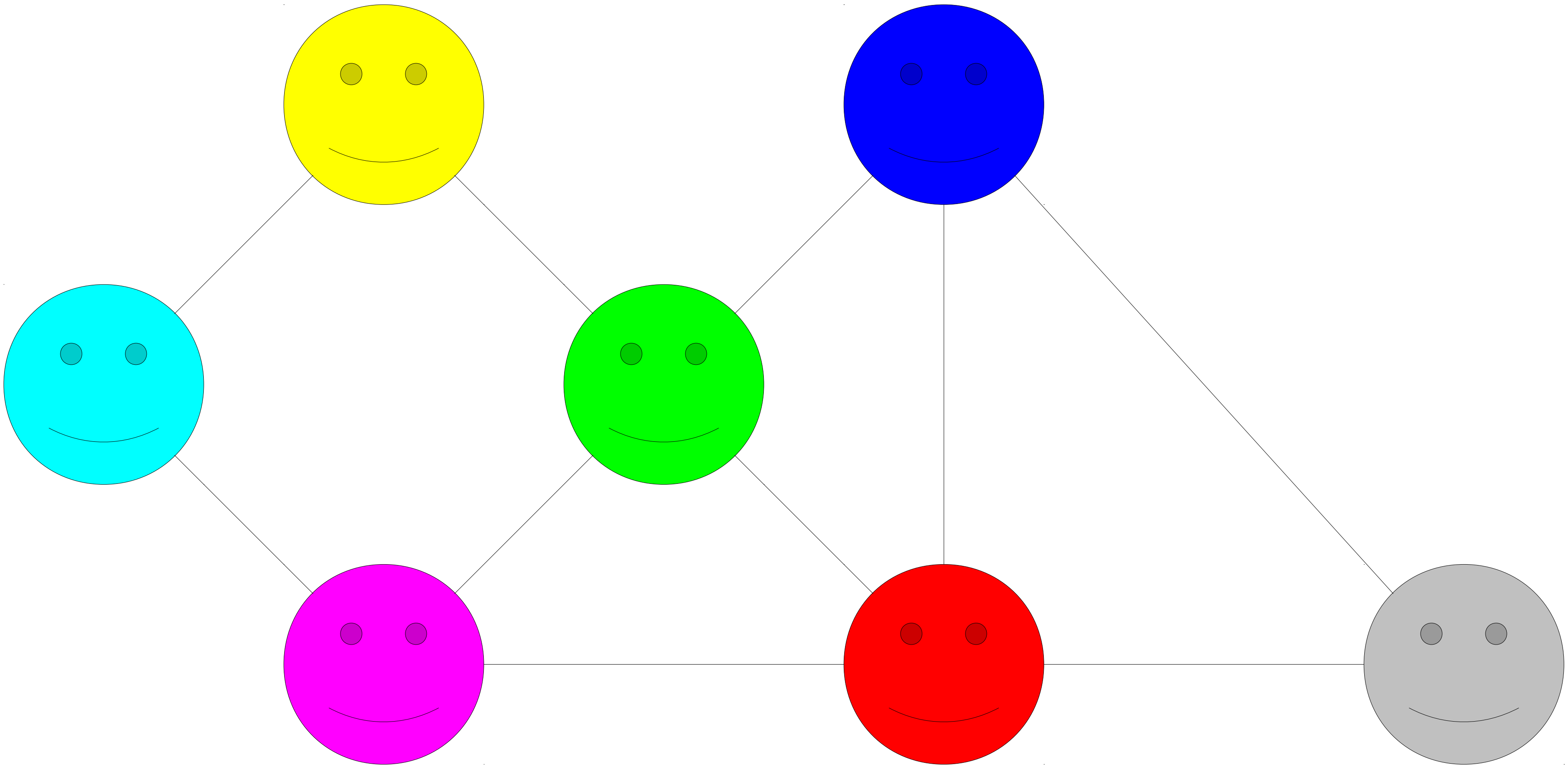
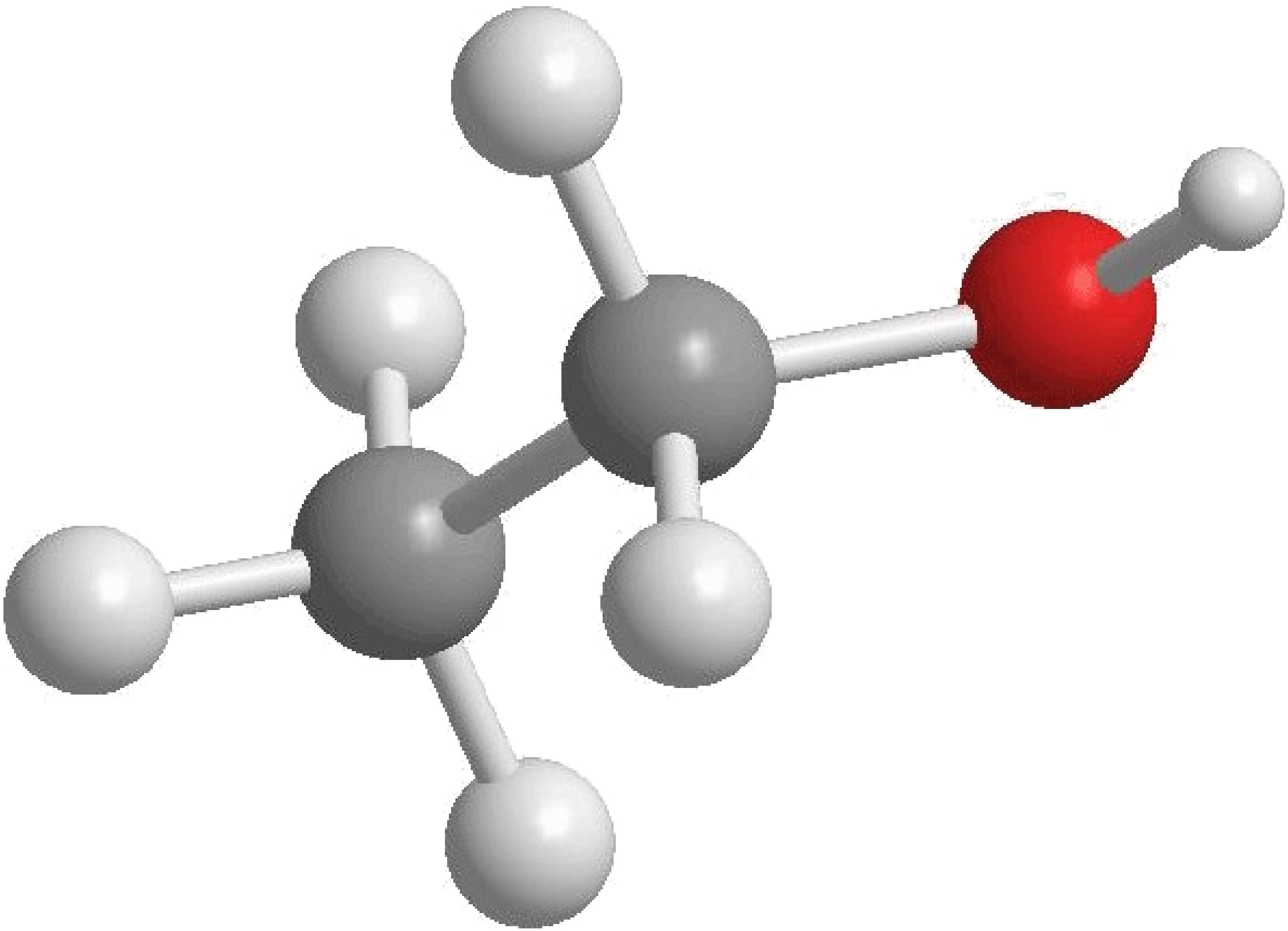
Graphs

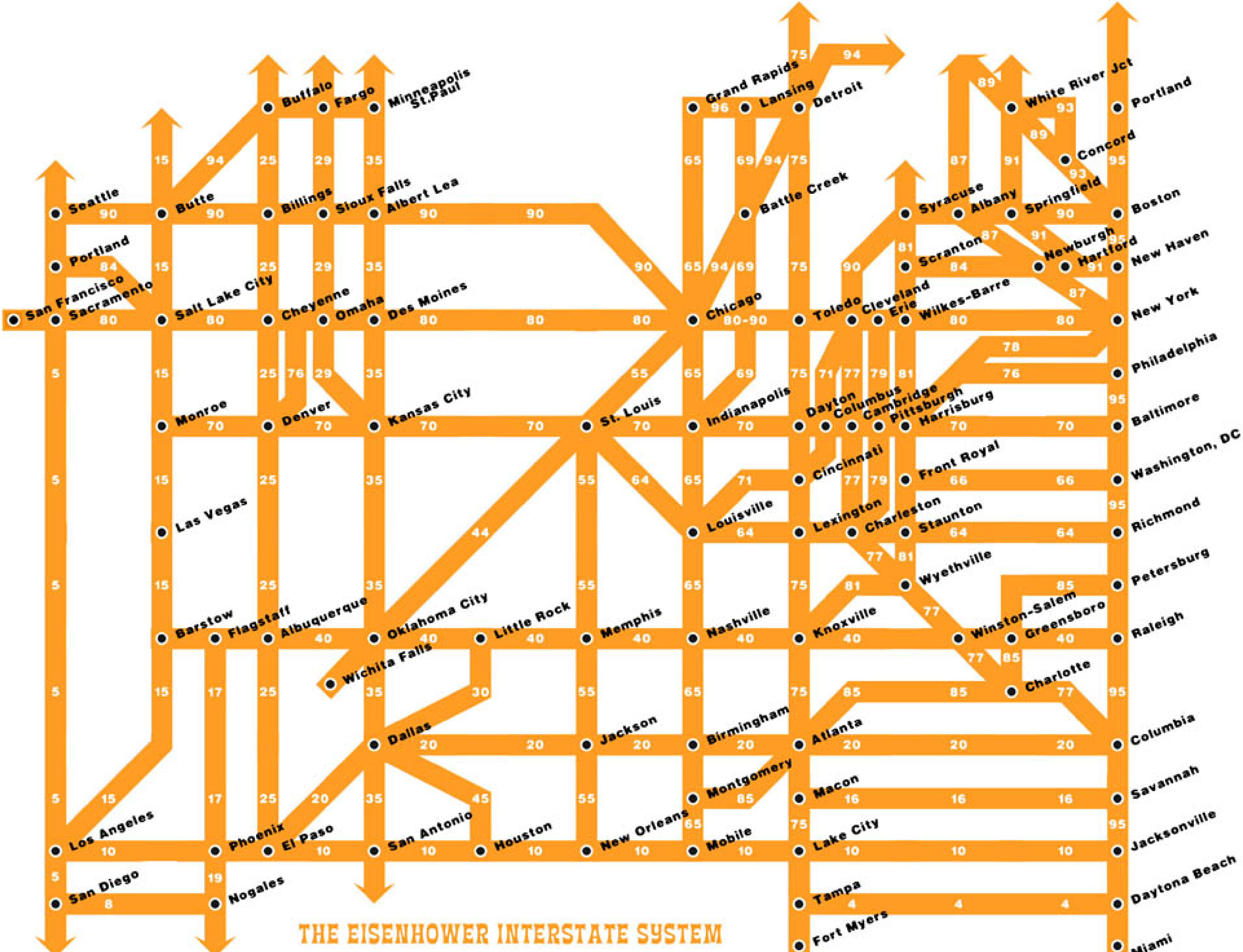
A Social Network



Chemical Bonds



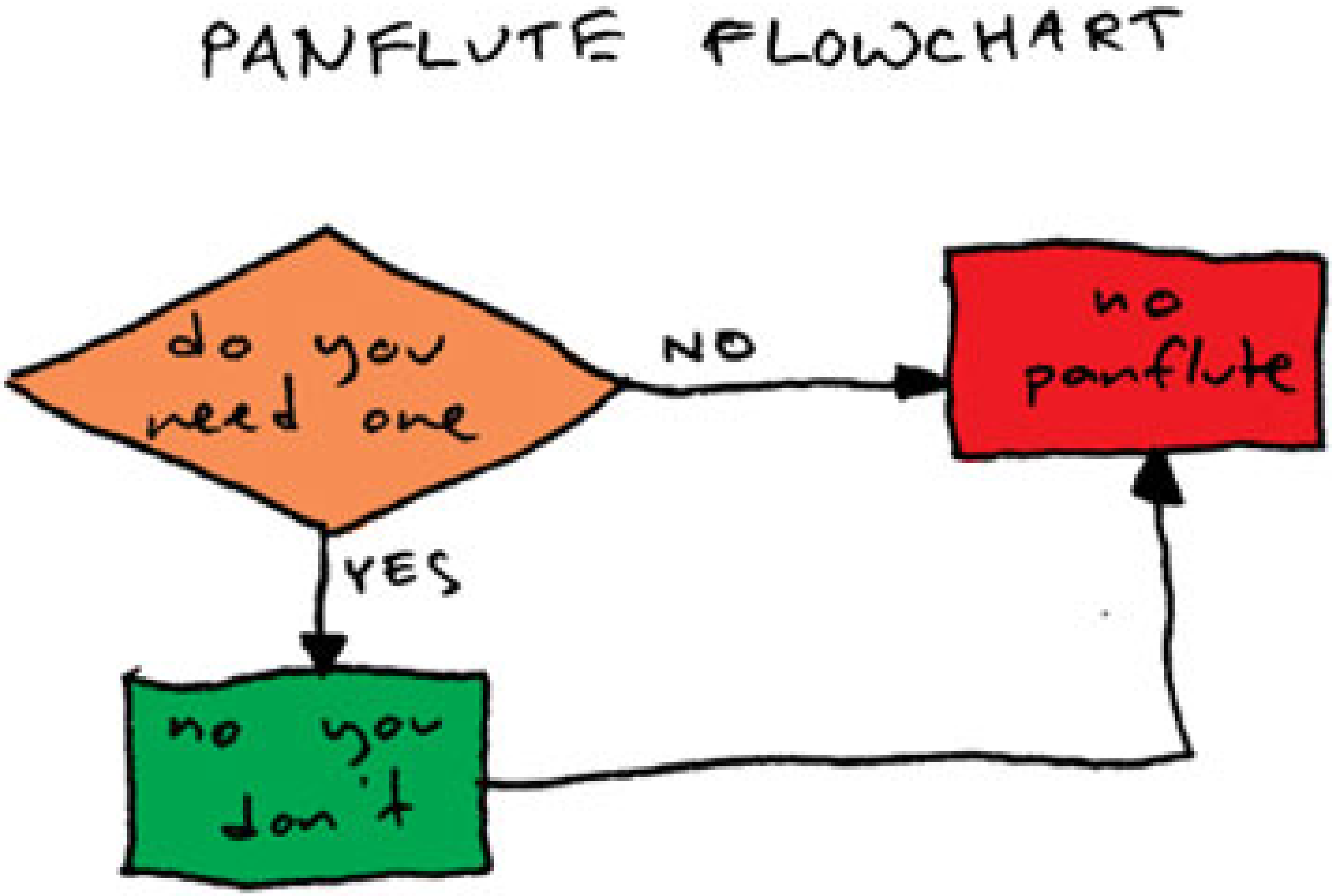
http://4.bp.blogspot.com/-xCtBJ8lKHqA/Tjm0BONWBRI/AAAAAAAAAK4/-mHrbAUOHHg/s1600/Ethanol2.gif



THE EISENHOWER INTERSTATE SYSTEM

(simplified)

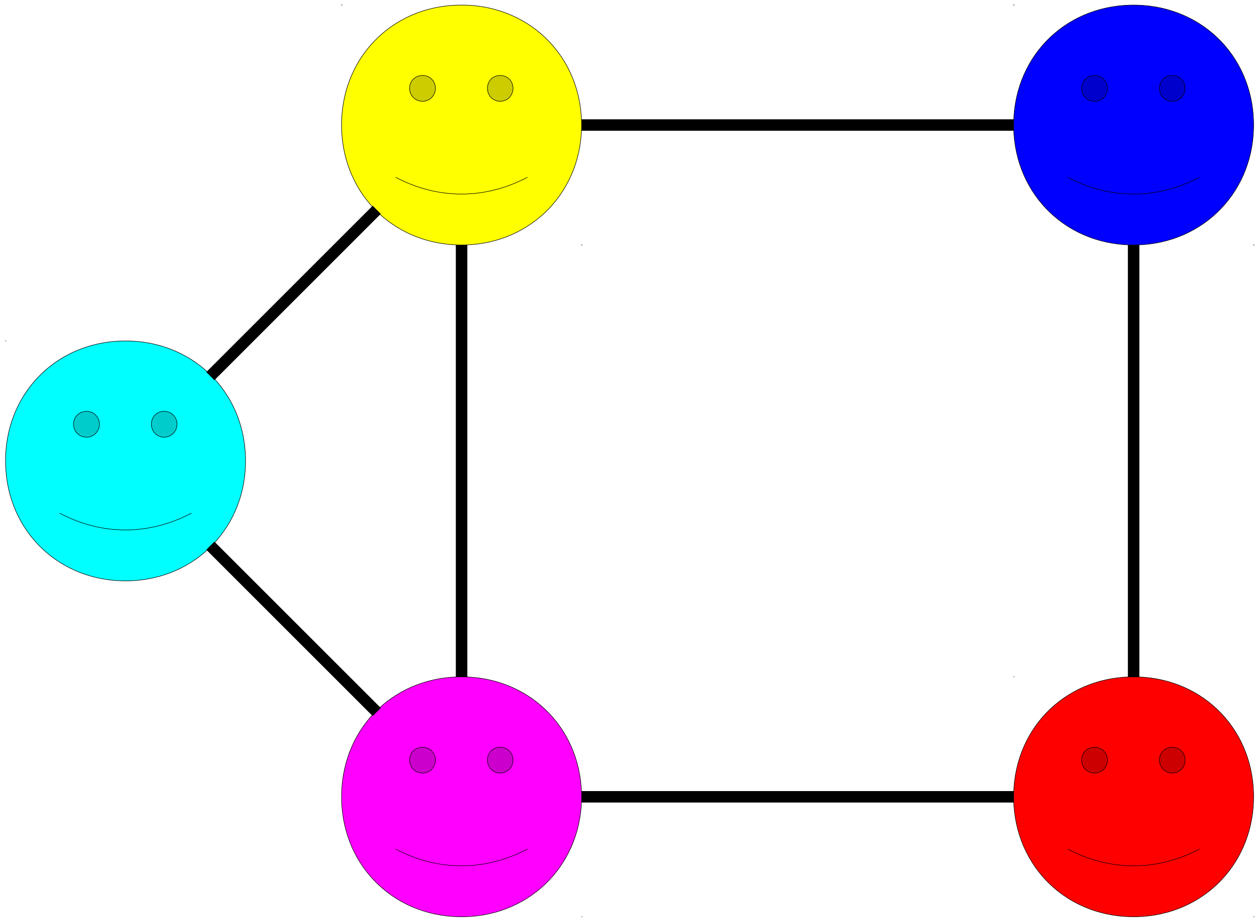
CHRIS YATES 2007


PANFLUTE FLOWCHART
do you need one
NO
no panflute
YES
no you don't

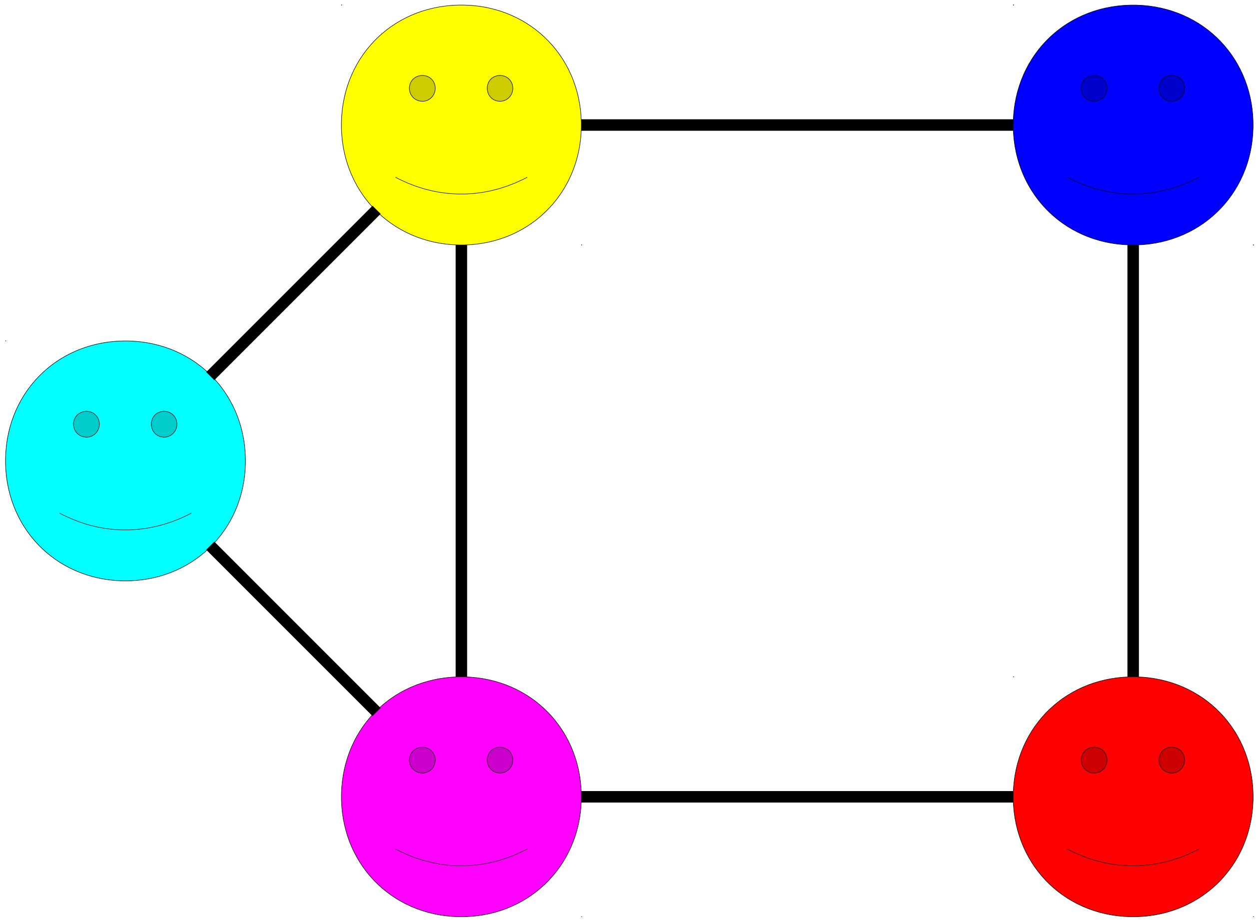
A ***graph*** is a mathematical structure for representing relationships.

A ***graph*** is a mathematical structure for representing relationships.

A ***graph*** is a mathematical structure for representing relationships.

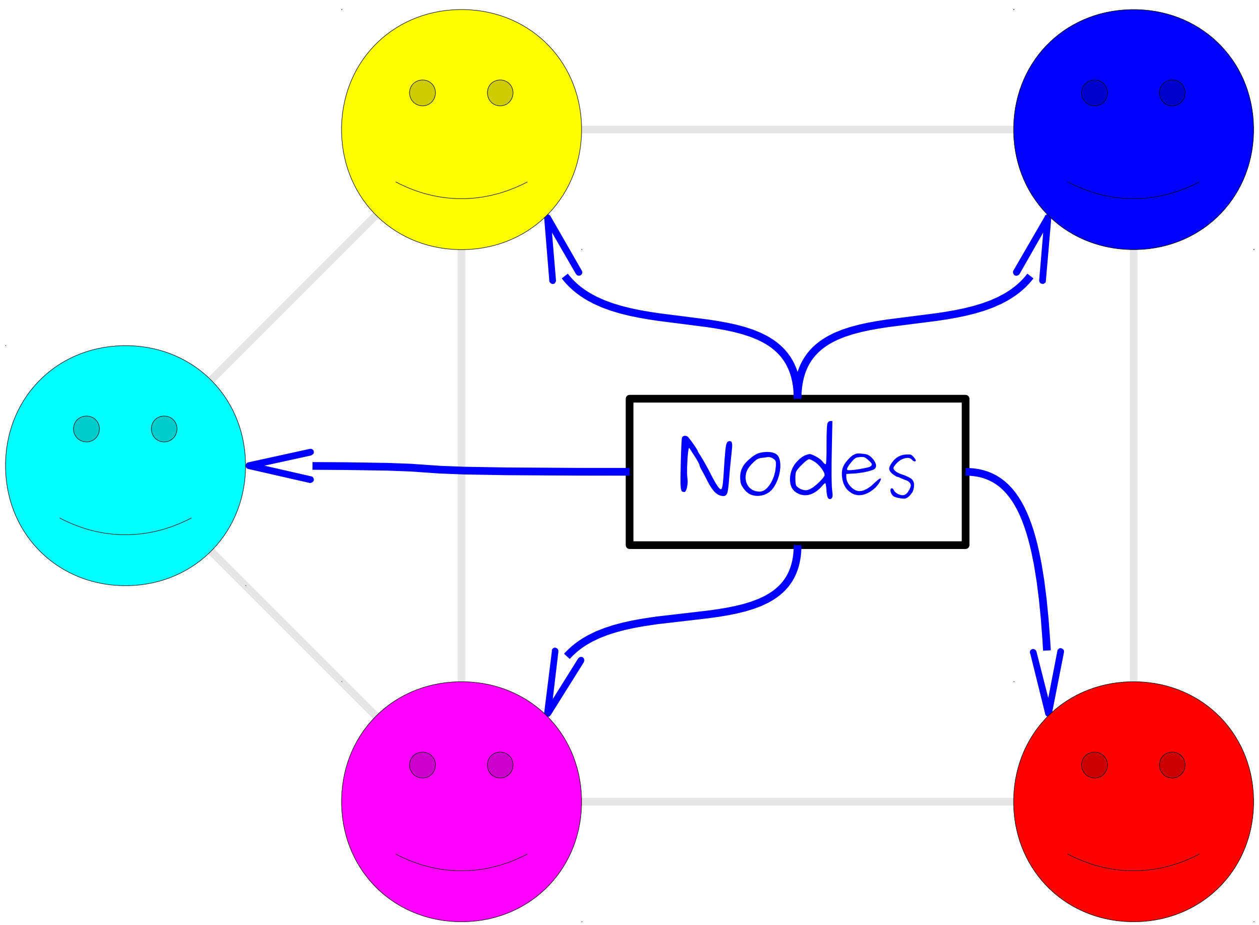


A ***graph*** is a mathematical structure for representing relationships.



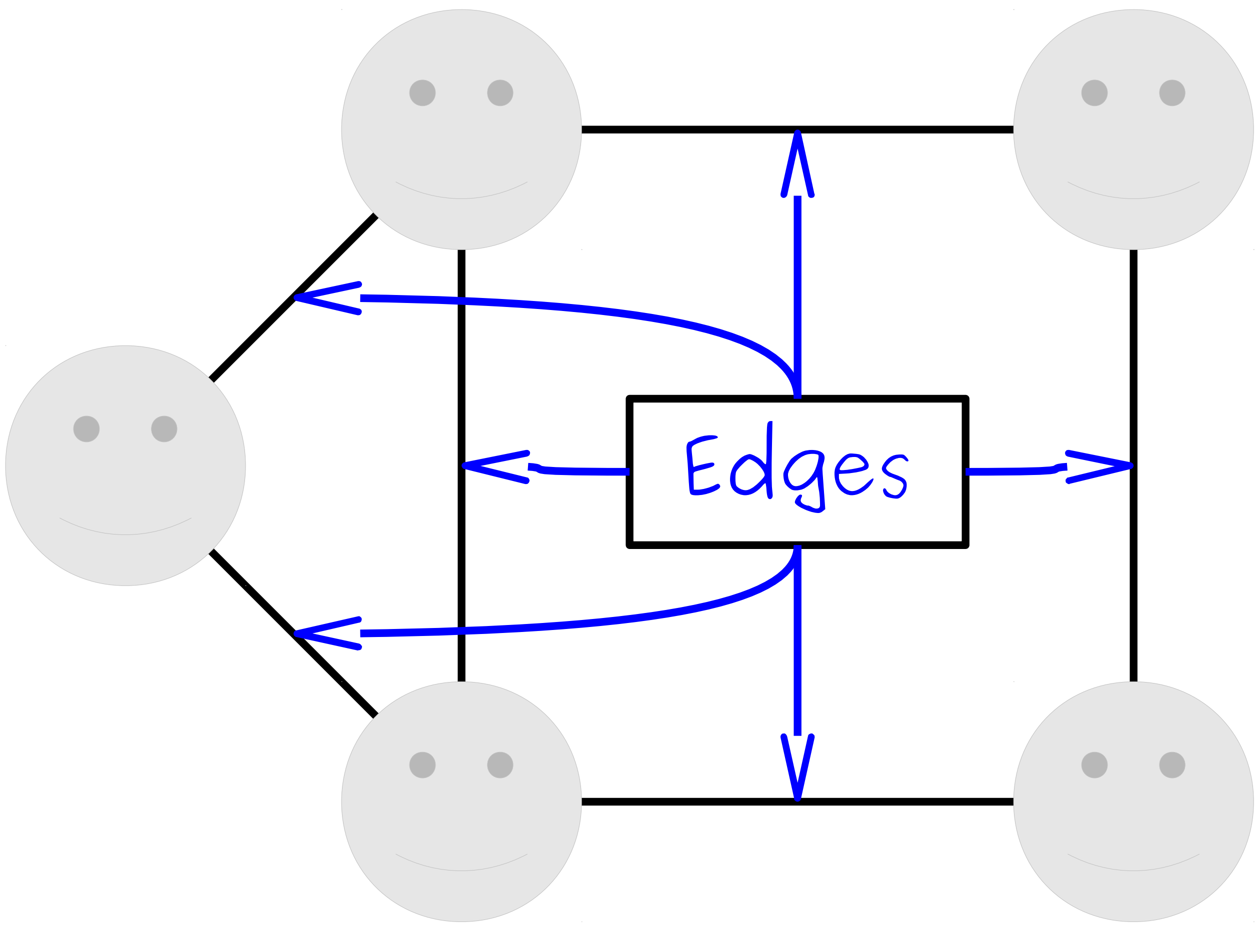
A graph consists of a set of ***nodes*** connected by ***edges***.

A ***graph*** is a mathematical structure for representing relationships.



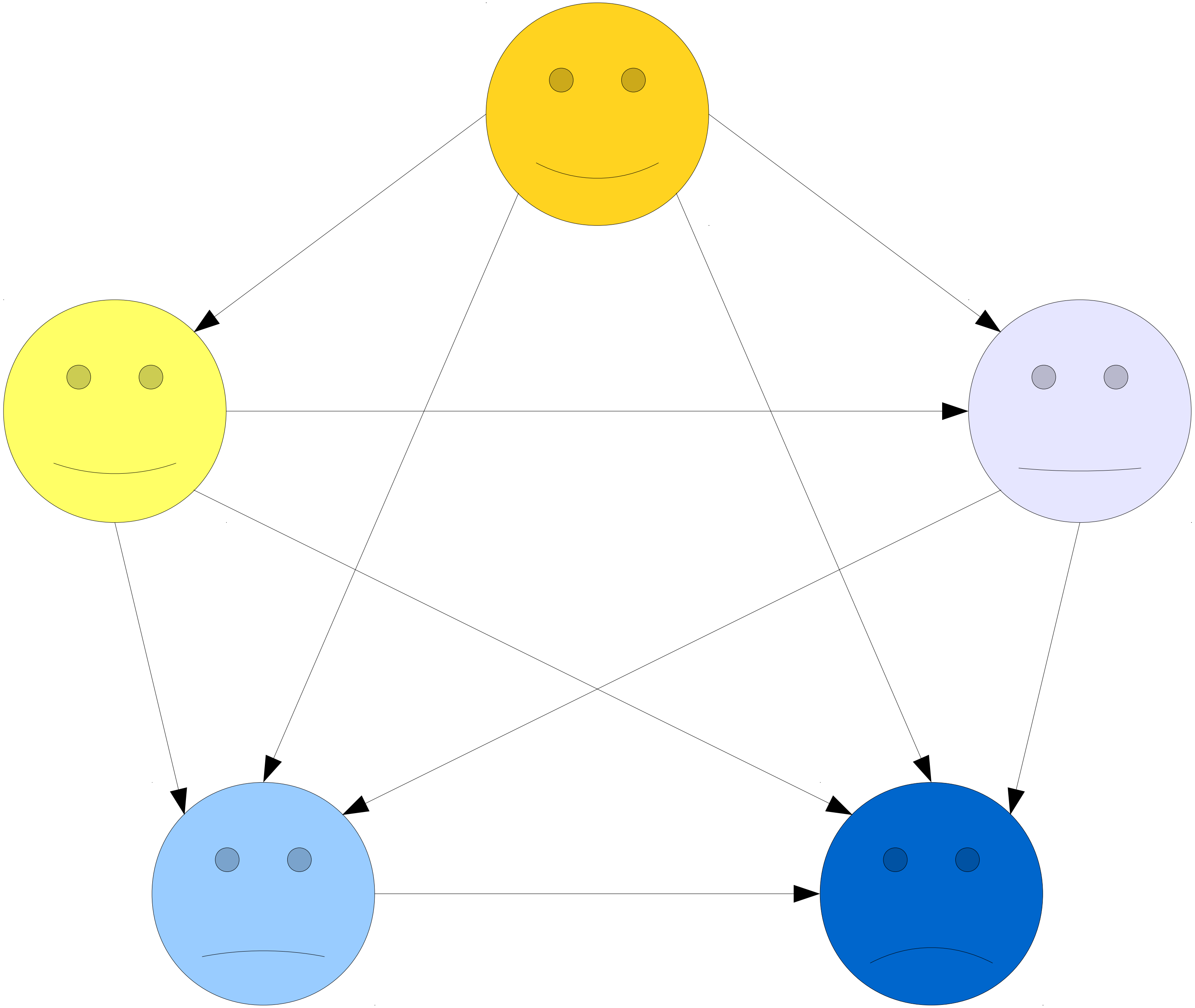
A graph consists of a set of ***nodes*** connected by ***edges***.

A ***graph*** is a mathematical structure for representing relationships.

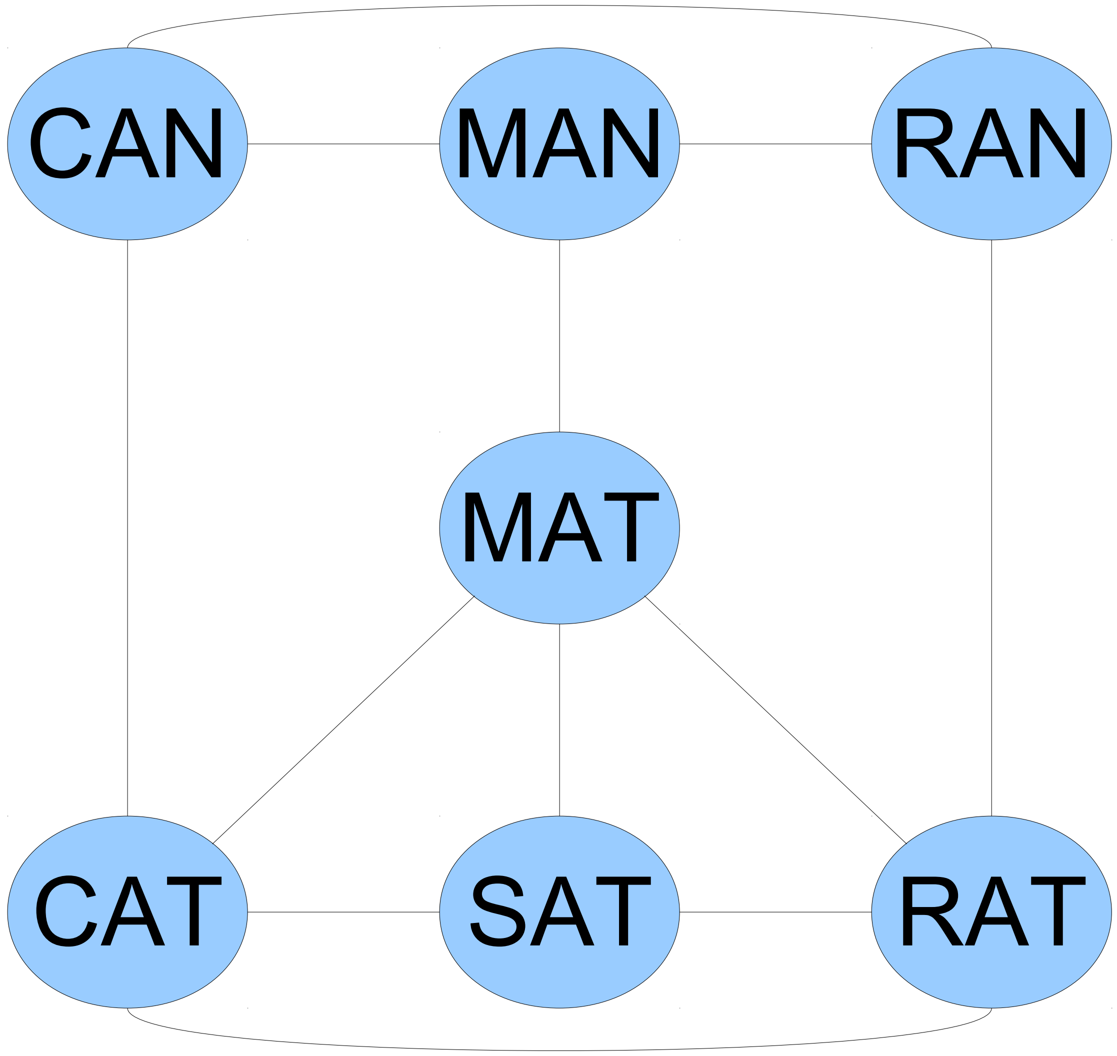


A graph consists of a set of ***nodes*** connected by ***edges***.

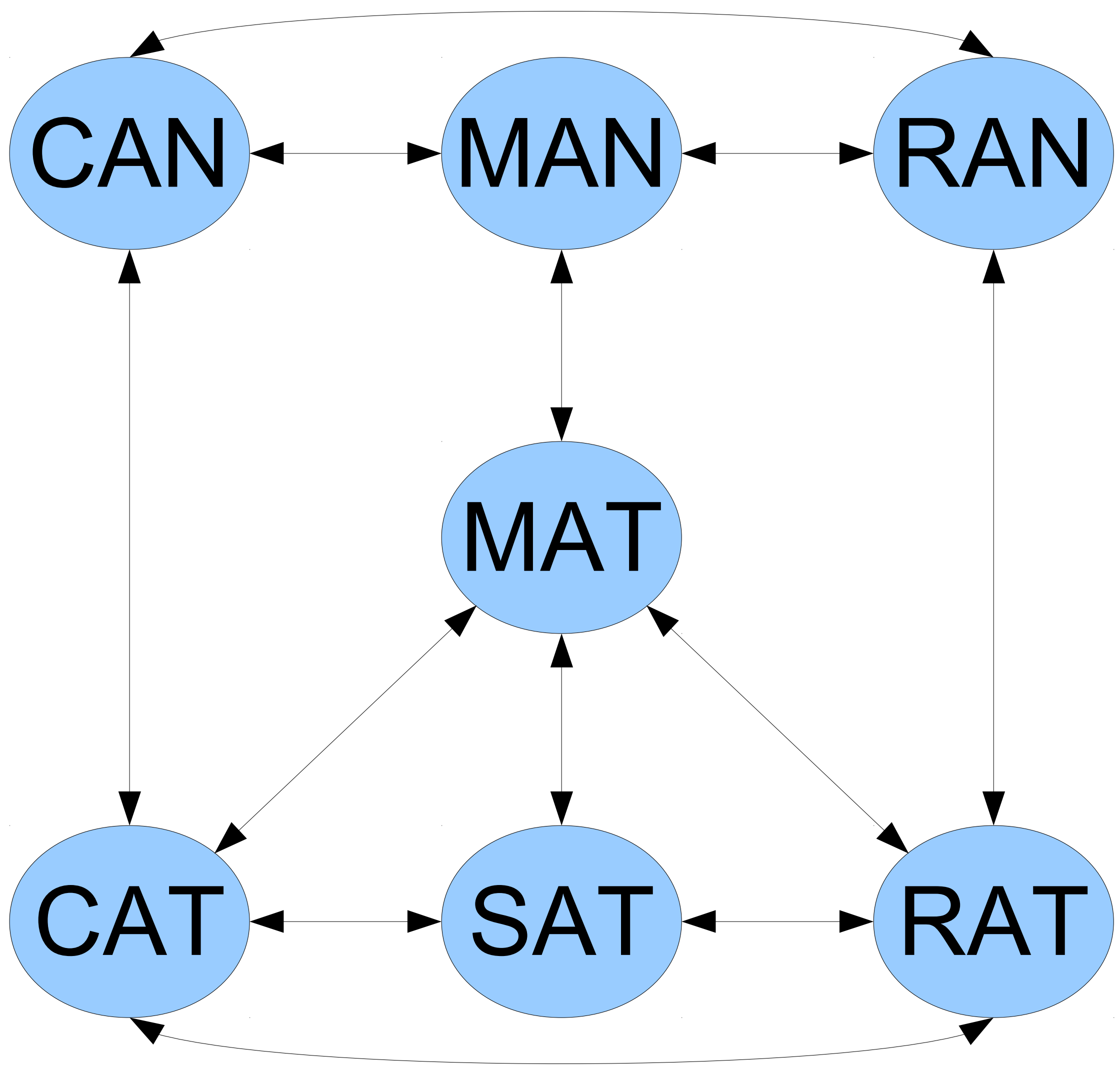
Some graphs are ***directed***.



Some graphs are ***undirected***.



Some graphs are ***undirected***.

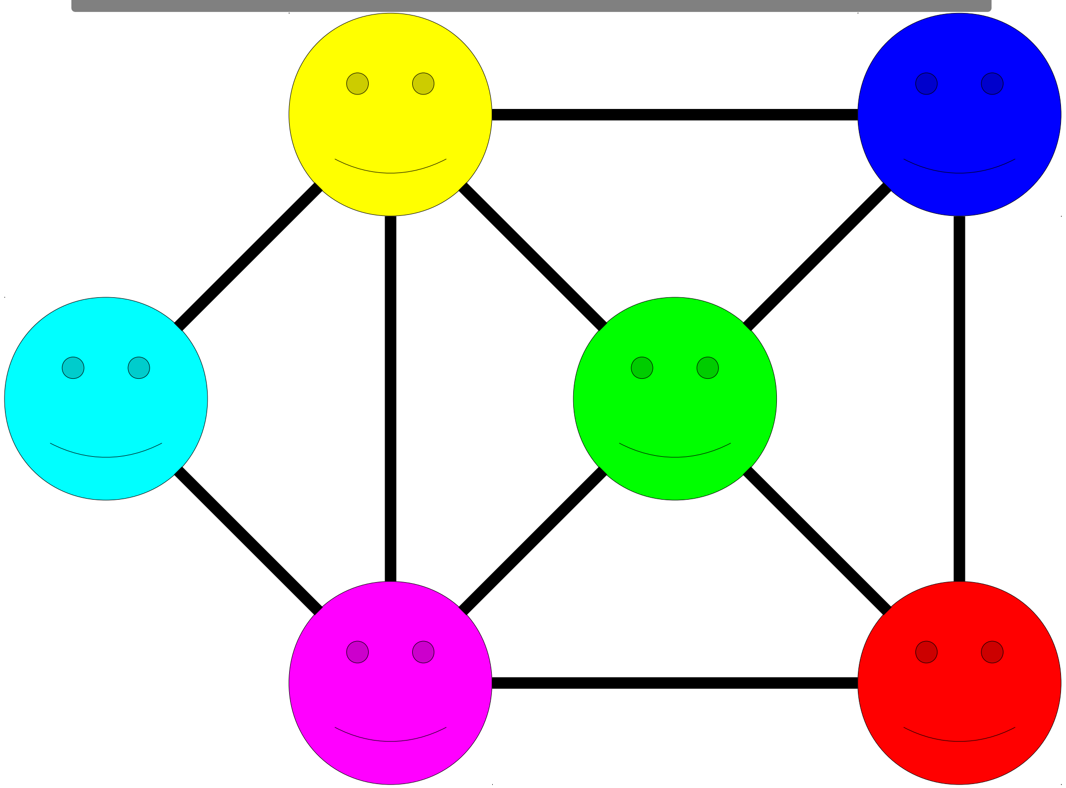


It sometimes helps to think of them as directed graphs with edges both ways.

How can we represent graphs in C++?

Representing Graphs

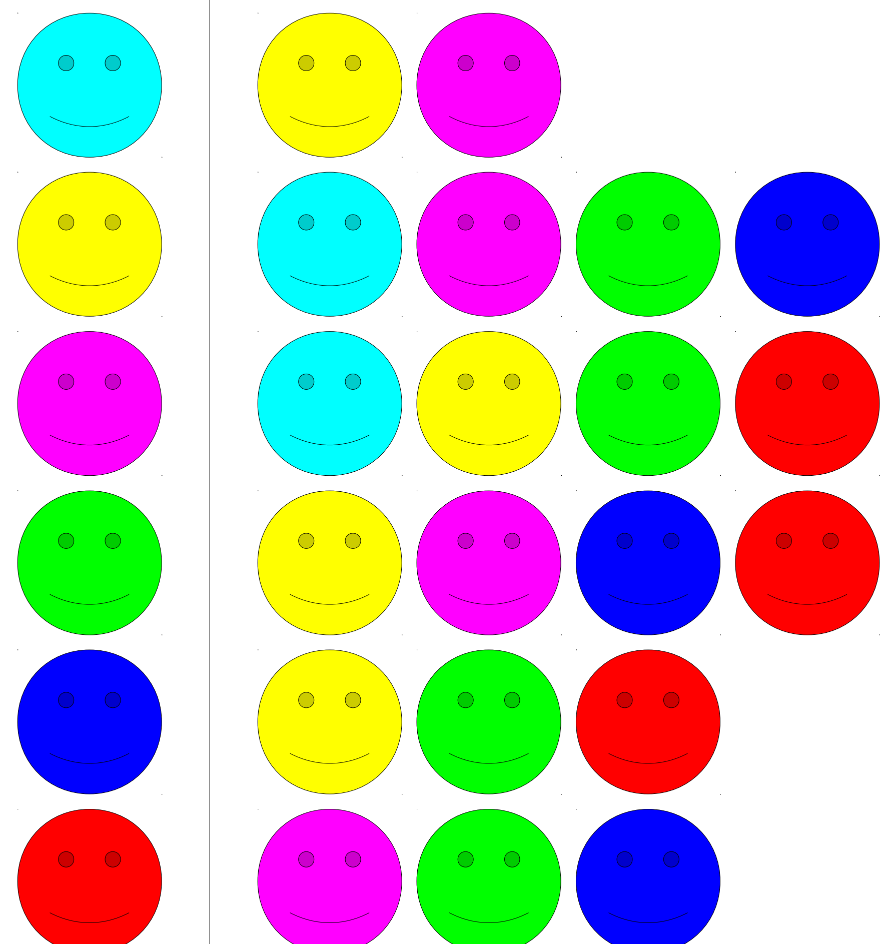
We can represent a graph as a map from nodes to the list of nodes each node is connected to.



Map<**Node**, Vector<**Node**>>

Node Vector<**Node**>

Node	Connected To



Representing Graphs

- The approach we just saw is called an ***adjacency list*** in comes in a number of different forms:

```
Map<string, Vector<string>>
Map<string, Set<string>>
HashMap<string, HashSet<string>>
Vector<Vector<int>>
```

- The core idea is that we have some kind of mapping associating each node with its outgoing edges.

Representing Graphs

The approach we just saw is called an ***adjacency list*** in comes in a number of different forms:

Map<string, Vector<string>>

Map<string, Set<string>>

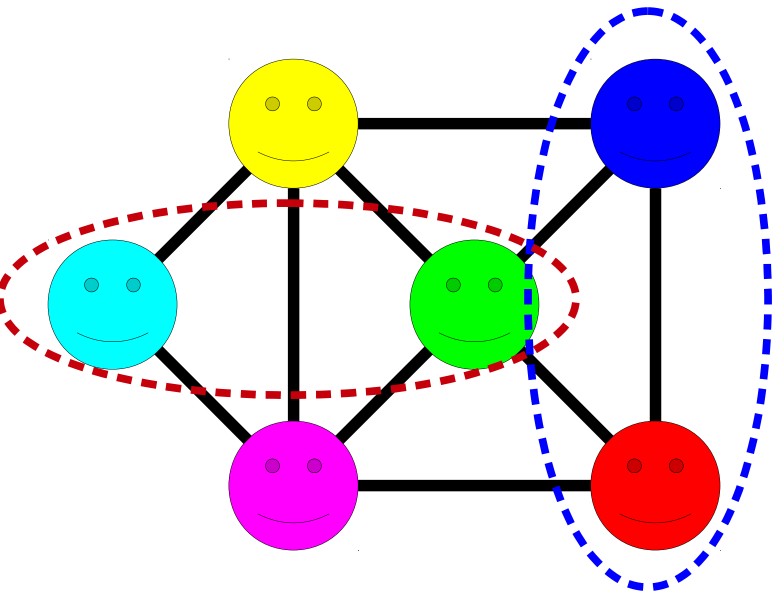
HashMap<string,

Vector<Ve

Question to ponder: where have you seen this before?

The core idea is that we have some kind of mapping associating each node with its outgoing edges.

Other Graph Representations

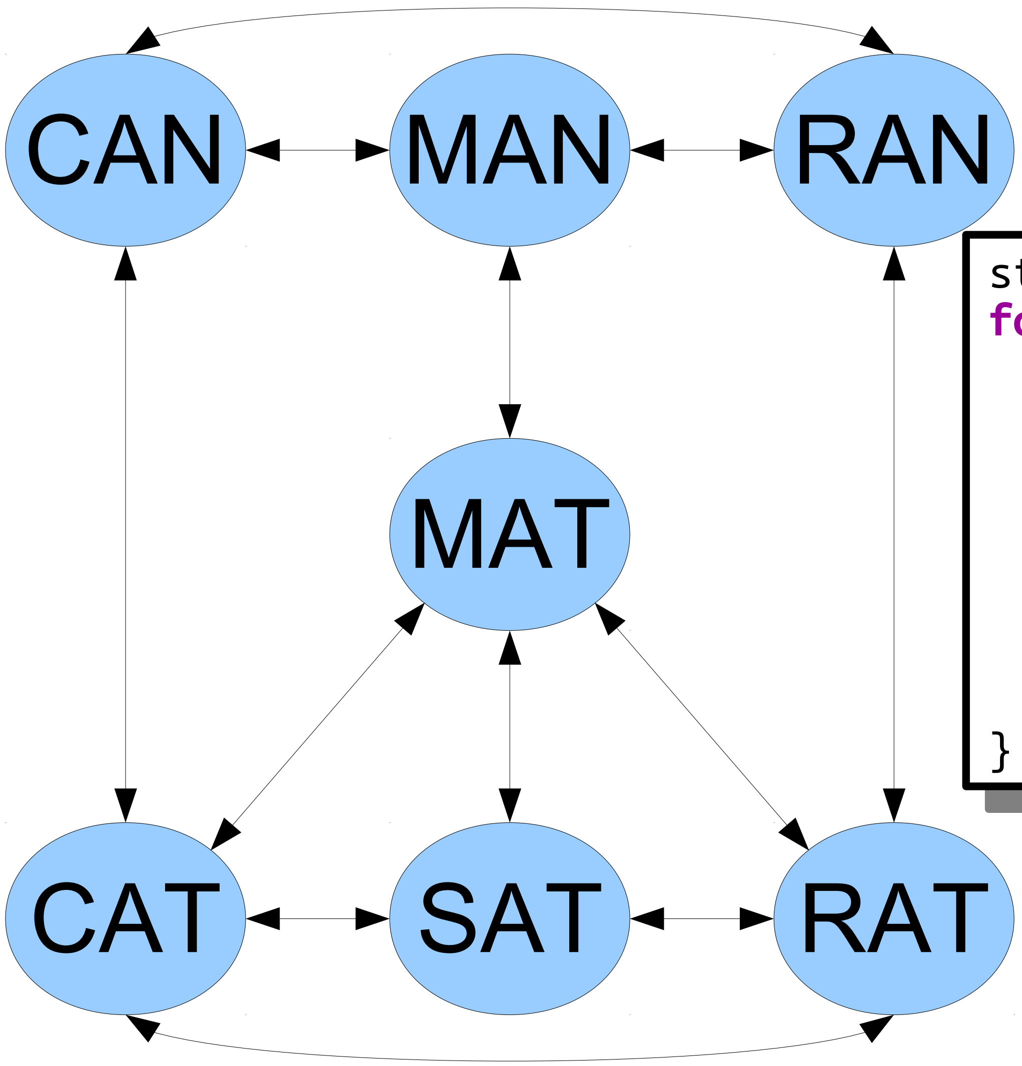


$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

This representation is called an *adjacency matrix*.

For those of you in Math 51: if A is an adjacency matrix for a graph G, what is the significance of the matrix A^2?

Other Representations




```
string word = /* … */;
for (int i = 0; i < word.size(); i++) {
    for (char ch = 'a'; ch <= 'z'; ch++) {
        string newWord = word;
        newWord[i] = ch;
        if (word != newWord &&
            lex.contains(newWord)) {
            /* … edge exists! … */
        }
    }
}
```

Lots of code works on implicit graphs. Drawing the picture often makes it clearer!

You'll find graphs just
about everywhere you look.

They're an ***extremely*** versatile and
powerful abstraction to be aware of.

Time-Out for Announcements!

Assignment 6

- Assignment 6 is due this Friday.
- Have questions?
    - Stop by the LaIR!
    - Ask your section leader!
    - Stop by Keith’s office hours on Tuesday!
    - Stop by Anton’s office hours on Wednesday!
    - Ask on Piazza!

Stanford 6:30 PM

CS+SG

CS+SOCIAL GOOD MIXER

Black Community Services Center

March 8th, 6:30-8:00pm

Mingle with your fellow students and faculty for a night of conversation about social good issues.

Tasty snacks & light refreshments will be served!

🍕 🍔 🍰 🍞 🍭

iMessage

Send

Engineers for a Sustainable World
STANFORD UNIVERSITY

Stanford Engineers for a Sustainable World is looking for a team member to travel to Indonesia for a 9 week fellowship this summer to work with rural development NGO IBEKA.

This is the fourth year of Engineers for a Sustainable World's collaboration with IBEKA on an Internet of Things Remote Monitoring System for Micro-hydroelectric Plants. Learn more about the project **here**.

We are looking for students of any year with skills and interests in Electrical Engineering, Computer Science, and Product Design. Participation requires enrolling in CEE177S Spring Quarter.

If interested, please email **riyav@stanford.edu** with a short paragraph explaining your interest and qualifications.

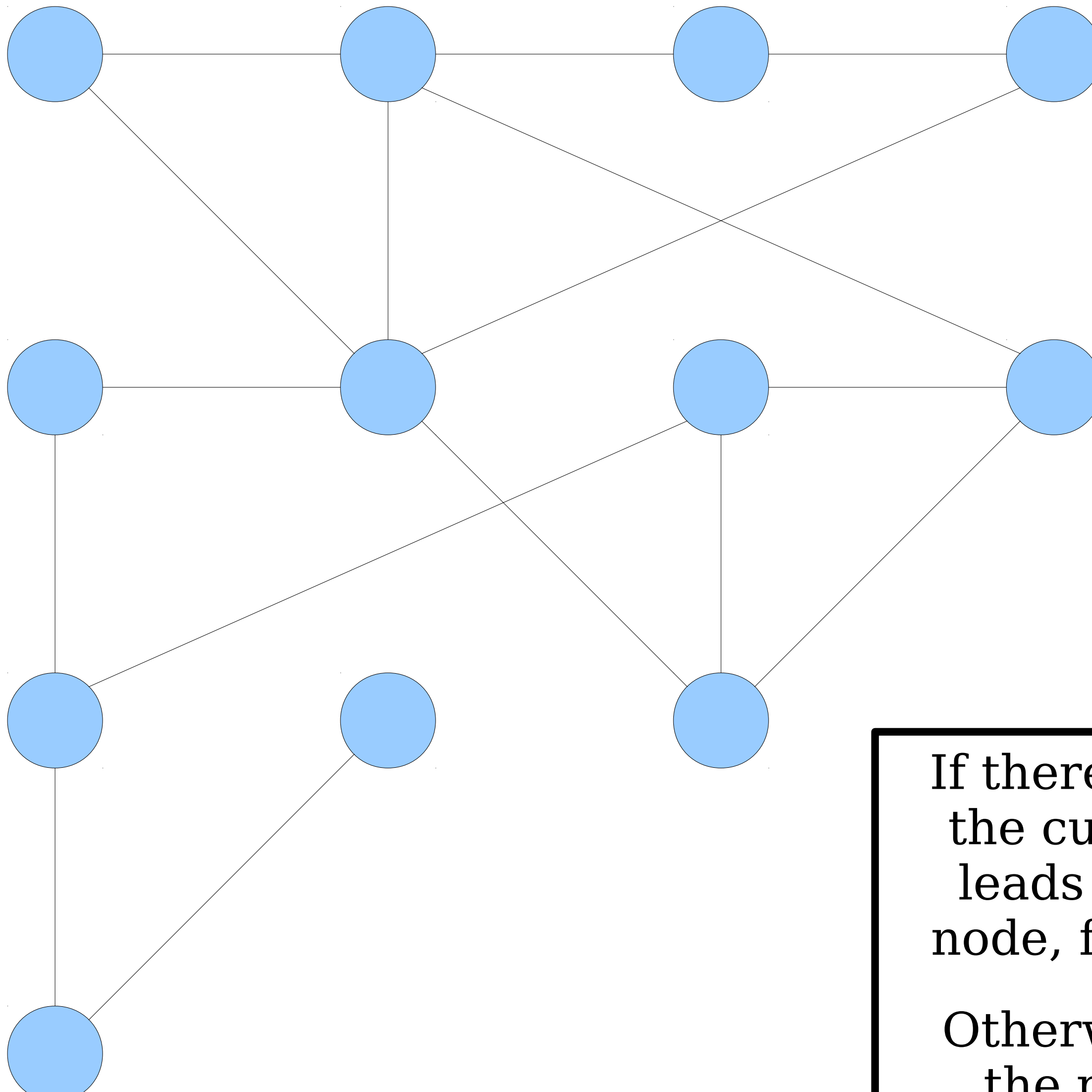
Back to CS106B!

Traversing Graphs

Iterating over a Graph

- In a singly-linked list, there's pretty much one way to iterate over the list: start at the front and go forward!
- In a binary search tree, there are many traversal strategies:
  - An ***inorder traversal*** that produces all the elements in sorted order.
  - A ***postorder traversal*** used to delete all the nodes in the BST.
- There are *many* ways to iterate over a graph, each of which have different properties.

