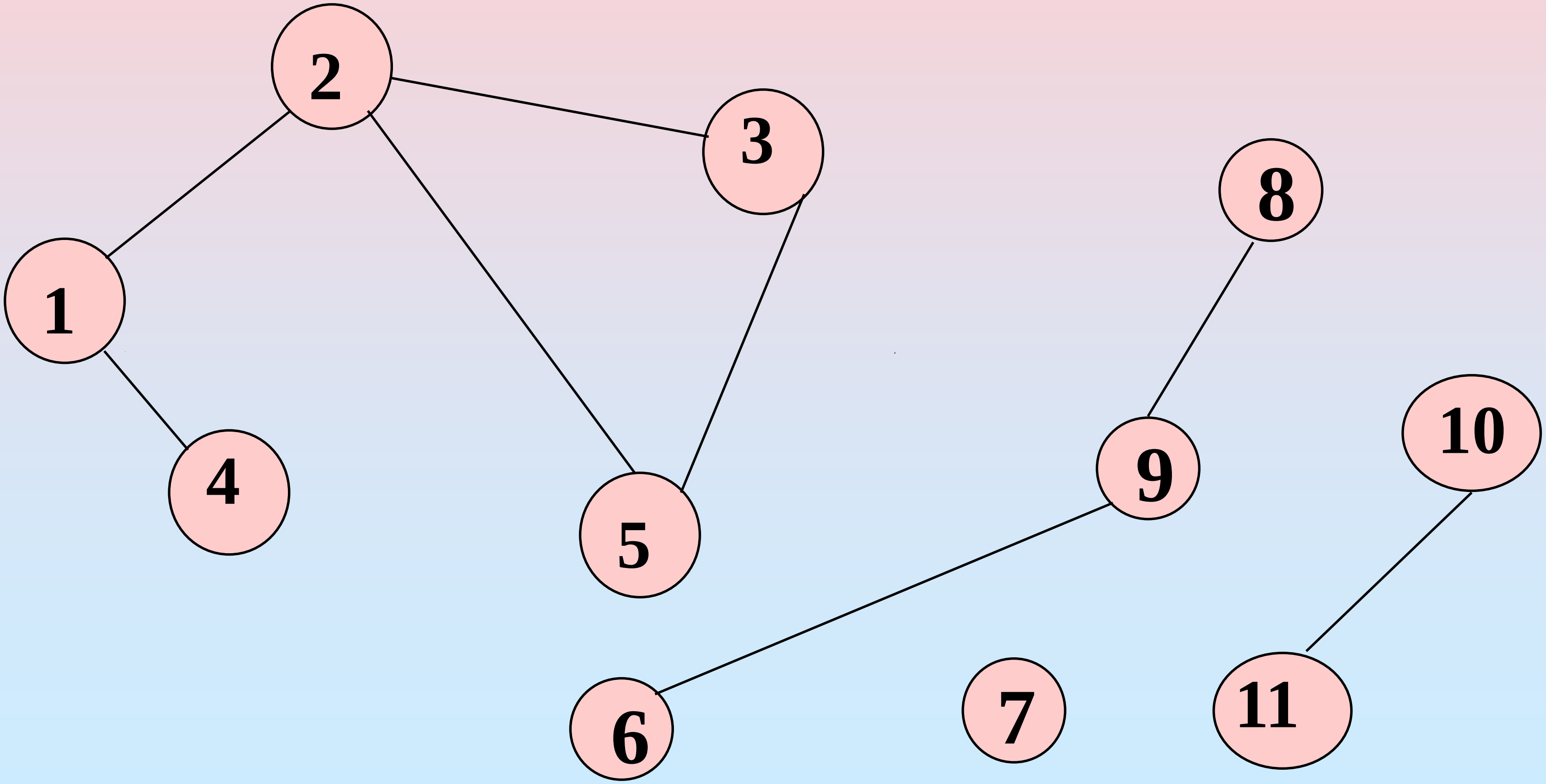
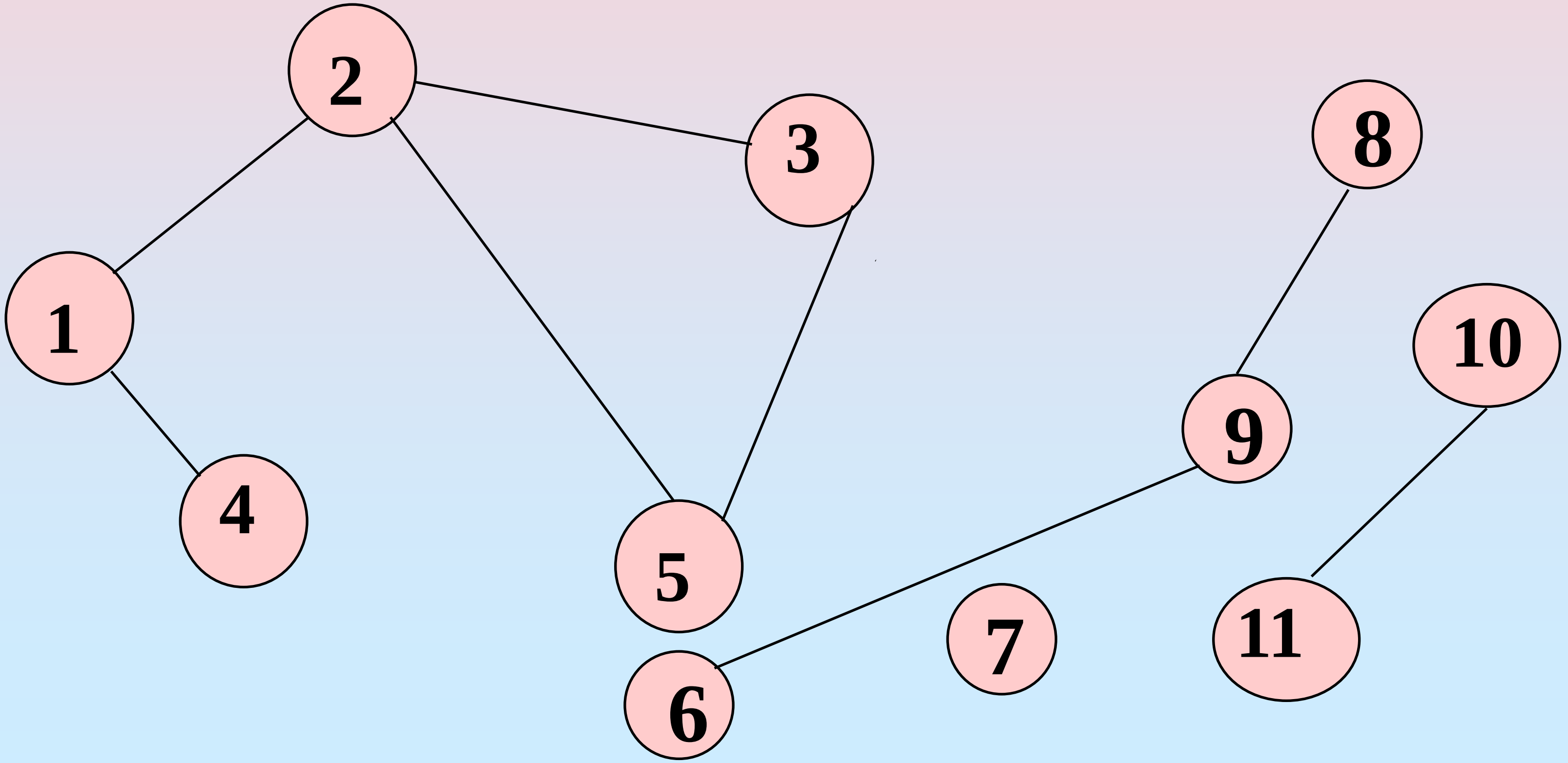
Graph Search Methods

- A vertex **u** is reachable from vertex **v** iff there is a path from **v** to **u**.



Graph Search Methods

- A search method starts at a given vertex v and visits/labels/marks every vertex that is reachable from v.



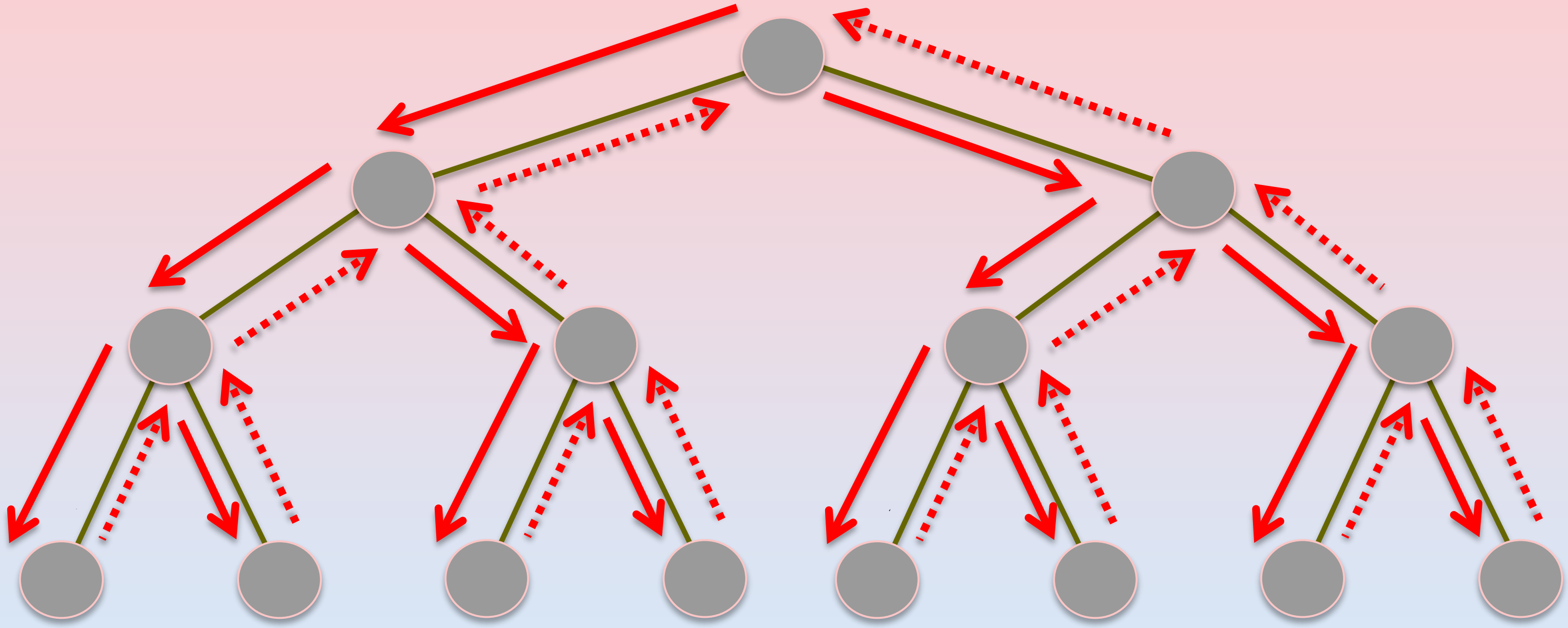
Graph Search Methods

- **Many graph problems are solved using a search method.**
  - **Path from one vertex to another.**
  - **Is the graph connected?**
  - **Find a spanning tree, etc.**
- **Commonly used search methods:**
  - **Depth-first search(DFS).**
  - **Breadth-first search(BFS).**

Depth-First Search

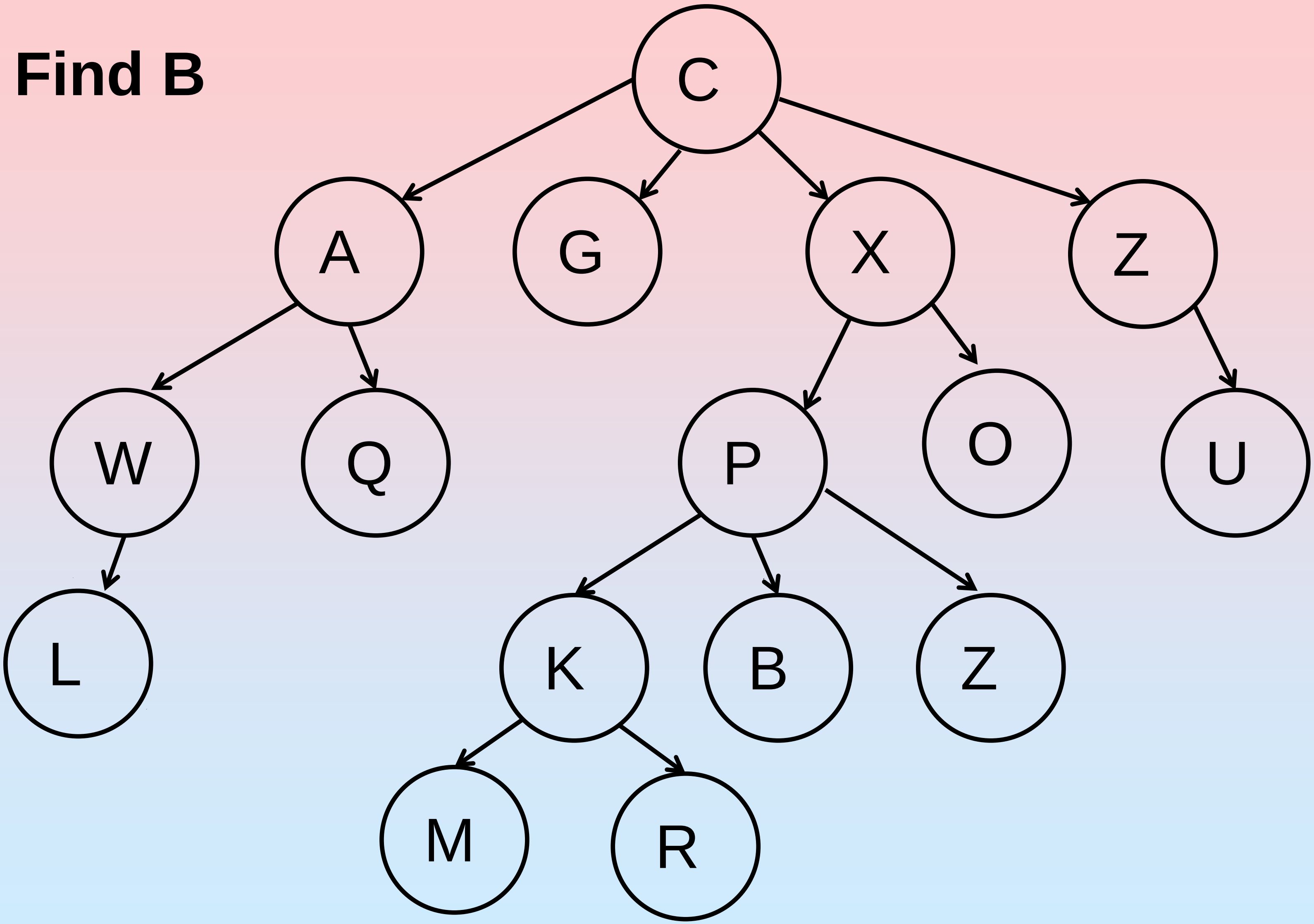
```
dfs ( v )
{  Label vertex v as reached.
   for (each unreached vertex u
                        adjacenct from v )
      dfs( u );
}
```

