Programming Abstractions

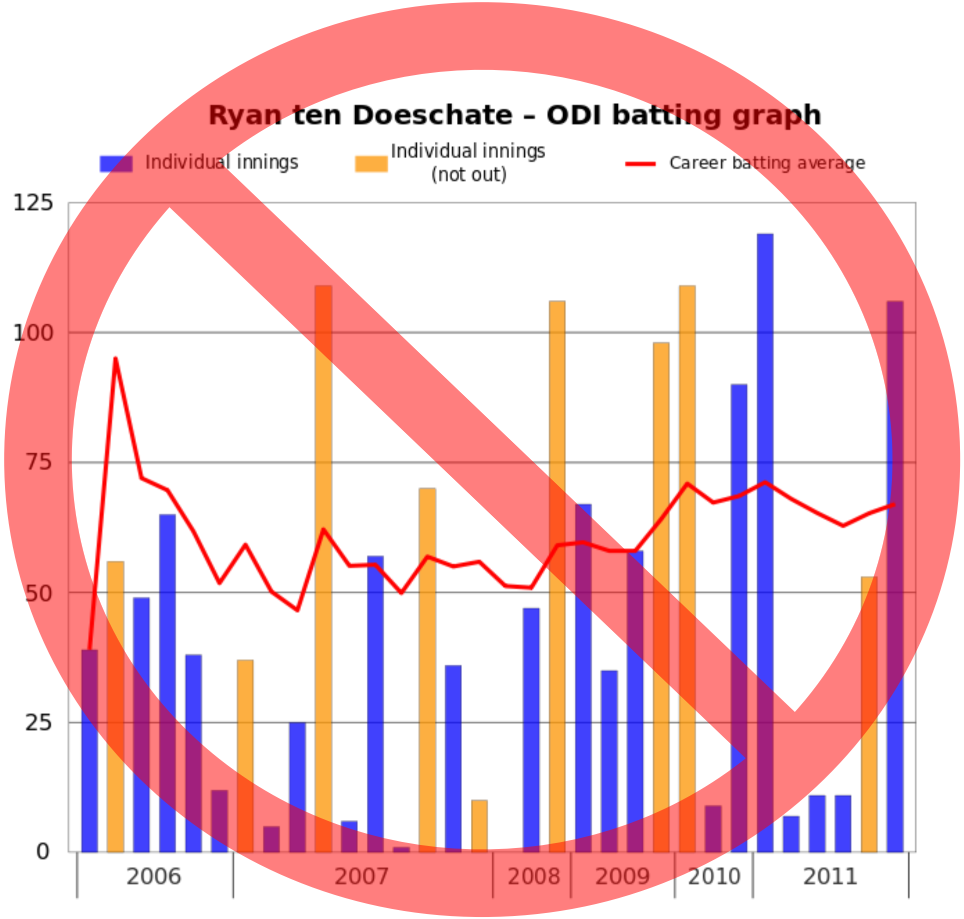
CS106B

Cynthia Bailey Lee
Julie Zelenski

Graphs

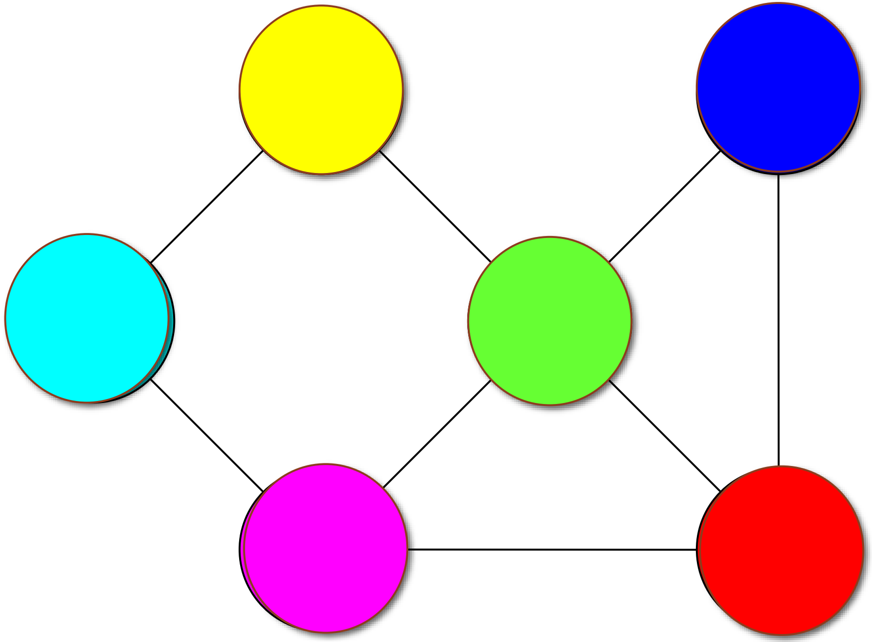
What are graphs? What are they good for?

Graph



Ryan ten Doeschate – ODI batting graph
Individual innings
Individual innings (not out)
Career batting average
125
100
75
50
25
0
2006
2007
2008
2009
2010
2011

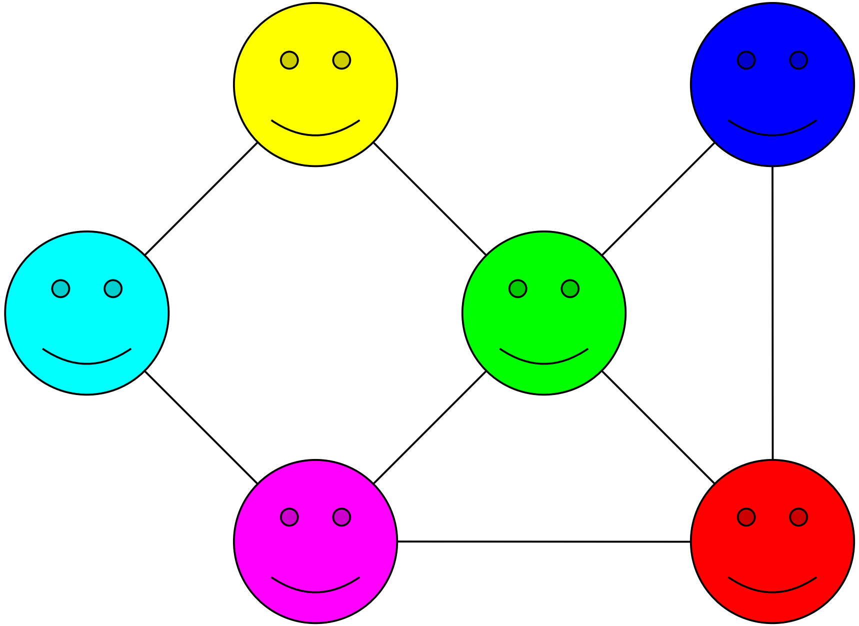
Graphs in Computer Science

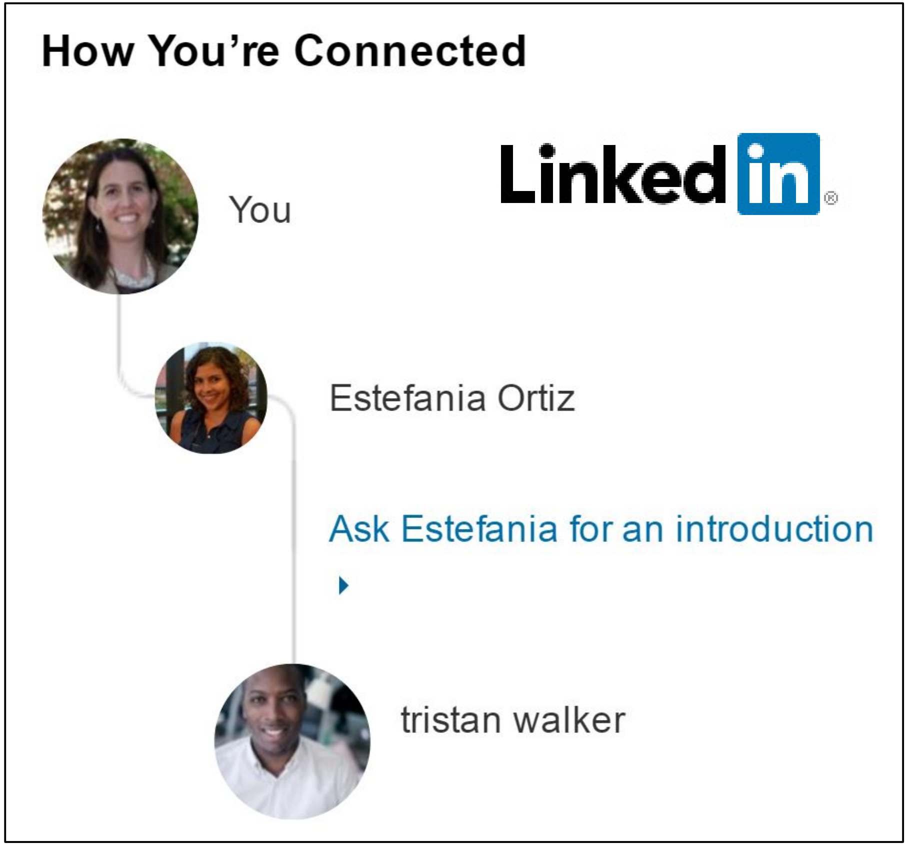


A graph is a mathematical structure for representing relationships

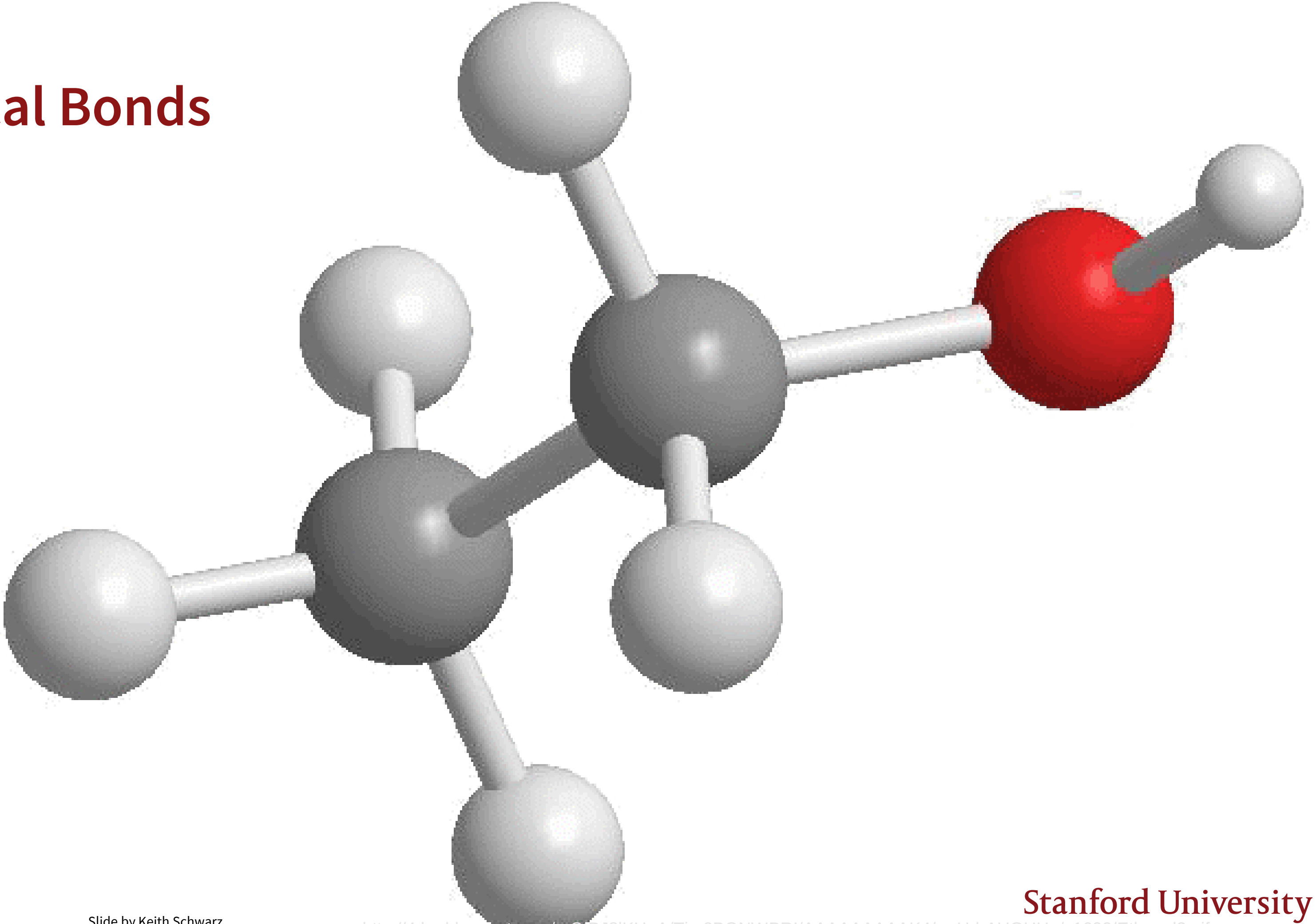
- A set V of **vertices** (or *nodes*)
- A set E of **edges** (or *arcs*) connecting a pair of vertices

