Data Structures

# 24. Graph Traversal/Searching

# Graph Traversal

- Given a graph G = (V, E), directed or undirected
  - Goal is to methodically explore every vertex and every edge
- Traversals of graphs are also called searches
- We can use either breadth-first or depth-first traversals
  - Breadth-first requires a queue
  - Depth-first requires a stack

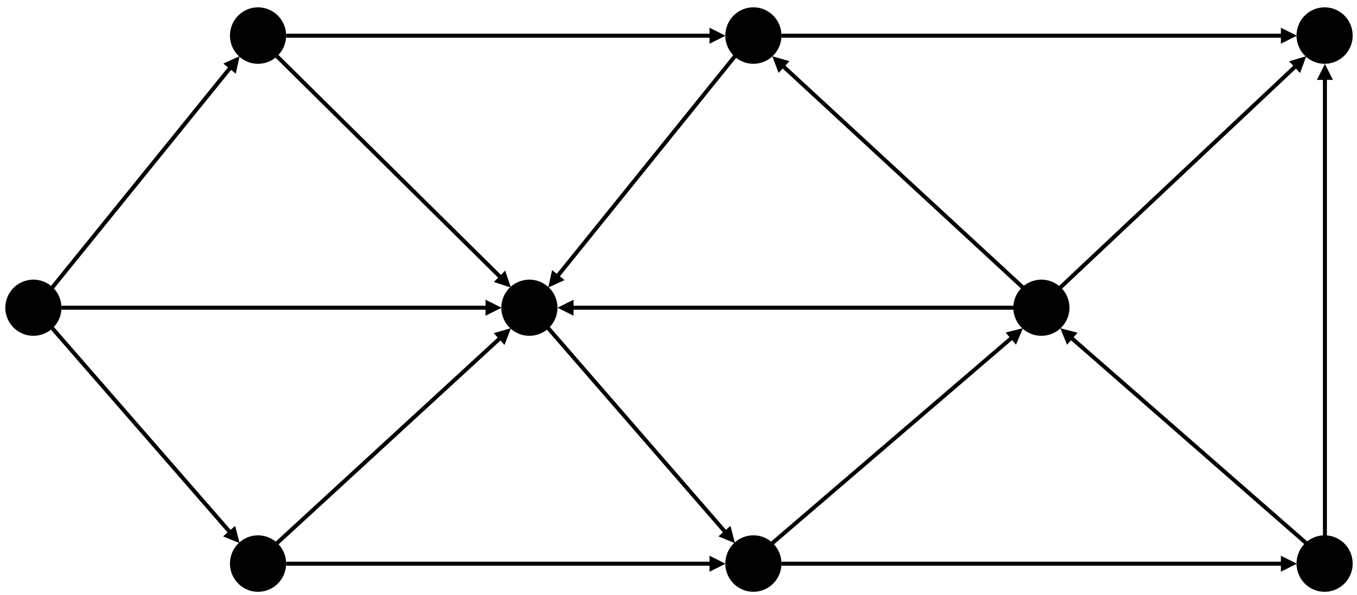
# Breadth-First Search

- Choose any vertex, mark it as visited and enqueue it into queue
- While the queue is not empty
  - Dequeue top vertex v from the queue
  - For each vertex adjacent to v that has not been visited
    - Mark it visited, and
    - Enqueue it into the queue

```
create a queue Q
mark v as visited and put v into Q
while Q is non-empty
   remove the head u of Q (Dequeue)
   mark and enqueue all (unvisited) neighbors of u
```

- The above algorithm continues until the queue is empty!
  - If there are no unvisited vertices, the graph is connected

# Breadth-First Search – Example

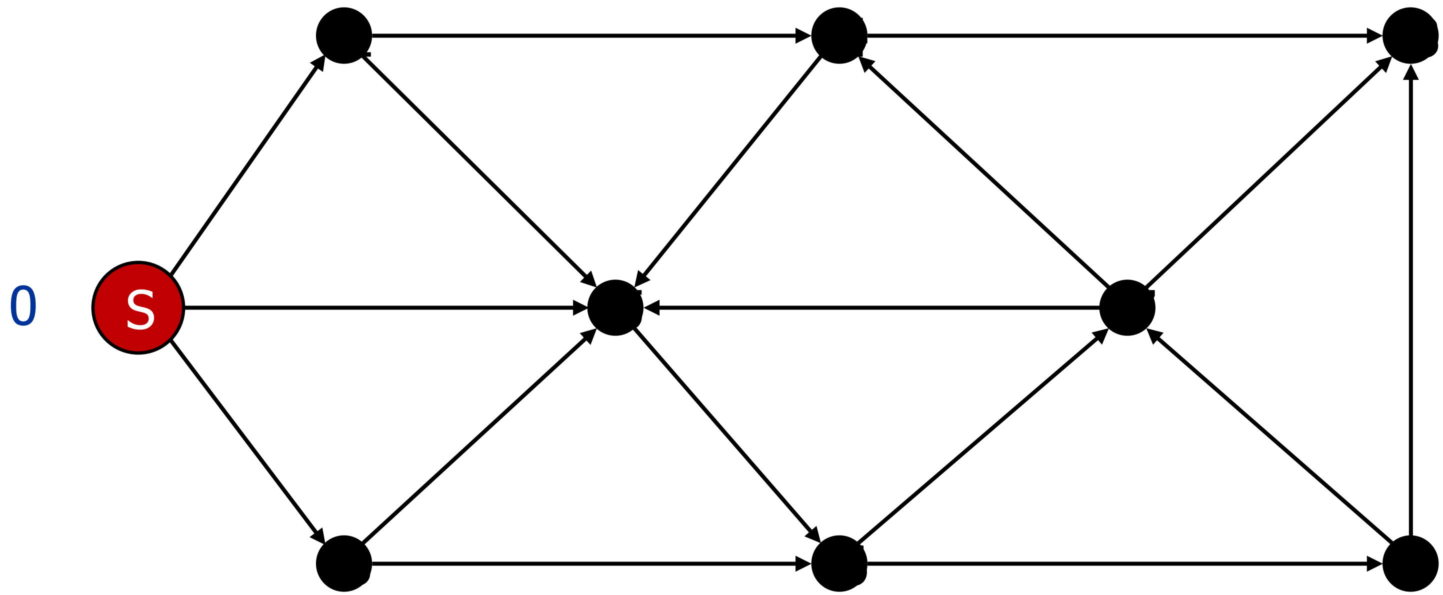


Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

```
1: Create a Queue Q
```

# Breadth-First Search – Example



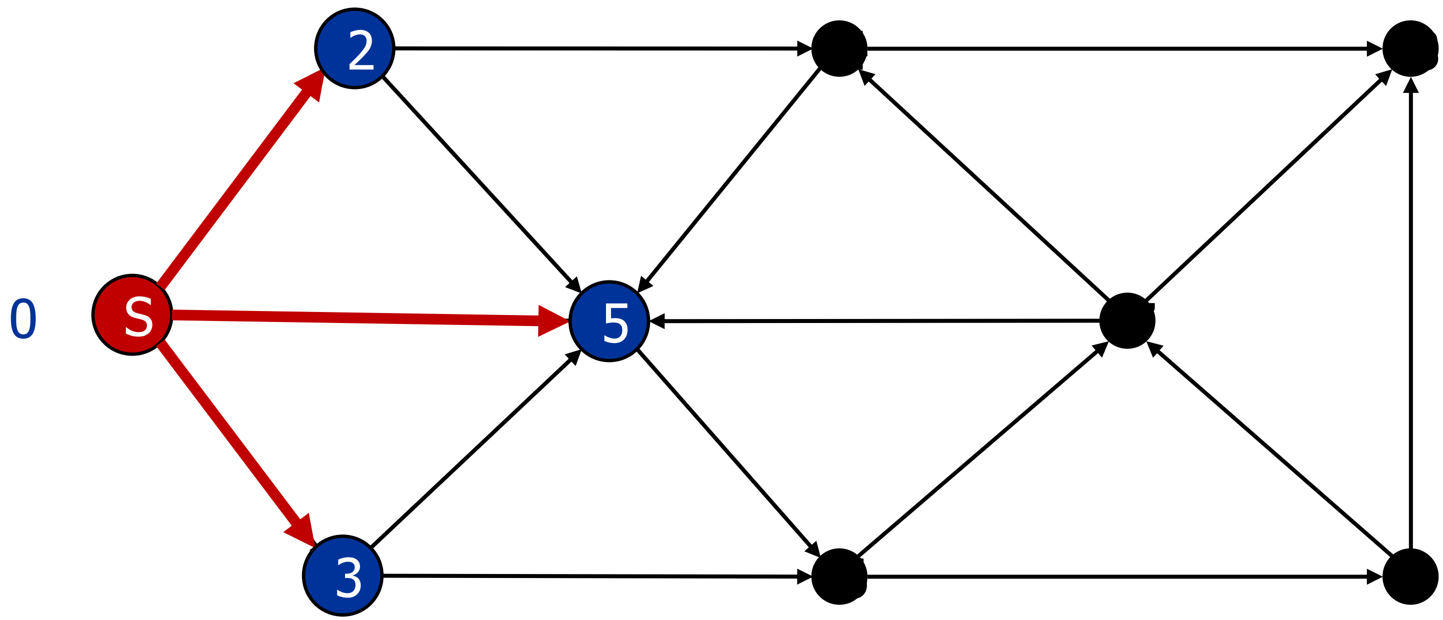

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

S

2: Mark S as visited and put S into Q

# Breadth-First Search – Example



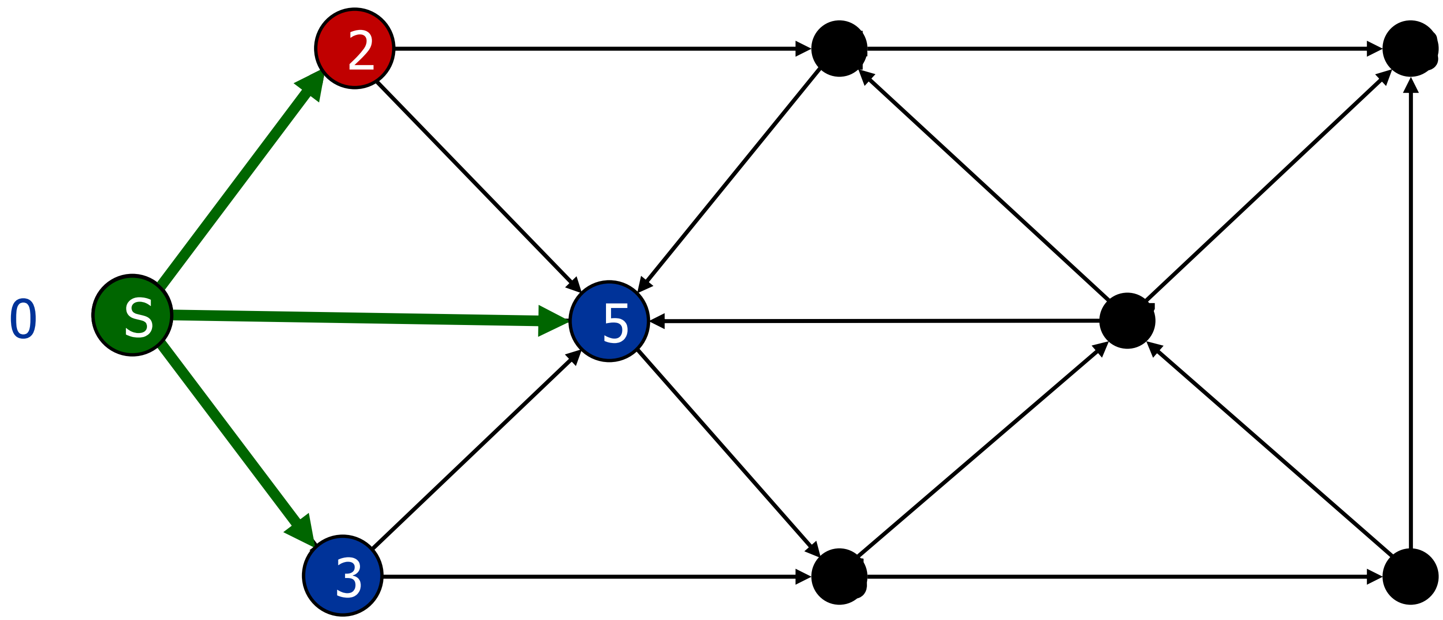


Queue (Q):

| |
|---|
| S |

```
3: While Q not empty
4:     v = dequeue Q  (i.e., S)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



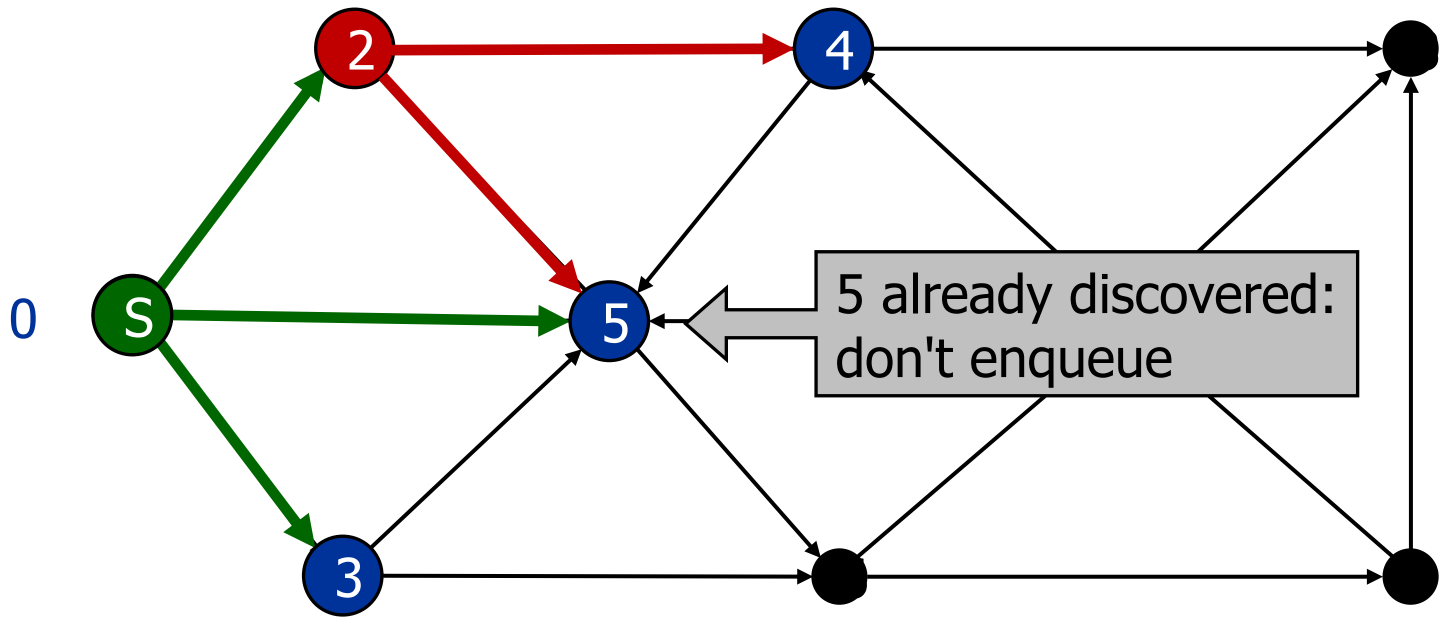


Queue (Q):

| |
|---|
| 2 3 5 |

```
3: While Q not empty
4:     v = dequeue Q (i.e., 2)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



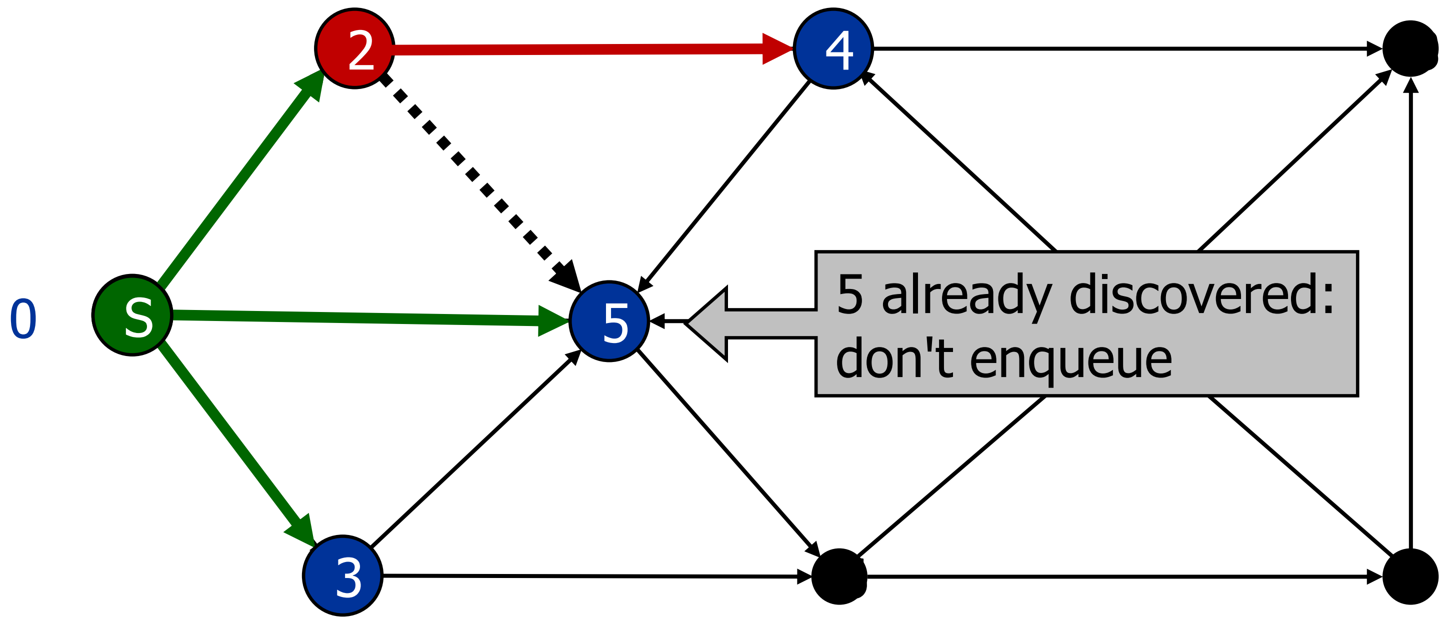


Queue (Q):

2 3 5

```
3: While Q not empty
4:    v = dequeue Q (i.e., 2)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



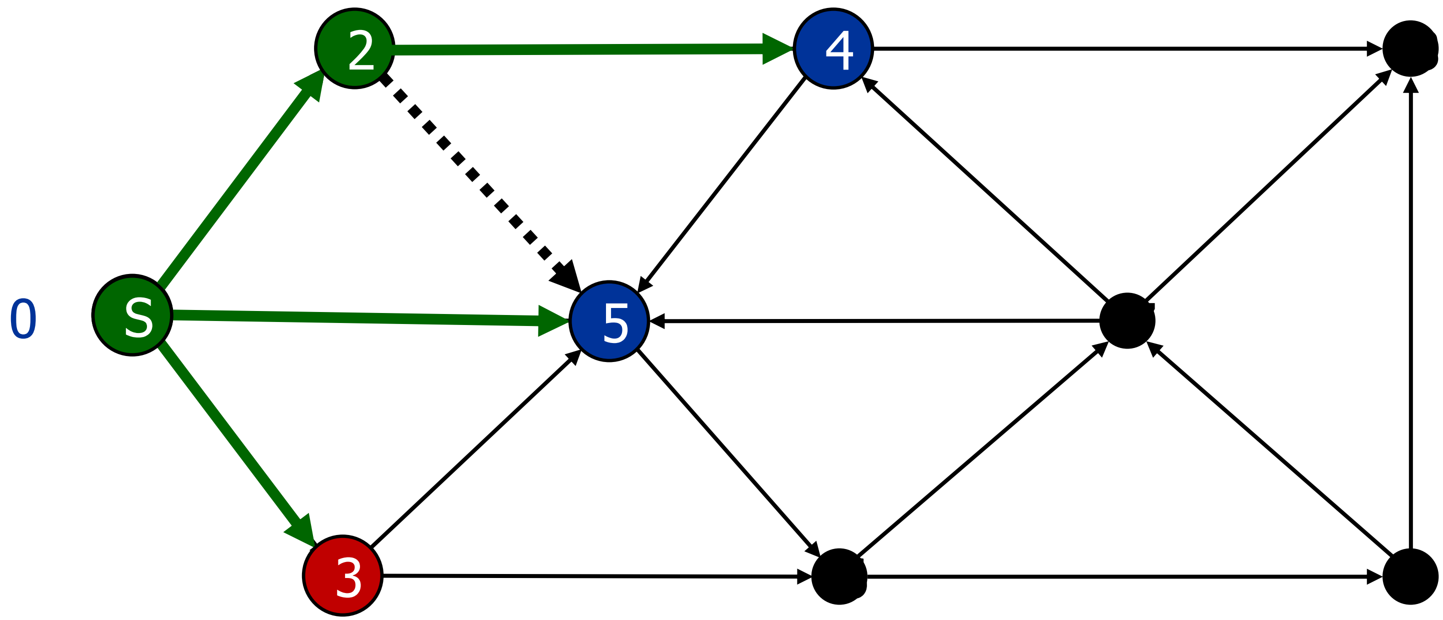

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

2 3 5

```
3: While Q not empty
4:    v = dequeue Q (i.e., 2)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



