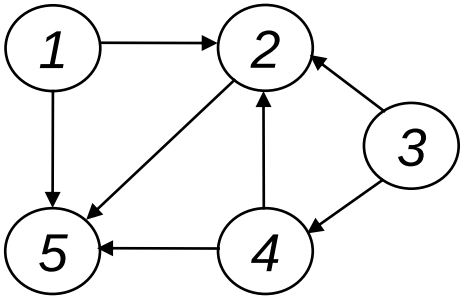
DFS

Depth-First Search

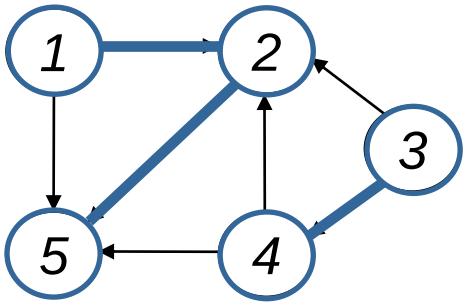
- **Input:**
  - G = (V, E) (No source vertex given!)



- **Goal**:
  - Explore the edges of G to "discover" every vertex in V starting at the most current visited node
  - Search may be repeated from multiple sources
- **Output:**
  - 2 **timestamps** on each vertex:
    - d[v] = discovery time
    - f[v] = finishing time (done with examining v's adjacency list)
  - Depth-first forest

Depth-First Search

- Search "deeper" in the graph whenever possible
- Edges are explored out of the most recently discovered vertex v that still has unexplored edges



- *After all edges of v have been explored, the search "backtracks" from the parent of v*
- *The process continues until all vertices reachable from the original source have been discovered*
- *If undiscovered vertices remain, choose one of them as a new source and repeat the search from that vertex*
- *DFS creates a "depth-first forest"*

DFS Additional Data Structures

- Global variable: time-stamp
  - Incremented when nodes are discovered or finished
- color[u] – similar to BFS
  - White before discovery, gray while processing and black when finished processing
- prev[u] – predecessor of u
- d[u], f[u] – discovery and finish times

$$1 \le d[u] < f[u] \le 2|V|$$

Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{                                  Initialize
   for each vertex u ∈ V
   {
      color[u] = WHITE;
       prev[u]=NIL;
       f[u]=inf; d[u]=inf;
   }
   time = 0;
   for each vertex u ∈ V
     if (color[u] == WHITE)
         DFS_Visit(u);
}
```

```
DFS_Visit(u)
{
   color[u] = GREY;
   time = time+1;
   d[u] = time;
   for each v ∈ Adj[u]
   {
      if(color[v] == WHITE){
         prev[v]=u;
         DFS_Visit(v);}
   }
   color[u] = BLACK;
   time = time+1;
   f[u] = time;
}
```

Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
       color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
       if(color[v] == WHITE){
          prev[v]=u;
          DFS_Visit(v);}
    }
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
       color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
       if(color[v] == WHITE){
          prev[v]=u;
          DFS_Visit(v);}
    }
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

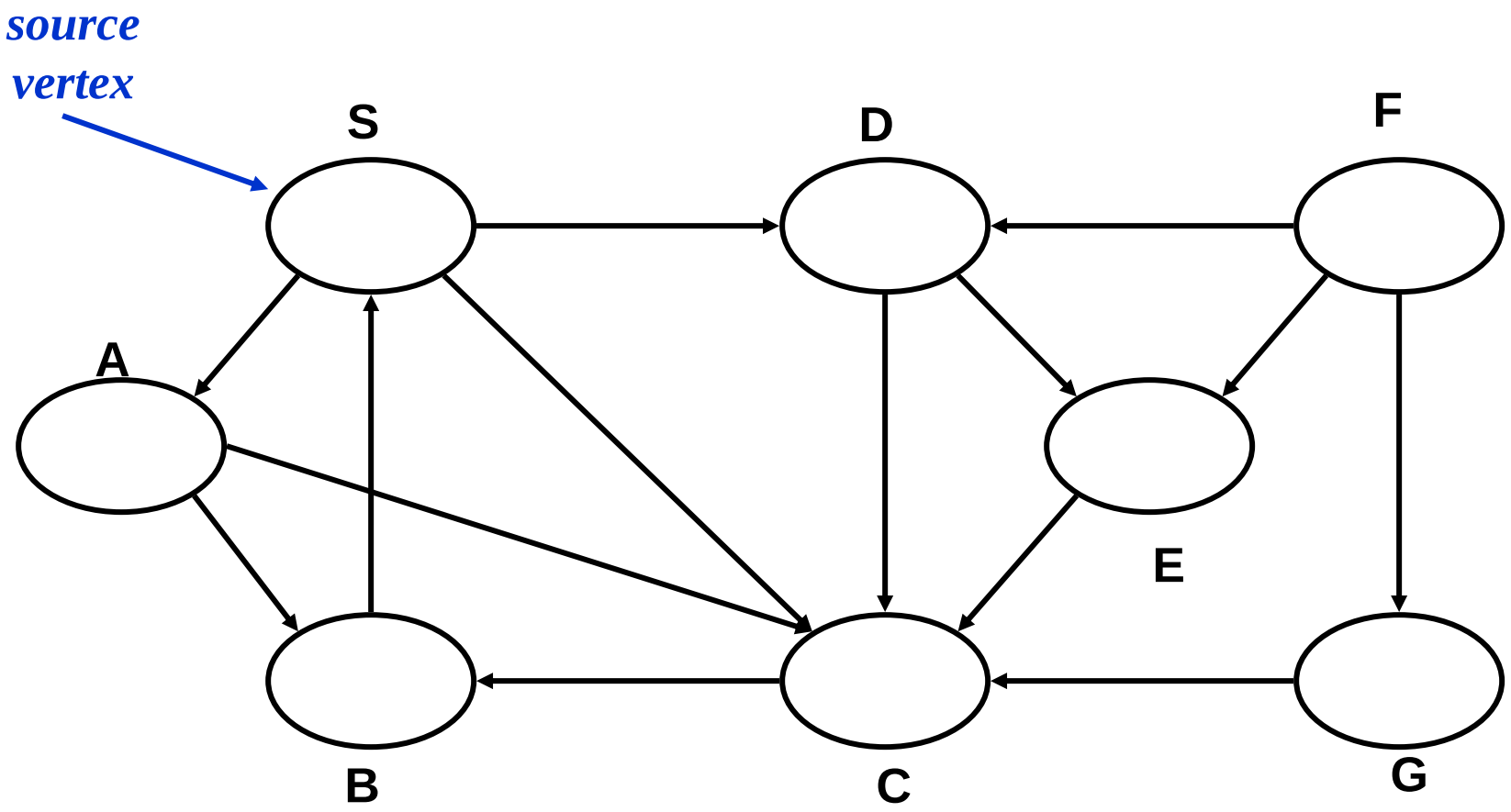
Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
       color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

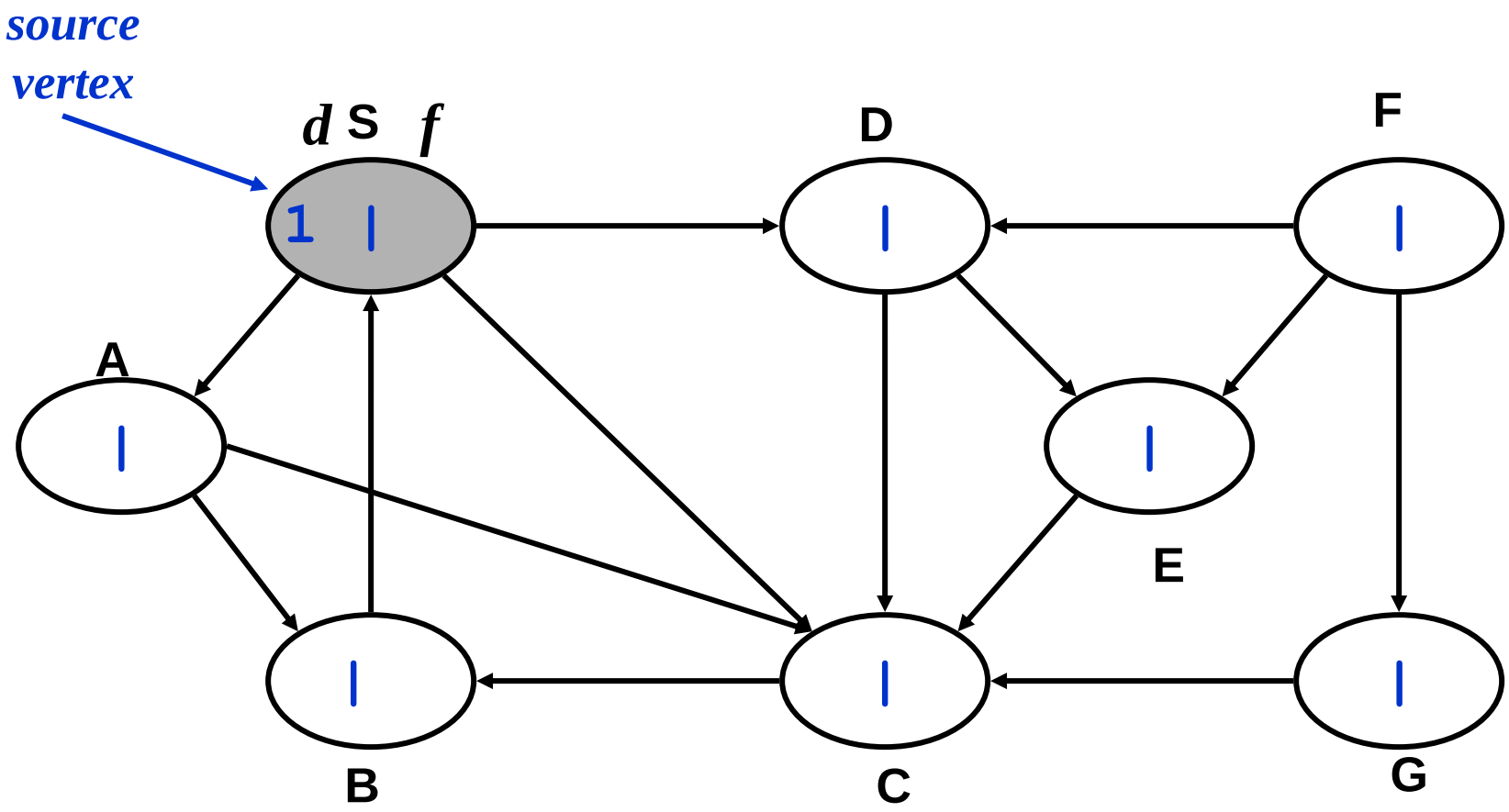
```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
       if(color[v] == WHITE){
          prev[v]=u;
          DFS_Visit(v);
    }}
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

Will all vertices eventually be colored black?

