go.osu.edu/cpcattend

Welcome to CPC!

Big Header

Small header

Paragraph text. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Code block

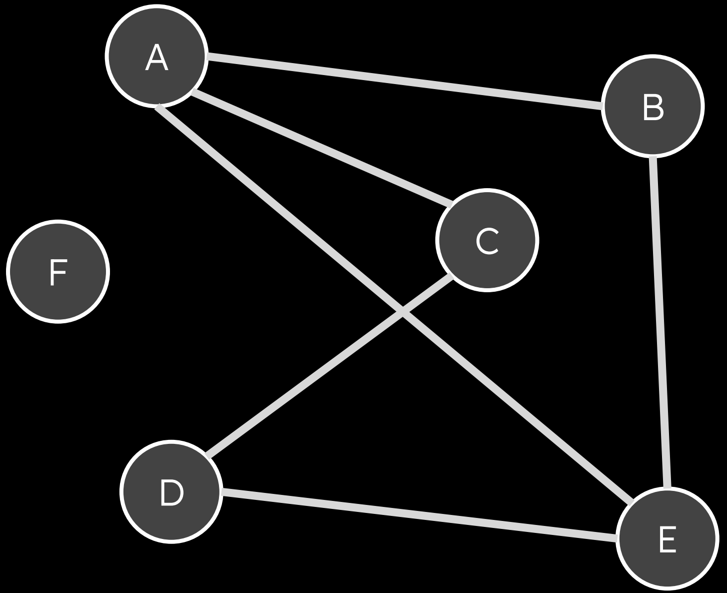
```
Queue<Integer> queue = new Queue<Integer>();
queue.add(5);
queue.add(6);
queue.peek();
queue.remove();
queue.remove();
```

Graphs

- A graph is an abstract representation of some set of objects (nodes) and connections between them (edges).

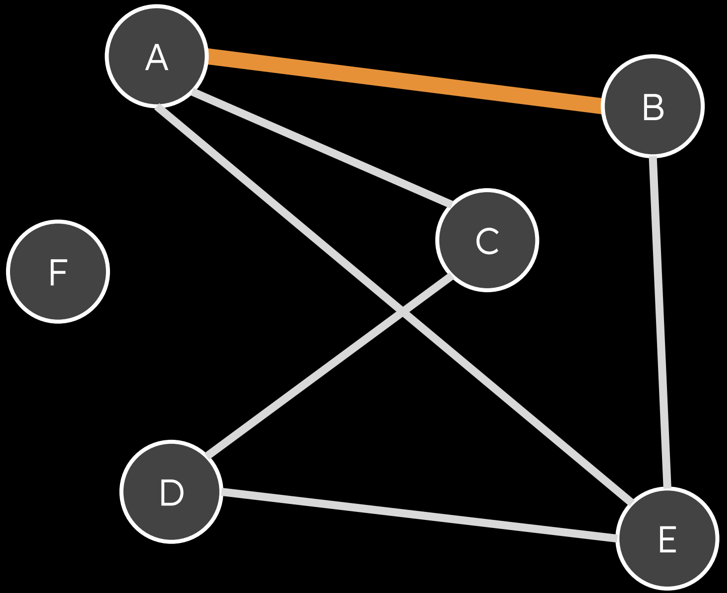
- A vertex set *V*
  { A, B, C, D, E, F }

- An edge set *E*
  { ..., (A, B), (D, C), ... }



Graphs

- A graph is an abstract representation of some set of objects (nodes) and connections between them (edges).
- A vertex set *V*
  { A, B, C, D, E, F }
- An edge set *E*
  { ..., **(A, B)**, (D, C), ... }

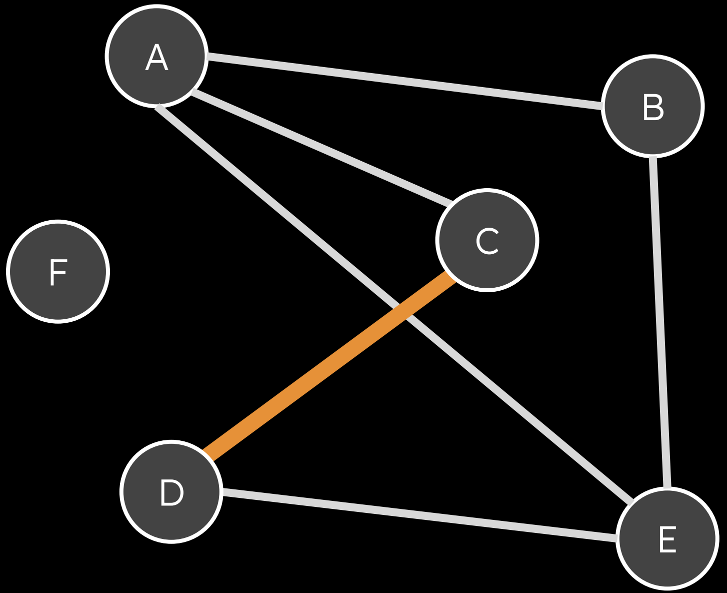


Graphs

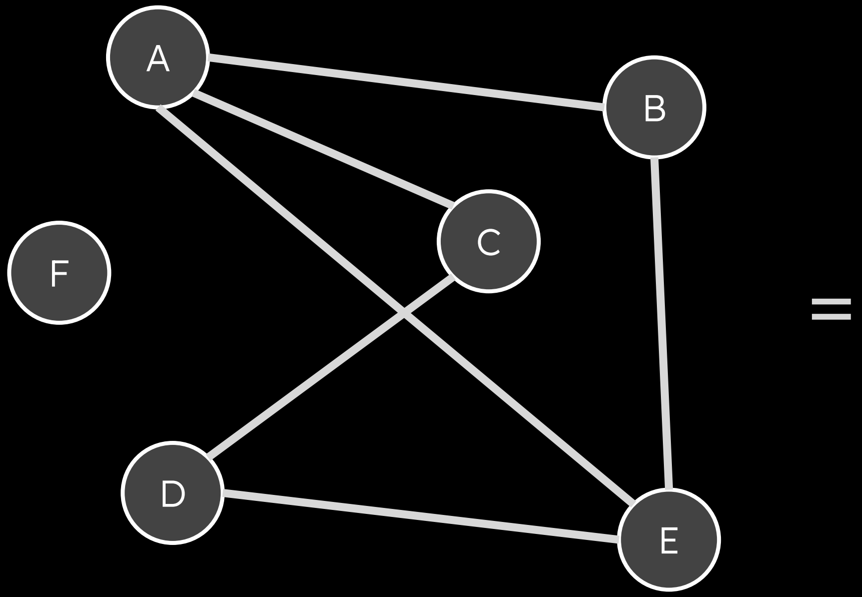
- A graph is an abstract representation of some set of objects (nodes) and connections between them (edges).

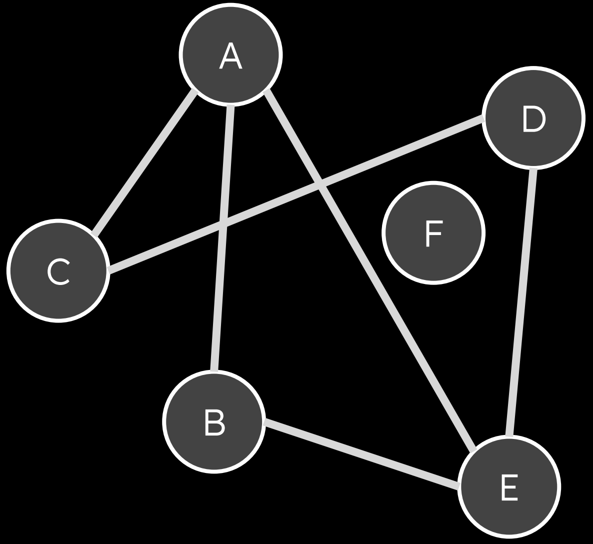
- A vertex set *V*
  { A, B, C, D, E, F }

- An edge set *E*
  { ..., (A, B), **(D, C)**, ... }

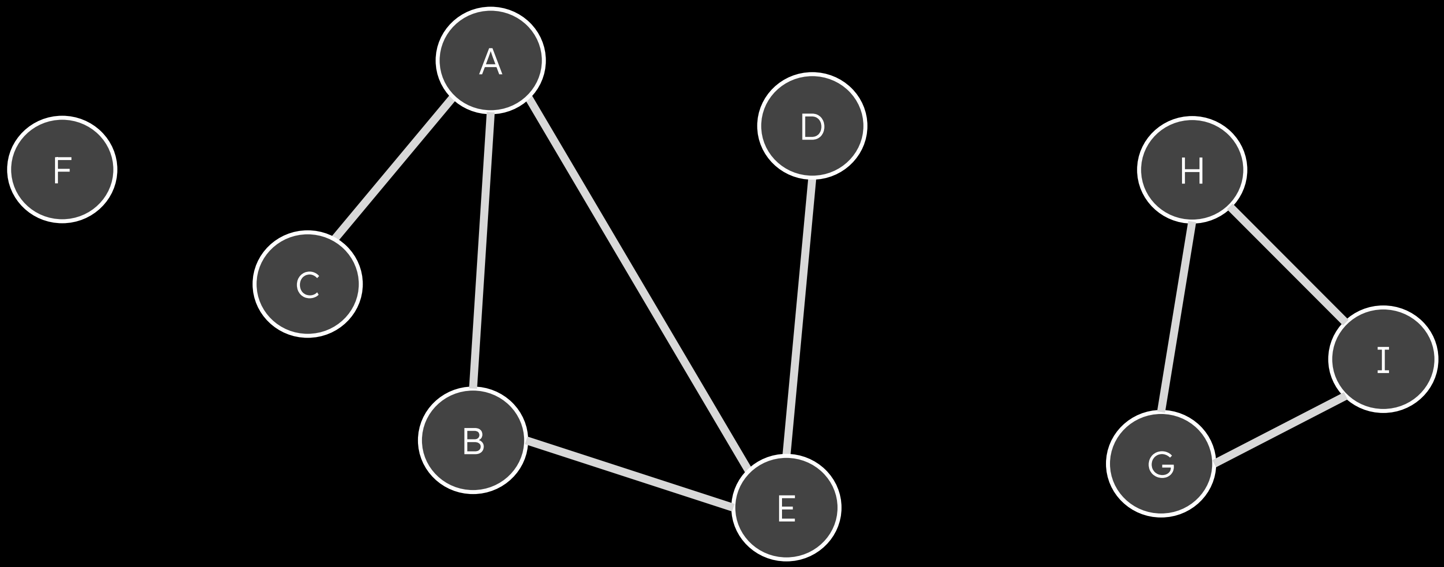


Graphs Drawings



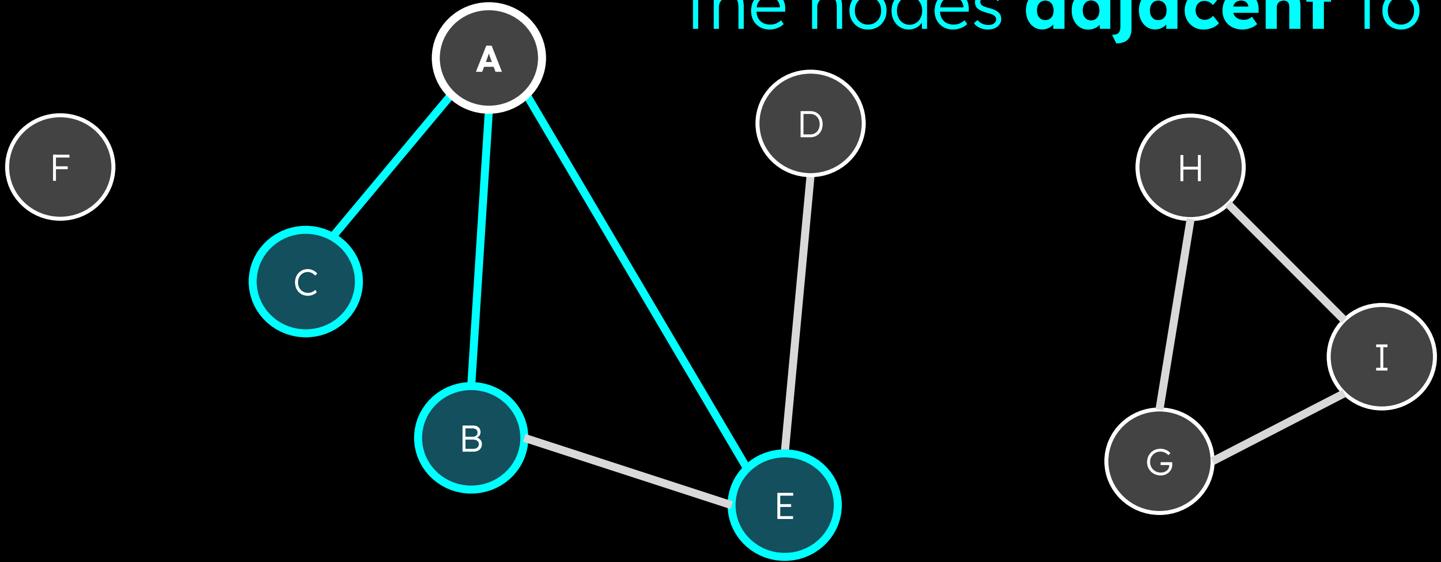


Graph Terminology



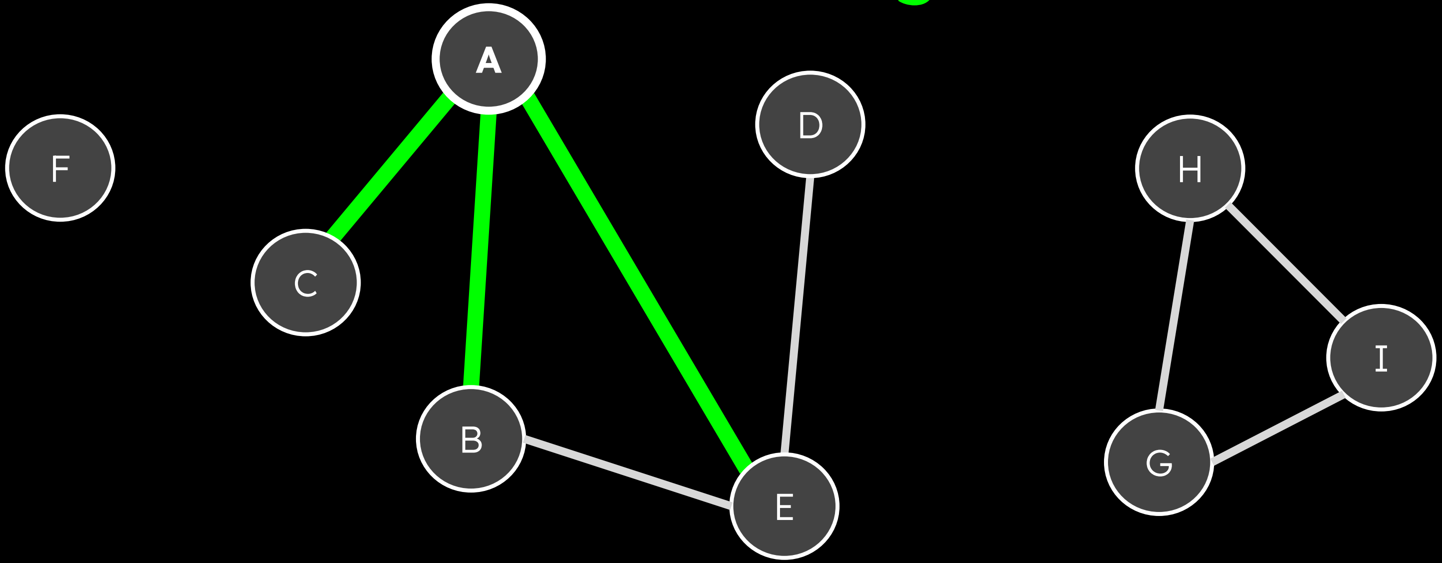
Graph Terminology

the **neighbors** of A
the nodes **adjacent** to A



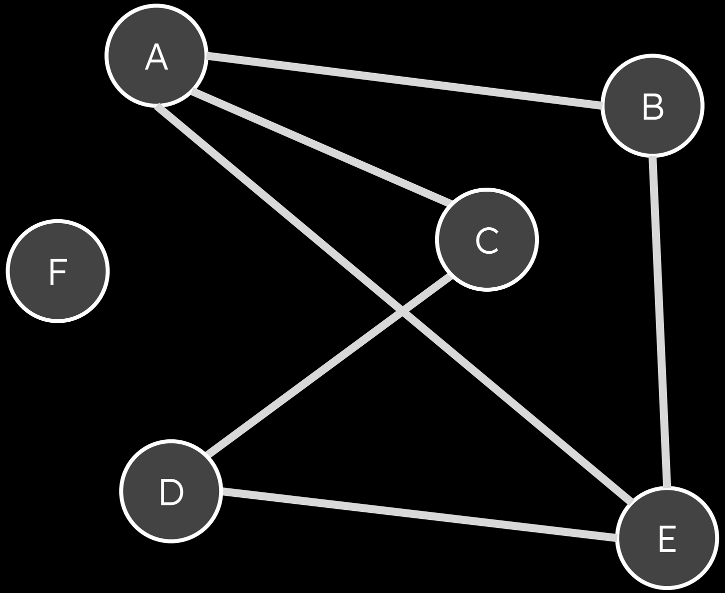
Graph Terminology

the **edges incident** on A



Graphs Representations

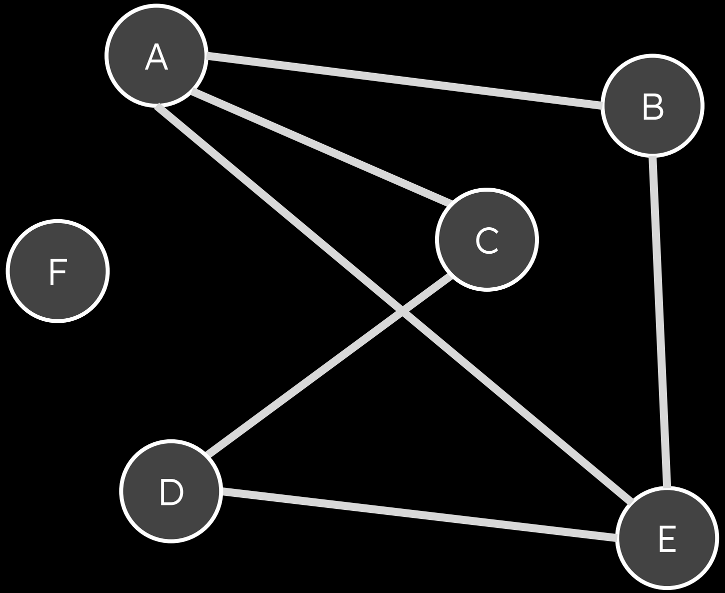
Adjacency List



- Map each node to a list of other nodes that are adjacent to it.

