

# Graph Theory: Depth First Search

William Fiset

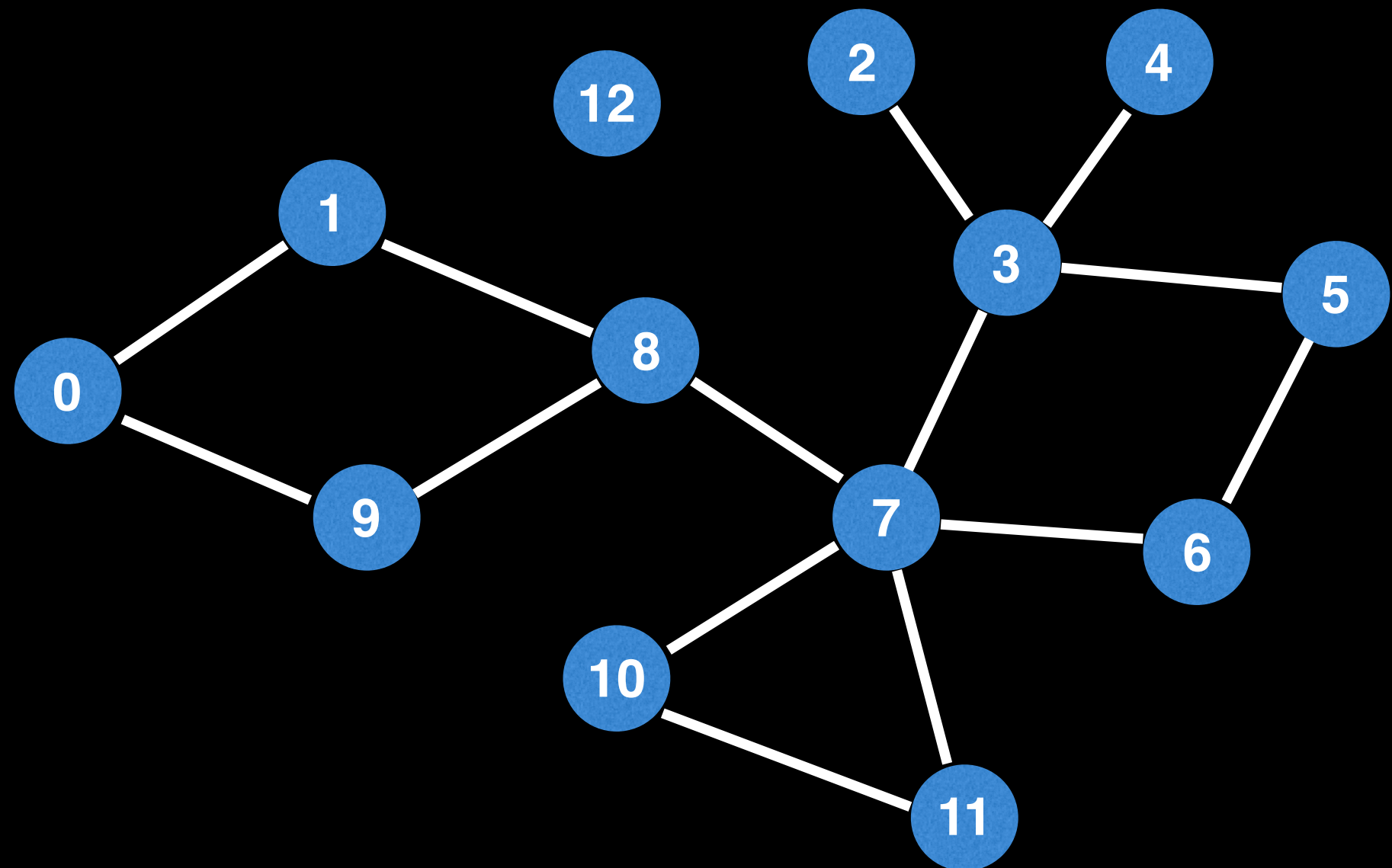
# DFS overview

The **Depth First Search (DFS)** is the most fundamental search algorithm used to explore nodes and edges of a graph. It runs with a time complexity of  $O(V+E)$  and is often used as a building block in other algorithms.

By itself the DFS isn't all that useful, but **when augmented** to perform other tasks such as count connected components, determine connectivity, or find bridges/articulation points then **DFS really shines**.

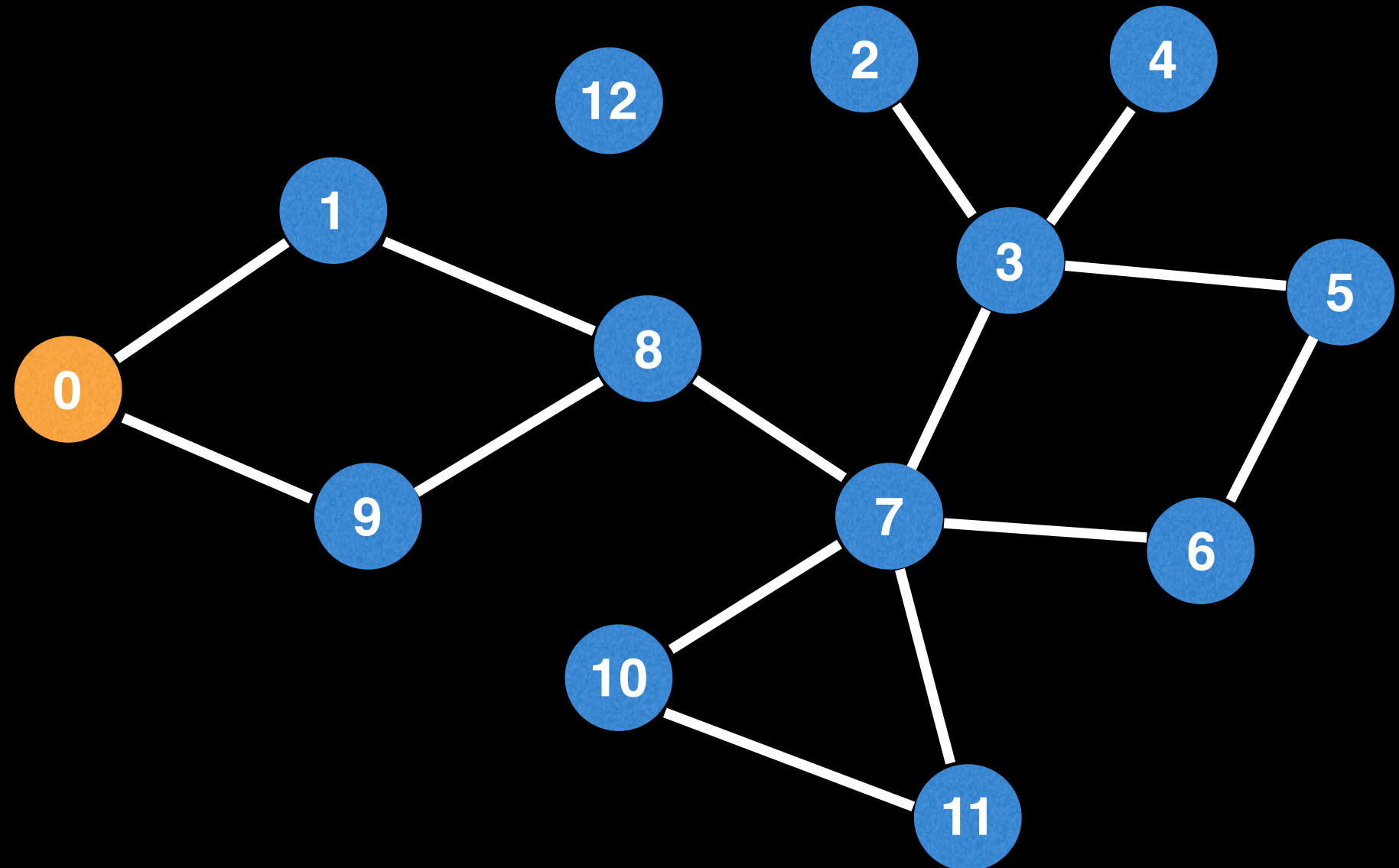
# Basic DFS

As the name suggests, a DFS plunges depth first into a graph without regard for which edge it takes next until it cannot go any further at which point it backtracks and continues.



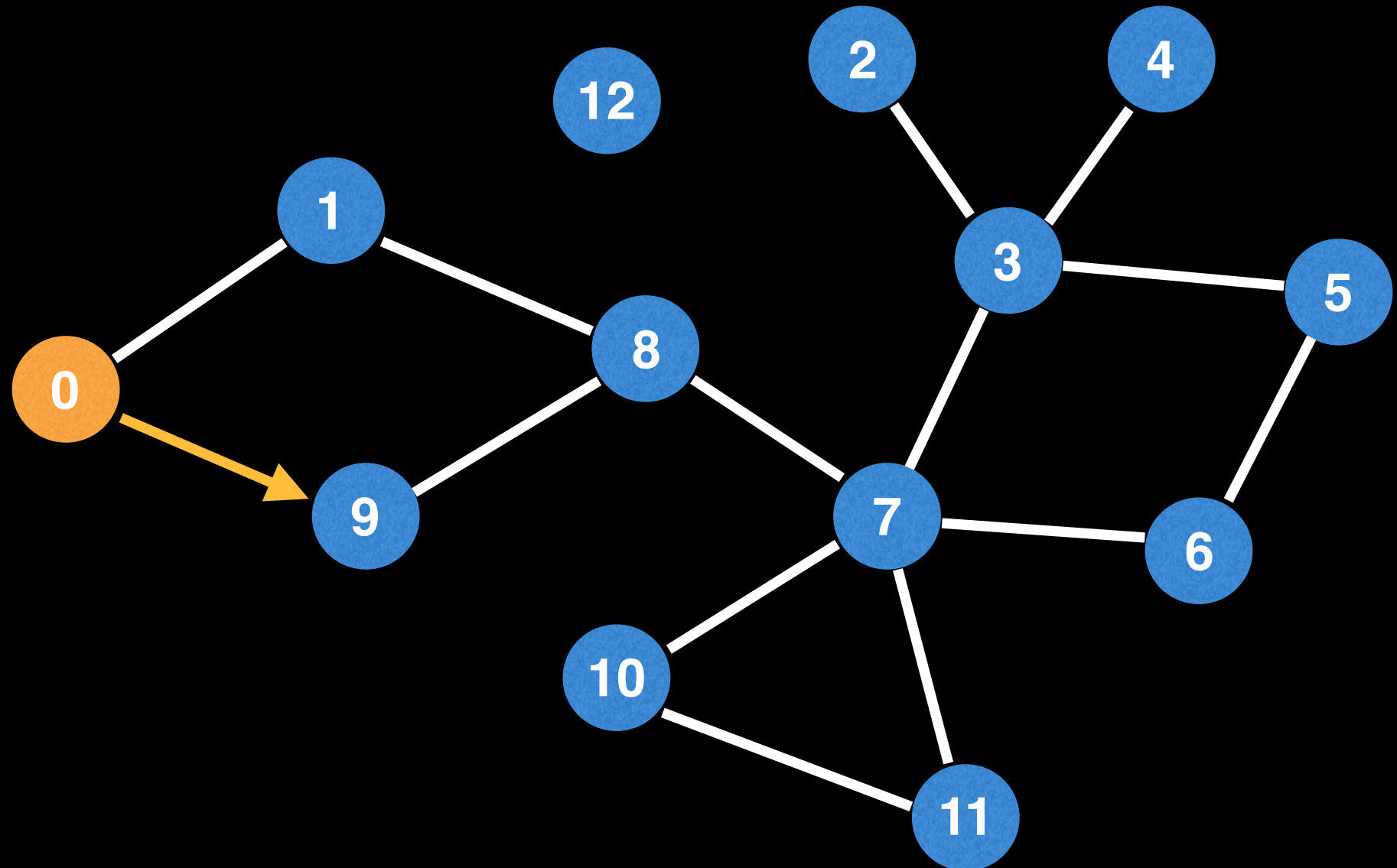
# Basic DFS

Start DFS at node 0



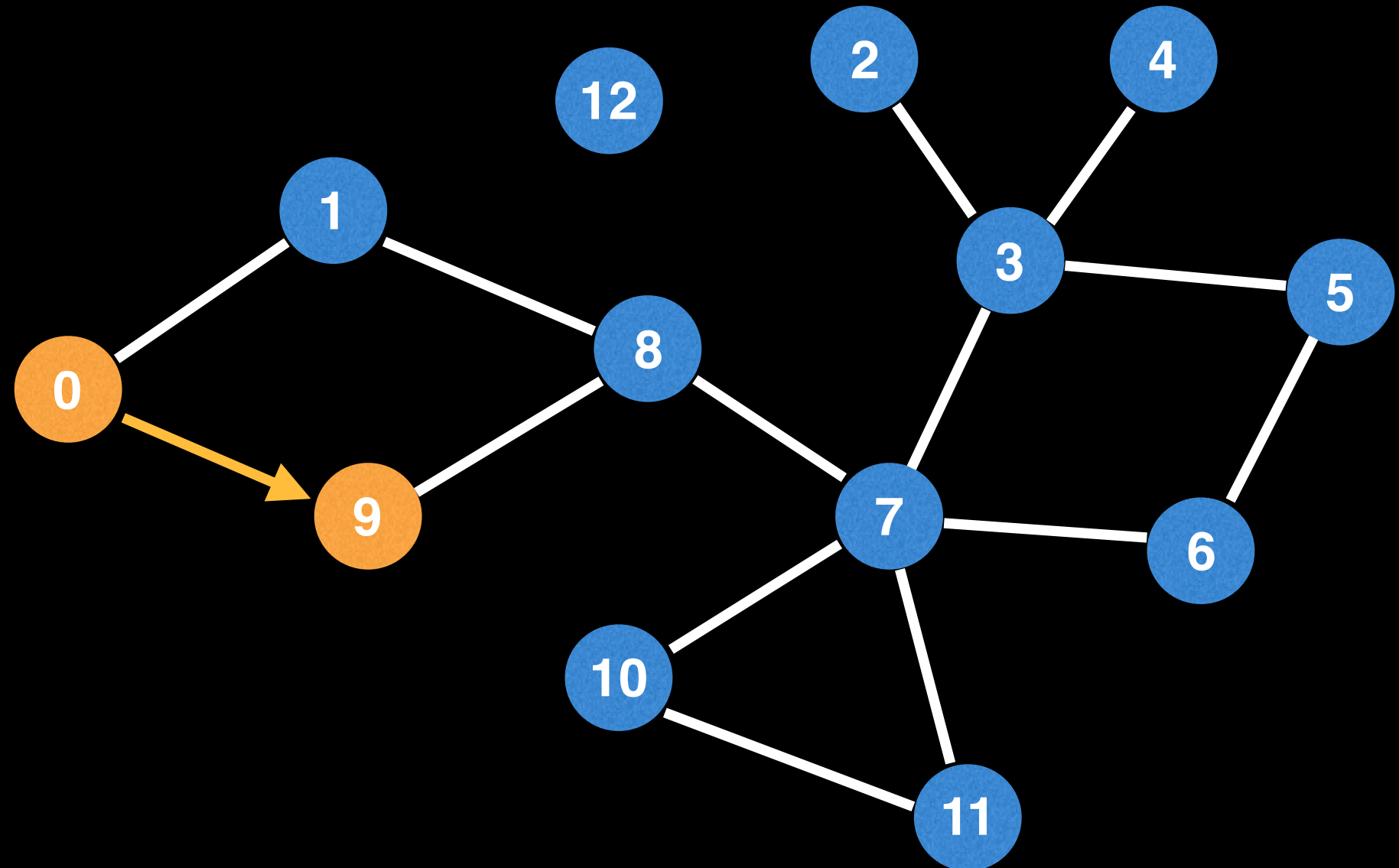
# Basic DFS

Pick an edge outwards from node 0



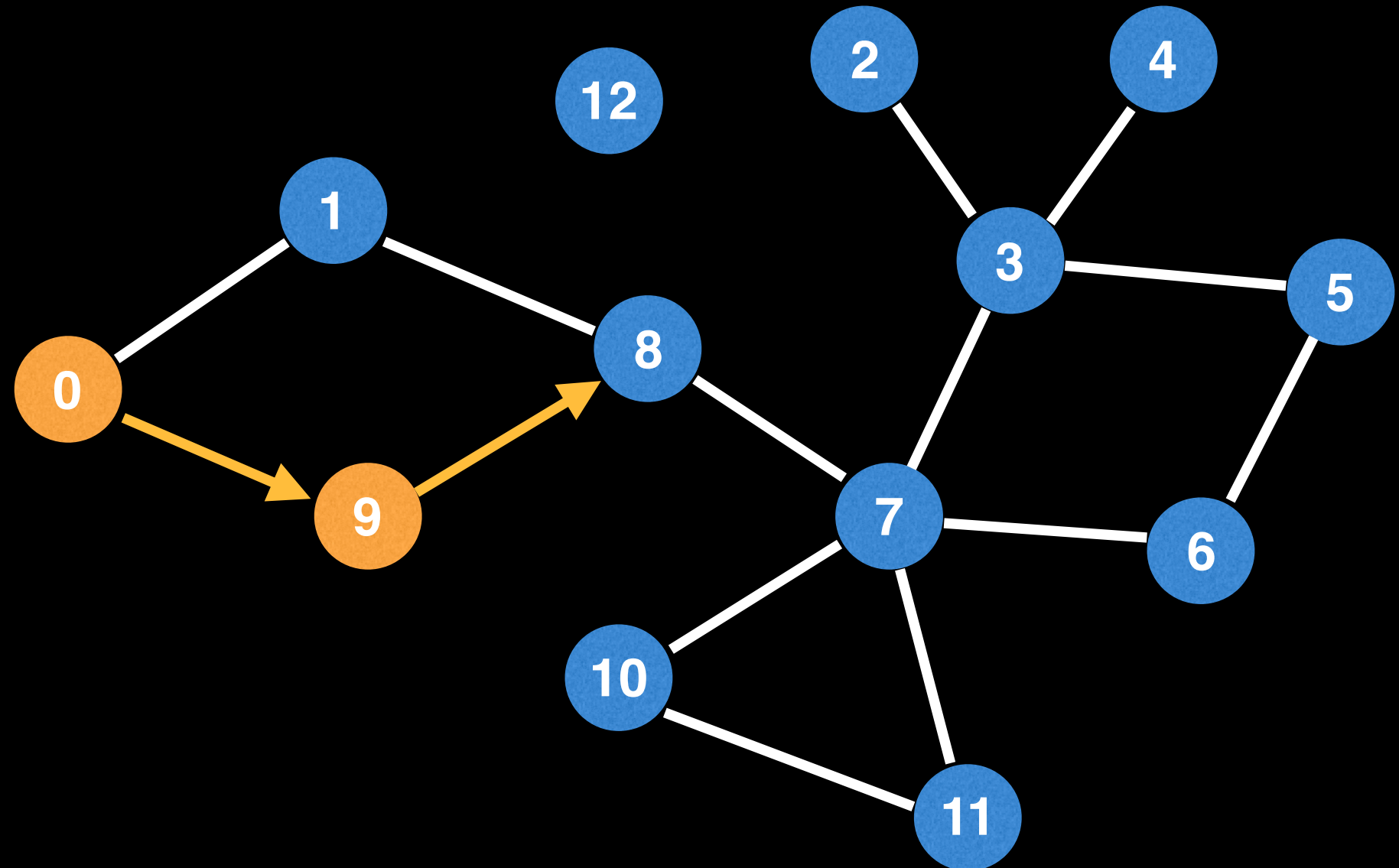
# Basic DFS

Once at 9 pick an edge outwards from node 9



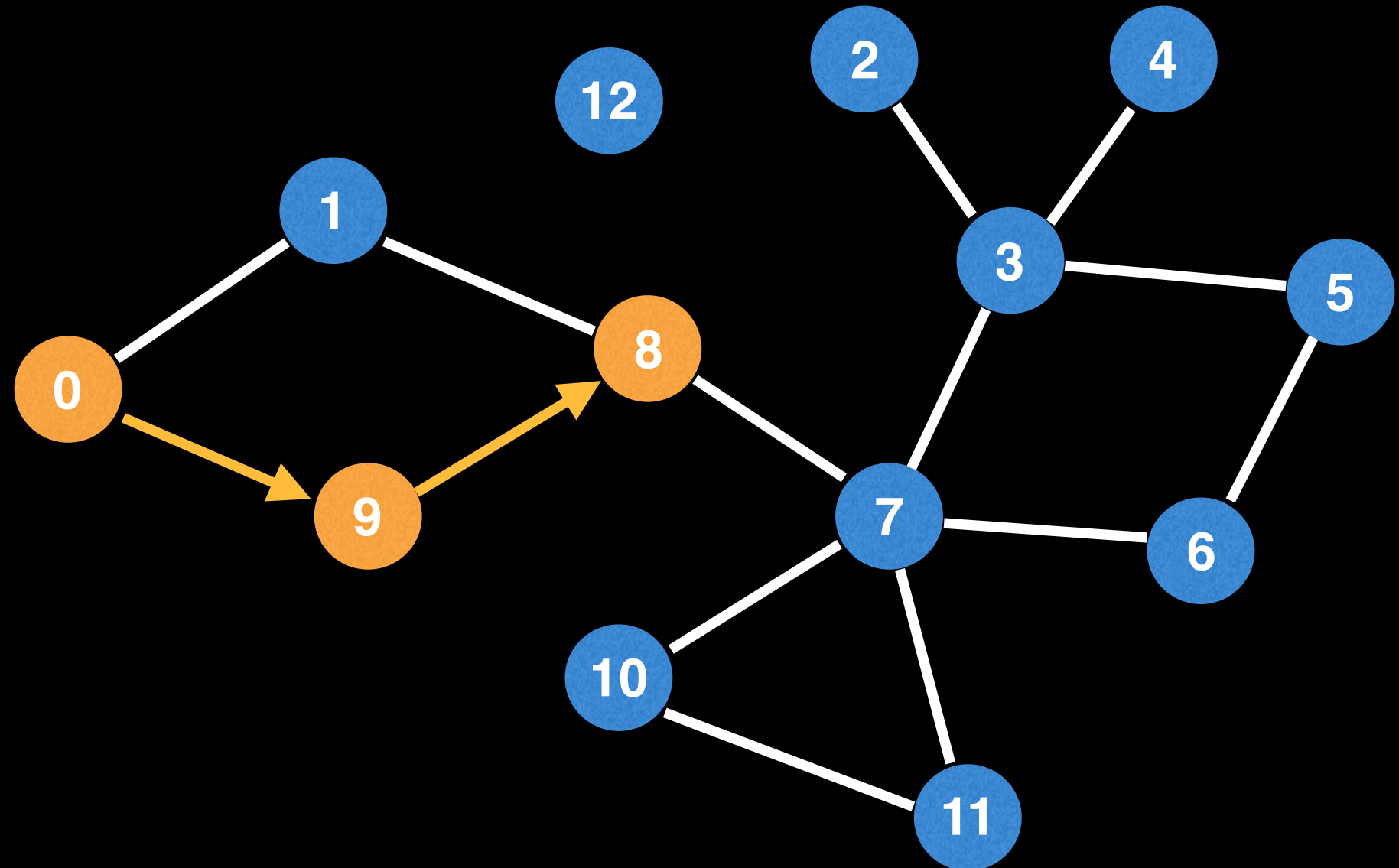
# Basic DFS

Go to node 8



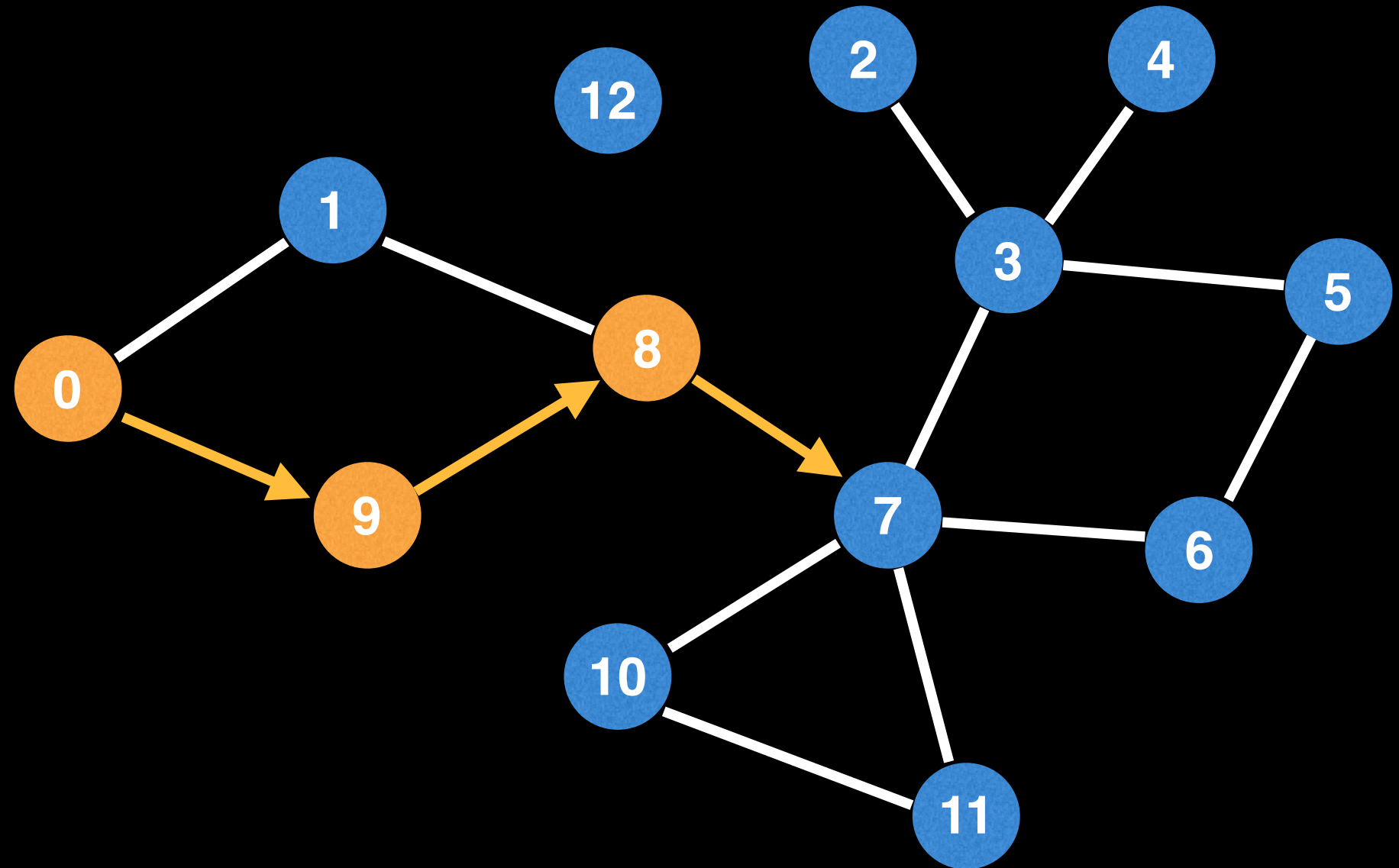
# Basic DFS

Pick an edge outwards from 8...