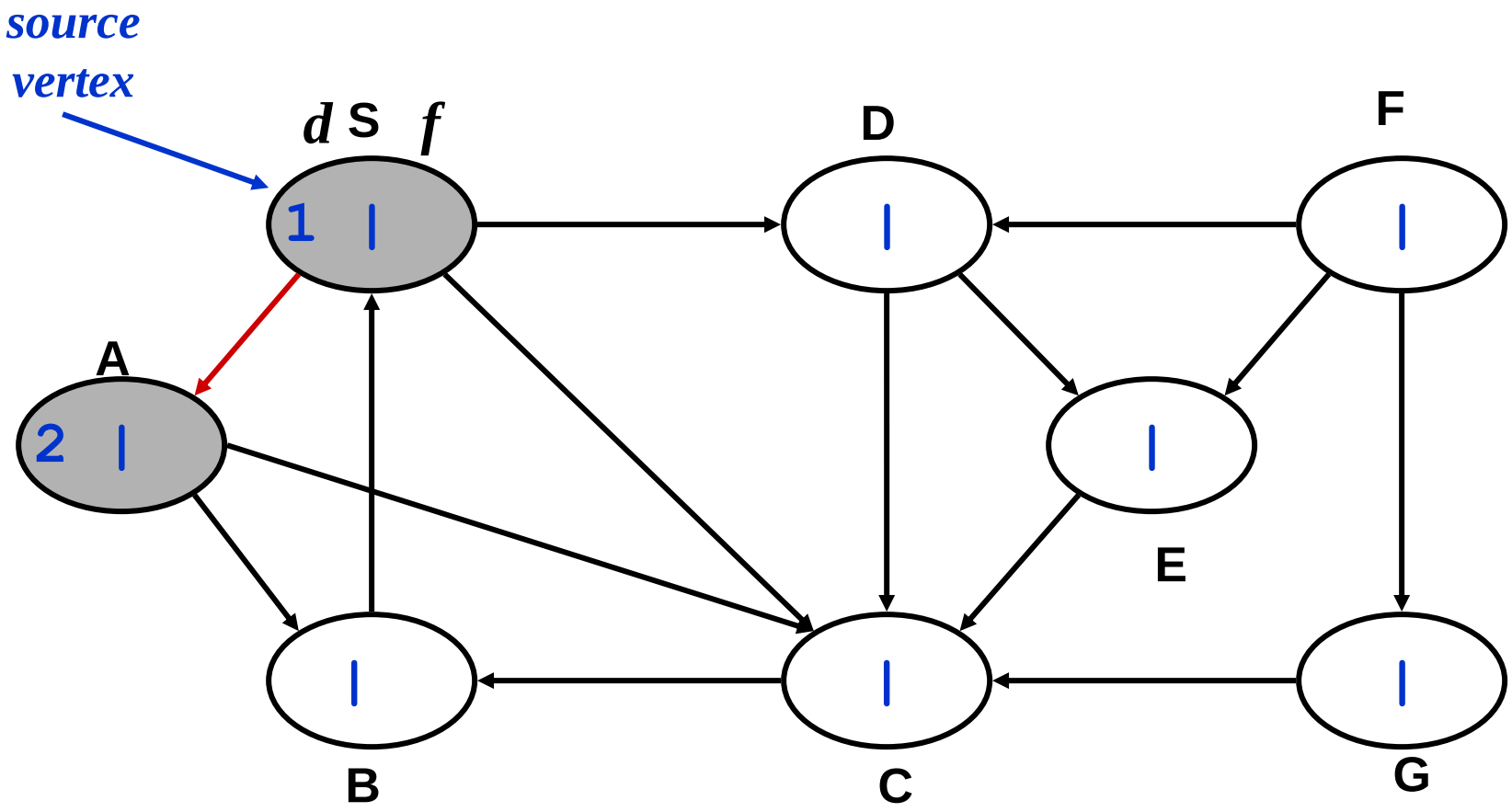
DFS Example



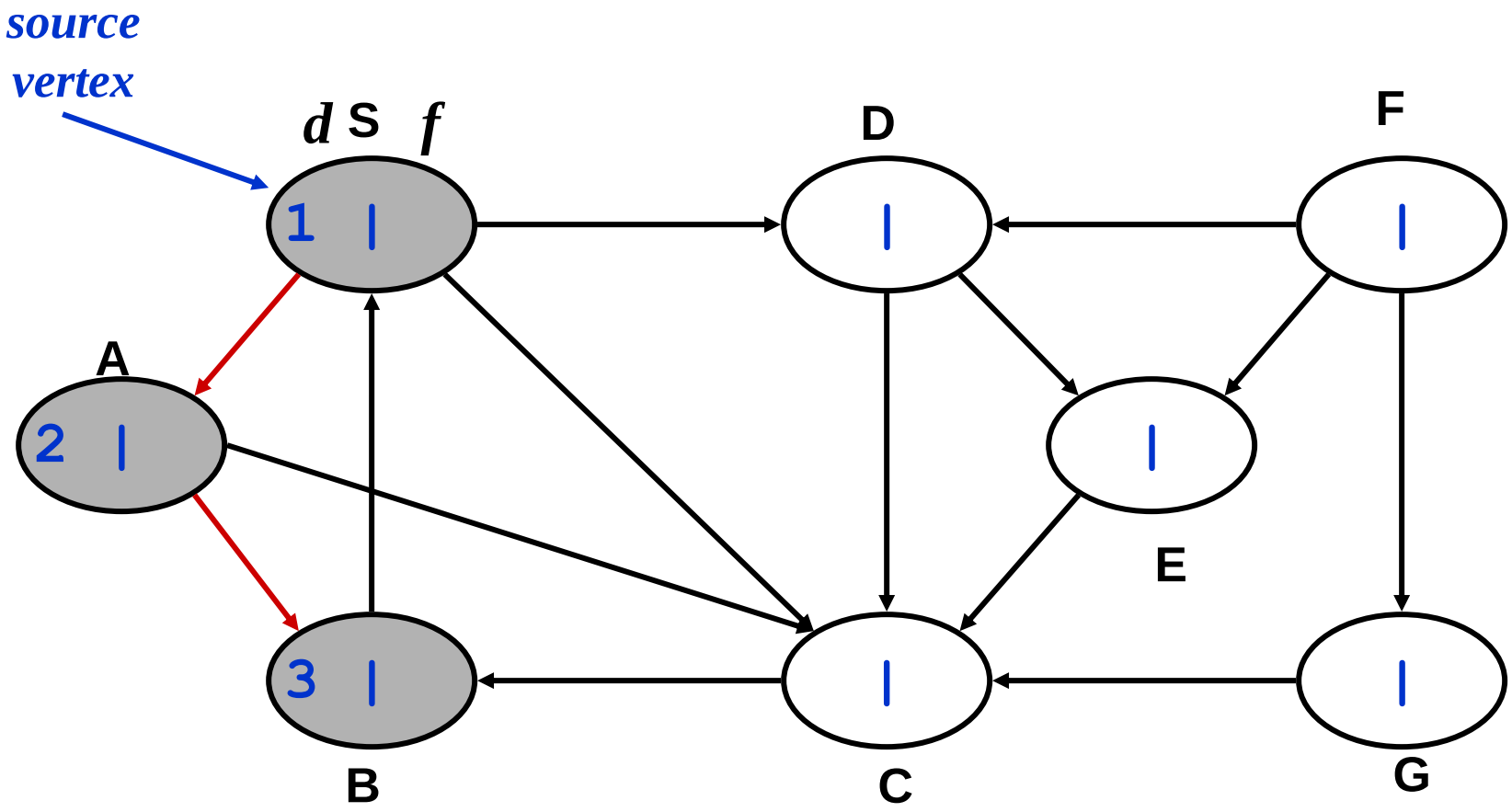
DFS Example



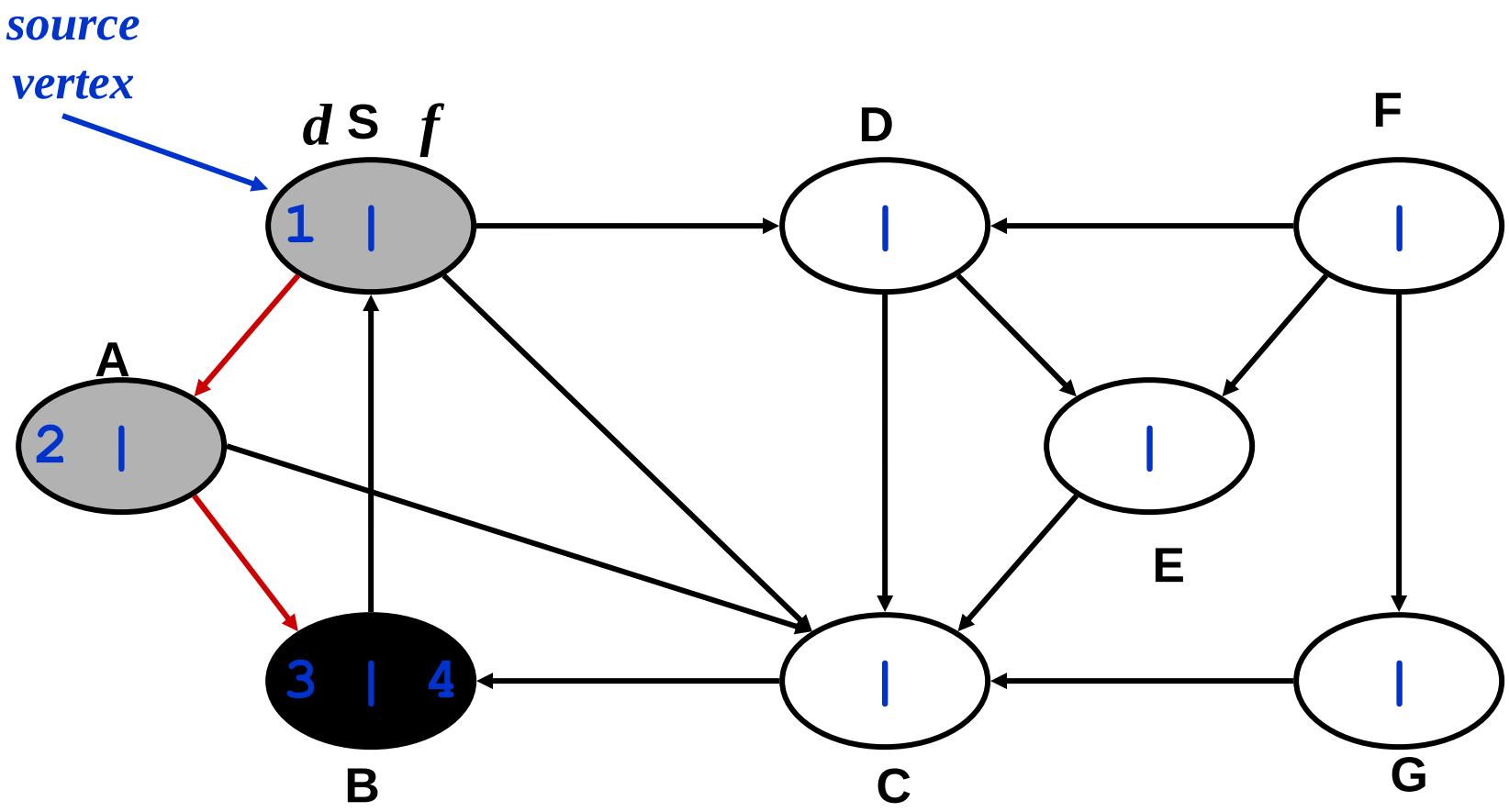
DFS Example



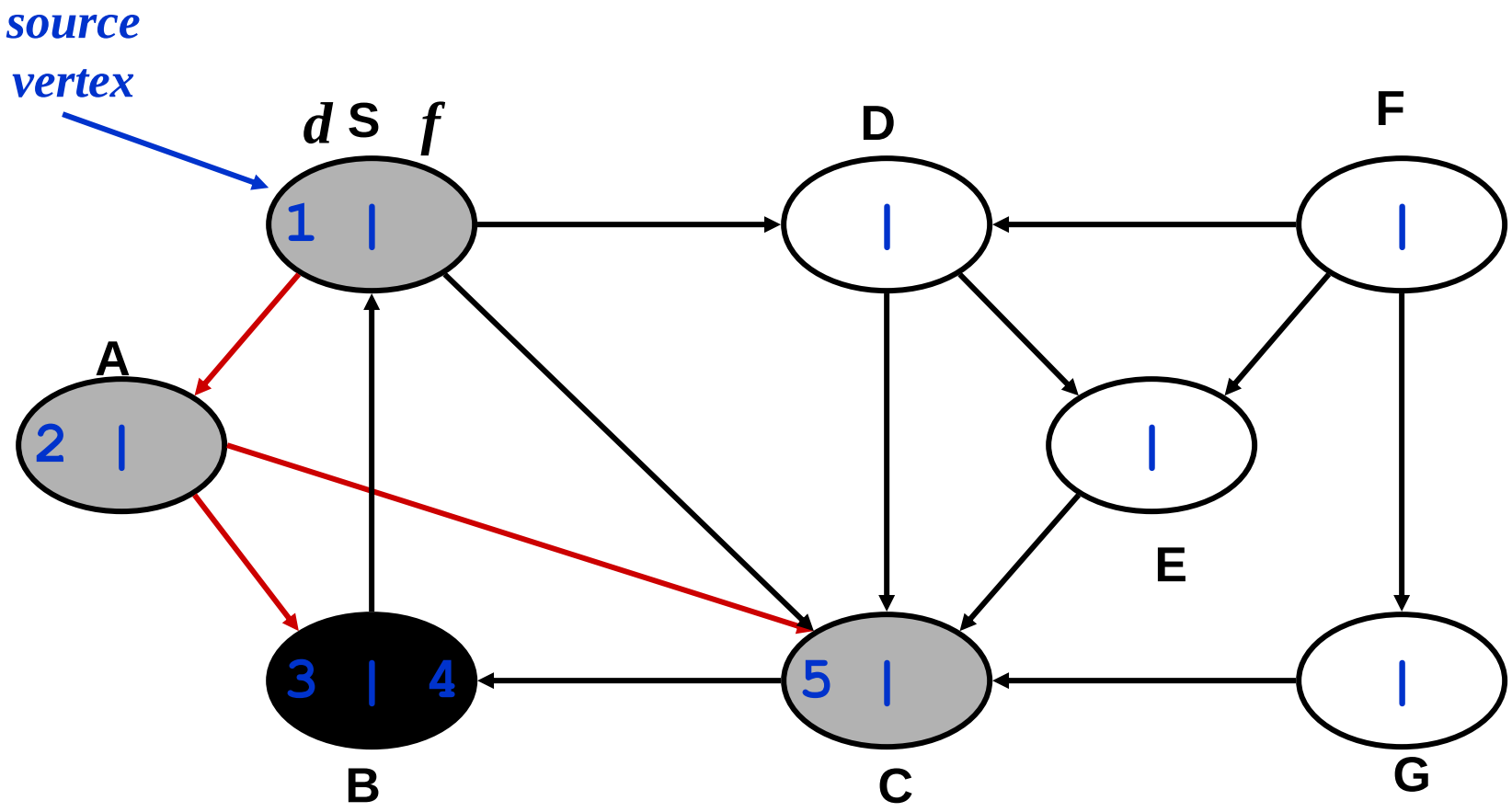
