# Graphs

Chapter 22 of Cormen's book

Giulia Bernardini
*giulia.bernardini@units.it*

Algorithmic Design a.y. 2024/2025

# BFS: Pseudocode

BFS($G$,$s$) - $G$ is represented by the adjacency lists $\mathrm{Adj}[\cdot]$ of its vertices

**for each** $u \in V \setminus \{s\}$
    $u.color \leftarrow \mathrm{white}$;
    $u.distance \leftarrow \infty$;

Initialisation

$s.color \leftarrow \mathrm{gray}$;
$s.distance \leftarrow 0$;
$Q \leftarrow \emptyset$;
enqueue($Q$,$s$);

Visit of the source

**while** $Q \neq \emptyset$
    $u \leftarrow$ dequeue($Q$);
    **for each** $v \in \mathrm{Adj}[u]$
        **if** $v.color = \mathrm{white}$
            $v.color \leftarrow \mathrm{gray}$;
            $v.distance \leftarrow u.distance + 1$;
            enqueue($Q$,$v$);
    $u.color \leftarrow \mathrm{black}$;

Visit of the other vertices

# BFS: Complexity

BFS($G$,$s$) - $G$ is represented by the adjacency lists $\mathrm{Adj}[\cdot]$ of its vertices

**for each** $u \in V \setminus \{s\}$
  $u.color \leftarrow$ white;
  $u.distance \leftarrow \infty$;
$s.color \leftarrow$ gray;
$s.distance \leftarrow 0$;
$Q \leftarrow \emptyset$;
enqueue($Q$,$s$);
**while** $Q \neq \emptyset$
  $u \leftarrow$ dequeue($Q$);
  **for each** $v \in \mathrm{Adj}[u]$
    **if** $v.color$ = white
      $v.color \leftarrow$ gray;
      $v.distance \leftarrow u.distance + 1$;
      enqueue($Q$,$v$);
  $u.color \leftarrow$ black;

Initialisation: $O(|V|)$

Visit of the source: $O(1)$

$O(1)$

$O(|\mathrm{Adj}[u]|)$

Visit of the other vertices: each iteration of the **for** loop enqueues $v \in \mathrm{Adj}[u]$ only if it is white, and it immediately turns its color to gray $\Longrightarrow$ each vertex is inserted in $Q$ at most once.
Cost of the **while** loop:

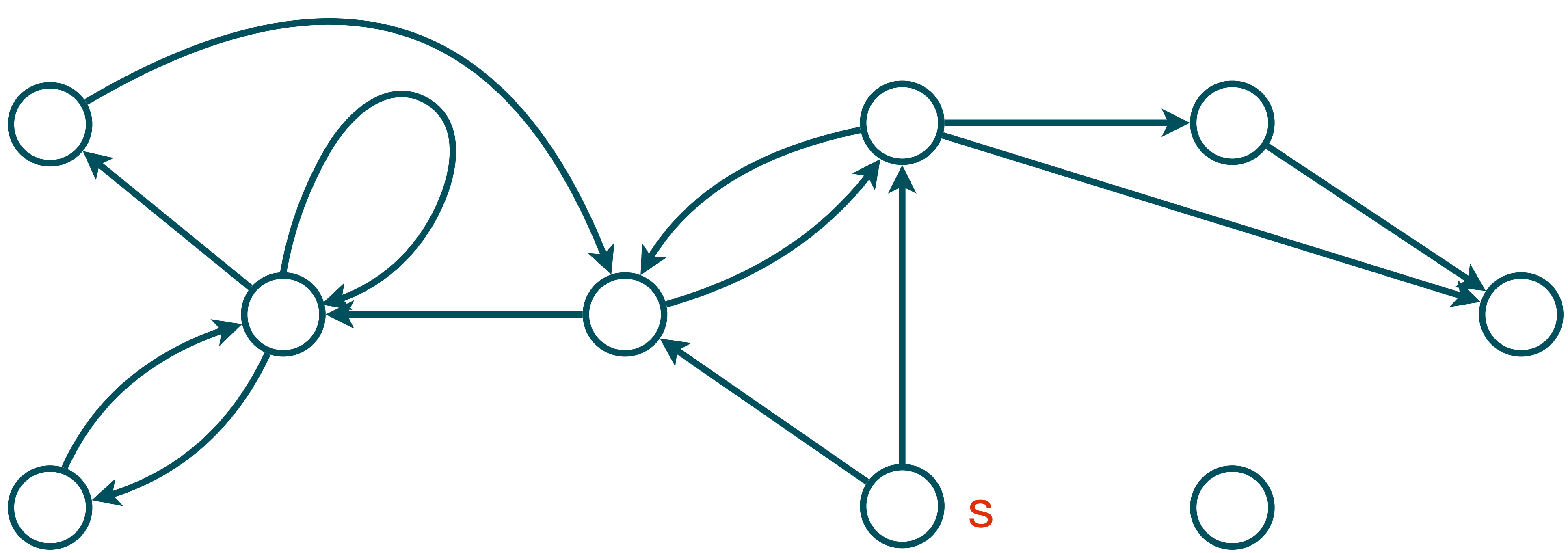
$$O\left(\sum_{u \in V} |\mathrm{Adj}[u]|\right) = O(|E|)$$

# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet;

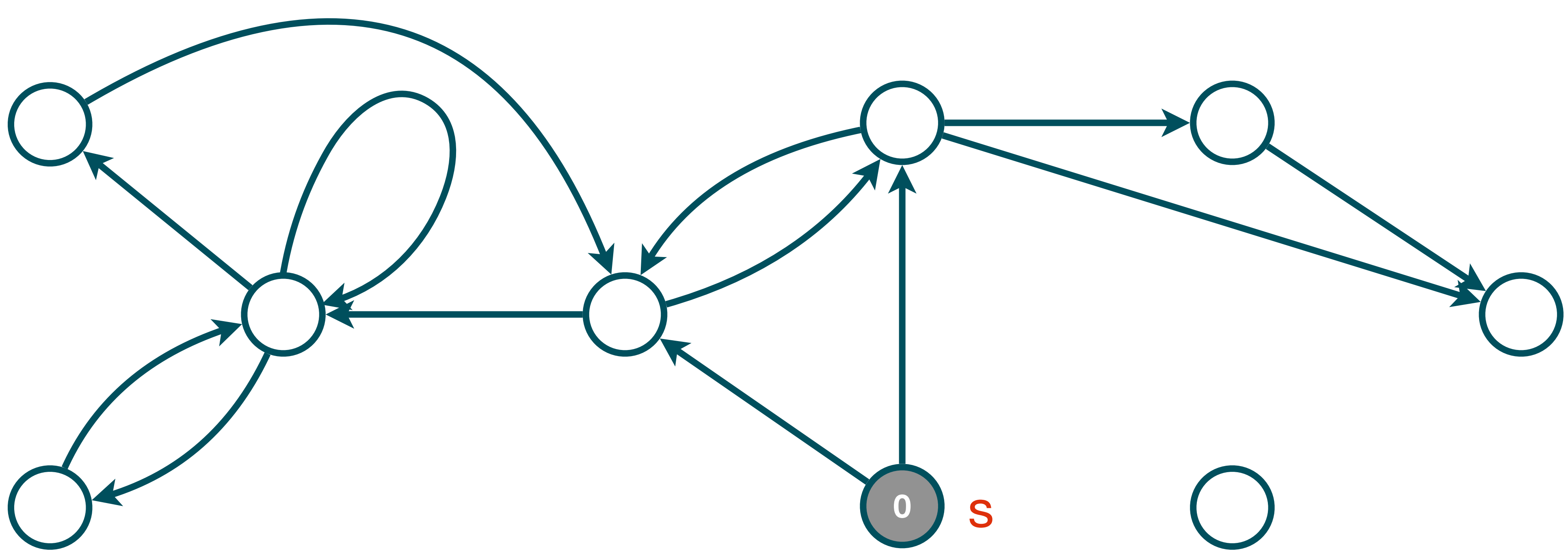


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours;

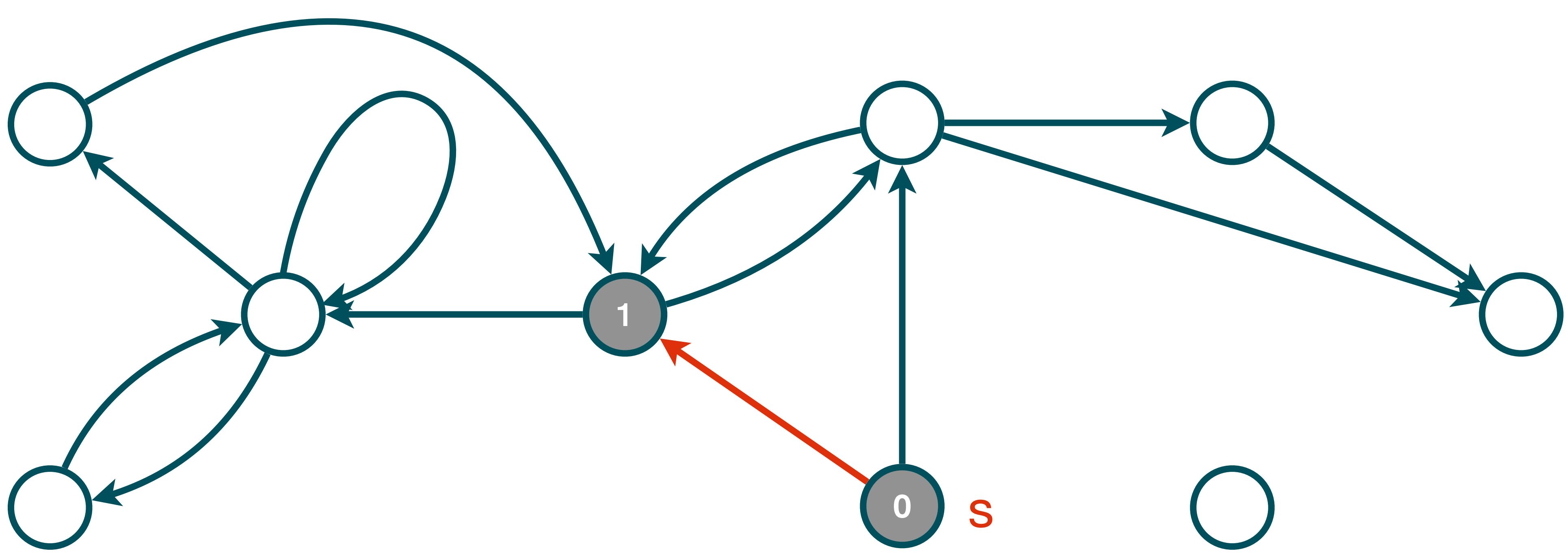


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours;