A Social Network



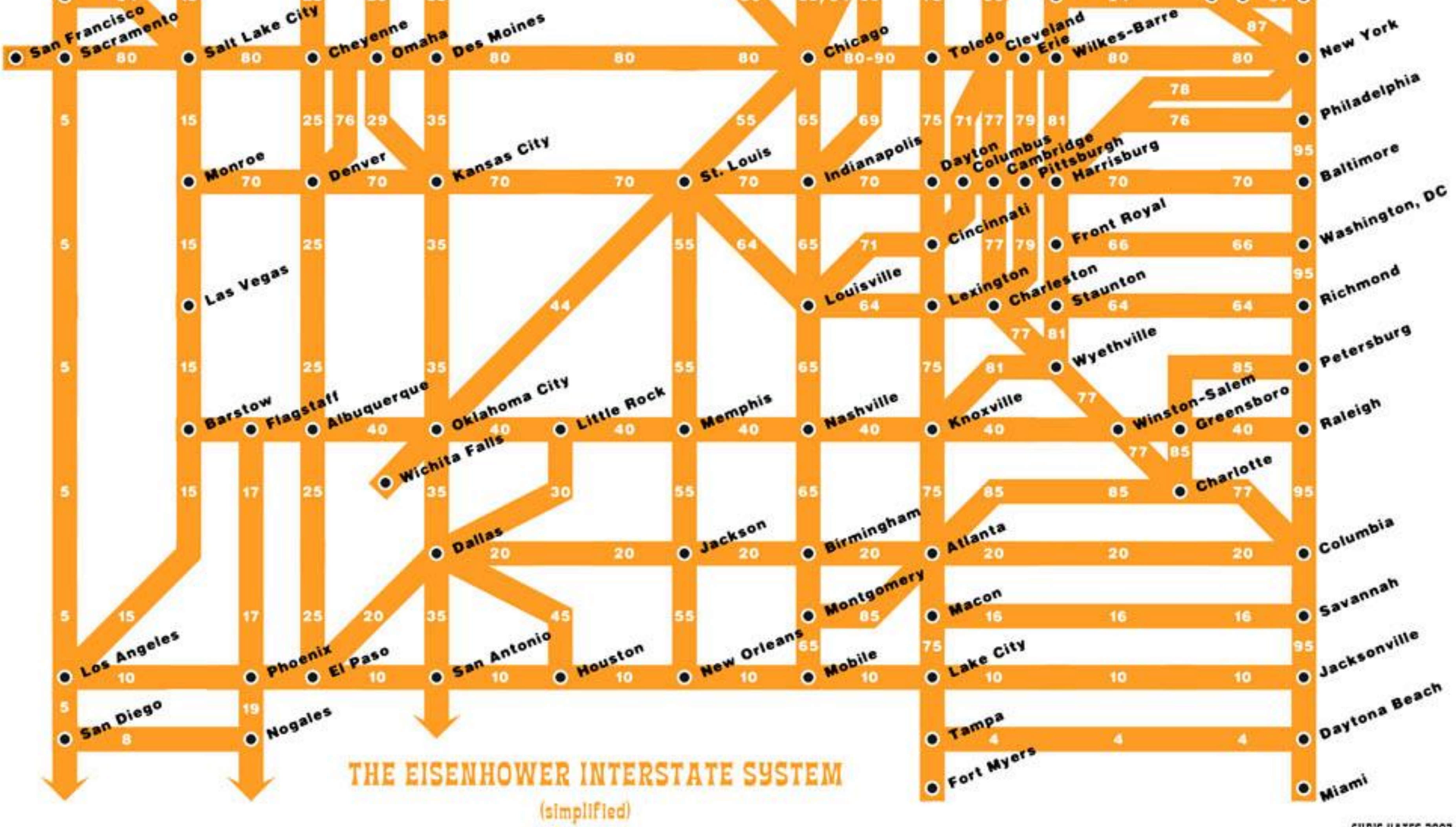


Chemical Bonds



Slide by Keith Schwarz

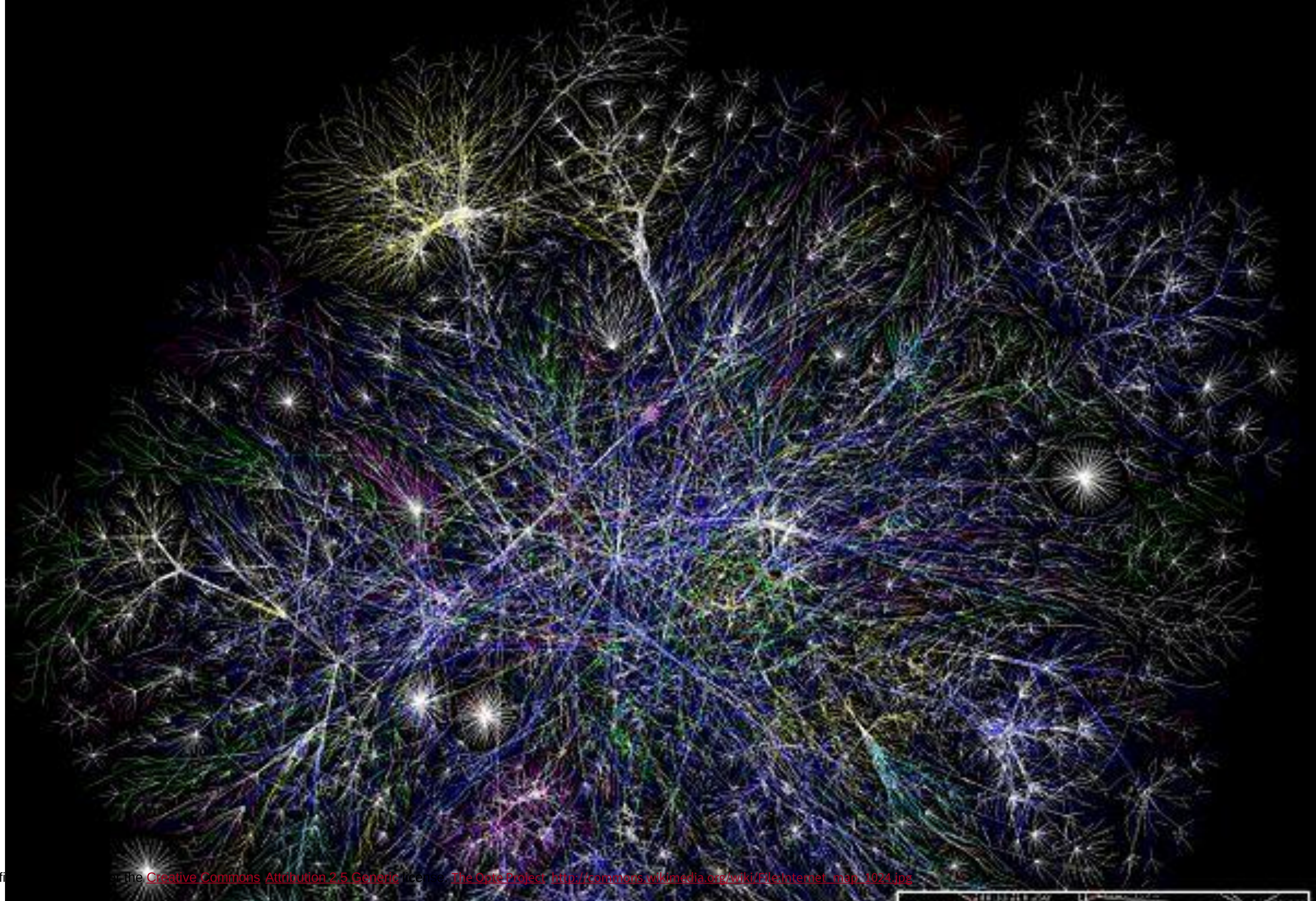
http://4.bp.blogspot.com/-xGIBJ8lKHqA/Tjm0BONWBRI/AAAAAAAAAK4/-mHrbAUOHHg/s1600/Ethanol2.gif



http://strangemaps.files.wordpress.com/2007/02/fullinterstatemap-web.jpg

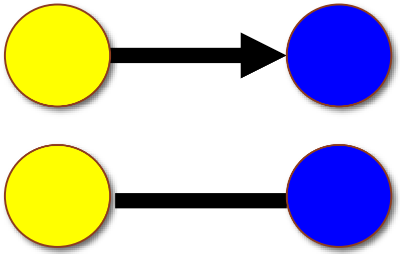
Slide by Keith Schwarz

Internet



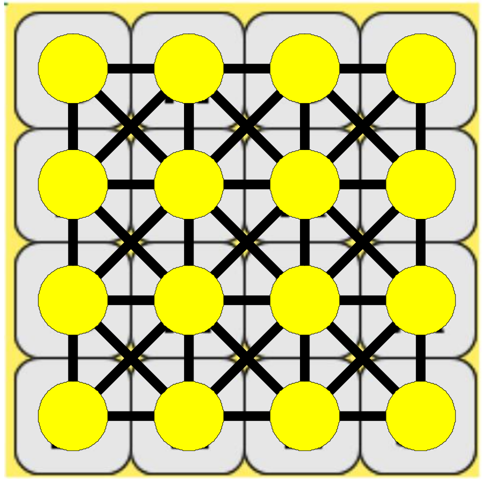
Graphs: basic terminology

- A set V of **vertices** (or *nodes*)
  - Often have an associated label
- A set E of **edges** (or *arcs*) connecting a pair of vertices
  - Often have an associated cost or weight
- A graph may be **directed** (an edge from A to B only allow you to go from A to B, not B to A)
- or **undirected** (an edge between A and B allows travel in both directions)
- We talk about the number of vertices or edges as the *size of the set*, using the set theory notation for size: **|V|** and **|E|**



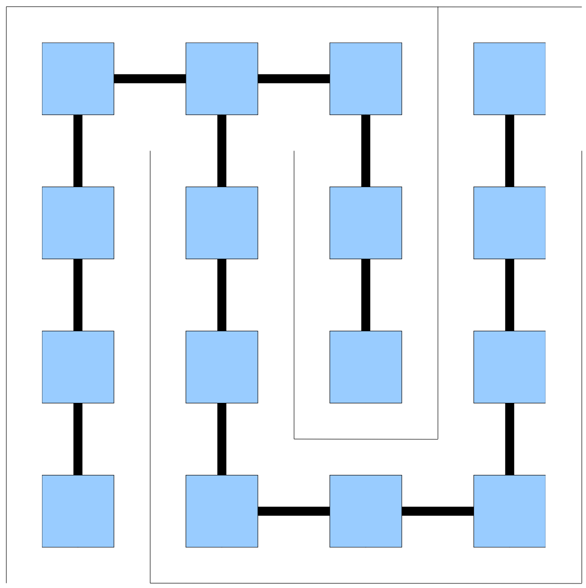
Boggle as a graph

Vertex = letter cube; Edge = connection to neighboring cube



Maze as graph

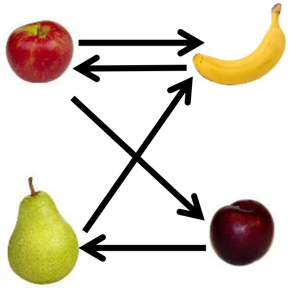
If a maze is a graph, what is a vertex and what is an edge?



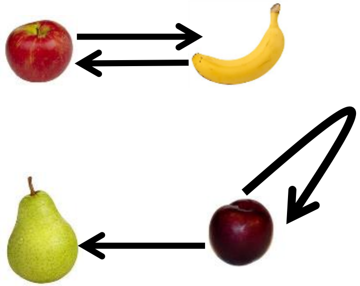
Graphs

All of the following are valid graphs:

A graph could be a single node



An example of a directed graph with 4 nodes



Graphs don't have to be connected (notice this one has two separated parts)

Graph Terminology

Graph terminology: Paths

path: A path from vertex *a* to *b* is a sequence of edges that can be followed starting from *a* to reach *b*.

neighbor or **adjacent***:* Two vertices connected directly by an edge.

reachable: Vertex *a* is *reachable* from *b* if a path exists from *a* to *b*.

connected: A graph is *connected* if every vertex is reachable from every other.

cycle: A path that begins and ends at the same node.

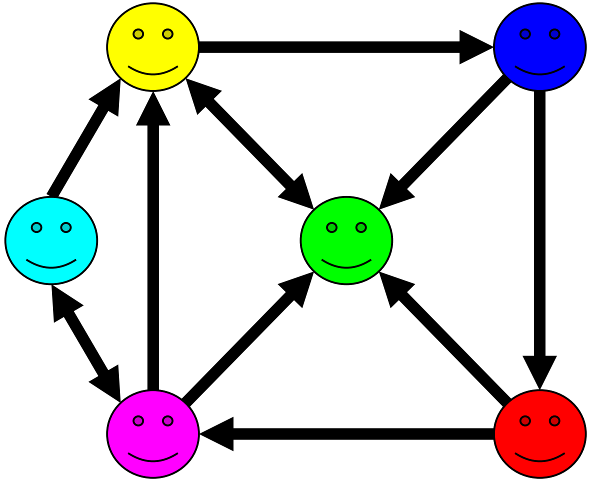
Representing Graphs

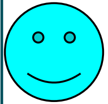
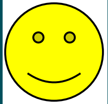
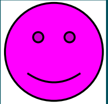
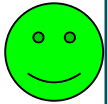
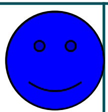
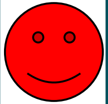
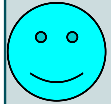
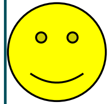
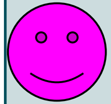
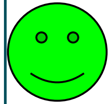
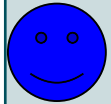
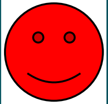
WAYS WE COULD IMPLEMENT A GRAPH CLASS

Adjacency Matrix

Representing Graphs: Adjacency matrix

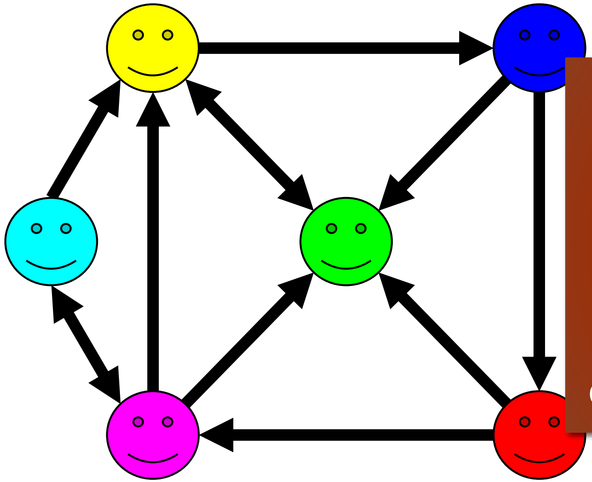
We can represent a graph as a **`Grid<bool>`** (unweighted)



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

Representing Graphs: Adjacency matrix

We can represent a graph as a **`Grid<bool>`** (unweighted)



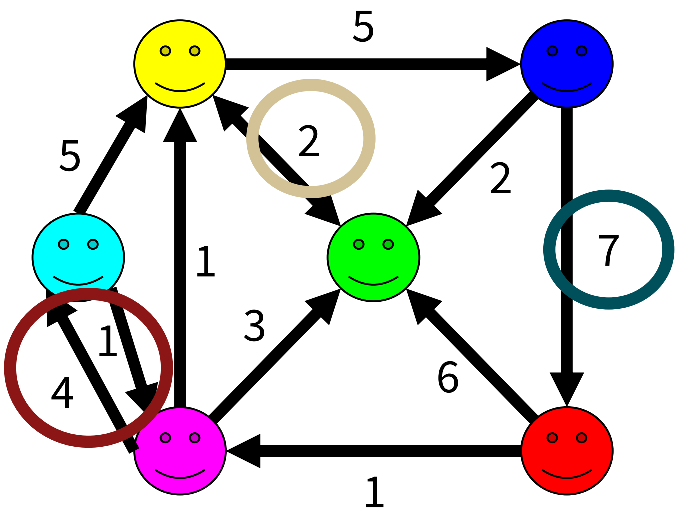
	0	1	1	0	0	0
	0	0	0	1	1	0
	1	1	0	1	0	0
	0	1	0	0	0	0
	0	0	0	1	0	1
	0	0	1	1	0	0

Your Turn: what aspect of the picture does this 0/false correspond to?

Your Turn: which edge in the picture does this 1/true correspond to?

Representing Graphs: Adjacency matrix

We can represent a graph as a **`Grid<int>`** (weighted)



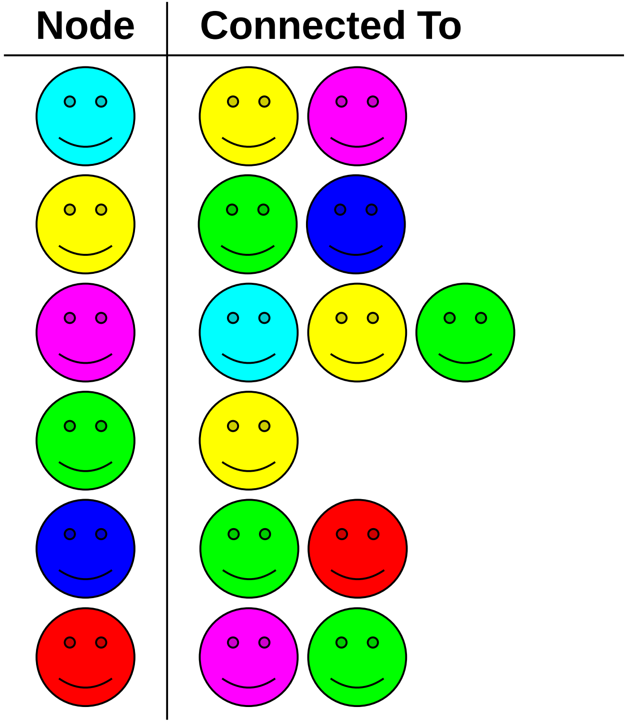
	cyan	yellow	magenta	green	blue	red
cyan	0	5	1	0	0	0
yellow	0	0	0	2	5	0
magenta	4	1	0	3	0	0
green	0	2	0	0	0	0
blue	0	0	0	2	0	7
red	0	0	1	6	0	0

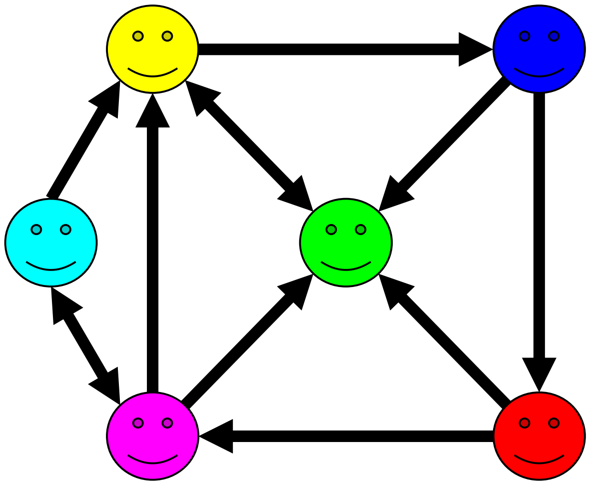
Adjacency List

Representing Graphs: Adjacency list

We can represent a graph as a map from nodes to the set of nodes each node is connected to.

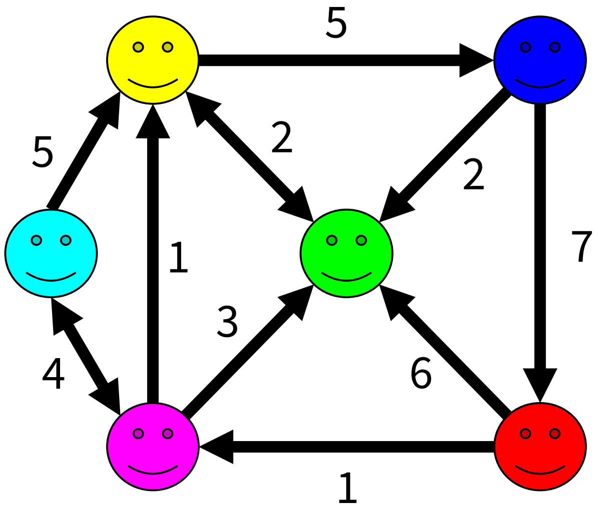
`Map<Node*, Set<Node*>>`



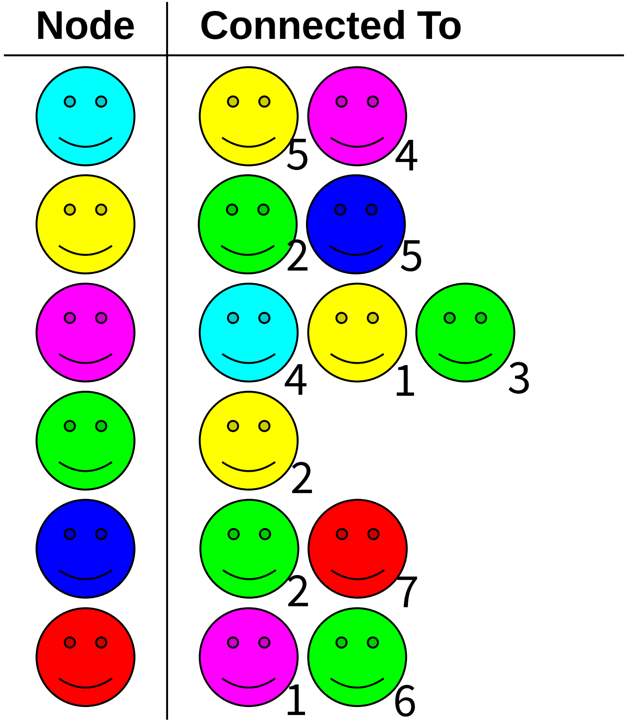


Representing Graphs: Adjacency list

We can represent a graph as a map from nodes to the set of nodes each node is connected to.




```
Map<Node*, Set<Edge*>>
```



Stanford University

Slide by Keith Schwarz

Your Turn: choosing an implementation

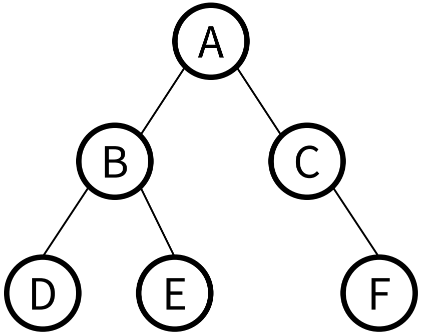
- Which implementation would you choose for the following circumstances:
  - Nodes = Facebook accounts (about 3 billion of them)
  - Edges = the two accounts are “Friends” with each other
- Answer each of the following:
  - Should your graph be weighted or unweighted?
  - Should your graph be directed or undirected?
  - Should you use Adjacency Matrix or Adjacency List?
  - And explain why ☺

Breadth-First Search

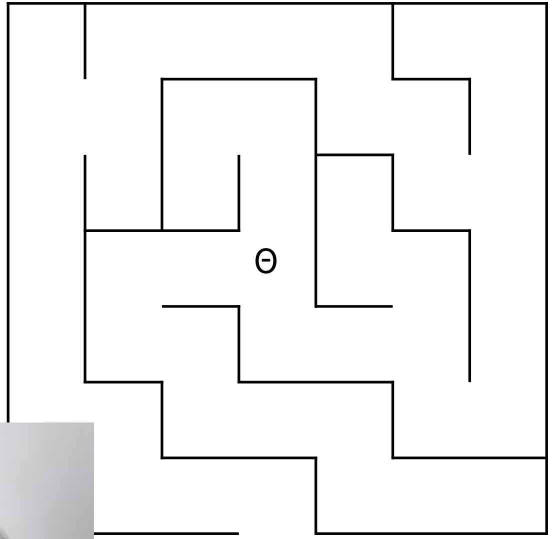
WE'VE SEEN BFS BEFORE THIS QUARTER!

