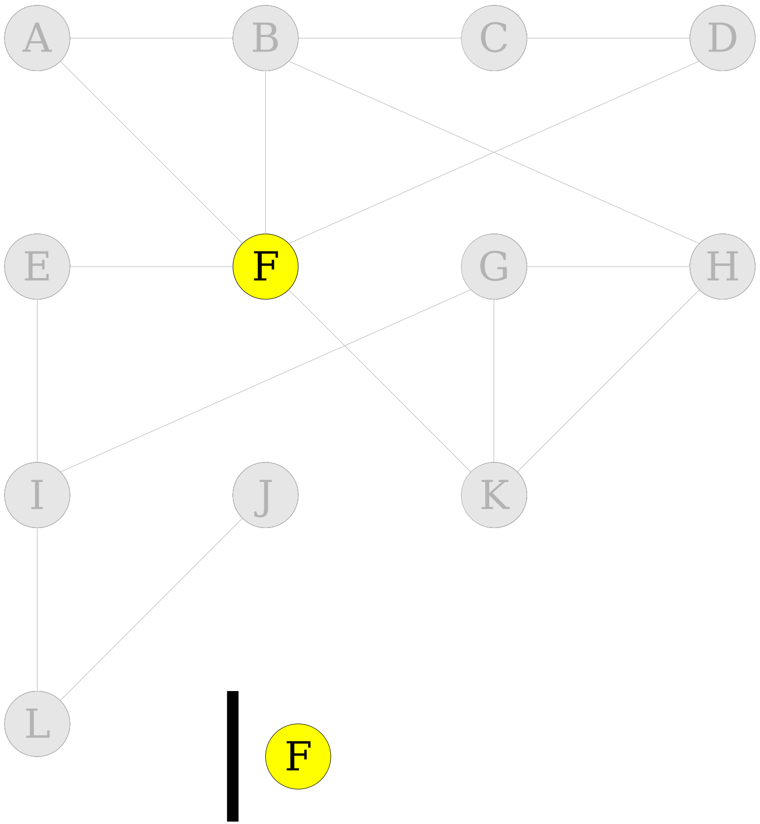



**TO START:**

(1) Color all nodes GREY to mean UNVISITED

(2) Queue is empty

# A BFS algorithm for graphs with a special property…



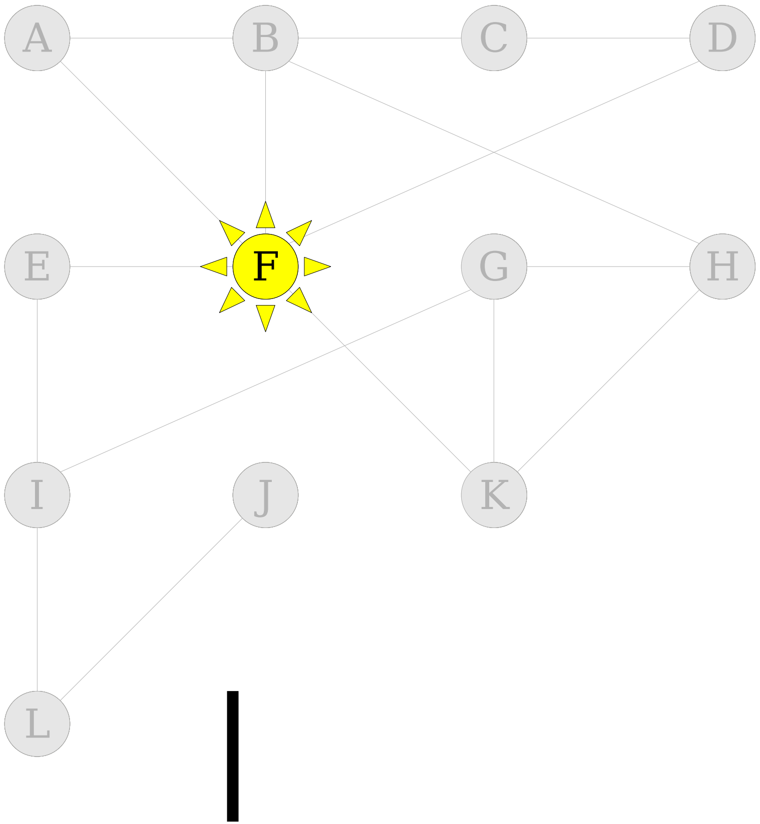

**TO START:**

(1) Color all nodes GREY to mean UNVISITED

(2) Queue is empty

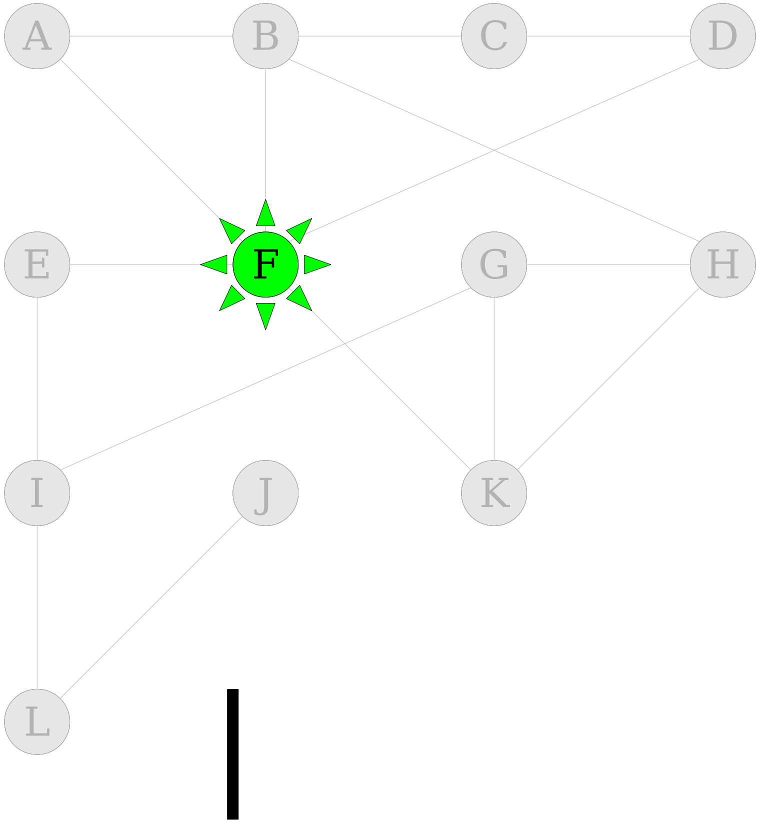
(3) Enqueue the desired **start** node, change its color to mark it VISITED

# A BFS algorithm for graphs with a special property…

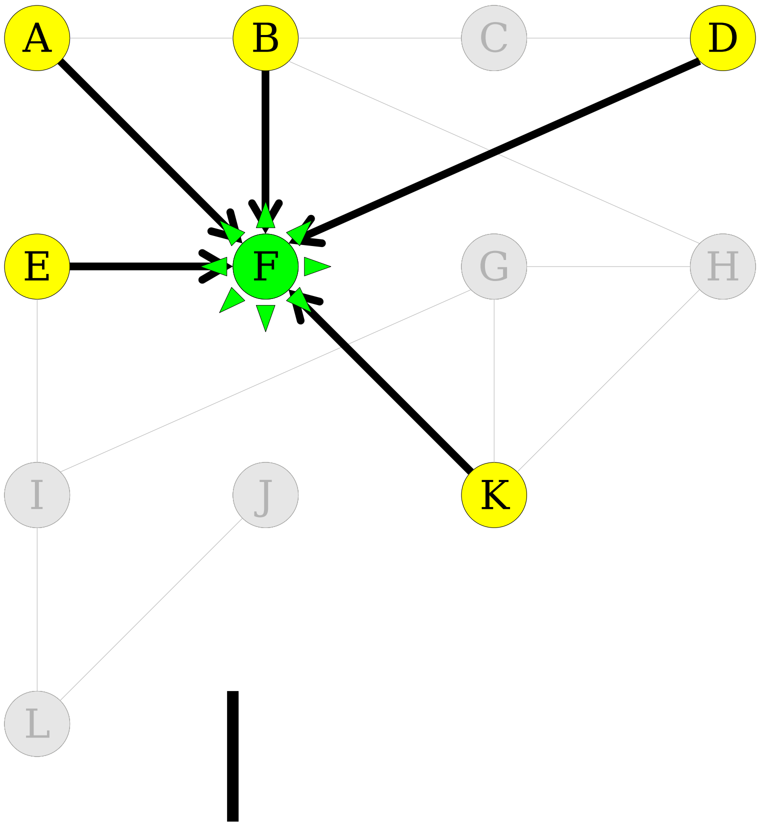



**LOOP PROCEDURE:**
(1) Dequeue a node
(2) Set current node's UNVISITED neighbors' parent pointers to current node, then enqueue them (and mark them visited when we enqueue)