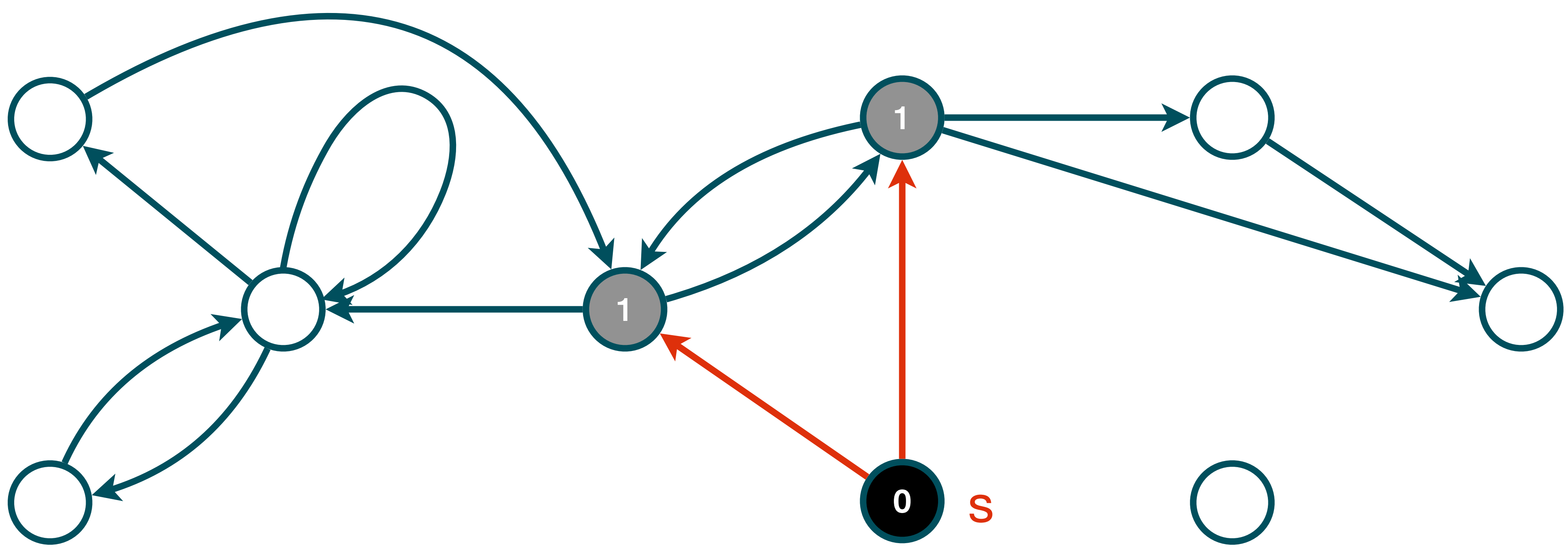


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

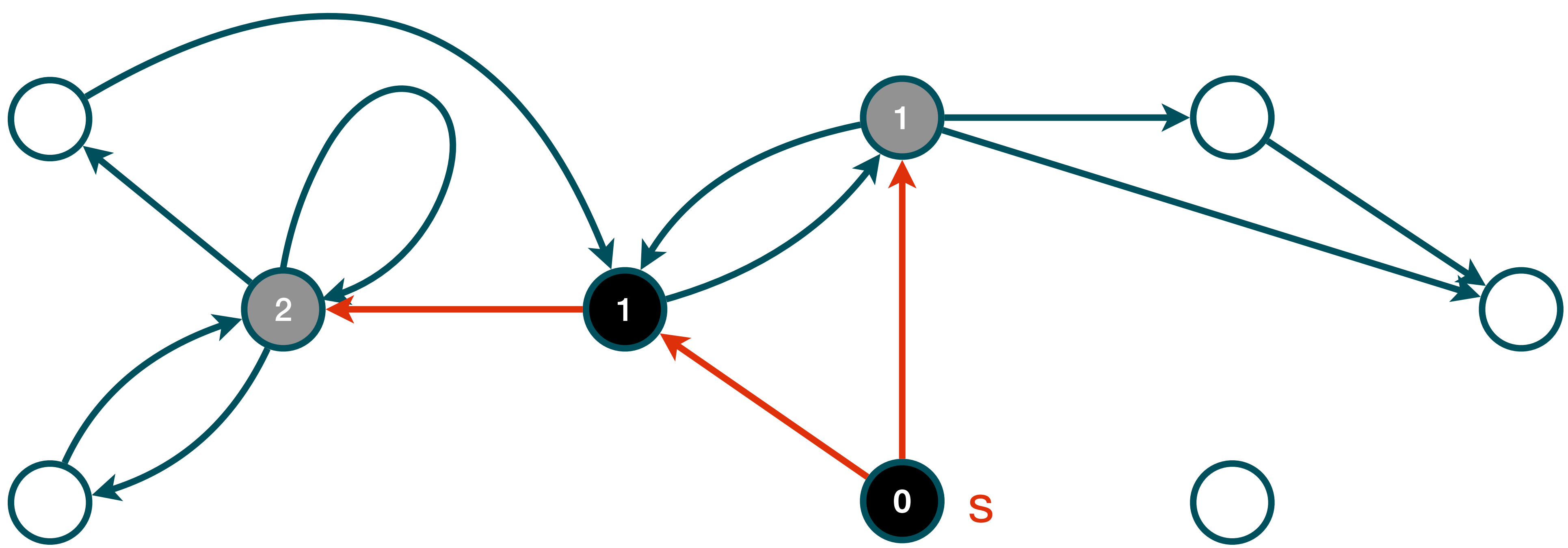


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

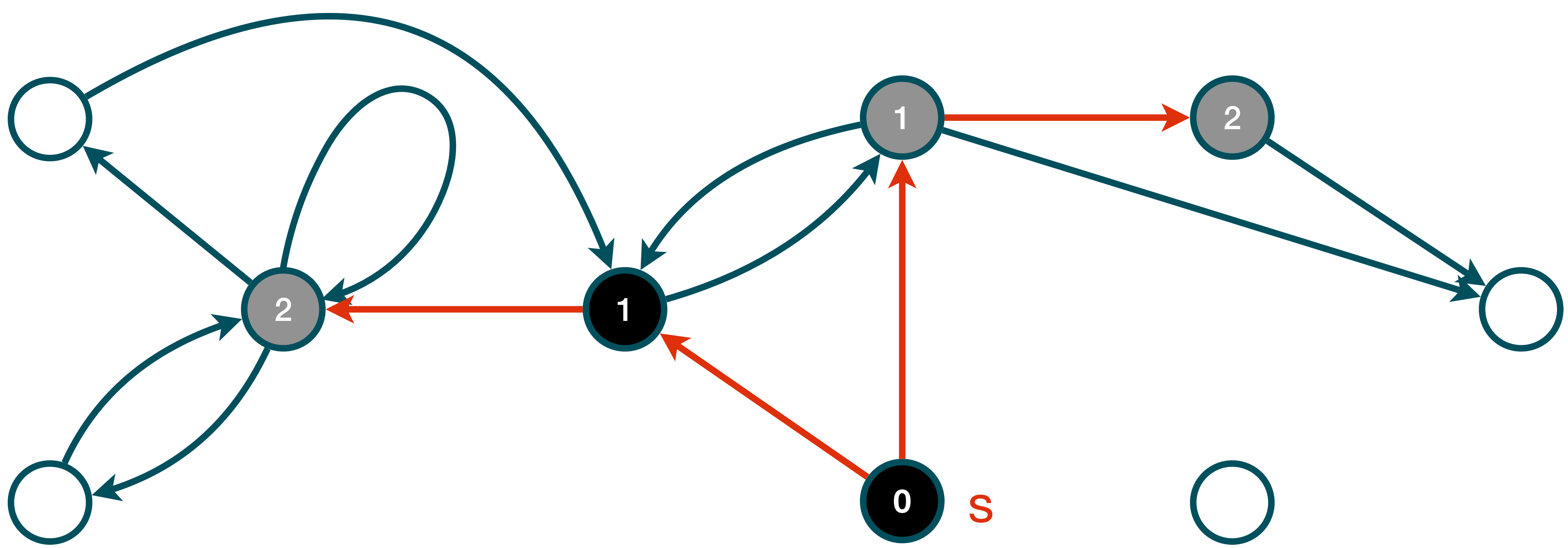


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

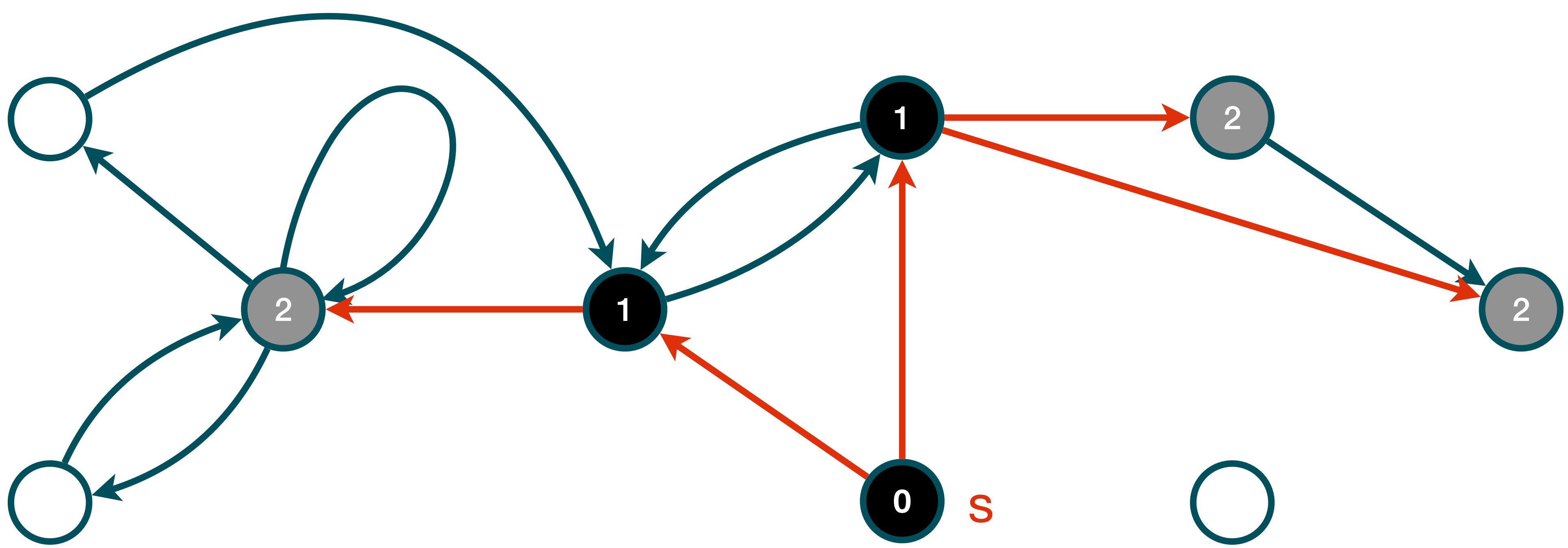


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

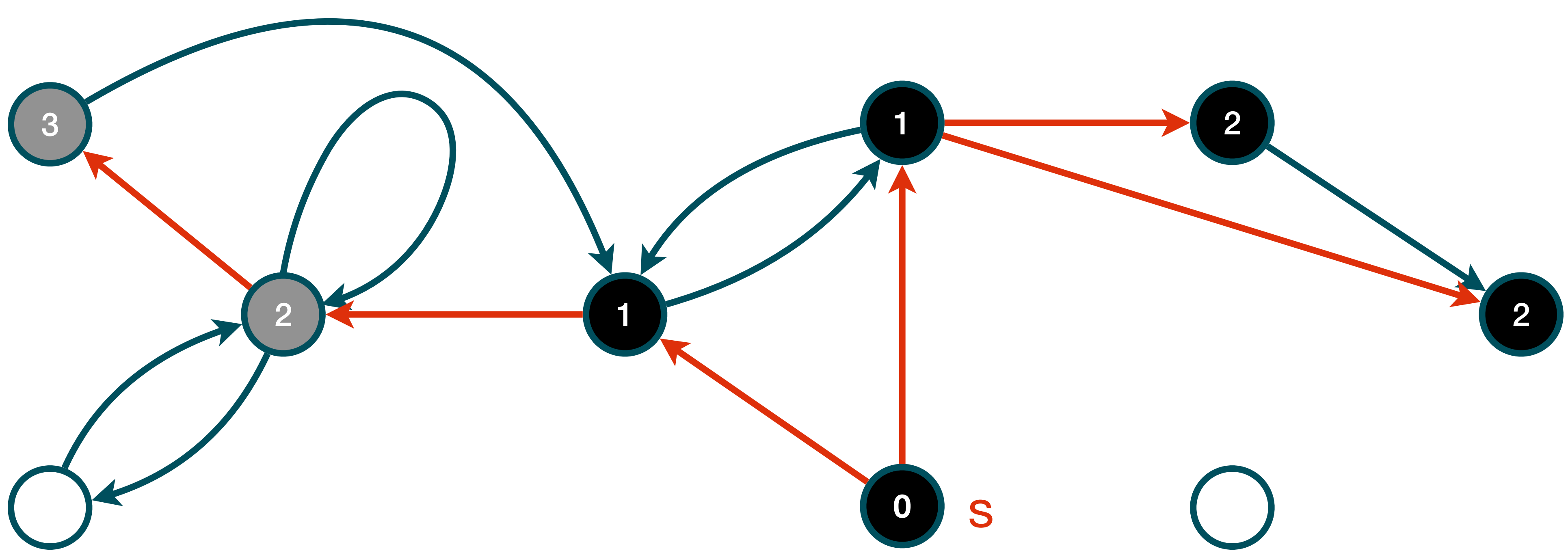


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

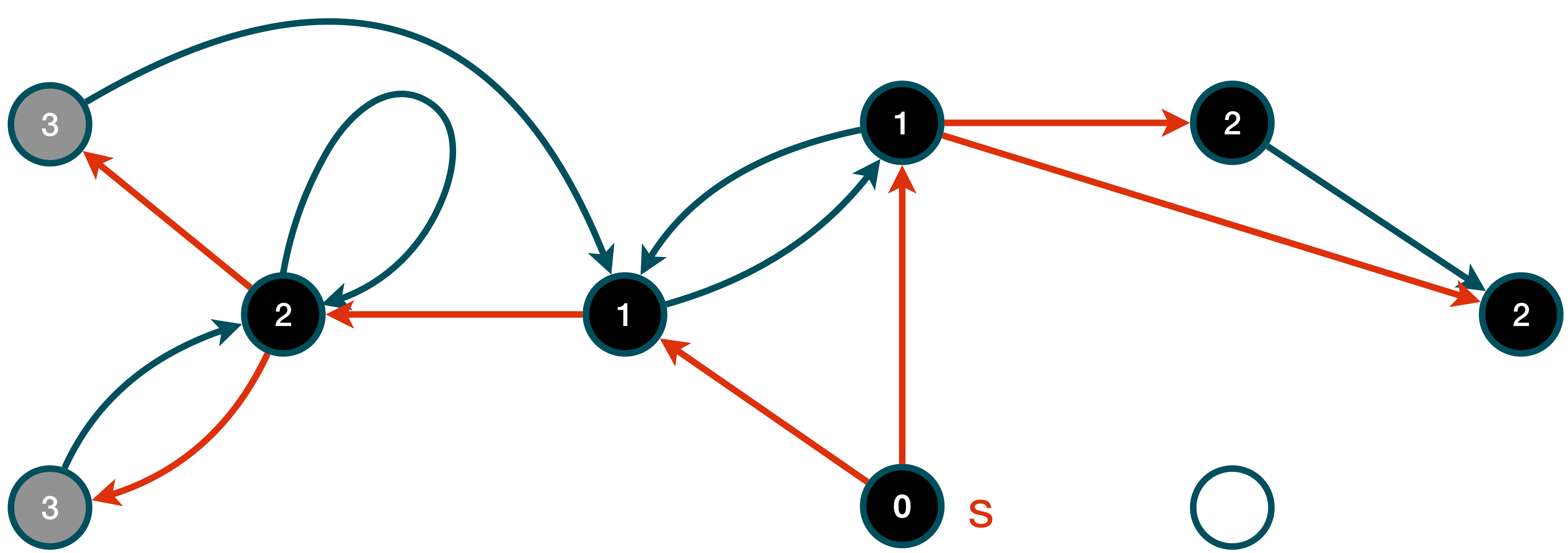


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

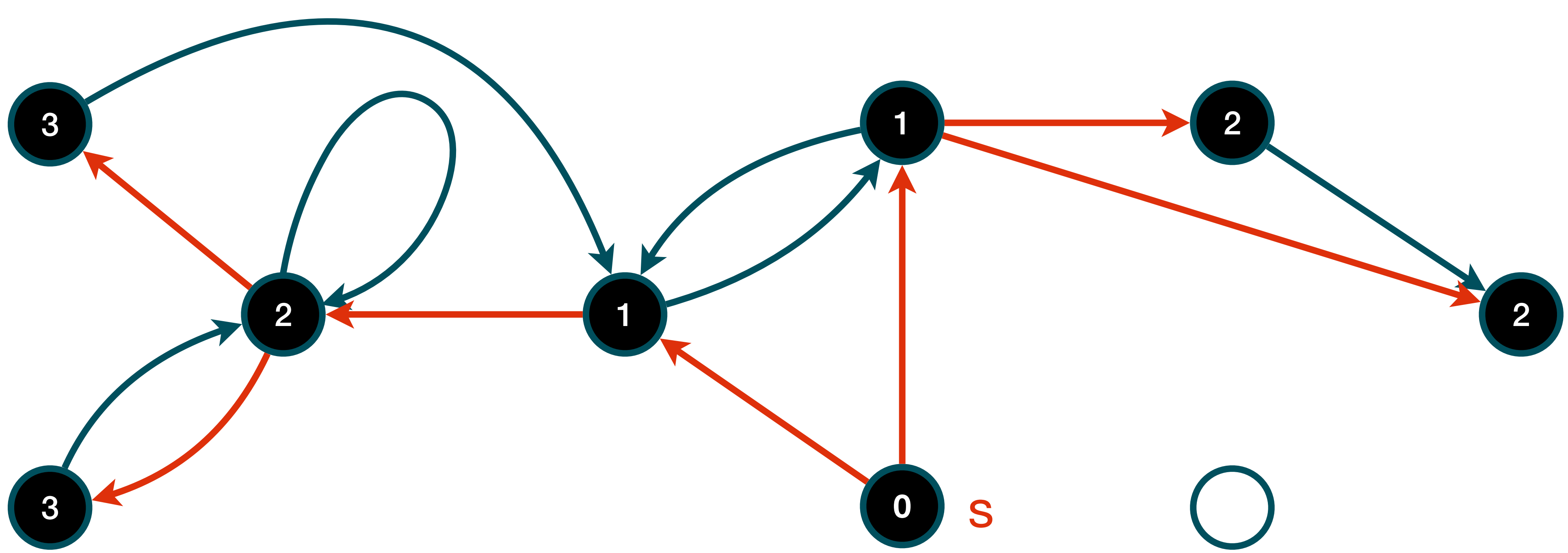


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

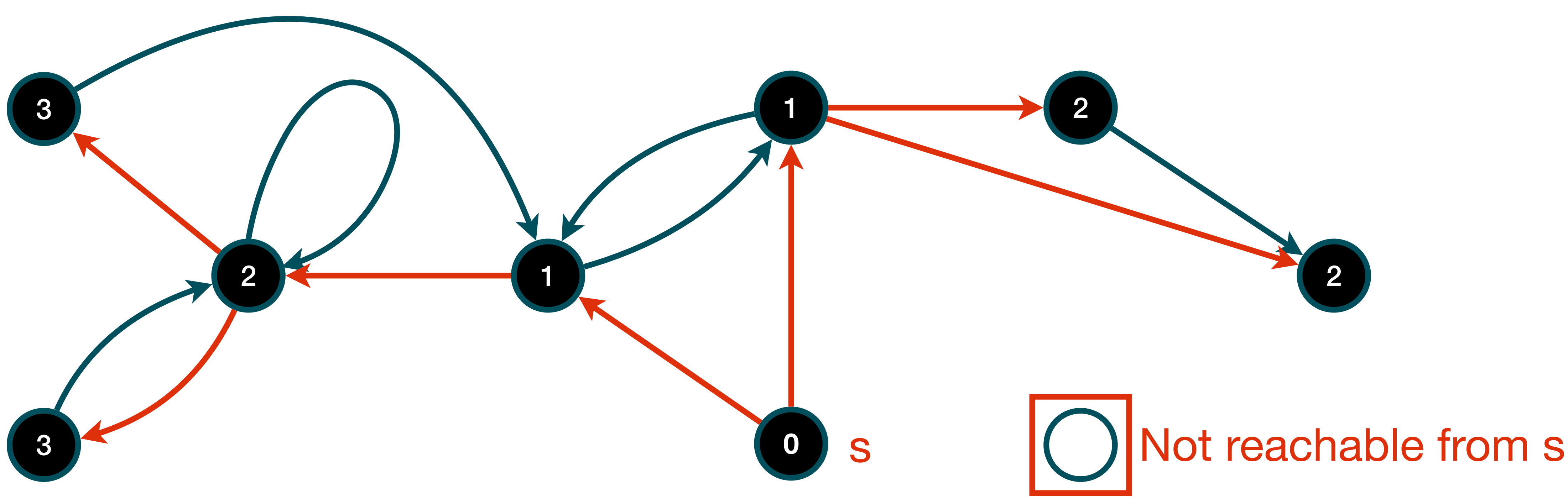


# Breadth-First Search

The visiting order is related to the distance from a source node: the closer a node to the source, the sooner it will be visited

BFS produces a breadth-first tree: the tree consisting of the shortest paths from the source to any reachable node

White nodes have not been discovered yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.



# BFS: Properties

**Lemma 1.** The time complexity of BFS is $O(|V|+|E|)$ (linear in the size of the adjacency-list representation of G)

**Lemma 2.** Let $Q=[v_1,\dots,v_n]$ be the queue at any iteration of BFS. Then $v_i.distance \le v_{i+1}.distance$ and $v_n.distance \le v_1.distance+1$, for all $i=1,\dots,n-1$

Lemma 2 tells us that, at any iteration, if the head node of Q is at distance *d* from *s*, Q only contains nodes at distance *d* or *d*+1 from *s*; possible nodes at distance *d*+2 will be only enqueued after all nodes at distance *d* have been dequeued.

