Graph Traversals II

Instructor: Meng-Fen Chiang

COMPCSI220: WEEK 10

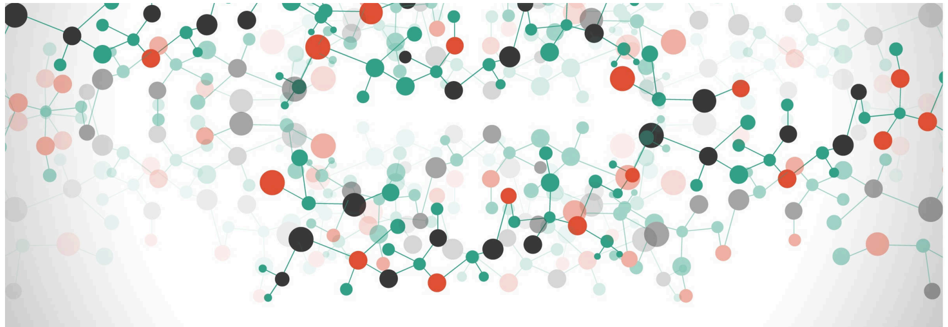
THE UNIVERSITY OF AUCKLAND
Te Whare Wananga o Tamaki Makaurau
NEW ZEALAND

Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz and Tanya Gvozdeva

OUTLINE

- Graph Traversal Algorithms
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Priority-first Search (PFS)

- Implementation
  - Stack – DFS
  - Queue – BFS
  - Priority Queues – PFS



Graph Traversals

Algorithm 1 Visit.

function VISIT(node s of digraph G)

$color[s] \leftarrow$ Grey

$pred[s] \leftarrow$ Null

while there is a Grey node **do**

choose a Grey node u

if u has a WHITE (out-)neighbour **then**

choose such a (out-)neighbour v

$color[v] \leftarrow$ Grey

$pred[v] \leftarrow u$

else

$color[u] \leftarrow$ Black

how to choose?

Implementation of the List of Frontiers

- The implementation of the list that stores the frontiers (grey nodes) will directly affect the order we traverse the graph nodes. Three types of implementations will be discussed and result in three different traversal strategies:

- Stack – Depth-first search (DFS)

- Queue – Breadth-first search (BFS)

- Priority Queues – Priority first search (PFS)

The Abstract Data Type: Stack

- Special list in which all operations occur at the same end (top) (last in first out).
- Add an element to the list (INSERT or PUSH).
- Delete and element (DELETE or POP).
- Return top element without deleting it (GETTOP or PEEK)

Depth-first Search Algorithm (DFS)

- DFS is a specific implementation of our fundamental graph traversal algorithm (also known as depth-first traversal)

- It specifies that we select the next grey vertex to pick as the youngest remaining grey vertex.

Depth-first-search (DFS) Algorithm

Algorithm 1 Depth-first search algorithm

function DFS(digraph G)
    stack S
    array $colour[0..n-1], pred[0..n-1], seen[0..n-1], done[0..n-1]$
    for $u \in V(G)$ **do**
        $colour[u] \leftarrow$ WHITE
        $pred[u] \leftarrow$ null
    $time \leftarrow 0$
    for $s \in V(G)$ **do**
        if $colour[s] =$ WHITE **then**
            DFSVISIT(s)
    return $pred, seen, done$

Iterative View of DFSVISIT

Algorithm 2 Depth-first visit algorithm.

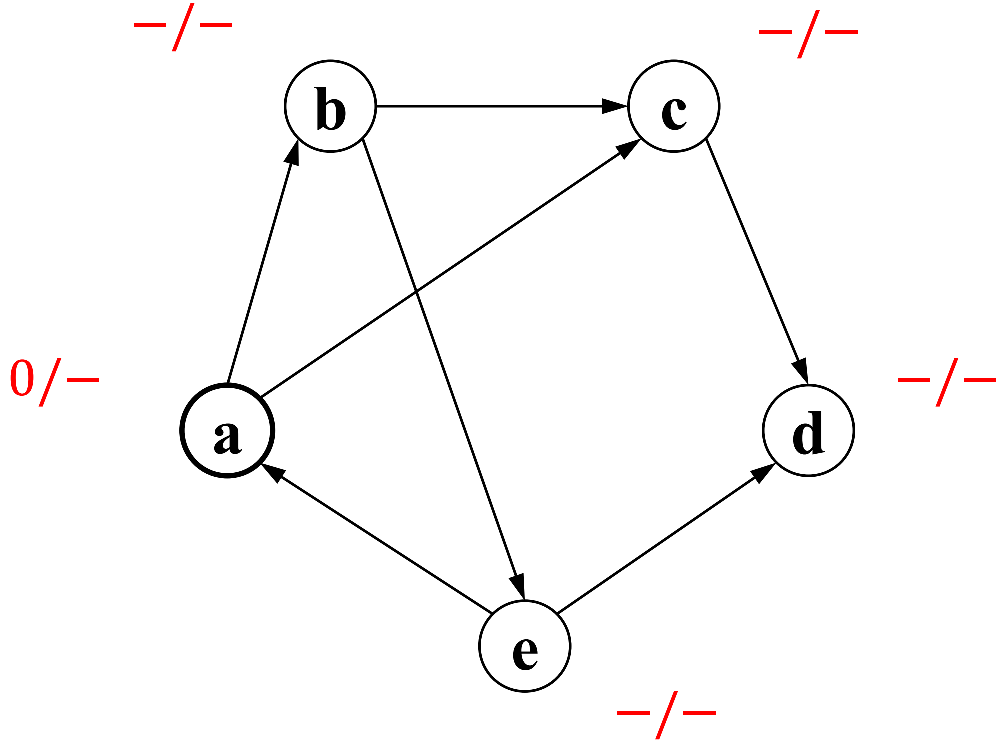
function DFSVISIT(node s)
    $color[s] \leftarrow$ GREY
    $seen[s] \leftarrow time; time \leftarrow time + 1$
    $S.insert(s)$
    while not $S.isEmpty()$ **do**
        $u \leftarrow S.peek()$
        if there is a neighbour v with $colour[v] =$ WHITE **then**
            $colour[v] \leftarrow$ GREY; $pred[v] \leftarrow u$
            $seen[v] \leftarrow time; time \leftarrow time + 1$
            $S.insert(v)$
        else
            $S.delete()$
            $colour[u] \leftarrow$ BLACK
            $done[u] \leftarrow time; time \leftarrow time + 1$

Recursive View of DFSVISIT

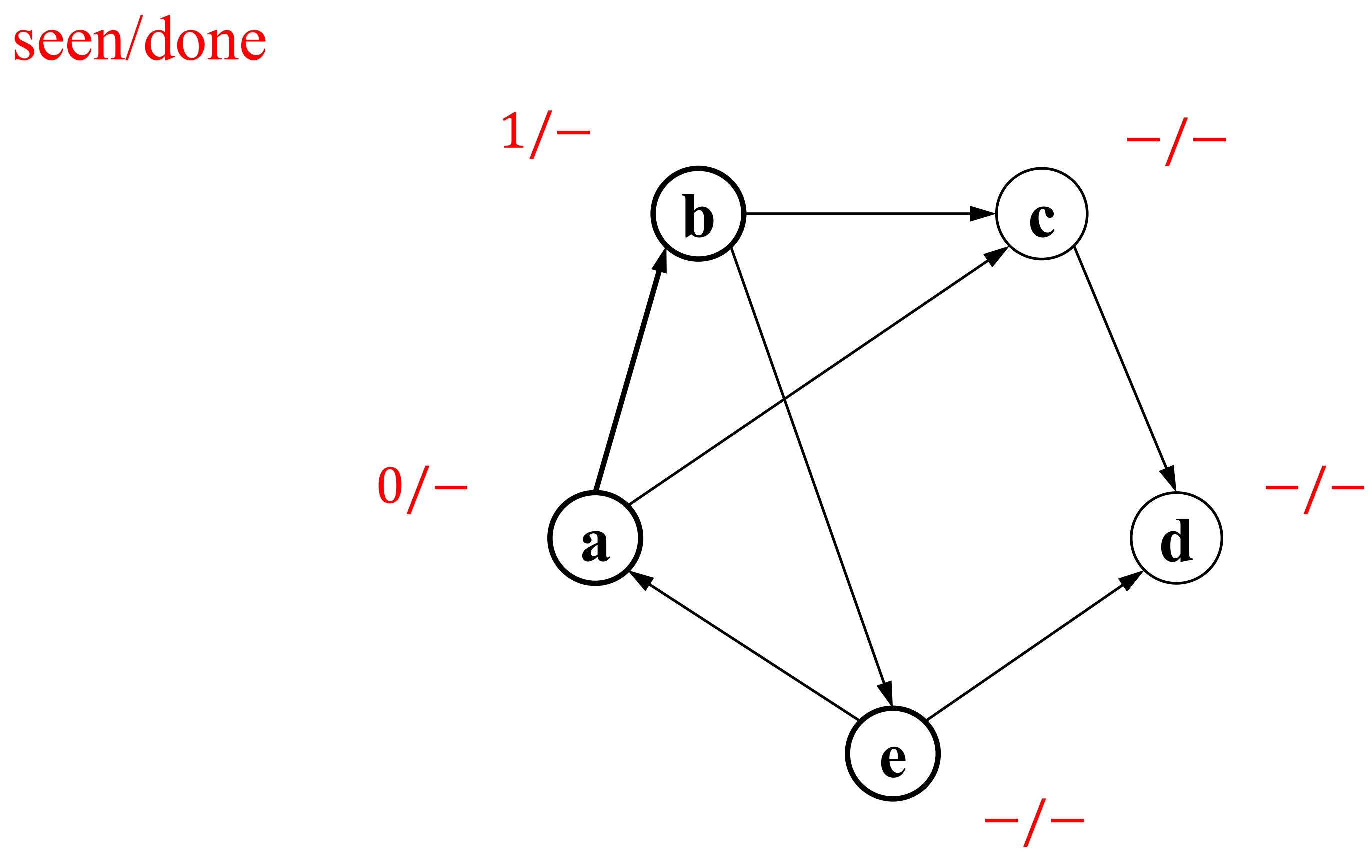
Algorithm 3 Recursive DFS visit algorithm.

function REC_DFSVISIT(node s)
    $color[s] \leftarrow$ GREY
    $seen[s] \leftarrow time; time \leftarrow time + 1$
    for each v adjacent to s **do**
        if $colour[v] =$ WHITE **then**
            $pred[v] \leftarrow s$
            REC_DFSVISIT(v)
    $colour[s] \leftarrow$ BLACK
    $done[s] \leftarrow time; time \leftarrow time + 1$

A DFS example (1)

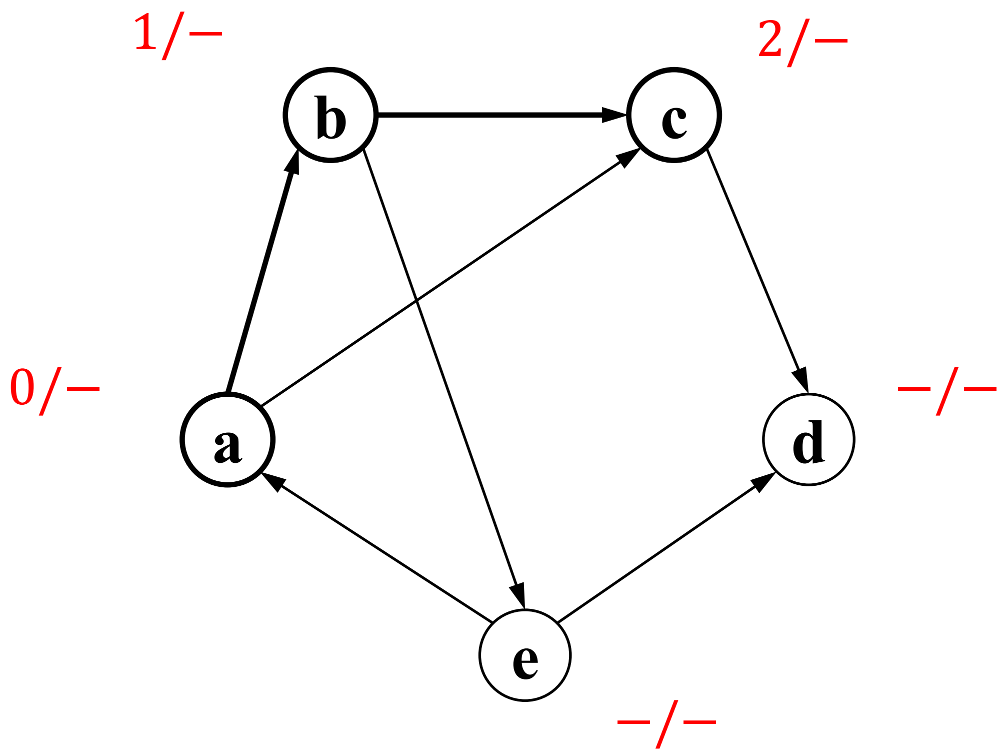


A DFS example (1)



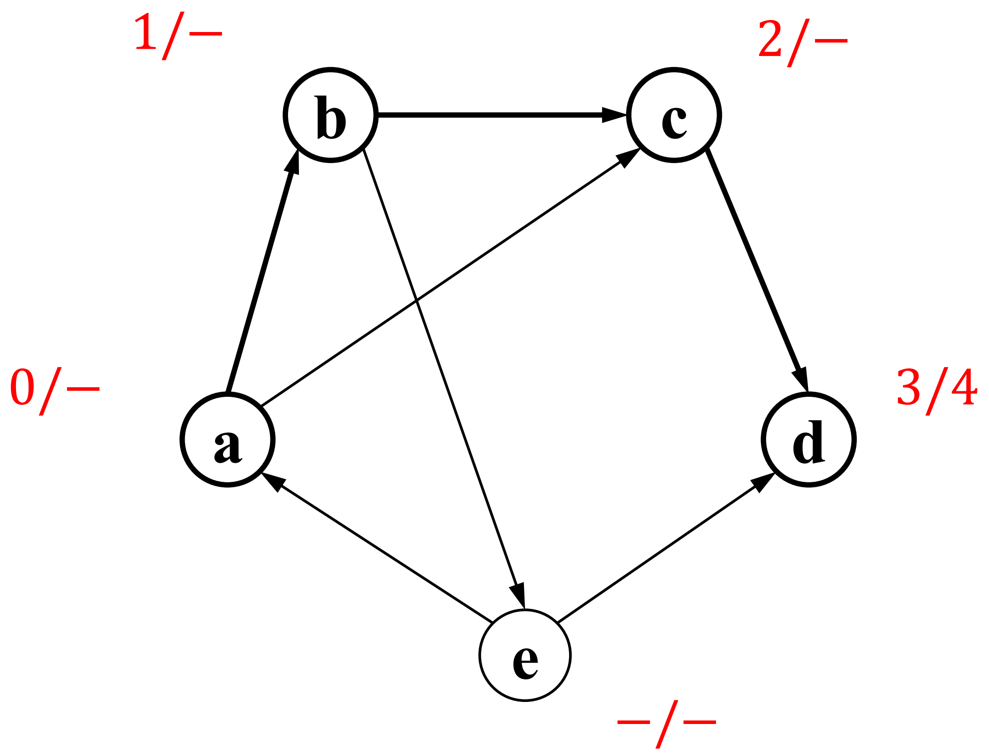
A DFS example (1)

seen/done



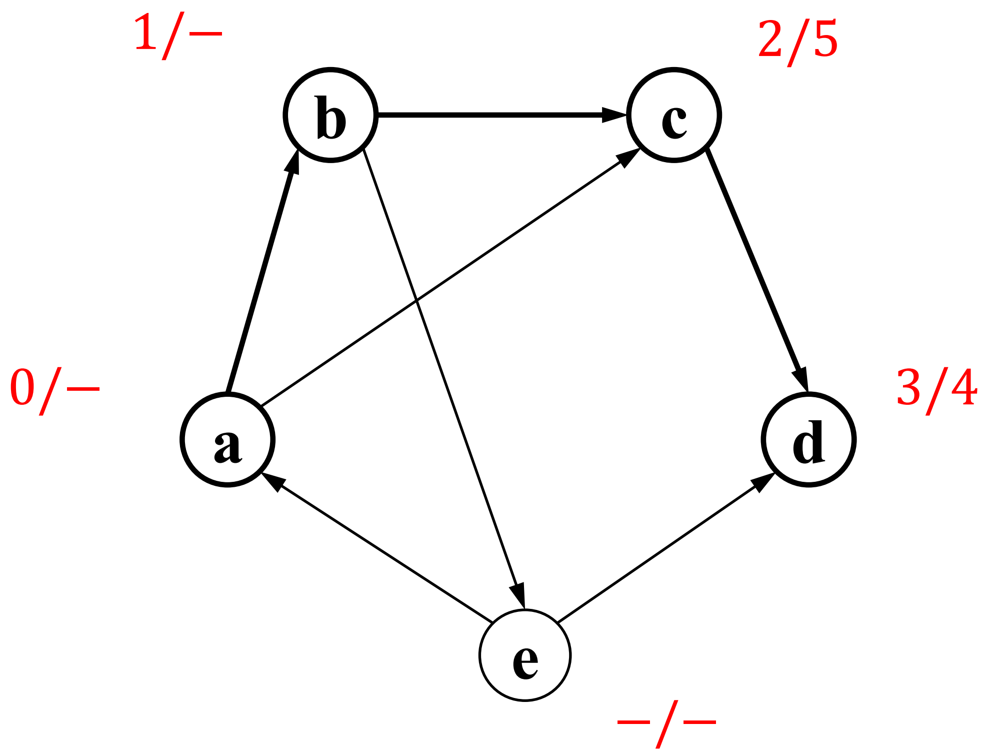
A DFS example (1)

seen/done

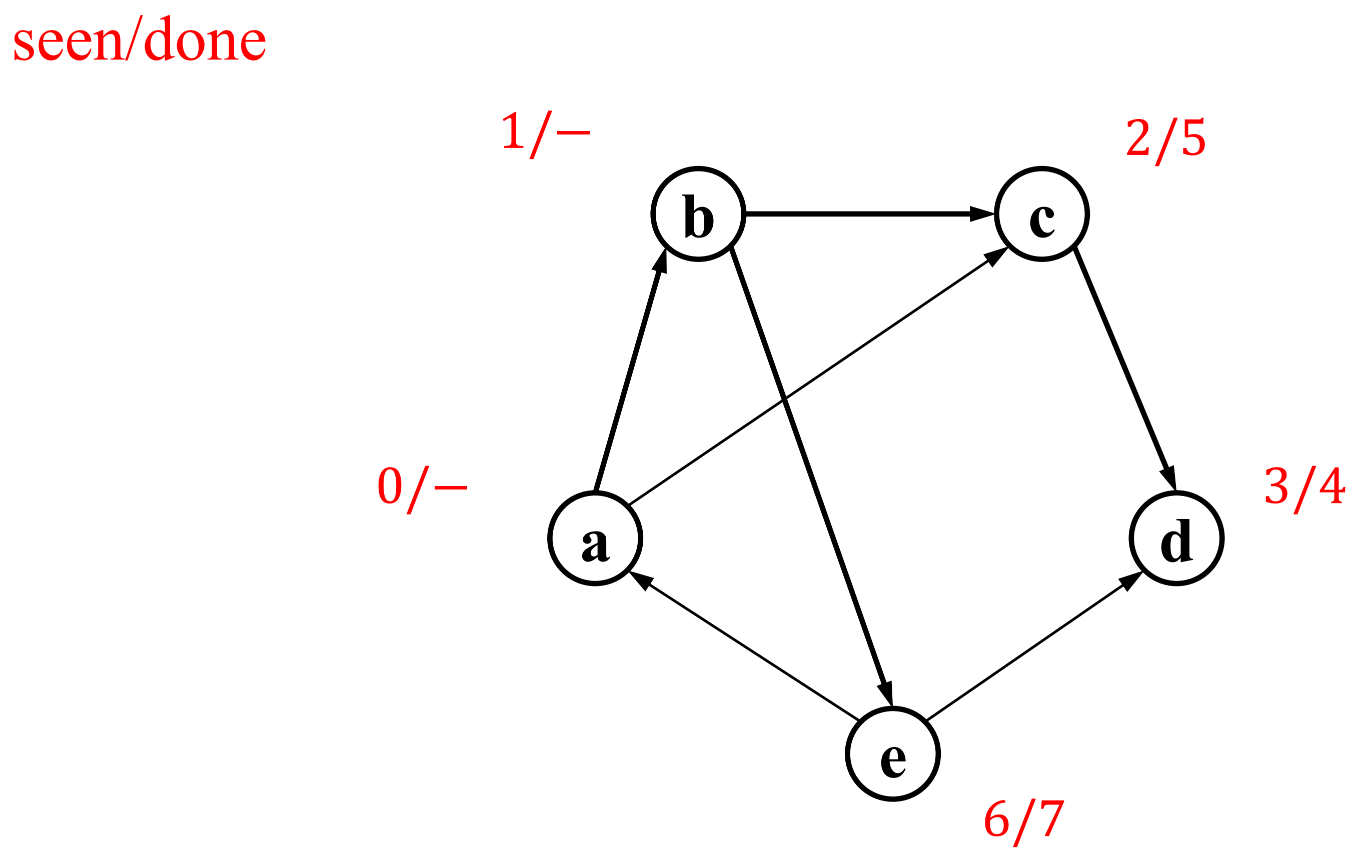


A DFS example (1)

seen/done

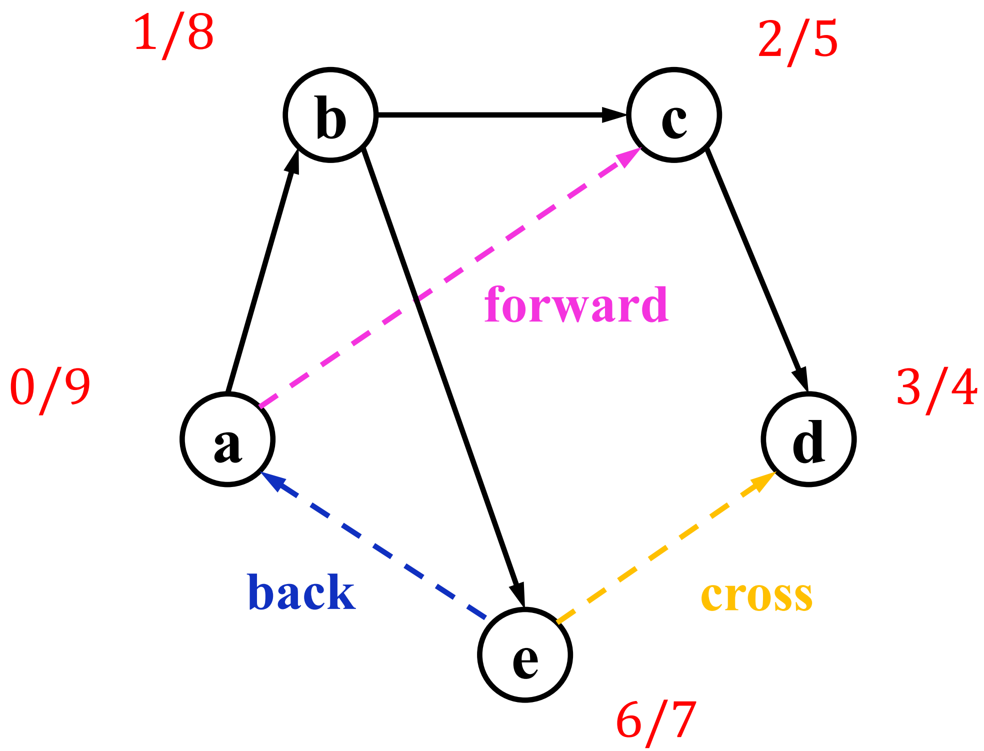


A DFS example (1)

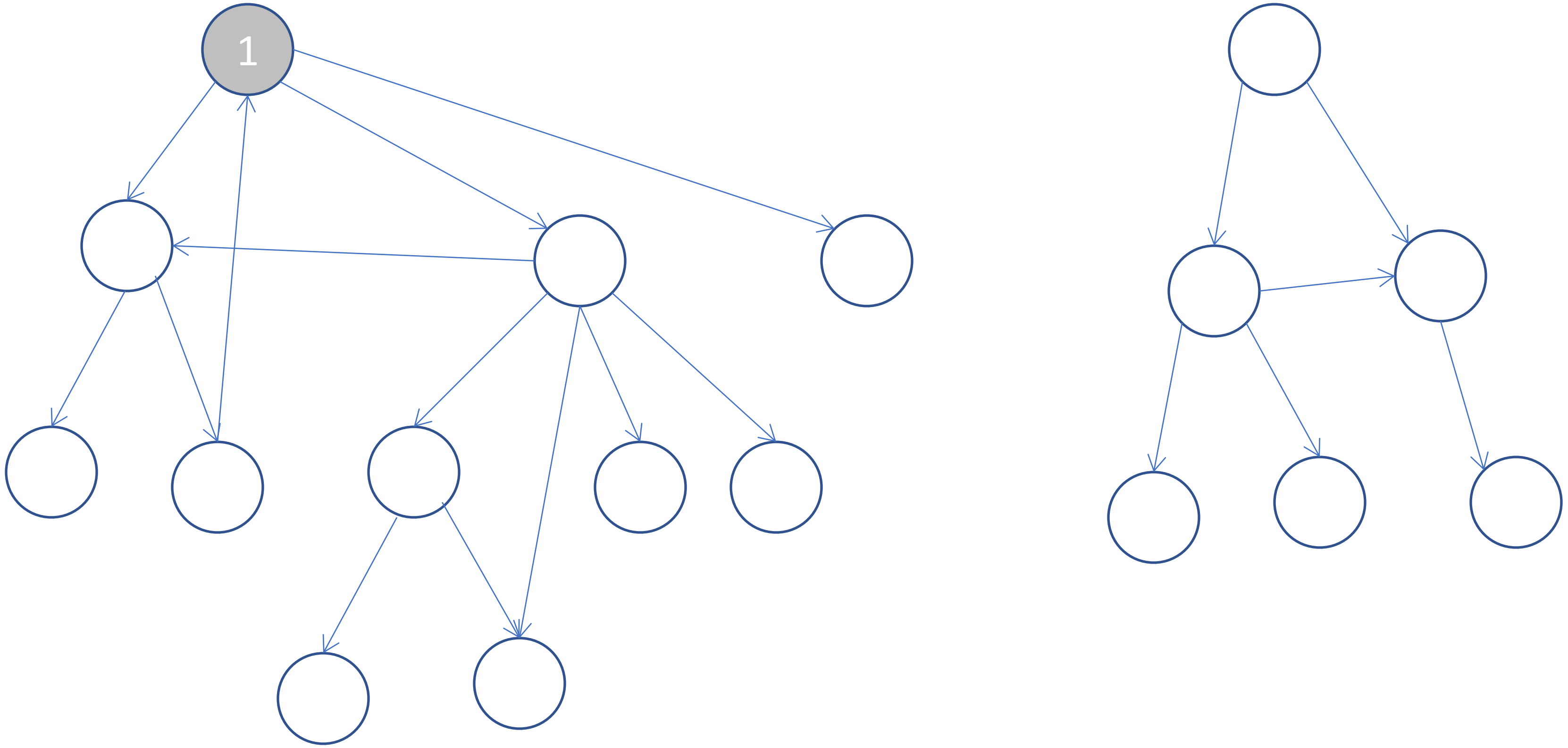


A DFS example (1)

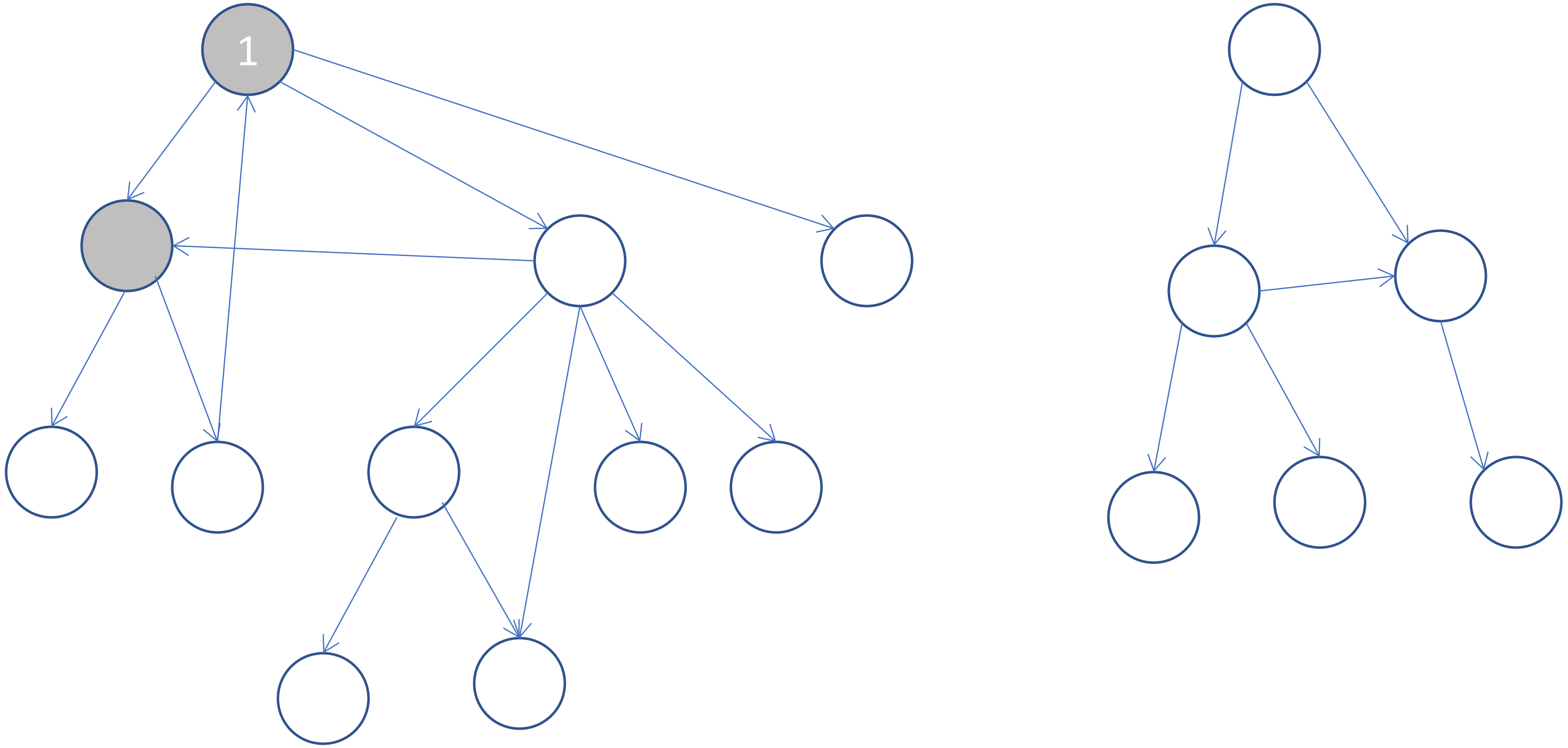
seen/done



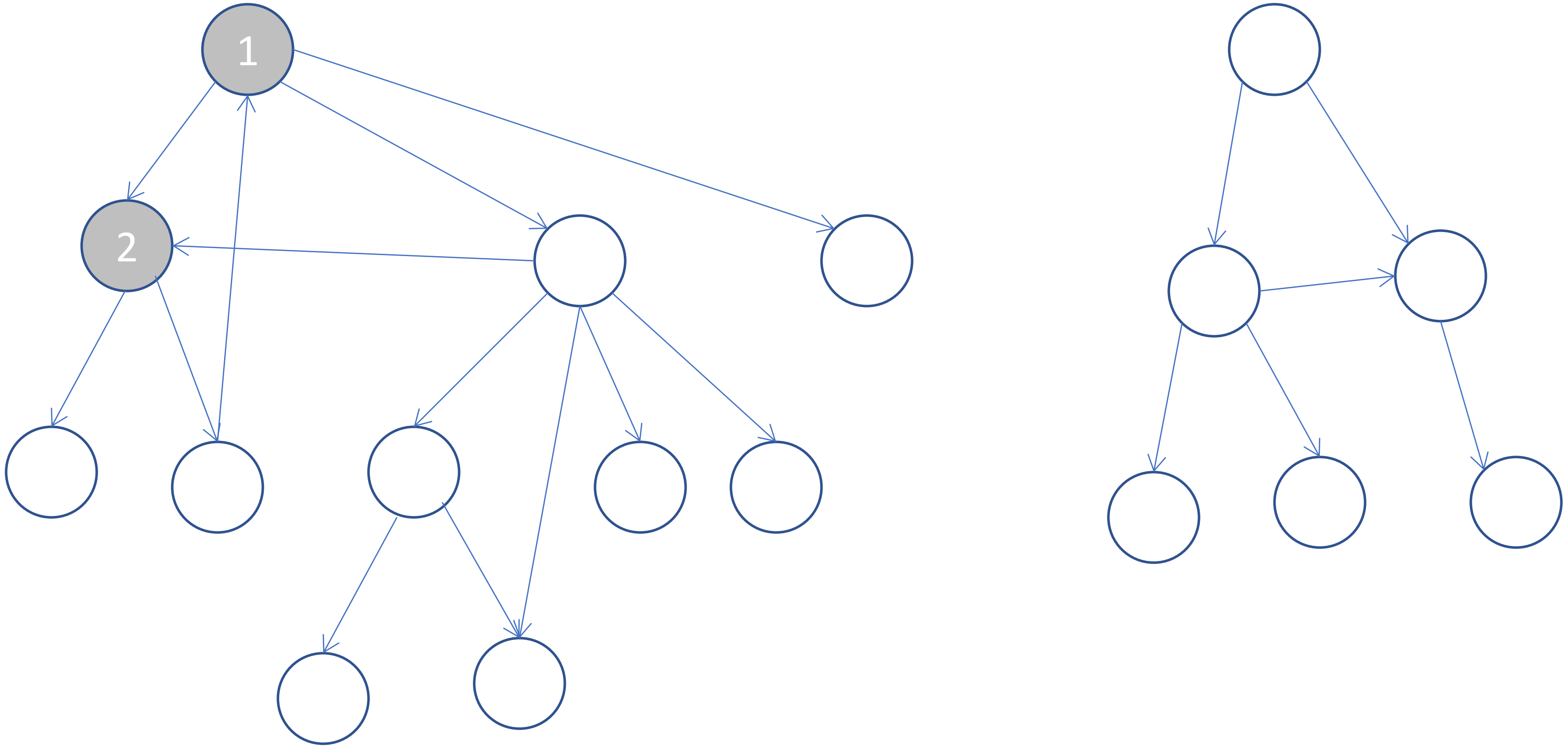
A DFS example (2)



A DFS example (2)

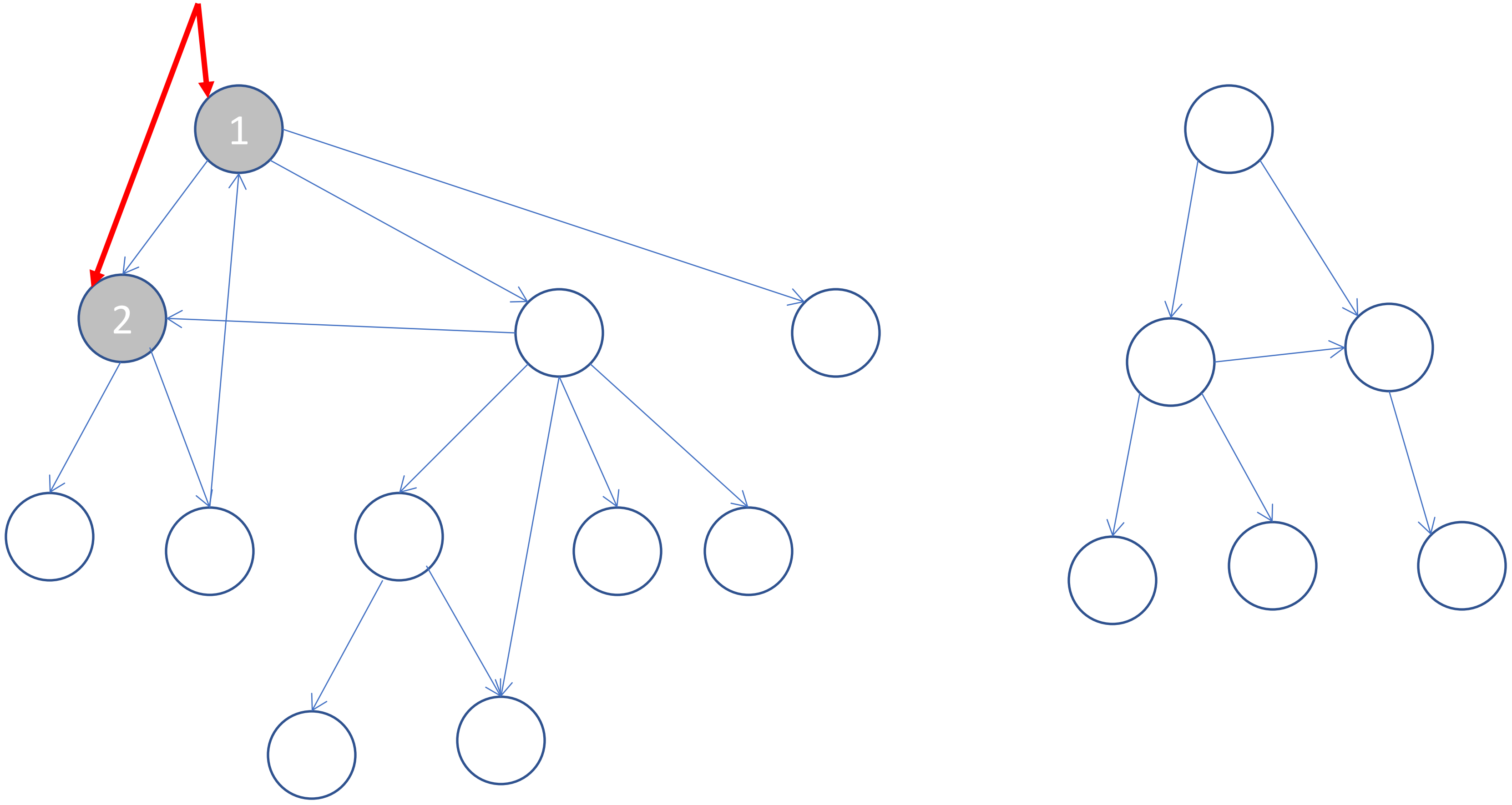


A DFS example (2)



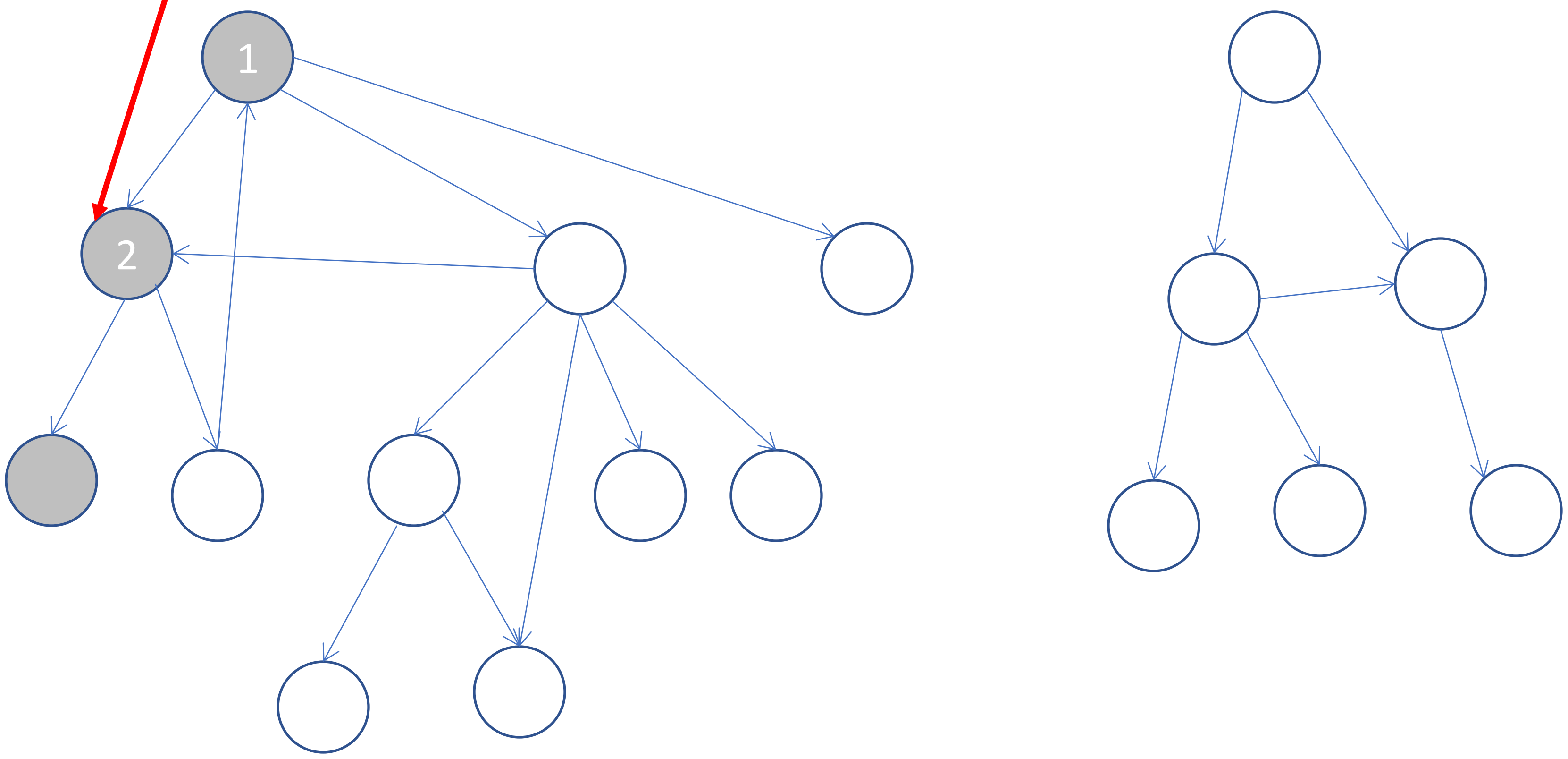
A DFS example (2)

Two grey to chose
from…which one?

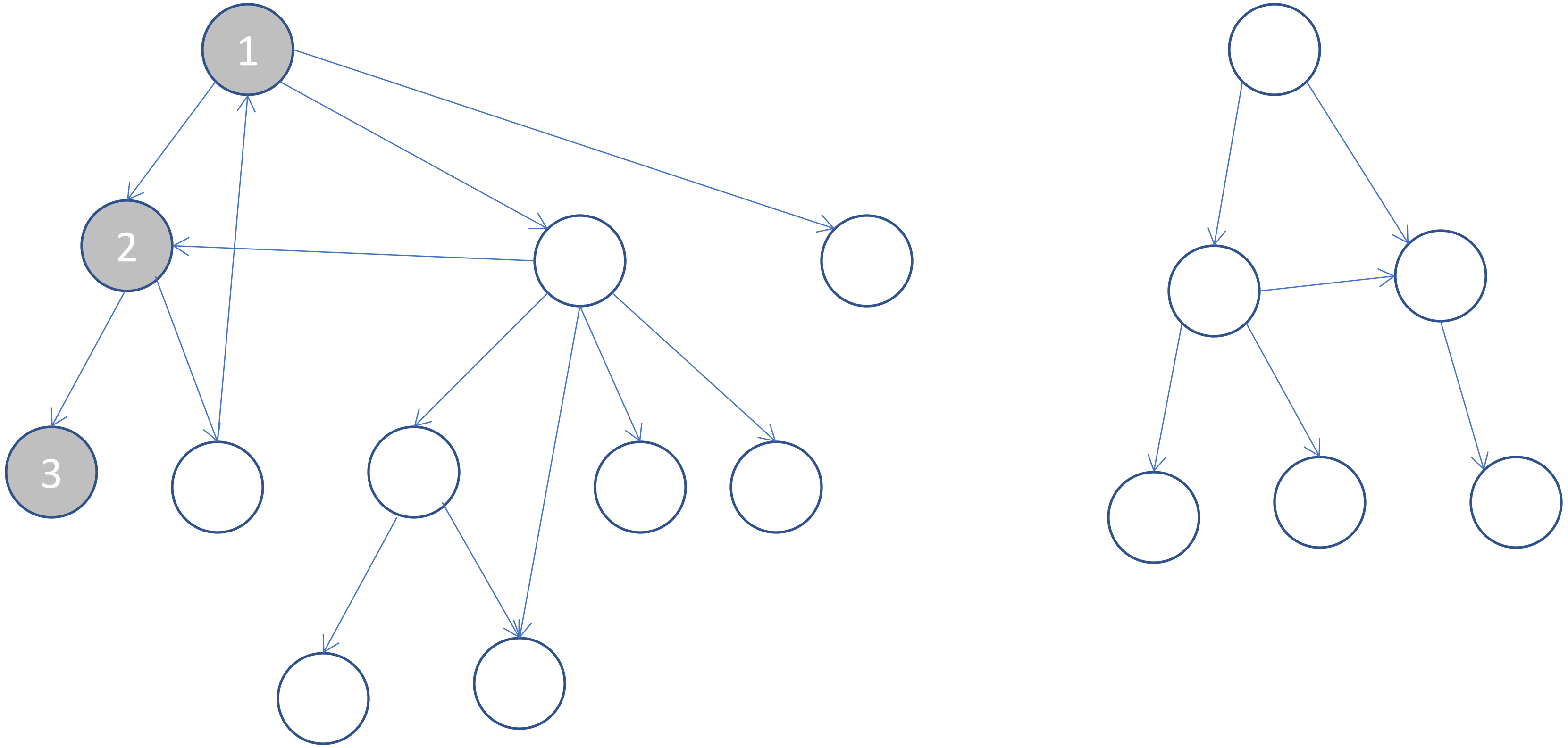


A DFS example (2)

2 is the "youngest"

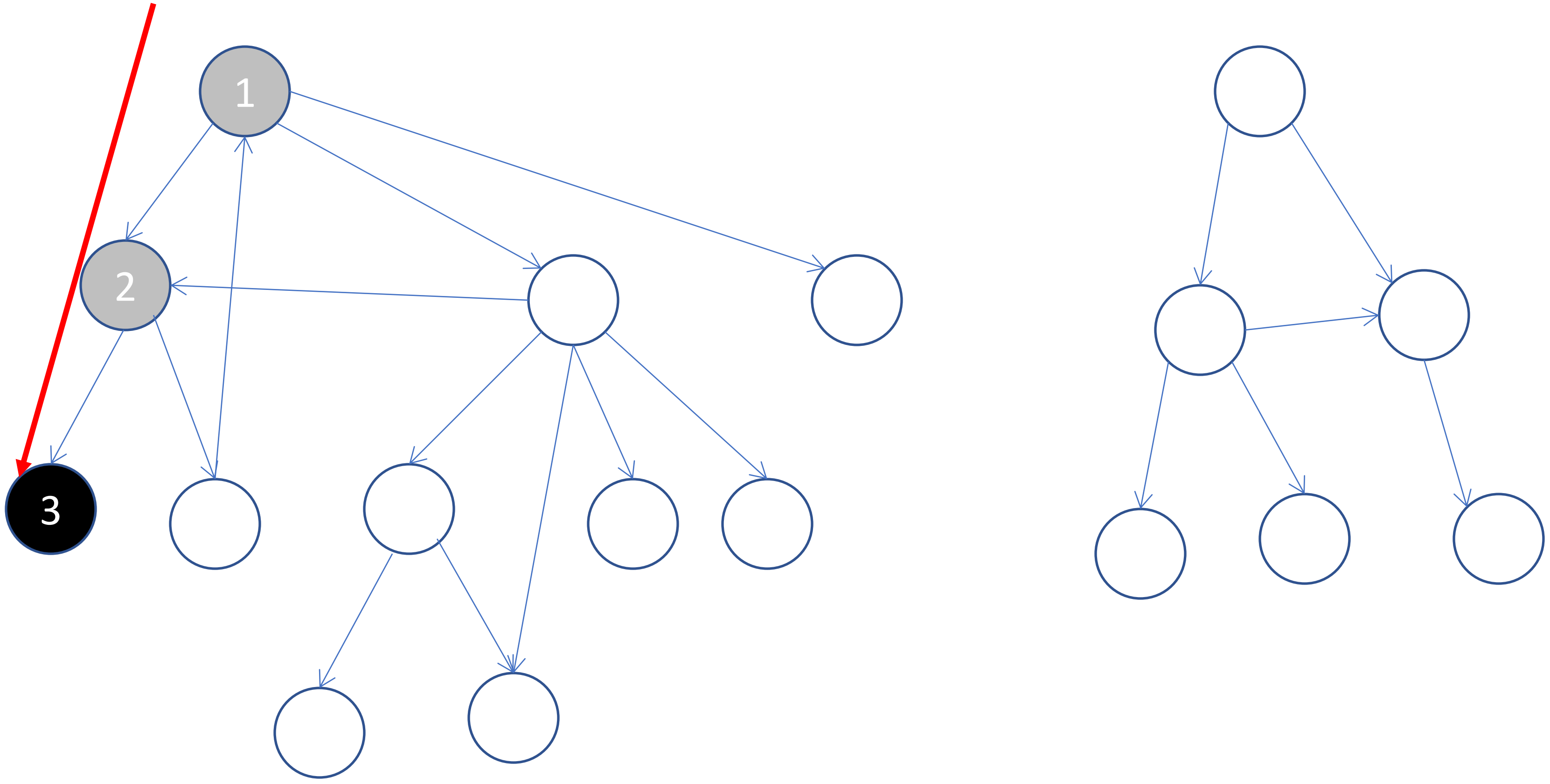


A DFS example (2)

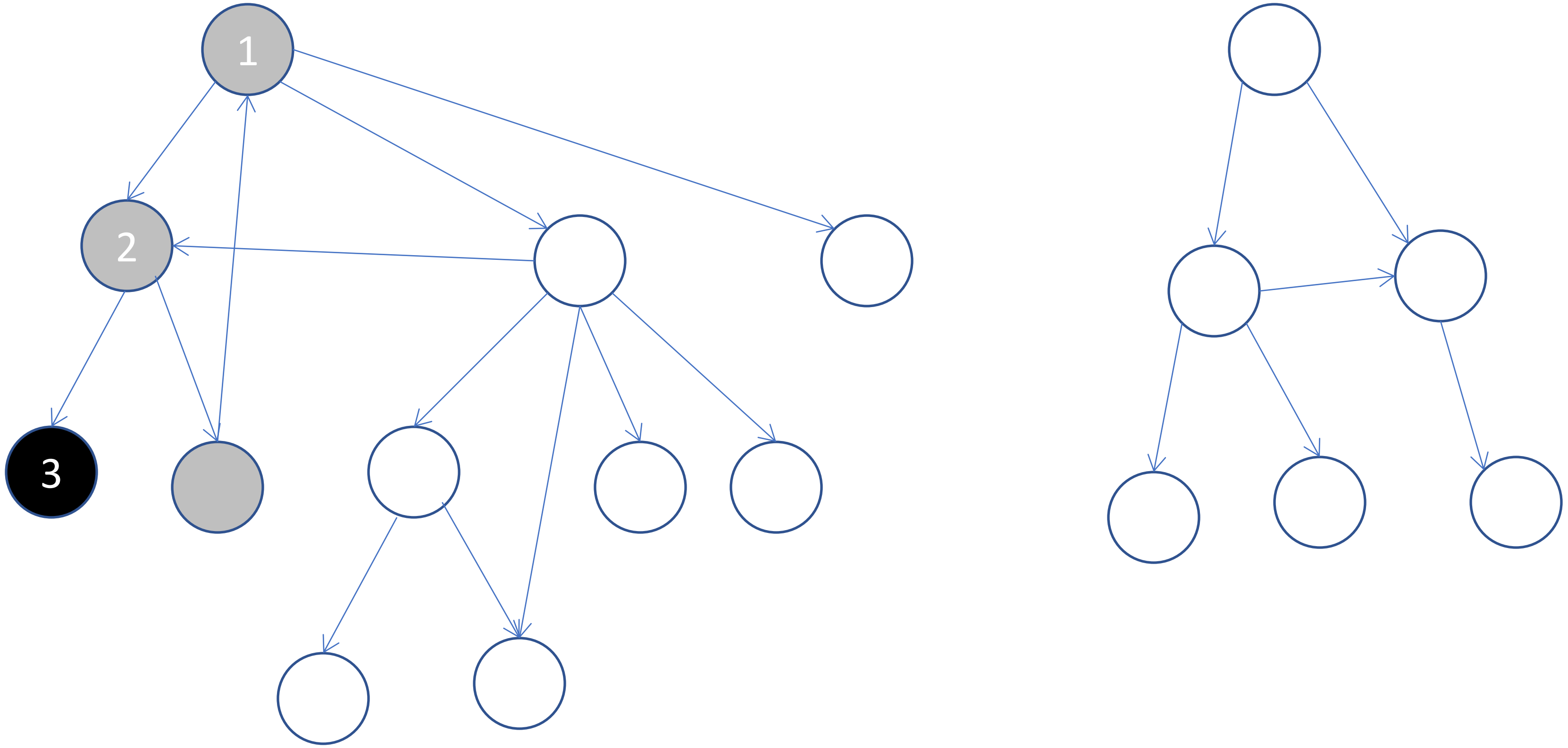


A DFS example (2)