Depth-First Search (DFS) in a Binary Tree



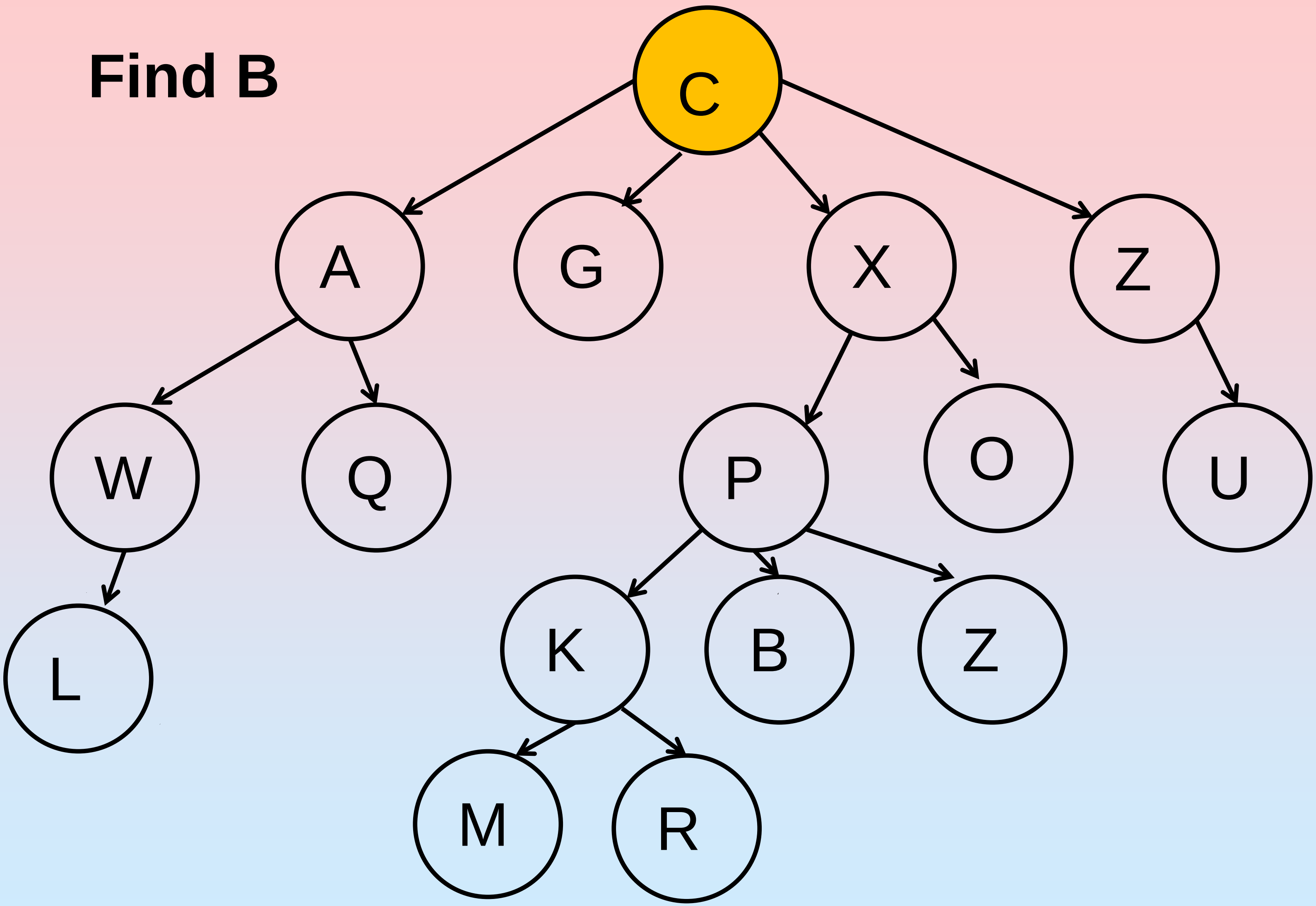
Depth First Search of Tree

Find B



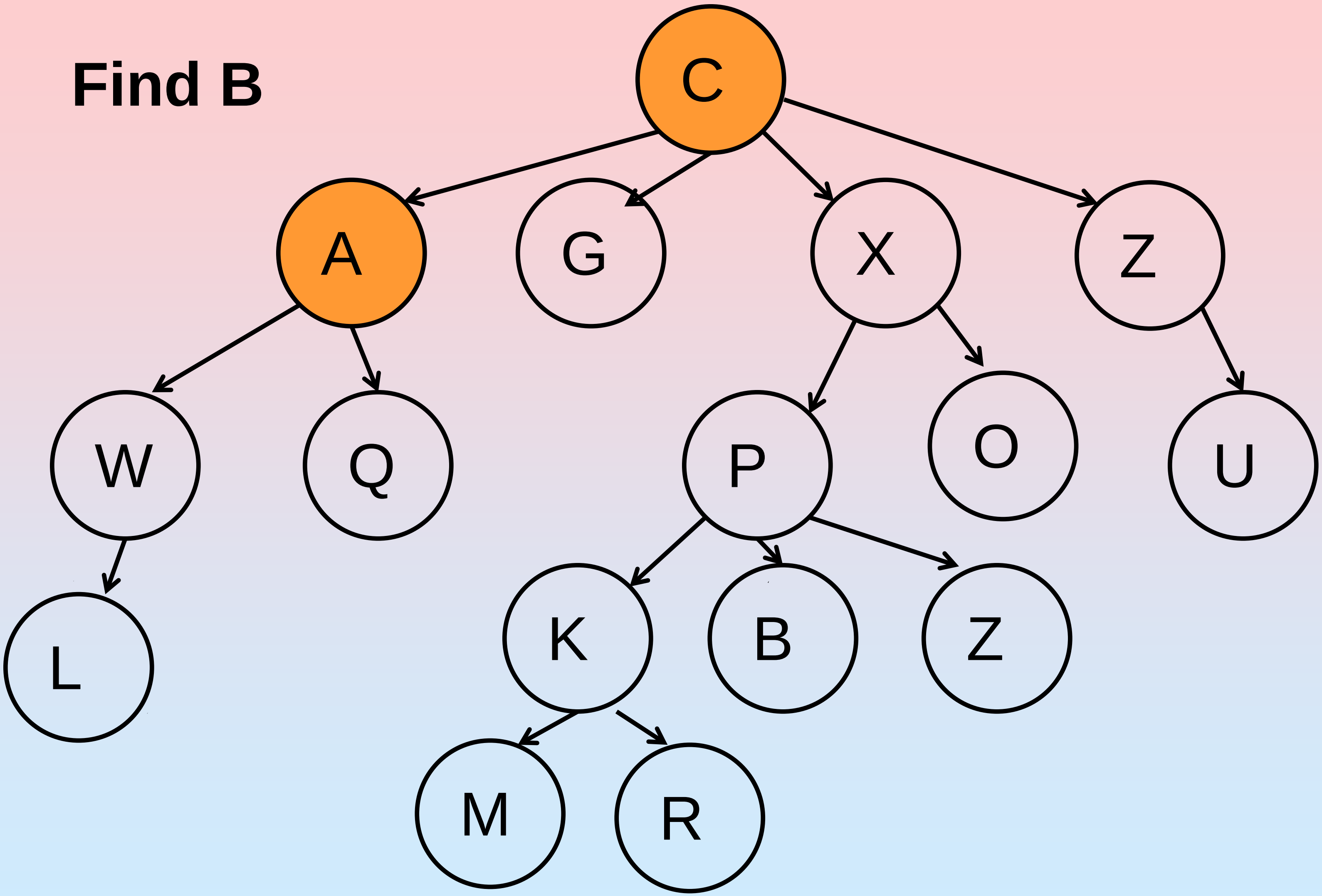
Depth First Search of Tree

Find B



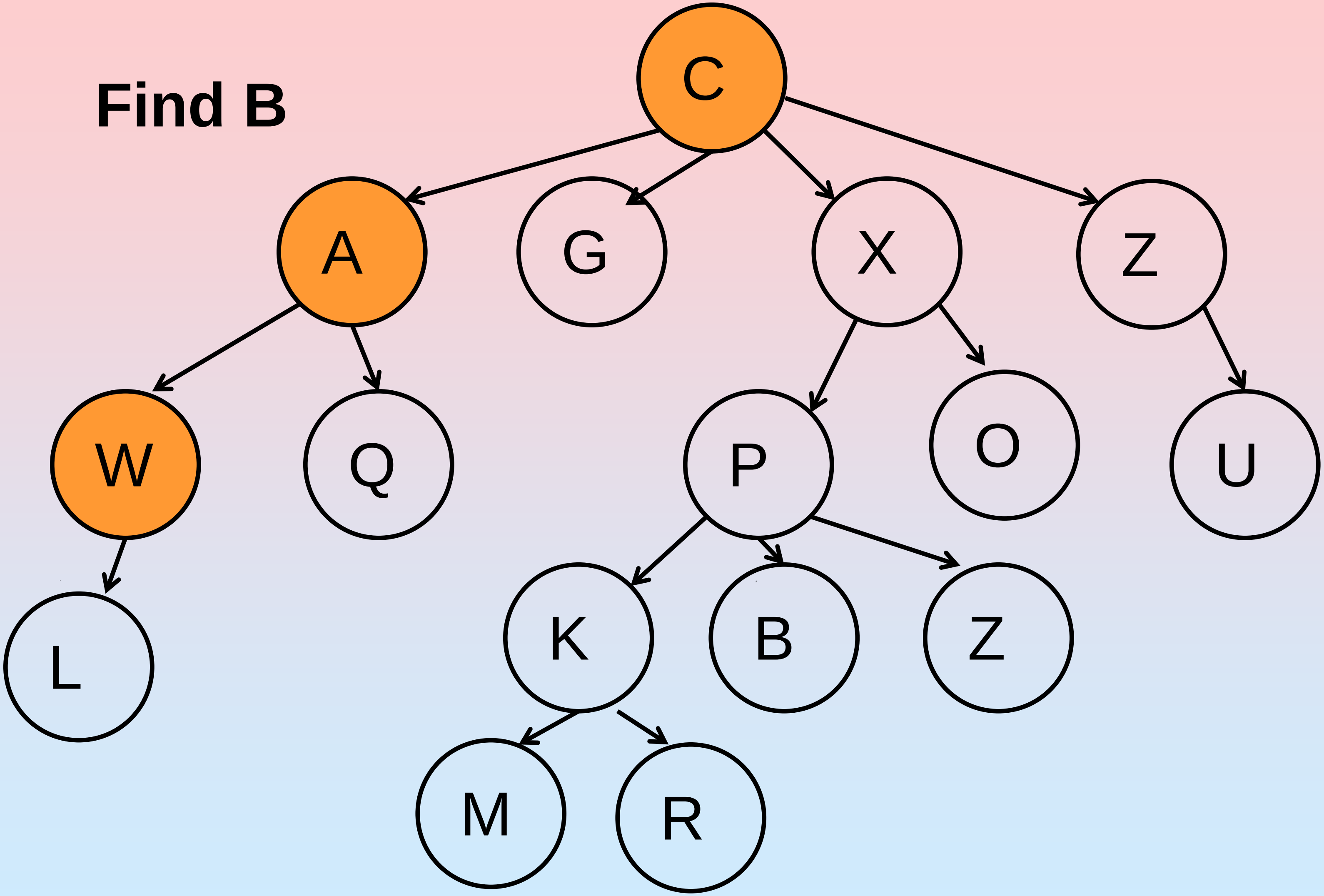
Depth First Search of Tree

Find B

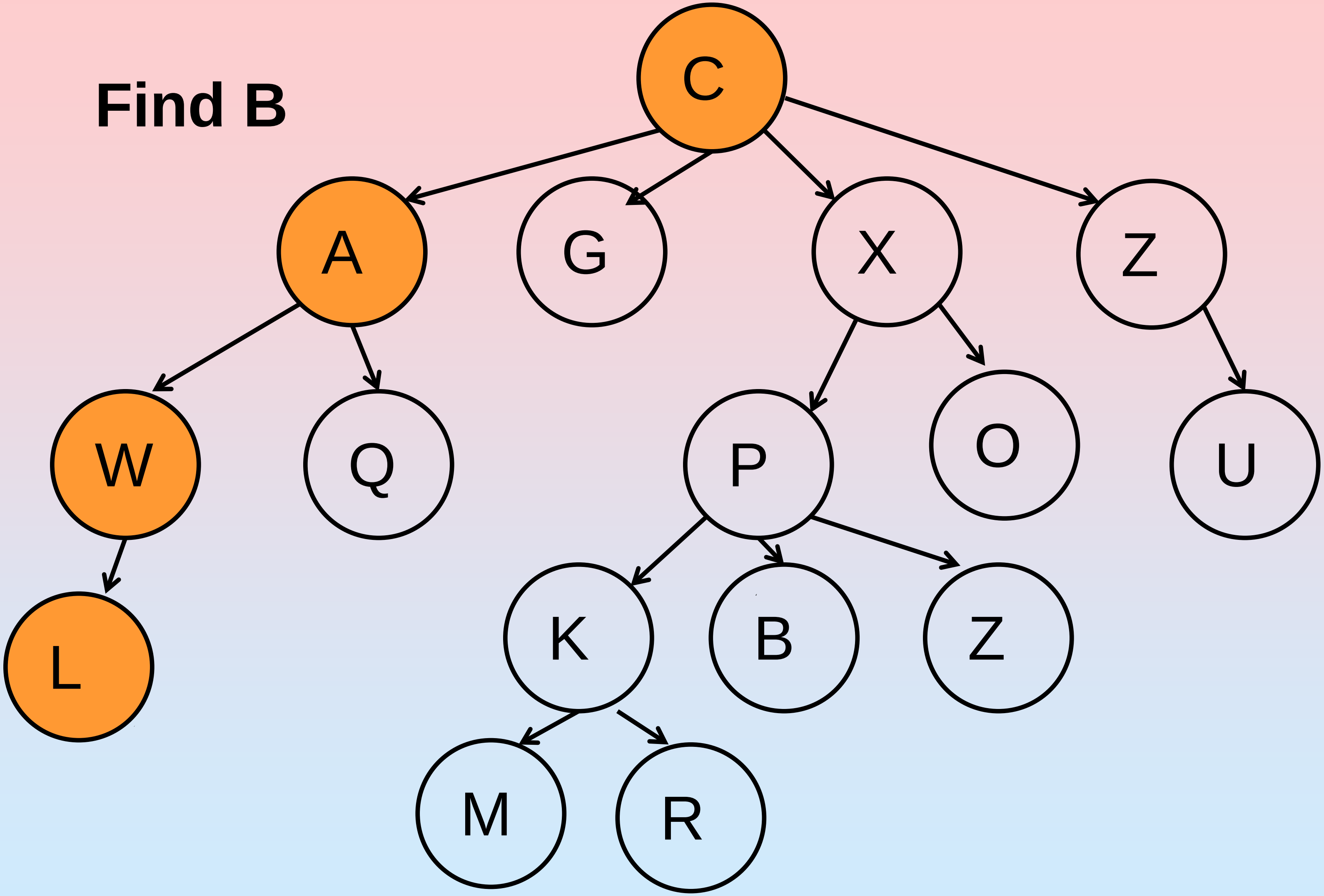


Depth First Search of Tree

Find B

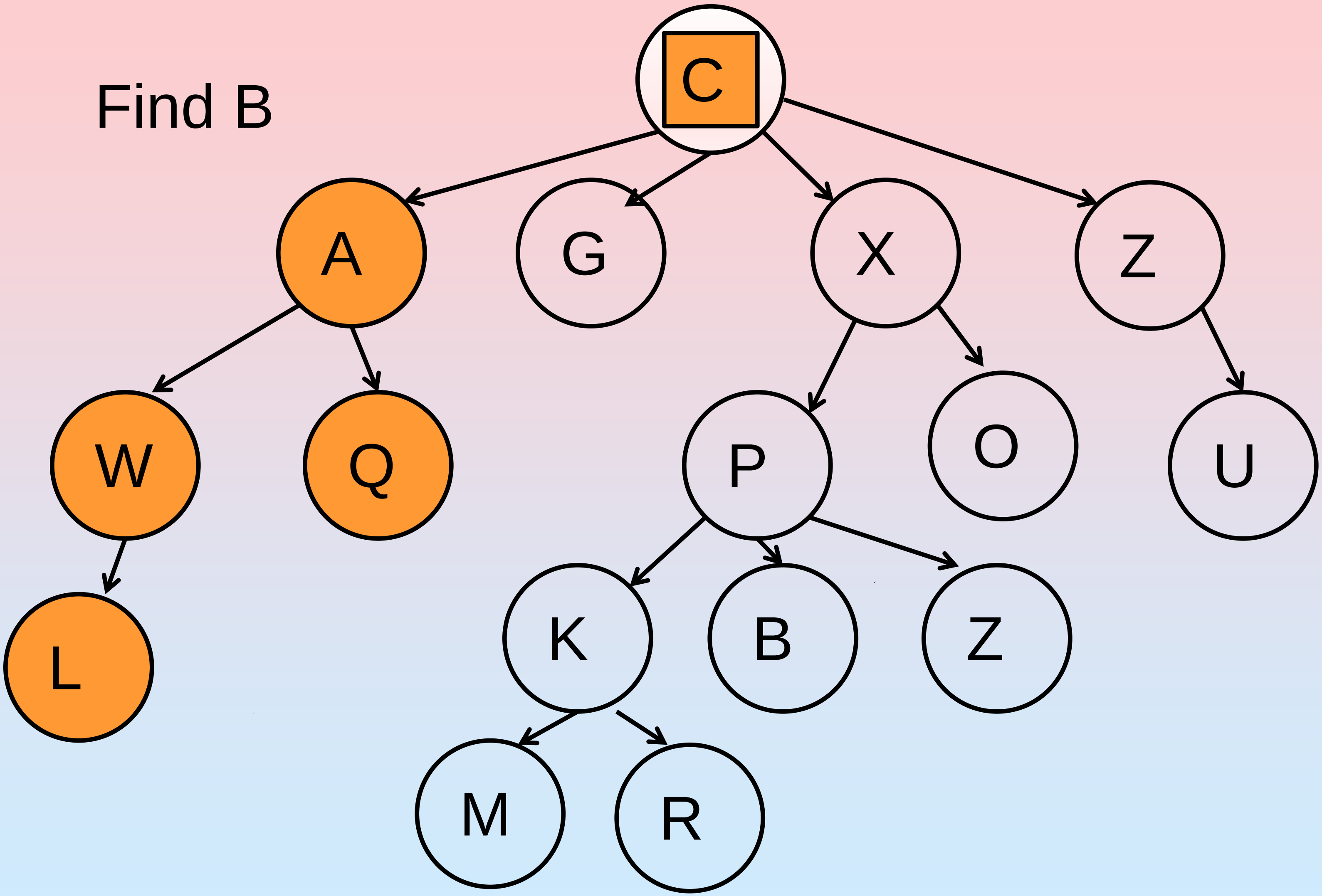