Graphs Representations

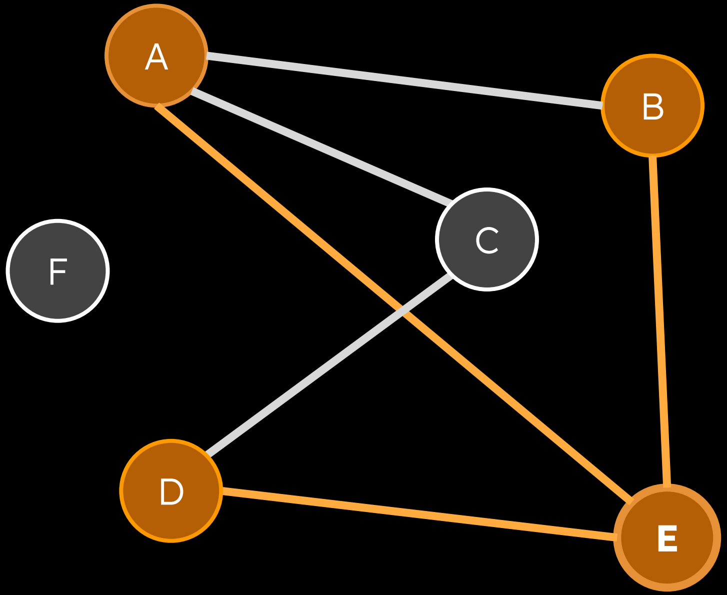
Adjacency List



```
A → [B, C, D]
B → [A, E]
C → [A, D]
D → [C, E]
E → [A, B, D]
F → []
```

Graphs Representations

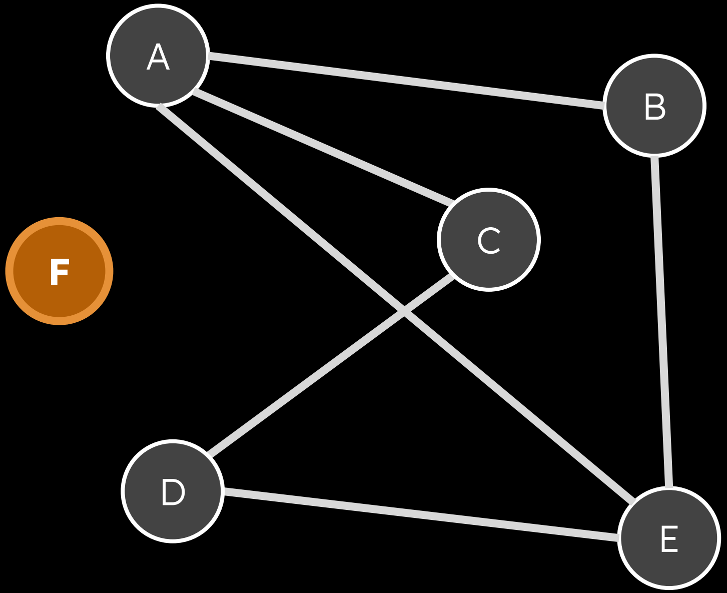
Adjacency List



```
A → [B, C, D]
B → [A, E]
C → [A, D]
D → [C, E]
E → [A, B, D]
F → []
```

Graphs Representations

Adjacency List



```
A → [B, C, D]
B → [A, E]
C → [A, D]
D → [C, E]
E → [A, B, D]
F → [ ]
```

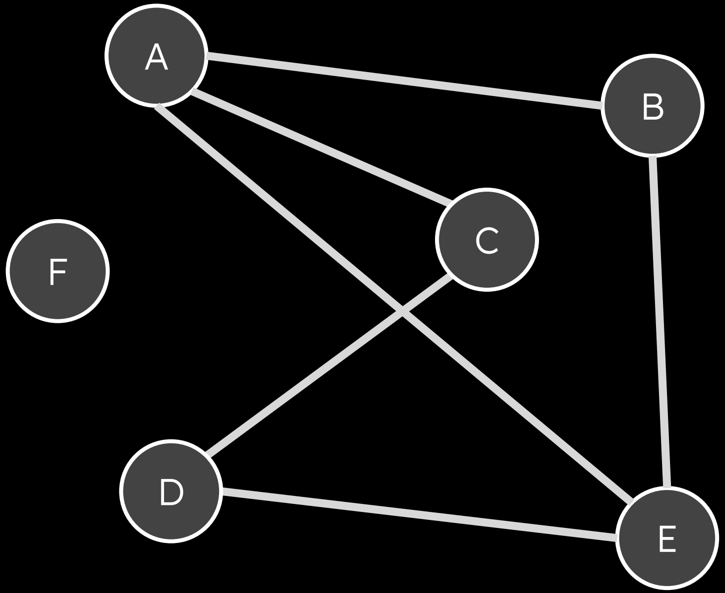
Graphs Representations

Adjacency List

Operation	Time
Construction	$\Theta(V + E)$
Get adjacent vertices	$\Theta(\deg(v_i))$
Check if vertices are adjacent	$\Theta(\min(\deg(v_i), \deg(v_{\square})))$

Graphs Representations

Adjacency Matrix

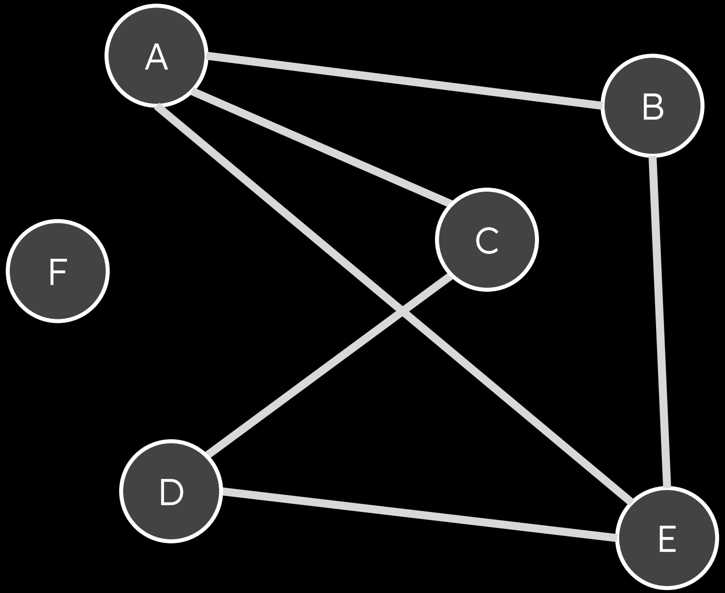


- Map vertex pairs (u, v) to 1 or 0, depending on if there is an edge between them.
- Typically, we use a two dimensional array *G* and set

  G[u][v] = 1 if (u, v) in E
  G[u][v] = 0 otherwise

Graphs Representations

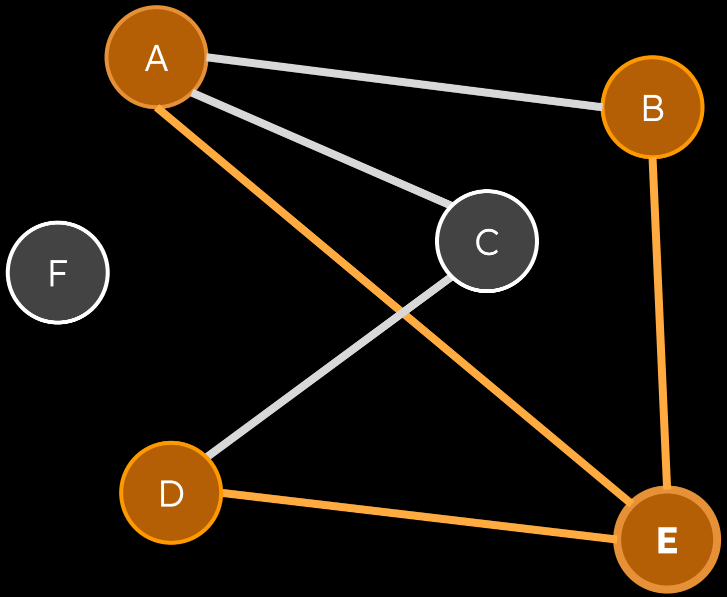
Adjacency Matrix



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

Graphs Representations

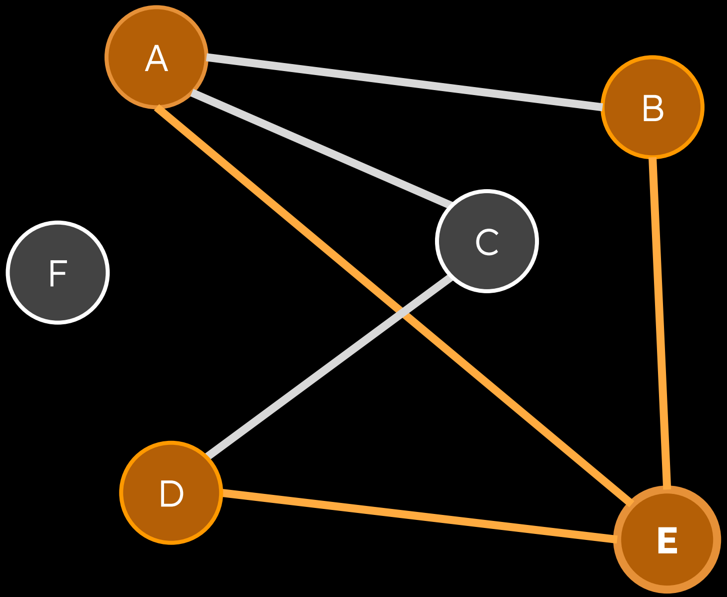
Adjacency Matrix



$$\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 1 & 1 & 0 & 1 & 0 \\
B & 1 & 0 & 0 & 0 & 1 & 0 \\
C & 1 & 0 & 0 & 1 & 0 & 0 \\
D & 0 & 0 & 1 & 0 & 1 & 0 \\
\boxed{E} & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
F & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Graphs Representations

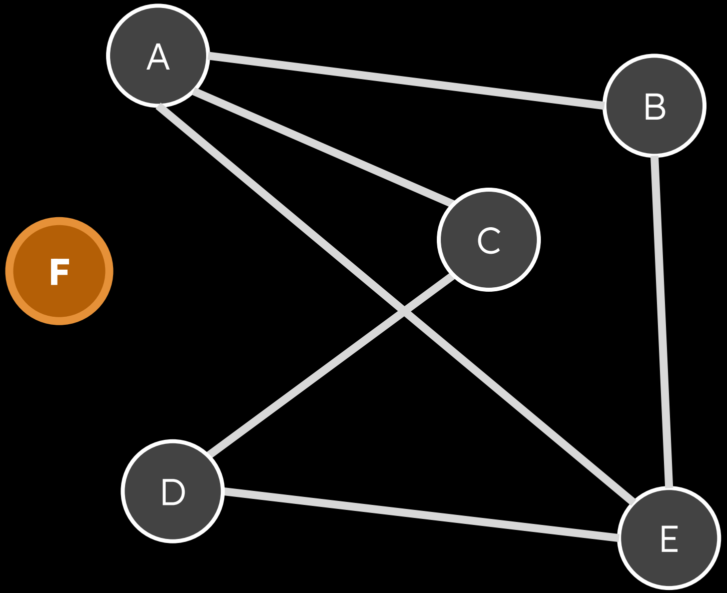
Adjacency Matrix



$$\begin{array}{c|cccccc} & A & B & C & D & \boxed{E} & F \\ \hline A & 0 & 1 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 0 & 0 & 1 & 0 \\ C & 1 & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 0 & 1 & 0 & 1 & 0 \\ E & 1 & 1 & 0 & 1 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Graphs Representations

Adjacency Matrix



$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 0 & 0 & 1 & 0 \\ C & 1 & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 0 & 1 & 0 & 1 & 0 \\ E & 1 & 1 & 0 & 1 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Graphs Representations

Adjacency Matrix

Operation	Time
Construction	$\Theta(V^2)$
Get adjacent vertices	$\Theta(V)$
Check if vertices are adjacent	$\Theta(1)$

Graphs Representations

Comparison

Adjacency List

- Faster neighbor retrieval
- Uses less memory
- More straightforward implementation

Adjacency Matrix

- Faster adjacency check
- Cool linear algebra

Graphs Representations

Comparison

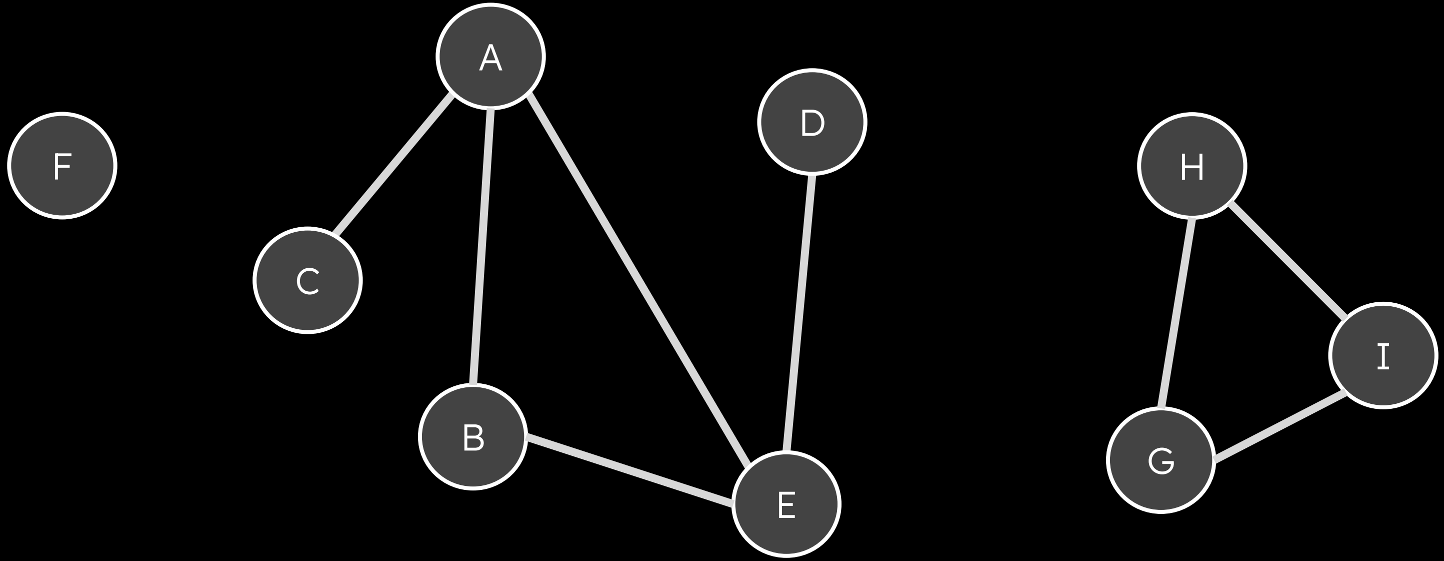
Adjacency List

- **Faster neighbor retrieval**
- Uses less memory
- **More straightforward implementation**

