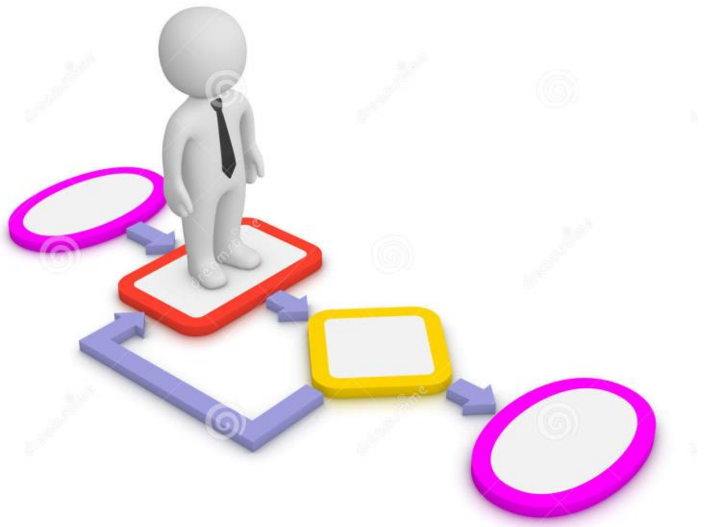
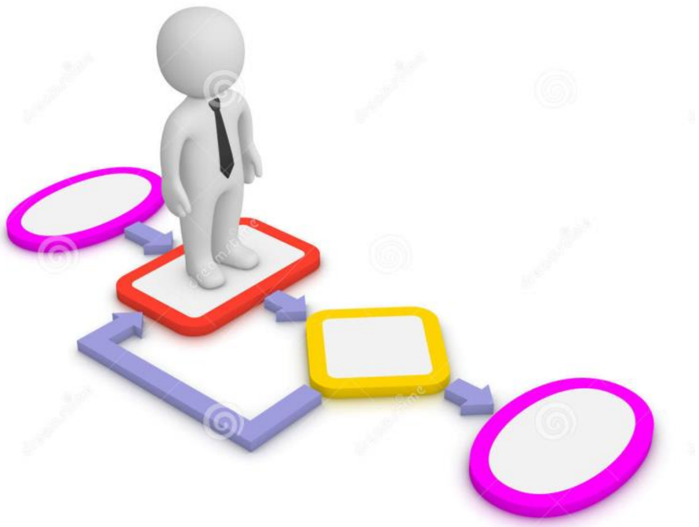


Design & Analysis of Algorithms

Saman Riaz (PhD)
Associate Professor
RSCI, Lahore

Lecture # 12

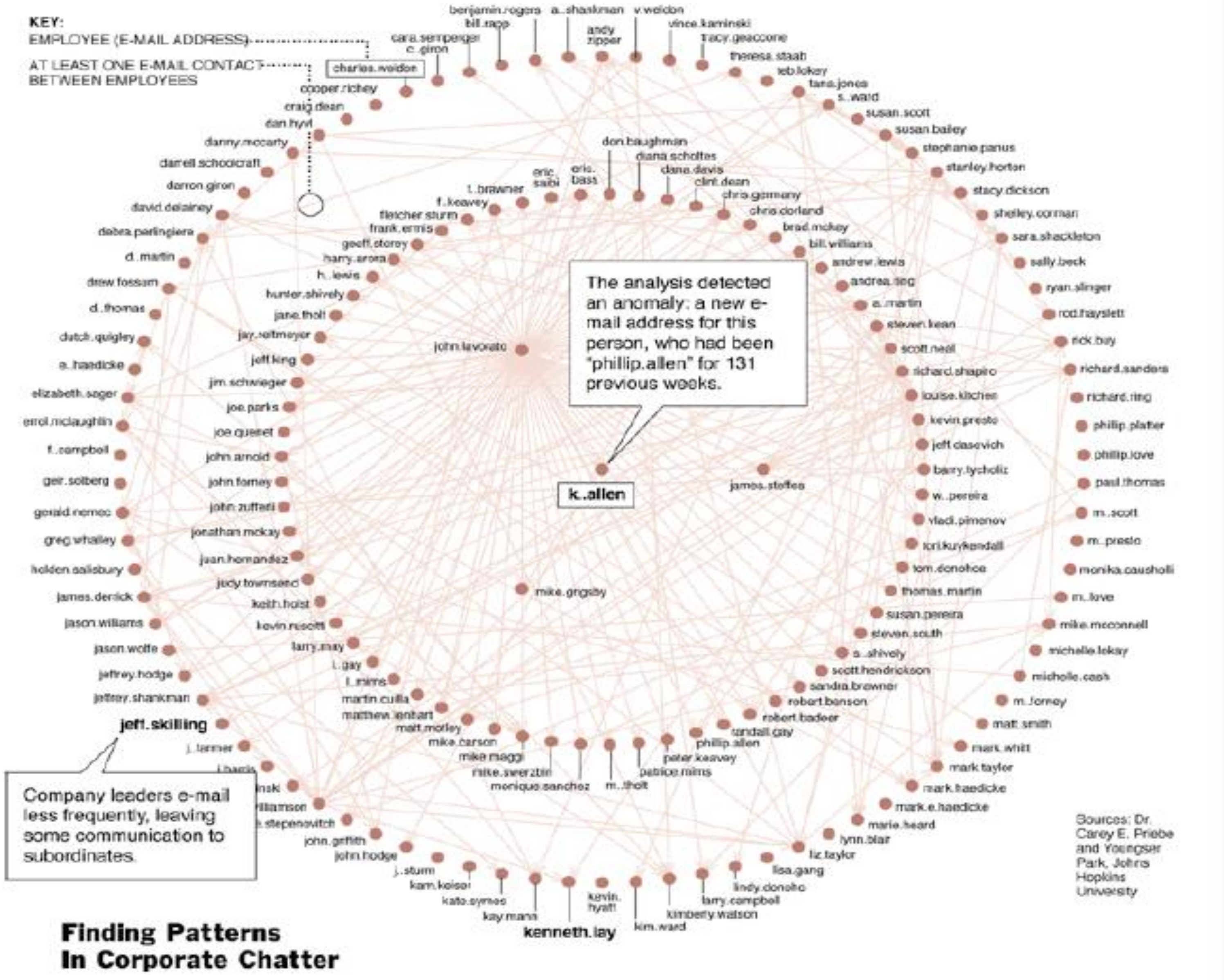


ELEMENTARY GRAPH ALGORITHMS

GRAPHS

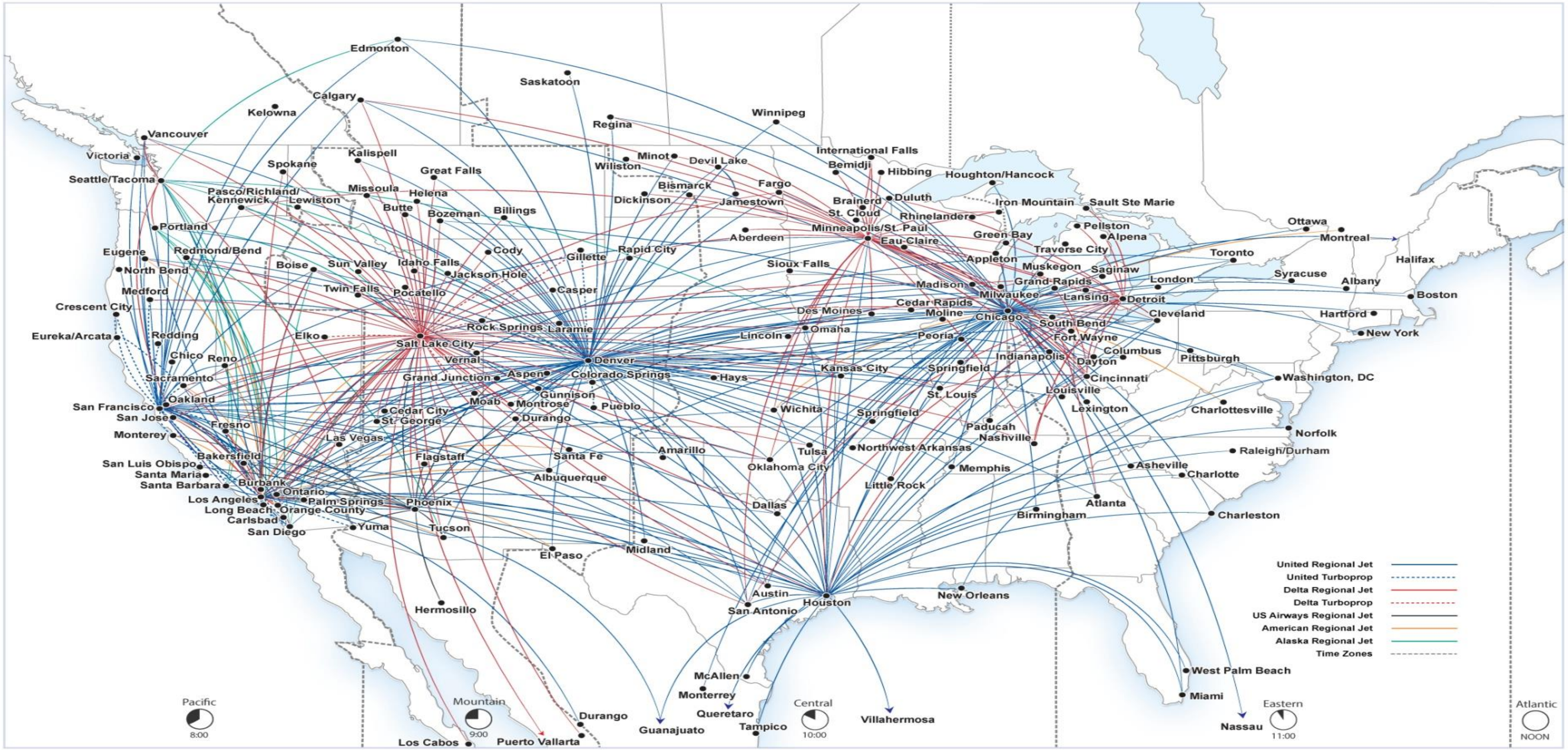
Basic definitions and applications

One week of Enron emails



Route Map

SkyWest AIRLINES®



UNITED EXPRESS

DELTA CONNECTION

U·S AIRWAYS EXPRESS

American Eagle

Alaska Airlines Operated by SkyWest

Framingham heart study

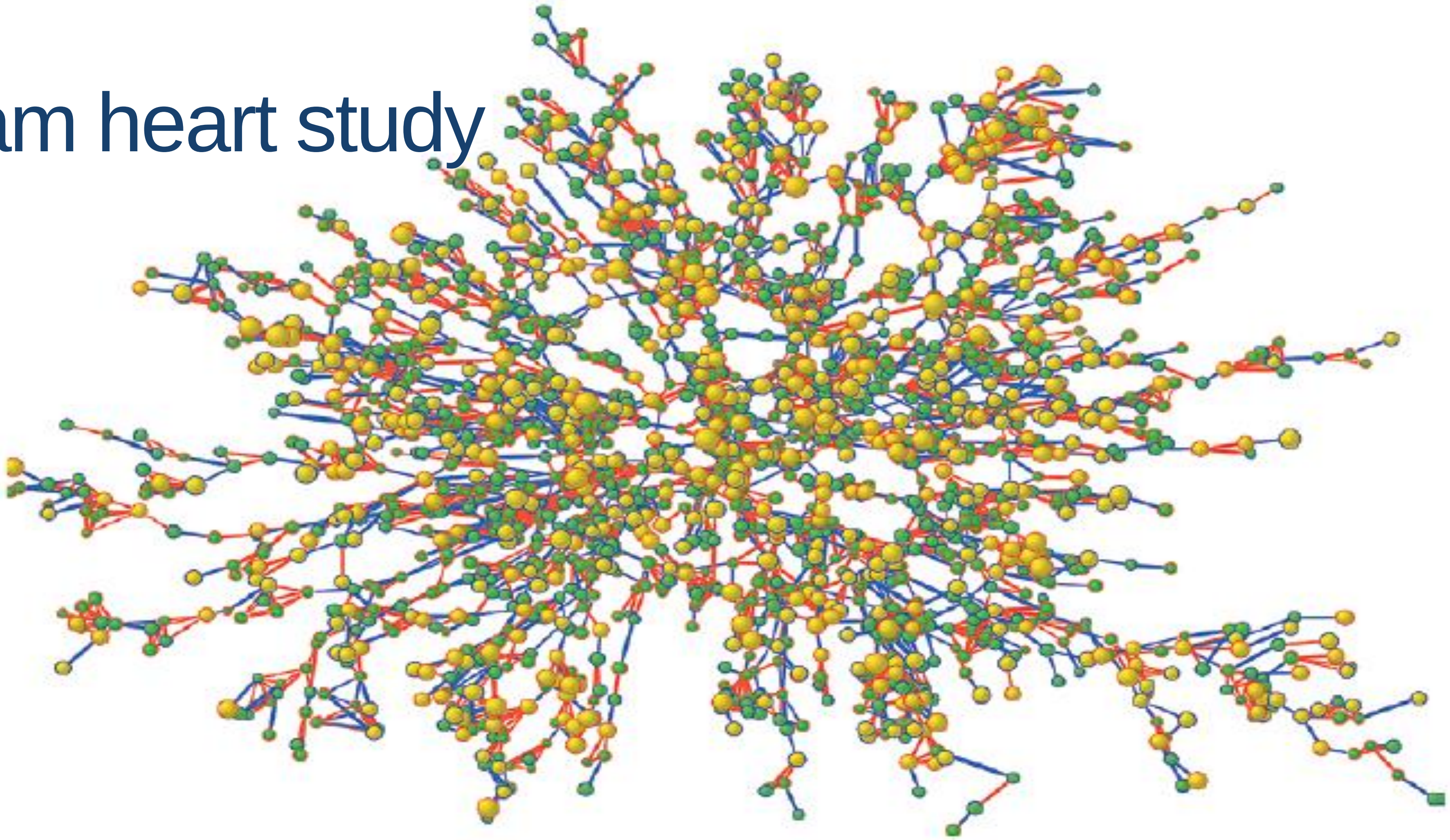


Figure 1. Largest Connected Subcomponent of the Social Network in the Framingham Heart Study in the Year 2000.

Each circle (node) represents one person in the data set. There are 2200 persons in this subcomponent of the social network. Circles with red borders denote women, and circles with blue borders denote men. The size of each circle is proportional to the person's body-mass index. The interior color of the circles indicates the person's obesity status: yellow denotes an obese person (body-mass index, ≥30) and green denotes a nonobese person. The colors of the ties between the nodes indicate the relationship between them: purple denotes a friendship or marital tie and orange denotes a familial tie.

"The Spread of Obesity in a Large Social Network over 32 Years" by Christakis and Fowler in New England Journal of Medicine, 2007

Some graph applications

graph	node	edge
communication	telephone, computer	fiber optic cable
circuit	gate, register, processor	wire
mechanical	joint	rod, beam, spring
financial	stock, currency	transactions
transportation	street intersection, airport	highway, airway route
internet	class C network	connection
game	board position	legal move
social relationship	person, actor	friendship, movie cast
neural network	neuron	synapse
protein network	protein	protein-protein interaction
molecule	atom	bond

Graphs

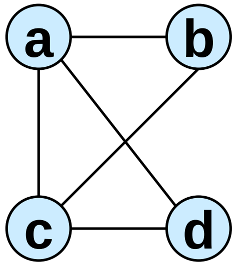
- *Graph G* = (V, E)
  - V = set of vertices
  - E = set of edges $\subseteq (V \times V)$
- Types of graphs
  - Undirected: edge $(u, v) = (v, u)$; for all v, $(v, v) \notin E$ (No self loops.)
  - Directed: (u, v) is edge from u to v, denoted as $u \rightarrow v$. Self loops are allowed.
  - Weighted: each edge has an associated weight, given by a weight function $w : E \rightarrow \mathbf{R}$.
  - Dense: $|E| \approx |V|^2$.
  - Sparse: $|E| << |V|^2$.
- $|E| = O(|V|^2)$

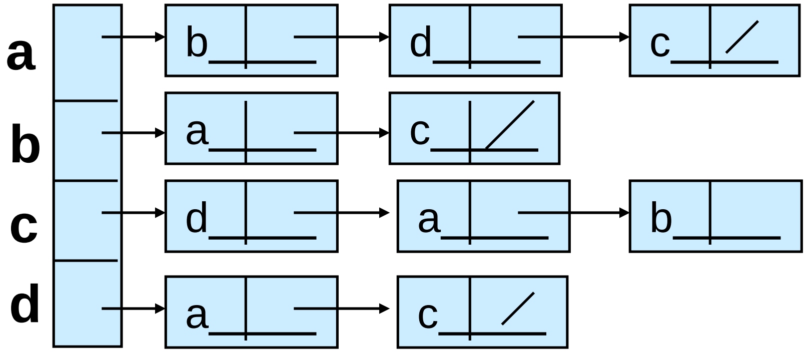
Graphs

- If $(u, v) \in E$, then vertex v is adjacent to vertex u.
- Adjacency relationship is:
  - Symmetric if G is undirected.
  - Not necessarily so if G is directed.
- If G is connected:
  - There is a path between every pair of vertices.
  - $|E| \geq |V| - 1$.
  - Furthermore, if $|E| = |V| - 1$, then G is a tree.

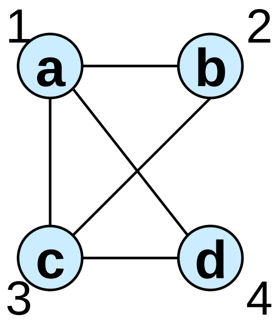
Representation of Graphs

- Two standard ways.
  - Adjacency Lists.





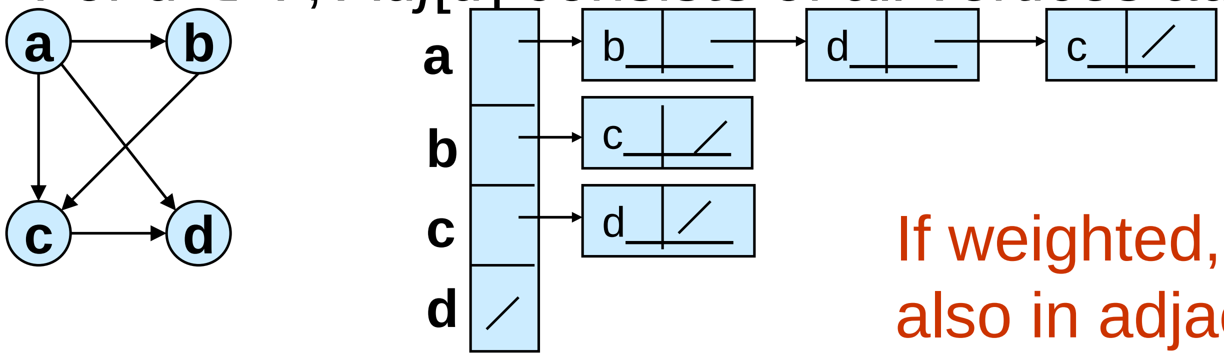
  - Adjacency Matrix.



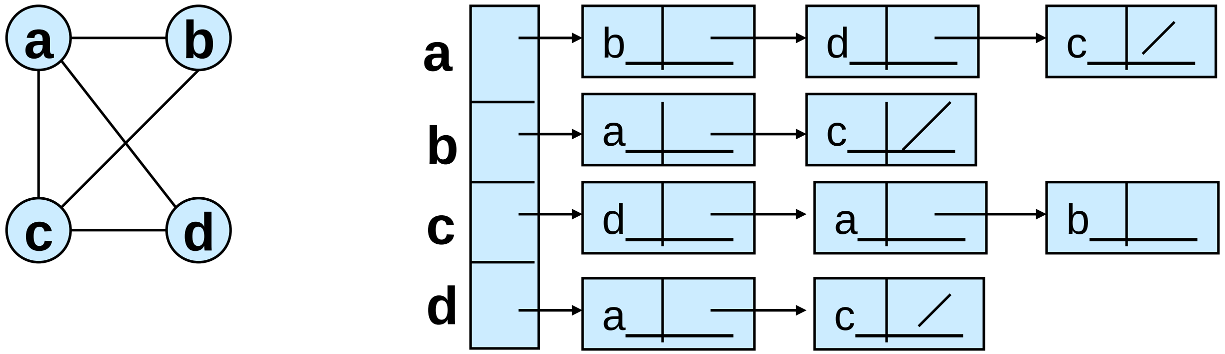
	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Adjacency Lists

- Consists of an array *Adj* of $|V|$ lists.
- One list per vertex.
- For $u \in V$, $Adj[u]$ consists of all vertices adjacent to u.



If weighted, store weights also in adjacency lists.



Storage Requirement

- For directed graphs:
  - Sum of lengths of all adj. lists is
    $$\sum_{v \in V} \text{out-degree}(v) = |E|$$
    No. of edges leaving v
  - Total storage: $\Theta(V+E)$
- For undirected graphs:
  - Sum of lengths of all adj. lists is
    $$\sum_{v \in V} \text{degree}(v) = 2|E|$$
    No. of edges incident on v. Edge (u,v) is incident on vertices u and v.
  - Total storage: $\Theta(V+E)$

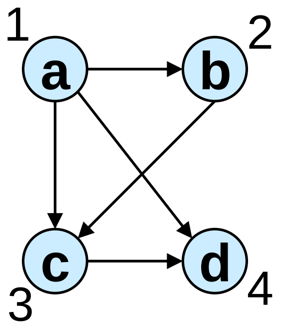
Pros and Cons: adj list

- Pros
  - Space-efficient, when a graph is sparse.
  - Can be modified to support many graph variants.
- Cons
  - Determining if an edge $(u,v) \in G$ is not efficient.
    - Have to search in u's adjacency list. $\Theta(\text{degree}(u))$ time.
    - $\Theta(V)$ in the worst case.

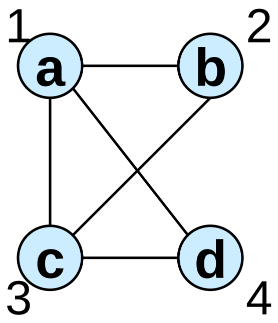
Adjacency Matrix

- $|V| \times |V|$ matrix A.
- Number vertices from 1 to $|V|$ in some arbitrary manner.
- A is then given by:

$$A[i, j] = a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

$A = A^{\mathrm{T}}$ for undirected graphs.

Space and Time

- **Space:** $\Theta(V^2)$.
  - Not memory efficient for large graphs.
- **Time:** to list all vertices adjacent to u: $\Theta(V)$.
- **Time:** to determine if $(u, v) \in E$: $\Theta(1)$.
- Can store weights instead of bits for weighted graph.

Graph-searching Algorithms

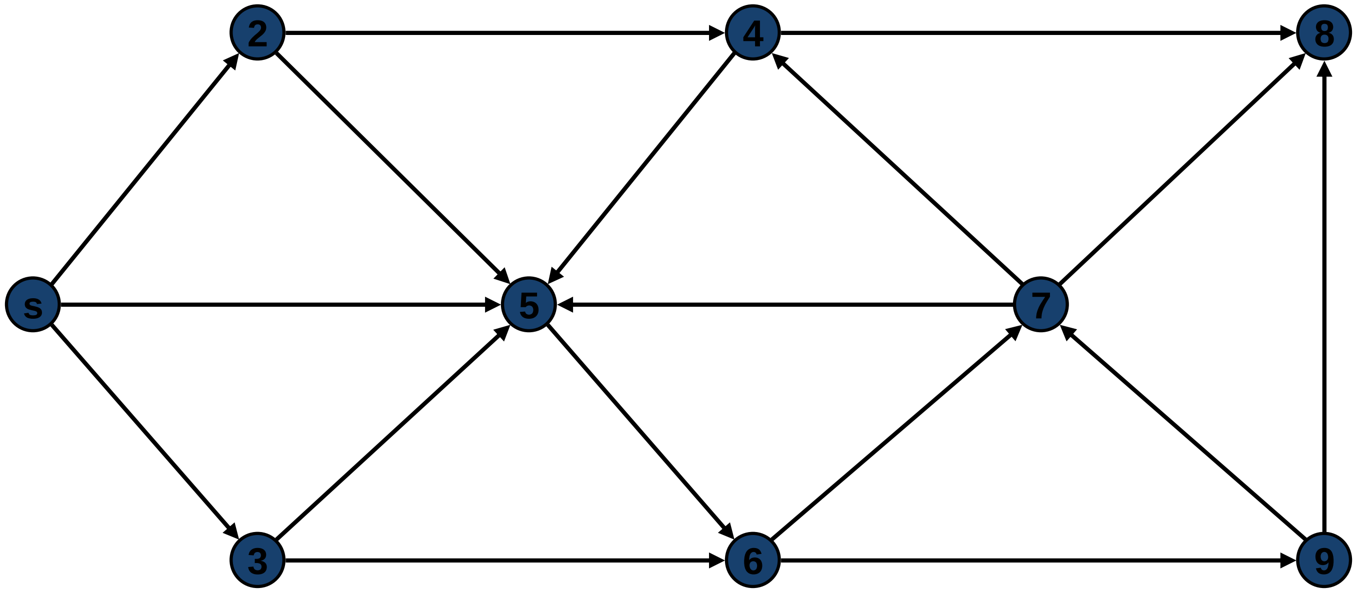
- Searching a graph:
  - Systematically follow the edges of a graph to visit the vertices of the graph.
- Used to discover the structure of a graph.
- Standard graph-searching algorithms.
  - Breadth-first Search (BFS).
  - Depth-first Search (DFS).

Breadth-first Search

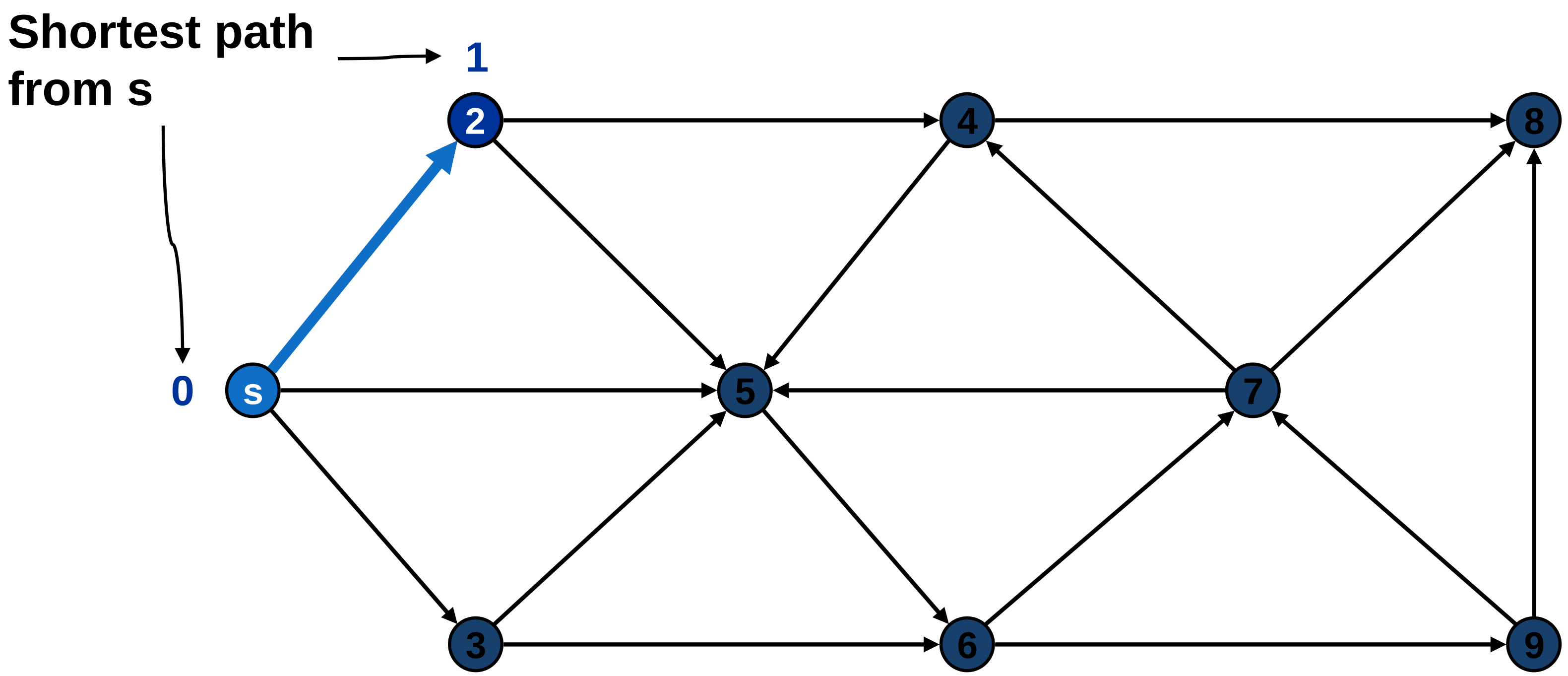
- **Input:** Graph $G = (V, E)$, either directed or undirected, and ***source vertex*** $s \in V$.
- **Output:**
  - $d[v]$ = distance (smallest # of edges, or shortest path) from s to v, for all $v \in V$. $d[v] = \infty$ if v is not reachable from s.
  - $\pi[v] = u$ such that (u, v) is last edge on shortest path $s \rightsquigarrow v$.
    - u is v's predecessor.
  - Builds breadth-first tree with root s that contains all reachable vertices.

Definitions:
Path between vertices u and v: Sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $u=v_1$ and $v=v_k$, and $(v_i,v_{i+1}) \in E$, for all $1 \le i \le k-1$.
Length of the path: Number of edges in the path.

Breadth First Search



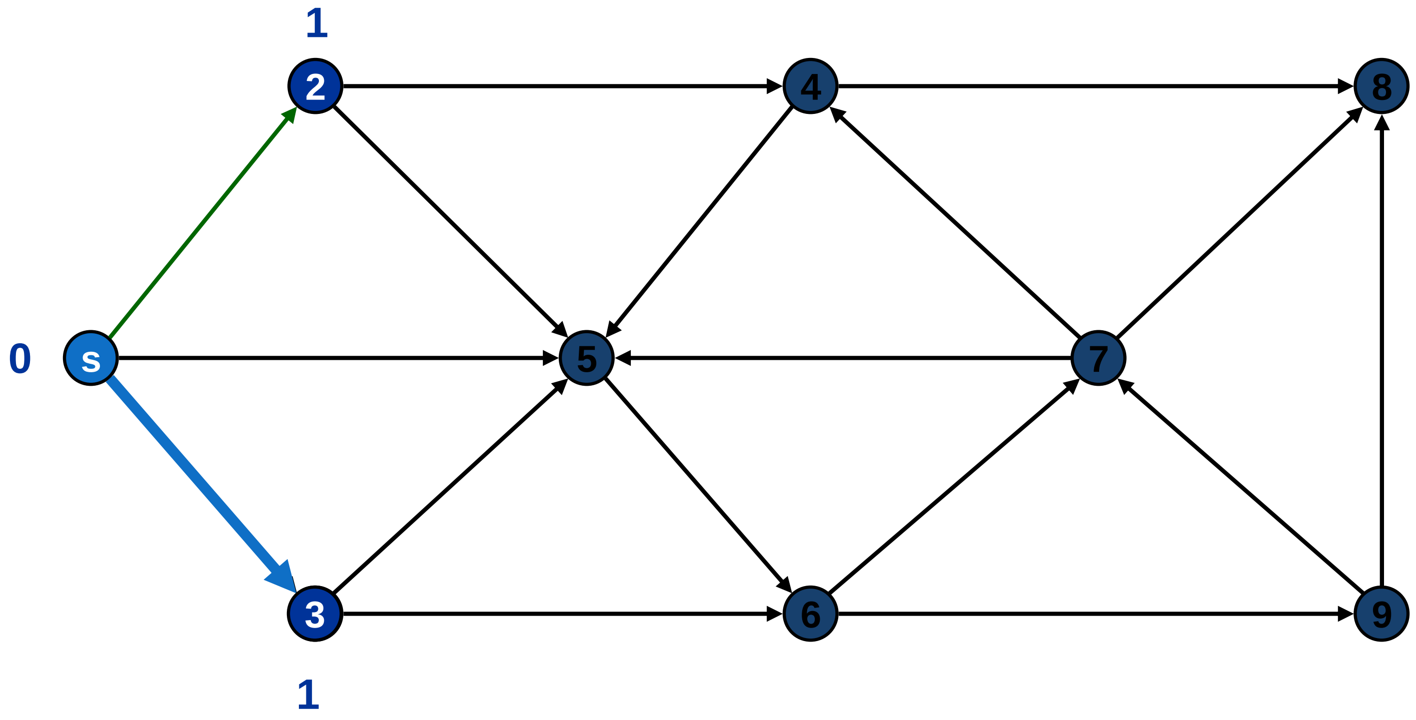
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: s

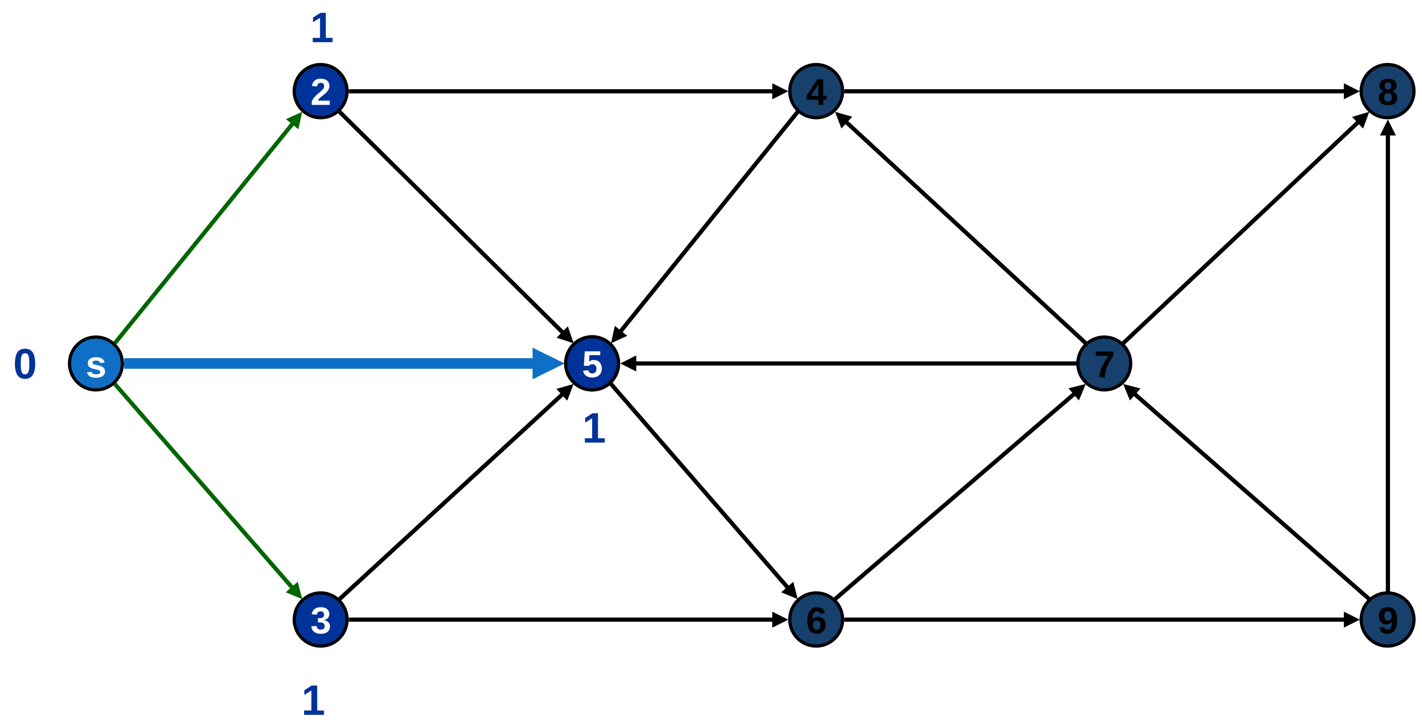
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: s 2

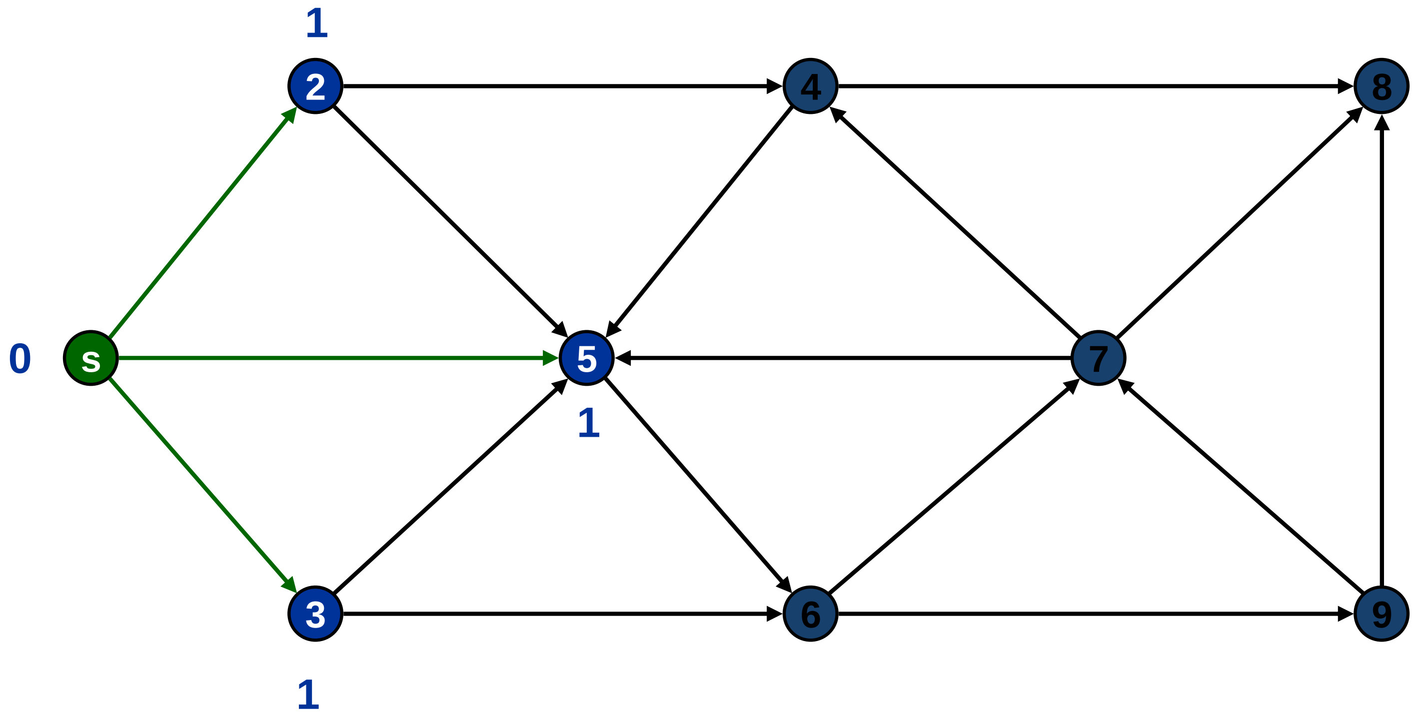
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: s 2 3

