Programming Abstractions

CS106B

Cynthia Lee

Graphs Topics

Graphs!

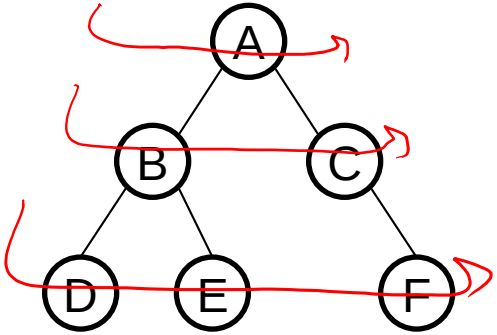
1. **Basics**
   - What are they? How do we represent them?
2. **Theorems**
   - What are some things we can prove about graphs?
3. **Breadth-first search on a graph**
   - Spoiler: just a very, very small change to tree version
4. **Dijkstra's shortest paths algorithm**
   - Spoiler: just a very, very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

Breadth-First Search

We've seen BFS before this quarter!

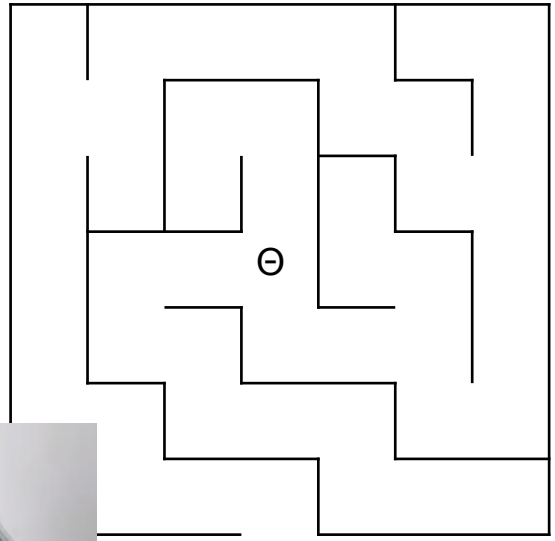
BFS in this class so far

Word Ladder Assignment



Trees

Slime Mold



Maze

BFS Code (Maze version)

```
bool solveMazeQueue(Maze & maze, Point start) {
    Vector<Direction> compass;
    compass += WEST, SOUTH, EAST, NORTH;
    Queue<Point> toExplore;
    toExplore.enqueue(start);
    while(!toExplore.isEmpty()){
        Point current = toExplore.dequeue();
        if (maze.isOutside(current)) return true;
        if (maze.isMarked(current)) continue;
        maze.markSquare(current);
        pause(200);
        for (Direction dir : compass) {
            if (!maze.wallExists(current, dir)) {
                toExplore.enqueue(adjacentPoint(current,dir));
            }
        }
    }
    return false;
}
```

While there are still things in the queue, keep exploring!

1. Make a Queue to remember places we want to explore in the future

2. Enqueue starting point

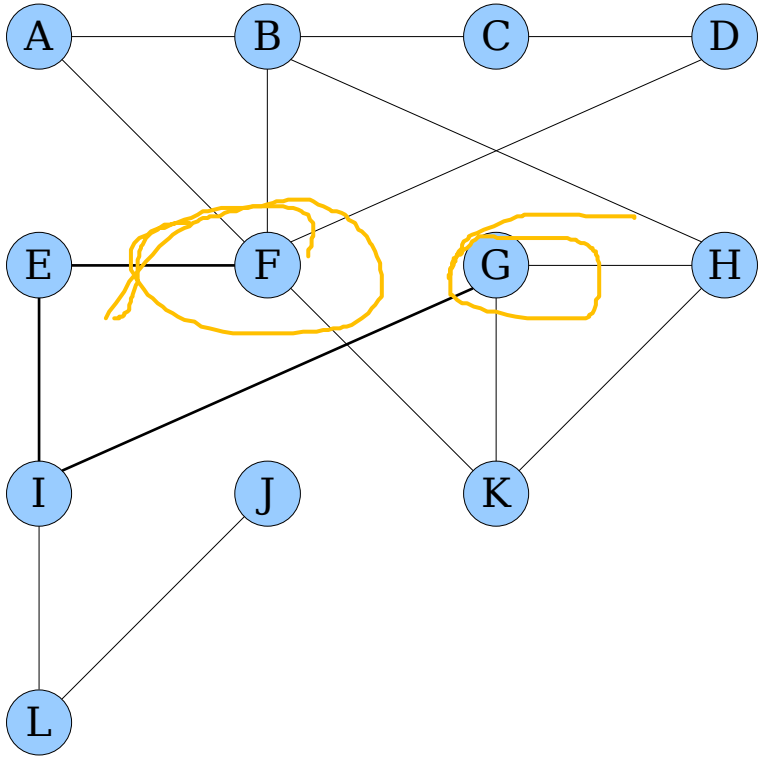
4. Dequeue a point and visit it

5. Enqueue any new points you discover while visiting the current point

Breadth-First Search in a Graph

Graph algorithms

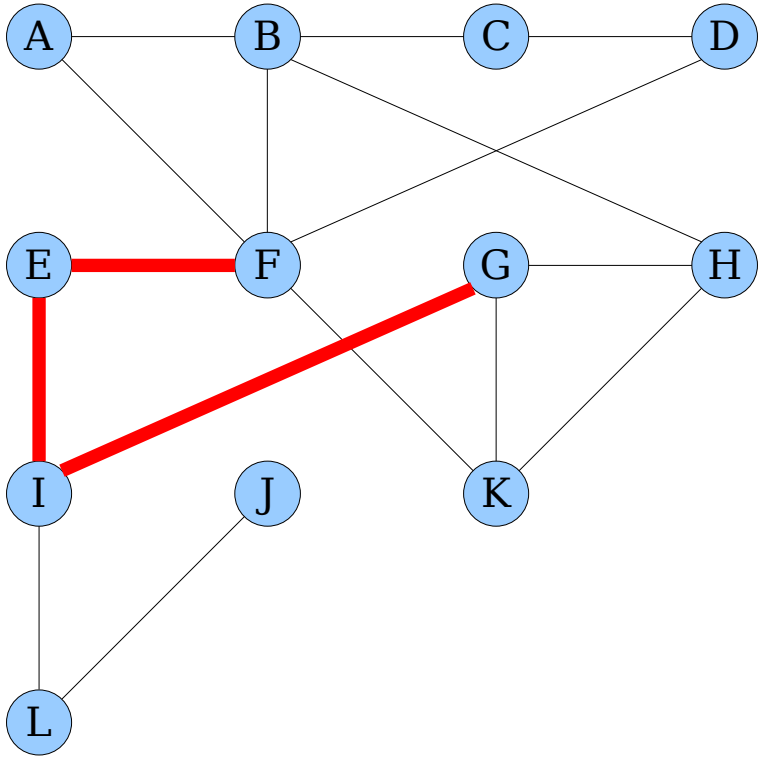
Breadth-First Search



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

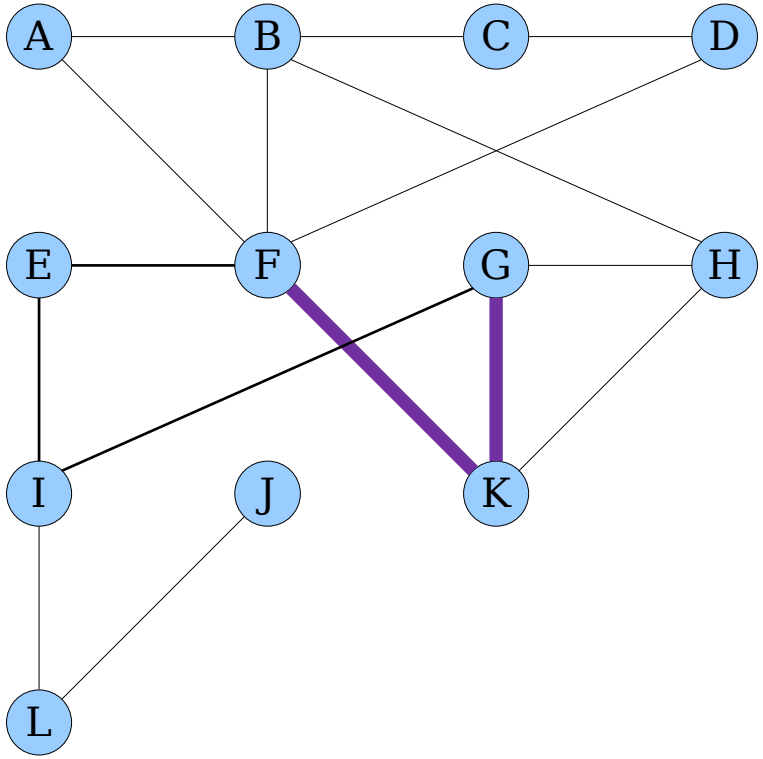


BFS is useful for finding the <u>shortest path</u> between two nodes.

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

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

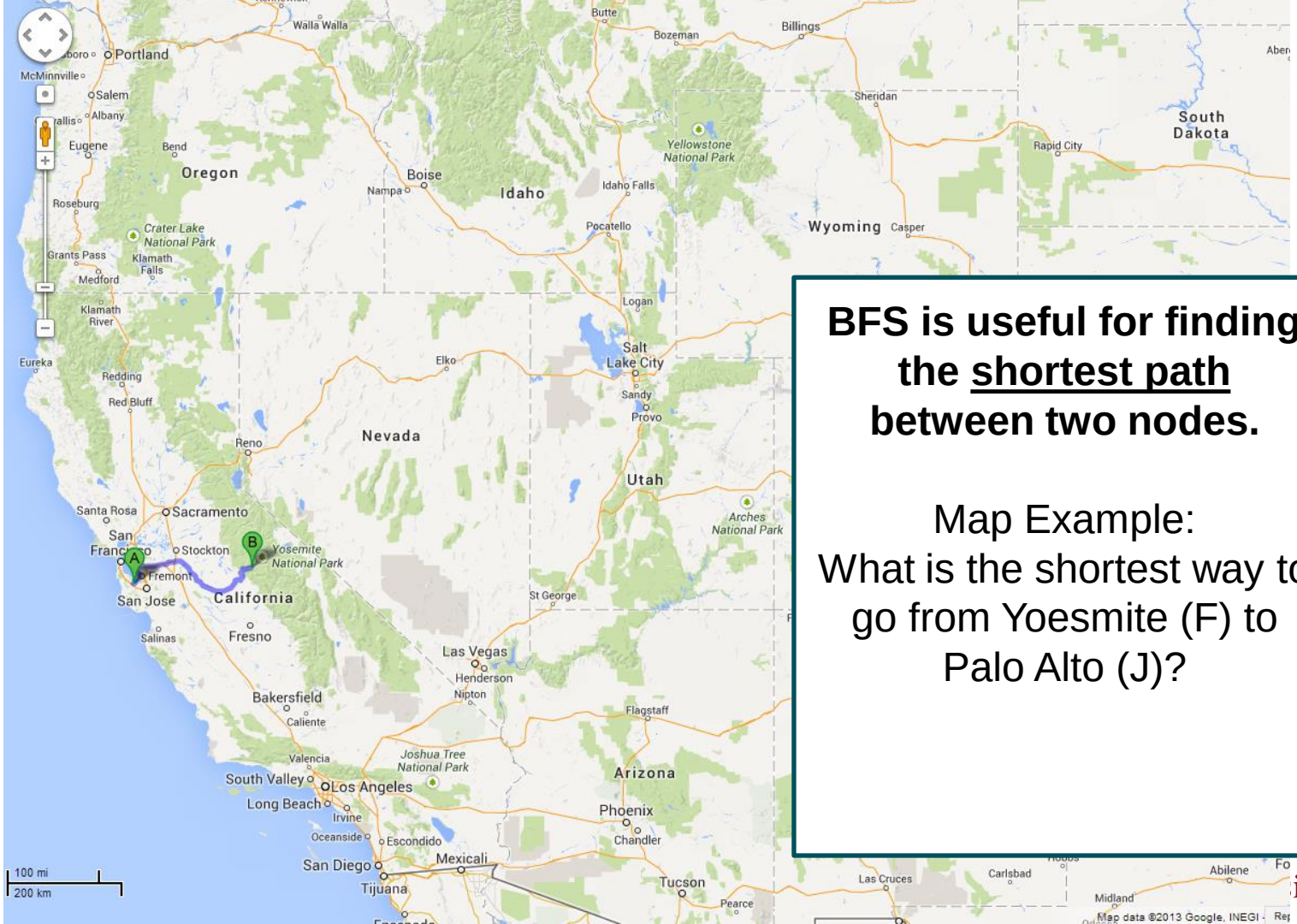
Breadth-First Search



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?