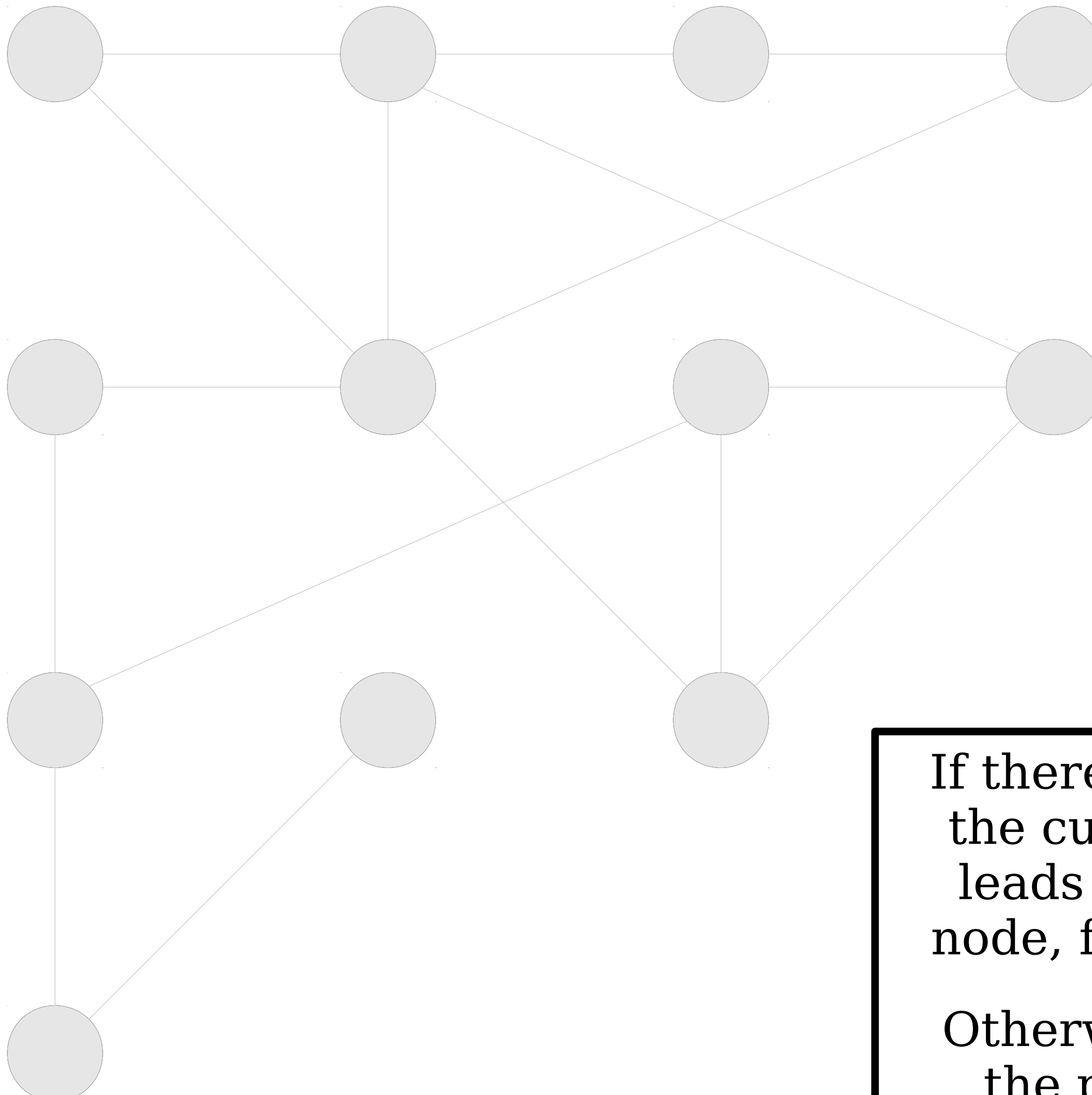
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

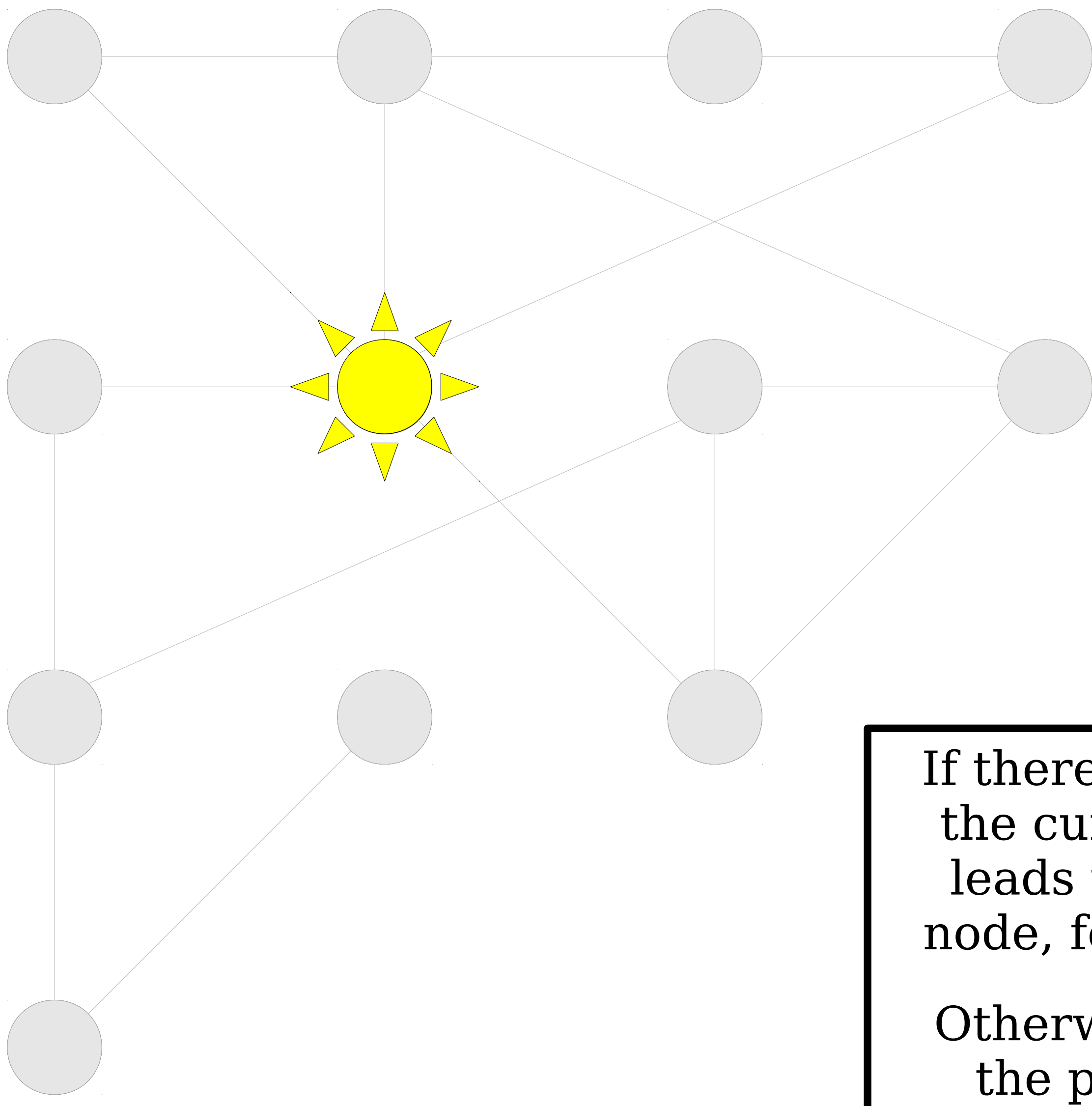
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

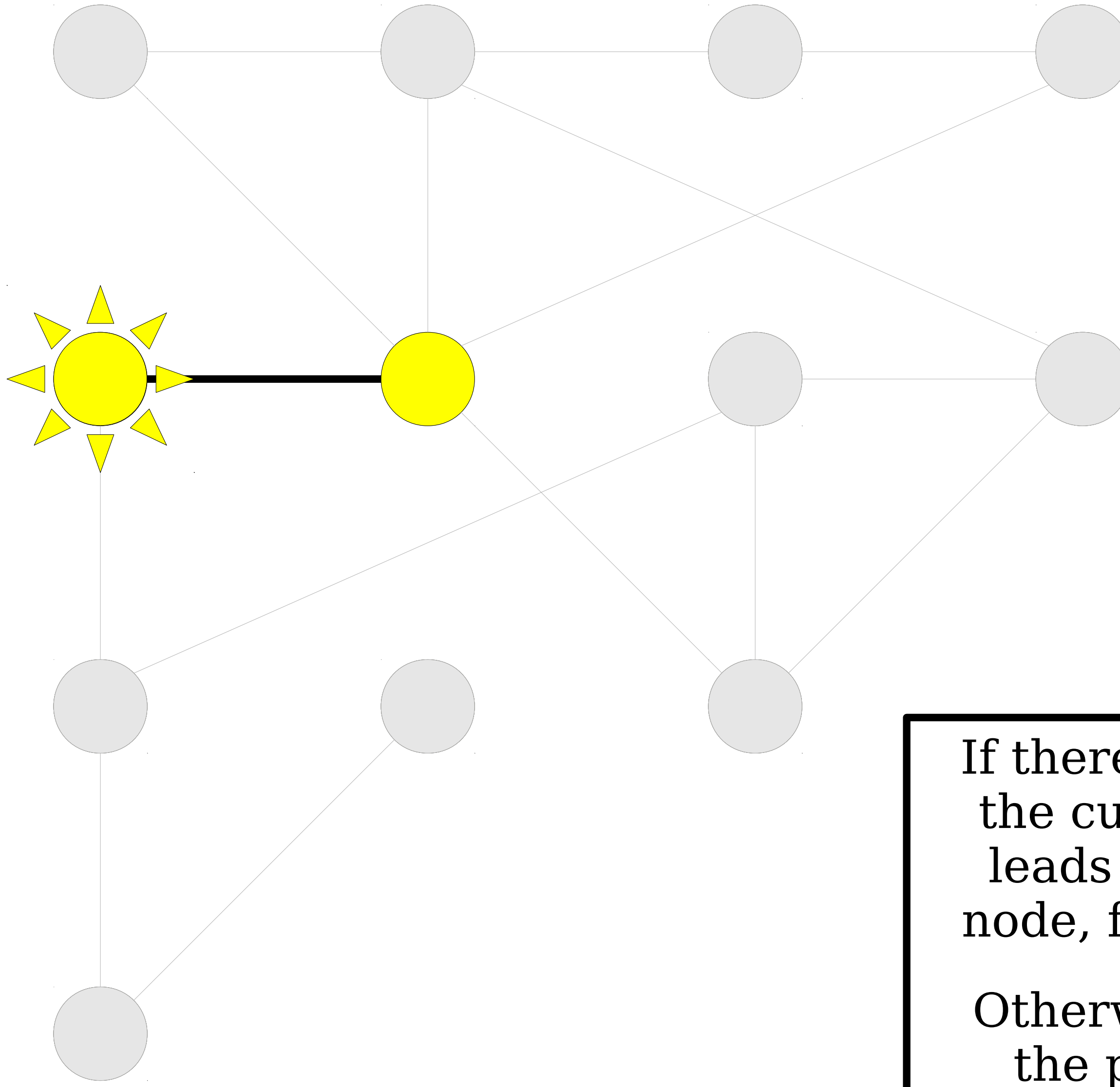
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

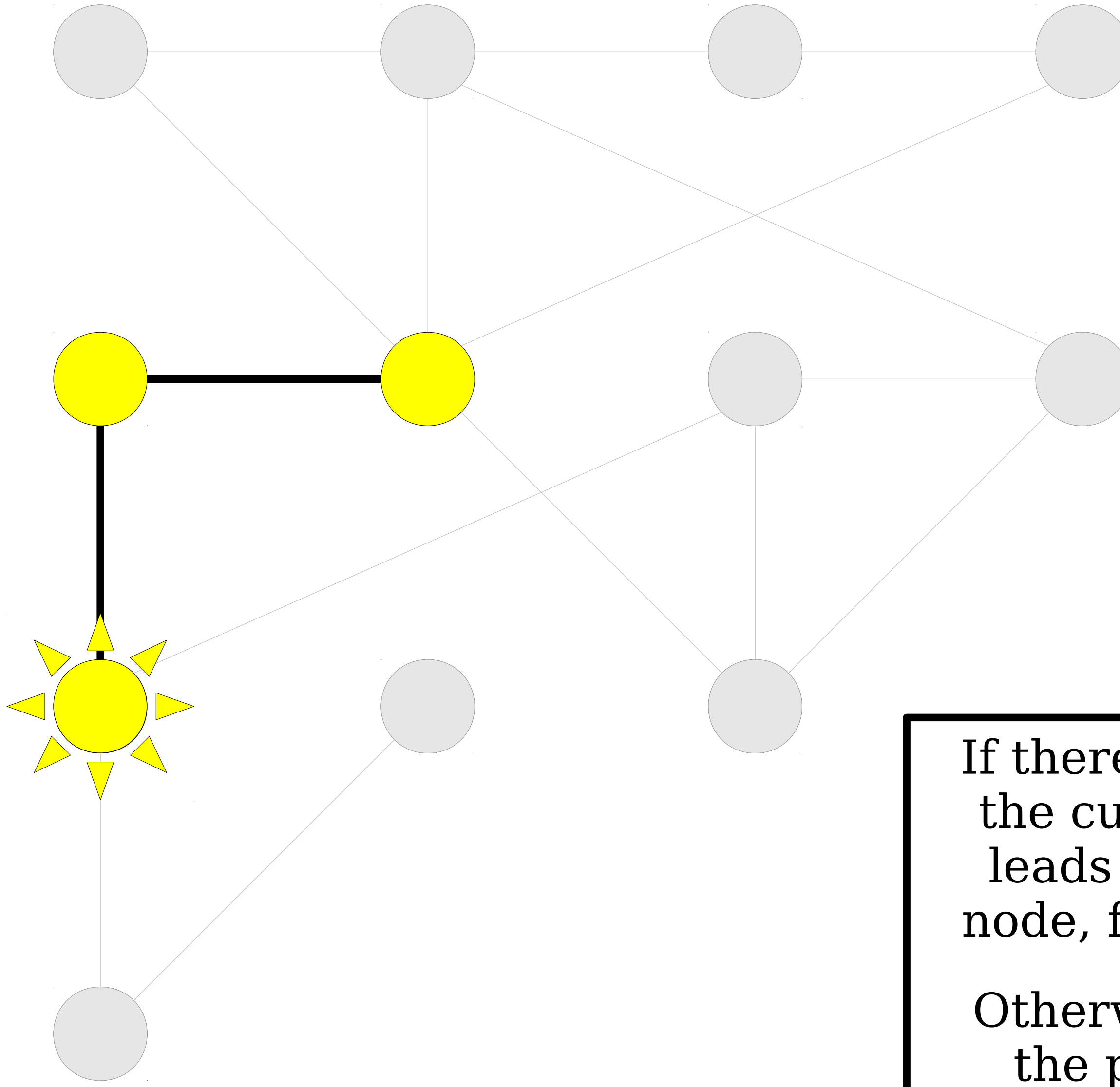
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

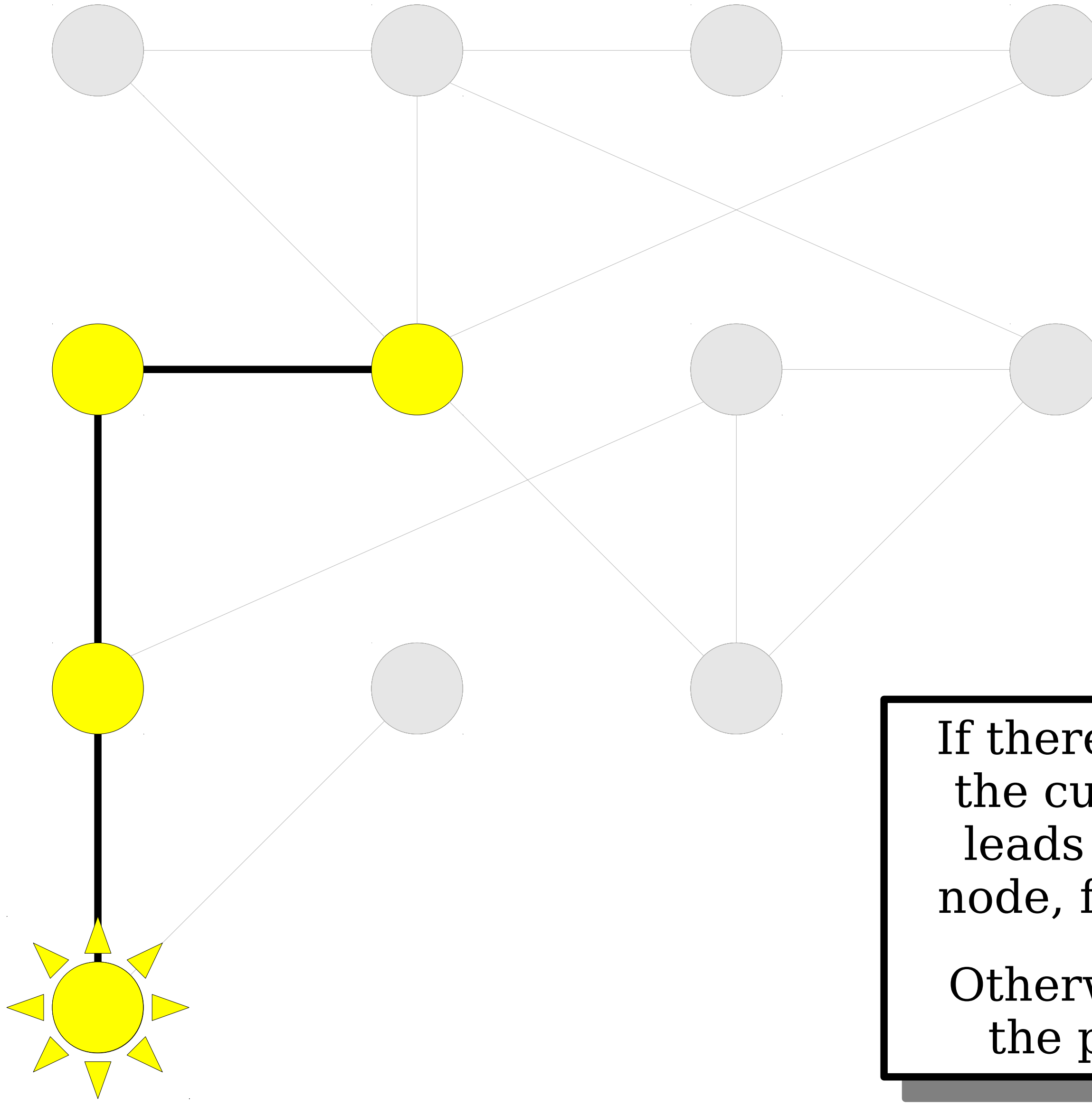
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

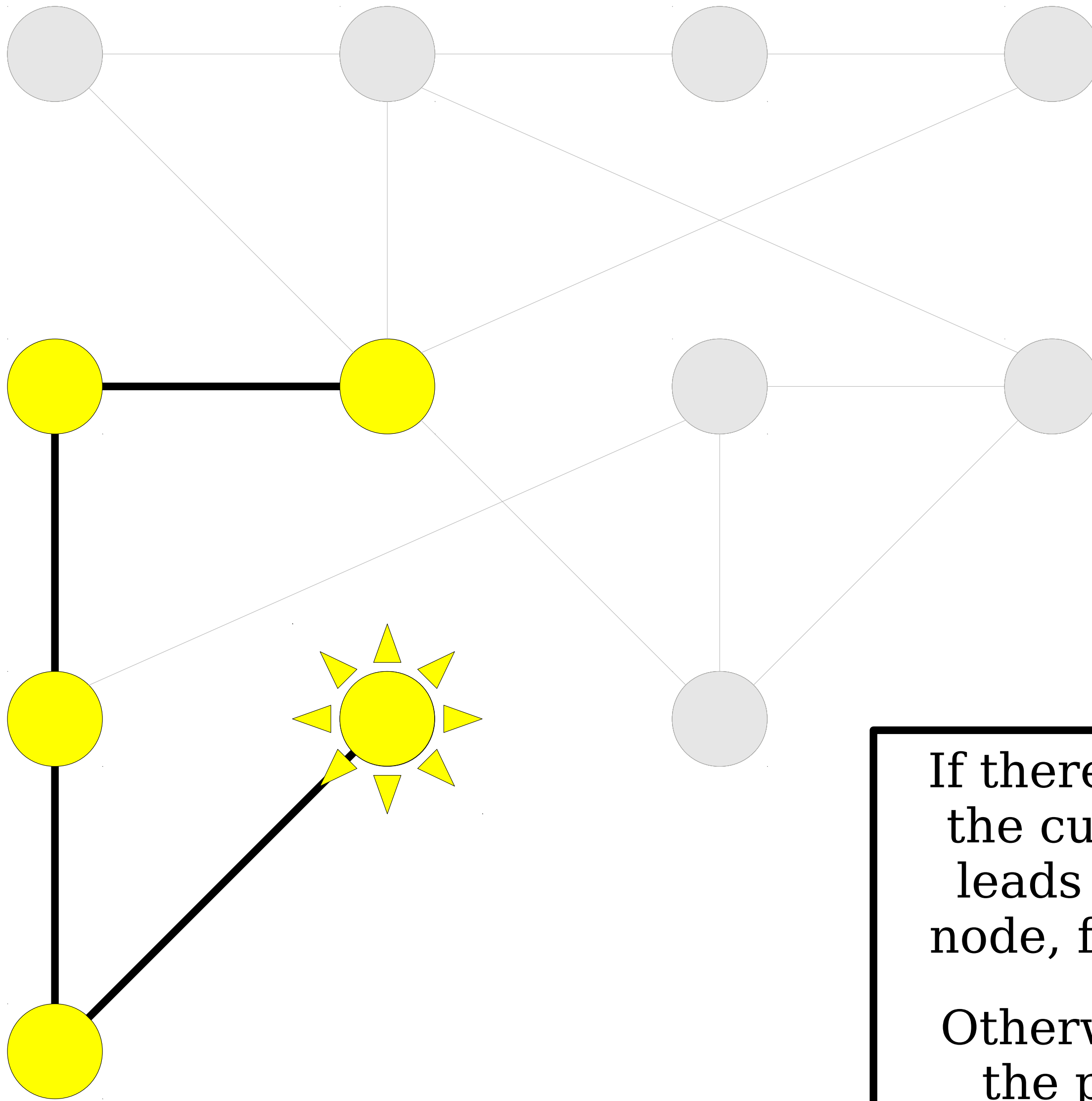
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

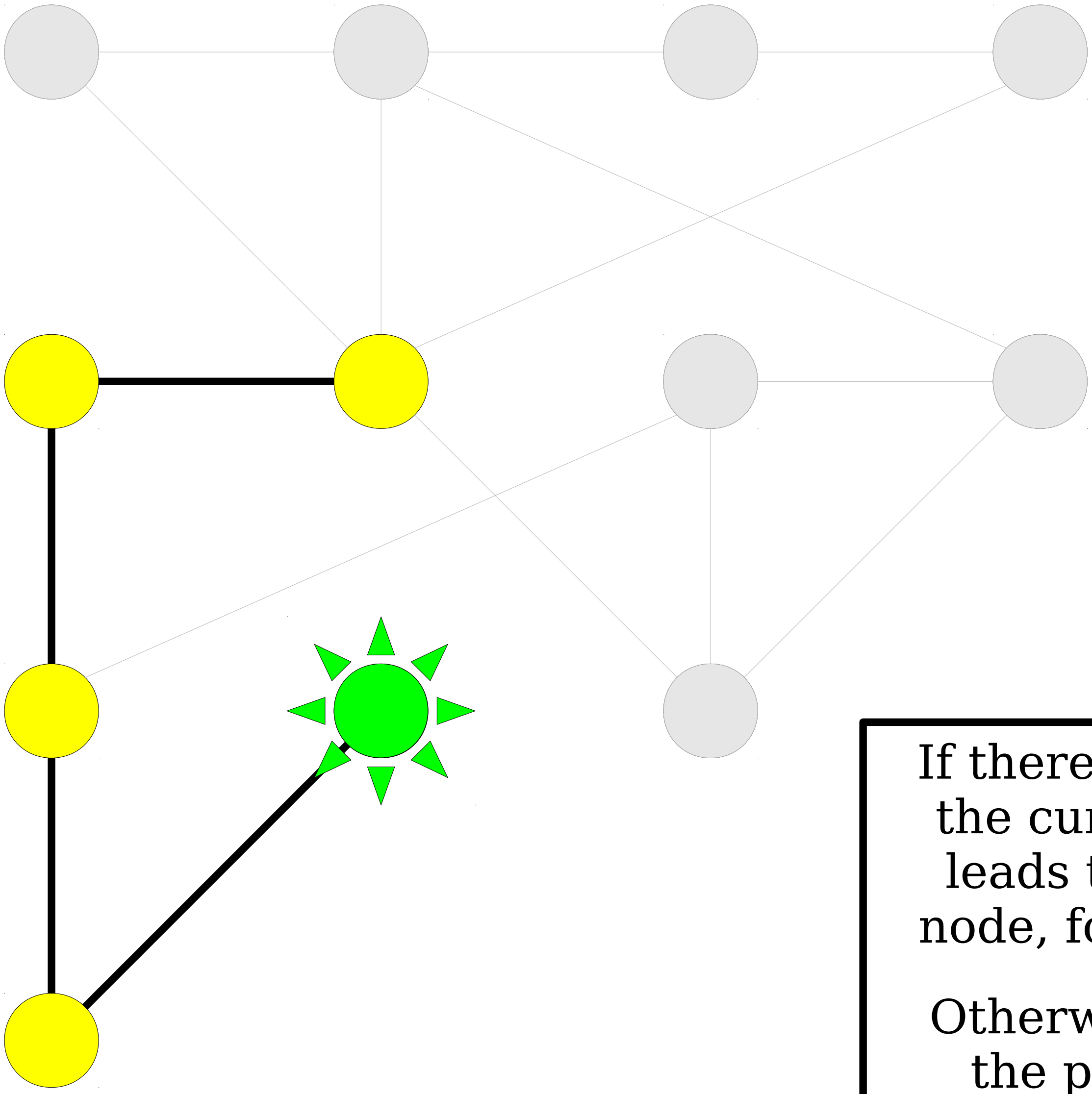
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

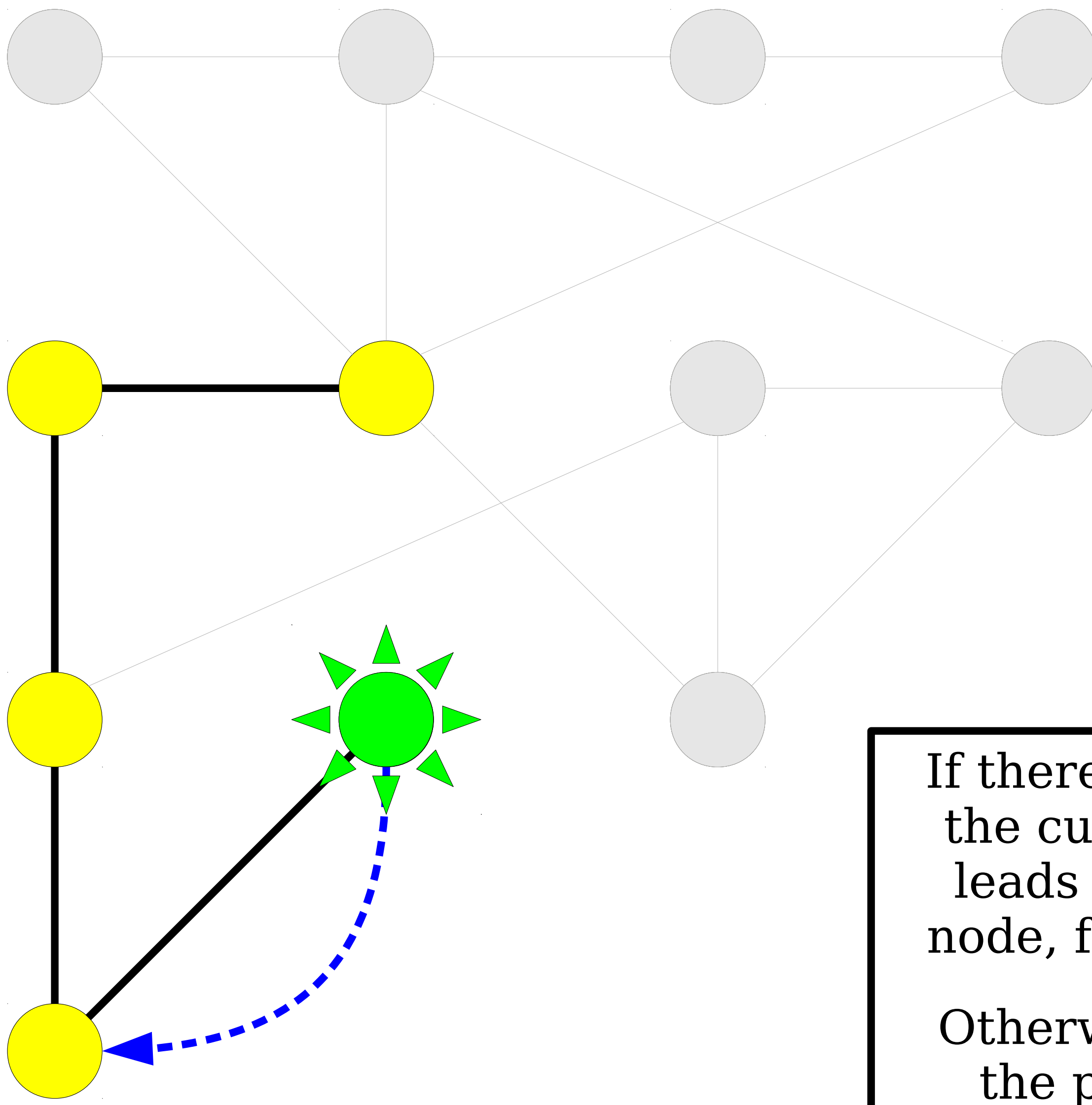
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

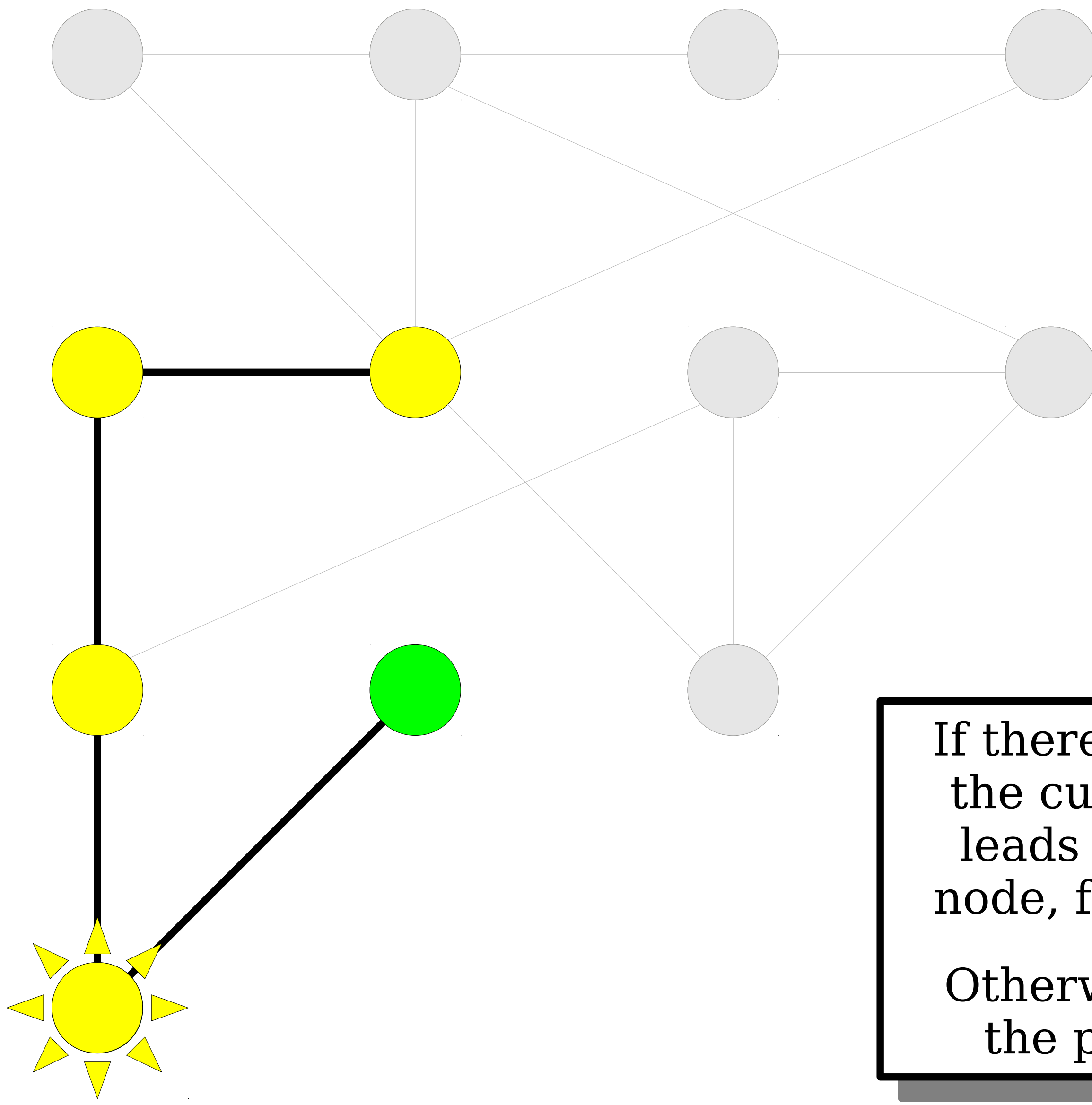
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

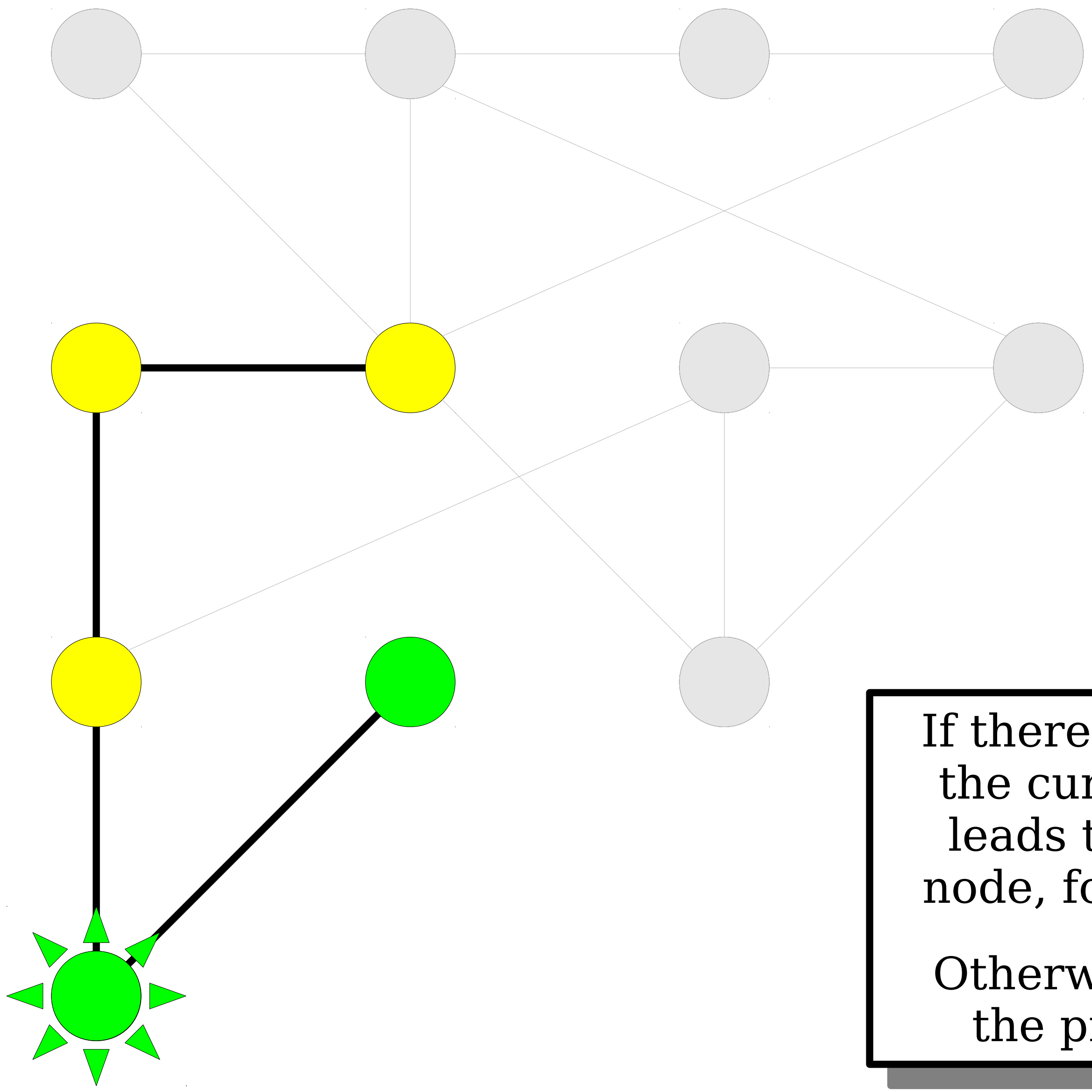
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

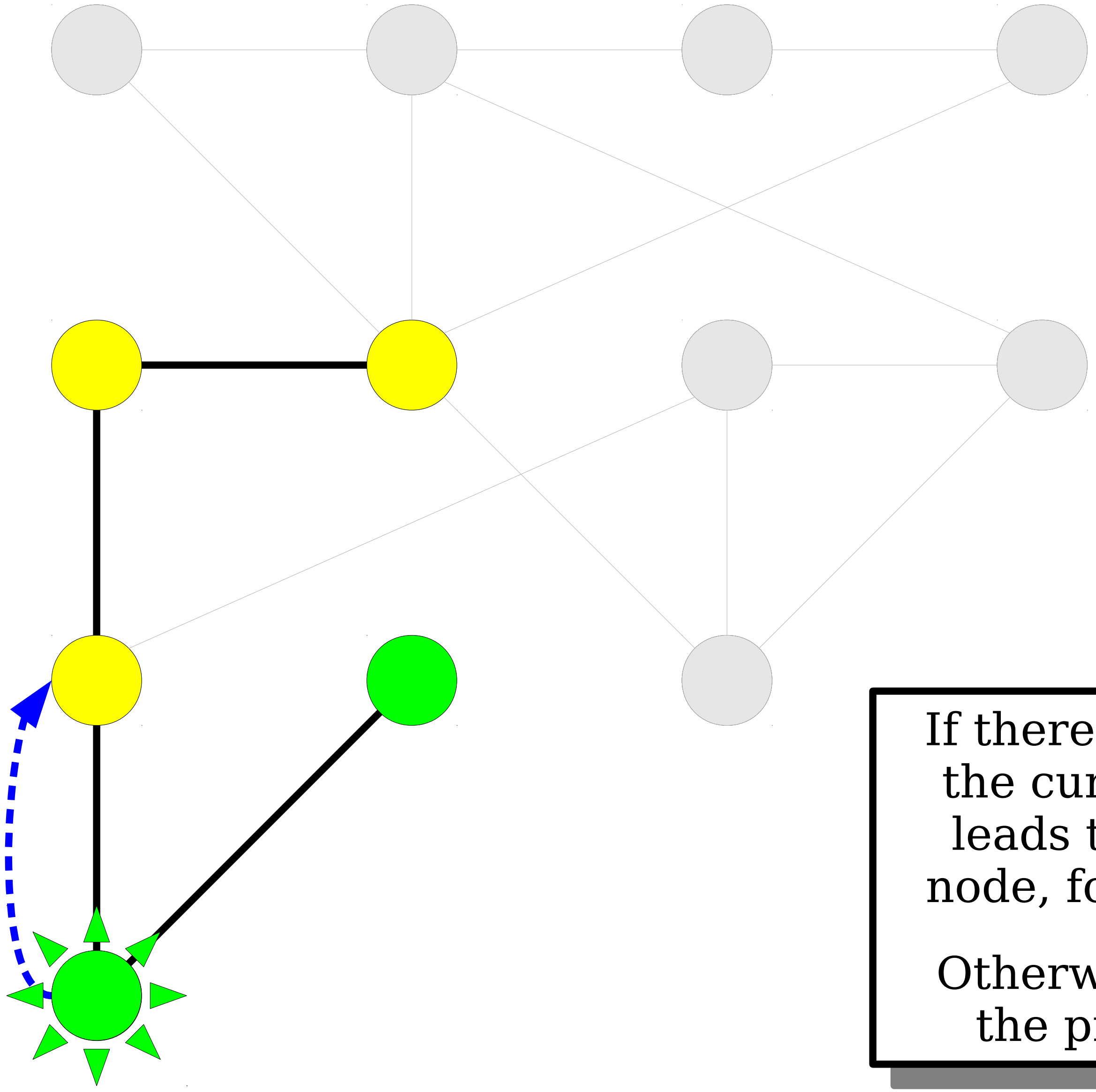
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

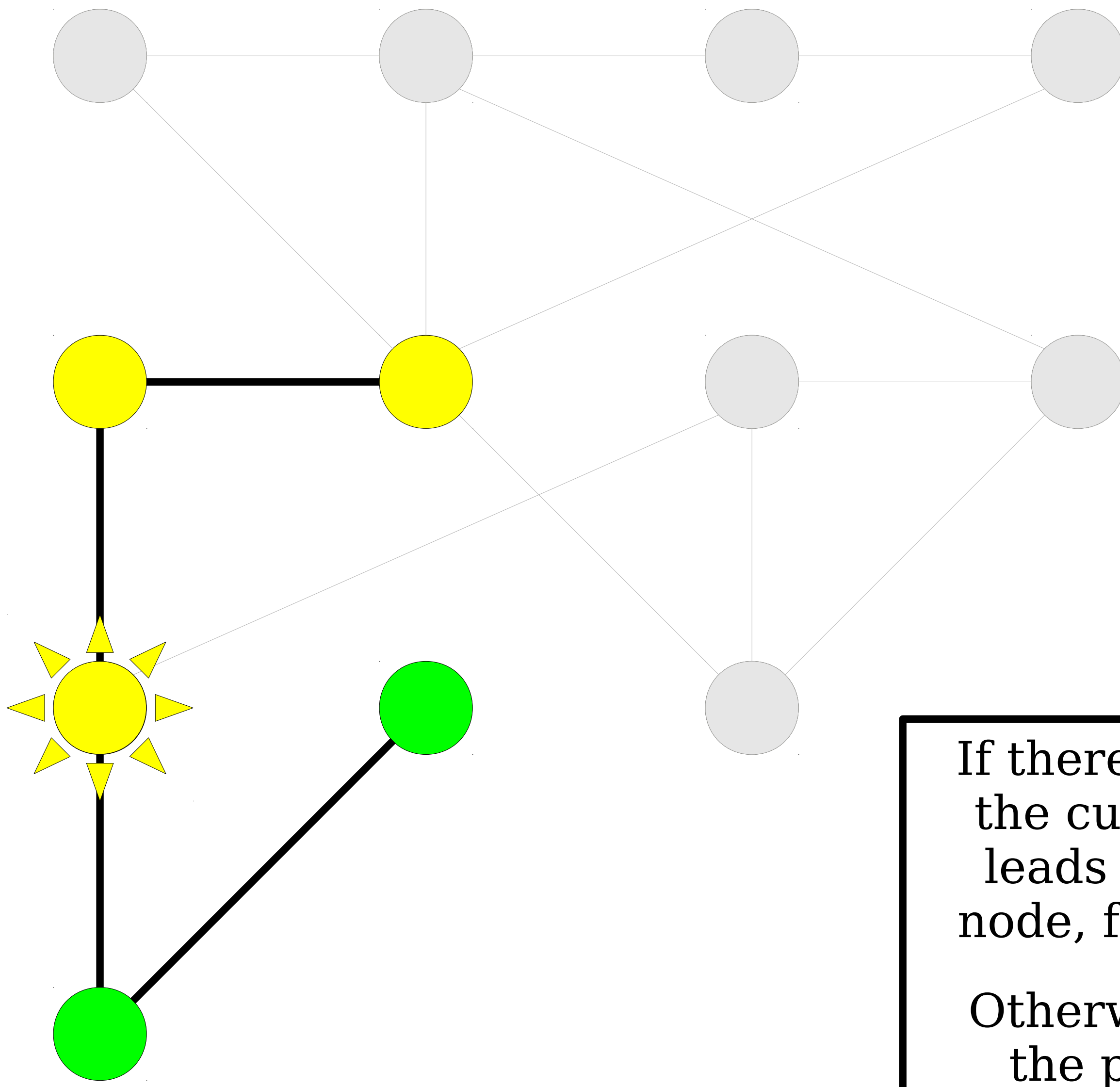
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

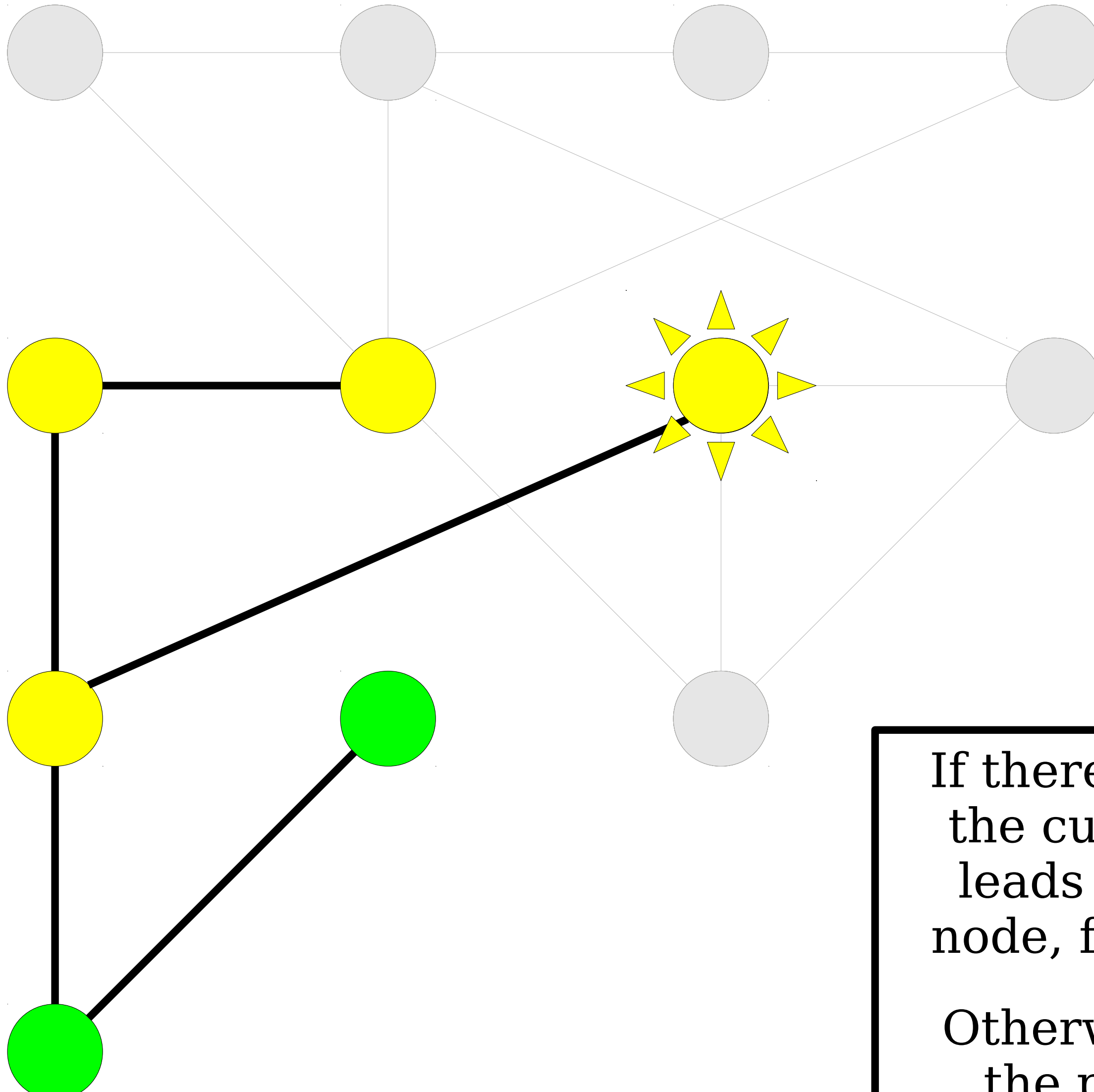
Depth-First Search



If there’s an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

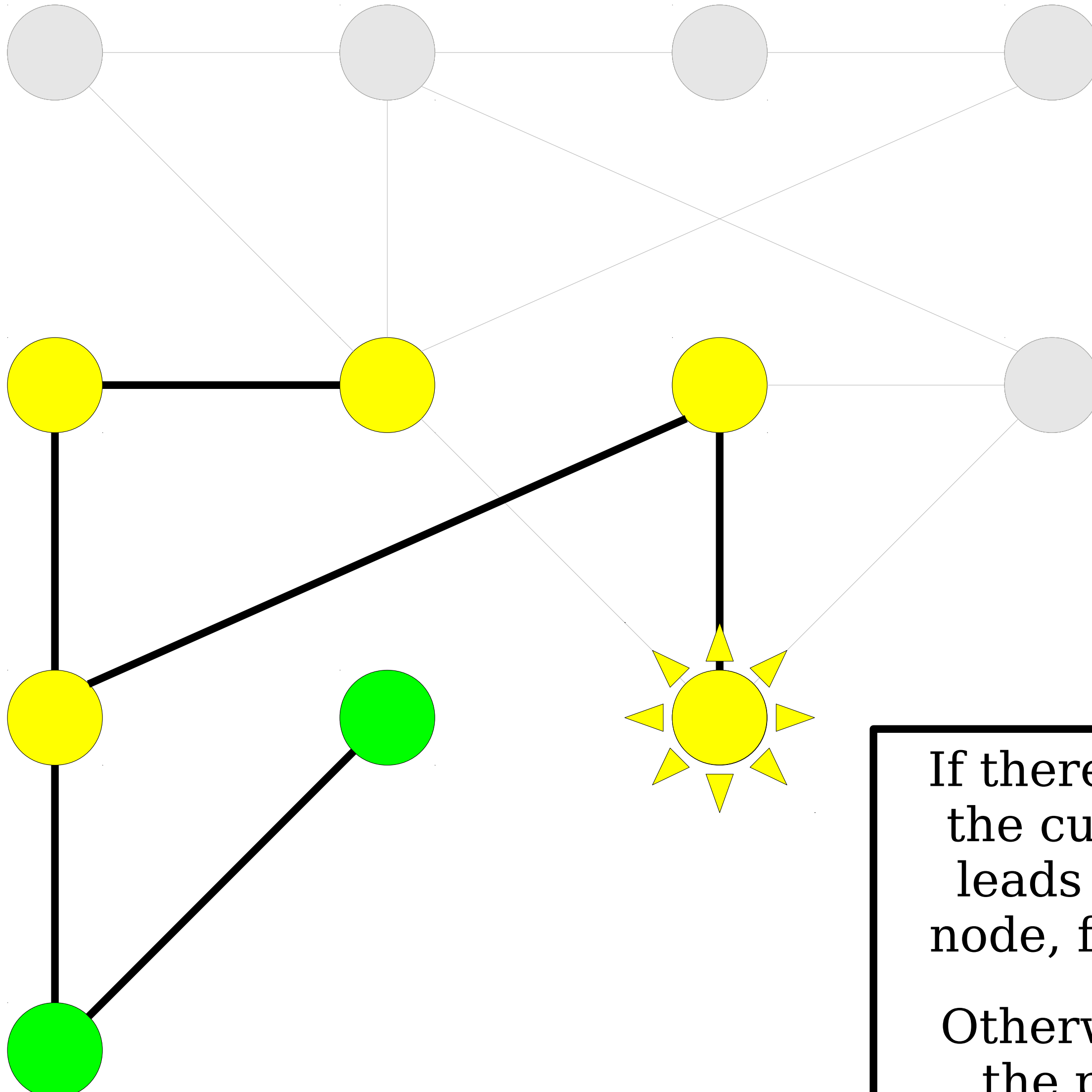
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

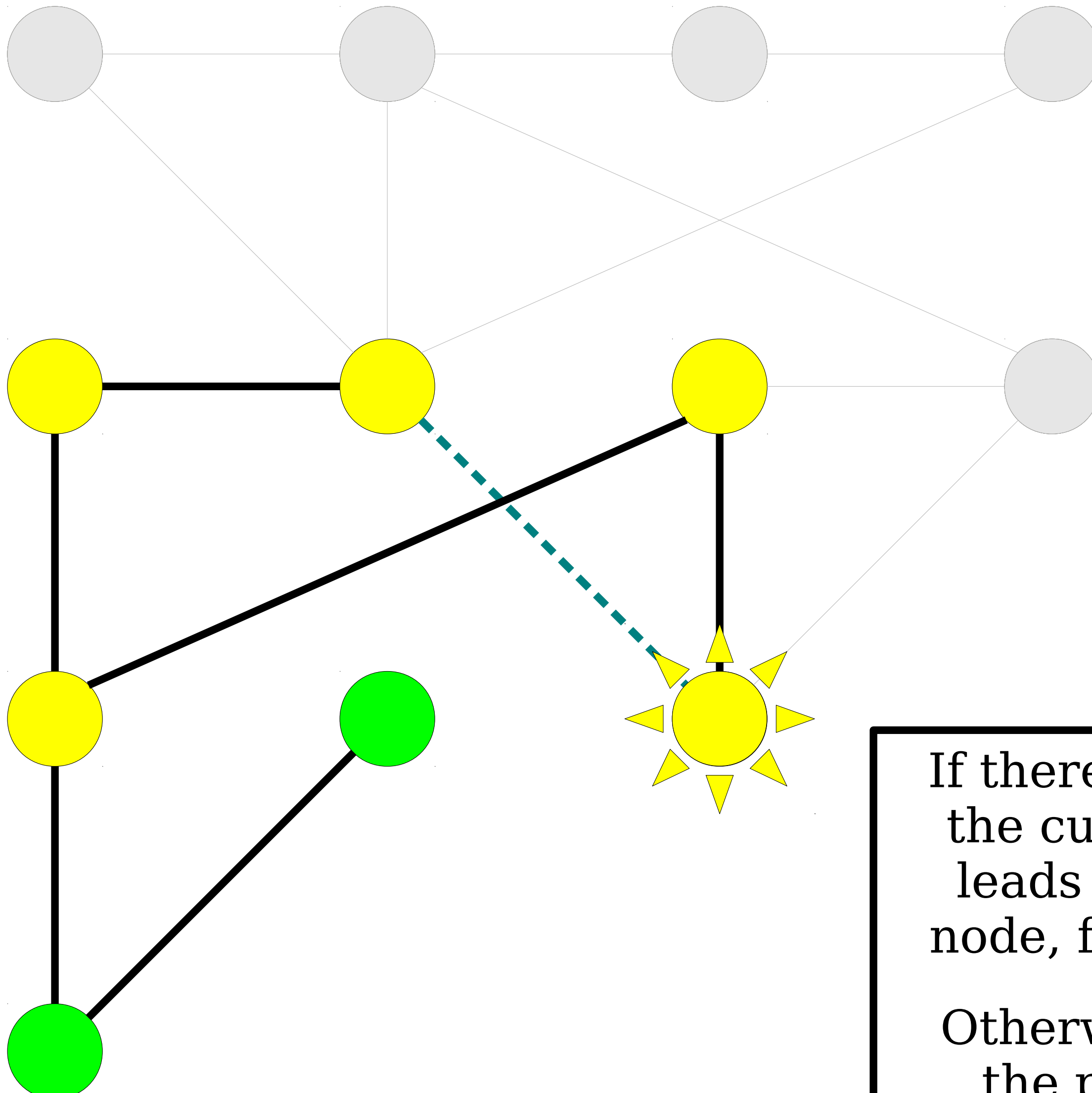
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

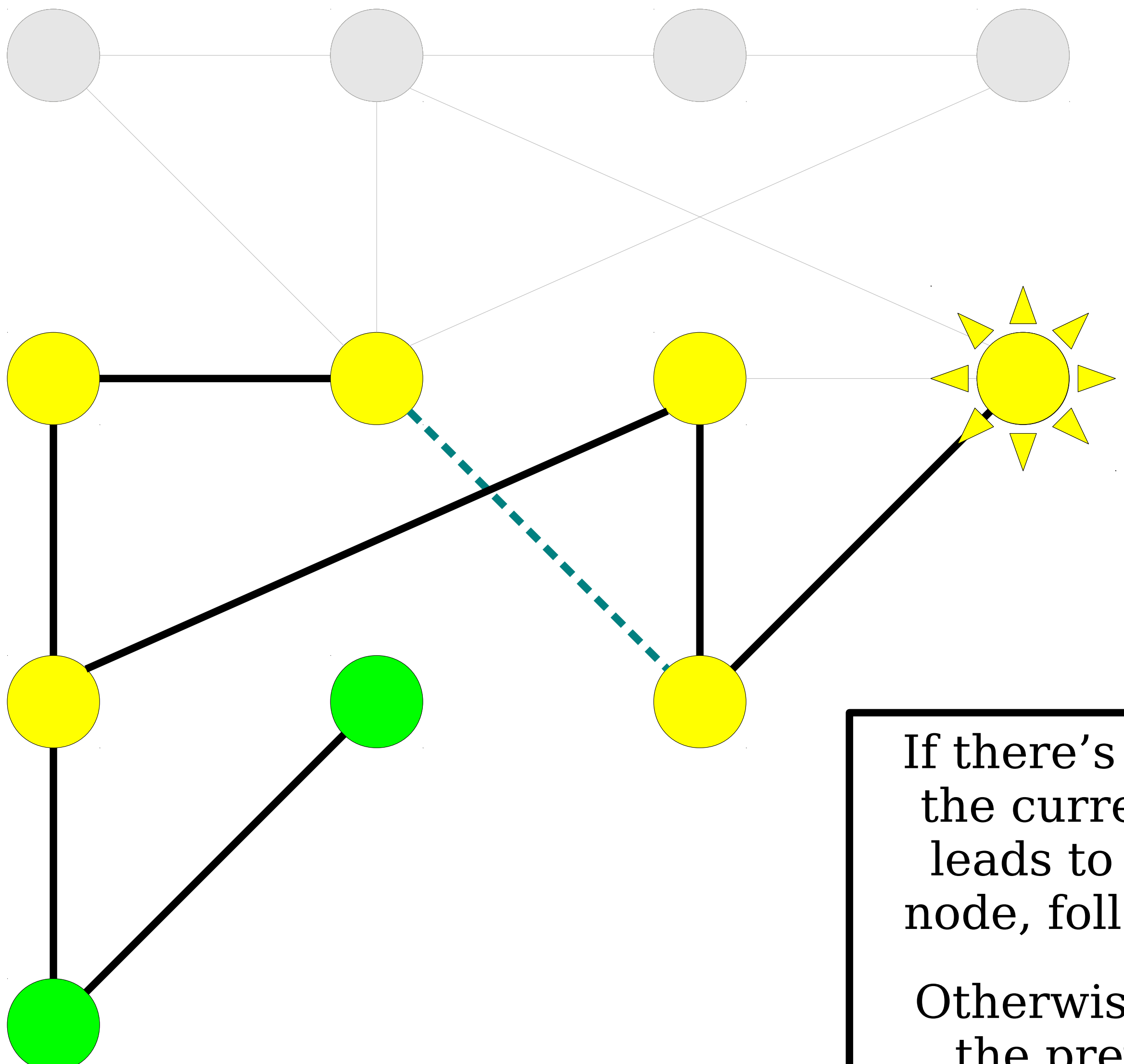
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

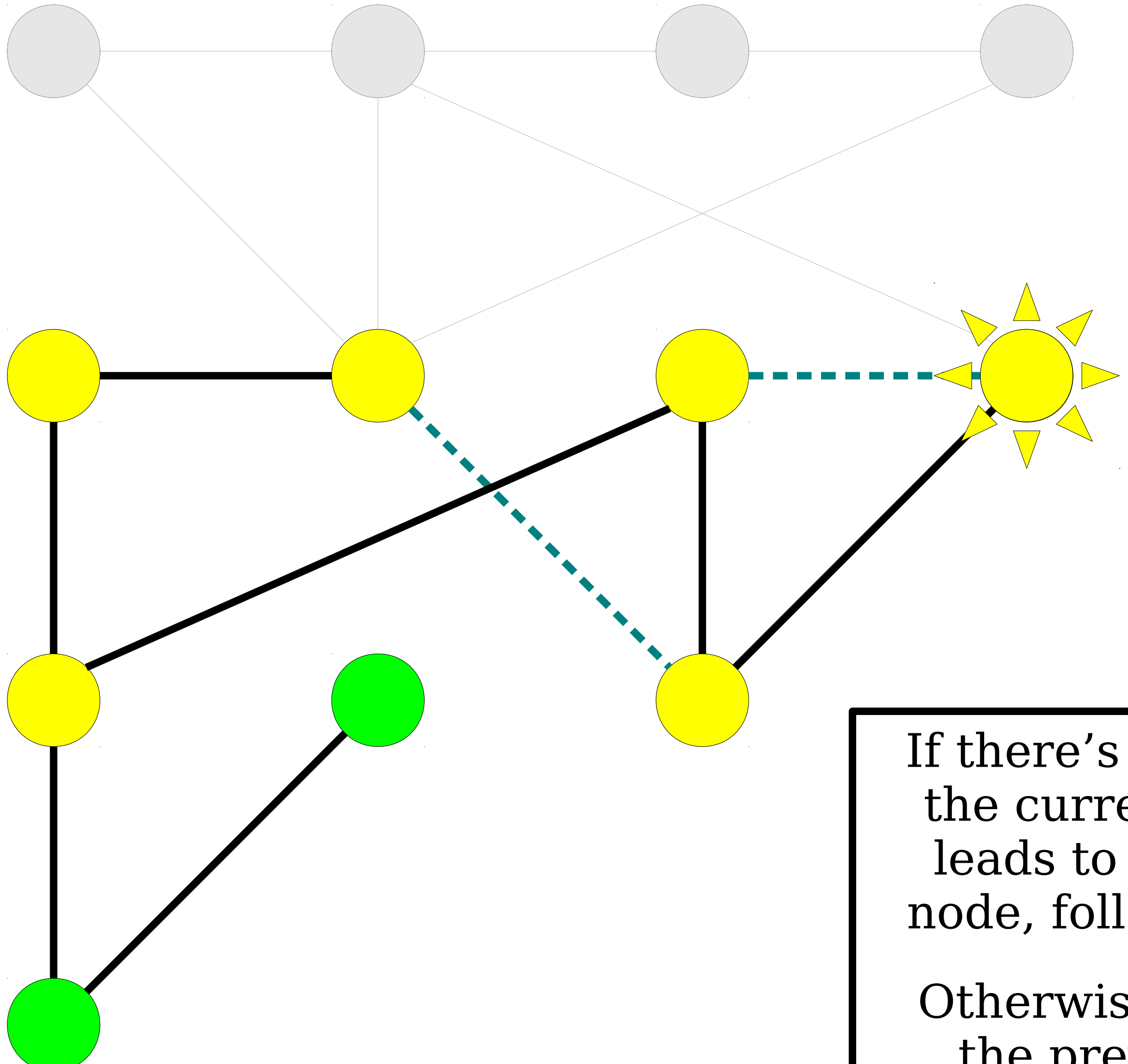
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