DFS Example



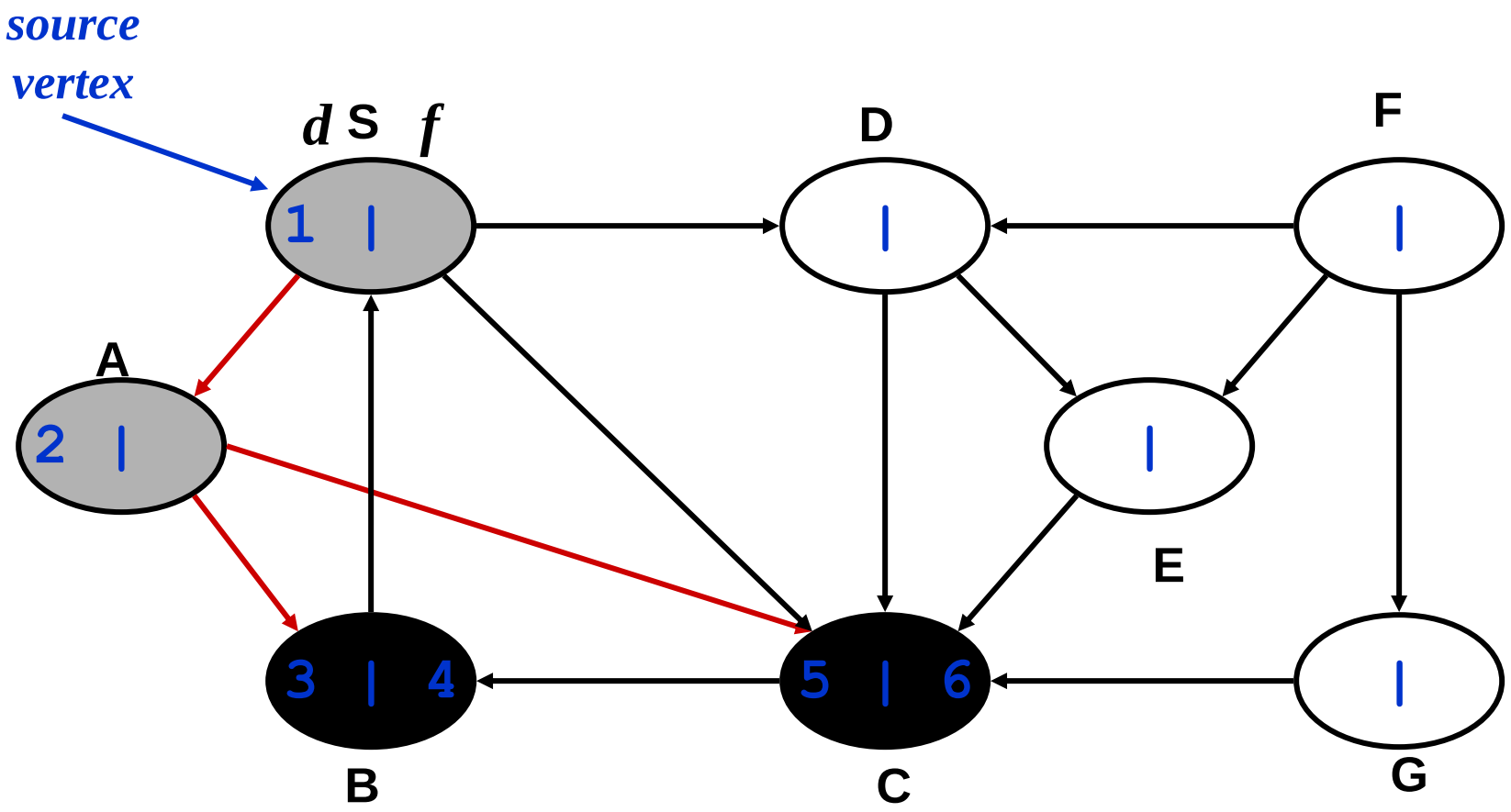
DFS Example



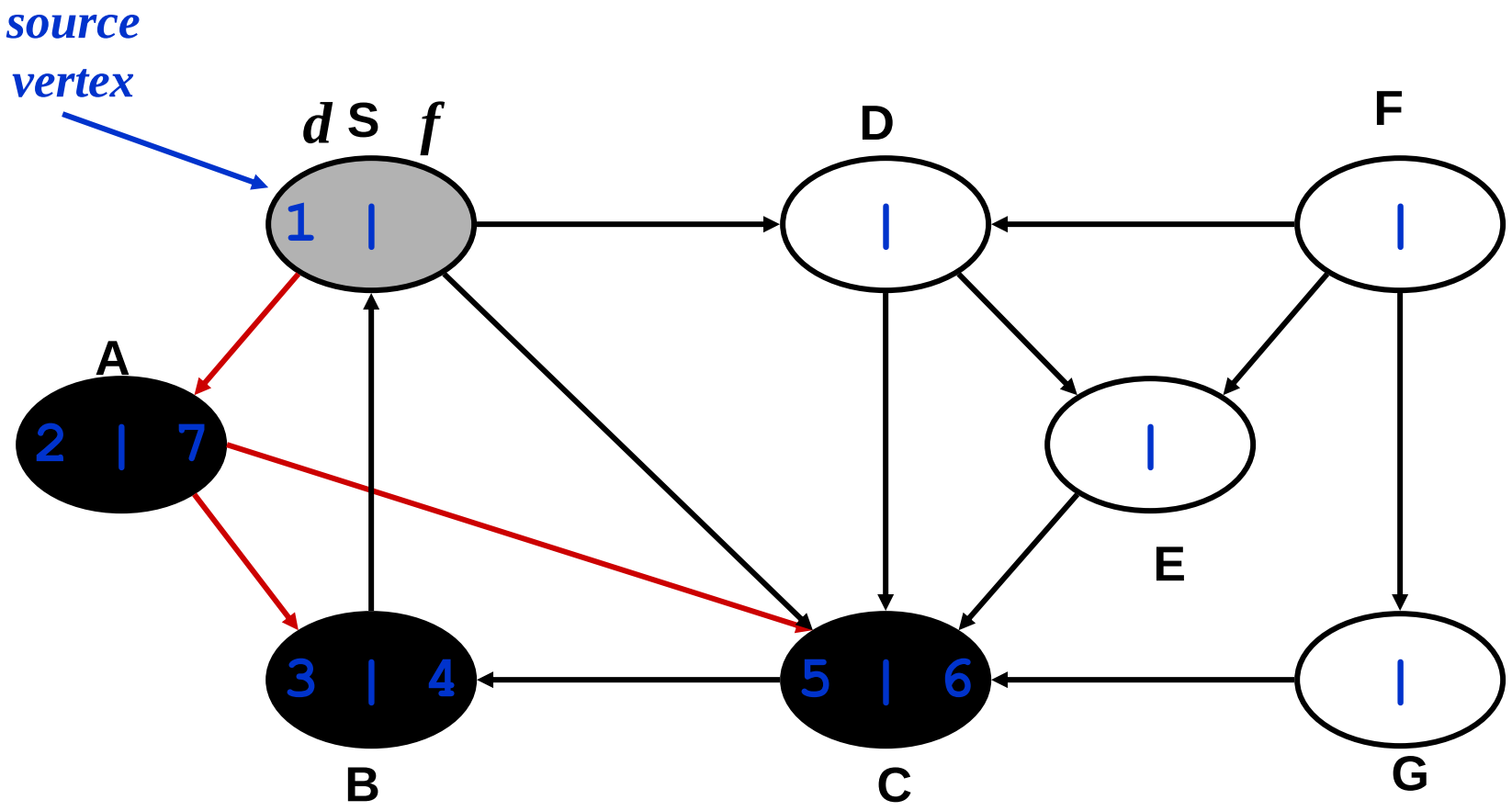
DFS Example



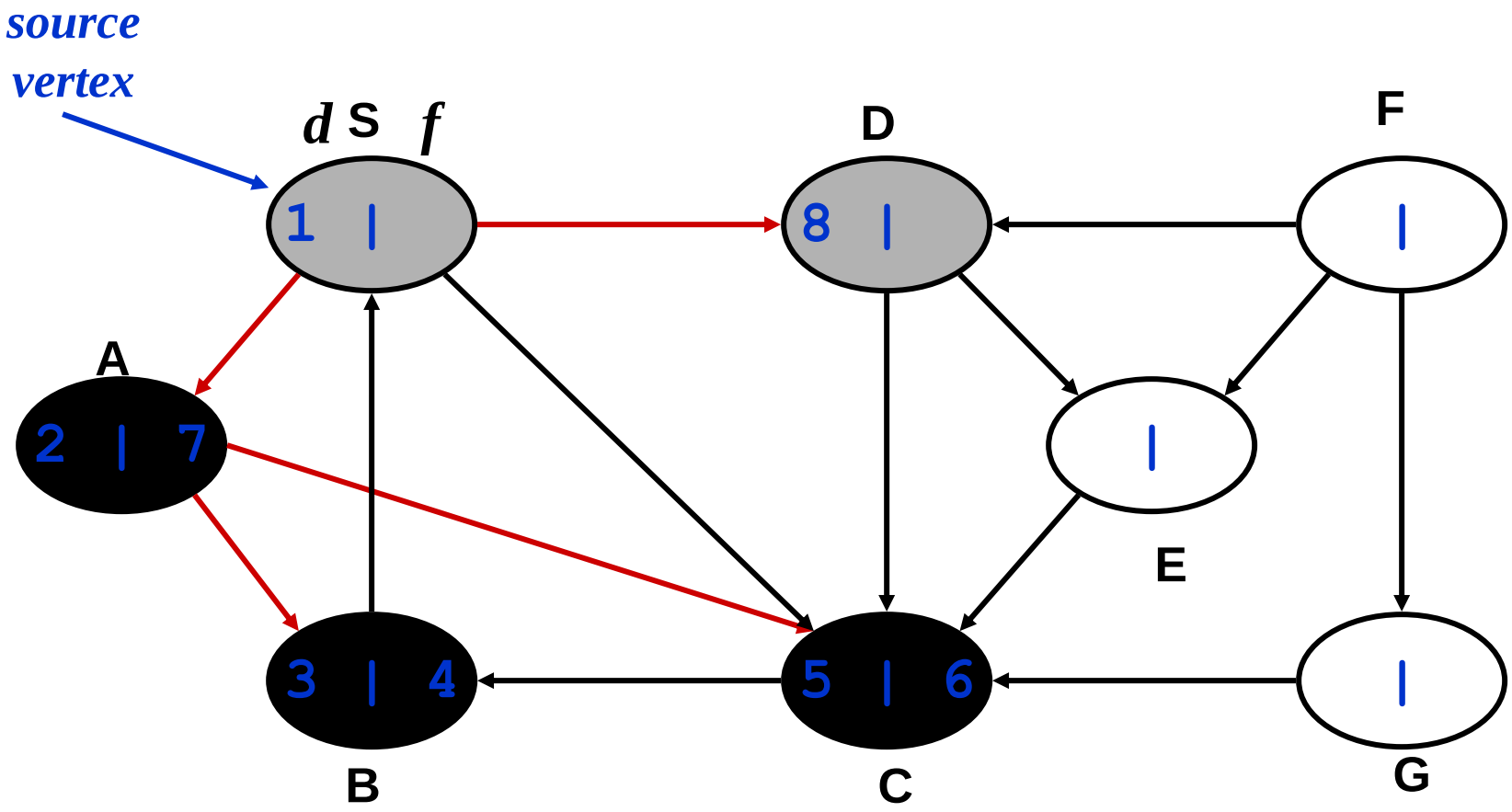
DFS Example



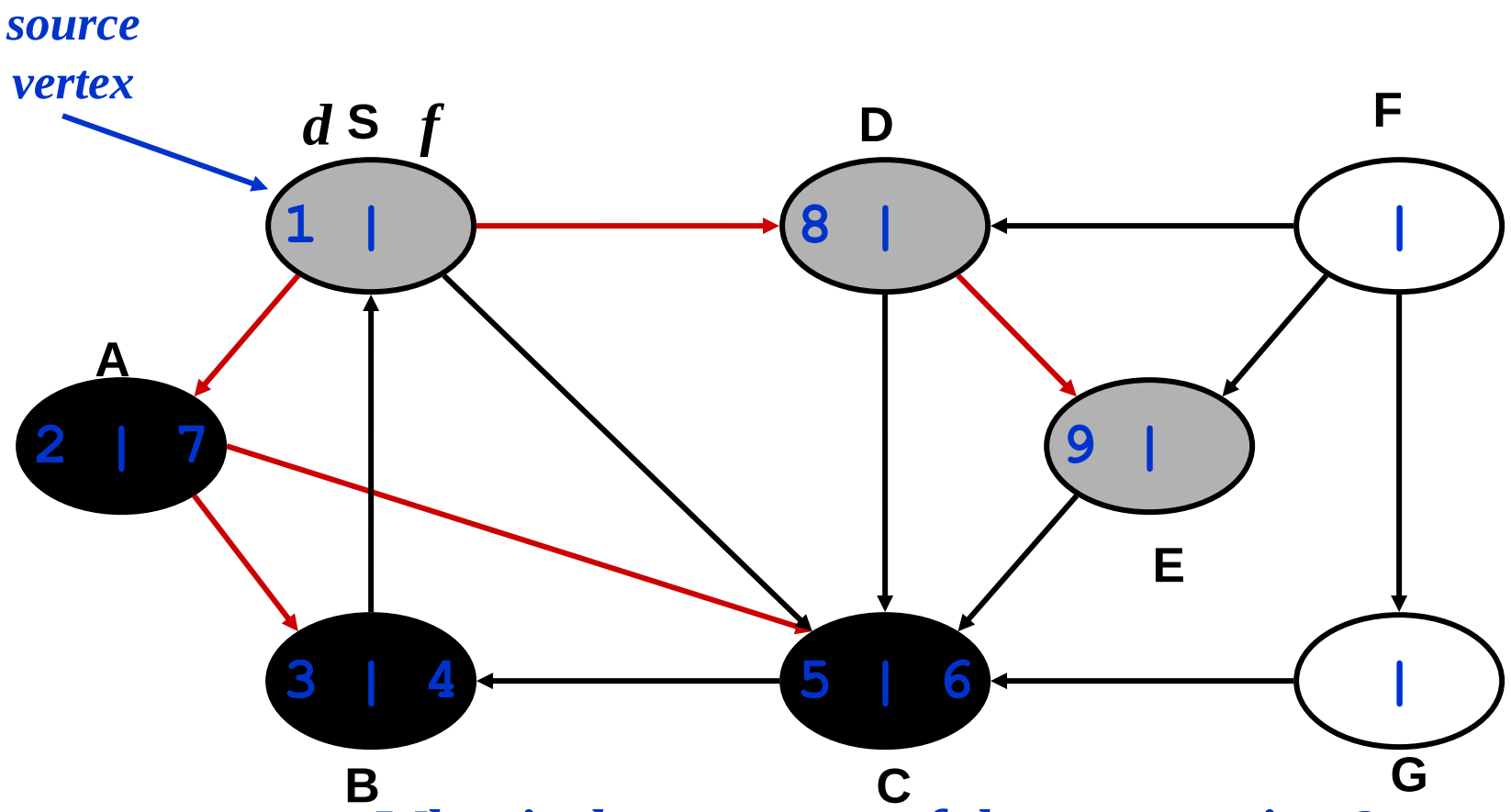