Adjacency Matrix

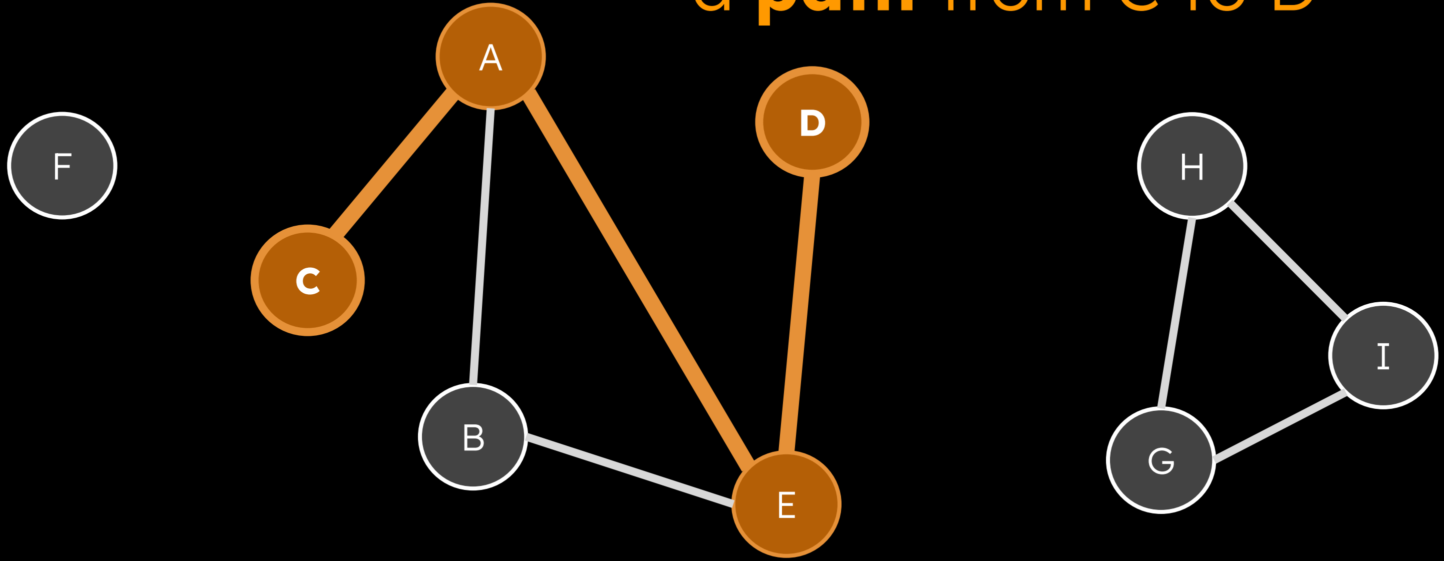
- Faster adjacency check
- Cool linear algebra

Graph Terminology



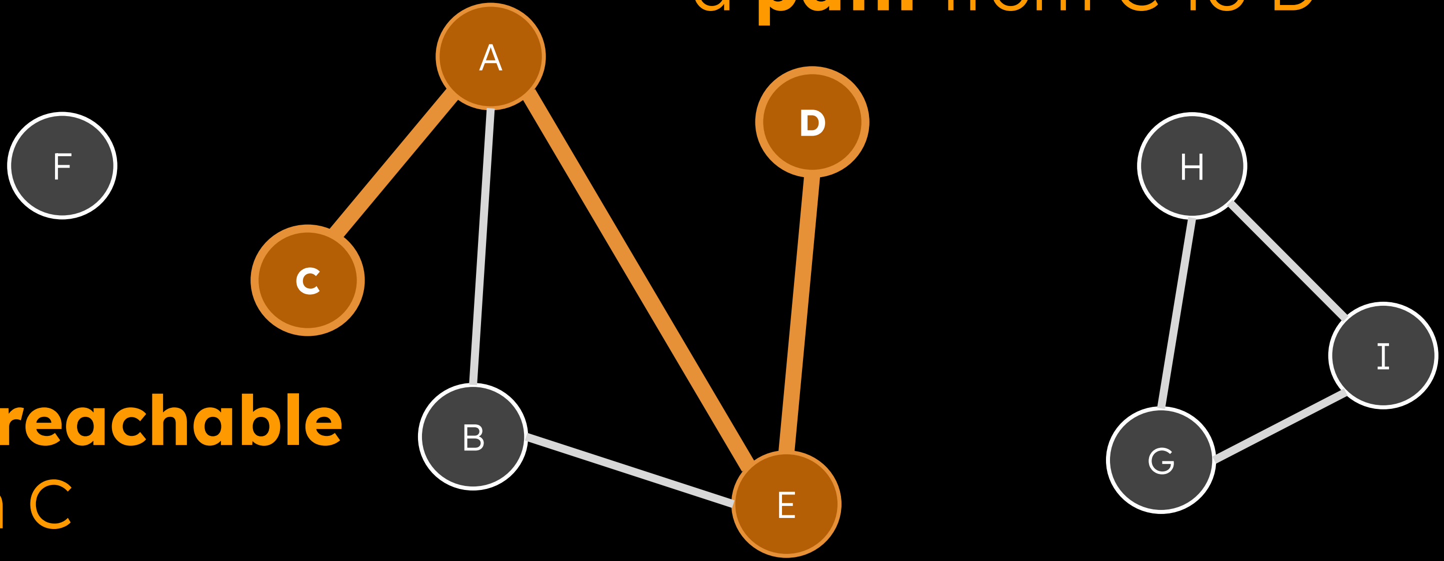
Graph Terminology

a **path** from C to D



Graph Terminology

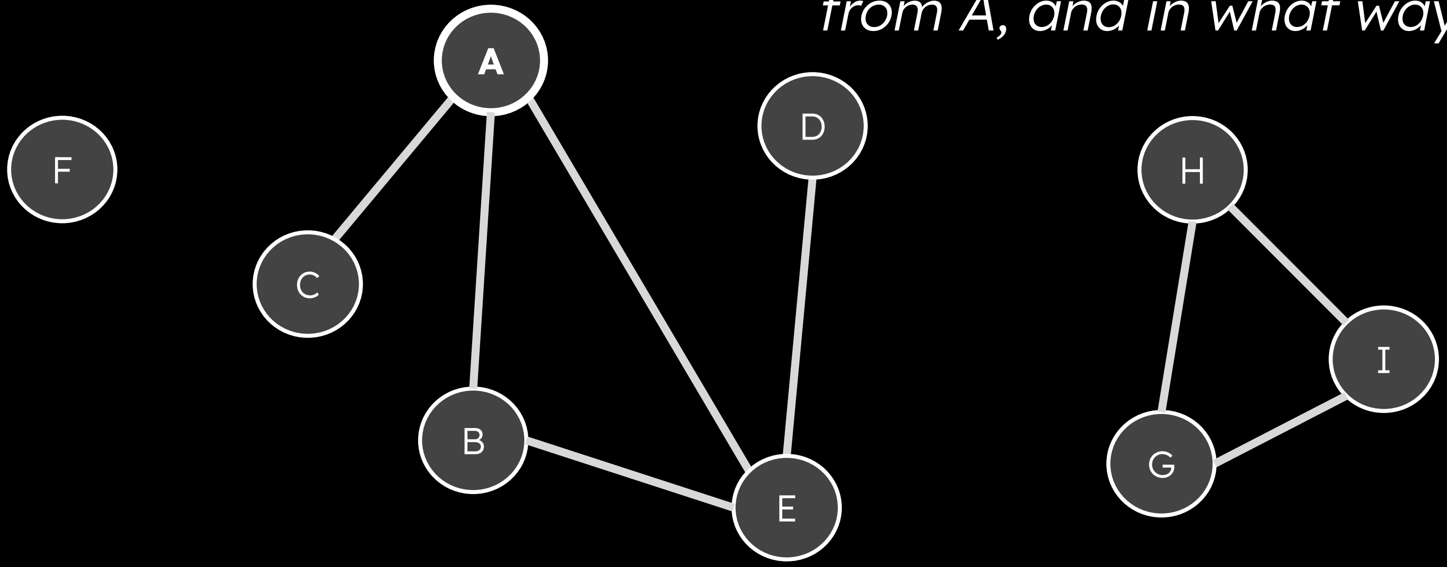
a **path** from C to D



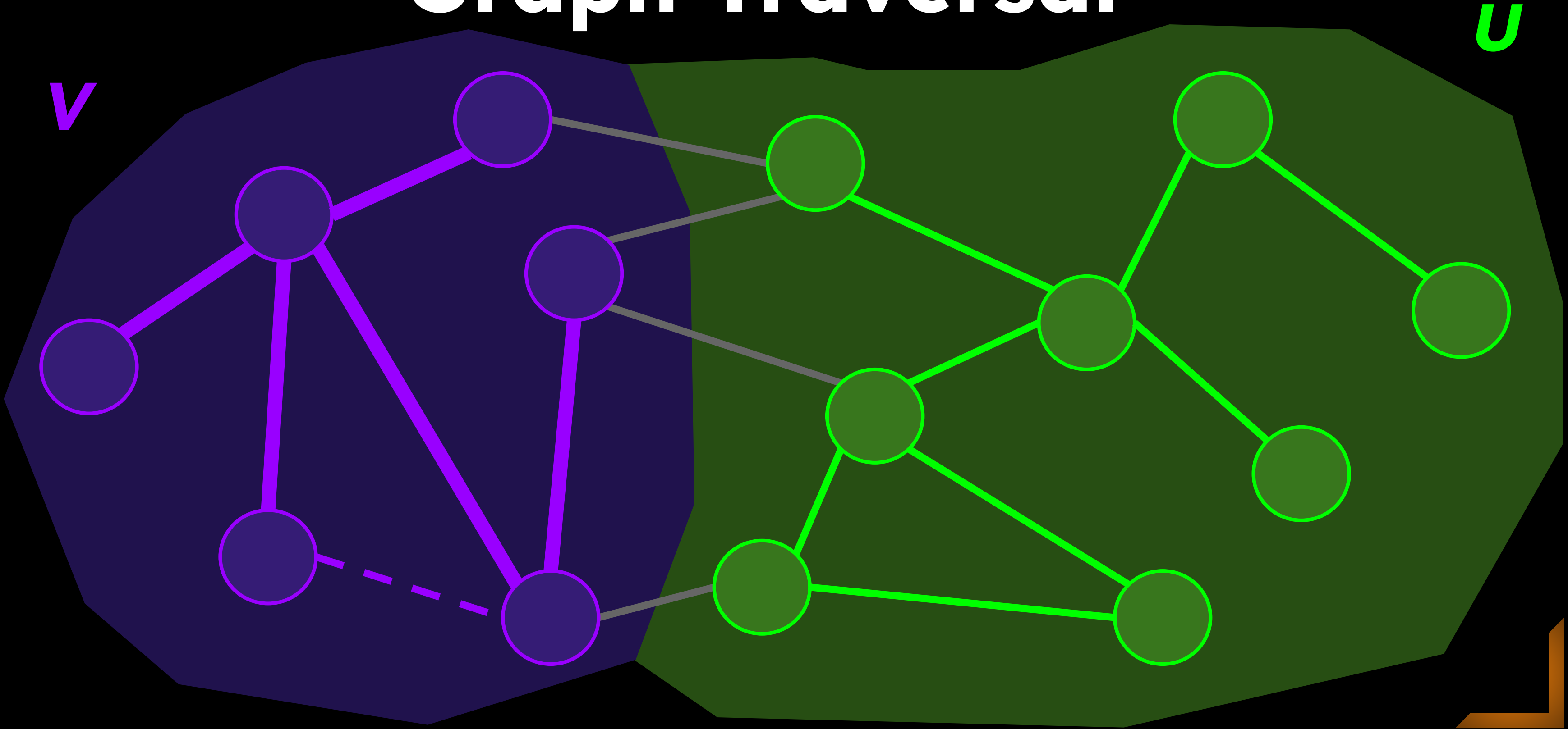
D is **reachable** from C

Graph Terminology

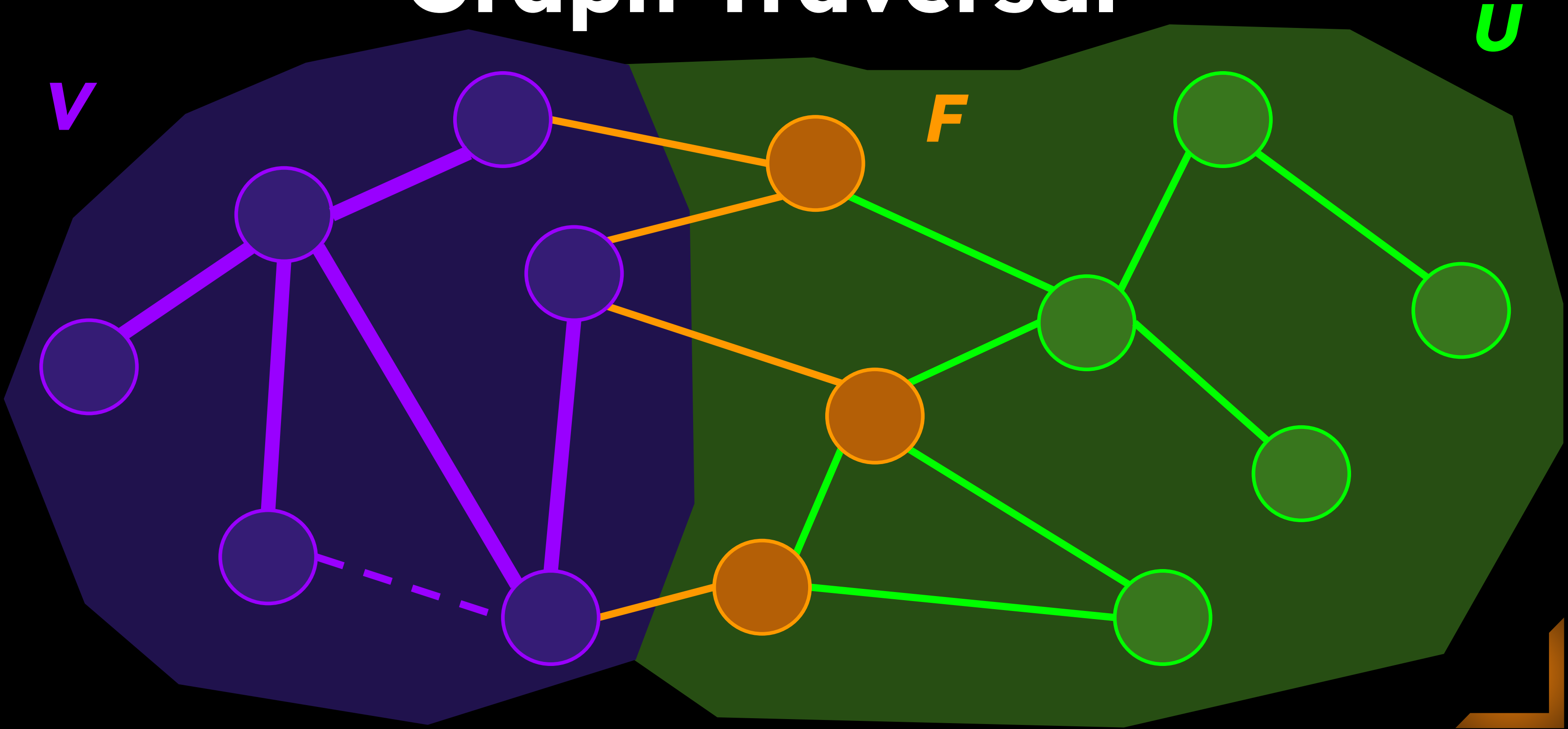
Which nodes are reachable from A, and in what way?



