APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS

KTU ASSIST

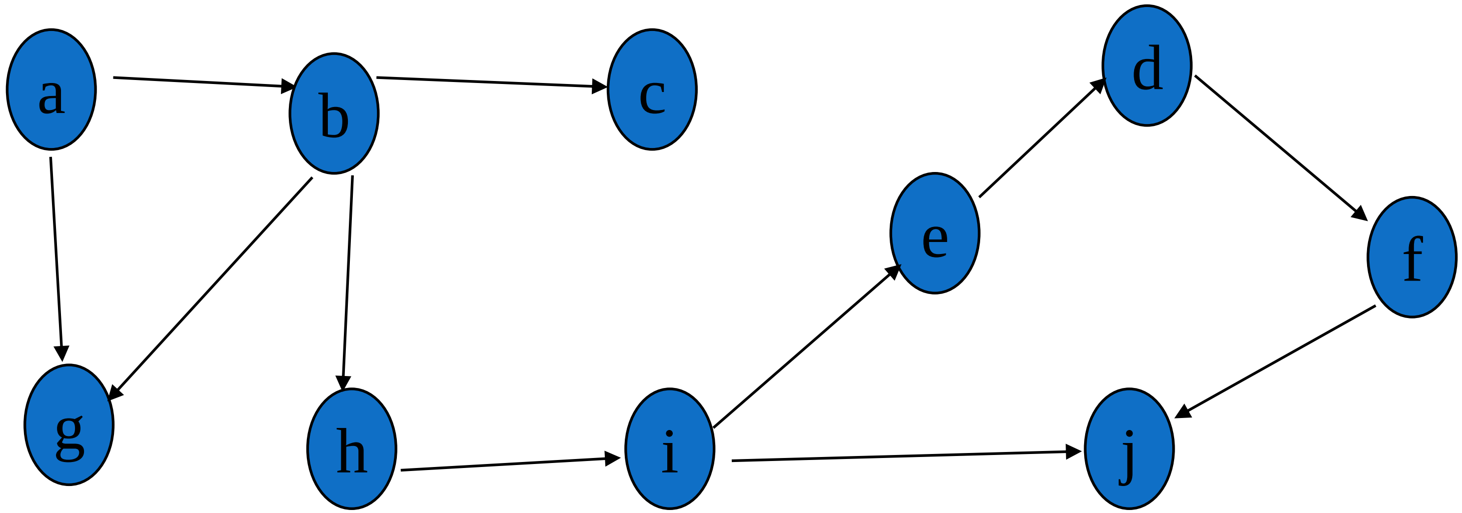
Depth First Search & Breadth First Search

Graph Search (traversal)

- Graph traversal : searching a vertex in a graph
- There are two graph traversal techniques and they are as follows
  - **DFS (Depth First Search)**
  - **BFS (Breadth First Search)**
- DFS - go as far as possible along a single path until reach a dead end (a vertex with no edge out or no neighbor unexplored) then backtrack
- BFS - one explore a graph level by level away (explore all neighbors first and then move on)

Depth-First Search (DFS)

- The basic idea behind this algorithm is that it traverses the graph using recursion
  - Go as far as possible until you reach a deadend
  - Backtrack to the previous path and try the next branch
  - The graph below, started at node a, would be visited in the following order: a, b, c, g, h, i, e, d, f, j



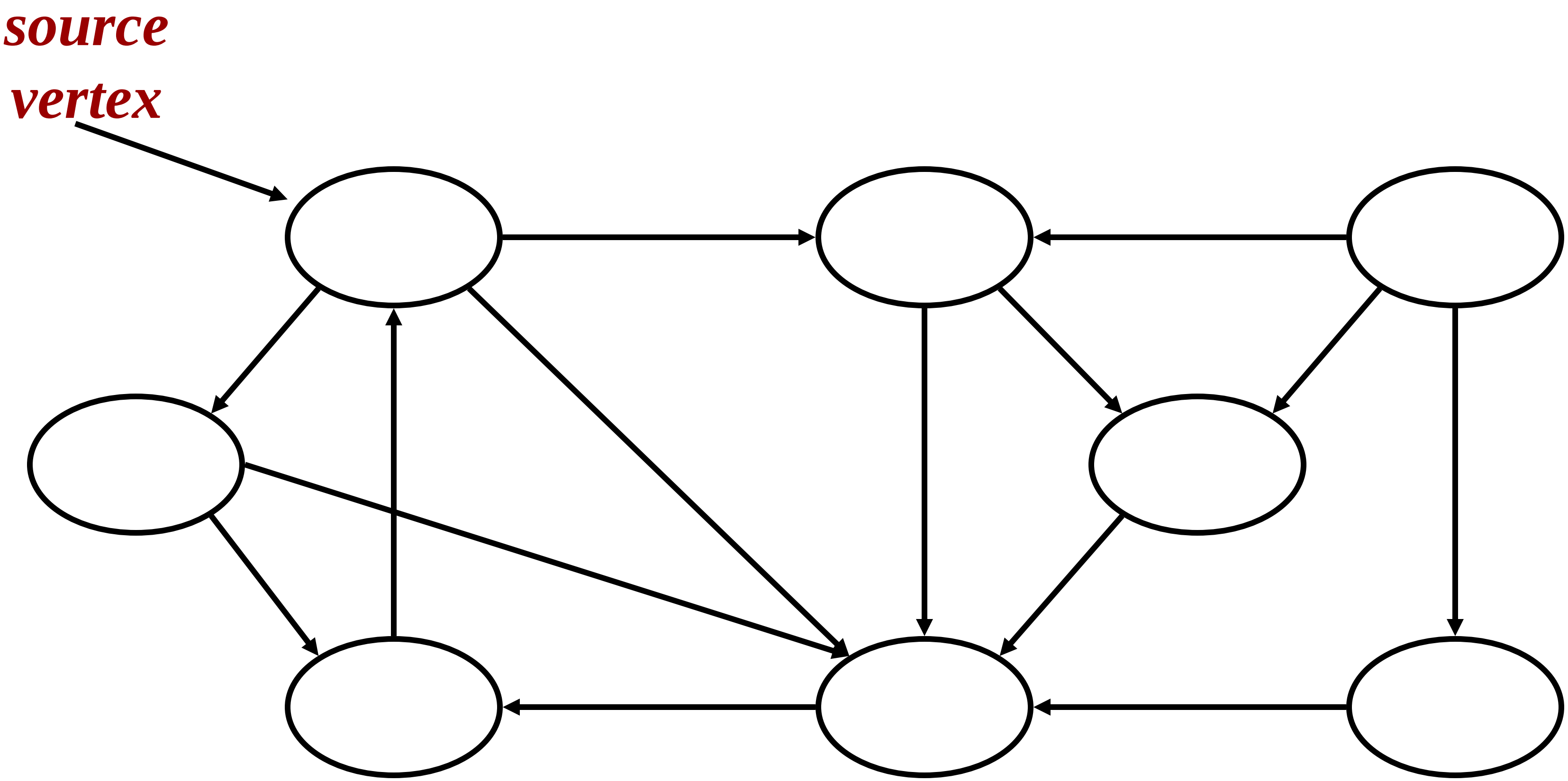
DFS: Color Scheme

- Vertices initially colored white
- Then colored gray when discovered
- Then black when finished

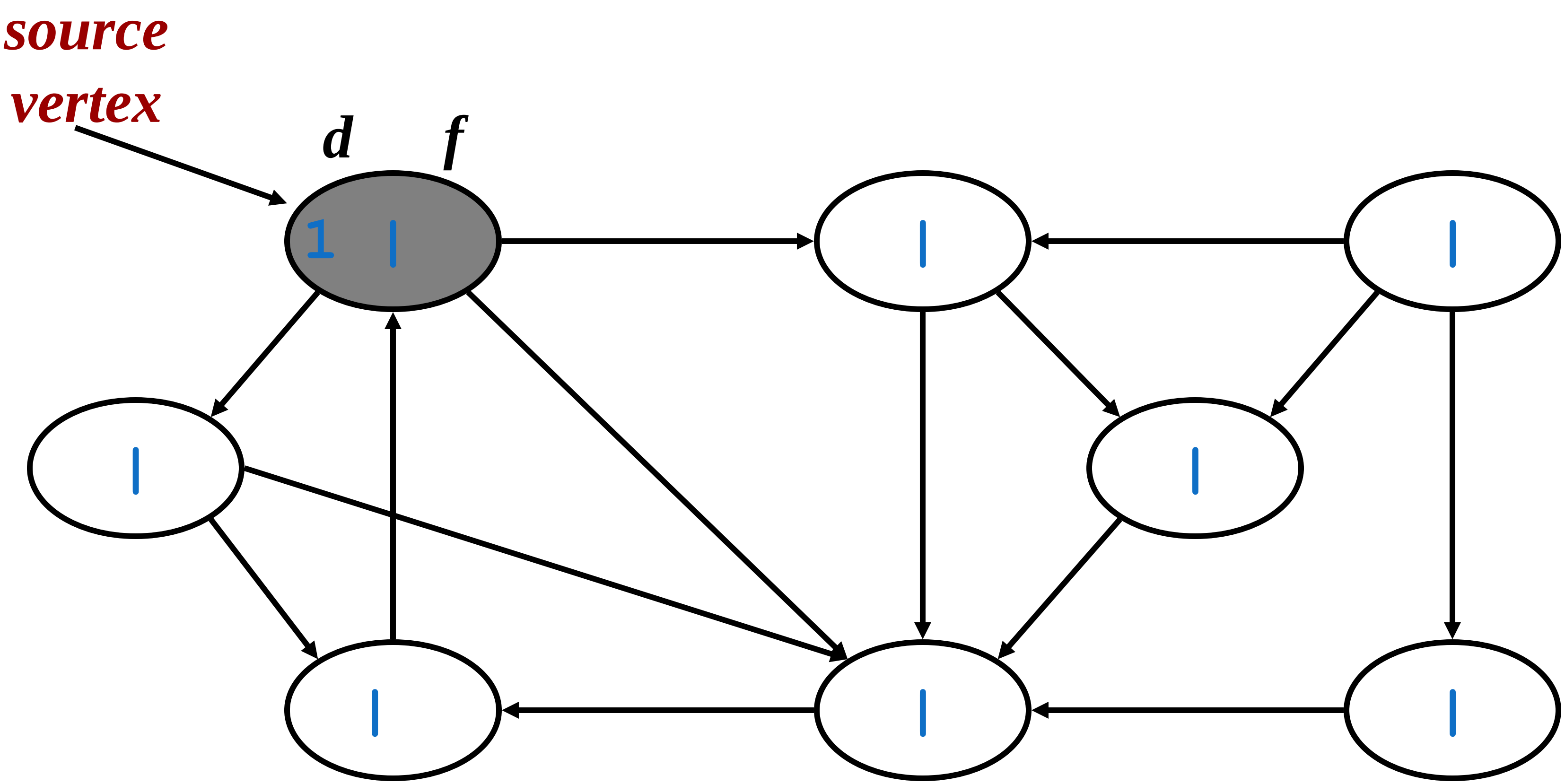
DFS: Time Stamps

- Discover time d[u]: when u is first discovered
- Finish time f[u]: when backtrack from u
- d[u] < f[u]

