Programming Abstractions

CS106X

Cynthia Lee

Upcoming Topics

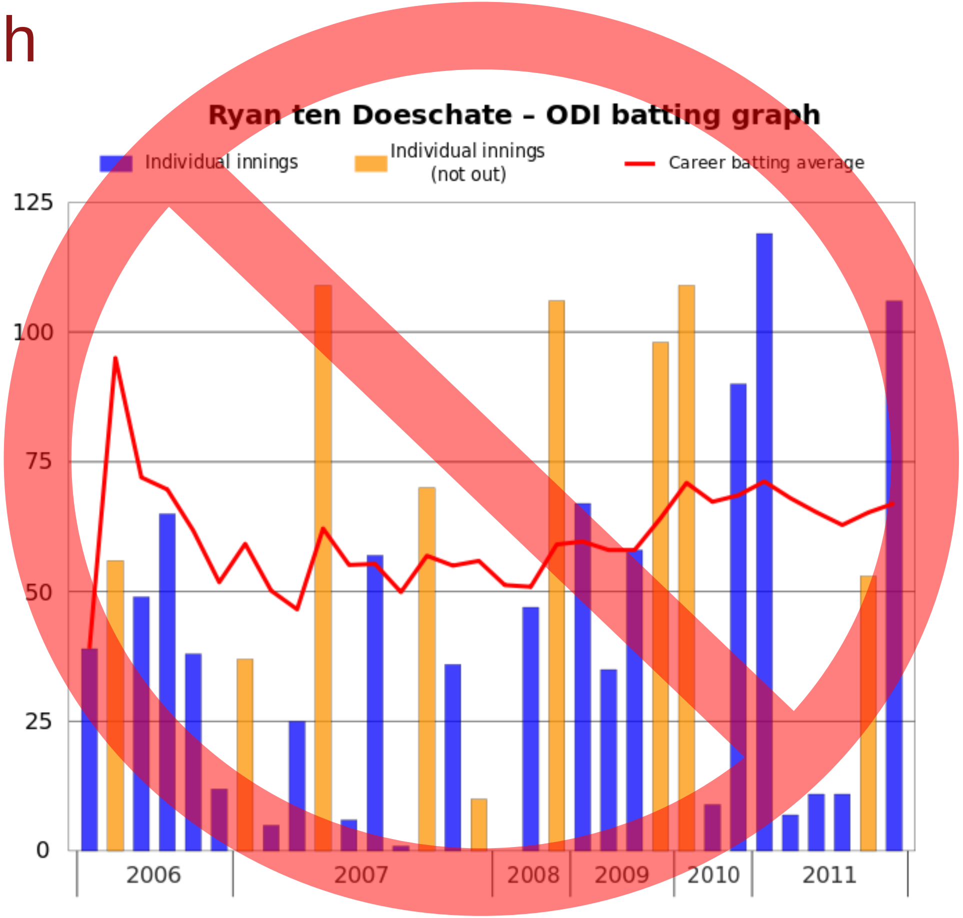
Graphs!

1. **Basics**
   - What are they? How do we represent them?
2. **Theorems**
   - What are some things we can prove about graphs?
3. **Breadth-first search** on a graph
   - Spoiler: just a very, very small change to tree version
4. **Dijkstra's shortest paths algorithm**
   - Spoiler: just a very, very small change to BFS
5. **A* shortest pathsalgorithm**
   - Spoiler: just a very, very small change to Dijkstra's
6. **Minimum Spanning Tree**
   - Kruskal's algorithm

Graphs

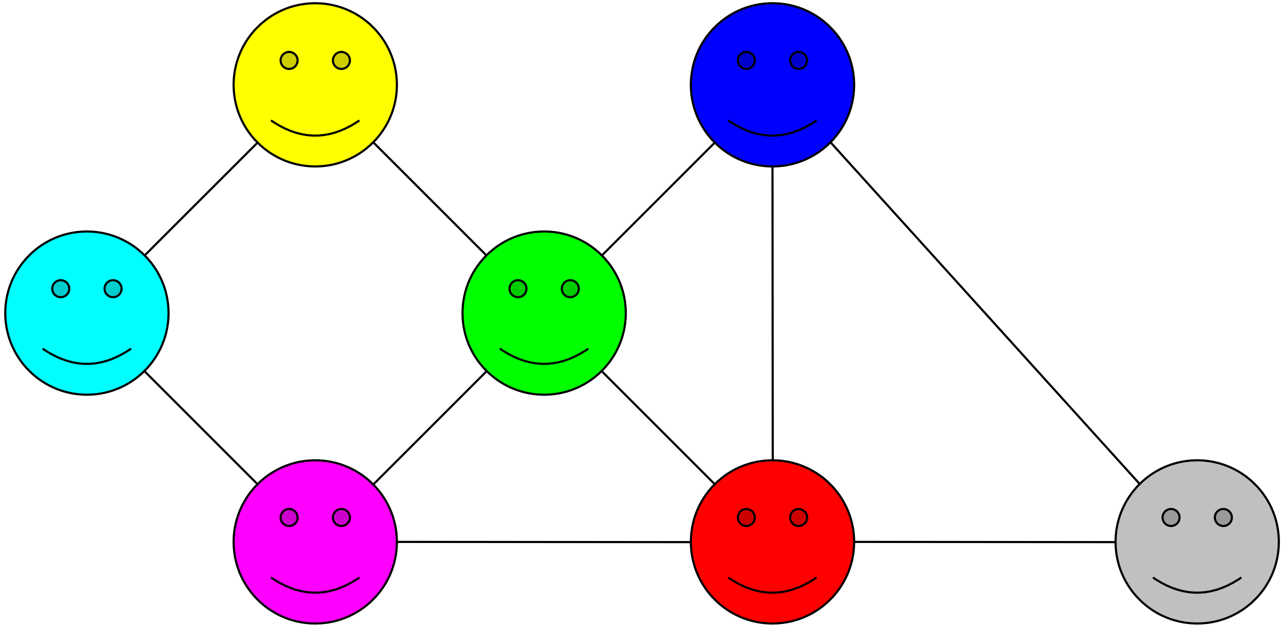
What are graphs? What are they good for?

Graph

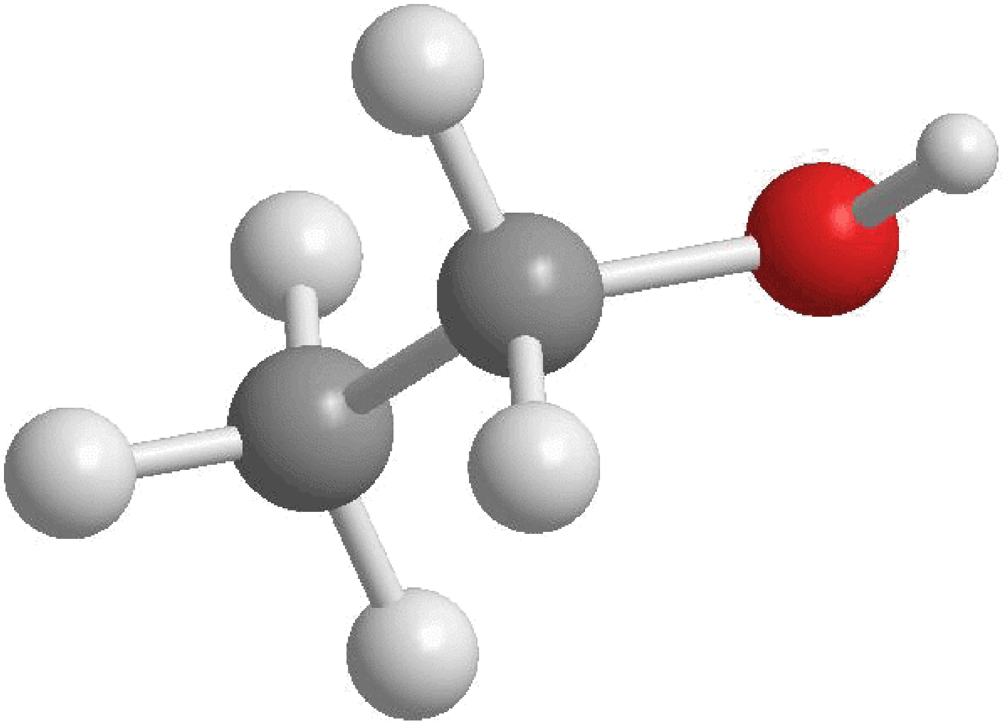


Ryan ten Doeschate - ODI batting graph
Individual innings
Individual innings (not out)
Career batting average
125
100
75
50
25
0
2006
2007
2008
2009
2010
2011

A Social Network

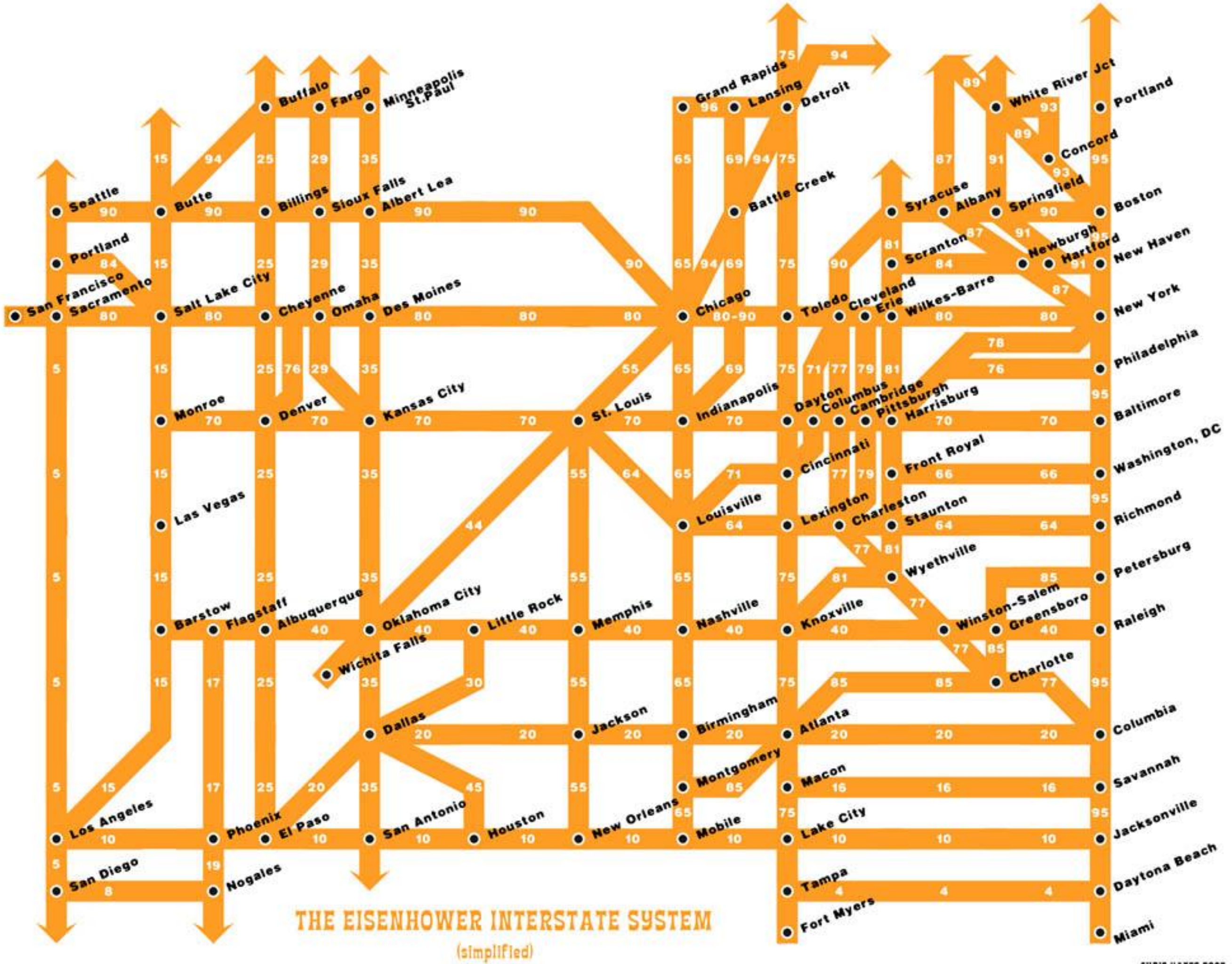


Chemical Bonds



Slide by Keith Schwarz

http://4.bp.blogspot.com/-xCtBJ8lKHqA/Tjm0BONWBRI/AAAAAAAAAK4/-mHrbAUOHHg/s1600/Ethanol2.gif

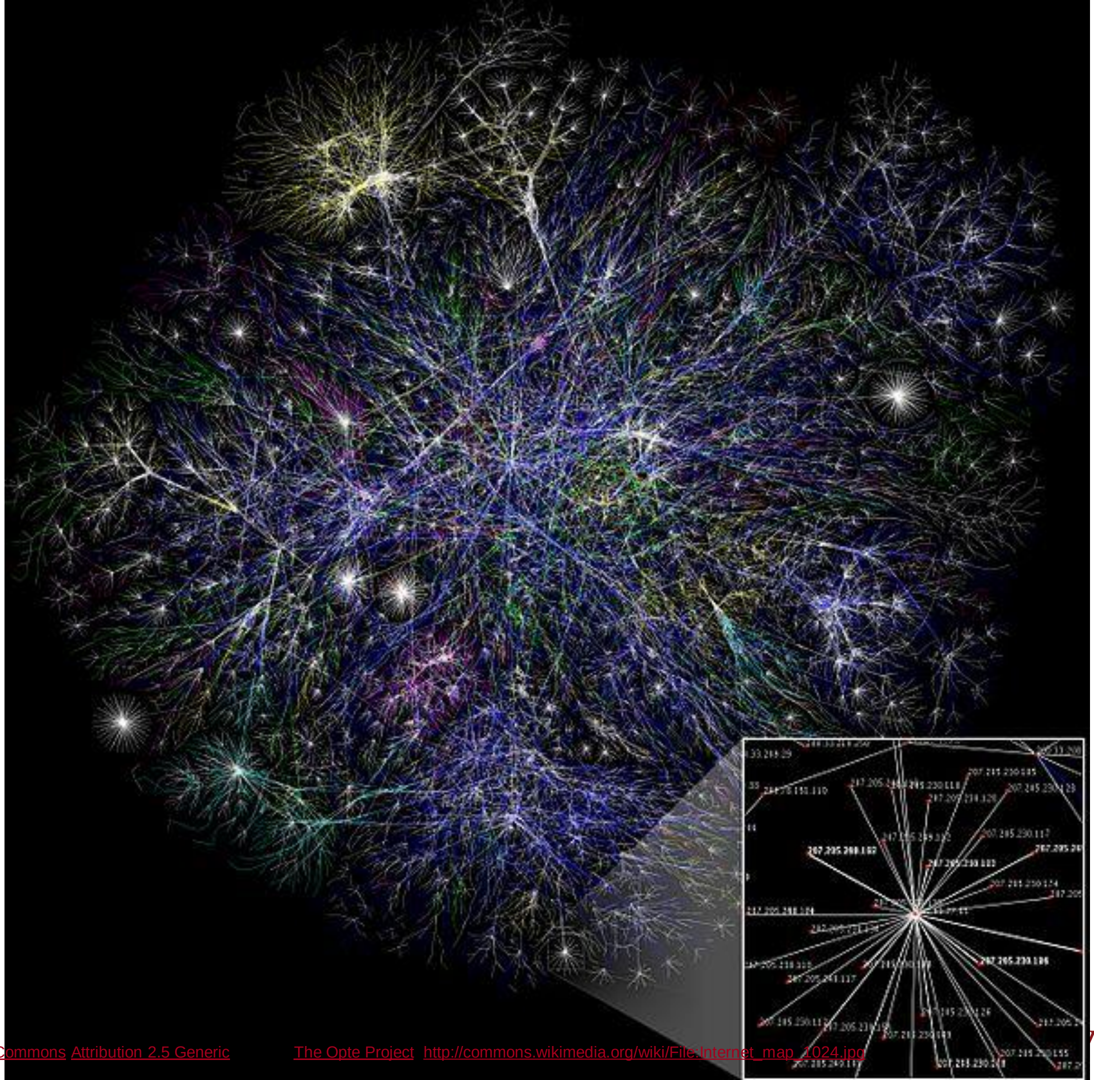


http://strangemaps.files.wordpress.com/2007/02/fullinterstatemap-web.jpg

Slide by Keith Schwarz

University

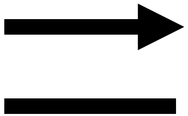
Internet



A graph is a mathematical structure for representing relationships

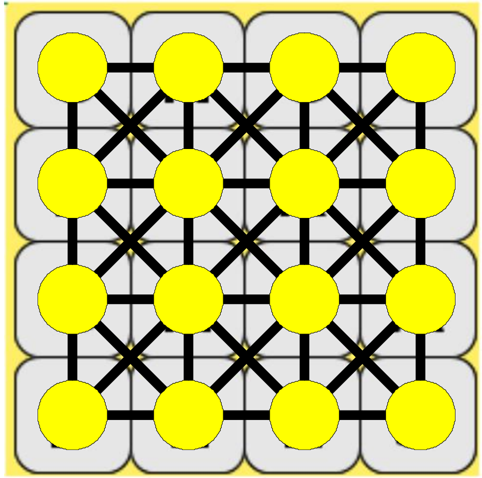
Consists of:

- A set V of **vertices** (or *nodes*)
  - Often have an associated label
- A set E of **edges** (or *arcs*)
  - Consist of two endpoint 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 notation |V| and |E|



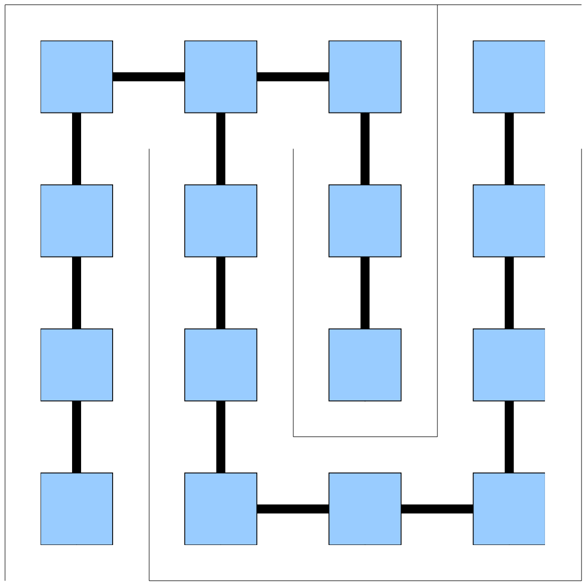
Boggle as a graph

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?