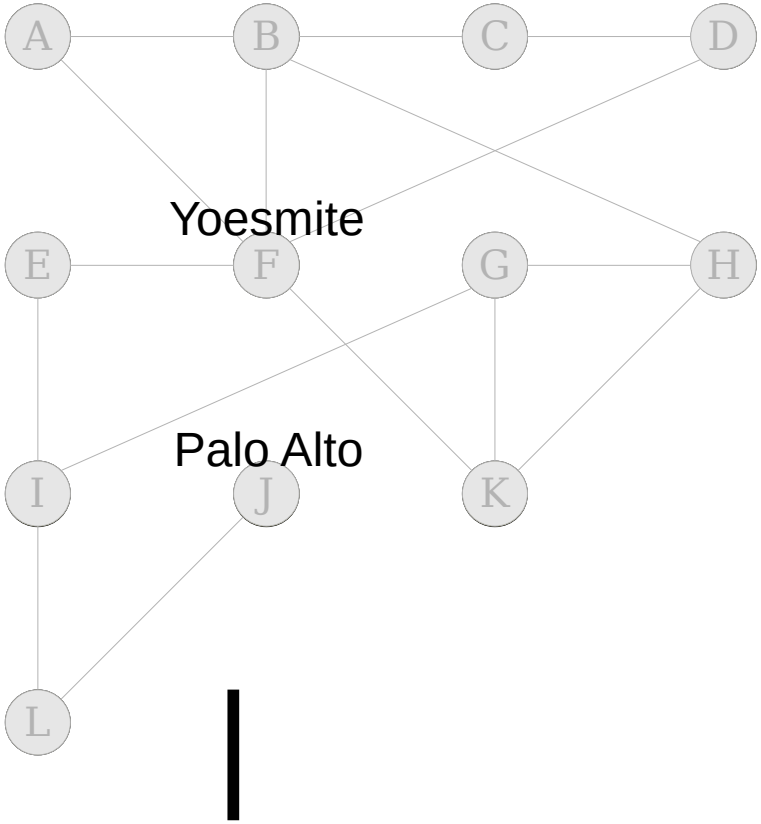
Way 2: F->K->G
2 edges



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

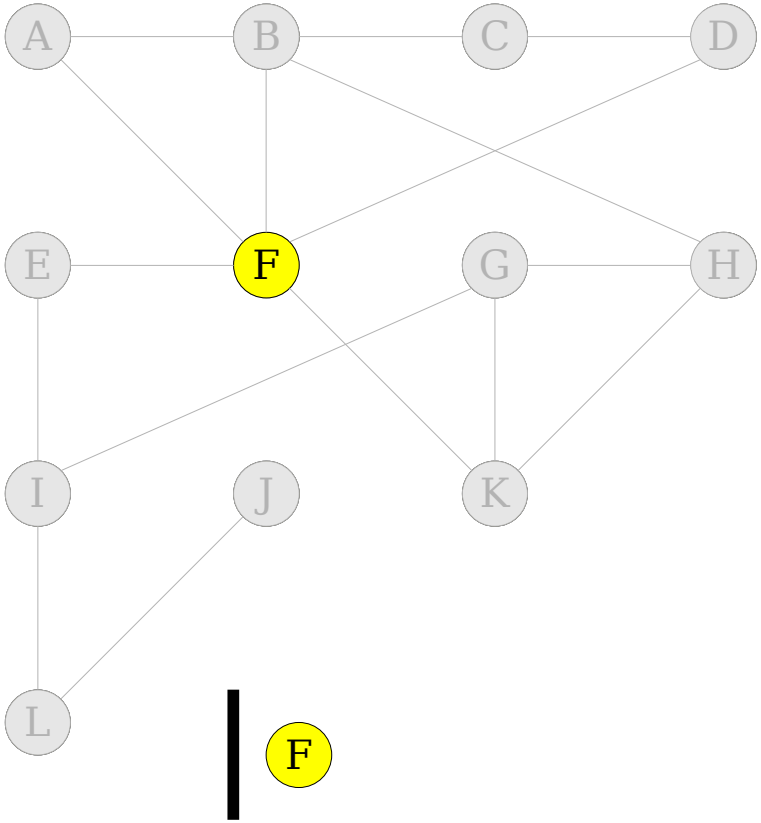
Map Example:
What is the shortest way to go from Yoesmite (F) to Palo Alto (J)?

Breadth-First Search



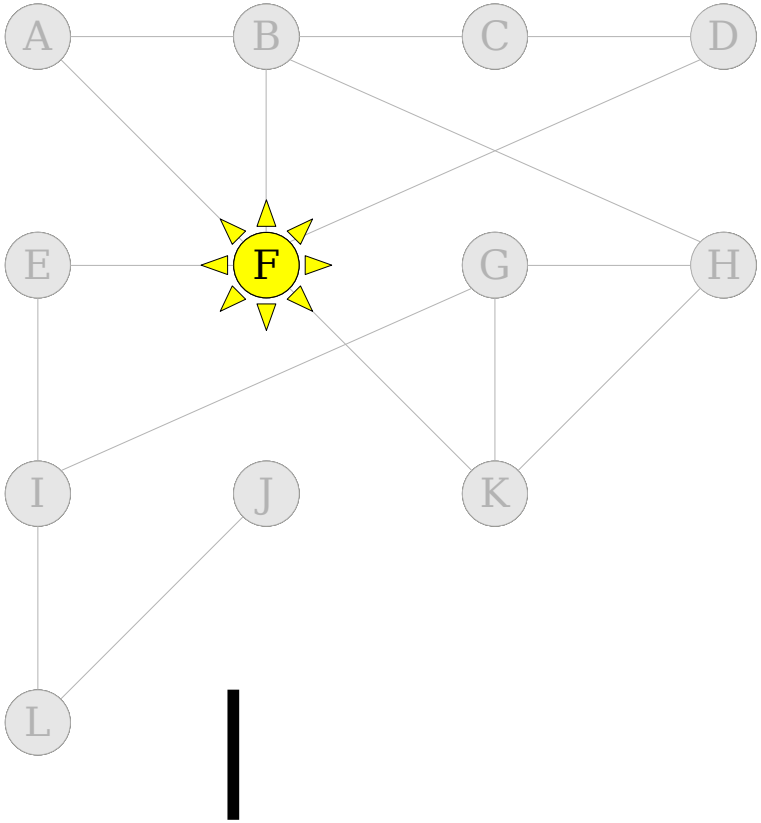
TO START:
(1)Color all nodes GREY
(2)Queue is empty

Breadth-First Search



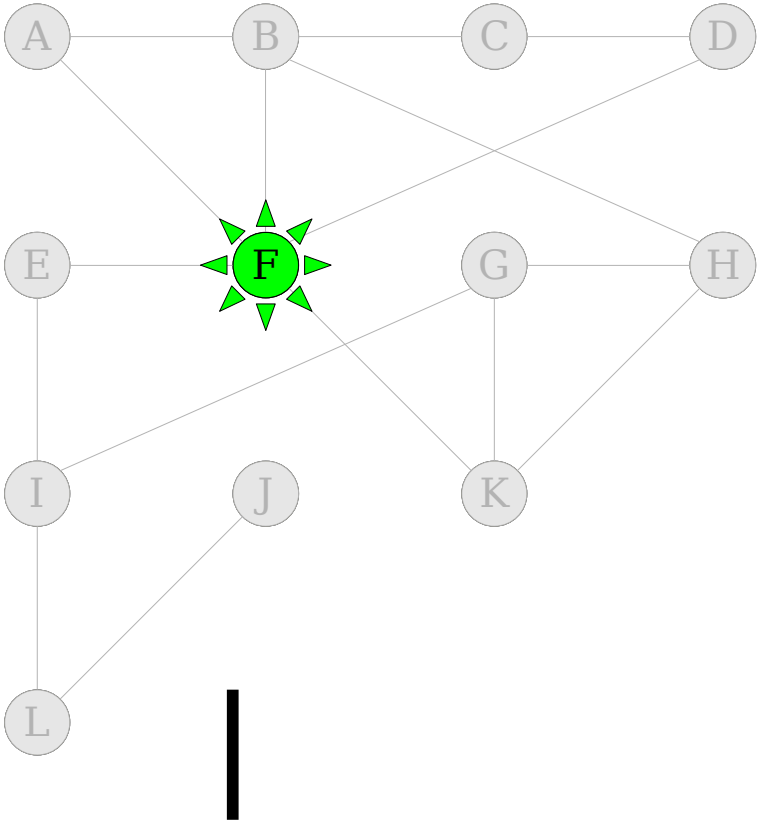
TO START (2):
(1)Enqueue the desired **start** node
(2)Note that anytime we enqueue a node, we mark it YELLOW

Breadth-First Search

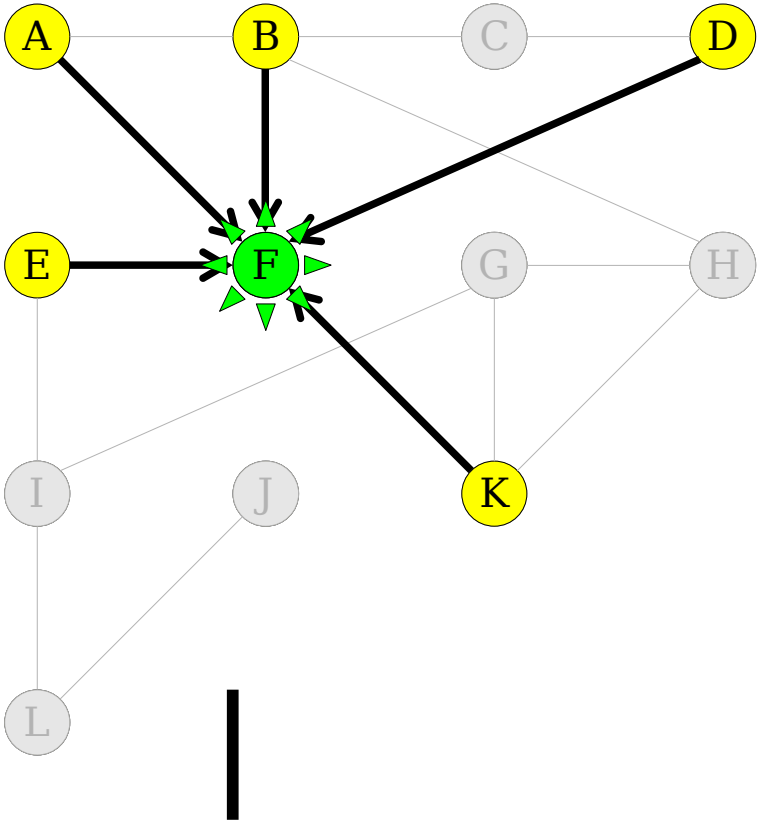


LOOP PROCEDURE:
(1)Dequeue a node
(2)Mark current node GREEN
(3)Set current node's GREY neighbors' parent pointers to current node, then enqueue them (remember: set them YELLOW)