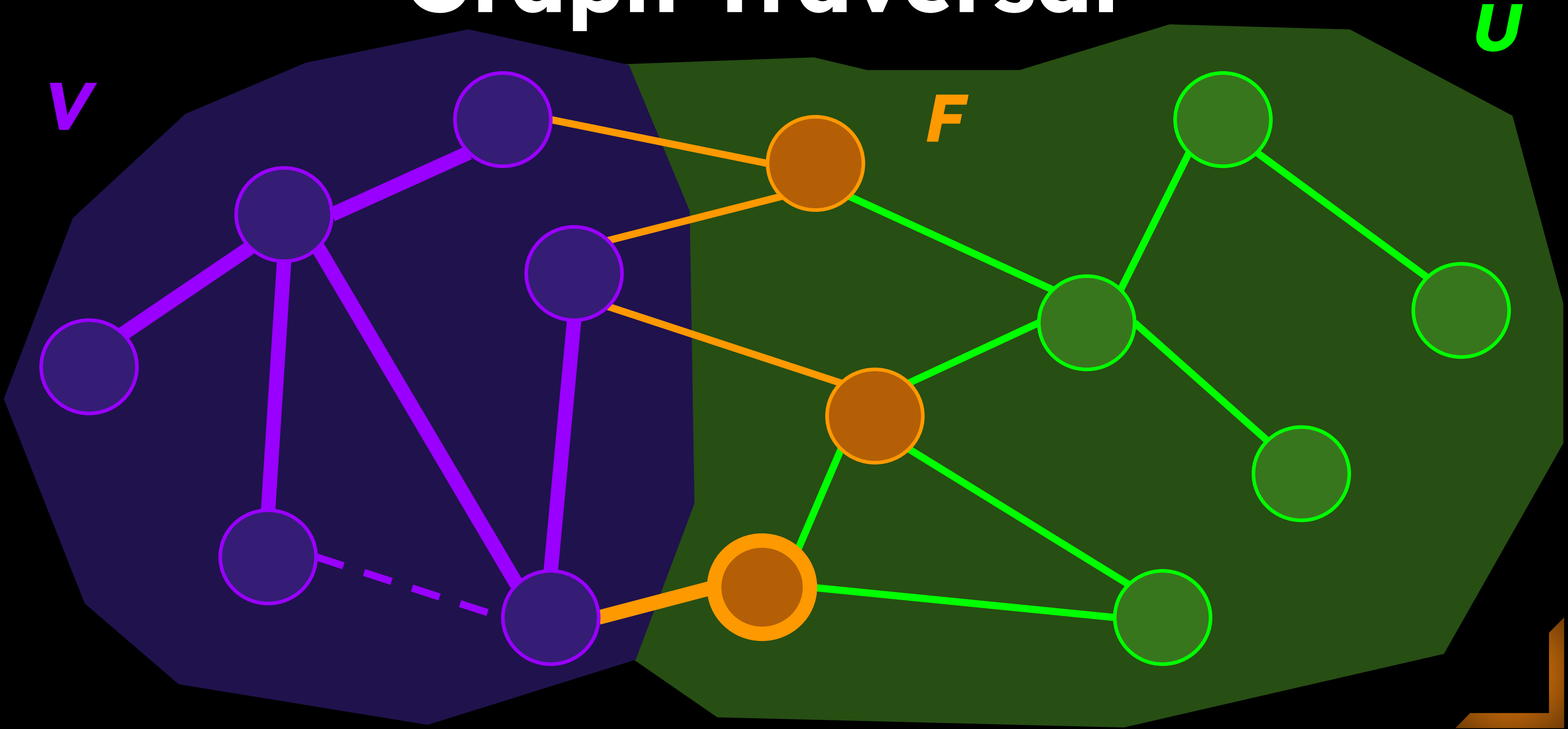
Graph Traversal



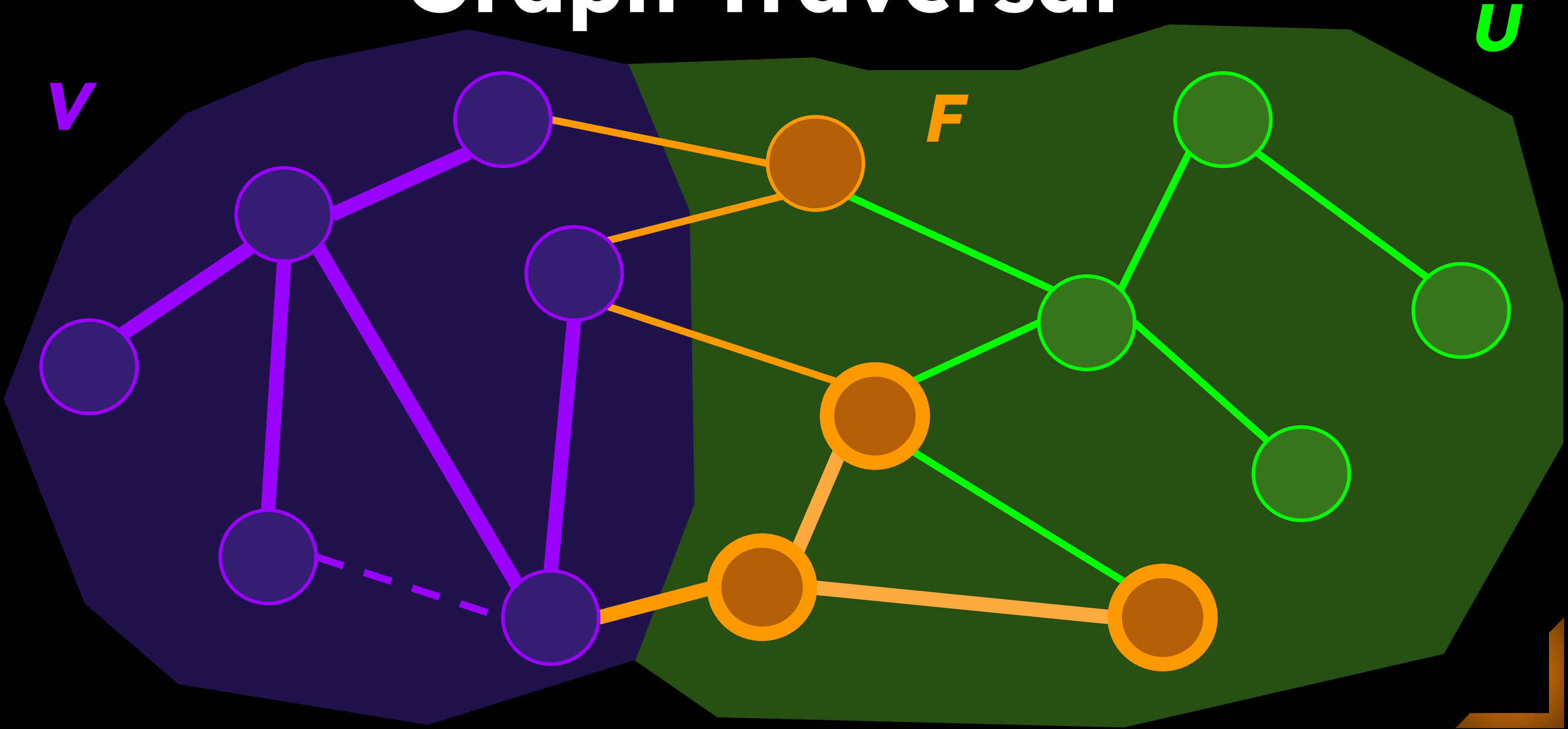
Graph Traversal



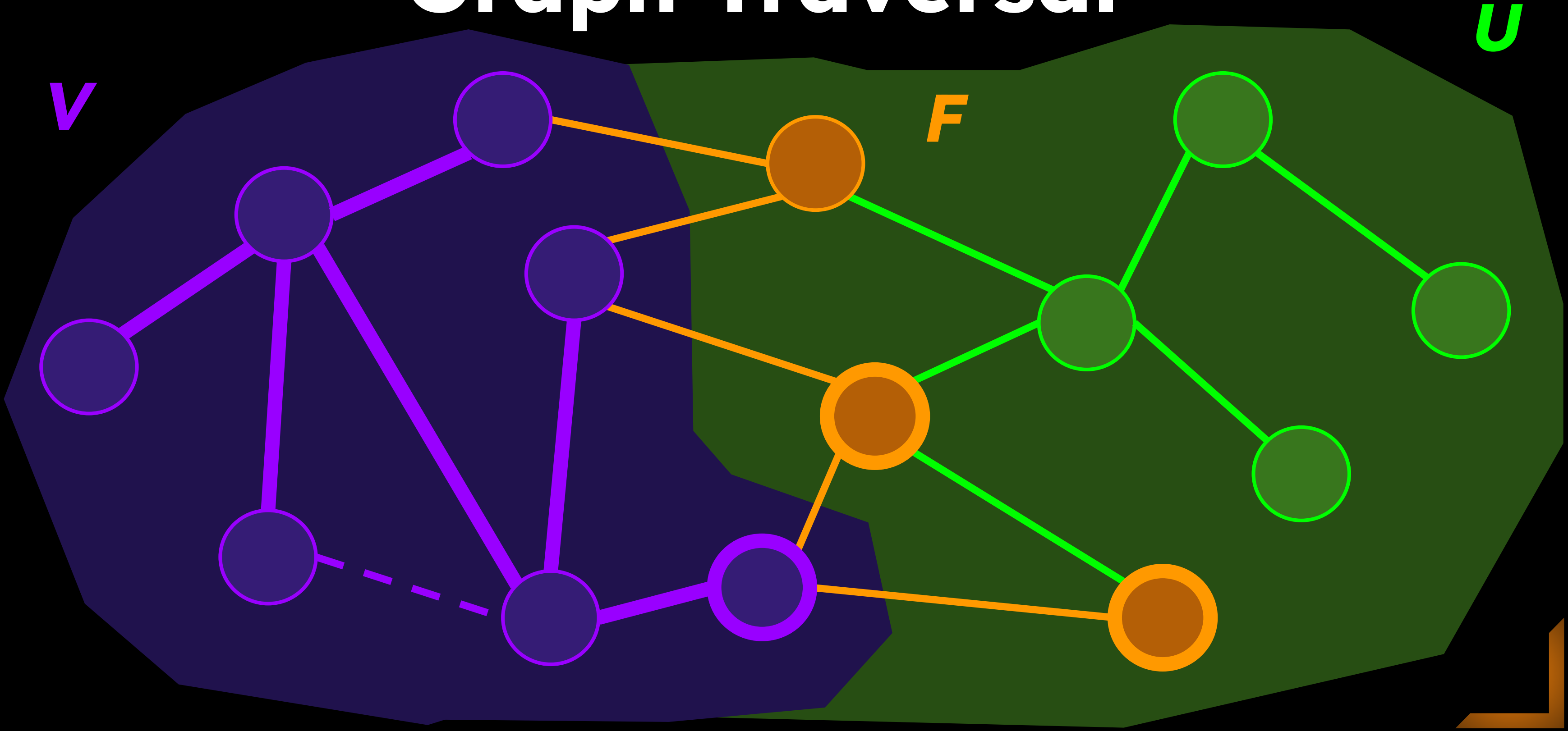
Graph Traversal



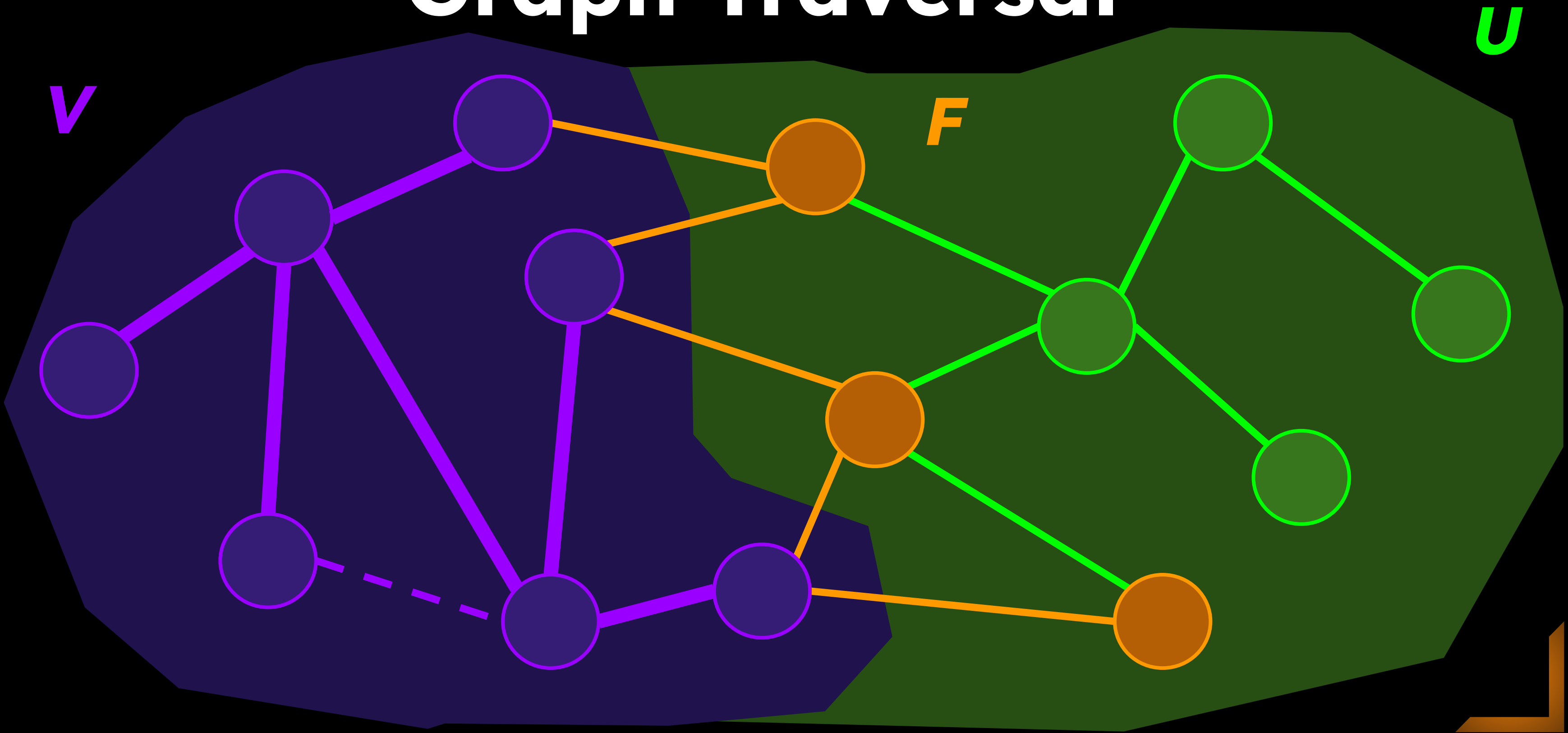
Graph Traversal



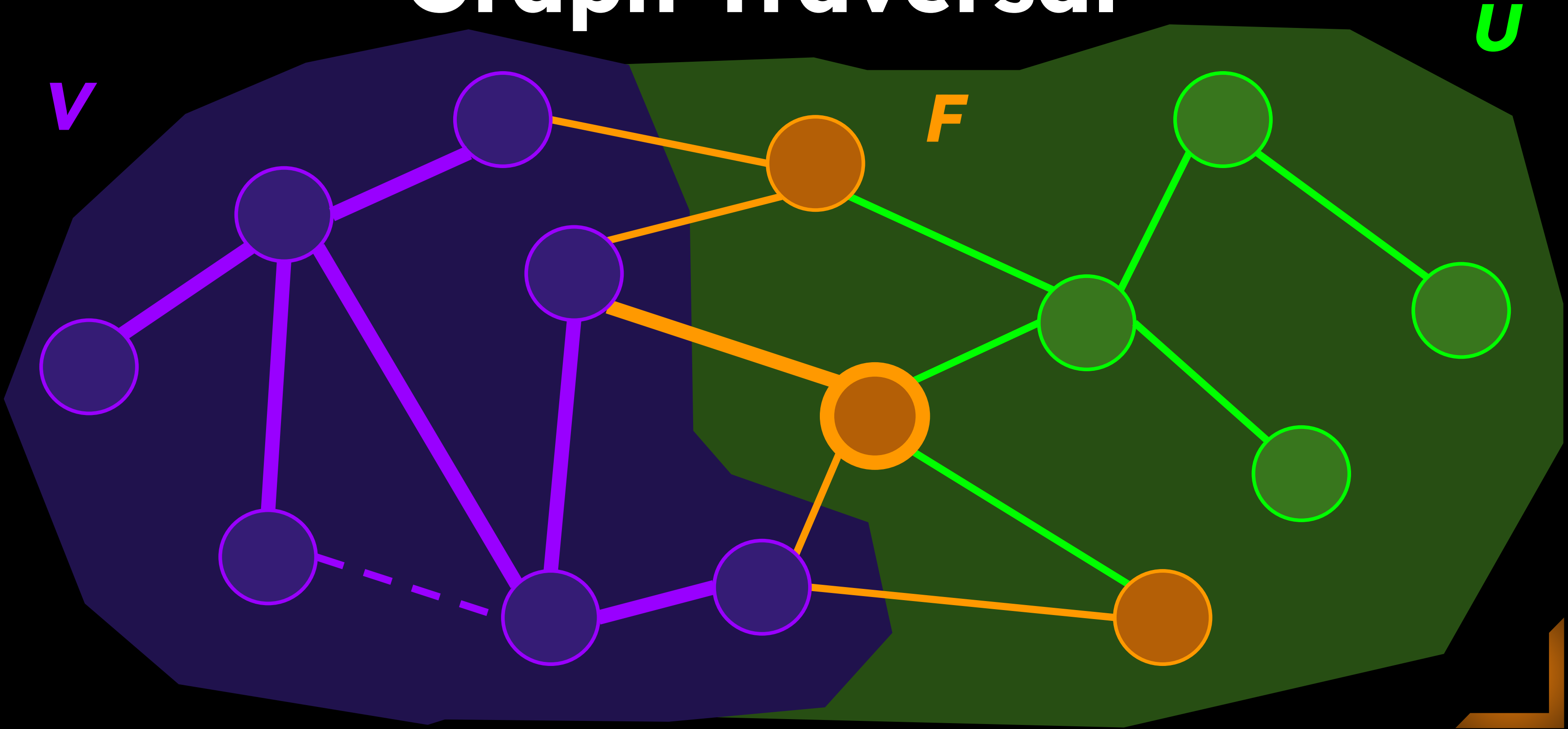
Graph Traversal



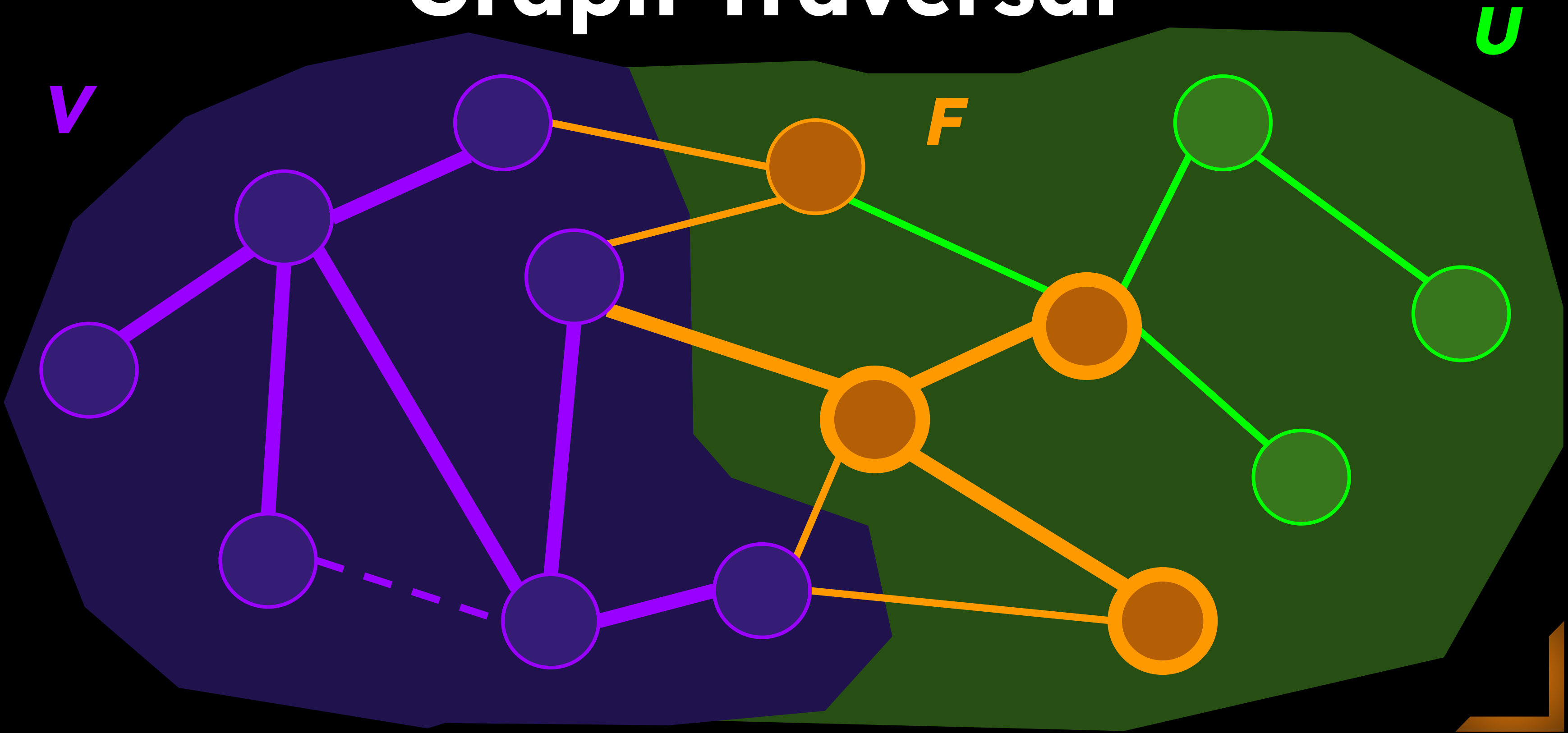
Graph Traversal



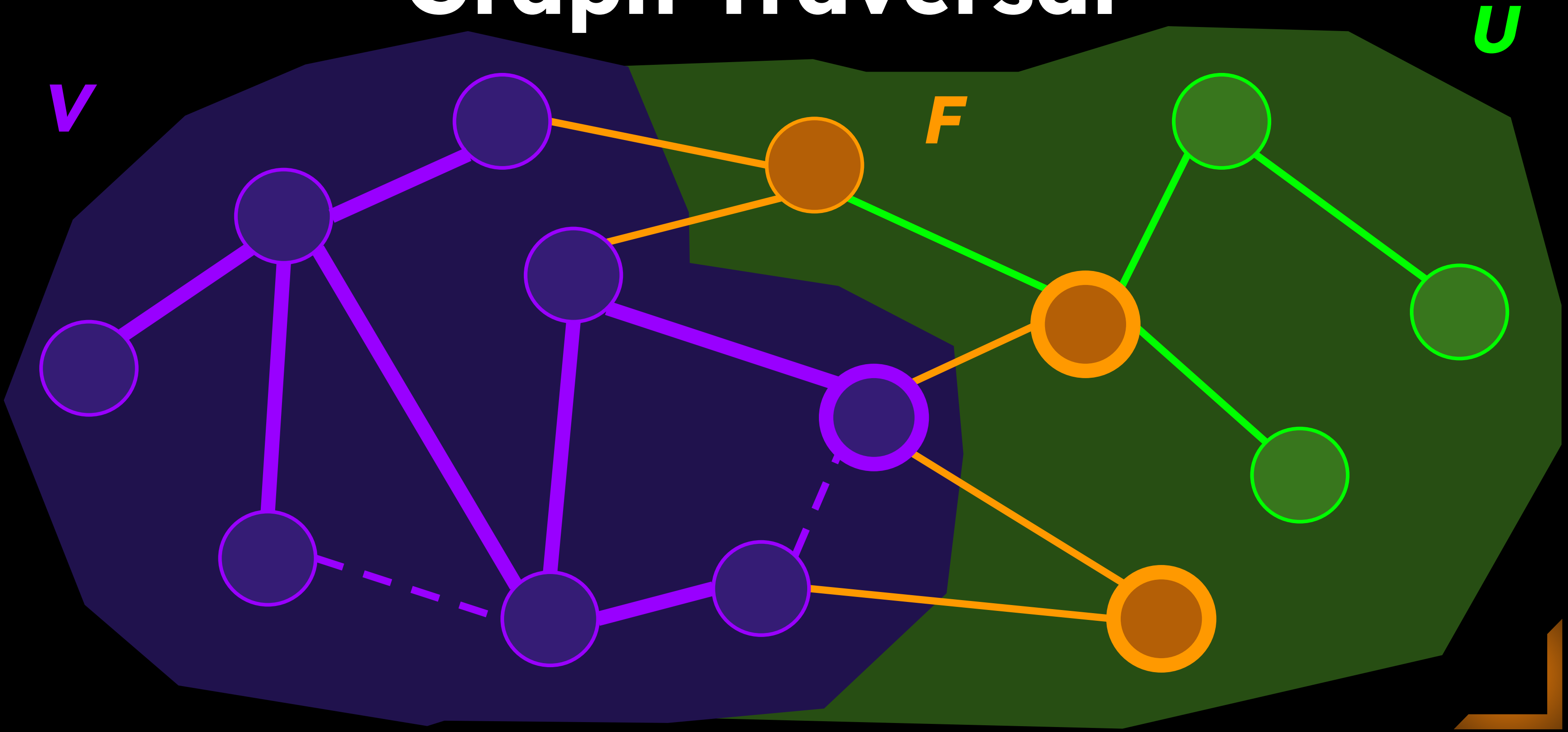
Graph Traversal