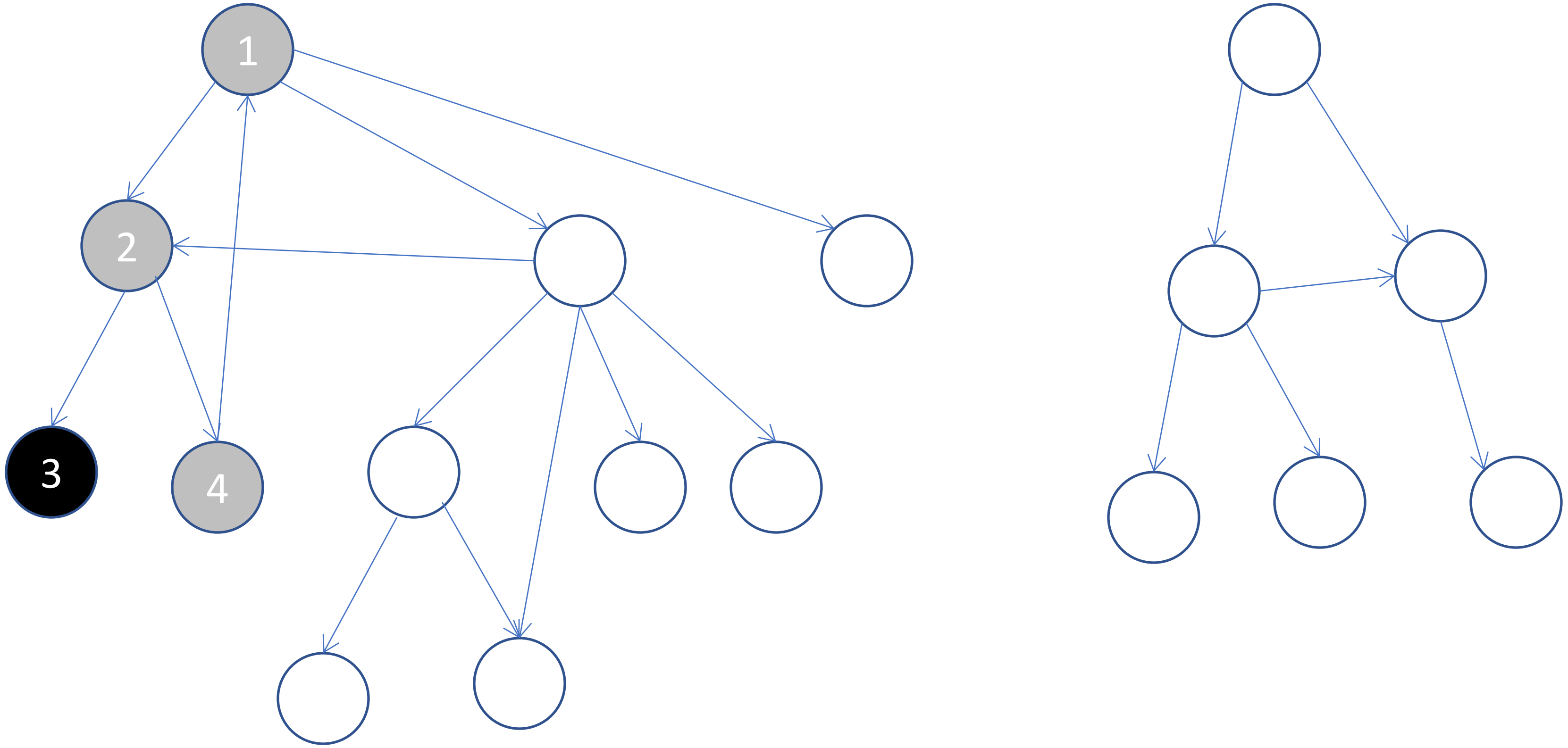
Node 3 does not have anything to offer so it turns black



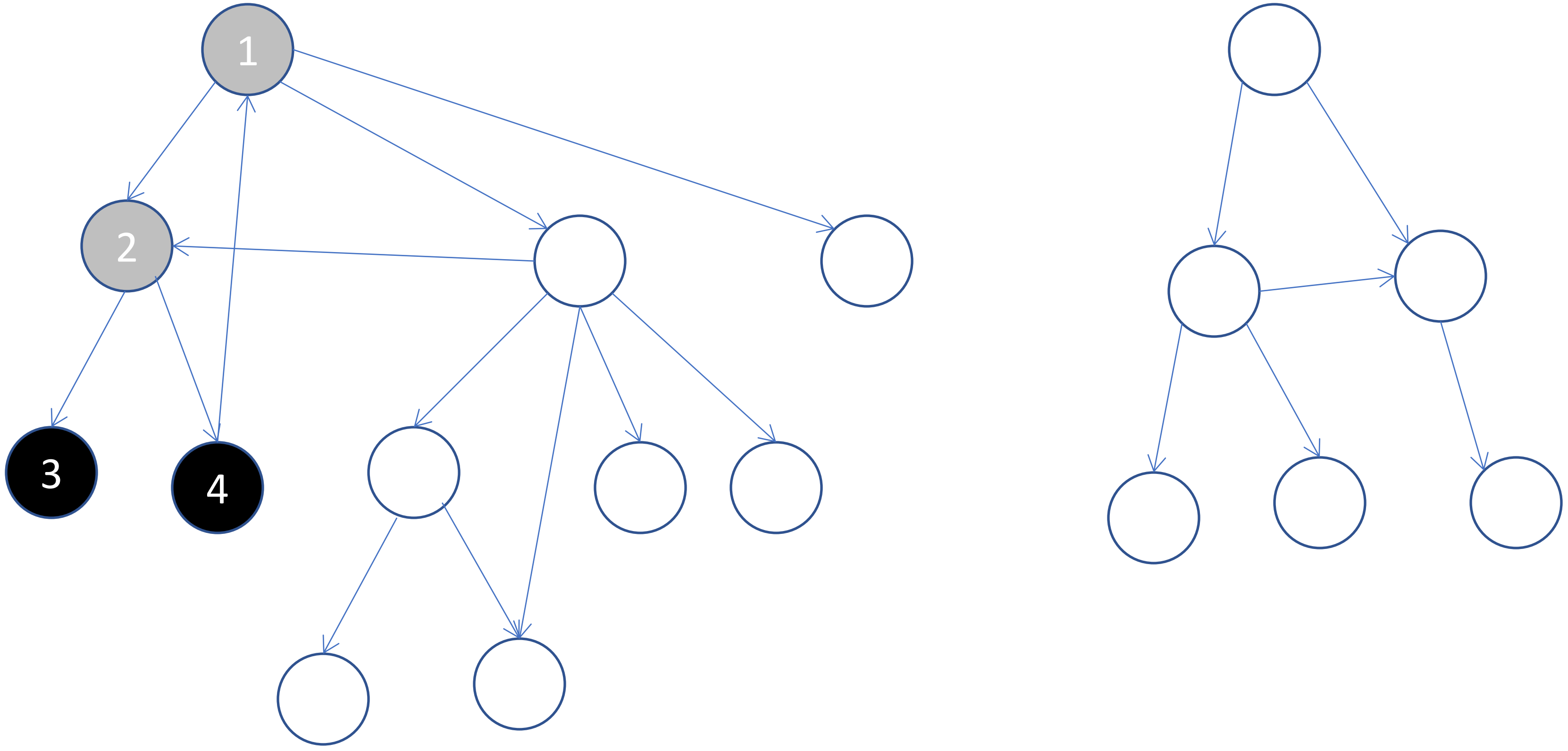
A DFS example (2)



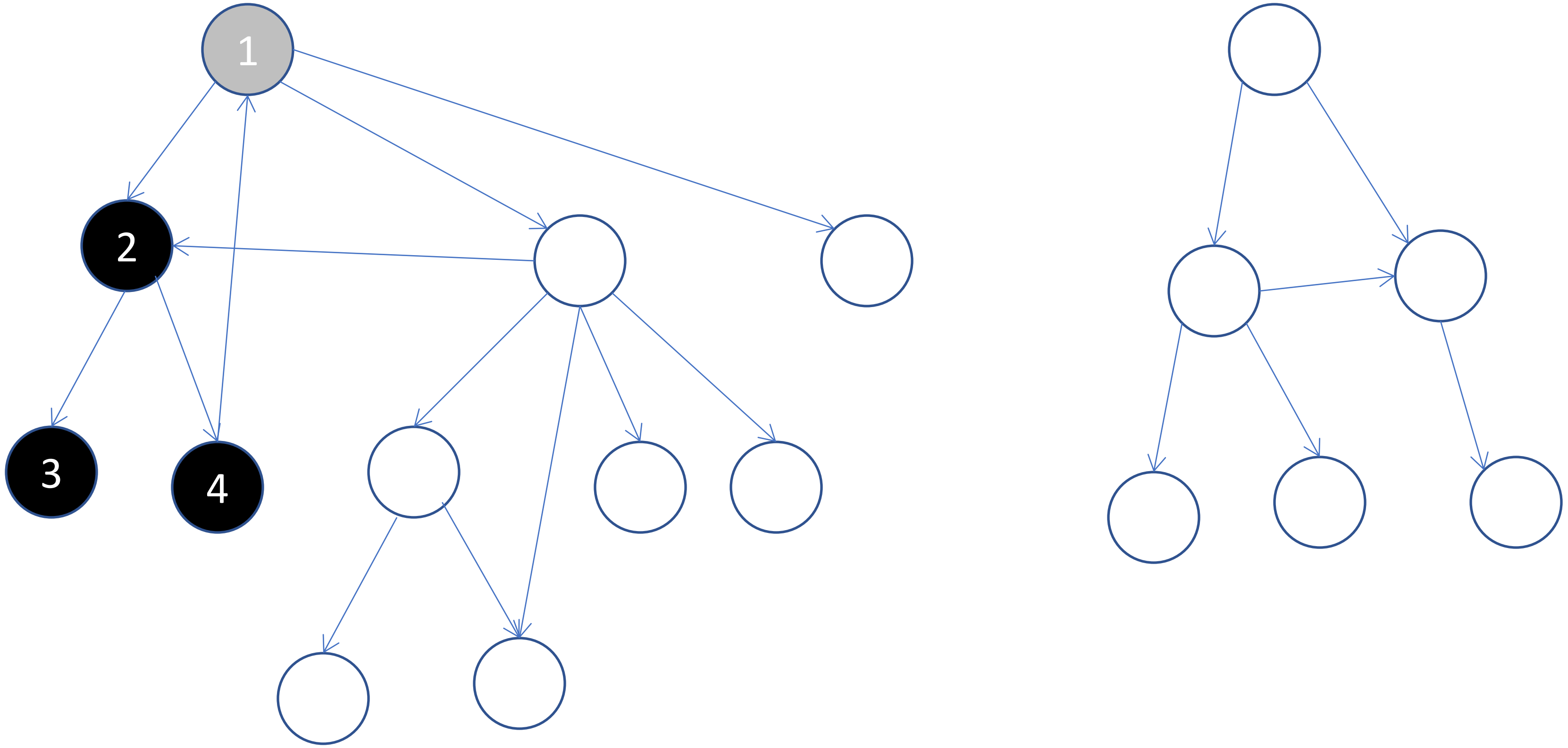
A DFS example (2)



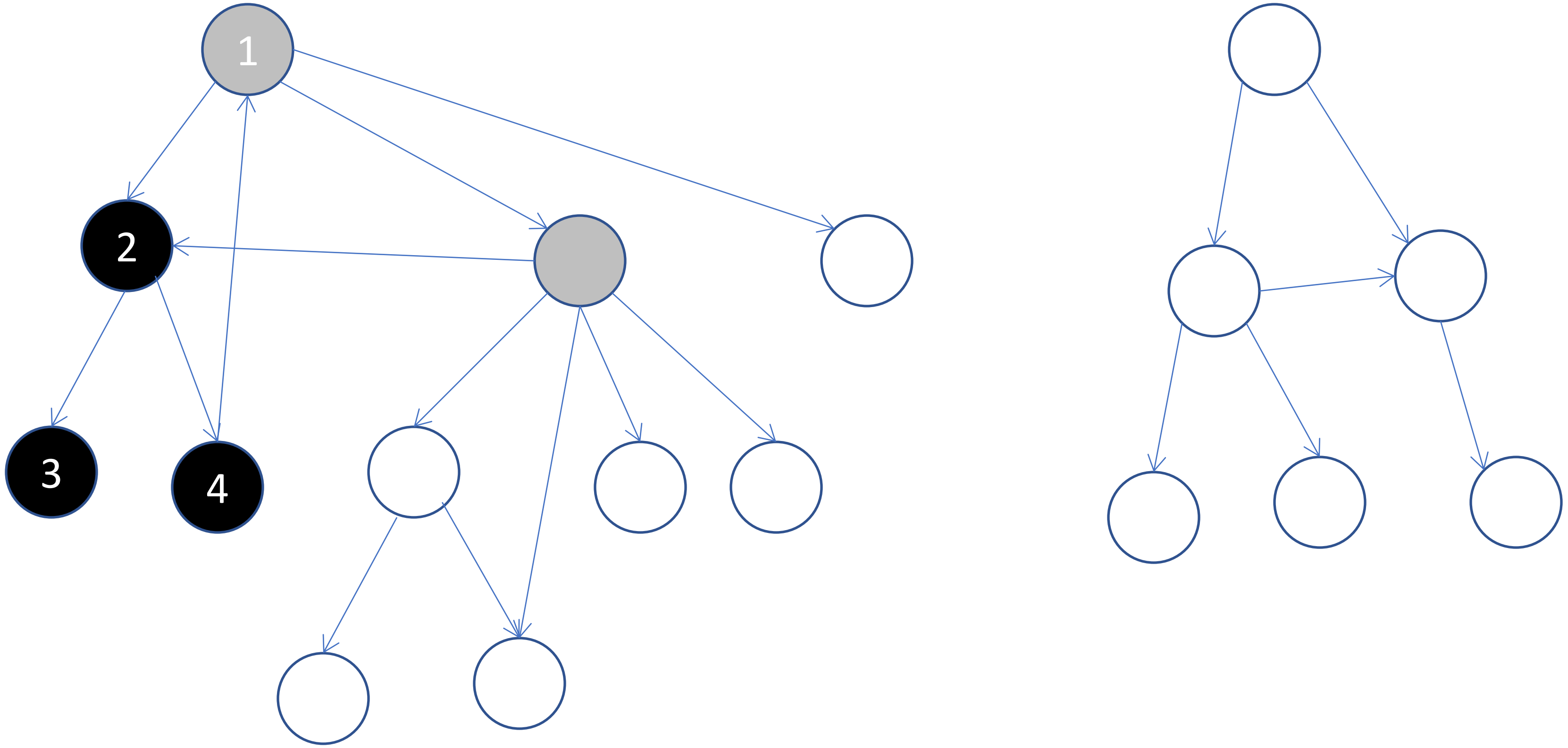
A DFS example (2)



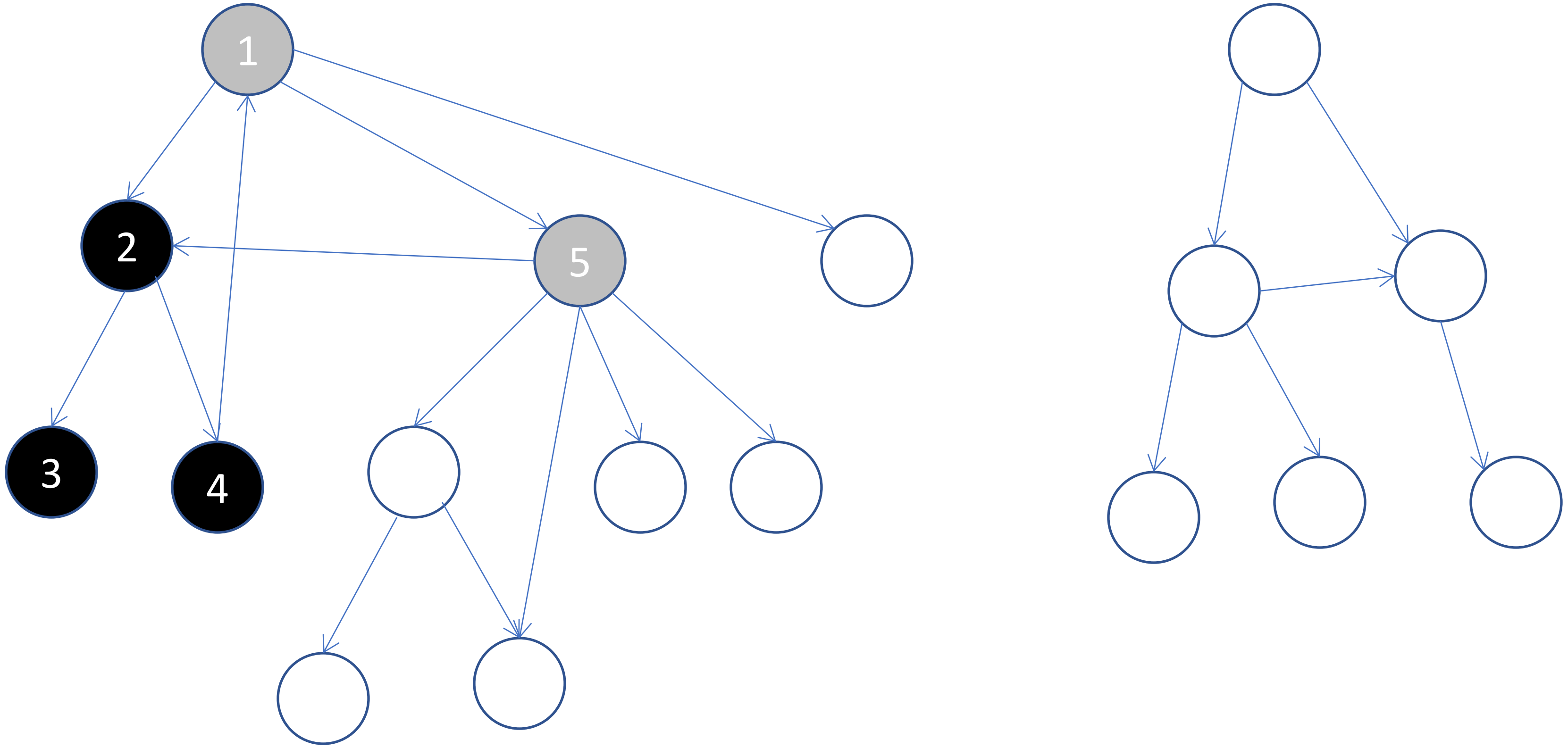
A DFS example (2)



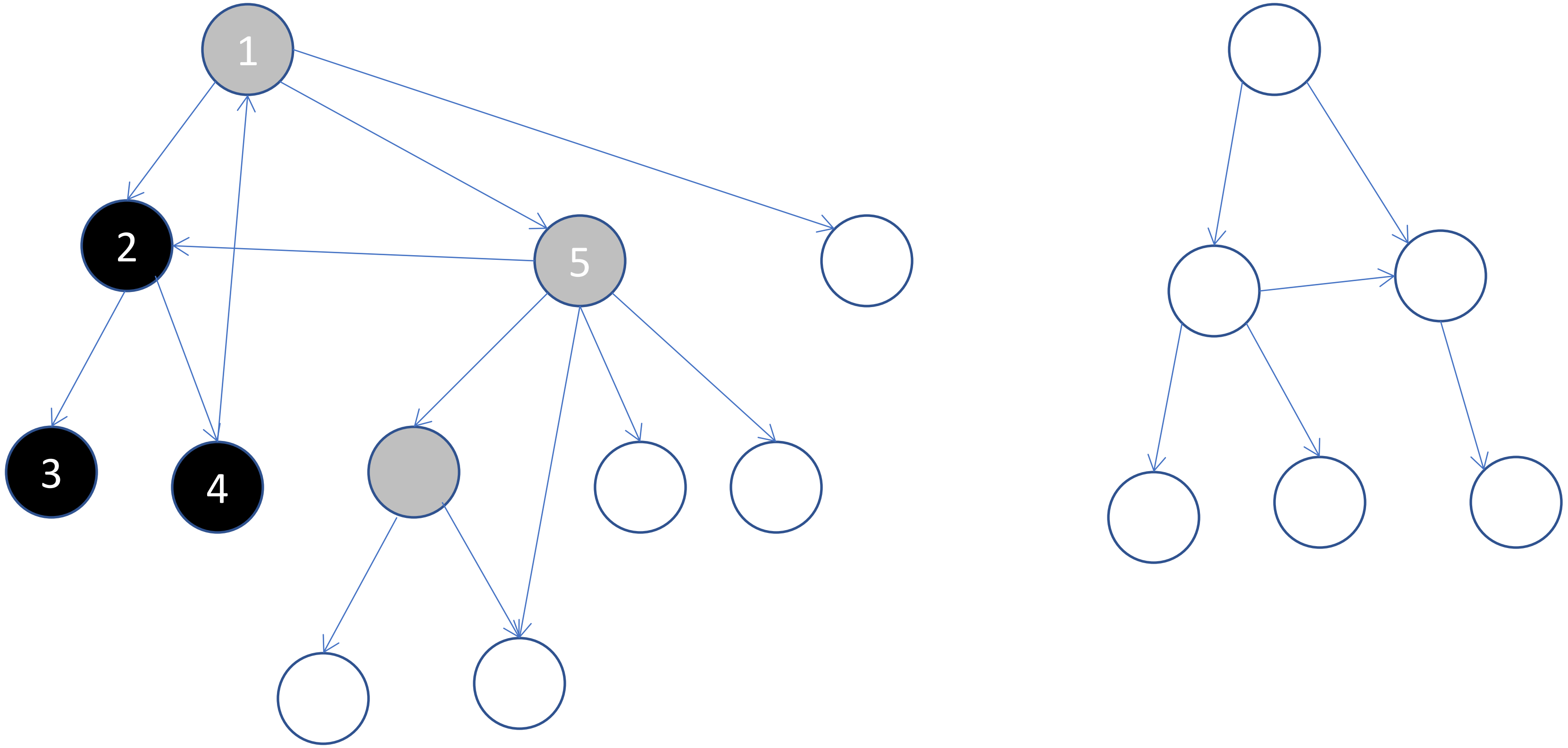
A DFS example (2)



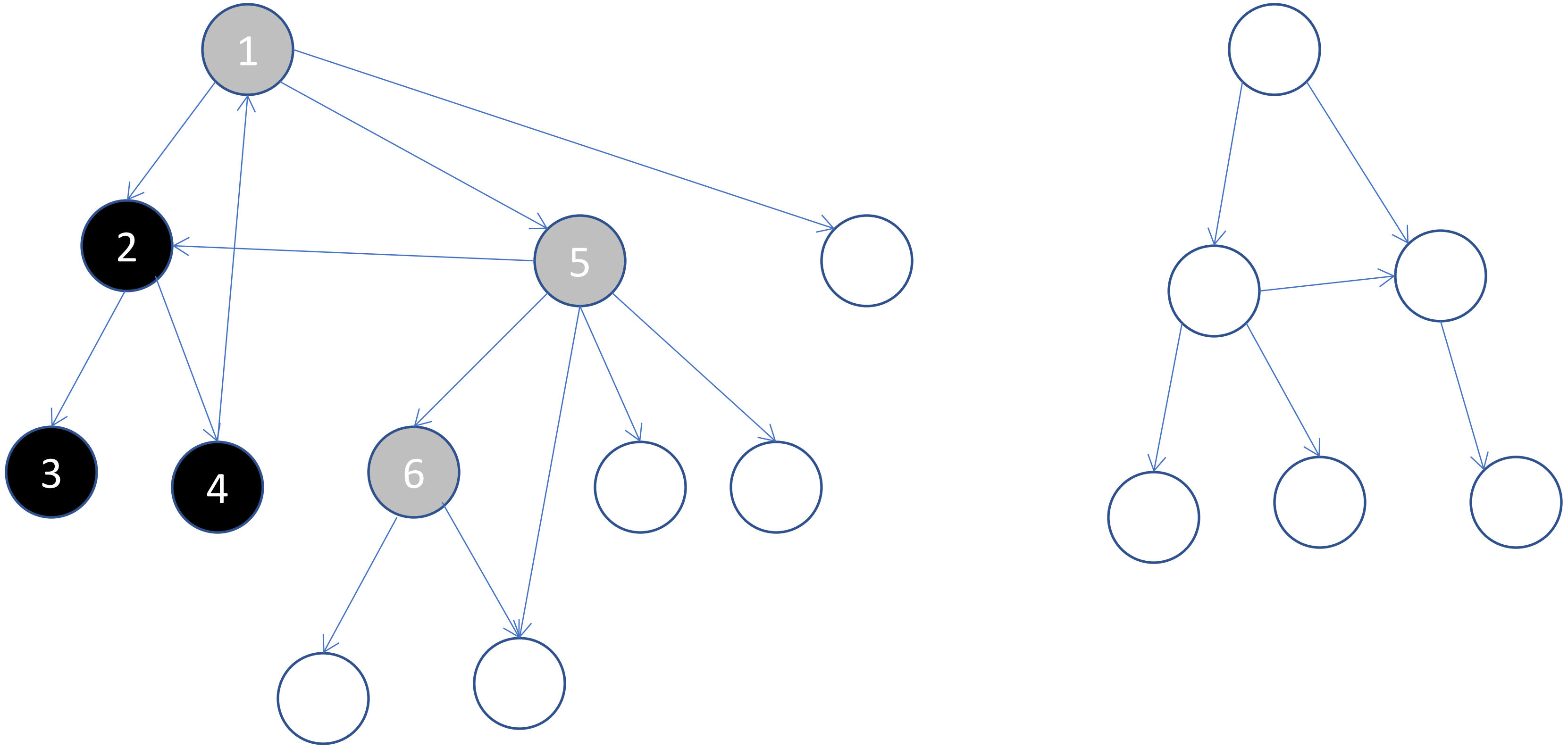
A DFS example (2)



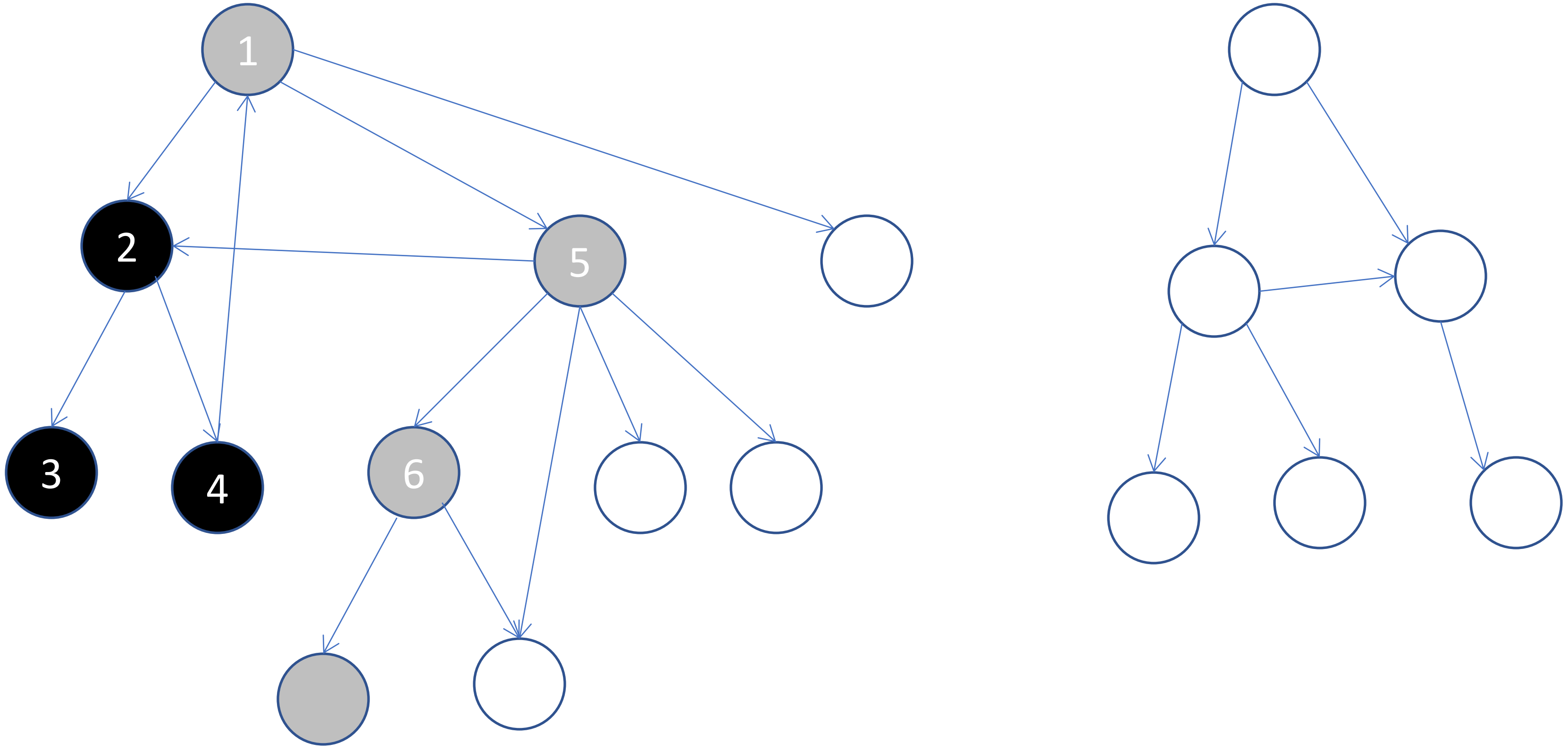
A DFS example (2)



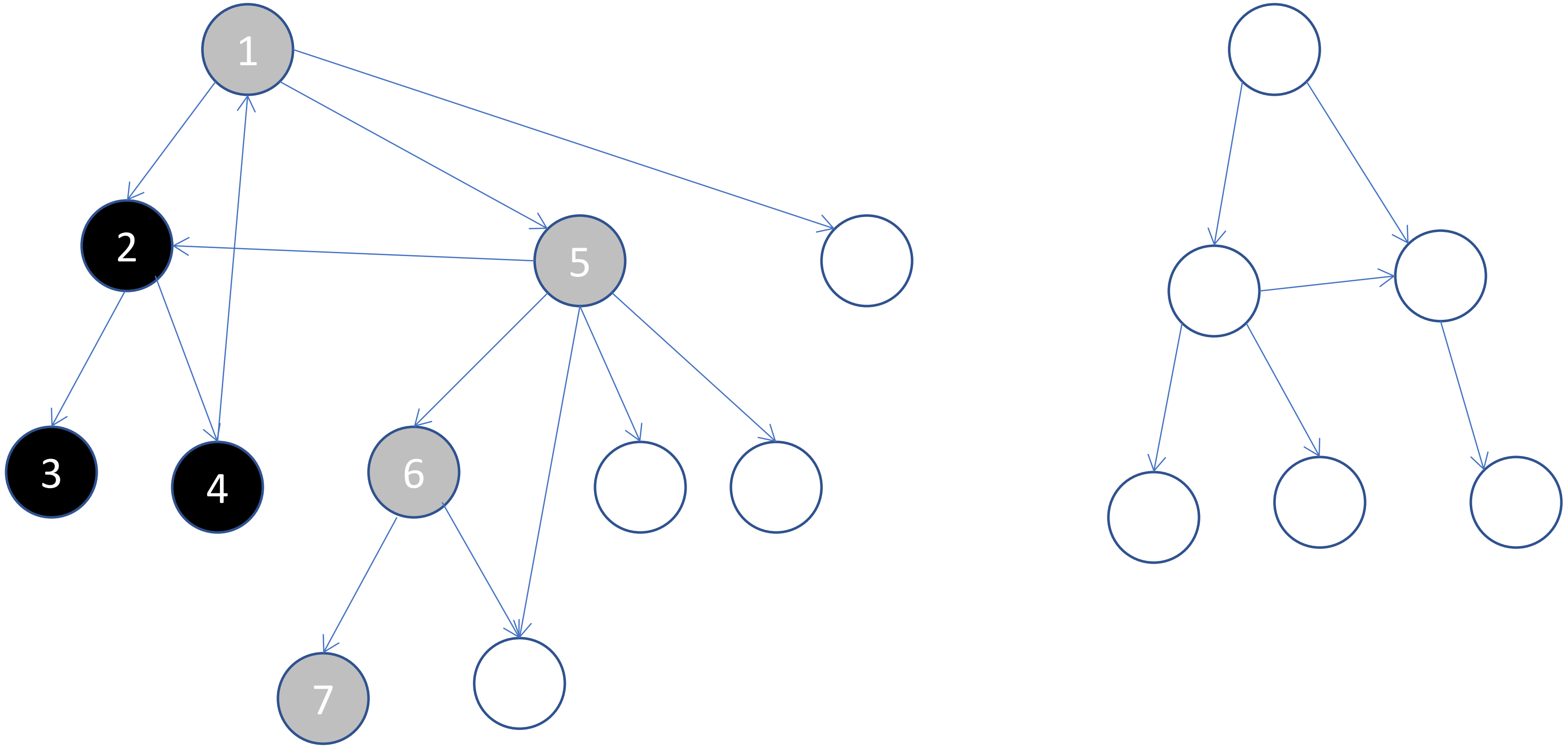
A DFS example (2)



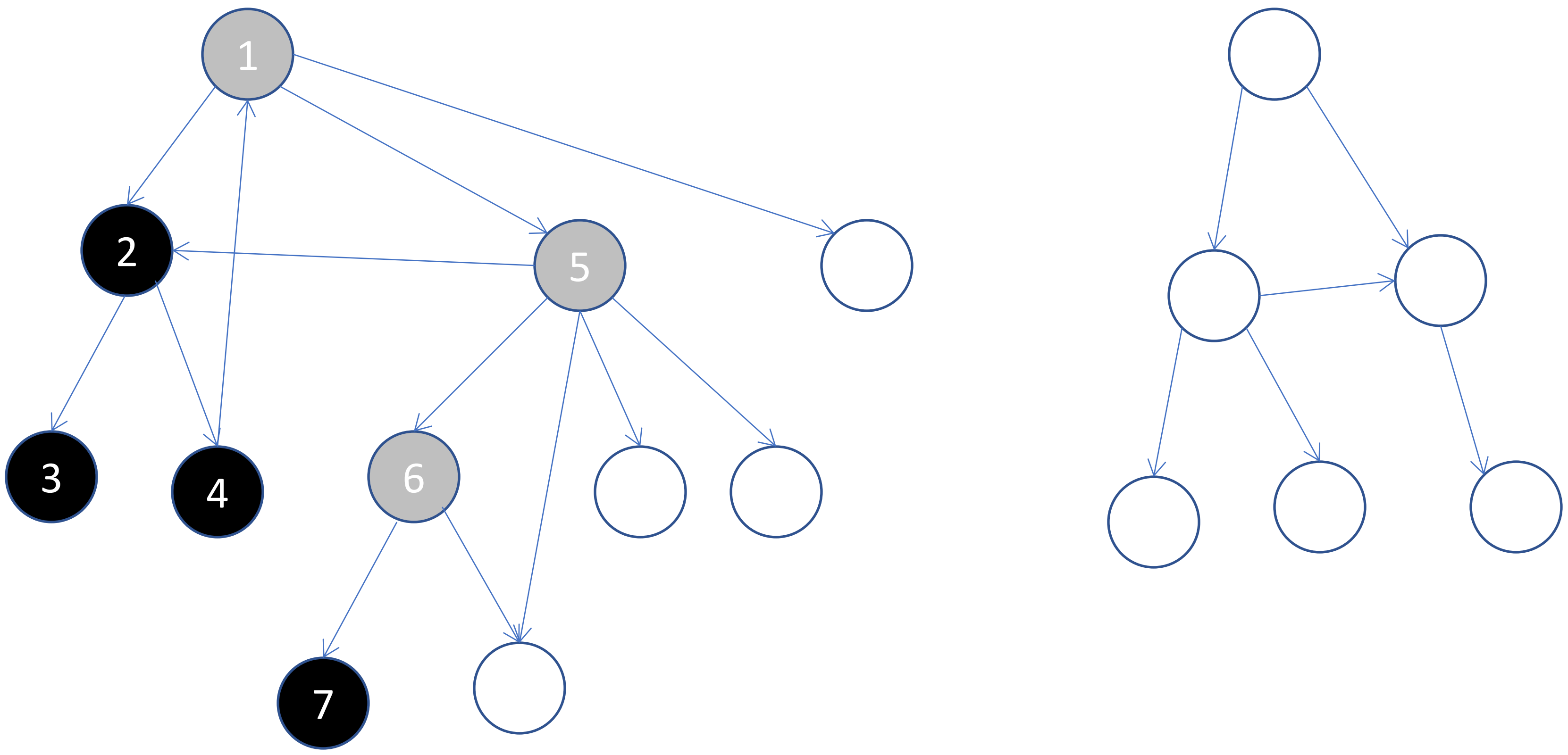
A DFS example (2)



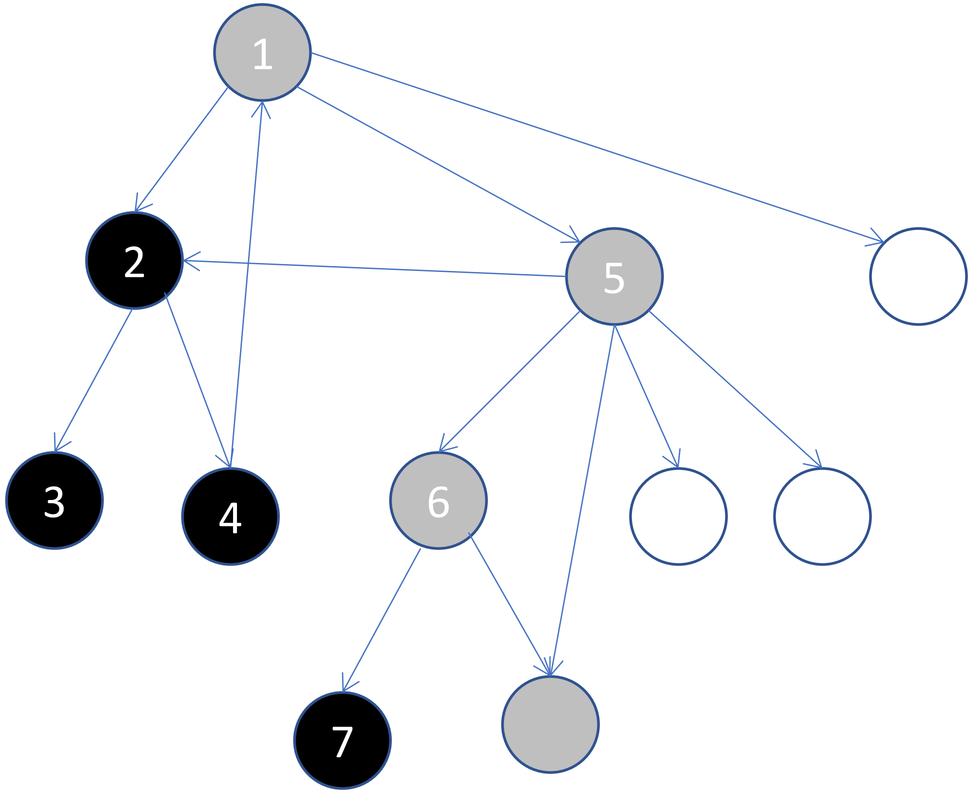
A DFS example (2)

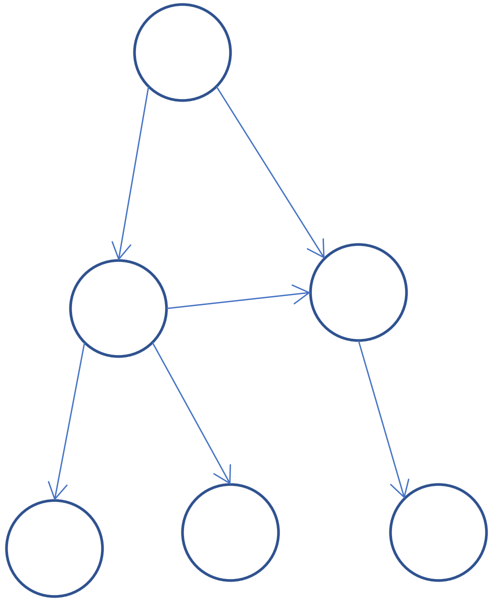


A DFS example (2)

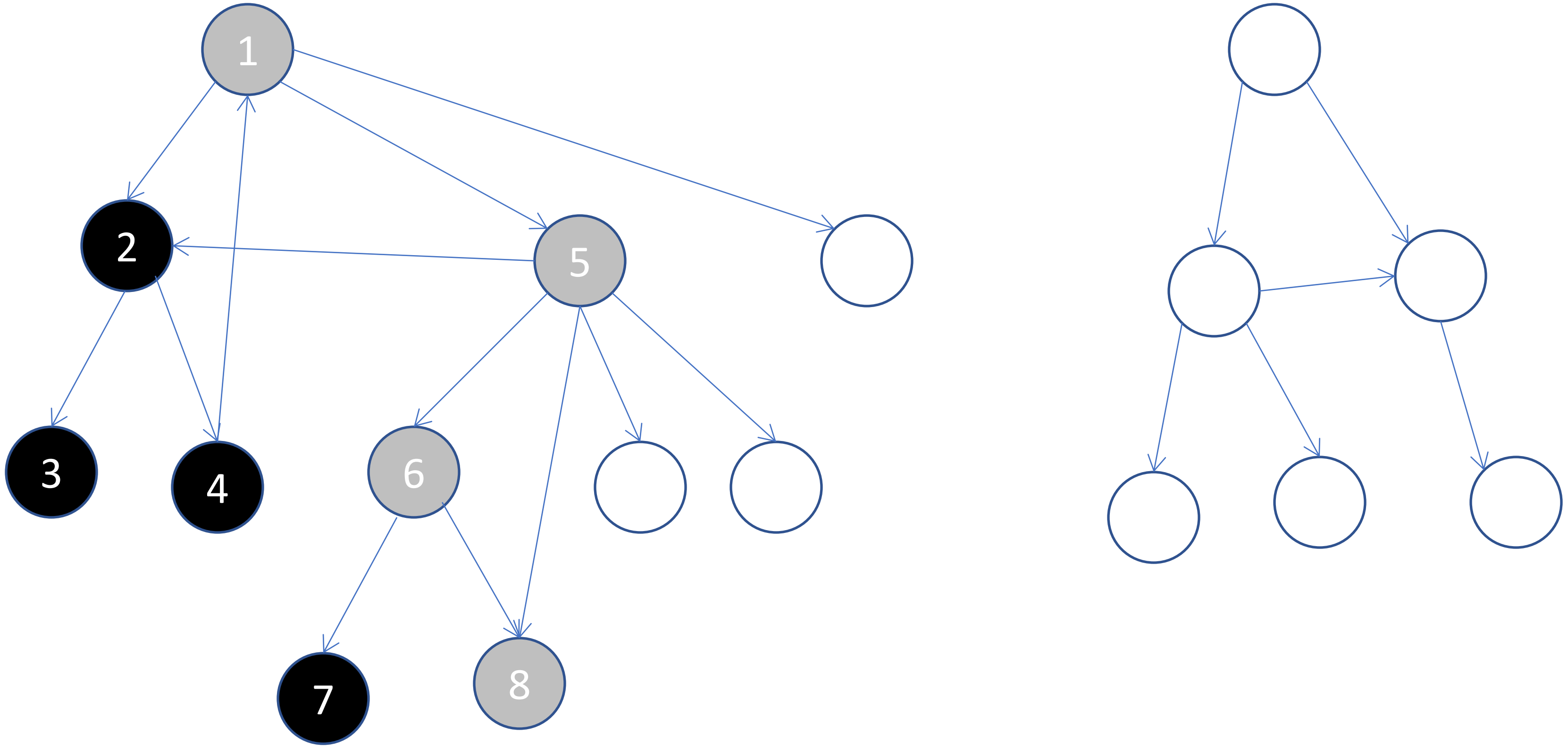


A DFS example (2)

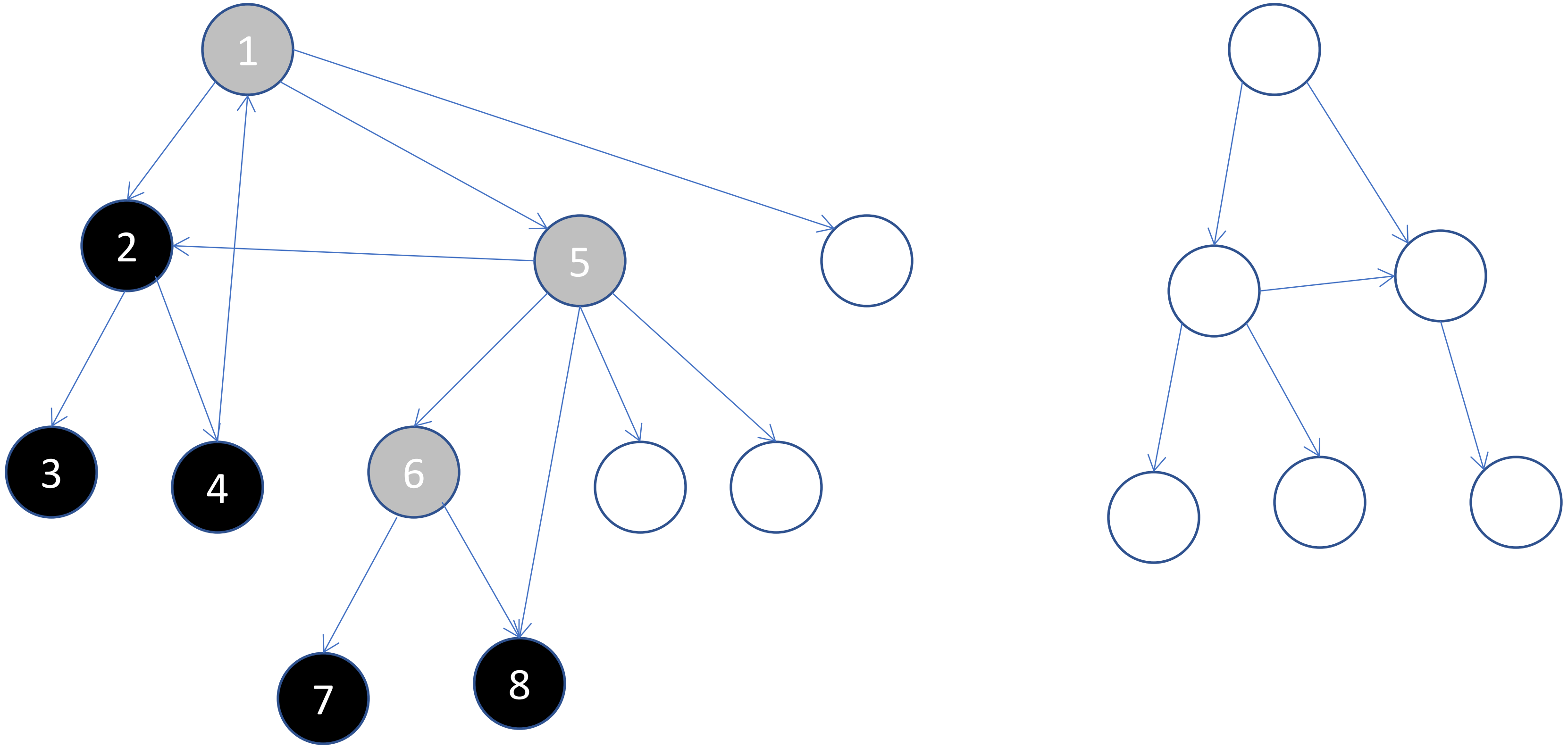




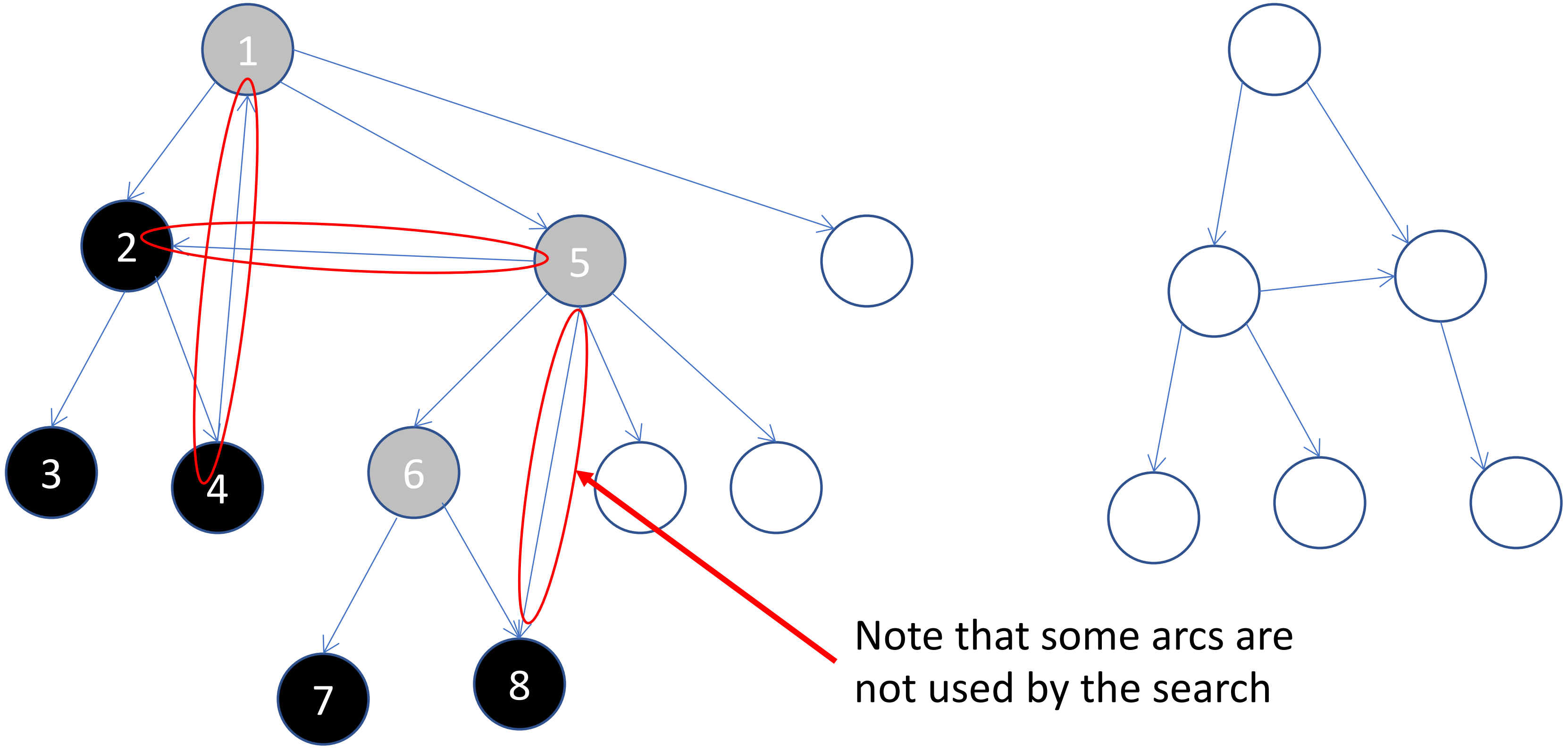
A DFS example (2)