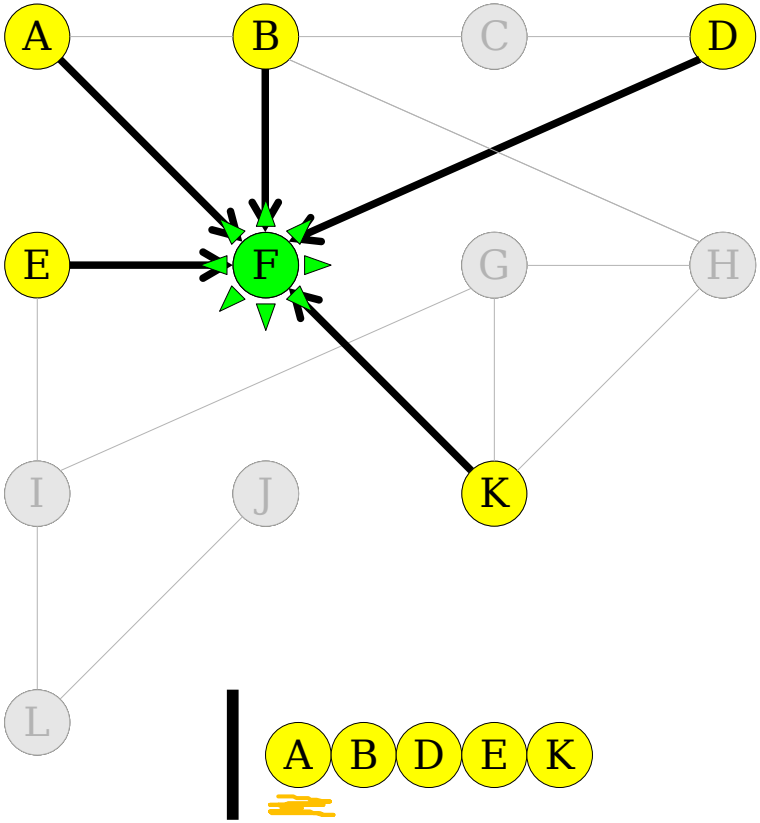
Breadth-First Search



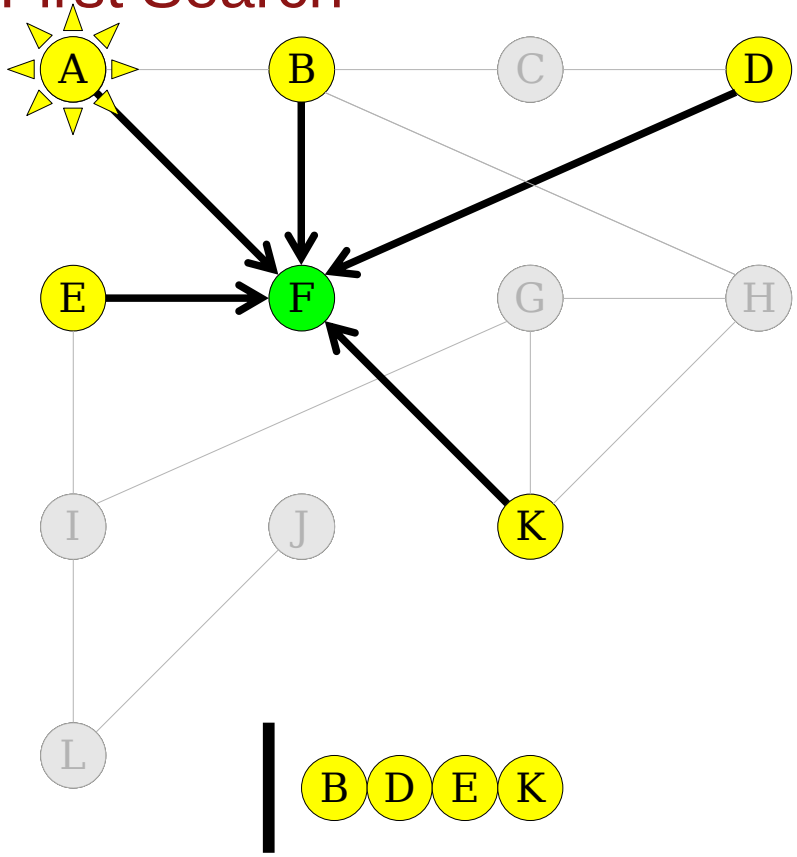
Breadth-First Search



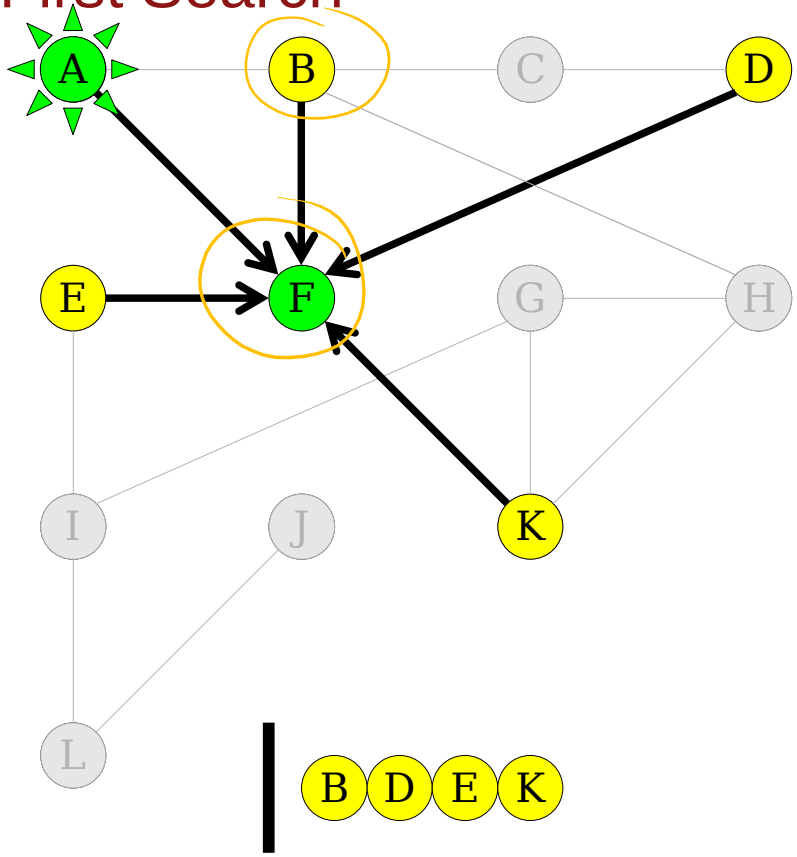
Breadth-First Search



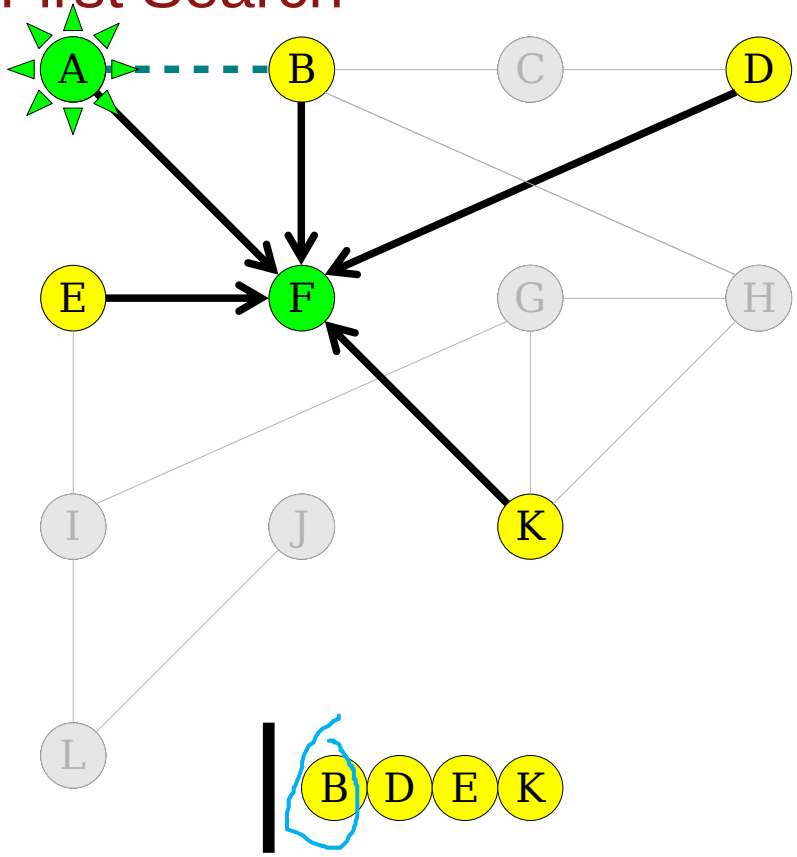
Breadth-First Search



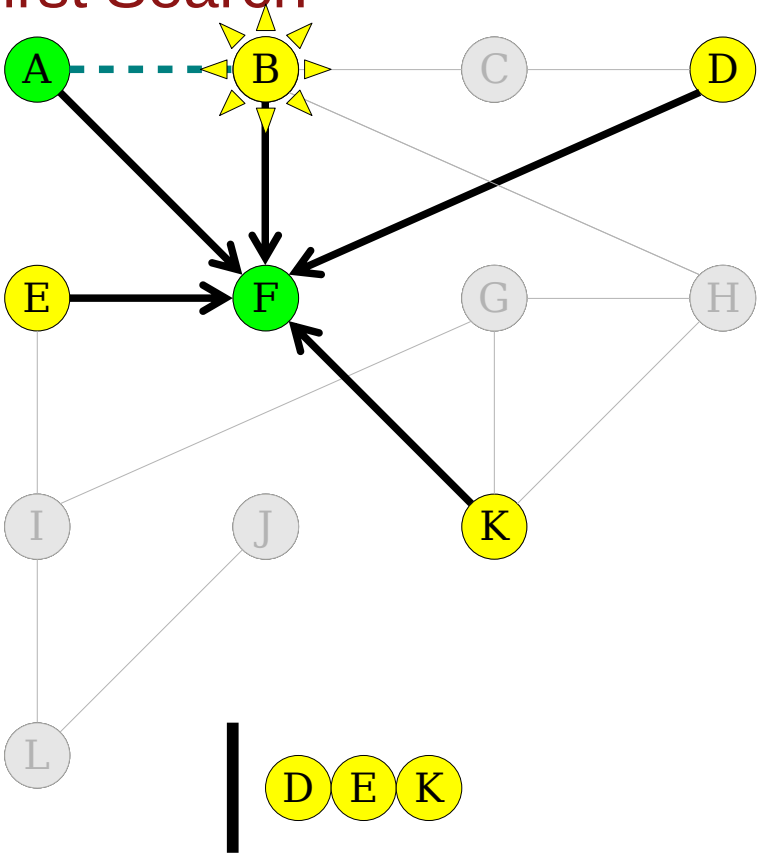
Breadth-First Search



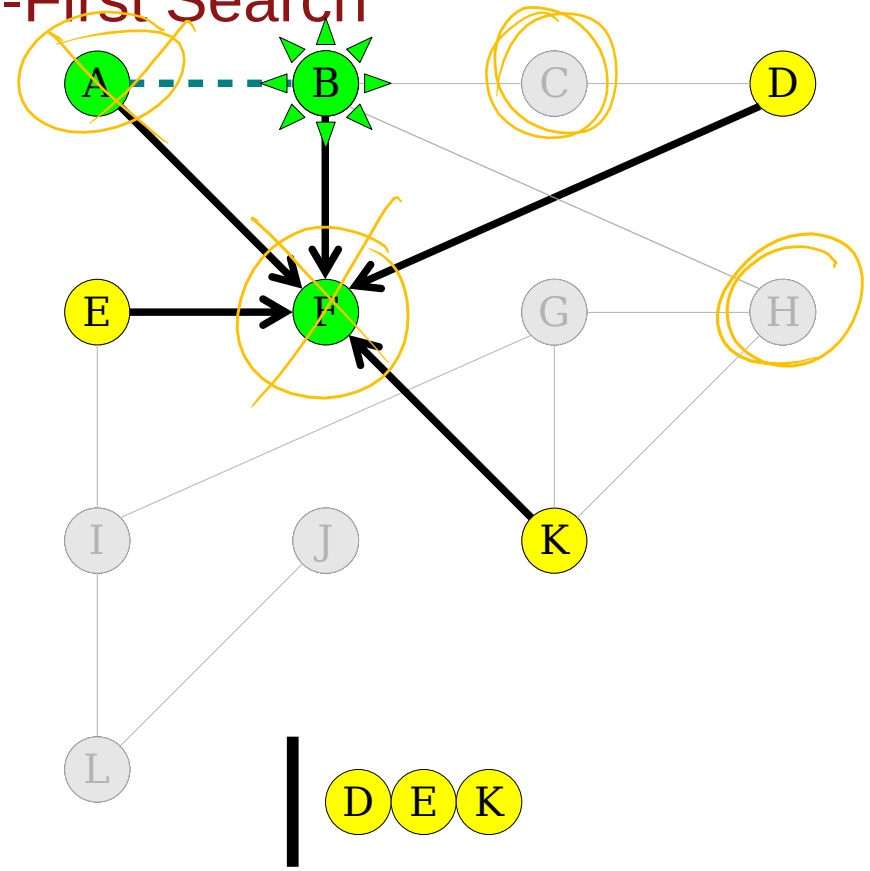
Breadth-First Search