**Lemma 3.** Let $d(v,s)$ be the distance between *v* and *s*, for any $v \in V$. Then:

(i) $v.distance \neq \infty \iff v$ is reachable from *s*

(ii) if $v.distance \neq \infty \implies v.distance = d(v,s)$

# DFS: Pseudocode

DFS(*G*) - *G* is represented by the adjacency lists Adj[·] of its vertices

```
for each u ∈ V
        u.color←white;
t←0;
for each u ∈ V
        if u.color = white
              DFS_visit(G,u)
```

Initialisation

Start the search from a new source

```
DFS_visit(G,u)
    t←t+1;
    u.d←t;
    u.color←gray;
    for each v ∈ Adj[u]
        if v.color = white
            DFS_visit(G,v);
    u.color←black;
    t←t+1;
    u.f←t;
```

Visit the graph recursively

# DFS: Complexity

DFS($G$) - $G$ is represented by the adjacency lists Adj[$\cdot$] of its vertices

```
for each u ∈ V
        u.color←white;
t←0;
for each u ∈ V
        if u.color = white
              DFS_visit(G,u)
```

Initialisation: $O(|V|)$

Start the search from a new source: this only happens when a vertex is white $\Longrightarrow$ $O(|V|)$ calls

```
DFS_visit(G,u)
    t←t+1;
    u.d←t;
    u.color←gray;
    for each v ∈ Adj[u]
         if  v.color = white
              DFS_visit(G,v);
    u.color←black;
    t←t+1;
    u.f←t;
```

Visit the graph recursively: this procedure is only called on white vertices, which are immediately painted gray

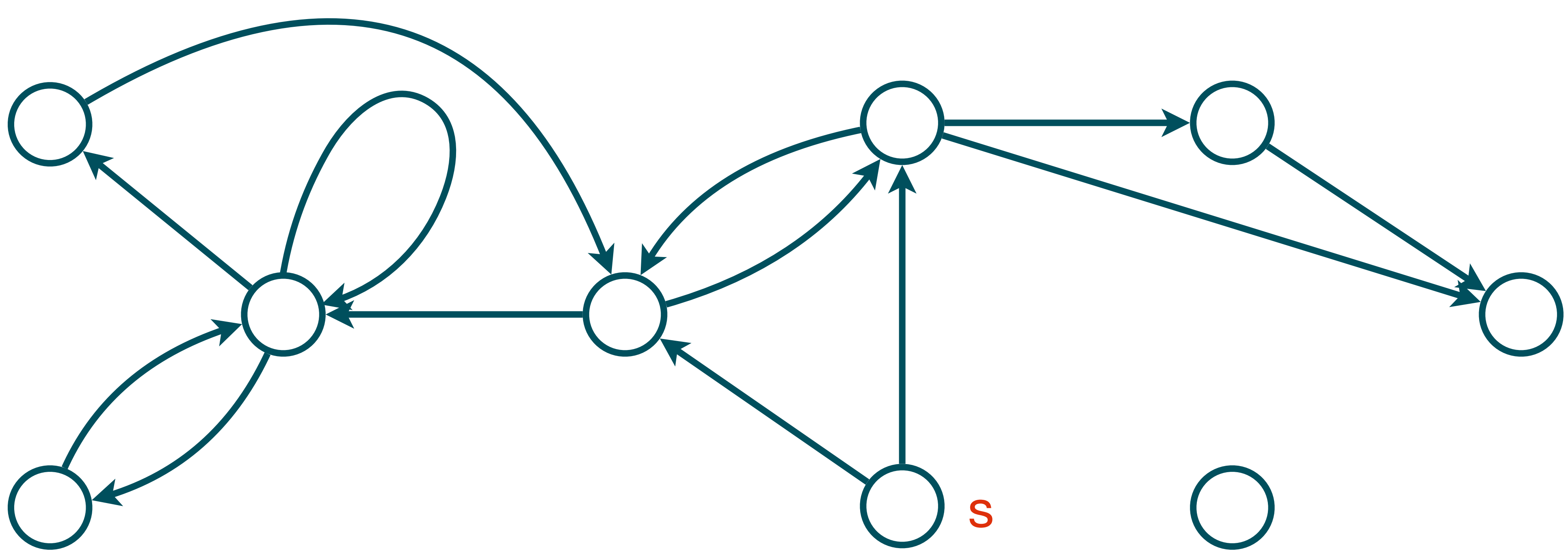
$$\Longrightarrow O\left(\sum_{u\in V} |\mathrm{Adj}[u]|\right) = O(|E|)$$

# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



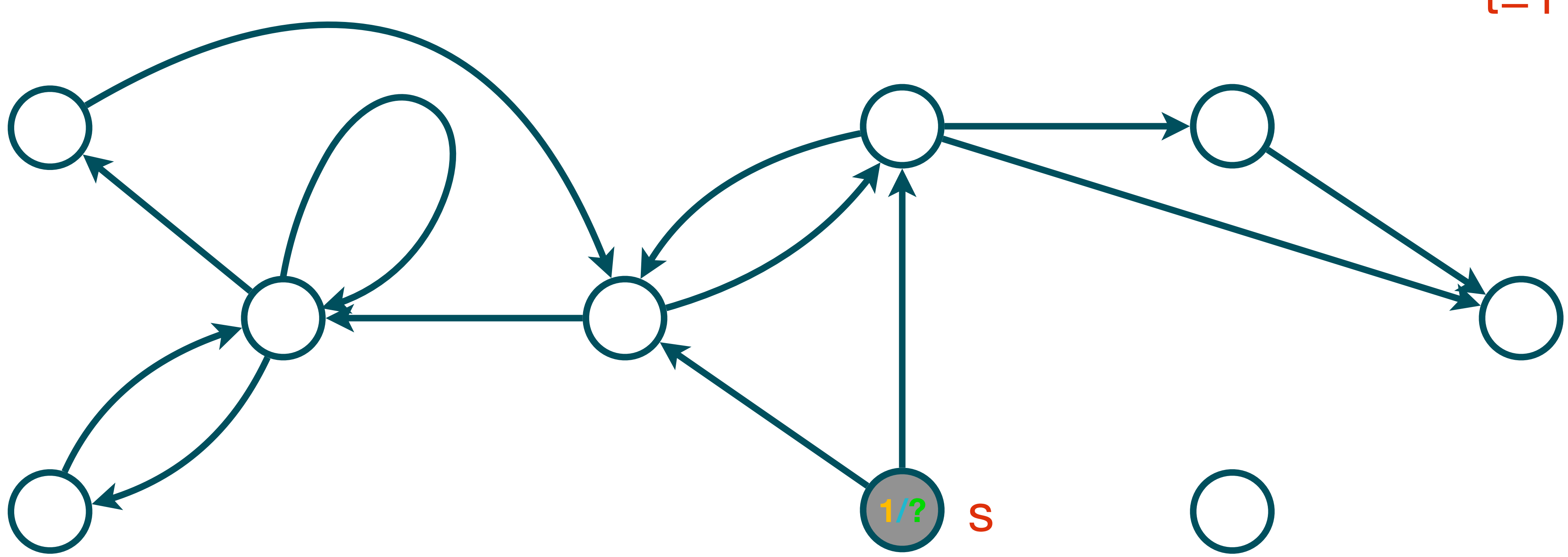
# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