BFS in this class so far

Assignment



Trees



Maze

Slime Mold

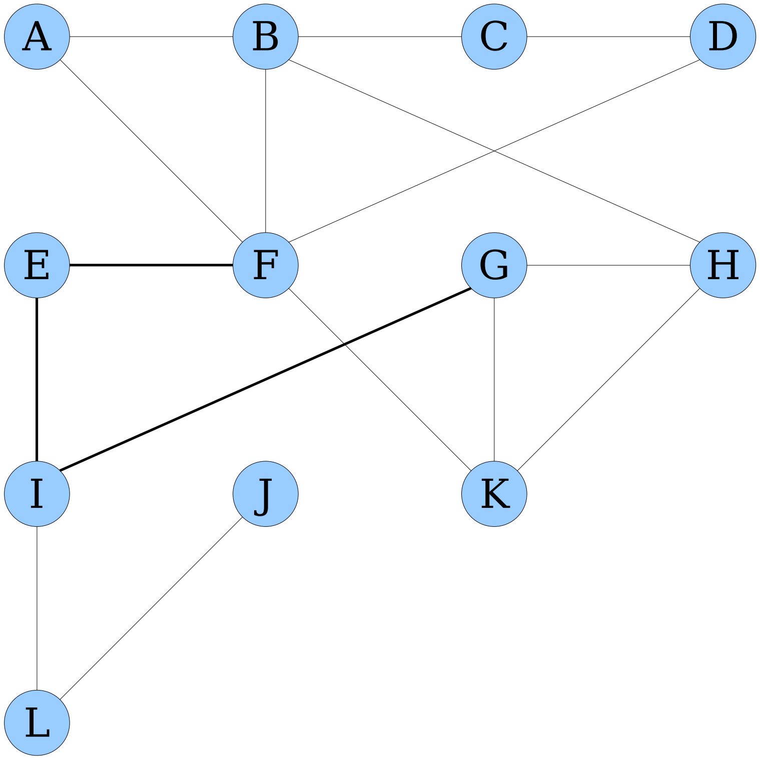
Generic BFS algorithm pseudocode

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

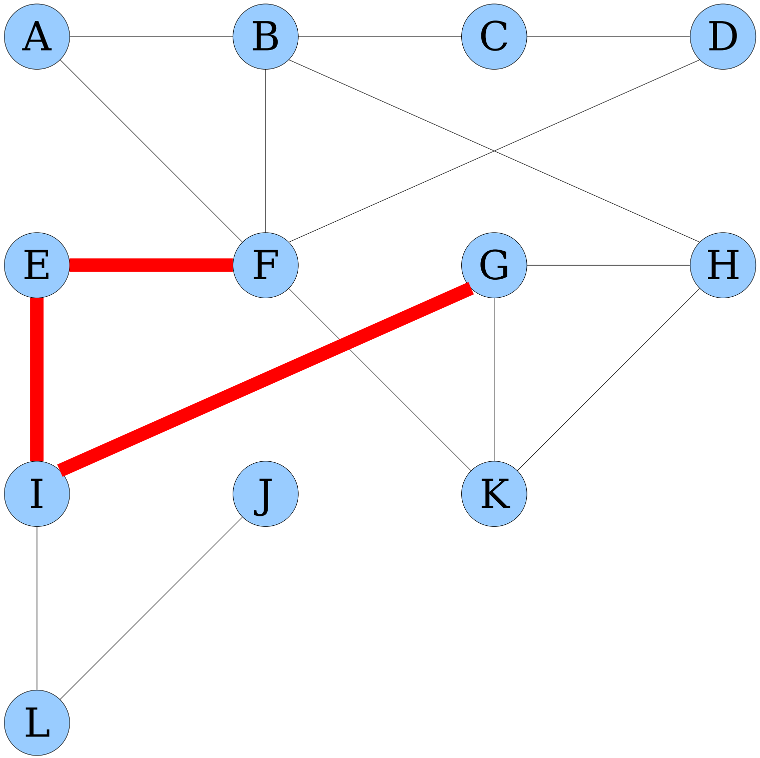
GRAPH ALGORITHMS

Breadth-First Search



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

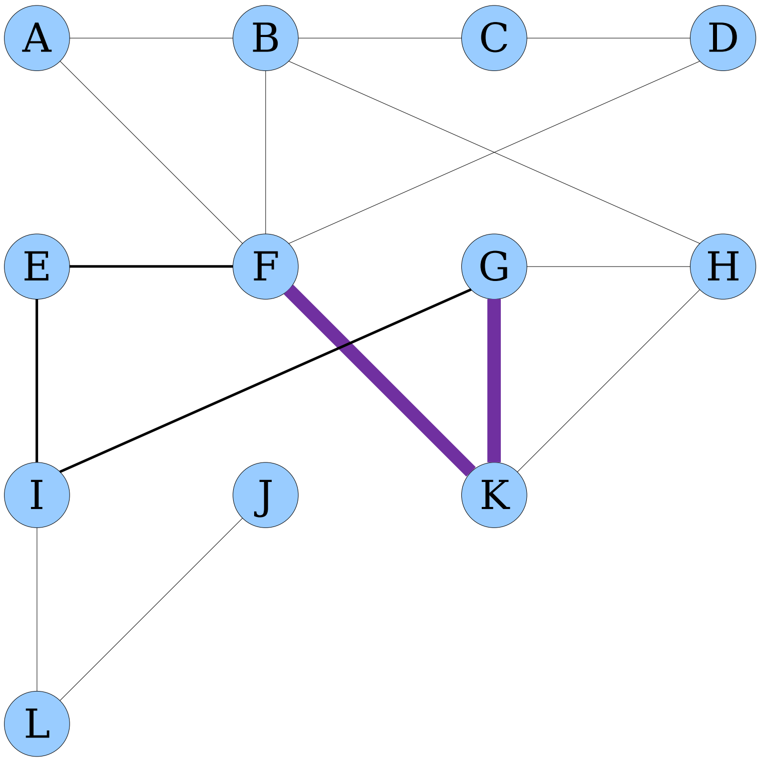
Breadth-First Search



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

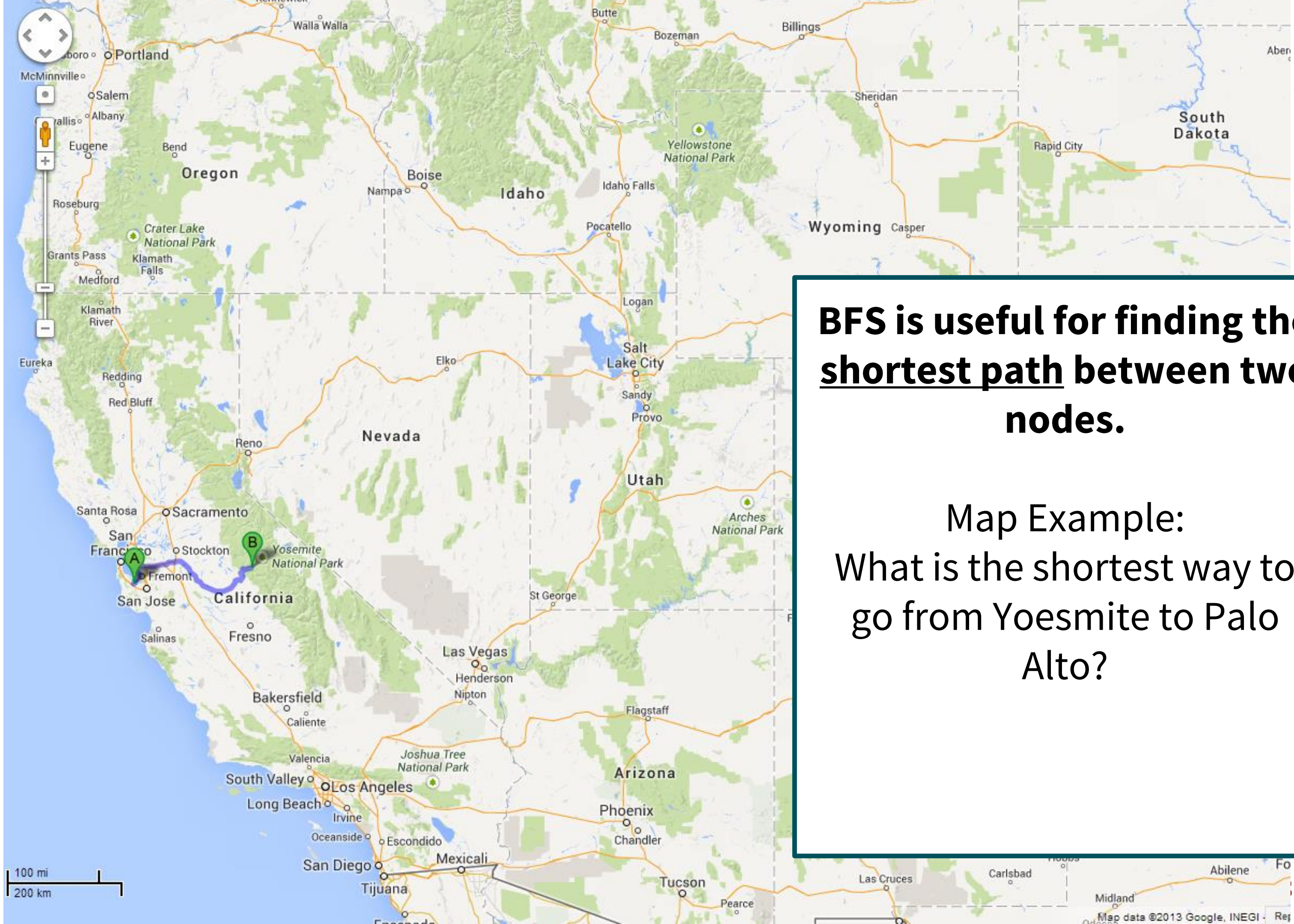
One way (not the shortest):
F->E->I->G
3 edges

Breadth-First Search



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

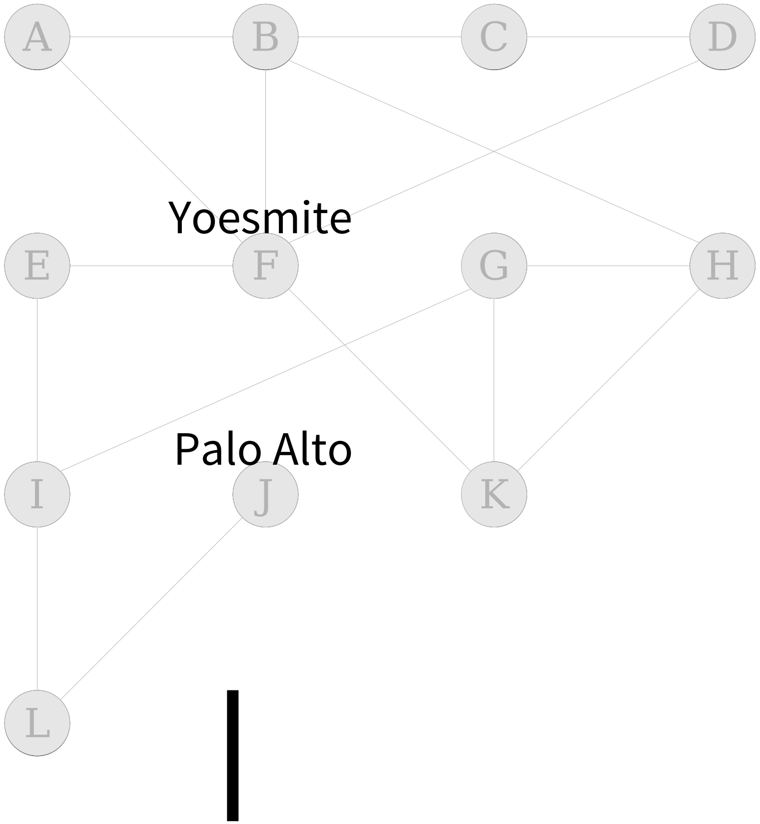
Shortest way:
F->K->G
2 edges



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

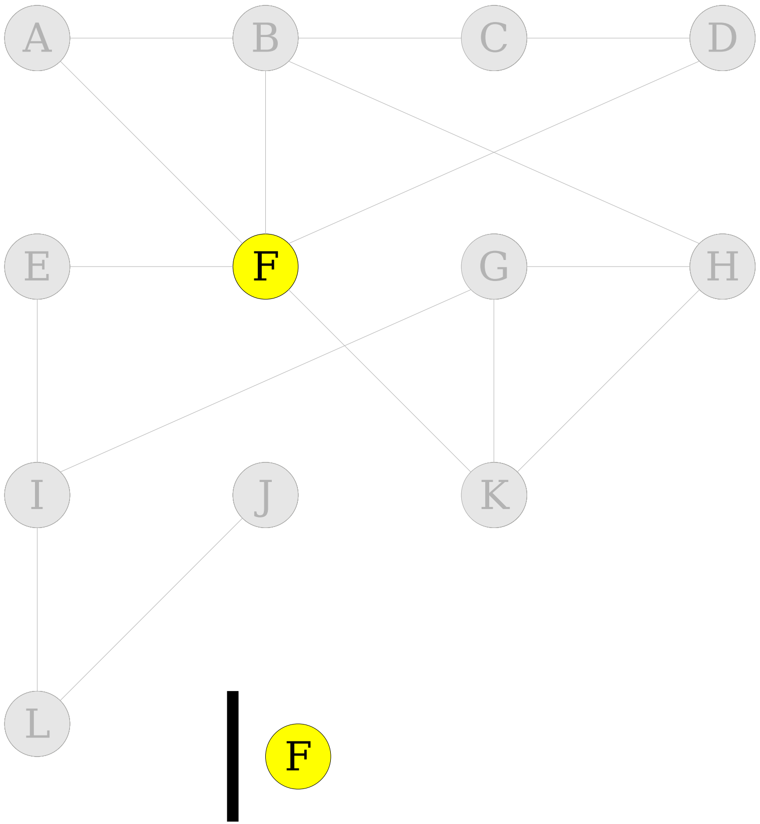
Map Example:
What is the shortest way to go from Yoesmite to Palo Alto?