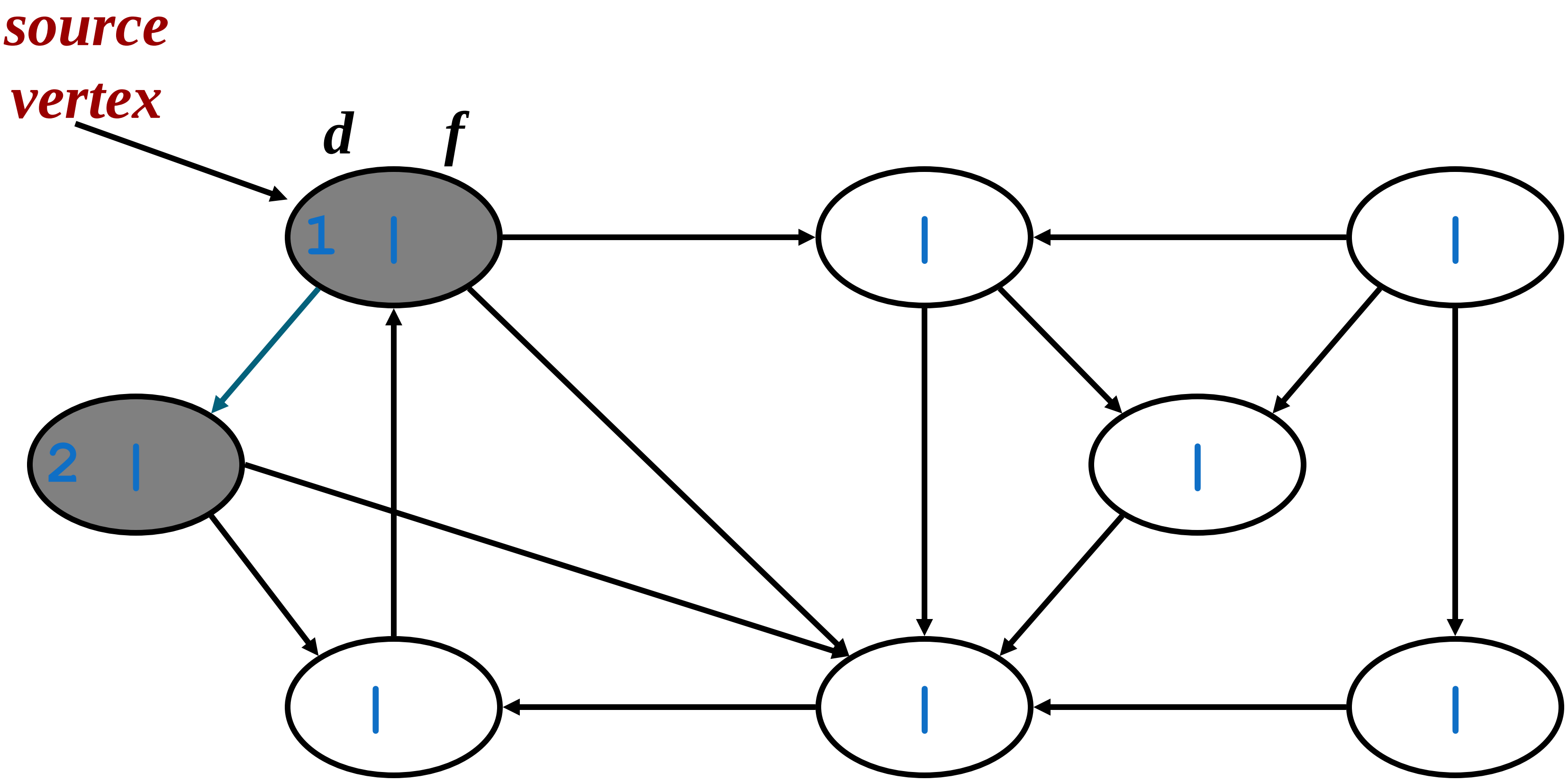
DFS Example



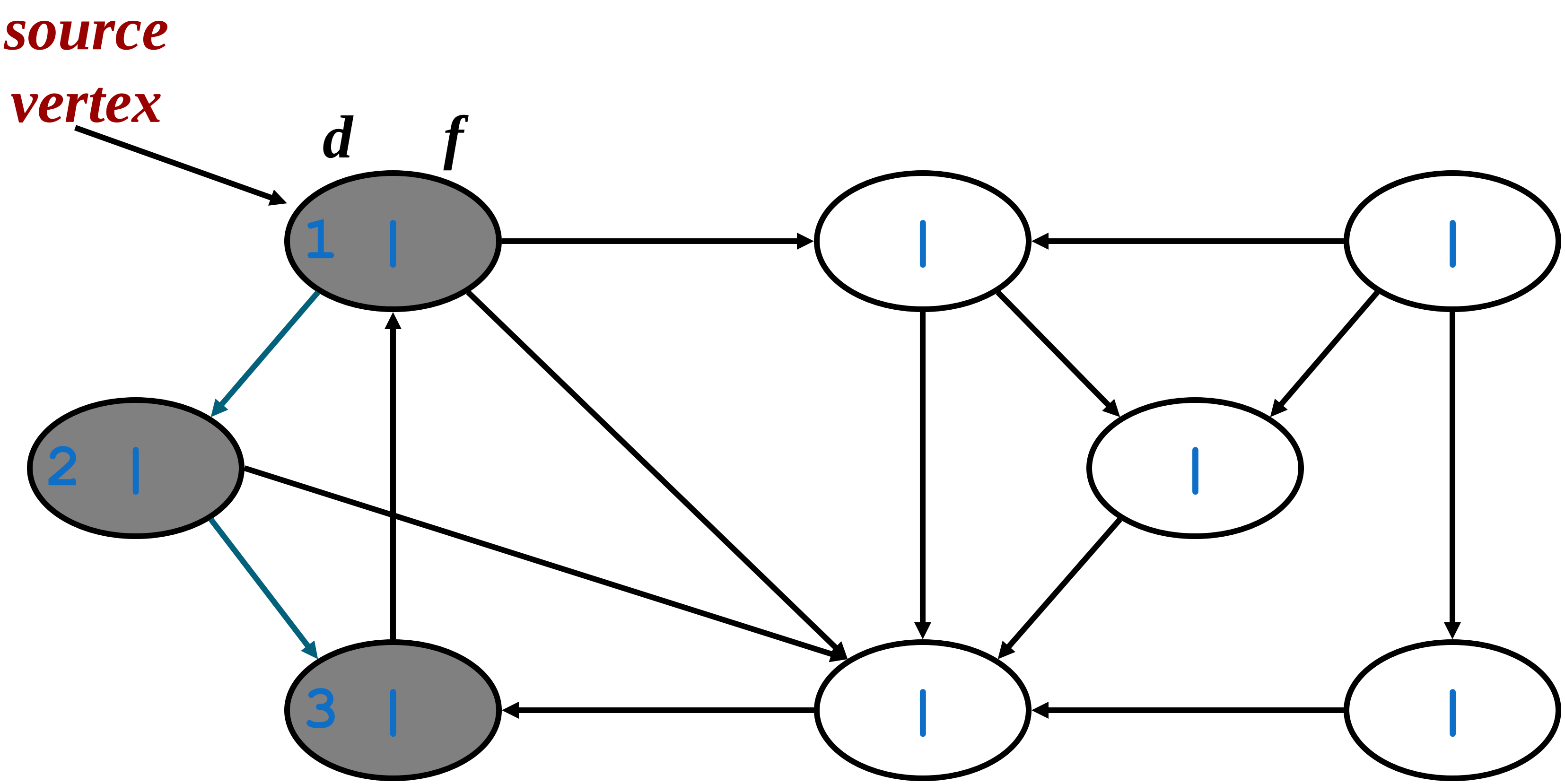
DFS Example



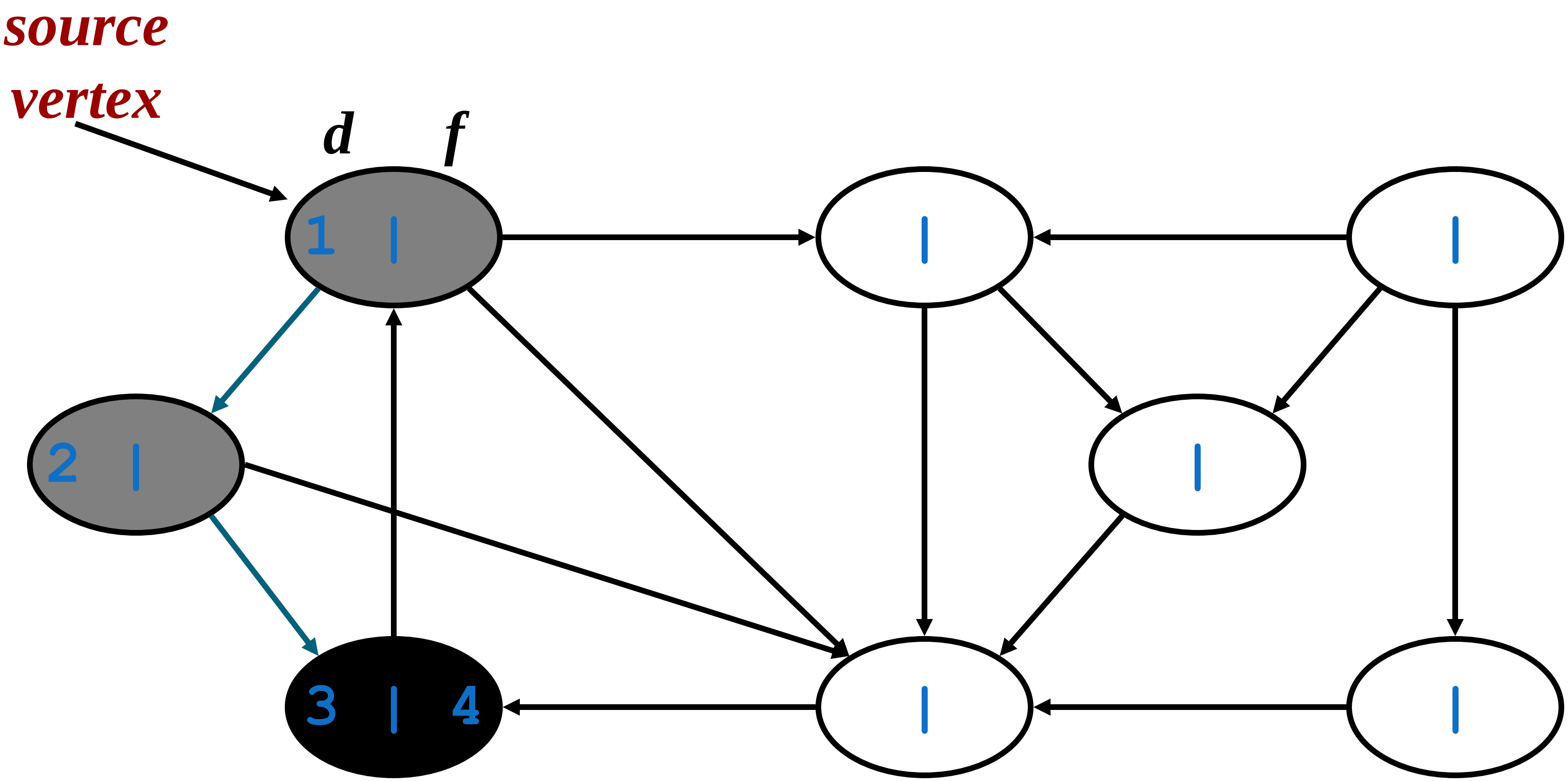
DFS Example



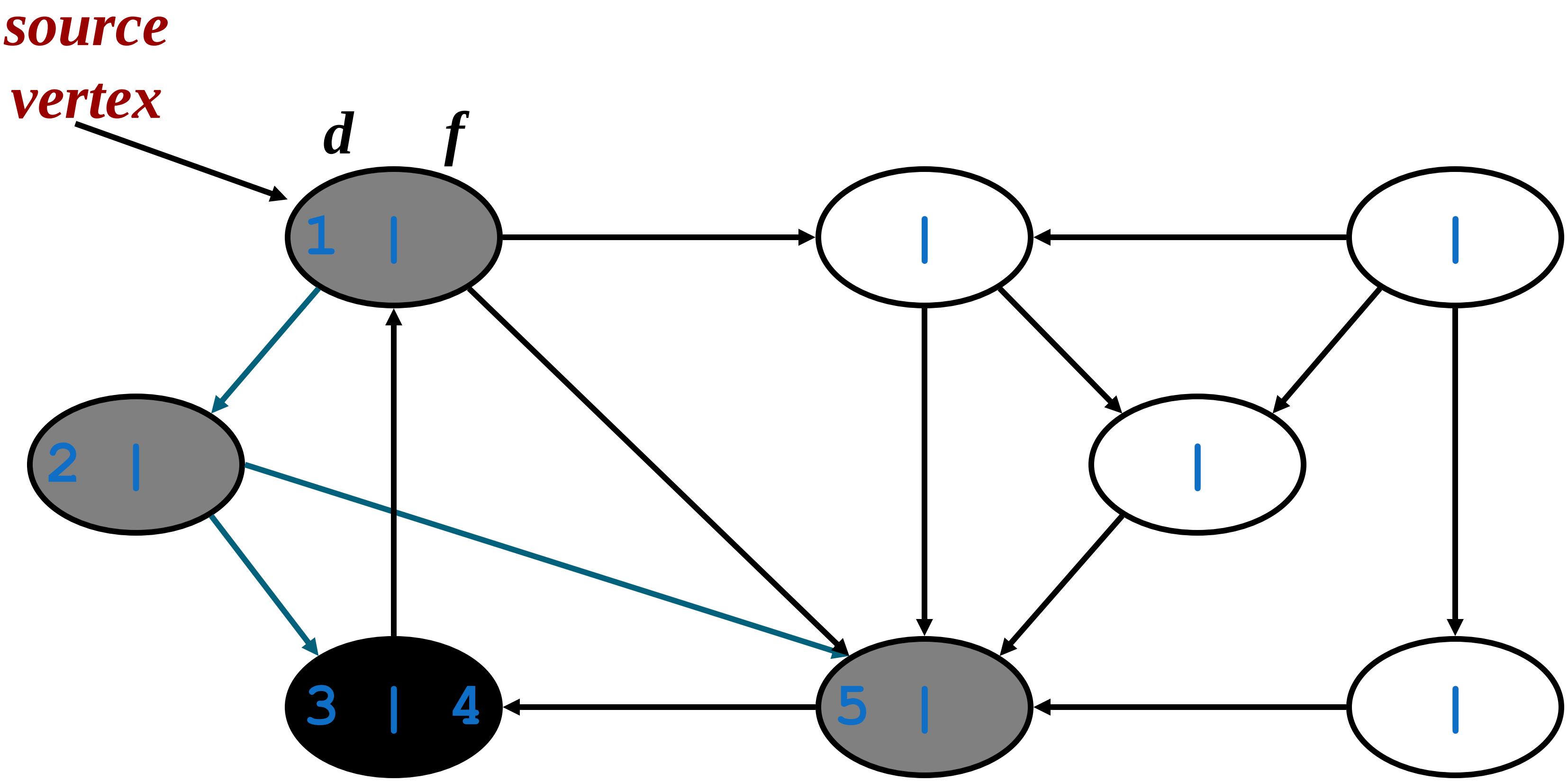
DFS Example



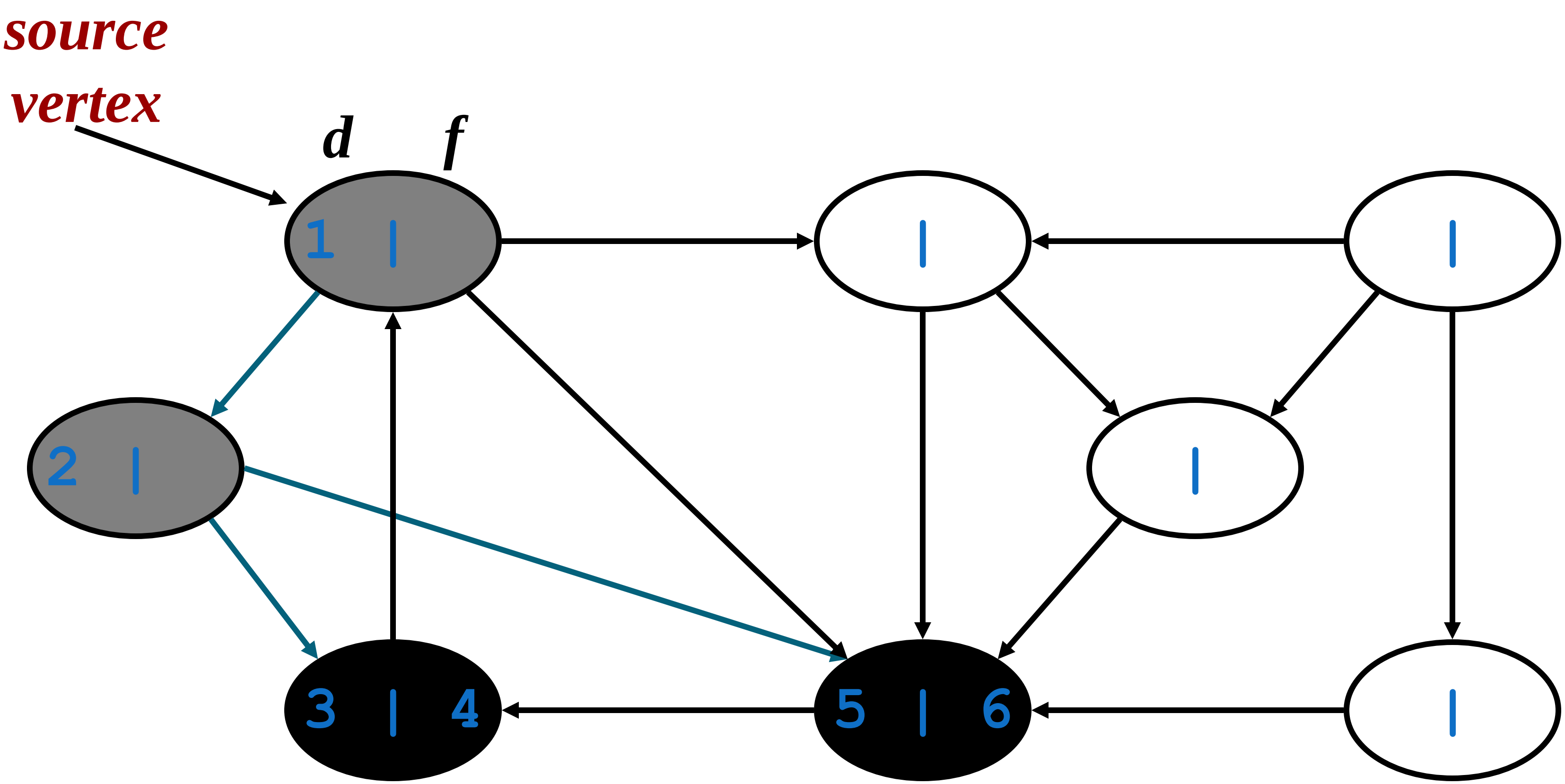
DFS Example



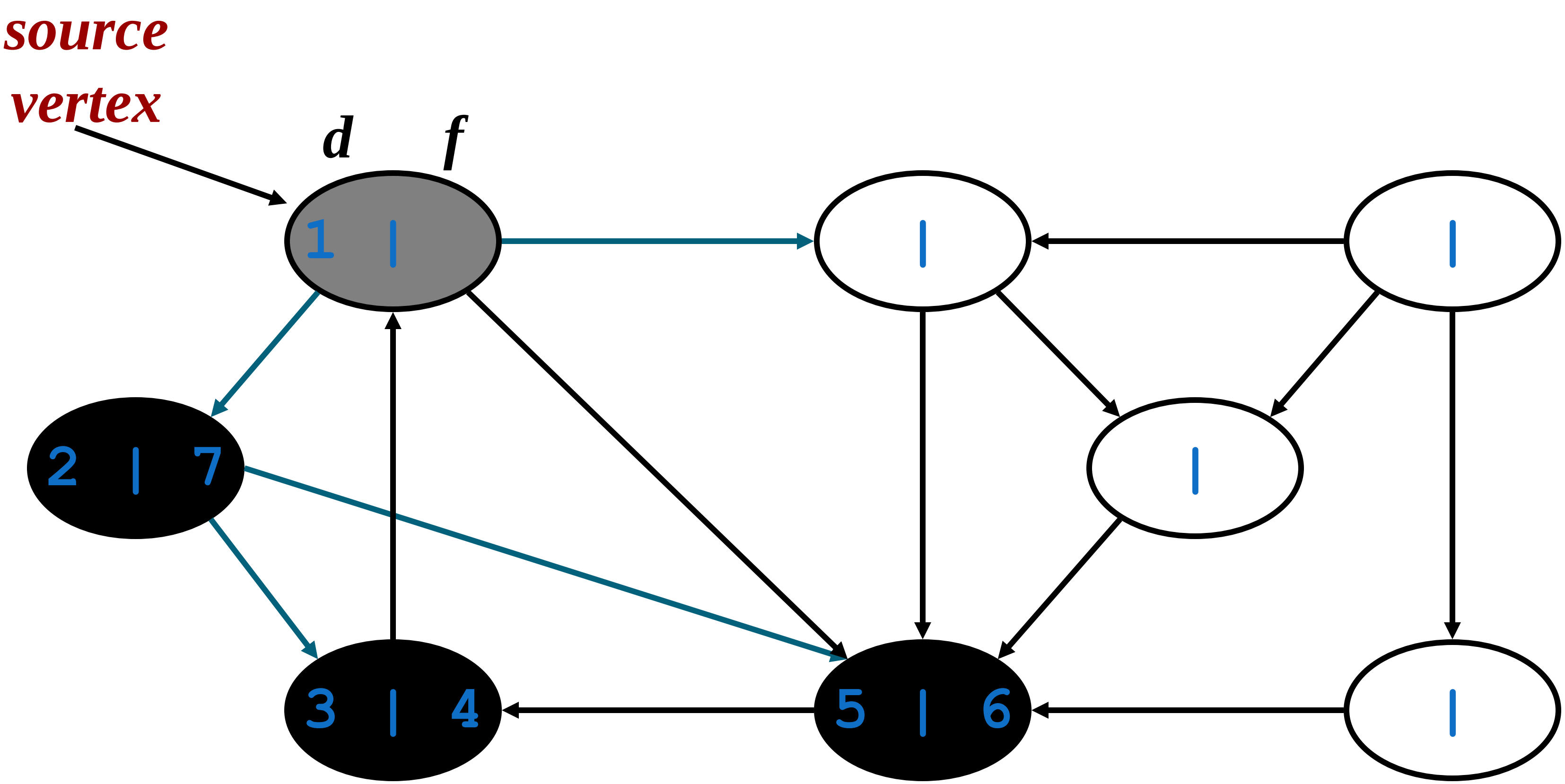
DFS Example



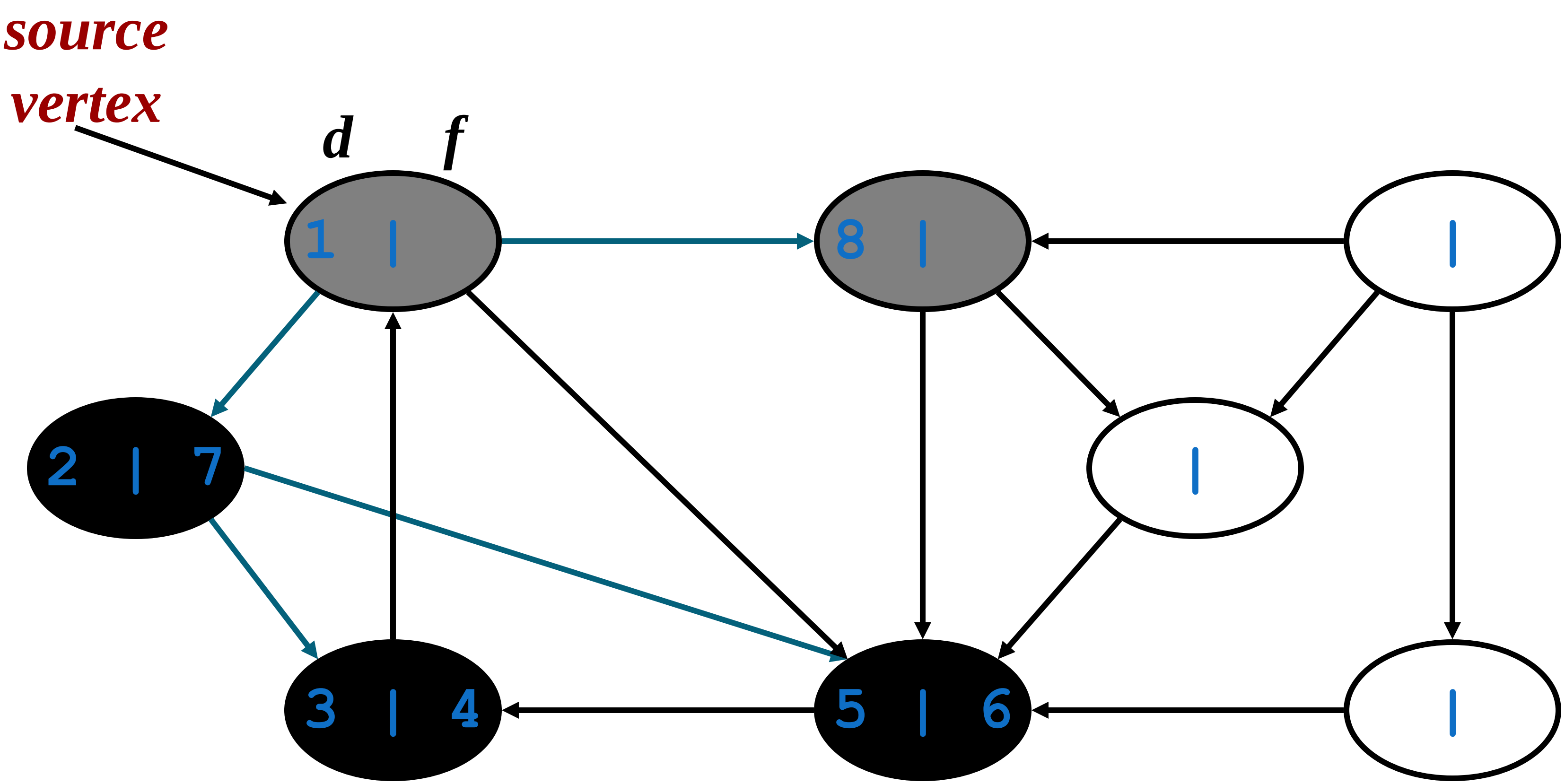
DFS Example



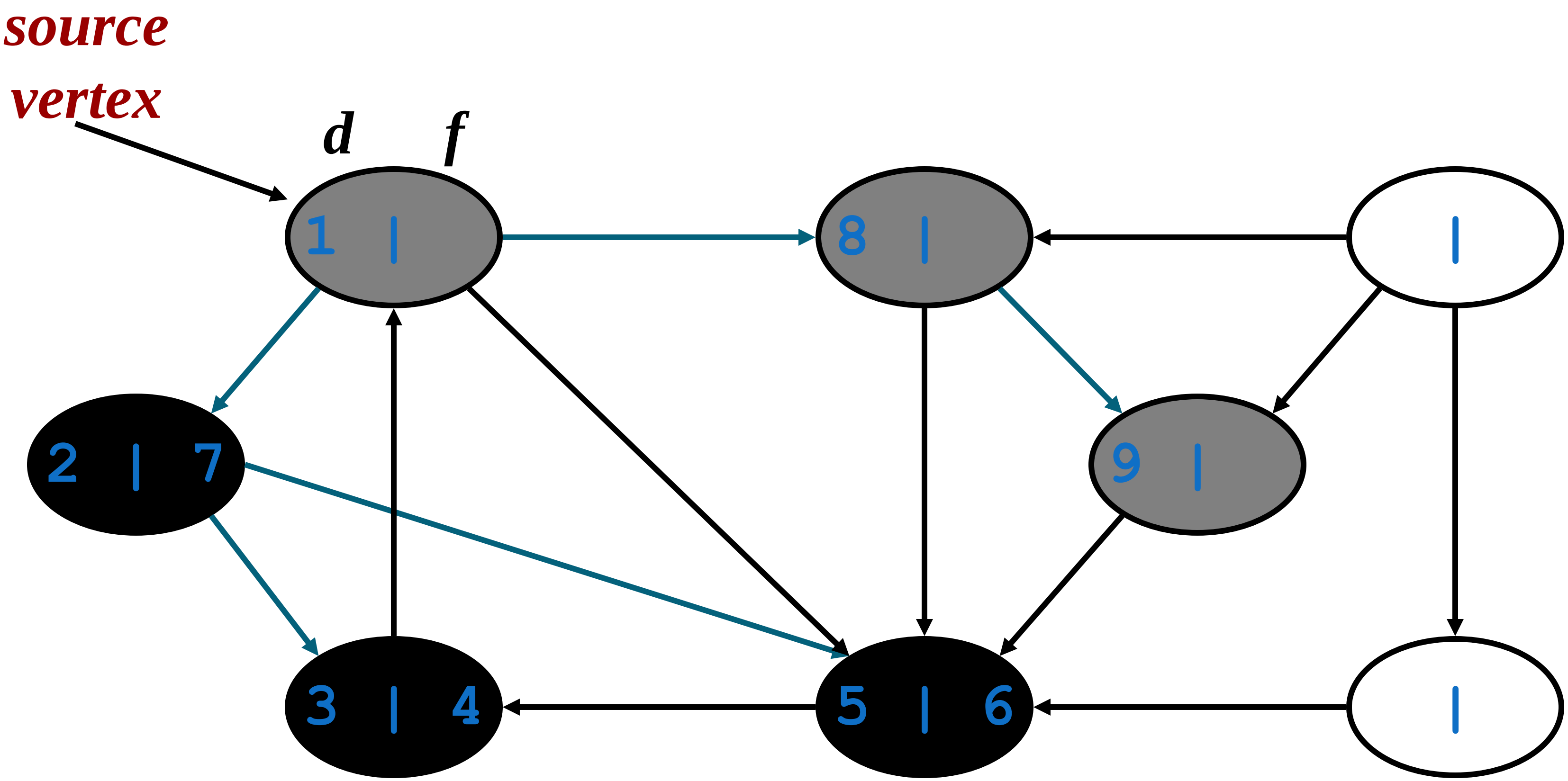
DFS Example



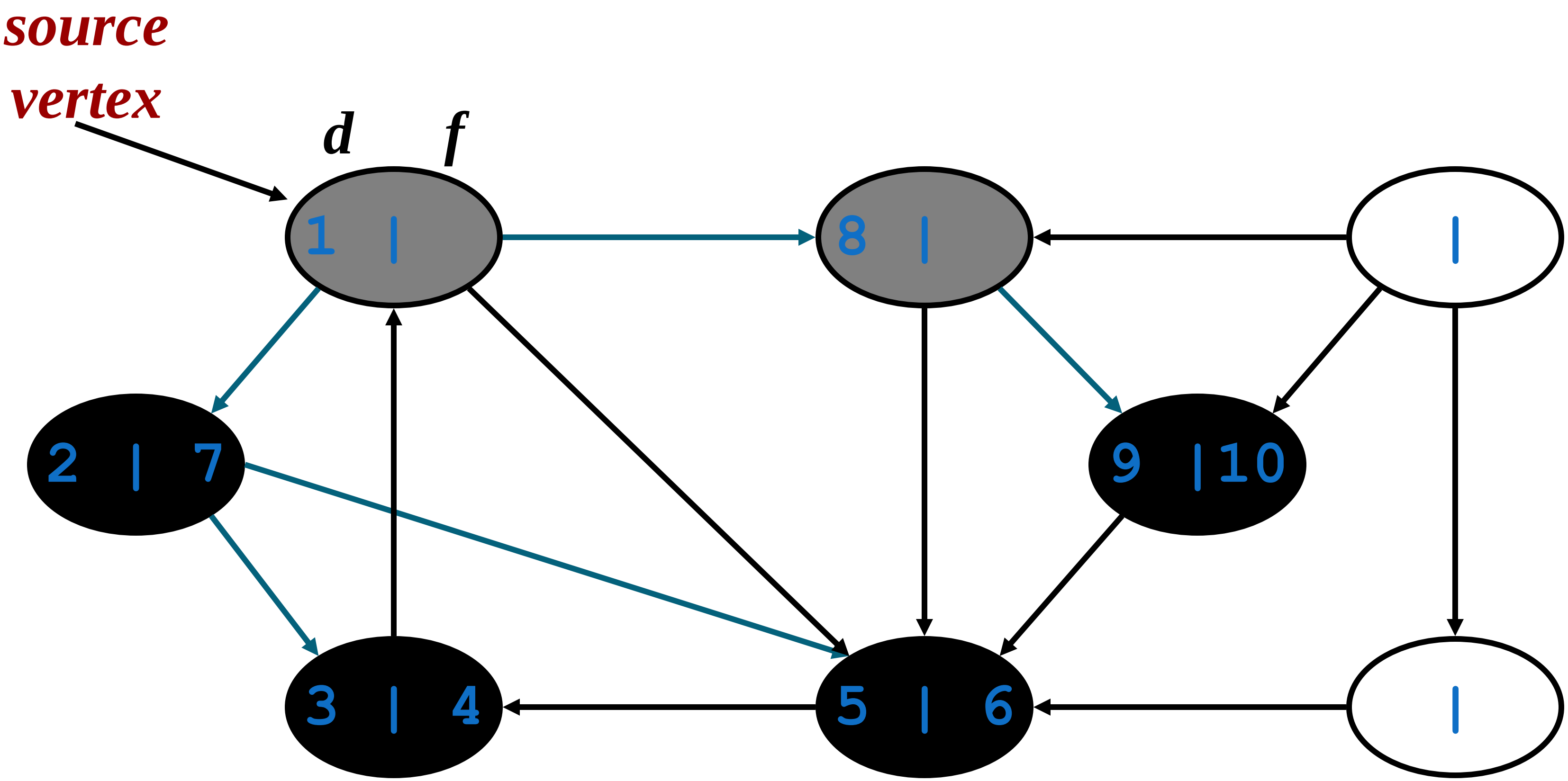
DFS Example



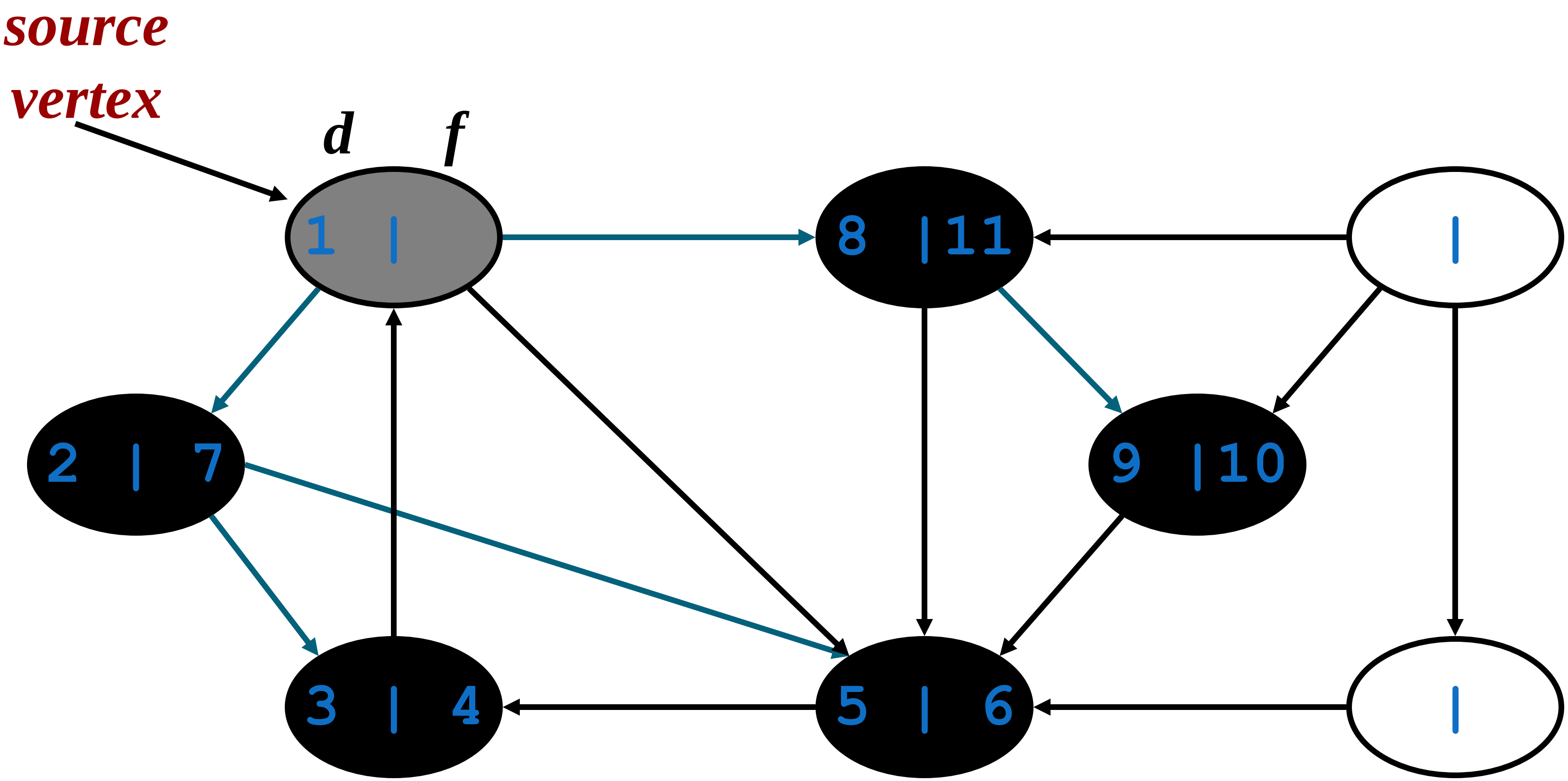