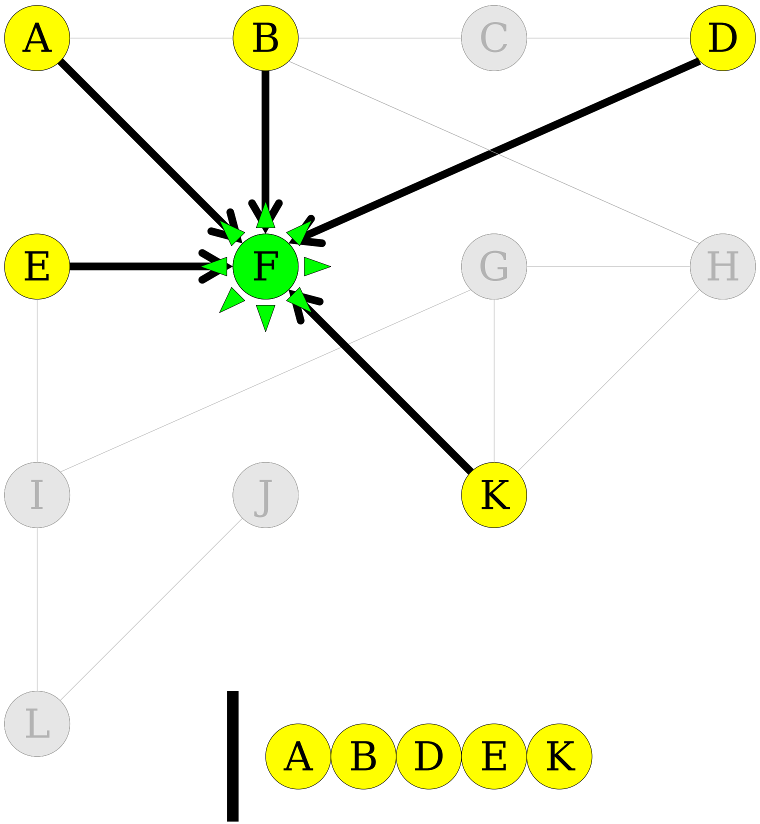
# Breadth-First Search



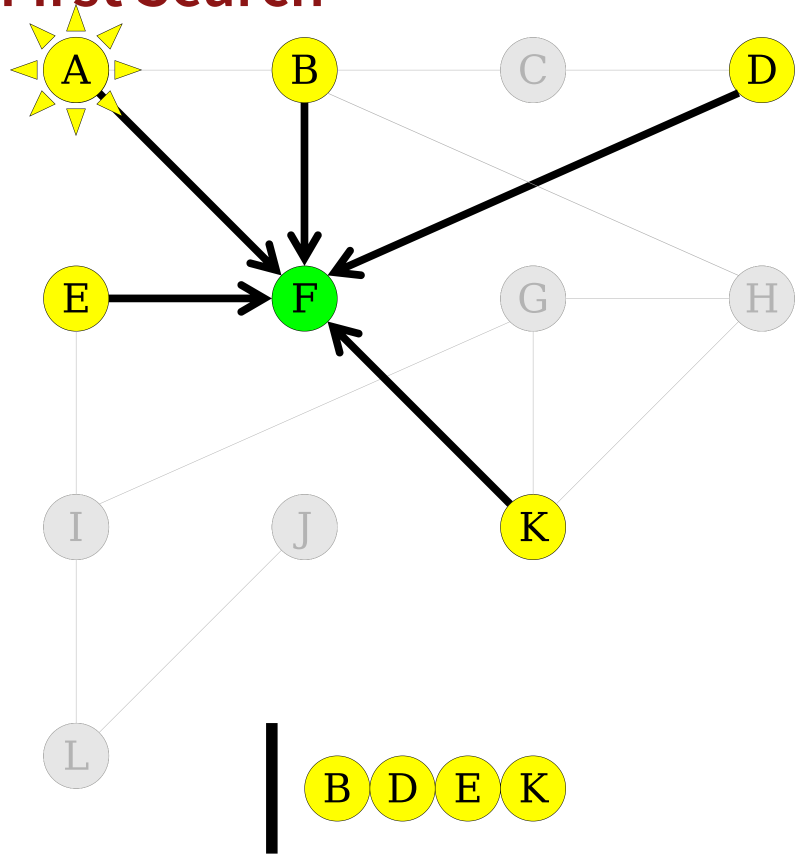
# Breadth-First Search



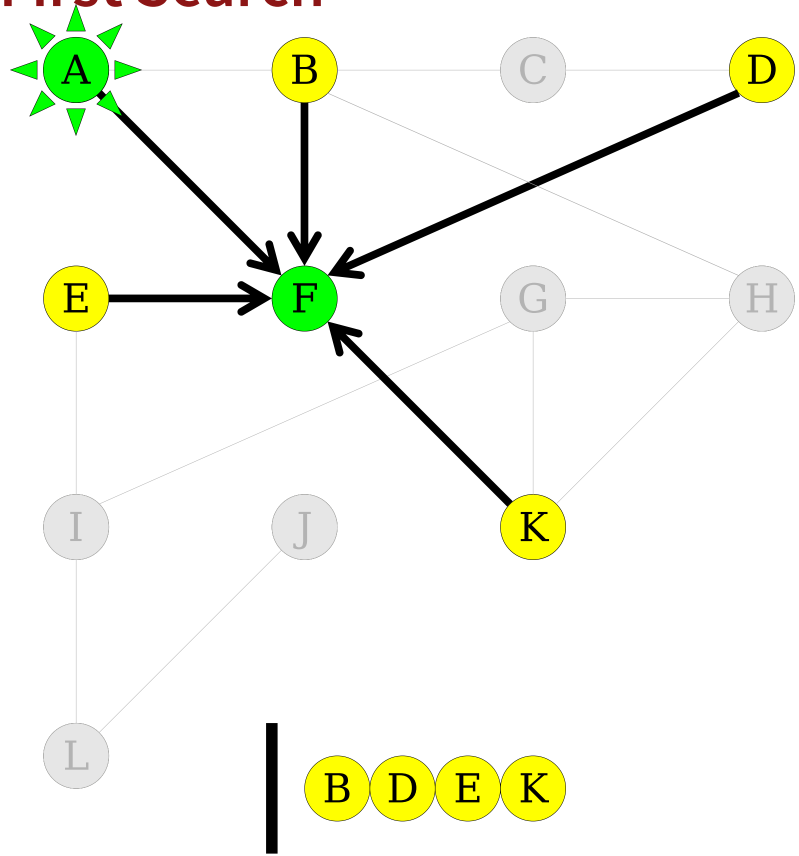
# Breadth-First Search



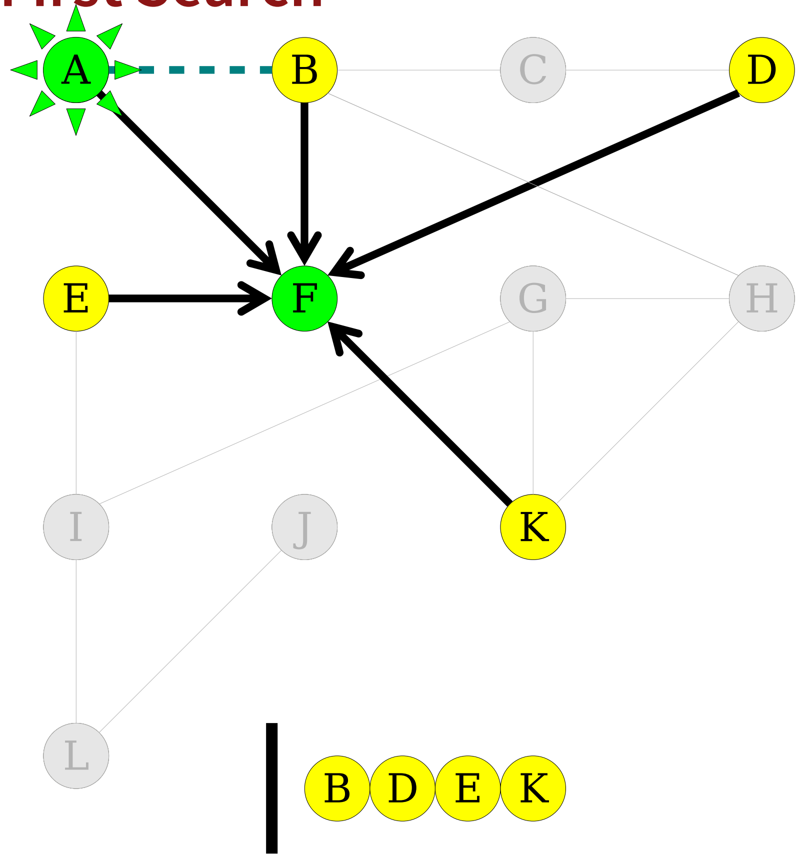
# Breadth-First Search



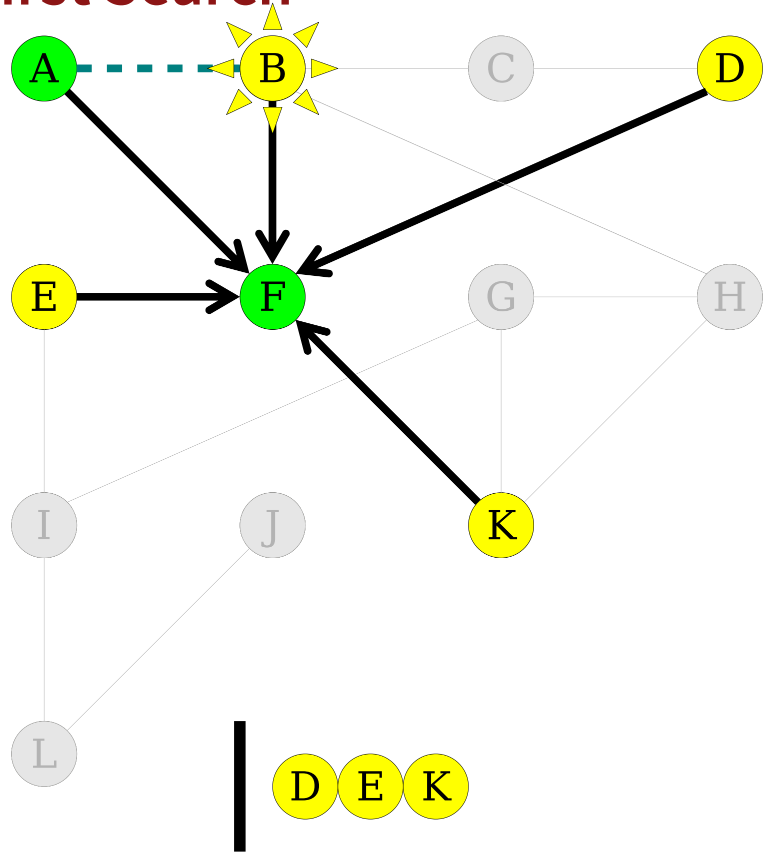
# Breadth-First Search



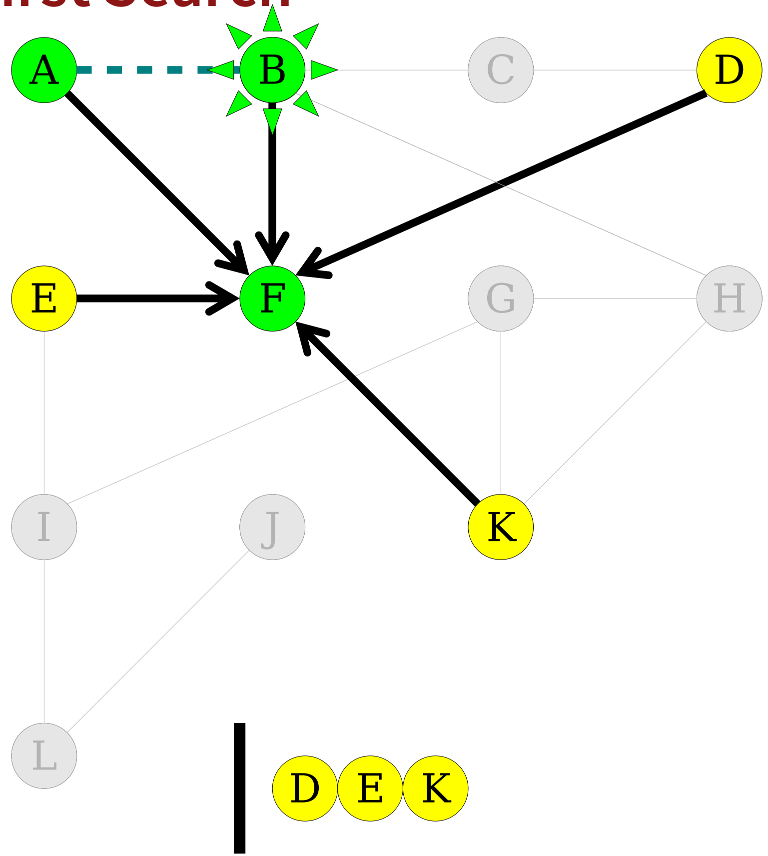
# Breadth-First Search



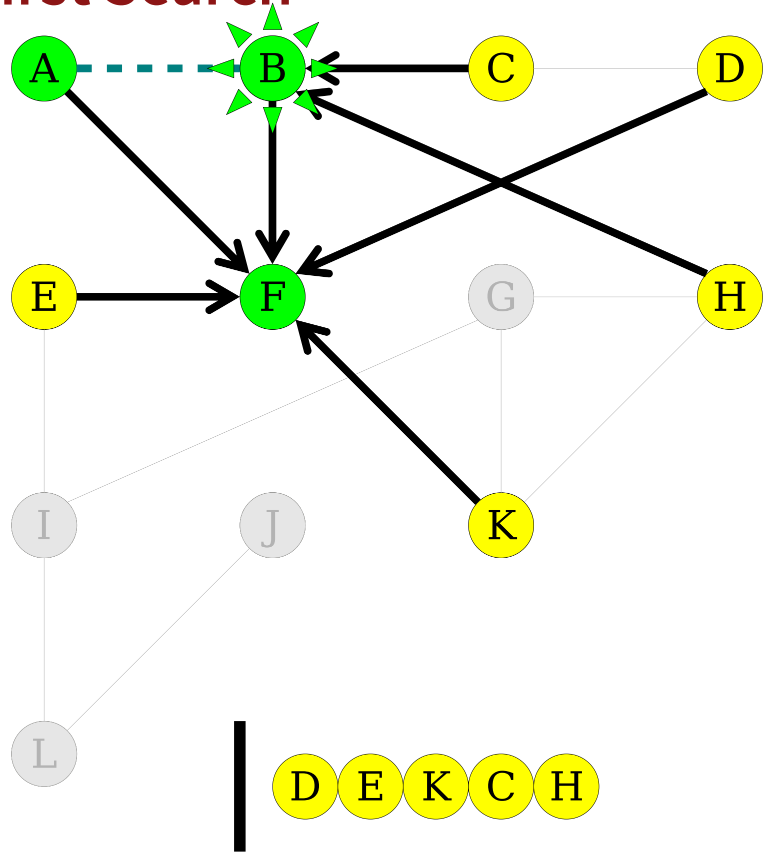
# Breadth-First Search



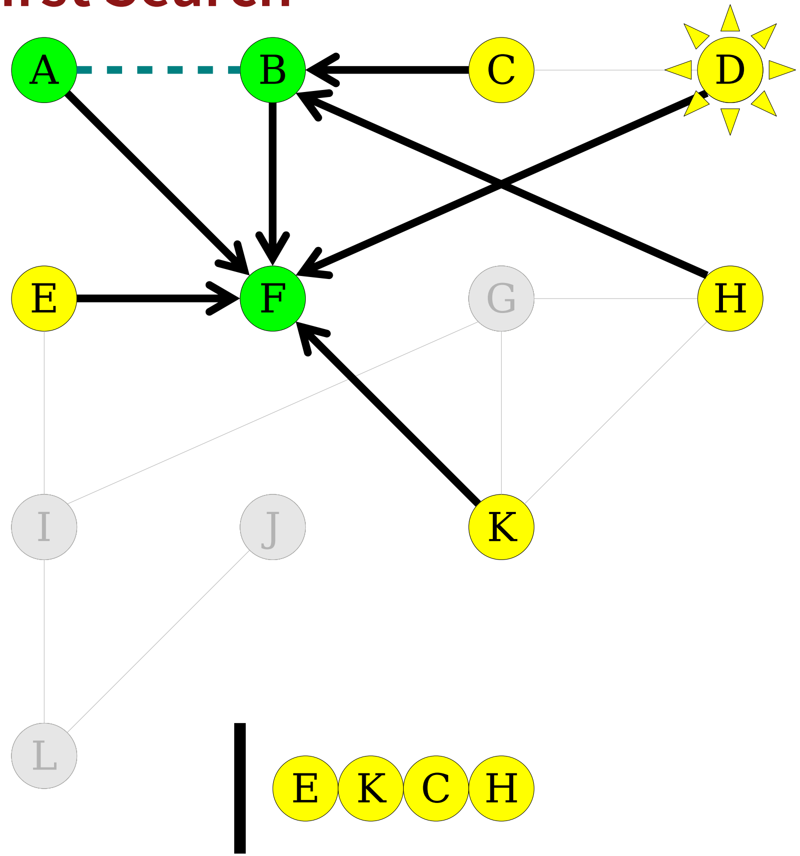
