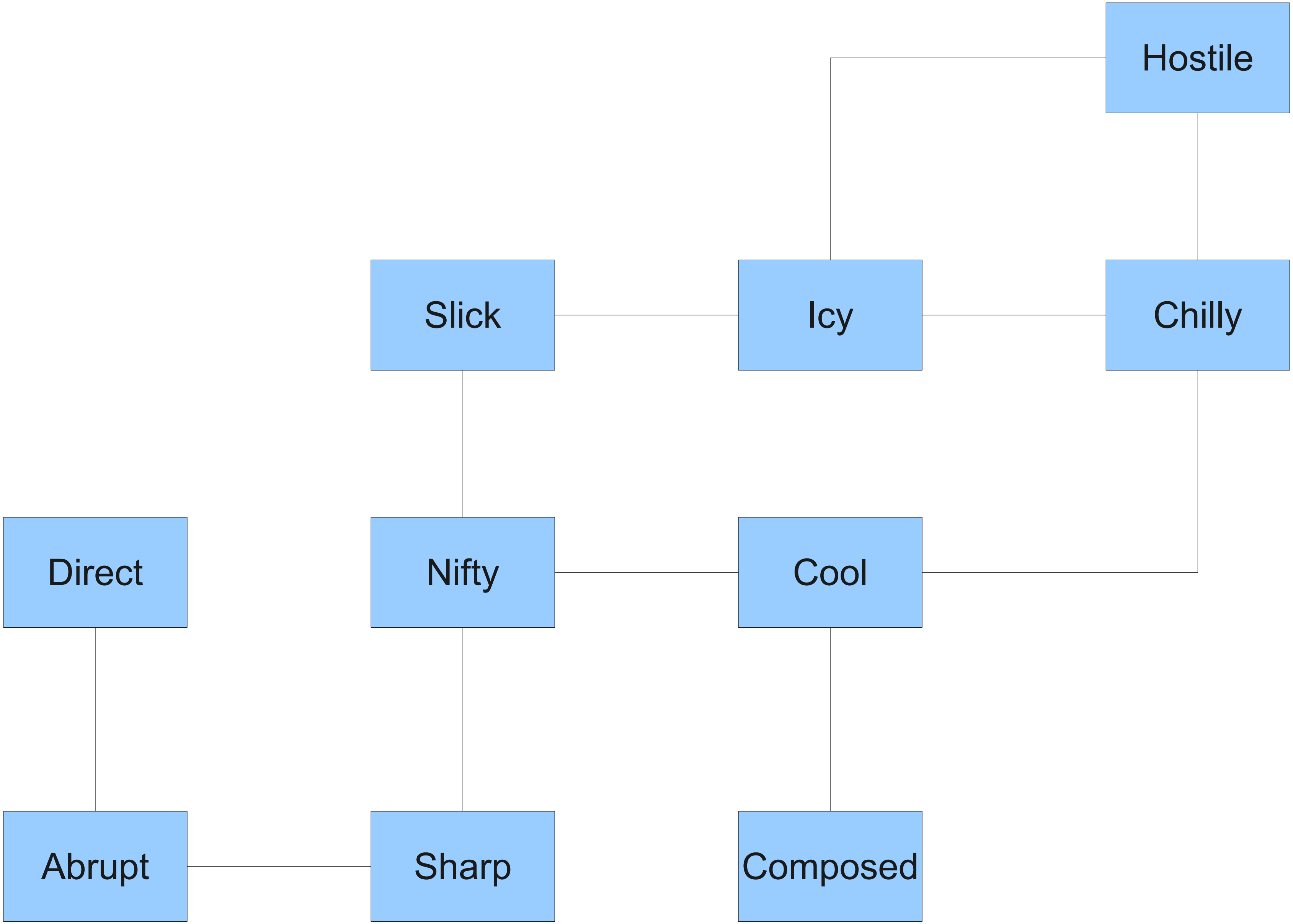
Introduction to Graphs

CS2110, Spring 2011
Cornell University

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

Each graph is a set of **nodes** connected by **edges**.

Synonym Graph



Hostile

Slick

Icy

Chilly

Direct

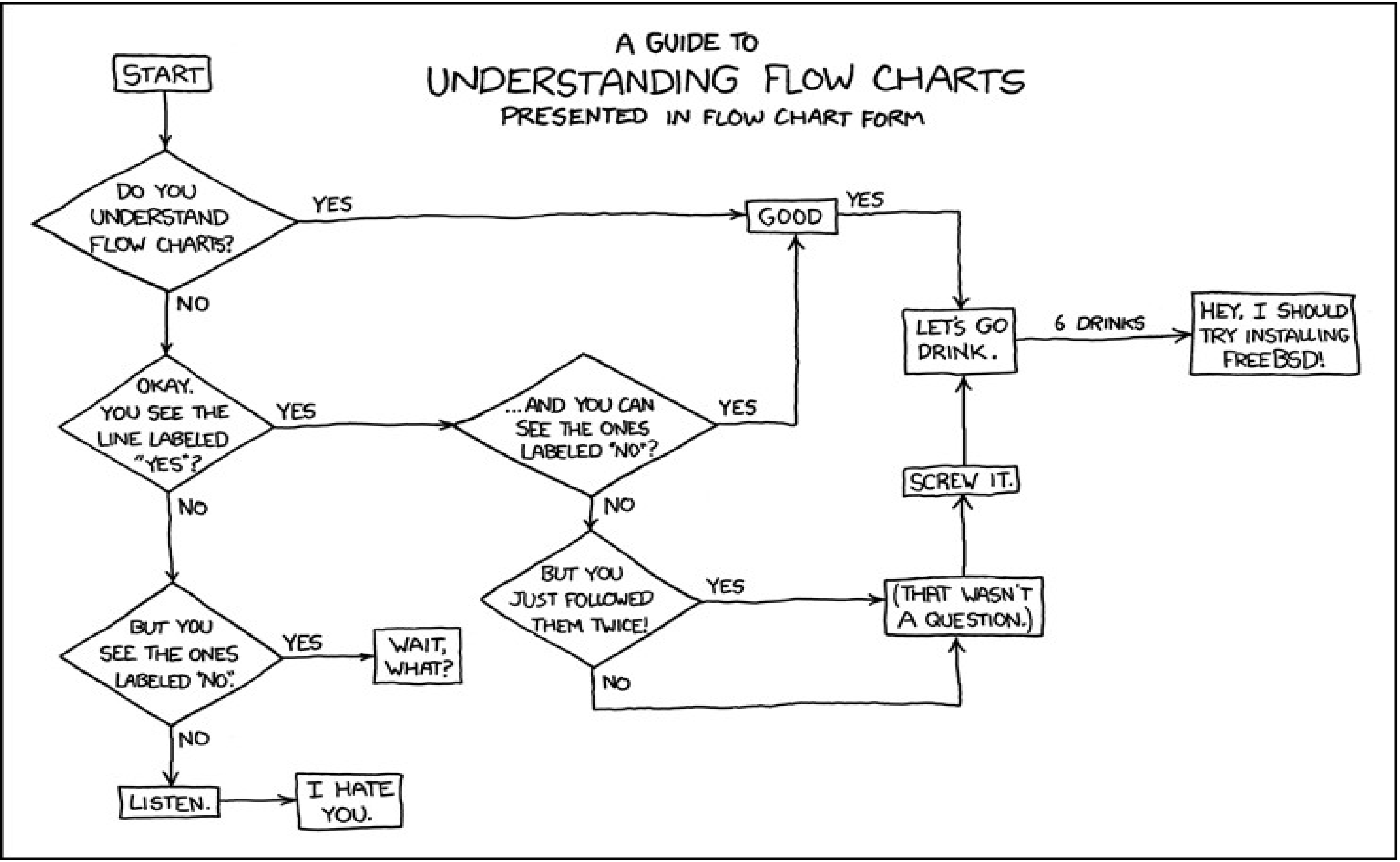
Nifty

Cool

Abrupt

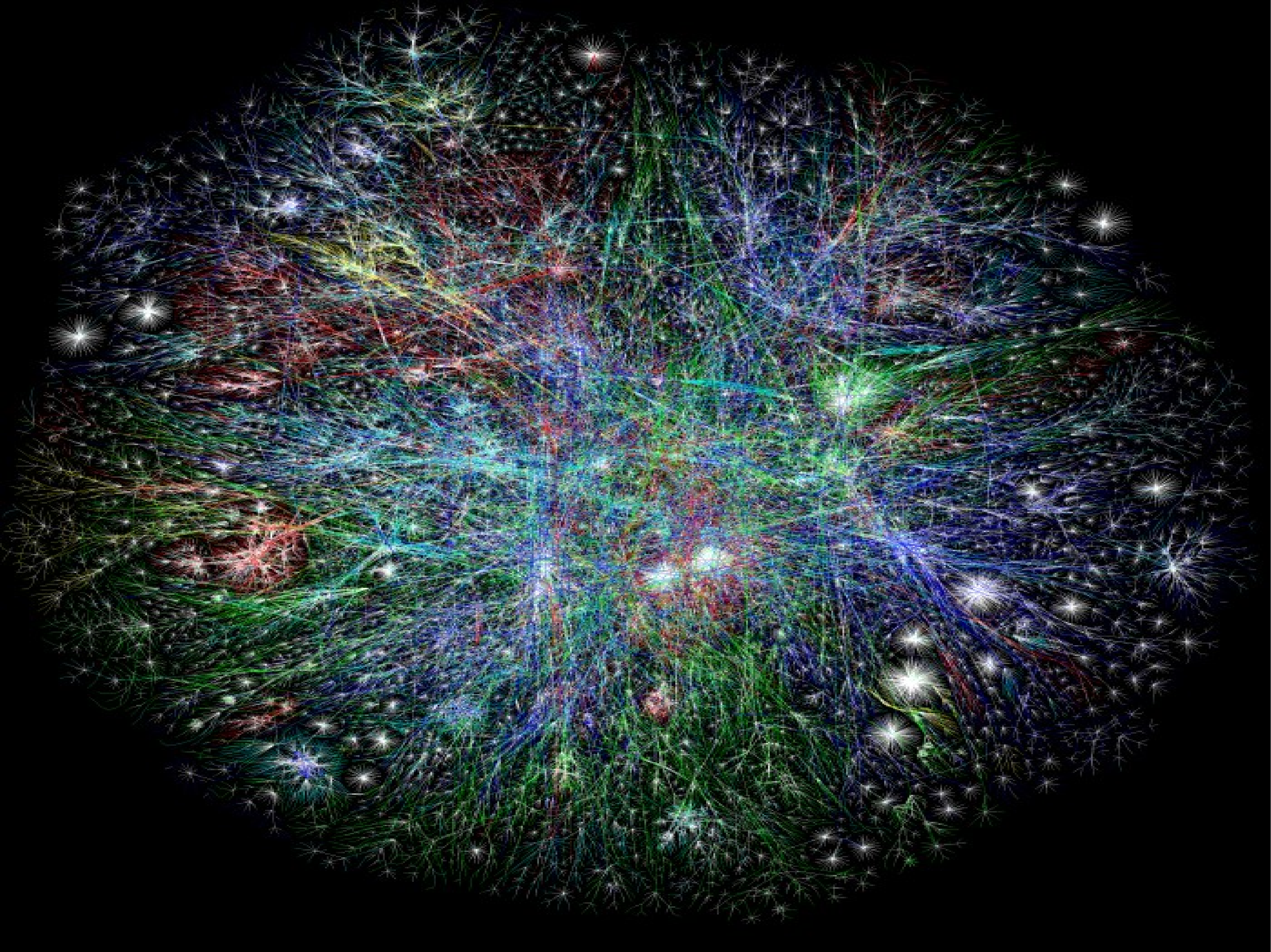
Sharp

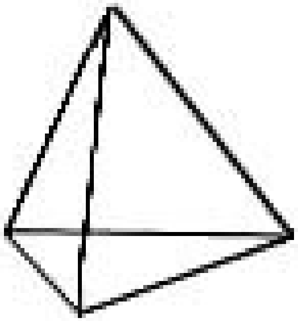
Composed

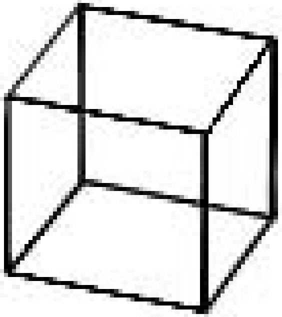

A GUIDE TO
UNDERSTANDING FLOW CHARTS
PRESENTED IN FLOW CHART FORM
START
DO YOU UNDERSTAND FLOW CHARTS?
YES
GOOD
YES
NO
LET'S GO DRINK.
6 DRINKS
HEY, I SHOULD TRY INSTALLING FREEBSD!
OKAY. YOU SEE THE LINE LABELED "YES"?
YES
...AND YOU CAN SEE THE ONES LABELED "NO"?
YES
NO
NO
SCREW IT.
BUT YOU JUST FOLLOWED THEM TWICE!
YES
(THAT WASN'T A QUESTION.)
NO
BUT YOU SEE THE ONES LABELED "NO".
YES
WAIT, WHAT?
NO
LISTEN.
I HATE YOU.

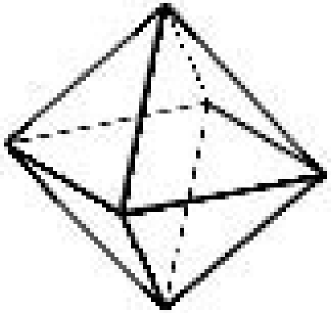

NORTH AMERICA
EUROPE
ASIA
AFRICA
SOUTH AMERICA
🟥 DESTINATIONS
🟦 AIRCRAFT ON ROUTE
VIRTUAL WORLD MAP

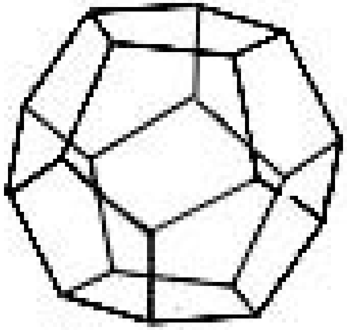
facebook®

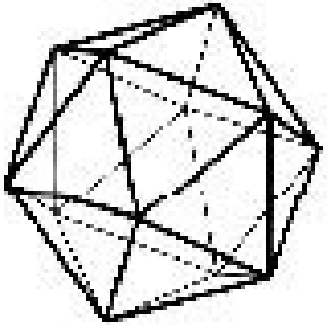



tetrahedron


cube


octahedron


dodecahedron

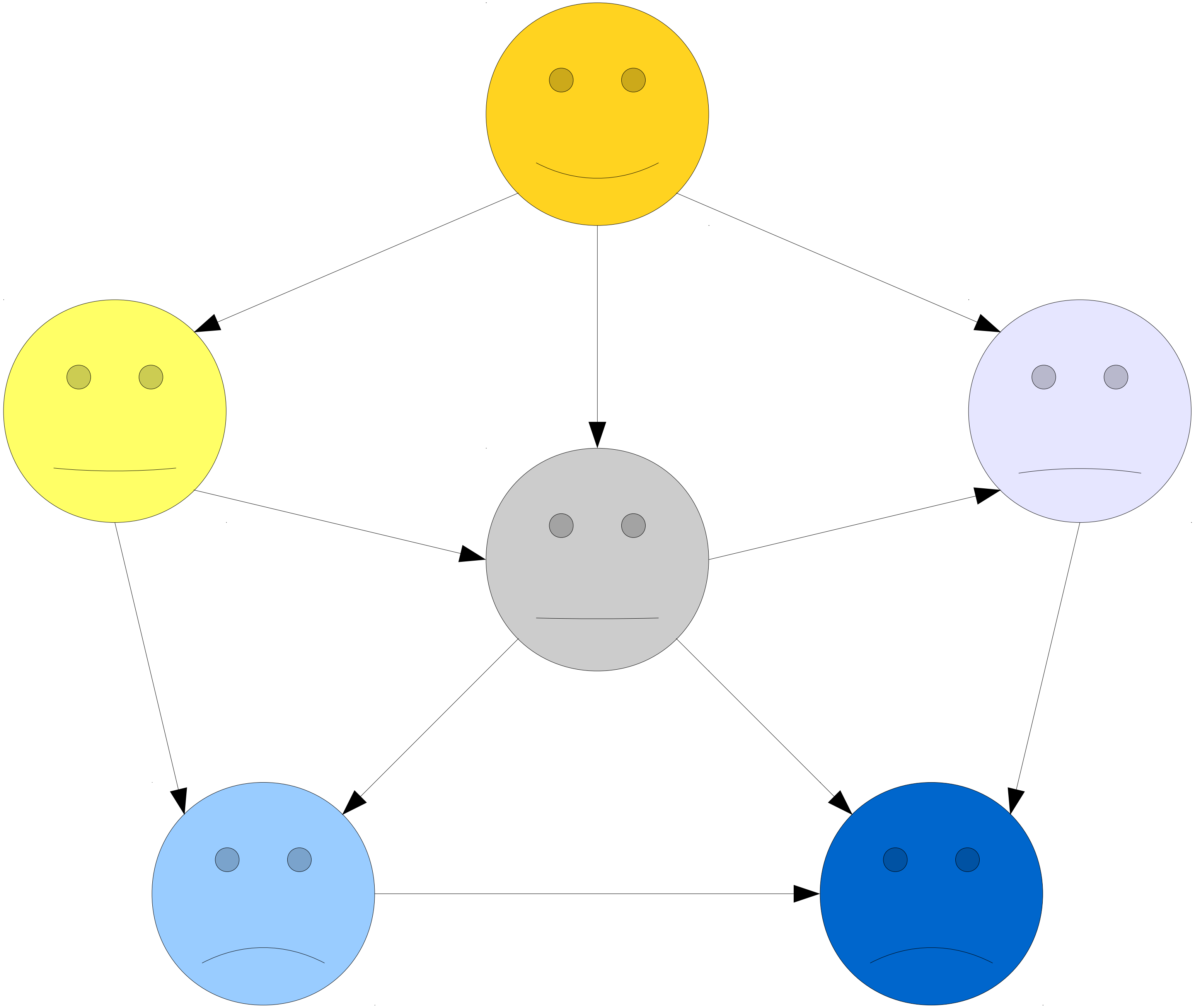

icosahedron

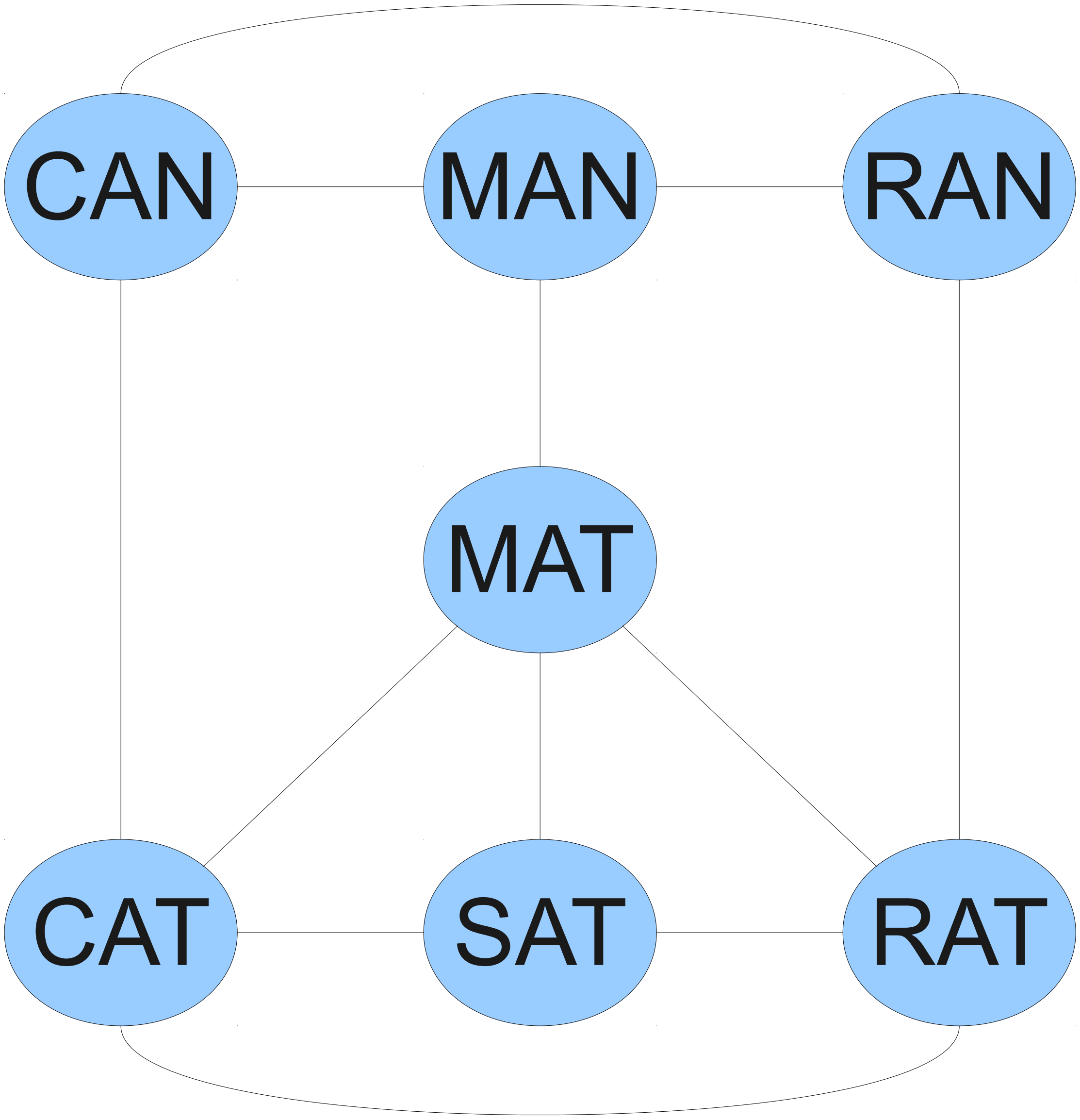
Goals for Today

- Learn the **formalisms** behind graphs.
- Learn different **representations** for graphs.
- Learn about **paths** and **cycles** in graphs.
- See three ways of **exploring** a graph.
- Explore **applications** of graphs to real-world problems.
- Explore algorithms for **drawing** graphs.

Formalisms

- A (directed) **graph** is a pair G = (V, E) where
  - V are the **vertices** (nodes) of the graph.
  - E are the **edges** (arcs) of the graph.
- Each edge is a pair (u, v) of the **start** and **end** (or **source** and **sink**) of the edge.




CAN
MAN
RAN
MAT
CAT
SAT
RAT

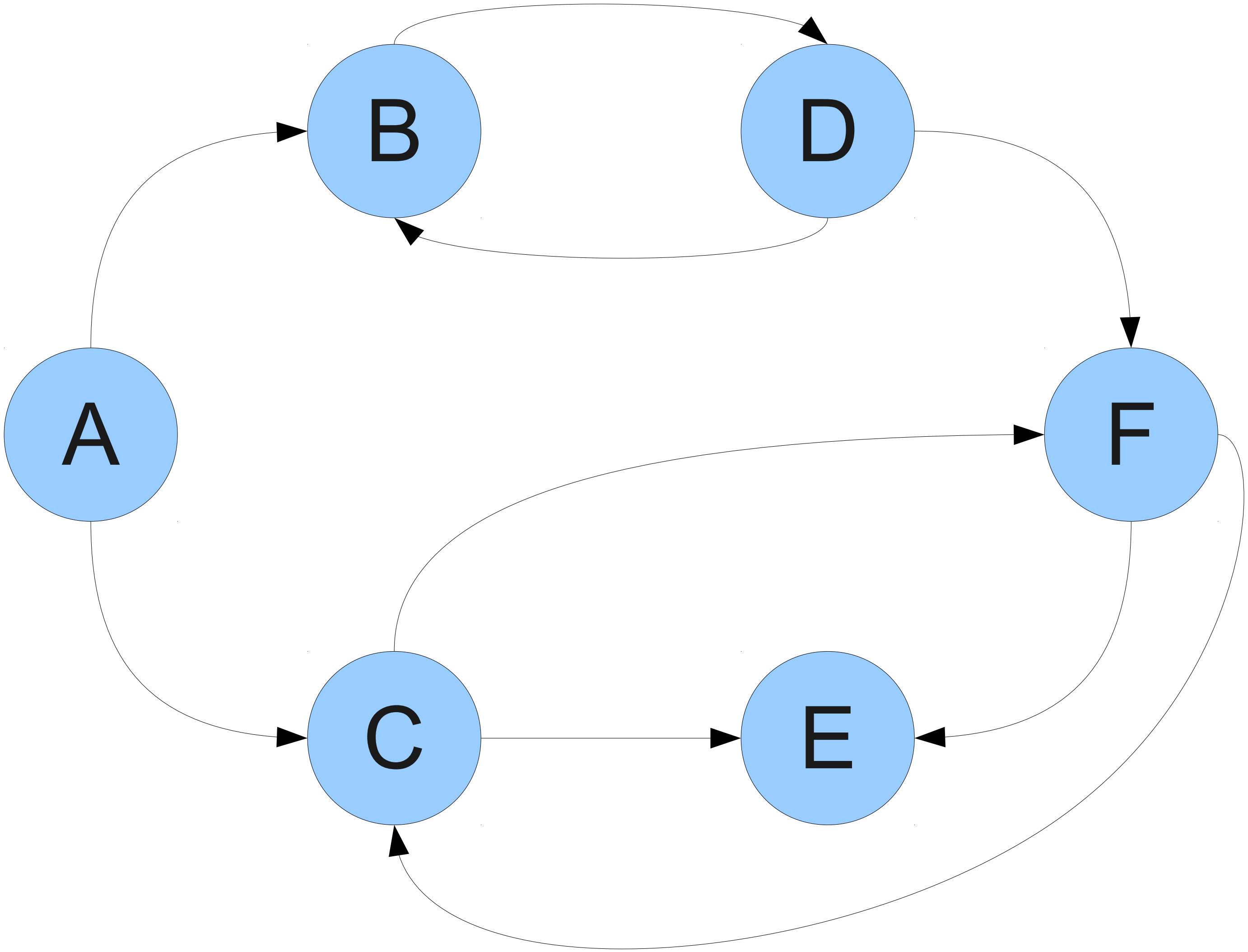
Directed and Undirected Graphs

- A graph is **directed** if its edges specify which is the start and end node.
  - Encodes asymmetric relationship.
- A graph is **undirected** if the edges don't distinguish between the start and end nodes.
  - Encodes symmetric relationship.
- An undirected graph is a **special case** of a directed graph (just add edges both ways).

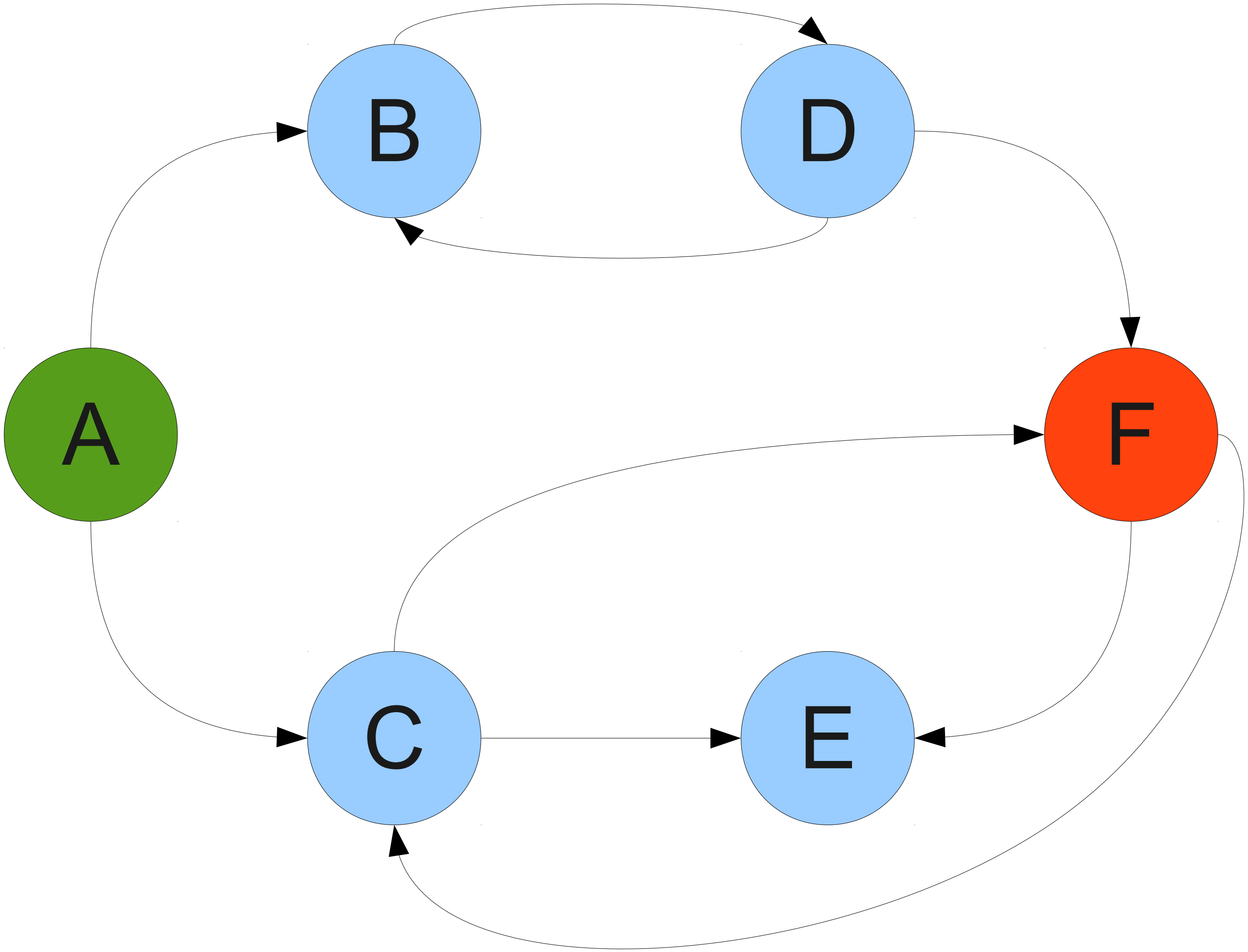
How Big is a Graph G = (V, E)?

- Two measures:
    - Number of vertices: |V| (often denoted *n*)
    - Number of edges: |E| (often denoted *m*)
- |E| can be at most $O(|V|^2)$
- A graph is called **sparse** if it has few edges. A graph with many edges is called **dense**.

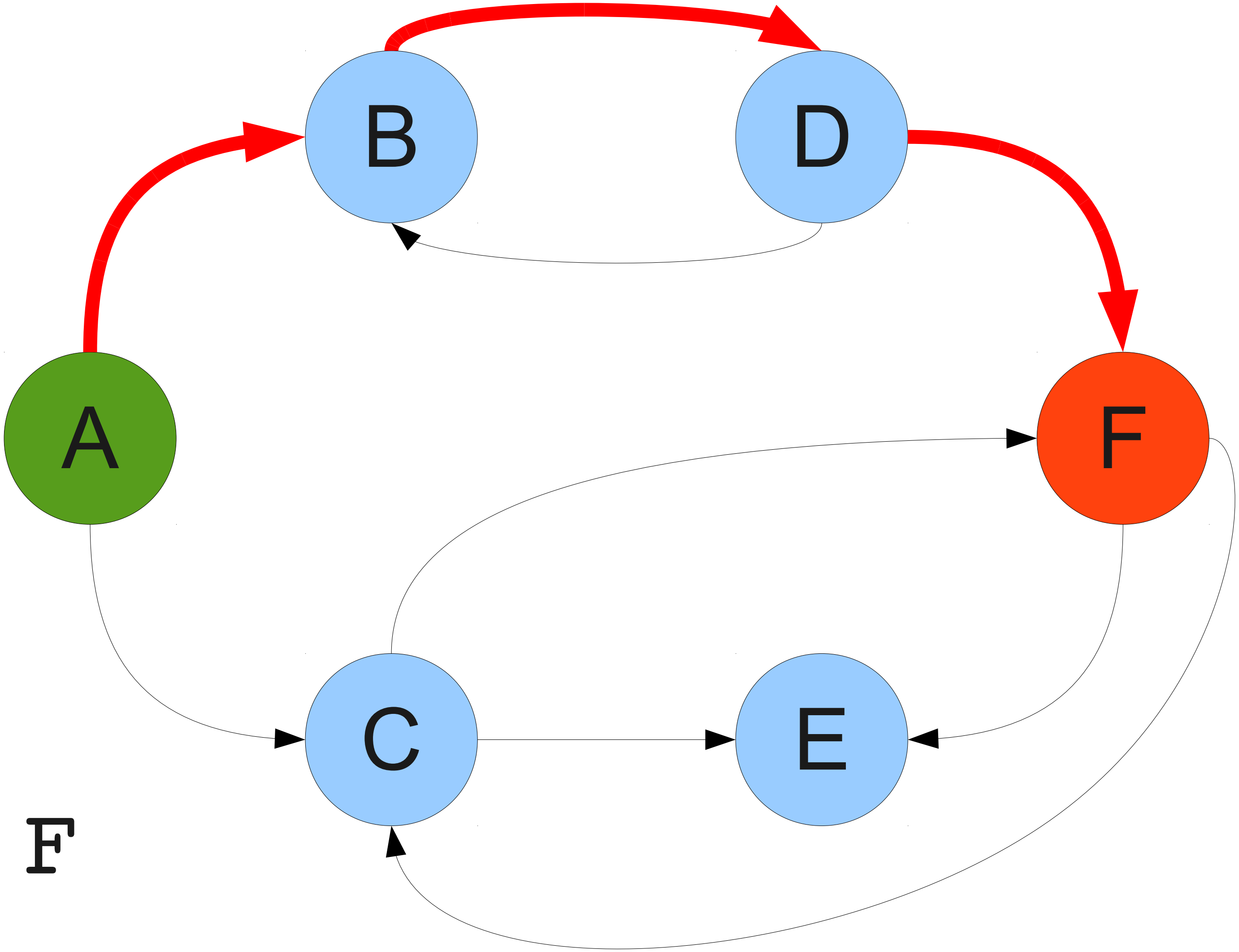
Navigating a Graph



Navigating a Graph

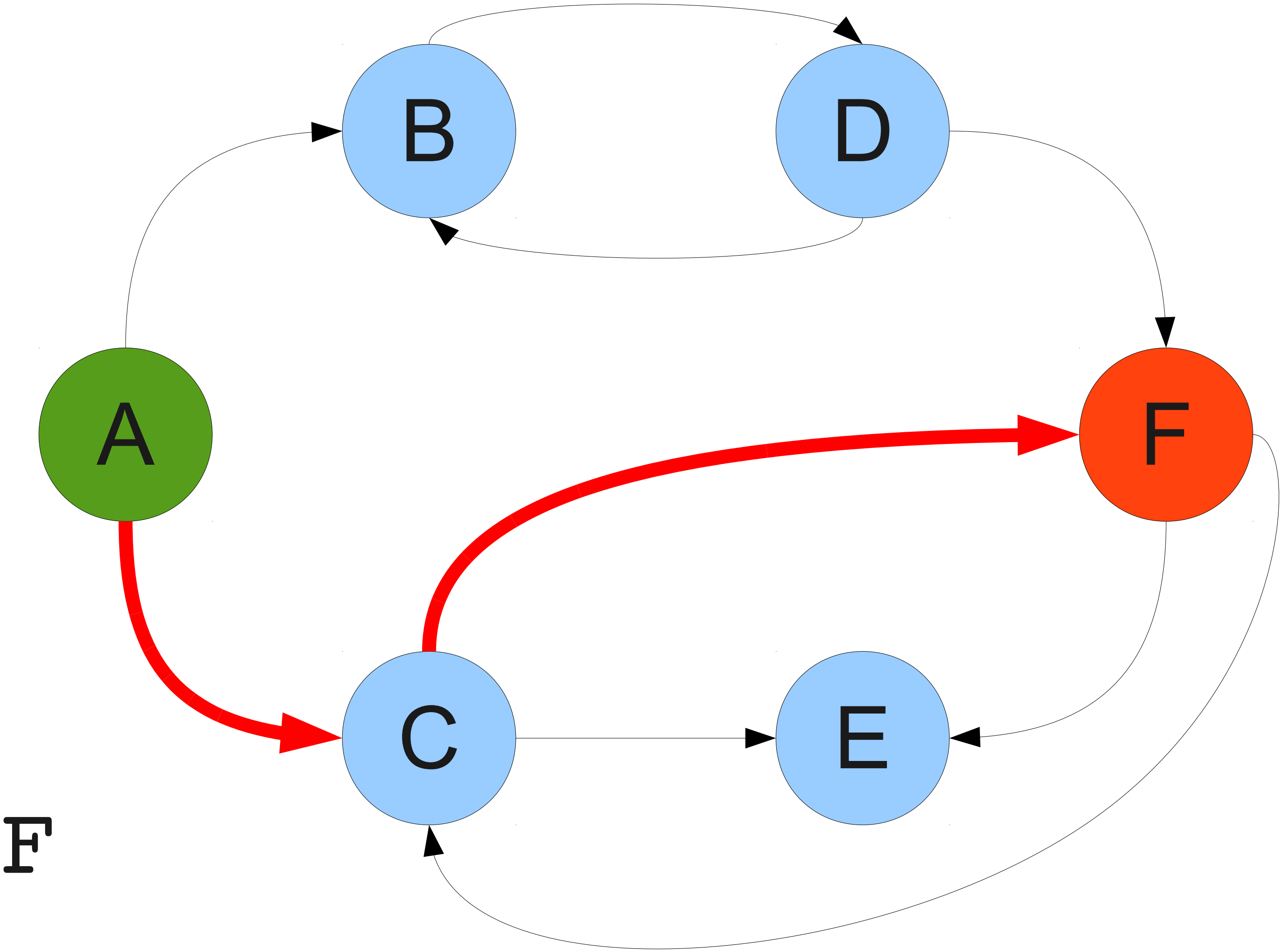


Navigating a Graph



`A B D F`

Navigating a Graph

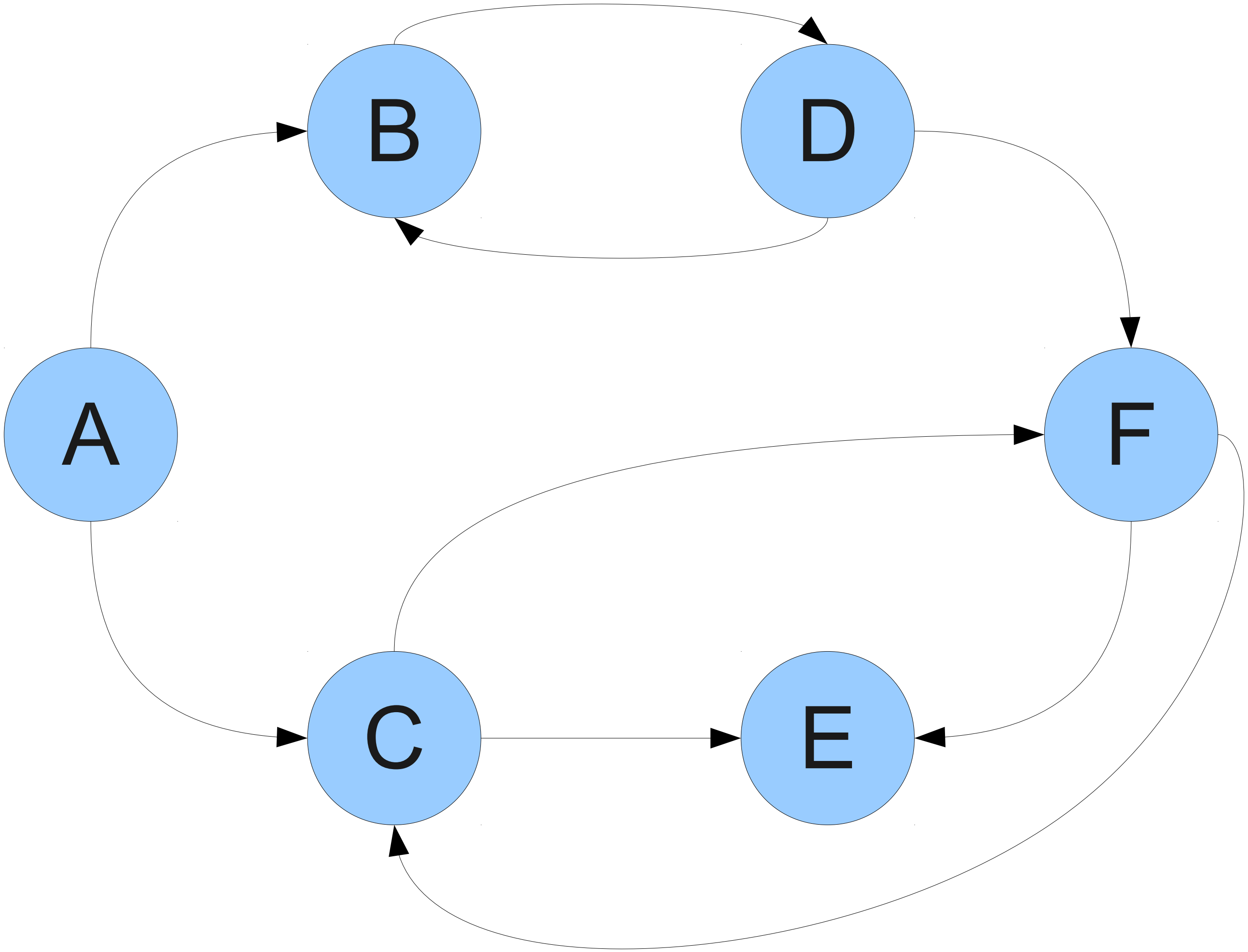


`A C F`

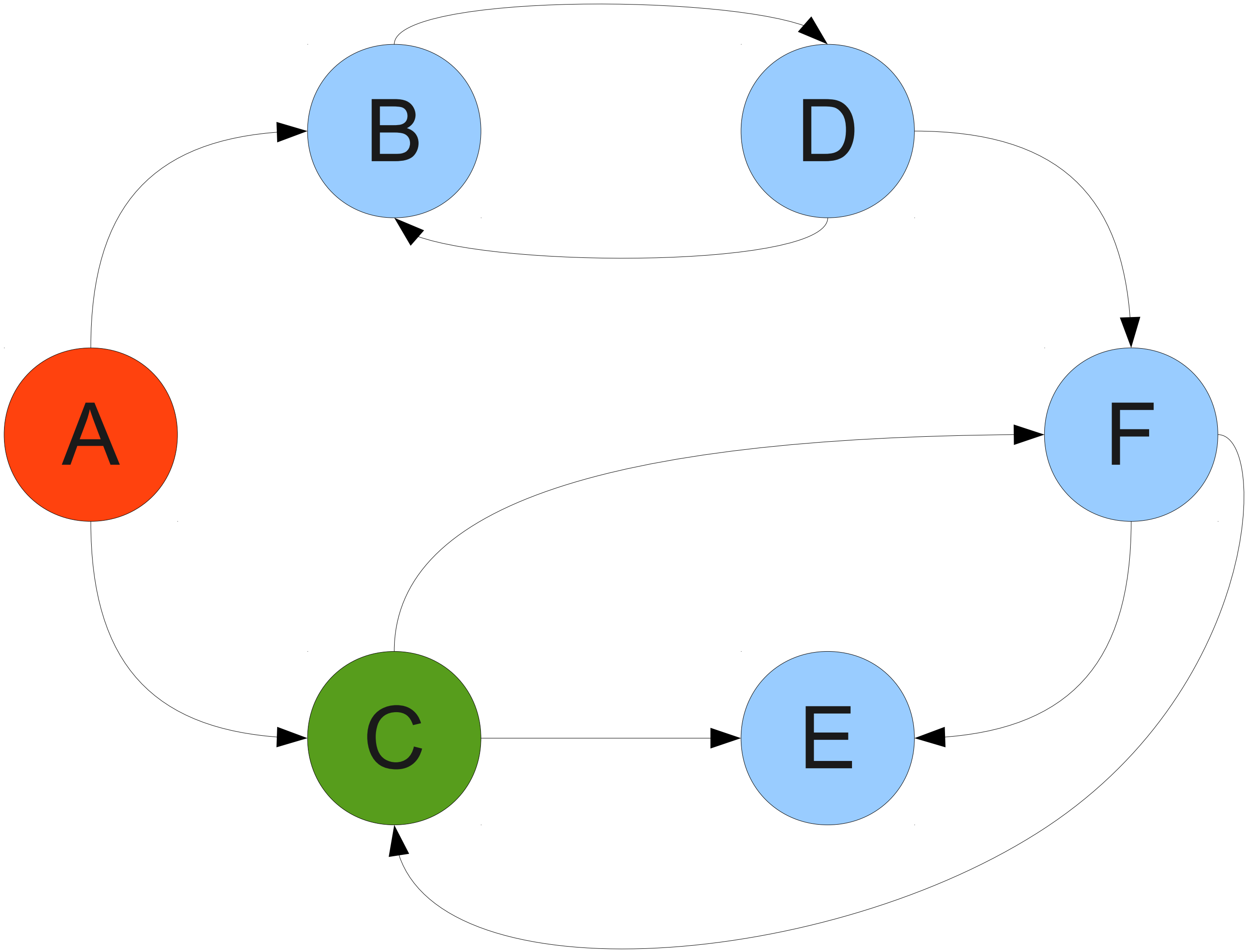
A **path** from v_0 to v_n is a list of edges
$(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)$.

The **length** of a path is the number of edges it contains.

Navigating a Graph

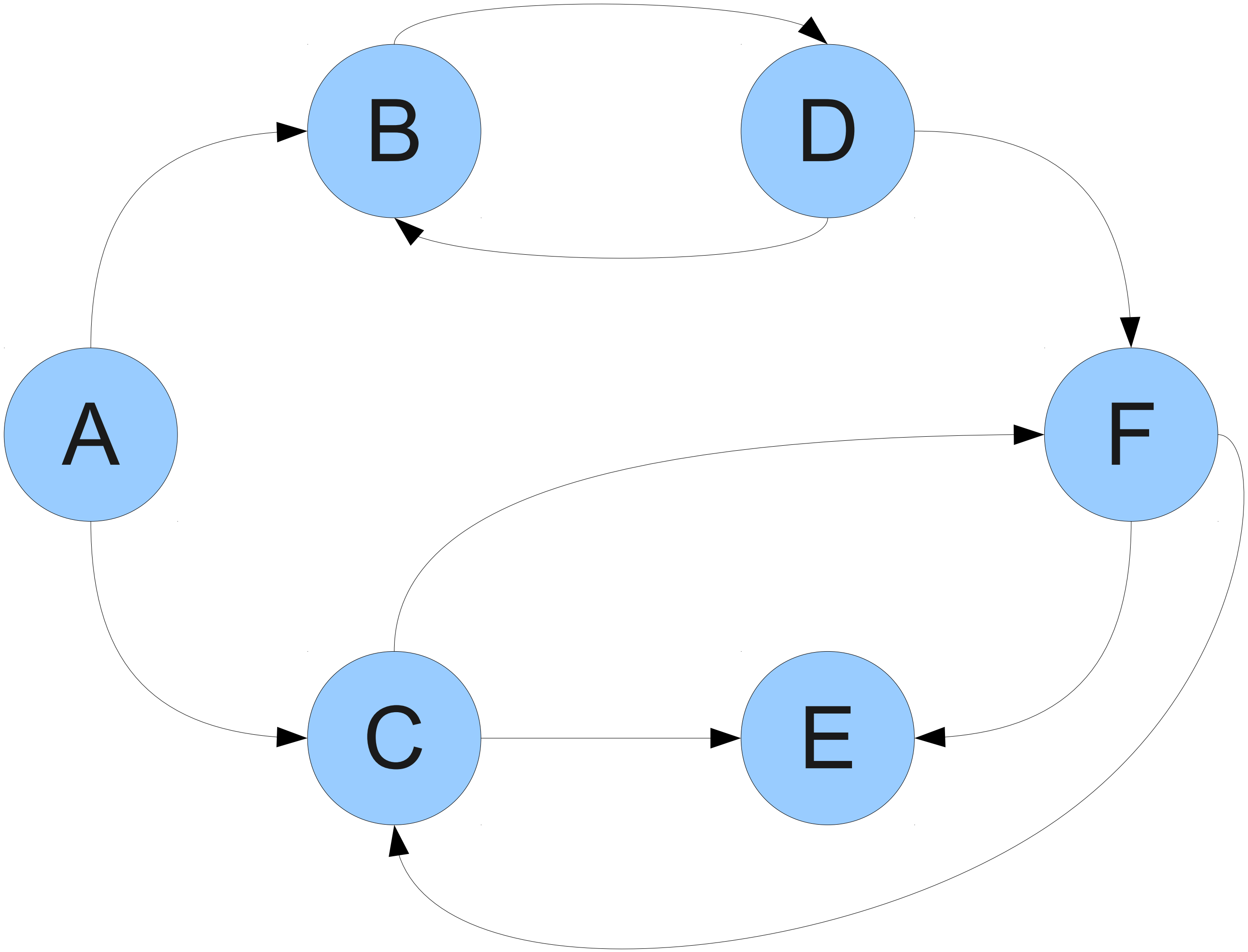


Navigating a Graph

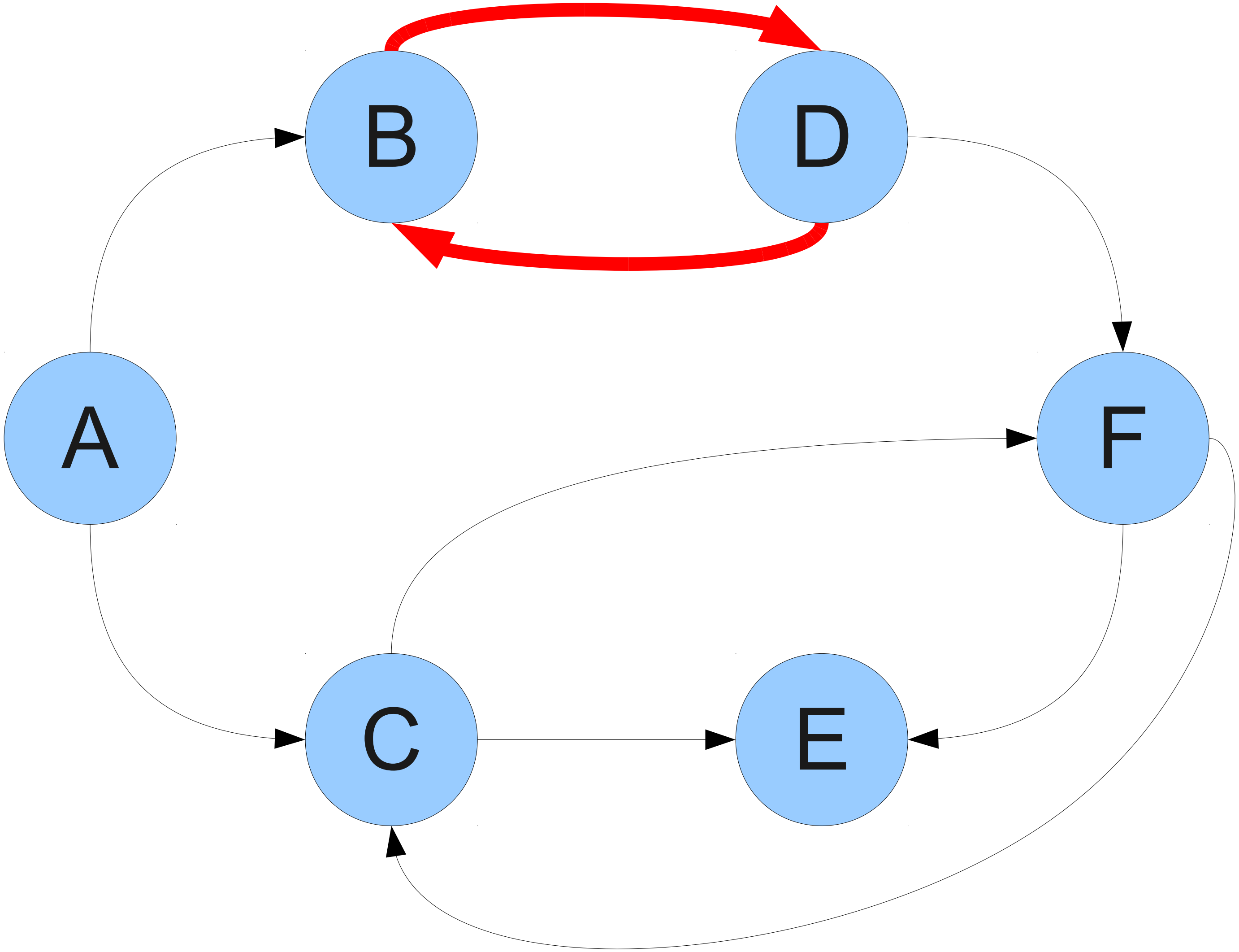


A node v is **reachable** from node u if there is a path from u to v.