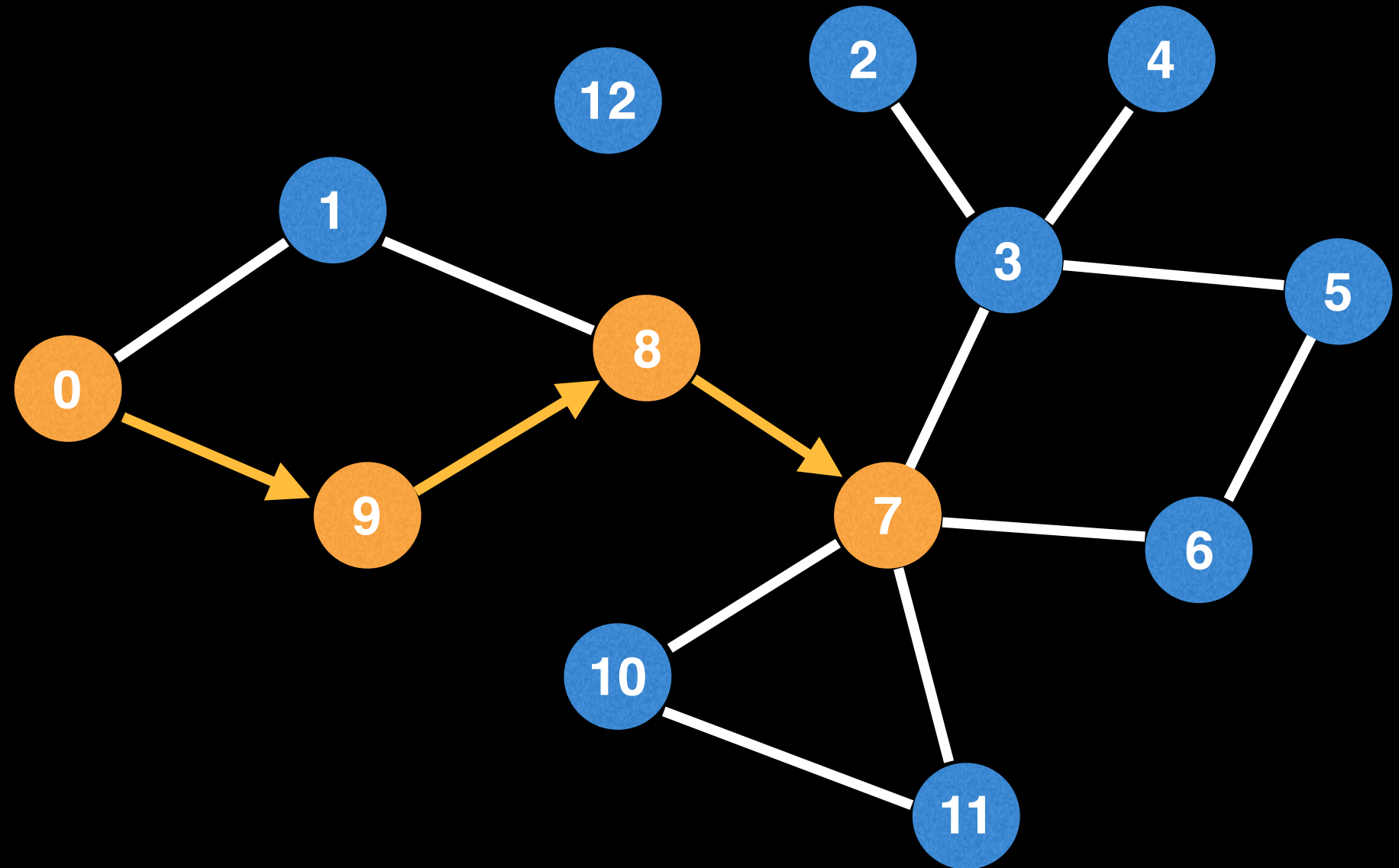




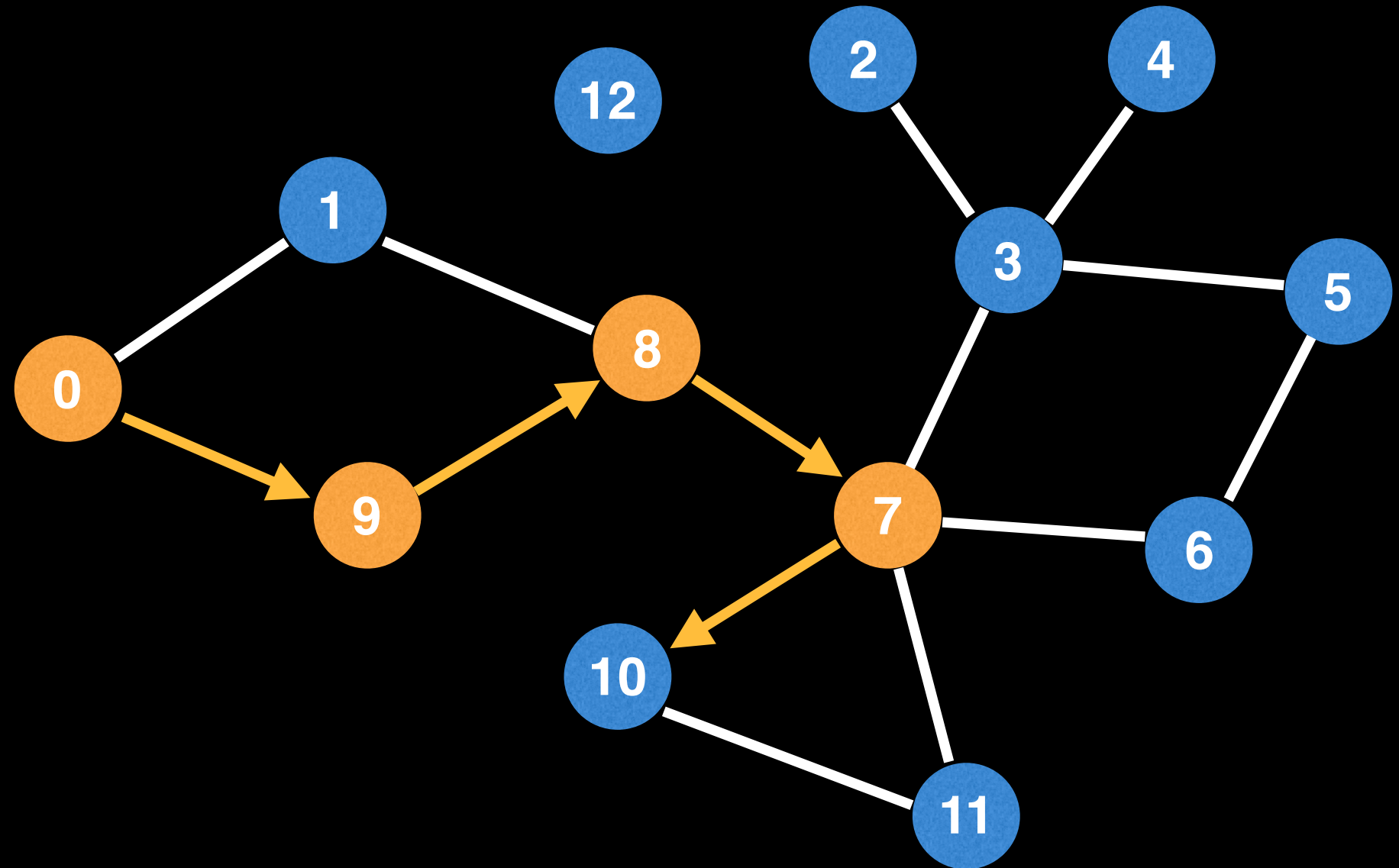
# Basic DFS



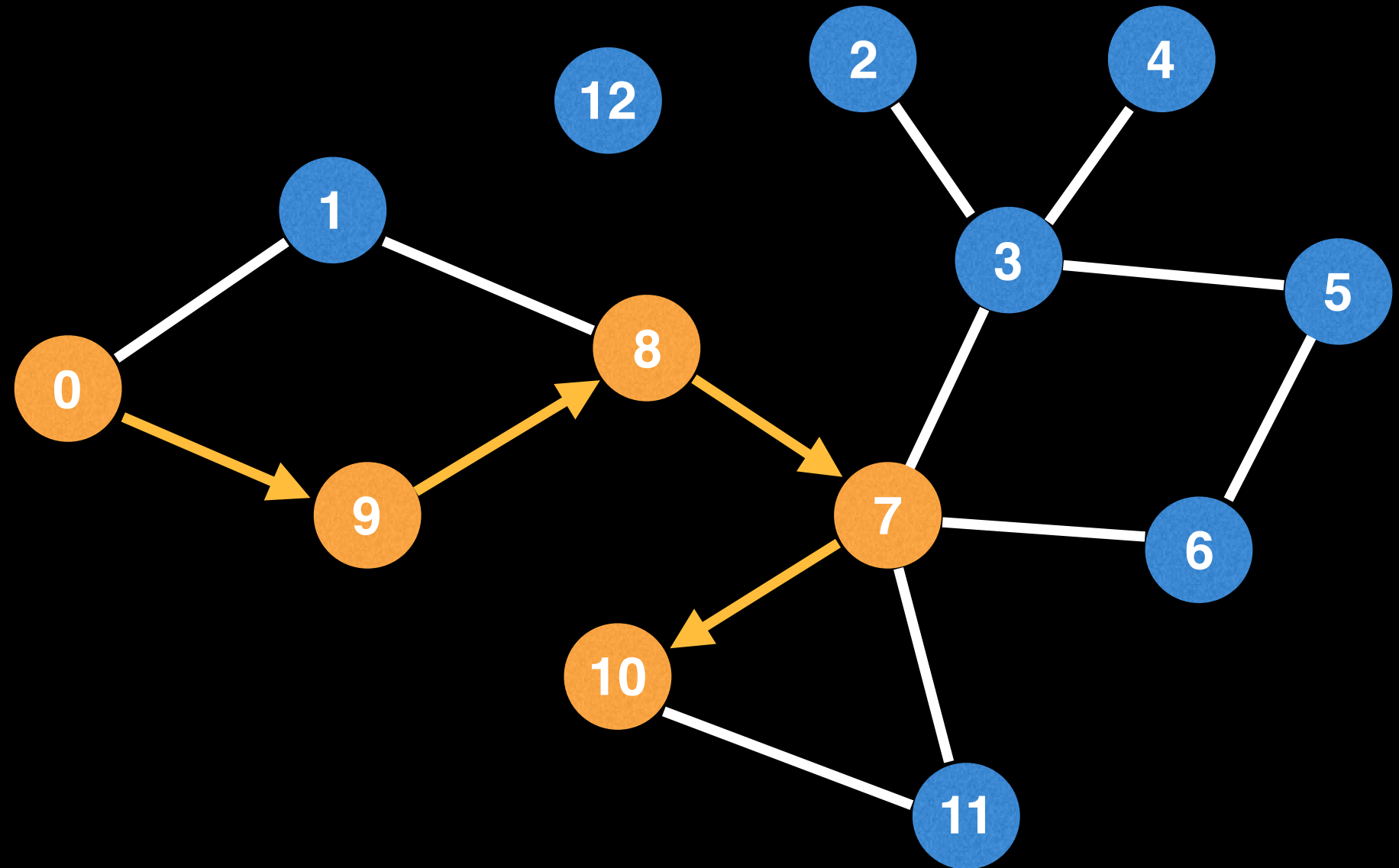
# Basic DFS



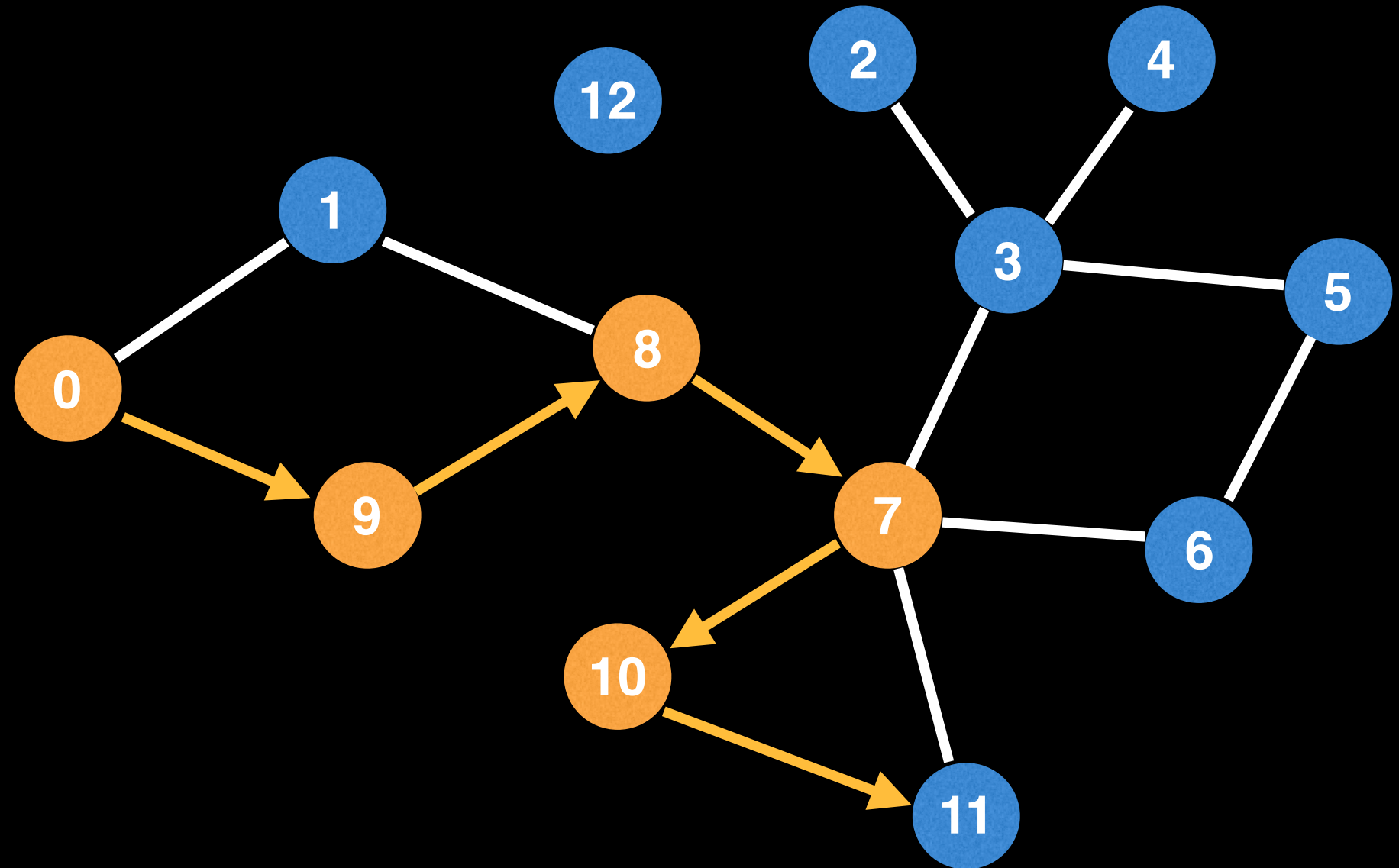
# Basic DFS



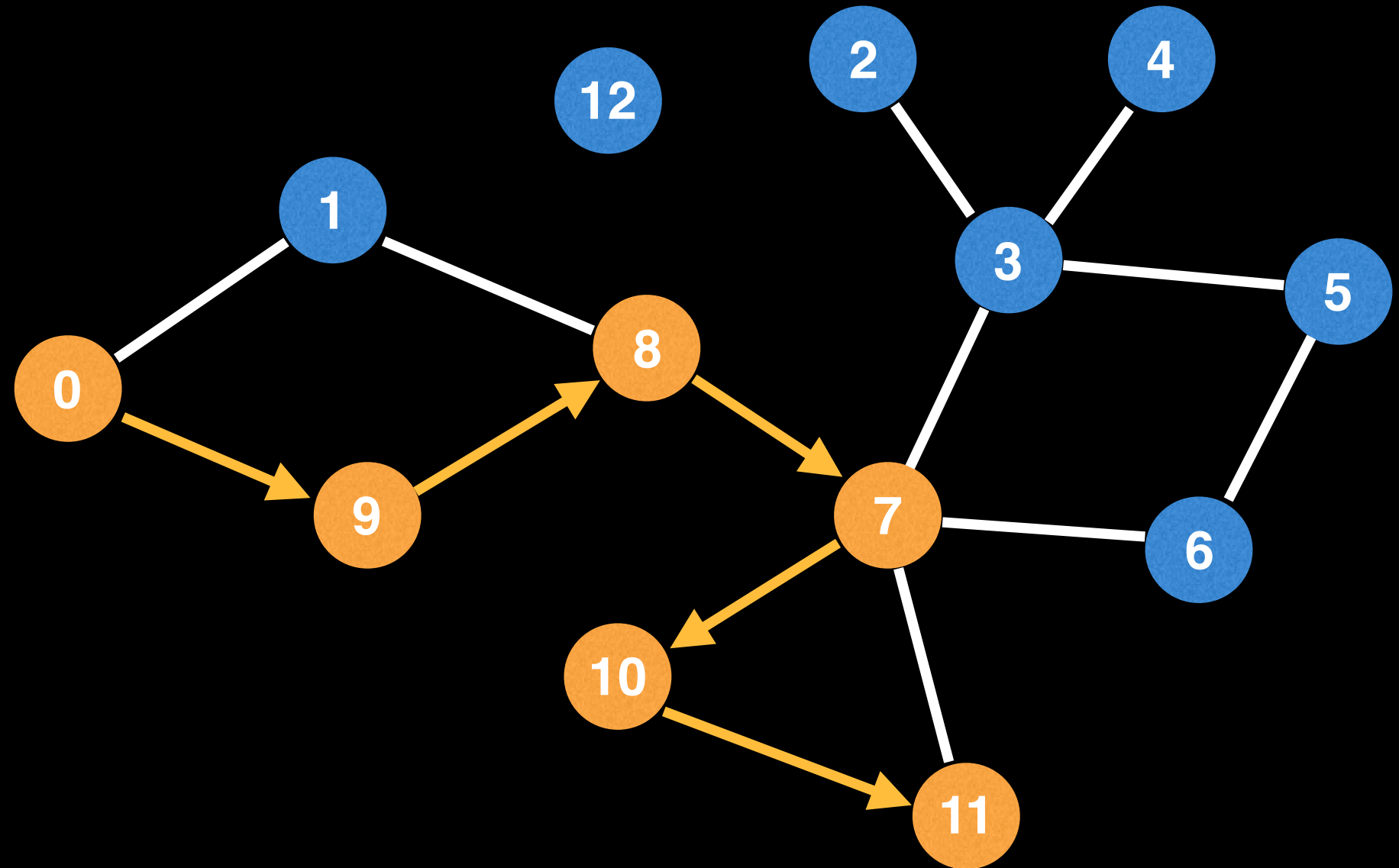
# Basic DFS



# Basic DFS

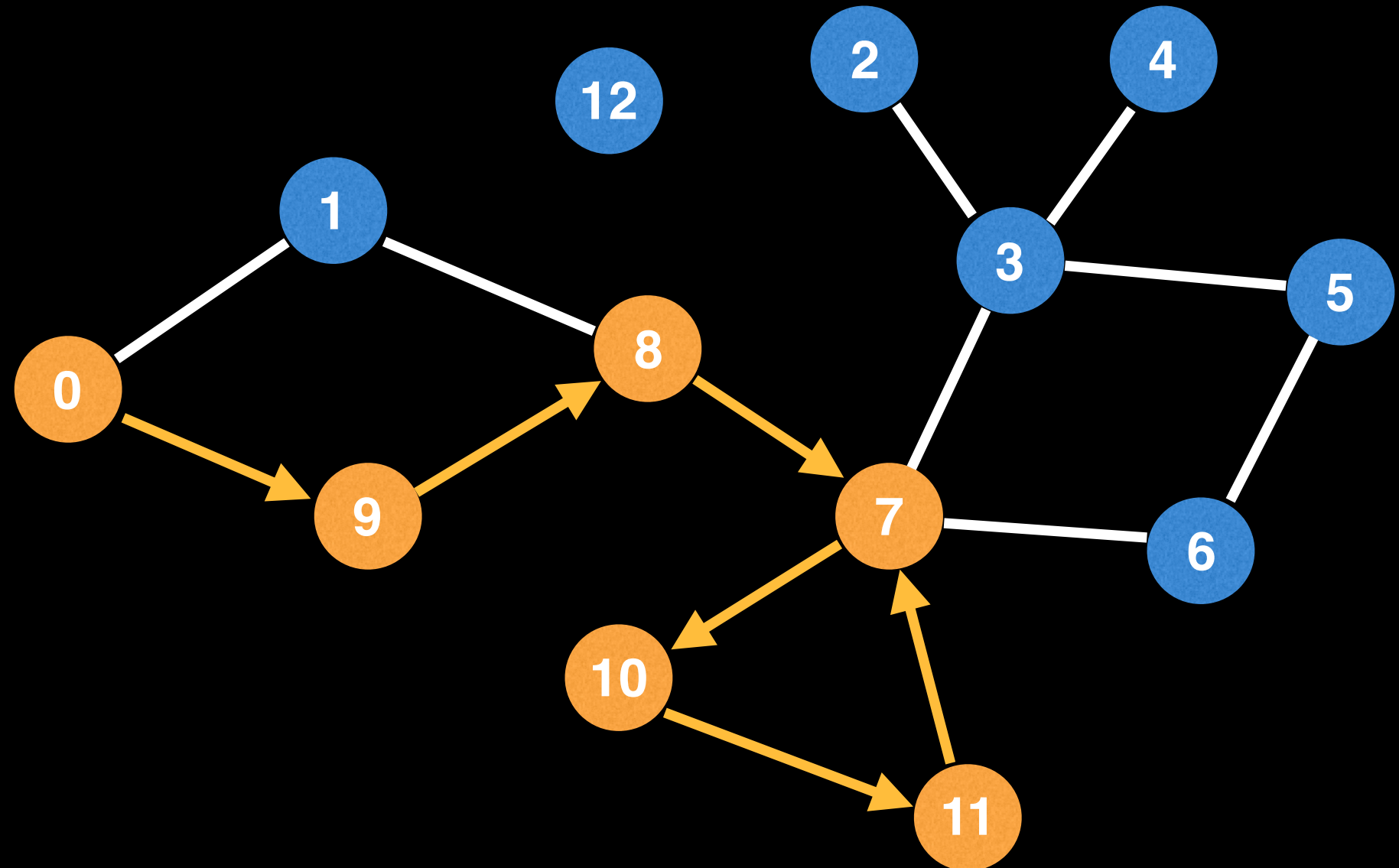


# Basic DFS

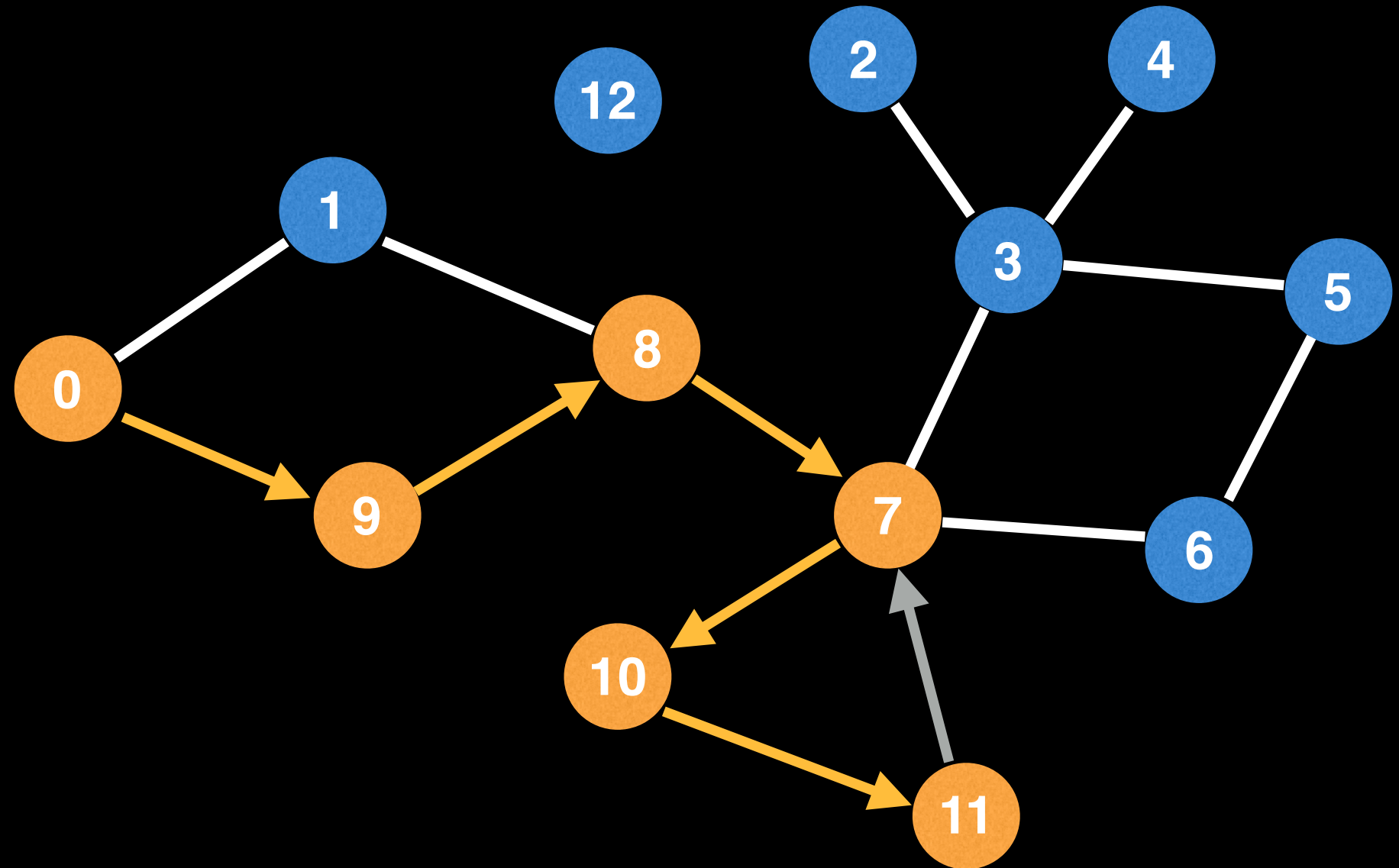


# Basic DFS

Make sure you don't re-visit visited nodes!