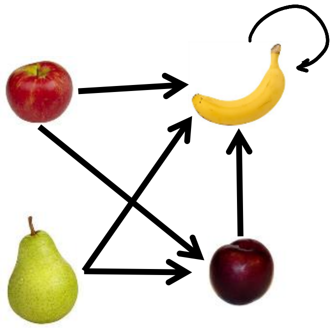


Graphs

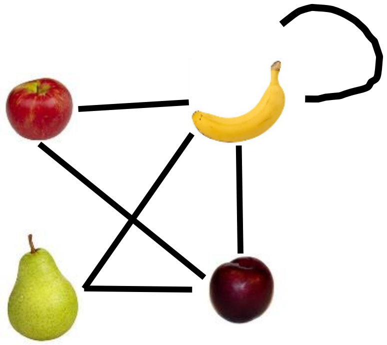
How do we represent graphs in code?

Graph terminology

This is a DIRECTED graph



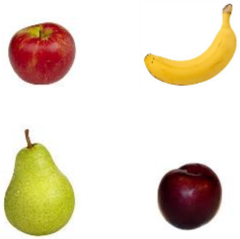
This is an UNDIRECTED graph



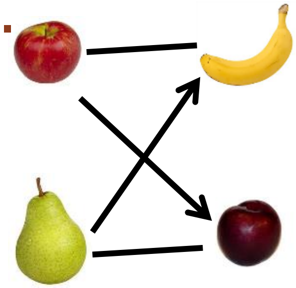
Graph terminology

Which of the following is a correct graph?

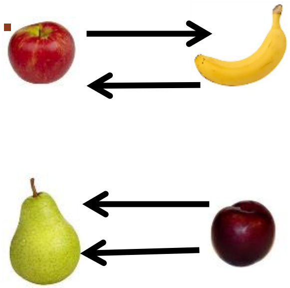
A.



B.



C.

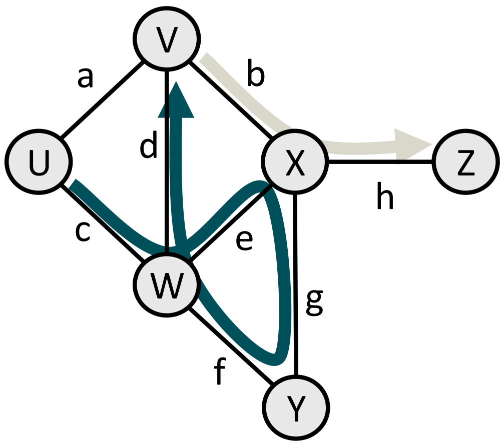


D. None of the above/other/more than one of the above

Paths

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

- can be represented as vertices visited, or edges taken
- example, one path from *V* to *Z*: {b, h} or {V, X, Z}
- What are two paths from U to Y?



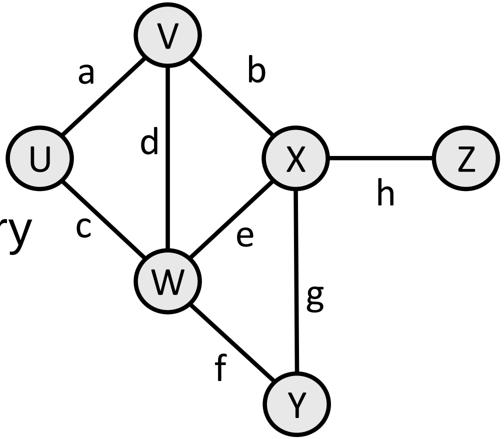
path length: Number of vertices or edges contained in the path.

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

- example: V and X

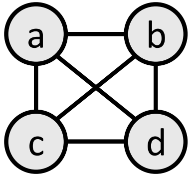
Reachability, connectedness

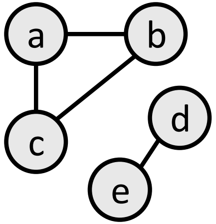
reachable: Vertex *a* is *reachable* from *b* if a path exists from *a* to *b*.



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

complete: If every vertex has a direct edge to every other.





Loops and cycles

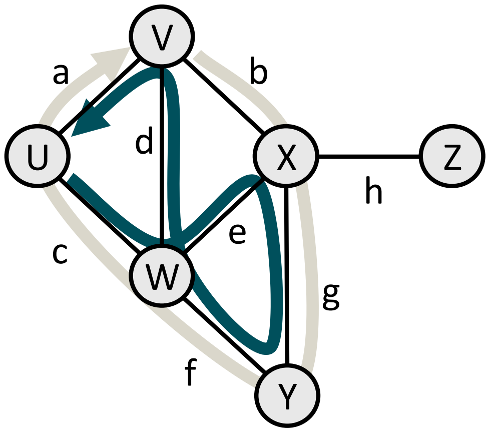
cycle: A path that begins and ends at the same node.

- example: {V, X, Y, W, U, V}.
- example: {U, W, V, U}.

- **acyclic graph**: One that does not contain any cycles.

loop: An edge directly from a node to itself.

- Many graphs don't allow loops.

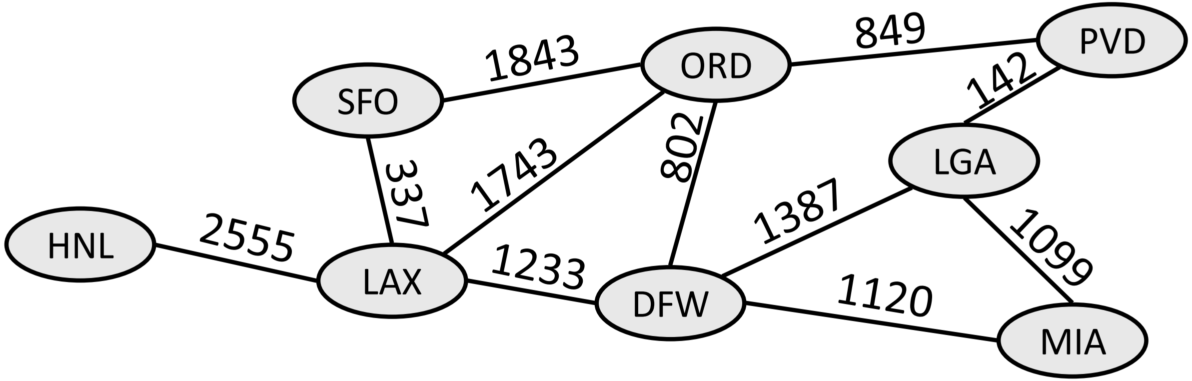


Weighted graphs

weight: Cost associated with a given edge.

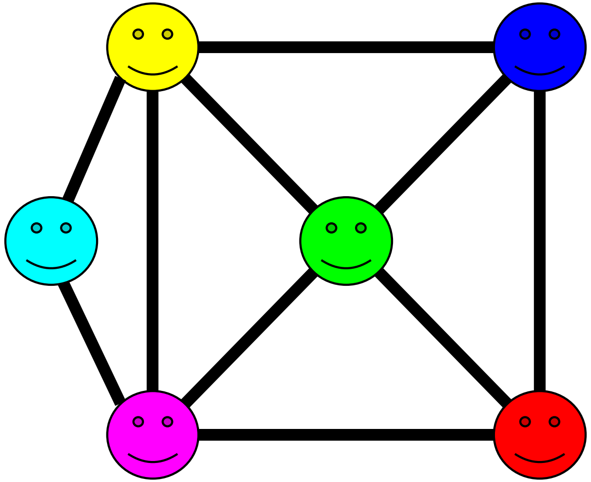
- Some graphs have weighted edges, and some are unweighted.
- Edges in an unweighted graph can be thought of as having equal weight (e.g. all 0, or all 1, etc.)
- Most graphs do not allow negative weights.

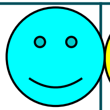
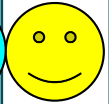
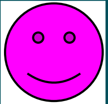
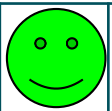
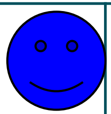
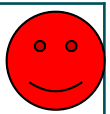
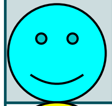
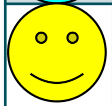
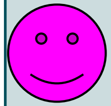
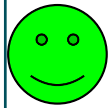
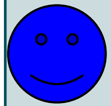
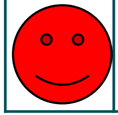
example: graph of airline flights, weighted by miles between cities:



Representing Graphs: Adjacency Matrix

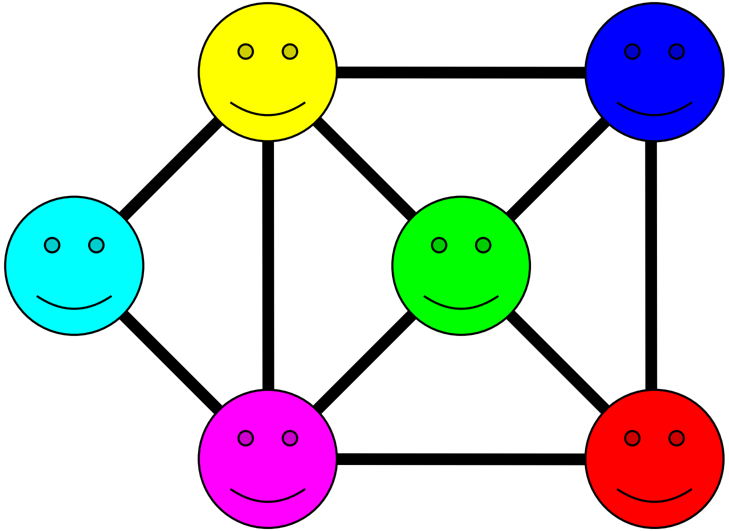
We can represent a graph as a
Grid<bool> (unweighted)
or
Grid<int> (weighted)



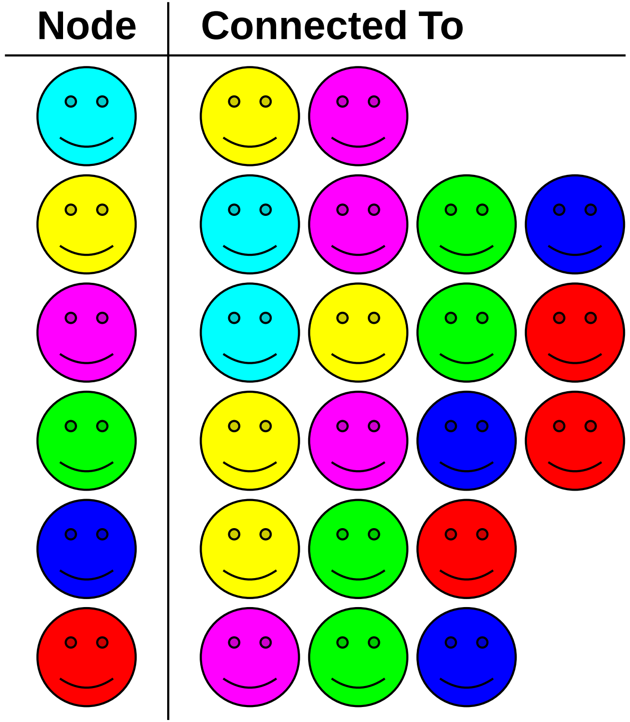
						
	0	1	1	0	0	0
	1	0	1	1	1	0
	1	1	0	1	0	1
	0	1	1	0	1	1
	0	1	0	1	0	1
	0	0	1	1	1	0

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*>>`



Common ways of representing graphs

Adjacency list:

- Map<Node*, Set<Node*>>

Adjacency matrix:

- Grid<bool> unweighted
- Grid<int> weighted

How many of the following are true?

- Adjacency **list** can be used for **directed** graphs
- Adjacency **list** can be used for **undirected** graphs
- Adjacency **matrix** can be used for **directed** graphs
- Adjacency **matrix** can be used for **undirected** graphs

(A) 0 (B) 1 (C) 2 (D) 3 (E) 4

Graphs

Theorems about graphs

Graphs lend themselves to fun theorems and proofs of said theorems!

Any graph with 6 vertices contains either a triangle (3 vertices with all pairs having an edge) or an empty triangle (3 vertices no two pairs having an edge)

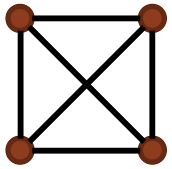
Eulerian graphs

Let G be an **undirected graph**

A graph is **Eulerian** if it can
drawn without lifting the pen
and without repeating edges

Is this graph Eulerian?

A. Yes
B. No



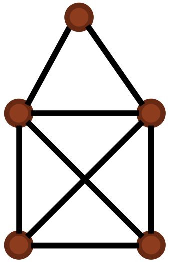
Eulerian graphs

Let G be an **undirected graph**

A graph is **Eulerian** if it can
drawn without lifting the pen
and without repeating edges

What about this graph

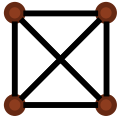
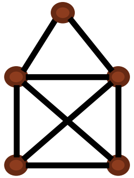
A. Yes
B. No



Our second graph theorem

Definition: Degree of a vertex: number of edges adjacent to it

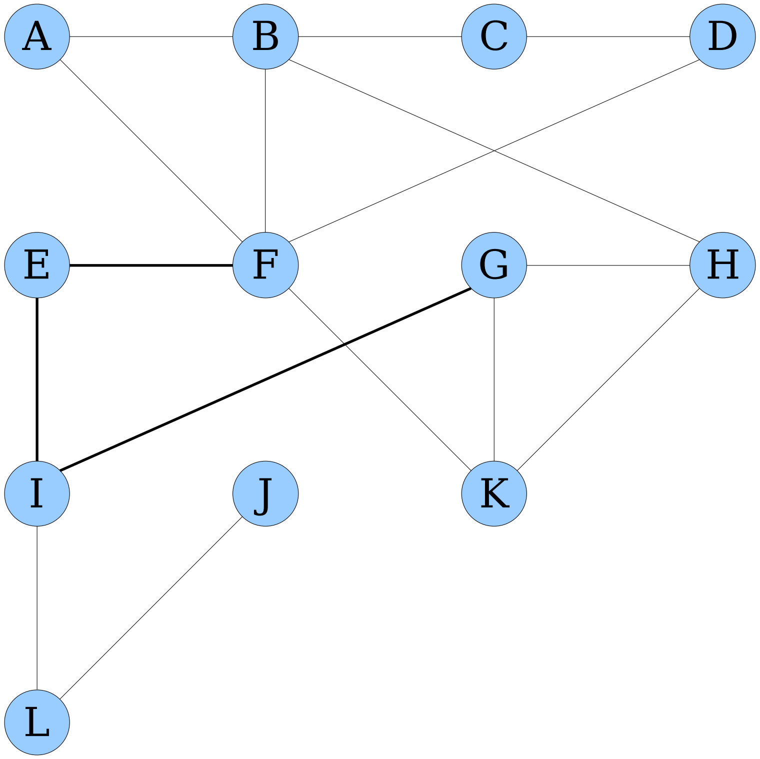
Euler's theorem: a connected graph is Eulerian iff the number of vertices with odd degrees is either 0 or 2 (eg all vertices or all but two have even degrees)

Does it work for  and  ?