Depth First Search of Tree



Depth First Search of Tree

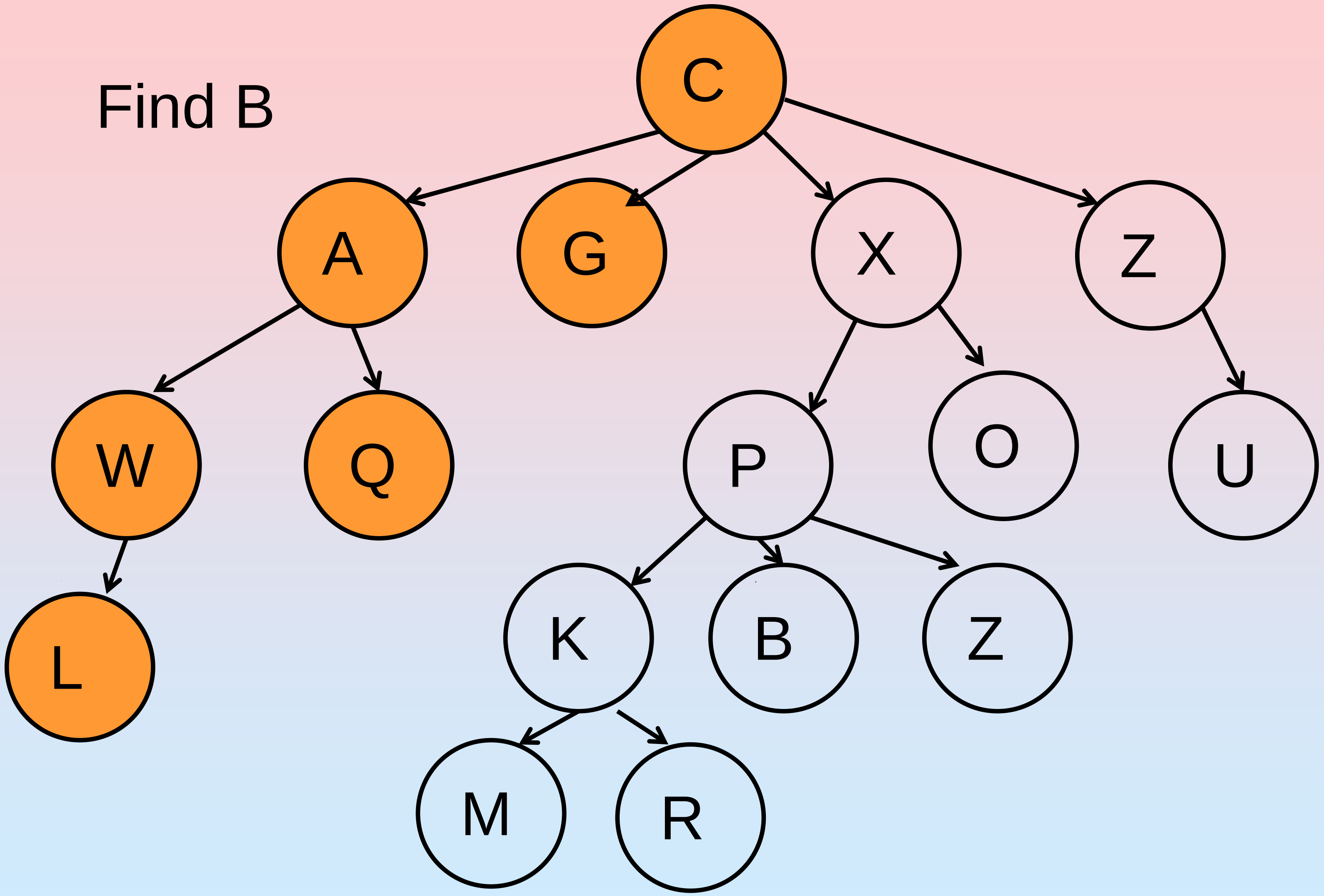
Find B



11

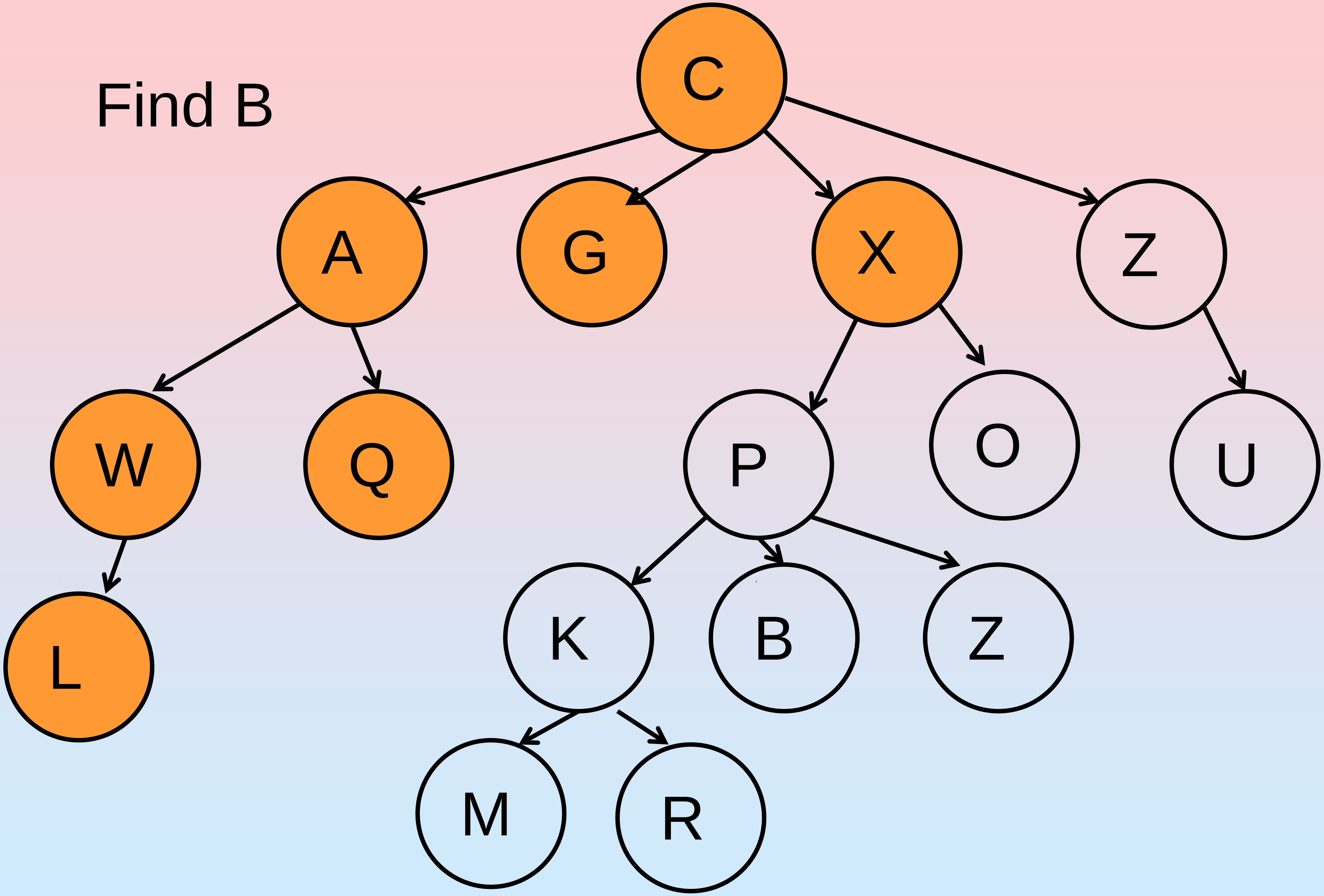
Depth First Search of Tree

Find B



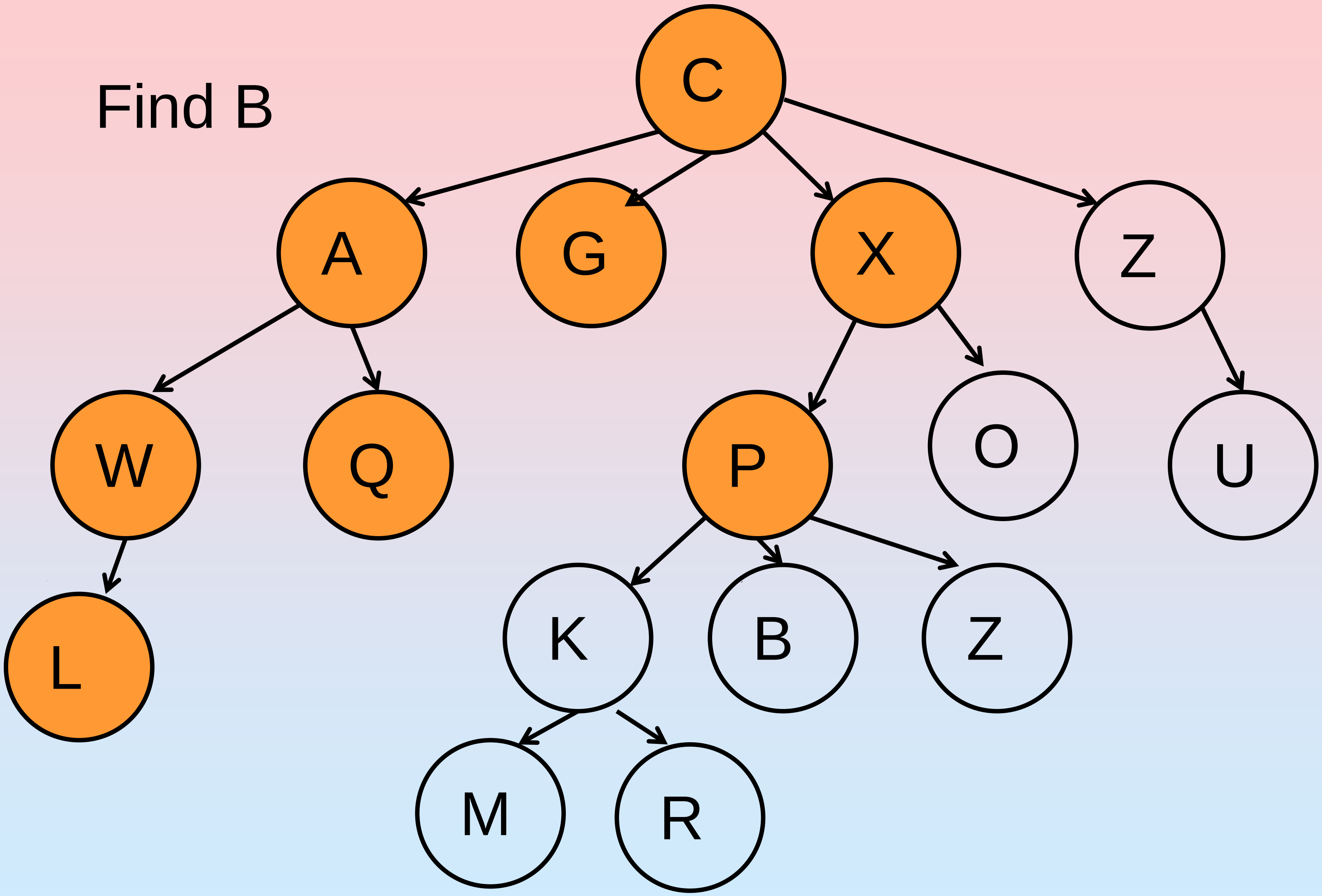
Depth First Search of Tree

Find B



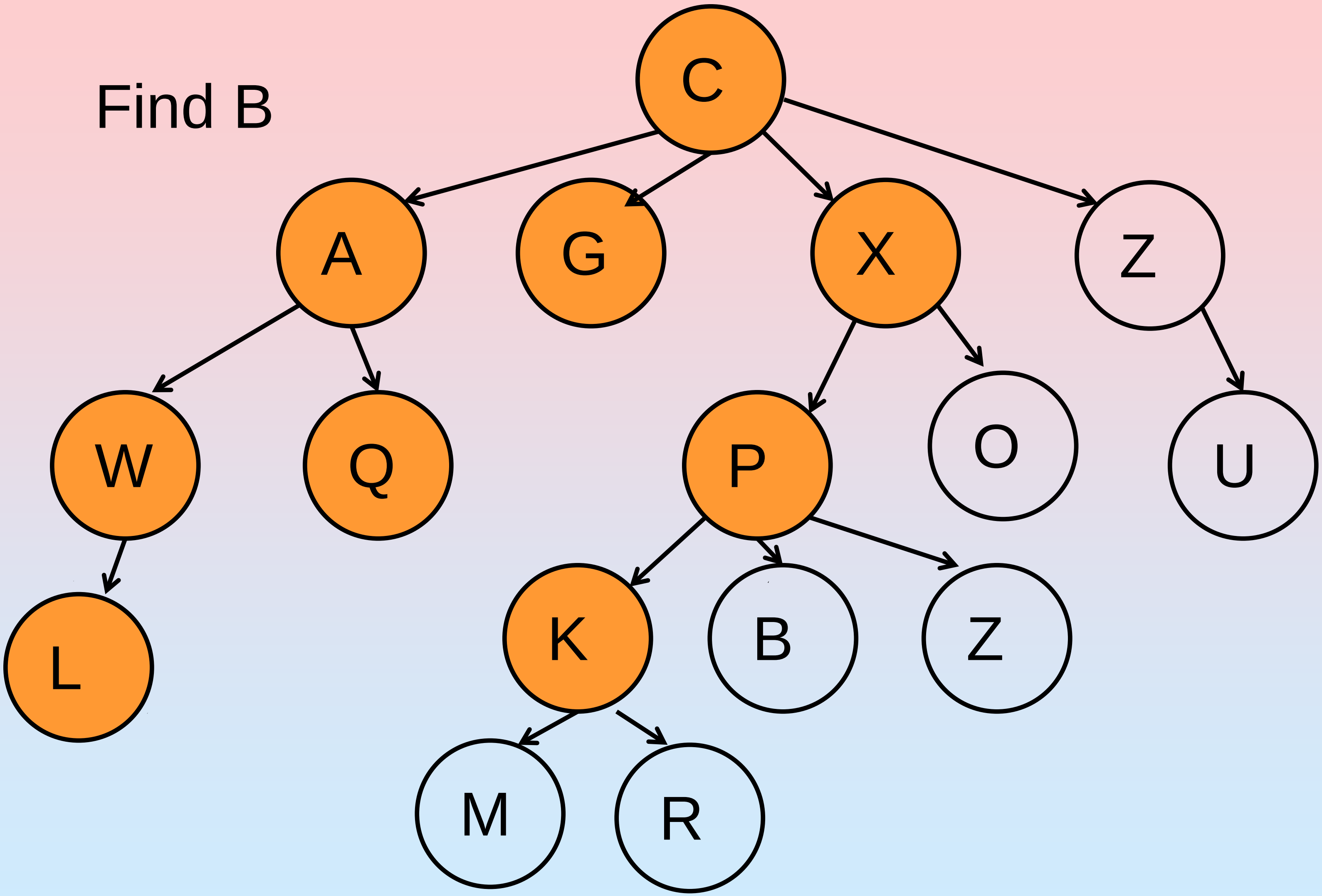
Depth First Search of Tree

Find B



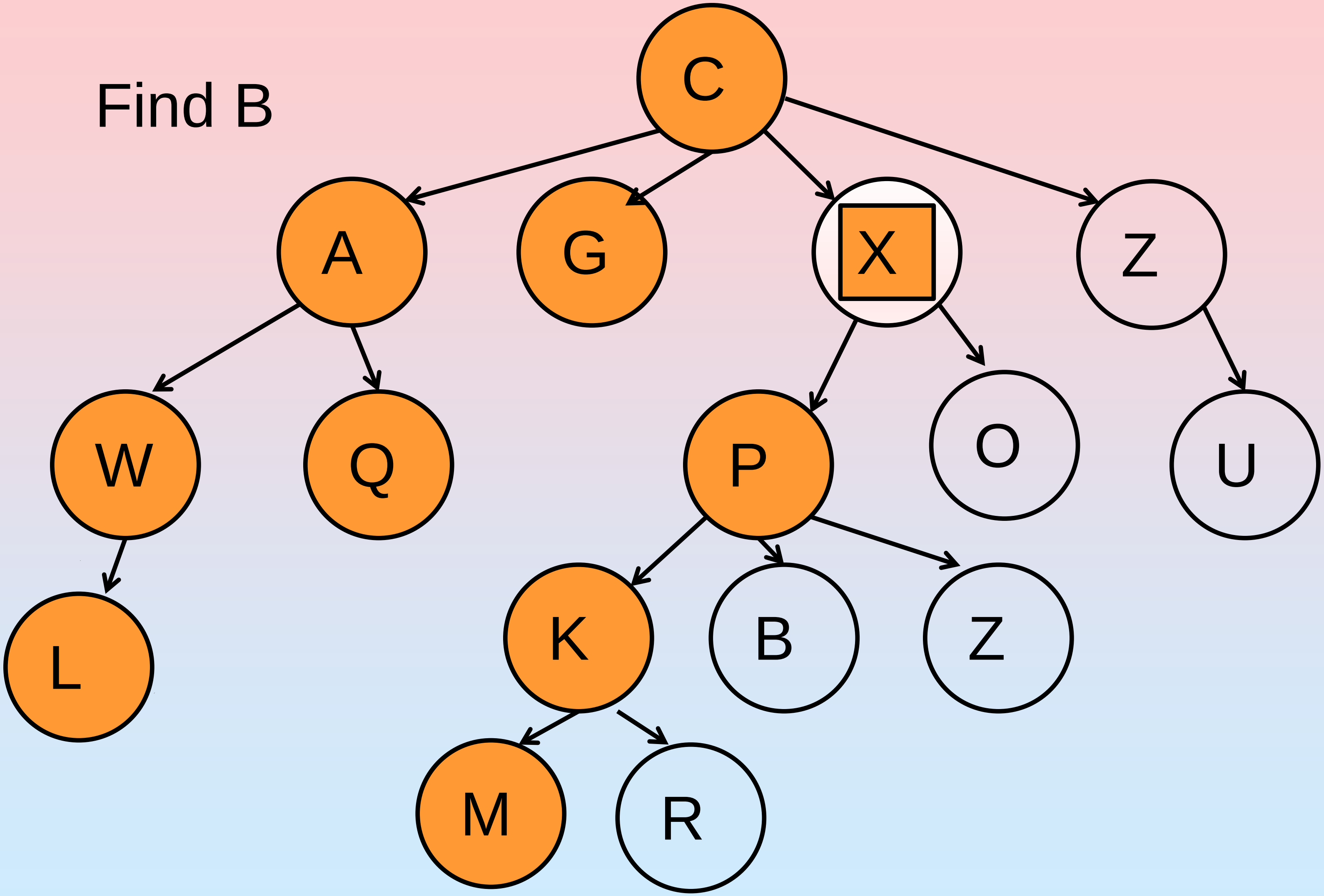