DFS Example



DFS Example

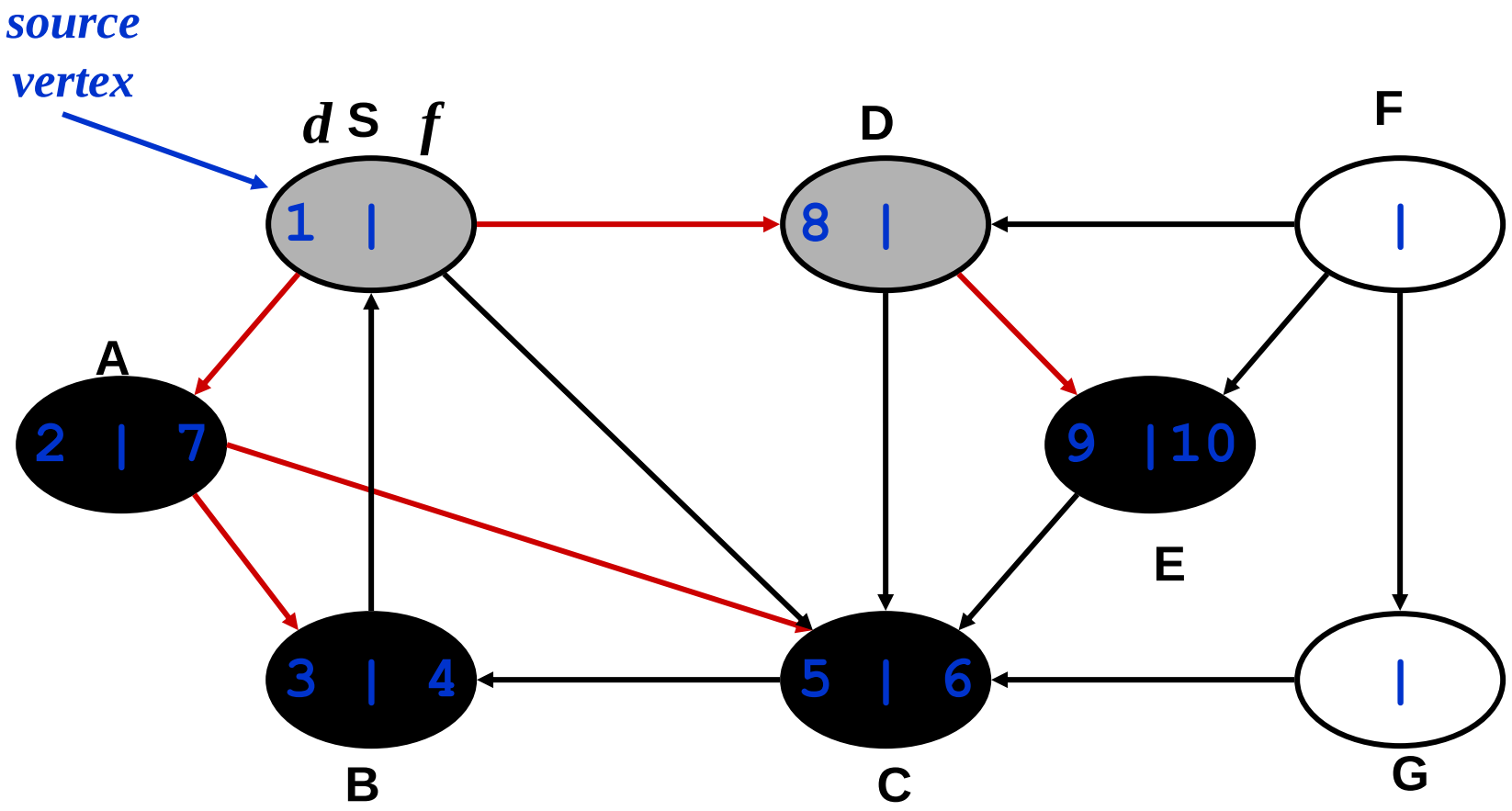


DFS Example

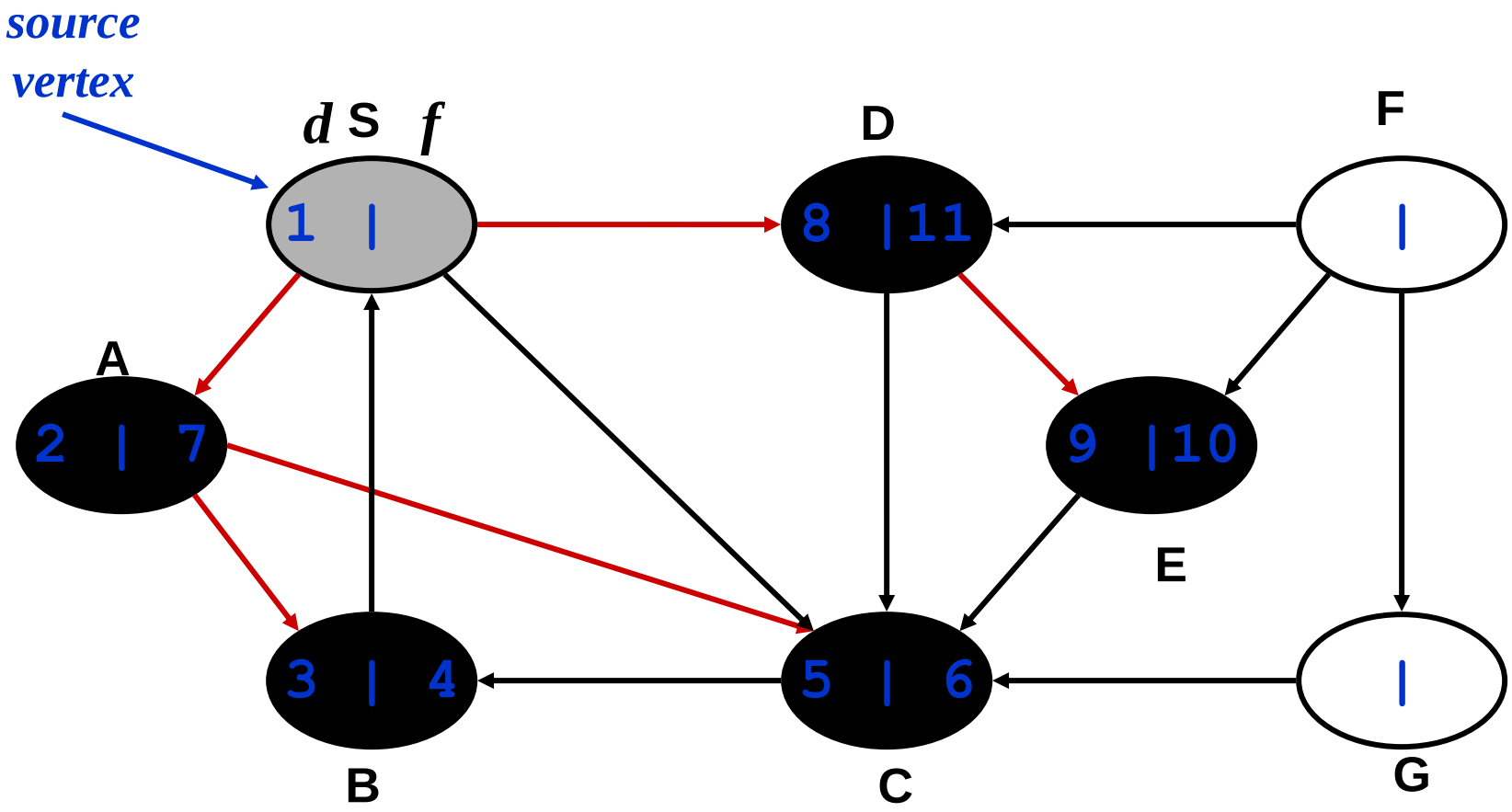


What is the structure of the grey vertices?
What do they represent?