Breadth-First Search

Graph algorithms

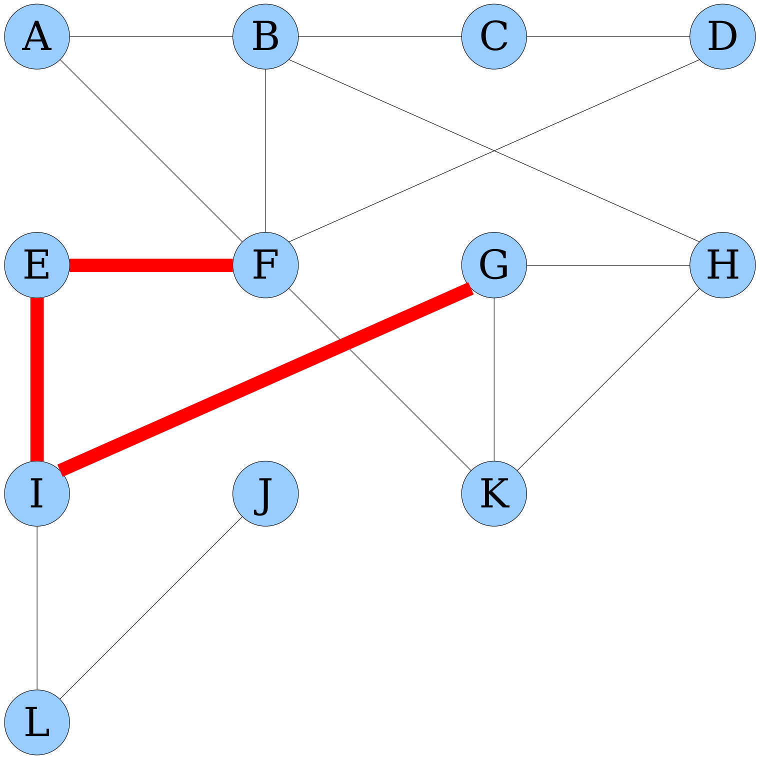
Breadth-First Search



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

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

Breadth-First Search

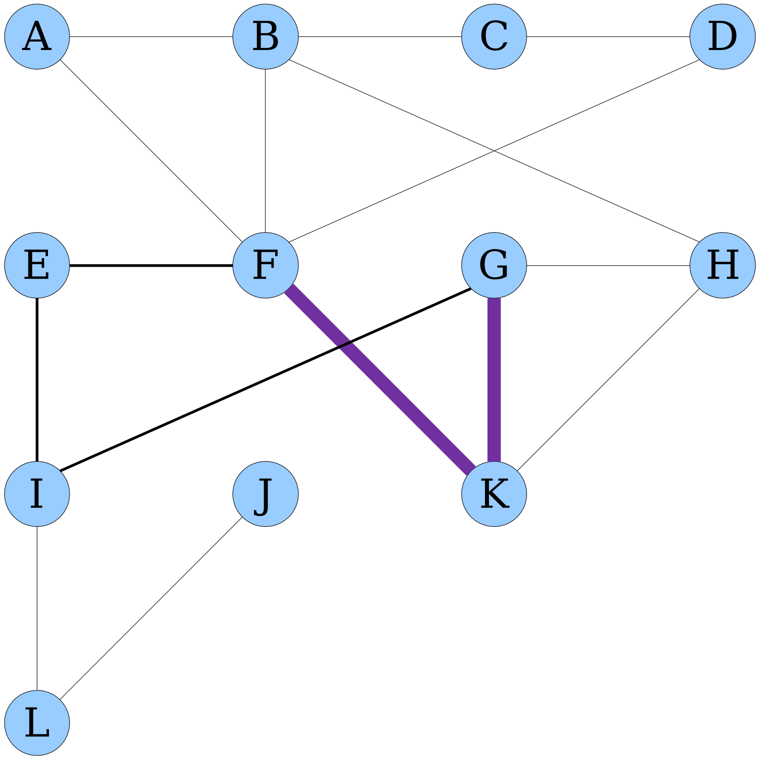


BFS is useful for finding the <u>shortest path</u> between two nodes.

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

Way 1: F->E->I->G
3 edges

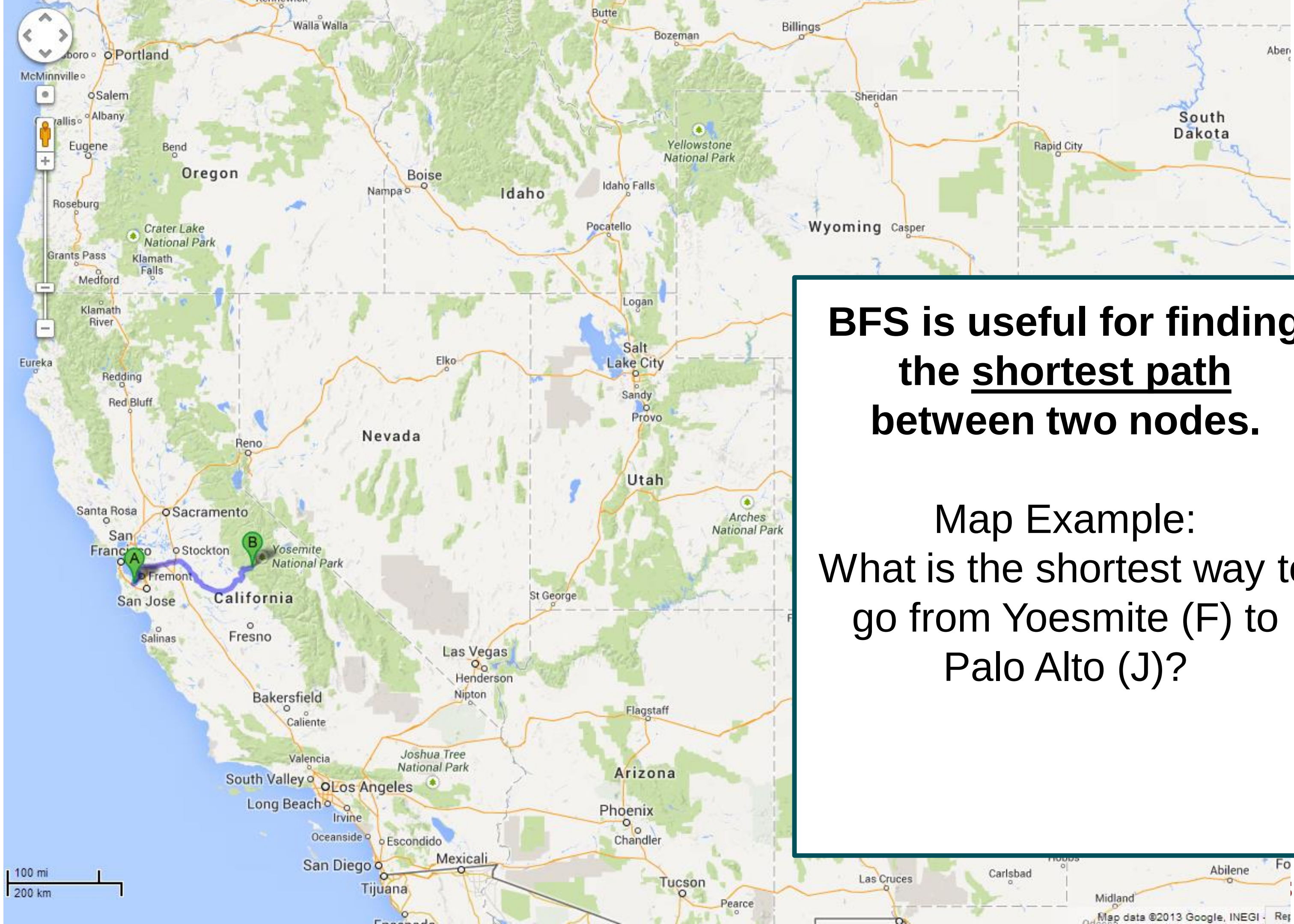
Breadth-First Search



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

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

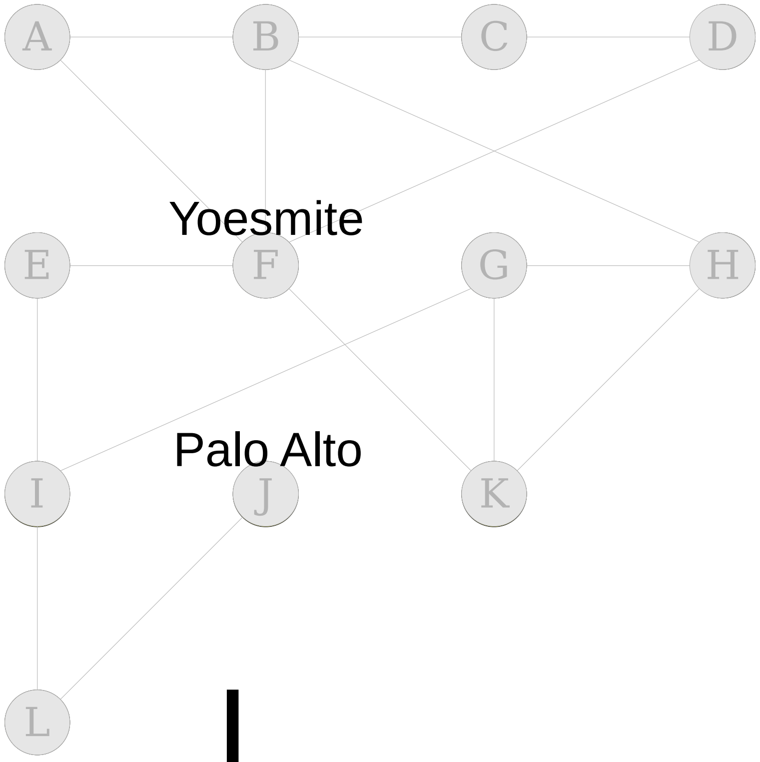
Way 2: F->K->G
2 edges



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

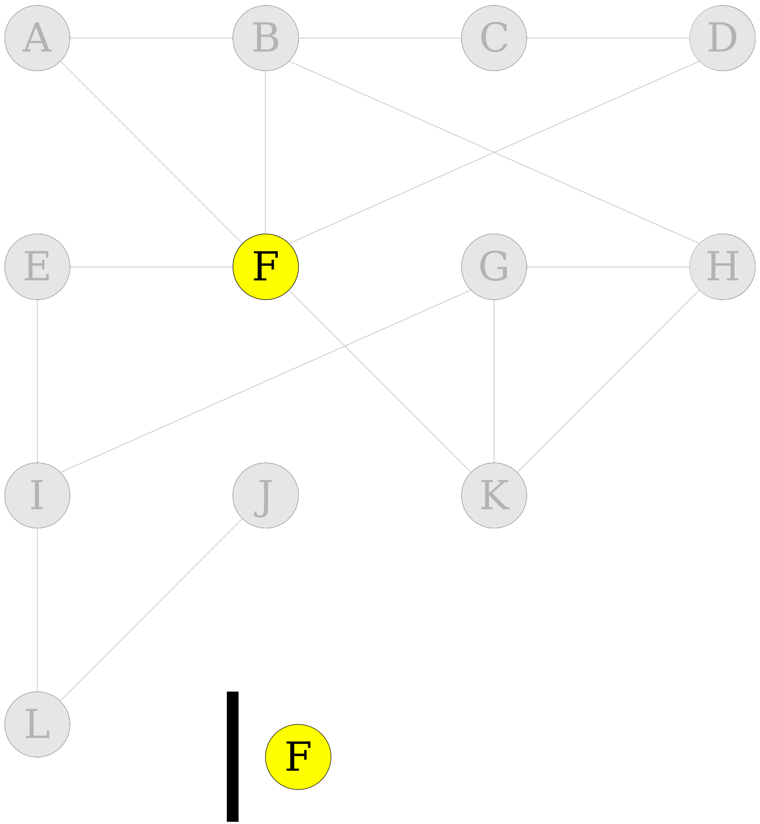
Map Example:
What is the shortest way to go from Yoesmite (F) to Palo Alto (J)?

Breadth-First Search



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

Breadth-First Search

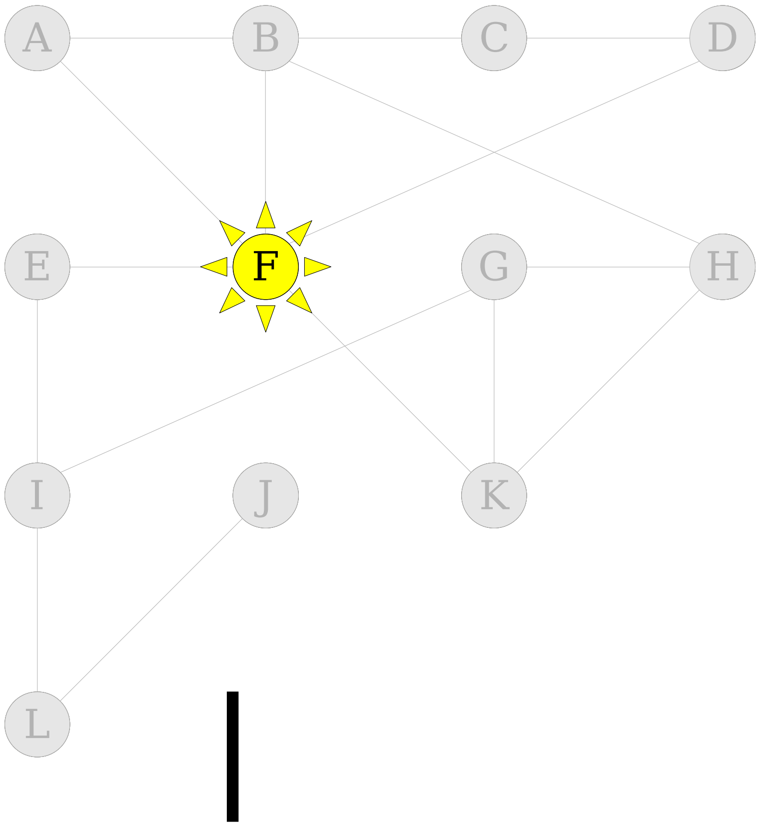


TO START (2):

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

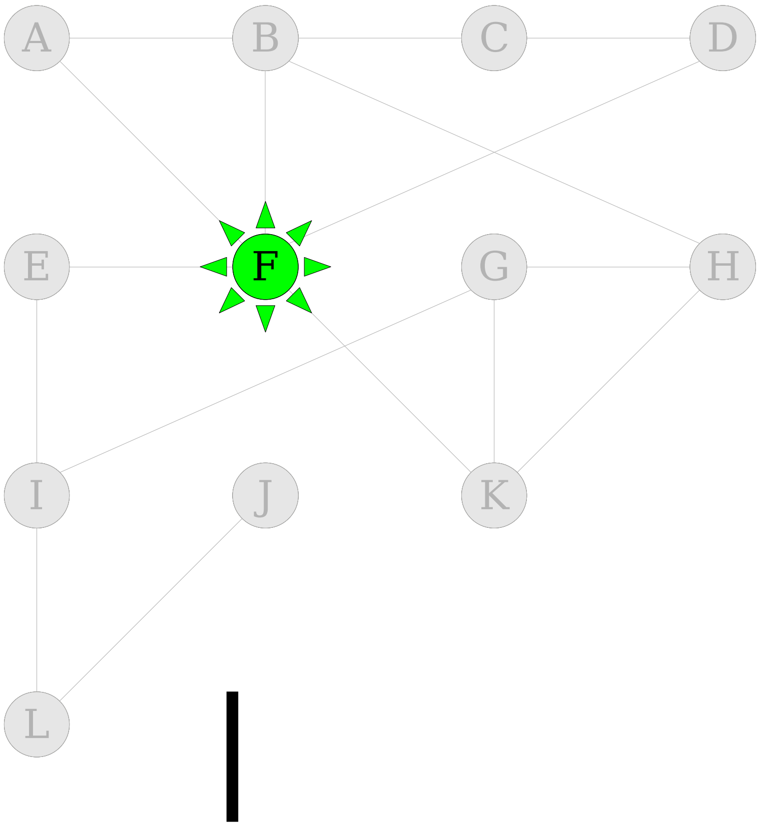
(2) Note that anytime we enqueue a node, we mark it YELLOW