A BFS algorithm for graphs with a special property…



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

A BFS algorithm for graphs with a special property…

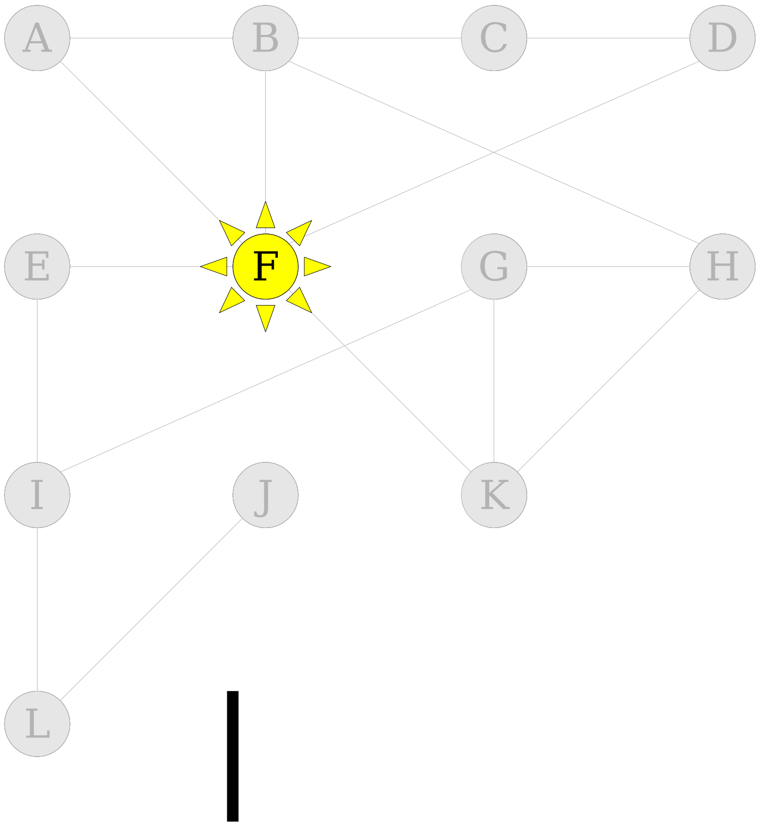


TO START (2):

(1) Enqueue the desired **start** node

(2) Note that anytime we enqueue a node, we mark it YELLOW

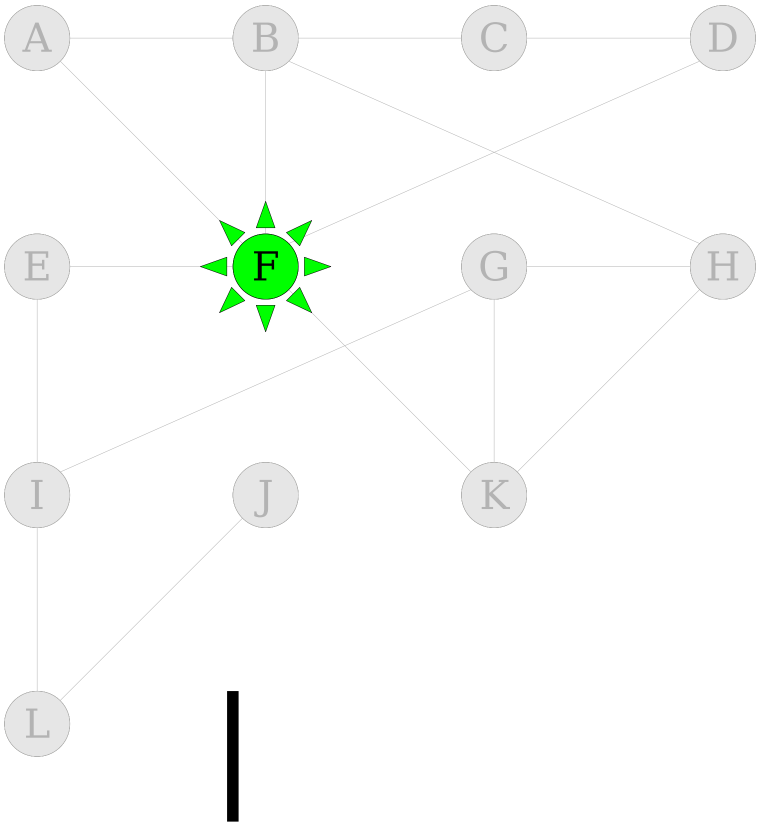
A BFS algorithm for graphs with a special property…



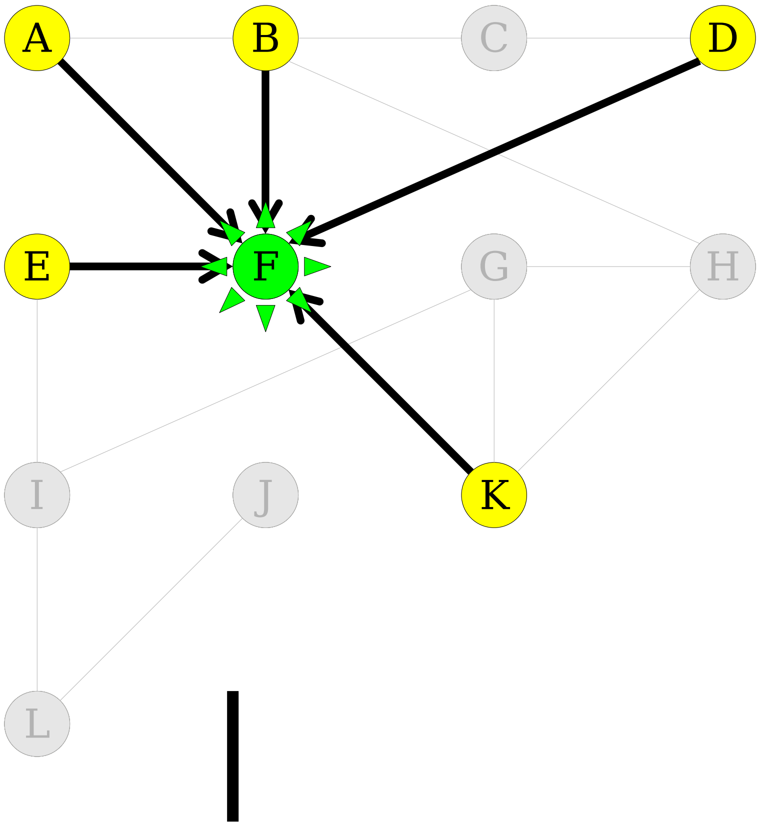
LOOP PROCEDURE:

(1) Dequeue a node
(2) Mark current node GREEN
(3) Set current node's GREY neighbors' parent pointers to current node, then enqueue them (remember: set them YELLOW)

