COMP90038
Algorithms and Complexity

Lecture 6: Graph Traversal
(with thanks to Harald Søndergaard)

Toby Murray

toby.murray@unimelb.edu.au

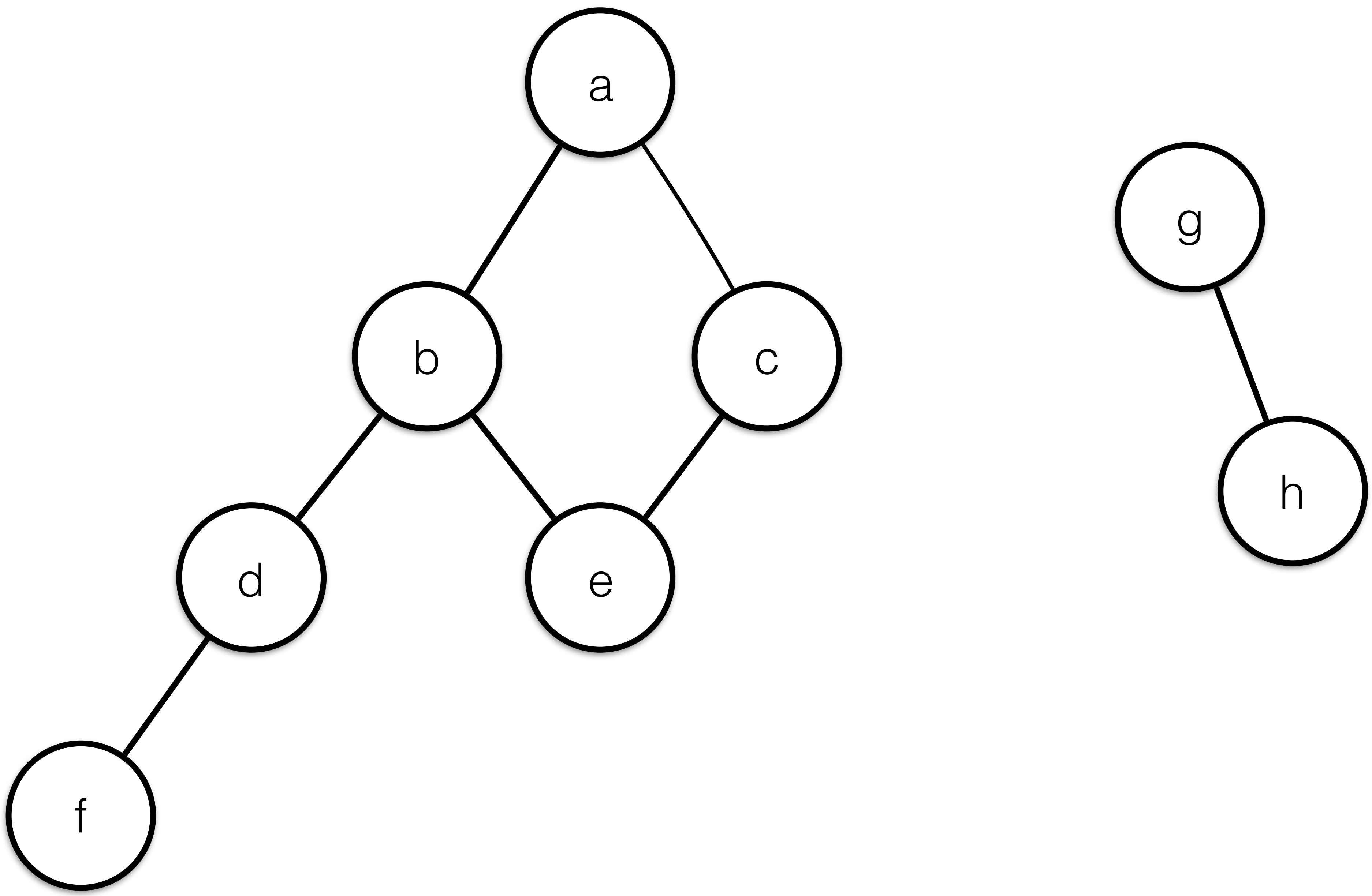
DMD 8.17 (Level 8, Doug McDonell Bldg)

http://people.eng.unimelb.edu.au/tobym

@tobycmurray

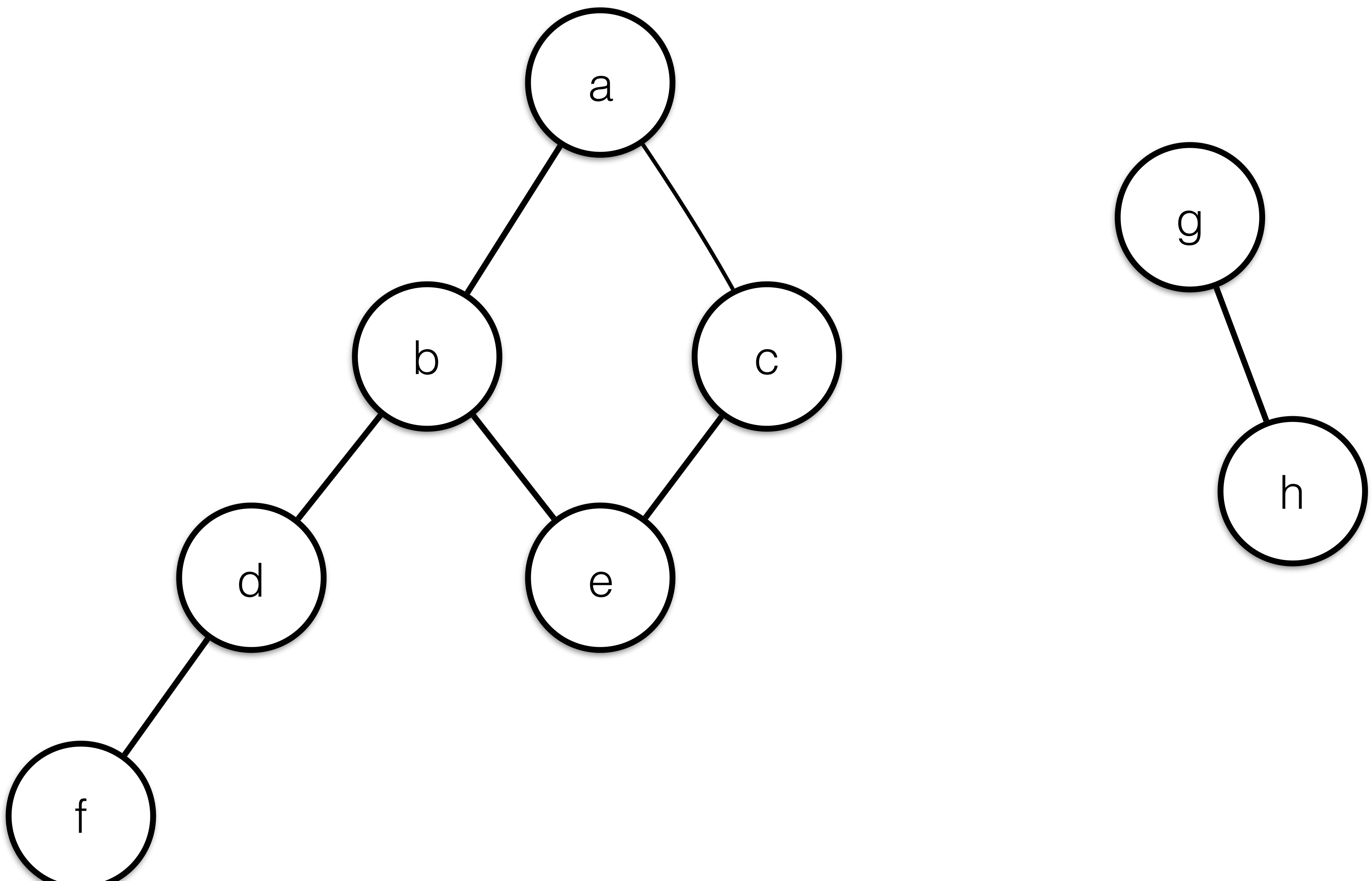
Breadth-First and Depth-First Traversal

- Used to **systematically** explore **all** nodes of a graph



Depth-First Traversal

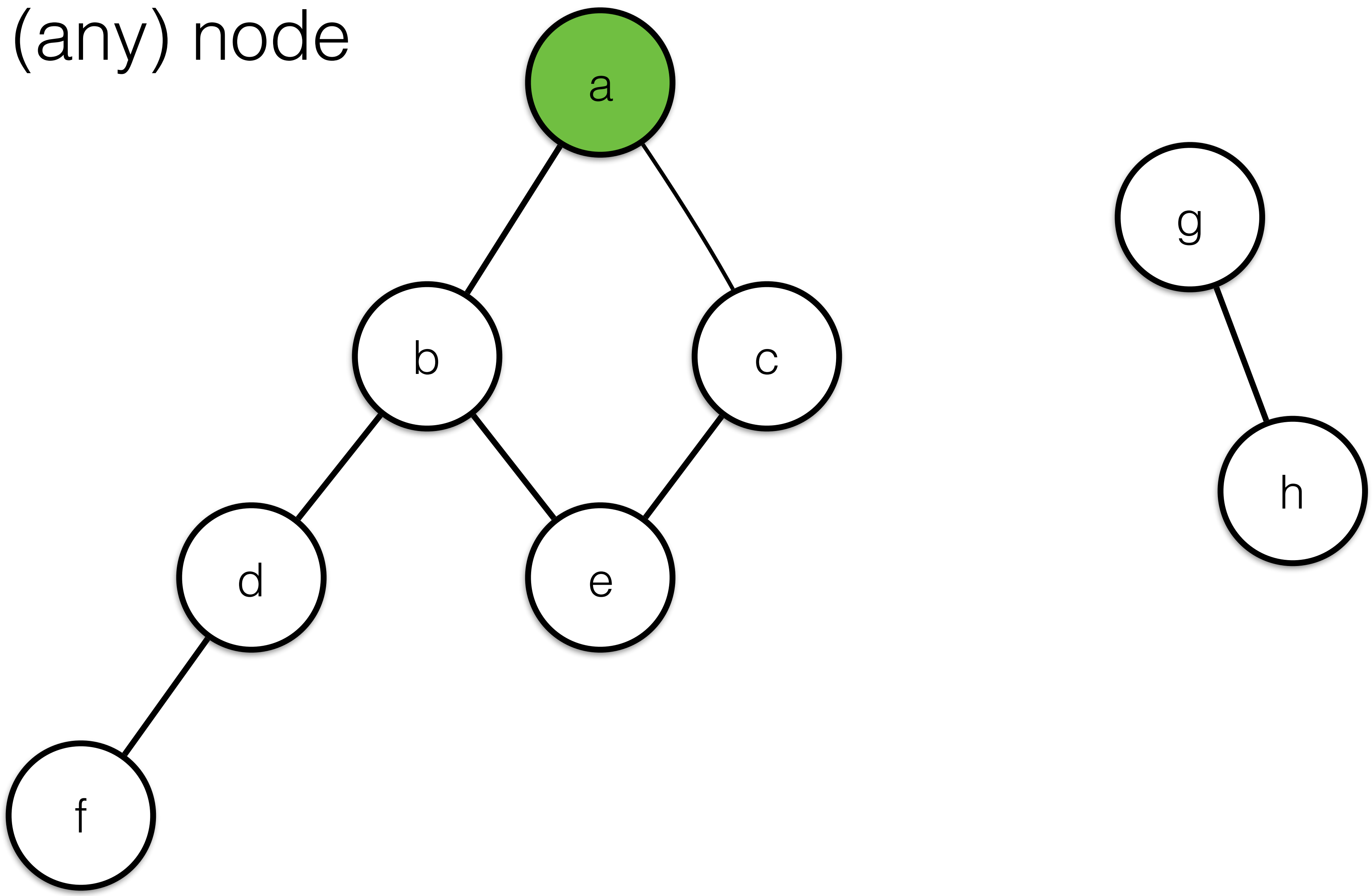
THE UNIVERSITY OF MELBOURNE



Depth-First Traversal

Nodes visited in this order: a

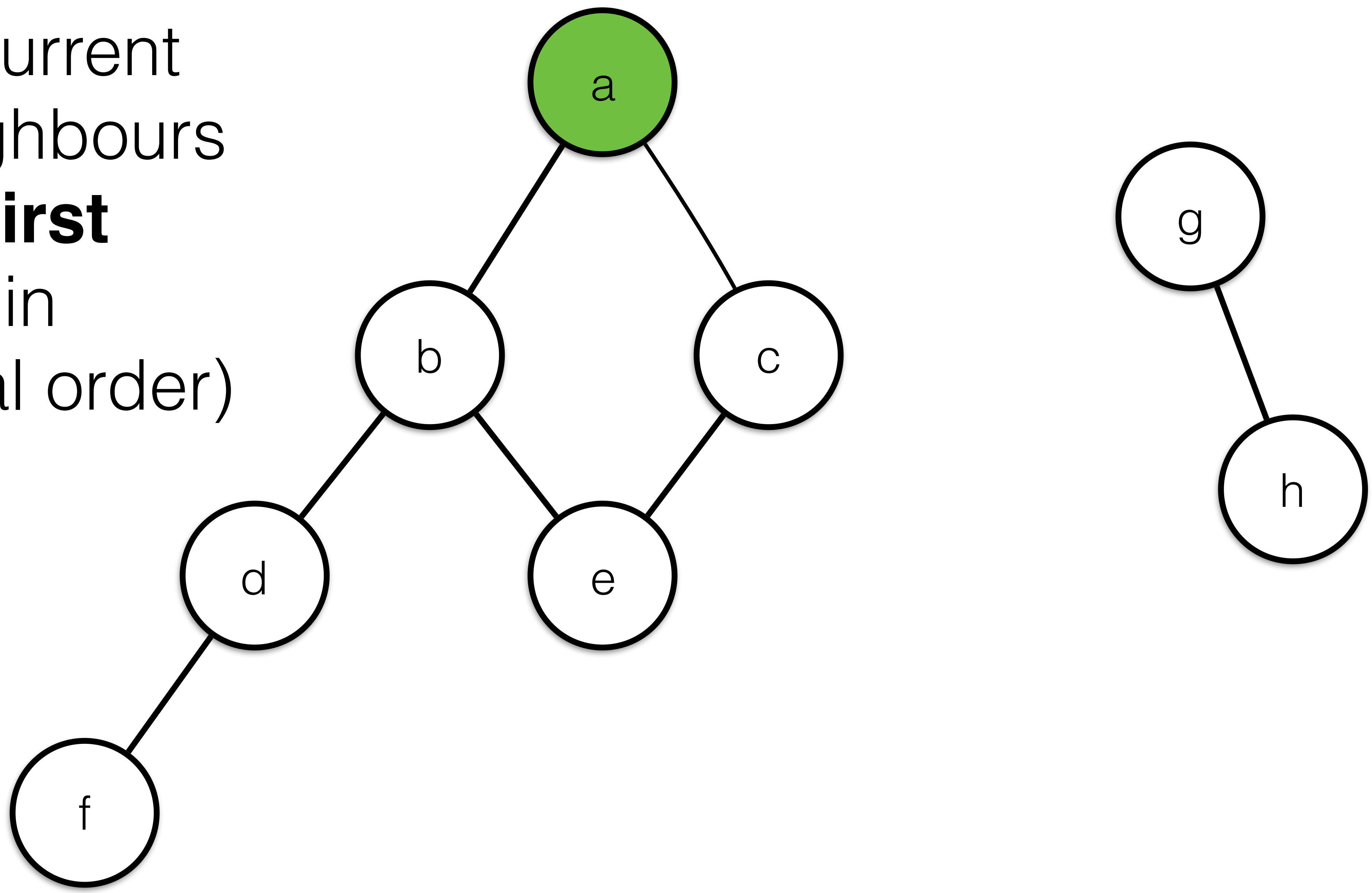
Start with (any) node



Depth-First Traversal

Nodes visited in this order: a

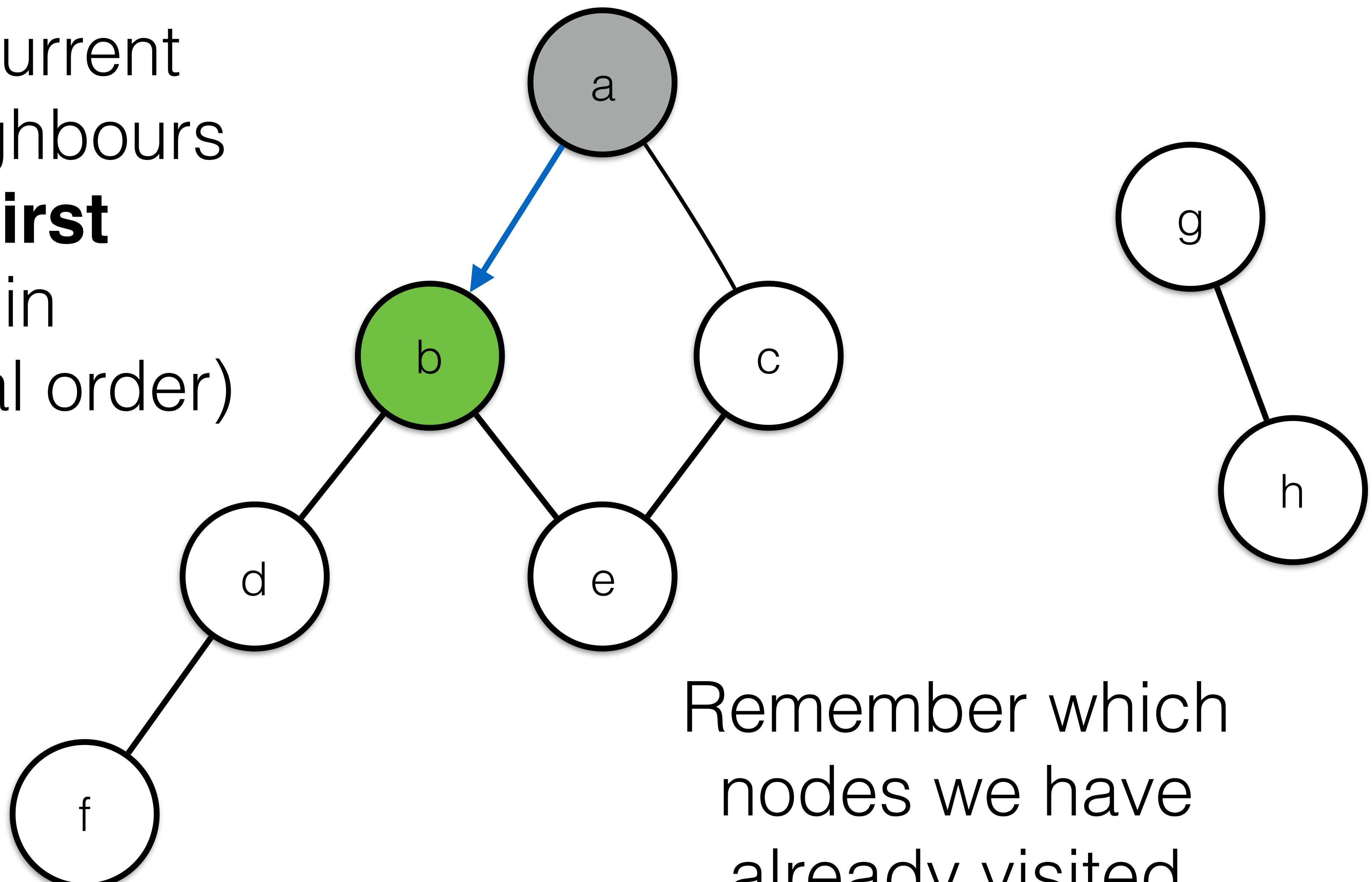
Visit the current node's neighbours **depth first** (here in alphabetical order)



Depth-First Traversal

Nodes visited in this order: a b

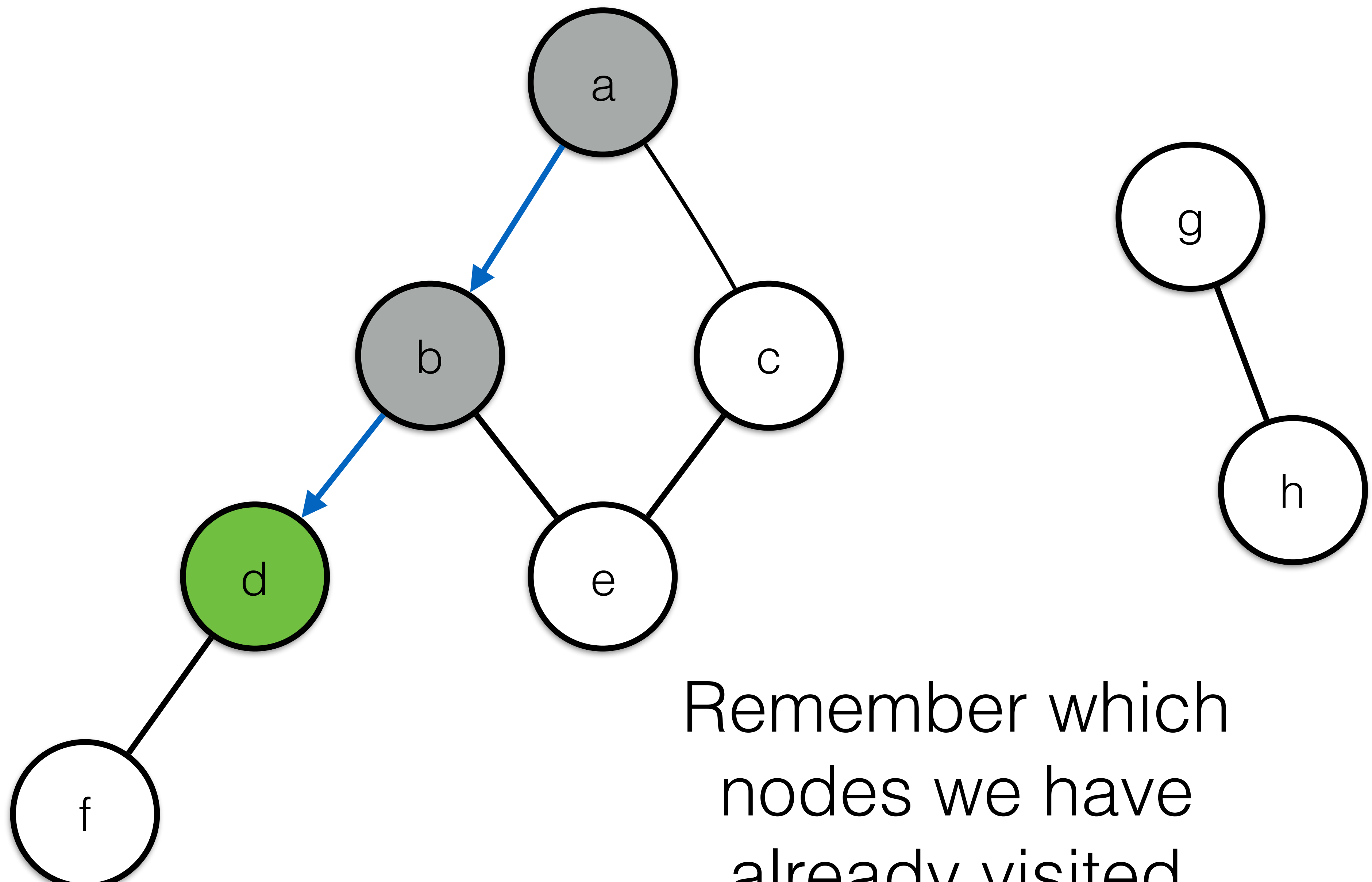
Visit the current node's neighbours **depth first** (here in alphabetical order)



Remember which nodes we have already visited to avoid revisiting them

Depth-First Traversal

Nodes visited in this order: a b d

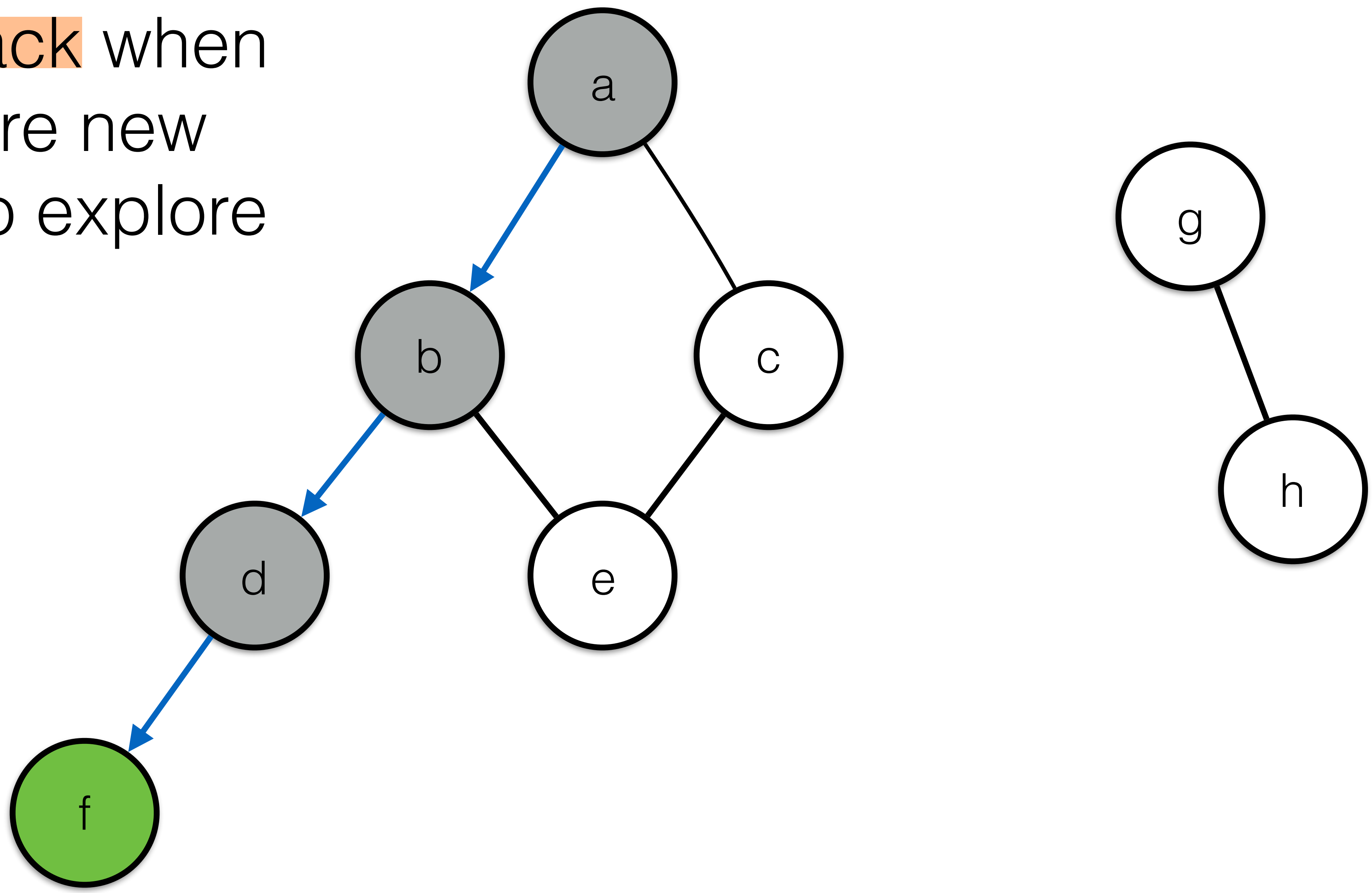


Remember which nodes we have already visited to avoid revisiting them

Depth-First Traversal

Nodes visited in this order: a b d f

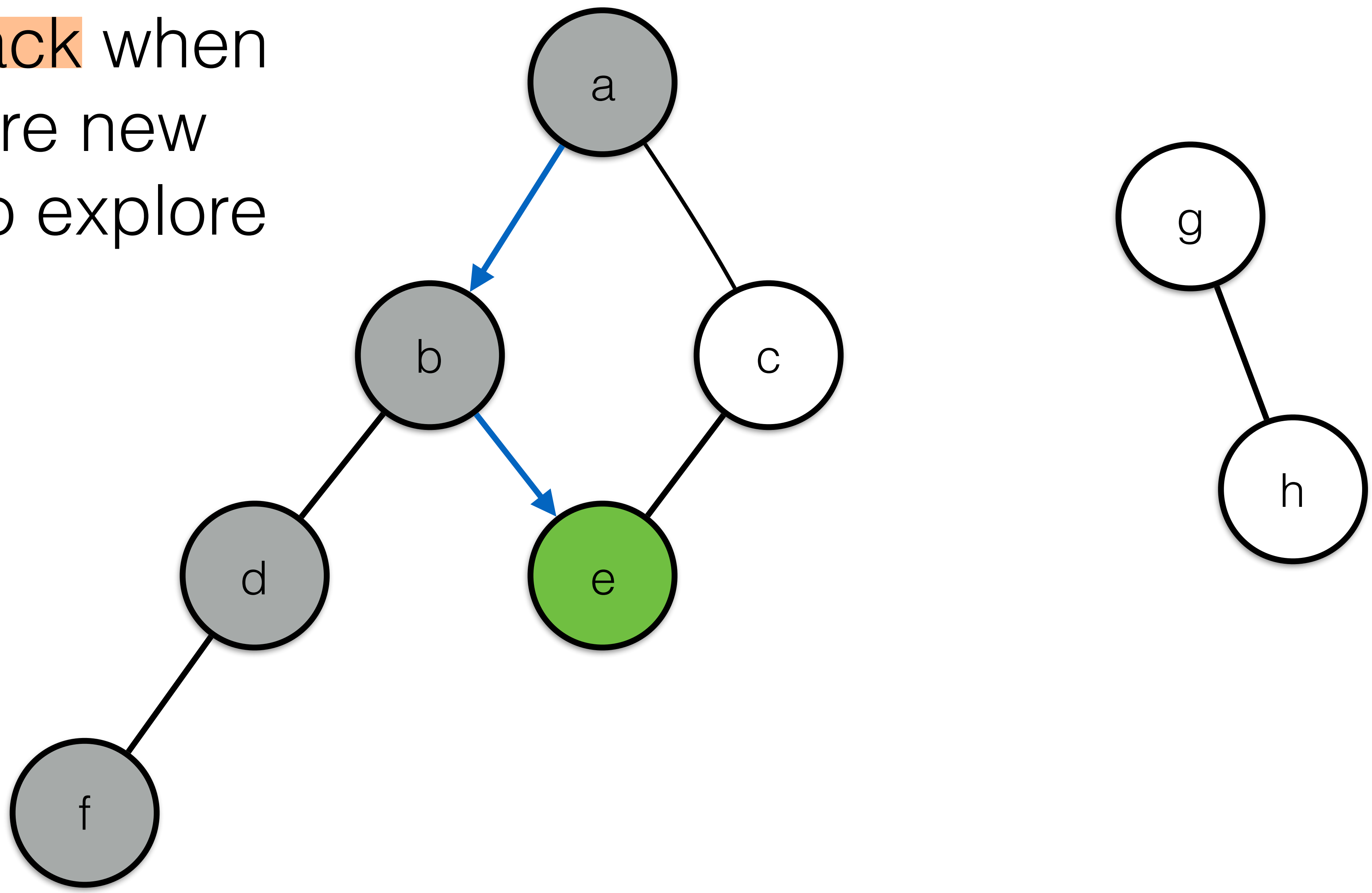
Back-track when no more new nodes to explore



Depth-First Traversal

Nodes visited in this order: a b d f e

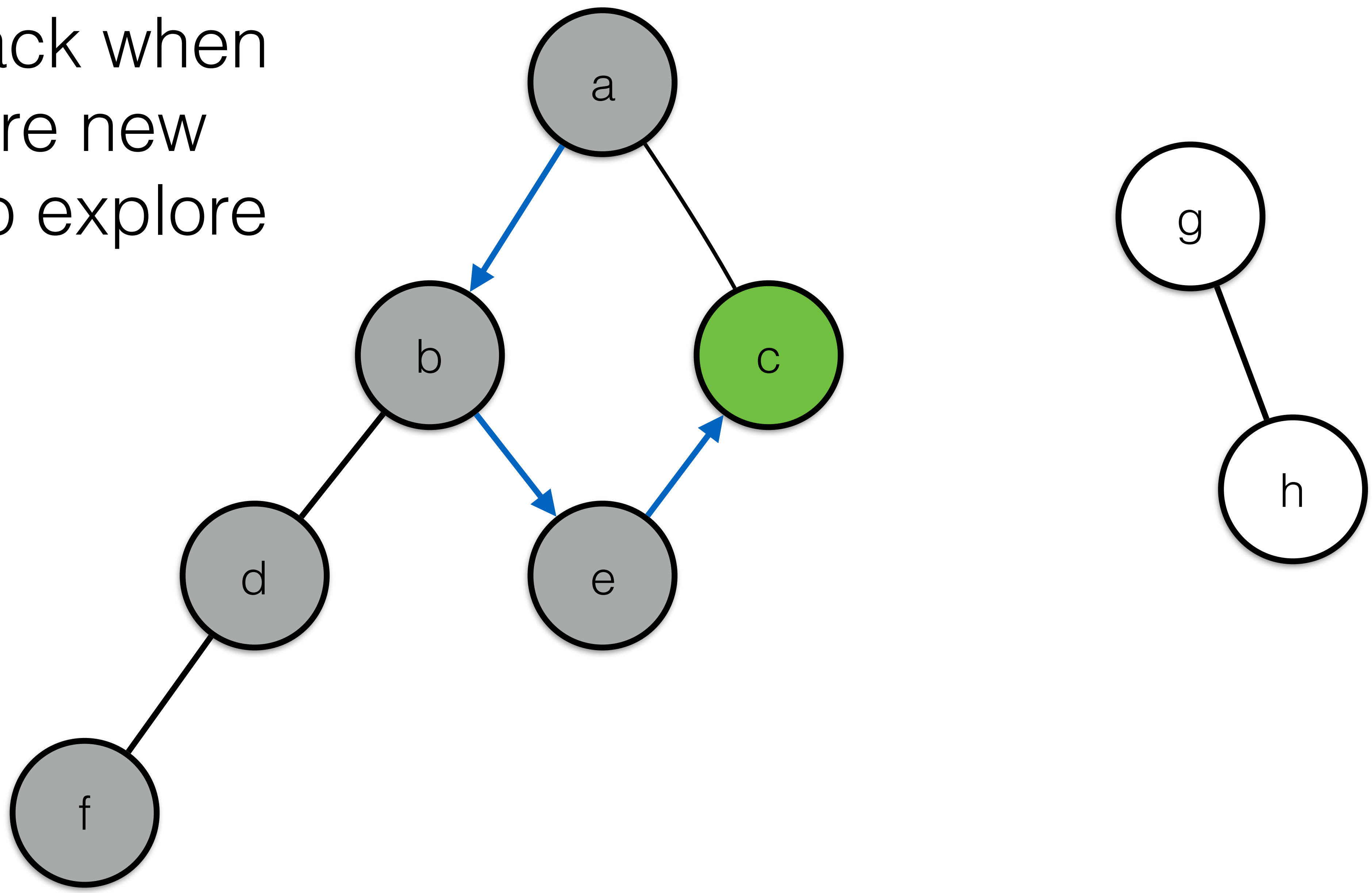
Back-track when no more new nodes to explore