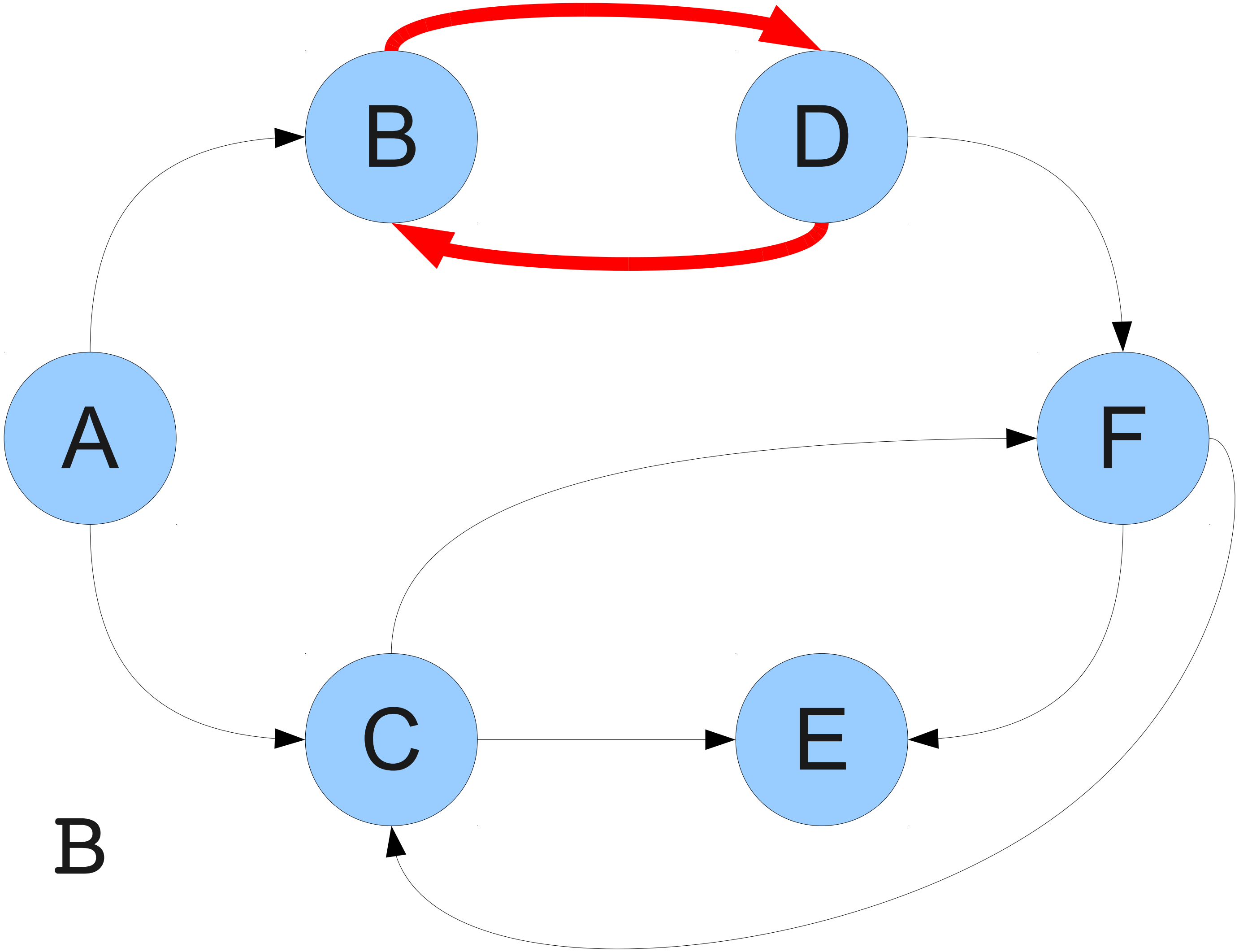
Navigating a Graph



Navigating a Graph

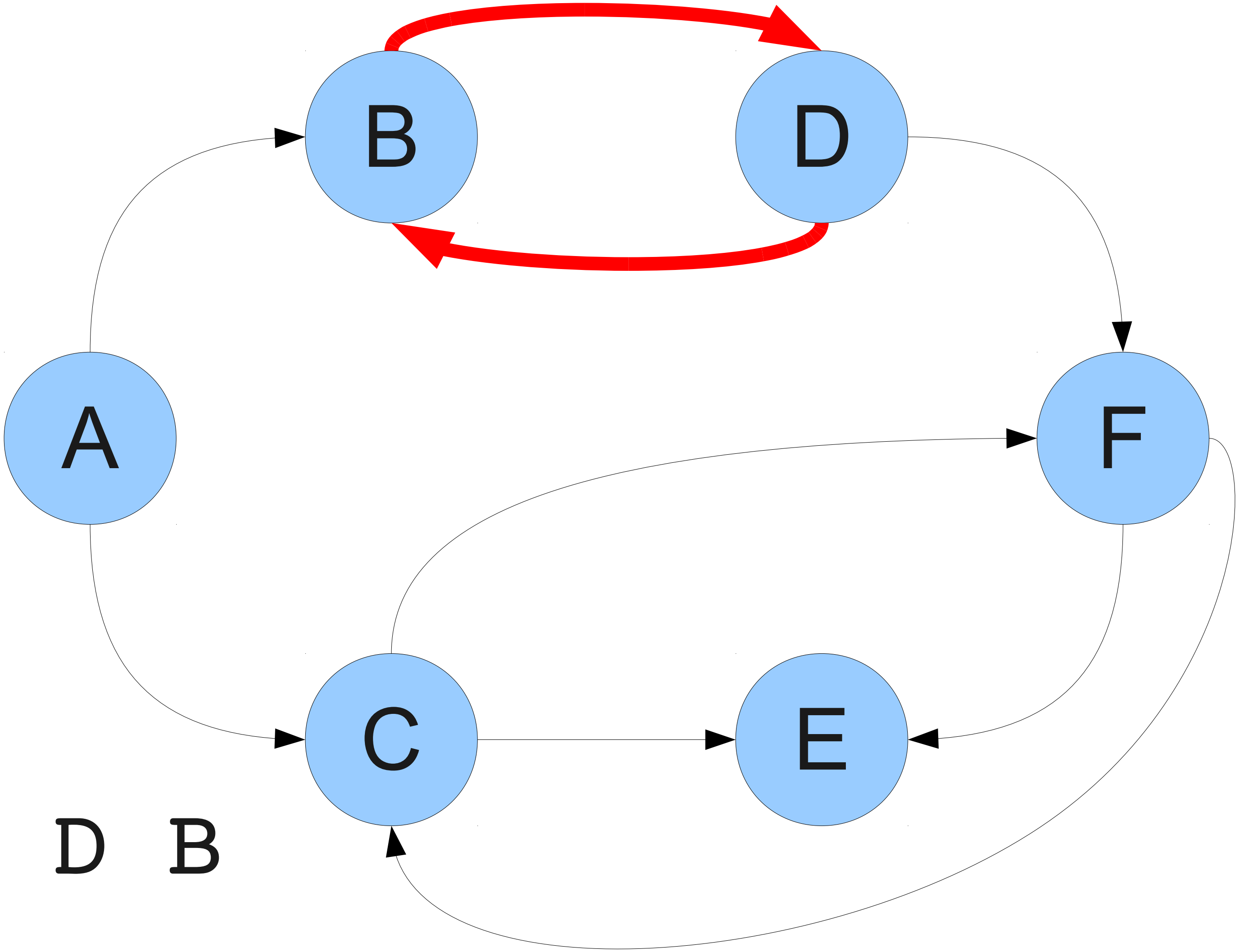


Navigating a Graph



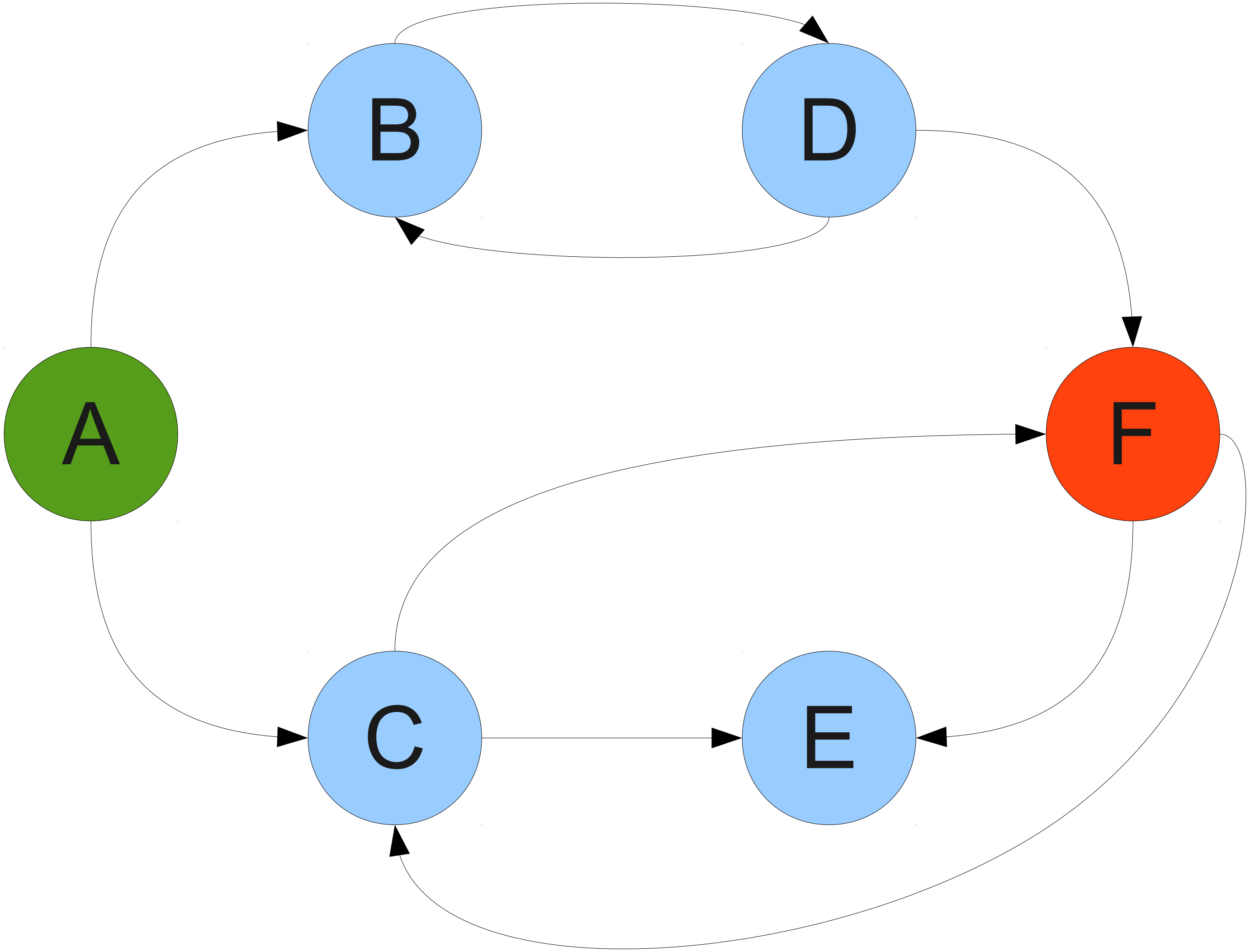
B D B

Navigating a Graph

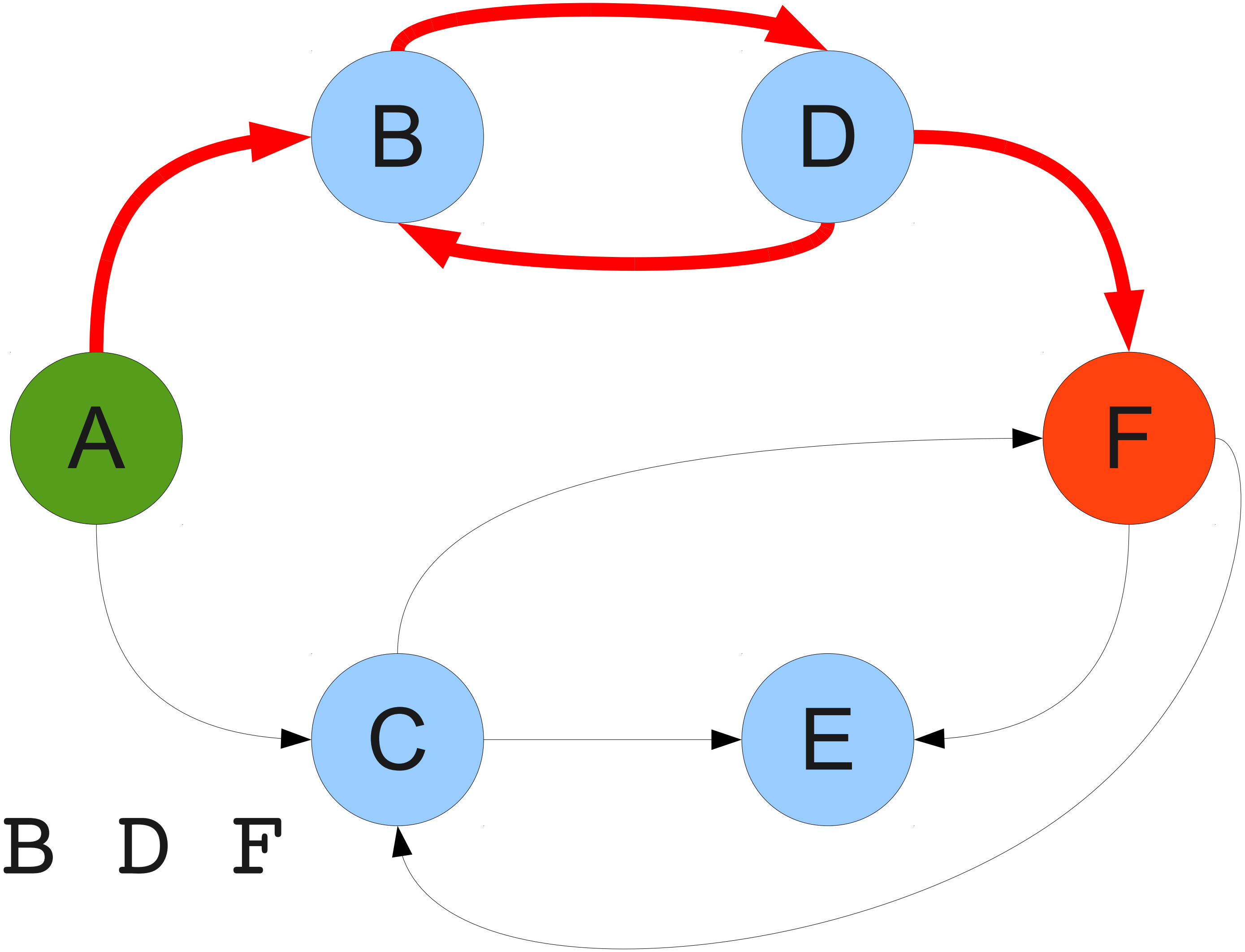


B D B D B

Navigating a Graph



Navigating a Graph



A B D B D F

A **cycle** in a graph is a set of edges

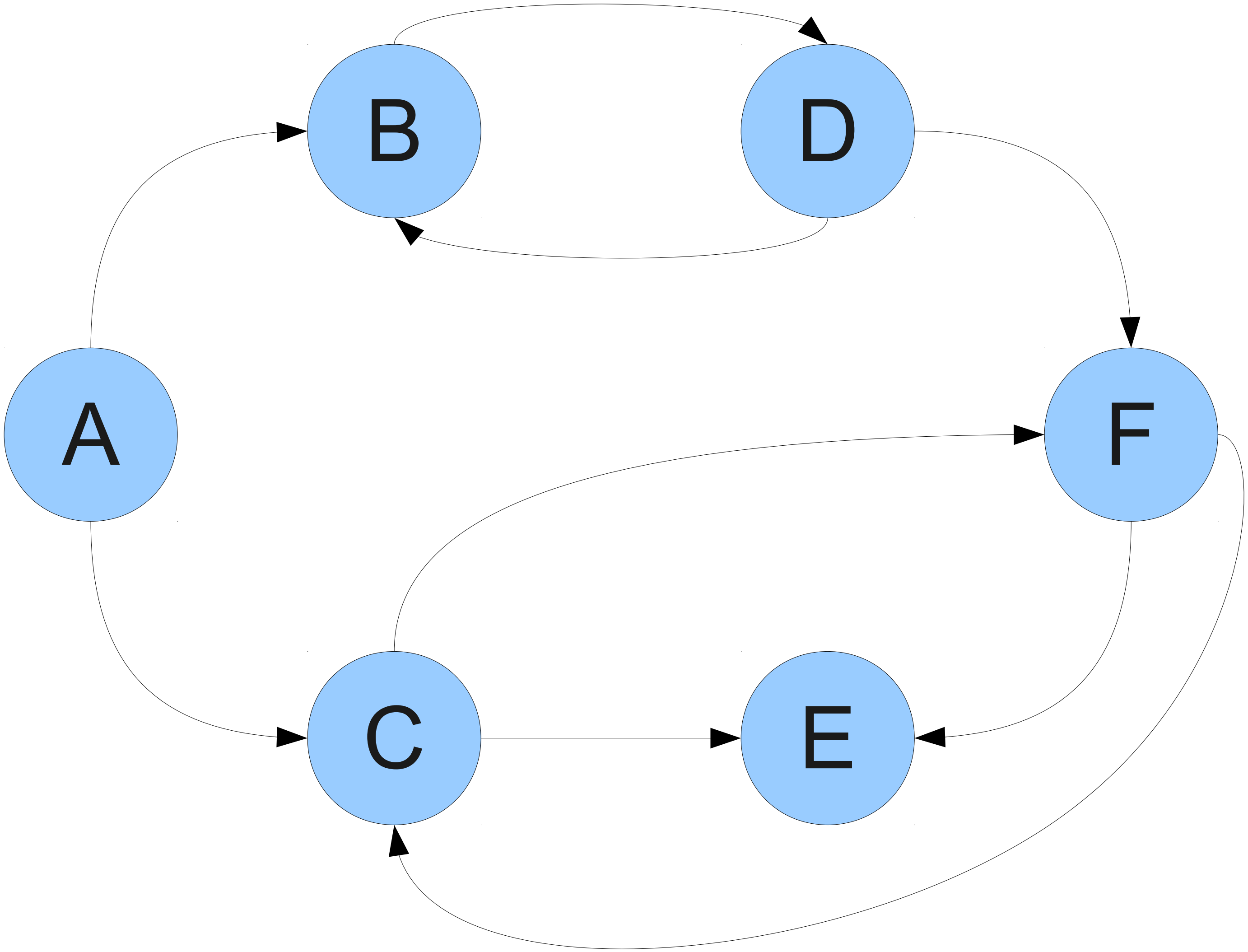
$(v_0, v_1), (v_1, v_2), \ldots, (v_n, v_0)$

that starts and ends at the same node.

A **simple path** is a path that does not contain a cycle.

A **simple cycle** is a cycle that does not contain a smaller cycle

Properties of Nodes



The **indegree** of a node is the number of edges entering that node.

The **outdegree** of a node is the number of edges leaving that node.

In an undirected graph, these are the same and are called the **degree** of the node.

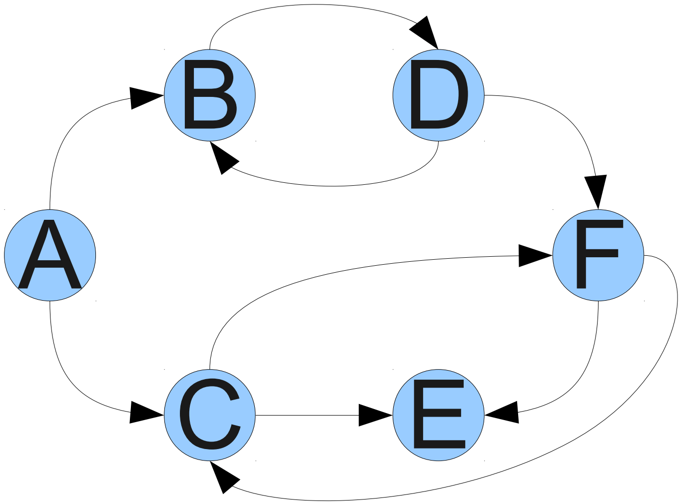
Summary of Terminology

- A **path** is a series of edges connecting two nodes.
    - The **length** of a path is the number of edges in the path.
    - A node v is **reachable** from u if there is a path from u to v.
- A **cycle** is a path from a node to itself.
- A **simple path** is a path without a cycle.
- A **simple cycle** is a cycle that does not contain a nested cycle.
- The **indegree** and **outdegree** of a node are the number of edges entering/leaving it.

Representing Graphs

Adjacency Matrices

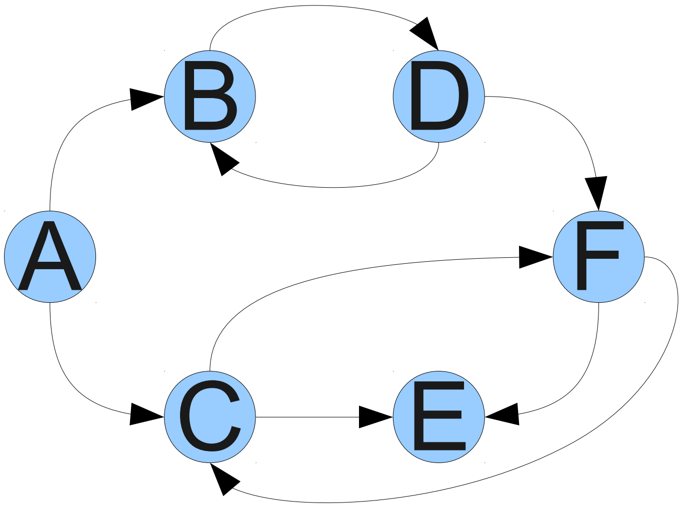
- n x n grid of boolean values.
- Element A_{ij} is 1 if edge from i to j, 0 else.
- Memory usage is $O(n^2)$
- Can check if an edge exists in O(1).
- Can find all edges entering or leaving a node in O(n).

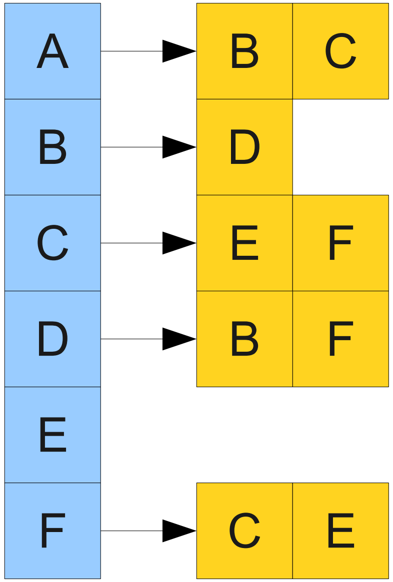


	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	0	1	0	0
C	0	0	0	0	1	1
D	0	1	0	0	0	1
E	0	0	0	0	0	0
F	0	0	1	0	1	0

Adjacency Lists

- List of edges leaving each node.
- Memory usage is O(m+n)
- Find edges leaving a node in $O(d^+(u))$
- Check if edge exists in $O(d^+(u))$

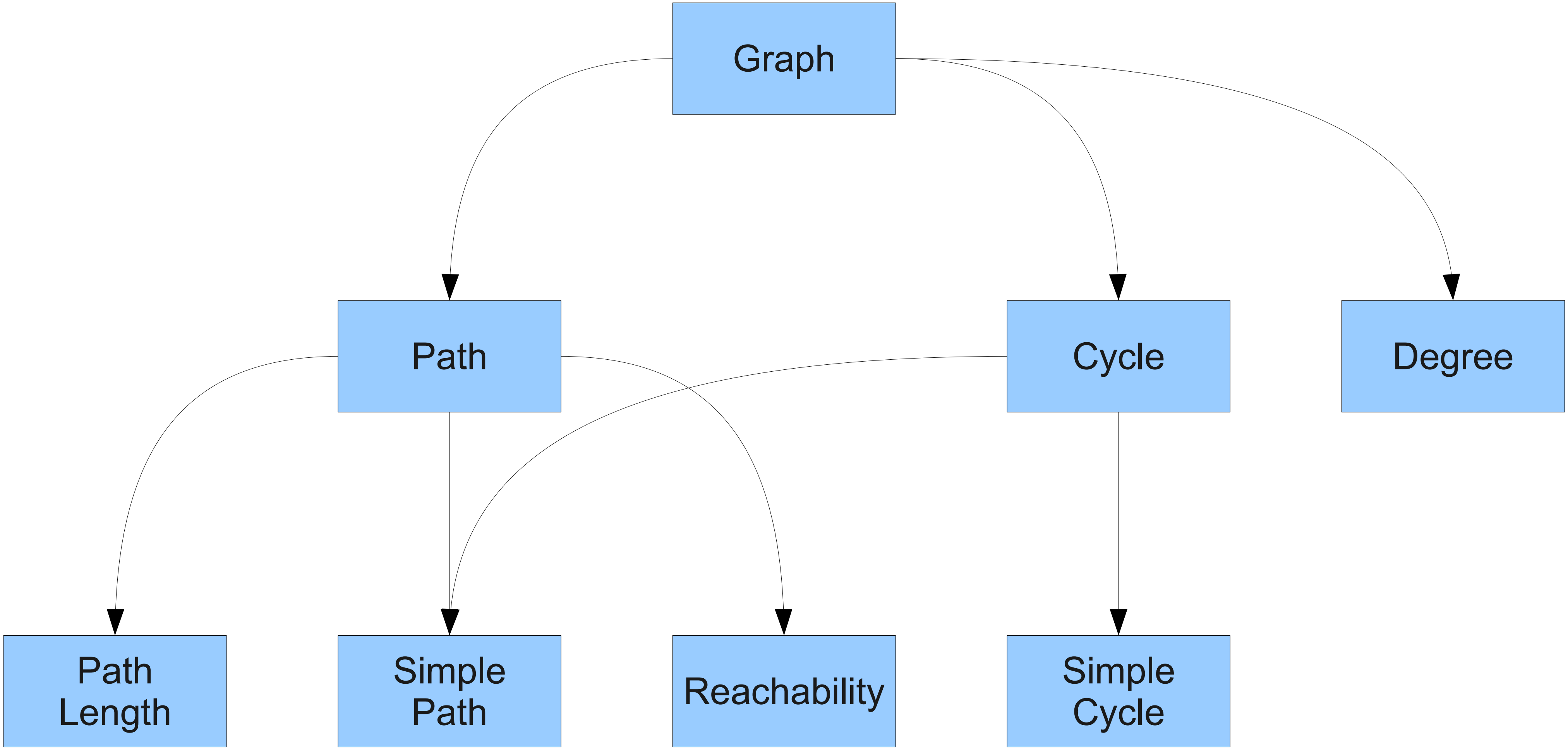




A → B C
B → D
C → E F
D → B F
E
F → C E

Graph Algorithms

Representing Prerequisites



Graph

Path

Cycle

Degree

Path Length

Simple Path

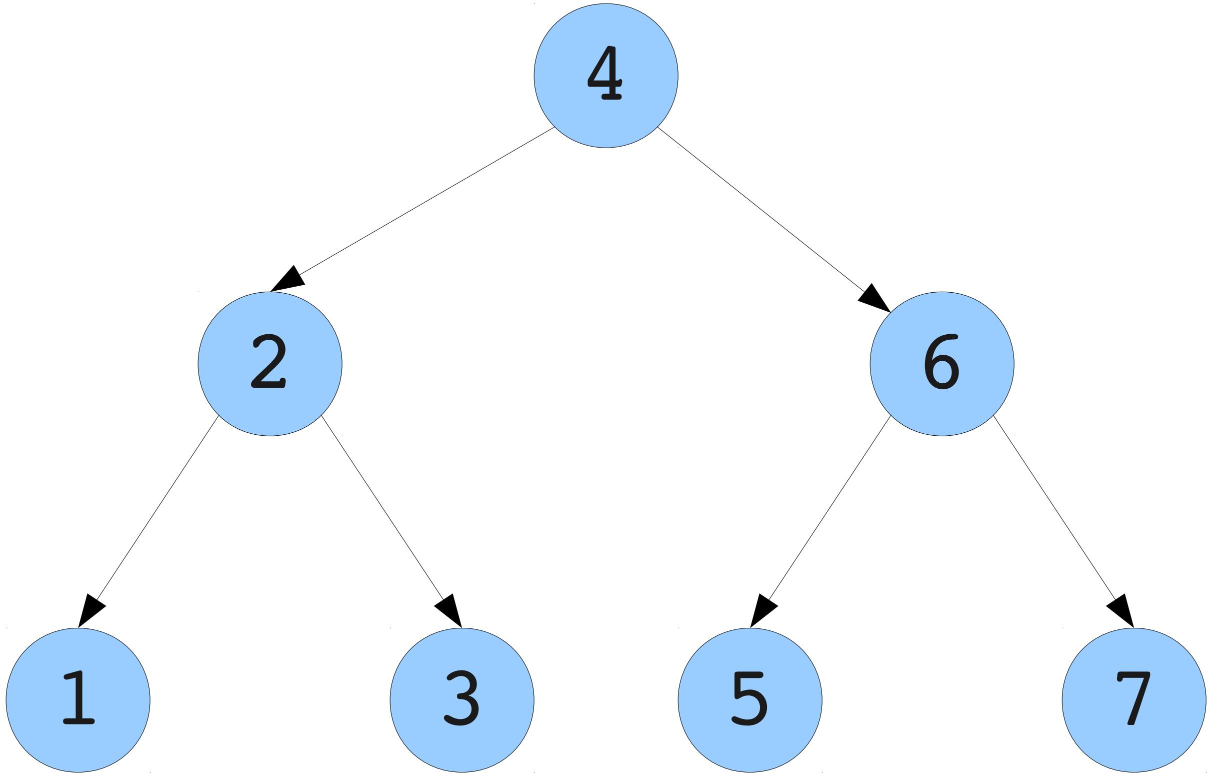
Reachability

Simple Cycle

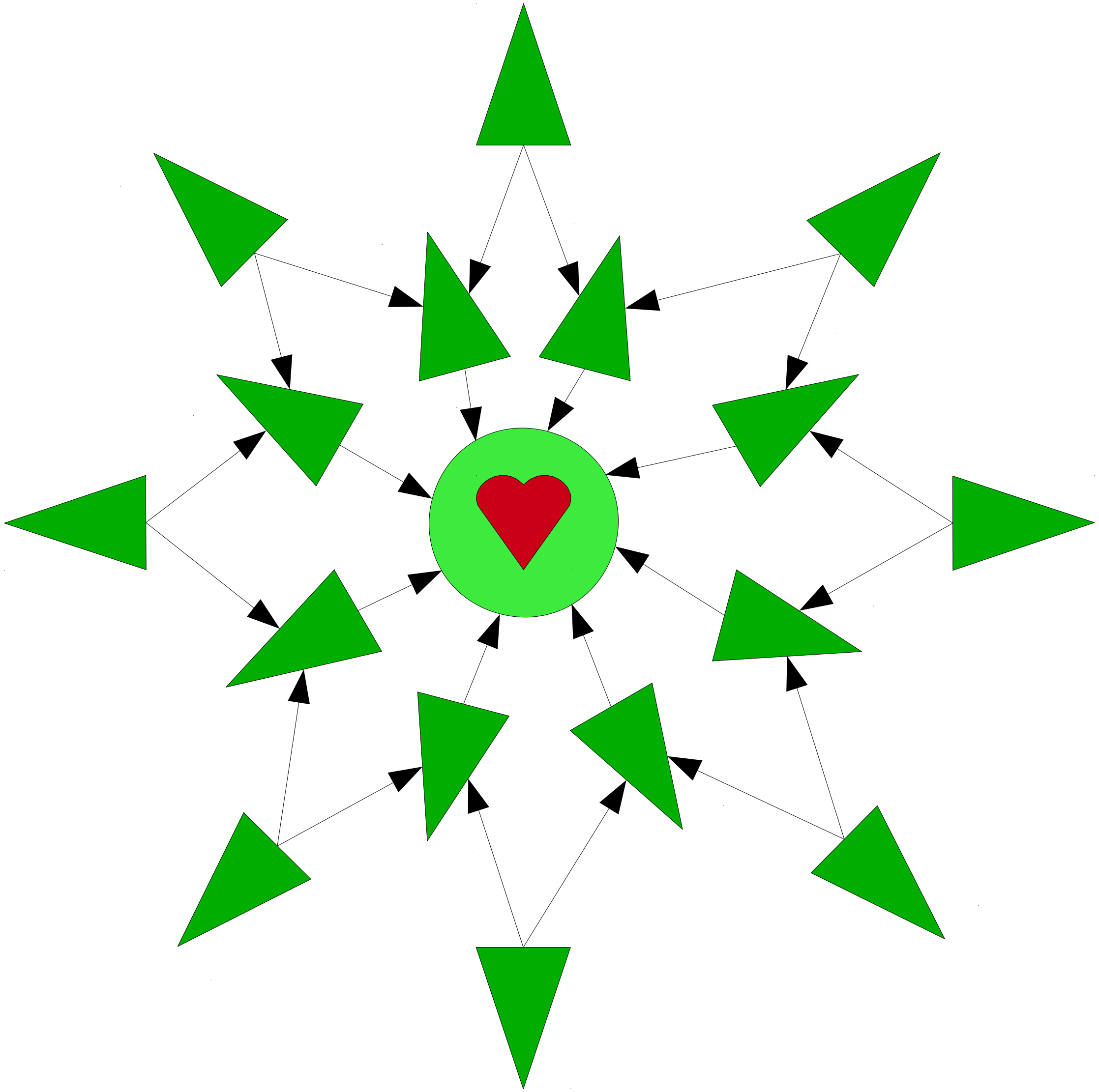
A **directed acyclic graph** (DAG) is a directed graph with no cycles.