# Breadth-First Search



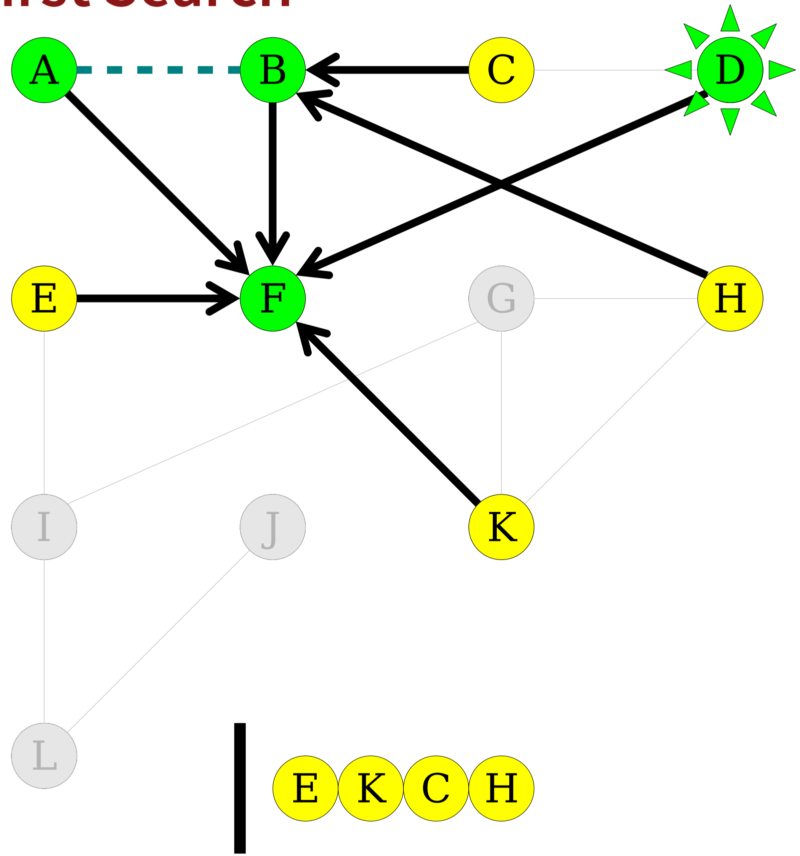
# Breadth-First Search



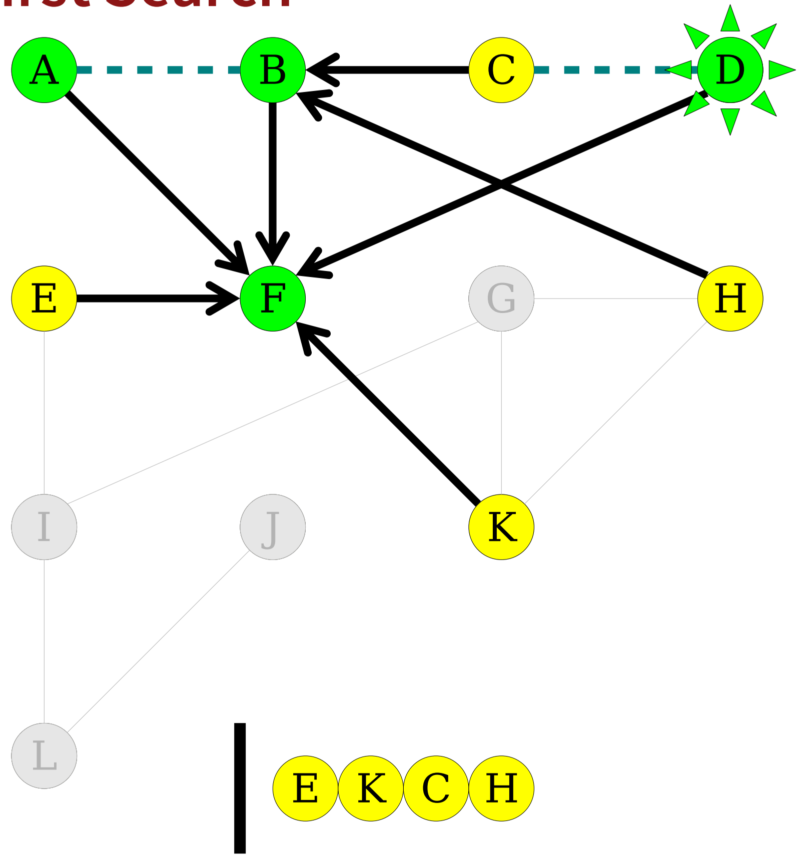
# Breadth-First Search



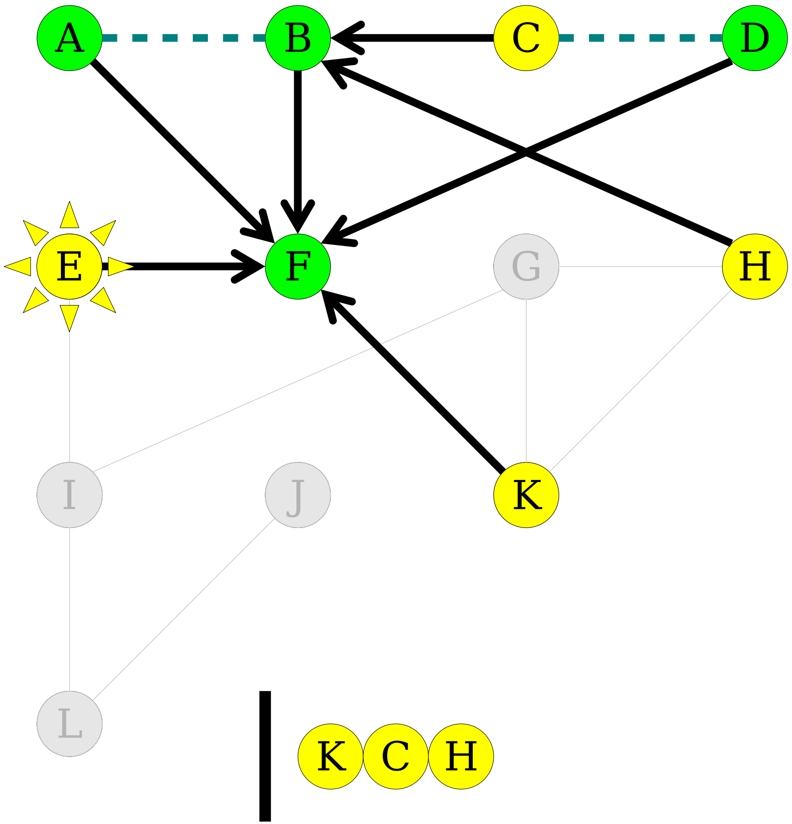
# Breadth-First Search



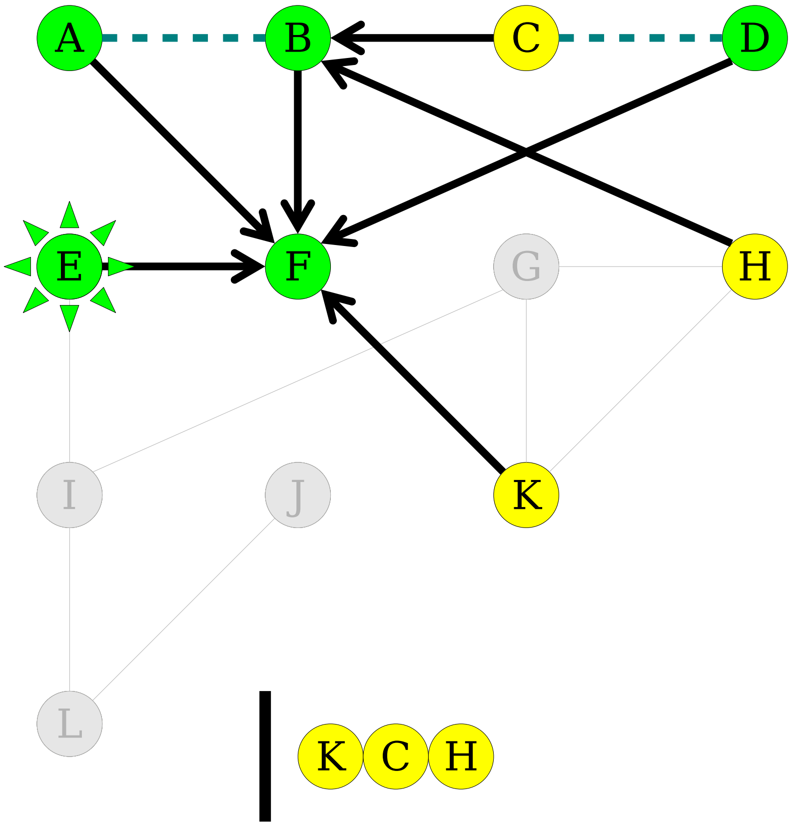
# Breadth-First Search



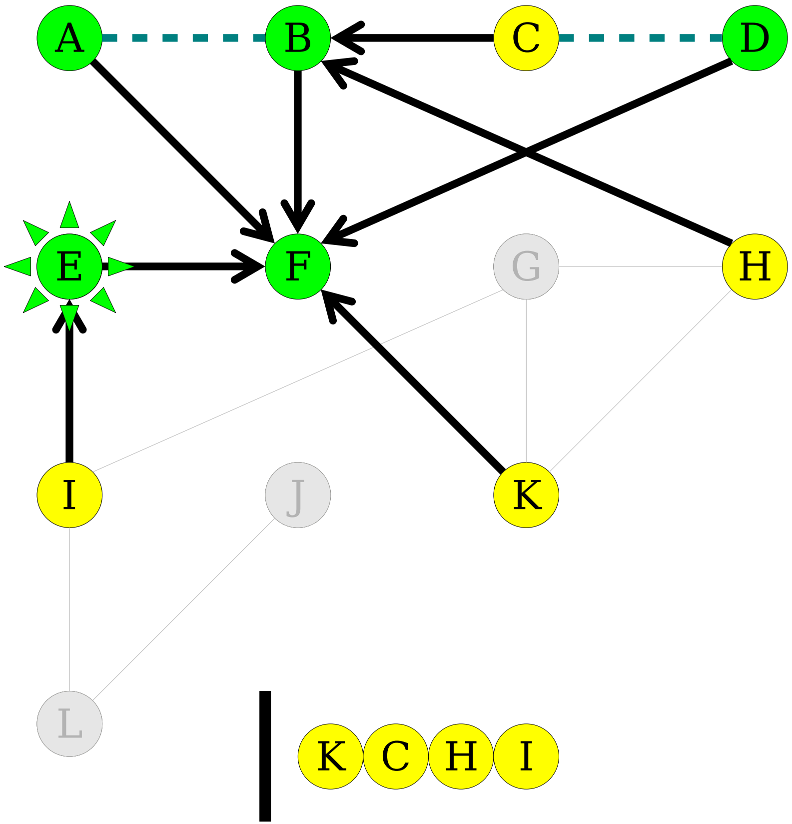
# Breadth-First Search



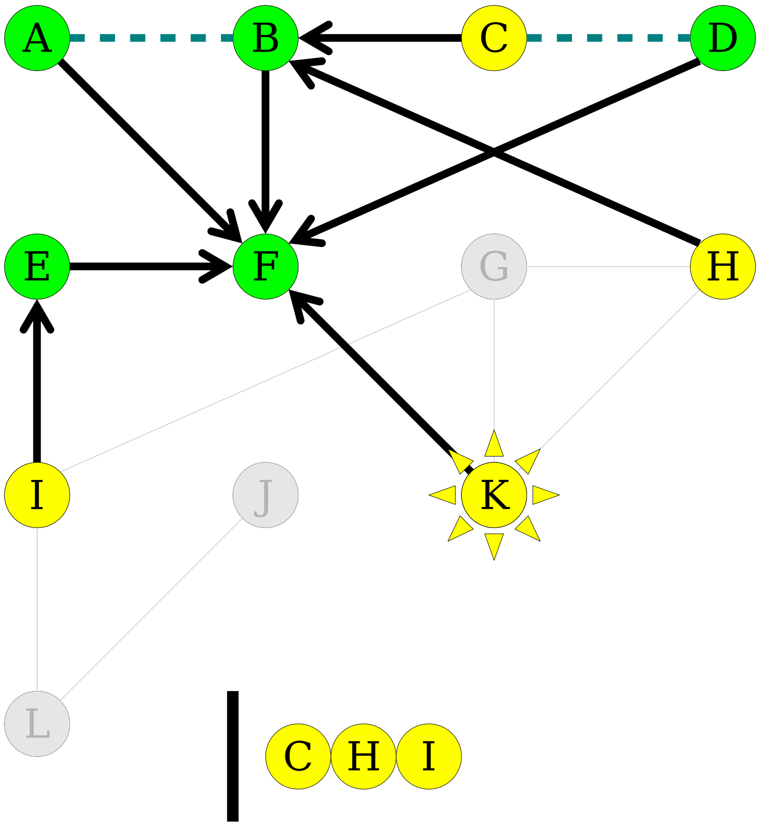
# Breadth-First Search



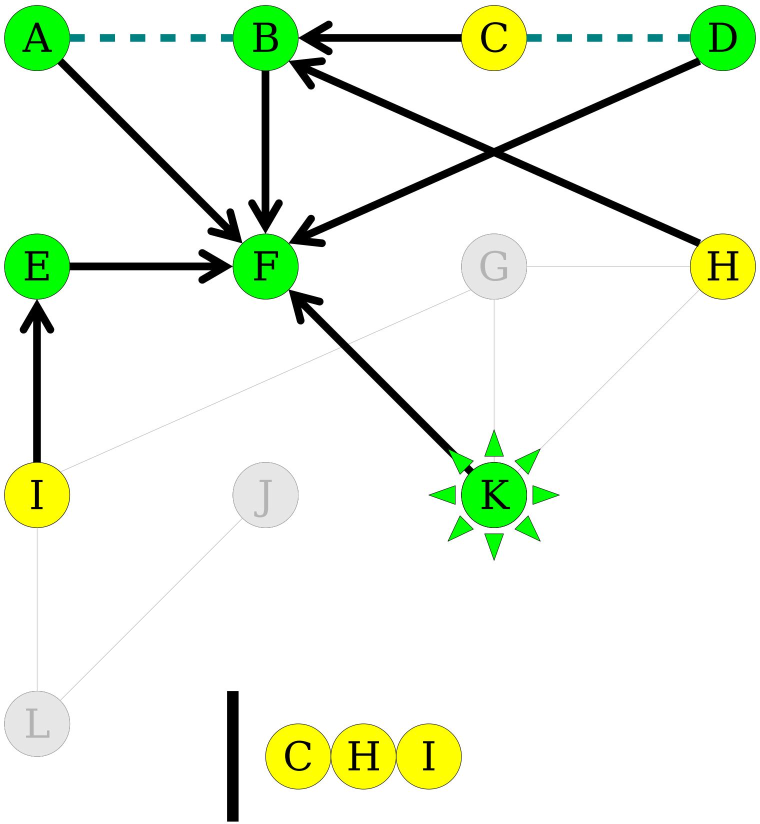