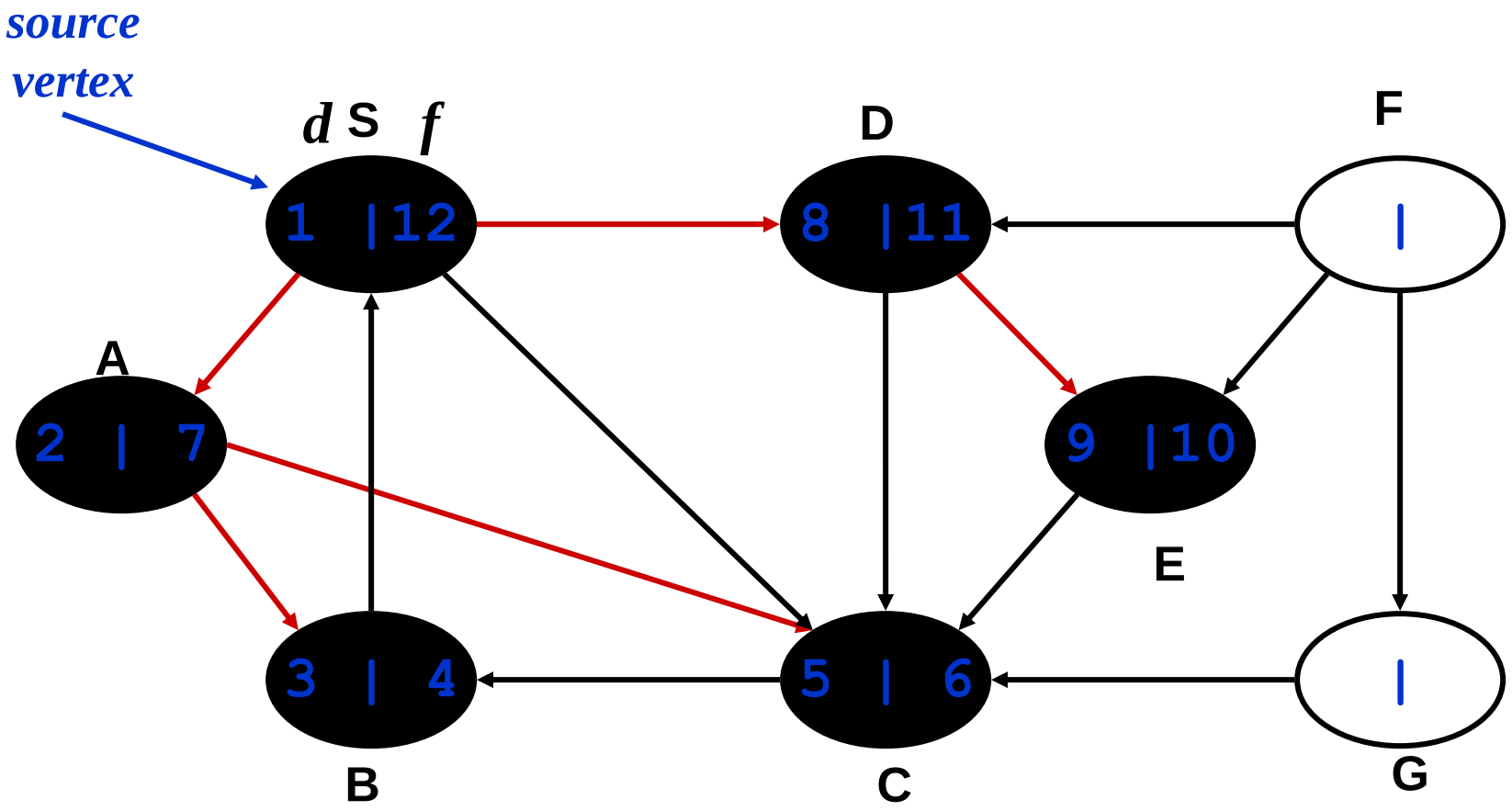
DFS Example



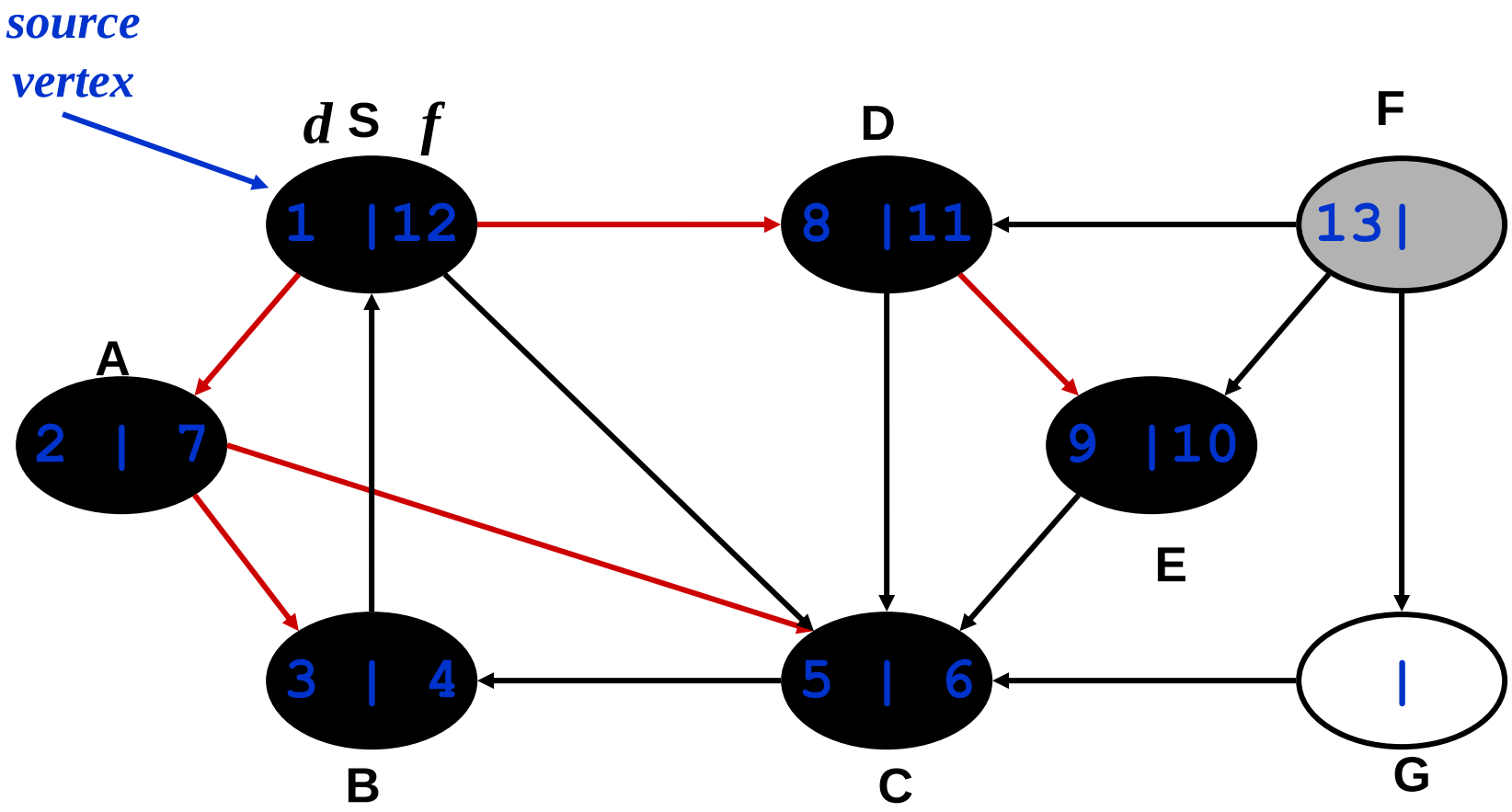
DFS Example



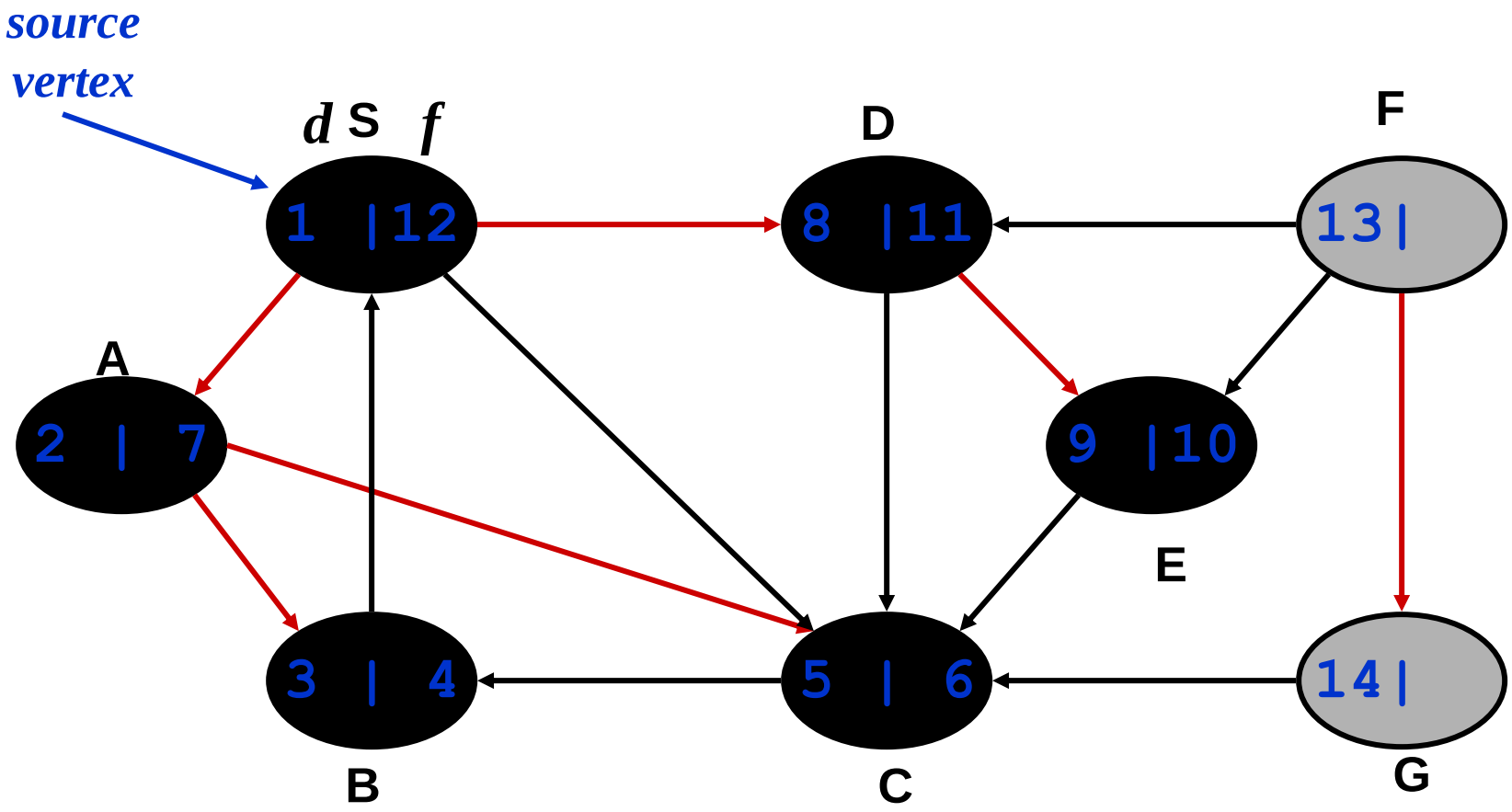
DFS Example



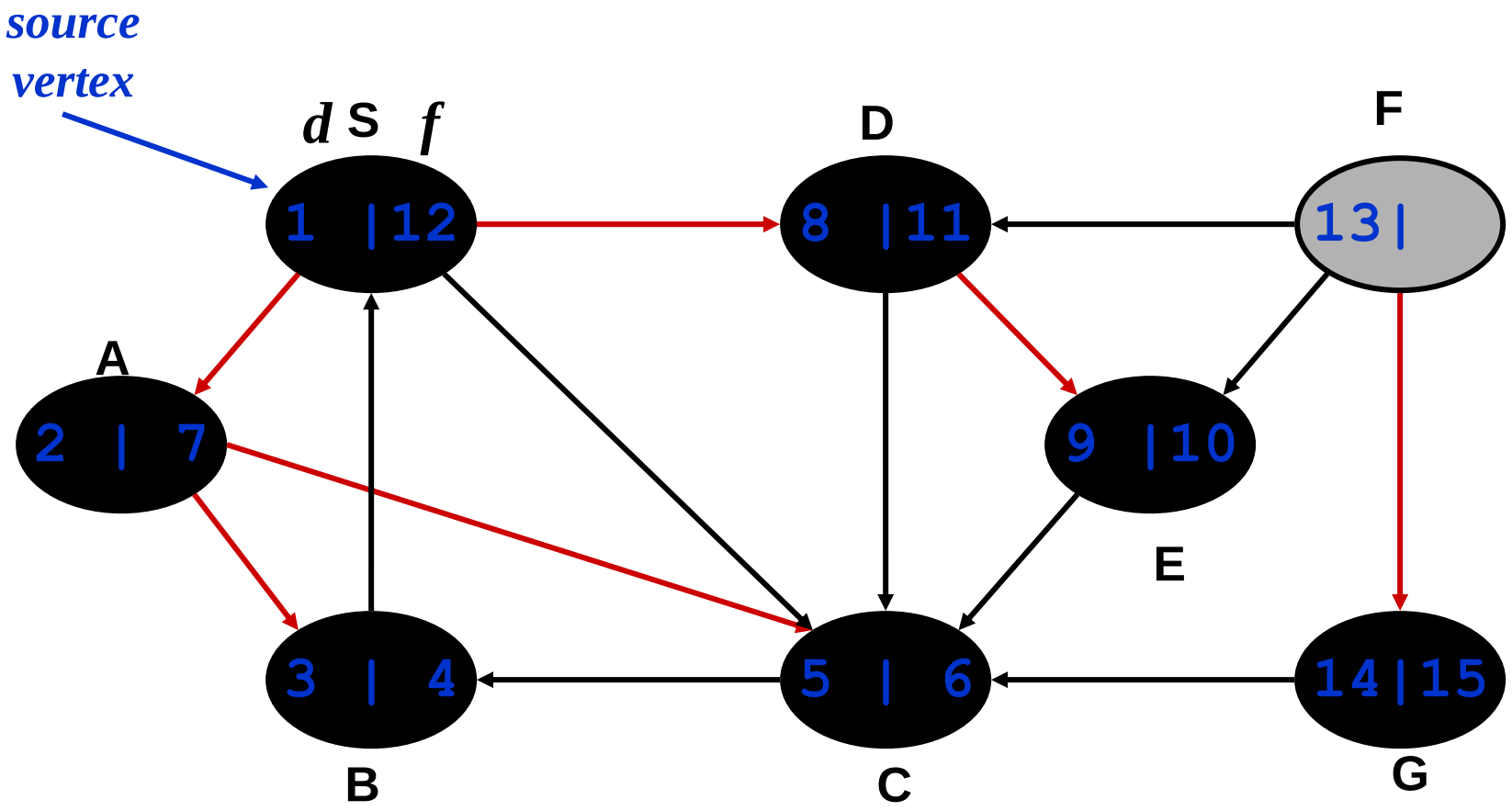
DFS Example



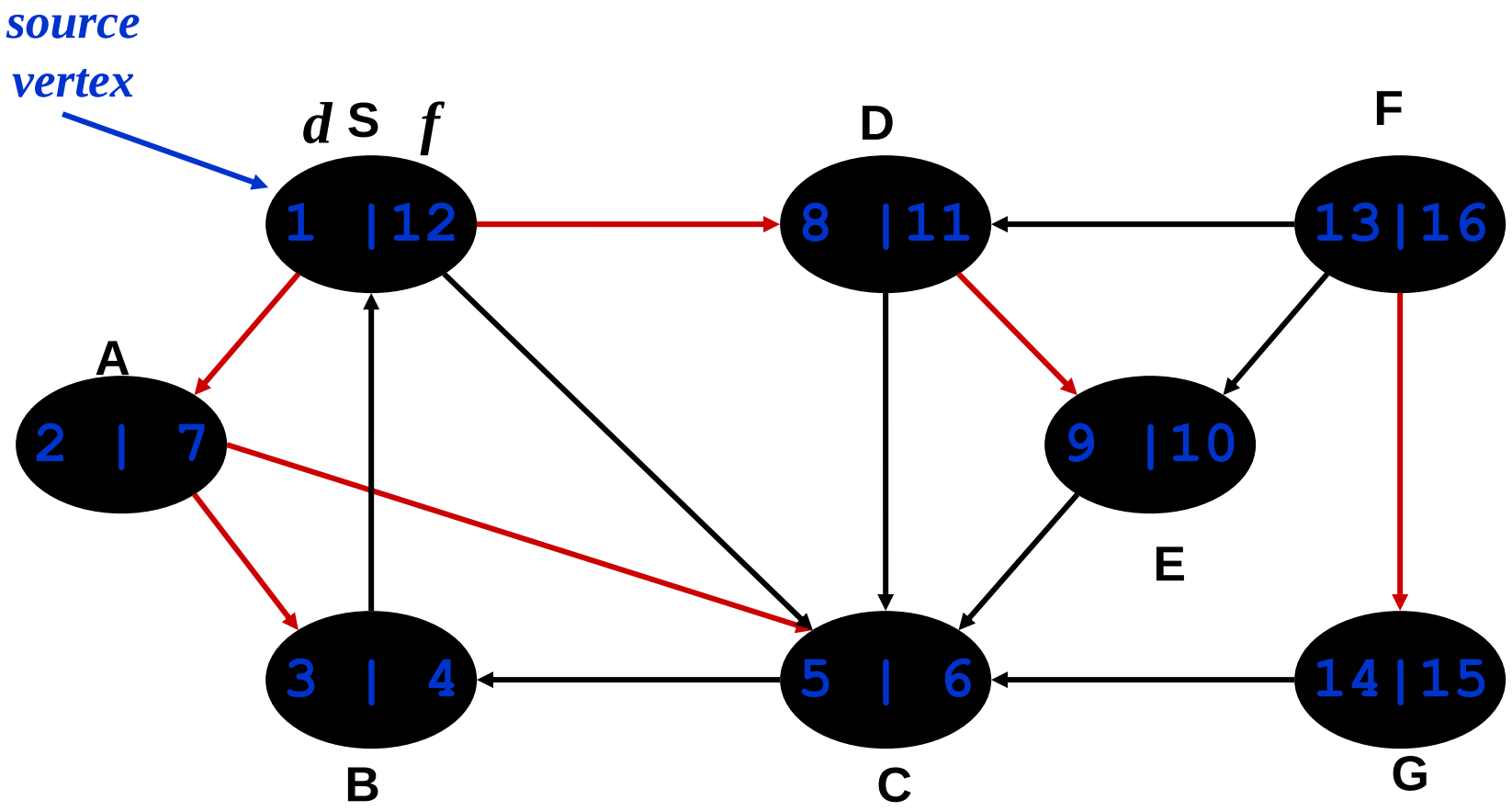
DFS Example



DFS Example



DFS Example



Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
       color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
       if (color[v] == WHITE)
          prev[v]=u;
          DFS_Visit(v);
    }
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

What will be the running time?

Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
        color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

O(V)

O(V)

```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
        if (color[v] == WHITE)
            prev[v]=u;
            DFS_Visit(v);
    }
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

O(V)

Running time: $O(V^2)$ because call DFS_Visit on each vertex, and the loop over Adj[] can run as many as $|V|$ times

Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
       color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
       if (color[v] == WHITE)
          prev[v]=u;
          DFS_Visit(v);
    }
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

BUT, there is actually a tighter bound.

How many times will DFS_Visit() actually be called?

Depth-First Search: The Code

```
    Data: color[V], time,
       prev[V],d[V], f[V]
DFS(G) // where prog starts
{
    for each vertex u ∈ V
    {
       color[u] = WHITE;
        prev[u]=NIL;
        f[u]=inf; d[u]=inf;
    }
    time = 0;
    for each vertex u ∈ V
      if (color[u] == WHITE)
          DFS_Visit(u);
}
```