A DFS example (2)

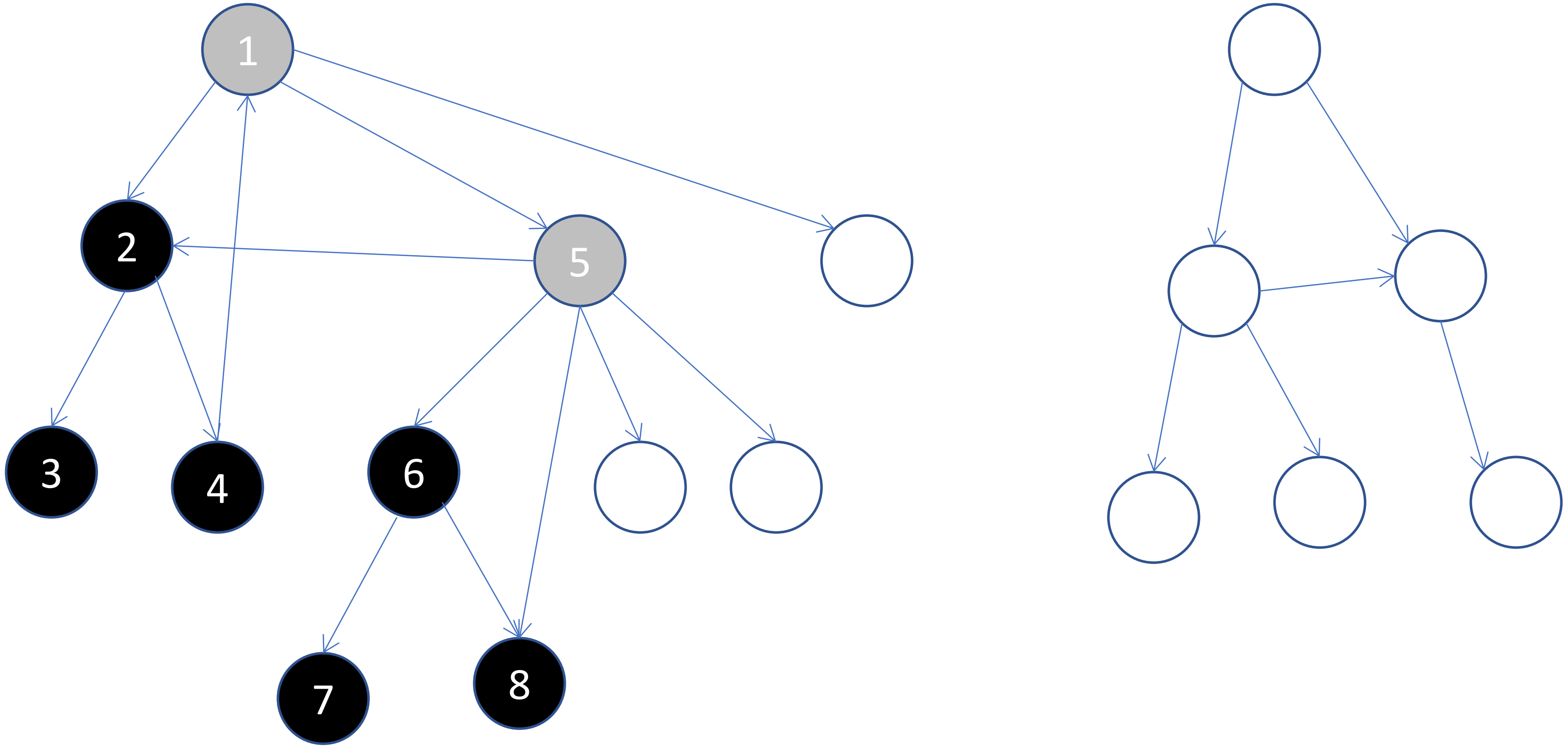


A DFS example (2)

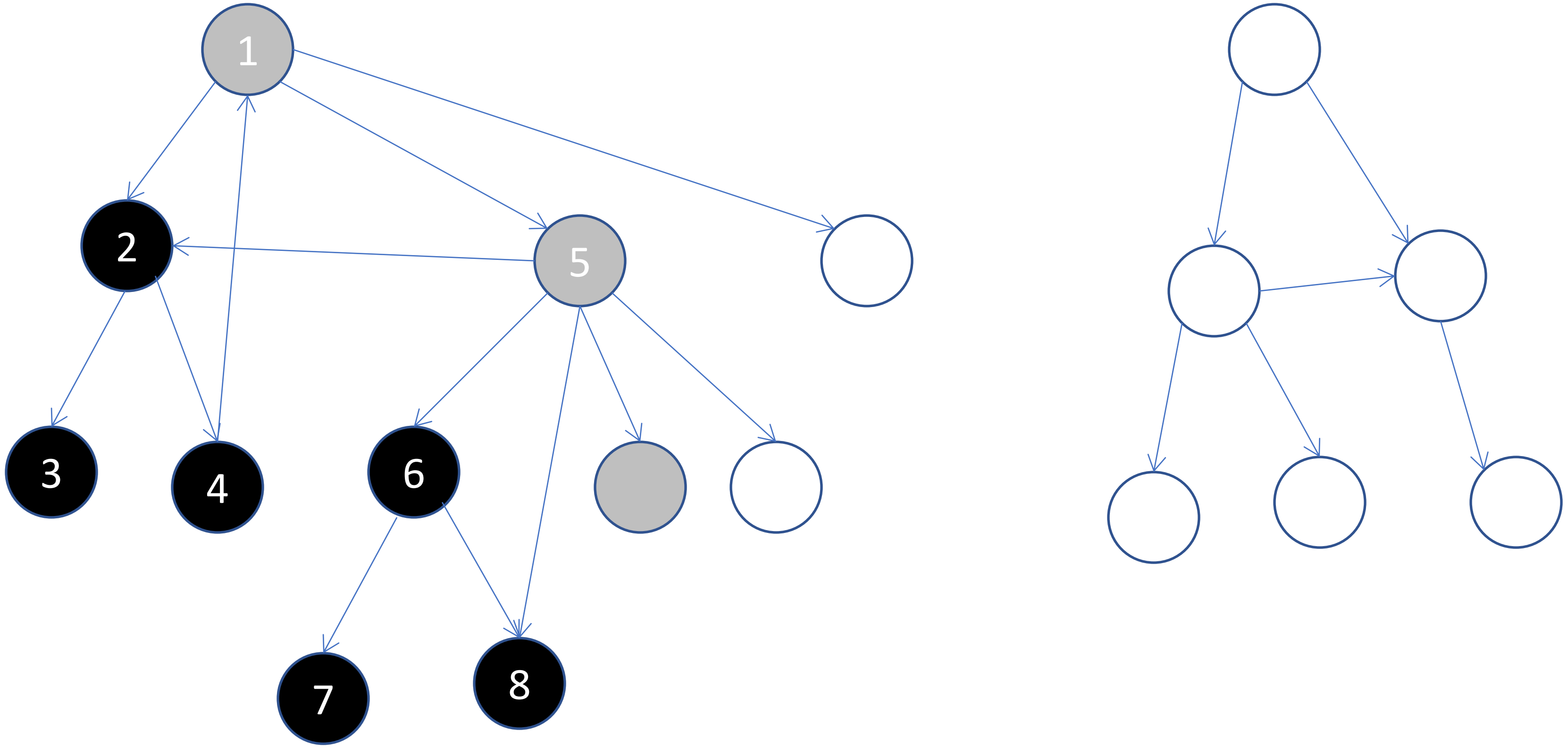


Note that some arcs are not used by the search

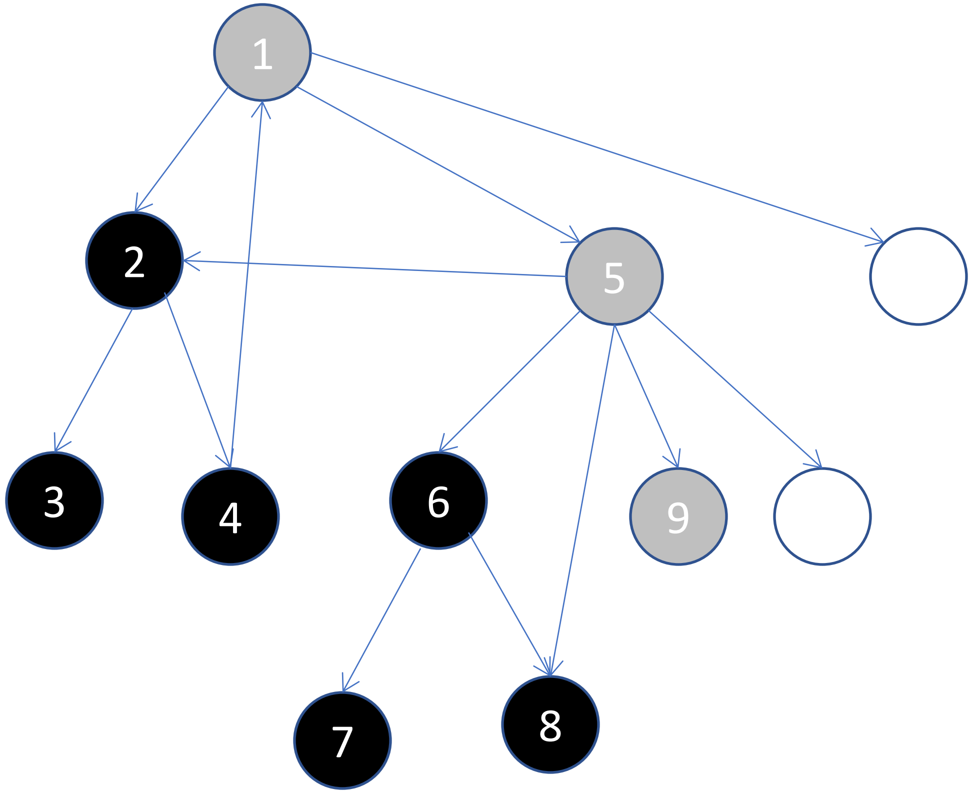
A DFS example (2)

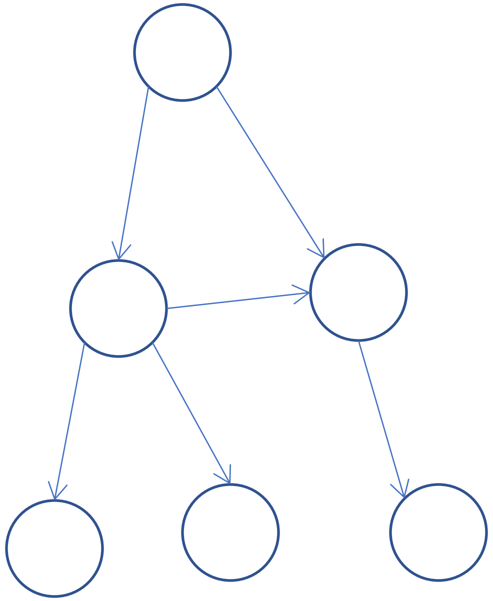


A DFS example (2)

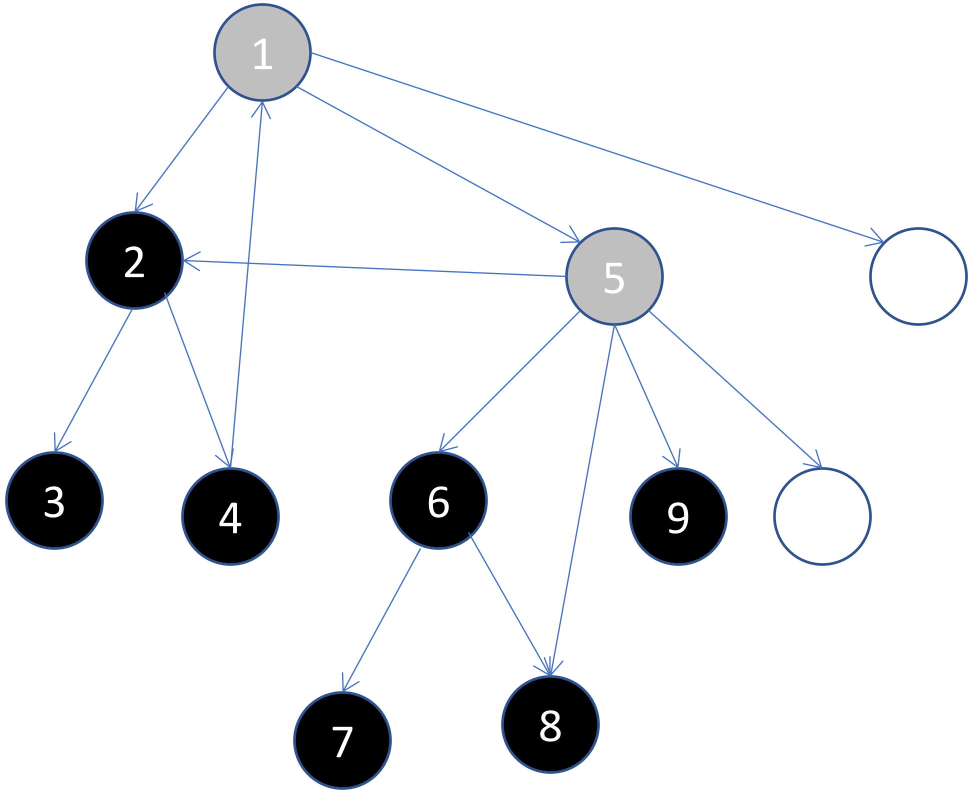


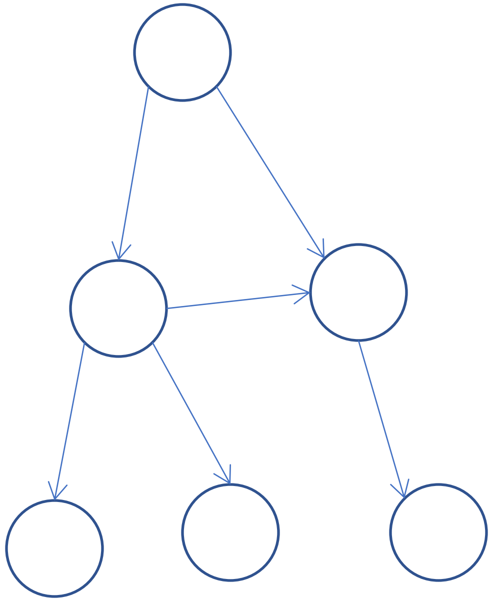
A DFS example (2)



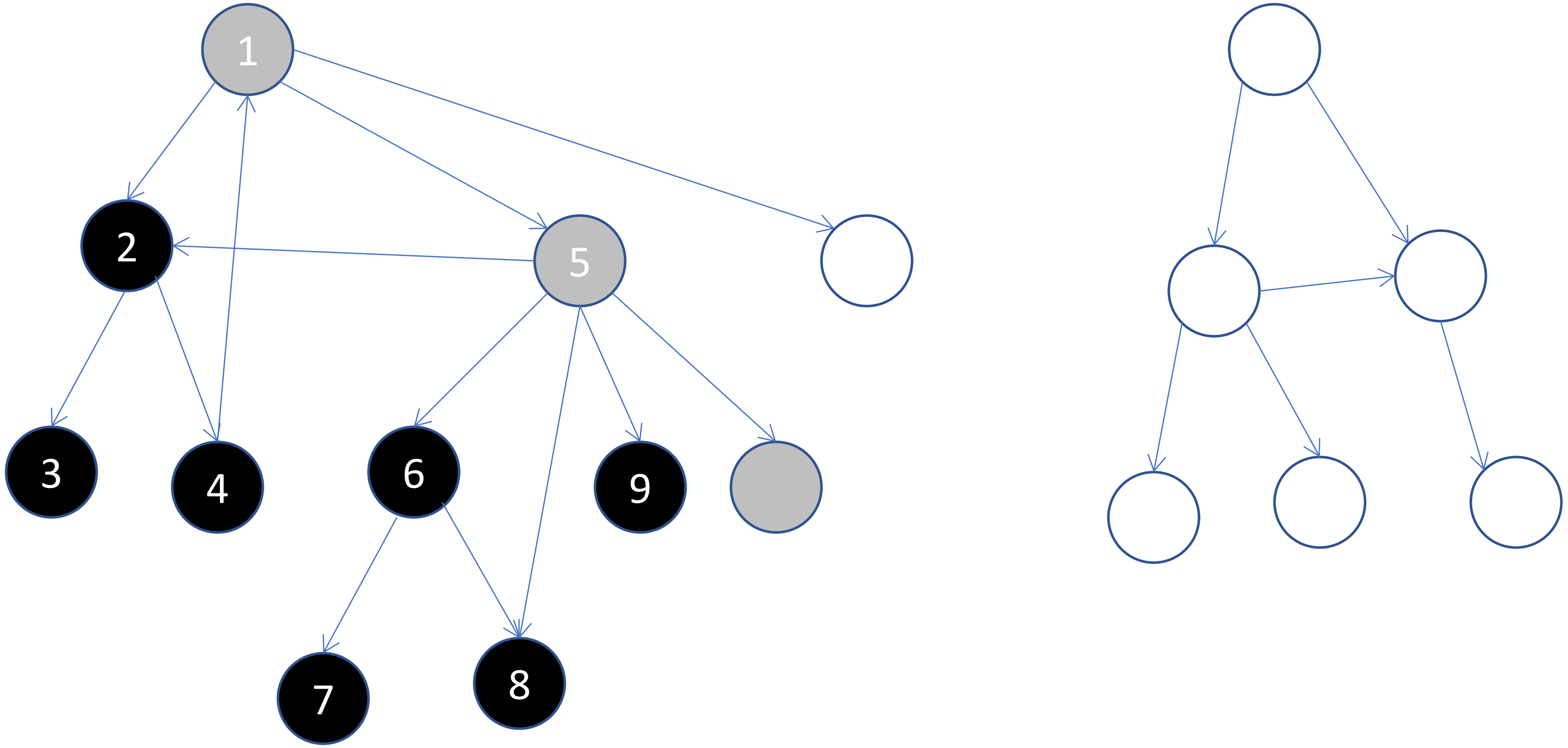


A DFS example (2)

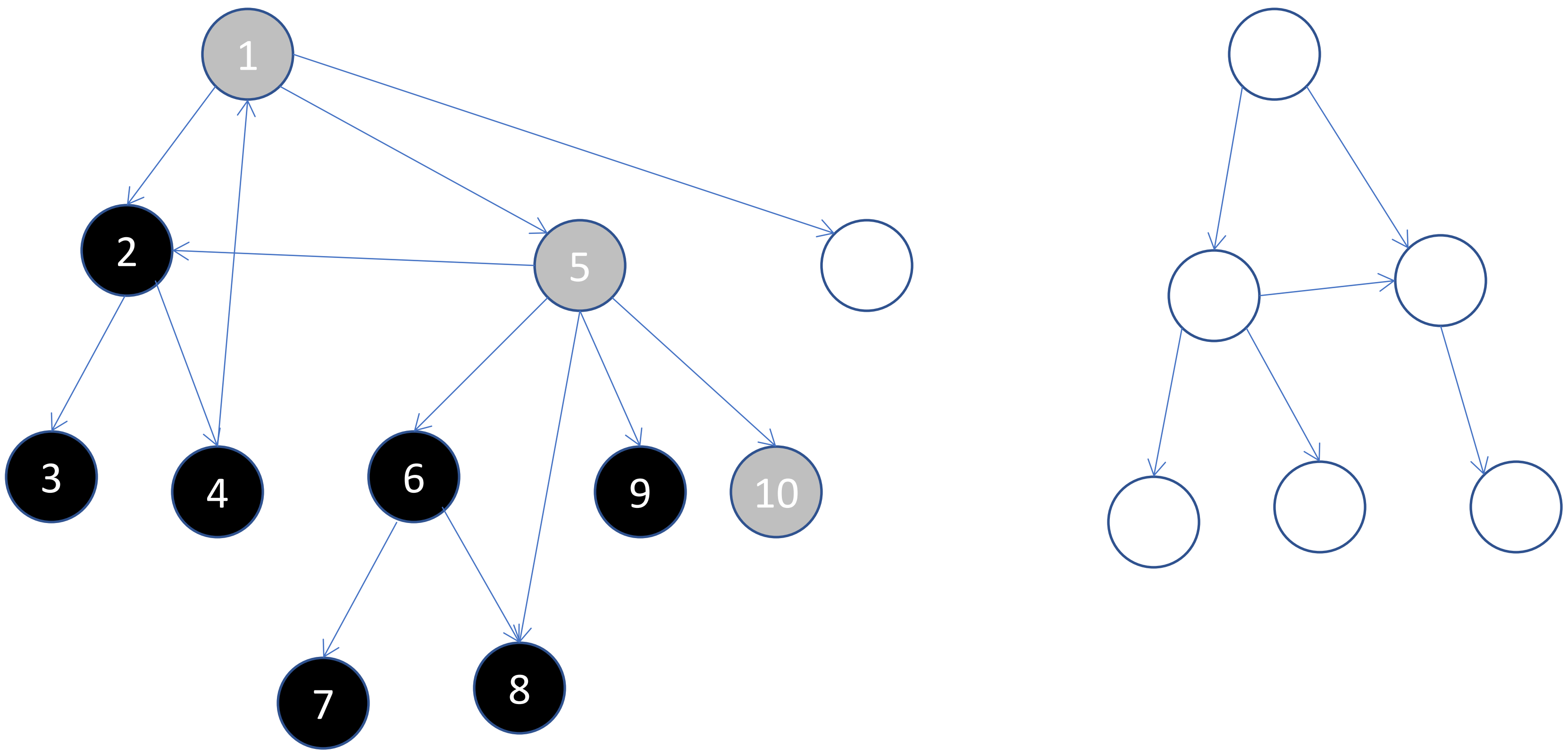




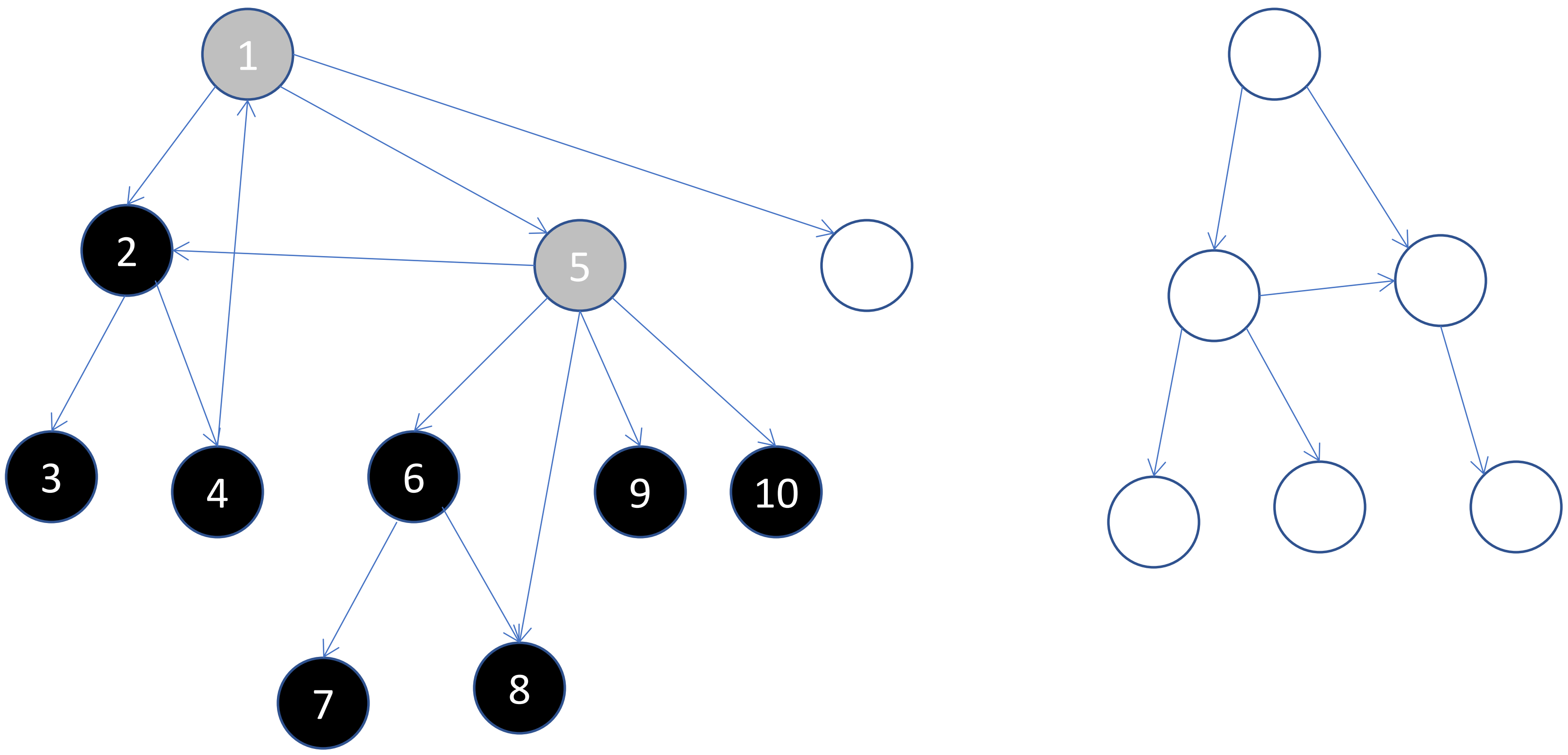
A DFS example (2)



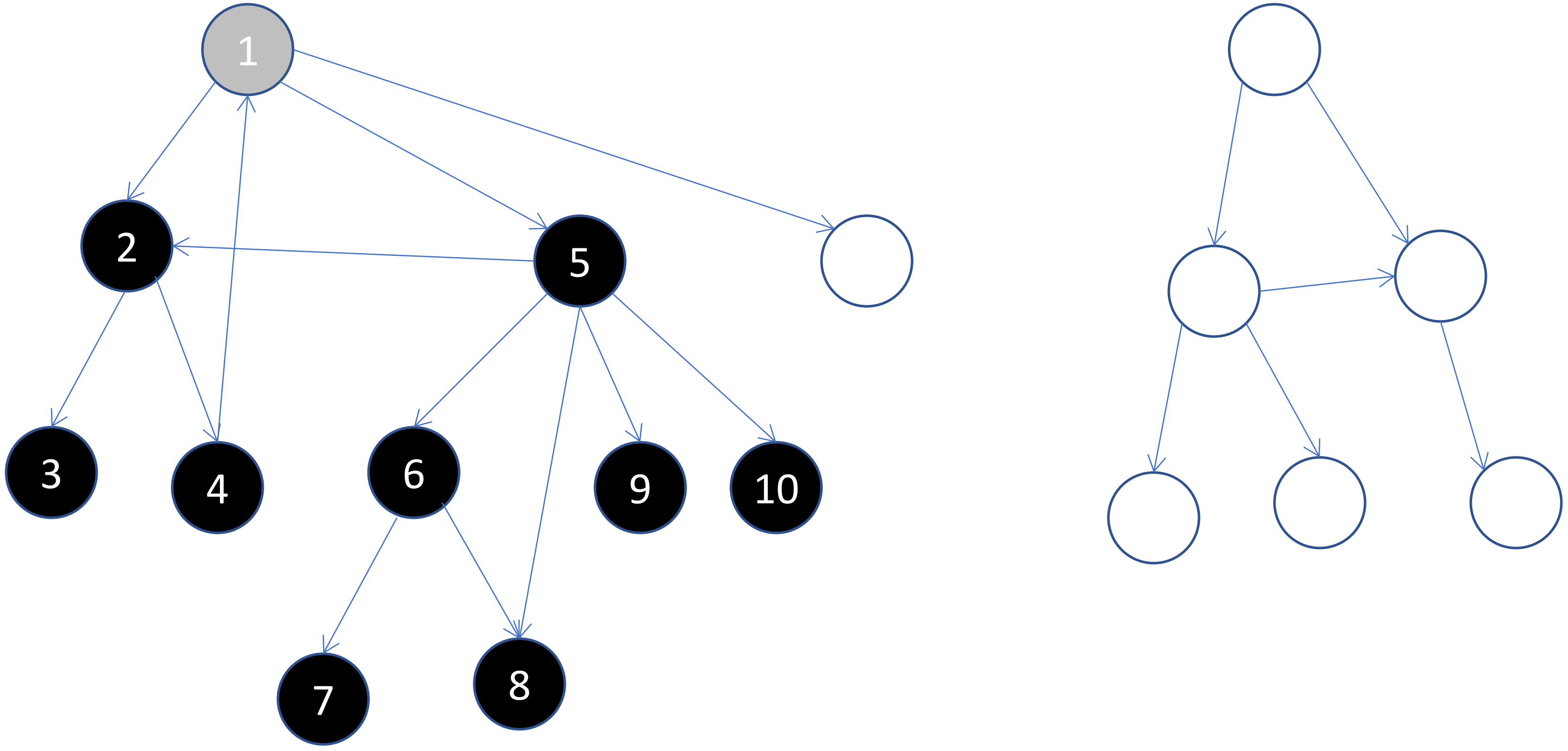
A DFS example (2)



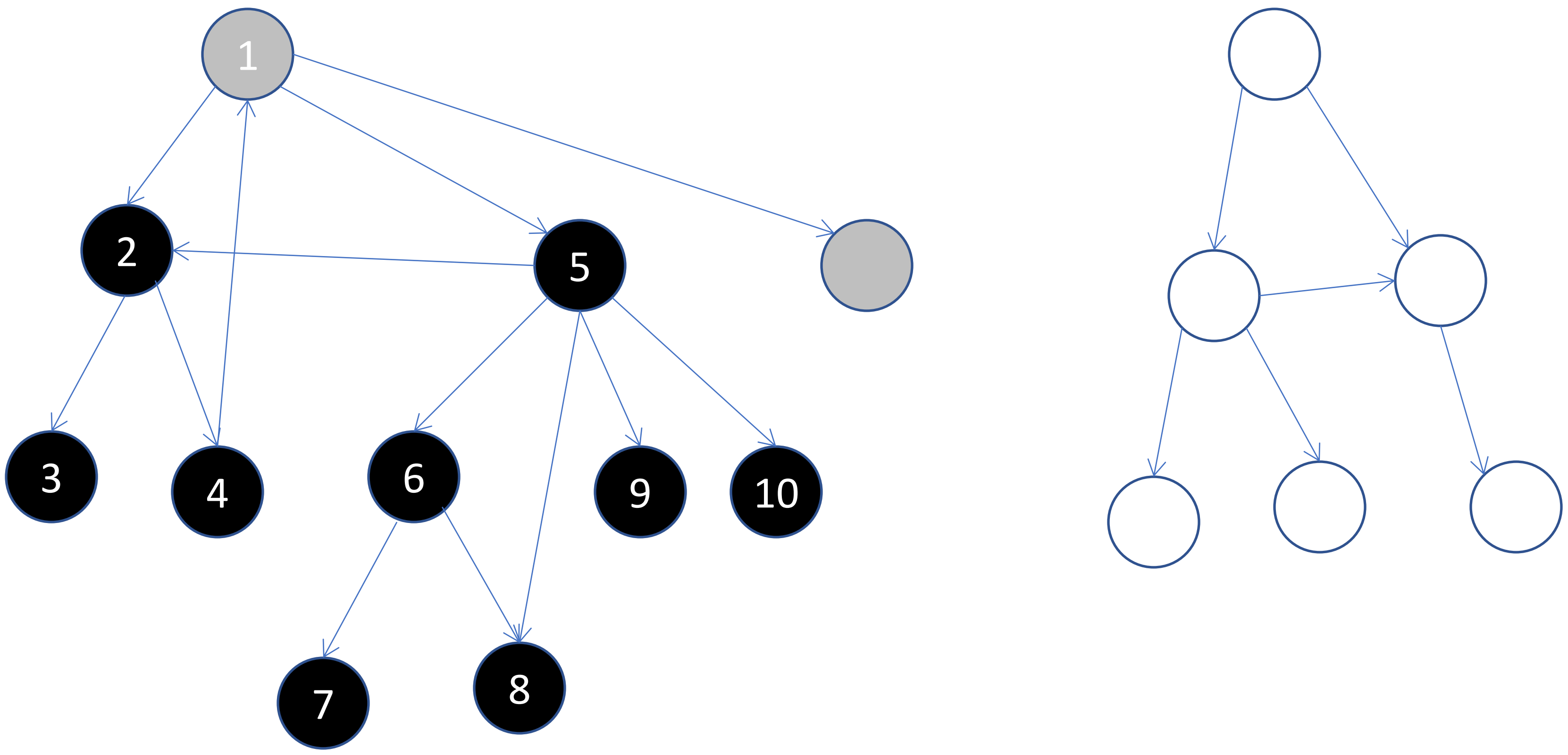
A DFS example (2)



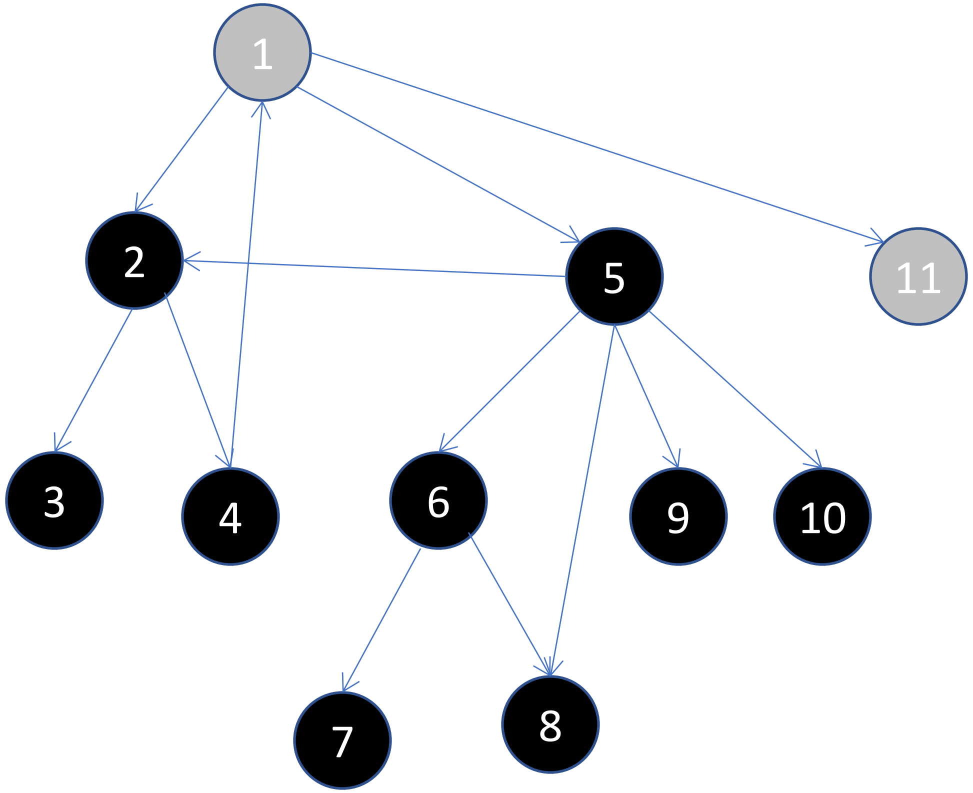
A DFS example (2)

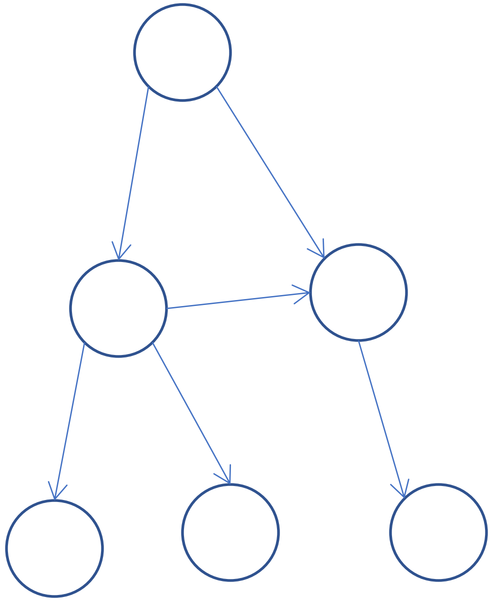


A DFS example (2)

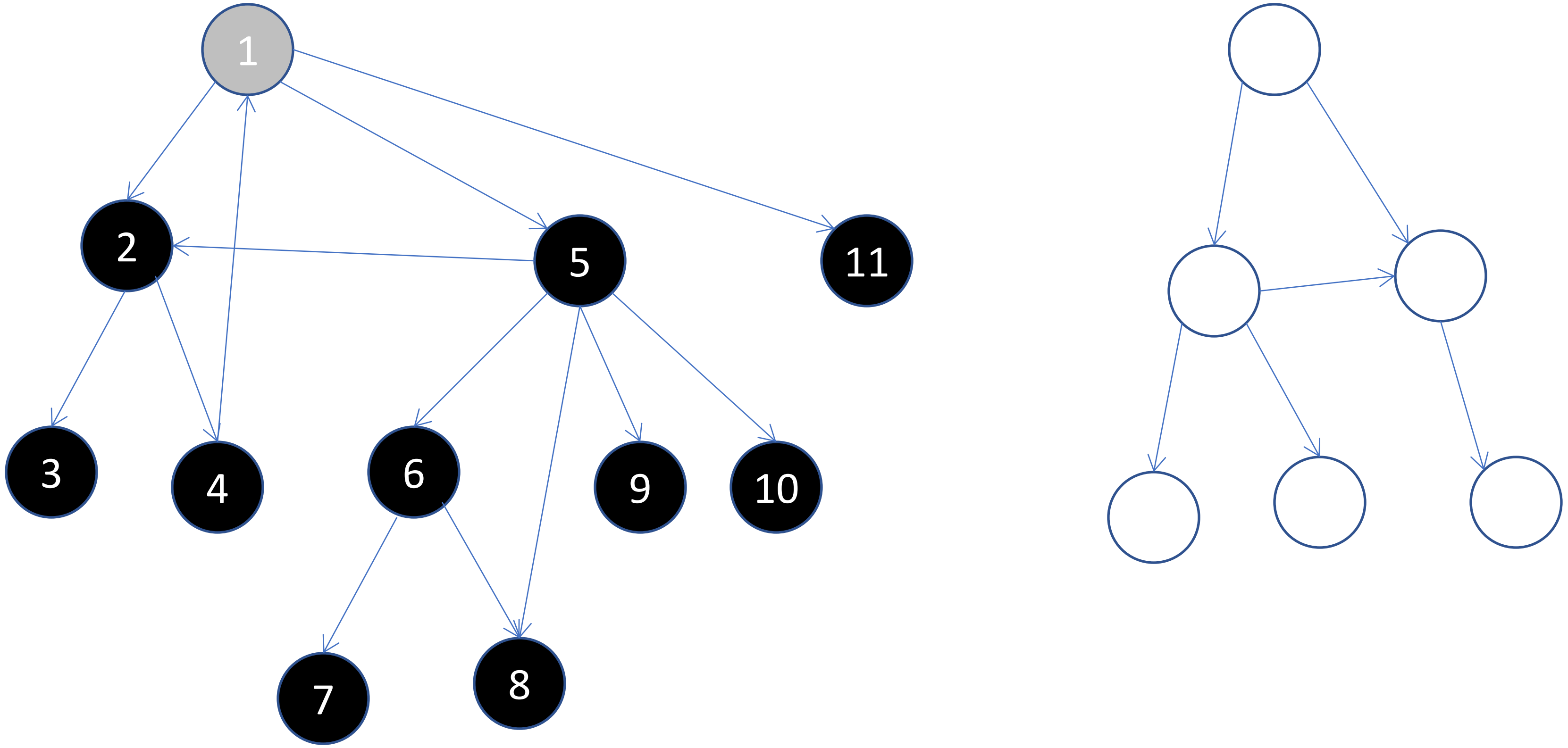


A DFS example (2)

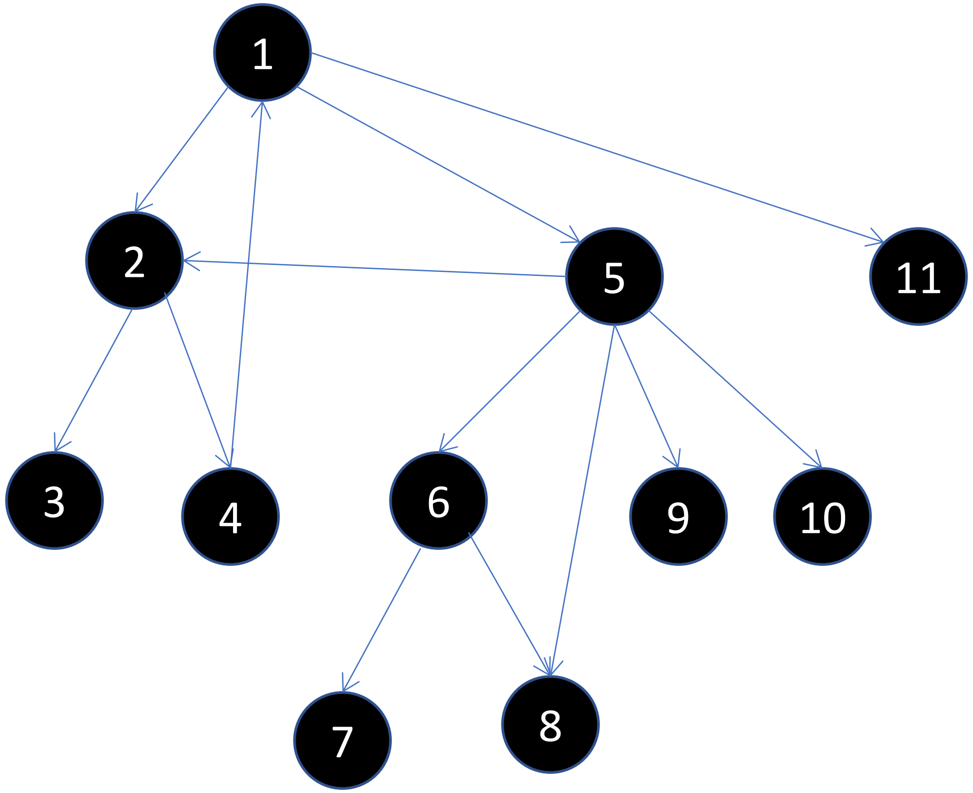


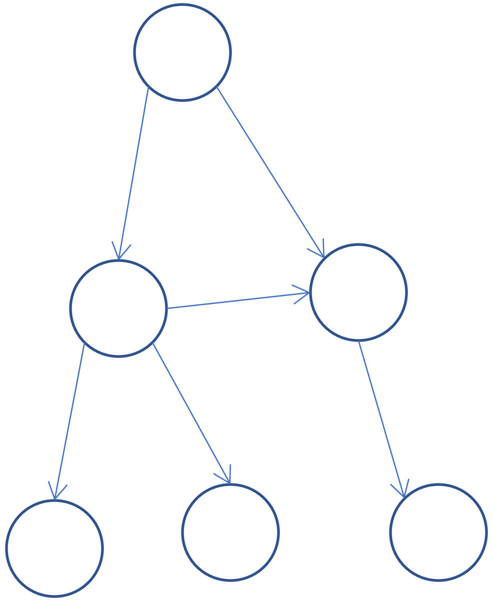


A DFS example (2)



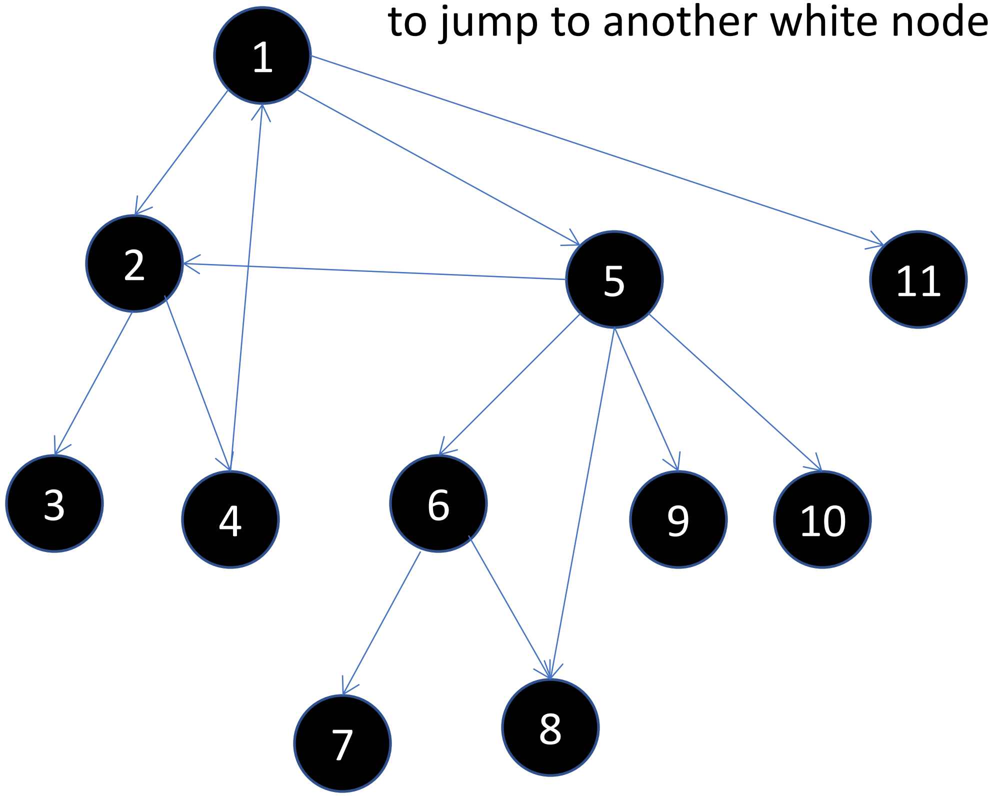
A DFS example (2)

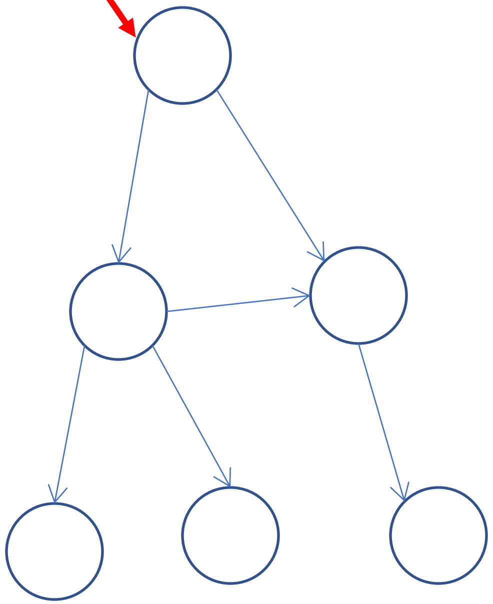




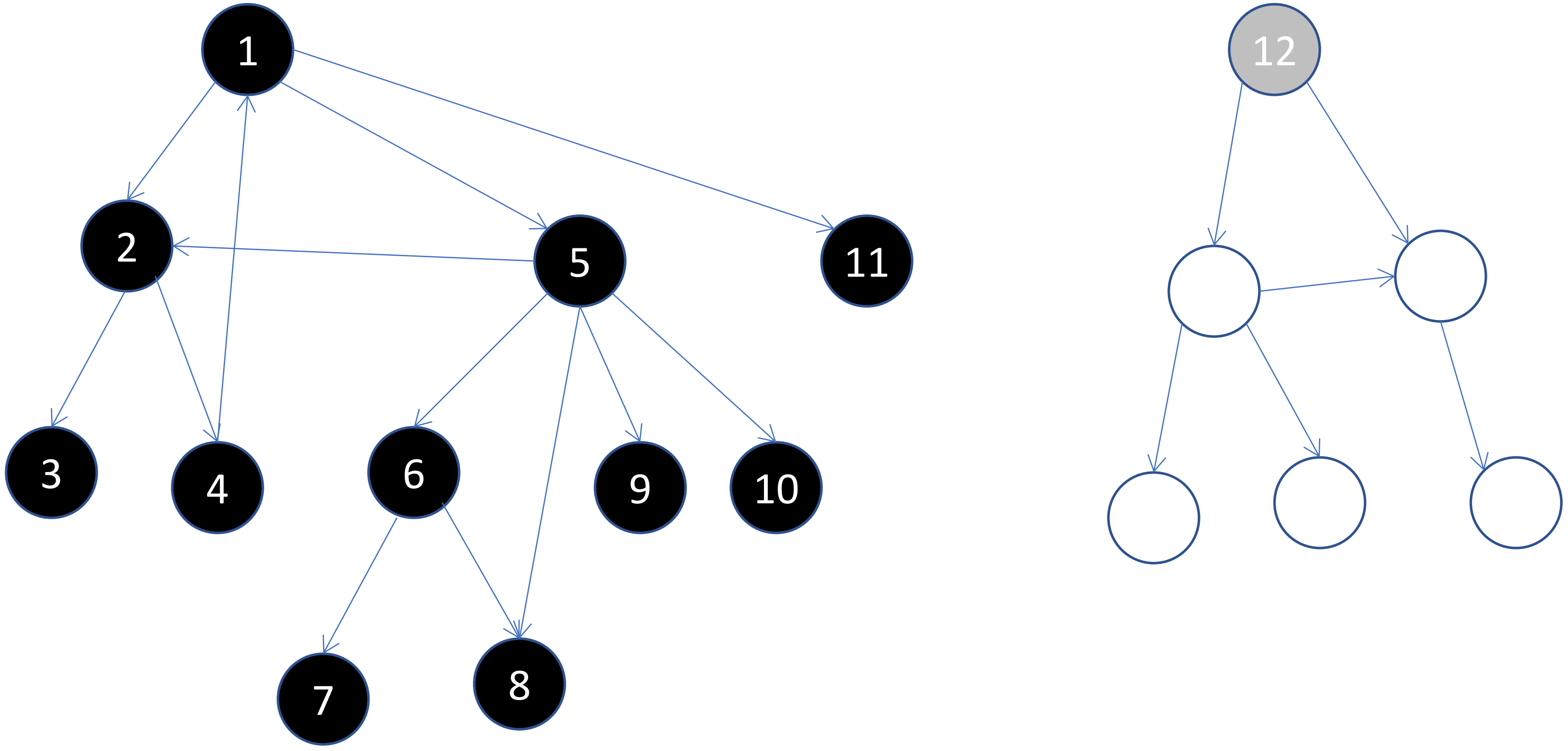
A DFS example (2)

Now we are done with the inner loop starting from node 1. The outer loop allows to jump to another white node