# Breadth-First Search



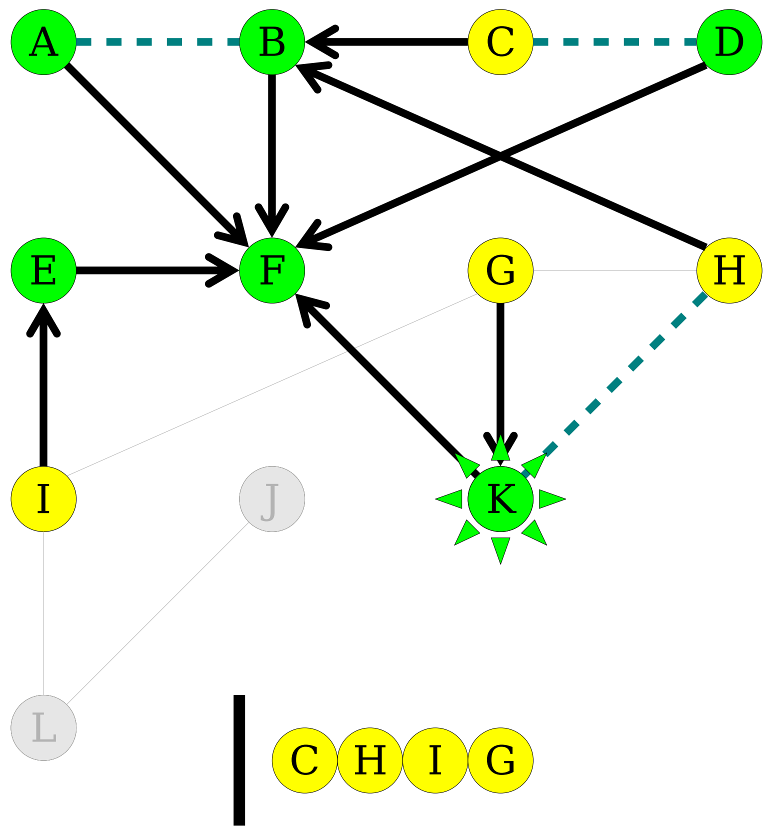
# Breadth-First Search



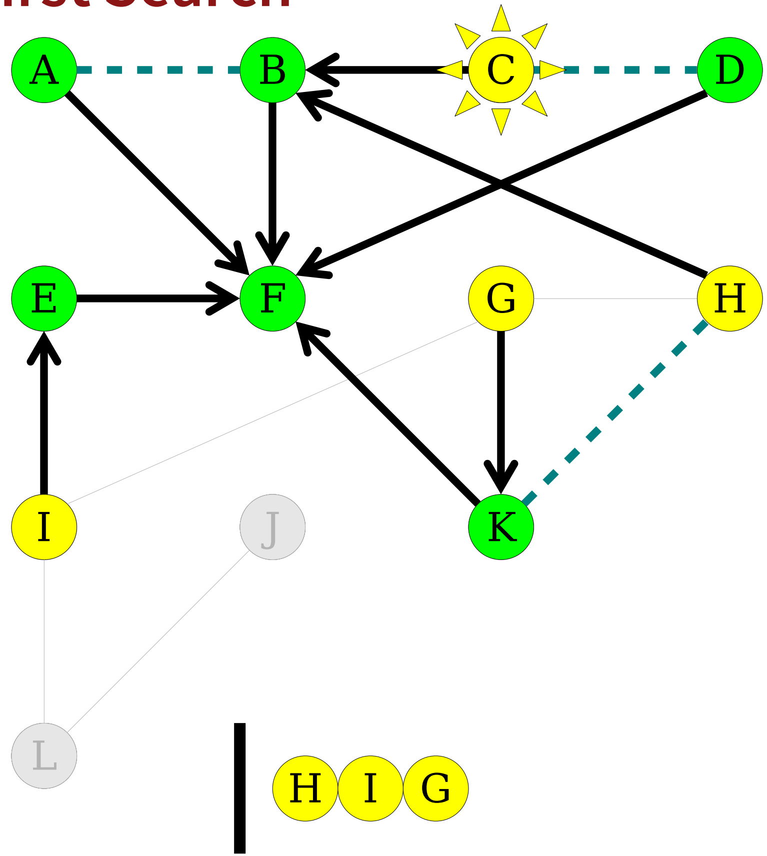
# Breadth-First Search



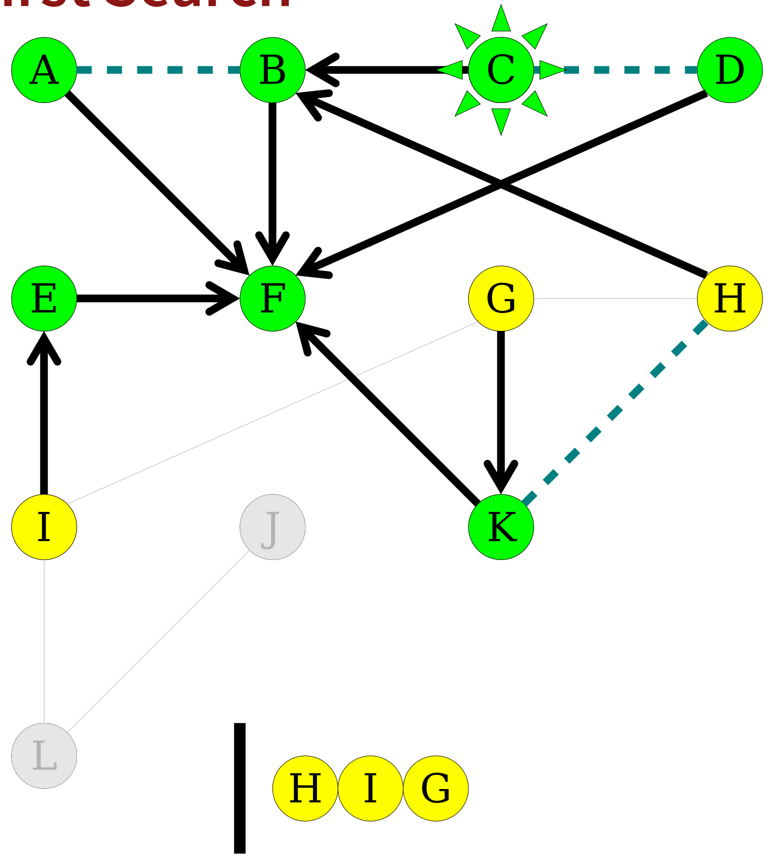
# Breadth-First Search



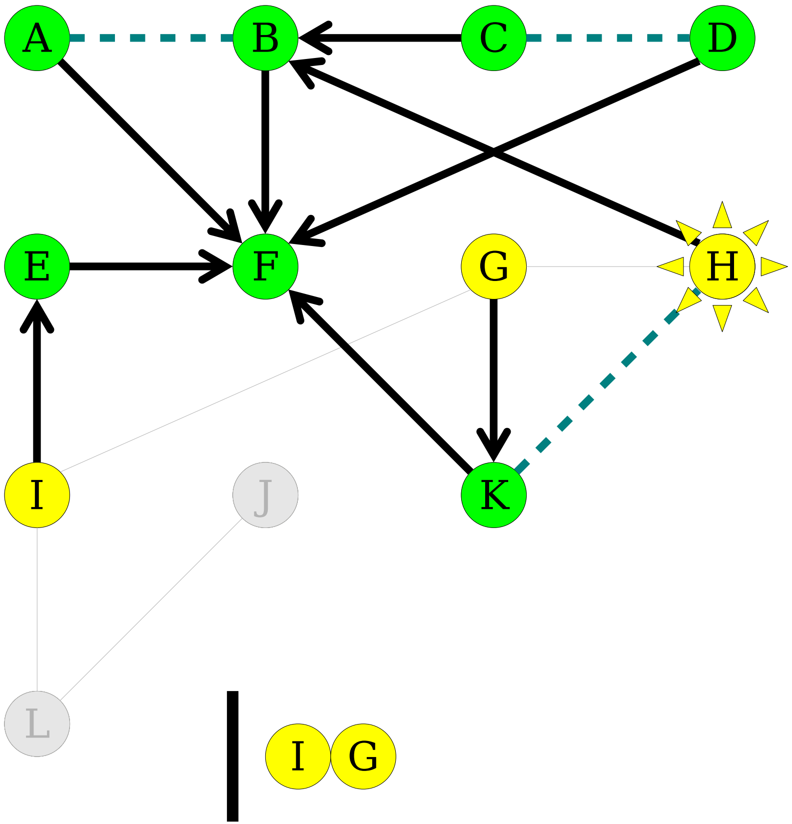
# Breadth-First Search



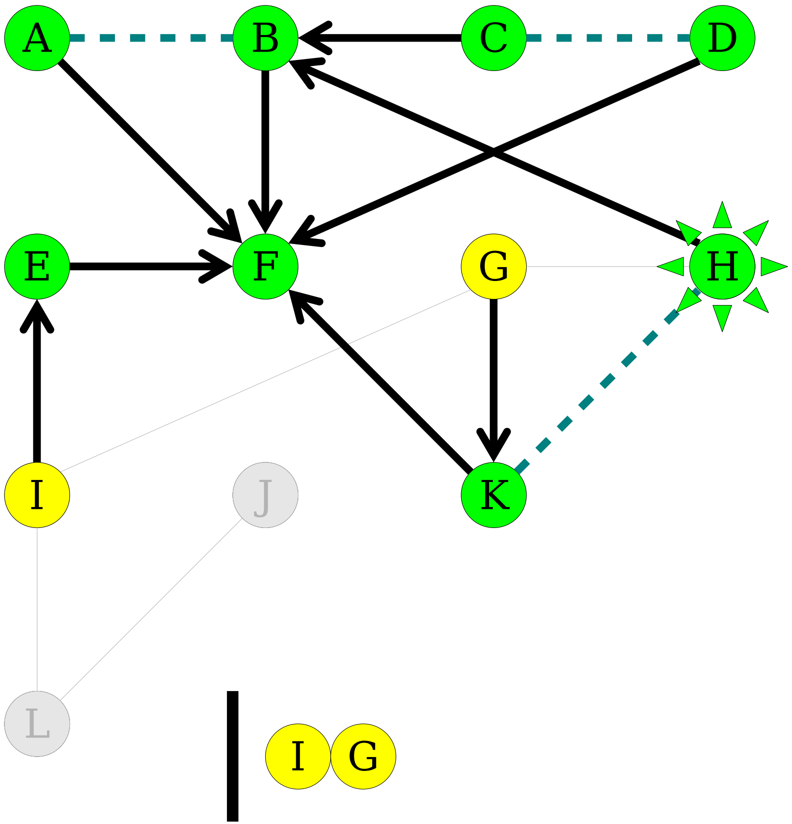
# Breadth-First Search



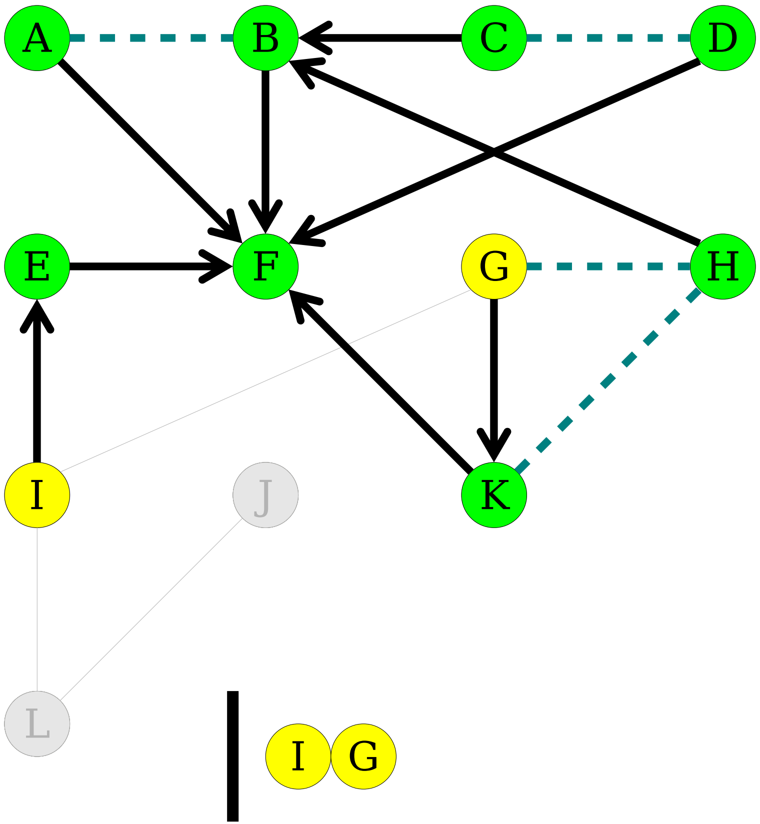
# Breadth-First Search



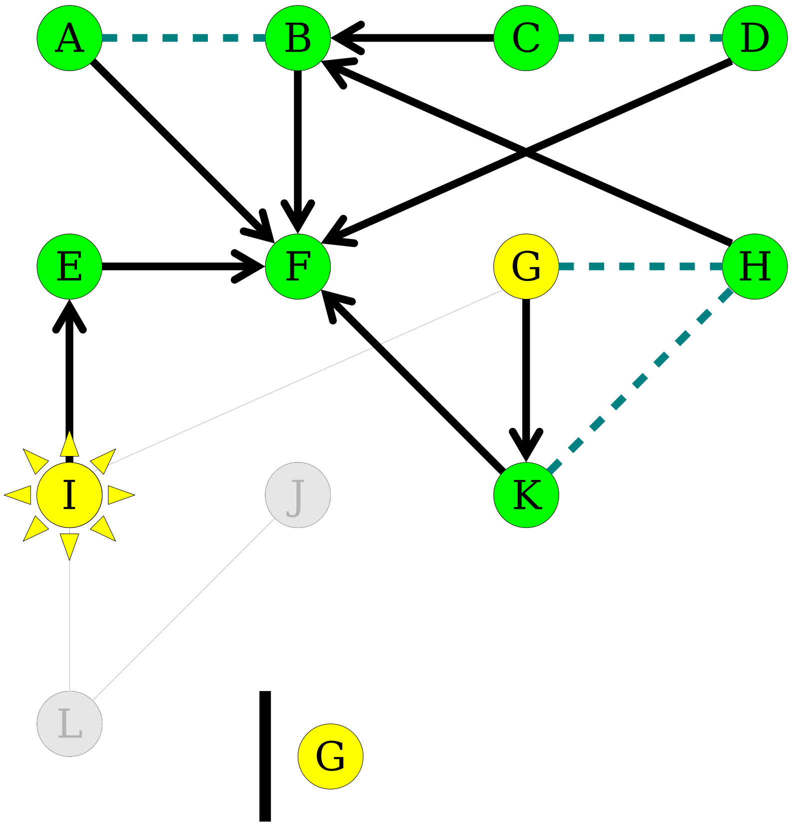
# Breadth-First Search



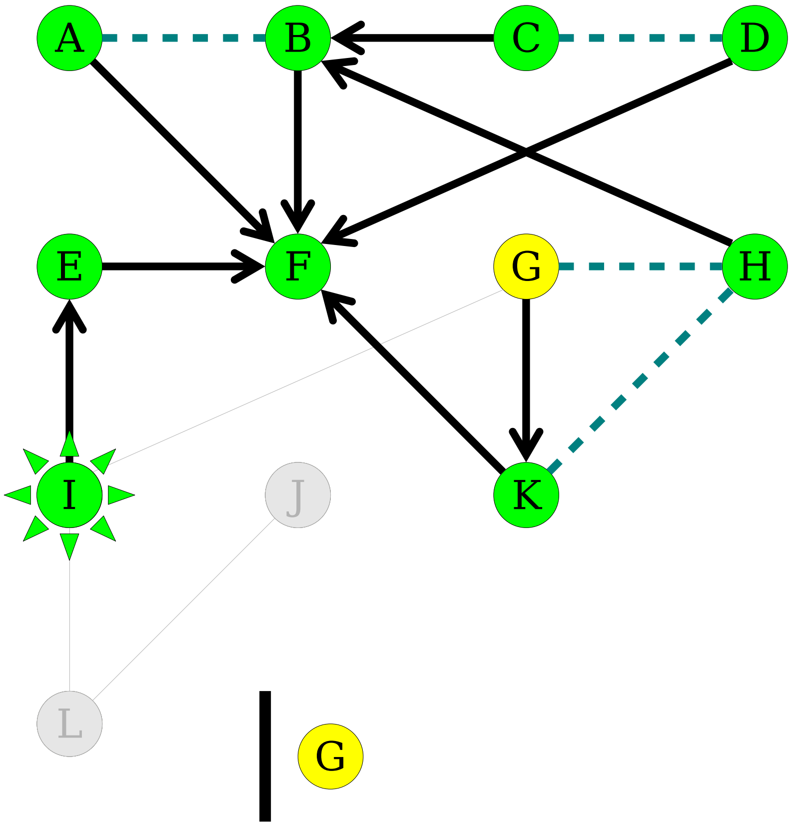