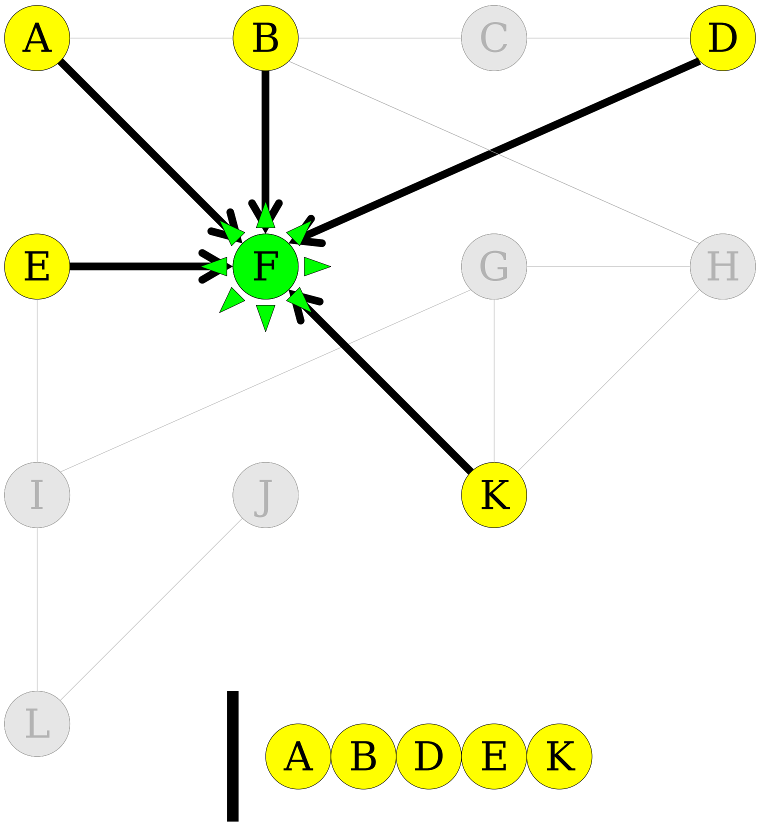
Breadth-First Search



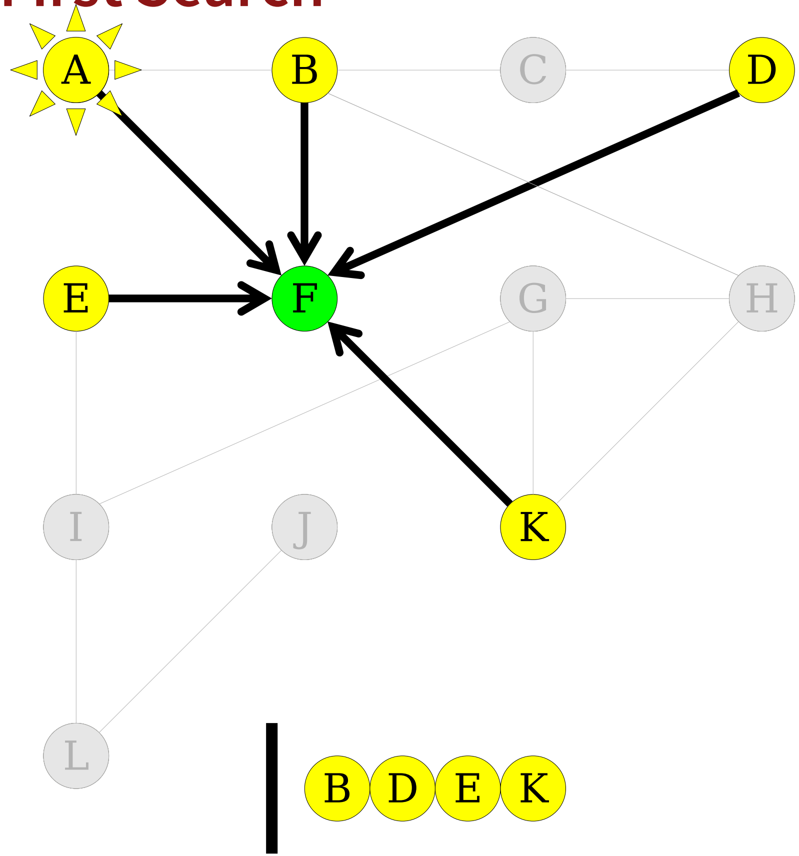
Breadth-First Search



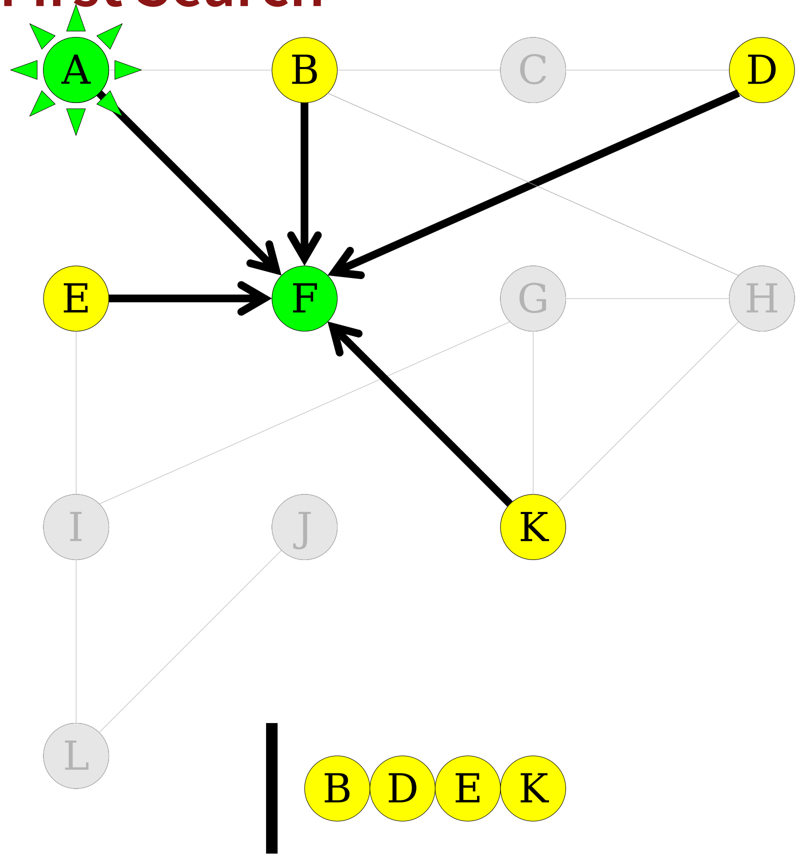
Breadth-First Search



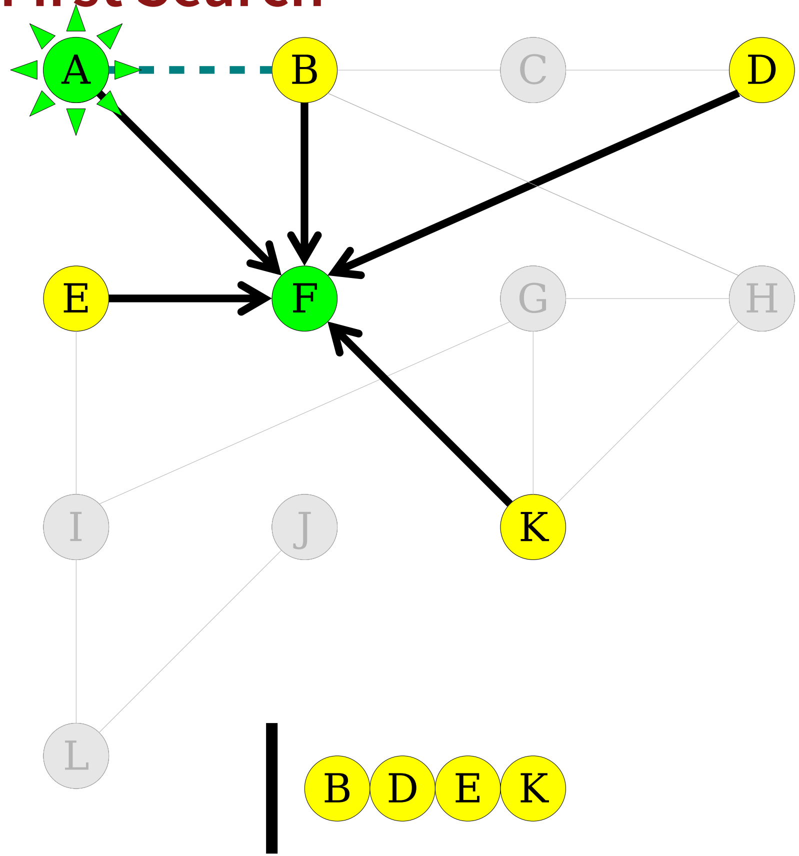
Breadth-First Search



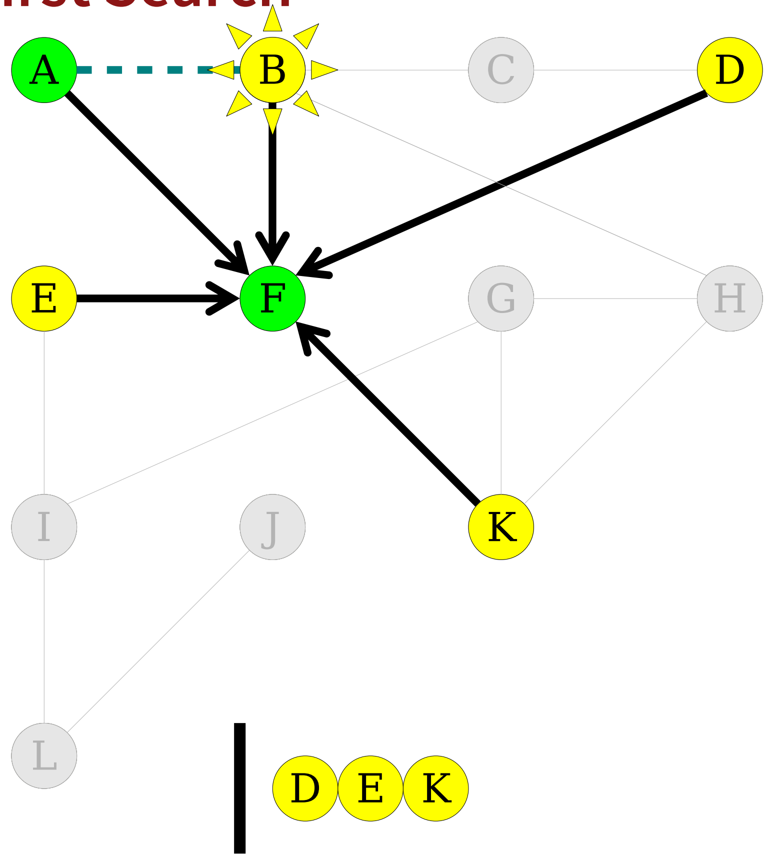
Breadth-First Search



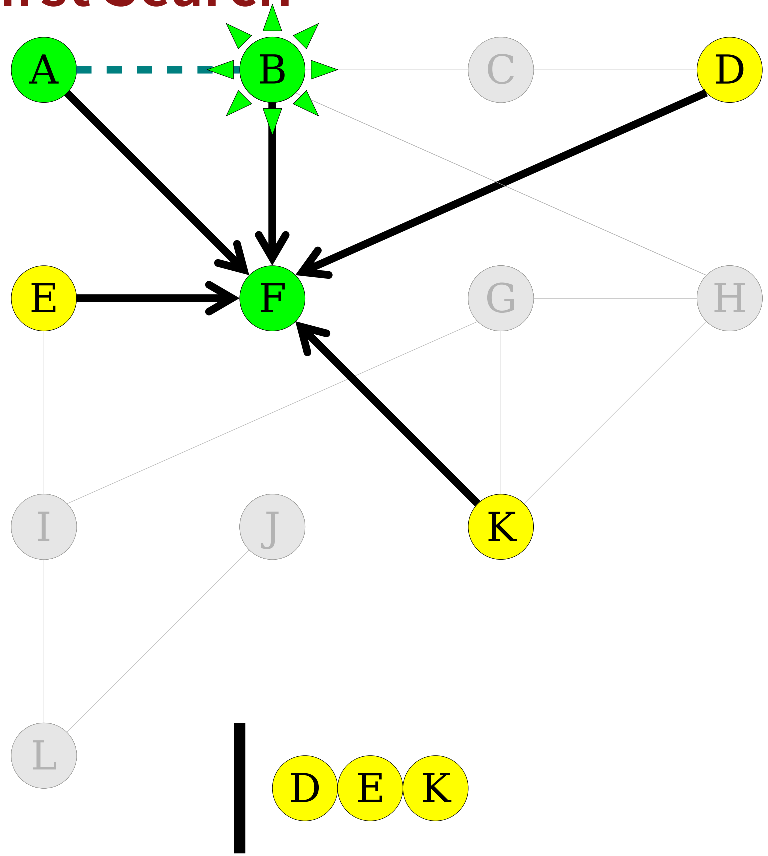
Breadth-First Search



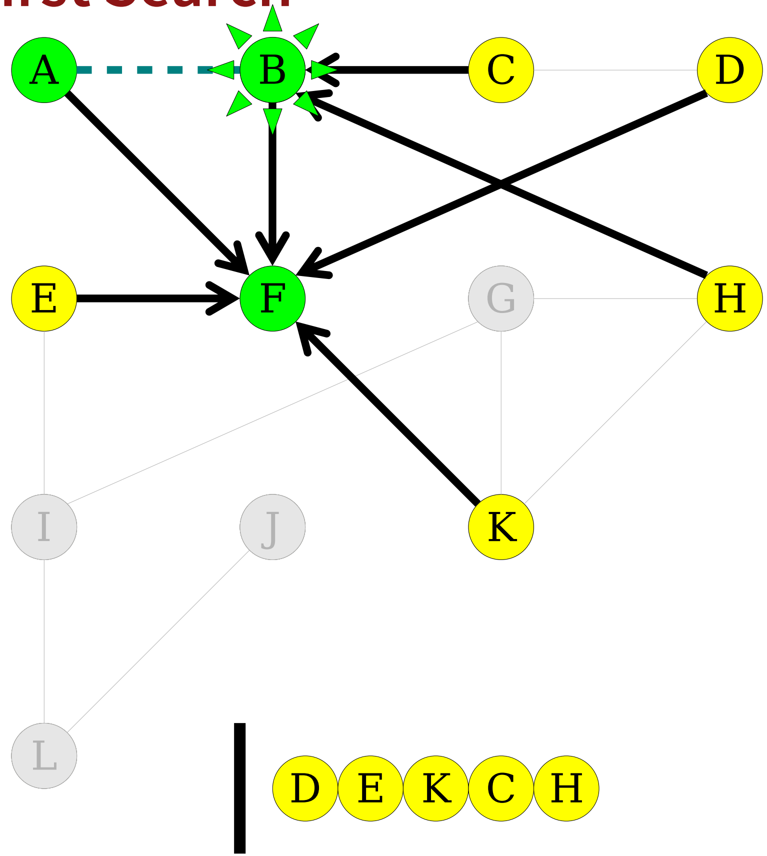
Breadth-First Search



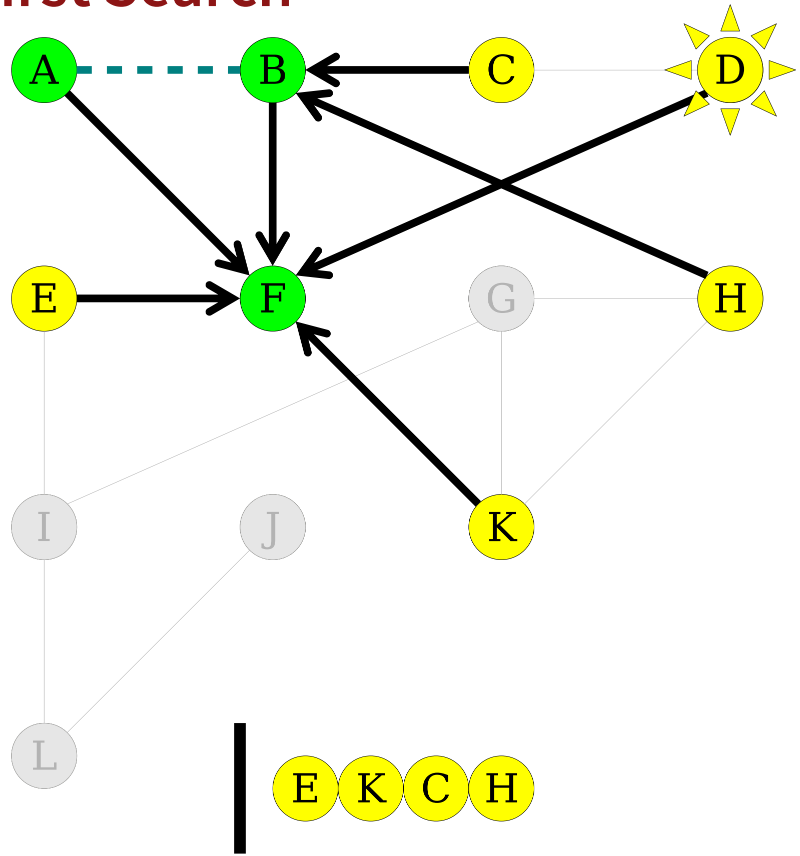
Breadth-First Search



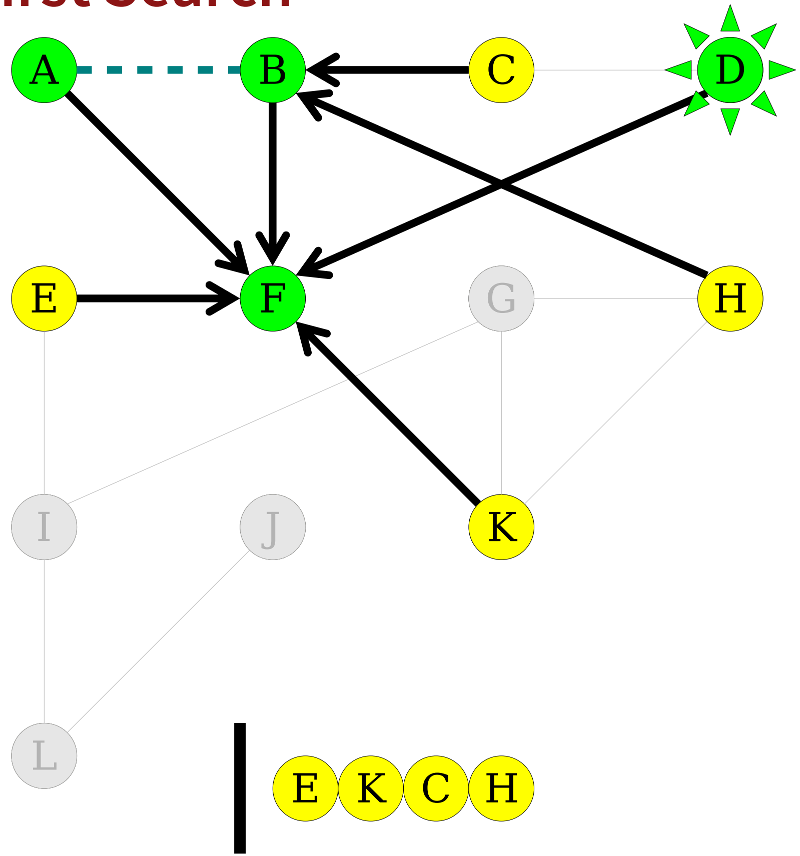
Breadth-First Search



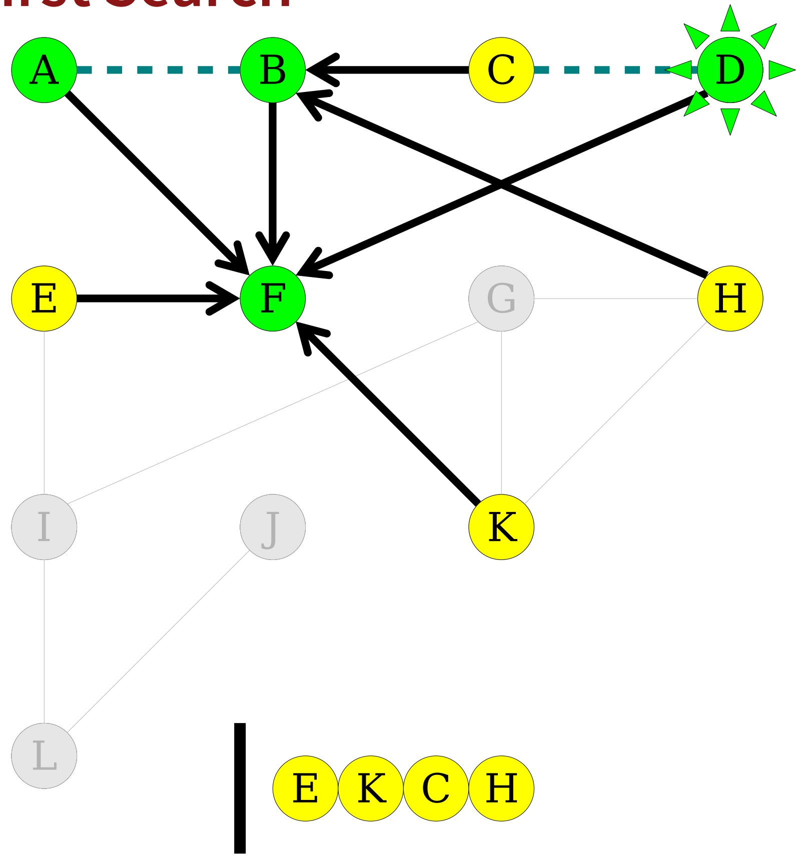
Breadth-First Search



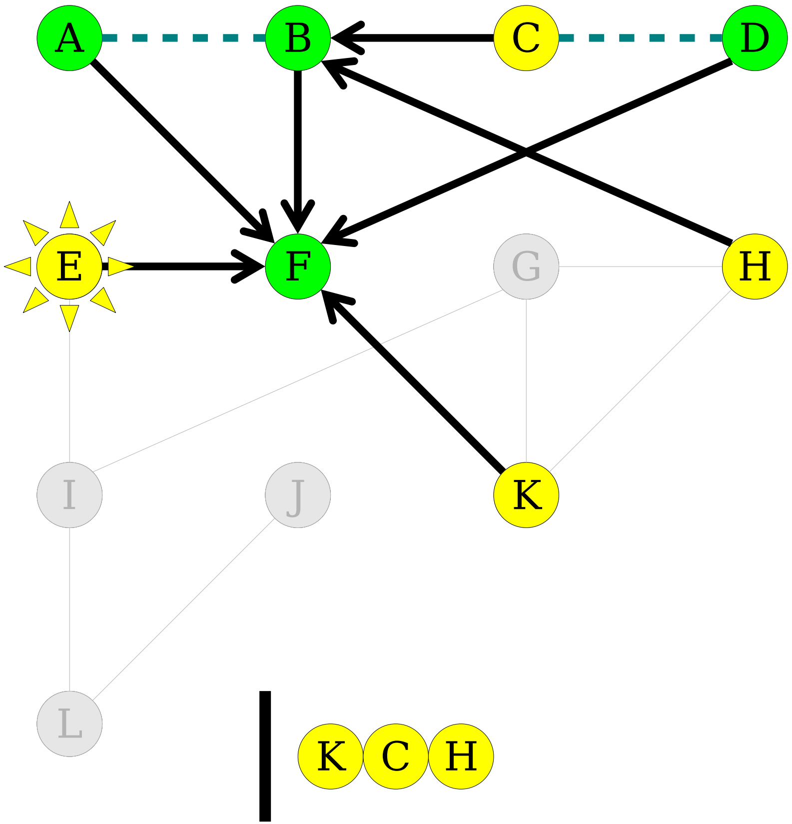
Breadth-First Search