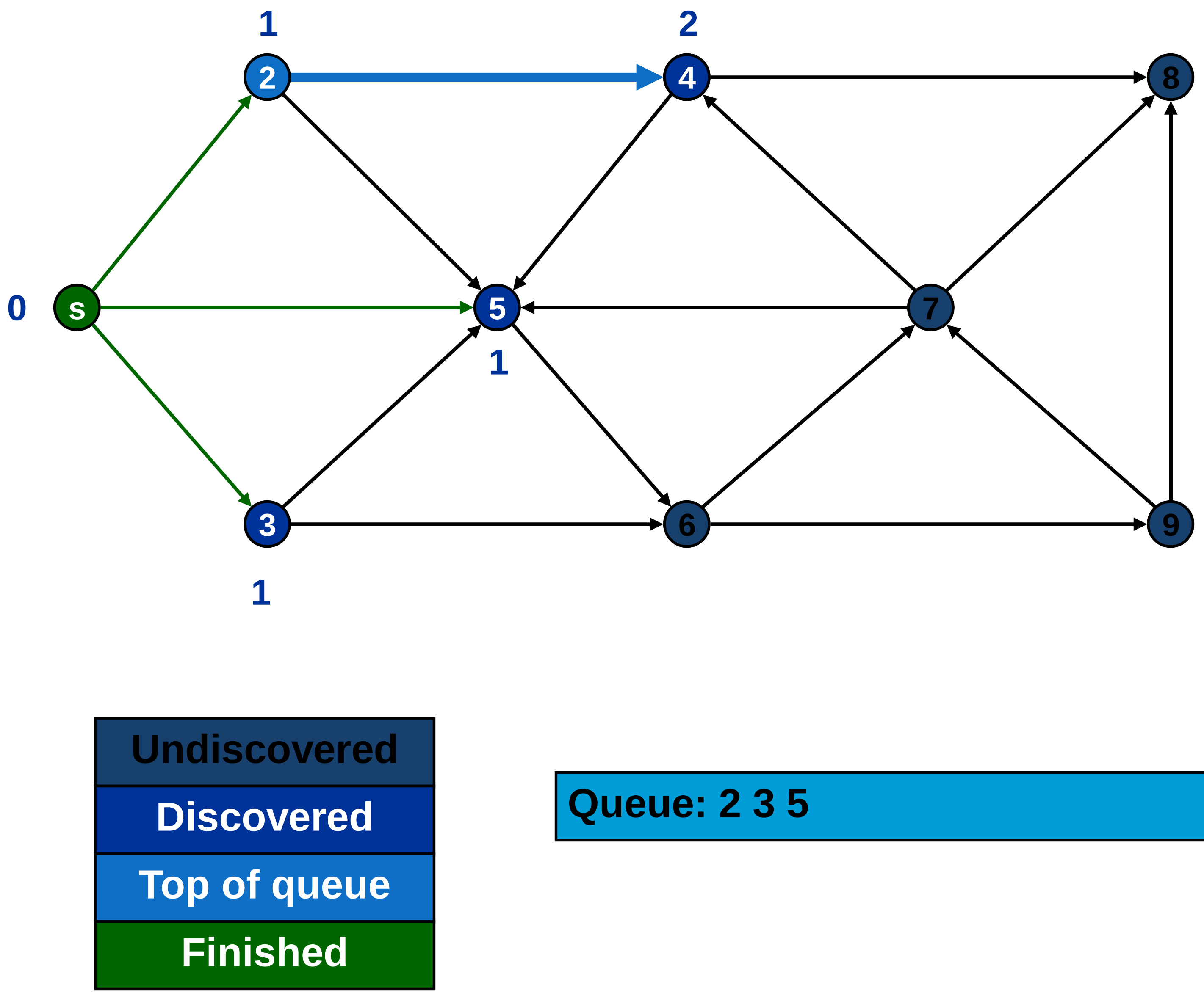
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

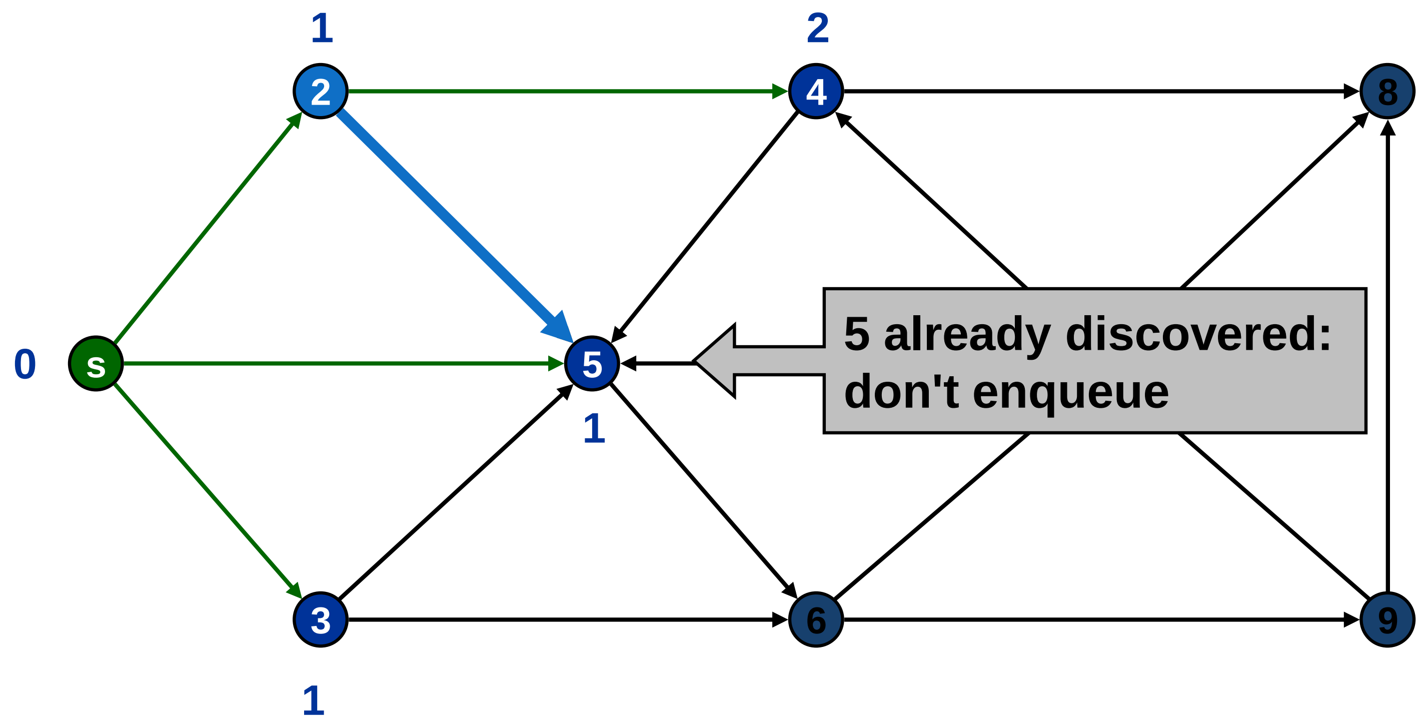
Queue: 2 3 5

Breadth First Search



Queue: 2 3 5

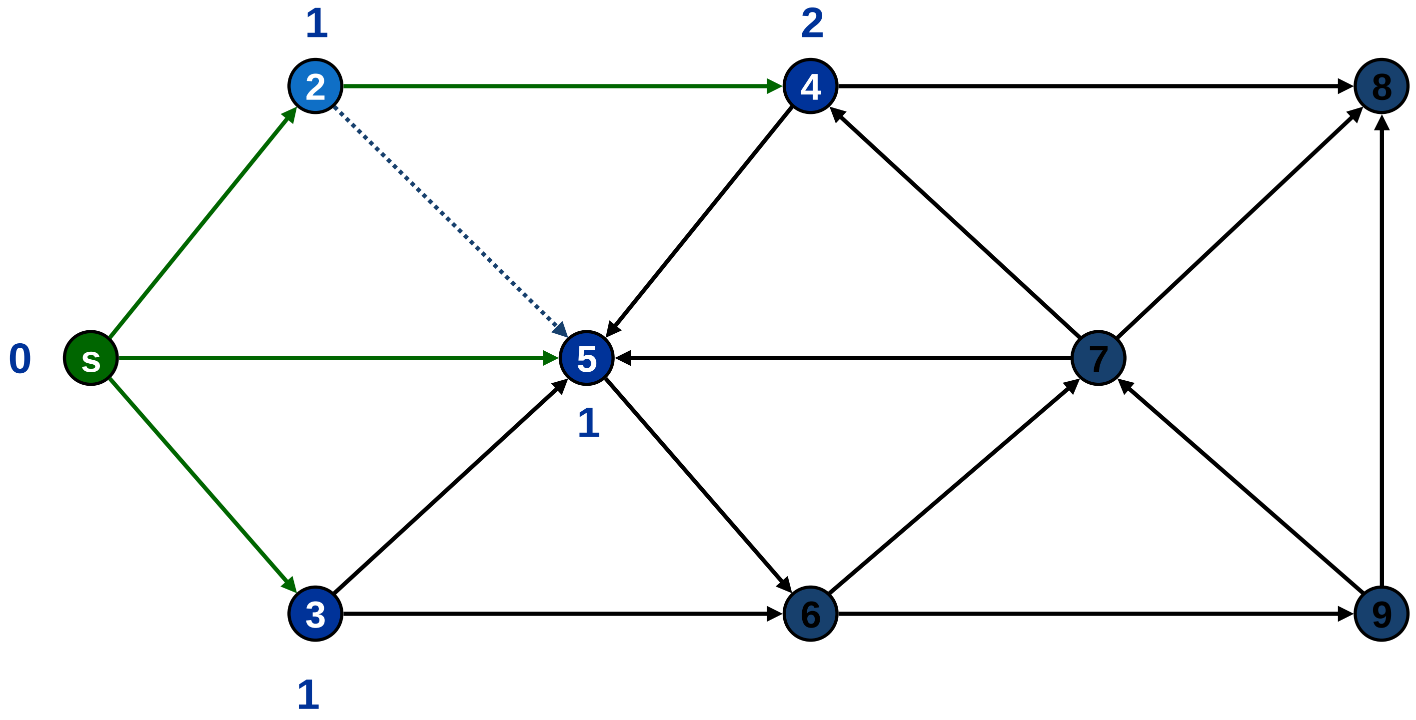
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 2 3 5 4

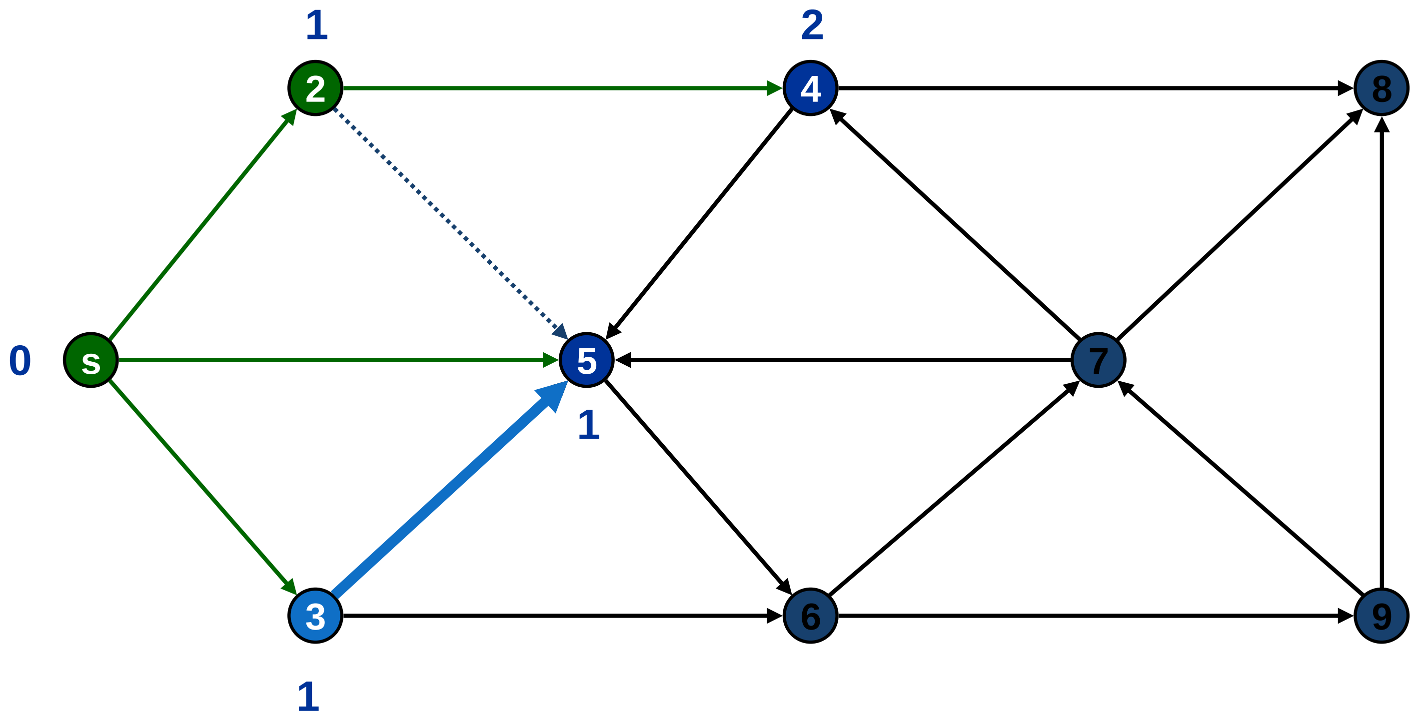
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 2 3 5 4

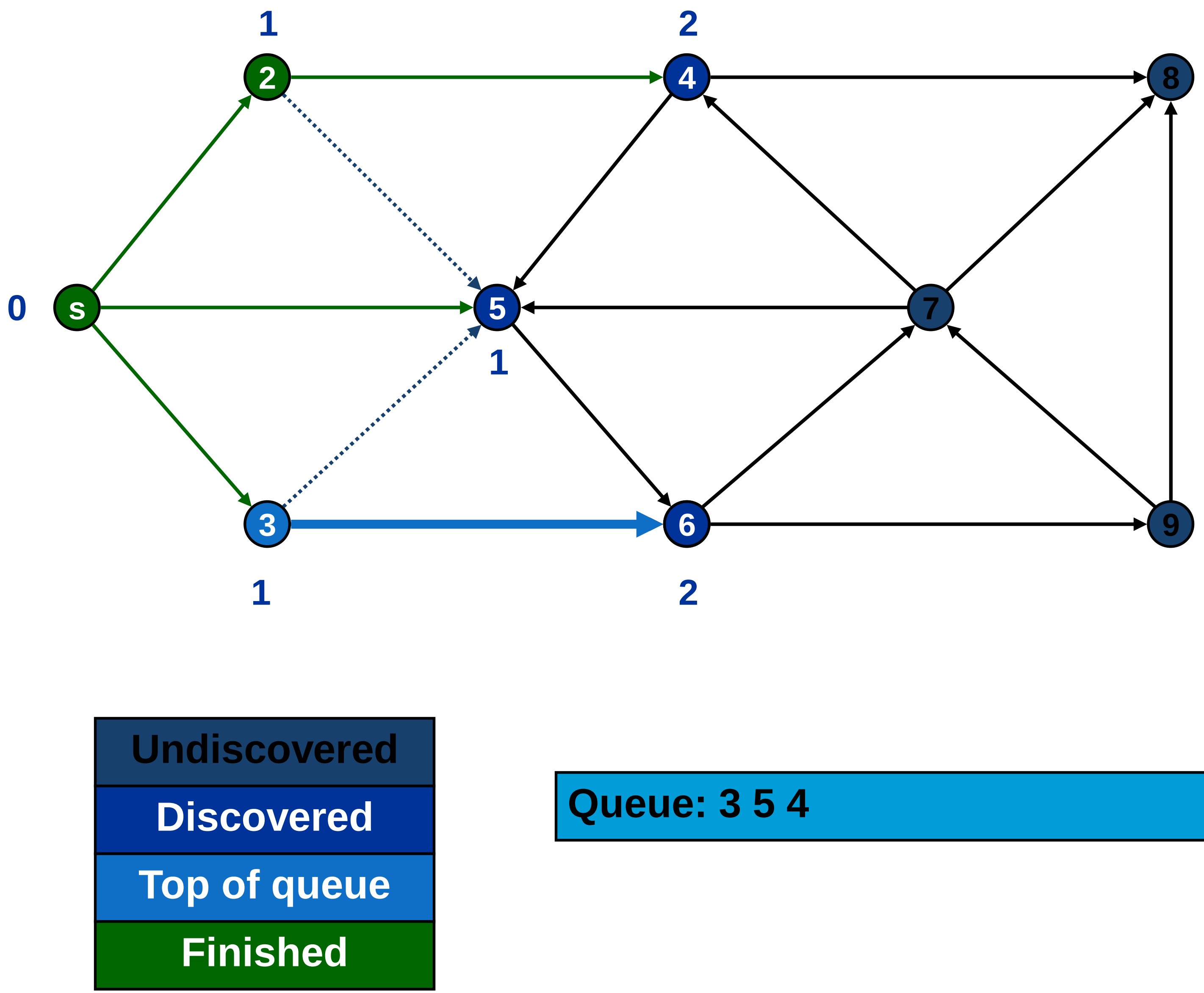
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

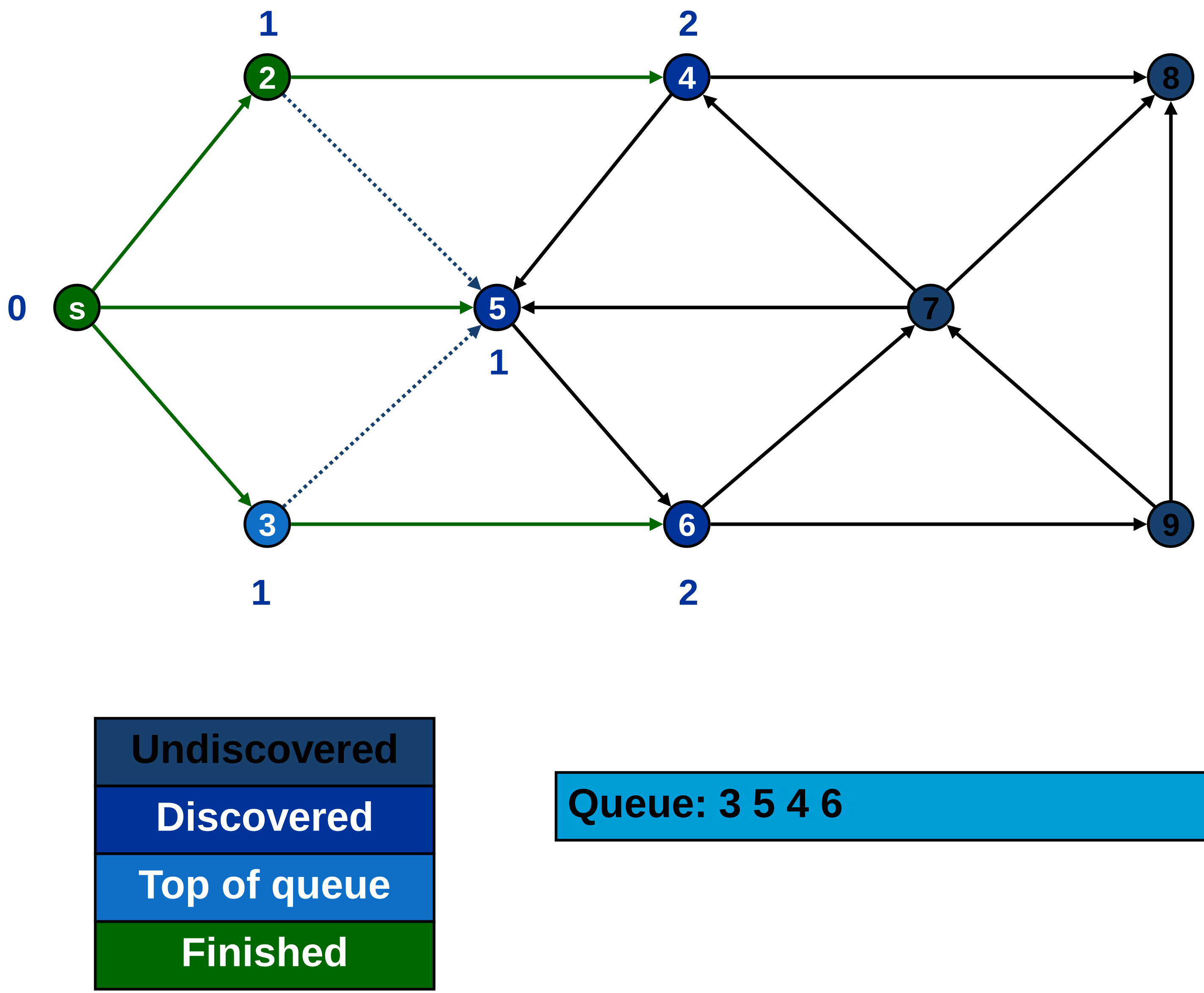
Queue: 3 5 4

Breadth First Search



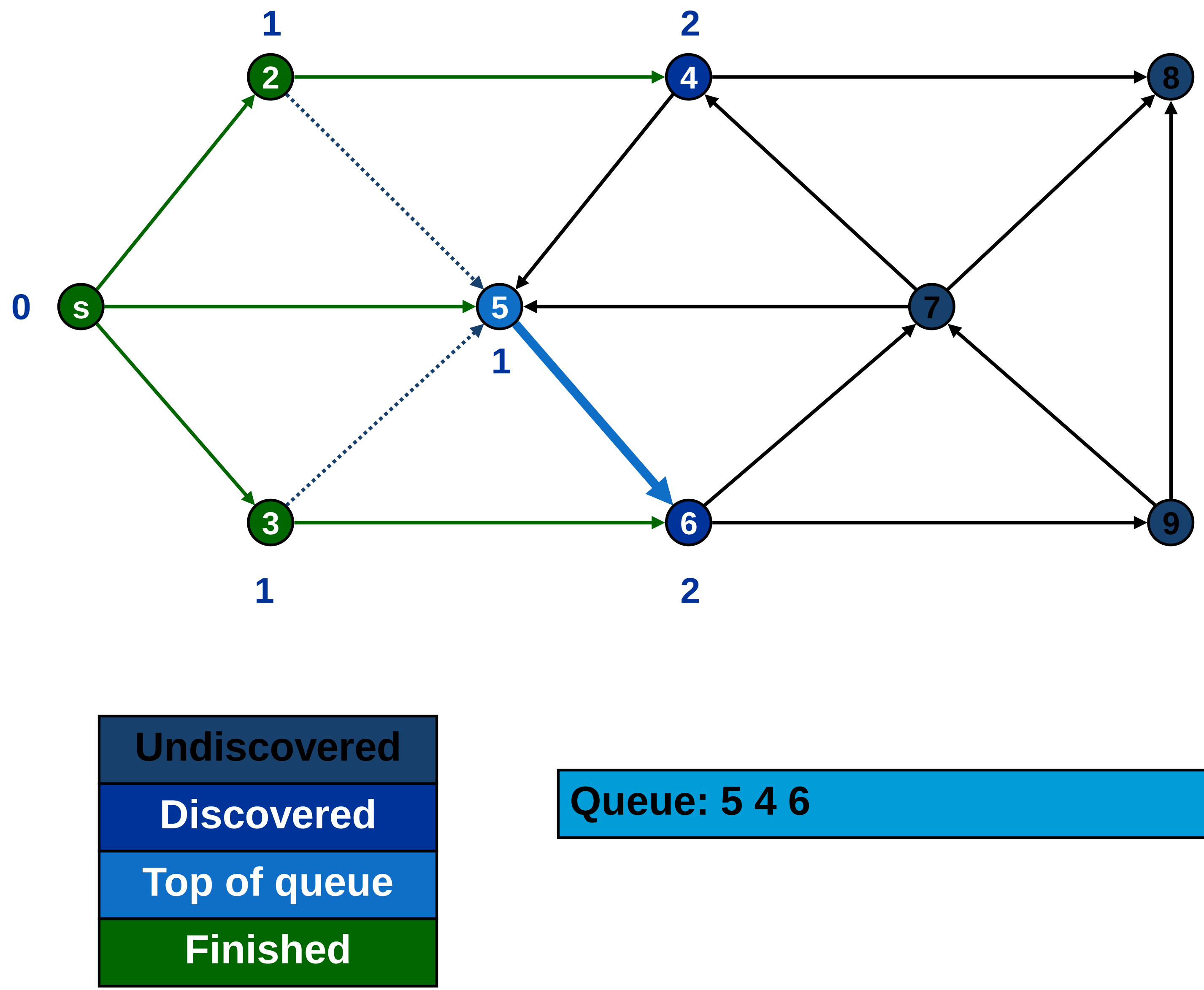
Queue: 3 5 4

Breadth First Search



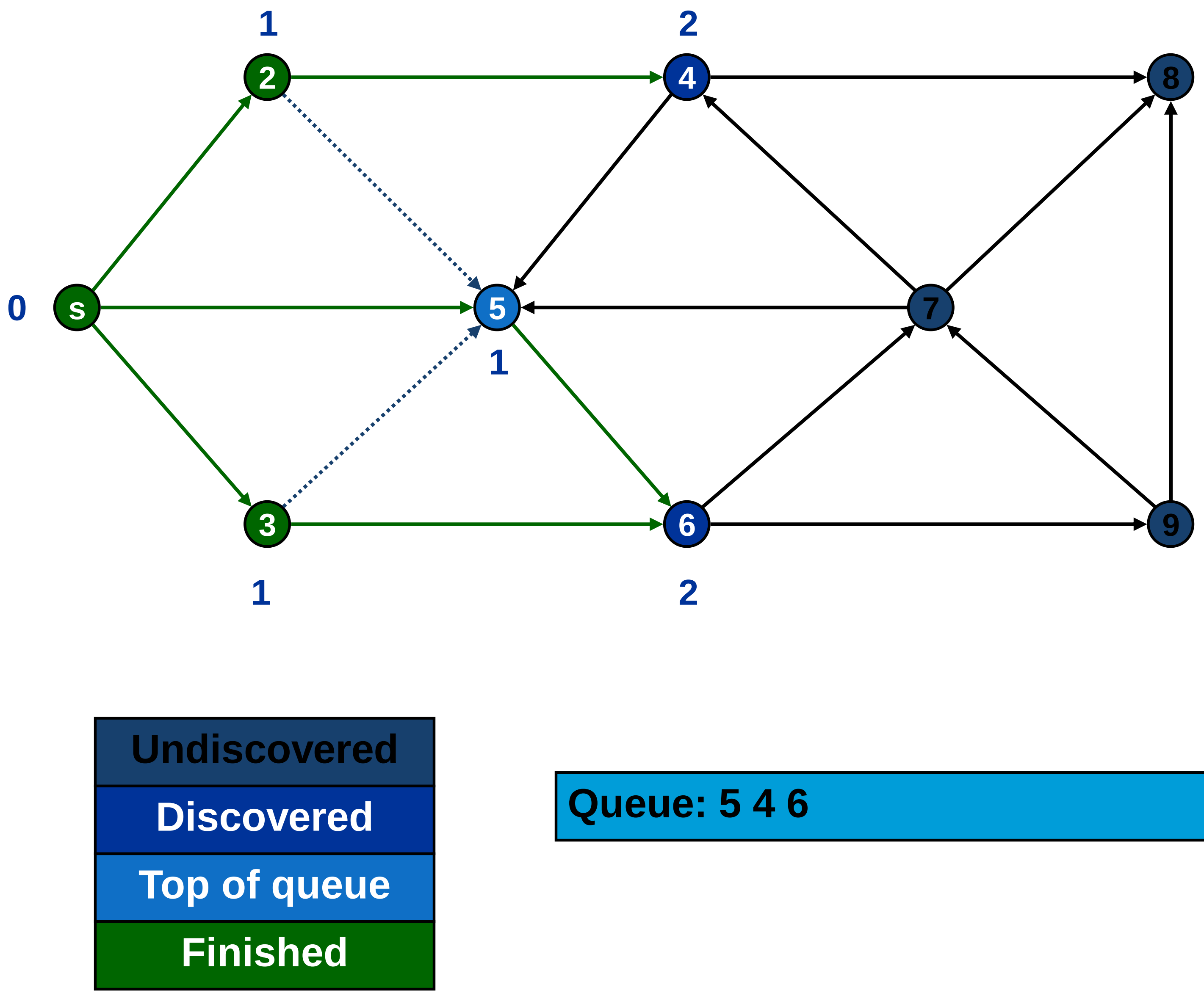
Queue: 3 5 4 6

Breadth First Search



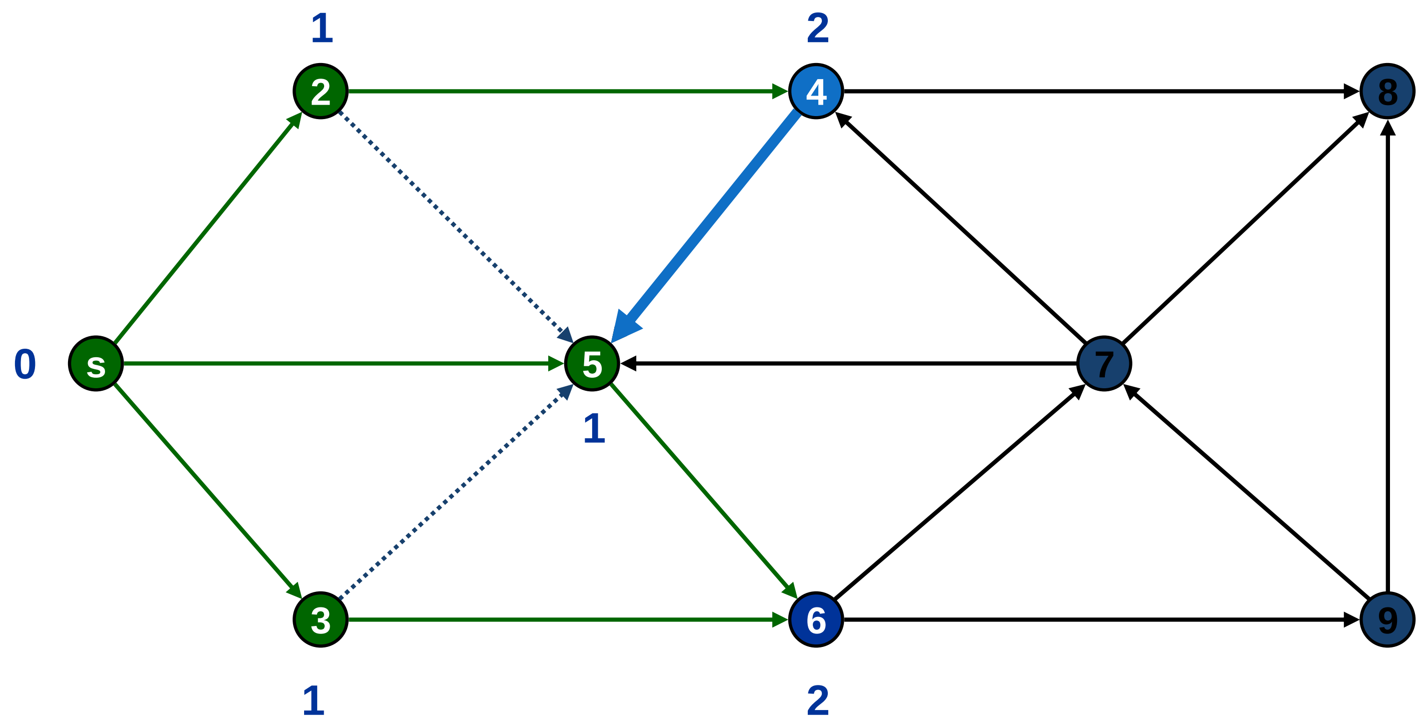
Queue: 5 4 6

Breadth First Search



Queue: 5 4 6

Breadth First Search



Undiscovered

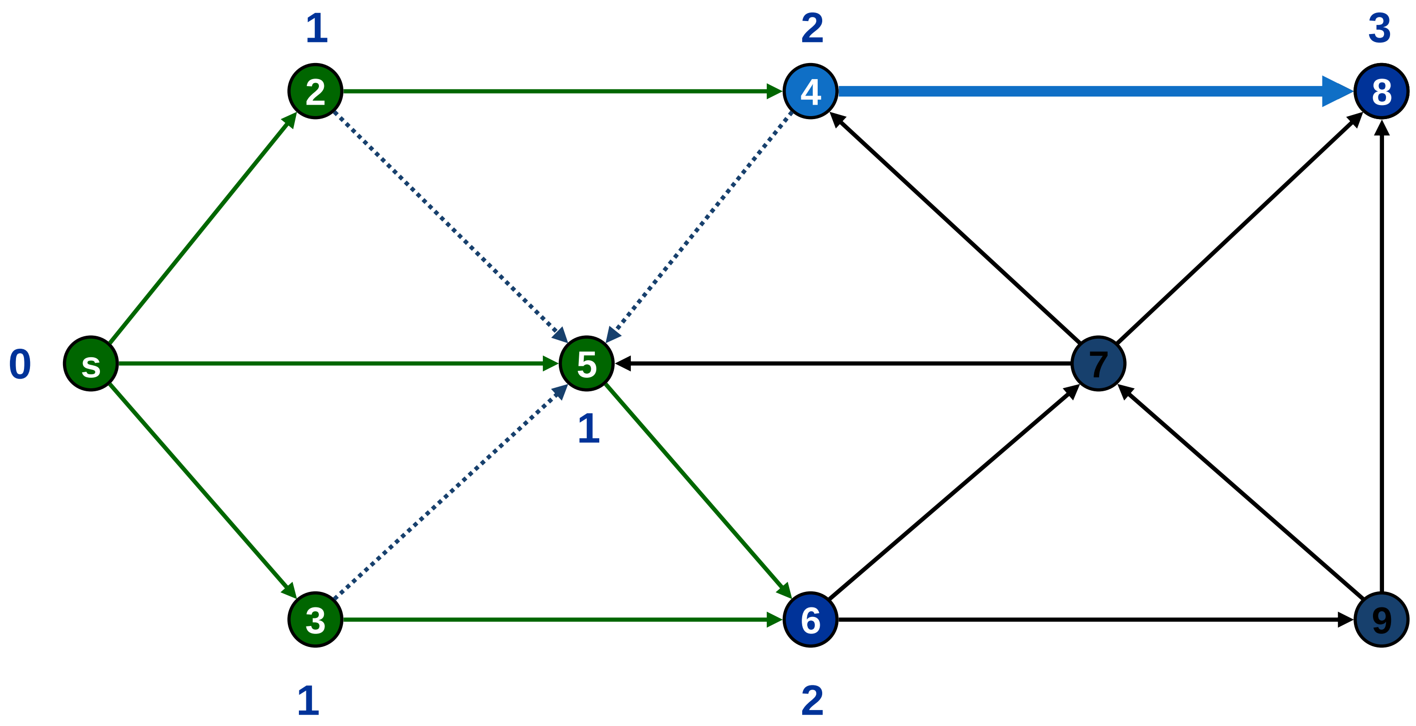
Discovered

Top of queue

Finished

Queue: 4 6

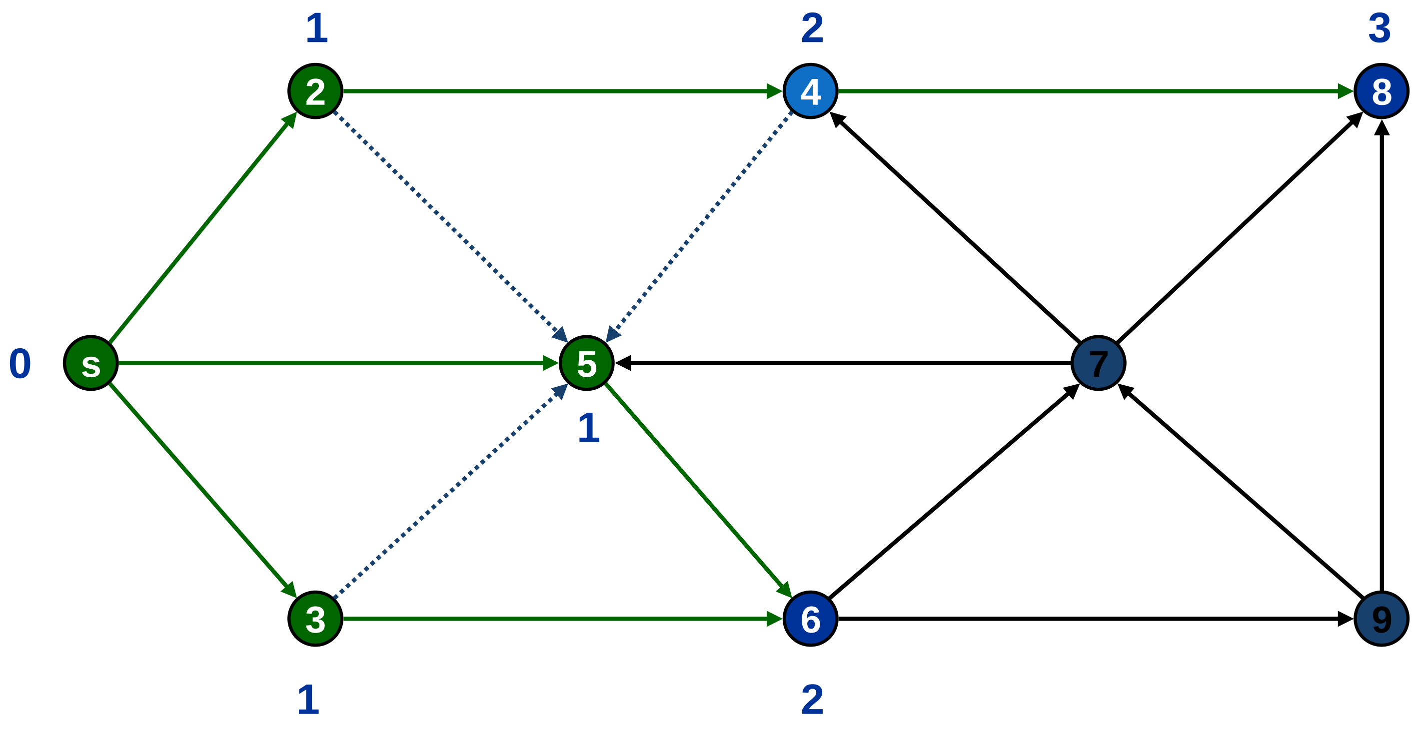
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 4 6