DFS Example



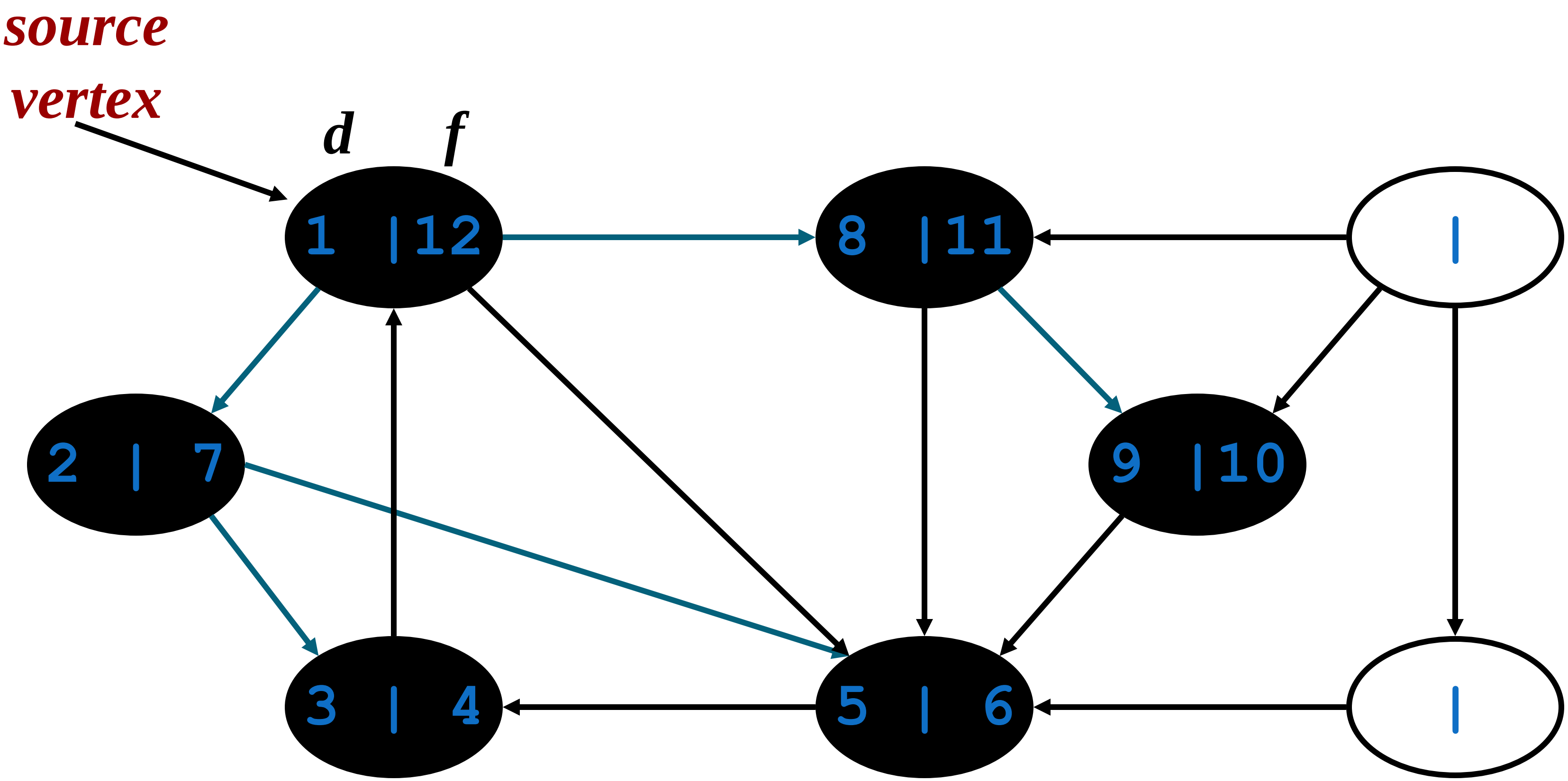
DFS Example



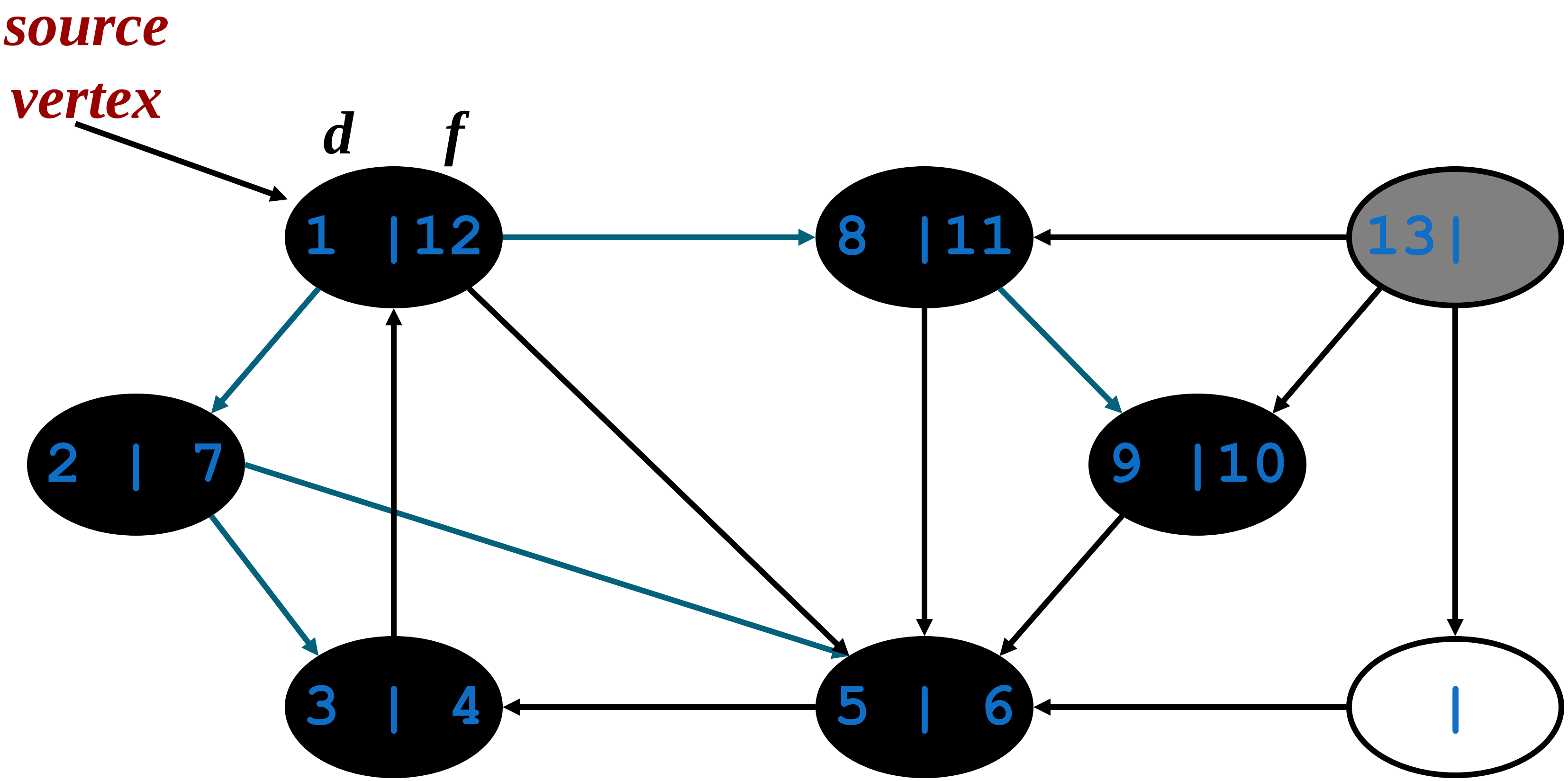
DFS Example



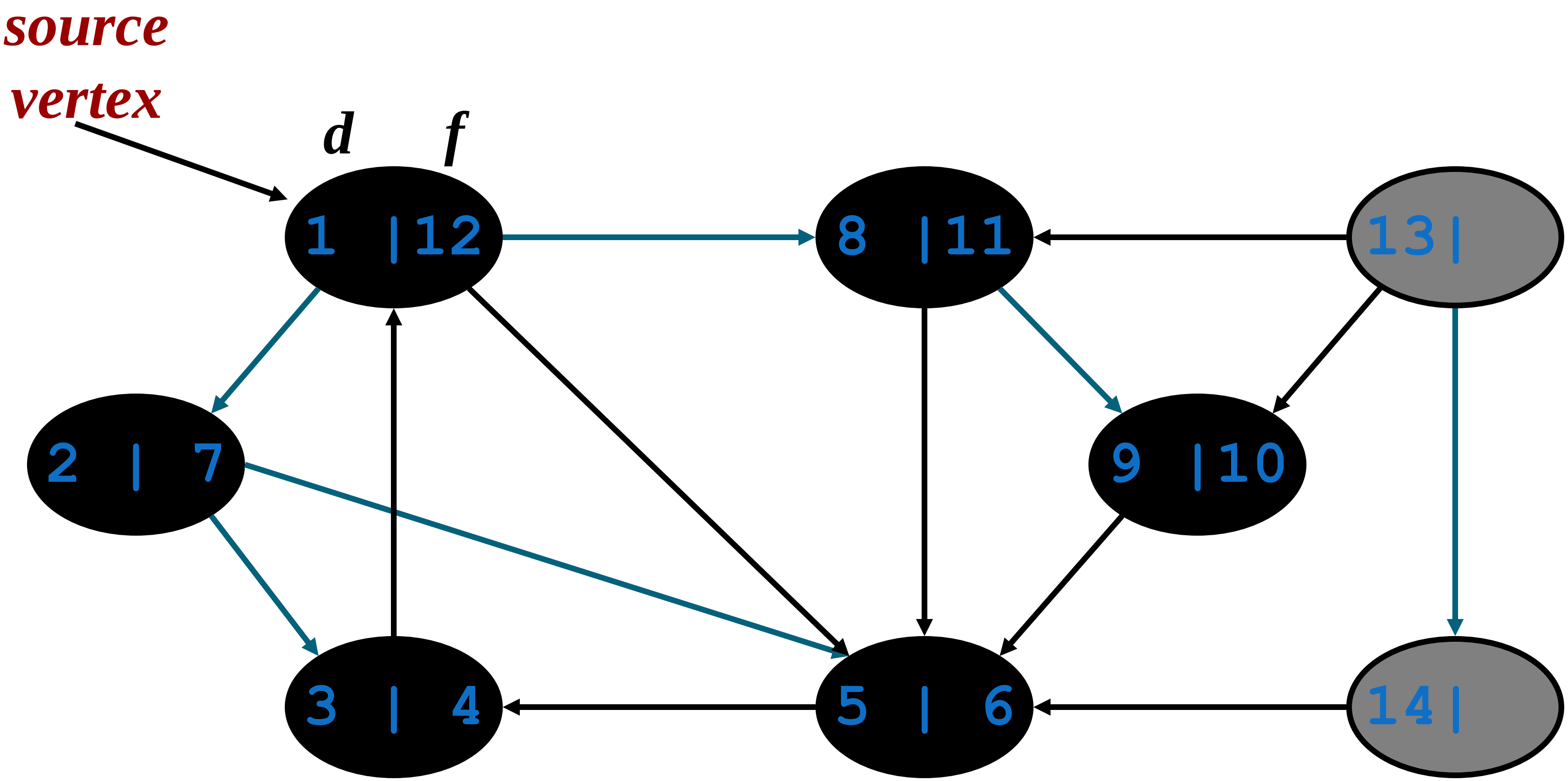
DFS Example



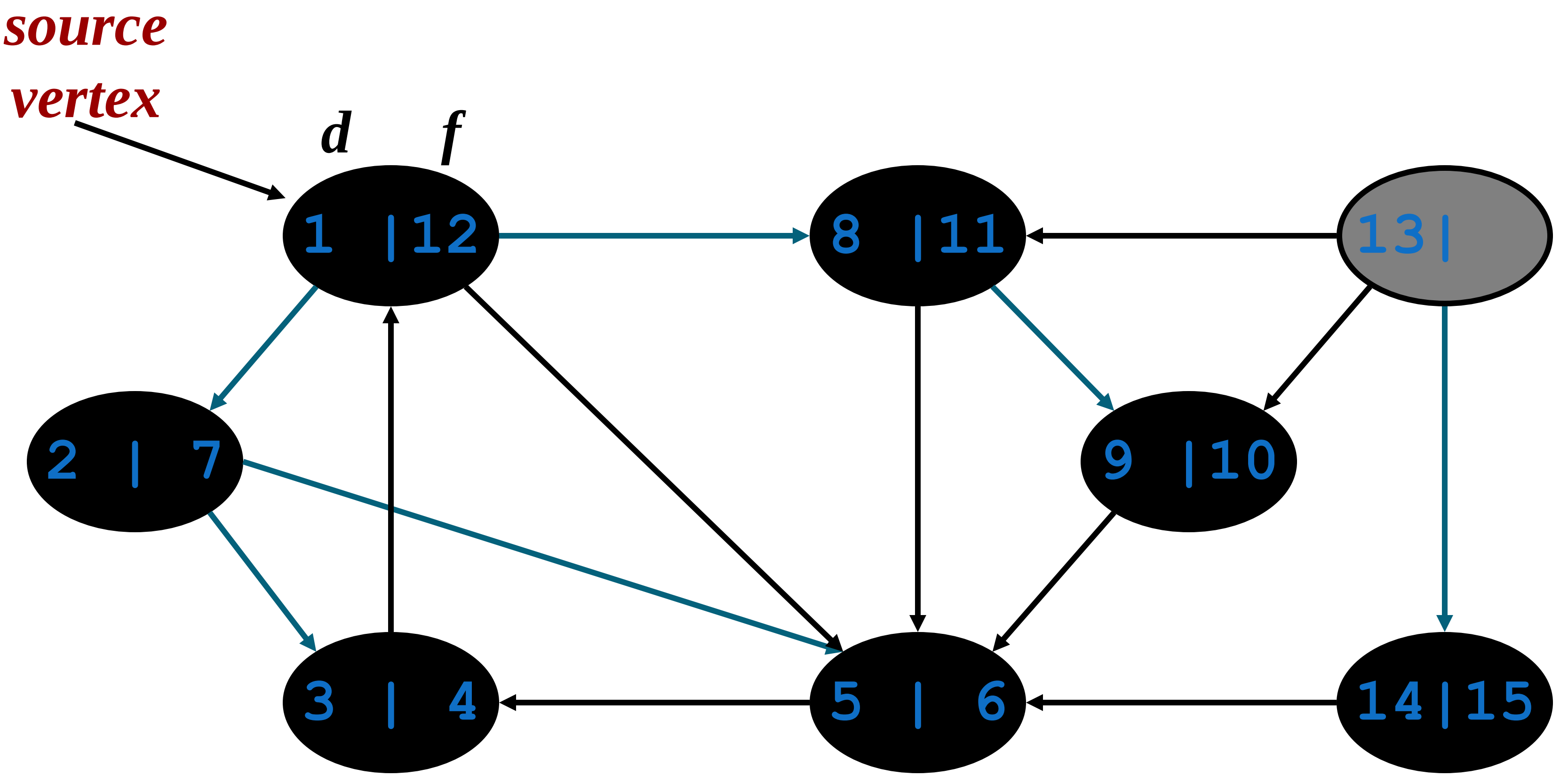
DFS Example



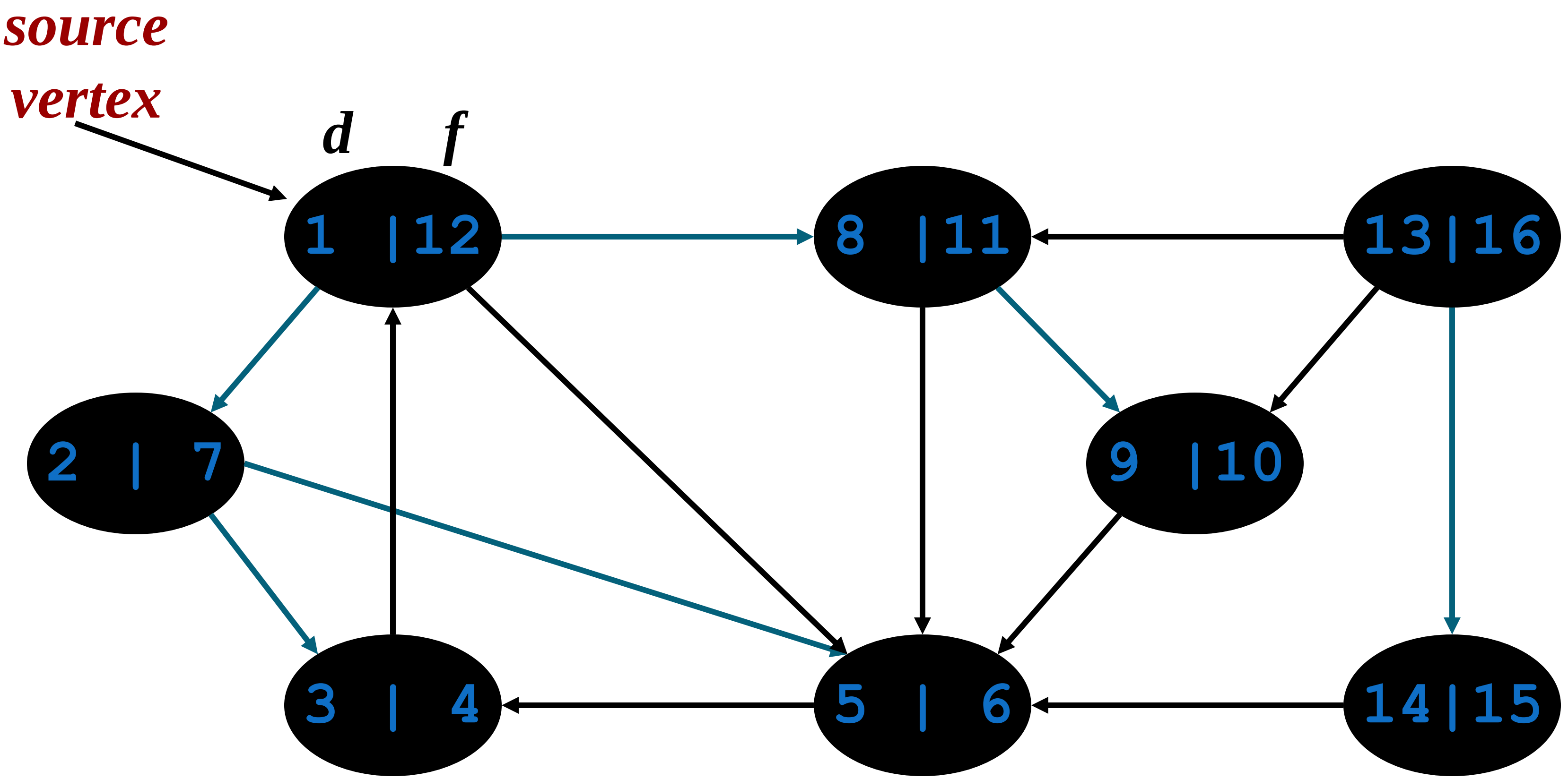
DFS Example



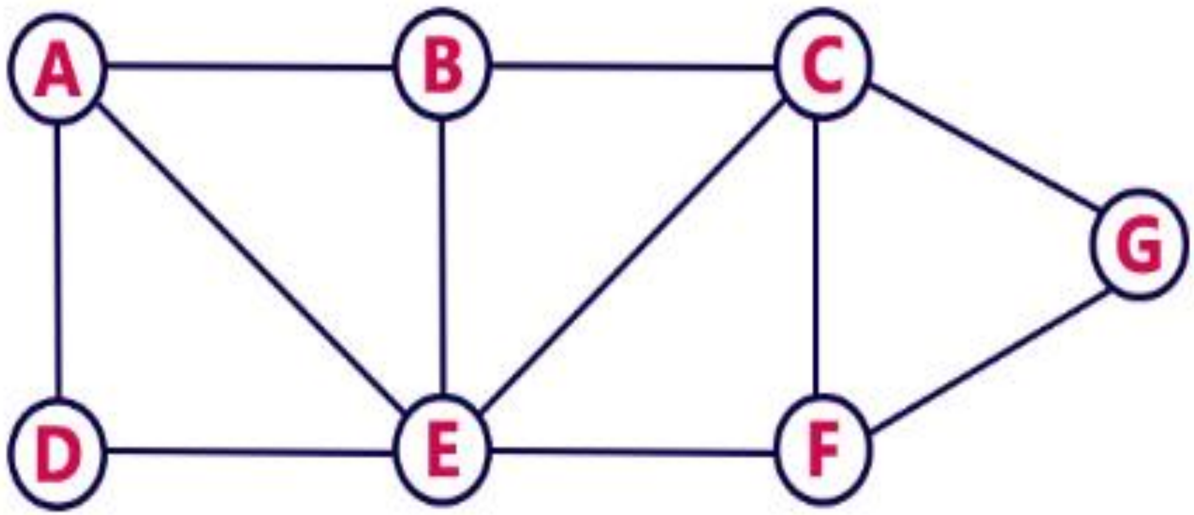
DFS Example



DFS Example

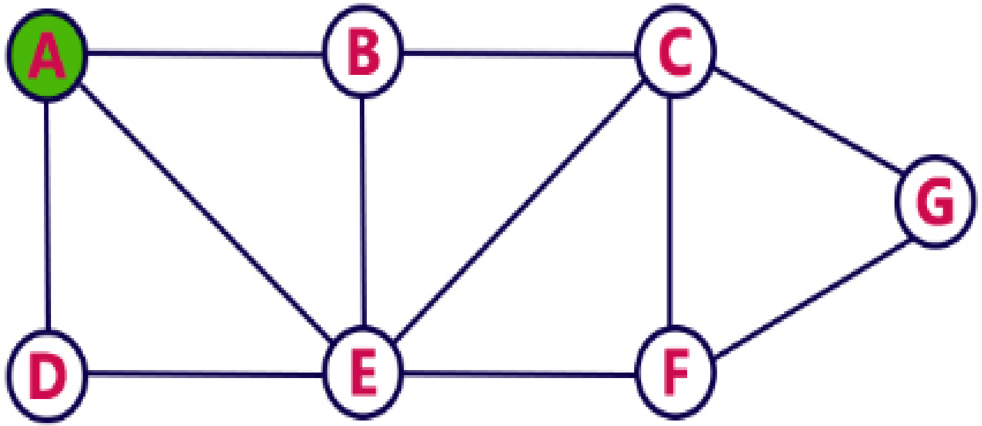


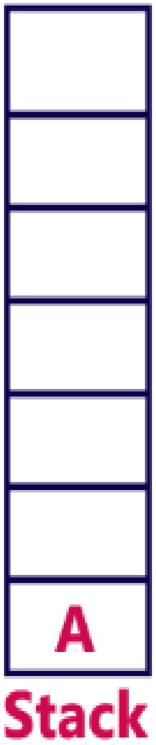
Consider the following example graph to perform DFS traversal