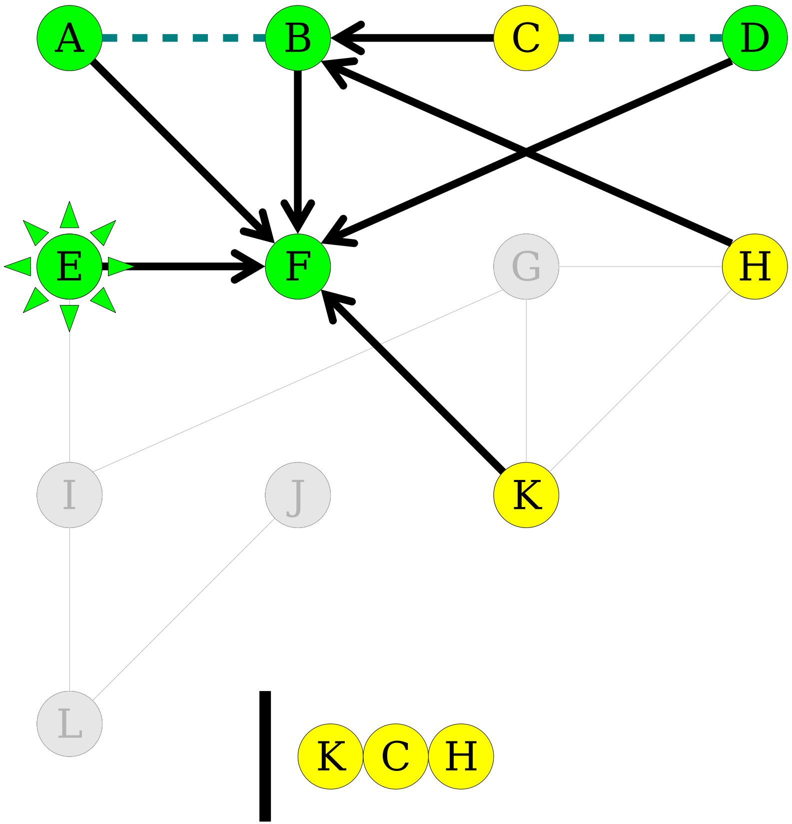
Breadth-First Search



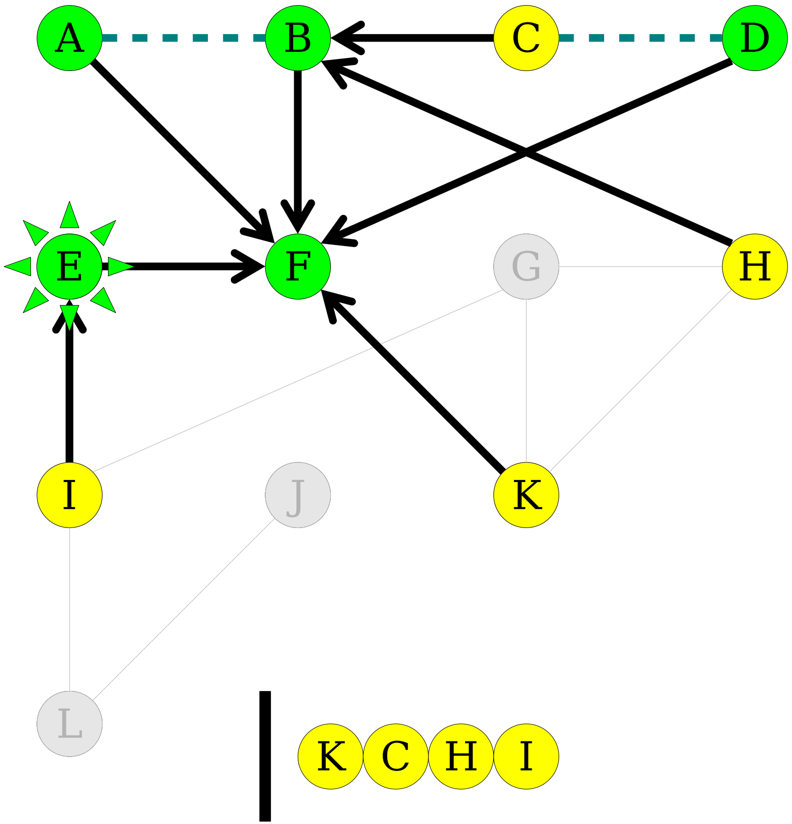
Breadth-First Search



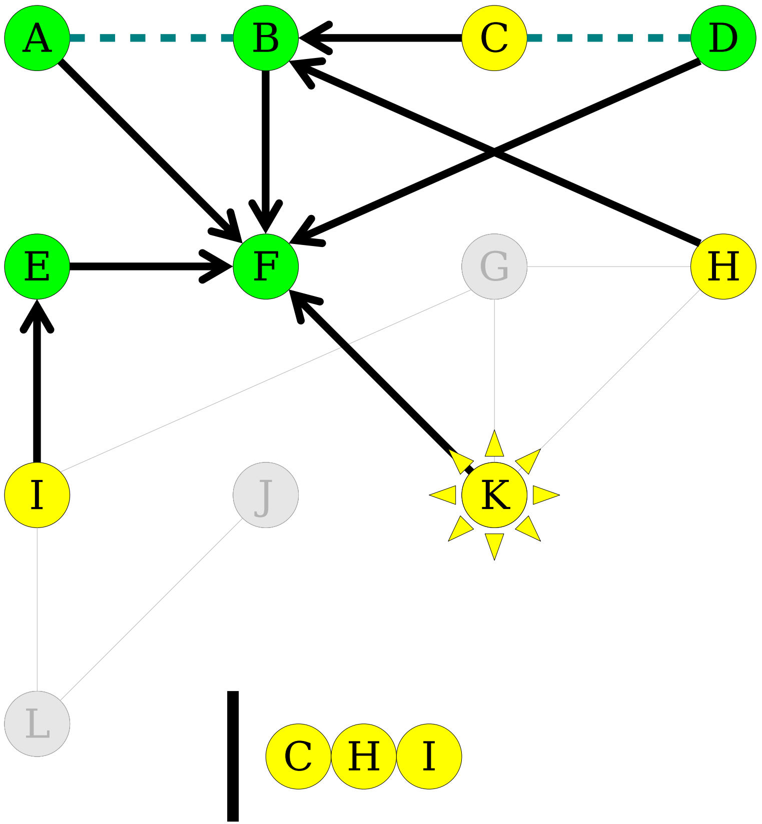
Breadth-First Search



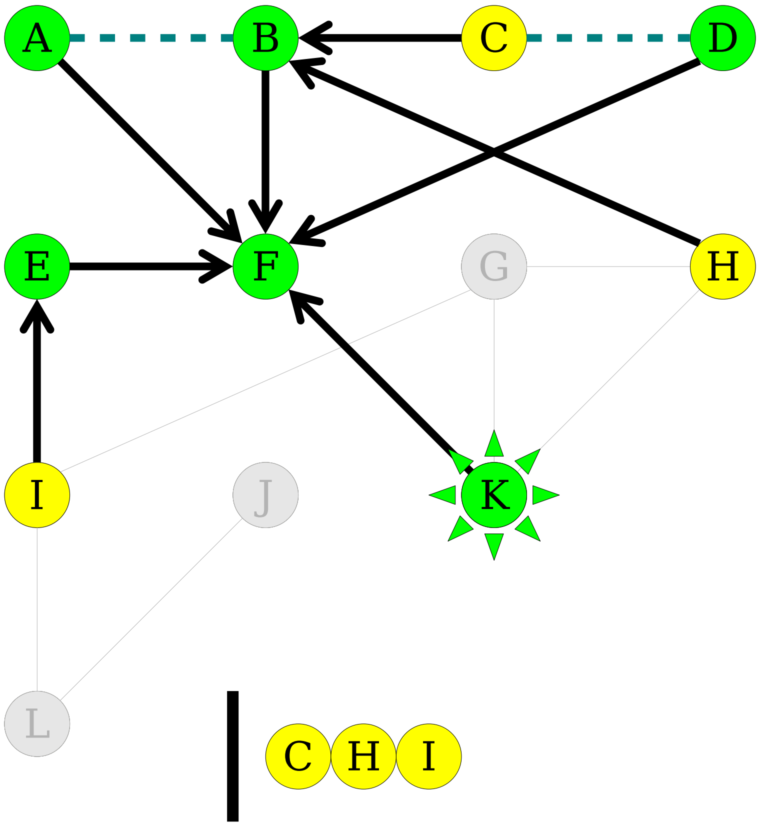
Breadth-First Search



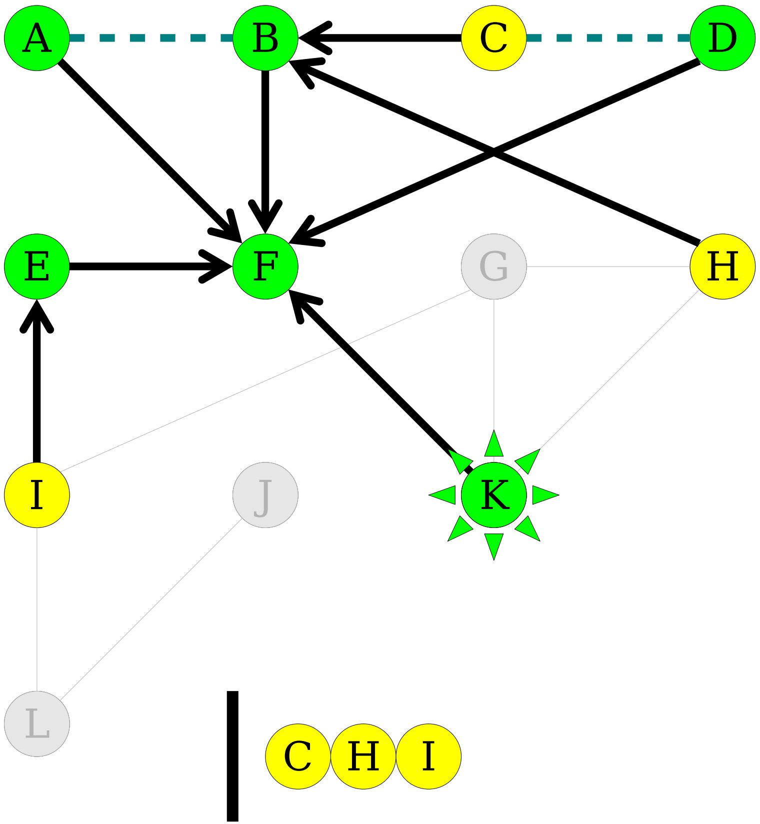
Breadth-First Search



Breadth-First Search



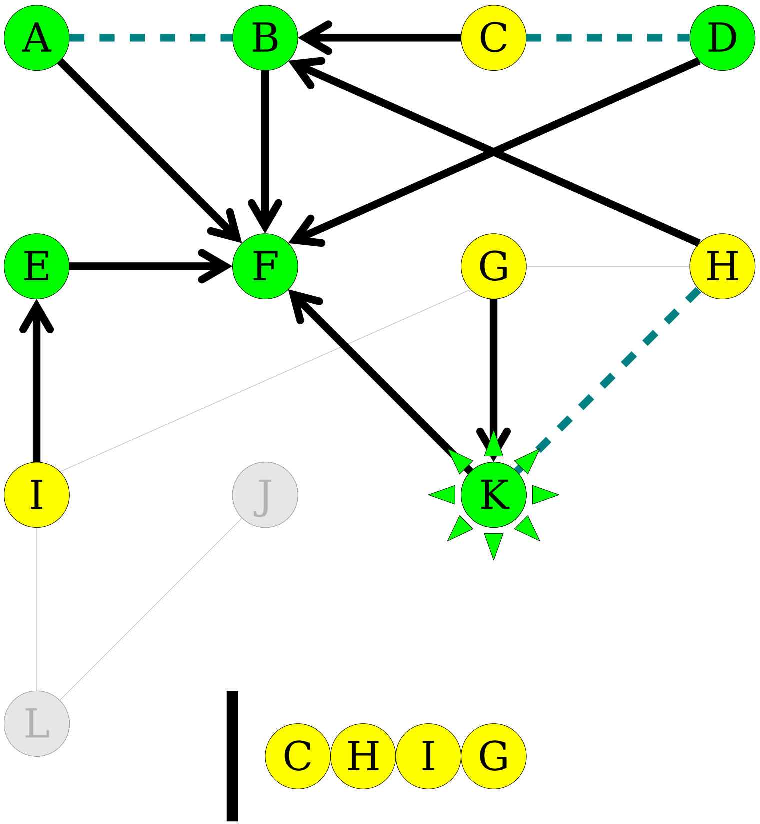
Breadth-First Search



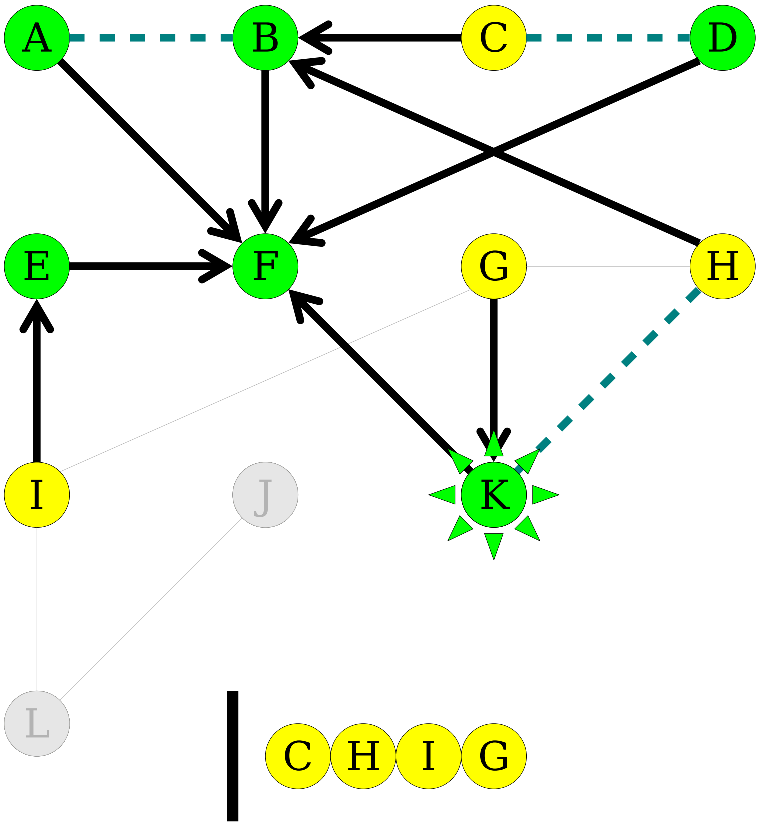
You predict the next slide!

A. K's neighbors F,G,H are yellow and in the queue and their parents are pointing to K
B. K's neighbors G,H are yellow and in the queue and their parents are pointing to K
C. K's neighbors G,H are yellow and in the queue
D. Other/none/more