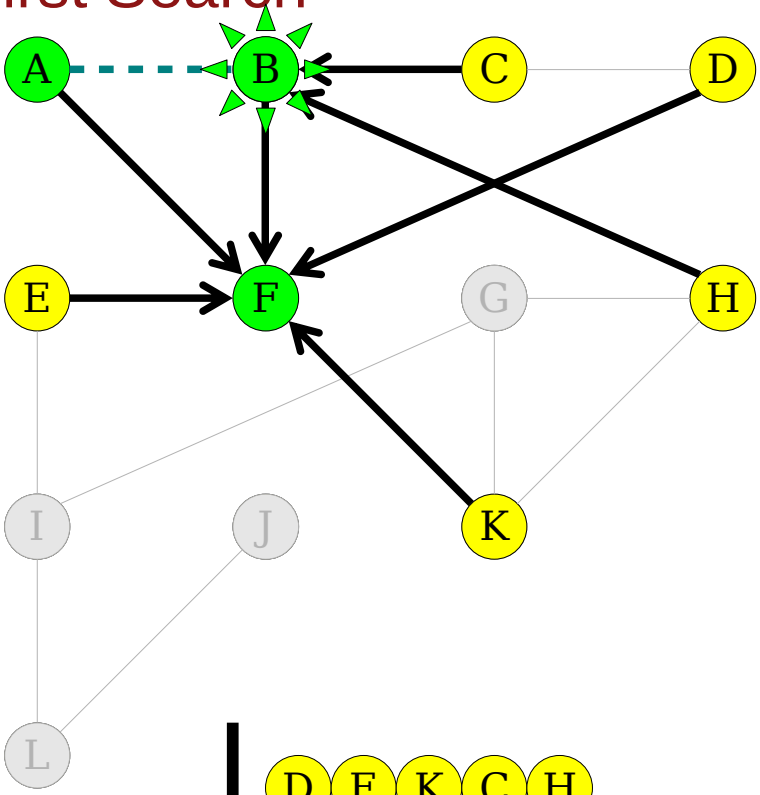
Breadth-First Search



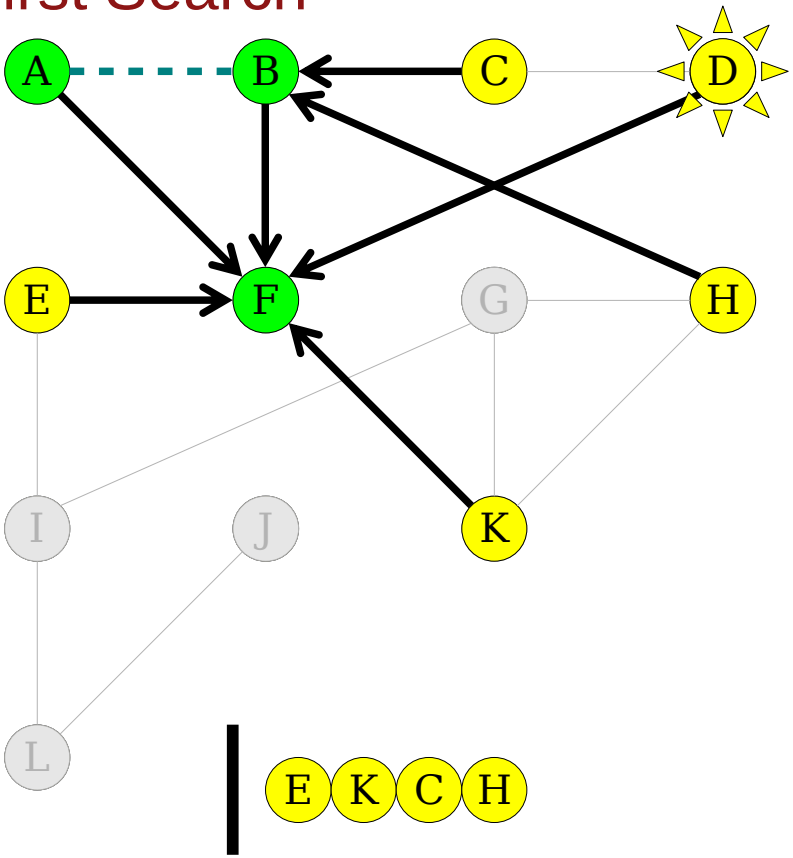
Breadth-First Search



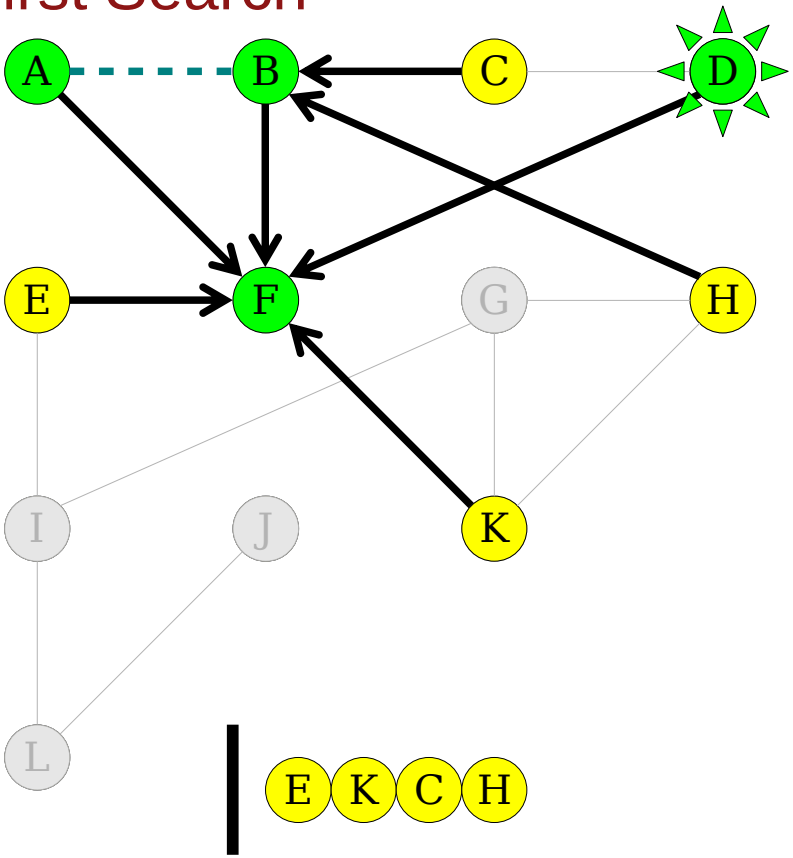
Breadth-First Search



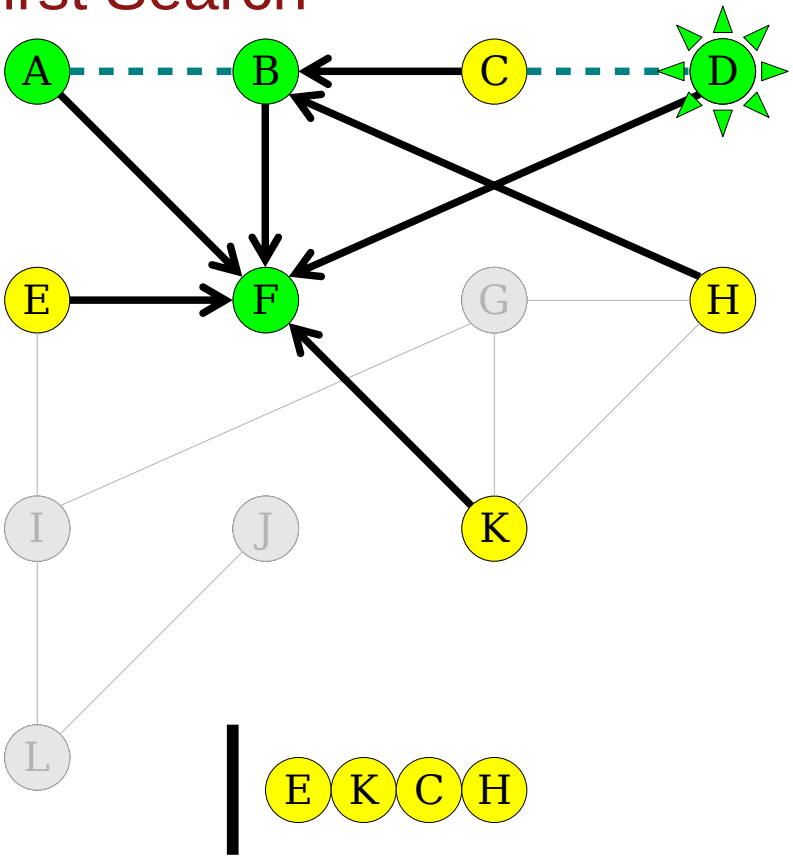
Breadth-First Search



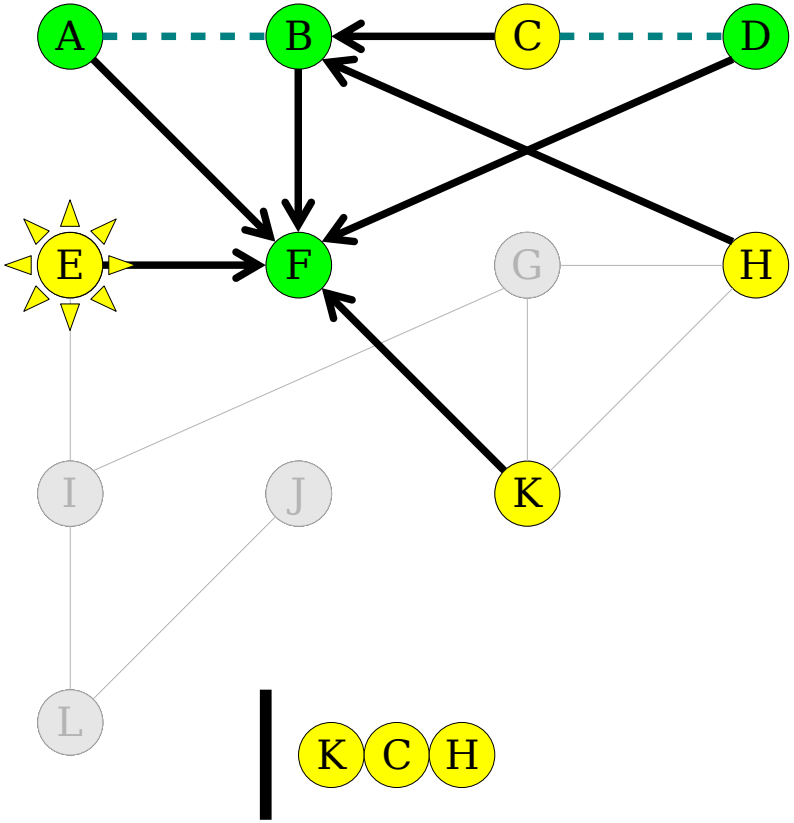
Breadth-First Search



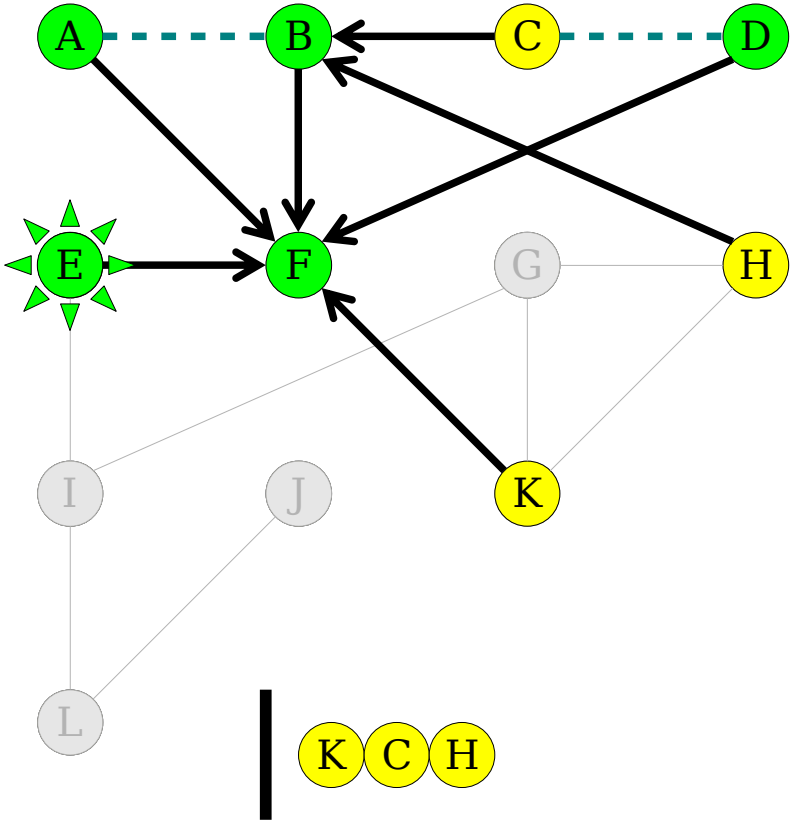
Breadth-First Search