Examples of DAGs

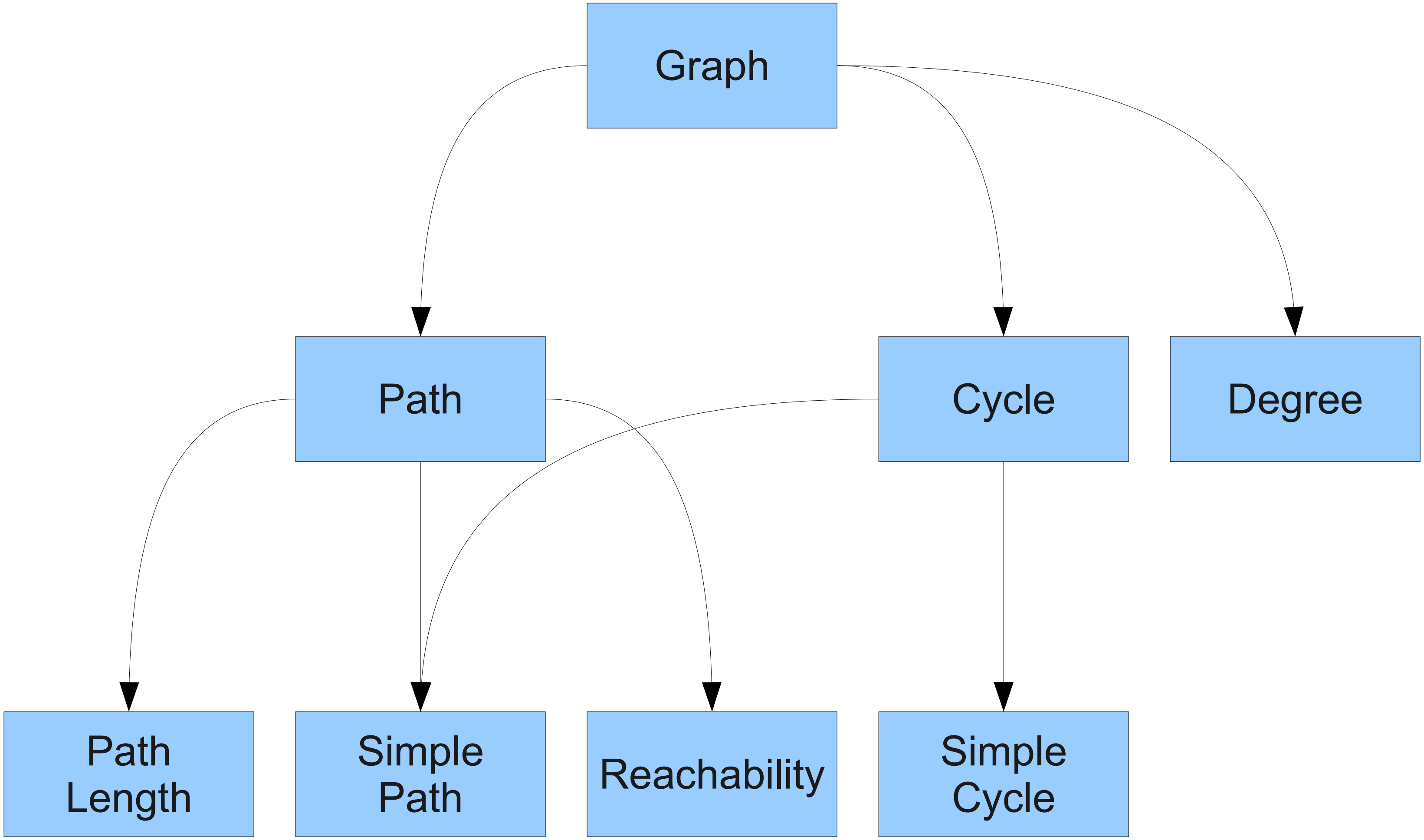
Examples of DAGs



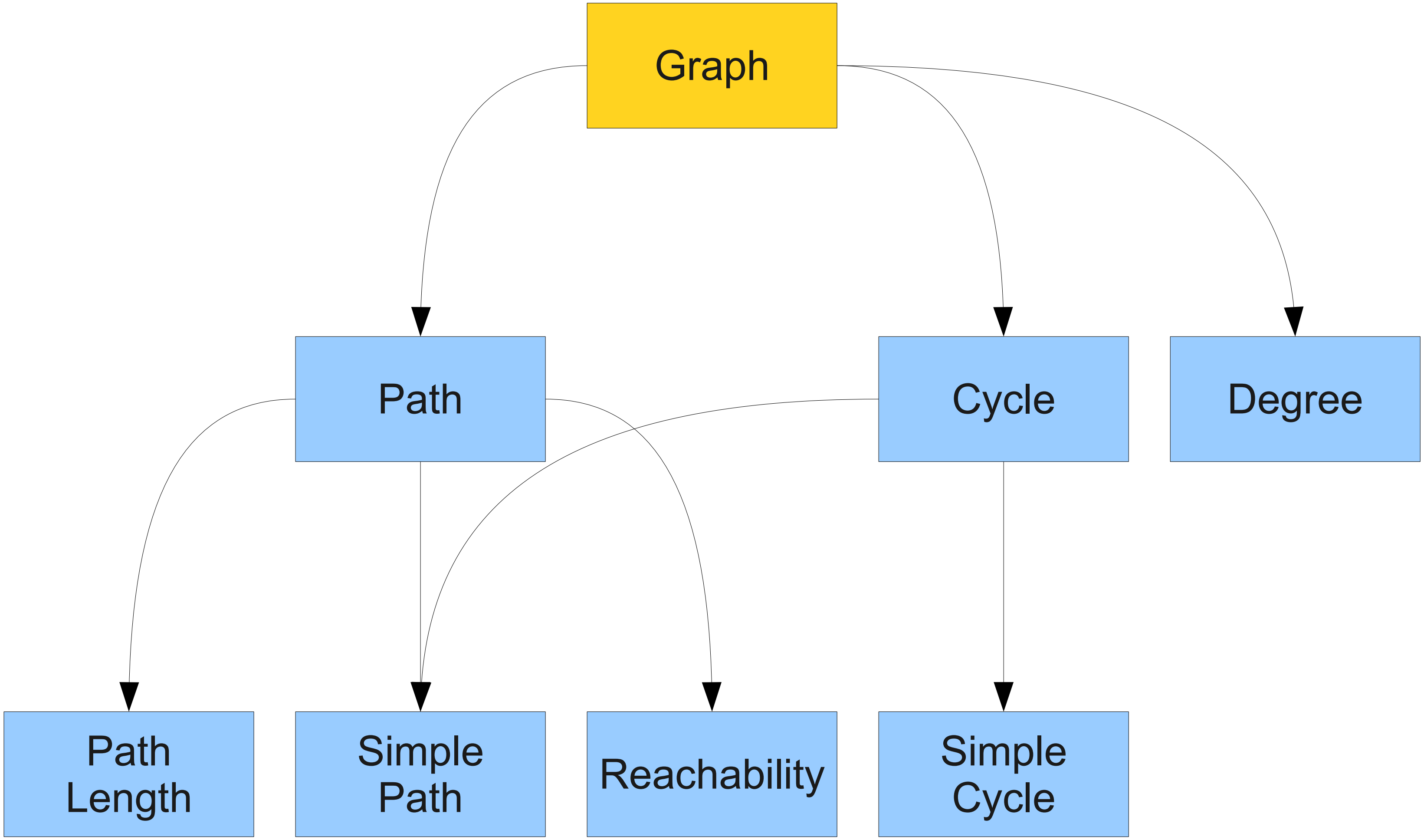
Examples of DAGs



Traversing a DAG



Traversing a DAG



Graph

Path

Cycle

Degree

Path Length

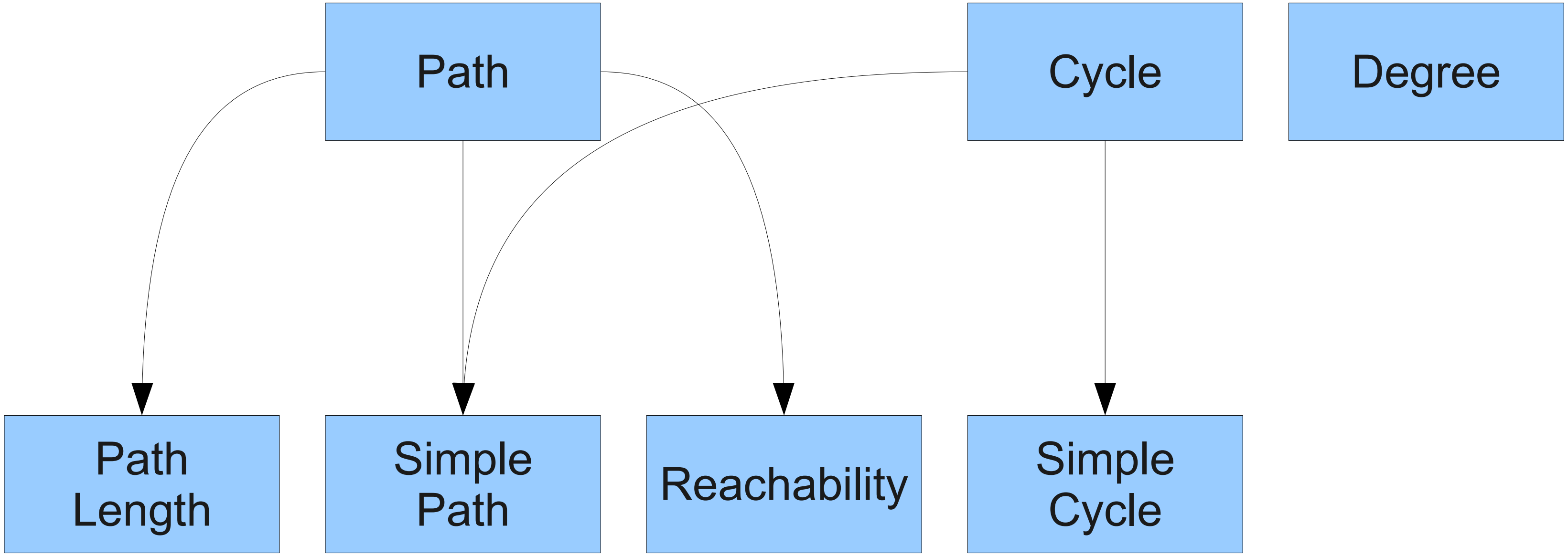
Simple Path

Reachability

Simple Cycle

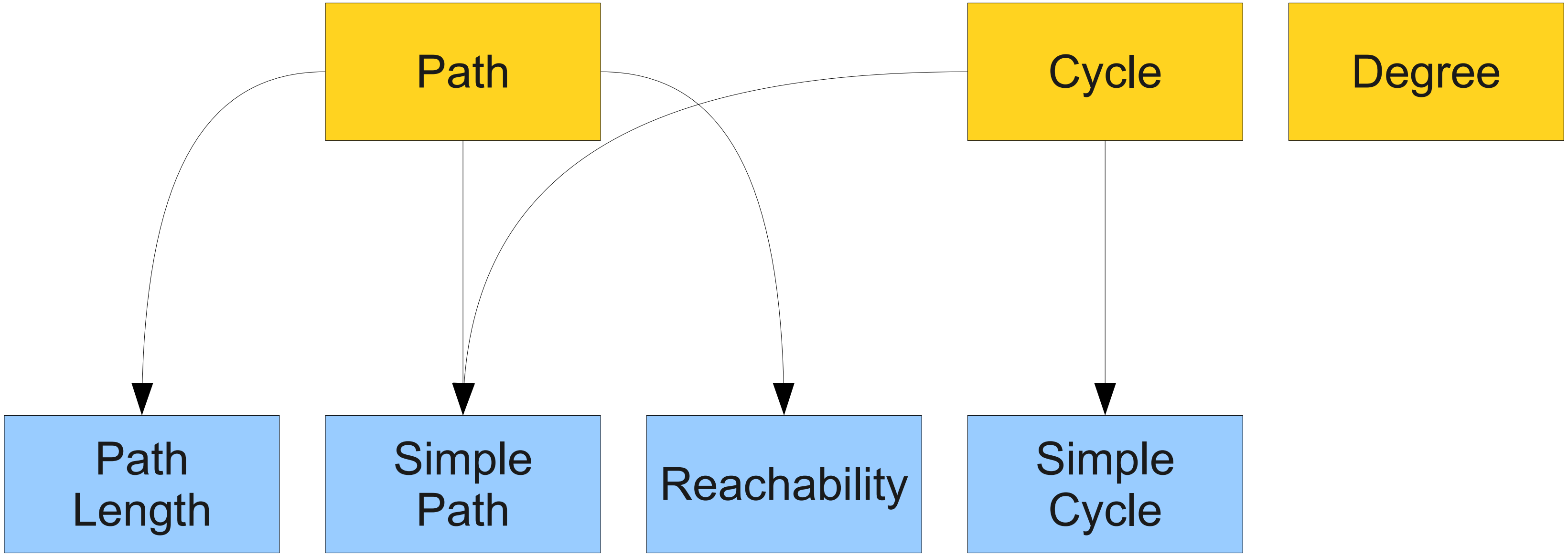
Graph

Traversing a DAG



Traversing a DAG

Graph



Path

Cycle

Degree

Path Length

Simple Path

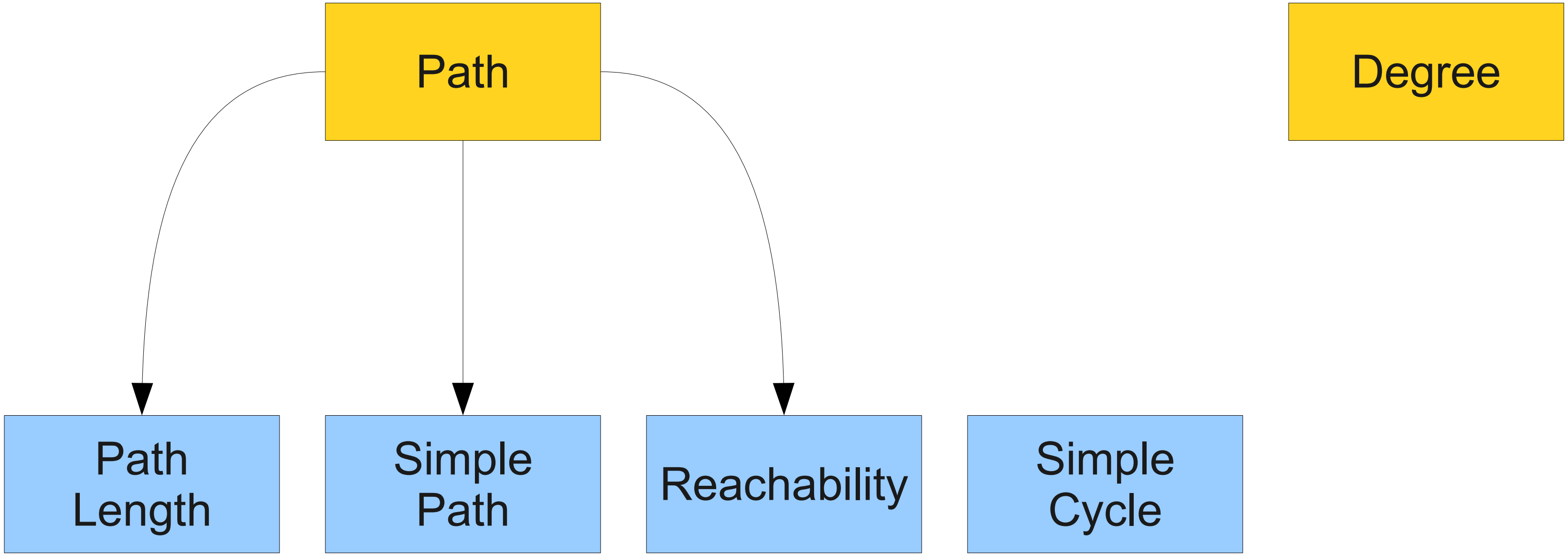
Reachability

Simple Cycle

Traversing a DAG

Graph

Cycle



Path

Degree

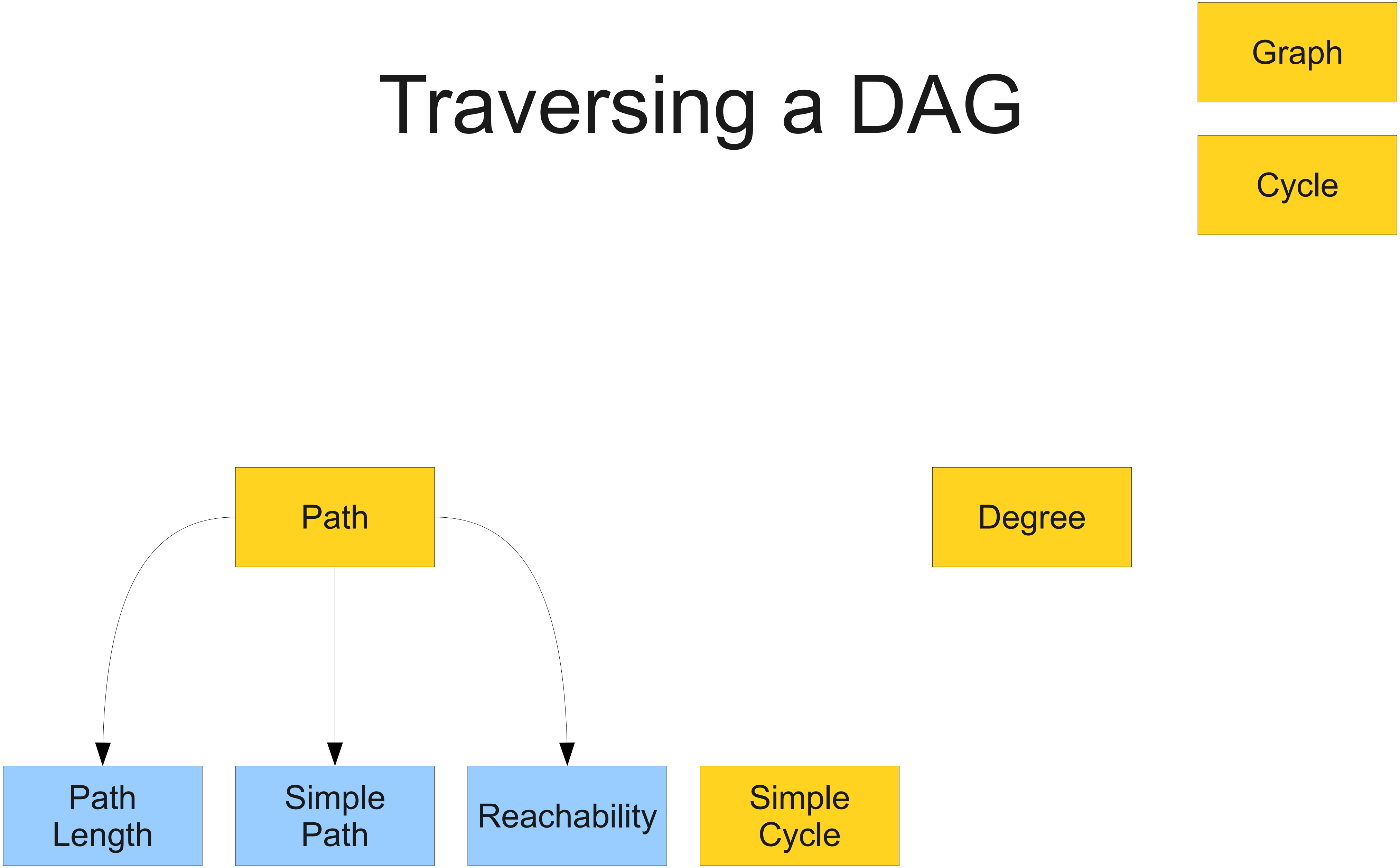
Path Length

Simple Path

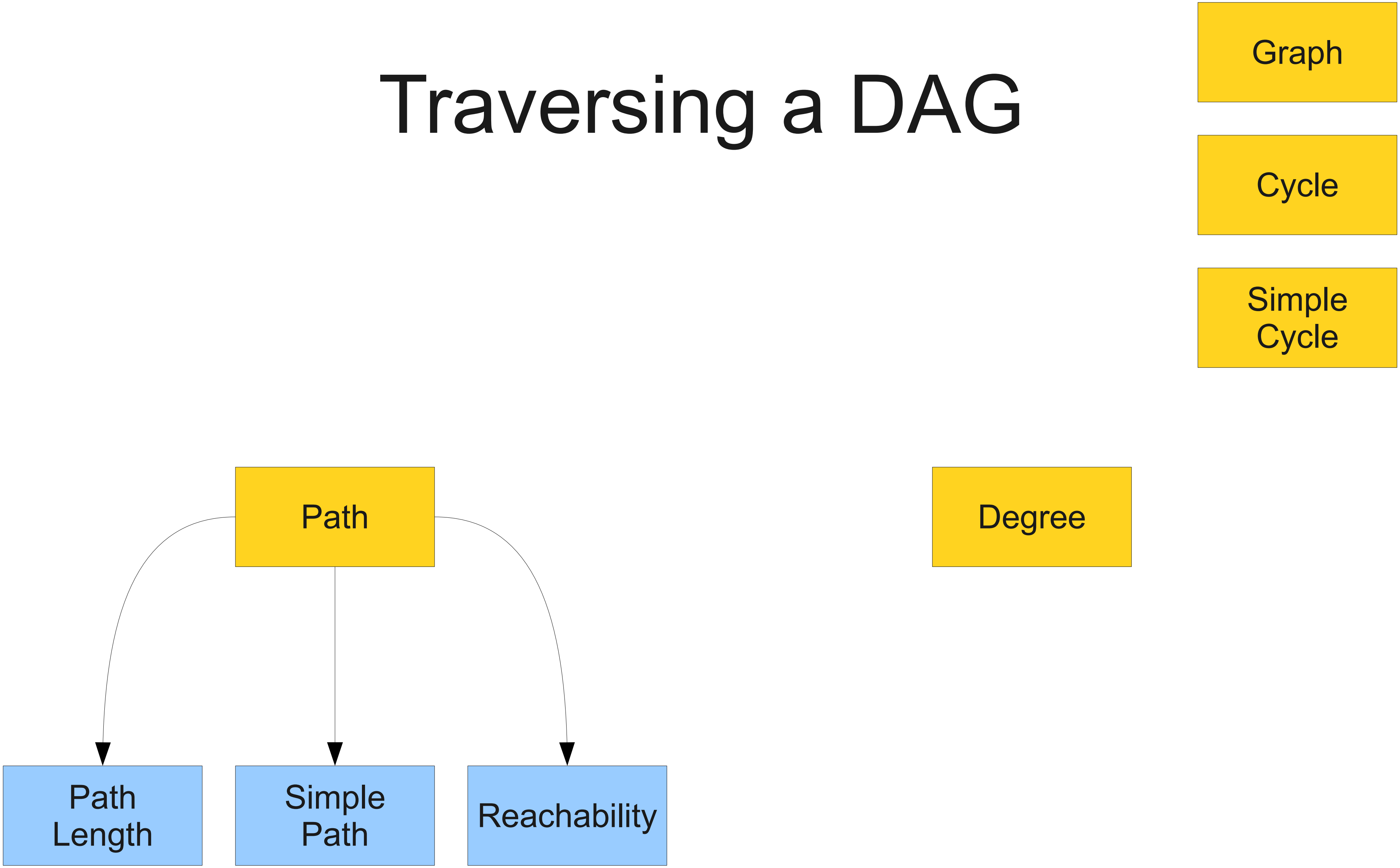
Reachability

Simple Cycle

Traversing a DAG



Traversing a DAG



Graph

Cycle

Simple Cycle

Path

Degree

Path Length

Simple Path

Reachability

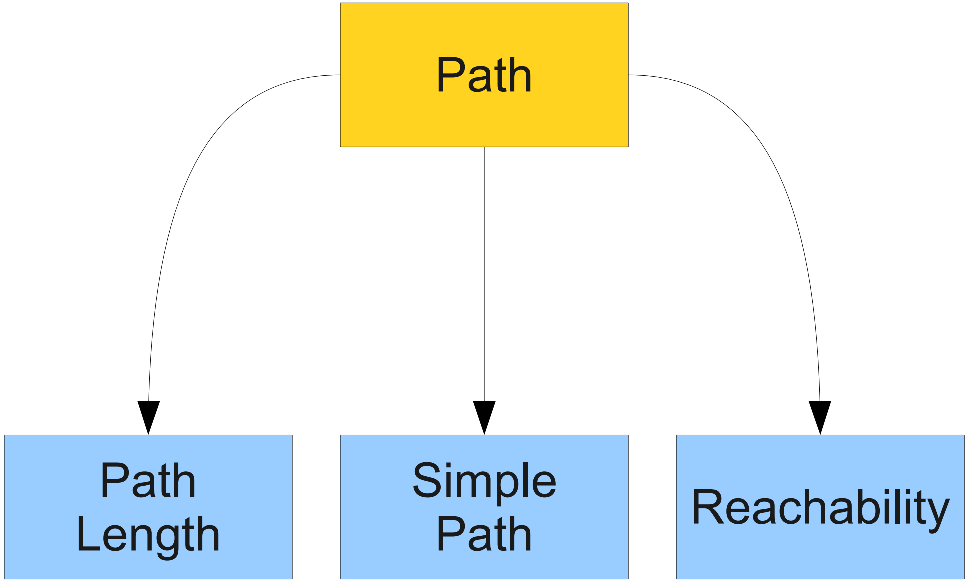
Traversing a DAG

Graph

Cycle

Simple Cycle

Degree



Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

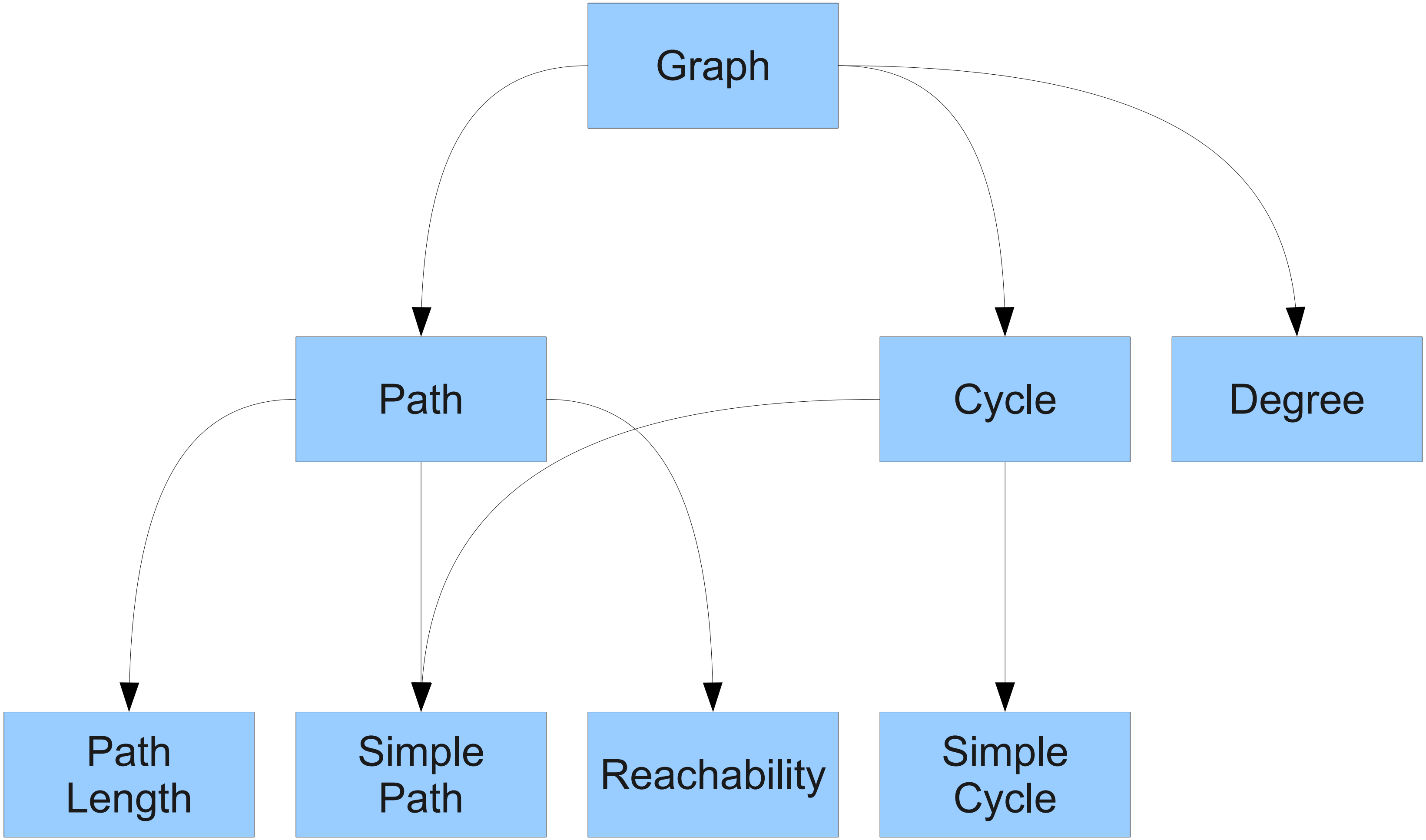
Path

Path Length

Simple Path

Reachability

Traversing a DAG



Graph

Path

Cycle

Degree

Path Length

Simple Path

Reachability

Simple Cycle

- Graph
- Cycle
- Simple Cycle
- Degree
- Path
- Path Length
- Simple Path
- Reachability

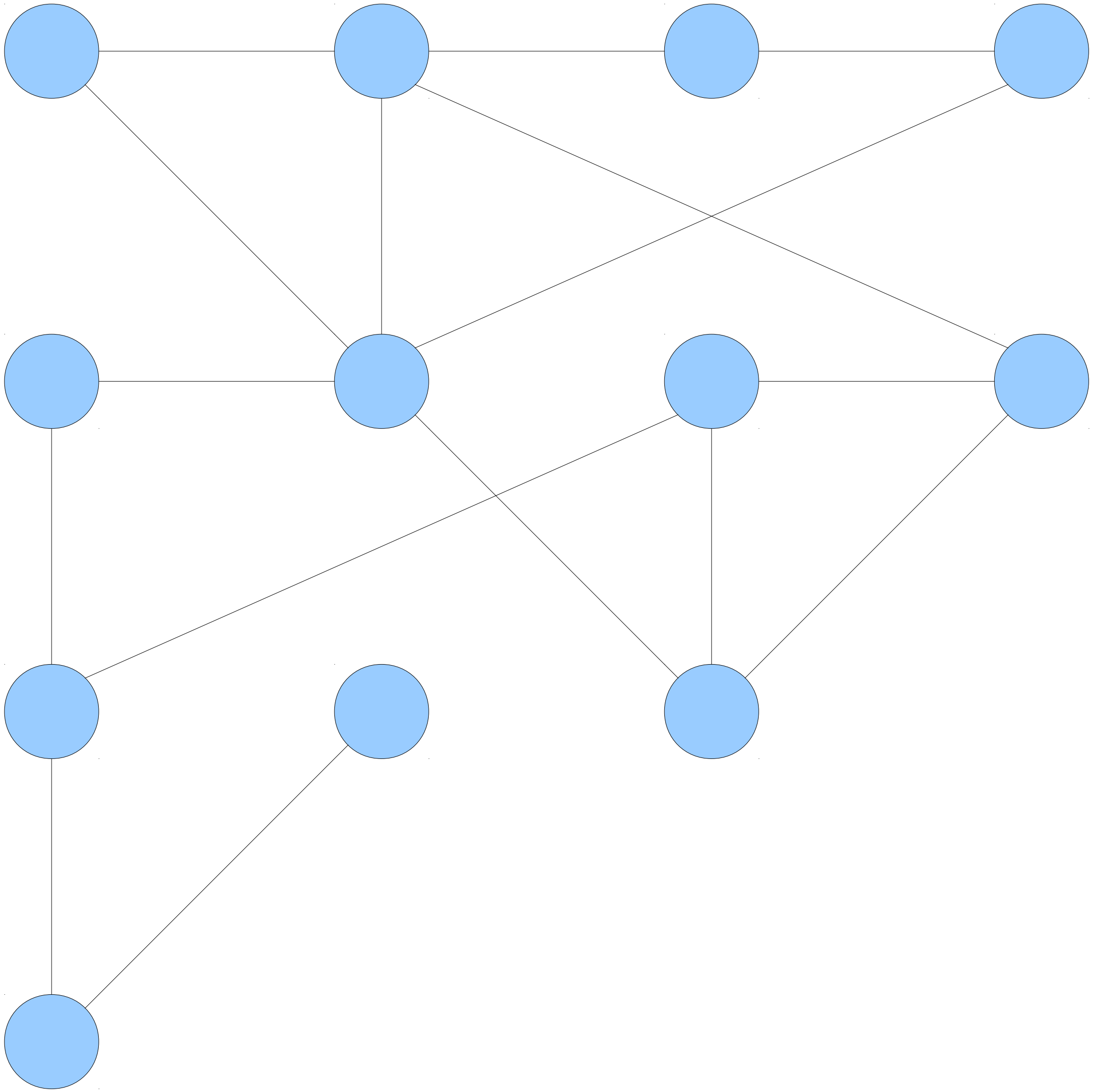
Topological Sort

- Order the nodes of a DAG so no node is picked before its parents.
- Algorithm:
    - Find a node with no incoming edges (indegree 0)
    - Remove it from the graph.
    - Add it to the resulting ordering.
- Not necessarily unique.
    - Question: When is it unique?

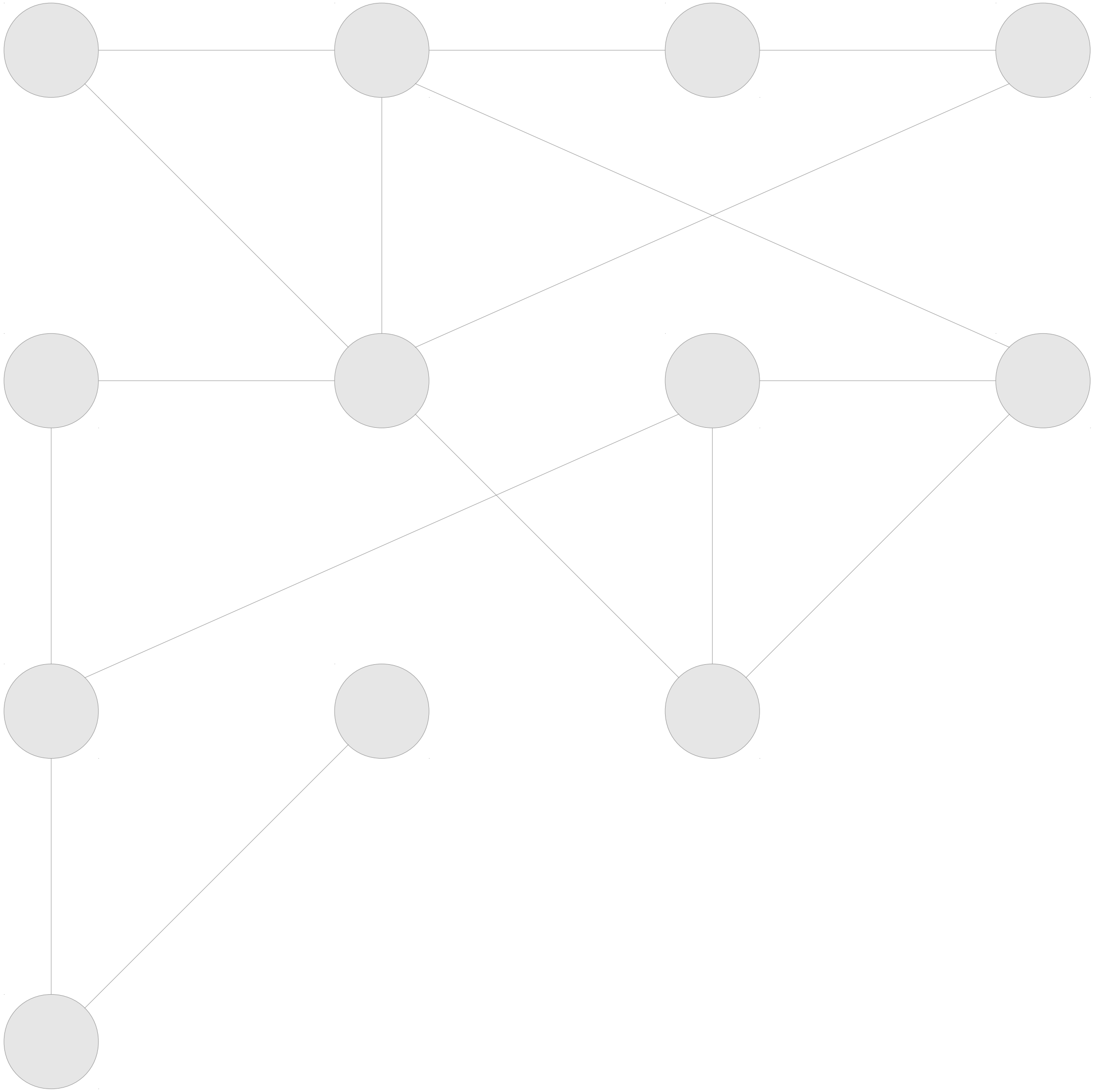
Analyzing Topological Sort

- Assumes at each step that the DAG has a node with indegree zero. Is this always true?
- Claim one: **Every DAG has such a node**.
  - Proof sketch: If this isn't true, then each node has at least one incoming edge. Start at any node and keep following backwards across that edge. Eventually you will find the same node twice and have found a cycle.
- Claim two: **Removing such a node leaves the DAG a DAG**.
  - Proof sketch: If the resulting graph has a cycle, the old graph had a cycle as well.