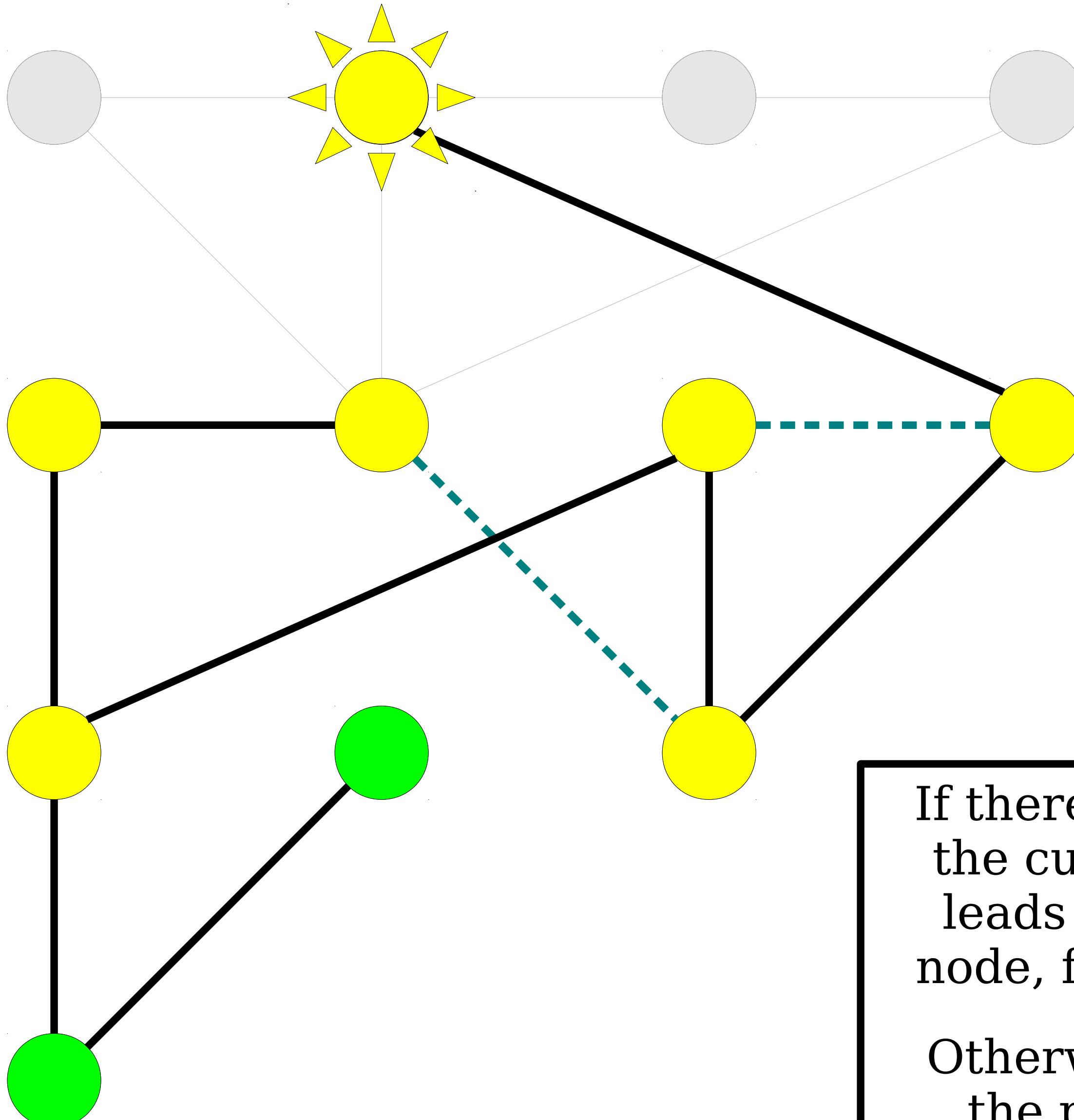
Depth-First Search



If there’s an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

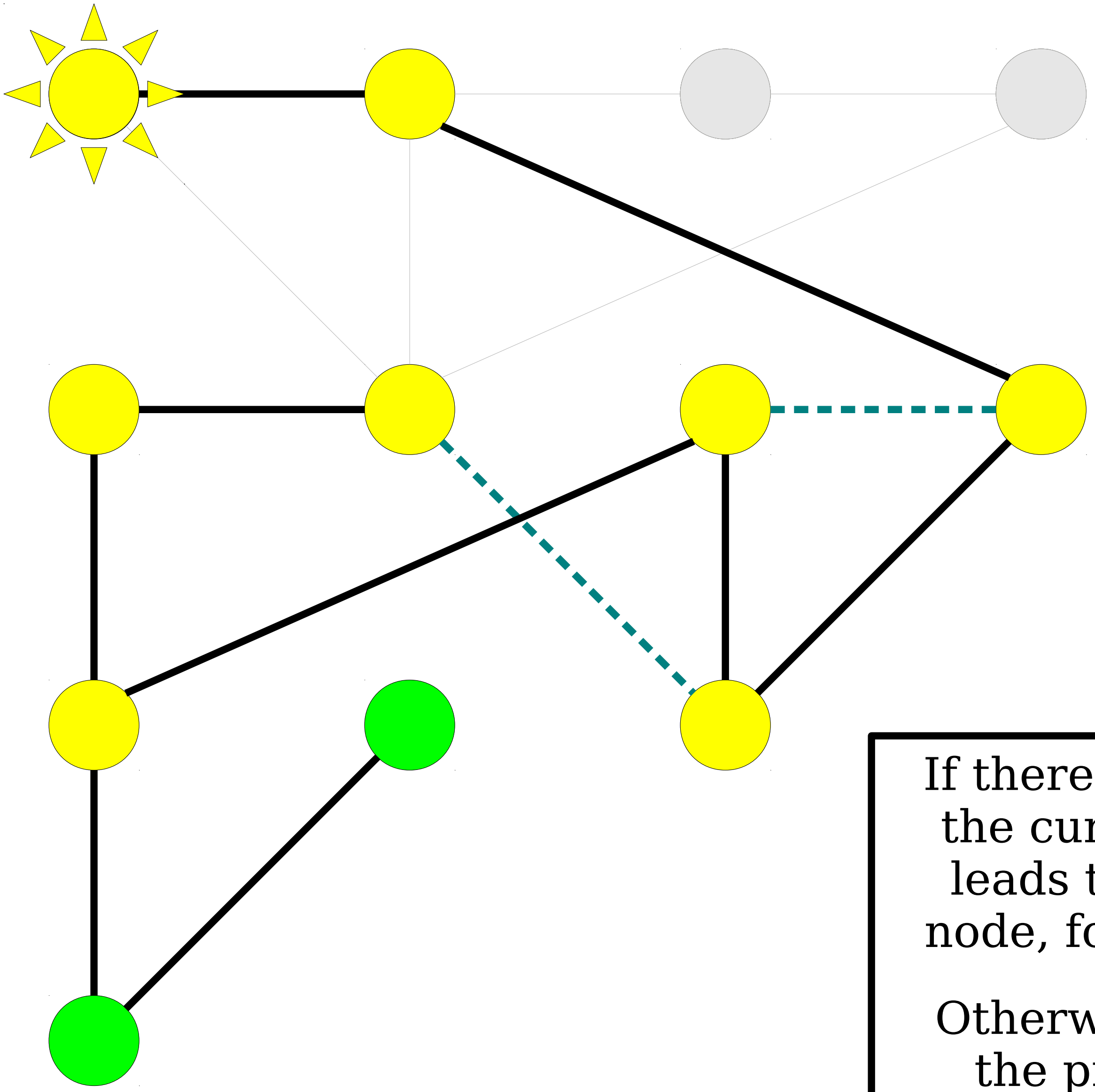
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

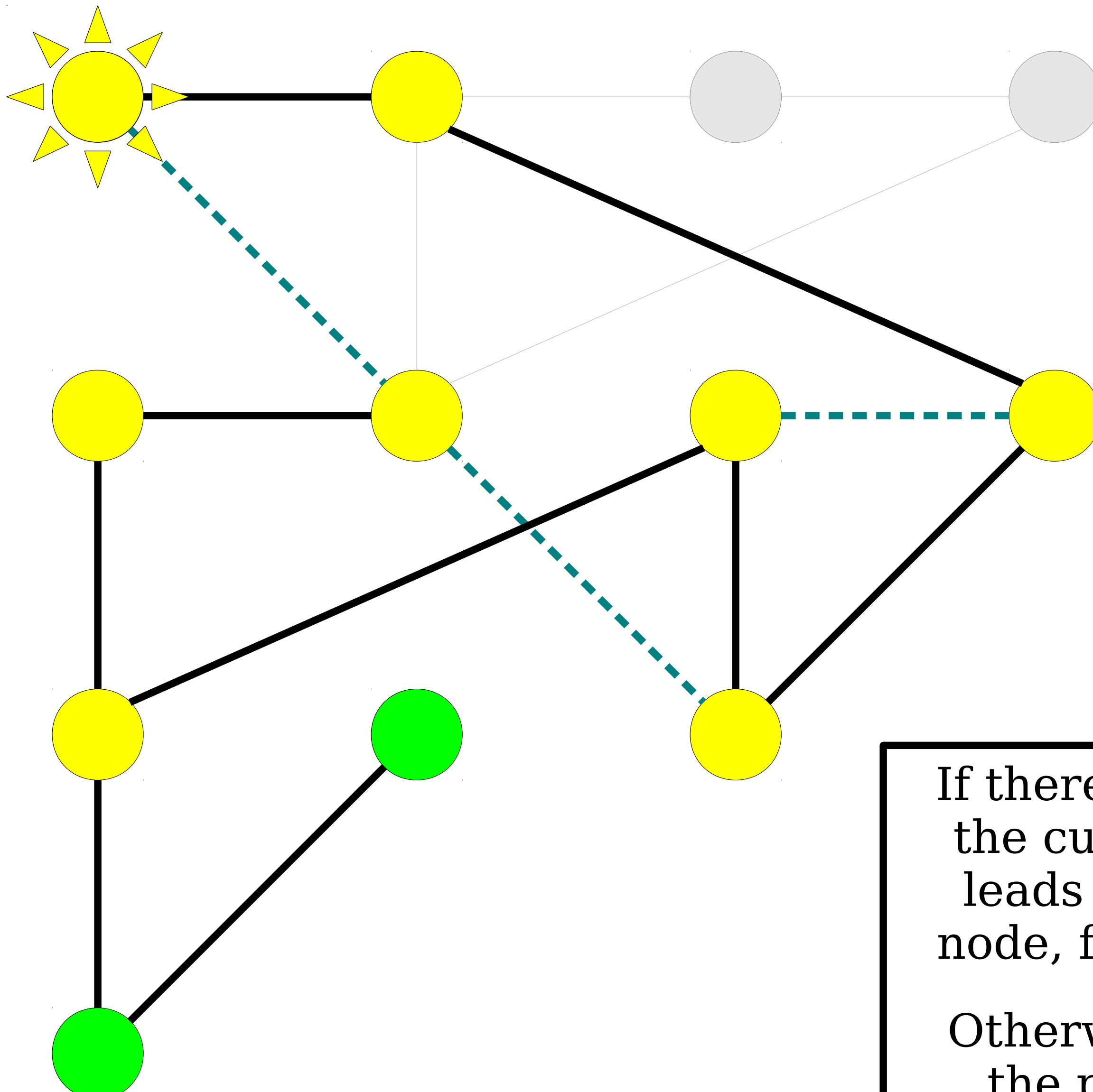
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

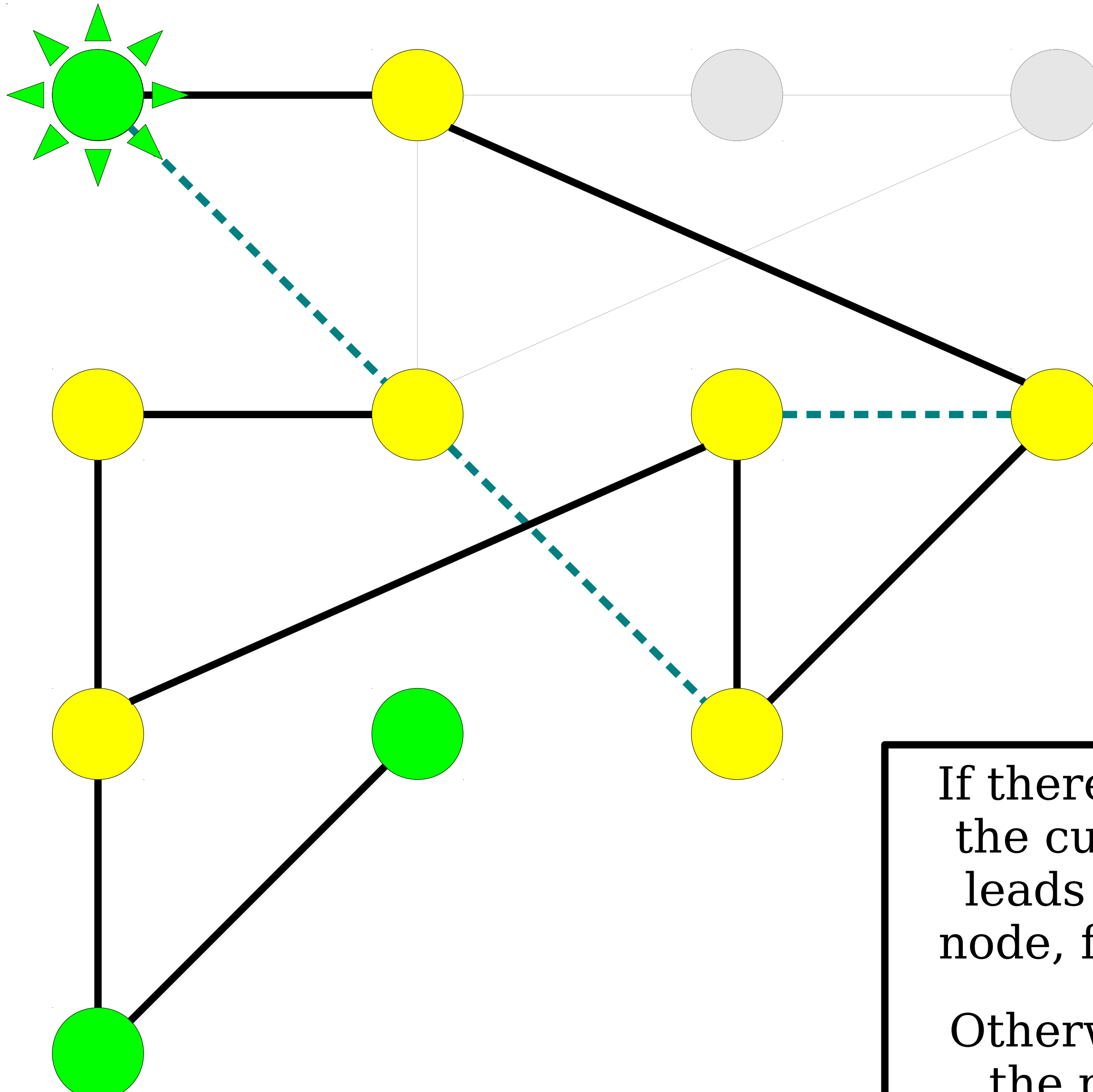
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

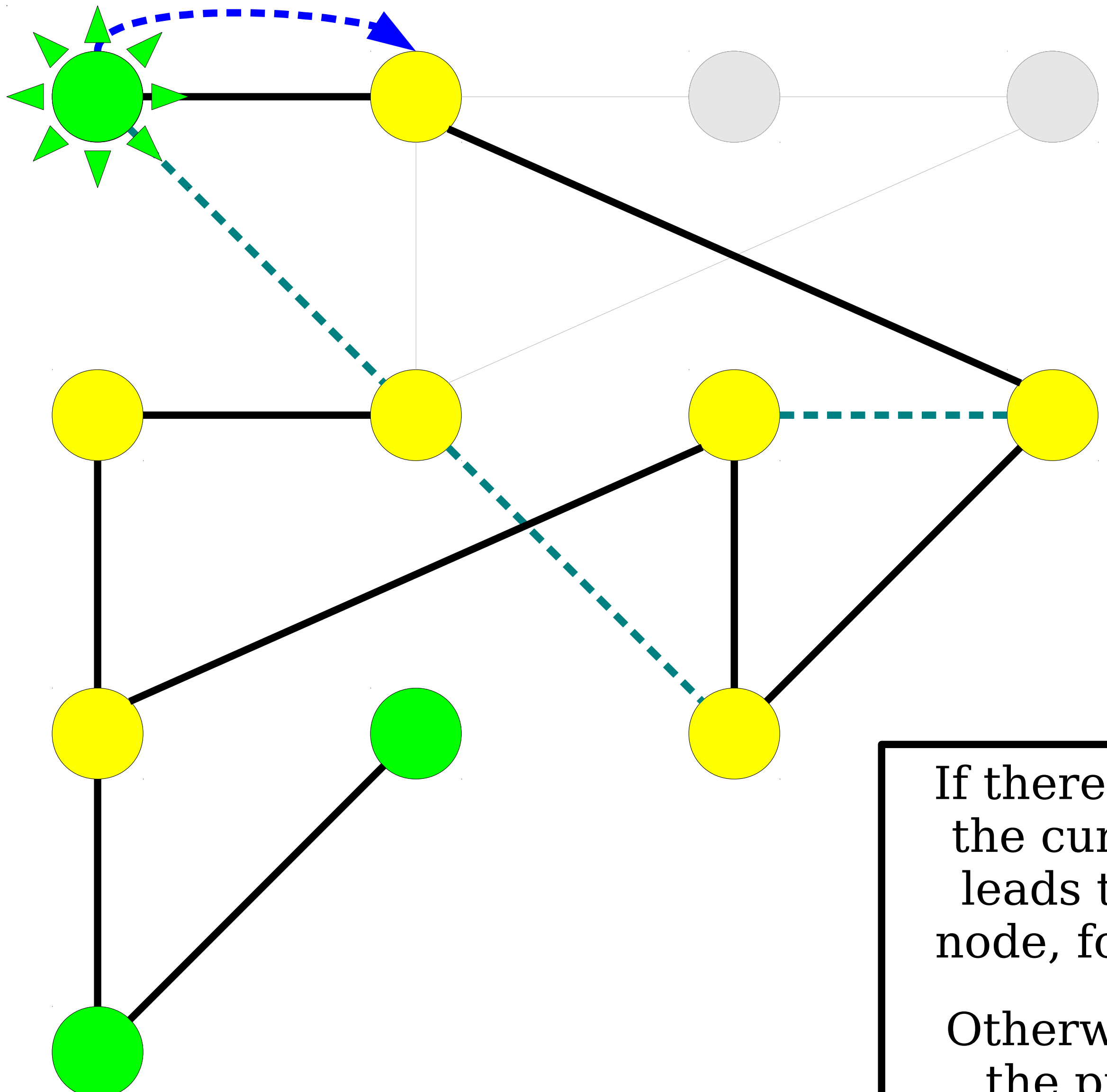
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

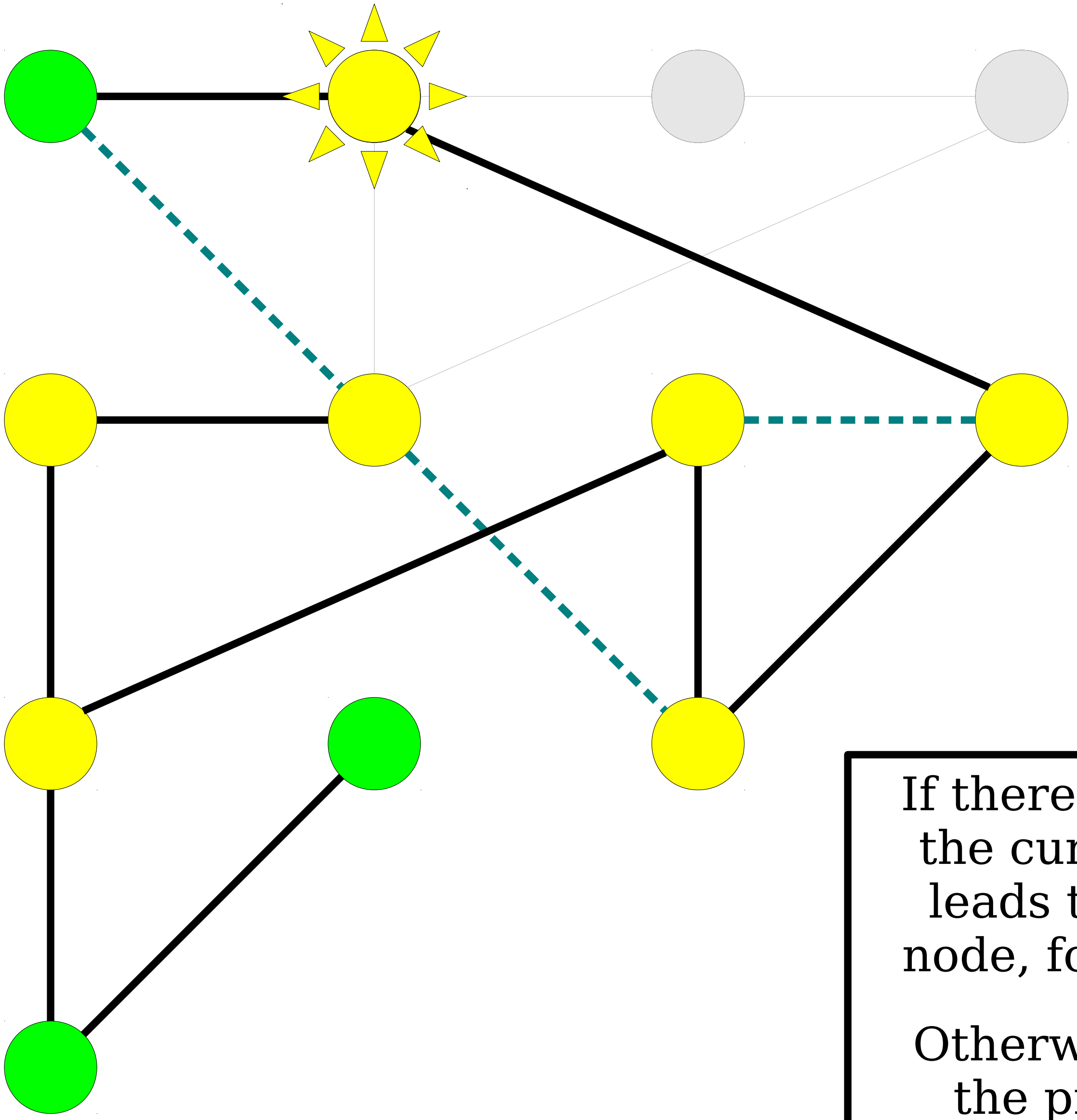
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

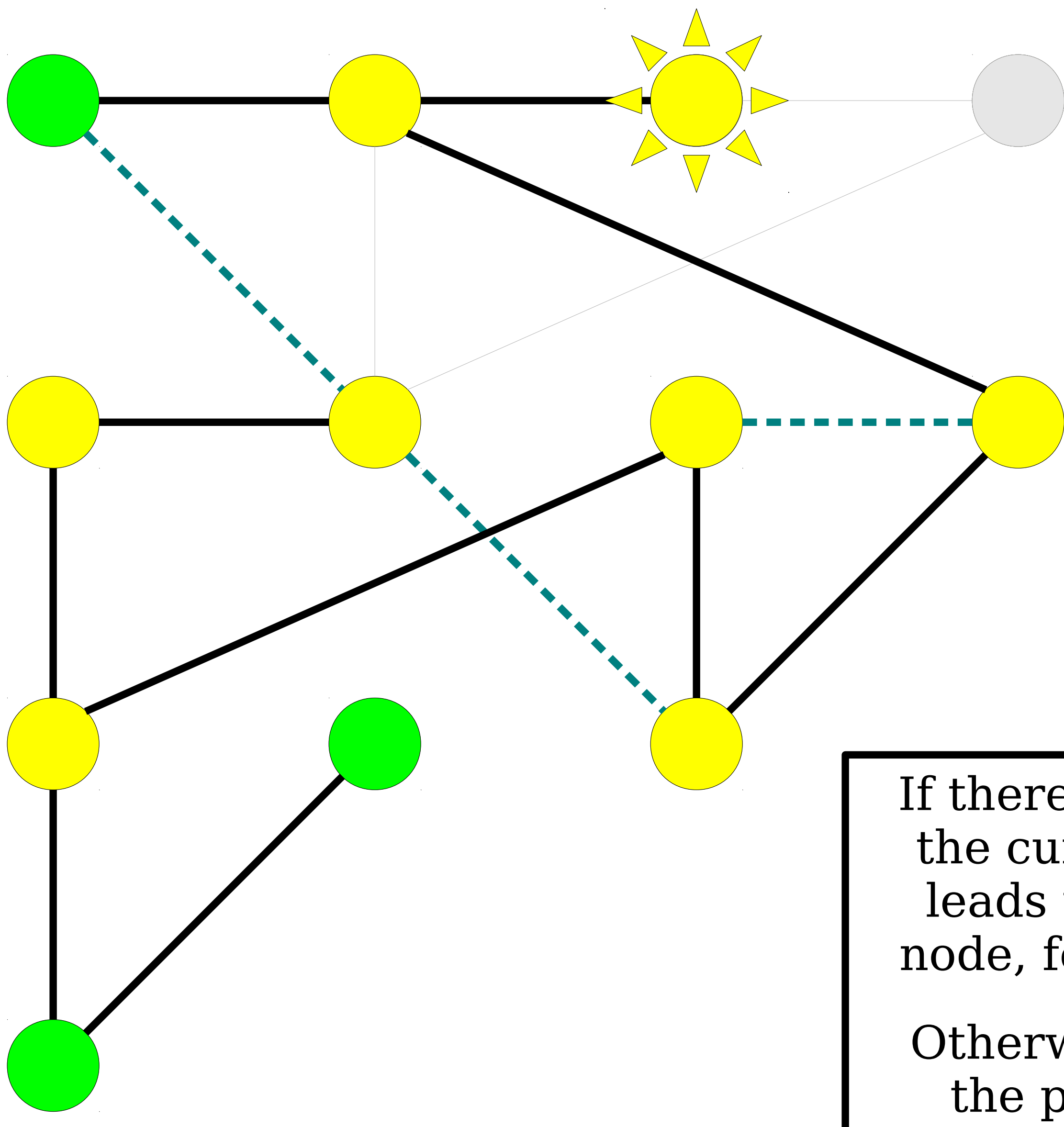
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

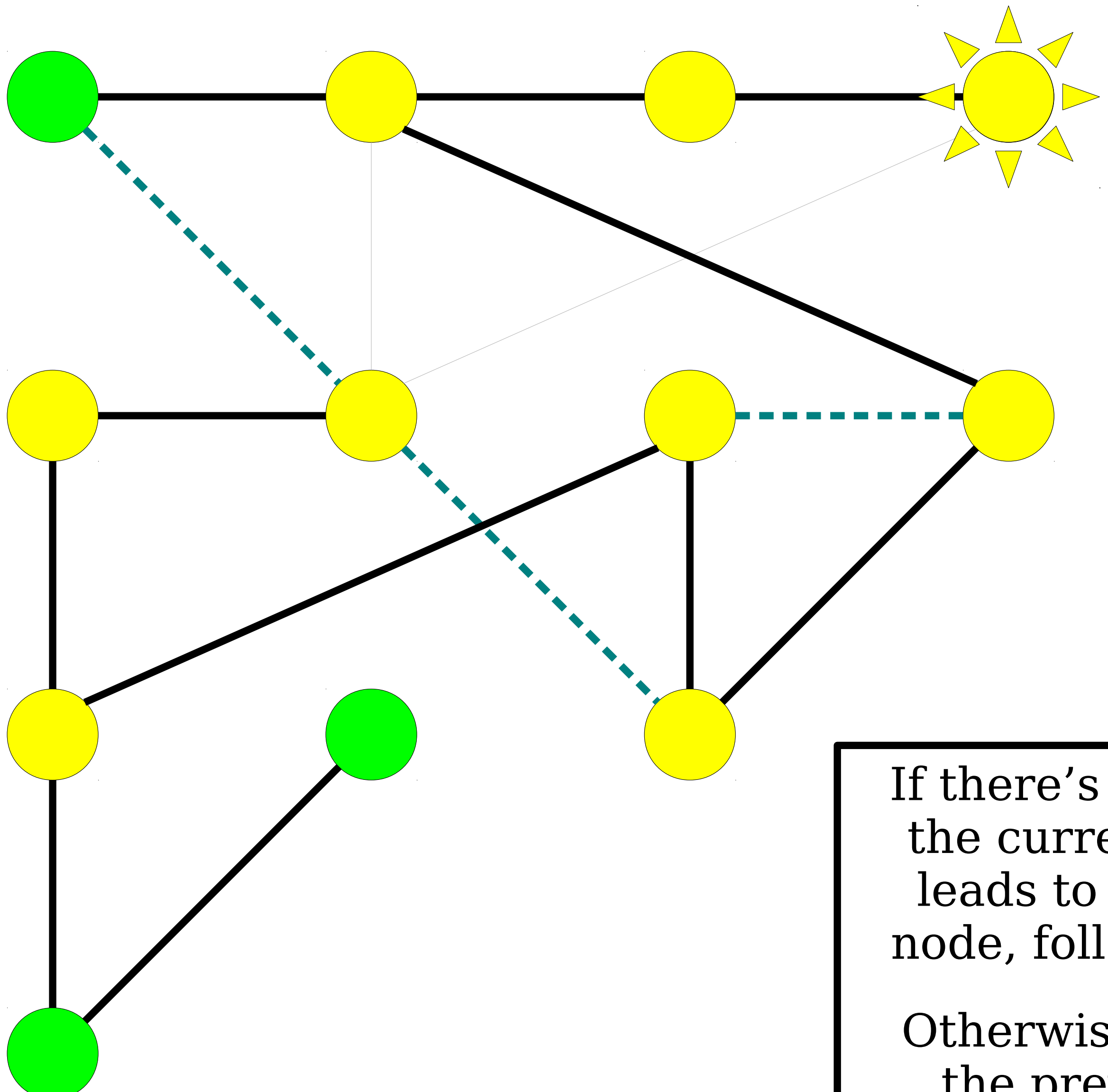
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

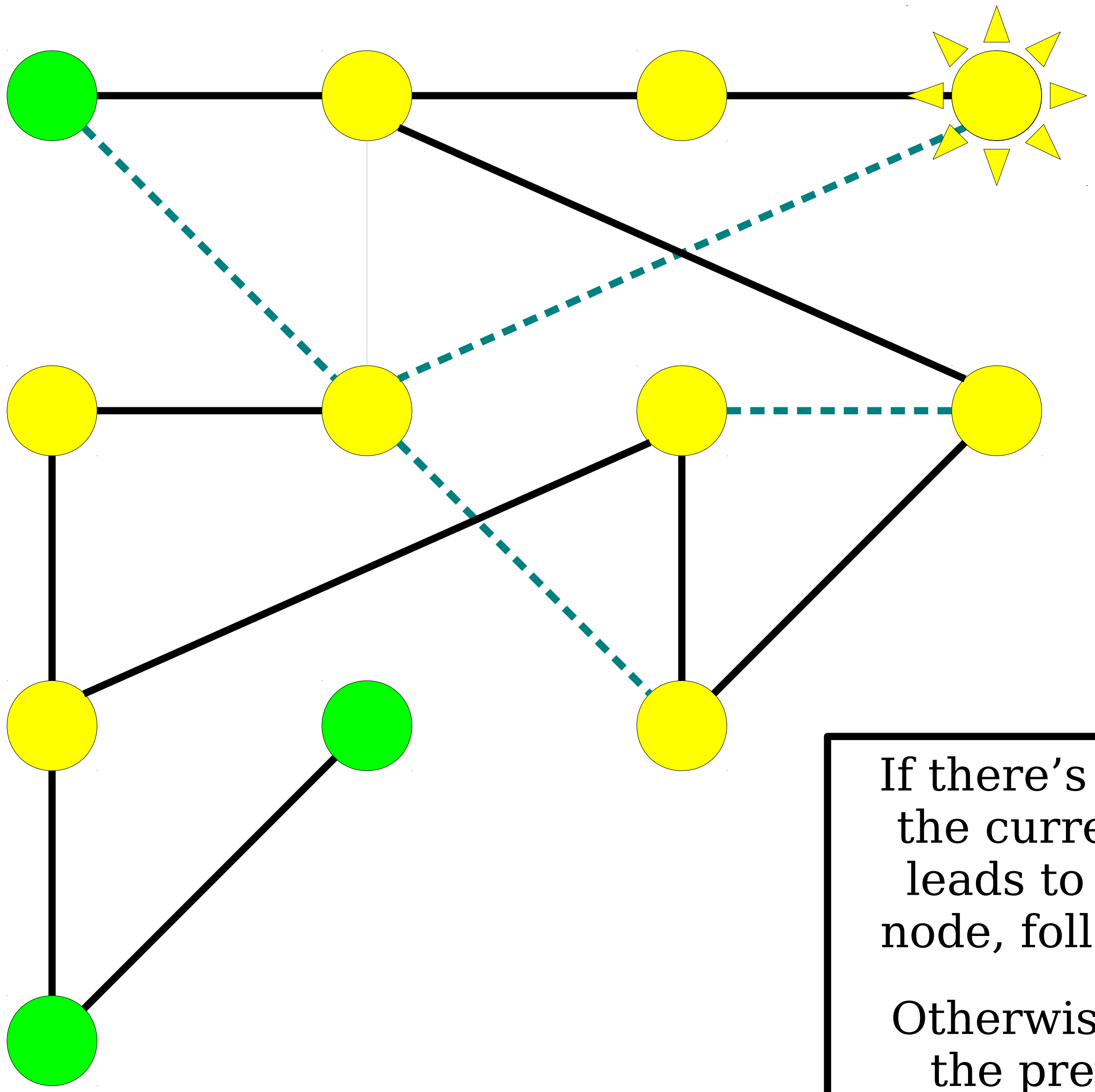
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

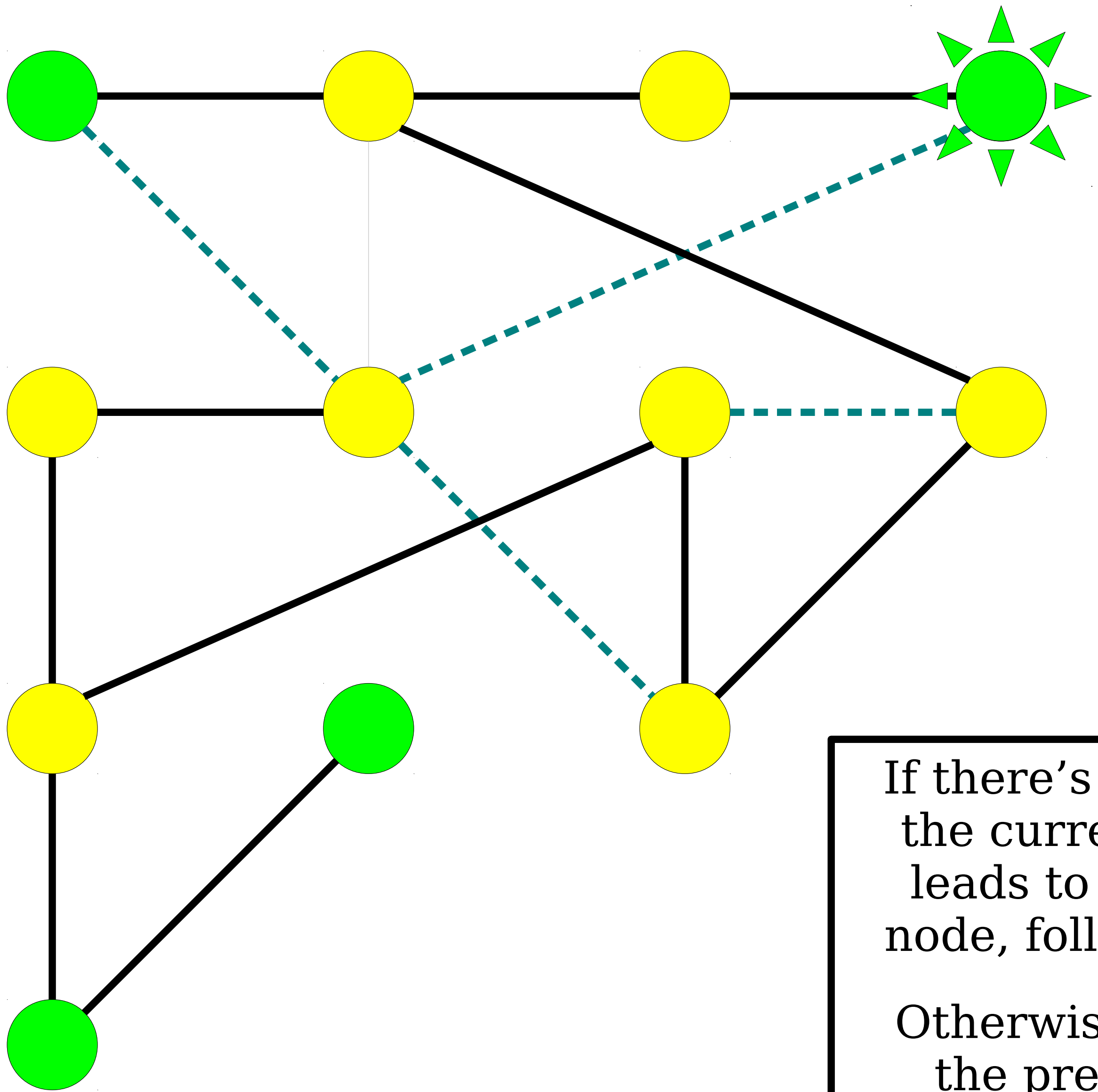
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

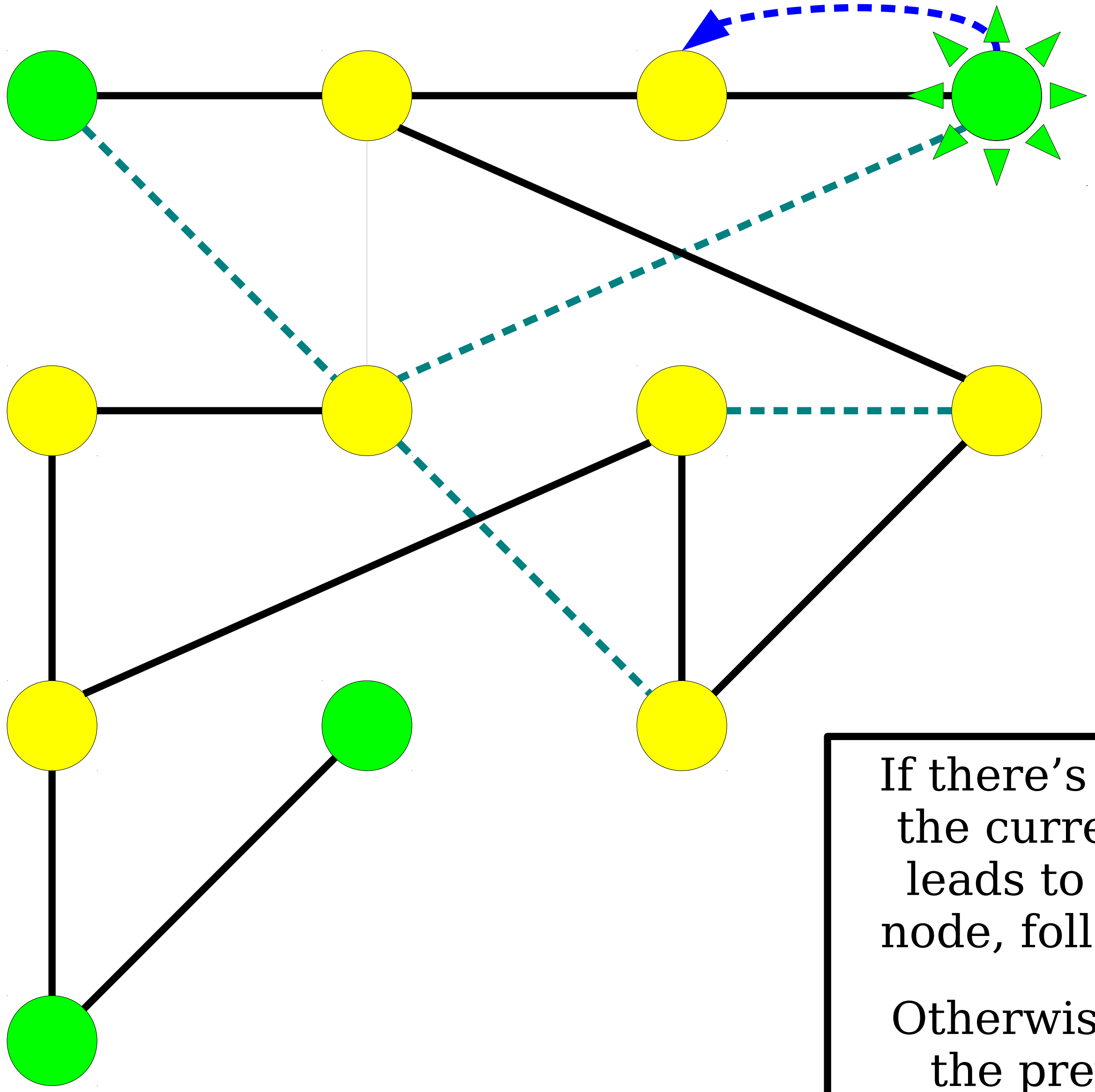
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

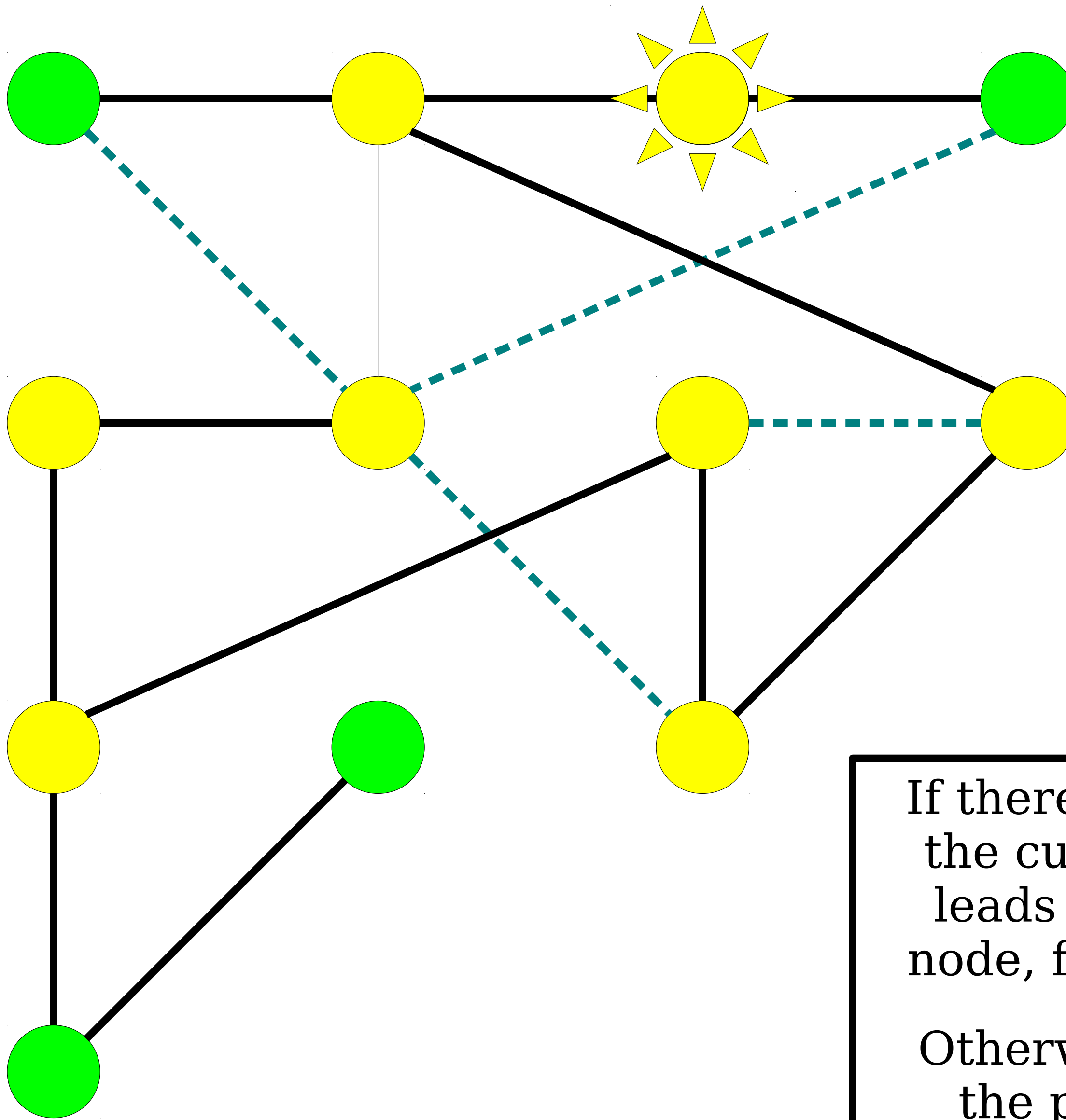
Depth-First Search



If there’s an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

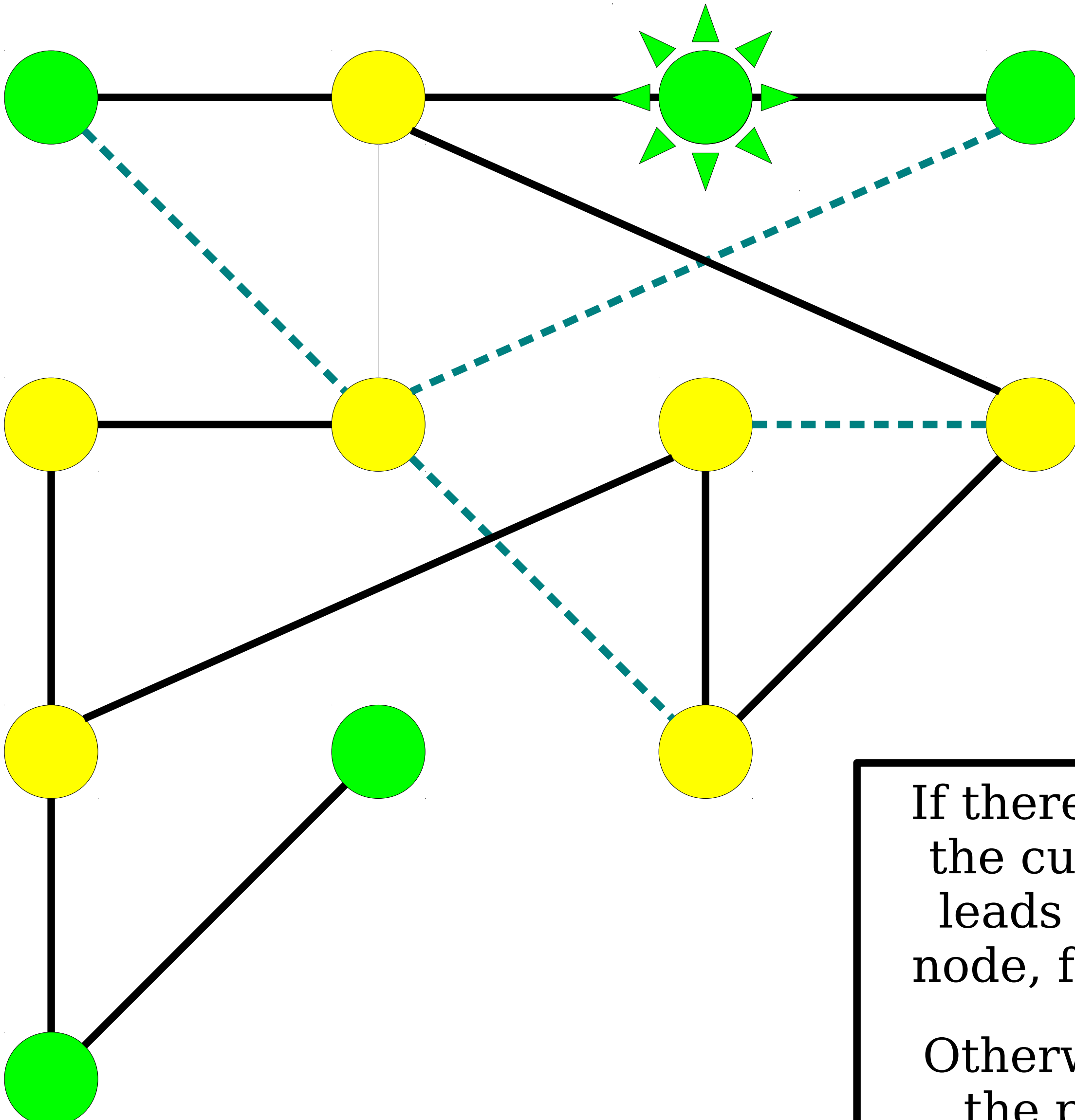
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

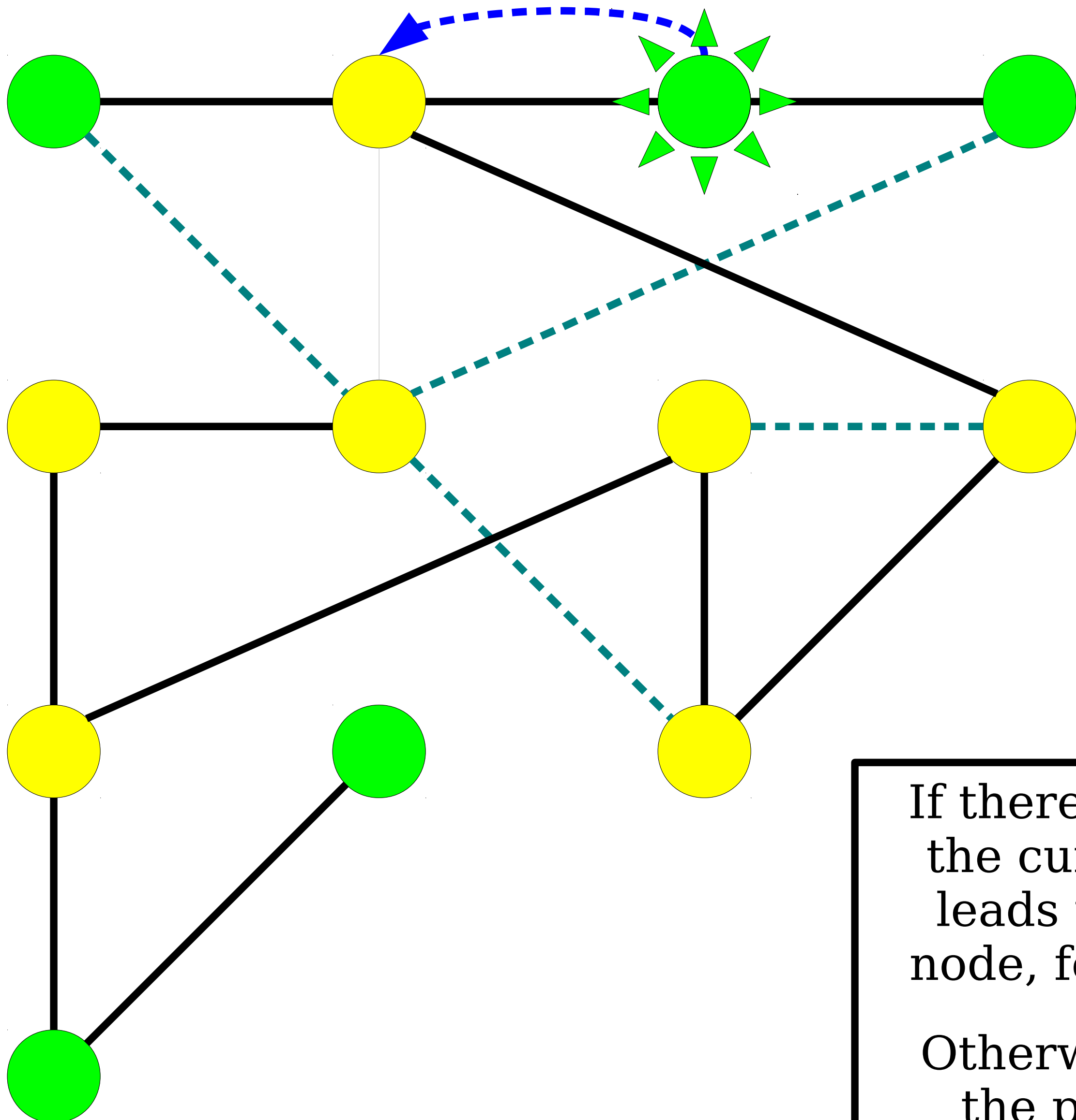
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

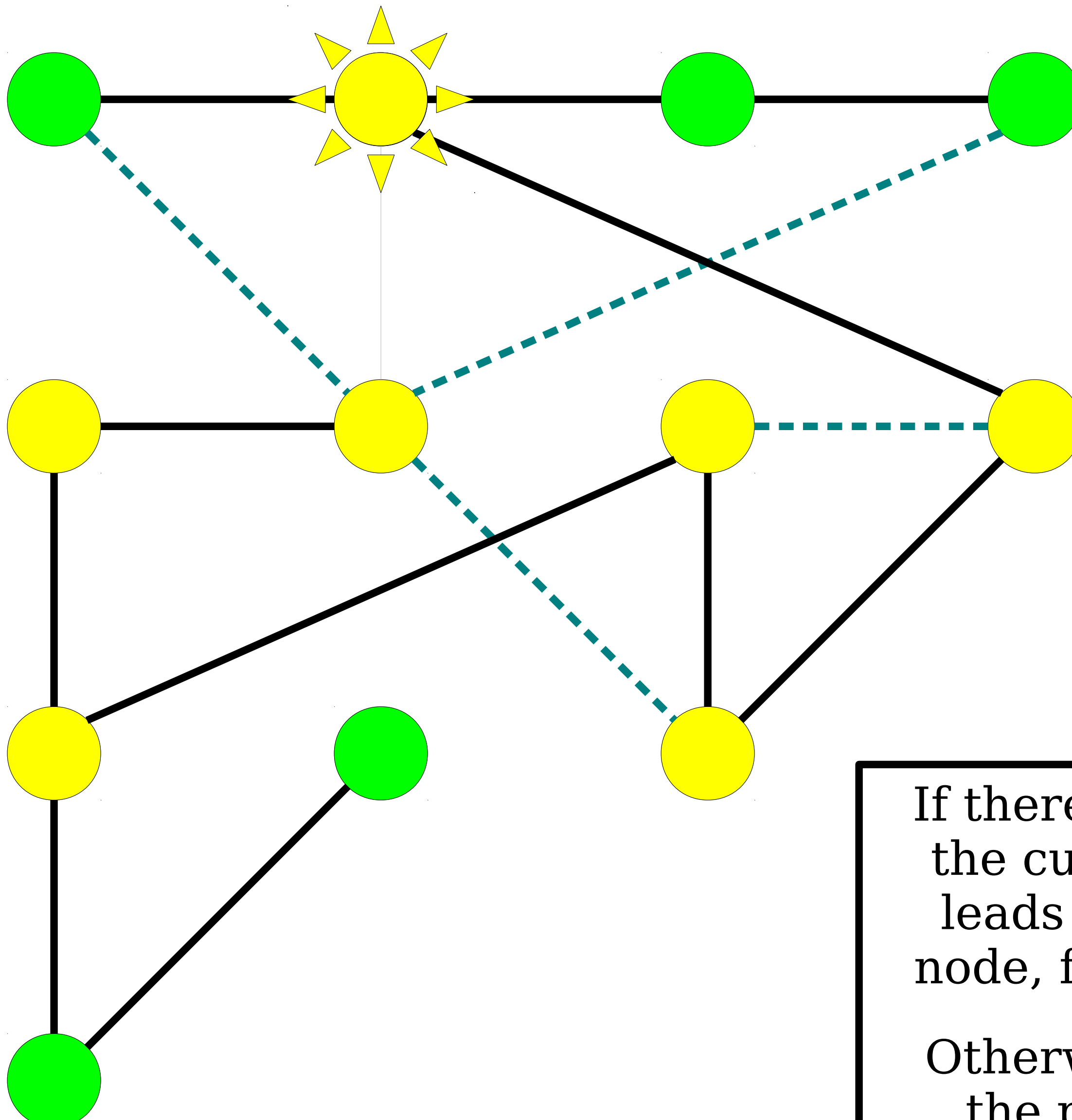
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

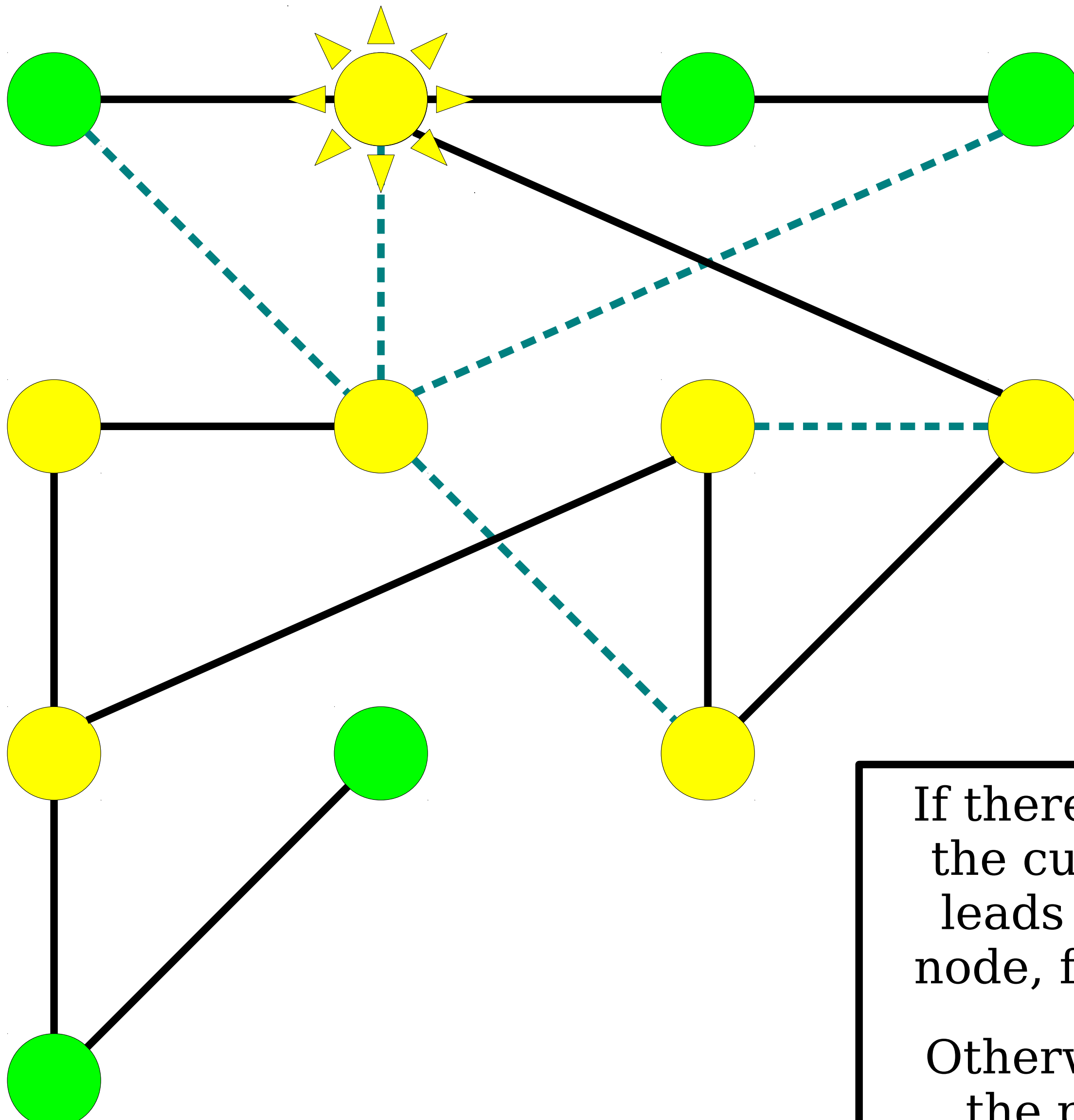
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