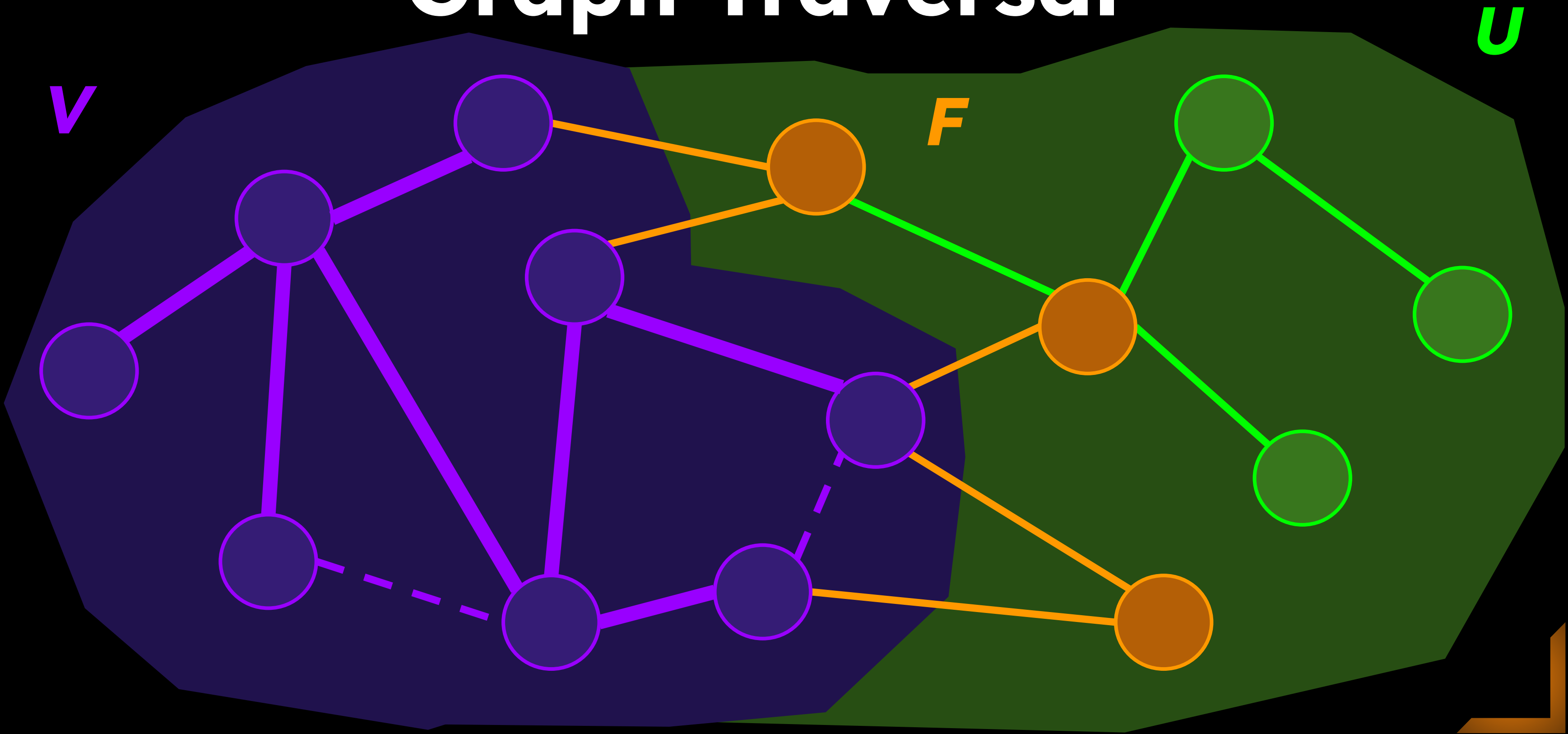
Graph Traversal



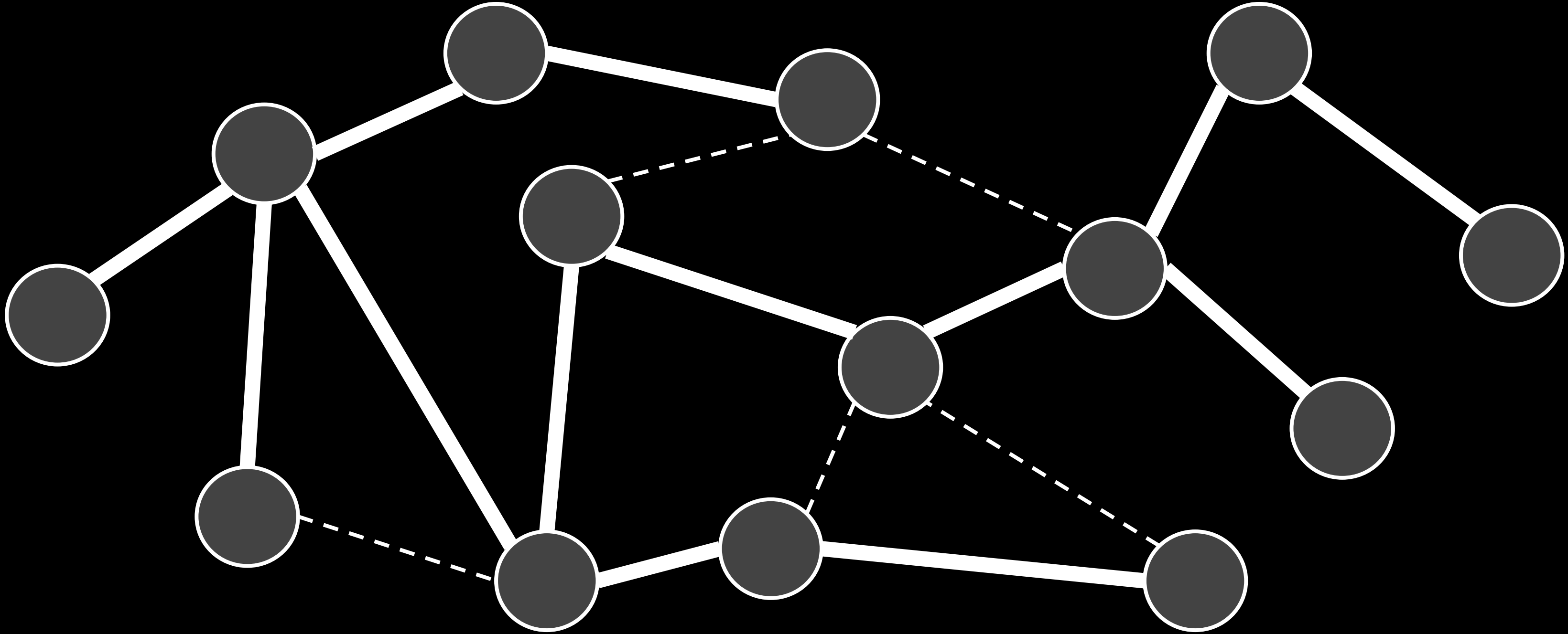
Graph Traversal



Graph Traversal



Graph Traversal

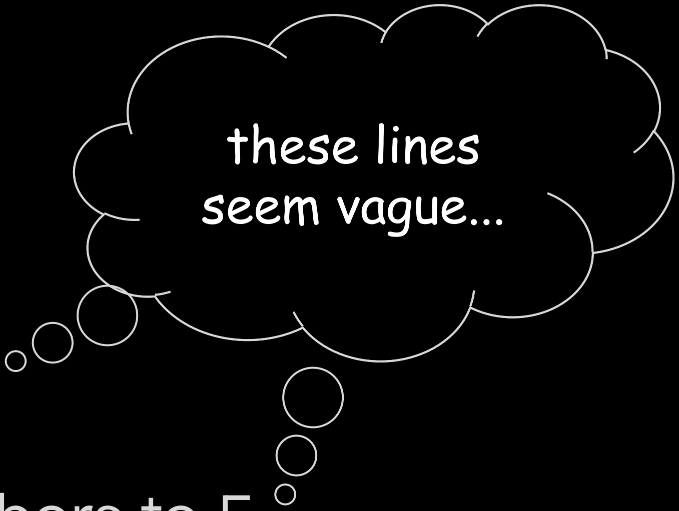


Graph Traversal

```
function traverse(G, s)
    F = { s }
    V = {}
    while |F| > 0
        remove v from F
        if v not in V
            add v's neighbors to F
            process v
            add v to V
```