Depth First Search of Tree

Find B



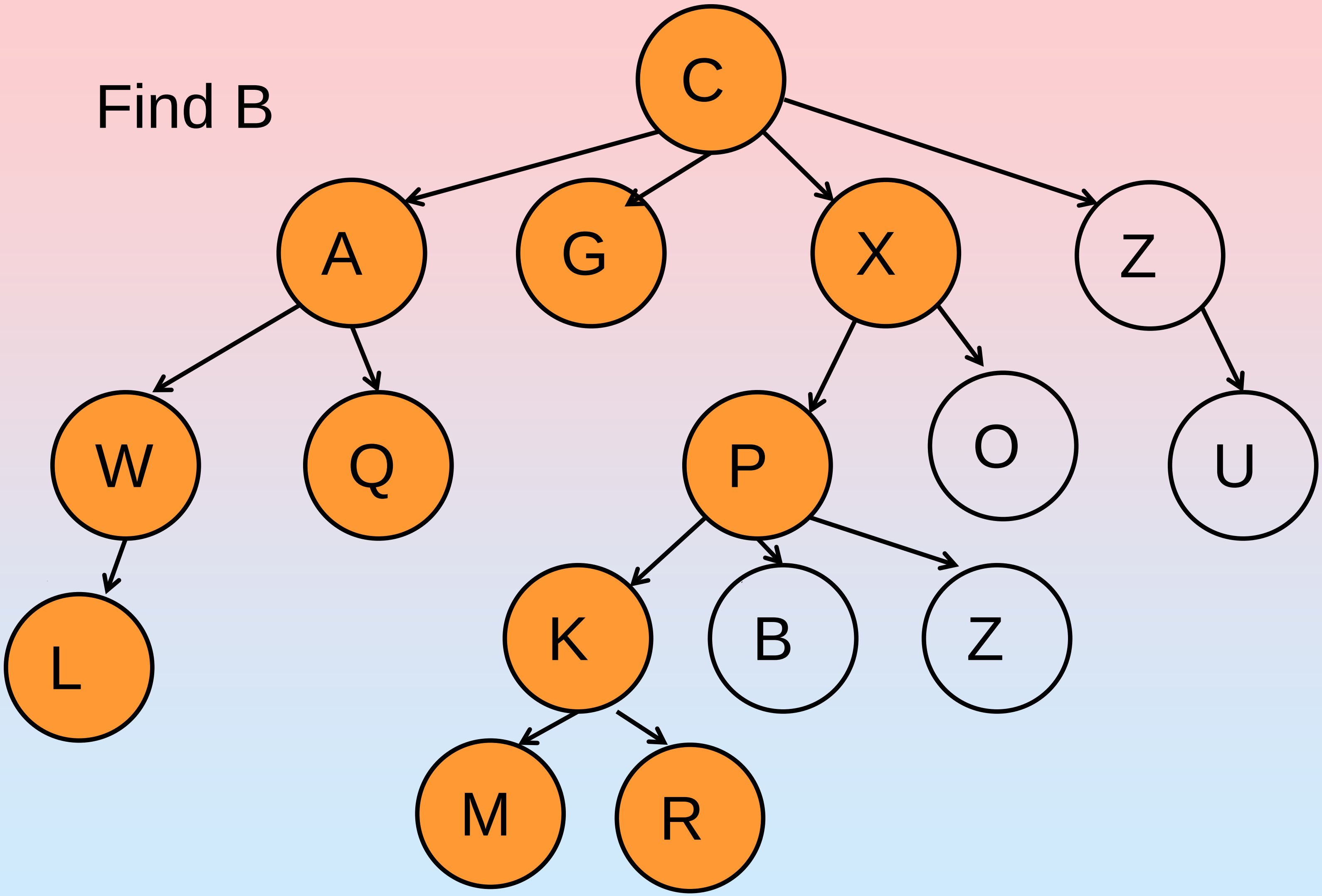
Depth First Search of Tree

Find B

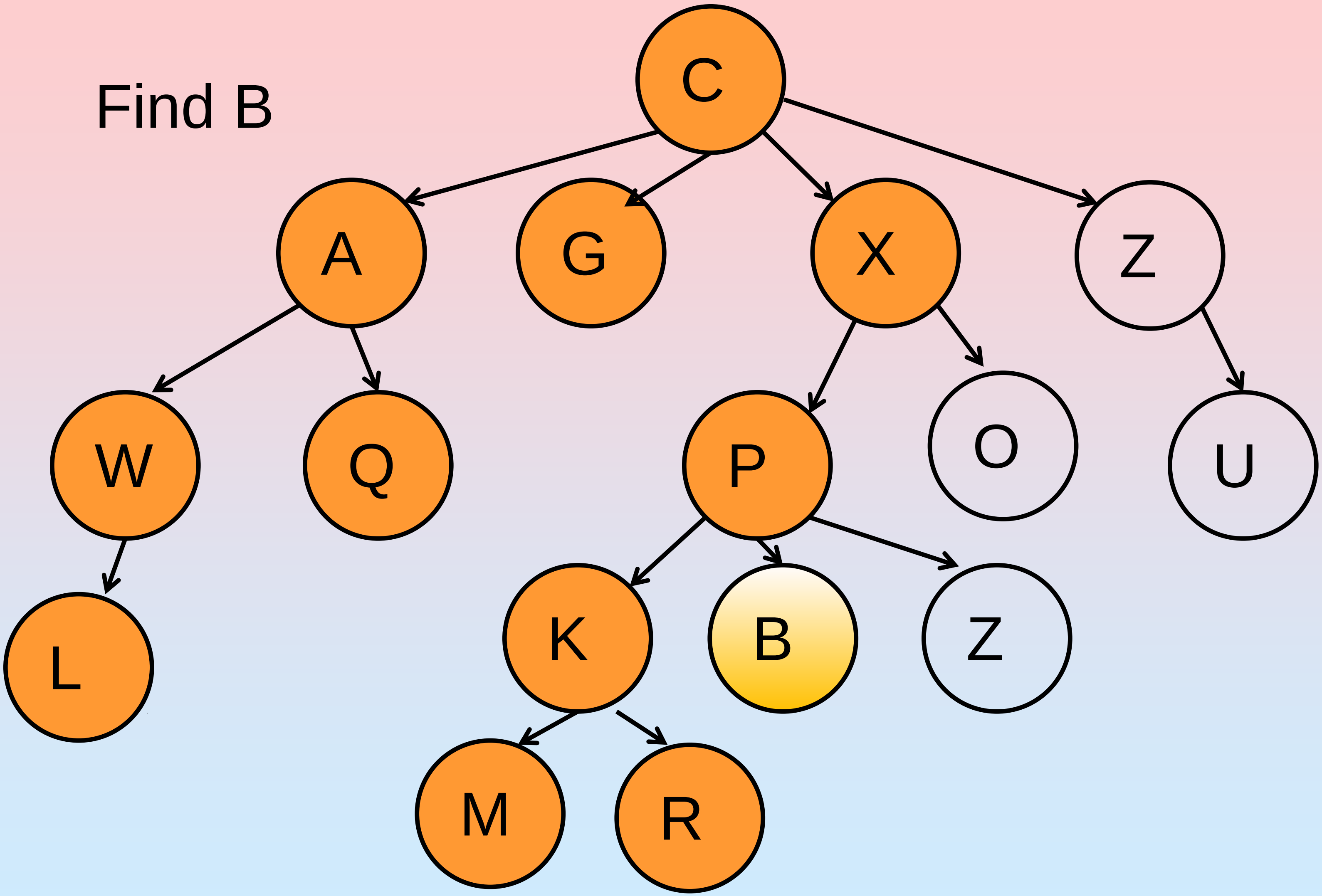


Depth First Search of Tree

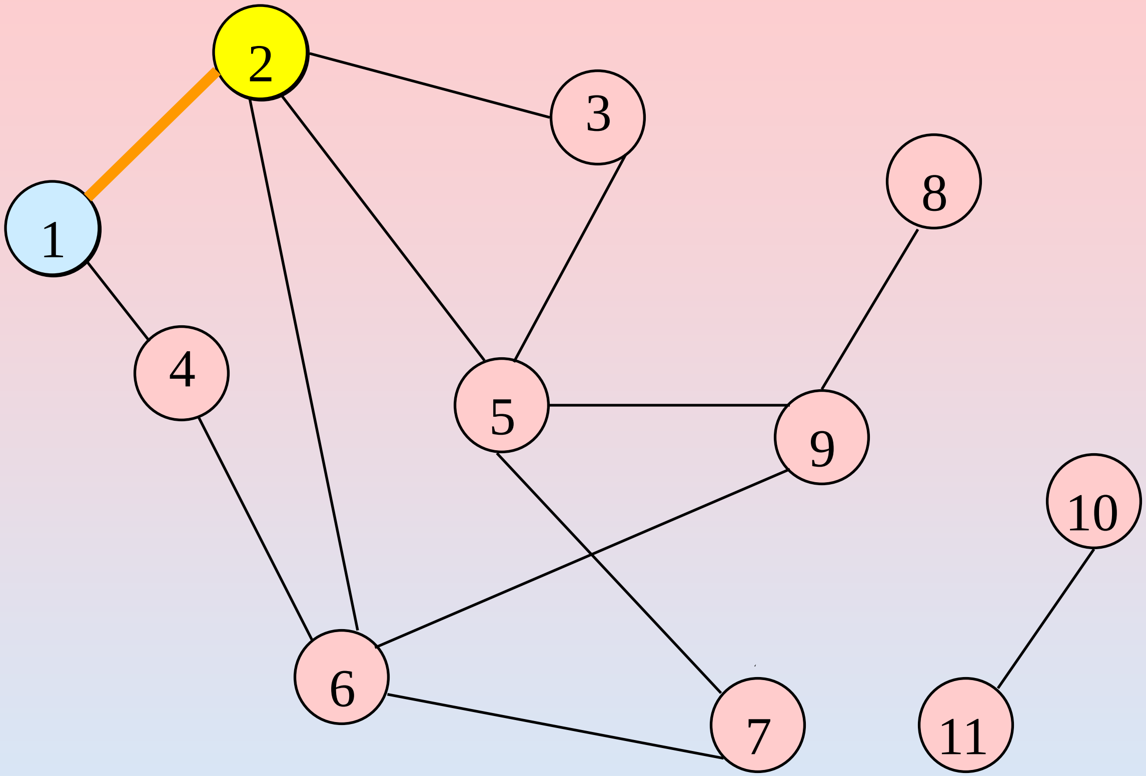
Find B



Depth First Search of Tree



Depth-First Search of Graph

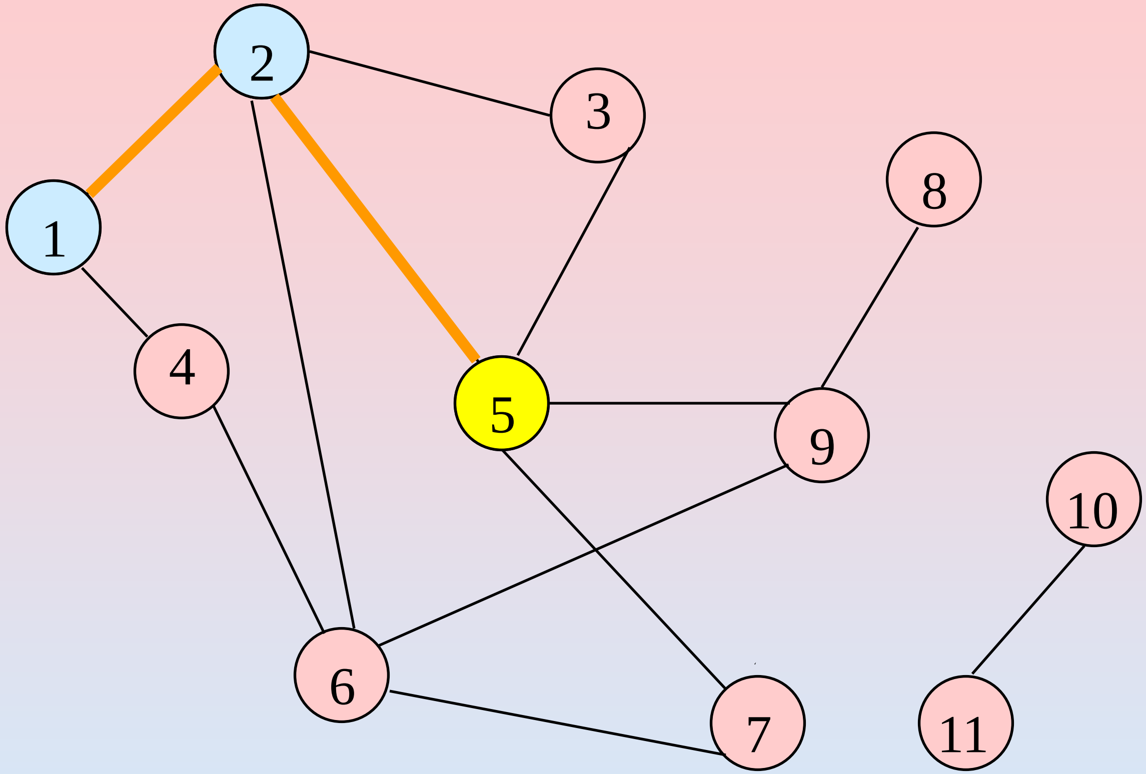


Start search at vertex 1.