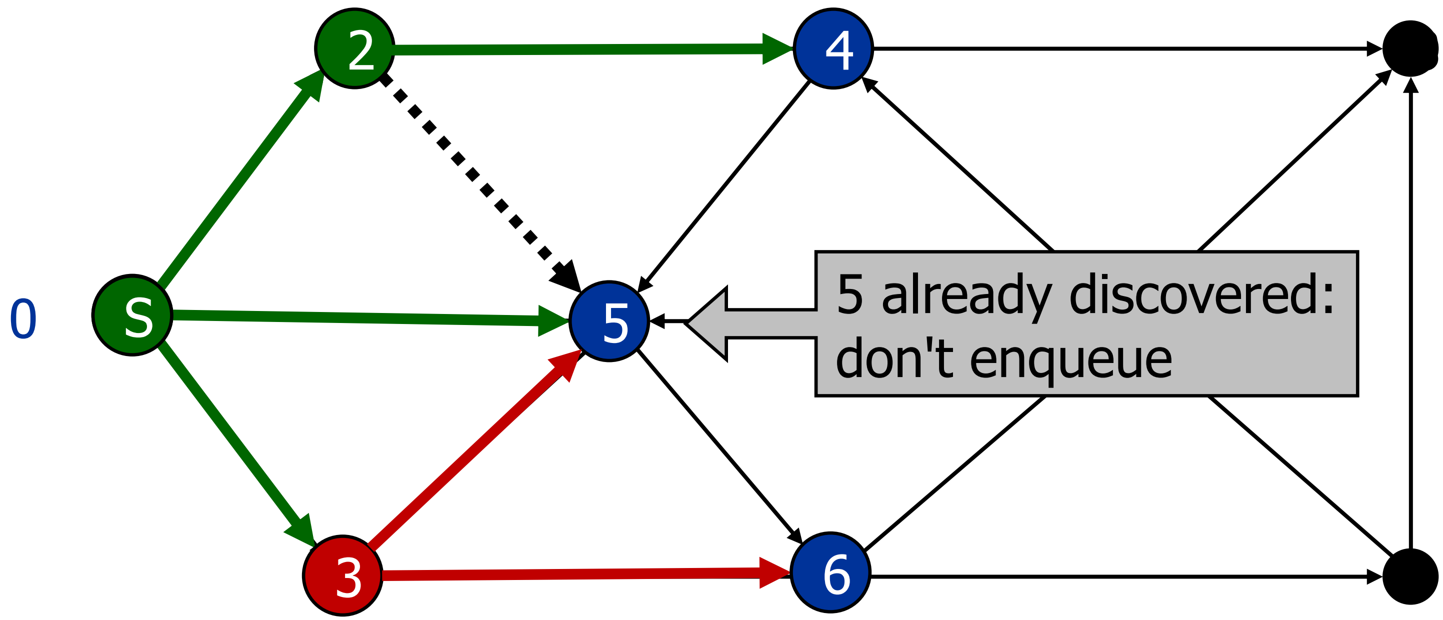


Queue (Q):

3 5 4

```
3: While Q not empty
4:    v = dequeue Q (i.e., 3)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



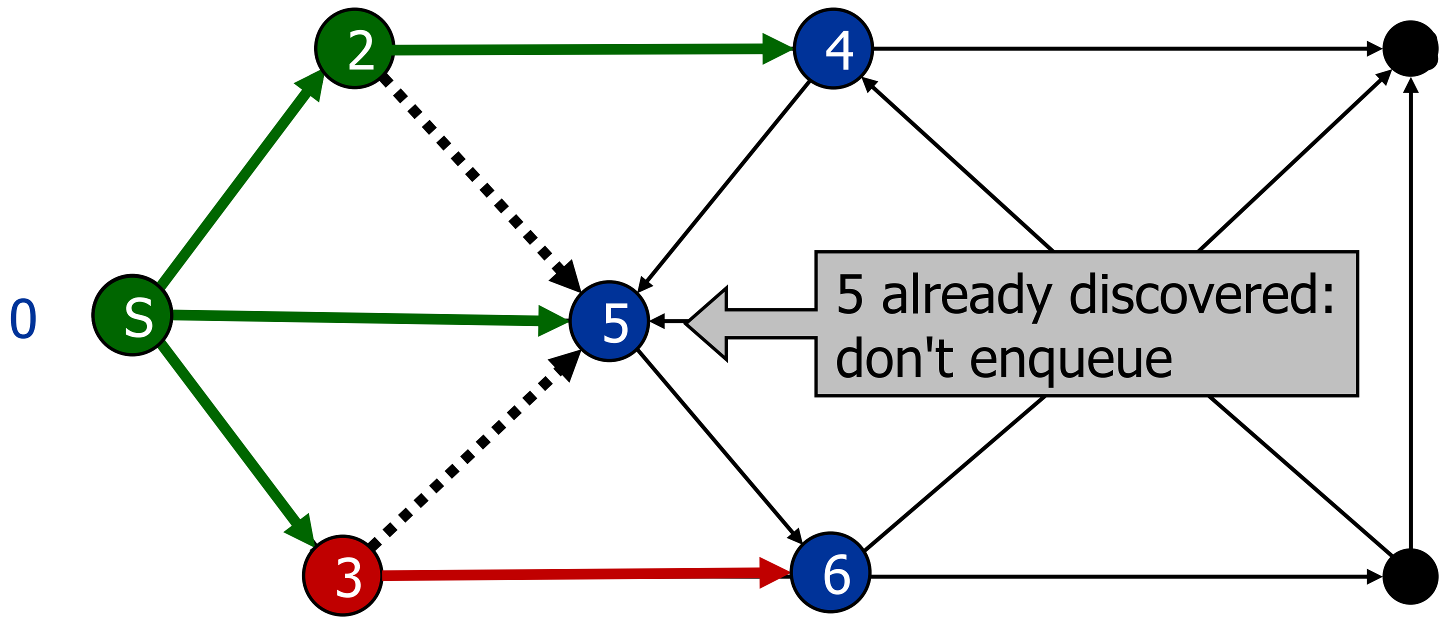

| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue (Q):

3 5 4

```
3: While Q not empty
4:    v = dequeue Q (i.e., 3)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example




Undiscovered

Discovered

Top of queue

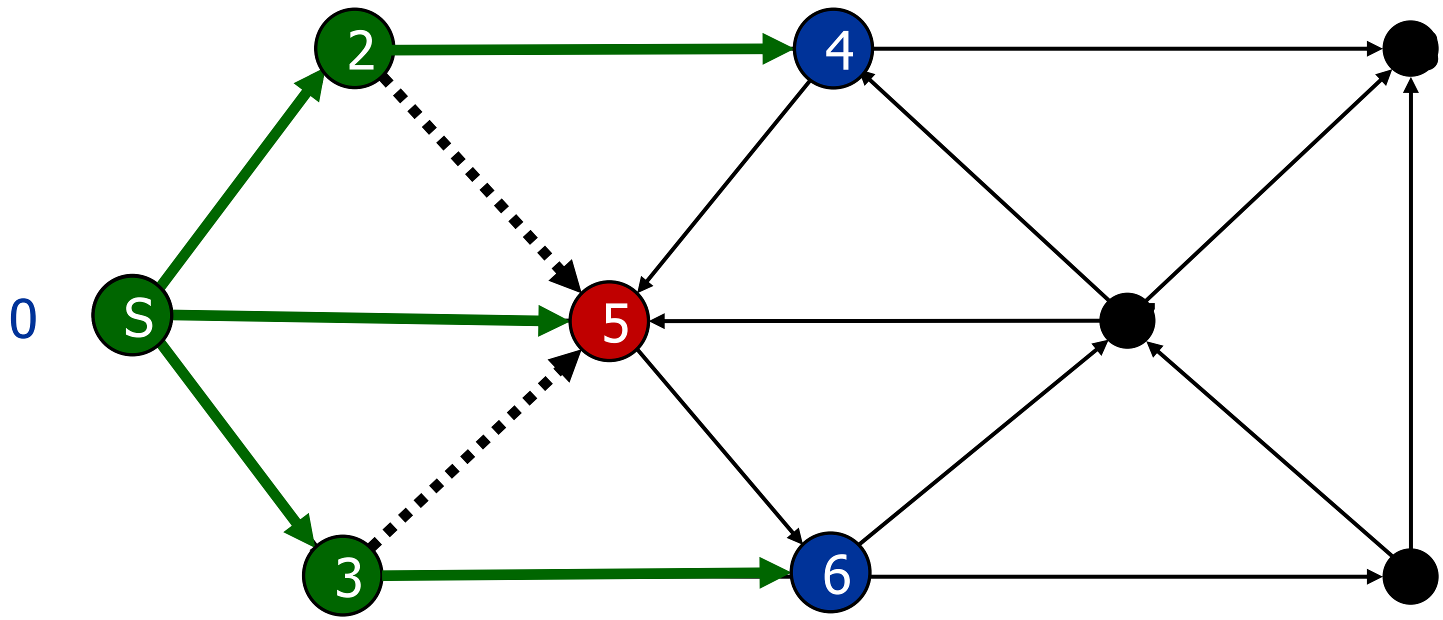
Finished

Queue (Q):

3 5 4

```
3: While Q not empty
4:    v = dequeue Q (i.e., 3)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



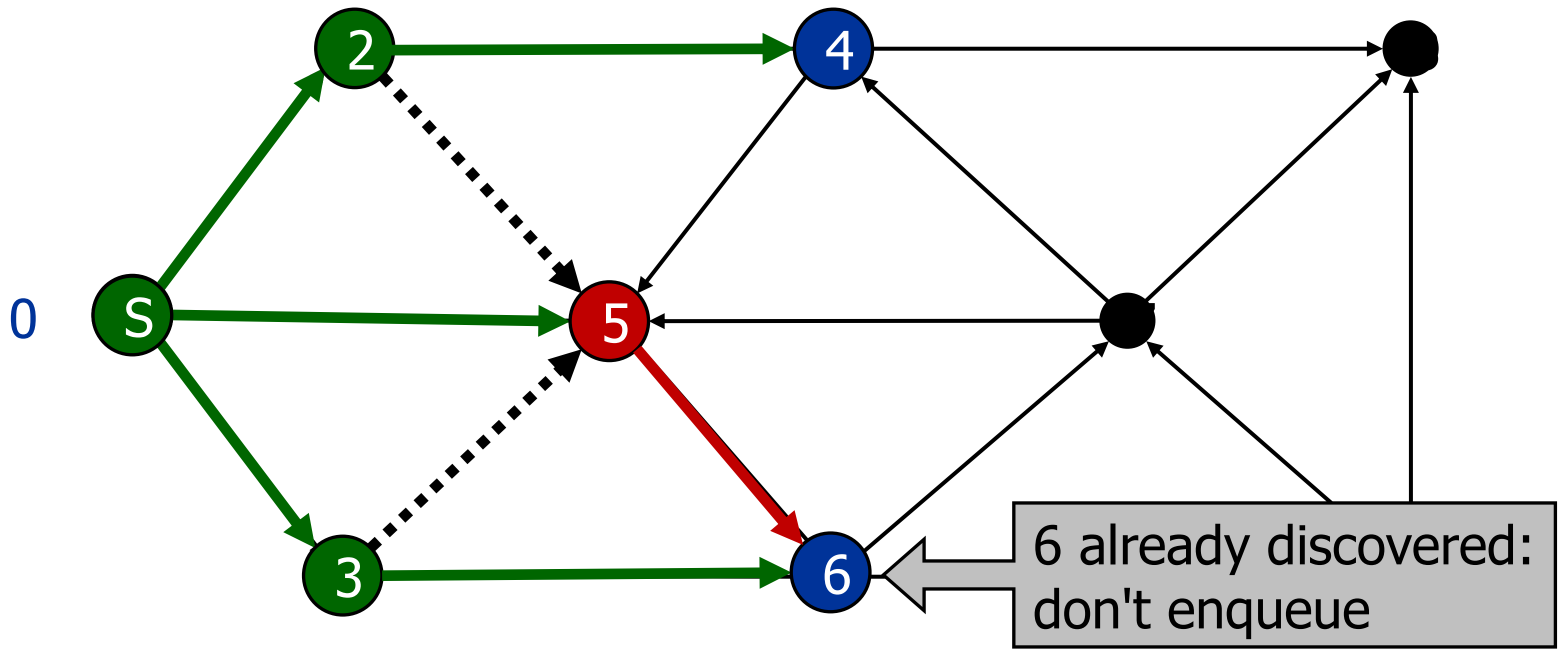


Queue (Q):

5 4 6

```
3: While Q not empty
4:     v = dequeue Q (i.e., 5)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



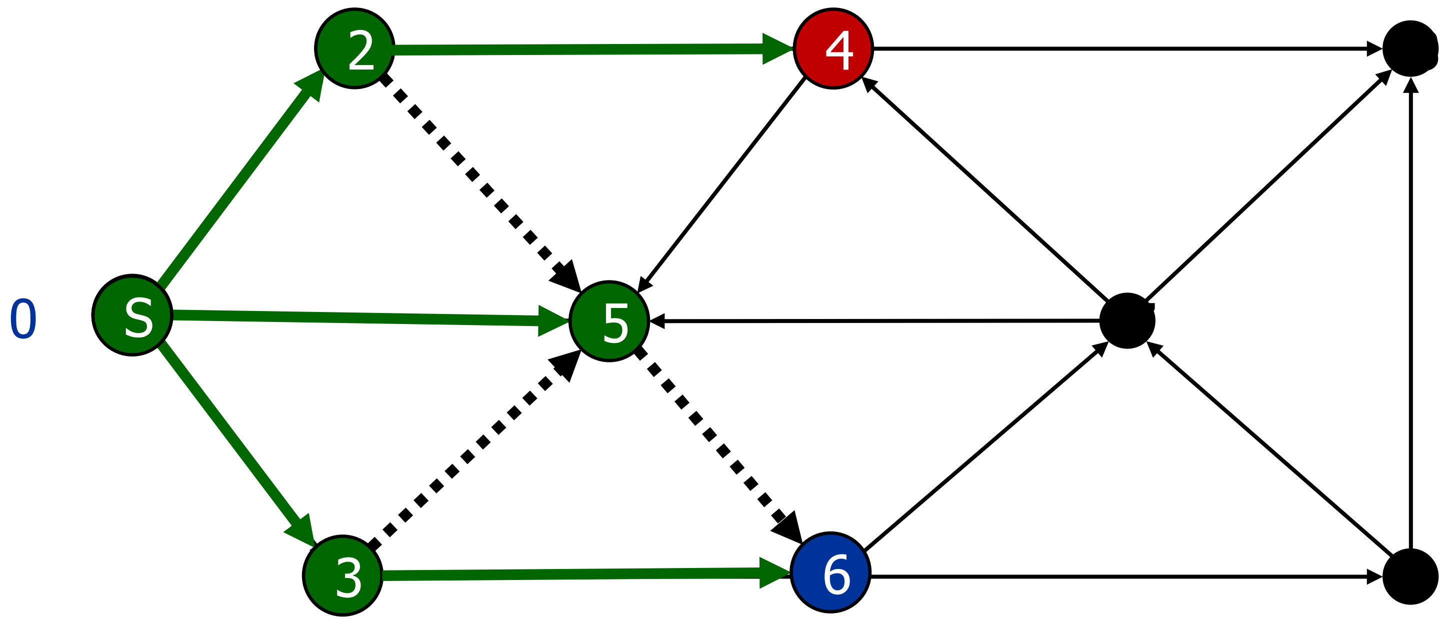

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

5 4 6

```
3: While Q not empty
4:     v = dequeue Q (i.e., 5)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



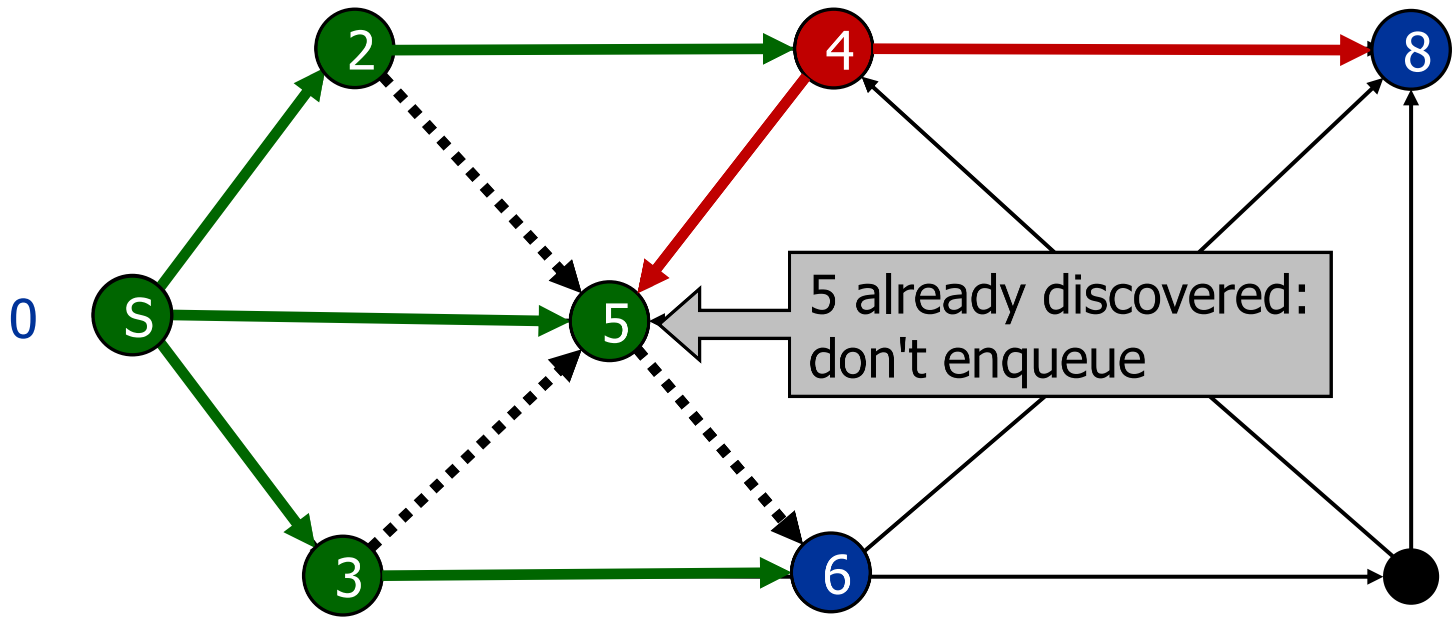


Queue (Q):

4 6

```
3: While Q not empty
4:     v = dequeue Q (i.e., 4)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



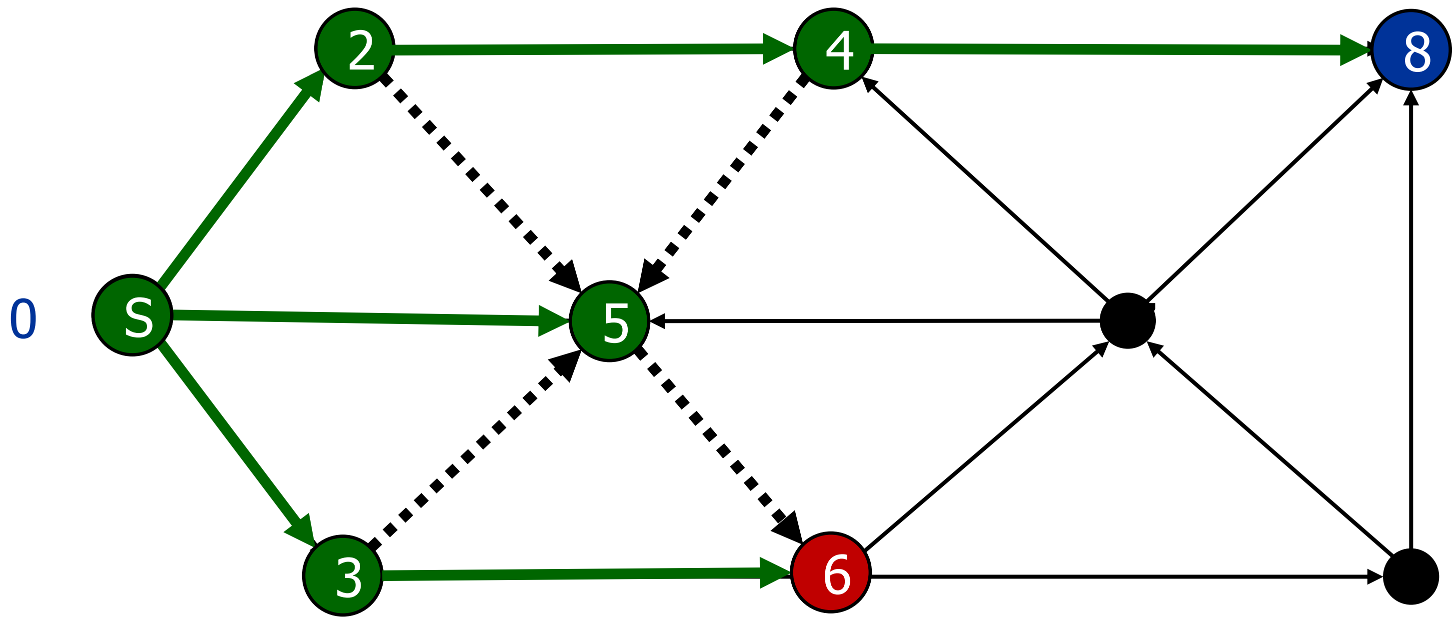

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

4 6

```
3: While Q not empty
4:    v = dequeue Q (i.e., 4)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



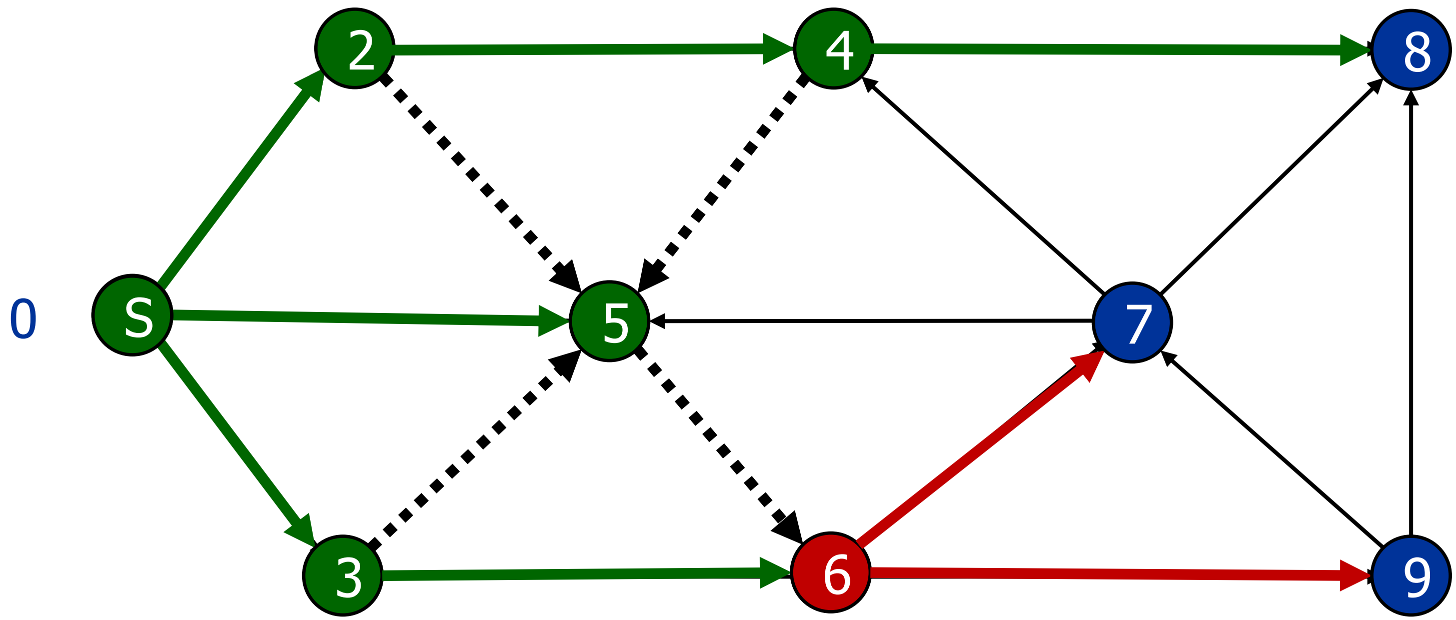

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

6 8