t=1

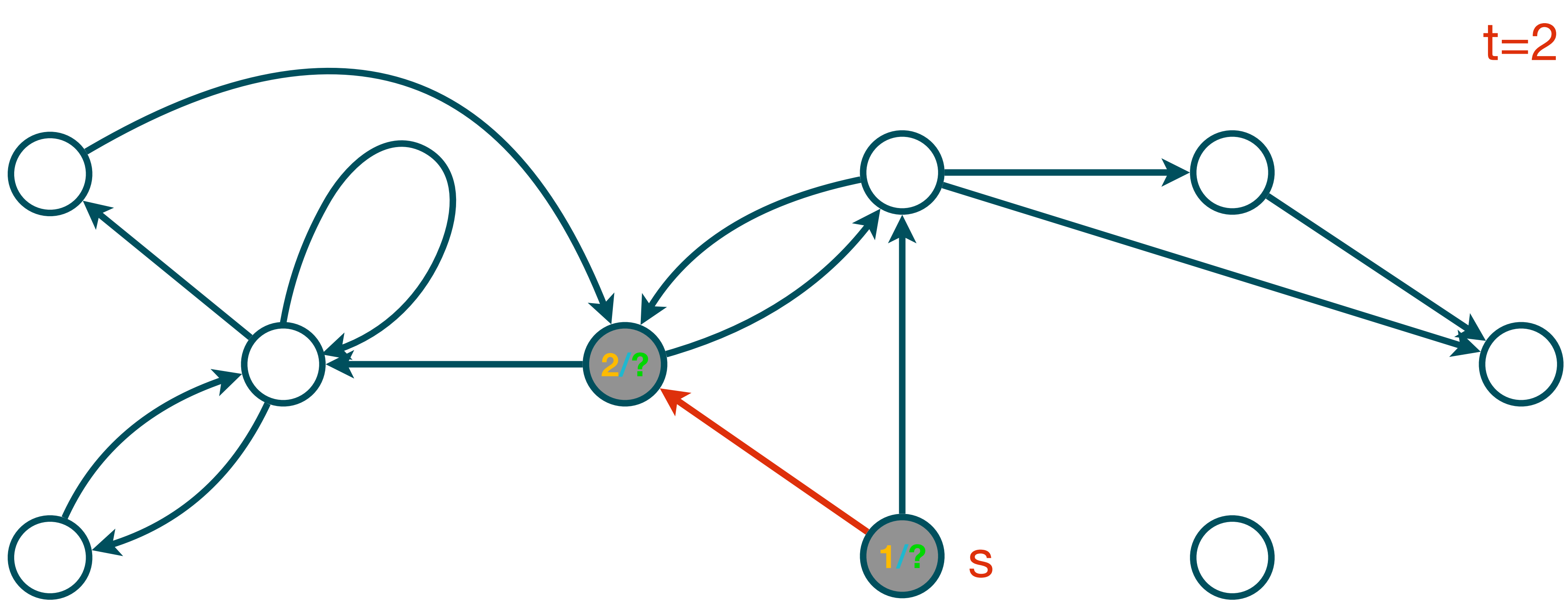


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

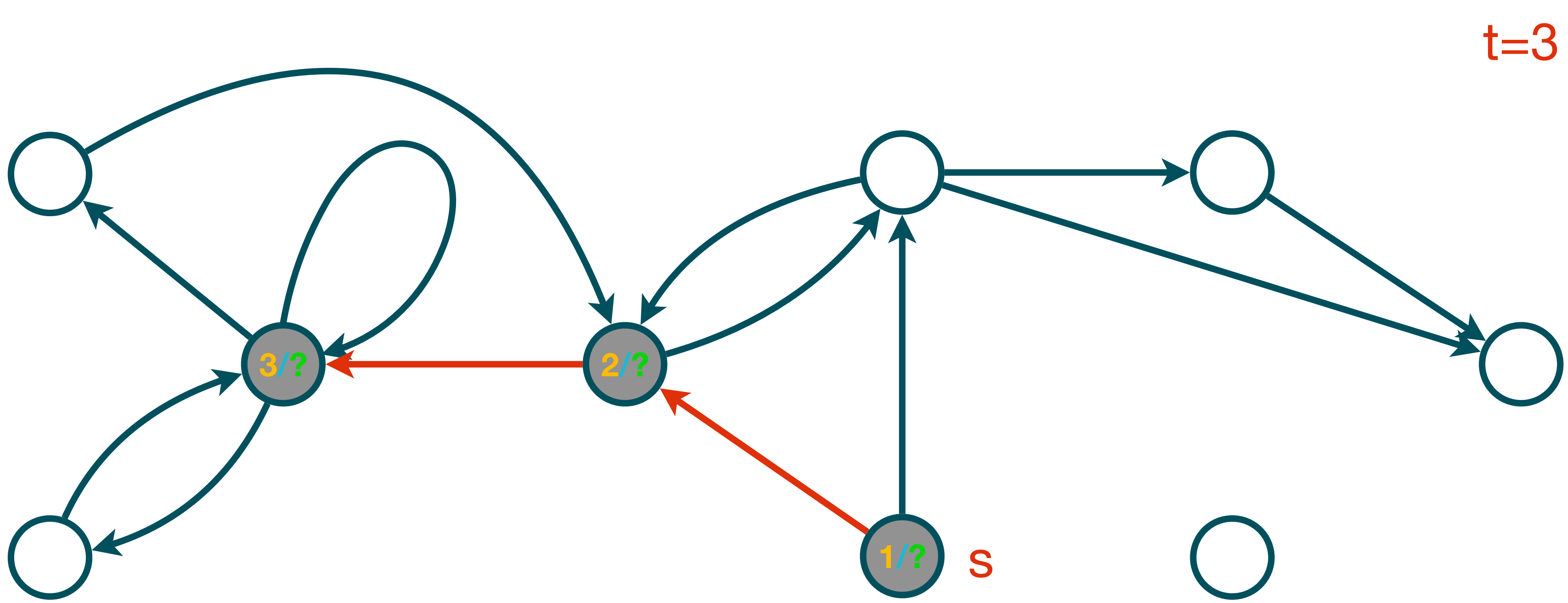


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

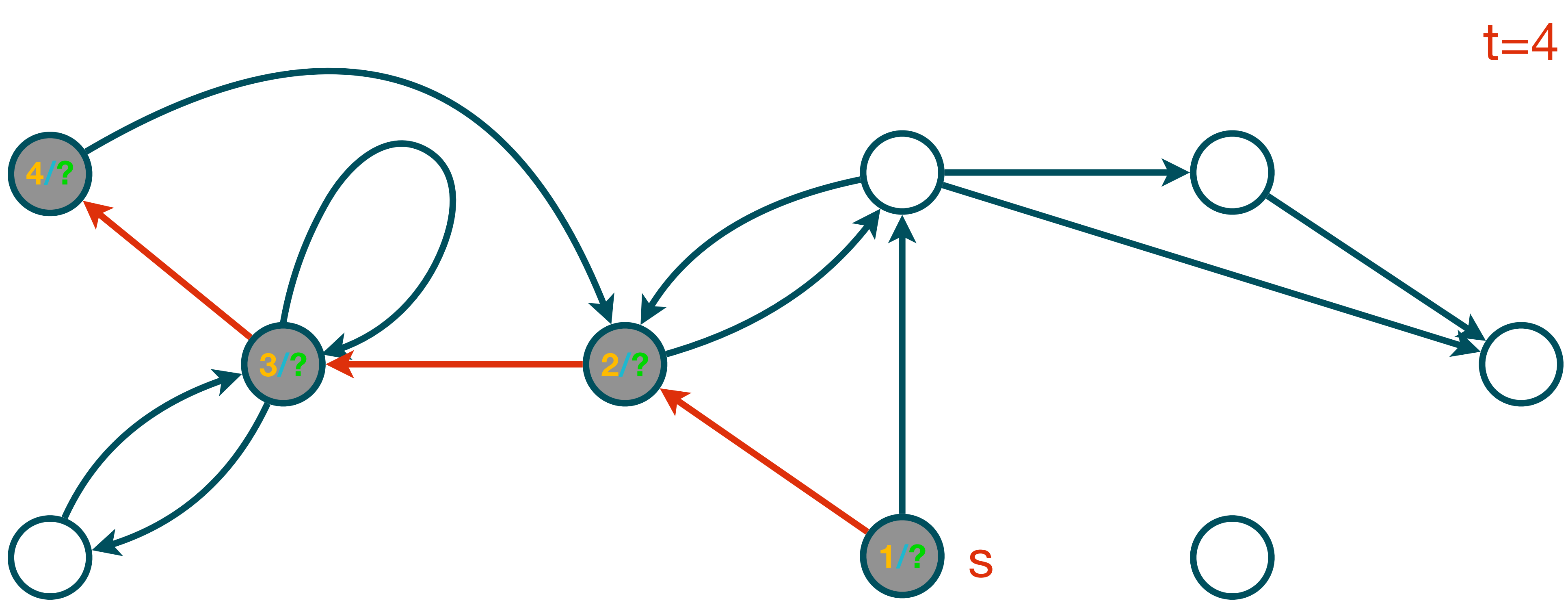


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



# Depth-First Search

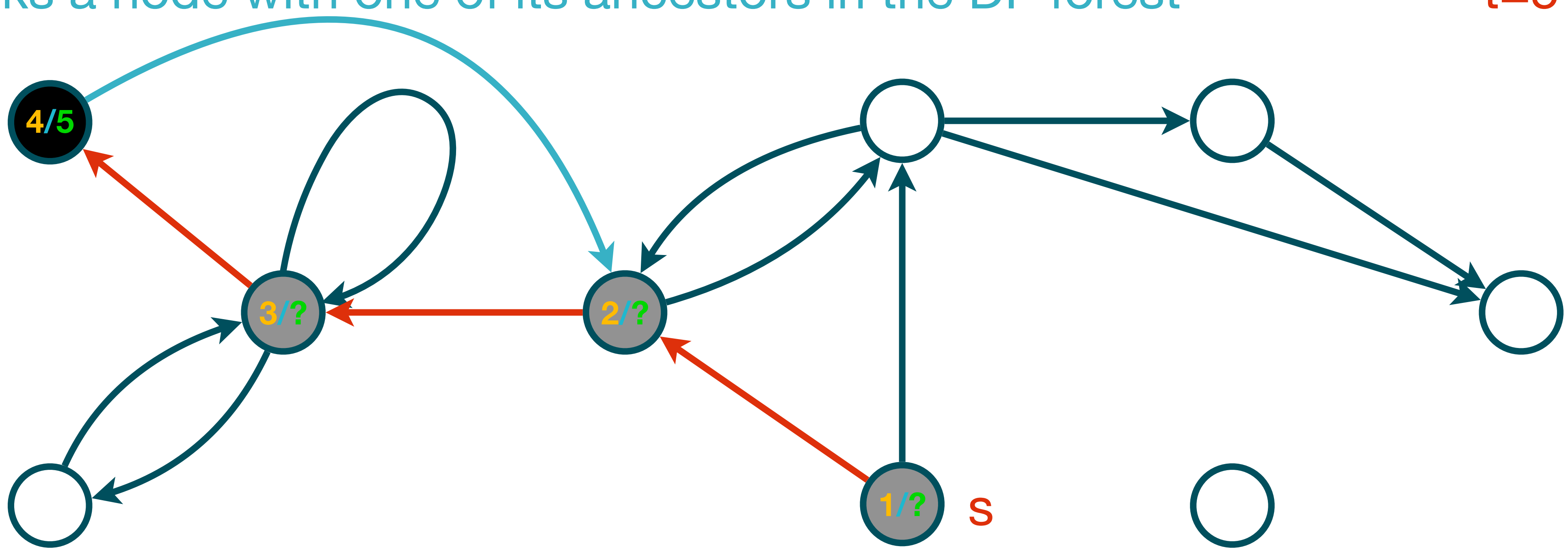
Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