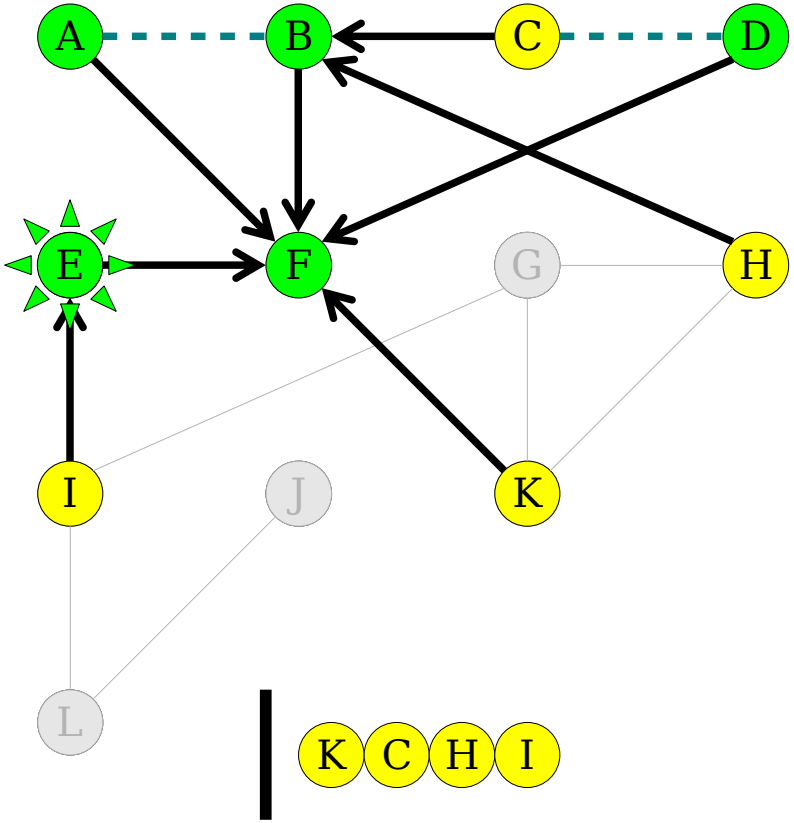
Breadth-First Search



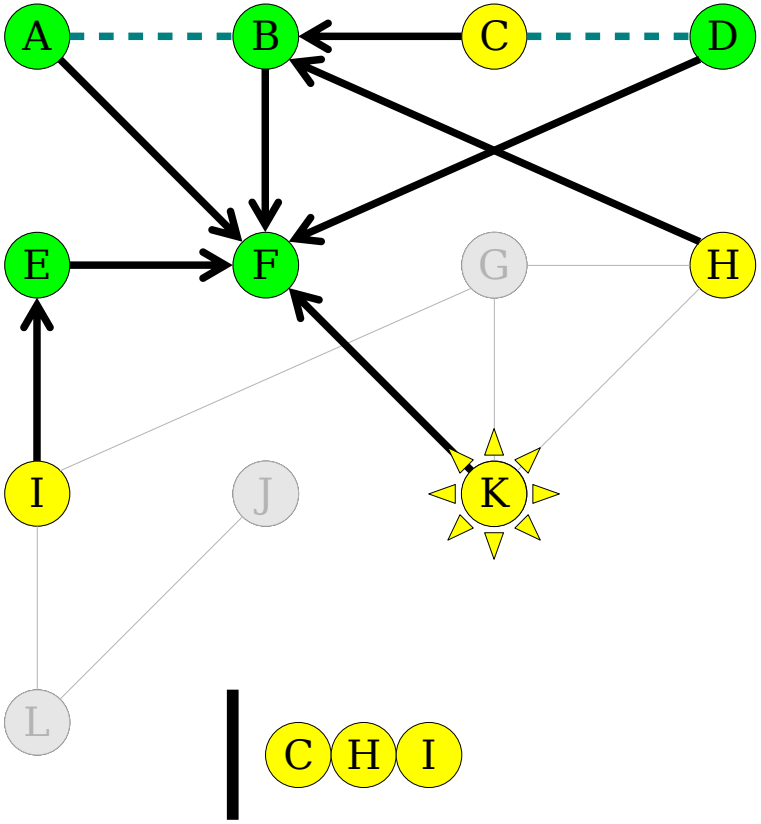
Breadth-First Search



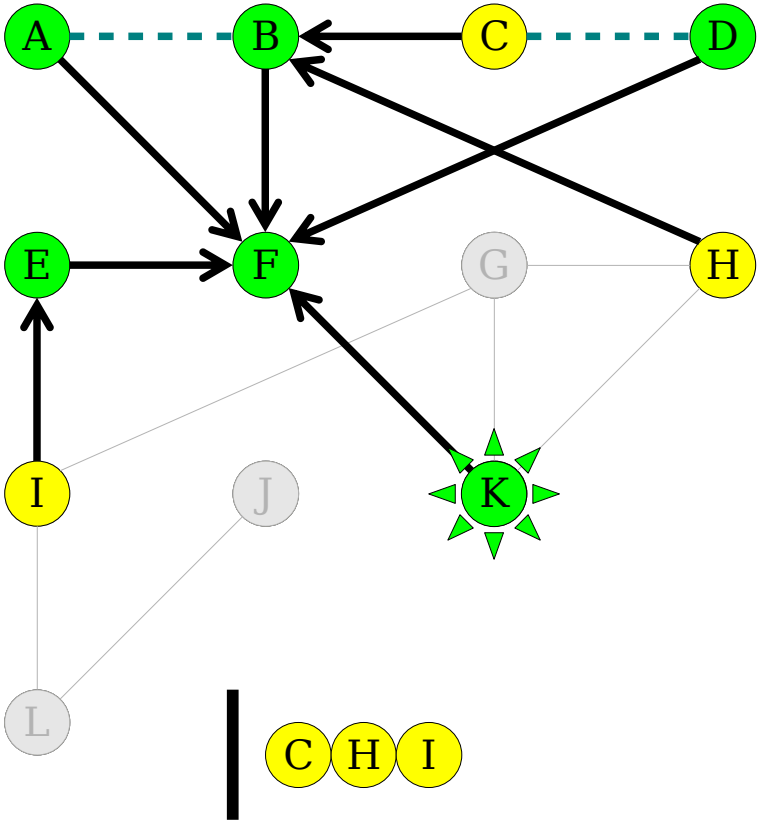
Breadth-First Search



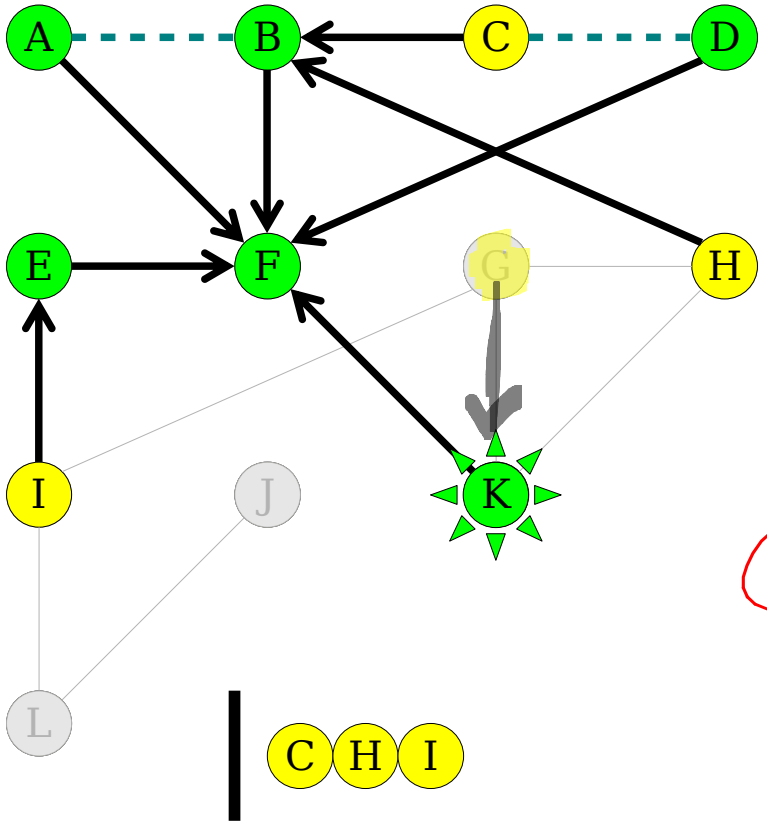
Breadth-First Search



Breadth-First Search



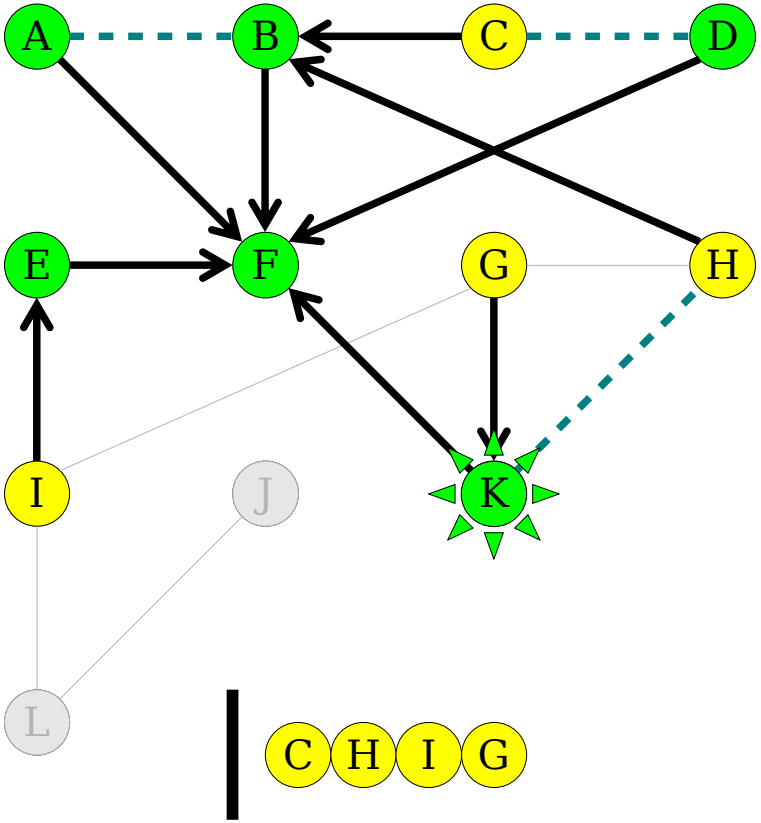
Breadth-First Search



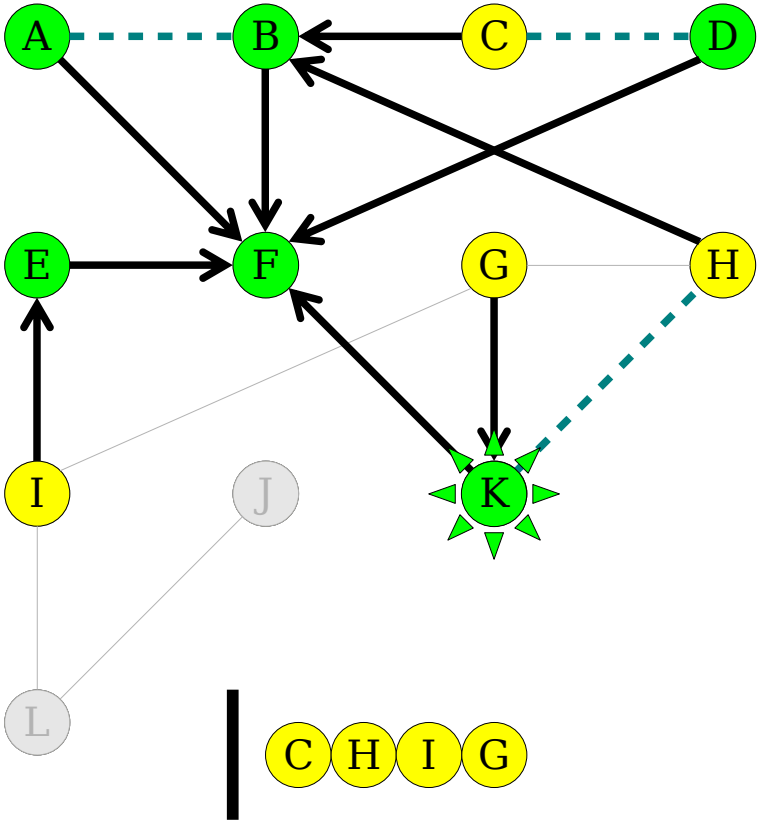
You predict the next slide!