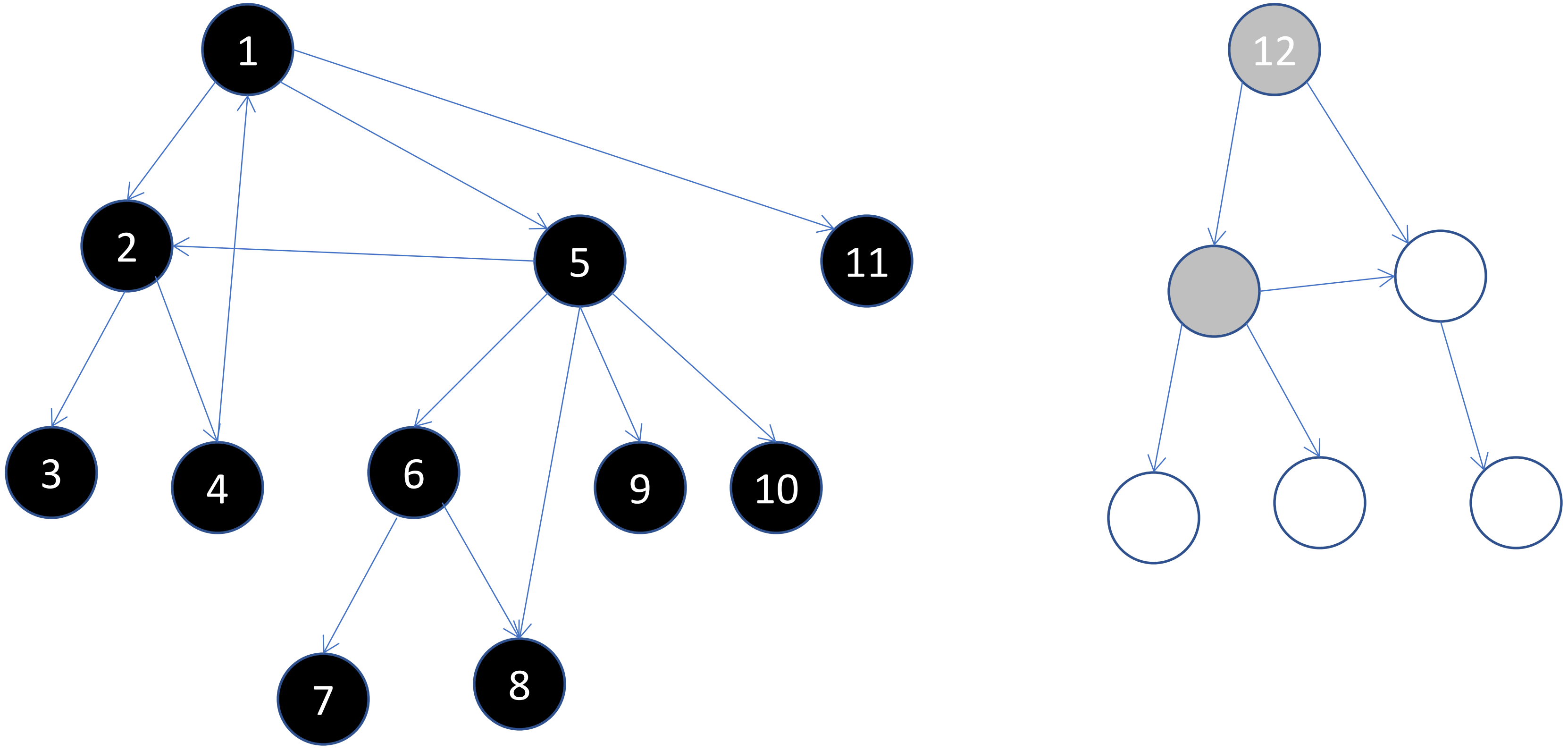




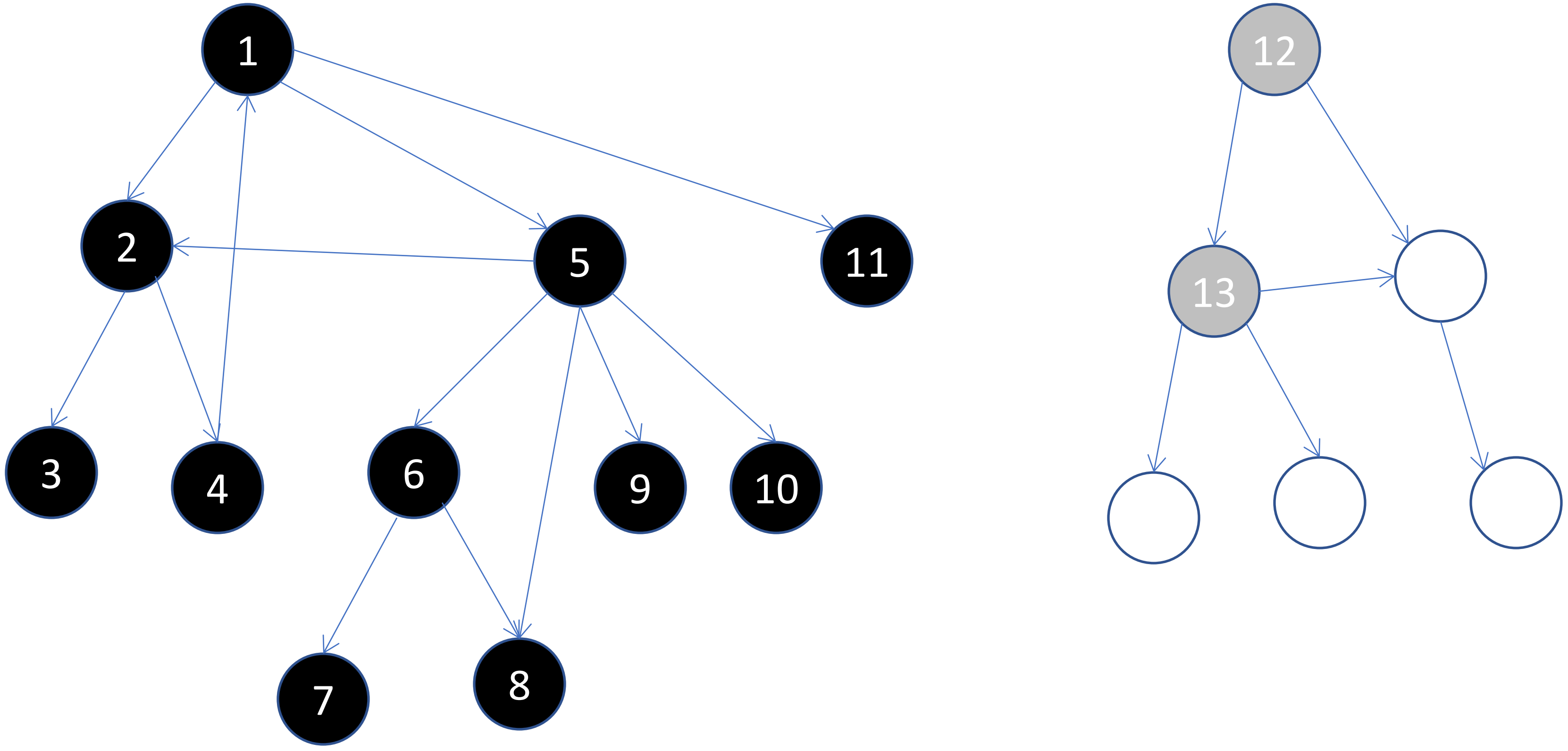
A DFS example (2)



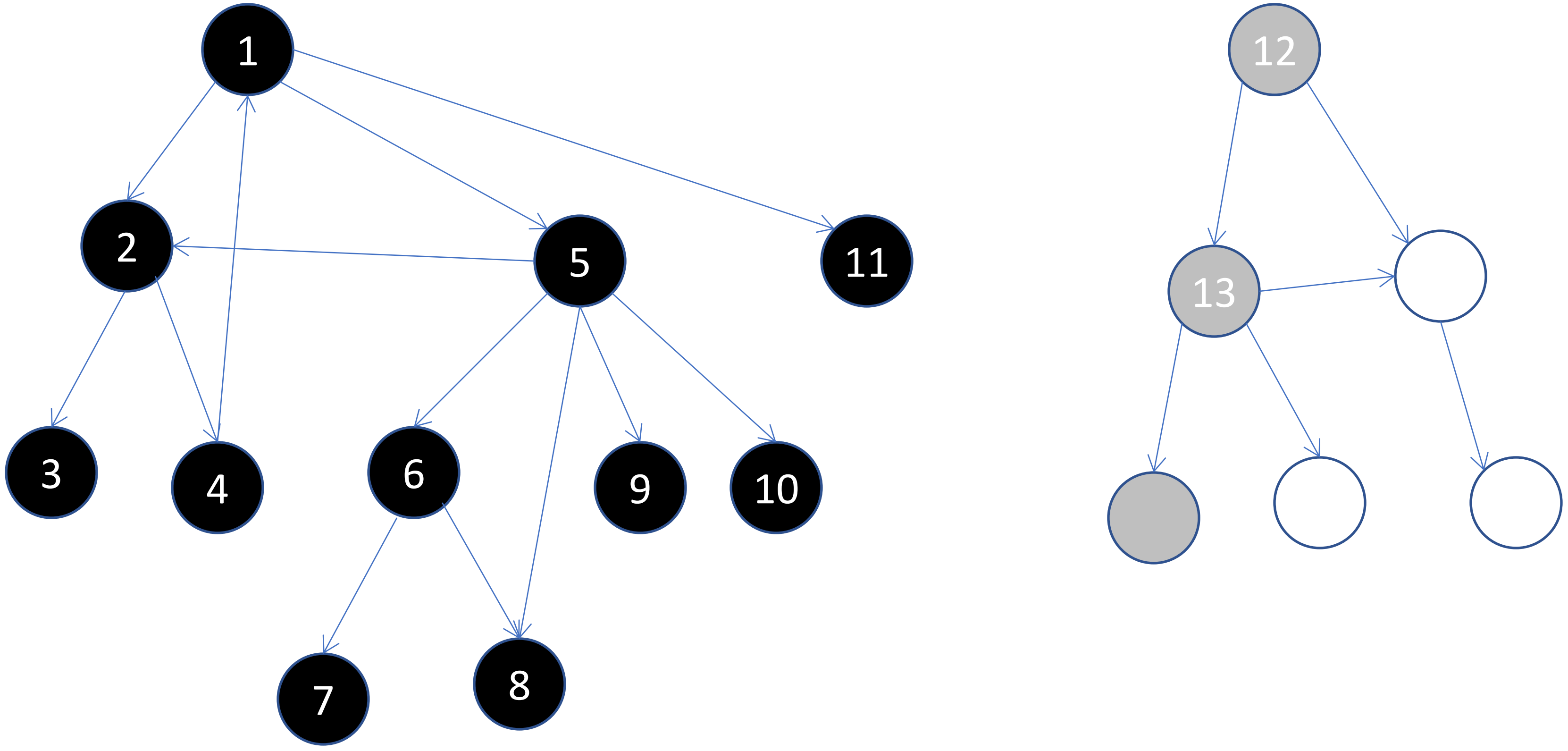
A DFS example (2)



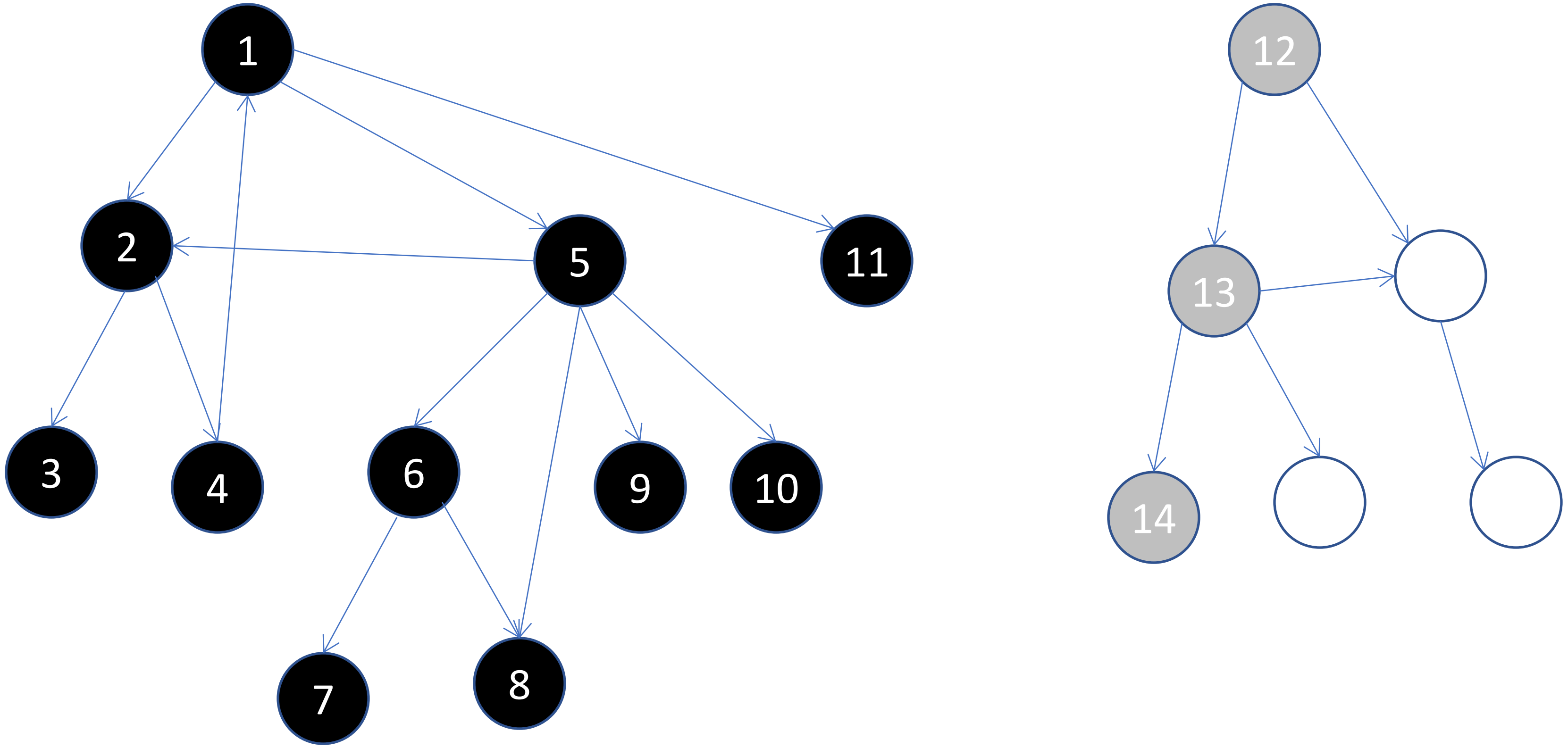
A DFS example (2)



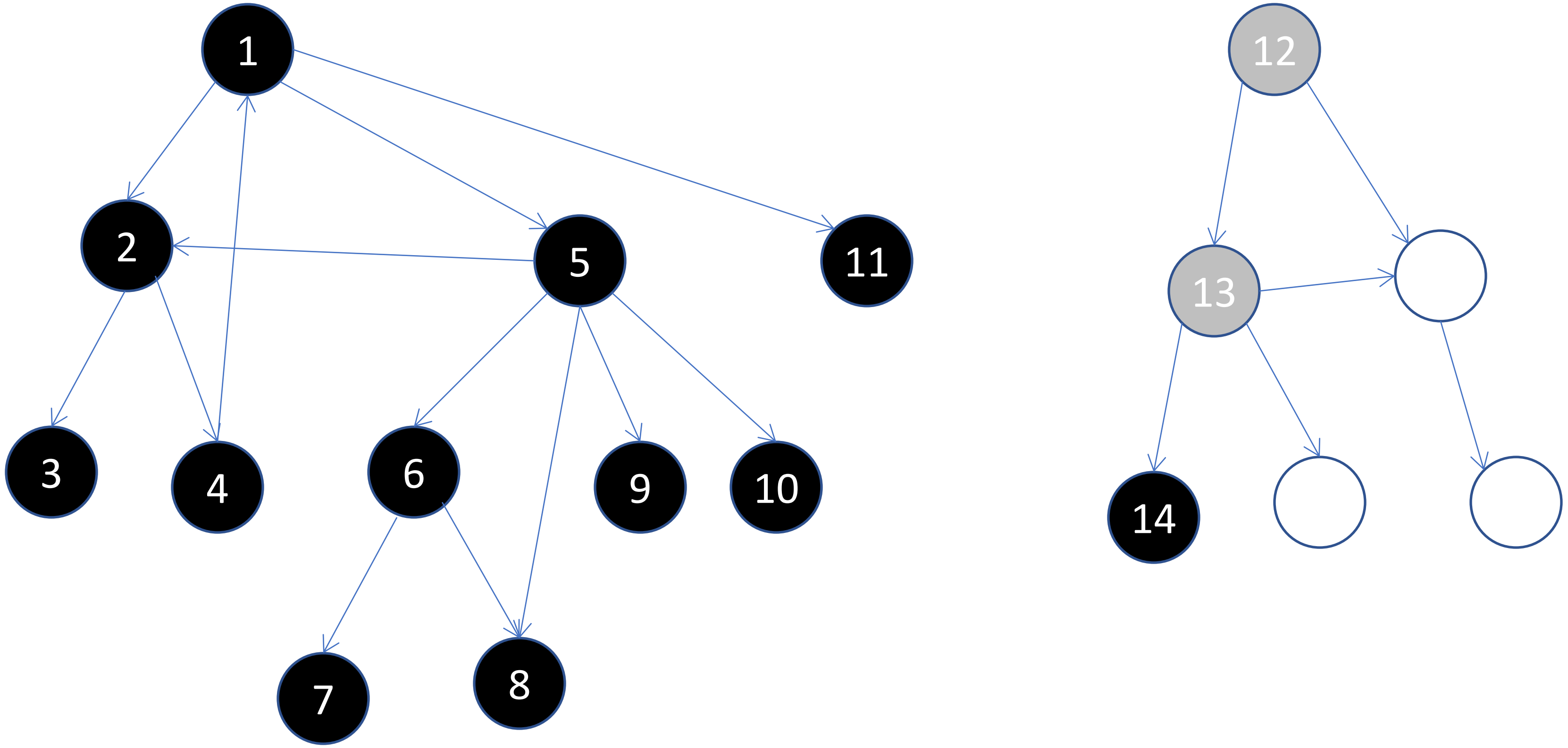
A DFS example (2)



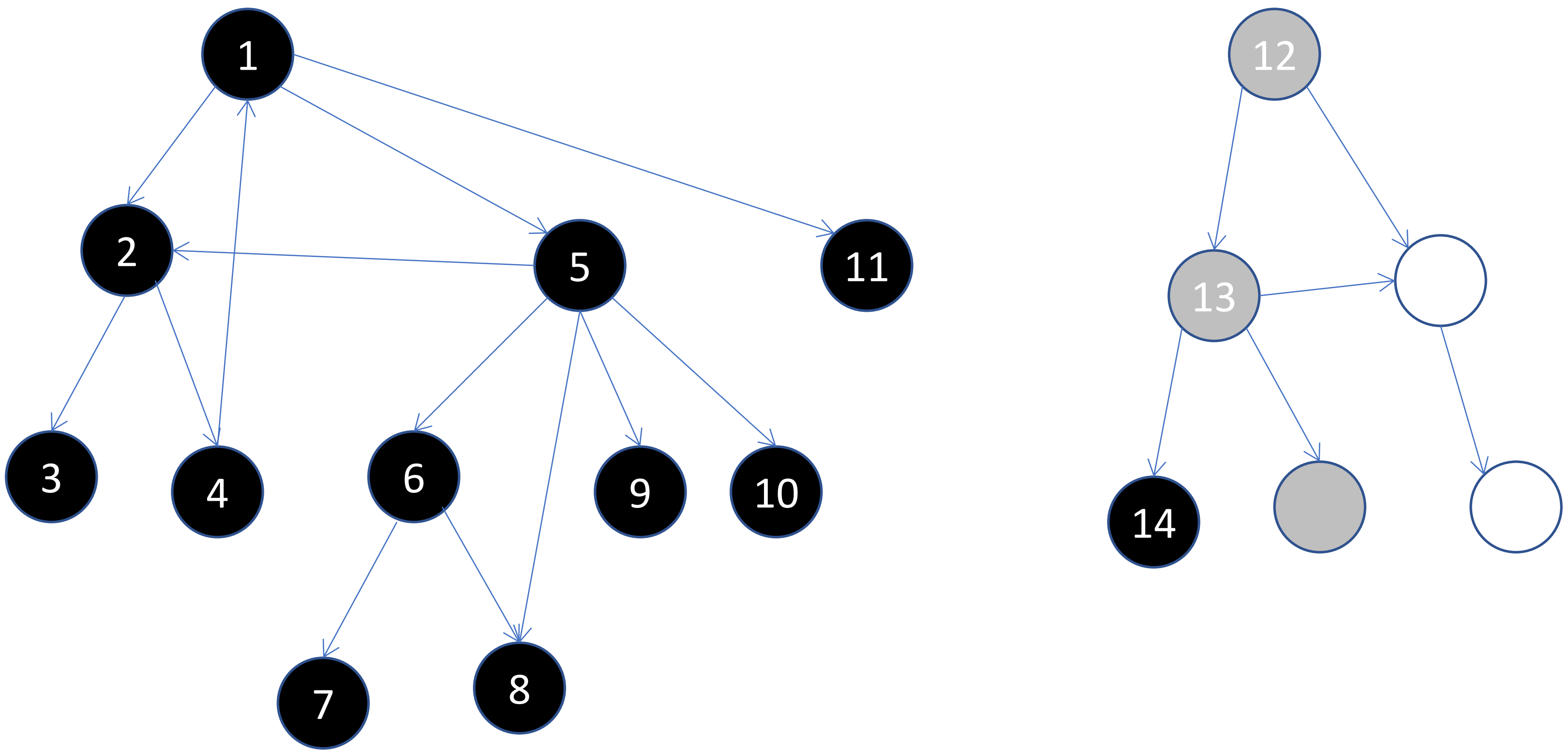
A DFS example (2)



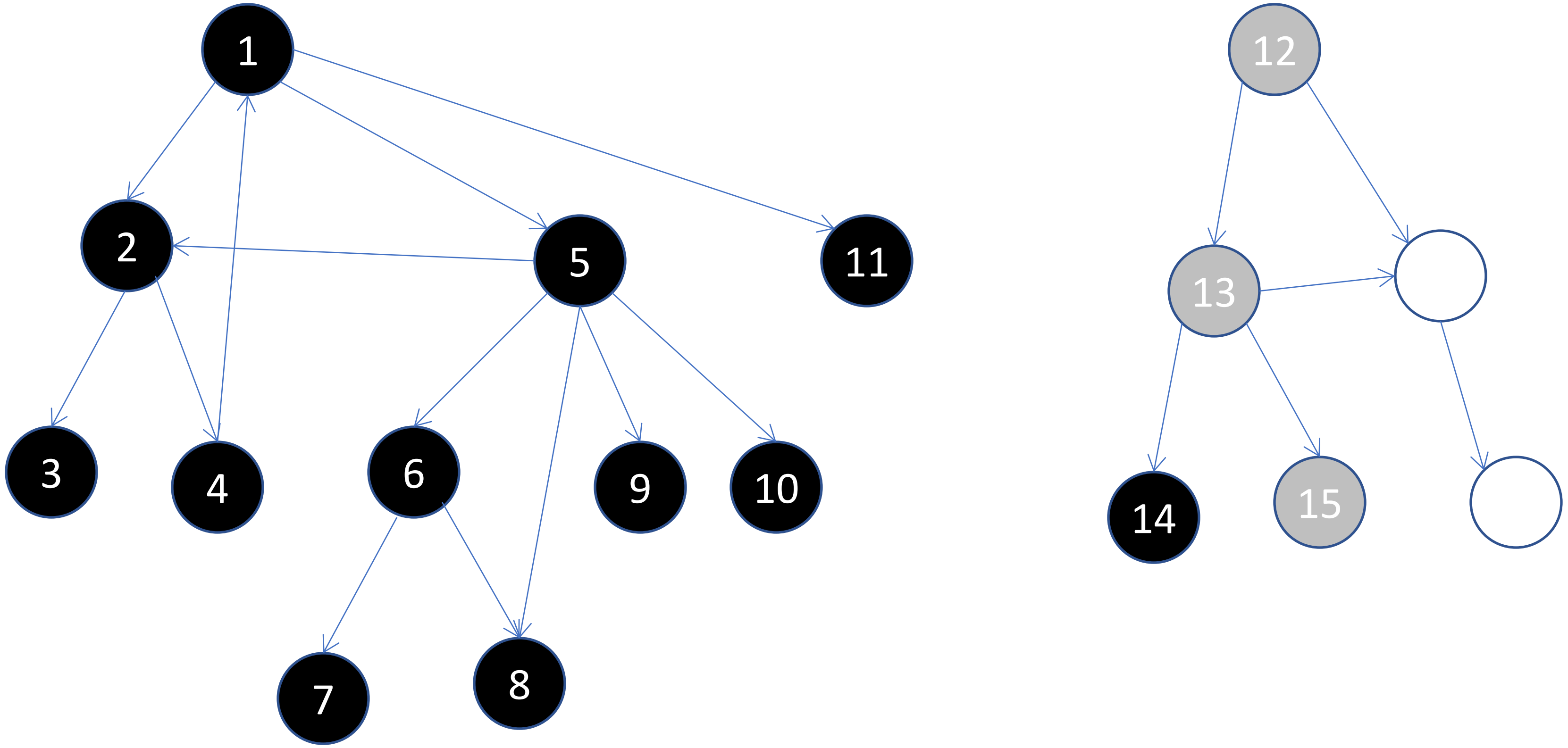
A DFS example (2)



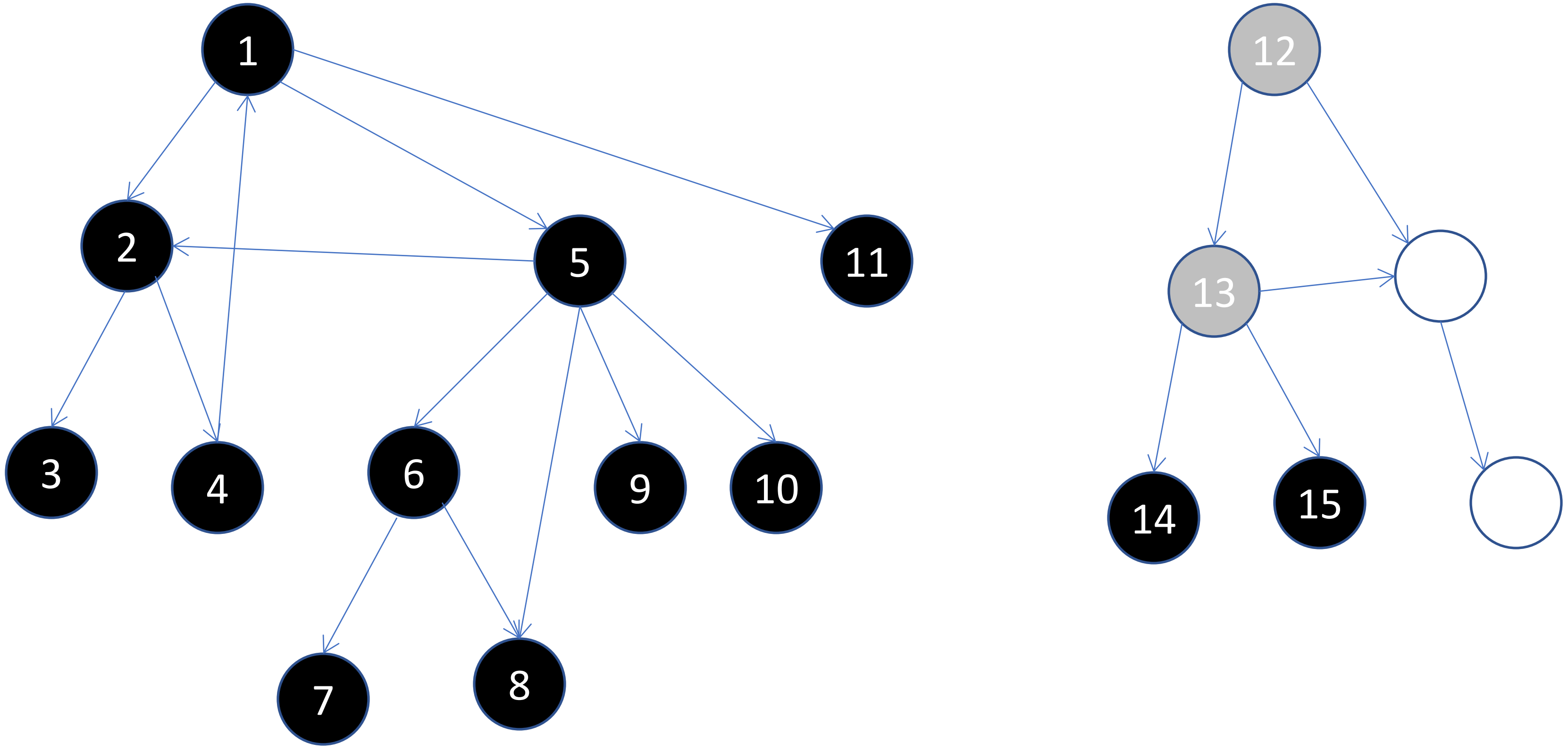
A DFS example (2)



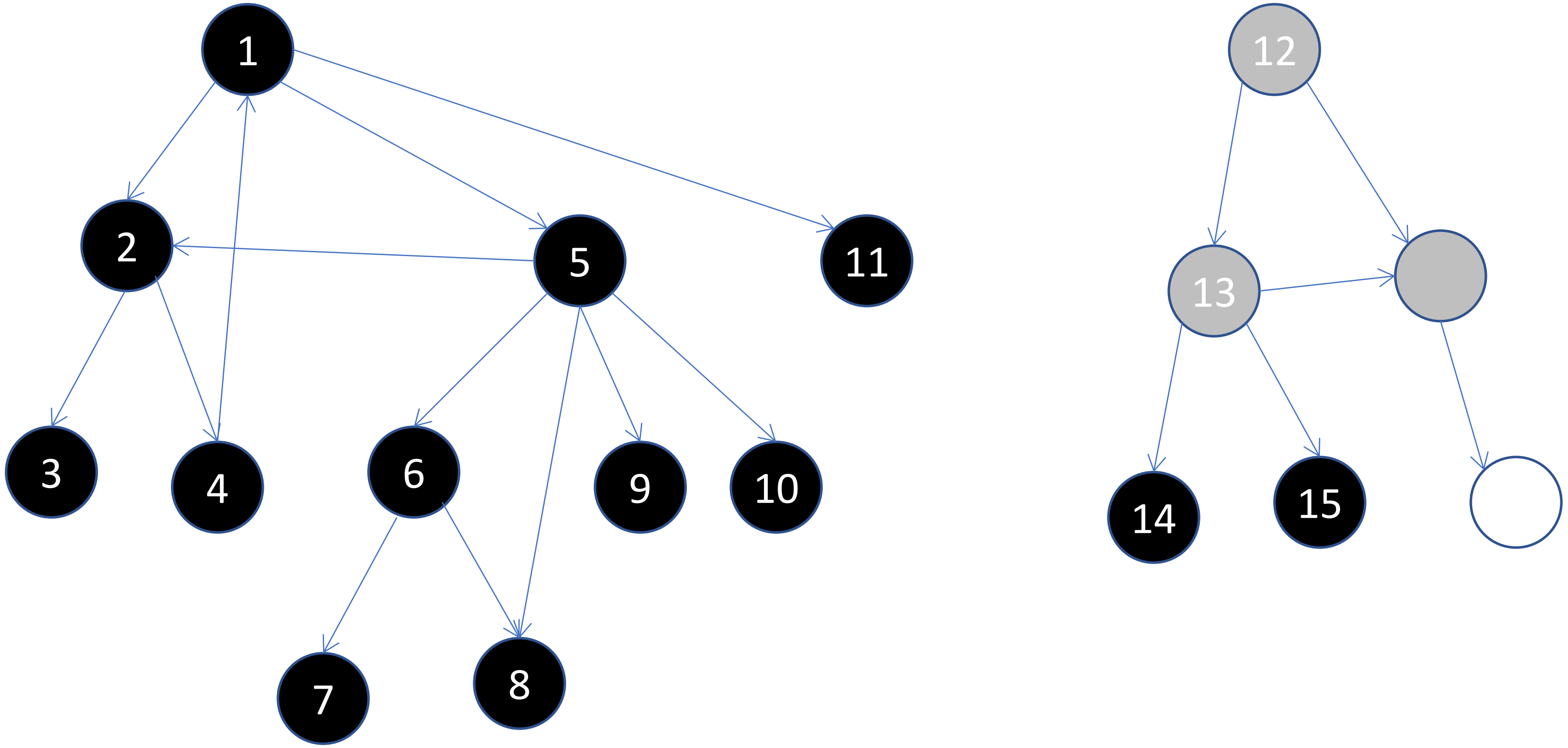
A DFS example (2)



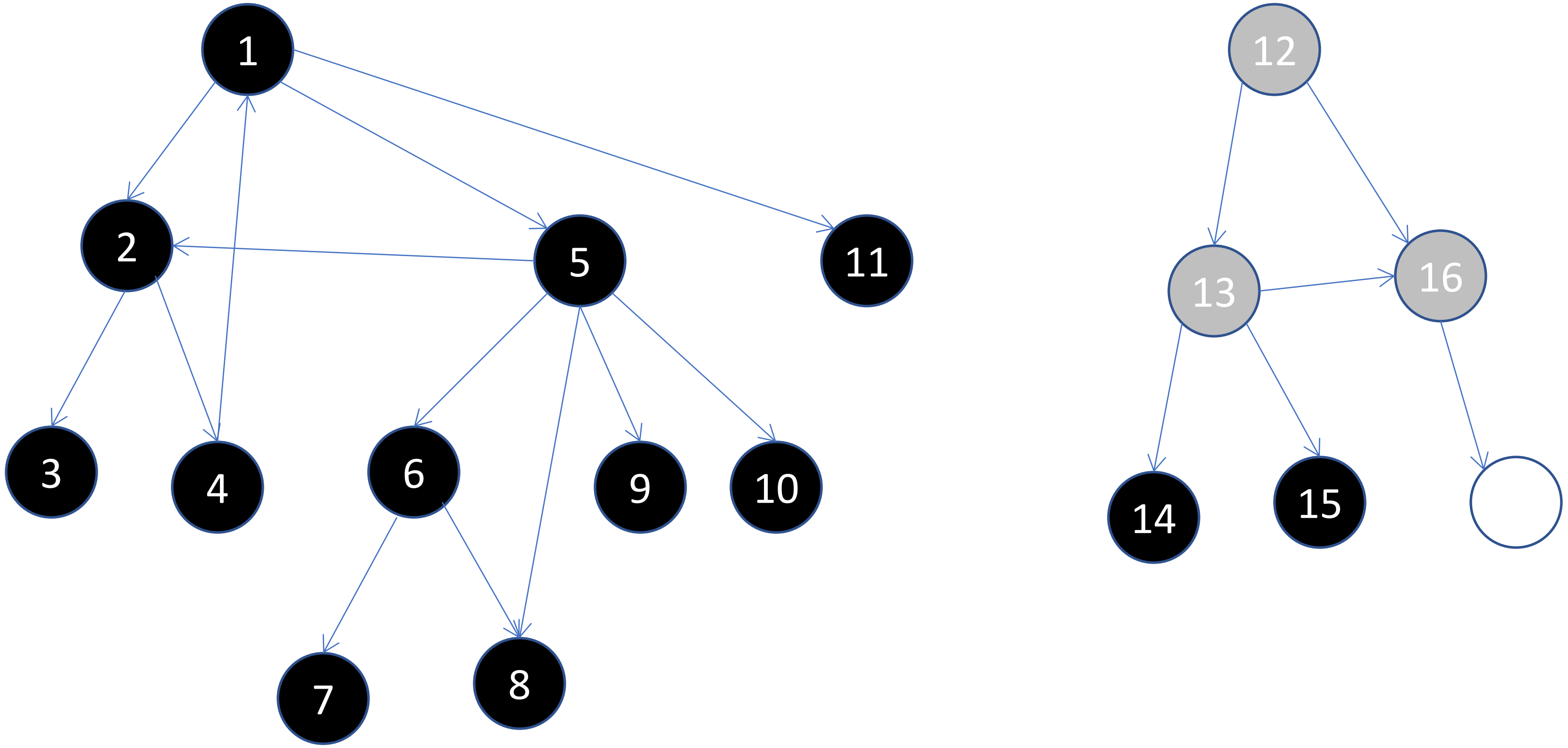
A DFS example (2)



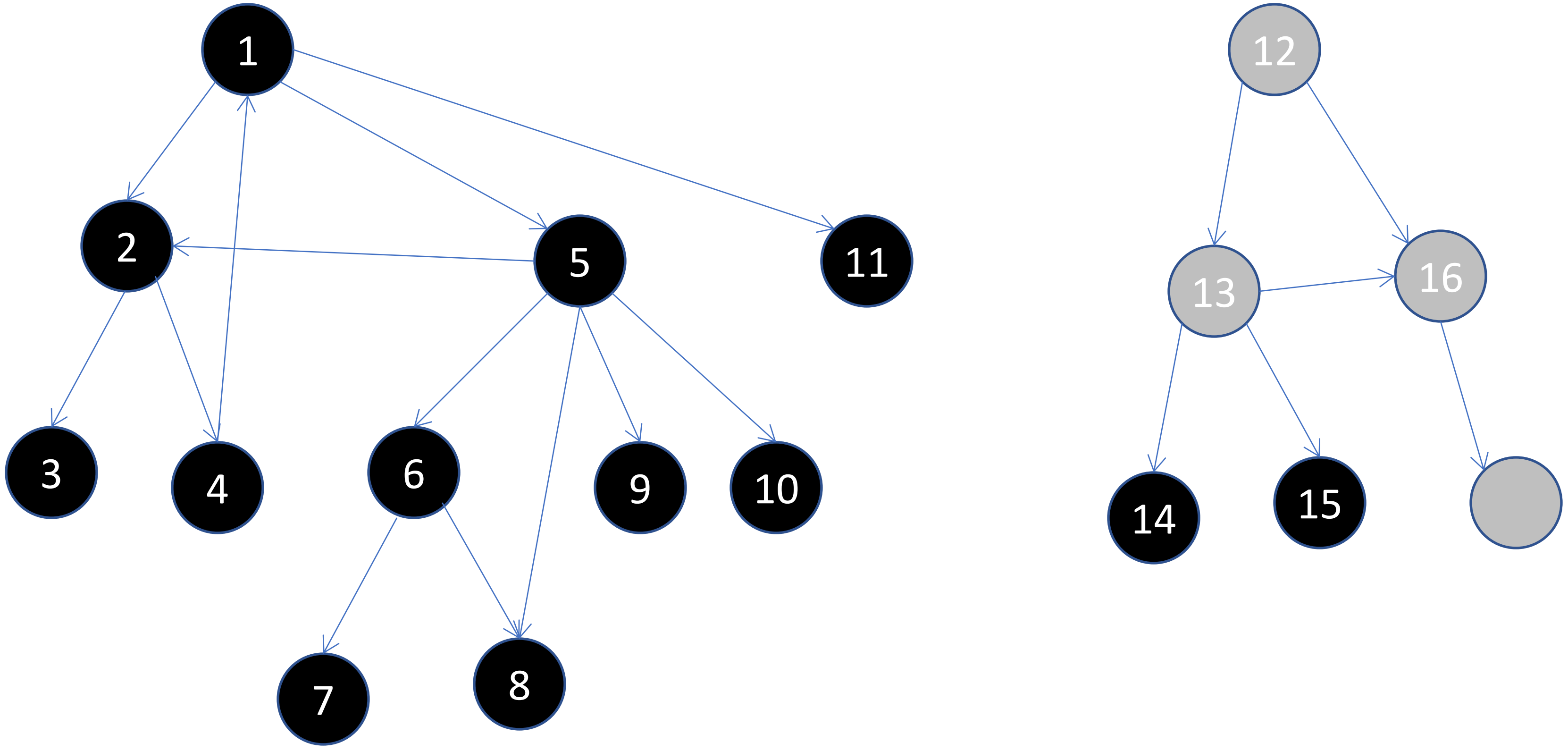
A DFS example (2)



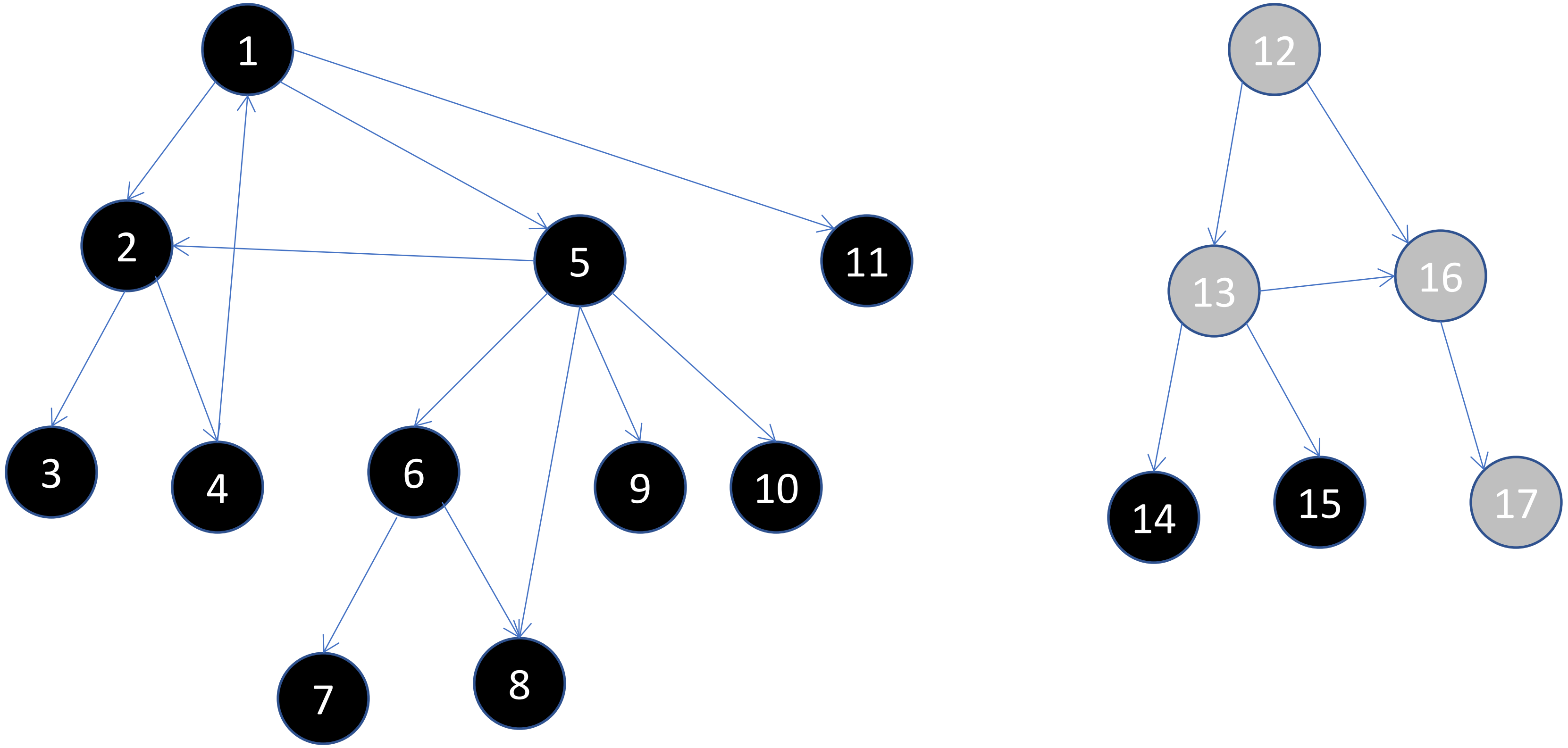
A DFS example (2)



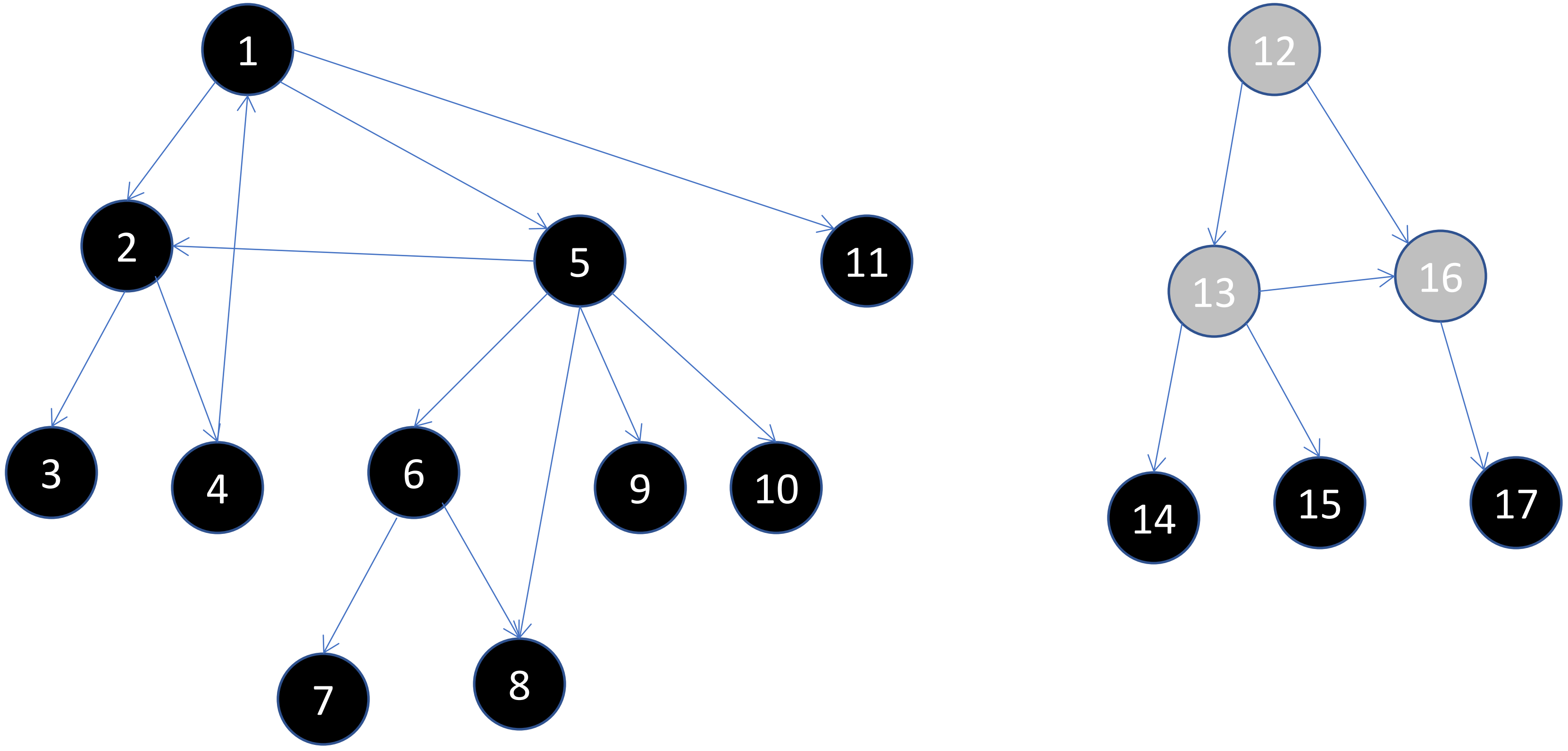
A DFS example (2)



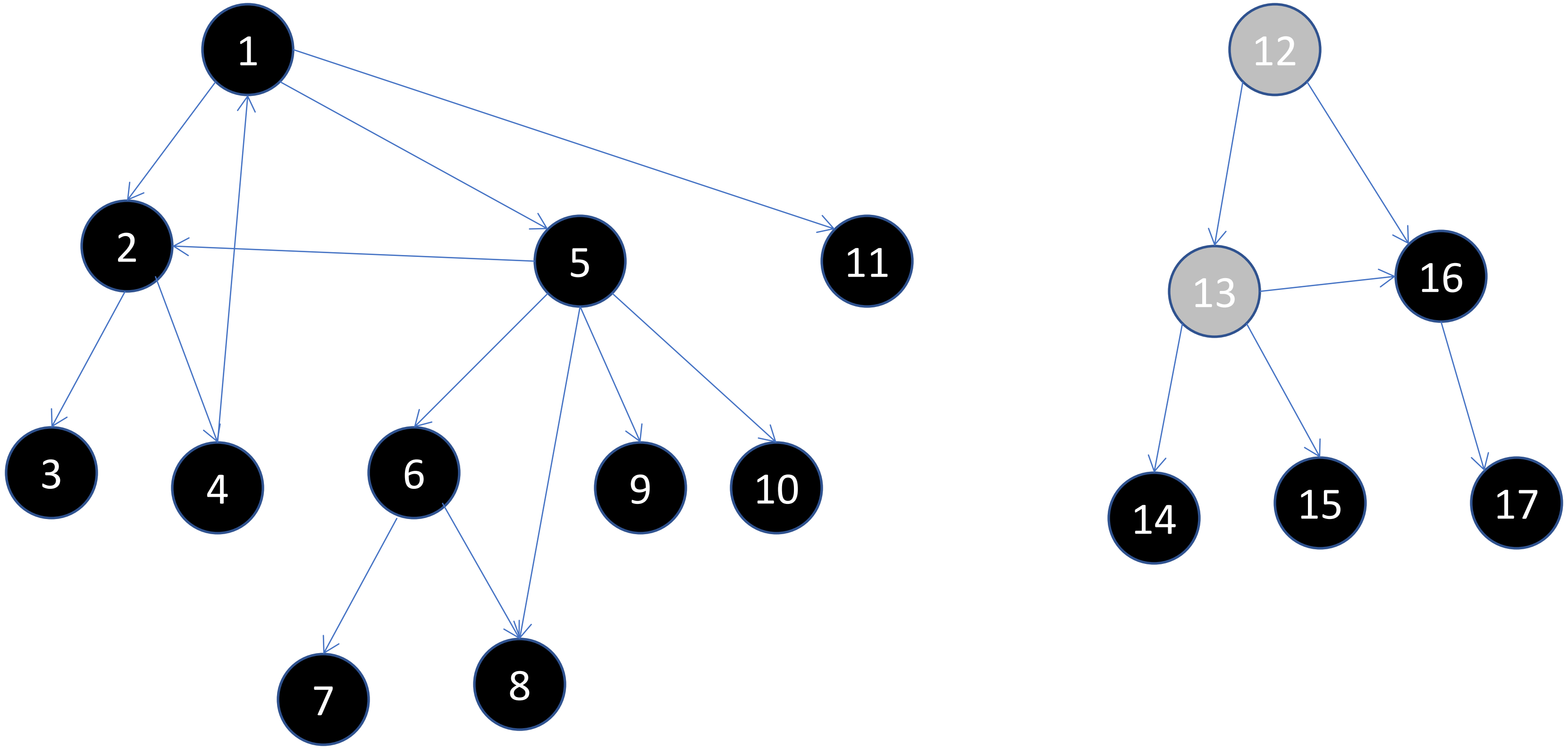
A DFS example (2)



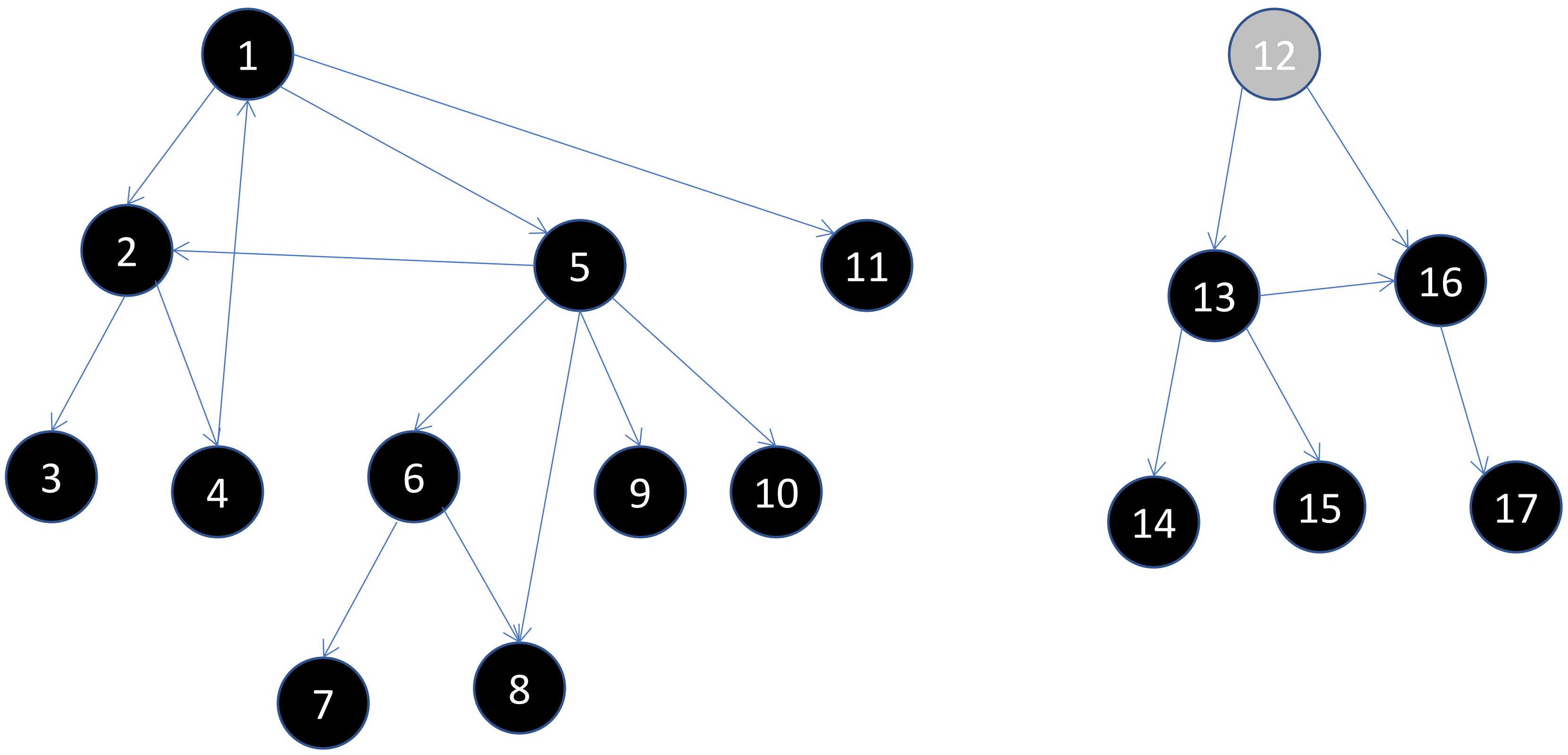
A DFS example (2)