Back edge: links a node with one of its ancestors in the DF forest

t=5

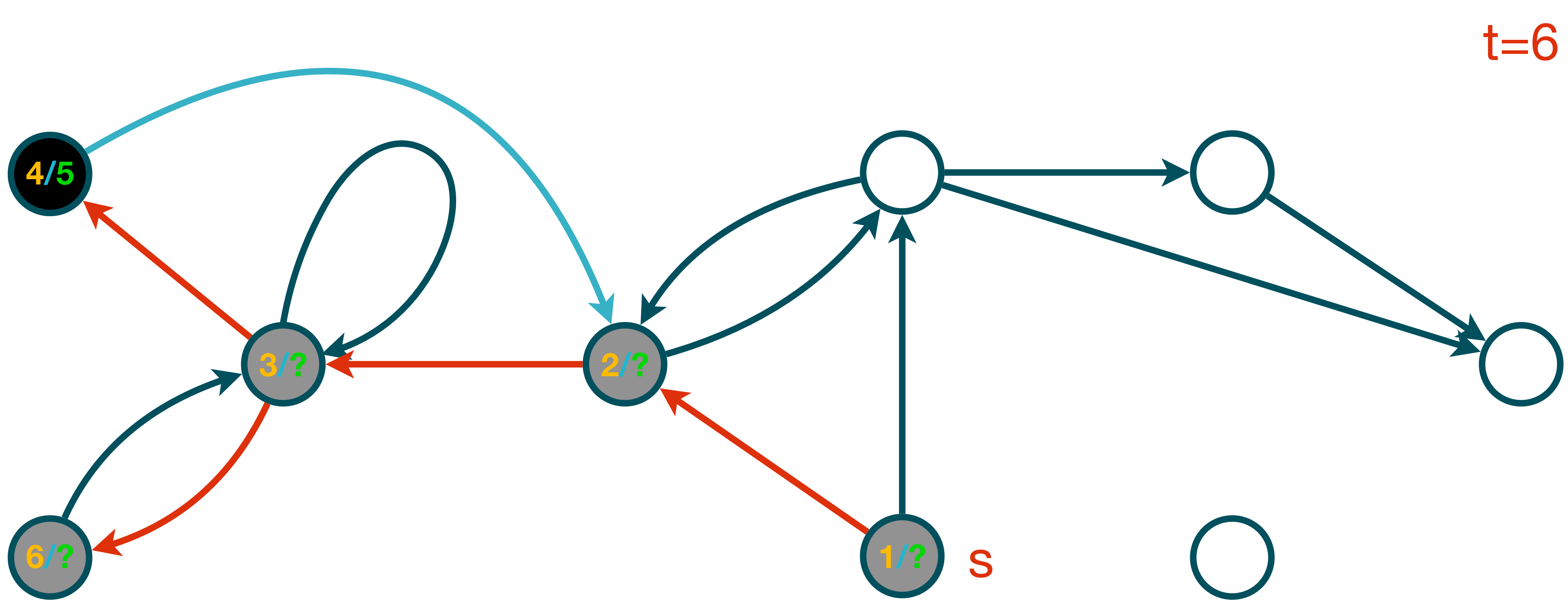


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

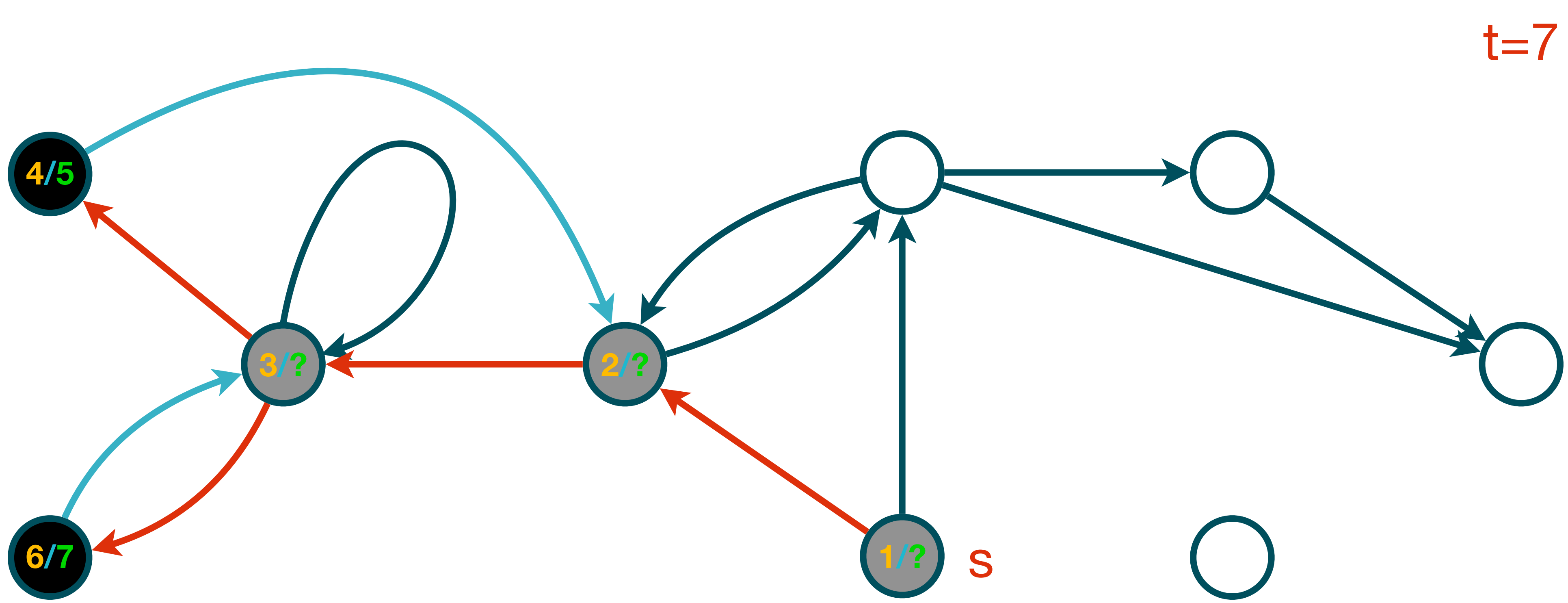


# Depth-First Search

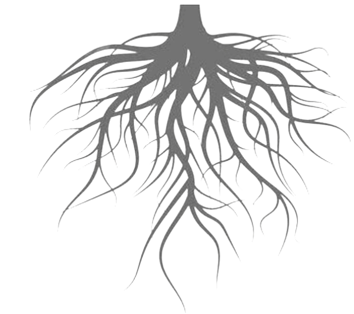
Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



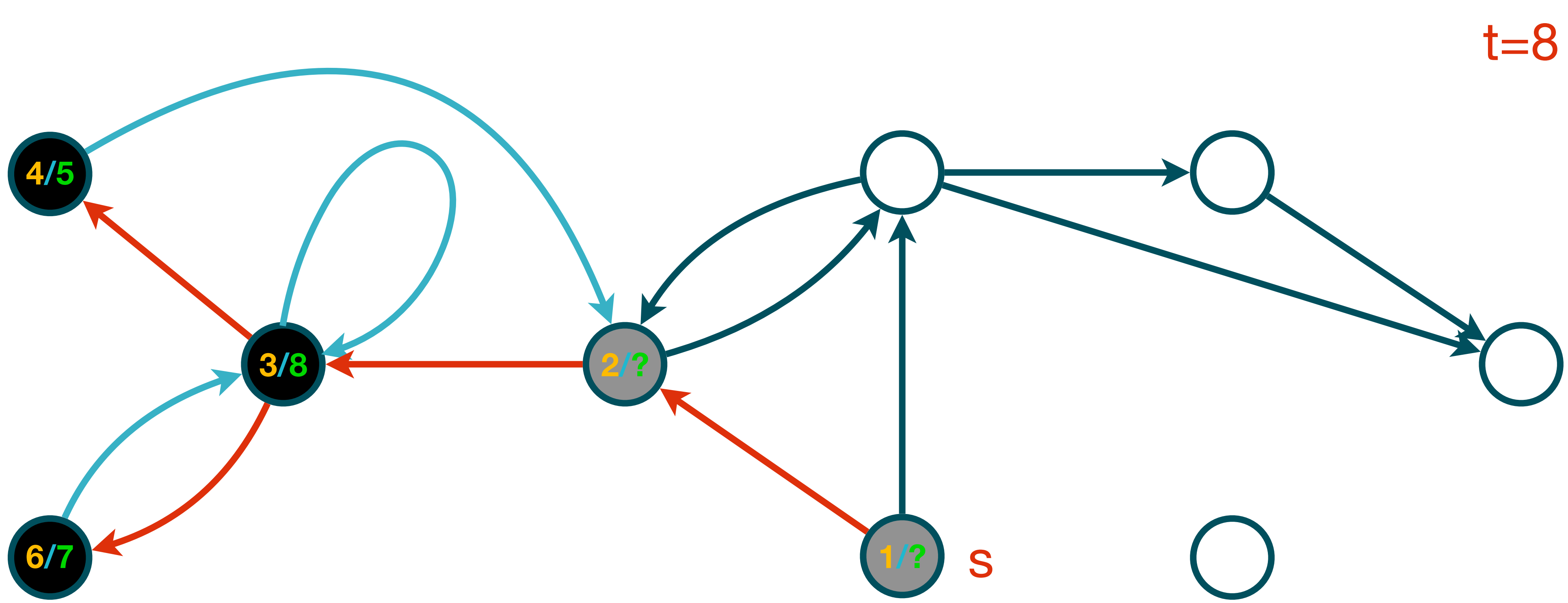
# Depth-First Search



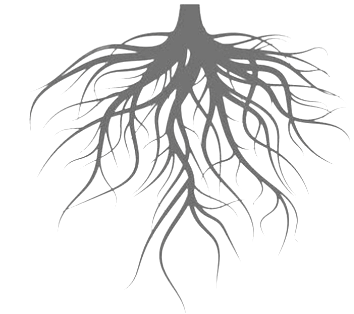
Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



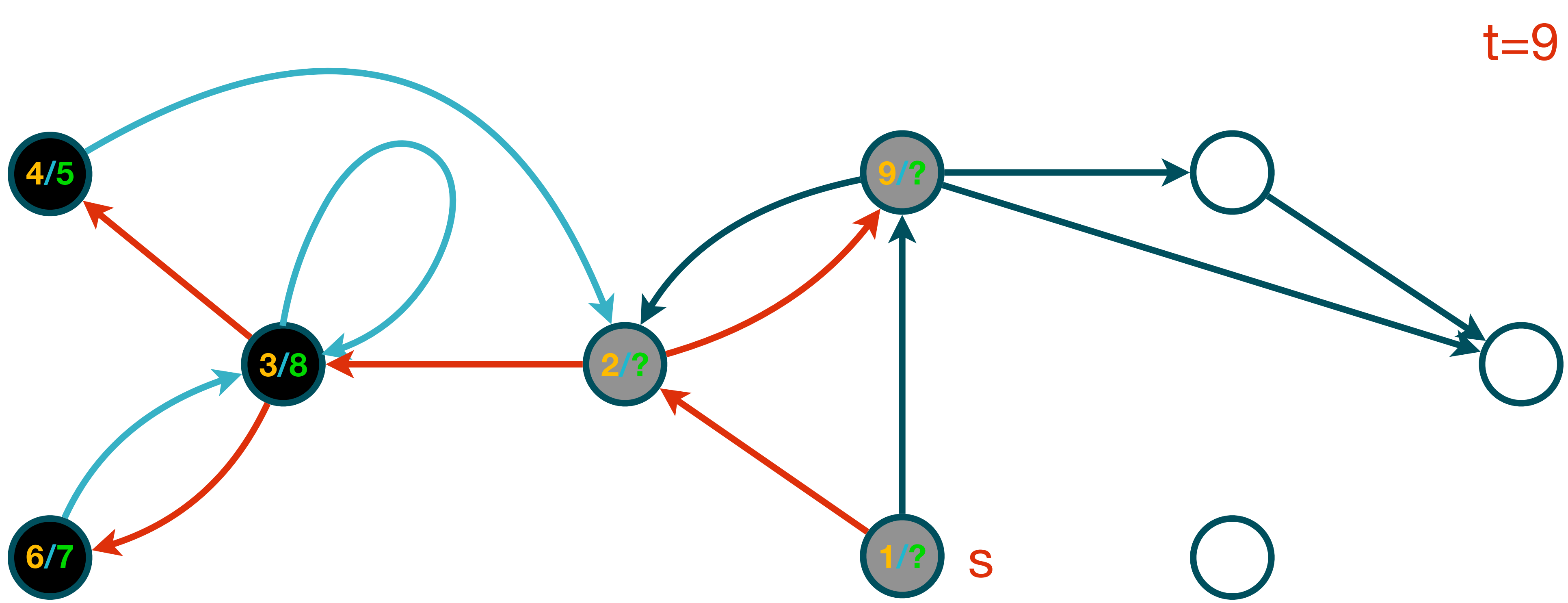
# Depth-First Search



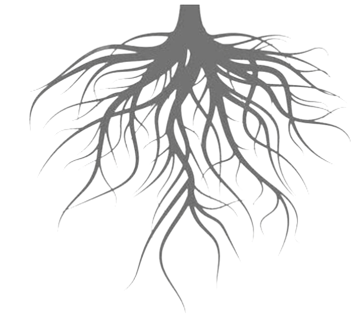
Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