Breadth First Search



Undiscovered

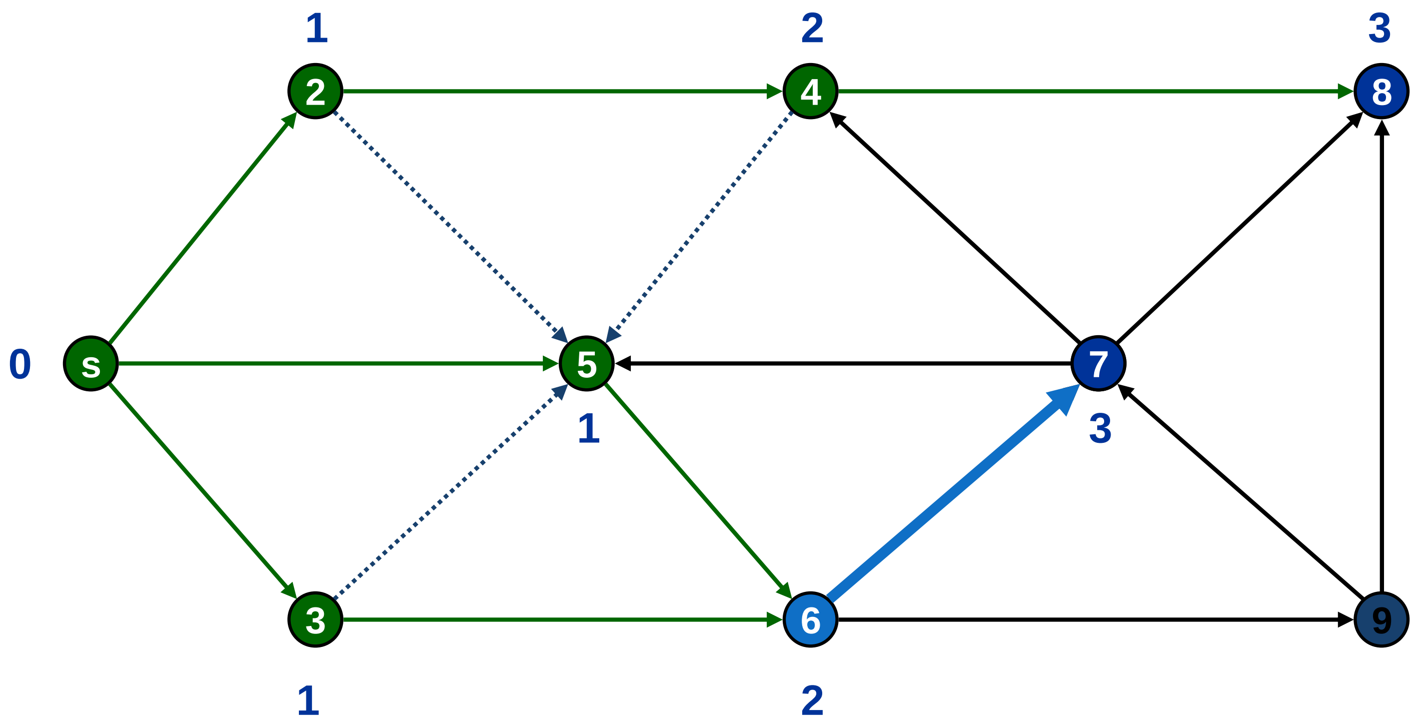
Discovered

Top of queue

Finished

Queue: 4 6 8

Breadth First Search



Undiscovered

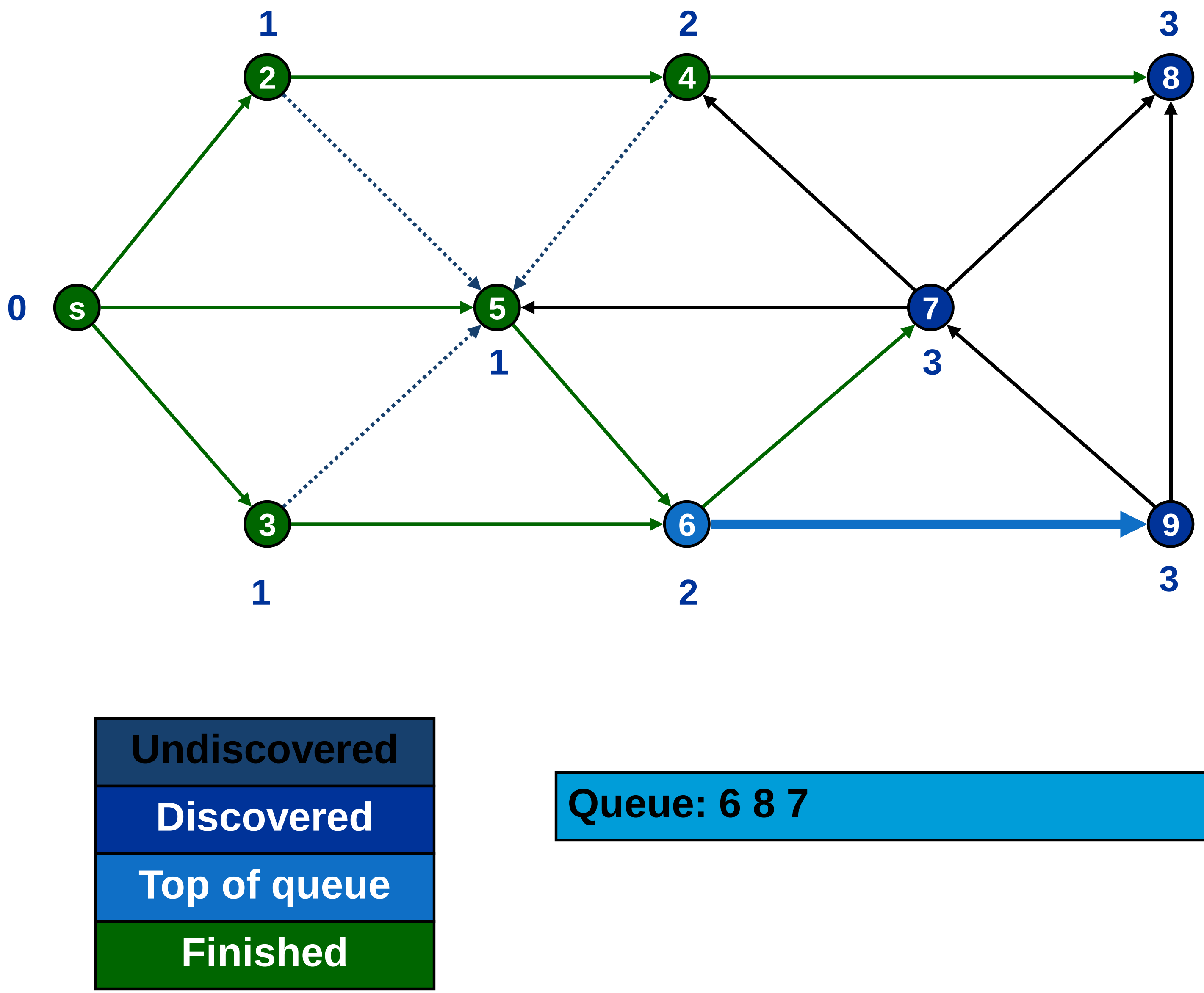
Discovered

Top of queue

Finished

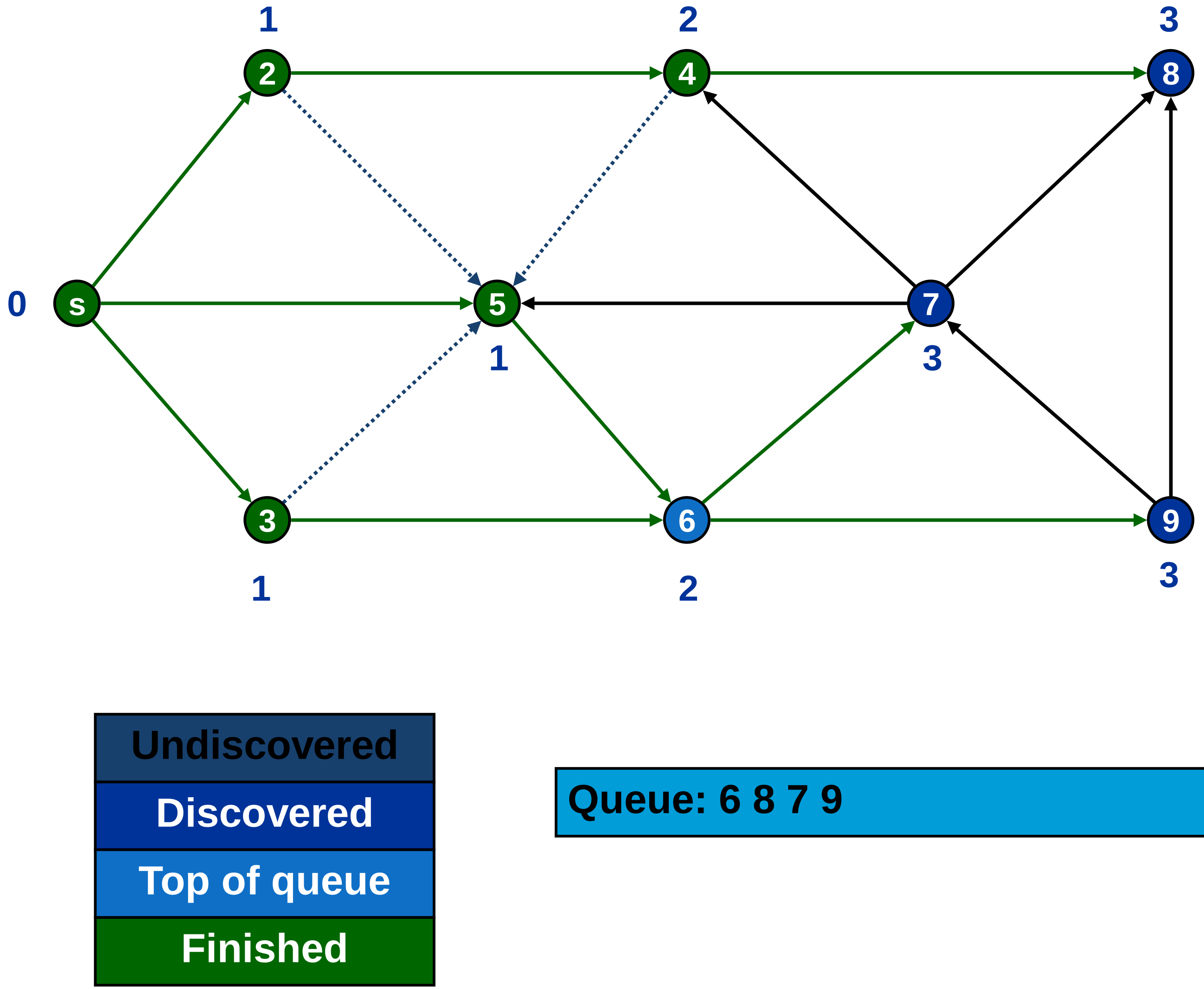
Queue: 6 8

Breadth First Search



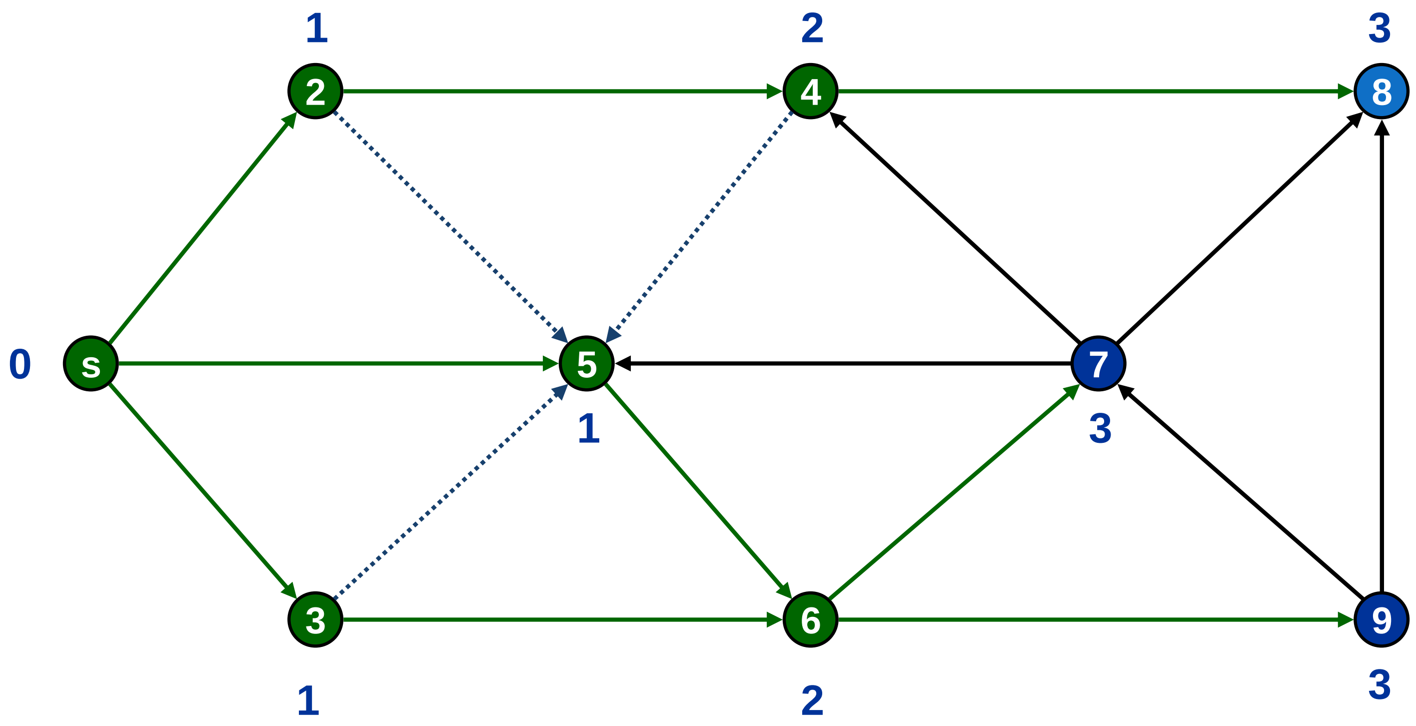
Queue: 6 8 7

Breadth First Search



Queue: 6 8 7 9

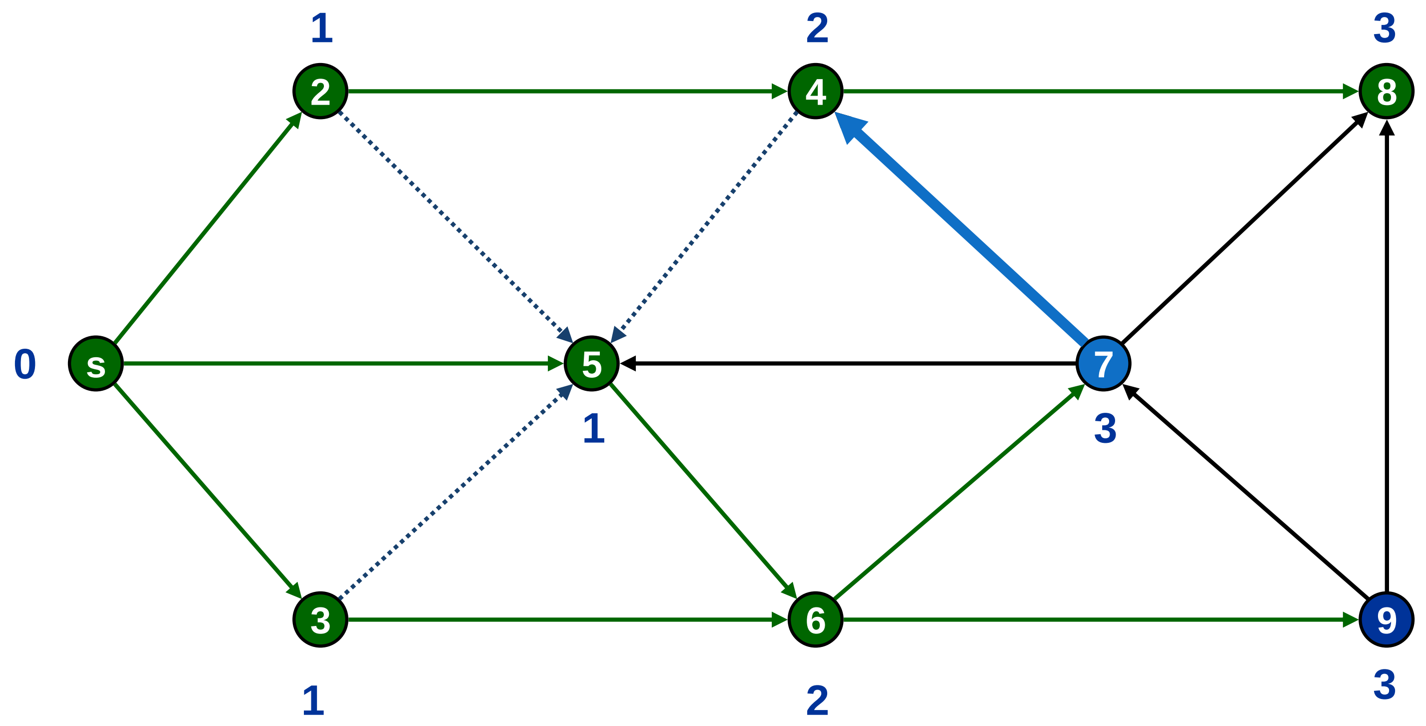
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 8 7 9

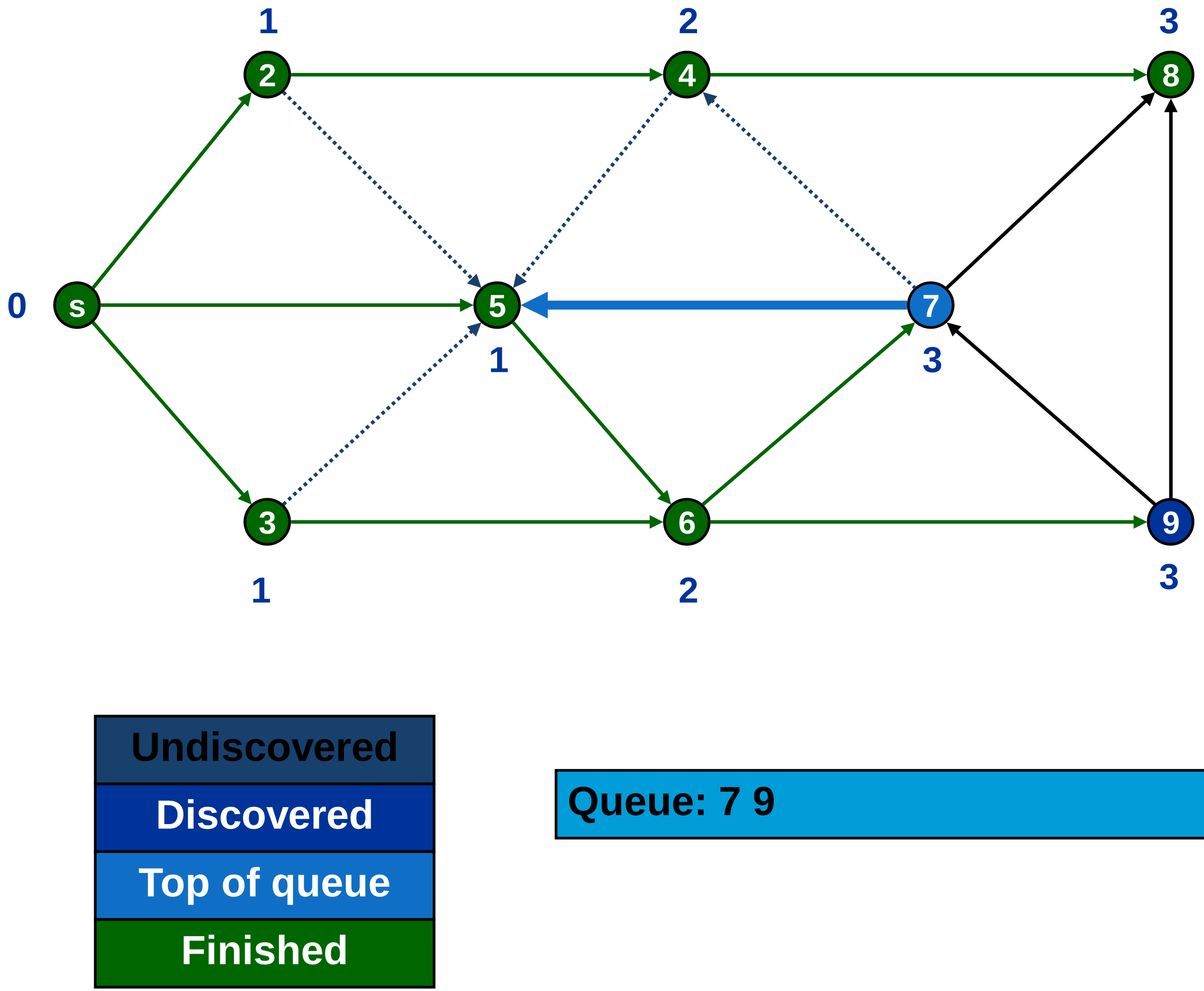
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

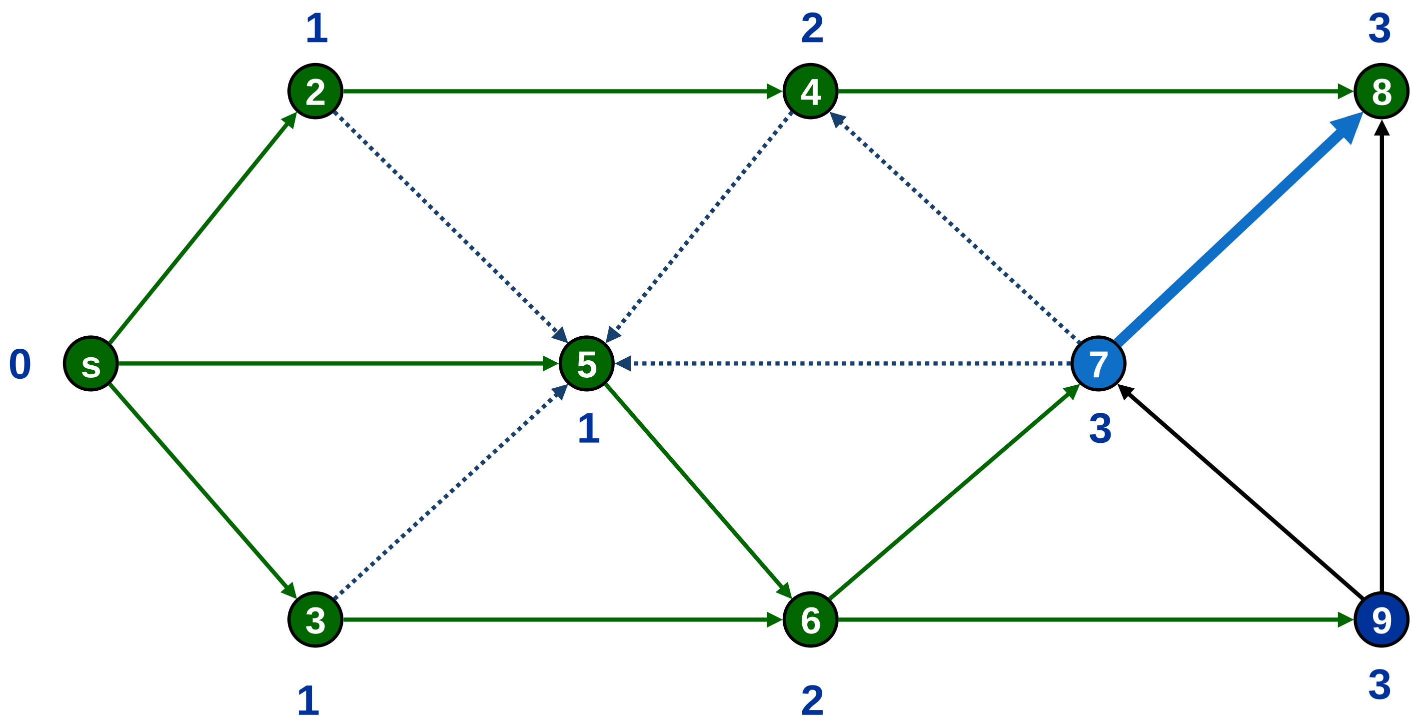
Queue: 7 9

Breadth First Search



Queue: 7 9

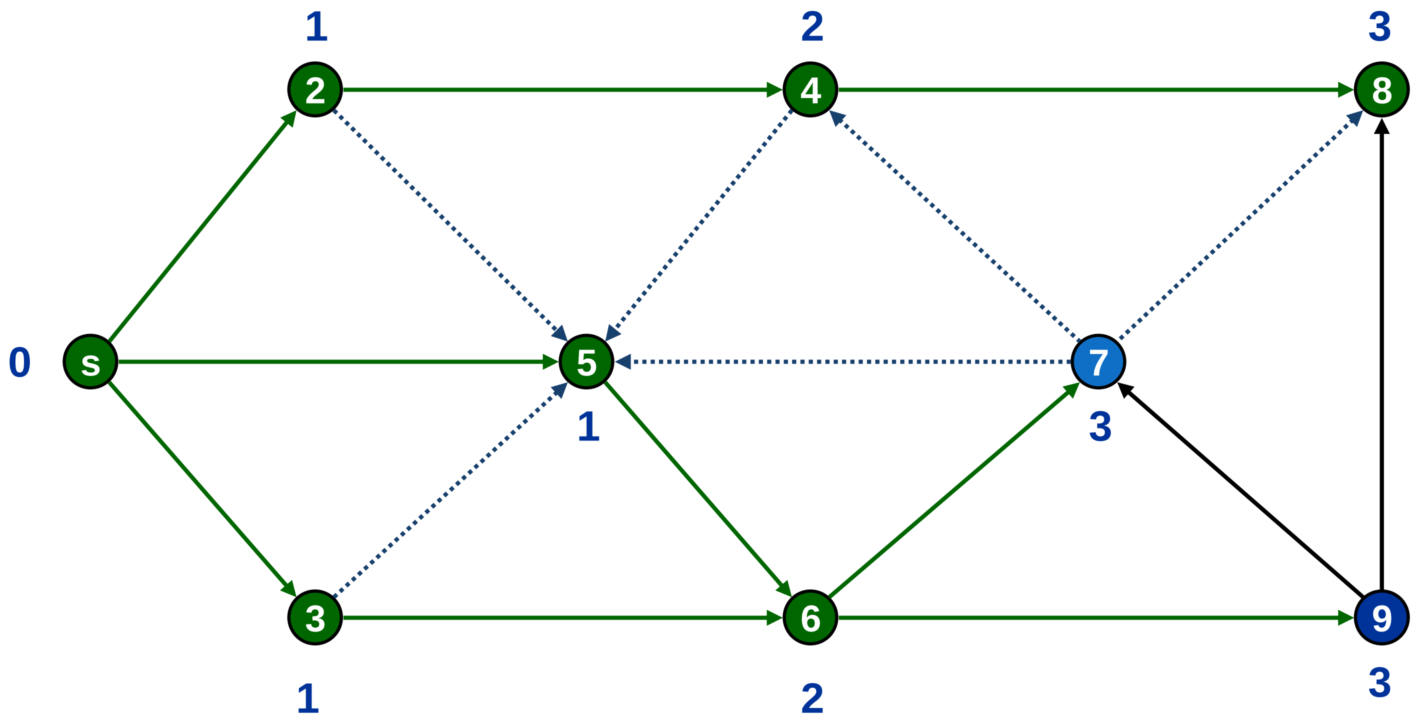
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 7 9

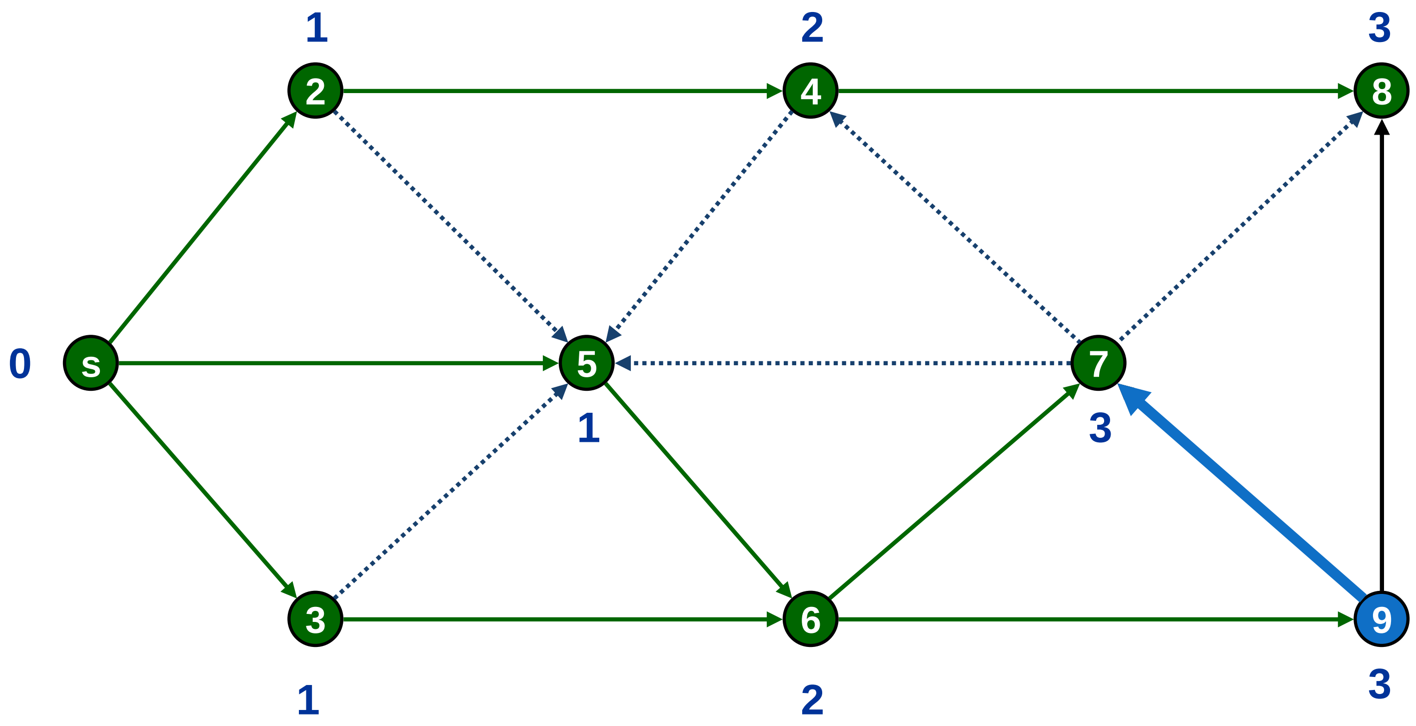
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 7 9

Breadth First Search



Undiscovered

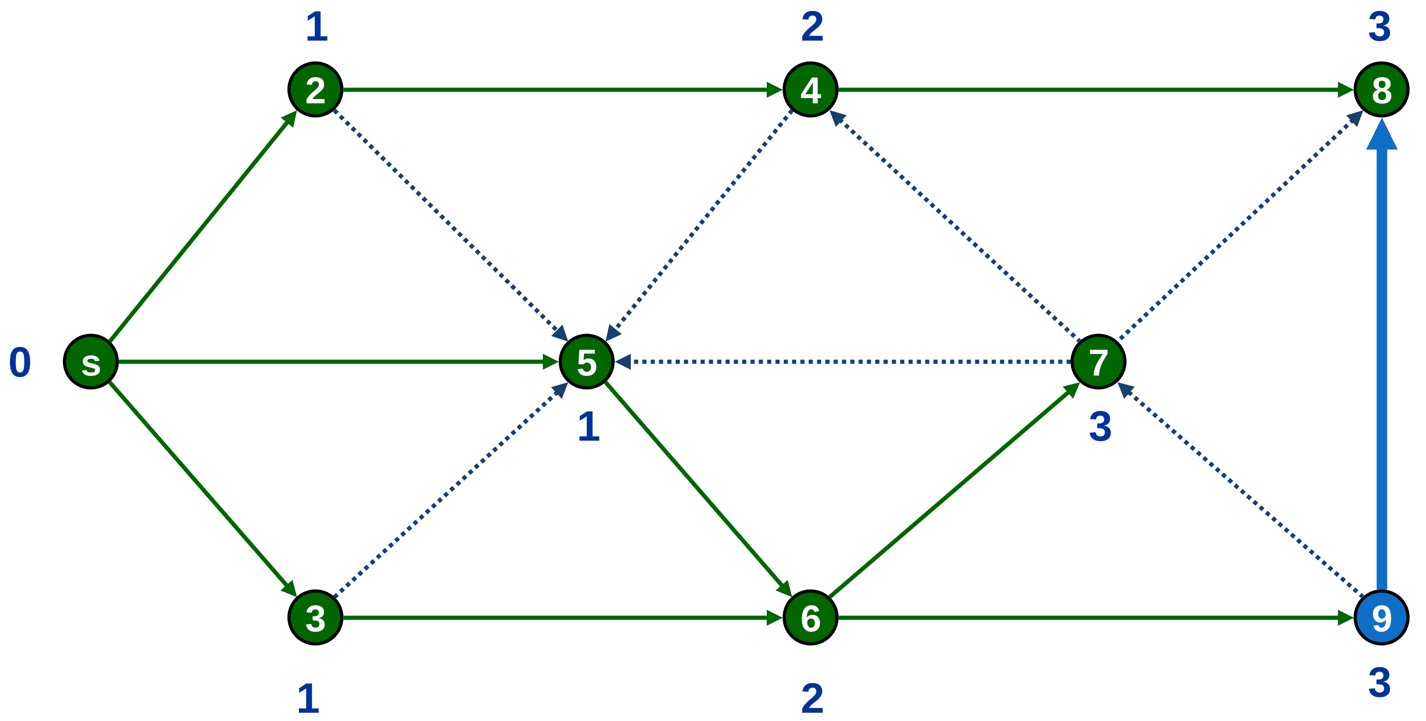
Discovered

Top of queue

Finished

Queue: 9

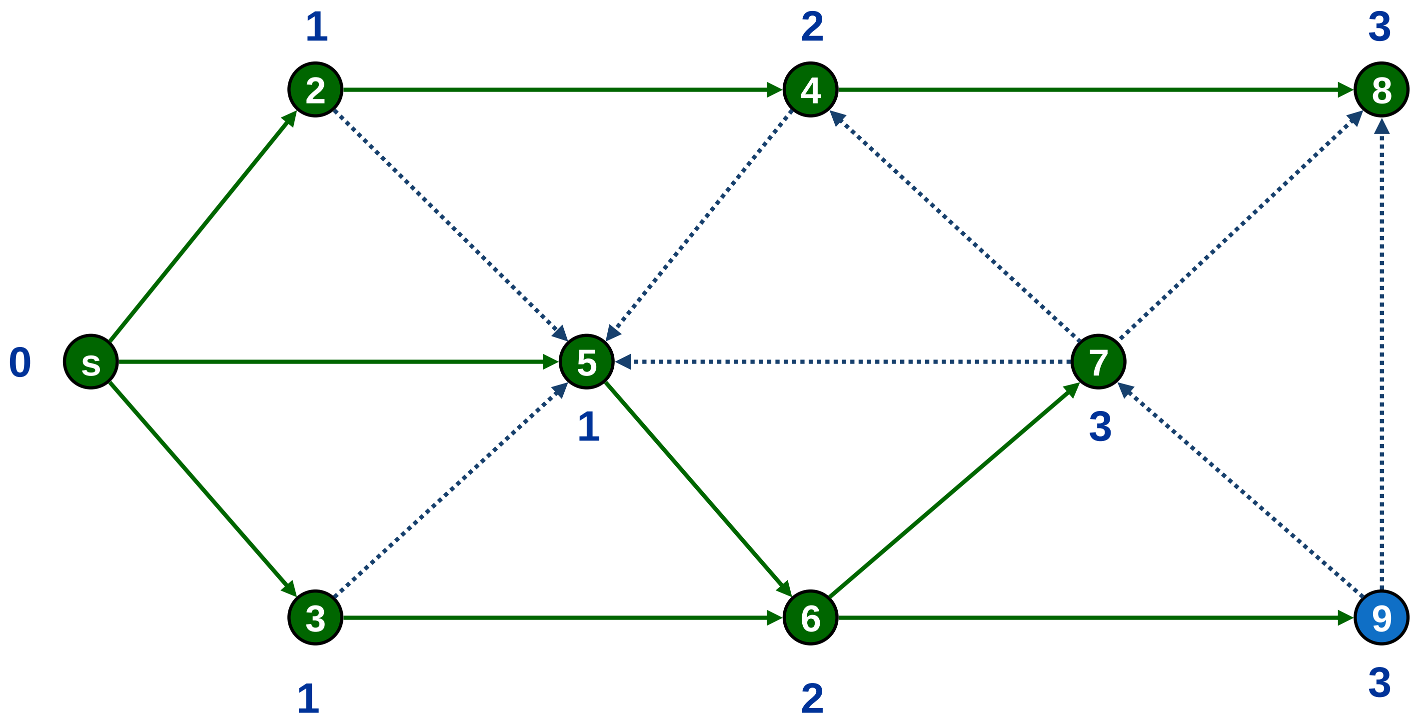
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 9