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

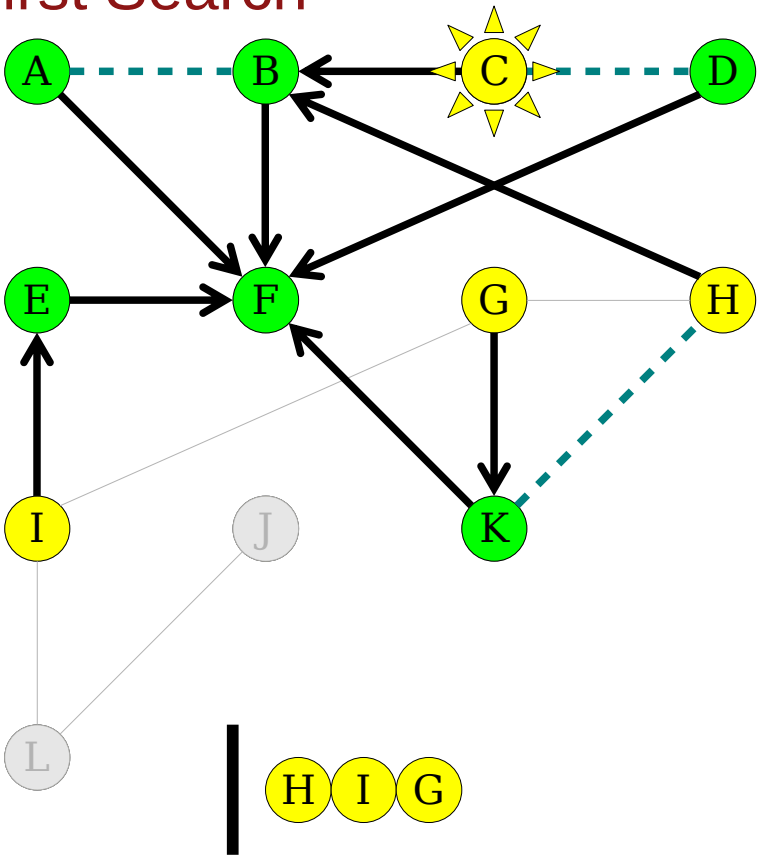
Breadth-First Search



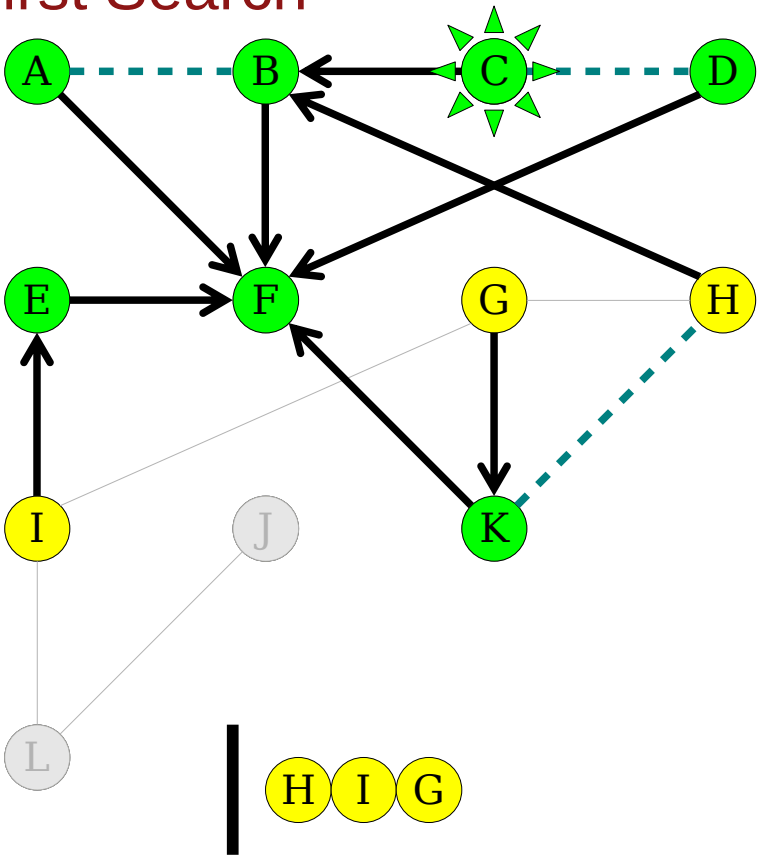
Breadth-First Search



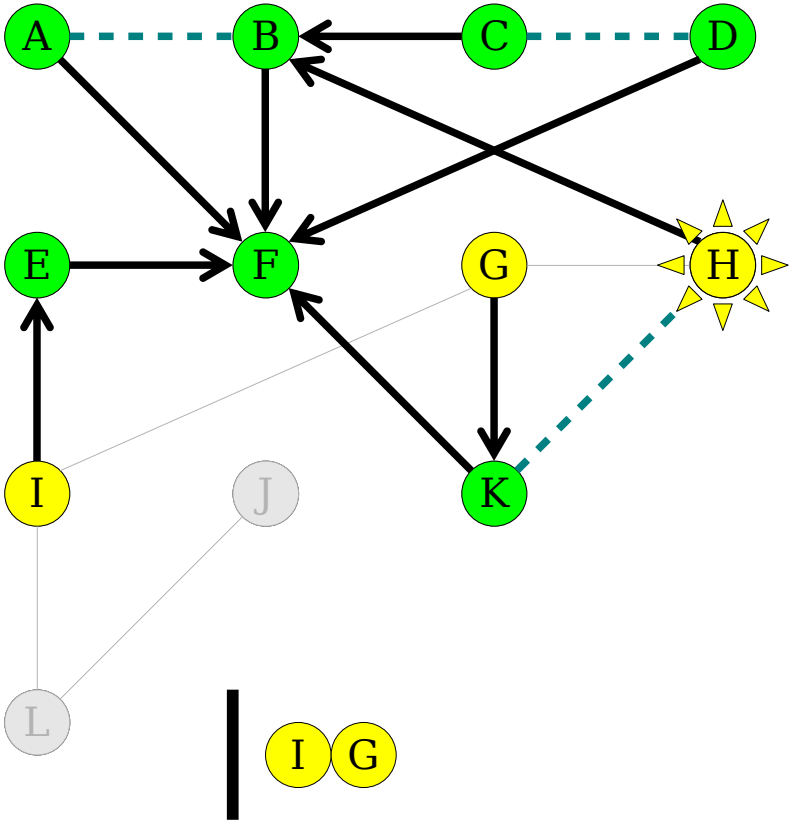
Breadth-First Search



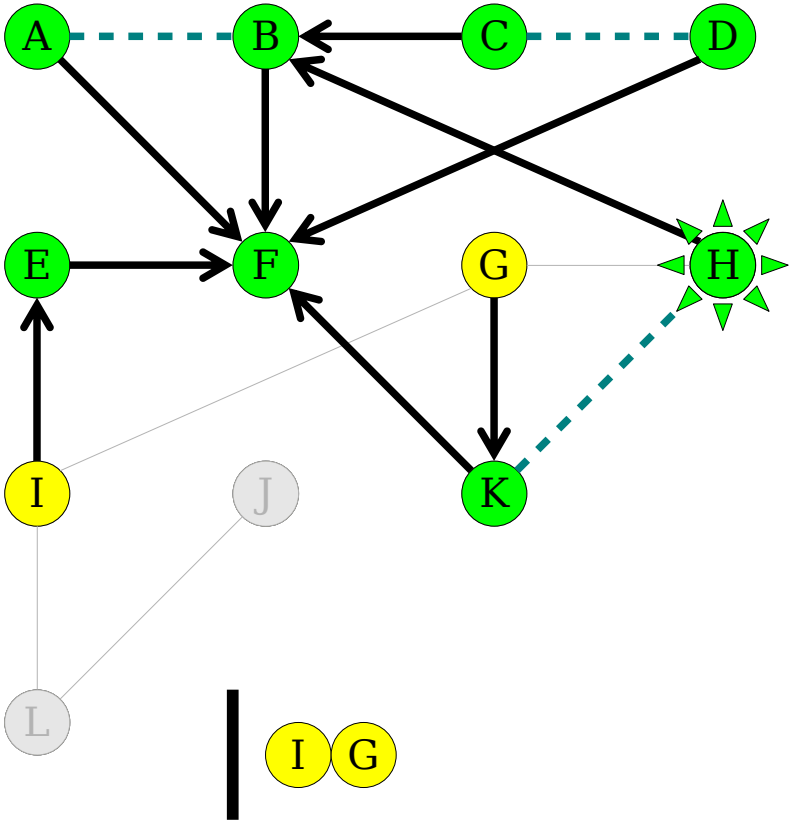
Breadth-First Search



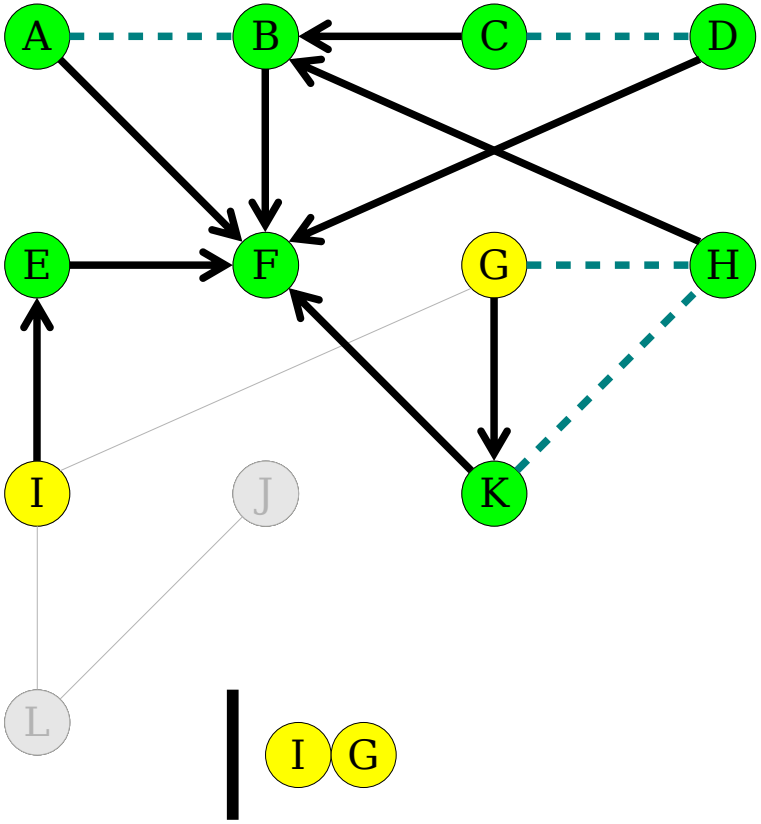
Breadth-First Search



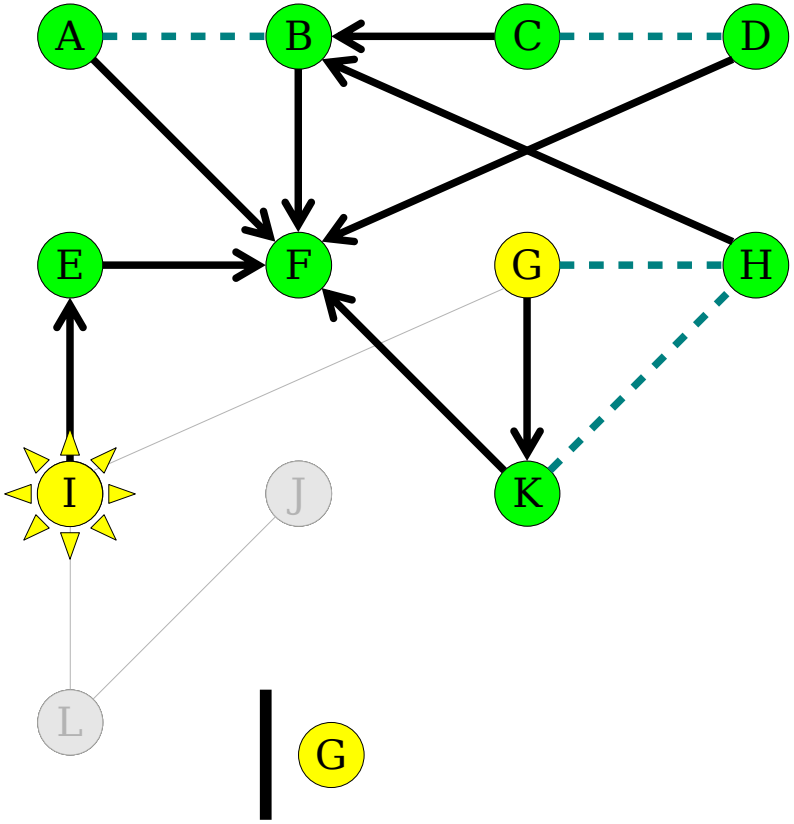
Breadth-First Search



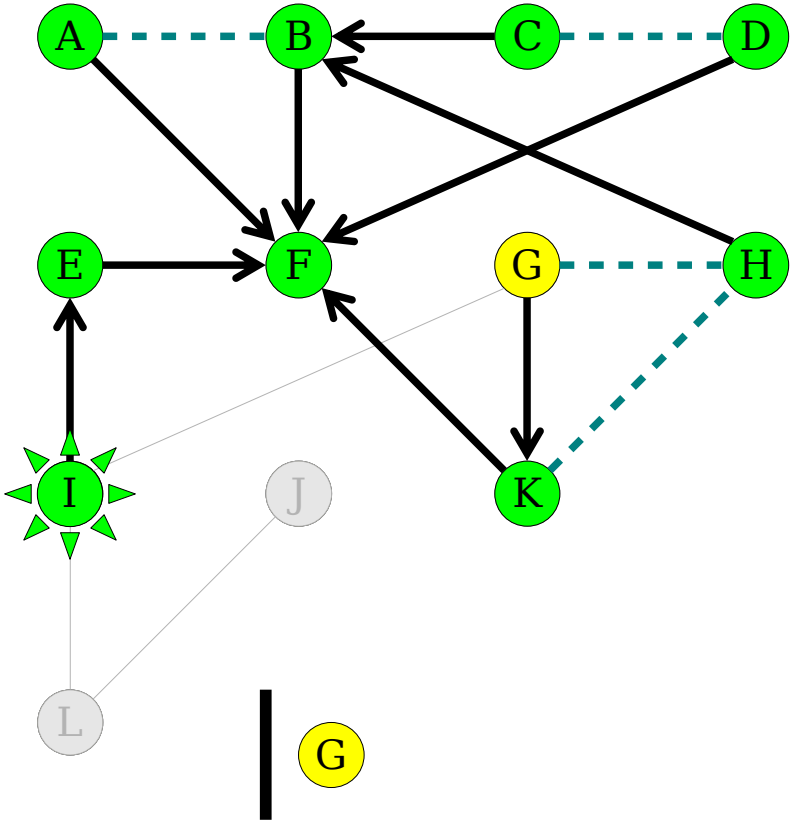
Breadth-First Search



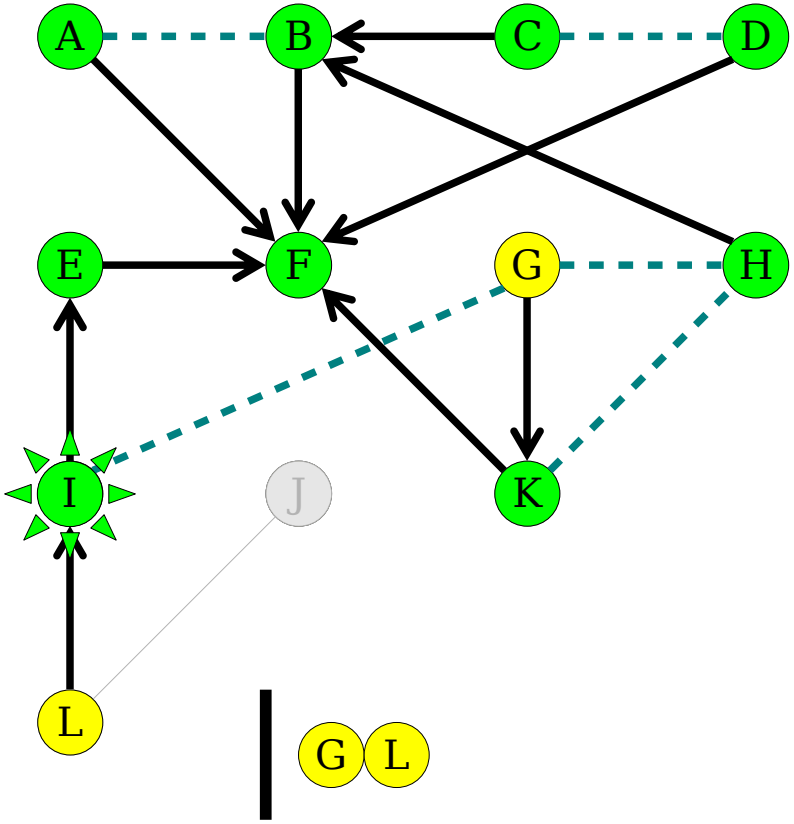
Breadth-First Search



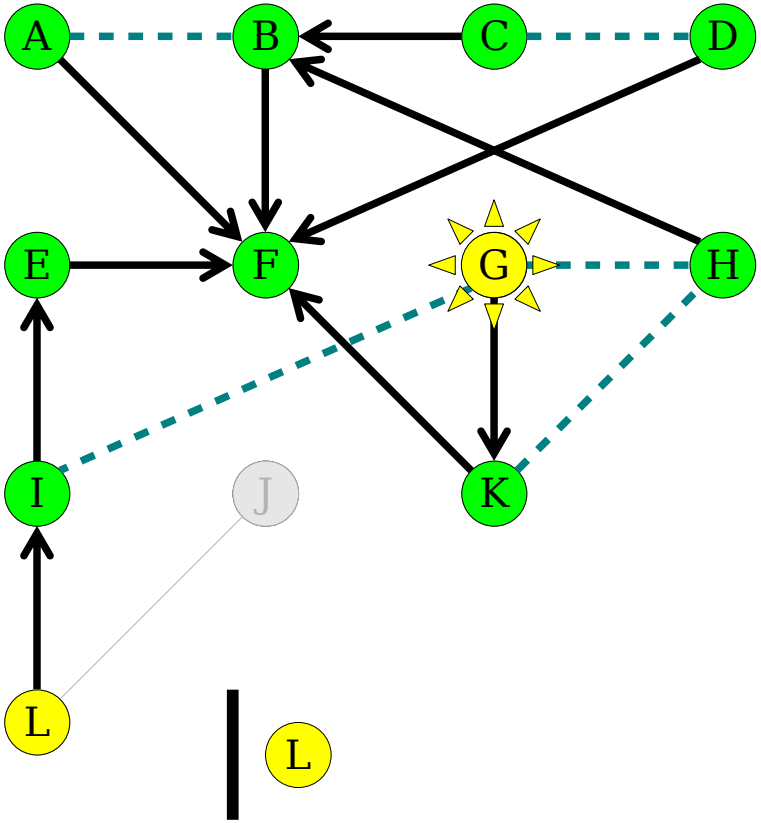
Breadth-First Search



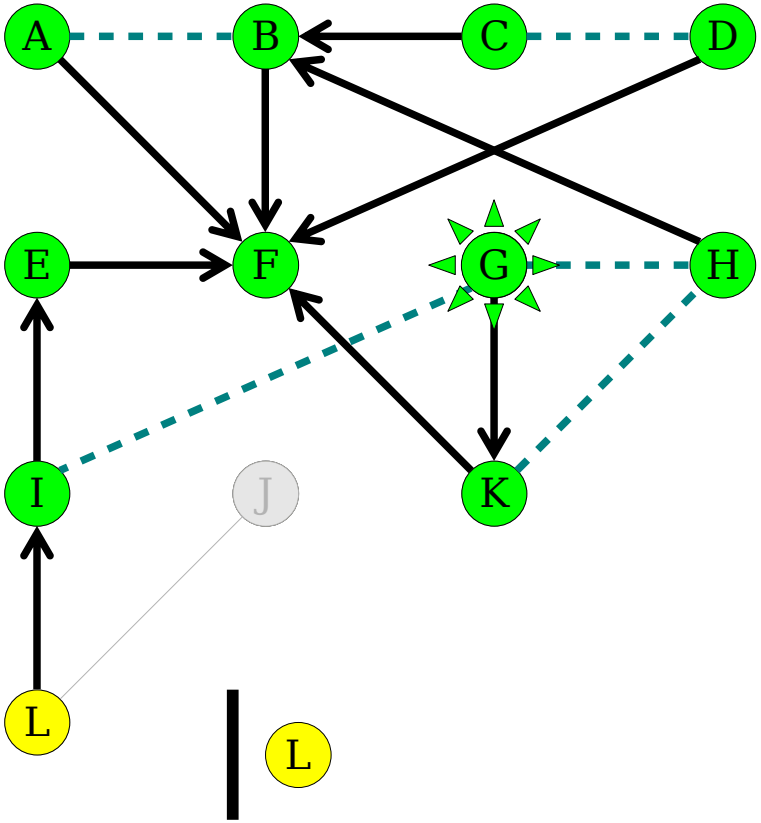
Breadth-First Search



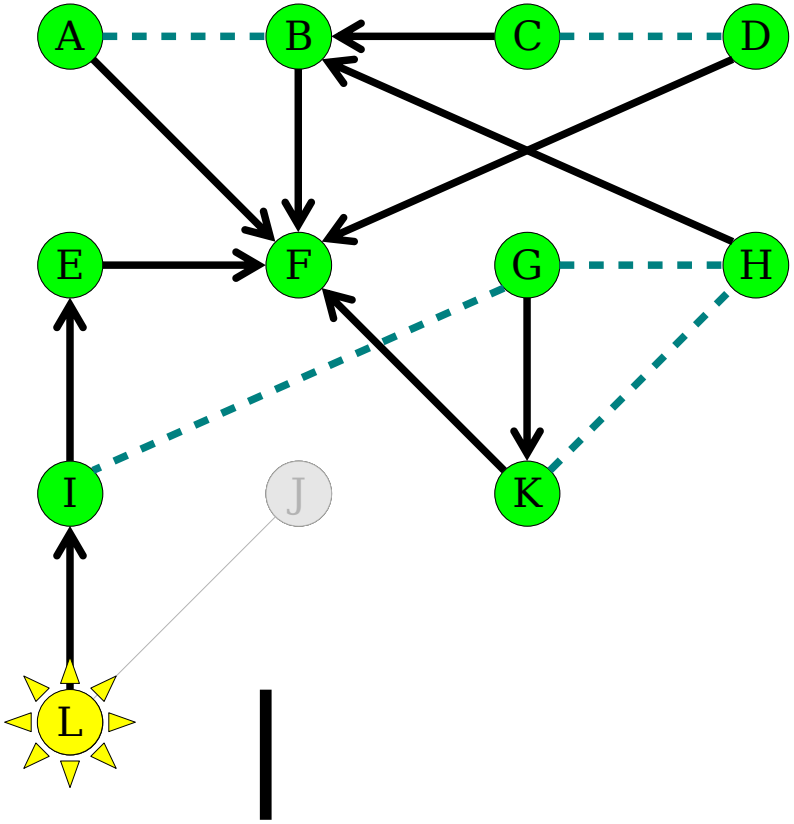
Breadth-First Search



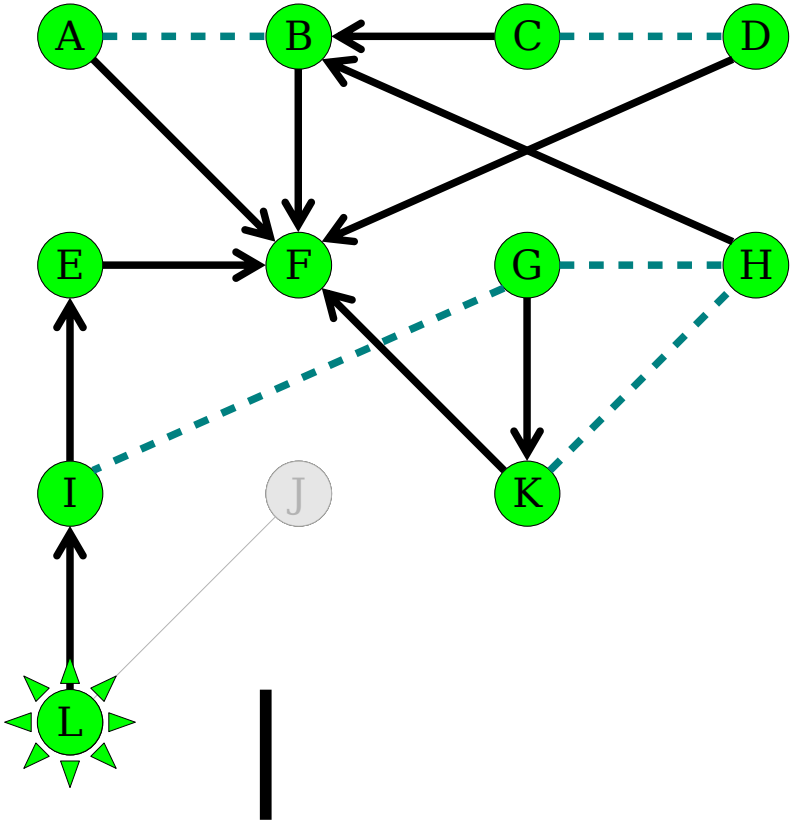
Breadth-First Search



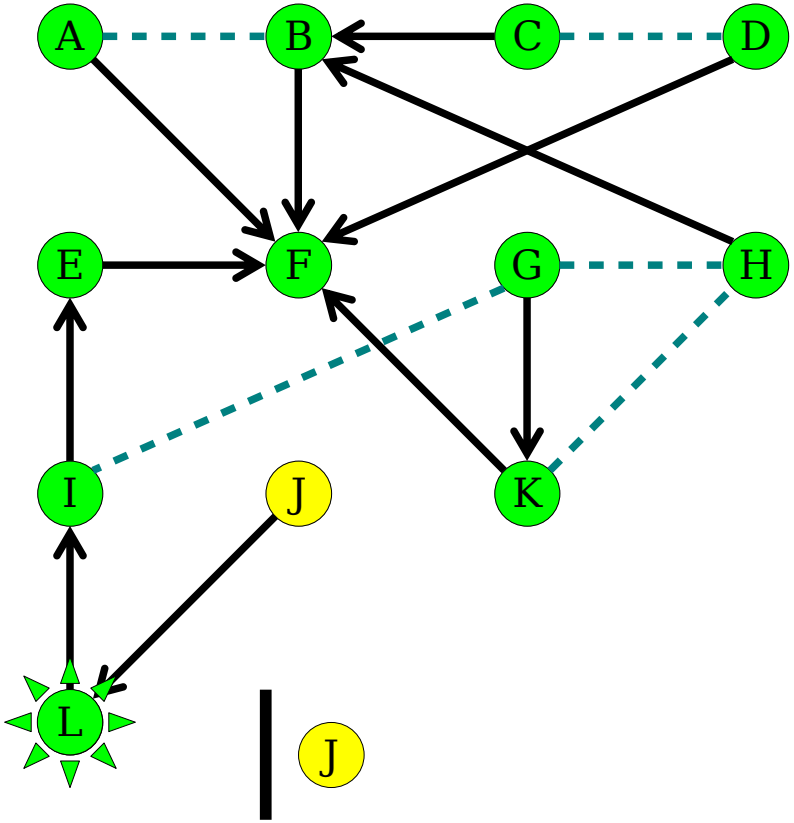
Breadth-First Search



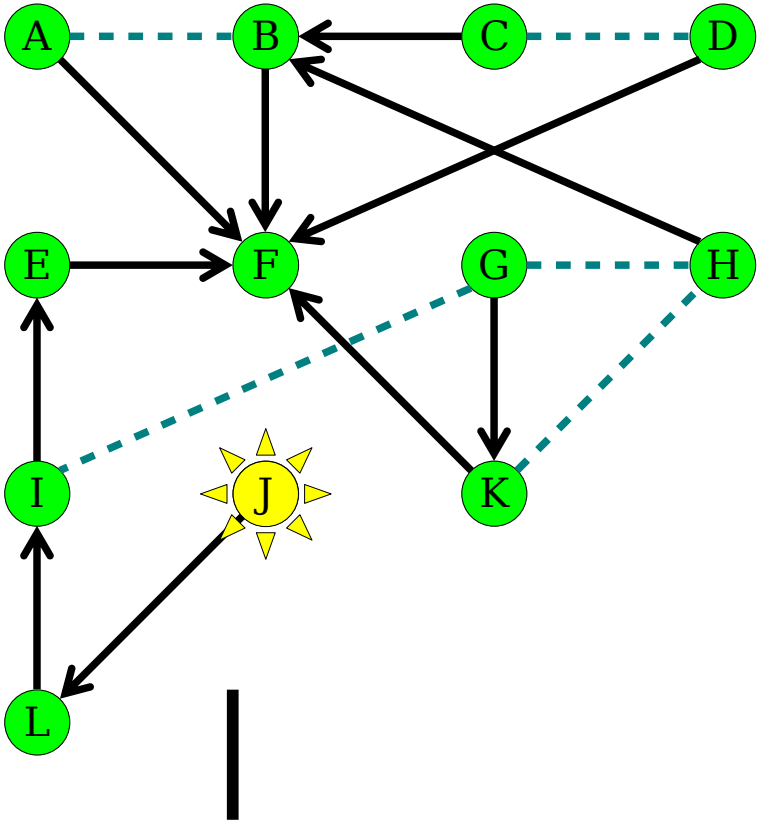