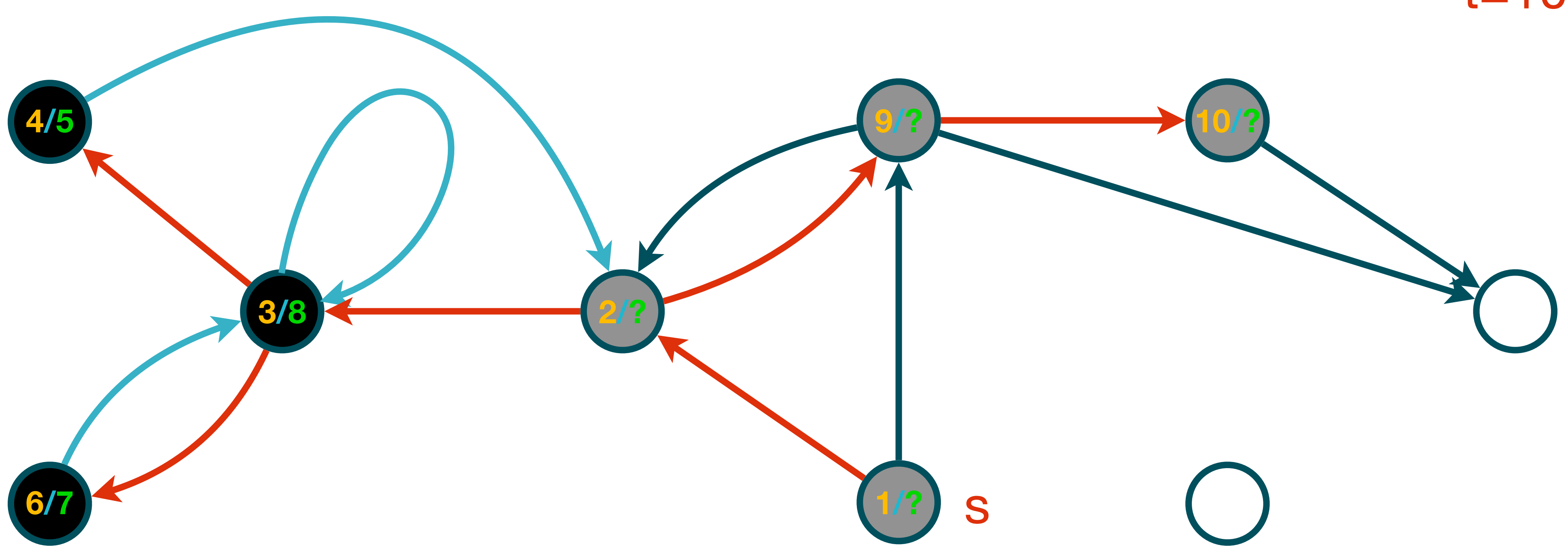
# Depth-First Search



Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

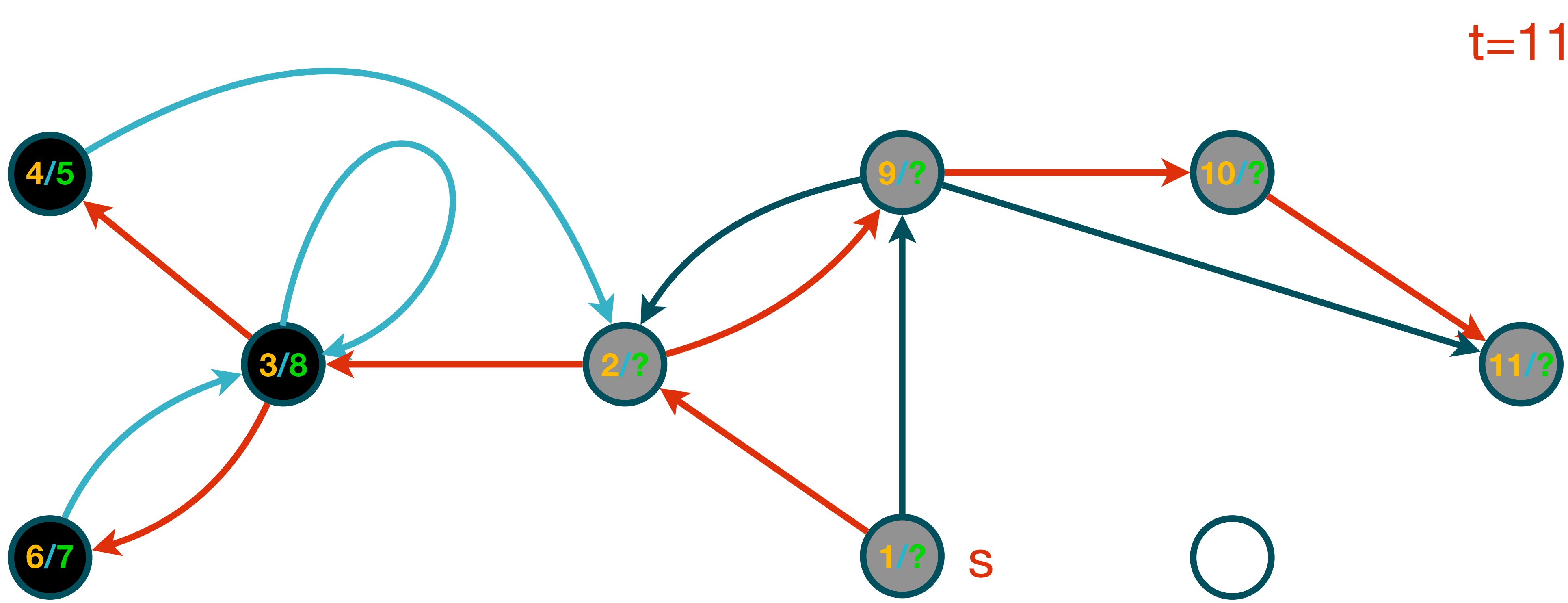


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

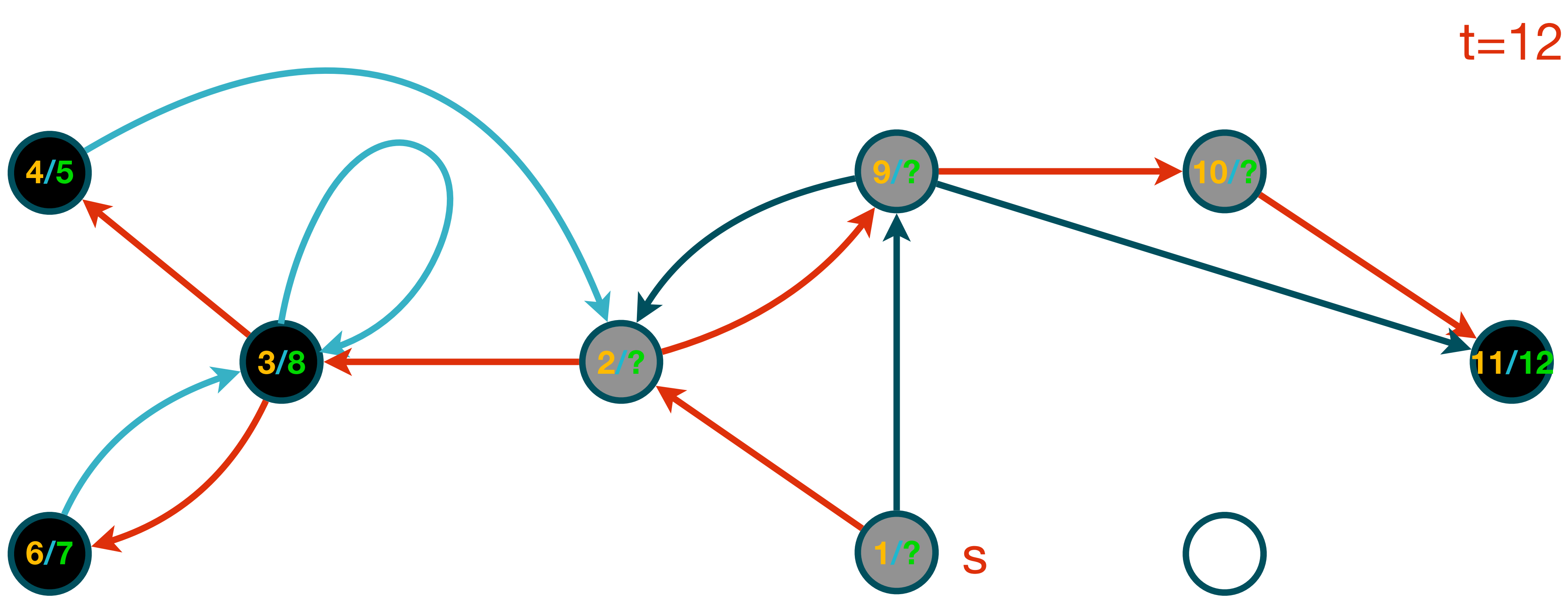


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



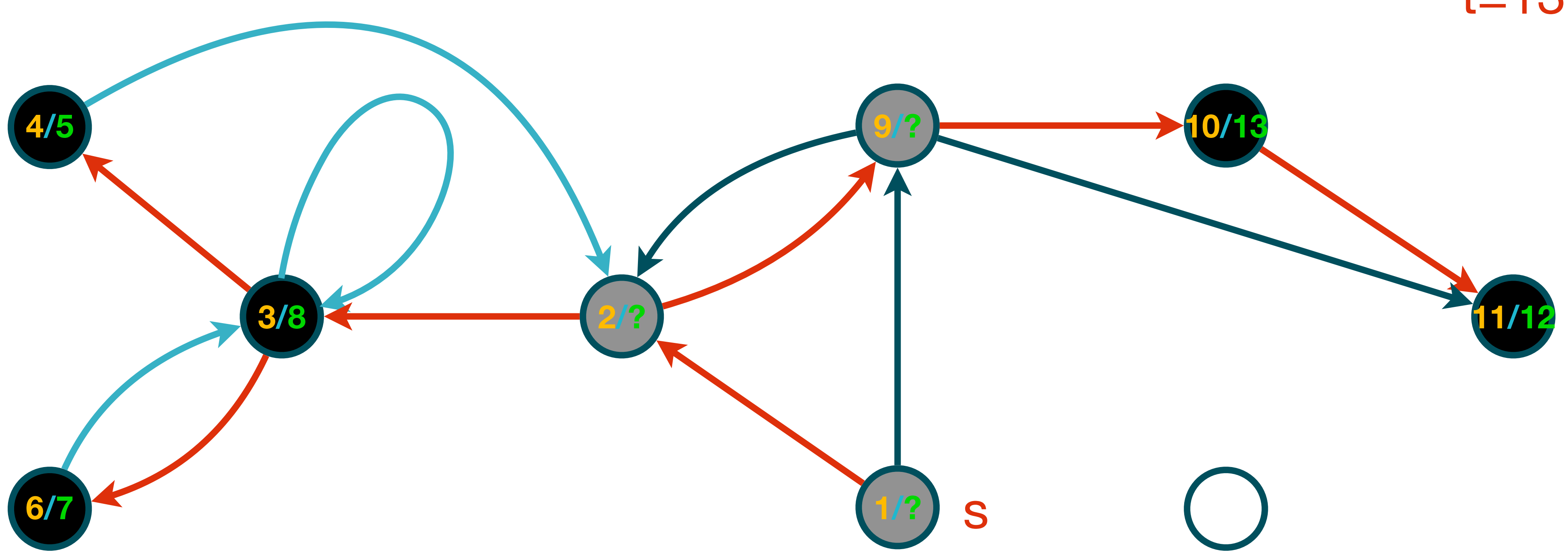
# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

t=13



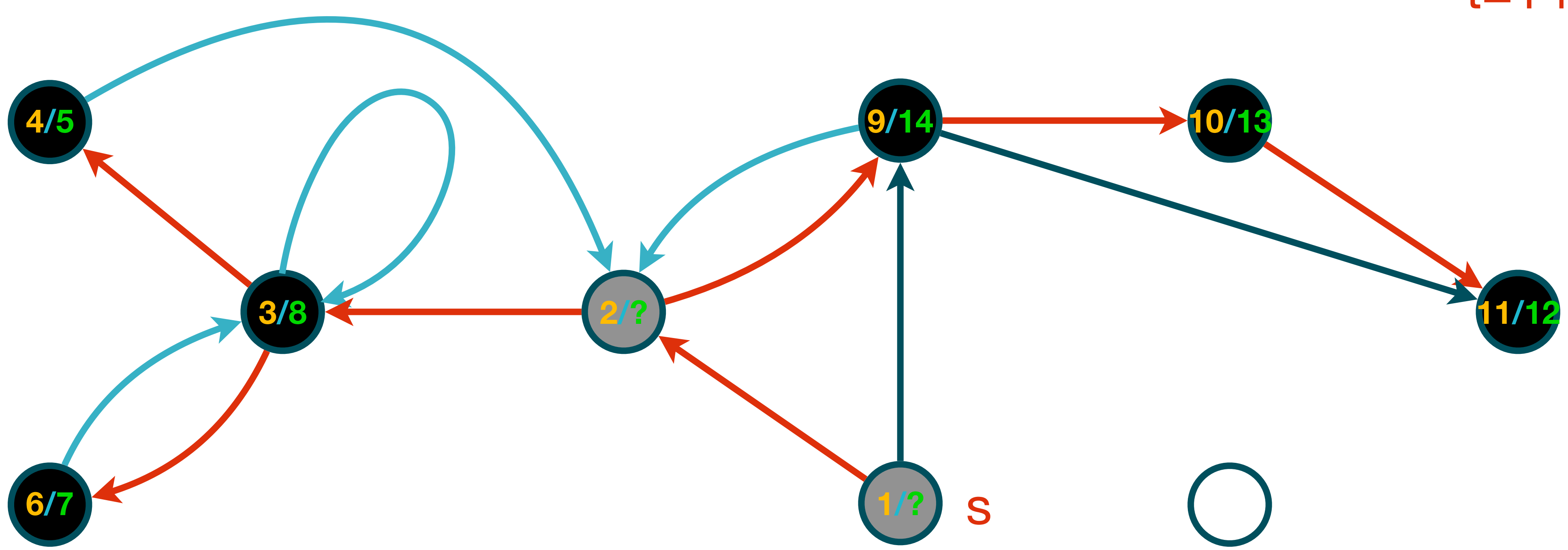
# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

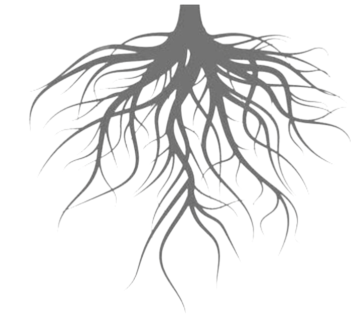
Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

t=14



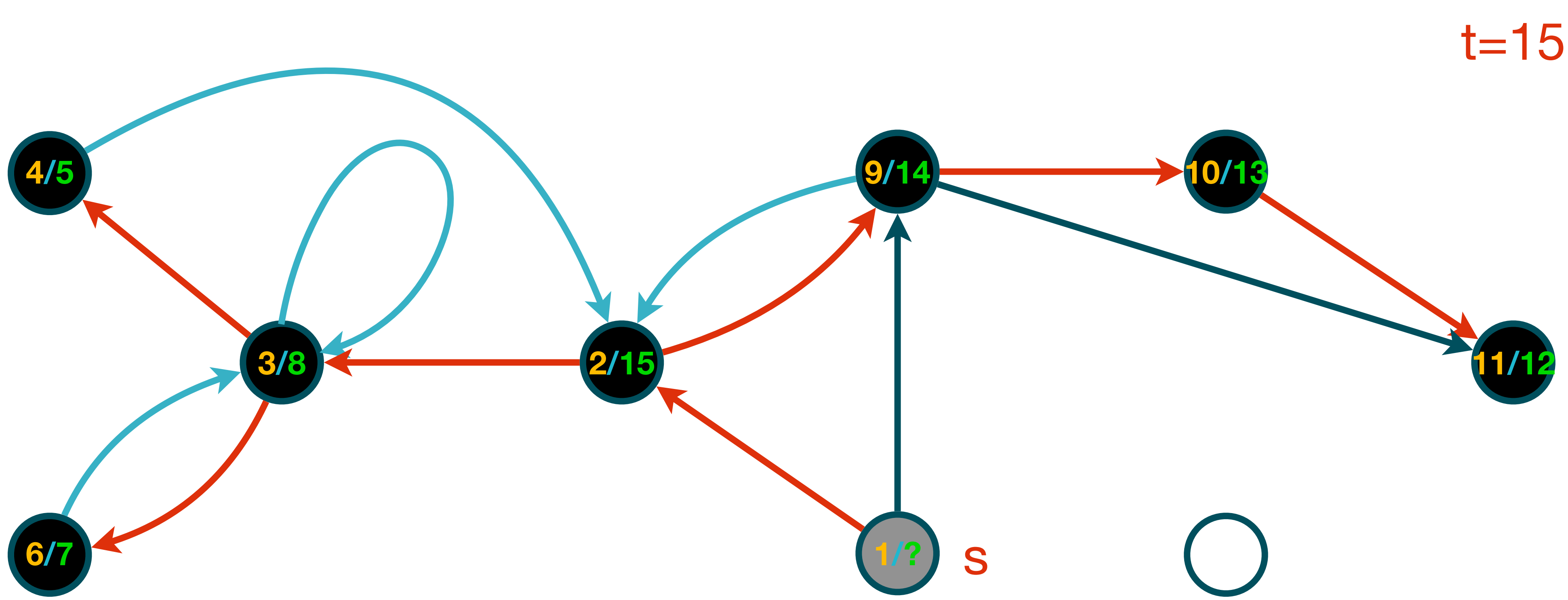
# Depth-First Search



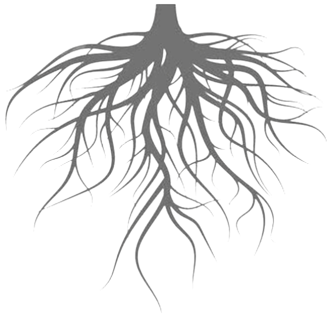
Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.