Label vertex 1 and do a depth first search from either 2 or 4.

Suppose that vertex 2 is selected.

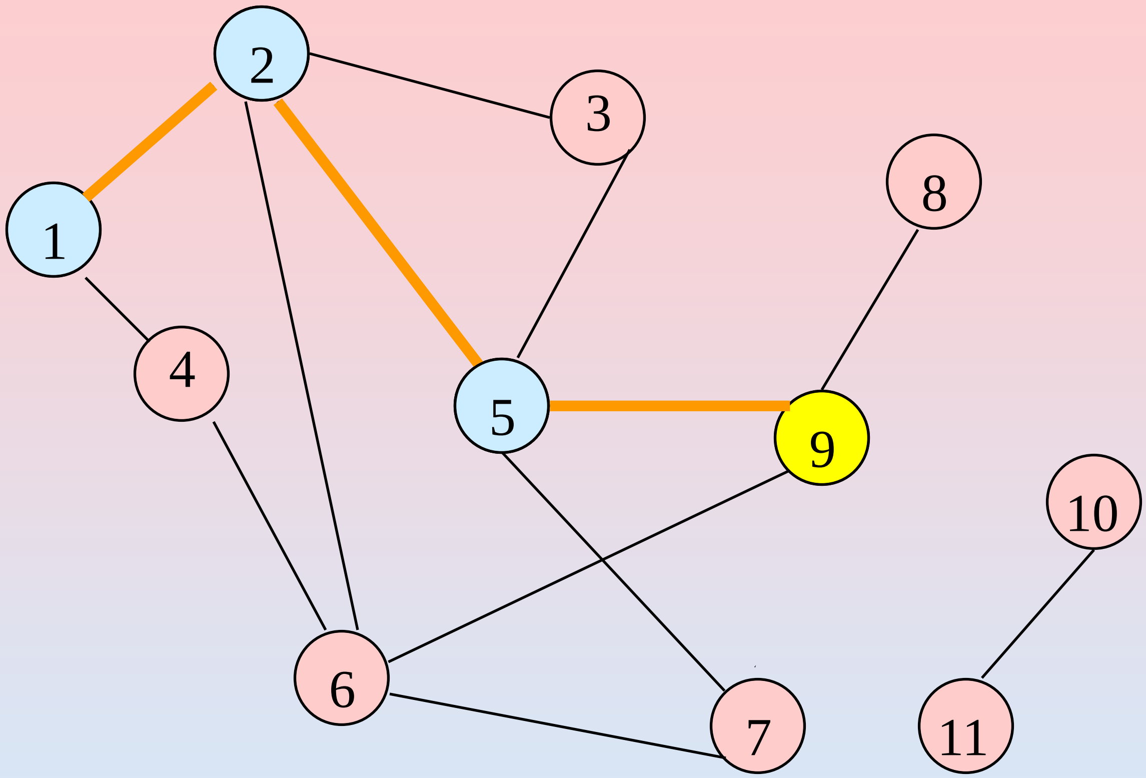
Depth-First Search of Graph



Label vertex 2 and do a depth first search from either 3, 5, or 6.

Suppose that vertex 5 is selected.

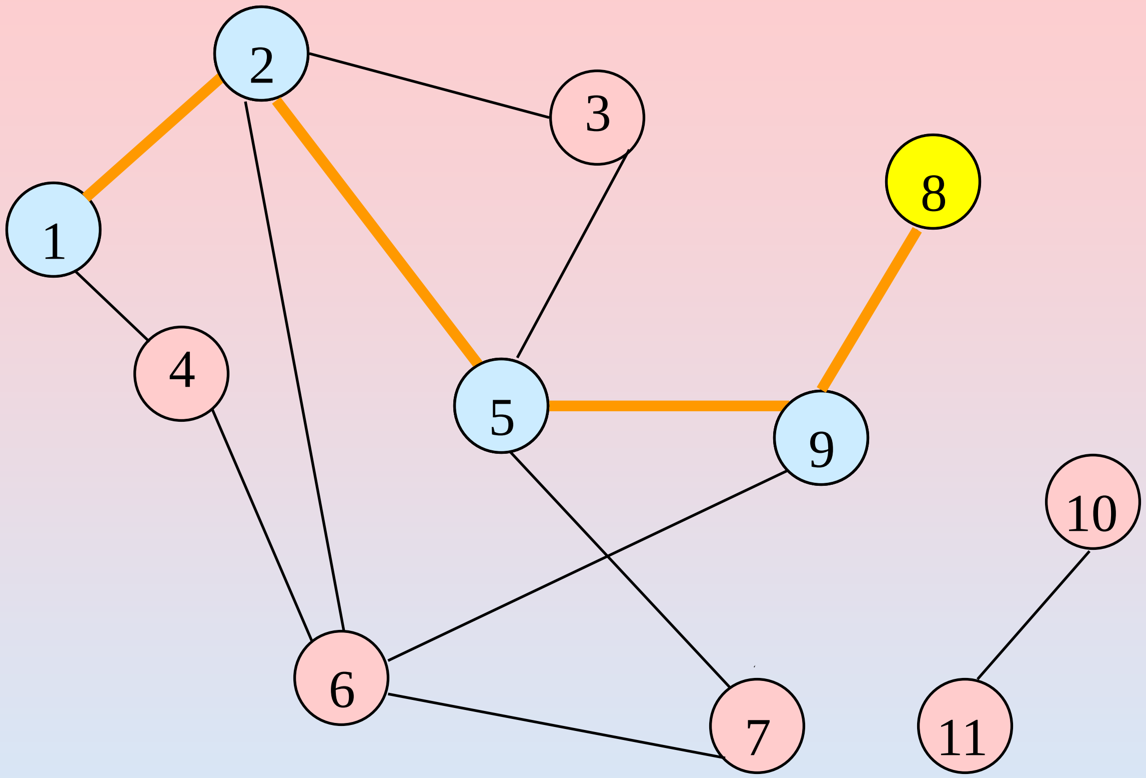
Depth-First Search of Graph



Label vertex 5 and do a depth first search from either 3, 7, or 9.

Suppose that vertex 9 is selected.

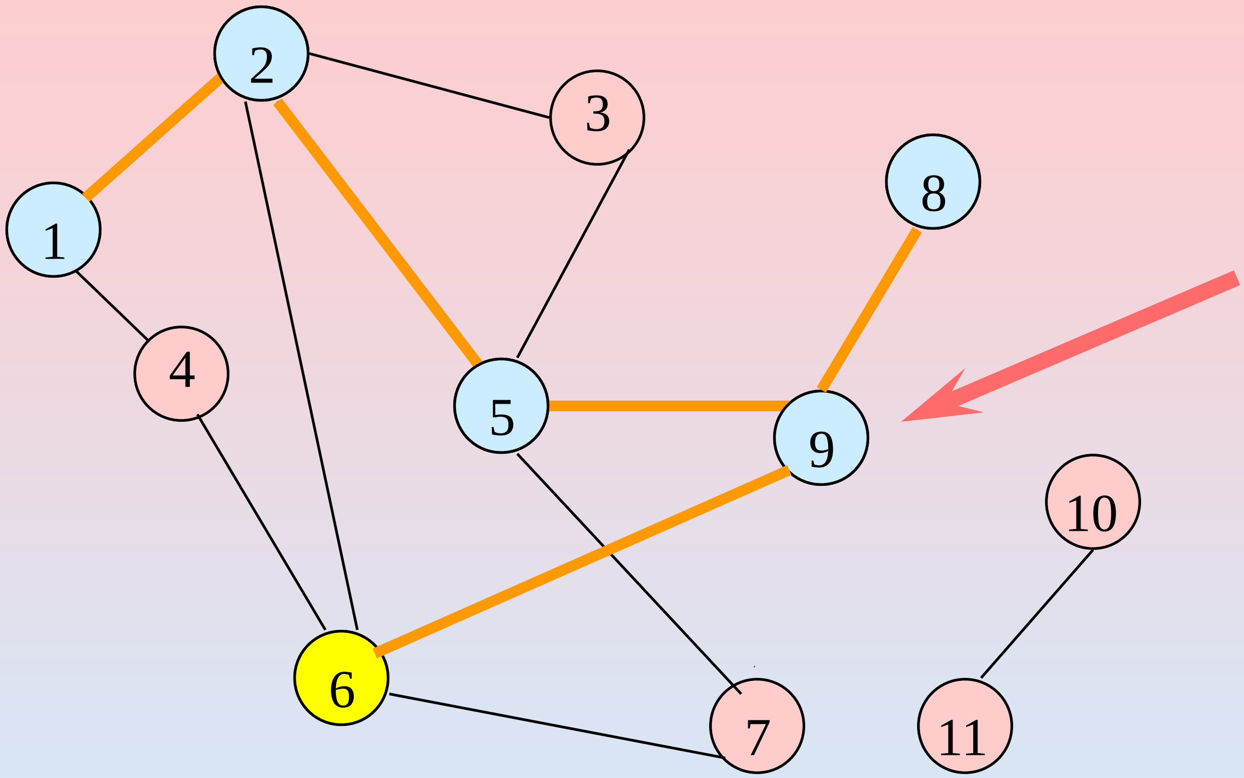
Depth-First Search of Graph



Label vertex 9 and do a depth first search from either 6 or 8.

Suppose that vertex 8 is selected.

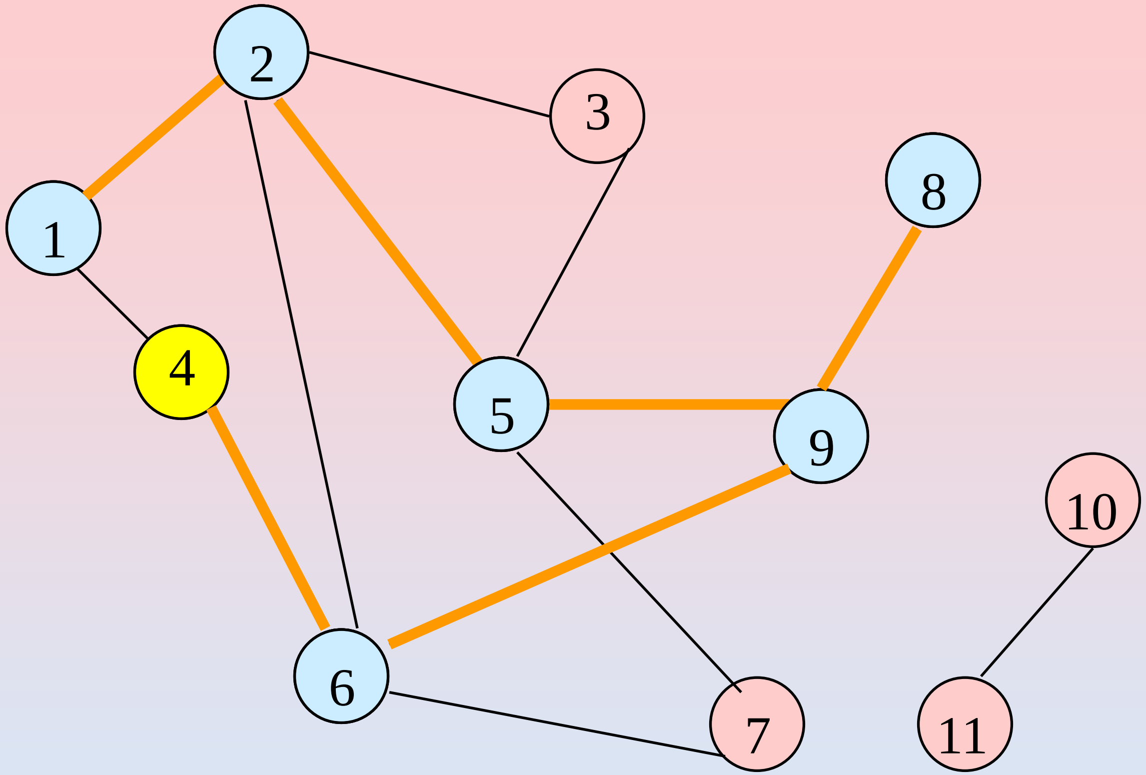
Depth-First Search of Graph



Label vertex 8 and return to vertex 9.

From vertex 9 do a DFS(6).

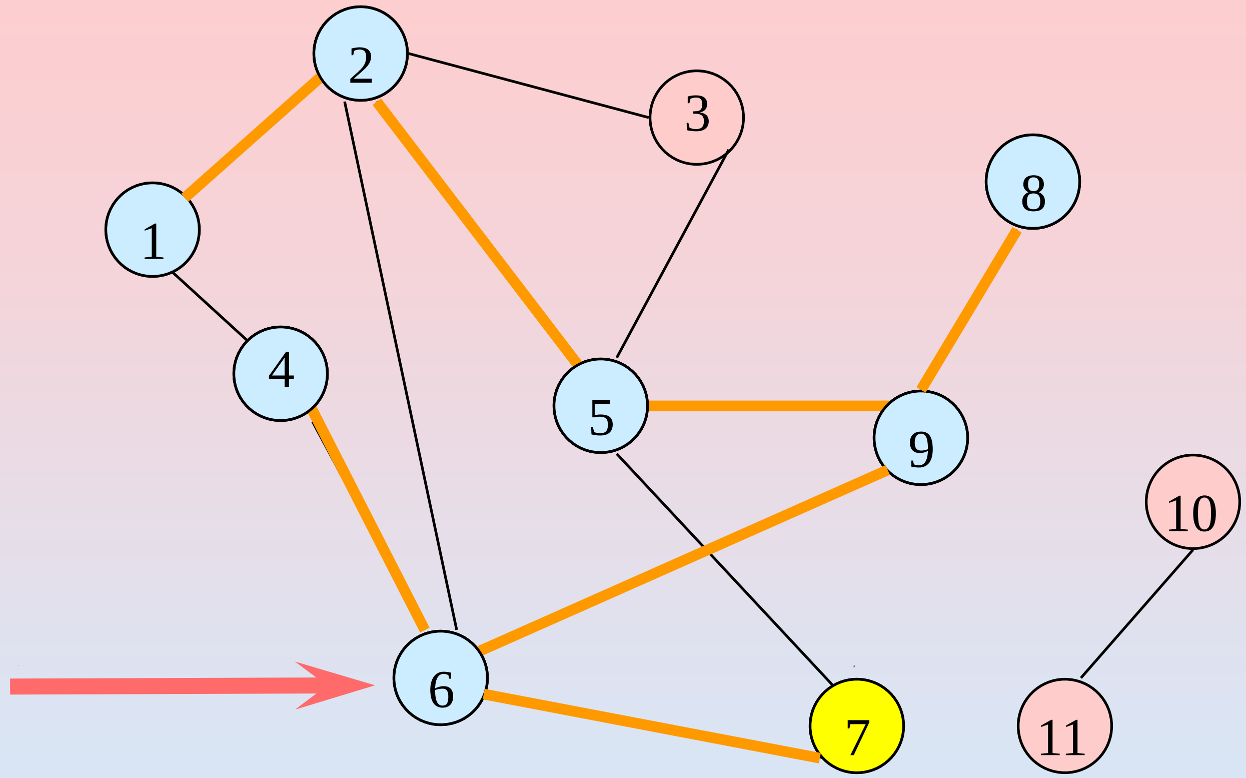
Depth-First Search of Graph



- **Label vertex 6 and do a depth first search from either 4 or 7.**

Suppose that vertex 4 is selected.