```
3: While Q not empty
4:     v = dequeue Q (i.e., 6)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



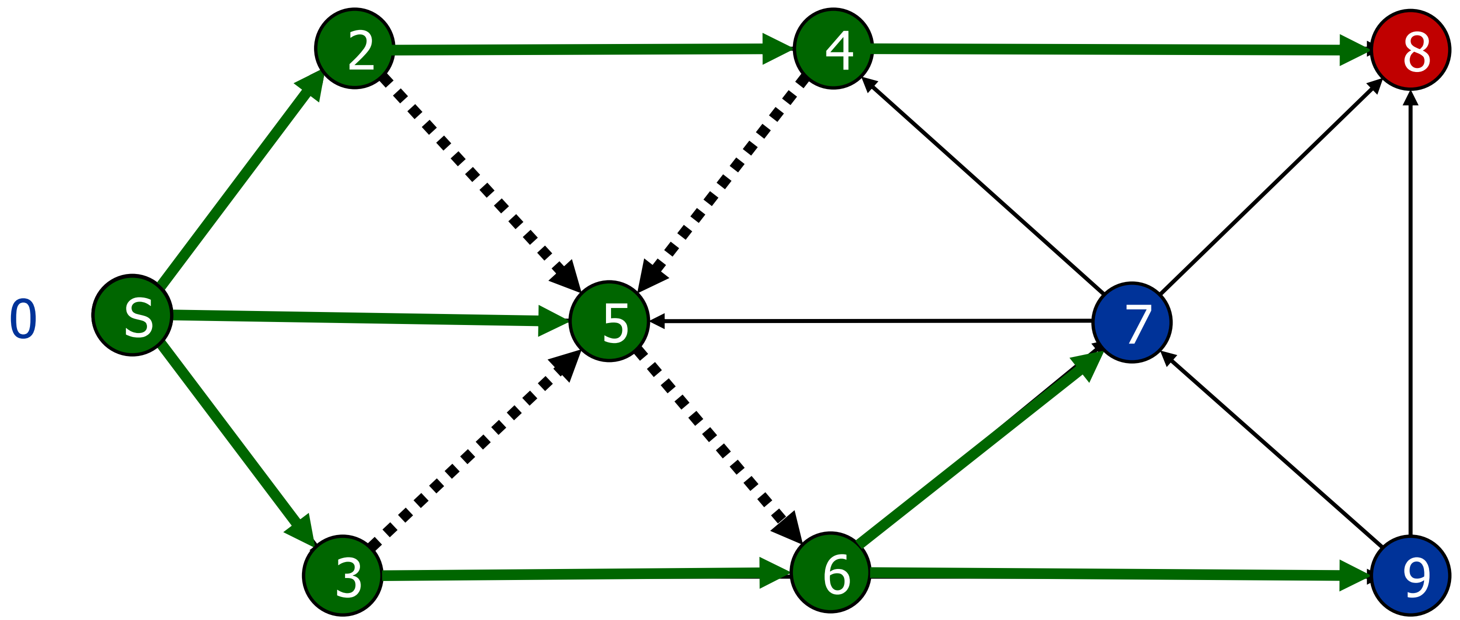

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

6 8

```
3: While Q not empty
4:     v = dequeue Q (i.e., 6)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



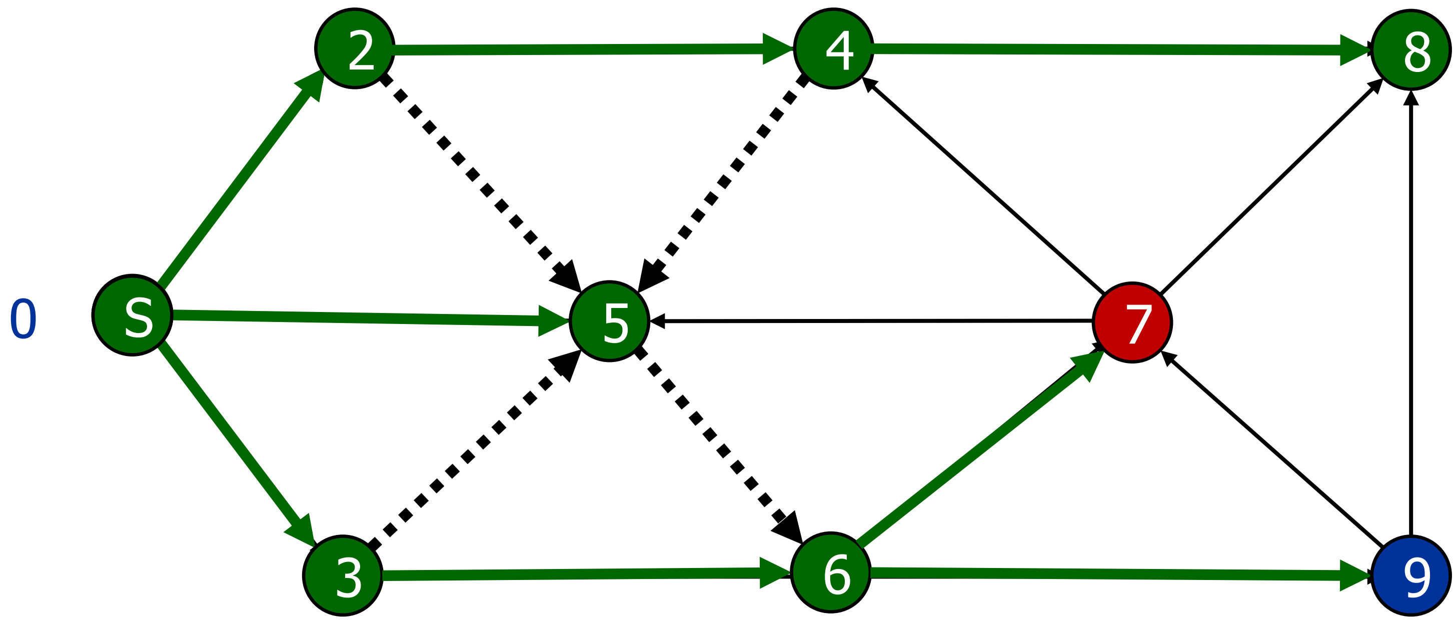

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

8 7 9

```
3: While Q not empty
4:     v = dequeue Q (i.e., 8)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



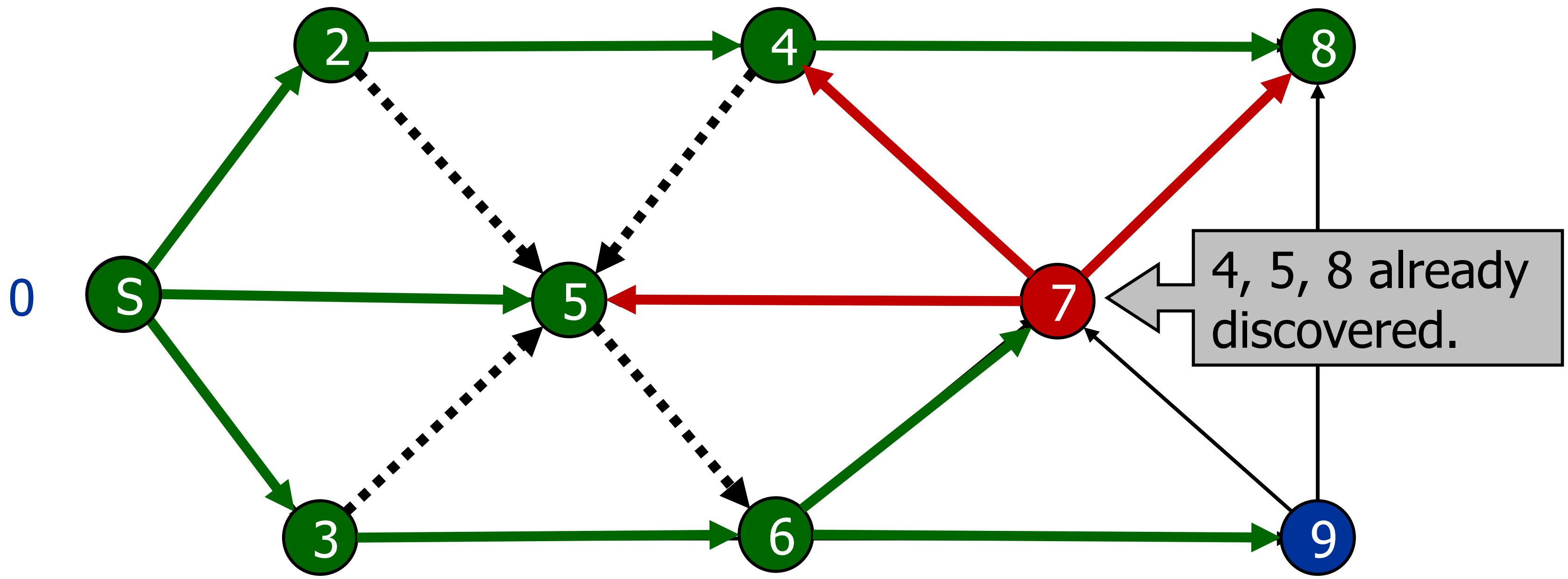


Queue (Q):

7 9

```
3: While Q not empty
4:    v = dequeue Q (i.e., 7)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



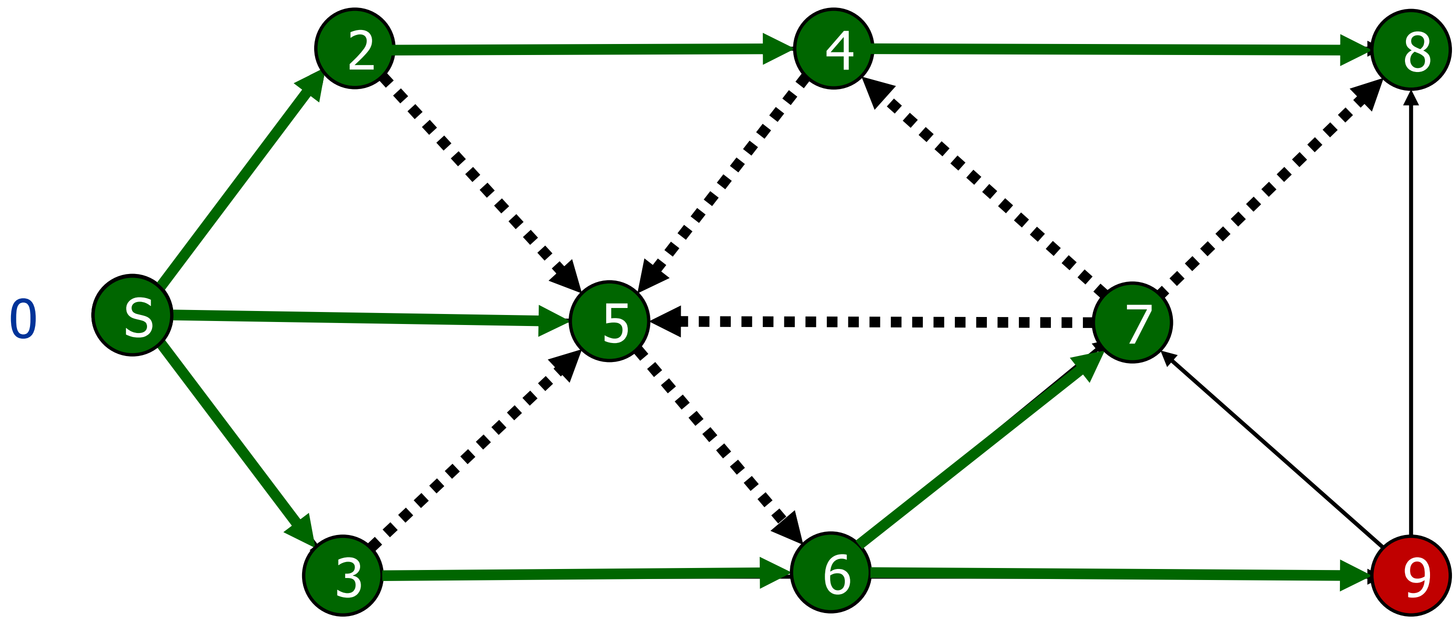

Undiscovered
Discovered
Top of queue
Finished

Queue (Q):

7 9

```
3: While Q not empty
4:     v = dequeue Q (i.e., 7)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



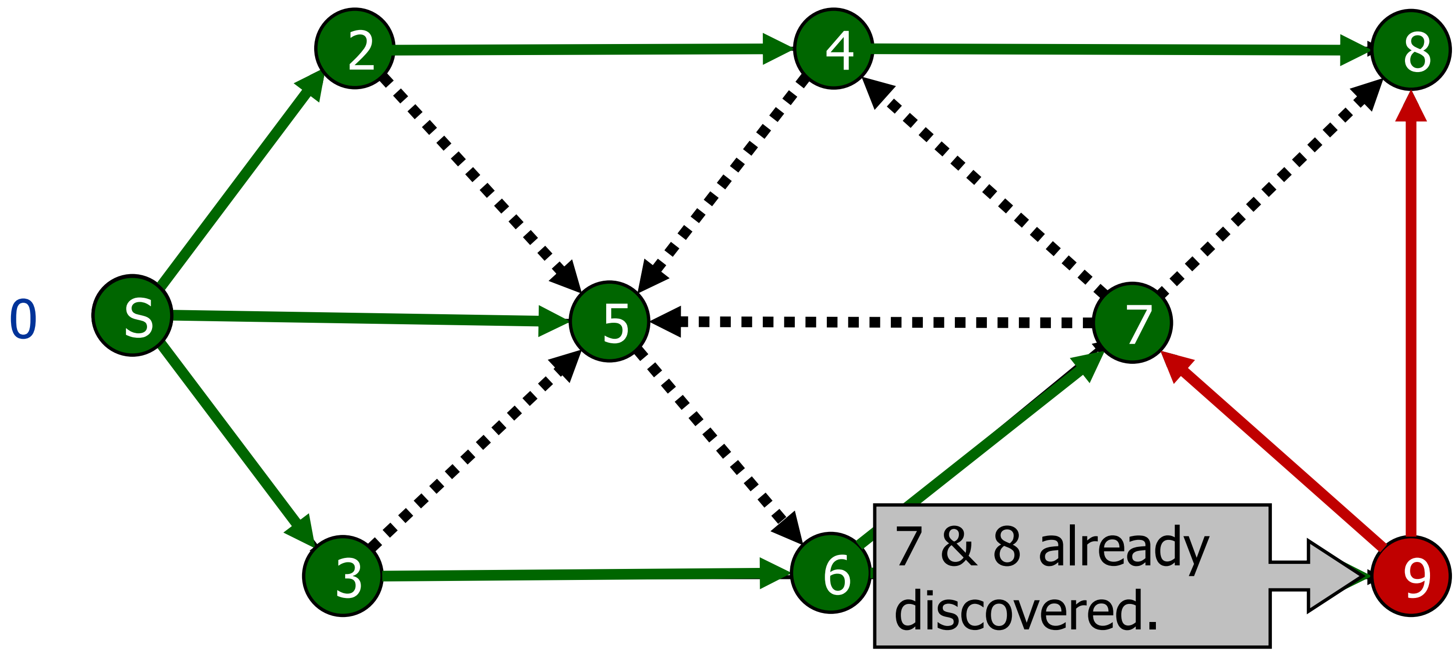


Queue (Q):

9

```
3: While Q not empty
4:     v = dequeue Q (i.e., 9)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



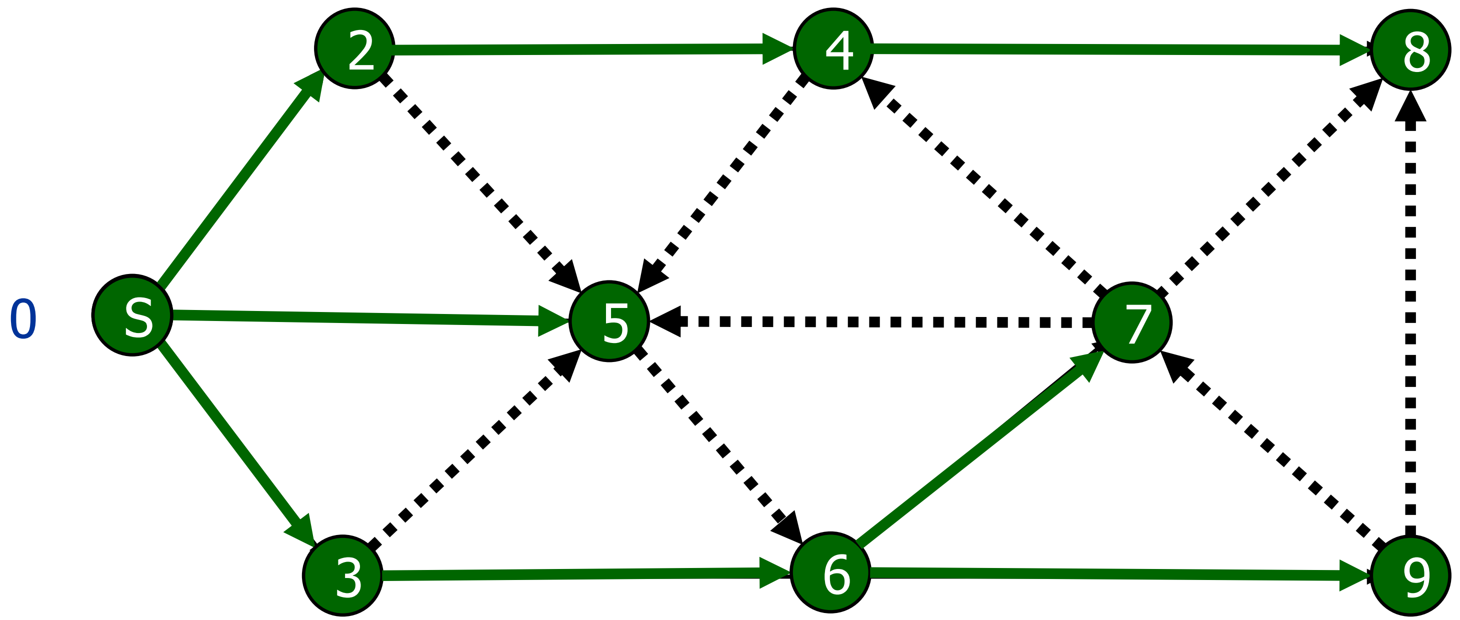


Queue (Q):

9

```
3: While Q not empty
4:    v = dequeue Q (i.e., 9)
5:    mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example