Step 1:

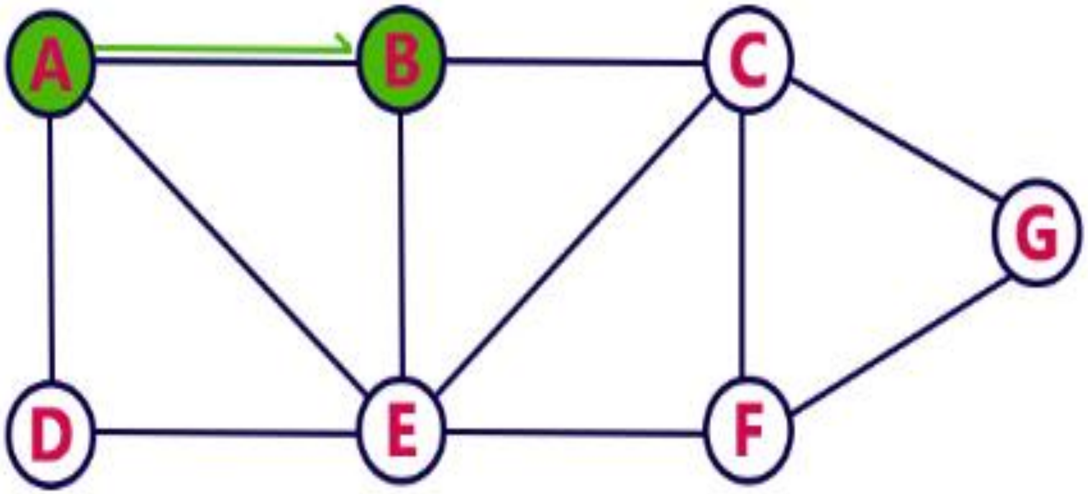
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.

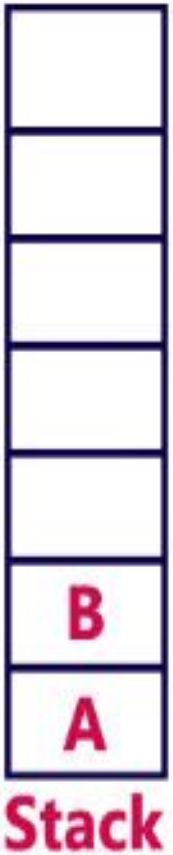




Step 2:

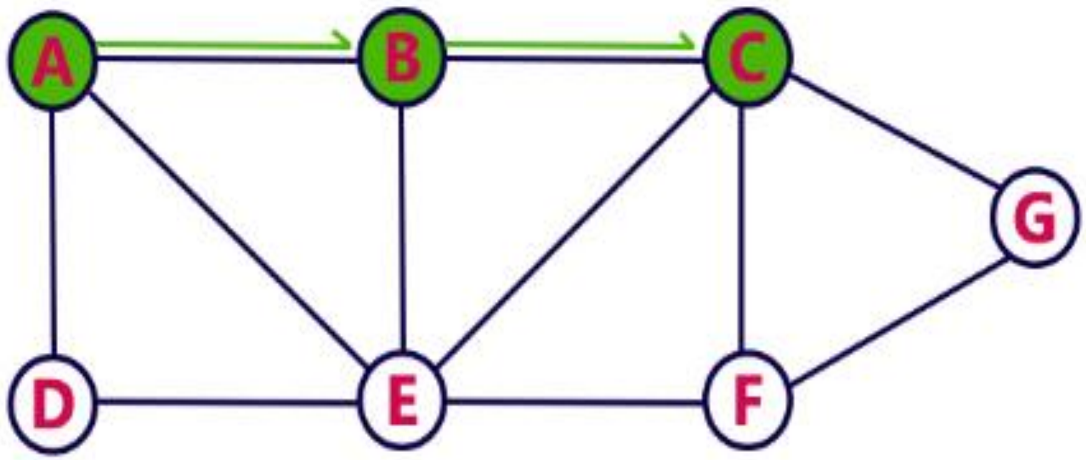
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

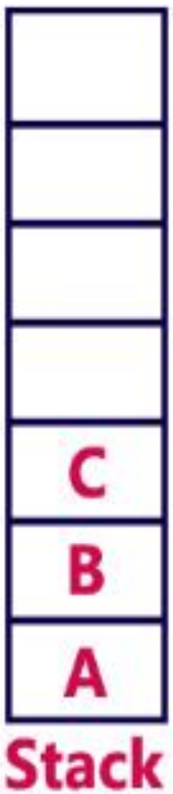




Step 3:

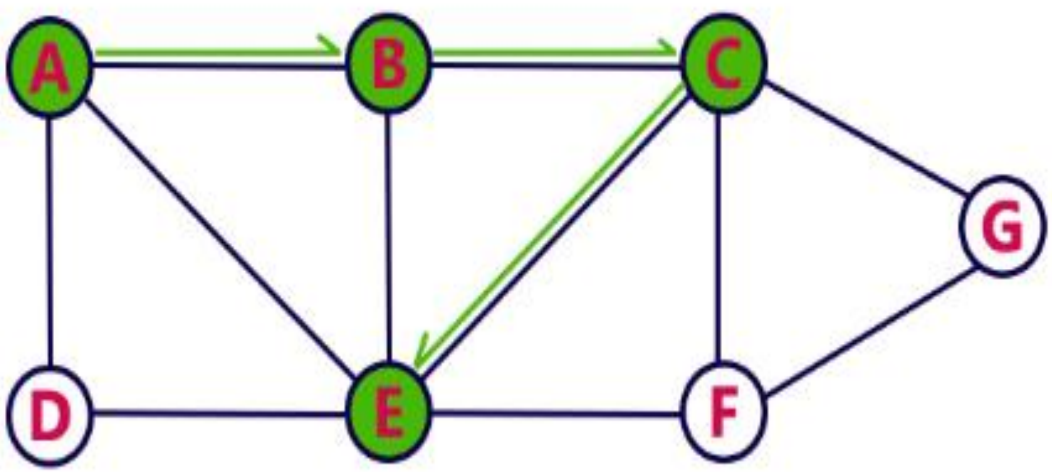
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.

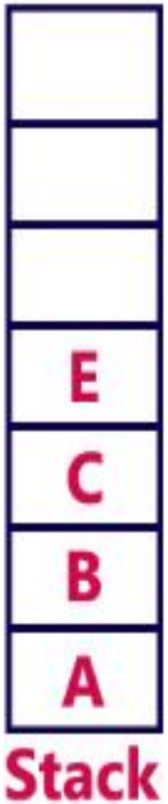




Step 4:

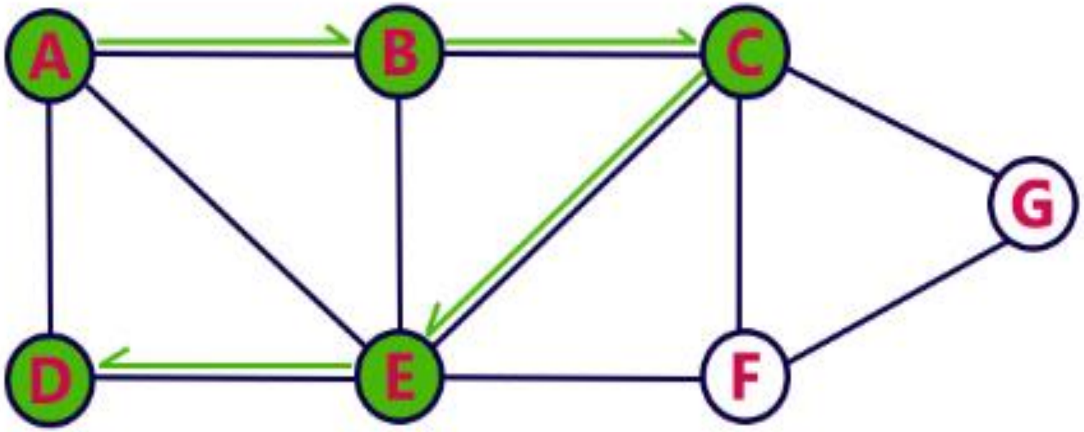
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack

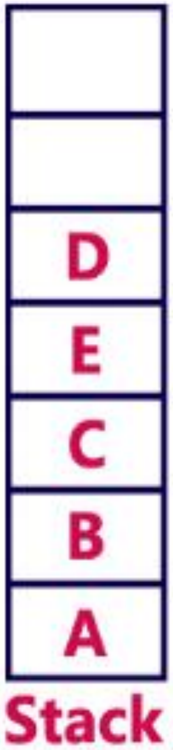




Step 5:

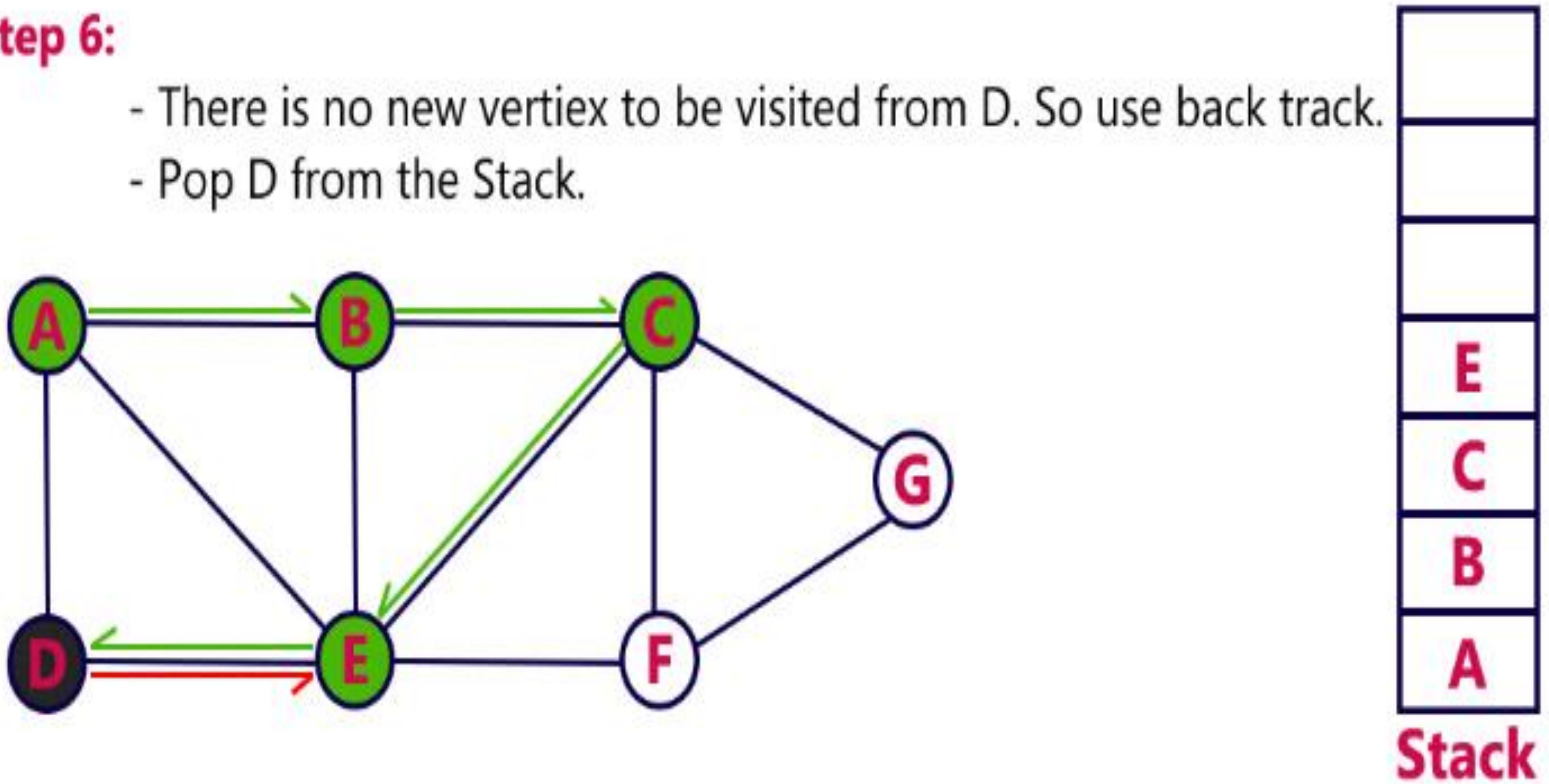
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack





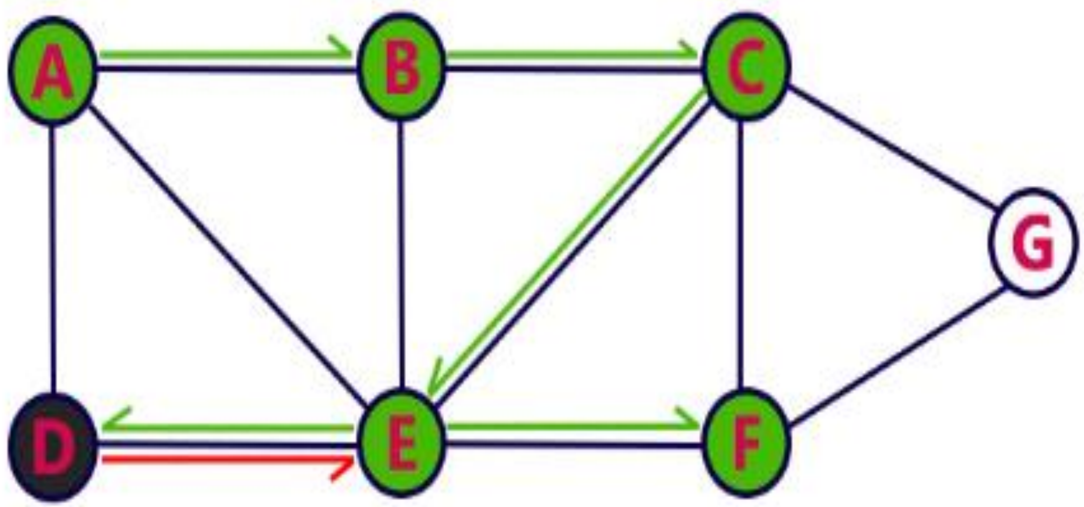
Step 6:

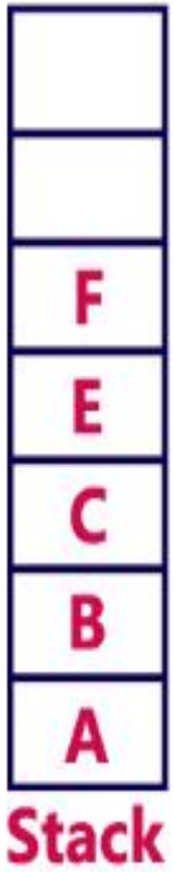
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.



Step 7:

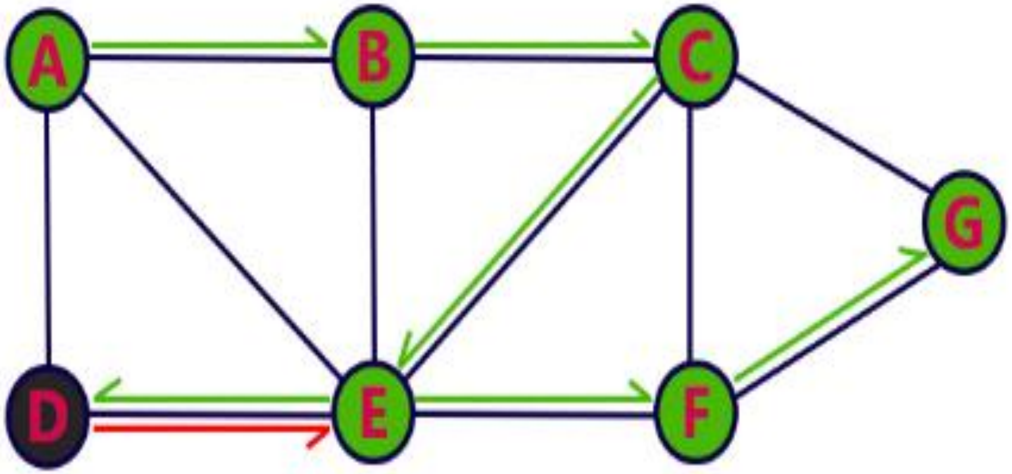
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.

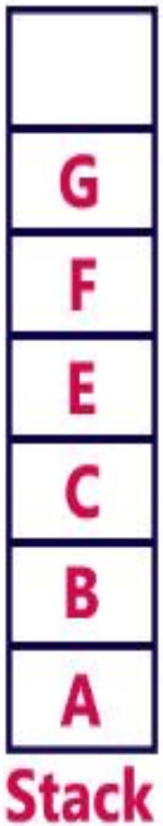




Step 8:

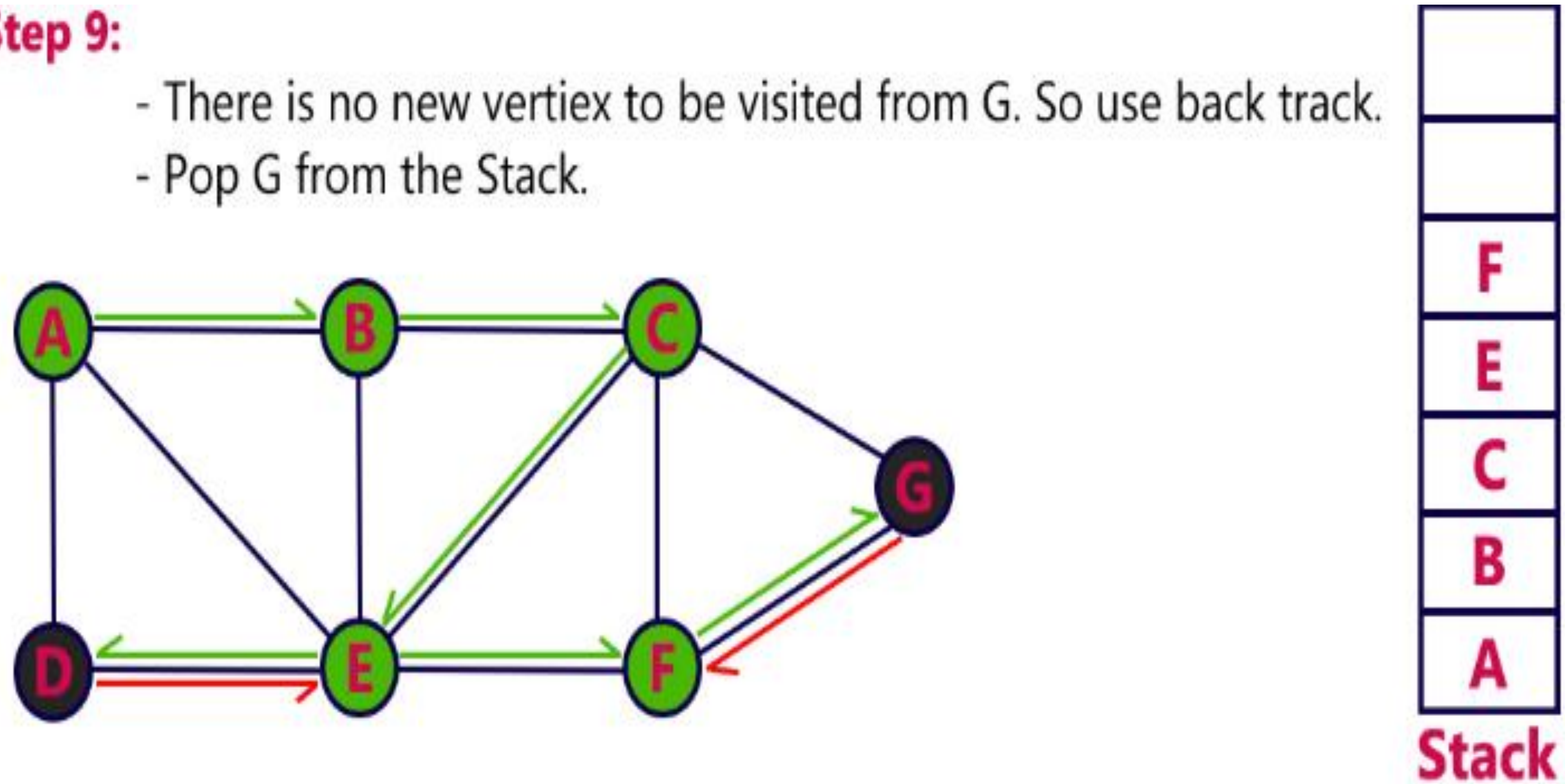
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.