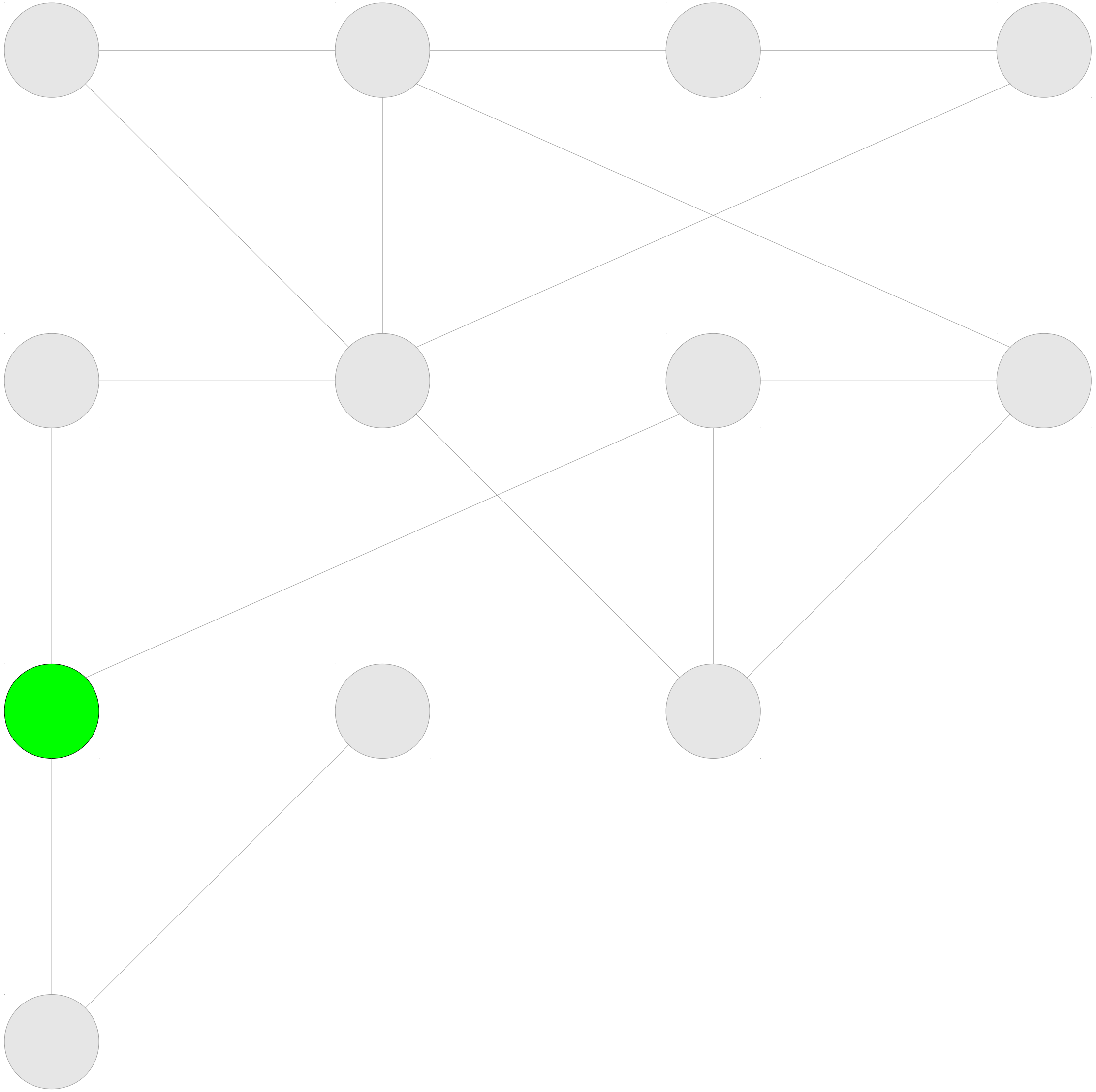
Traversing an Arbitrary Graph



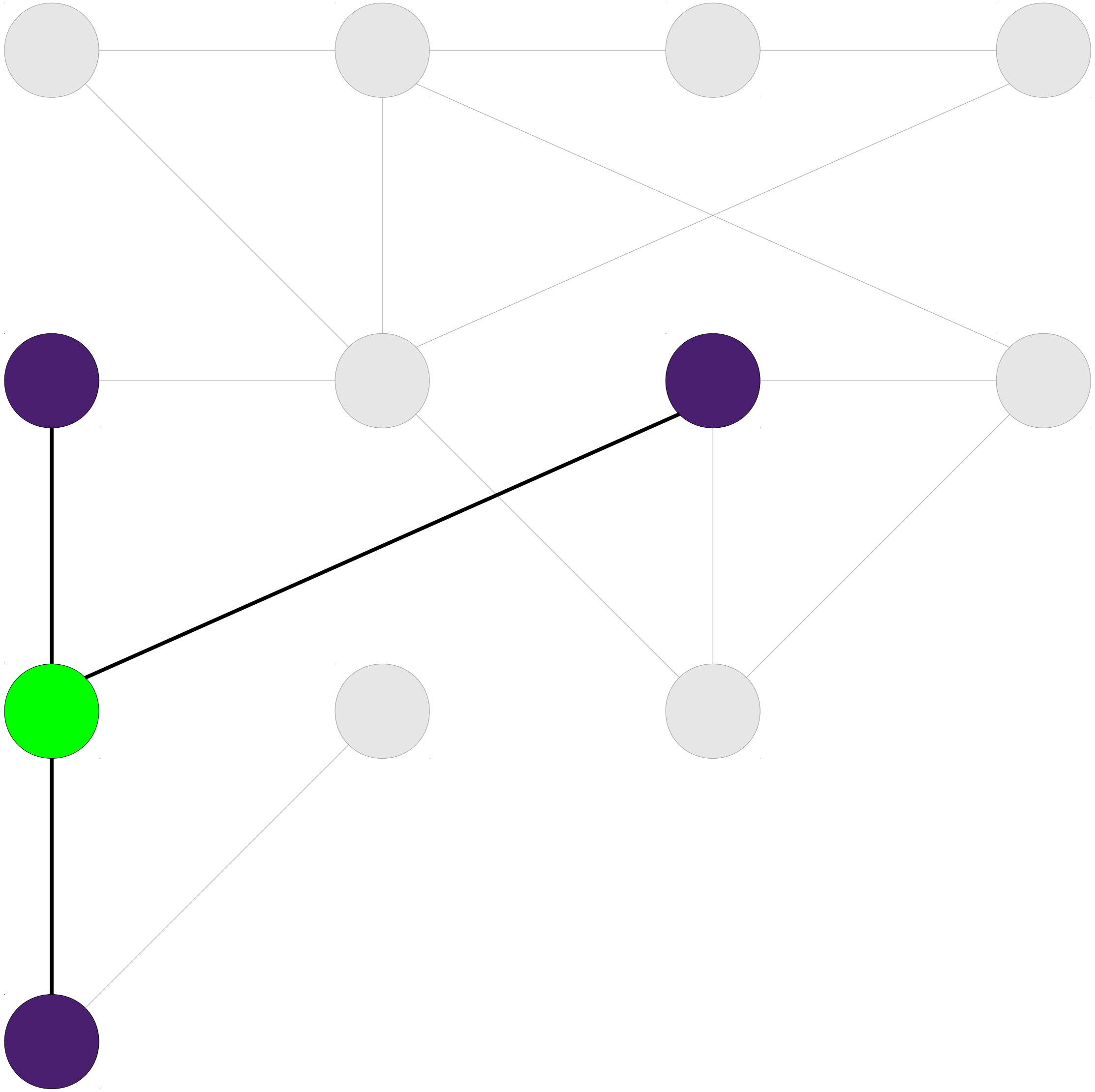
Traversing an Arbitrary Graph



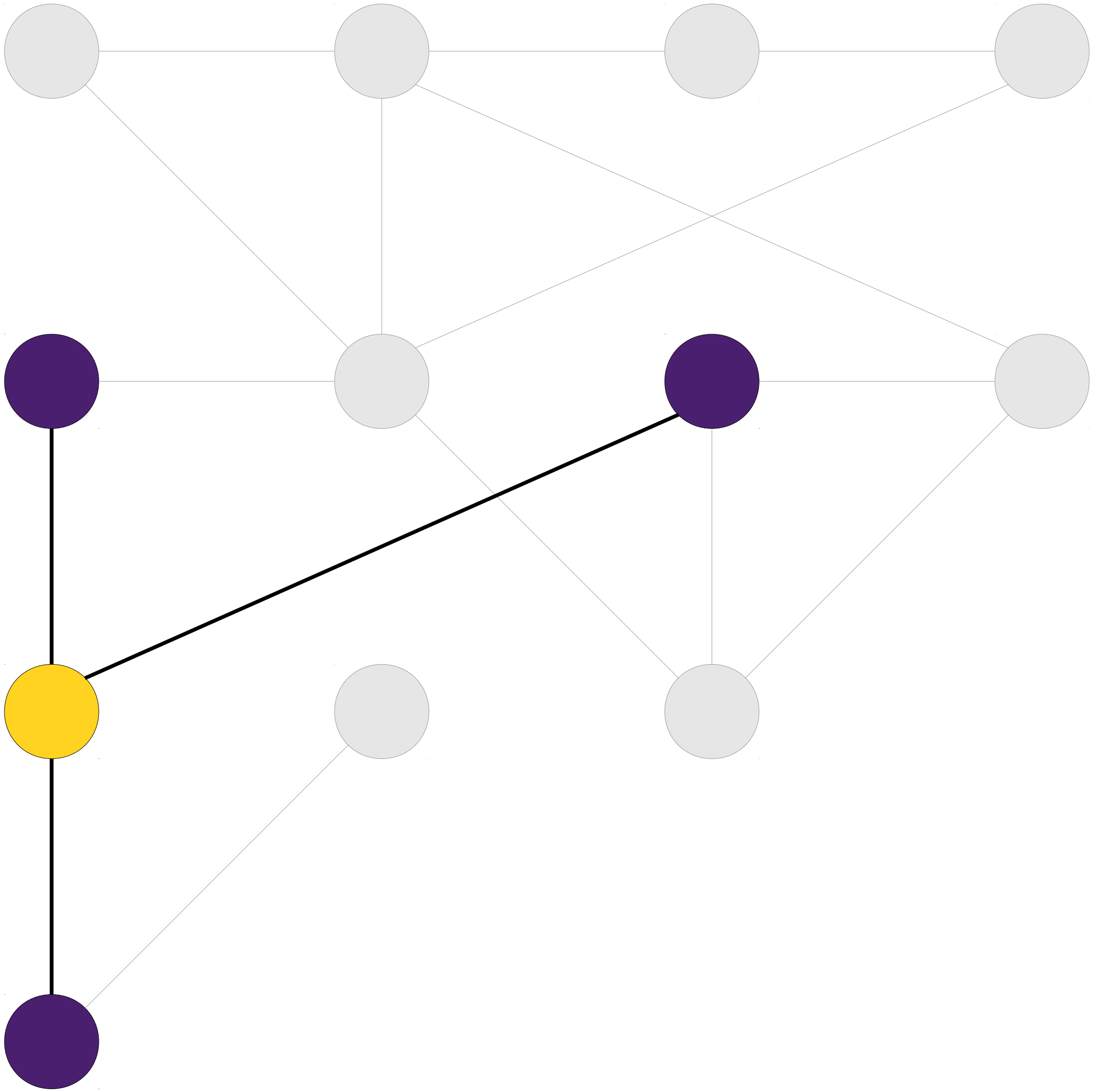
Traversing an Arbitrary Graph



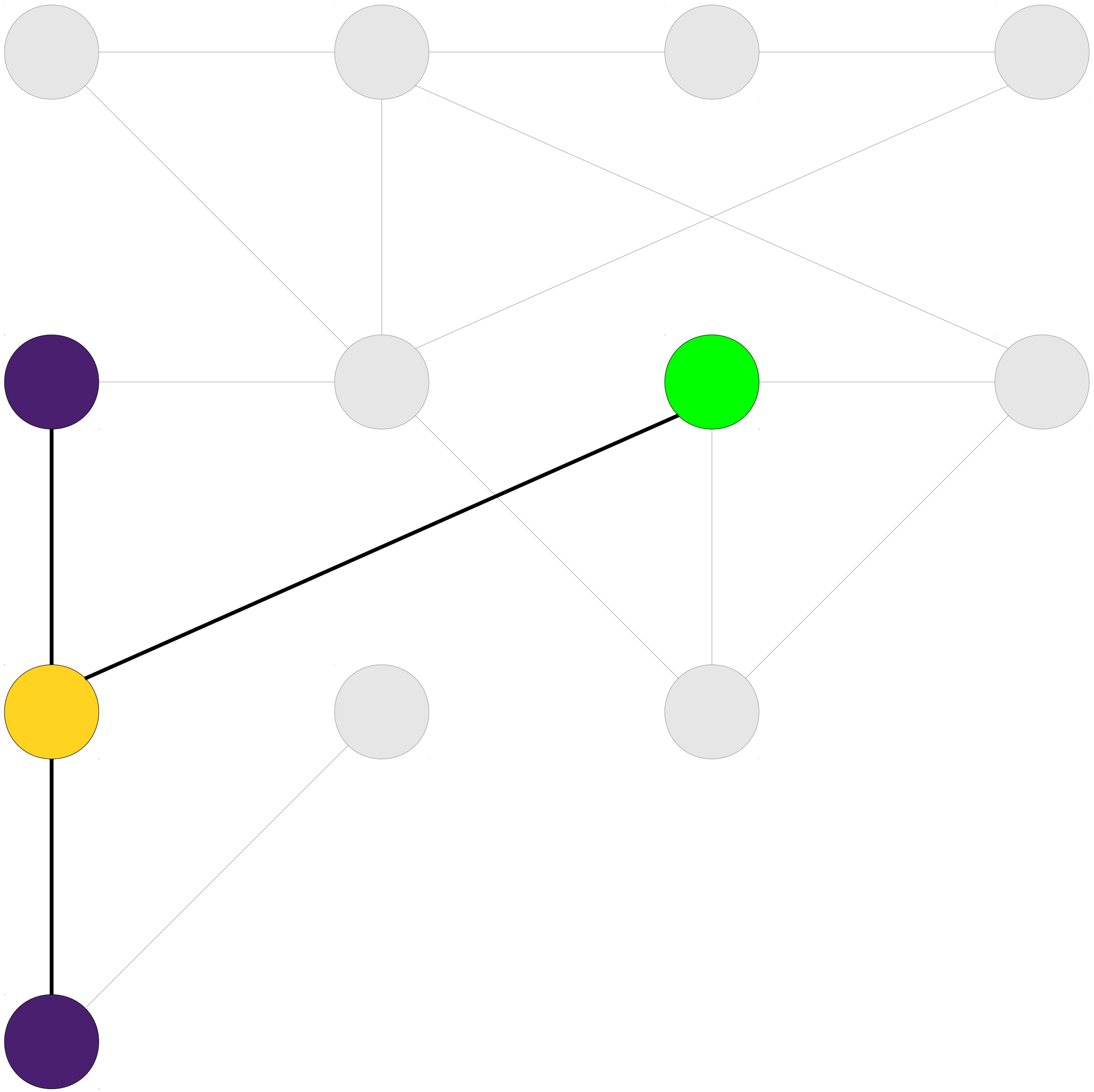
Traversing an Arbitrary Graph



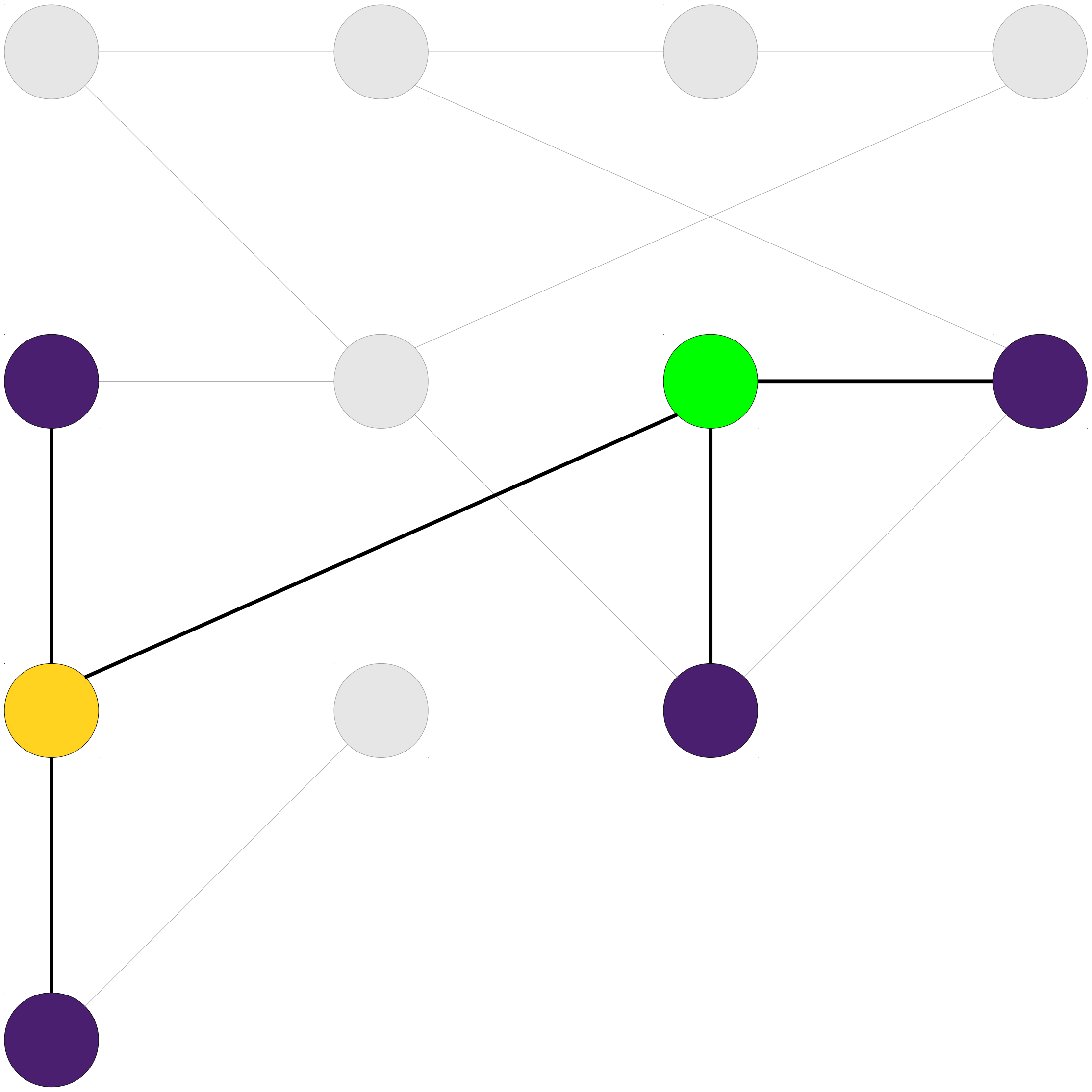
Traversing an Arbitrary Graph



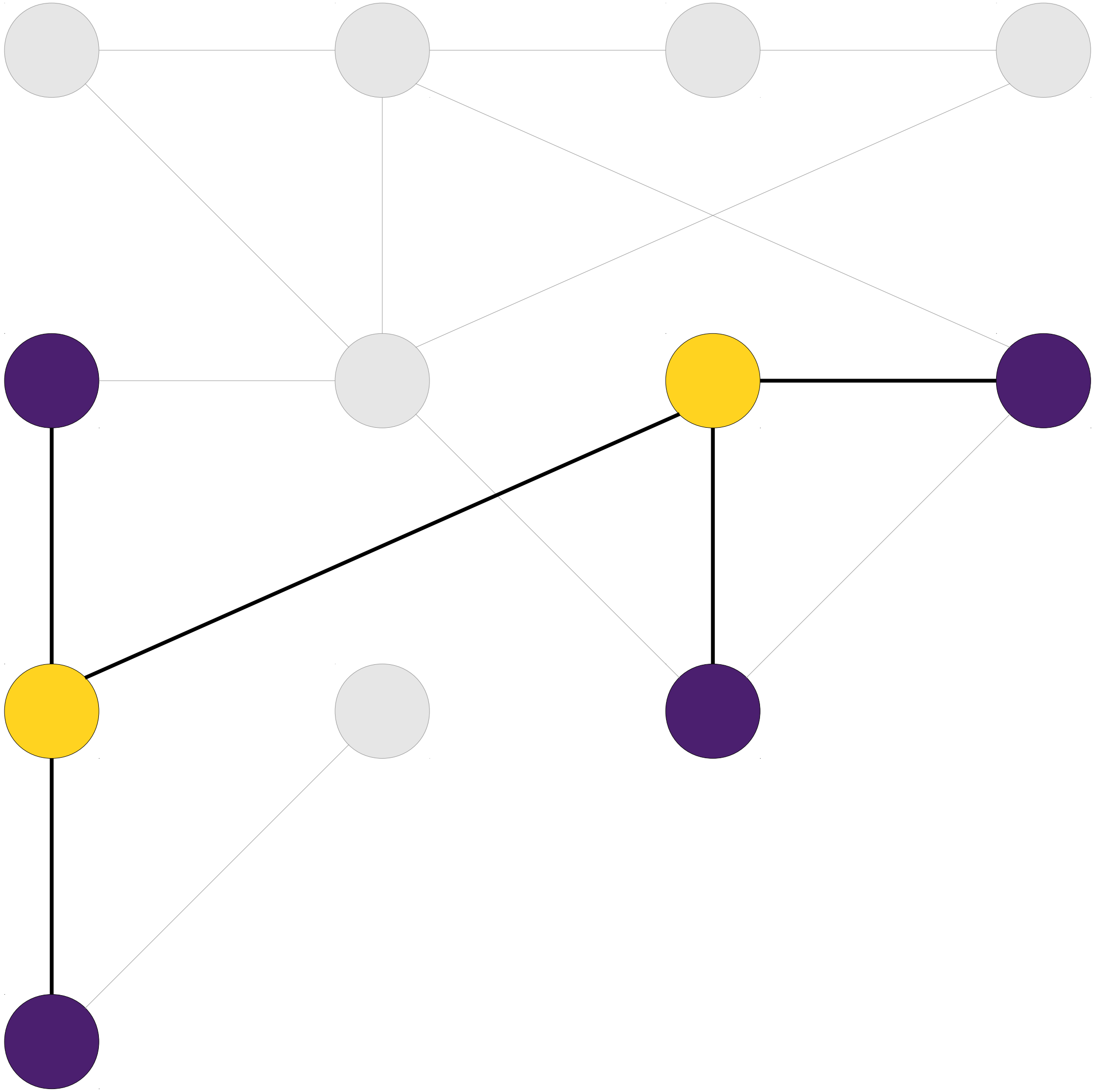
Traversing an Arbitrary Graph



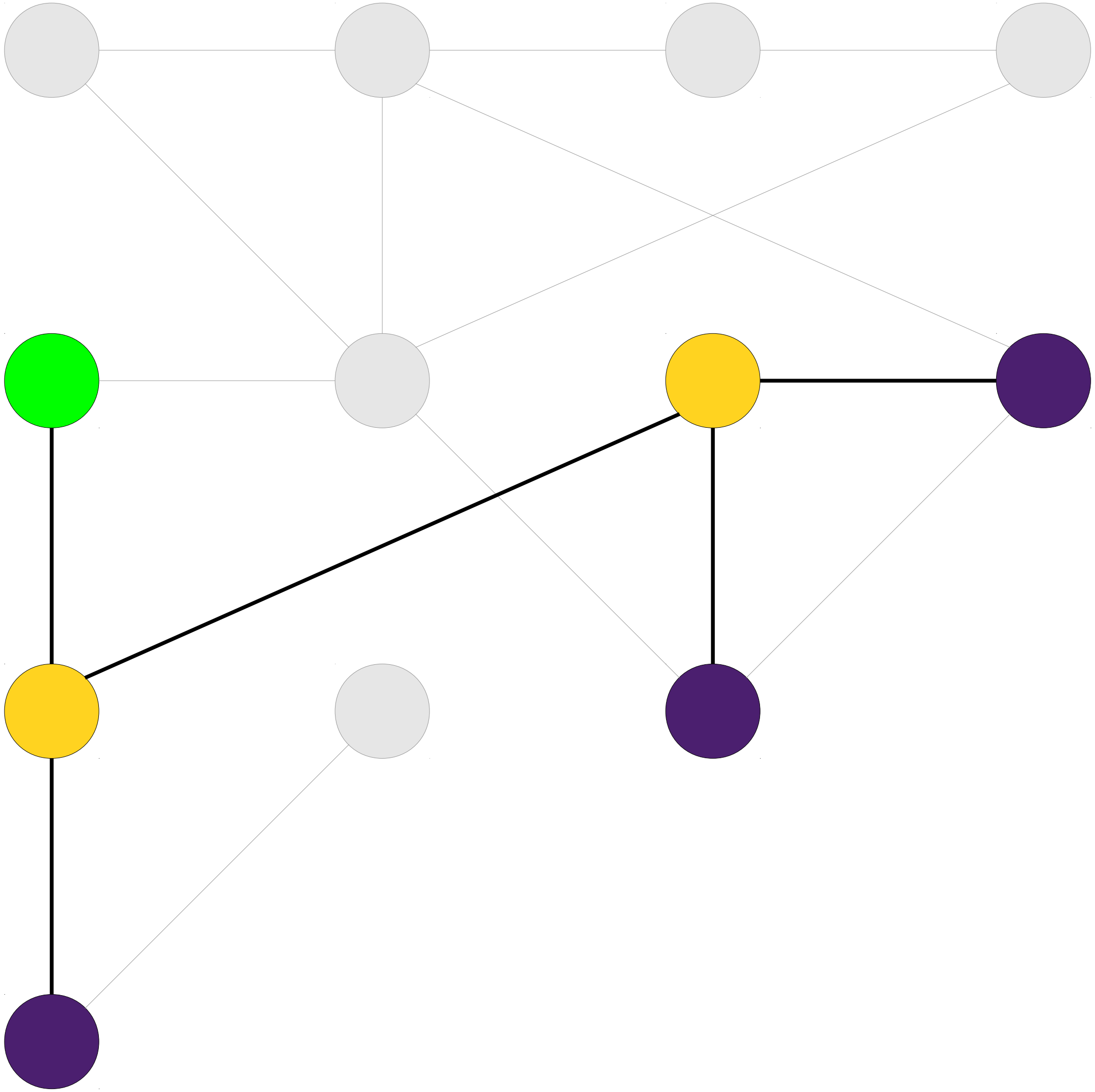
Traversing an Arbitrary Graph



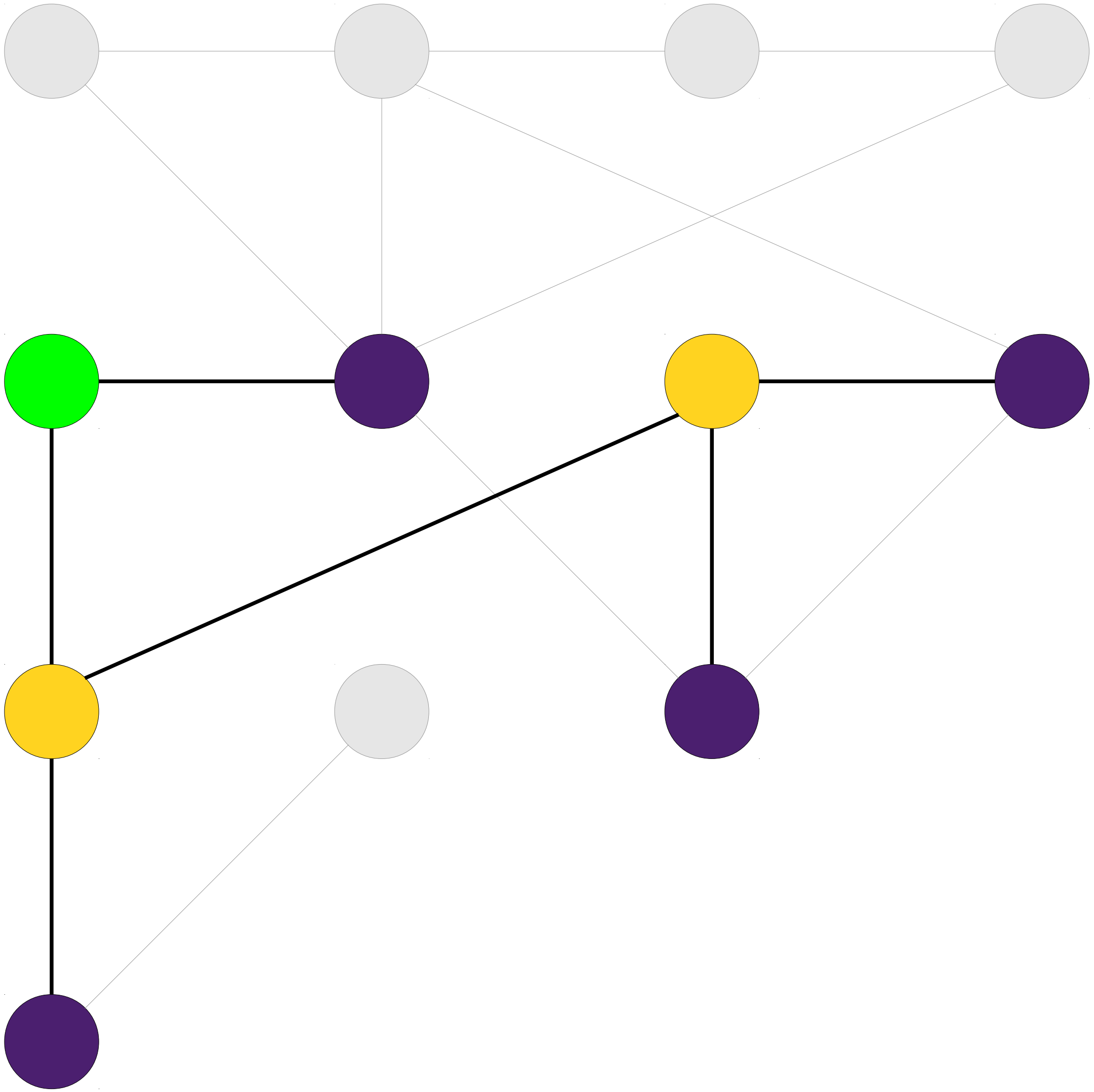
Traversing an Arbitrary Graph



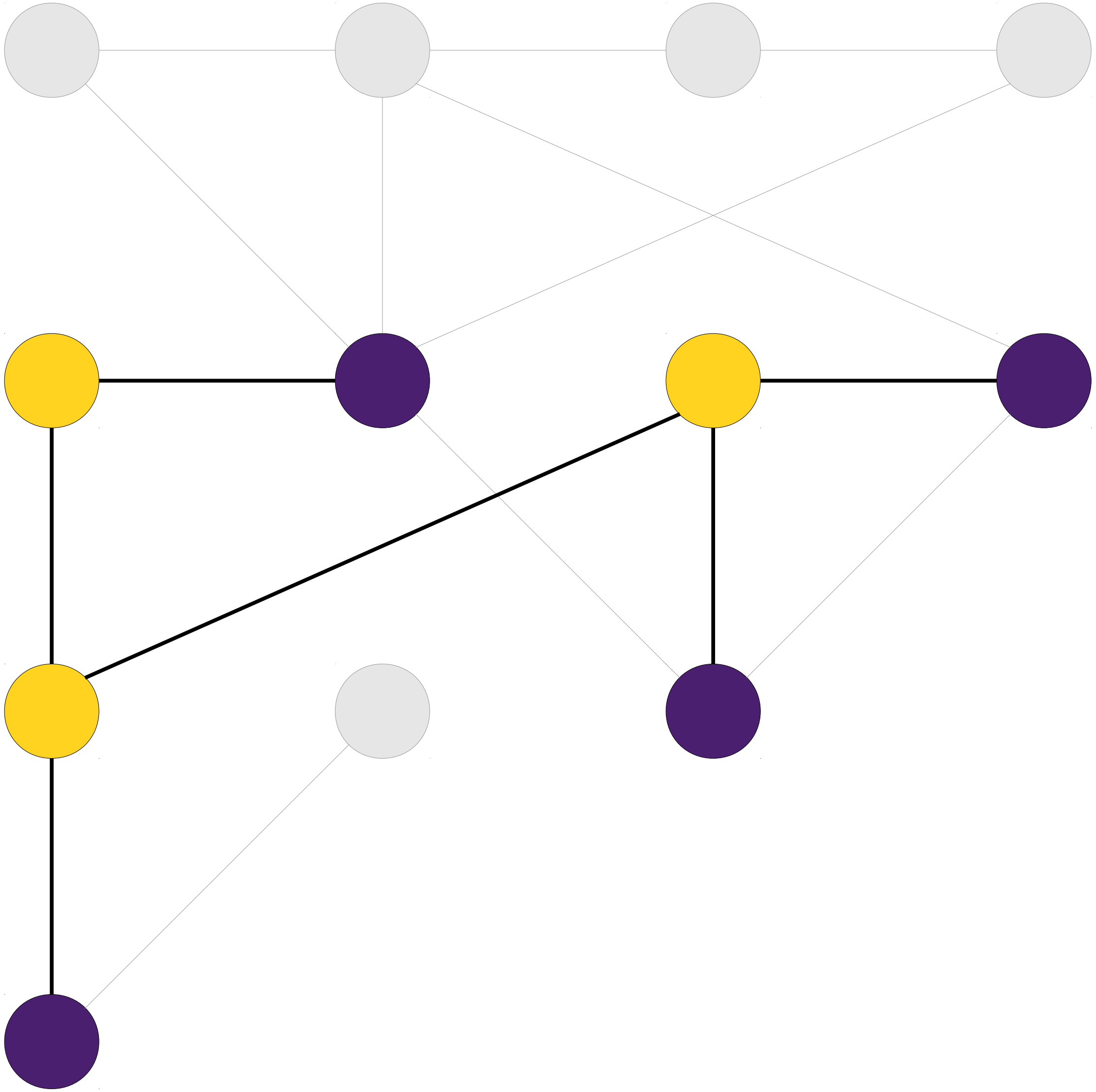
Traversing an Arbitrary Graph



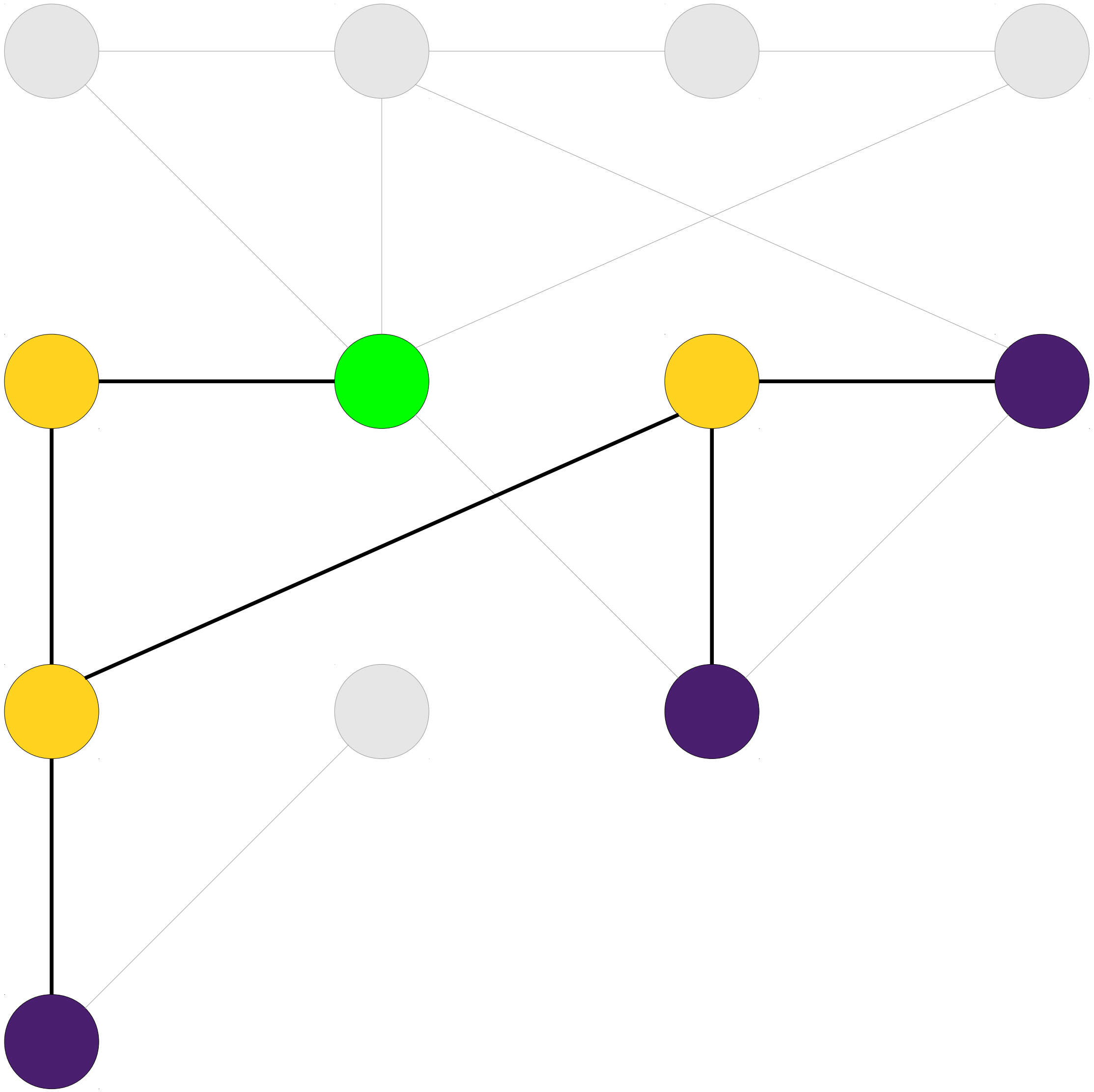
Traversing an Arbitrary Graph



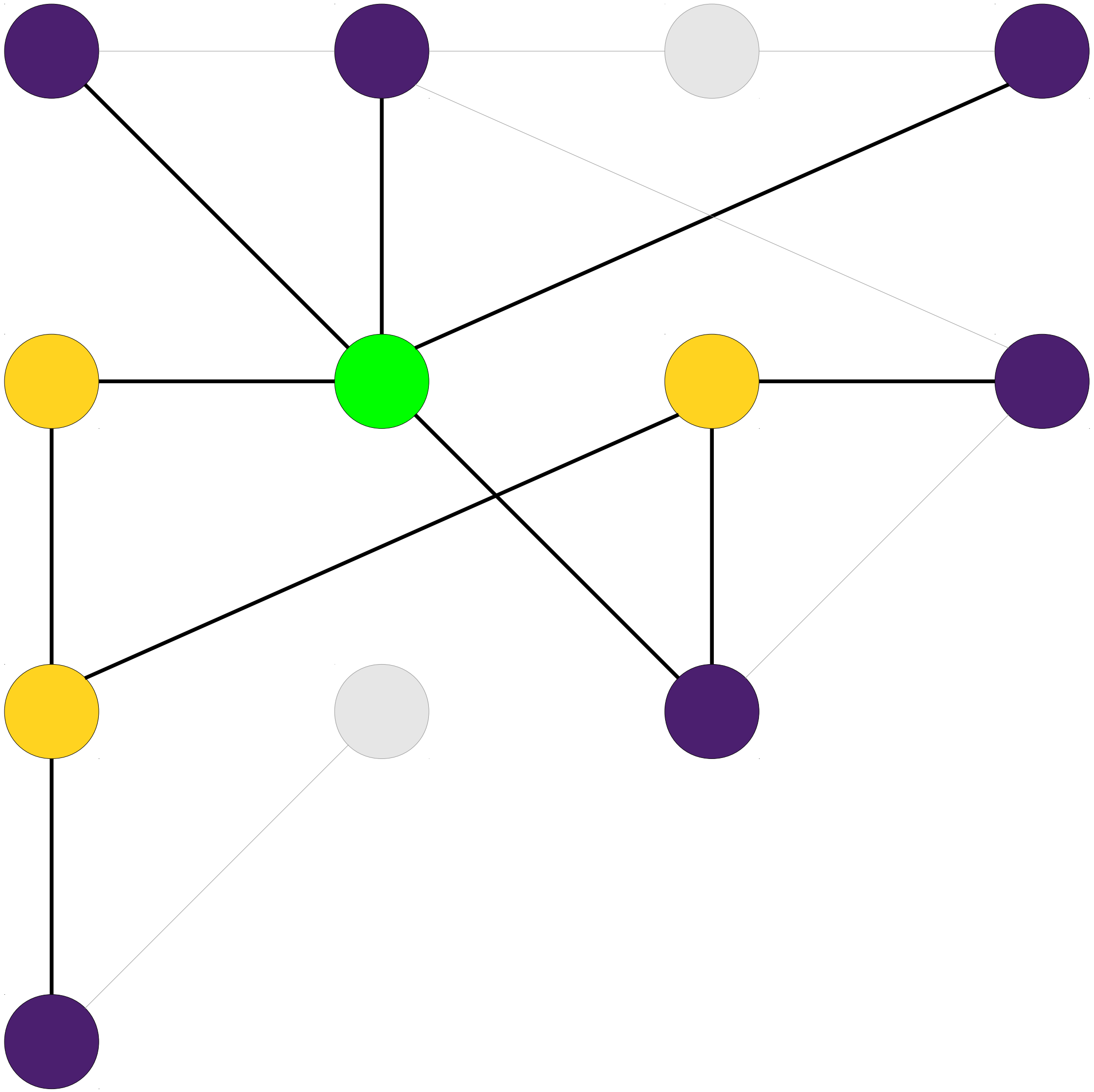
Traversing an Arbitrary Graph



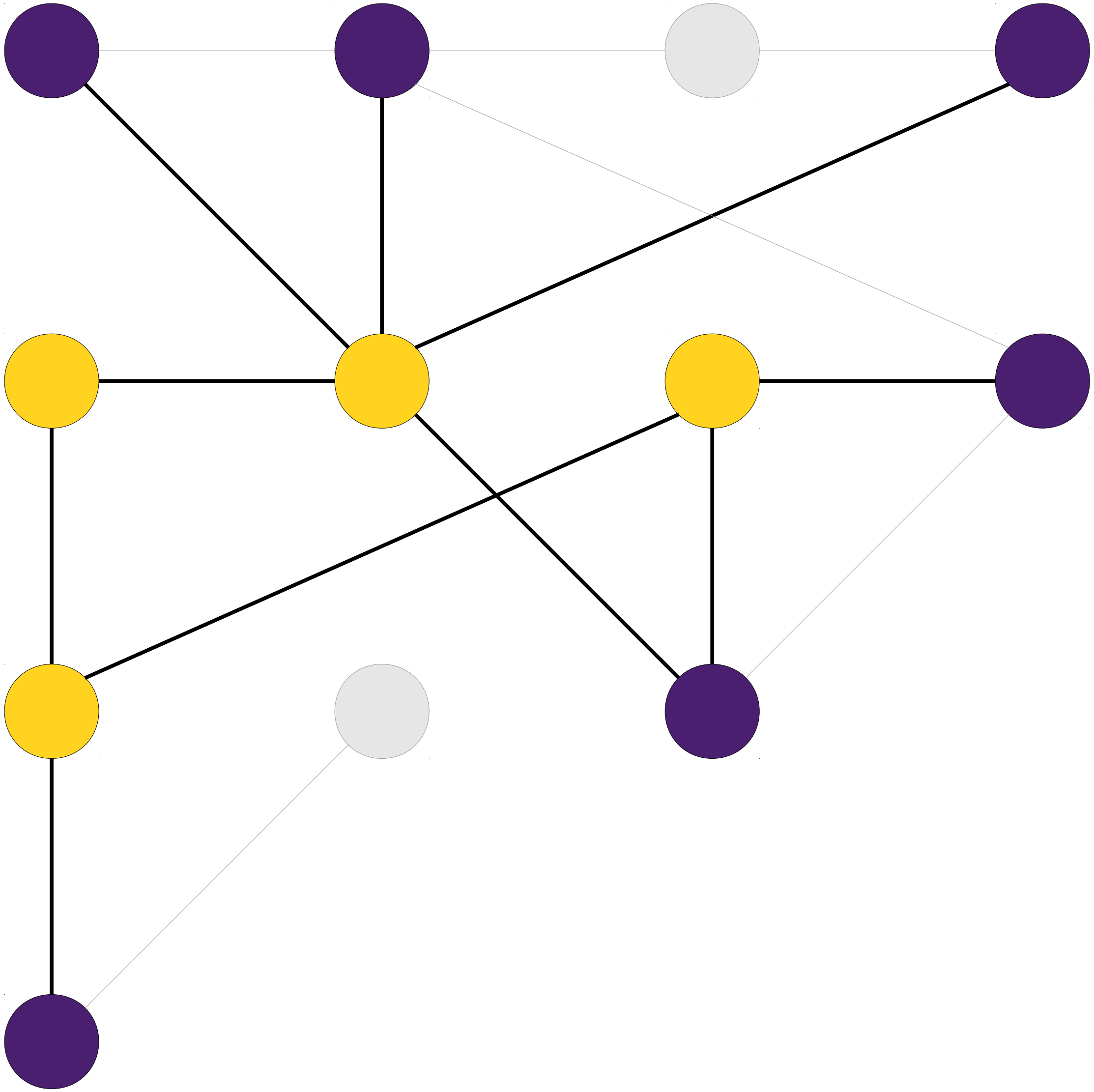
Traversing an Arbitrary Graph



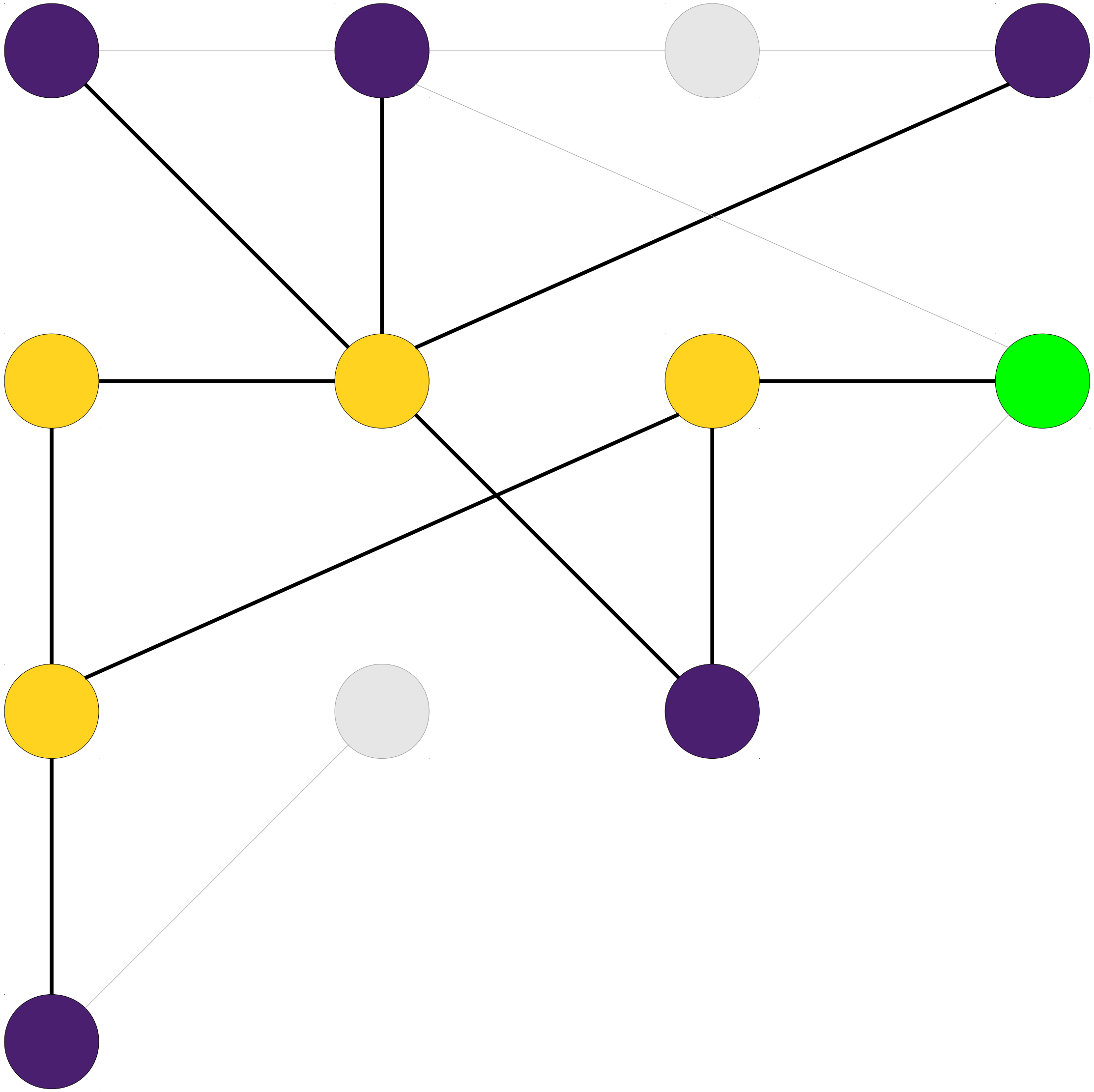
Traversing an Arbitrary Graph



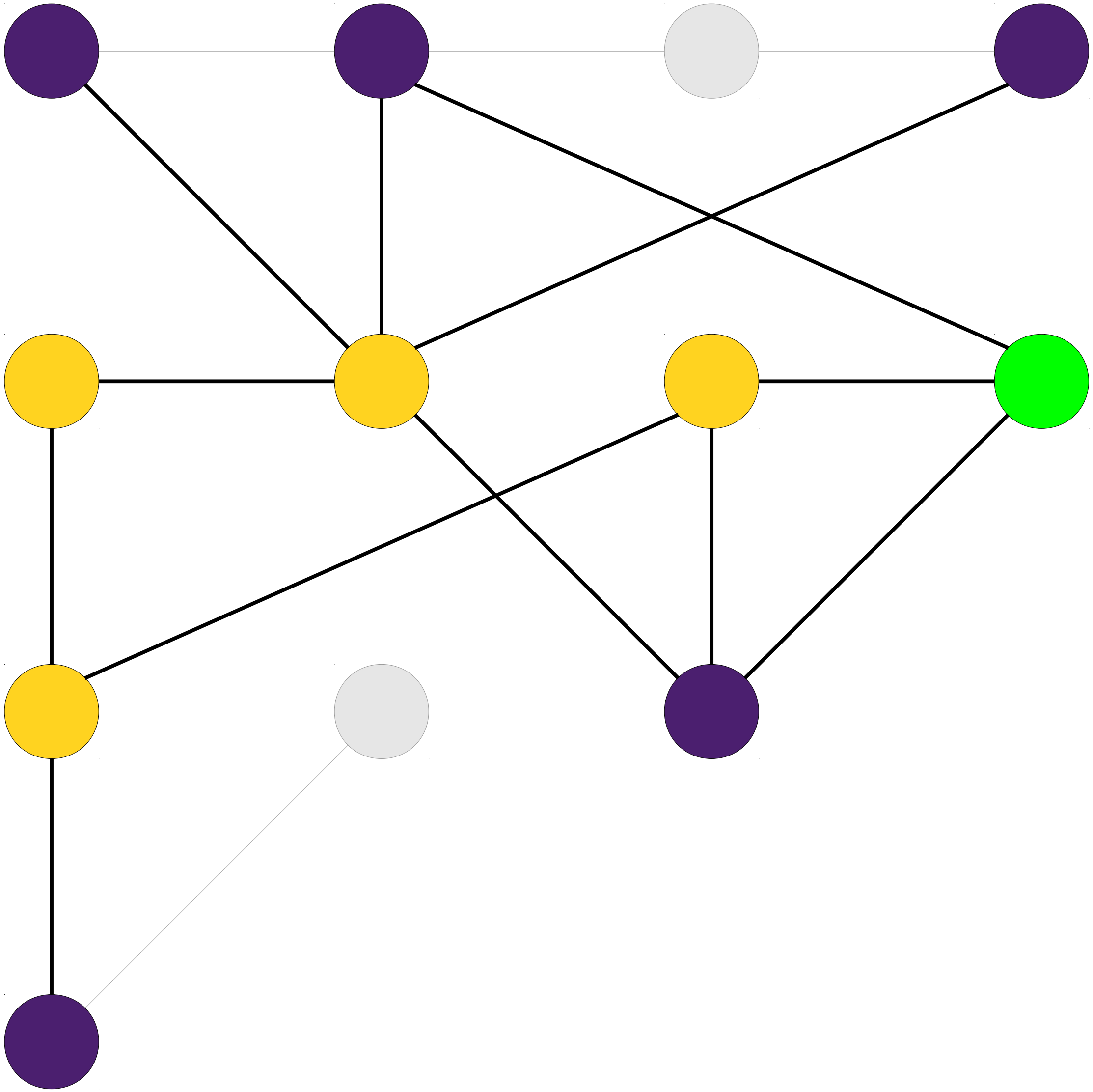
Traversing an Arbitrary Graph



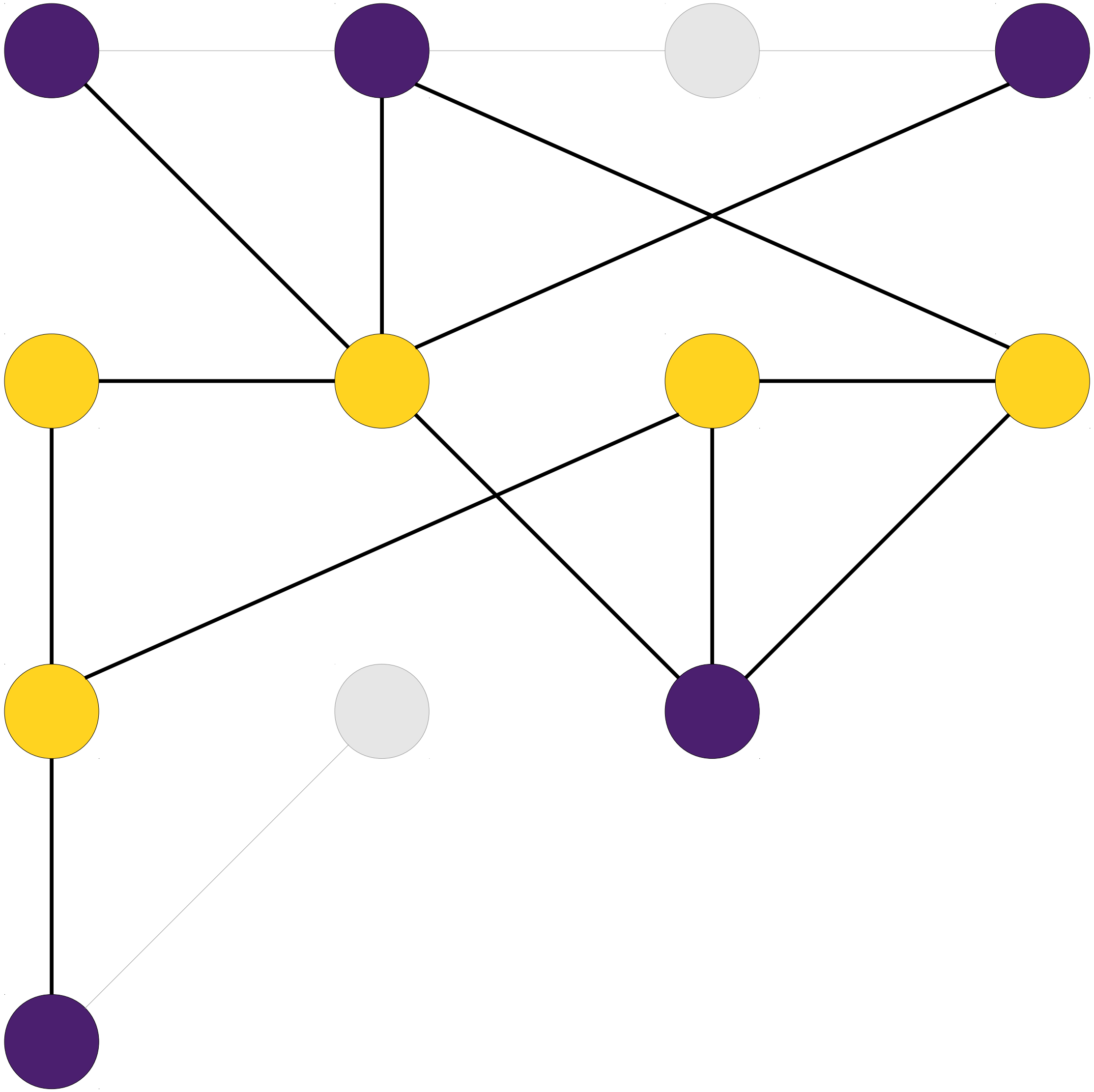
Traversing an Arbitrary Graph



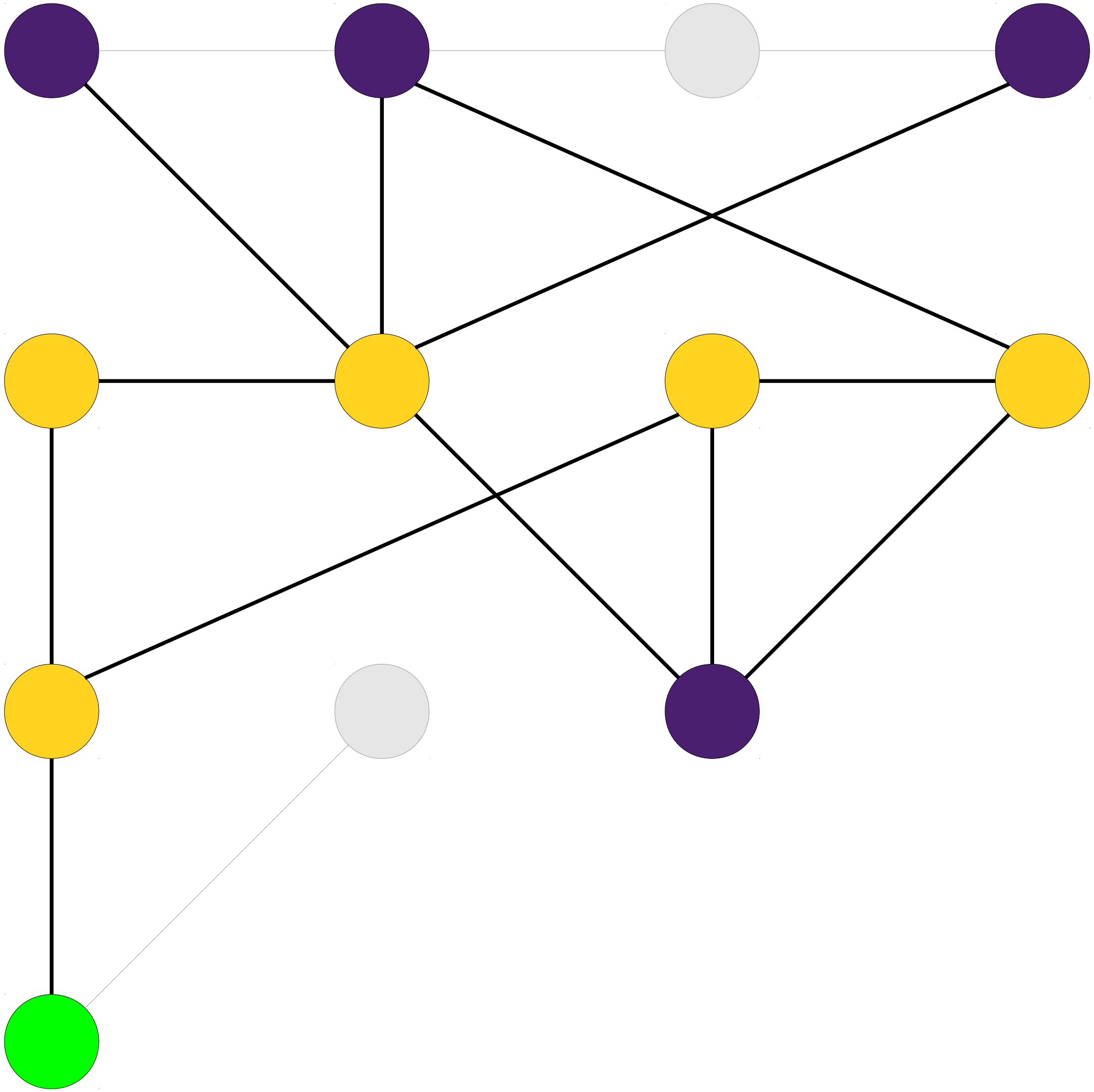
Traversing an Arbitrary Graph



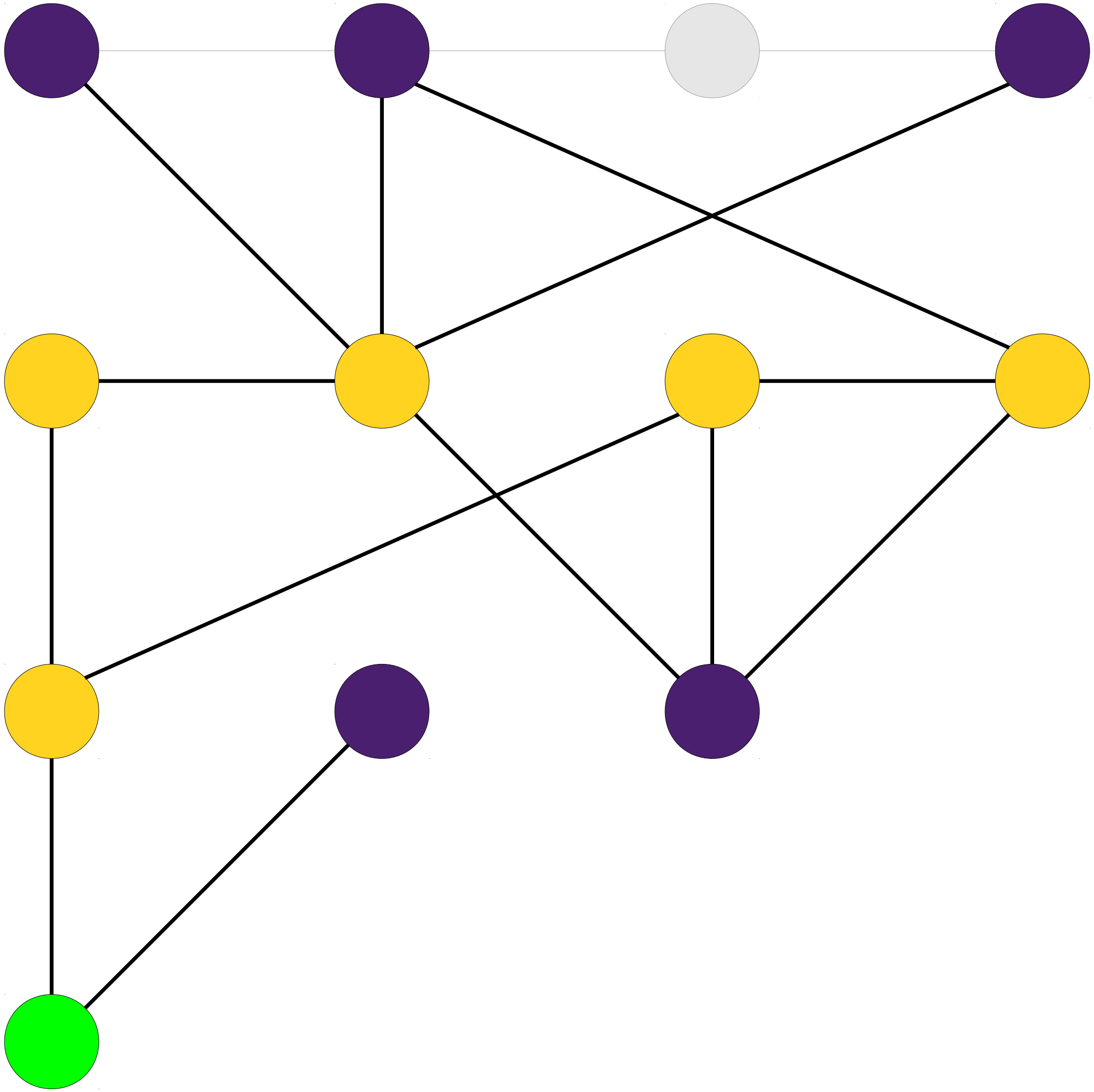
Traversing an Arbitrary Graph



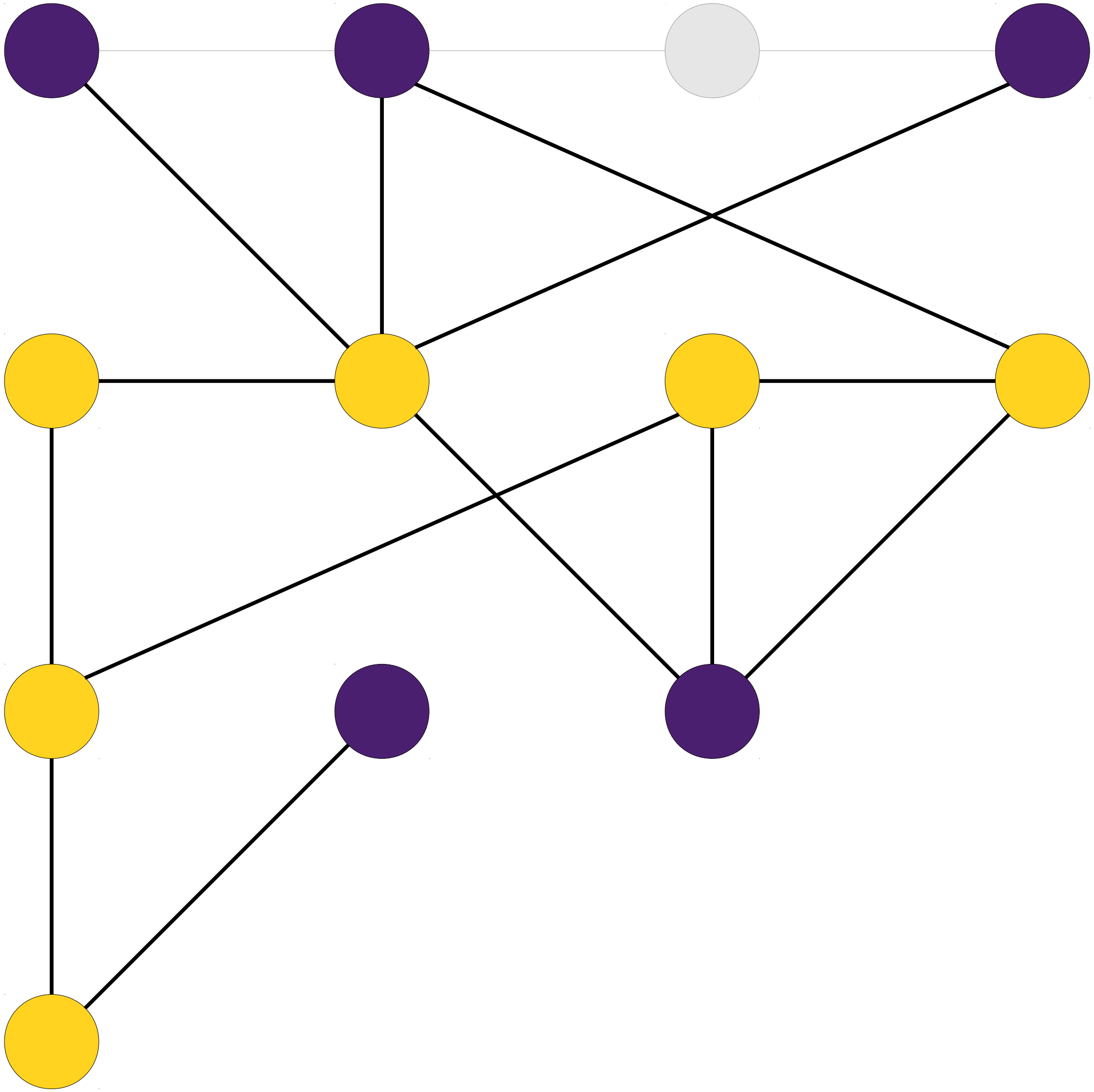
Traversing an Arbitrary Graph



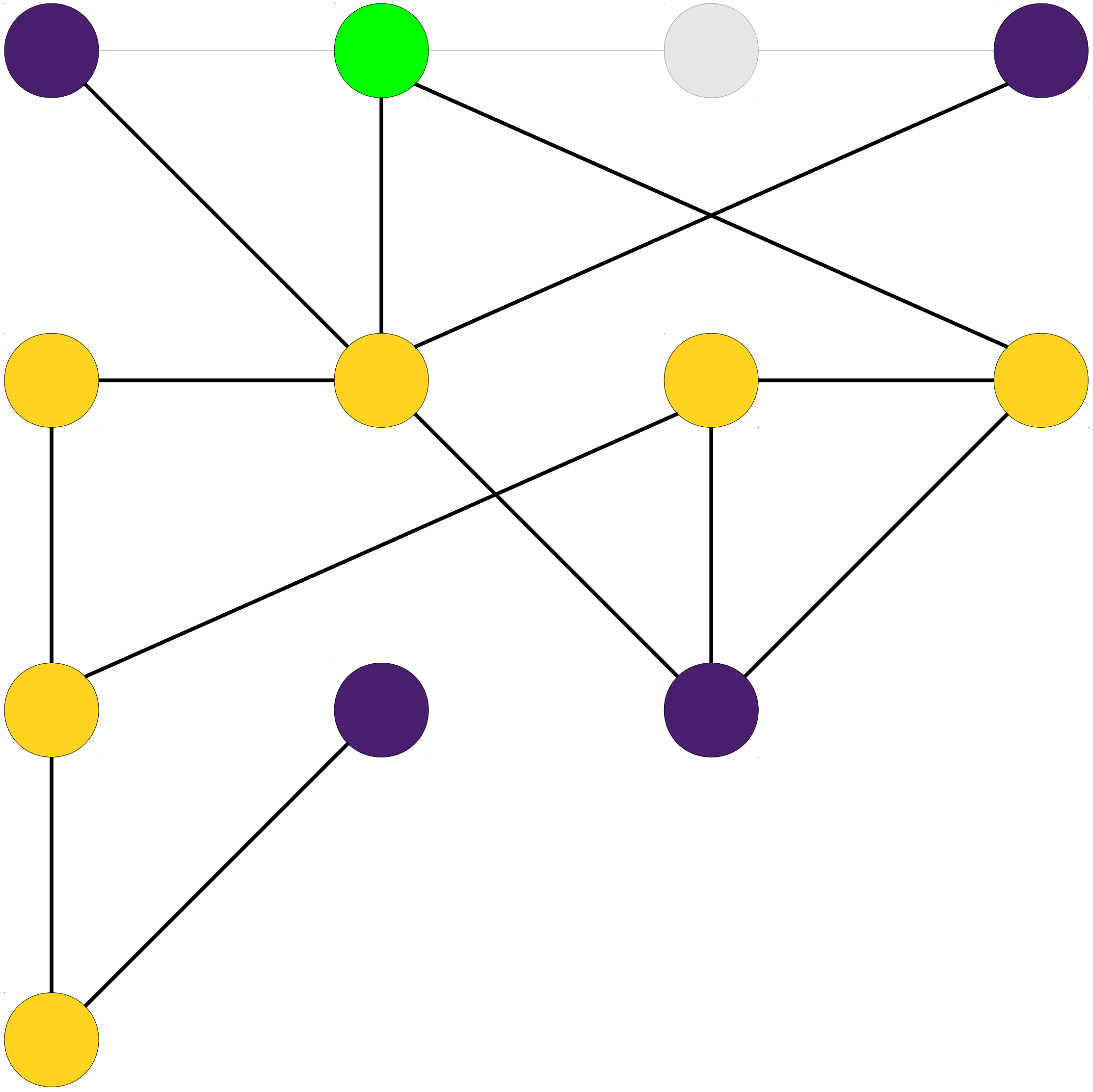
Traversing an Arbitrary Graph



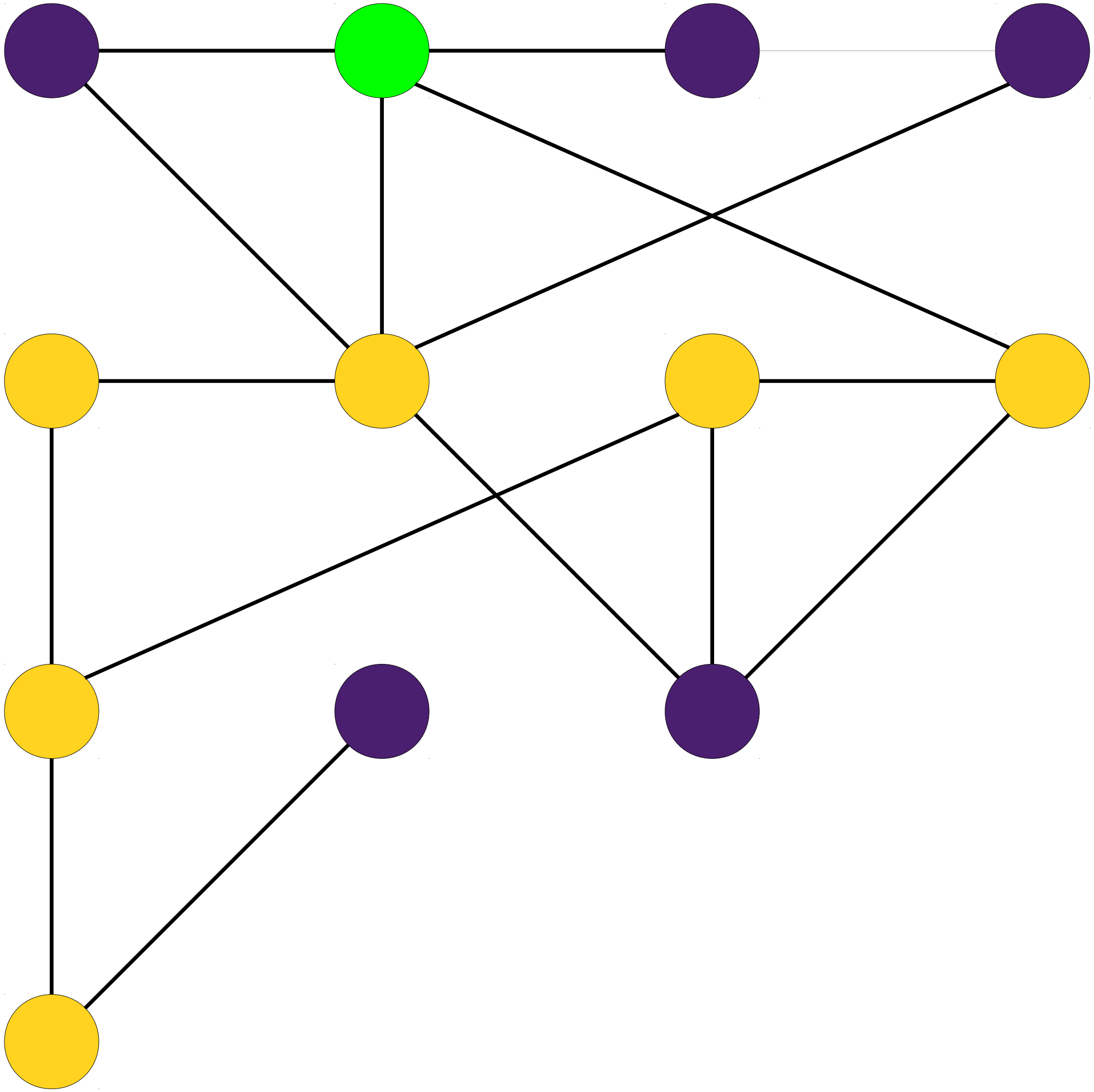
Traversing an Arbitrary Graph



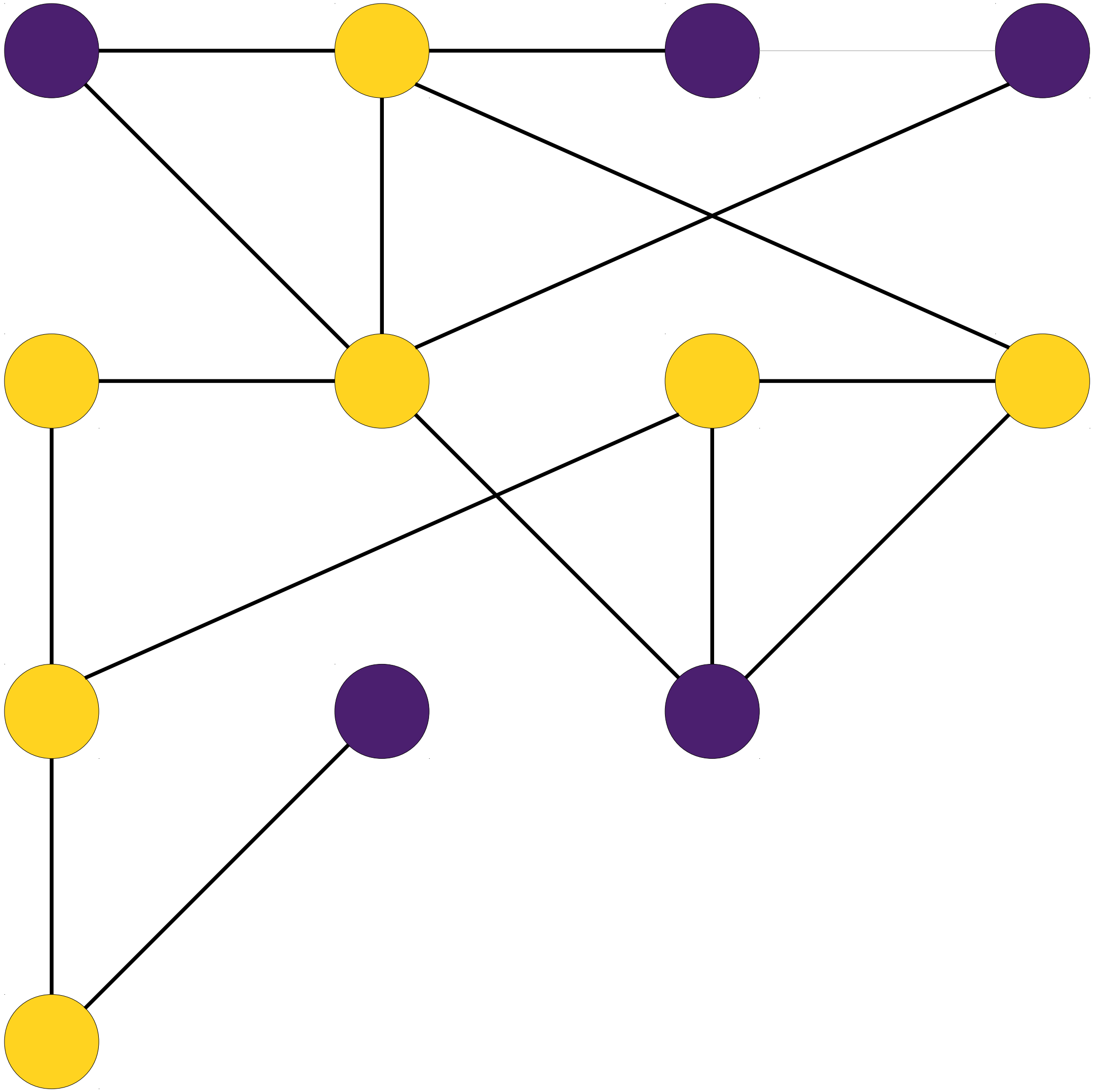
Traversing an Arbitrary Graph



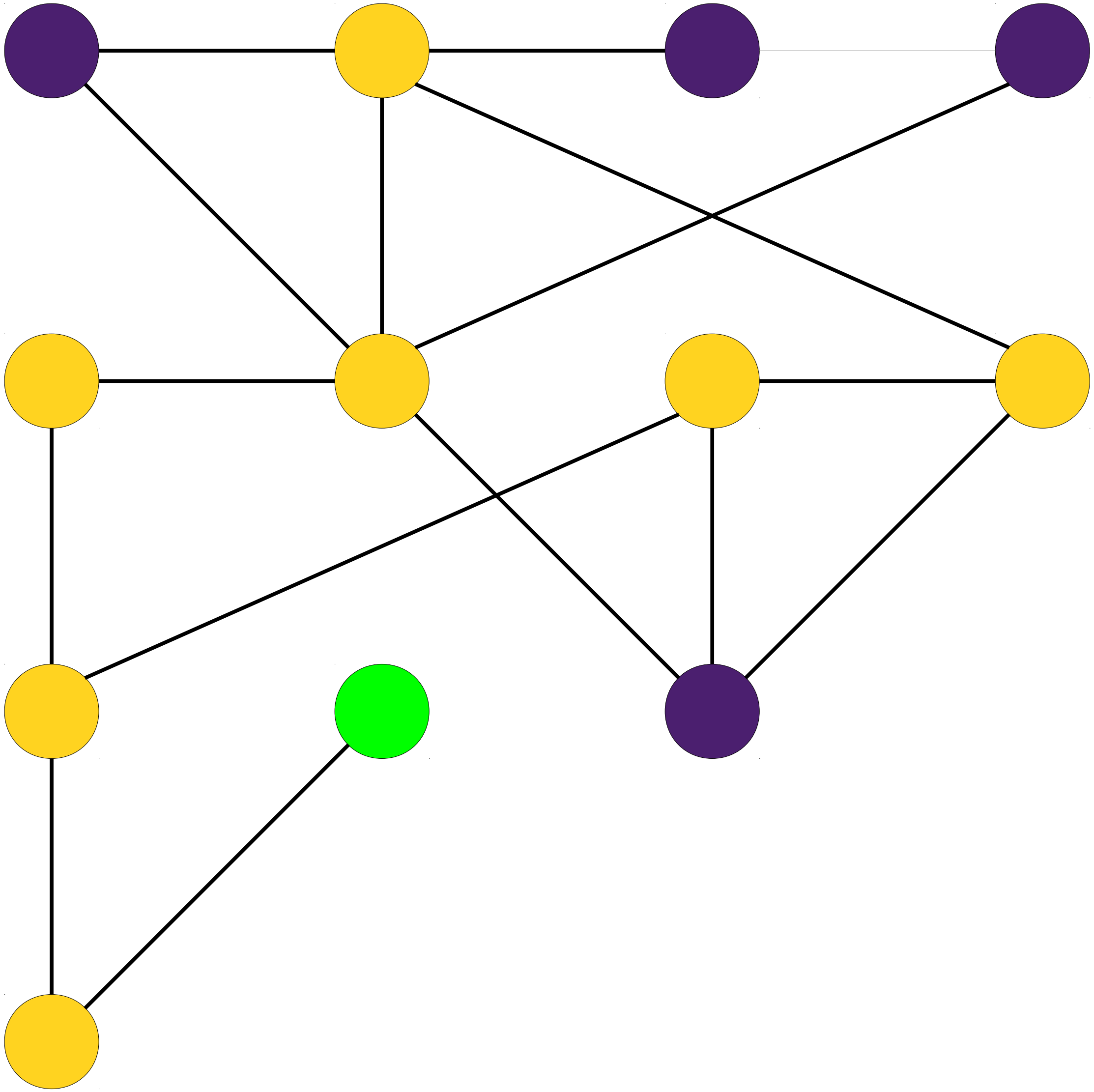
Traversing an Arbitrary Graph