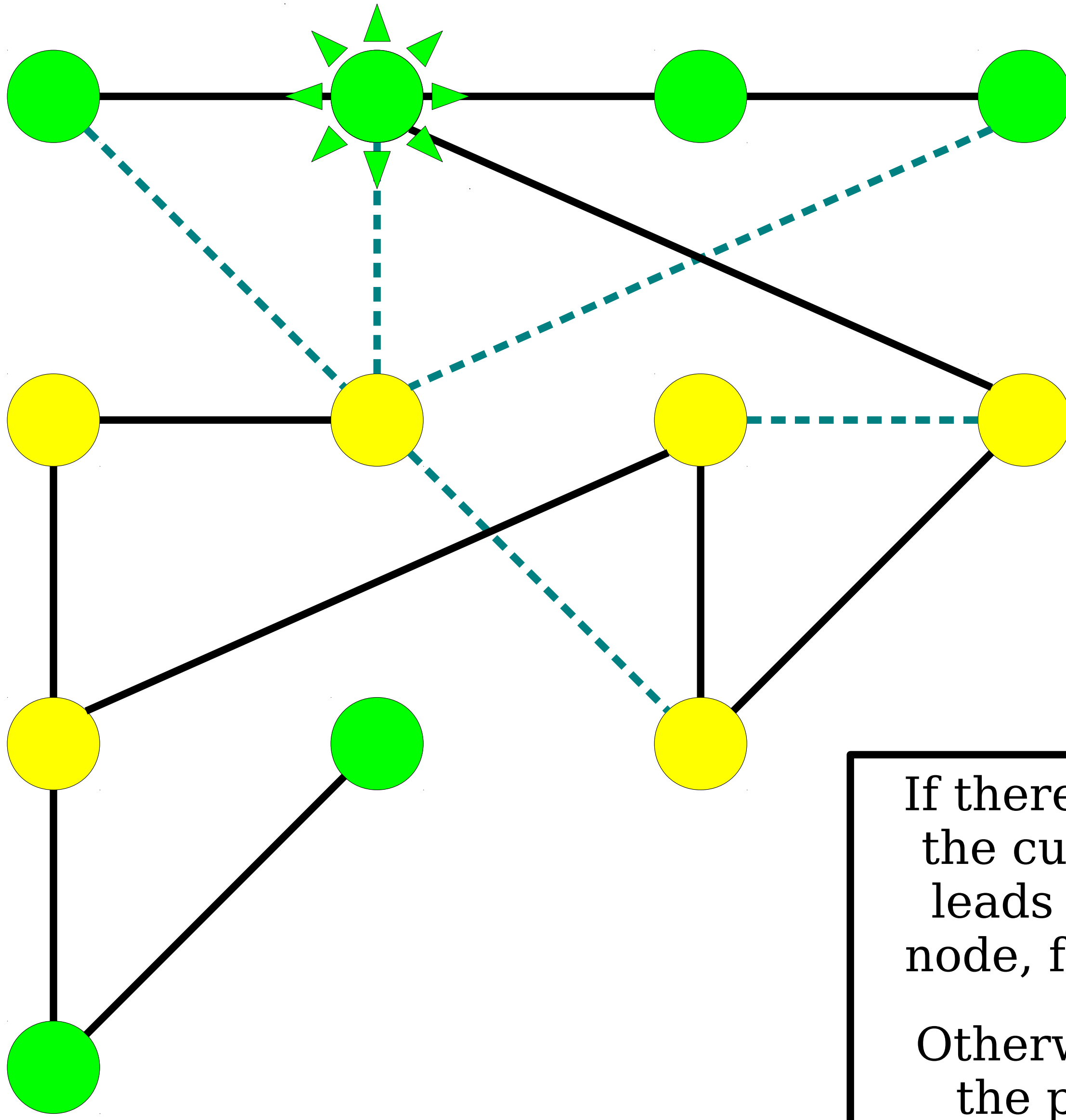
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

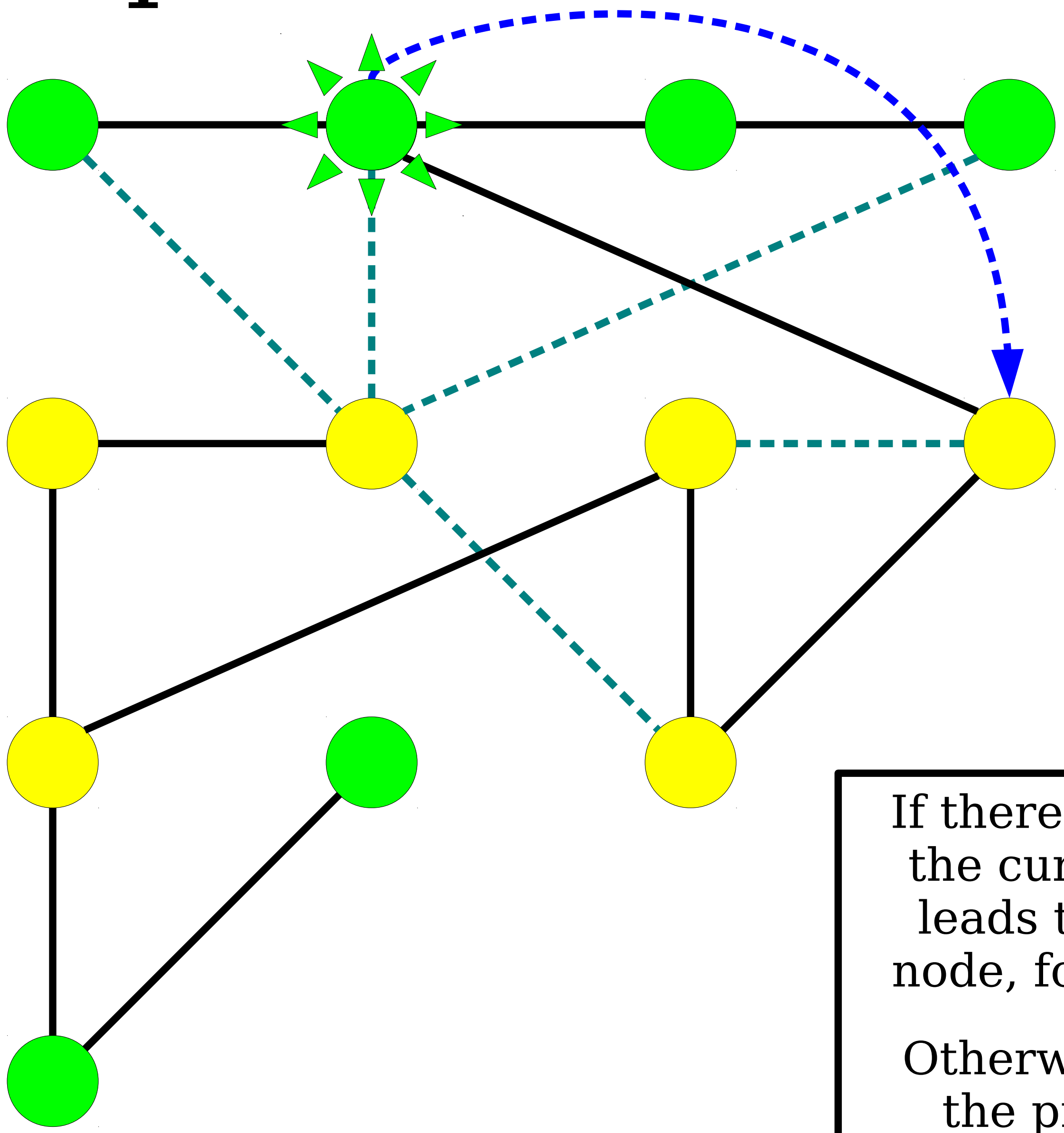
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

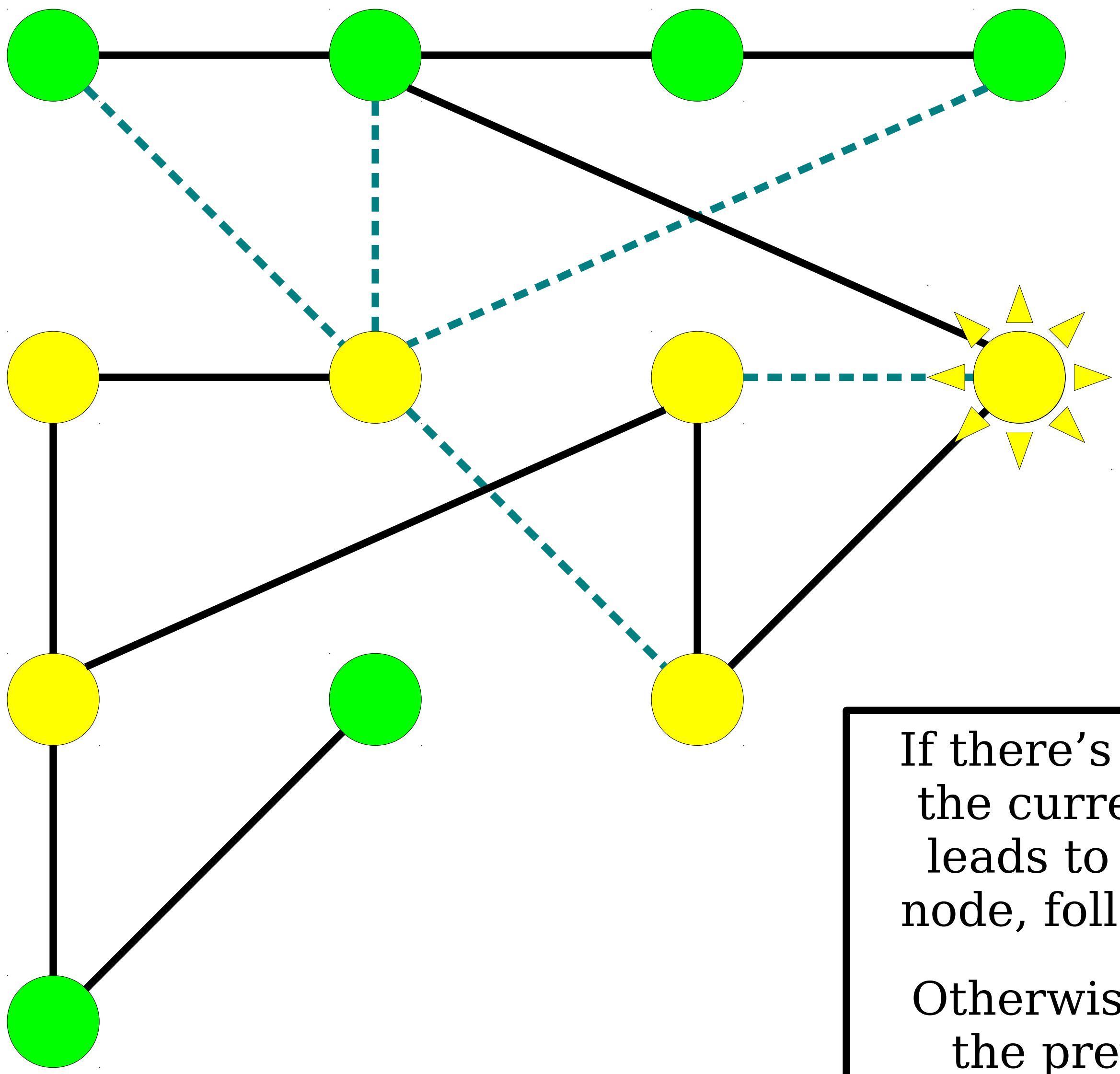
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

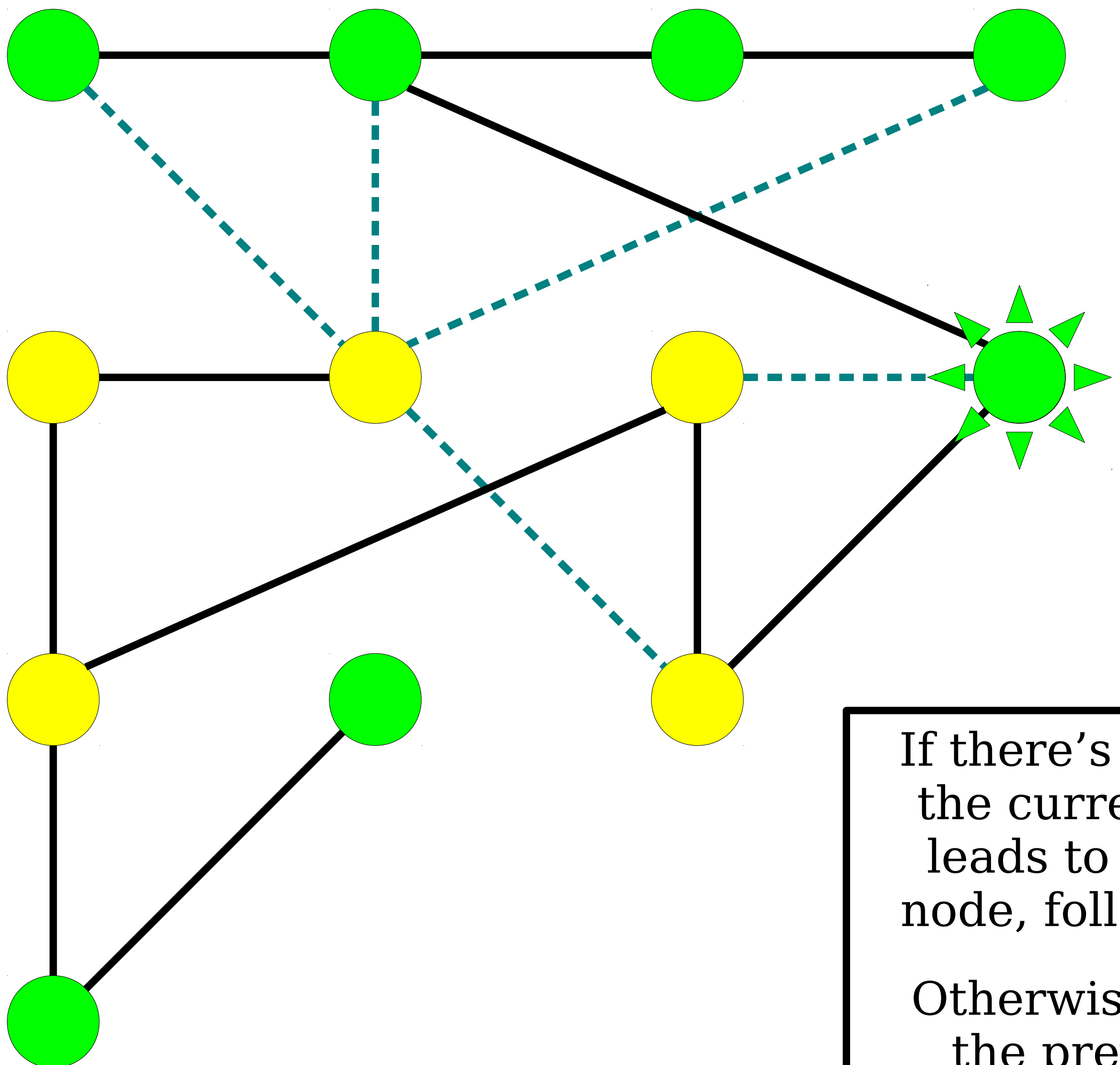
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

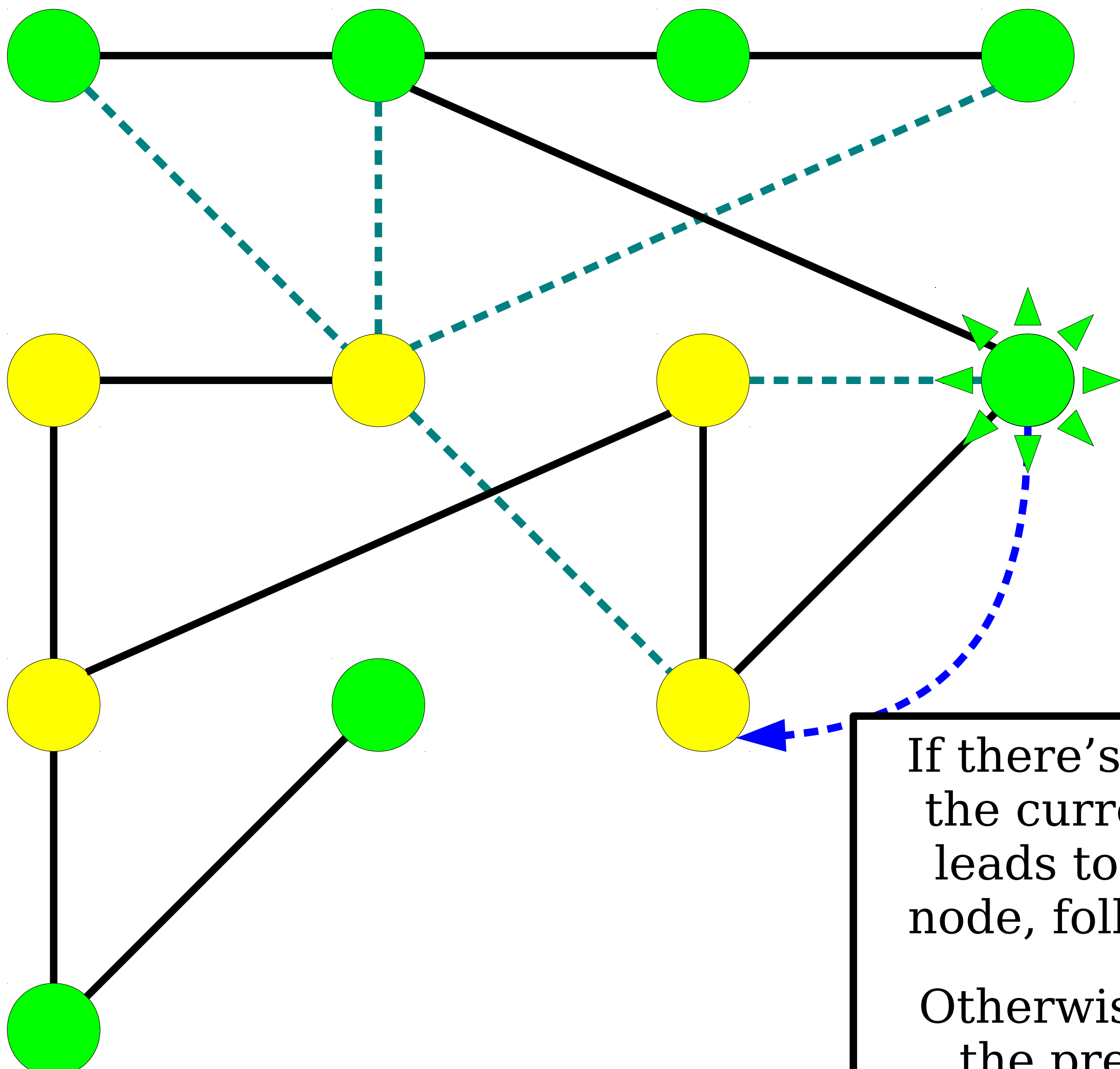
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

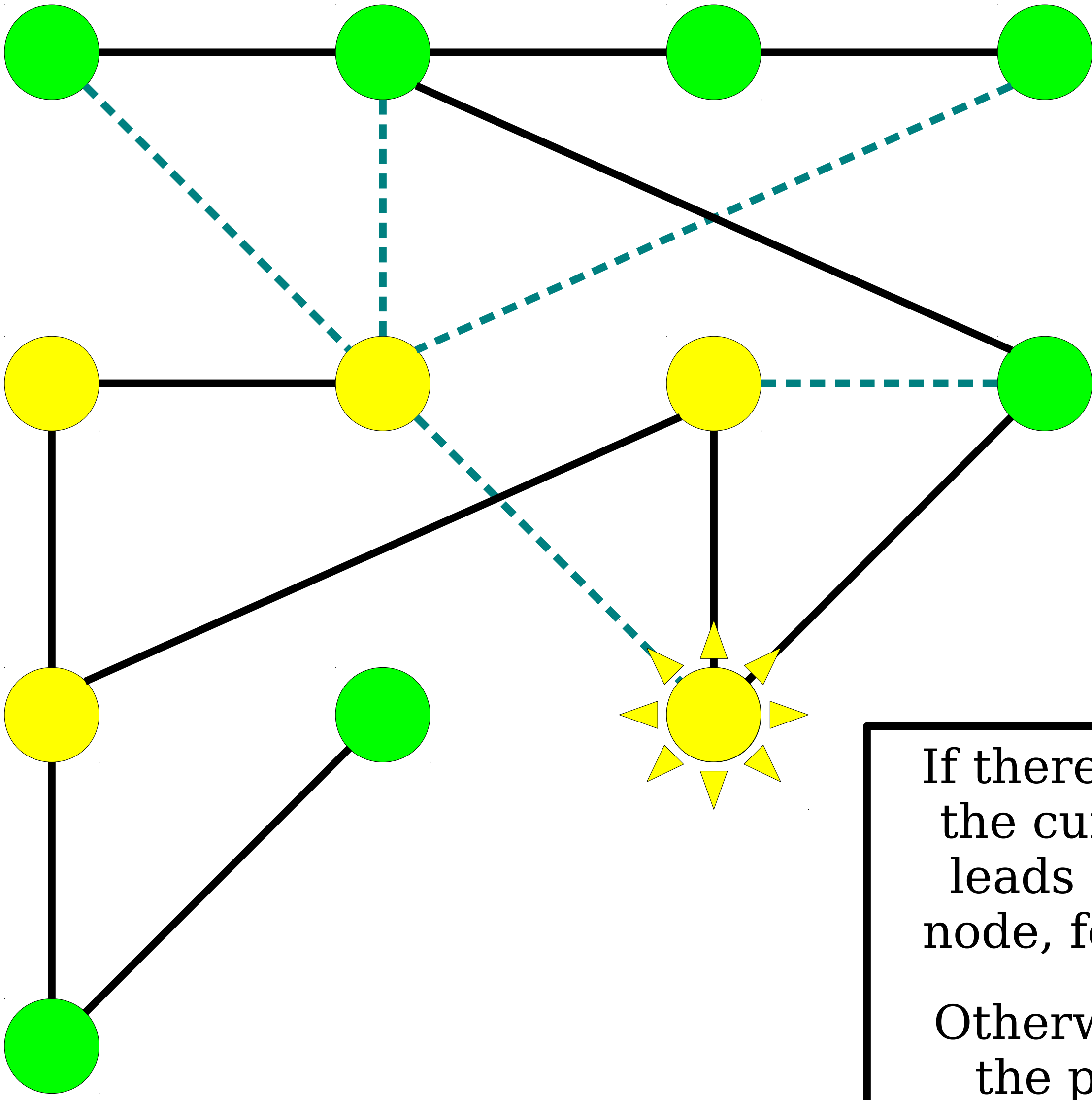
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

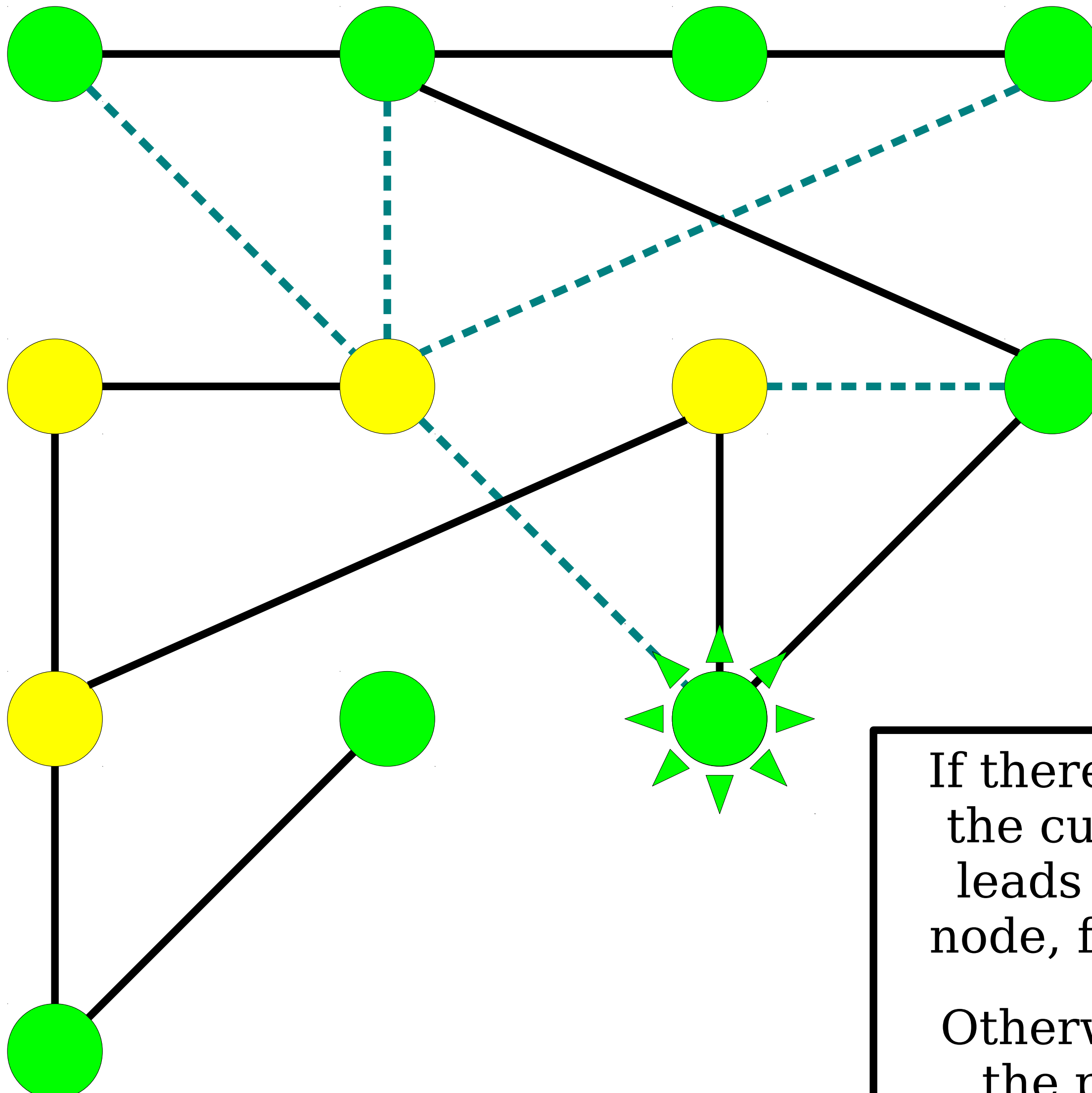
Depth-First Search



If there’s an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

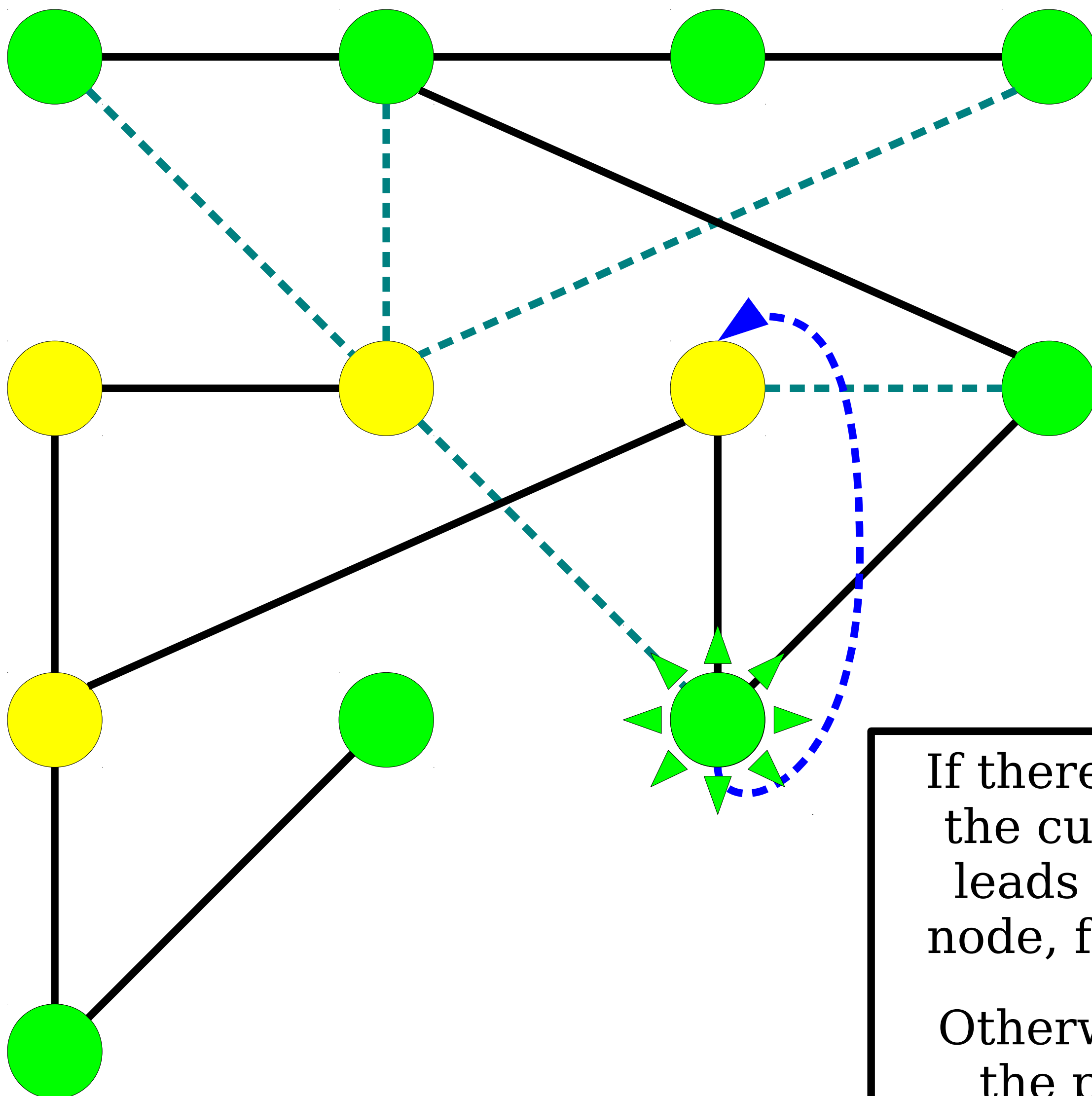
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

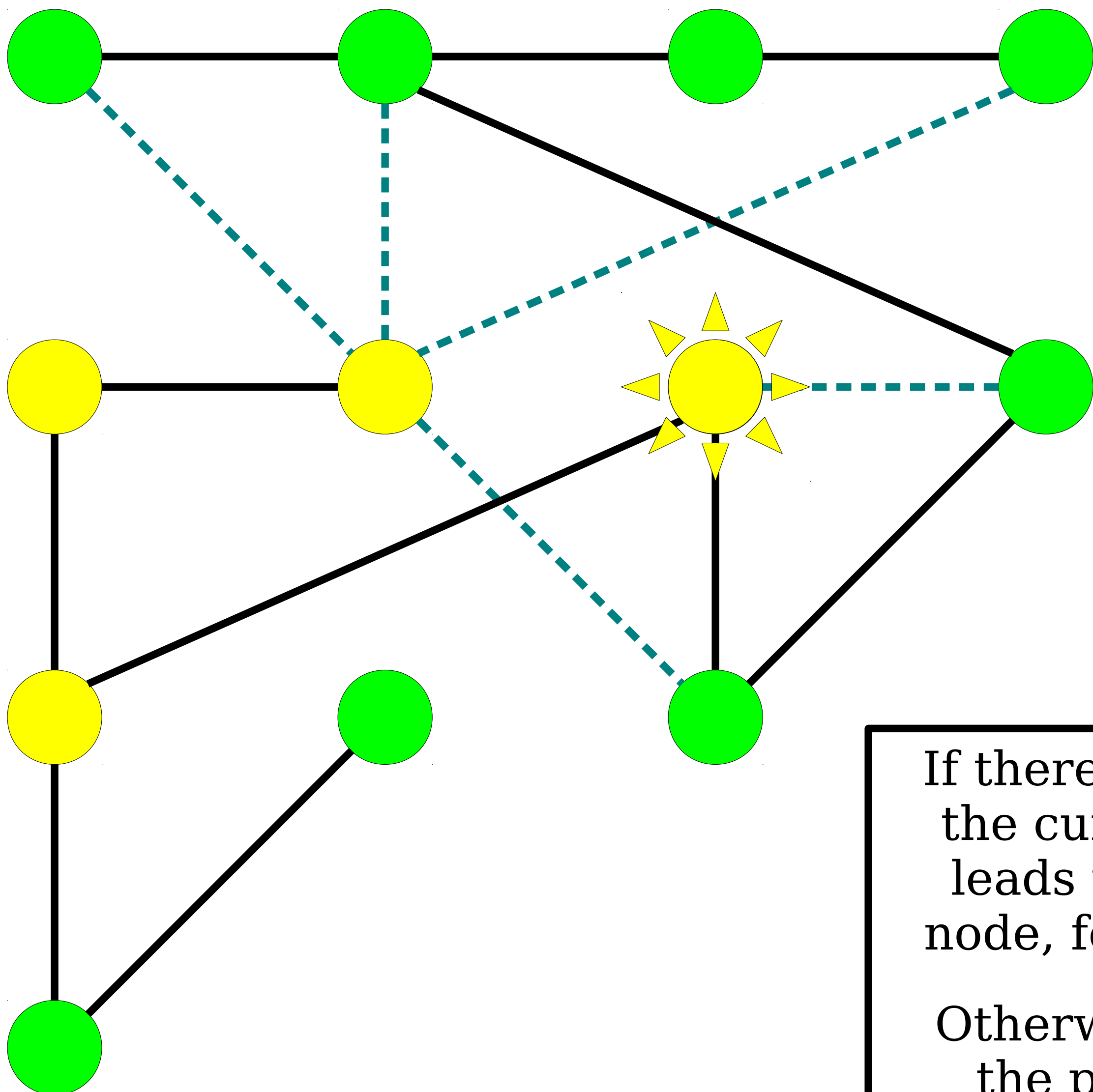
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

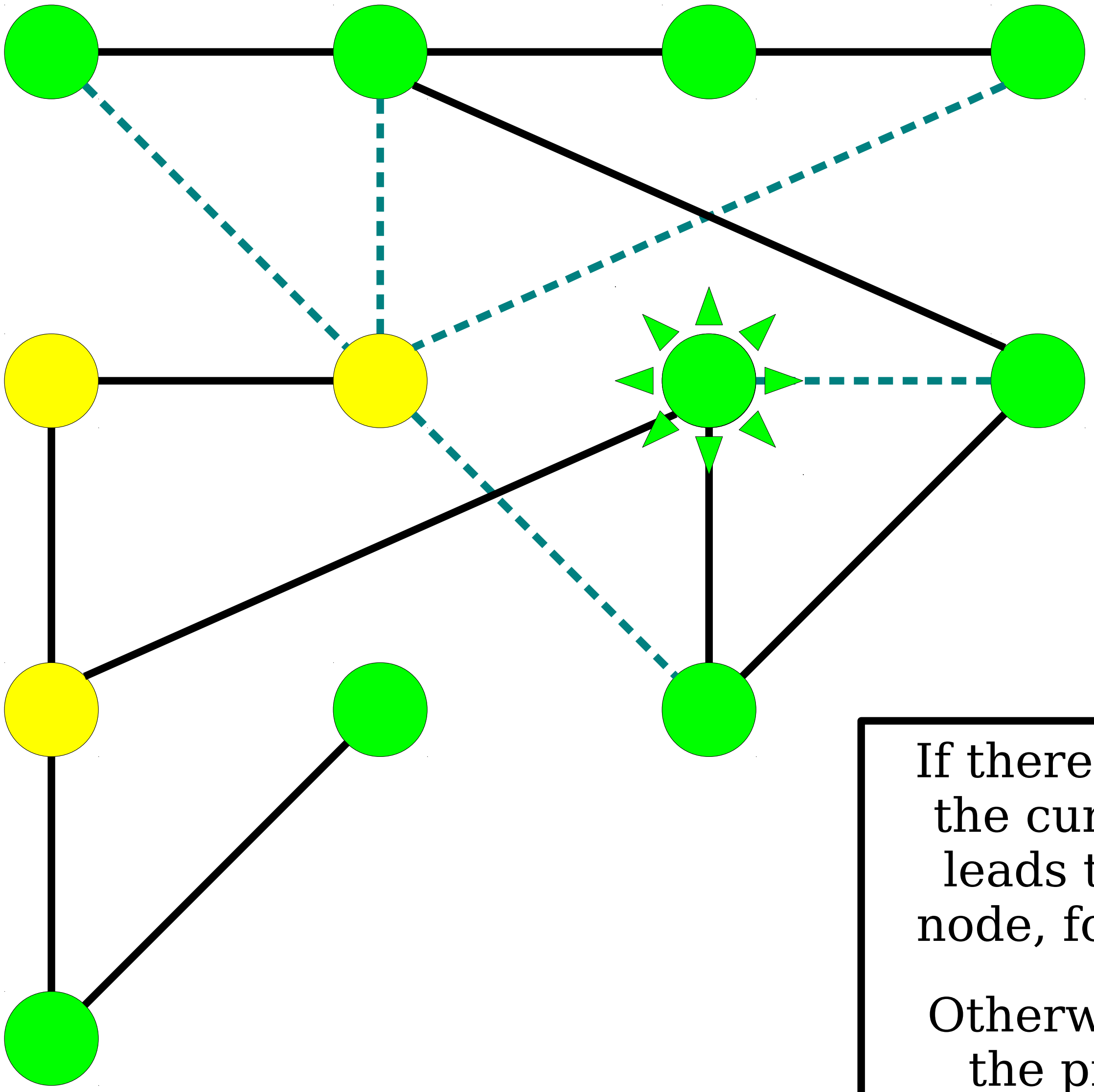
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

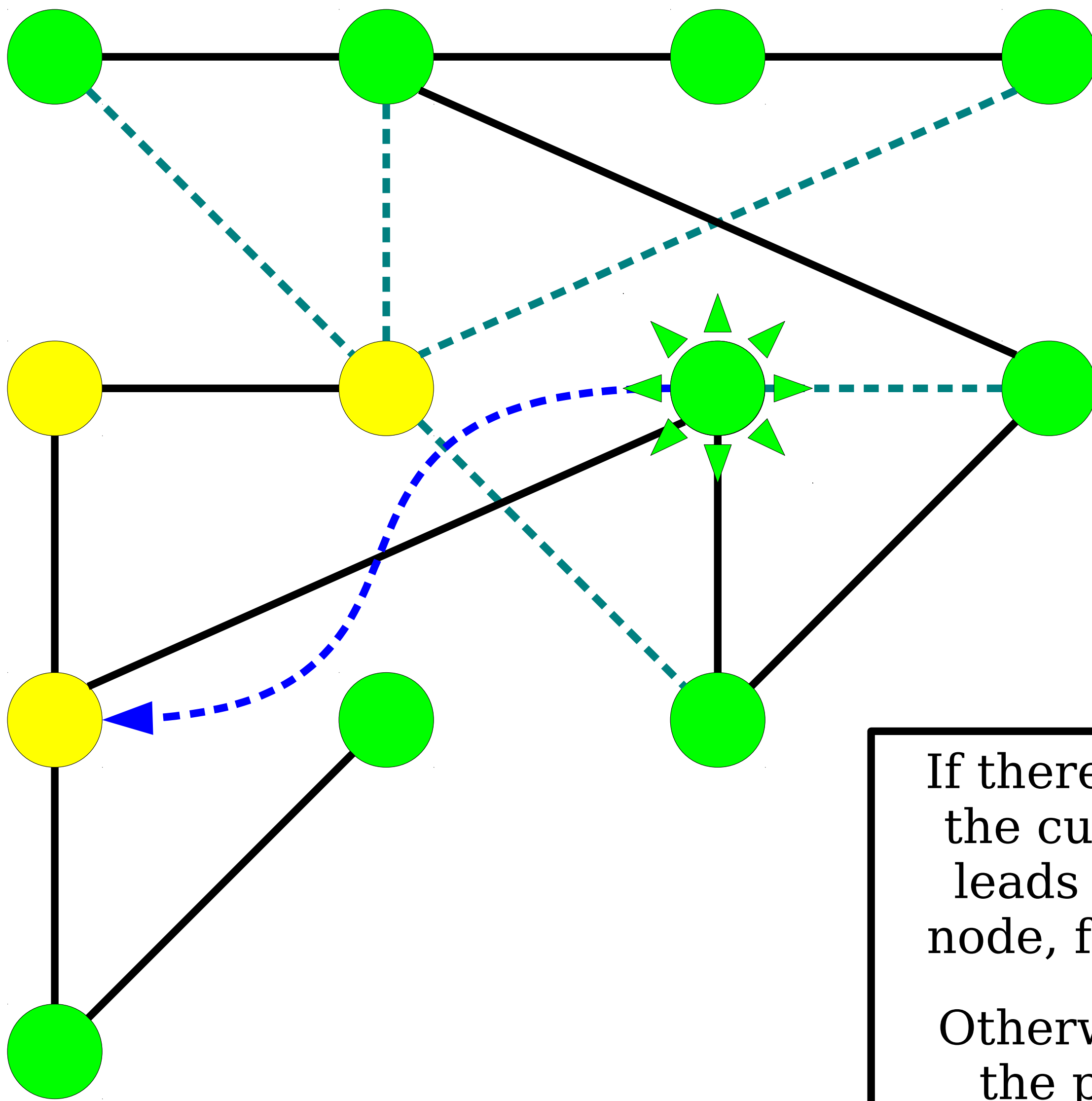
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

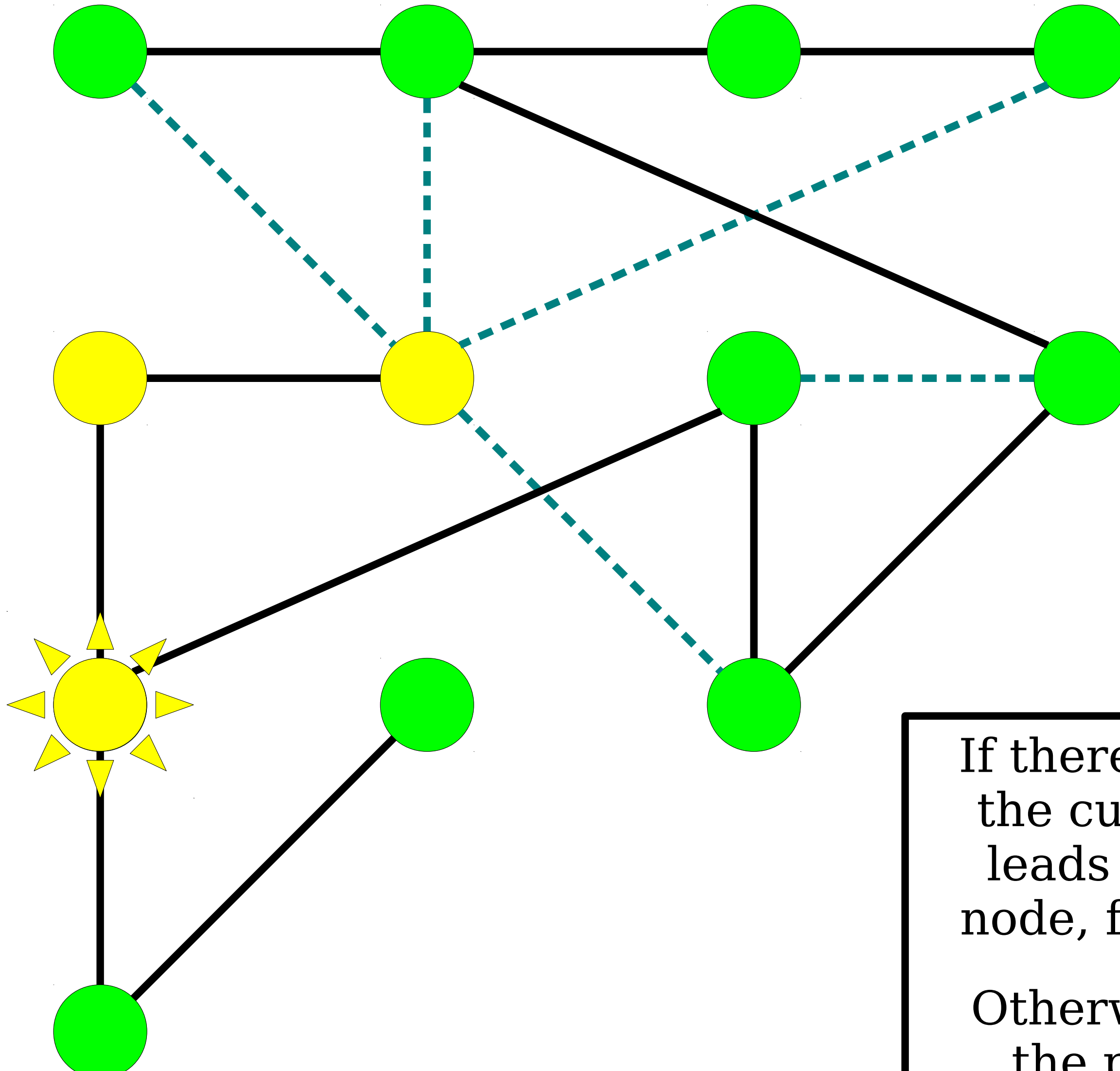
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

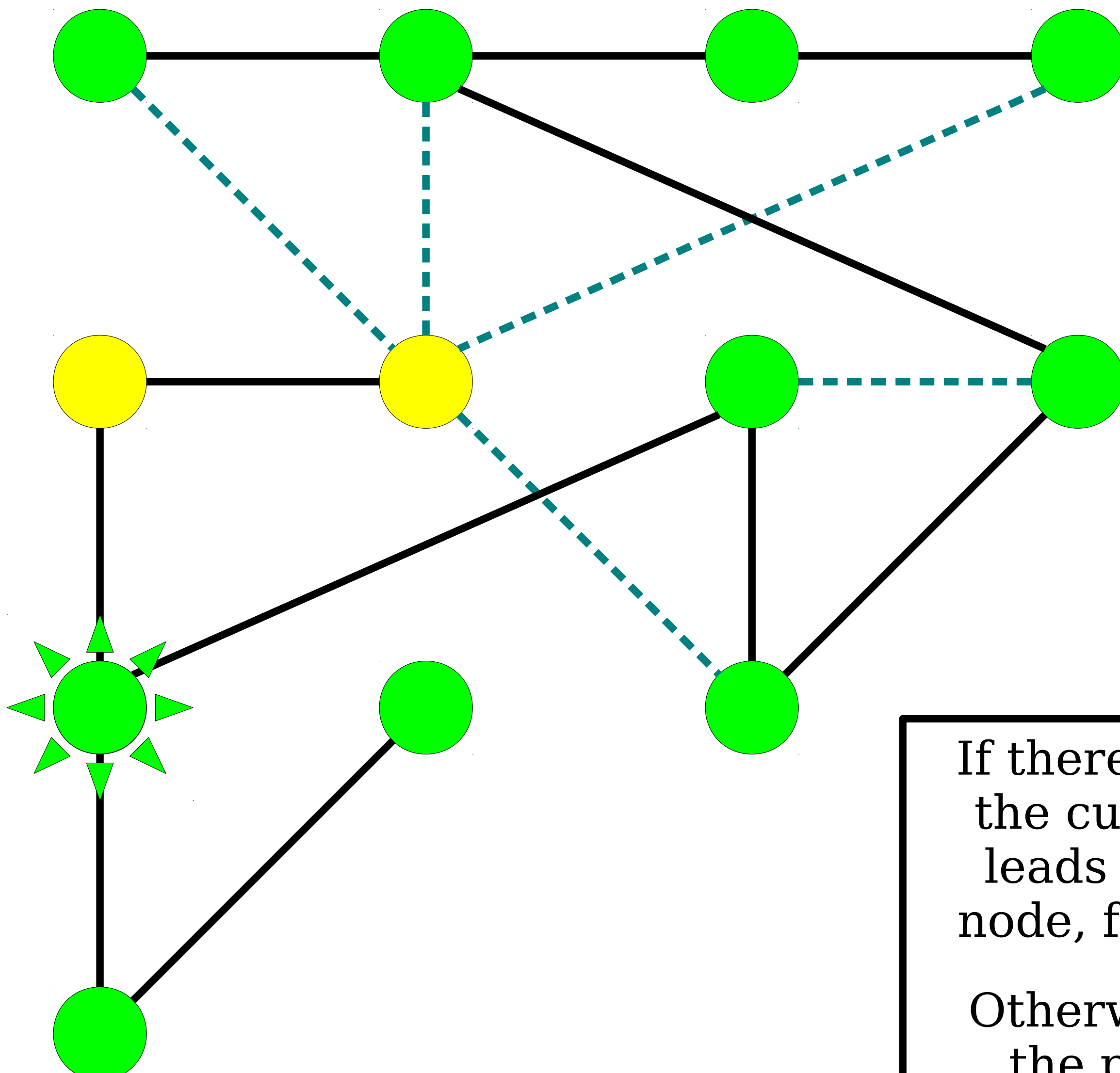
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

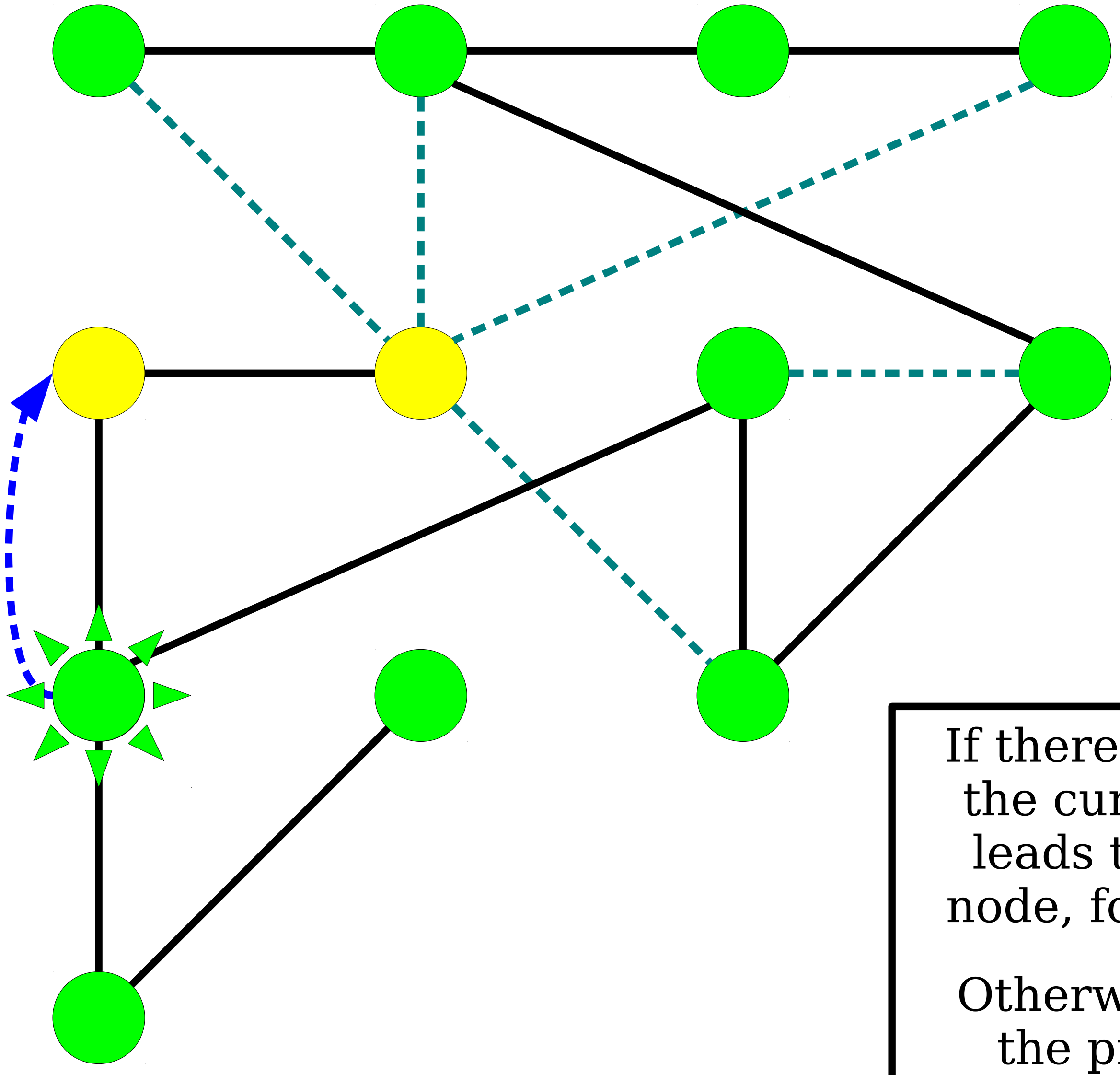
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

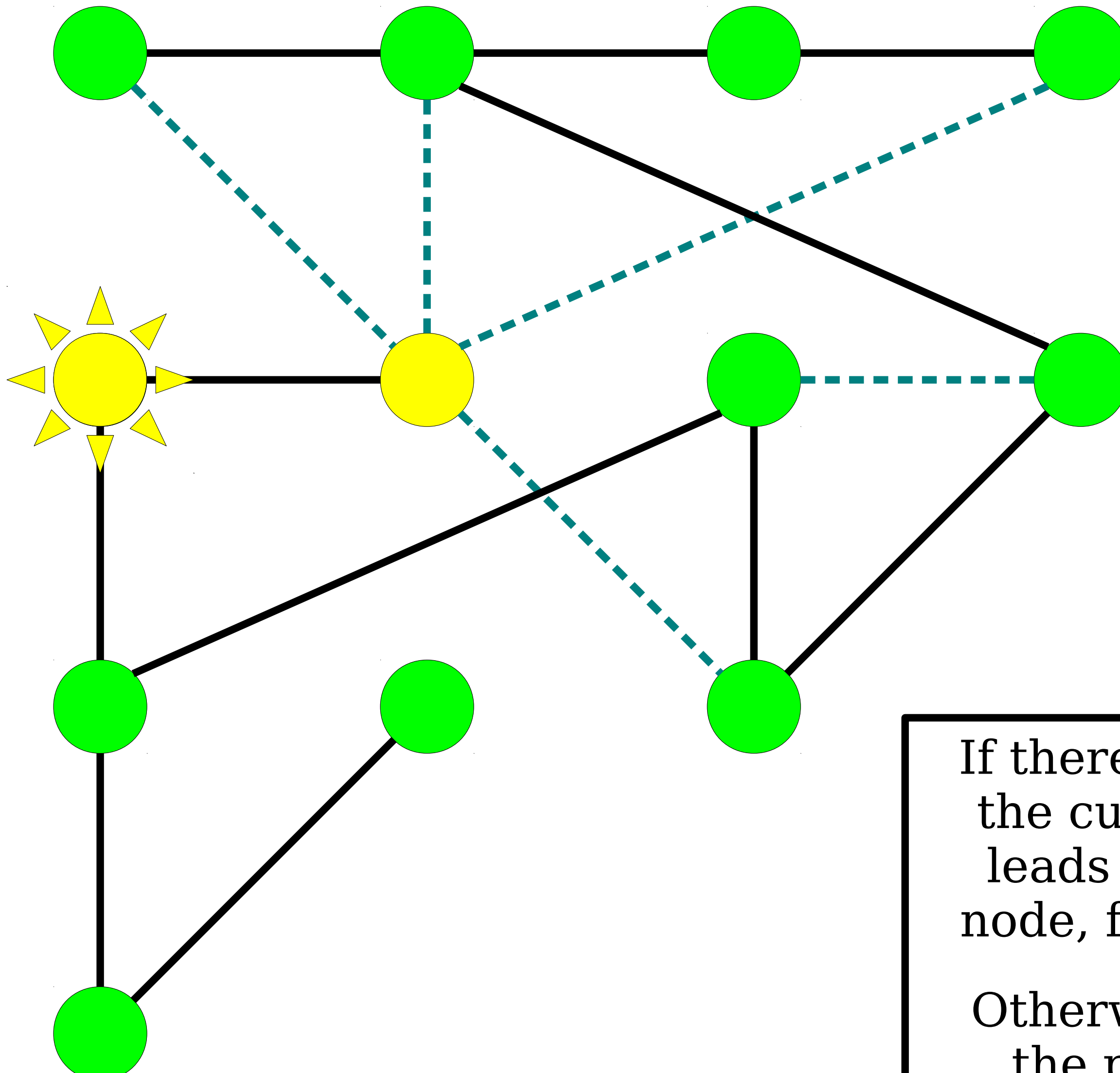
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

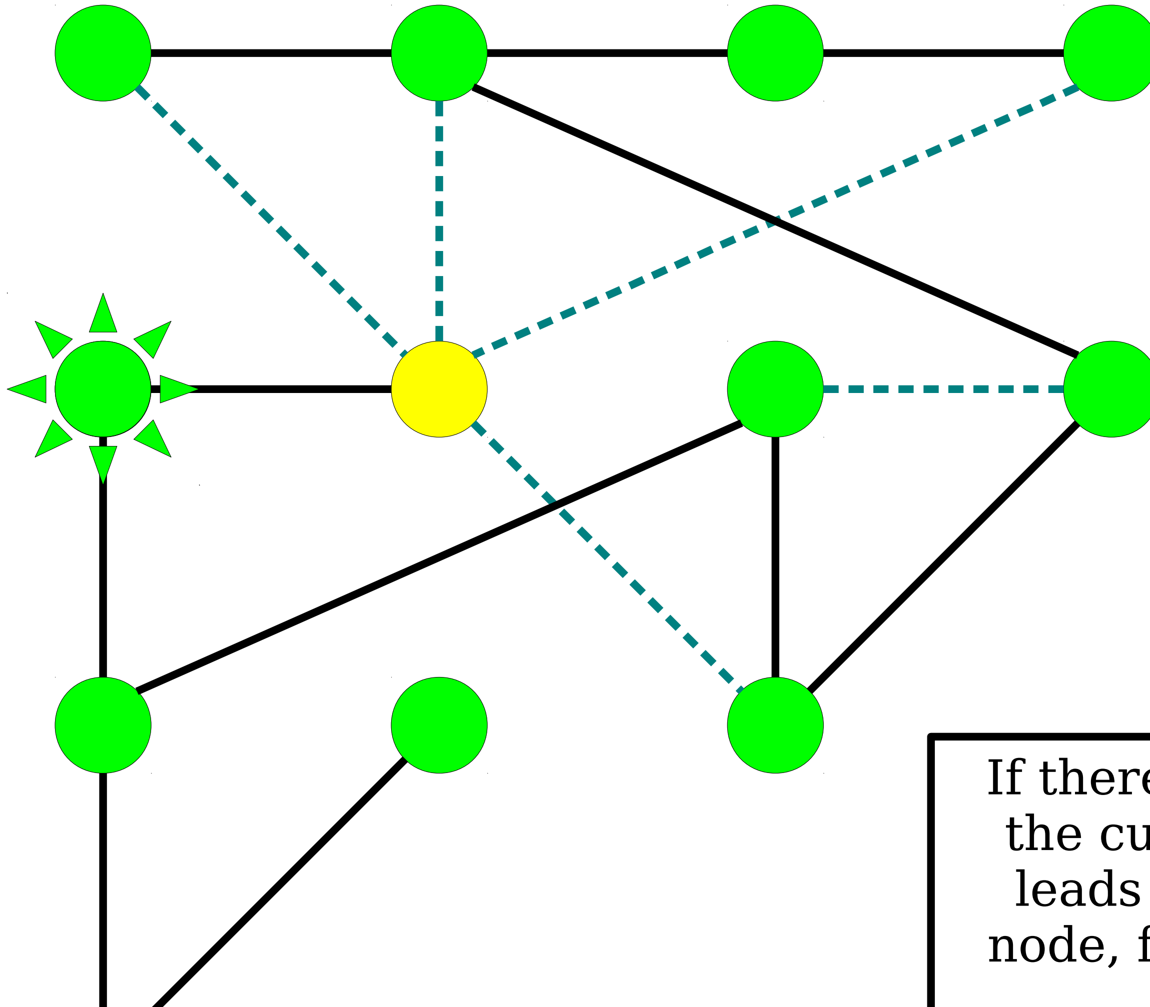
Depth-First Search



If there’s an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

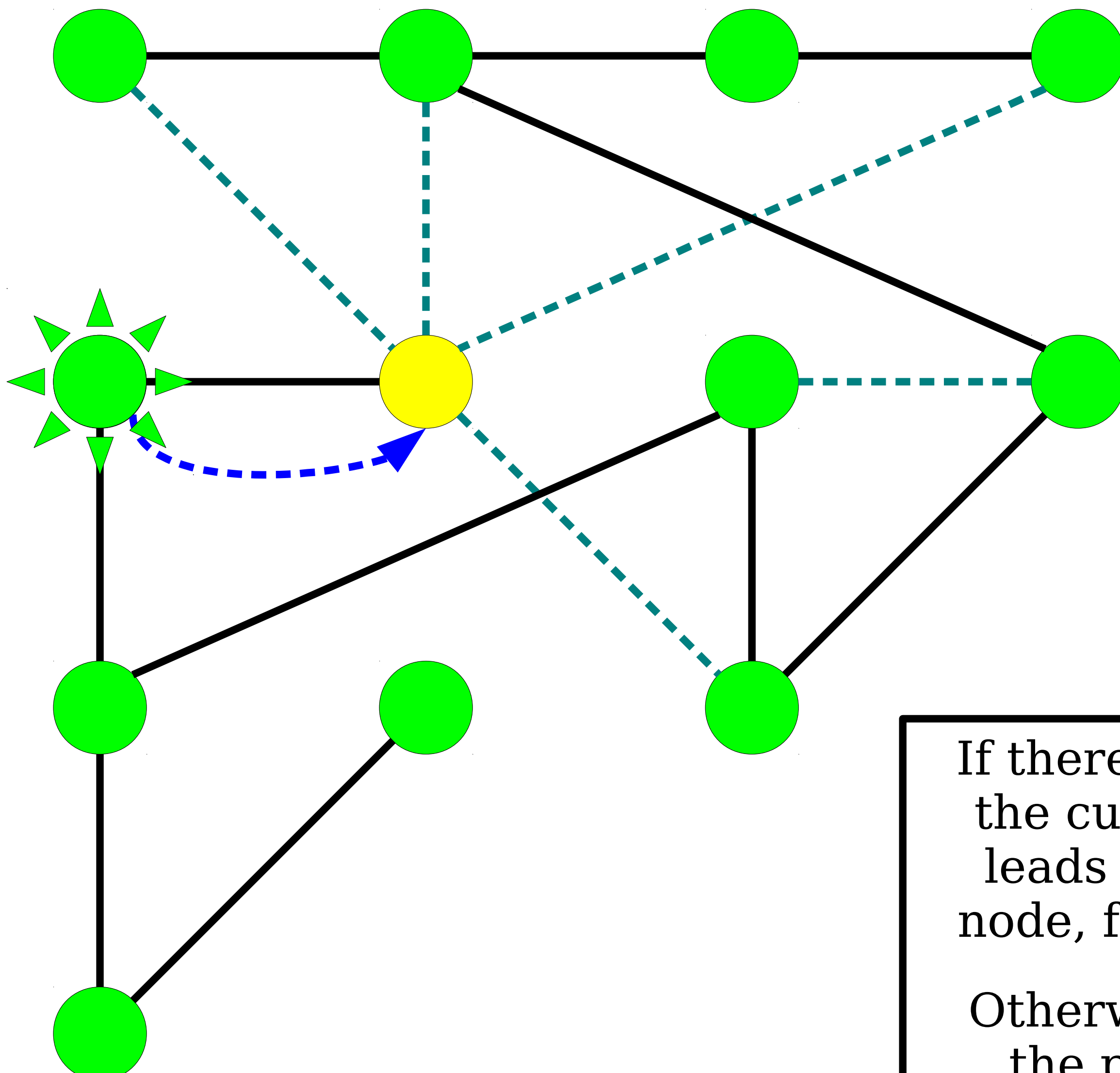
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