Graph Traversal

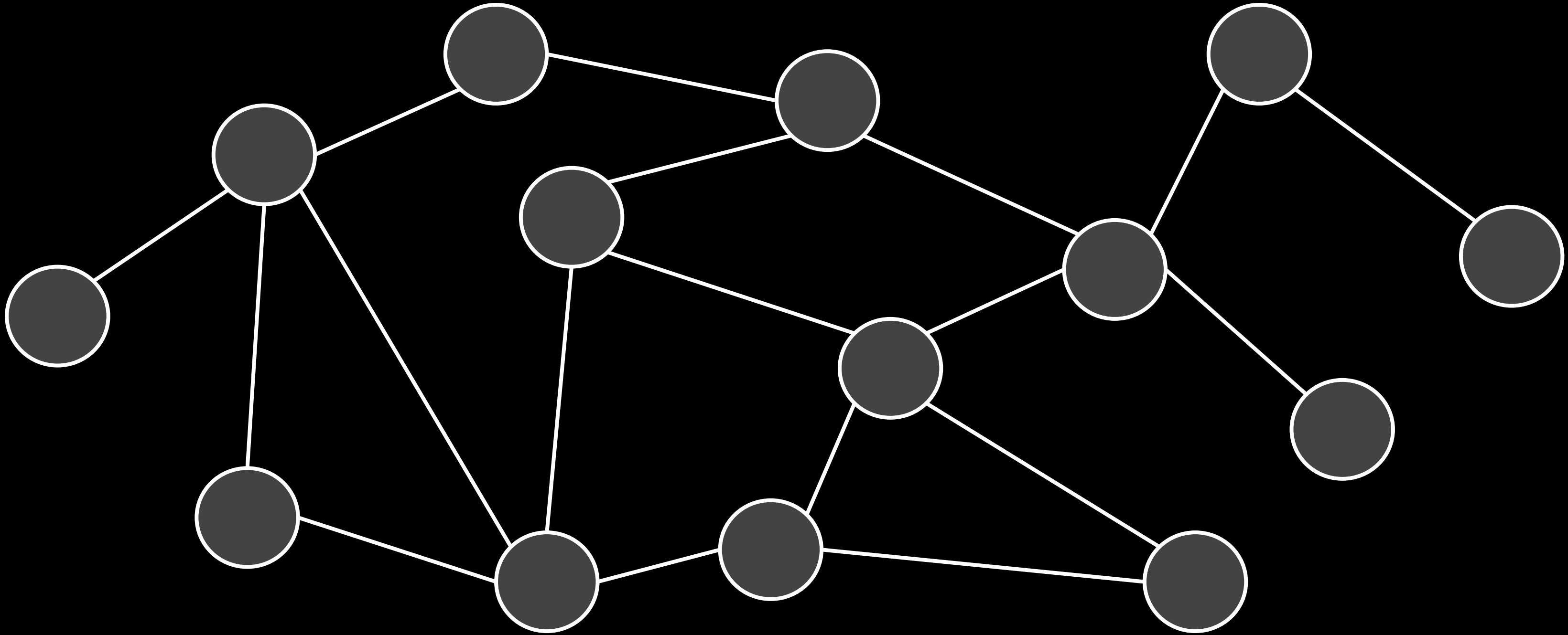
```
function traverse(G, s)
    F = { s }
    V = {}
    while |F| > 0
        remove v from F
        if v not in V
            add v's neighbors to F
            process v
            add v to V
```



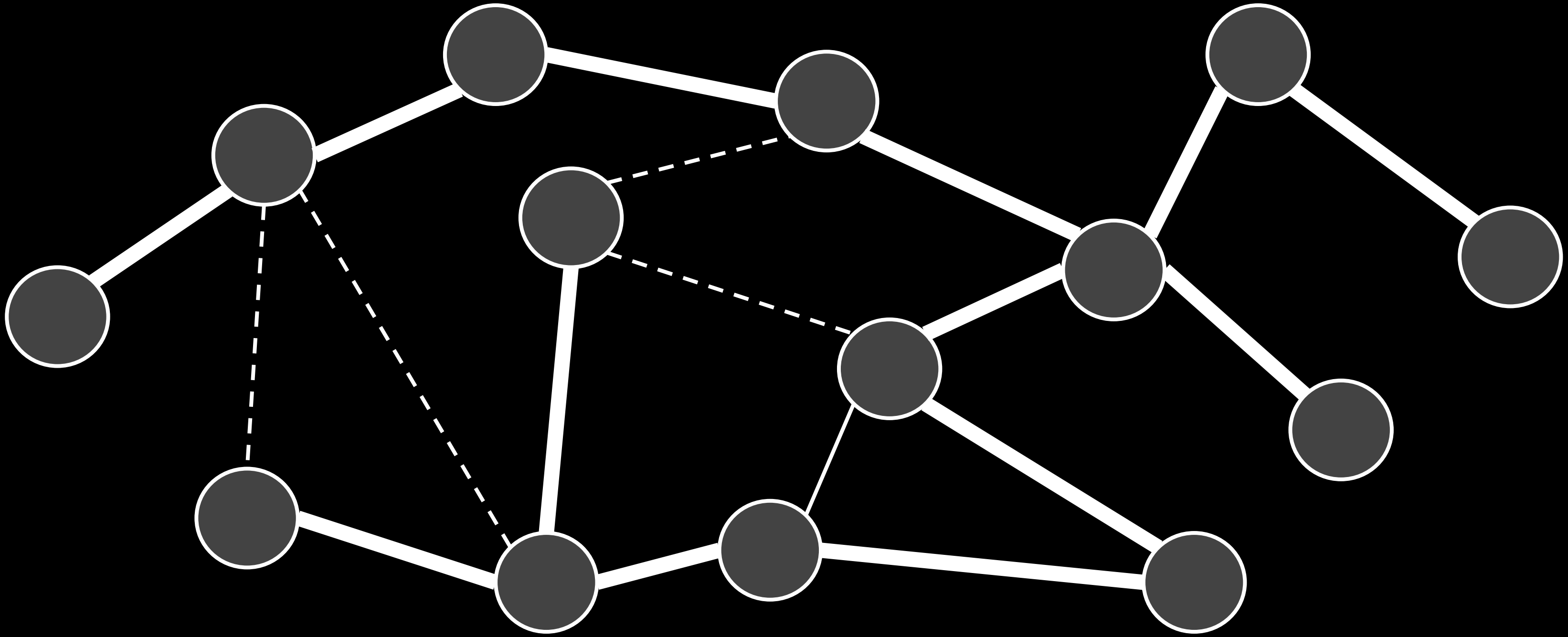
Depth-First Search

```
function traversal(G, s)
    F = new stack [s]
    V = {}
    while |F| > 0
        v = F.pop()
        if v not in V
            for each vertex u adjacent to v
                F.push(u)
            process v
            add v to V
```

Depth-First Search



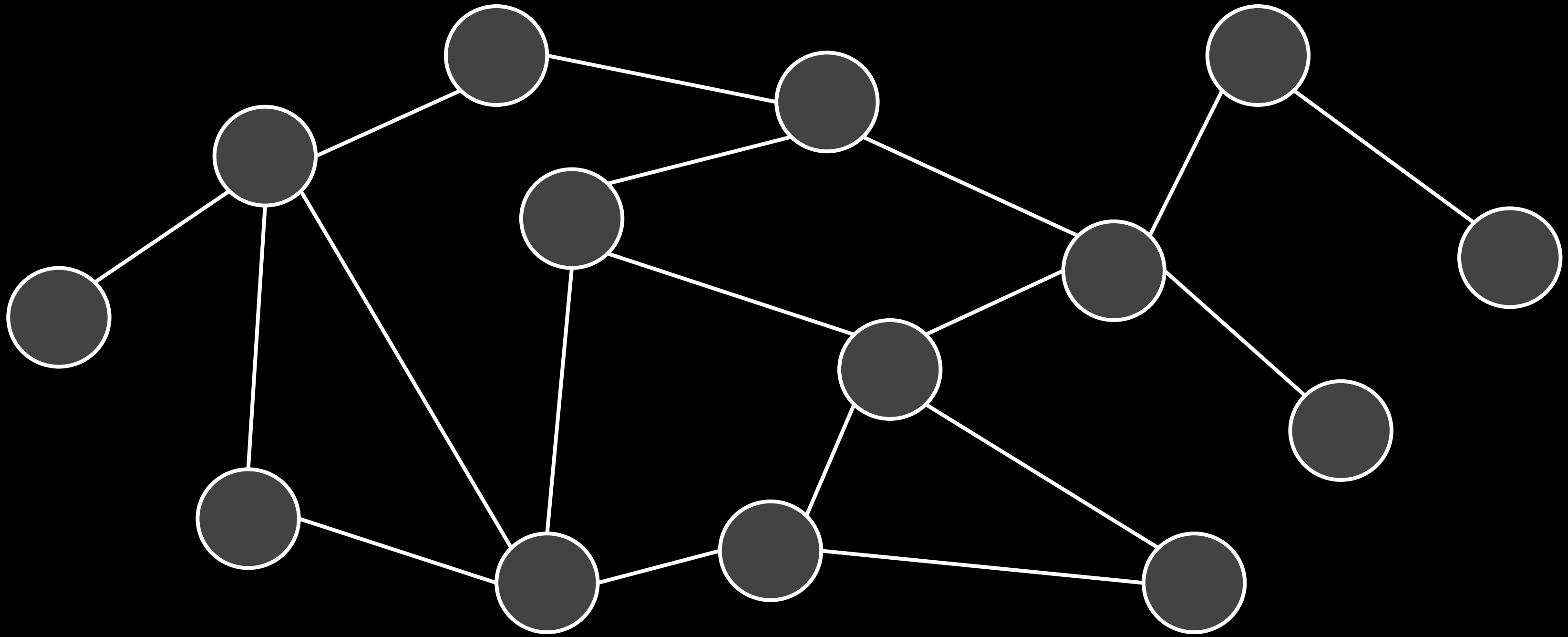
Depth-First Search



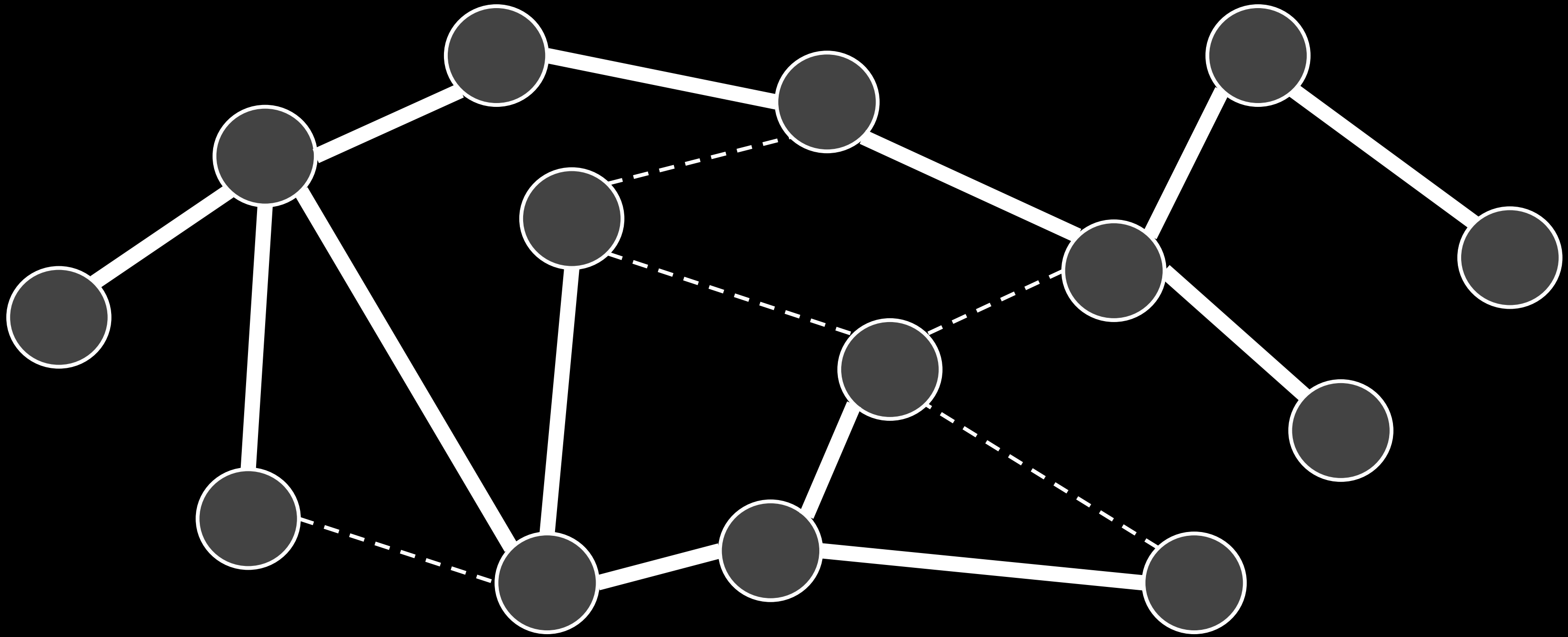
Breadth-First Search

```
function traversal(G, s)
    F = new queue [s]
    V = {}
    while |F| > 0
        v = F.dequeue()
        if v not in V
            for each vertex u adjacent to v
                F.enqueue(u)
            process v
            add v to V
```