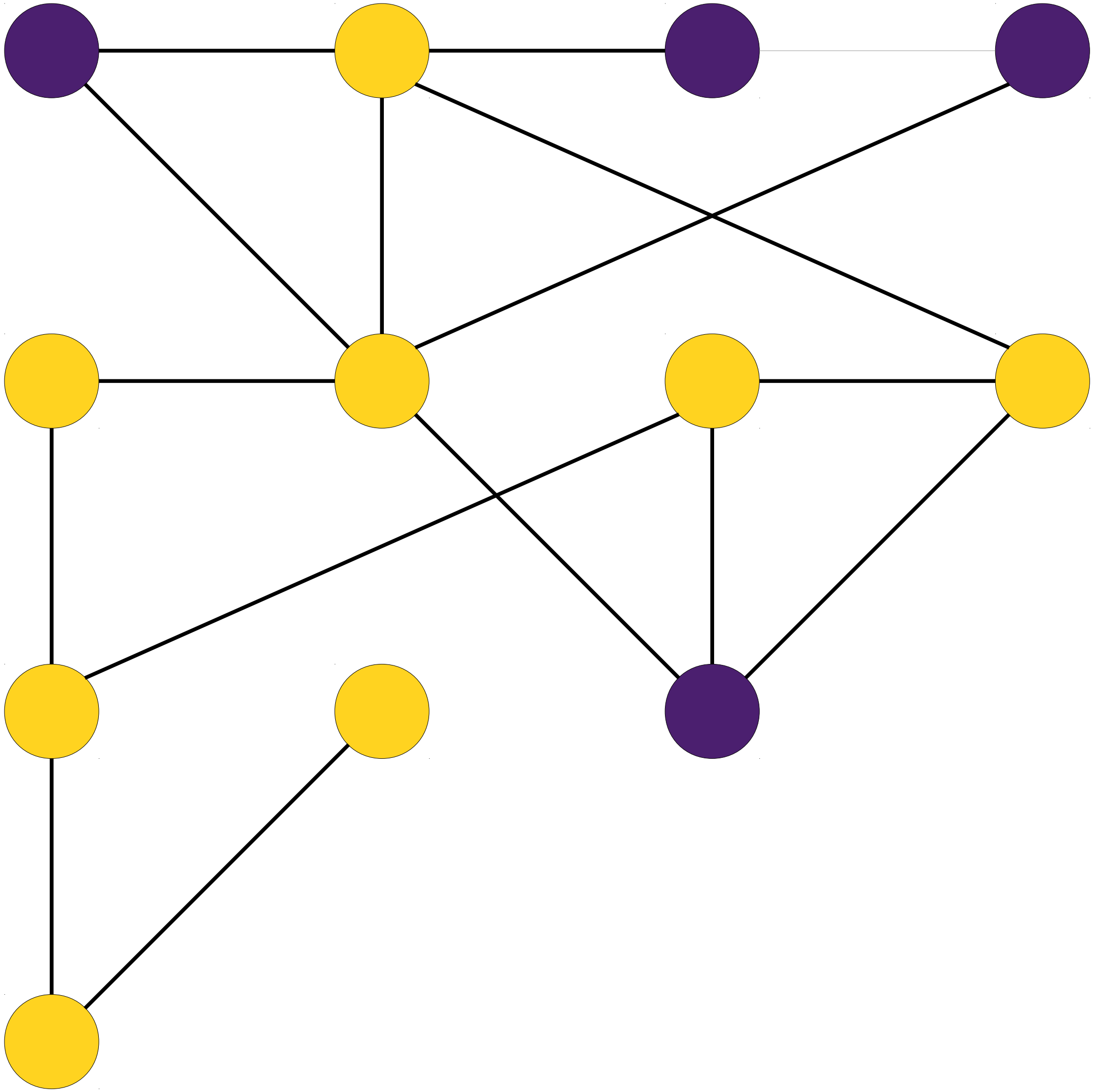
Traversing an Arbitrary Graph



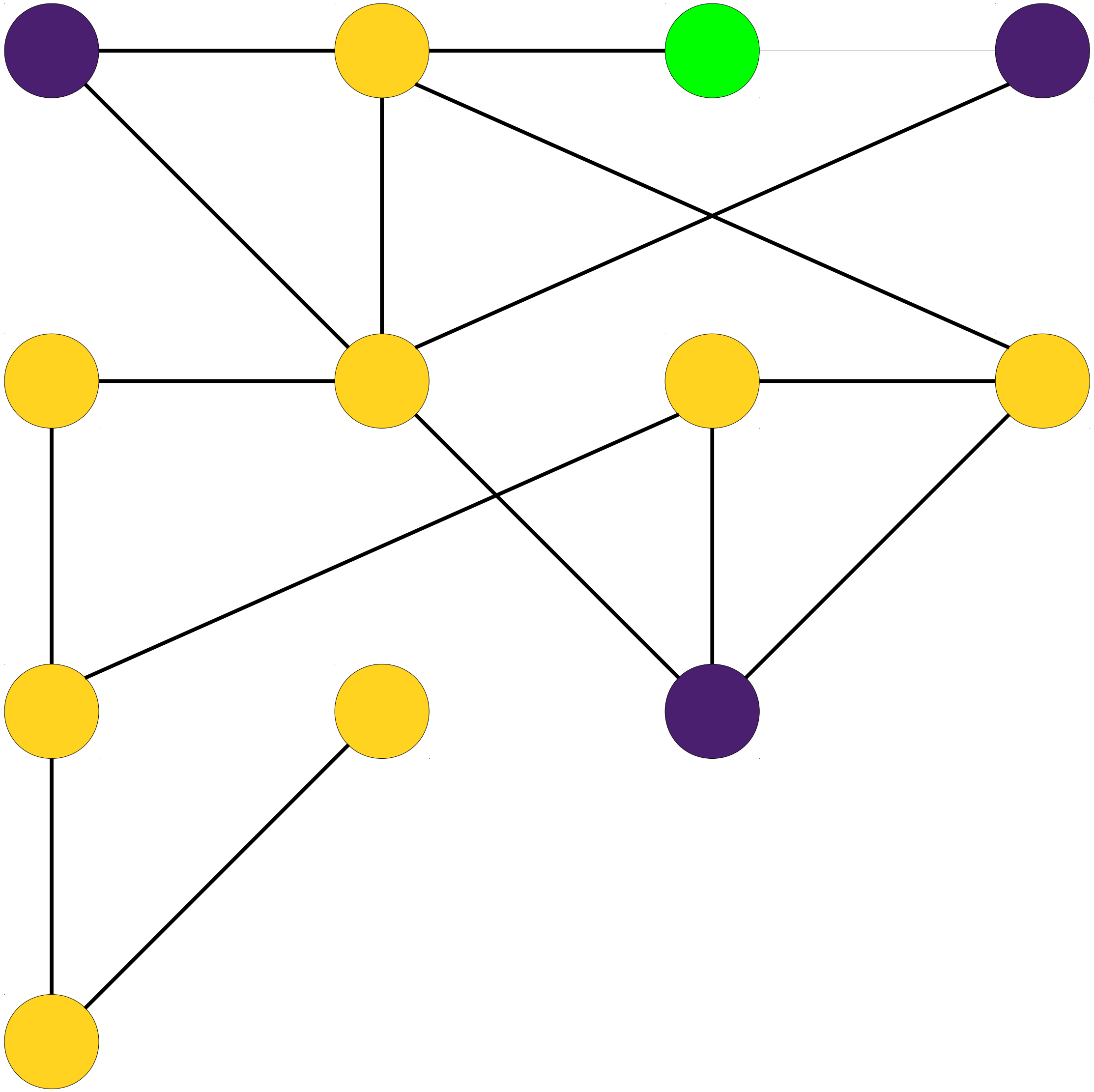
Traversing an Arbitrary Graph



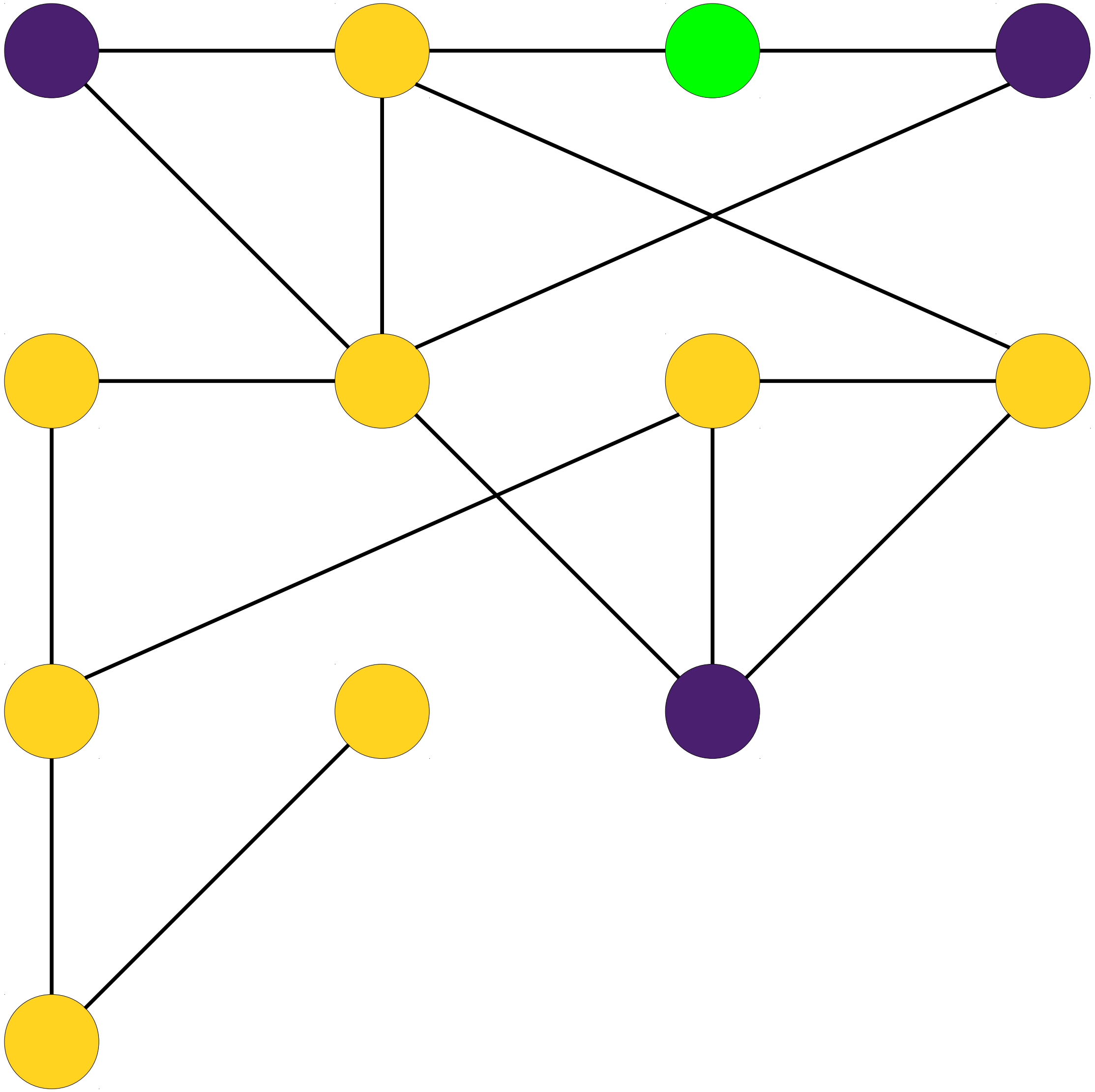
Traversing an Arbitrary Graph



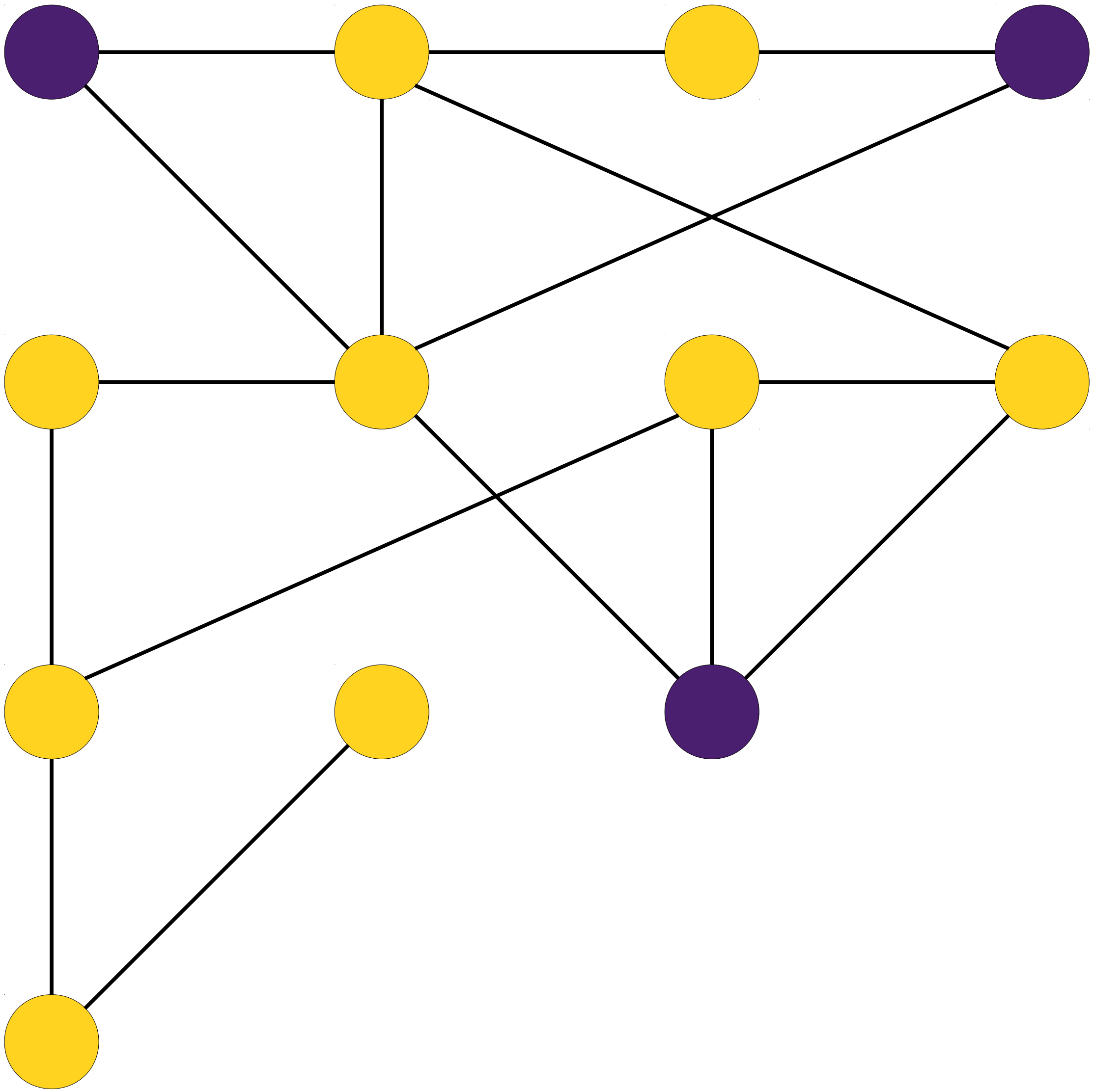
Traversing an Arbitrary Graph



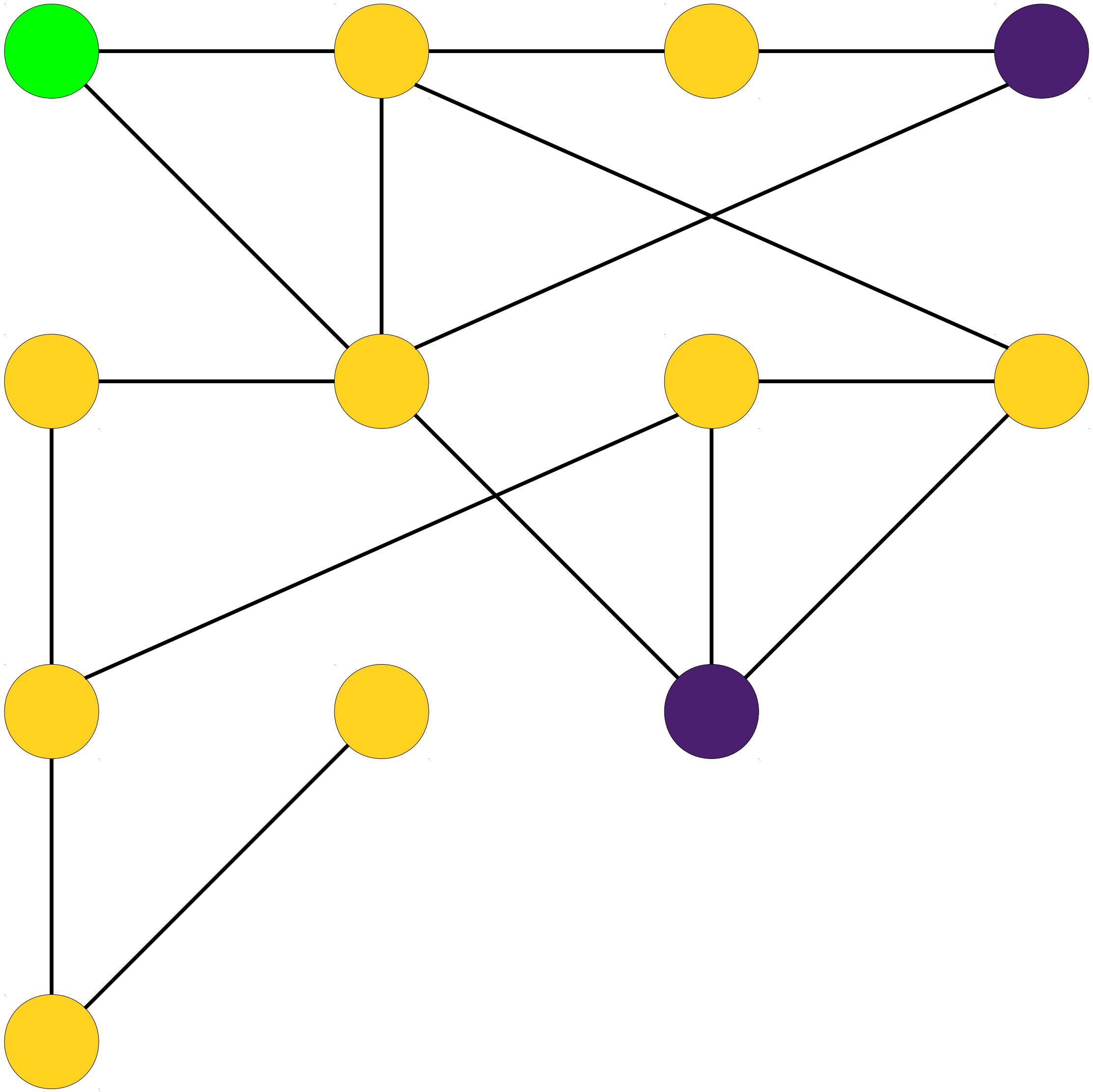
Traversing an Arbitrary Graph



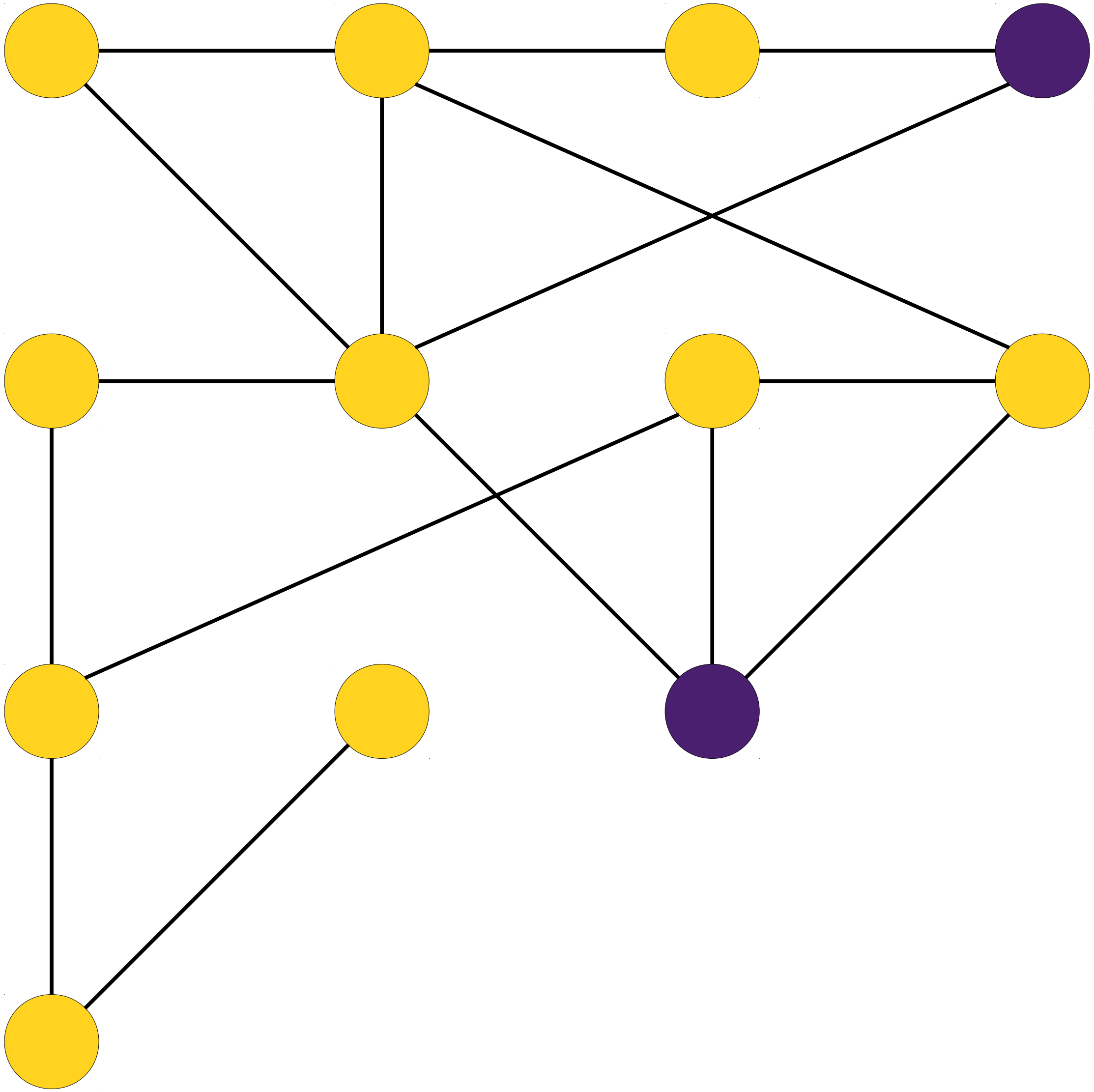
Traversing an Arbitrary Graph



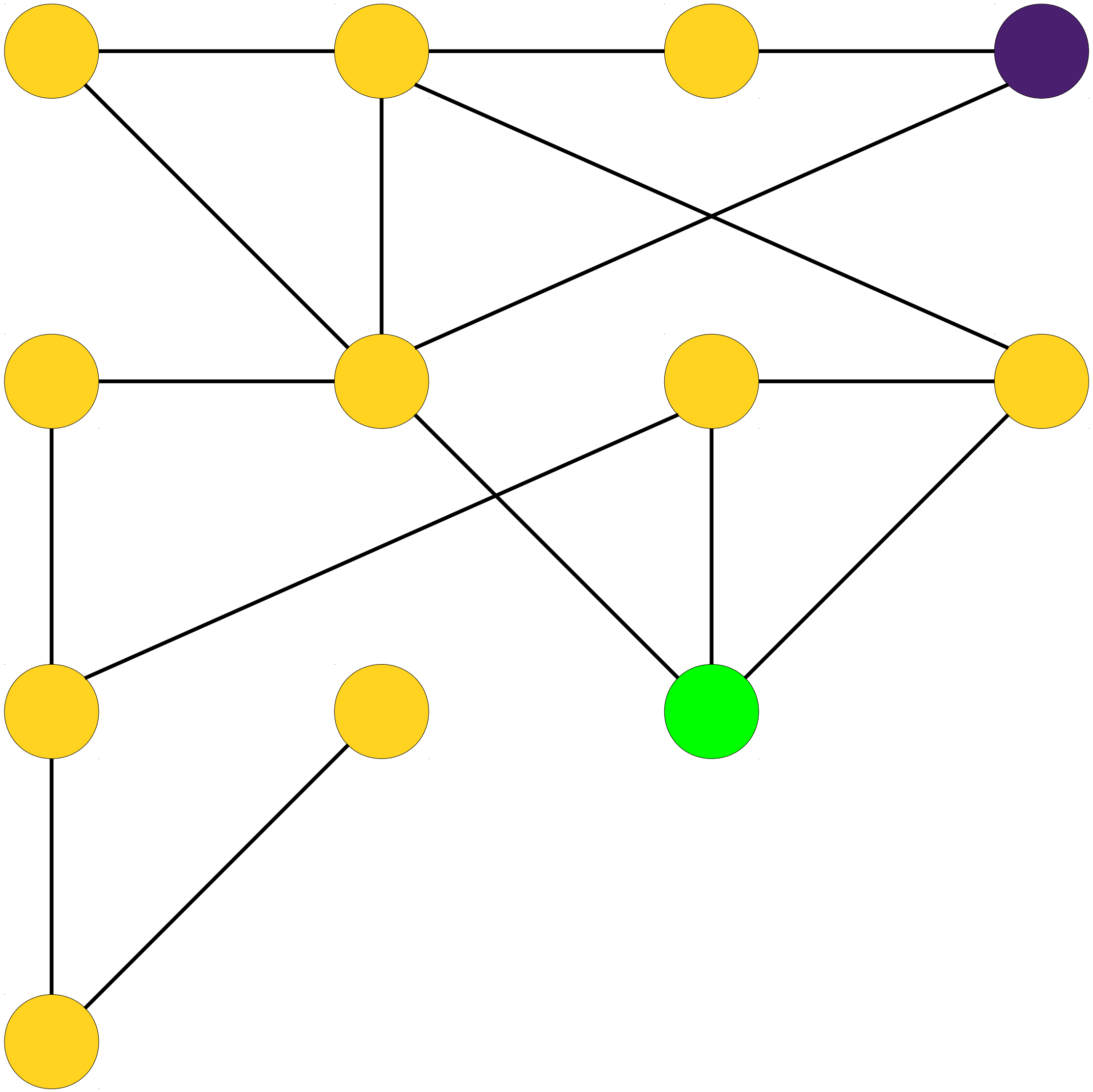
Traversing an Arbitrary Graph



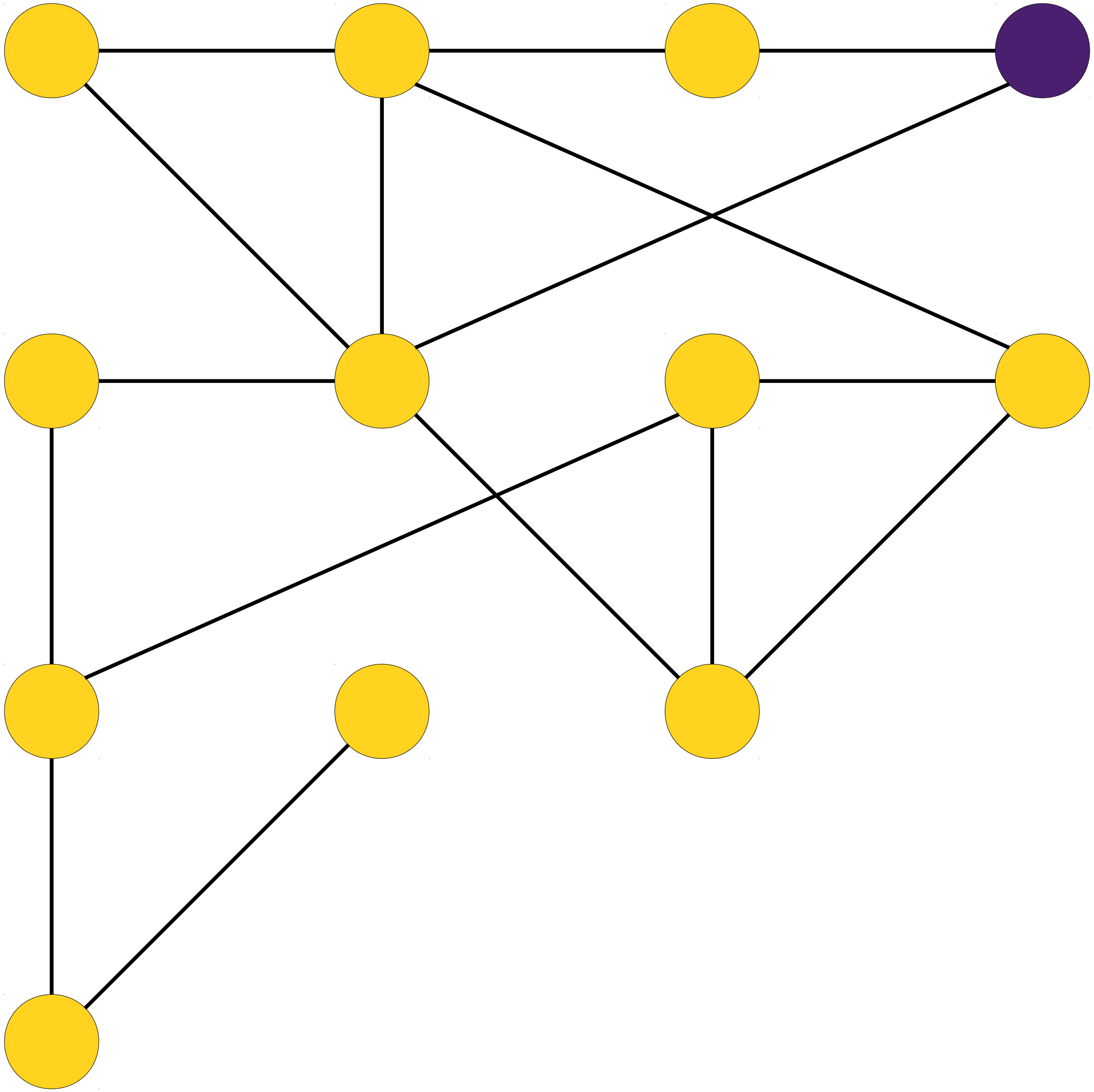
Traversing an Arbitrary Graph



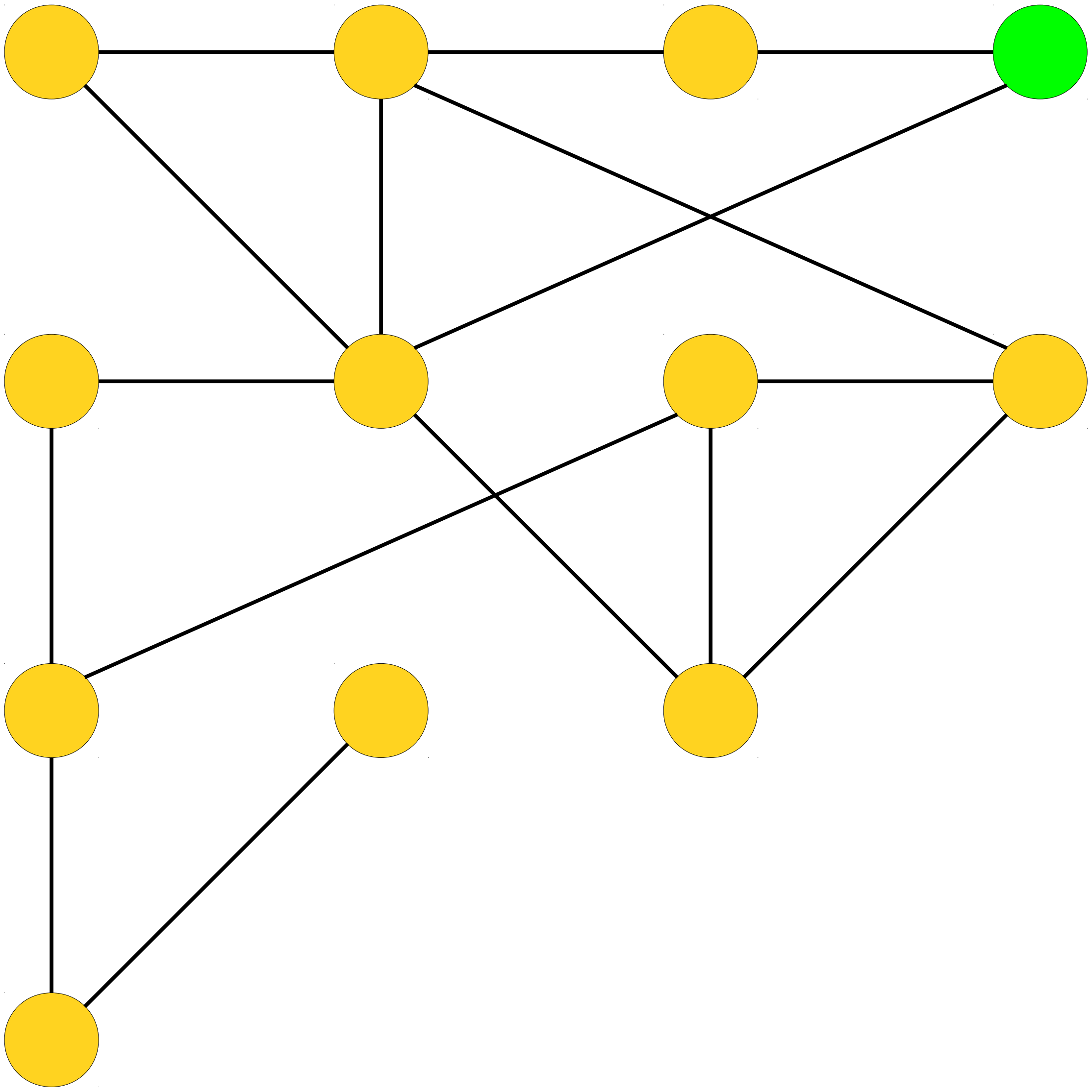
Traversing an Arbitrary Graph



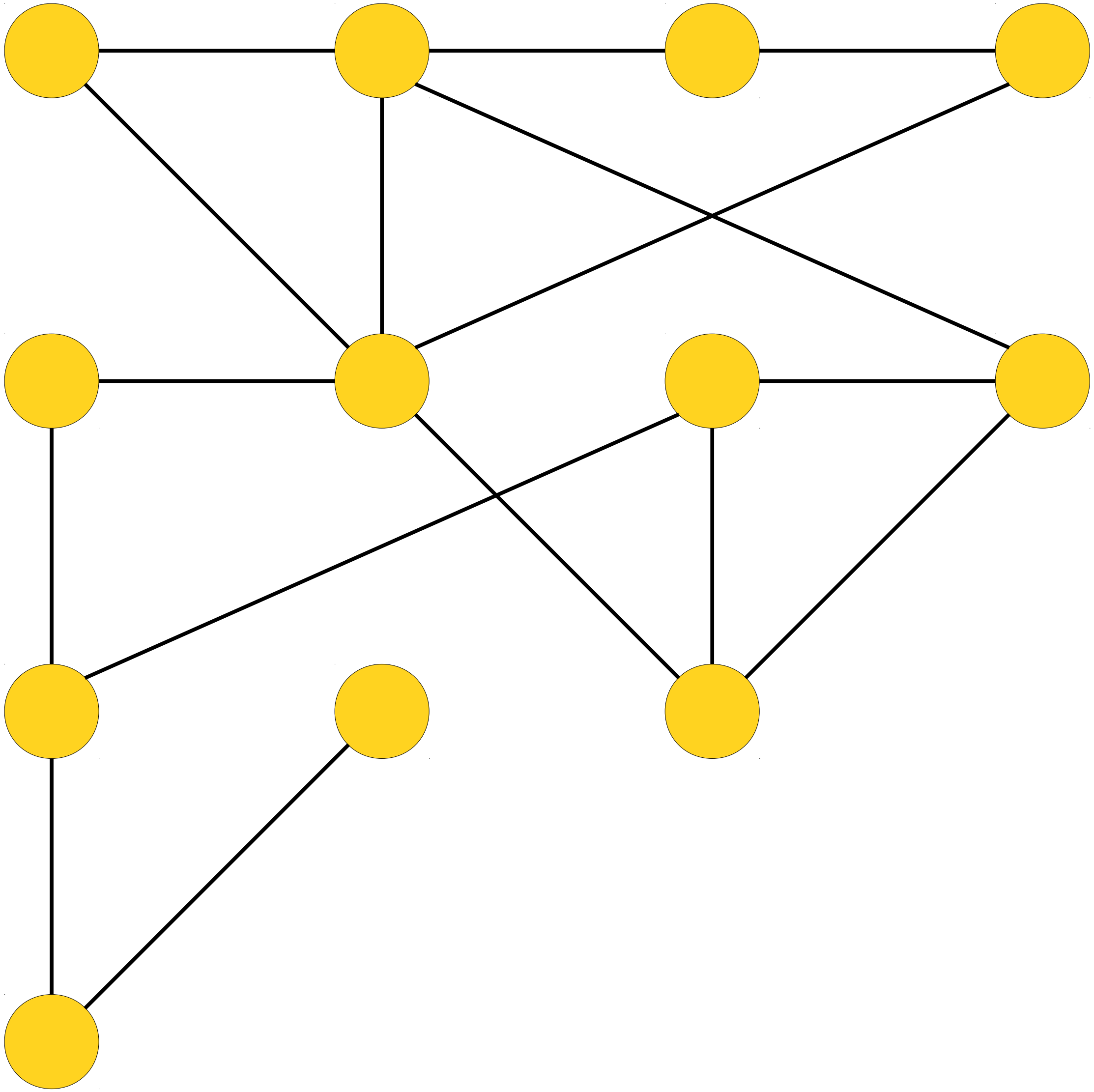
Traversing an Arbitrary Graph



Traversing an Arbitrary Graph



Traversing an Arbitrary Graph



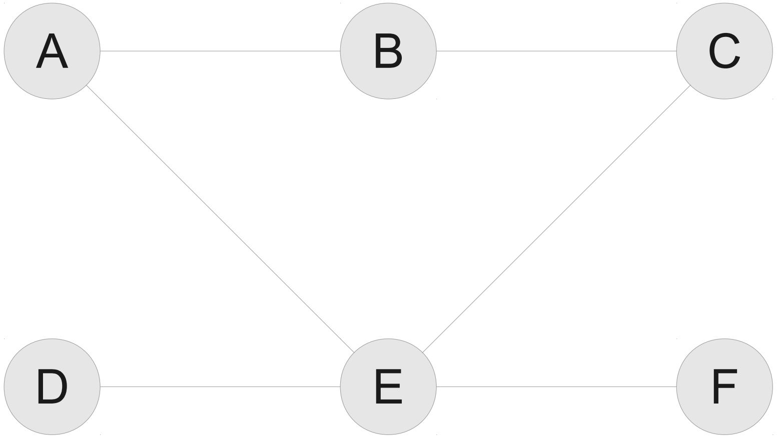
General Graph Search Algorithm

- Maintain a collection C of nodes to visit.
- Initialize C with some set of nodes.
- While C is not empty:
  - Pick a node v out of C.
  - Follow all outgoing edges from v, adding each unvisited node found this way to C.
- Eventually explores all nodes reachable from the starting set of nodes. (Why?)

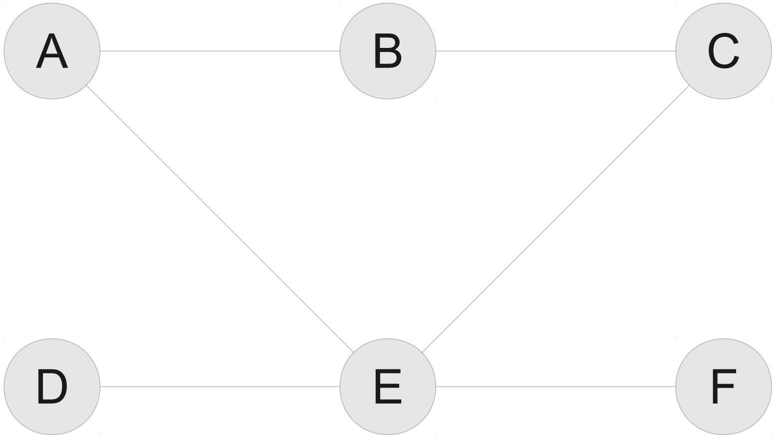
Depth-First Search

- Specialization of the general search algorithm where nodes to visit are put on a **stack**.
- Explores down a path as far as possible, then backs up.
- Simple graph search algorithm useful for exploring a complete graph.
- Useful as a subroutine in many important graph algorithms.
- Runs in $O(m + n)$ with adjacency lists, $O(n^2)$ with adjacency matrix.