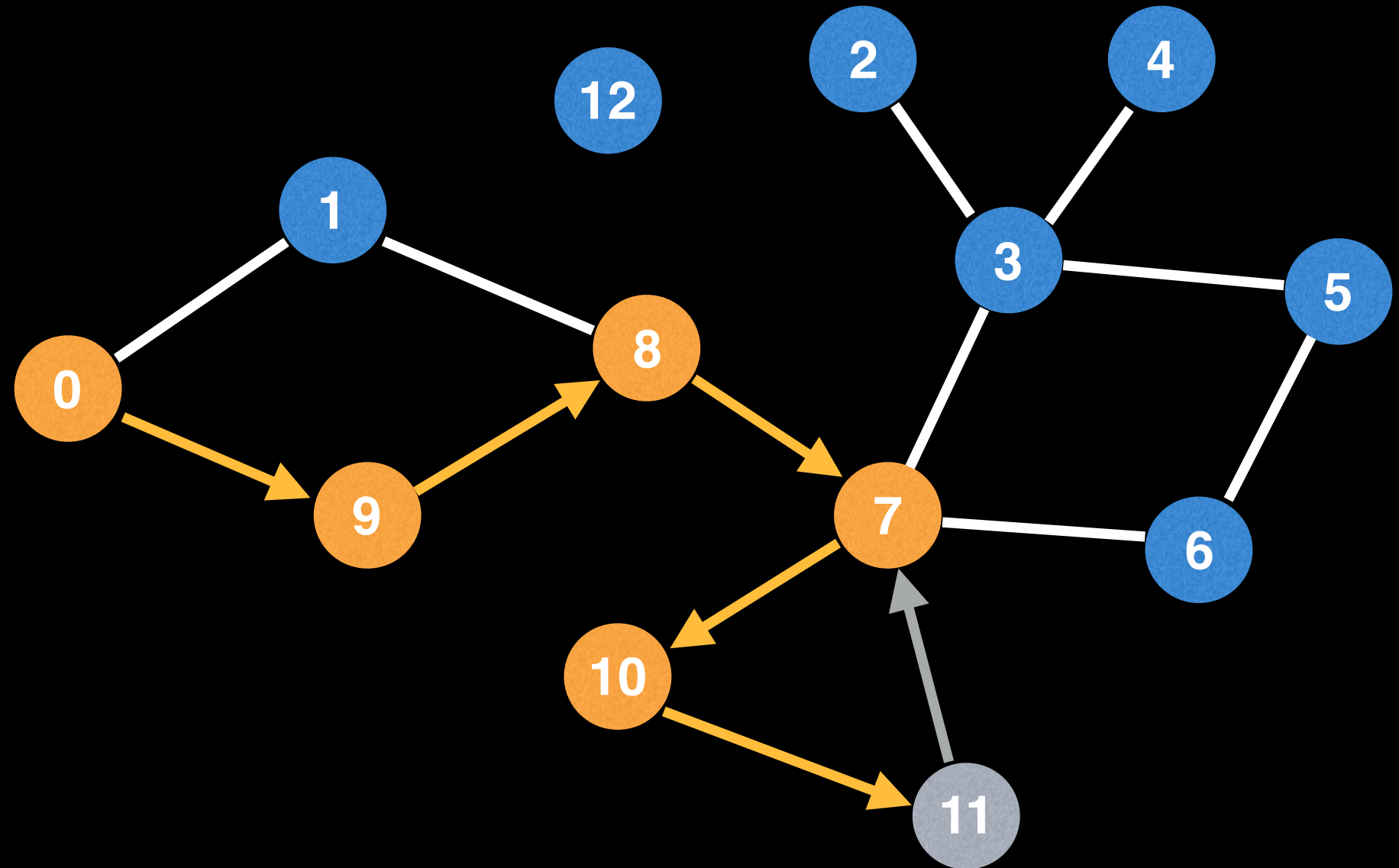


# Basic DFS

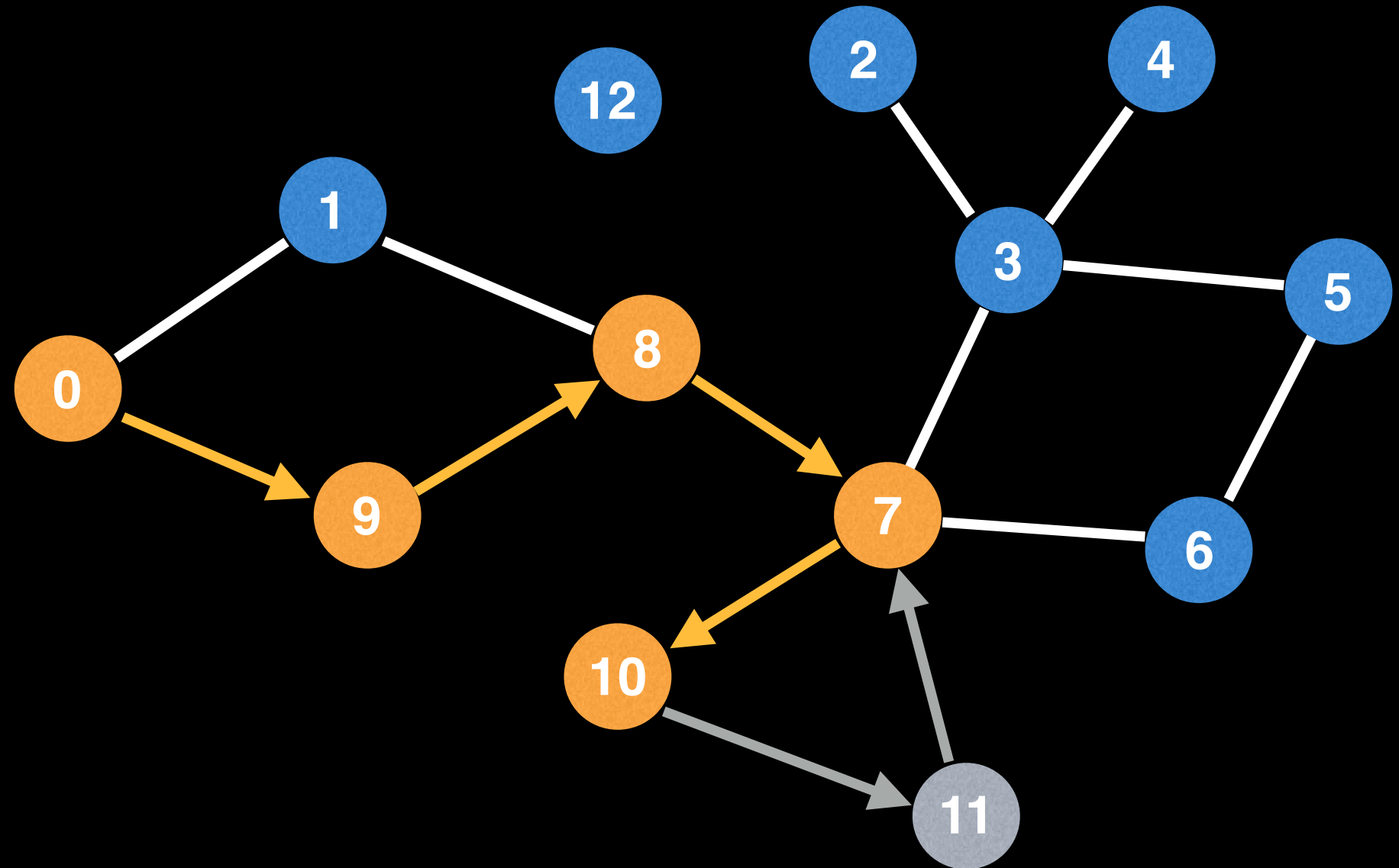




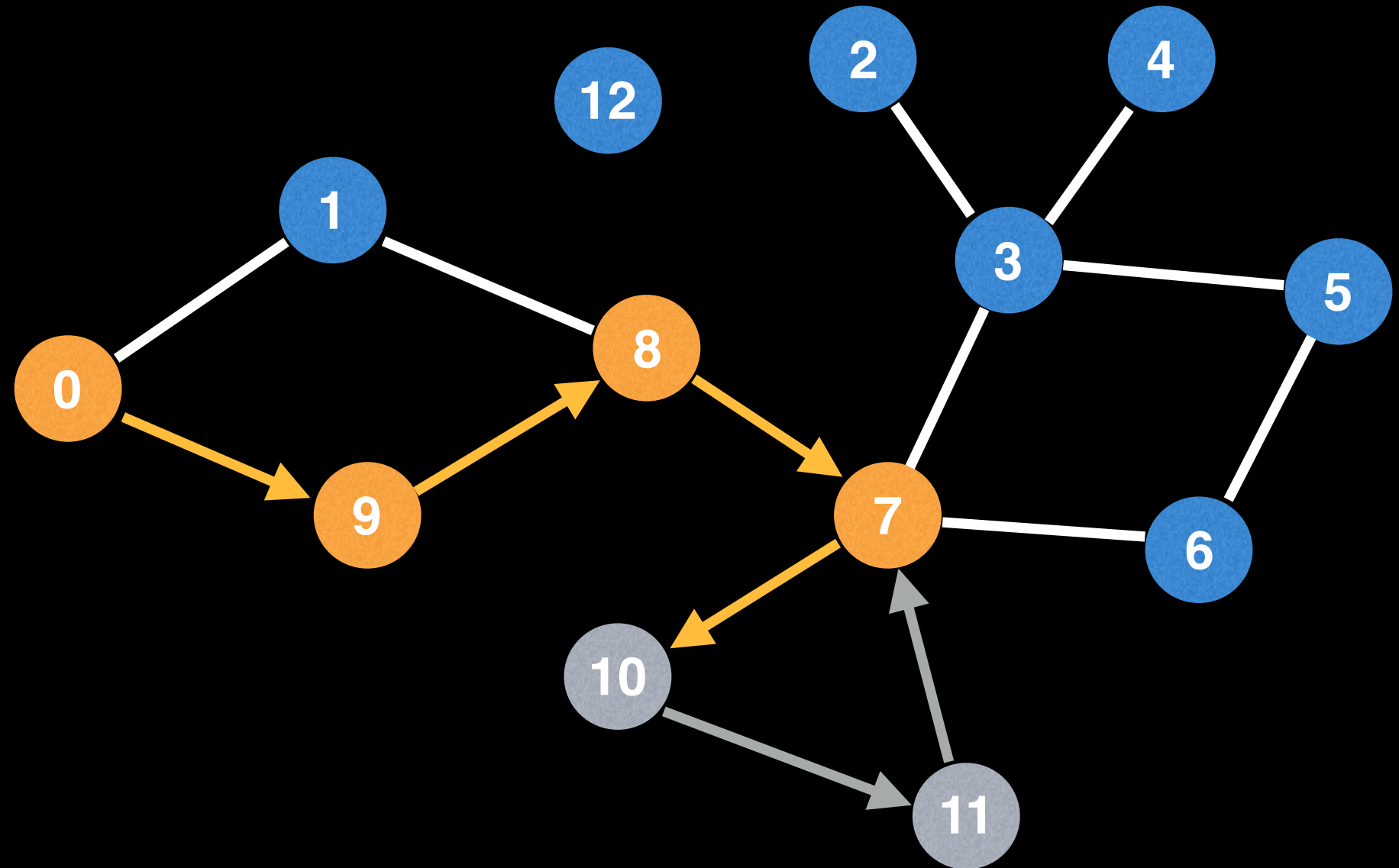
# Basic DFS



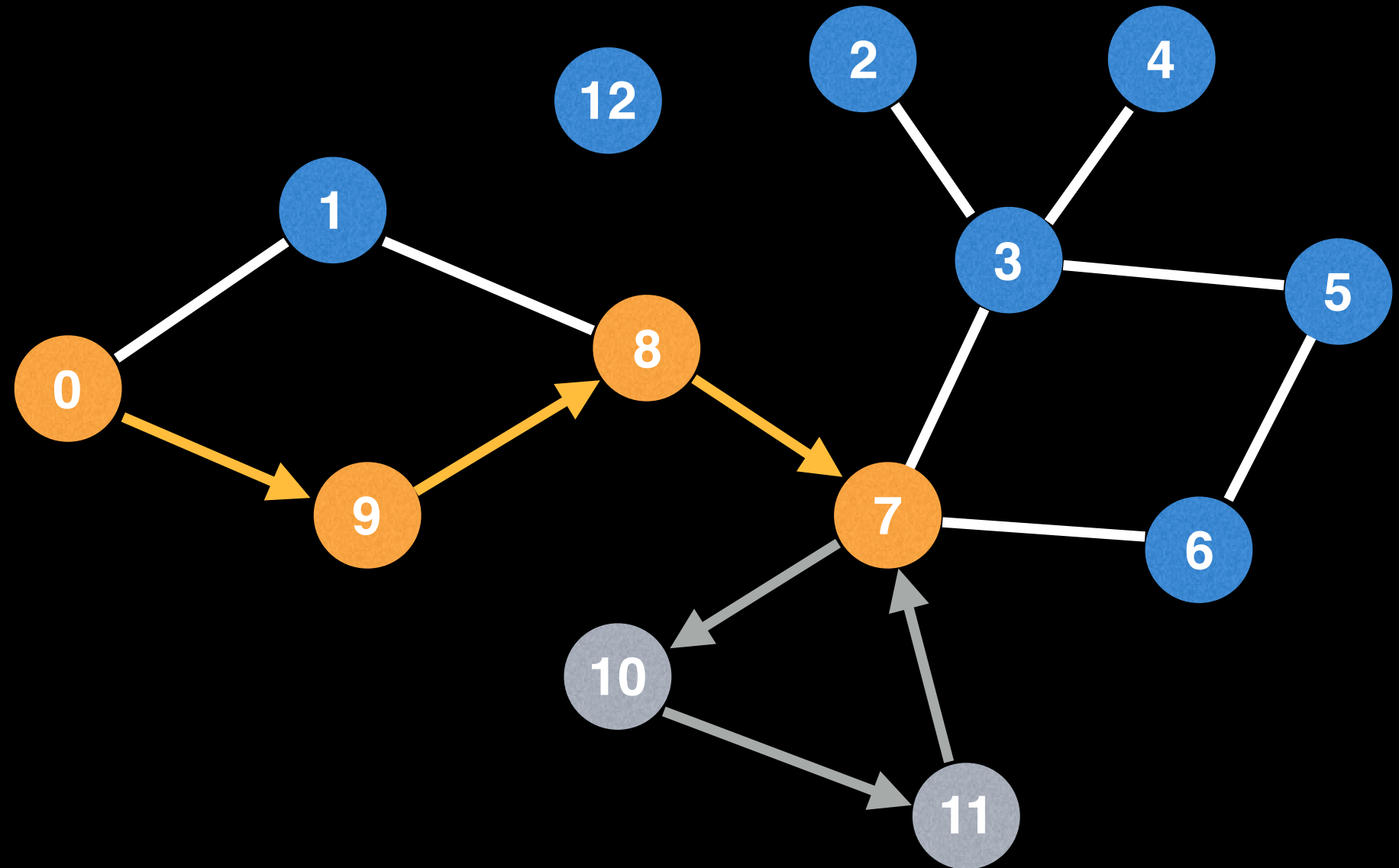
# Basic DFS



# Basic DFS

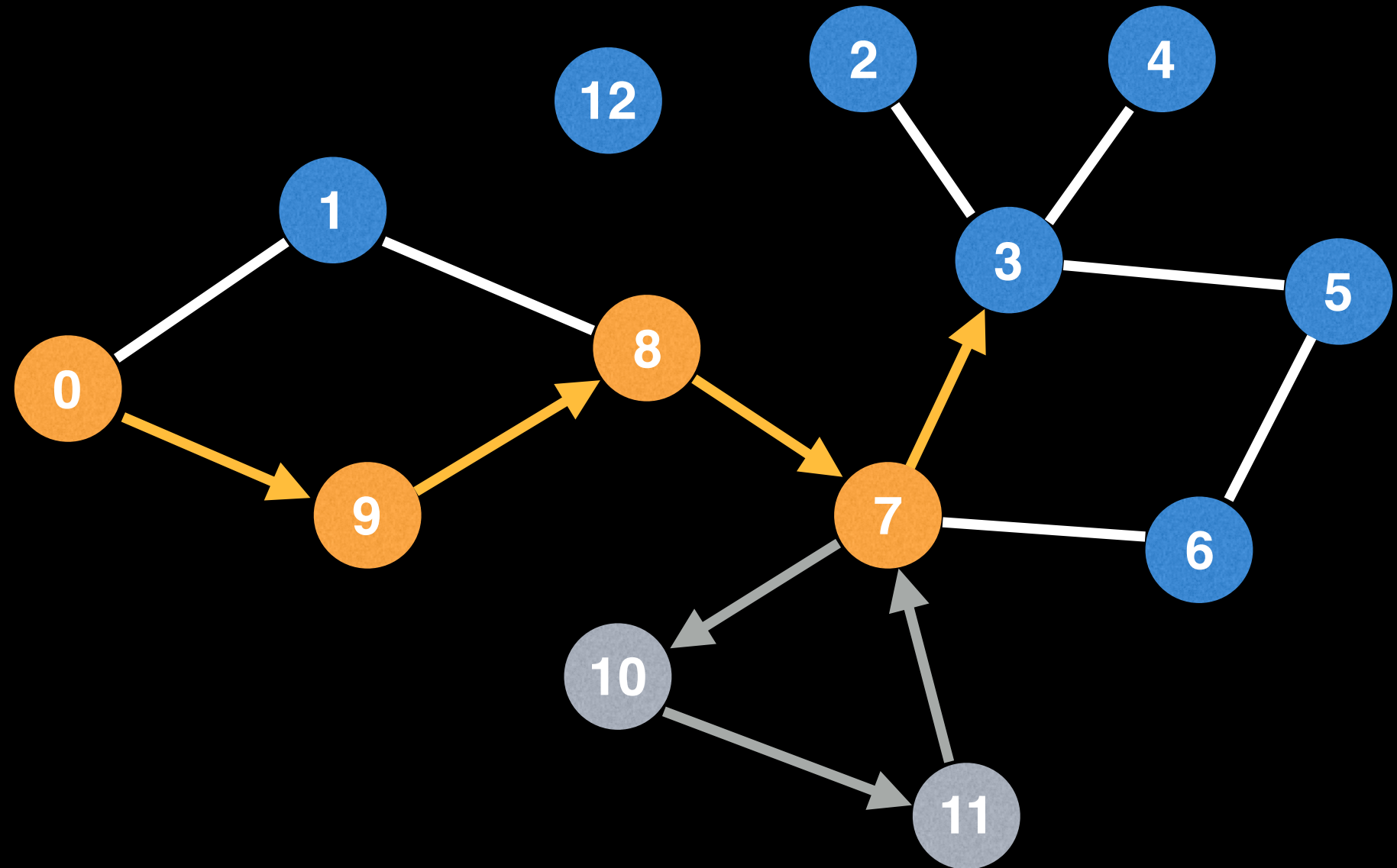


# Basic DFS

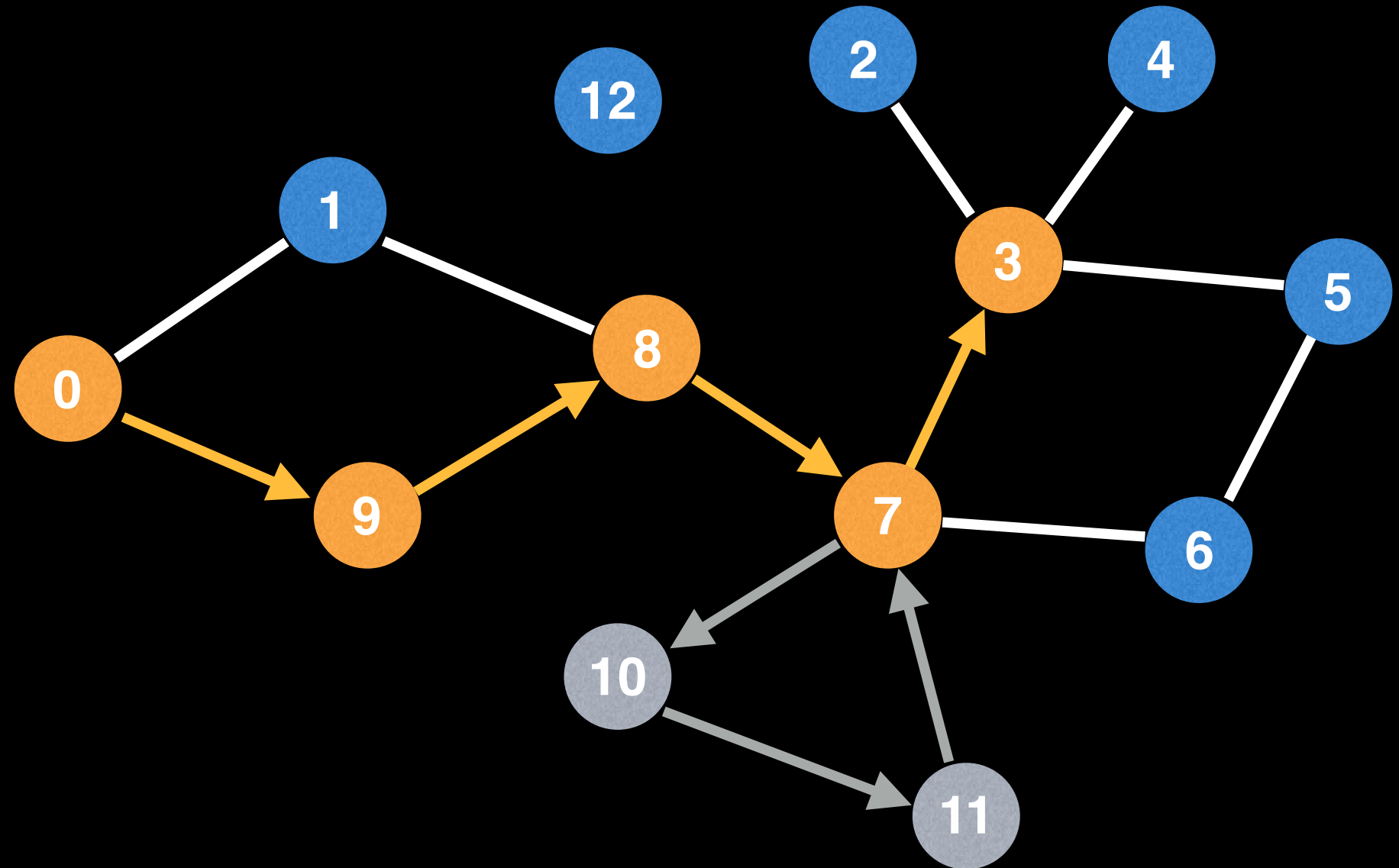


# Basic DFS

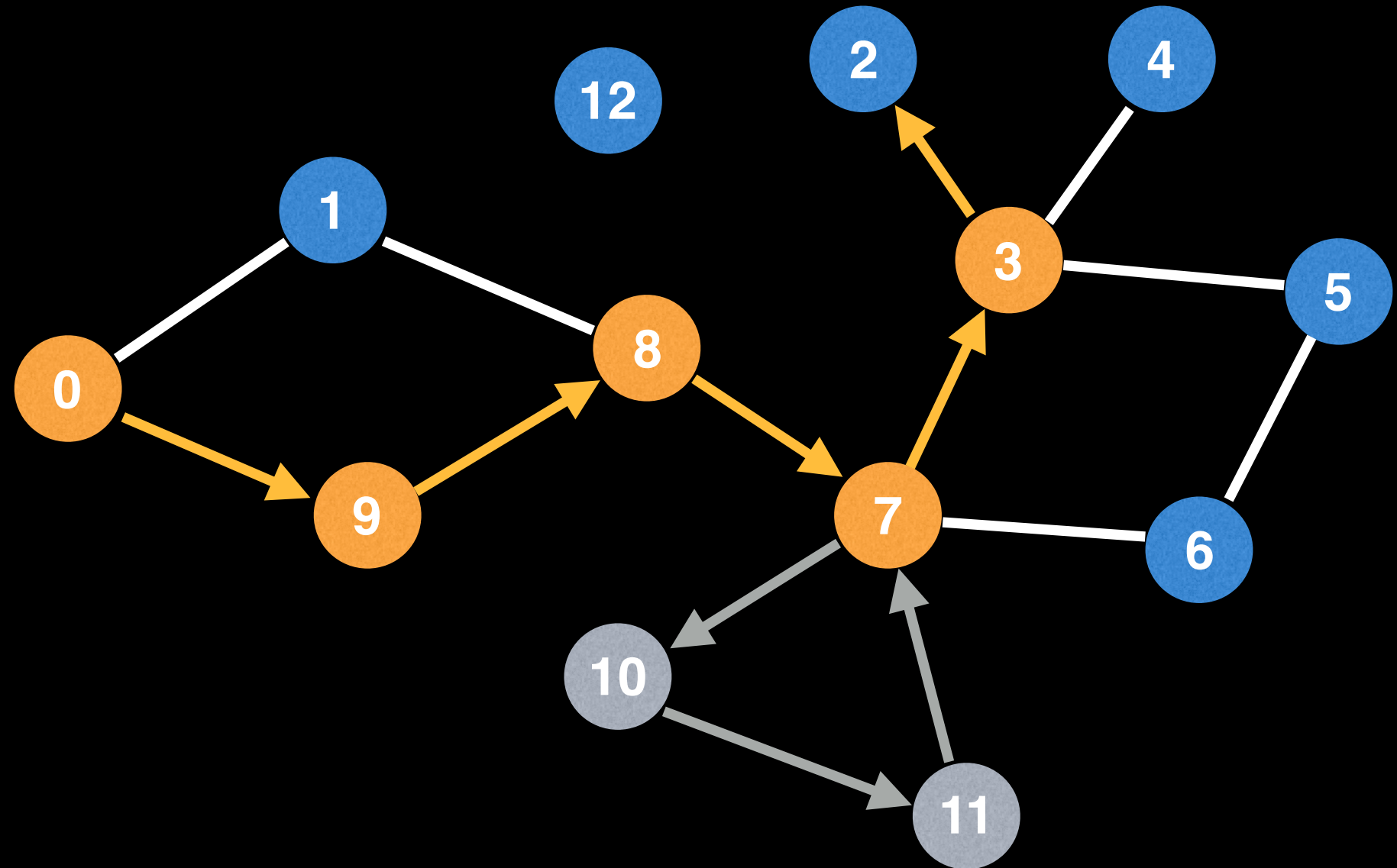
We haven't finished visiting all the neighbours of 7 so continue DFS in another direction