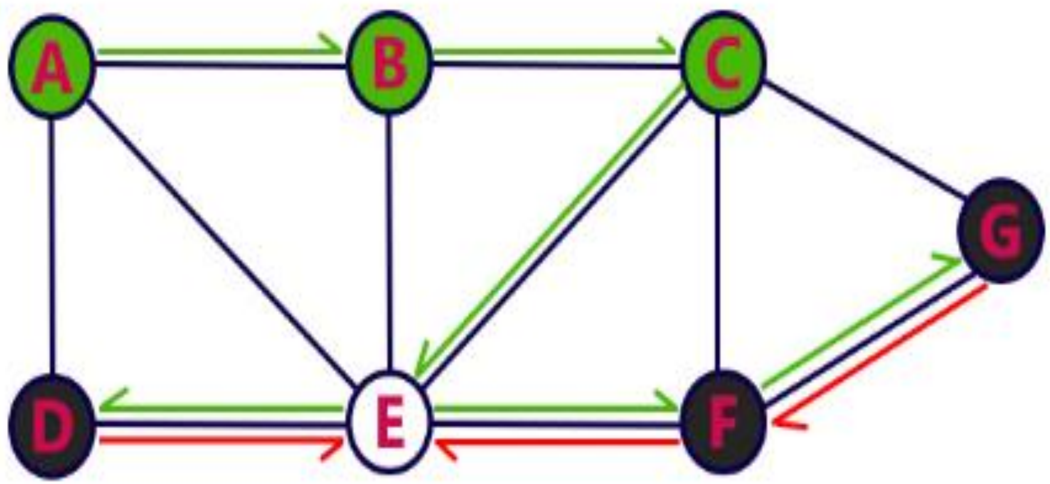
Step 9:

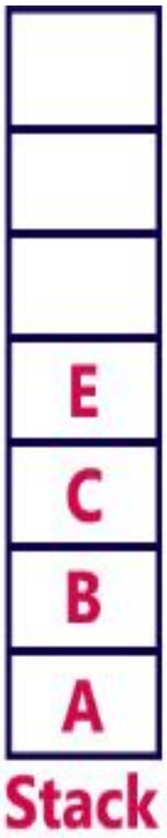
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.



Step 10:

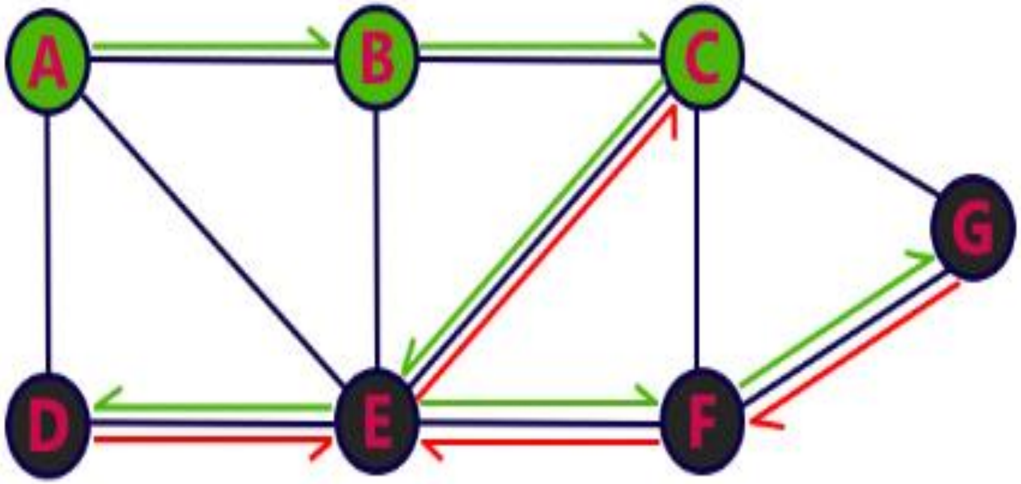
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.

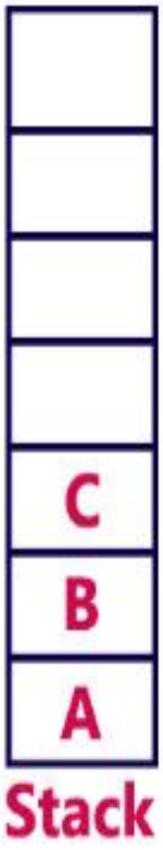




Step 11:

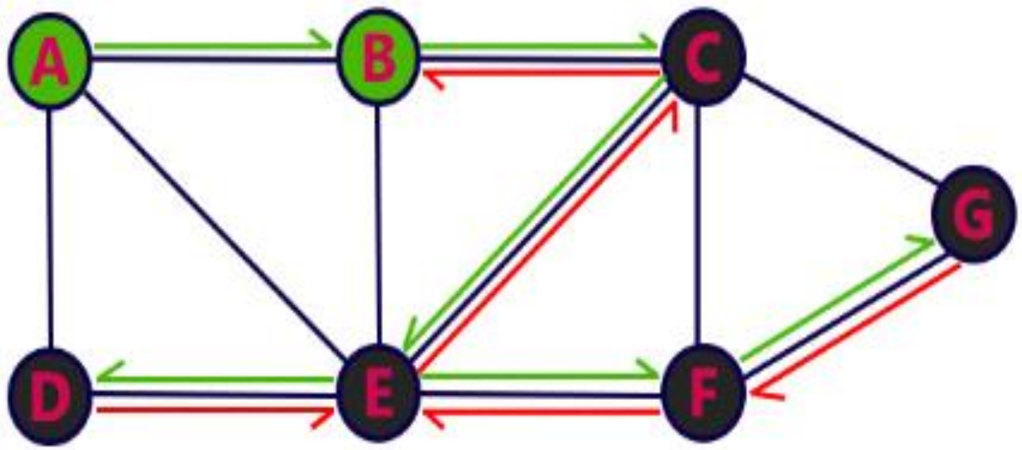
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

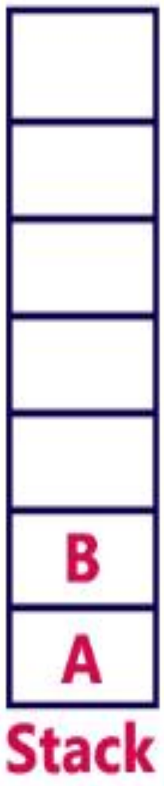




Step 12:

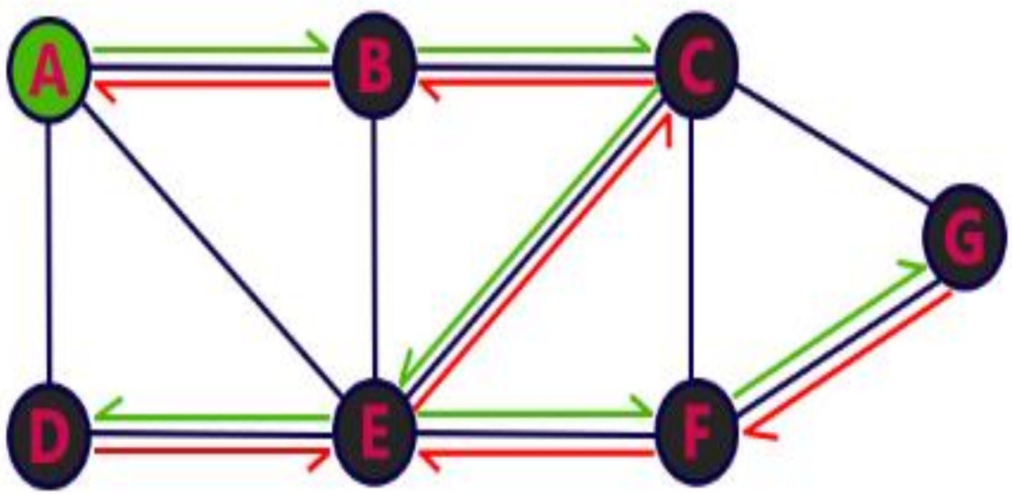
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.

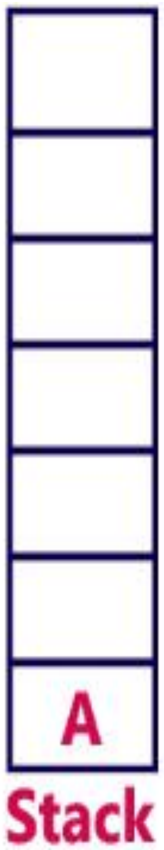




Step 13:

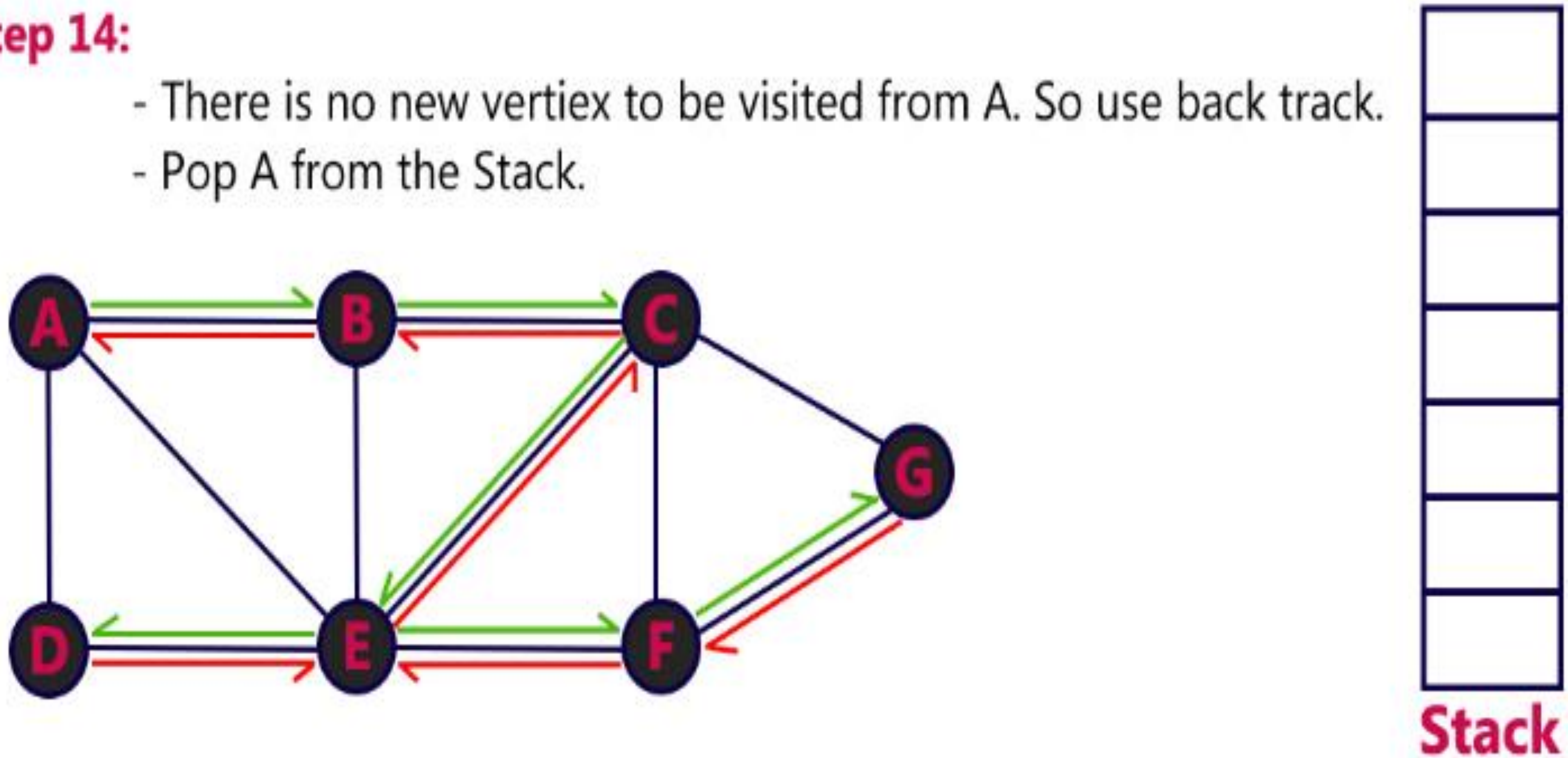
- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.



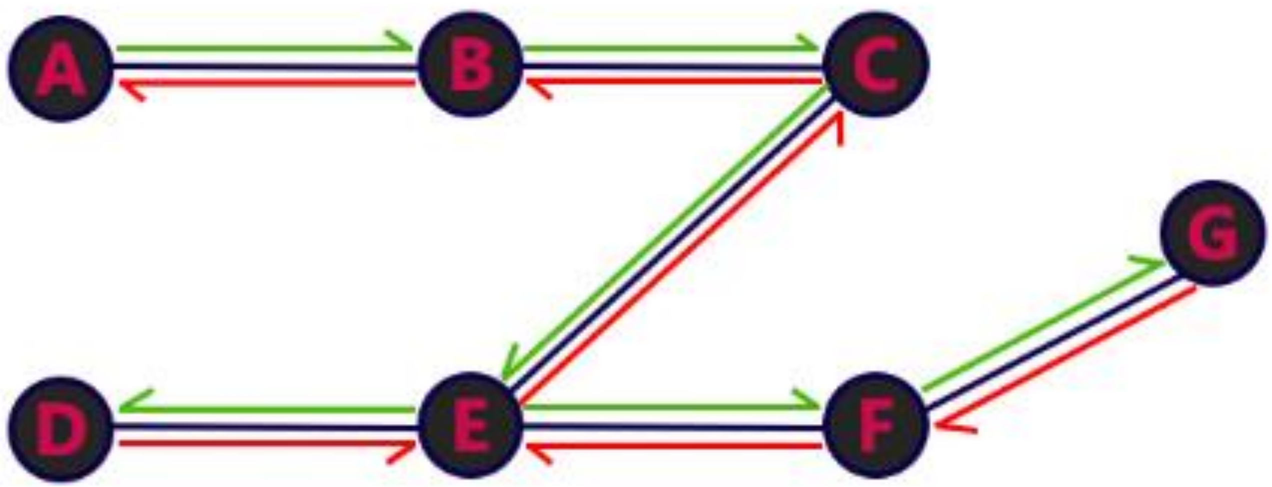


Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.



- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.

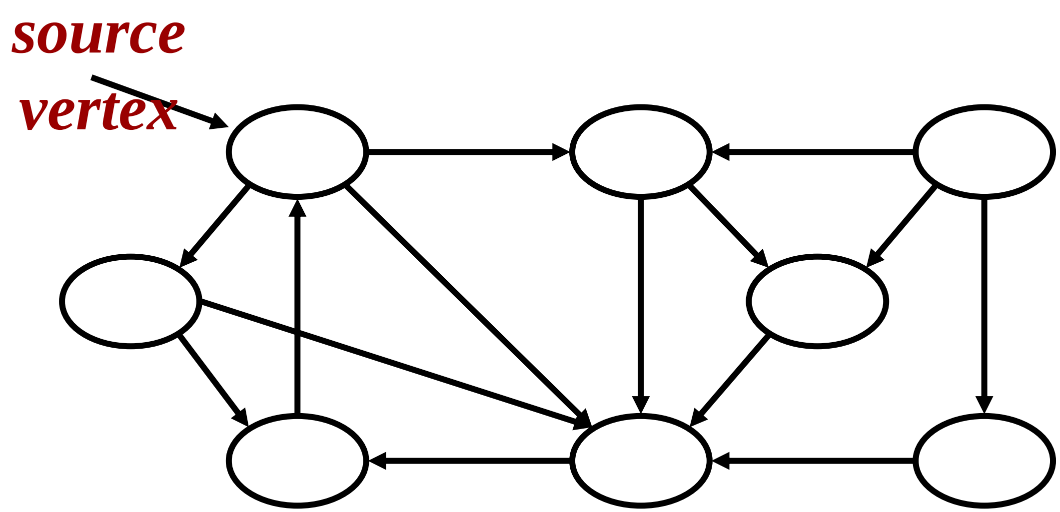


DFS: Algorithm

```
DFS(G)
    for each vertex u in V,
        color[u]=white;  p[u]=NIL
    time=0;
    for each vertex u in V
        if (color[u]=white)
            DFS-VISIT(u)
```

DFS: Algorithm (Cont.)

```
DFS-VISIT(u)
    color[u]=gray;
    time = time + 1;
    d[u] = time;
    for each v in Adj(u)  do
        if (color[v] = white)
            p[v] = u;
            DFS-VISIT(v);
    color[u] = black;
    time = time + 1; f[u]= time;
```



DFS: Complexity Analysis

Initialization complexity is O(V)

DFS_VISIT is called exactly once for each vertex O(V)

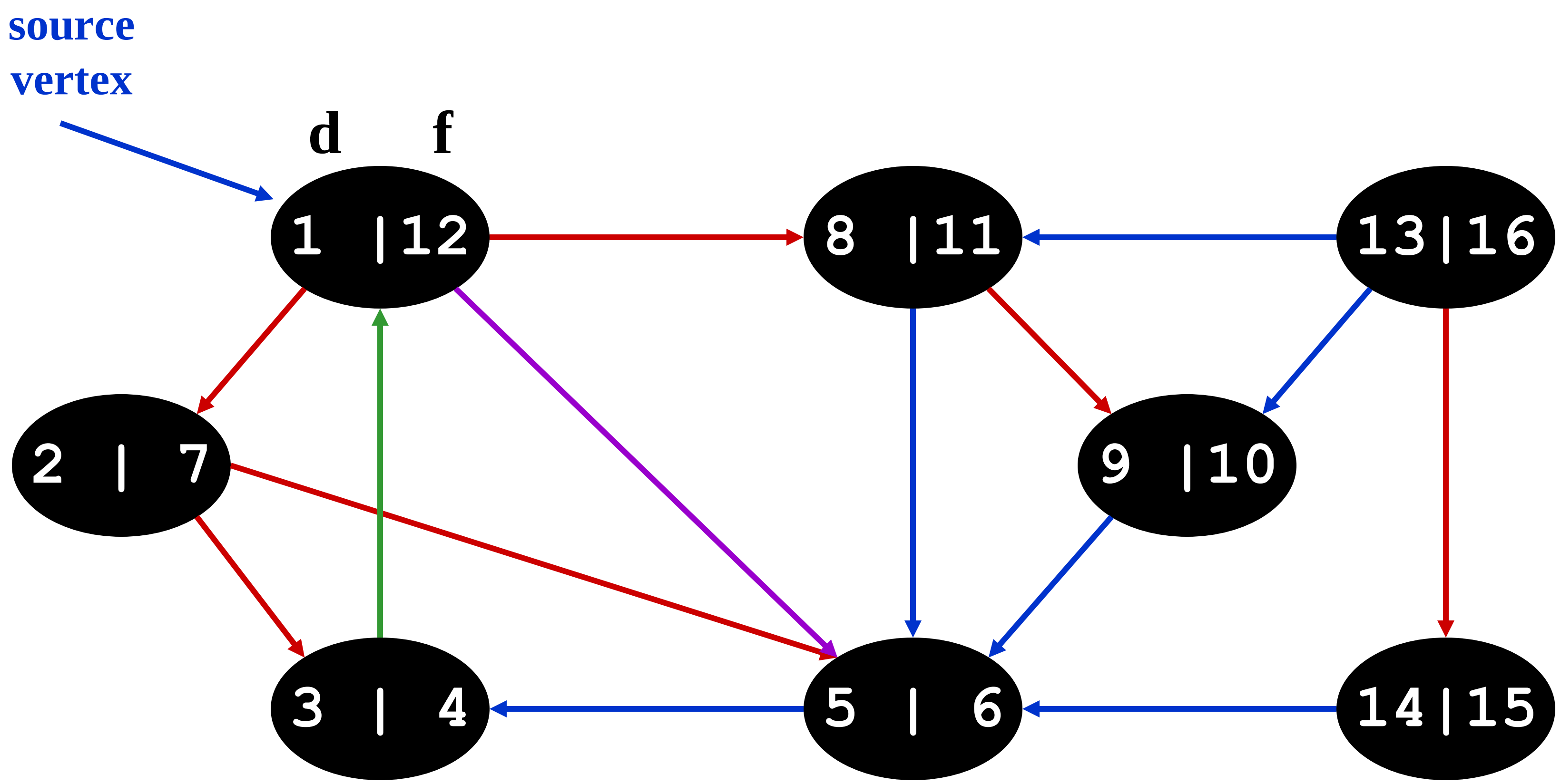
And DFS_VISIT scans all the edges $\Sigma|Adj(u)| \rightarrow$ O(E)

Overall complexity is O(V + E)

DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  - *Tree edge*: encounter new (white) vertex
  - *Back edge*: from descendent to ancestor
  - *Forward edge*: from ancestor to descendent
  - *Cross edge*: between a tree or subtrees
- Note: tree & back edges are important; most algorithms don't distinguish forward & cross