Breadth-First Search



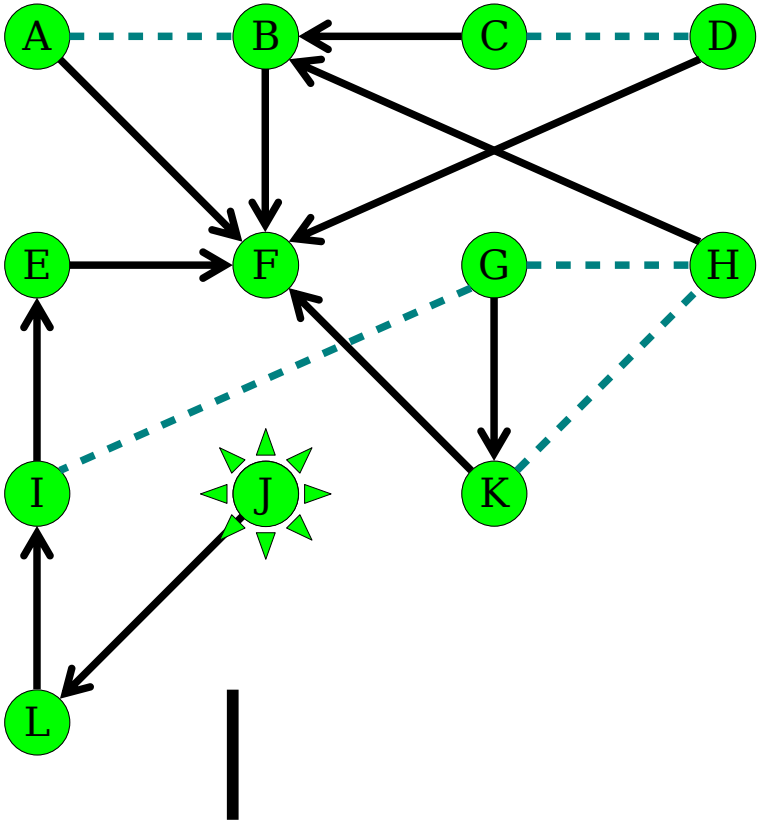
Breadth-First Search



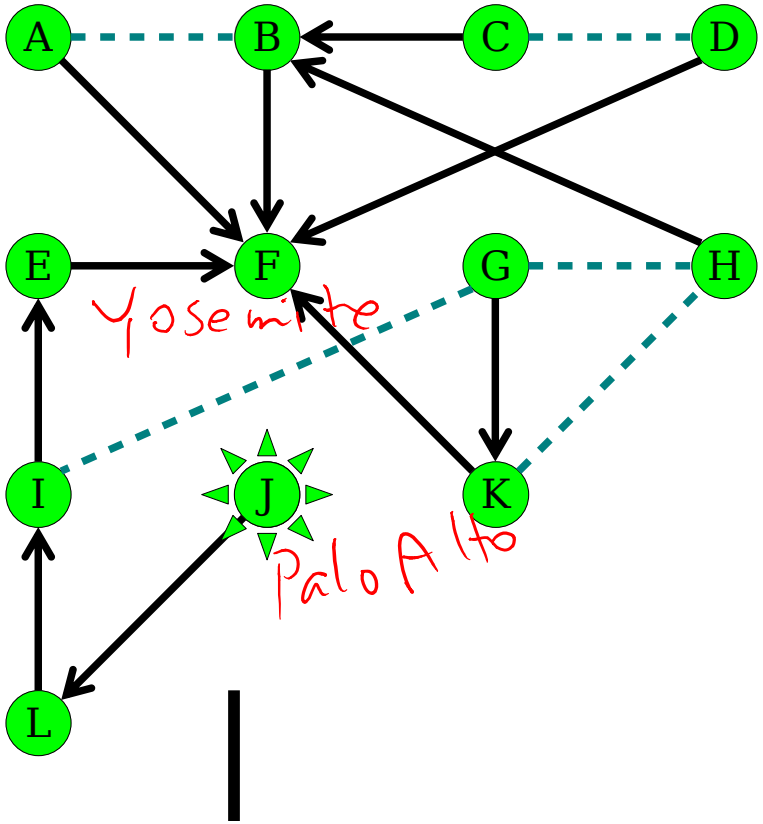
Breadth-First Search



Breadth-First Search



Breadth-First Search



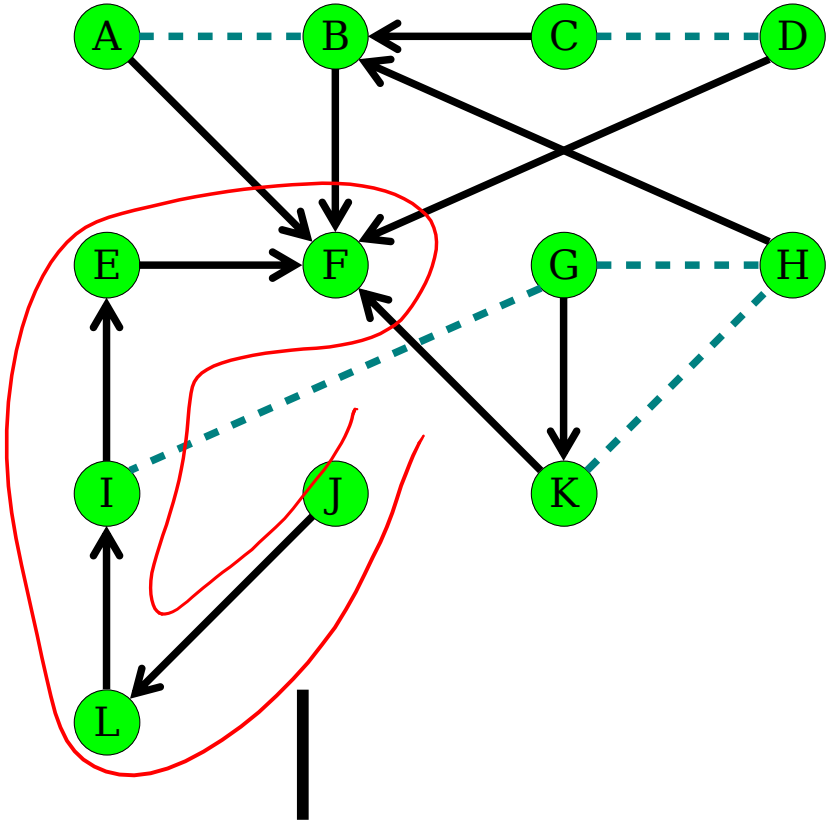
Done!

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

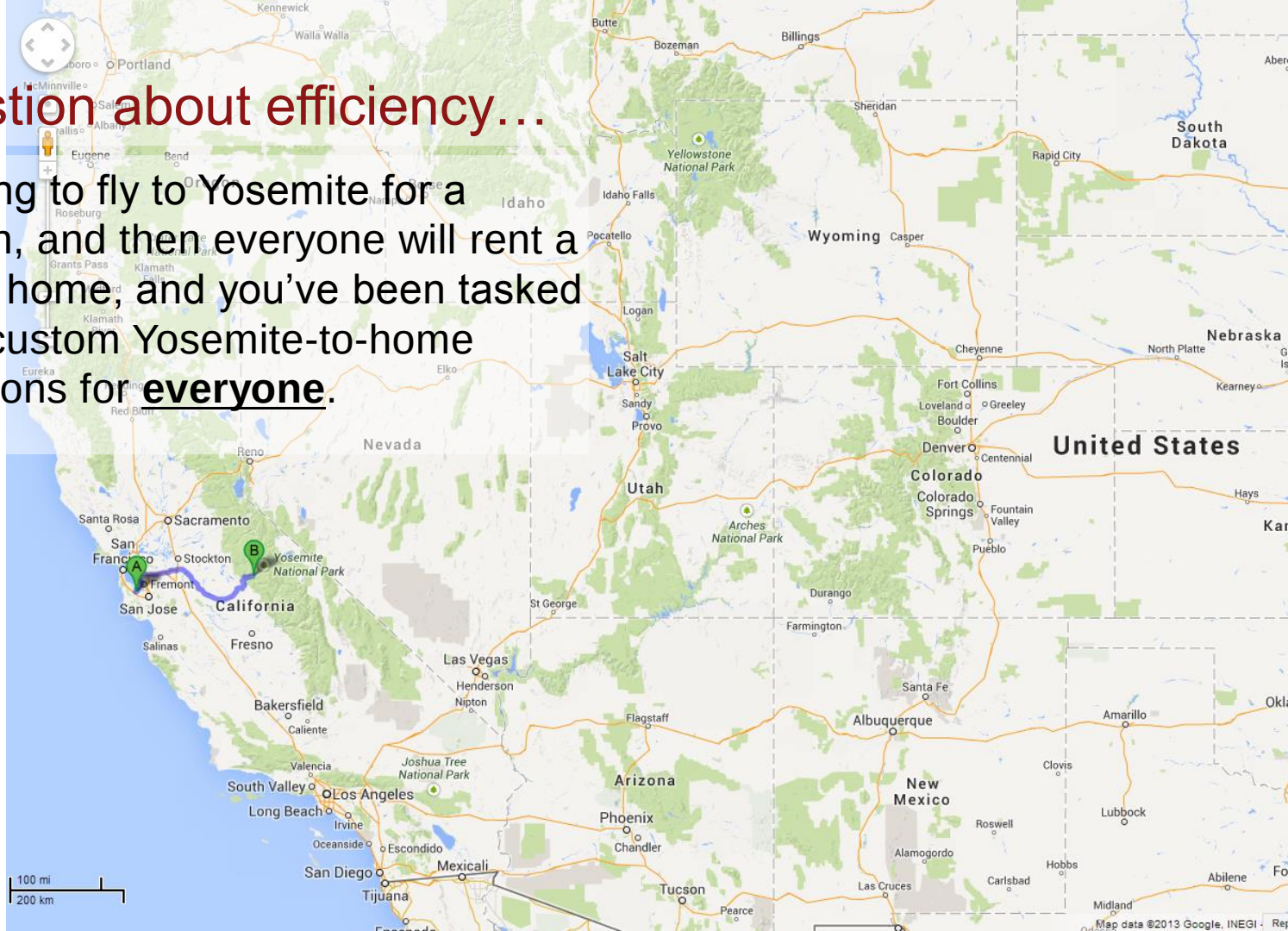
(3) If you follow the parent pointers from the desired end point, you will get back to the start point, and it will be the shortest way to do that

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…

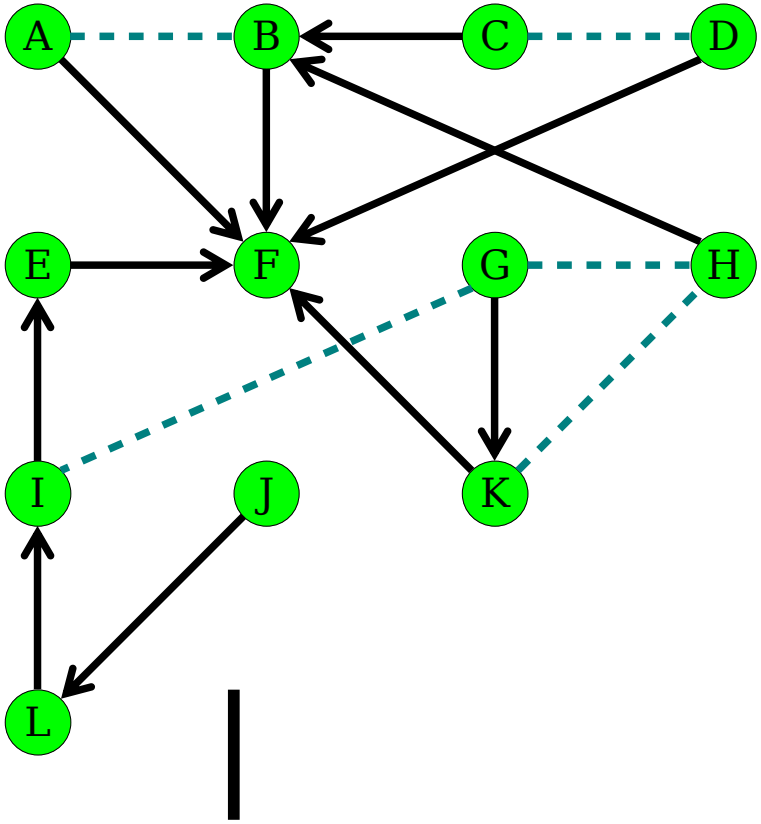
You calculated the shortest path for yourself to return home from the reunion (Yosemite to Palo Alto) and let's just say that it took time **X** = O((|E| + |V|)log|V|)

- 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 and 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!!***

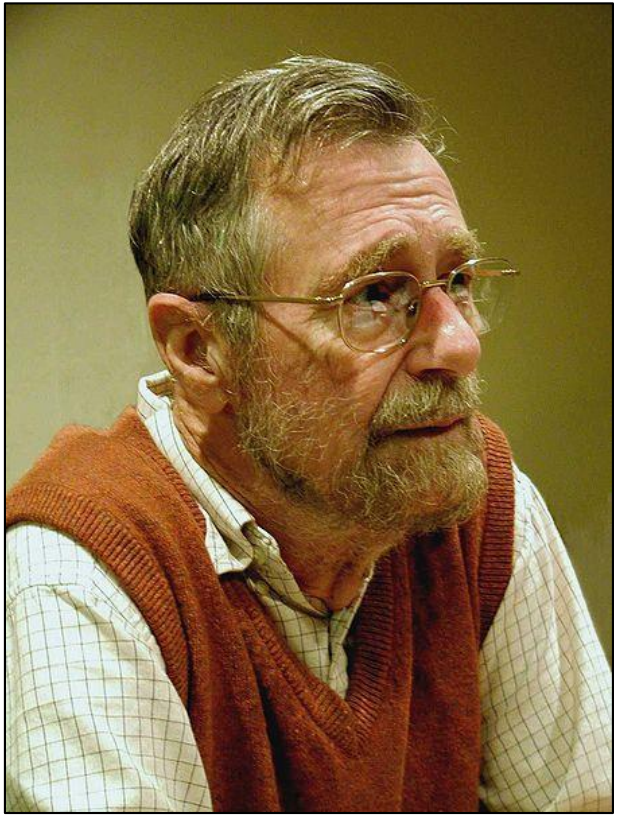
Dijkstra's Shortest Paths