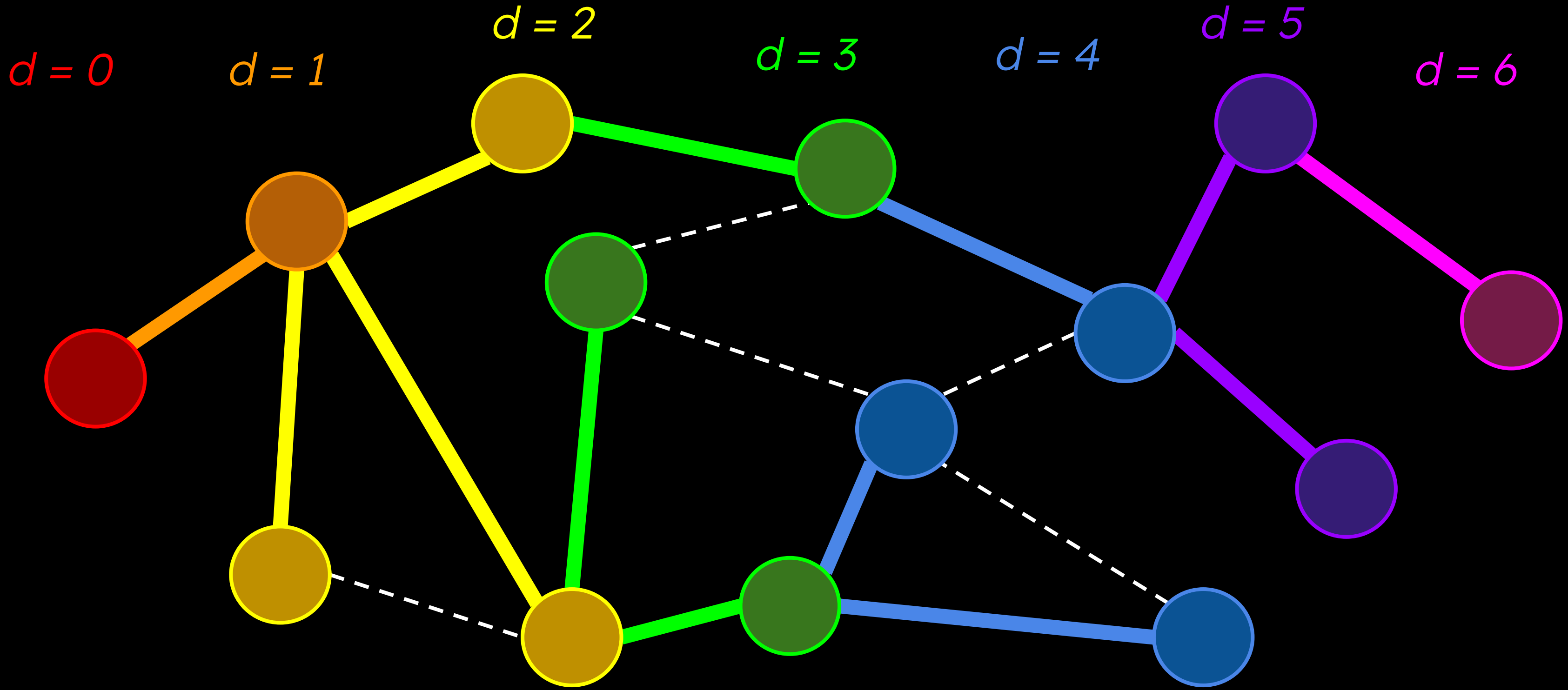
Breadth-First Search



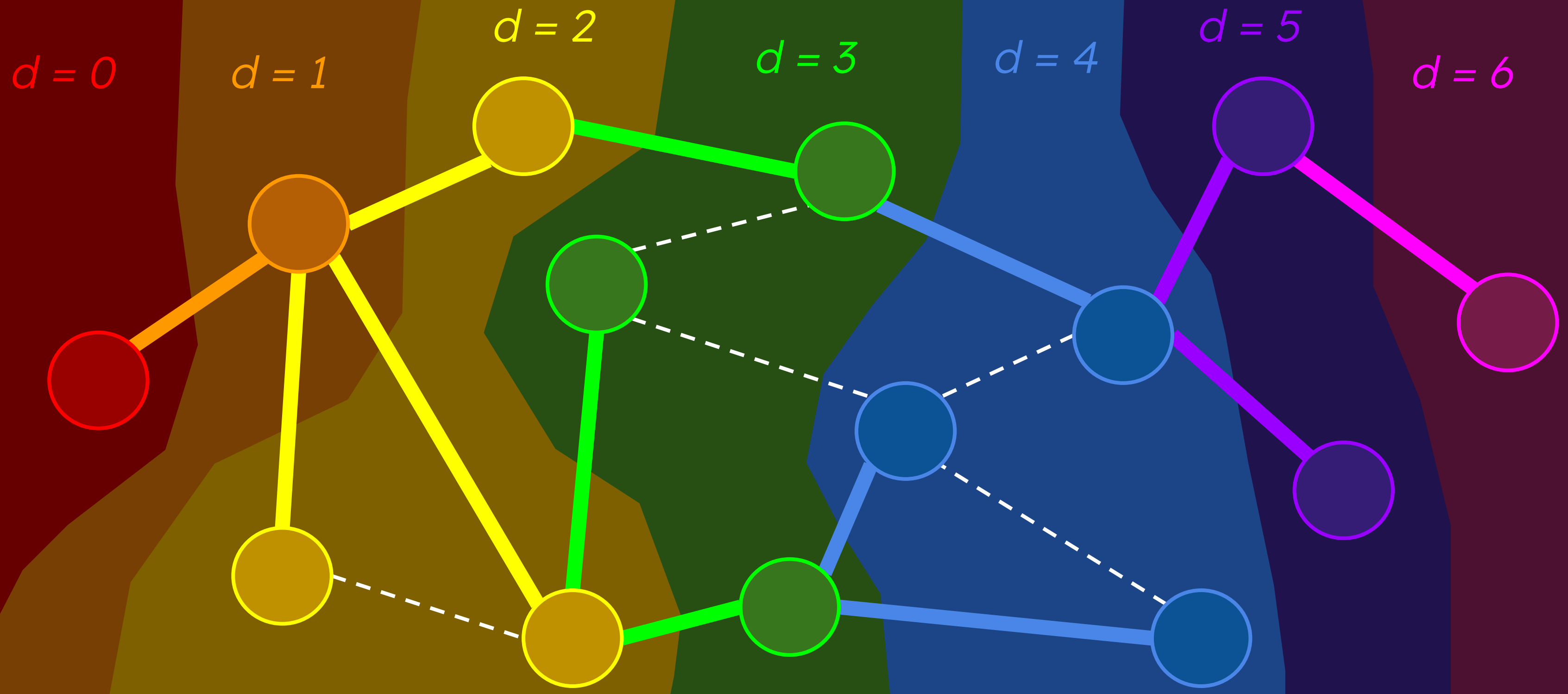
Breadth-First Search



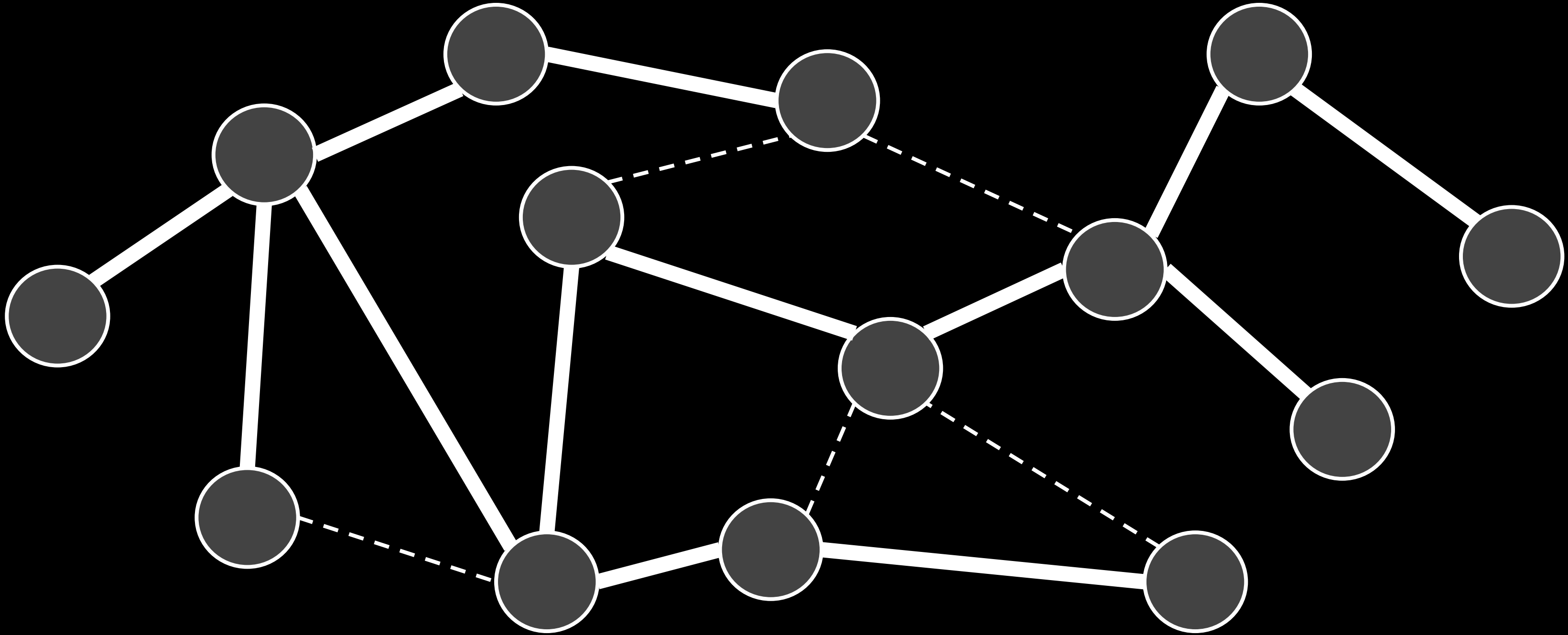
Breadth-First Search



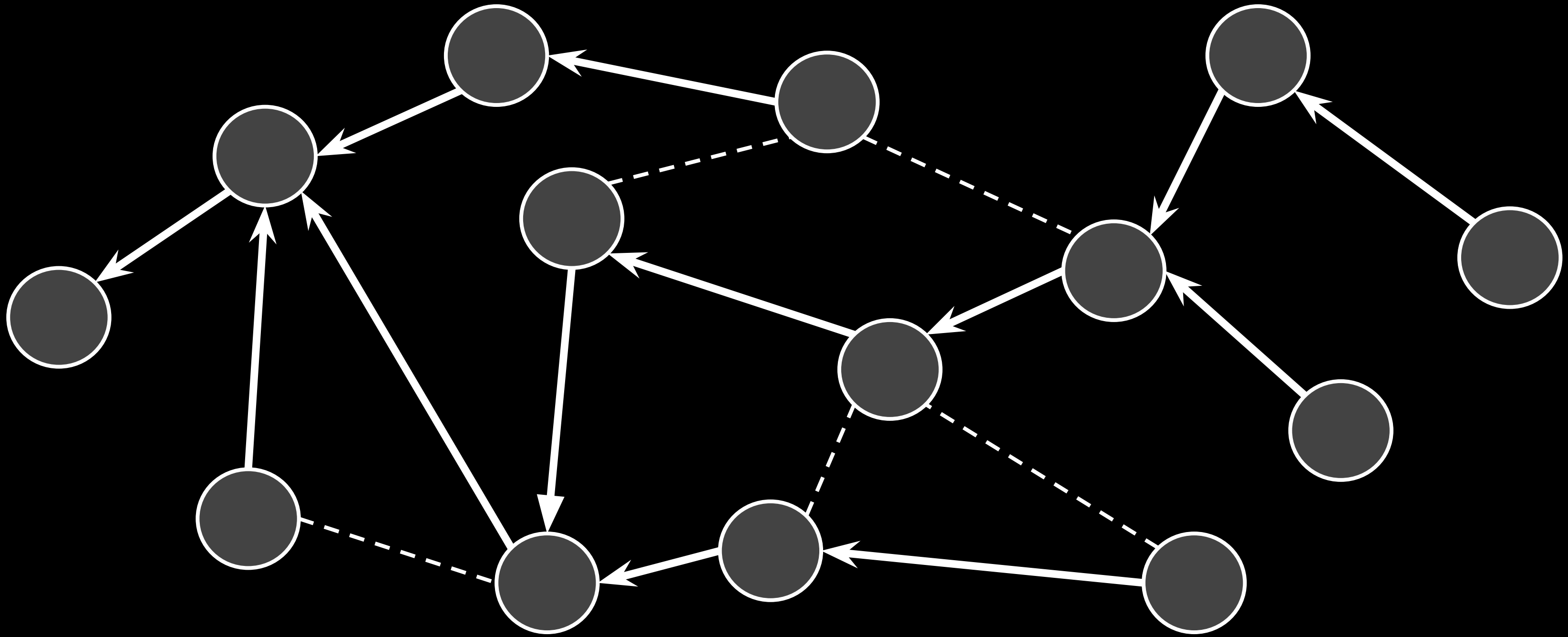
Breadth-First Search



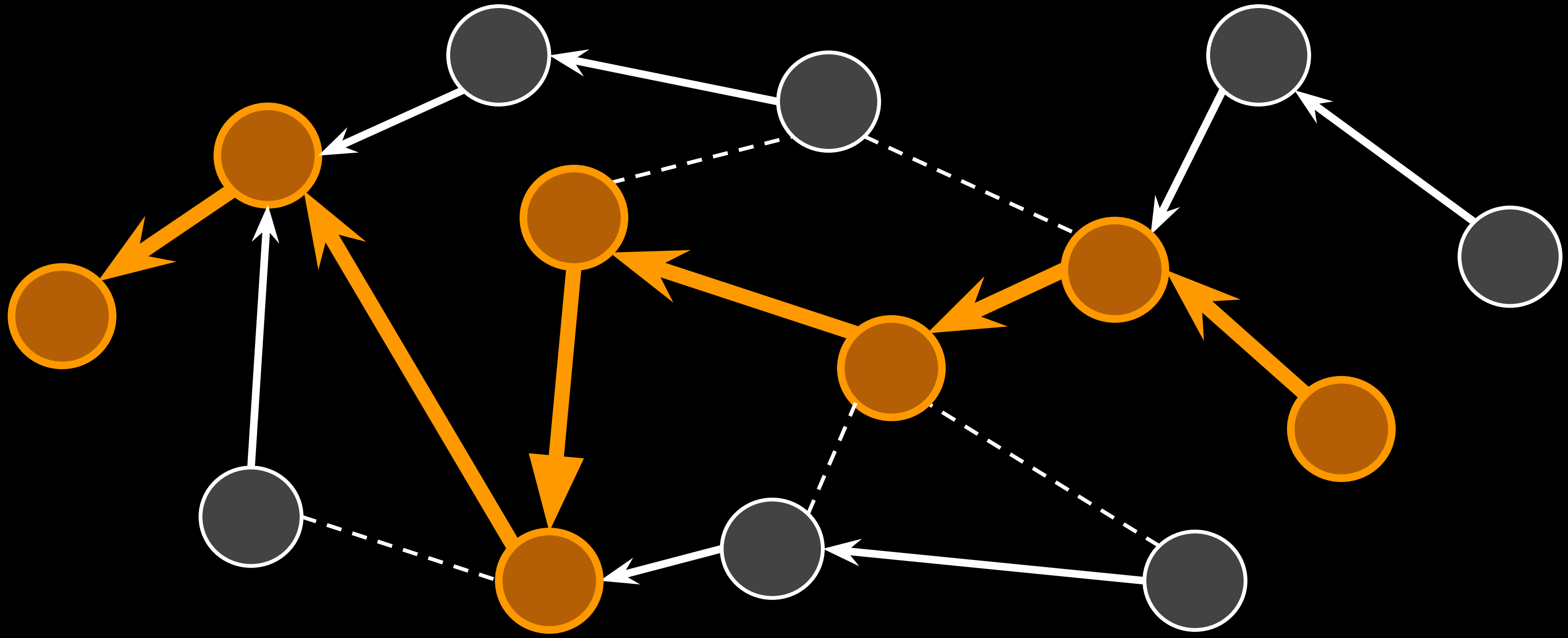
Path Reconstruction



Path Reconstruction



Path Reconstruction



Path Reconstruction

```
function traversal(G, s)
    F = new queue [(null, s)]
    V = {}
    P = empty map                    (parent pointers)
    while |F| > 0
        (p, v) = F.dequeue()
        if v not in V
            for each vertex u adjacent to v
                F.enqueue((v, u))
            P[v] = p                 (set the parent)
            add v to V
```

Path Reconstruction

```
function get-path(P, dest)
    path = new list
    x = dest
    while $x \neq$ null
        add x to path
        x = P[x]
    reverse path
    return path
```

Implicit Graphs

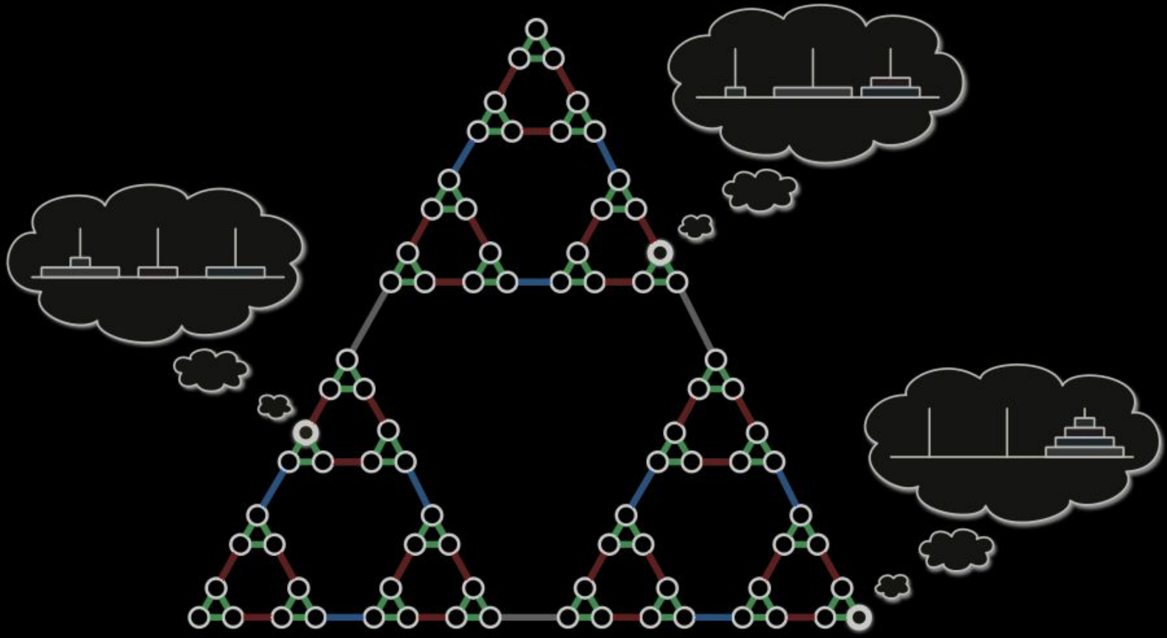


Figure 5.8. The configuration graph of the 4-disk Tower of Hanoi.

Implicit Graphs

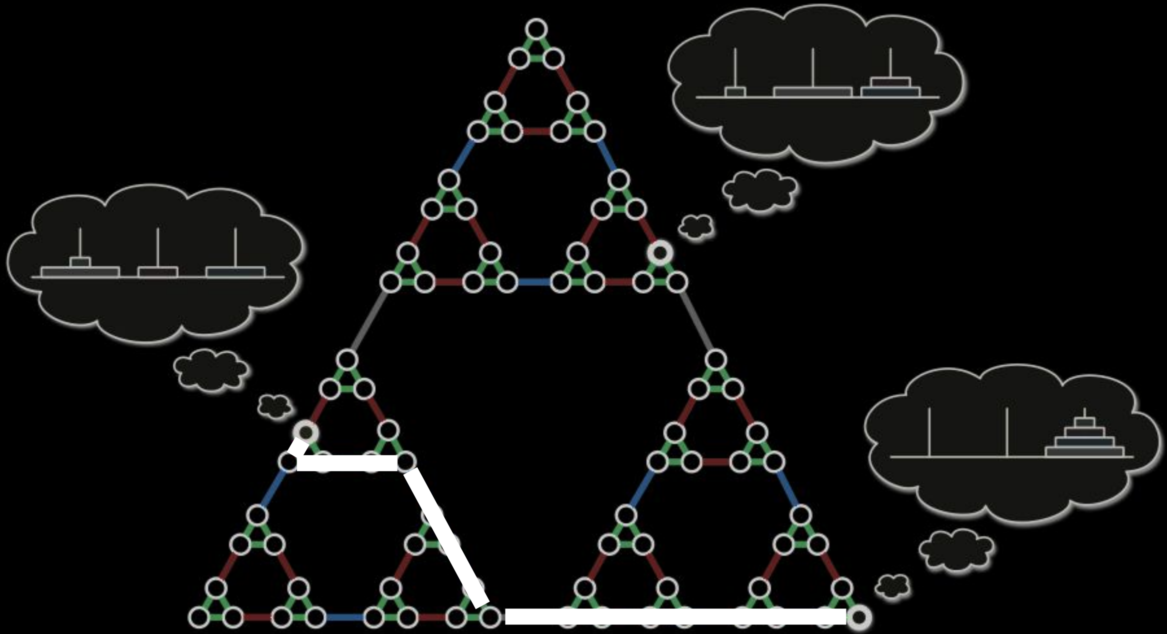


Figure 5.8. The configuration graph of the 4-disk Tower of Hanoi.