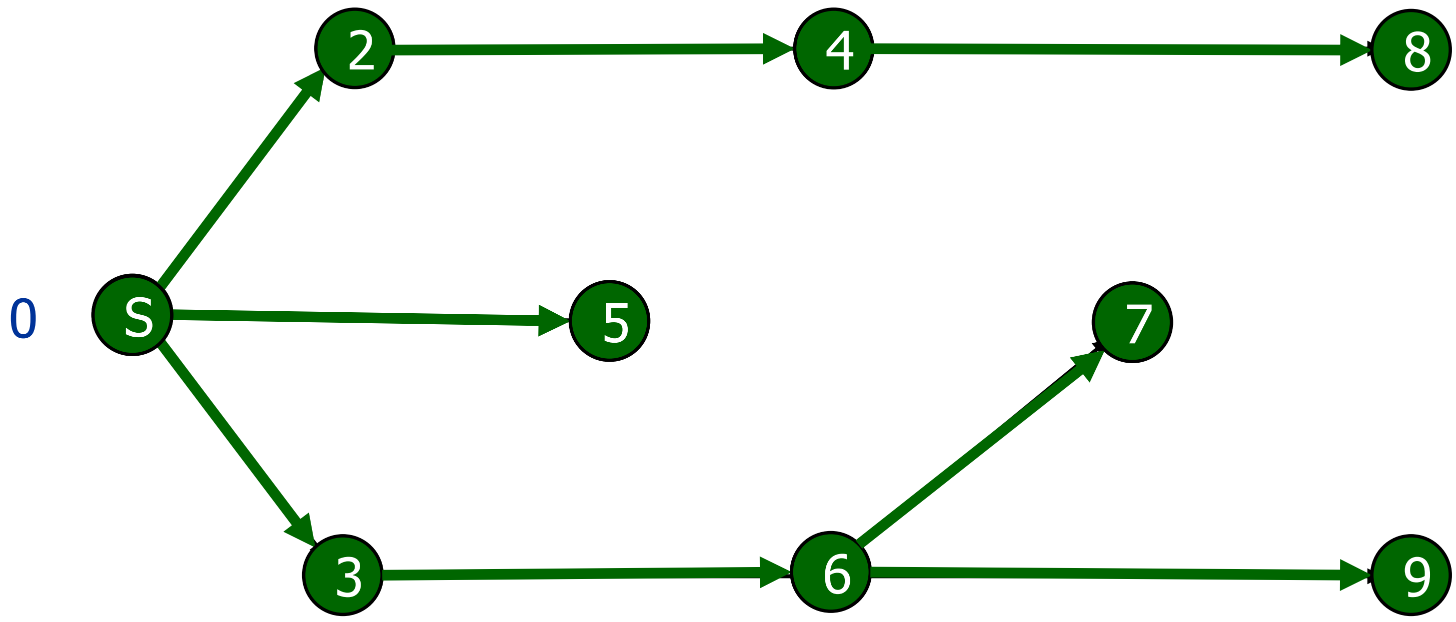




Queue (Q):

```
3: While Q not empty
4:     v = dequeue Q (i.e., NULL)
5:     mark & enqueue all (unvisited) neighbors of v
```

# Breadth-First Search – Example



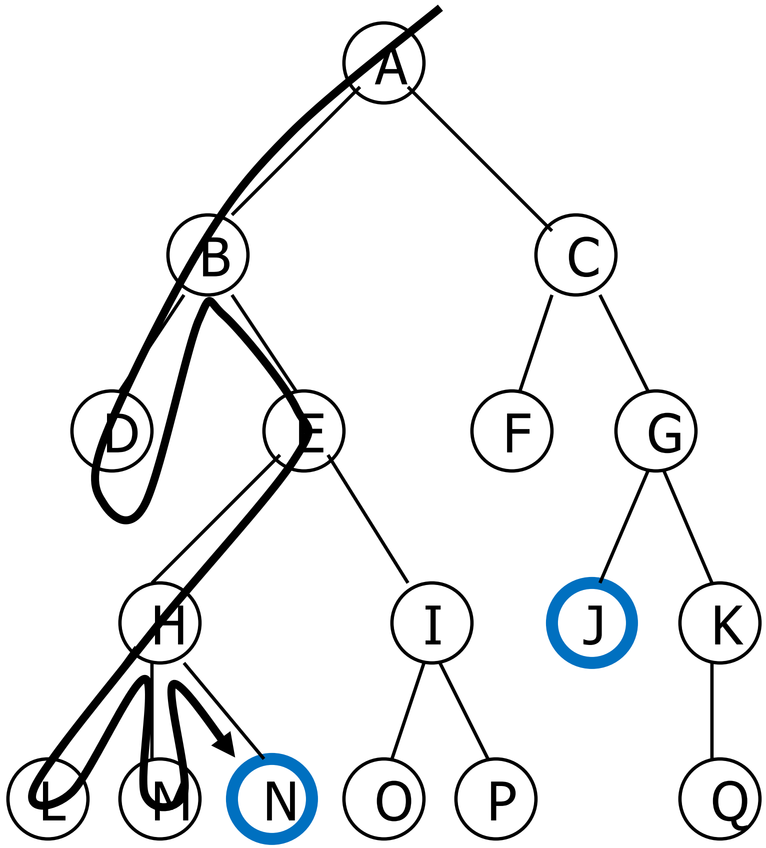

Breadth-First Search (BFS) tree rooted at S containing all nodes of the graph

# Breadth-First Search – Properties

- Given a graph `G=(V,E)` and source vertex `S`, the following holds for the BFS algorithm
  - Systematically explores the edges of `G` to “discover” every vertex reachable from `S`
  - Creates a BFS tree rooted at `S` that contains all such vertices
  - Discovers all vertices at distance `k` from `S` before discovering any vertices at distance `k+1`

# Depth-First Search – Trees

- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path
- For example, after searching A, then B, then D, the search backtracks and tries another path from B
- N will be found before J
- Node are explored in the order A B D E H L M N I O P C F G J K Q

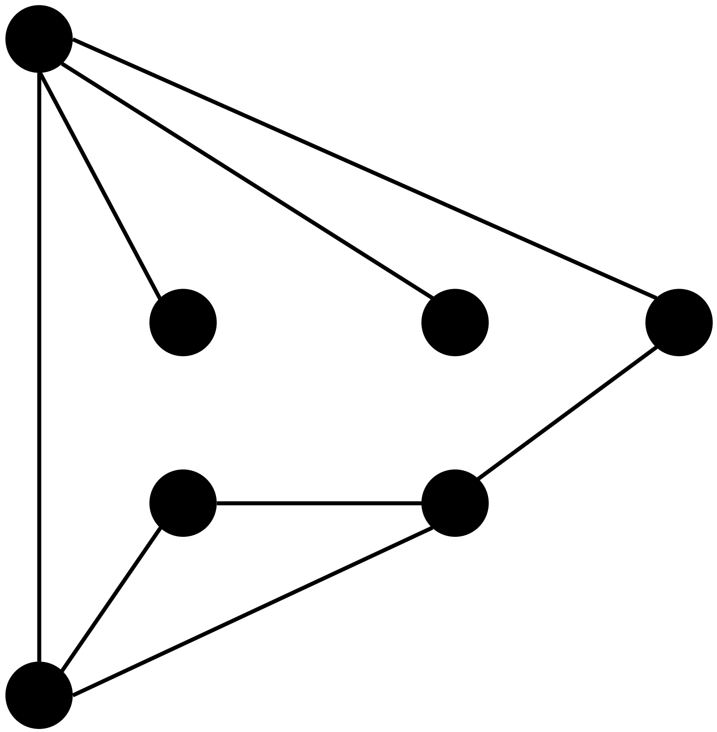


# Depth-First Search

- Choose any vertex, mark it as visited
- From that vertex:
  - If there is another adjacent vertex not yet visited, go to it
  - Otherwise, go back to the most previous vertex that has not yet had all of its adjacent vertices visited and continue from there
- Continue until no visited vertices have unvisited adjacent vertices

```
Create a stack S
Mark v as visited and push v onto S
while S is non-empty
    peek at the top u of S
    if u has an (unvisited) neighbor w
        mark w and push it onto S
    else
        pop S
```

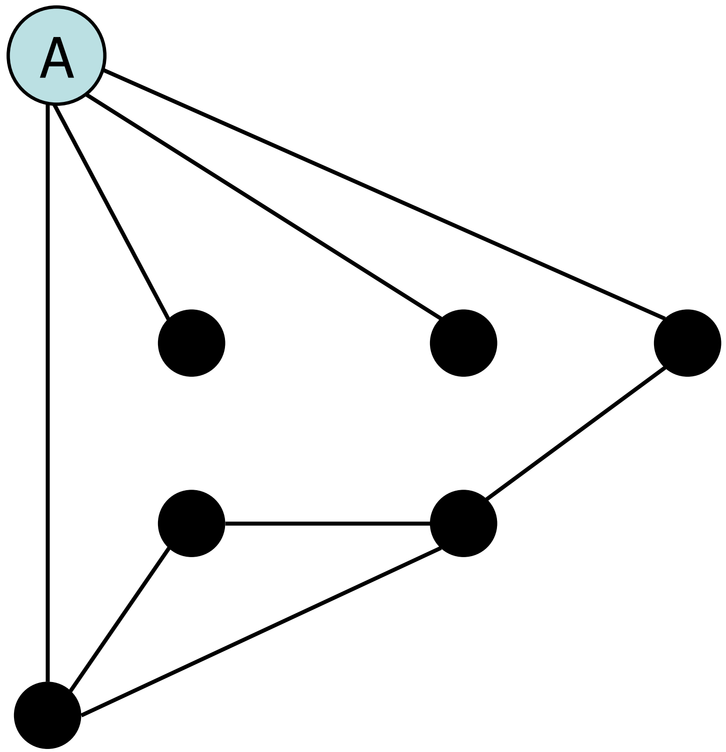
# Depth-First Search – Example



**Adjacency List**

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

# Depth-First Search – Example

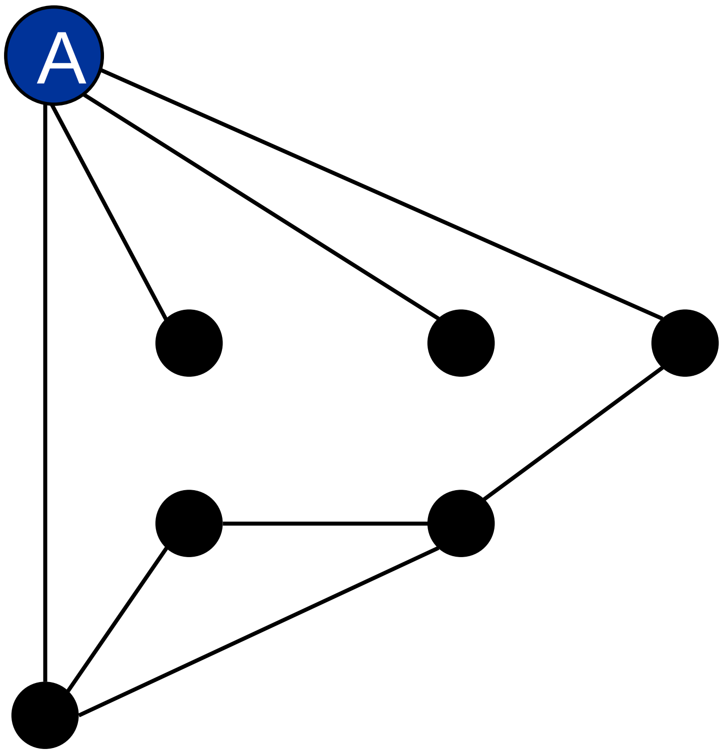



<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

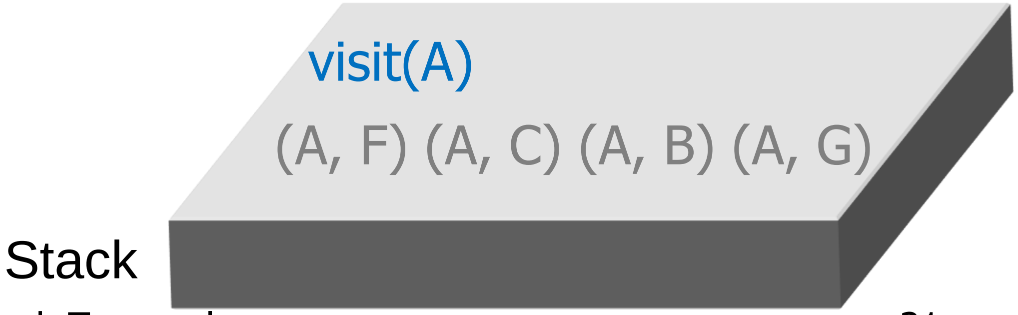
| |
|---|
| Undiscovered |
| Marked |
| Active |
| Finished |