# Breadth-First Search



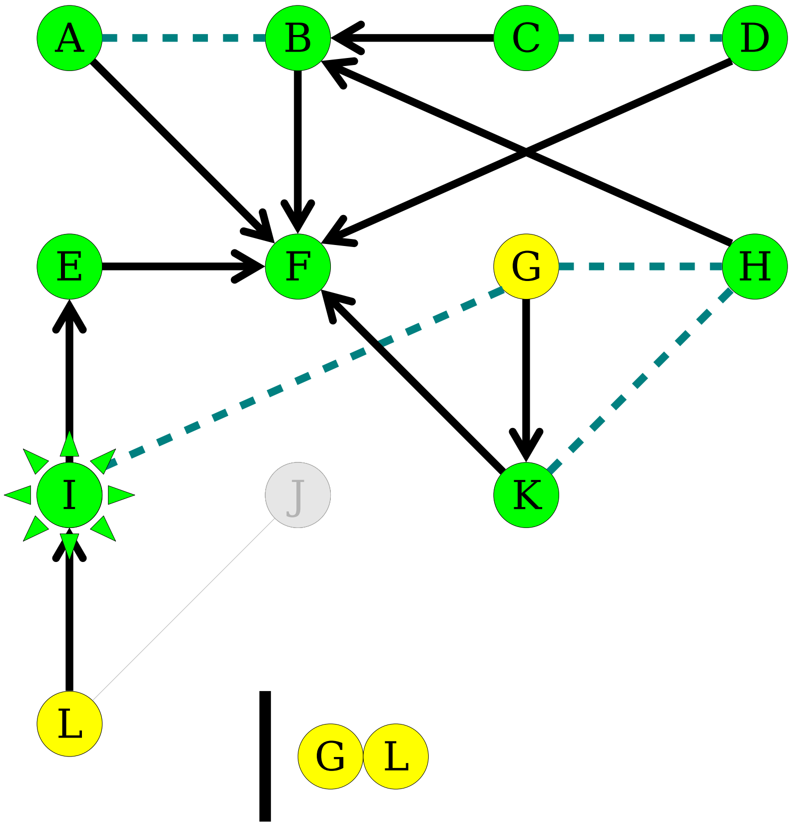
# Breadth-First Search



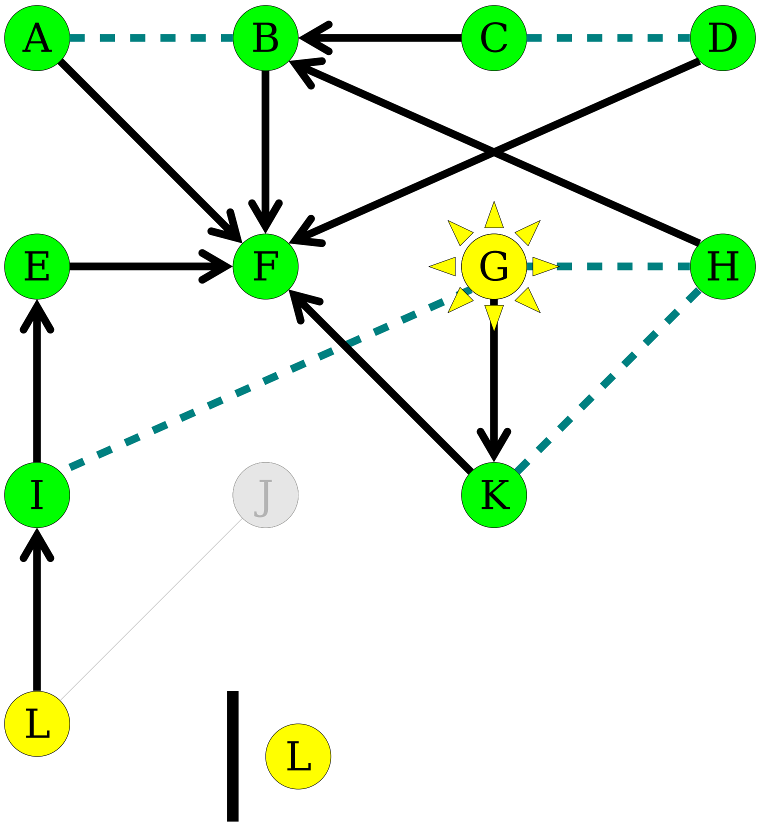
# Breadth-First Search



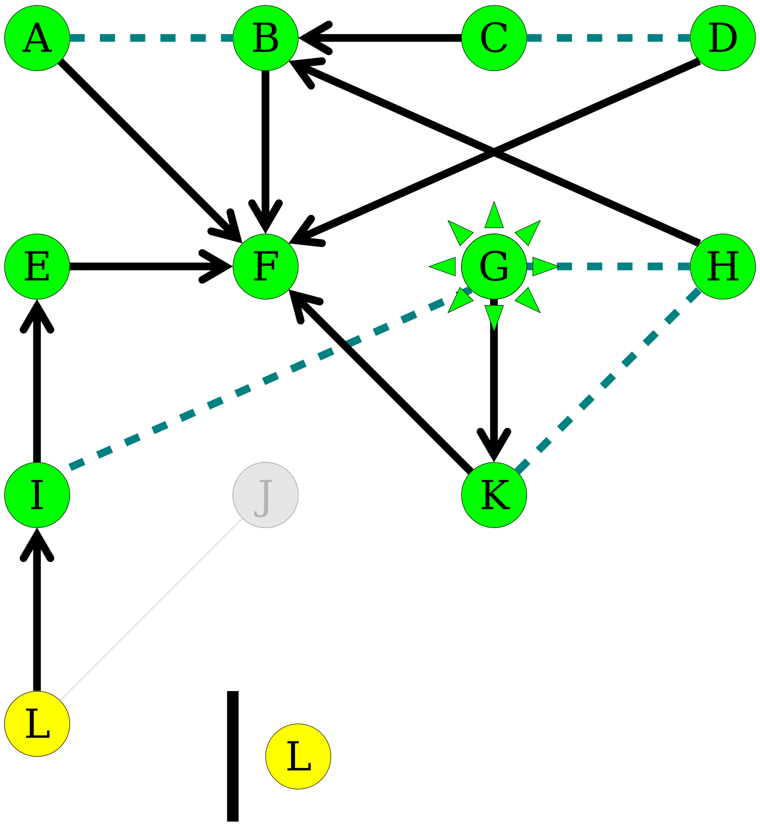
# Breadth-First Search



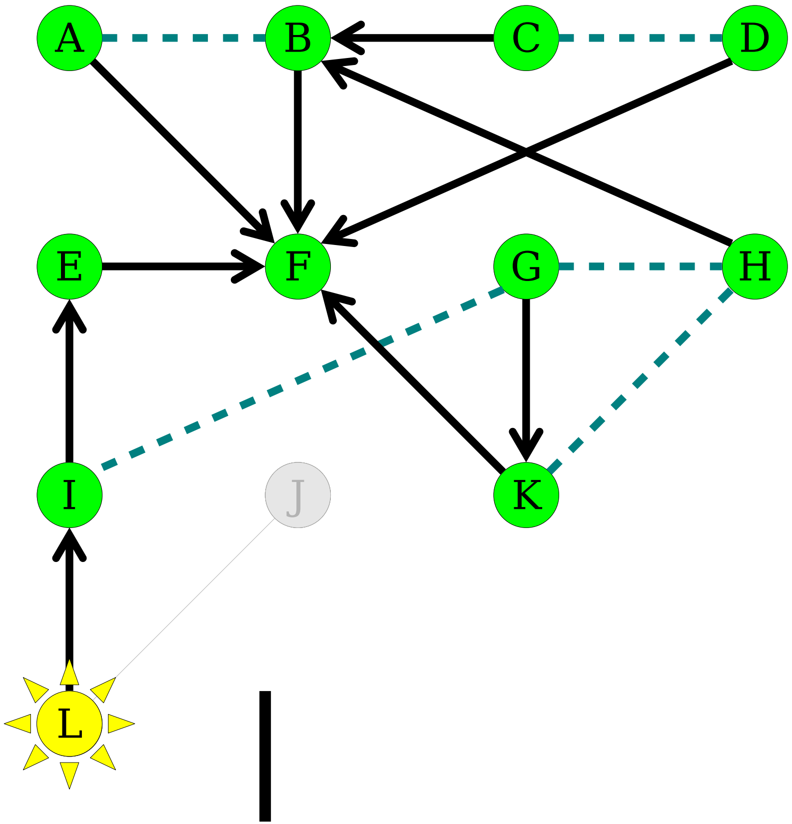
# Breadth-First Search



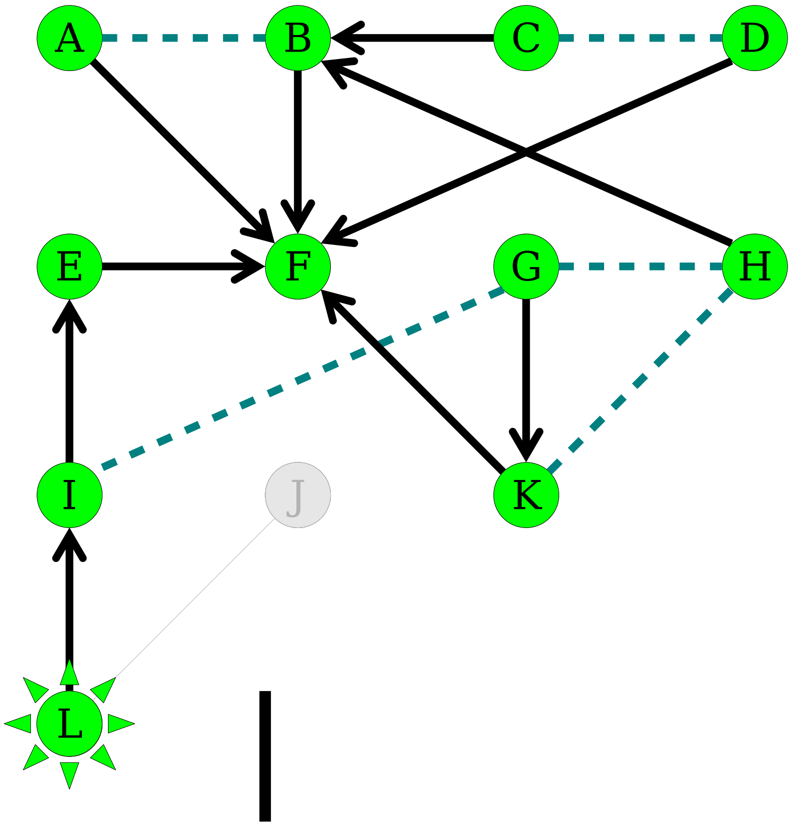
# Breadth-First Search



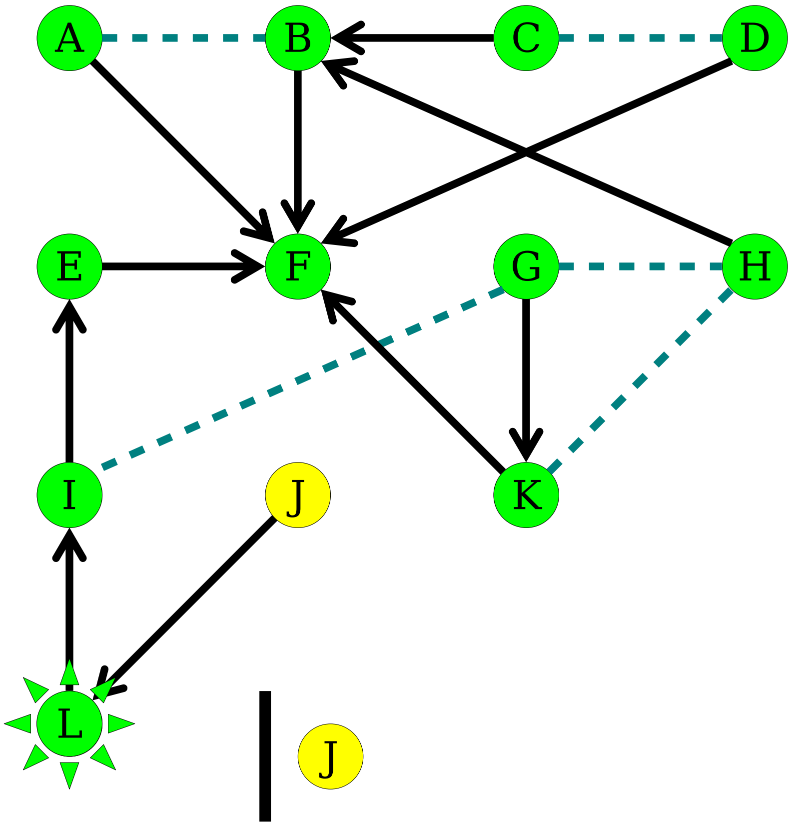
# Breadth-First Search



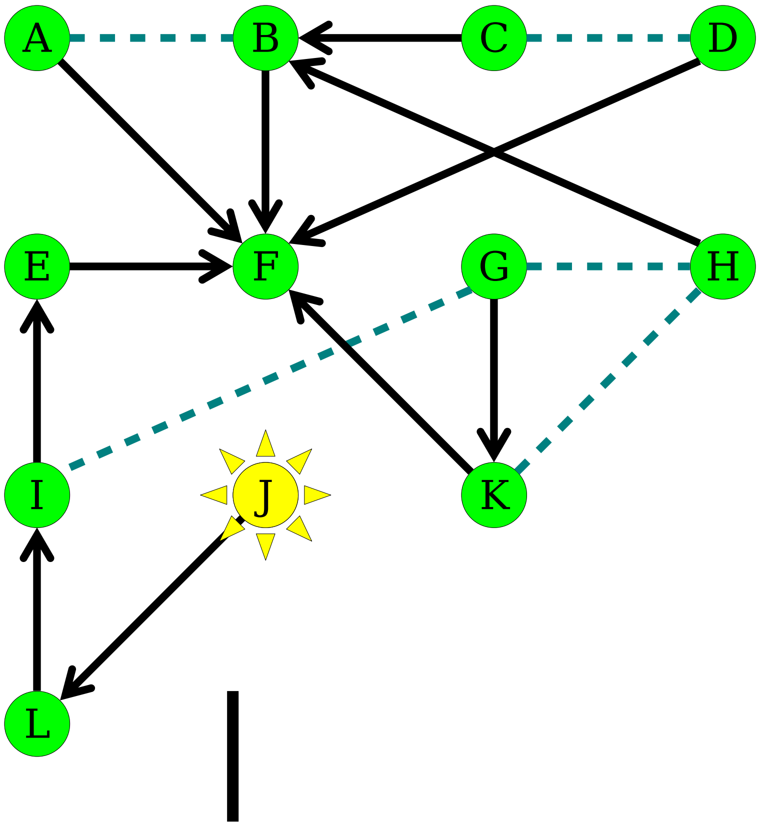
