CIS 351-Data Structure-BSF-DFS
April 14, 2020

Dr. Farzana Rahman

Syracuse University

Breadth First Search & Depth First Search

- **breadth first search:** Any search algorithm that considers neighbors of a *vertex* (node), that is, outgoing *edges* (links) of the vertex's predecessor in the search, before any outgoing edges of the vertex
- **depth first search:** Any search algorithm that considers outgoing *edges* (links o *children*) of a *vertex* (node) before any of the vertex's (node) *siblings*, that is, outgoing edges of the vertex's predecessor in the search.

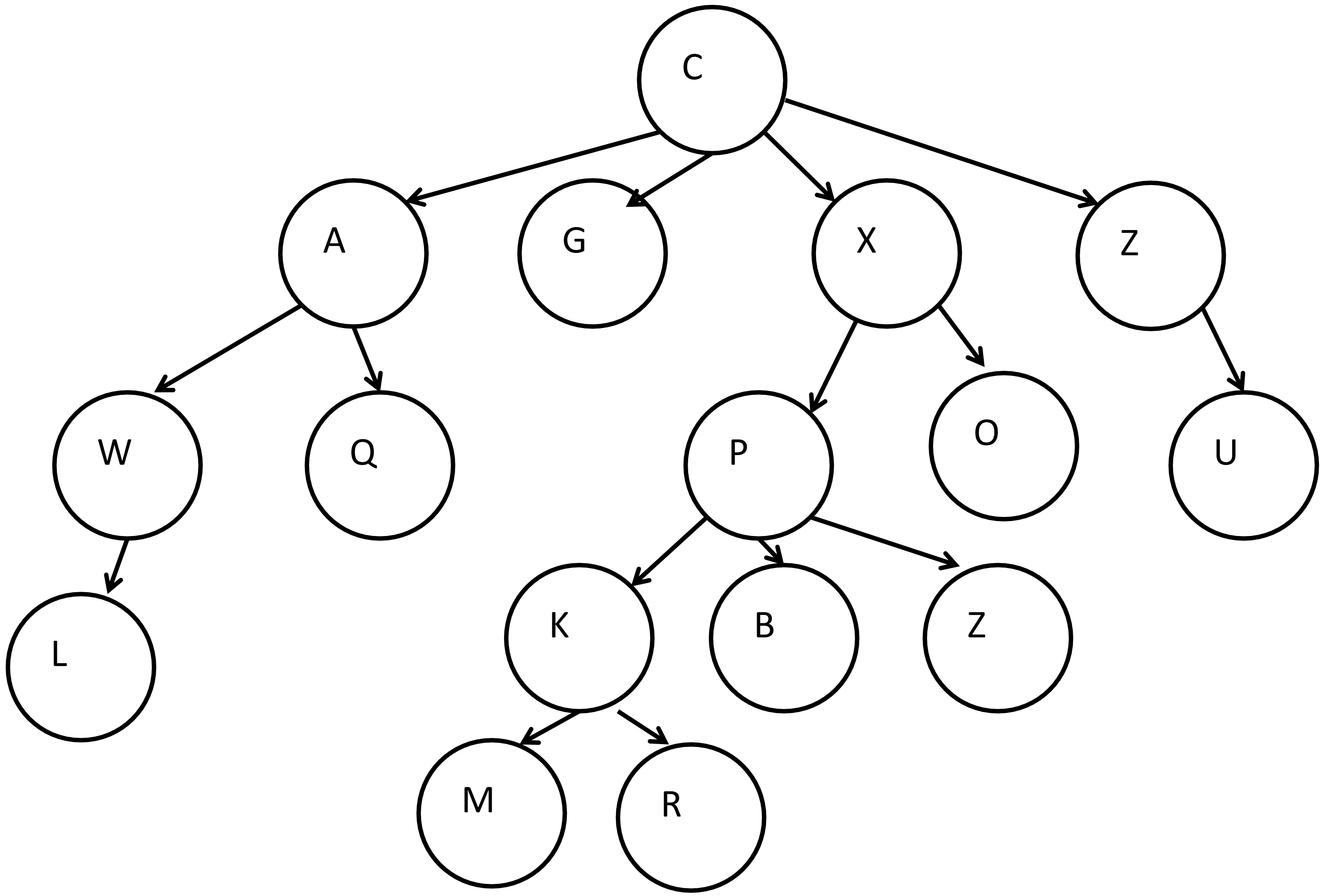
BFS-DFS why ?

- In general, BFS is better for problems related to finding the shortest paths or somewhat related problems.
- While DFS on the other end helps more in connectivity problems and also in finding cycles in graph

Breadth First

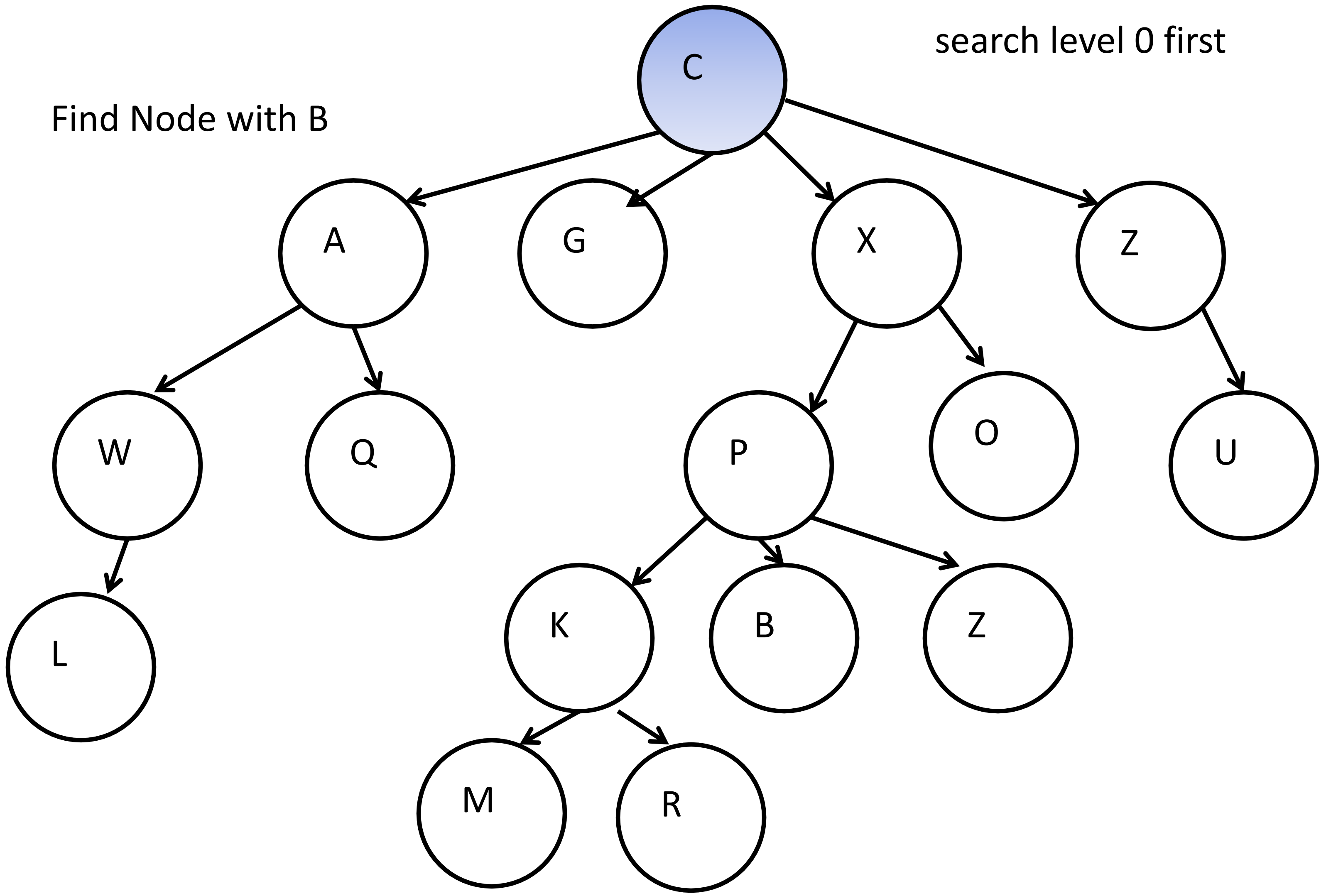
- A level order traversal of a tree could be used as a breadth first search
- Search all nodes in a level before going down to the next level