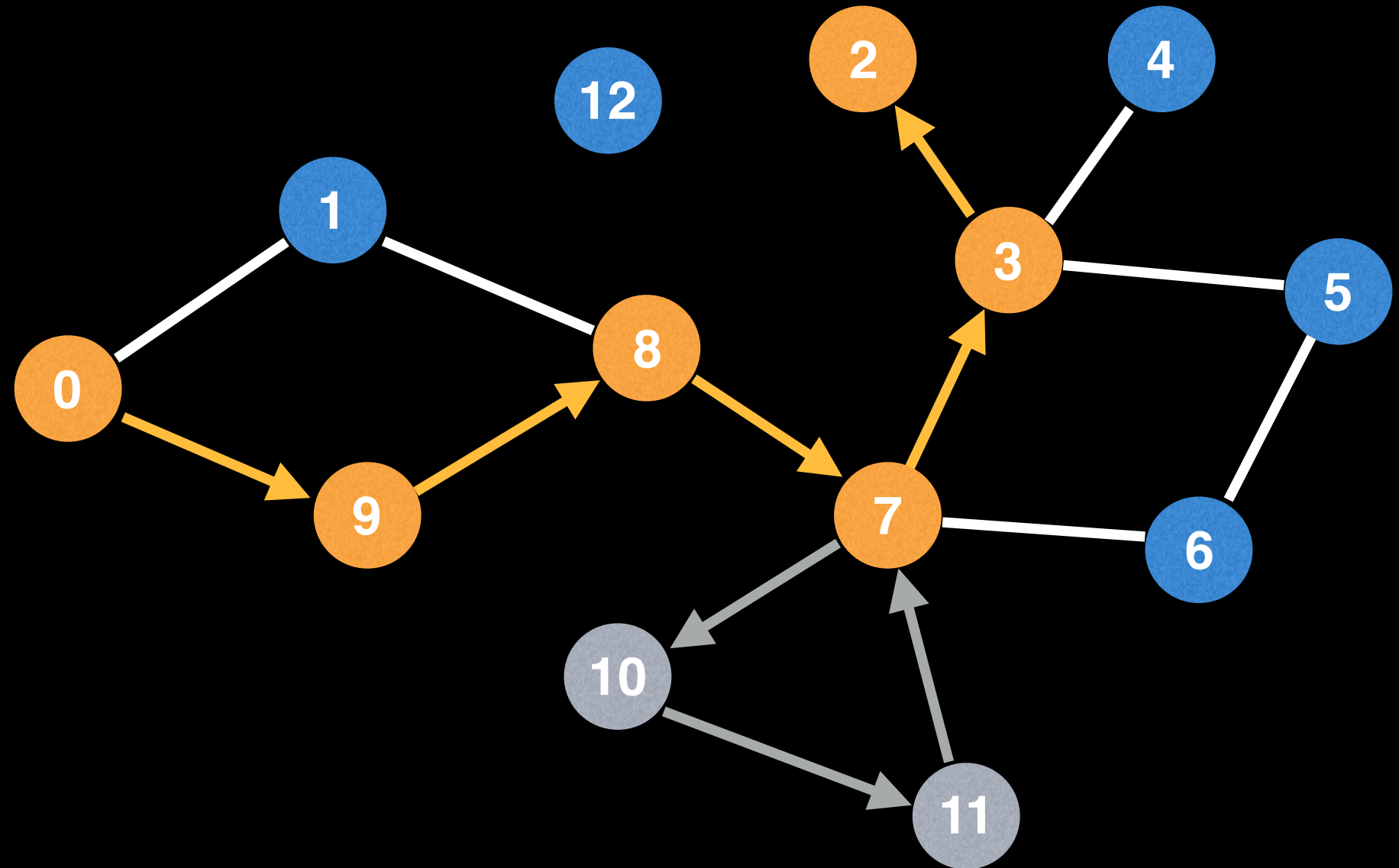
# Basic DFS



# Basic DFS



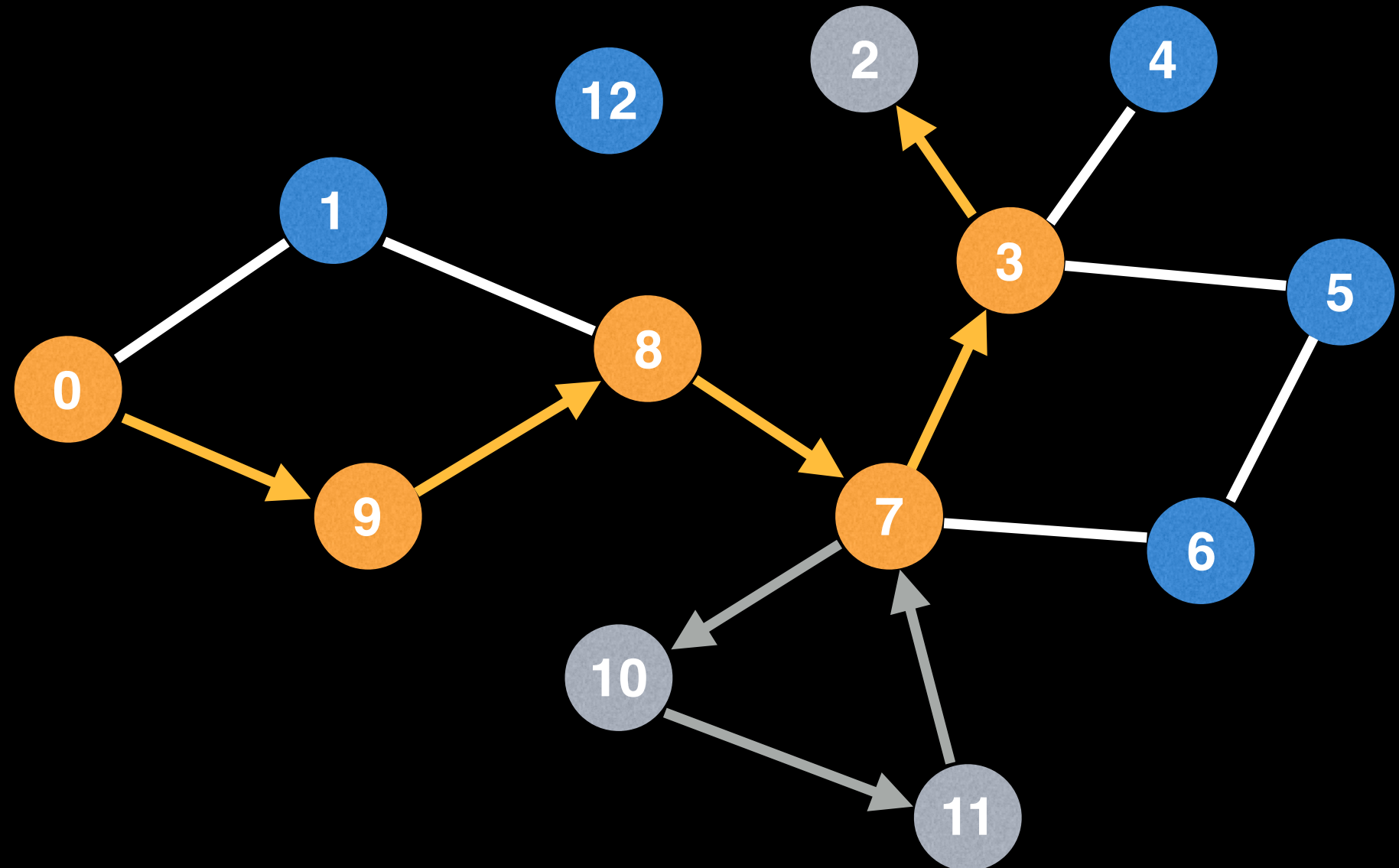
# Basic DFS



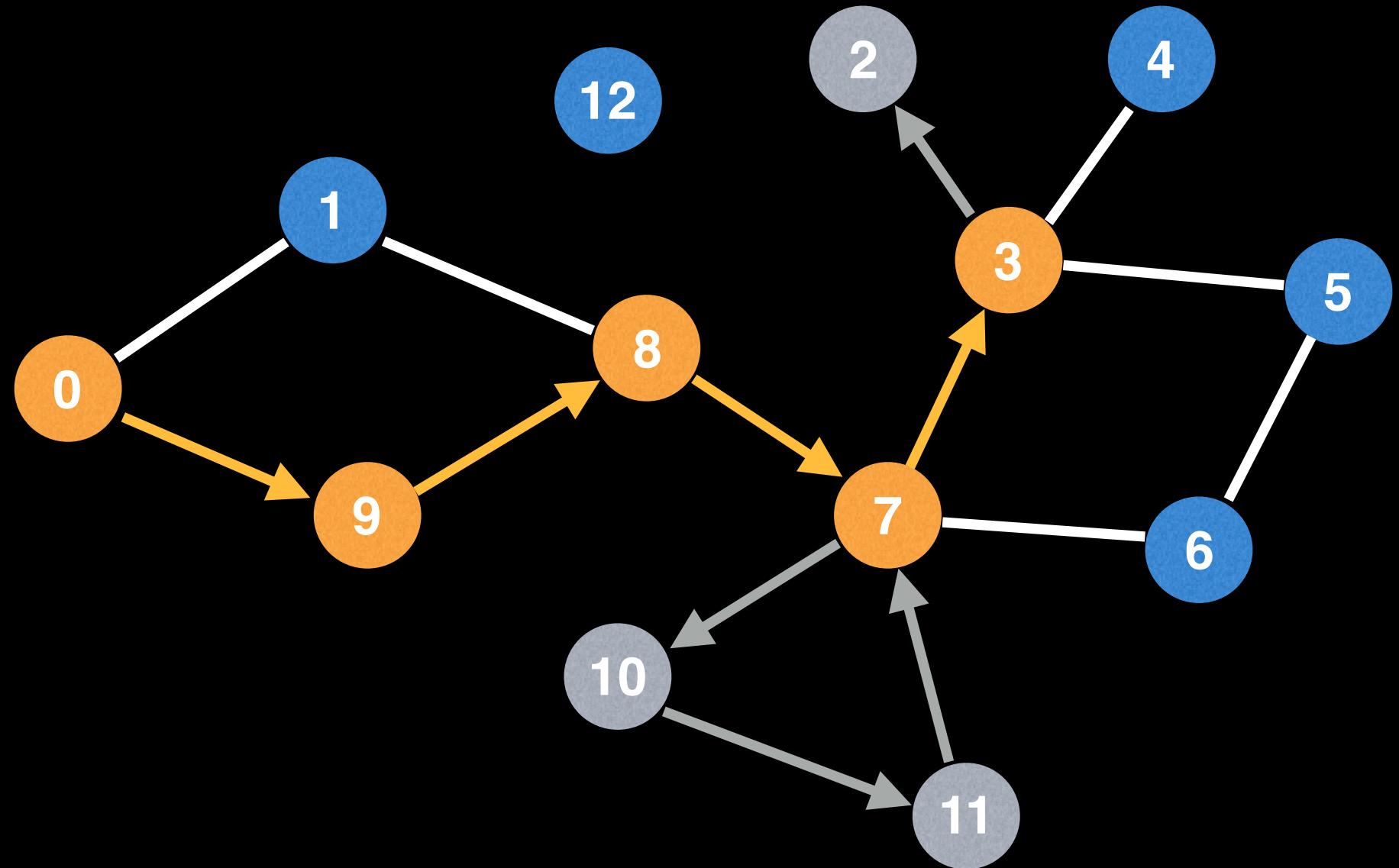


# Basic DFS

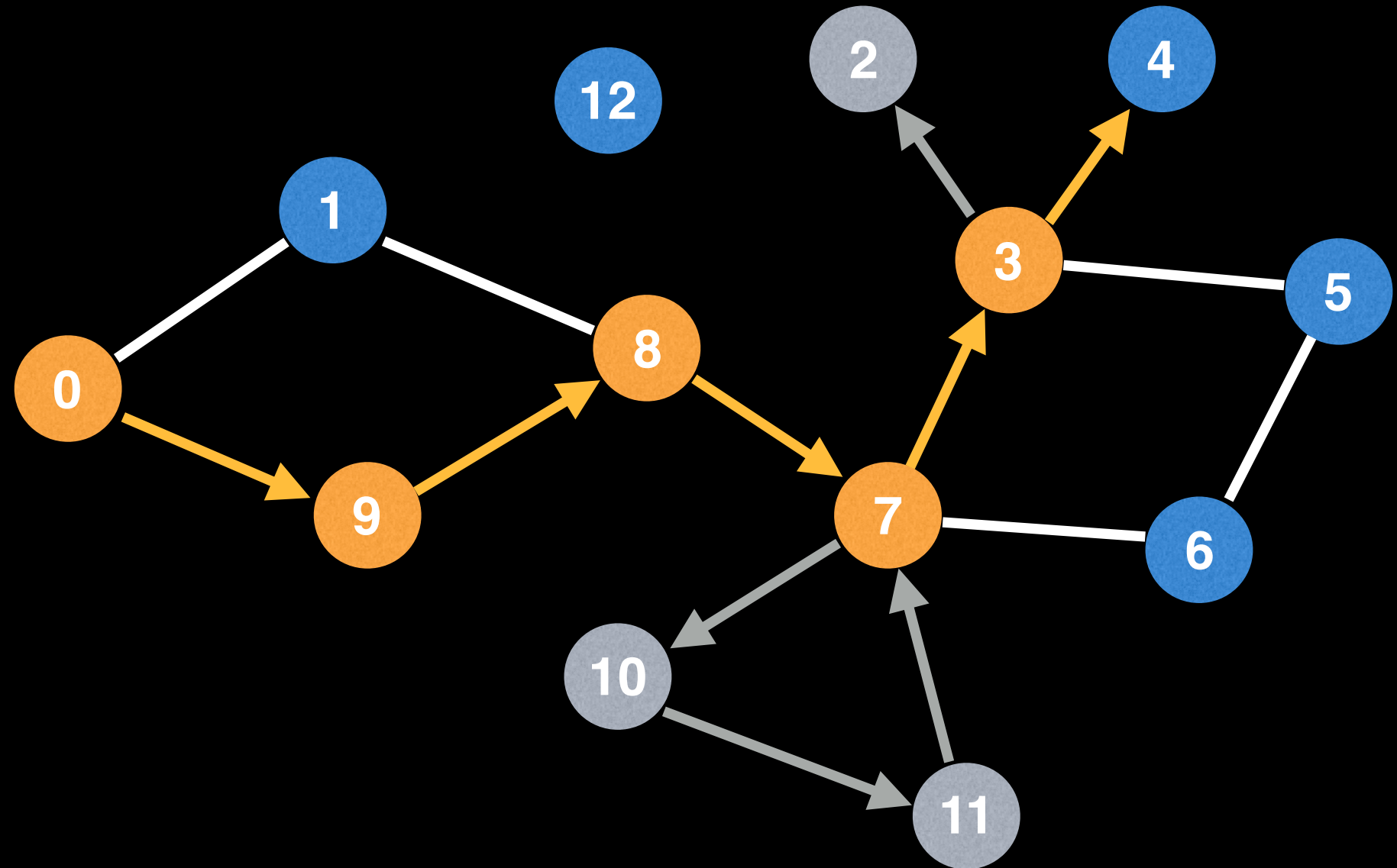
Backtrack when a dead end is reached.