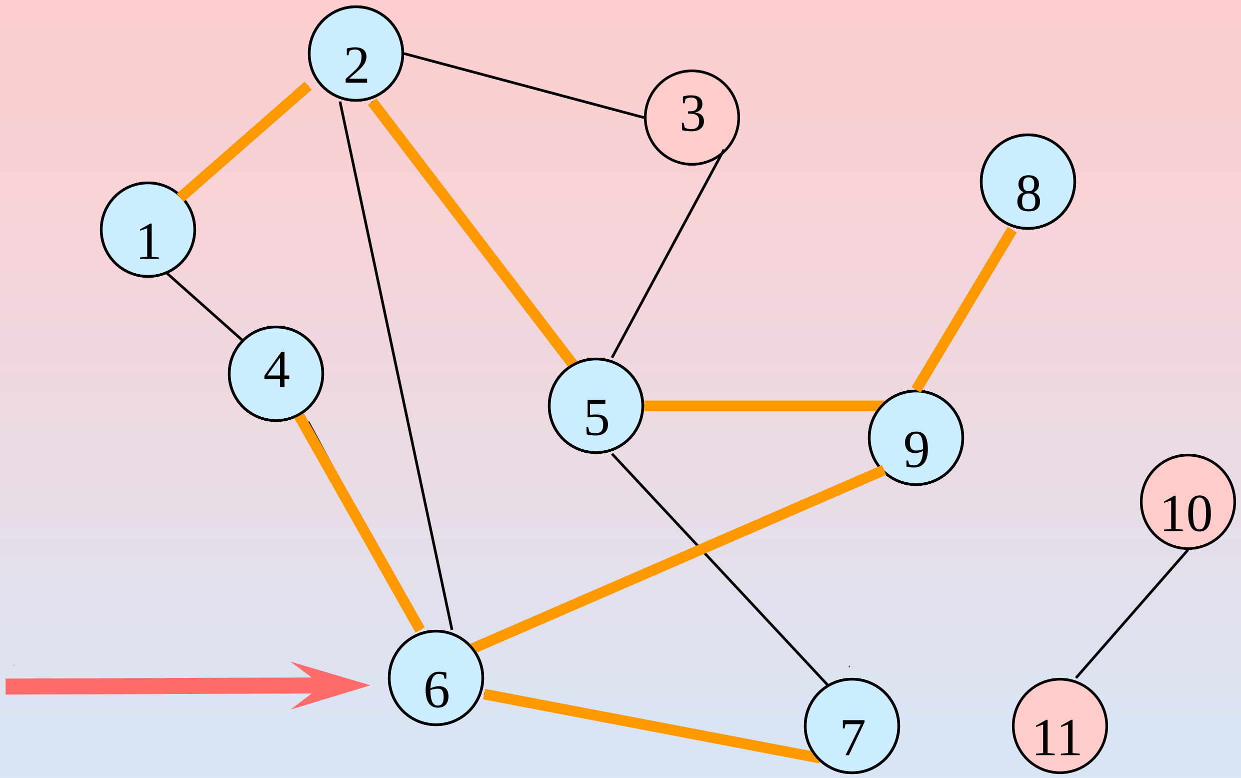
Depth-First Search of Graph



Label vertex 4 and return to 6.

From vertex 6 do a dfs(7).

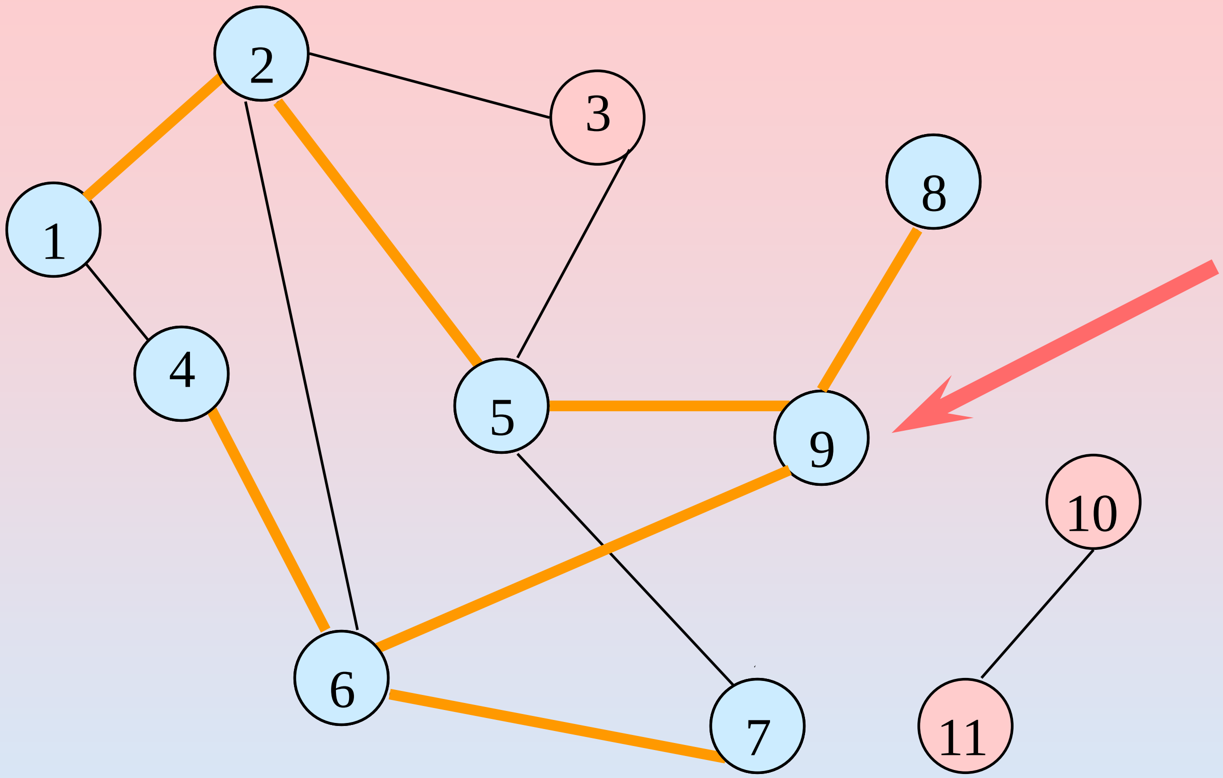
Depth-First Search of Graph



Label vertex 7 and return to 6.

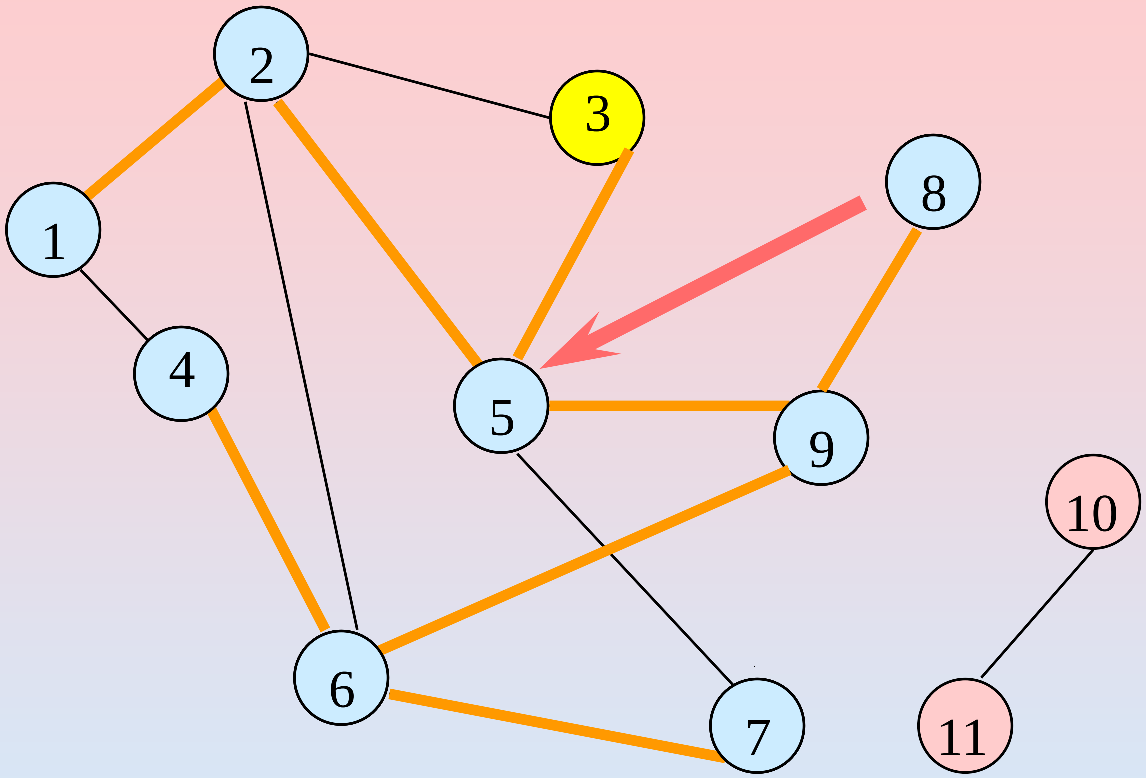
Return to 9.

Depth-First Search of Graph



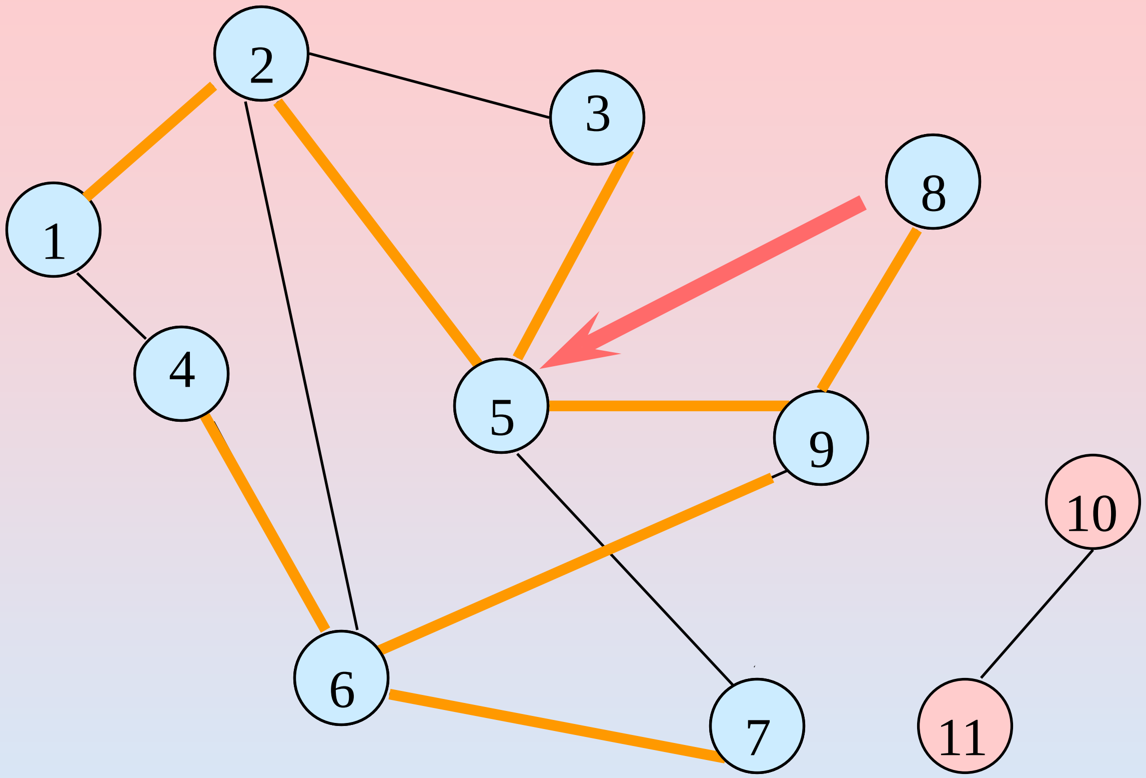
Return to 5.

Depth-First Search of Graph



Do a dfs(3).

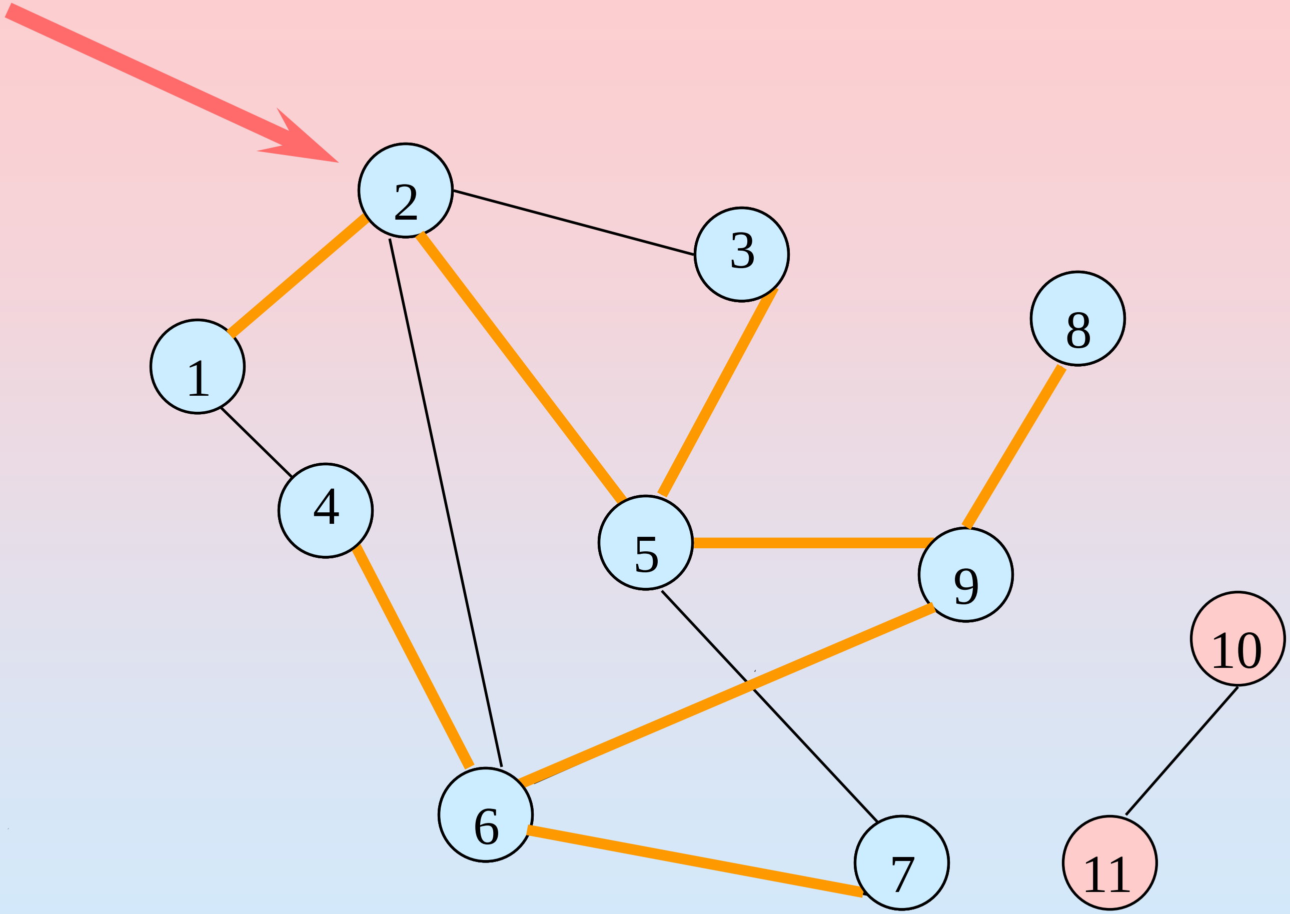
Depth-First Search of Graph



Label 3 and return to 5.