Breadth First Search of Tree



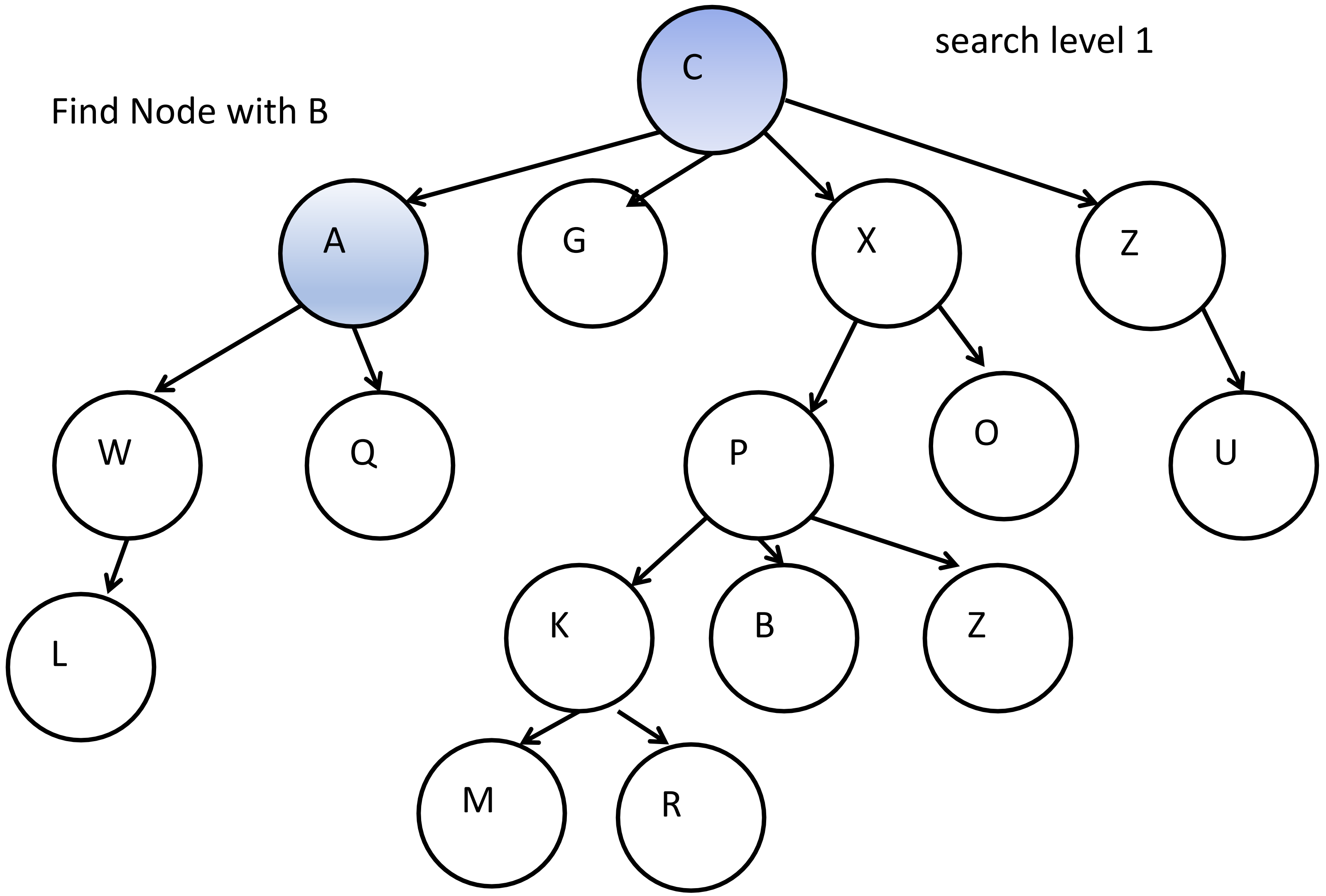
Find Node with B

search level 0 first



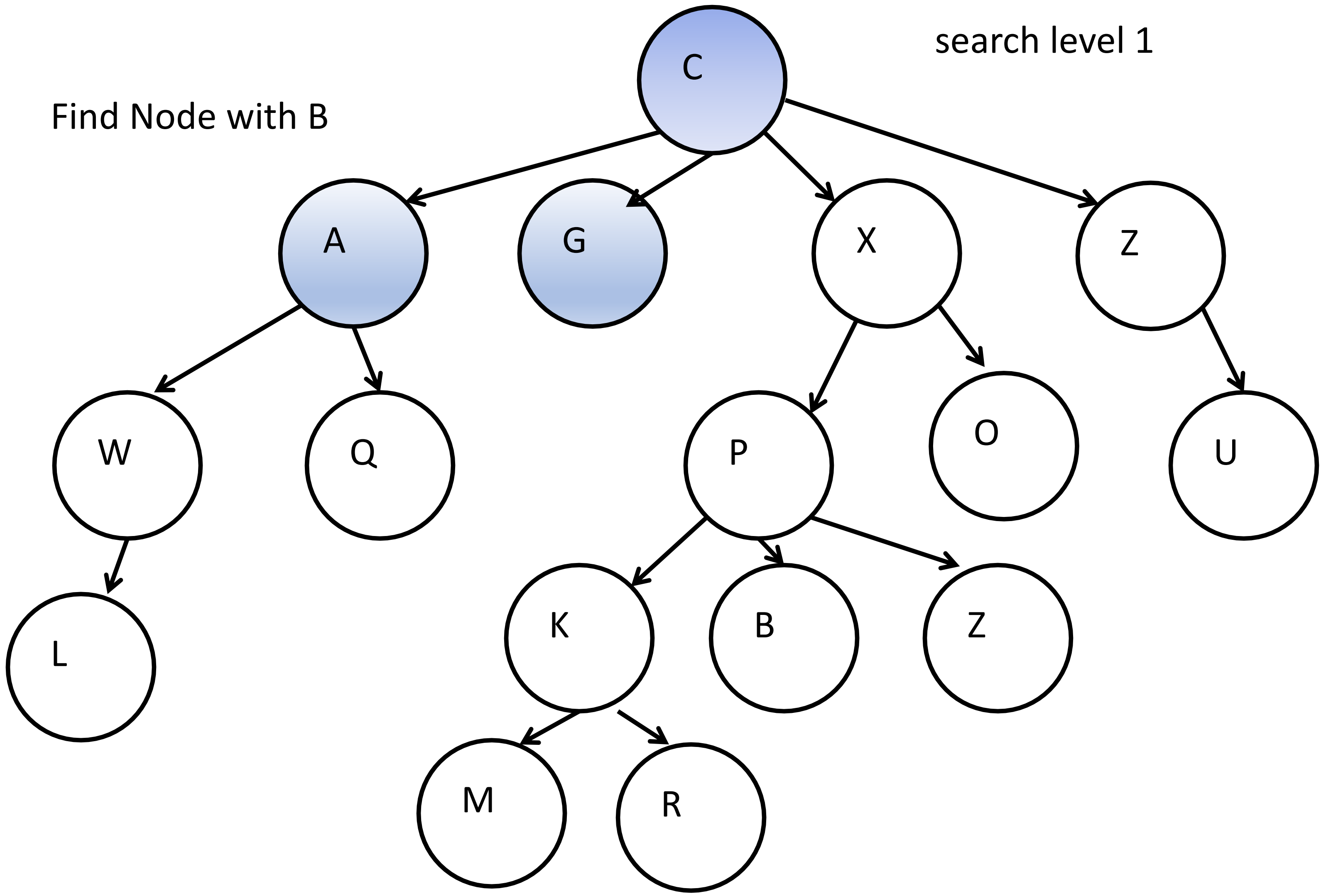
Find Node with B

search level 1



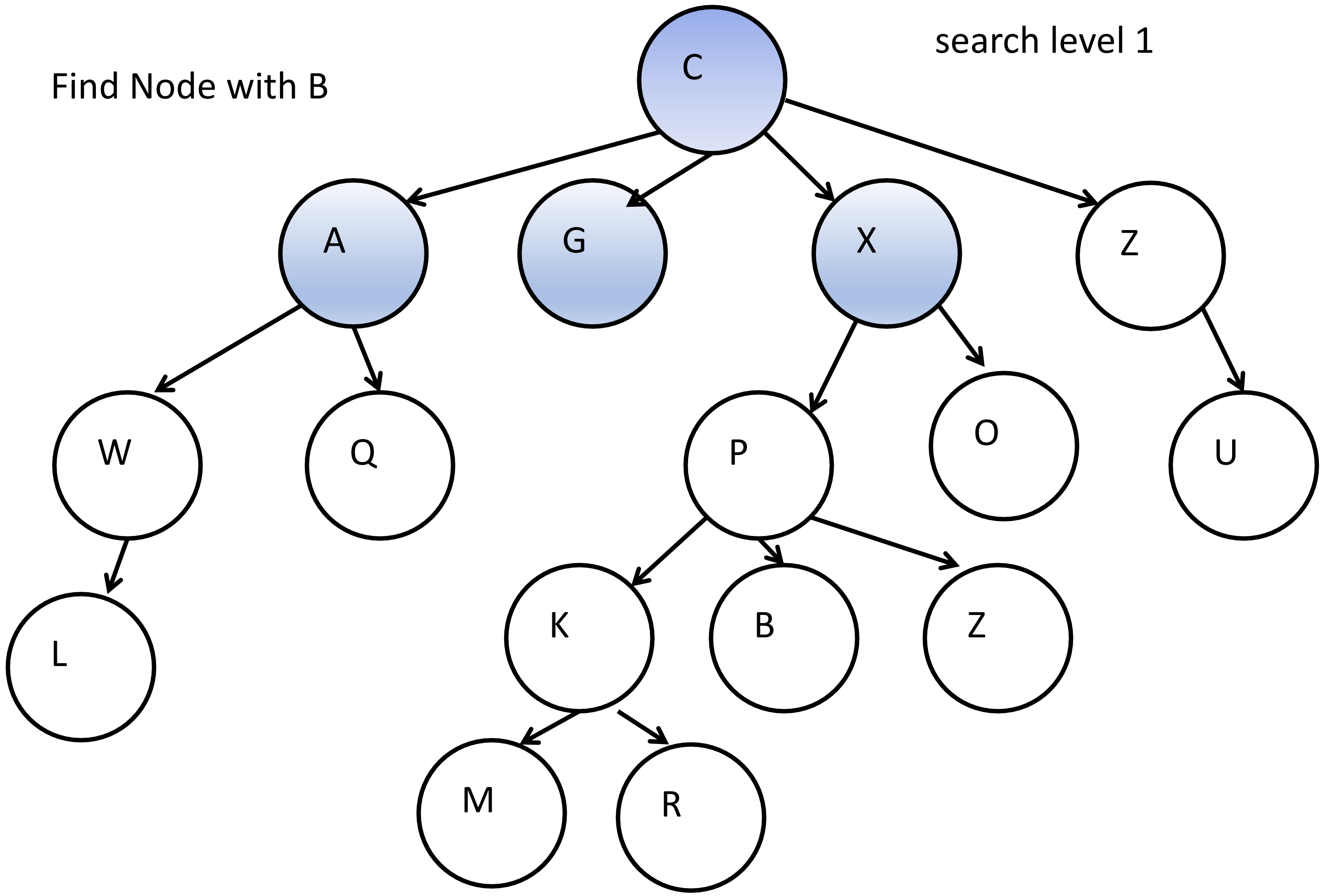
search level 1

Find Node with B



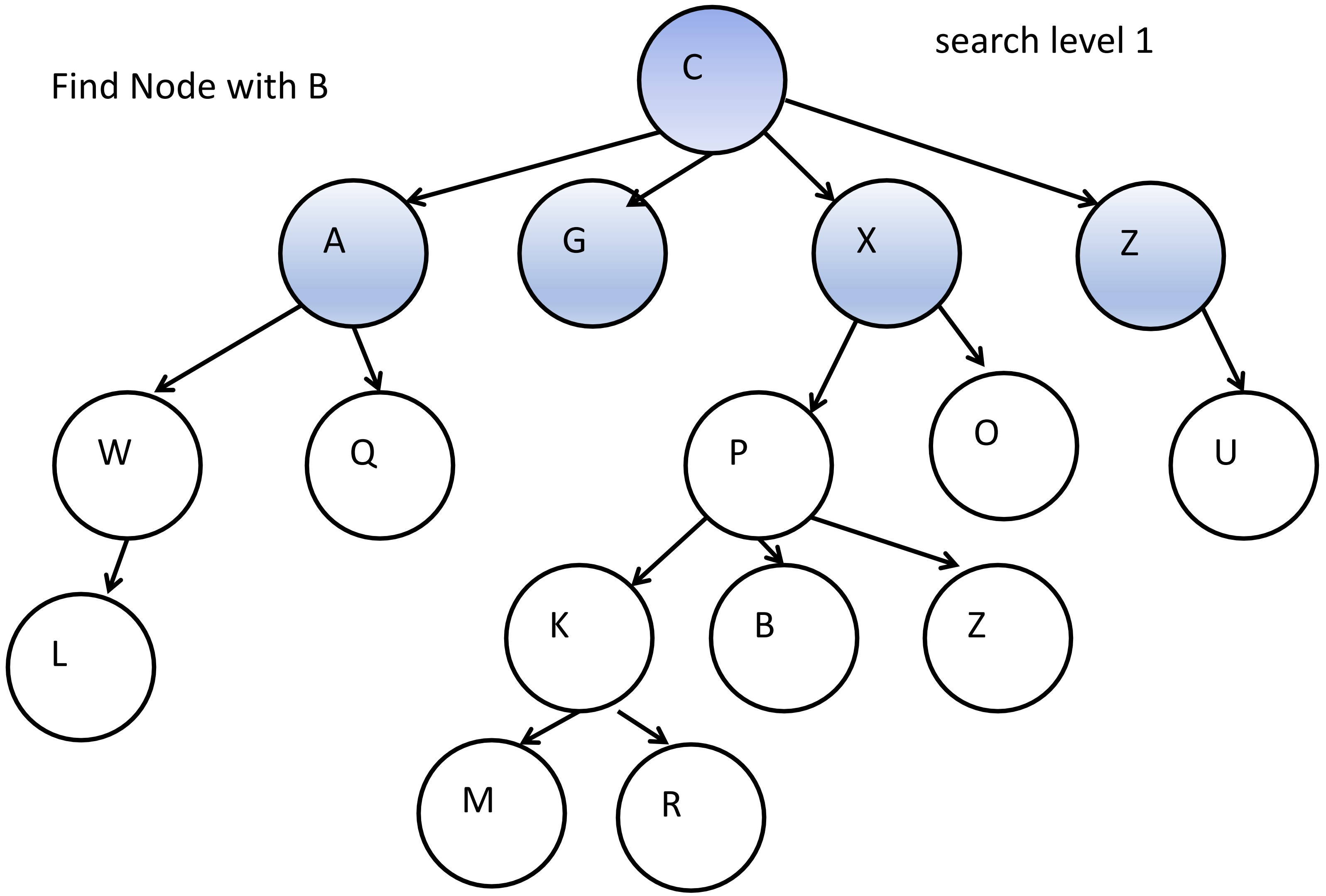
Find Node with B

search level 1



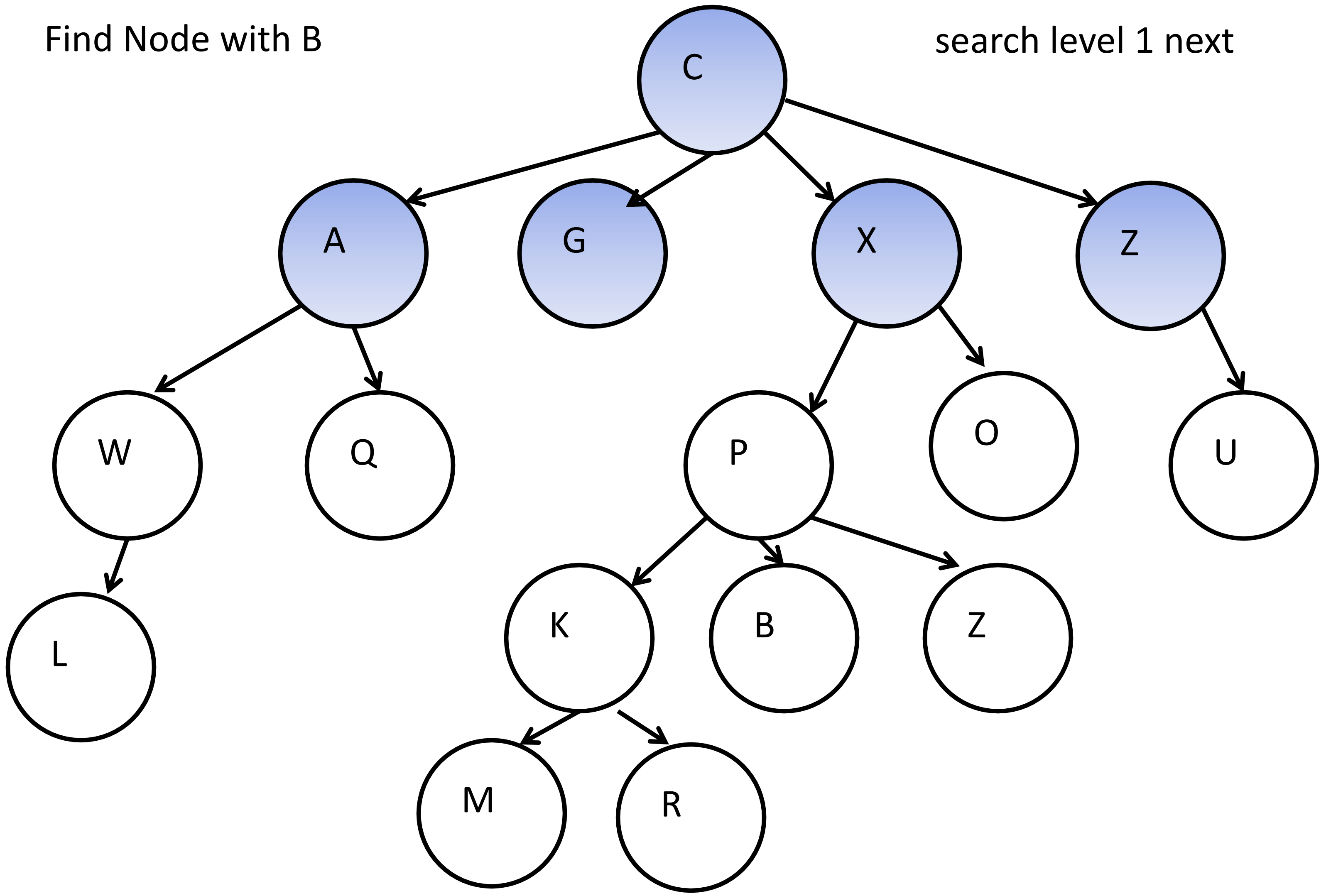
Find Node with B

search level 1



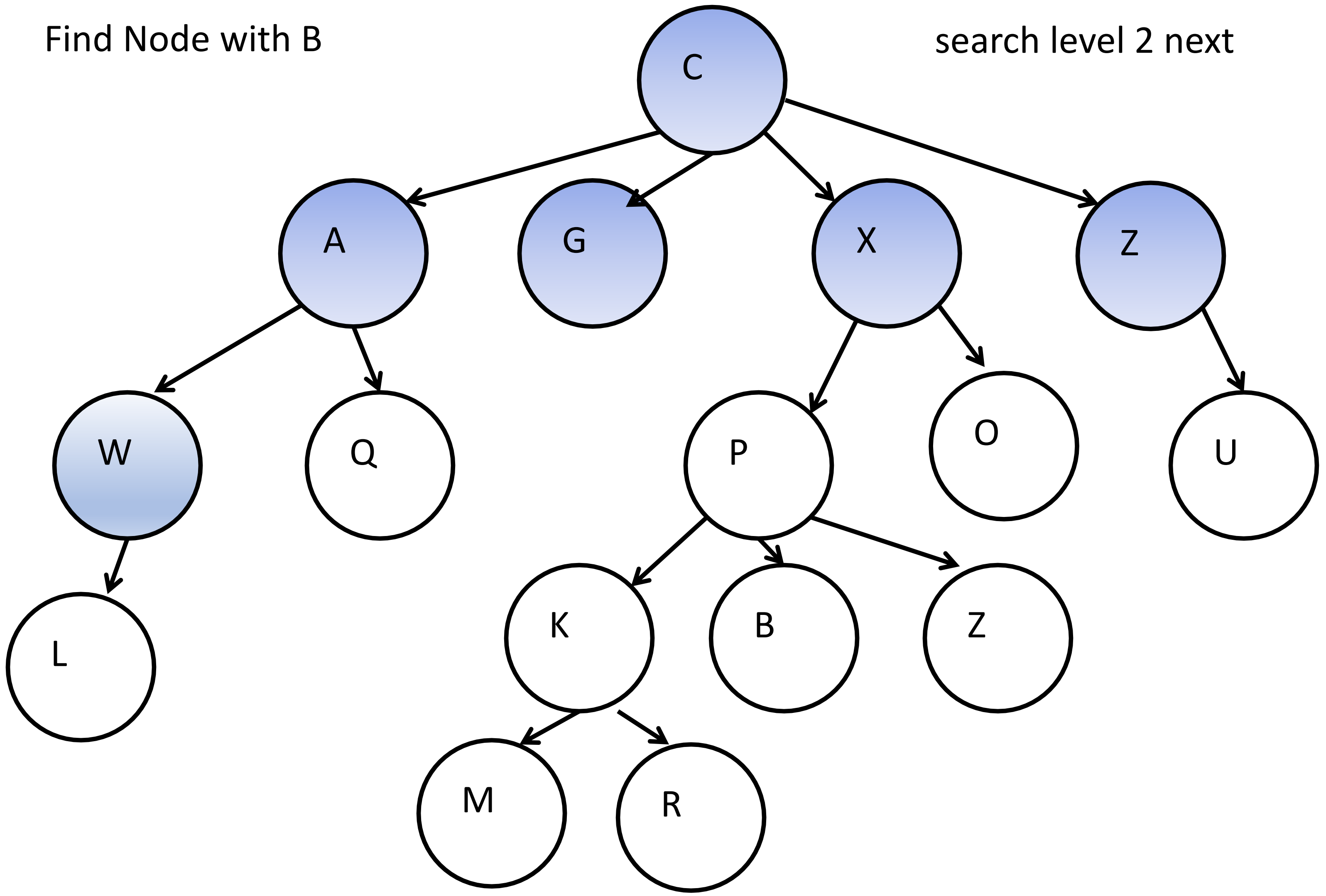
Find Node with B

search level 1 next



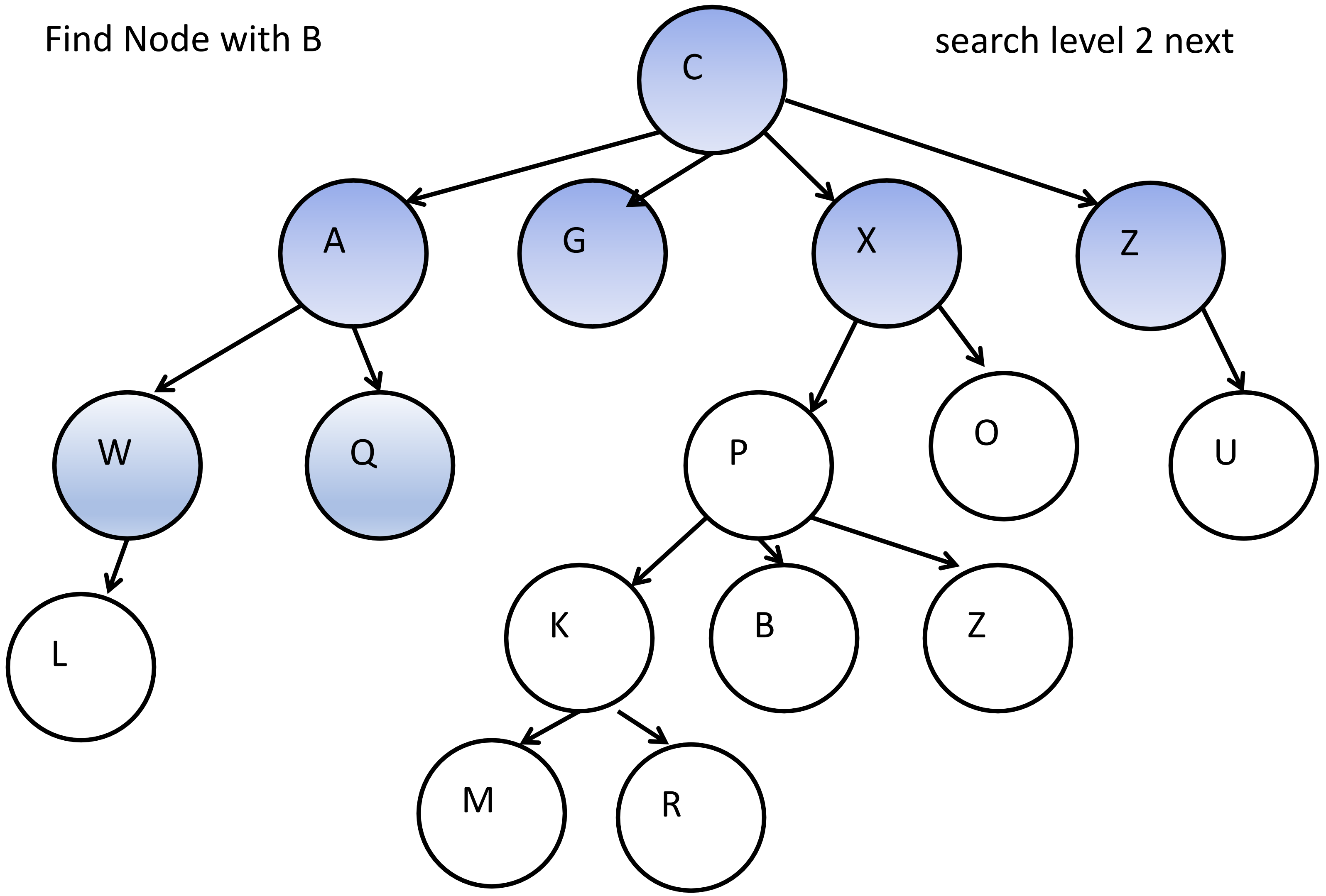
Find Node with B

search level 2 next



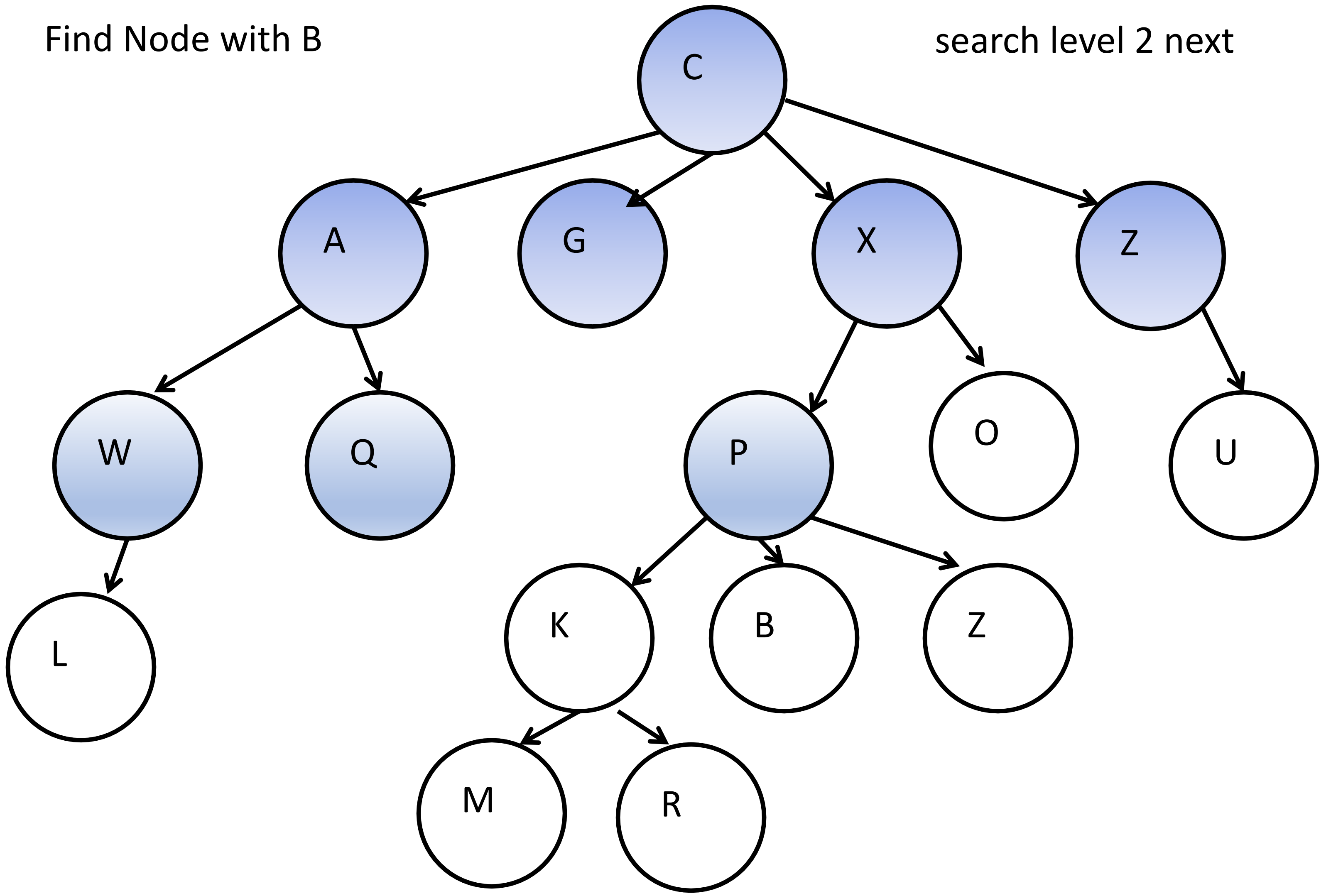
Find Node with B

search level 2 next



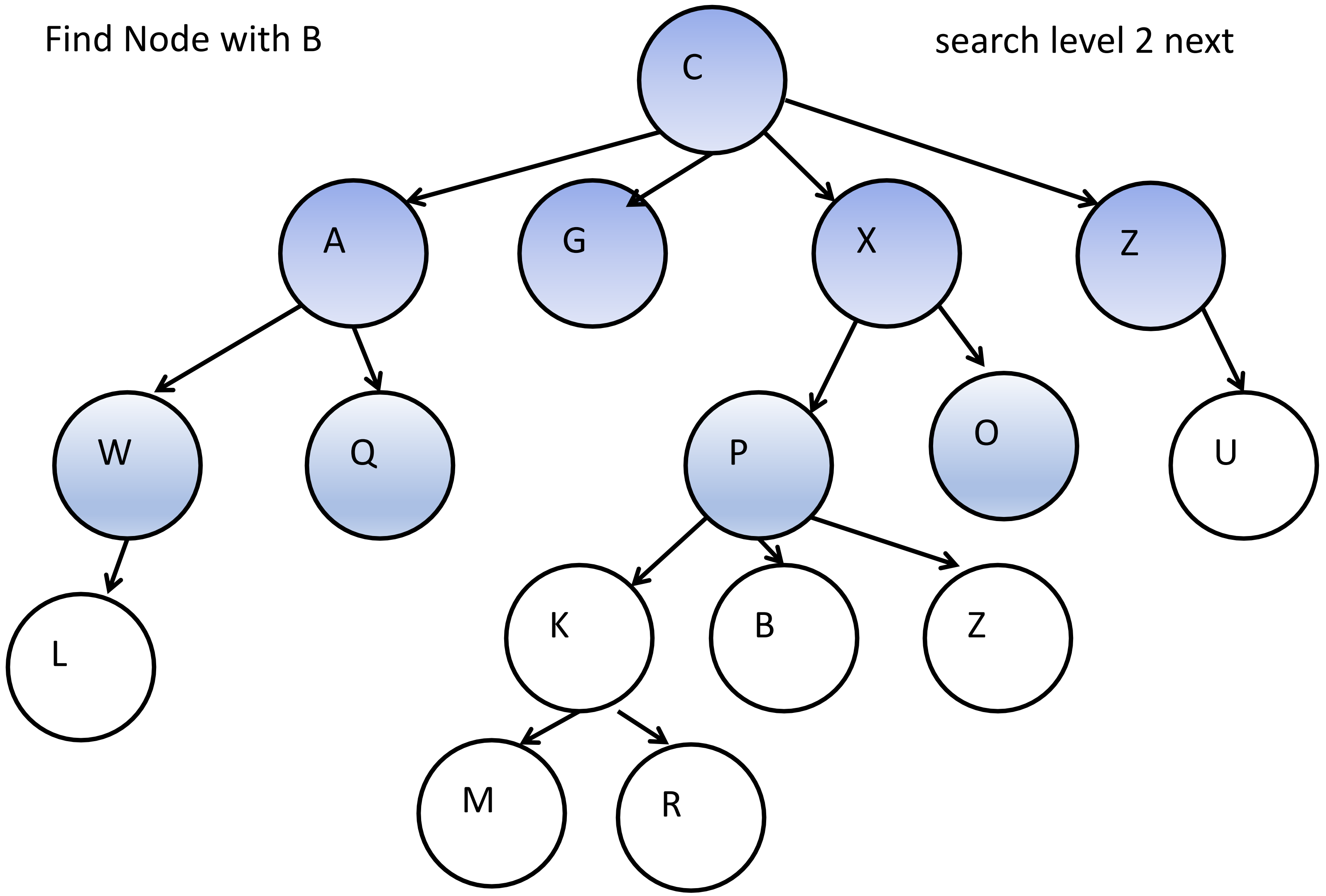
Find Node with B

search level 2 next



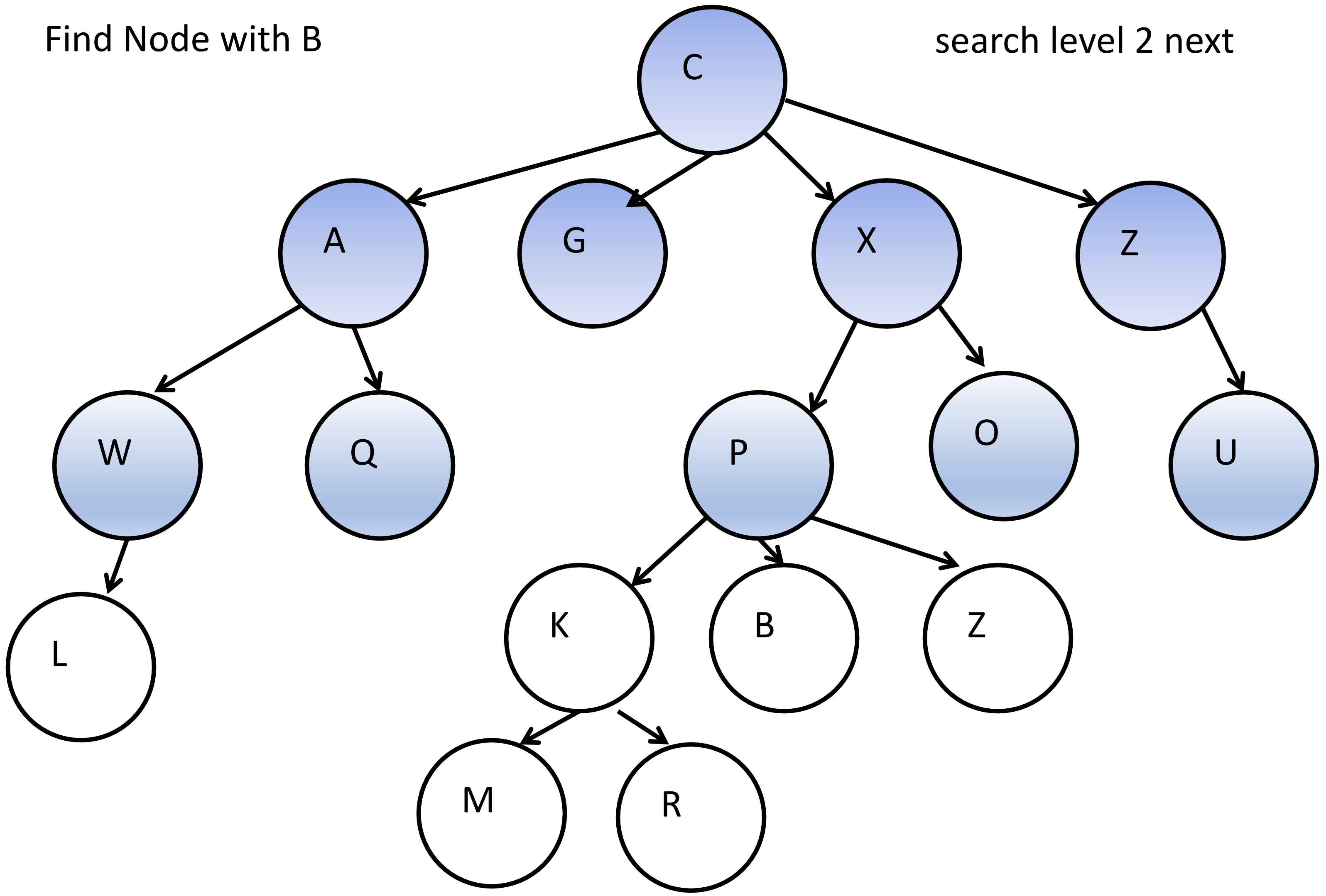
Find Node with B

search level 2 next



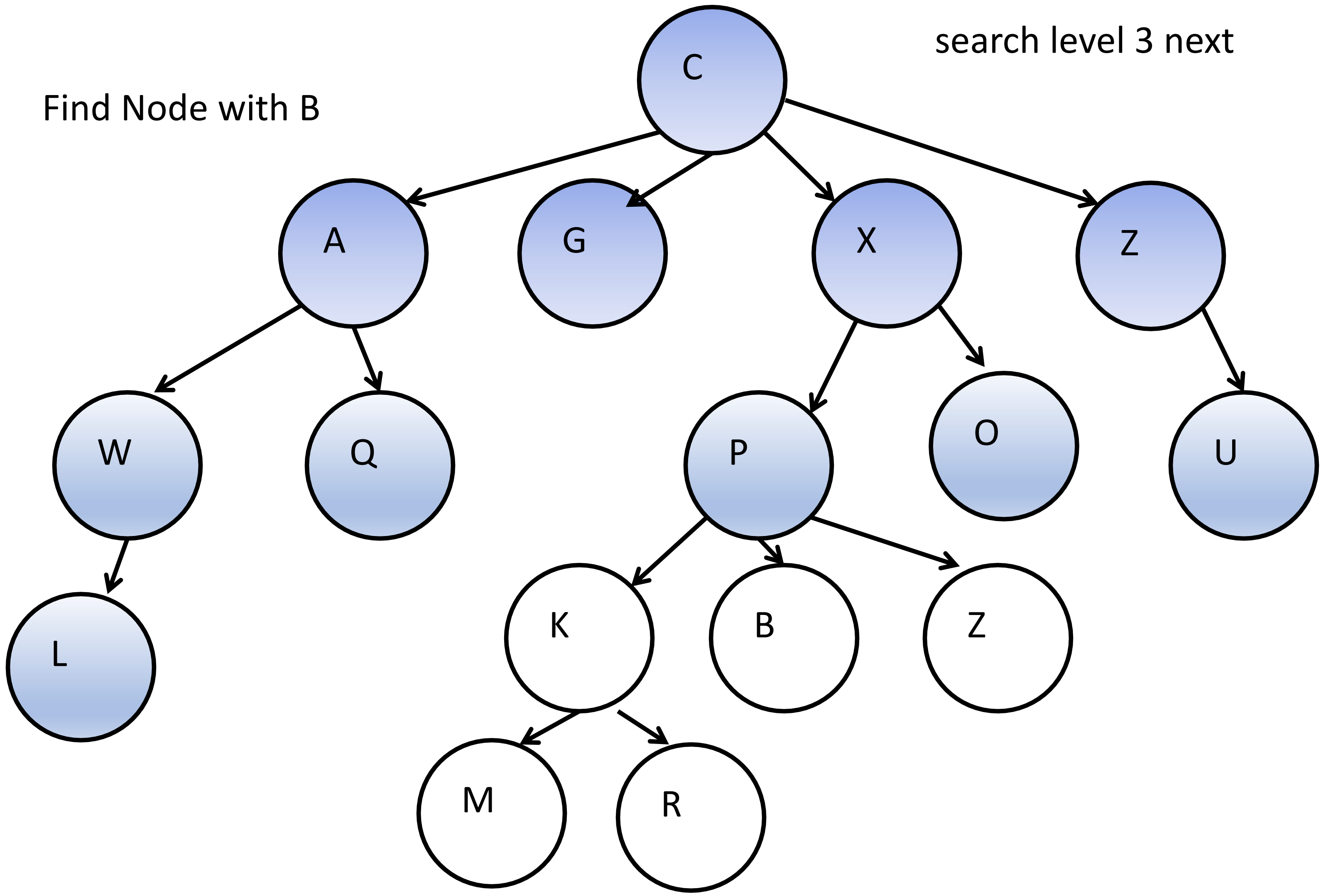
Find Node with B

search level 2 next



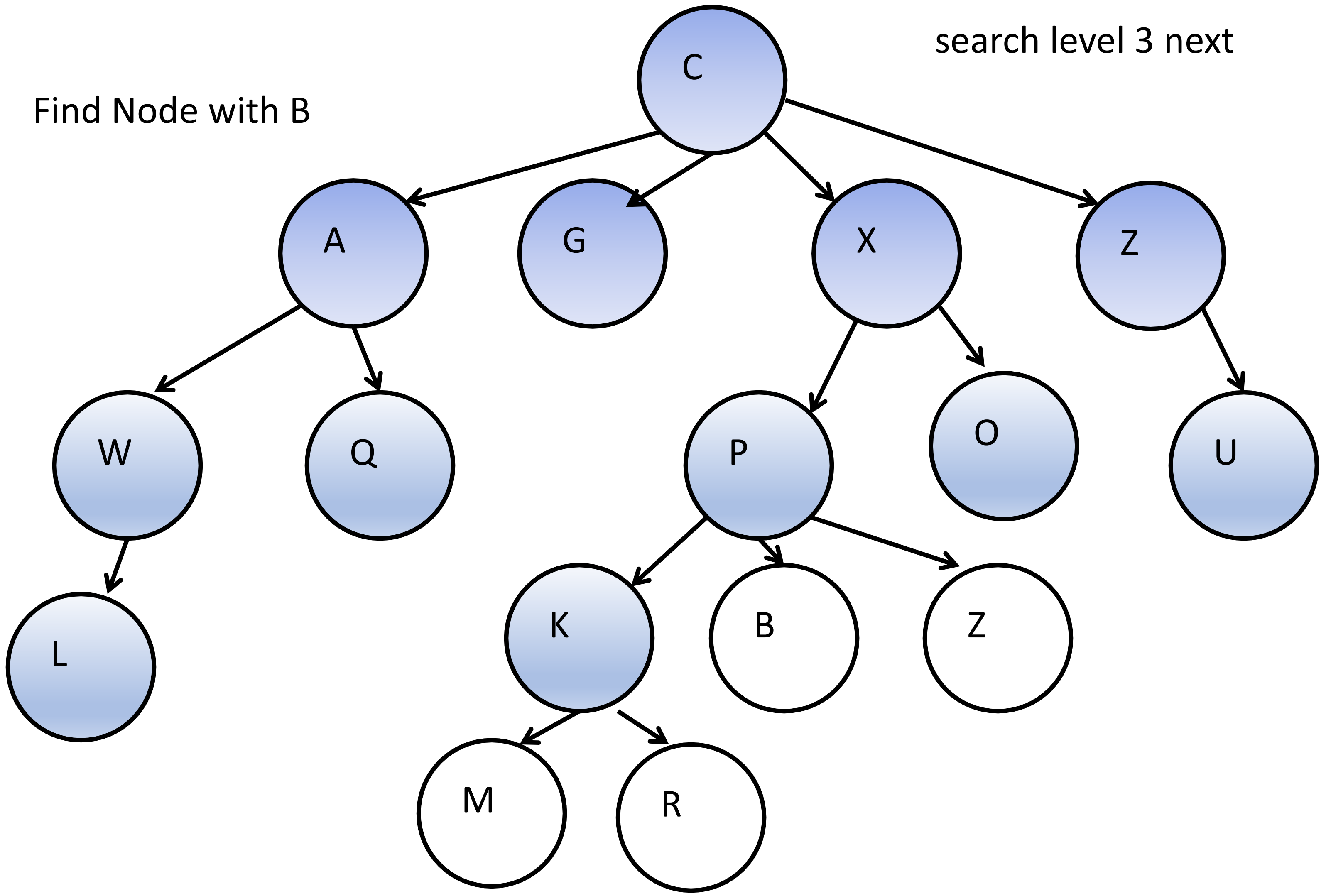
Find Node with B

search level 3 next



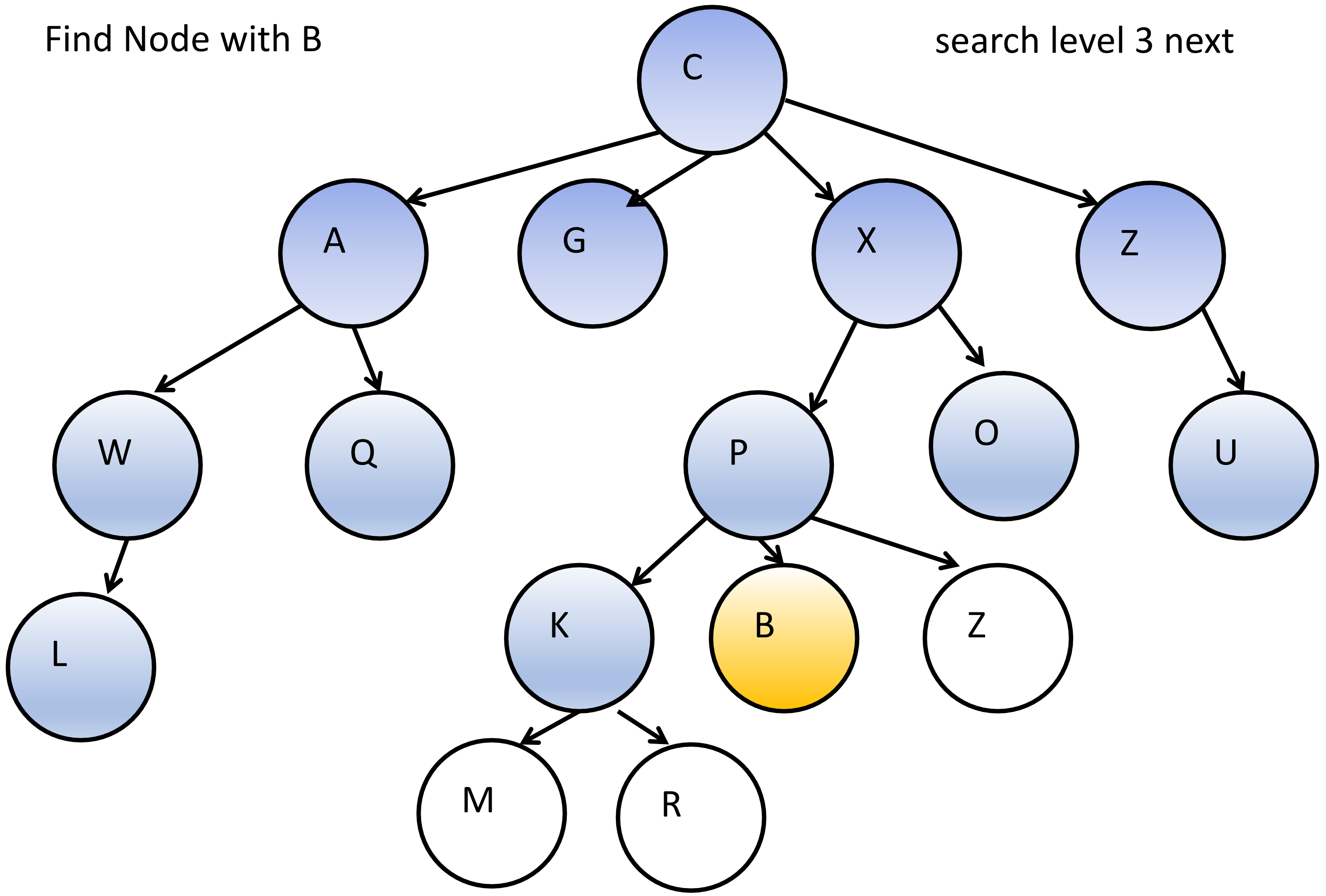
Find Node with B

search level 3 next



Find Node with B

search level 3 next