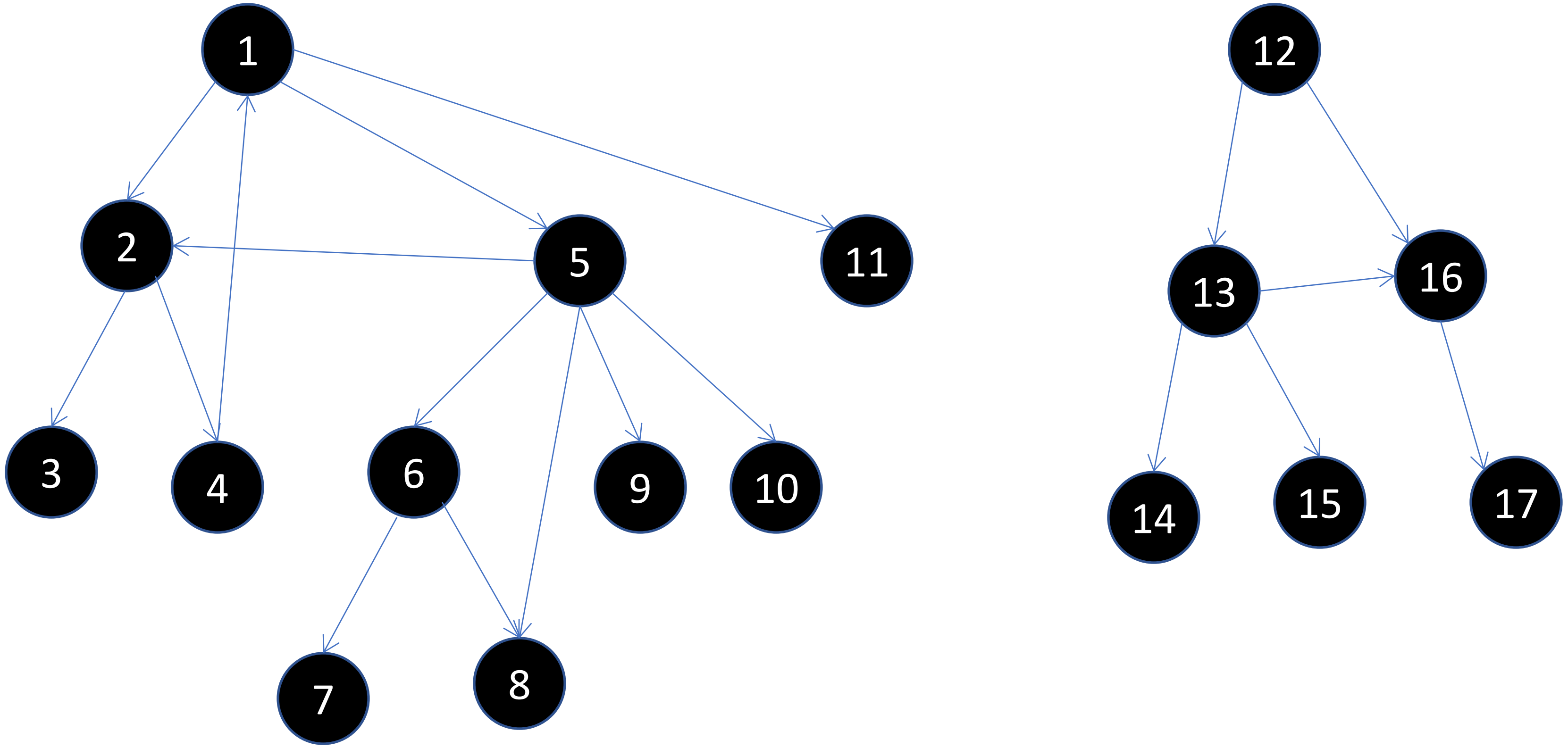
A DFS example (2)



A DFS example (2)



A DFS example (2)



RECAP: Four Kinds of Arcs

- In a search forest F of a graph G, we can find four different kinds of arcs:

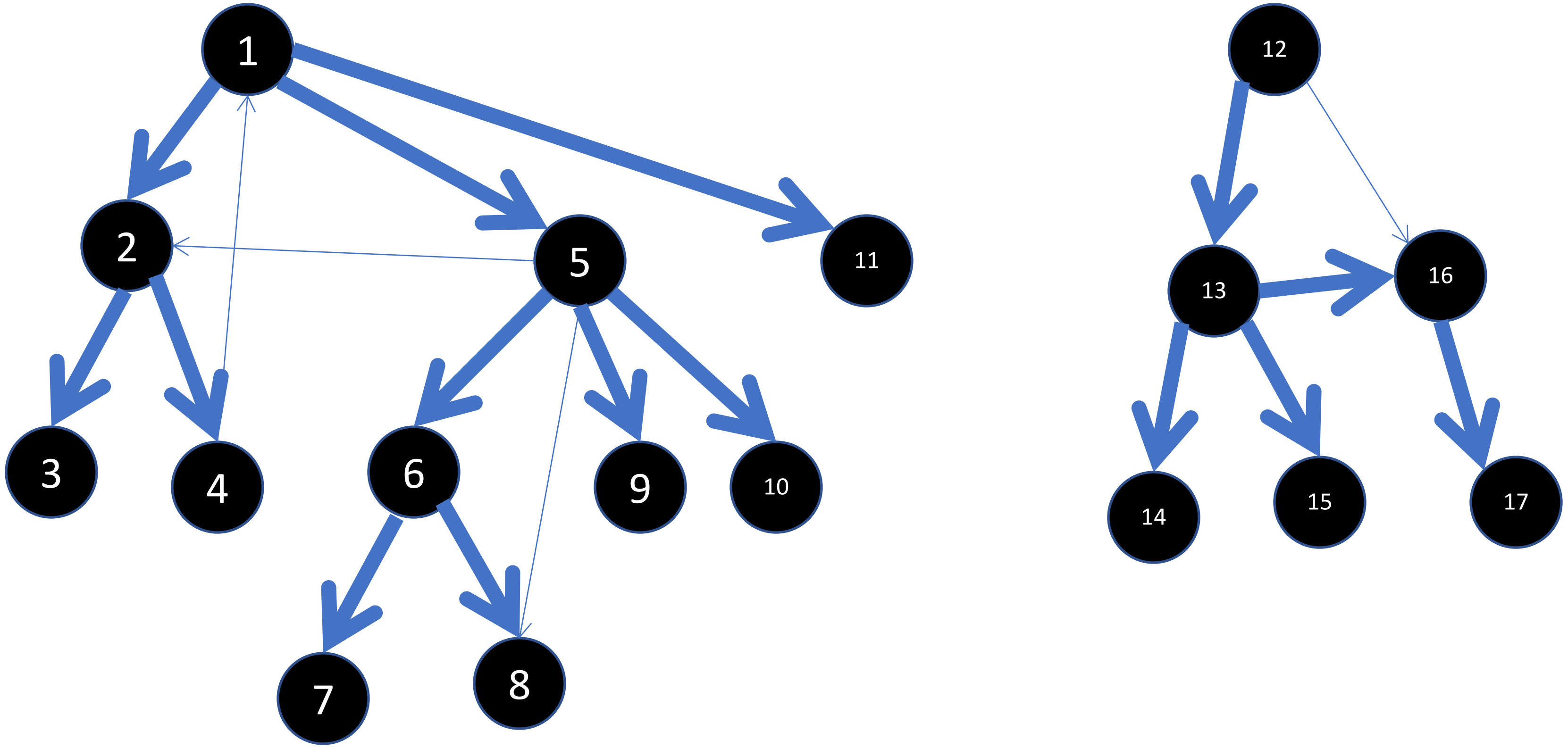
- Tree arc: an arc in G that connects a vertex in G to one of its immediate descendants in the tree of F that the vertex belongs to, i.e., if the arc belongs to the tree

- Forward arc: an arc that does not belong to a tree in F but that connects a vertex to one of its descendants in the tree

- Back arcs: an arc that does not belong to a tree in F but that connects a vertex to one of its ancestors in the tree

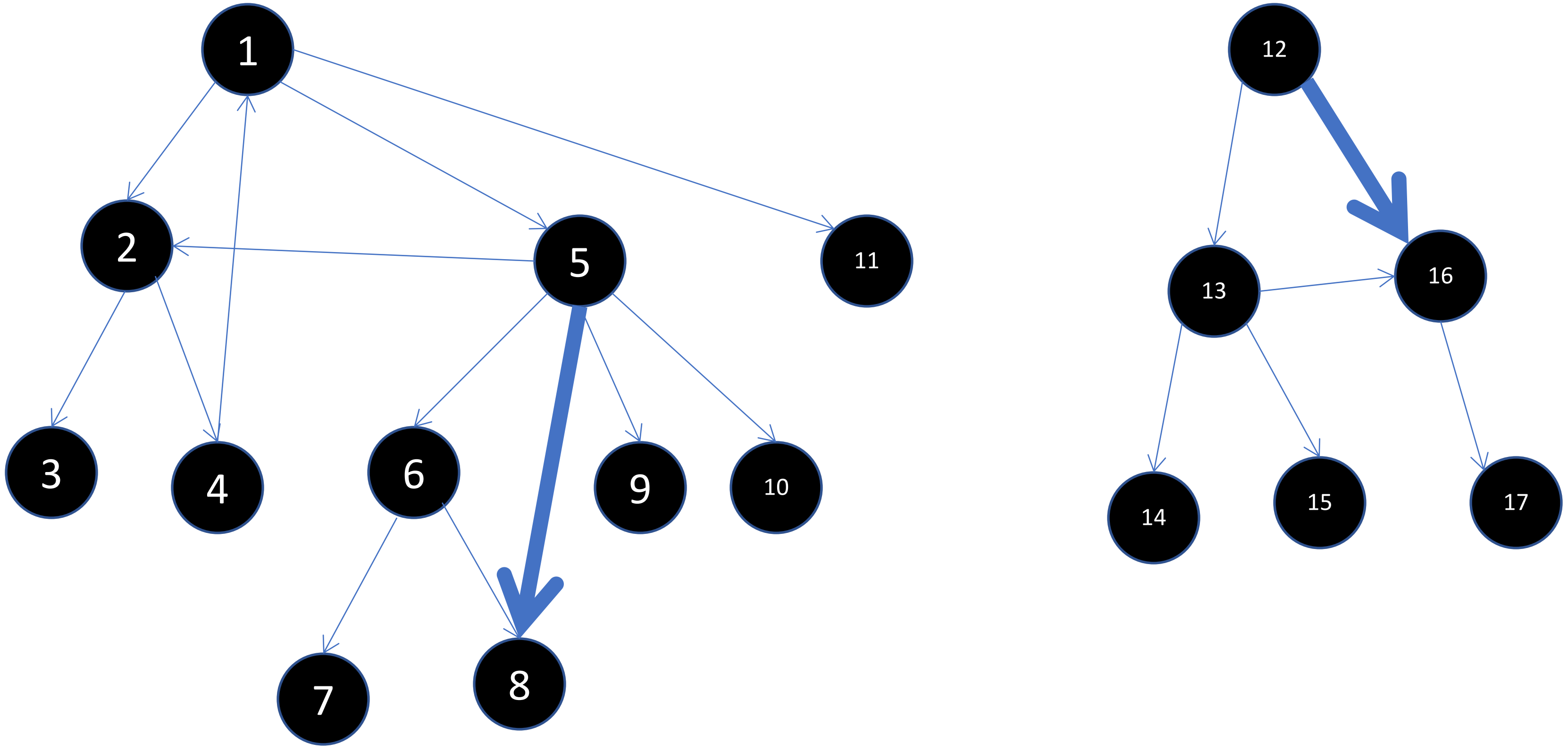
- Cross arcs: arcs that fall into neither of the above categories

DFS Traversal: Tree Arcs



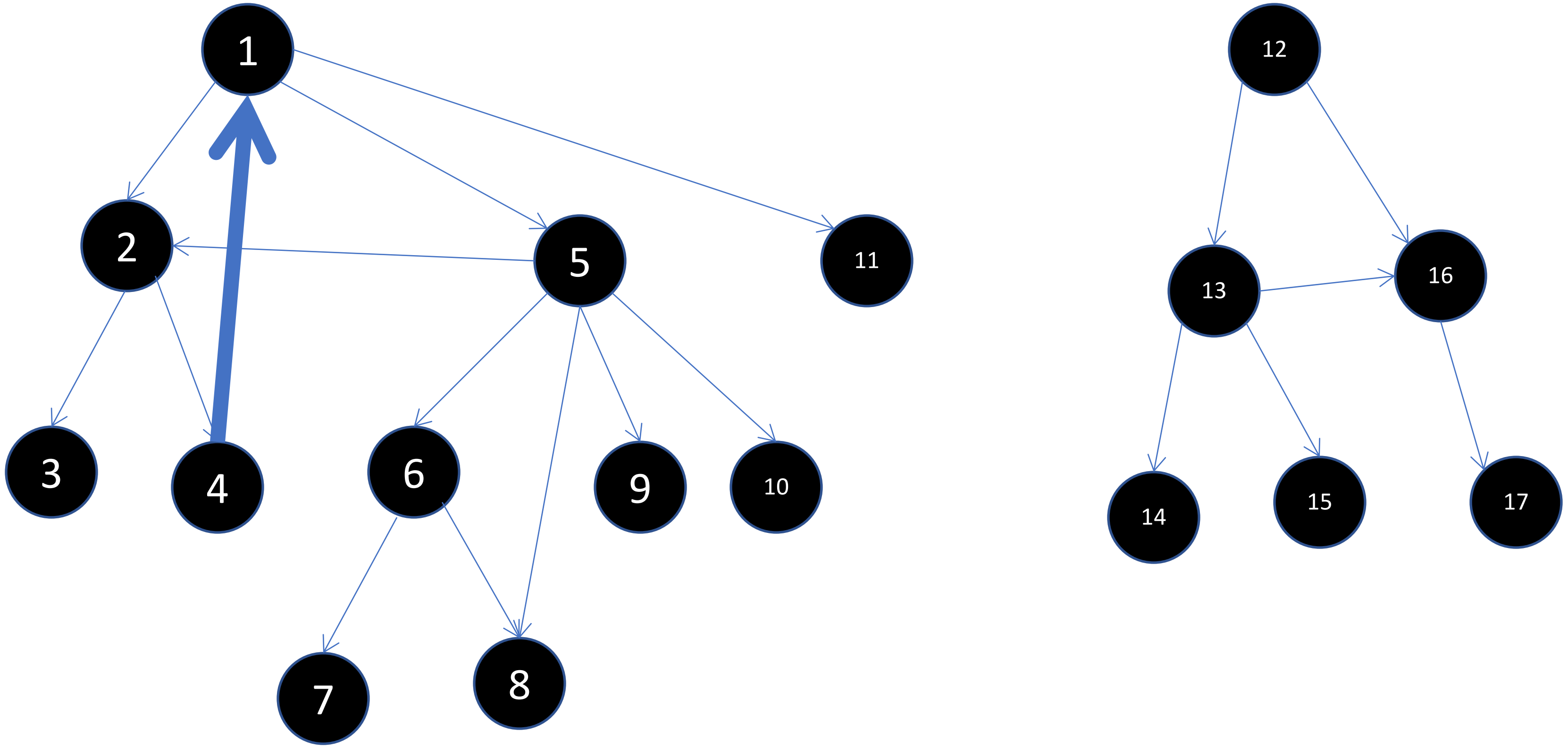
DFS Traversal: Forward Arcs

Forward arc: an arc that does not belong to a tree in F but that connects a vertex to one of its descendants in the tree



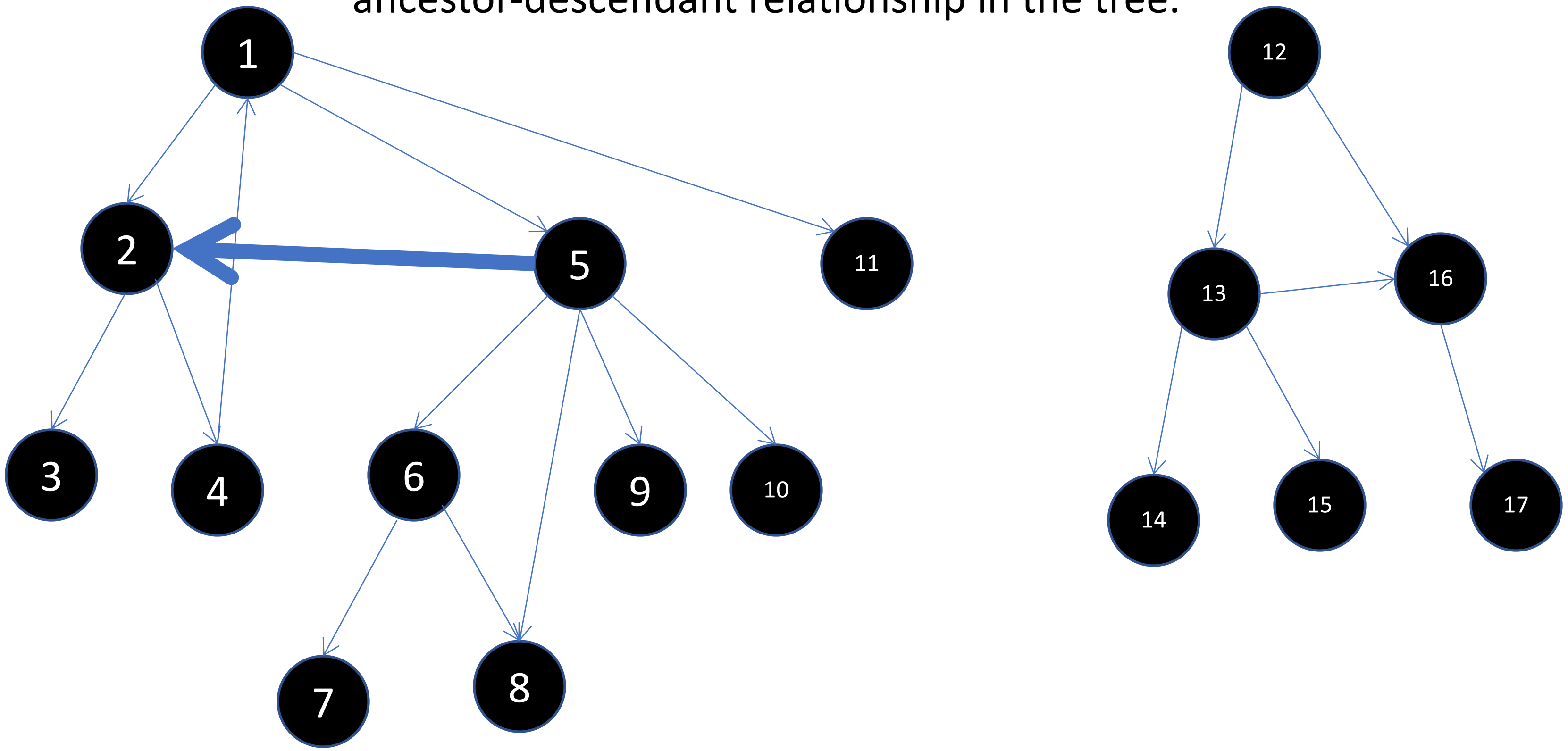
DFS Traversal: Back Arc

Back arcs: an arc that does not belong to a tree in F but that connects a vertex to one of its ancestors in the tree



DFS Traversal: Cross Arc

Cross arcs: an arc that does not belong to a tree in F and connects nodes that do not form an ancestor-descendant relationship in the tree.



Basic properties of depth-first search

- Each call to DFSVISIT(v) terminates only when all nodes reachable from v via a path of white nodes have been seen.

- Suppose that (v,w) is an arc of a digraph. Cases:
  - tree or forward arc: $seen[v]<seen[w]<done[w]<done[v]$;
  - back arc: $seen[w]<seen[v]< done[v]< done[w]$;
  - cross arc: $seen[w]<done[w]<seen[v]<done[v]$.
- Note that there are no cross edges on a graph G. (Why?)

Using DFS to Determine Ancestors of a Tree

- **Theorem**: Suppose that we have performed DFS on a digraph G, resulting in a search forest F. Let $v, w \in V(G)$ and suppose that $seen[v] < seen[w]$.

- If v is an ancestor of w in F, then

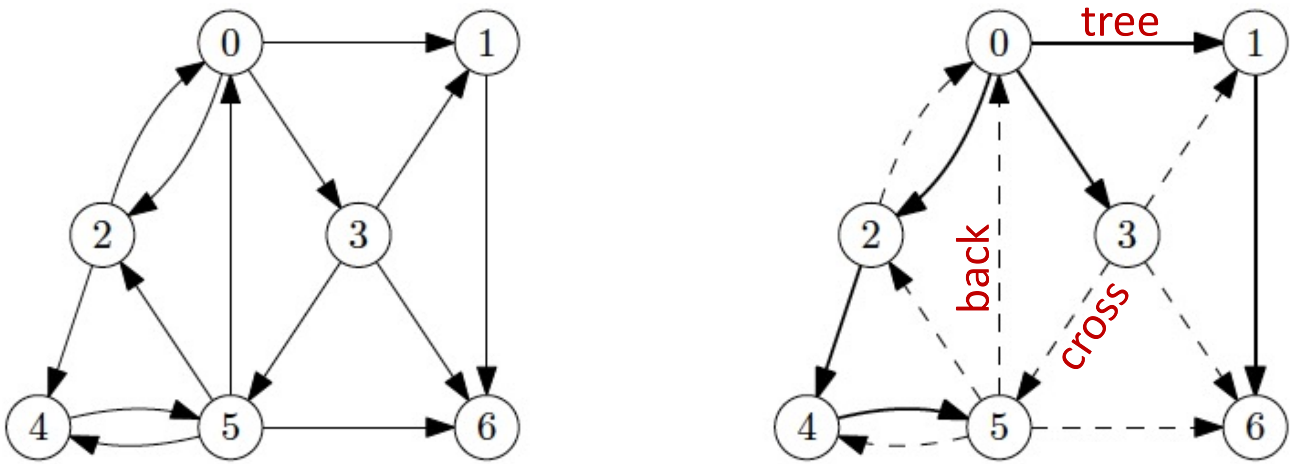
  $seen[v]<seen[w]<done[w]<done[v]$.

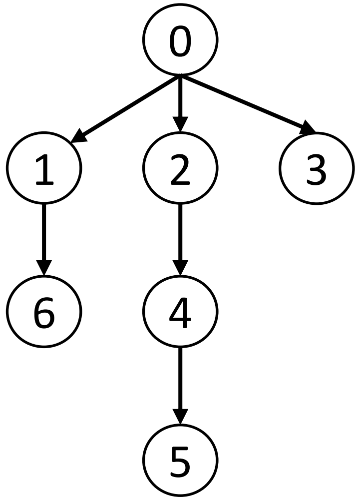
- If v is not an ancestor of w in F, then

  $seen[v]<done[v]<seen[w]<done[w]$.

Example 23.2

Example 23.2. A digraph and its DFS search tree, rooted at node 0. The dashed arcs indicate the original arcs that are not part of the DFS search tree.

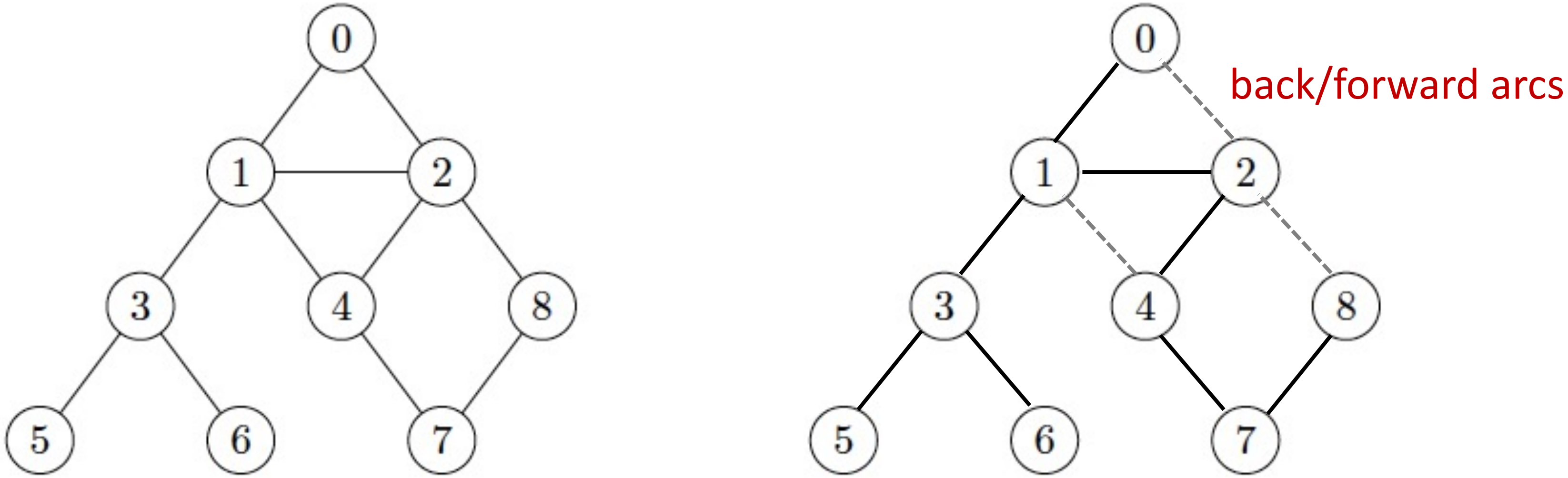




Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

Example 23.3

Example 23.3. Use the nodes on the right to draw the search tree you obtain by running DFS on the graph on the left, starting at vertex 0. Use dashed edges to indicate edges that are not arcs in the search tree.

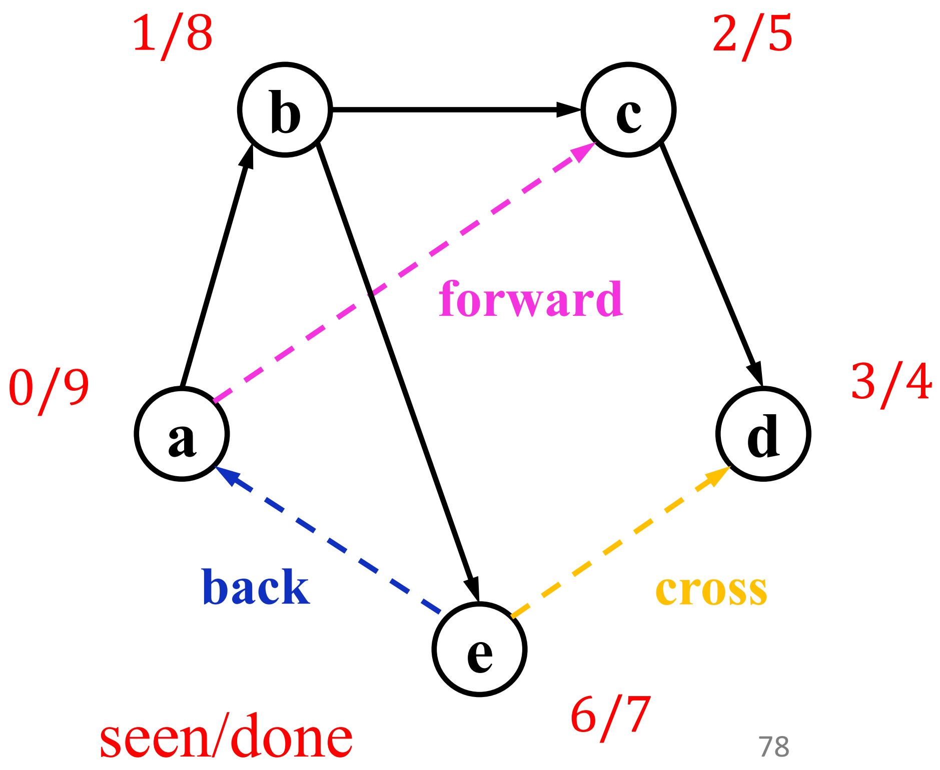


Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal). 8 tree arcs, 3 back/forward arcs, no cross arcs in graph

Example 23.7

- Explain how to determine (u,v) in DFS algorithm whether it is a tree-, back-, forward- or cross-arc?

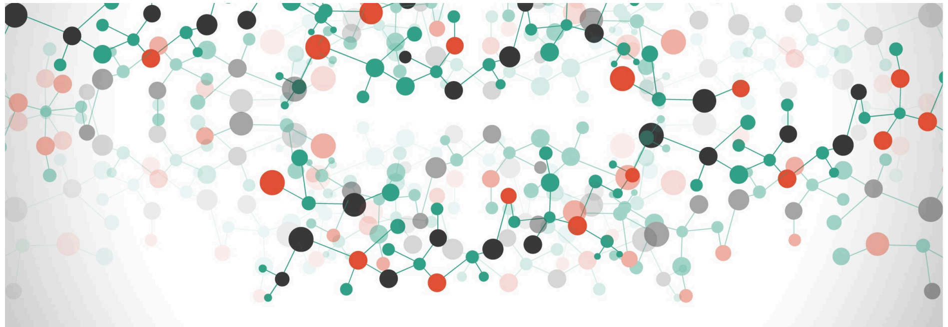
1. If v is white, then (u,v) is a **tree** arc
2. If v is grey, then (u,v) is a **back** arc
3. If v is black, then (u,v) is
   - a **cross** arc($seen[v]<seen[u]$, $done[v]<seen[u]$), or
   - a **forward** arc($seen[u]<seen[v]<done[v]<done[u]$).



OUTLINE

- Graph Traversal Algorithms
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Priority-first Search (PFS)

- Implementation
  - Stack – DFS
  - Queue – BFS
  - Priority Queues – PFS



Breadth-first Search Algorithm (BFS)

- BFS is also a specific implementation of our fundamental graph traversal algorithm (and also known as breadth-first traversal)
- It specifies that we select the next grey vertex to pick as the oldest remaining grey vertex.

The Abstract Data Type: Queue

- Special list in which insertion occurs at one end (tail) and deletion occurs at the other end (head) (first in first out principle).

- Add an element to the list (INSERT or PUSH or ENQUEUE).

- Delete an element (DELETE or POP or DEQUEUE).

- Return the head element without deleting it (GETHEAD or PEEK).

Breadth-first Search Algorithm (BFS)

Algorithm 4 Breadth-first search algorithm

function BFS(digraph G)
    queue Q
    array $colour[0..n-1], pred[0..n-1], d[0..n-1]$
    for $u \in V(G)$ **do**
        $colour[u] \leftarrow$ WHITE
        $pred[u] \leftarrow$ null
    for $s \in V(G)$ **do**
        if $colour[s] =$ WHITE **then**
            BFSVISIT(s)
    return $pred, d$

Iterative View of BFSVISIT

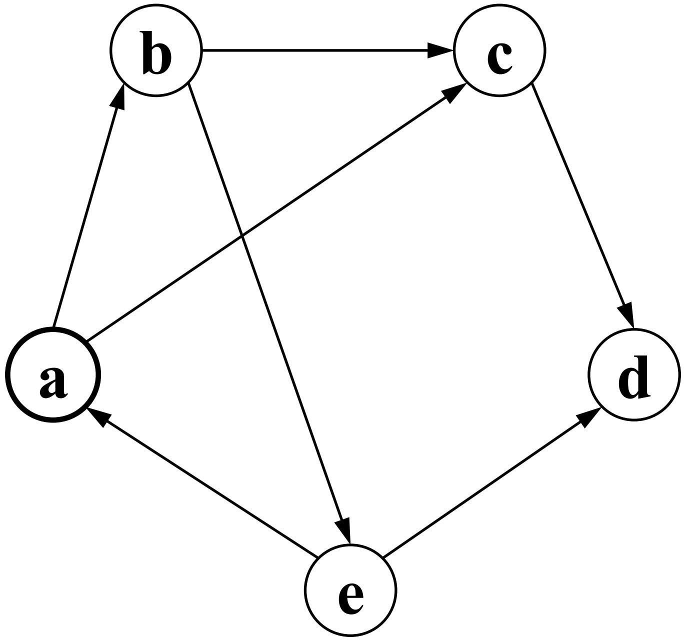
Algorithm 5 Breadth-first visit algorithm.

function BFSVISIT(node s)
    $color[s] \leftarrow$ GREY
    $d[s] \leftarrow 0$
    $Q.insert(s)$
    while not $Q.isEmpty()$ **do**
        $u \leftarrow Q.peek()$
        for each v adjacent to u **do**
            if $colour[v] =$ WHITE **then**
                $colour[v] \leftarrow$ GREY; $pred[v] \leftarrow u; d[v] \leftarrow d[u] + 1$
                $Q.insert(v)$
        $Q.delete()$
        $colour[u] \leftarrow$ BLACK

A BFS example (1)

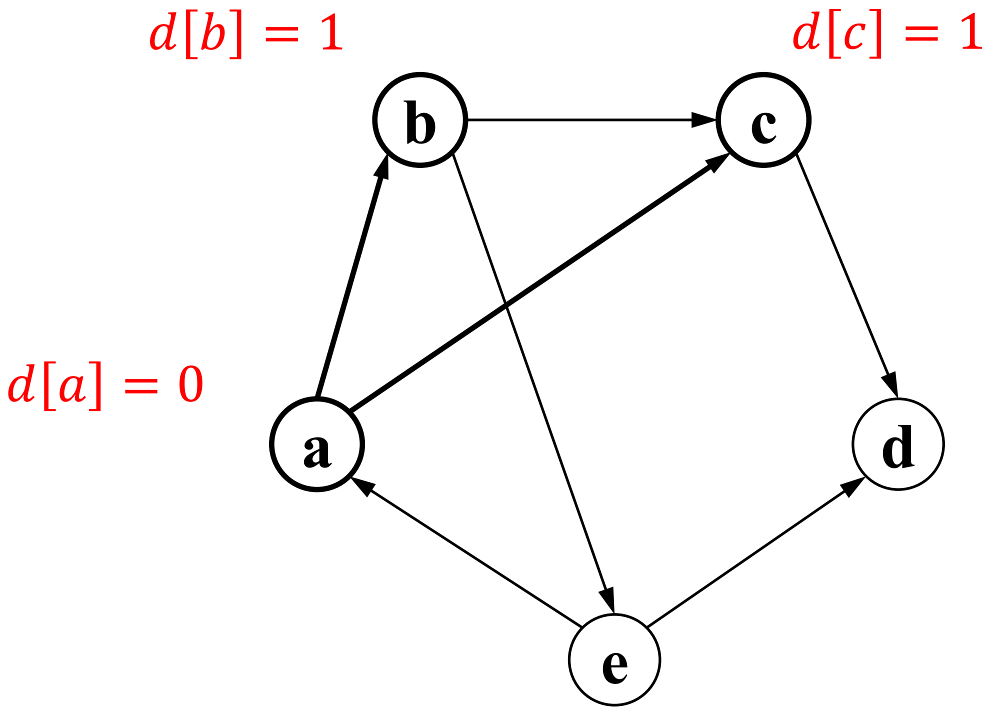
Queue: a

$d[a] = 0$



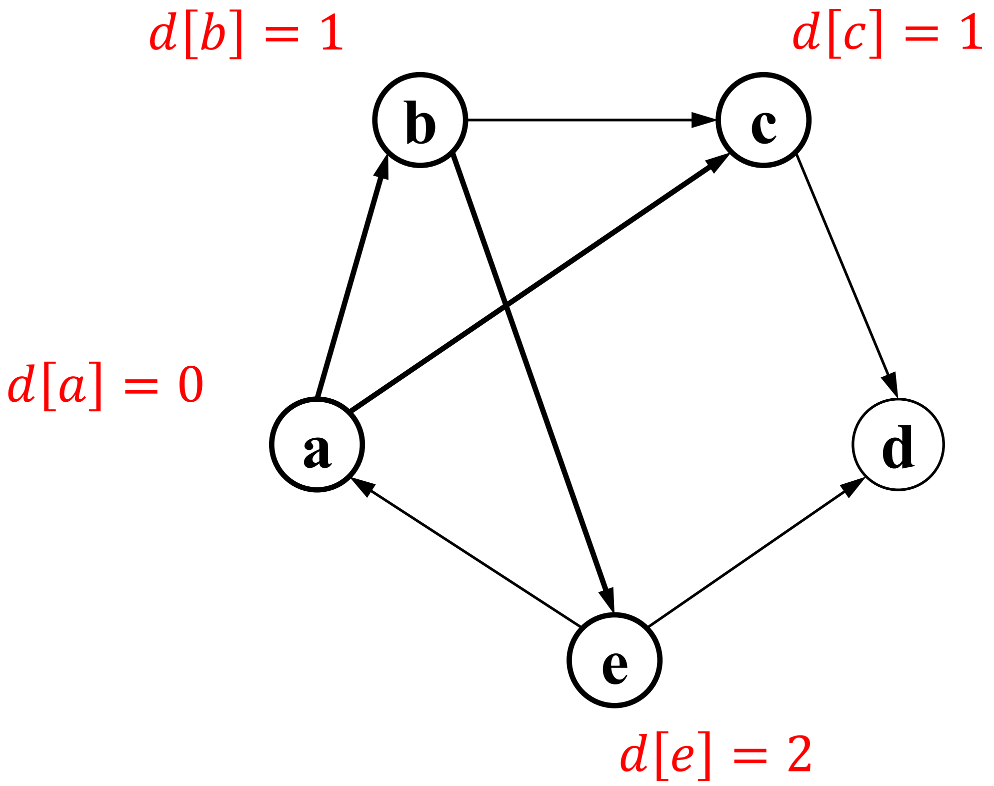
A BFS example (1)

Queue: b c



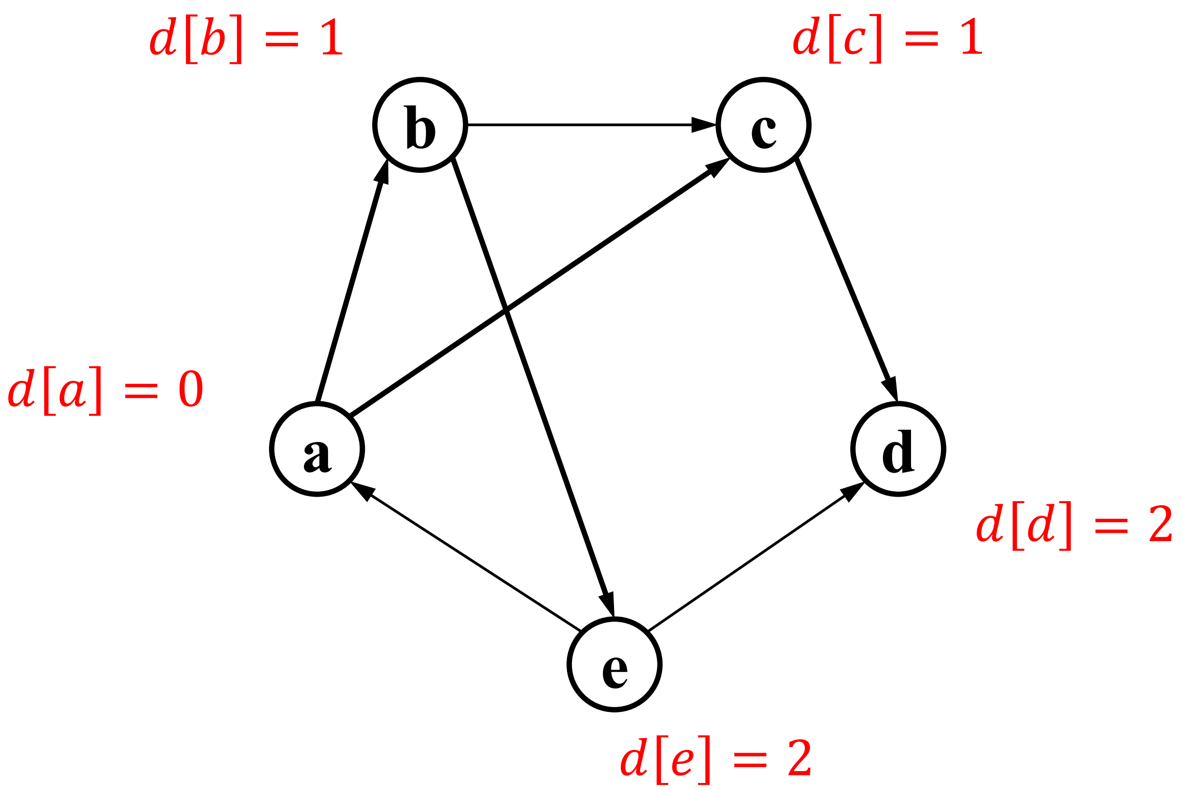
A BFS example (1)

Queue: c e

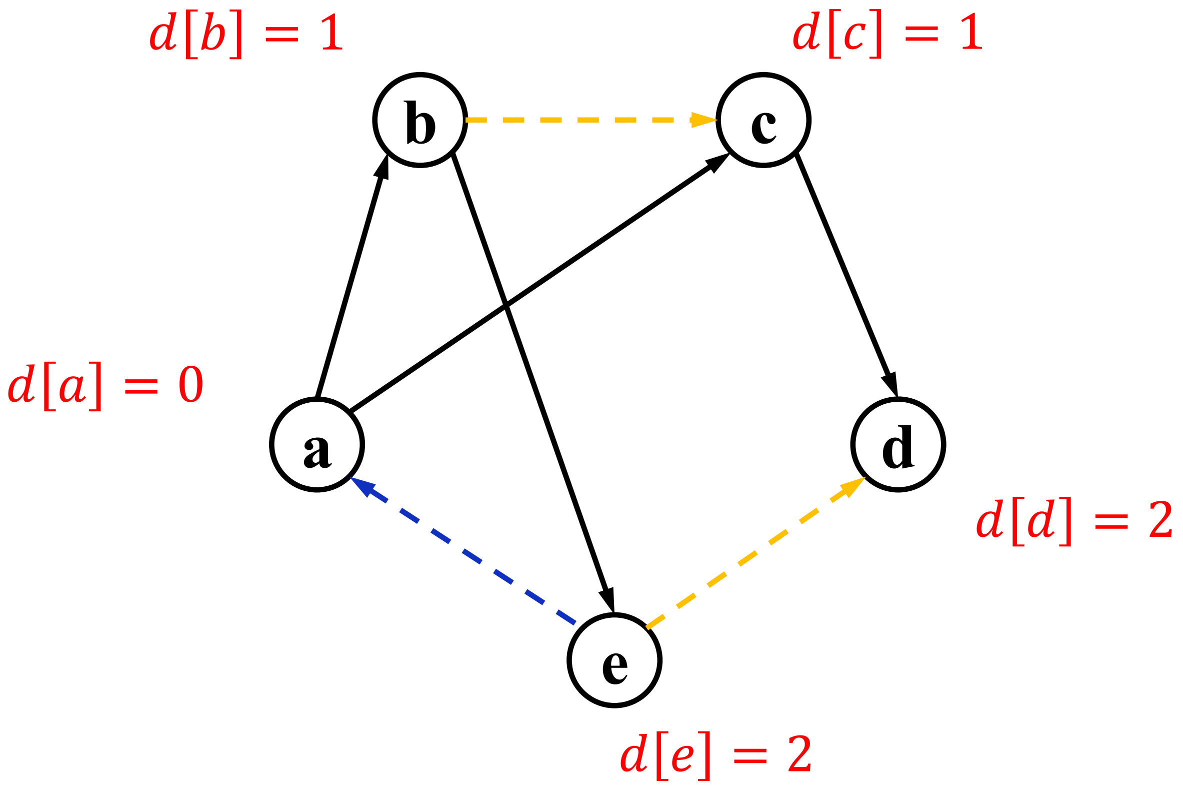


A BFS example (1)

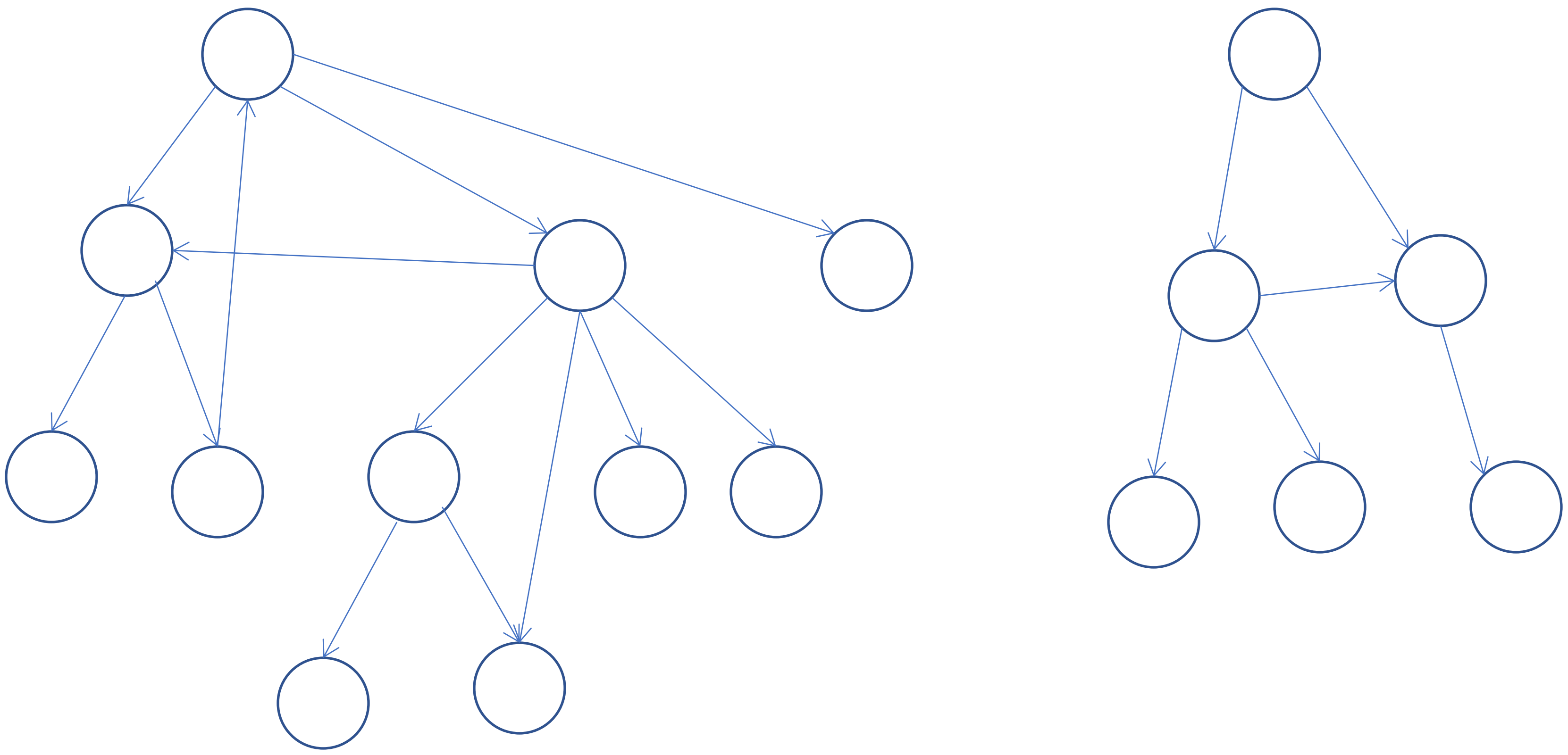
Queue: e d



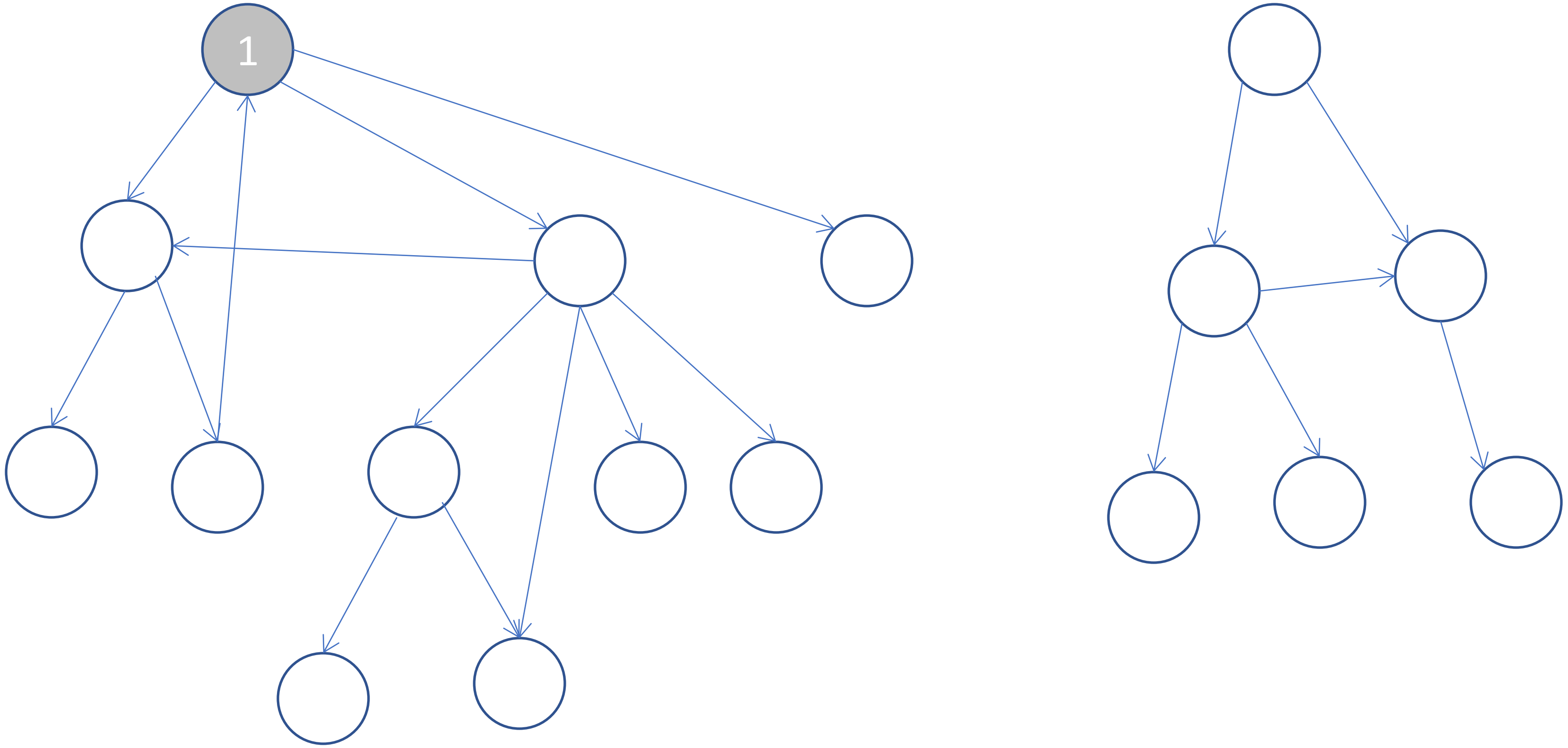
A BFS example (1)