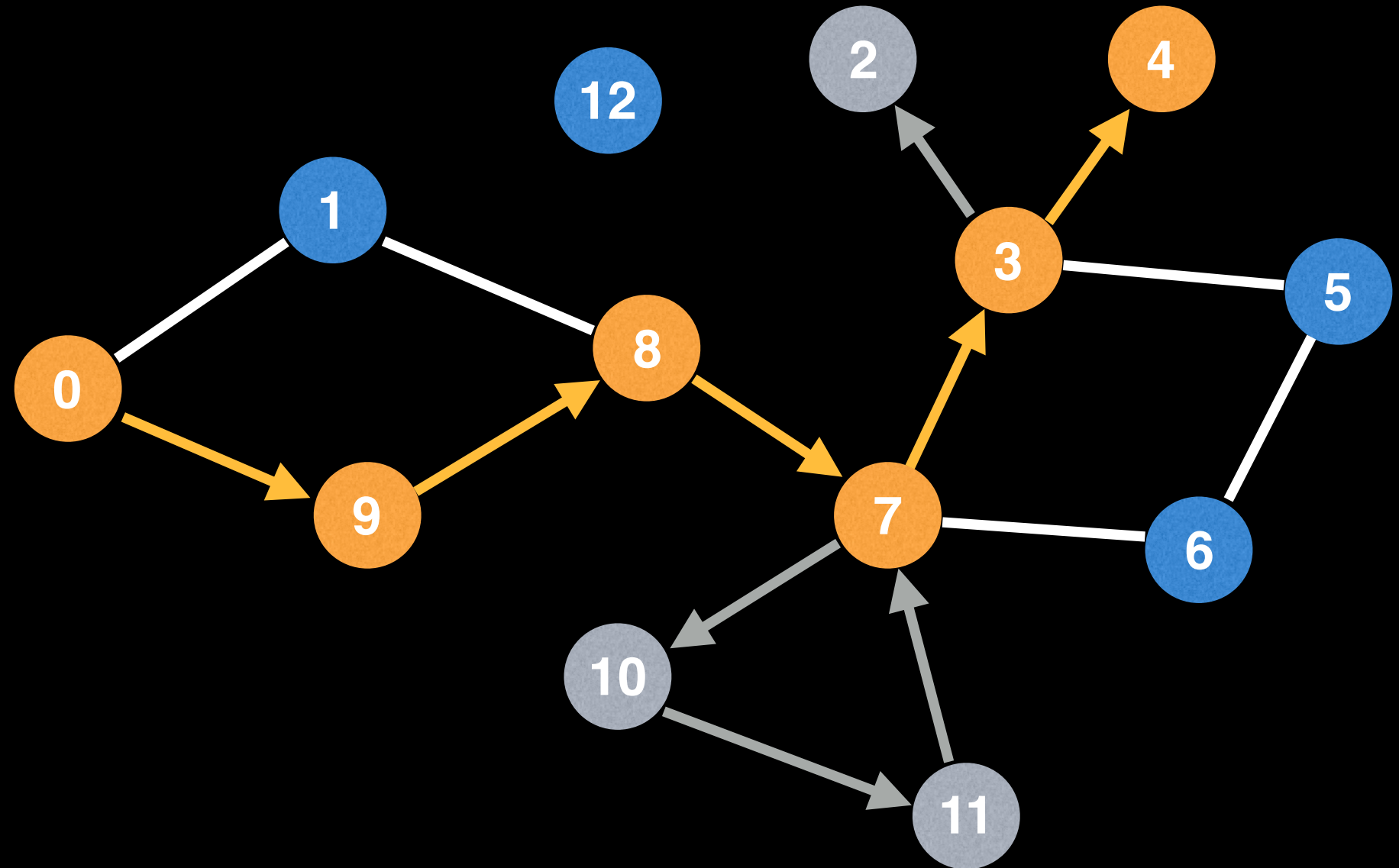
# Basic DFS



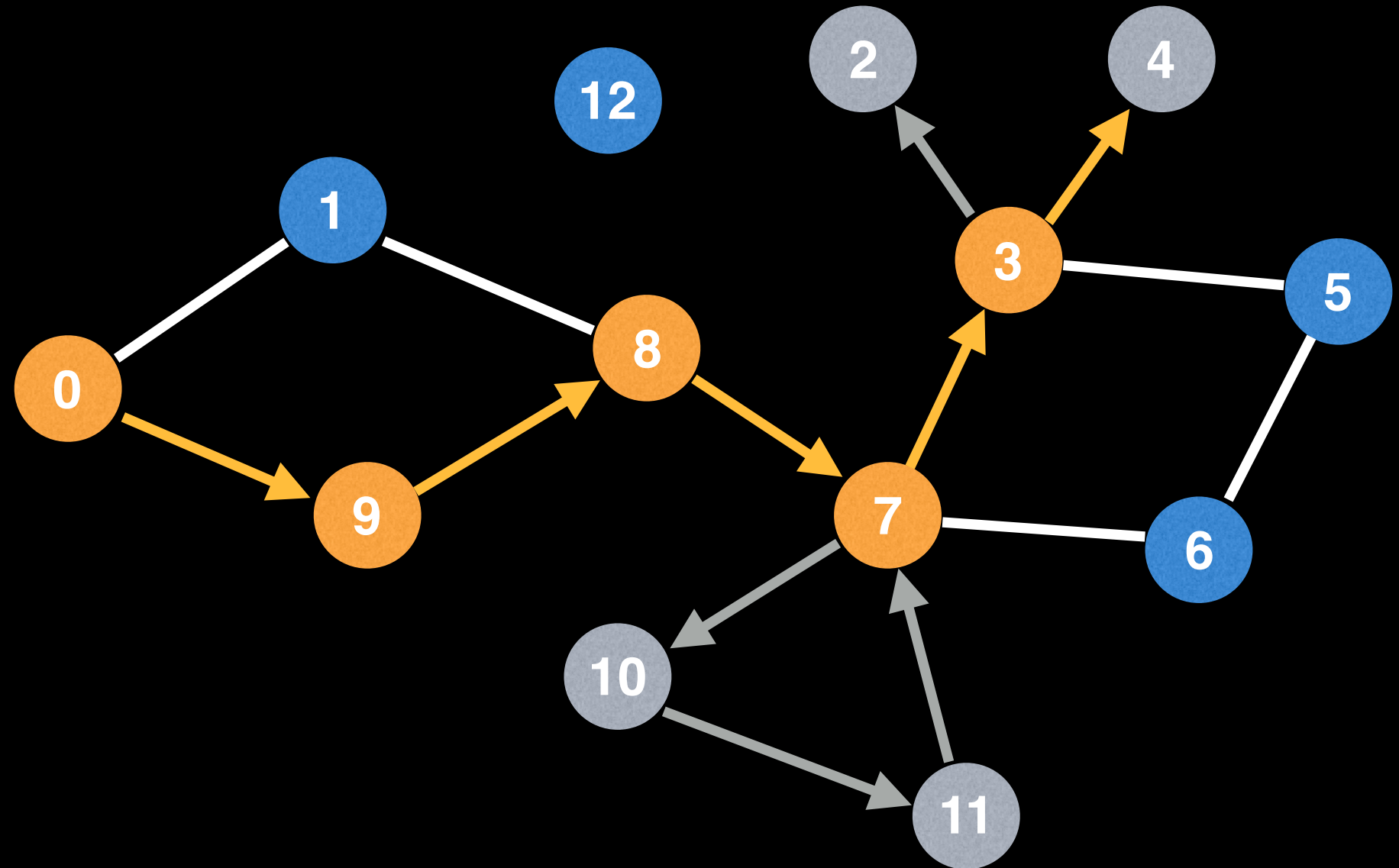
# Basic DFS



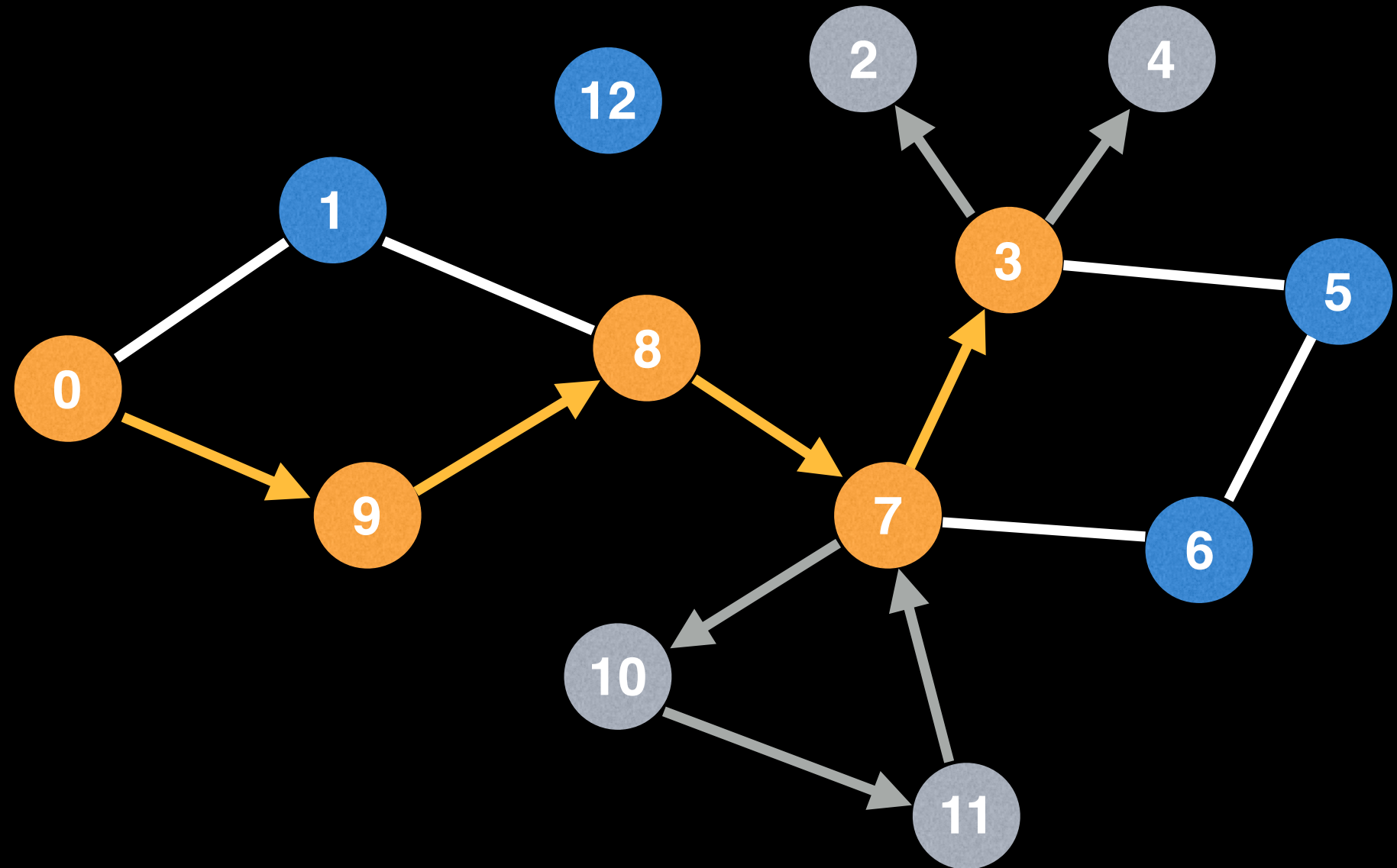
# Basic DFS



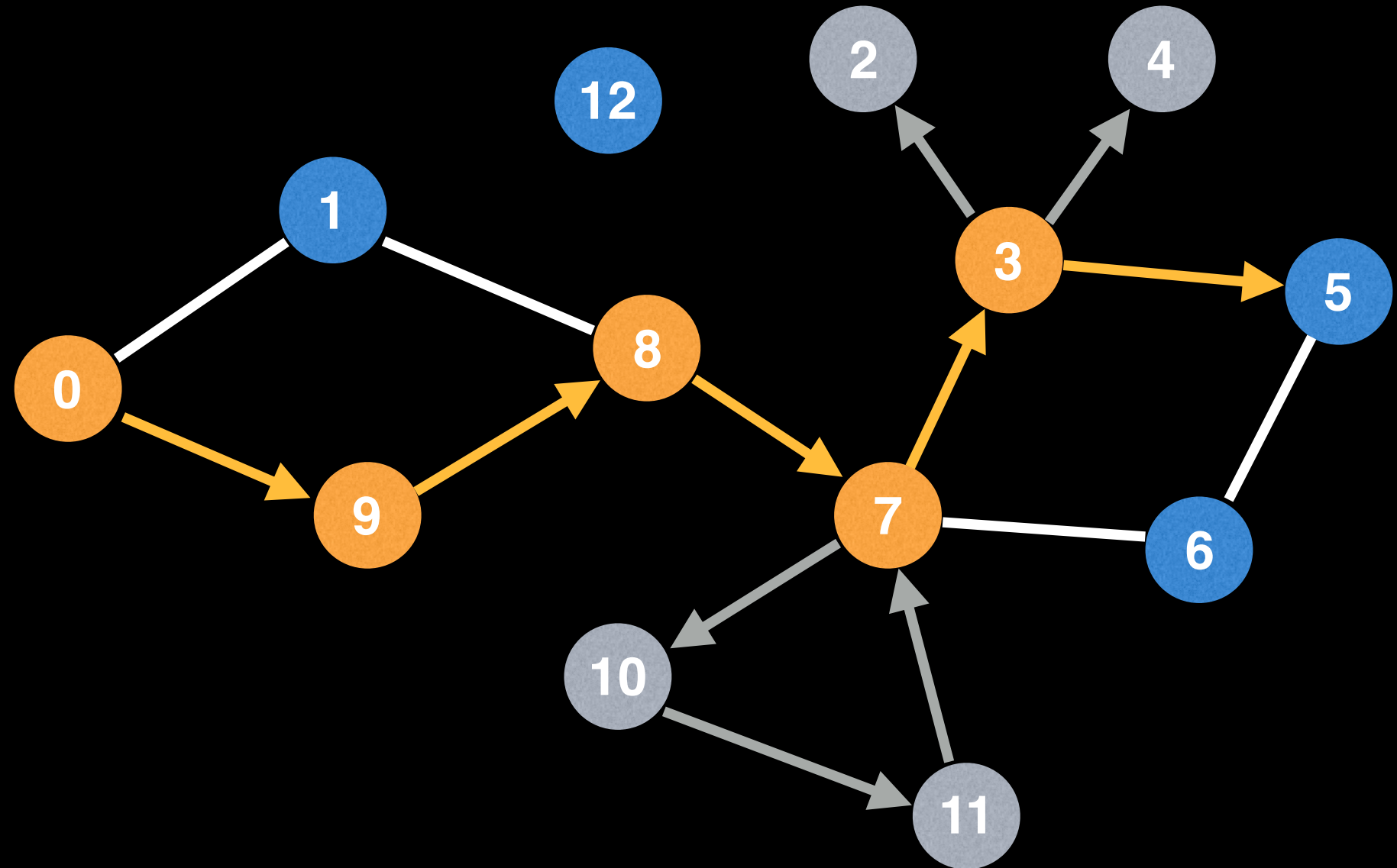
# Basic DFS



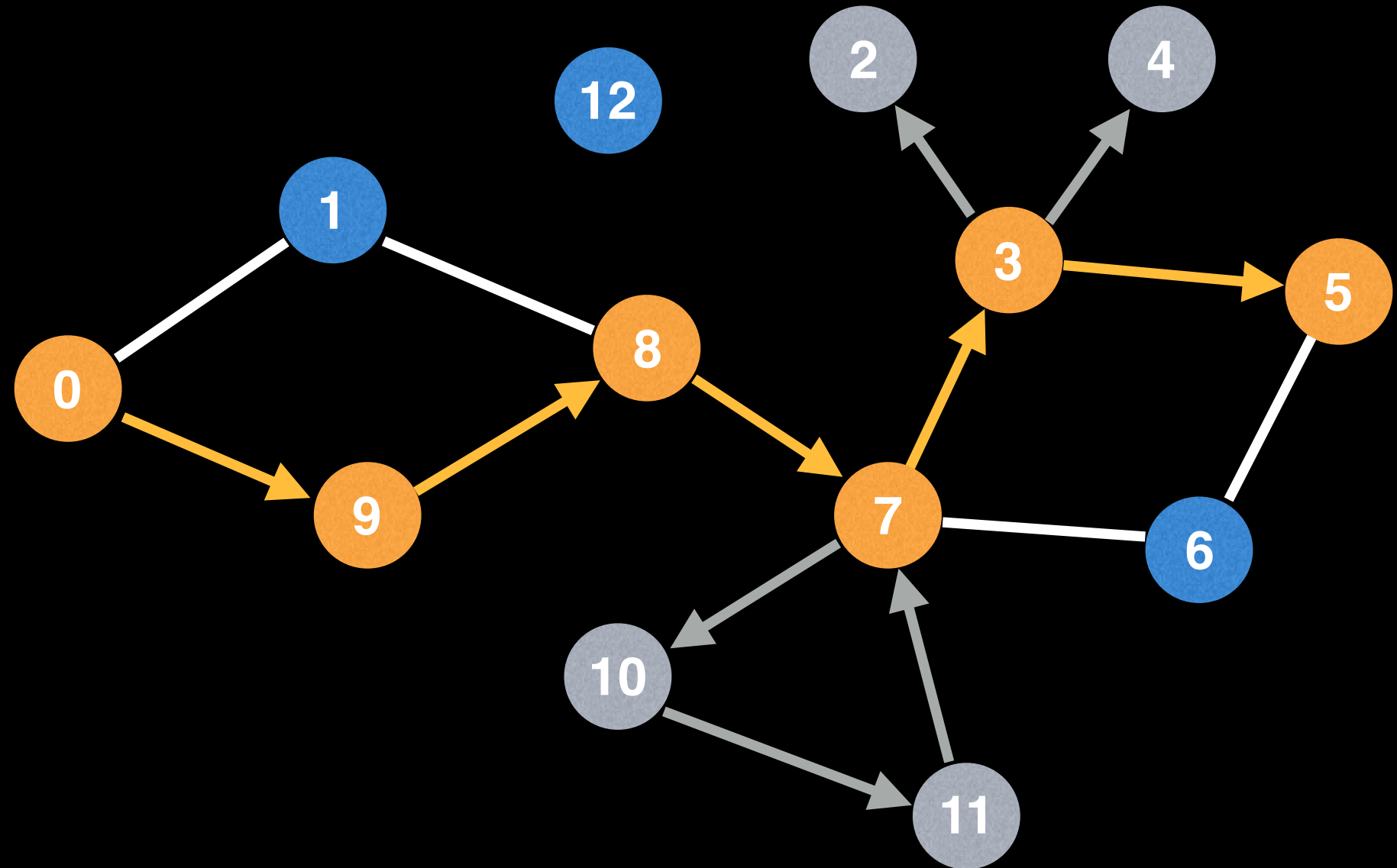
# Basic DFS



# Basic DFS

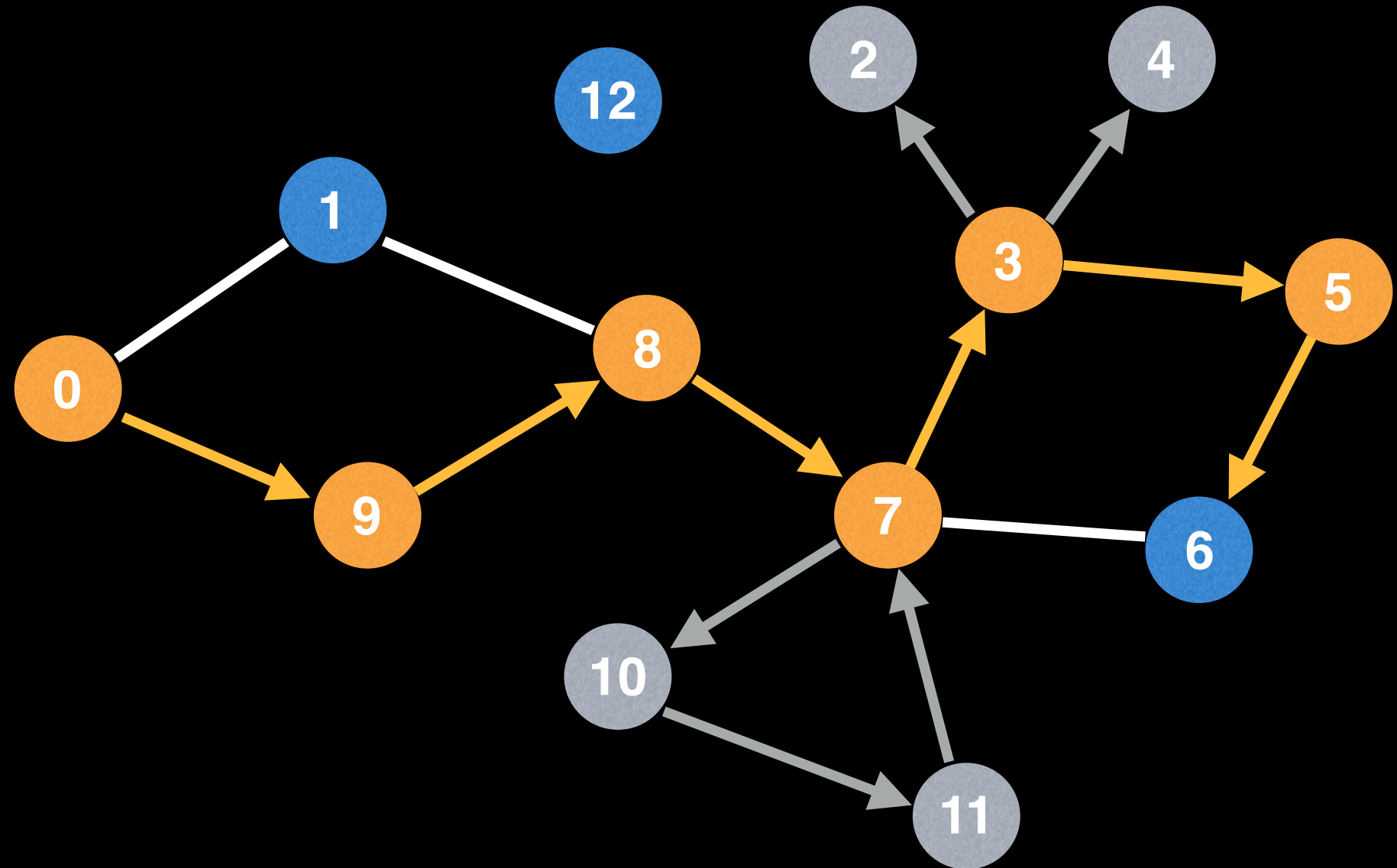


# Basic DFS

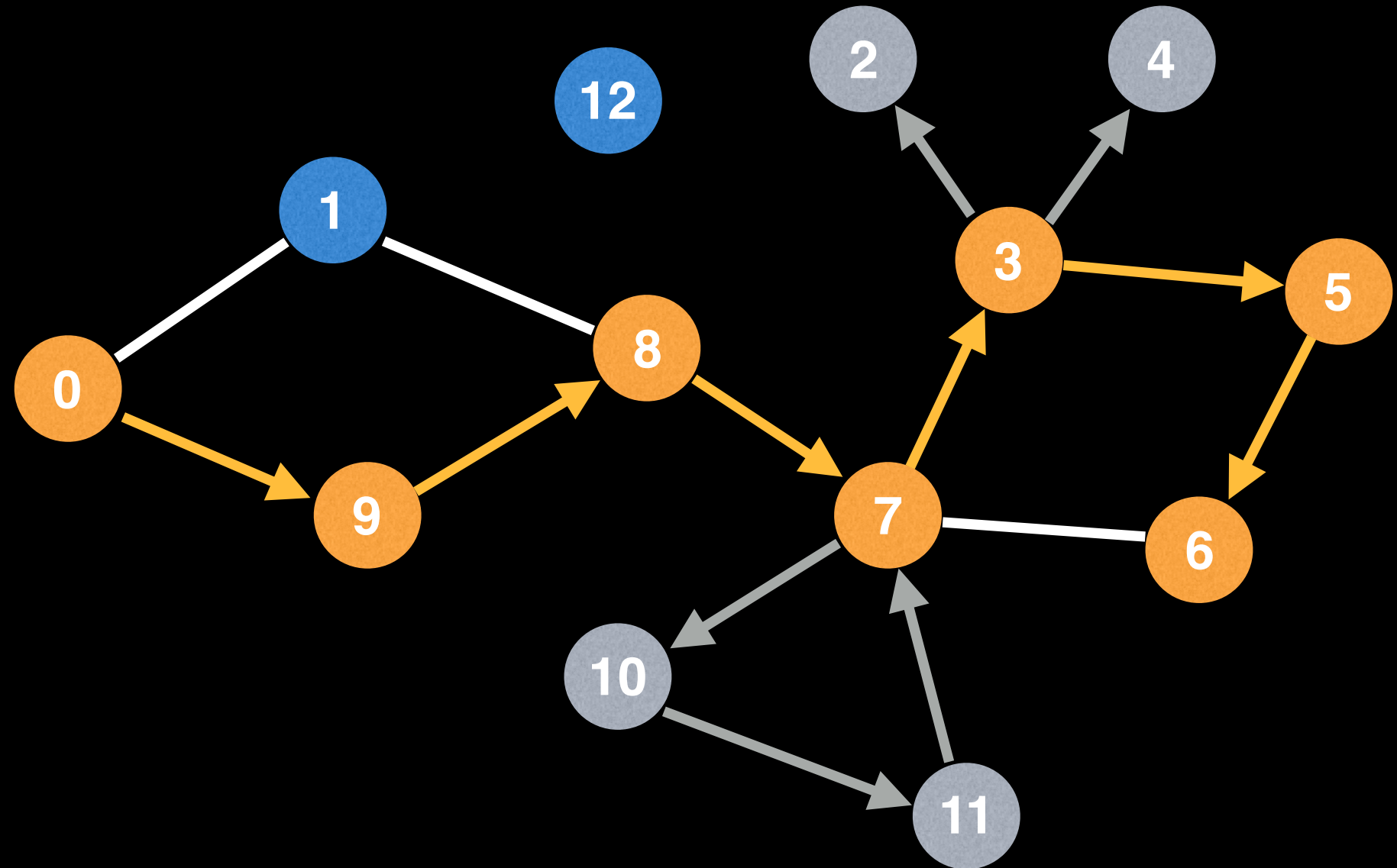




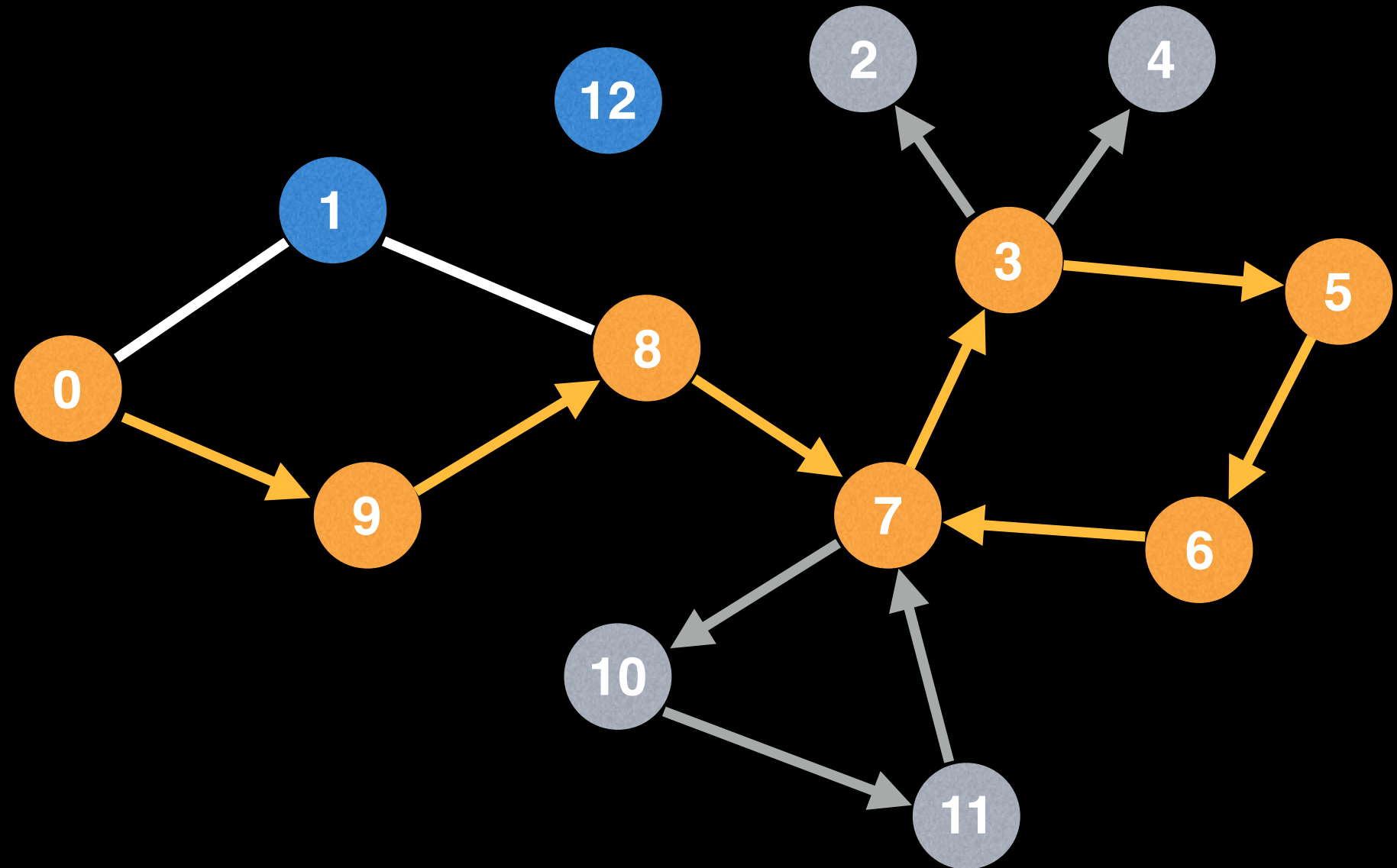
# Basic DFS



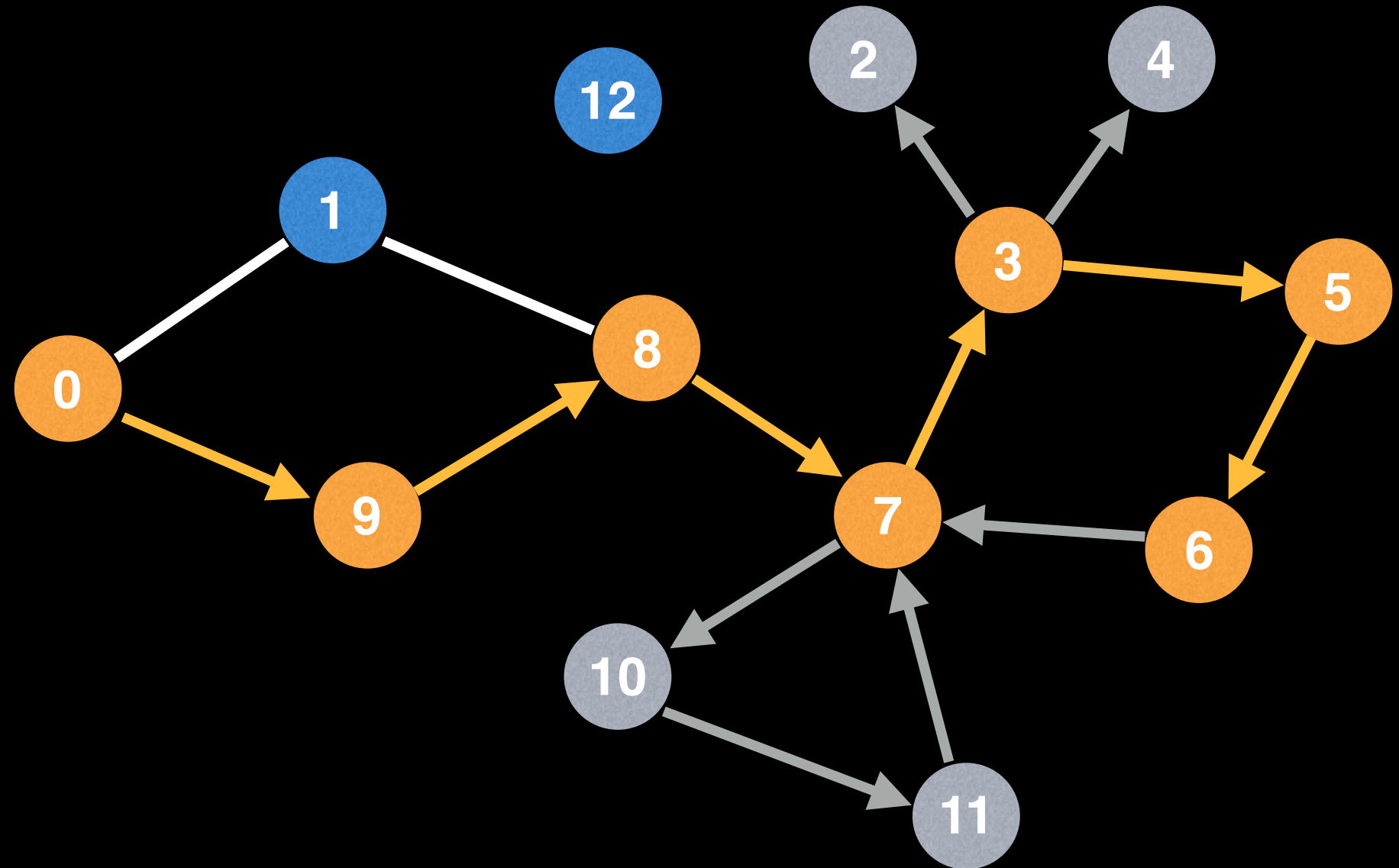
# Basic DFS



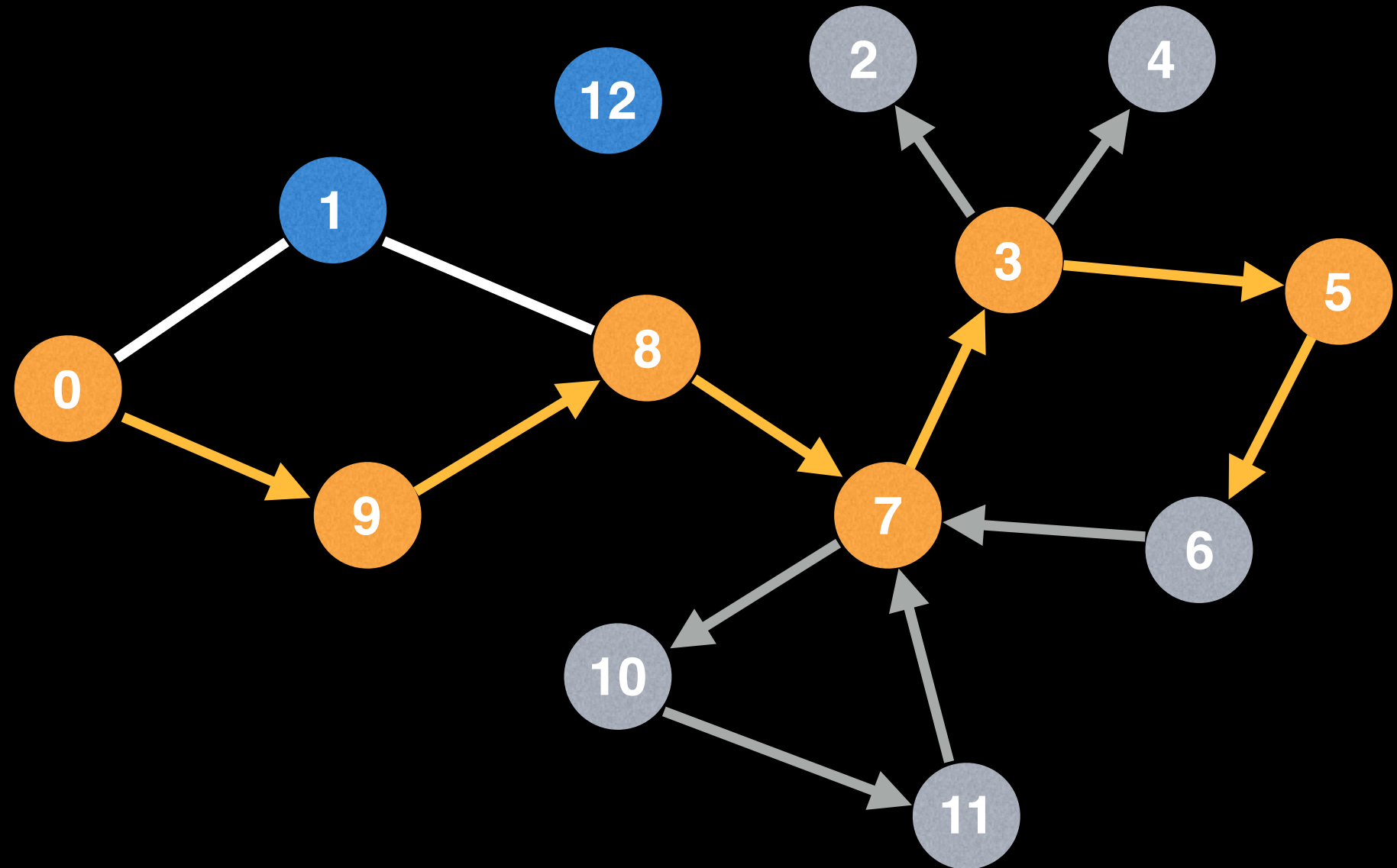
# Basic DFS



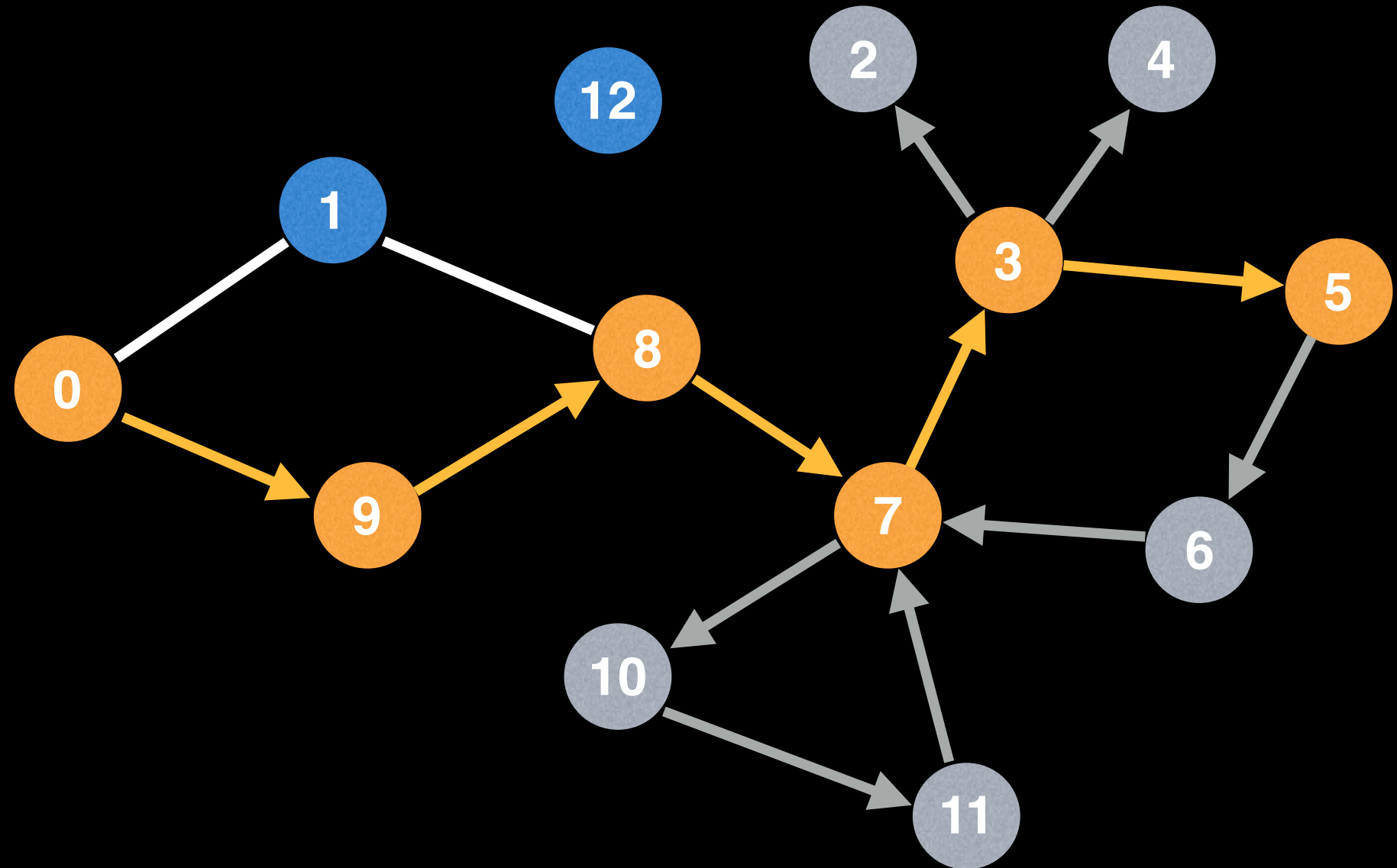
# Basic DFS



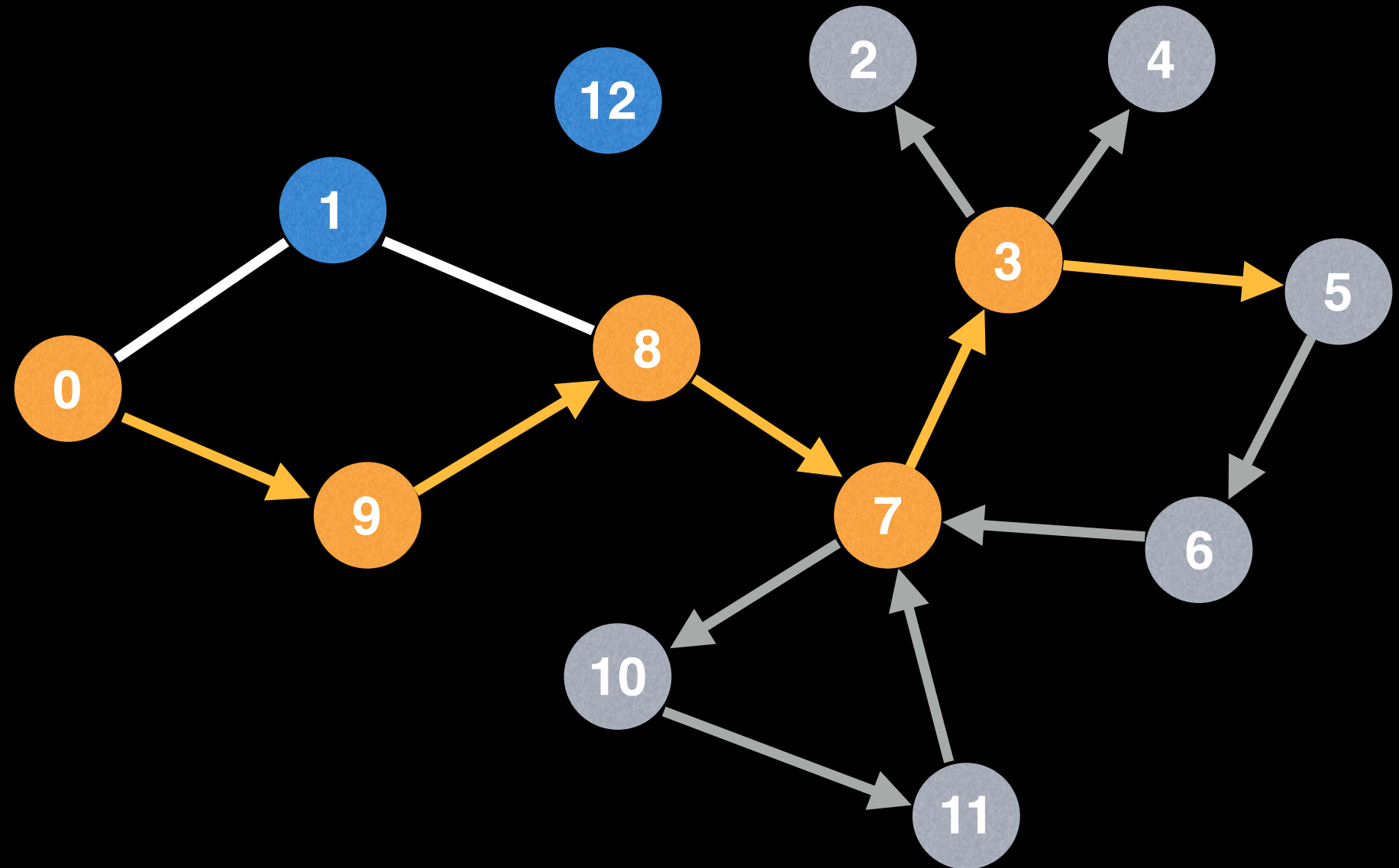
# Basic DFS



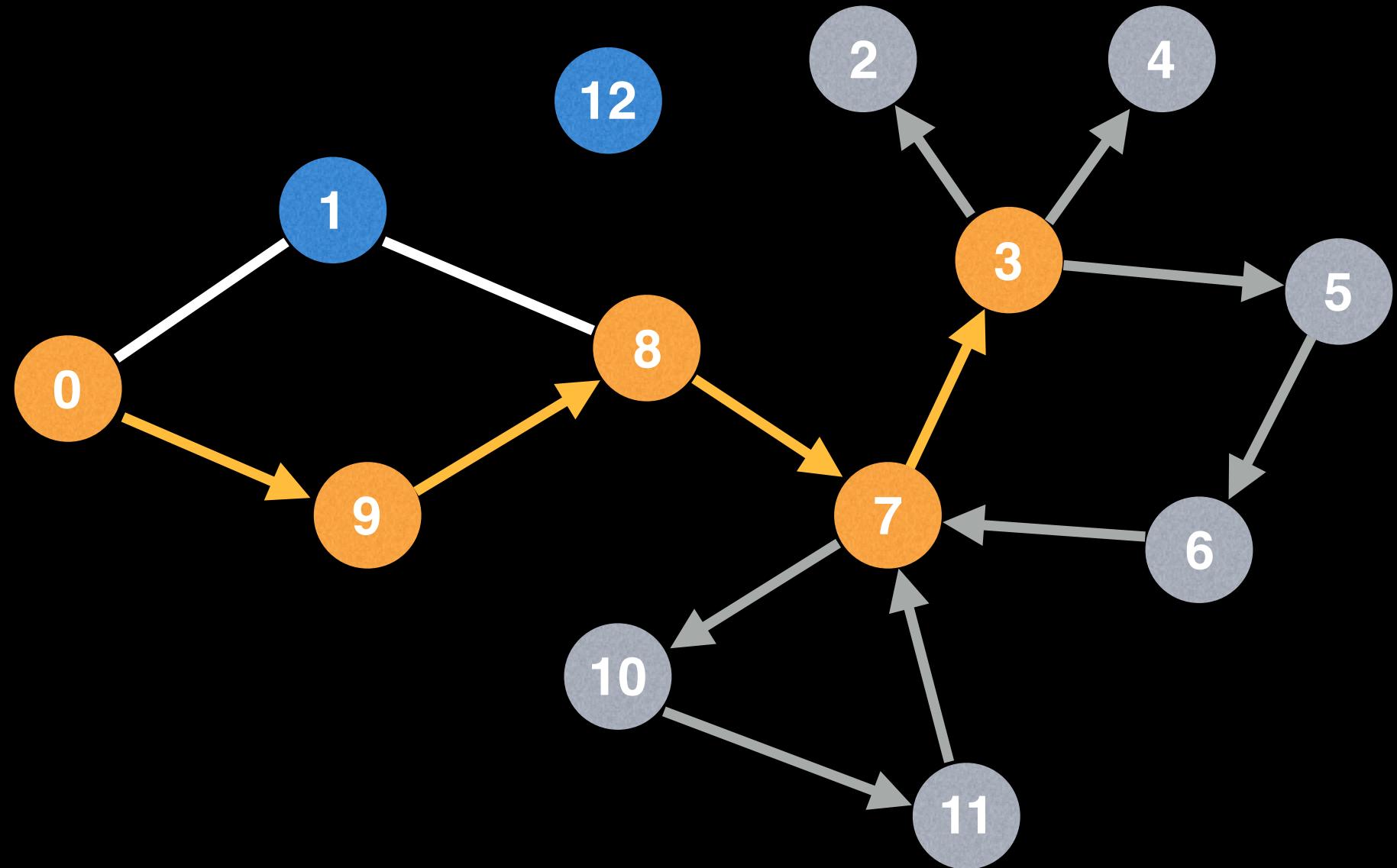
# Basic DFS



# Basic DFS

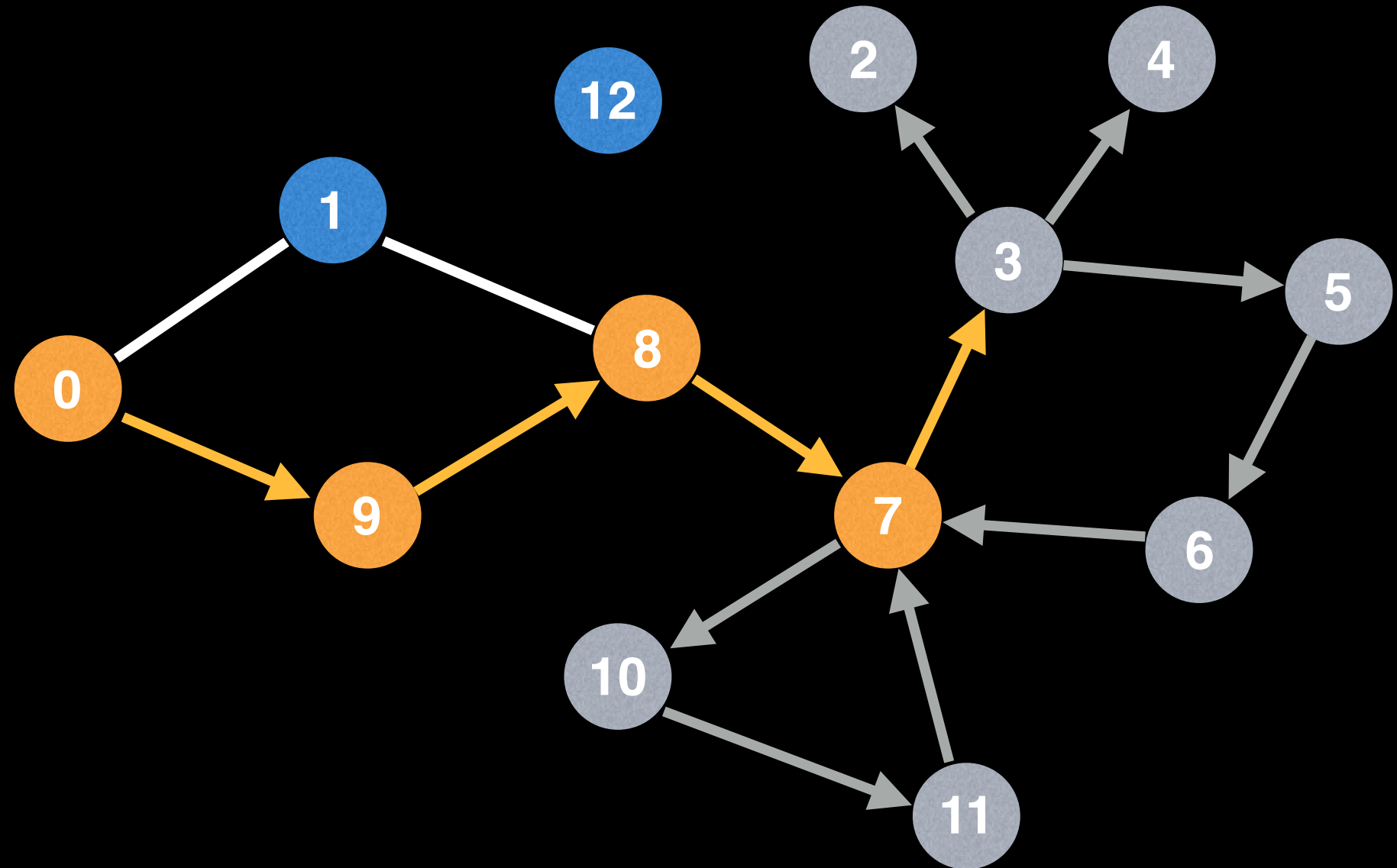


# Basic DFS

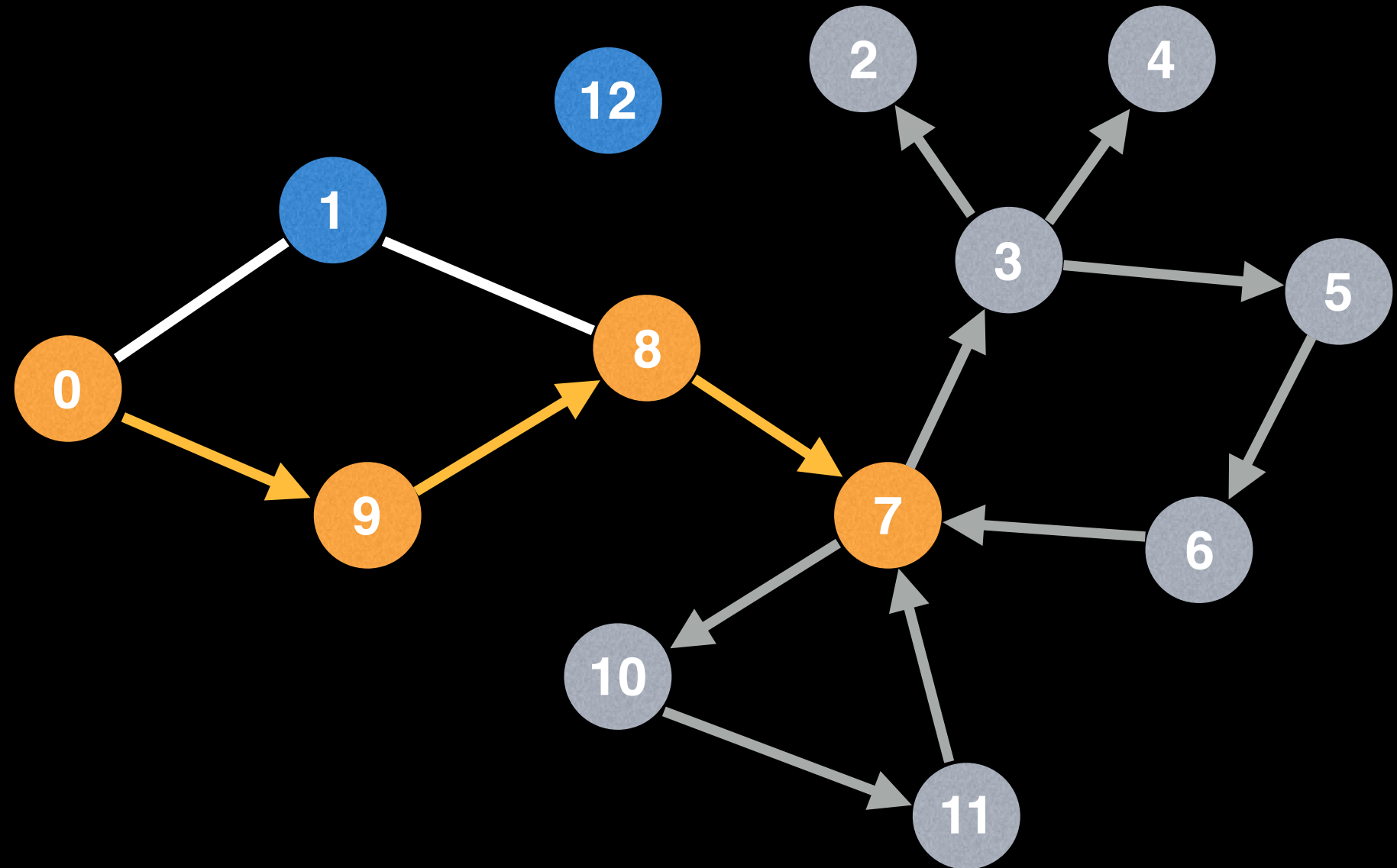




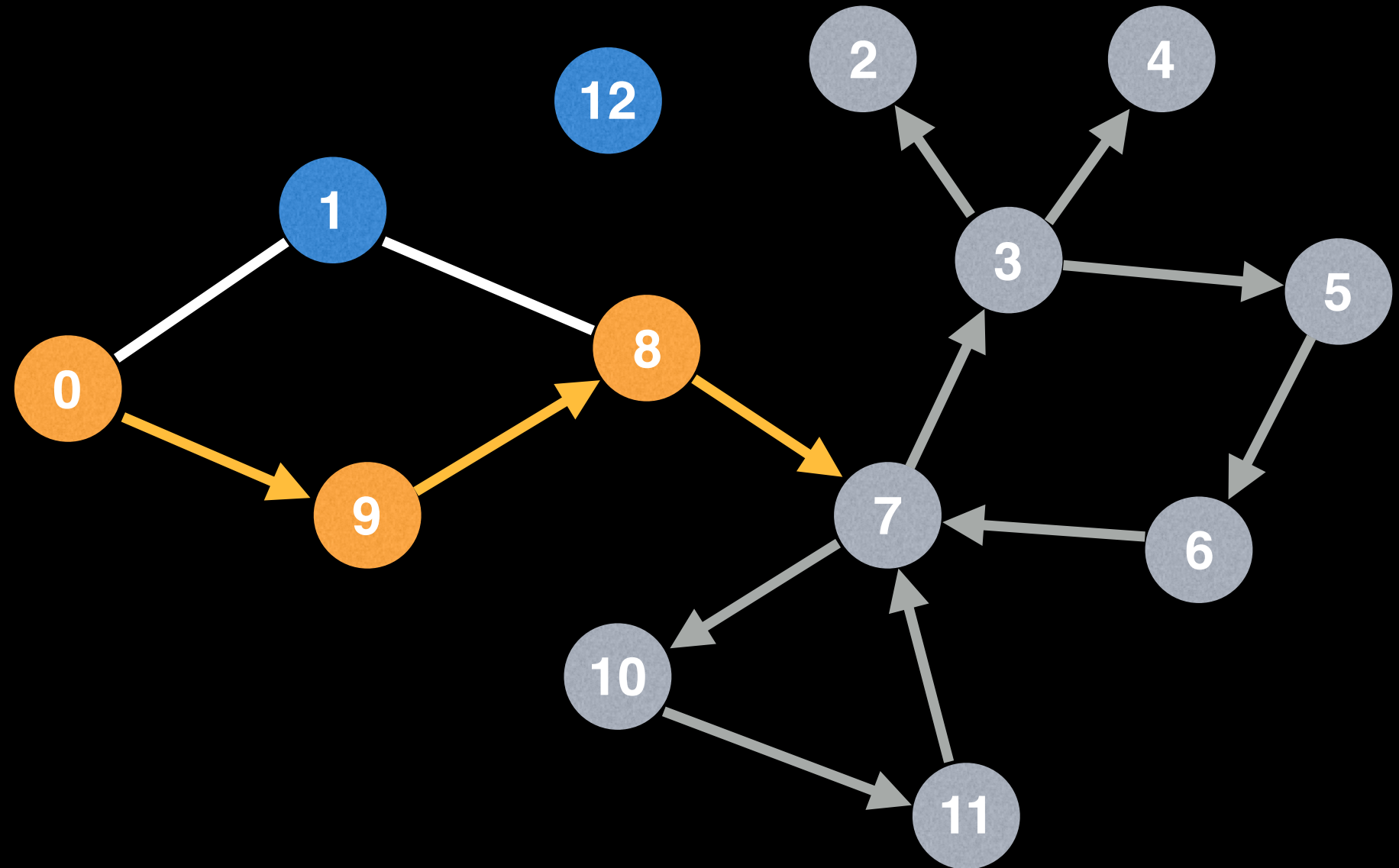
# Basic DFS



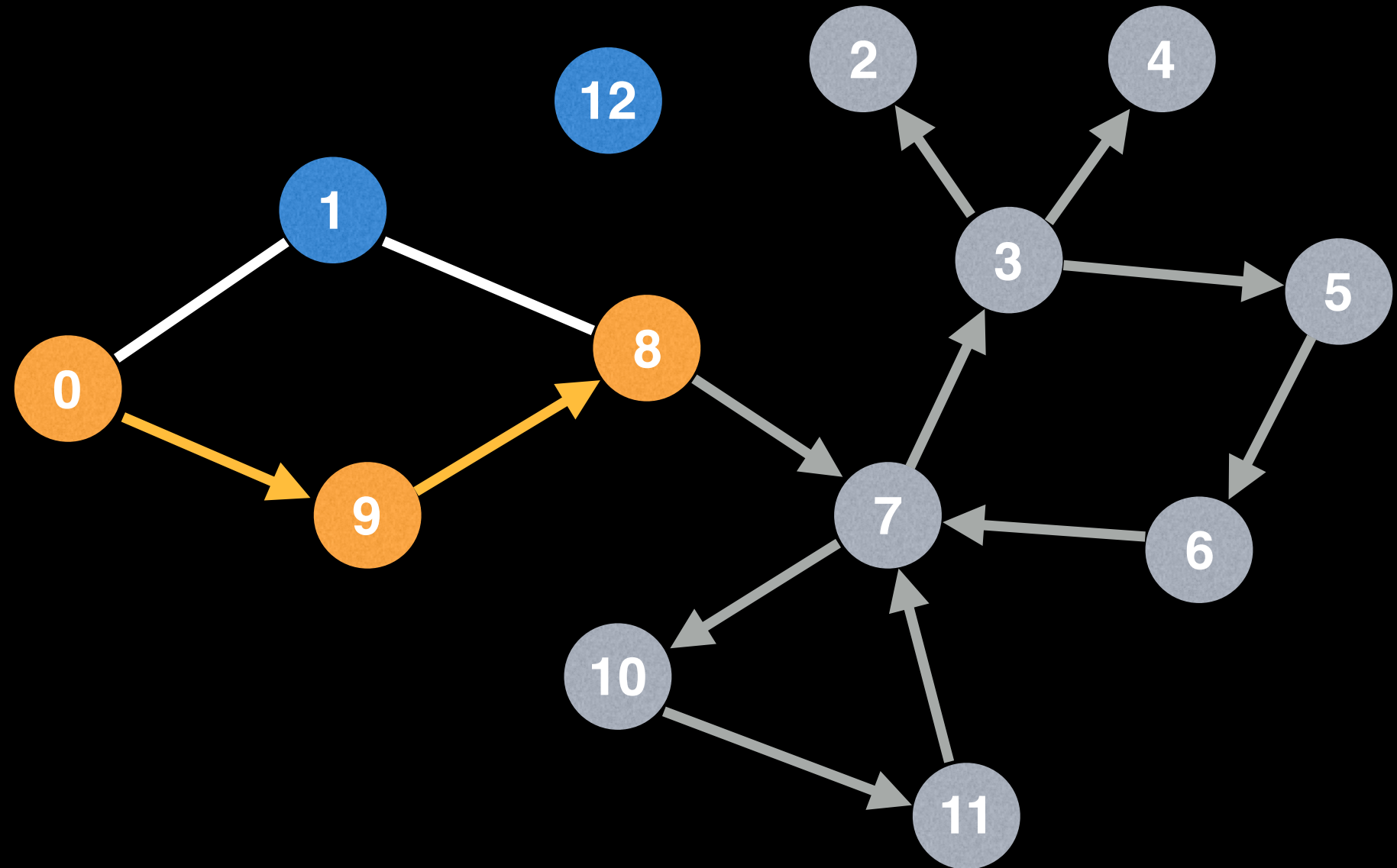
# Basic DFS



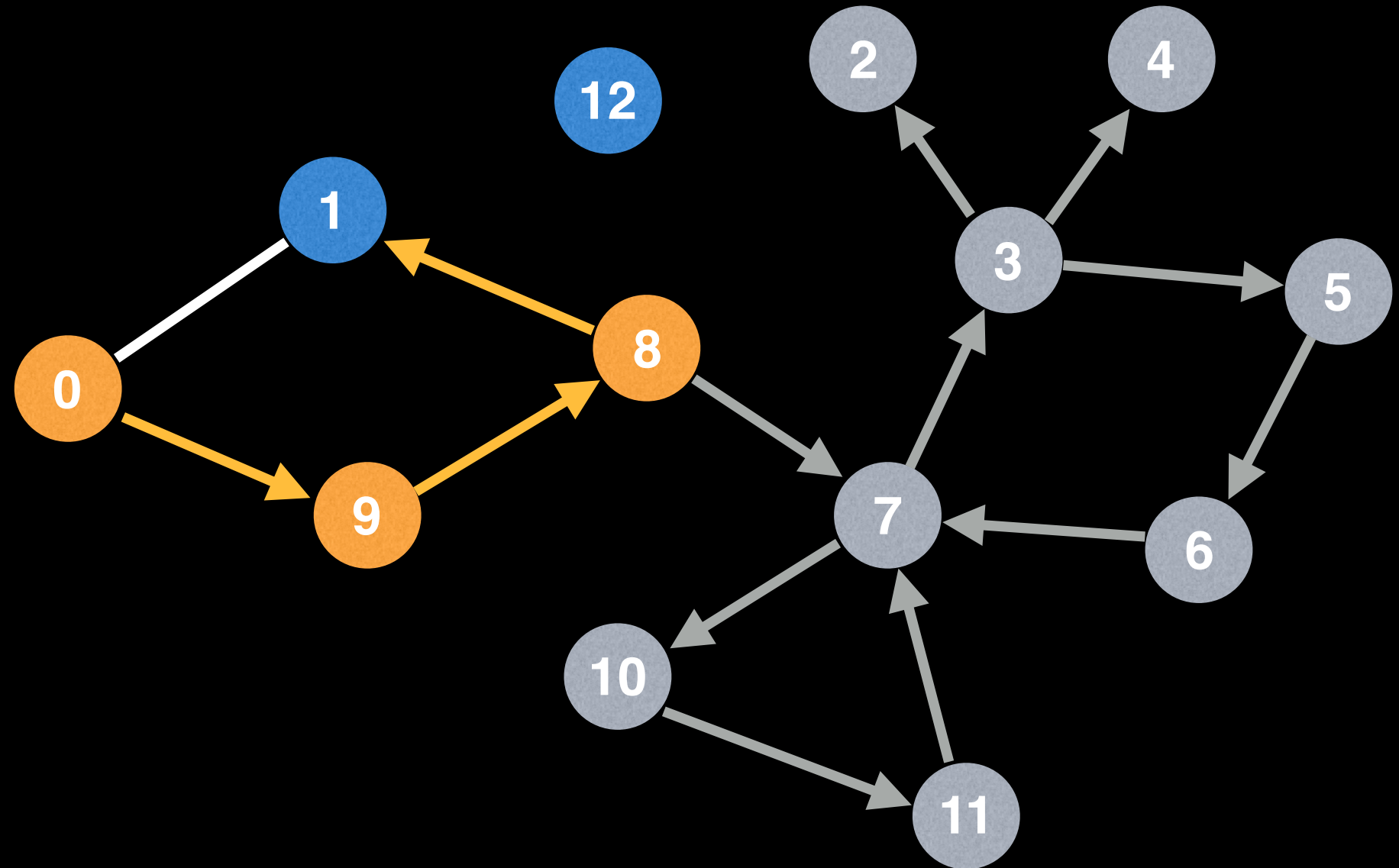
# Basic DFS



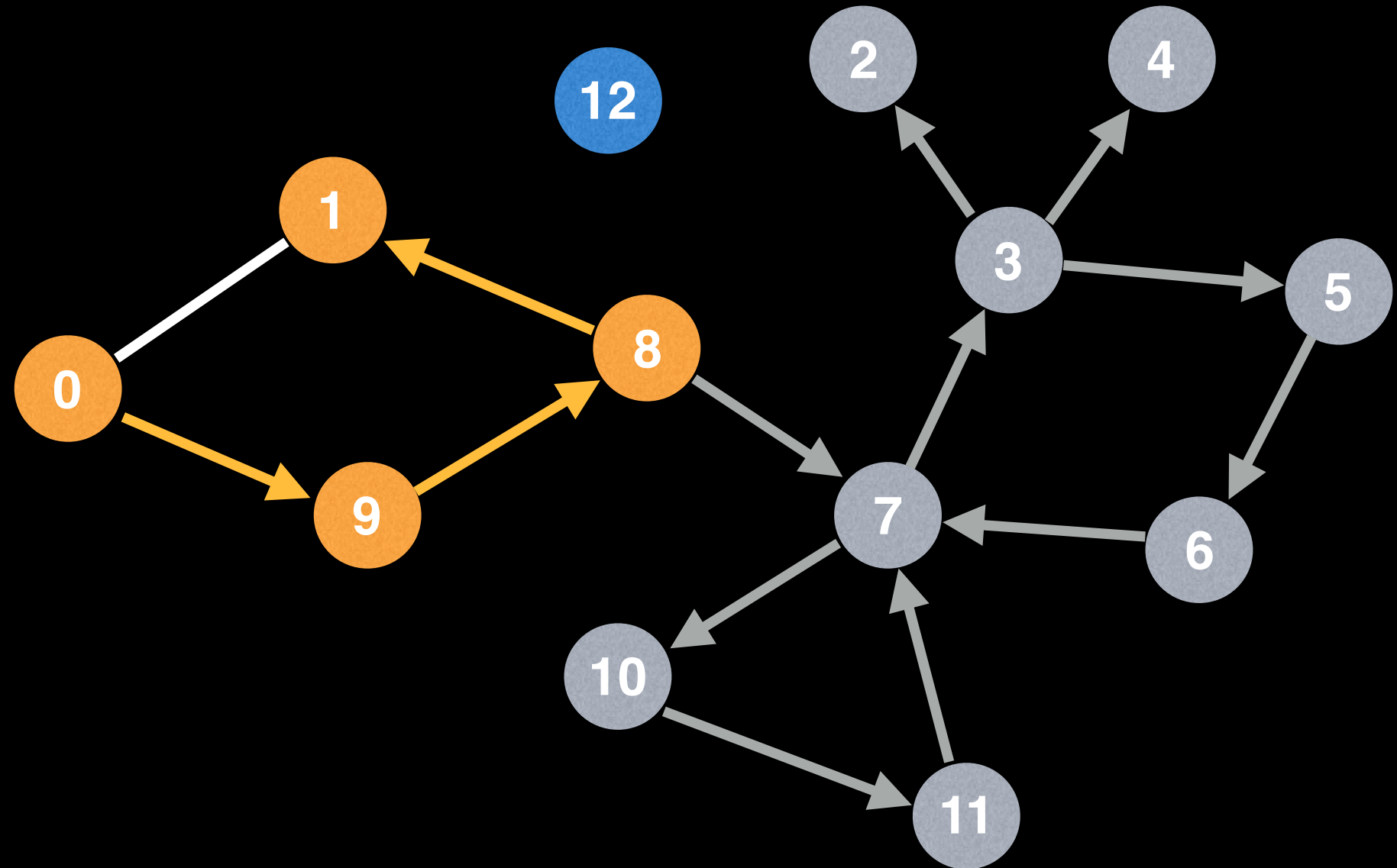
# Basic DFS



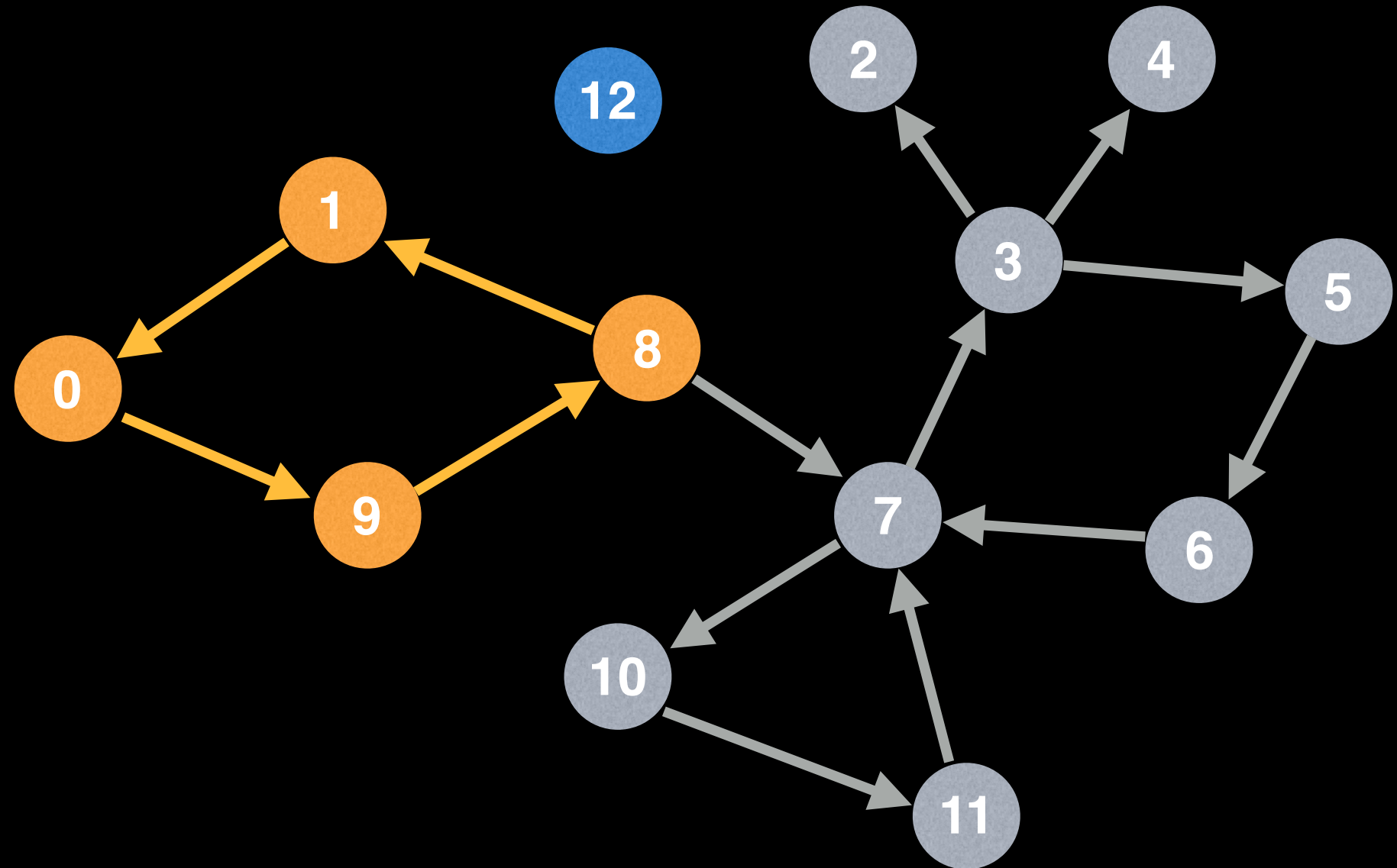
# Basic DFS



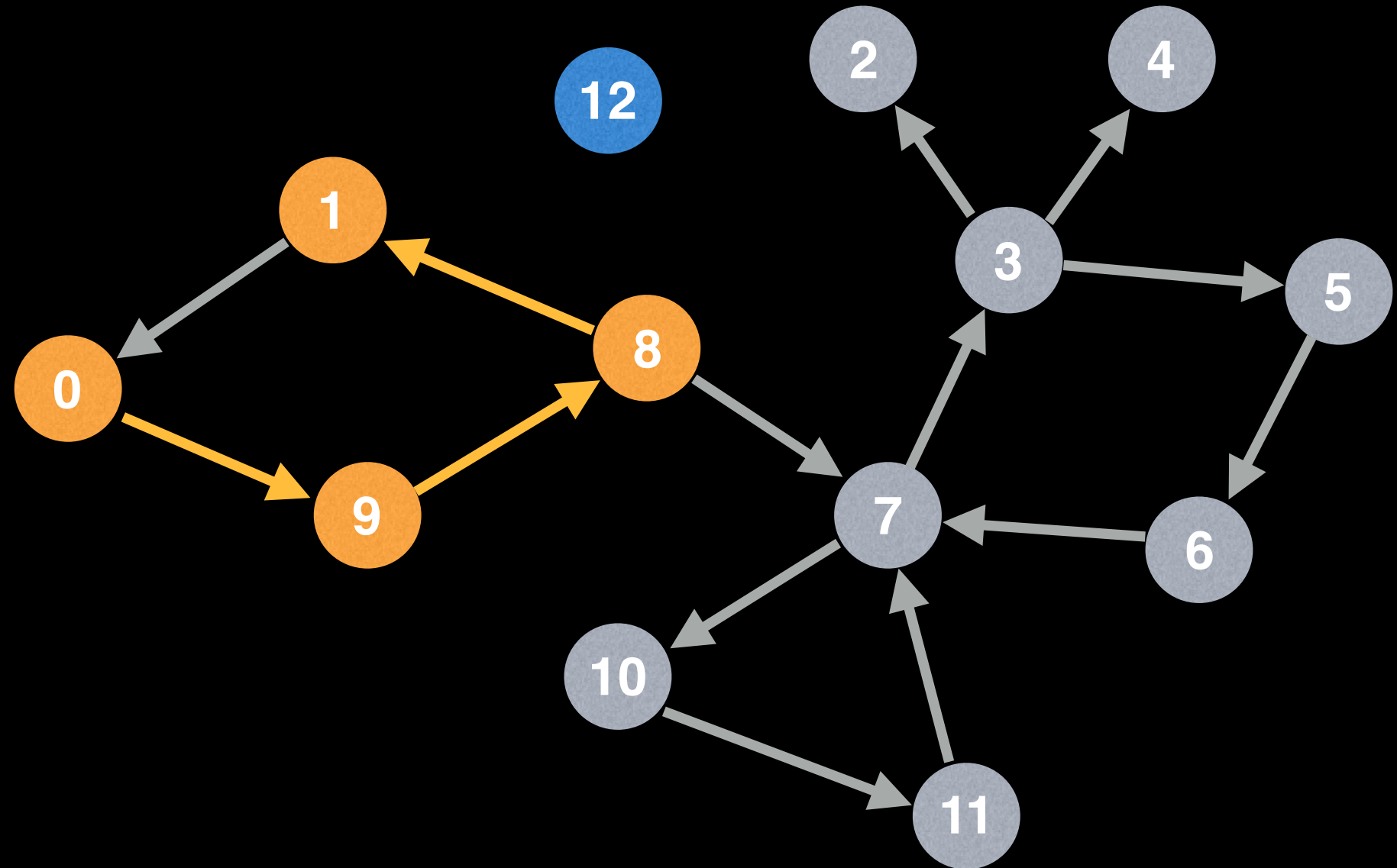
# Basic DFS



# Basic DFS

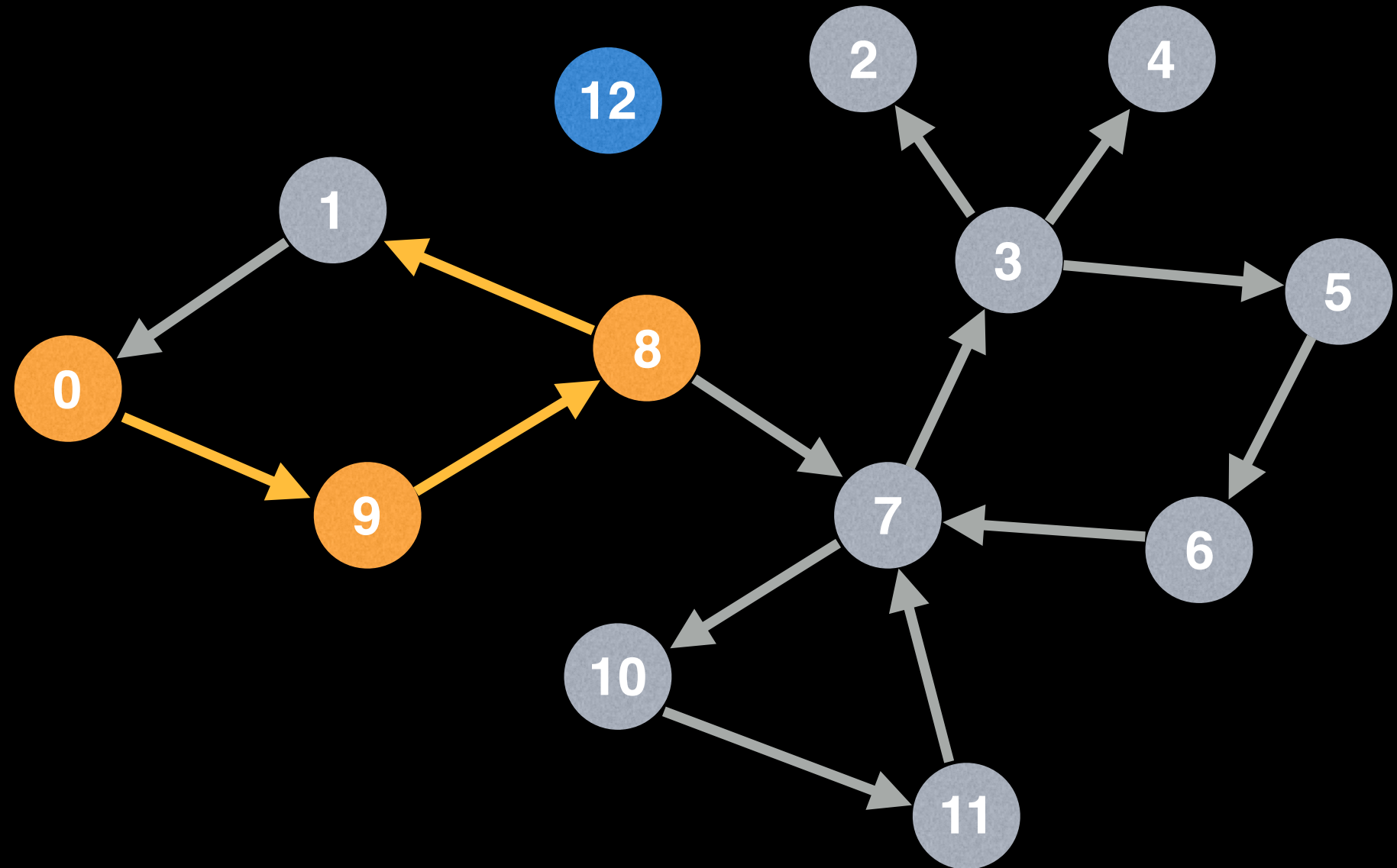