How to do breadth-first searching

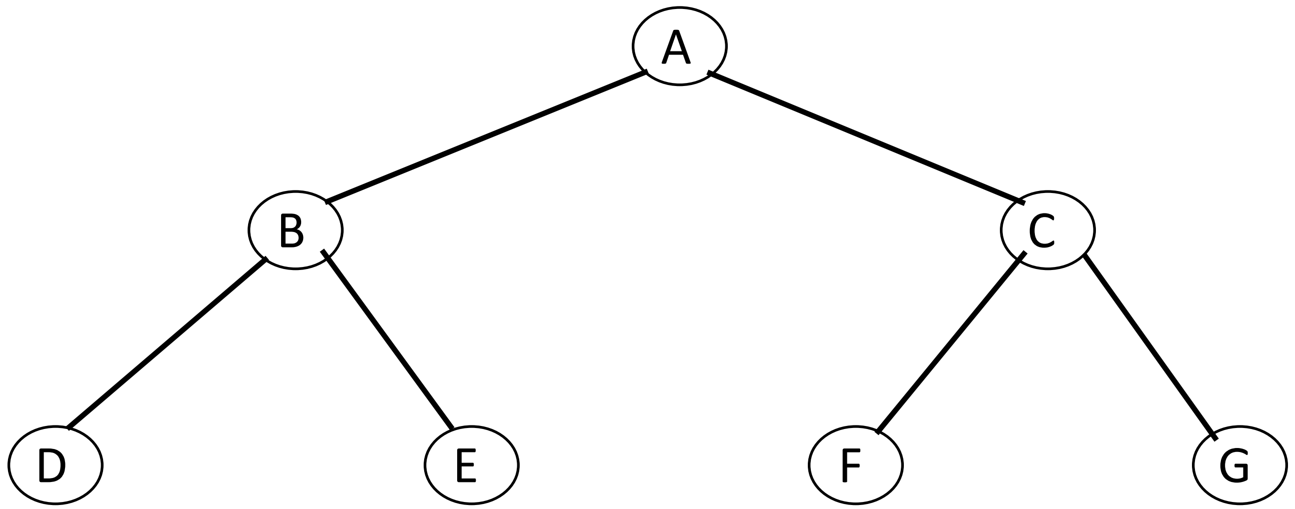
-
  ```
  Put the root node on a queue;
  while (queue is not empty) {
      remove a node from the queue;
      if (node is a goal node) return success;
      put all children of node onto the queue;
  }
  return failure;
  ```
- Just before starting to explore level n, the queue holds *all* the nodes at level n-1
- In a typical tree, the number of nodes at each level increases *exponentially* with the depth
- Memory requirements may be infeasible
- There is *no* “recursive” breadth-first search equivalent to recursive depth-first search

Pseudo-Code for Breadth-First Traversal

<u>breadth-first-traversal</u>

```
put root node onto a queue
while the queue is not empty
   dequeue the next node
   visit the node          e.g., print value
   enqueue the left child node
   enqueue the right child node
```

Breadth-First Search

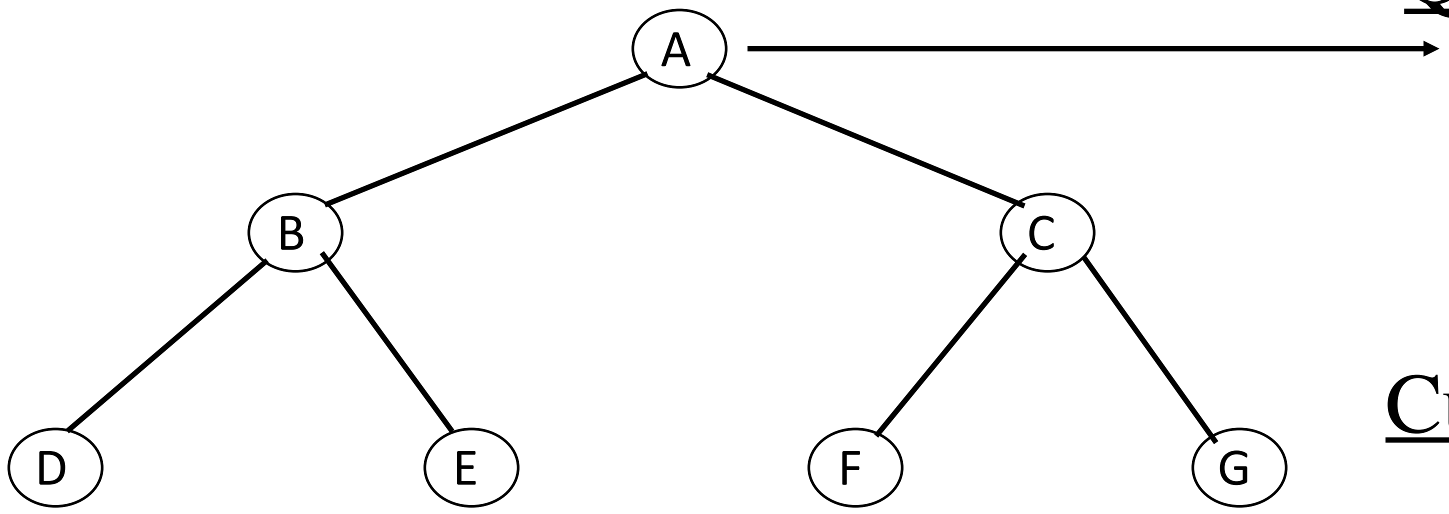


Queue:

Current:

A B C D E F G

Breadth-First Search

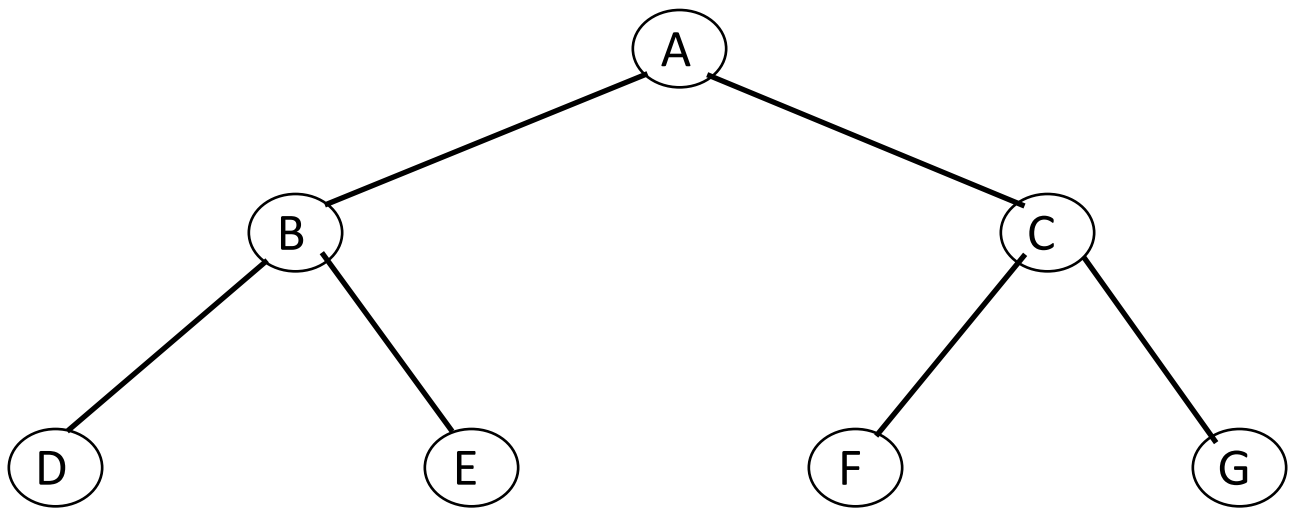


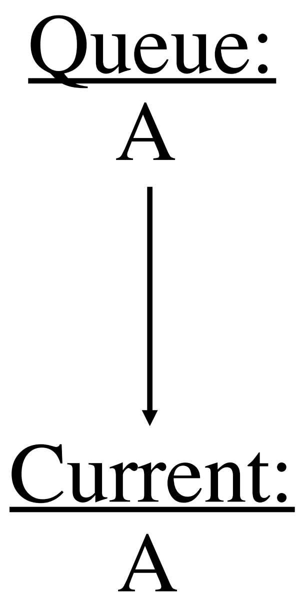
Queue:

A

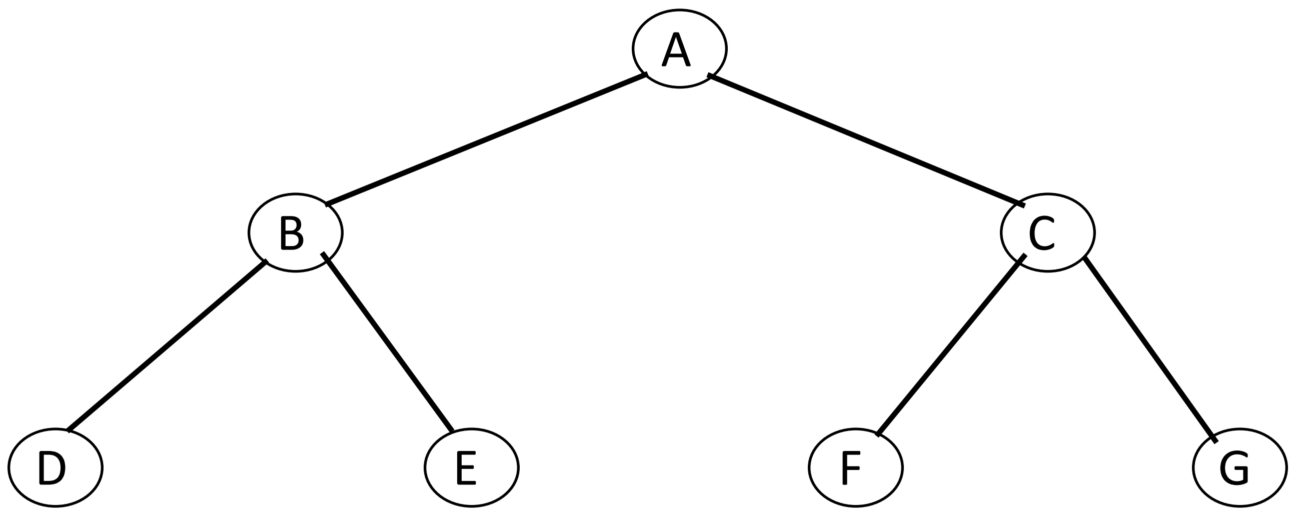
Current:

Breadth-First Search





Breadth-First Search

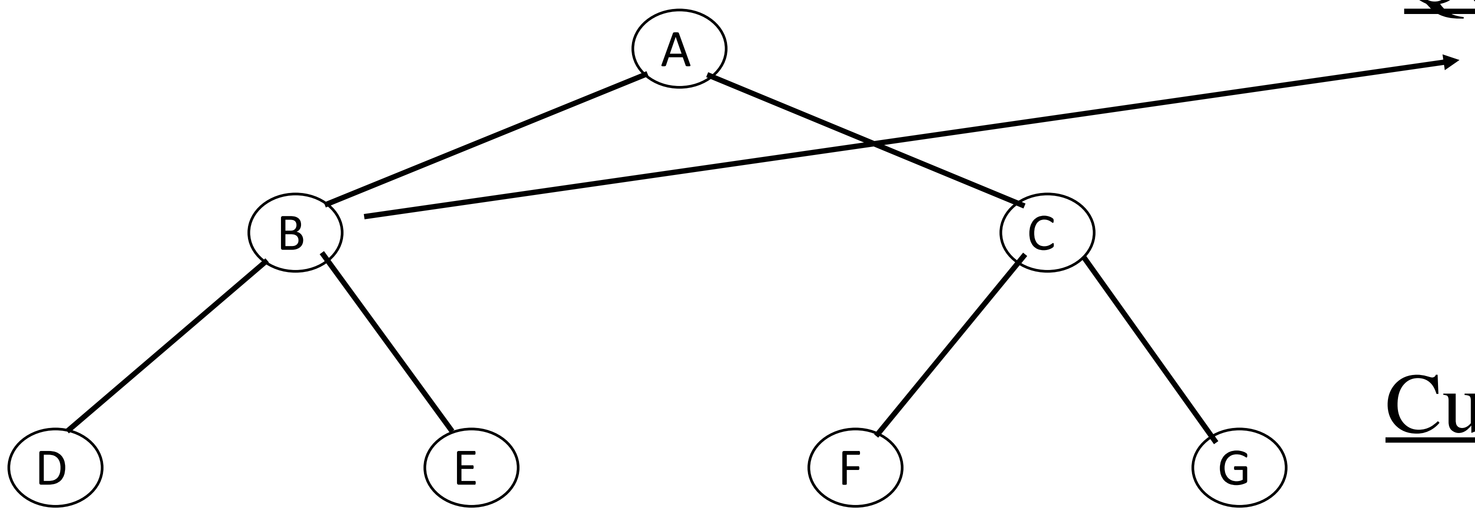


Queue:

Current:
A

A

Breadth-First Search



Queue:

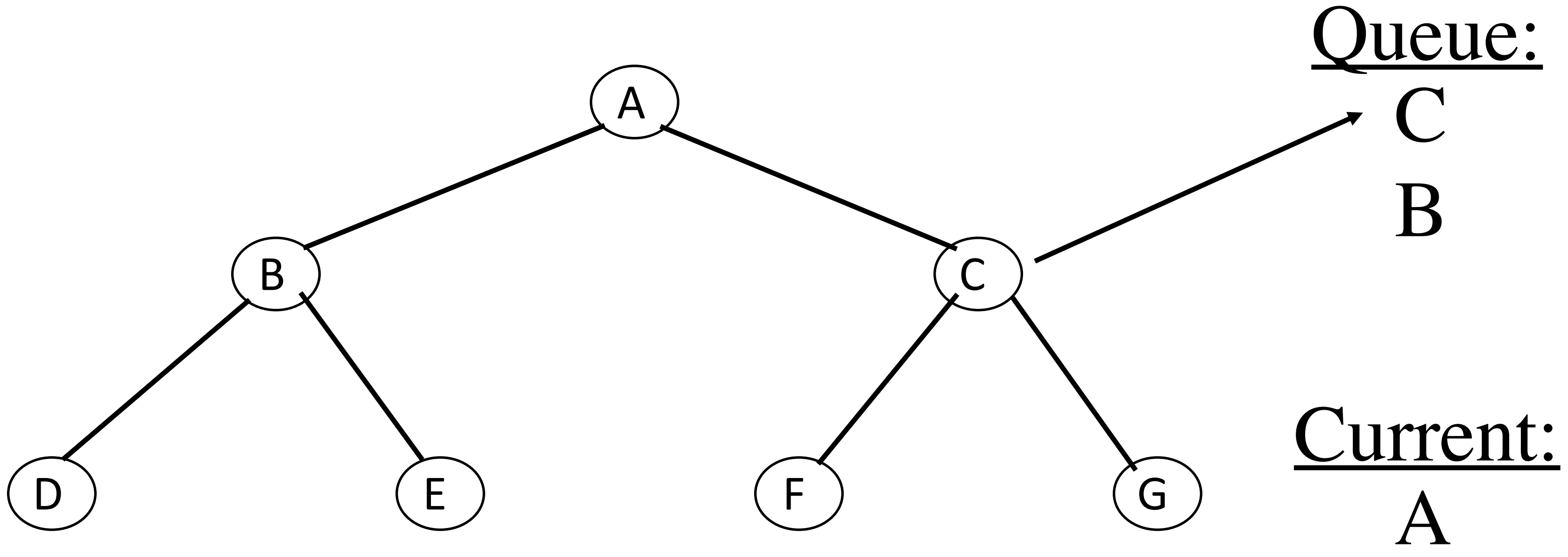
B

Current:

A

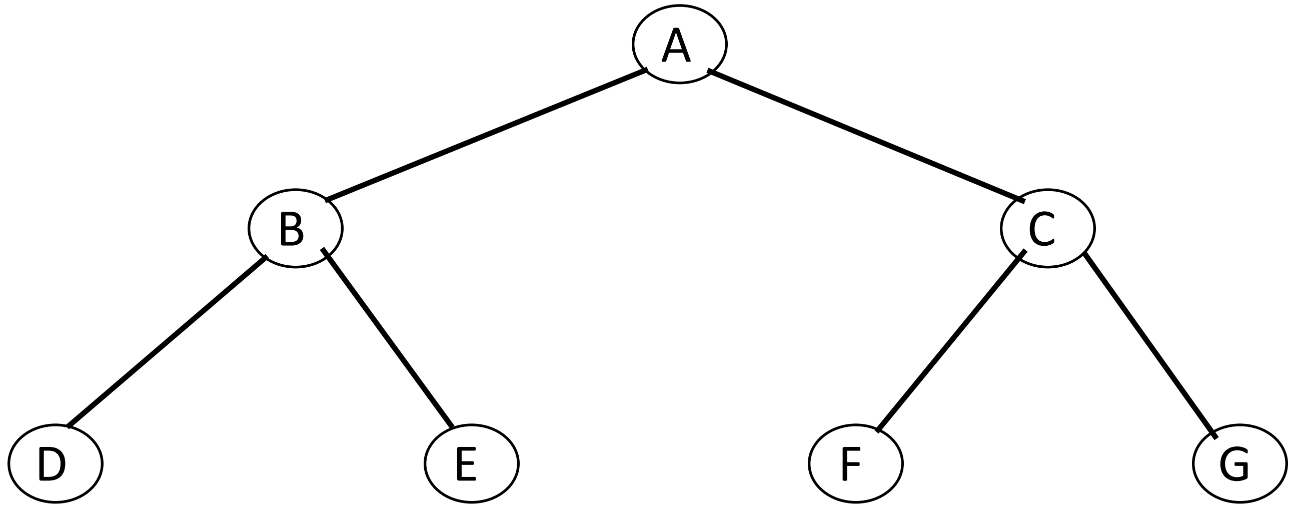
A

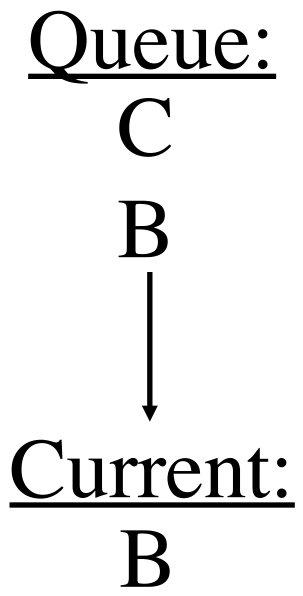
Breadth-First Search



A

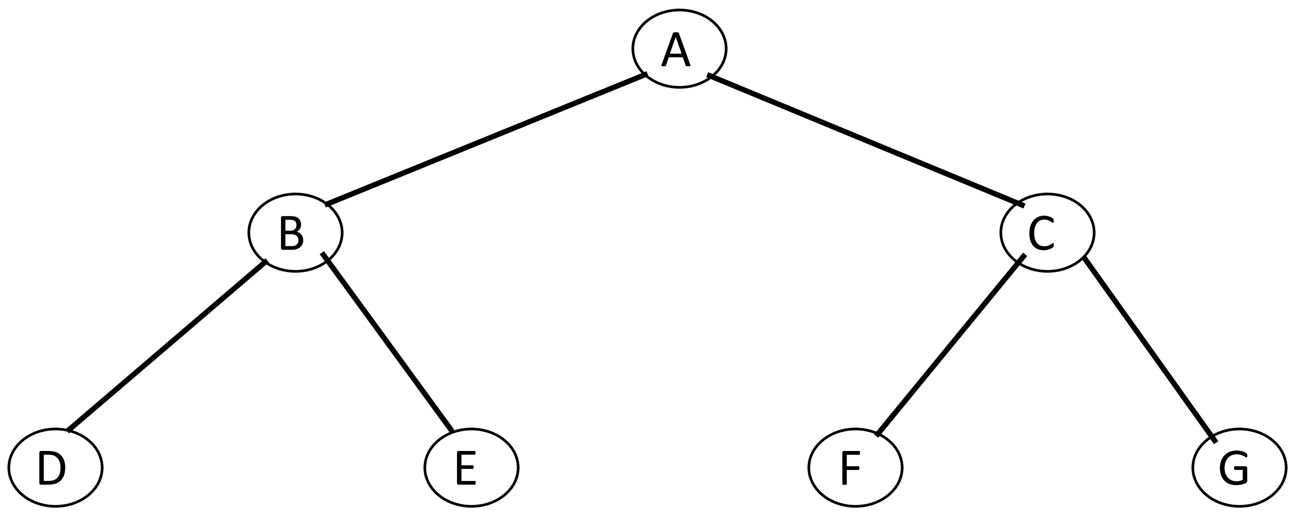
Breadth-First Search





A

Breadth-First Search

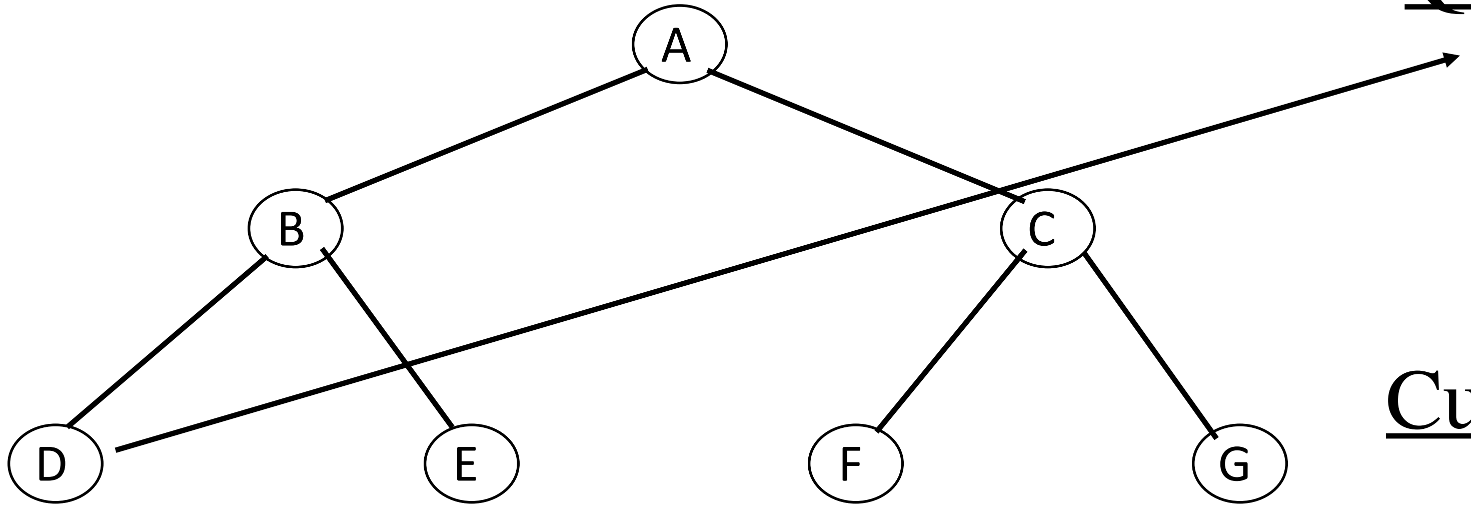


<u>Queue:</u>
C

<u>Current:</u>
B

A B

Breadth-First Search



Queue:

D

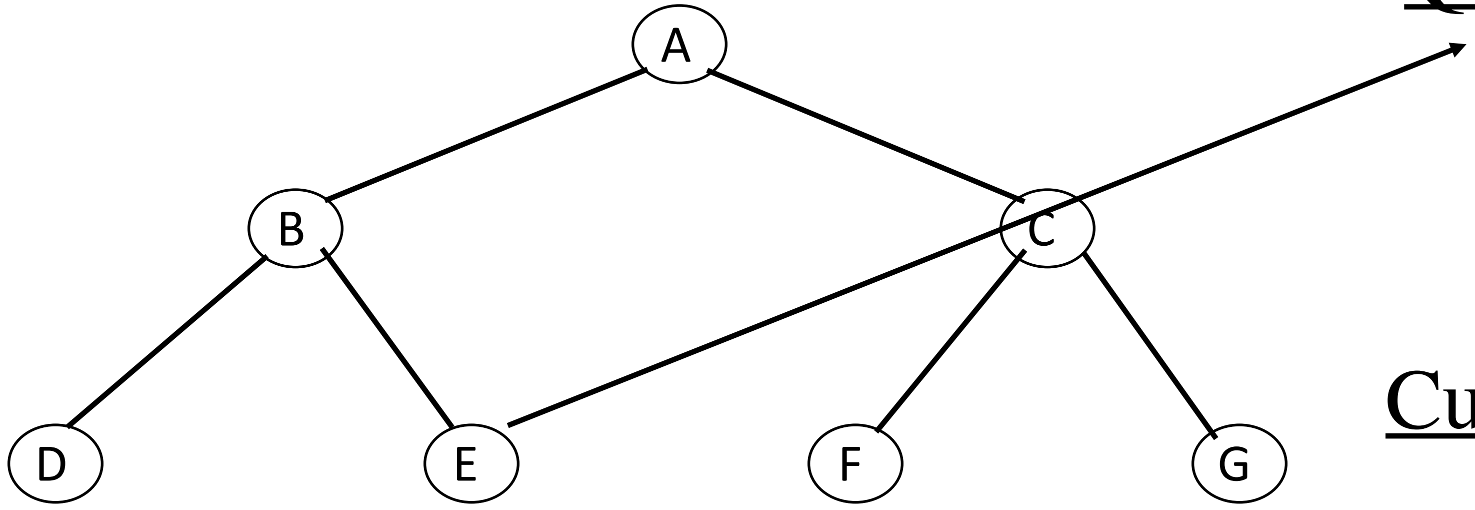
C

Current:

B

A B

Breadth-First Search



Queue:

E

D

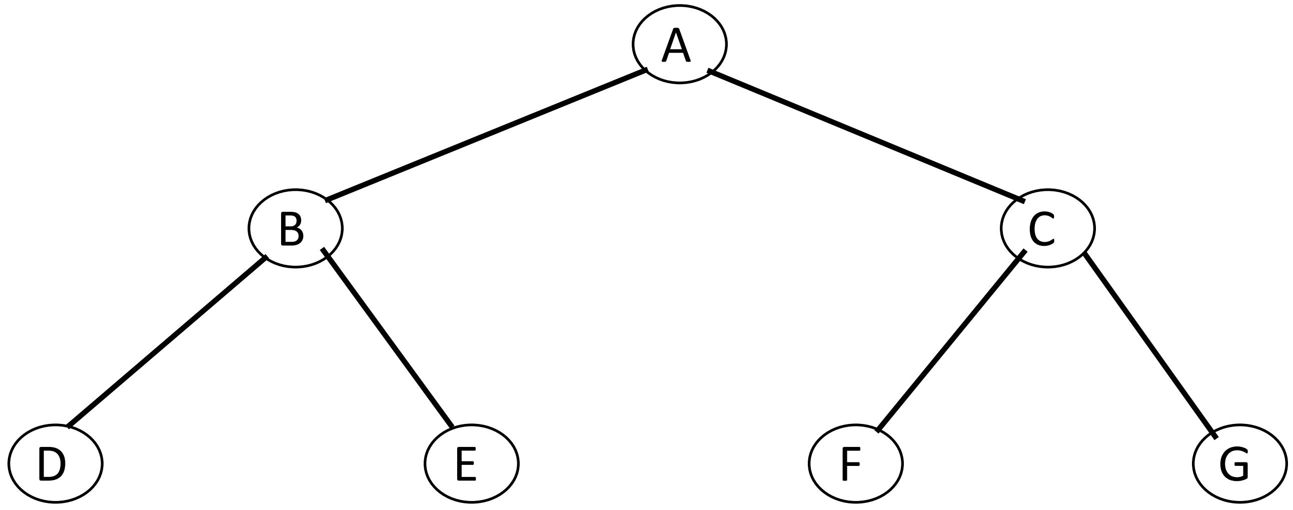
C

Current:

B

A B

Breadth-First Search

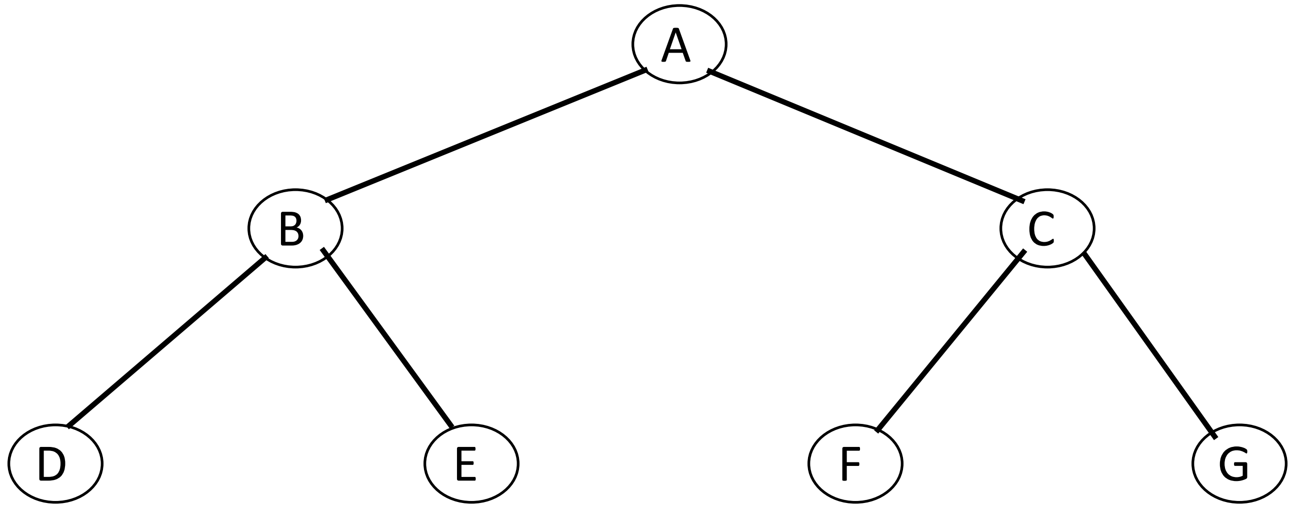


Queue:
E
D
C

Current:
C

A B

Breadth-First Search



Queue:

E

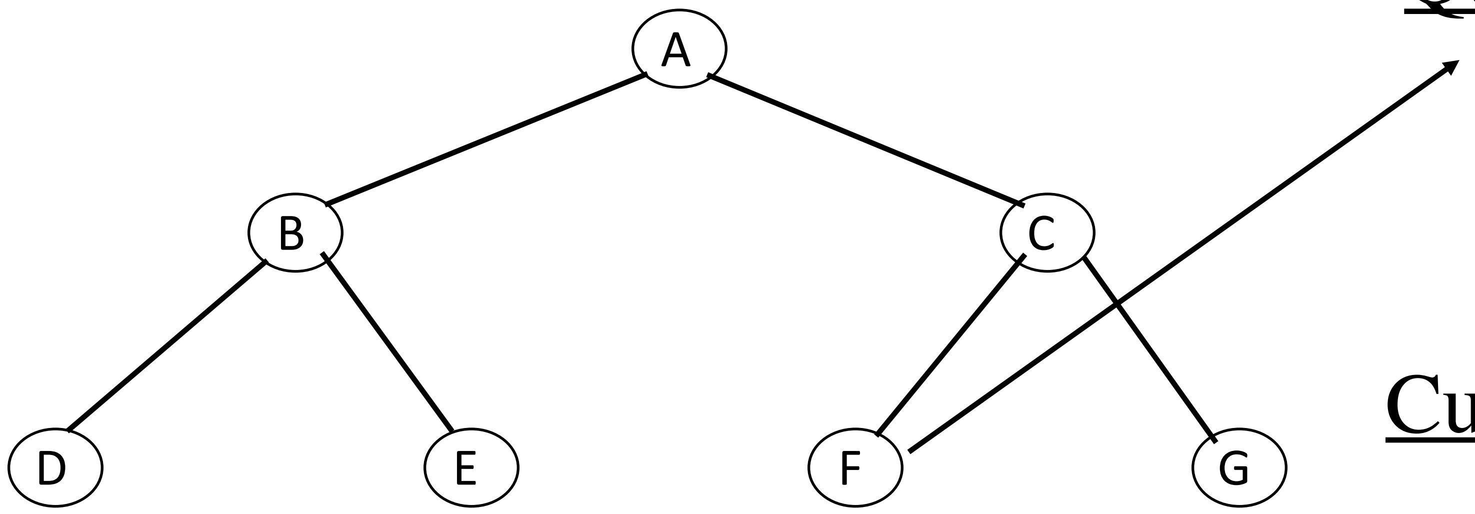
D

Current:

C

A B C

Breadth-First Search



Queue:

F

E

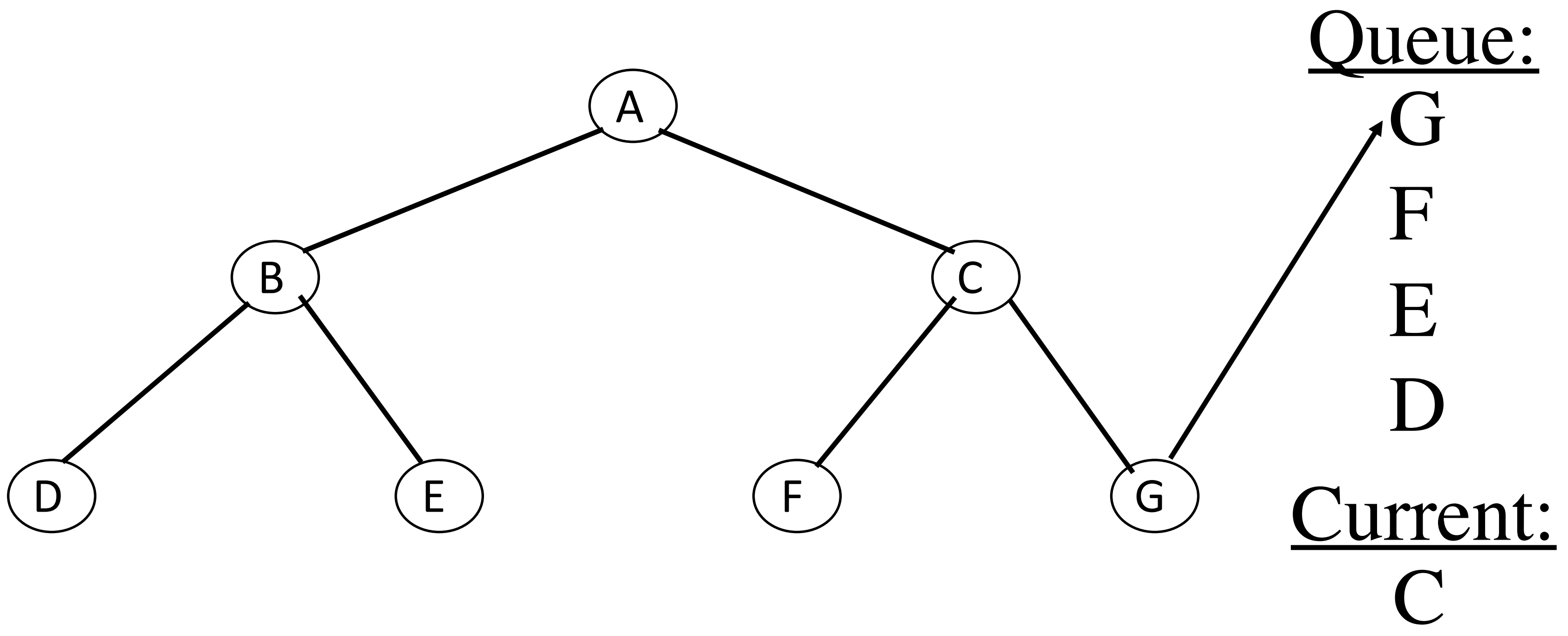
D

Current:

C

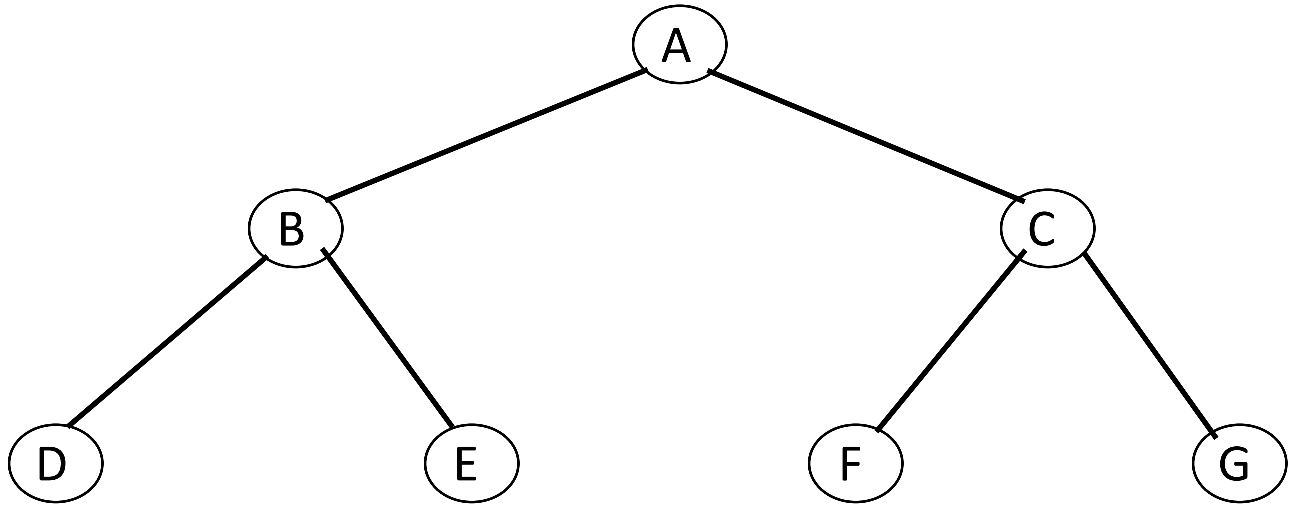
A B C

Breadth-First Search



A B C

Breadth-First Search



Queue:

G

F

E

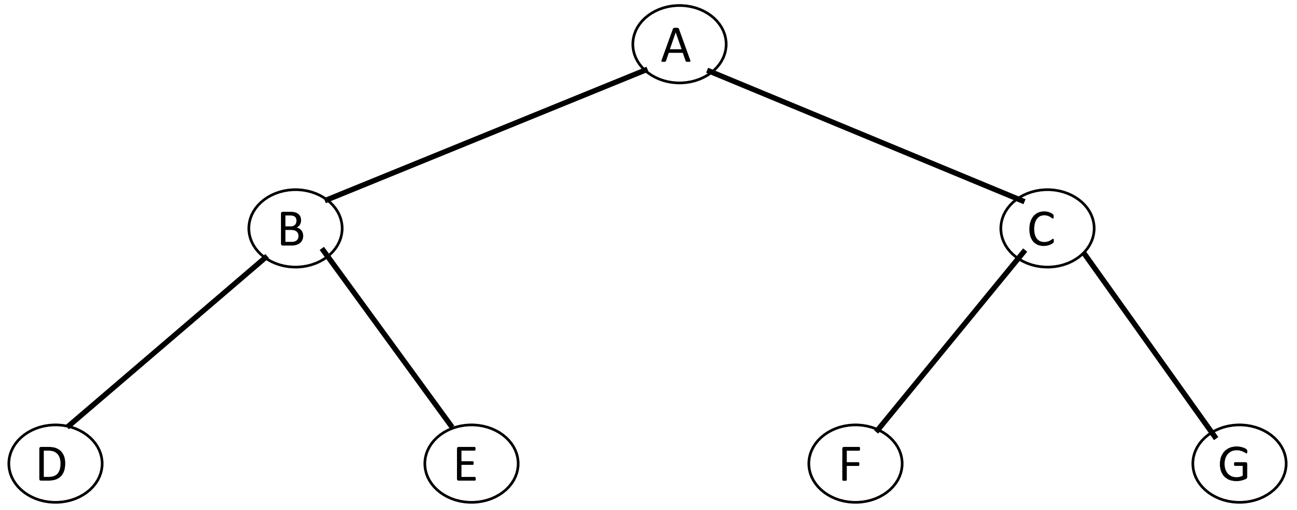
D

Current:

D

A B C

Breadth-First Search



Queue:

G

F

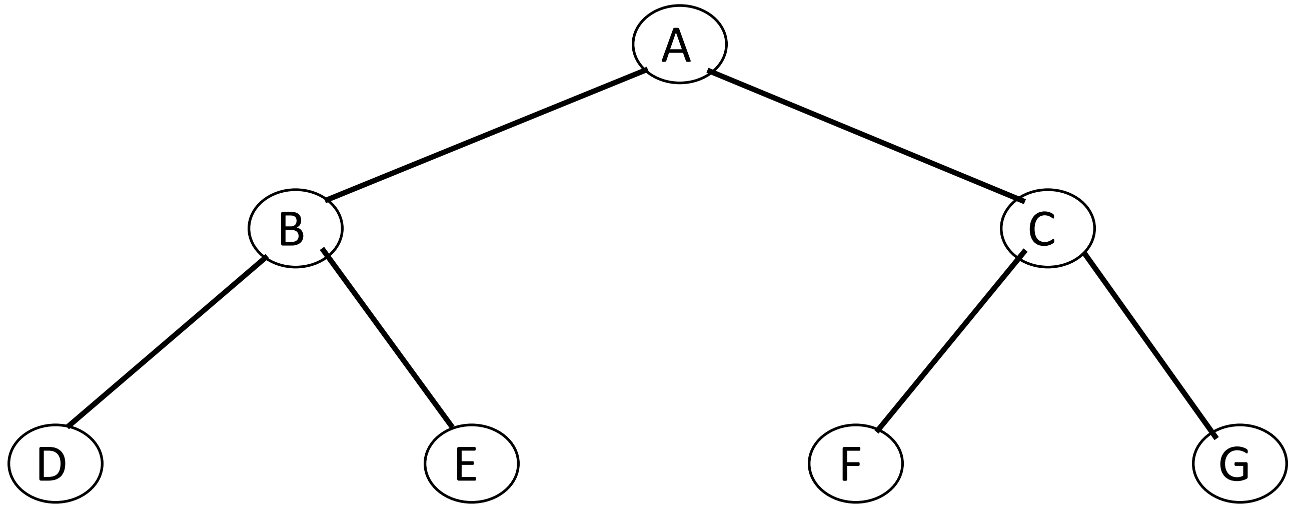
E

Current:

D

A B C D

Breadth-First Search



Queue:

G

F

E

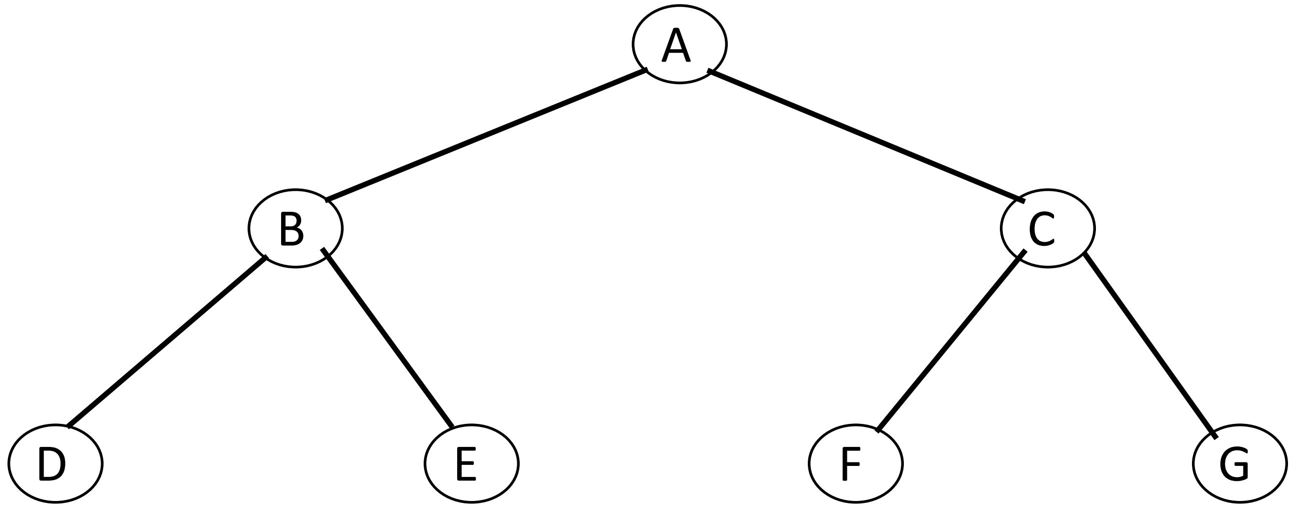
↓

Current:

E

A B C D

Breadth-First Search

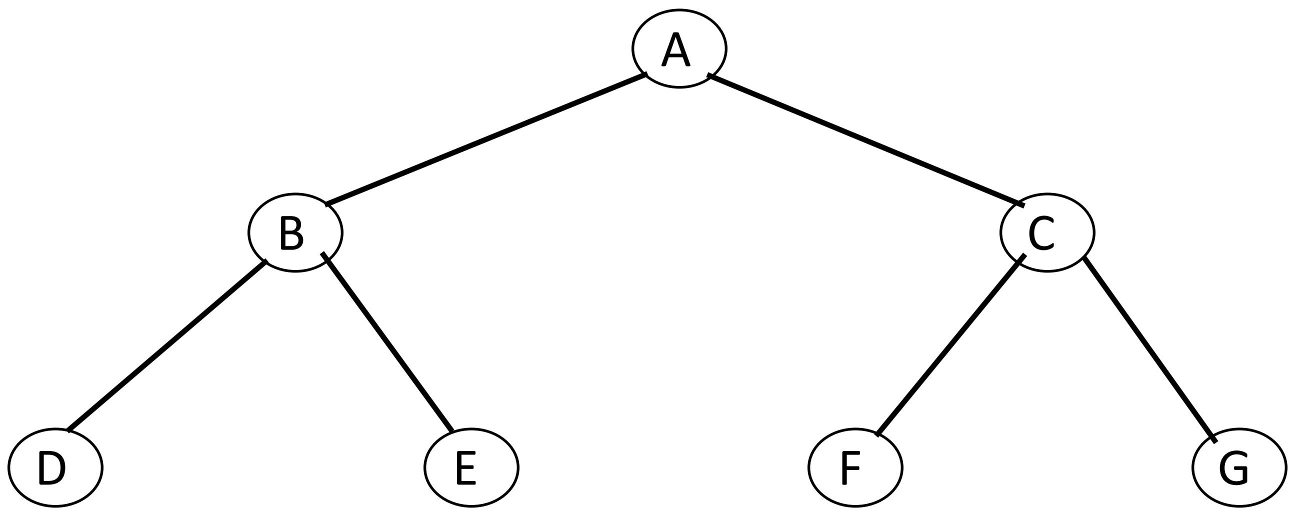


Queue:
G
F

Current:
E

A B C D E

Breadth-First Search

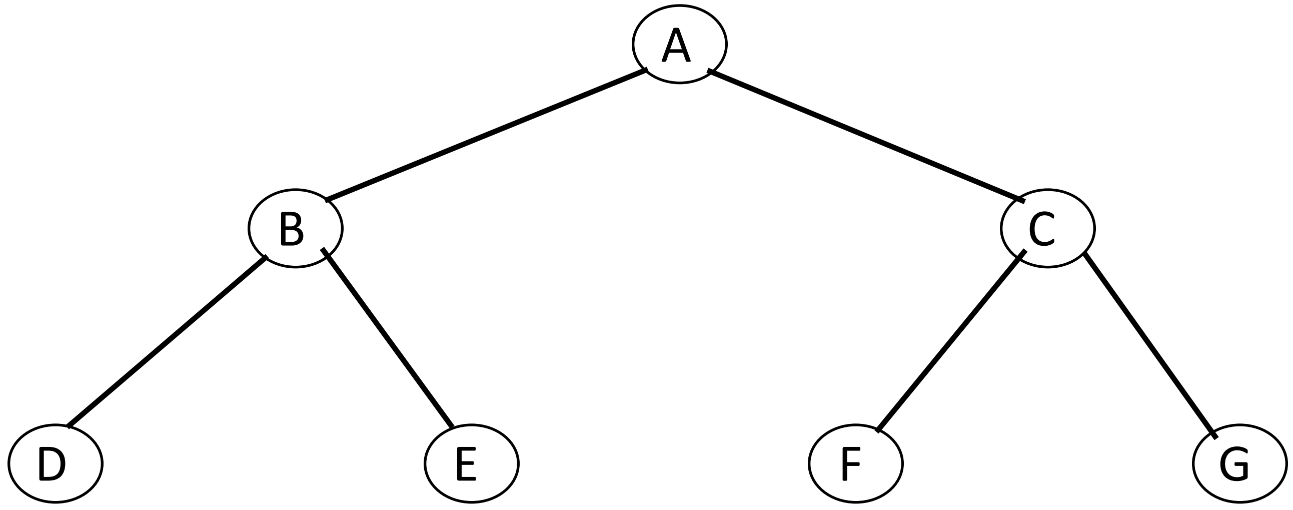


Queue:
G
F

Current:
F

A B C D E

Breadth-First Search

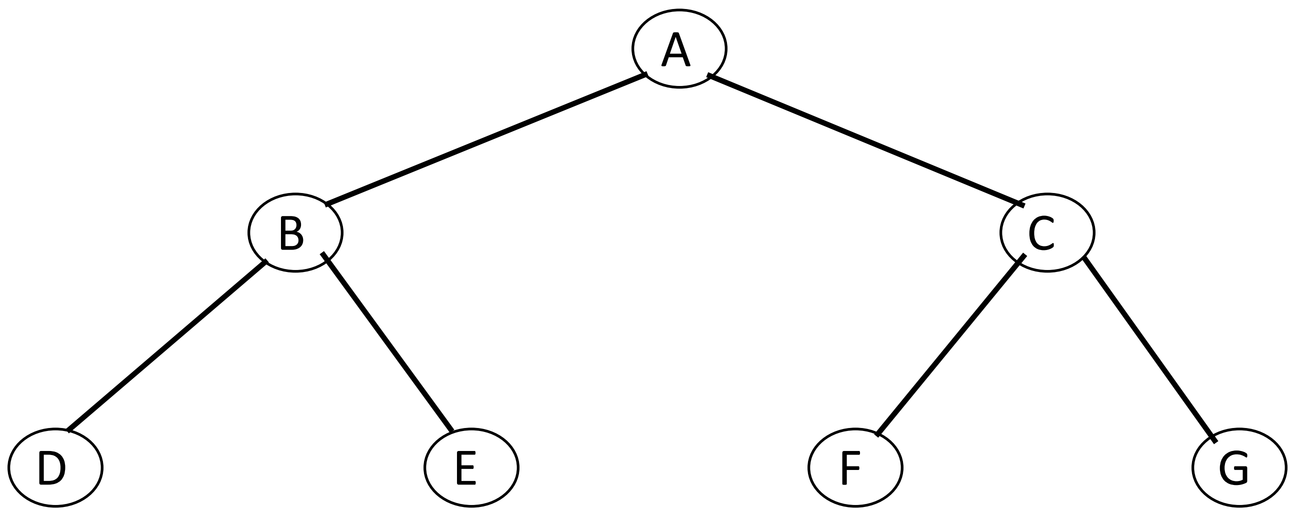


Queue:
G

Current:
F

A B C D E F

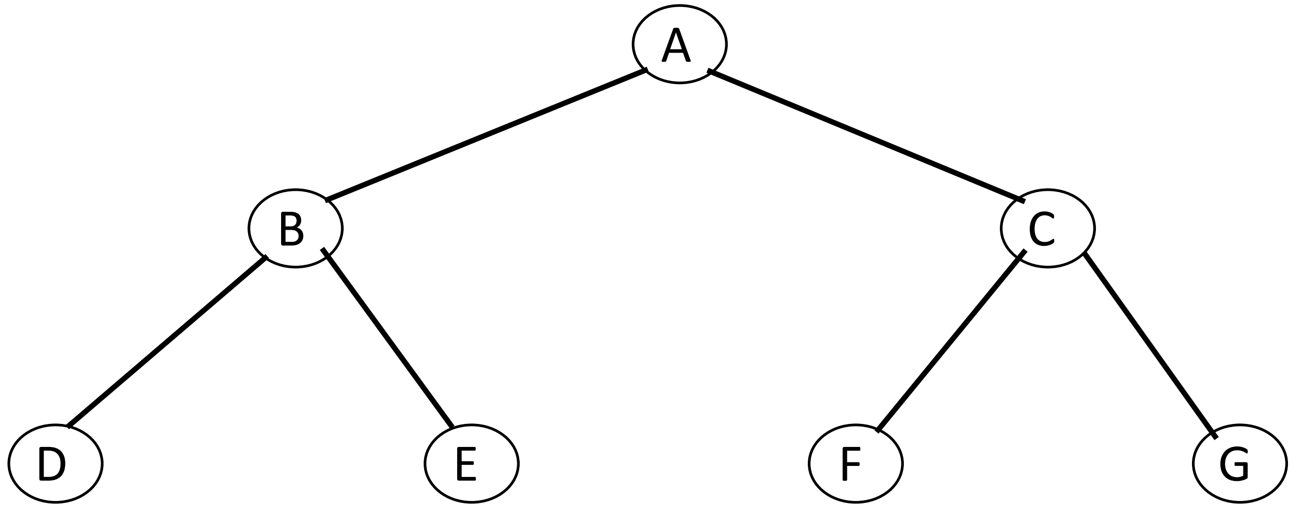
Breadth-First Search



$$\begin{tikzcd}
\underline{\text{Queue:}} \\
G \arrow[d] \\
\underline{\text{Current:}} \\
G
\end{tikzcd}$$