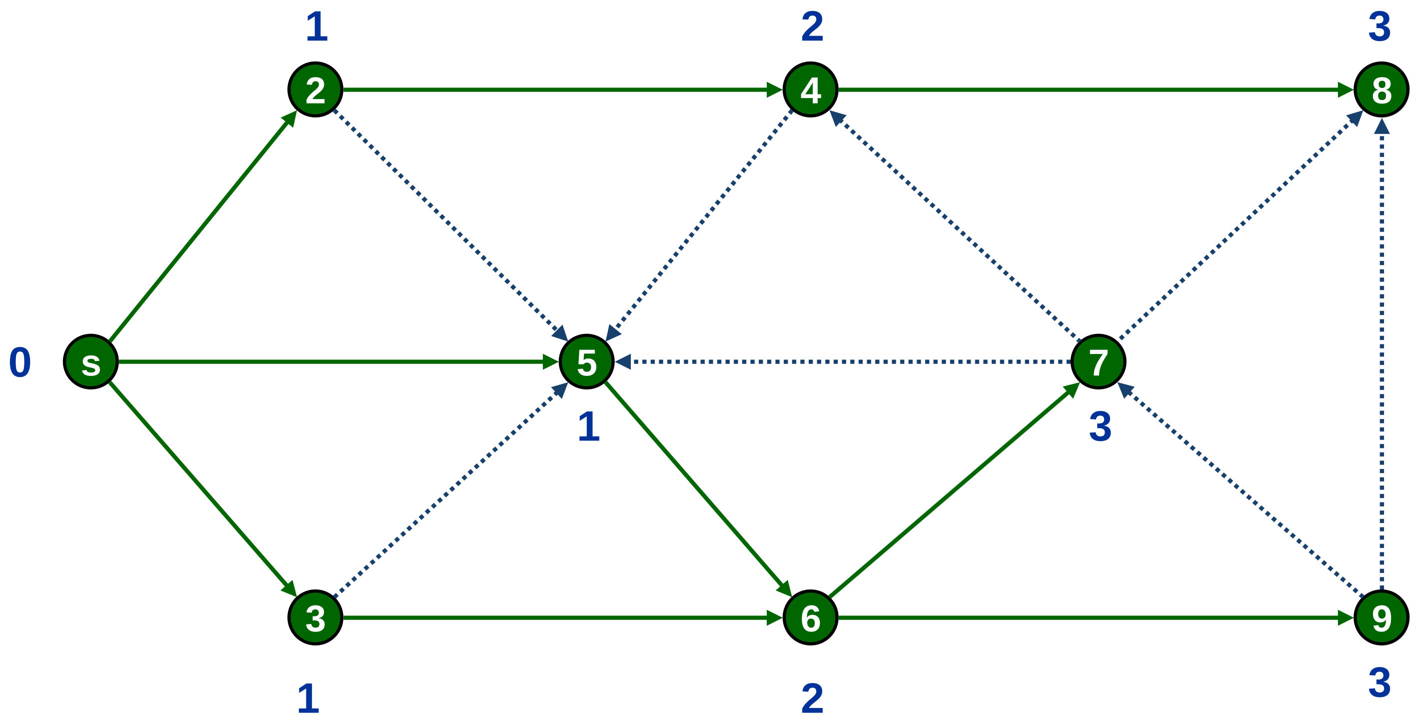
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue: 9

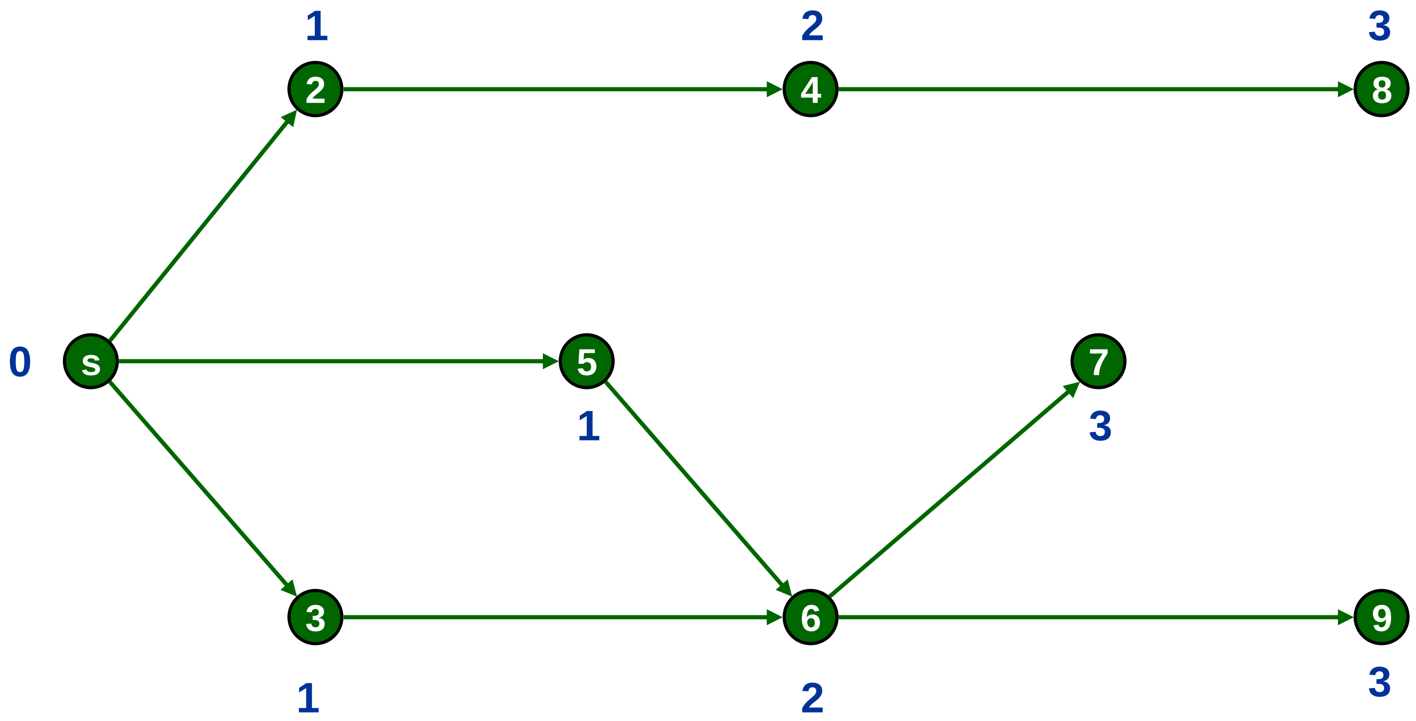
Breadth First Search



Undiscovered
Discovered
Top of queue
Finished

Queue:

Breadth First Search



Level Graph

Analysis of BFS

- Initialization takes $O(V)$.
- Traversal Loop
  - After initialization, each vertex is enqueued and dequeued at most once, and each operation takes $O(1)$. So, total time for queuing is $O(V)$.
  - The adjacency list of each vertex is scanned at most once. The sum of lengths of all adjacency lists is $\Theta(E)$.
- Summing up over all vertices => total running time of BFS is $O(V+E)$, linear in the size of the adjacency list representation of graph.

Depth first search

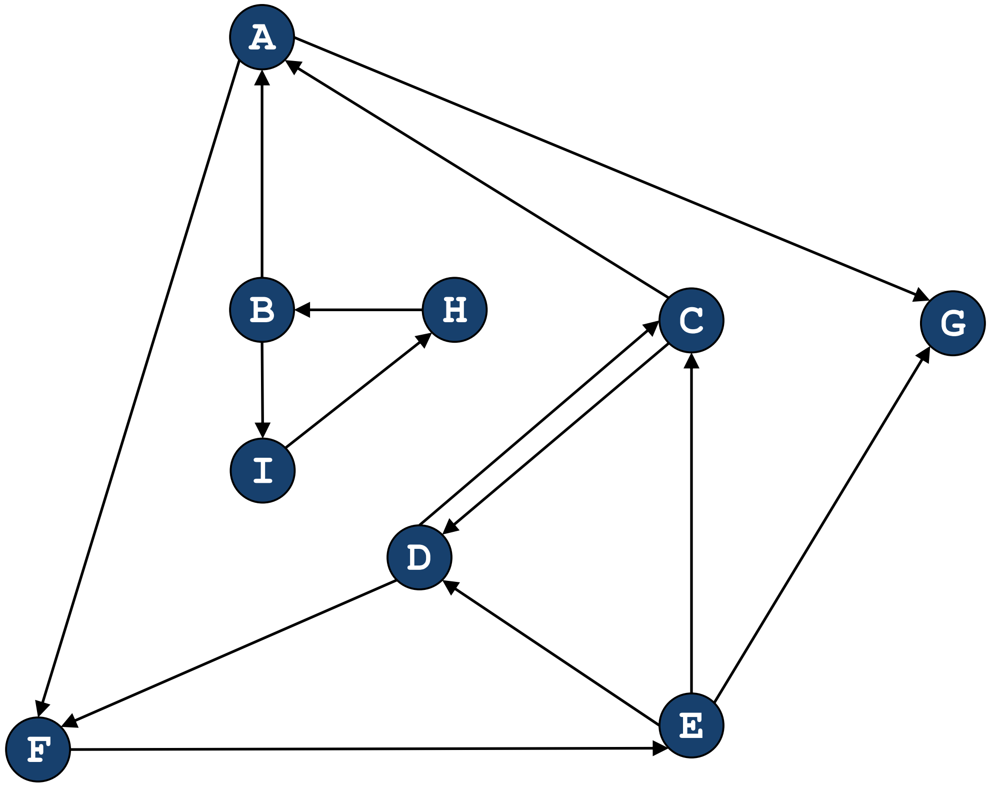
Depth-first Search (DFS)

- Explore edges out of the most recently discovered vertex *v*.
- When all edges of *v* have been explored, backtrack to explore other edges leaving the vertex from which *v* was discovered (its *predecessor*).
- "Search as deep as possible first."
- Continue until all vertices reachable from the original source are discovered.
- If any undiscovered vertices remain, then one of them is chosen as a new source and search is repeated from that source.

Depth-first Search

- **Input:** $G = (V, E)$, directed or undirected. No source vertex given!
- **Output:**
  - 2 **timestamps** on each vertex. Integers between 1 and 2|V|.
    - $d[v]$ = ***discovery time*** (v turns from white to gray)
    - $f[v]$ = ***finishing time*** (v turns from gray to black)
  - $\pi[v]$: predecessor of $v = u$, such that v was discovered during the scan of u's adjacency list.
- Uses the same coloring scheme for vertices as BFS.

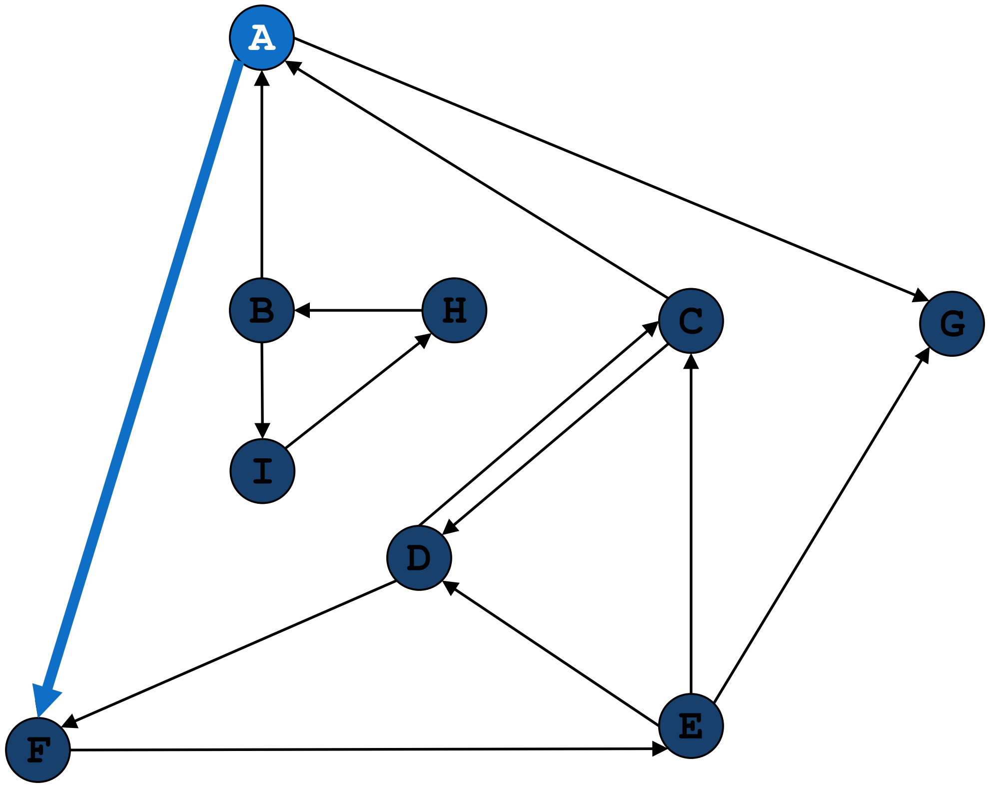
Directed Depth First Search



Adjacency Lists

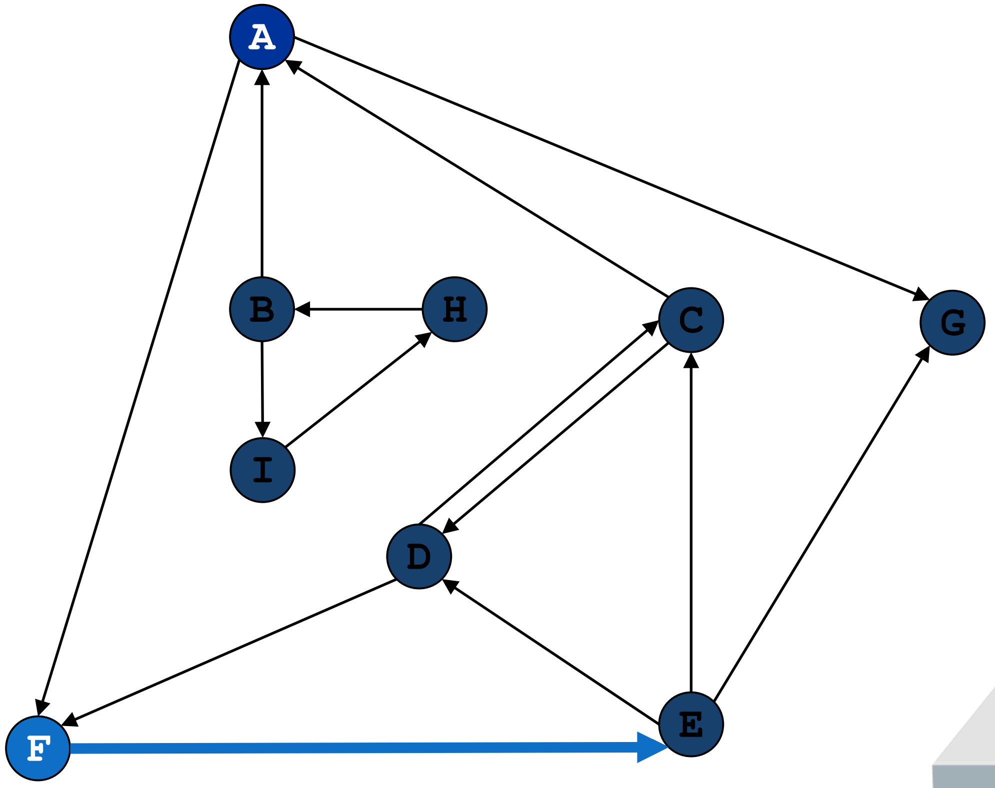
A: F G
B: A I
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

Directed Depth First Search



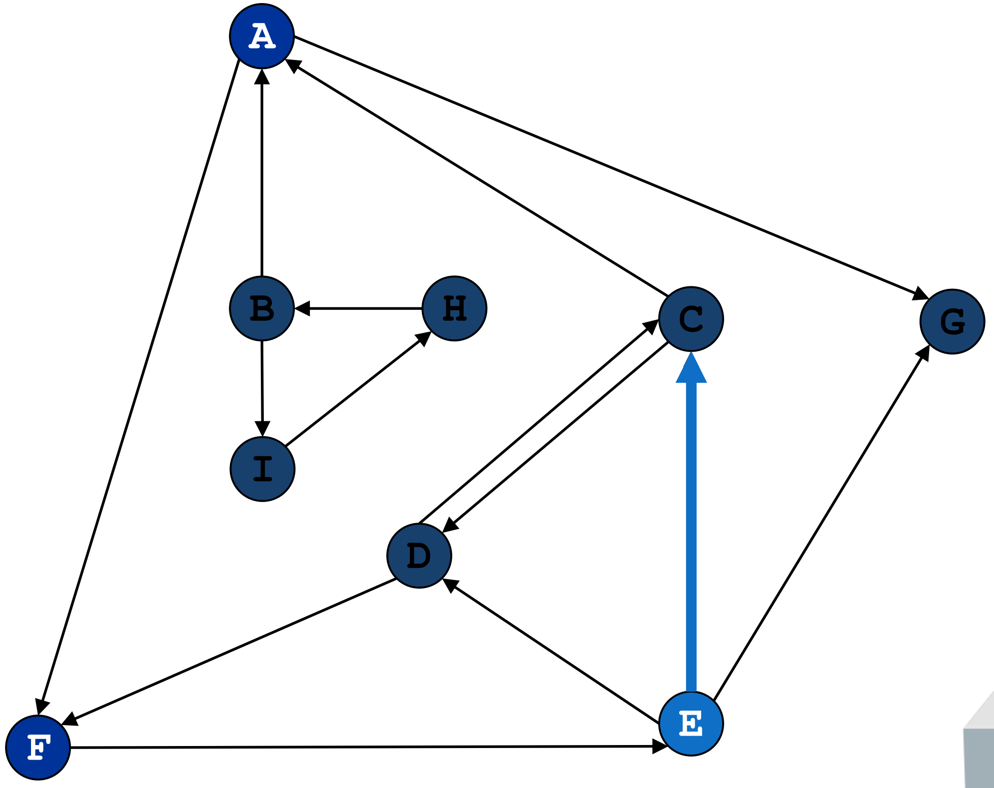
Function call stack:

Directed Depth First Search

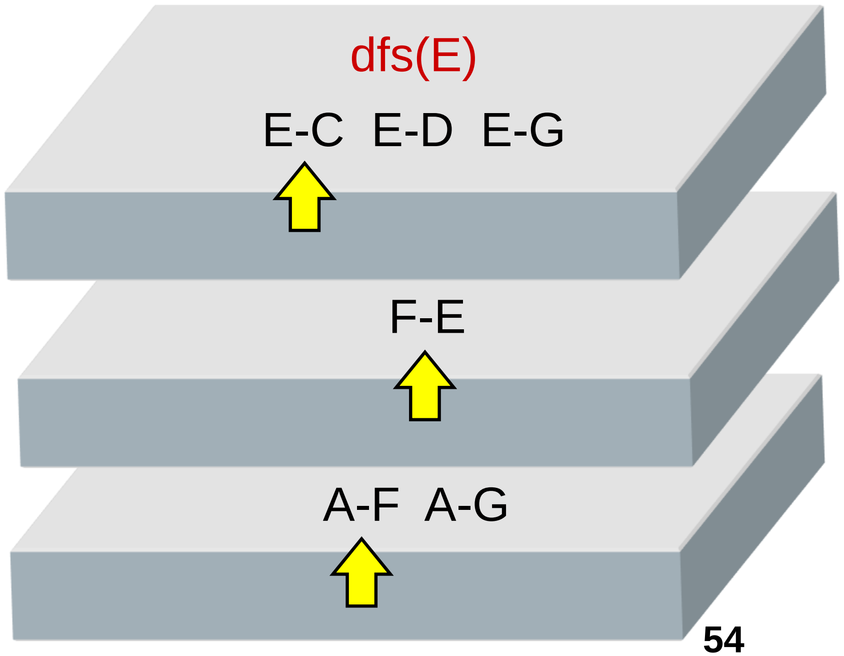


Function call stack:

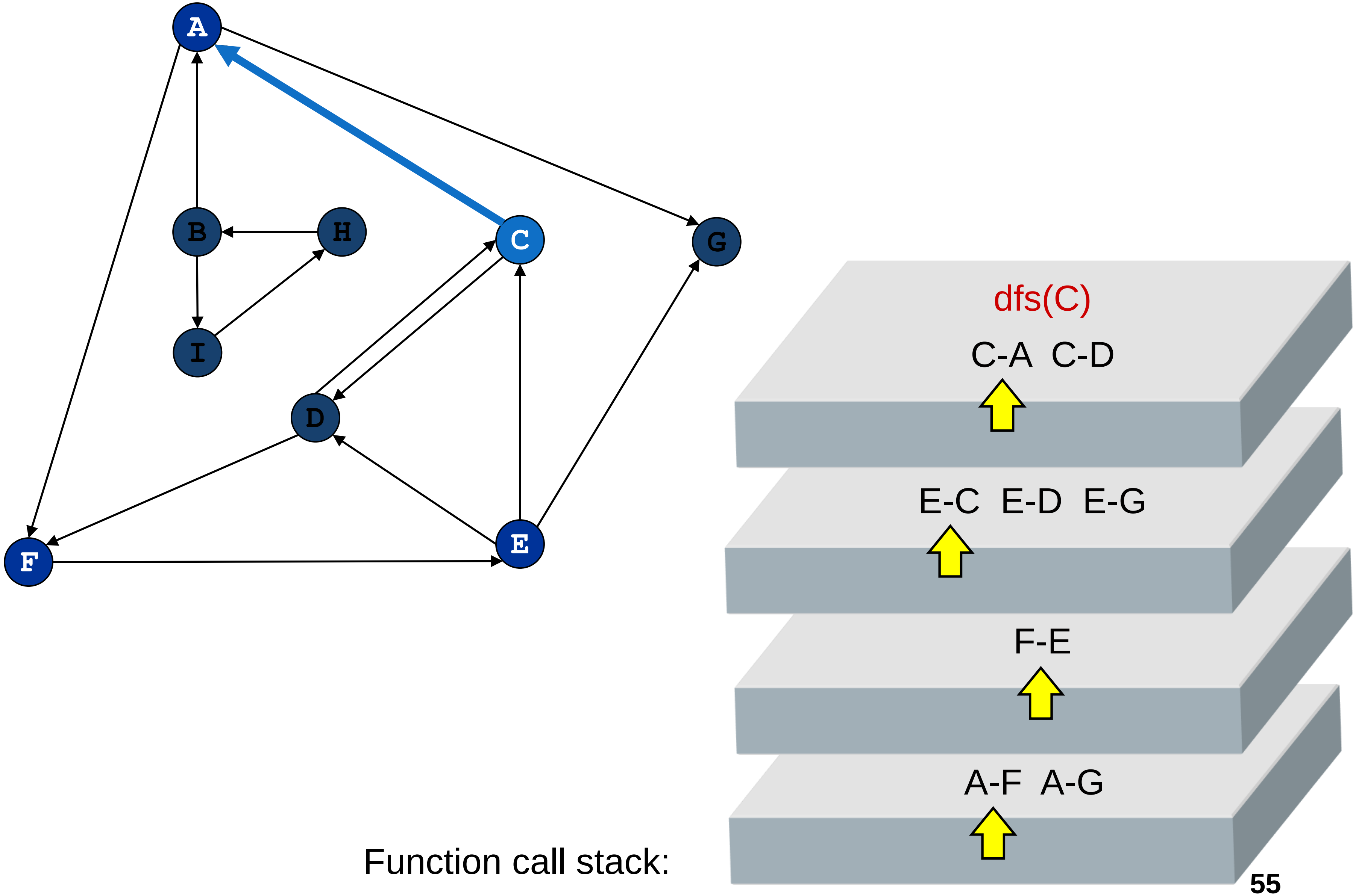
Directed Depth First Search



Function call stack:

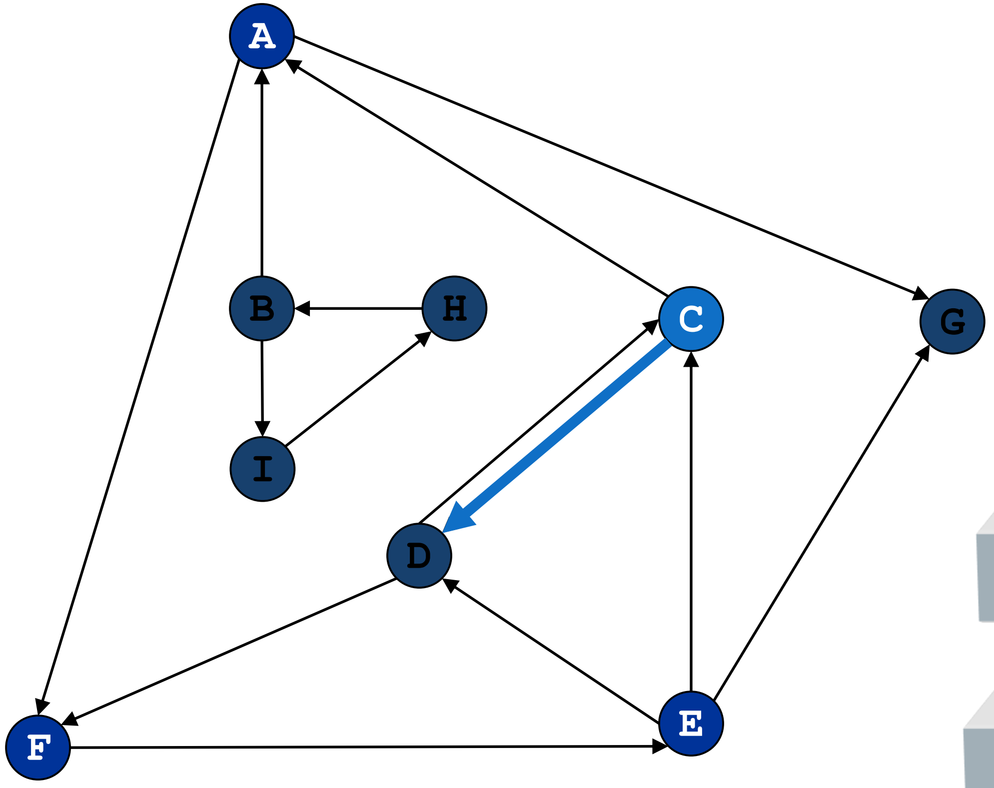


Directed Depth First Search

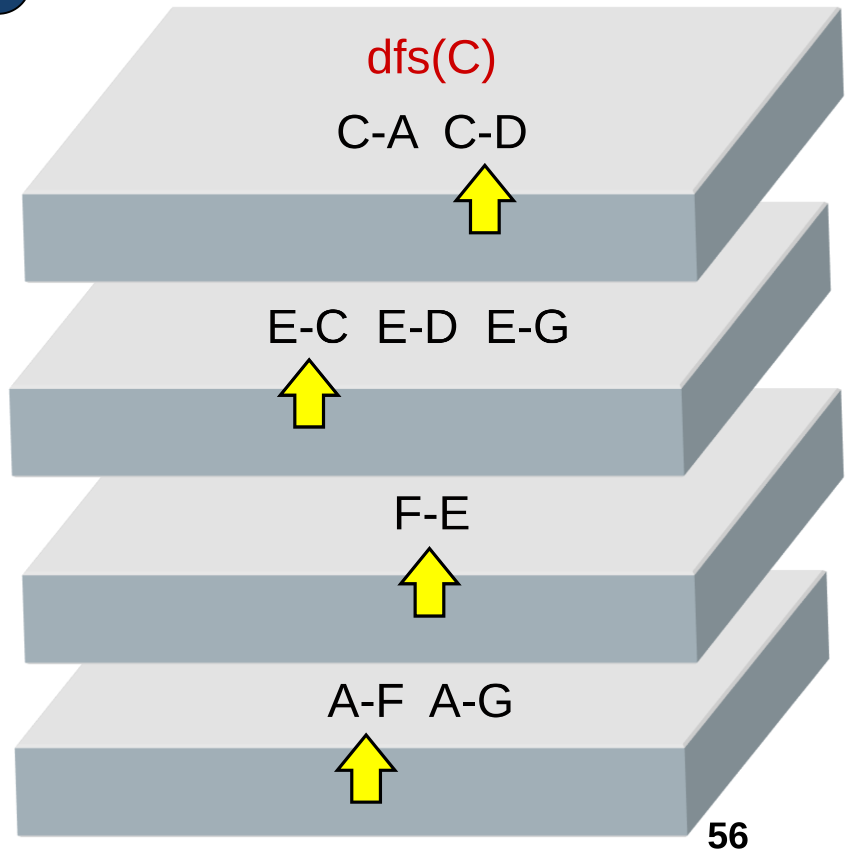


Function call stack:

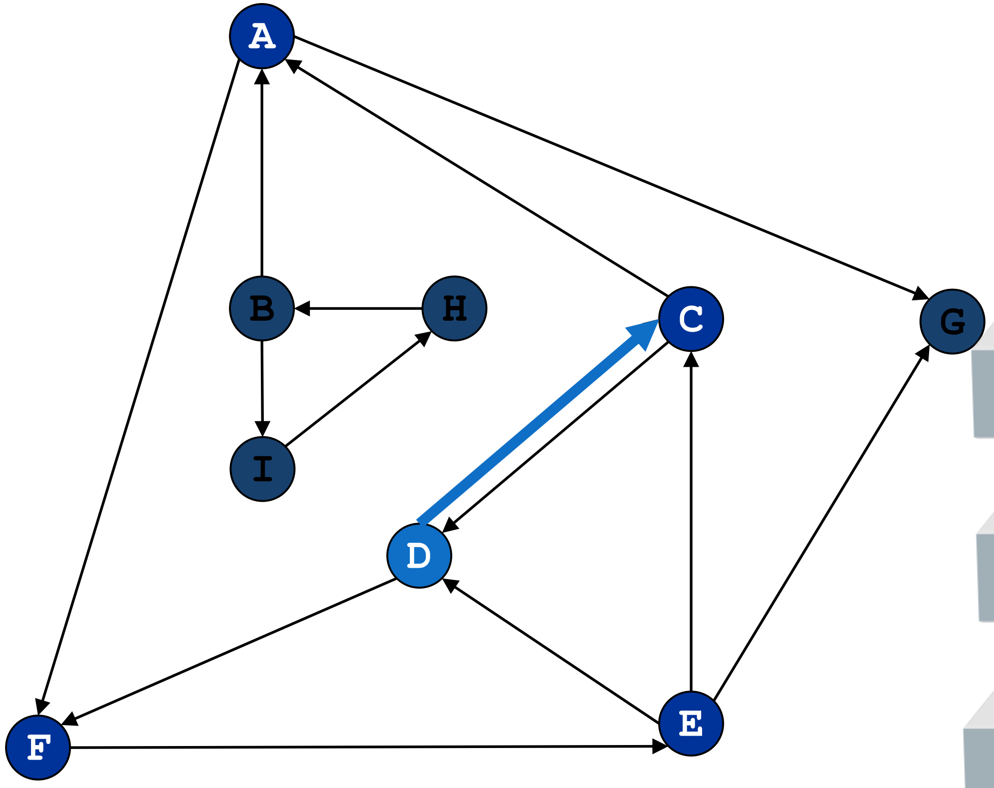
Directed Depth First Search



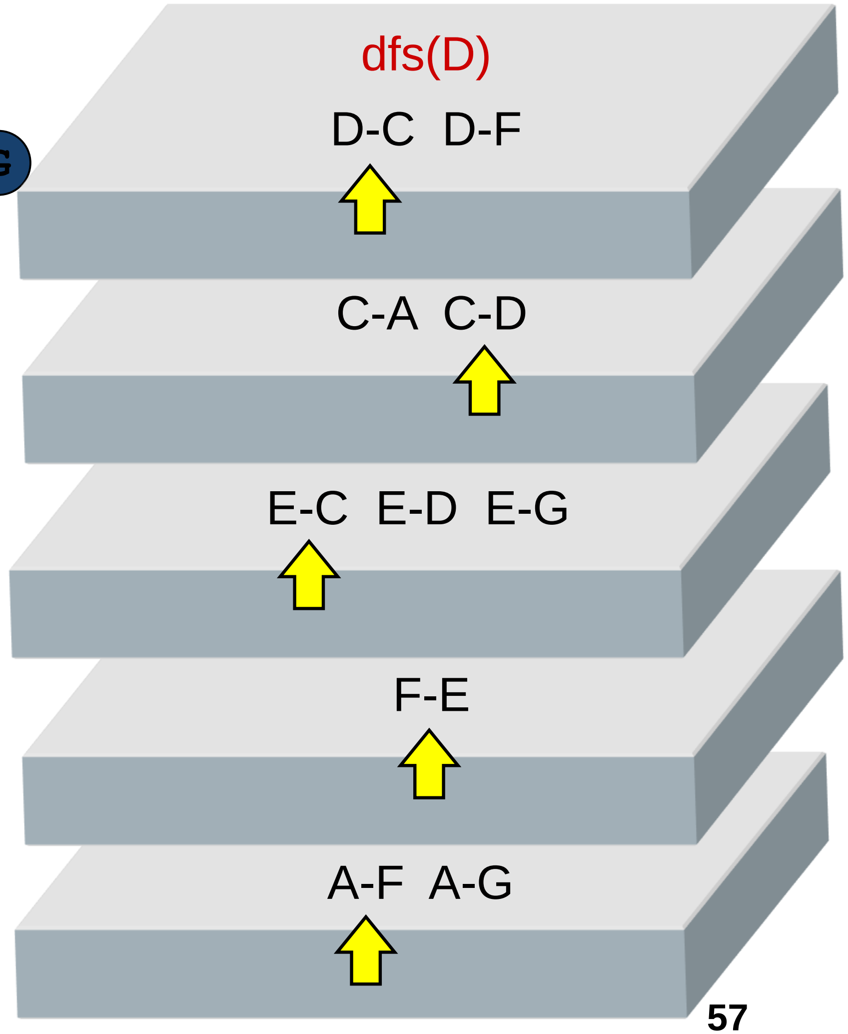
Function call stack:



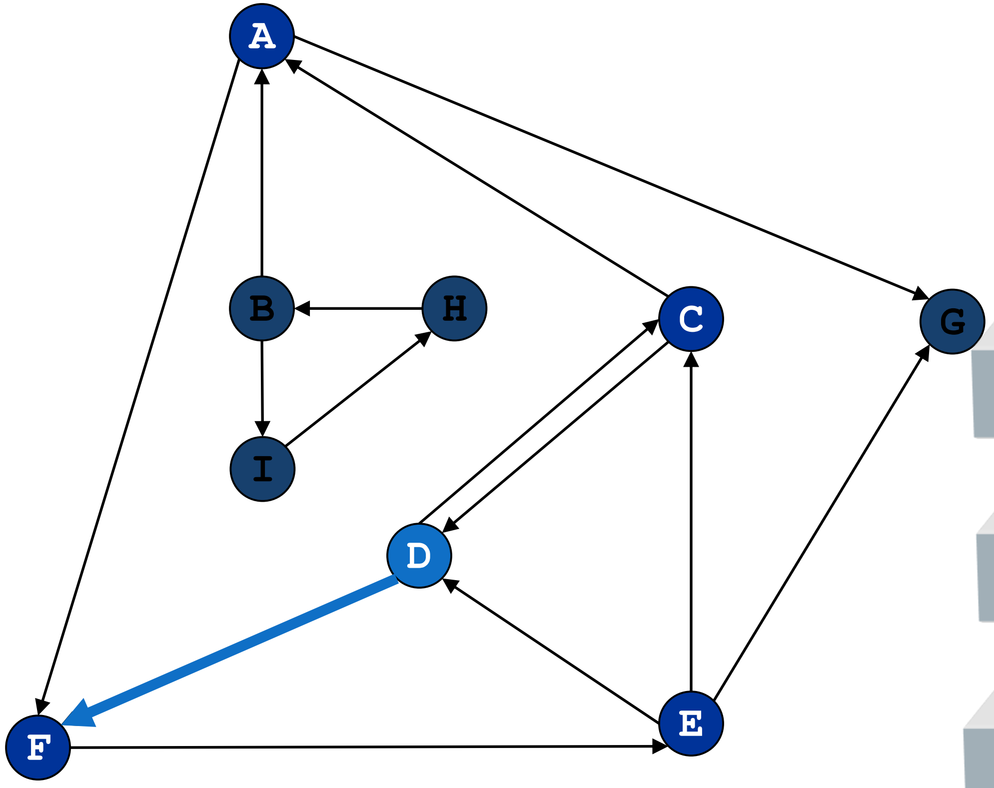
Directed Depth First Search



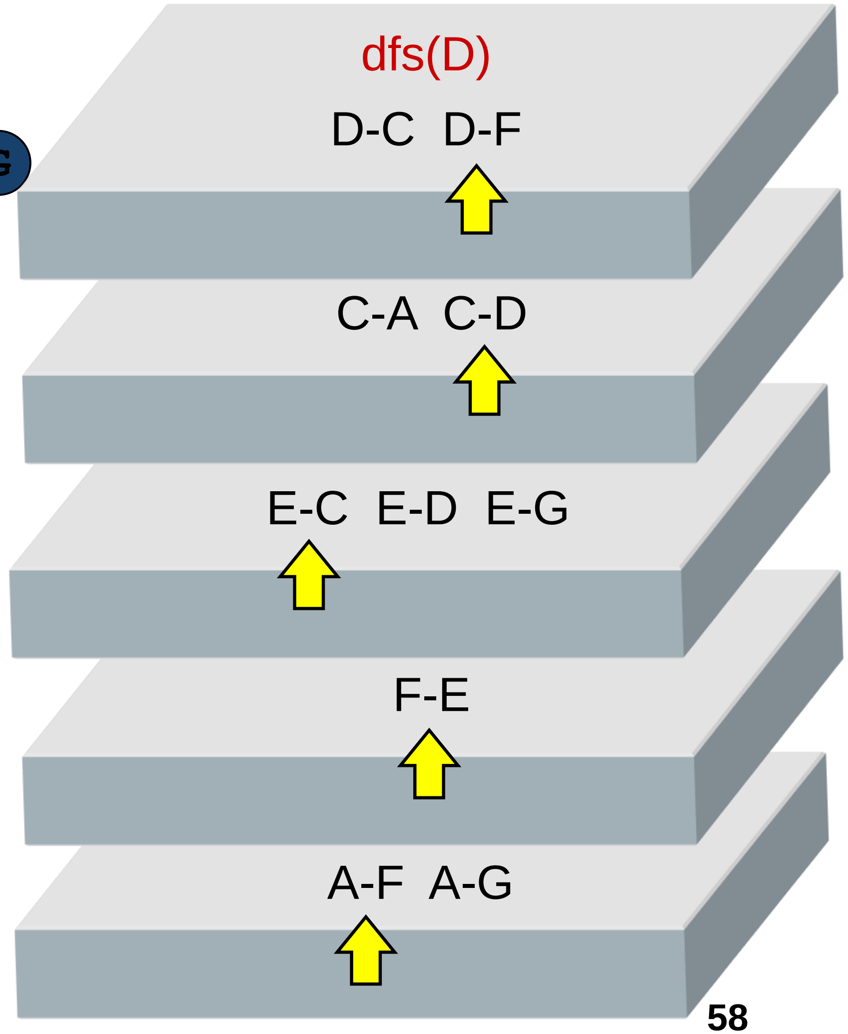
Function call stack:



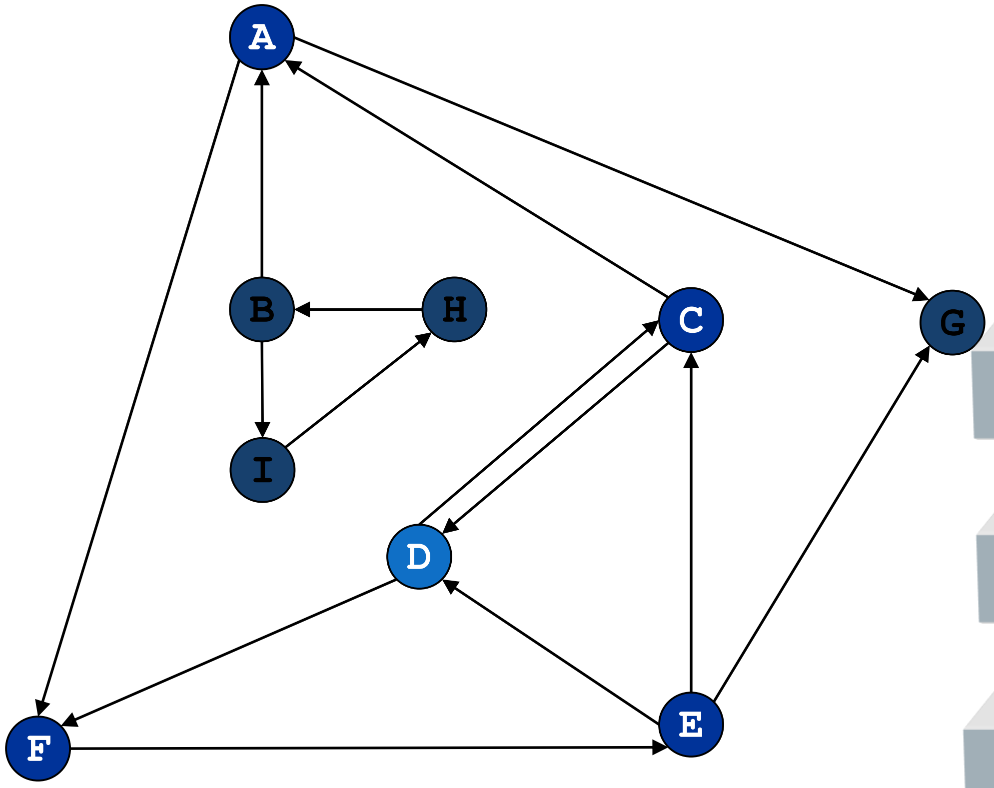
Directed Depth First Search



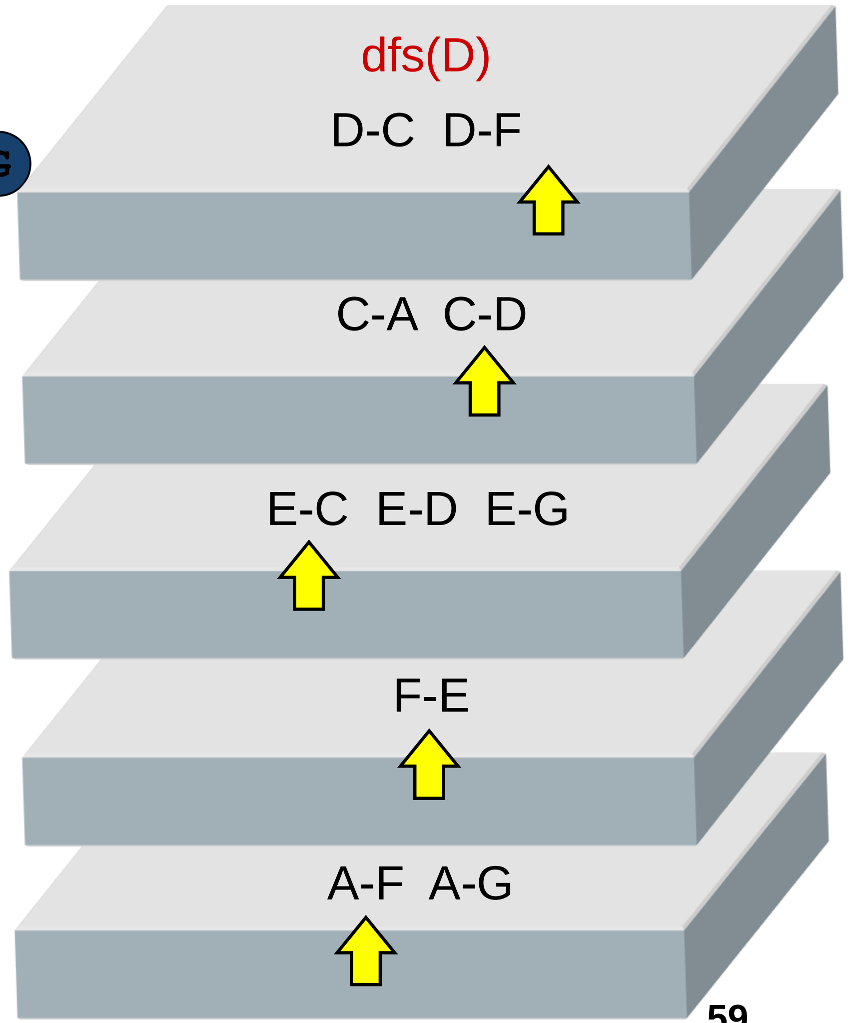
Function call stack:



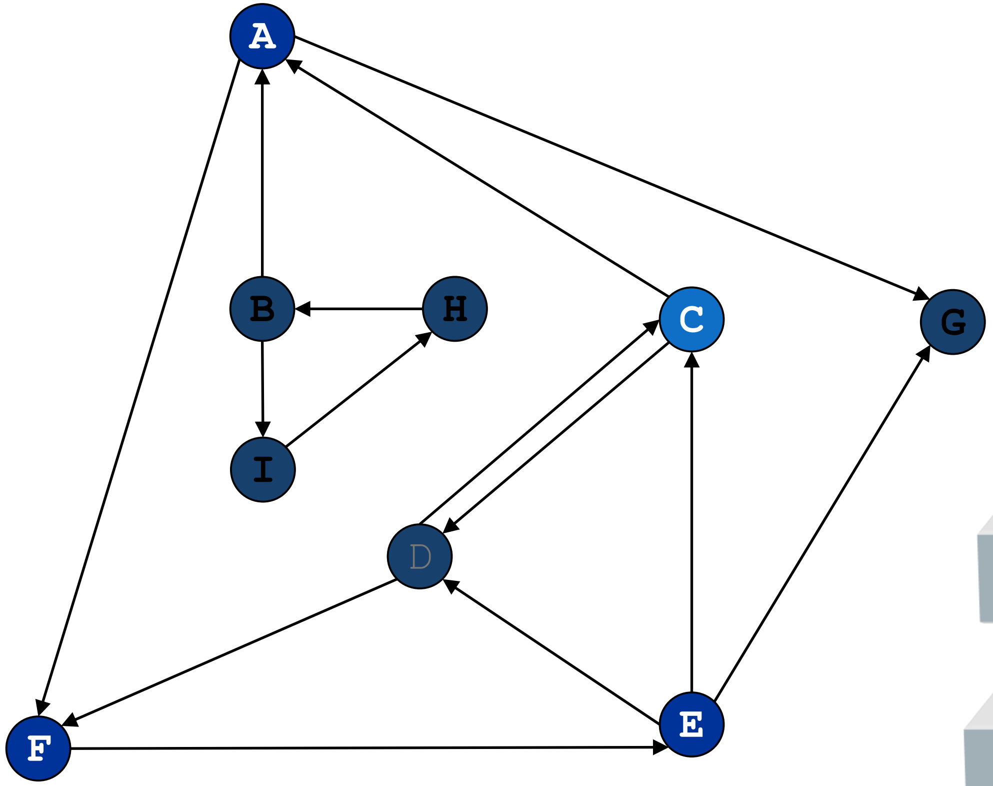
Directed Depth First Search



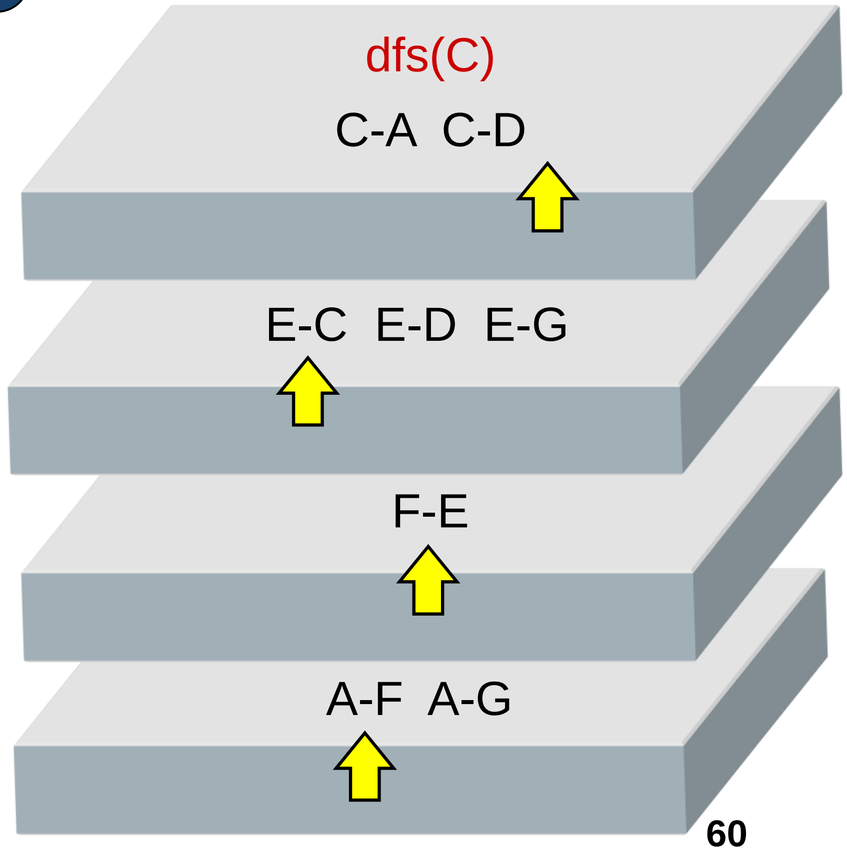
Function call stack:



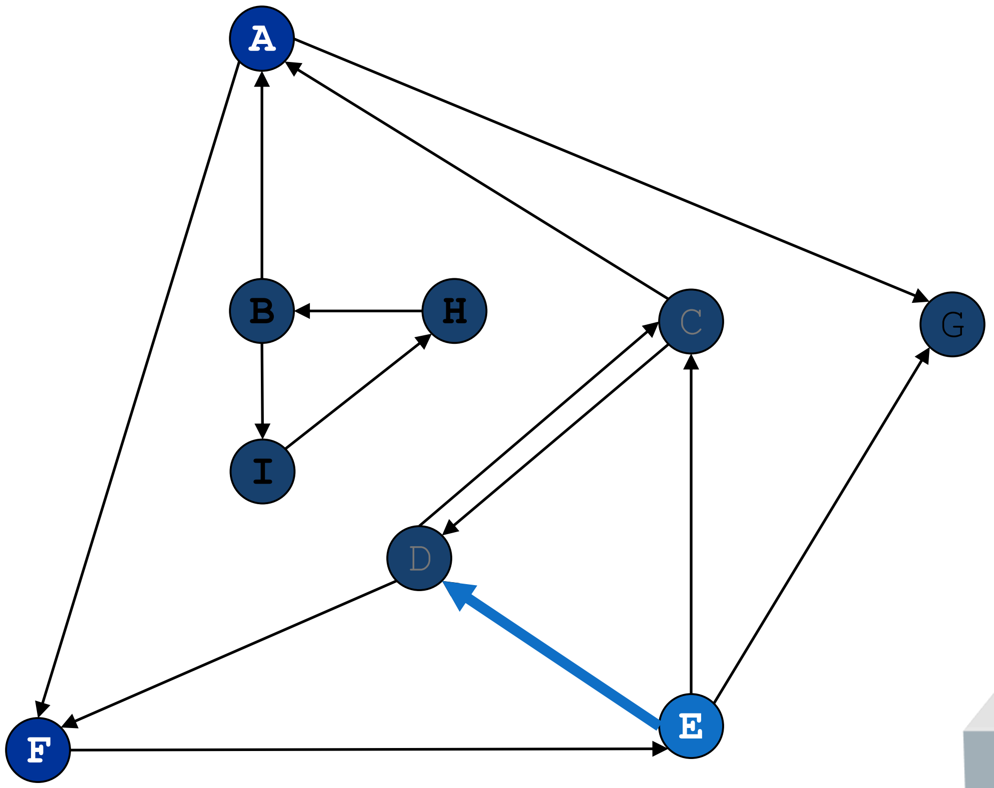
Directed Depth First Search



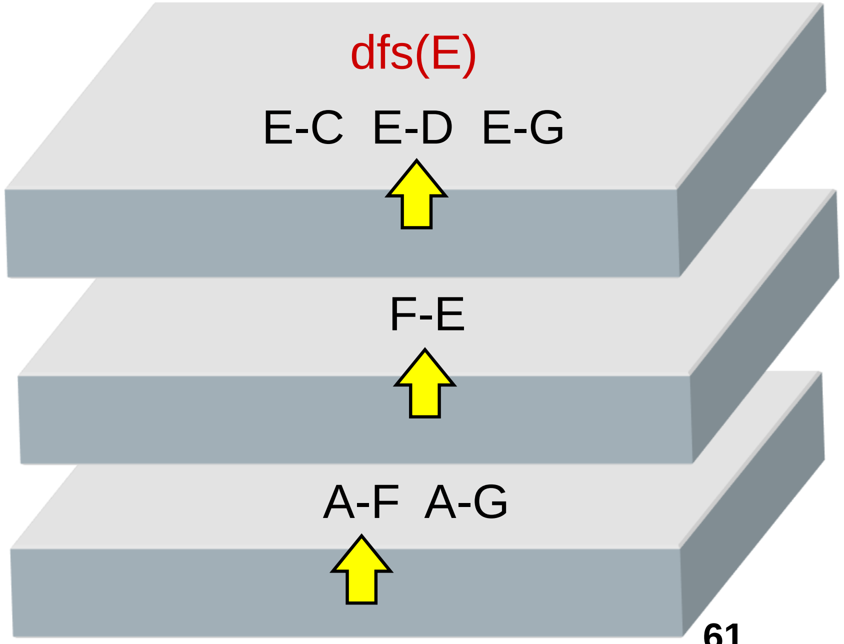
Function call stack:



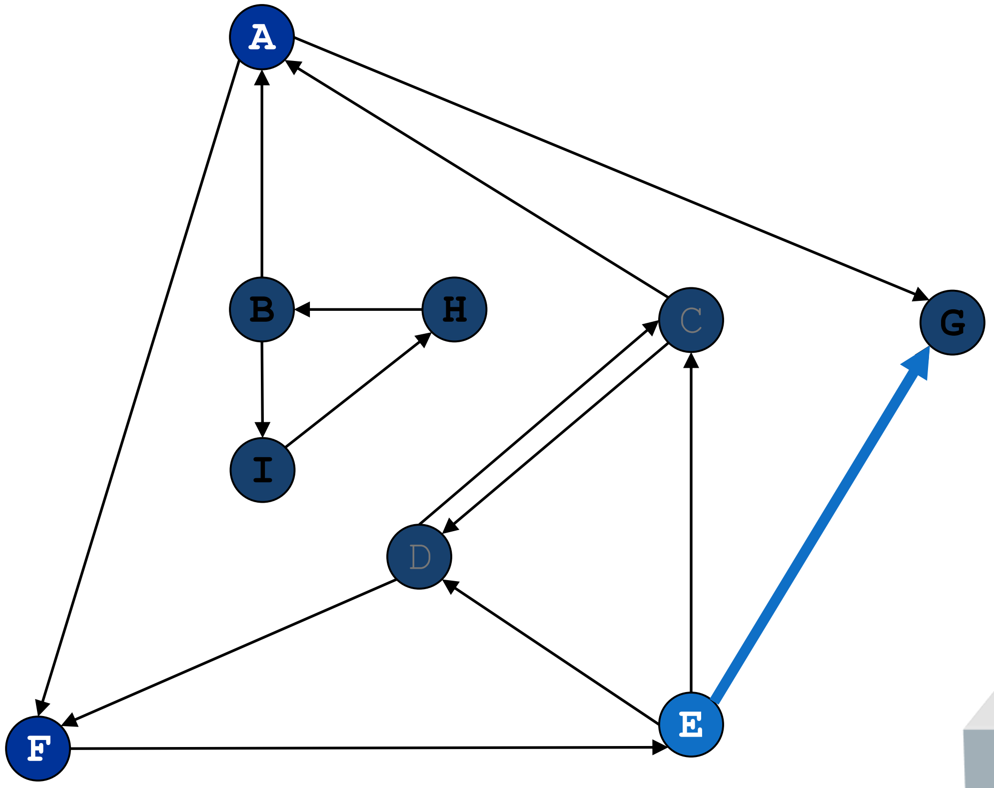
Directed Depth First Search

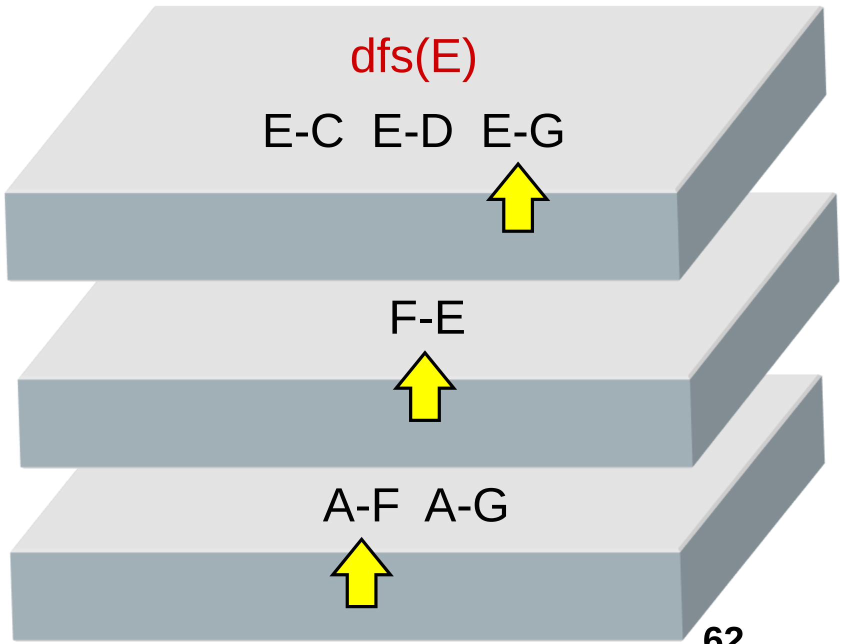


Function call stack:



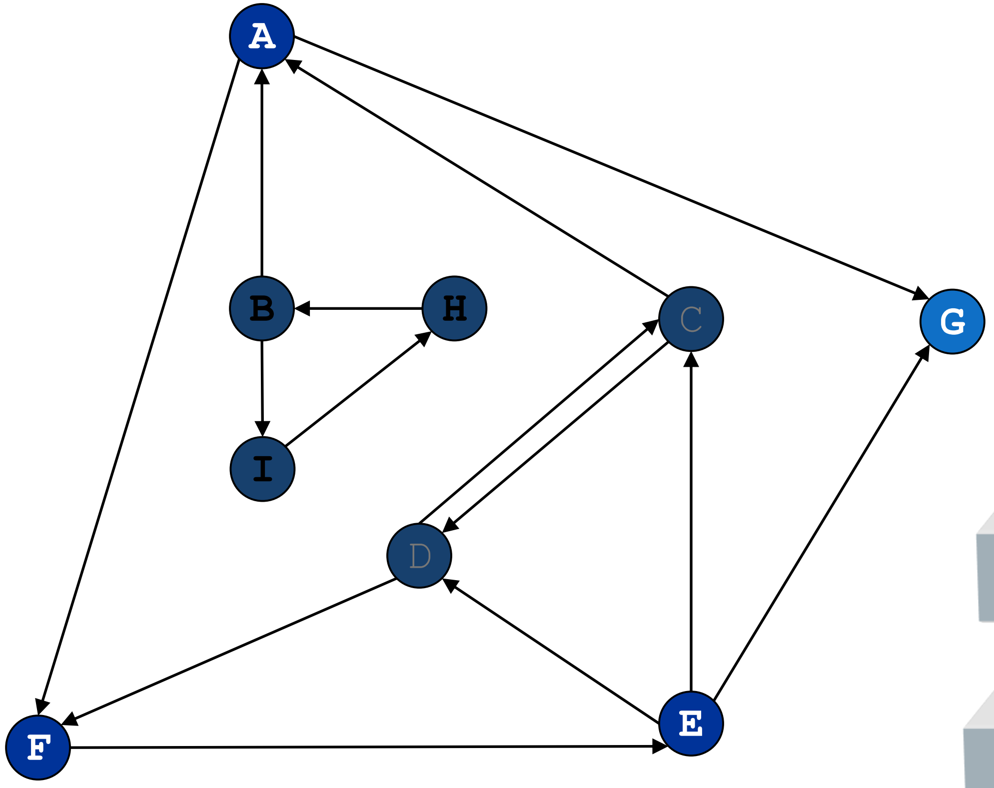
Directed Depth First Search



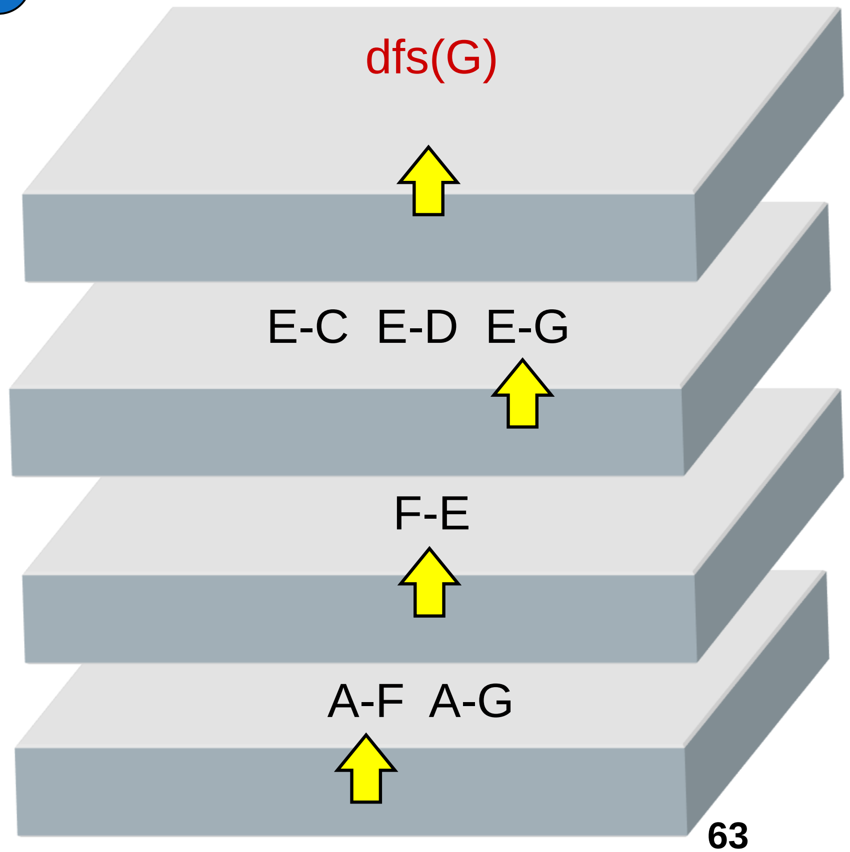


Function call stack:

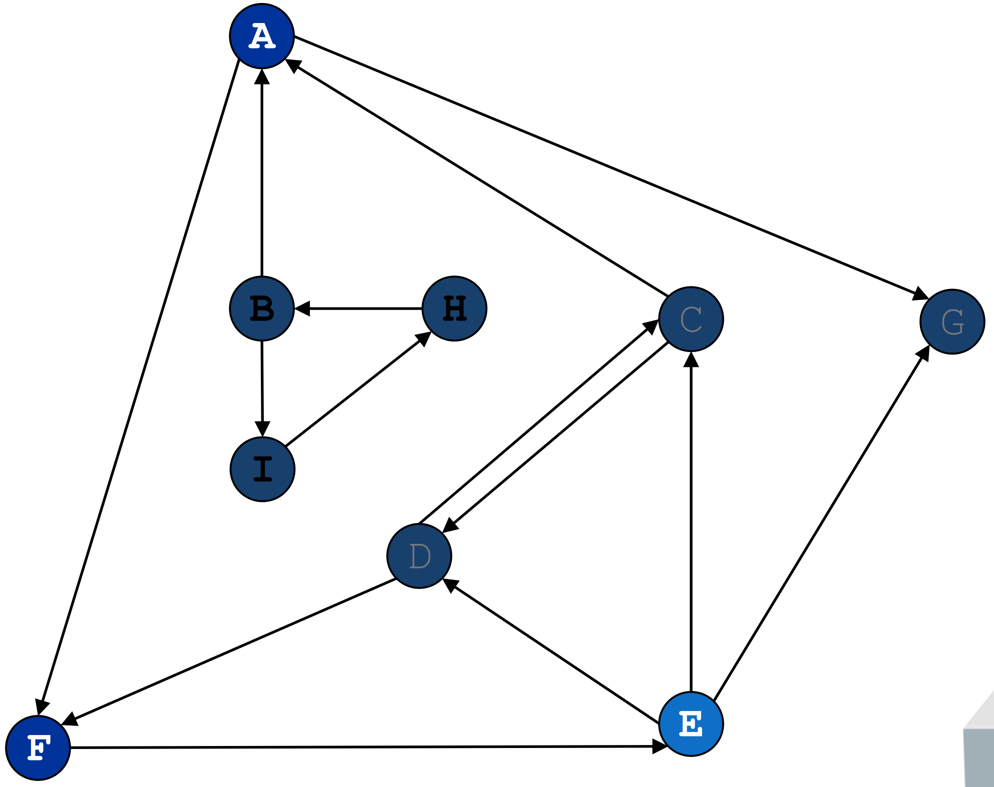
Directed Depth First Search



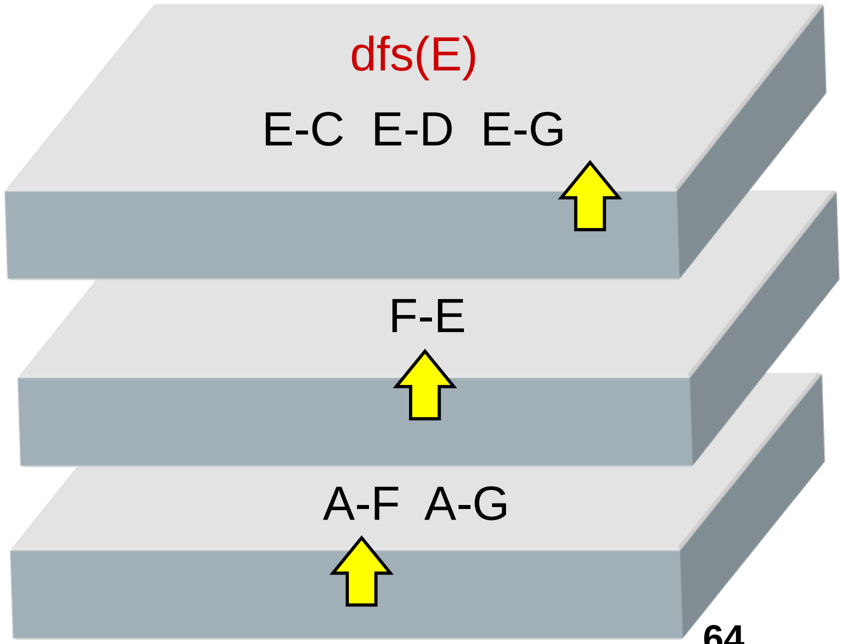
Function call stack:



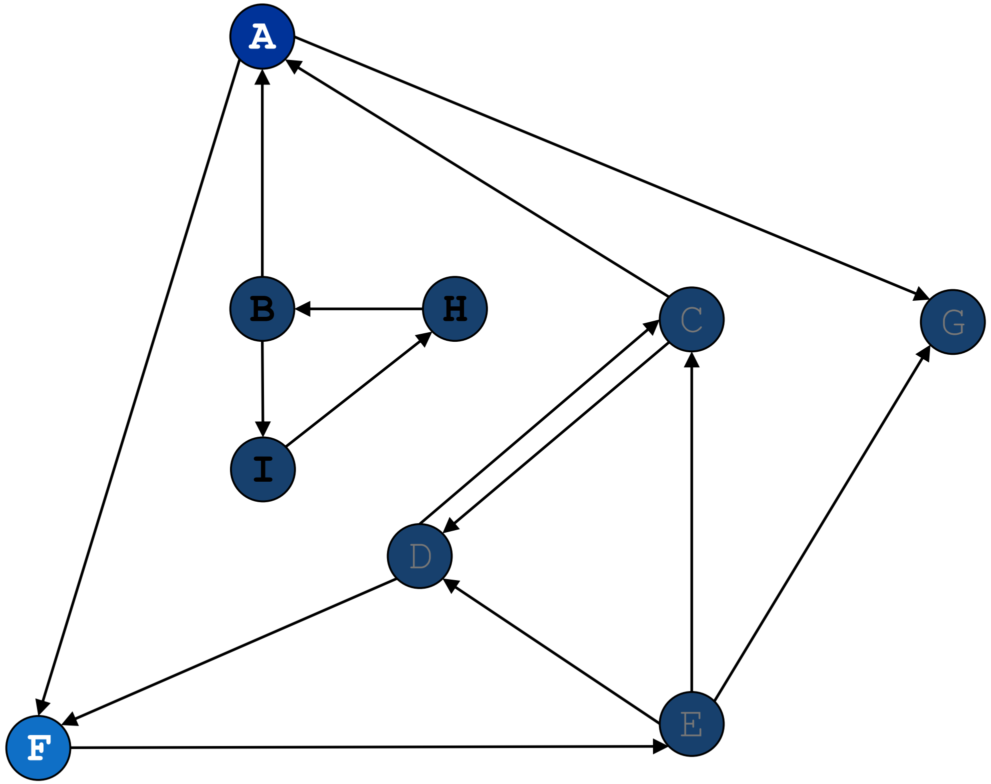
Directed Depth First Search



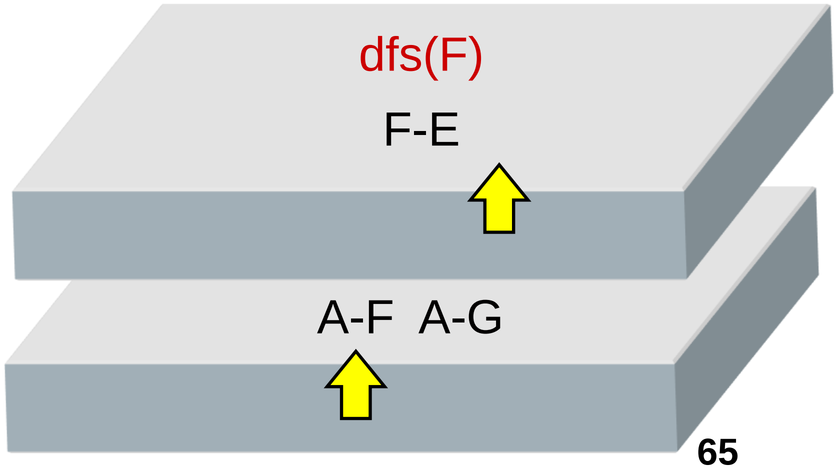
Function call stack:



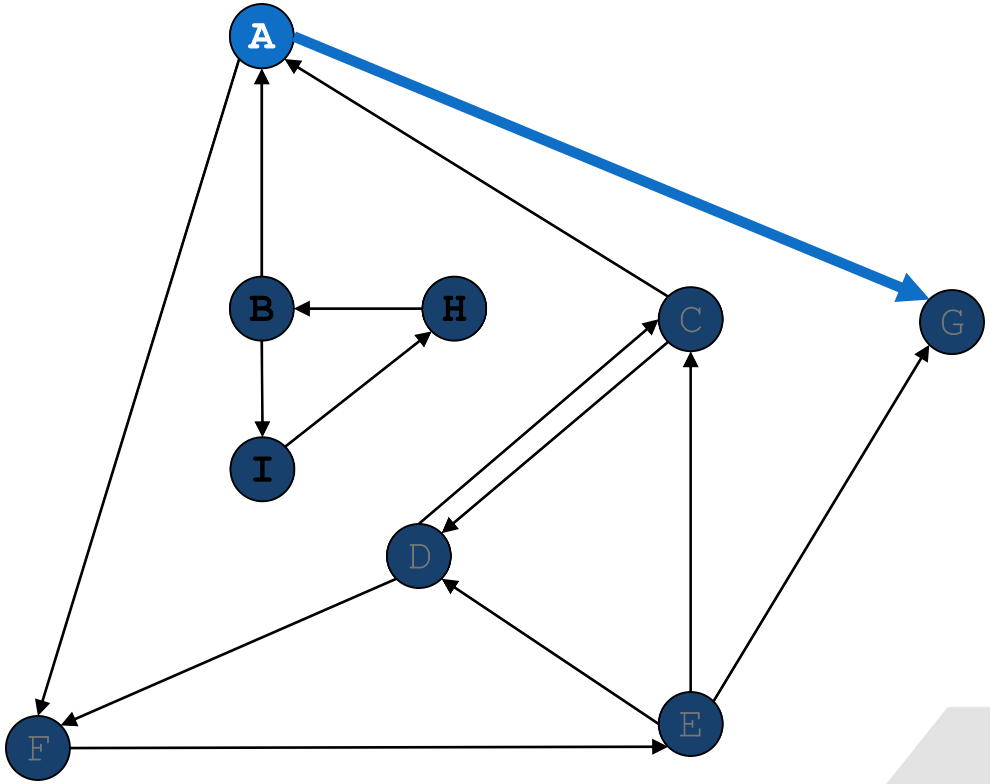
Directed Depth First Search



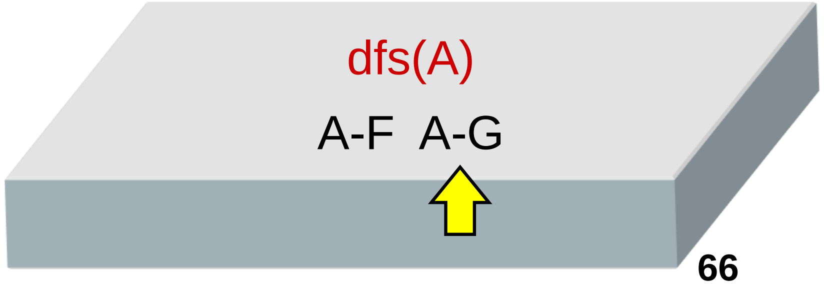
Function call stack:



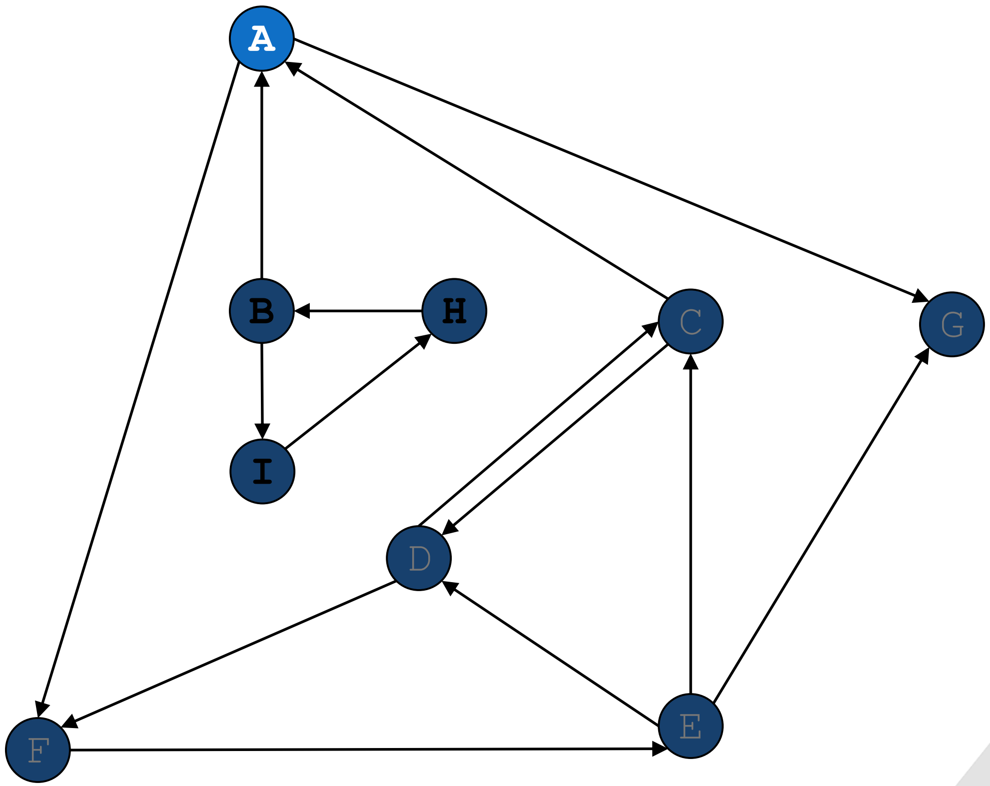
Directed Depth First Search



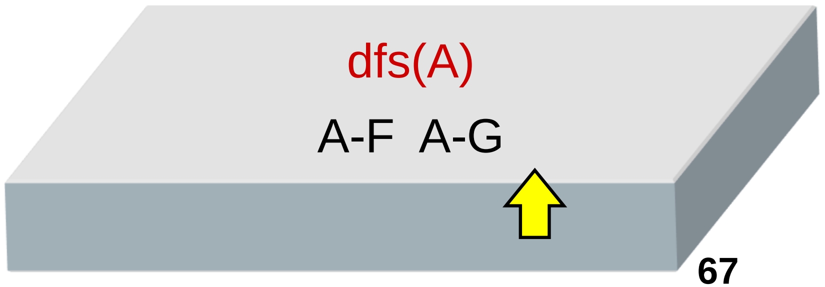
Function call stack:



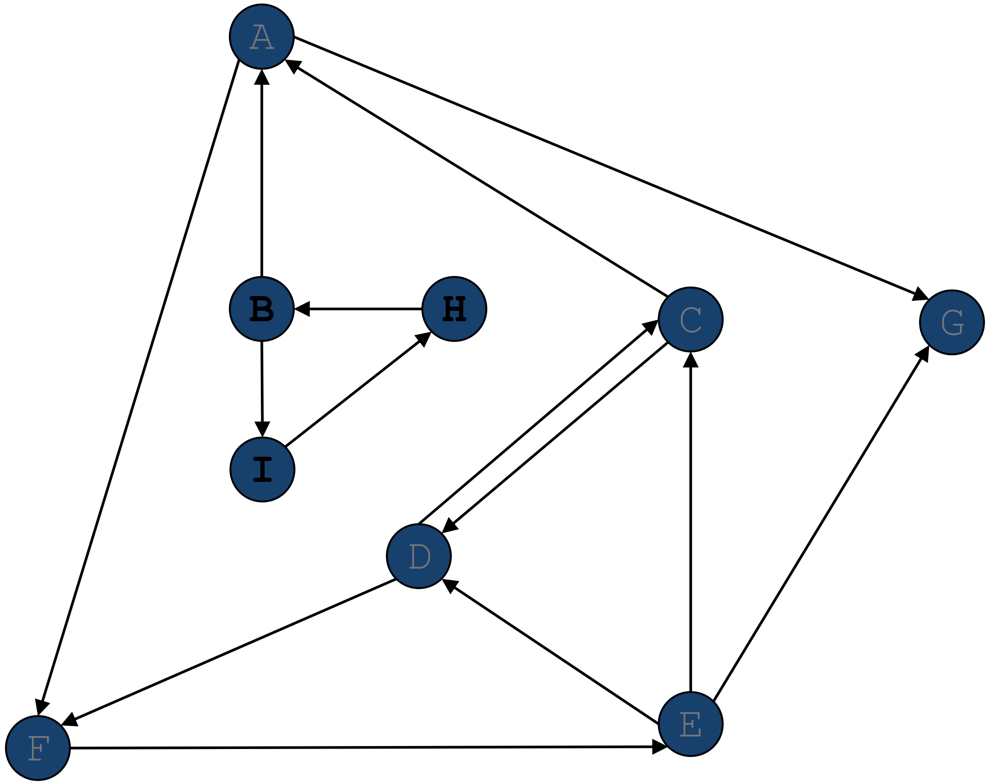
Directed Depth First Search



Function call stack:



Directed Depth First Search



Nodes reachable from A: A, C, D, E, F, G

Pseudo-code

DFS(*G*)

```
for each vertex u ∈ V[G]
    do color[u] ← white
        π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if color[u] = white
        then DFS-Visit(u)
```

DFS-Visit(*u*)

```
color[u] ← GRAY  ∇ White vertex u has been discovered
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = WHITE
        then π[v] ← u
            DFS-Visit(v)
color[u] ← BLACK    ∇ Blacken u; it is finished.
f[u] ← time ← time + 1
```

Uses a global timestamp ***time***.

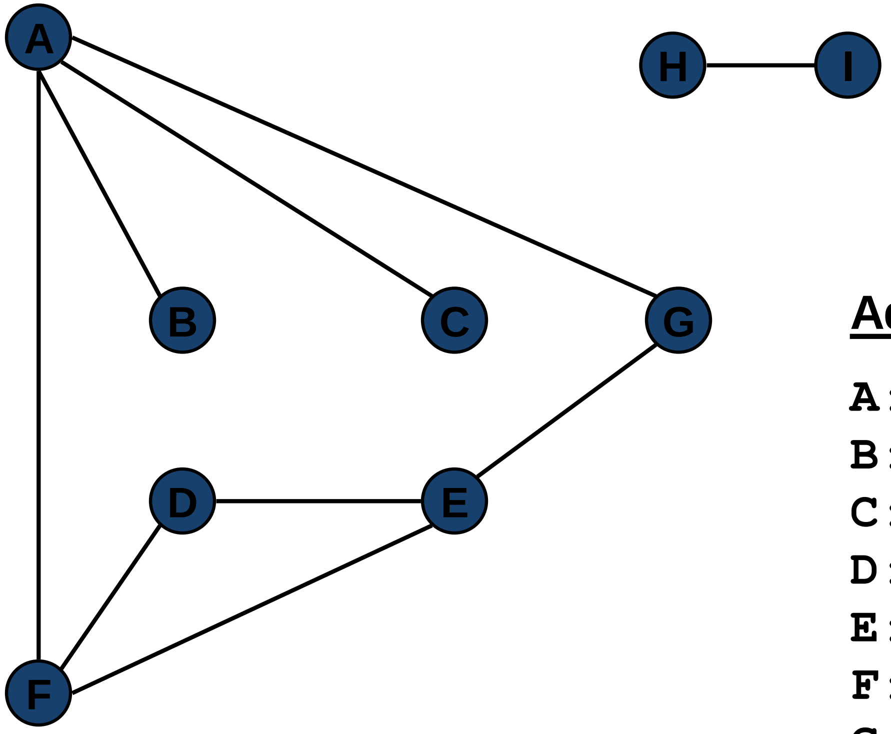
Analysis of DFS

- Loops on lines 1-2 & 5-7 take $\Theta(V)$ time, excluding time to execute DFS-Visit.

- DFS-Visit is called once for each white vertex $v \in V$ when it's painted gray the first time. Lines 3-6 of DFS-Visit is executed |Adj[v]| times. The total cost of executing DFS-Visit is $\sum_{v \in V} |\text{Adj}[v]| = \Theta(E)$

- Total running time of DFS is $\Theta(V+E)$.

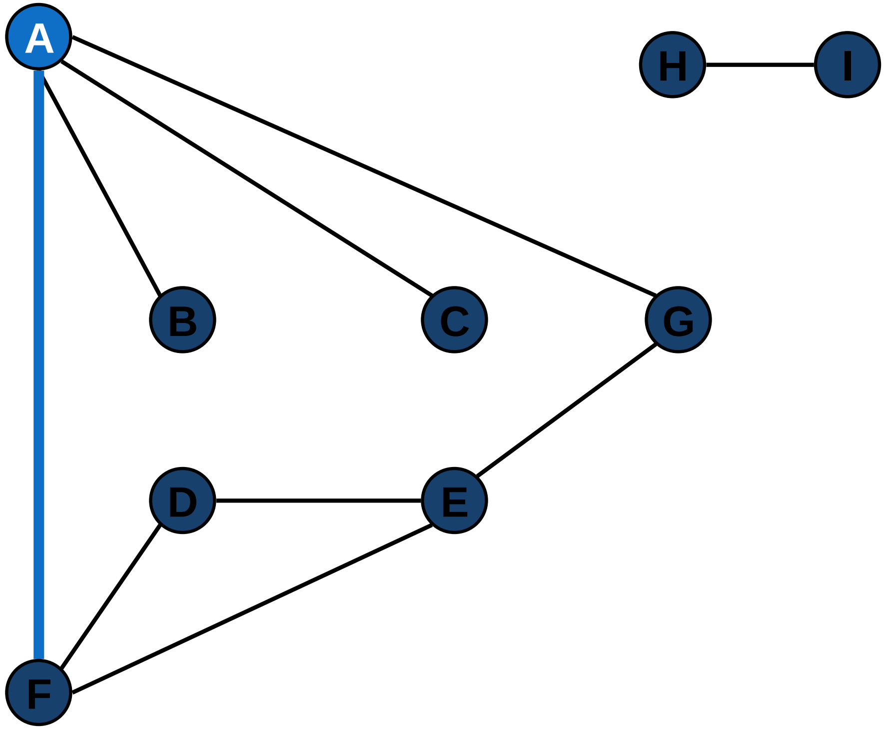
Undirected Depth First Search



Adjacency Lists

```
A:   F C B G
B:   A
C:   A
D:   F E
E:   G F D
F:   A E D
G:   E A
H:   I
I:   H
```

Undirected DFS



F newly discovered

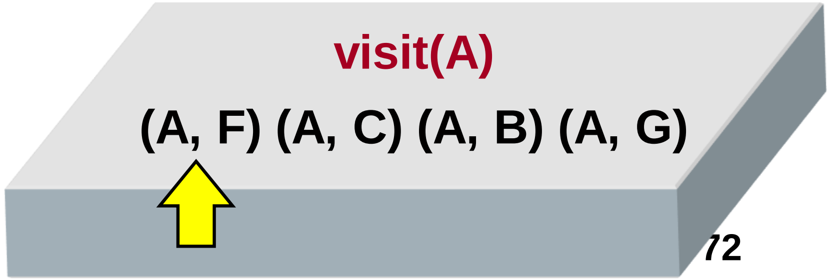
Undiscovered

Marked

Active

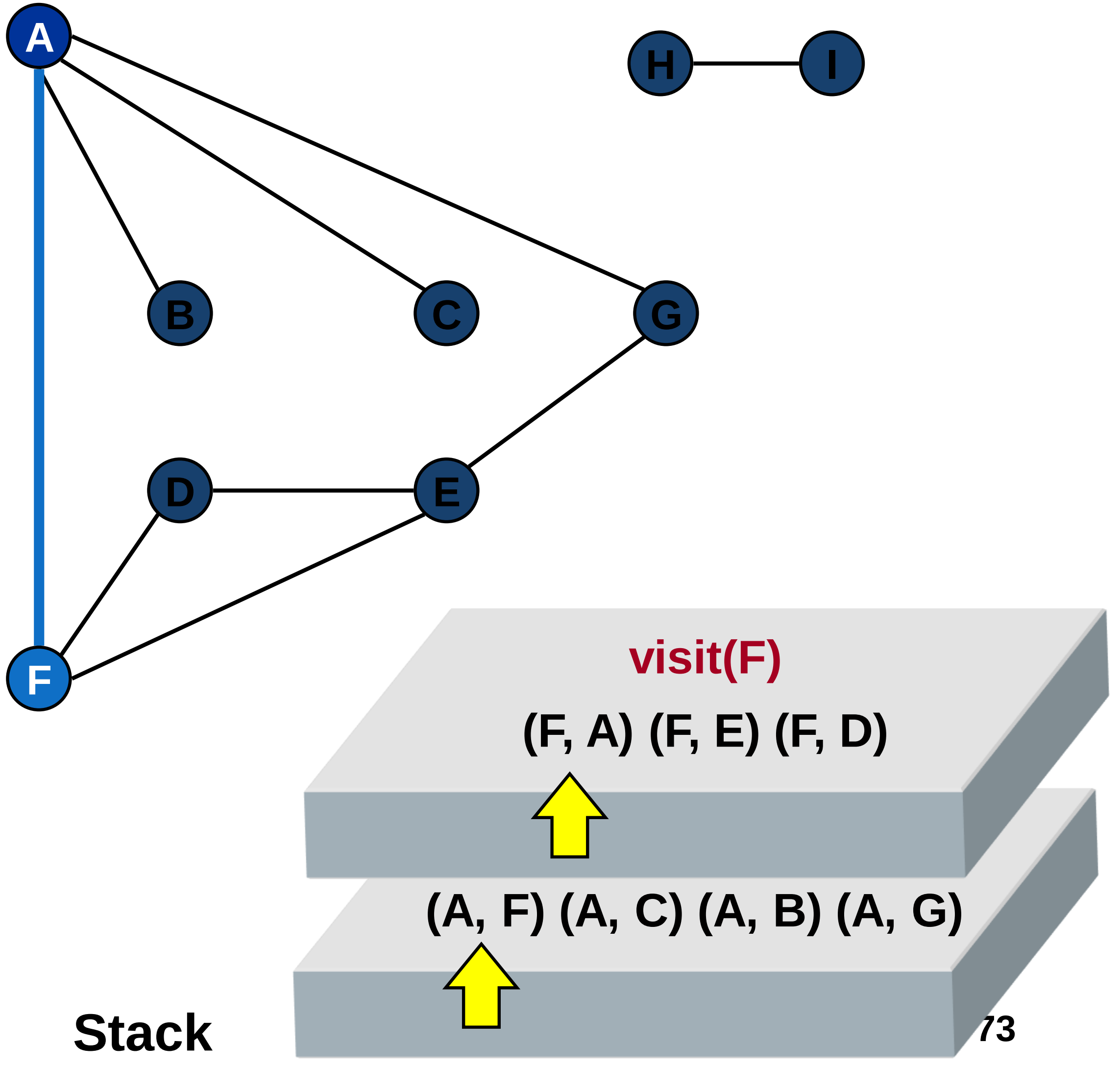
Finished

Stack



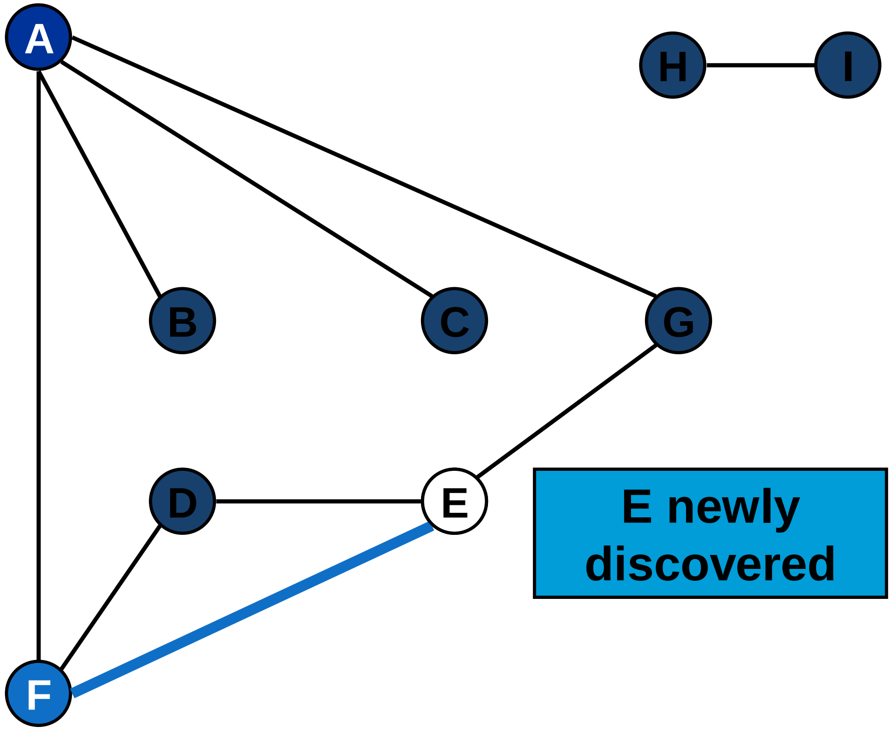
Undirected Depth First Search

A already marked



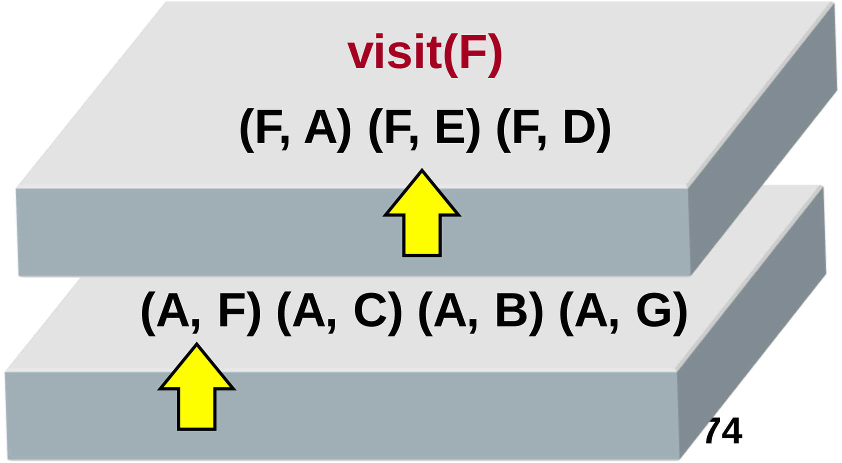
Undiscovered
Marked
Active
Finished

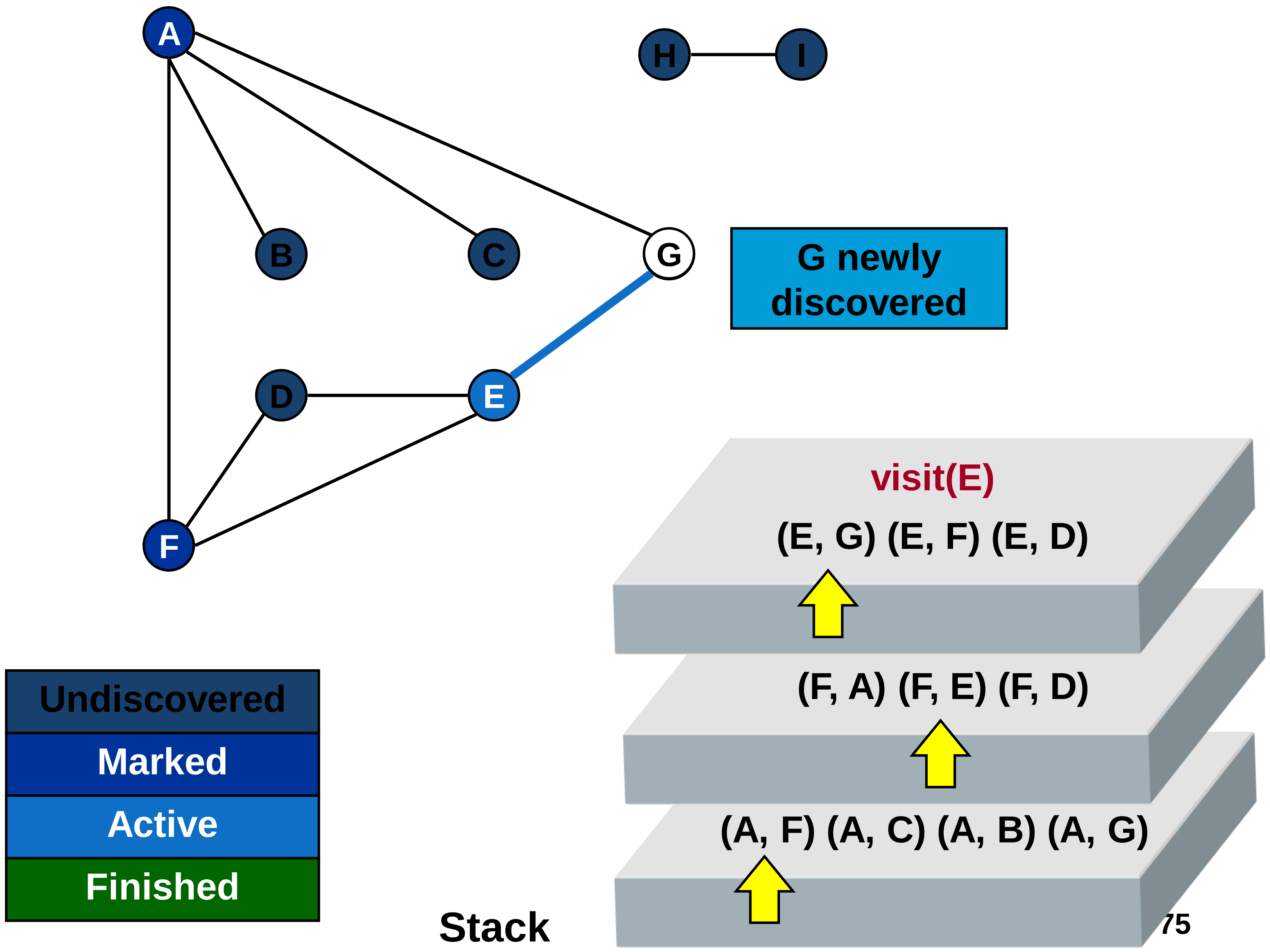
Undirected DFS



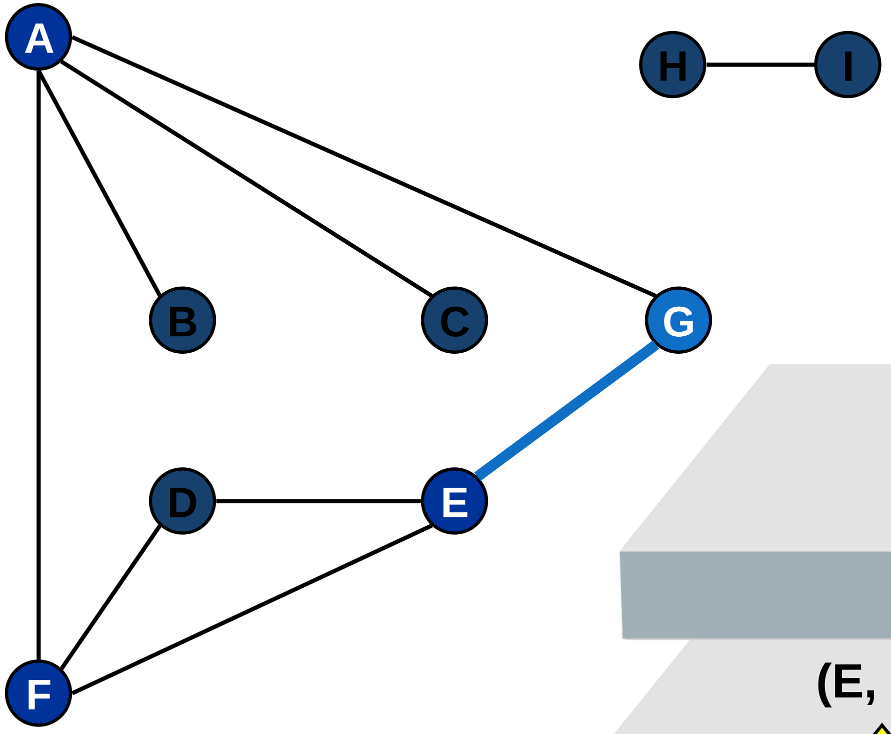
Undiscovered
Marked
Active
Finished

Stack

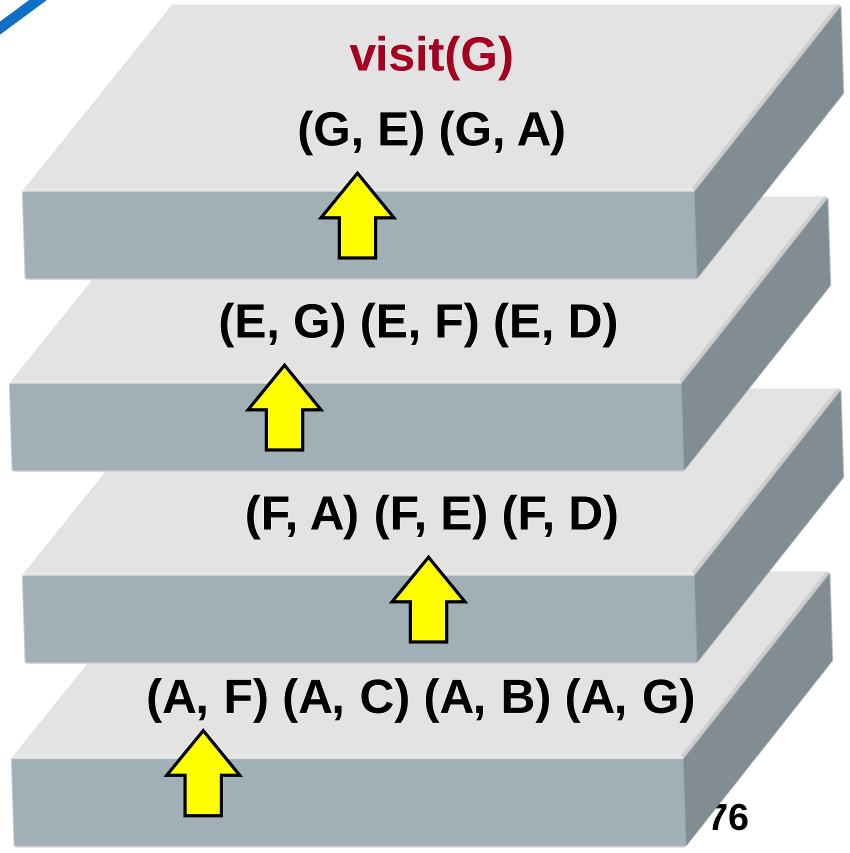



A
H
I
B
C
G
G newly discovered
D
E
F
visit(E)
(E, G) (E, F) (E, D)
(F, A) (F, E) (F, D)
(A, F) (A, C) (A, B) (A, G)
Undiscovered
Marked
Active
Finished
Stack

E already marked

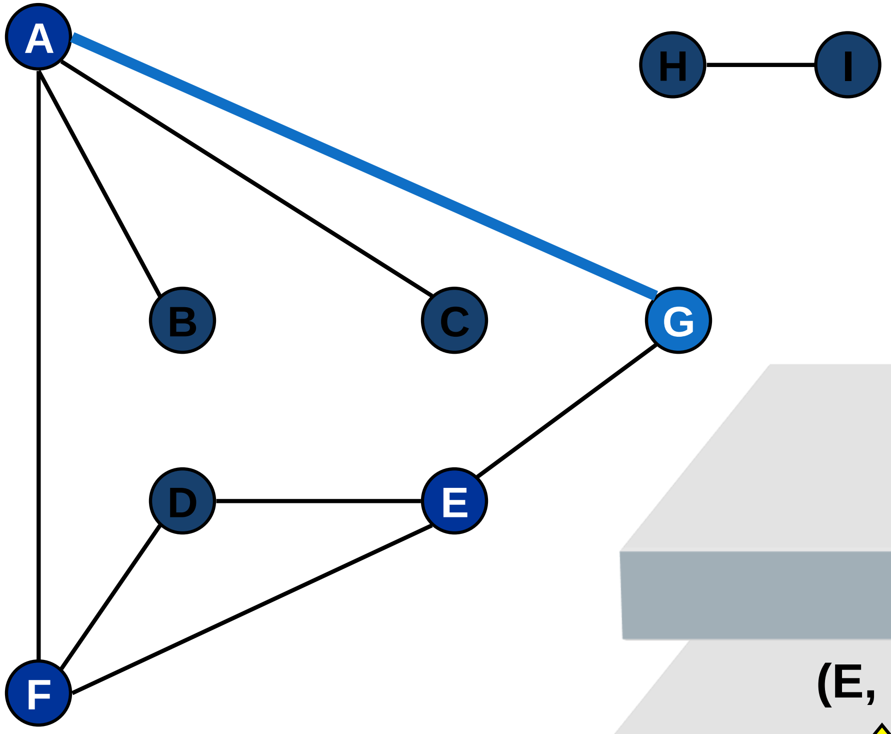


Undiscovered
Marked
Active
Finished



Stack

A already marked

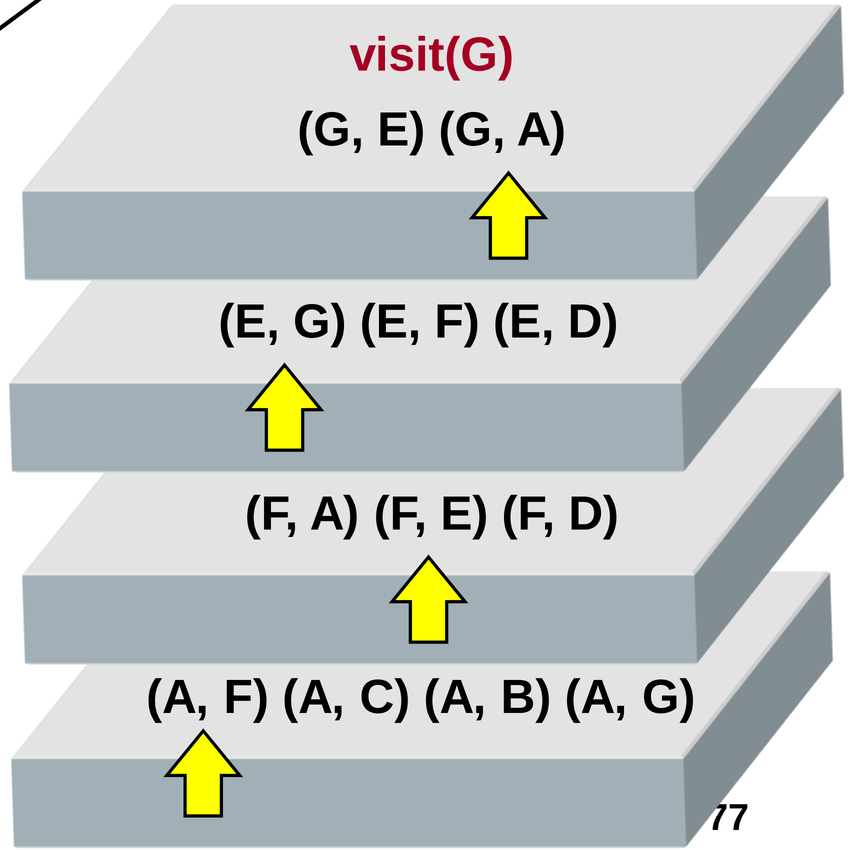


Undiscovered

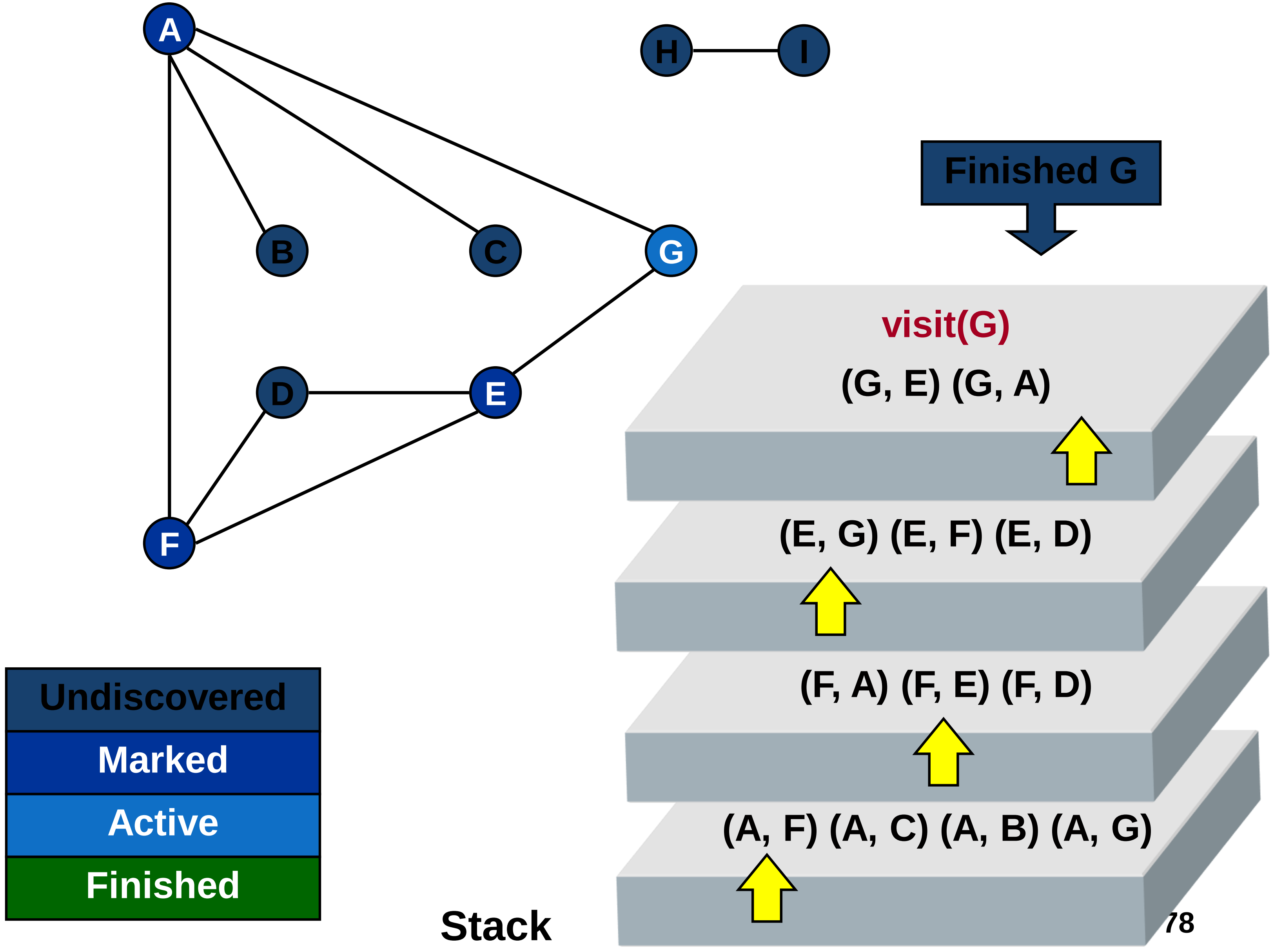
Marked

Active

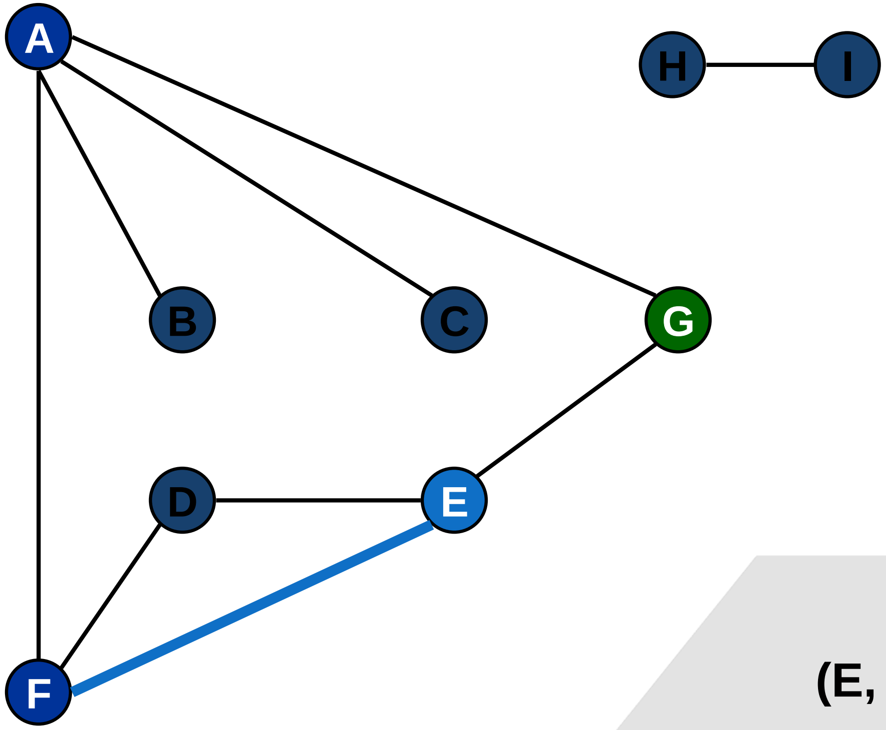
Finished



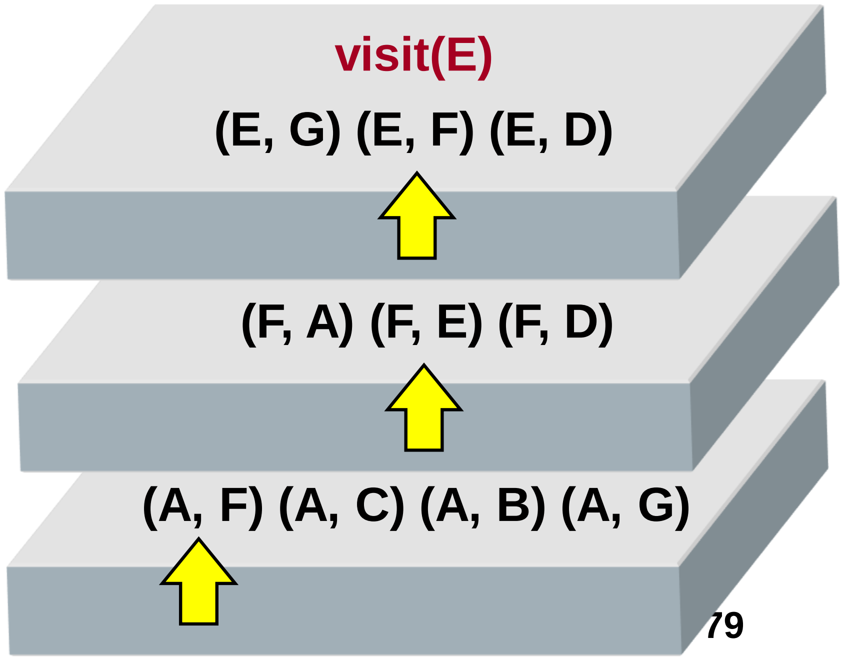
Stack


A
B
C
G
D
E
F
H
I
Finished G
visit(G)
(G, E) (G, A)
(E, G) (E, F) (E, D)
(F, A) (F, E) (F, D)
(A, F) (A, C) (A, B) (A, G)
Stack
Undiscovered
Marked
Active
Finished