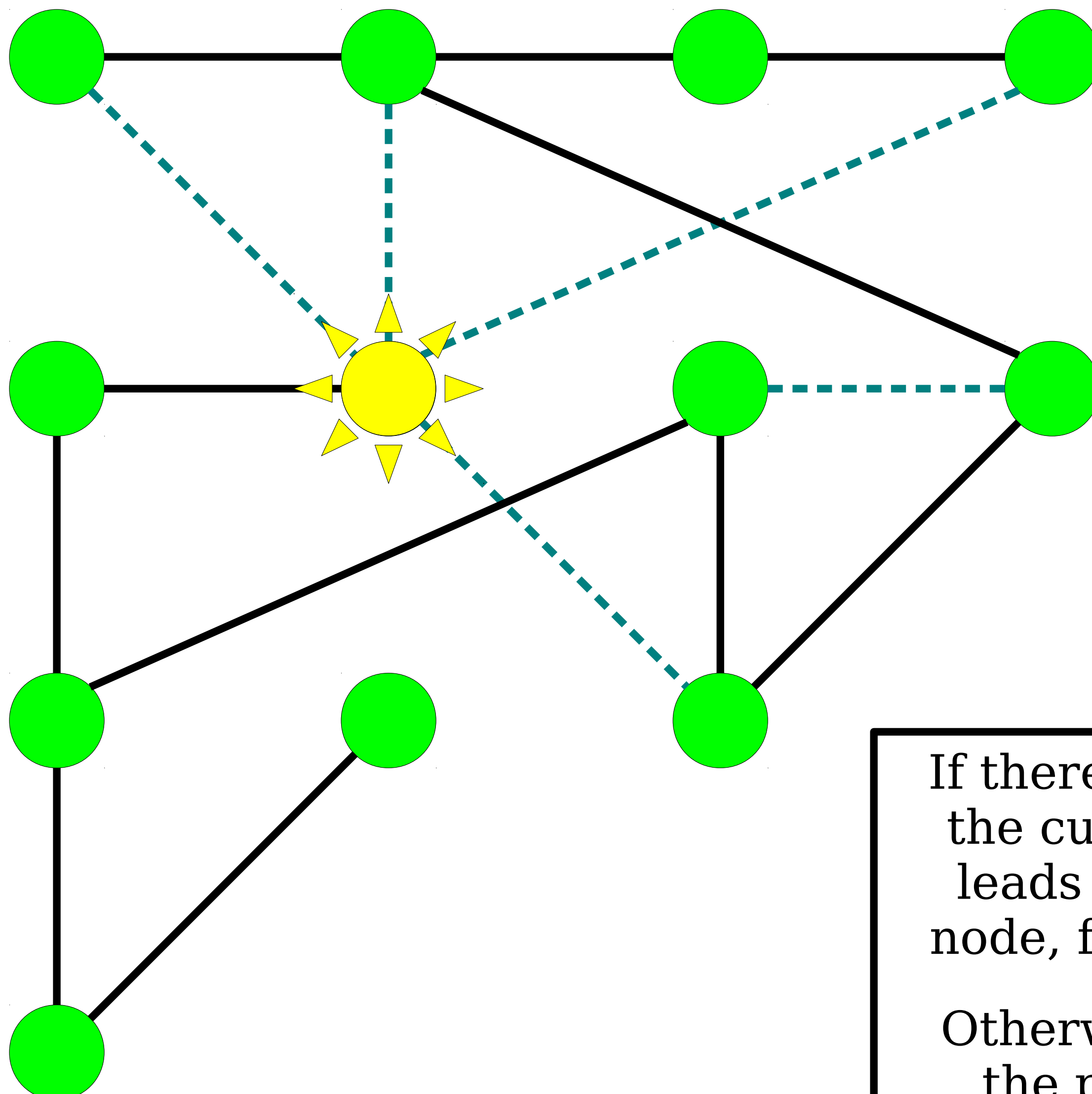
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

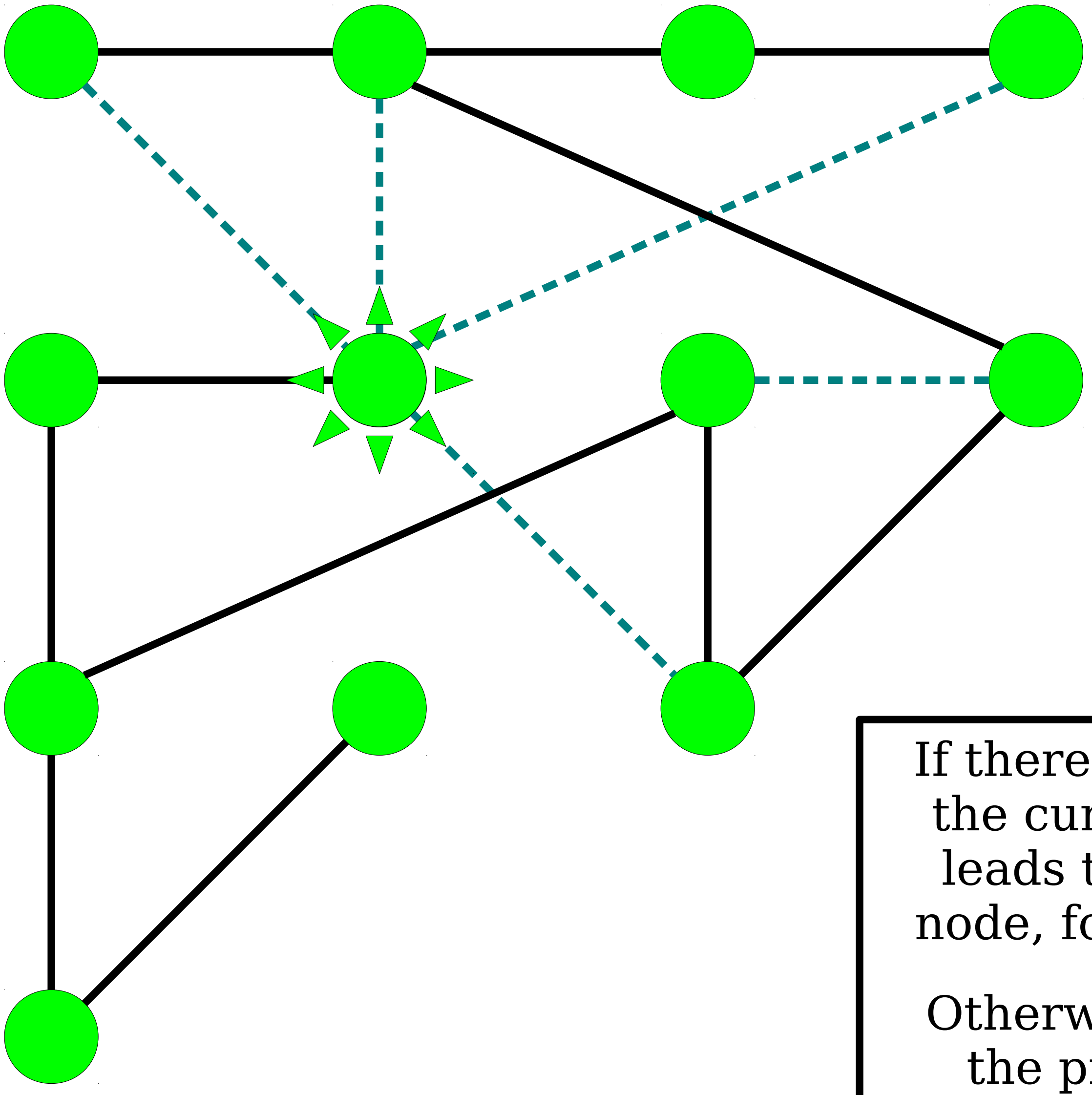
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

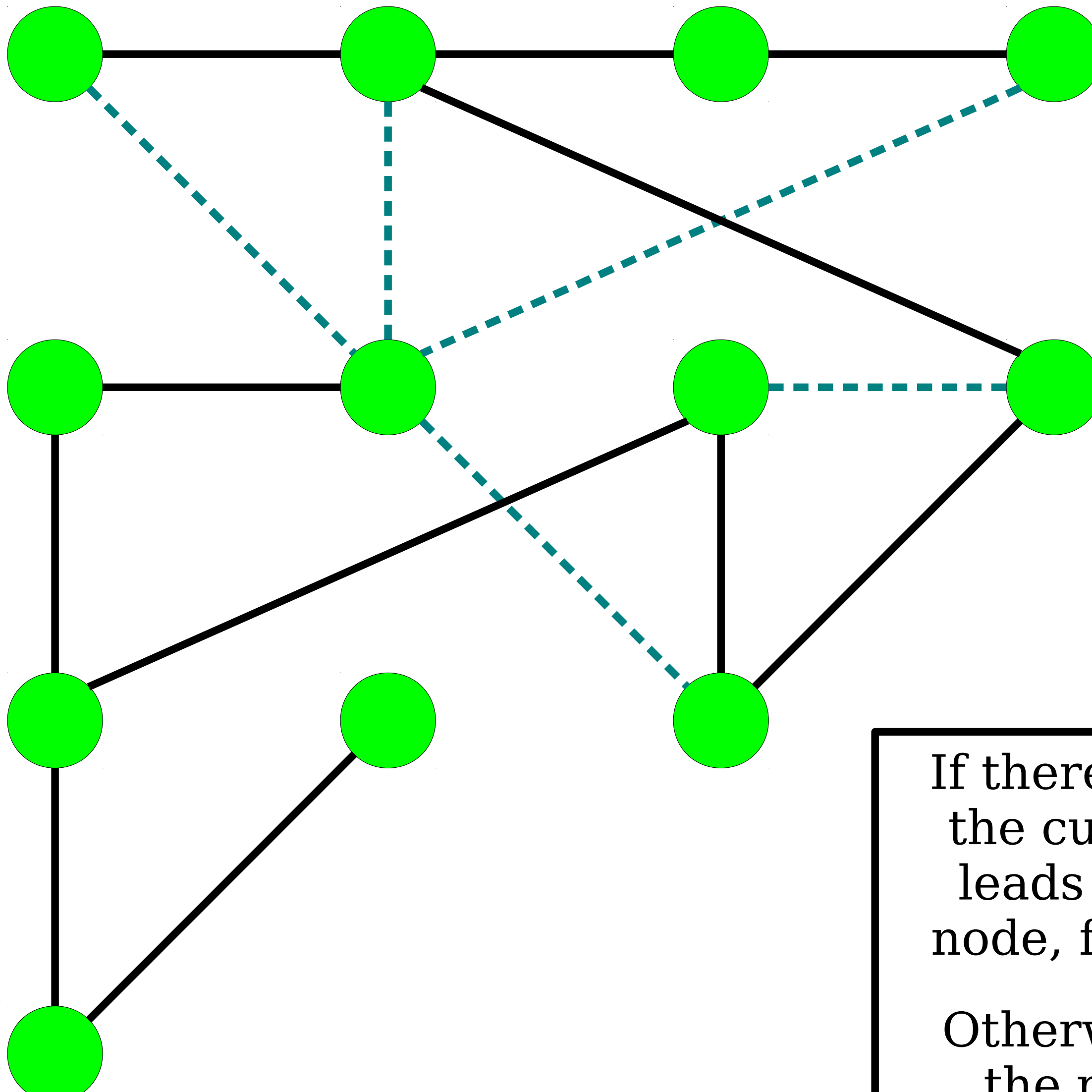
Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

Depth-First Search



If there's an edge from the current node that leads to an unvisited node, follow that edge.

Otherwise, back up to the previous node.

DFS: The Rundown

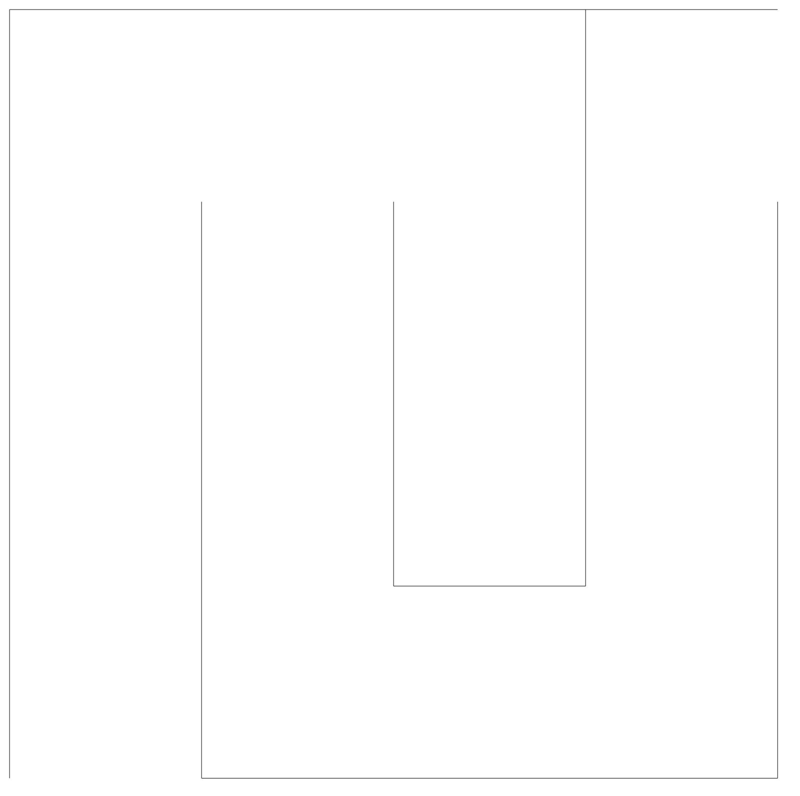
- Starting a DFS at a given node will find all nodes *reachable* from that node.
- If you have an adjacency list, doing a DFS in an n-node, m-edge graph takes time $O(m + n)$ and uses space $O(n)$.
  - Talk to me after class for details!
- There are some ***beautiful*** properties of the order in which DFS visits nodes (take CS161!), but until you know them the order of nodes found can look pretty random. Come talk to me after class for more info.

Depth-First Search

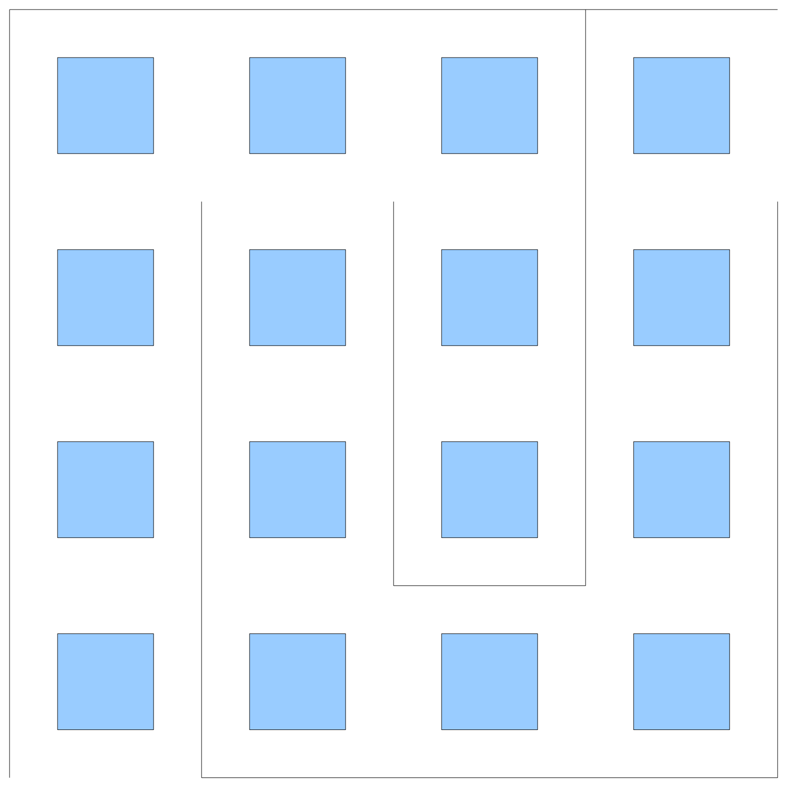
- To do a depth-first search (DFS) from a node *u*, do the following:
  - If *u* isn't ***gray*** (unvisited), stop.
  - Color *u* ***yellow*** (active).
  - For each neighbor *v* of *u:*
    - Recursively run DFS from *v*.
  - Color *u* ***green*** (done).
- The code for DFS is amazingly short!

A Whimsical Application

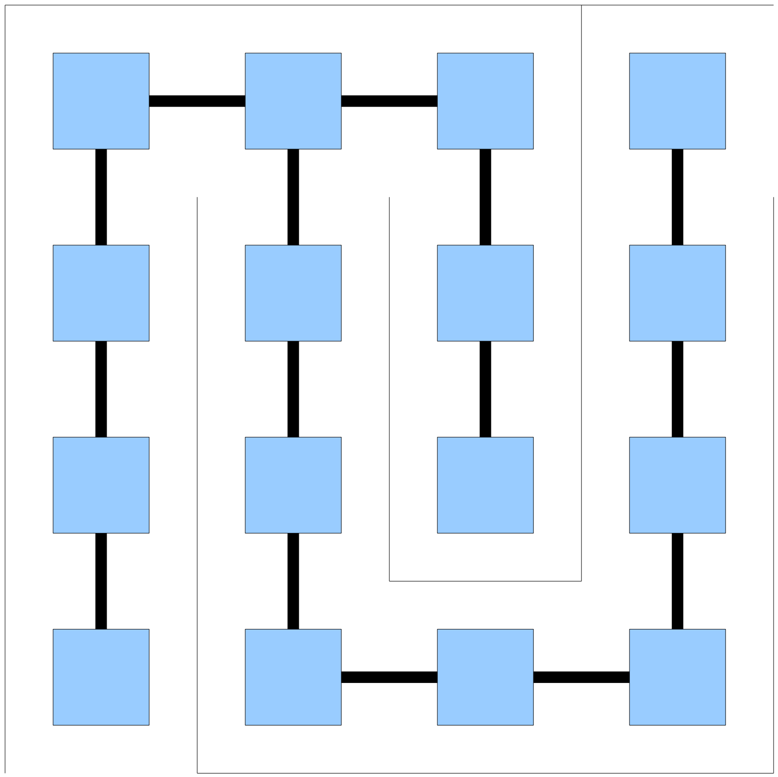
Mazes as Graphs



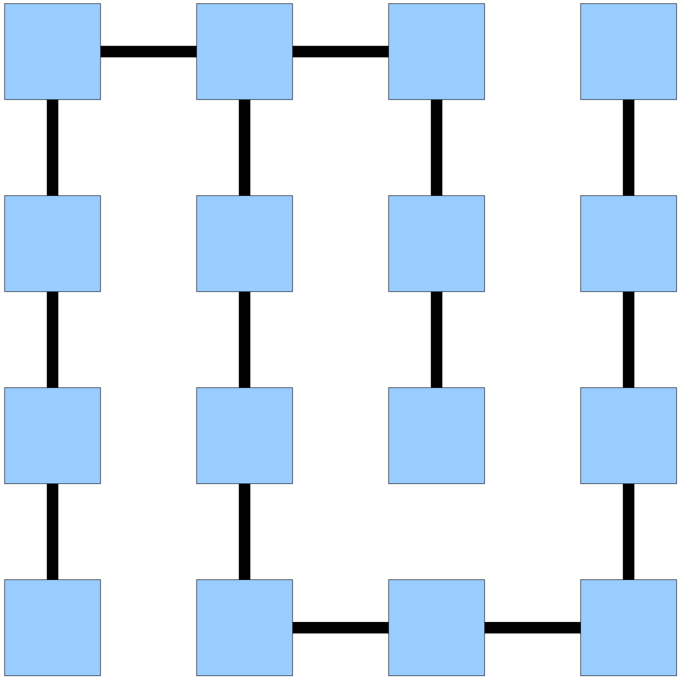
Mazes as Graphs



Mazes as Graphs

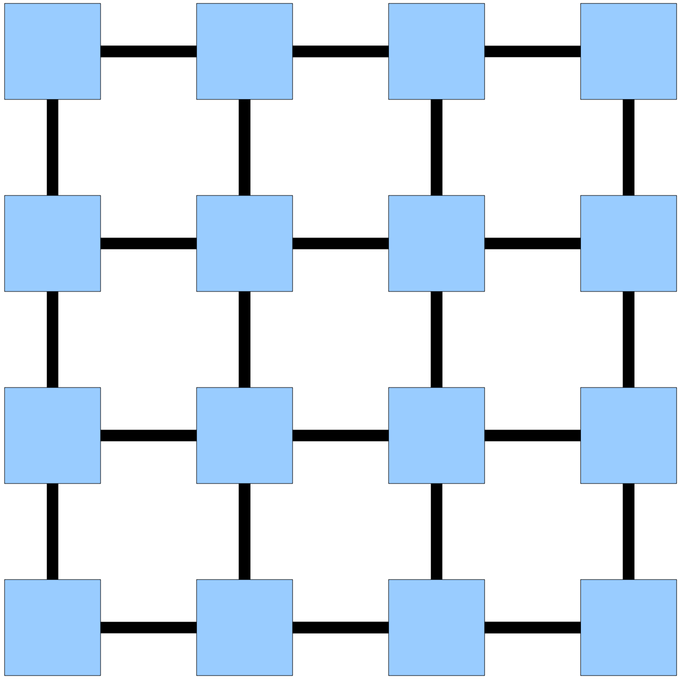


Mazes as Graphs



Creating a Maze with DFS

- Create a ***grid graph*** of the appropriate size.

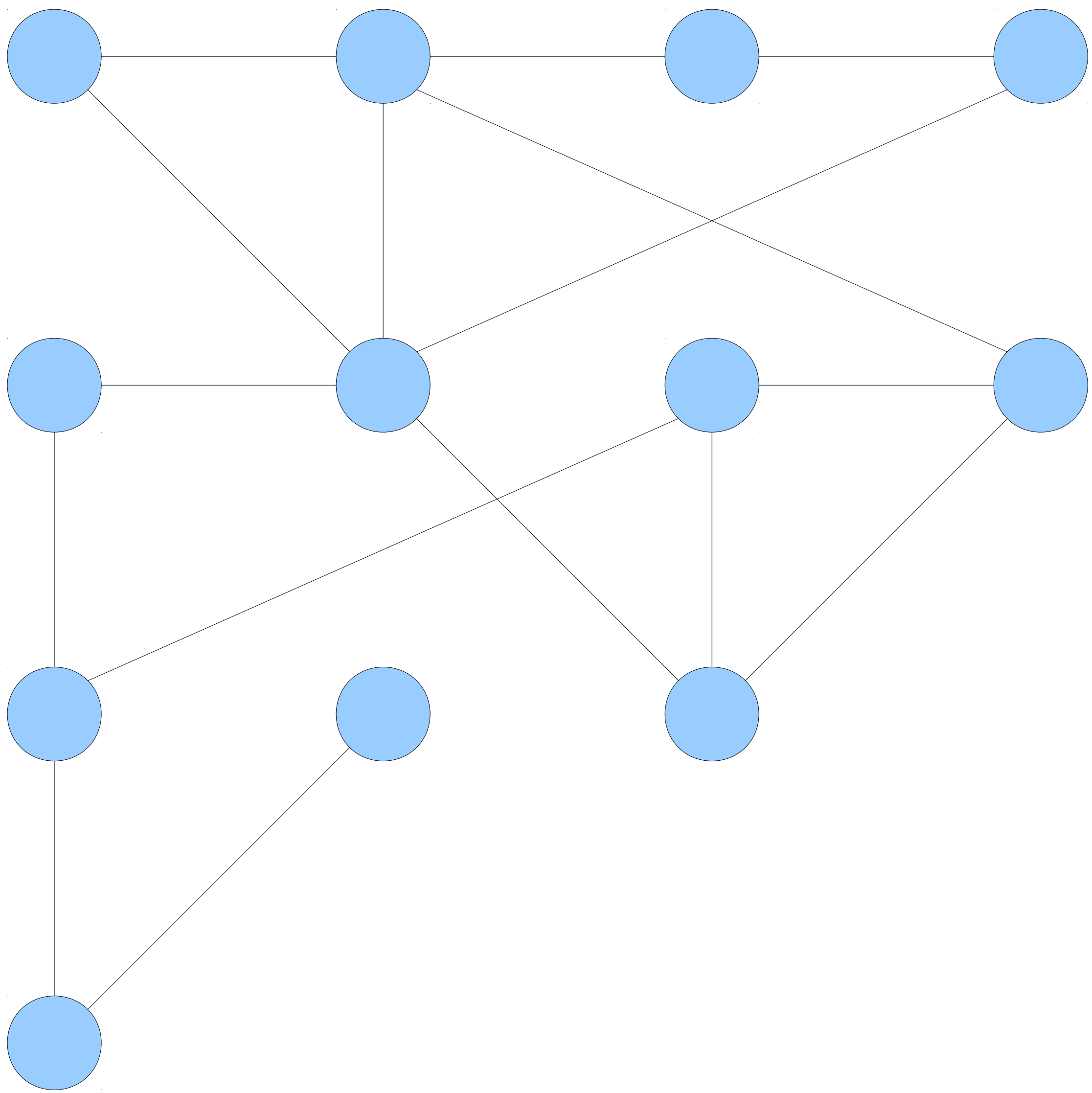


- Starting at any node, run a depth-first search, choosing neighbor orderings at random.
- The resulting DFS tree is a maze with one solution.

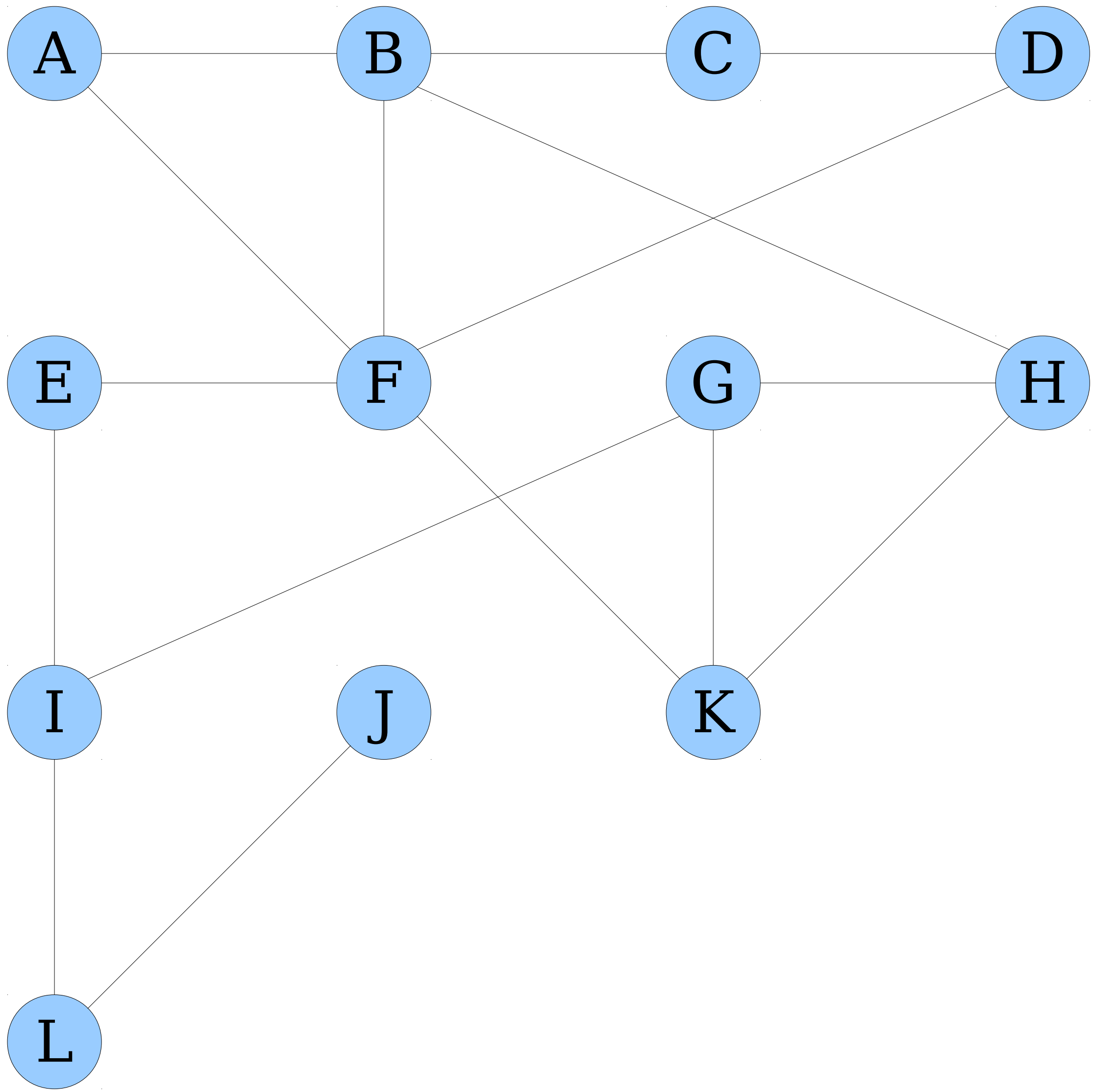
Breadth-First Search

How do you find the fastest route from one point to another?