# Basic DFS

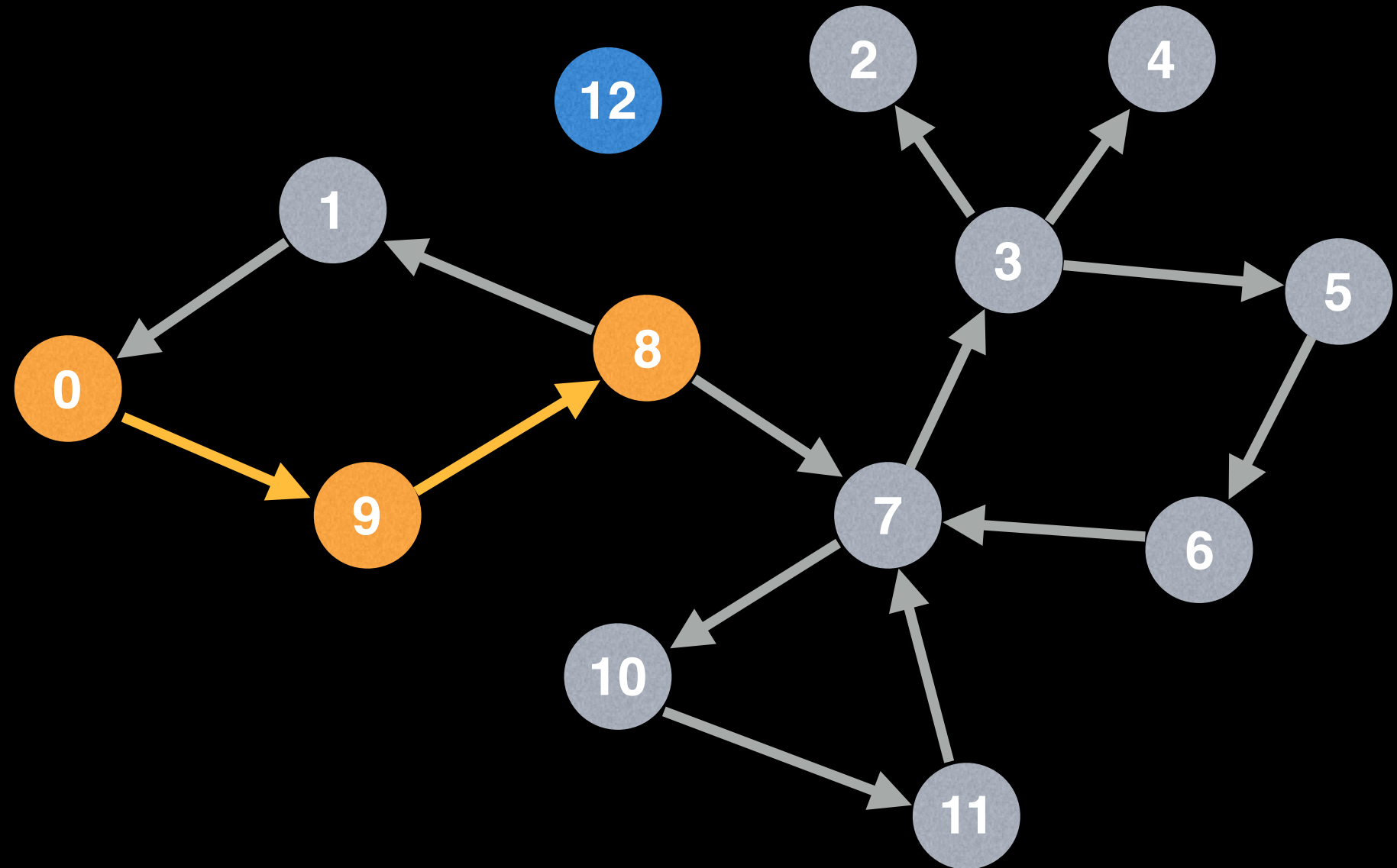




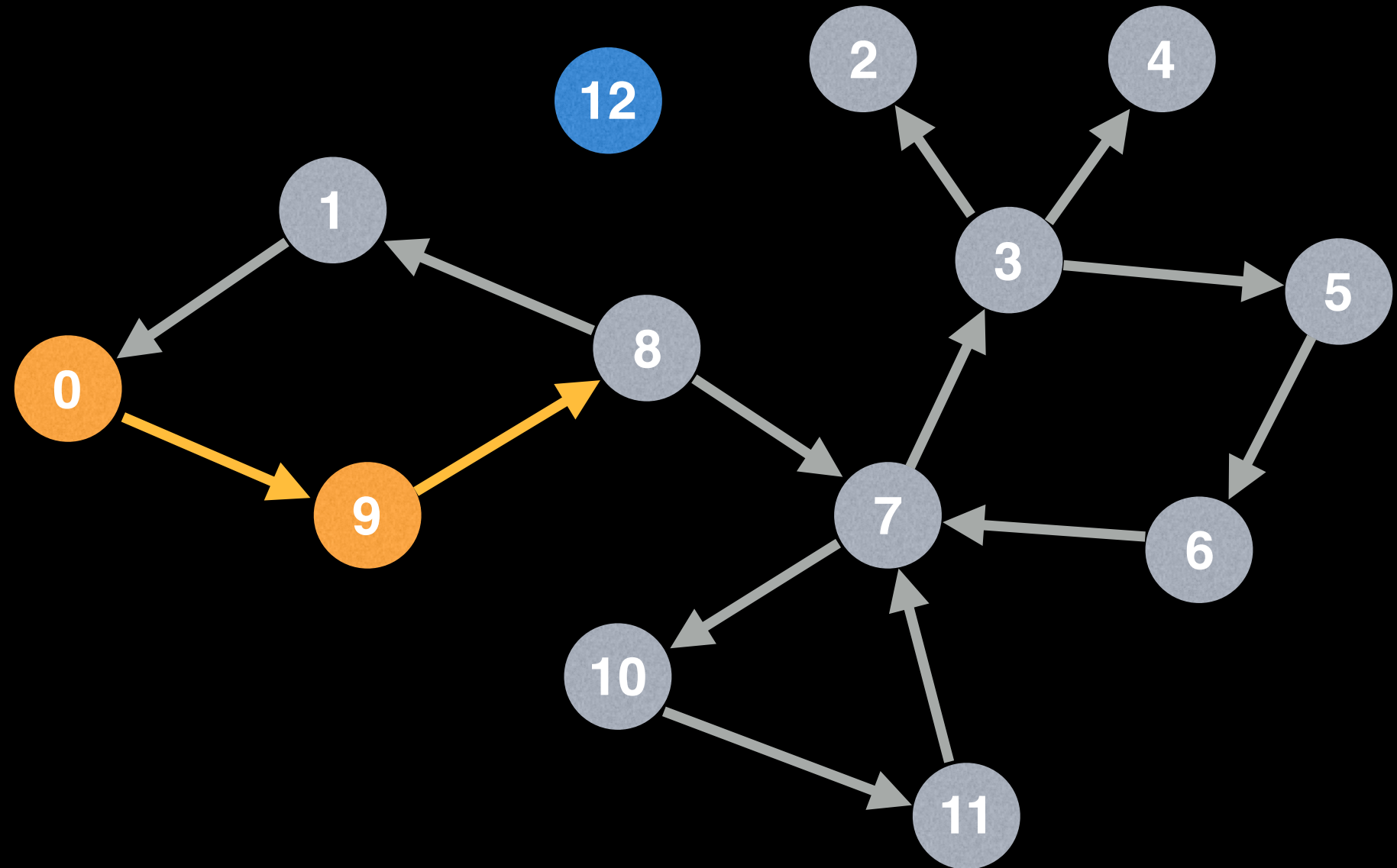
# Basic DFS



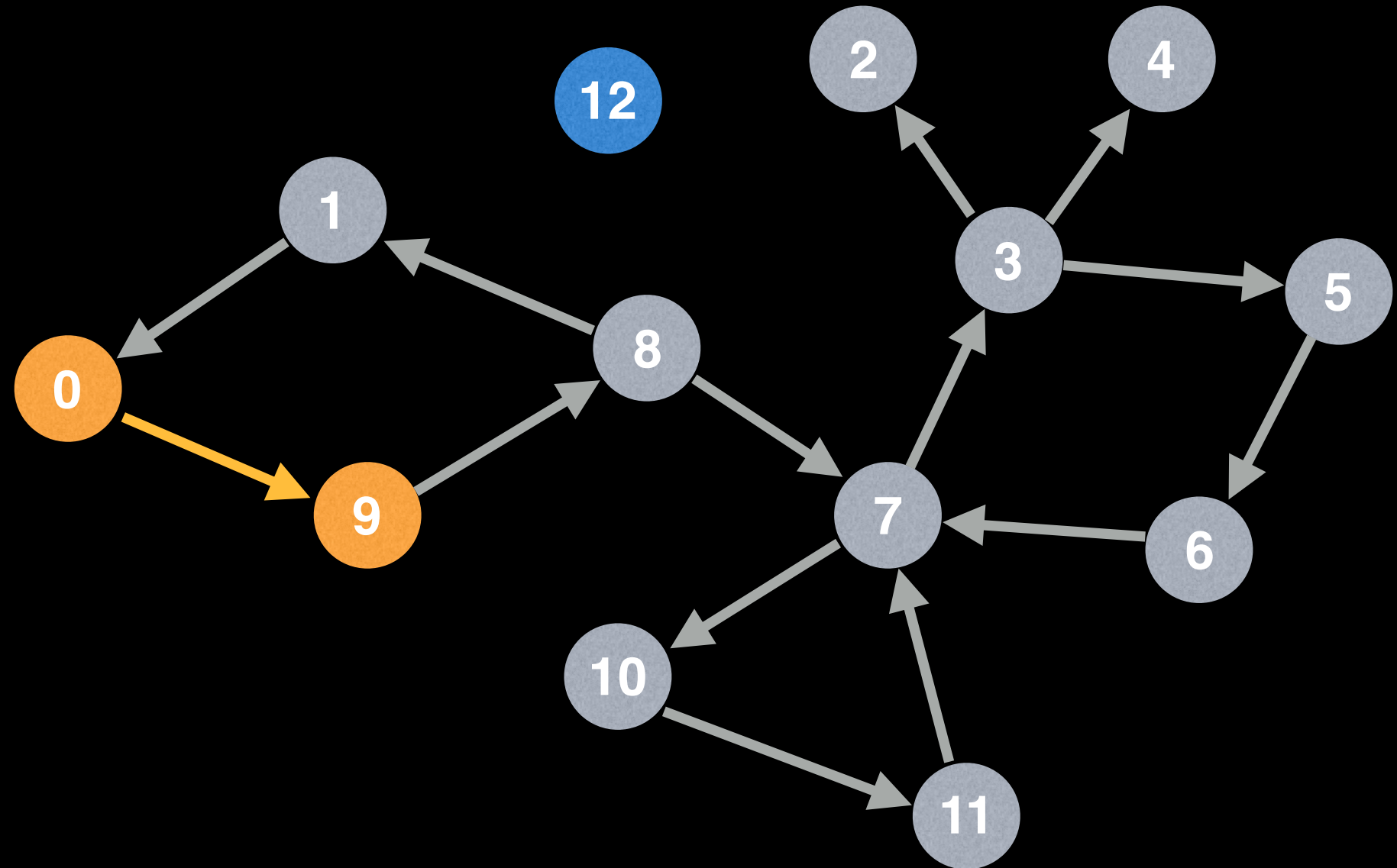
# Basic DFS



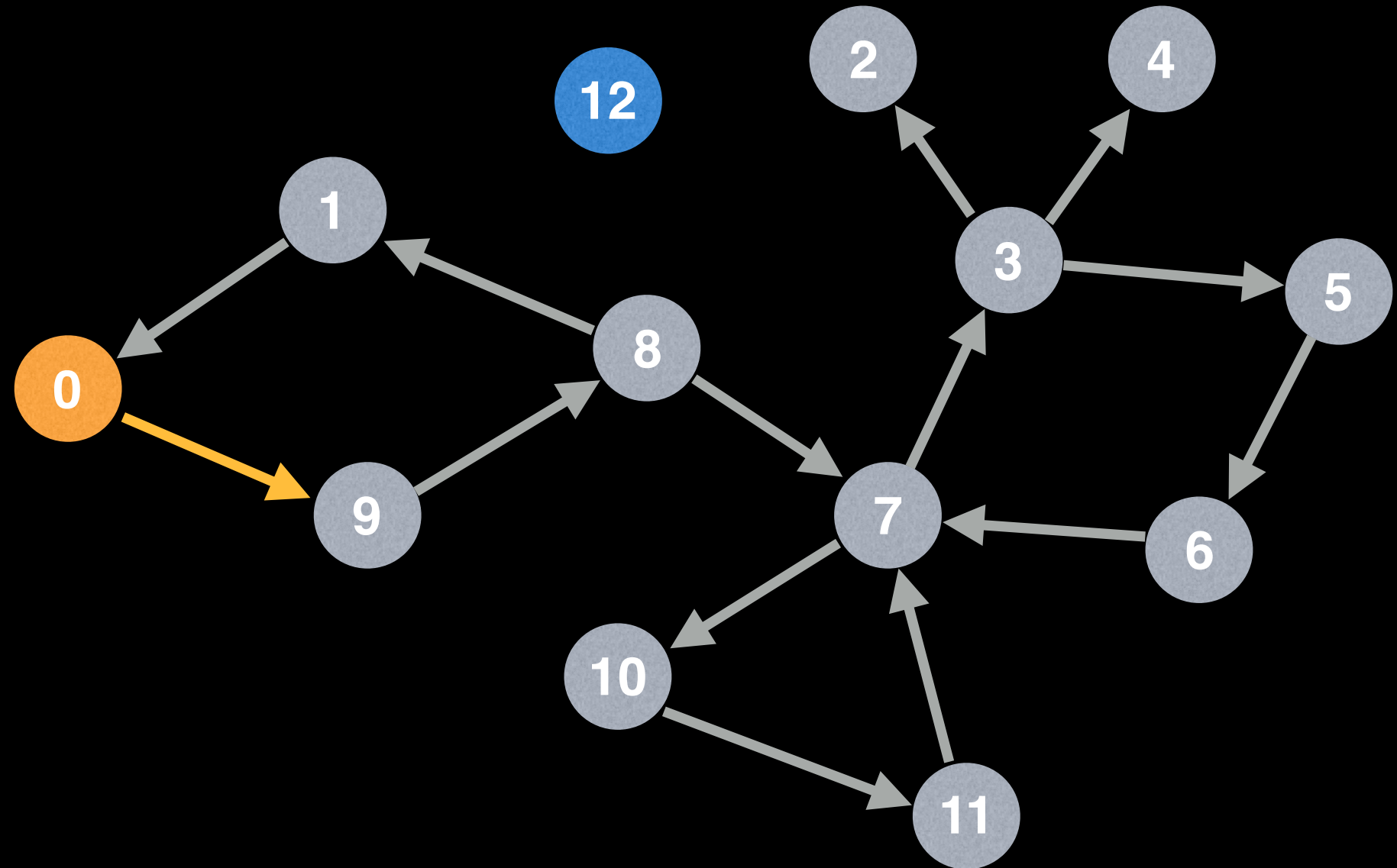
# Basic DFS



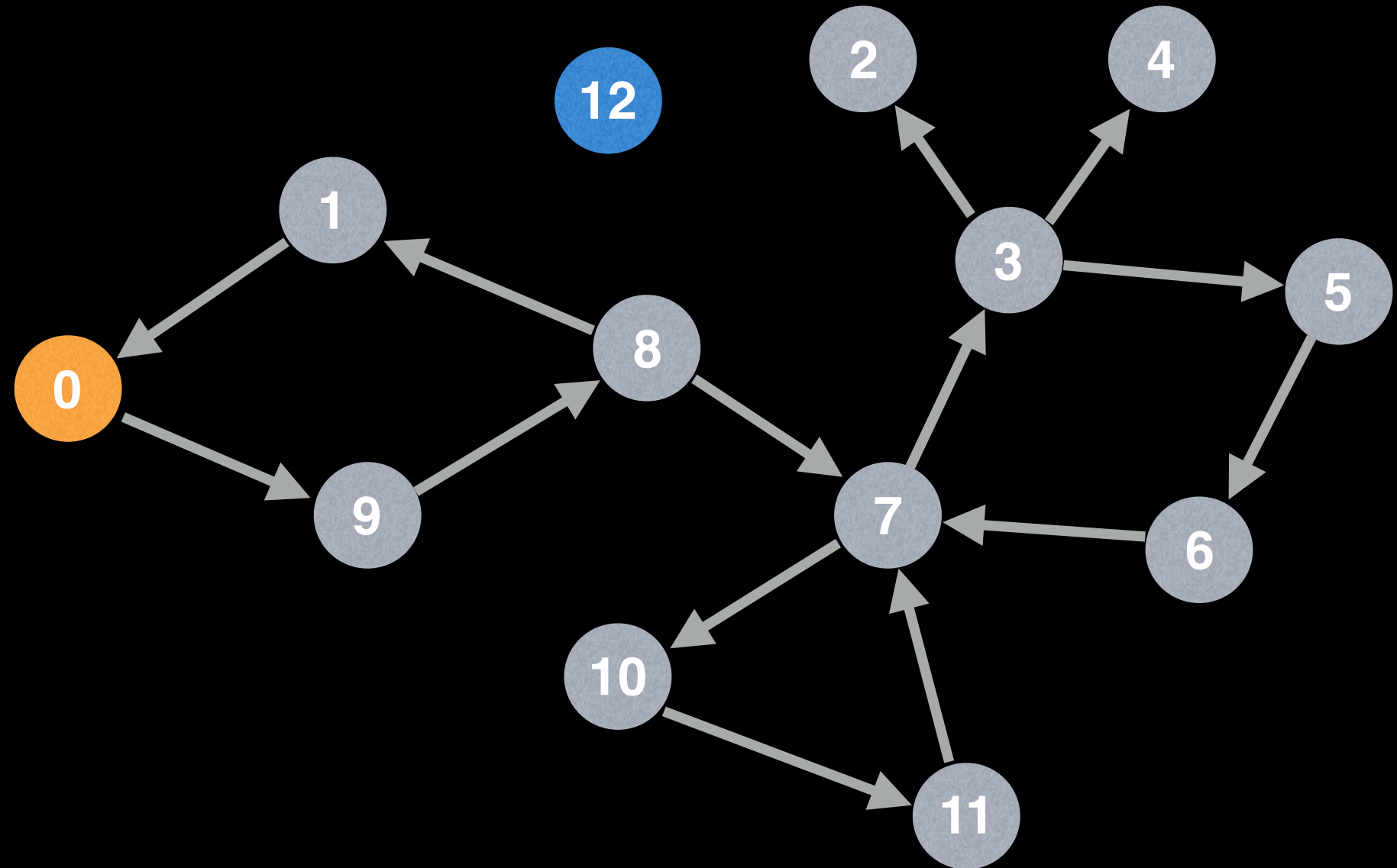
# Basic DFS



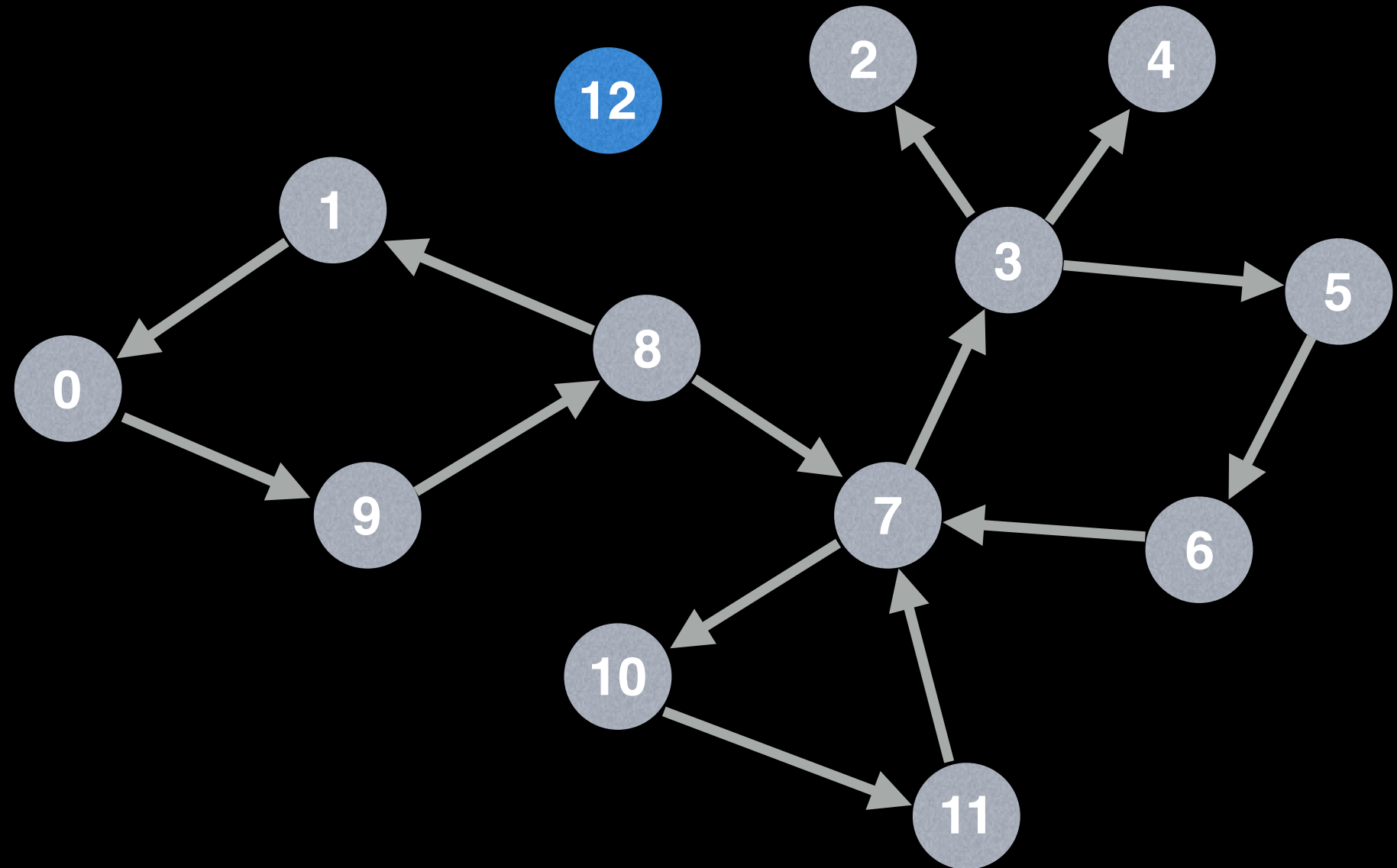
# Basic DFS



# Basic DFS



# Basic DFS



```
# Global or class scope variables
n = number of nodes in the graph
g = adjacency list representing graph
visited = [false, ..., false] # size n
```

```
function dfs(at):
    if visited[at]: return
    visited[at] = true

    neighbours = graph[at]
    for next in neighbours:
        dfs(next)
```

```
# Start DFS at node zero
start_node = 0
dfs(start_node)
```



```
# Global or class scope variables
```

```
n = number of nodes in the graph  
g = adjacency list representing graph  
visited = [false, ..., false] # size n
```

```
function dfs(at):  
    if visited[at]: return  
    visited[at] = true  
  
    neighbours = graph[at]  
    for next in neighbours:  
        dfs(next)
```

```
# Start DFS at node zero  
start_node = 0  
dfs(start_node)
```

```
# Global or class scope variables  
n = number of nodes in the graph  
g = adjacency list representing graph  