Let’s get coding!

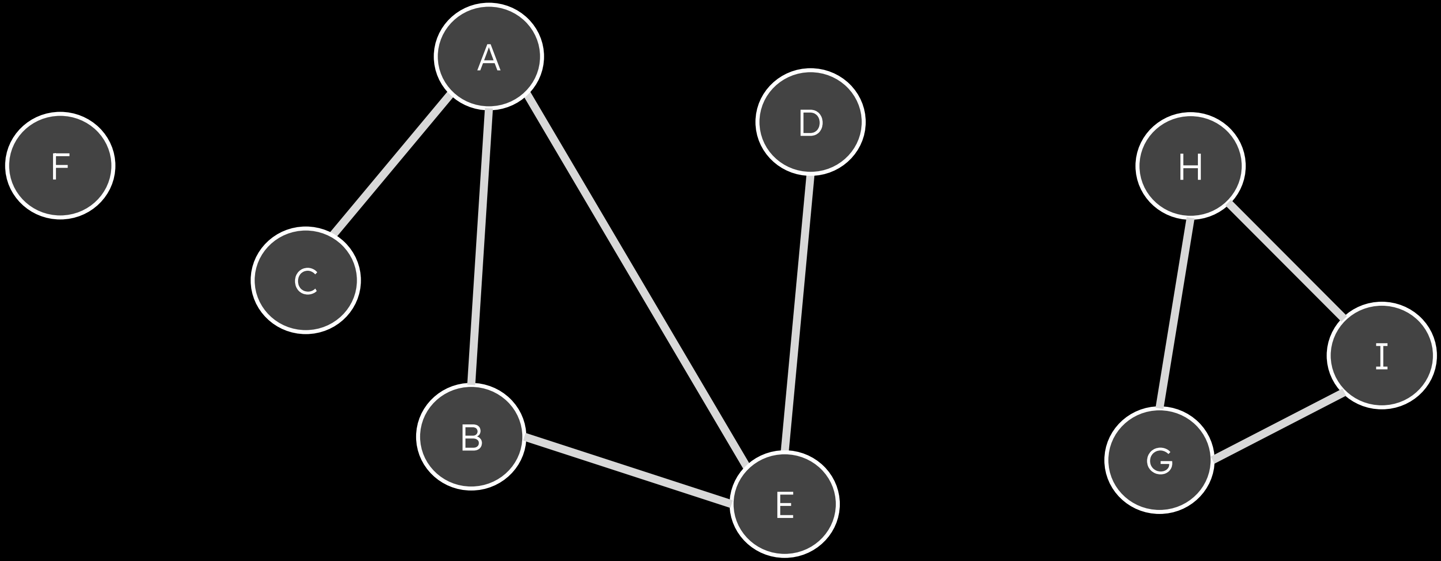
Problems

go.osu.edu/cpcmeetingproblems

Attendance

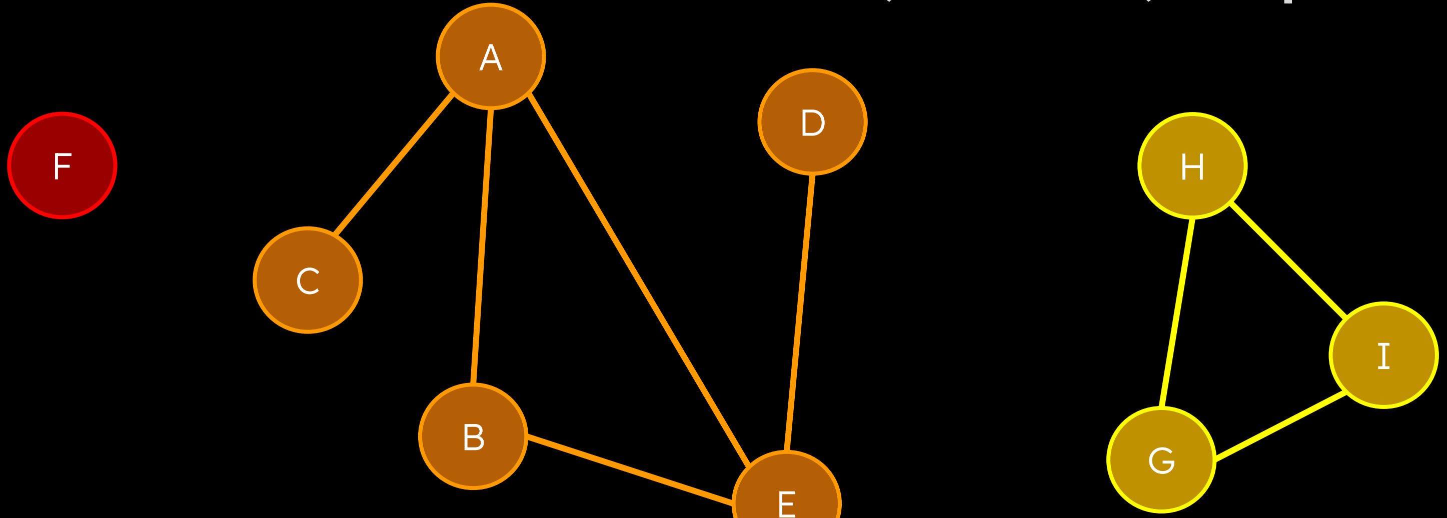
go.osu.edu/cpcattend

Graph Terminology



Graph Terminology

the **(connected) components**

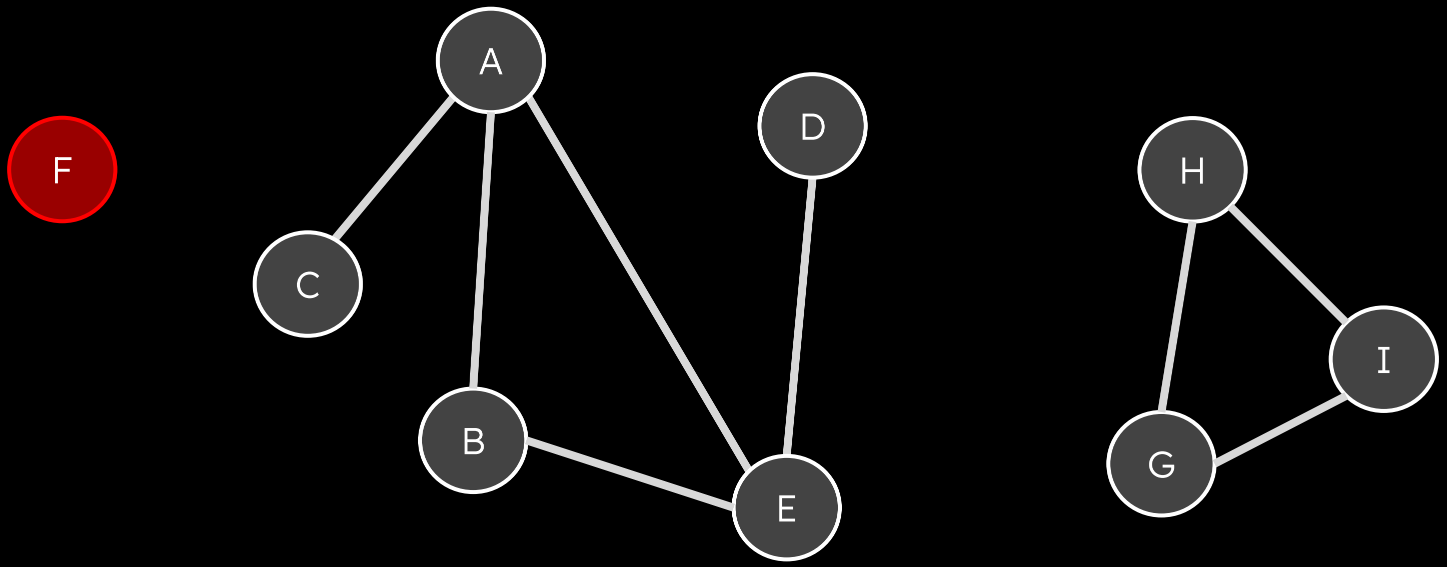


Components

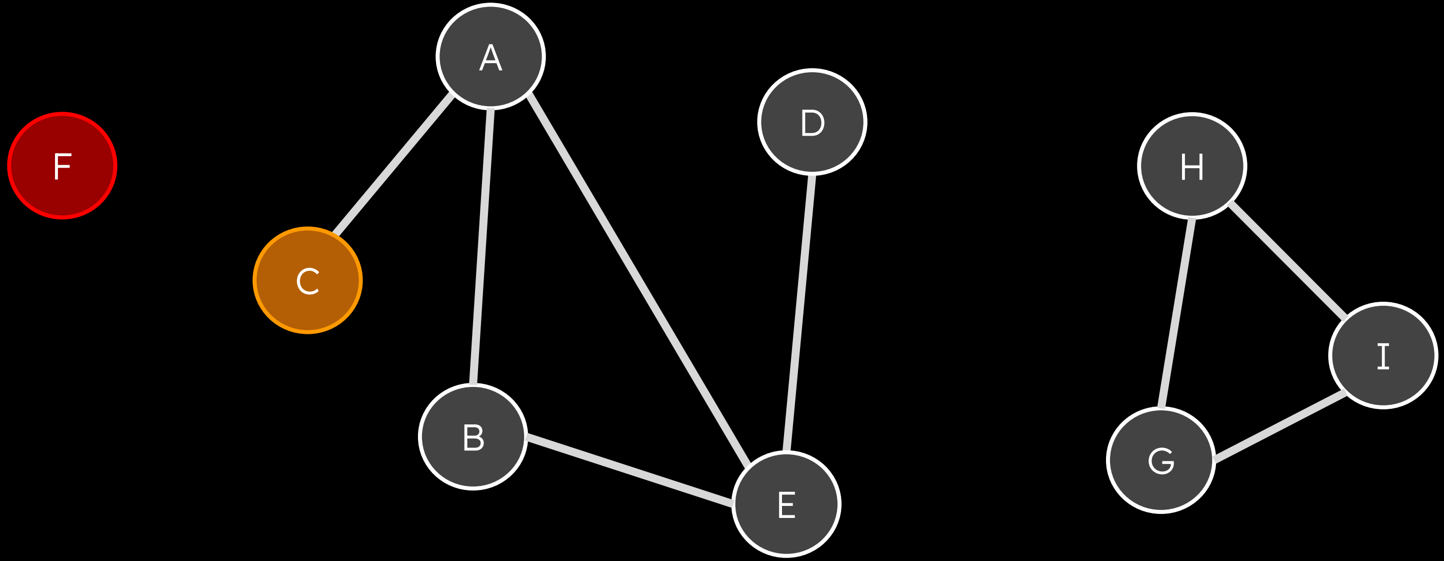
```
function components(G)
    V = {}
    c = 0
    for v in G.V
        if v not in V
            traverse(G, v, V, c)
            c += 1
```

```
function traverse(G, s, V, c)
    F = new queue [s]
    while |F| > 0
        v = F.dequeue()
        if v not in V
            for each vertex u
            adjacent to v
                F.enqueue(u)
                mark v as
                component c
                add v to V
```

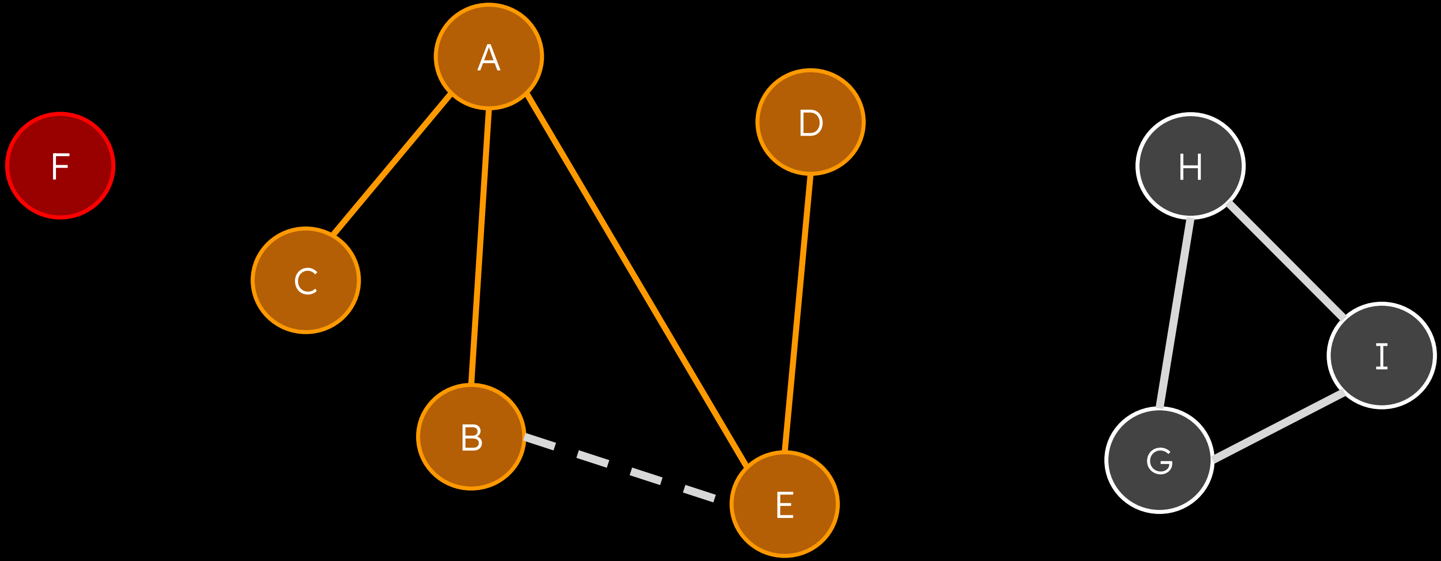
Components



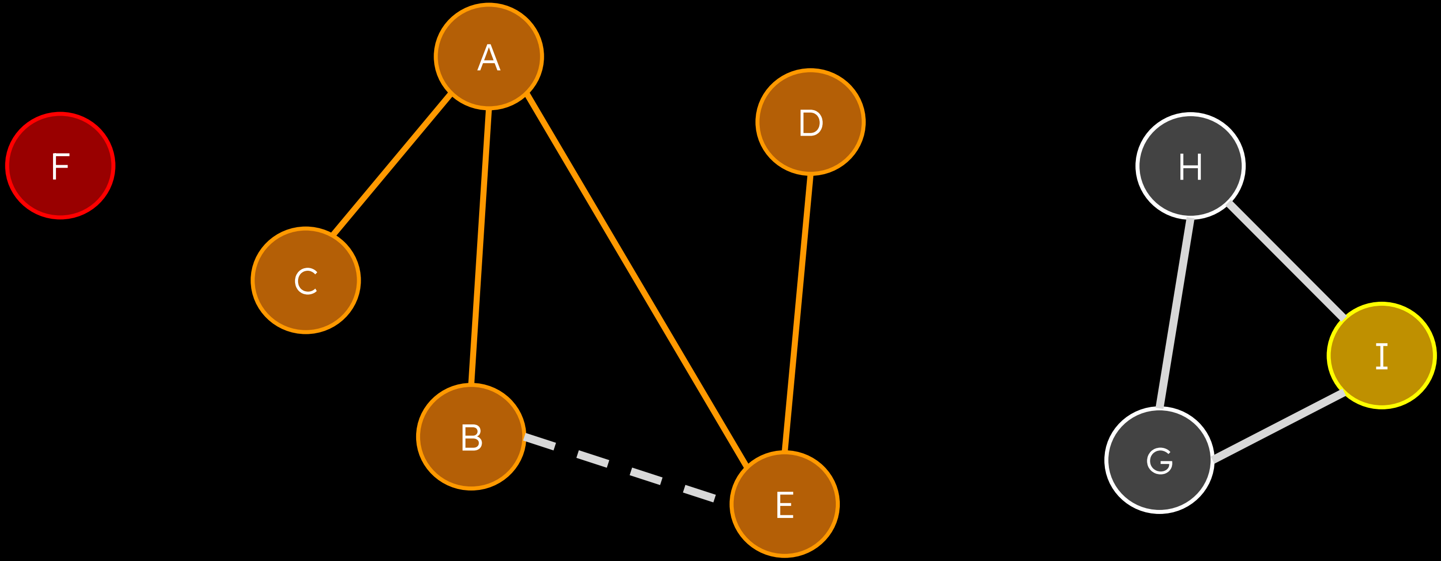
Components



Components



Components



Components