Depth-first search

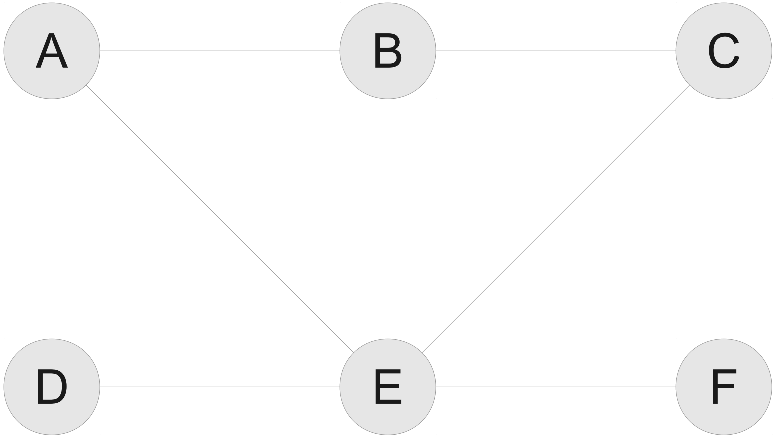


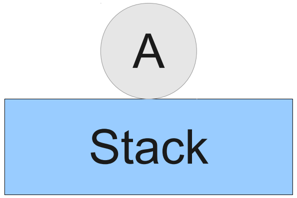
Depth-first search



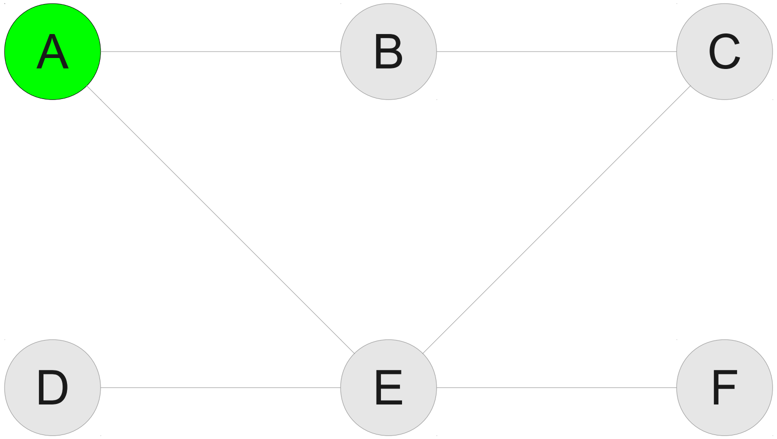
Stack

Depth-first search



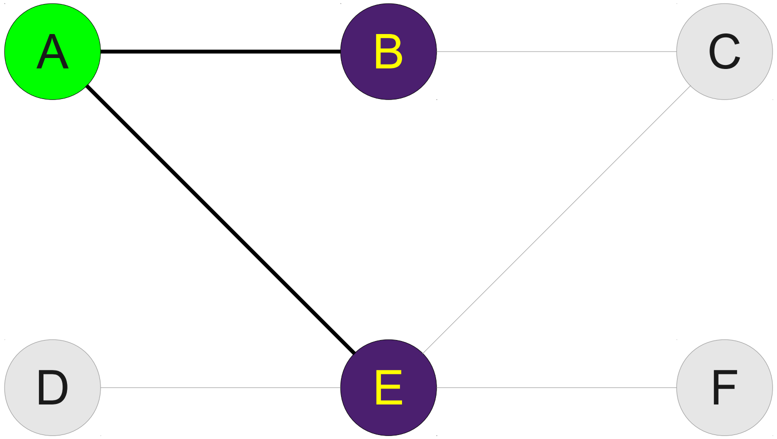


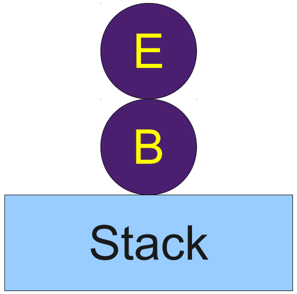
Depth-first search



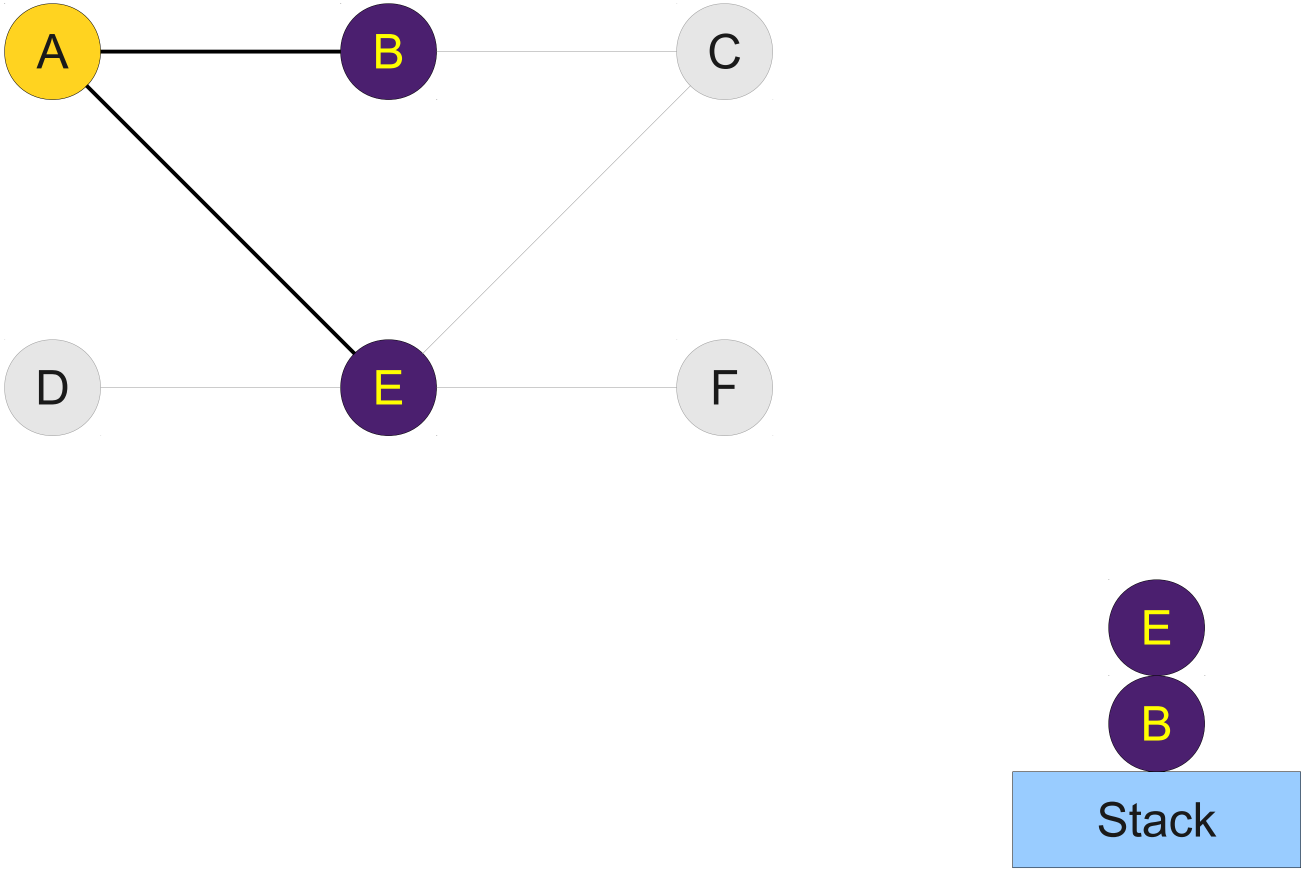
Stack

Depth-first search

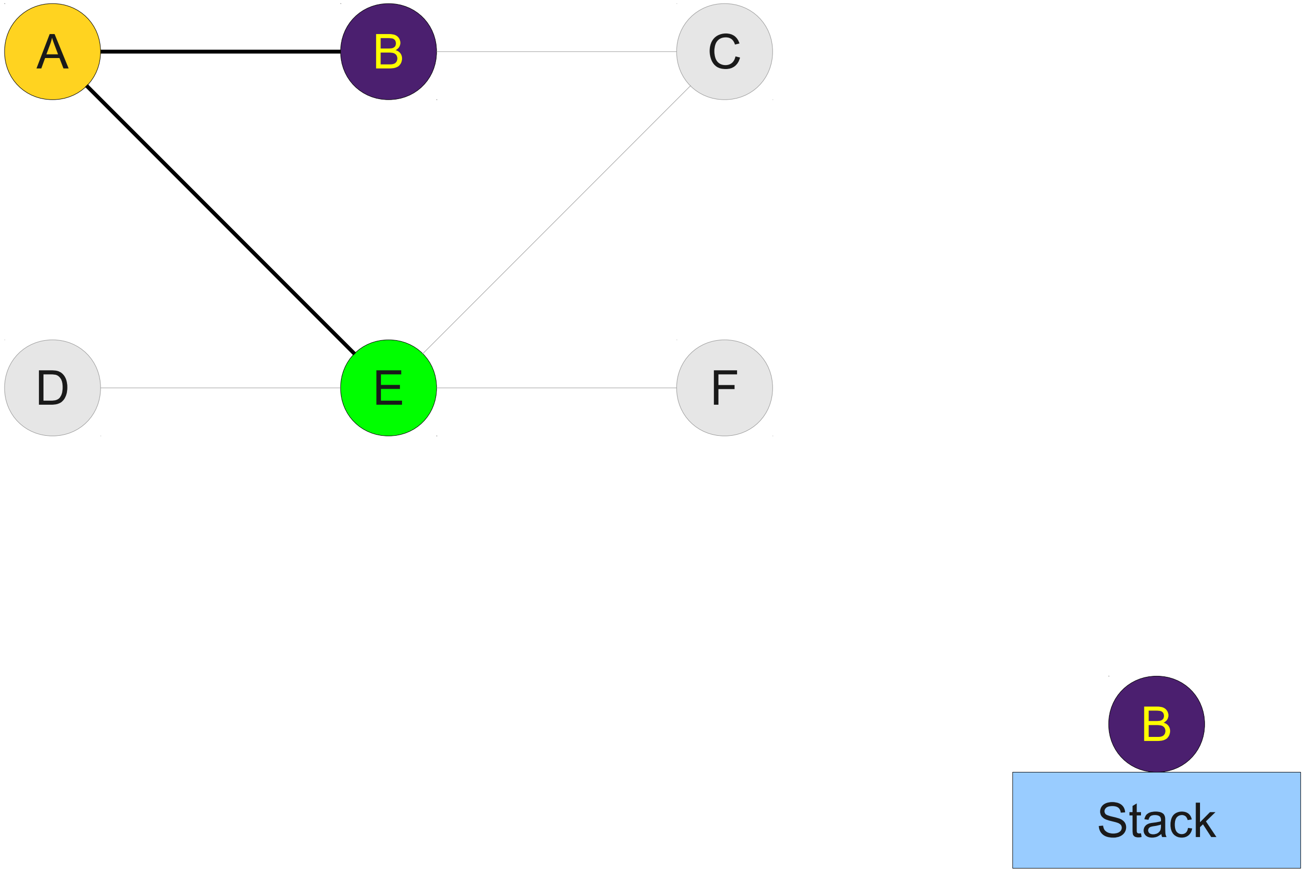




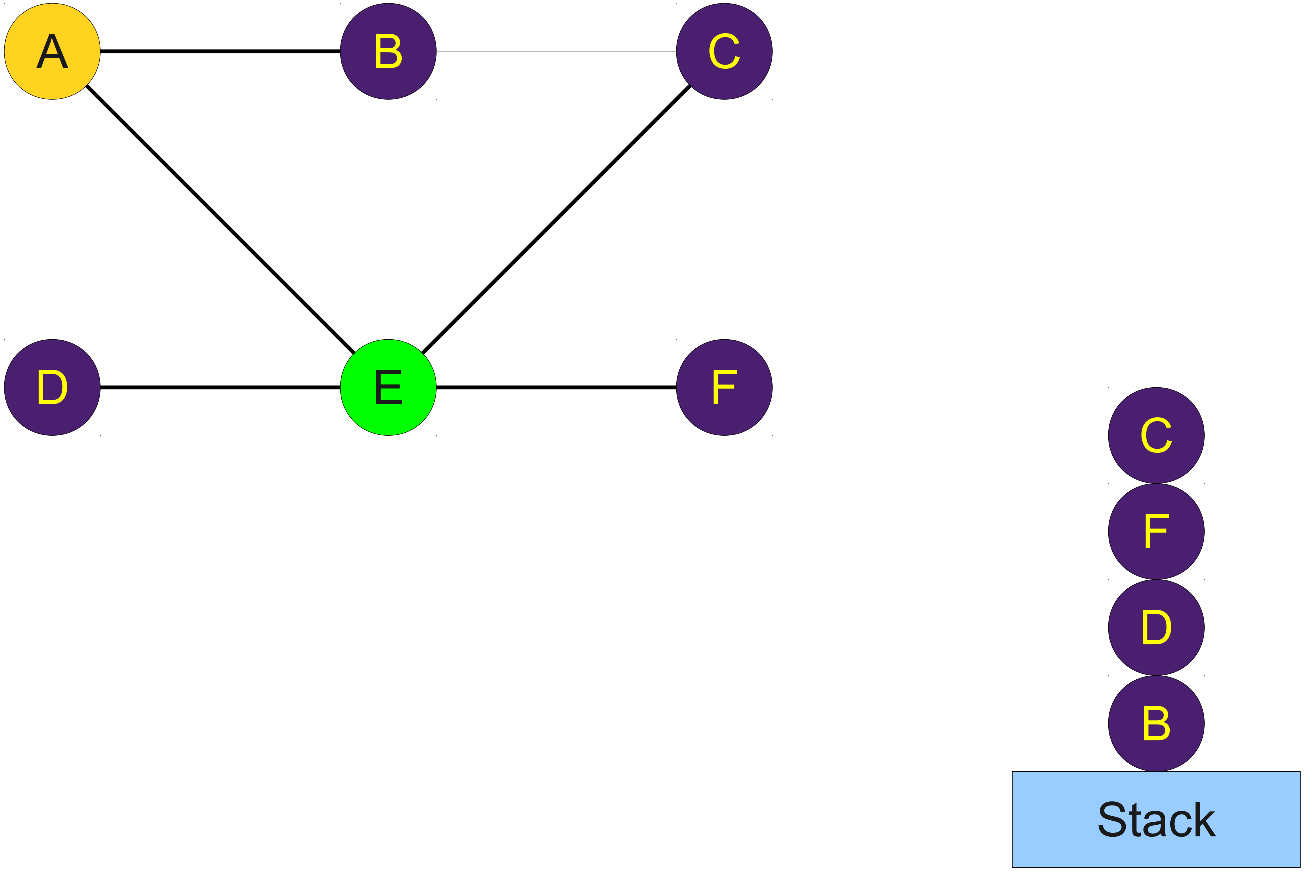
Depth-first search



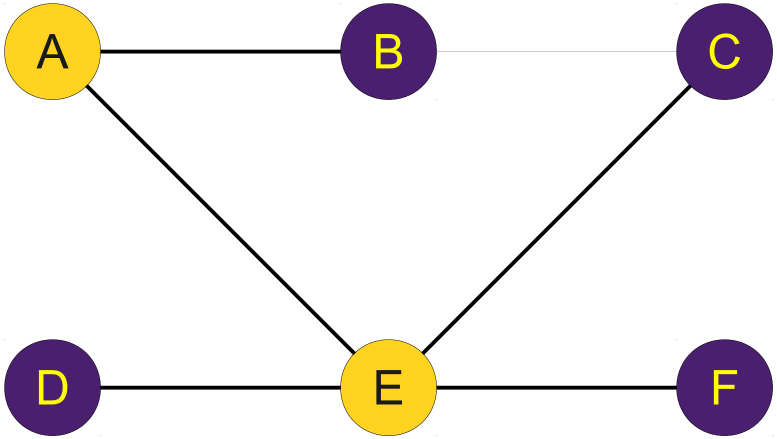
Depth-first search

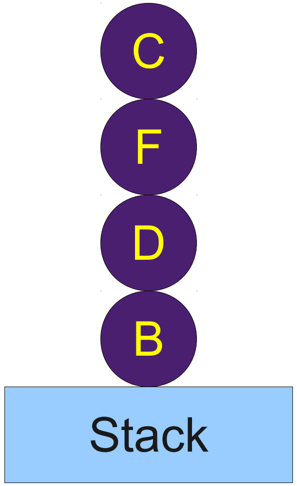


Depth-first search

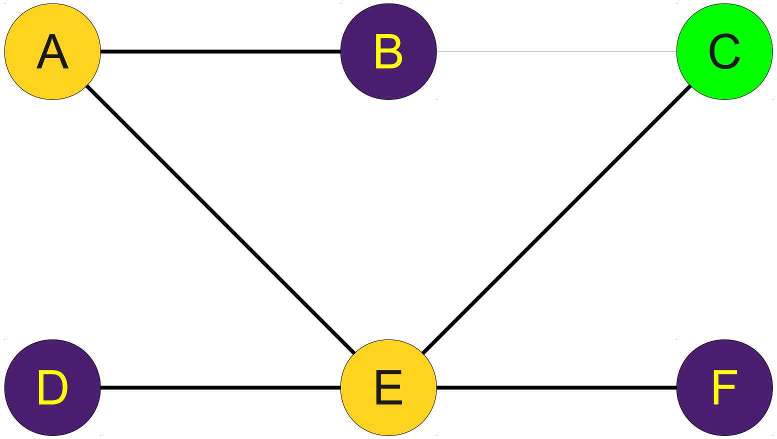


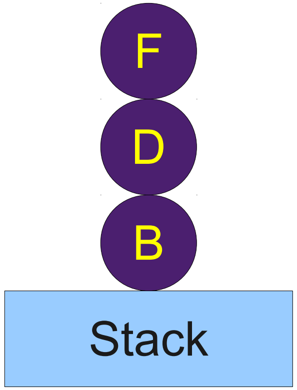
Depth-first search



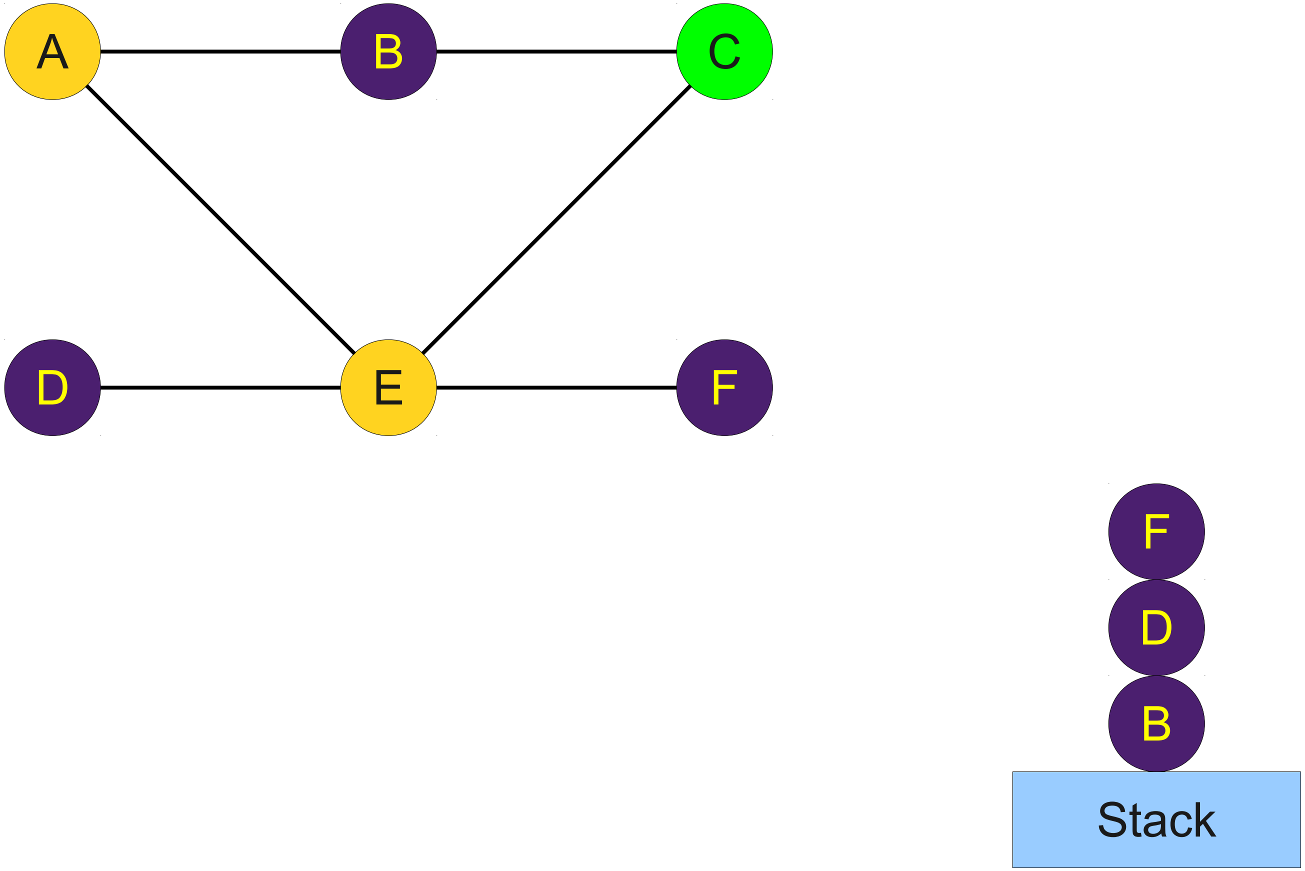


Depth-first search

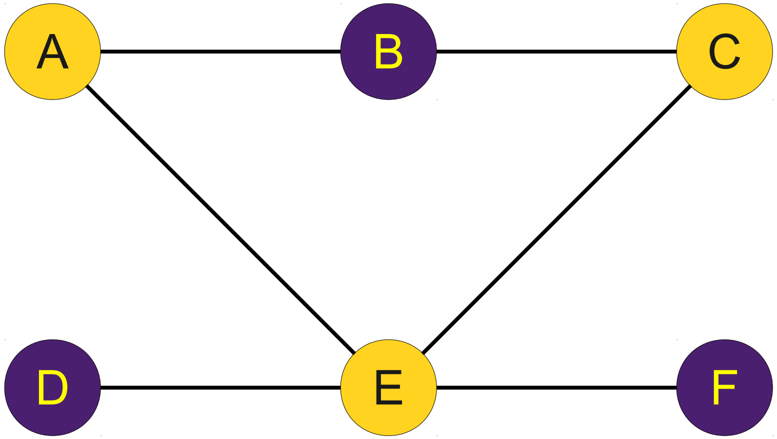


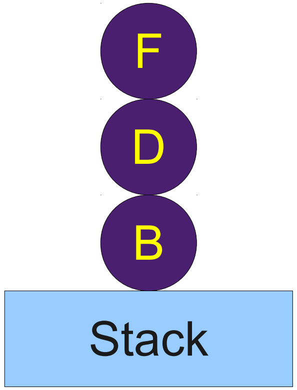


Depth-first search

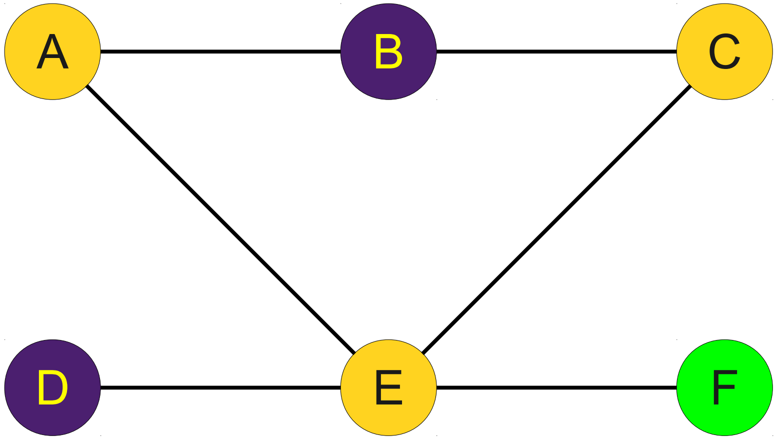


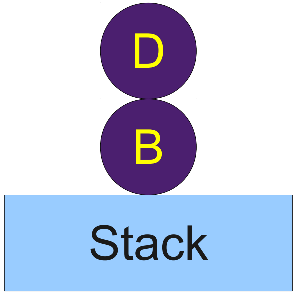
Depth-first search



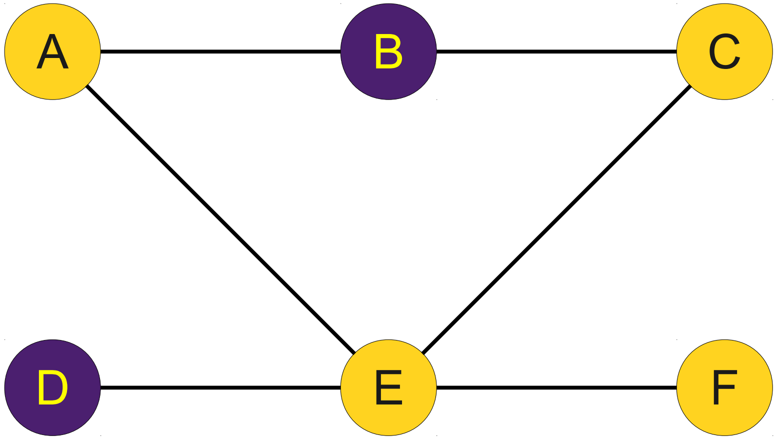


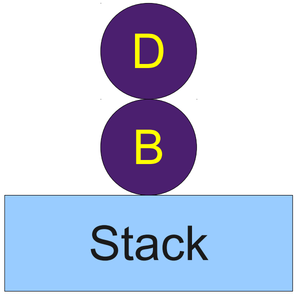
Depth-first search



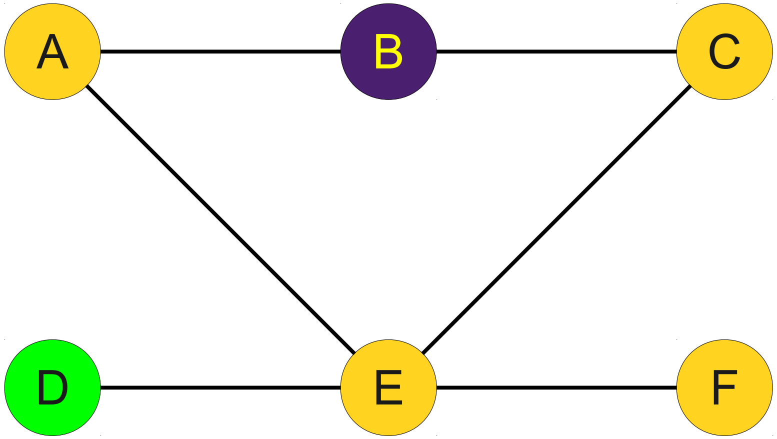


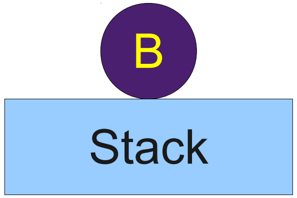
Depth-first search



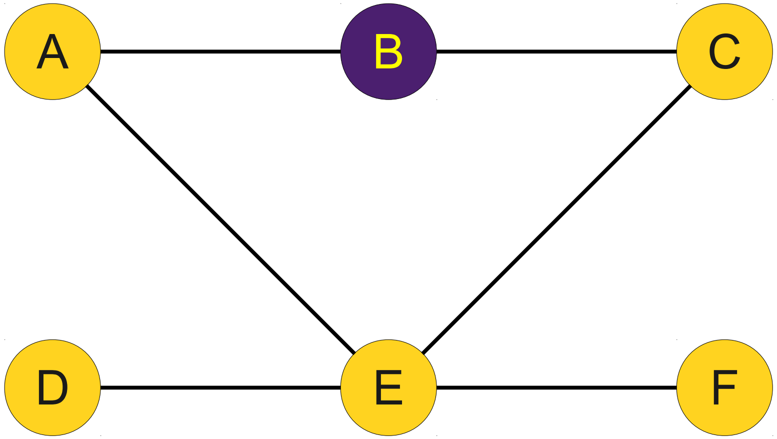


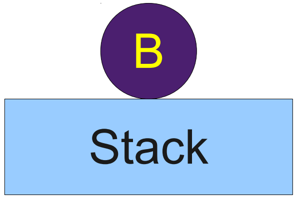
Depth-first search



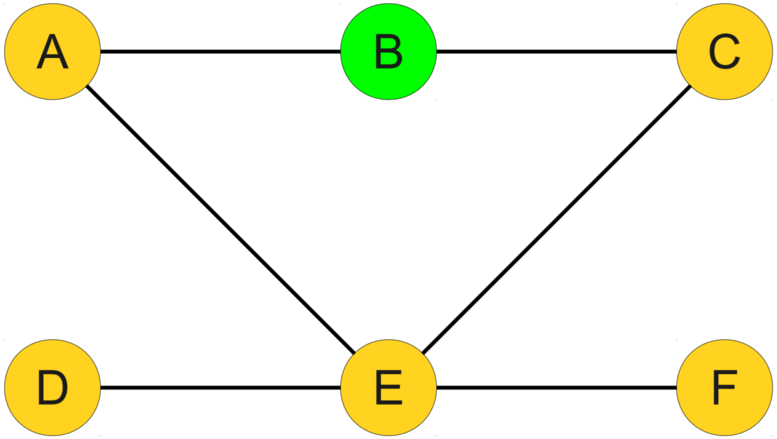


Depth-first search



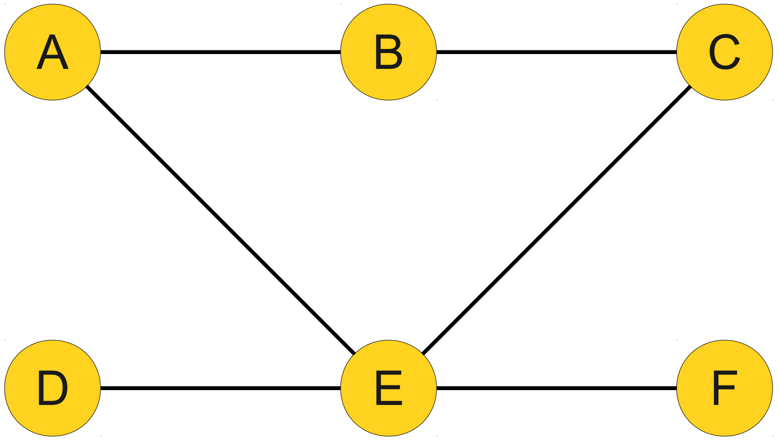


Depth-first search



Stack

Depth-first search

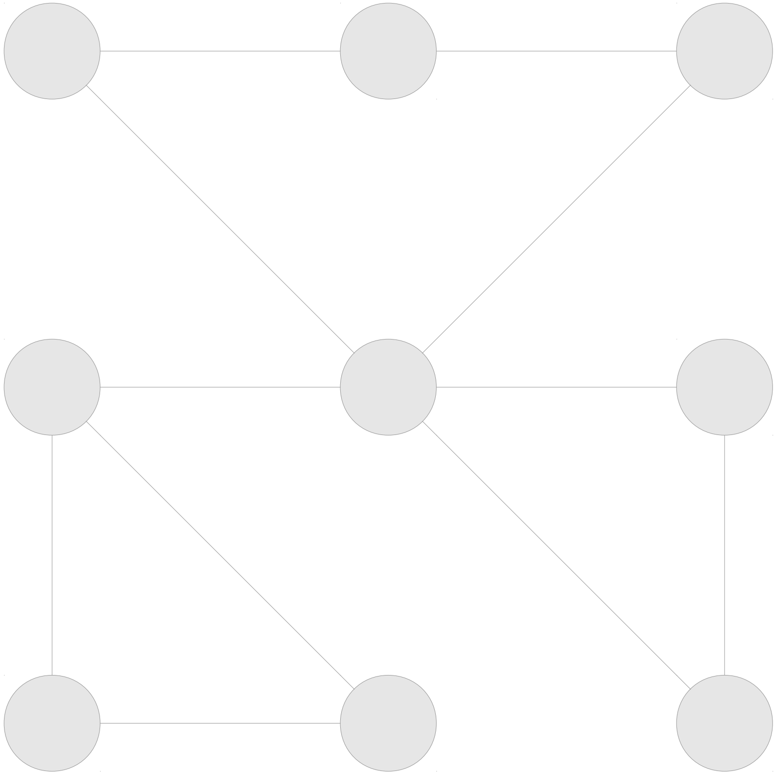


Stack

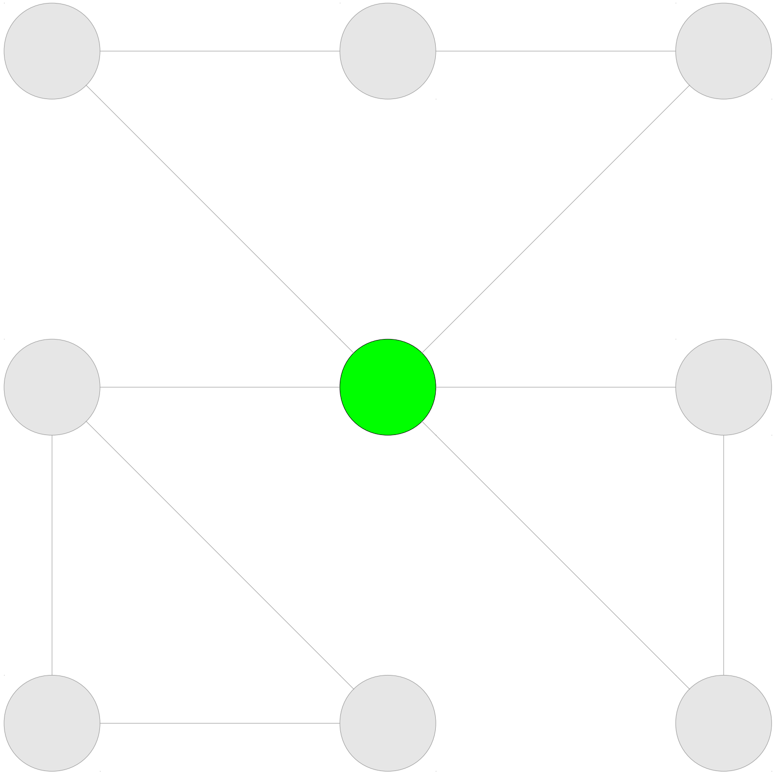
Implementing DFS

```
DFS(Node v, Set<Node> visited) {
    if (v is in visited) return;
    Add v to visited;

    for (Node u connected to v)
        DFS(u, visited);
}
```