visited = [false, ..., false] # size n
```

```
function dfs(at):  
    if visited[at]: return  
    visited[at] = true  
  
    neighbours = graph[at]  
    for next in neighbours:  
        dfs(next)
```

```
# Start DFS at node zero
```

```
start_node = 0  
dfs(start_node)
```

```
# Global or class scope variables
n = number of nodes in the graph
g = adjacency list representing graph
visited = [false, ..., false] # size n
```

```
function dfs(at):
    if visited[at]: return
    visited[at] = true

    neighbours = graph[at]
    for next in neighbours:
        dfs(next)
```

```
# Start DFS at node zero
start_node = 0
dfs(start_node)
```

```
# Global or class scope variables
n = number of nodes in the graph
g = adjacency list representing graph
visited = [false, ..., false] # size n
```

```
function dfs(at):
    if visited[at]: return
    visited[at] = true

    neighbours = graph[at]
    for next in neighbours:
        dfs(next)
```

```
# Start DFS at node zero
start_node = 0
dfs(start_node)
```

```
# Global or class scope variables
n = number of nodes in the graph
g = adjacency list representing graph
visited = [false, ..., false] # size n
```

```
function dfs(at):
    if visited[at]: return
    visited[at] = true

    neighbours = graph[at]
    for next in neighbours:
        dfs(next)
```

```
# Start DFS at node zero
start_node = 0
dfs(start_node)
```

```
# Global or class scope variables
n = number of nodes in the graph
g = adjacency list representing graph
visited = [false, ..., false] # size n
```

```
function dfs(at):
    if visited[at]: return
    visited[at] = true
```

```
    neighbours = graph[at]
    for next in neighbours:
        dfs(next)
```

```
# Start DFS at node zero
start_node = 0
dfs(start_node)
```

```
# Global or class scope variables
n = number of nodes in the graph
g = adjacency list representing graph
visited = [false, ..., false] # size n
```

```
function dfs(at):
    if visited[at]: return
    visited[at] = true

    neighbours = graph[at]
    for next in neighbours:
        dfs(next)
```

```
# Start DFS at node zero
start_node = 0
dfs(start_node)
```

# What else can DFS do?

We can augment the DFS algorithm to:

- Compute a graph's minimum spanning tree.
- Detect and find cycles in a graph.
- Check if a graph is bipartite.
- Find strongly connected components.
- Topologically sort the nodes of a graph.
- Find bridges and articulation points.
- Find augmenting paths in a flow network.
- Generate mazes.