# Breadth-First Search



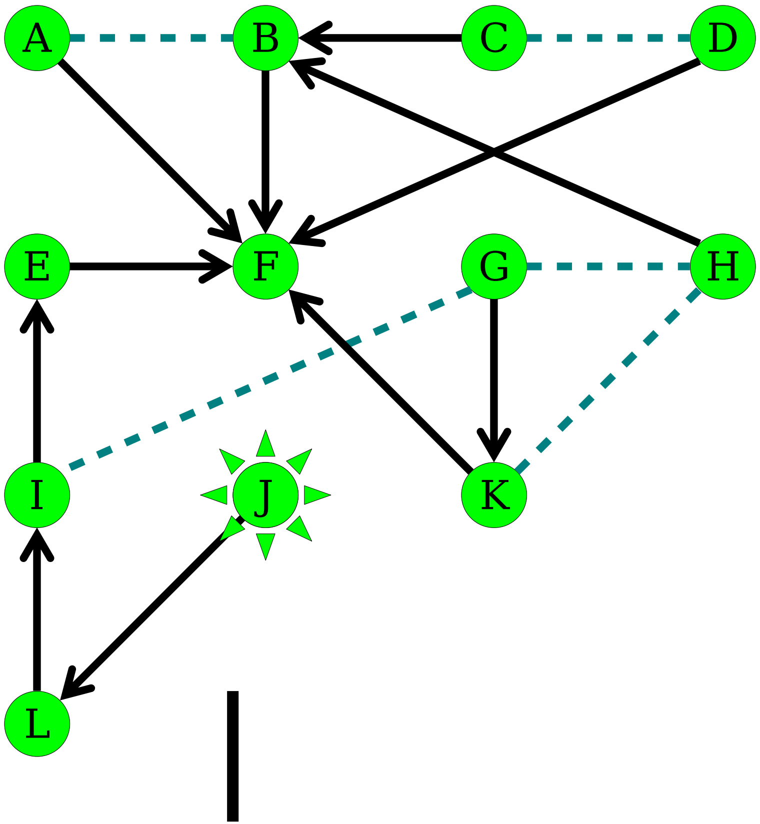
# Breadth-First Search



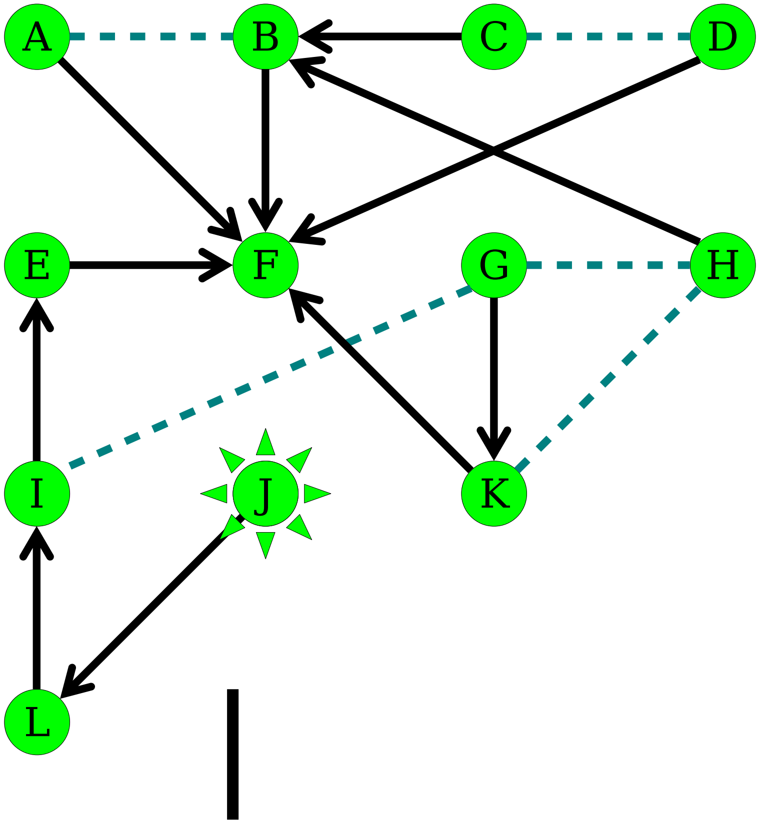
# Breadth-First Search



# Breadth-First Search



# Breadth-First Search



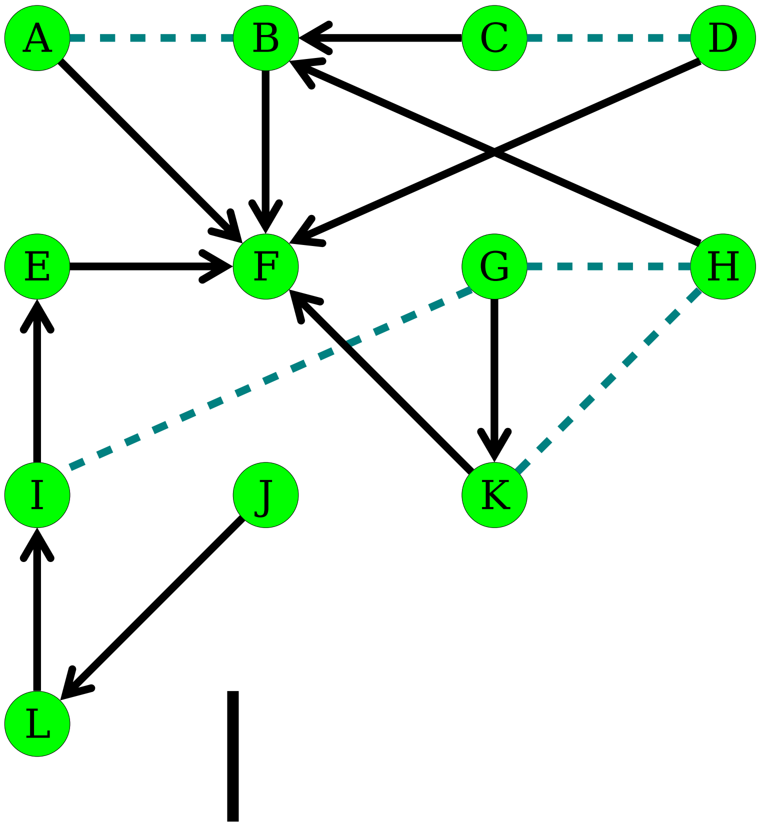

**Done!**

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

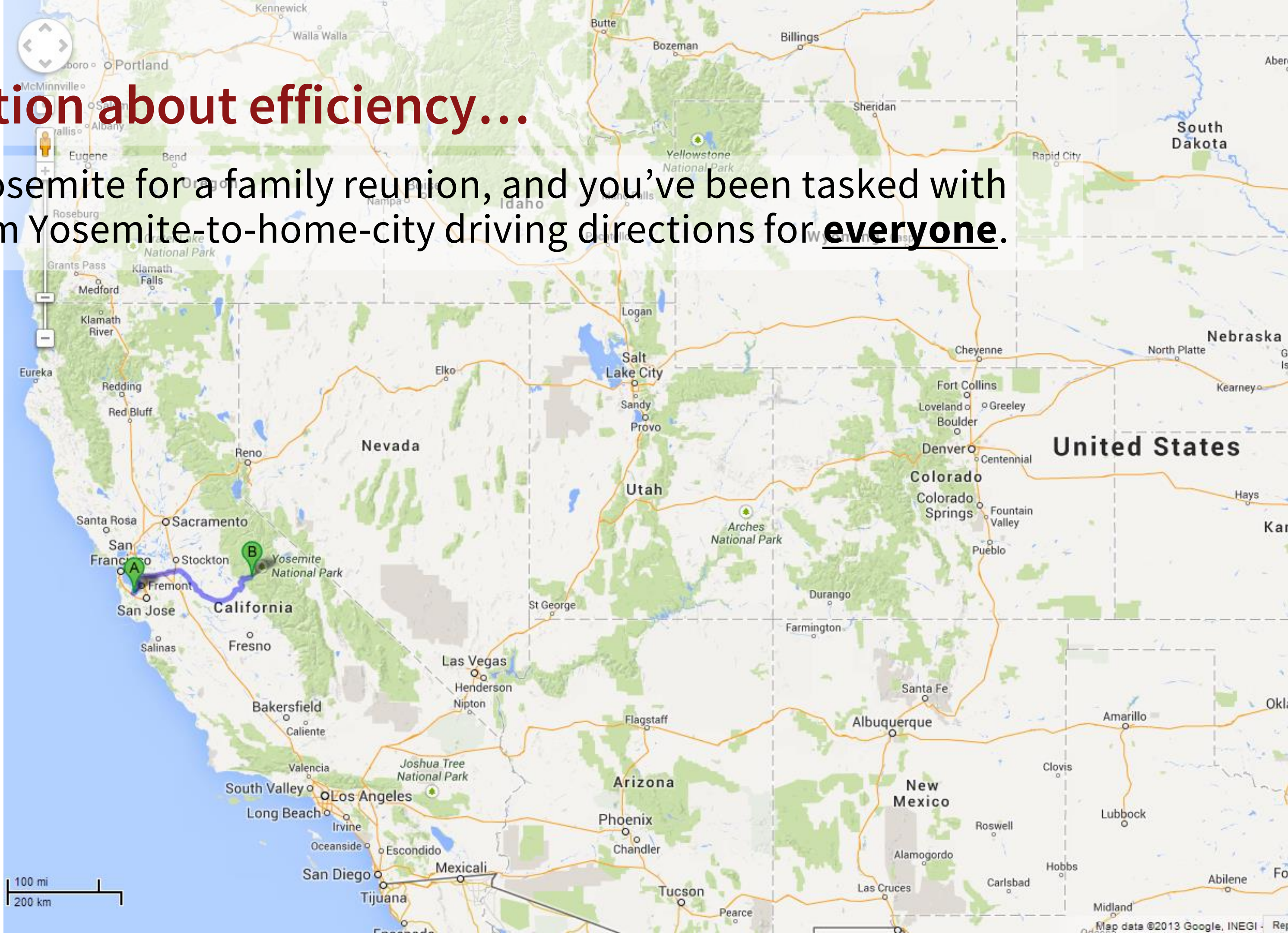
(3) If you follow the parent pointers from the desired end point, you will get back to the start point, and it will be the shortest way to do that