Breadth-First Search

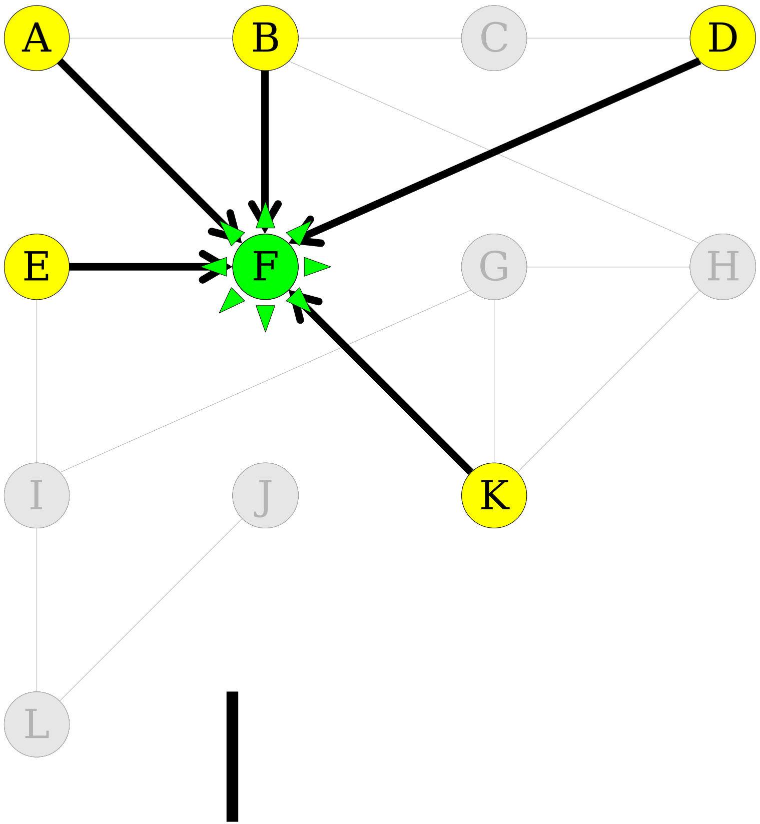


LOOP PROCEDURE:
(1)Dequeue a node
(2)Mark current node GREEN
(3)Set current node’s GREY neighbors’ parent pointers to current node, then enqueue them (remember: set them YELLOW)