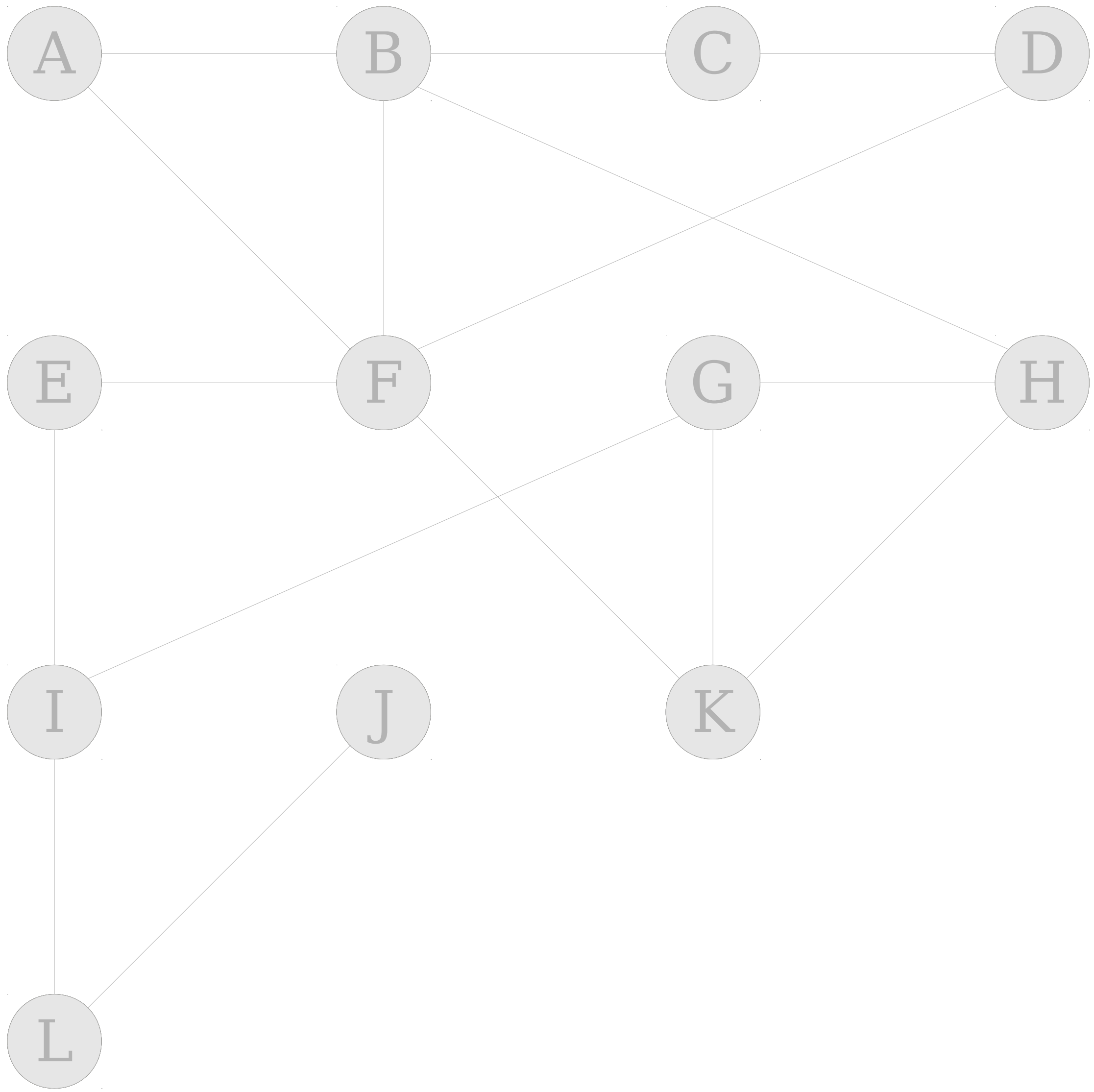
Breadth-First Search



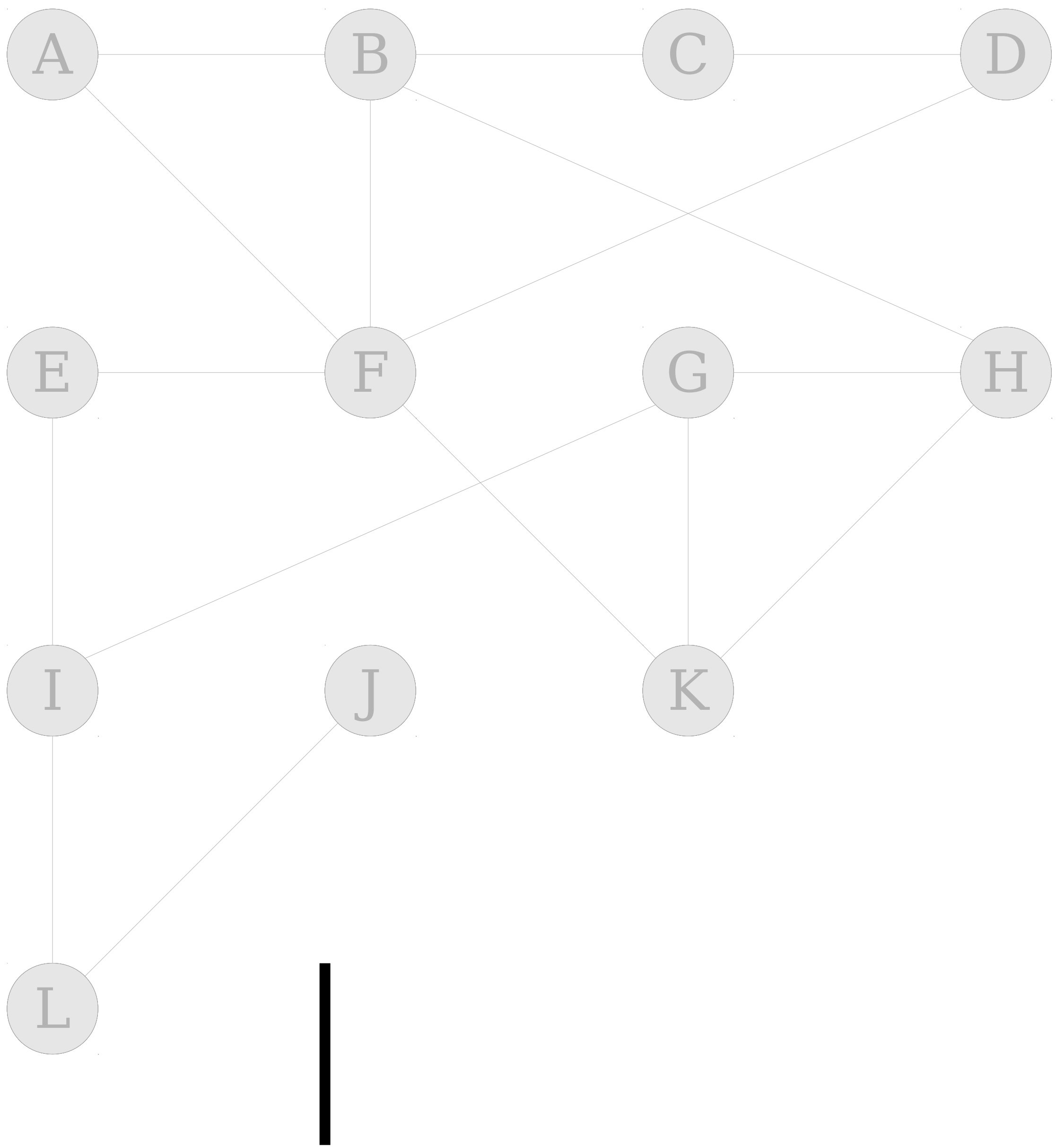
Breadth-First Search



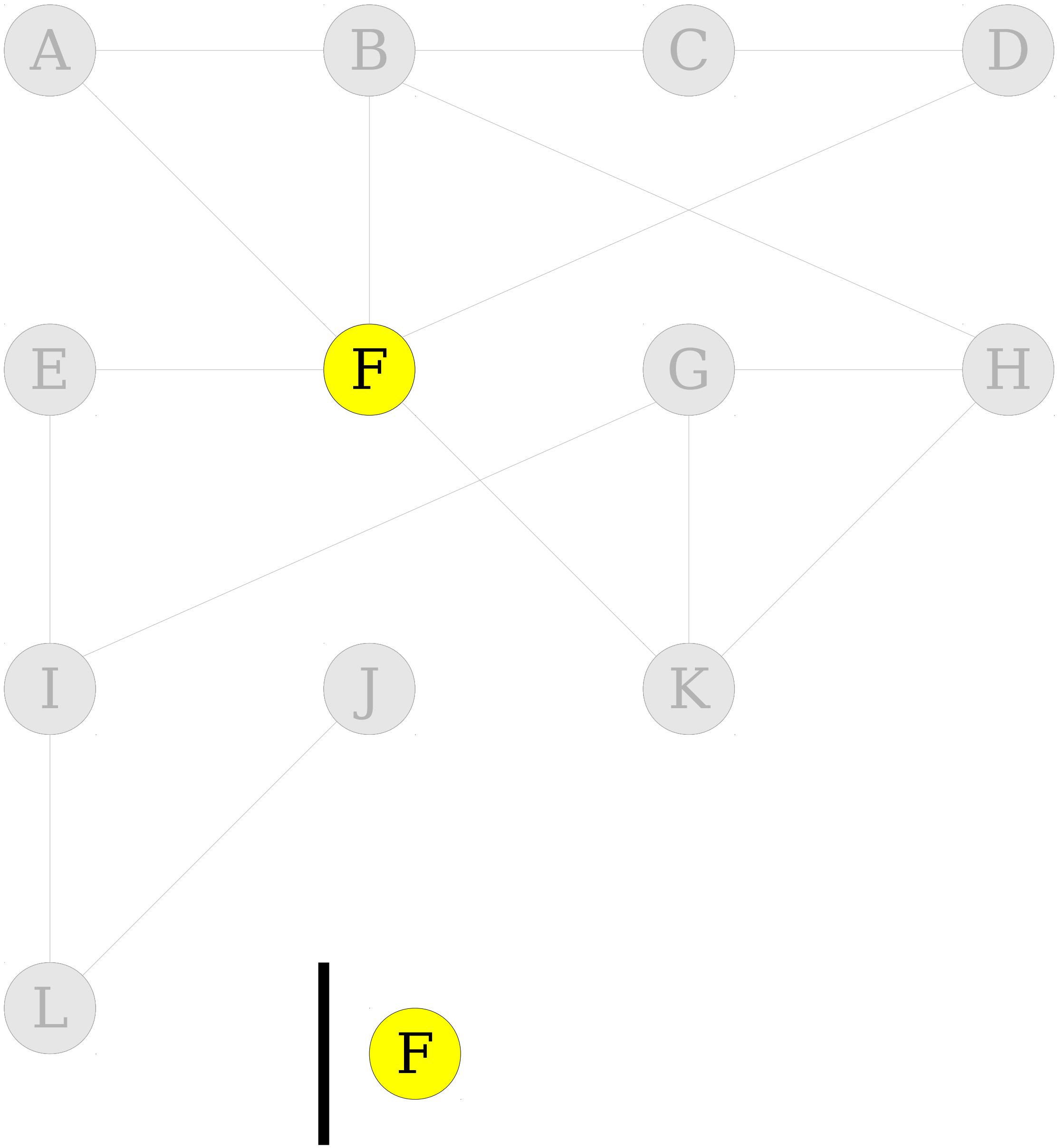
Breadth-First Search



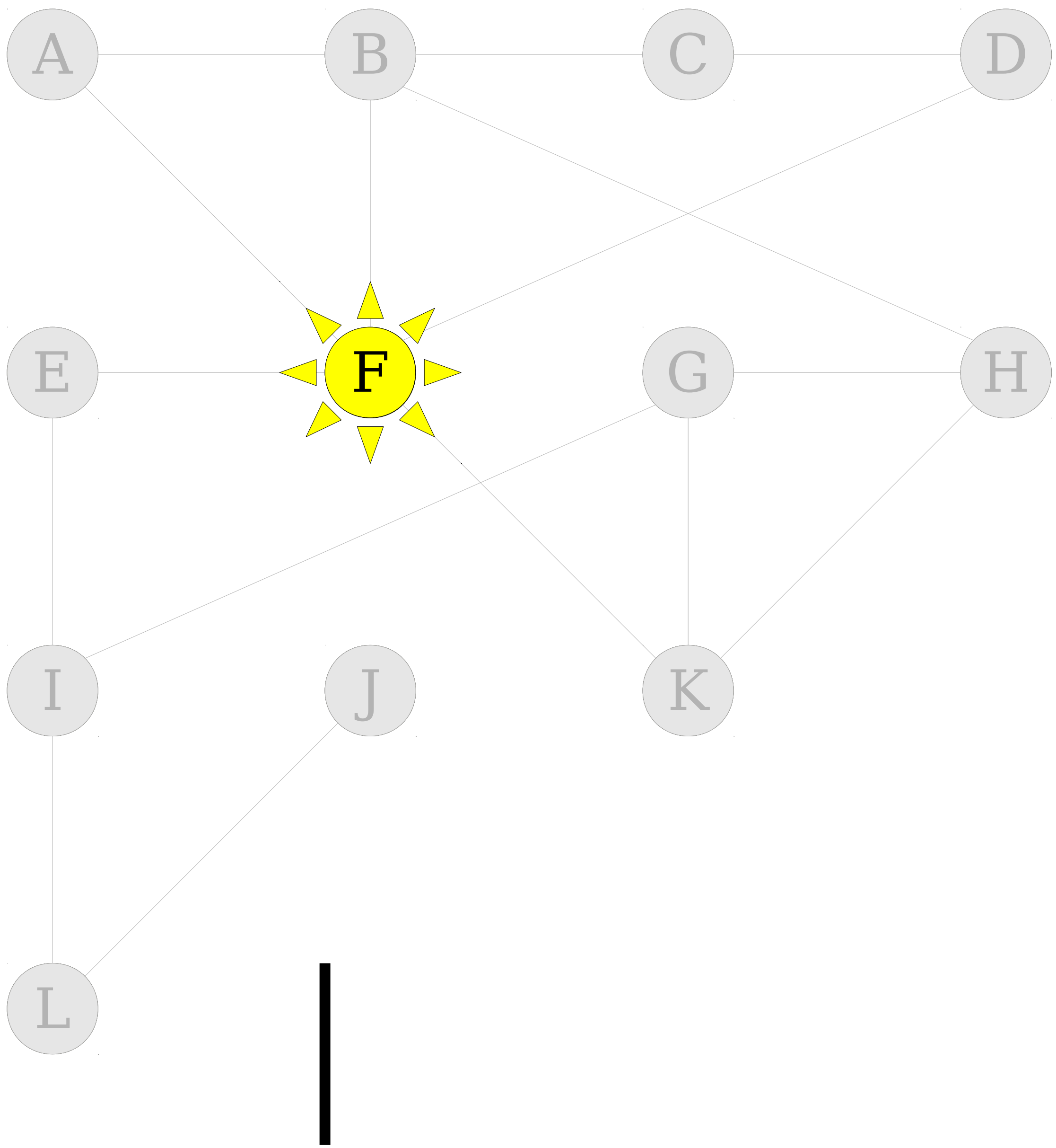
Breadth-First Search



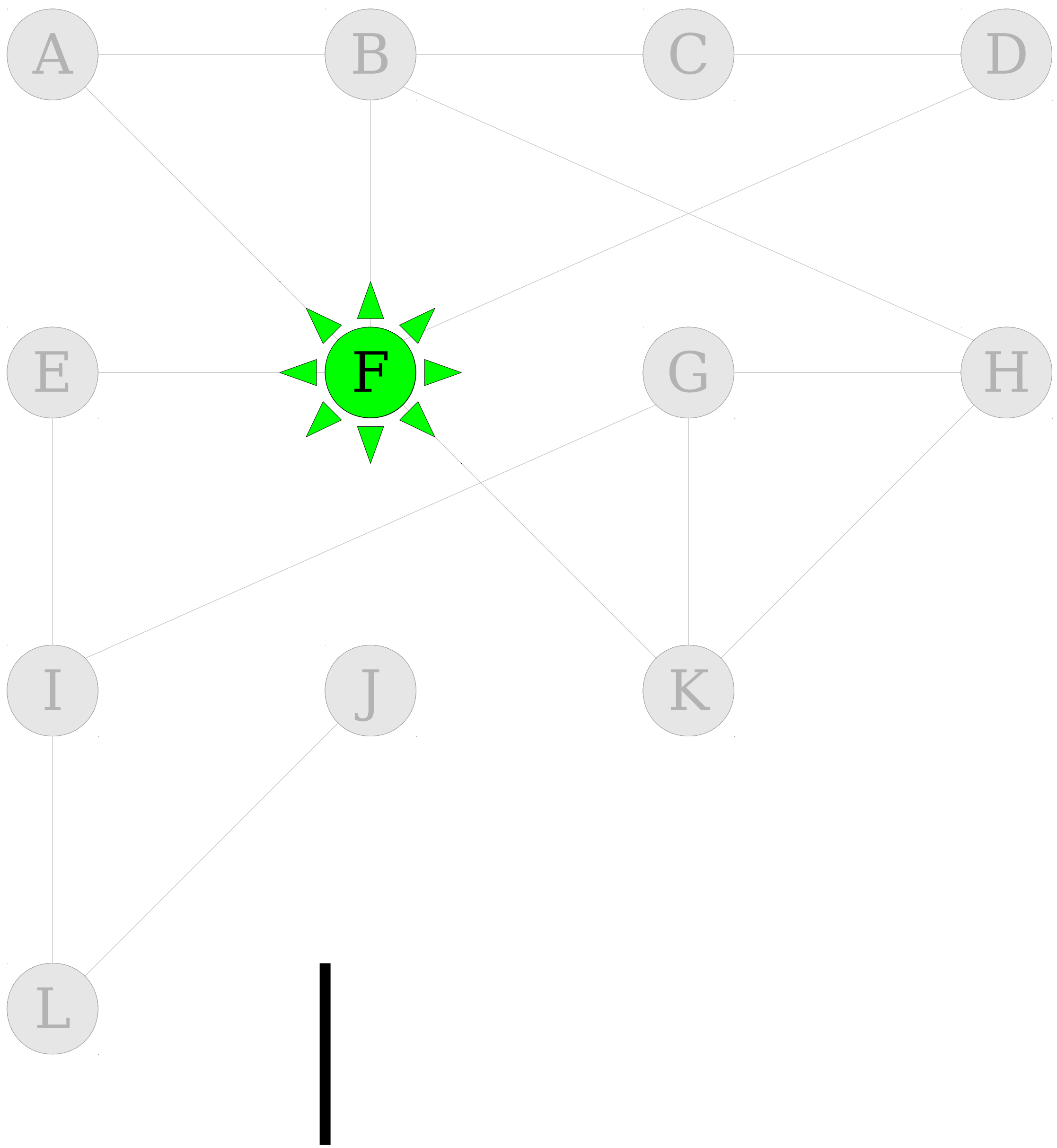
Breadth-First Search



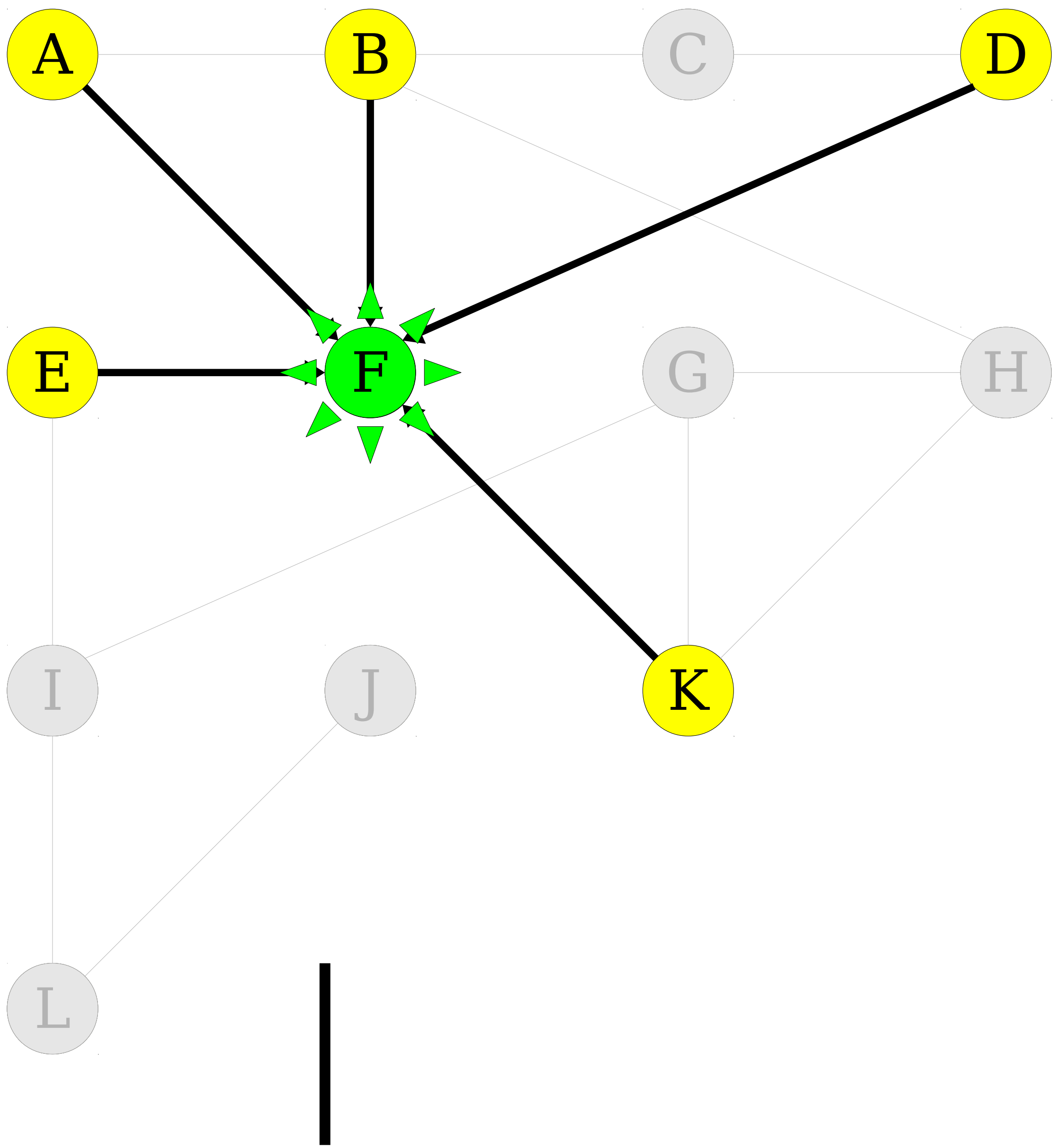
Breadth-First Search



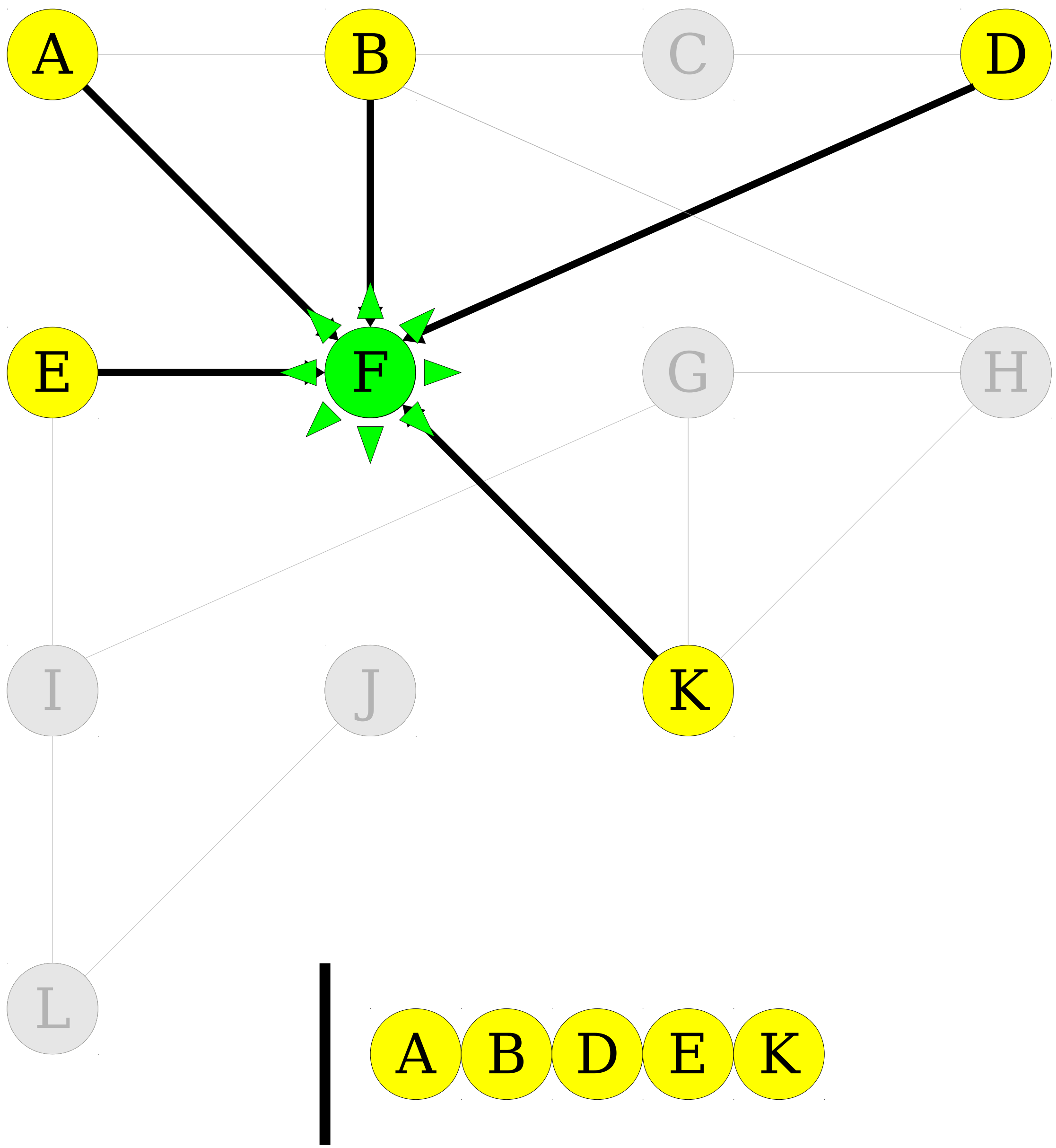
Breadth-First Search



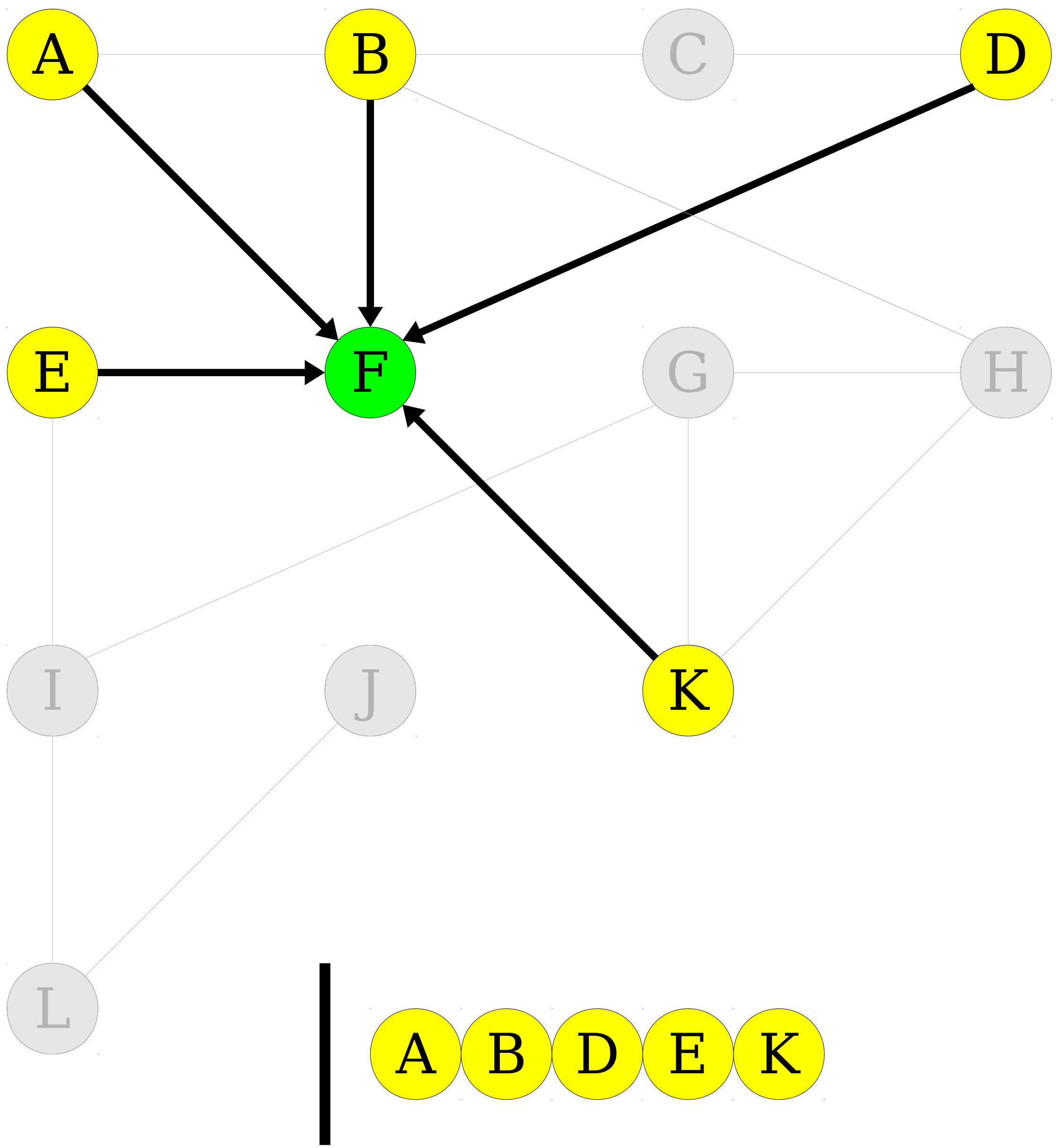
Breadth-First Search



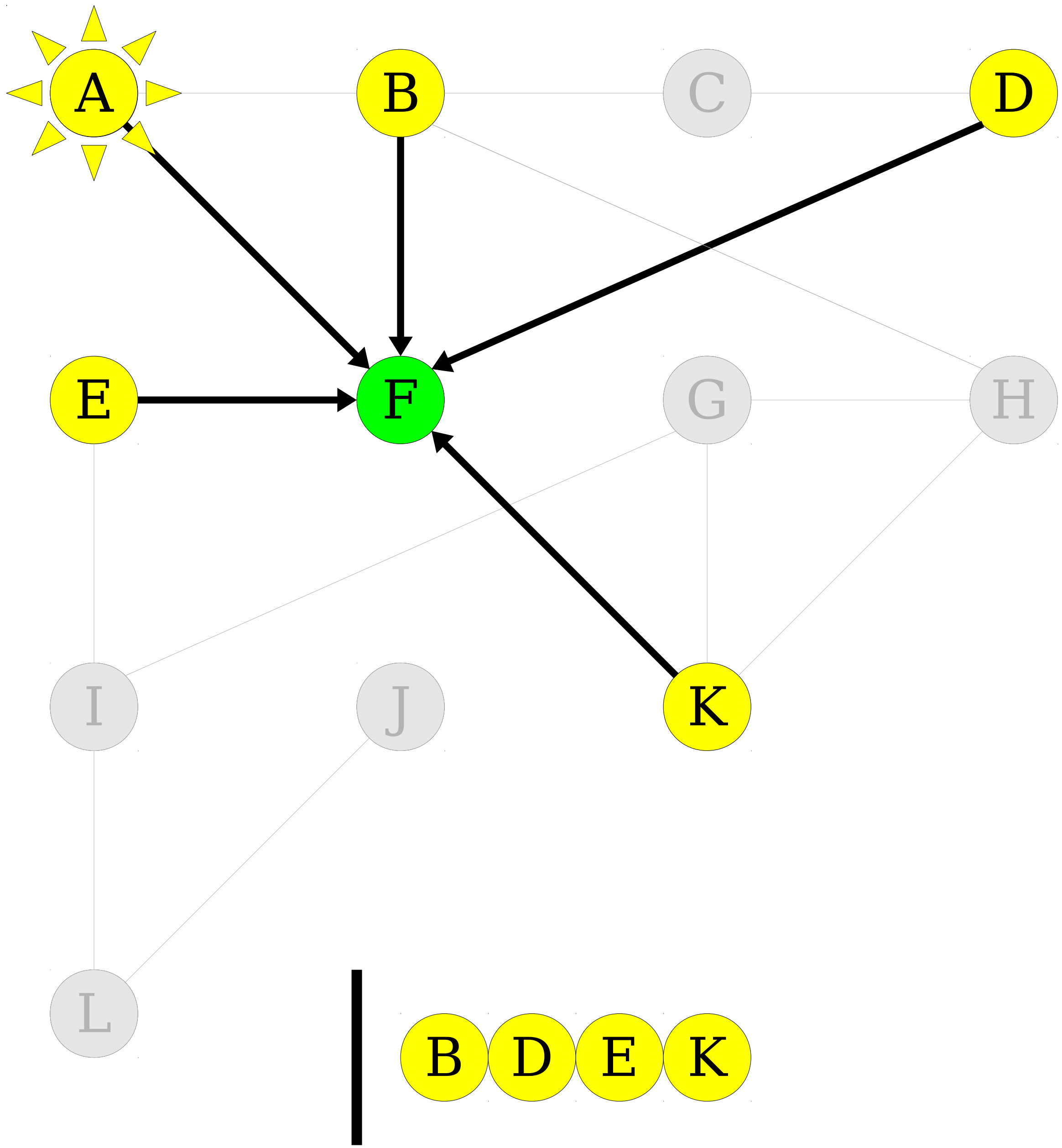
Breadth-First Search



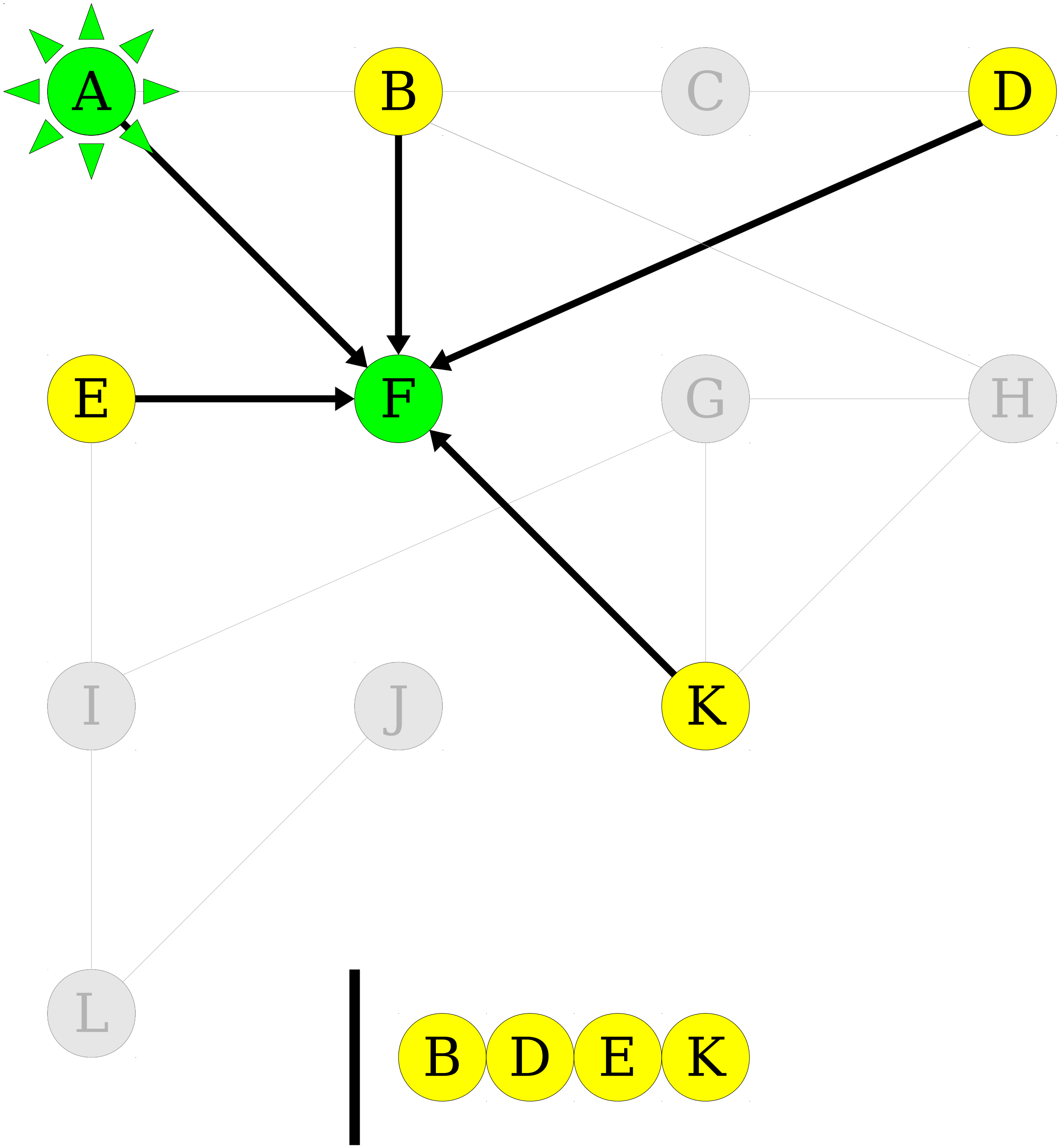
Breadth-First Search



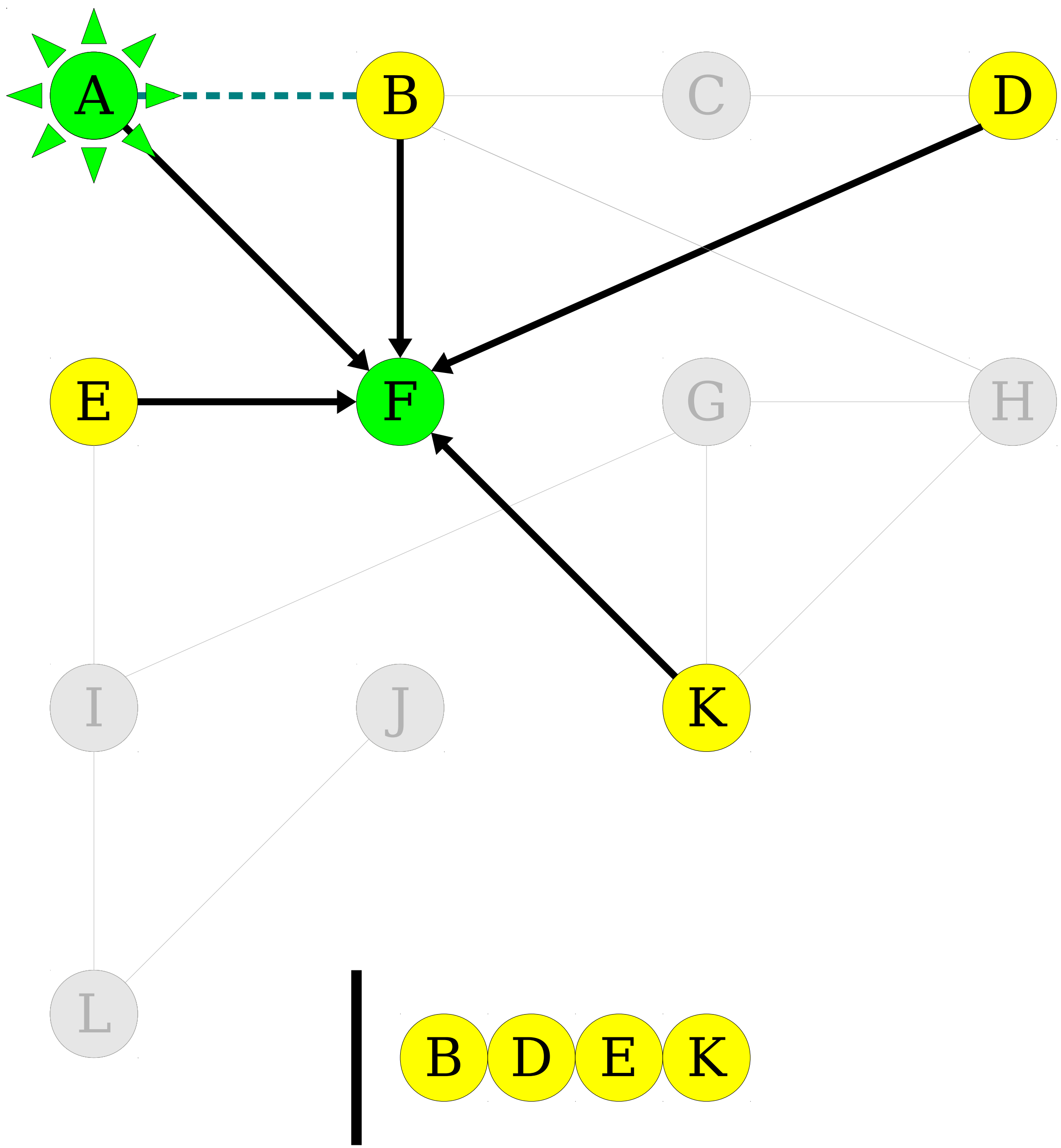
Breadth-First Search



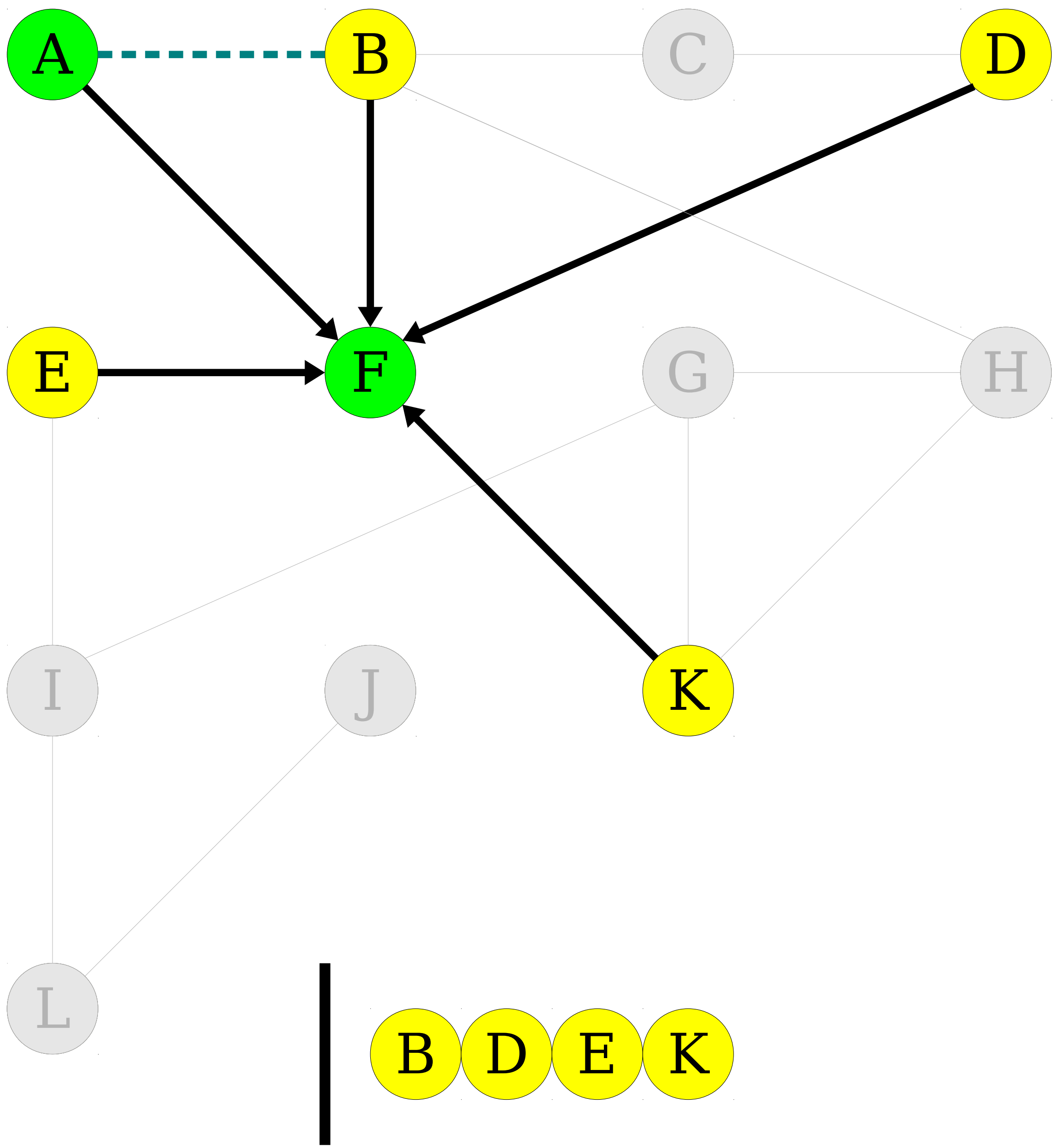
Breadth-First Search



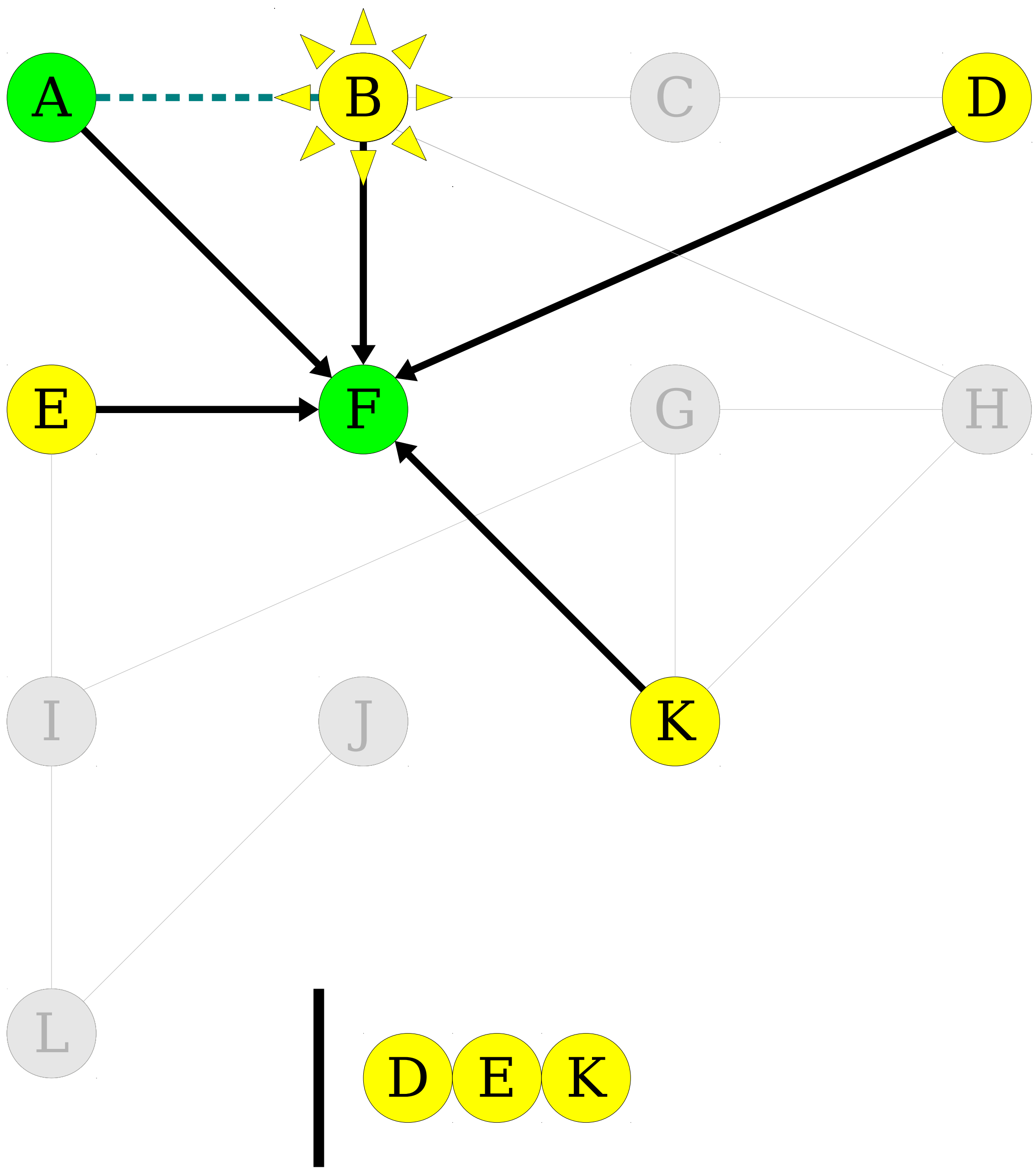
Breadth-First Search



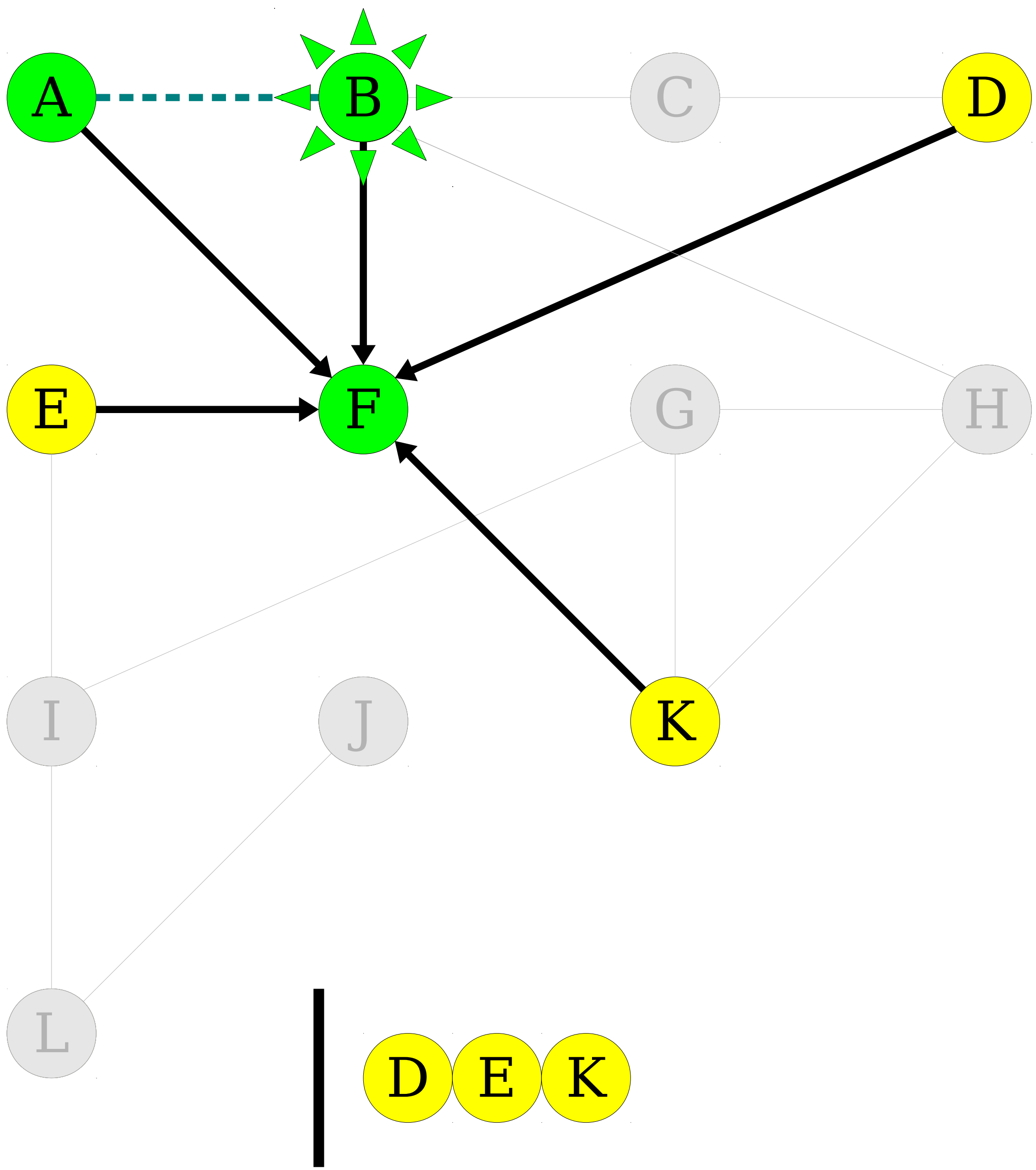
Breadth-First Search



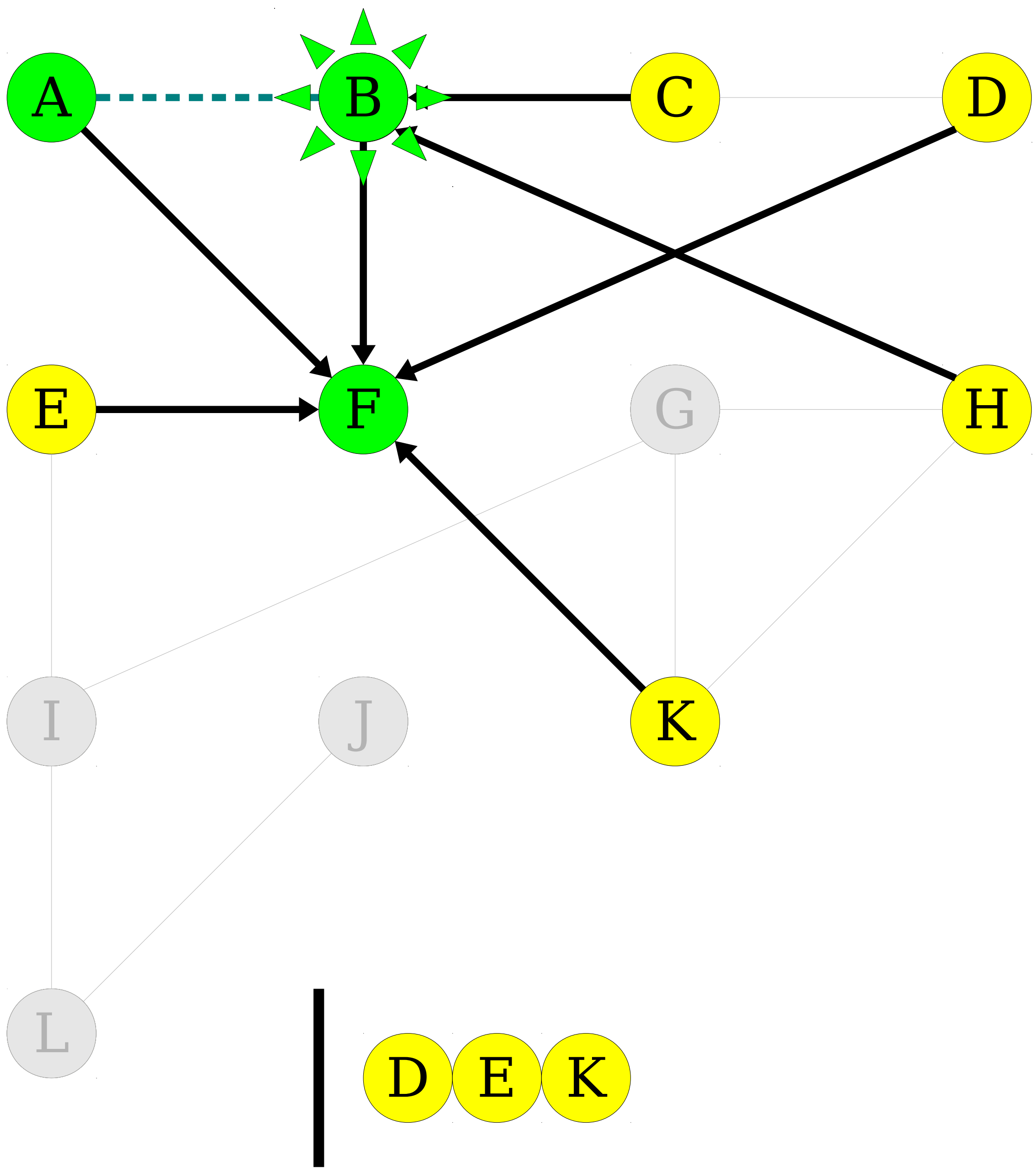
Breadth-First Search



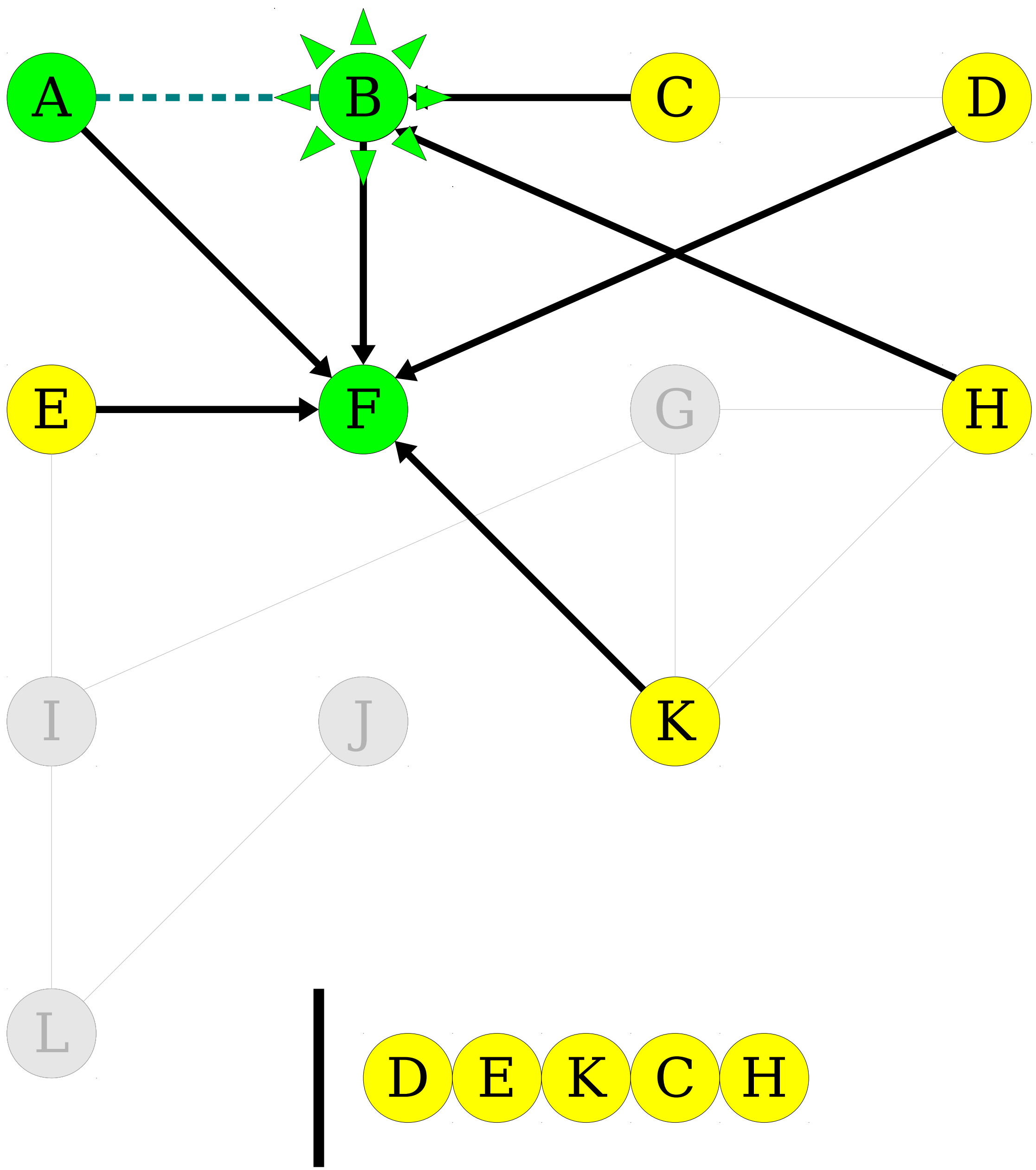
Breadth-First Search



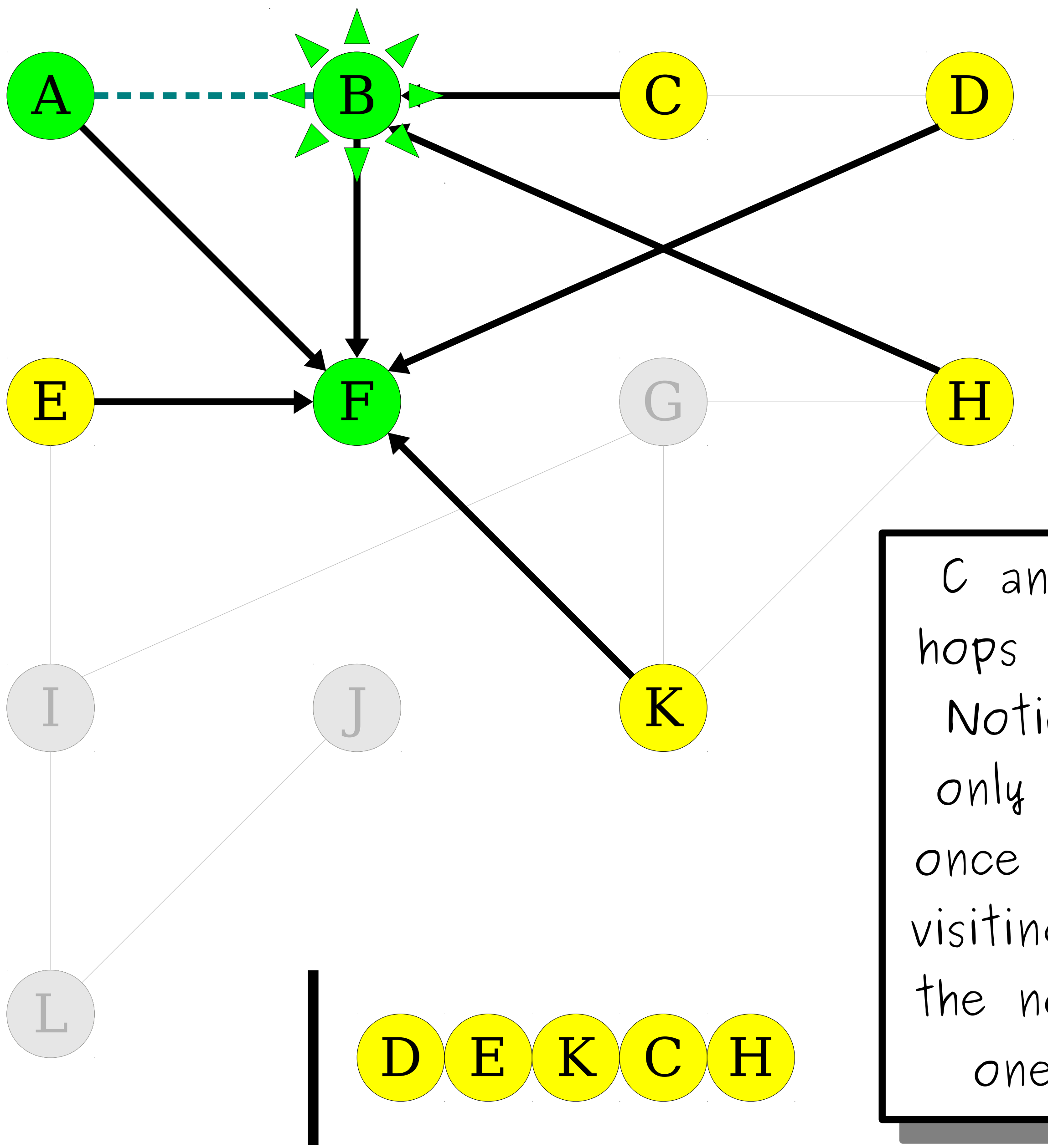
Breadth-First Search



Breadth-First Search

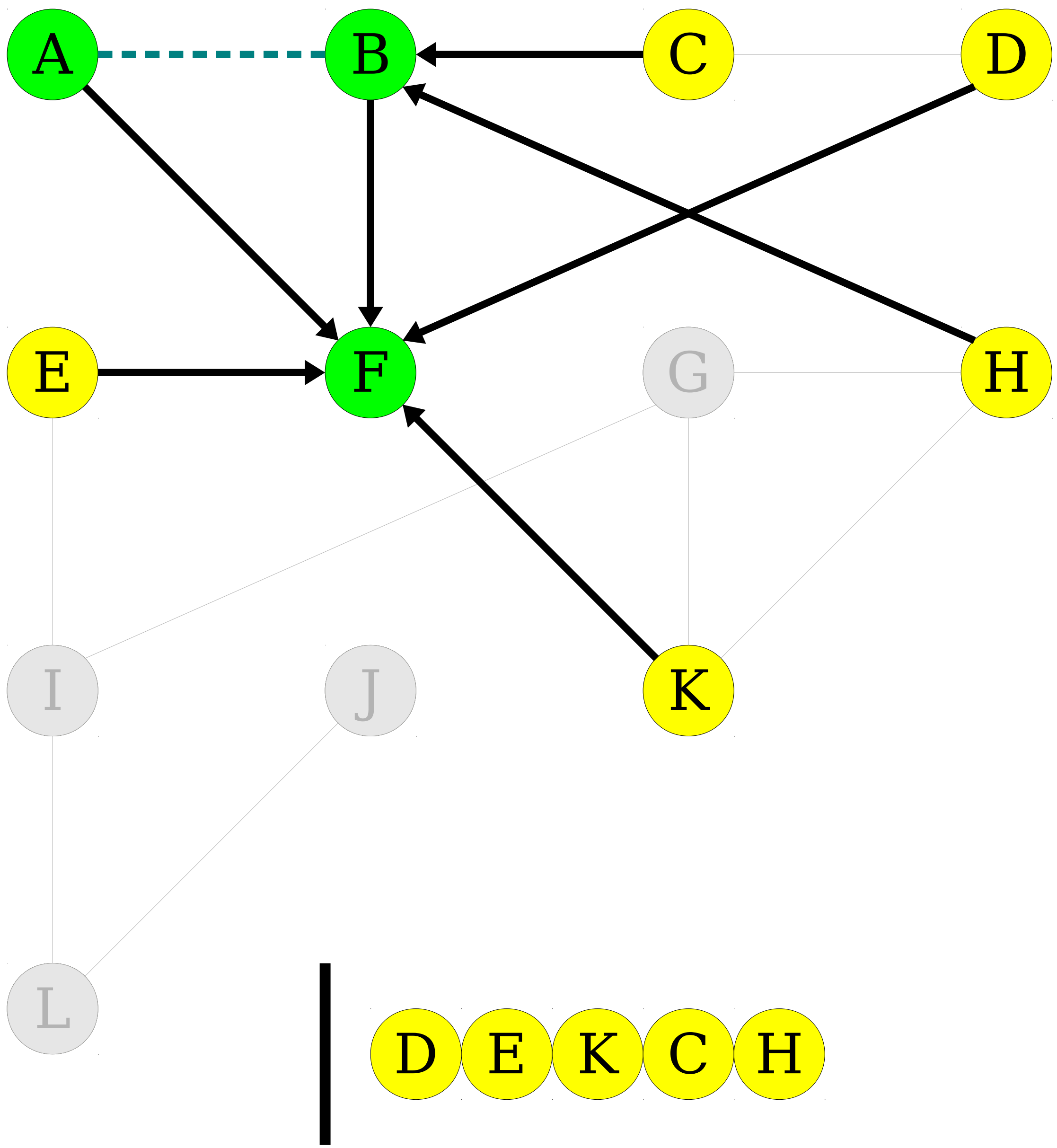


Breadth-First Search

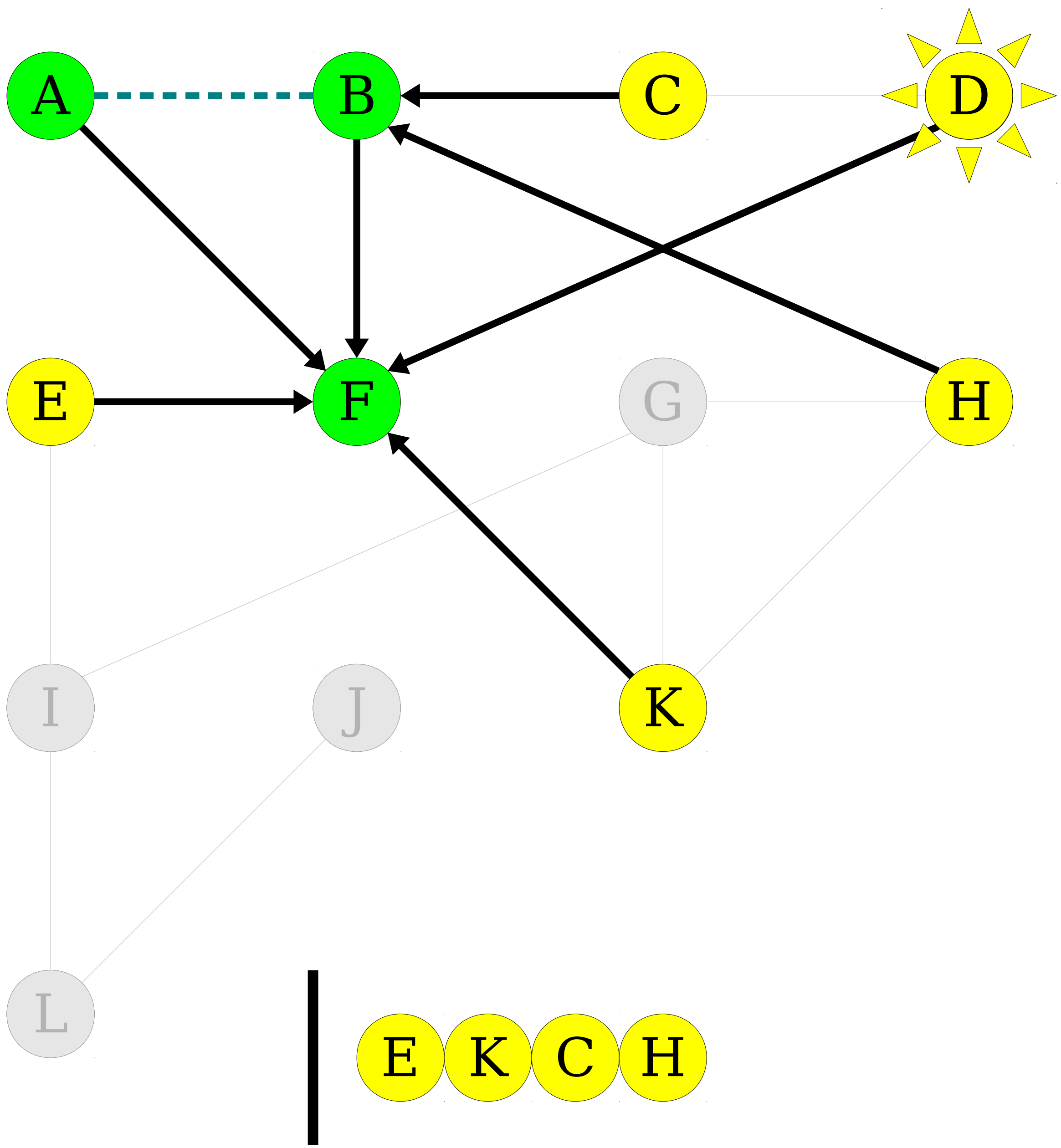


C and H are two hops away from F. Notice that we'll only get to them once we've finished visiting the rest of the nodes that are one hop away!