# 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…

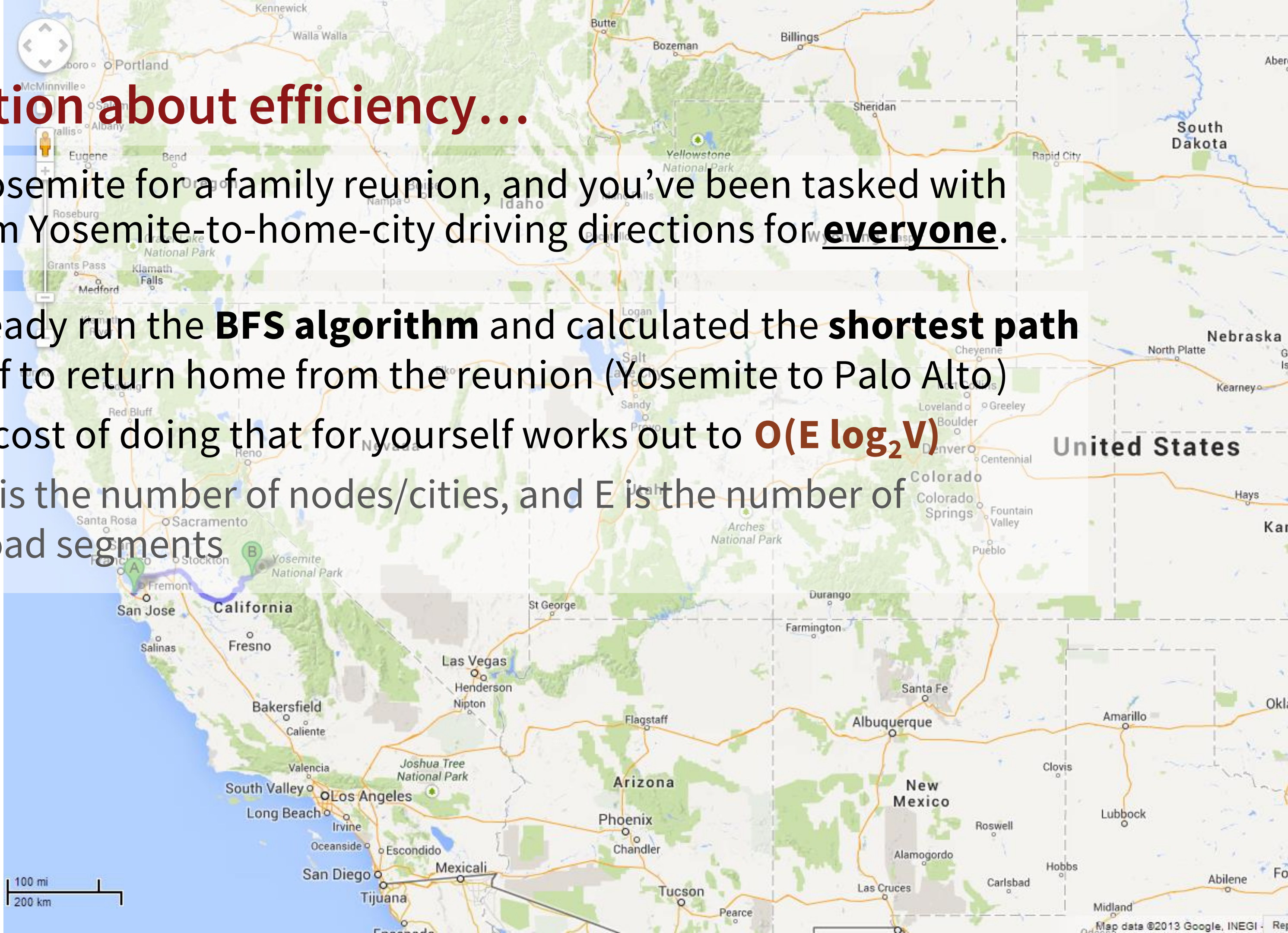
You’re all in Yosemite for a family reunion, and you’ve been tasked with making custom Yosemite-to-home-city driving directions for **everyone**.



# 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 **<u>everyone</u>**.

- 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



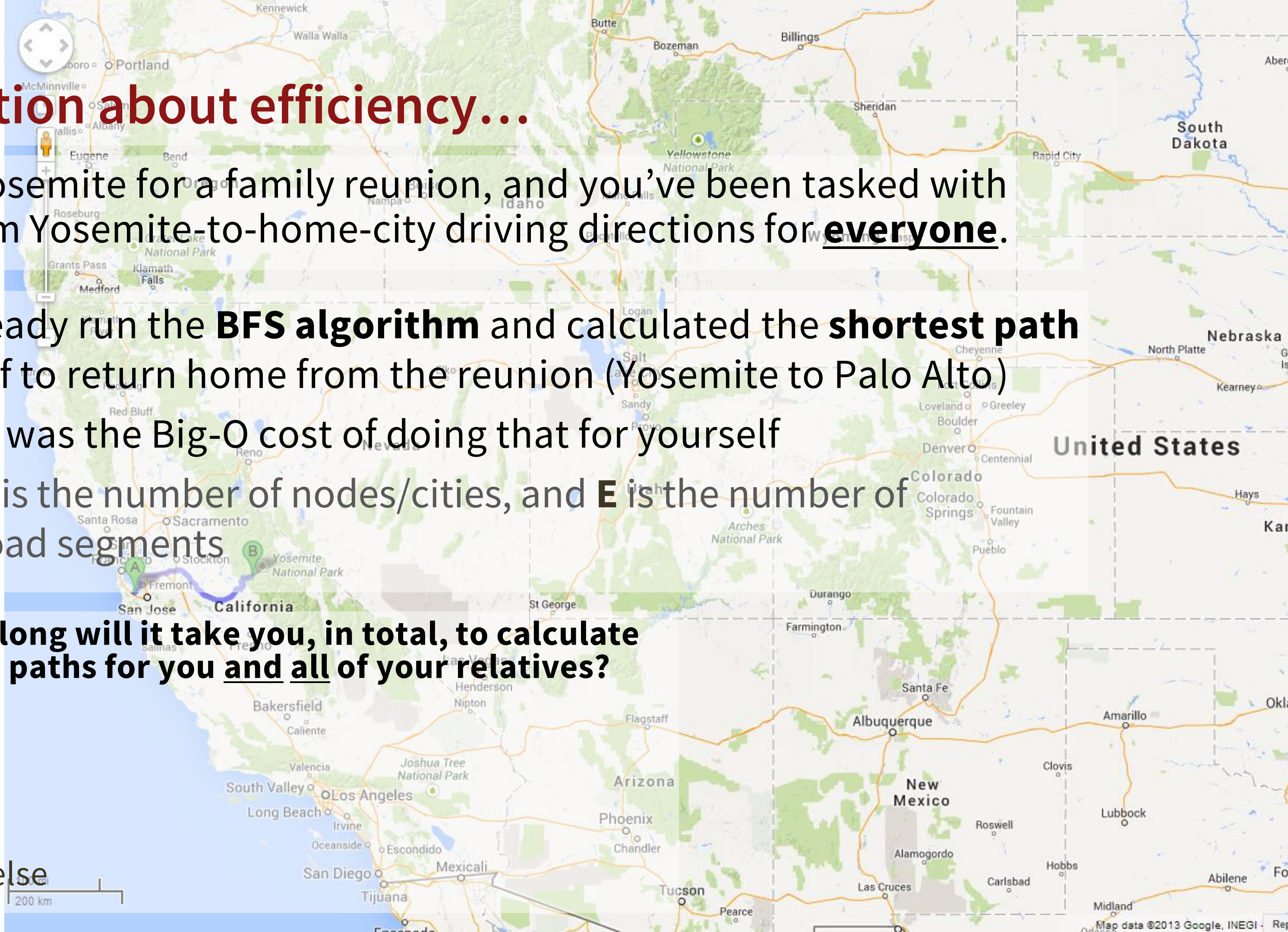
# Quick question about efficiency…

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**.