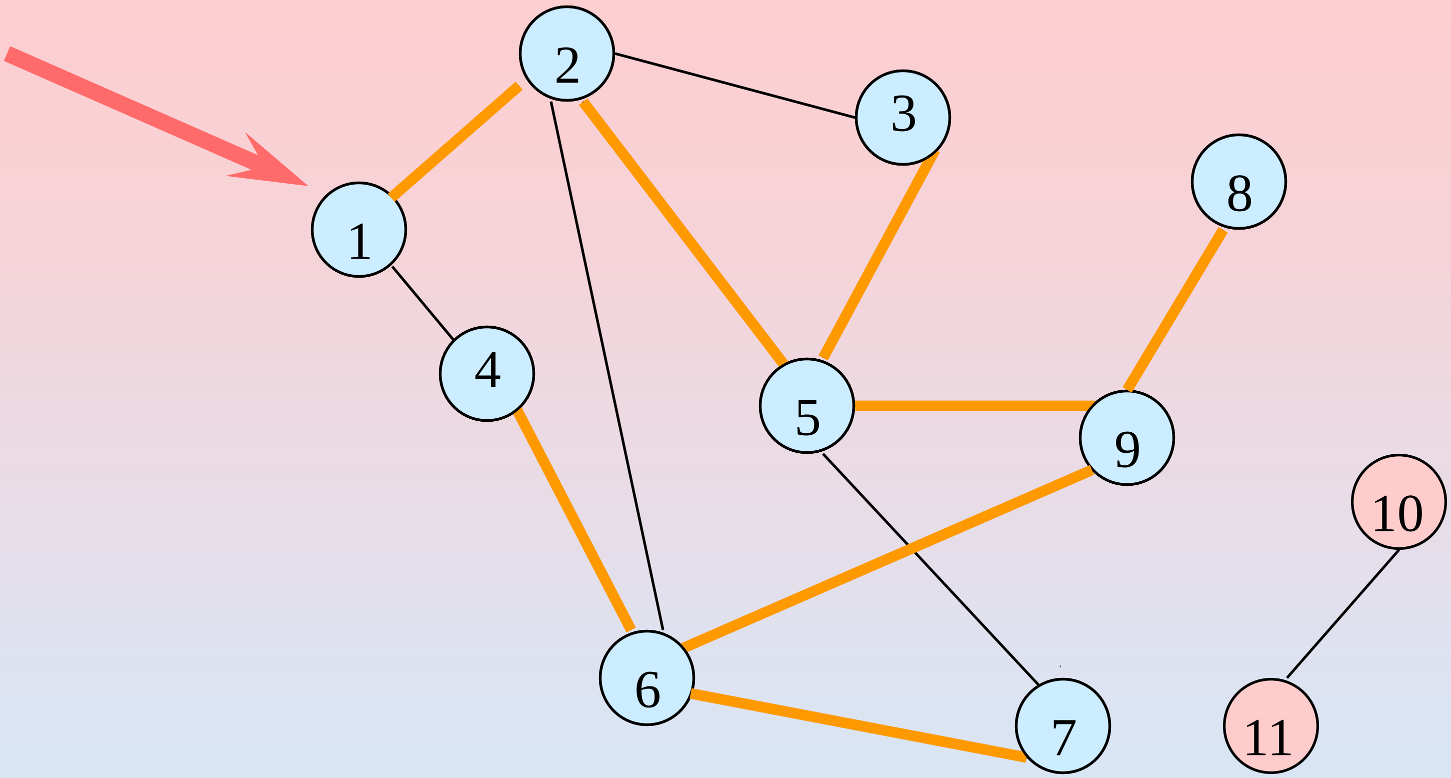
Return to 2.

Depth-First Search of Graph



Return to 1.

Depth-First Search of Graph



Return to invoking method.

Path from Vertex v to Vertex u

- Start a depth-first search at vertex v.
- Terminate when vertex u is visited or when dfs ends (whichever occurs first).
- Time Complexity :
  - $O(n^2)$ when <u>adjacency matrix used</u> : If the graph is implemented as an adjacency matrix (a n x n array), then, for each node, need to traverse an entire row of length n in the matrix to discover all its outgoing edges. Note that each row in an adjacency matrix corresponds to a node in the graph, and the said row stores information about edges stemming from the node. So, the complexity of DFS is $O(n * n) = O(n^2)$.

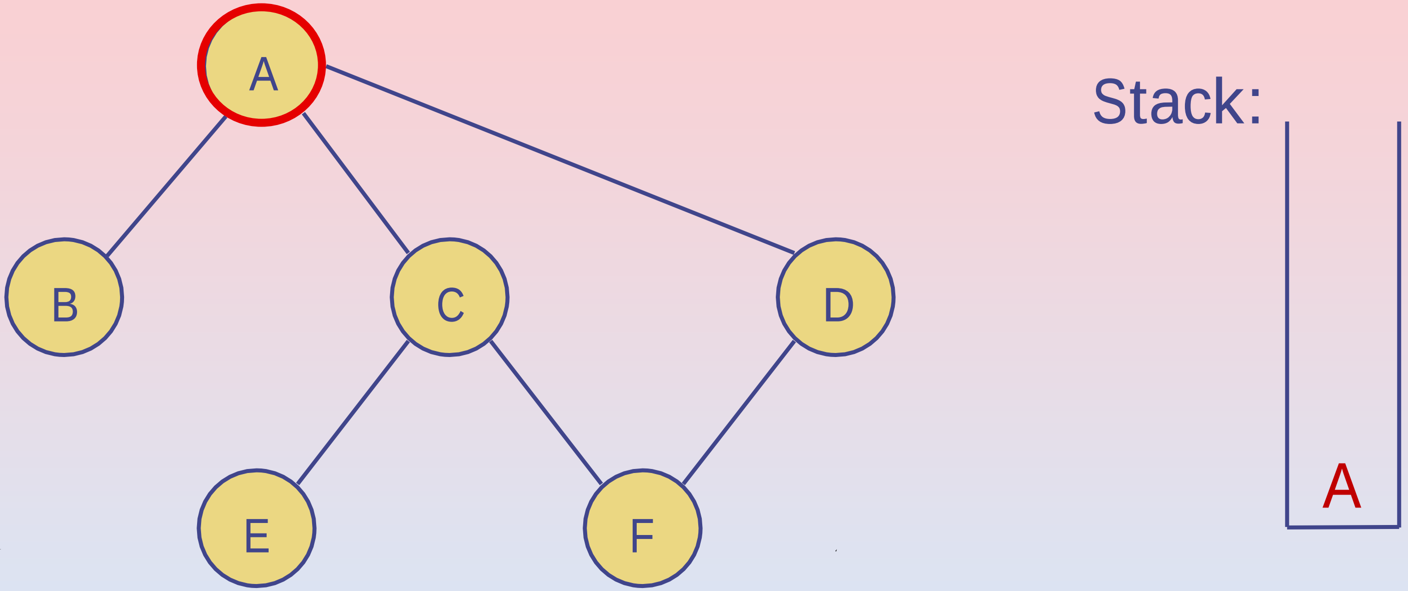
Path from Vertex v to Vertex u

- **Time Complexity :**
  - **O(n+e) when <u>adjacency lists used</u> (e is number of edges) : If the graph is implemented using adjacency lists, wherein each node maintains a list of all its adjacent edges, then, for each node, need to discover all its neighbors by traversing its adjacency list just once in linear time. For a directed graph, the sum of the sizes of the adjacency lists of all the nodes is e (total number of edges). So, the complexity of DFS is O(n) + O(e) = O(n + e).**
  - **For an undirected graph, each edge will appear twice in the adjacency list: for an edge ab, a would appear in adjacency list of b, and b would appear in adjacency list of a. So, the overall complexity will be O(n) + O (2e) ~ O(n + e).**

DFS implemetation

- **Depth first search typically implemented with stack, implicit with recursion or iteratively with an explicit stack**
- **Start with a node.**
- **Push that node onto the stack.**
- **Each time a node is popped off the stack, push all of the new neighbors of that node onto the stack.**

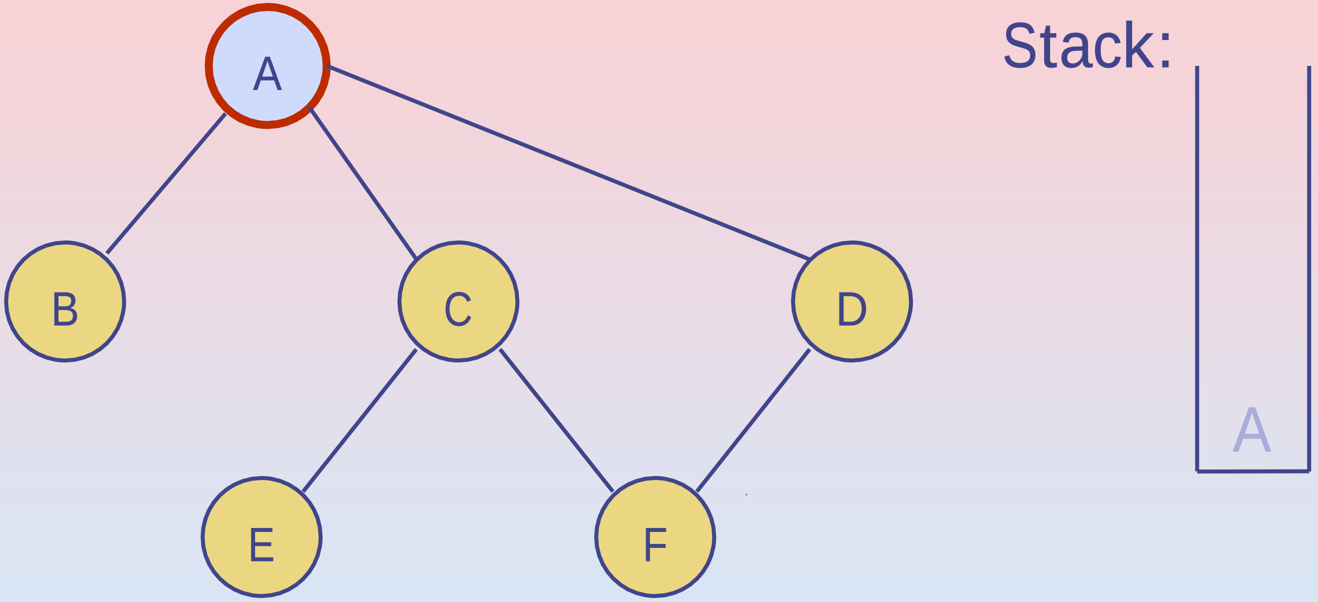
DFS implemetation



Start with a node. Let’s start at A!

Push the A onto the stack.

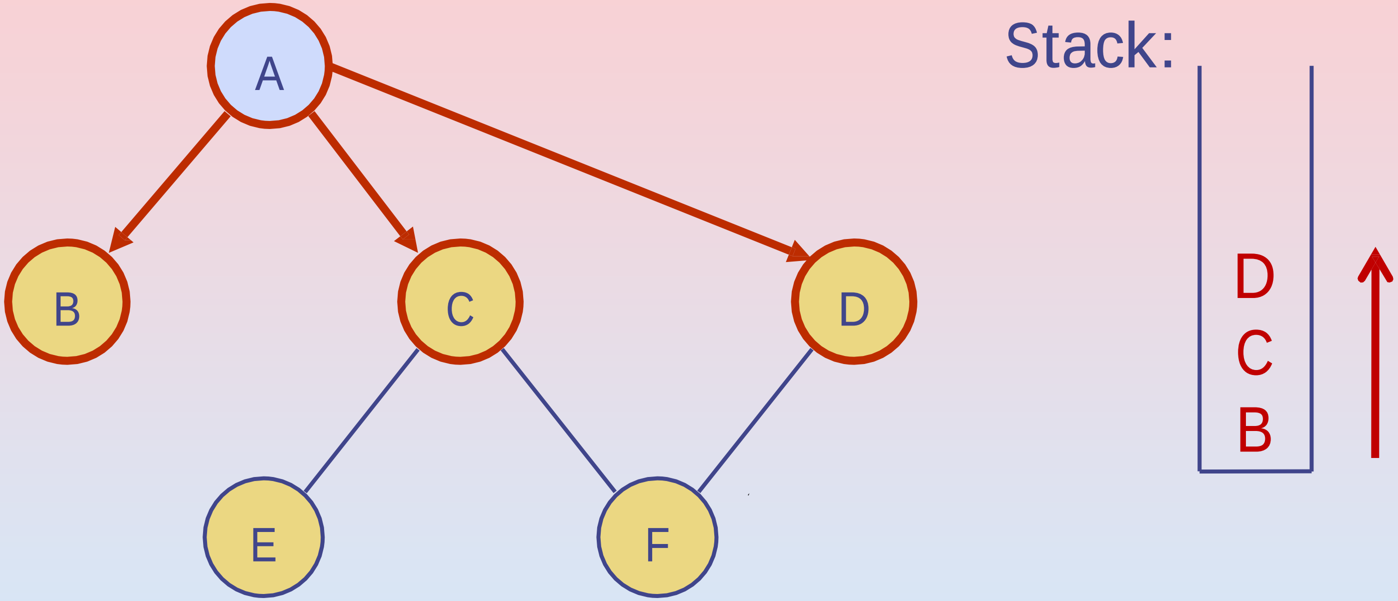
DFS implemetation



Pop a node off the stack.

A

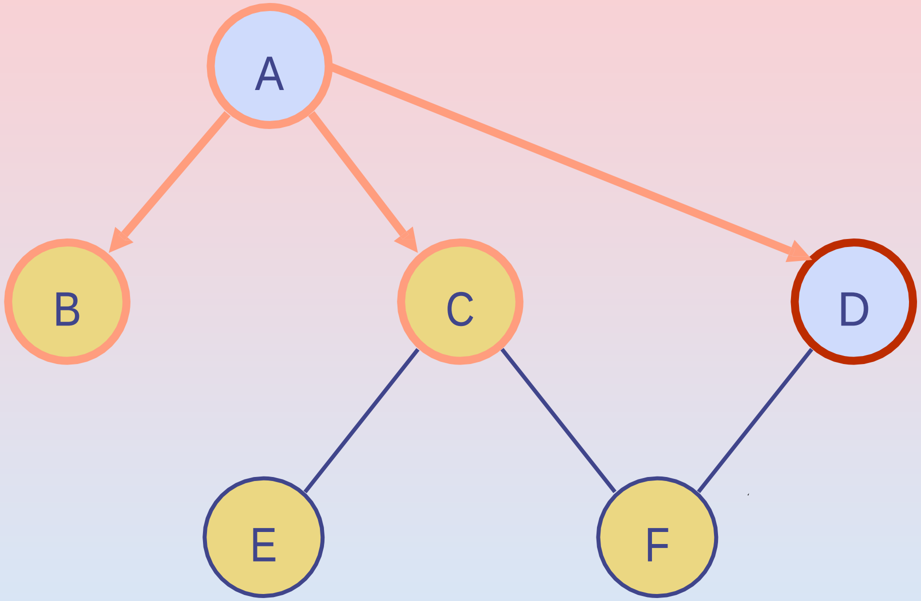
DFS implemetation

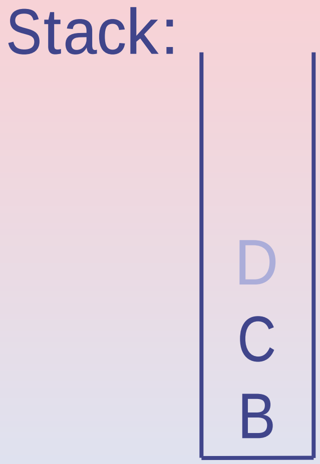


Push the new neighbors of root A onto the stack.