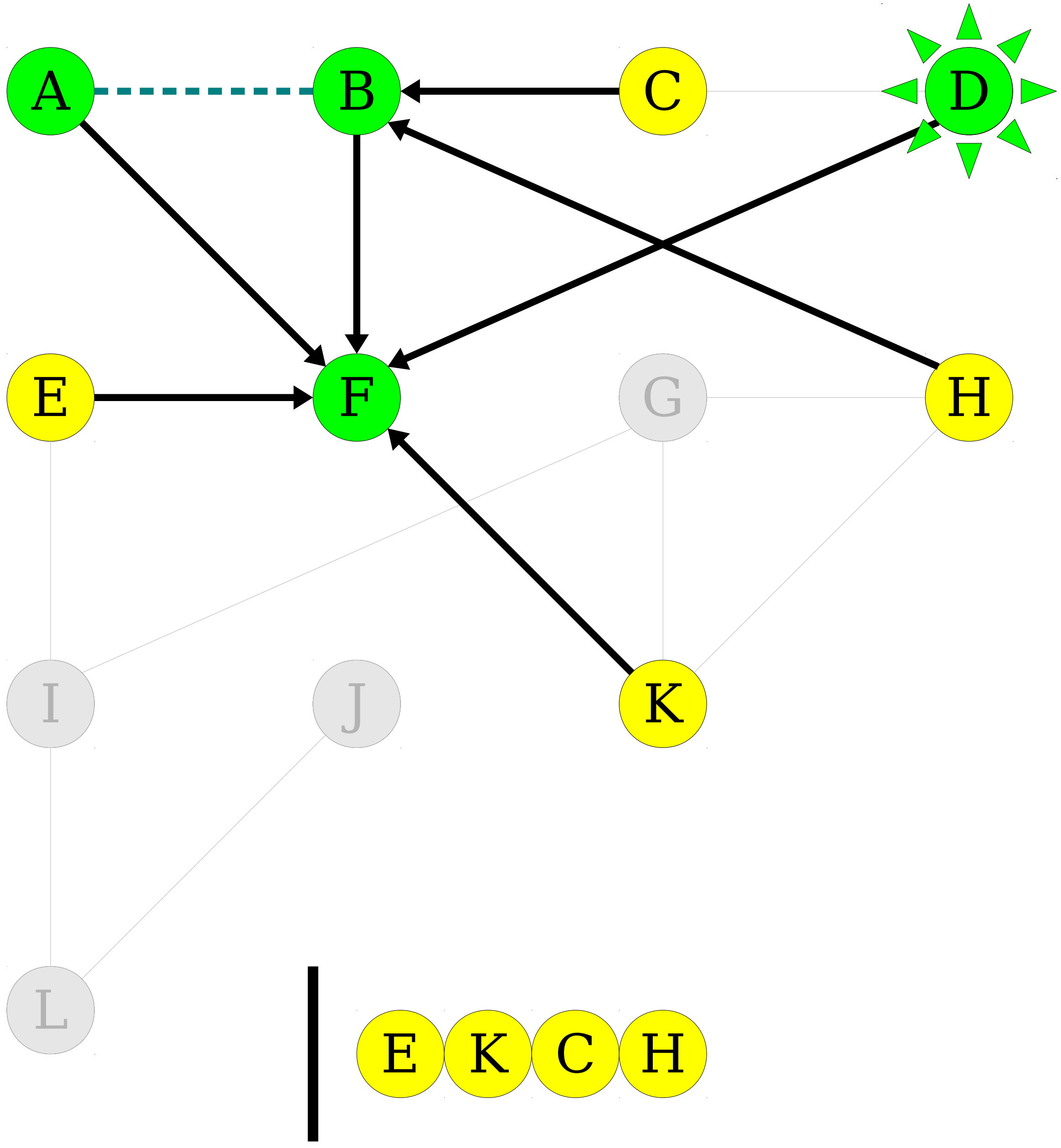
Breadth-First Search



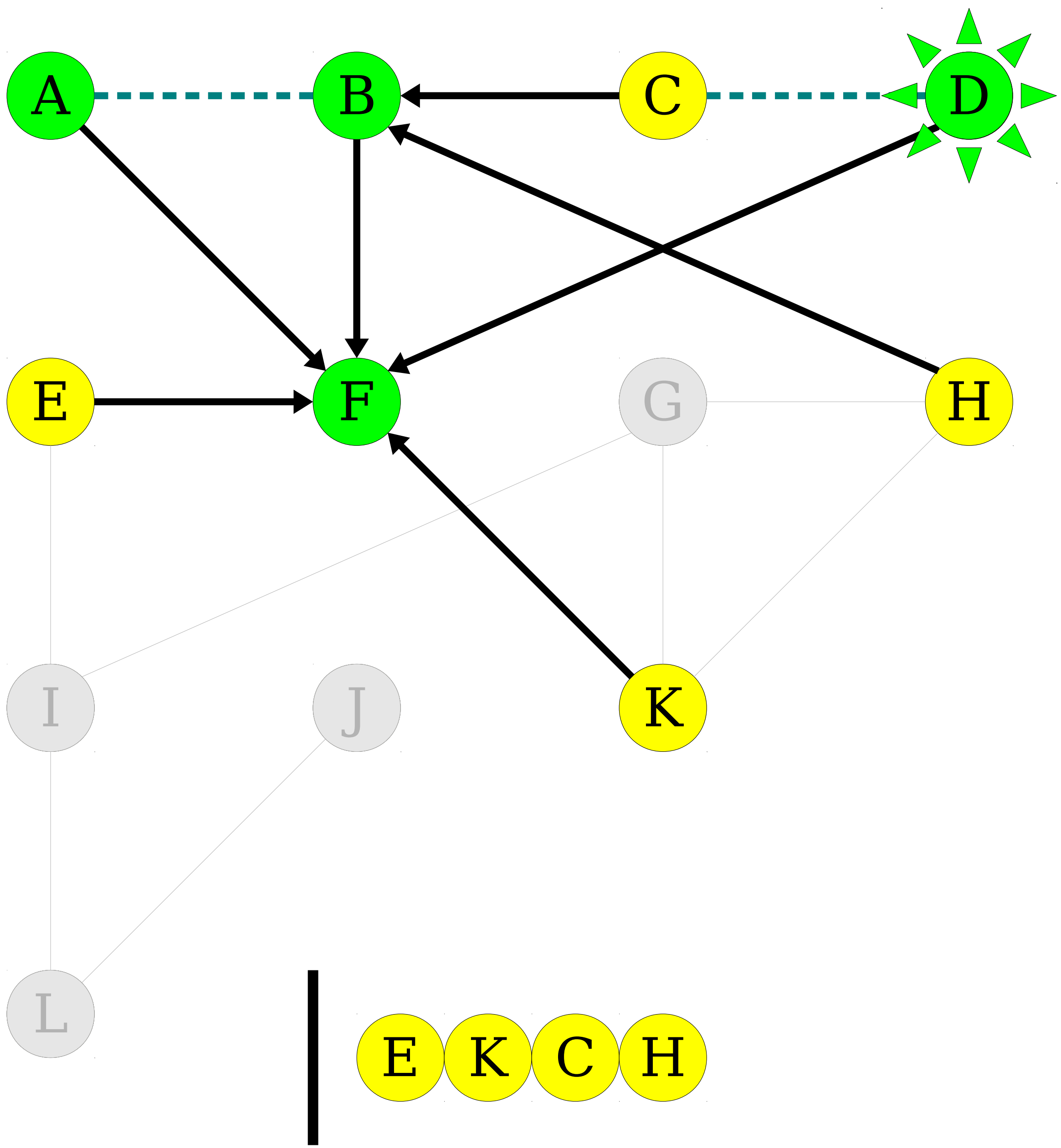
Breadth-First Search



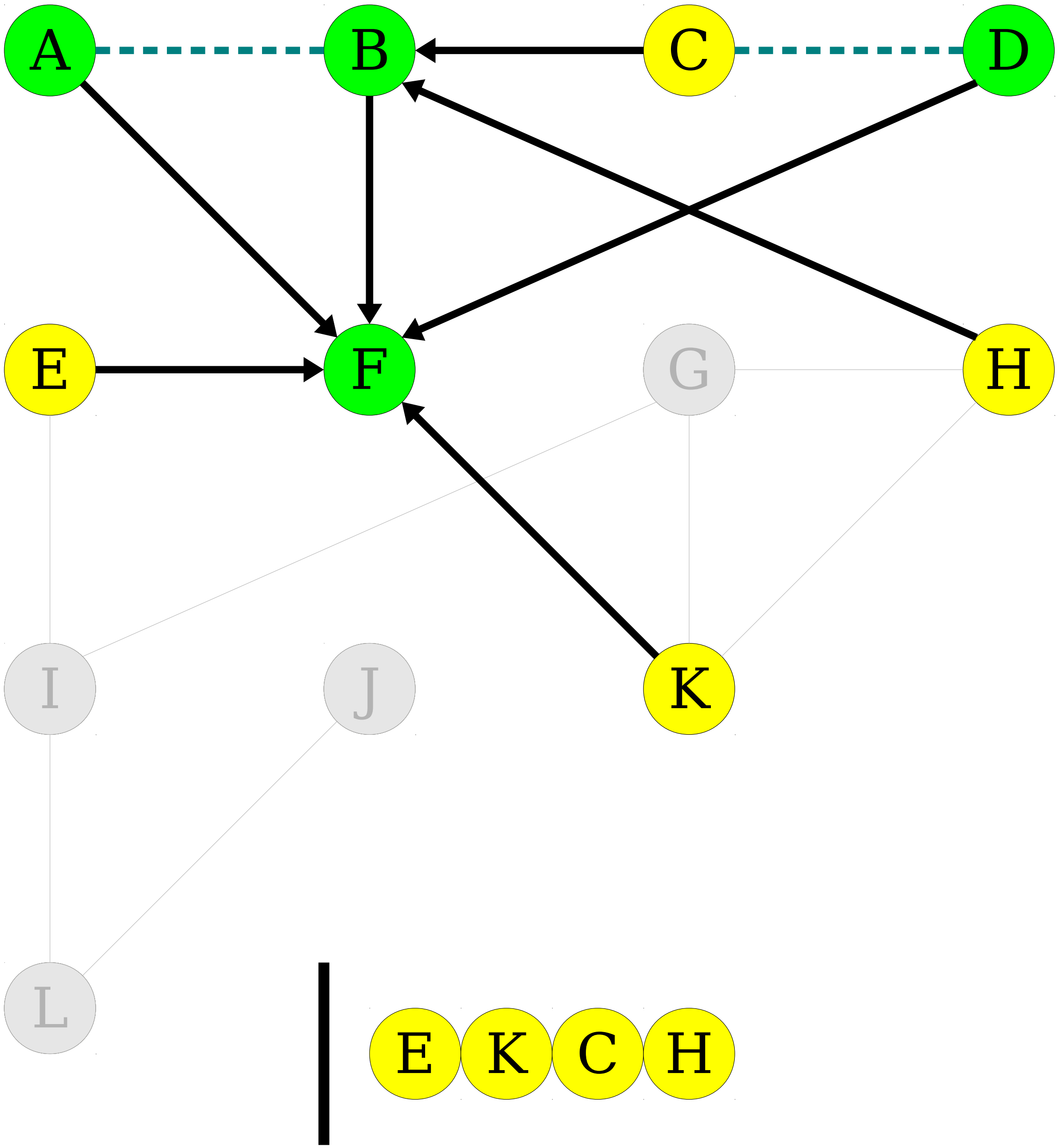
Breadth-First Search



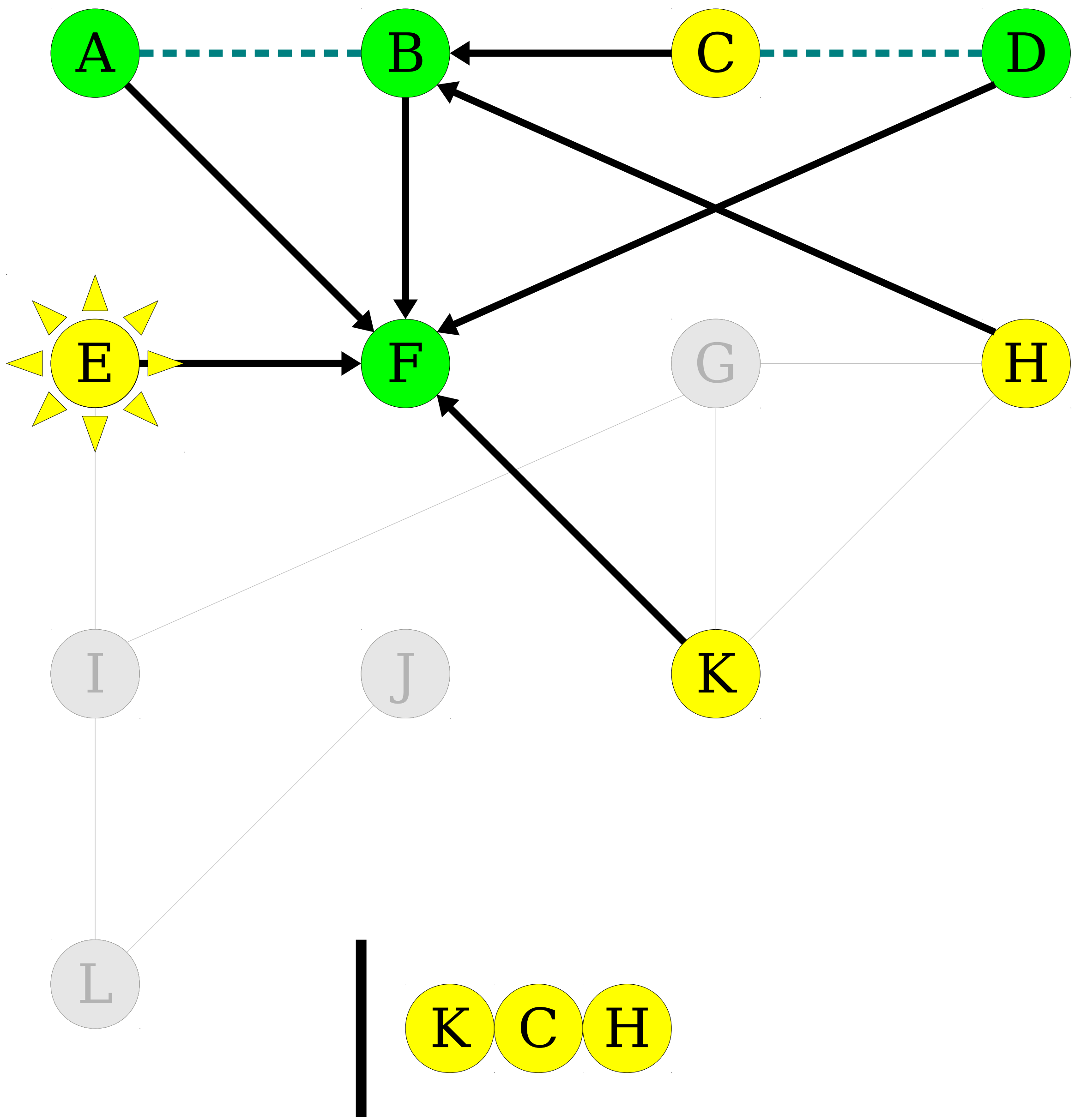
Breadth-First Search



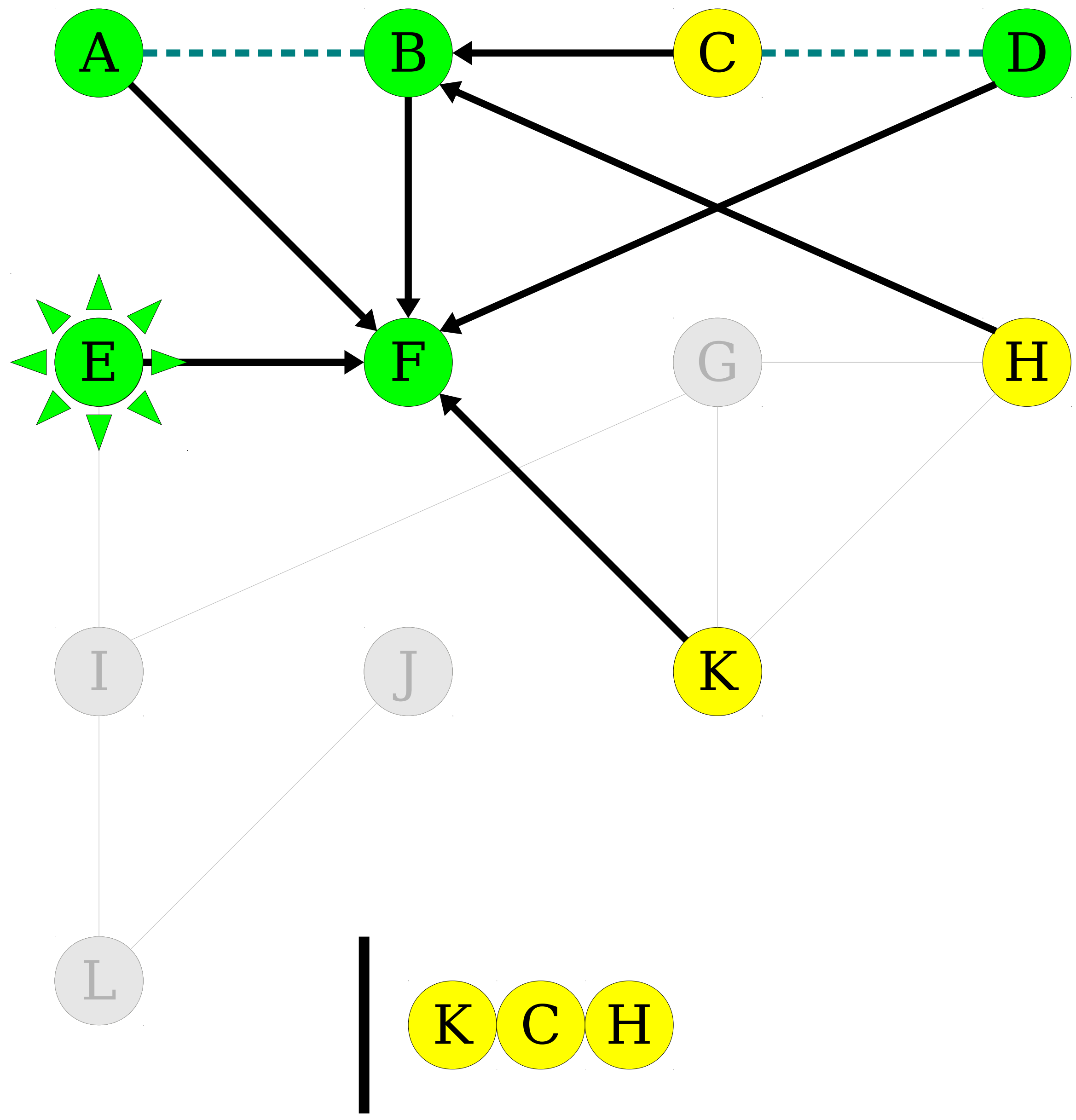
Breadth-First Search



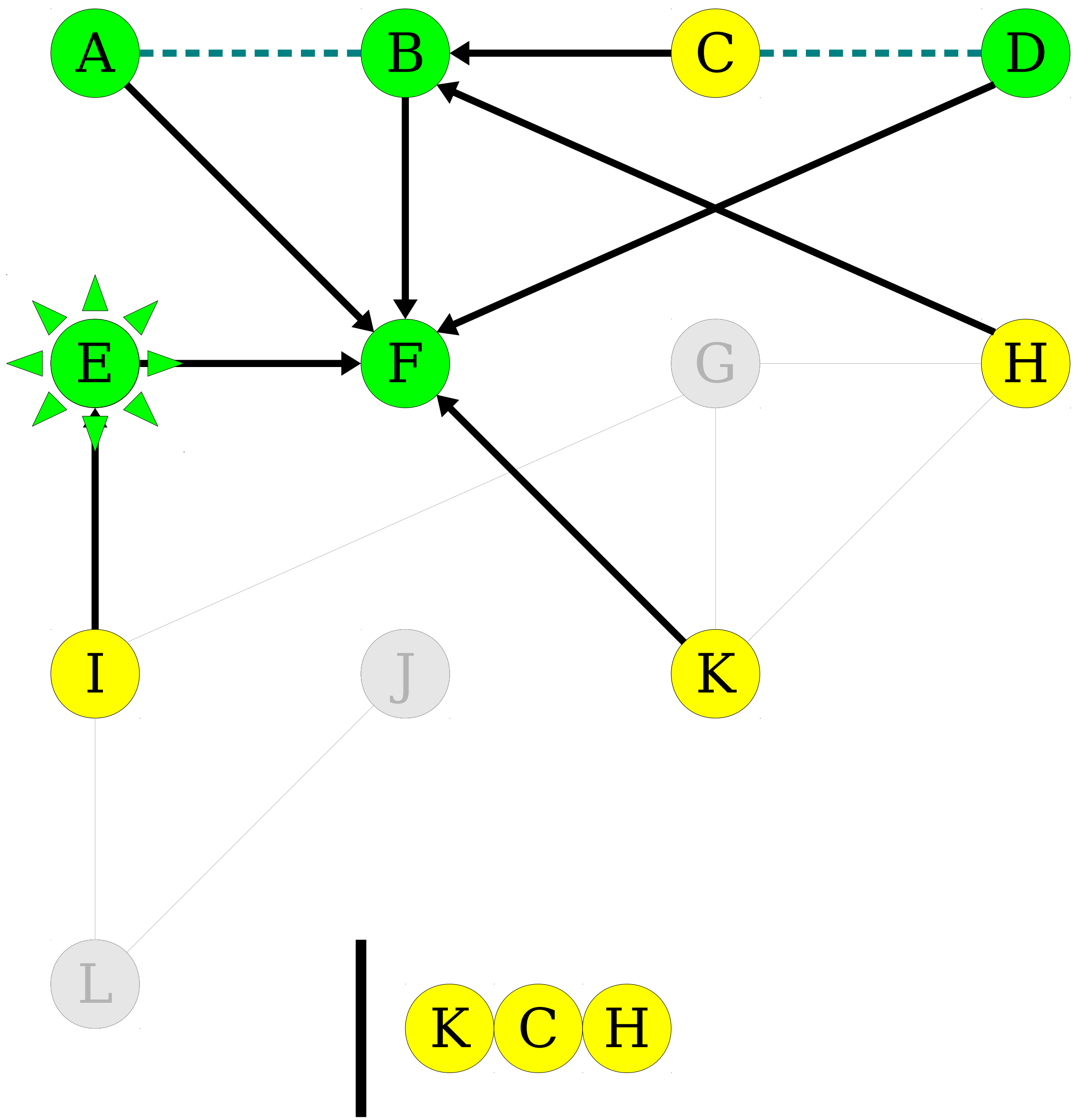
Breadth-First Search



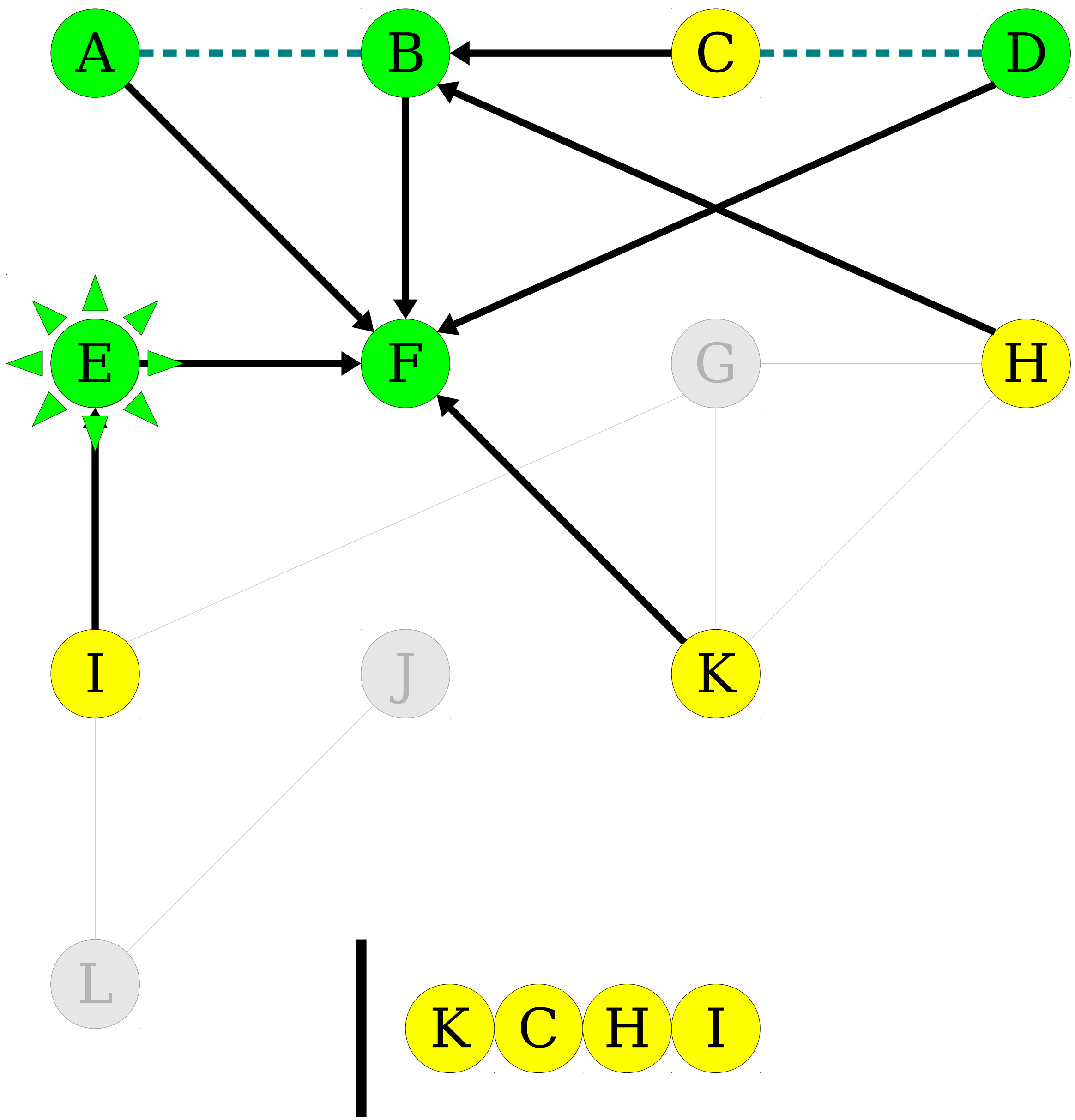
Breadth-First Search



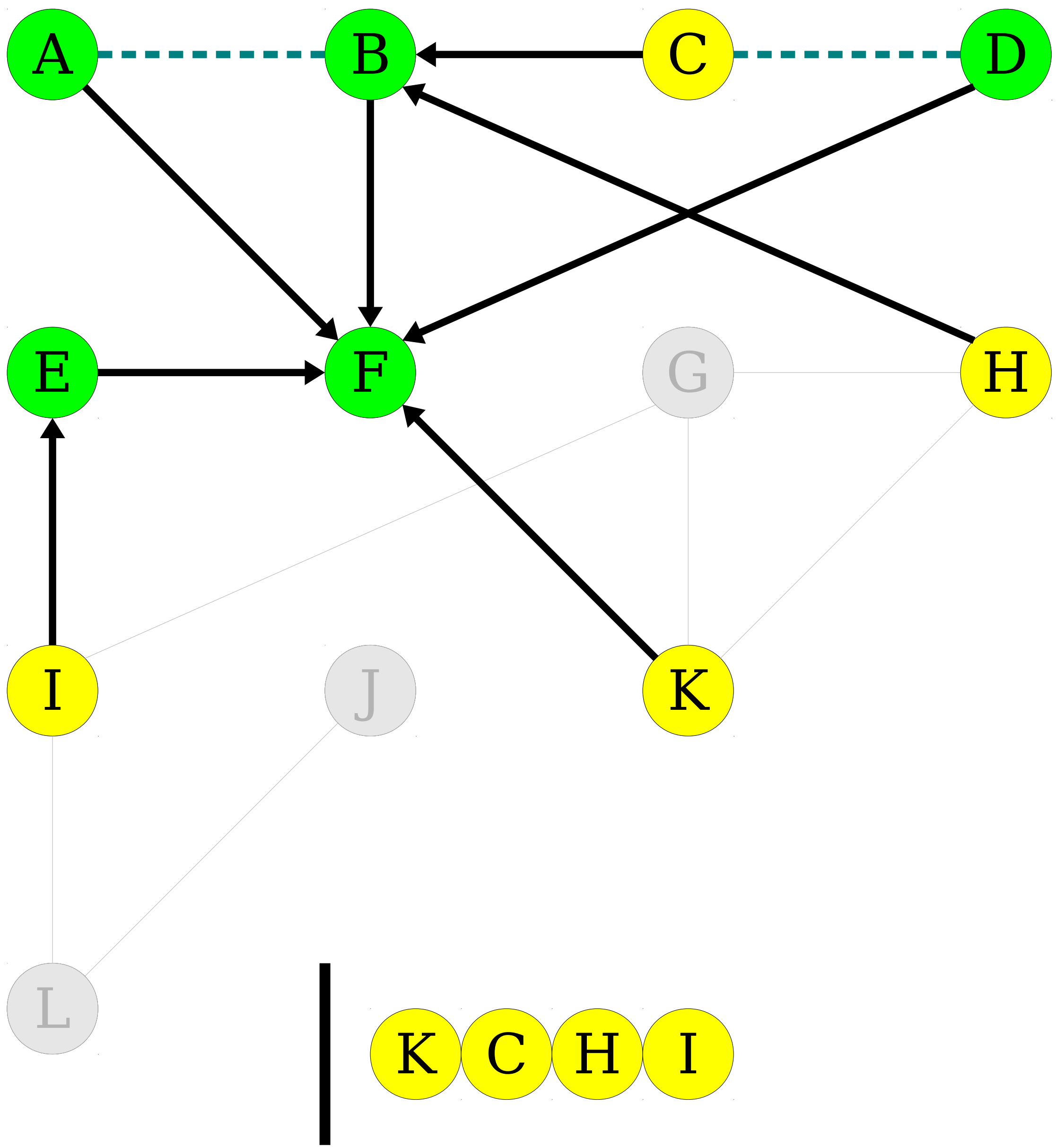
Breadth-First Search



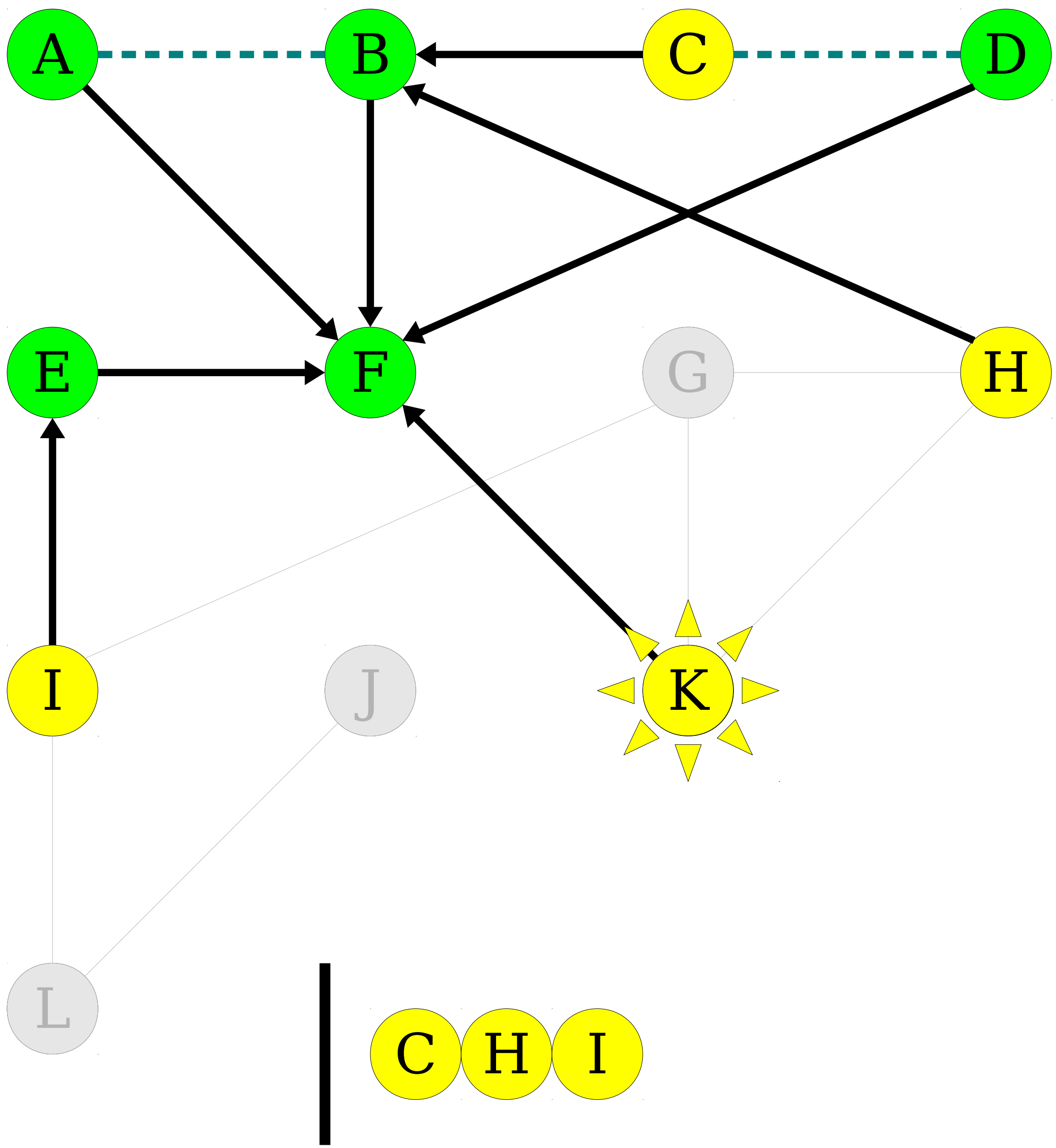
Breadth-First Search



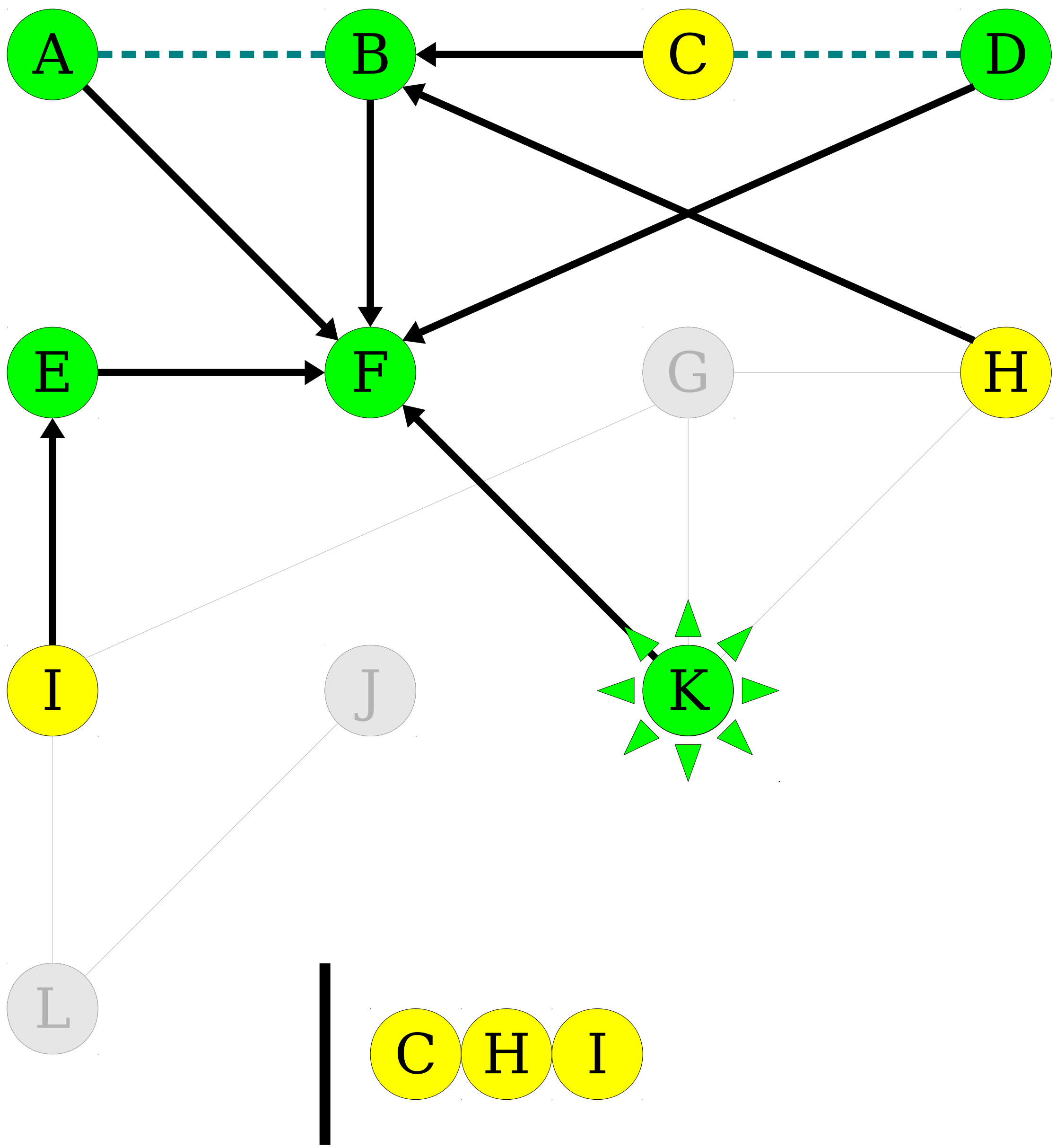
Breadth-First Search



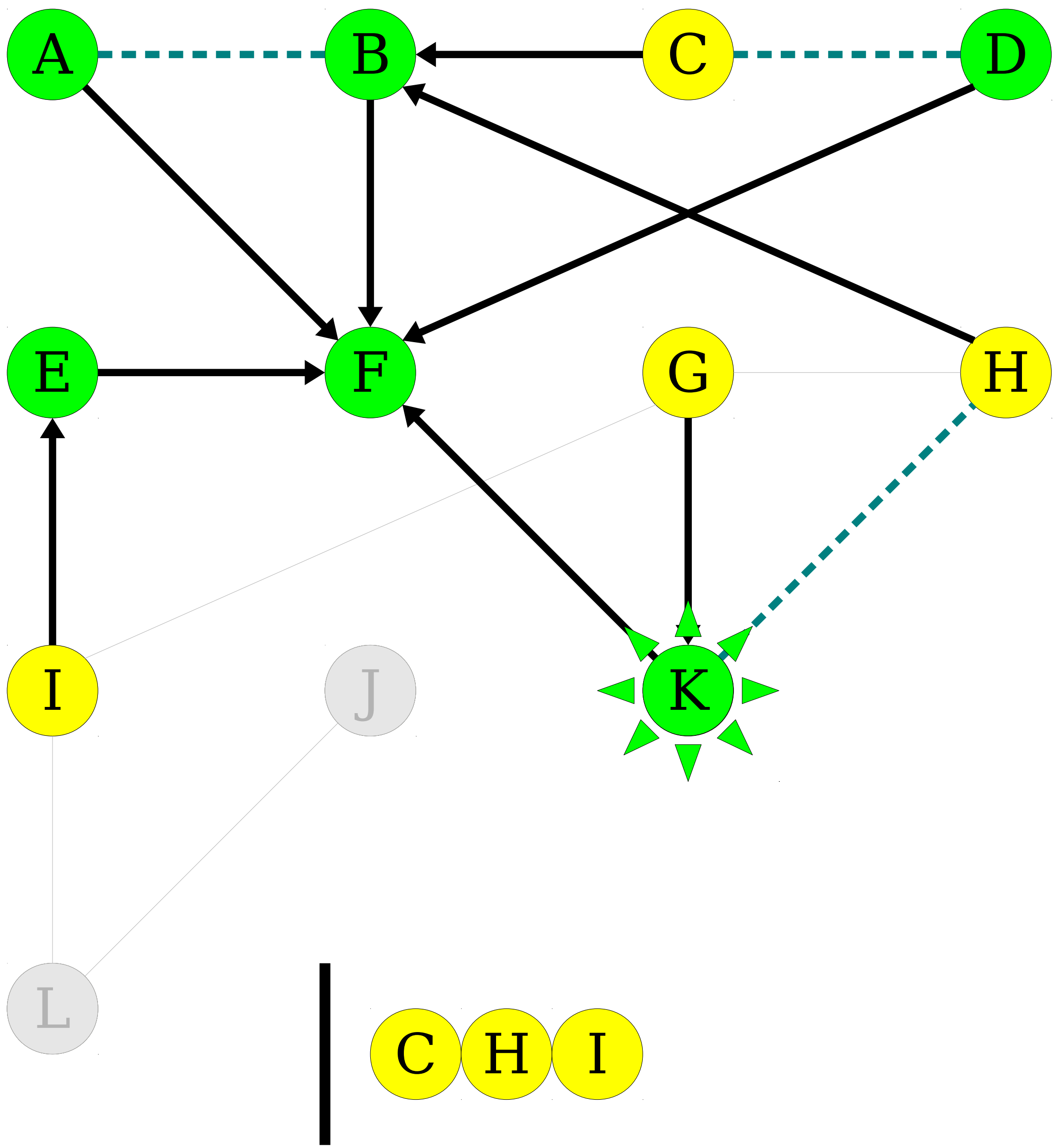
Breadth-First Search



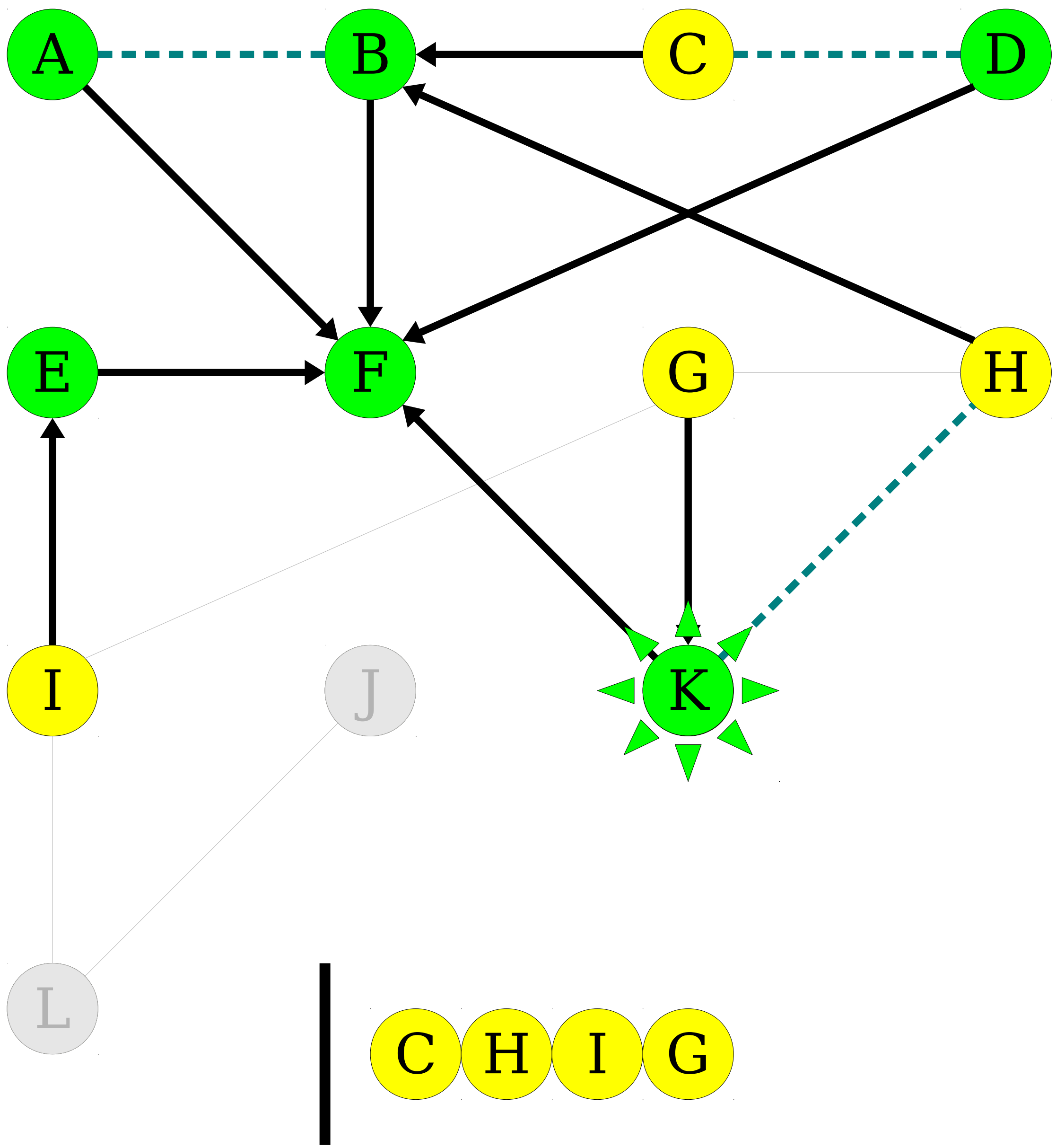
Breadth-First Search



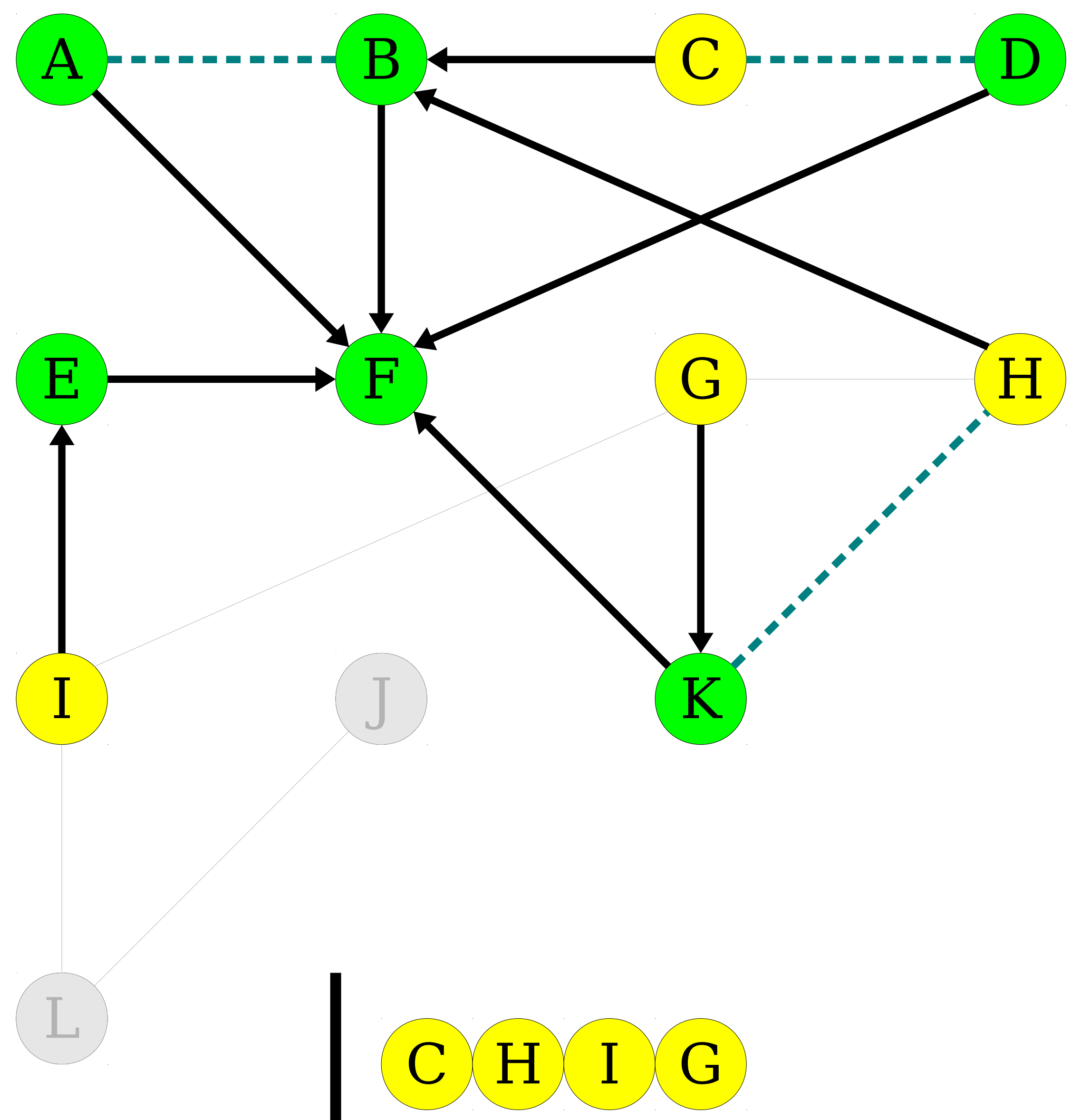
Breadth-First Search



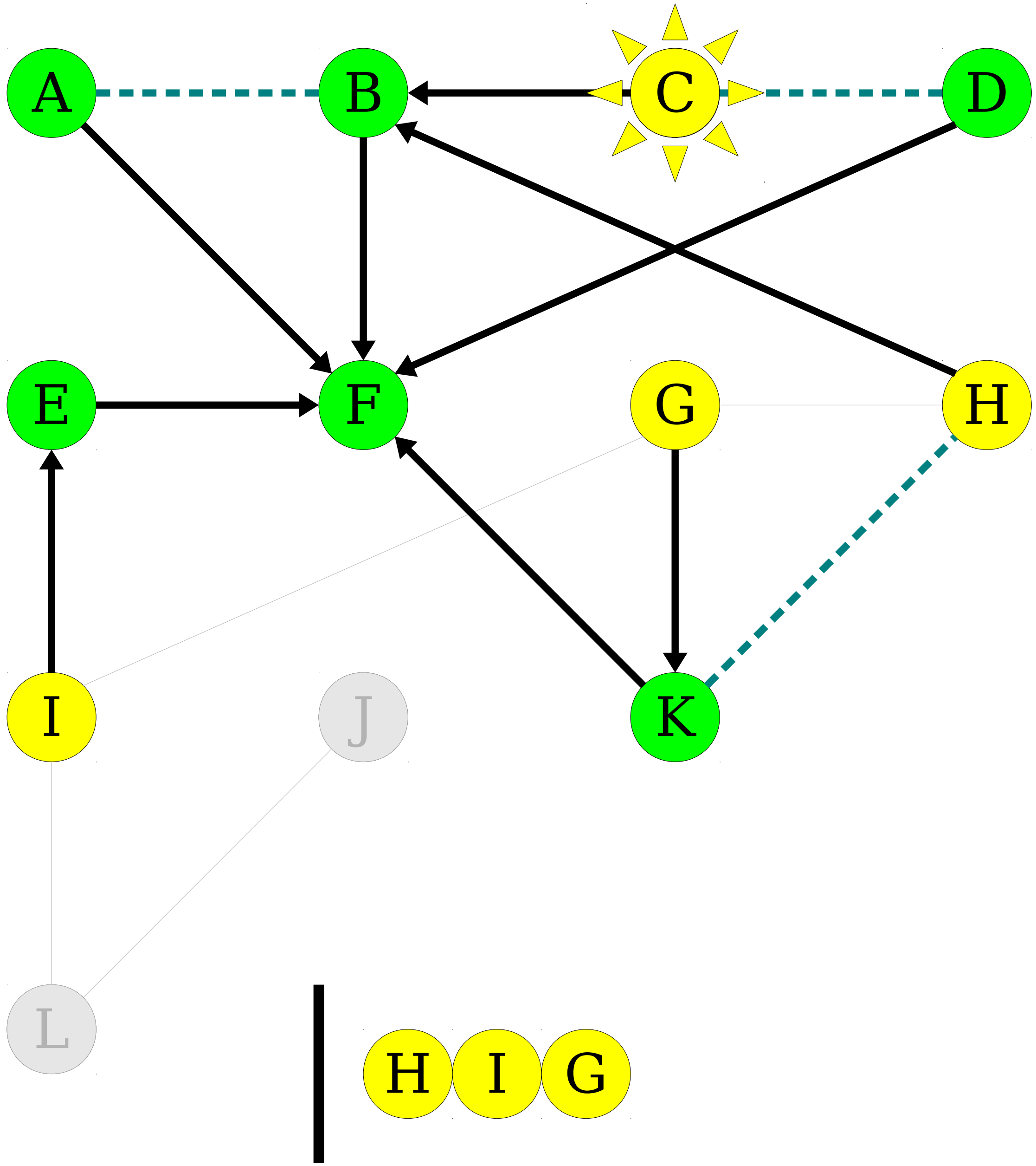
Breadth-First Search



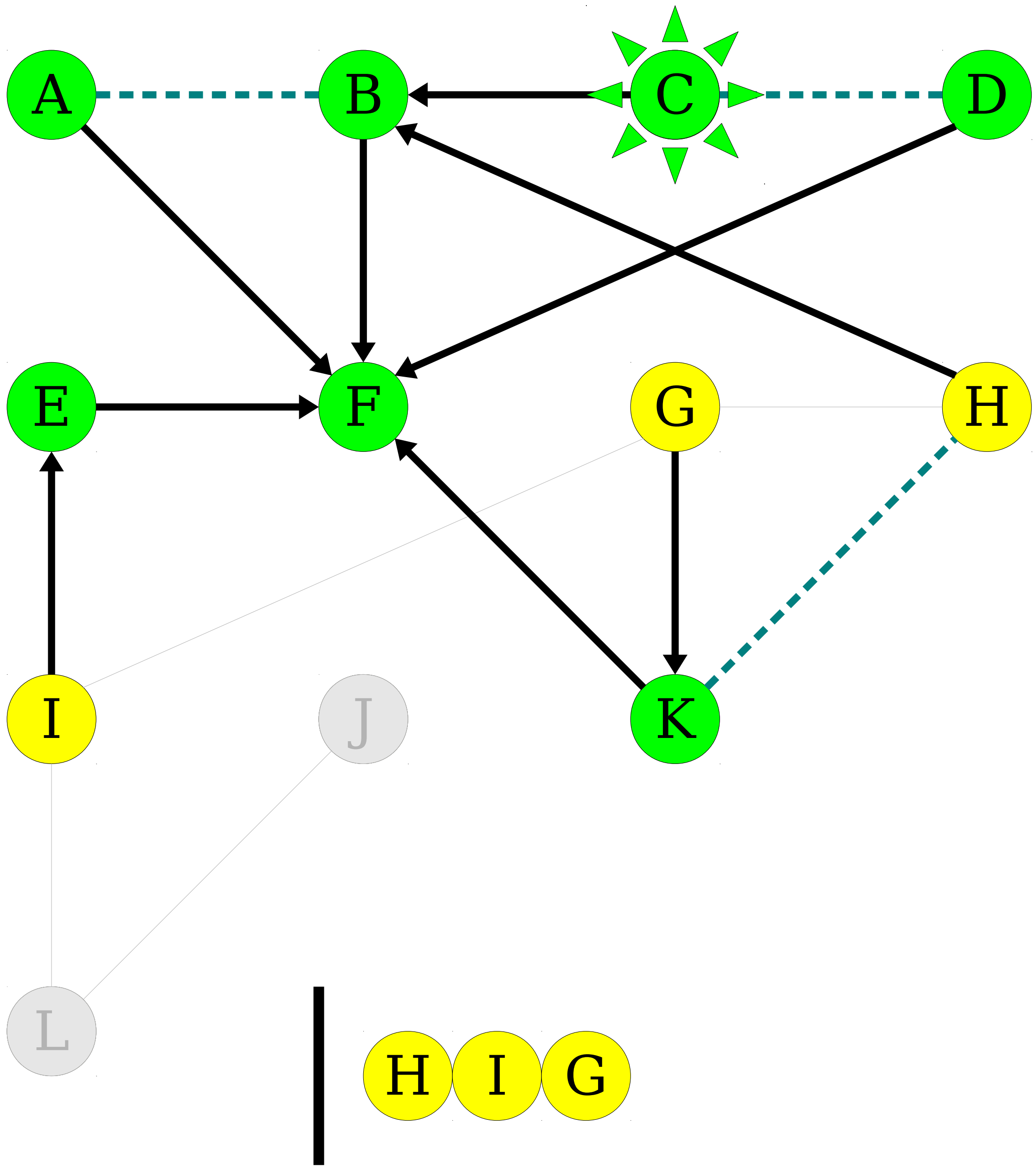
Breadth-First Search



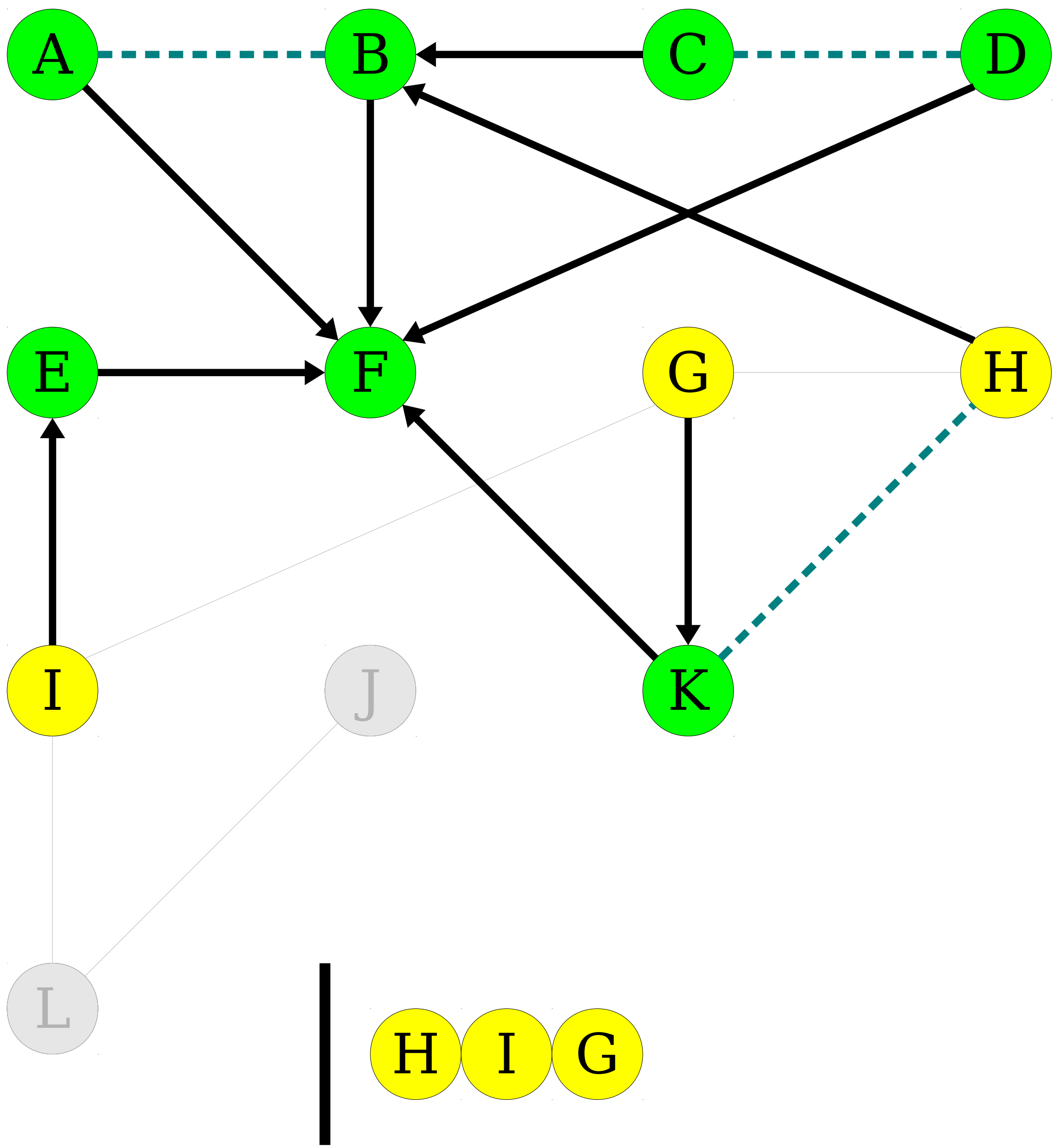
Breadth-First Search



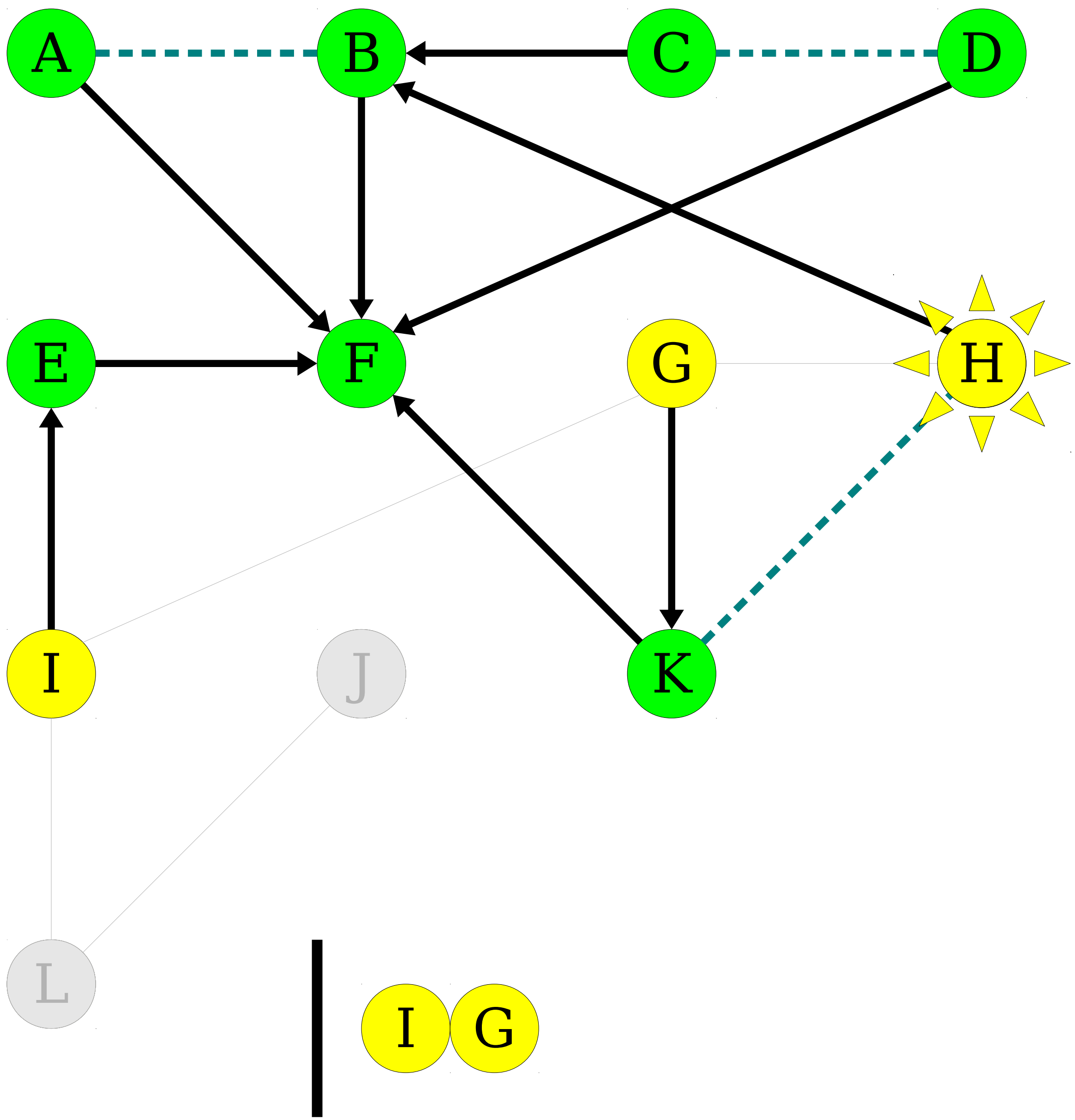
Breadth-First Search



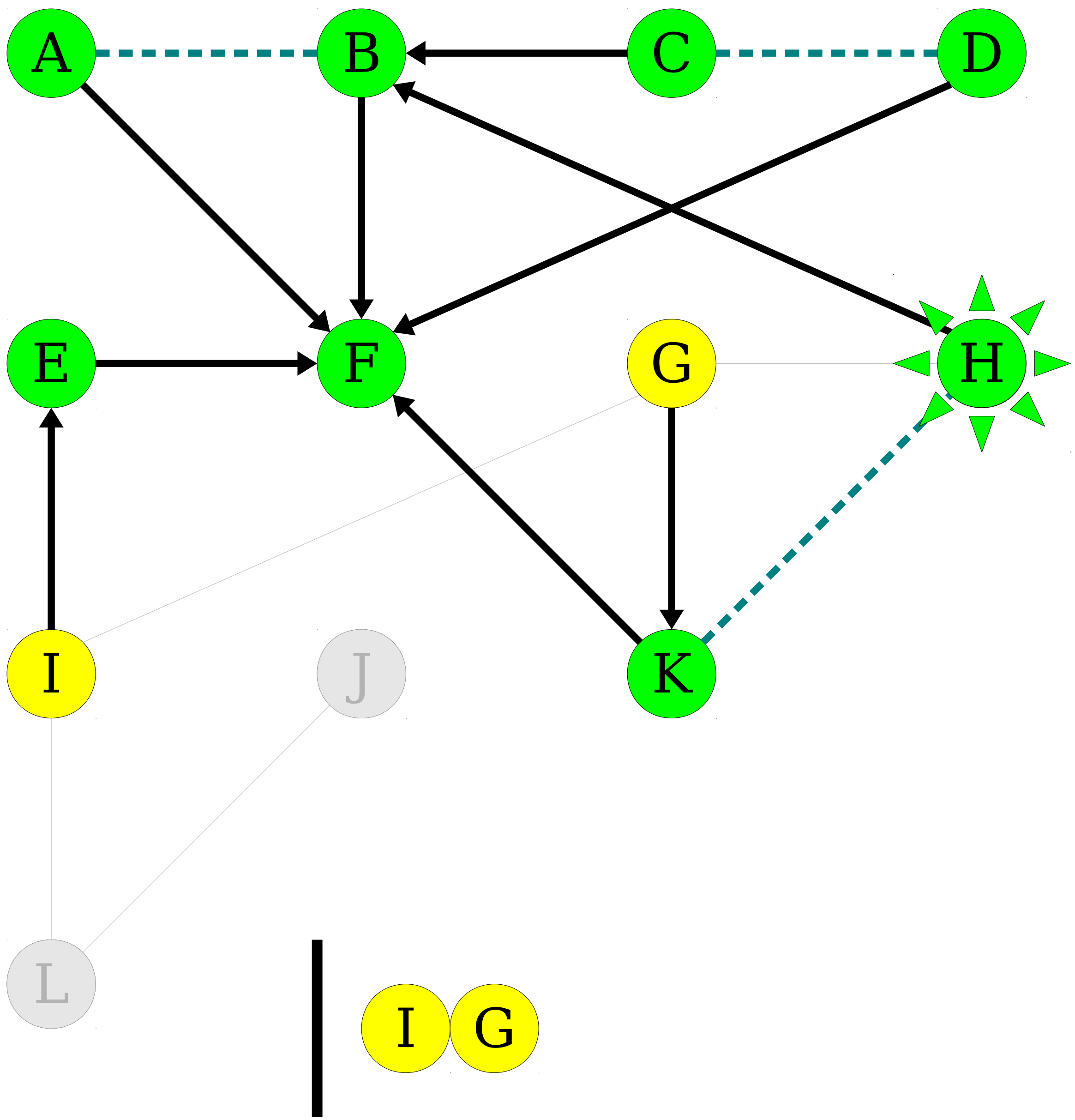
Breadth-First Search



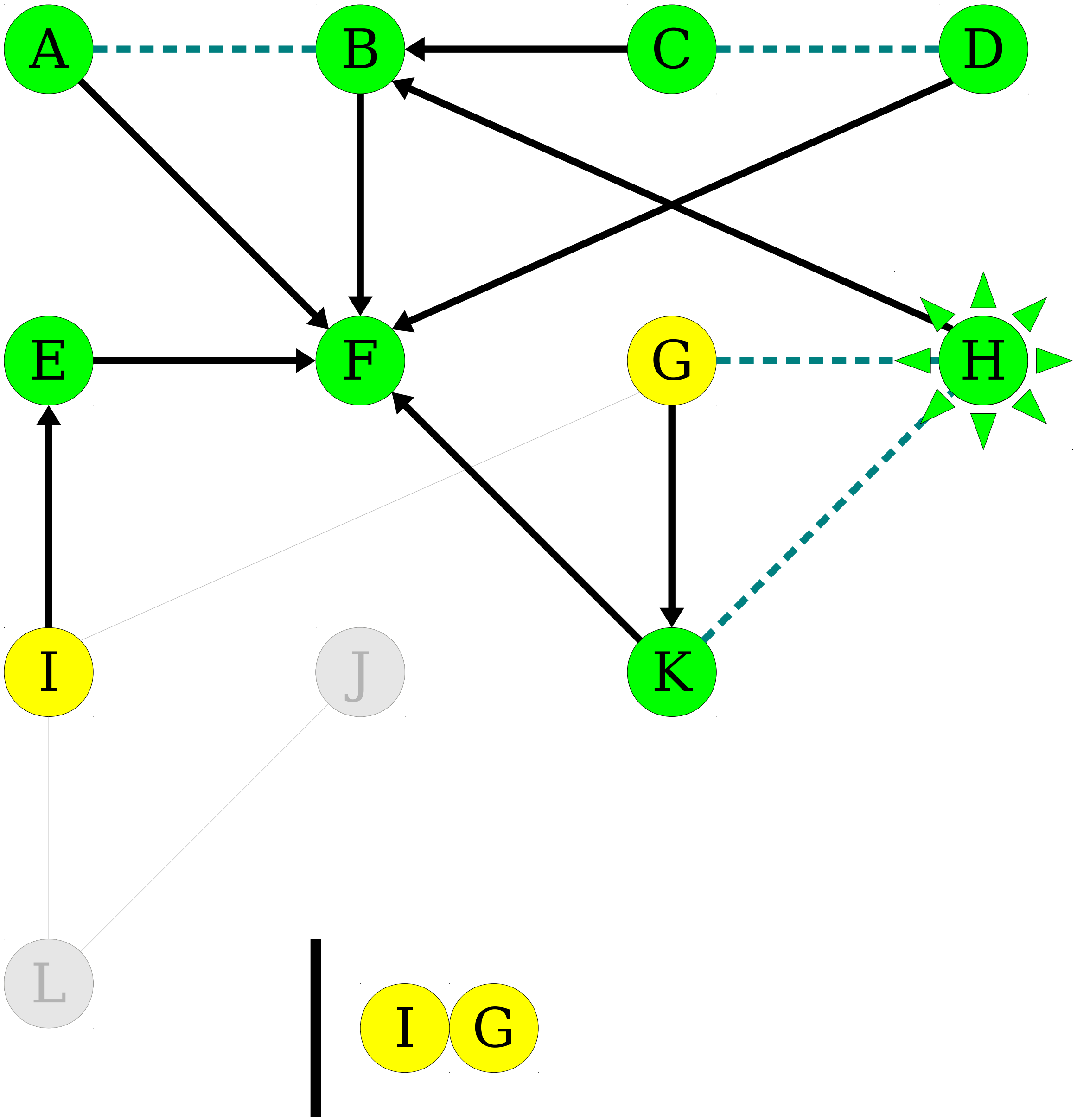
Breadth-First Search



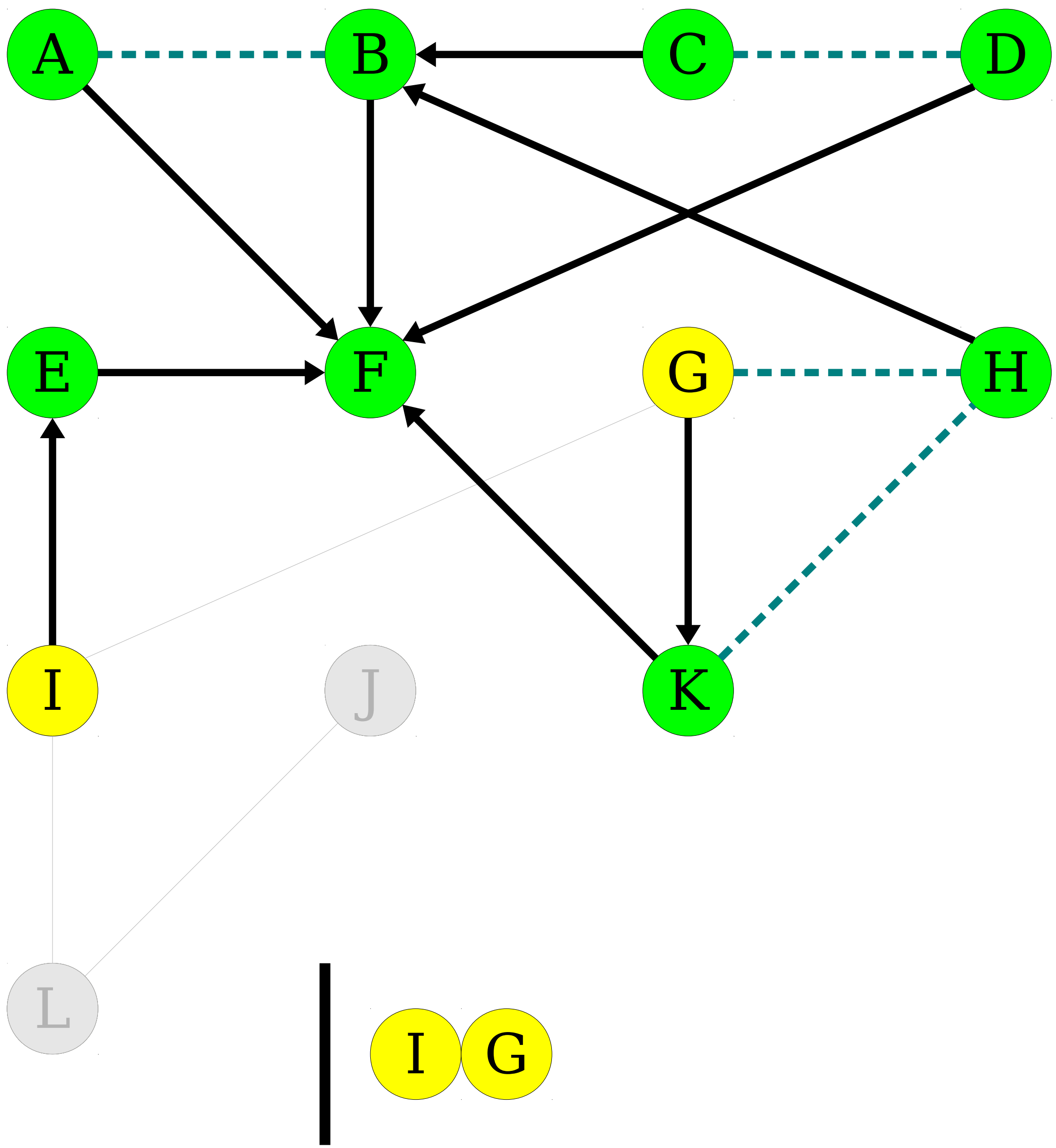
Breadth-First Search



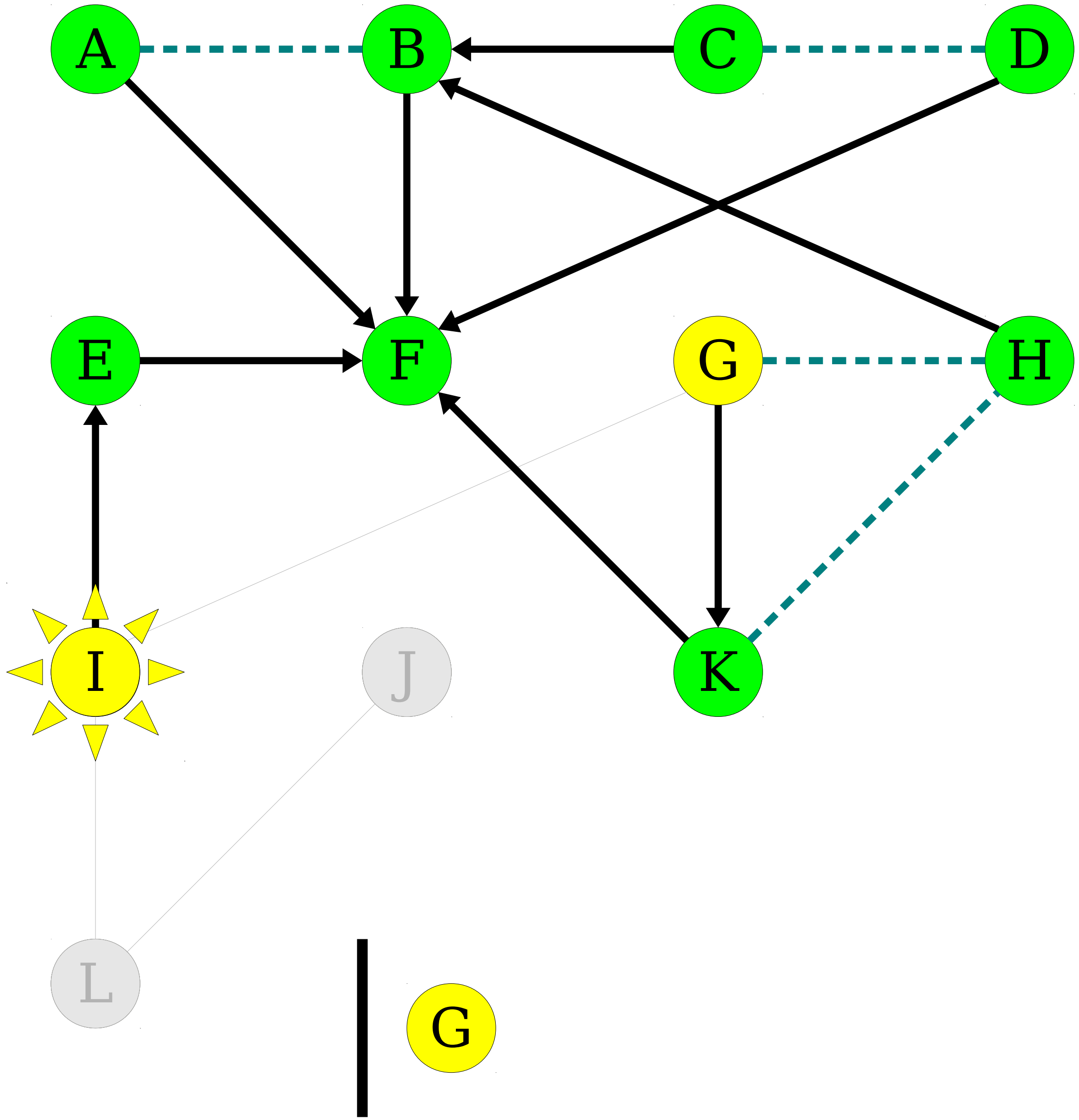
Breadth-First Search



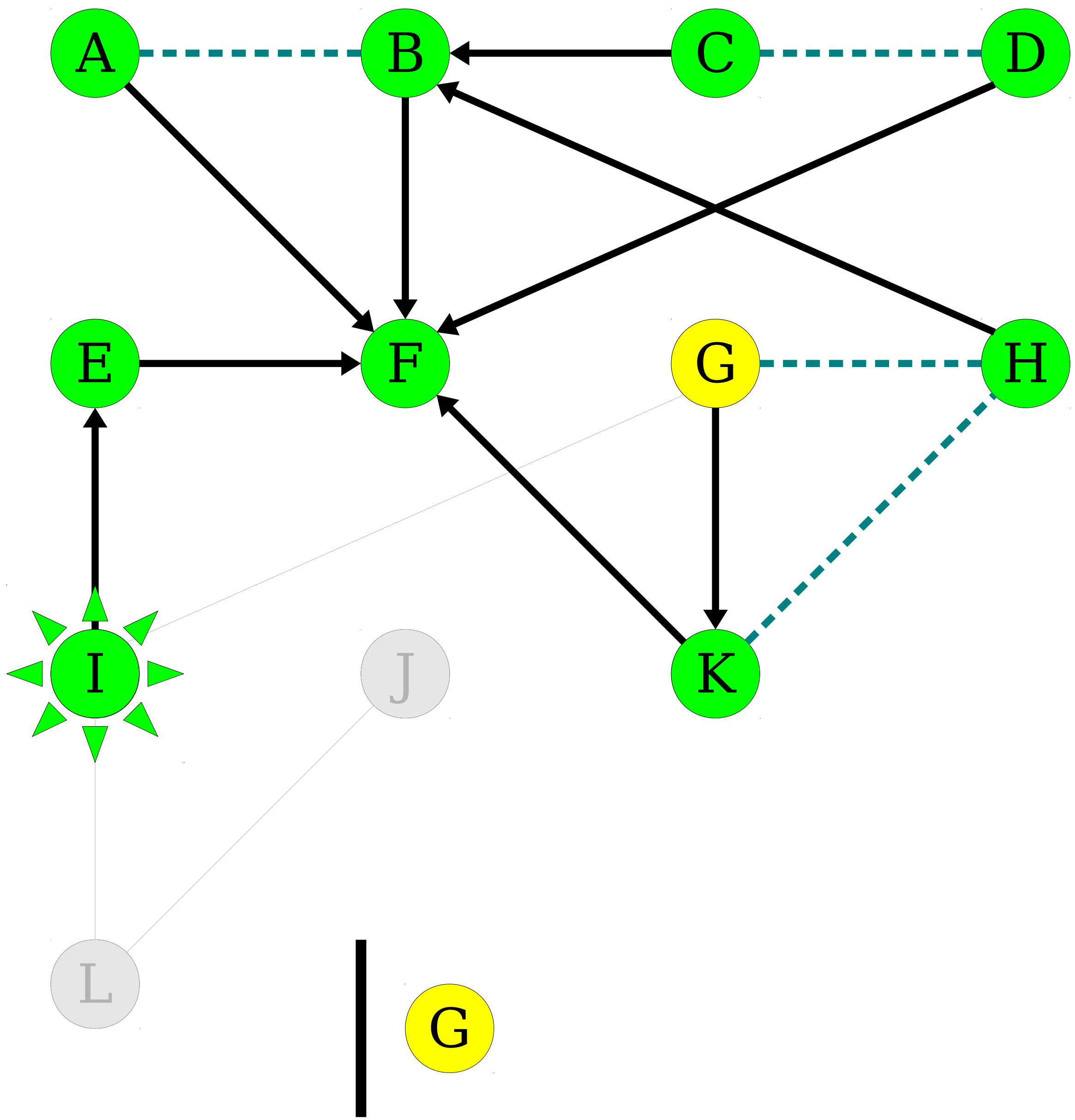
Breadth-First Search



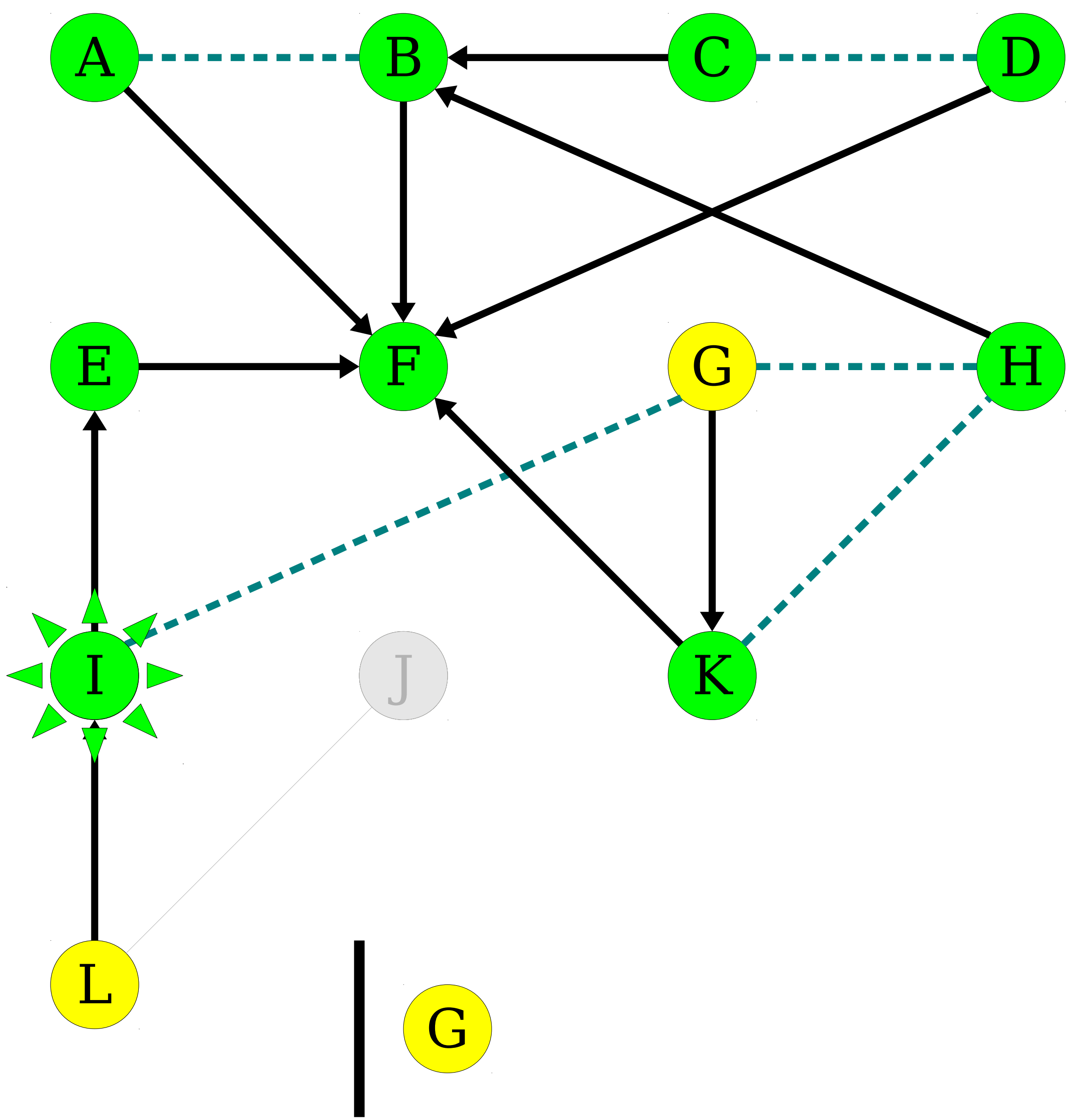
Breadth-First Search



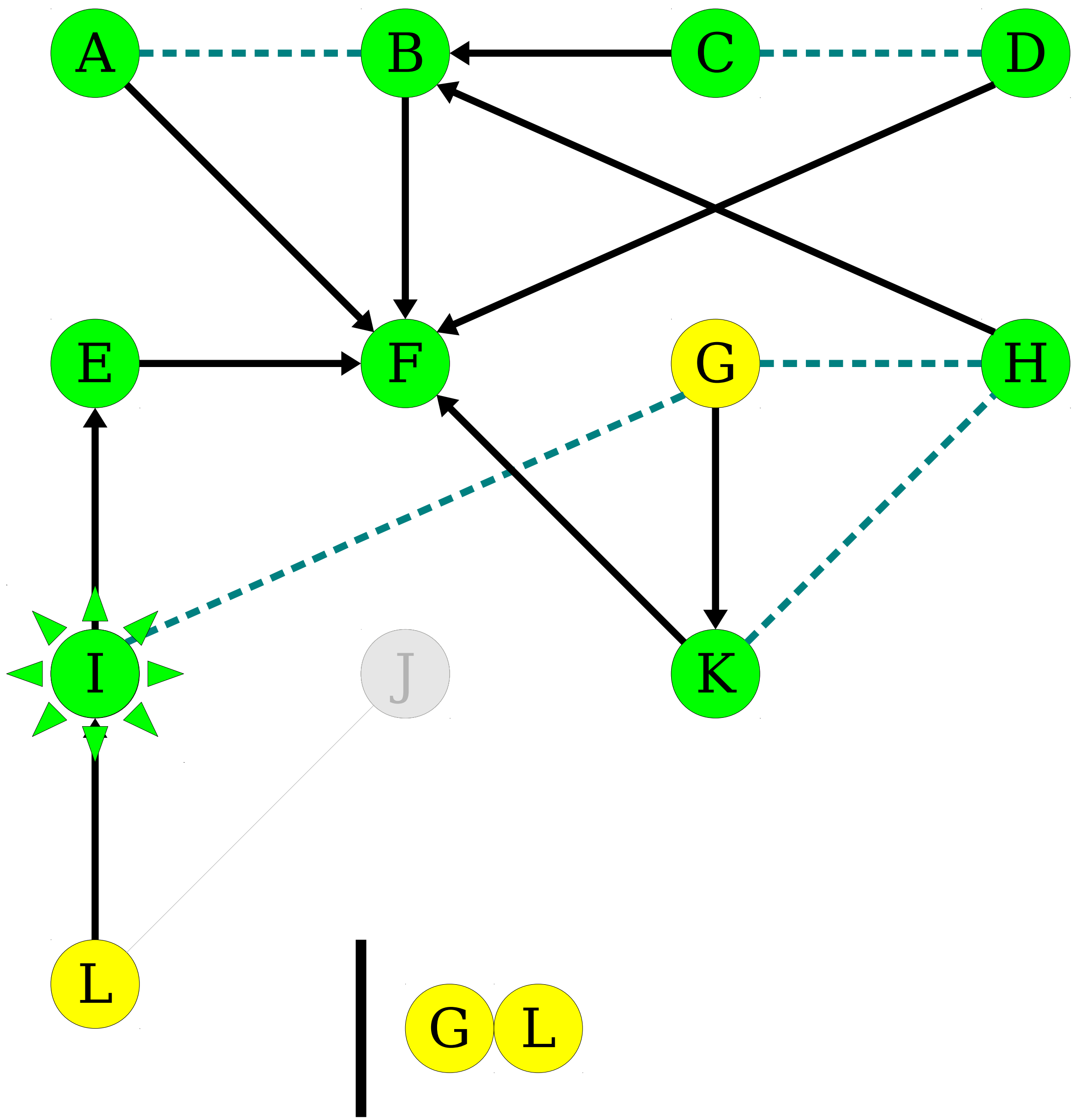
Breadth-First Search



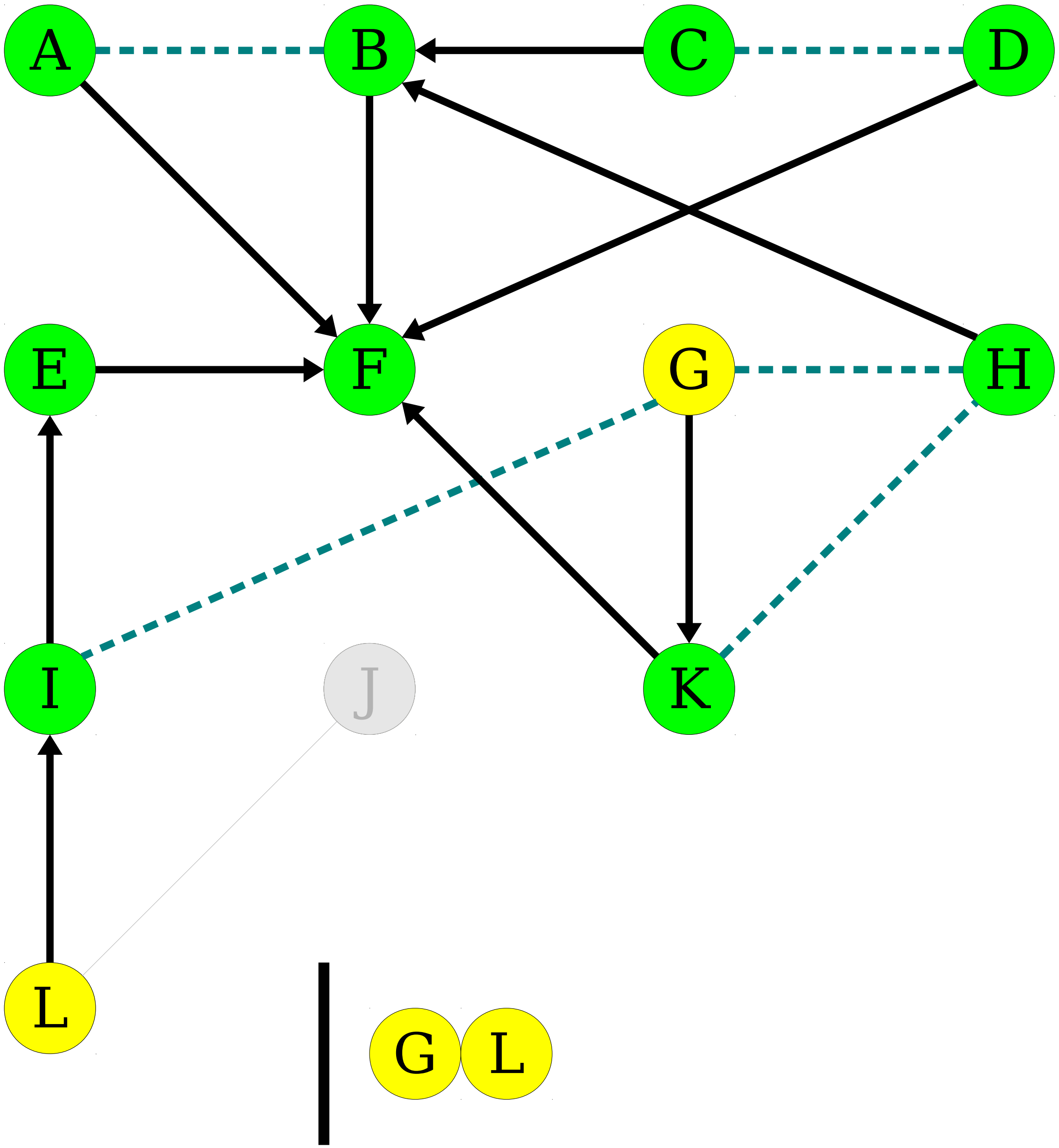
Breadth-First Search



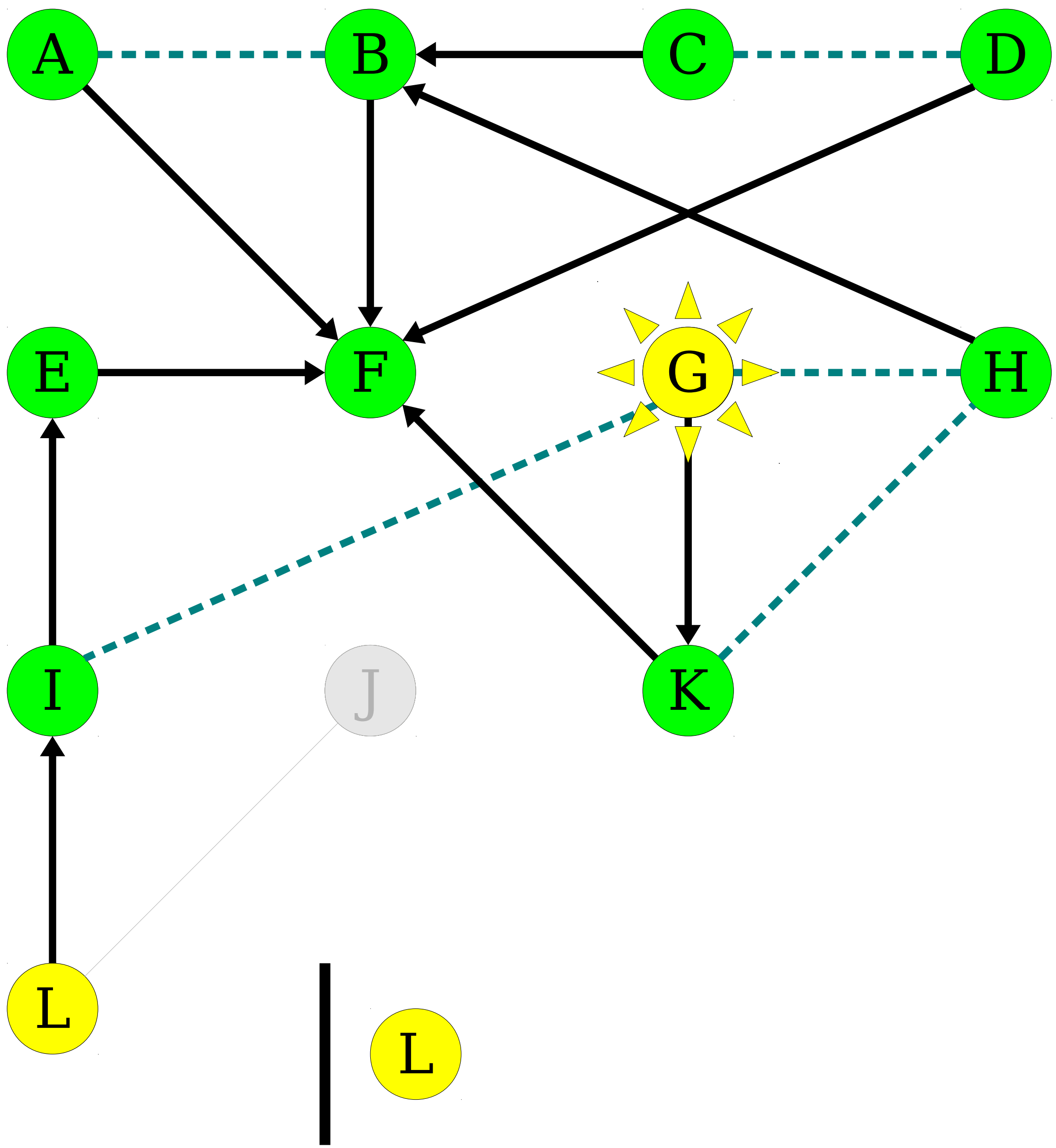
Breadth-First Search



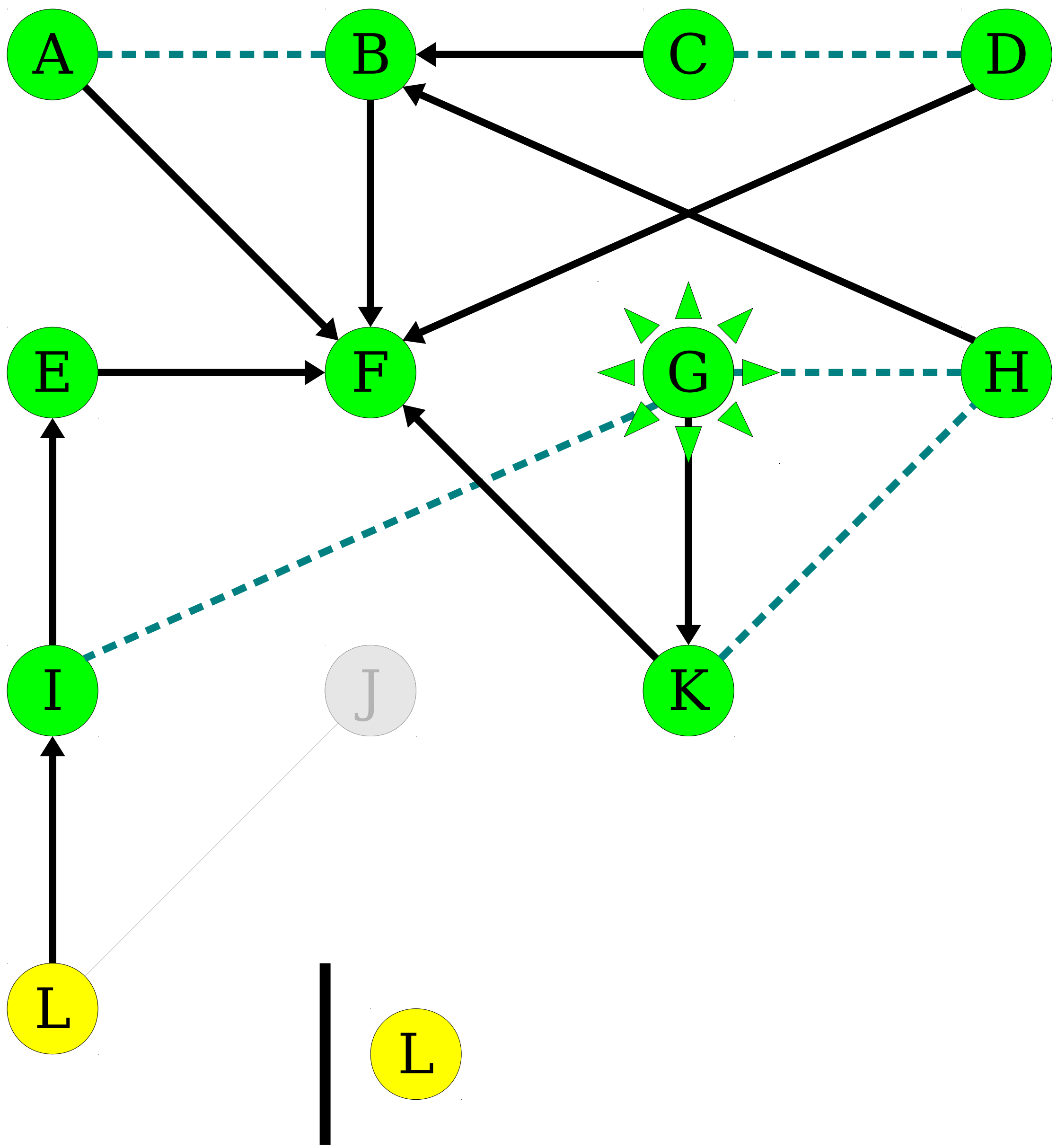