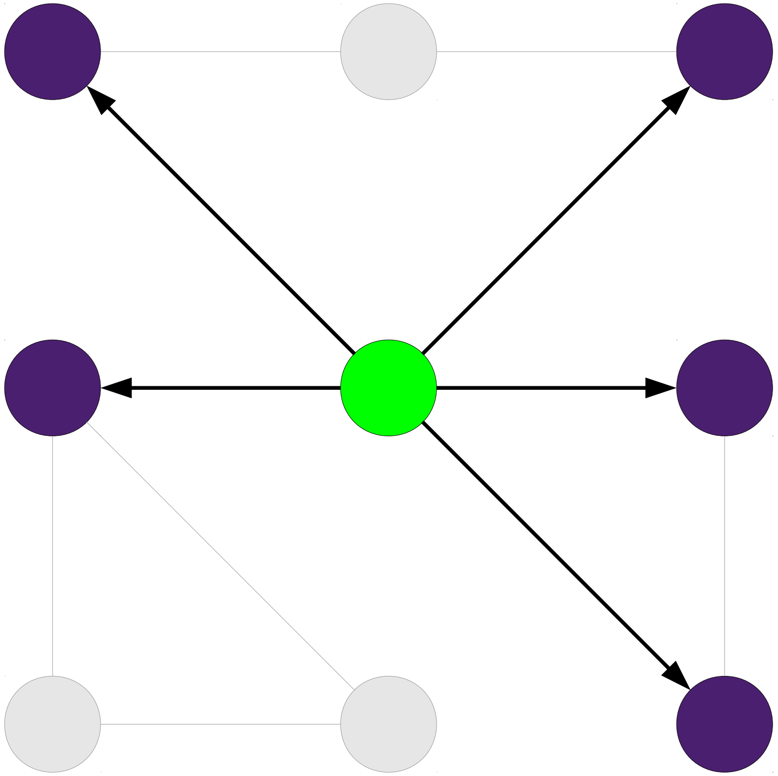
Graph Search Trees



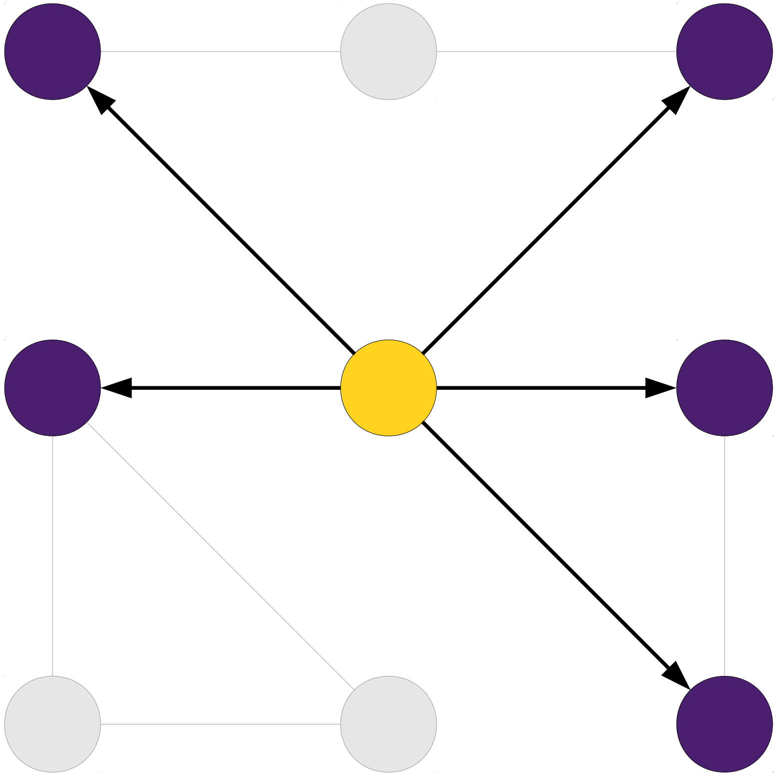
Graph Search Trees



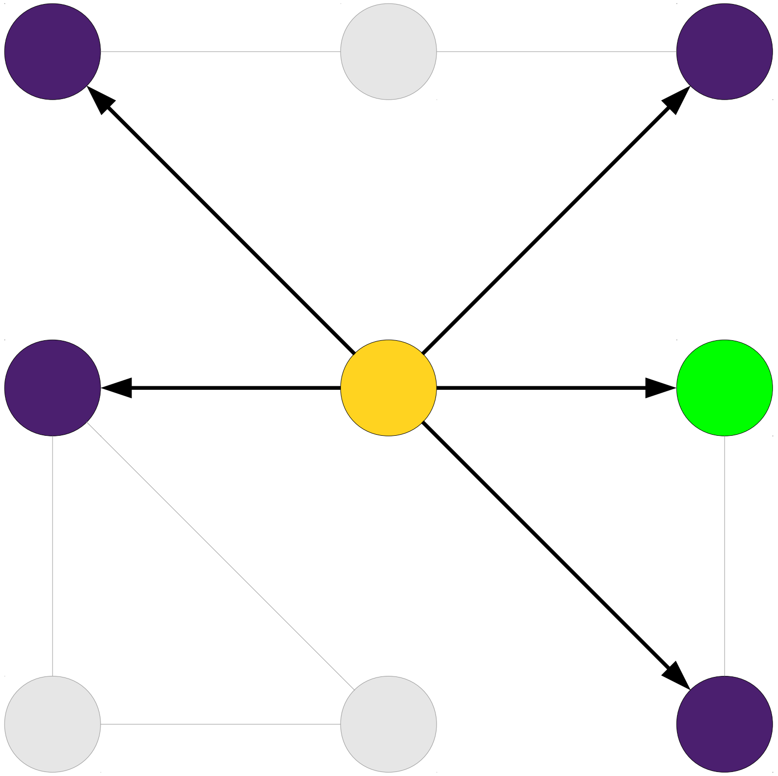
Graph Search Trees



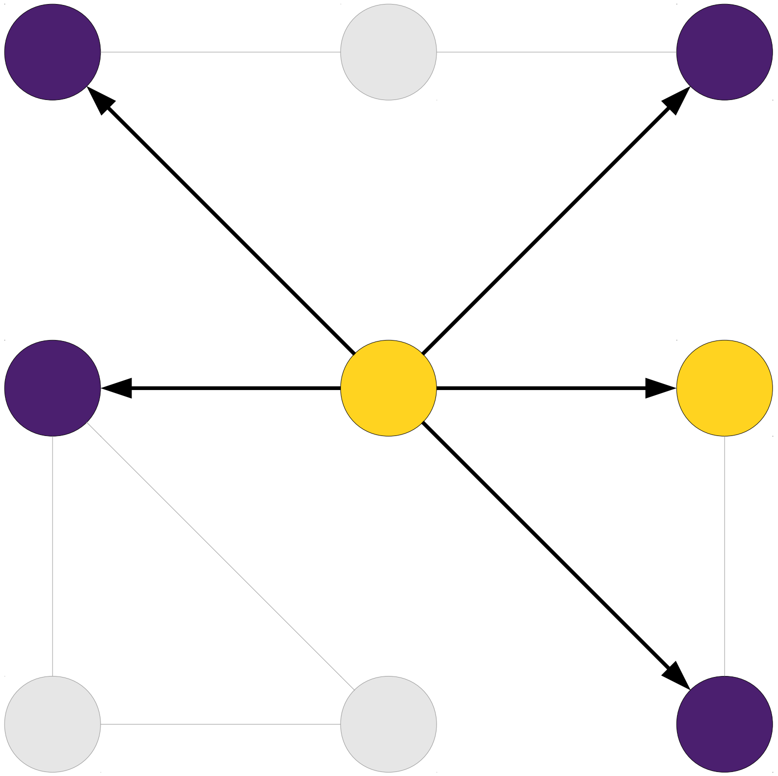
Graph Search Trees



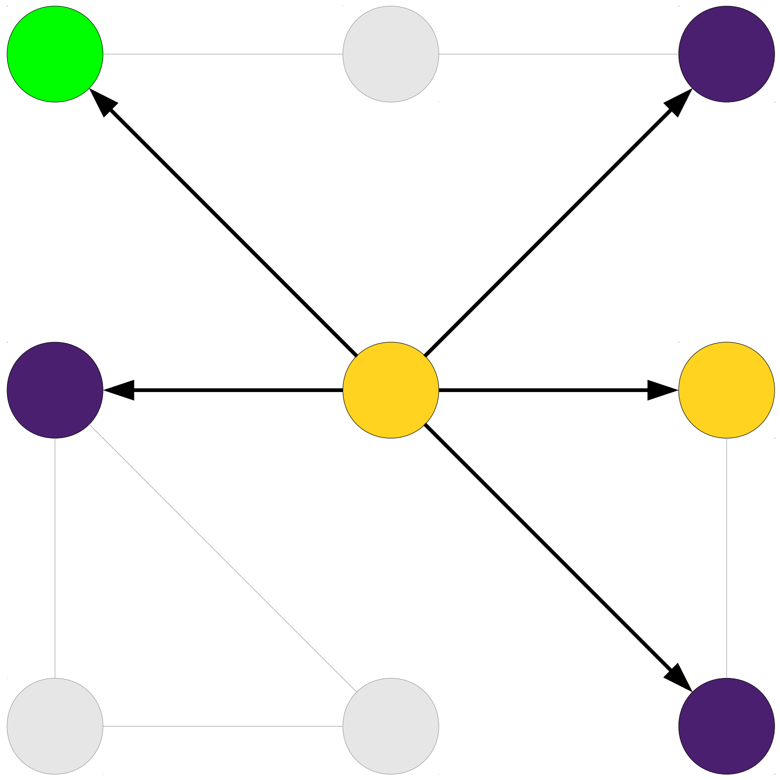
Graph Search Trees



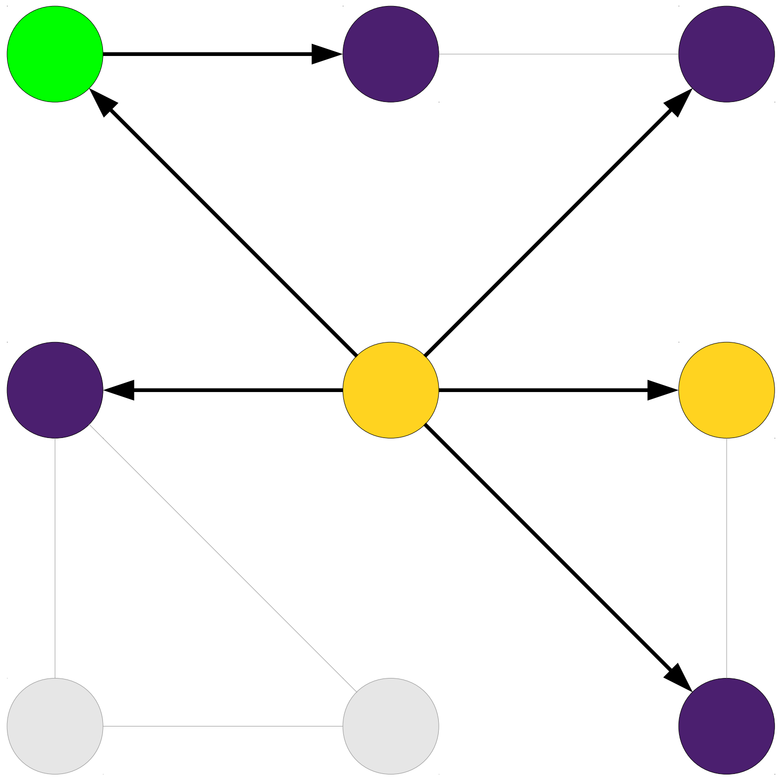
Graph Search Trees



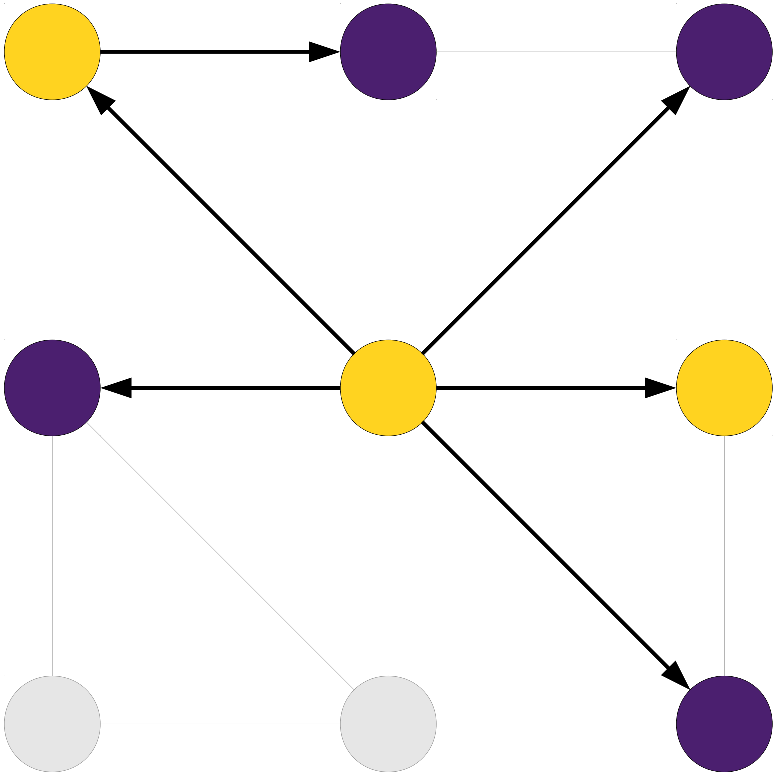
Graph Search Trees



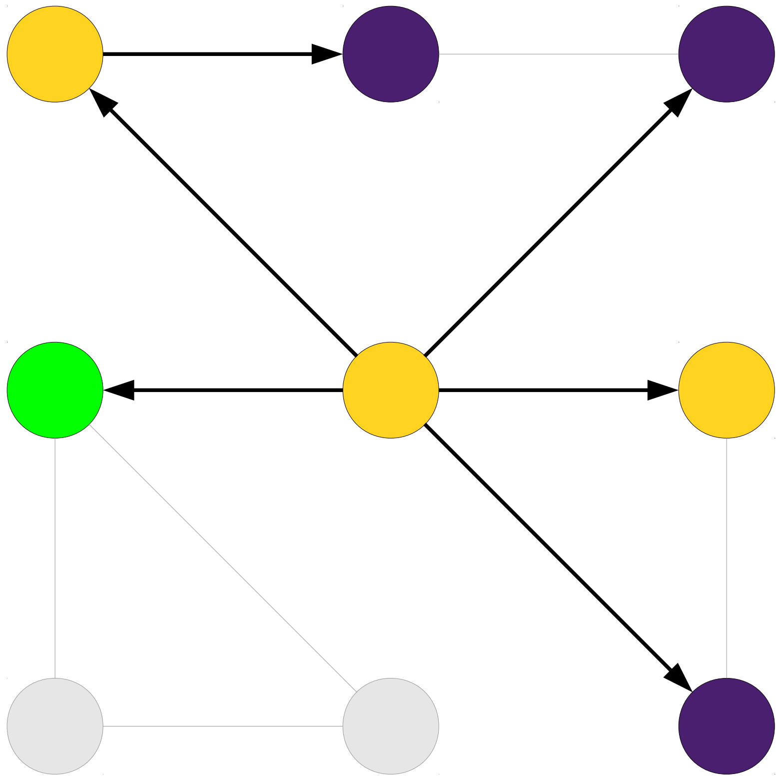
Graph Search Trees



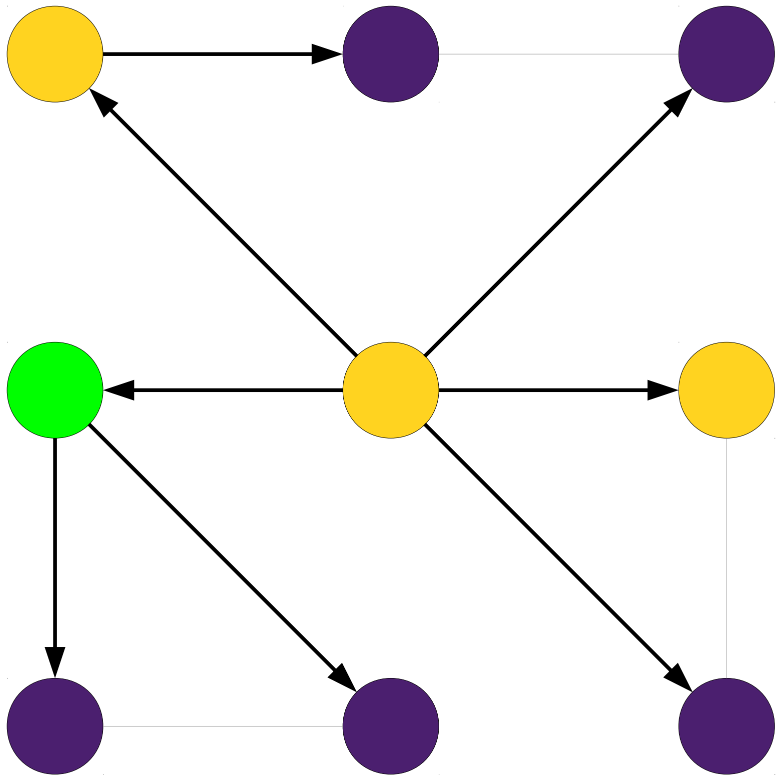
Graph Search Trees



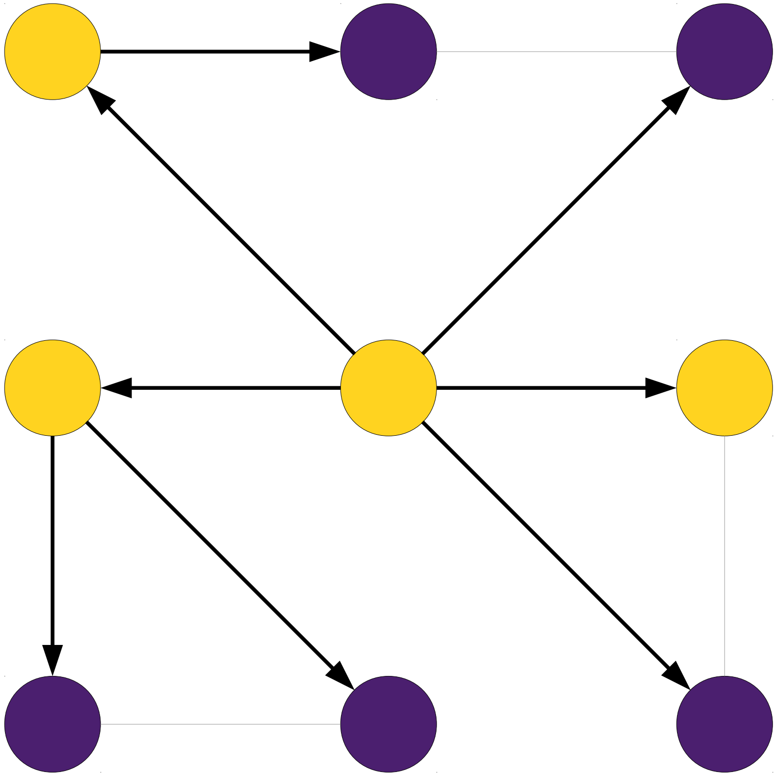
Graph Search Trees



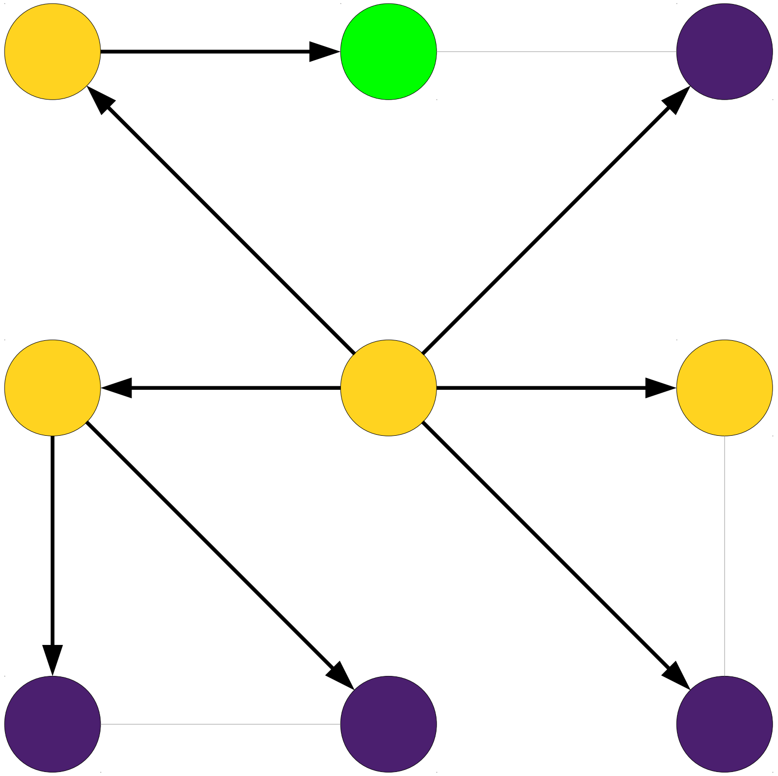
Graph Search Trees



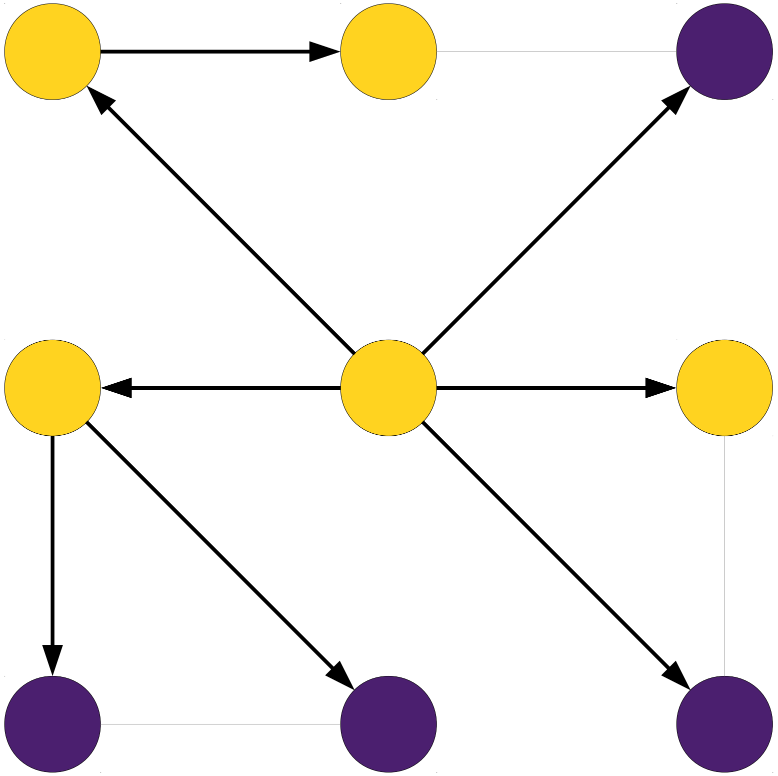
Graph Search Trees



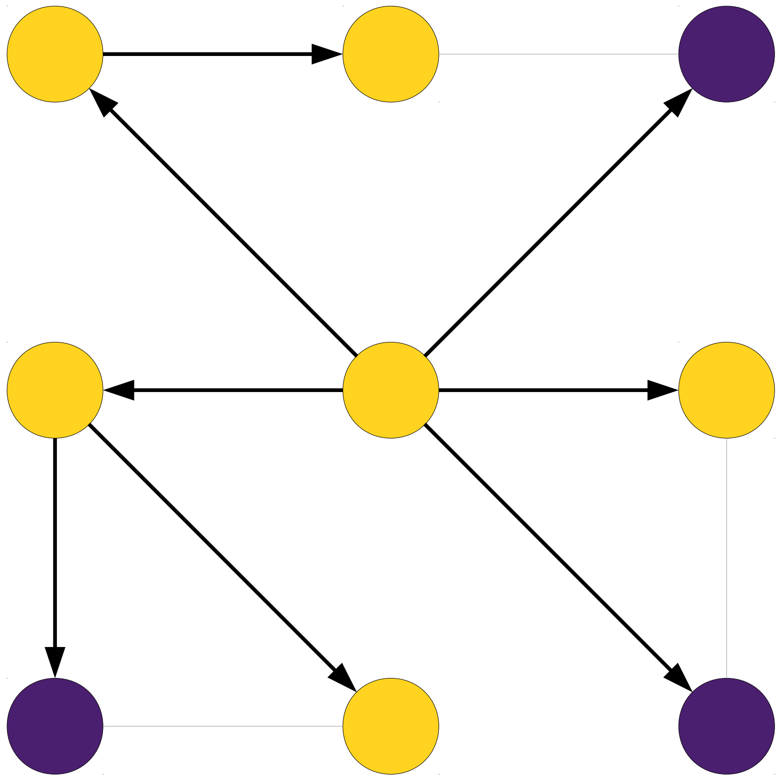
Graph Search Trees



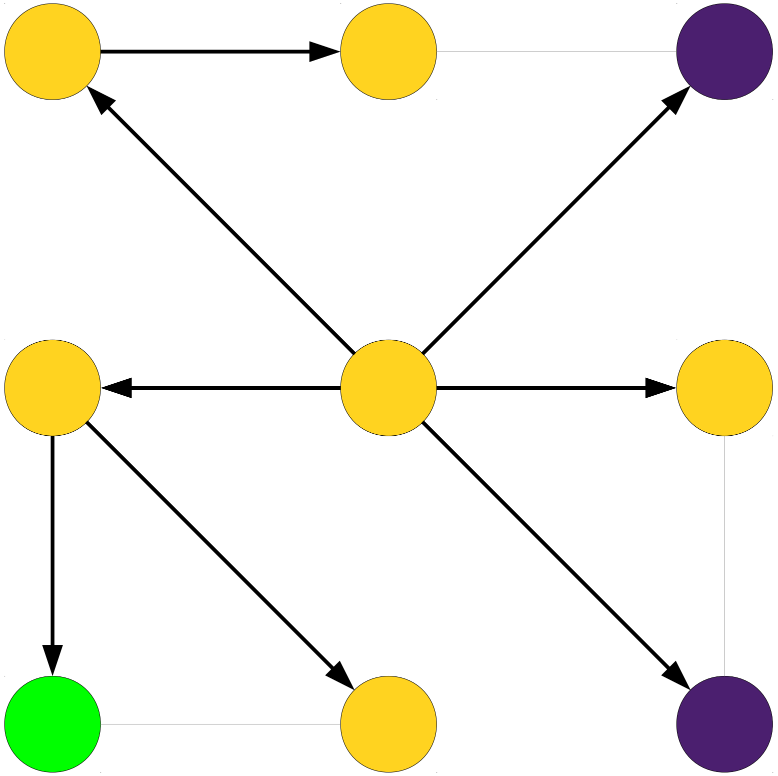
Graph Search Trees



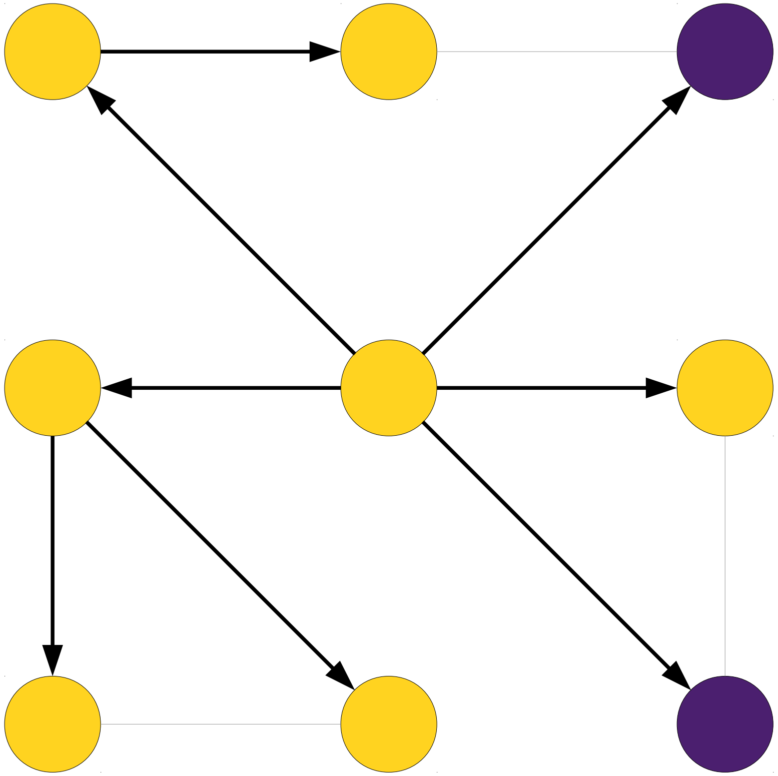
Graph Search Trees



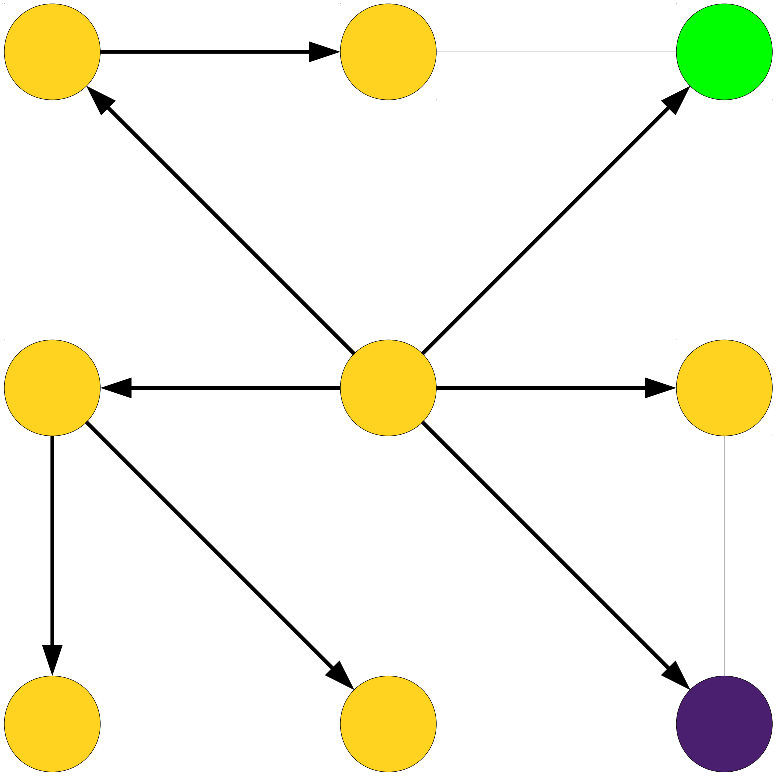
Graph Search Trees



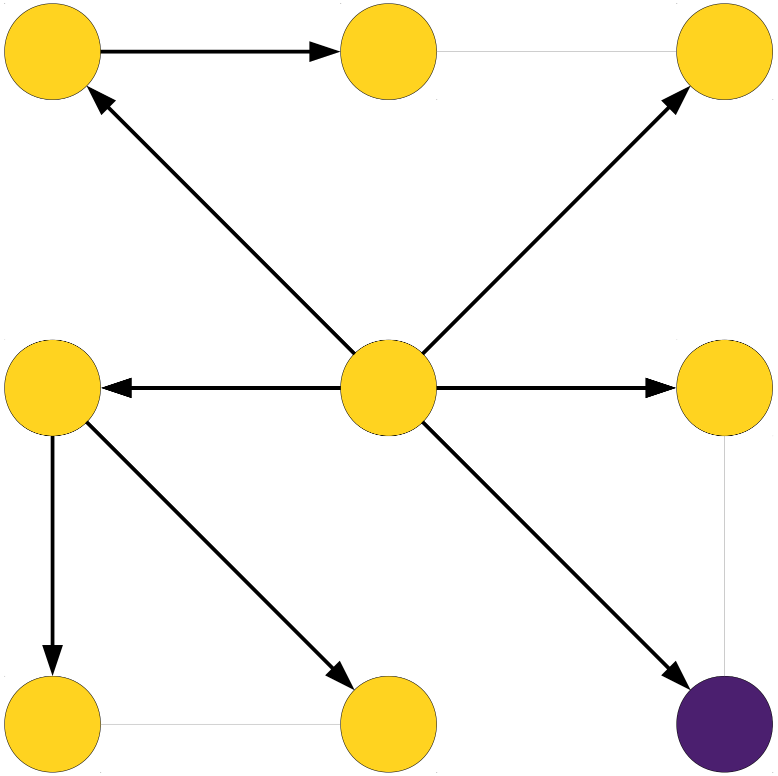
Graph Search Trees



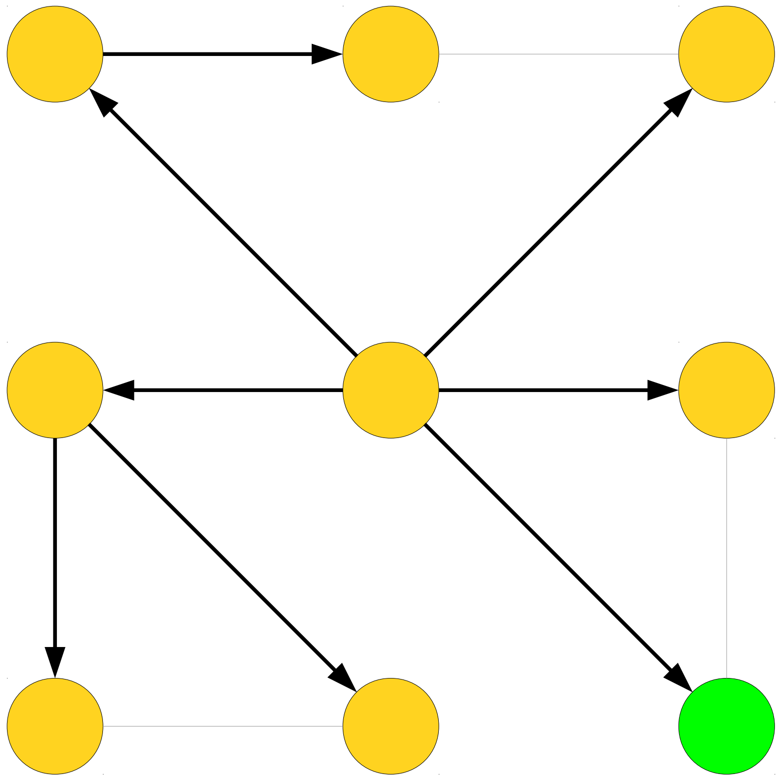
Graph Search Trees



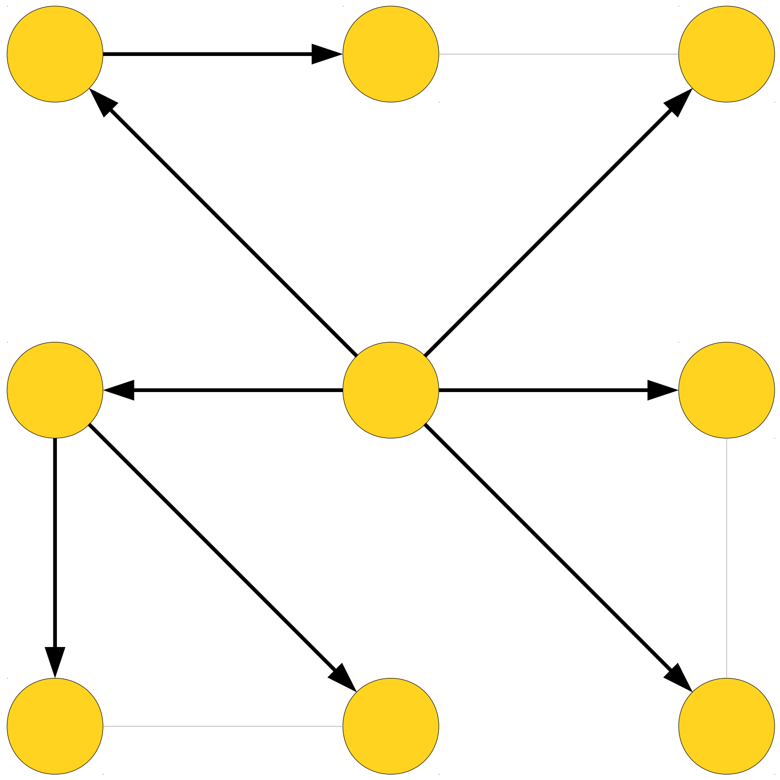
Graph Search Trees



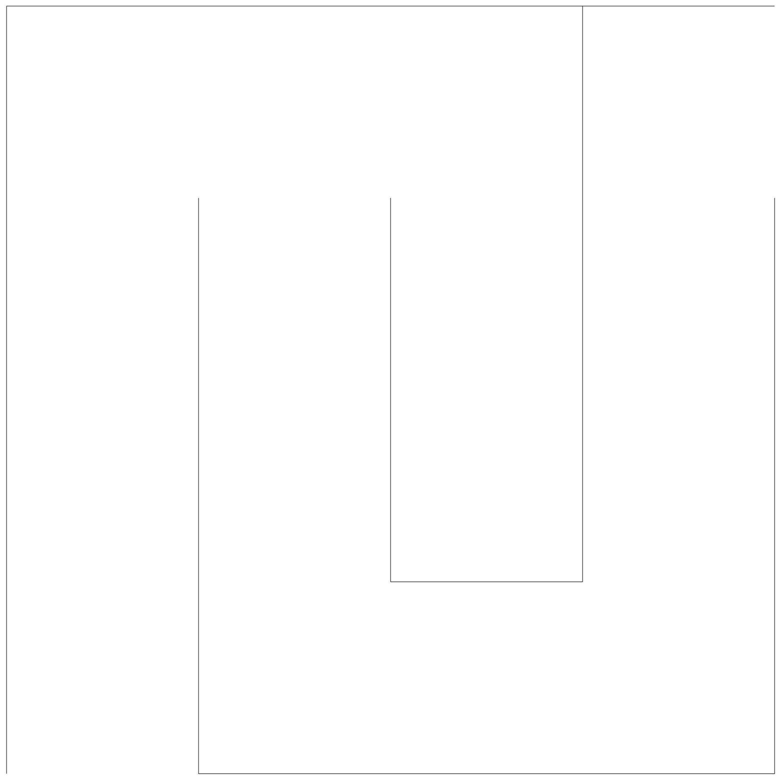
Graph Search Trees



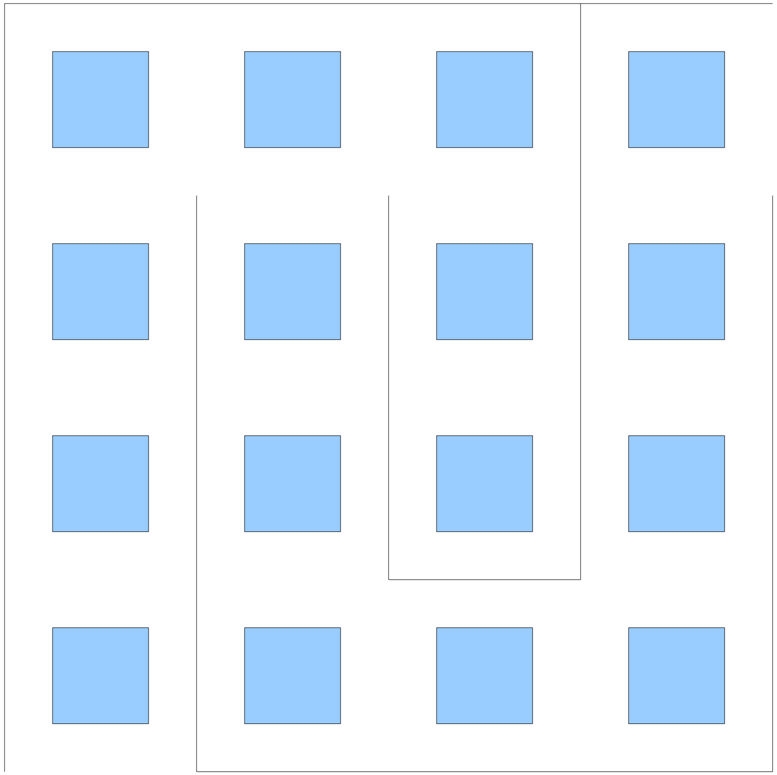
Graph Search Trees



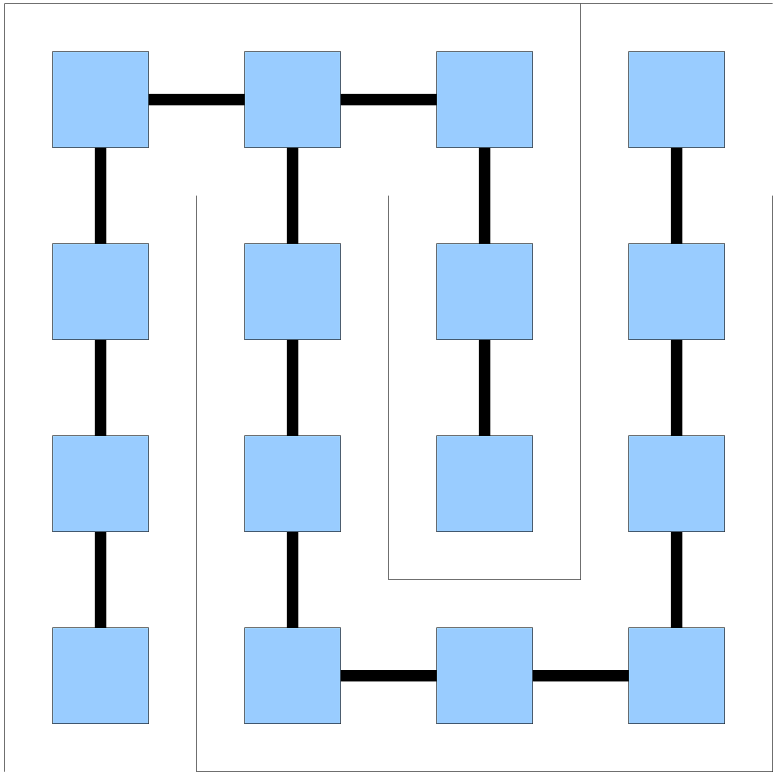
Mazes as Graphs



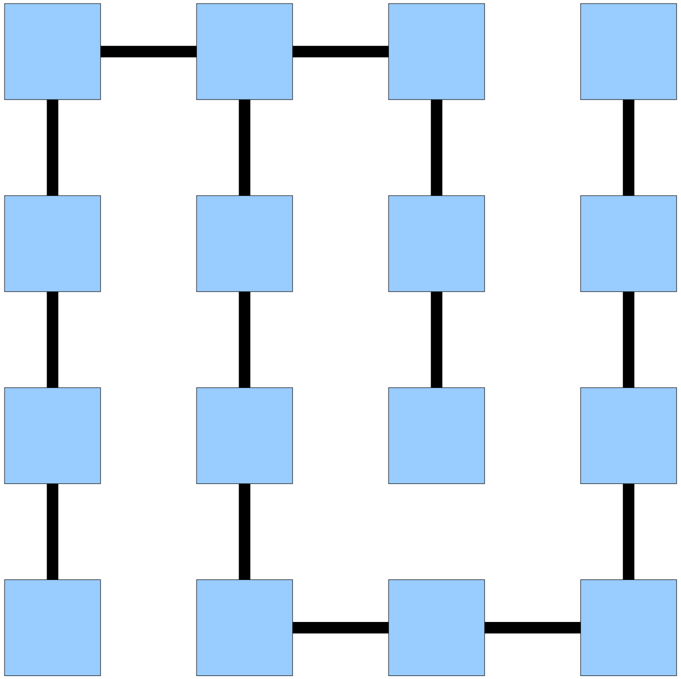
Mazes as Graphs



Mazes as Graphs



Mazes as Graphs

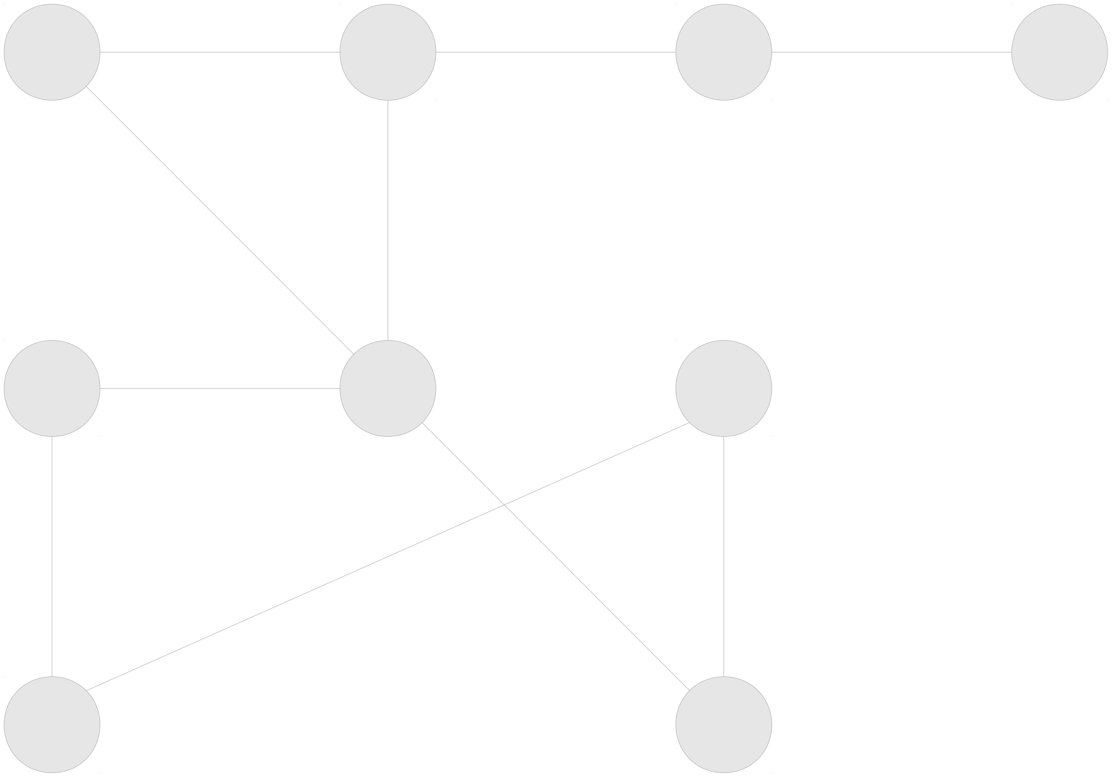


Creating a Maze with DFS

- Create a **grid graph** of the appropriate size.
- Starting at any node, run a depth-first search, adding the arcs to the stack in **random order**.
- The resulting DFS tree is a maze with one solution.

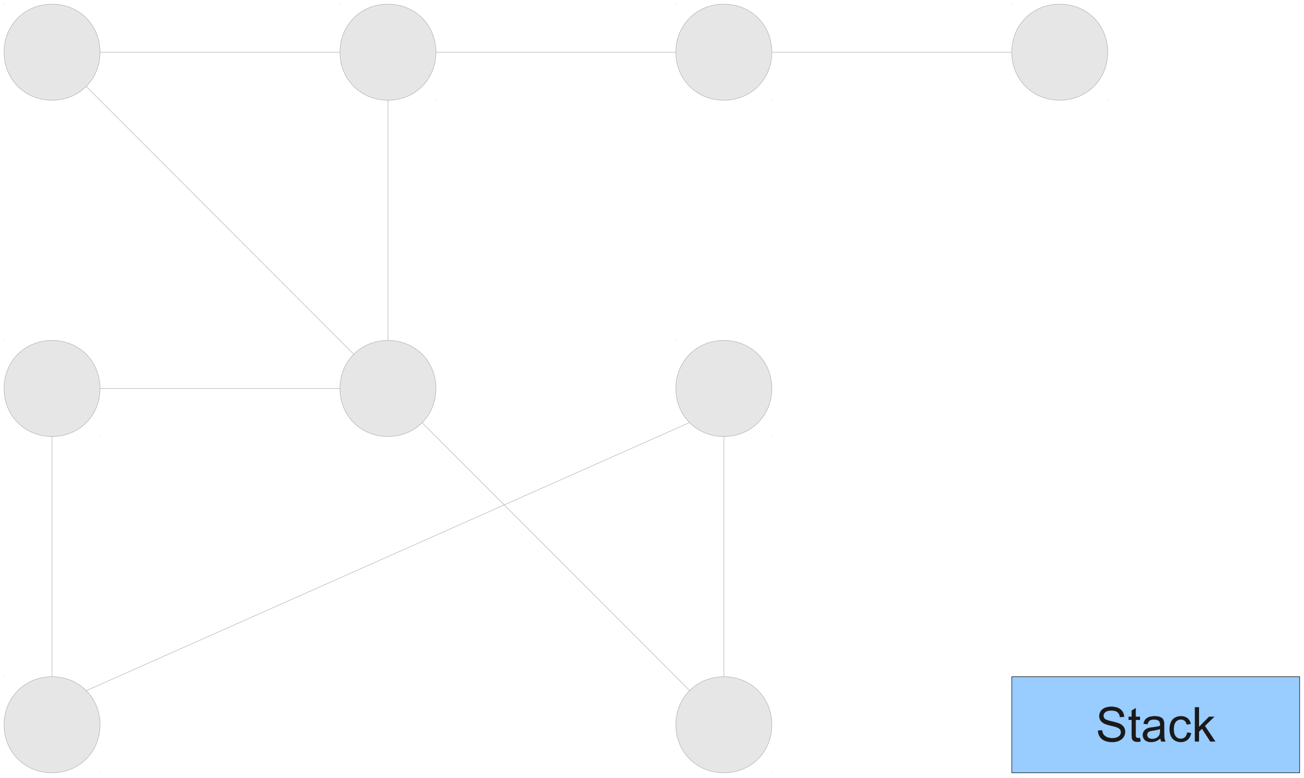
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding specific nodes.



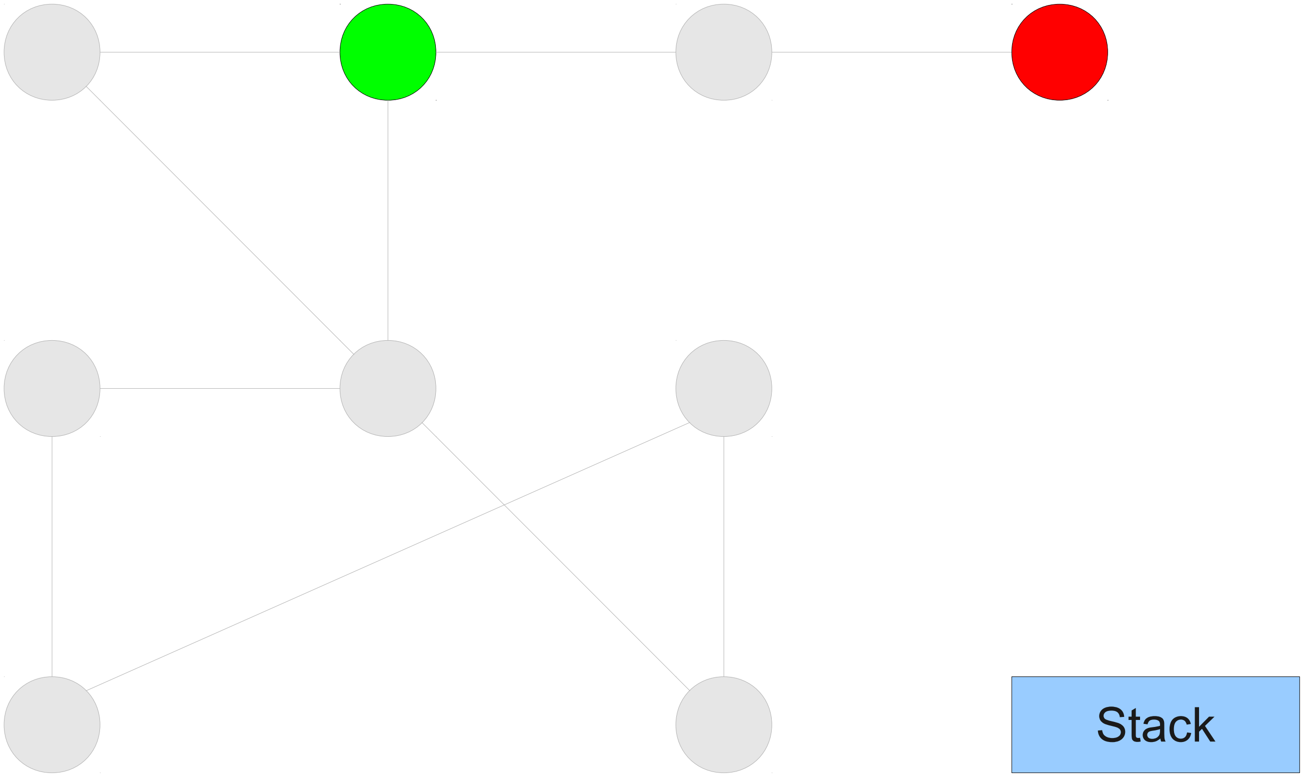
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding specific nodes.



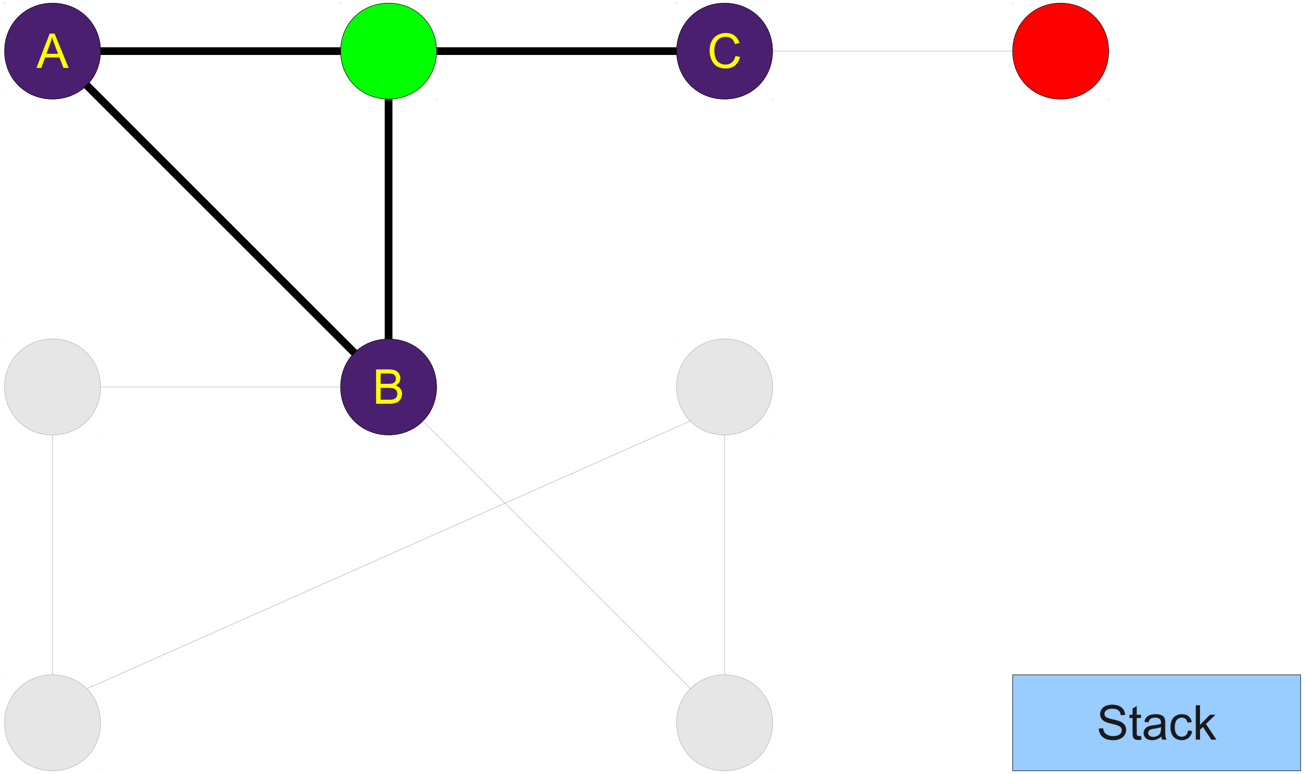
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding specific nodes.



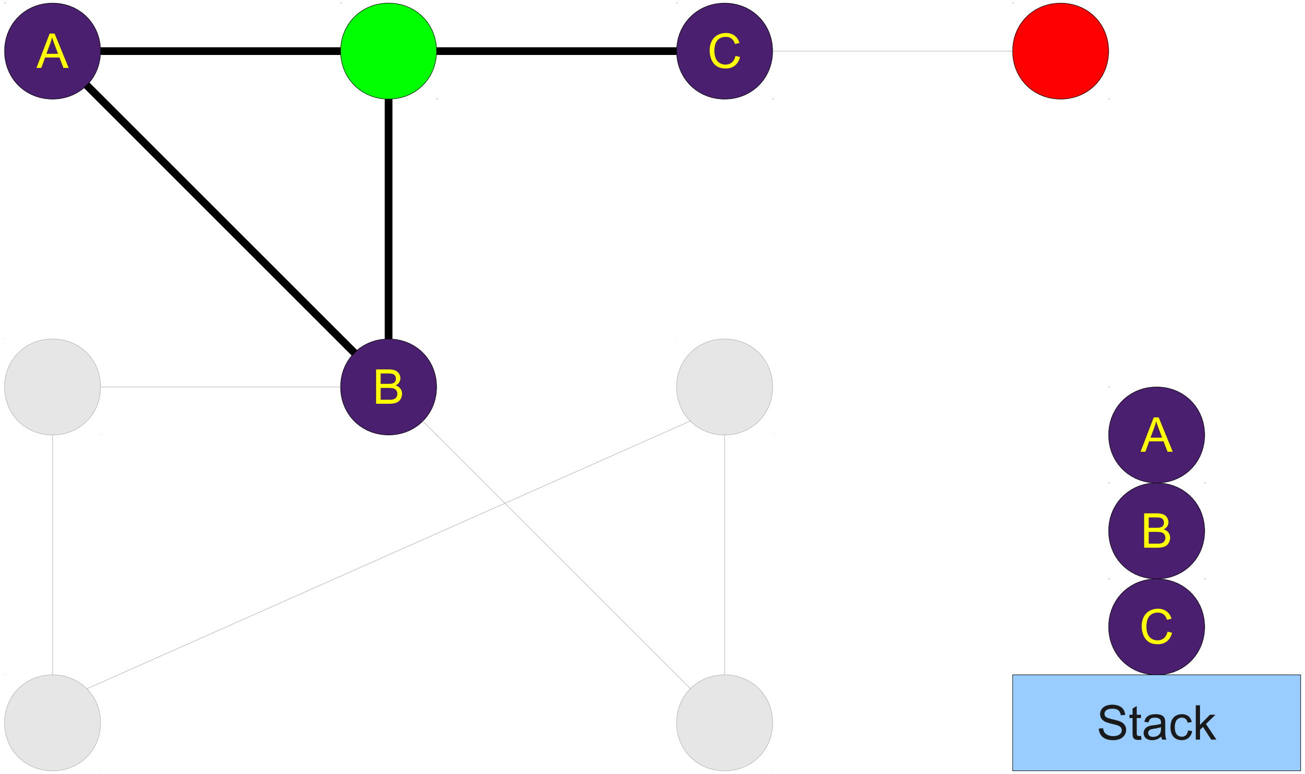
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding specific nodes.



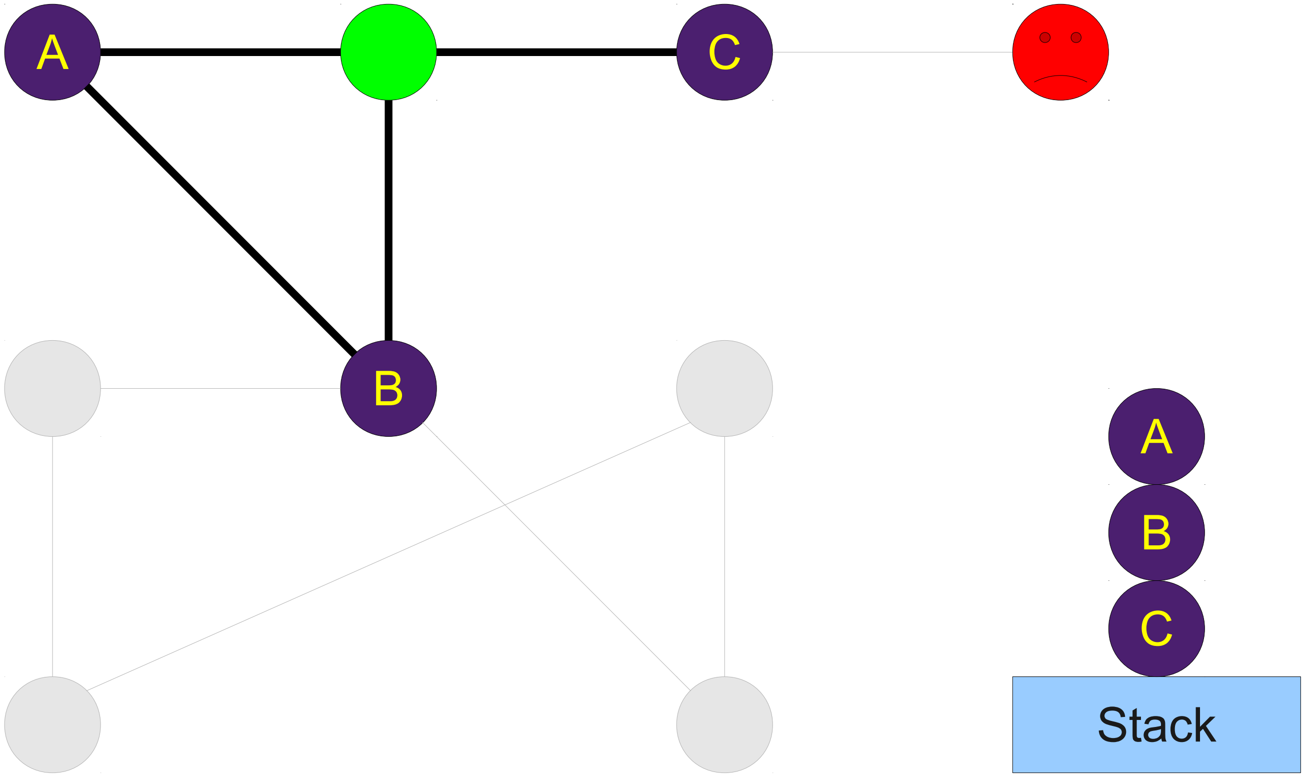
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding specific nodes.



Problems with DFS

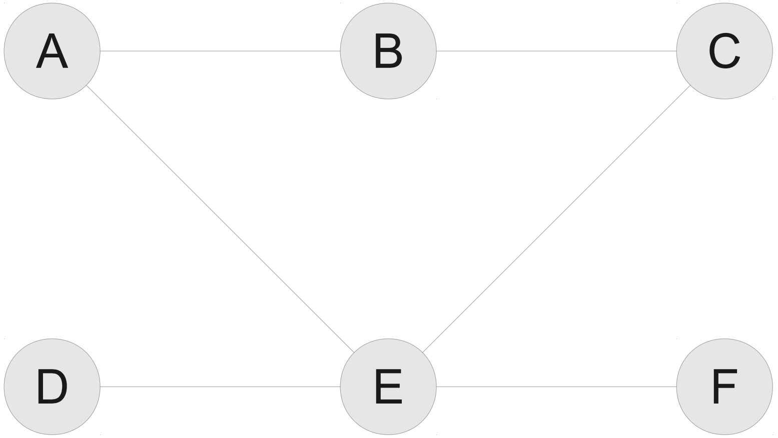
- Useful when trying to explore everything.
- Not good at finding specific nodes.



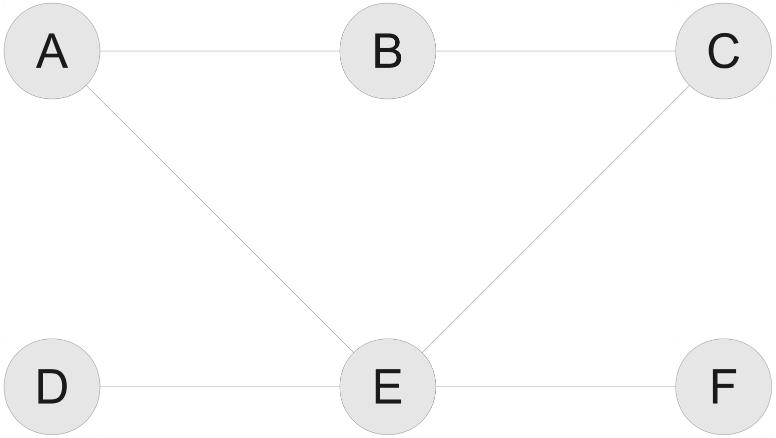
Breadth-First Search

- Specialization of the general search algorithm where nodes to visit are put into a **queue**.
- Explores nodes one hop away, then two hops away, etc.
- Finds path with fewest edges from start node to all other nodes.
- Runs in $O(m + n)$ with adjacency lists, $O(n^2)$ with adjacency matrix.

Breadth-first search

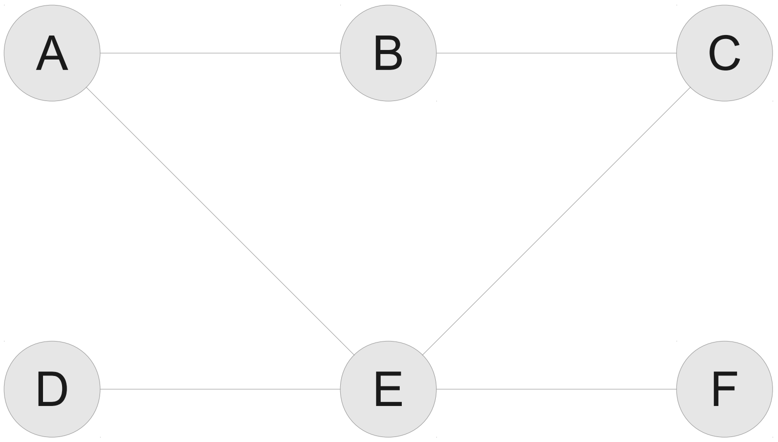


Breadth-first search



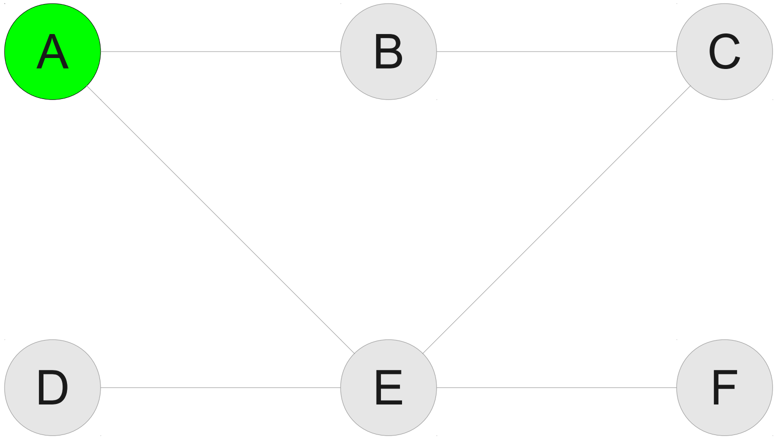
Queue

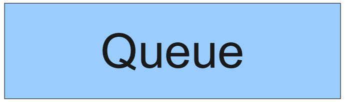
Breadth-first search



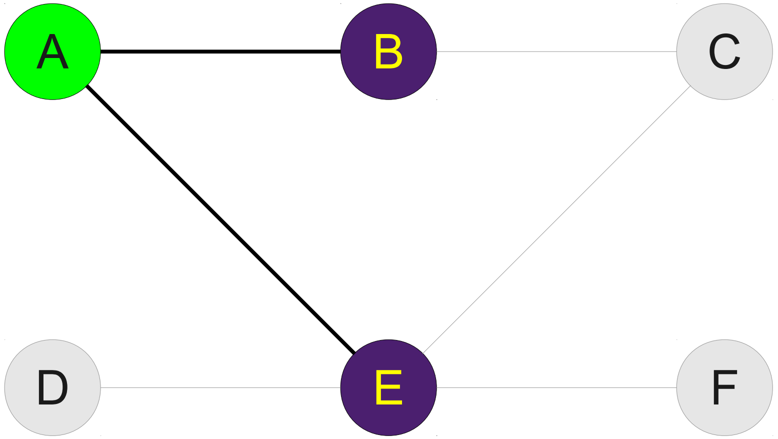
Queue A

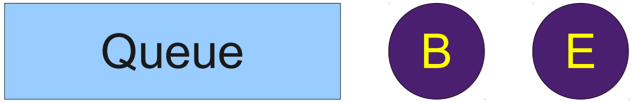
Breadth-first search



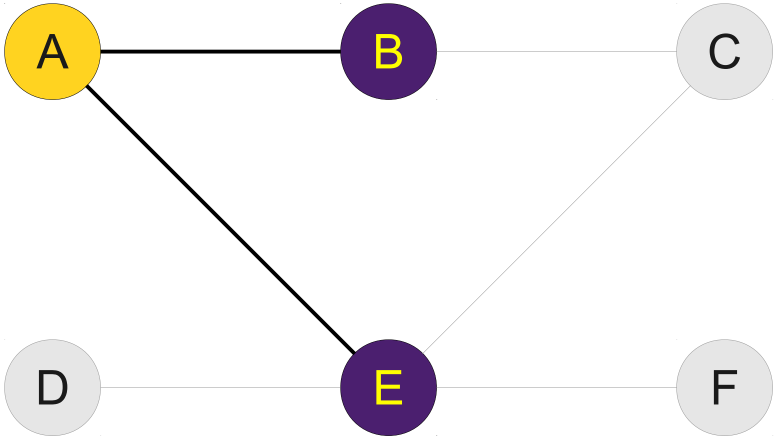


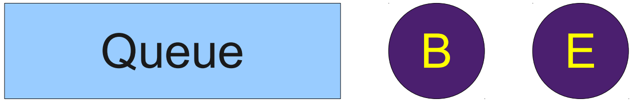
Breadth-first search



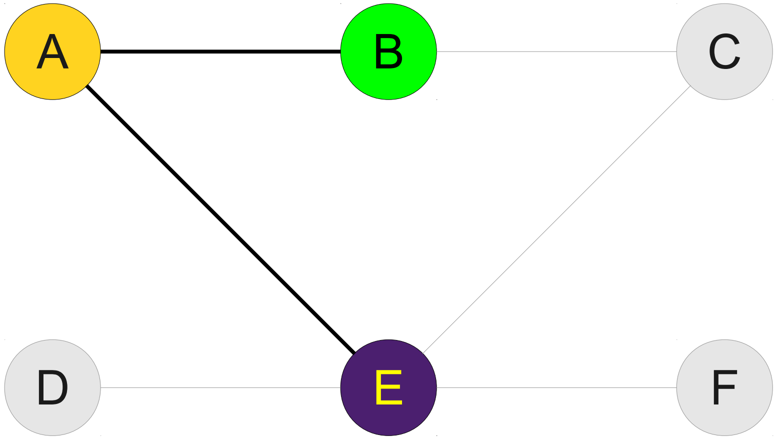


Breadth-first search

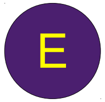




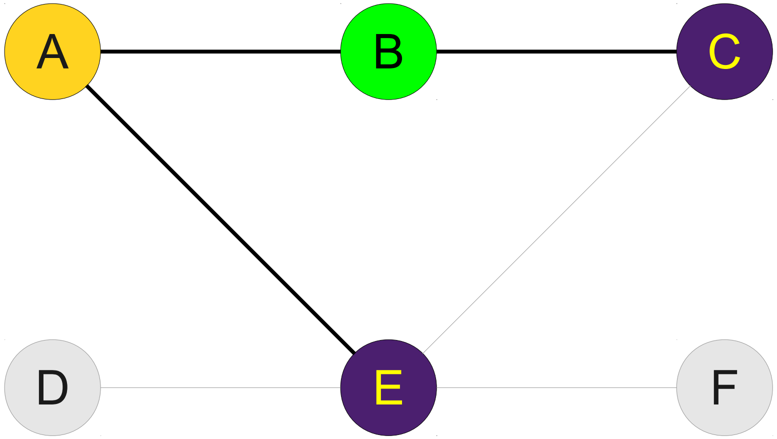
Breadth-first search

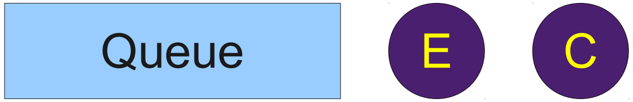


Queue

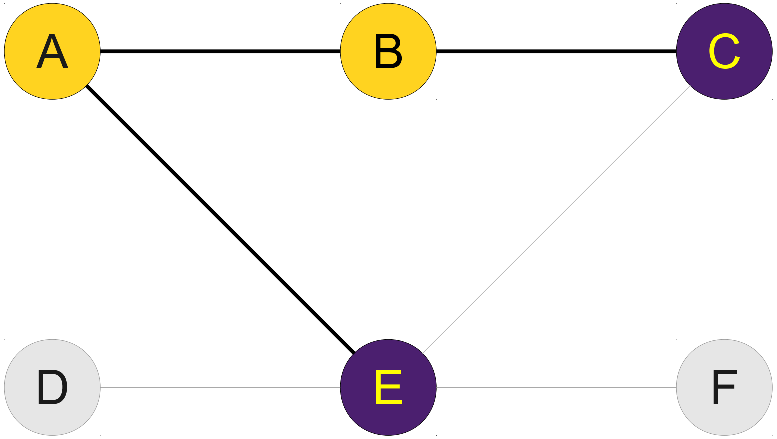


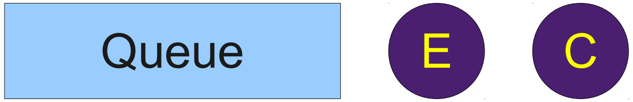
Breadth-first search



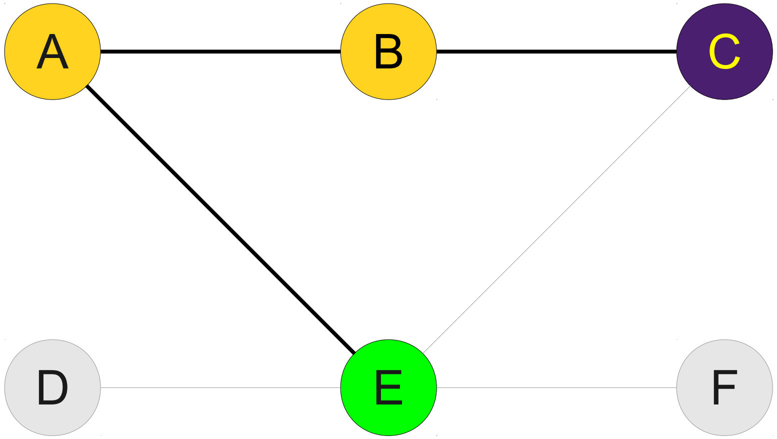


Breadth-first search

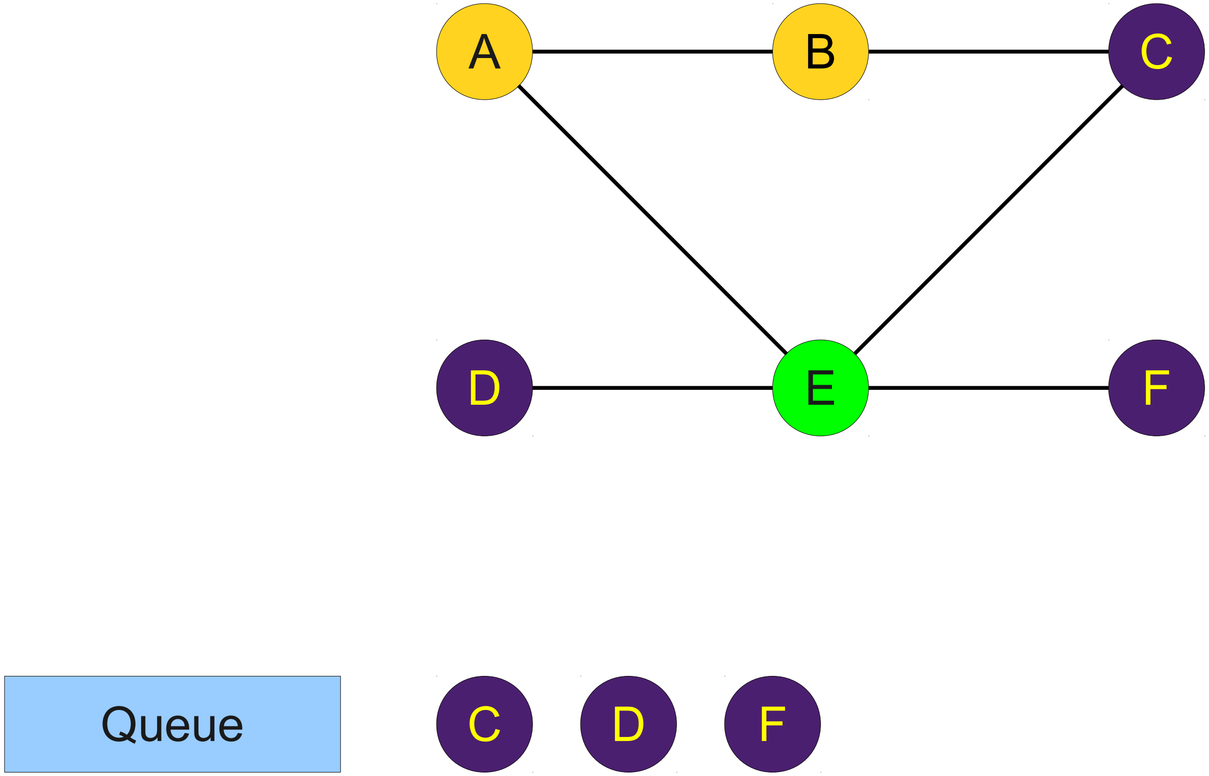




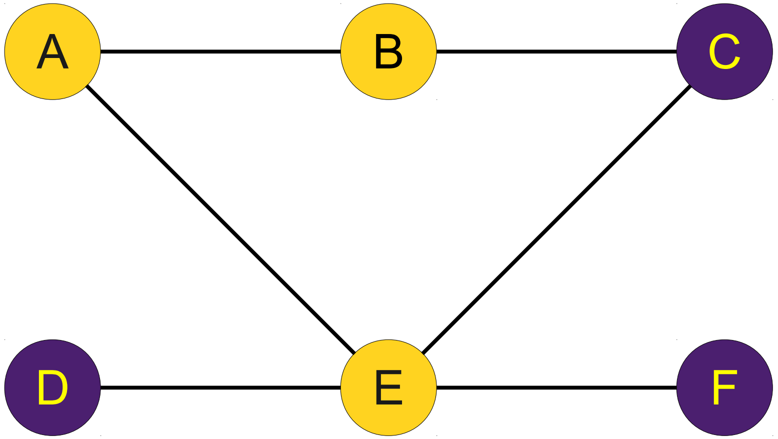
Breadth-first search



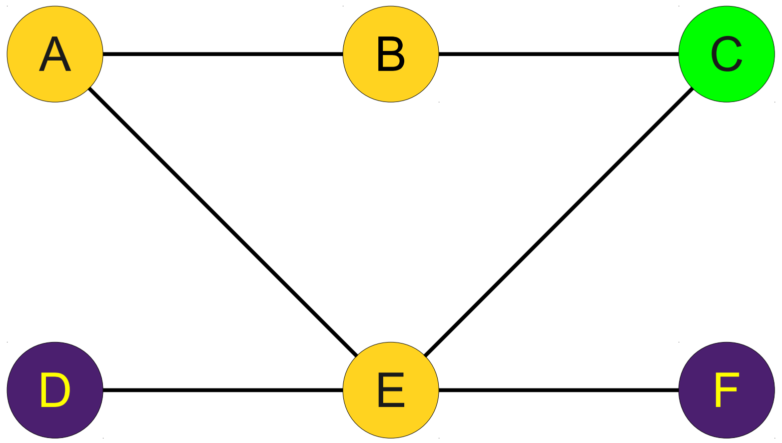
Breadth-first search

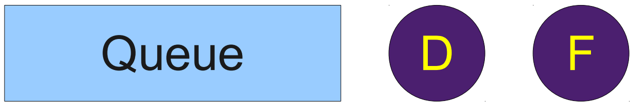


Breadth-first search

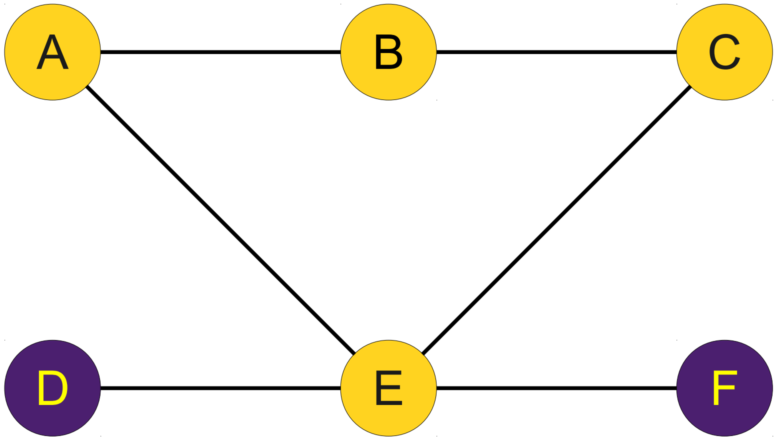


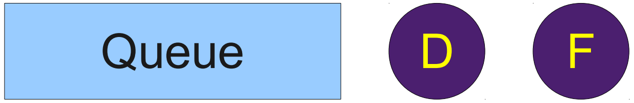
Breadth-first search



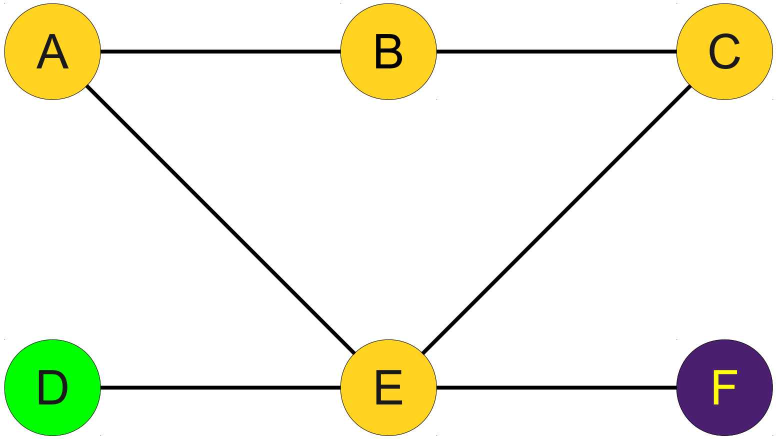


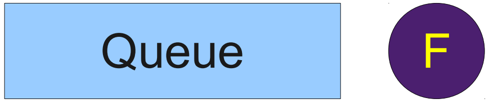
Breadth-first search



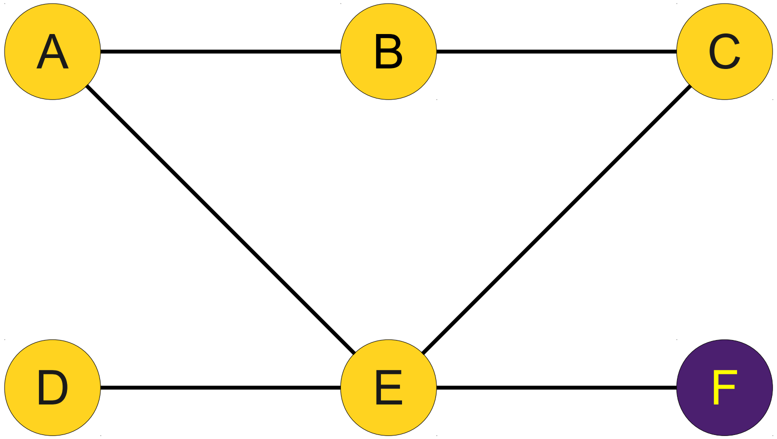


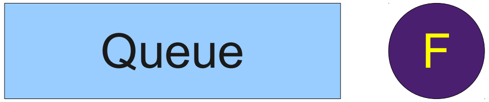
Breadth-first search



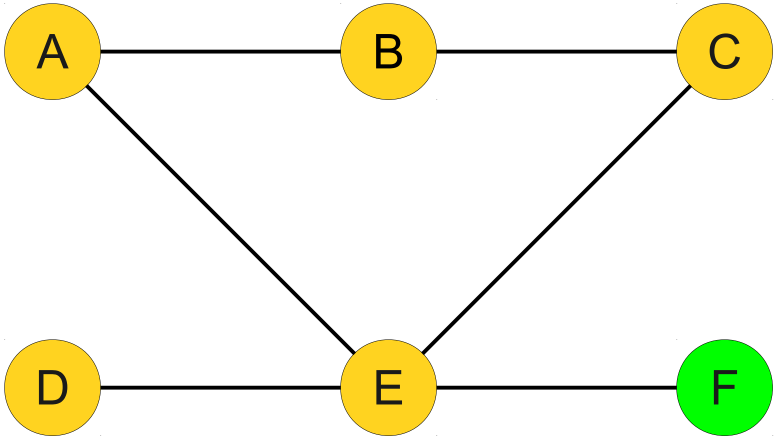


Breadth-first search



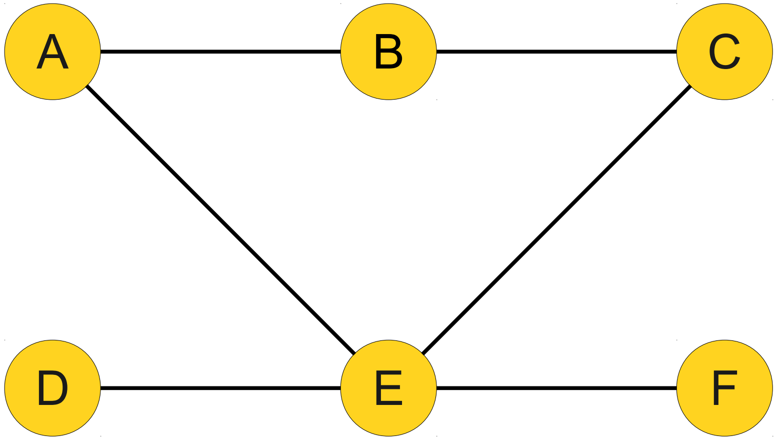


Breadth-first search



Queue

Breadth-first search



Queue

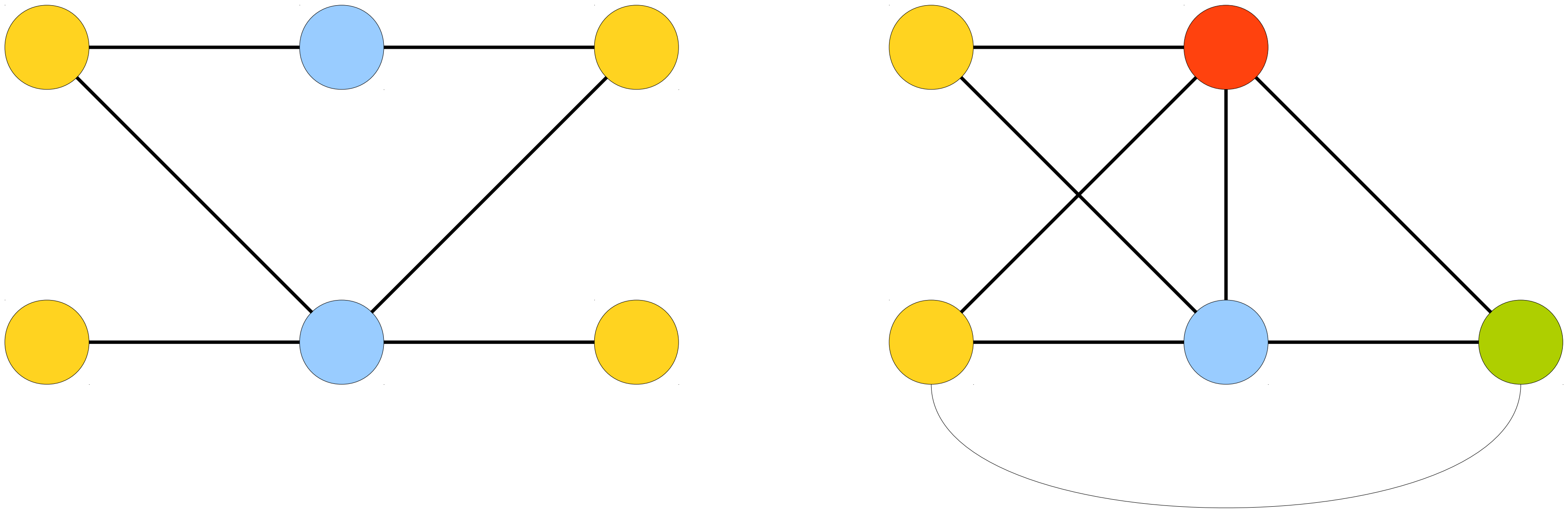
Implementing BFS

```
BFS(Node v, Set<Node> visited) {
   Create a Queue<Node> of nodes to visit;
   Add v to the queue;

   while (The queue is not empty) {
      Dequeue a node from the queue, let it be u;

      if (u has been visited) continue;
      Add u to the visited set;

      for (Node w connected to u)
         Enqueue w in the queue;
   }
}
```

Classic Graph Algorithms

Graph Coloring

- Given a graph G, assign **colors** to the nodes so that no edge has endpoints of the same color.
- The **chromatic number** of a graph is the fewest number of colors needed to color it.