# Depth-First Search – Example

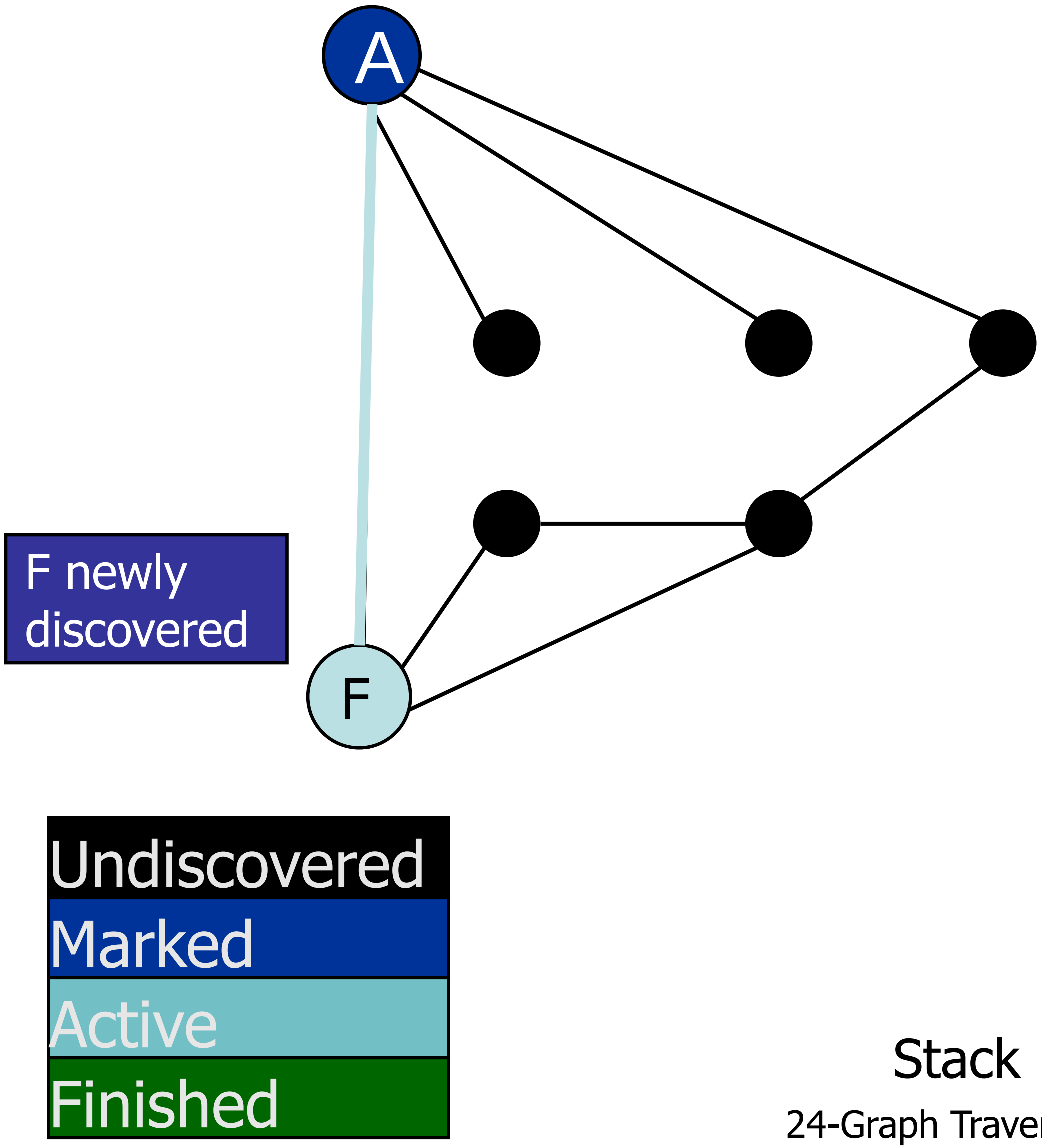



Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

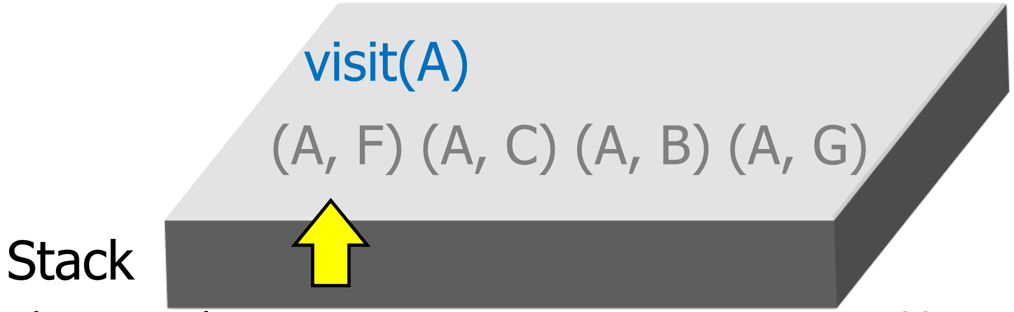



# Depth-First Search – Example




<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

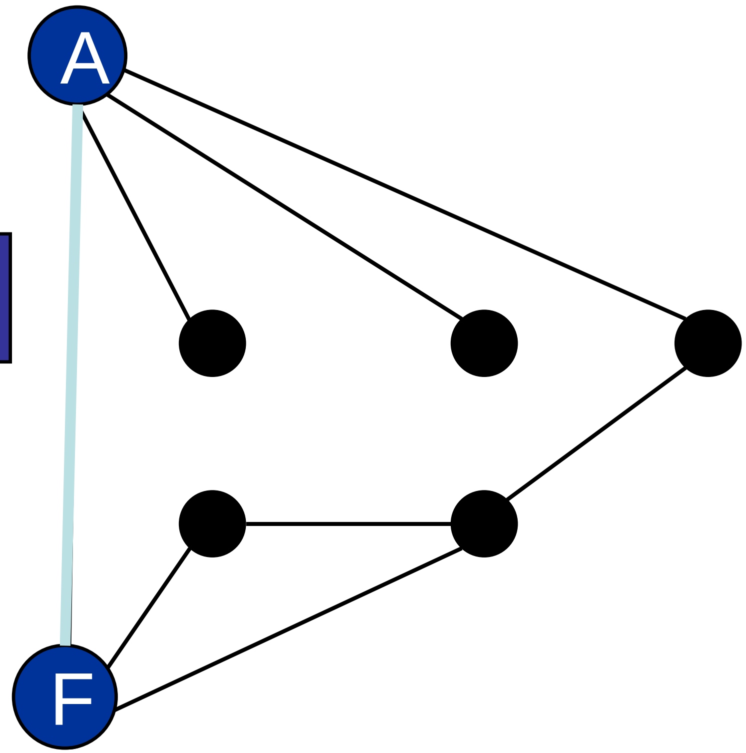


# Depth-First Search – Example

**Adjacency List**

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

A already marked



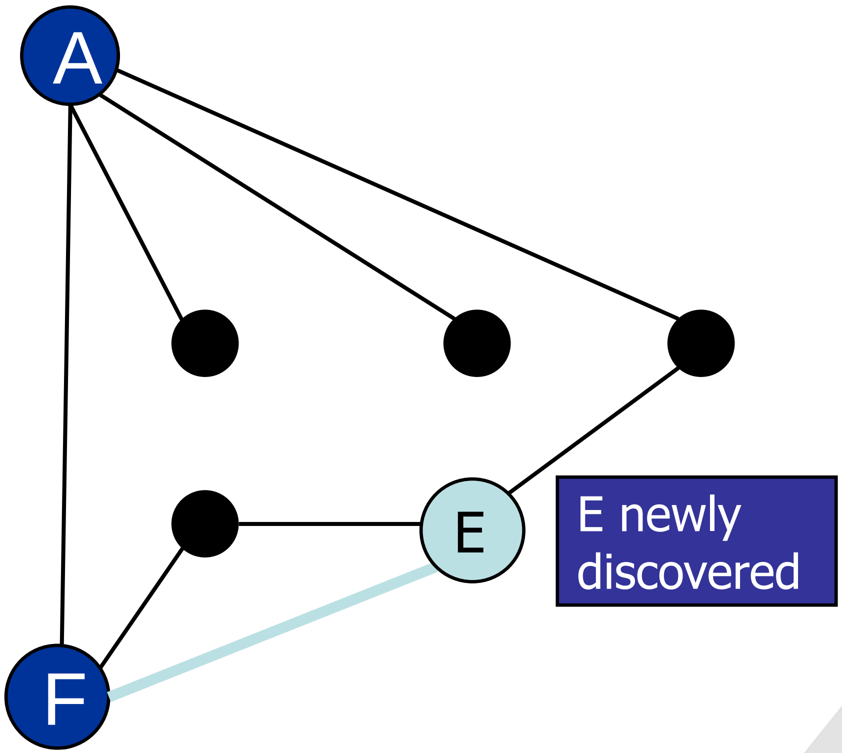

Undiscovered
Marked
Active
Finished


Stack

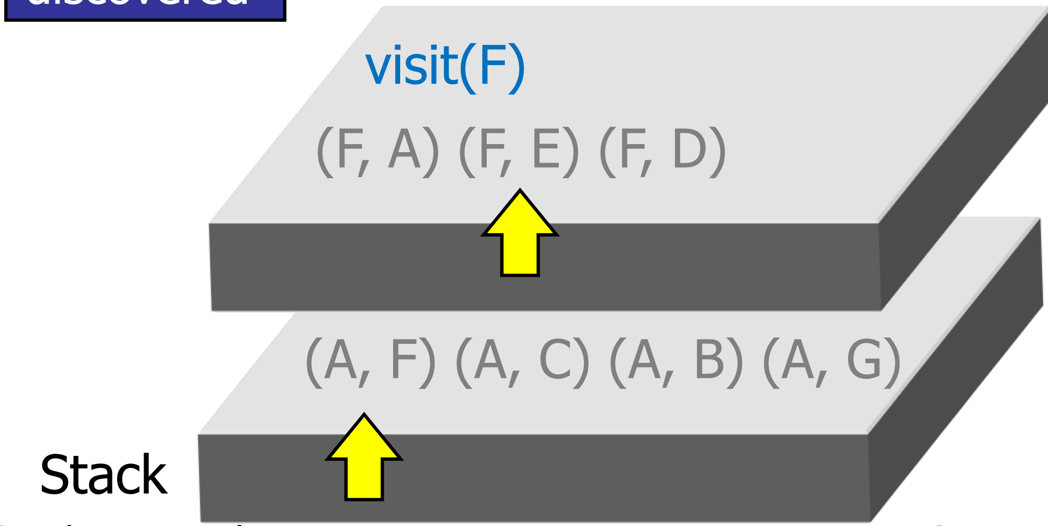
# Depth-First Search – Example

**Adjacency List**

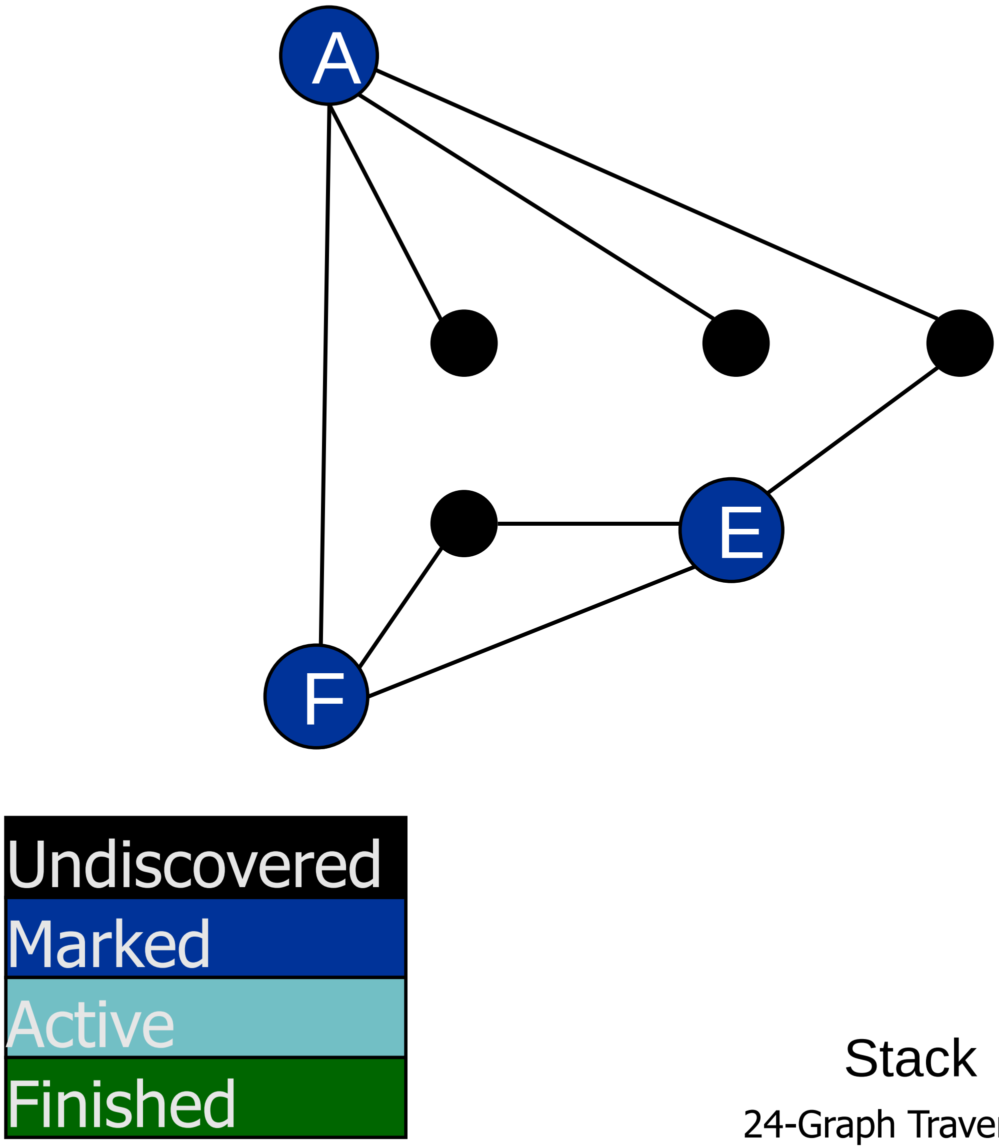
A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




Undiscovered
Marked
Active
Finished

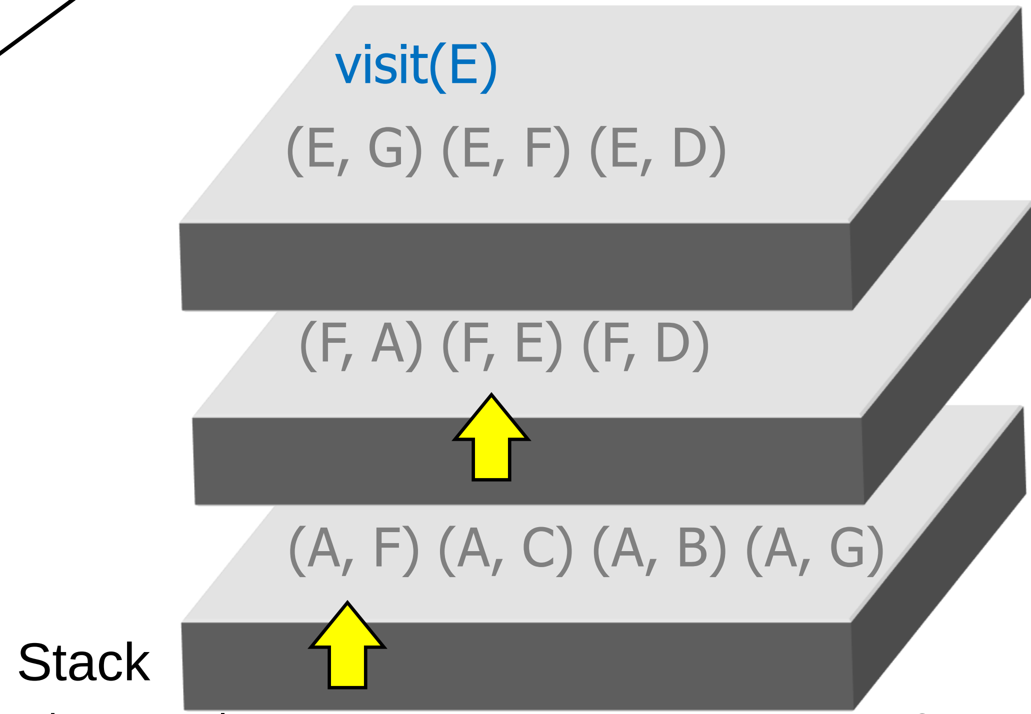


# Depth-First Search – Example




**Adjacency List**

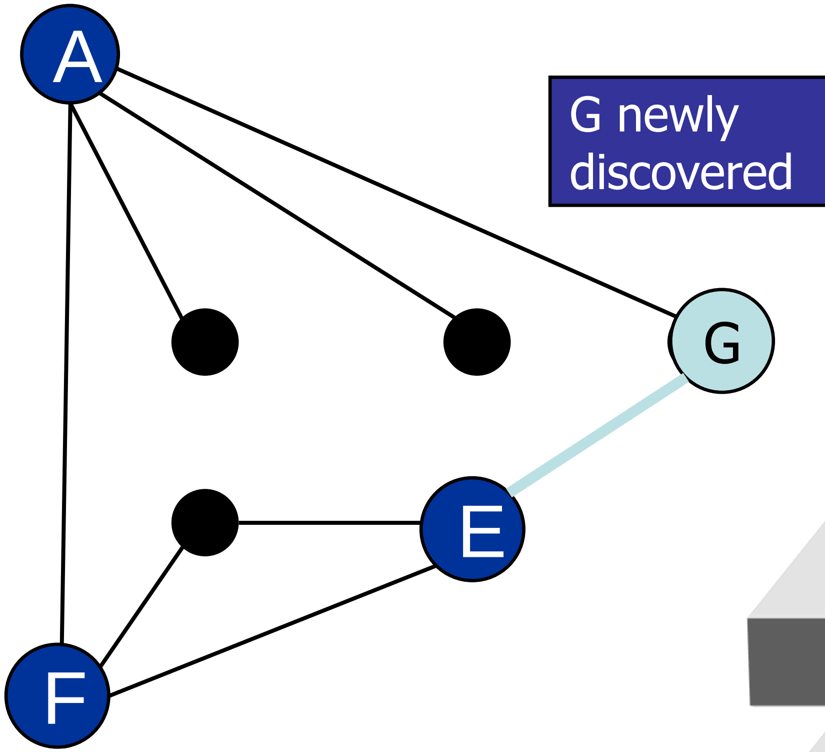
A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