DFS Example

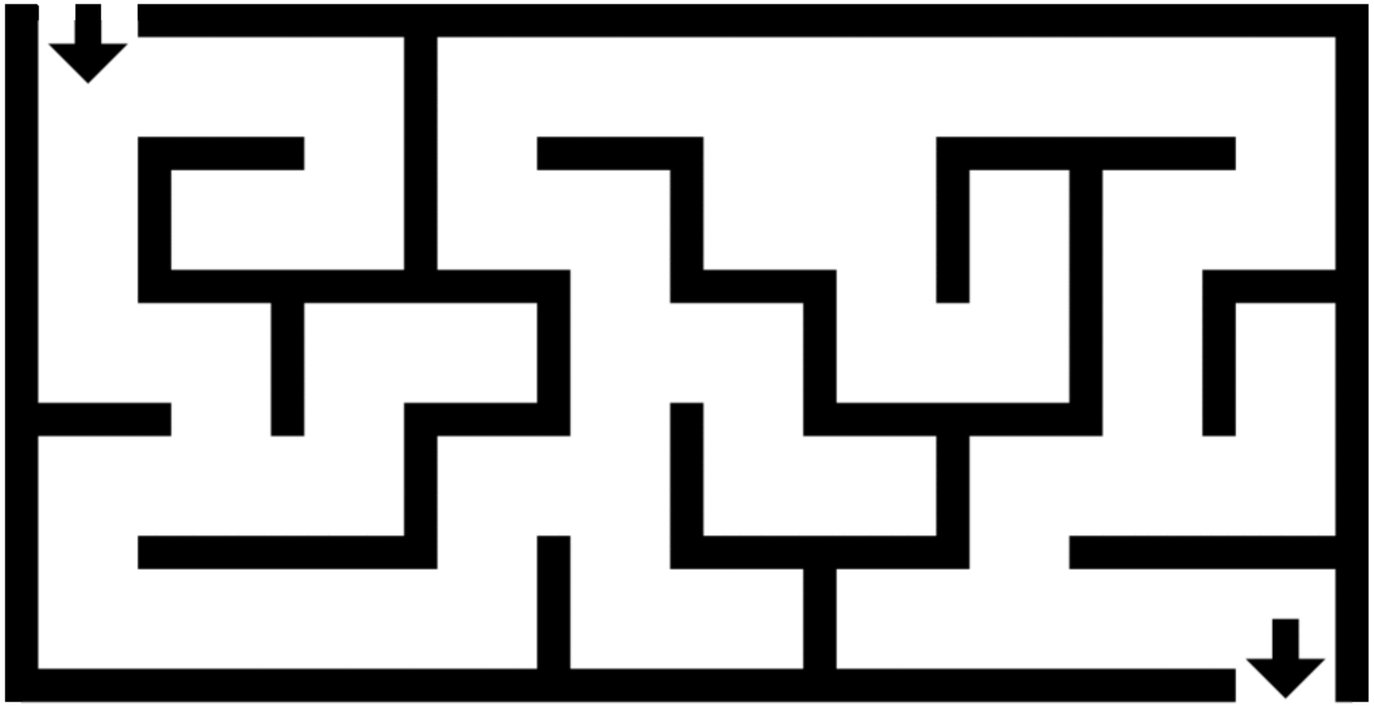


Tree edges Back edges Forward edges Cross edges

DFS: Application

- Topological Sort
- Simulation of games
- Scheduling events

Solving Maze Problem using DFS



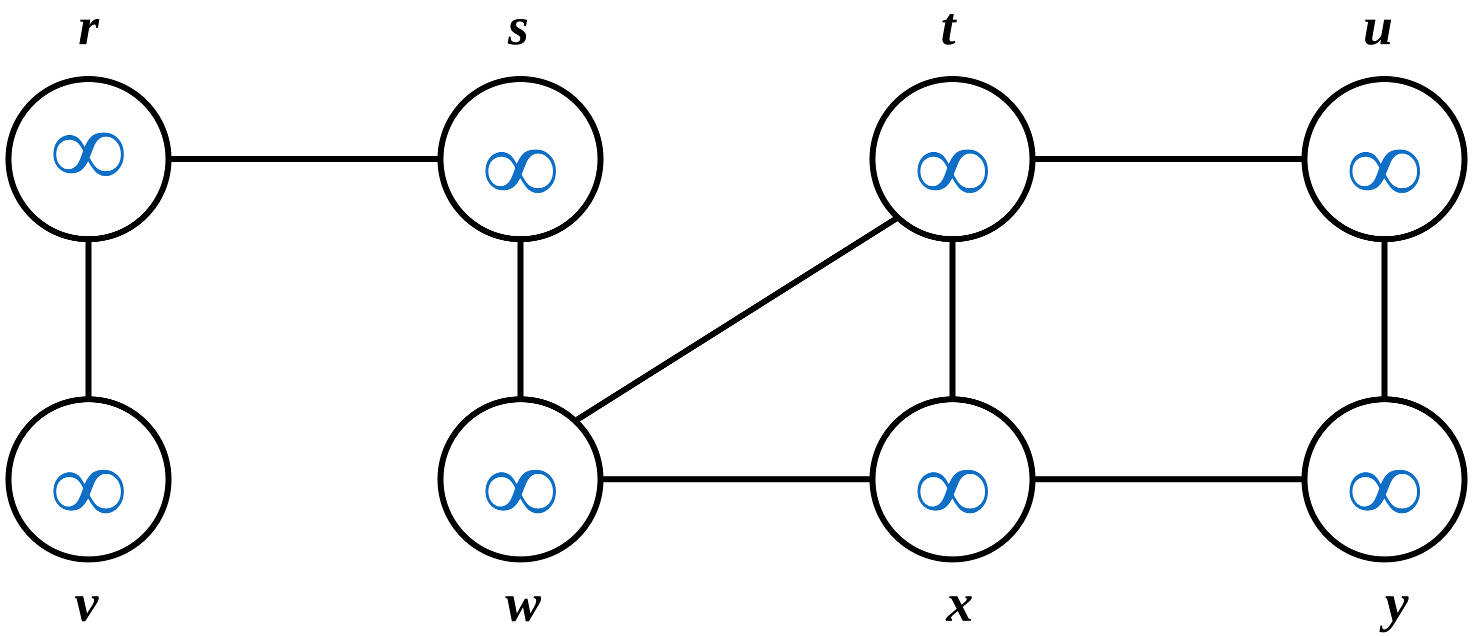
Breadth-first Search (BFS)

- Search for all vertices that are directly reachable from the root (called level 1 vertices)
- After mark all these vertices, visit all vertices that are directly reachable from any level 1 vertices (called level 2 vertices), and so on.
- Level k vertices are directly reachable from a level k – 1 vertices

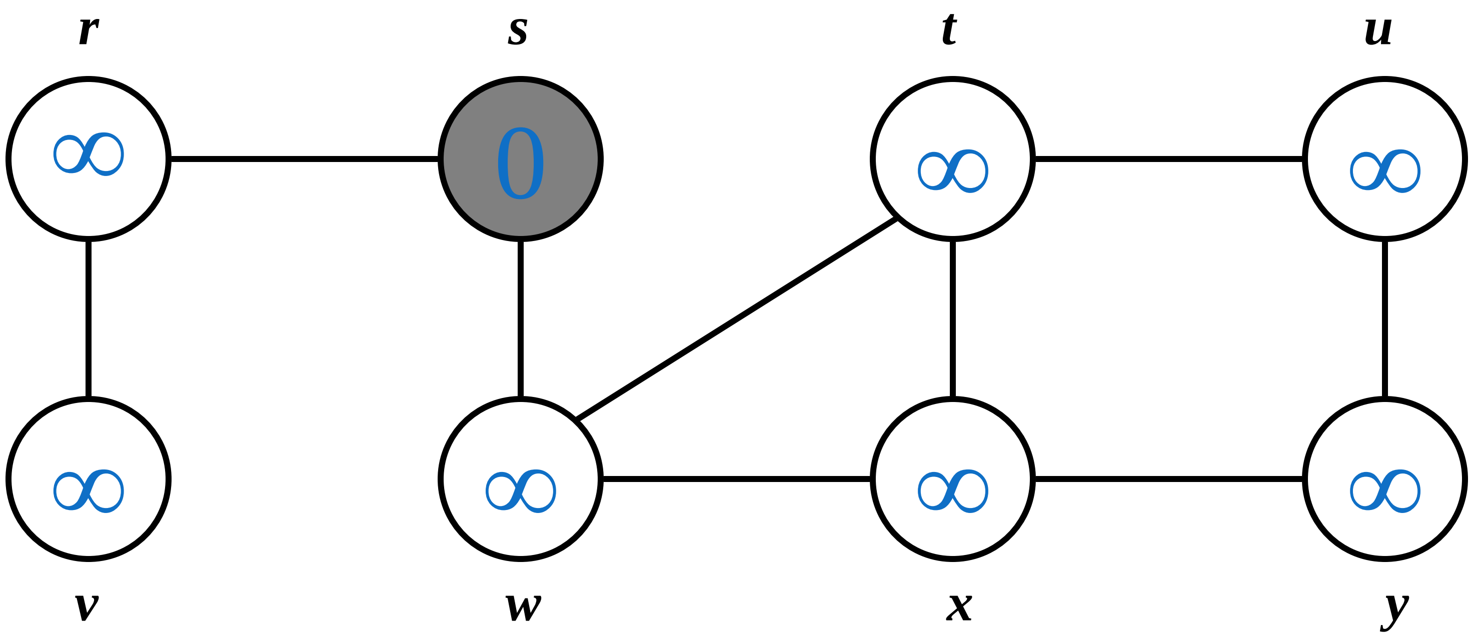
BFS: the Color Scheme

- White vertices have not been discovered
  - All vertices start out white
- Grey vertices are discovered but not fully explored
  - They may be adjacent to white vertices
- Black vertices are discovered and fully explored
  - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of grey vertices

Example

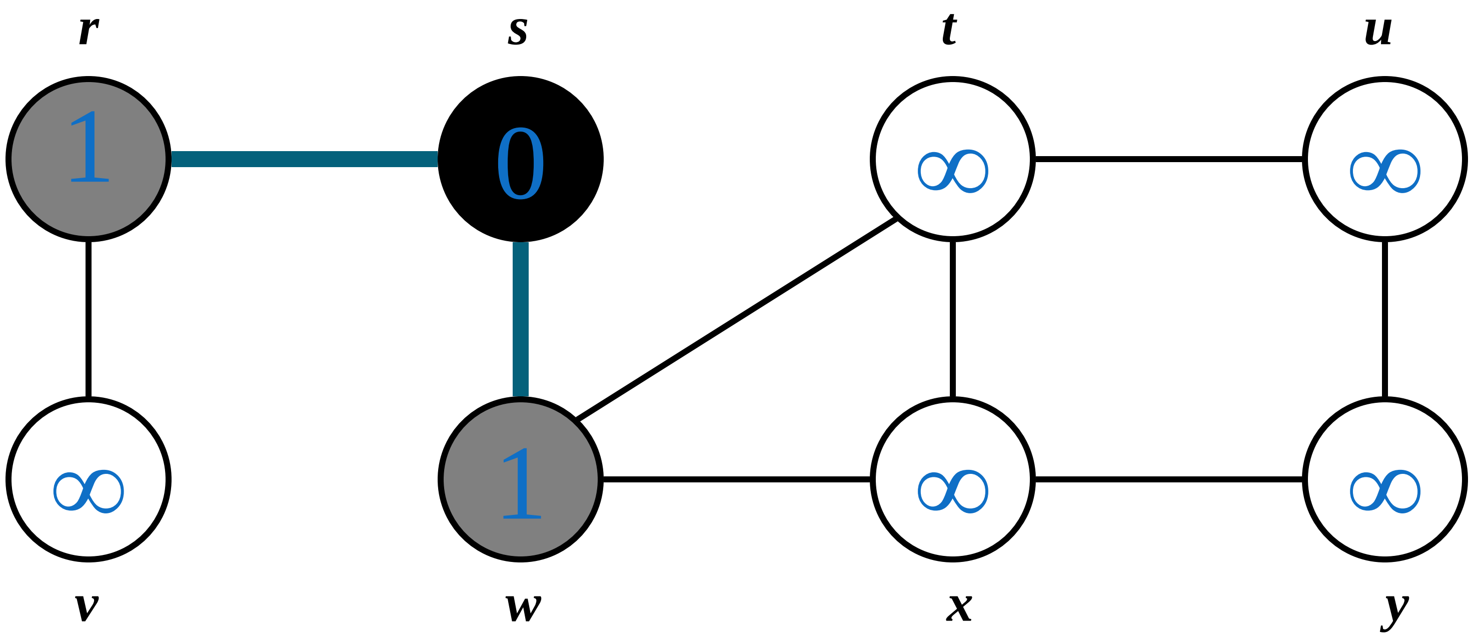


Example



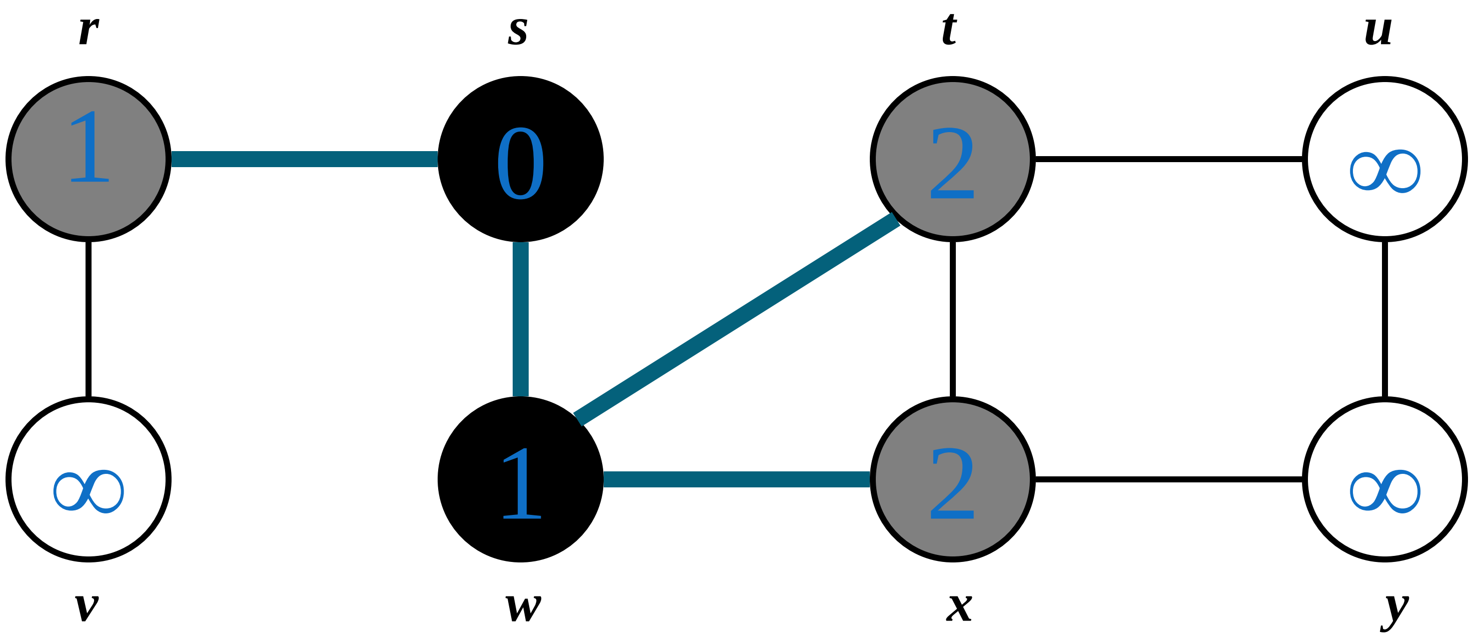
Q: $\boxed{s}$

Example



Q: | w | r |

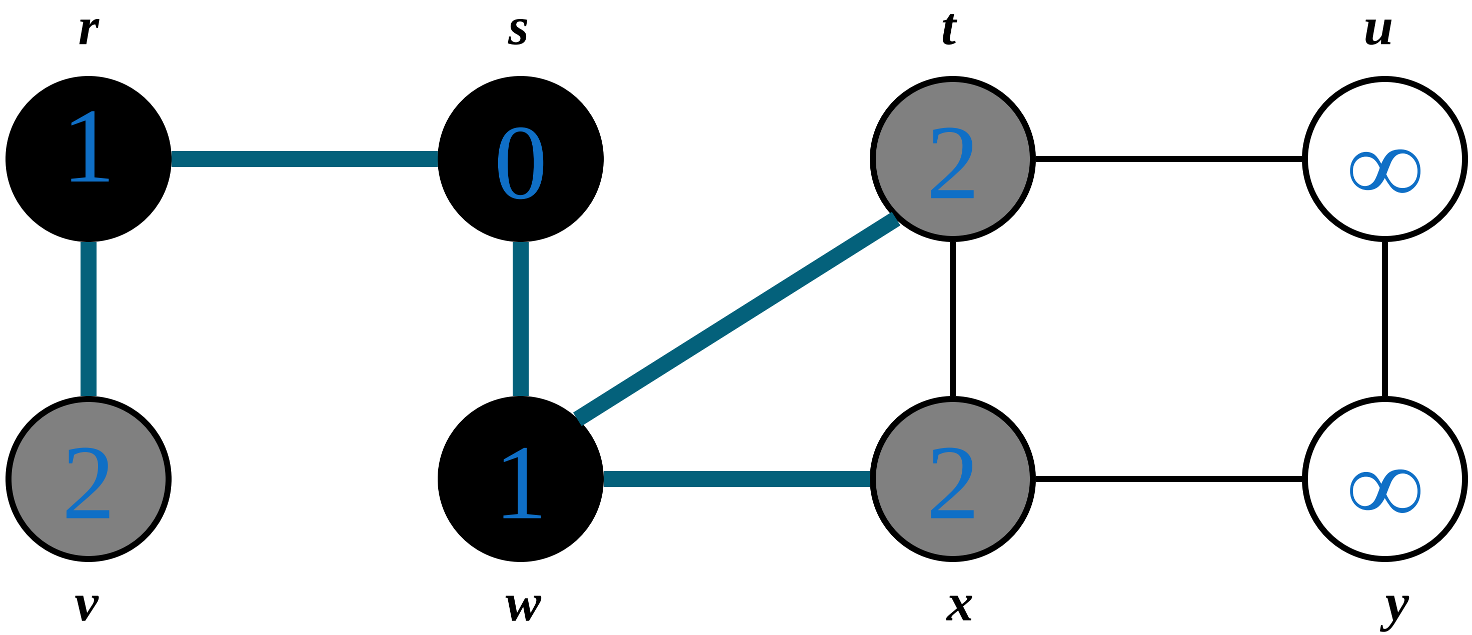
Example



Q:

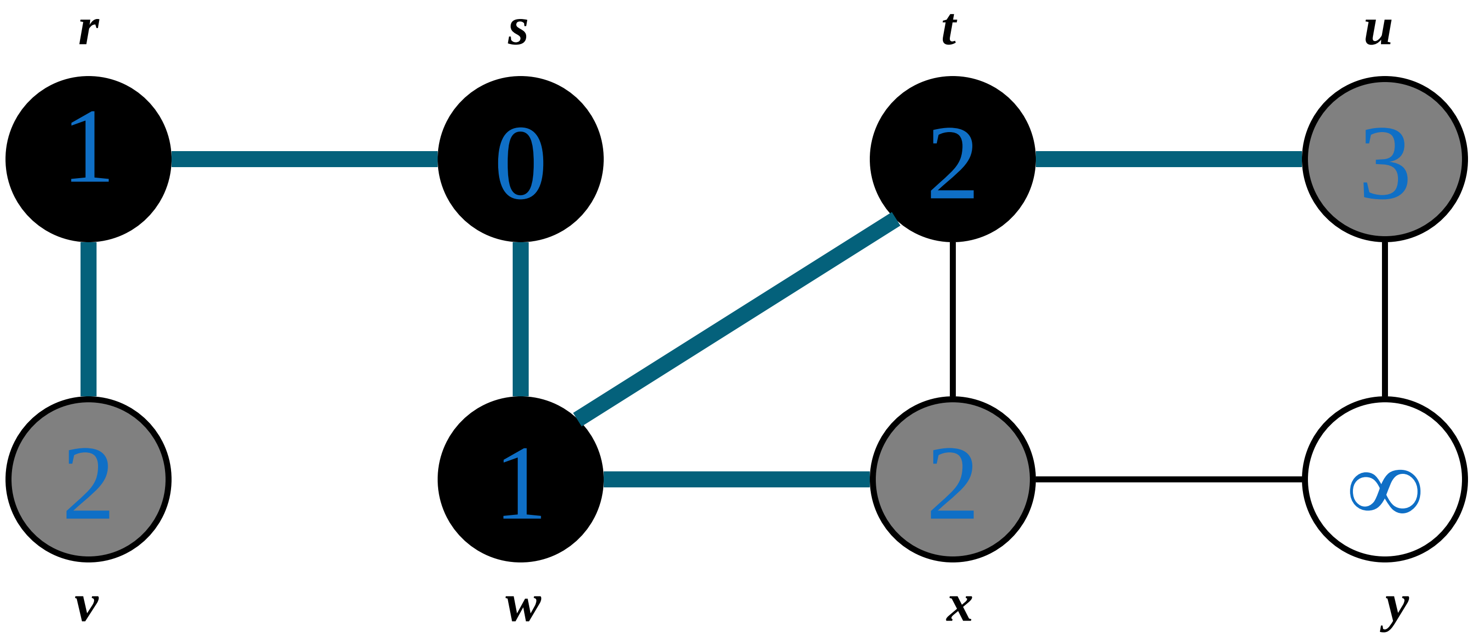
r	*t*	*x*

Example



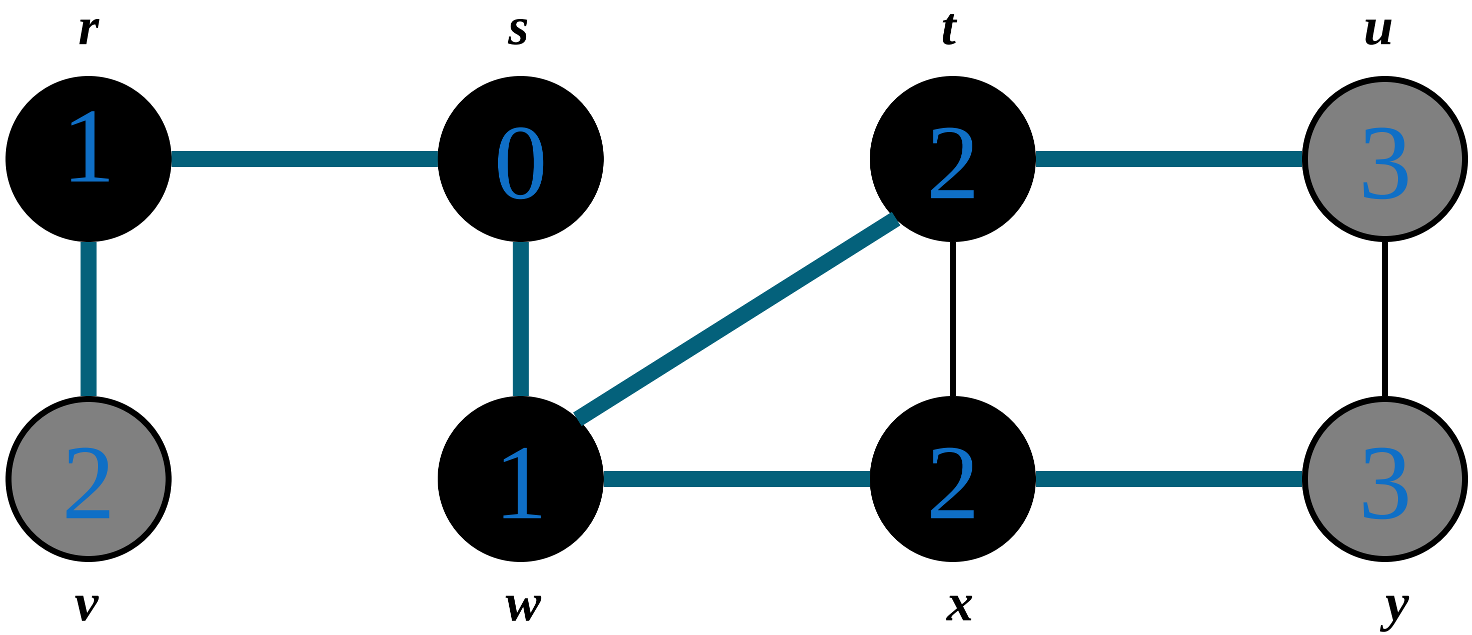
Q: | t | x | v |

Example



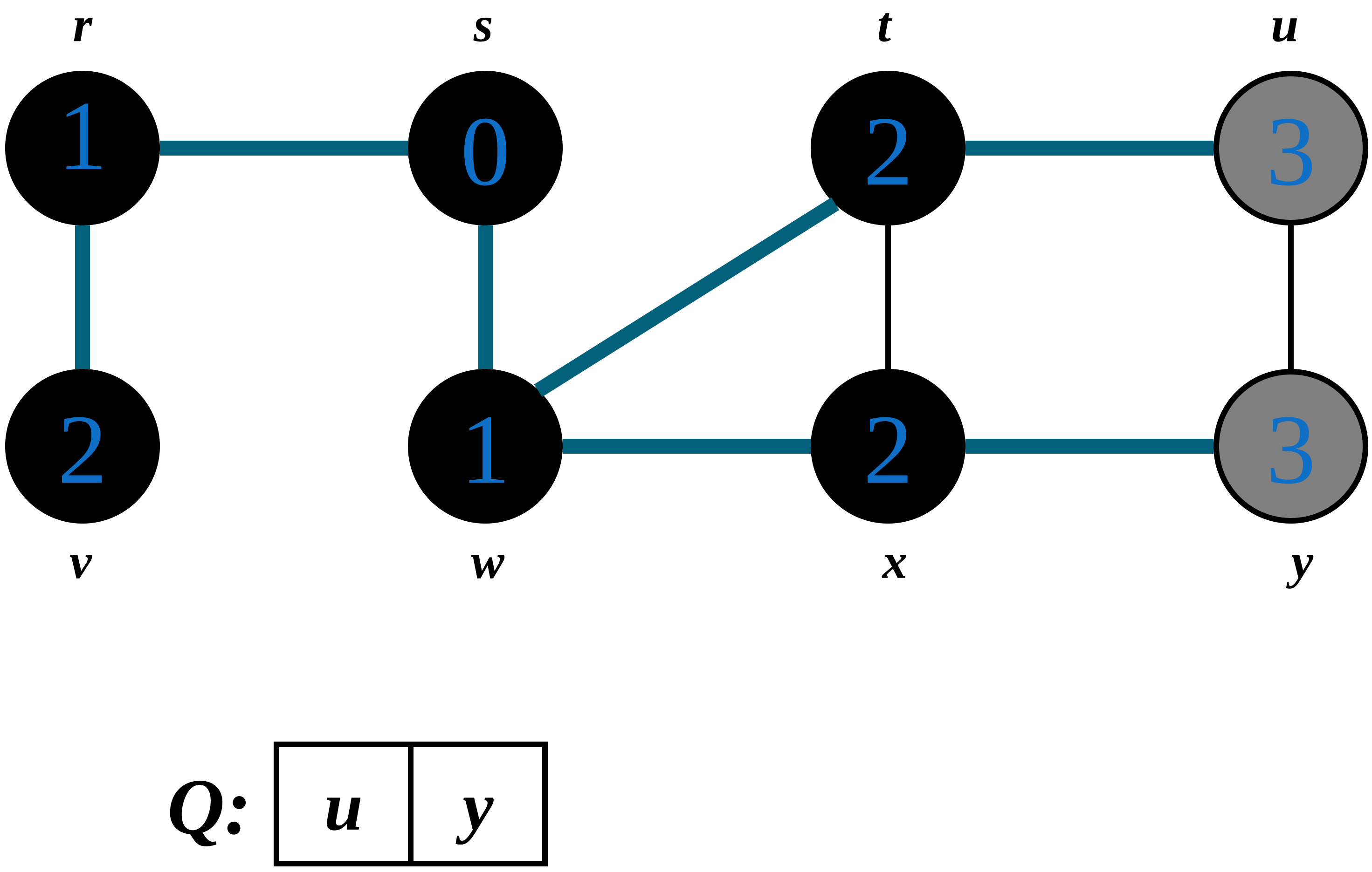
Q: | x | v | u |

Example

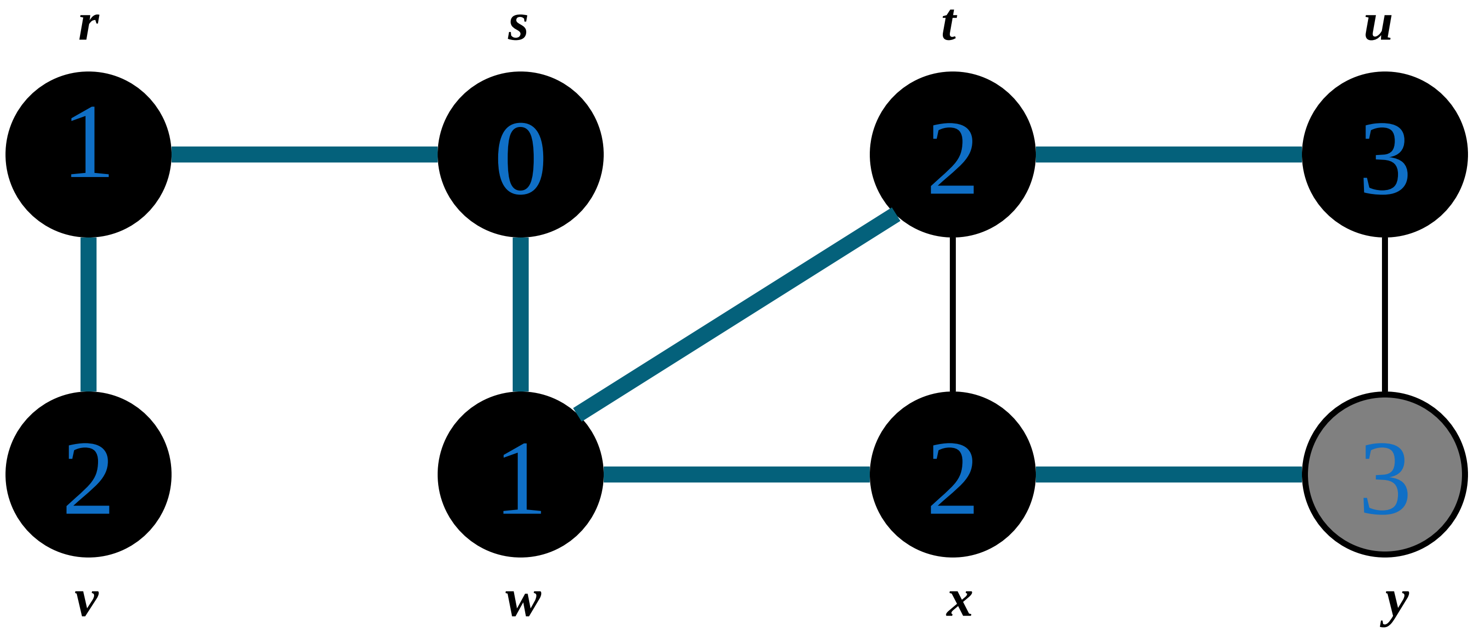


Q: | v | u | y |

Example

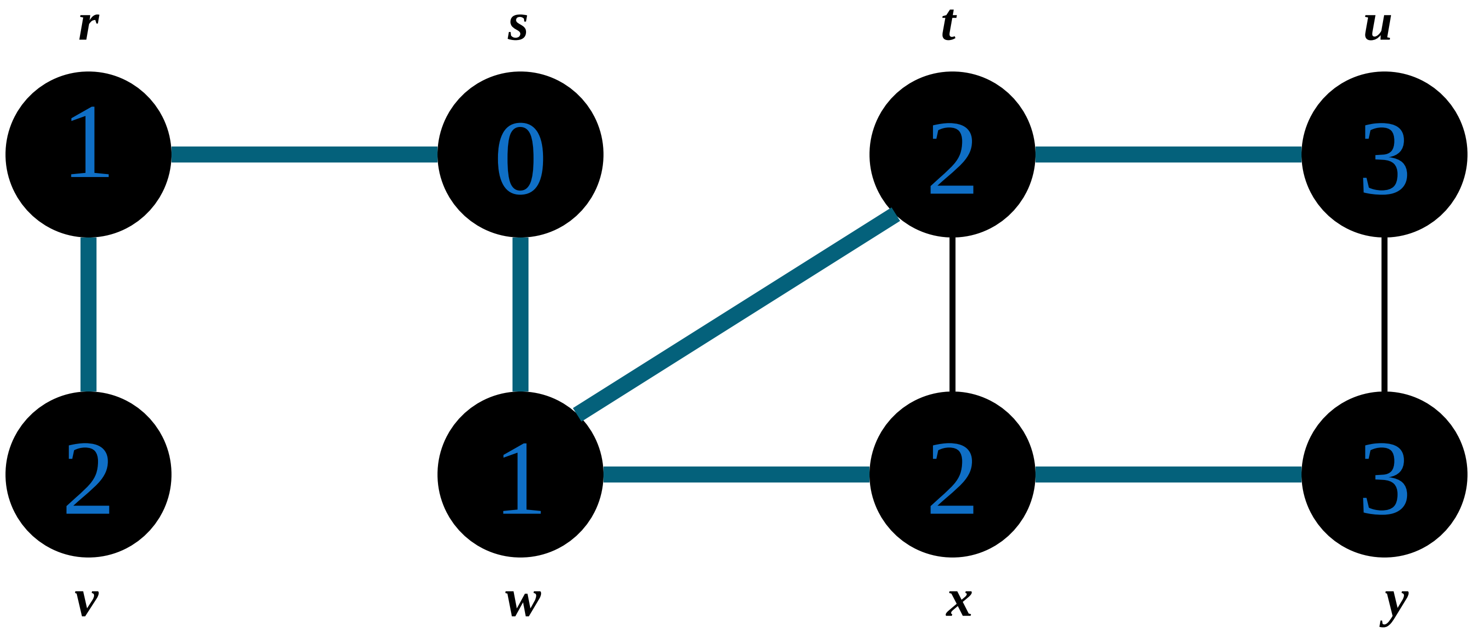


Example



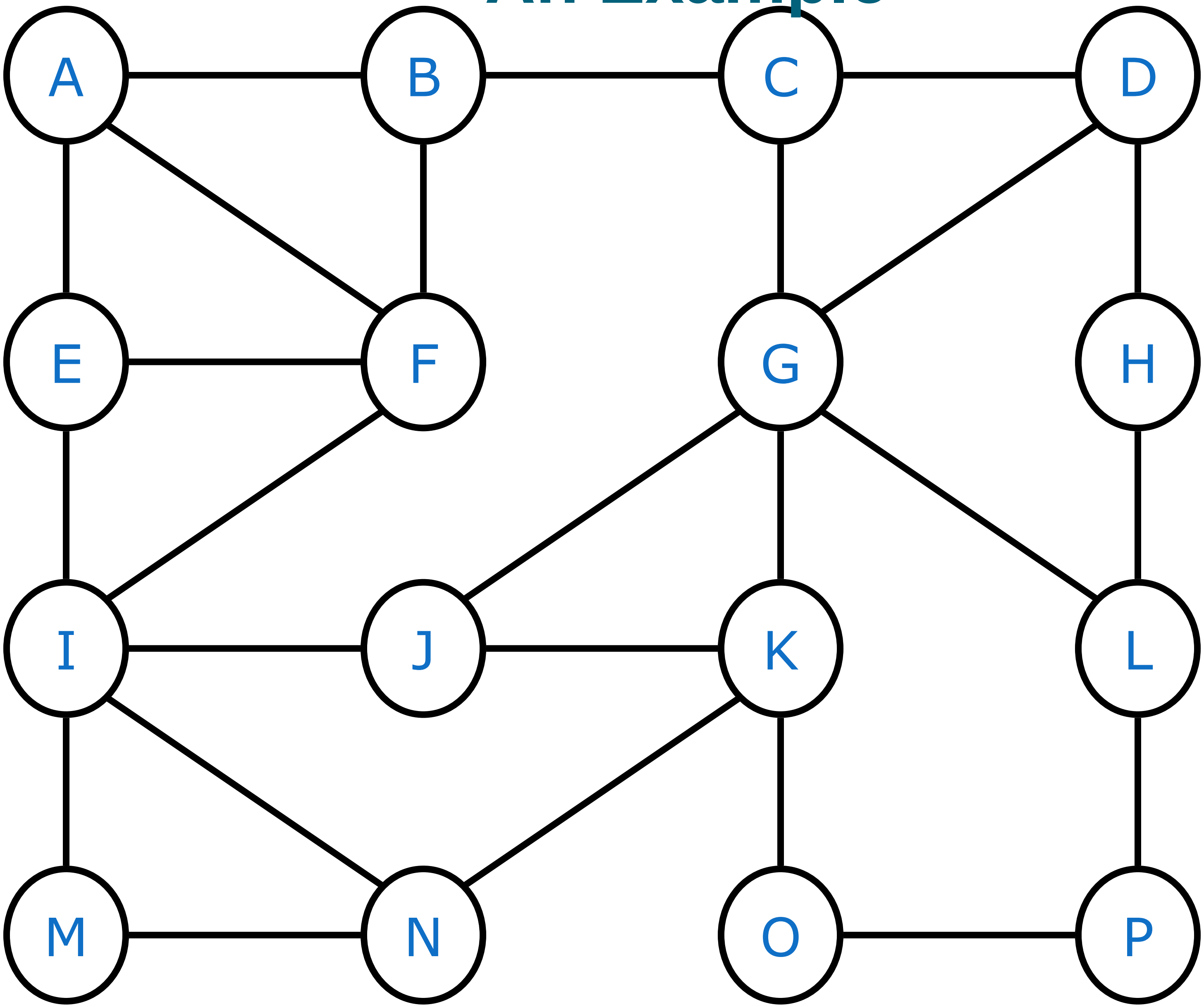
Q: $\boxed{y}$

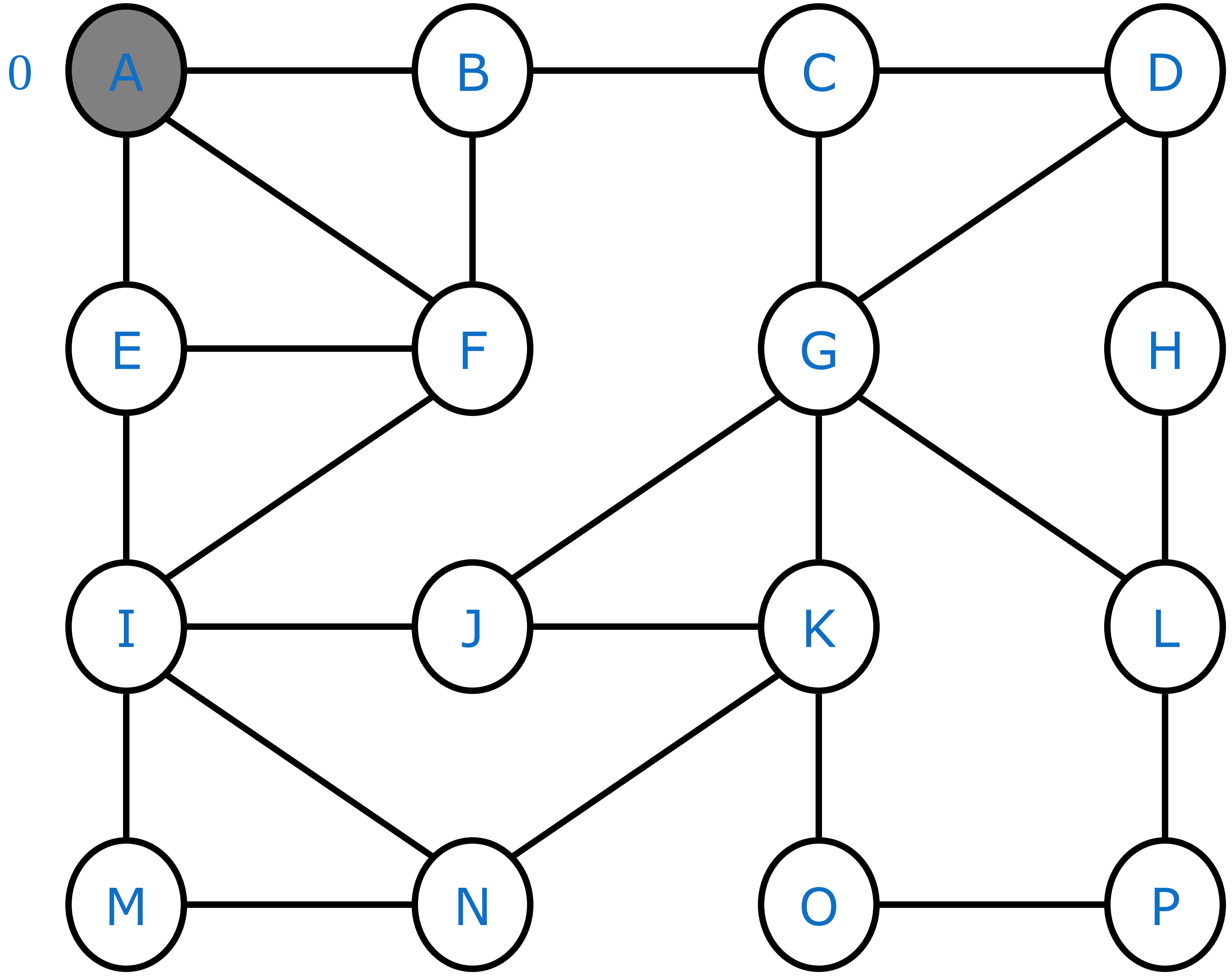
Example

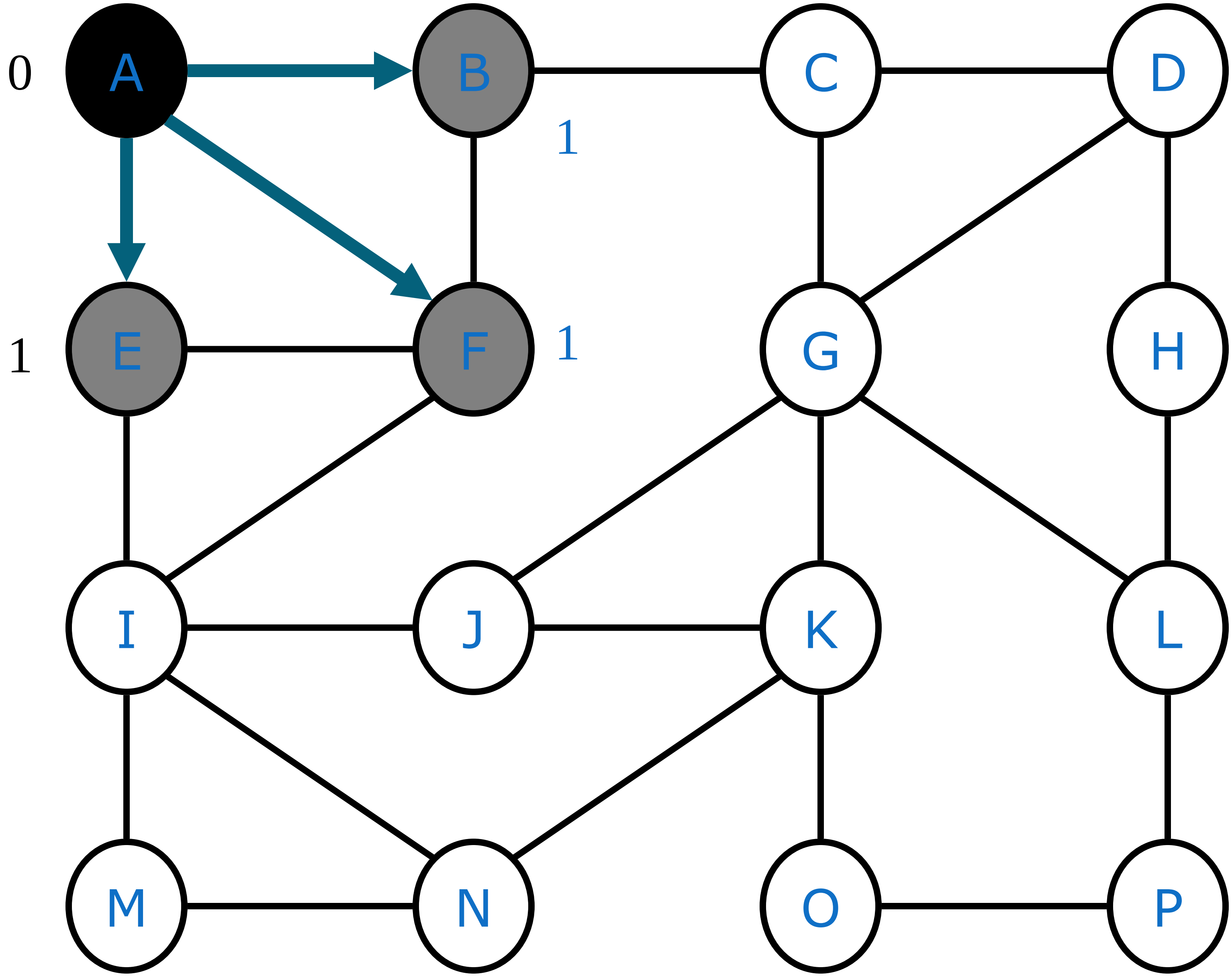


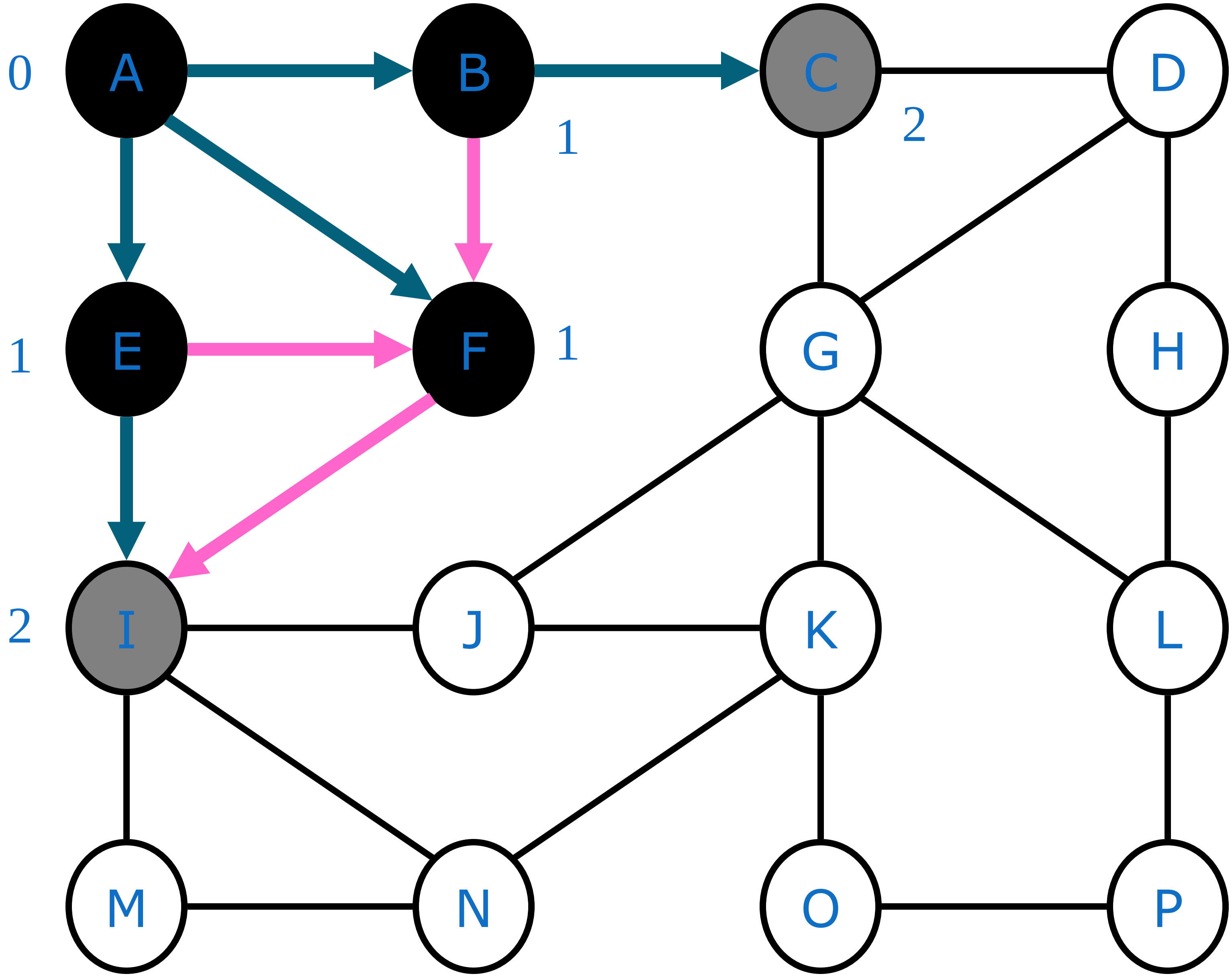
Q: Ø

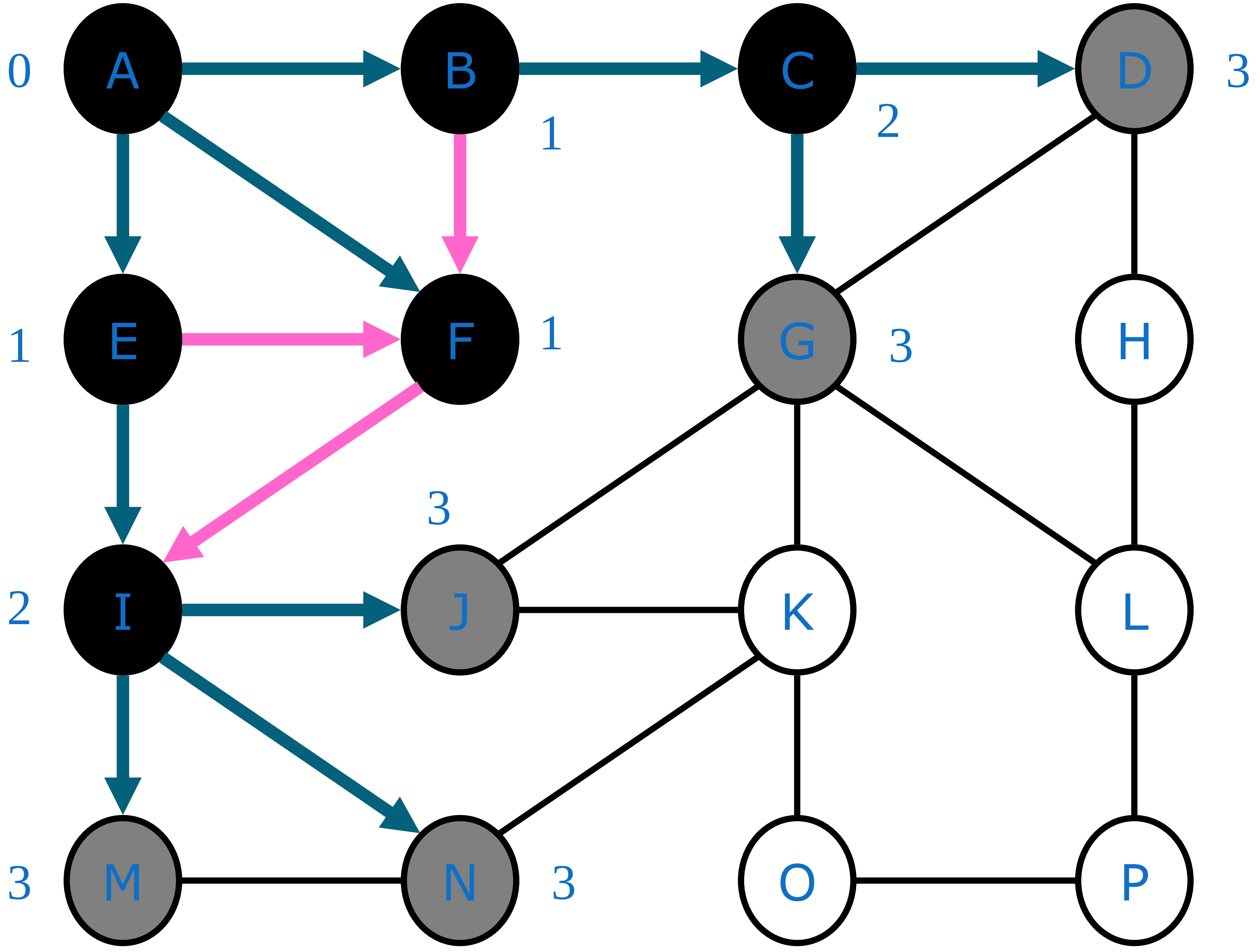
An Example



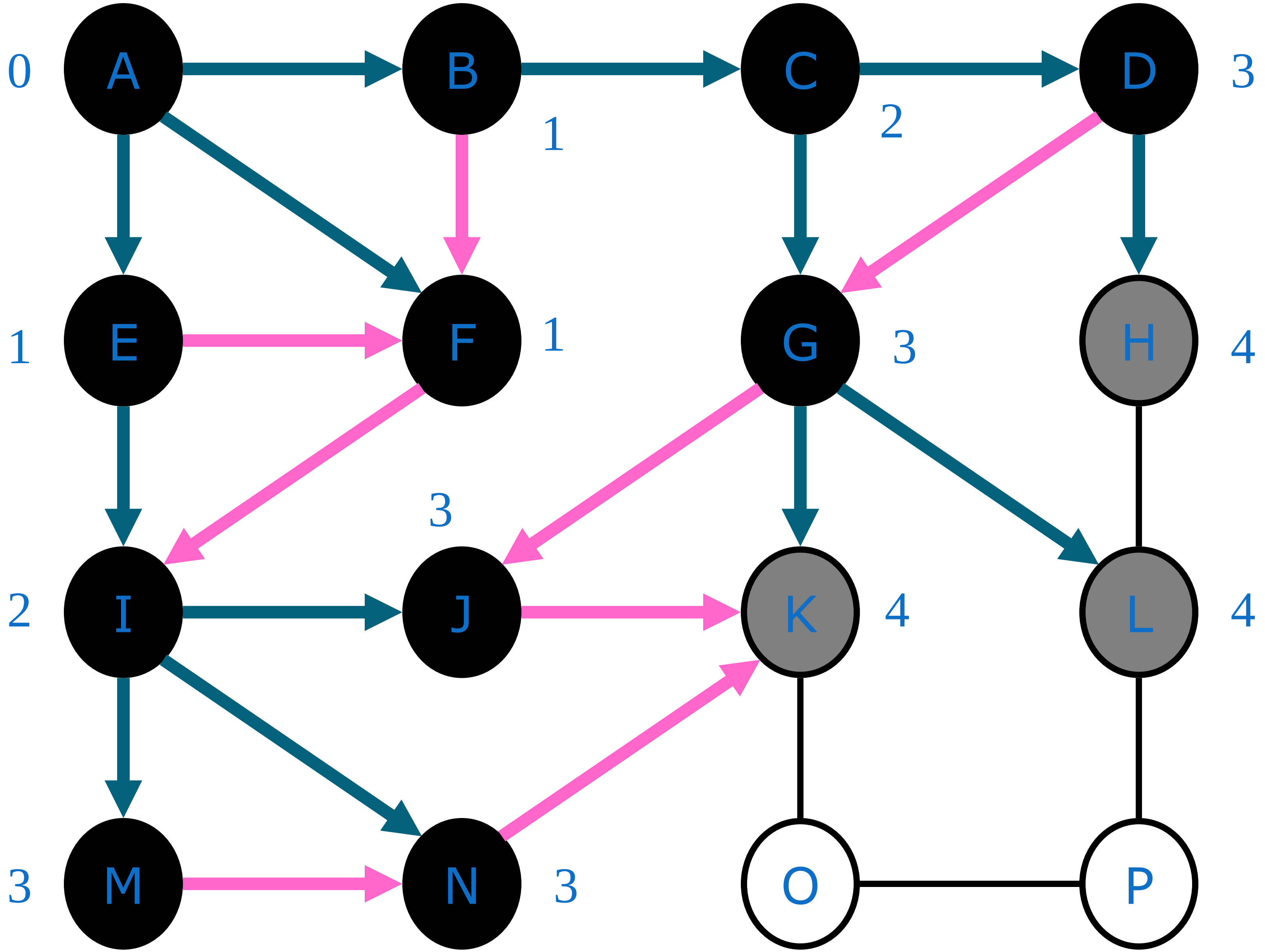

0
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P

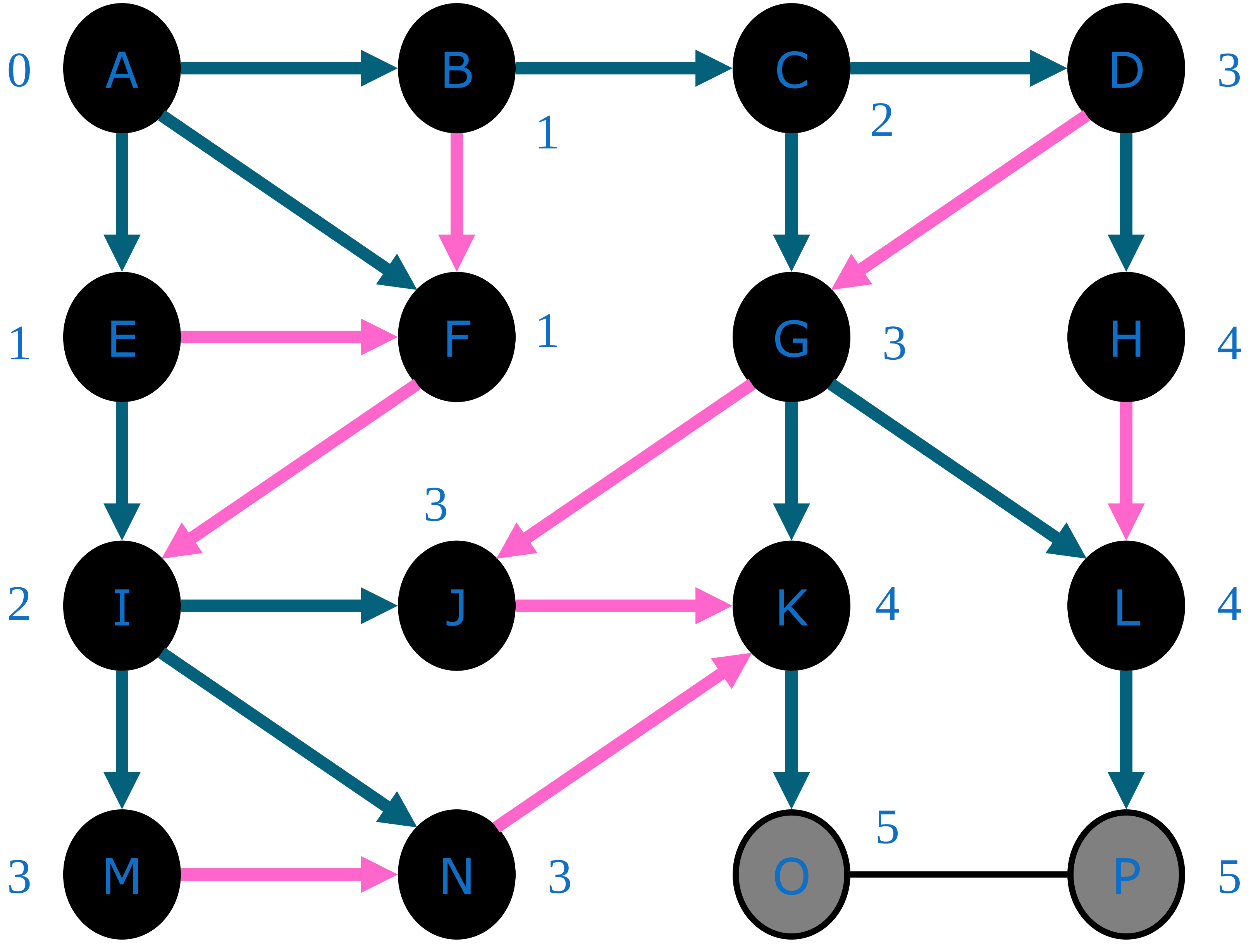

0
A
B
C
D
1
E
F
G
H
1
1
I
J
K
L
M
N
O
P

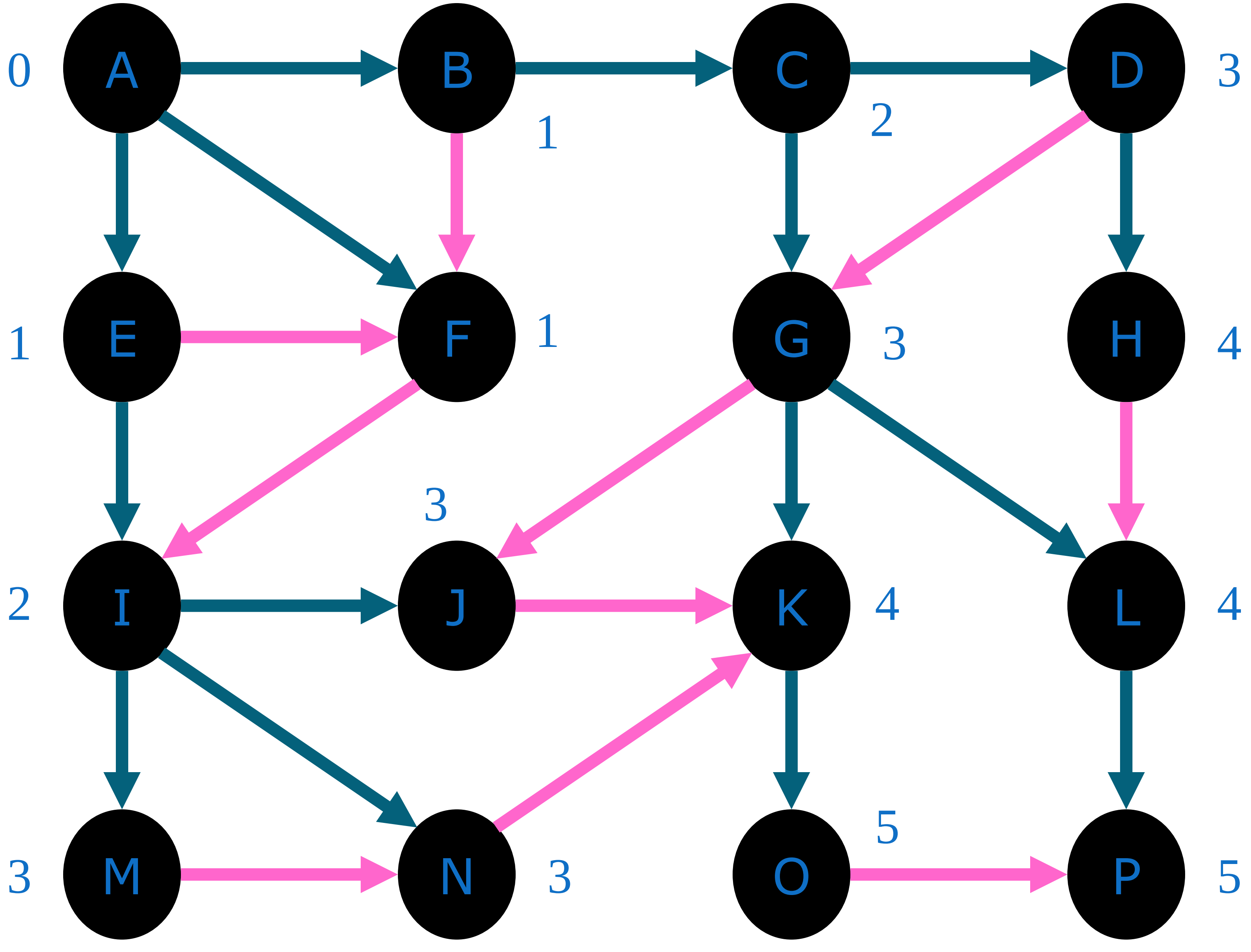

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
0
1
2
1
1
2



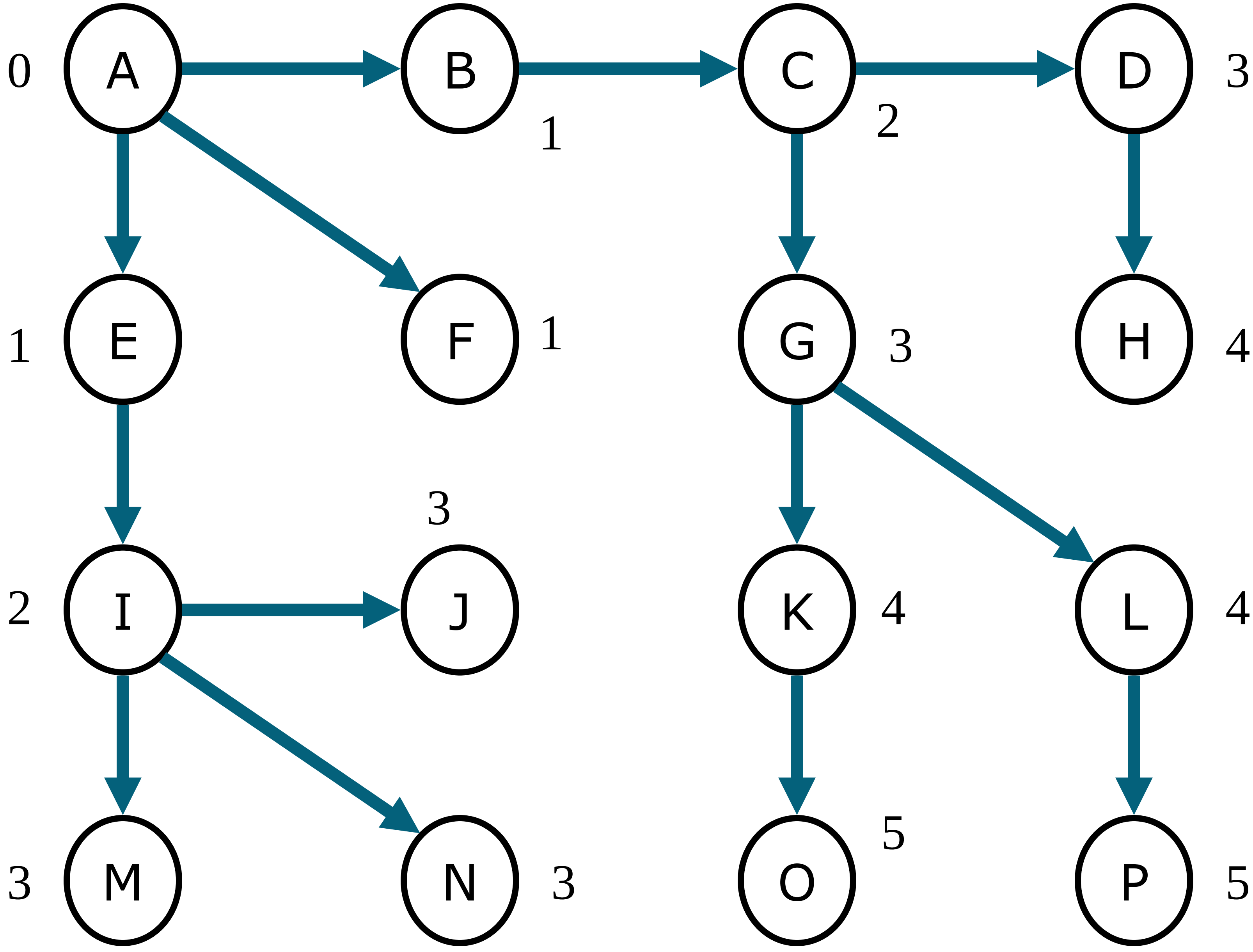
0
A
B
C
D
3
1
2
1
E
F
1
G
3
H
3
2
I
J
K
L
3
M
N
3
O
P


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
0
1
2
3
1
1
3
4
2
3
4
4
3
3


0
A
B
C
D
3
1
2
1
E
F
1
G
3
H
4
3
2
I
J
K
4
L
4
5
3
M
N
3
O
P
5



0
A
B
C
D
3
1
2
E
F
G
H
1
1
3
4
3
I
J
K
L
2
4
4
M
N
O
P
3
3
5
5


0
A
B
1
C
2
D
3