```
DFS_Visit(u)
{
    color[u] = GREY;
    time = time+1;
    d[u] = time;
    for each v ∈ Adj[u]
    {
       if (color[v] == WHITE)
          prev[v]=u;
          DFS_Visit(v);
    }
    color[u] = BLACK;
    time = time+1;
    f[u] = time;
}
```

So, running time of DFS = O(V+E)

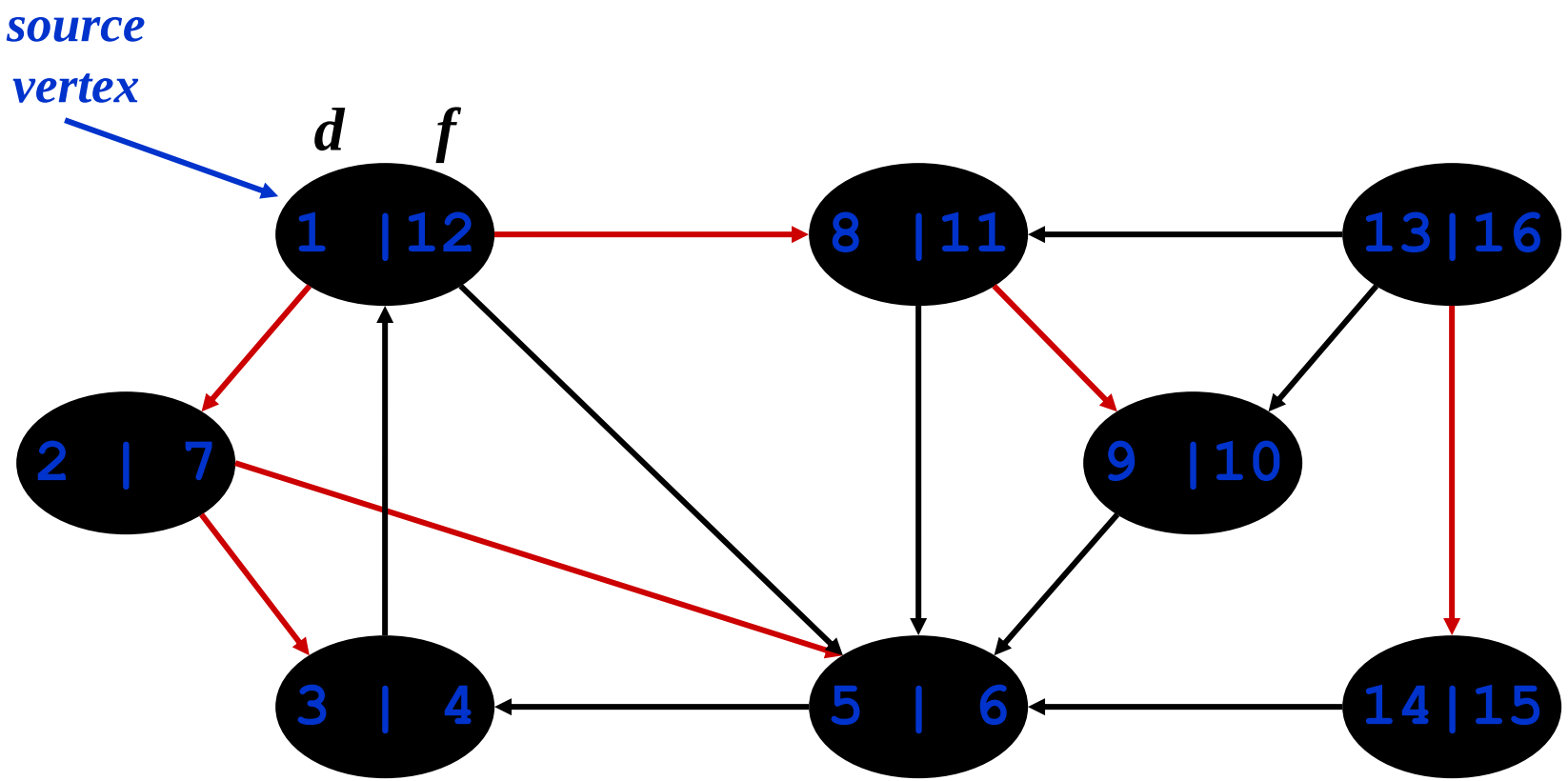
Depth-First Sort Analysis

- This running time argument is an informal example of *amortized analysis*
  - "Charge" the exploration of edge to the edge:
    - Each loop in DFS_Visit can be attributed to an edge in the graph
    - Runs once per edge if directed graph, twice if undirected
    - Thus loop will run in O(E) time, algorithm O(V+E)
      - Considered linear for graph, b/c adj list requires O(V+E) storage
  - Important to be comfortable with this kind of reasoning and analysis

DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  - *Tree edge*: encounter new (white) vertex
    - The tree edges form a spanning forest
    - *Can tree edges form cycles? Why or why not?*
      - *No*