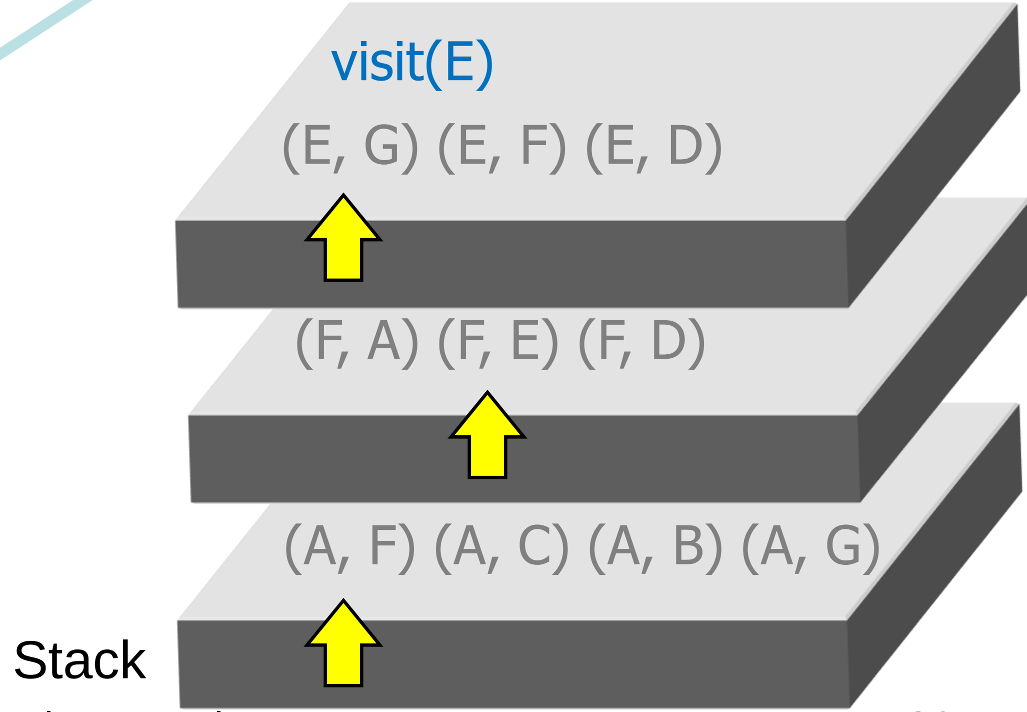


# Depth-First Search – Example

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



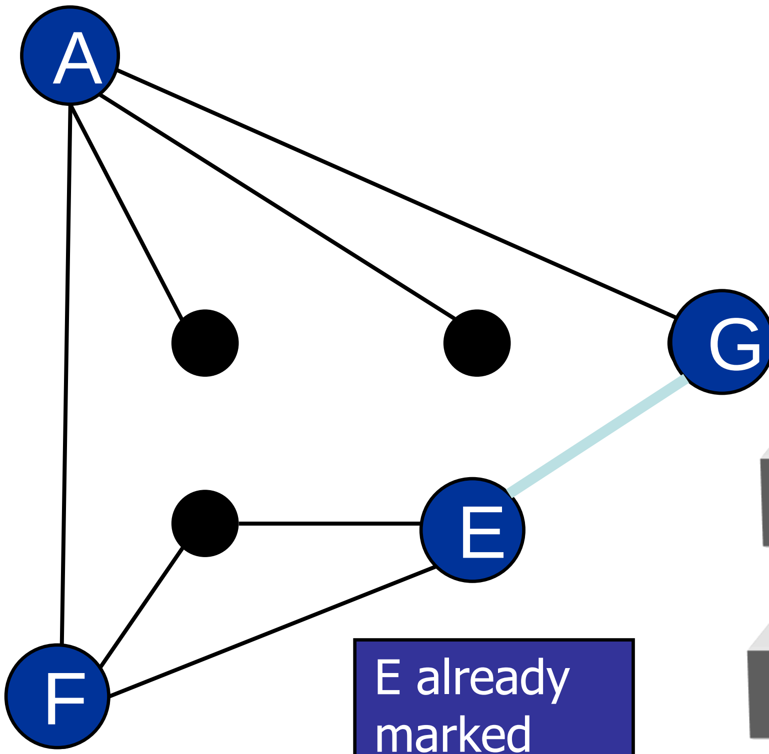




| |
|---|
| Undiscovered |
| Marked |
| Active |
| Finished |

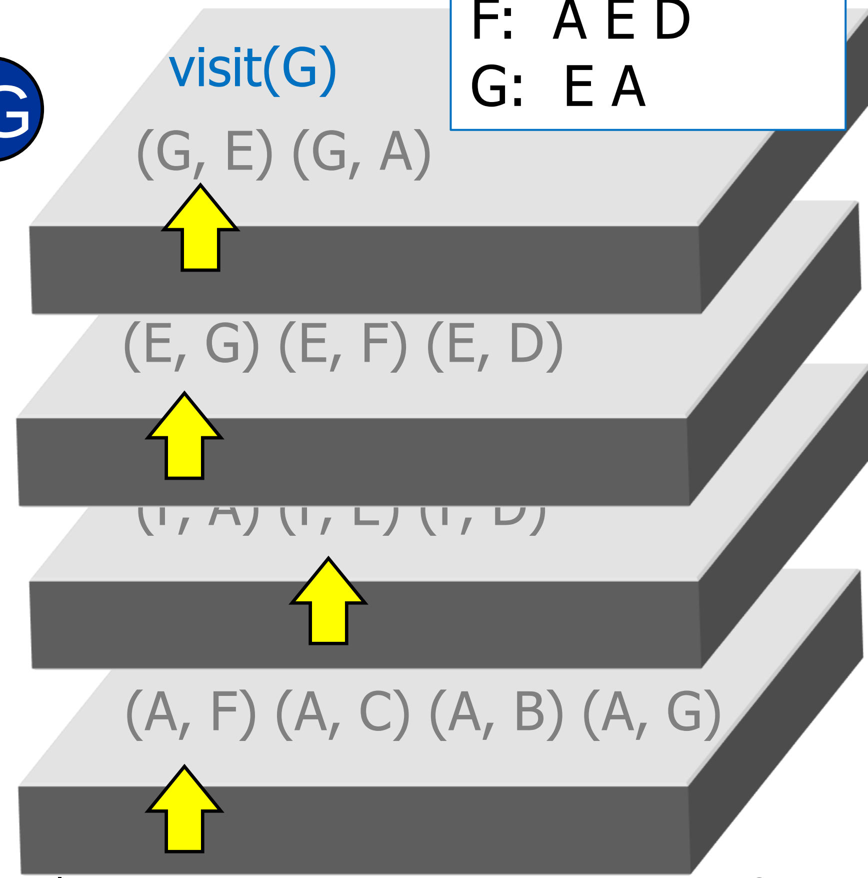
# Depth-First Search – Example

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




E already marked



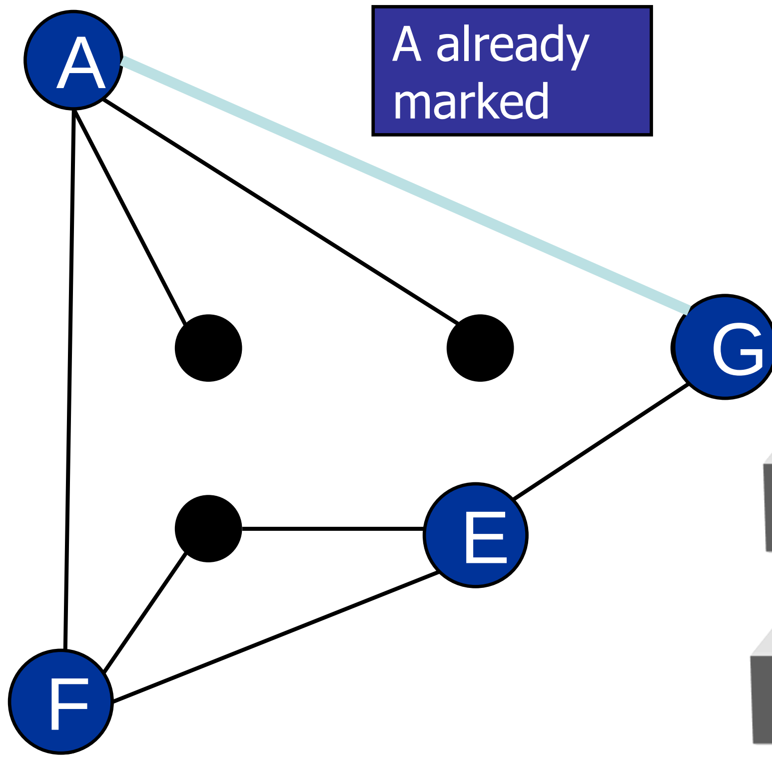

Stack

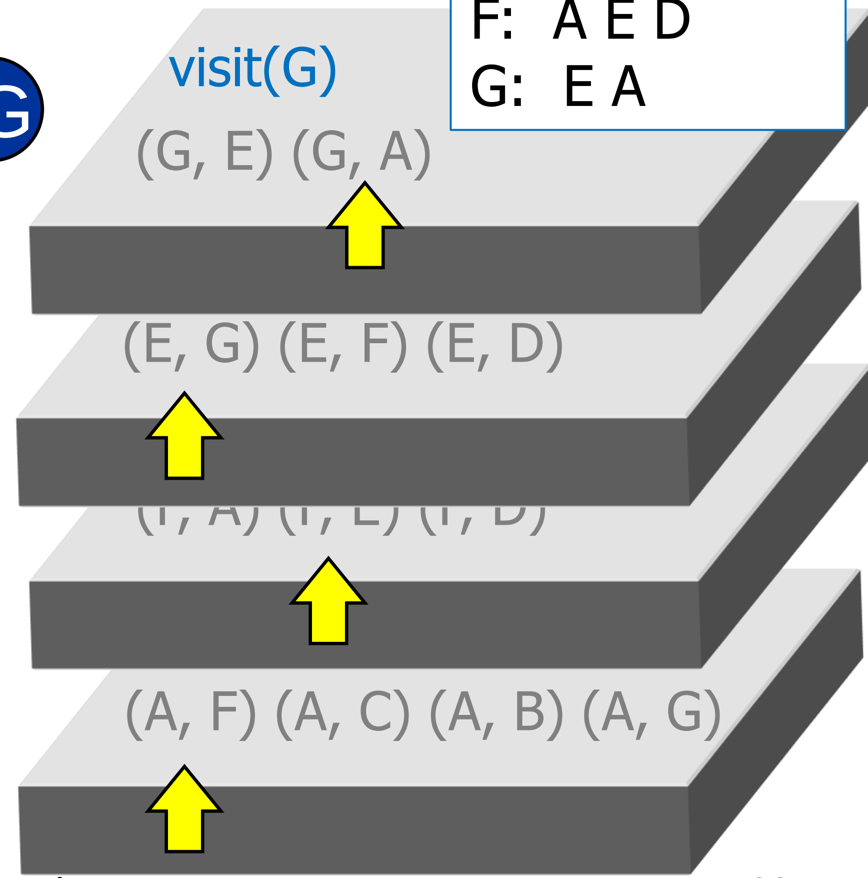
Undiscovered
Marked
Active
Finished

# Depth-First Search – Example

**Adjacency List**

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



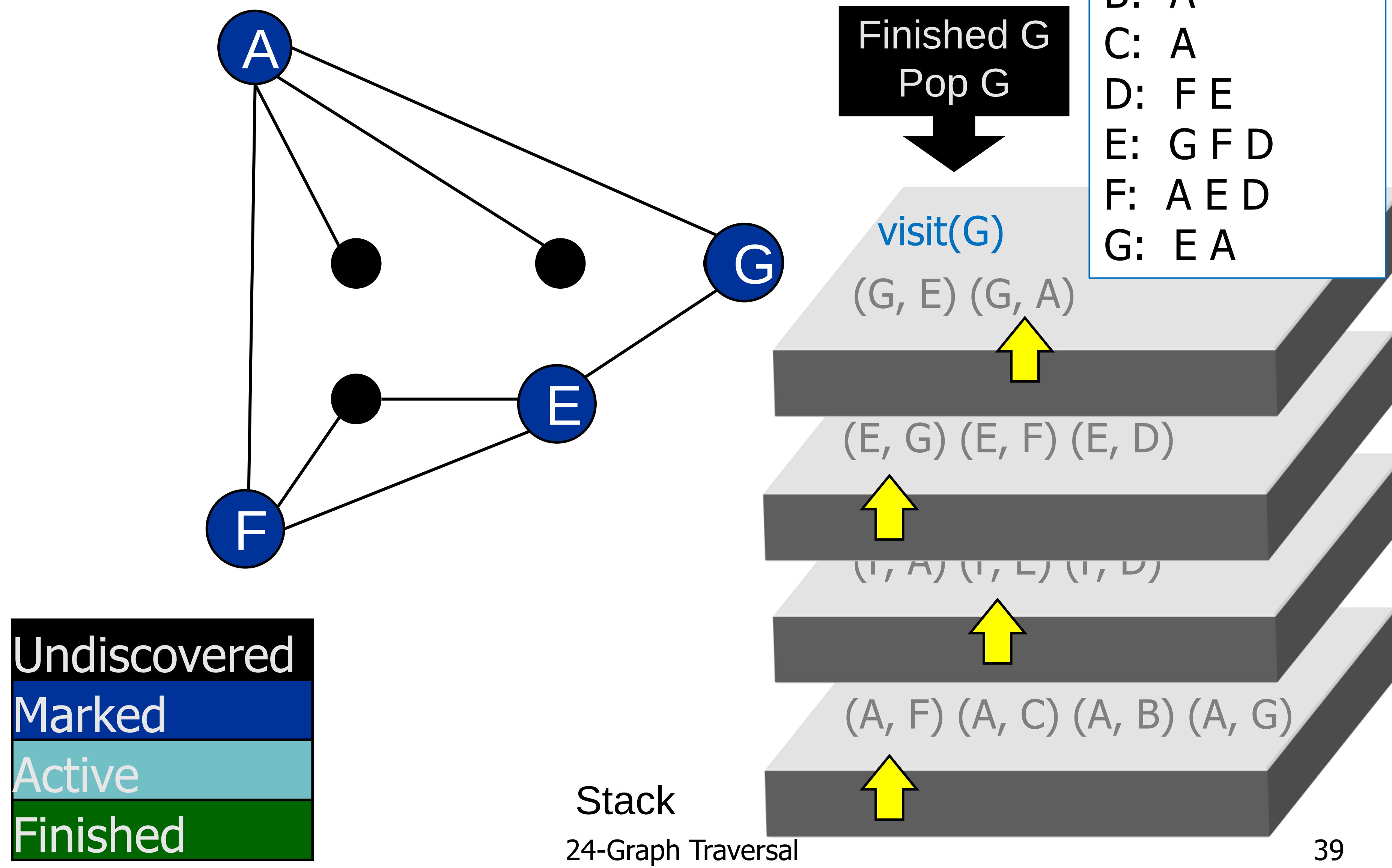

Stack

Undiscovered
Marked
Active
Finished

# Depth-First Search – Example

Adjacency List

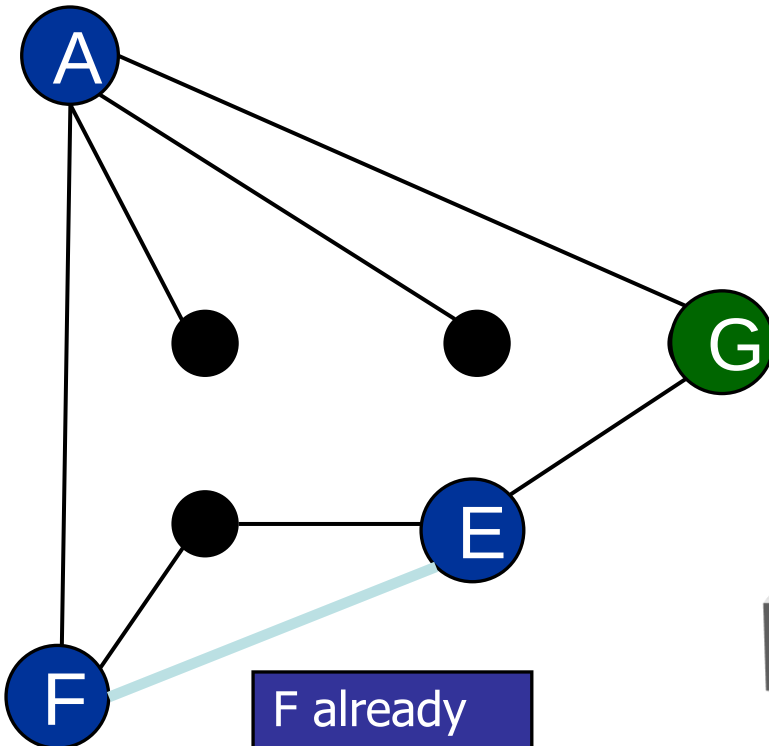
A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



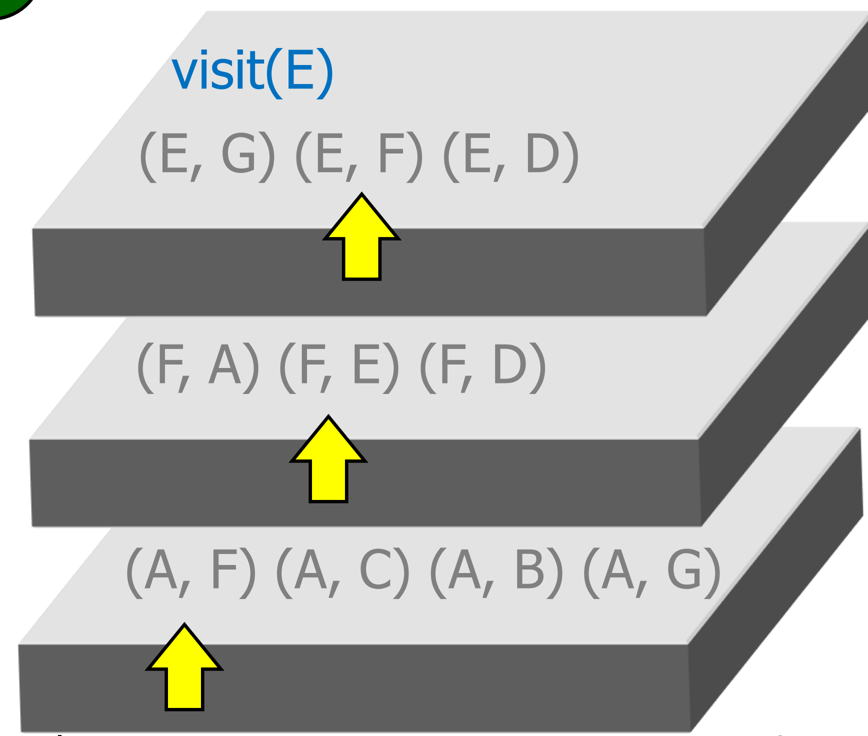
# Depth-First Search – Example

<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




F already marked



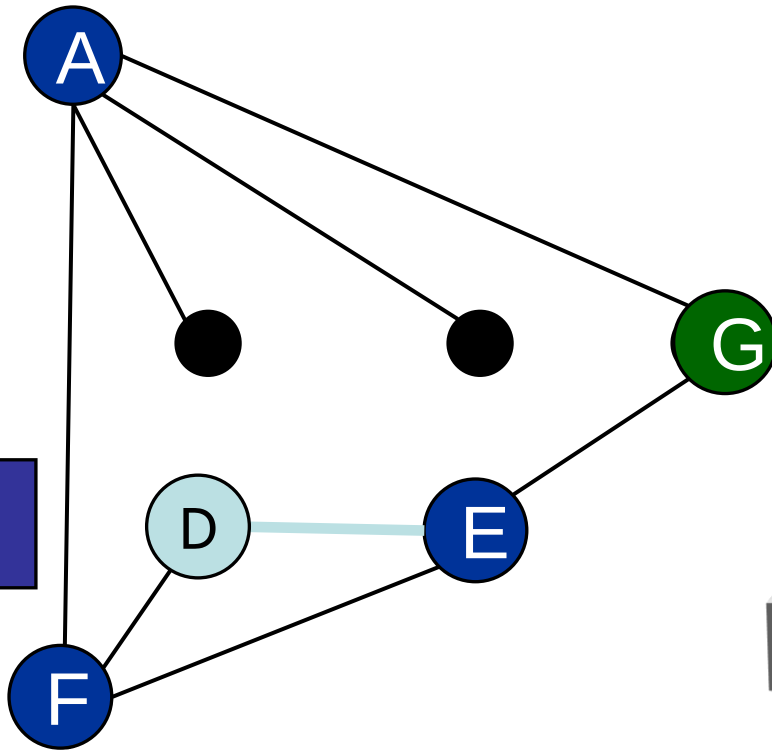

Stack

Undiscovered
Marked
Active
Finished

# Depth-First Search – Example

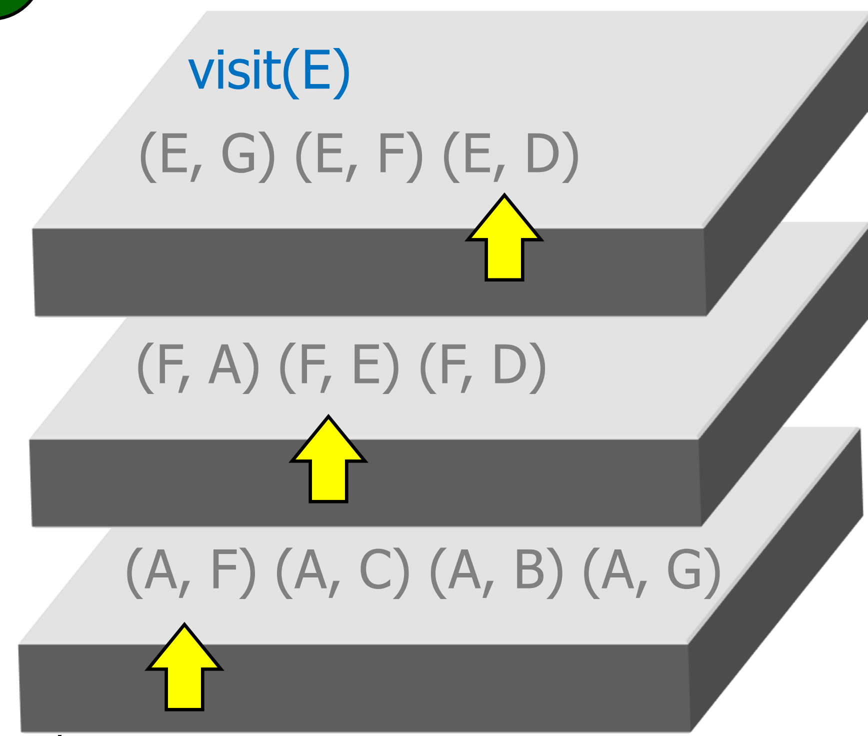
Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



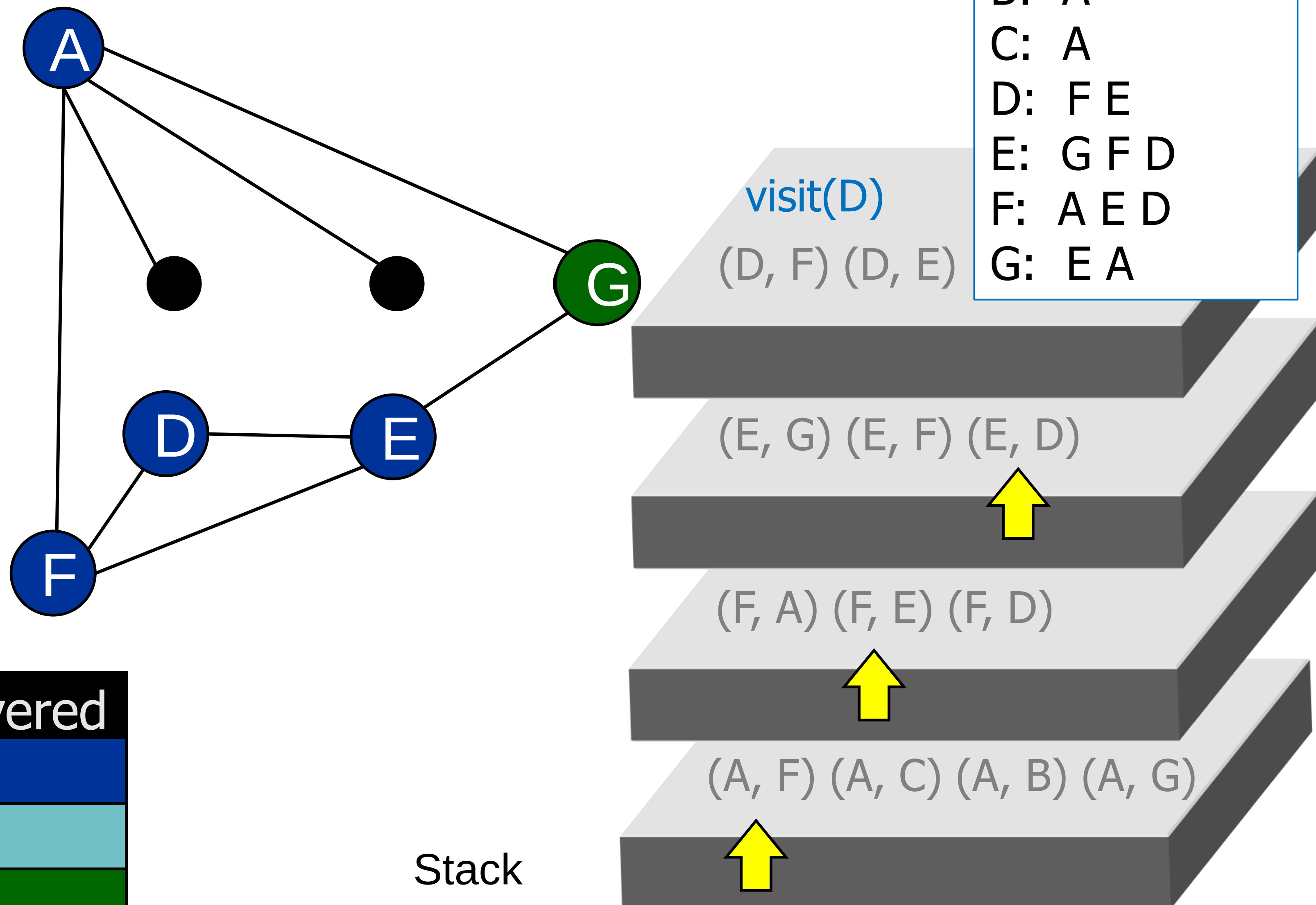

D newly discovered

Undiscovered
Marked
Active
Finished




Stack

# Depth-First Search – Example




Adjacency List

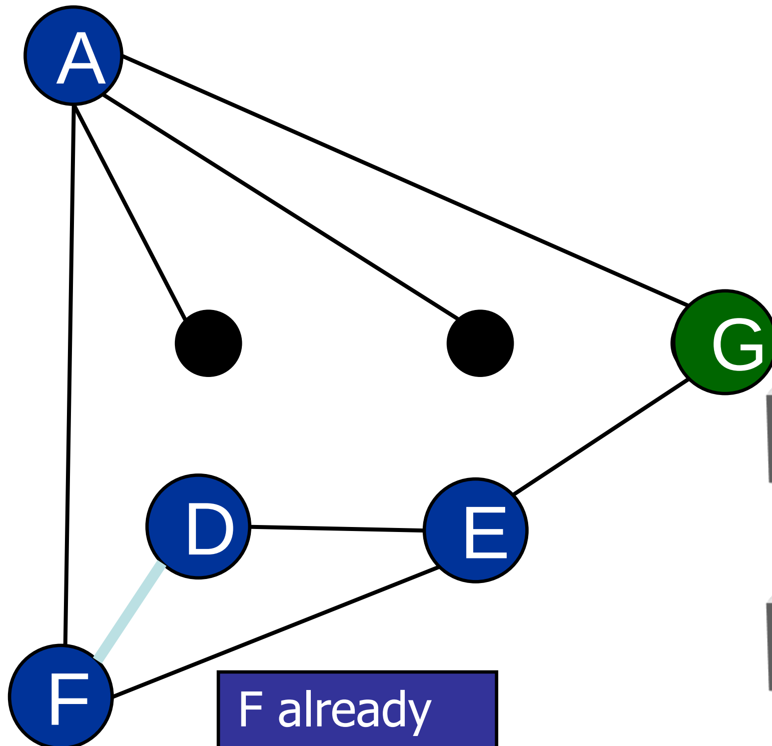
A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