1
E
F
1
G
3
H
4
3
2
I
J
K
4
L
4
5
3
M
N
3
O
P
5

BFS: Algorithm

```
BFS(G, s)
for each vertex u in V – {s},
      color[u] = white;
      d[u] = infinity;
      p[u] = NIL
color[s] = GRAY;      d[s] = 0;      p[s] = NIL;     Q = empty queue
ENQUEUE(Q,s)
while (Q not empty)
      u = DEQUEUE(Q)
      for each v ∈ Adj[u]
         if  color[v] = WHITE
            then color[v] = GREY
                  d[v] = d[u] + 1; p[v] = u
                  ENQUEUE(Q, v);
      color[u] = BLACK;
```

BFS: Complexity Analysis

- Queuing time is O(V) and scanning all edges requires O(E)
- Overhead for initialization is O (V)
- So, total running time is O(V+E)

BFS: Application

- Shortest path problem
- GPS Navigation systems
- Broadcasting in Network
- Peer to Peer Network (BitTorrent)
- Parsing social graphs


Cathedral Pkwy
Symphony Space
Museum of the City of New York
14 min
3.1 miles
15 min
2.6 miles
92nd Street Y
American Museum of Natural History
The Metropolitan Museum of Art
The Dakota
The Loeb Boathouse
Rumsey Playfield
UPPER EAST SIDE
Central Park Zoo
Hunter College
Google
LENOX HILL


376
346
345
317
316
Asquith
Delisle
Zealandia
Rosetown
Goose L.


FACEBOOK (EXAMPLE)
Search for...
GO

END