DFS Example

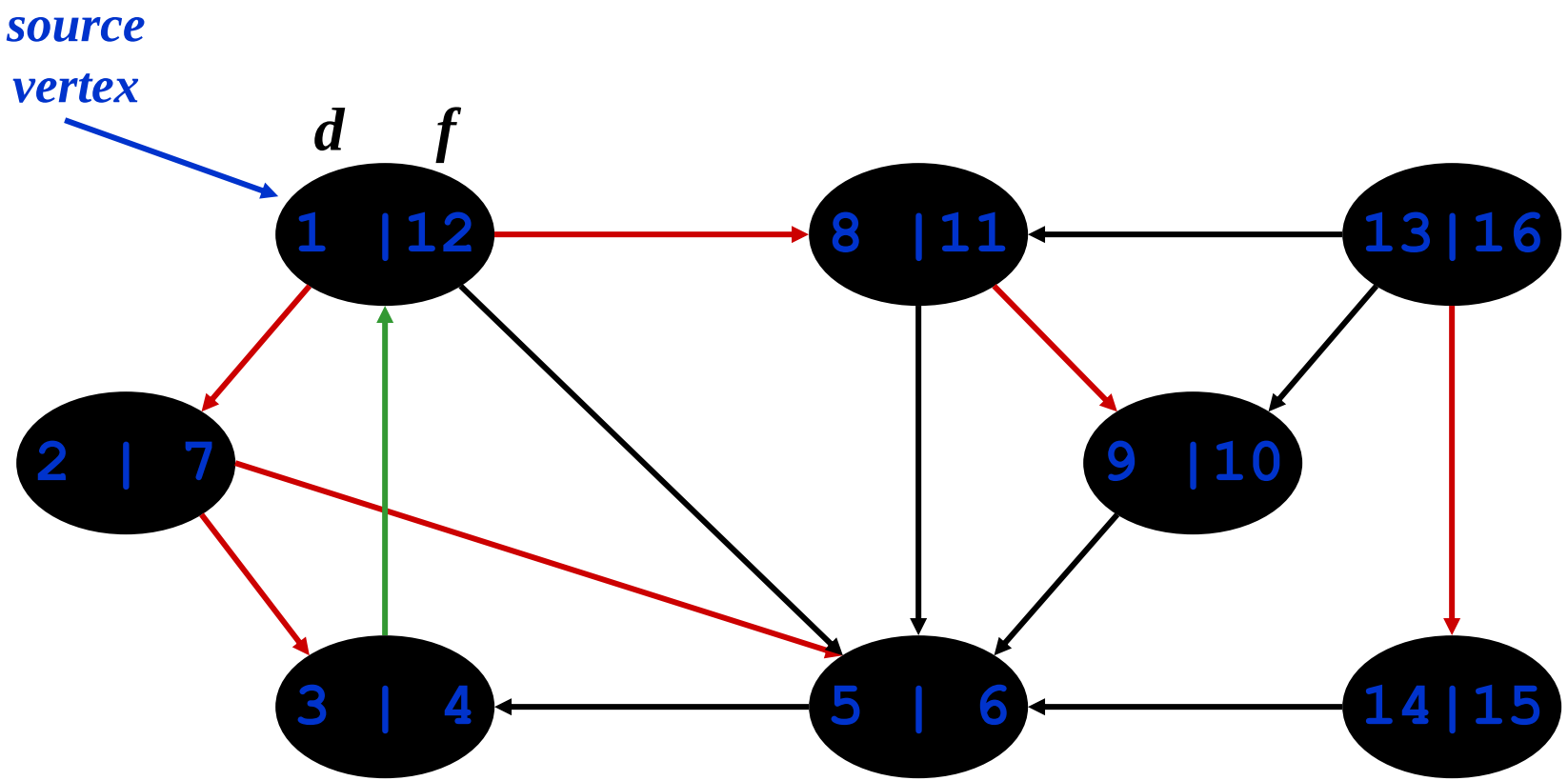


Tree edges

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
    - Encounter a grey vertex (grey to grey)
    - Self loops are considered as to be back edge.

DFS Example

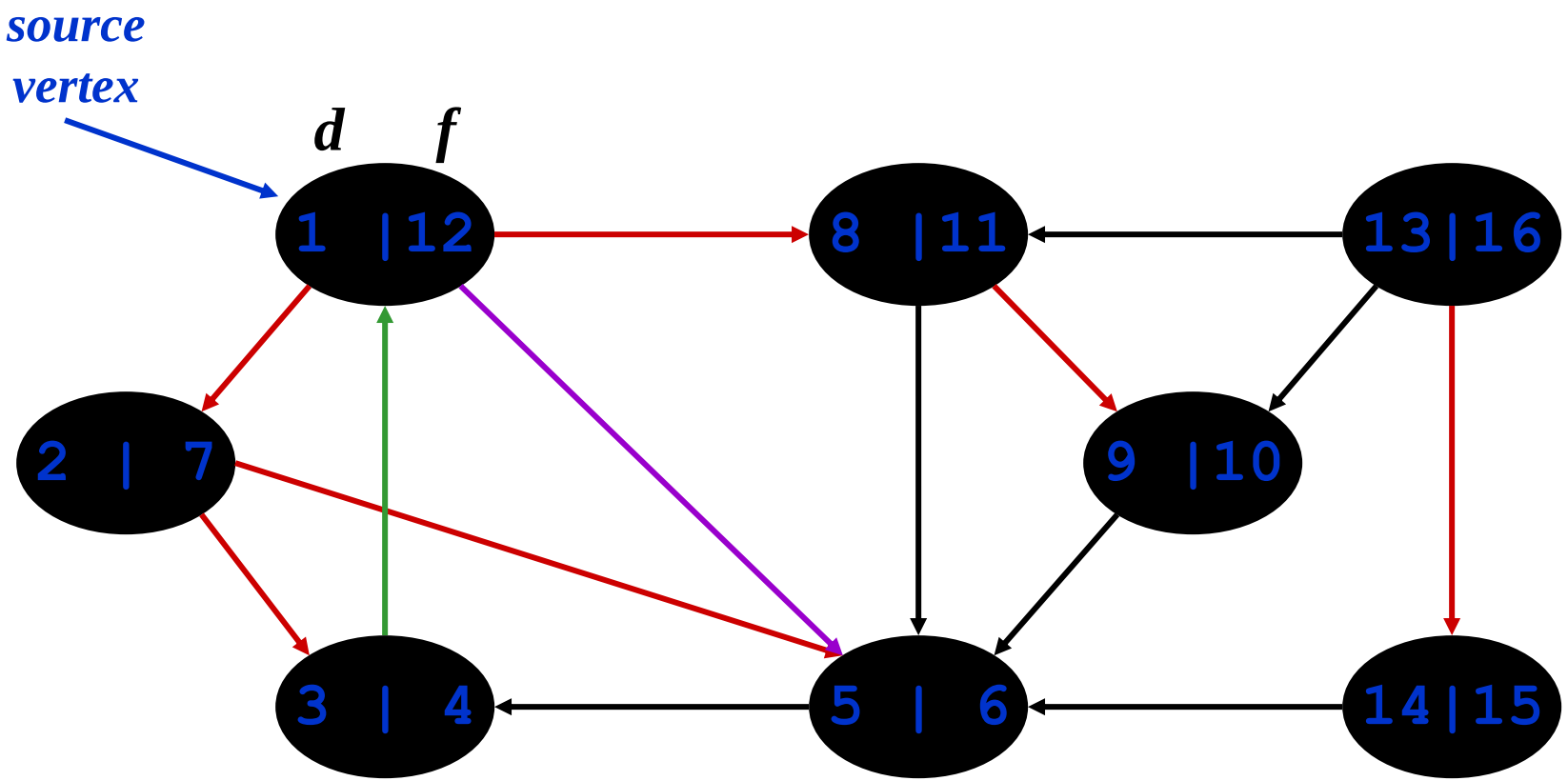


Tree edges *Back edges*

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
    - Not a tree edge, though
    - From grey node to black node

DFS Example

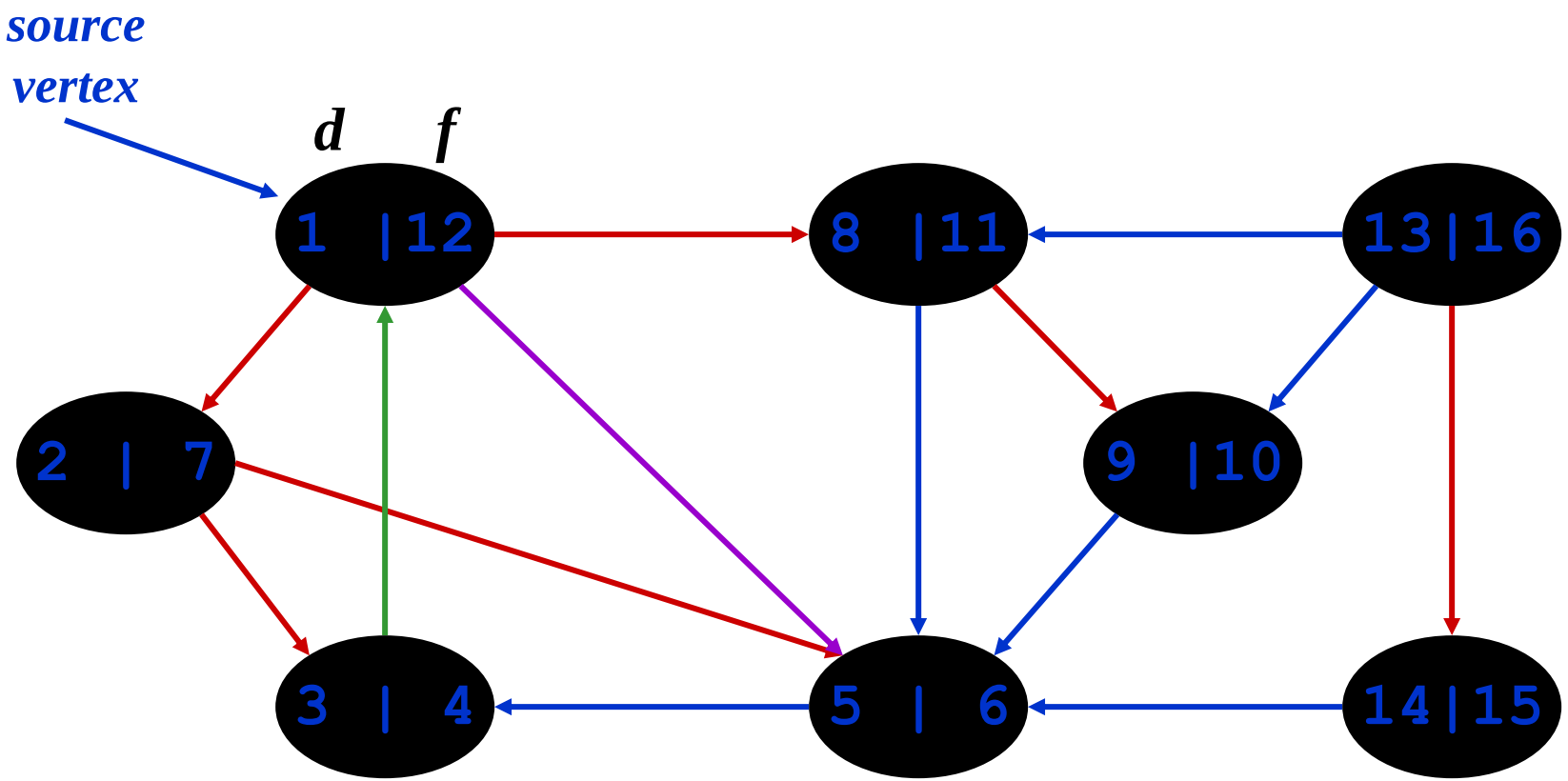


Tree edges *Back edges* *Forward edges*

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
    - From a grey node to a black node

DFS Example



Tree edges *Back edges* *Forward edges* *Cross edges*

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

More about the edges

- Let (u,v) is an edge.
  - If (color[v] = WHITE) then (u,v) is a tree edge
  - If (color[v] = GRAY) then (u,v) is a back edge
  - If (color[v] = BLACK) then (u,v) is a forward/cross edge
    - Forward Edge: d[u]<d[v]
    - Cross Edge: d[u]>d[v]

Depth-First Search - Timestamps