| |
|---|
| Undiscovered |
| Marked |
| Active |
| Finished |

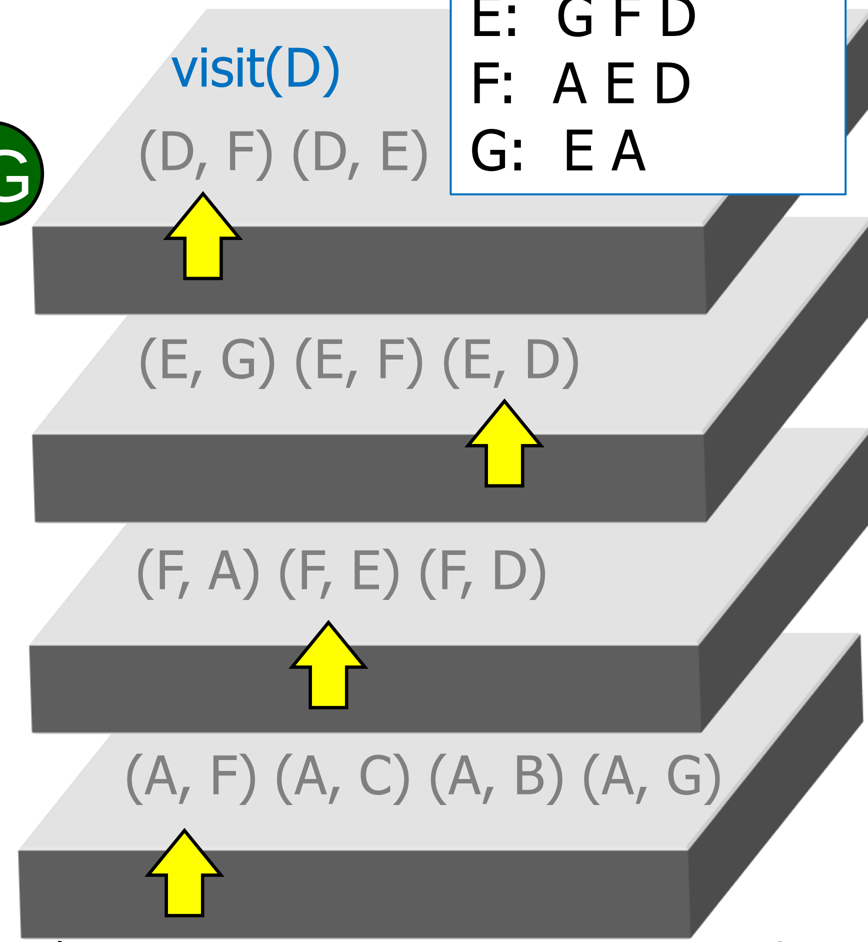
# Depth-First Search – Example

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




F already marked



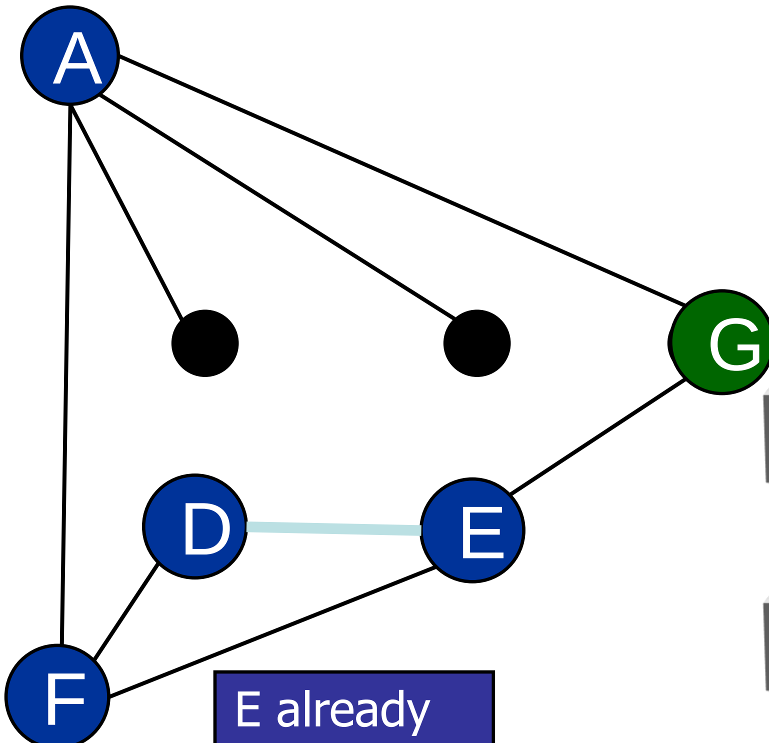

Stack

- Undiscovered
- Marked
- Active
- Finished

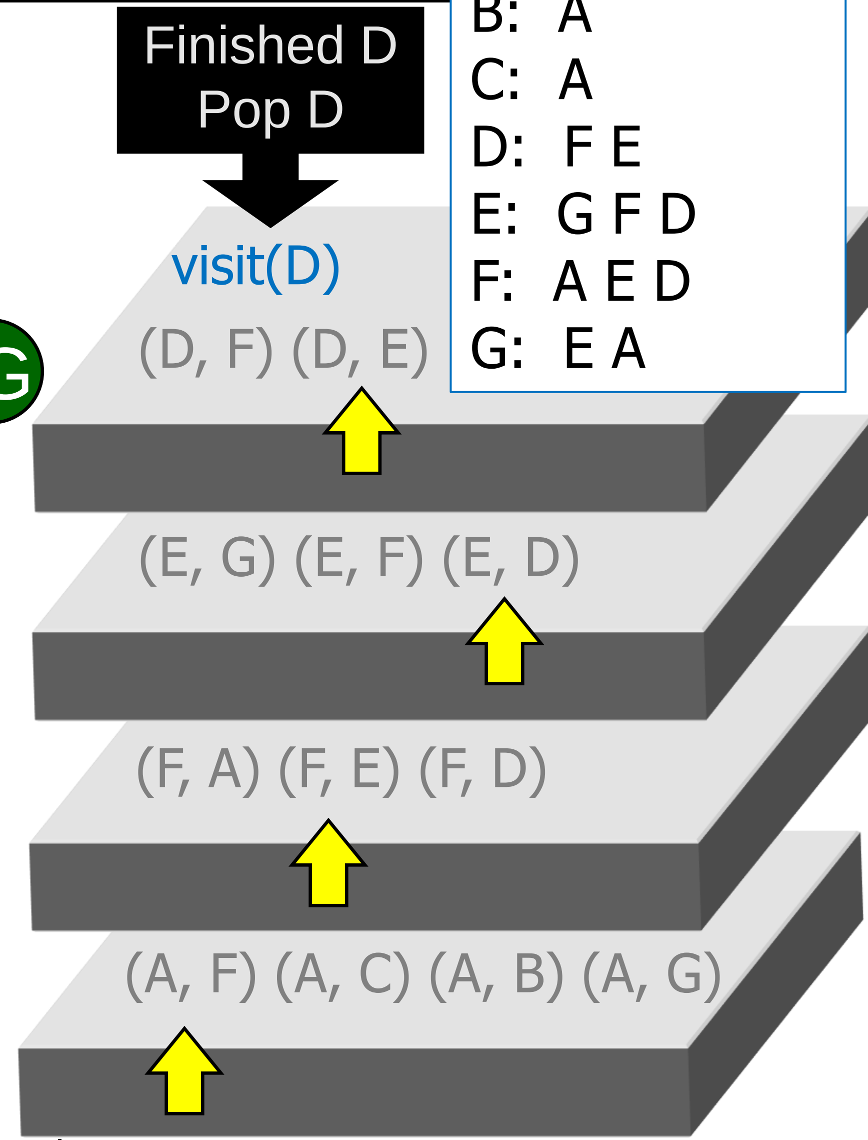
# Depth-First Search – Example

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



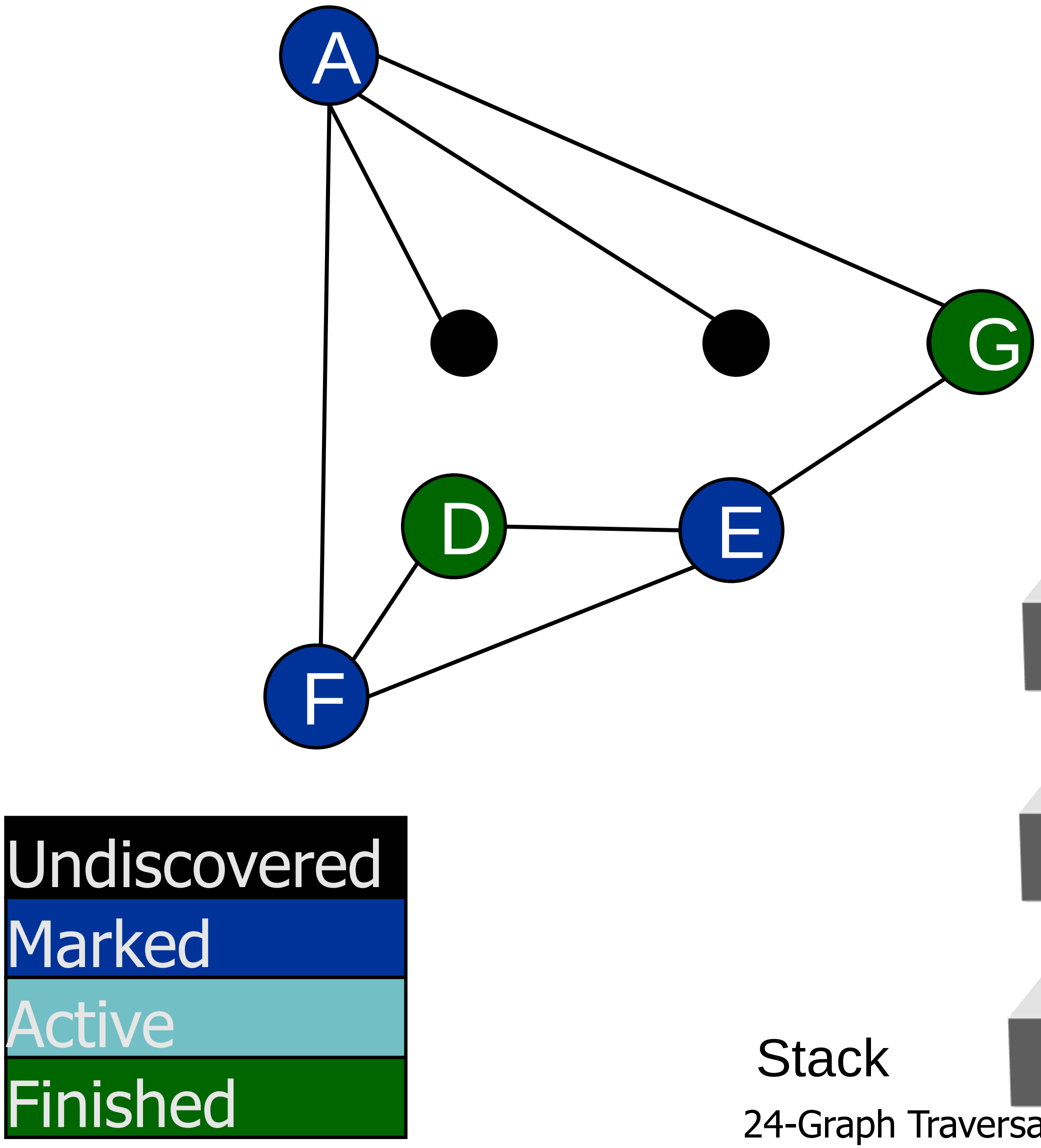

E already marked

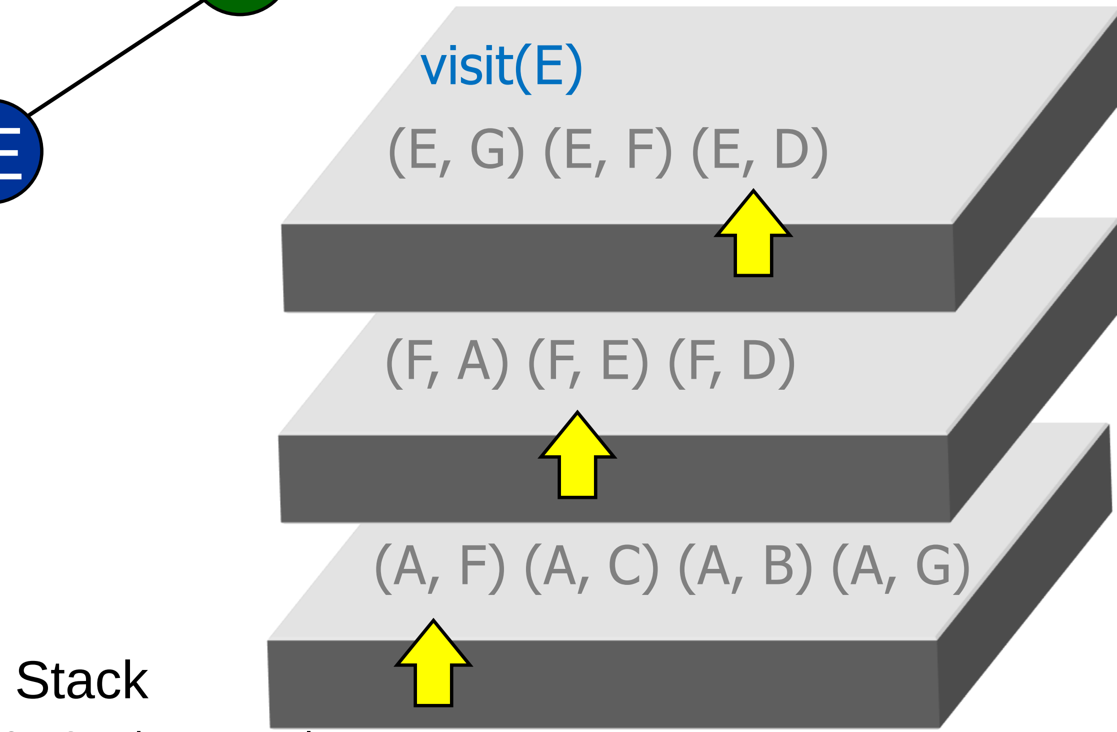



Stack

Undiscovered
Marked
Active
Finished

# Depth-First Search – Example



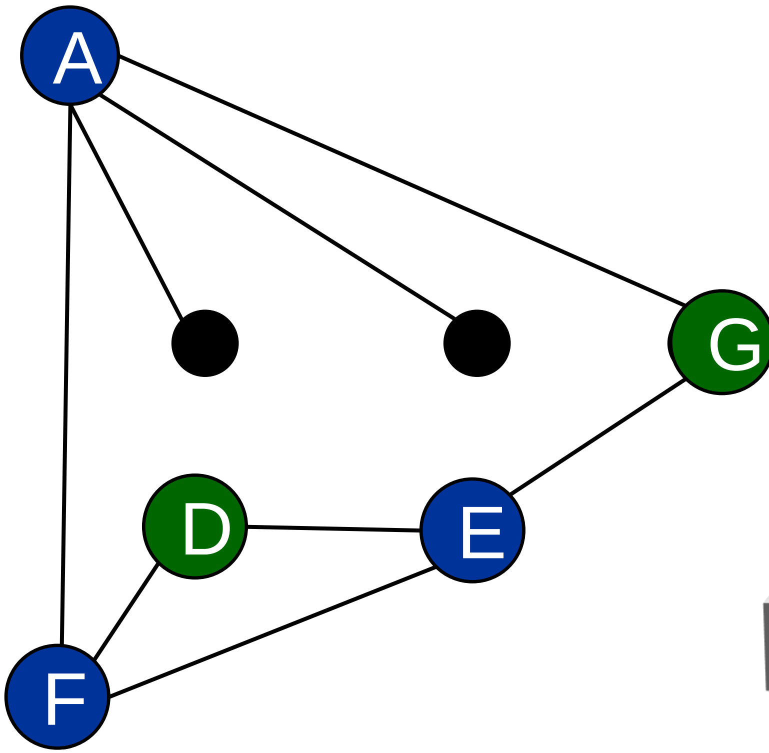

**Adjacency List**

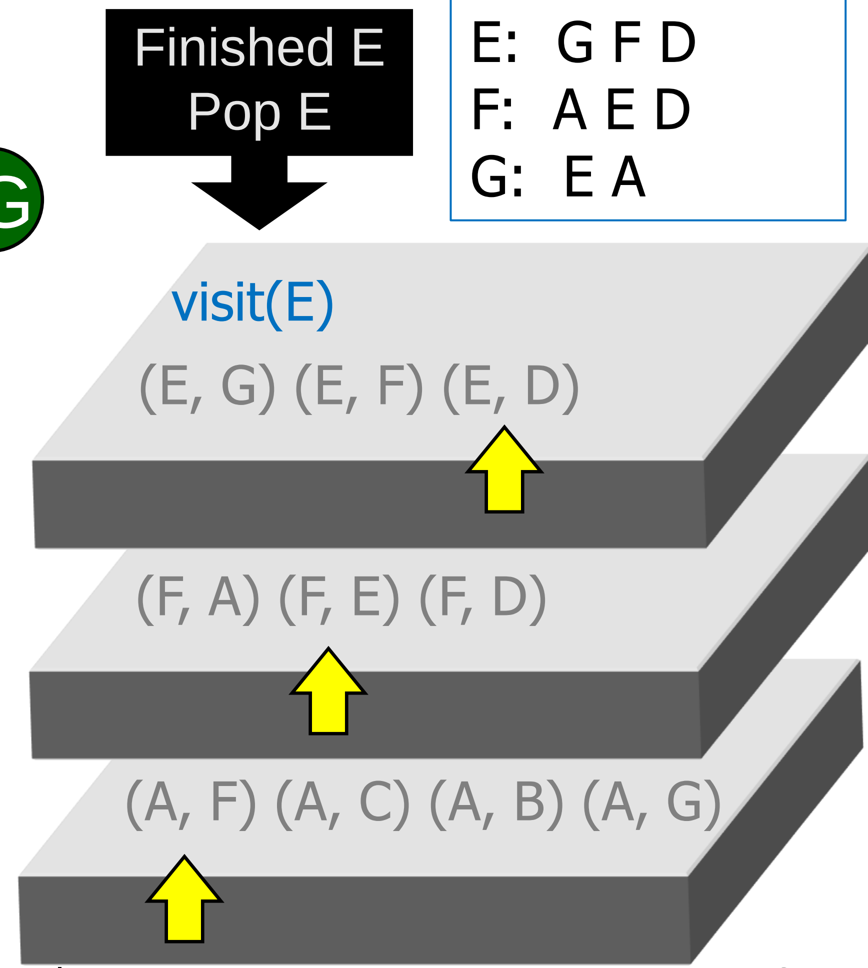
A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

# Depth-First Search – Example

<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

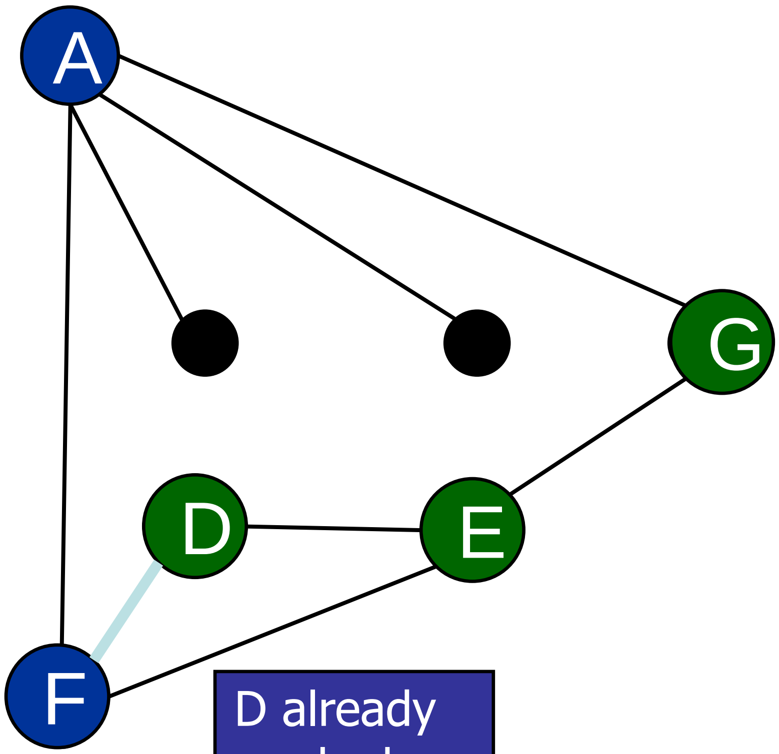



Stack

Undiscovered
Marked
Active
Finished

# Depth-First Search – Example

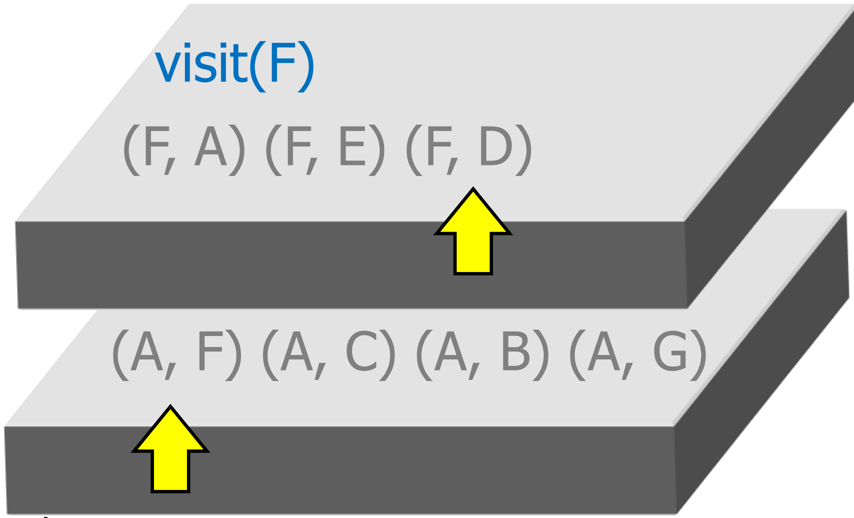



D already marked

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

Finished F
Pop F




Stack

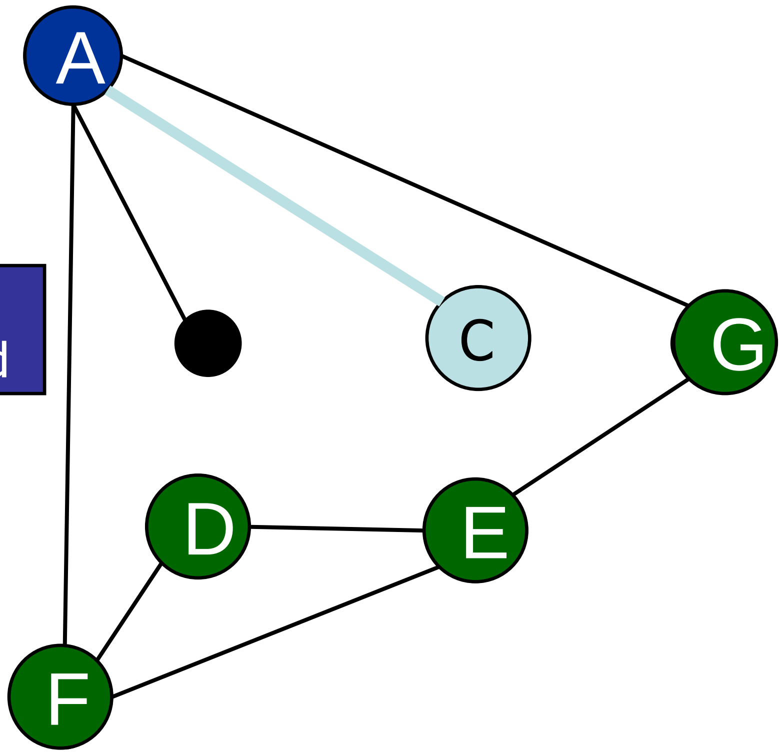
| |
|---|
| Undiscovered |
| Marked |
| Active |
| Finished |