Breadth-First Search



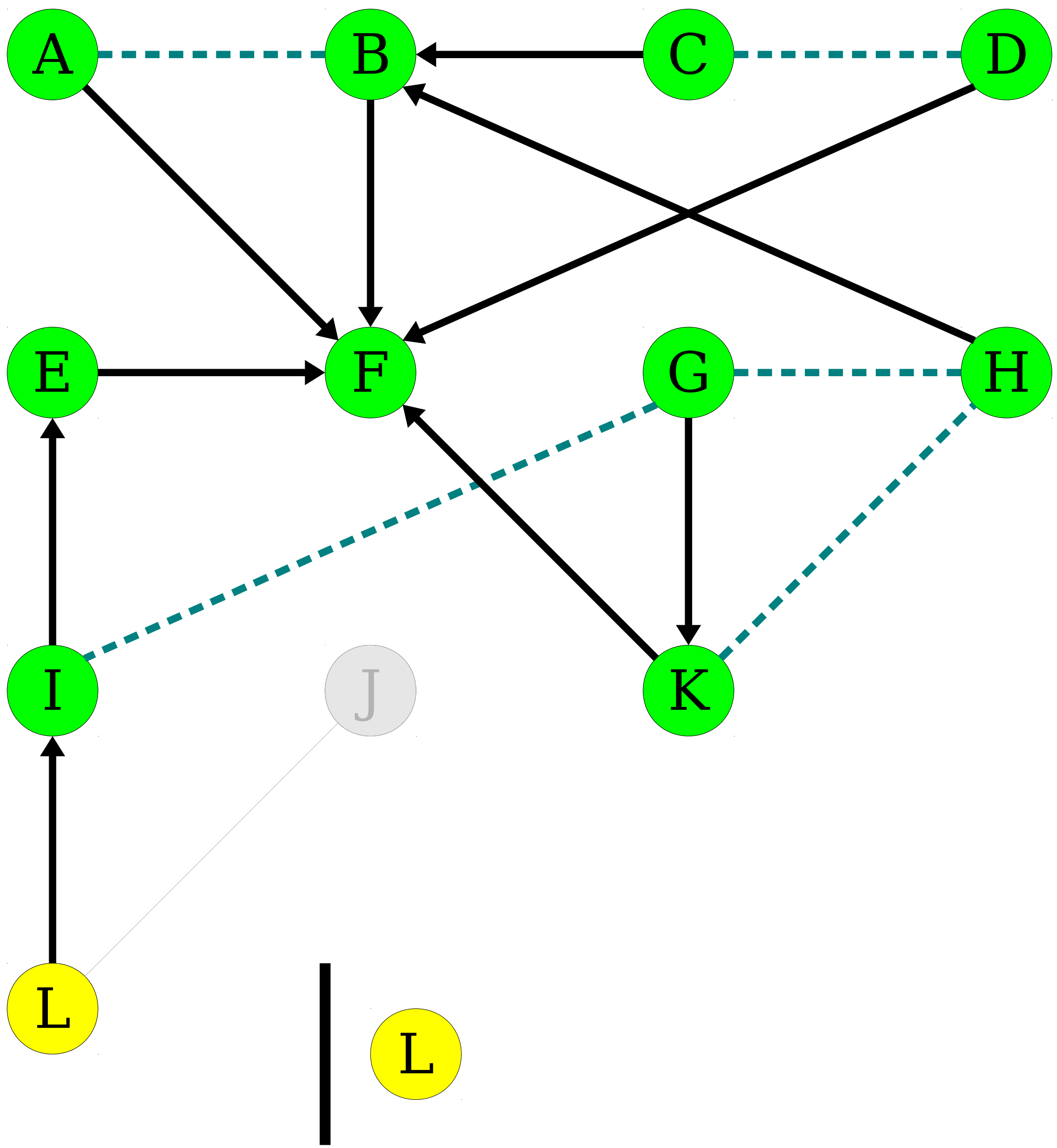
Breadth-First Search



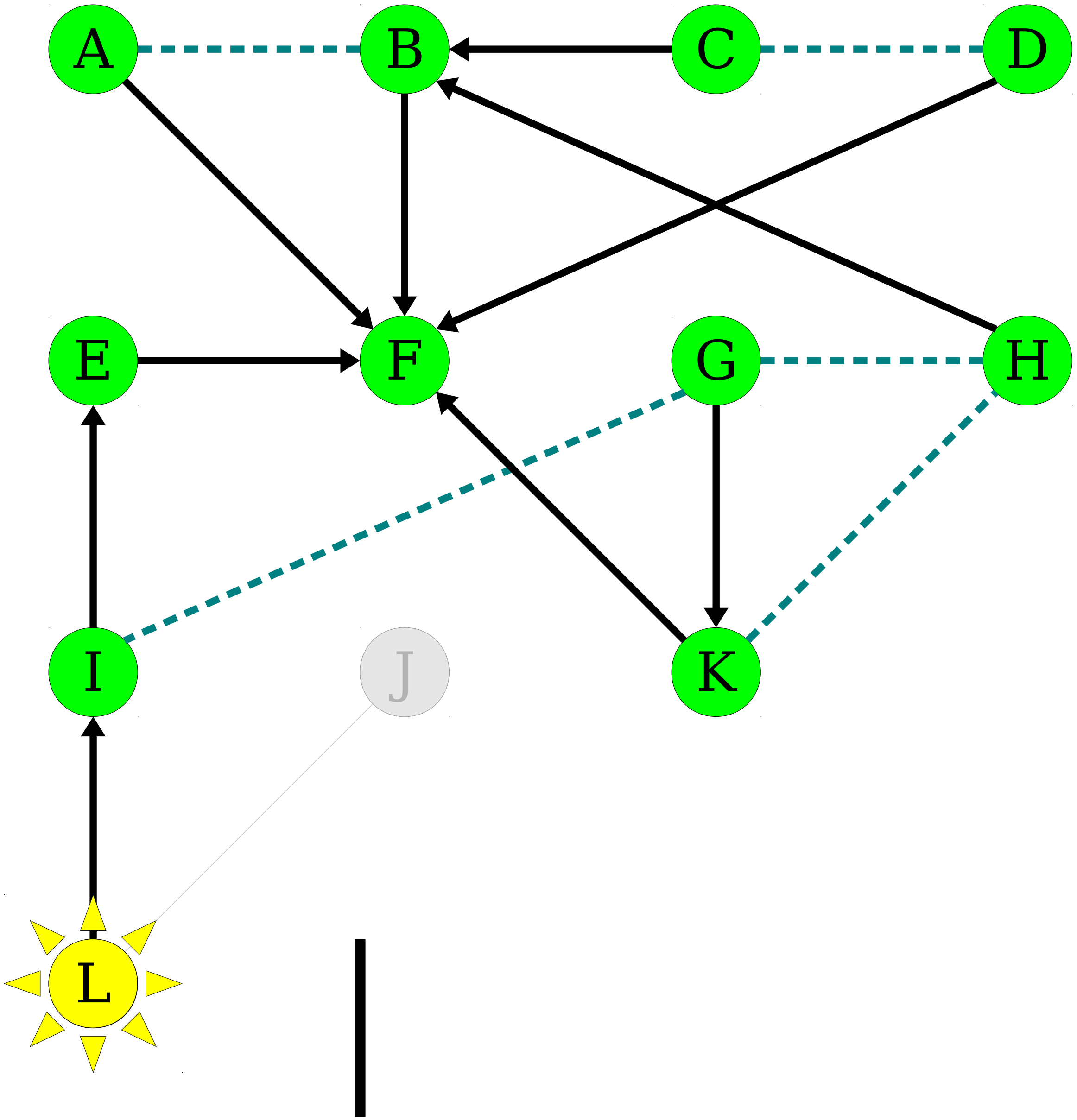
Breadth-First Search



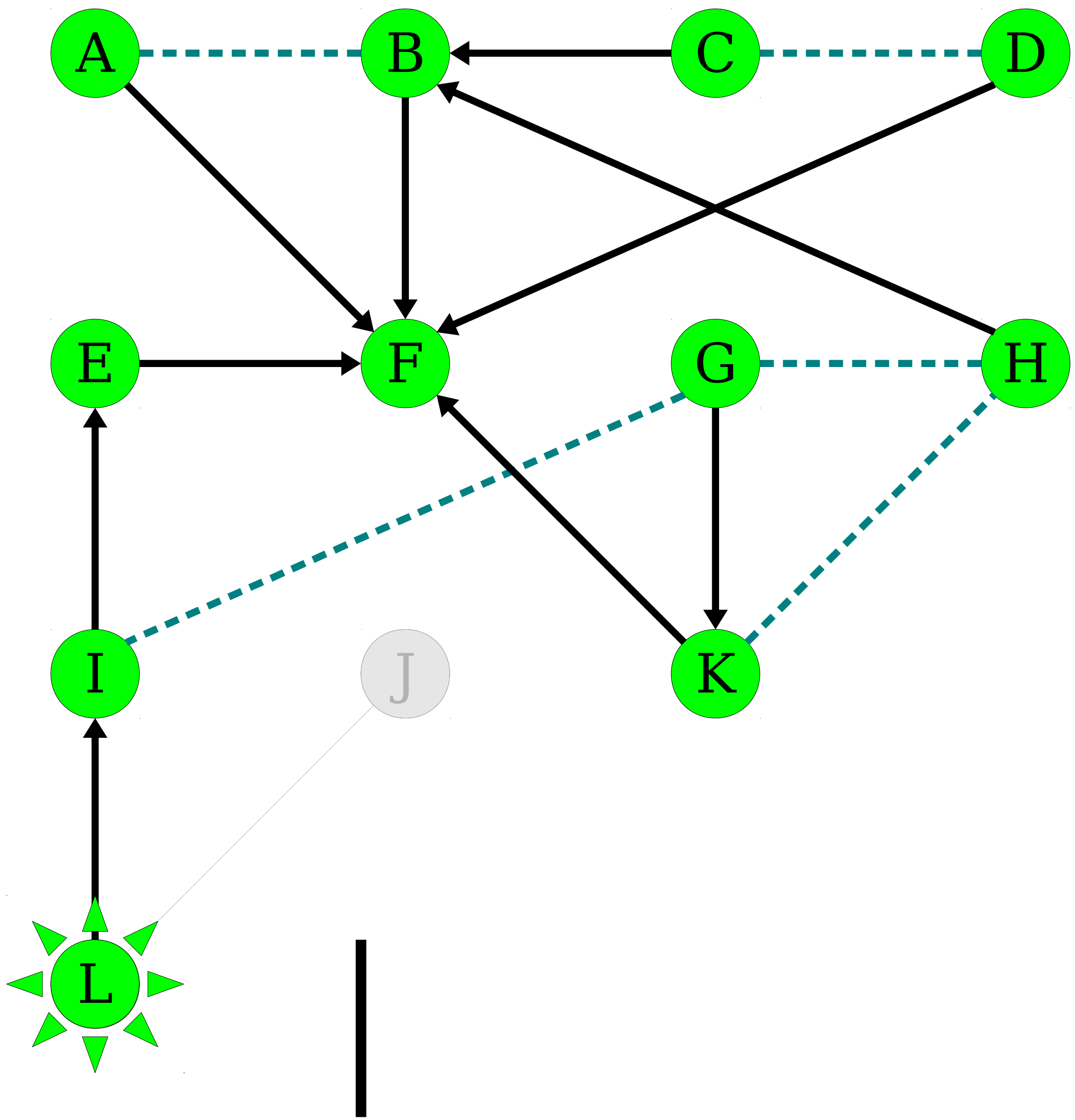
Breadth-First Search



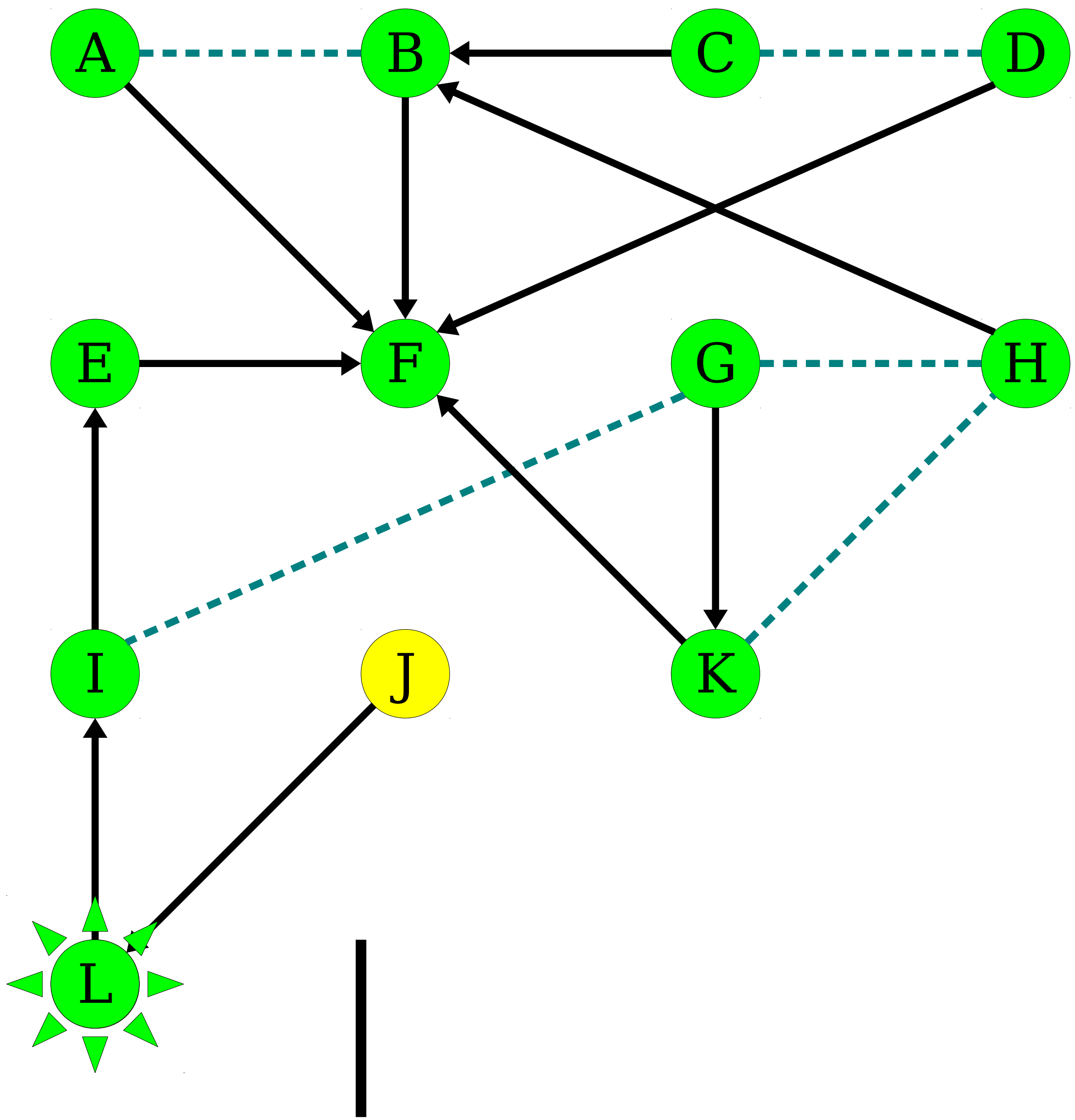
Breadth-First Search



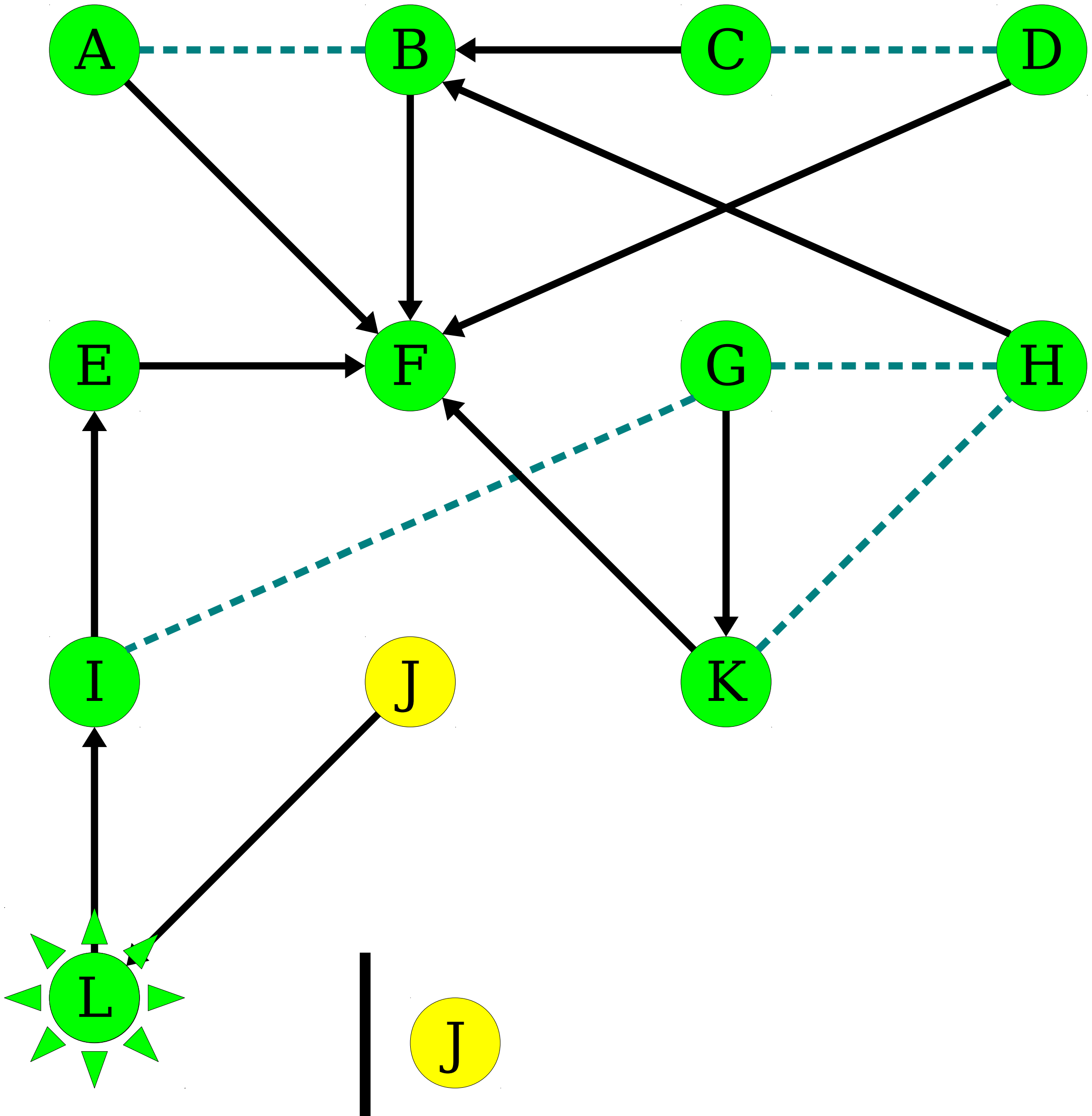
Breadth-First Search



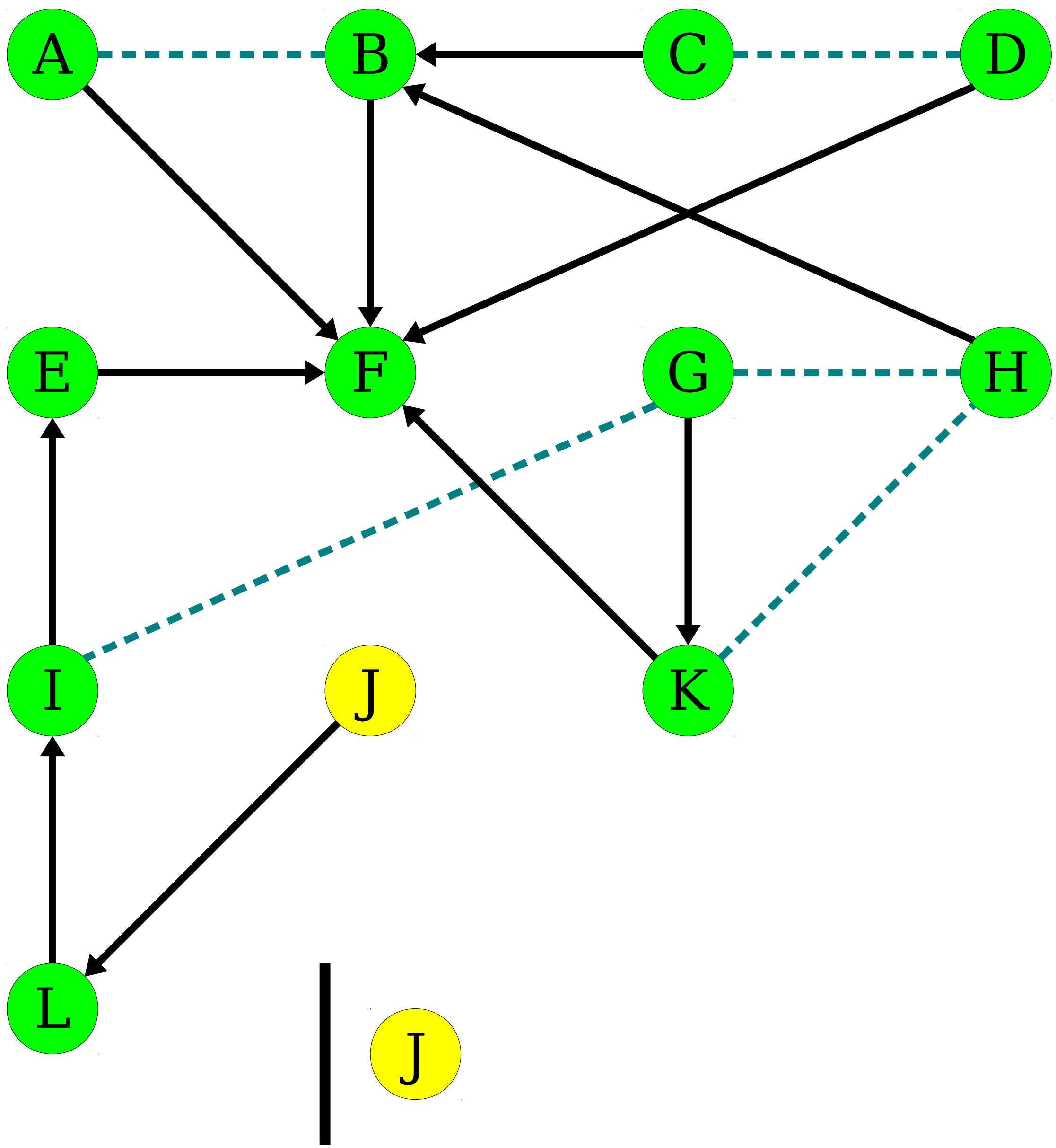
Breadth-First Search



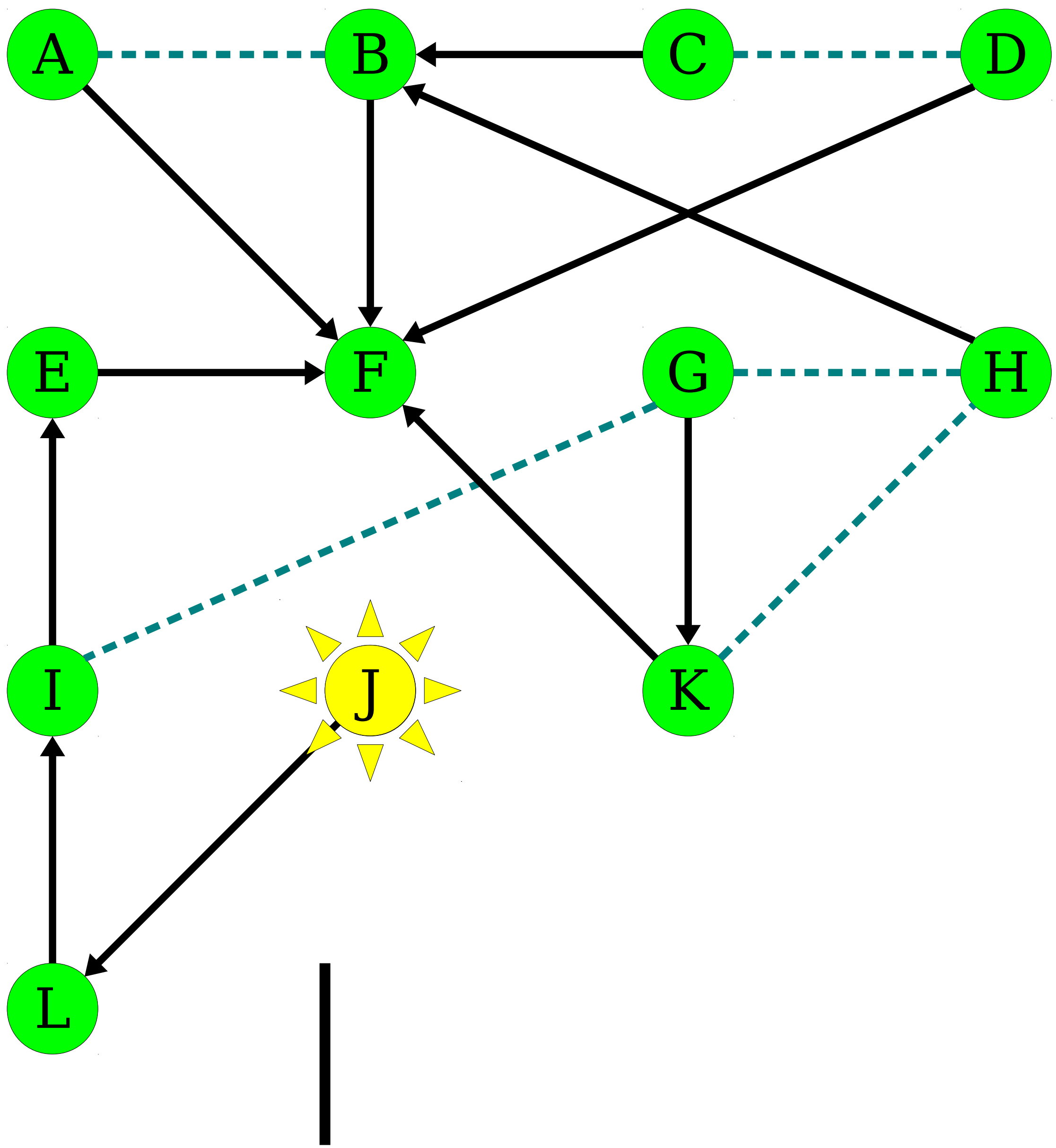
Breadth-First Search



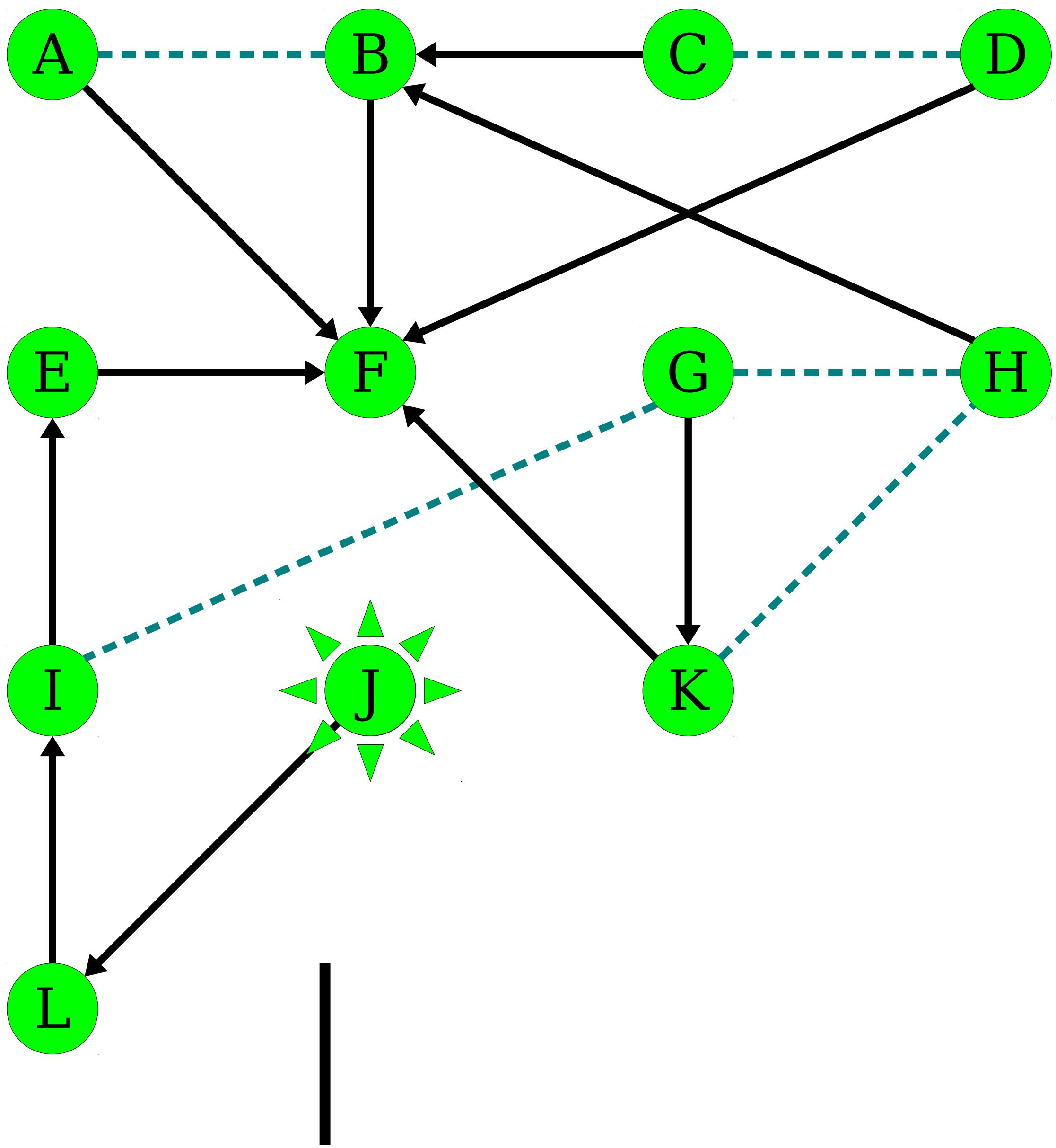
Breadth-First Search



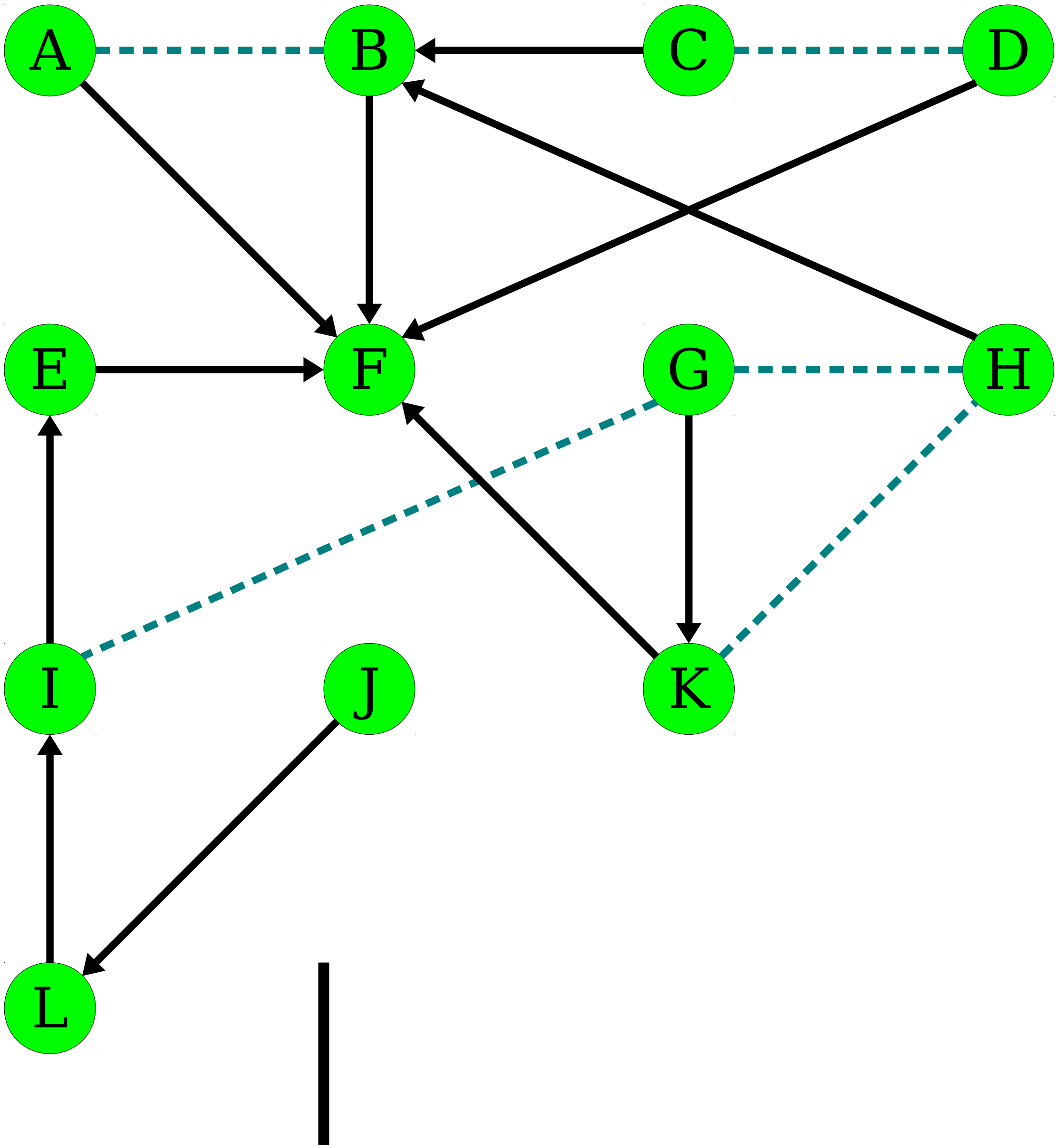
Breadth-First Search



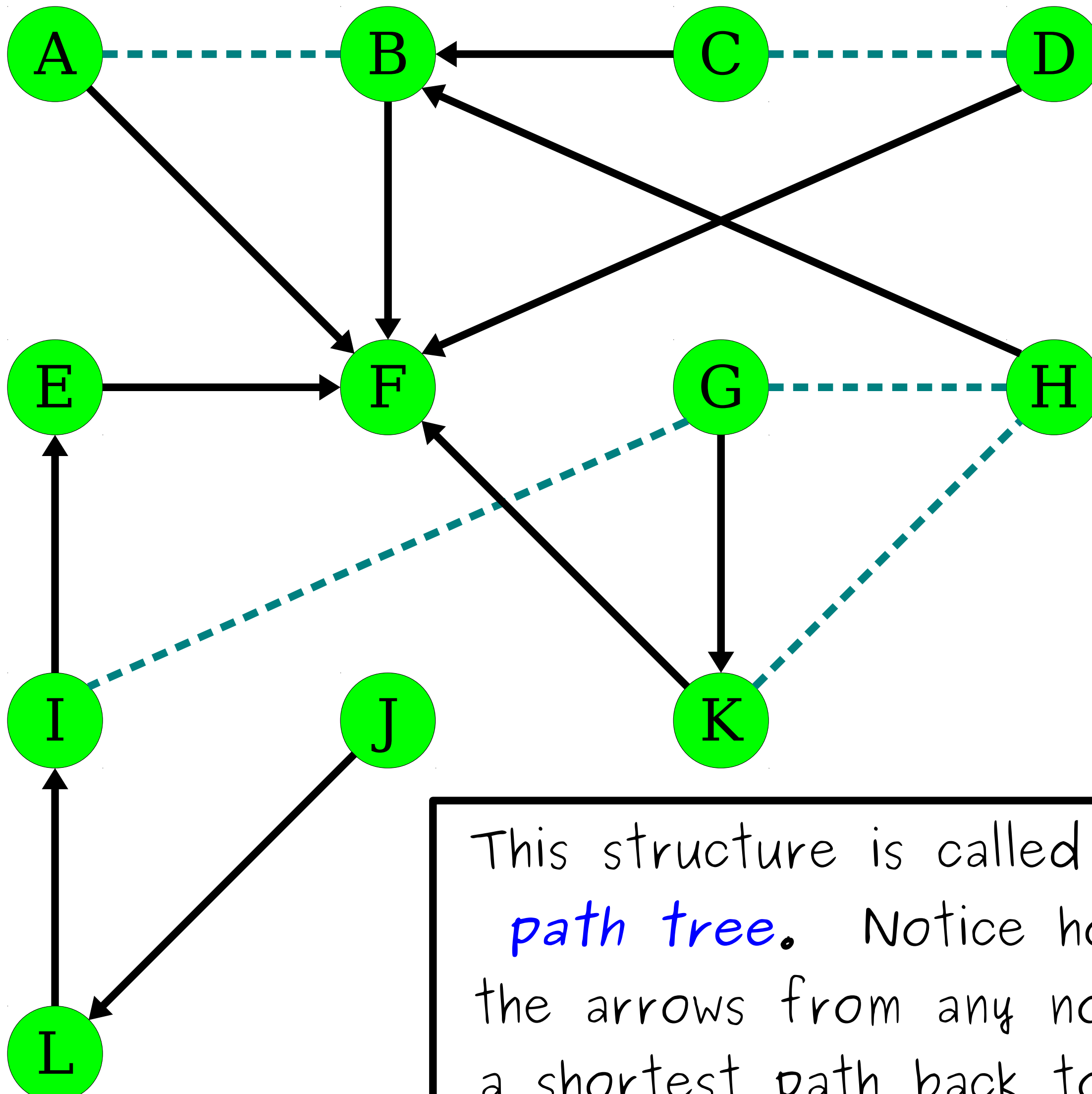
Breadth-First Search



Breadth-First Search



Breadth-First Search



This structure is called a *shortest-path tree*. Notice how following the arrows from any node will trace a shortest path back to the root in reverse.

BFS: The Rundown

- Breadth-first search will find all nodes reachable from the starting node.
- It will visit them in *increasing order of distance*.
- In an n-node, m-edge graph, it takes time $O(m + n)$ and uses space $O(n)$.
  - Though in practice, the space usage is much higher than in DFS.
  - Curious where this runtime comes from? Come talk to me after class!

BFS: The Logic

- Color all nodes ***gray***.
- Make a worklist with the starting node.
- Color the starting node ***yellow***.
- While the worklist isn't empty:
  - Dequeue a node from the worklist.
  - Color that node ***green***.
  - For each adjacent node:
    - If that node is ***gray***:
      - Color the node ***yellow***.
      - Add it to the queue.

Next Time

- ***Shortest Paths***
  - Dijkstra's Algorithm.
  - Shortest-Path Trees
  - A* Search.