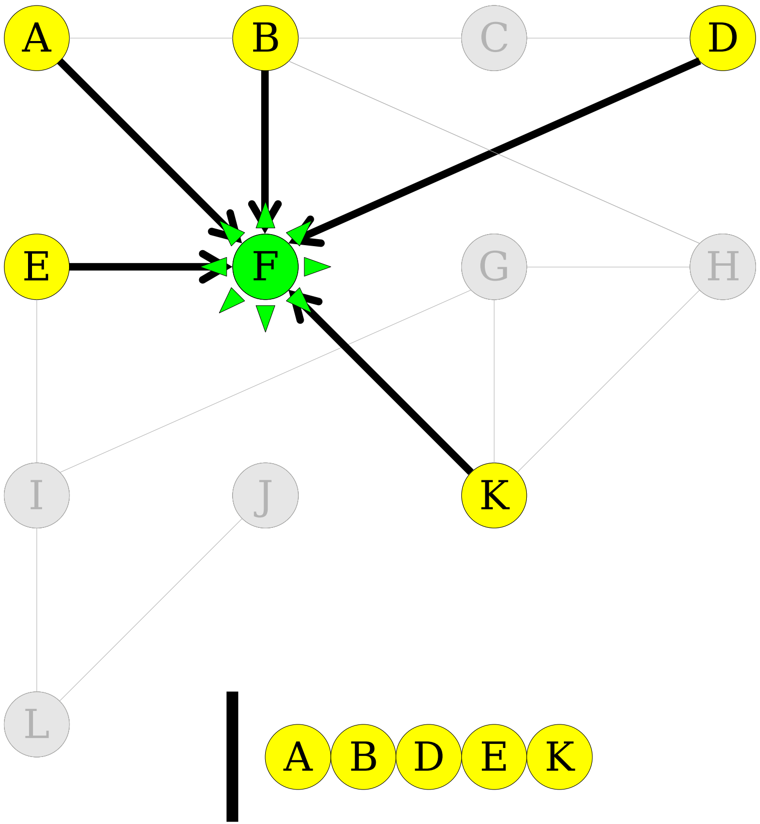
Breadth-First Search



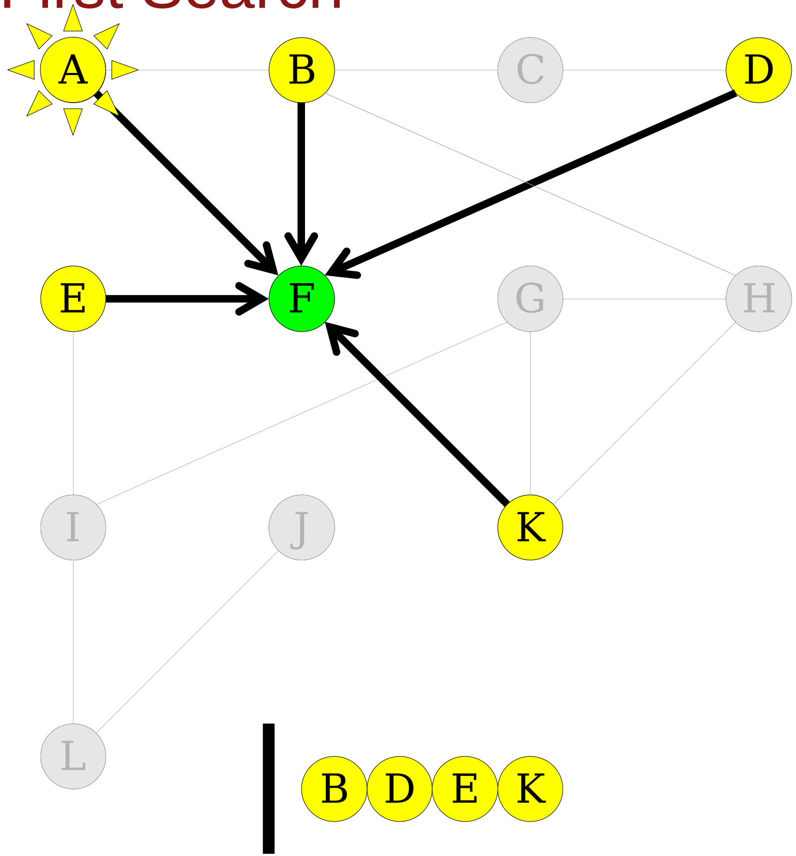
Breadth-First Search



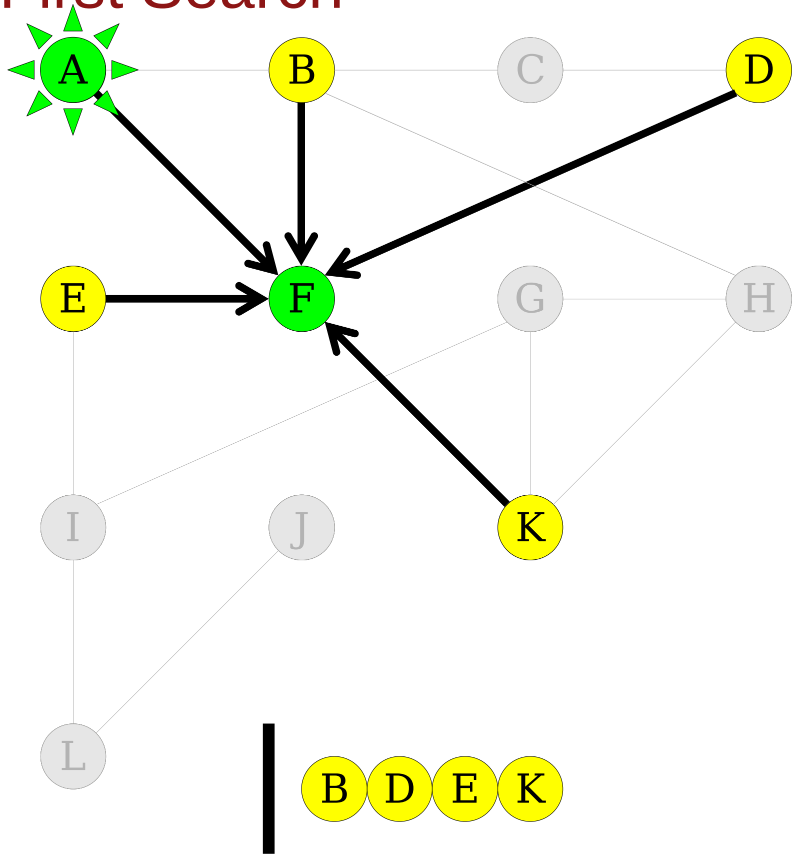
Breadth-First Search



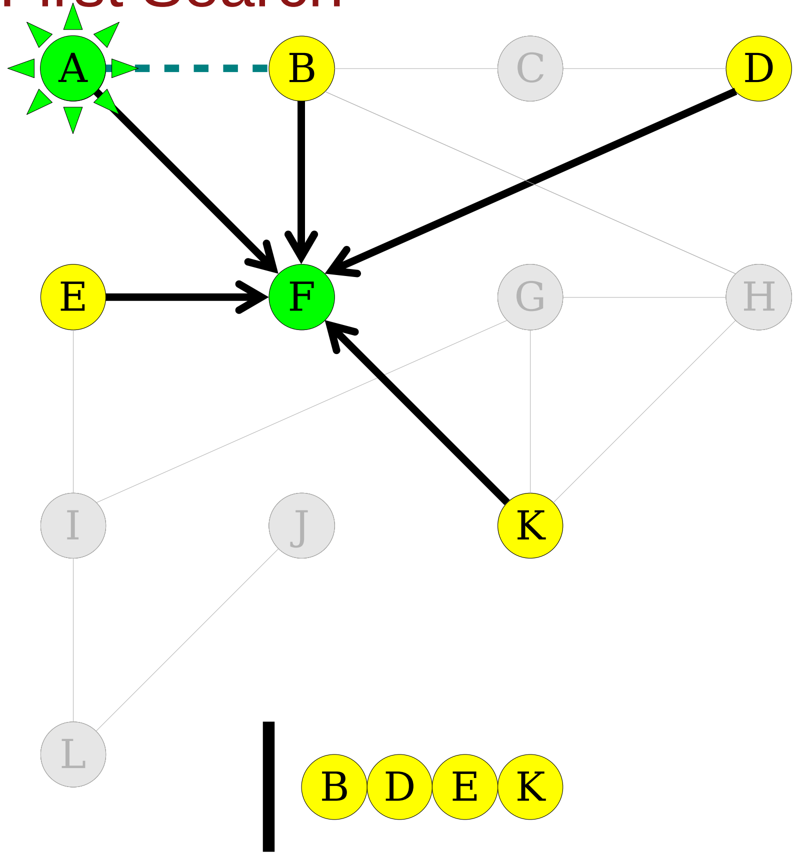
Breadth-First Search



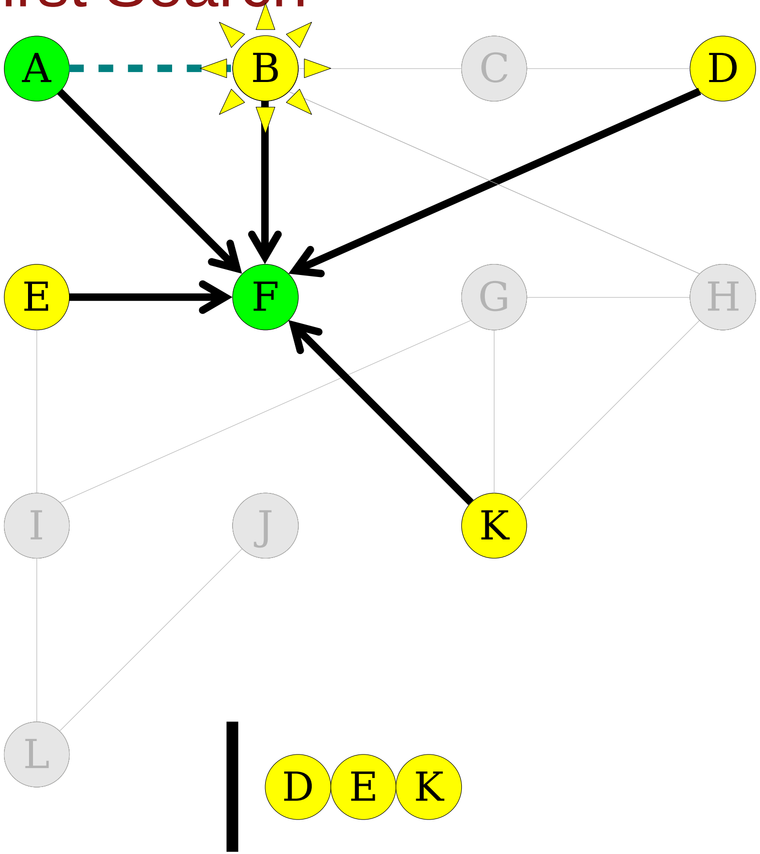
Breadth-First Search



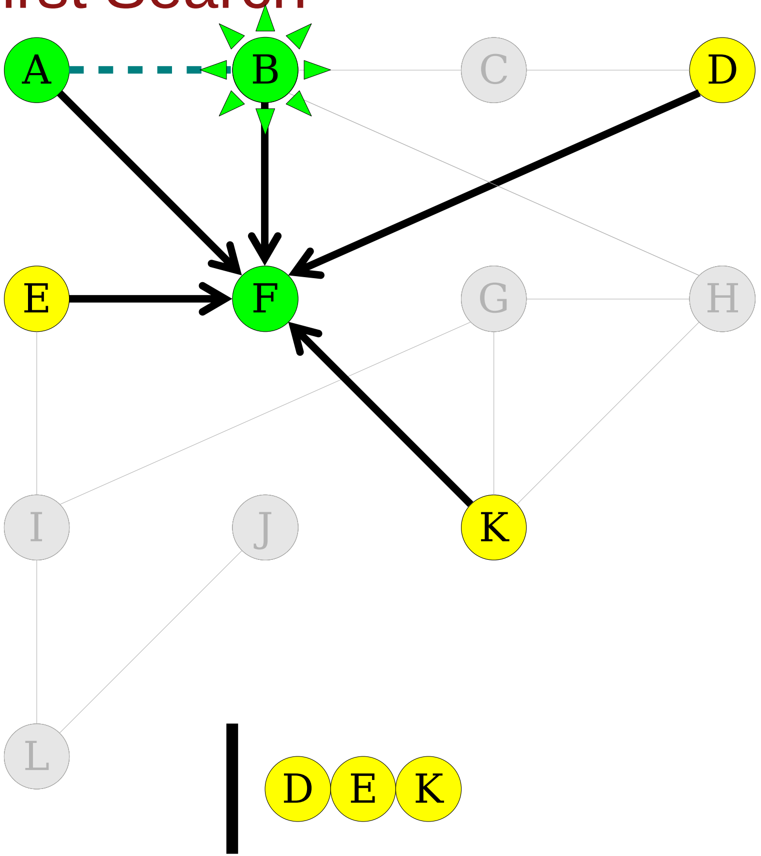
Breadth-First Search



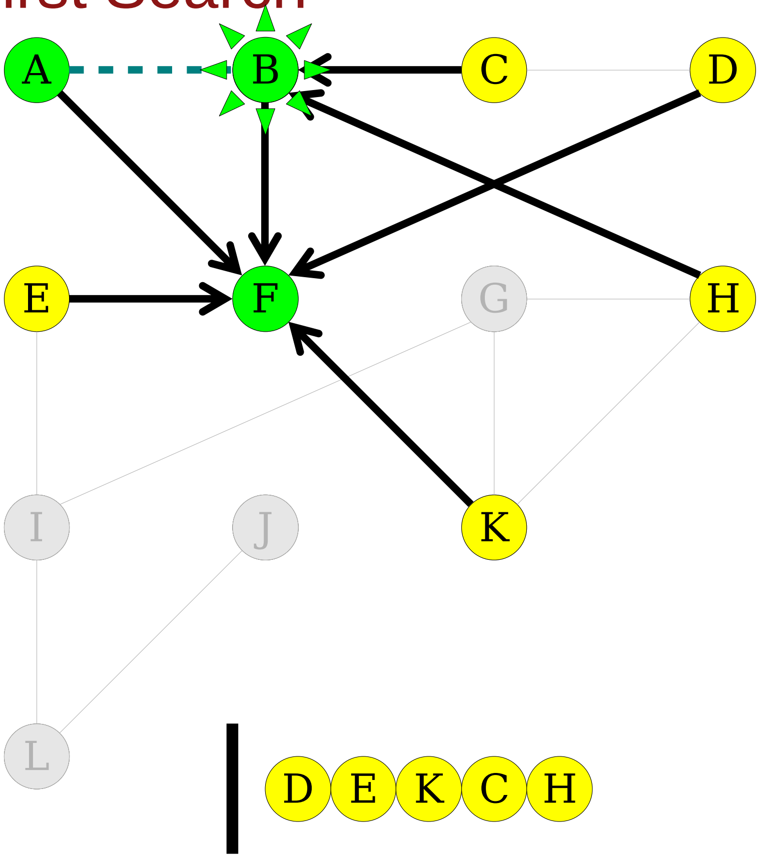
Breadth-First Search



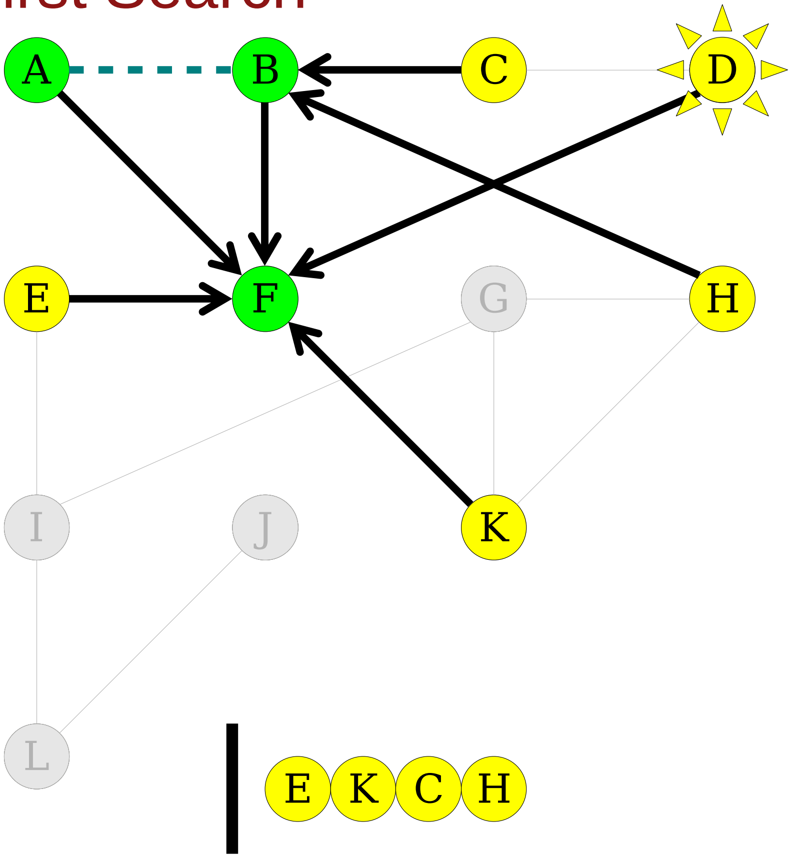
Breadth-First Search



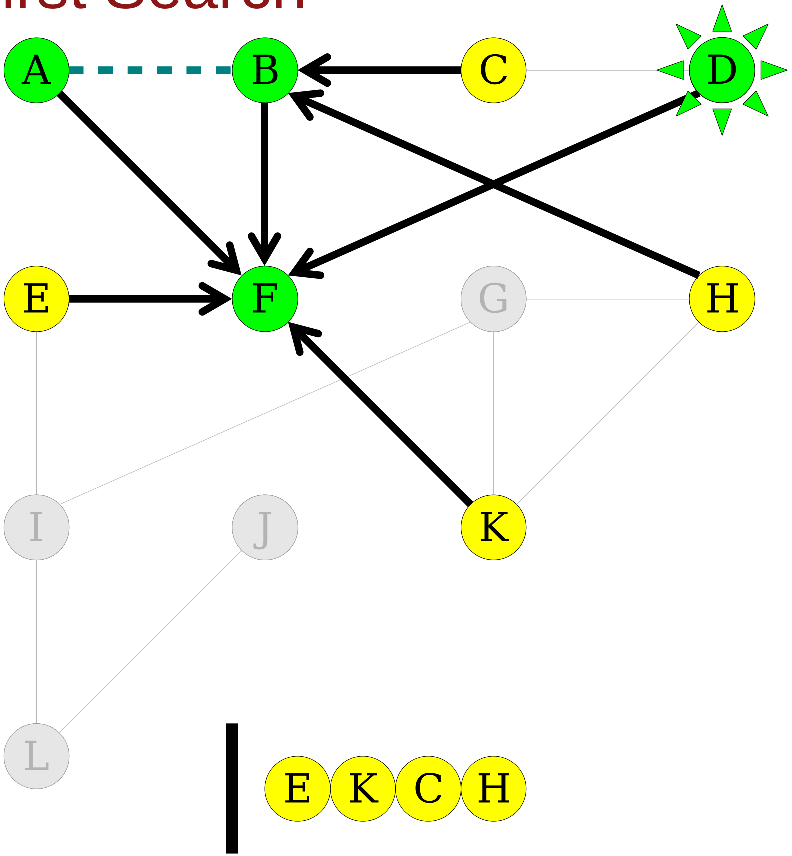
Breadth-First Search