(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-

EWD1036-0

On the cruelty of really teaching computing science

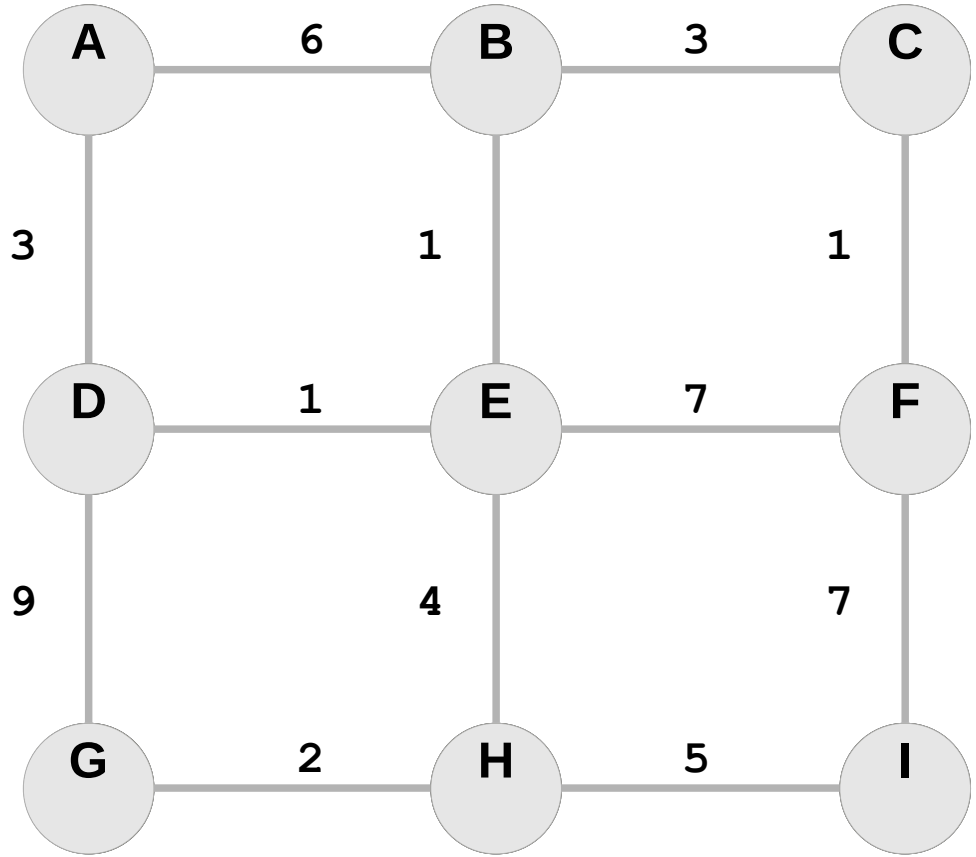
The second part of this talk pursues some of the scientific and educational consequences of the assumption that computers represent a radical novelty. In order to give this assumption clear contents, we have to be much more precise as to what we mean in this context by the adjective "radical". We shall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

The Shortest Path Problem

- Suppose that you have a graph representing different locations
- Each edge has an associated *cost* (or *weight*, *length*, etc)
- We'll assume costs are nonnegative*
- **Goal: Find the least-cost (or lowest-weight, lowest-length, etc) path from some node *u* to a node *v***

* else use the Bellman–Ford algorithm


A
6
B
3
C
3
1
1
D
1
E
7
F
9
4
7
G
2
H
5
I

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***.