Depth-First Traversal

Nodes visited in this order: a b d f e c

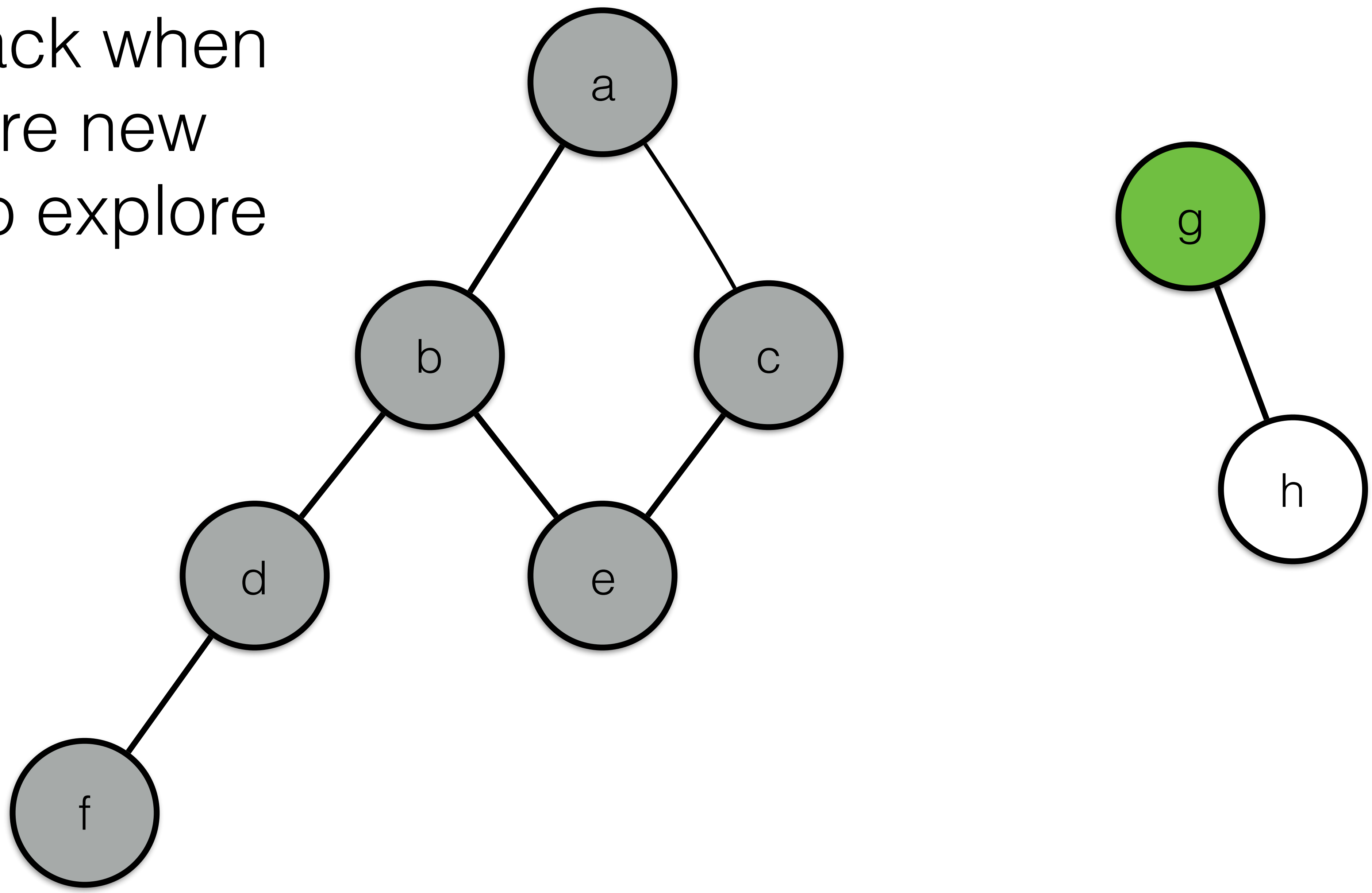
Back-track when
no more new
nodes to explore



Depth-First Traversal

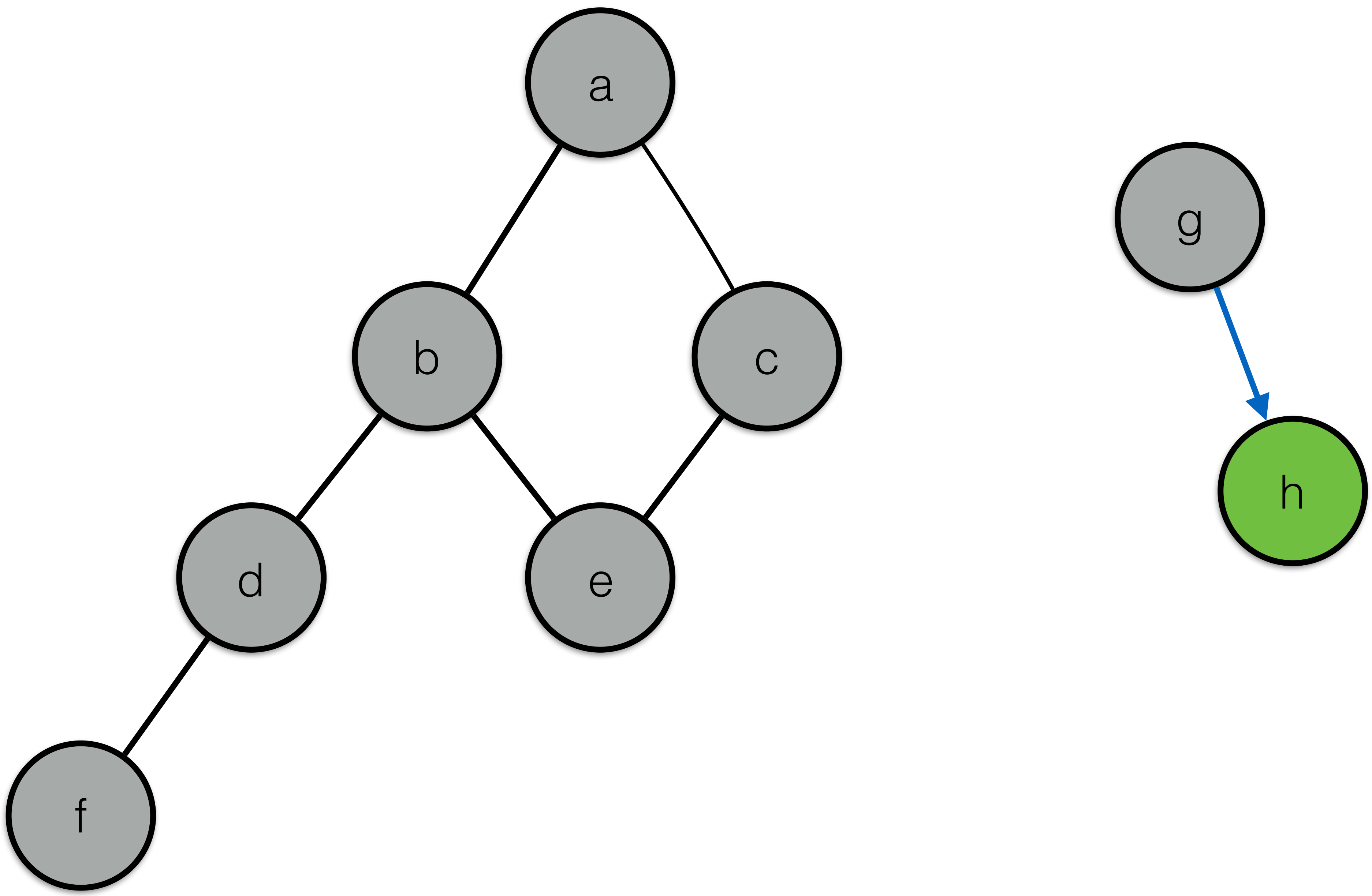
Nodes visited in this order: a b d f e c g

Back-track when
no more new
nodes to explore



Depth-First Traversal

Nodes visited in this order: a b d f e c g h



Depth-First Traversal: Stack Discipline

THE UNIVERSITY OF MELBOURNE

When back-tracking, we go back to the most recently-visited node that still has unvisited neighbours

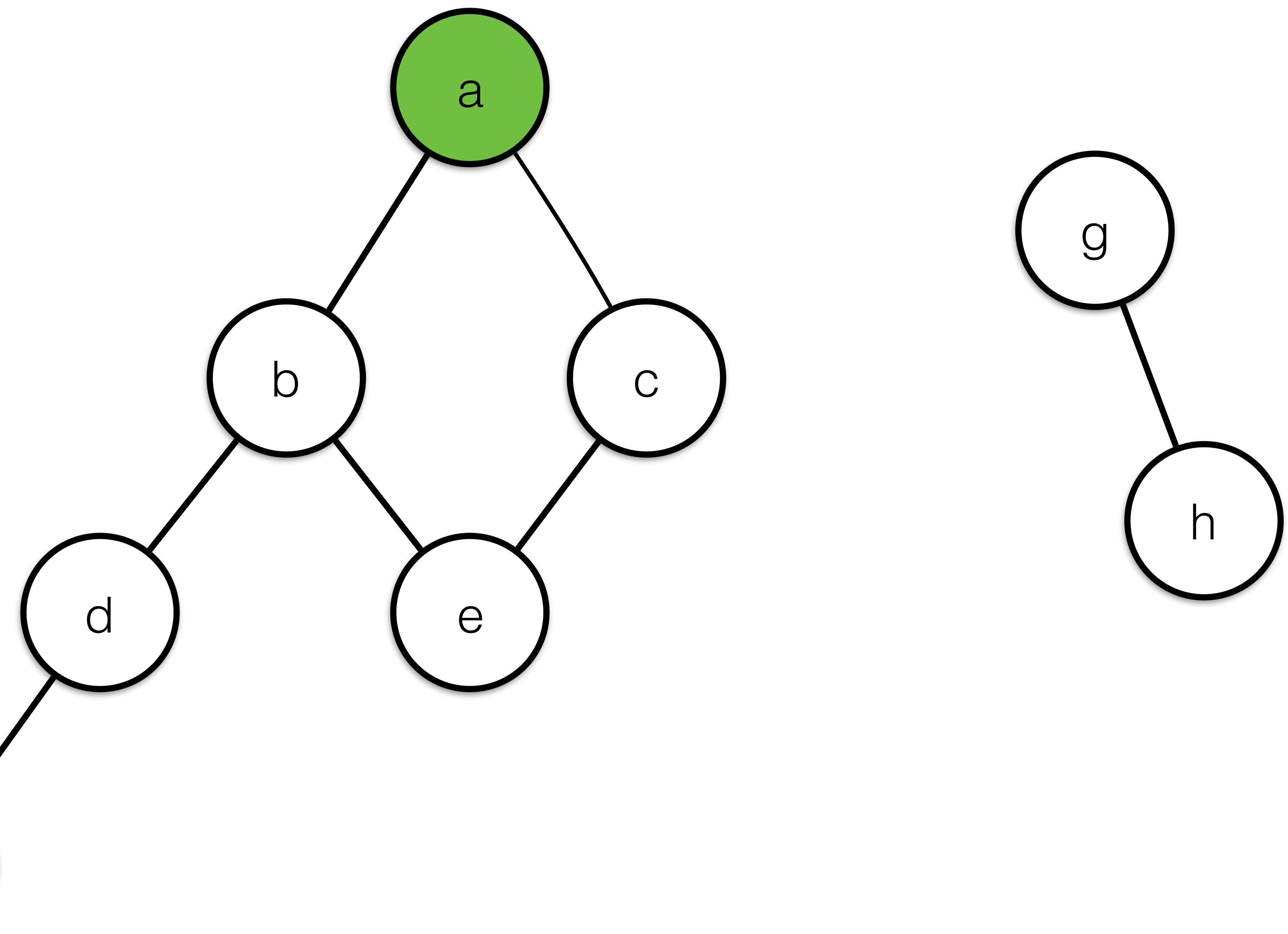
This is simulated by pushing each node onto a stack as it is visited.

Back-tracking then corresponds to popping the stack.

Depth-First Traversal: Stack Discipline

(Simulated) stack:

a

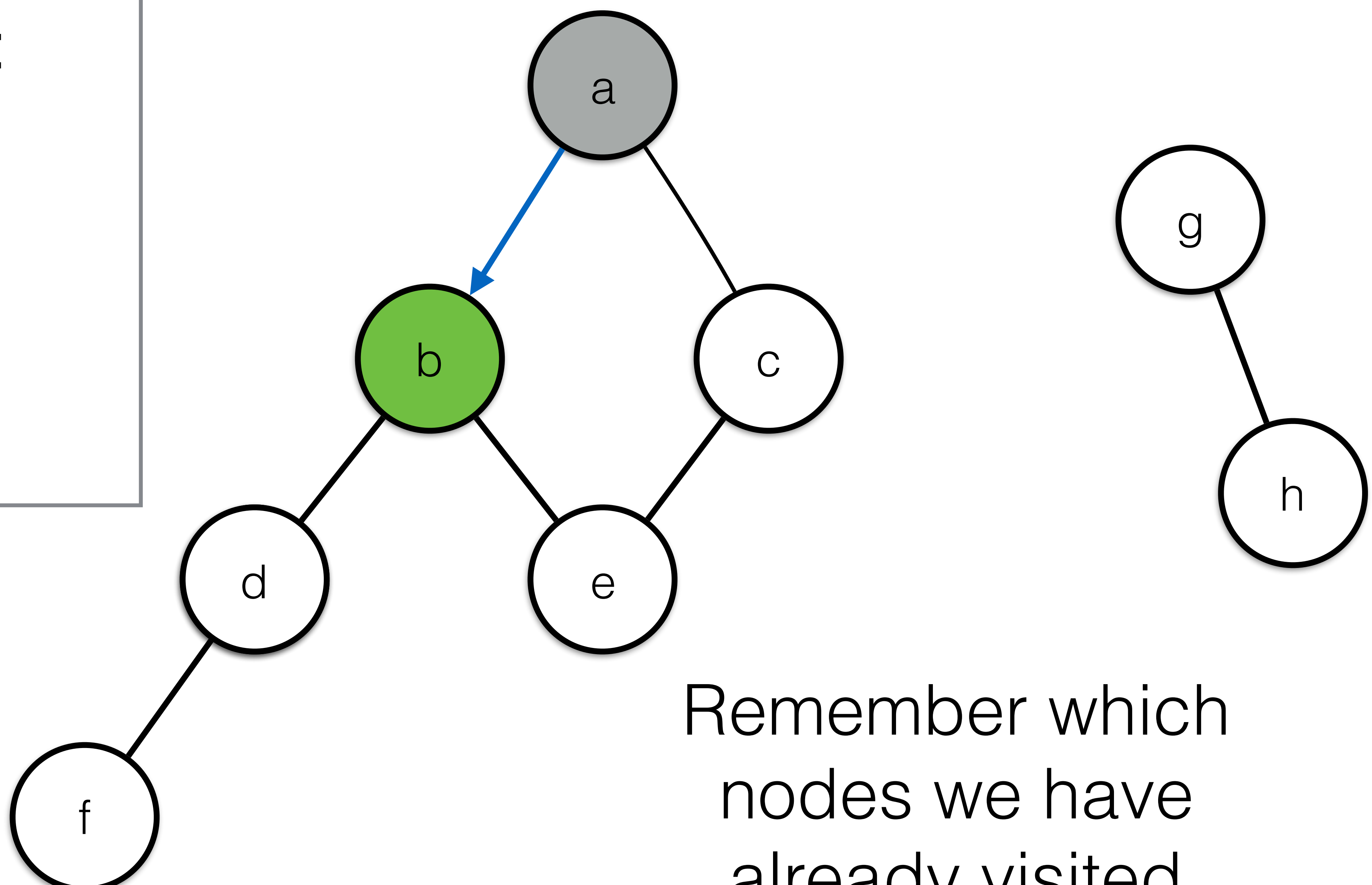


Depth-First Traversal: Stack Discipline

(Simulated) stack:

b
a

a b

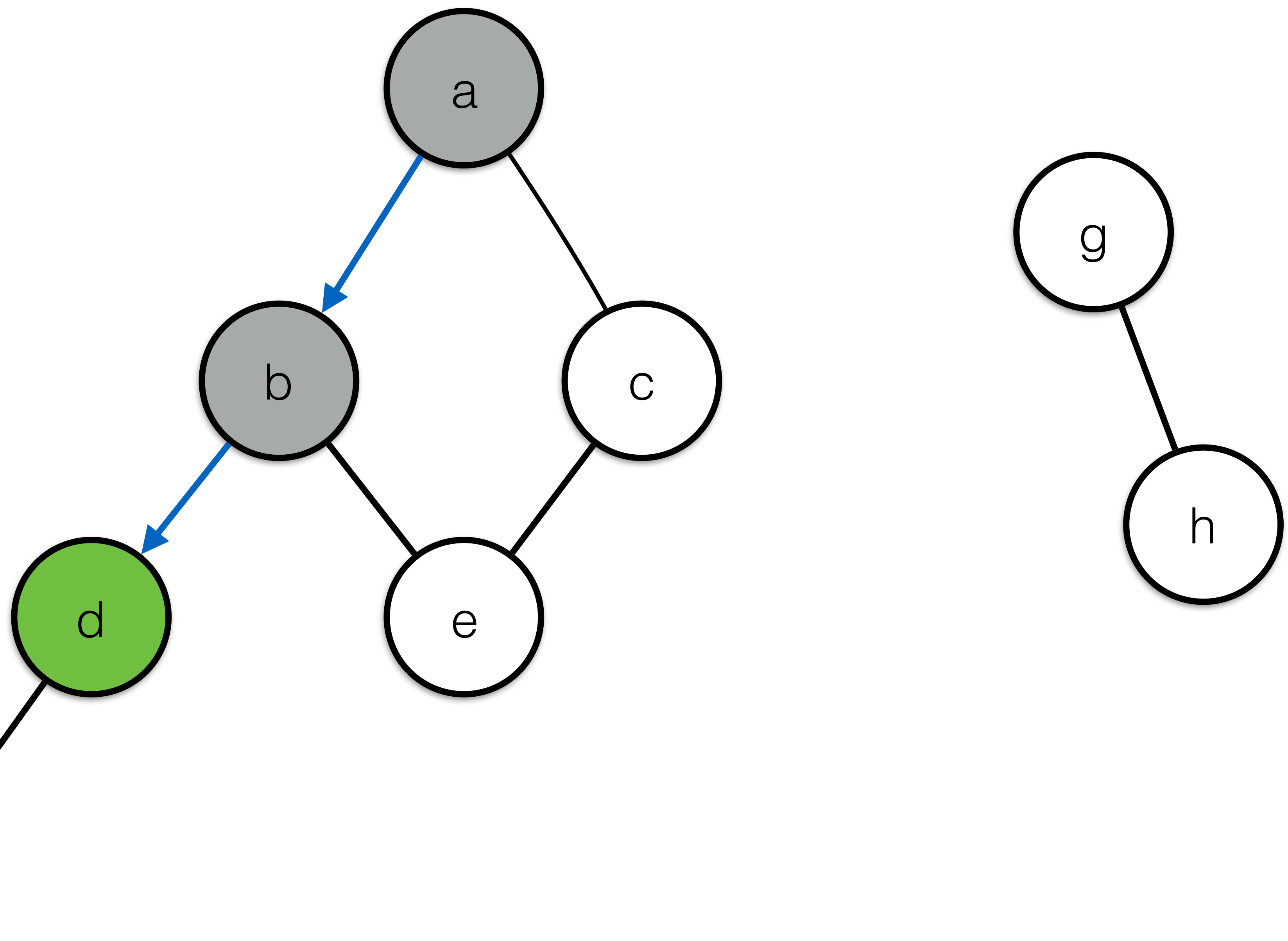


Remember which nodes we have already visited to avoid revisiting them

Depth-First Traversal: Stack Discipline

(Simulated) stack:

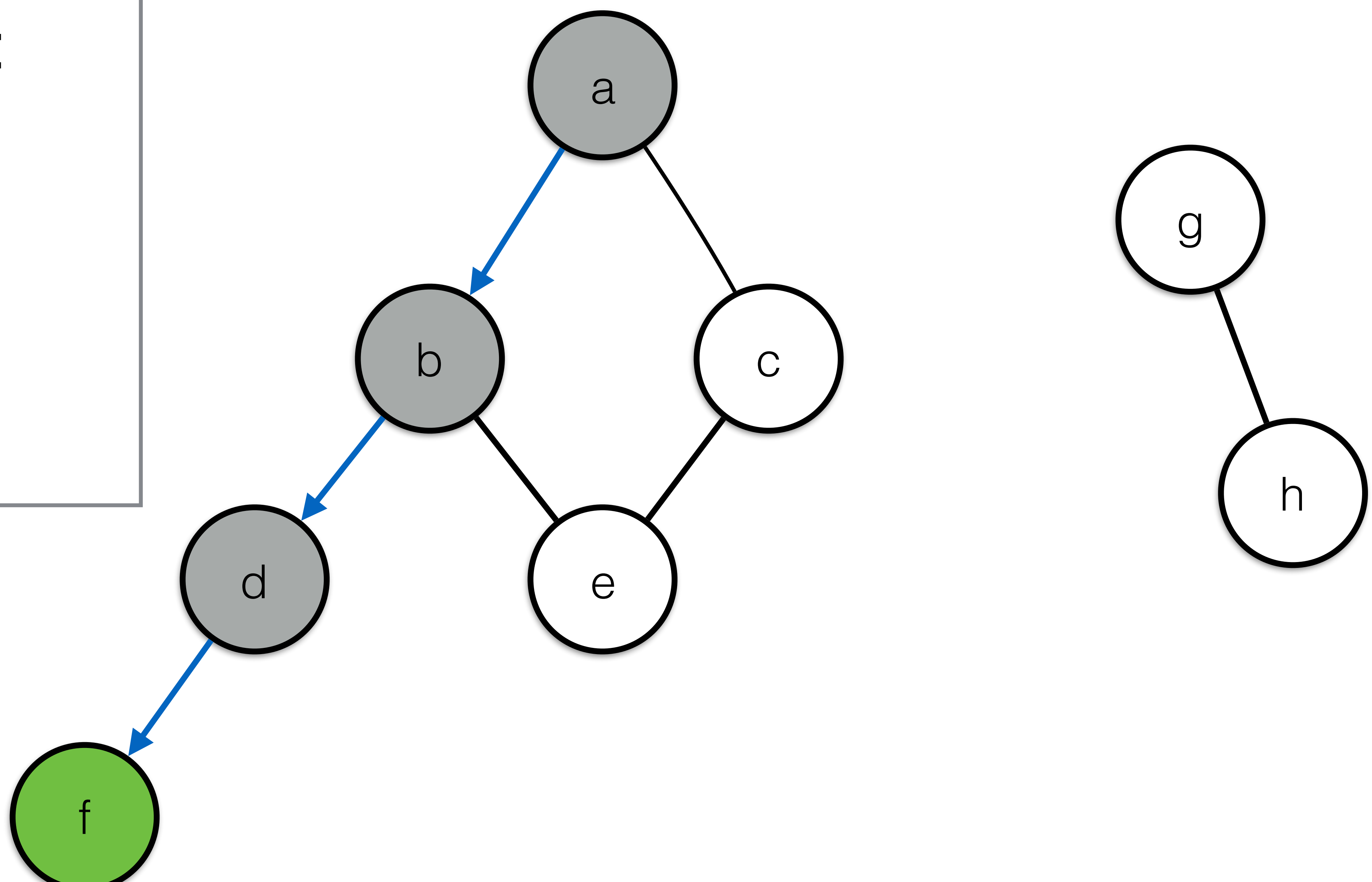
d
b
a



Depth-First Traversal: Stack Discipline

(Simulated) stack:

f
d
b
a

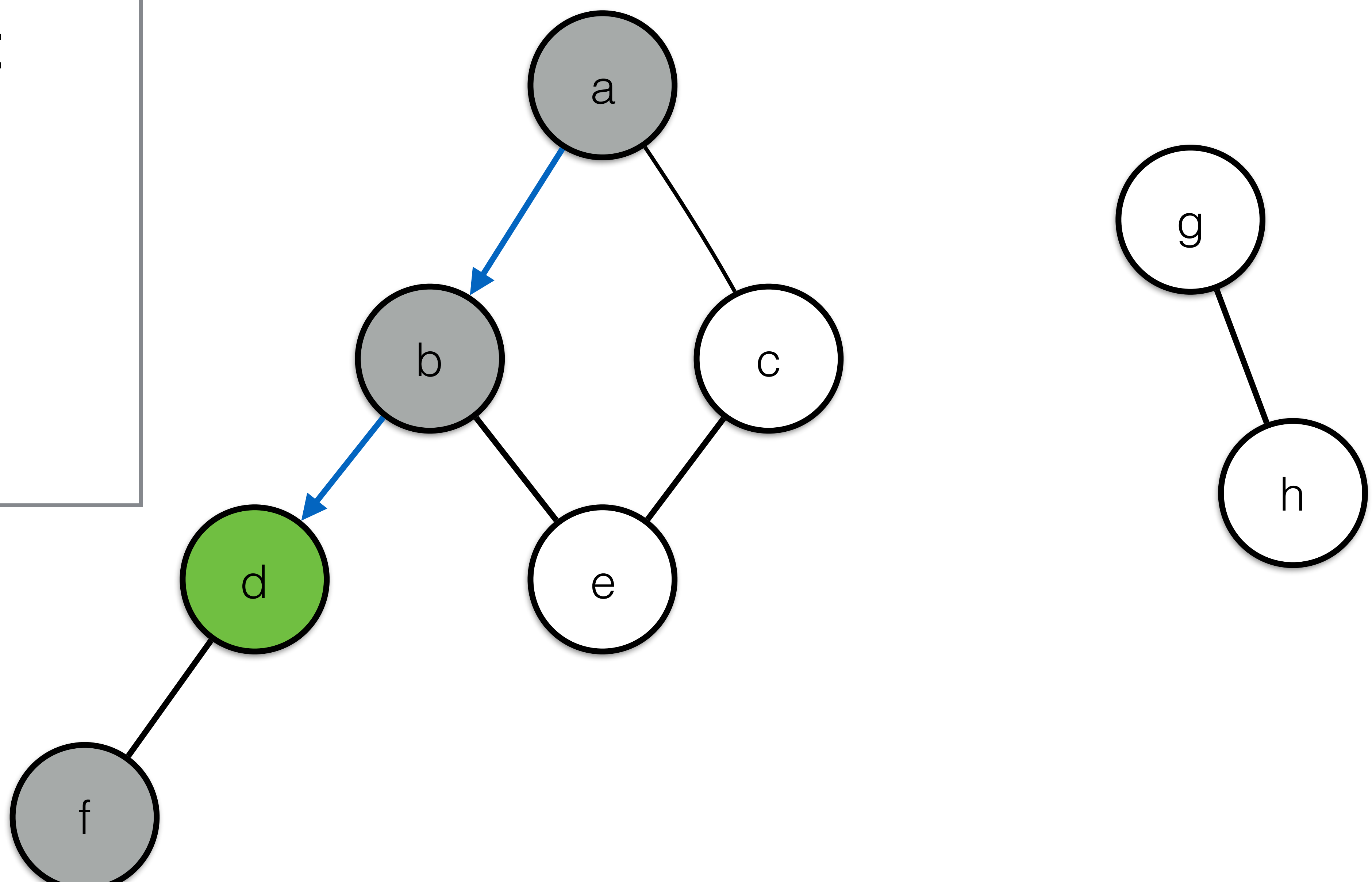


Depth-First Traversal: Stack Discipline

(back-tracking)

(Simulated) stack:

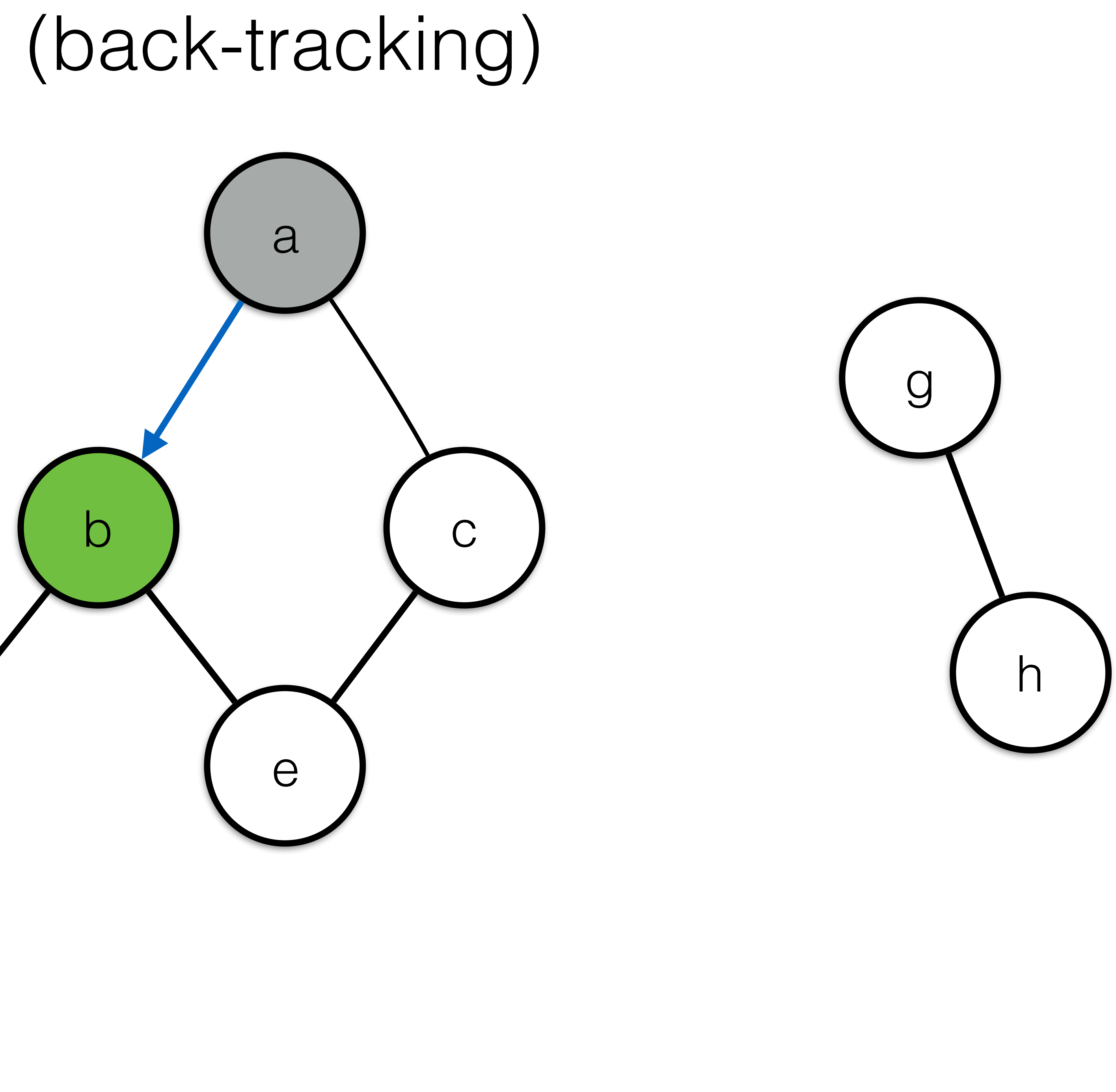
d
b
a



Depth-First Traversal: Stack Discipline

(Simulated) stack:

b
a



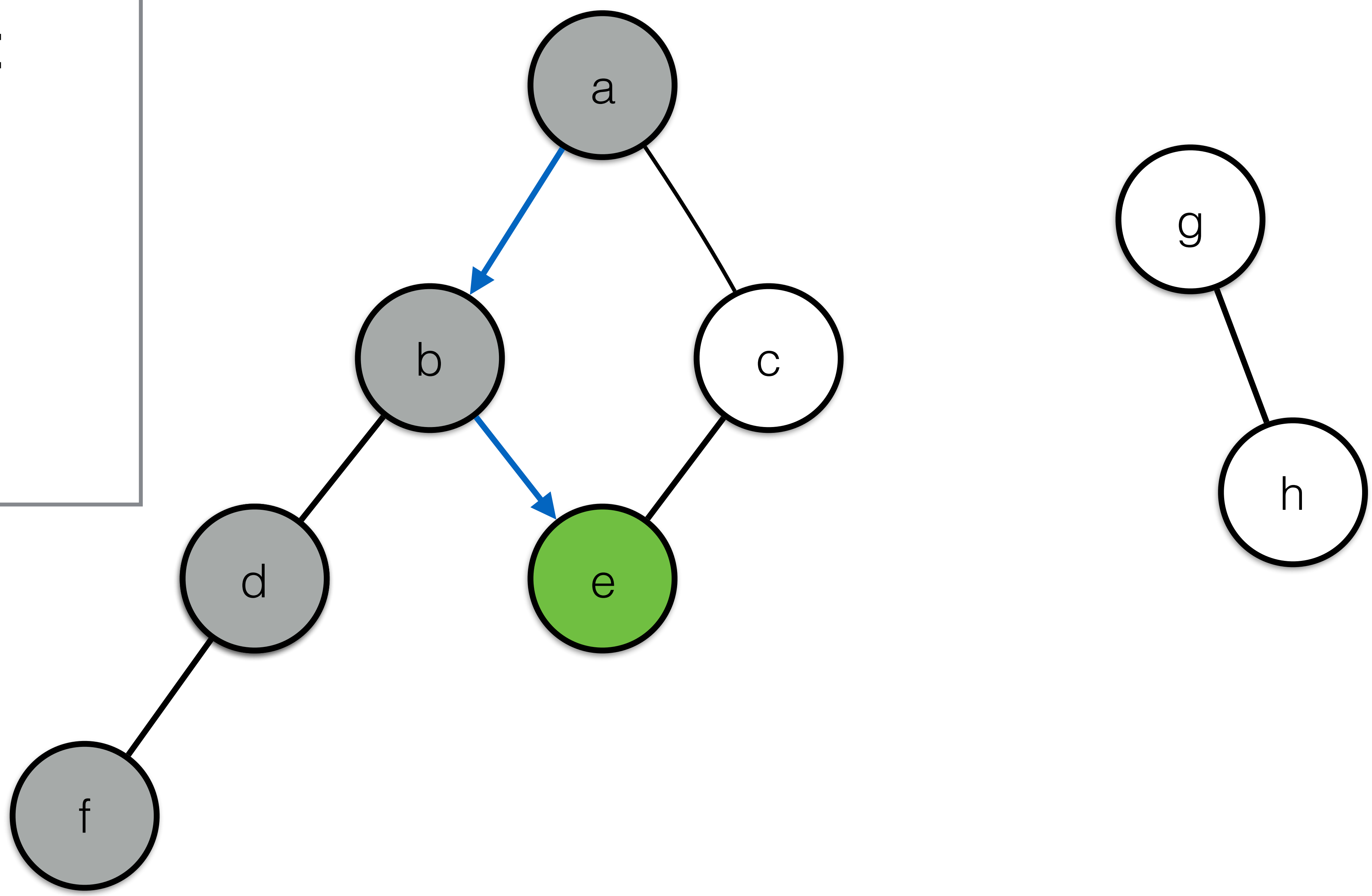
Depth-First Traversal: Stack Discipline

(Simulated) stack:

e

b

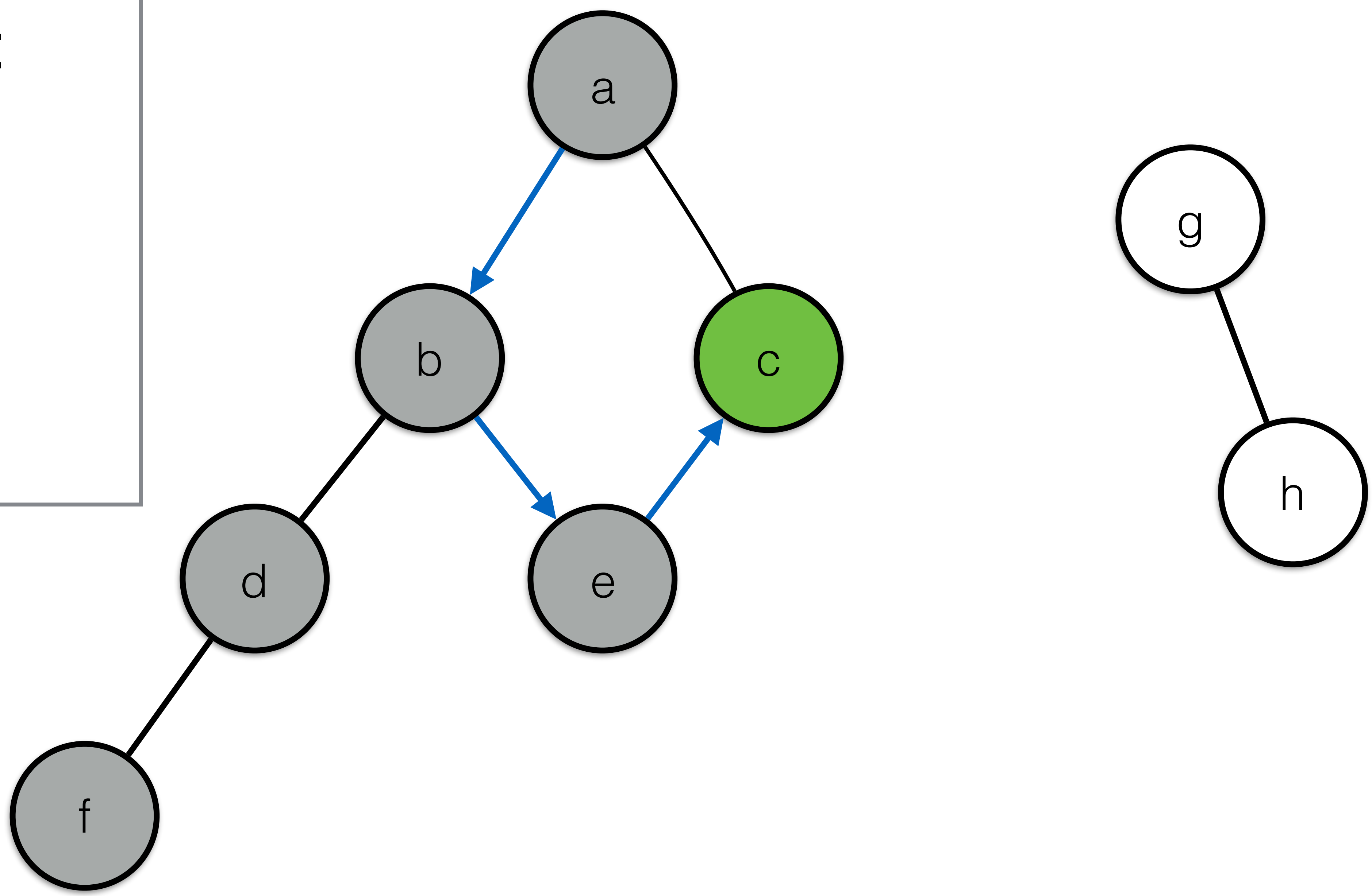
a



Depth-First Traversal: Stack Discipline

(Simulated) stack:

c
e
b
a

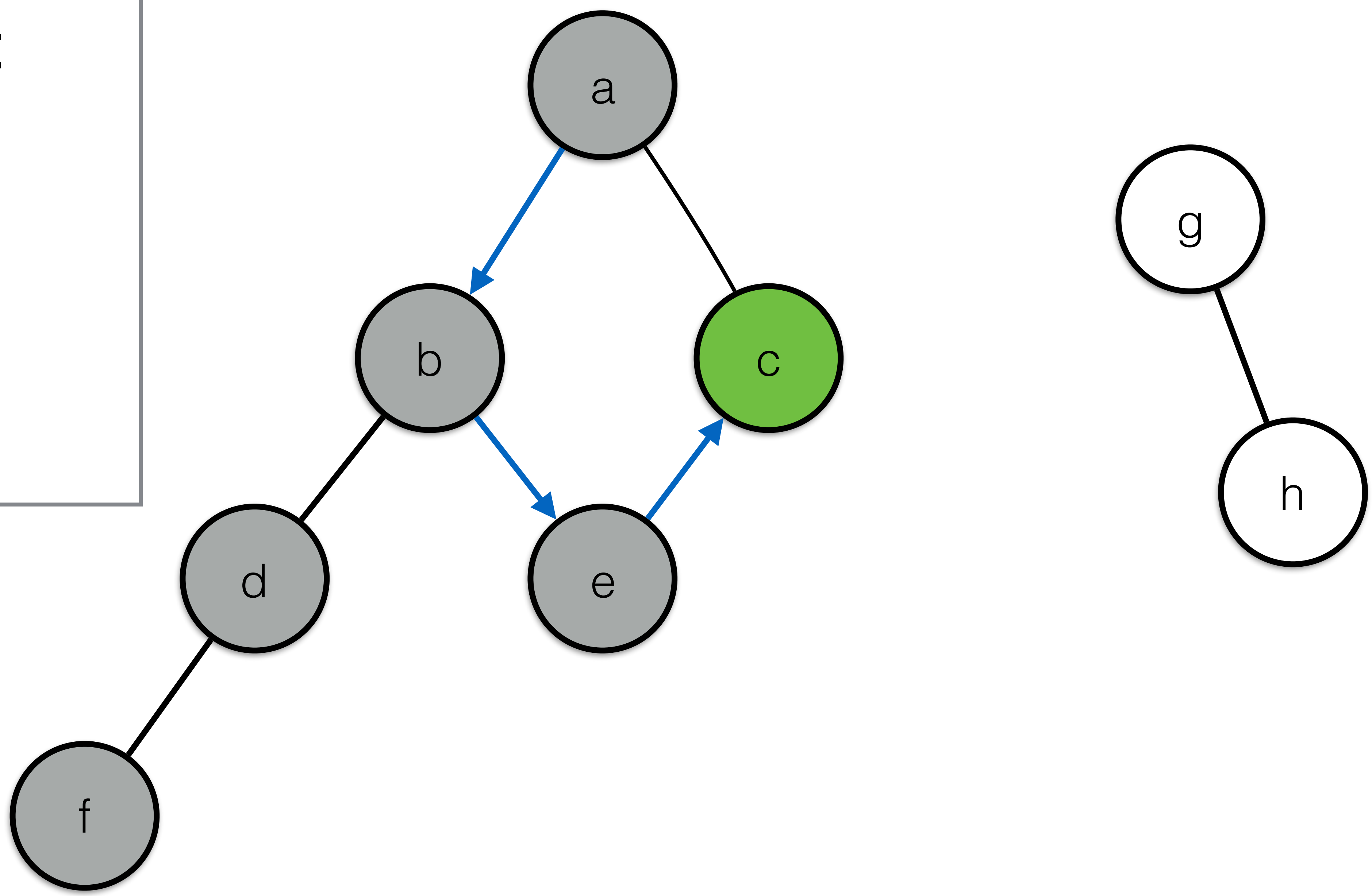


Depth-First Traversal: Stack Discipline

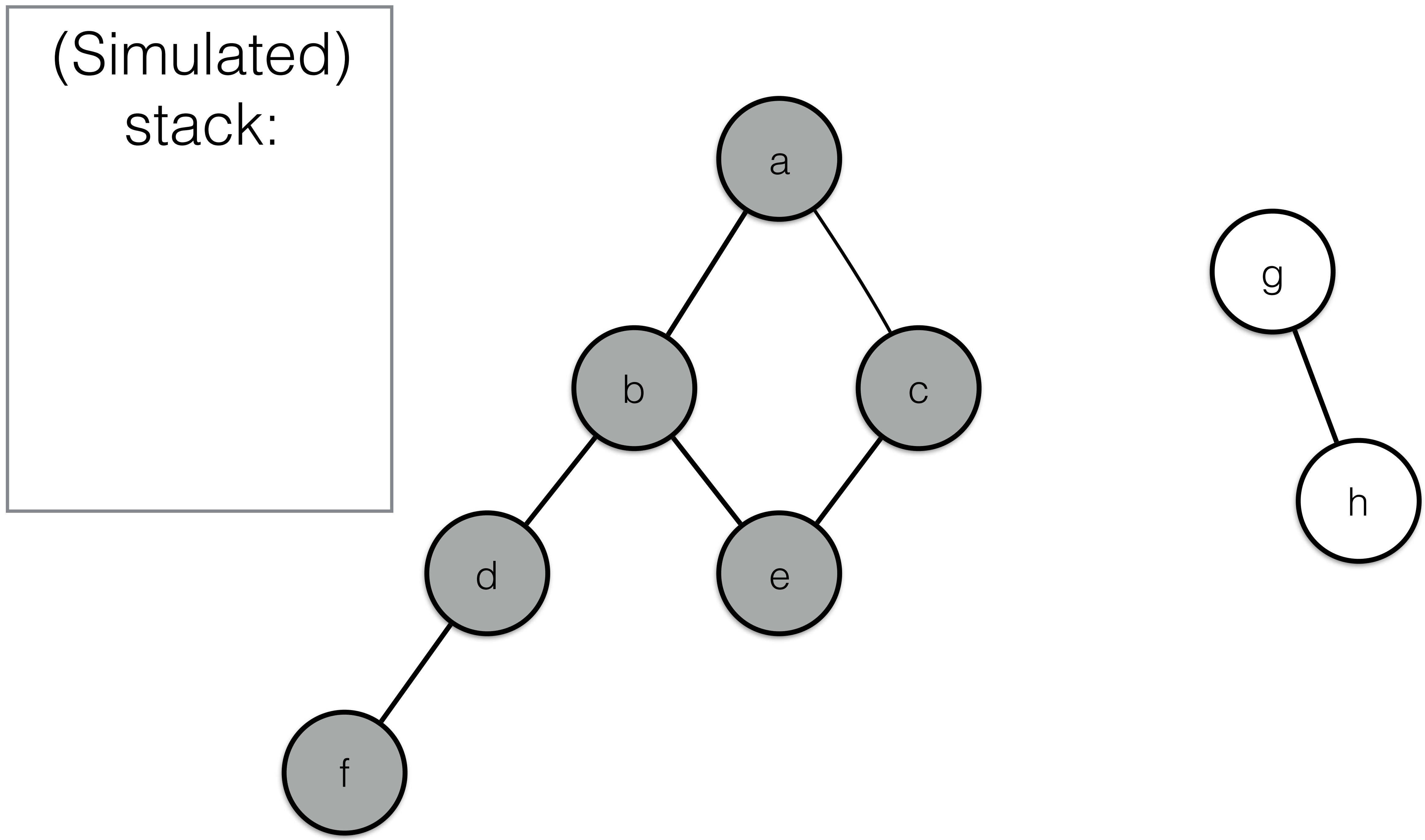
(Simulated) stack:

c
e
b
a

back-tracking …
details omitted …



Depth-First Traversal: Stack Discipline

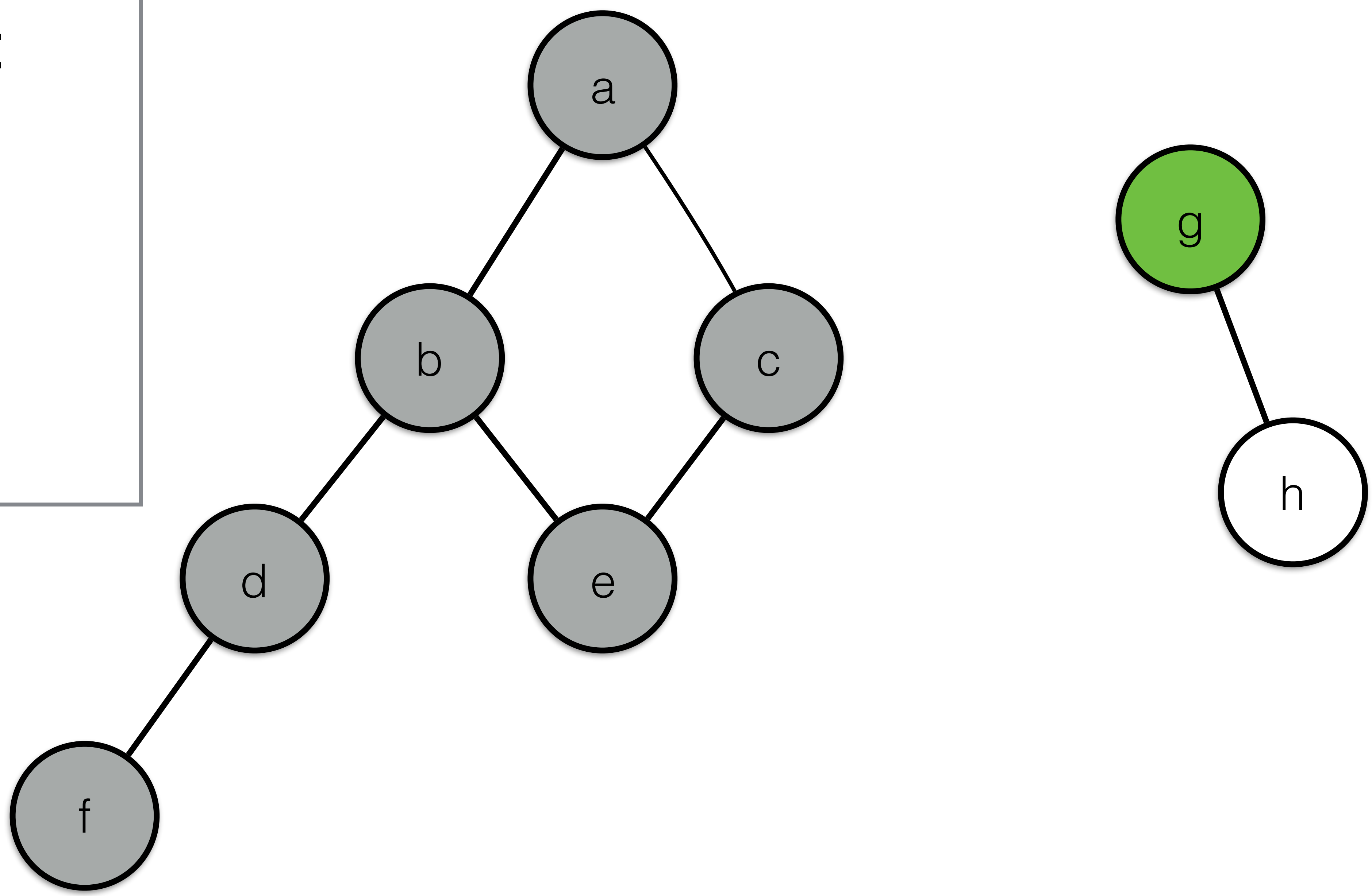


Depth-First Traversal: Stack Discipline

THE UNIVERSITY OF MELBOURNE

(Simulated) stack:

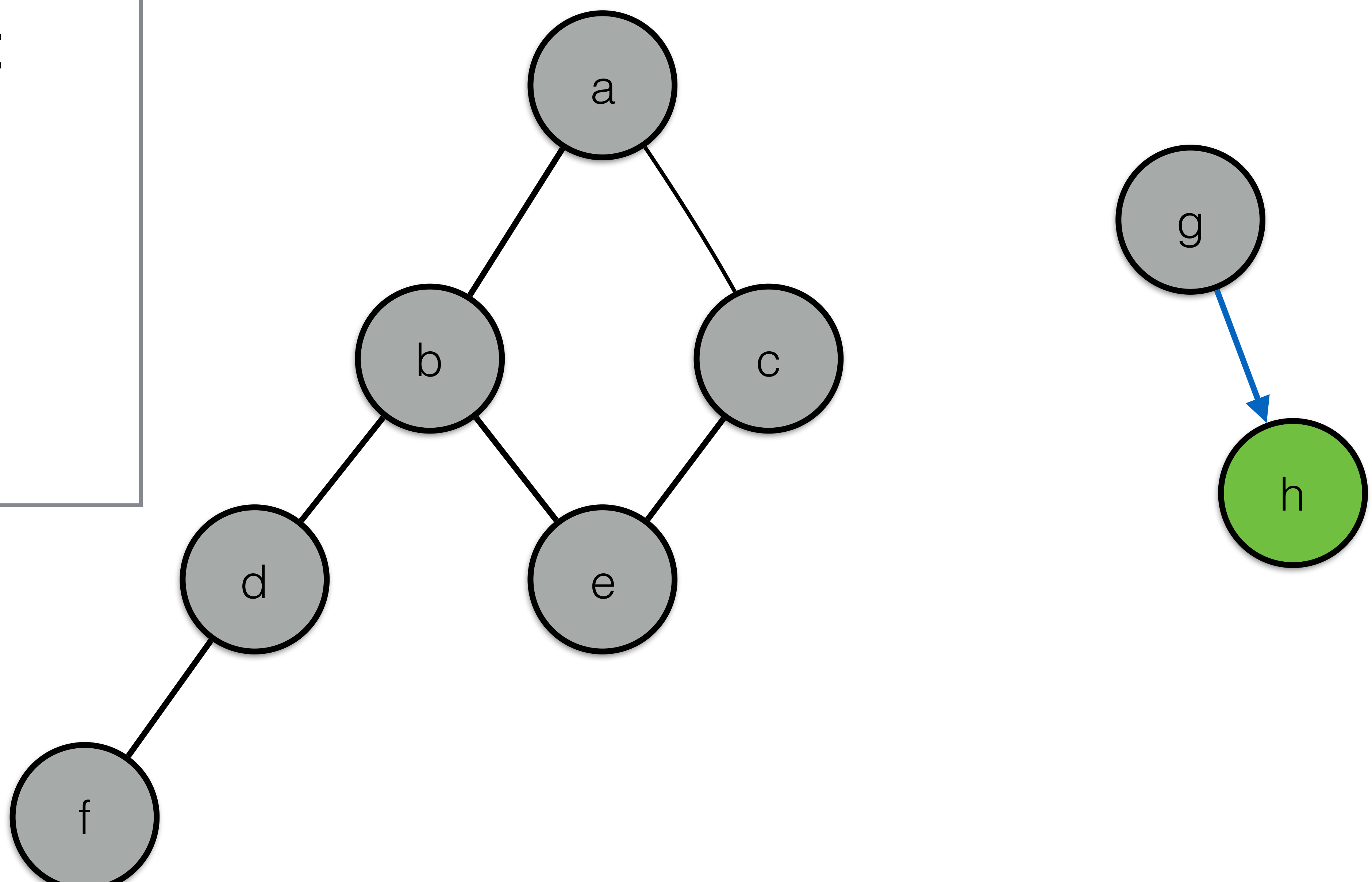
g



Depth-First Traversal: Stack Discipline

(Simulated) stack:

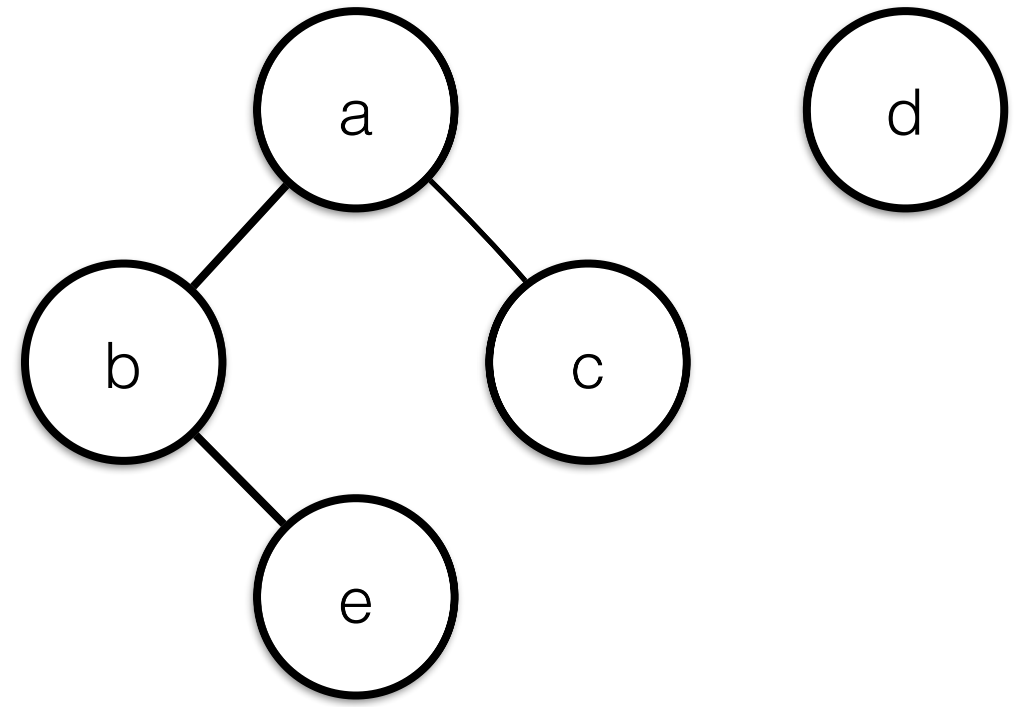
h
g



Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)

function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)

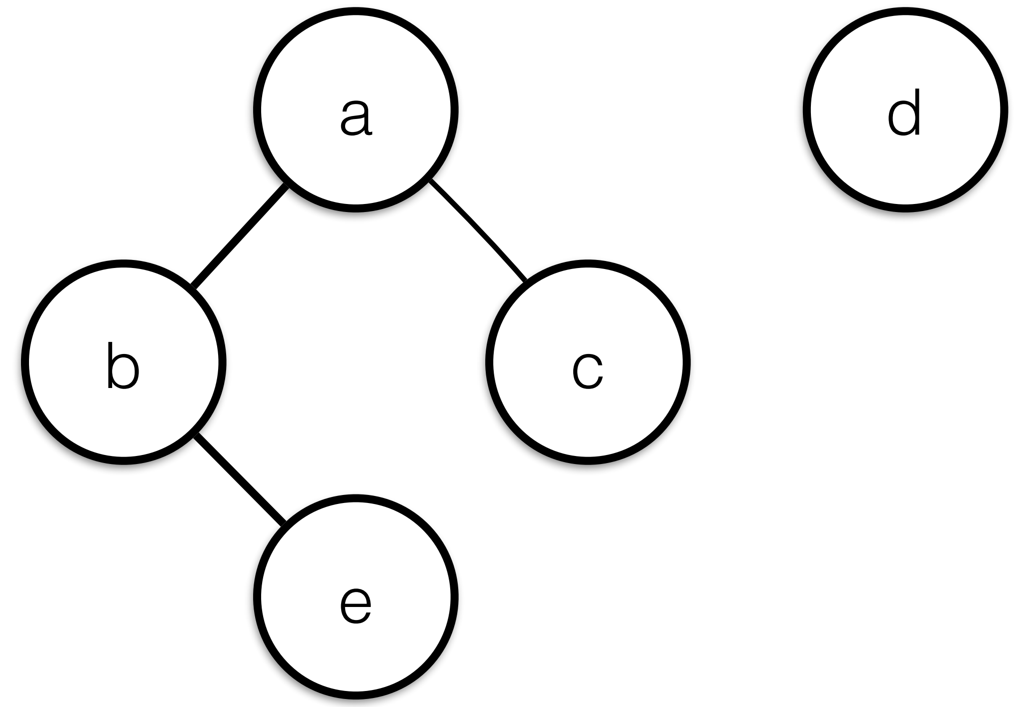


Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)

function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)



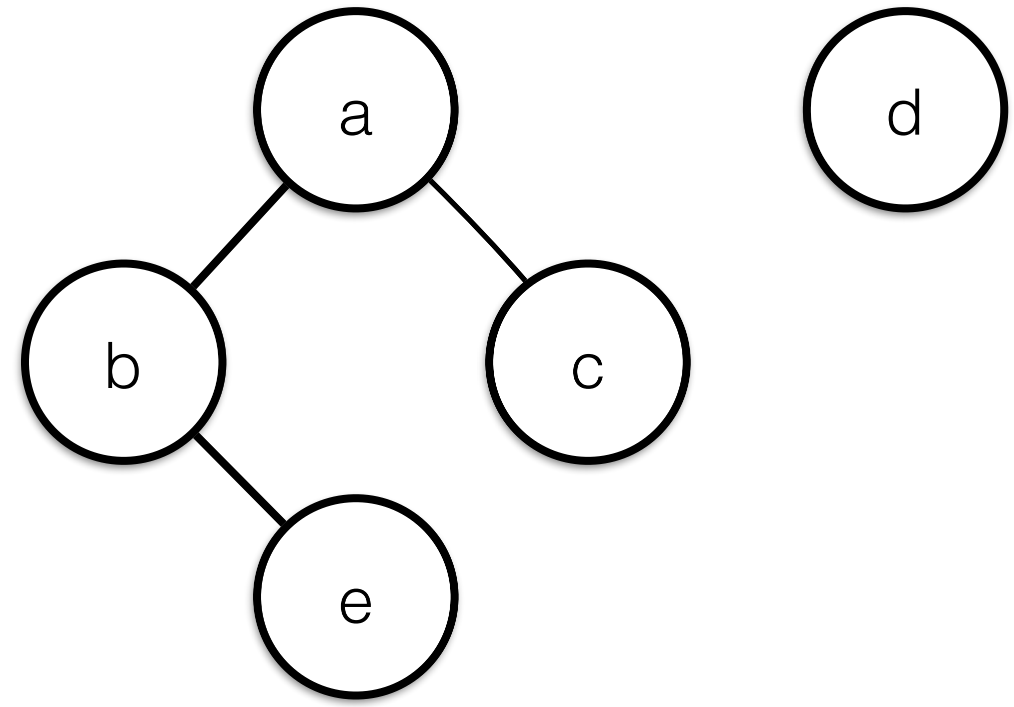
DFS(⟨V,E⟩)

Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      $\text{DfsExplore}(w)$



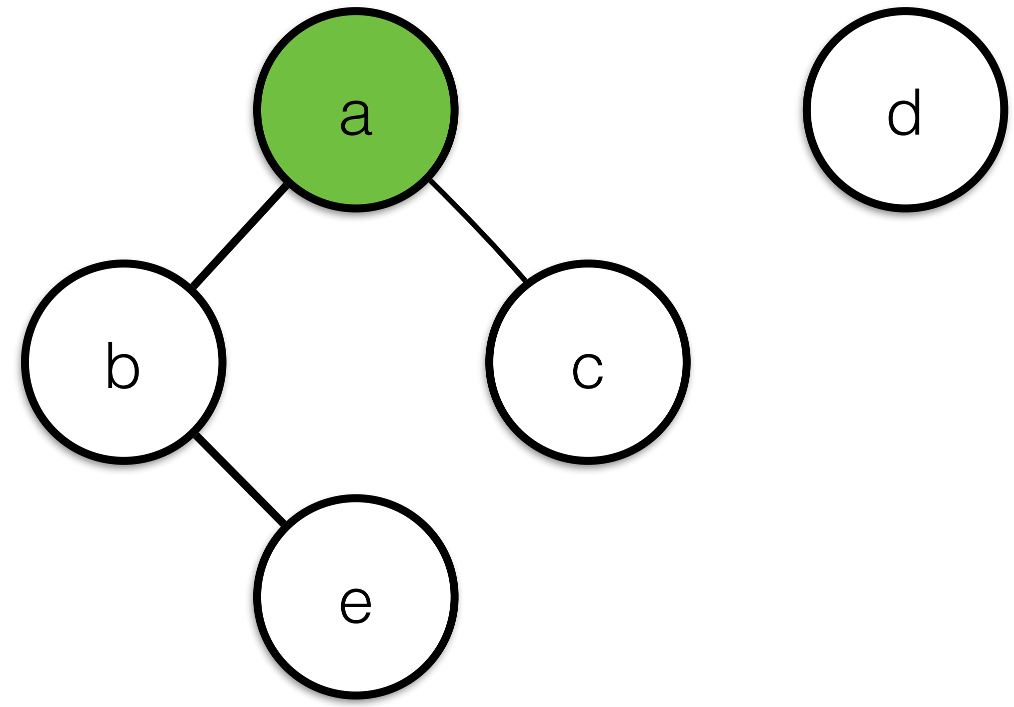
count:
0

DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



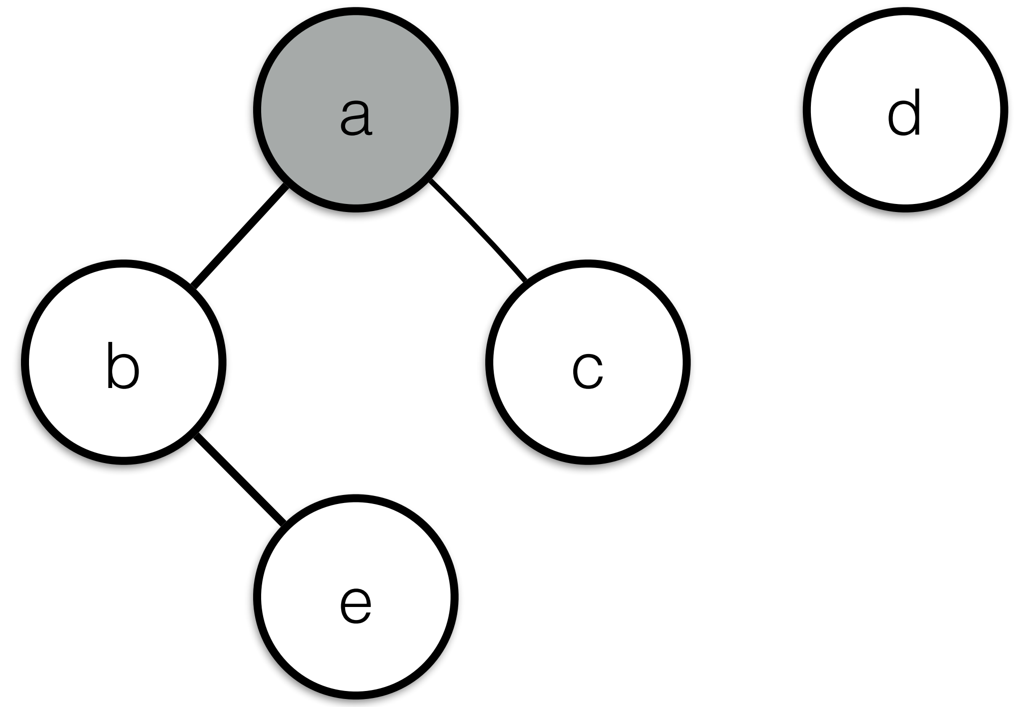
count:
0

DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



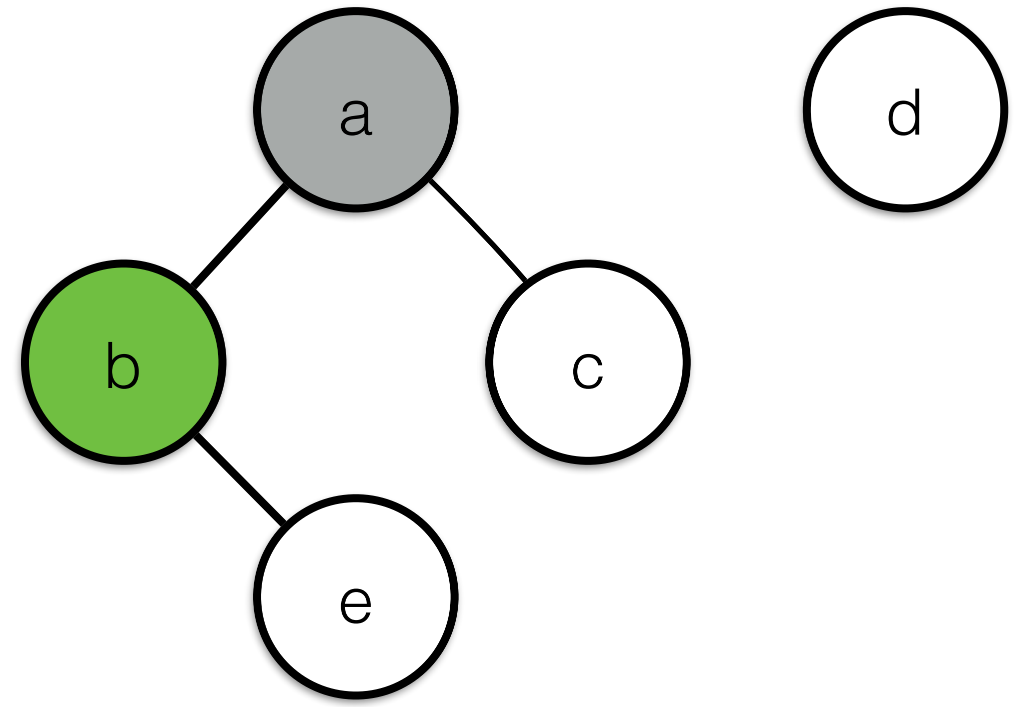
count:
1

DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



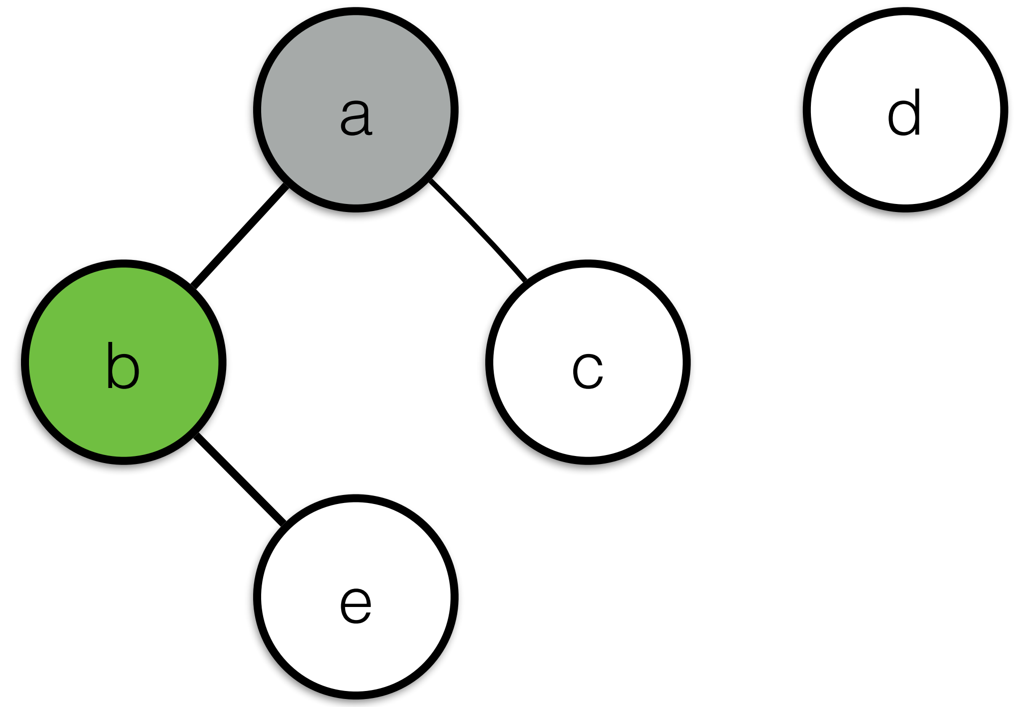
count:
1

DfsExplore(b)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      $\textsc{DfsExplore}(v)$

function $\textsc{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      $\textsc{DfsExplore}(w)$