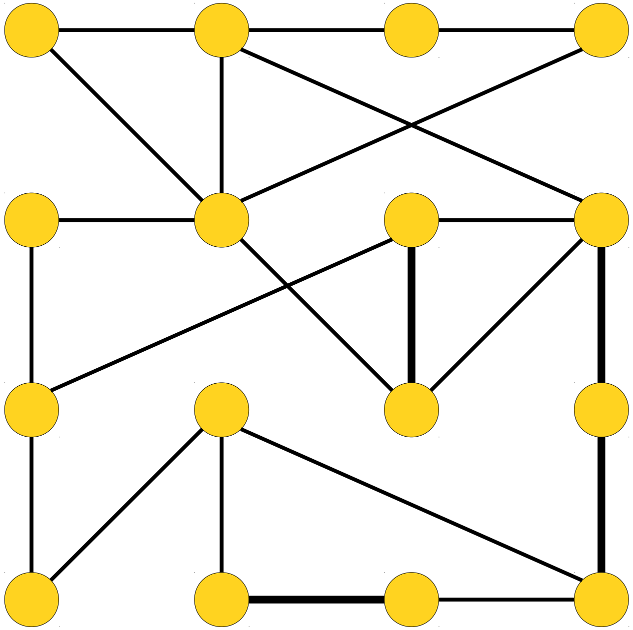


Graph Coloring is **Hard**.

- Determining whether a graph can be colored with *k* colors (for k > 2) is **NP-complete**.
- It is not known whether this problem can be solved in polynomial time.
- Want $1,000,000? **Find a polynomial-time algorithm** or **prove that none exists**.

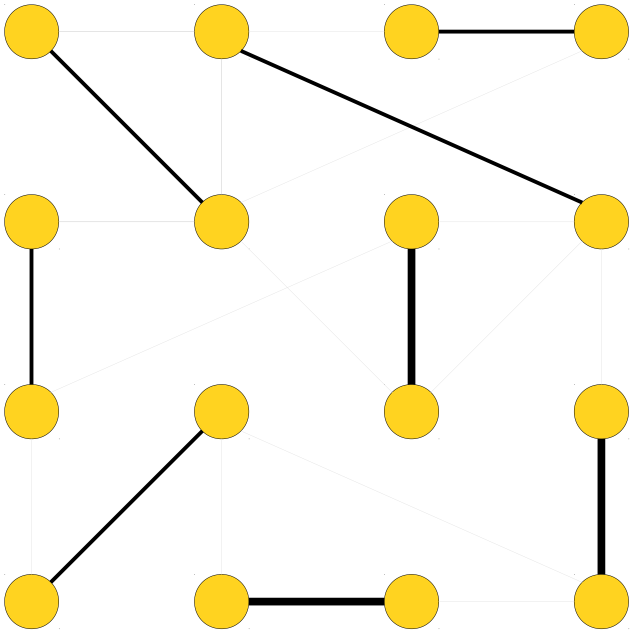
Matching

- A **matching** in a graph is a subset of the edges that don't share any endpoints.
- Intuitively, pairing up nodes in the graph.



Matching

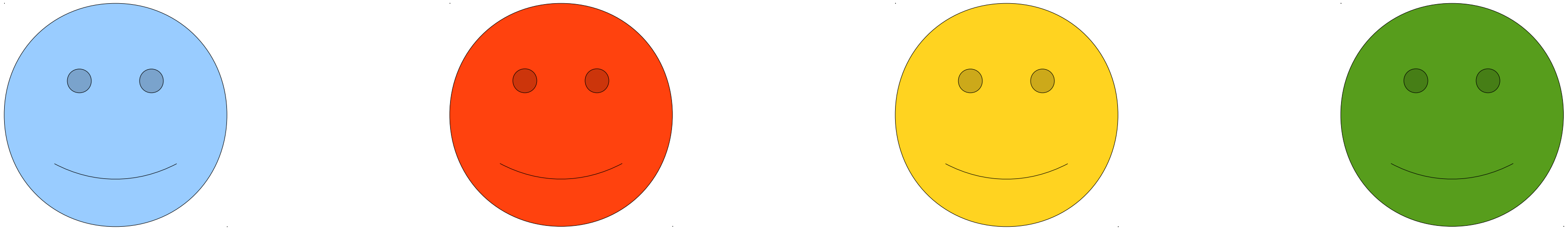
- A **matching** in a graph is a subset of the edges that don't share any endpoints.
- Intuitively, pairing up nodes in the graph.



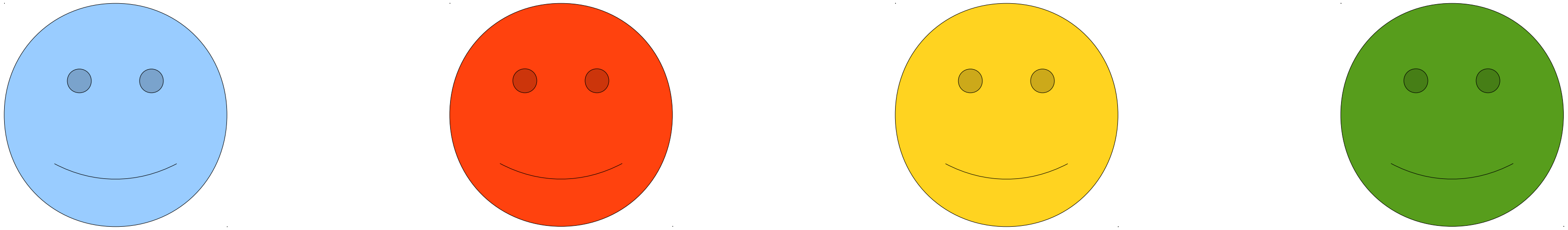
Applications of Matching

- Unlike graph coloring, matching can be done quickly.
- Sample application: divvying up desserts.

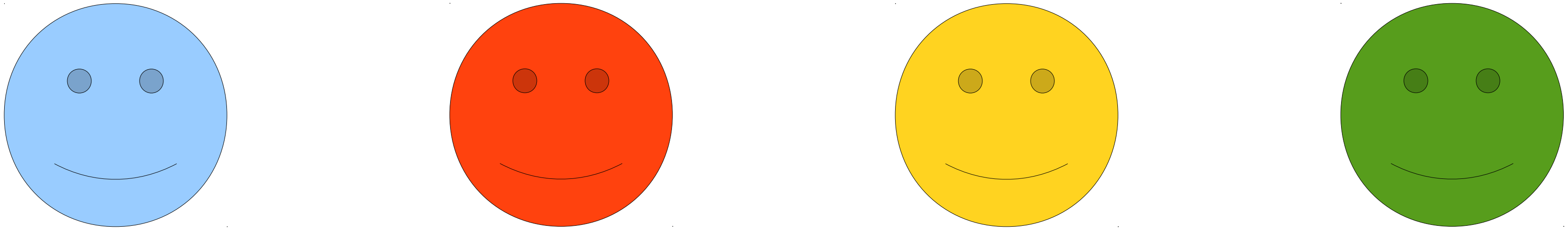
Divvying Up Desserts



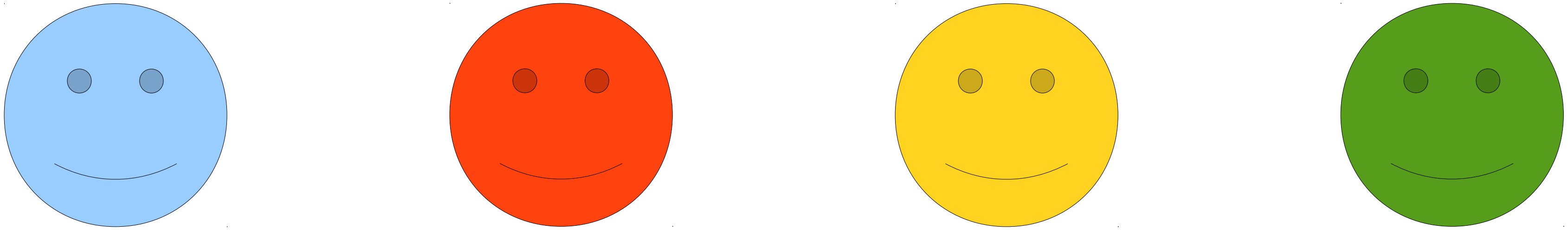
Divvying Up Desserts



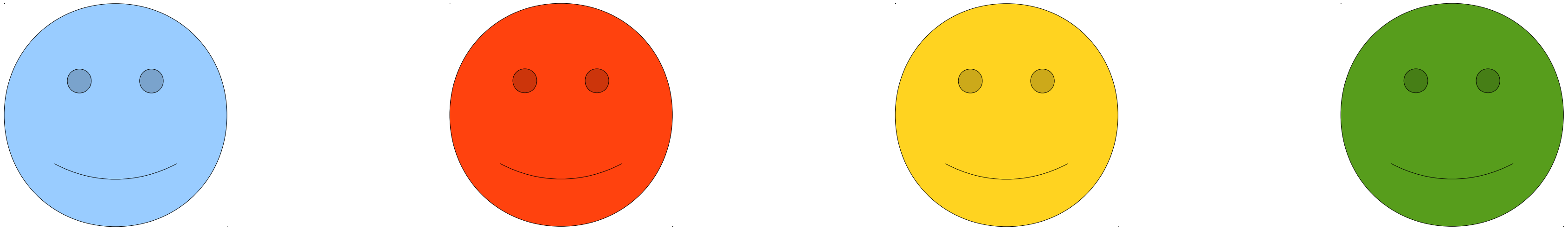
Divvying Up Desserts

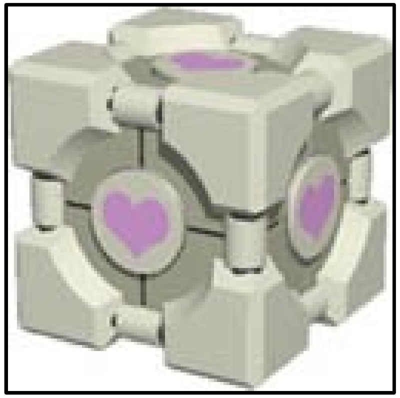


Divvying Up Desserts

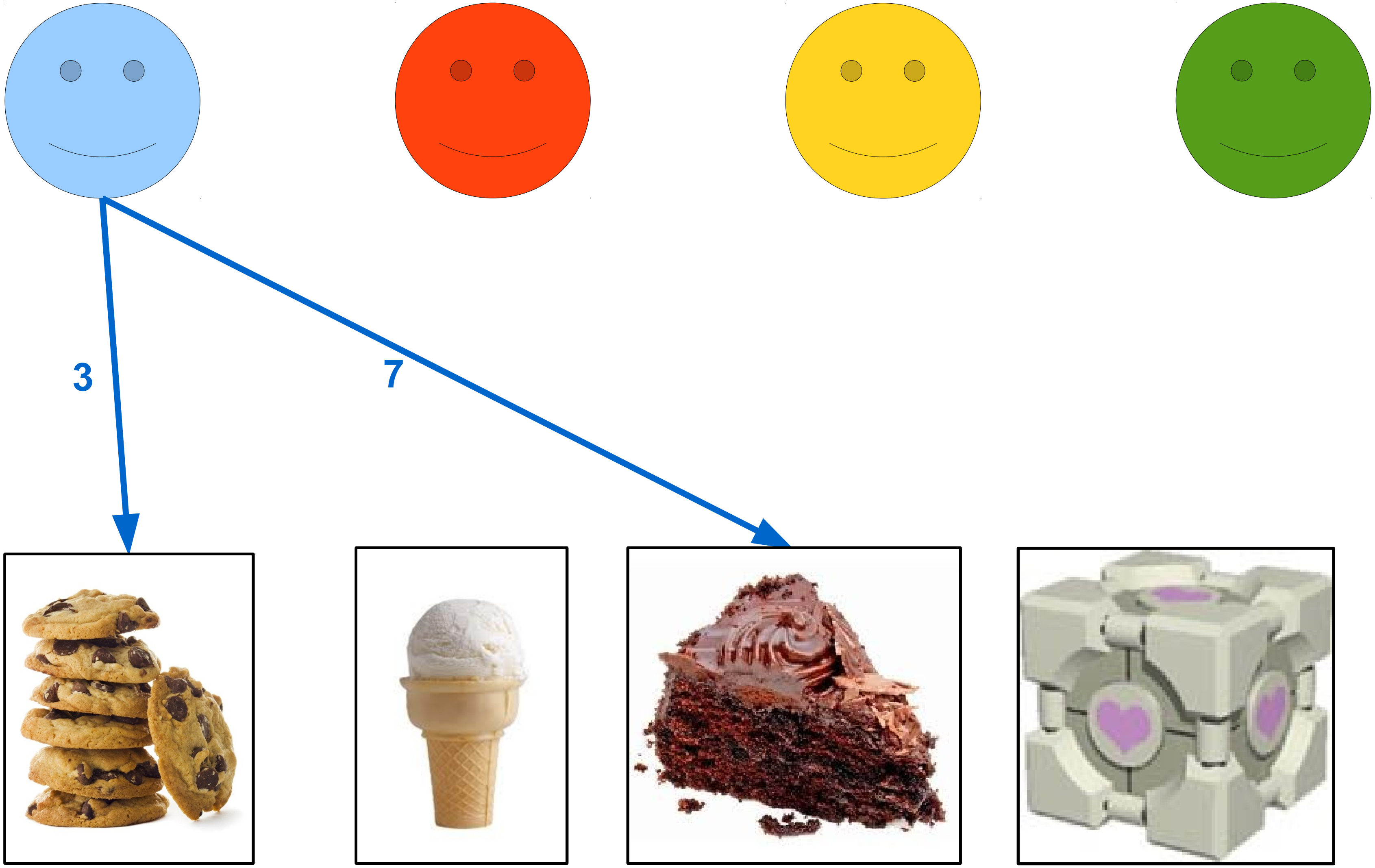


Divvying Up Desserts

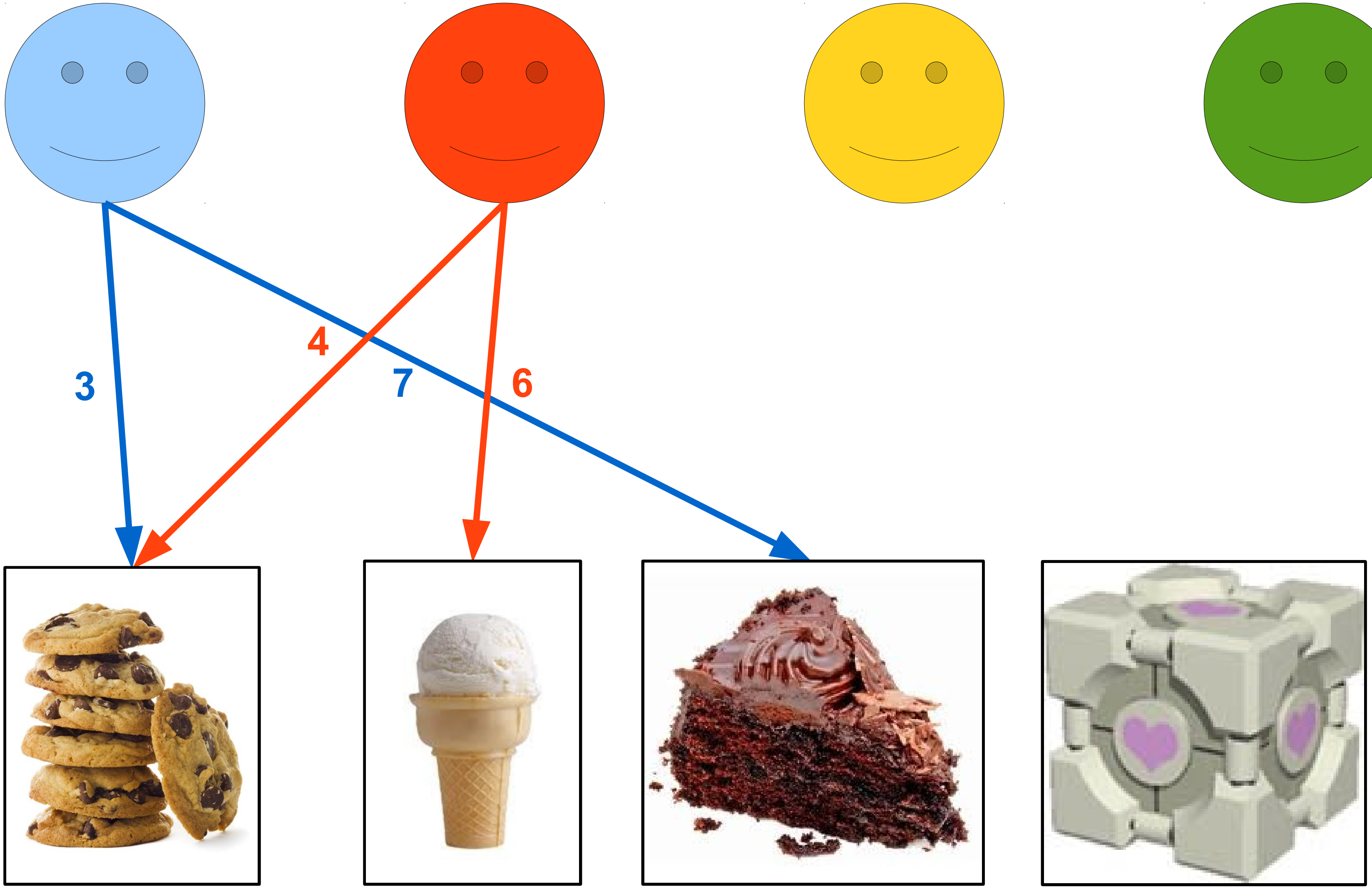




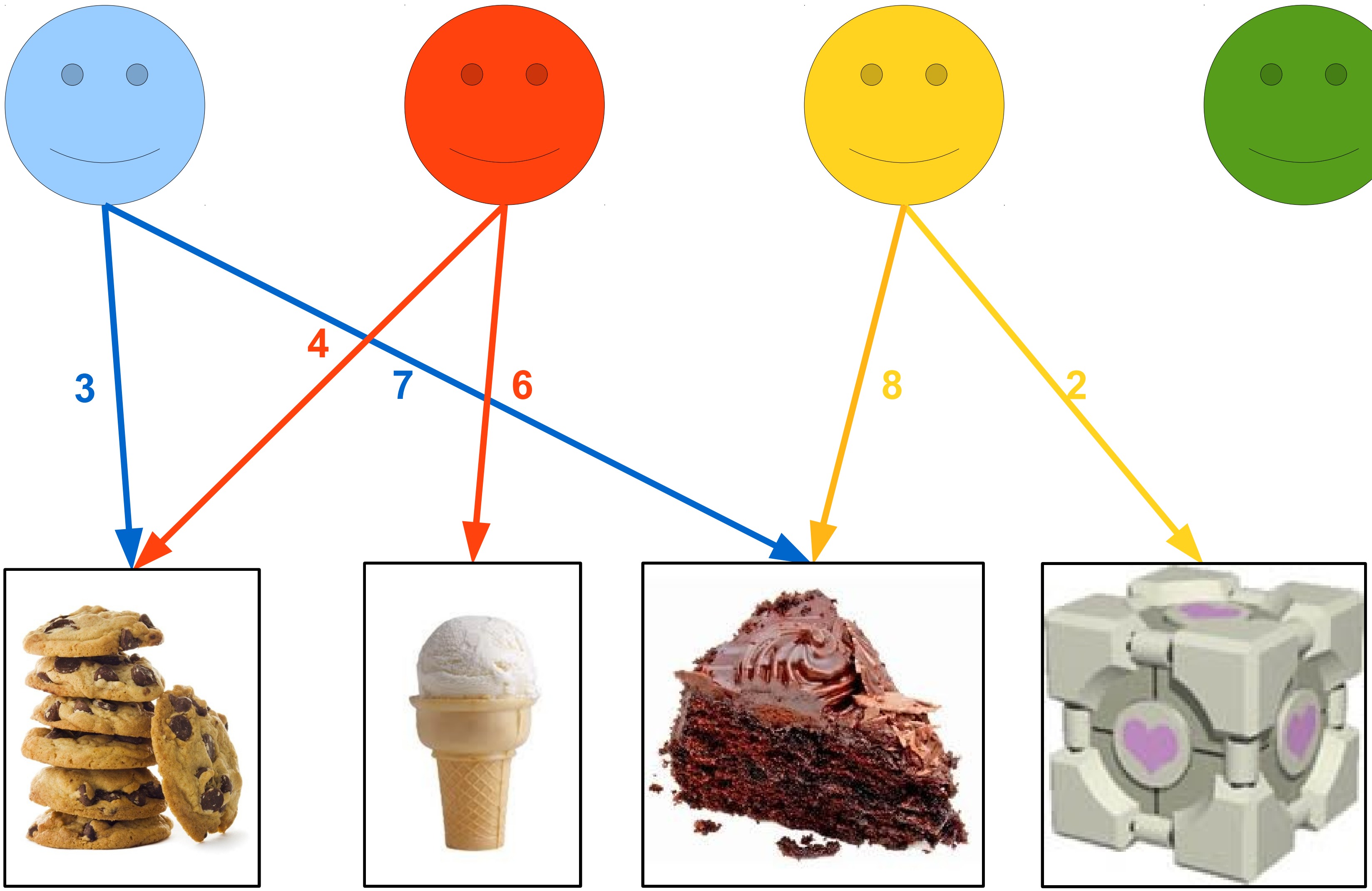
Divvying Up Desserts



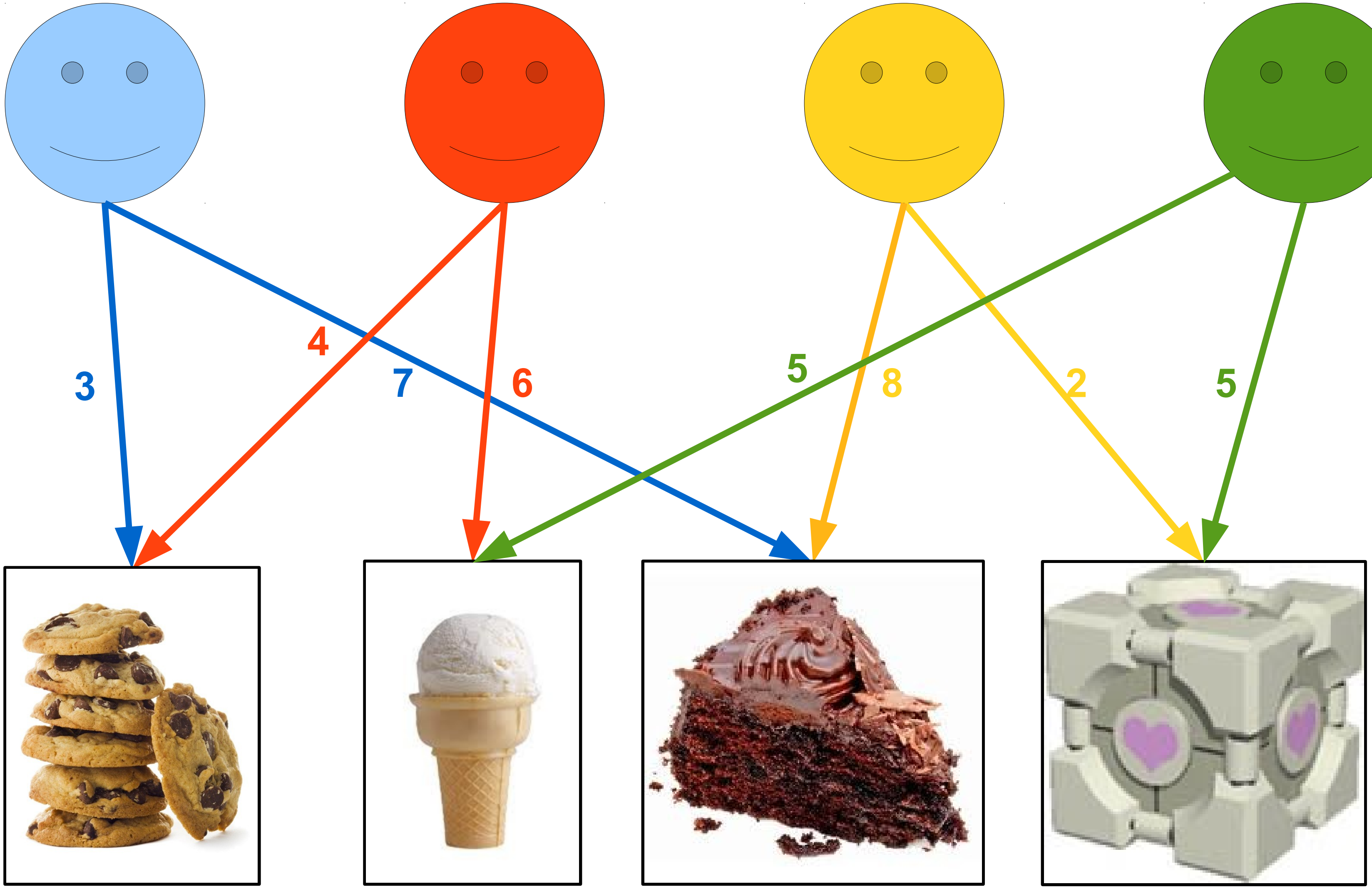
Divvying Up Desserts



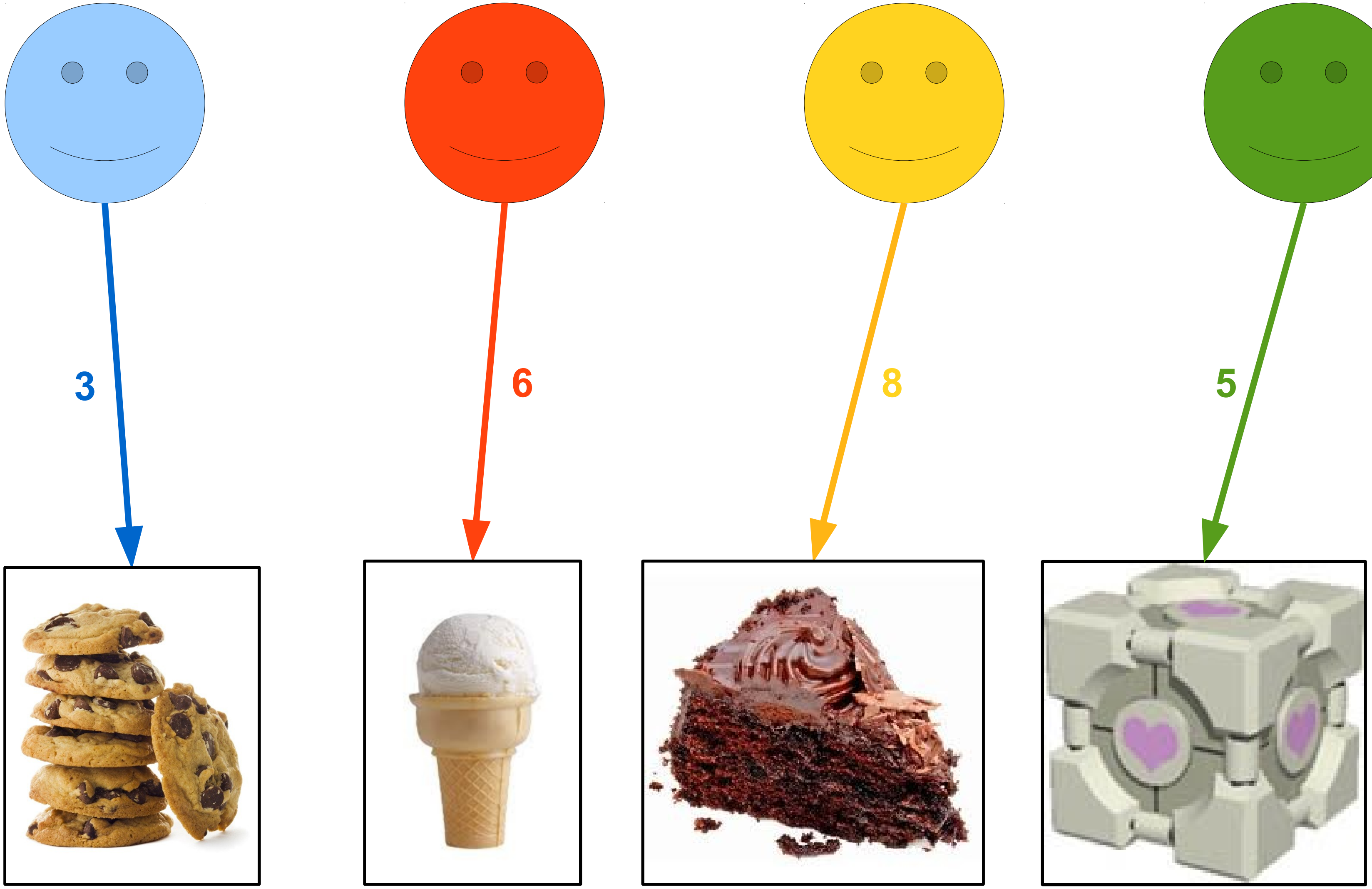
Divvying Up Desserts



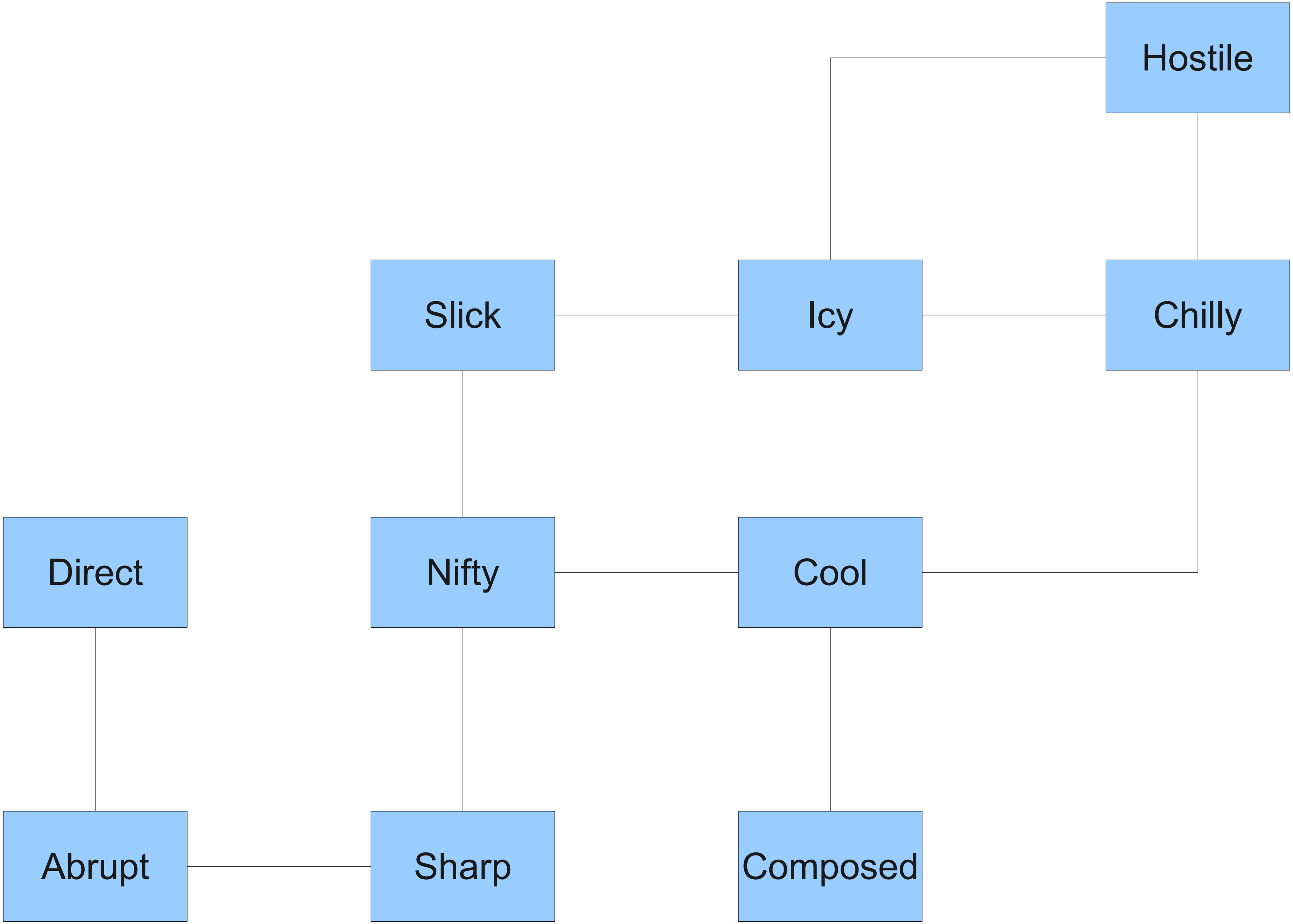
Divvying Up Desserts

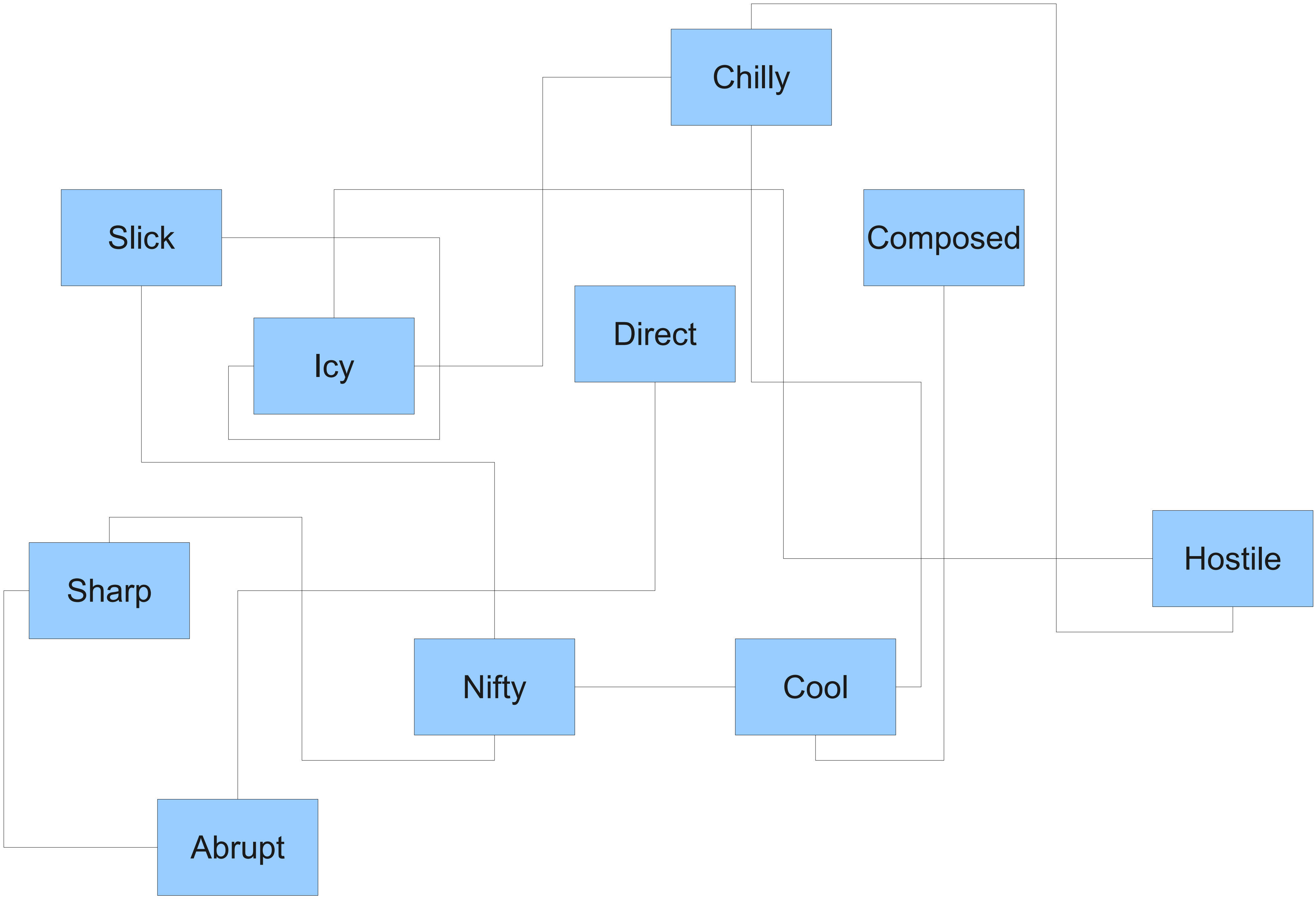


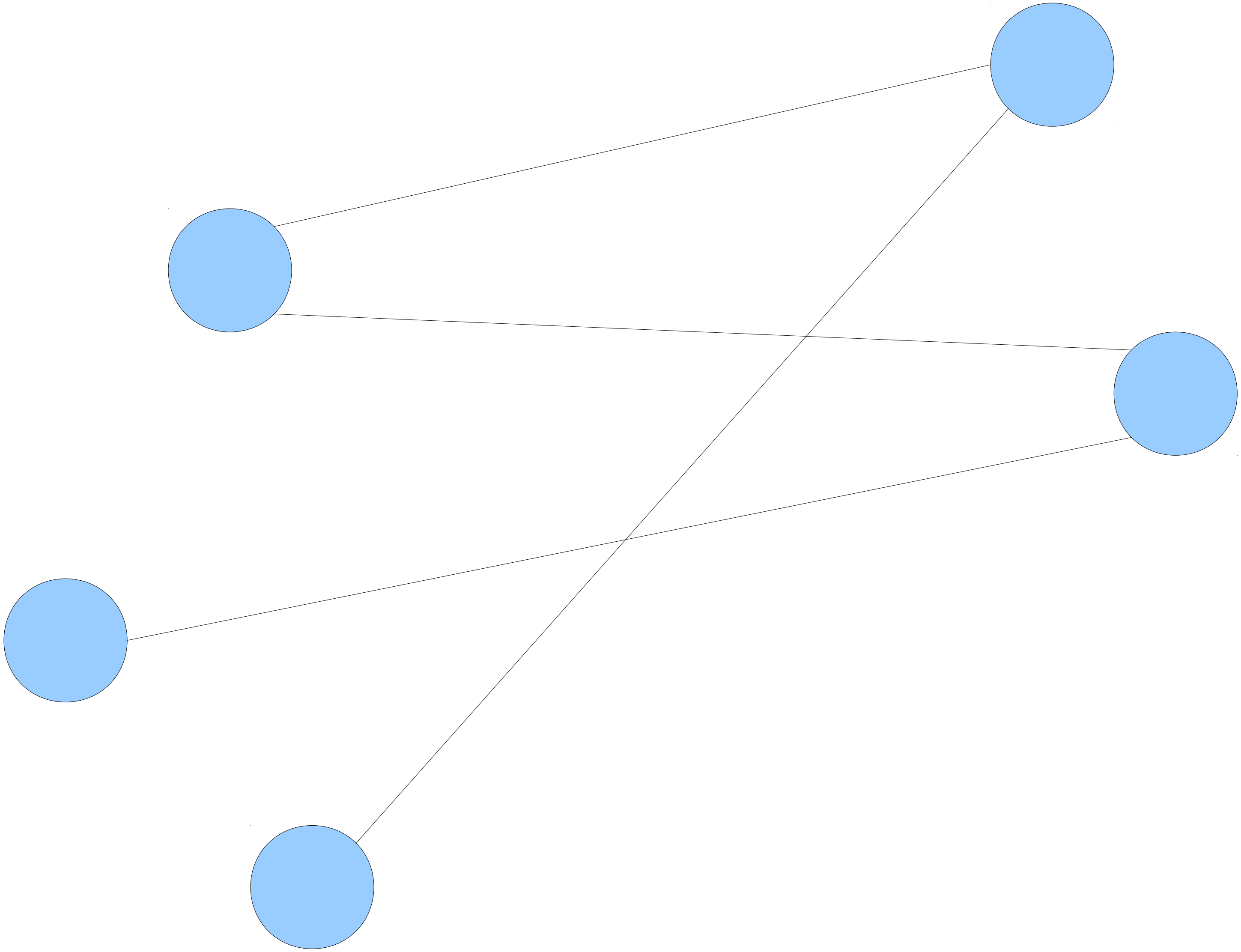
Divvying Up Desserts

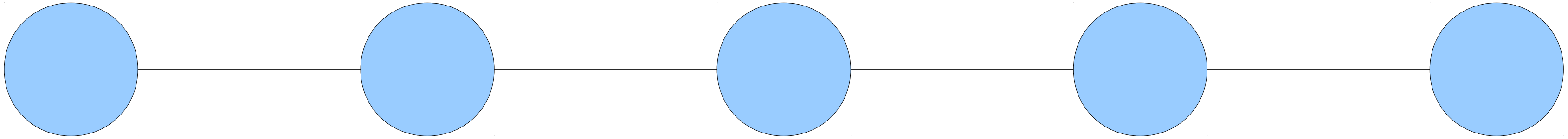


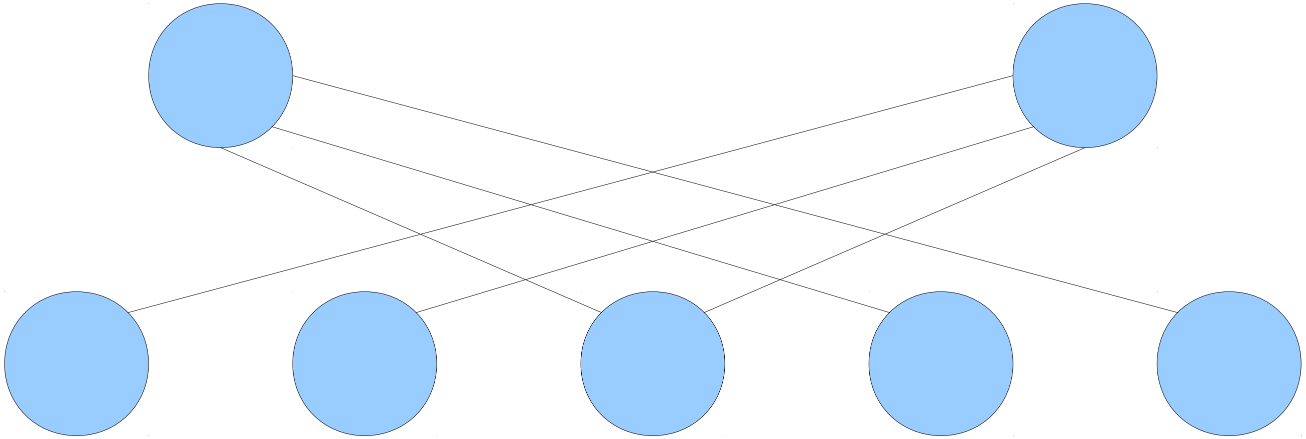
Drawing Graphs

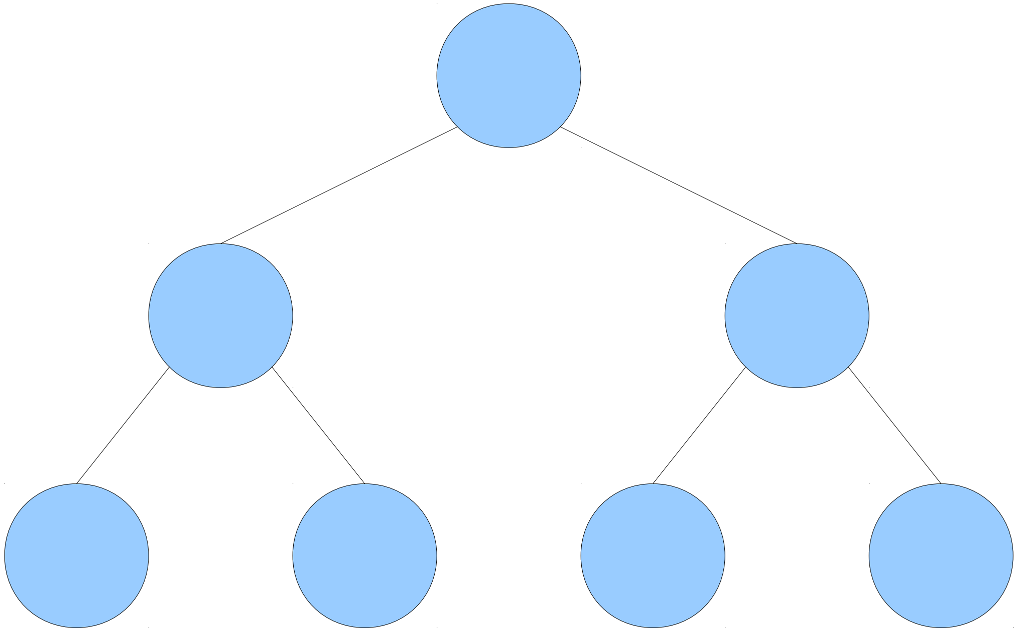

Hostile
Slick
Icy
Chilly
Direct
Nifty
Cool
Abrupt
Sharp
Composed


Chilly
Slick
Composed
Direct
Icy
Hostile
Sharp
Nifty
Cool
Abrupt









Idea: Treat the graph as a **physical system** that exerts forces on itself.

This is called a **force-directed layout algorithm**.

Summary

- **Graphs** are a powerful abstraction for modeling **relationships** and **connectivity**.
- **Adjacency lists** and **adjacency matrices** are two common representations of graphs.
- Directed acyclic graphs can be visited via a **topological sort**.
- **Depth-first search** is a simple graph exploration algorithm.
- **Breadth-first search** searches a graph one layer at a time.
- There are many classic algorithms on graphs:
    - **Graph coloring** tries to color nodes so no two nodes of the same color are connected.
    - **Matchings** represent pairing up of graph elements.
    - **Graph drawing** seeks to render aesthetically-pleasing graphs.