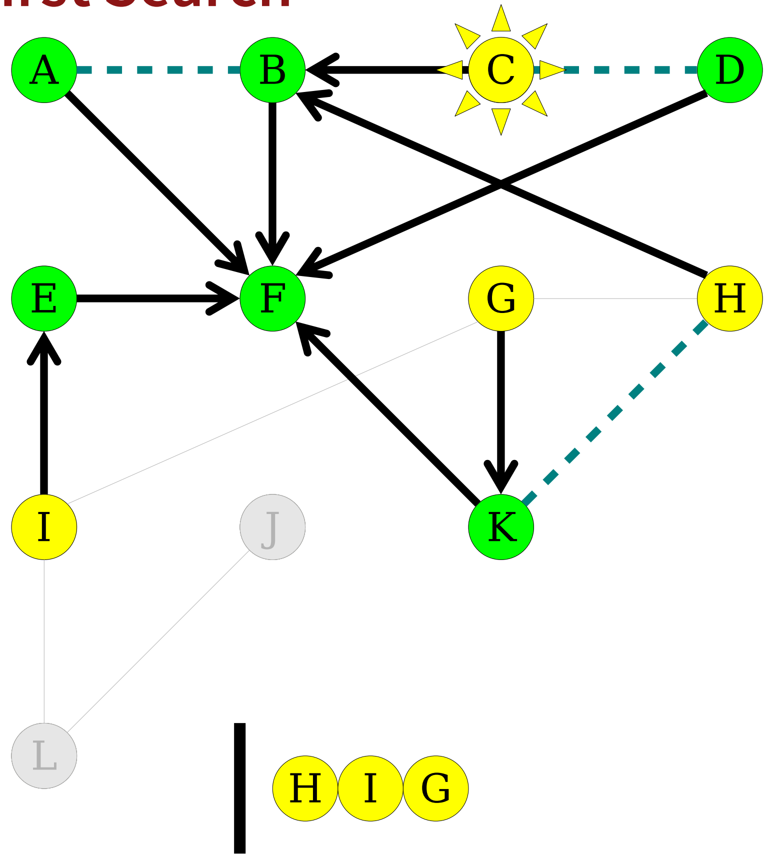
Breadth-First Search



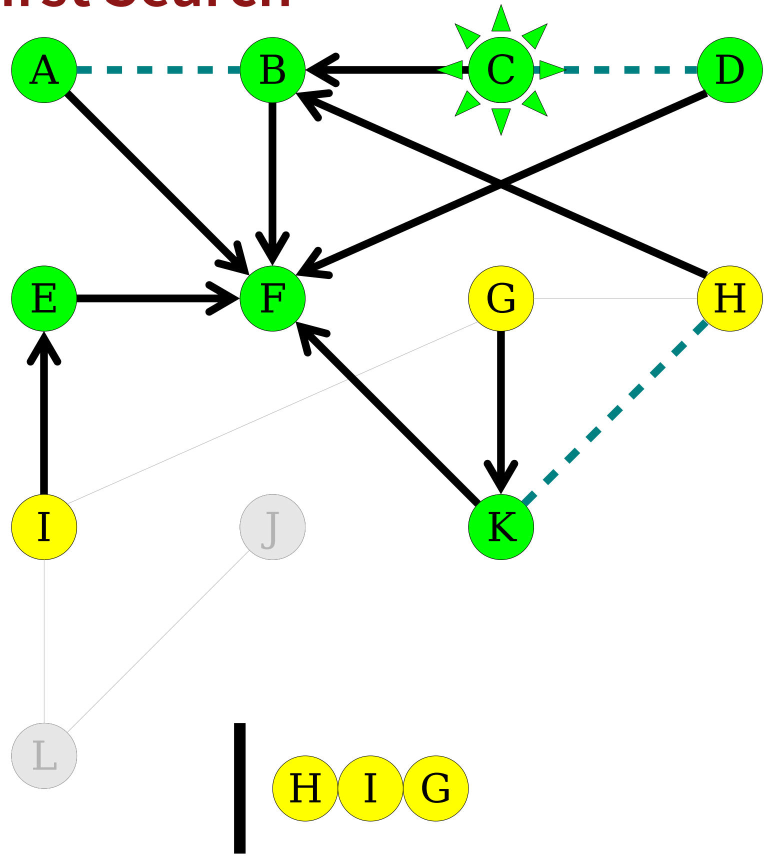
Breadth-First Search



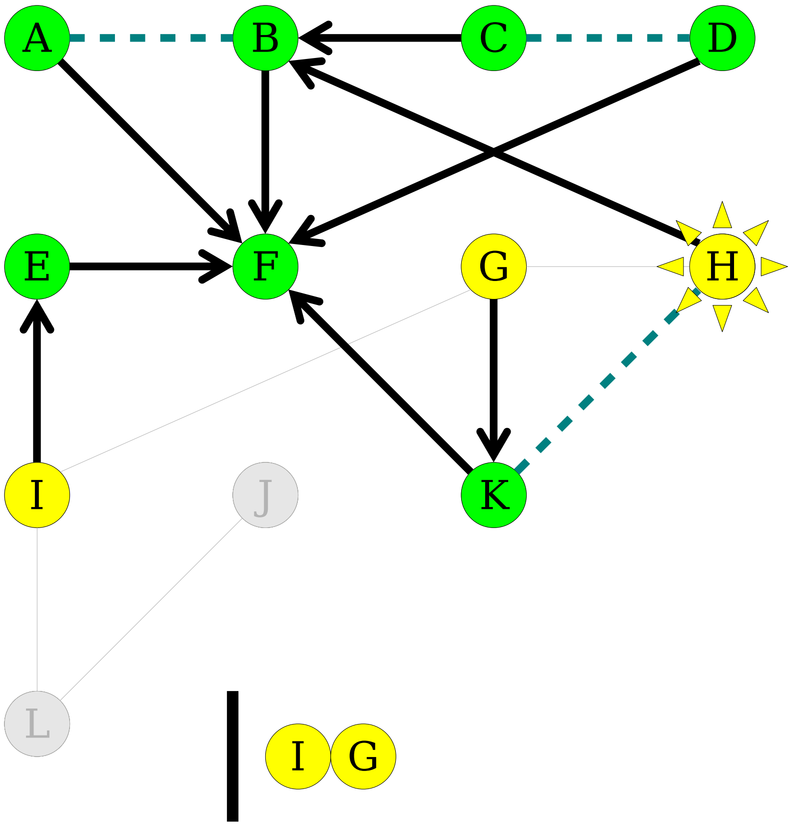
Breadth-First Search



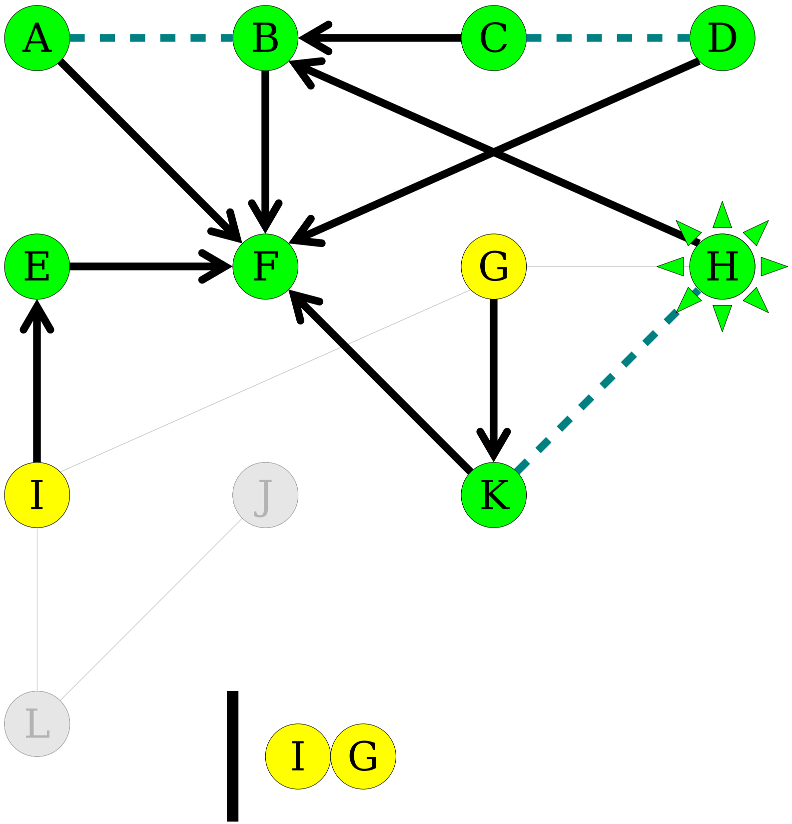
Breadth-First Search



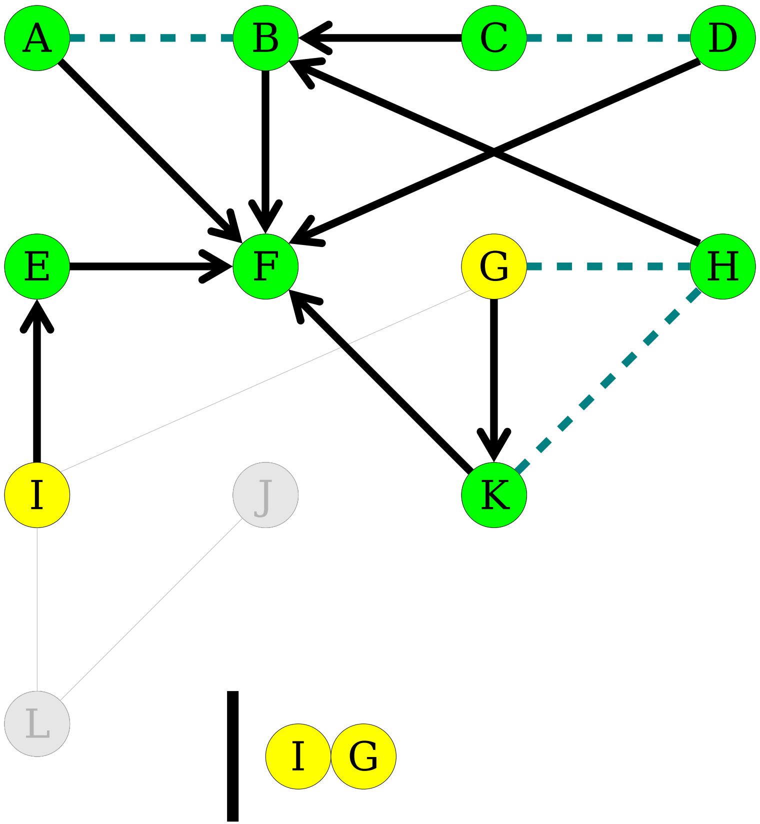
Breadth-First Search



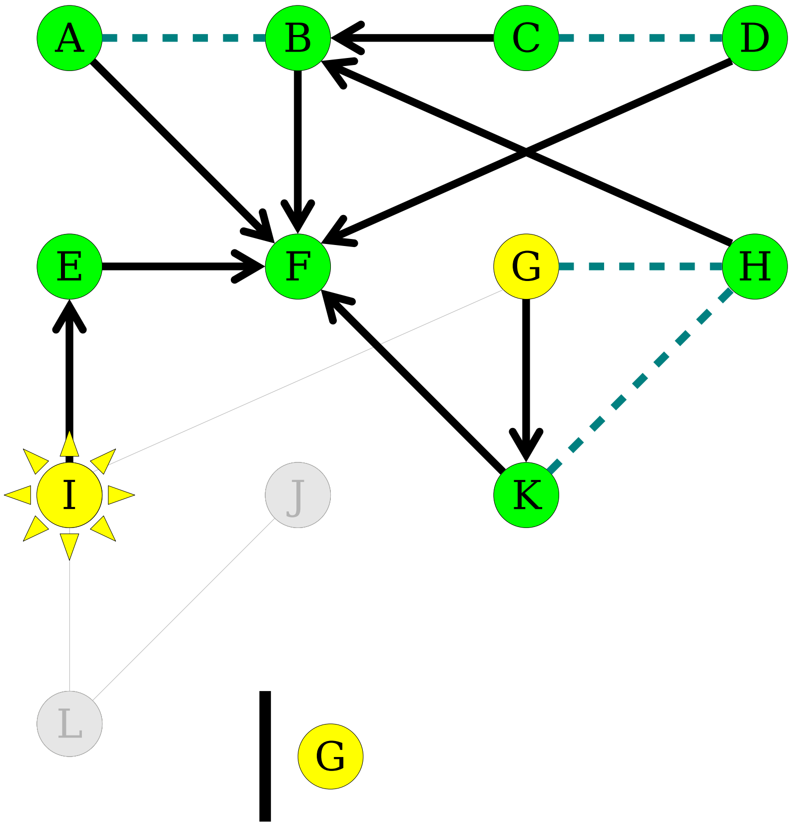
Breadth-First Search



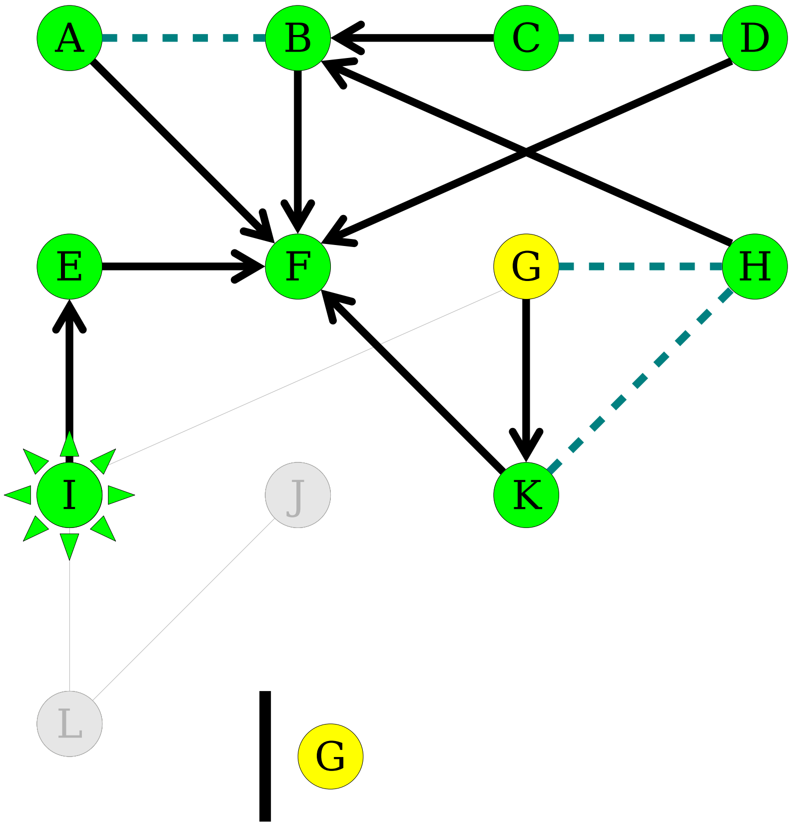
Breadth-First Search



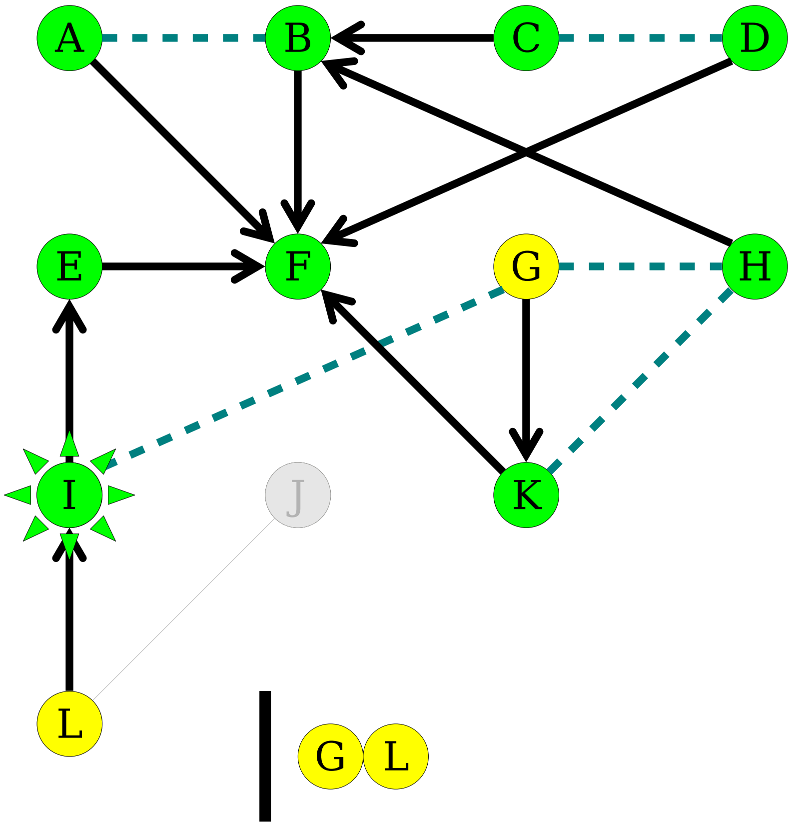
Breadth-First Search



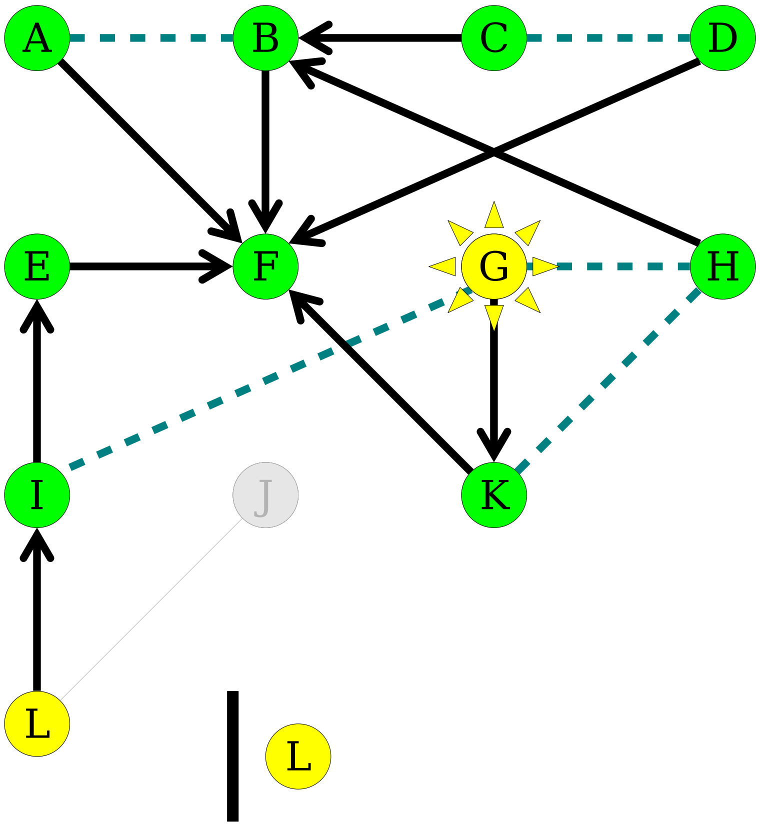
Breadth-First Search



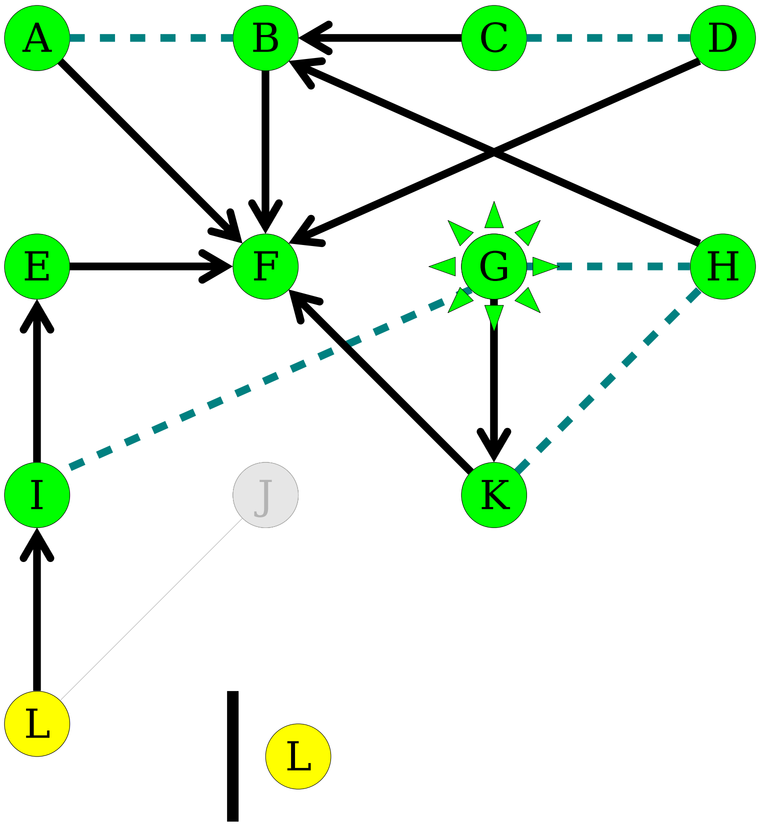
Breadth-First Search



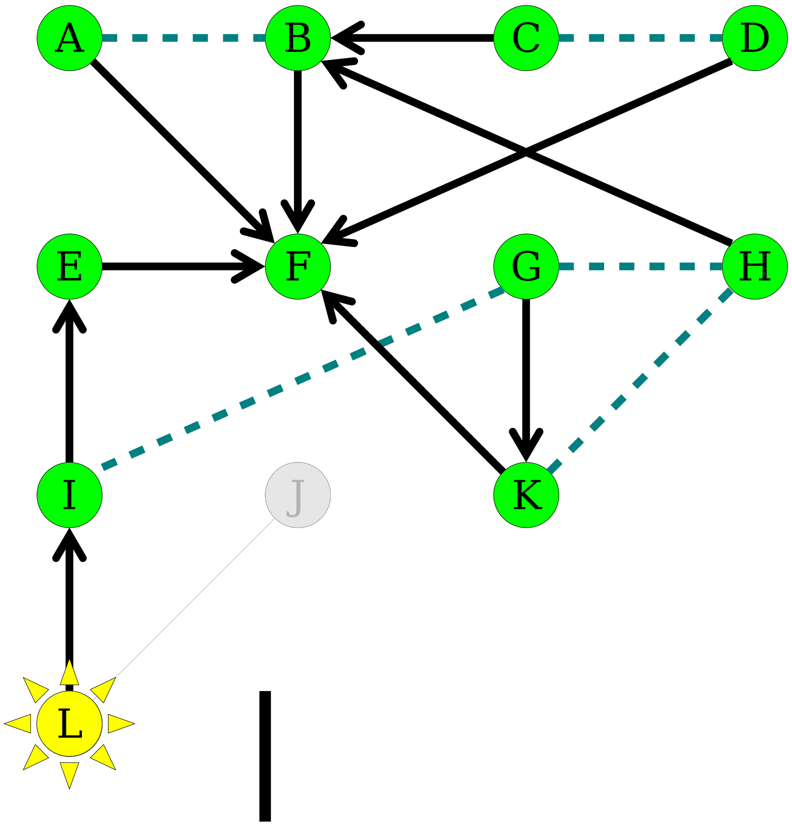
Breadth-First Search



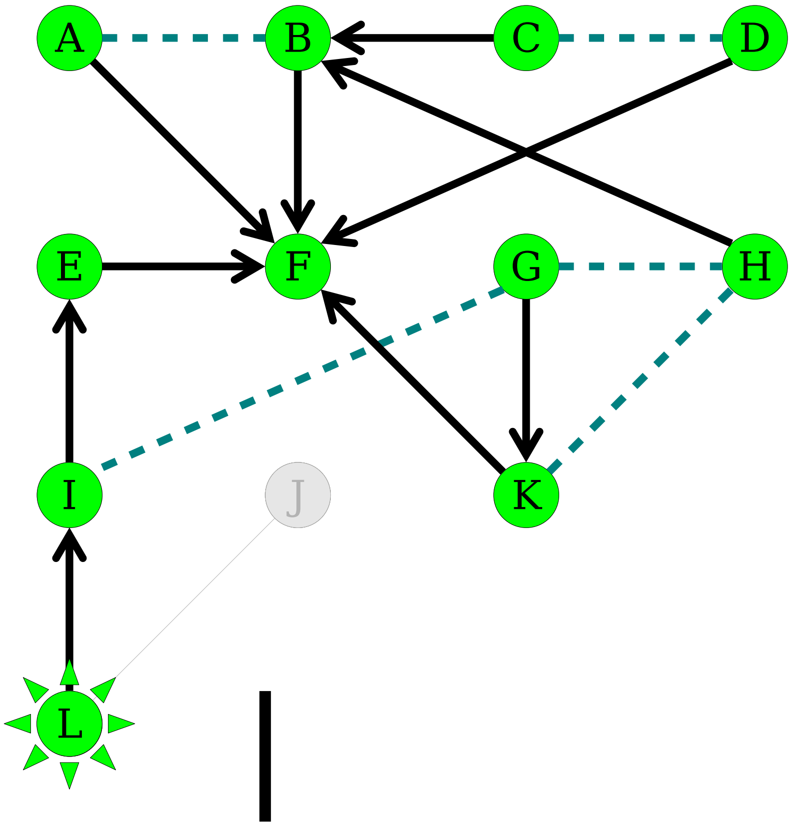
Breadth-First Search



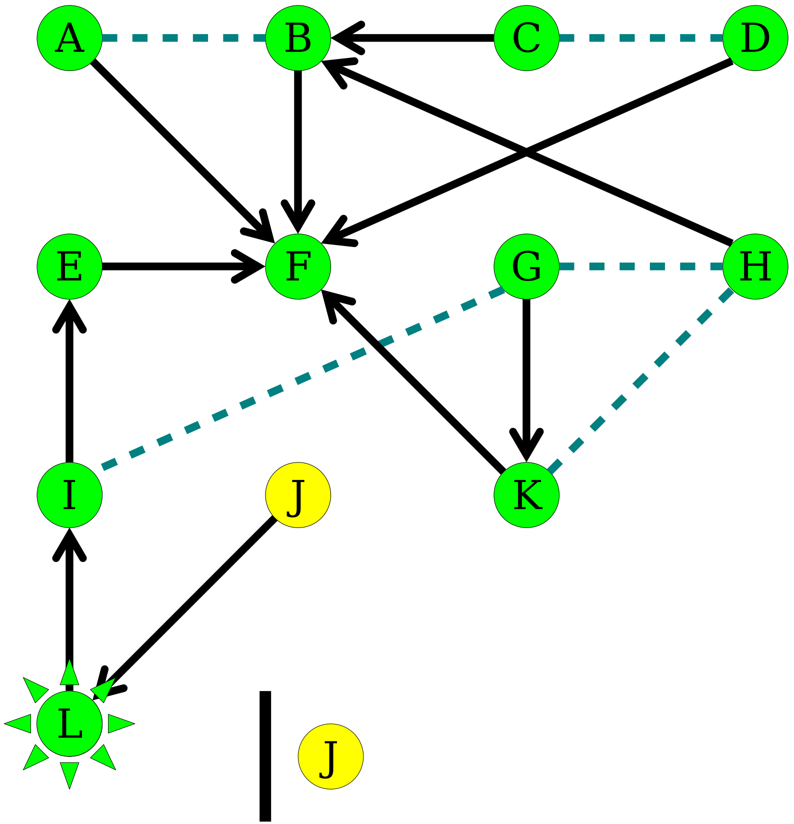
Breadth-First Search



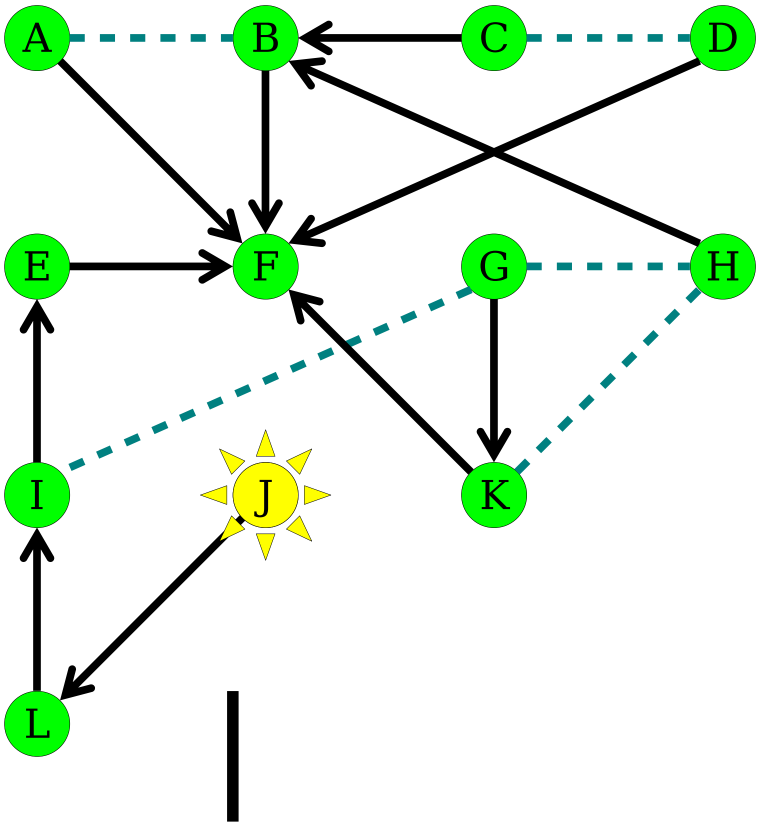
Breadth-First Search



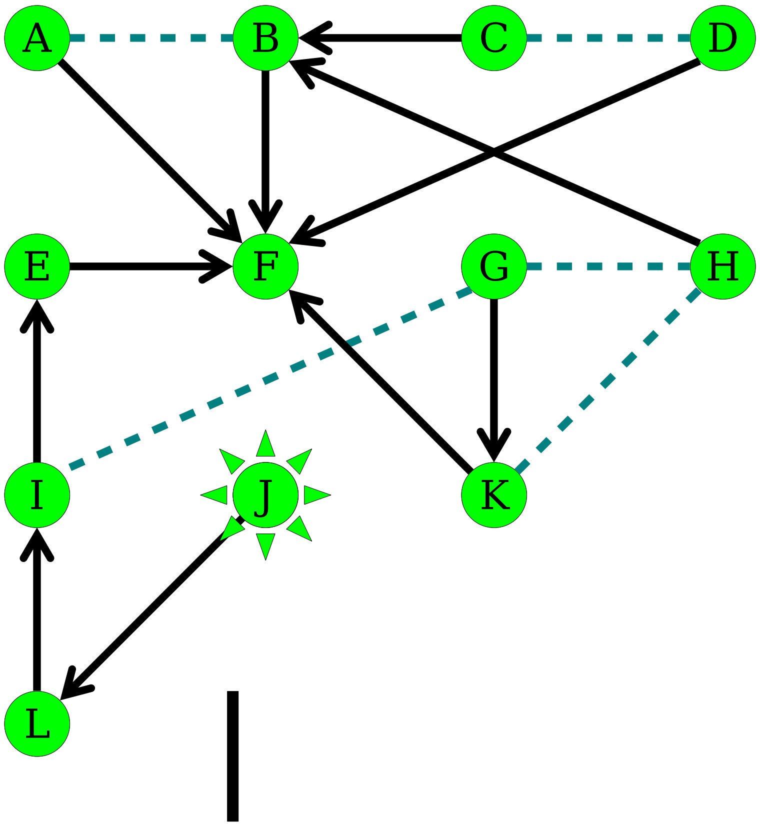
Breadth-First Search



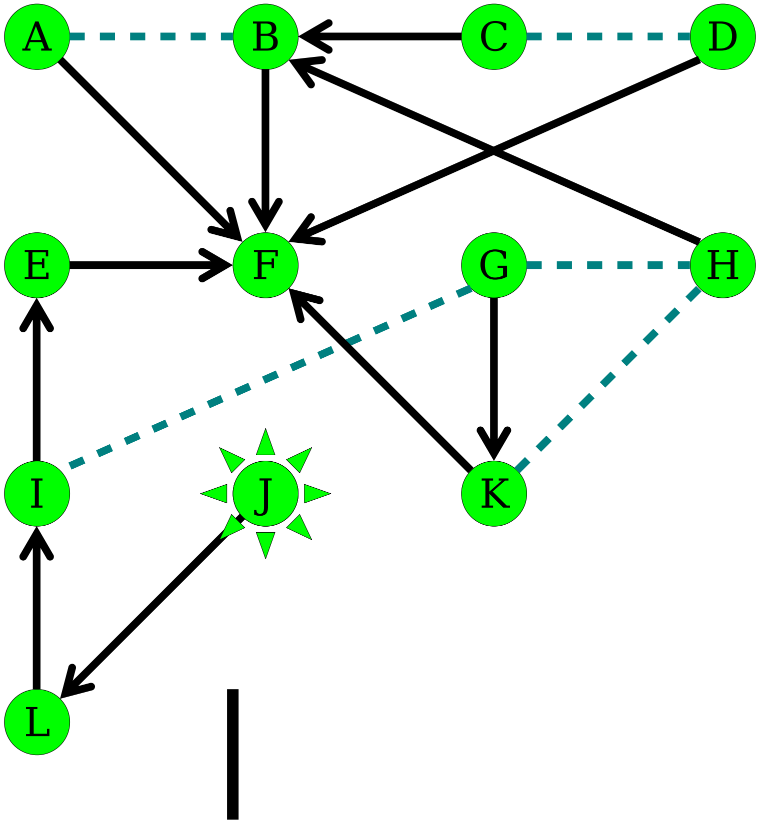
Breadth-First Search



Breadth-First Search



Breadth-First Search



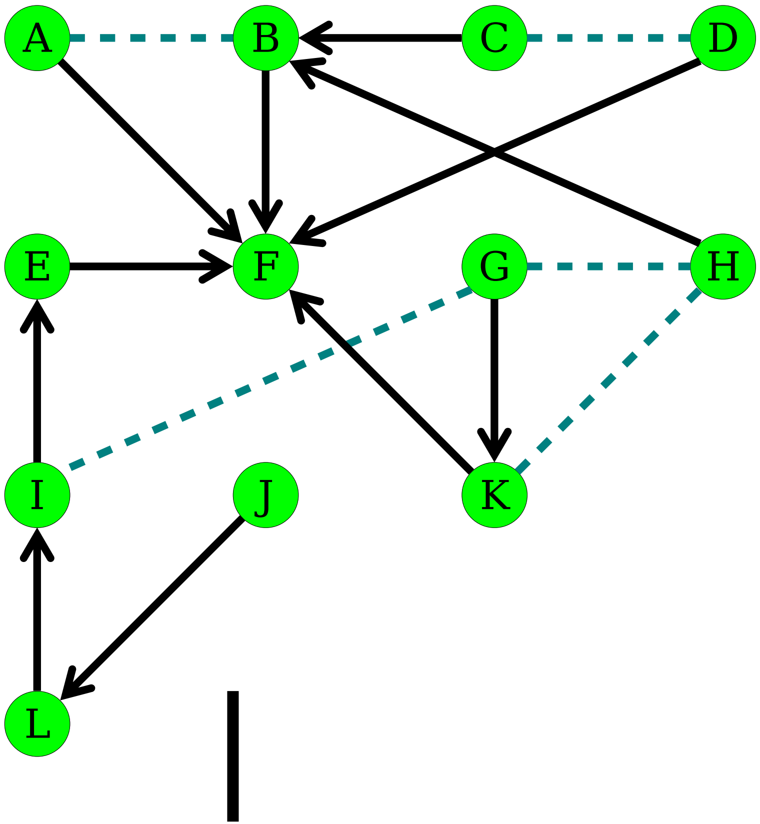
Done!

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 in every city in the western U.S.

Quick question about efficiency…

You’re all going to fly to Yosemite for a family reunion, and then everyone will rent a car and drive home, and you’ve been tasked with making custom Yosemite-to-home driving directions for **everyone**.

Quick question about efficiency…

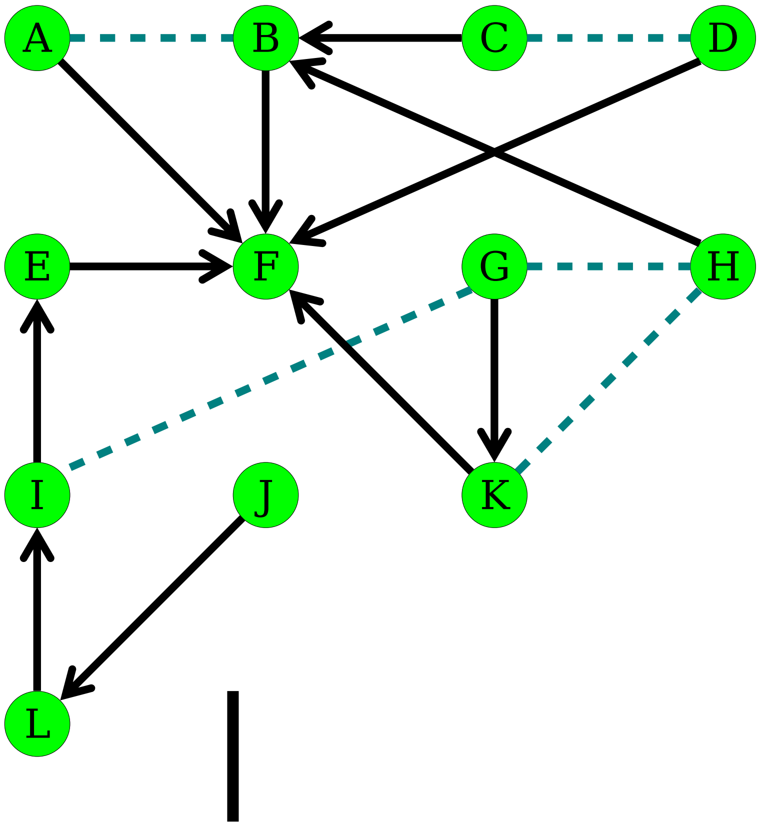
You calculated the shortest path for yourself to return home from the reunion (Yosemite to Palo Alto) and let's just say that it took time **X** = O((|E| + |V|)log|V|)