A B C D E F

Breadth-First Search

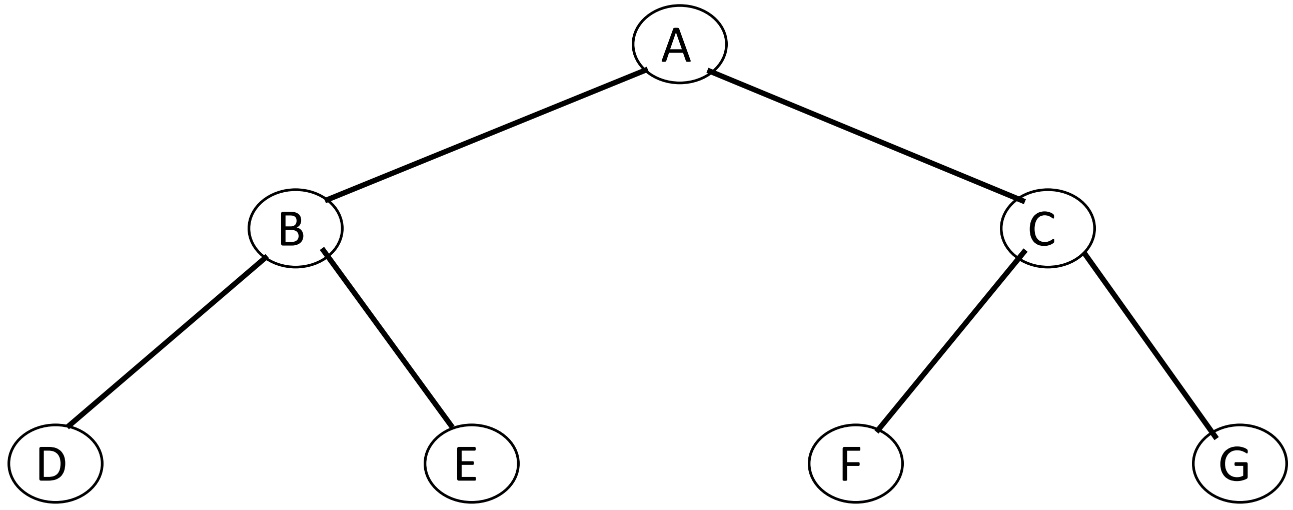


Queue:

Current:
G

A B C D E F G

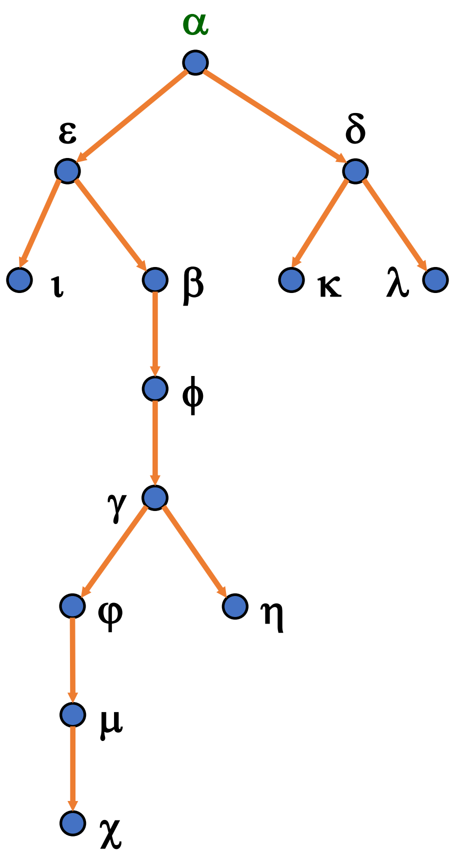
Breadth-First Search



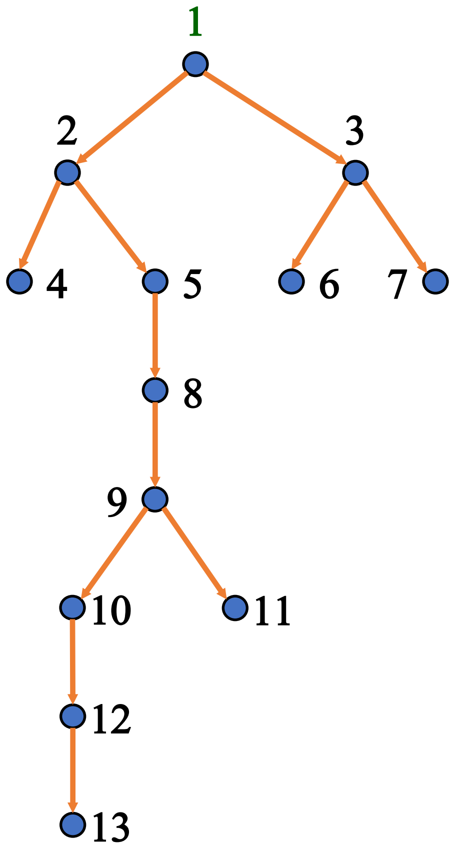
A B C D E F G

Breadth first tree traversal with a queue

- Enqueue root
- While queue is not empty
  - Dequeue a vertex and write it to the output list
  - Enqueue its children left-to-right



Step	*Output*	*Queue*
0		α
1	α	ε,δ
2	ε	δ,ι,β
3	δ	ι,β,κ,λ
4	ι	β,κ,λ
5	β	κ,λ,ϕ
6	κ	λ,ϕ
7	λ	ϕ
8	ϕ	γ
9	γ	φ,η
10	φ	η,μ
11	η	μ,χ
12	μ	χ
13	χ	

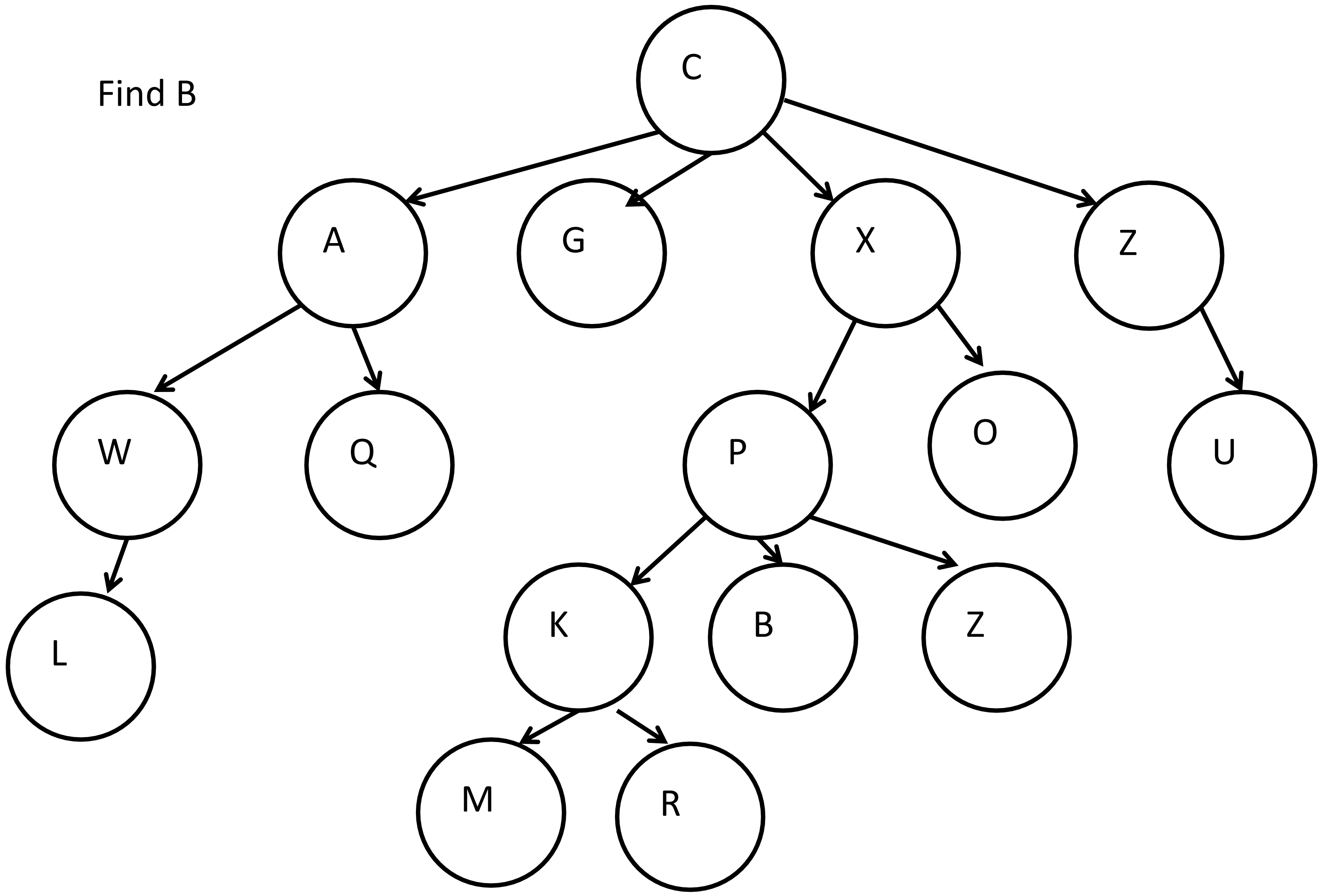


BFS - DFS

- Breadth first search typically implemented with a Queue
- Depth first search typically implemented with a stack, implicit with recursion or iteratively with an explicit stack
- which technique do I use?
  - depends on the problem