# Depth-First Search – Example

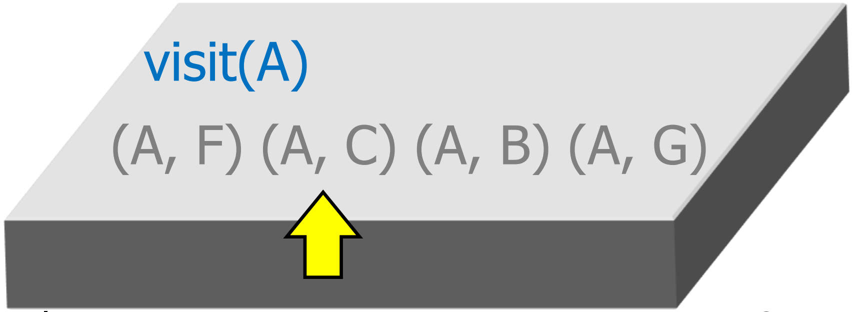
**Adjacency List**

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

C newly discovered



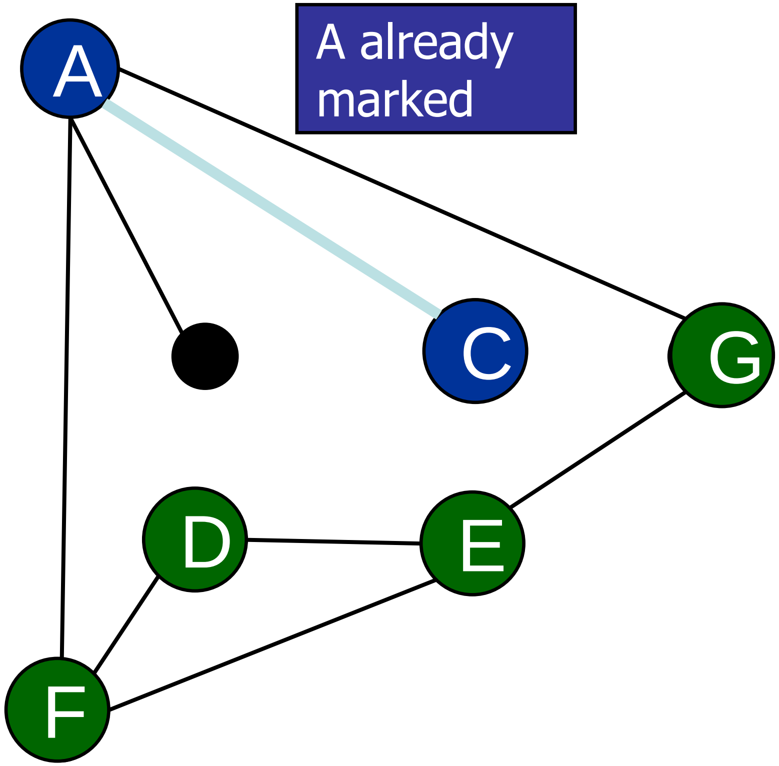

| |
|---|
| Undiscovered |
| Marked |
| Active |
| Finished |



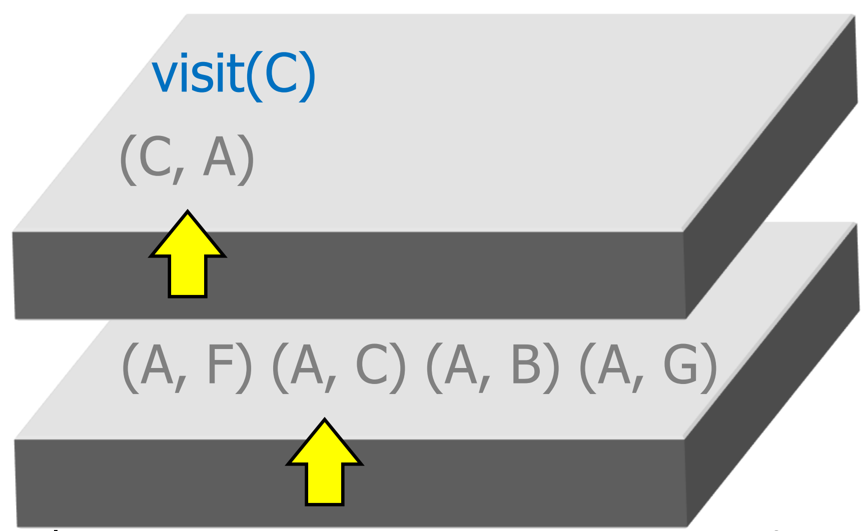
# Depth-First Search – Example

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




Undiscovered
Marked
Active
Finished

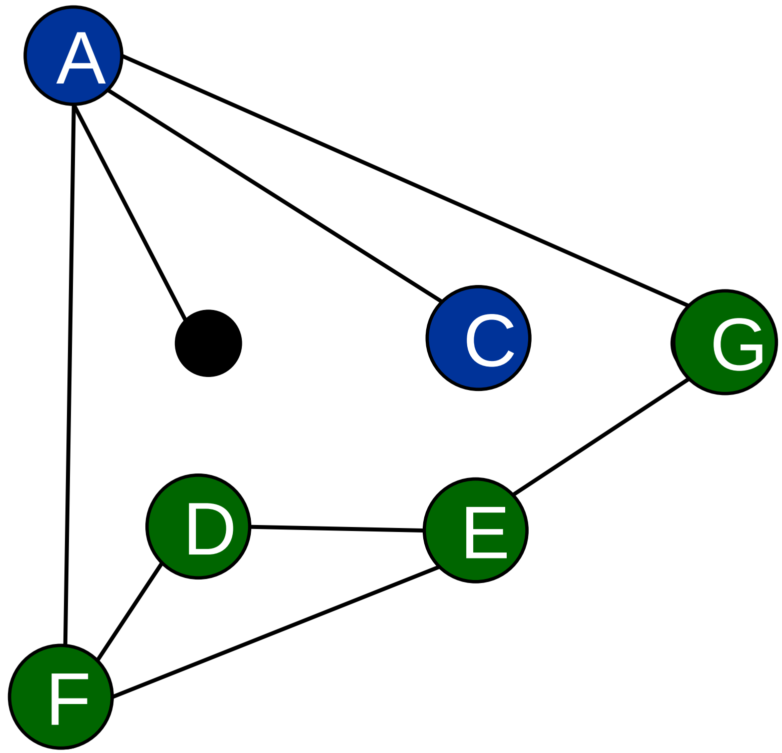


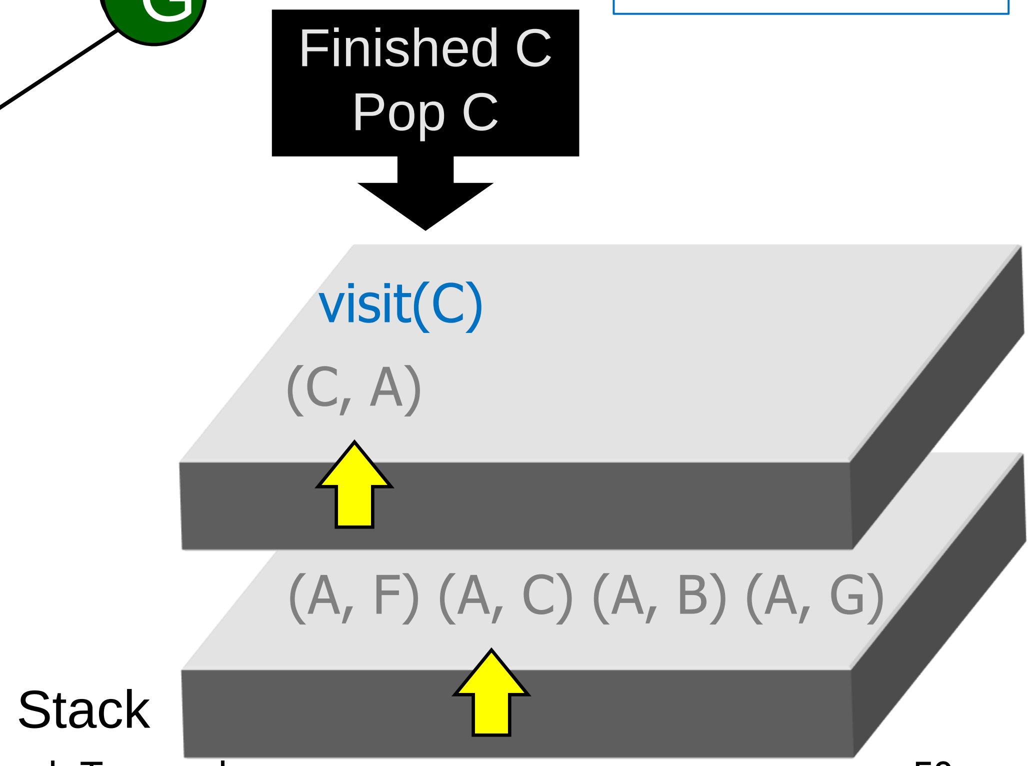

Stack

# Depth-First Search – Example

Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



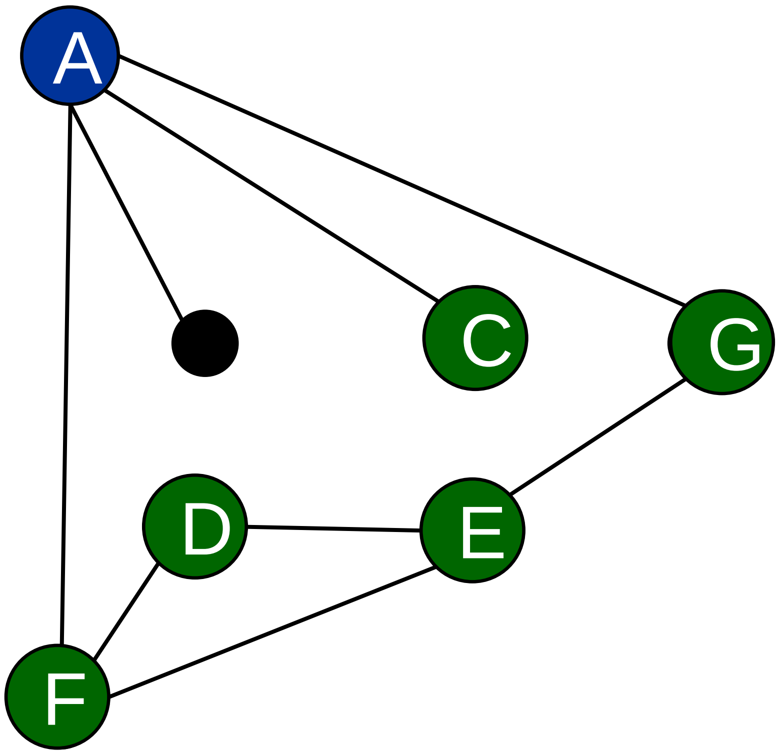




Undiscovered
Marked
Active
Finished

# Depth-First Search – Example

<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




Undiscovered
Marked
Active
Finished

visit(A)
(A, F) (A, C) (A, B) (A, G)

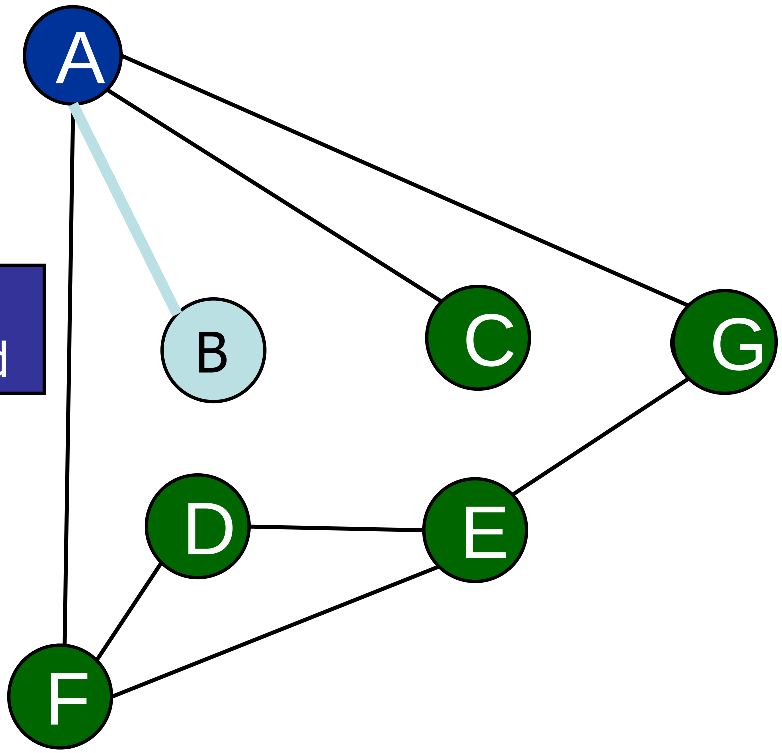
Stack

# Depth-First Search – Example

**Adjacency List**

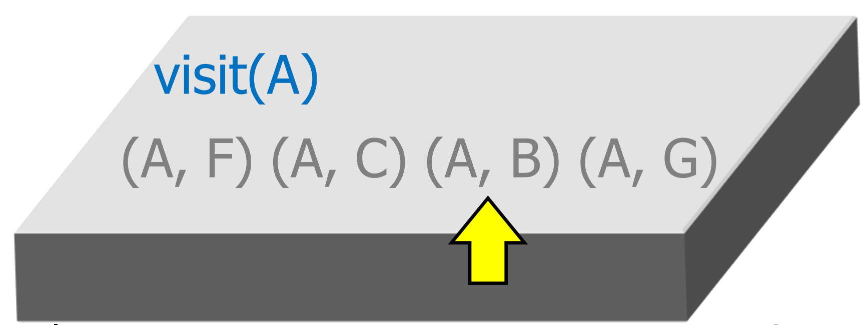
A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

B newly discovered




- Undiscovered
- Marked
- Active
- Finished

Stack

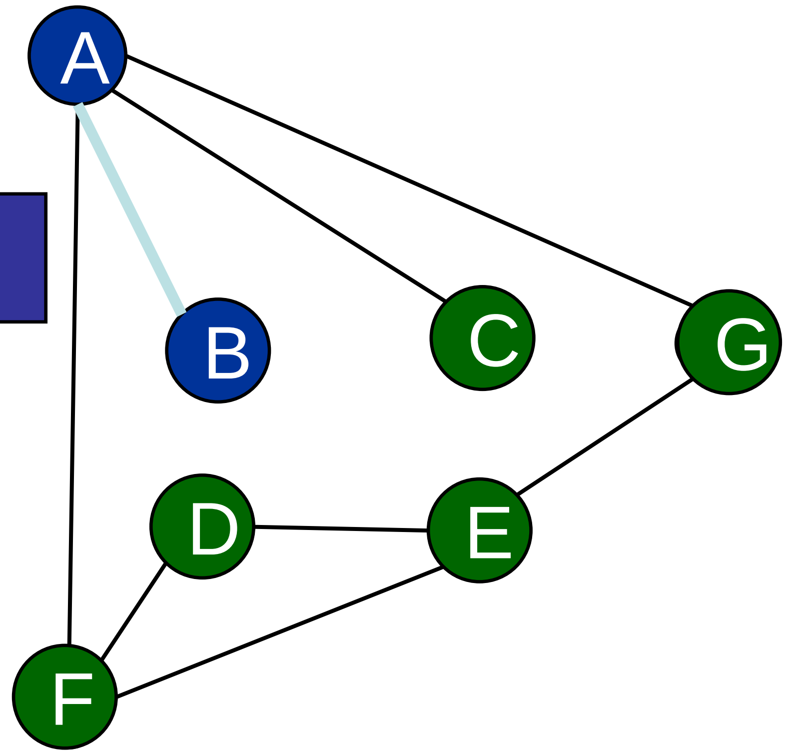


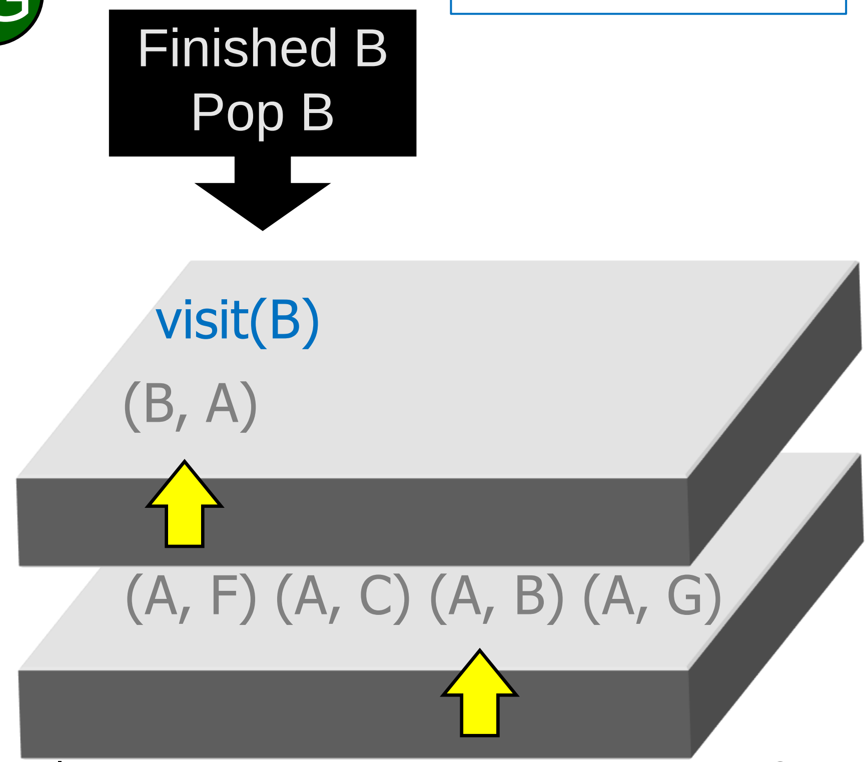
# Depth-First Search – Example

**Adjacency List**

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

A already marked



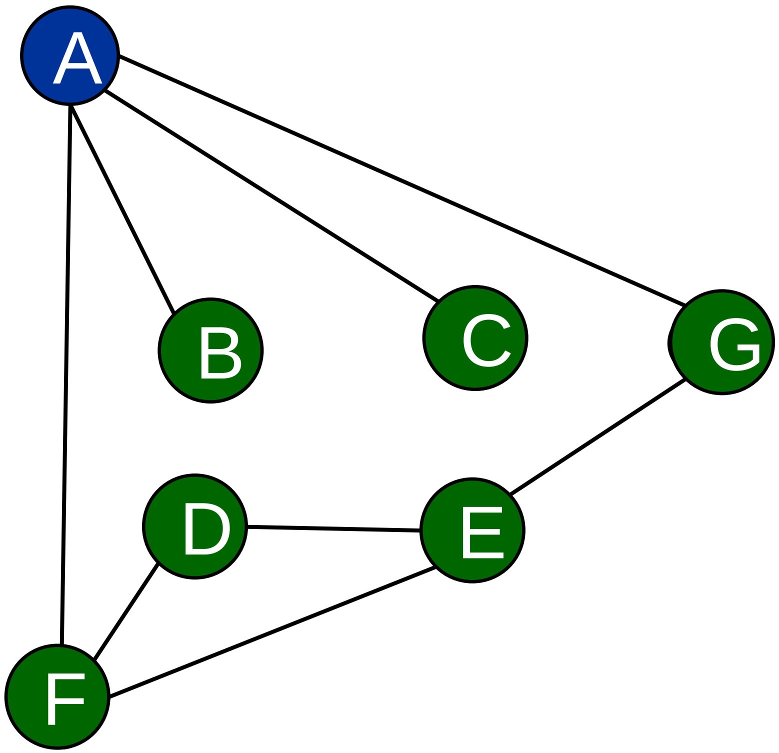

Stack

- Undiscovered
- Marked
- Active
- Finished

# Depth-First Search – Example

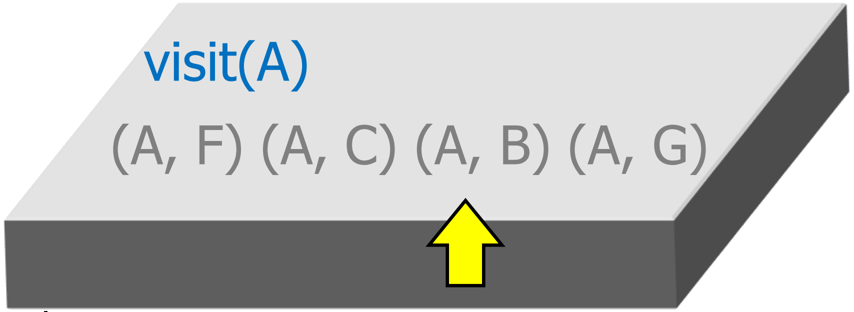
<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A




Undiscovered
Marked
Active
Finished

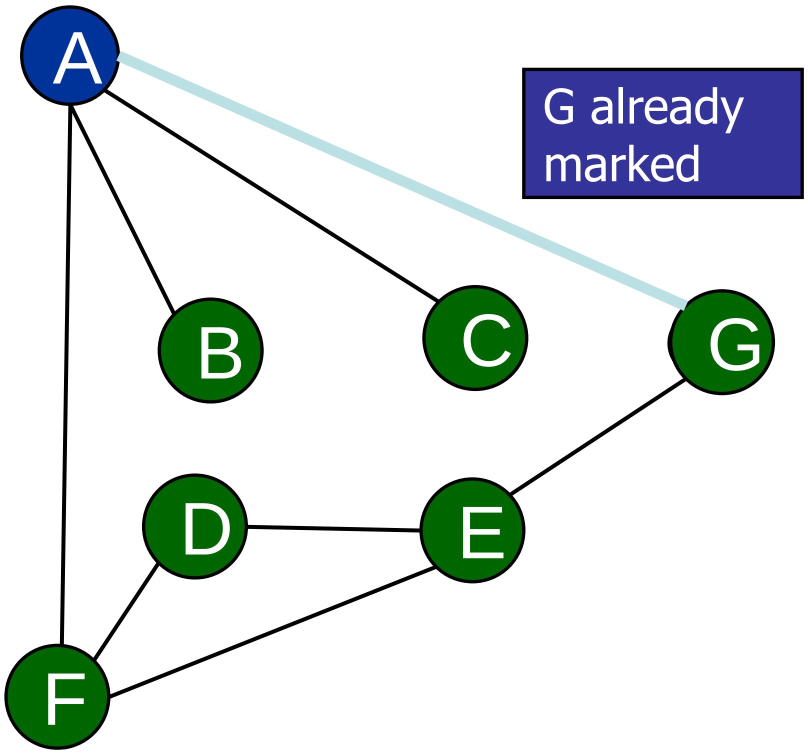
Stack



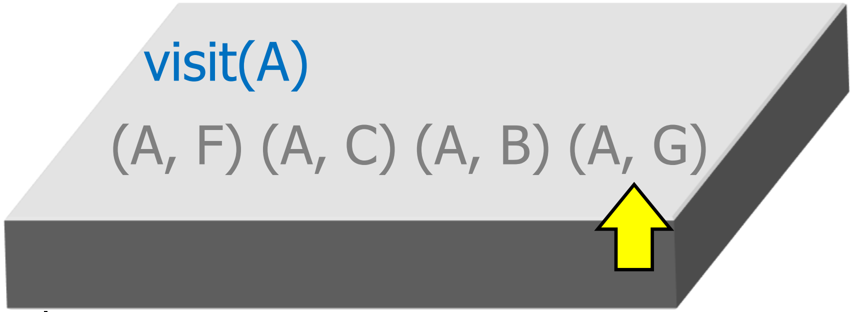
# Depth-First Search – Example

<u>Adjacency List</u>

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A



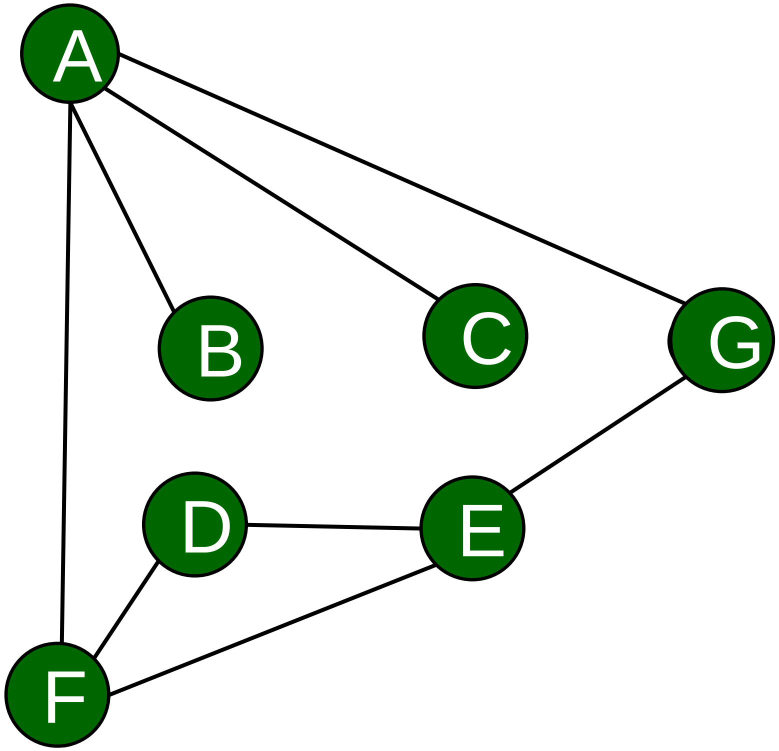

Finished A
Pop A




Stack

Undiscovered
Marked
Active
Finished

# Depth-First Search – Example




Adjacency List

A: F C B G
B: A
C: A
D: F E
E: G F D
F: A E D
G: E A

# BFS vs. DFS

- Depending on the application, either DFS or BFS could be advantageous

- Example: Consider your family tree

  - If you are searching for some of your siblings/cousins then it would be safe to assume that person would be on the bottom of the tree

  - Which approach is better in this case?

    - In general, both approaches have the same time complexity

    - In worst case, they need to visit all the nodes

# Applications of Graph Traversal

- Determining connectedness and finding connected sub-graphs
- Construct a BFS or DFS tree/forest from a graph
- Determining the path length from one vertex to all others
  - Find the shortest path from a vertex s to a vertex v (BFS)

# Any Question So Far?