- With respect to the number of cities |V|, and the number of edges or road segments |E|

How long will it take you, in total, to calculate the shortest path for you <u>and</u> all of your relatives?

A. O(|V|*X) *(for X as defined above)*
B. O(|E|*|V|* X) *(for X as defined above)*
C. X *(for X as defined above)*
D. Other/none/more

Breadth-First Search



THINGS TO NOTICE:

(4) If we go until the queue is empty, we now have the answer to the question "What is the shortest path to you from F?" for ***every single node in the graph!!***

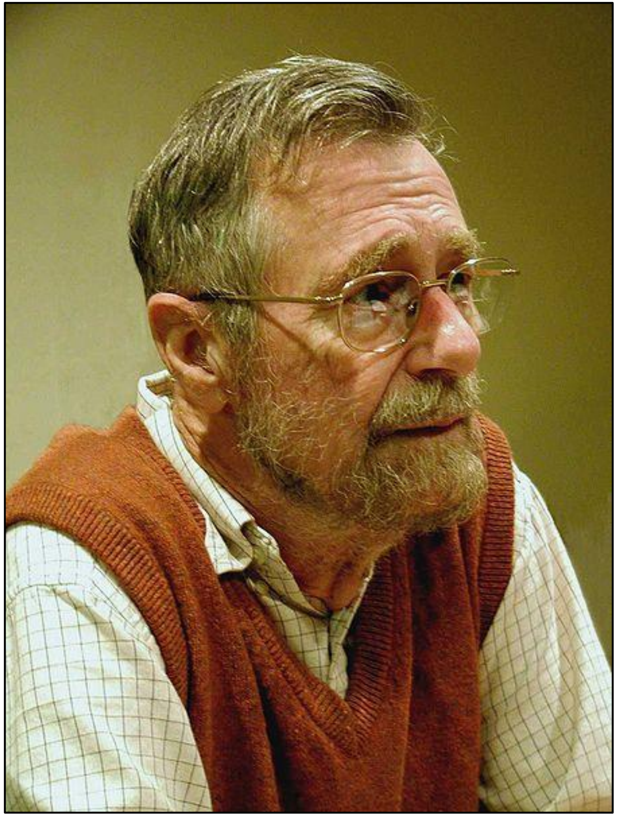
Dijkstra's Shortest Paths

(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.

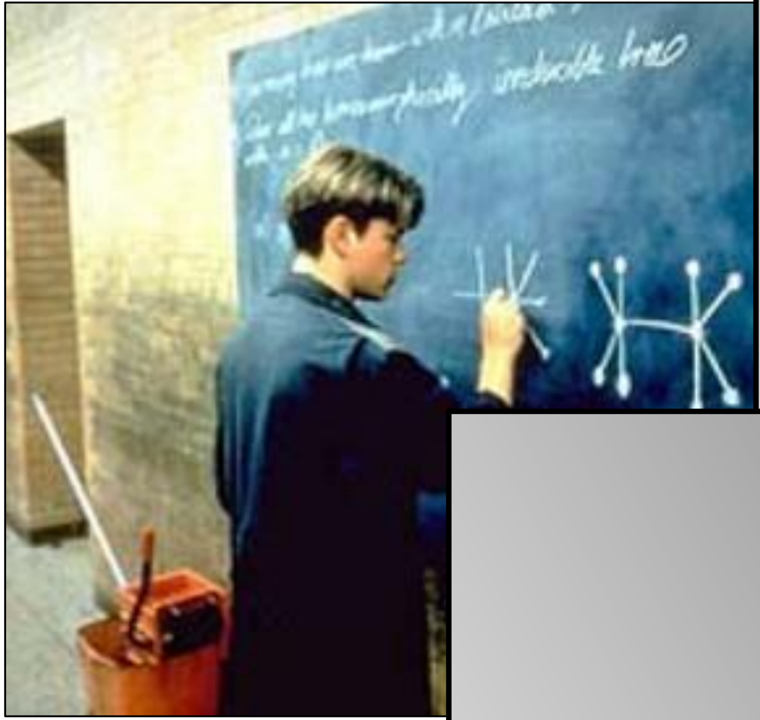
The usual way in which we plan today for tomorrow is in yesterday's vocabulary. We do so, because we try to get away with the concepts we are familiar with and that have acquired their meanings

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***.

The Good Will Hunting Problem

“Draw all the homeomorphically irreducible trees with n=10.”



“Draw all the homeomorphically irreducible trees with n=10.”

In this case **“trees”** simply means **connected, undirected graphs with no cycles**

“with n = 10” (i.e., has **10 nodes**)

“homeomorphically irreducible”

- **No nodes of degree 2 (nodes adjacent to exactly two other nodes) are allowed in your solutions**
  - For this problem, nodes of degree 2 are useless in terms of tree structure—they just act as a blip on an edge—and are therefore banned
- Have to be actually different
  - Ignore superficial changes in rotation or angles of drawing