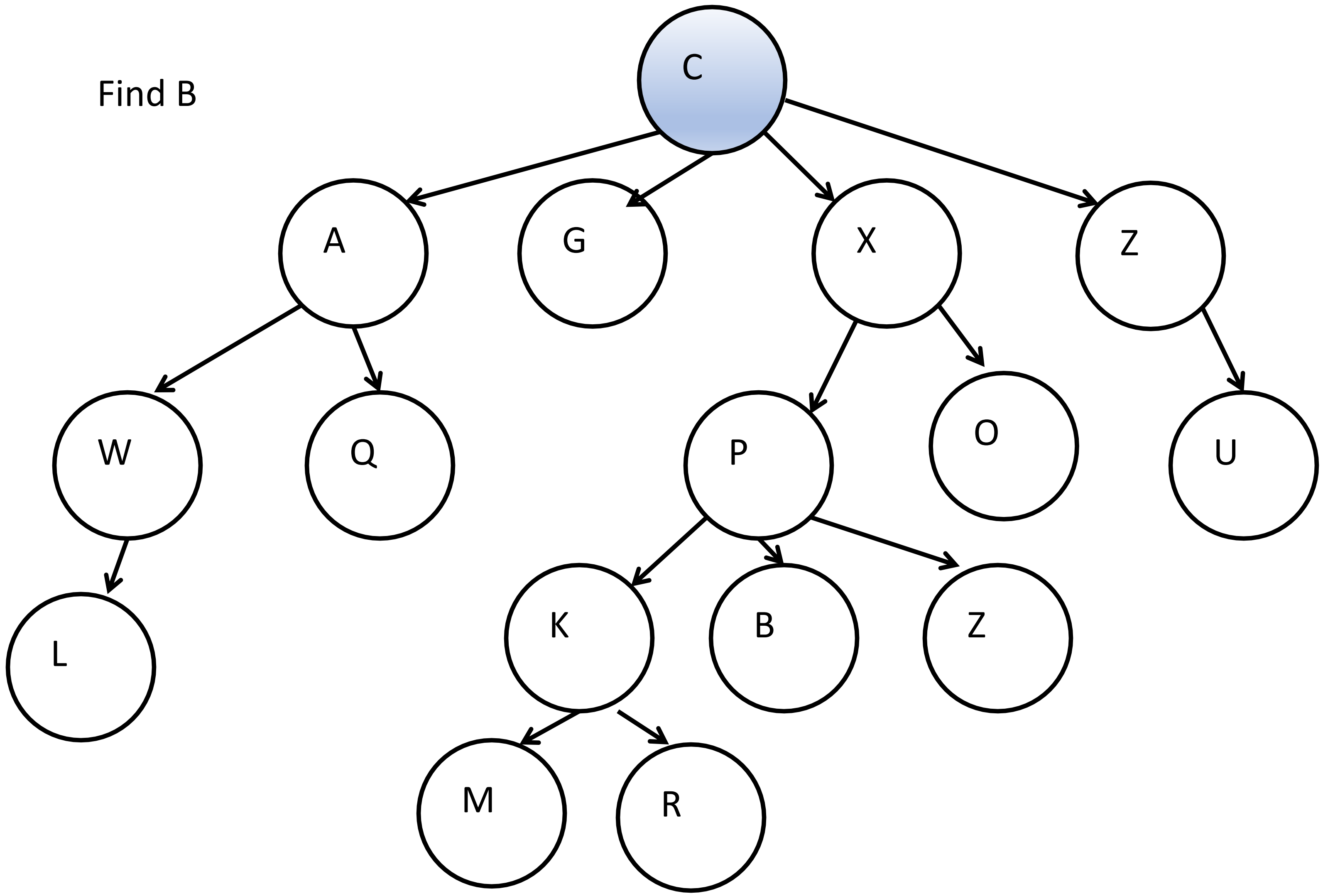
Depth First Search of Tree

Find B



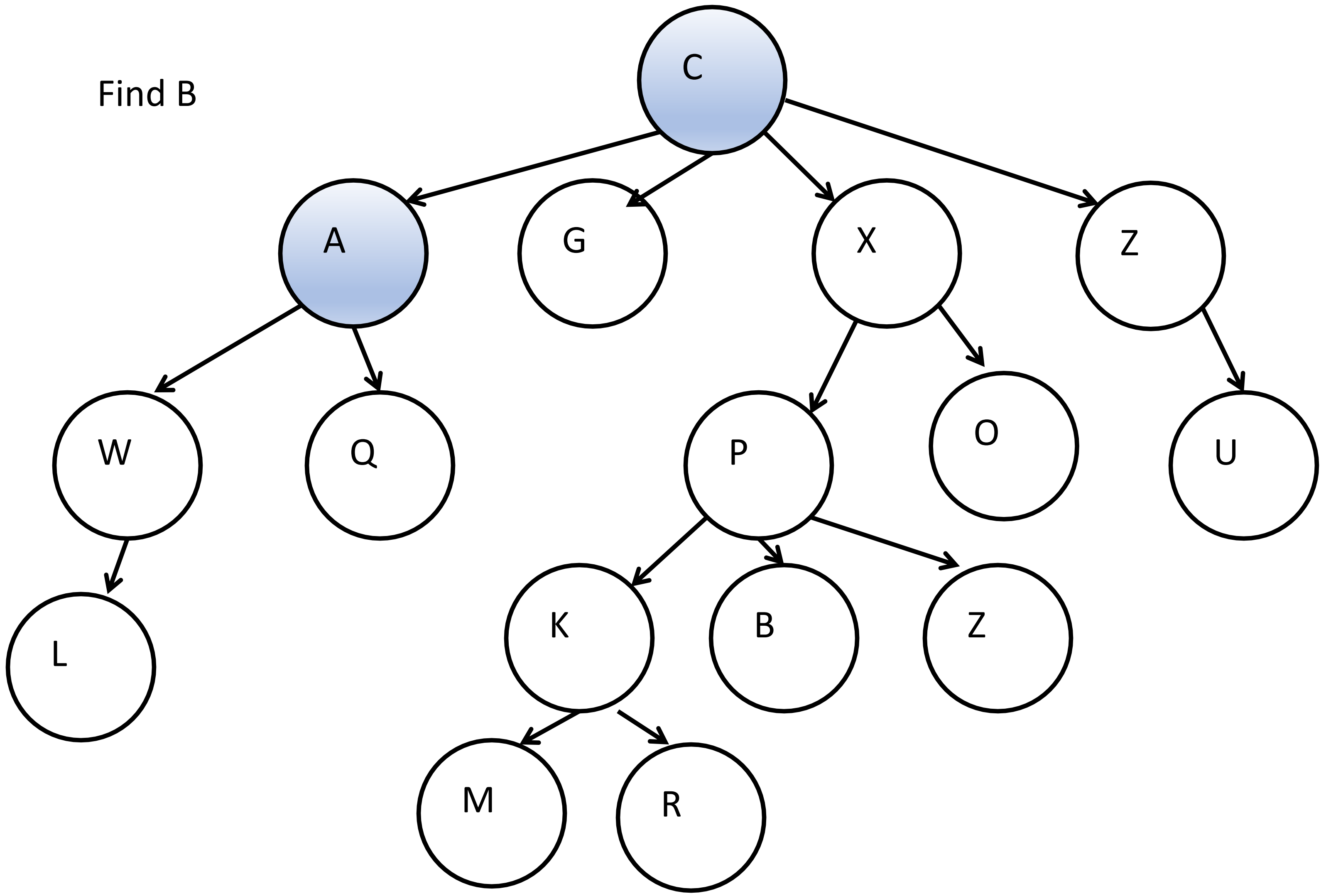
Depth First Search of Tree

Find B



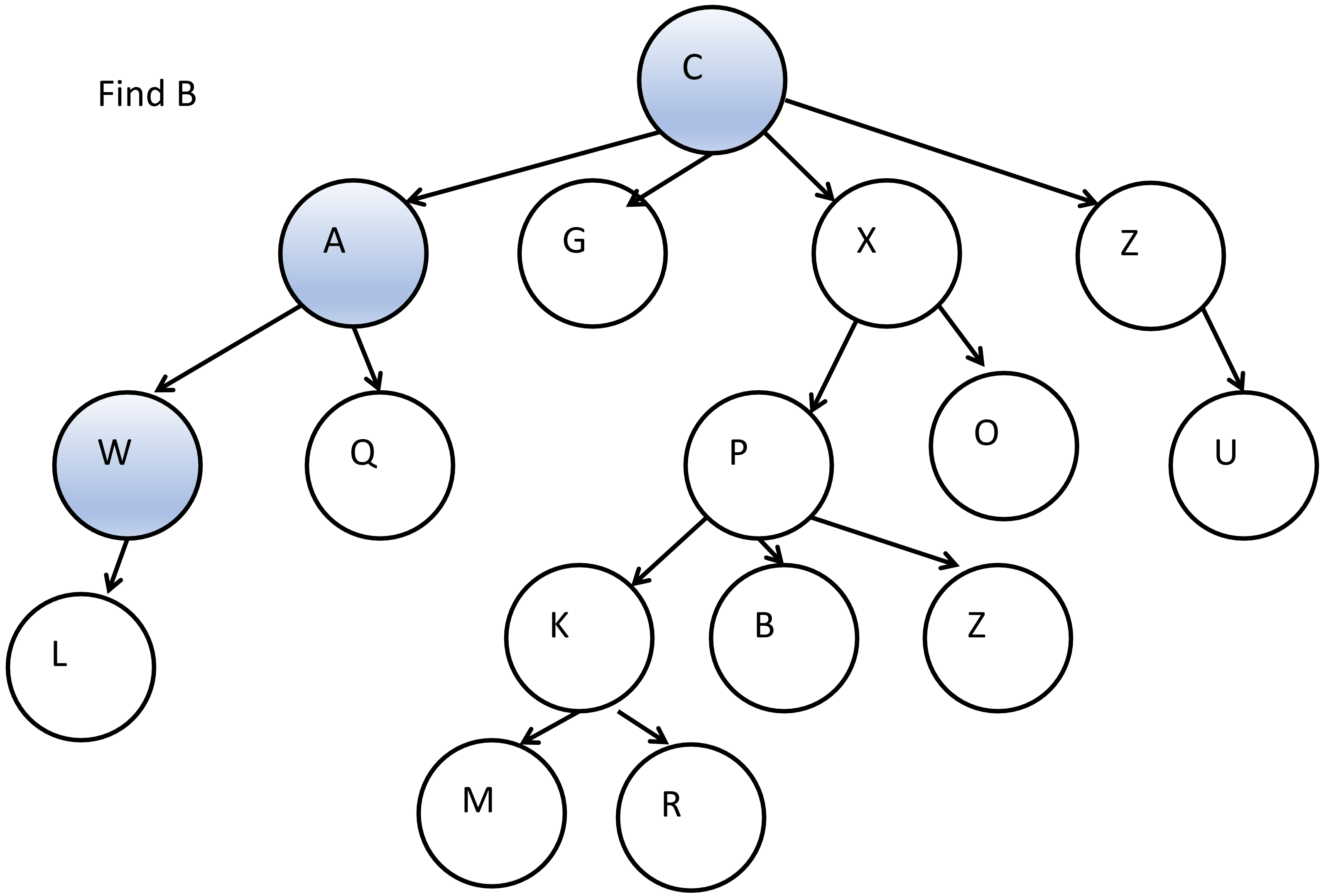
Depth First Search of Tree

Find B



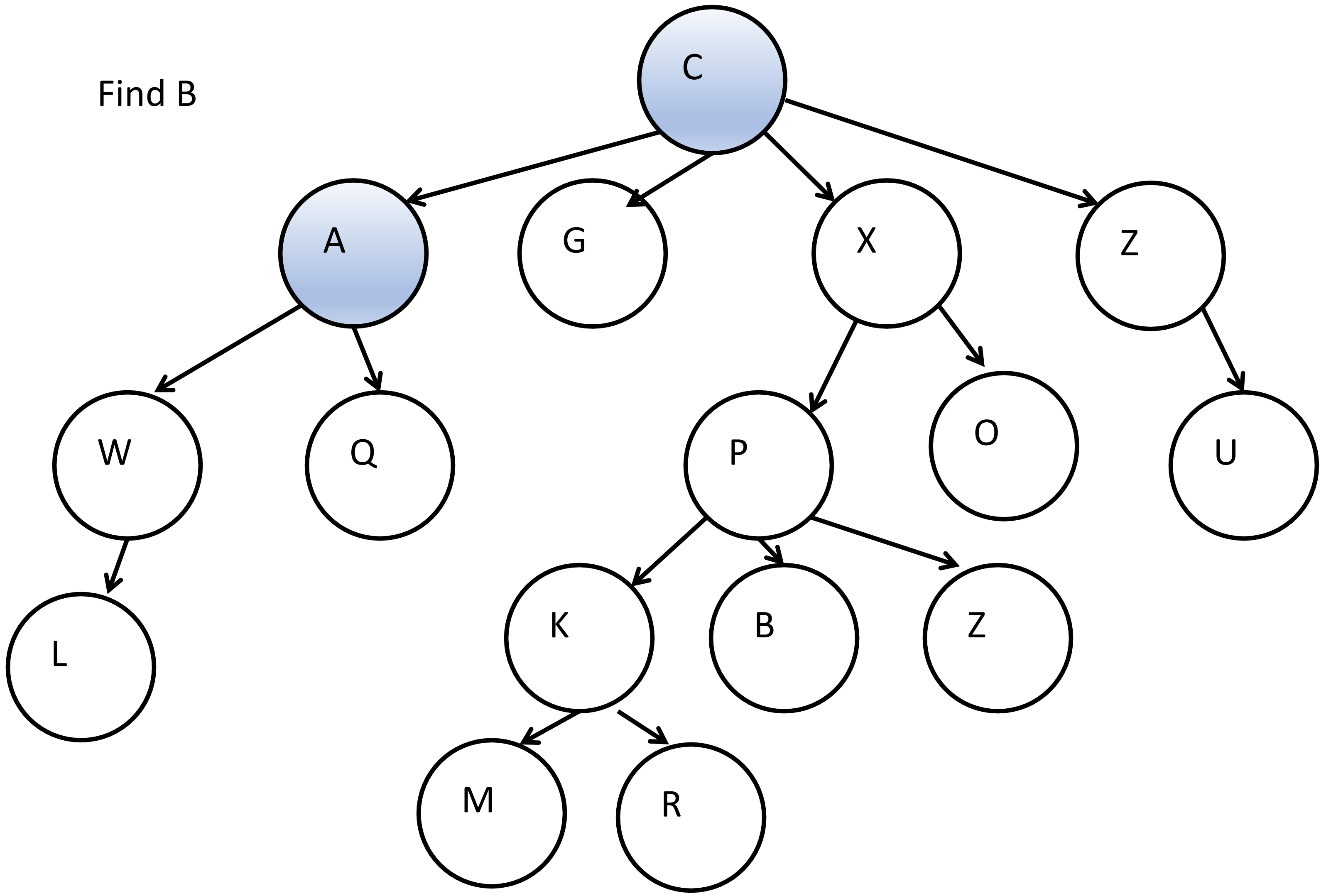
Depth First Search of Tree

Find B



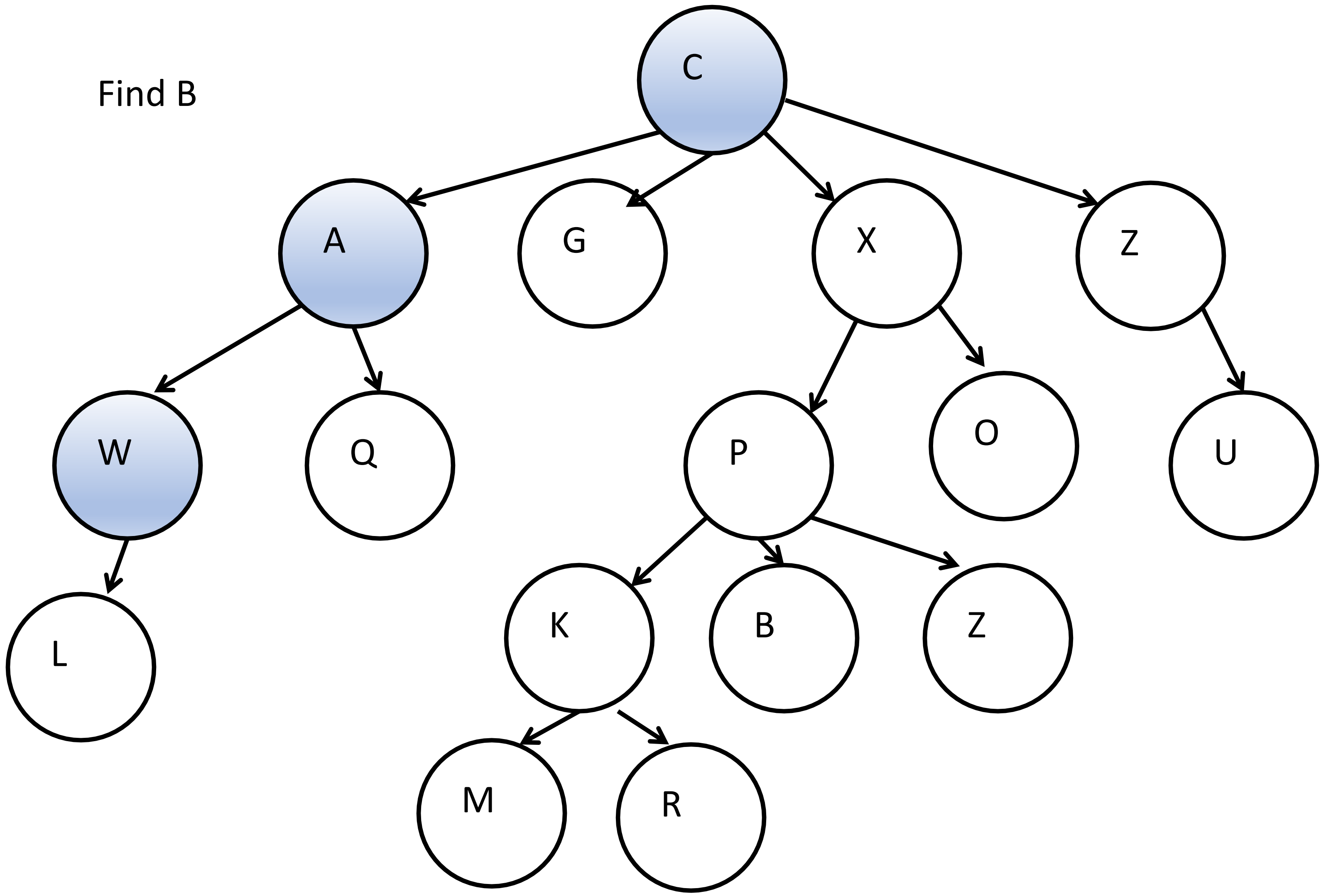
Depth First Search of Tree

Find B



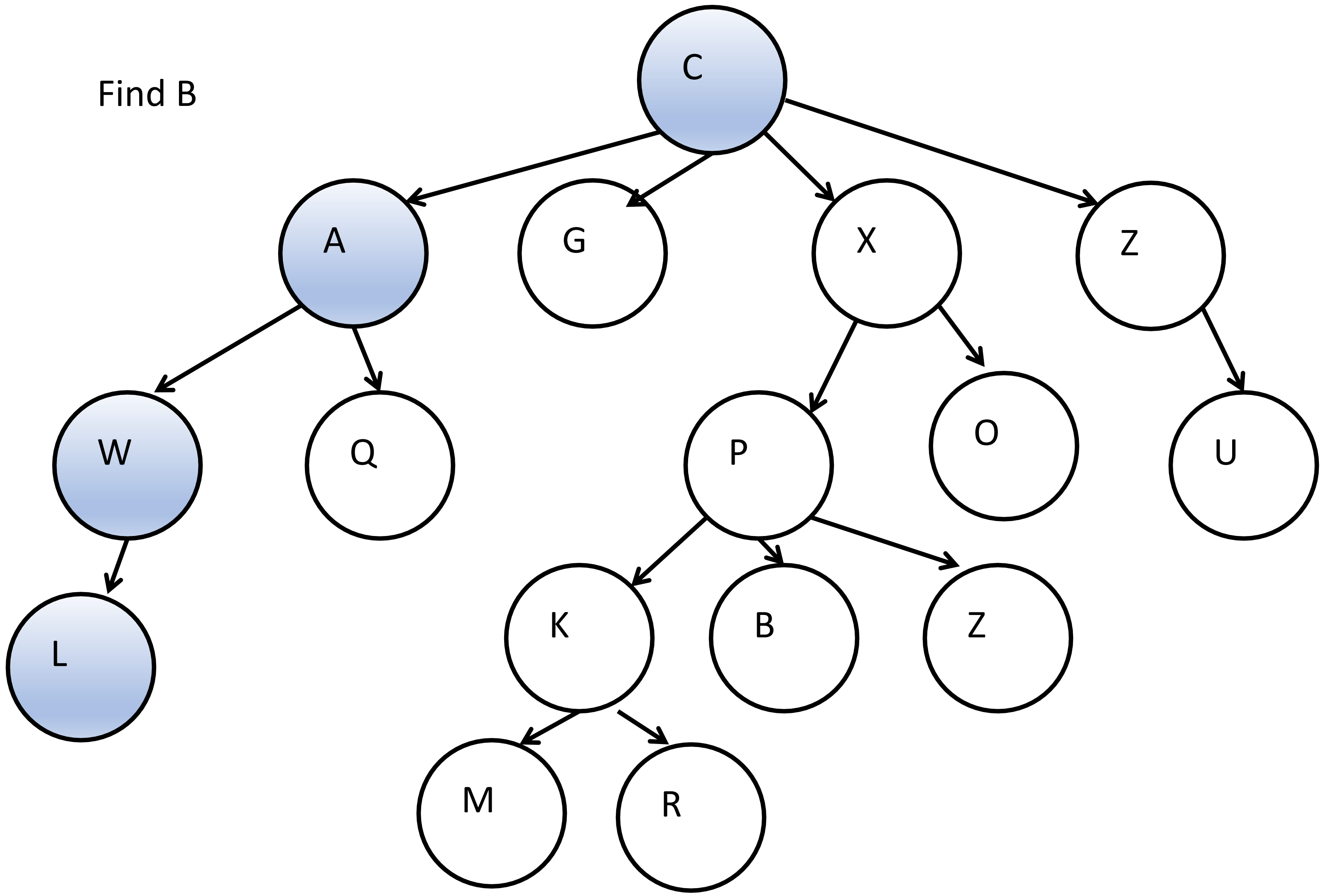
Depth First Search of Tree

Find B



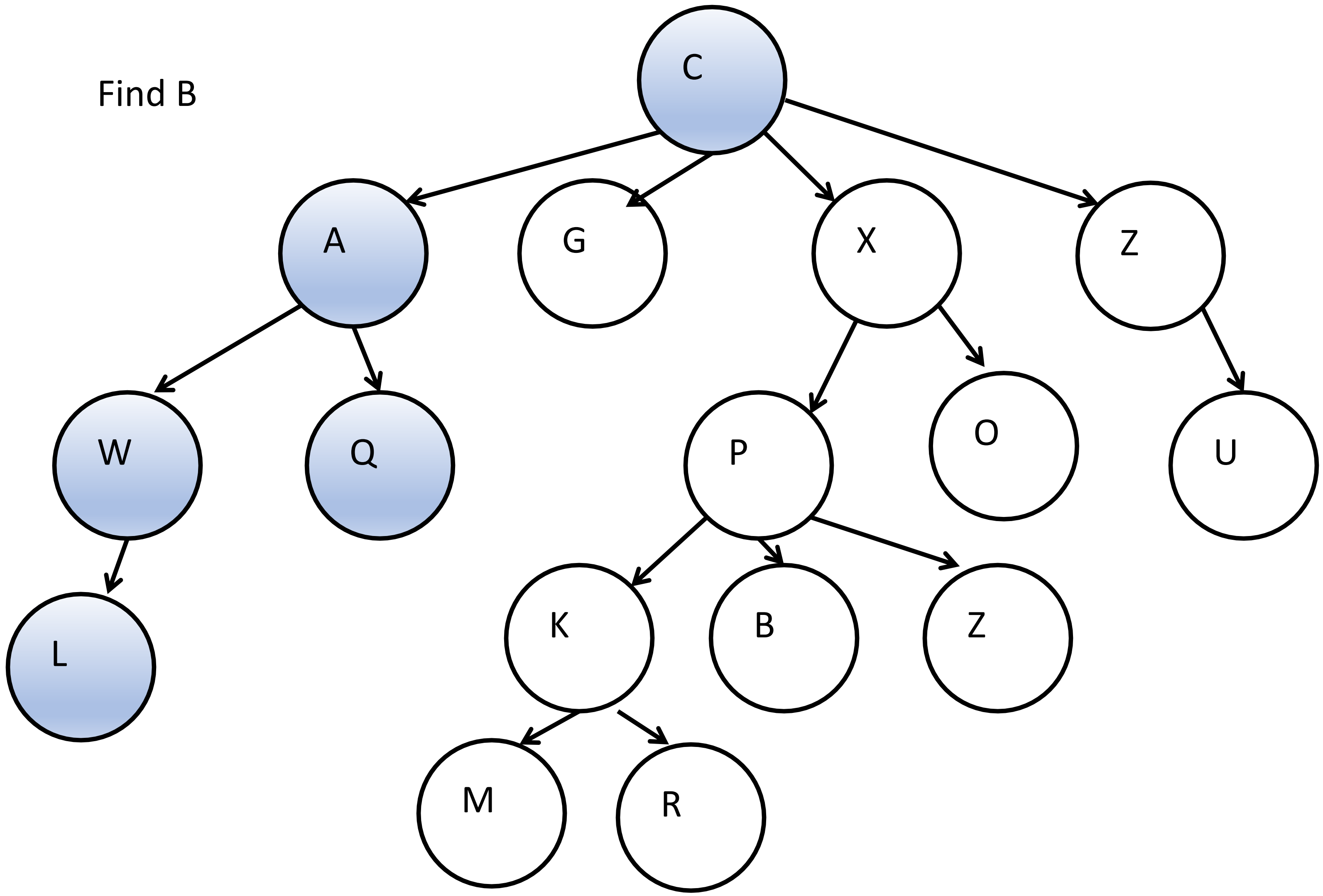
Depth First Search of Tree

Find B



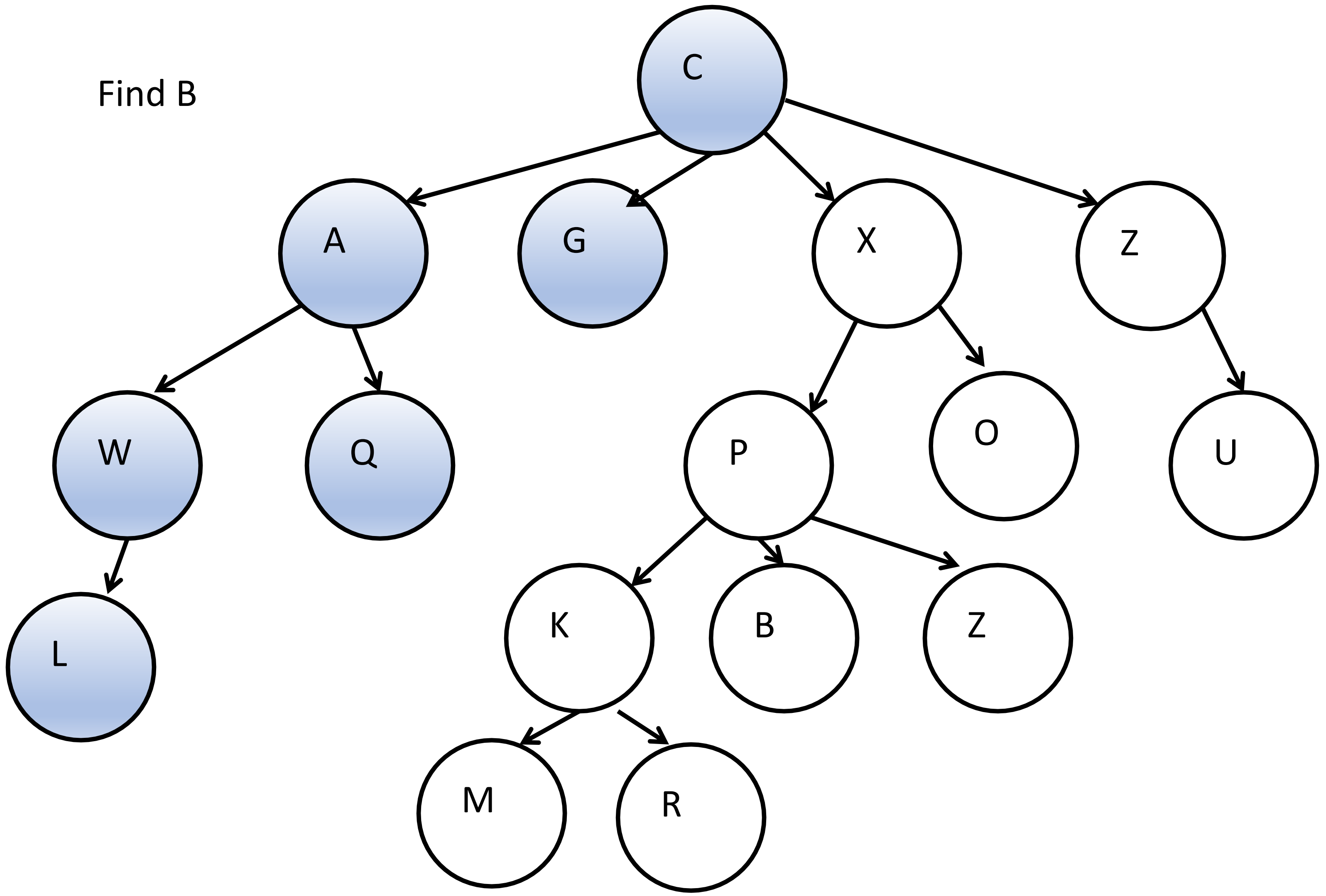
Depth First Search of Tree

Find B



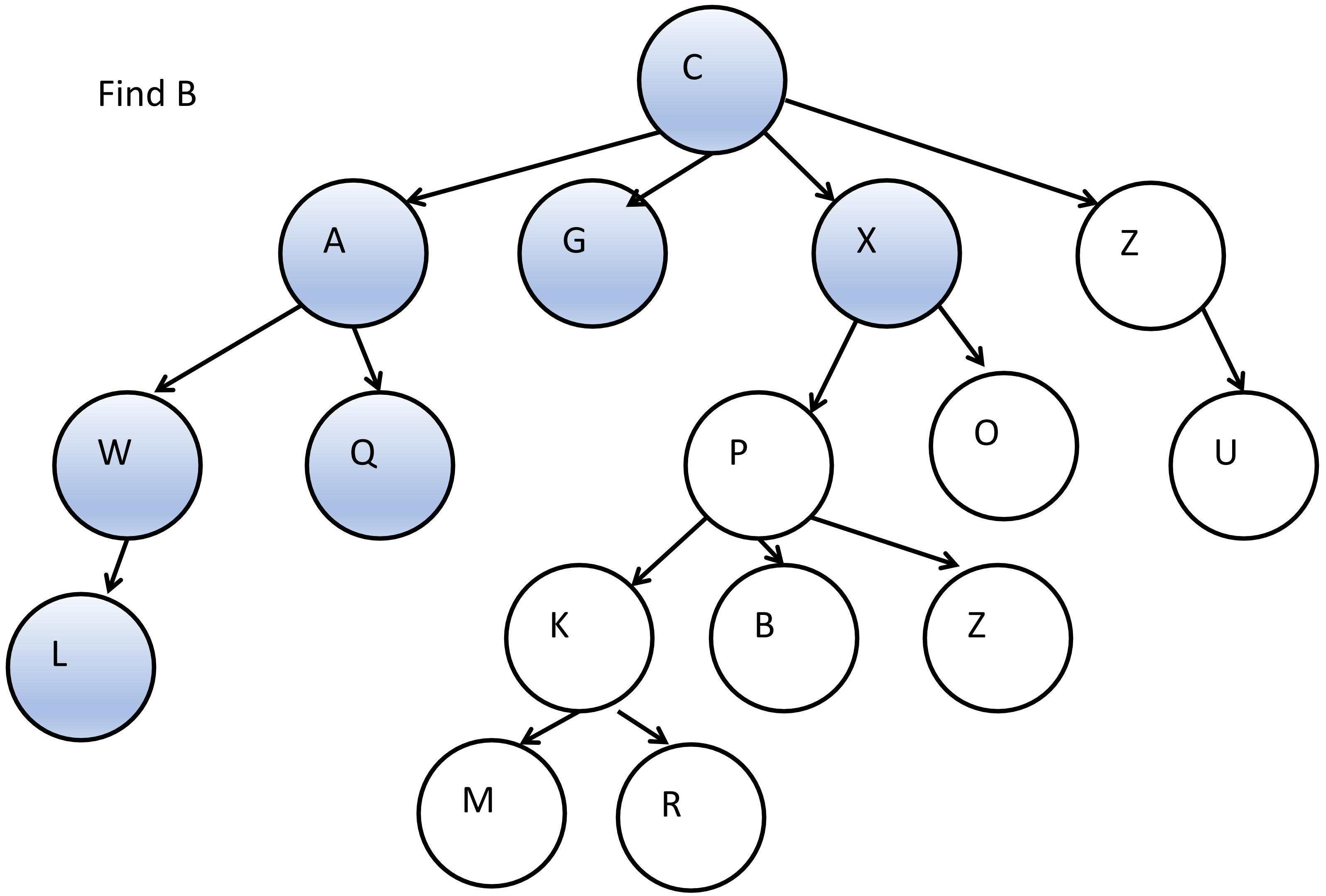
Depth First Search of Tree

Find B



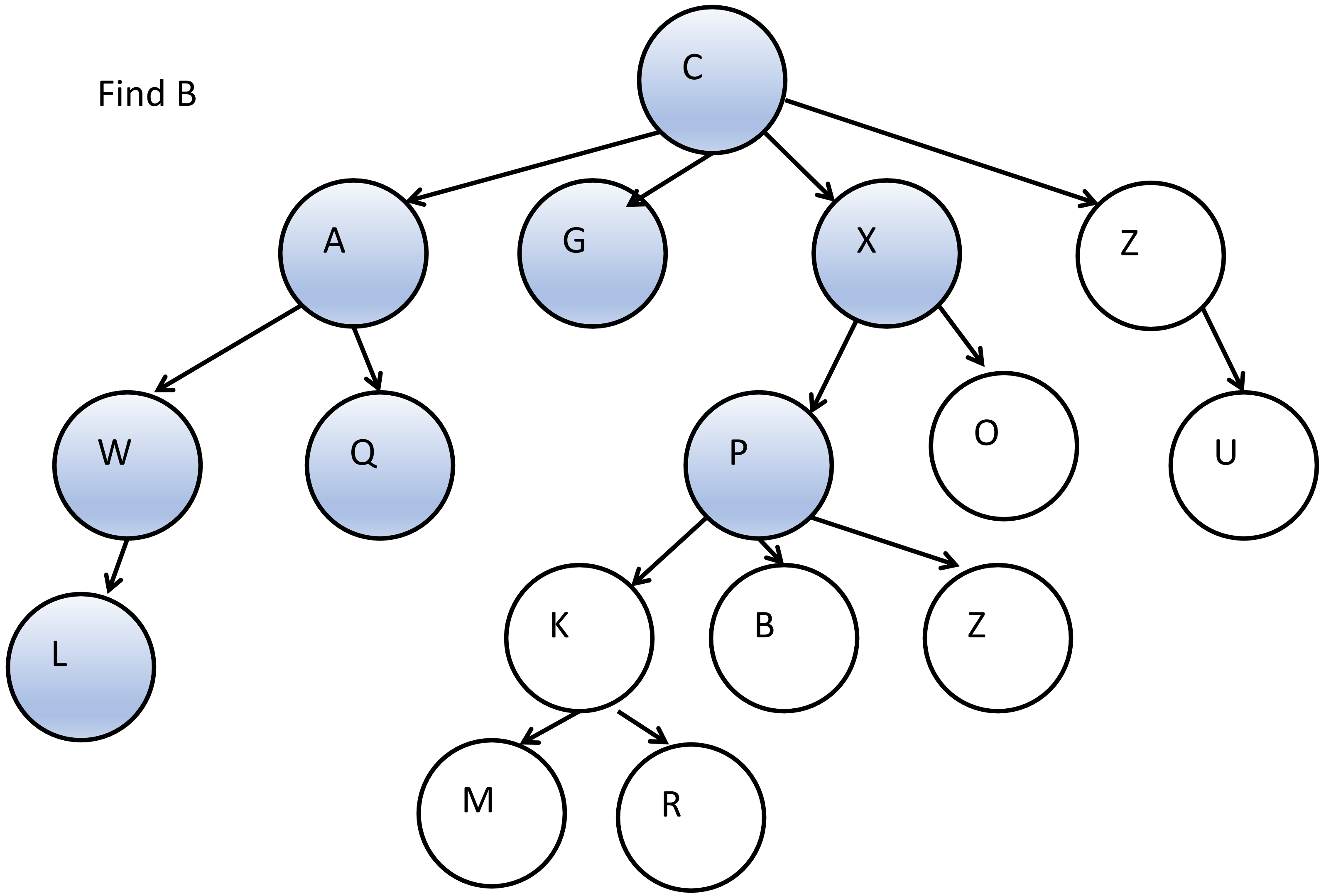
Depth First Search of Tree

Find B



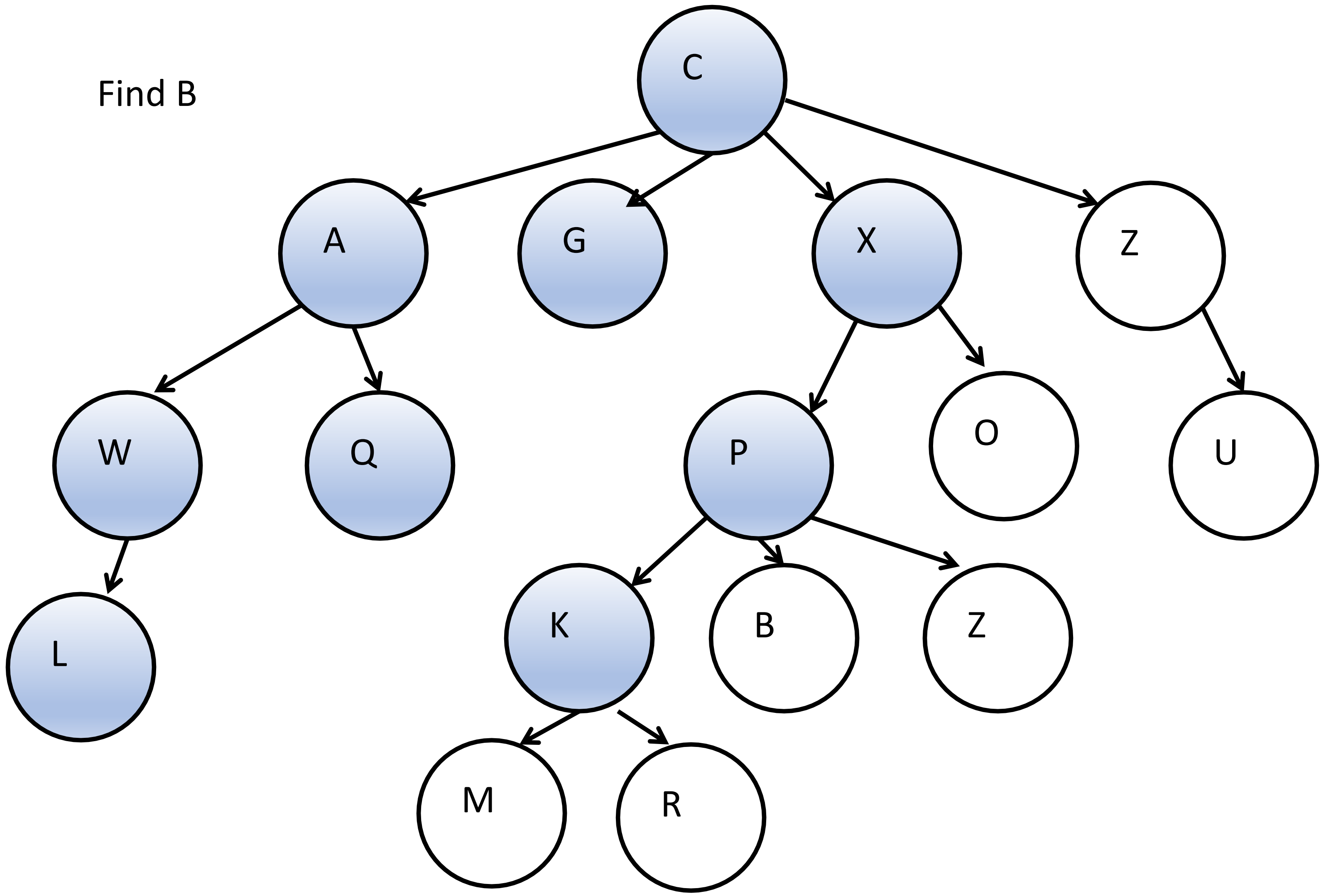
Depth First Search of Tree

Find B



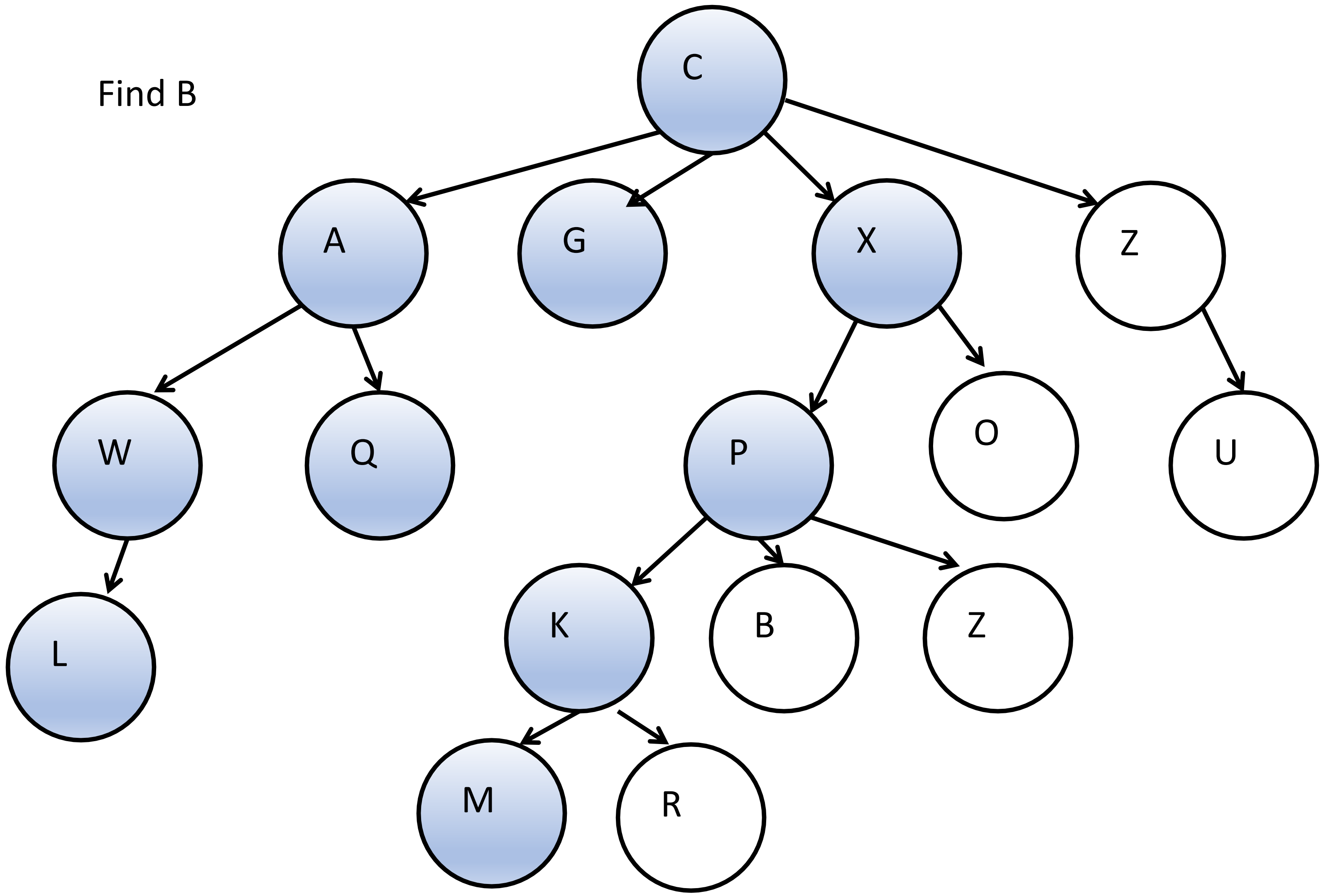
Depth First Search of Tree

Find B



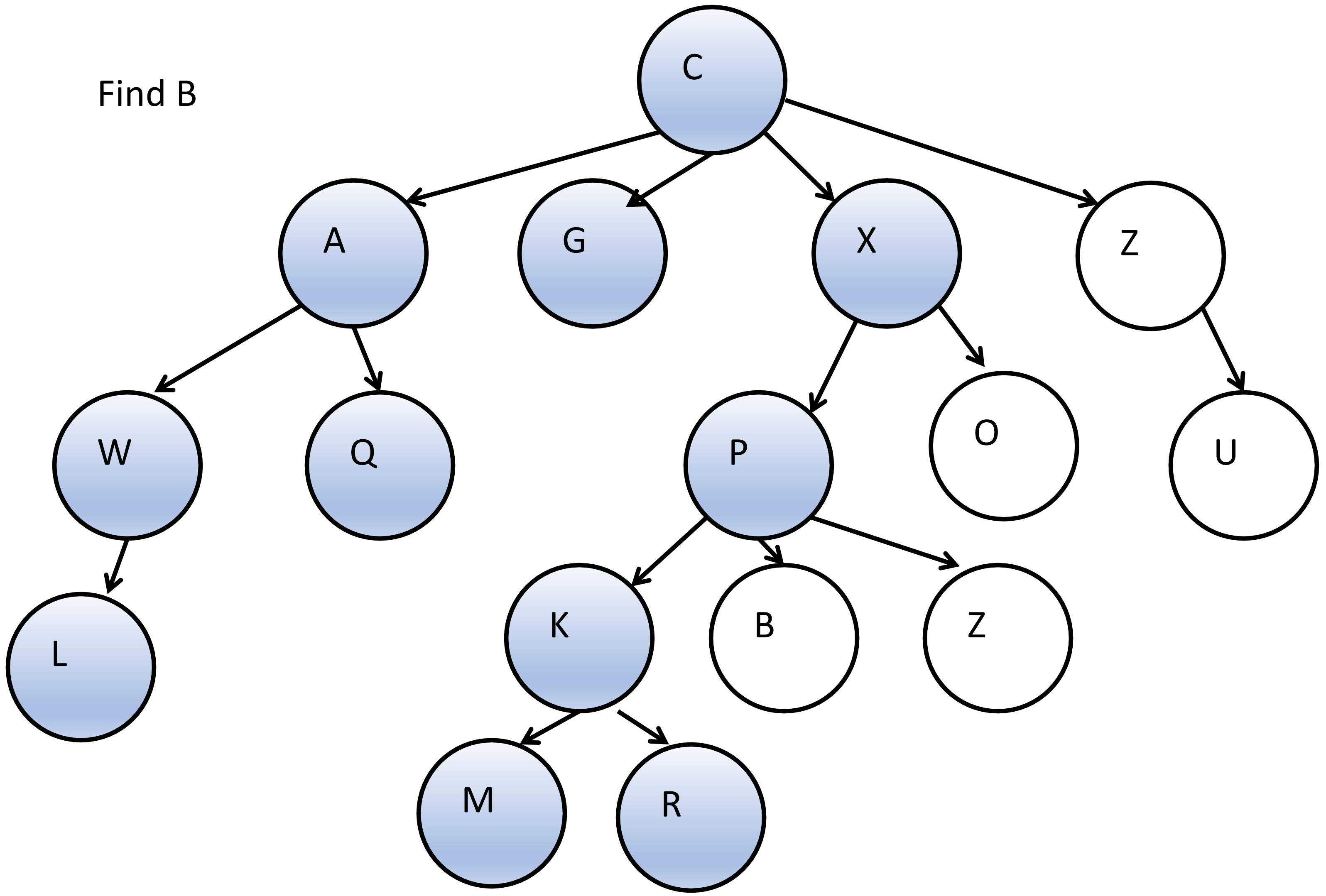
Depth First Search of Tree

Find B



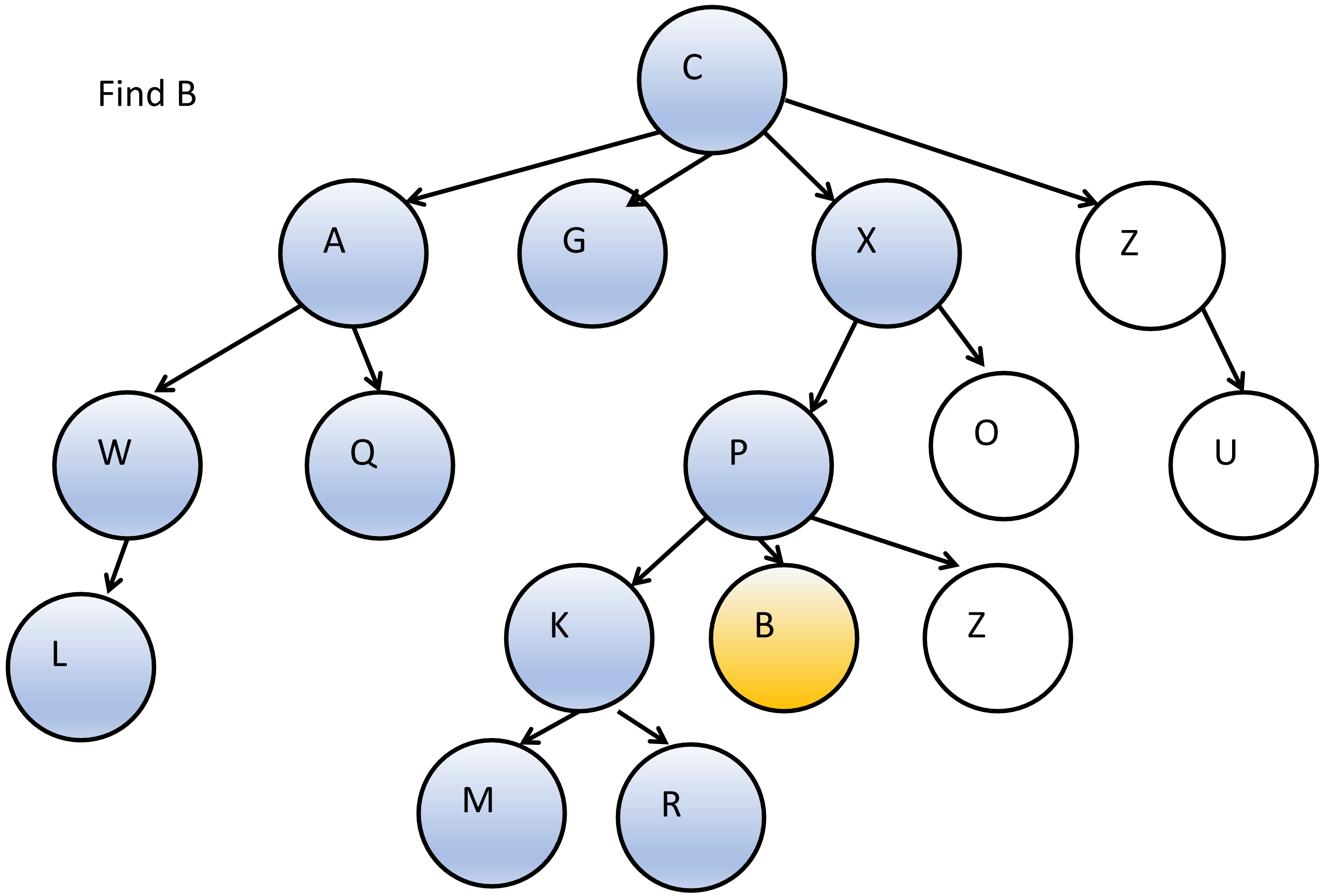
Depth First Search of Tree

Find B



Depth First Search of Tree

Find B



How to do depth-first searching

-
  ```
  Put the root node on a stack;
  while (stack is not empty) {
      remove a node from the stack;
      if (node is a goal node) return success;
      put all children of node onto the stack;
  }
  return failure;
  ```

- BFS is much worse memory-wise
  - BFS may store the whole search space
- In general
  - **BFS** is better if **goal is not deep**, if **infinite paths**, if many **loops**, if **small search space**