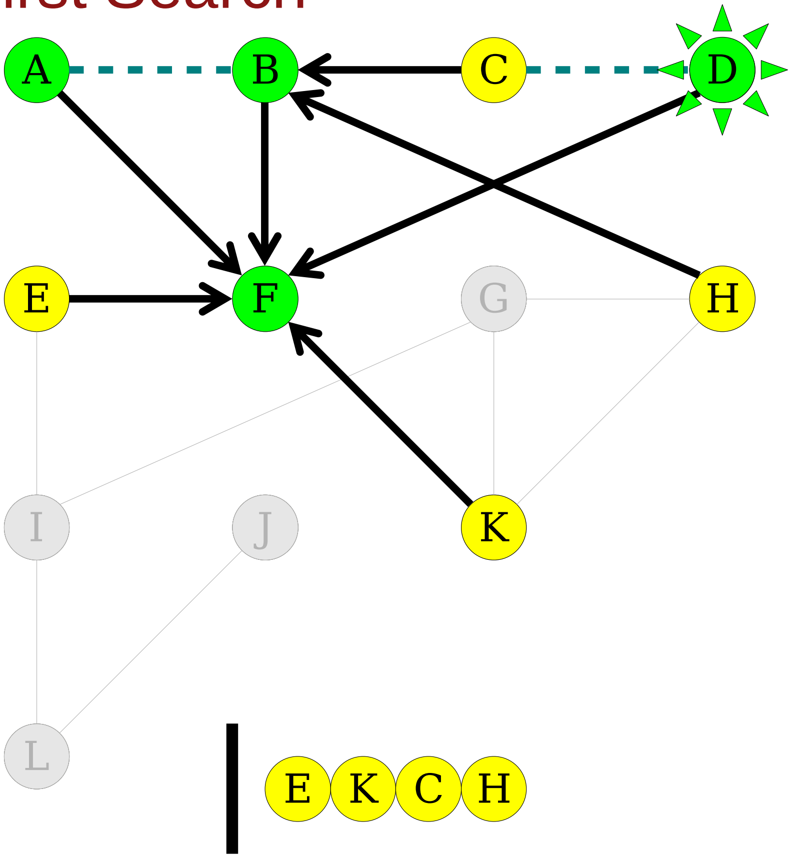
Breadth-First Search



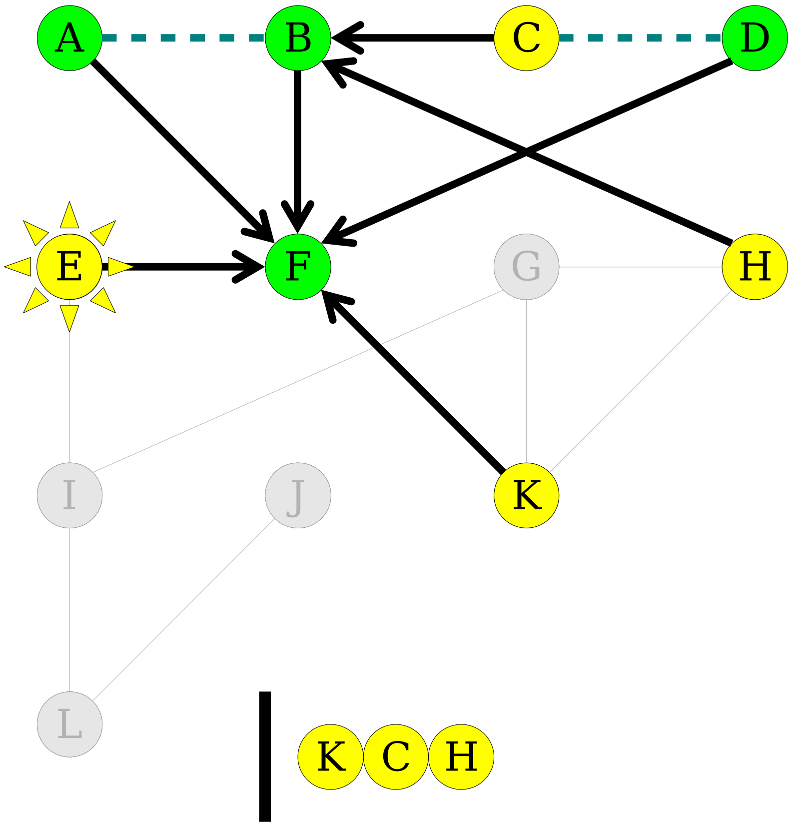
Breadth-First Search



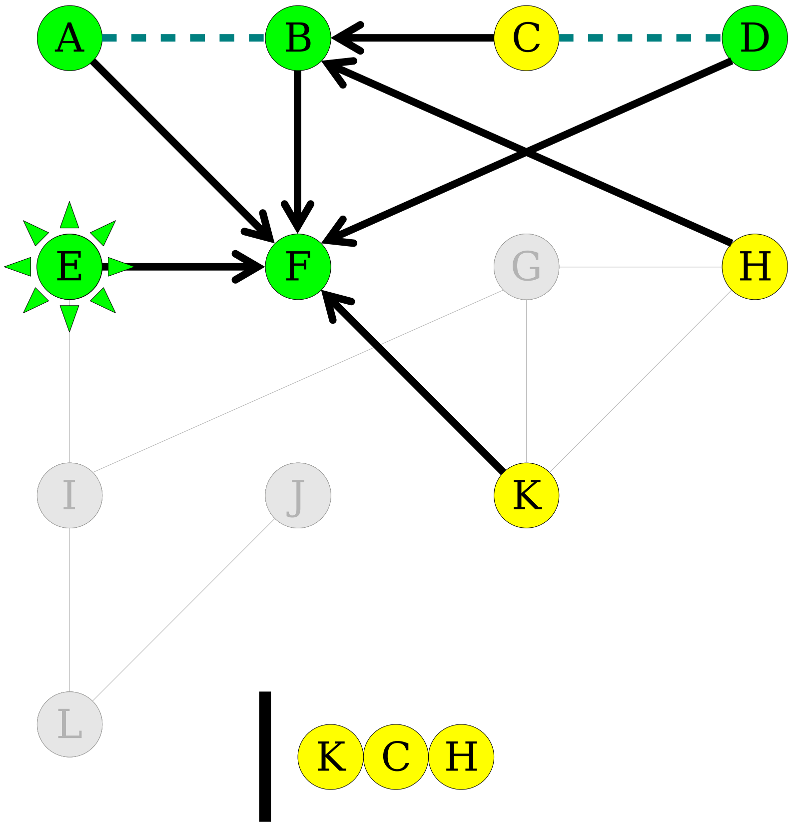
Breadth-First Search



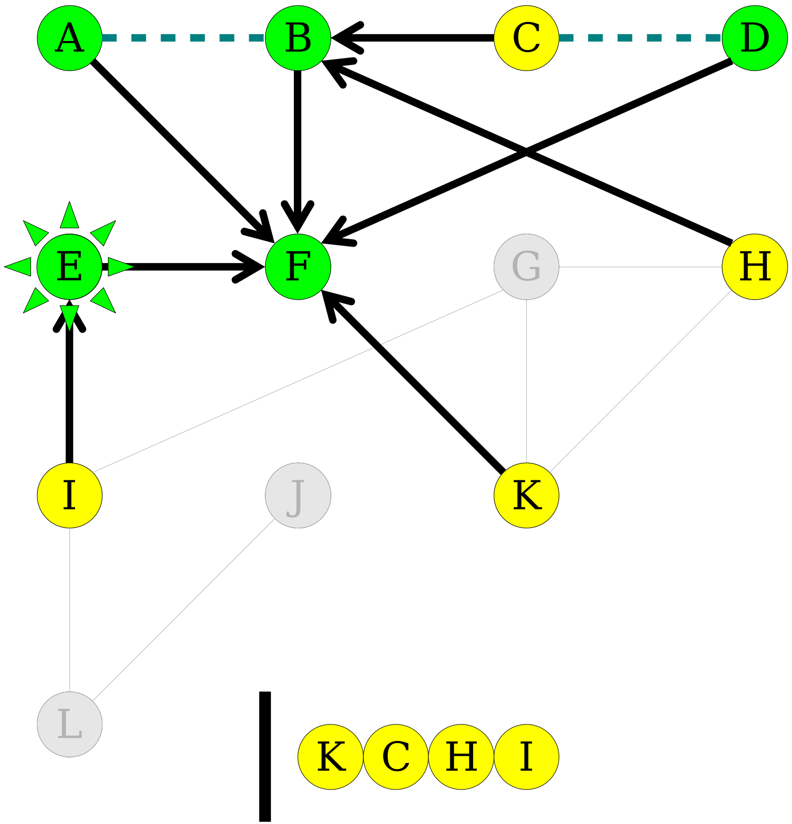
Breadth-First Search



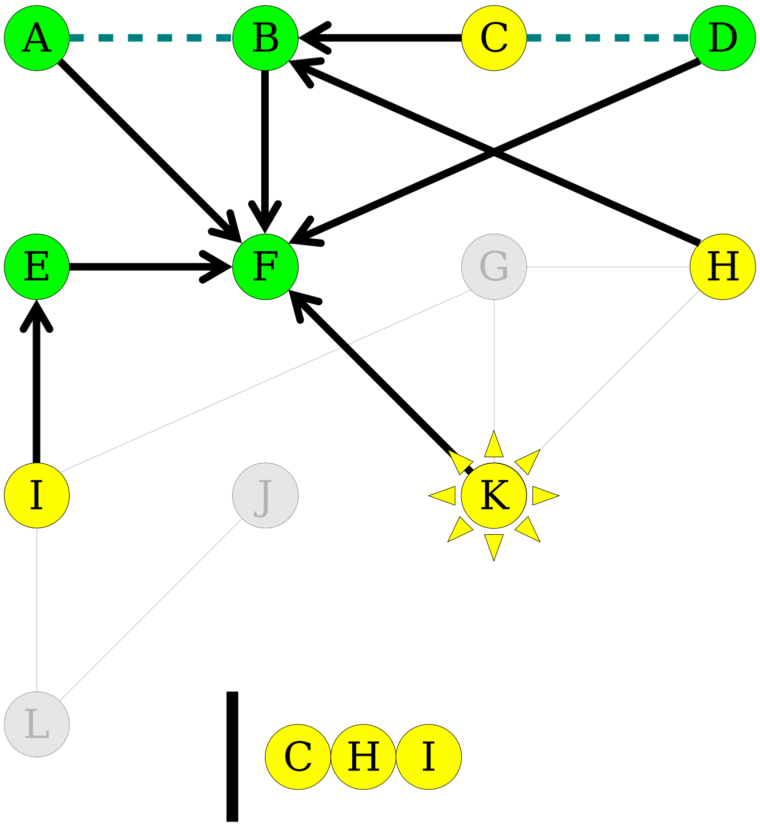
Breadth-First Search



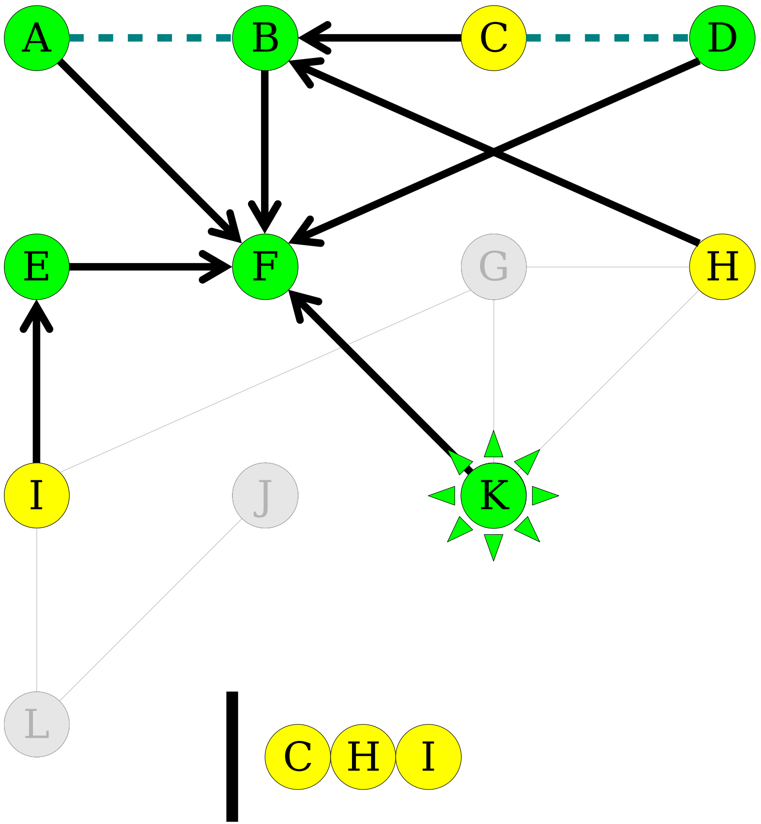
Breadth-First Search



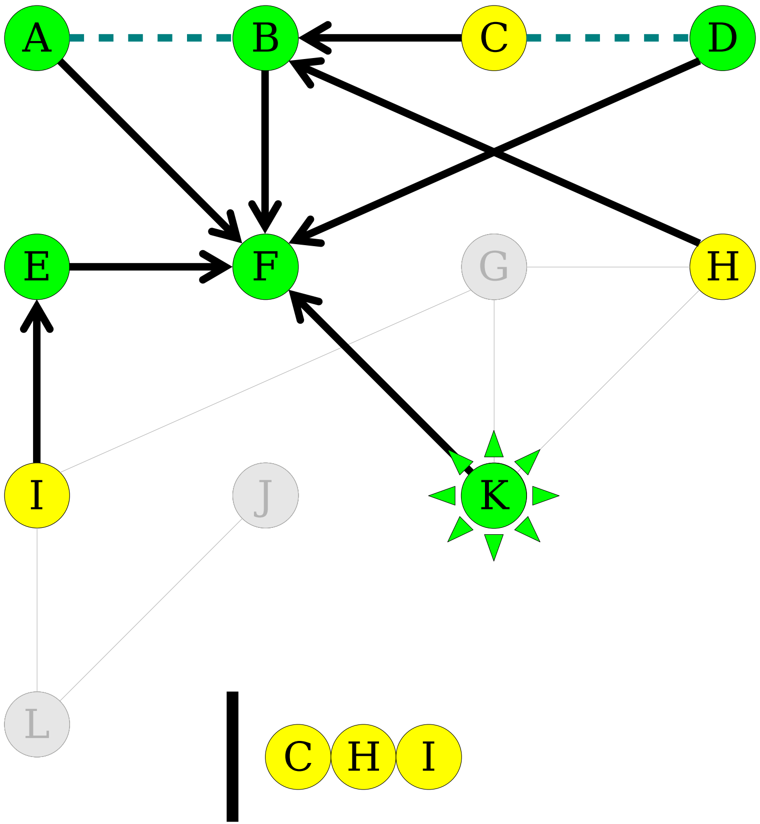
Breadth-First Search



Breadth-First Search



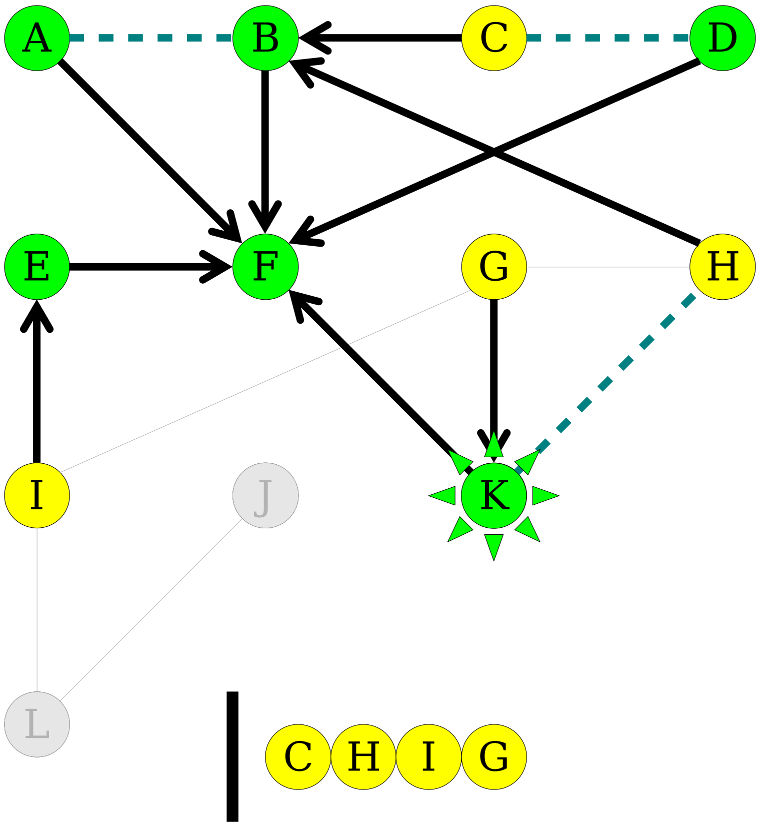
Breadth-First Search



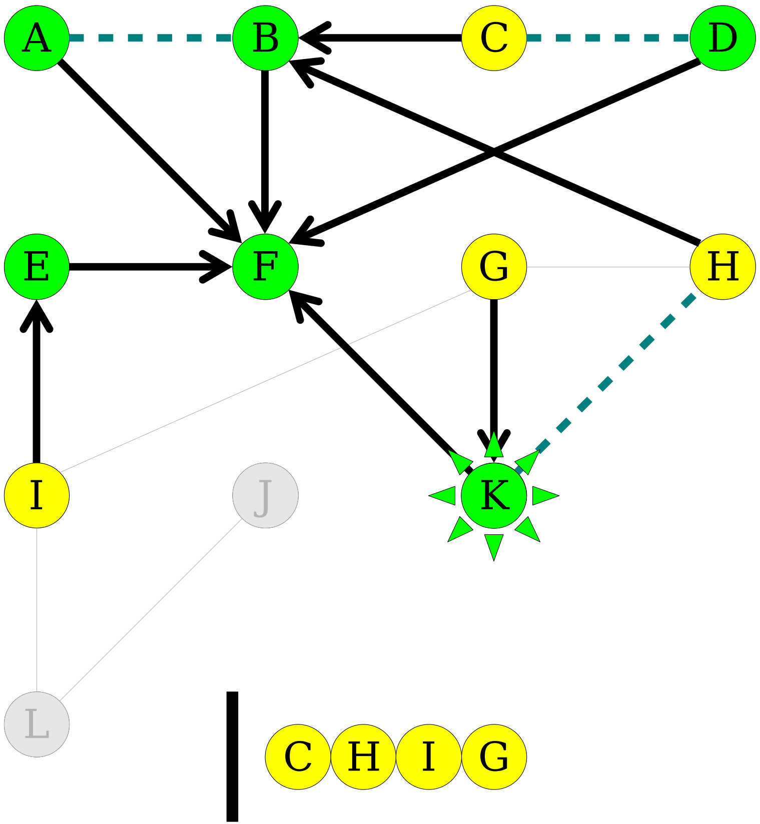
You predict the next slide!