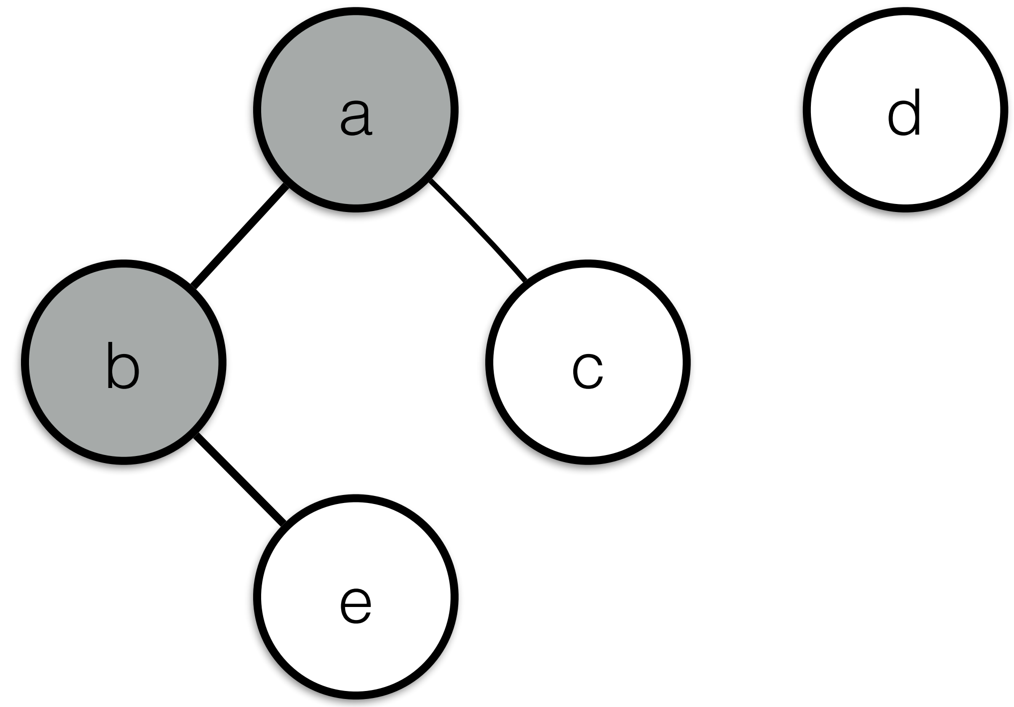
count:
2

DFSEXPLORE(b)
DFSEXPLORE(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            $\text{DfsExplore}(w)$



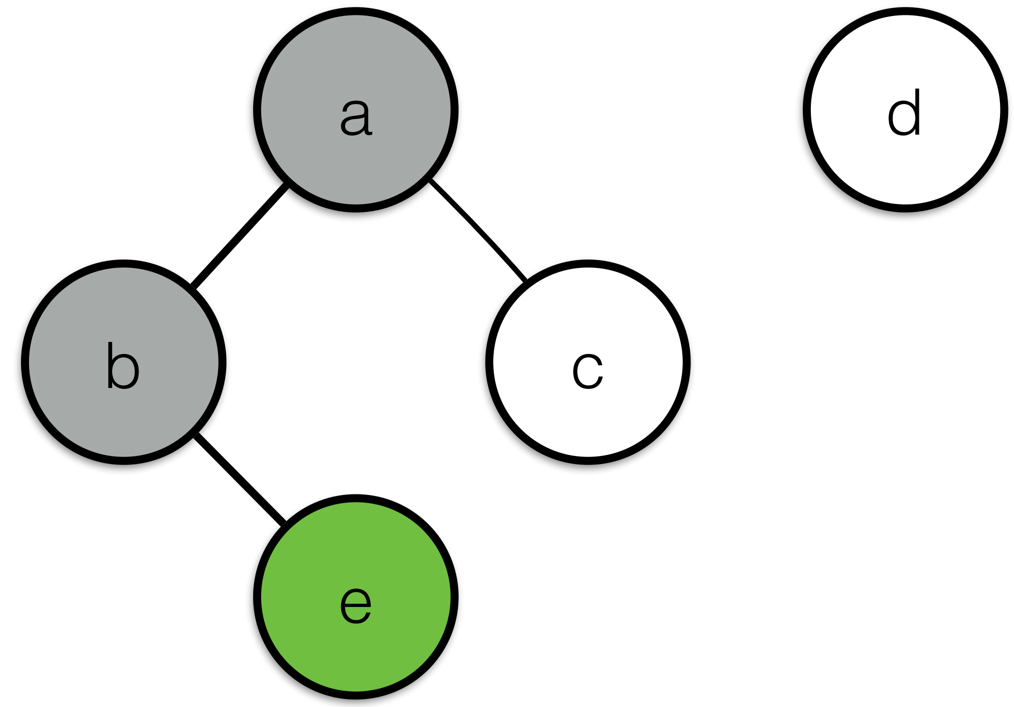
count:
2

DfsExplore(b)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)

function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)



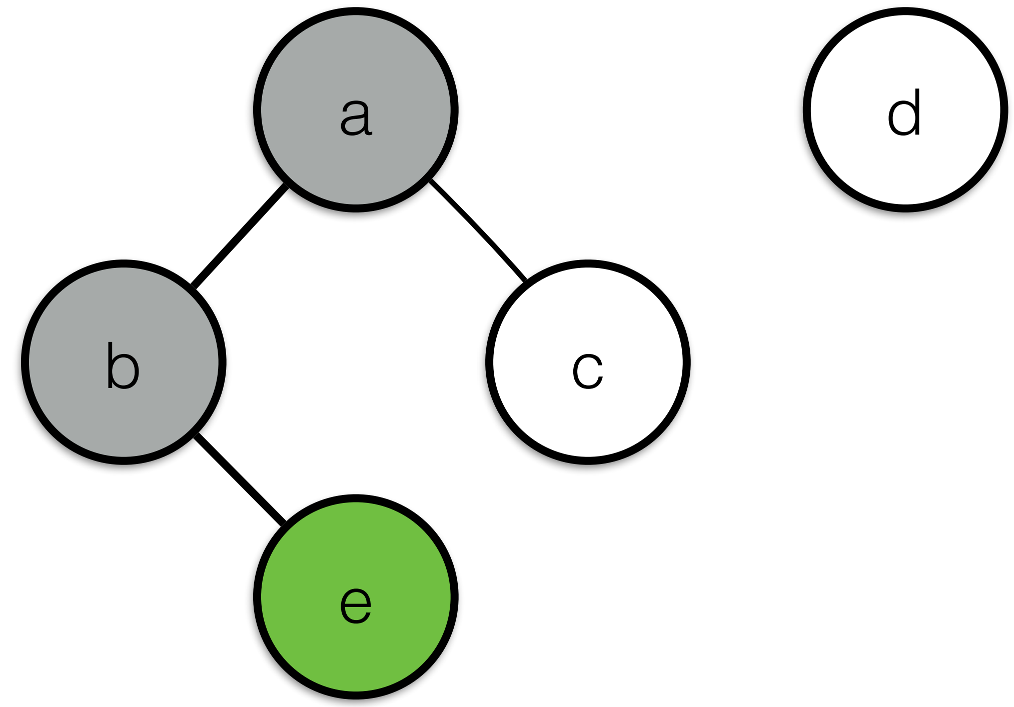
DfsExplore(e)
DfsExplore(b)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

count:
2

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



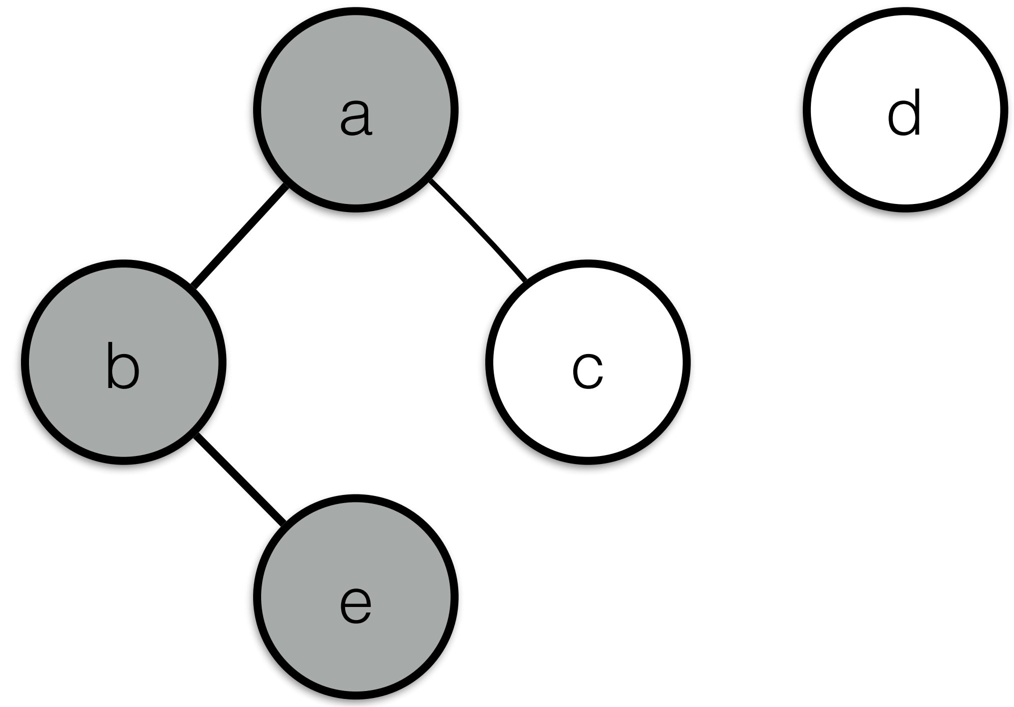
DfsExplore(e)
DfsExplore(b)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

count:
3

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



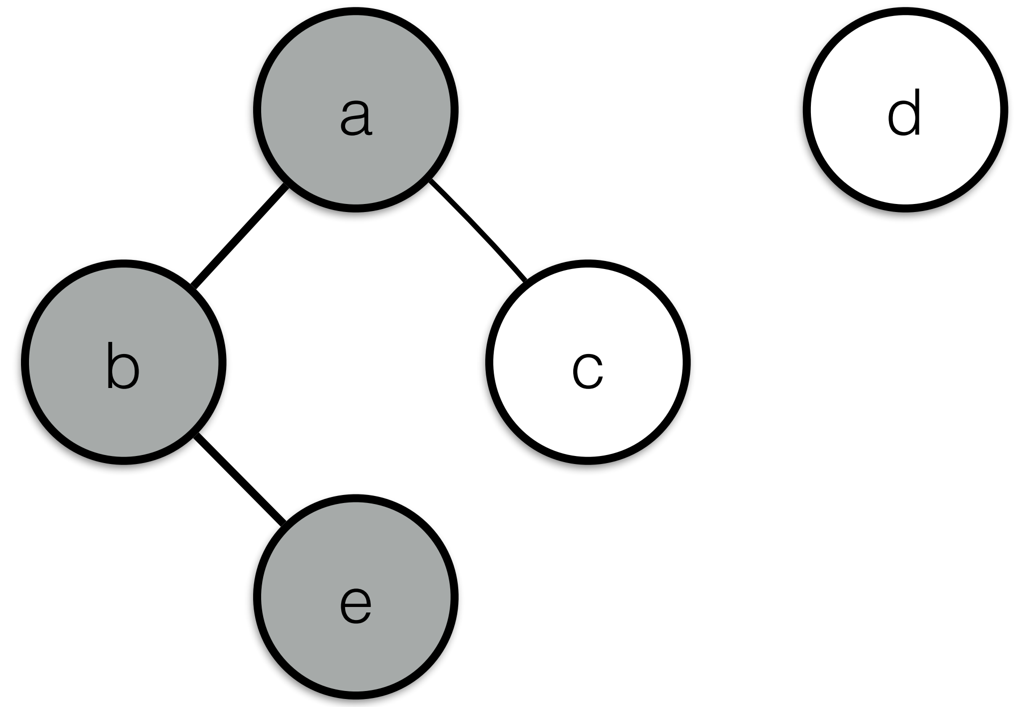
DfsExplore(e)
DfsExplore(b)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

count:
3

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            $\text{DfsExplore}(w)$



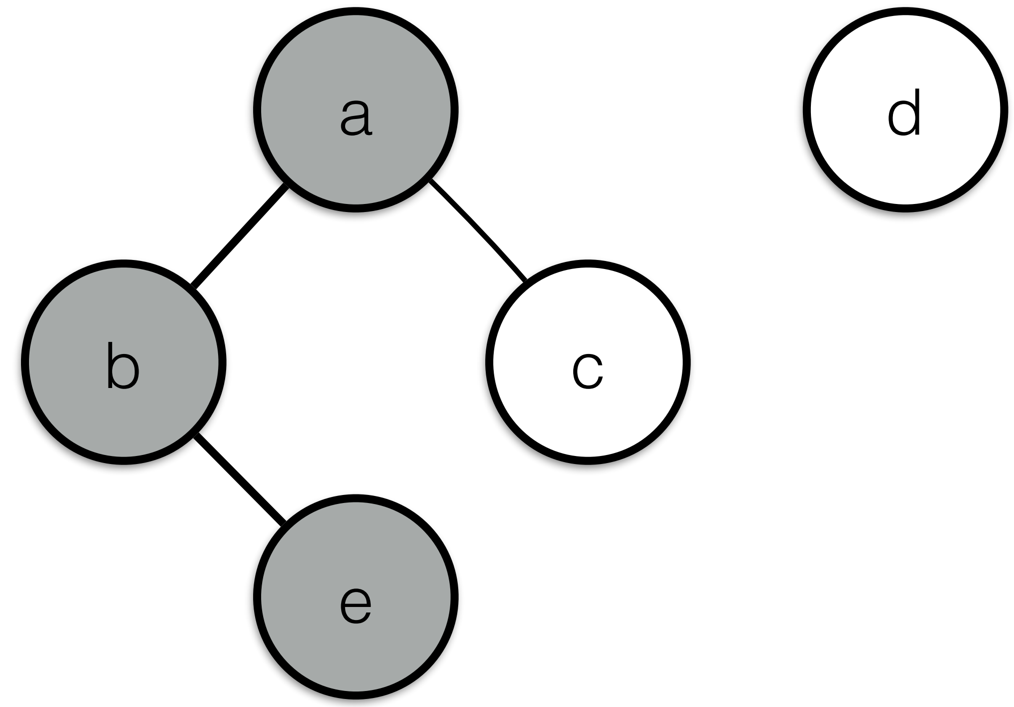
count:
3

DFSEXPLORE(b)
DFSEXPLORE(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            $\text{DfsExplore}(w)$



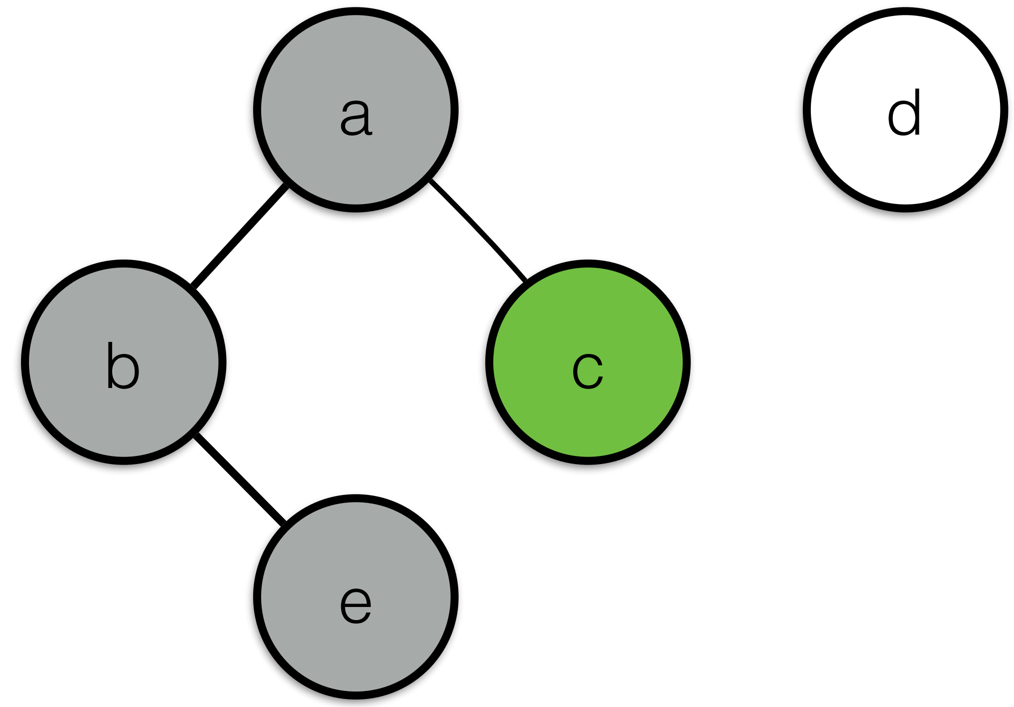
count:
3

DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

THE UNIVERSITY OF MELBOURNE

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)



function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)

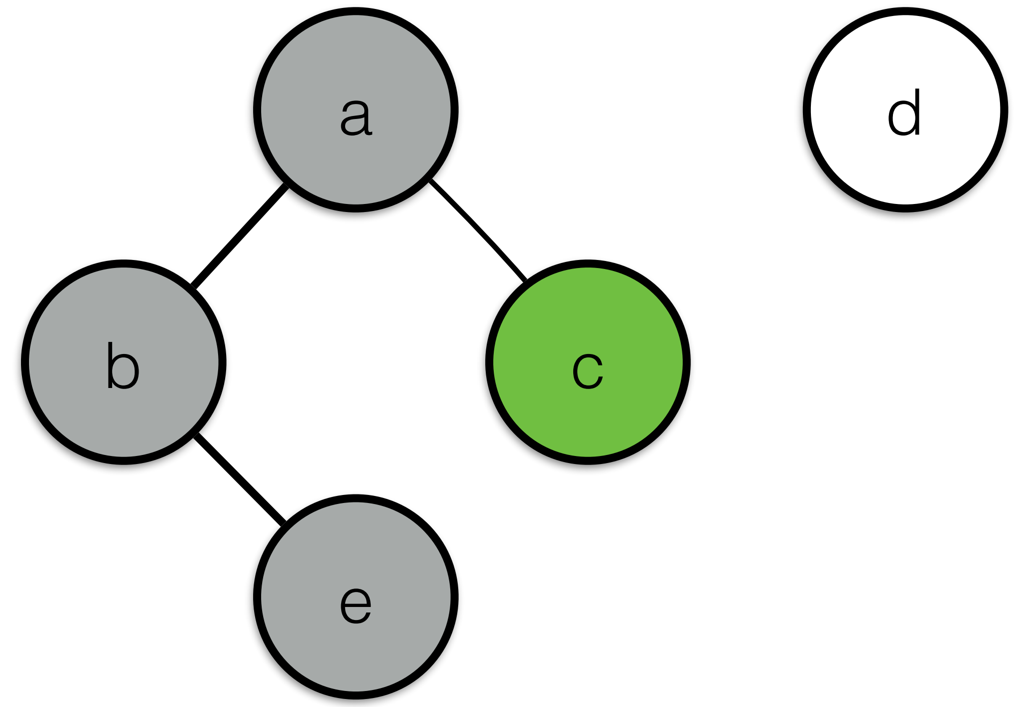
count:
3

DfsExplore(c)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



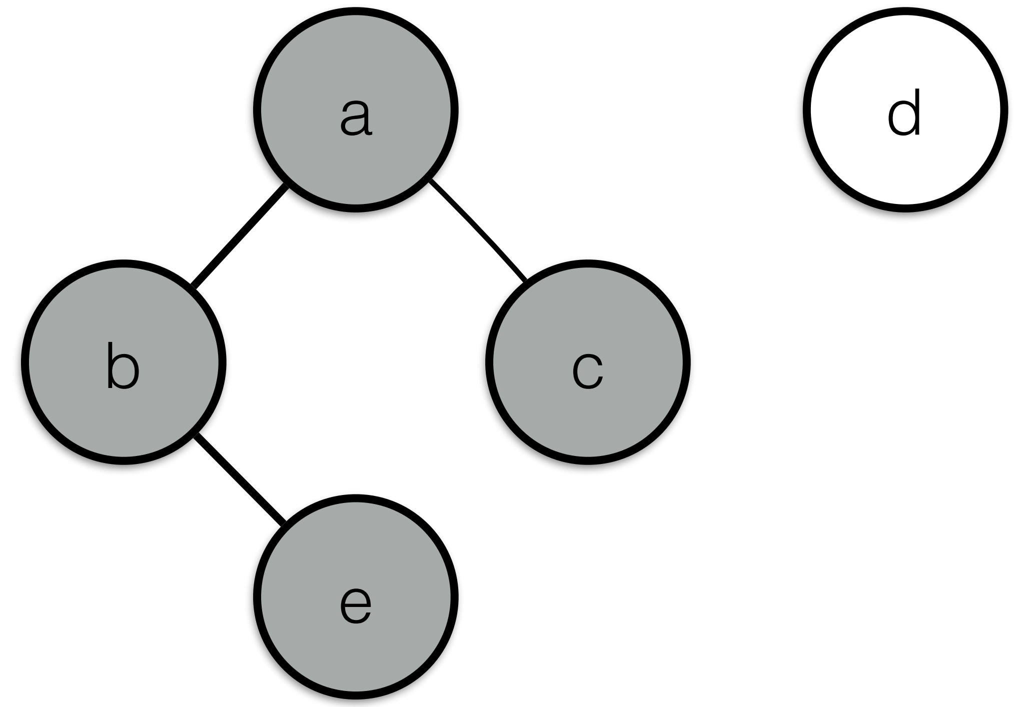
count:
4

DfsExplore(c)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)

function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)



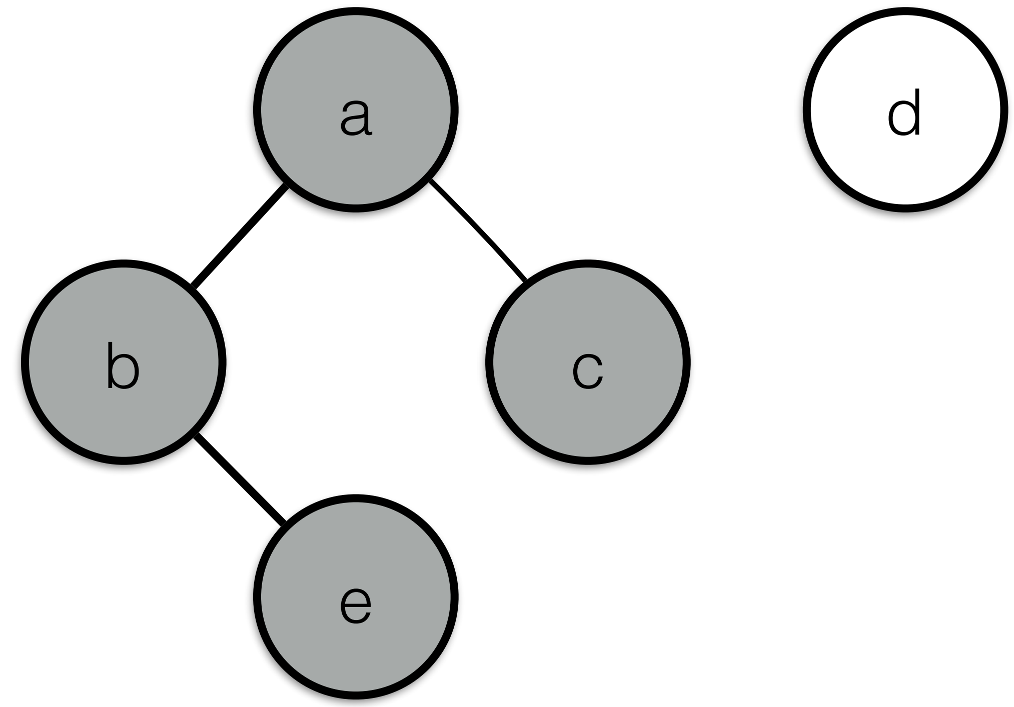
count:
4

DfsExplore(c)
DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)

function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)



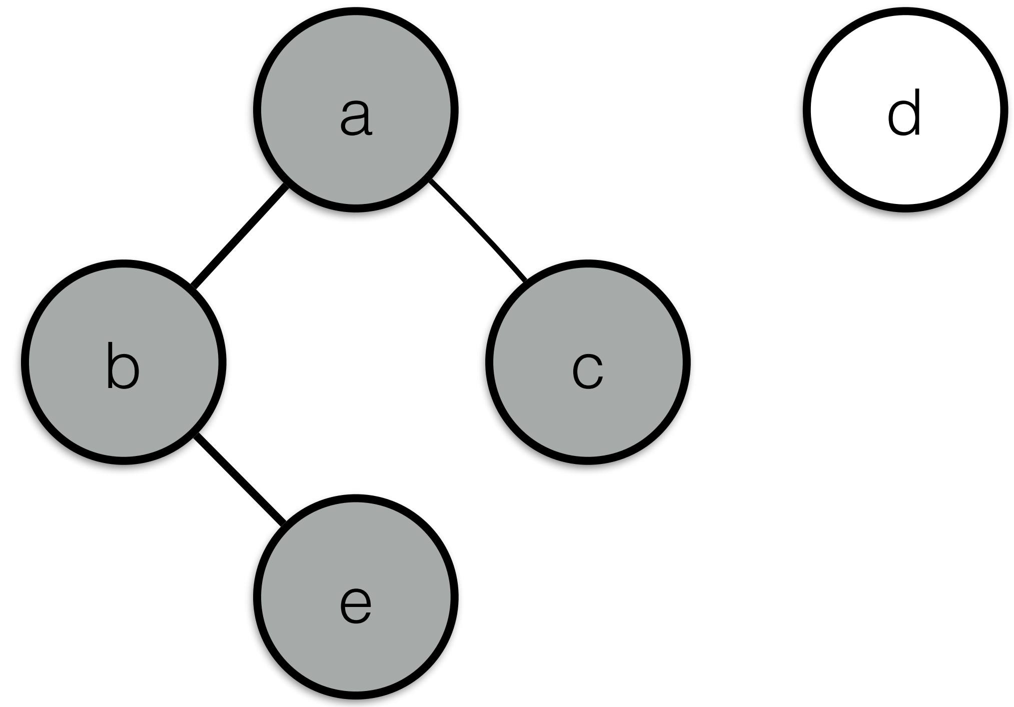
count:
4

DfsExplore(a)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      $\text{DfsExplore}(w)$



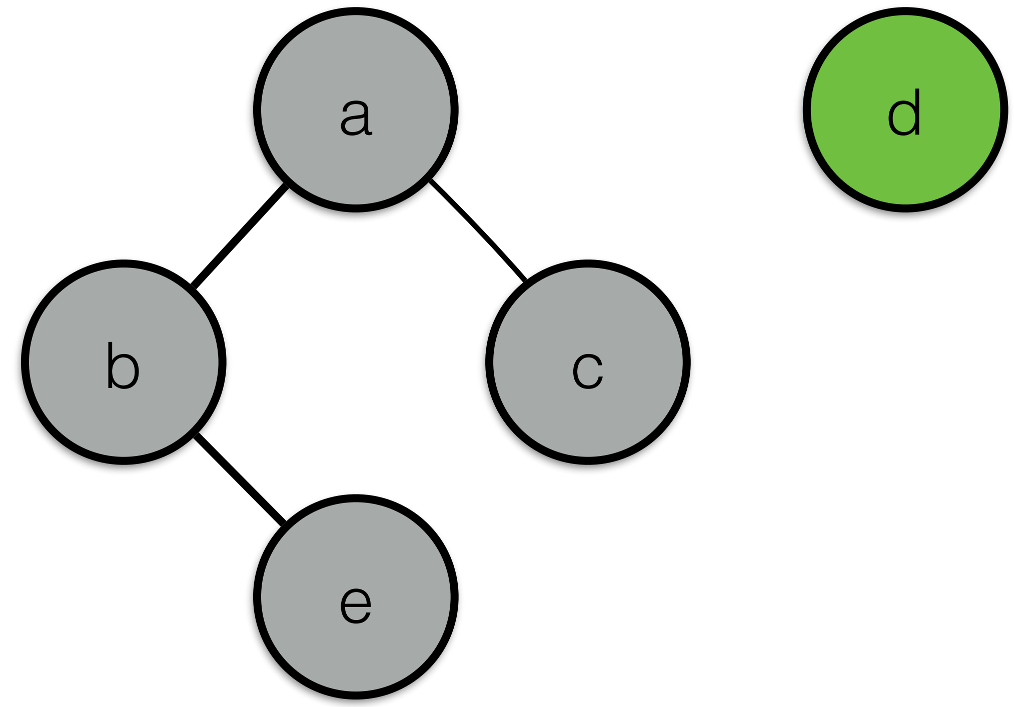
count:
4

DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E\rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      $\text{DfsExplore}(w)$



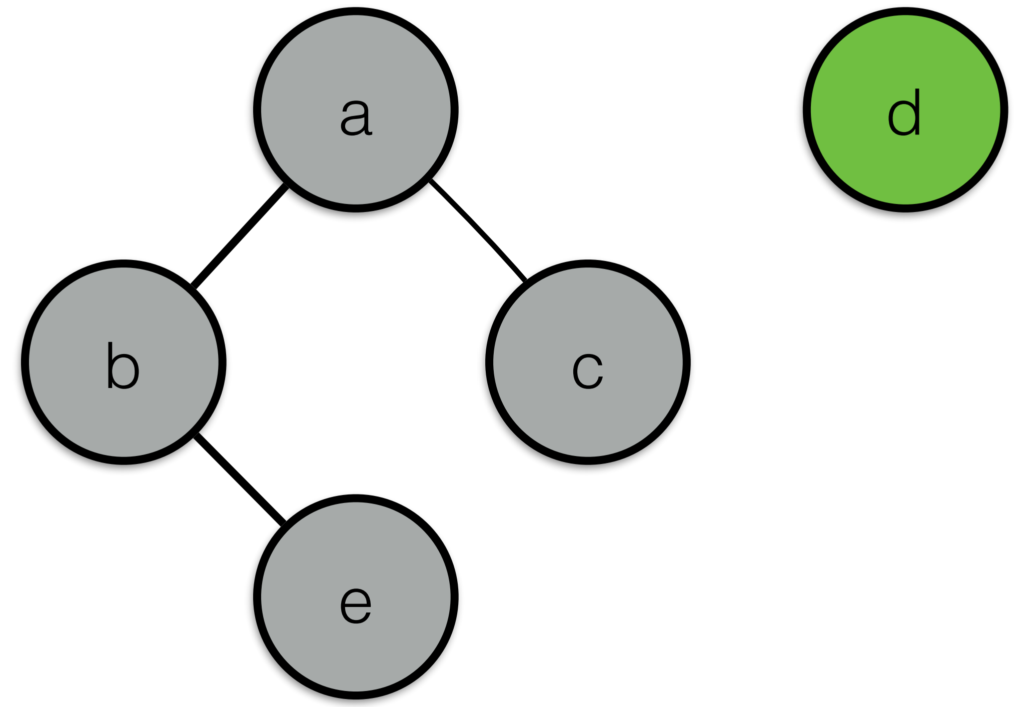
count:
4

DfsExplore(d)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E \rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      $\text{DfsExplore}(w)$



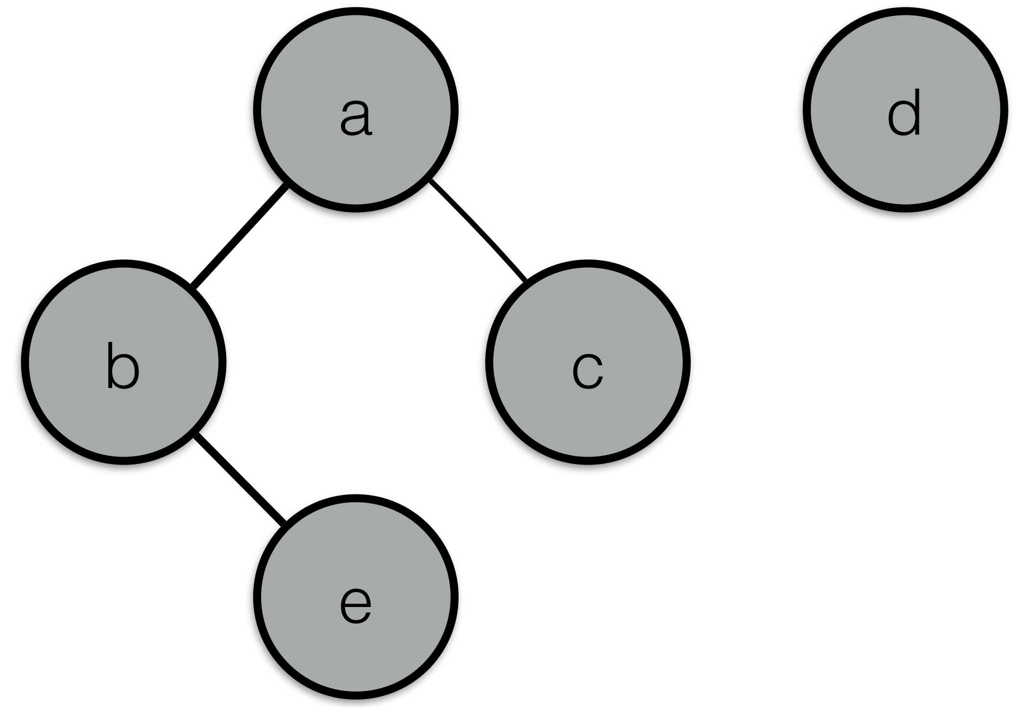
count:
5

DfsExplore(d)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      DfsExplore(v)

function DfsExplore(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      DfsExplore(w)