  - **DFS** is better if **many goals**, not many loops,
  - **DFS** is much better in terms of **memory**

Comparison of algorithms

- Depth-first searching:

  -
    ```
    Put the root node on a stack;
    while (stack is not empty) {
        remove a node from the stack;
        if (node is a goal node) return success;
        put all children of node onto the stack;
    }
    return failure;
    ```

- Breadth-first searching:

  -
    ```
    Put the root node on a queue;
    while (queue is not empty) {
        remove a node from the queue;
        if (node is a goal node) return success;
        put all children of node onto the queue;
    }
    return failure;
    ```

Depth- vs. breadth-first searching

- They differ in the order in which they visit nodes
- When a breadth-first search succeeds, it finds a minimum-depth (nearest the root) goal node
- When a depth-first search succeeds, the found goal node is not necessarily minimum depth
- For a large tree, breadth-first search memory requirements may be excessive
- For a large tree, a depth-first search may take an excessively long time to find even a very nearby goal node
- How can we combine the advantages (and avoid the disadvantages) of these two search techniques?

More on search techniques

- The searches we have been doing are blind searches, in which we have no prior information to help guide the search
  - If we have some measure of "how close" we are to a goal node, we can employ much more sophisticated search techniques
  - We will *not* cover these more sophisticated techniques