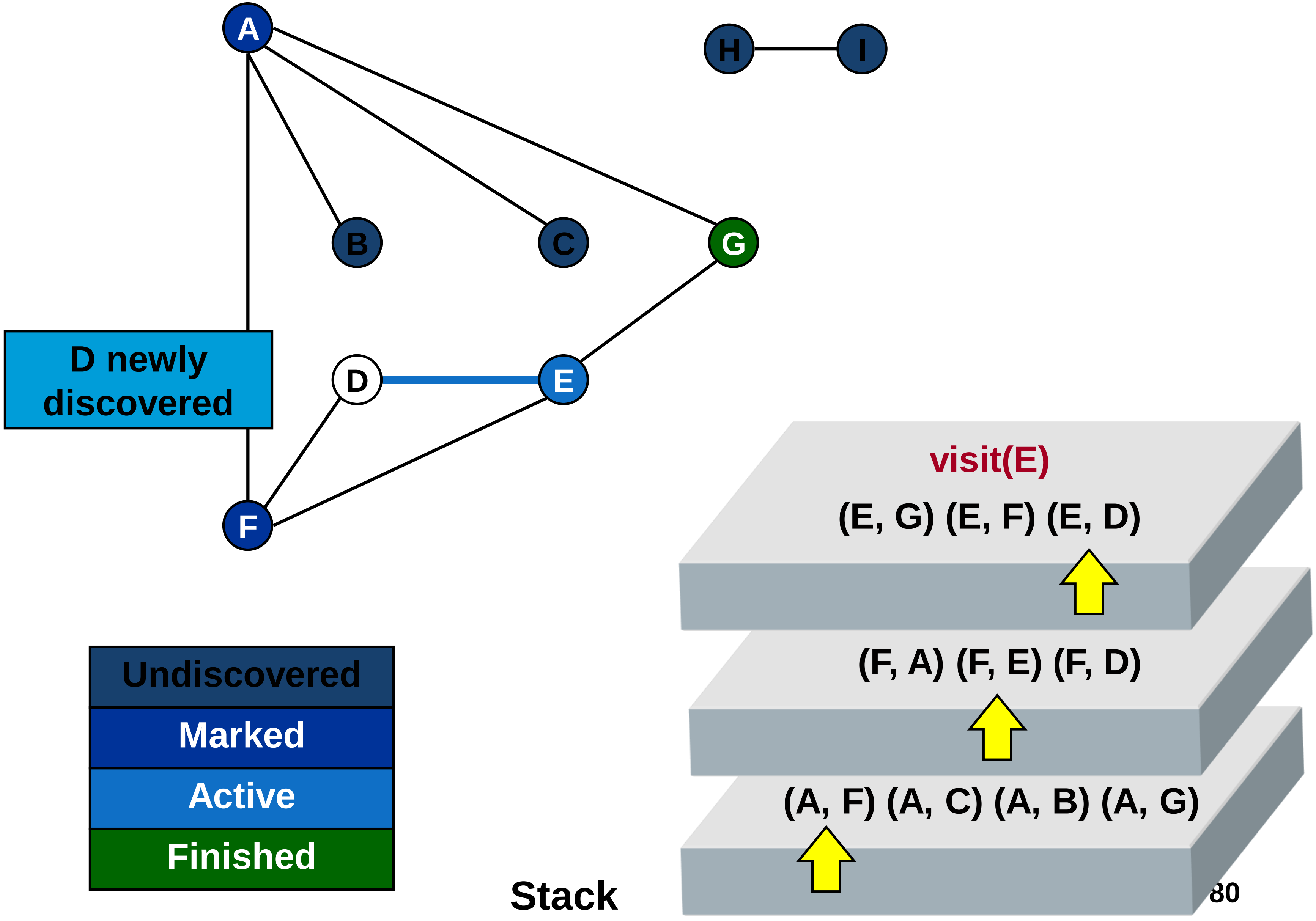
F already marked

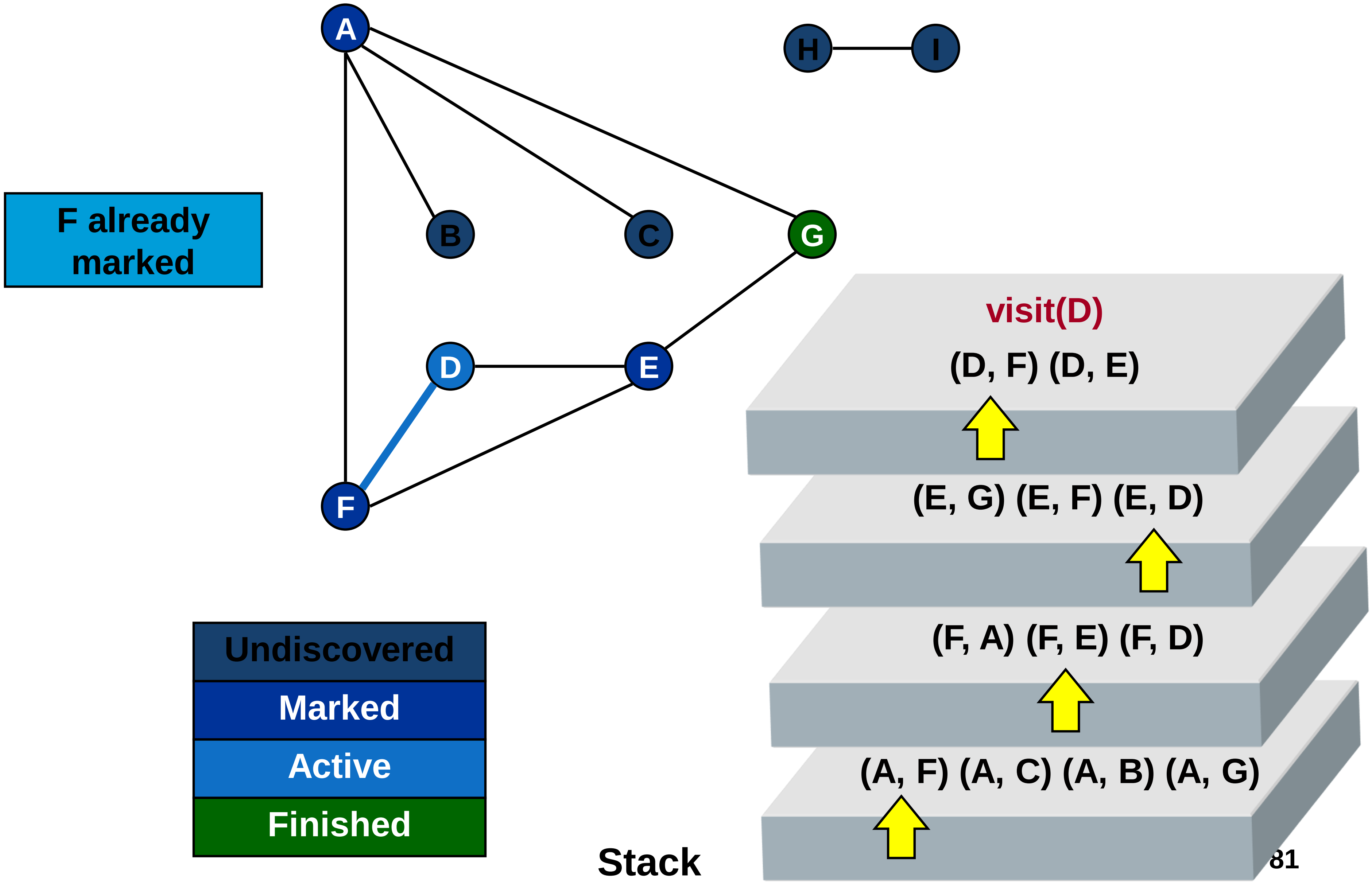


Undiscovered
Marked
Active
Finished

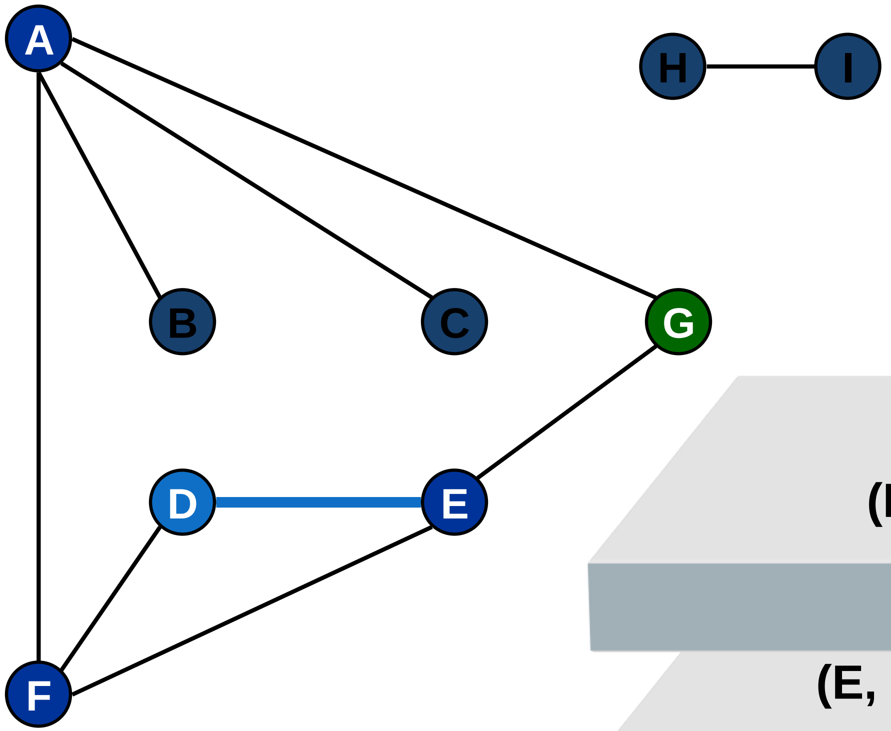


Stack


A
B
C
G
H
I
D newly discovered
D
E
F
Undiscovered
Marked
Active
Finished
visit(E)
(E, G) (E, F) (E, D)
(F, A) (F, E) (F, D)
(A, F) (A, C) (A, B) (A, G)
Stack


A
B
C
D
E
F
G
H
I
F already marked
Undiscovered
Marked
Active
Finished
visit(D)
(D, F) (D, E)
(E, G) (E, F) (E, D)
(F, A) (F, E) (F, D)
(A, F) (A, C) (A, B) (A, G)
Stack

E already marked



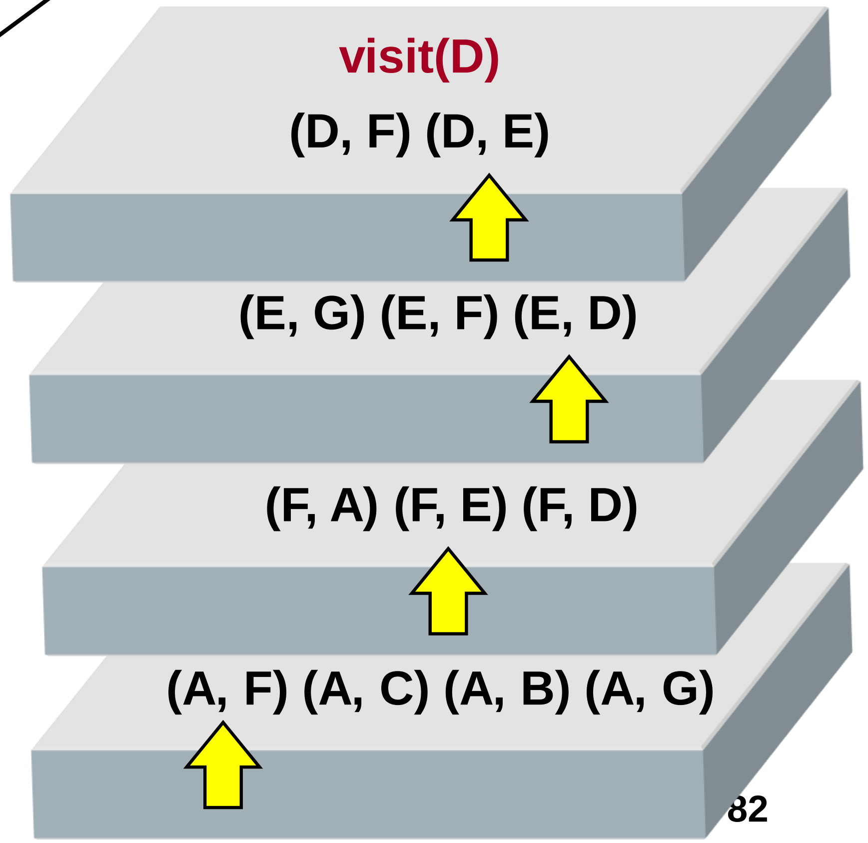
Undiscovered

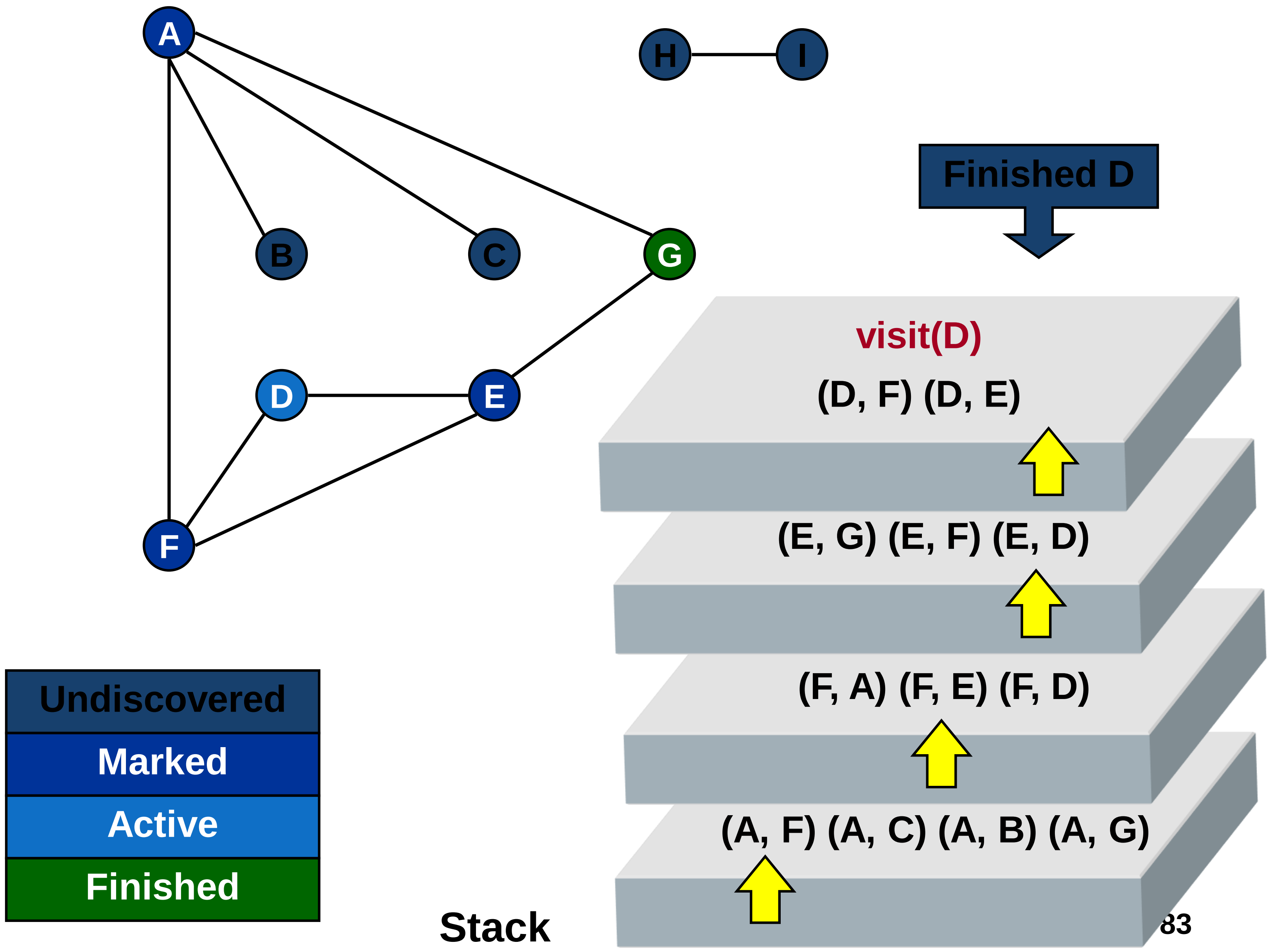
Marked

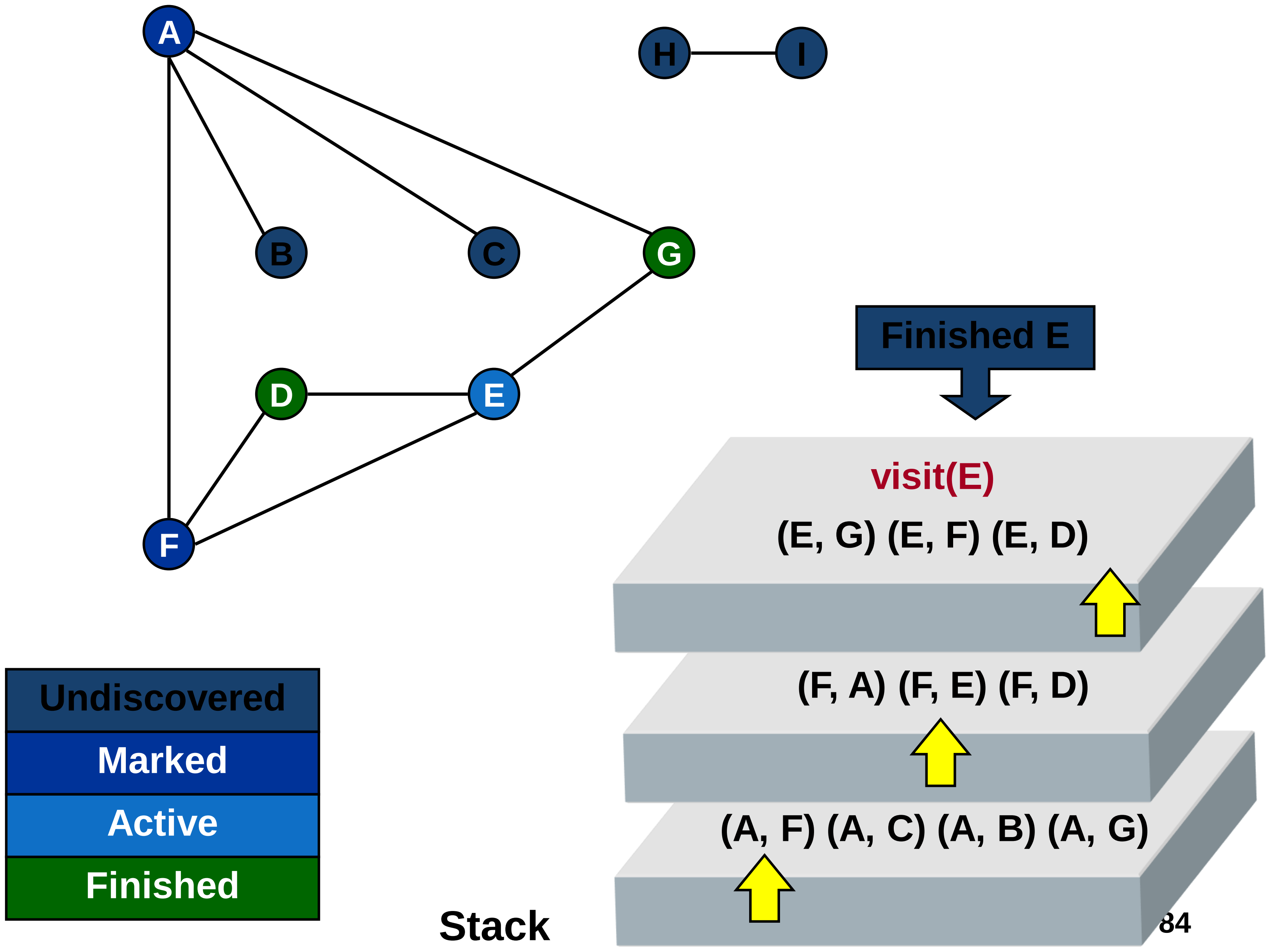
Active

Finished

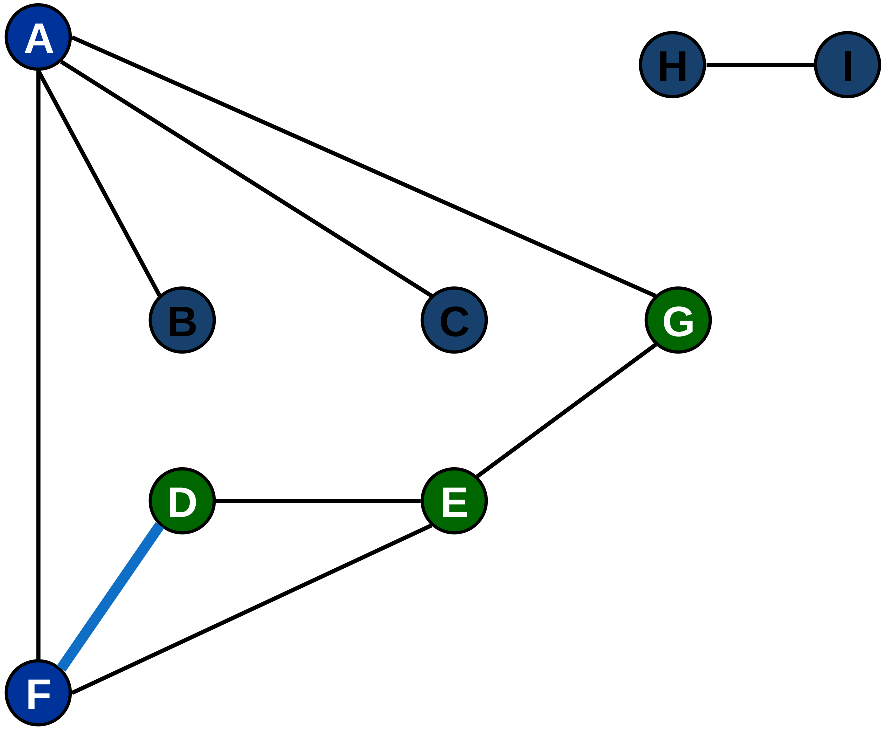
Stack



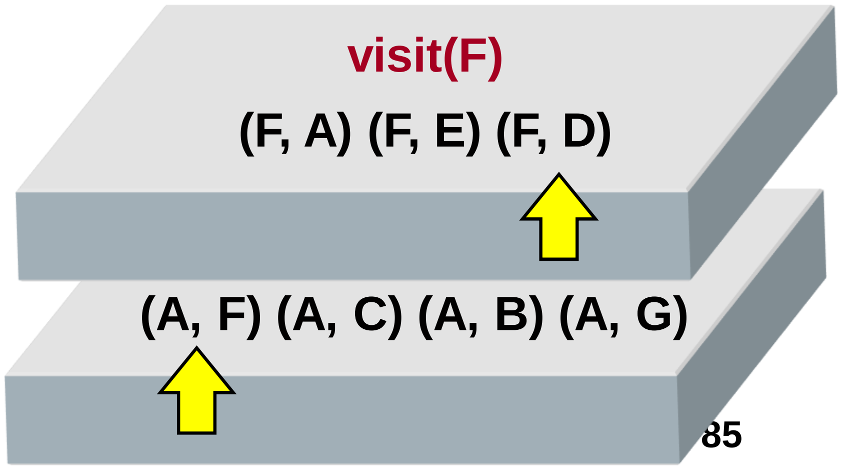

A
H
I
B
C
G
D
E
F
Finished D
visit(D)
(D, F) (D, E)
(E, G) (E, F) (E, D)
(F, A) (F, E) (F, D)
(A, F) (A, C) (A, B) (A, G)
Undiscovered
Marked
Active
Finished
Stack


A
H
I
B
C
G
D
E
F
Finished E
visit(E)
(E, G) (E, F) (E, D)
(F, A) (F, E) (F, D)
(A, F) (A, C) (A, B) (A, G)
Undiscovered
Marked
Active
Finished
Stack

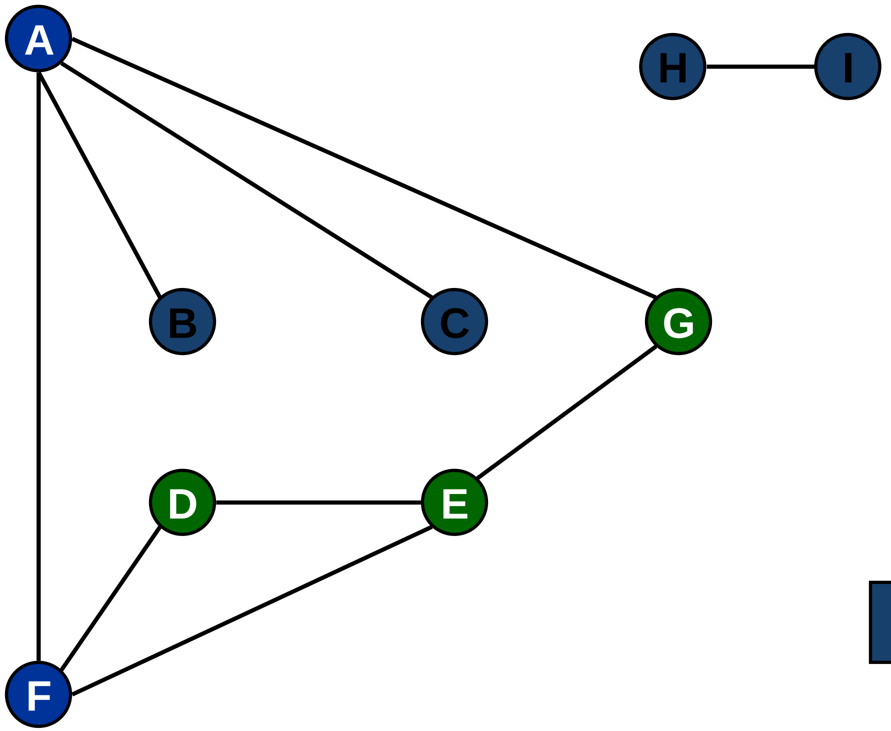
D already marked



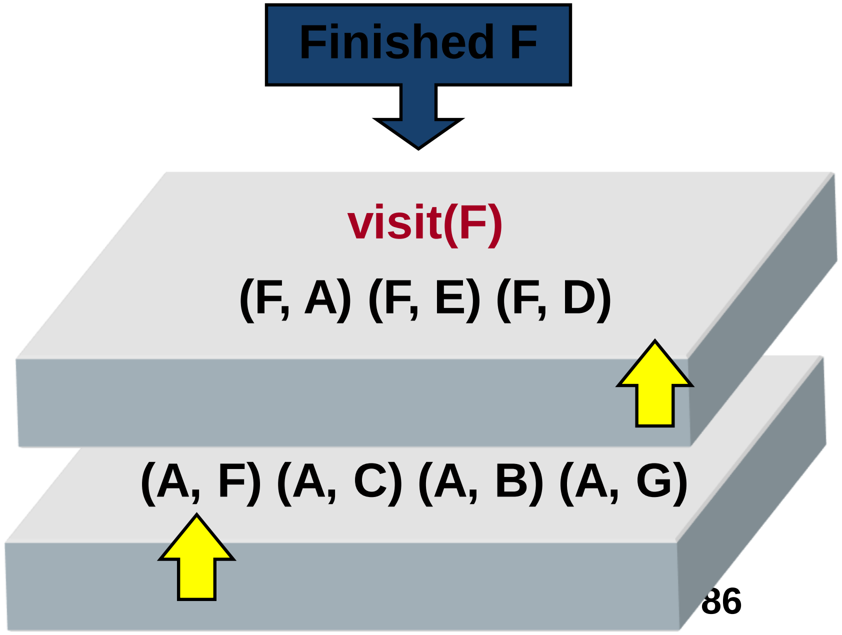
Undiscovered
Marked
Active
Finished



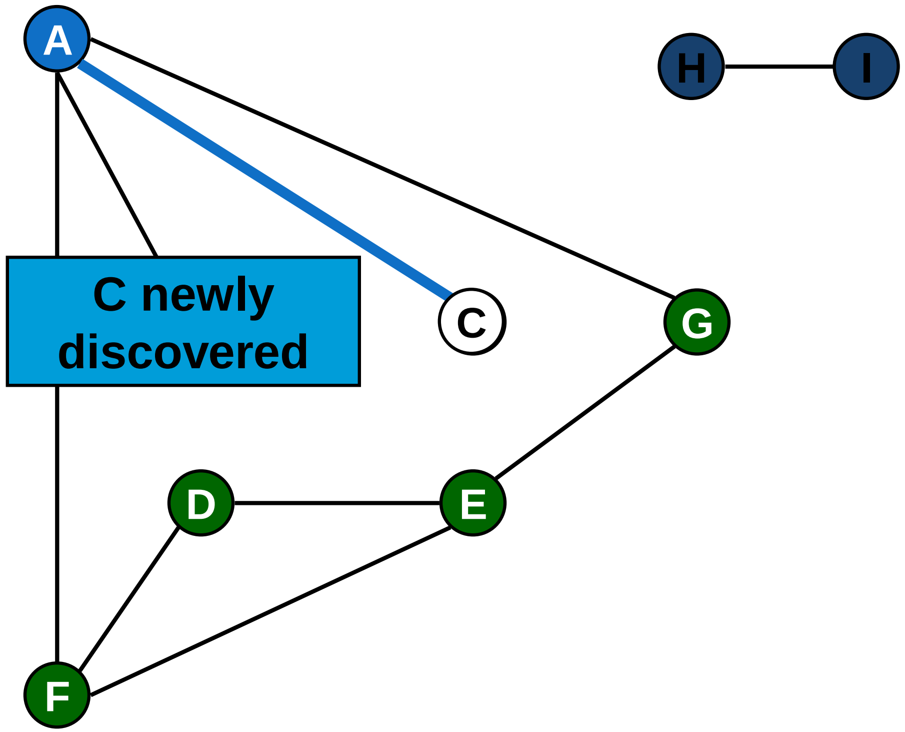
Stack



Undiscovered
Marked
Active
Finished

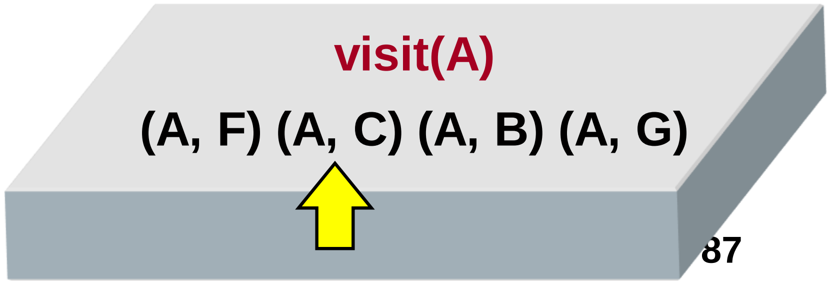


Stack

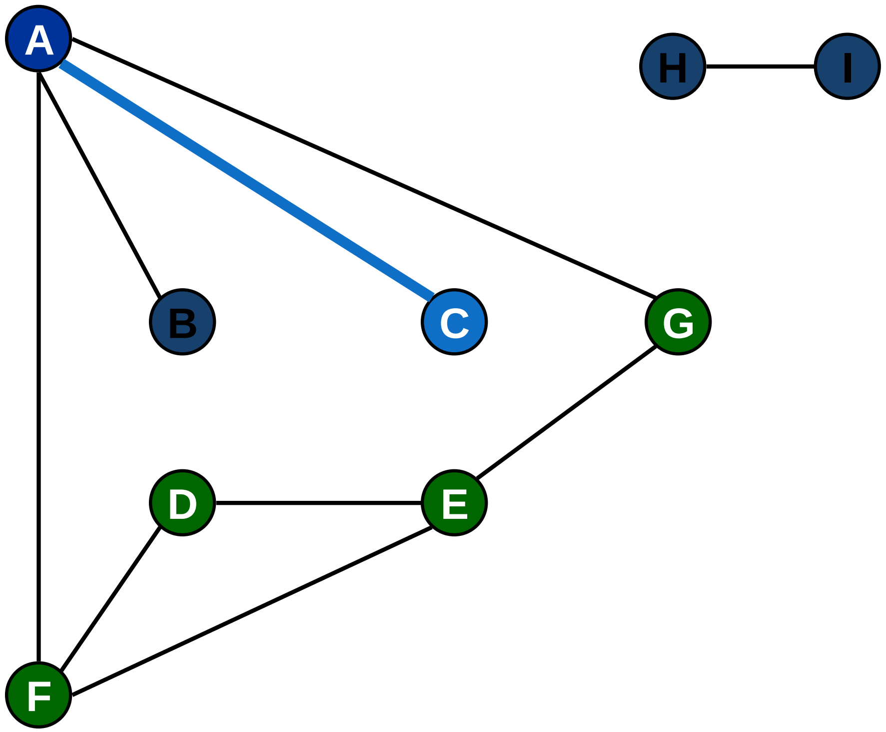


Undiscovered
Marked
Active
Finished

Stack

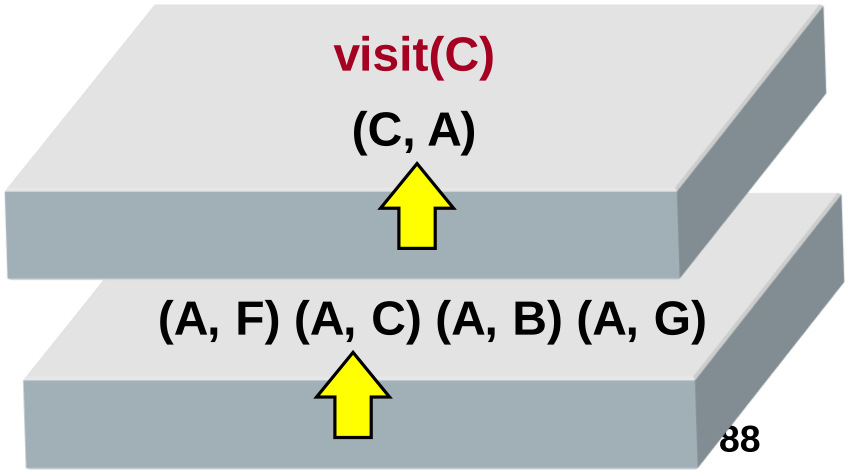


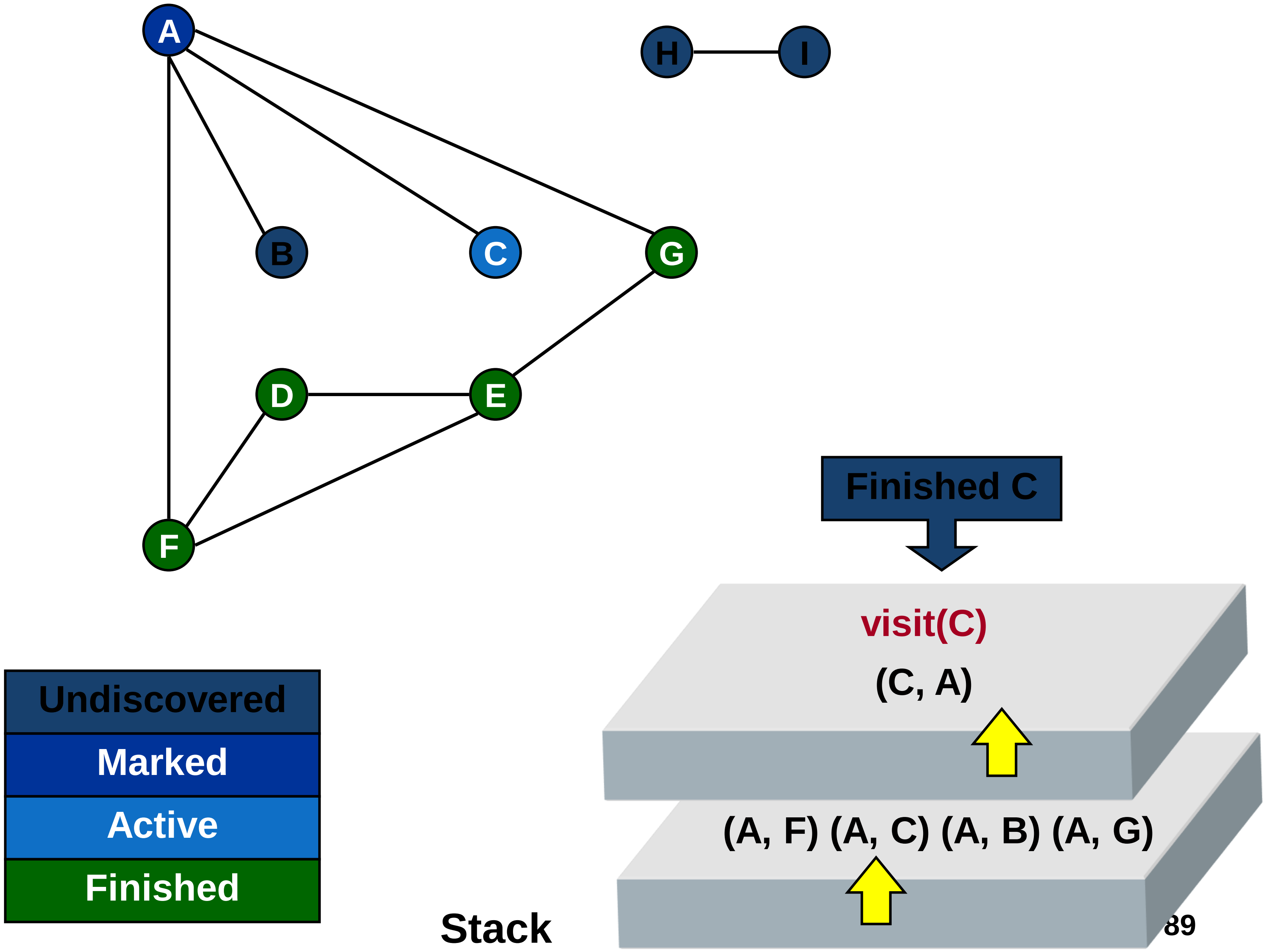
A already marked

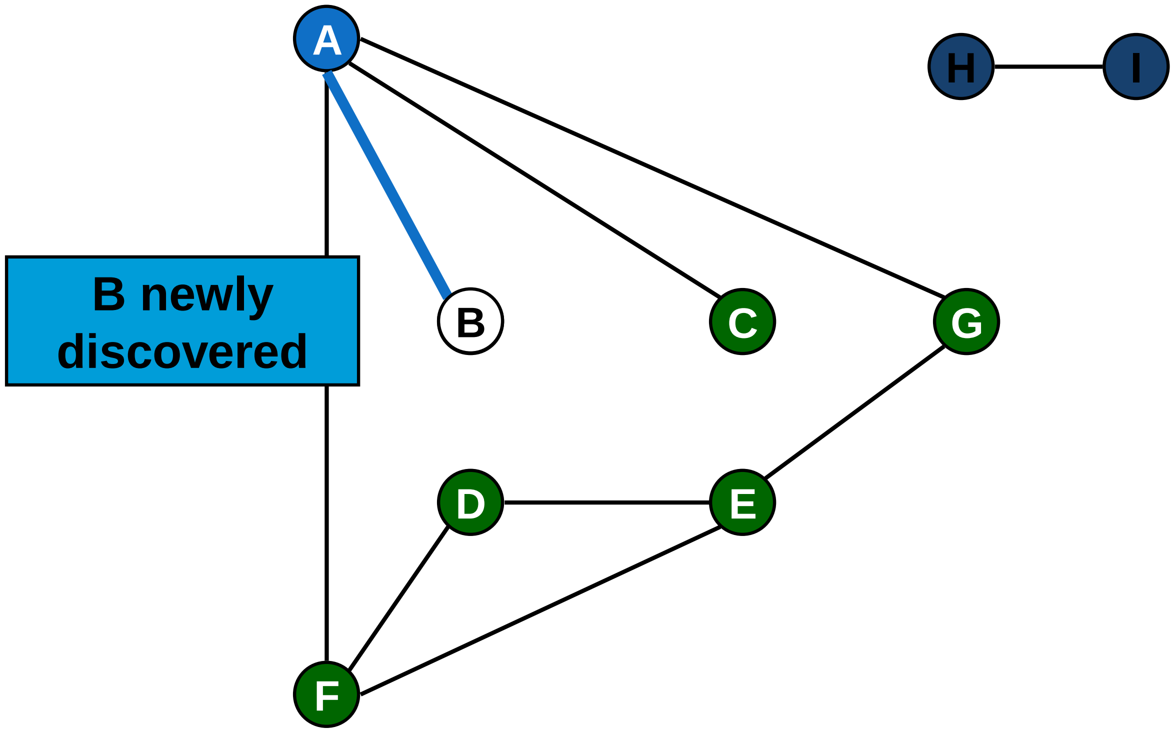


Undiscovered
Marked
Active
Finished

Stack




A
H
I
B
C
G
D
E
F
Finished C
visit(C)
(C, A)
(A, F) (A, C) (A, B) (A, G)
Undiscovered
Marked
Active
Finished
Stack