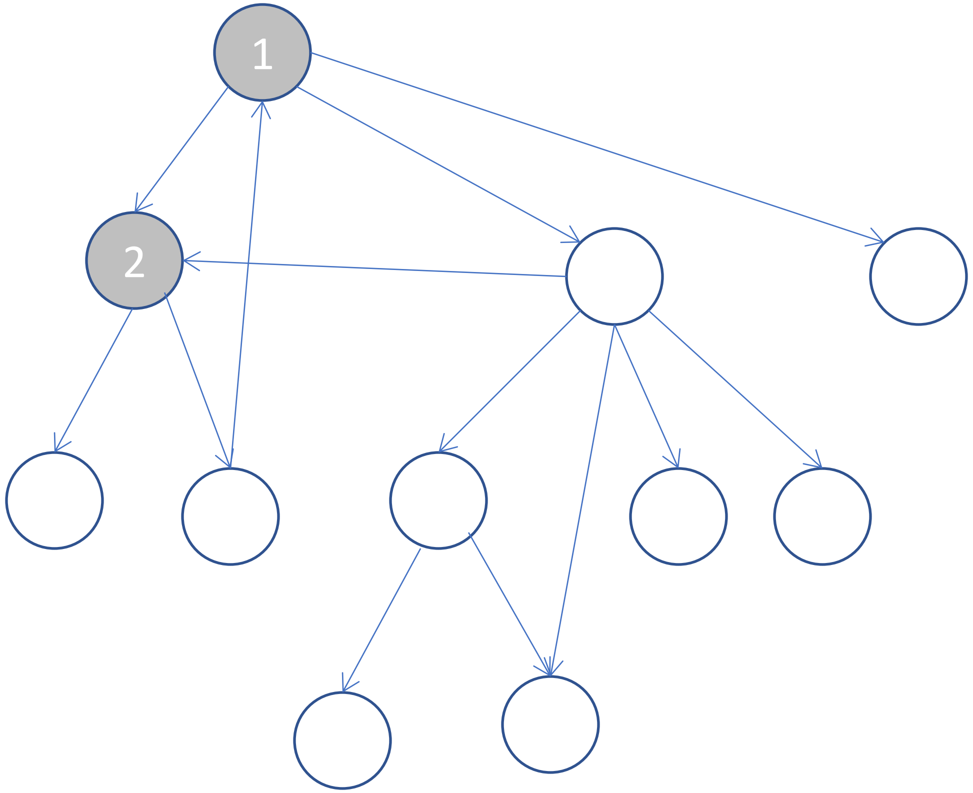
A BFS example (2)

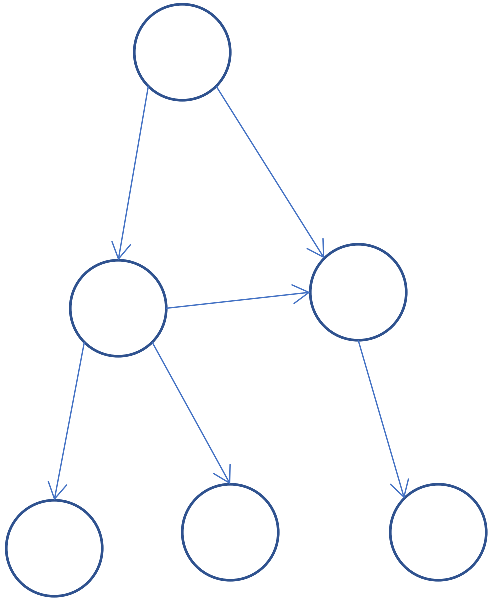


A BFS example (2)

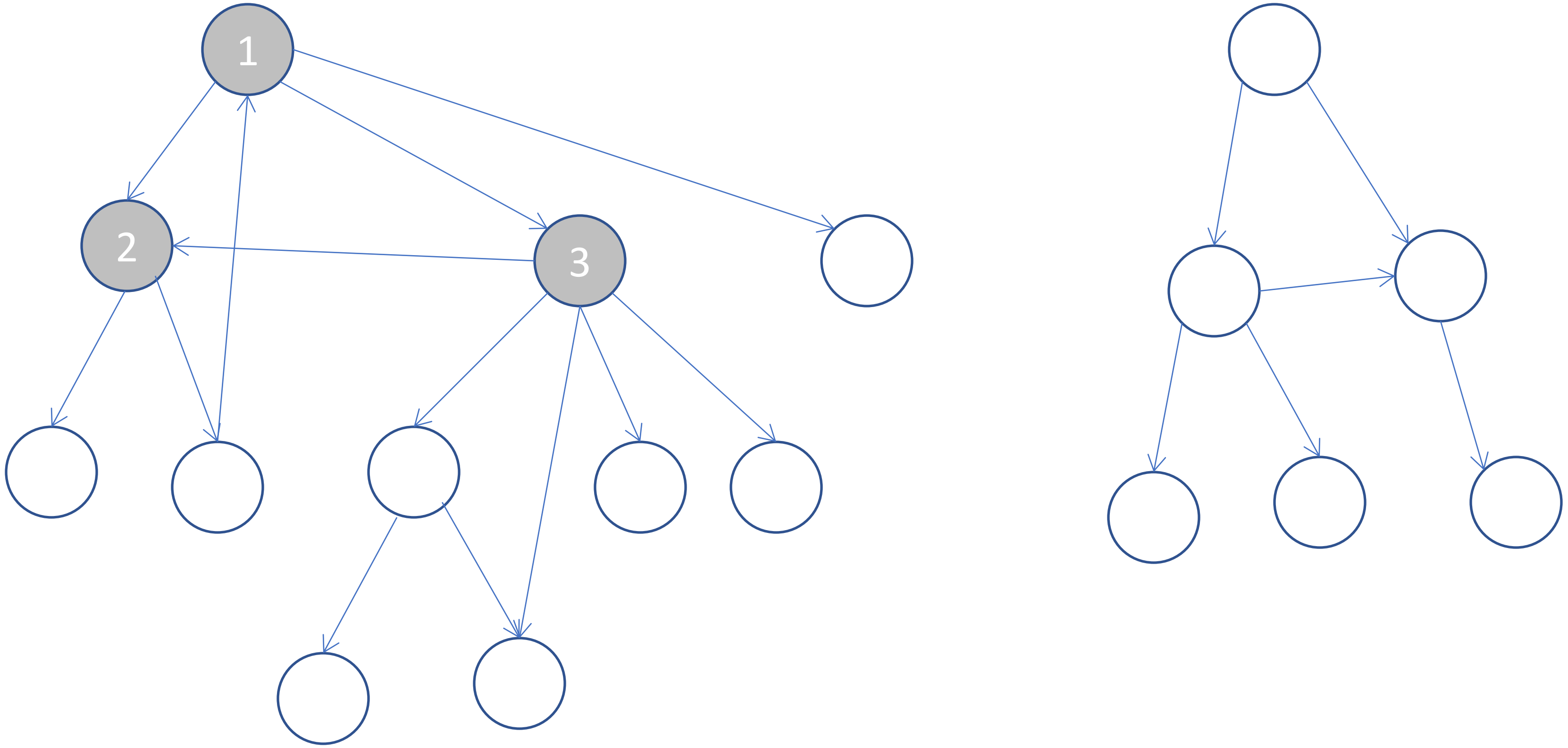


A BFS example (2)

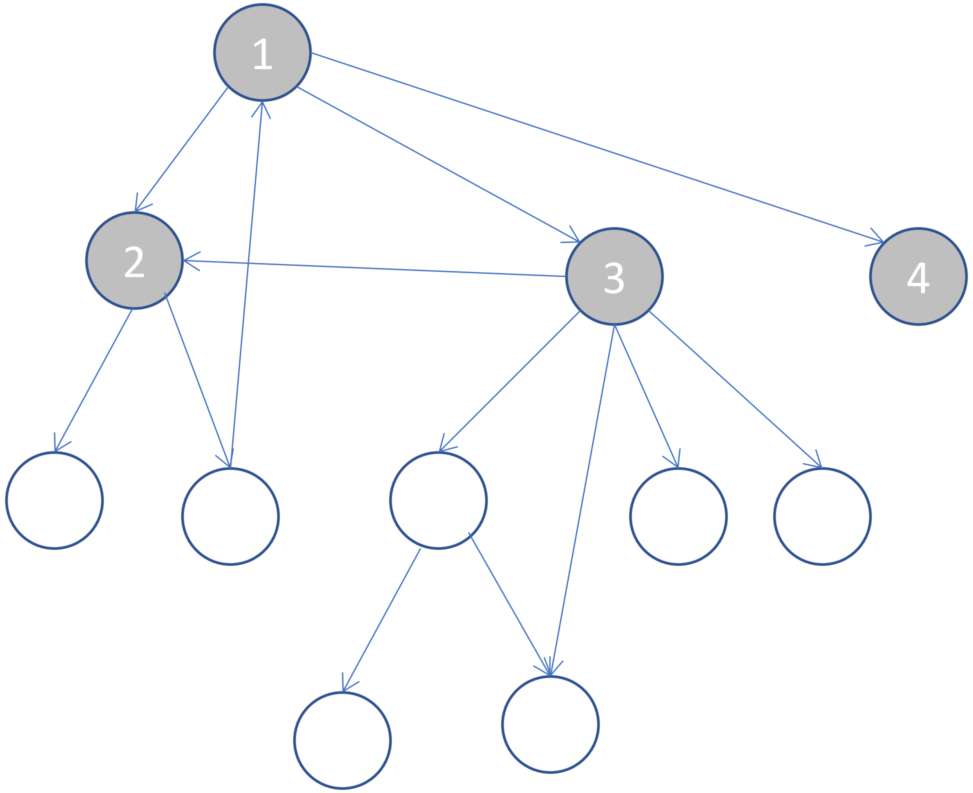


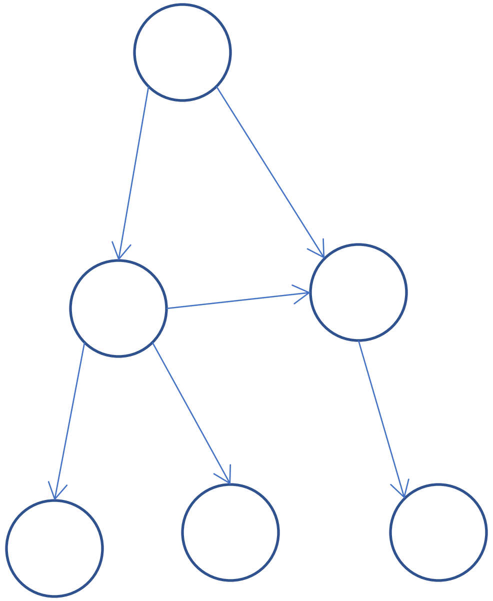


A BFS example (2)

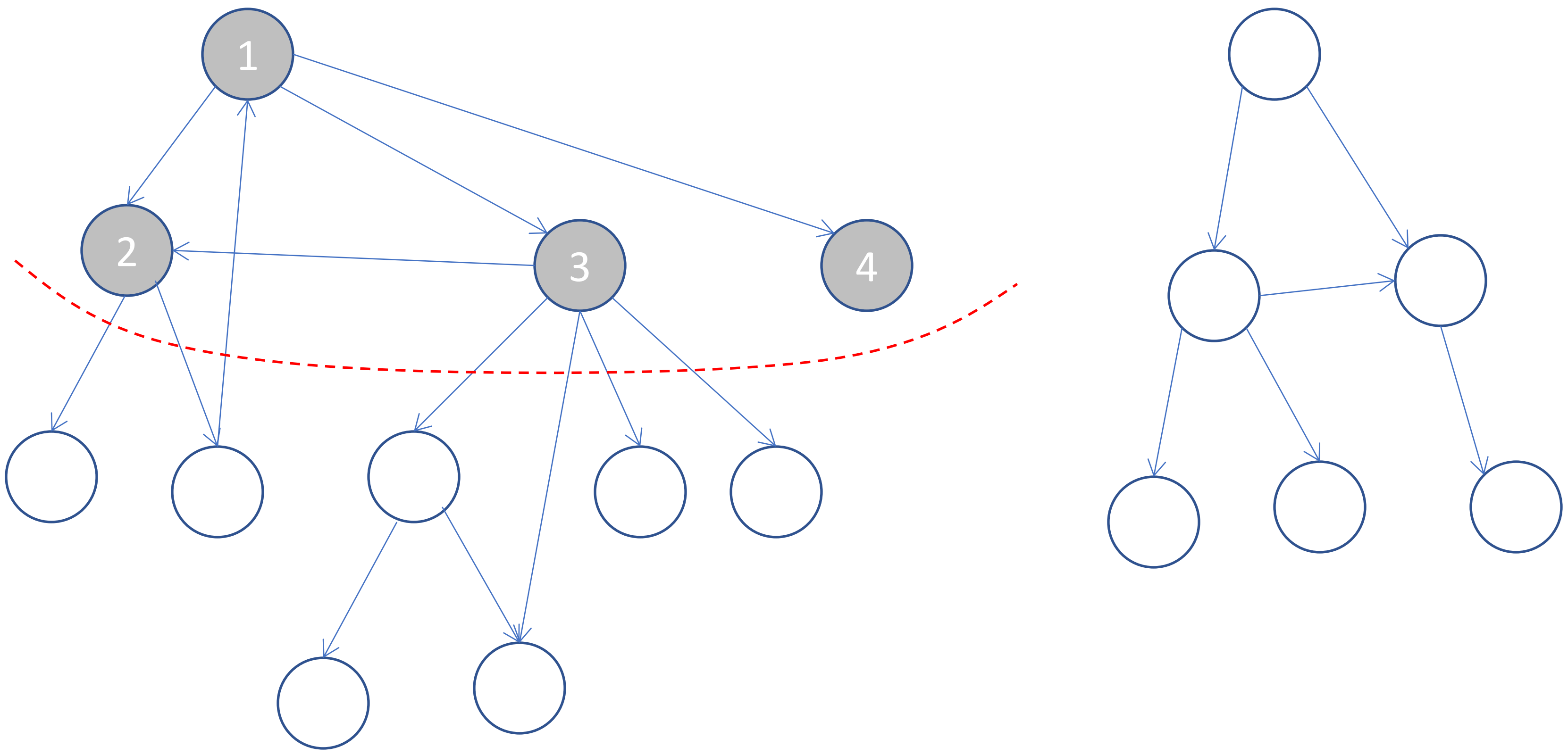


A BFS example (2)



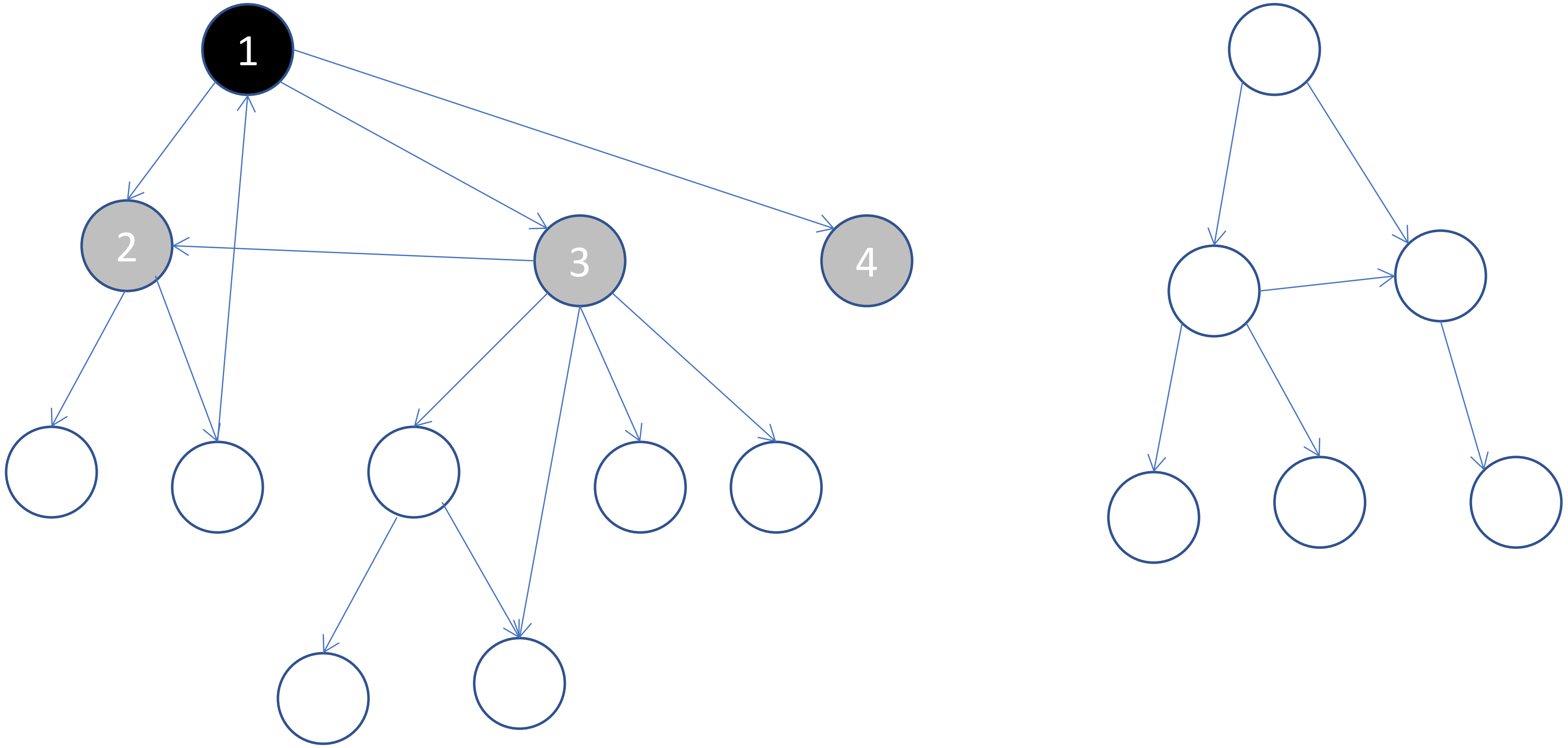


A BFS example (2)



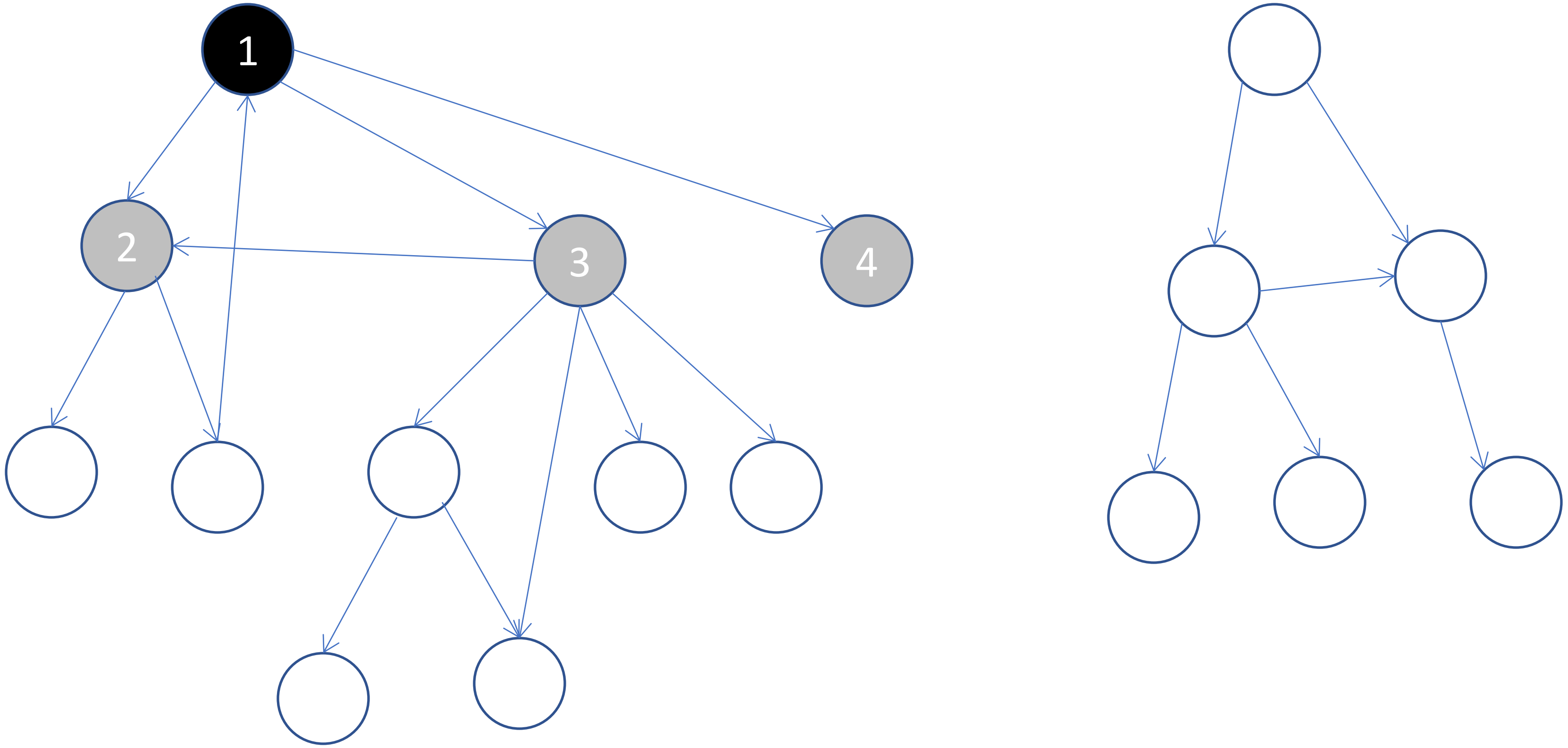
A BFS example (2)

Node 1 does not have any more white neighbour so goes black …



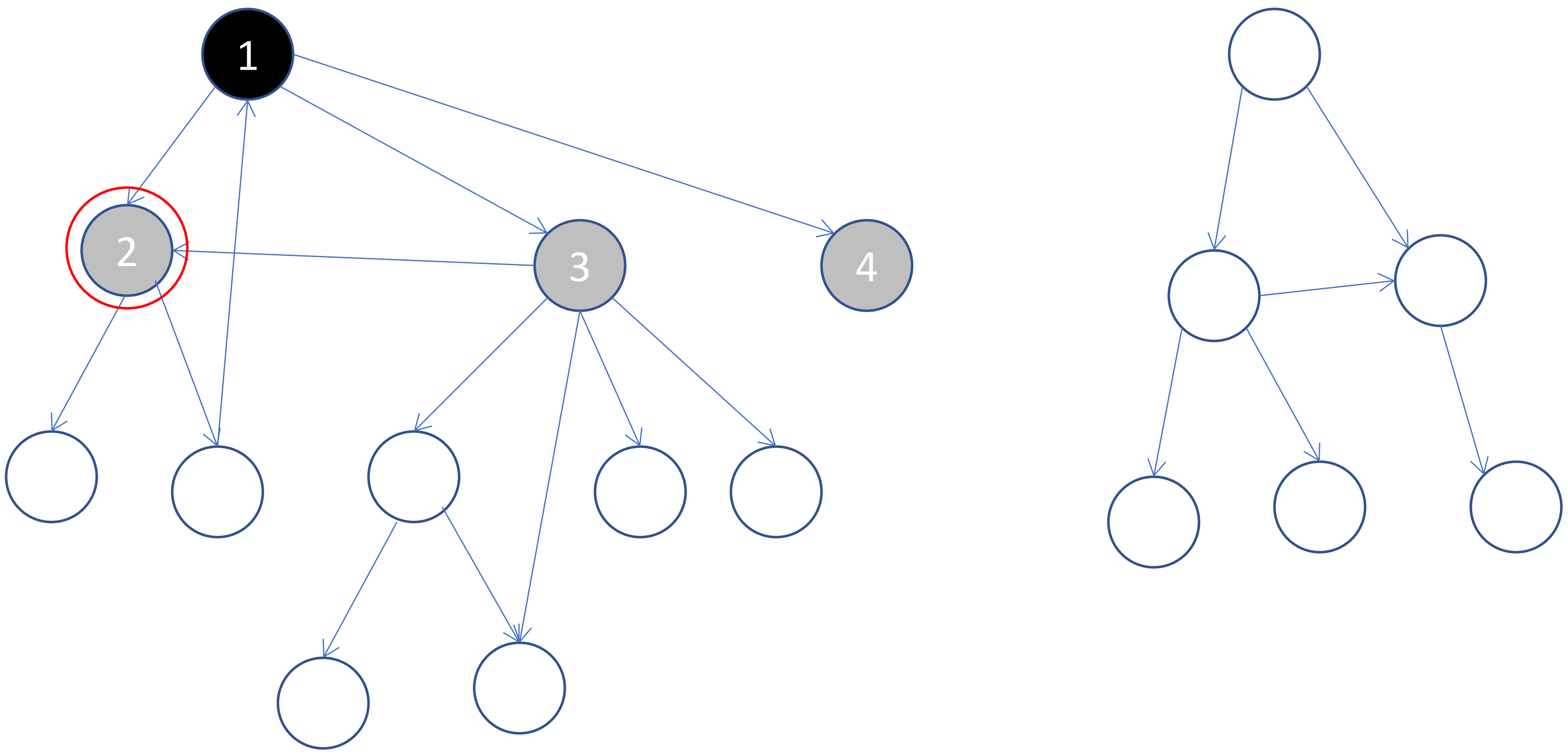
A BFS example (2)

Next grey node the inner loop selects is the oldest one. Which is it?

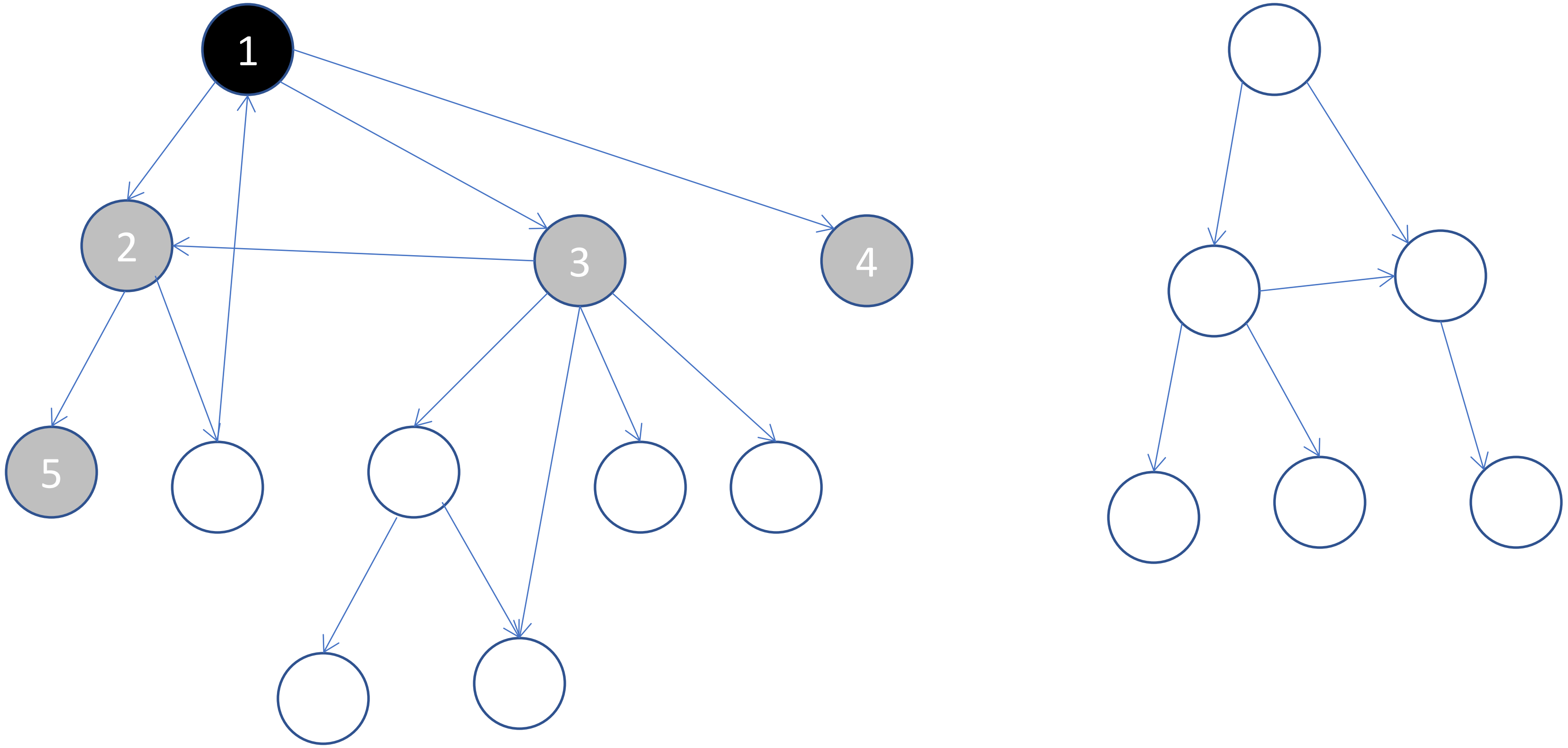


A BFS example (2)

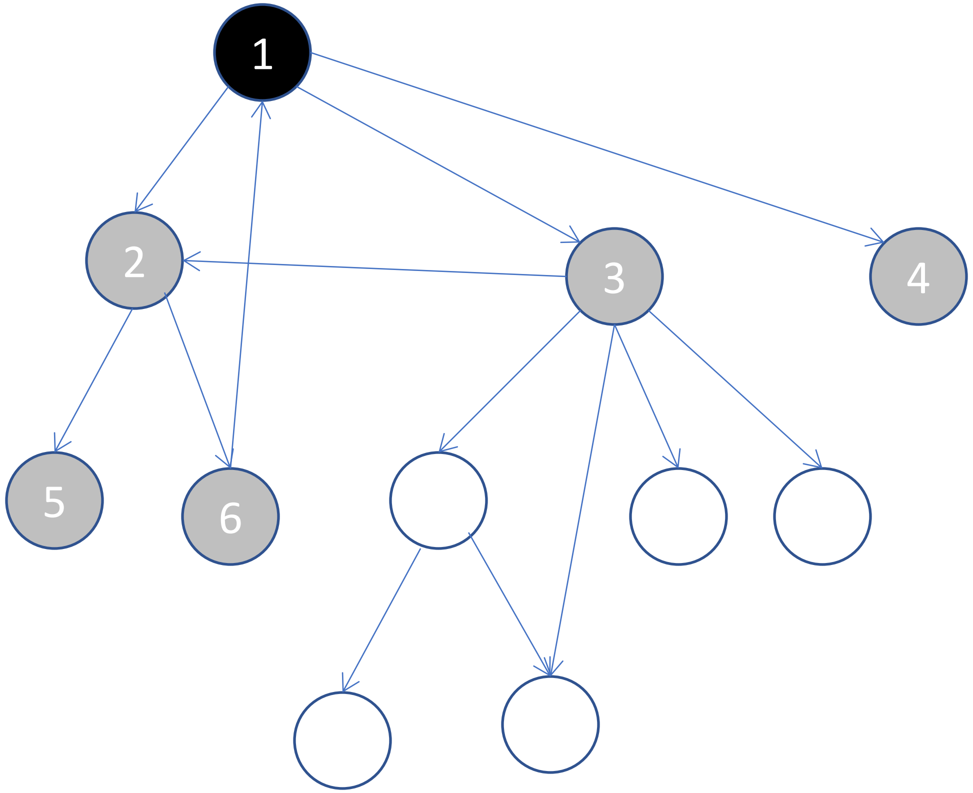
Next grey node the inner loop selects is the oldest one. 2

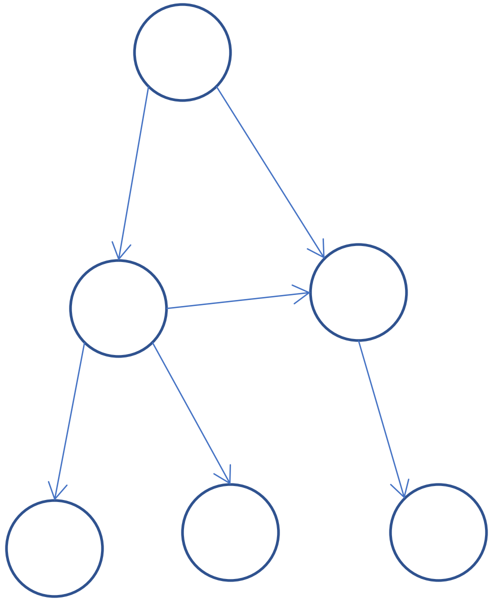


A BFS example (2)



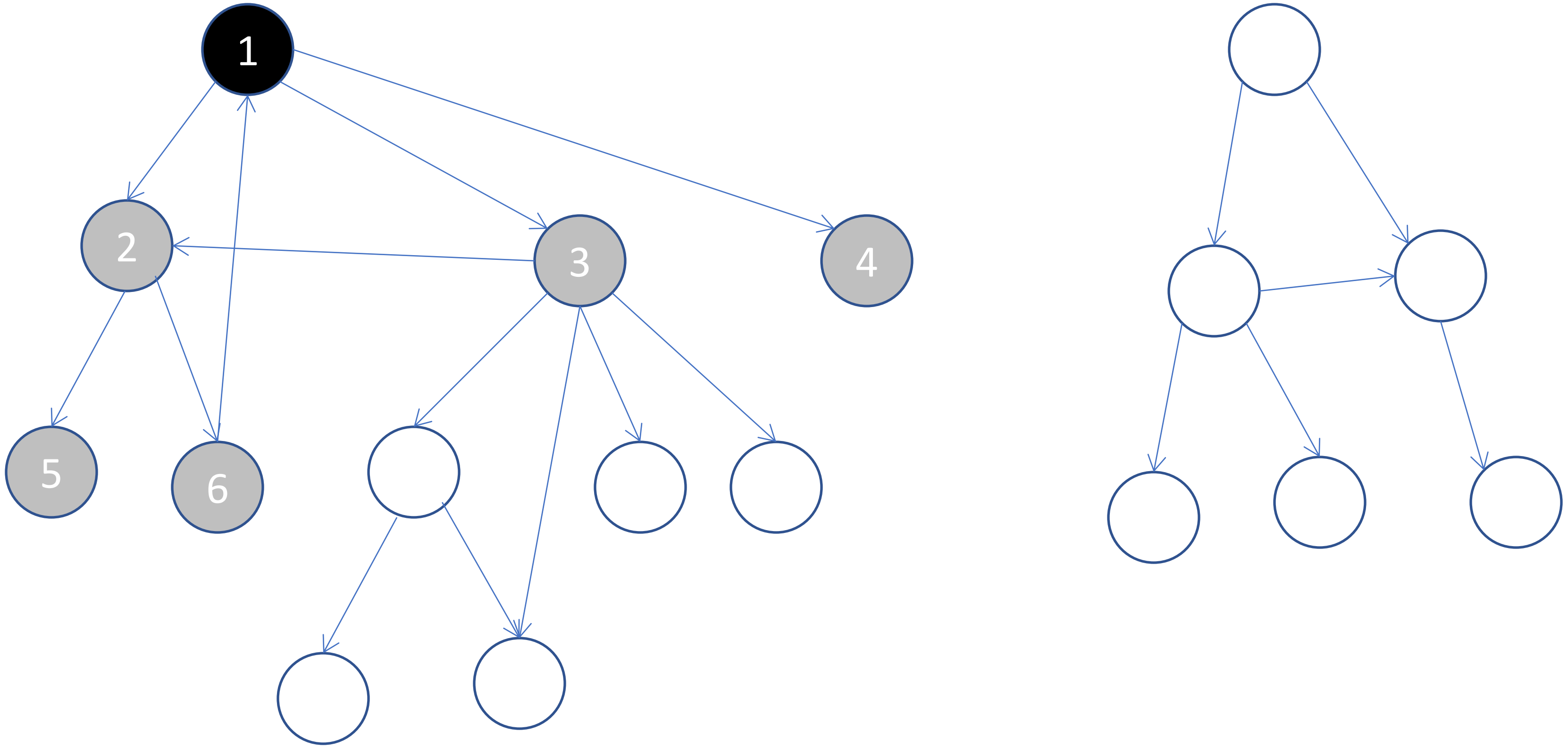
A BFS example (2)



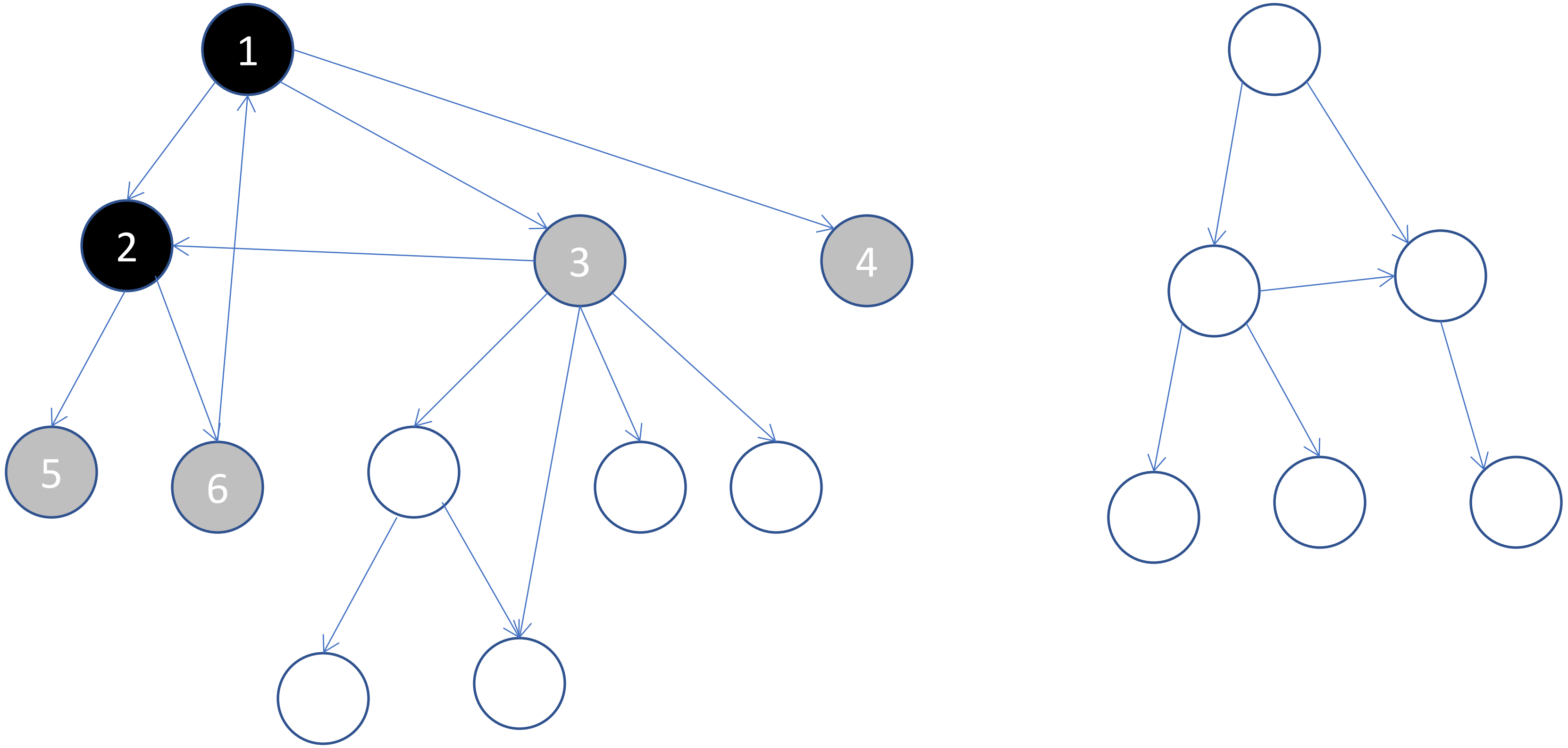


A BFS example (2)

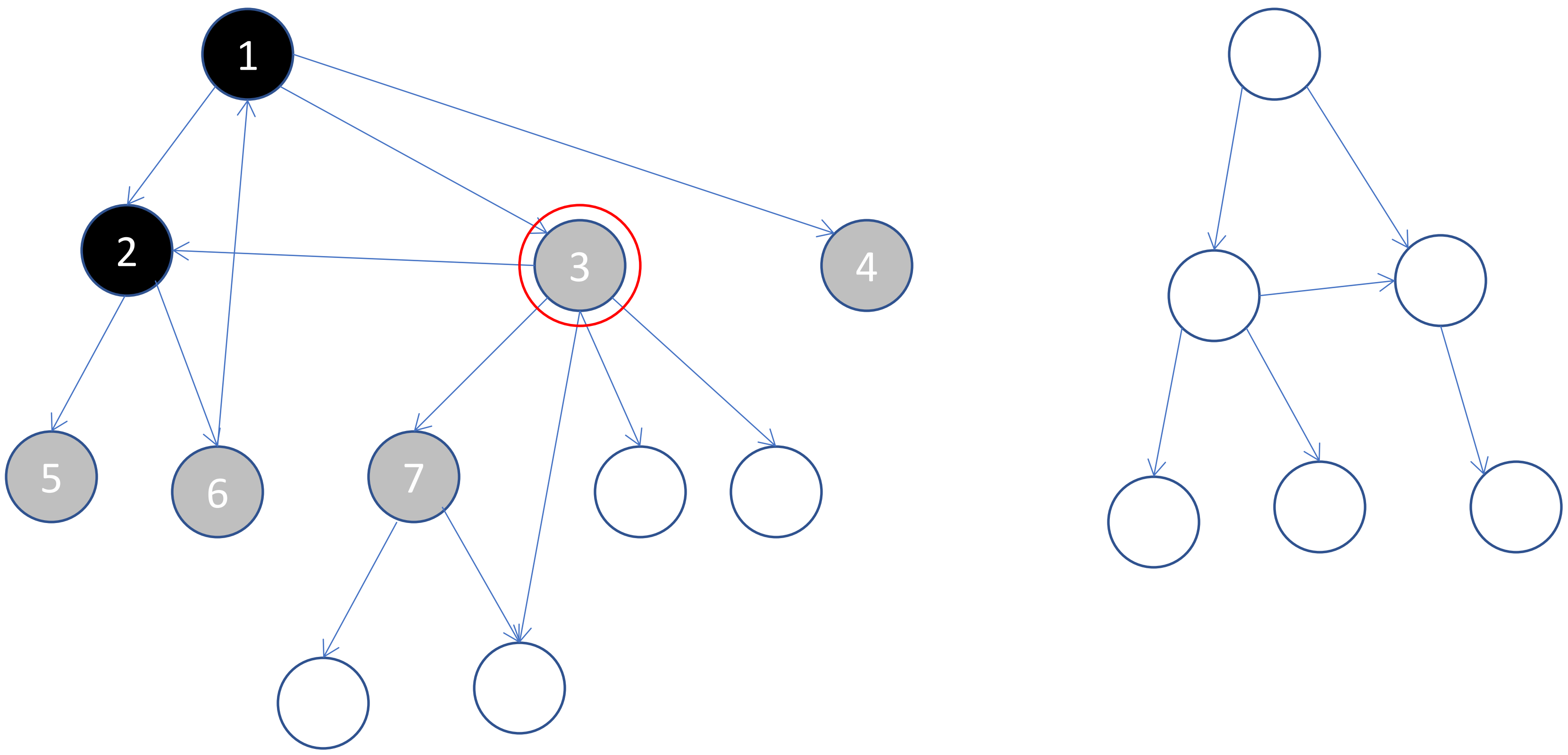
Any more white nodes from 2?



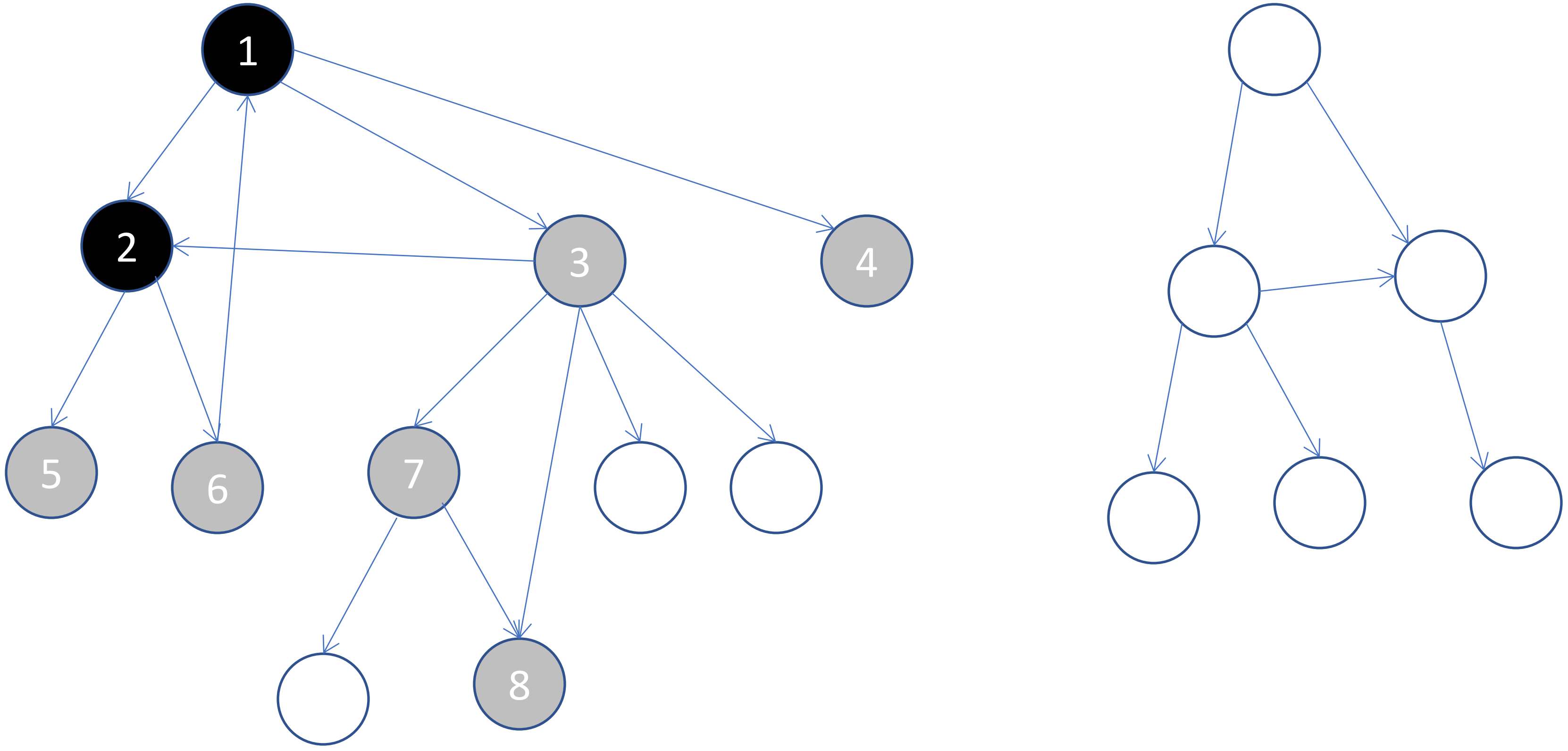
A BFS example (2)



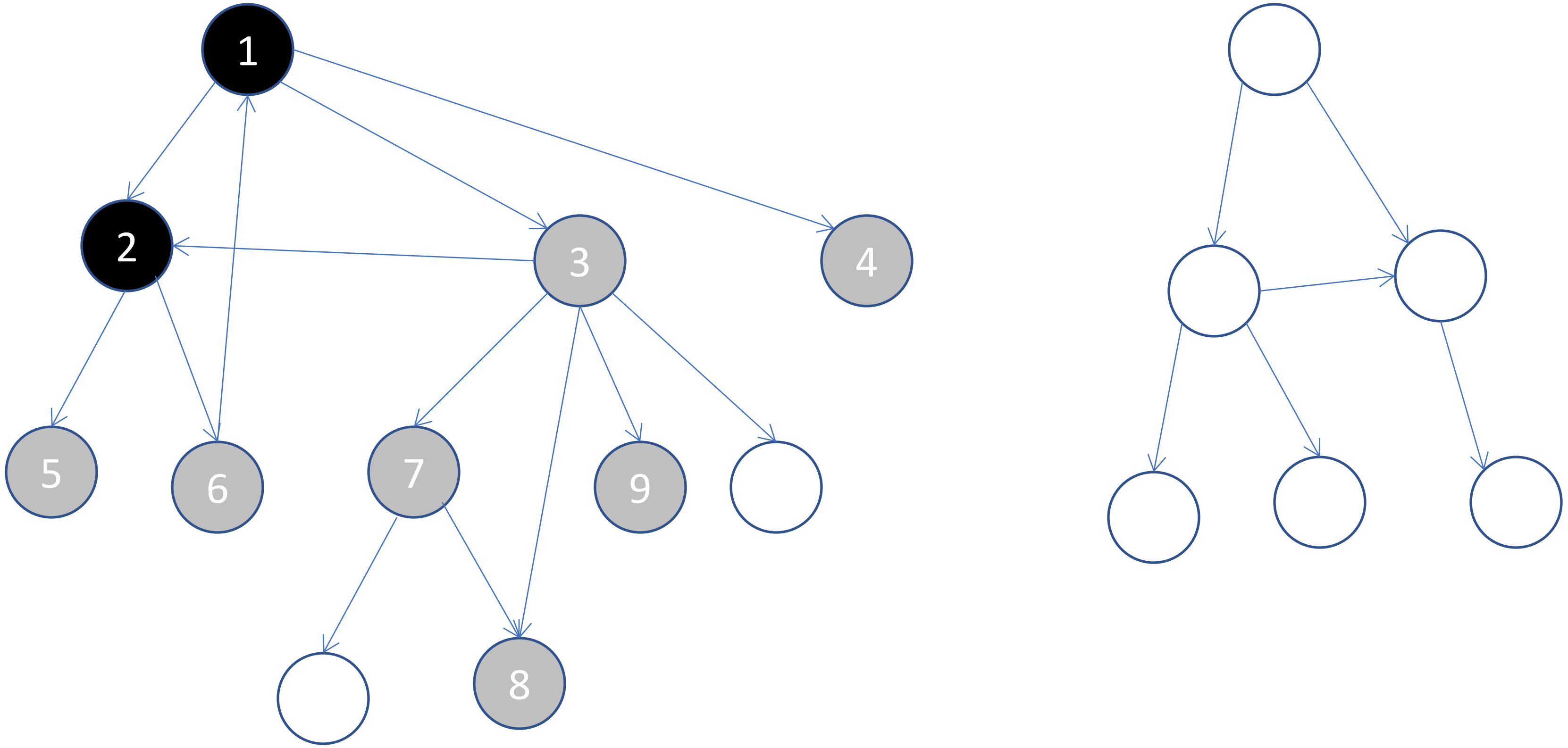
A BFS example (2)