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

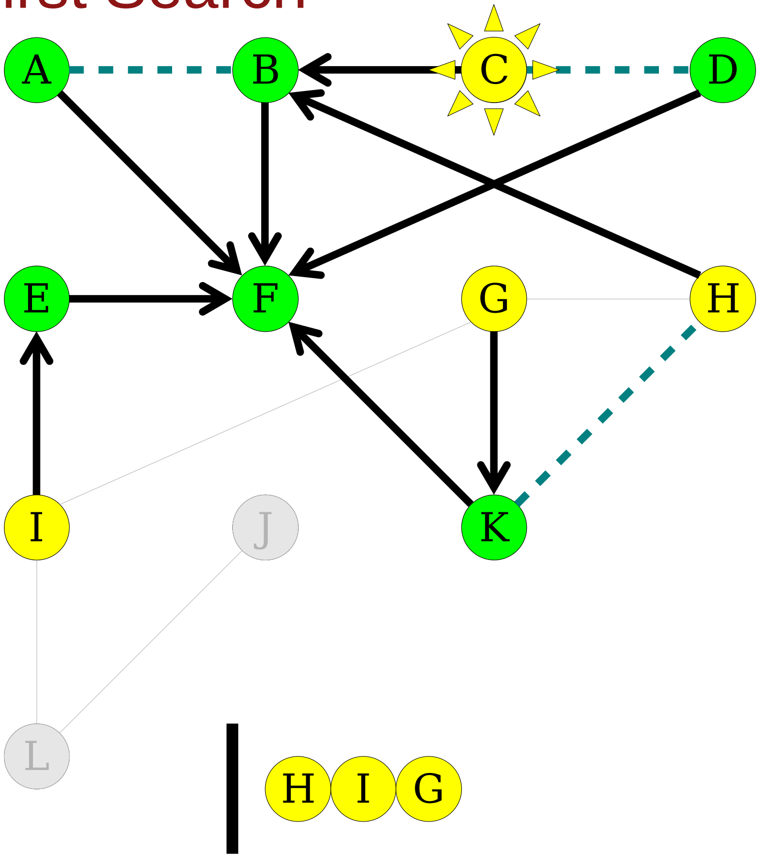
Breadth-First Search



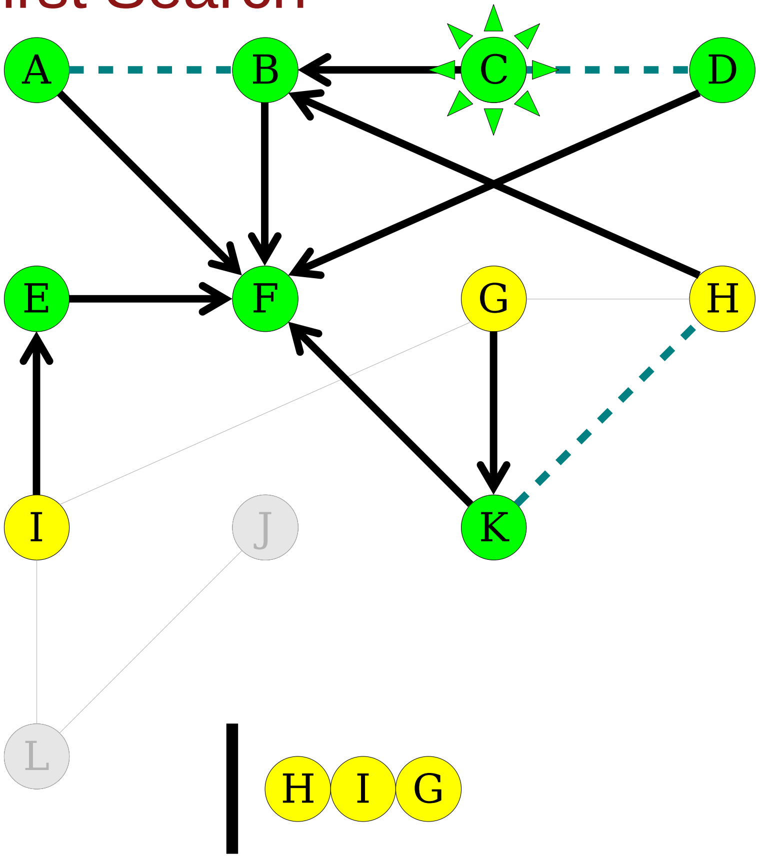
Breadth-First Search



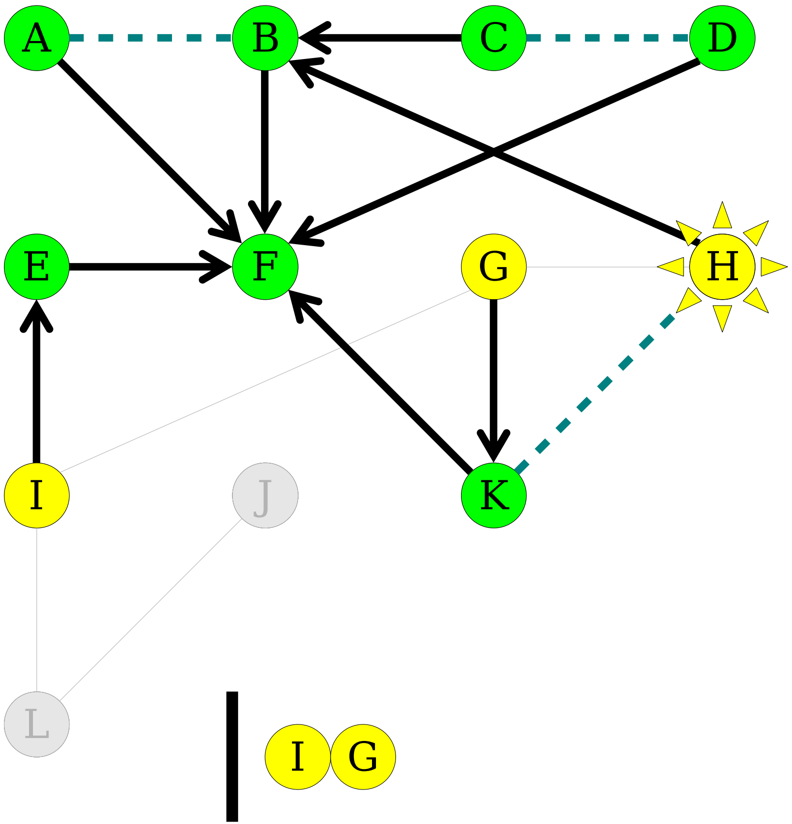
Breadth-First Search



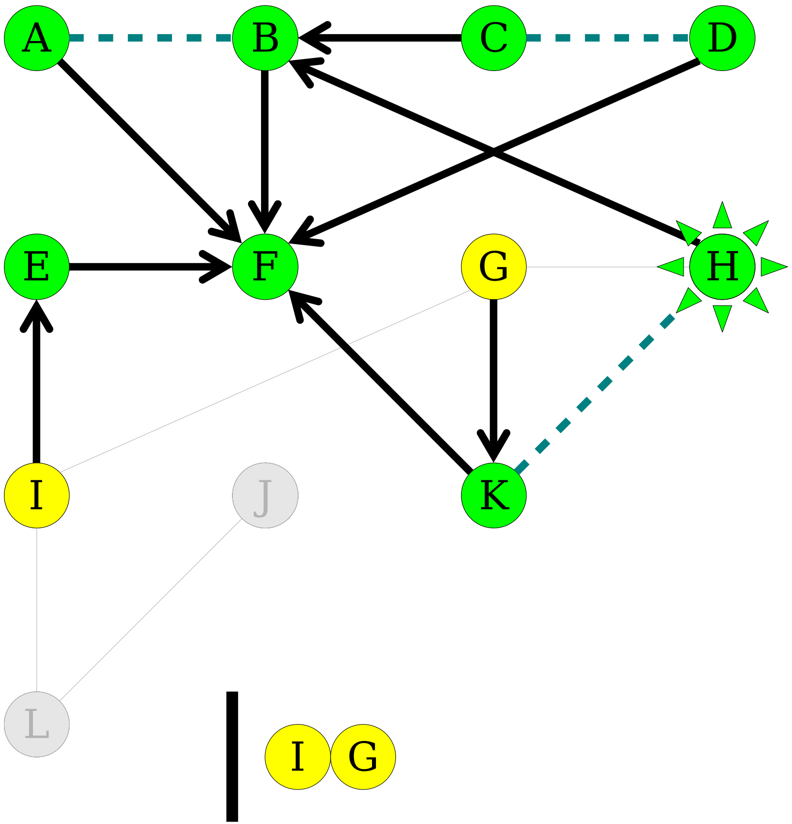
Breadth-First Search



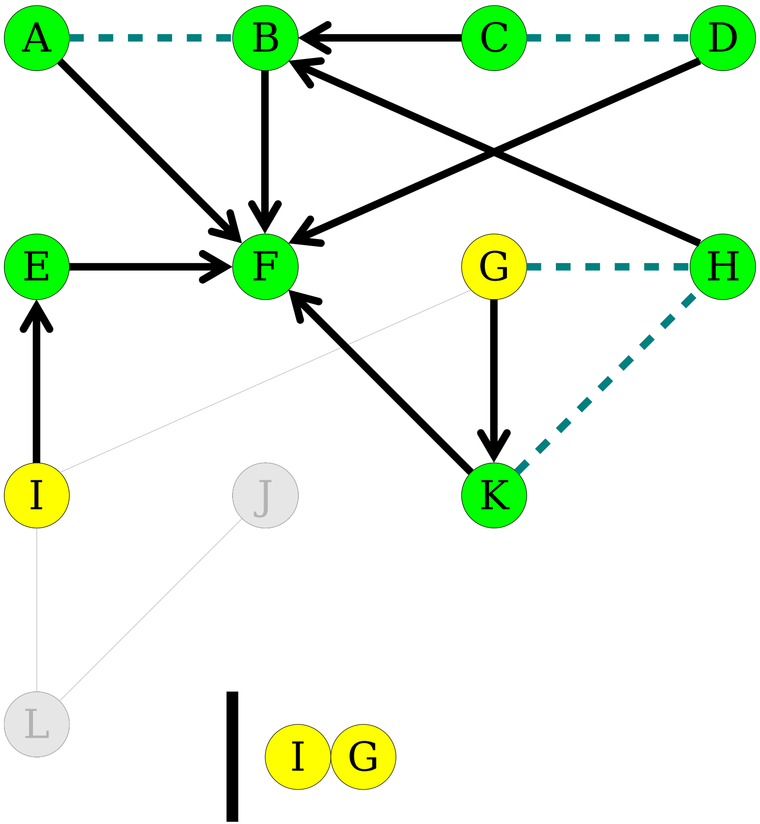
Breadth-First Search



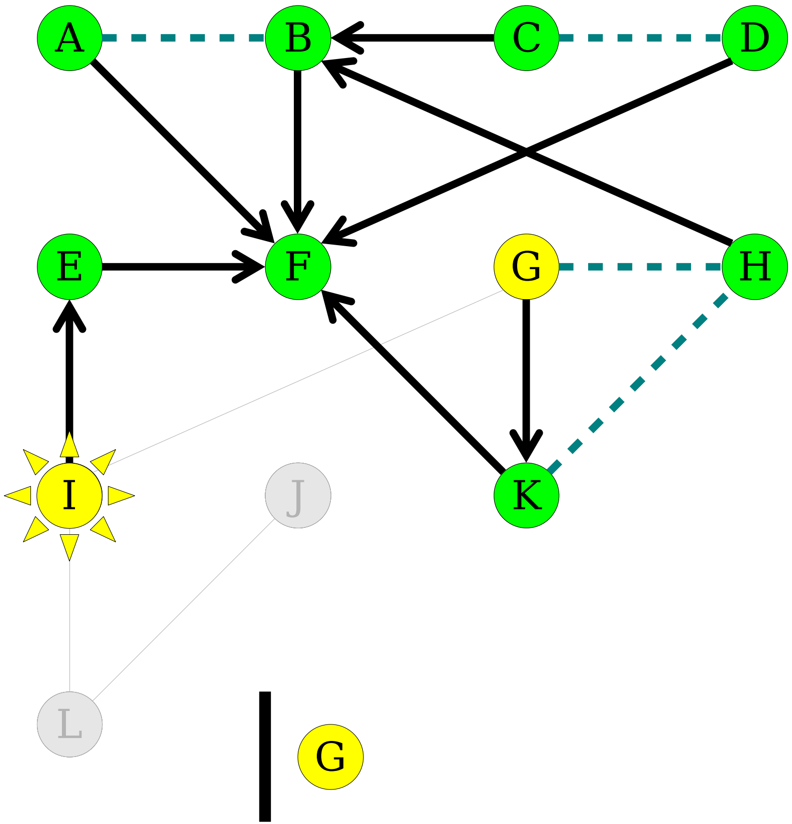
Breadth-First Search



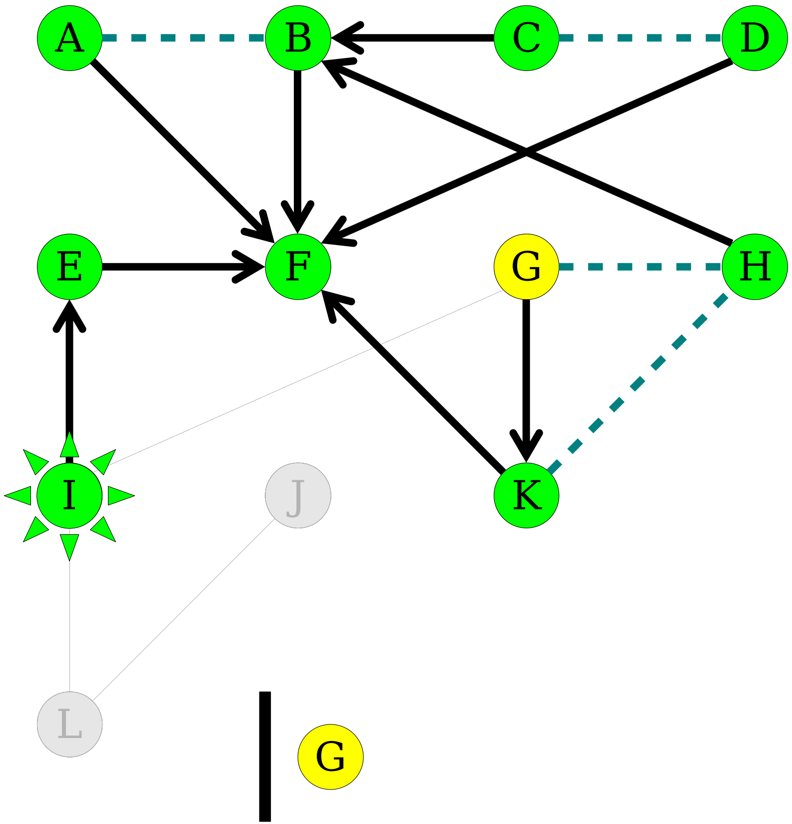
Breadth-First Search



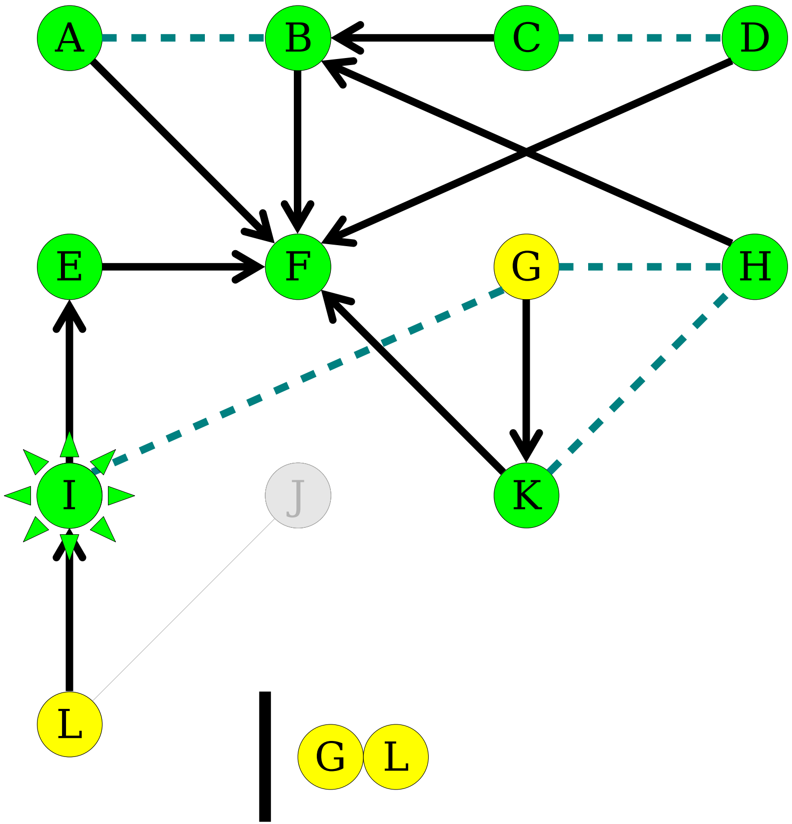
Breadth-First Search



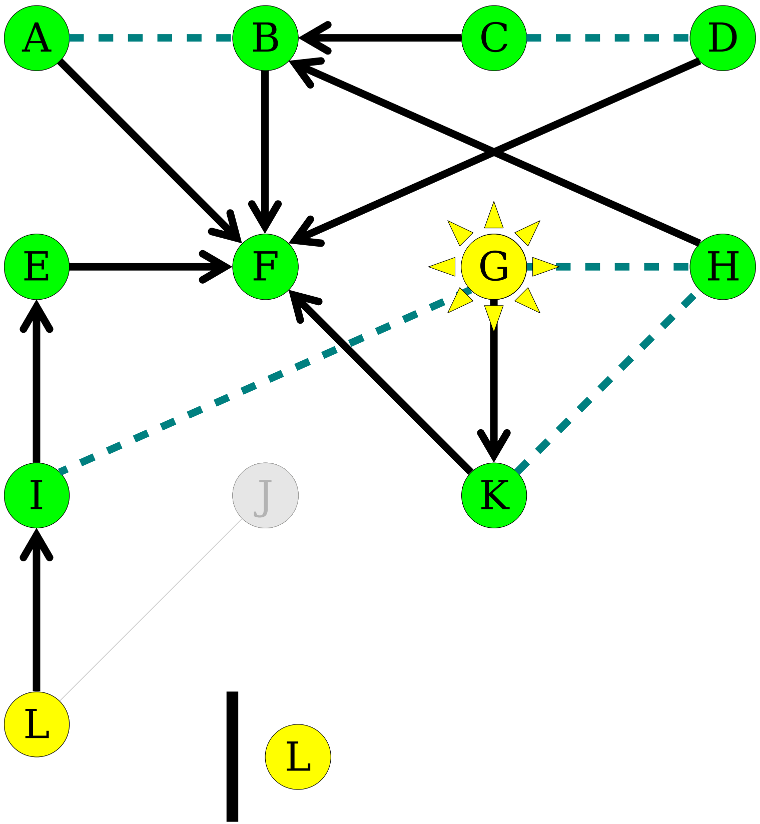
Breadth-First Search



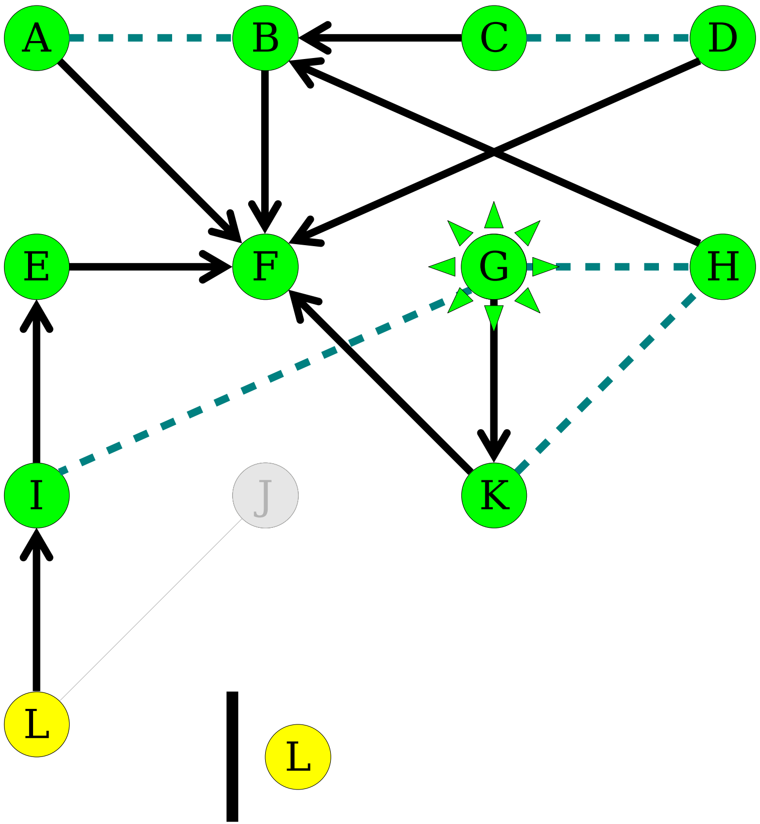
Breadth-First Search



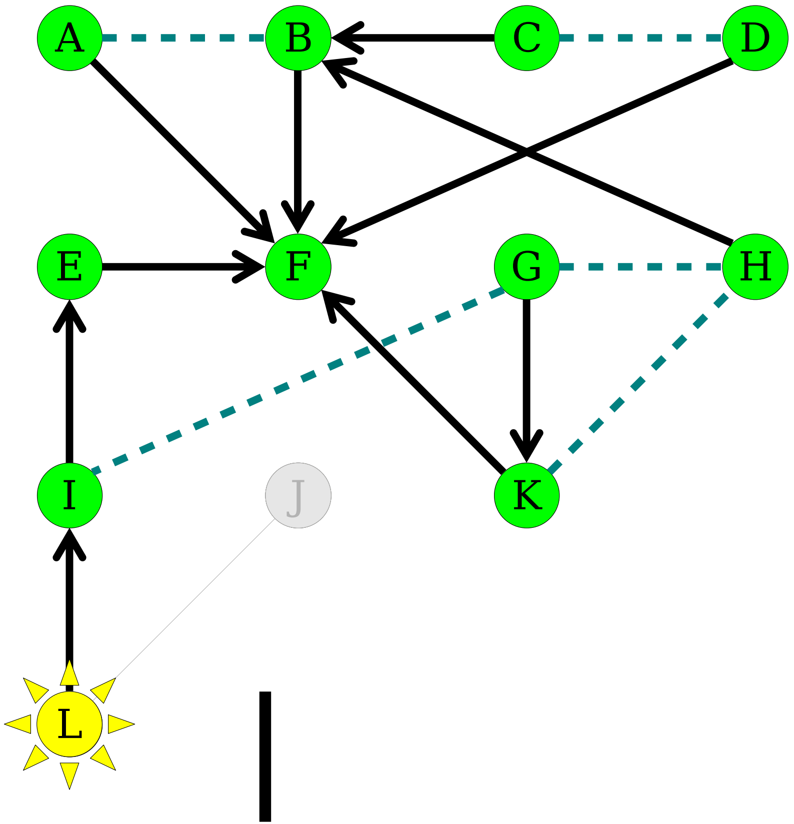
Breadth-First Search



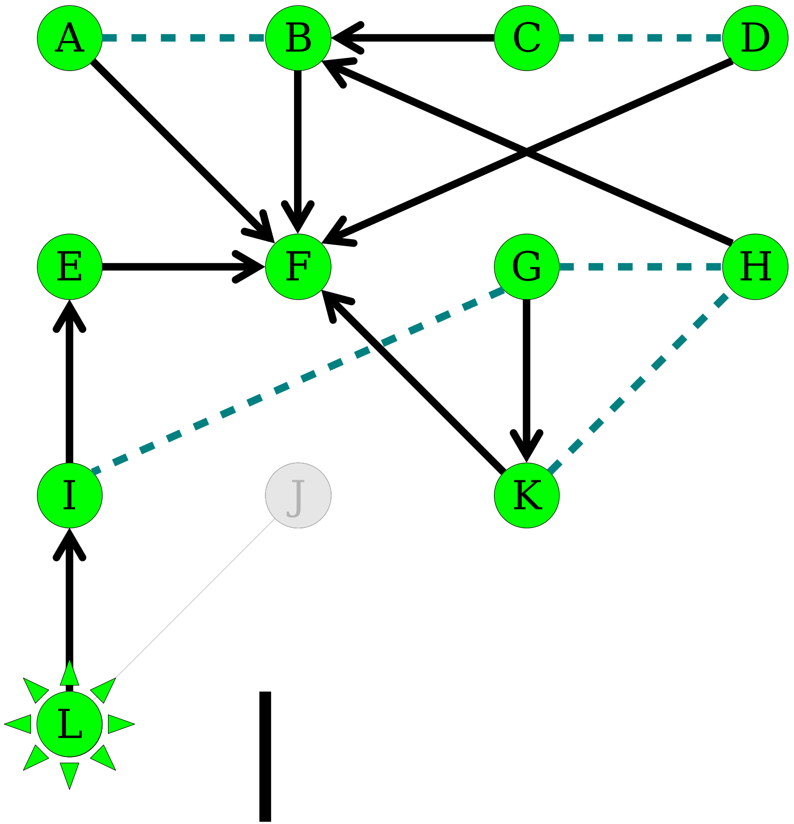
Breadth-First Search



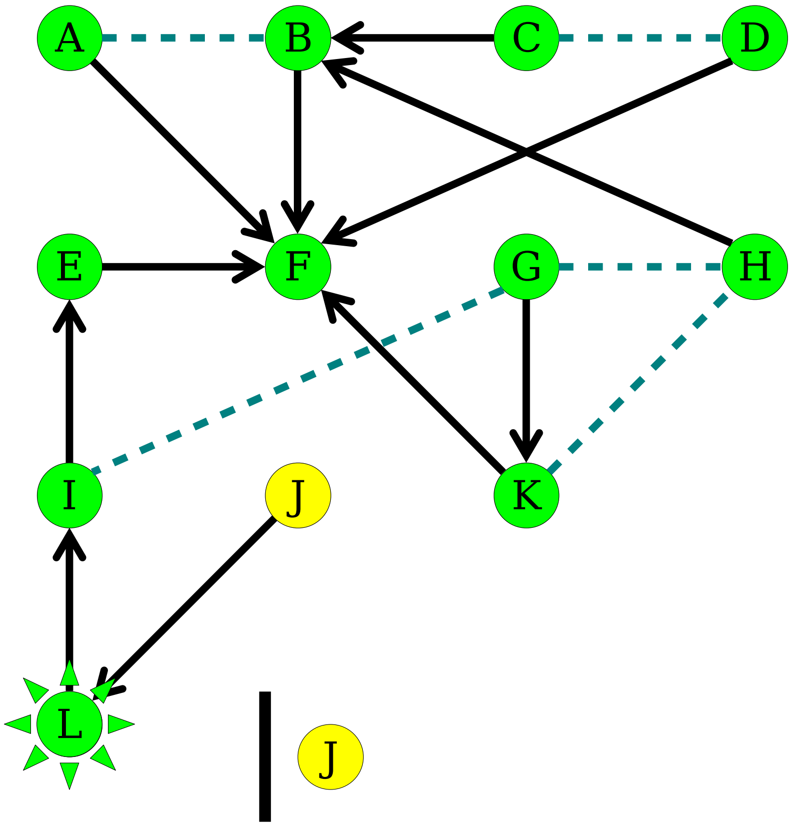
Breadth-First Search



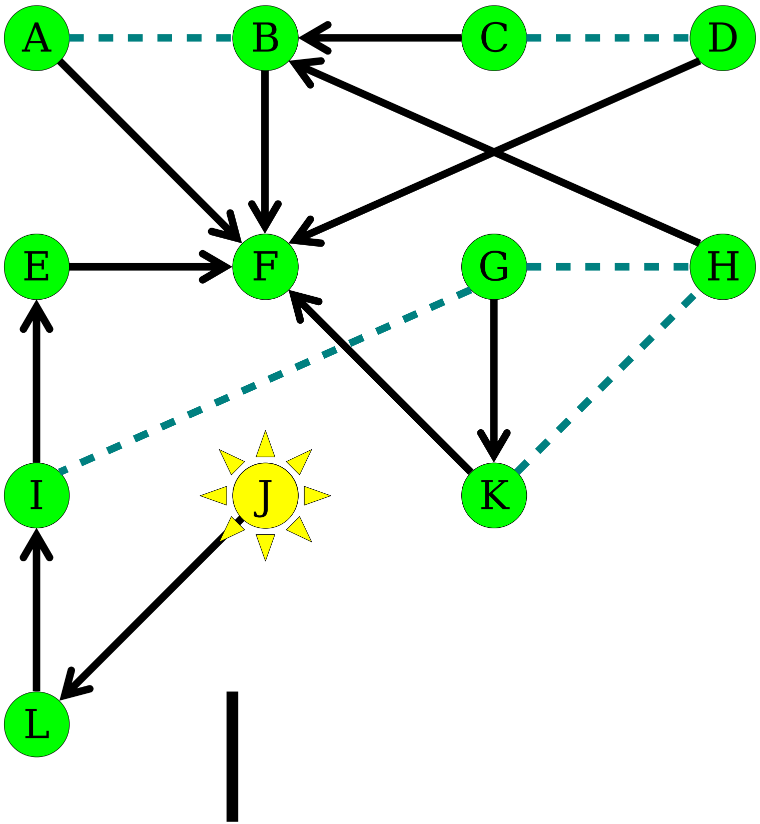
Breadth-First Search