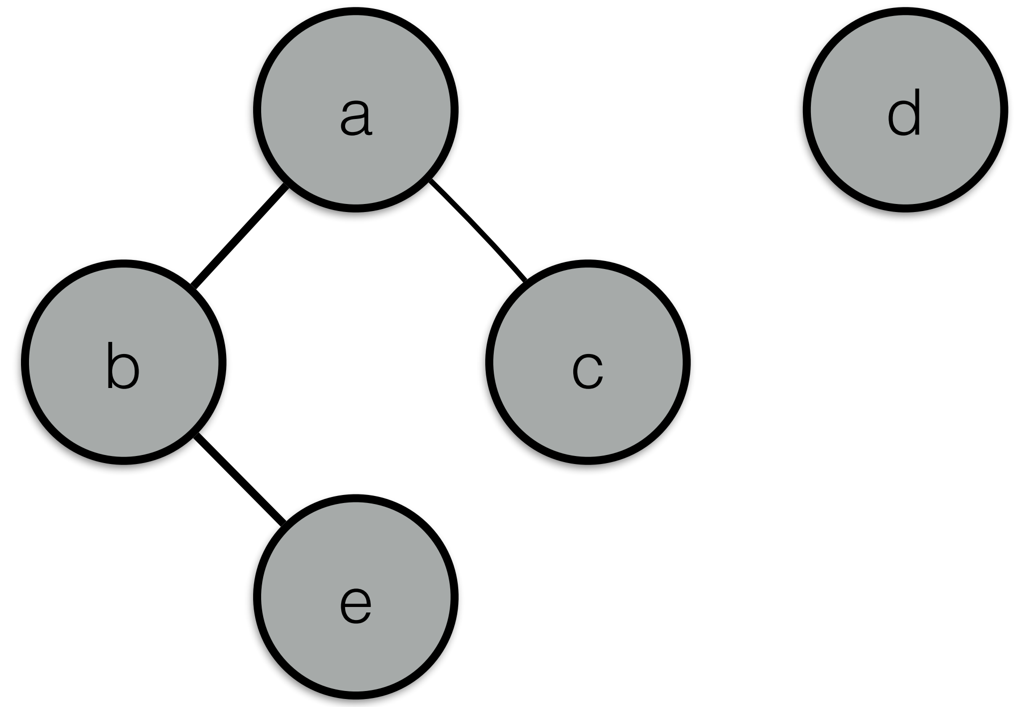
count:
5

DfsExplore(d)
DFS(⟨V,E⟩)
Call Stack

Depth-First Traversal: Recursive Implementation

function DFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            DfsExplore(v)

function DfsExplore(v)
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            DfsExplore(w)



count:
5

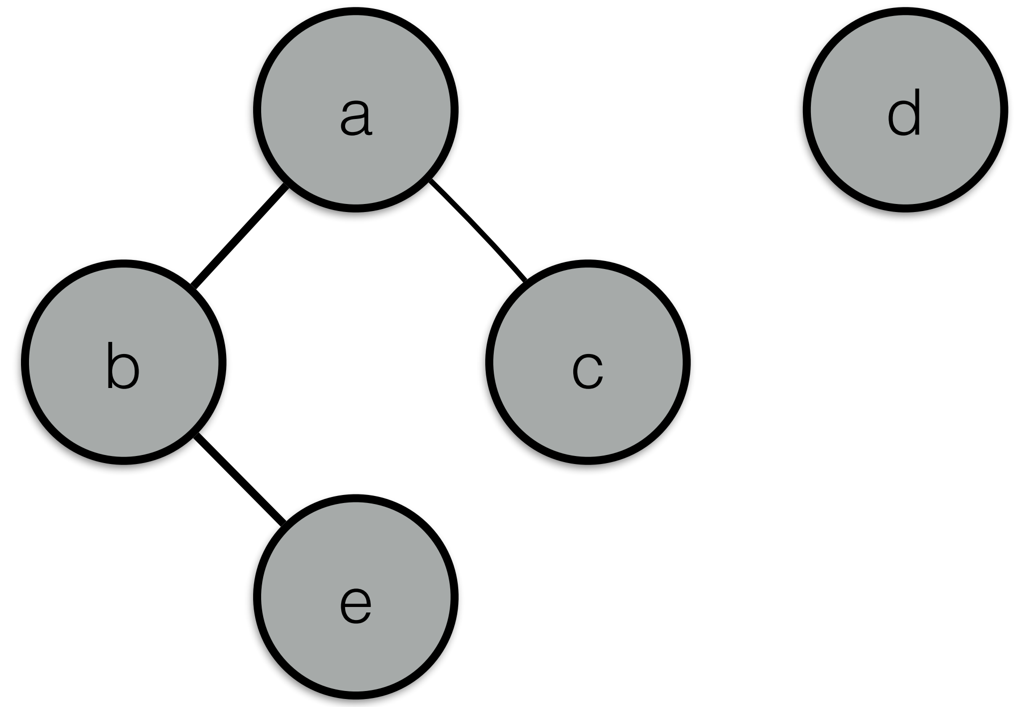
DFS($\langle$V,E$\rangle$)

Call Stack

Depth-First Traversal: Recursive Implementation

function $\text{DFS}(\langle V, E\rangle)$
    mark each node in V with 0
    $count \leftarrow 0$
    for each v in V **do**
        if v is marked with 0 **then**
            $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
    $count \leftarrow count + 1$
    mark v with $count$
    for each edge (v, w) **do**
        if w is marked with 0 **then**
            $\text{DfsExplore}(w)$



count:
5

Call Stack

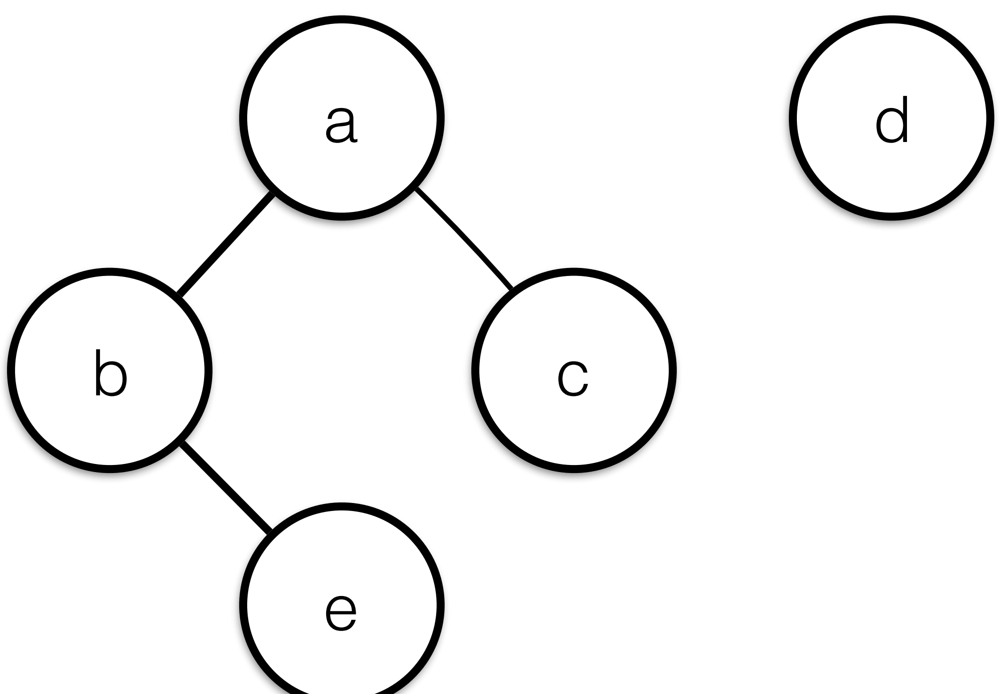
Depth-First Search: Recursive Algorithm Notes

- Works both for directed and undirected graphs.
- The "marking" of nodes is usually done by maintaining a separate array, **`mark`**, indexed by V .
- For example, when we wrote "mark v with count", that would be implemented as "**`mark[v] := count`**".
- How to find the nodes adjacent to v depends on the graph representation used.
- Using an adjacency **matrix** **`adj`**, we need to consider **`adj[v,w]`** for each w in V . Here the complexity of graph traversal is $\Theta(|V|^2)$.
- Using adjacency **lists**, for each v , we traverse the list **`adj[v]`**. In this case, the complexity of traversal is $\Theta(|V| + |E|)$.

Applications of Depth-First Search (DFS)

- Easy to adapt DFS to decide if a graph is connected.
- How?

After visiting the first node in the for loop, if count is equal to the number of nodes in the graph, thn the graph is connected.

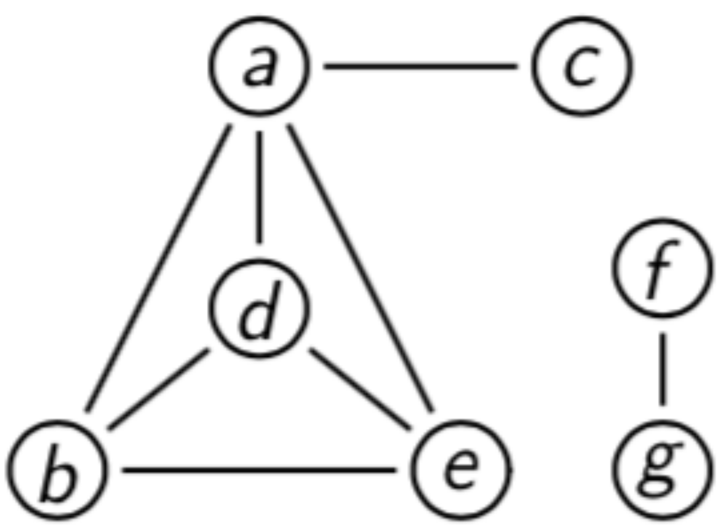


function $\text{DFS}(\langle V, E\rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked with 0 **then**
      $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do**
    if w is marked with 0 **then**
      $\text{DfsExplore}(w)$

Depth-First Search: Node Orderings

- We can order nodes **either** by the order in which they get **pushed onto** the stack, or by the order in which they are **popped from** the stack

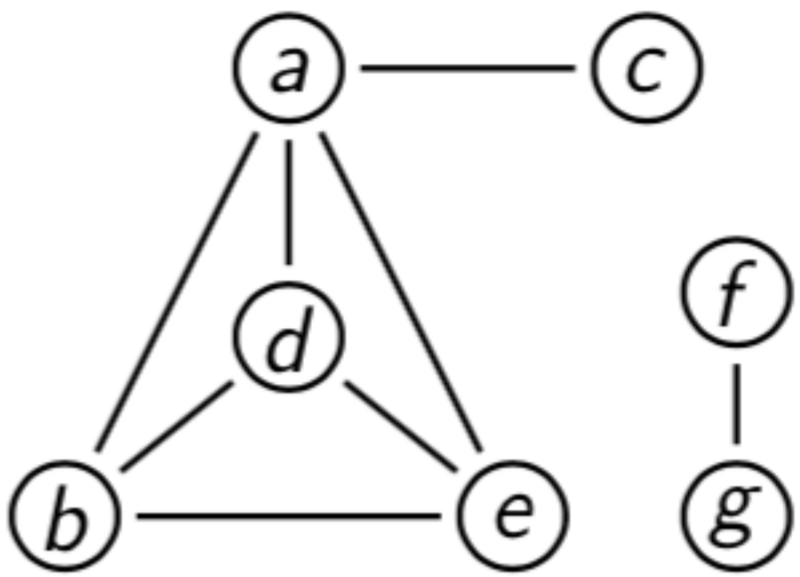


Levitin's compact stack notation

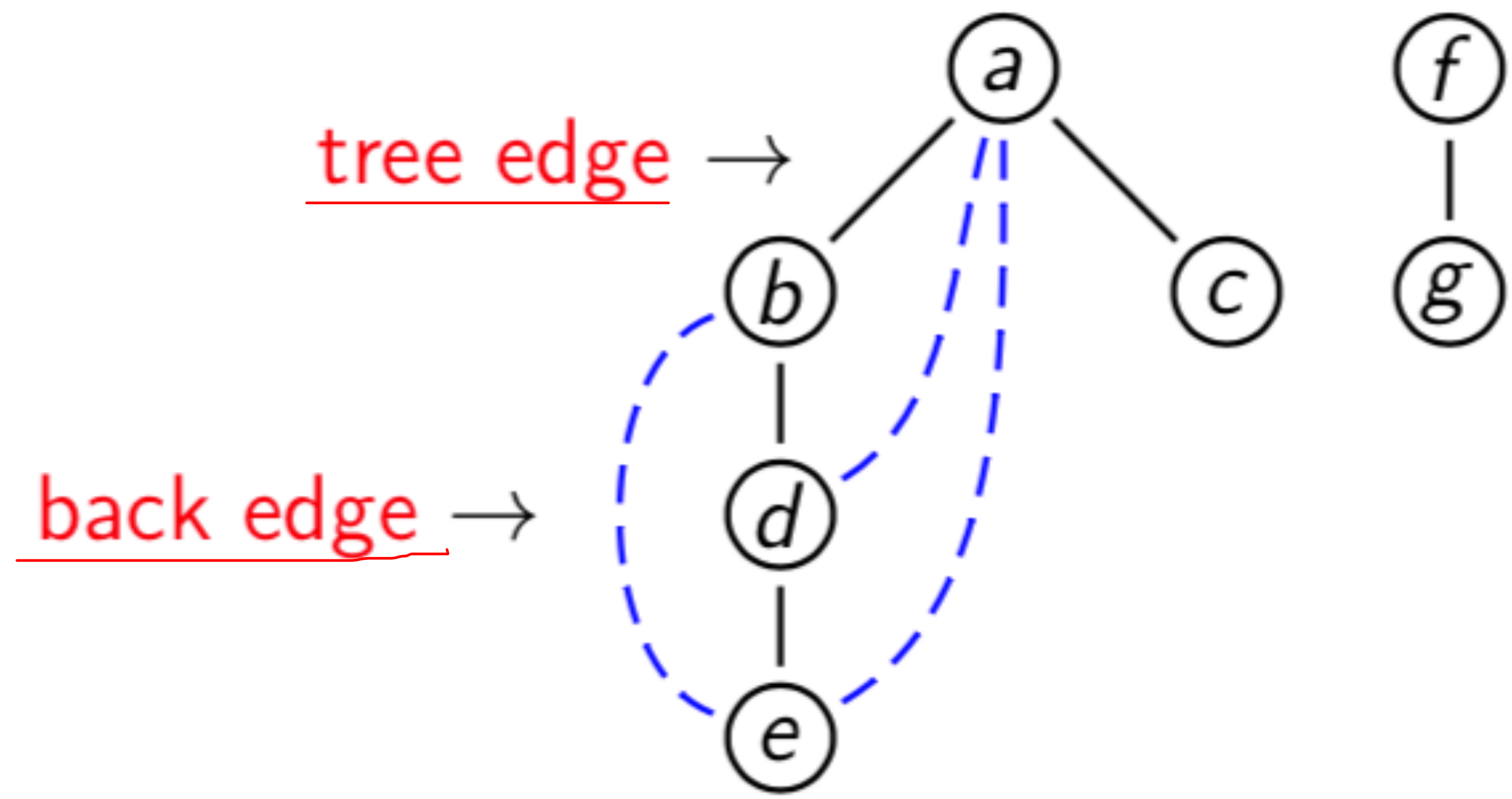
$$\begin{array}{lll} e_{4,1} & & \\ d_{3,2} & & \\ b_{2,3} & c_{5,4} & g_{7,6} \\ a_{1,5} & & f_{6,7} \end{array}$$

The first subscripts give the order in which nodes are pushed, the second the order in which they are popped off the stack.

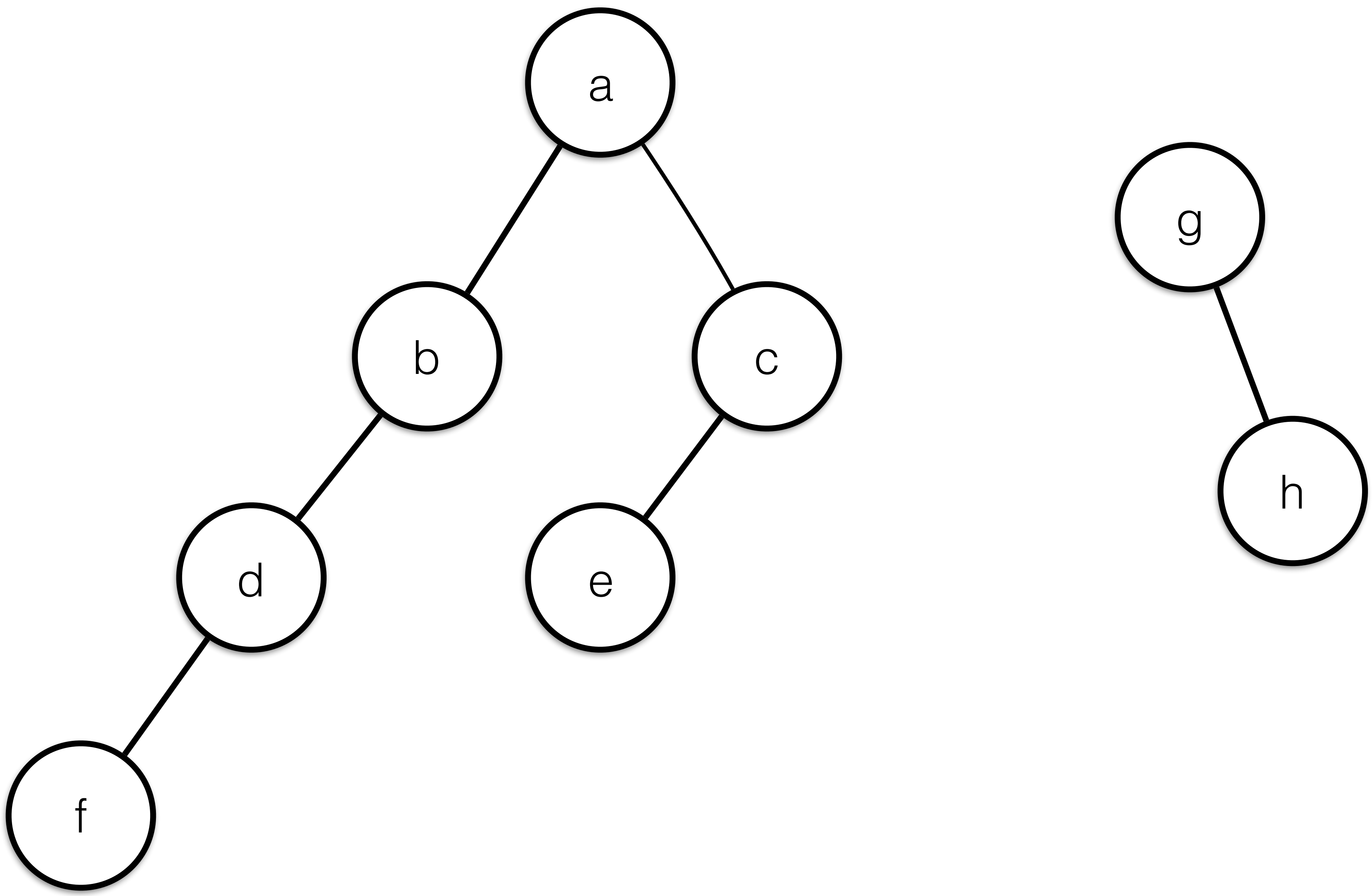
Depth-First Search Forest



DFS can be depicted by the resulting **DFS Forest** (DFS **Tree** for a connected graph)