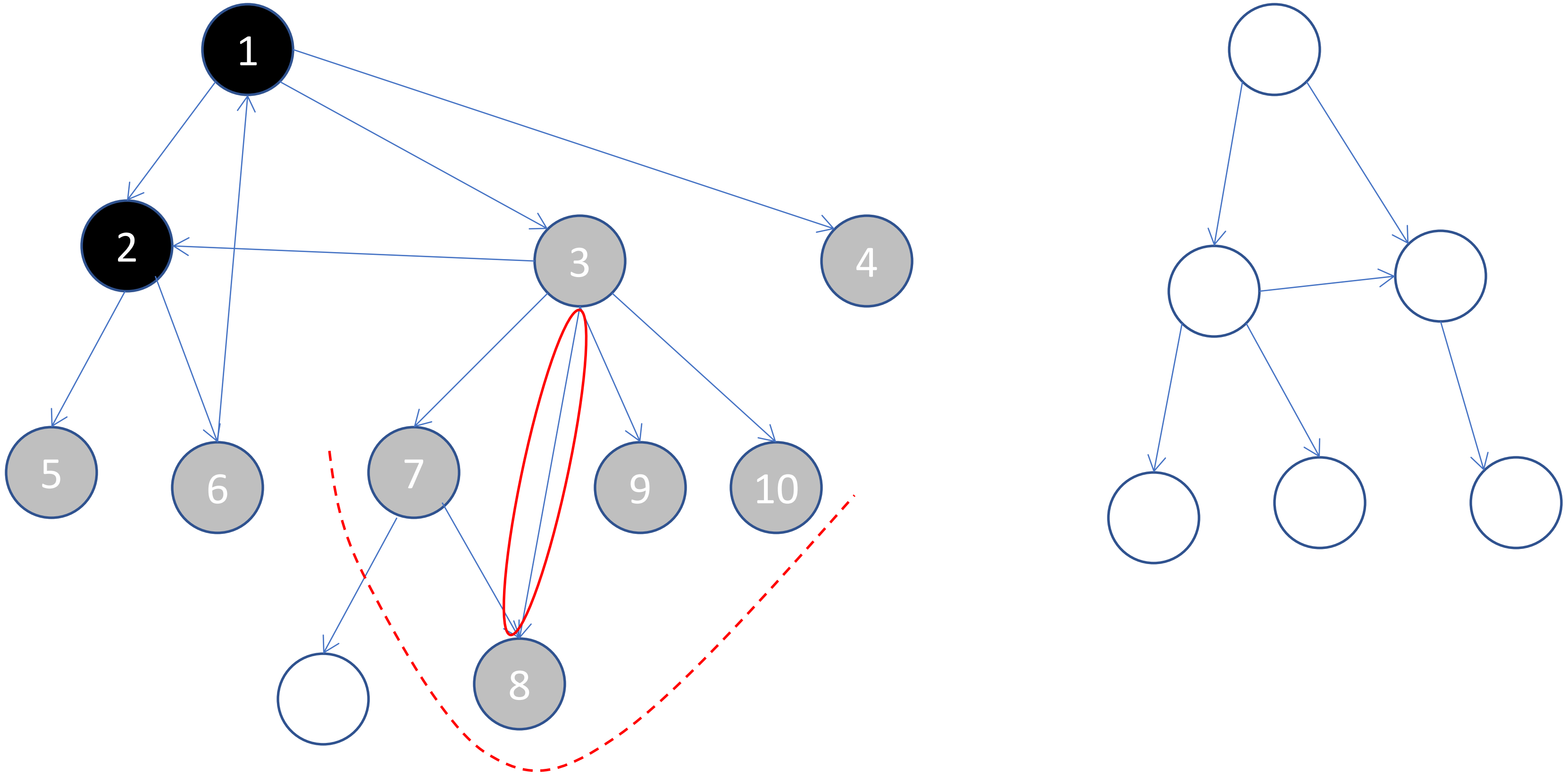
A BFS example (2)



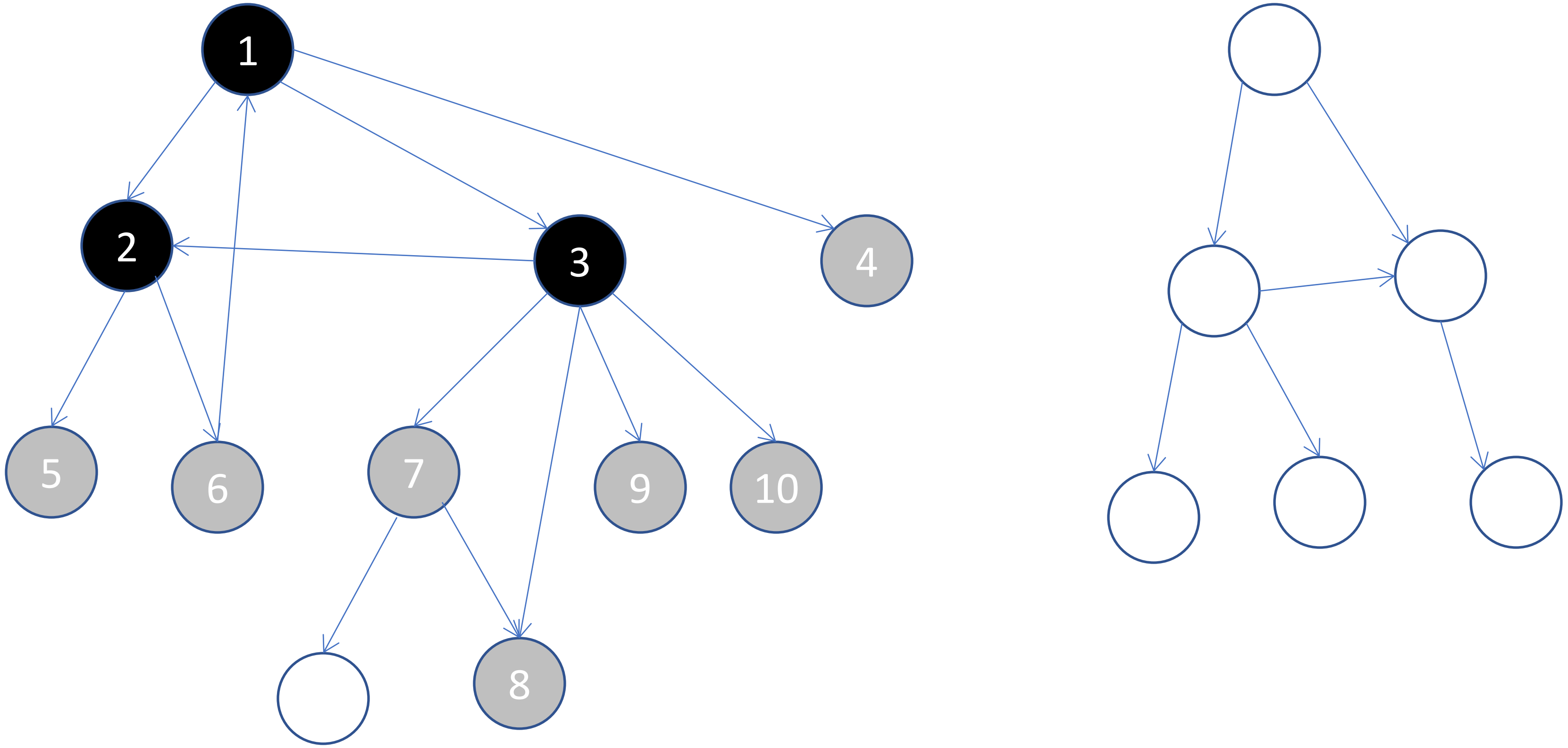
A BFS example (2)



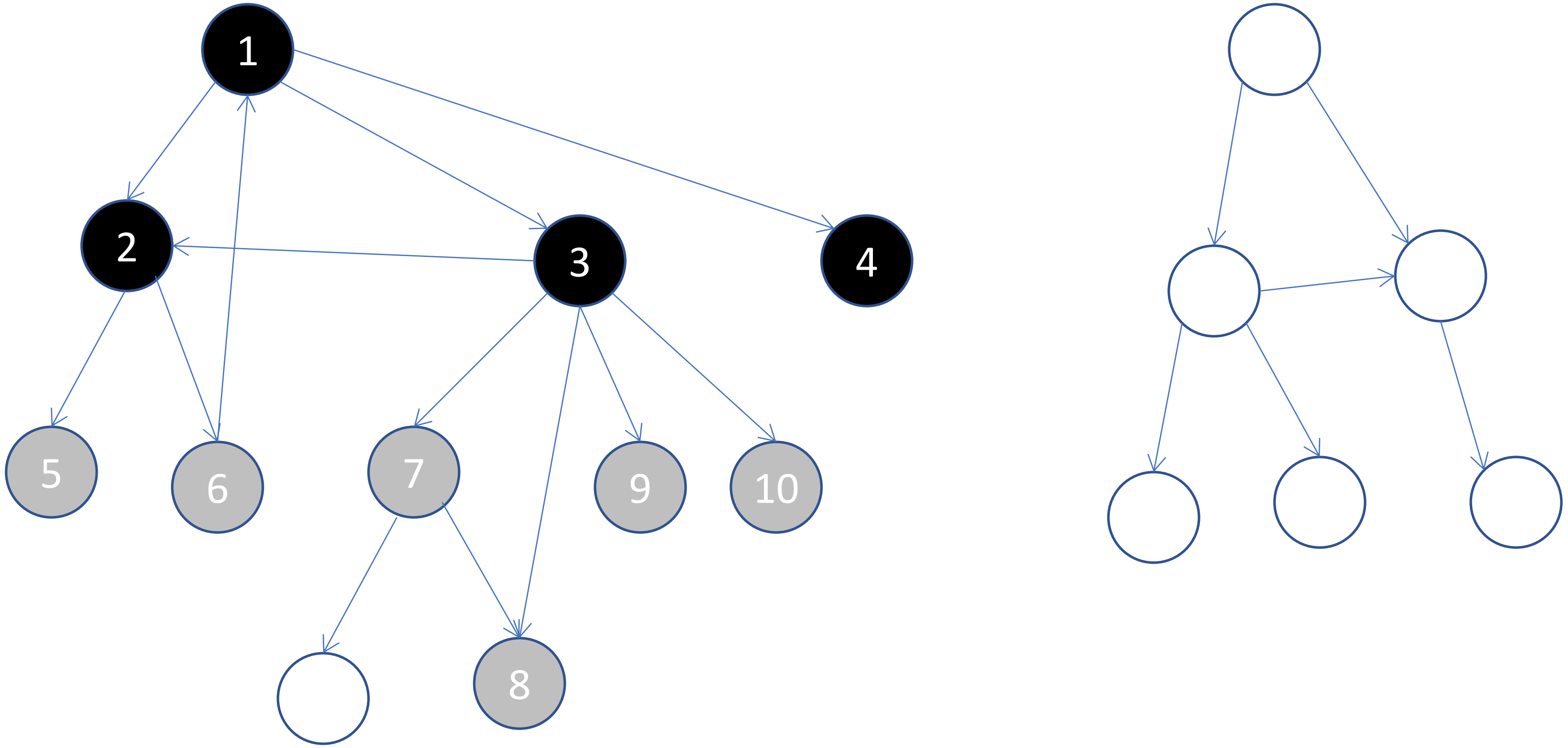
A BFS example (2)



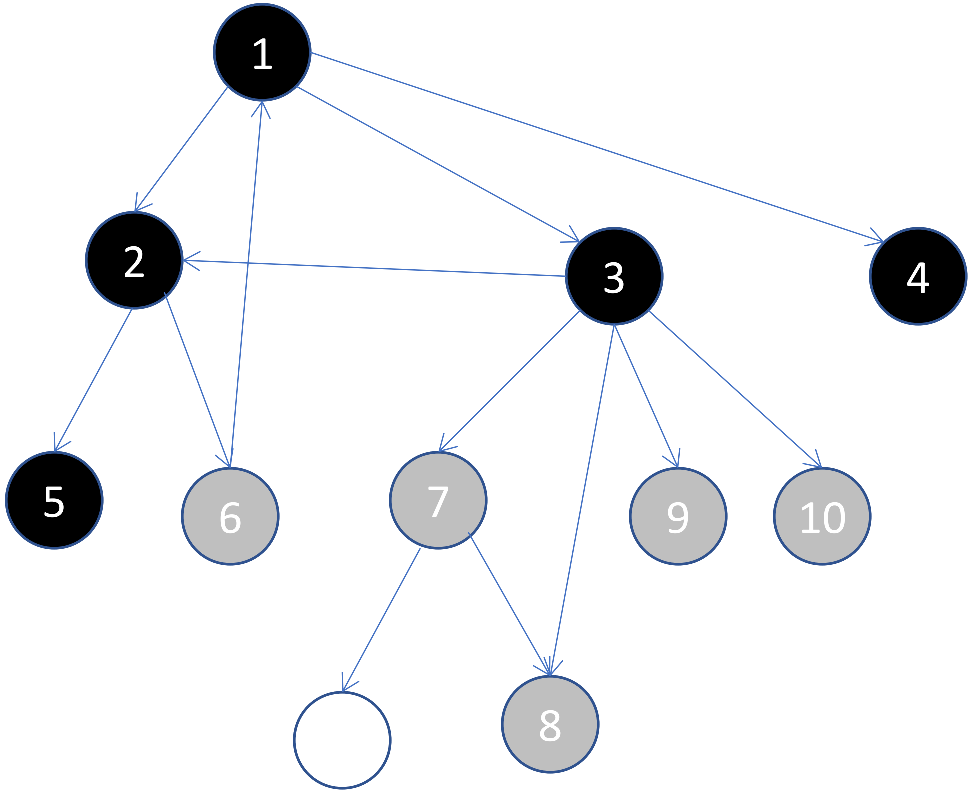
A BFS example (2)

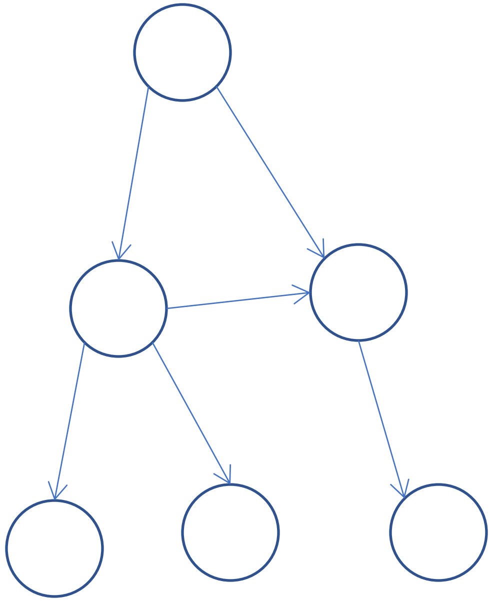


A BFS example (2)

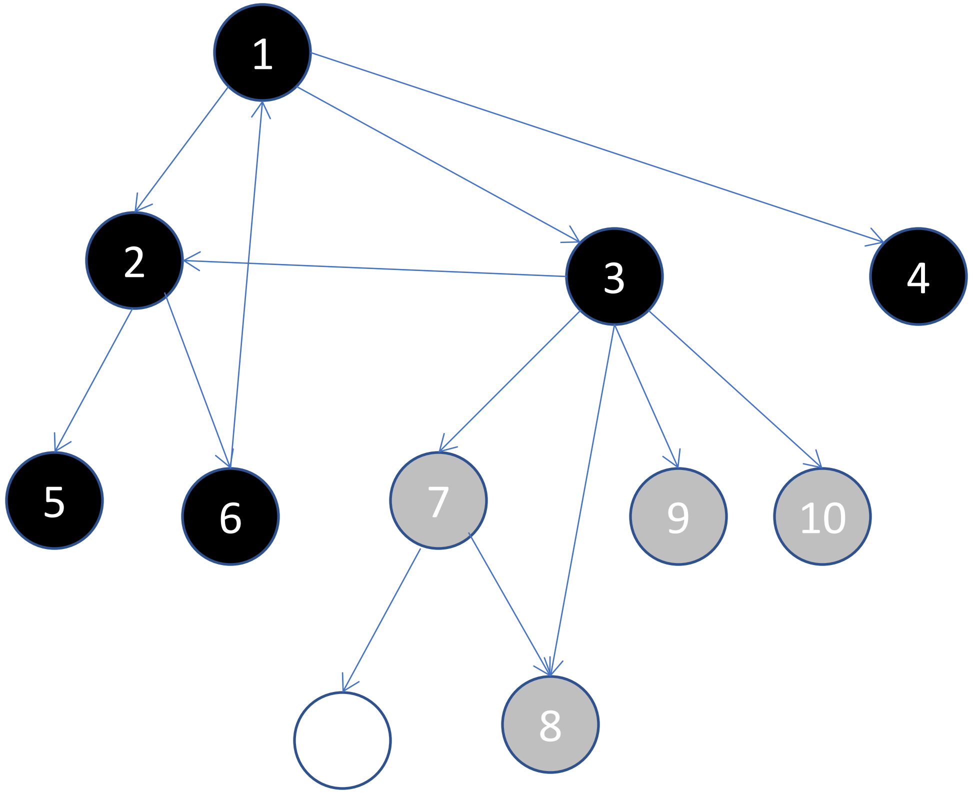


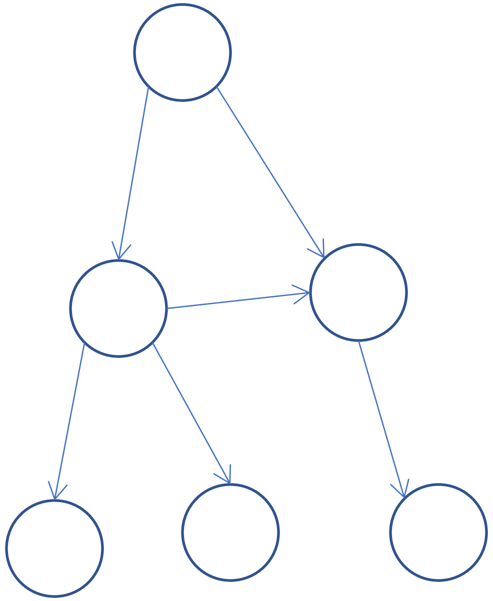
A BFS example (2)



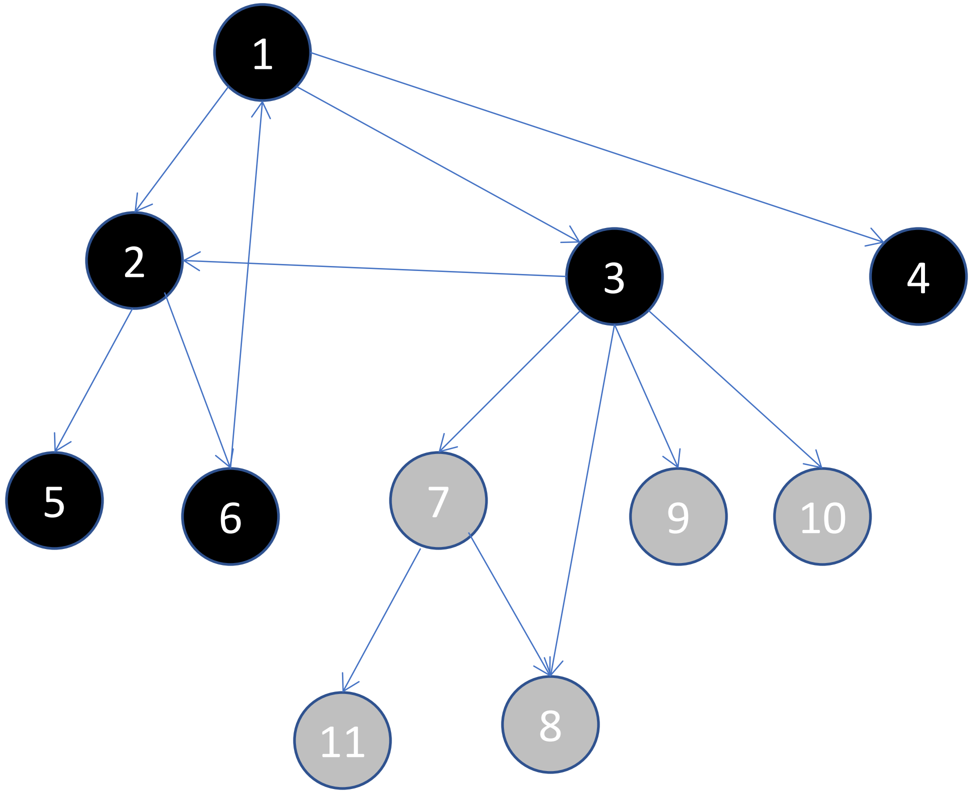


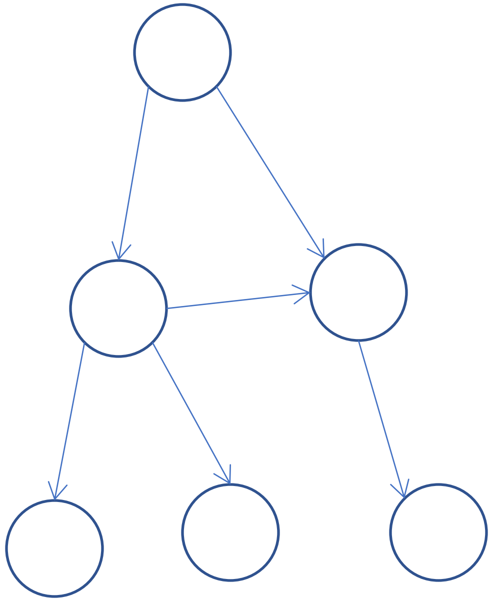
A BFS example (2)



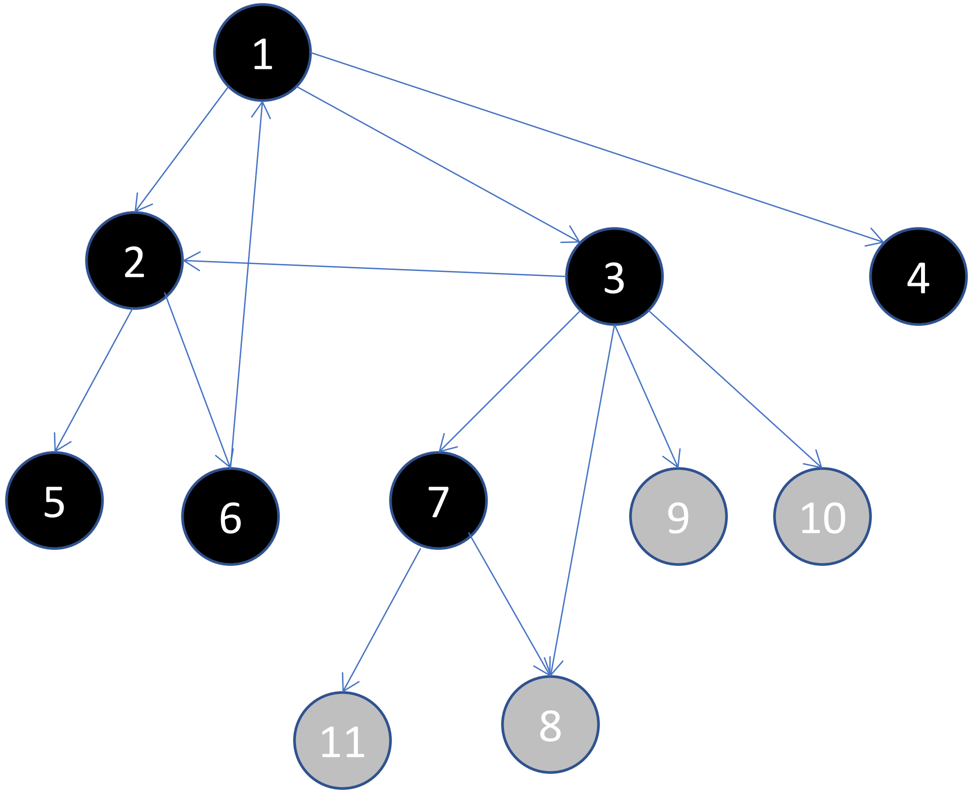


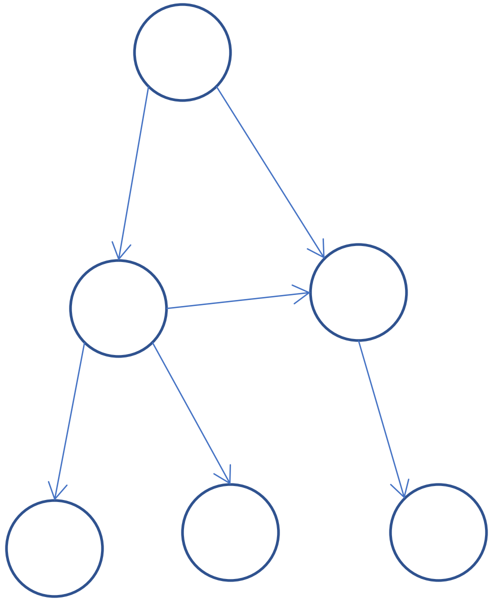
A BFS example (2)



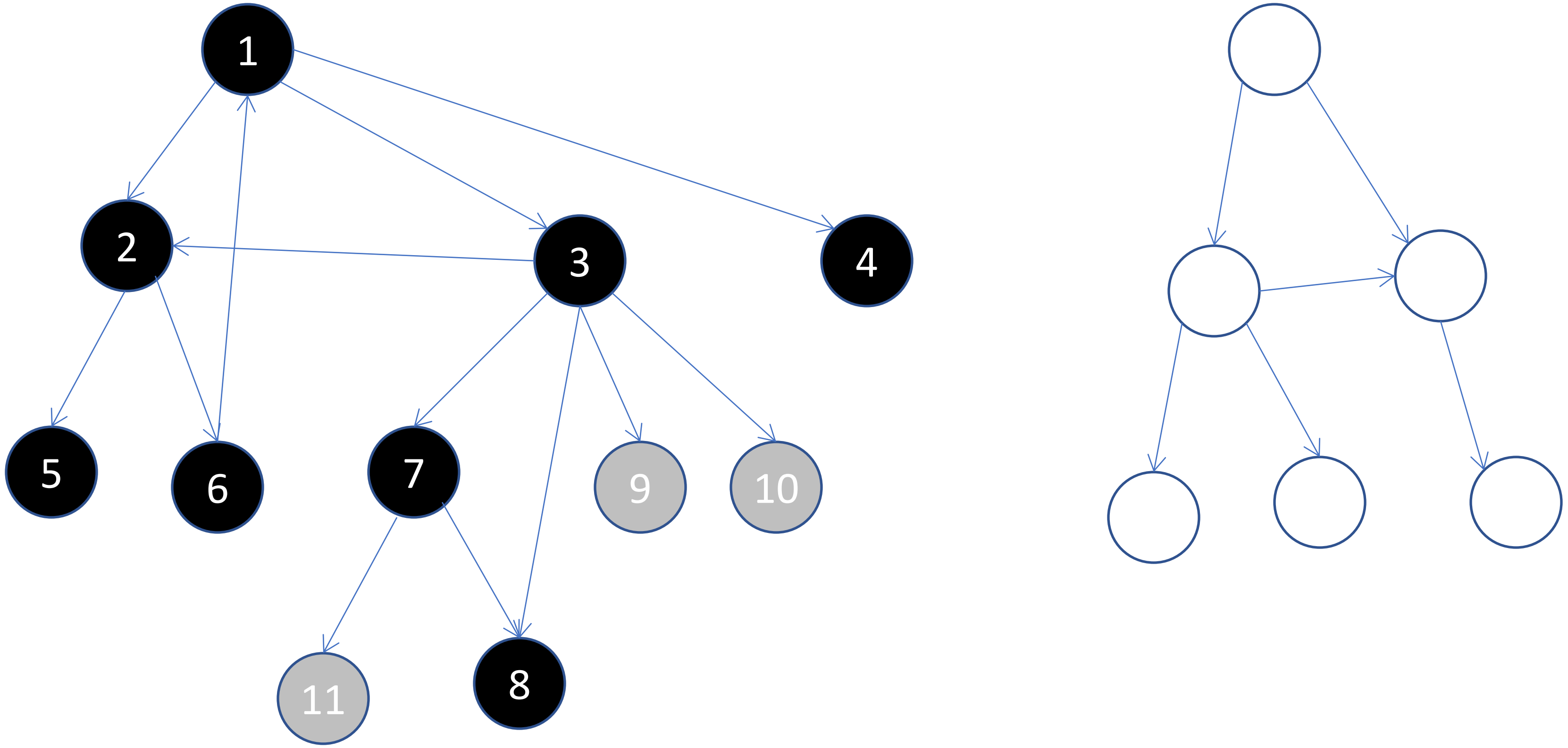


A BFS example (2)

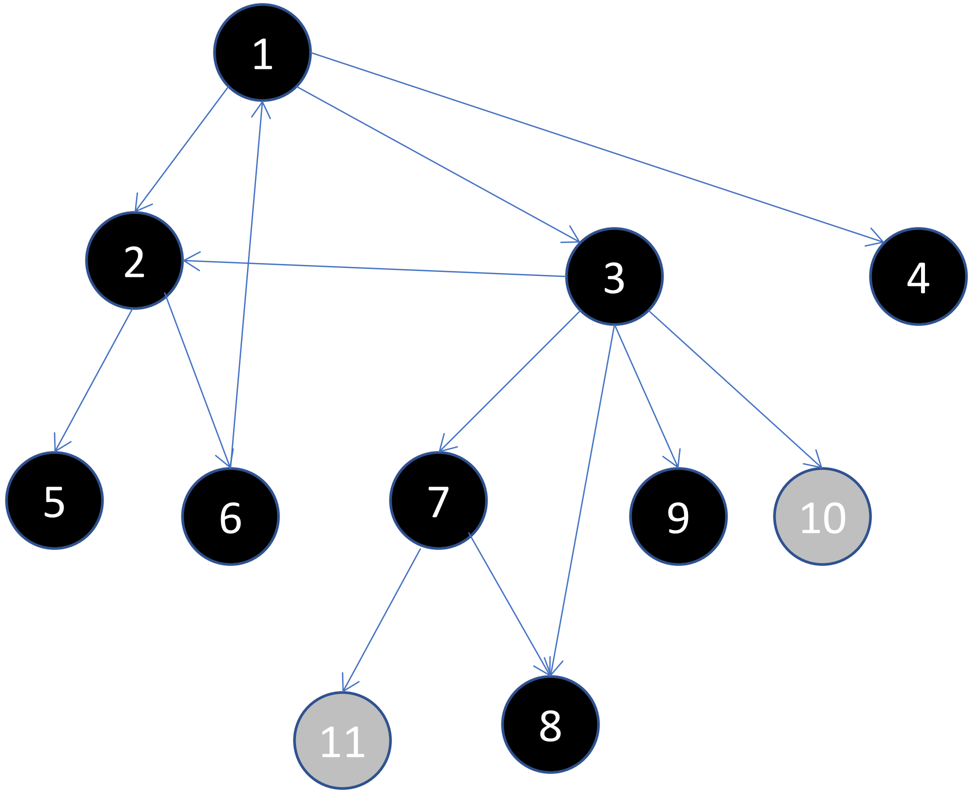


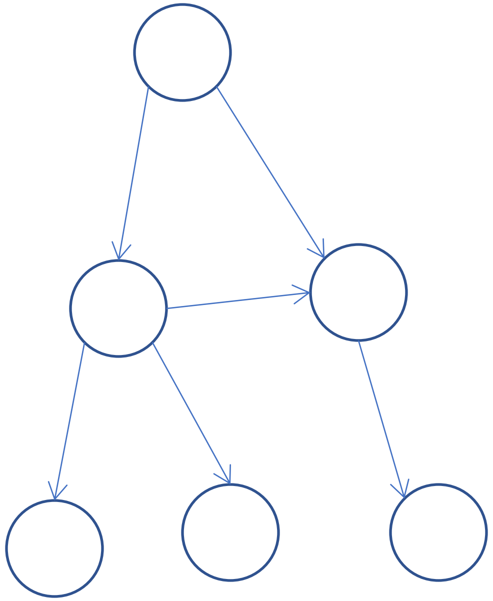


A BFS example (2)

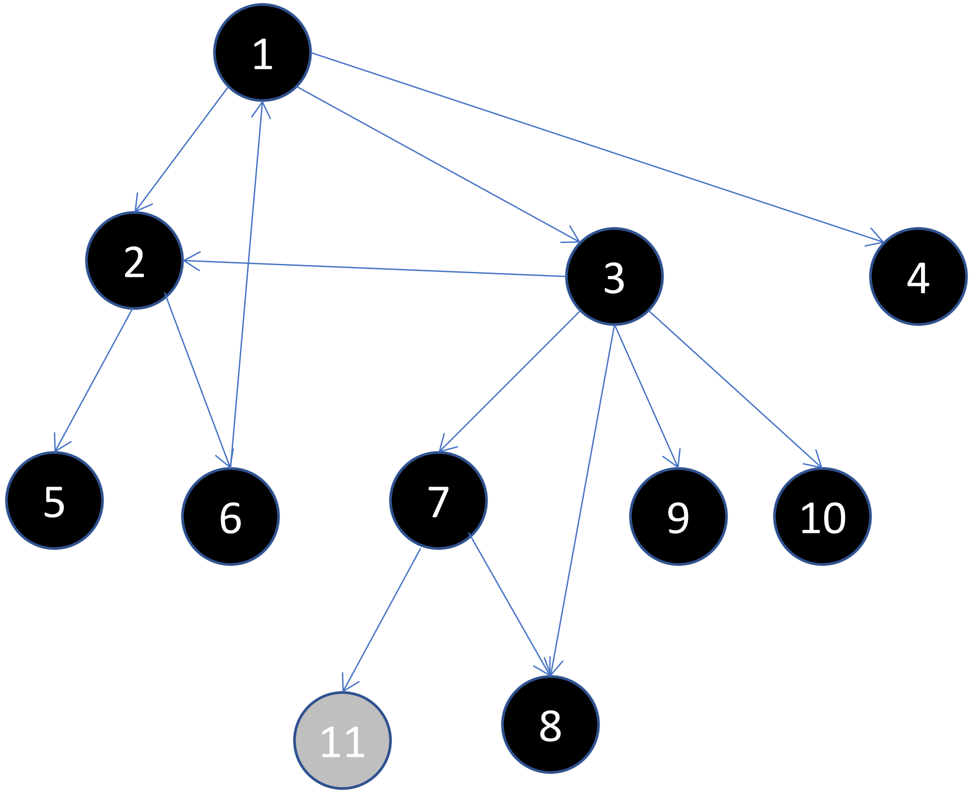


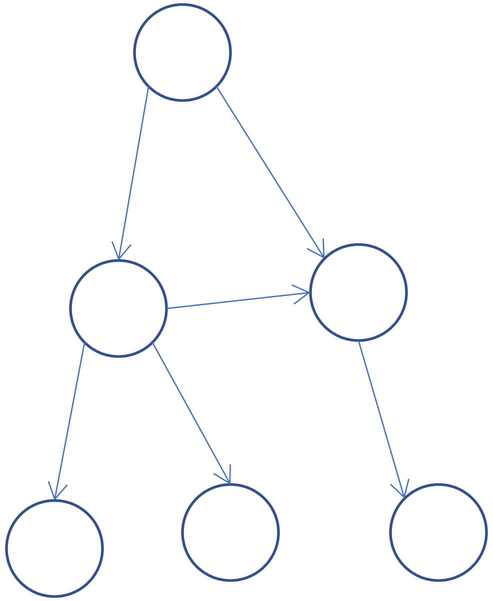
A BFS example (2)



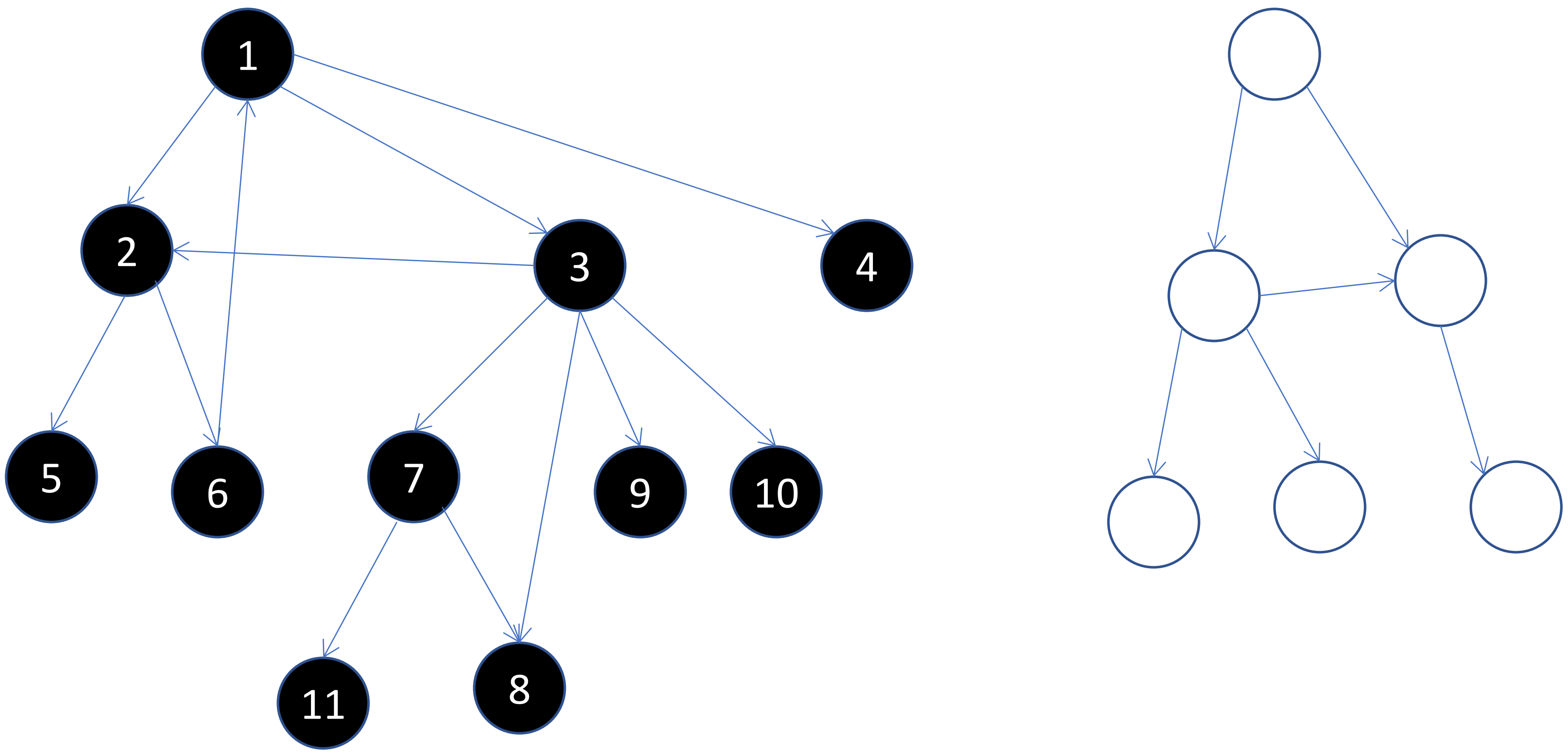


A BFS example (2)

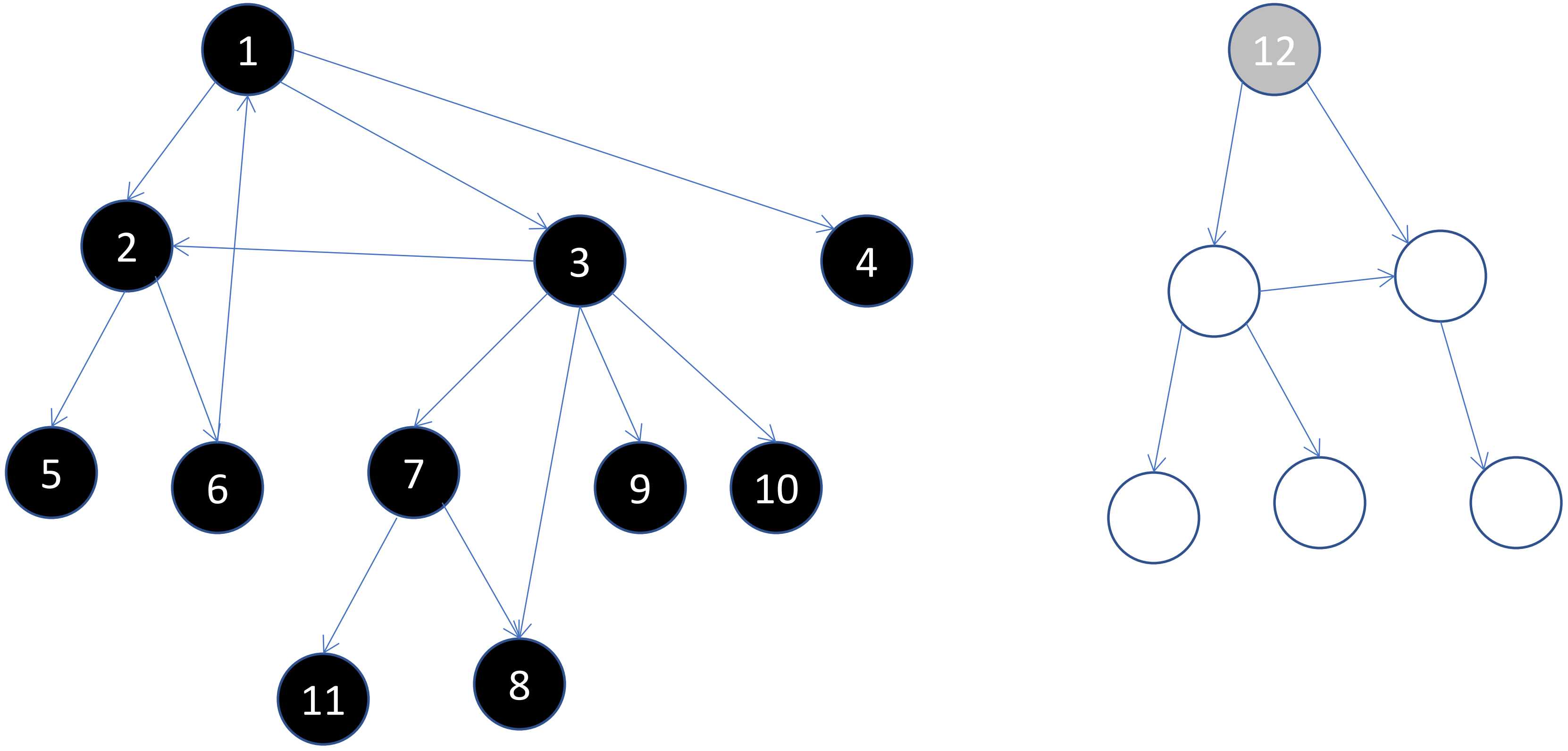




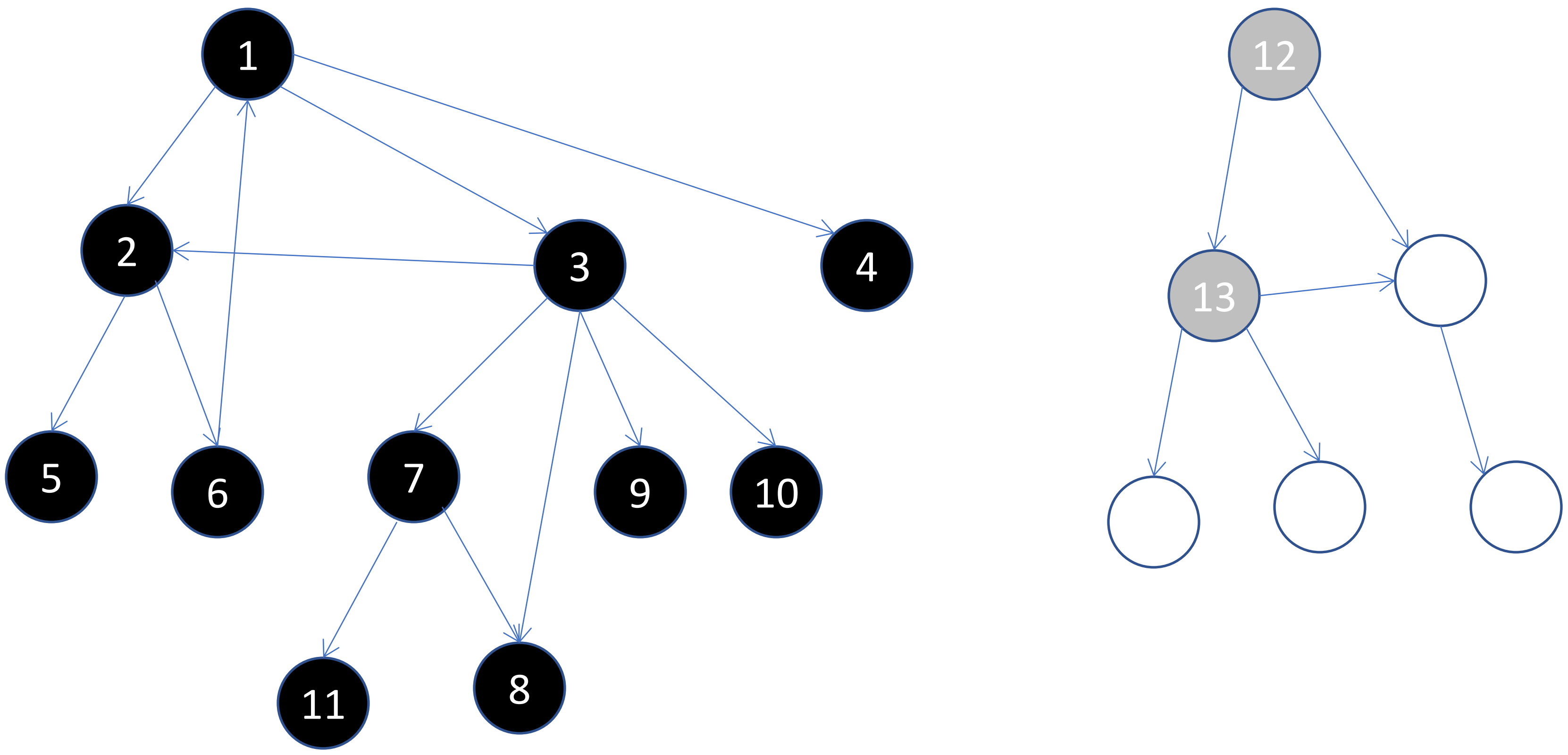
A BFS example (2)



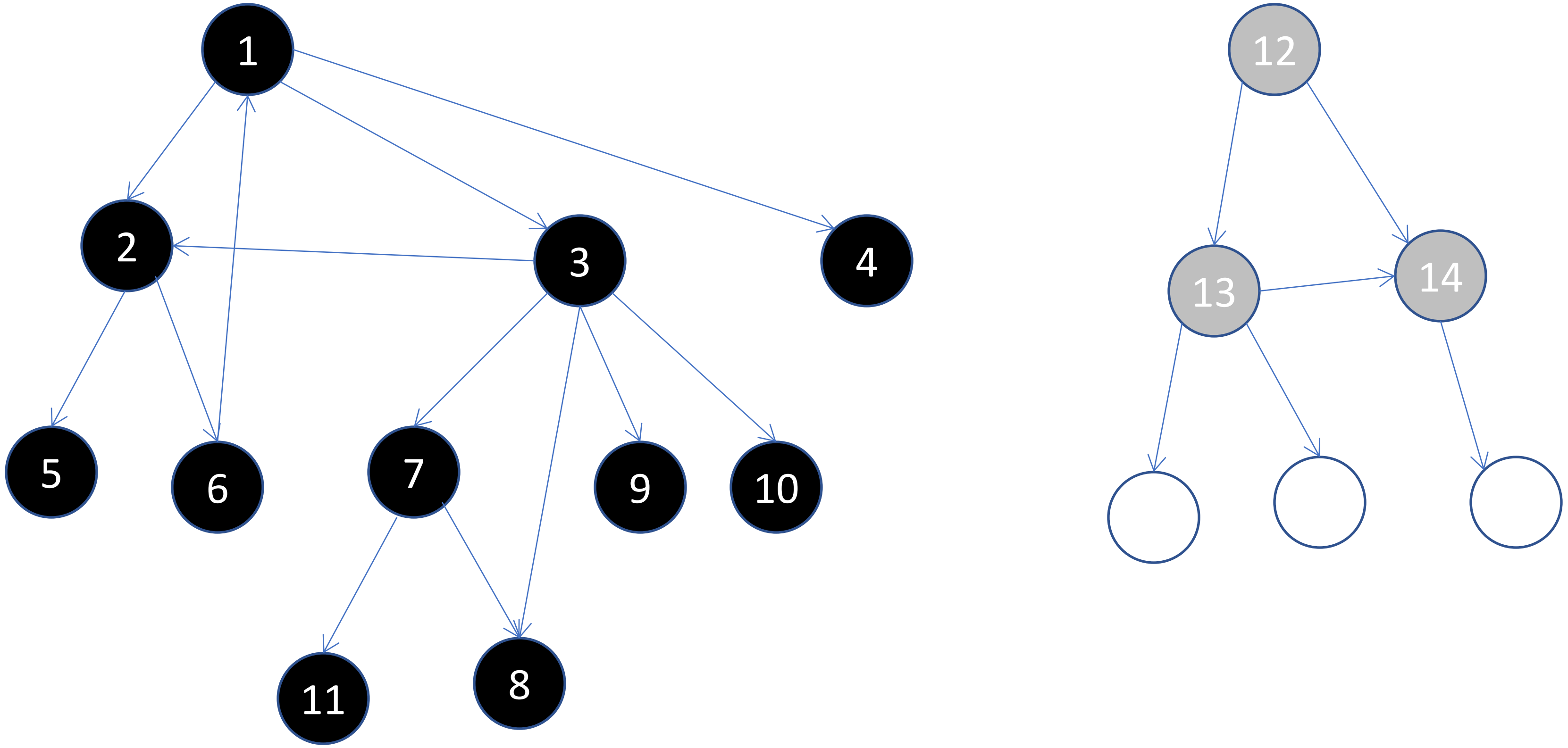
A BFS example (2)



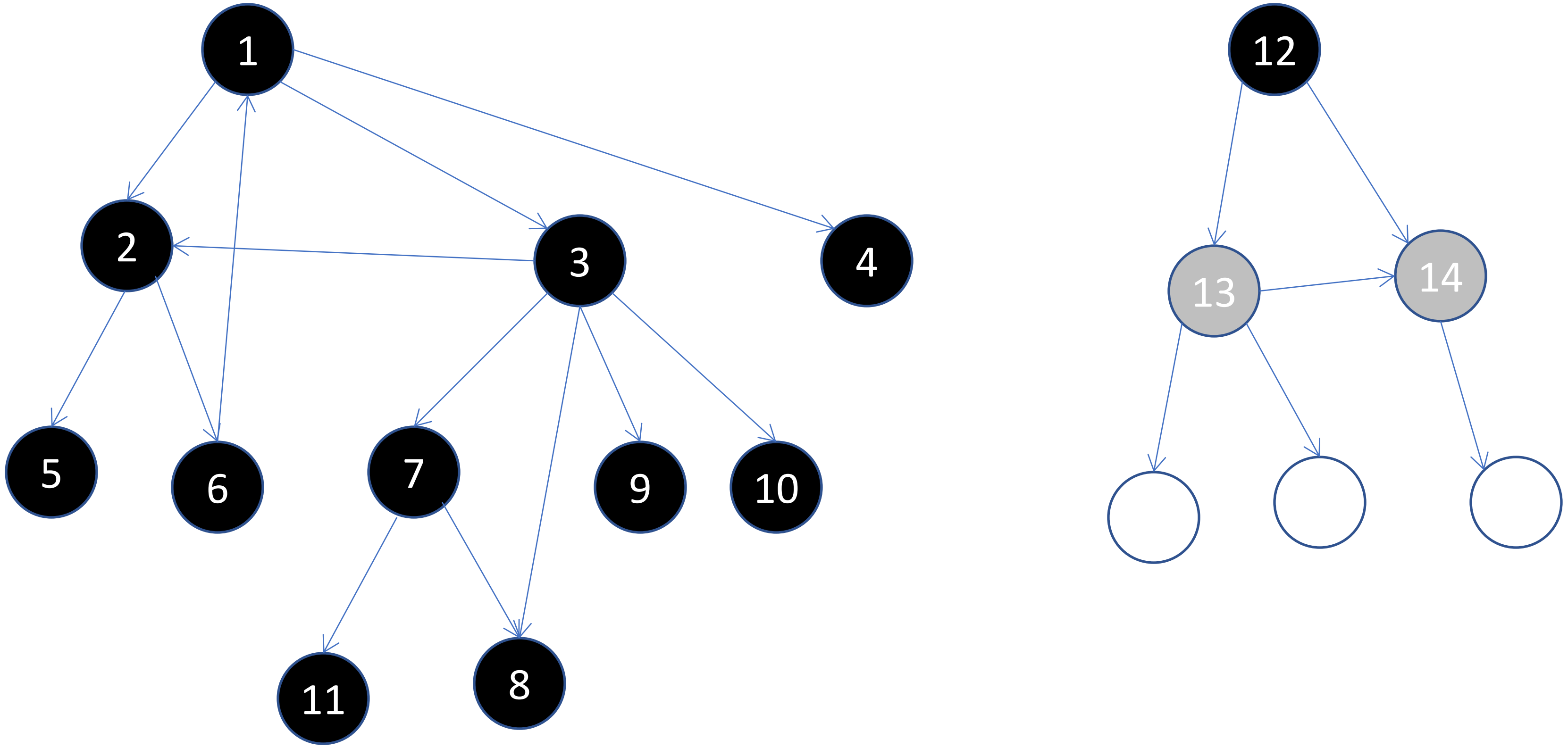
A BFS example (2)