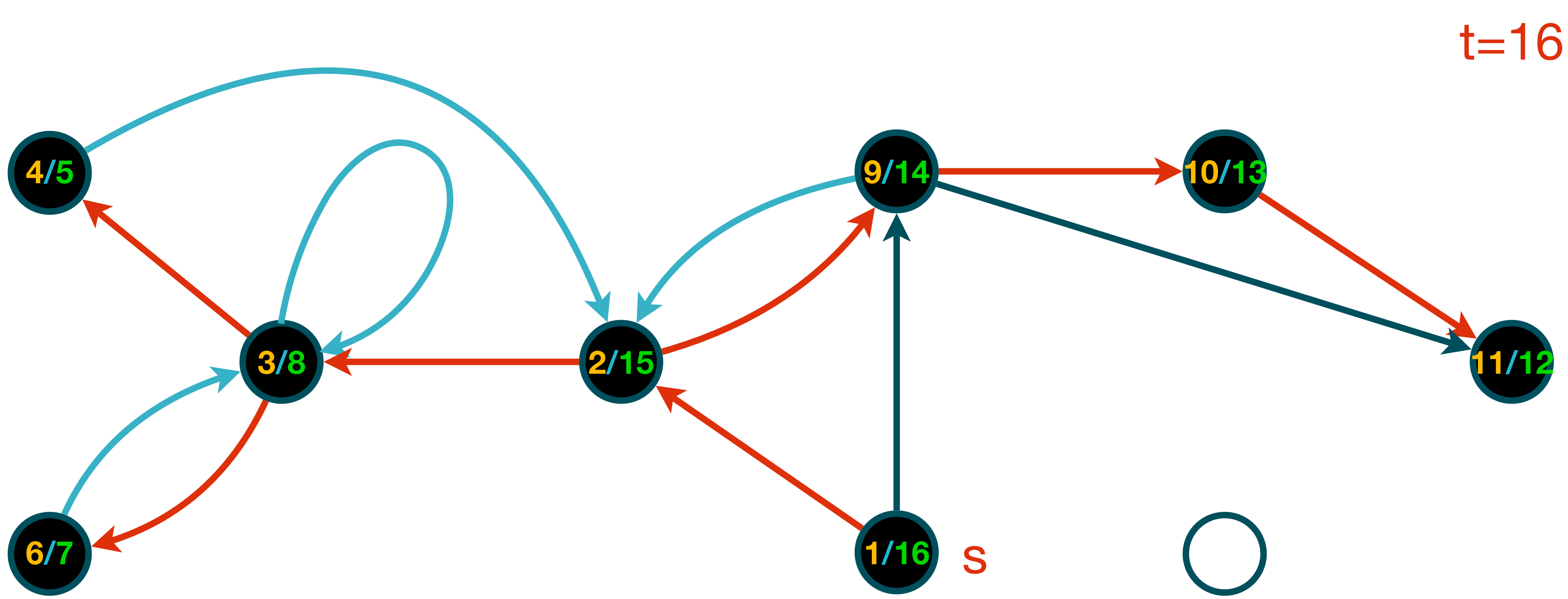
# Depth-First Search



Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

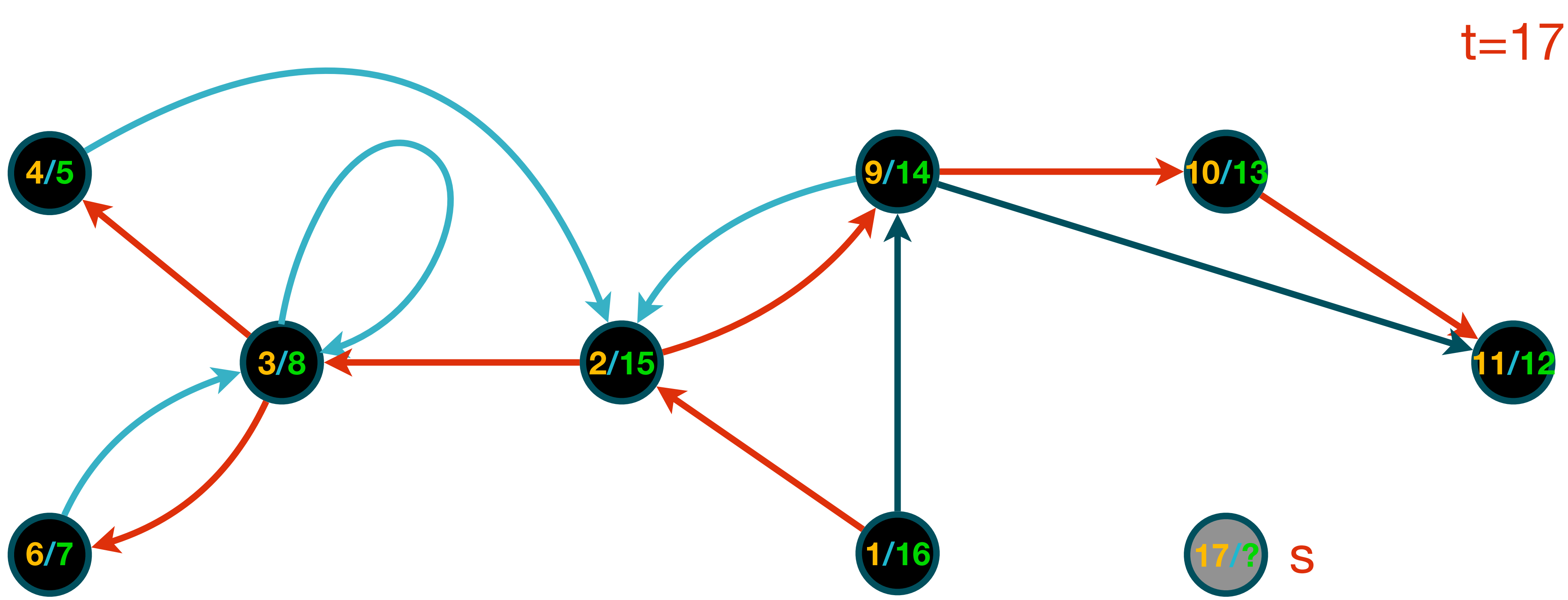


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

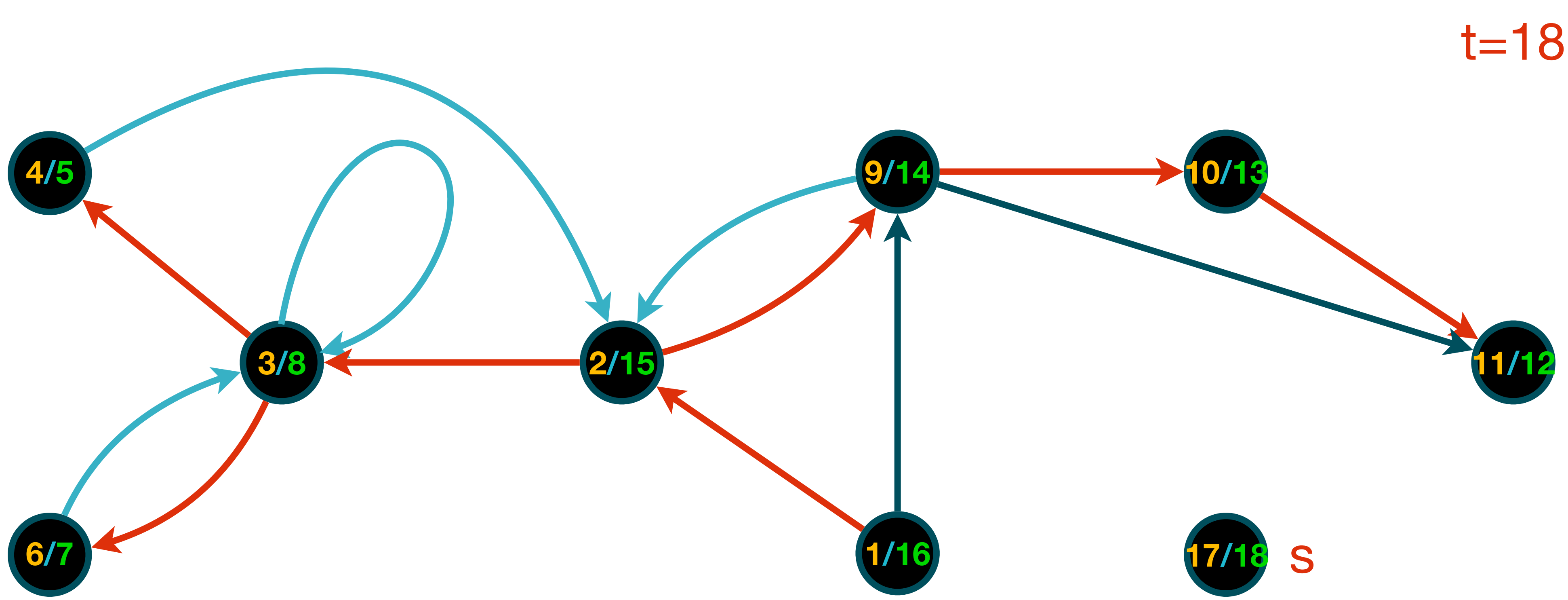


# Depth-First Search

Much like BFS, DFS colors the nodes of G during the visit.

Again, white nodes have not been visited yet; gray nodes have been discovered but have undiscovered neighbours; black nodes have been discovered and their neighbours too.

DFS assigns two timestamps to each node *v*: *v.d* records when *v* becomes gray, *v.f* records when it becomes black.