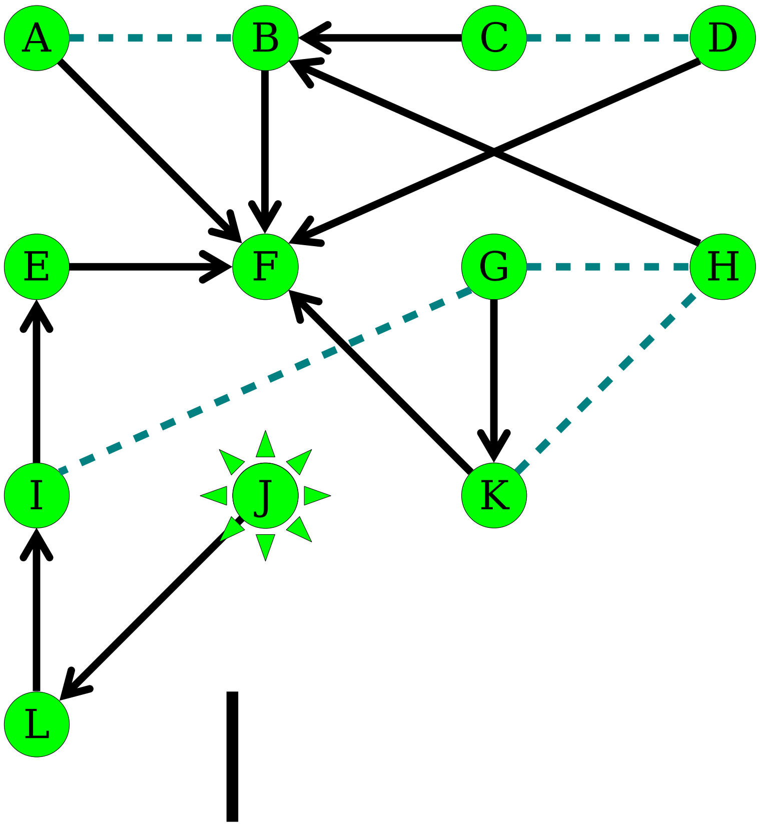
Breadth-First Search



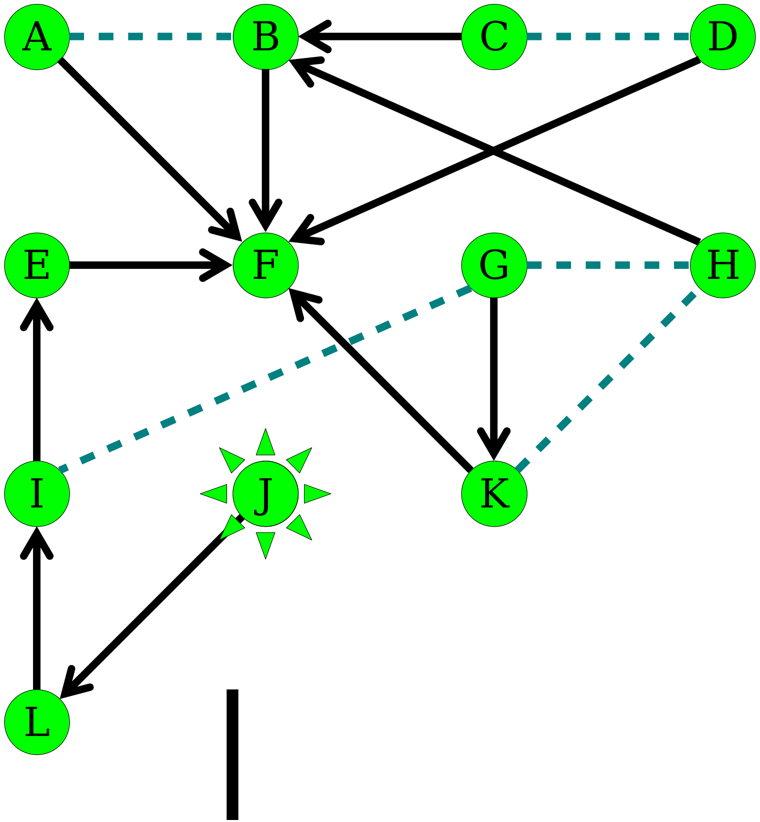
Breadth-First Search



Breadth-First Search



Breadth-First Search



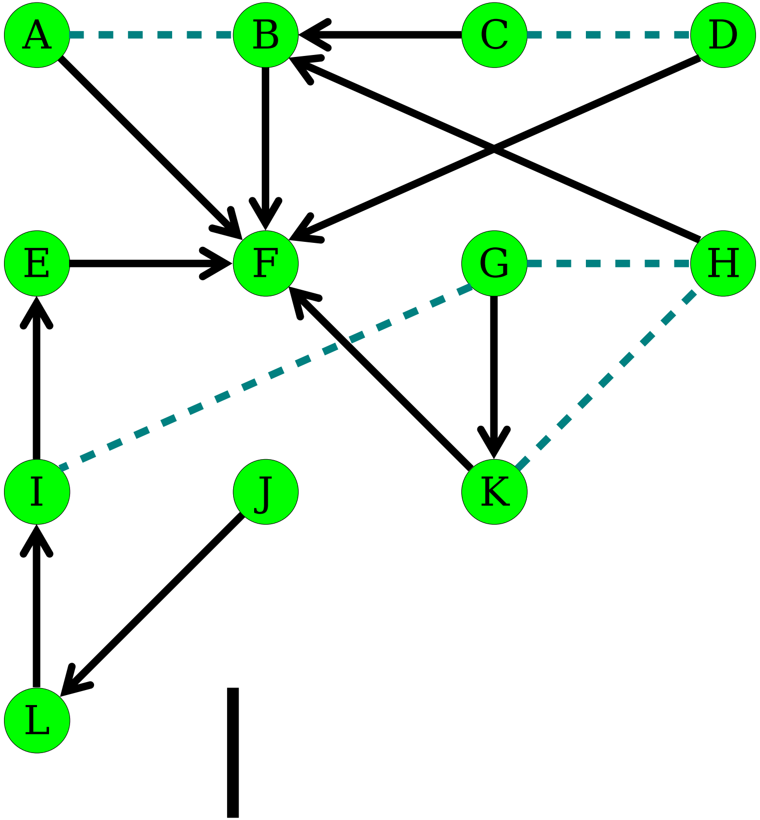
Done!

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

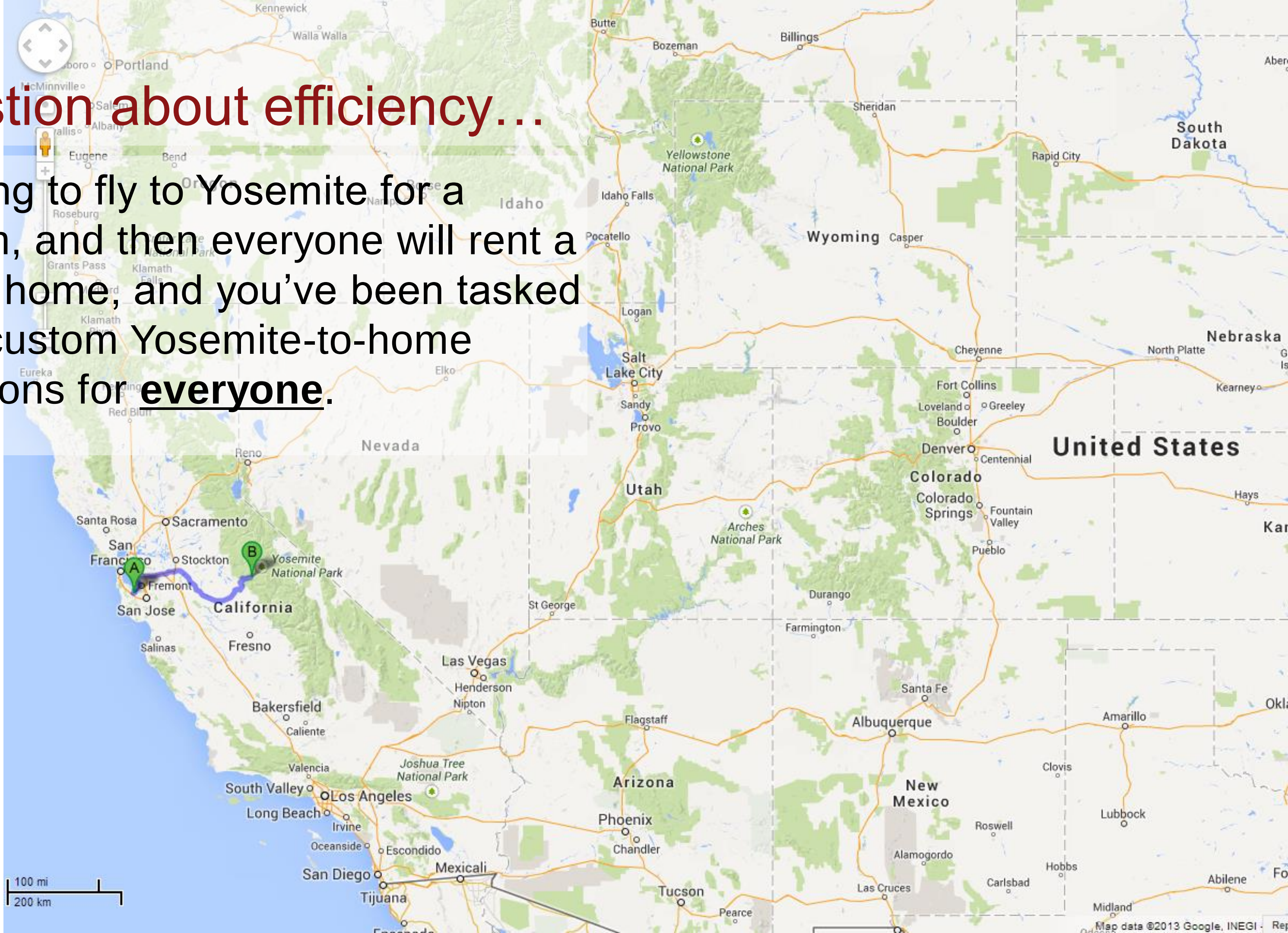
(3) If you follow the parent pointers from the desired end point, you will get back to the start point, and it will be the shortest way to do that

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…

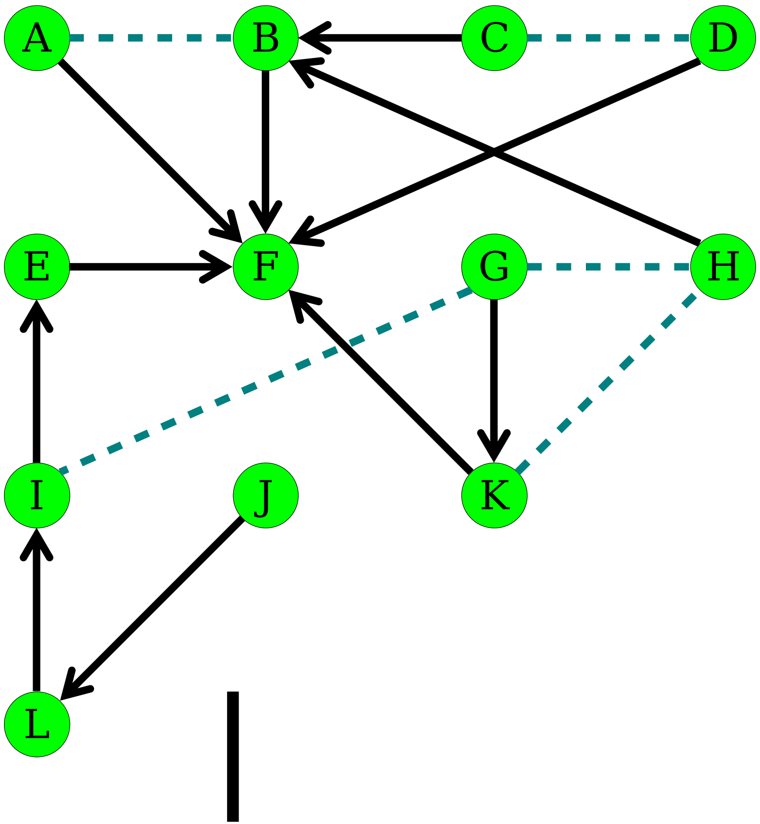
You calculated the shortest path for yourself to return home from the reunion (Yosemite to Palo Alto) and let's just say that it took time **X** = O((|E| + |V|)log|V|)

- 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)
B. O(|E|*|V|* X)
C. X
D. Other/none/more

Breadth-First Search



THINGS TO NOTICE:
(4) We now have the answer to the question "What is the shortest path to you from F?" for ***every single node in the graph!!***