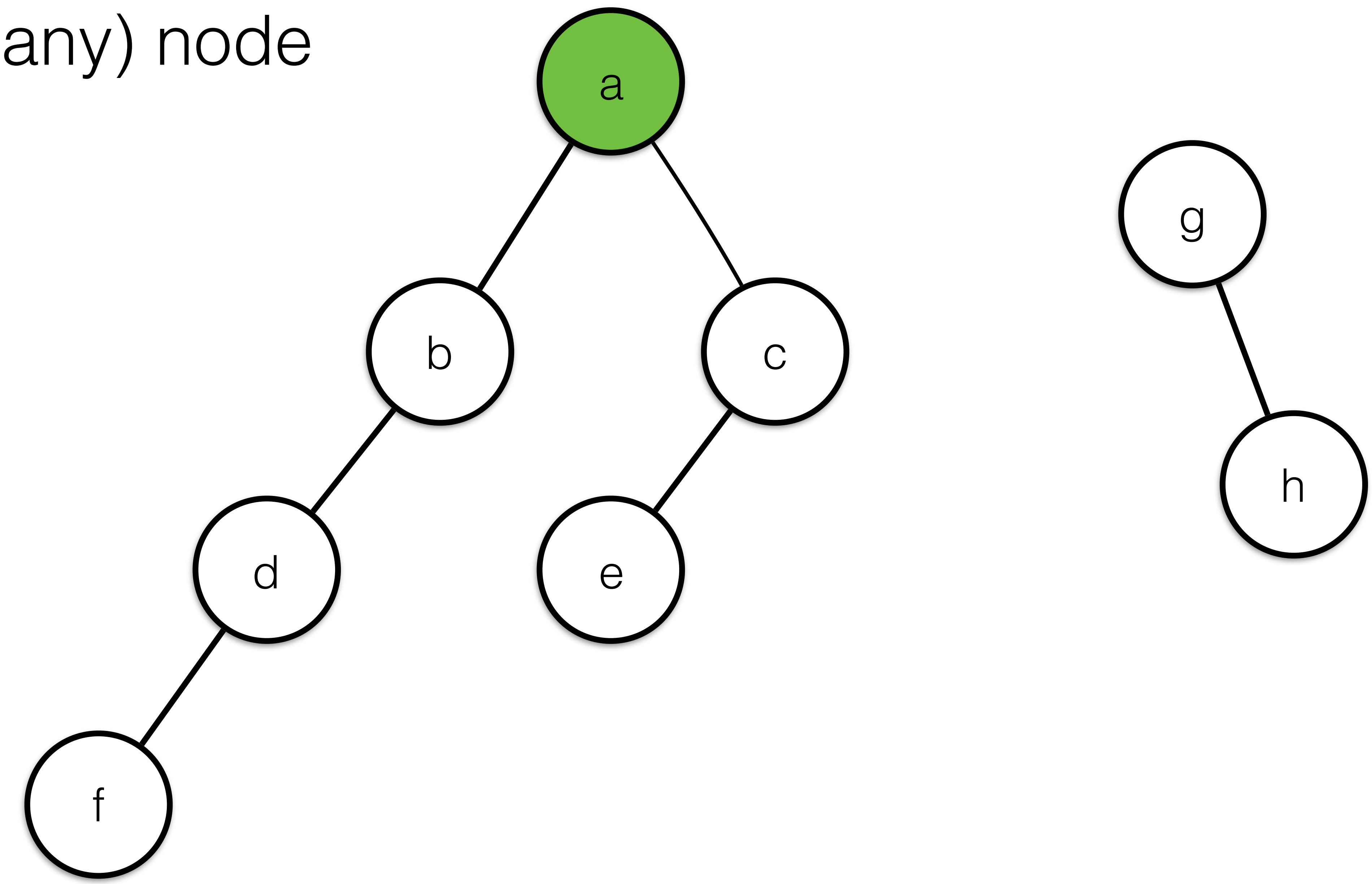
Breadth-First Traversal



Breadth-First Traversal

Nodes visited in this order: a

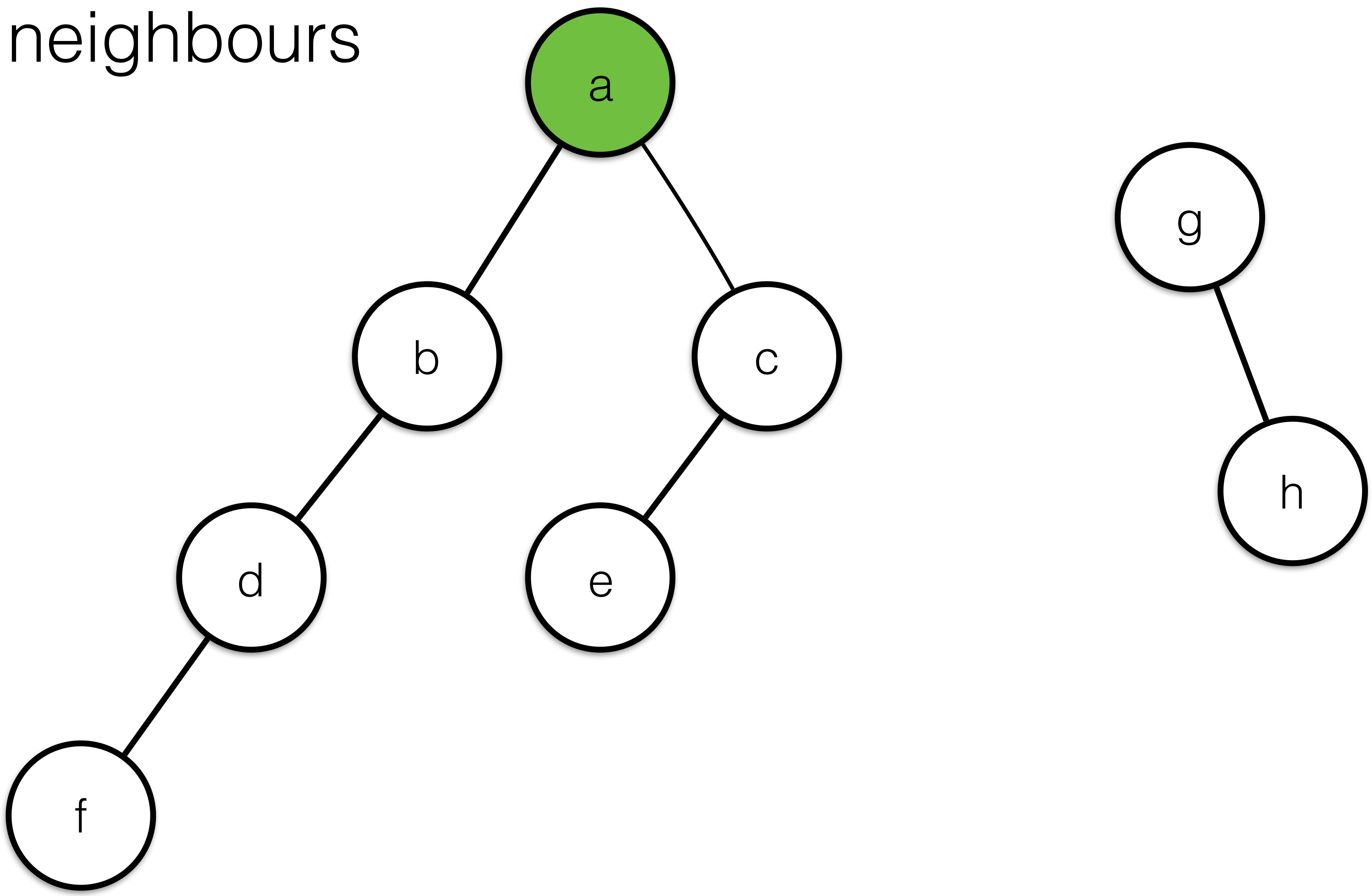
Start with (any) node



Breadth-First Traversal

Nodes visited in this order: a

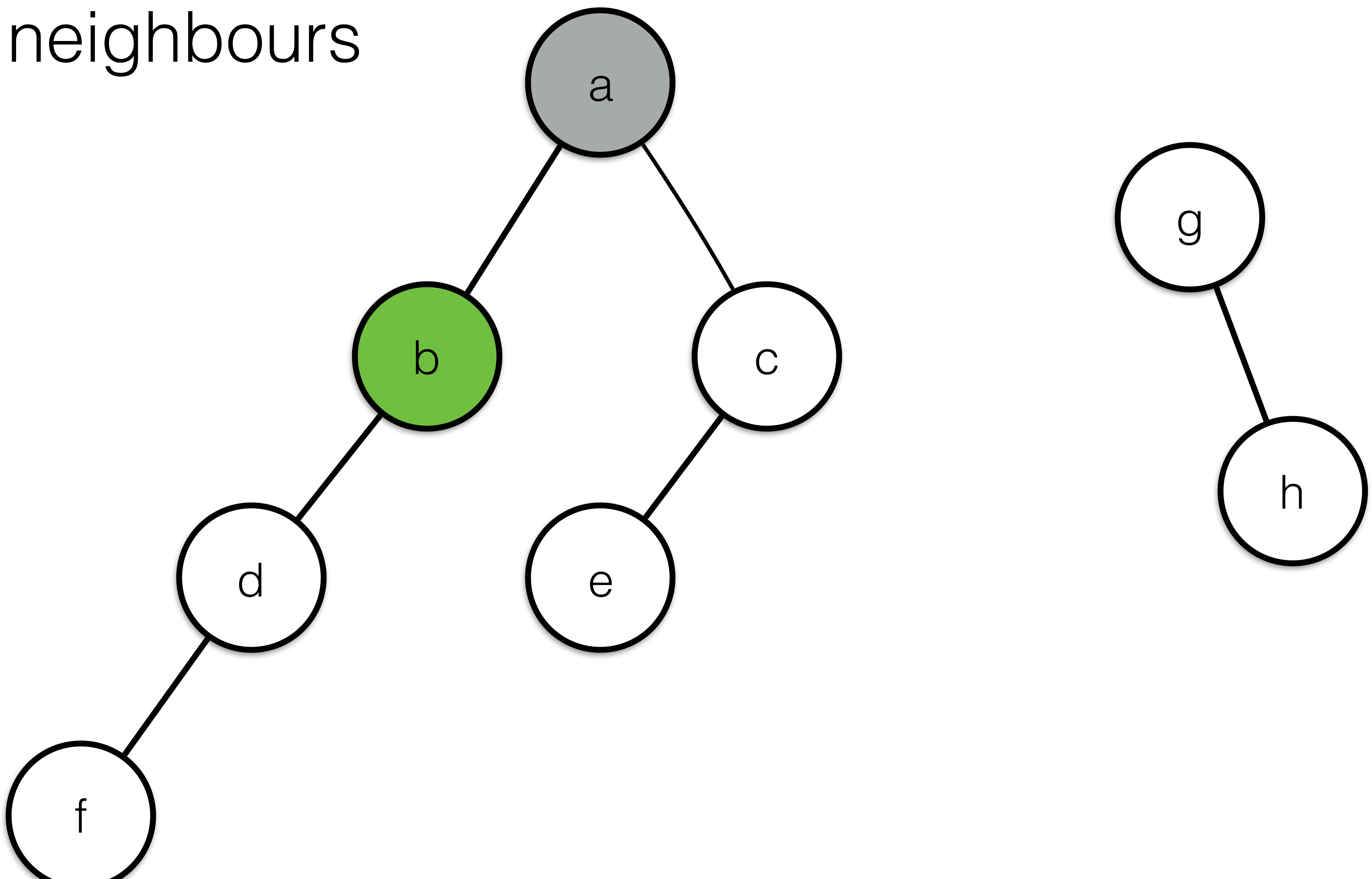
Visit all of its neighbours



Breadth-First Traversal

Nodes visited in this order: a b

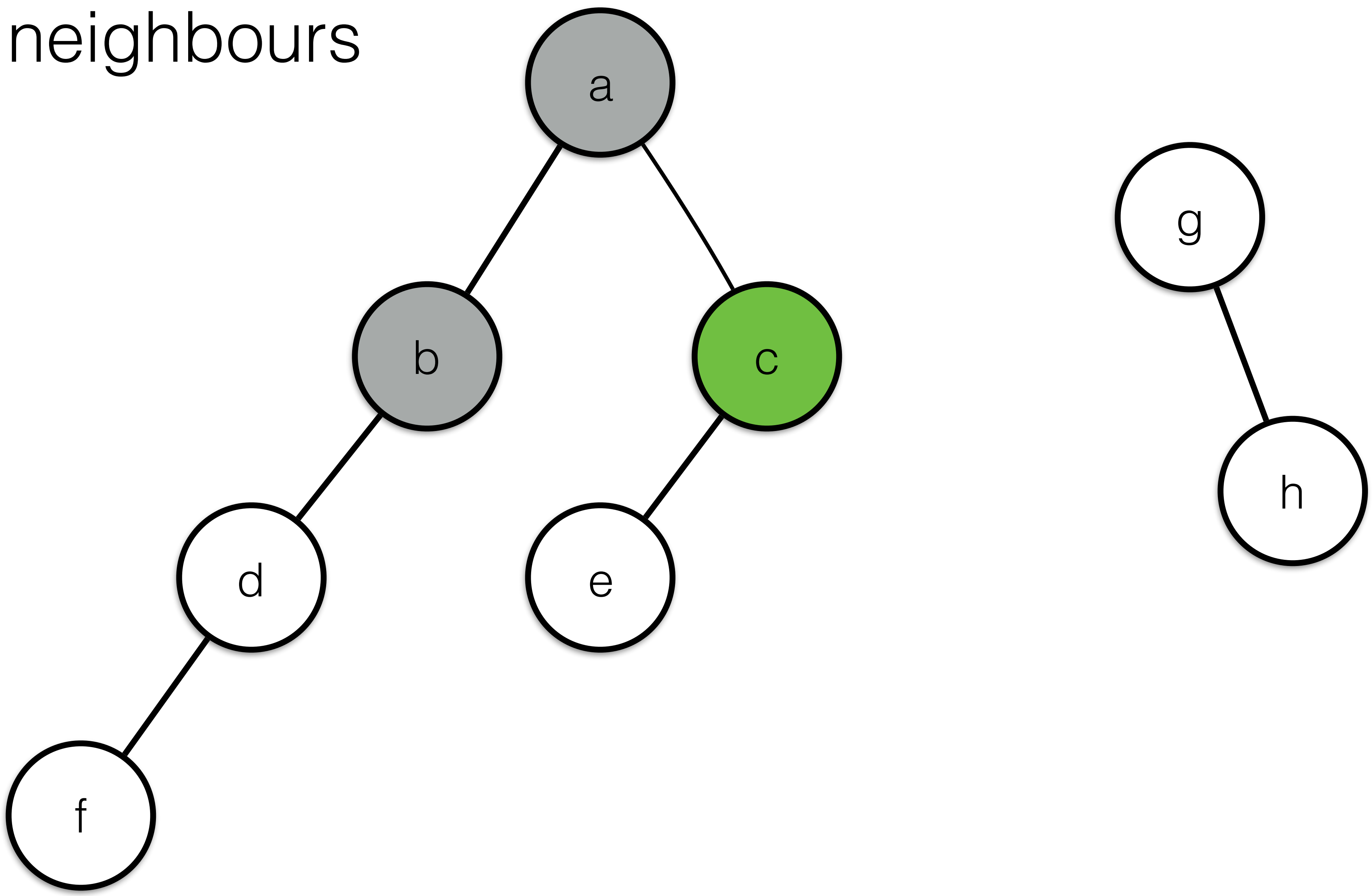
Visit all of its neighbours



Breadth-First Traversal

Nodes visited in this order: a b c

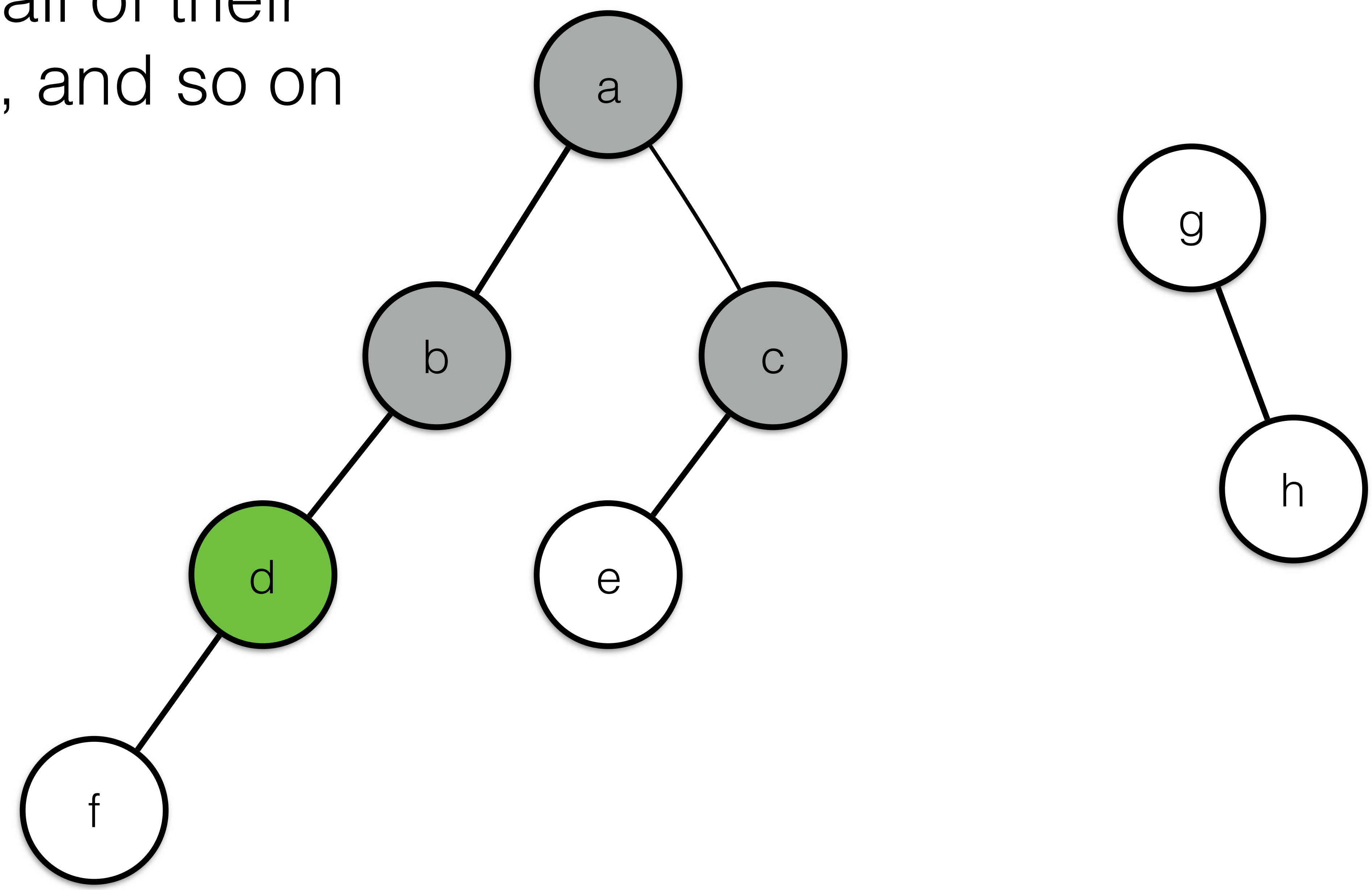
Visit all of its neighbours



Breadth-First Traversal

Nodes visited in this order: a b c d

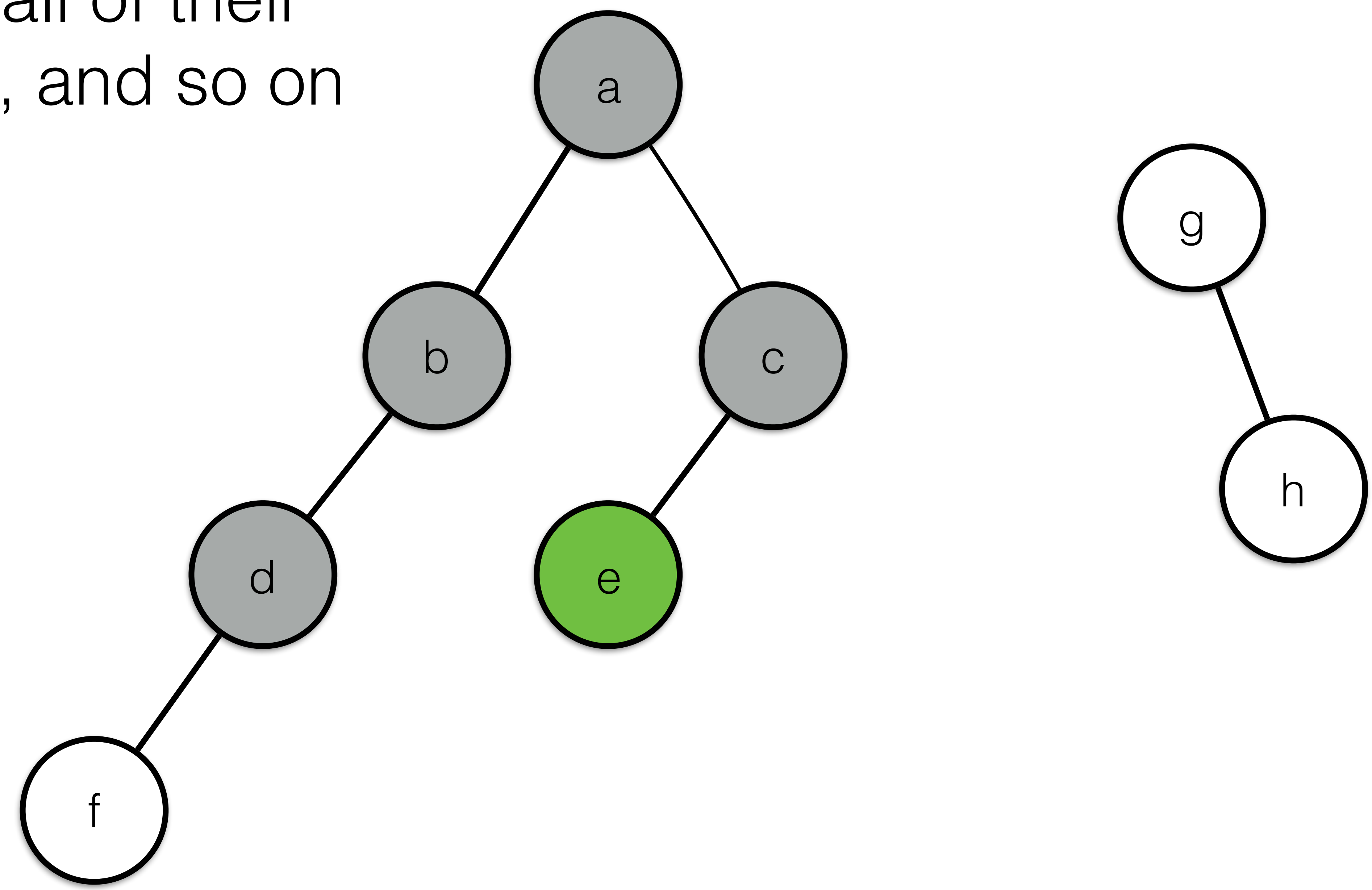
Then visit all of their neighbours, and so on



Breadth-First Traversal

Nodes visited in this order: a b c d e

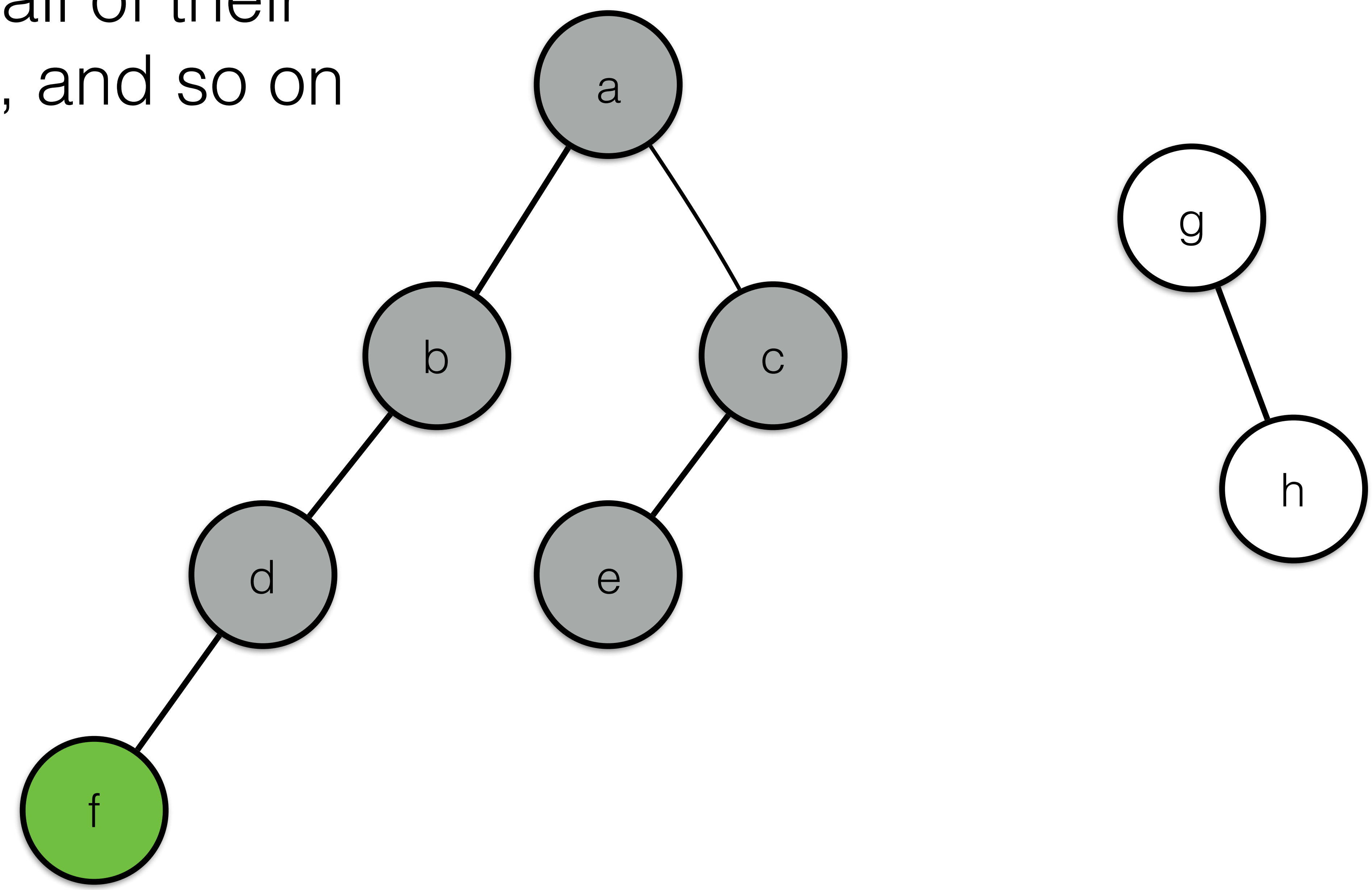
Then visit all of their neighbours, and so on



Breadth-First Traversal

Nodes visited in this order: a b c d e f

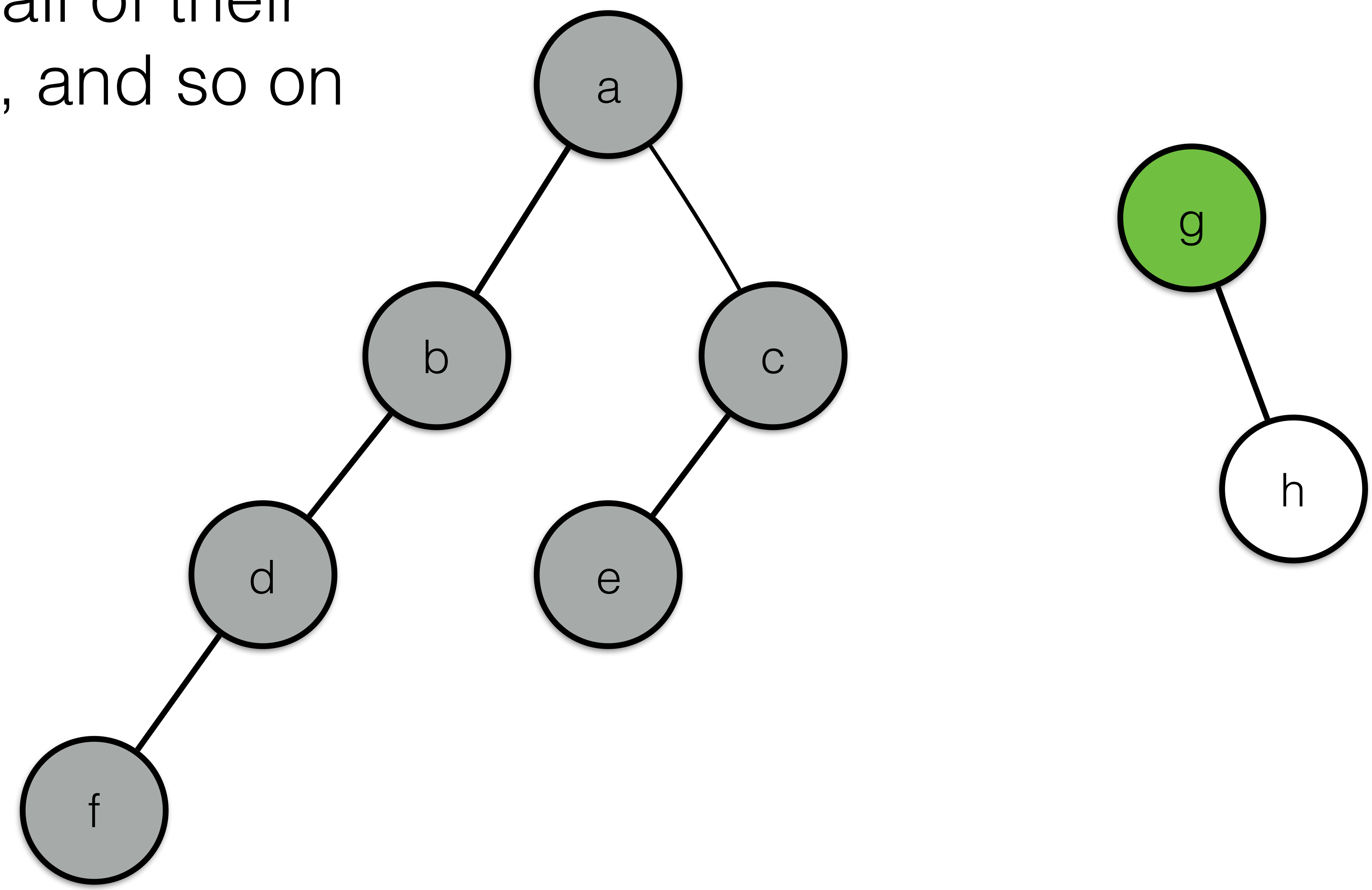
Then visit all of their neighbours, and so on



Breadth-First Traversal

Nodes visited in this order: a b c d e f g

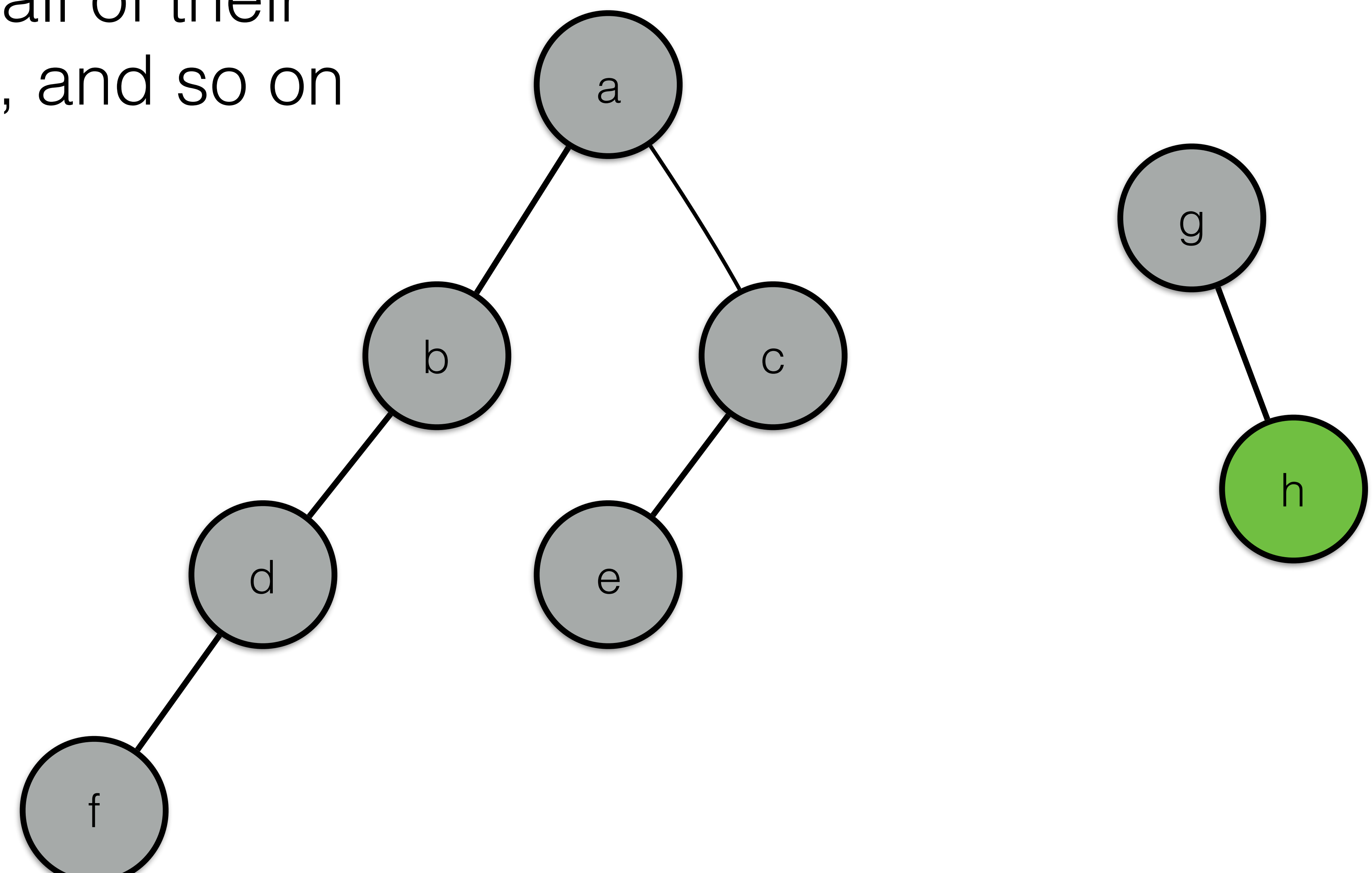
Then visit all of their neighbours, and so on



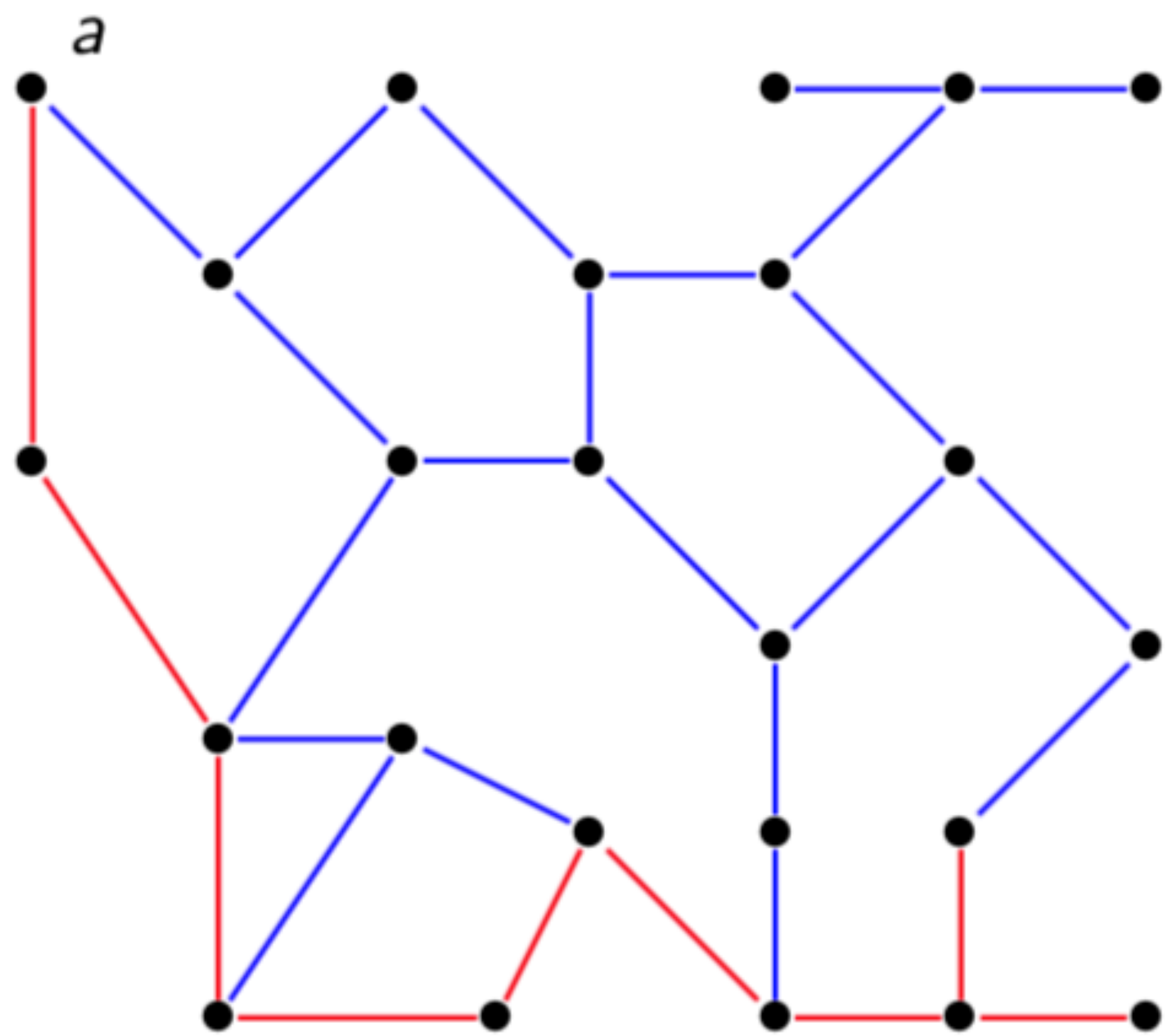
Breadth-First Traversal

Nodes visited in this order: a b c d e f g h

Then visit all of their neighbours, and so on

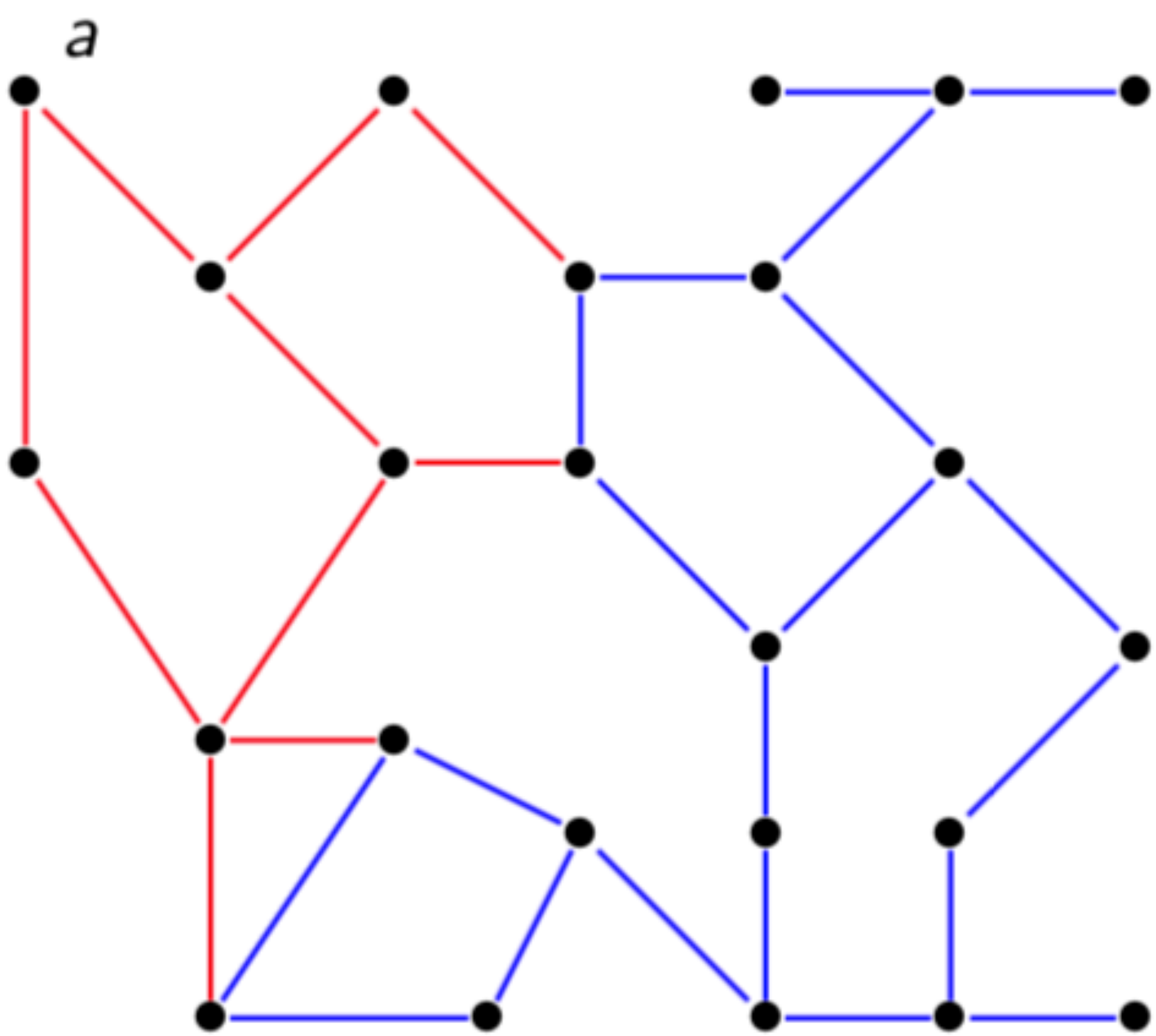


Depth-First vs Breadth-First Search



Typical Depth-First Search

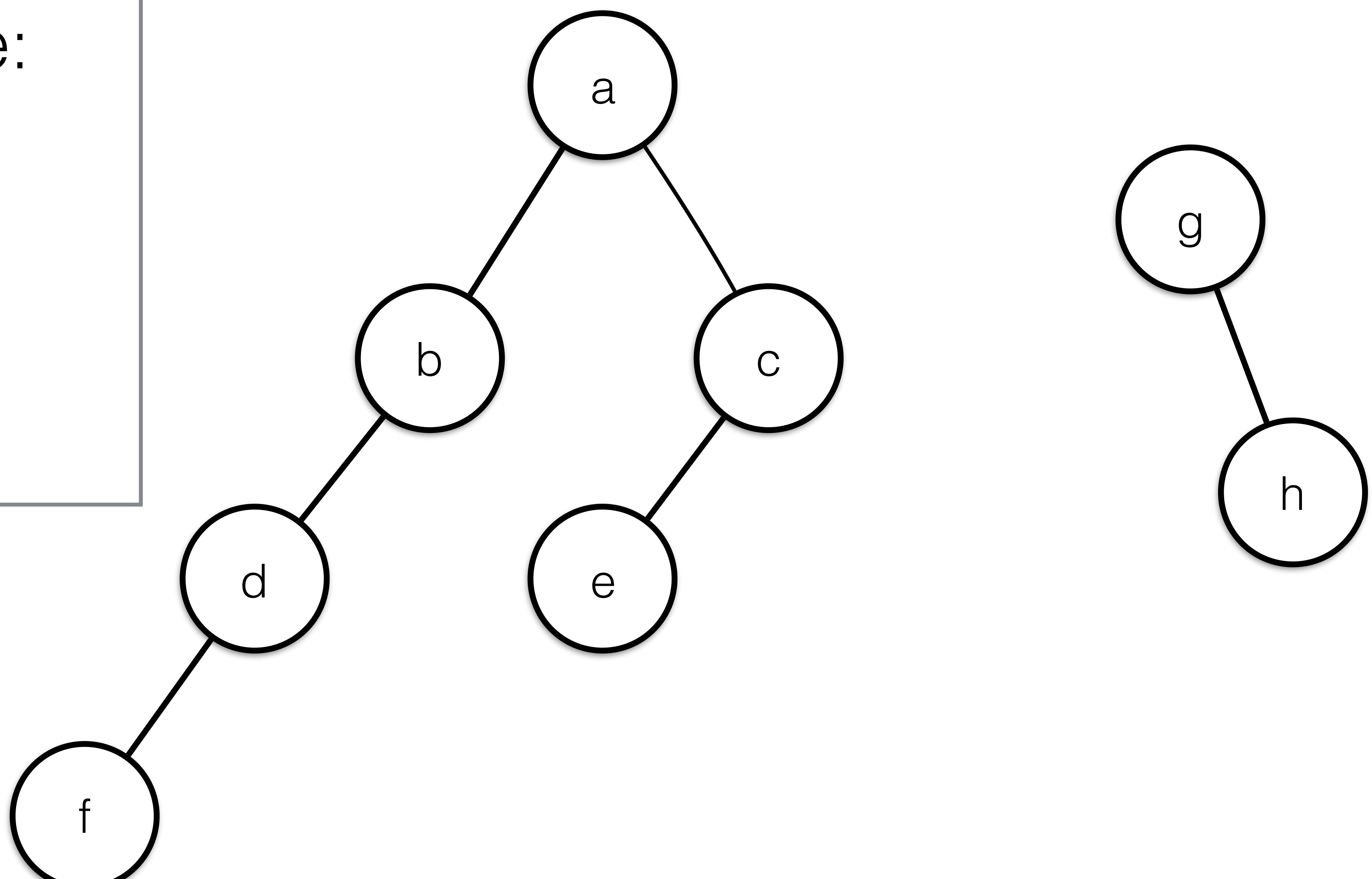
Depth-First vs Breadth-First Search



Typical Breadth-First Search

Breadth-First Search: Queue Discipline

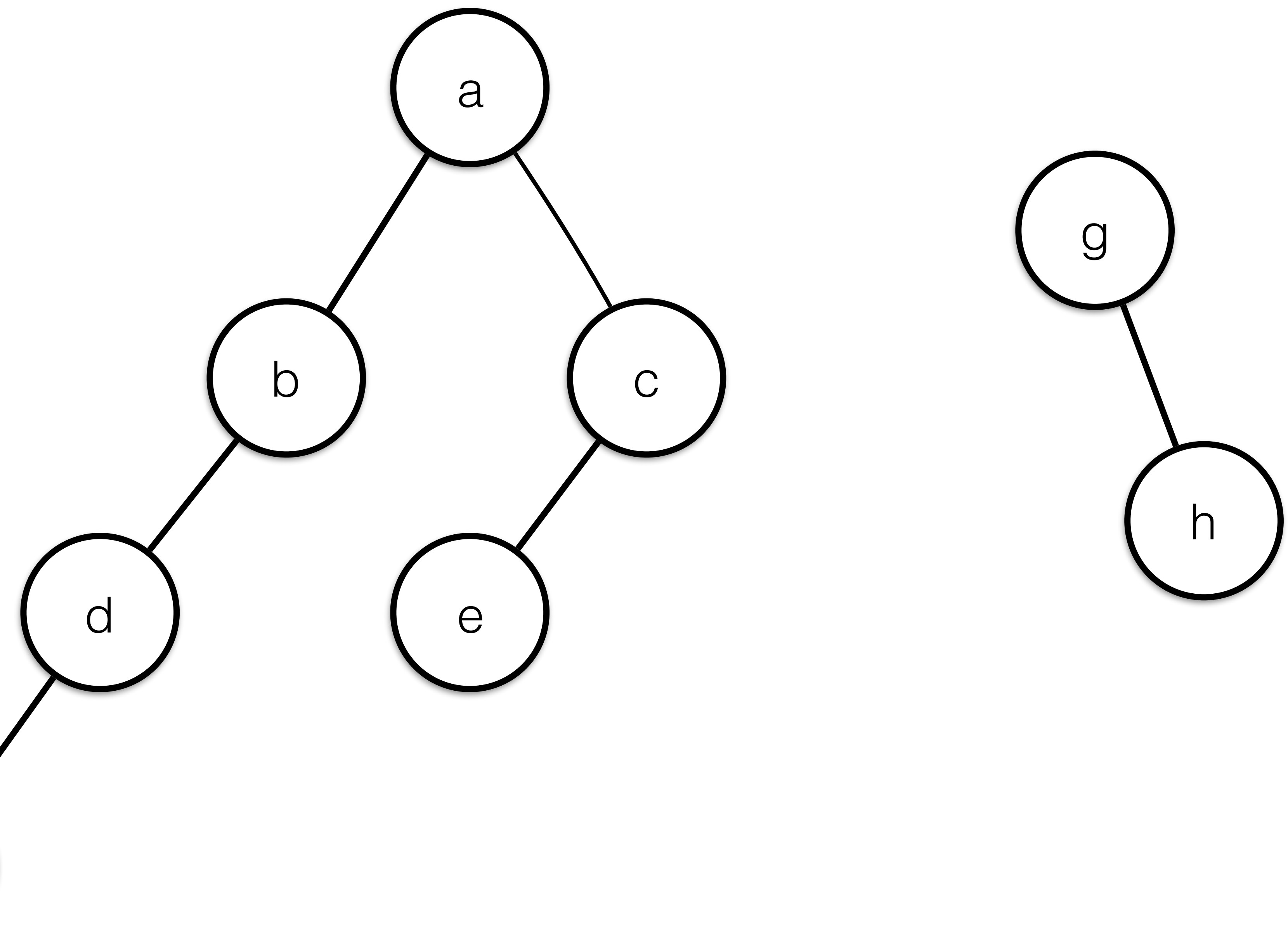
Traversal Queue:



Breadth-First Search: Queue Discipline

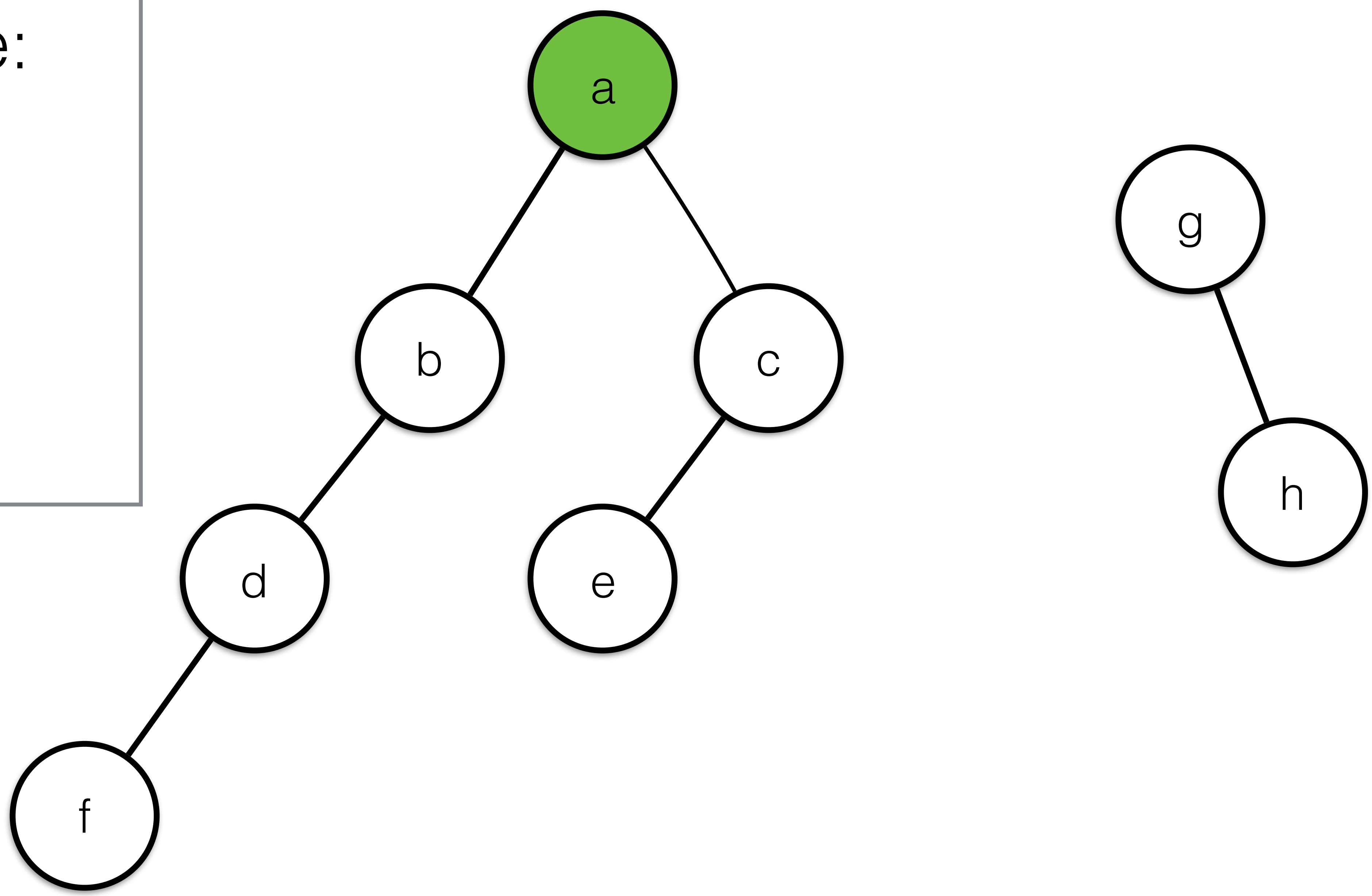
Traversal Queue:

a



Breadth-First Search: Queue Discipline

Traversal Queue:

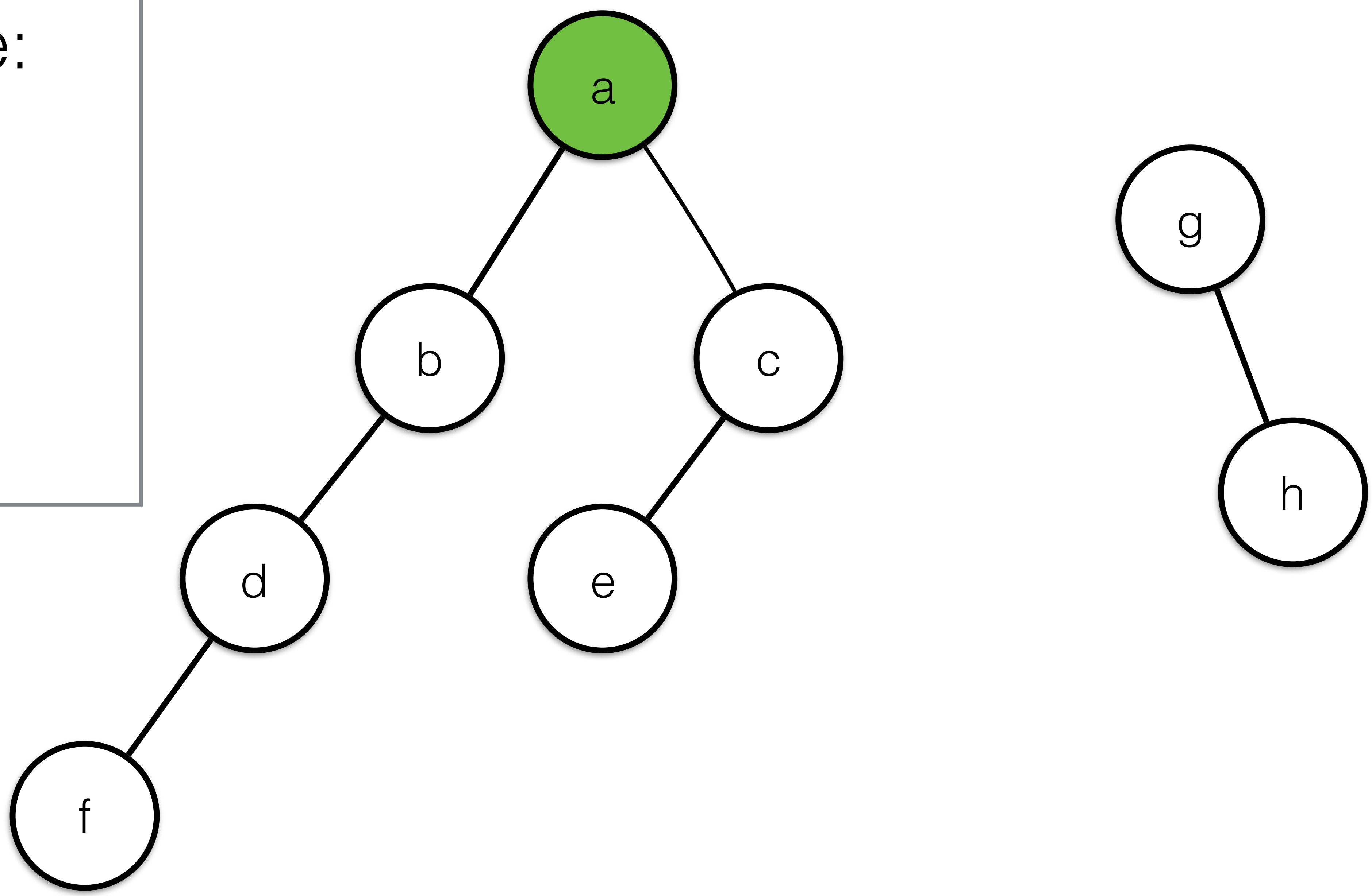


Breadth-First Search: Queue Discipline

THE UNIVERSITY OF MELBOURNE

Traversal Queue:

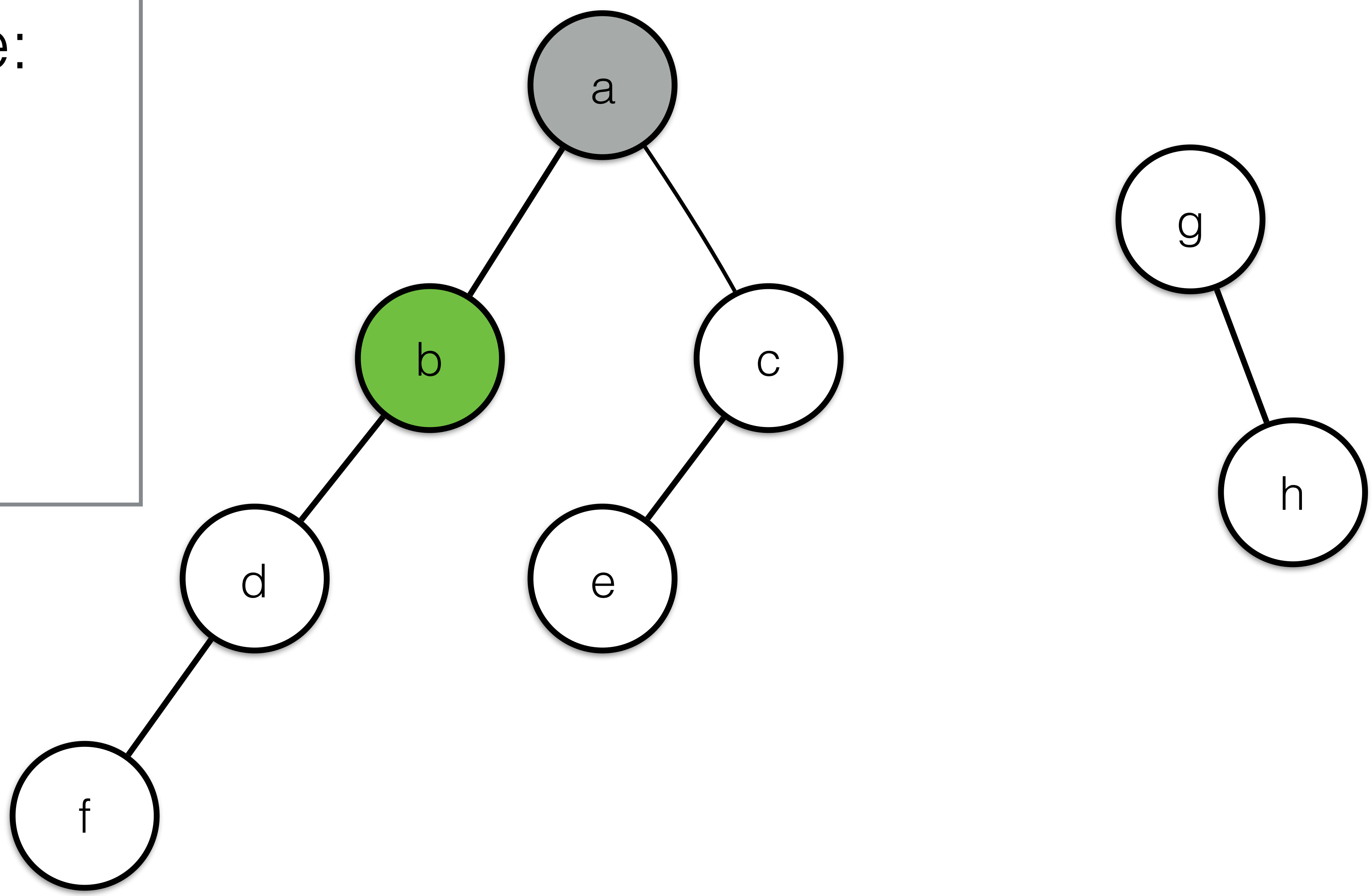
b
c



Breadth-First Search: Queue Discipline

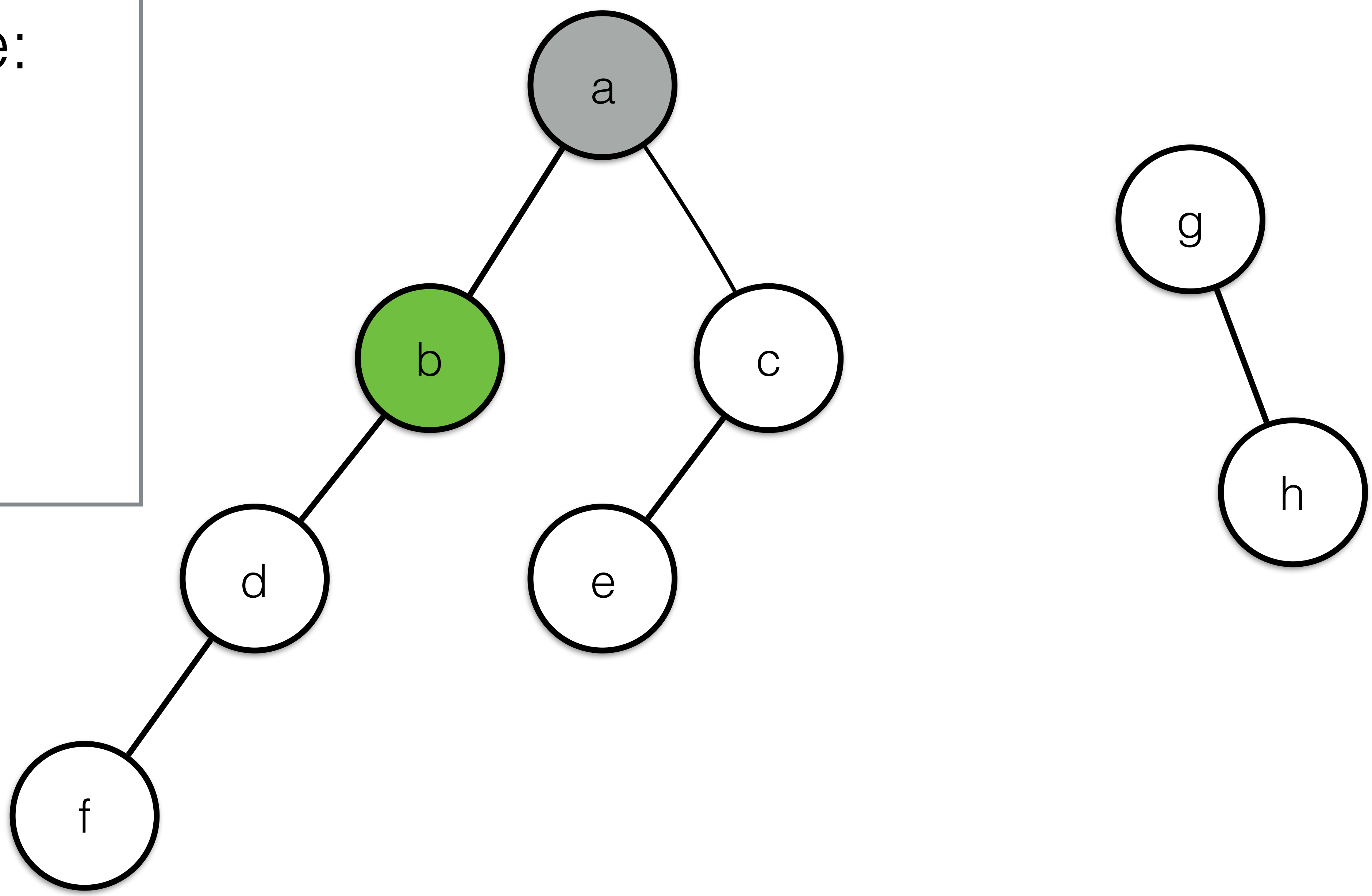
Traversal Queue:

c



Breadth-First Search: Queue Discipline

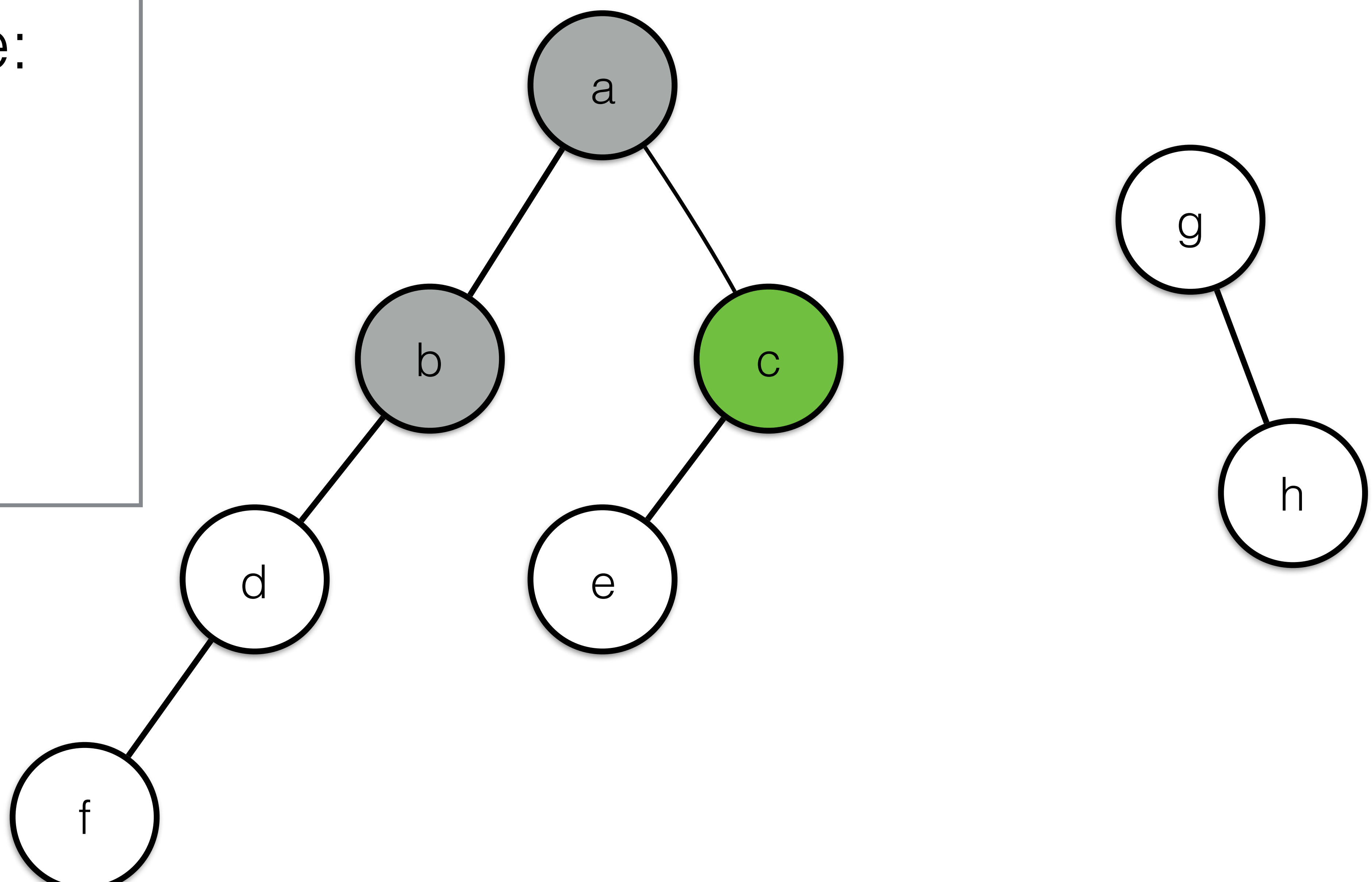
Traversal
Queue:
c
d



Breadth-First Search: Queue Discipline

Traversal Queue:

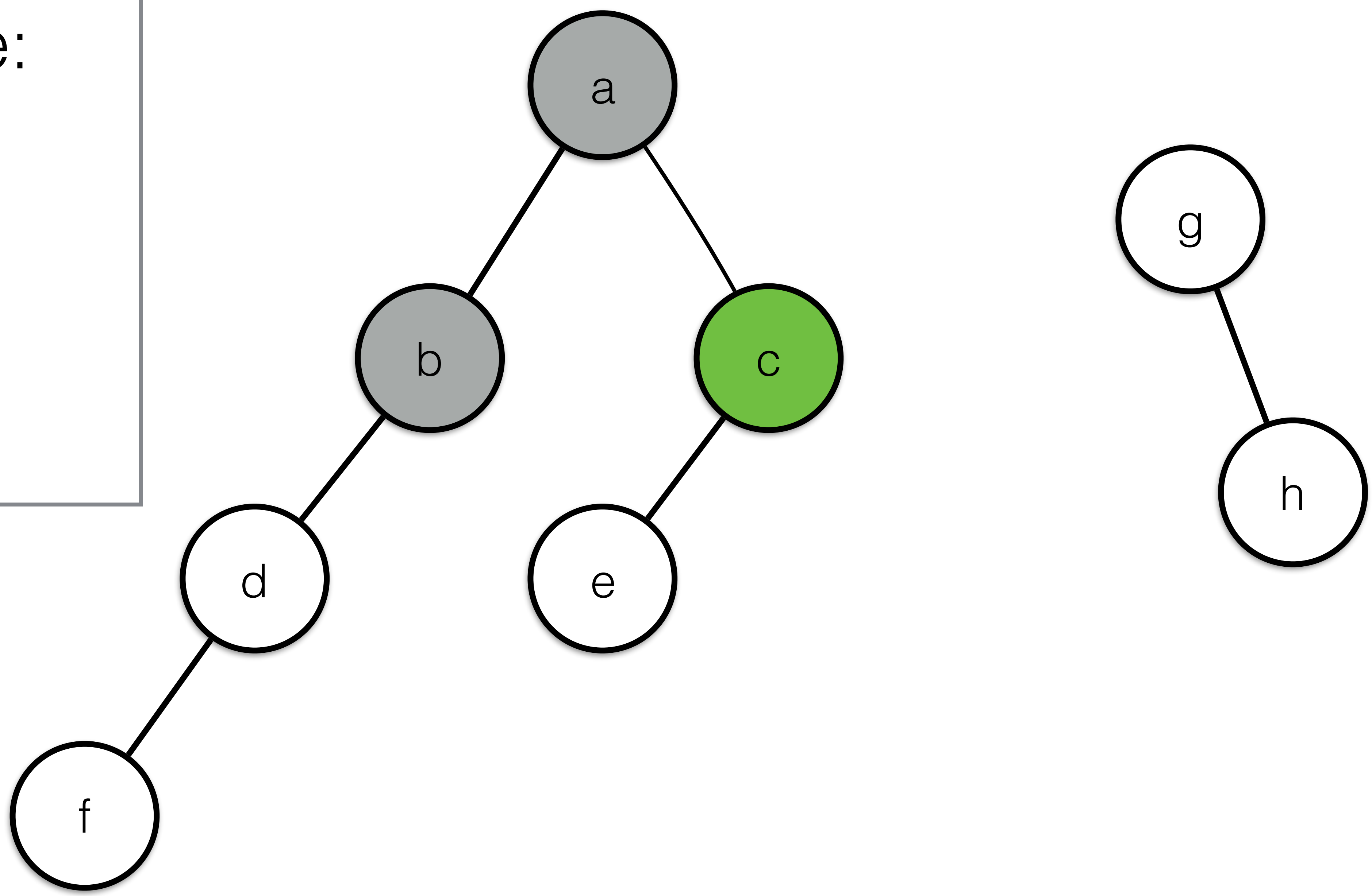
d



Breadth-First Search: Queue Discipline

Traversal Queue:

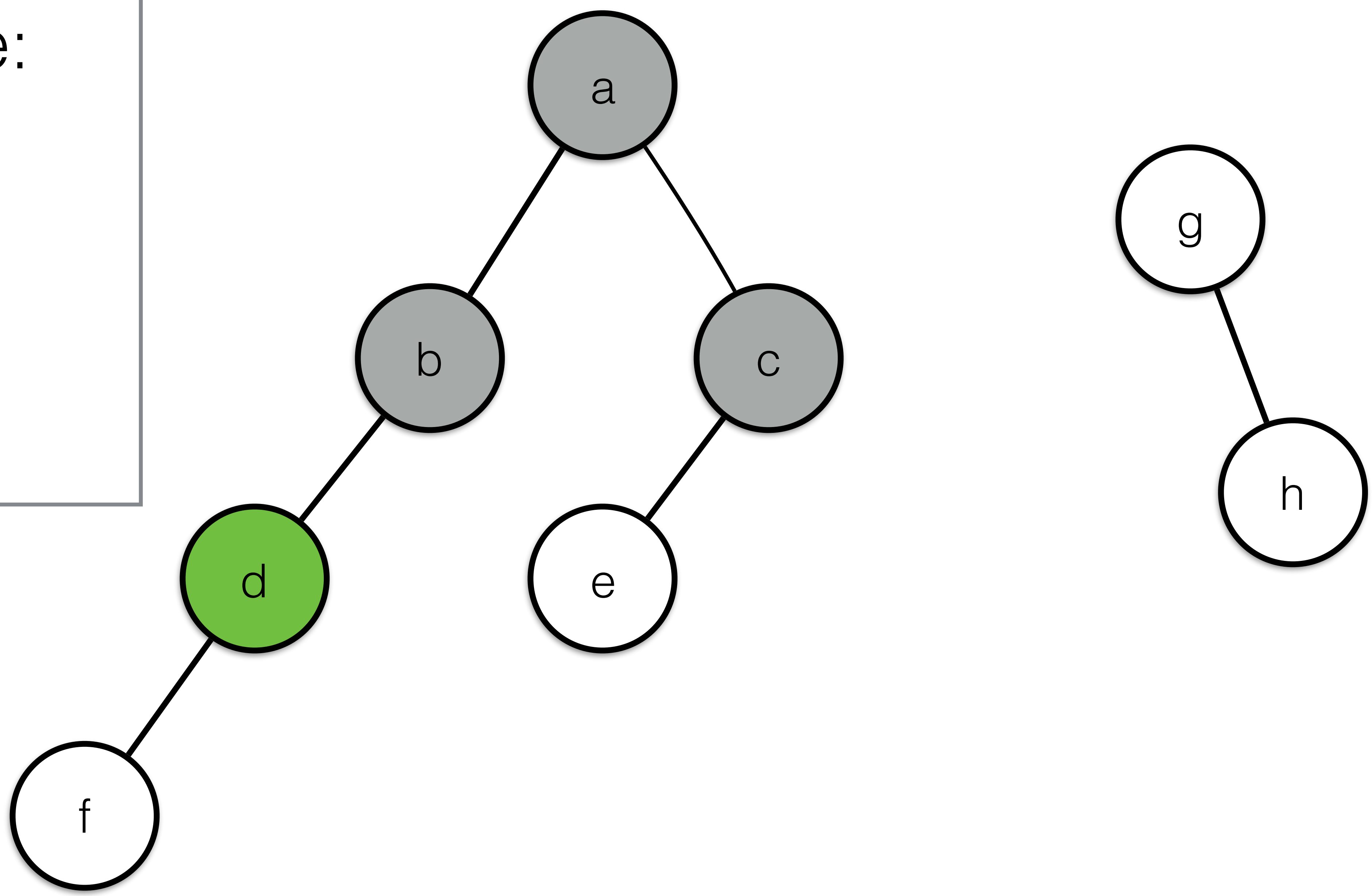
d
e



Breadth-First Search: Queue Discipline

Traversal Queue:

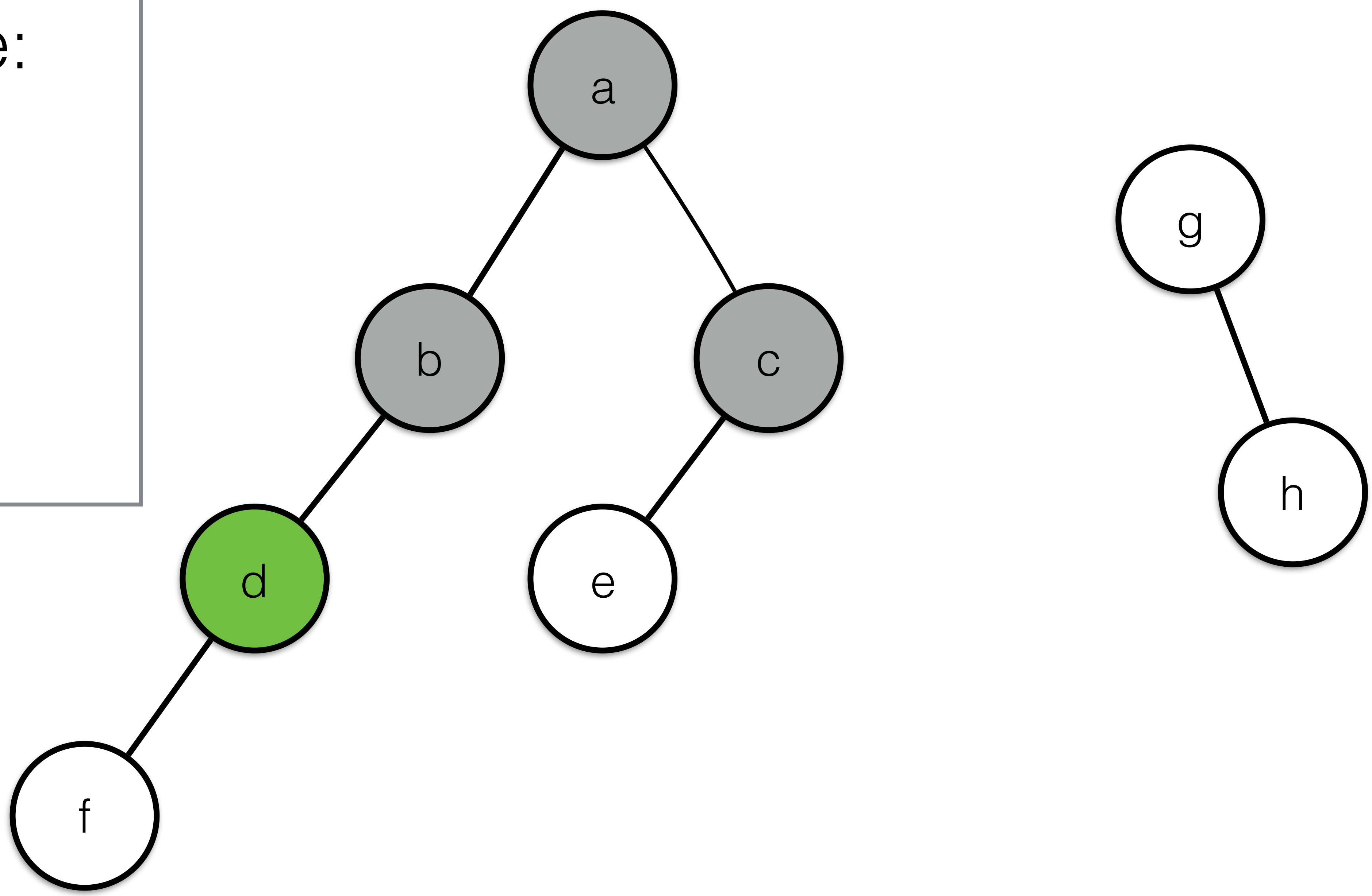
e



Breadth-First Search: Queue Discipline

Traversal Queue:

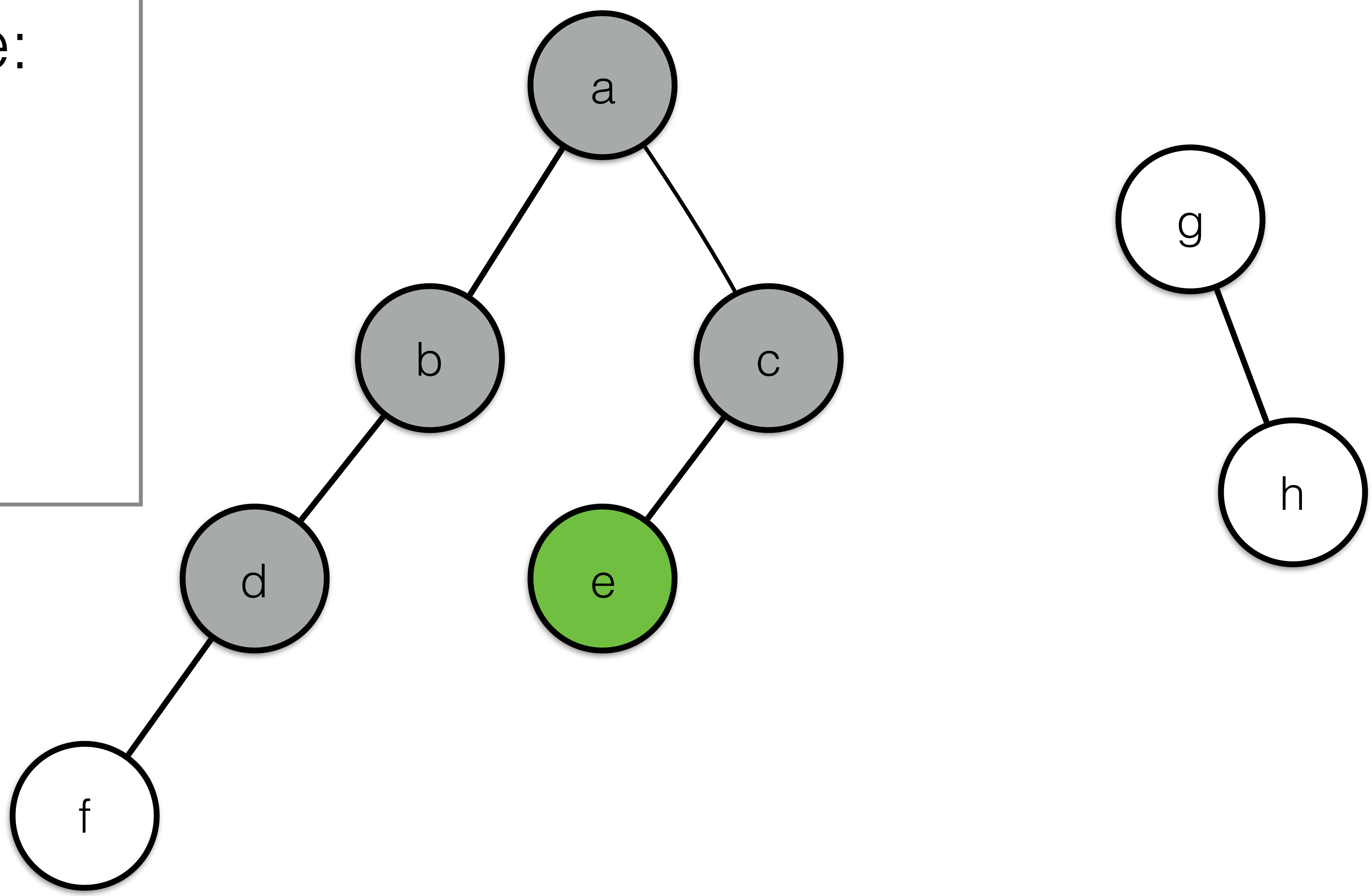
e
f



Breadth-First Search: Queue Discipline

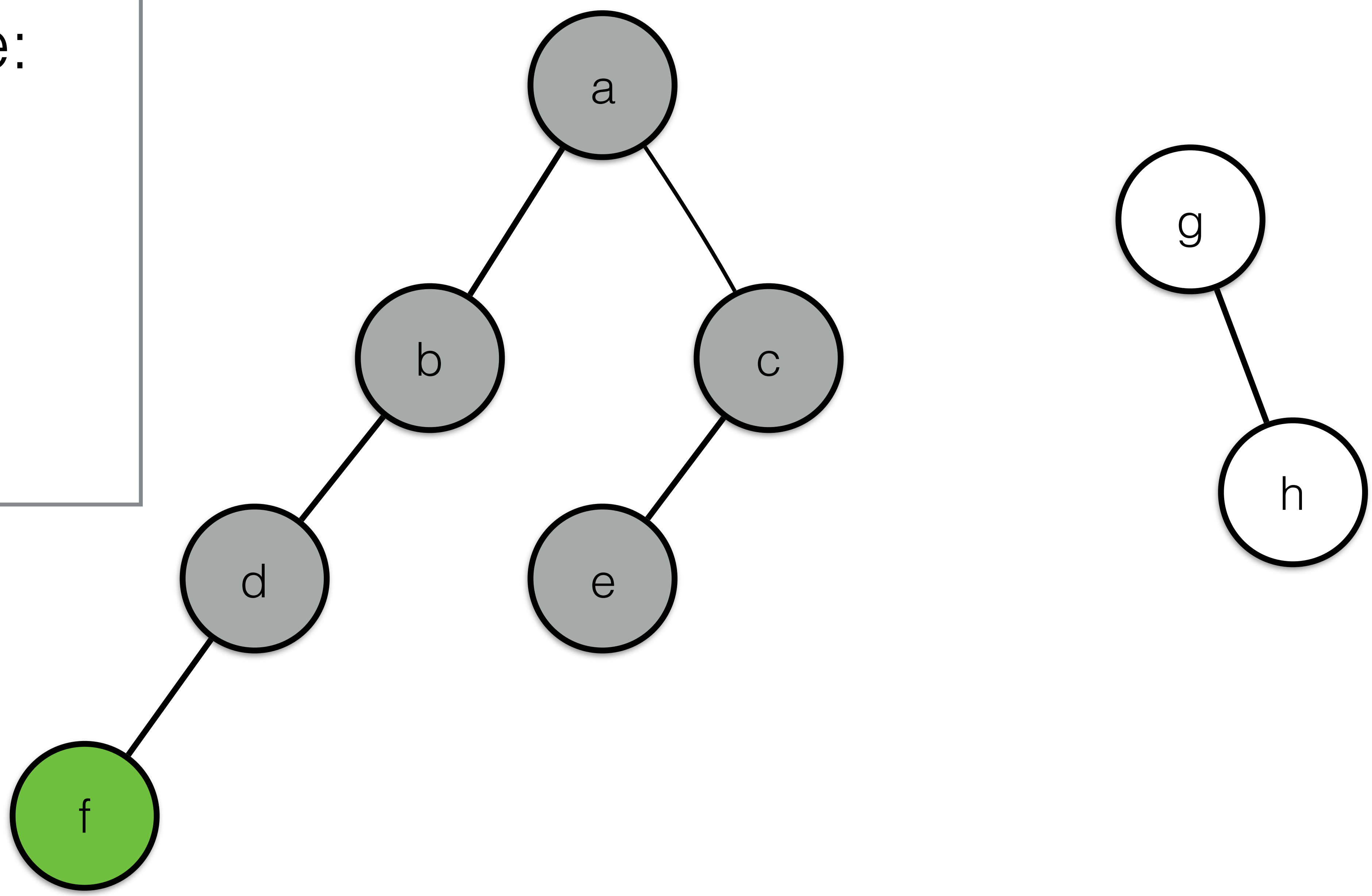
Traversal Queue:

f



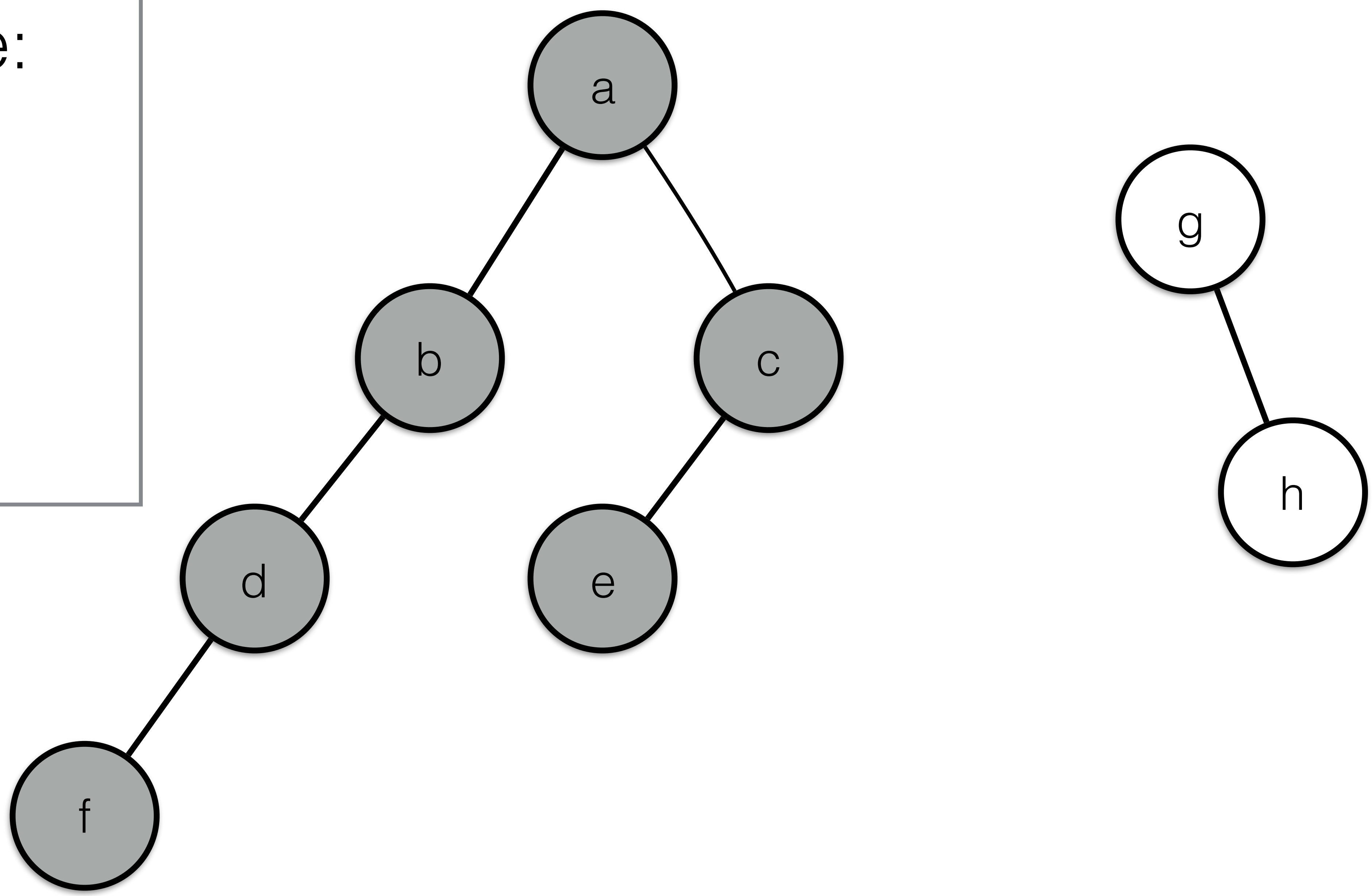
Breadth-First Search: Queue Discipline

Traversal Queue:



Breadth-First Search: Queue Discipline

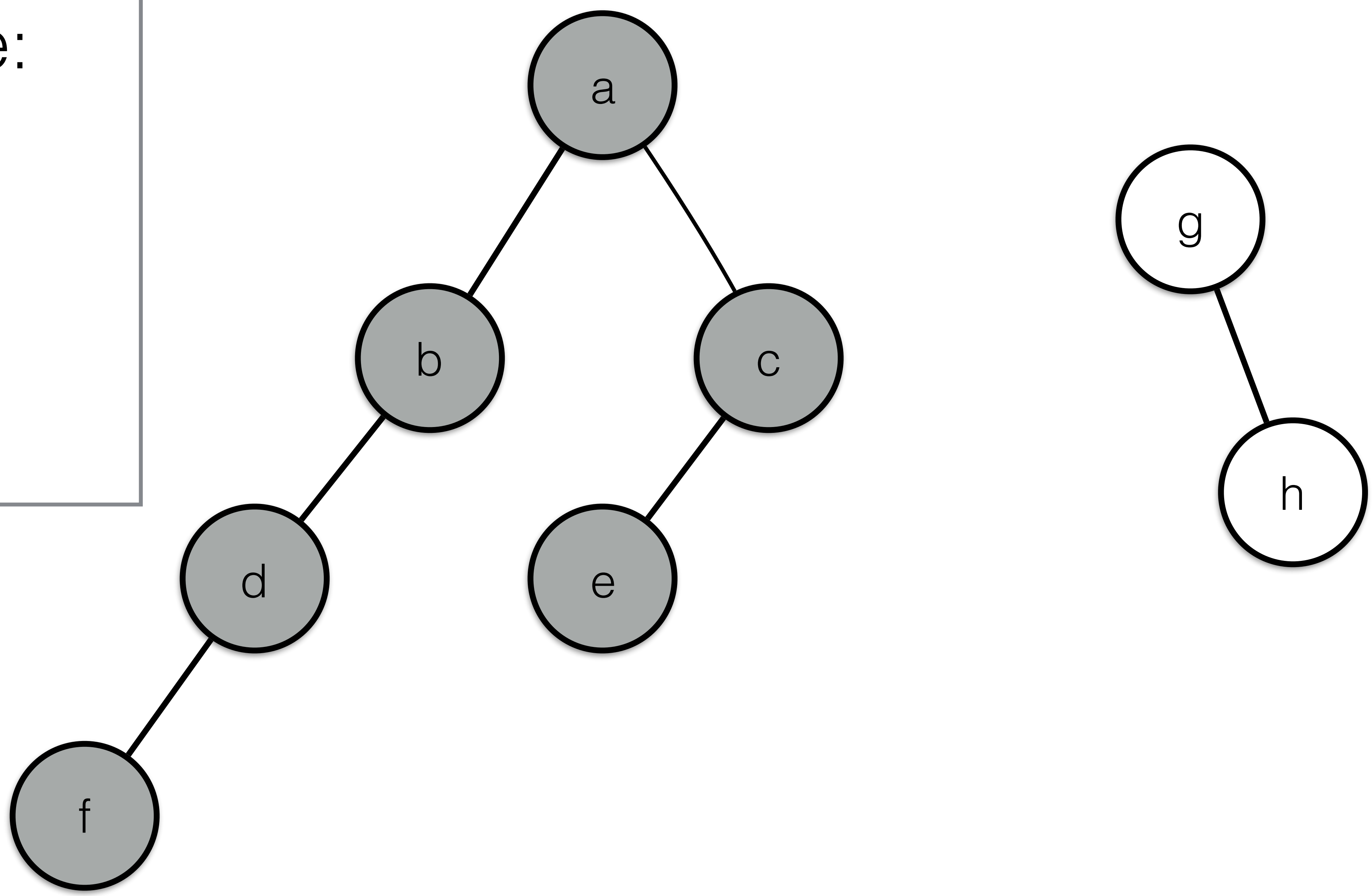
Traversal Queue:



Breadth-First Search: Queue Discipline

Traversal Queue:

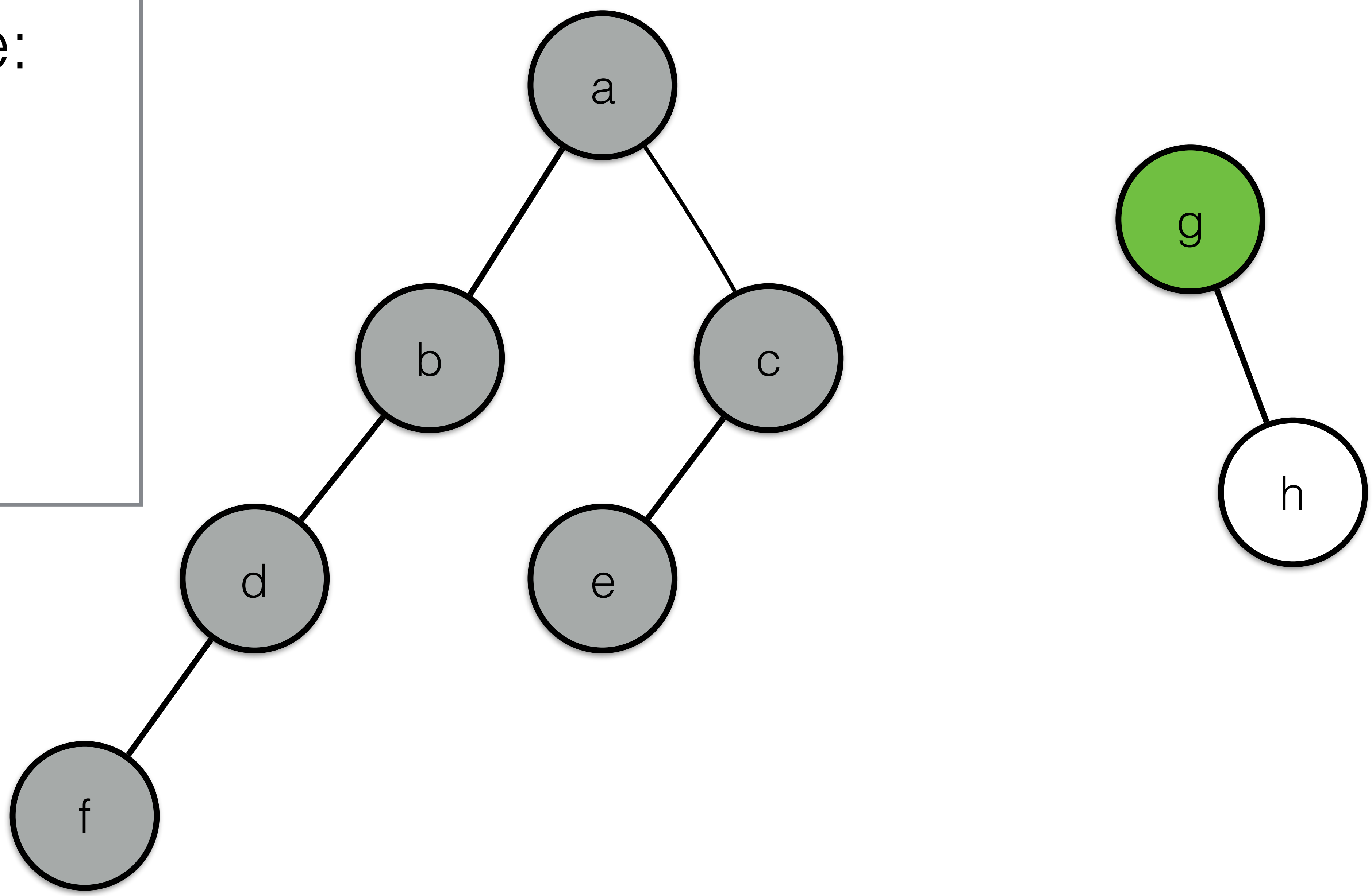
g



Breadth-First Search: Queue Discipline

THE UNIVERSITY OF MELBOURNE

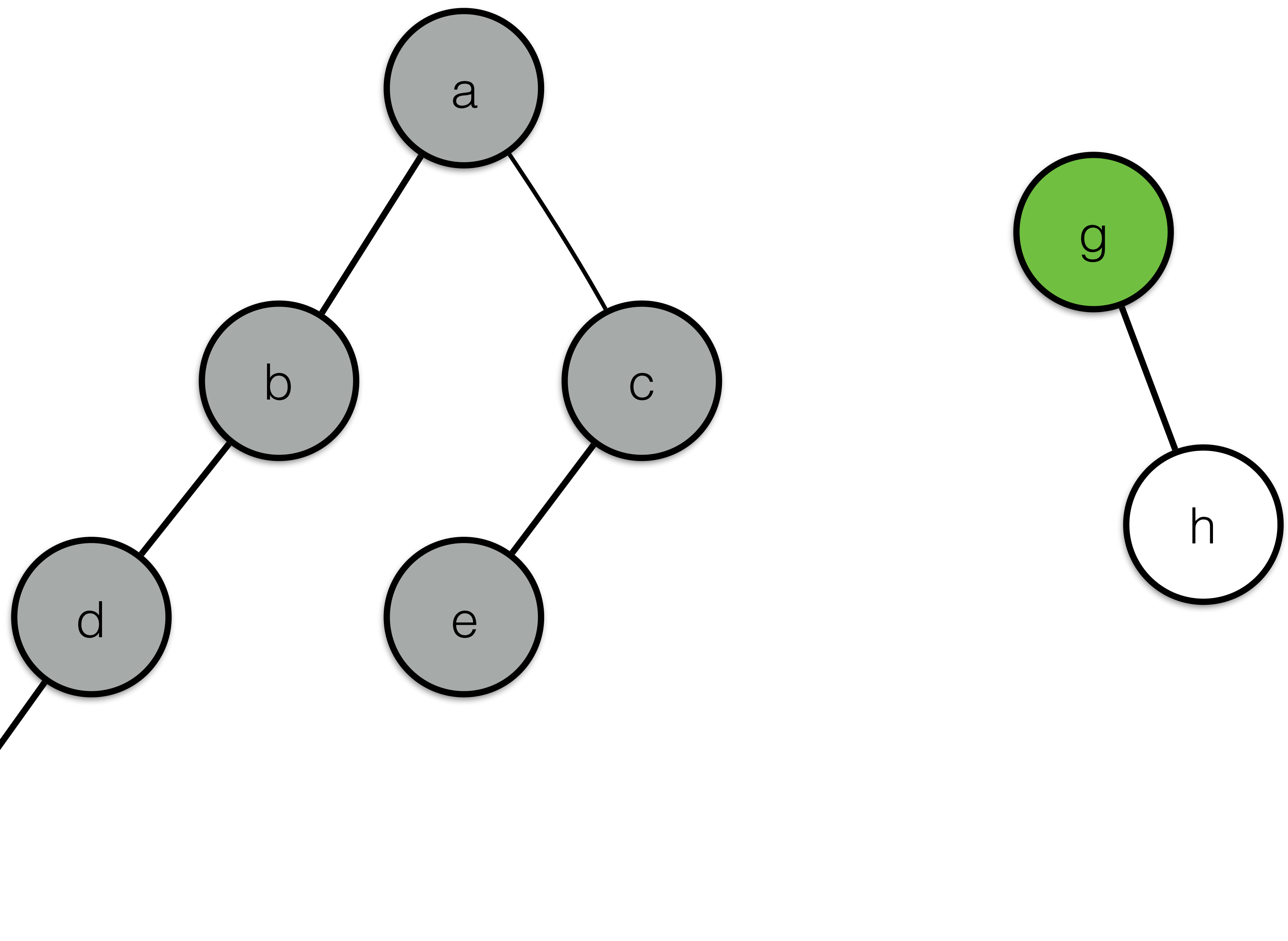
Traversal Queue:



Breadth-First Search: Queue Discipline

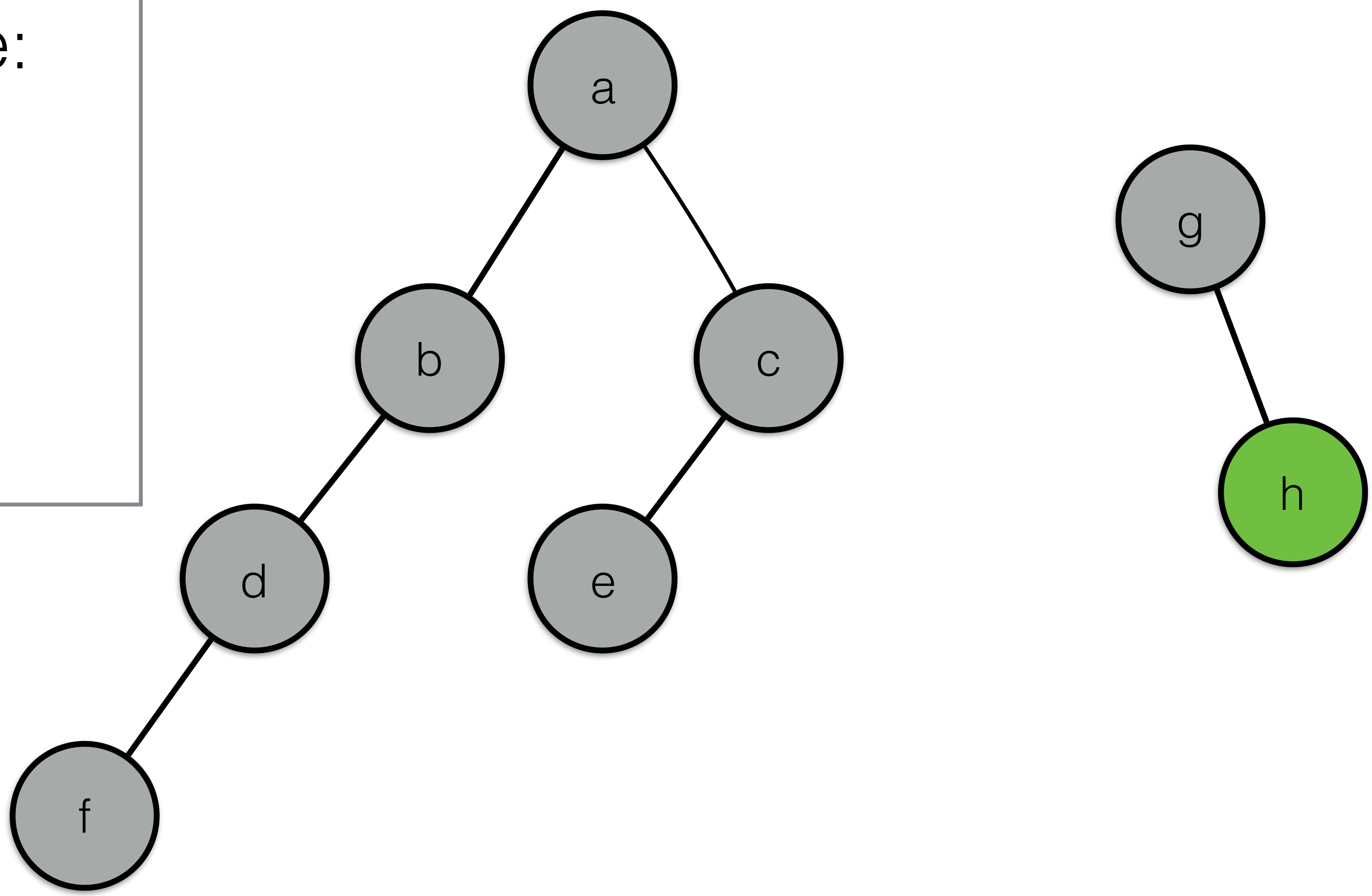
Traversal Queue:

h



Breadth-First Search: Queue Discipline

Traversal Queue:



Breadth-First Search Algorithm

THE UNIVERSITY OF MELBOURNE

function $\mathrm{BFS}(\langle V, E\rangle)$
    mark each node in V with 0
    $count \leftarrow 0$, $init(queue)$ ▷ create an empty queue
    for each v in V **do**
        if v is marked with 0 **then**
            $count \leftarrow count + 1$
            mark v with $count$
            $inject(queue, v)$ ▷ queue containing just v
            while $queue$ is non-empty **do**
                $u \leftarrow eject(queue)$ ▷ dequeues u
                for each edge (u, w) **do** ▷ w is u's neighbour
                    if w is marked with 0 **then**
                        $count \leftarrow count + 1$
                        mark w with $count$
                        $inject(queue, w)$ ▷ enqueues w

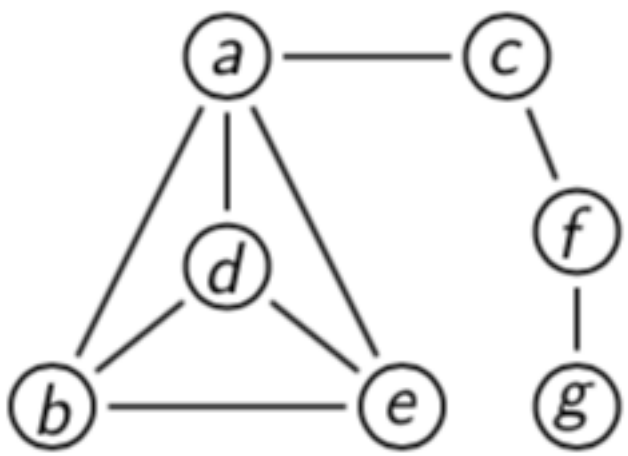
BFS Algorithm Notes

THE UNIVERSITY OF MELBOURNE

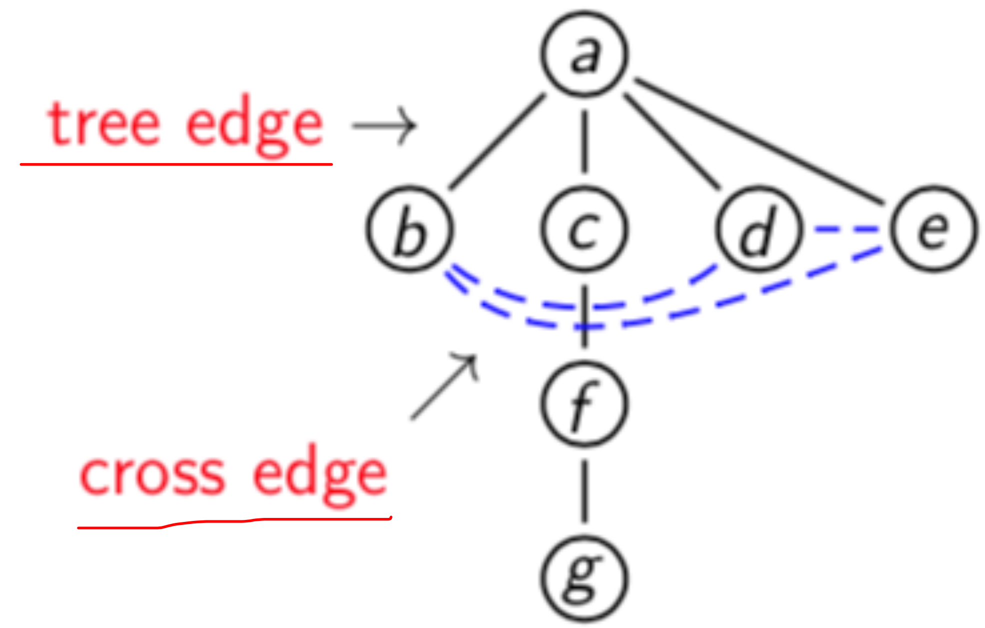
- BFS has the same complexity as DFS.
- Again, the same algorithm works for directed graphs as well.
- Certain problems are most easily solved by adapting BFS.
- For example, given a graph and two nodes, a and b in the graph, how would you find the length of the shortest path from a to b?

by the time that you reach node b in breadth-first search, it guarantees that you found the shortest path to get there

Breadth-First Search Forest



BFS **Tree** for this connected graph:



In general, we may get a BFS **Forest**

Topological Sorting

THE UNIVERSITY OF MELBOURNE

- We mentioned scheduling problems and their representation by directed graphs.

- Assume a directed edge from a to b means that task a must be completed before b can be started.

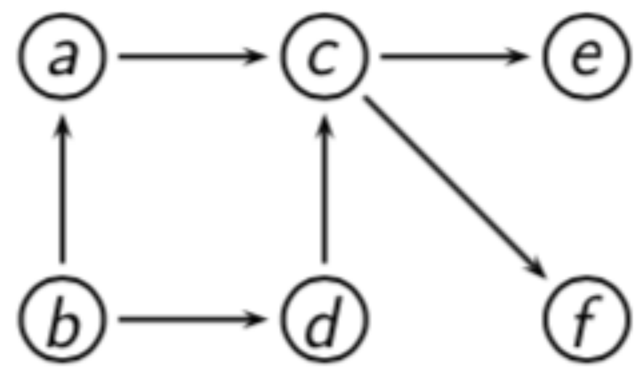
- The graph must be a dag; otherwise the problem cannot be solved.

- Assume the tasks are carried out by a single person, unable to multi-task.

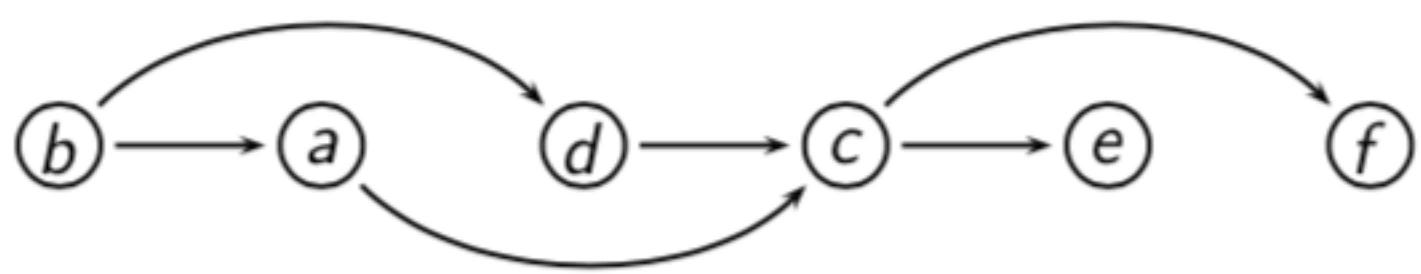
- Then we should try to **linearize** the graph, that is, order the nodes as a sequence $v_1, v_2, \ldots, v_n$ such that for each edge $(v_i, v_j) \in E$, we have v_i comes before v_j in the sequence (that is, v_i is scheduled to happen before v_j).

Topological Sorting Example

There are 4 ways to linearise the following graph



Here is one:

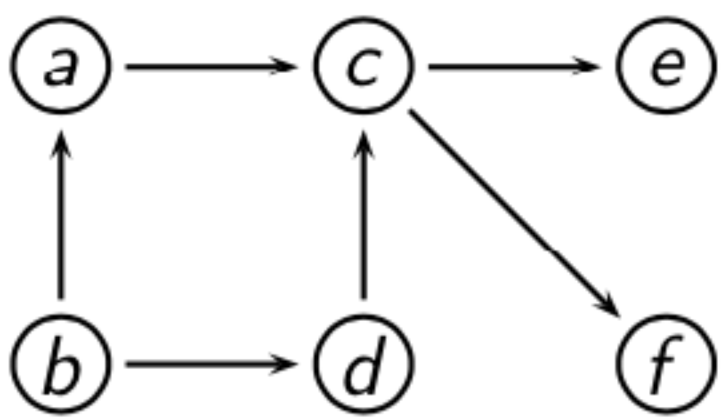


Topological Sorting Algorithm 1

- We can solve the top-sort problem with depth-first search:
  1. Perform DFS and note the order in which nodes are popped off the stack.
  2. List the nodes in the reverse of that order.
- This works because of the stack discipline.
- If (u,v) is an edge then it is possible (given some way of deciding ties) to arrive at a DFS stack with u sitting below v.
- Taking the "reverse popping order" ensures that u is listed before v.

Topological Sorting Example Again

Using the DFS method and resolving ties by using alphabetical order, the graph gives rise to the traversal stack shown on the right (the popping order shown in red):



$e_{3,1}$	$f_{4,2}$	
$c_{2,3}$		$d_{6,5}$
$a_{1,4}$		$b_{5,6}$

Taking the nodes in reverse popping order yields b, d, a, c, f , e.

Topological Sorting Algorithm 2

THE UNIVERSITY OF MELBOURNE

- An alternative method would be to repeatedly select a random **source** in the graph (that is, a node with no incoming edges), list it, and remove it from the graph (including removing its outgoing edges).

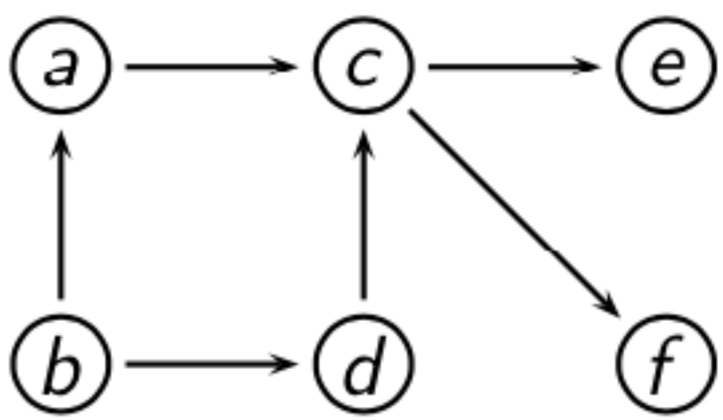
- This is a very natural approach, but it has the drawback that we repeatedly need to scan the graph for a source.

- However, it exemplifies the general principle of **decrease-and-conquer**.

Topological Sorting Example Again

THE UNIVERSITY OF MELBOURNE

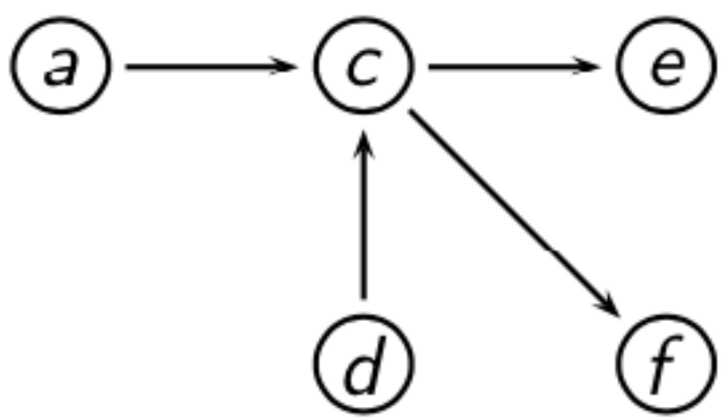
Using the source removal method (and resolving ties alphabetically):



Topological sorted order:

Topological Sorting Example Again

Using the source removal method (and resolving ties alphabetically):

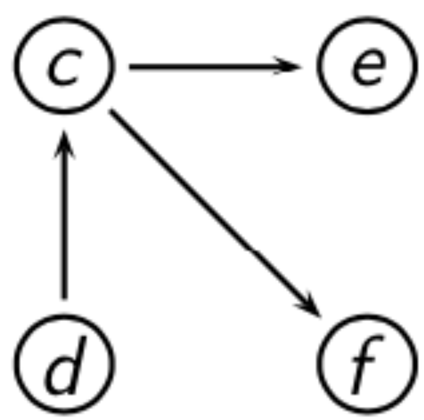


Topological sorted order:

b

Topological Sorting Example Again

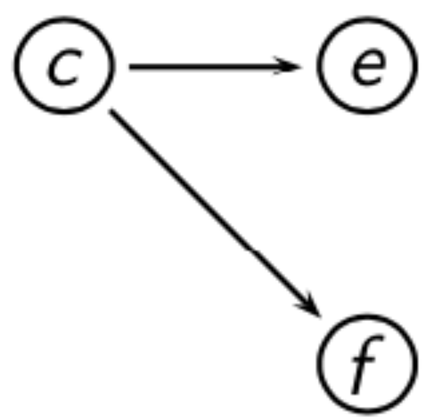
Using the source removal method (and resolving ties alphabetically):



Topological sorted order:
b, a

Topological Sorting Example Again

Using the source removal method (and resolving ties alphabetically):



Topological sorted order:
b, a, d

Topological Sorting Example Again

Using the source removal method (and resolving ties alphabetically):

Topological sorted order:
b, a, d, c

Topological Sorting Example Again

Using the source removal method (and resolving ties alphabetically):

Topological sorted order:
b, a, d, c, e

Topological Sorting Example Again

Using the source removal method (and resolving ties alphabetically):

Topological sorted order:
b, a, d, c, e, f

Next time

- So next we turn our attention to the very useful "decrease and conquer" principle
(Levitin Chapter 4).