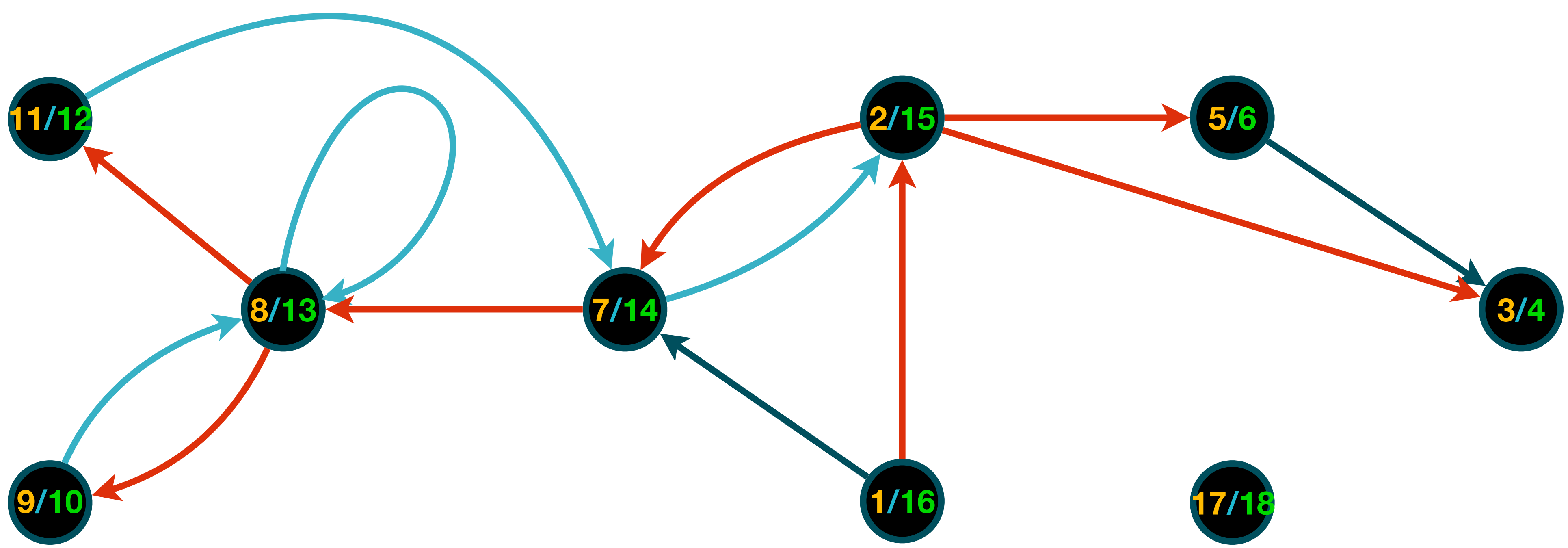


# Depth-First Search

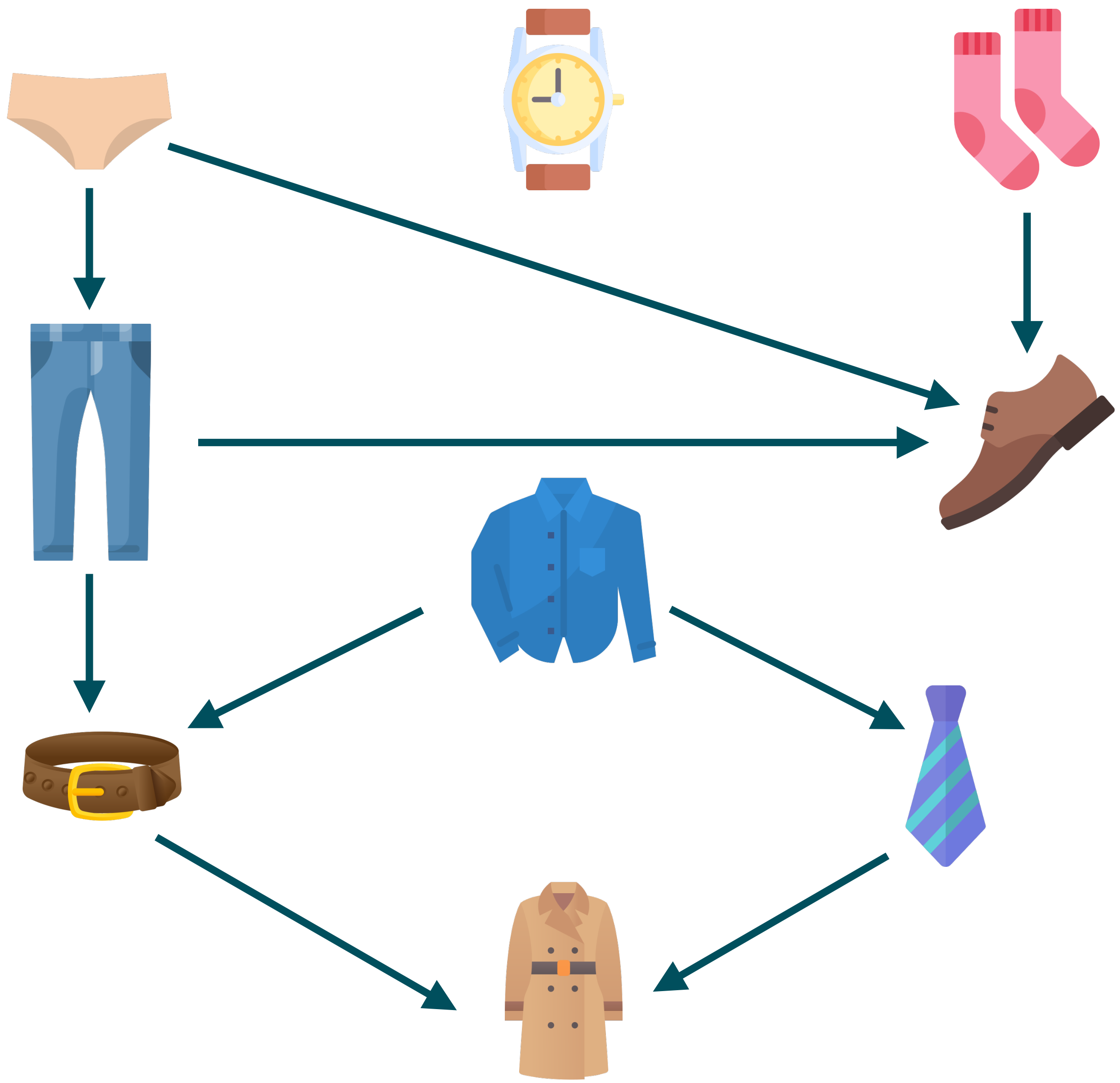
DFS produces a depth-first (DF) forest (a different tree for each source). Even for the same sources, this forest is not unique: it depends from the order in which the edges outgoing from each node are traversed. All the results are essentially equivalent.

The red edges are tree edges; the light blue edges are back edges, linking a node with one of its ancestors in the DF forest.

You can verify yourself that the result below is another possible outcome of DFS with the same two sources.

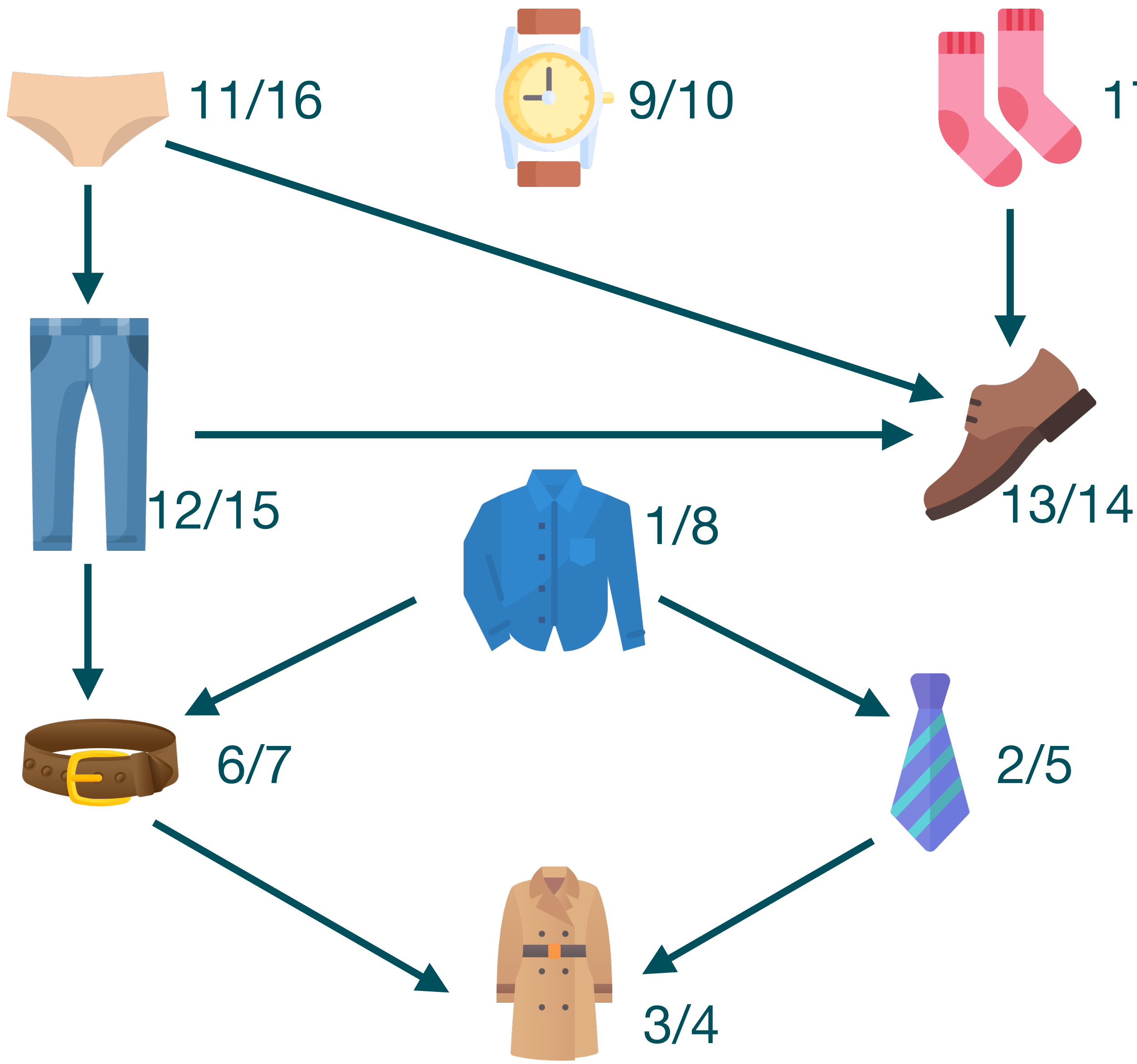


# An application: Topological Sort



An edge ($u$,$v$) indicates that item $u$ must be worn before item $v$.

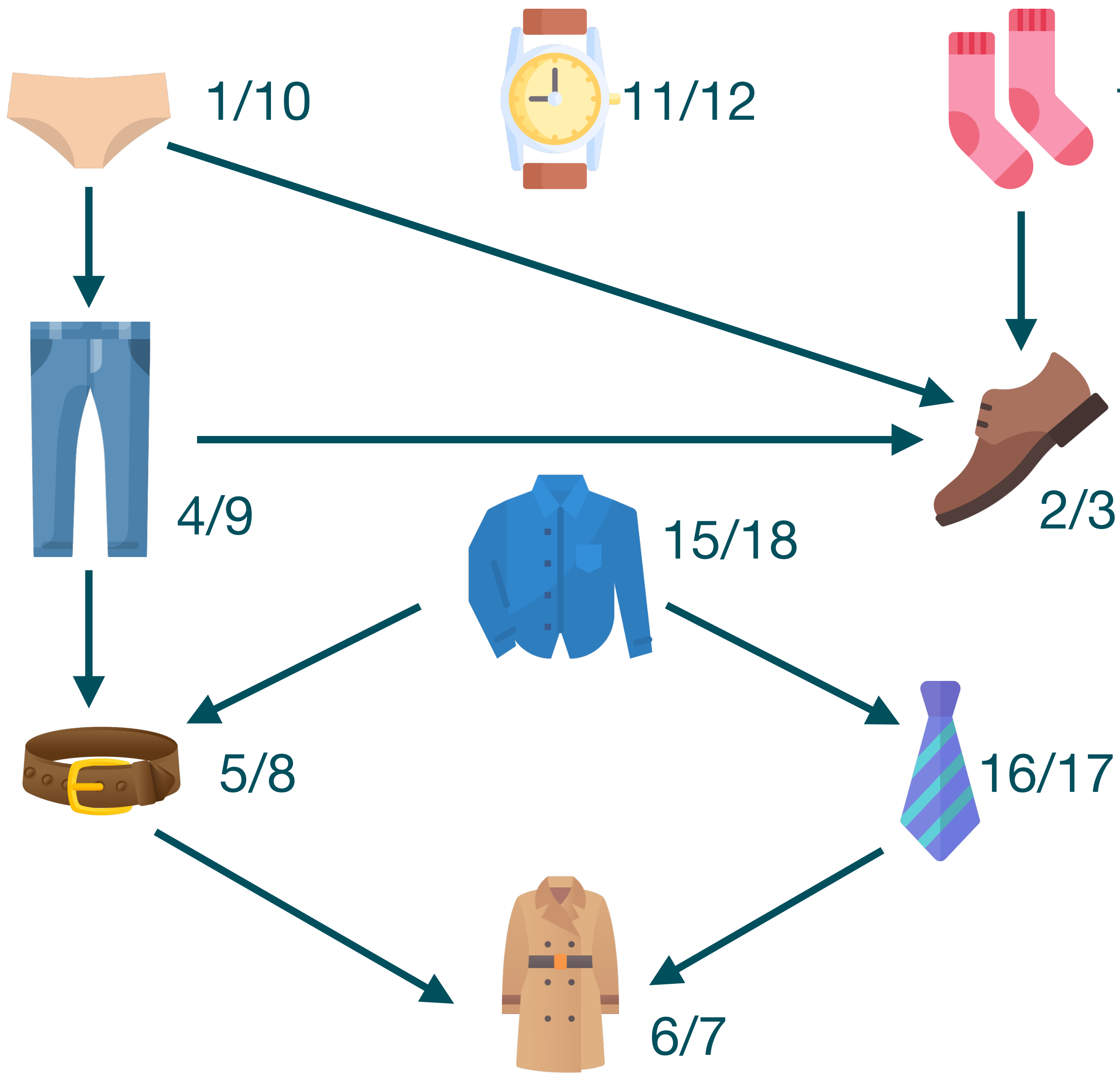
# An application: Topological Sort




TopologicalSort(G)
  DFS(G);
  $\tilde{V}[1, \ldots, |V|] \leftarrow V$ sorted w.r.t finishing time
  TopOrder←empty_stack;
  **for** $i = 1 \ldots |V|$
    TopOrder.push($\tilde{V}[i]$);
  **return** TopOrder;

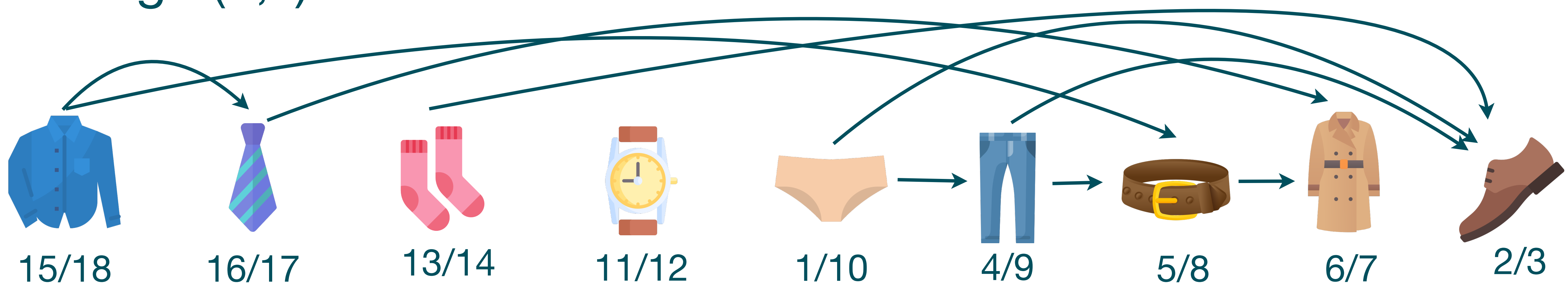
An edge ($u$,$v$) indicates that item $u$ must be worn before item $v$.

# An application: Topological Sort




TopologicalSort(G)
  DFS(G);
  $\tilde{V}[1,\ldots,|V|] \leftarrow V$ sorted w.r.t finishing time
  TopOrder←empty_stack;
  **for** $i = 1\ldots|V|$
    TopOrder.push($\tilde{V}[i]$);
  **return** TopOrder;

An edge ($u$,$v$) indicates that item $u$ must be worn before item $v$.



# Exercises

**EX (Cormen 17.1-1):** If the set of stack operations included a MULTIPUSH operation, which pushes k items onto the stack, would the O(1) bound on the amortized cost of stack operations continue to hold?

# Exercises

**EX1:** Given a connected, undirected graph, design an algorithm that assigns one of two colors (say blue or green) to each vertex in such a way that no edge links two vertices of the same color; or return FAIL if no such coloring is possible.

# Exercises

**EX2:** Give an $O(|V|)$-time algorithm that determines whether or not a given undirected graph contains a cycle. (*Hint: Think of the maximum number of edges that an acyclic undirected graph may have; use DFS and terminate it early when appropriate*).