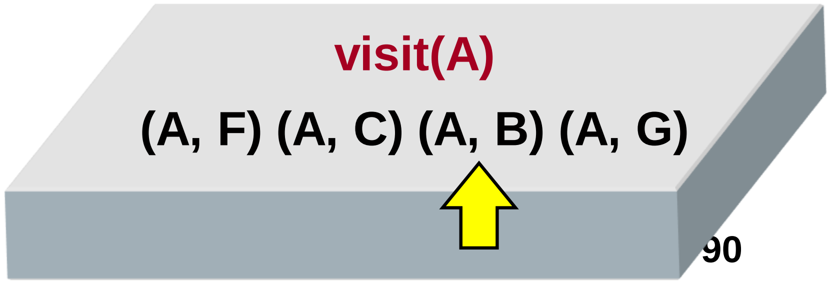


Undiscovered

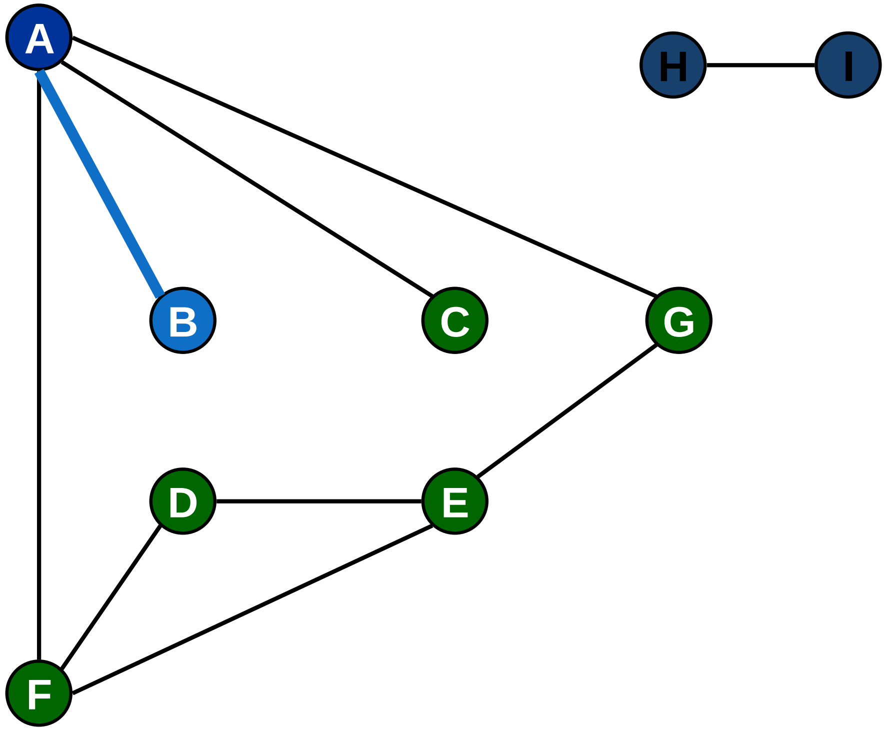
Marked

Active

Finished

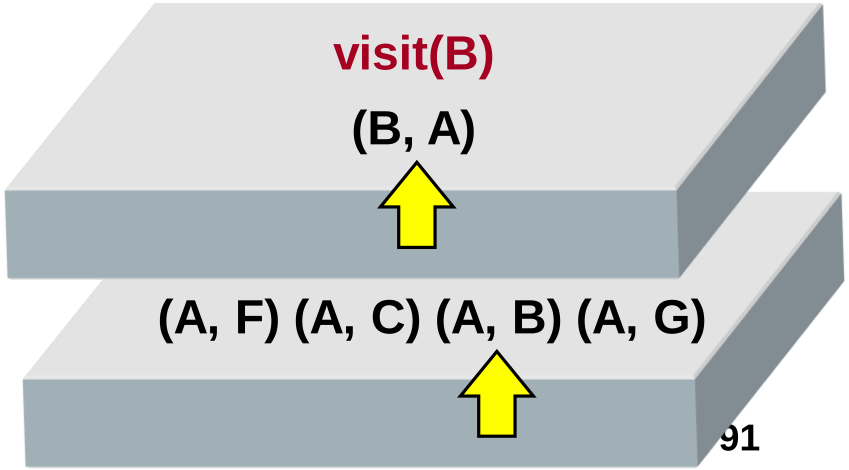


Stack

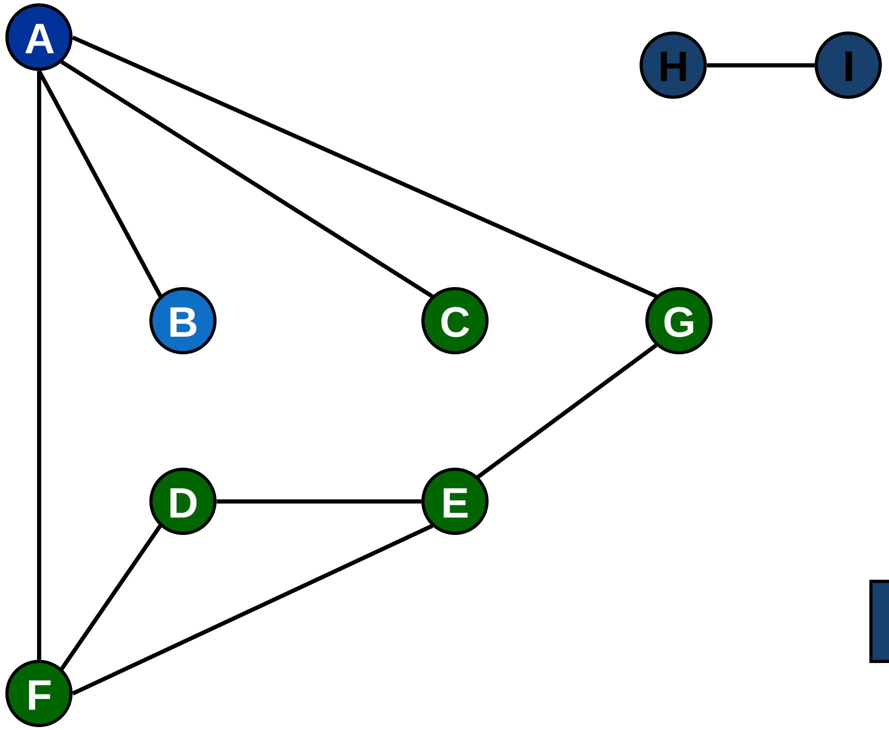


A already marked

Undiscovered
Marked
Active
Finished



Stack

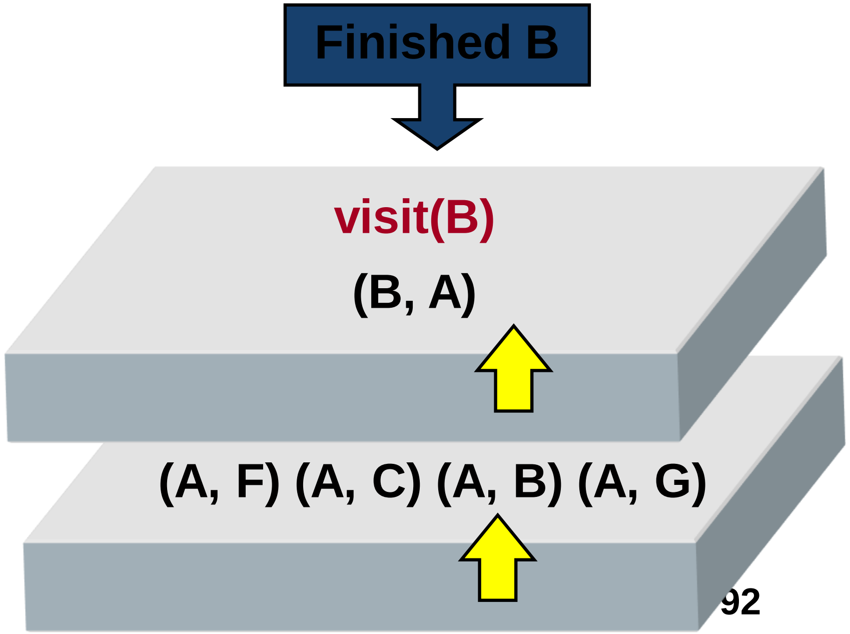


Undiscovered

Marked

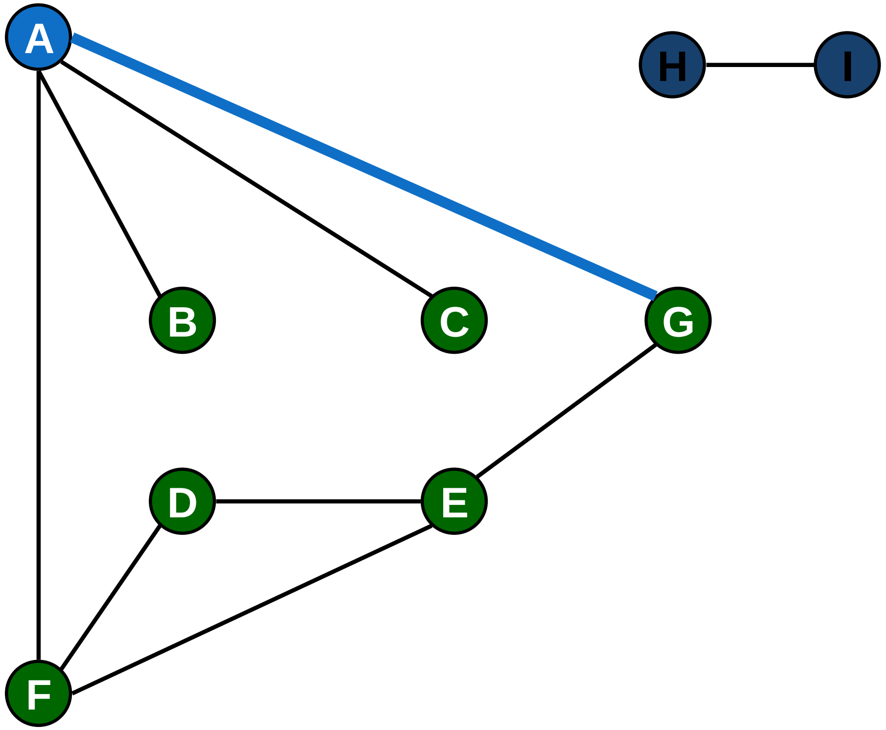
Active

Finished



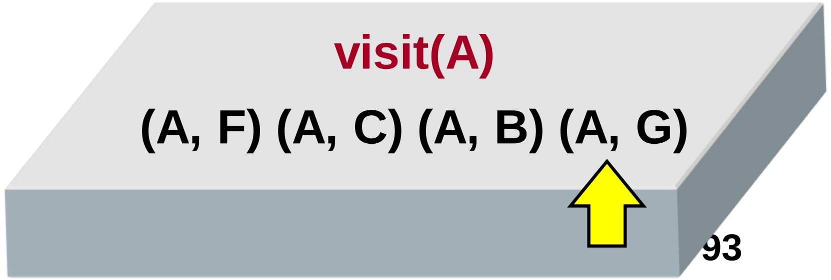
Stack

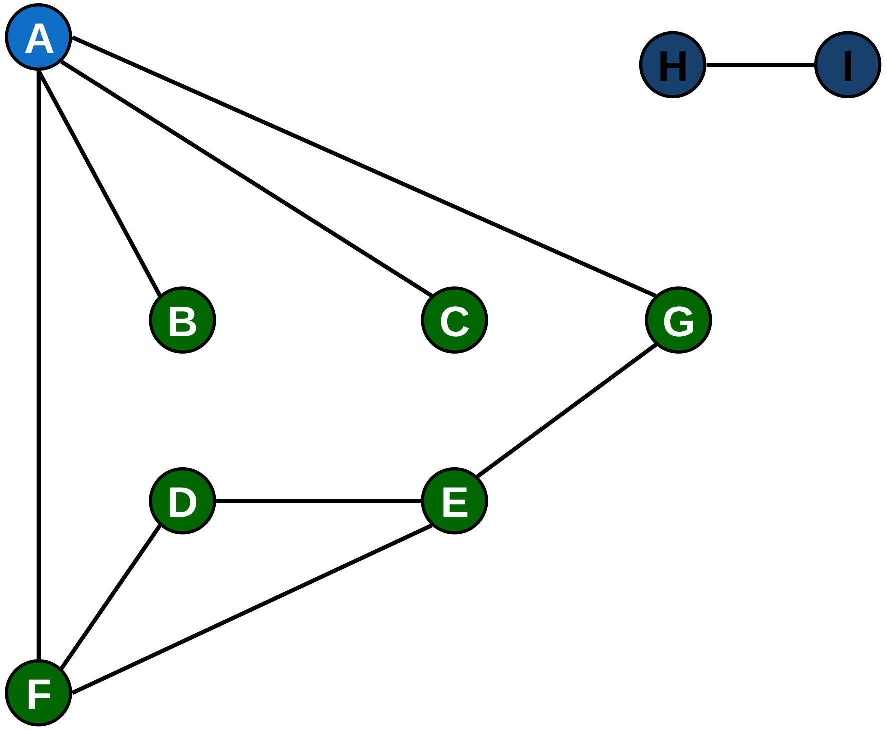
G already finished



Undiscovered
Marked
Active
Finished

Stack





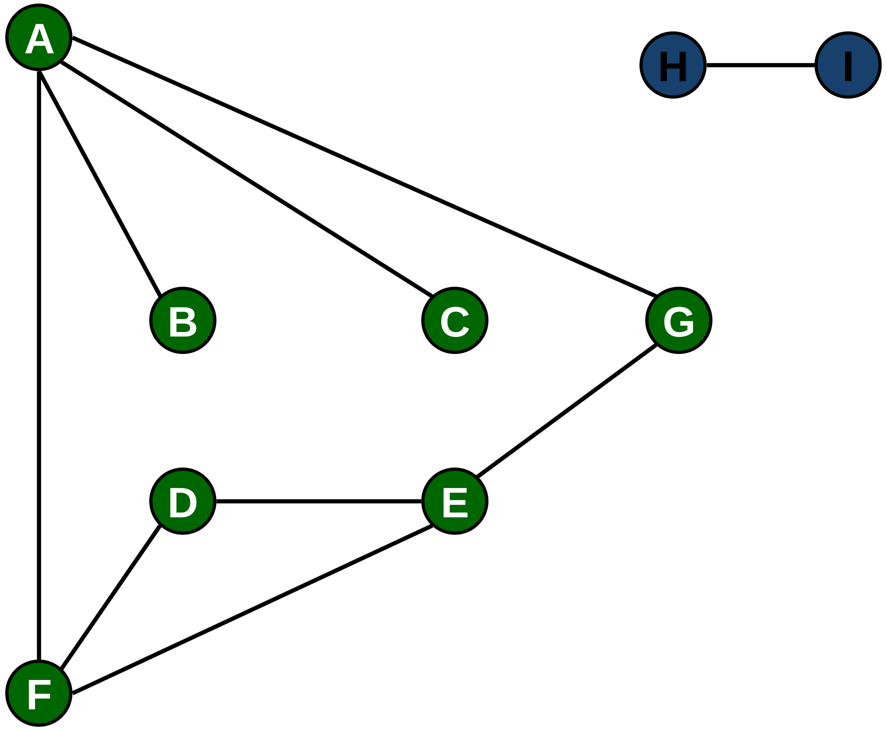
Finished A

visit(A)

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

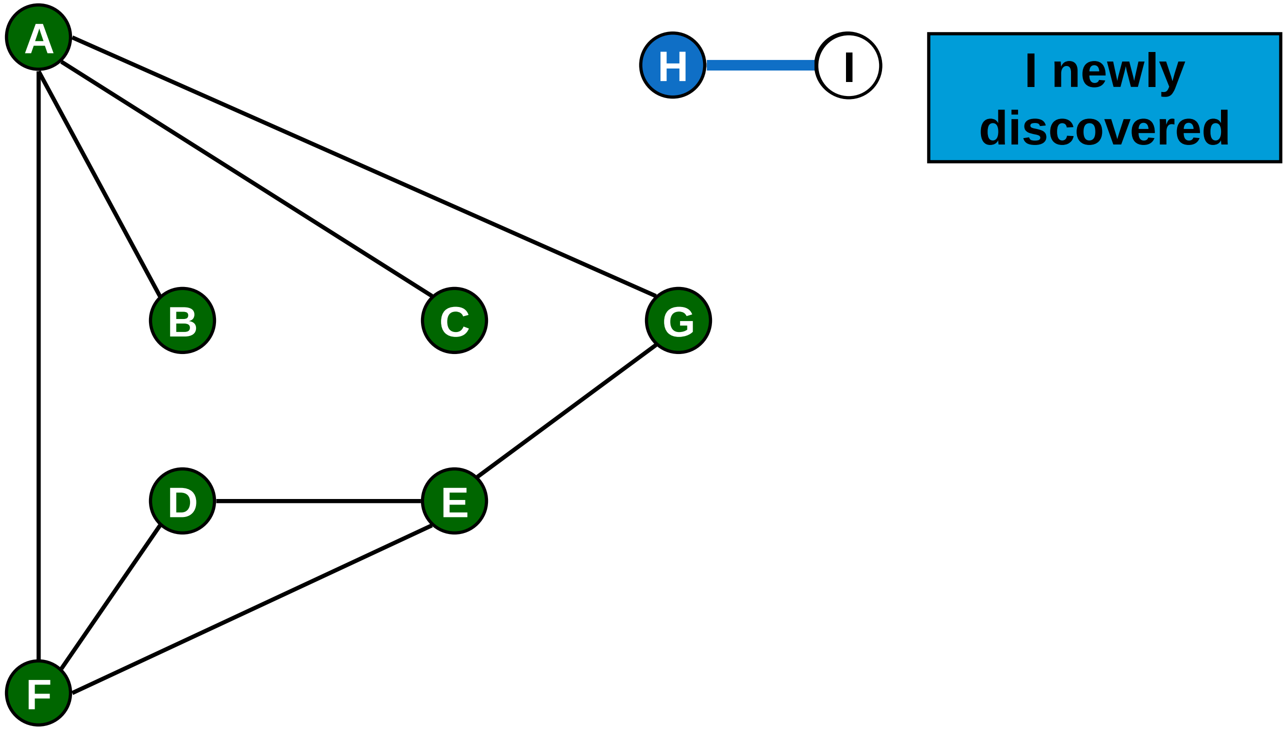
Stack

Undiscovered
Marked
Active
Finished

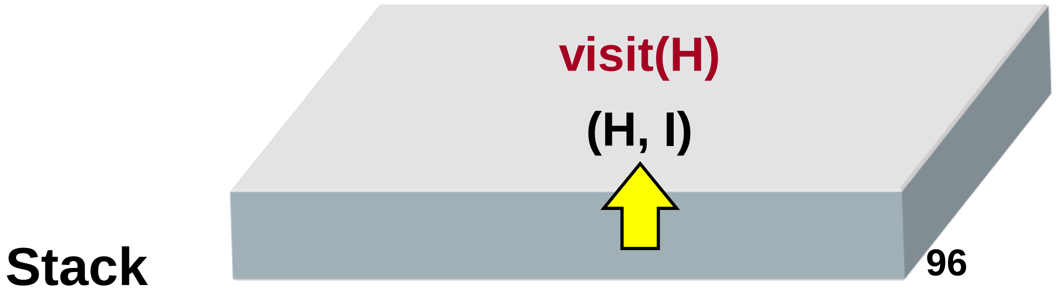


Undiscovered
Marked
Active
Finished

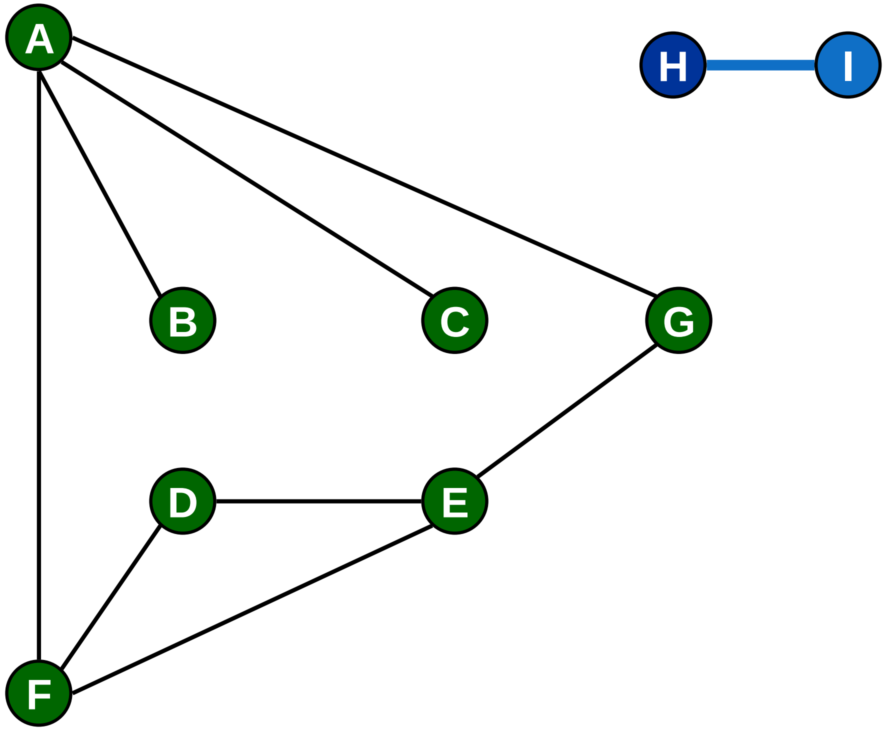
Stack


A
B
C
D
E
F
G
H
I
I newly discovered


Undiscovered
Marked
Active
Finished


visit(H)
(H, I)
Stack

H already marked



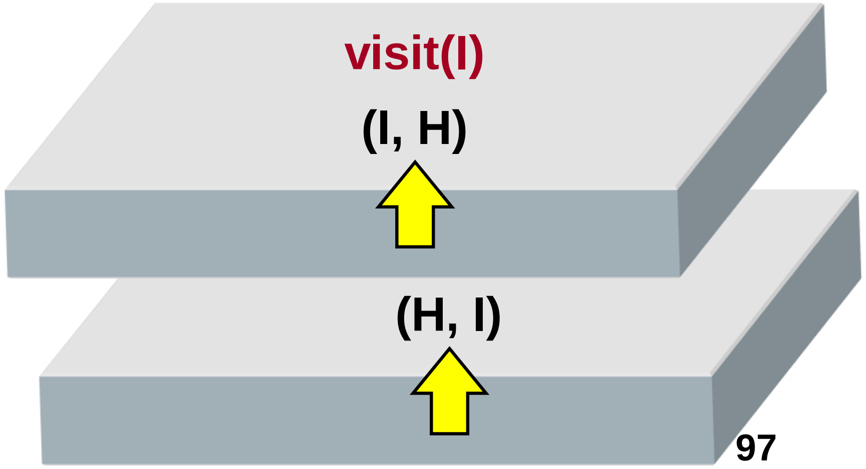
Undiscovered

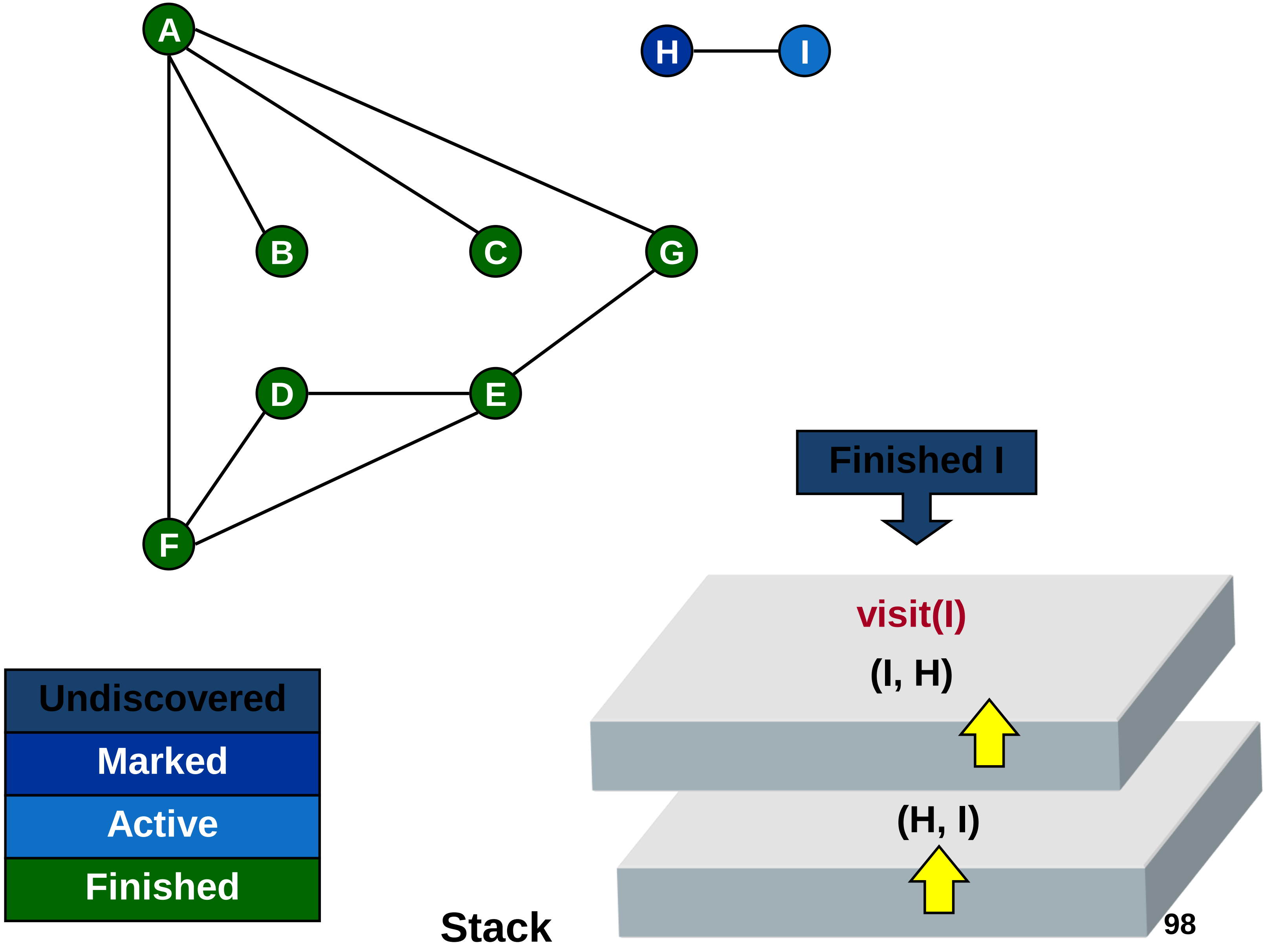
Marked

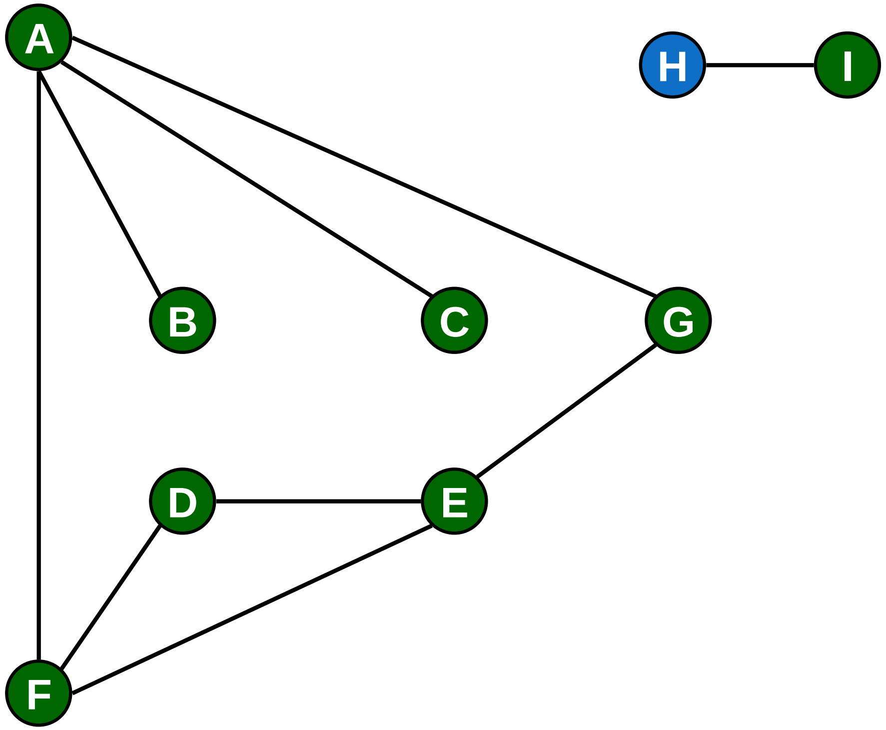
Active

Finished

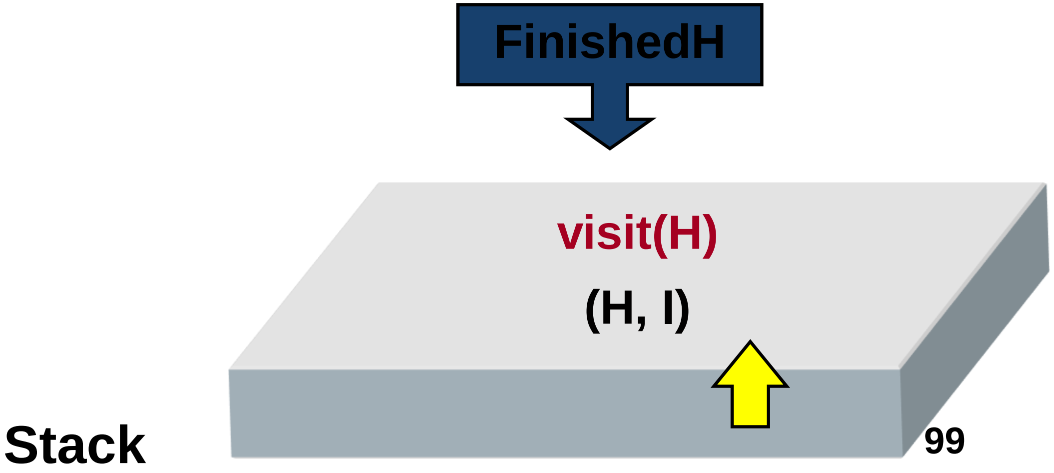
Stack

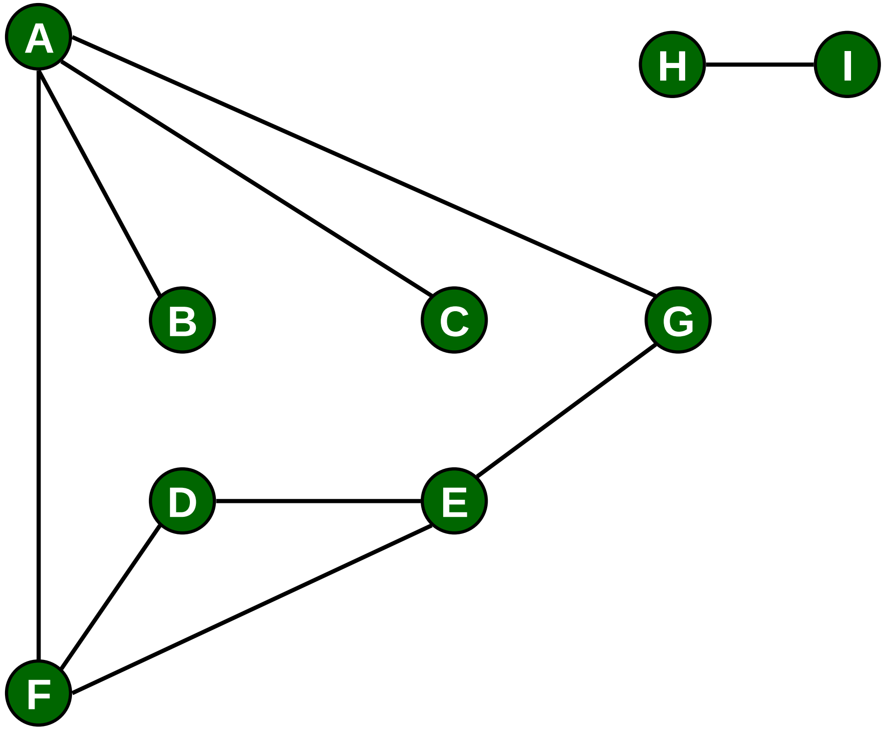



A
H
I
B
C
G
D
E
Finished I
F
visit(I)
(I, H)
Undiscovered
Marked
Active
Finished
(H, I)
Stack


A
H
I
B
C
G
D
E
F


Undiscovered
Marked
Active
Finished


FinishedH
visit(H)
(H, I)
Stack



Undiscovered
Marked
Active
Finished

Time complexity of BFS & DFS

BFS is slower than DFS. DFS is faster than BFS. Time Complexity of BFS = O(V+E) where V is vertices and E is edges. Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.