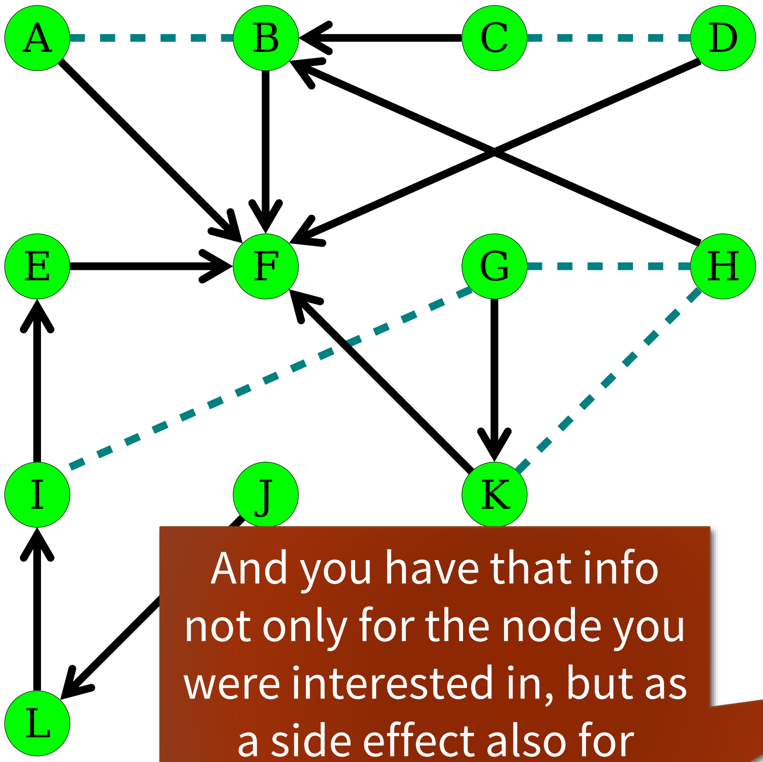
- You've already run the **BFS algorithm** and calculated the **shortest path** for yourself to return home from the reunion (Yosemite to Palo Alto)
- **$O(E \log_2 V)$** was the Big-O cost of doing that for yourself
  - Where **V** is the number of nodes/cities, and **E** is the number of edges/road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

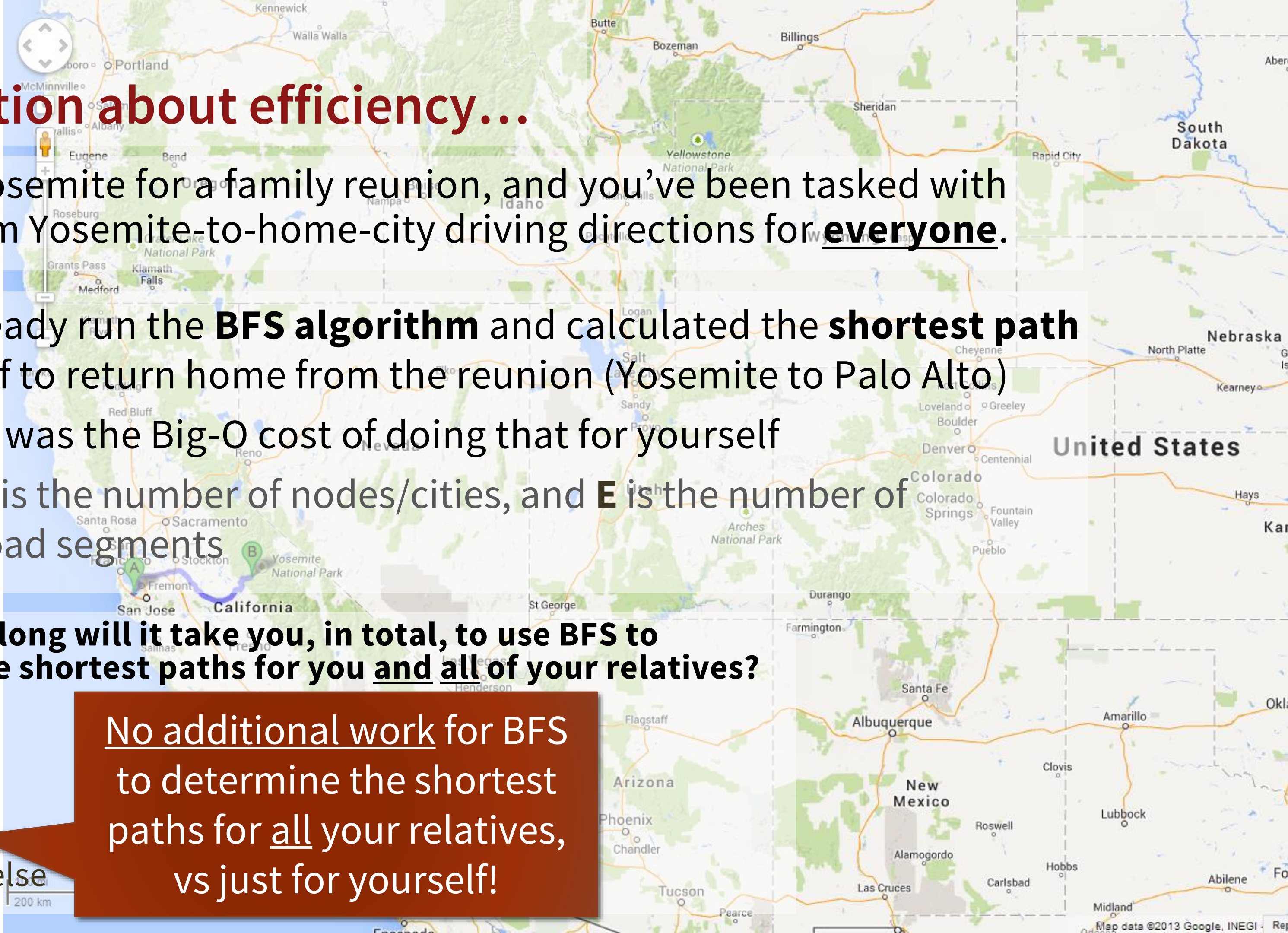
You're all in Yosemite for a family reunion, and you've been tasked with making custom Yosemite-to-home-city driving directions for **everyone**.

- 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!



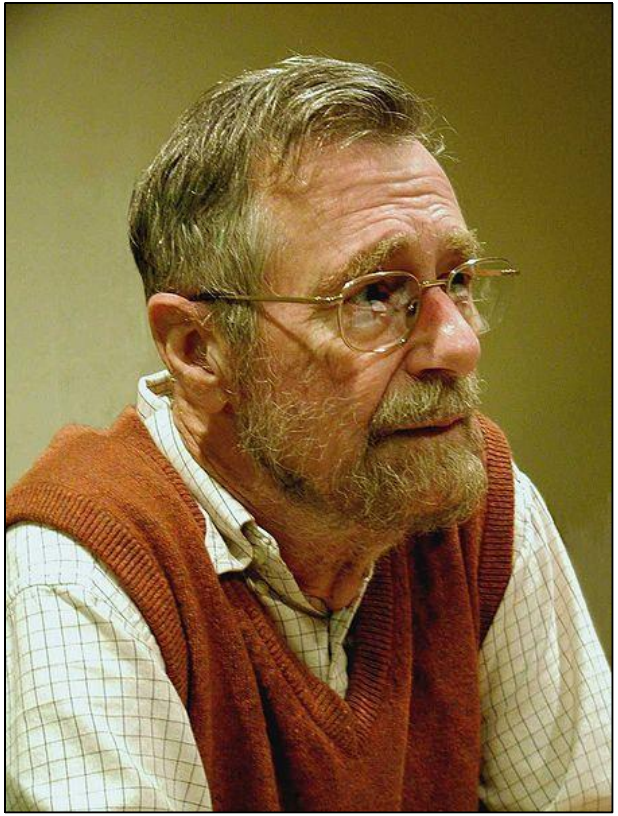
# Dijkstra’s Shortest Paths

(LIKE BREADTH-FIRST SEARCH, BUT **TAKES INTO ACCOUNT WEIGHT/DISTANCE** BETWEEN NODES)

# Edsger Dijkstra

1930-2002

- THE multiprogramming system (operating system)
  - Layers of abstraction!!
- Complier for a language that can do recursion
- Dining Philosopher's Problem (resource contention and deadlock)
- Dijkstra's algorithm
- "Goto considered harmful" (title given to his letter)



# The Structure of the "THE"-Multiprogramming System

Edsger W. Dijkstra
*Technological University, Eindhoven, The Netherlands*

A multiprogramming system is described in which all activities are divided over a number of sequential processes. These sequential processes are placed at various hierarchical levels, in each of which one or more independent abstractions have been implemented. The hierarchical structure proved to be vital for the verification of the logical soundness of the design and the correctness of its implementation.

KEY WORDS AND PHRASES: operating system, multiprogramming system, system hierarchy, system structure, real-time debugging, program verification, synchronizing primitives, cooperating sequential processes, system levels, input-output buffering, multiprogramming, processor sharing, multiprocessing*
CR CATEGORIES: 4.30, 4.32

## Introduction

In response to a call explicitly asking for papers "on timely research and development efforts," I present a progress report on the multiprogramming effort at the Department of Mathematics at the Technological University in Eindhoven.

Having very limited resources (viz. a group of six people of, on the average, half-time availability) and wishing to contribute to the art of system design—including all the stages of conception, construction, and verification,

Accordingly, I shall try to go beyond just reporting what we have done and how, and I shall try to formulate as well what we have learned.

I should like to end the introduction with two short remarks on working conditions, which I make for the sake of completeness. I shall not stress these points any further.

One remark is that production speed is severely slowed down if one works with half-time people who have other obligations as well. This is at least a factor of four; probably it is worse. The people themselves lose time and energy in switching over; the group as a whole loses decision speed as discussions, when needed, have often to be postponed until all people concerned are available.

The other remark is that the members of the group (mostly mathematicians) have previously enjoyed as good students a university training of five to eight years and are of Master's or Ph.D. level. I mention this explicitly because at least in my country the intellectual level needed for system design is in general grossly underestimated. I am convinced more than ever that this type of work is very difficult, and that every effort to do it with other than the best people is doomed to either failure or moderate success at enormous expense.

## The Tool and the Goal

The system has been designed for a Dutch machine, the EL X8 (N.V. Electrologica, Rijswijk (ZH)). Charac-

# On the cruelty of really teaching computing science

The second part of this talk pursues some of the scientific and educational consequences of the assumption that computers represent a radical novelty. In order to give this assumption clear contents, we have to be much more precise as to what we mean in this context by the adjective "radical". We shall do so in the first part of this talk, in which we shall furthermore supply evidence in support of our assumption.

The usual way in which we plan today for to-morrow is in yesterday's vocabulary. We do so, because we try to get away with the concepts we are familiar with and that have acquired their meanings

## Dijkstra's Algorithm

- Mark all nodes as gray.
- Mark the initial node ***s*** as yellow and at candidate distance **0**.
- Enqueue ***s*** into the priority queue with priority **0**.
- While not all nodes have been visited:
- Dequeue the lowest-cost node ***u*** from the priority queue.
- Color ***u*** green. The candidate distance ***d*** that is currently stored for node ***u*** is the length of the shortest path from ***s*** to ***u***.
- If ***u*** is the destination node ***t***, you have found the shortest path from ***s*** to ***t*** and are done.
- For each node ***v*** connected to ***u*** by an edge of length ***L***:
  - If ***v*** is gray:
    - Color ***v*** yellow.
    - Mark ***v***'s distance as ***d* + *L***.
    - Set ***v***'s parent to be ***u***.
    - Enqueue ***v*** into the priority queue with priority ***d* + *L***.
  - If ***v*** is yellow and the candidate distance to ***v*** is greater than ***d* + *L***:
    - Update ***v***'s candidate distance to be ***d* + *L***.
    - Update ***v***'s parent to be ***u***.
    - Update ***v***'s priority in the priority queue to ***d* + *L***.