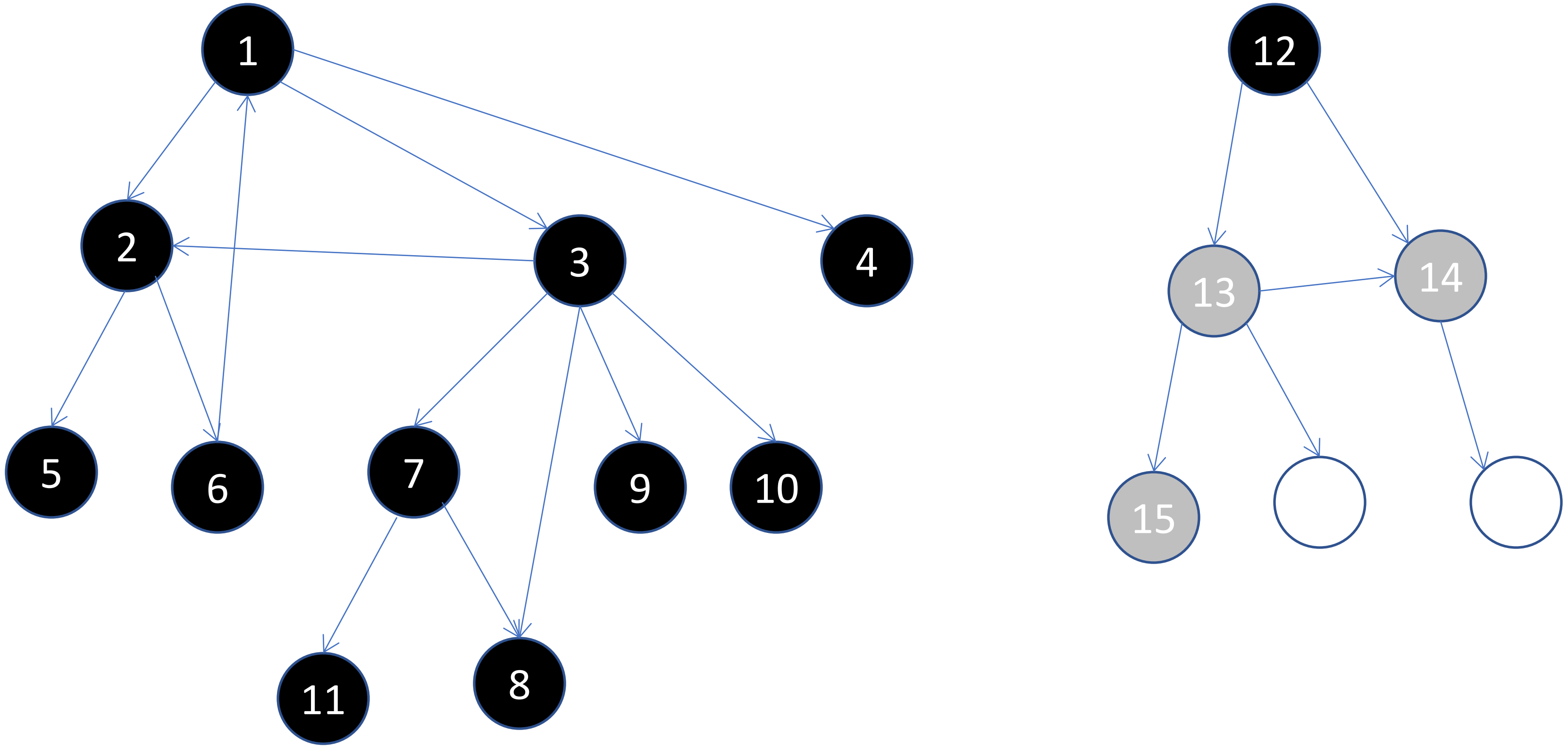
A BFS example (2)



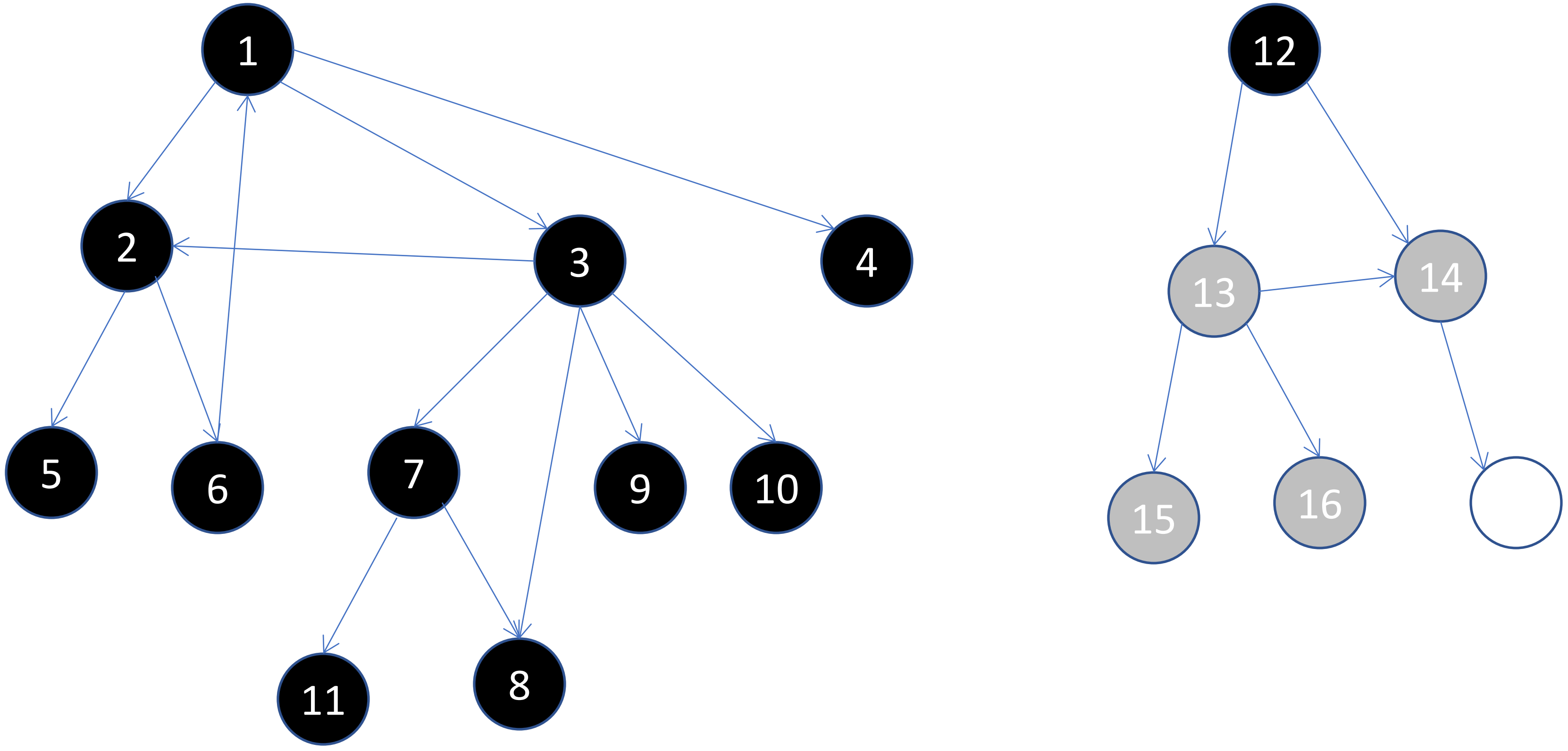
A BFS example (2)



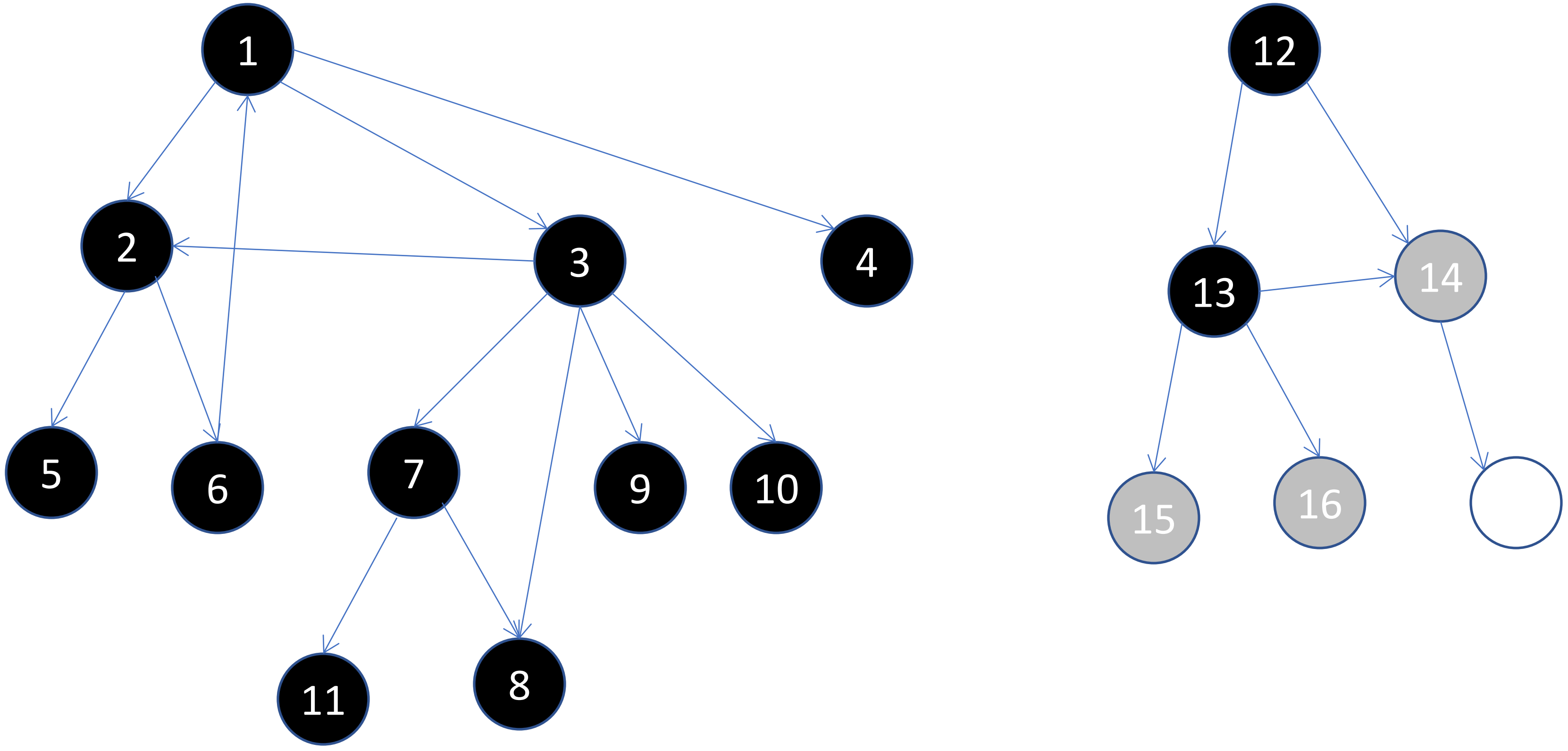
A BFS example (2)



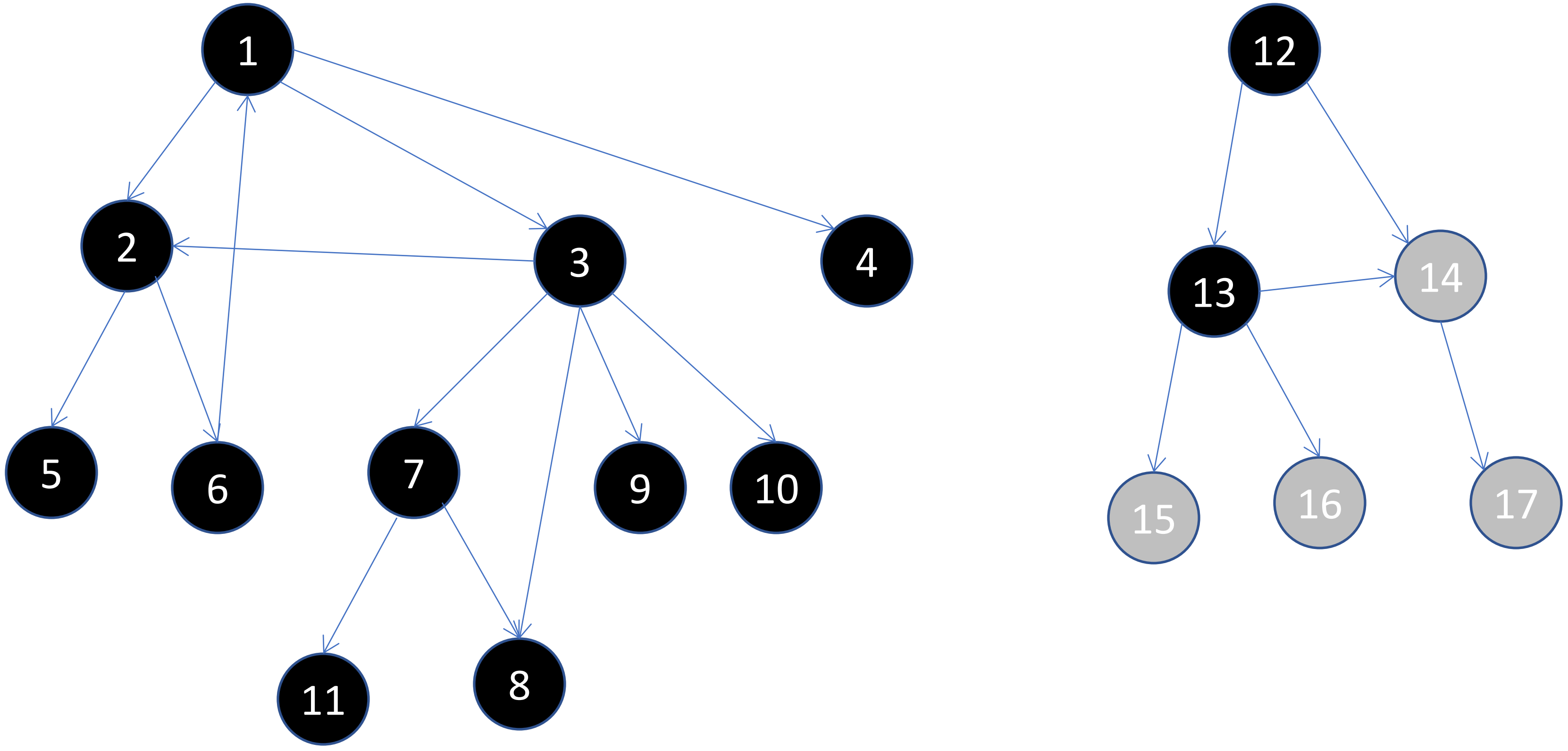
A BFS example (2)



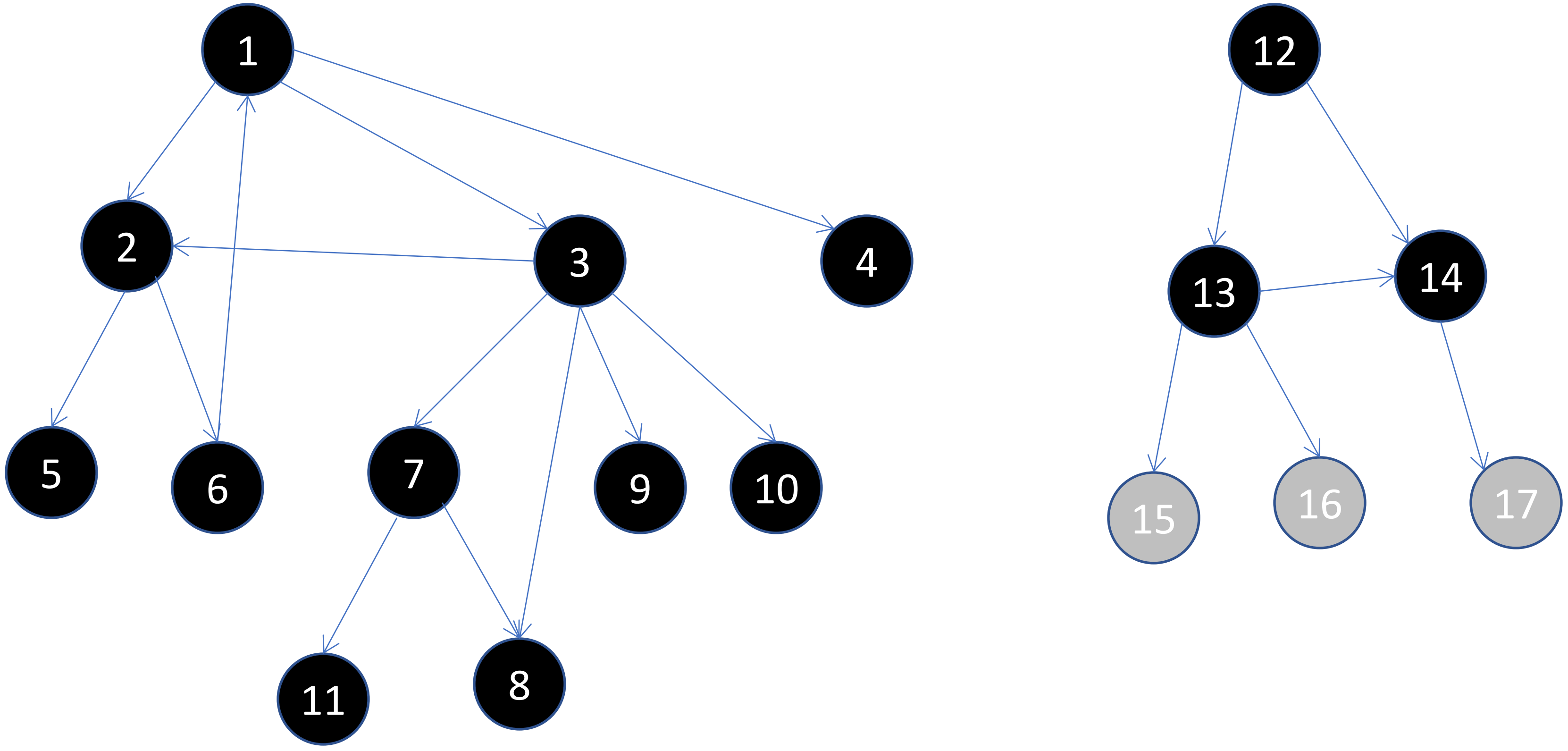
A BFS example (2)



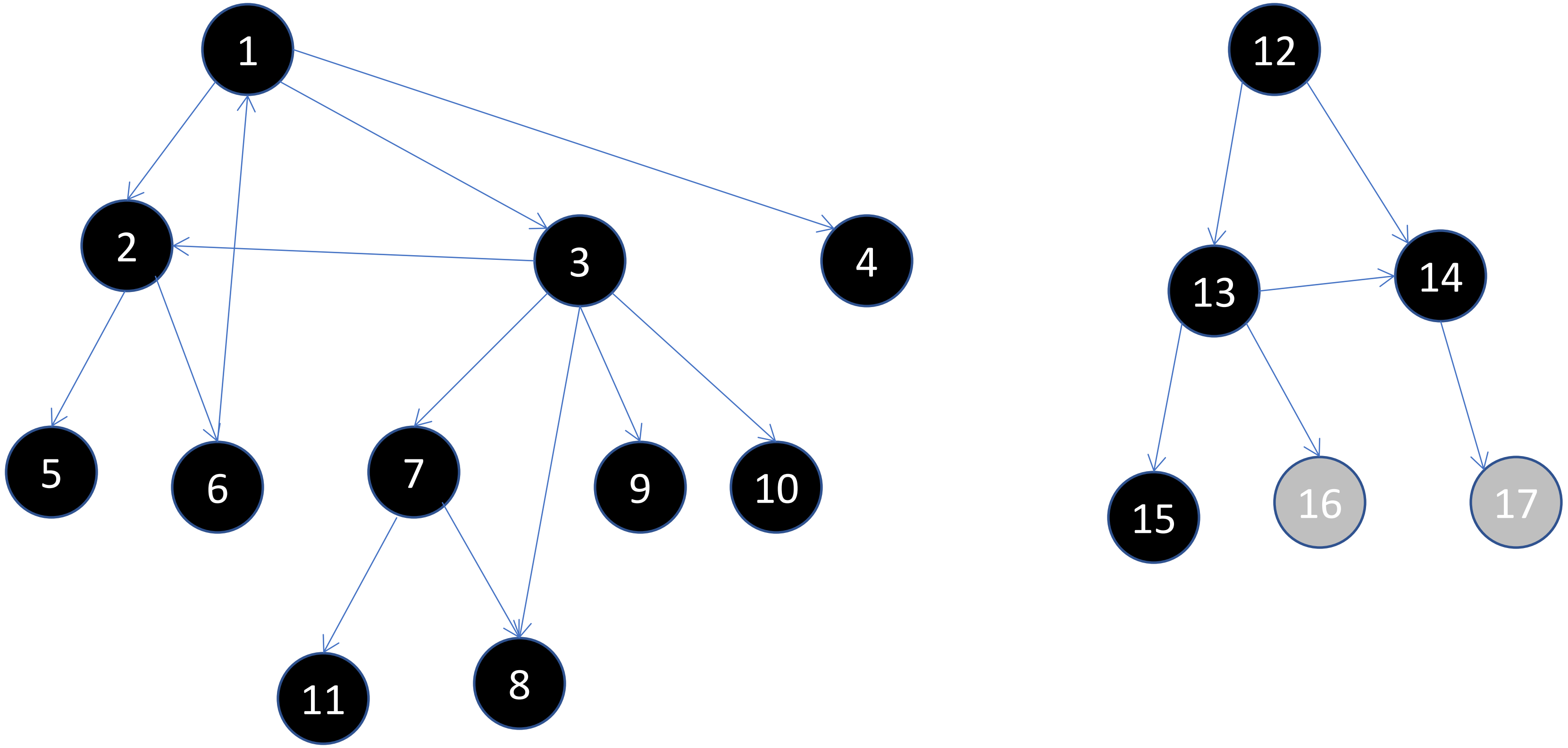
A BFS example (2)



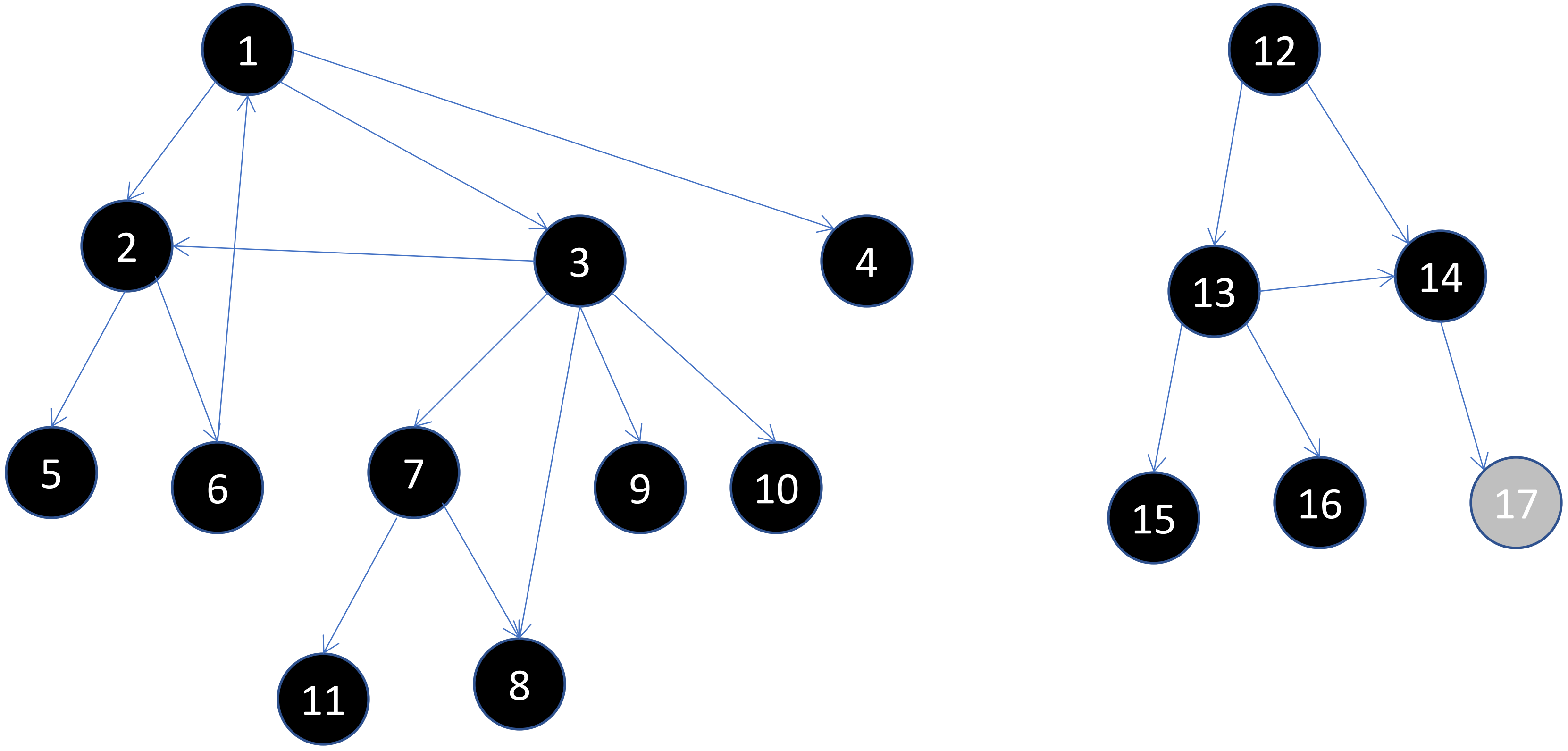
A BFS example (2)



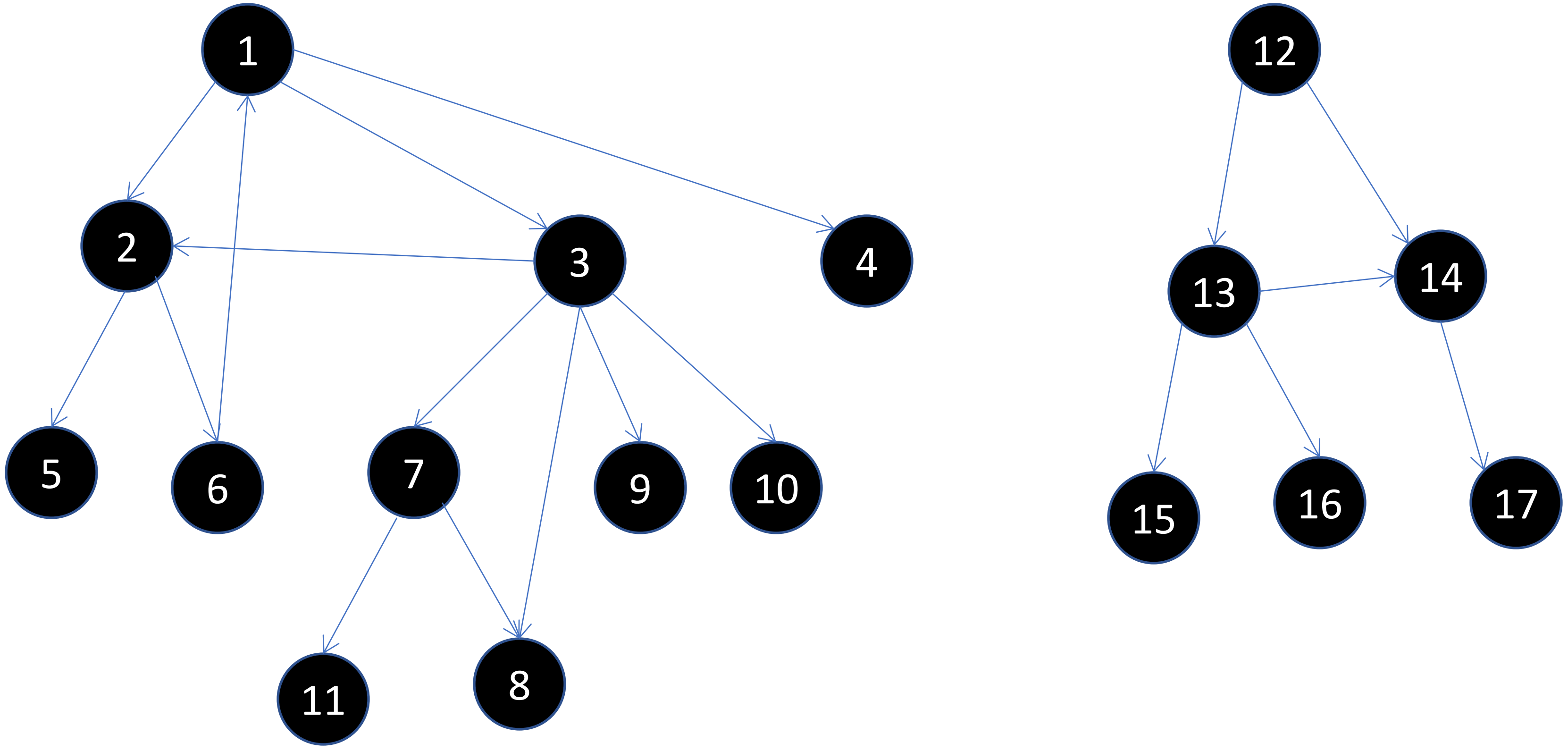
A BFS example (2)



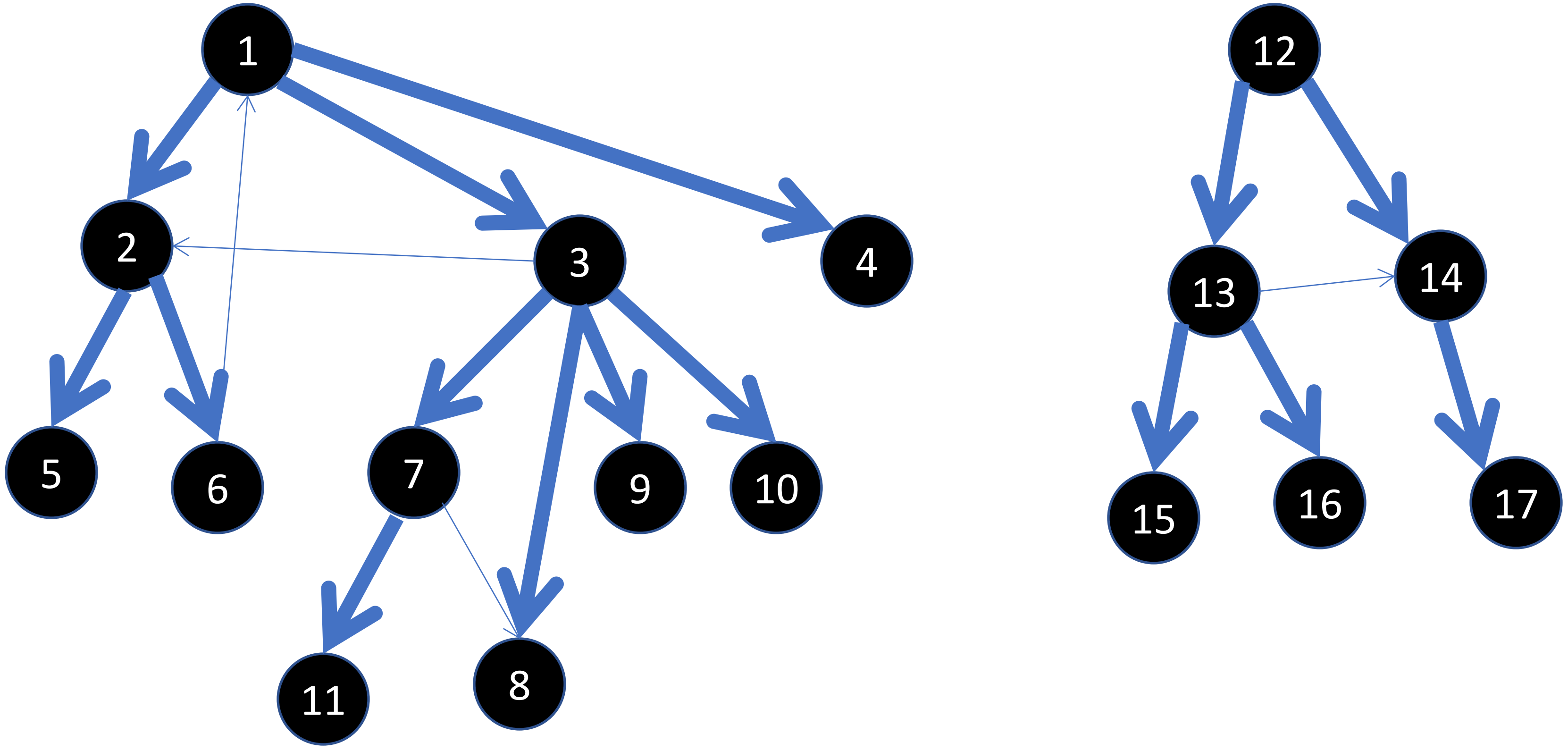
A BFS example (2)



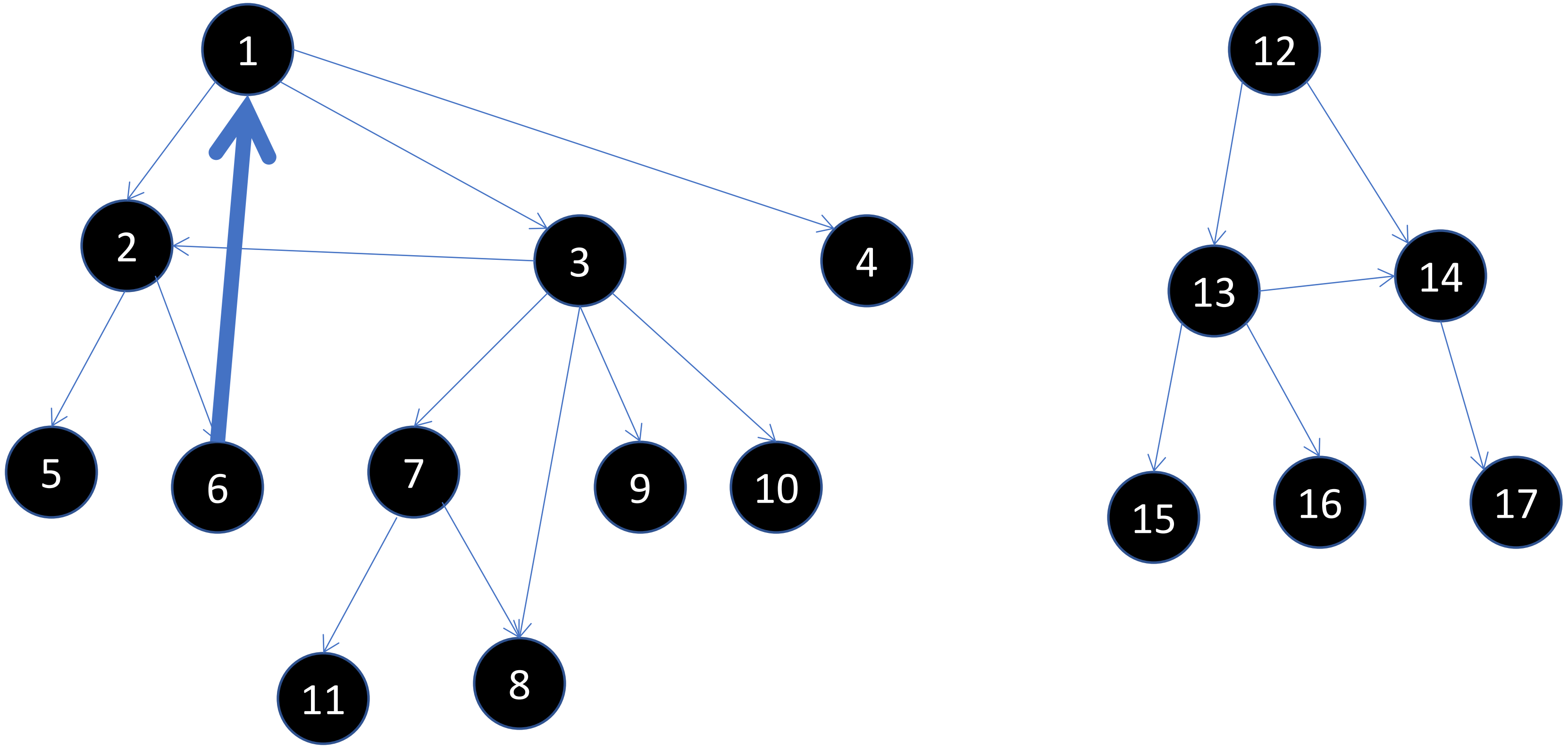
A BFS example (2)



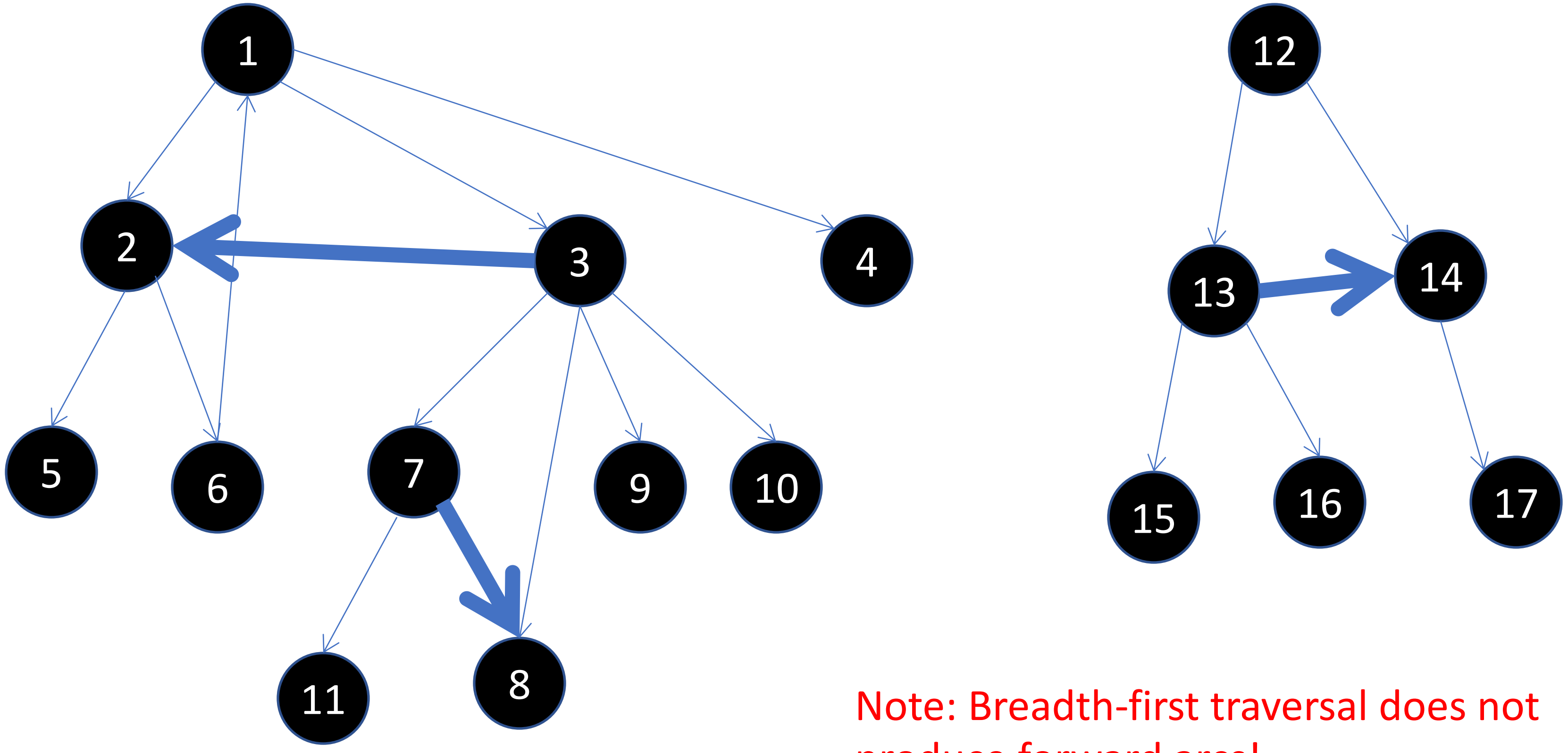
A BFS example (2): Tree Arcs



A BFS example (2): Back Arc



A BFS example (2): Cross Arcs

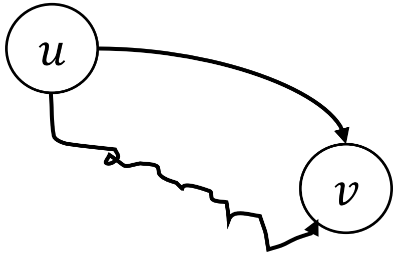


Note: Breadth-first traversal does not produce forward arcs!

Basic properties of Breadth-first Search

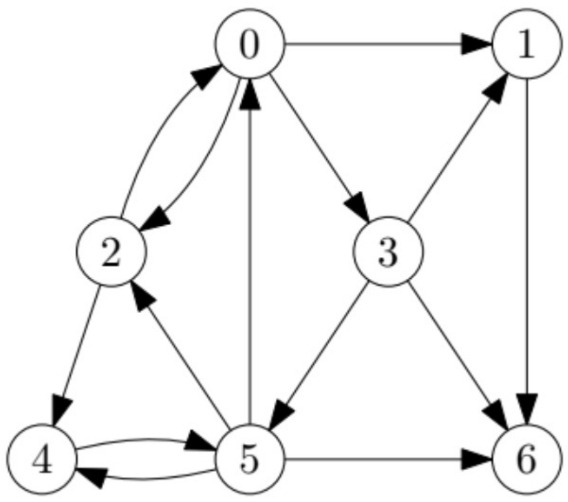
- BFS Property: There are no forward arcs when performing BFS on a digraph. Why?

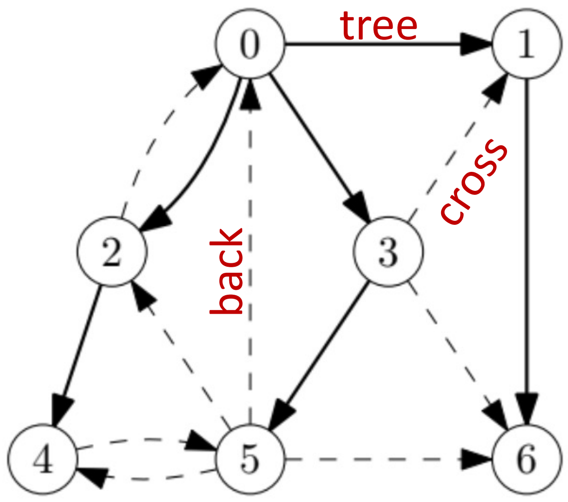
- **Proof**:
- Suppose there exist a forward arc. Then we have the following in the search forest.
- There is a directed path of tree arcs(at least with two arcs) from u to v, and there is an arc (u,v) that is not in any tree of the search forest. But then BFS explores v as a neighbour of u when v is white. So (u,v) is a tree arc, a contradiction.

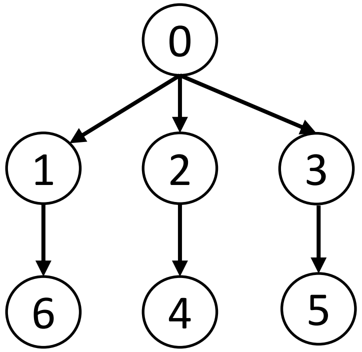


Example 24.2

- A digraph and its BFS search tree, rooted at node 0. The dashed arcs indicate the original arcs that are not part of the BFS search tree.



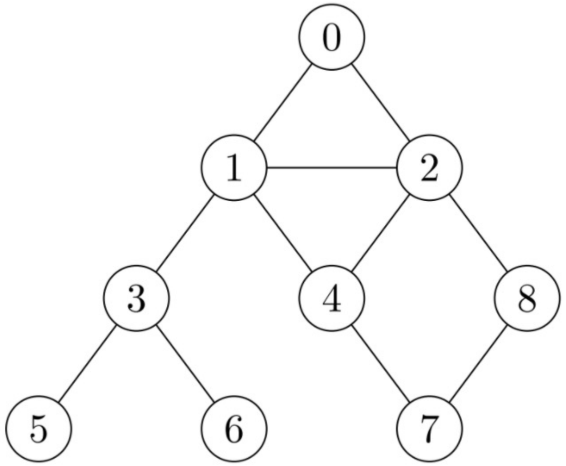


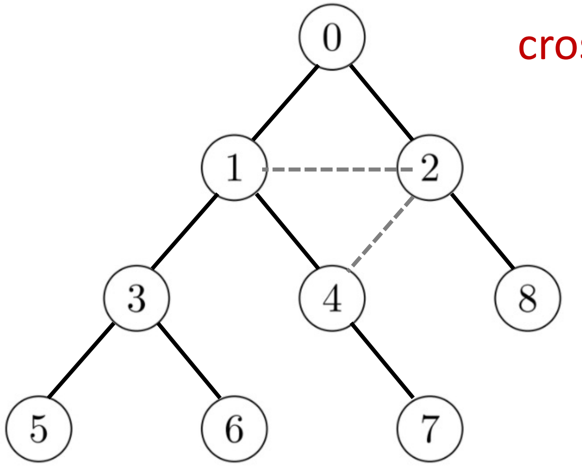


- Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

Example 24.3

- Use the nodes on the right to draw the search tree you obtain by running BFS on the graph on the left, starting at vertex 0. Use dashed edges to indicate edges that are not arcs in the search tree.





cross arcs

- Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

Forward arcs = backward arcs (no directions), but there are no forward arcs
→ all dotted arcs are cross arcs.

Time Complexity: DFS v.s. BFS

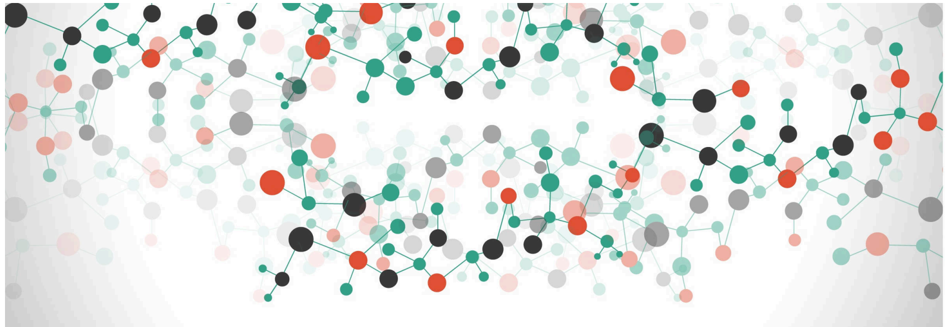
- Both BFS and DFS have same complexity
  - the next grey/white node has constant complexity

- With adjacency matrix: Need to find out-neighbours for n vertices. With $\Theta(n)$ per vertex, this means $\Theta(n^2)$.

- With adjacency list: the sequence for each node is only visited once and we need to visit all the edges. So the complexity is $\Theta(n+m)$.

OUTLINE

- Graph Traversal Algorithms
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Priority-first Search (PFS)
- Implementation
  - Stack – DFS
  - Queue – BFS
  - Priority Queues – PFS



Priority-first Search (PFS)

- In PFS, the next grey vertex is selected according to a priority value
- Typically, this is an integer, such that the grey vertex with the lowest value is selected first
- Priority value is assigned (often computed) no later than when the vertex turns grey

Priority-first-search (PFS) Algorithm

Algorithm 6 Priority-first search algorithm

```
function PFS(digraph G)
    priority queue Q
    array colour[0..n − 1], pred[0..n − 1]
    for u ∈ V(G) do
        colour[u] ← WHITE
        pred[u] ← null
    for s ∈ V(G) do
        if colour[s] = WHITE then
            PFSVISIT(s)
    return pred
```

Iterative View of PFSBVISIT

Algorithm 7 Priority-first visit algorithm.

function PFSVISIT(node s)

  $color[s] \leftarrow$ GREY

  $Q.insert(s, setKey(s))$

  while not $Q.isEmpty()$ **do**

    $u \leftarrow Q.peek()$

    if u has a neighbour v with $colour[v] =$ WHITE **then**

      $colour[v] \leftarrow$ GREY

      $Q.insert(v, setKey(v))$

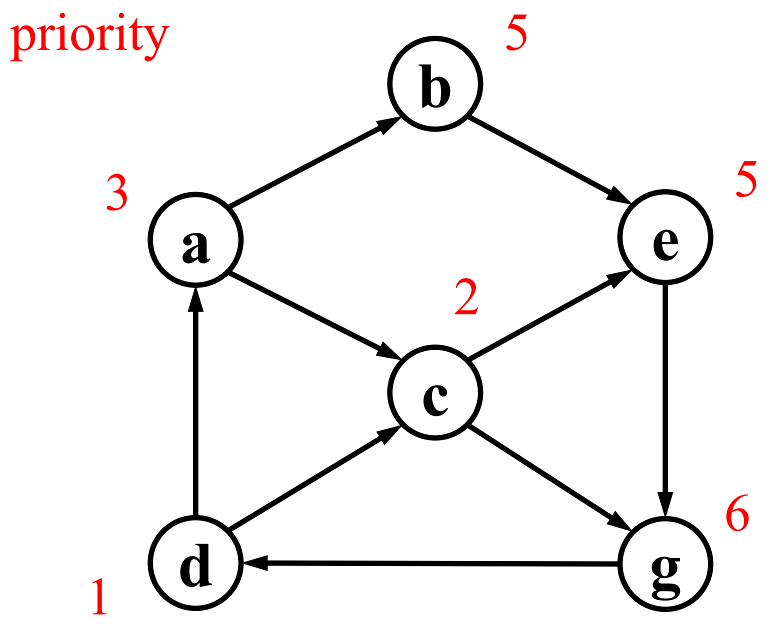
    else

      $Q.delete()$

      $colour[u] \leftarrow$ BLACK

A PFS example

- Start at a



(a,3)

(a,3) (b,5)

(a,3) (b,5) (c,2)

(a,3) (b,5) (c,2) (e,5)

(a,3) (b,5) (c,2) (e,5) (g,6)

(a,3) (b,5) (e,5) (g,6)

(b,5) (e,5) (g,6)

(e,5) (g,6)

(g,6)

(g,6) (d,1)

(g,6)

Priority-first Search (PFS)

- In simple PFS, the priority value of a vertex does not change.
- In advanced PFS, the priority value of a vertex may be updated again later.

- BFS and DFS are special cases of simple PFS.
  - In BFS, the priority values are the order in which the vertices turn grey (1, 2, 3, …).
  - In DFS, the priority values are the negative order in which the vertices turn grey (-1, -2, -3, …).

Time Complexity: PFS

- General time complexity is not very good: Need to find minimum priority value each time we need to select the next grey node
- If we search up to n vertices (say, in an array) for the minimum priority value, we need $\Omega(n^2)$
- This can be improved on a little by using a priority queue (e.g., a heap) for the grey vertices, but it's still slower than pure DFS or BFS
- Use PFS when there is an advantage in processing high priority vertices first (e.g., when this allows us to remove other vertices or edges from the (di)graph to be traversed, thereby reducing the search space)

SUMMARY

- Graph Traversal Algorithms
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Priority-first Search (PFS)

- Implementation
  - Stack – DFS
  - Queue – BFS
  - Priority Queues – PFS