A

DFS implemetation

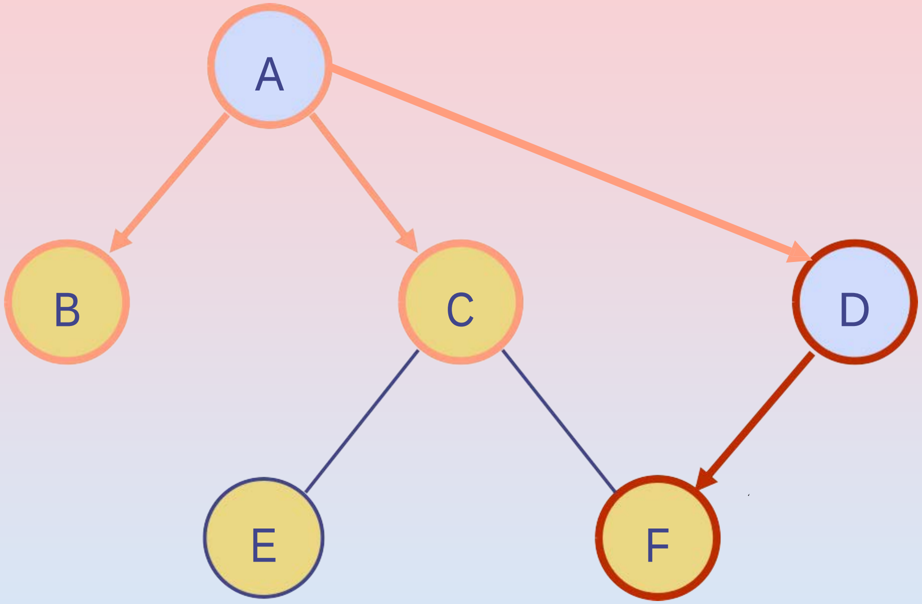




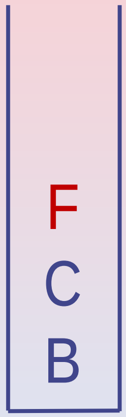
Pop a node off the stack.

A D

DFS implemetation



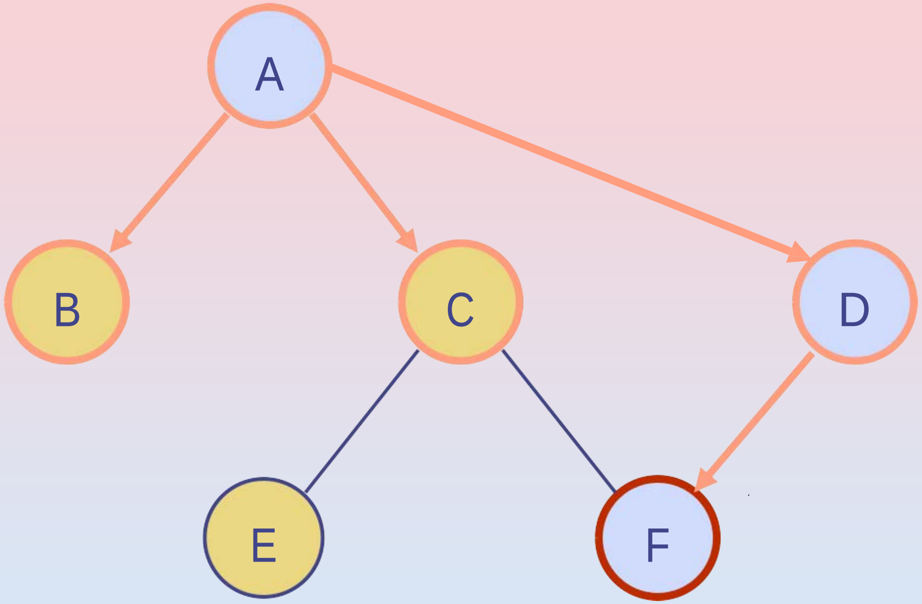
Stack:



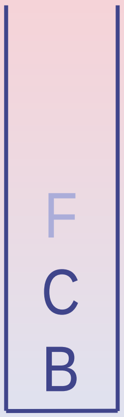
Push the new neighbors of D onto the stack.

A D

DFS implemetation



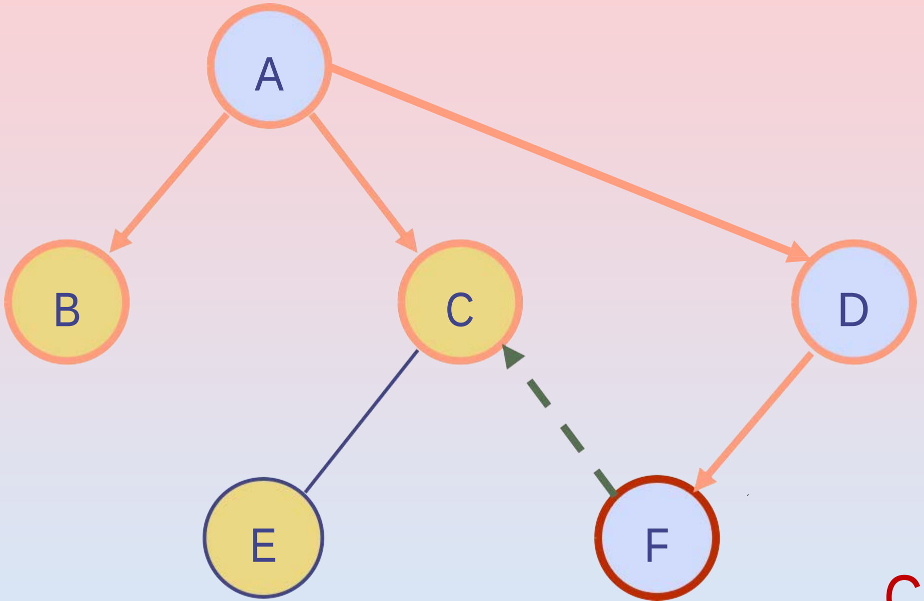
Stack:



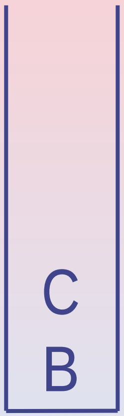
Pop a node off the stack.

A D F

DFS implemetation



Stack:

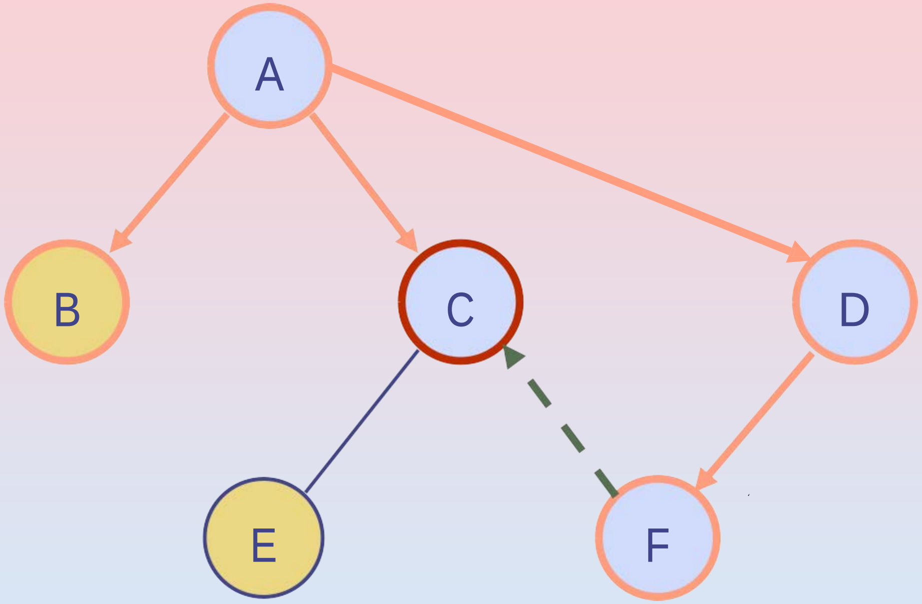


C is already visited!

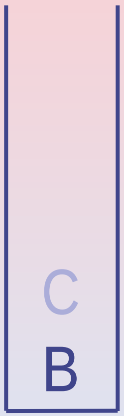
Push the new neighbors of F onto the stack.

A D F

DFS implemetation



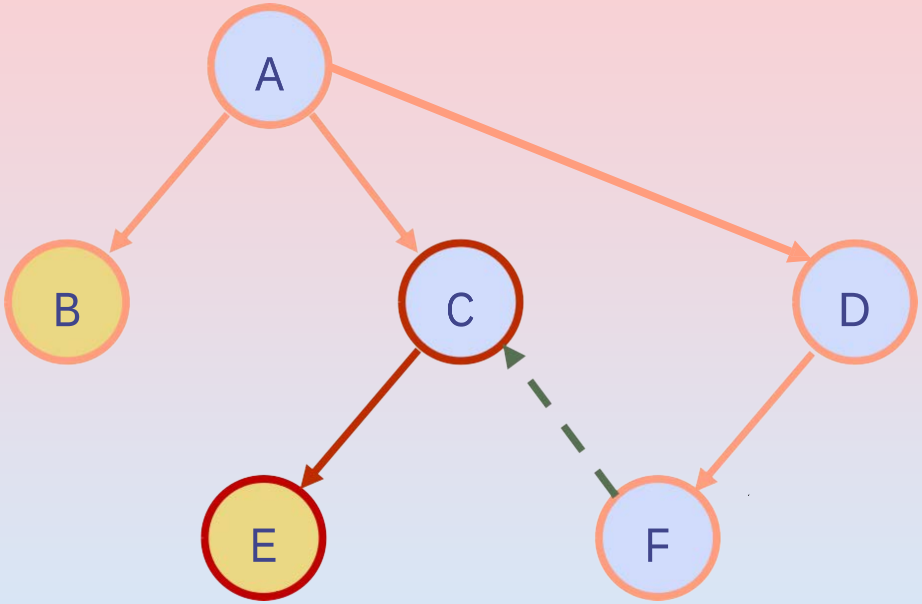
Stack:



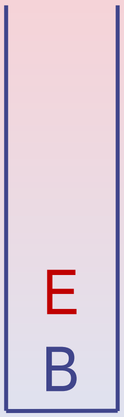
Pop a node off the stack.

A D F C

DFS implemetation



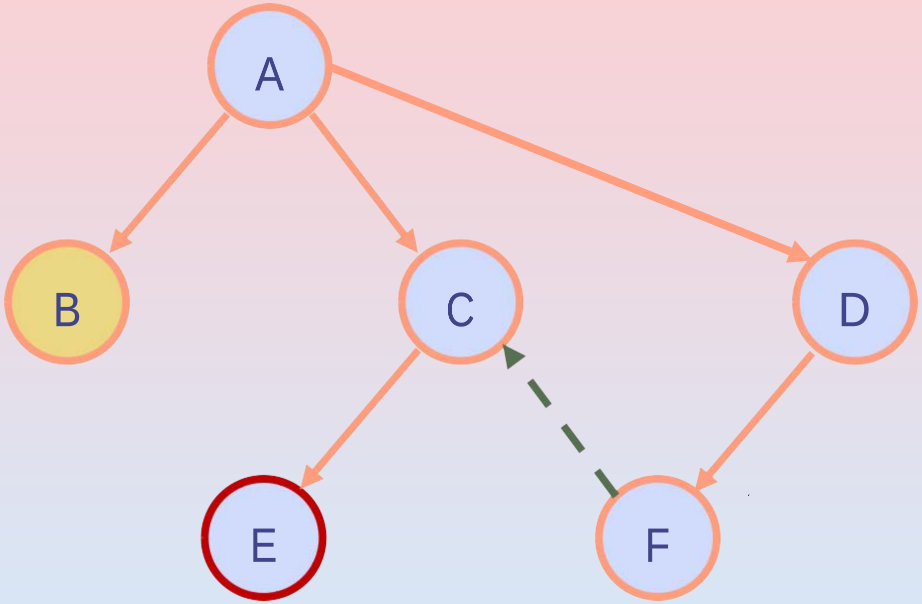
Stack:



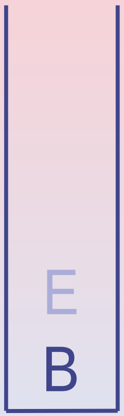
Push the new neighbors of C onto the stack.

A D F C

DFS implemetation



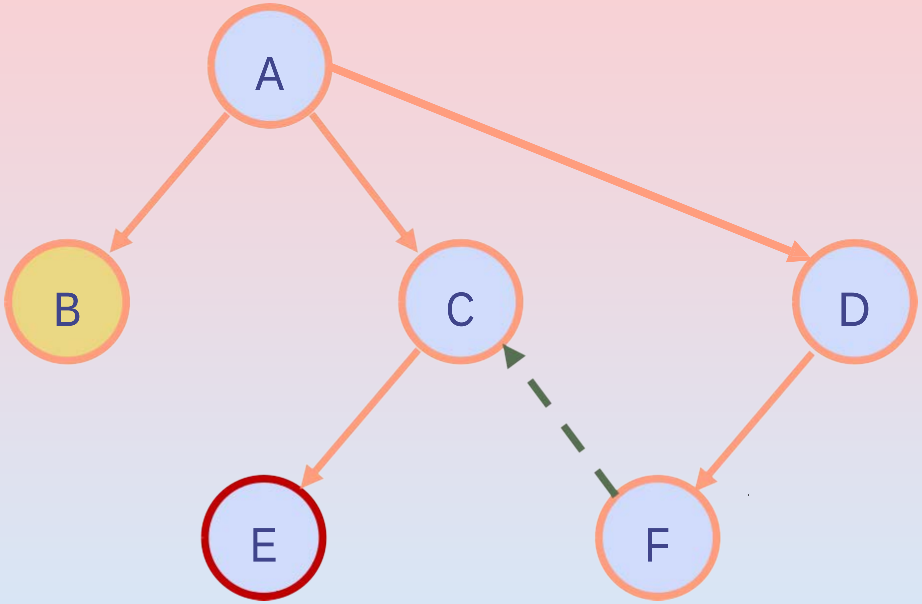
Stack:



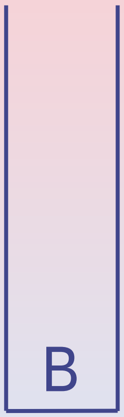
Pop a node off the stack.

A D F C E

DFS implemetation



Stack:

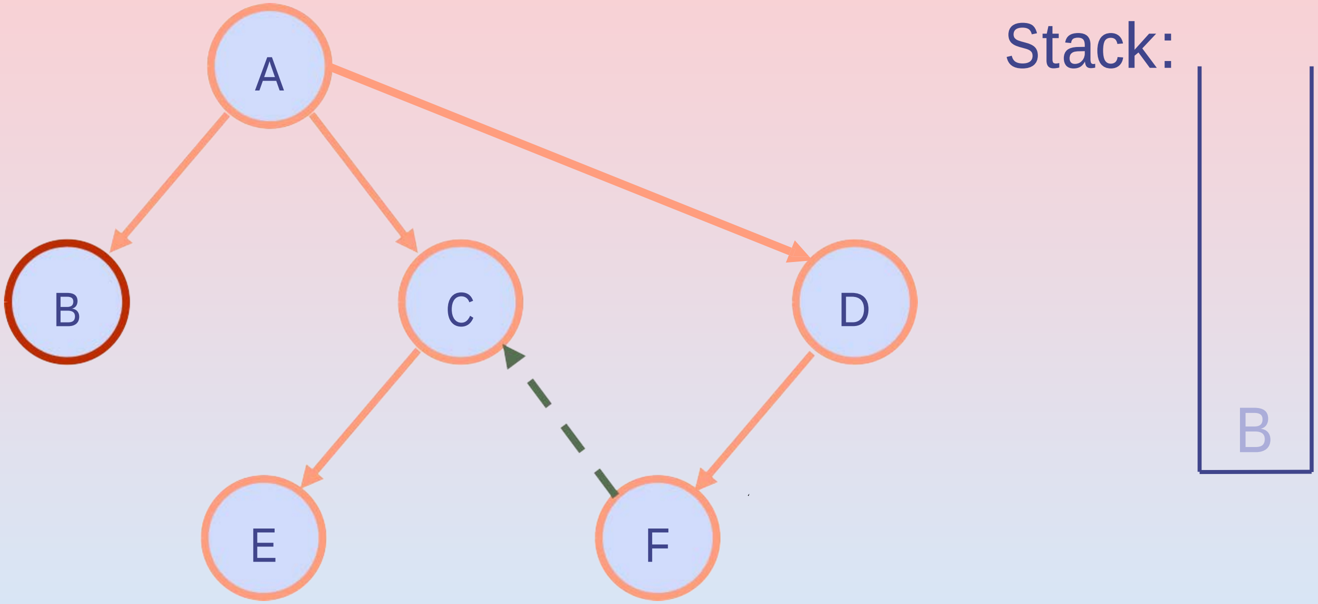


There are none!

Push the new neighbors of E onto the stack.

A D F C E

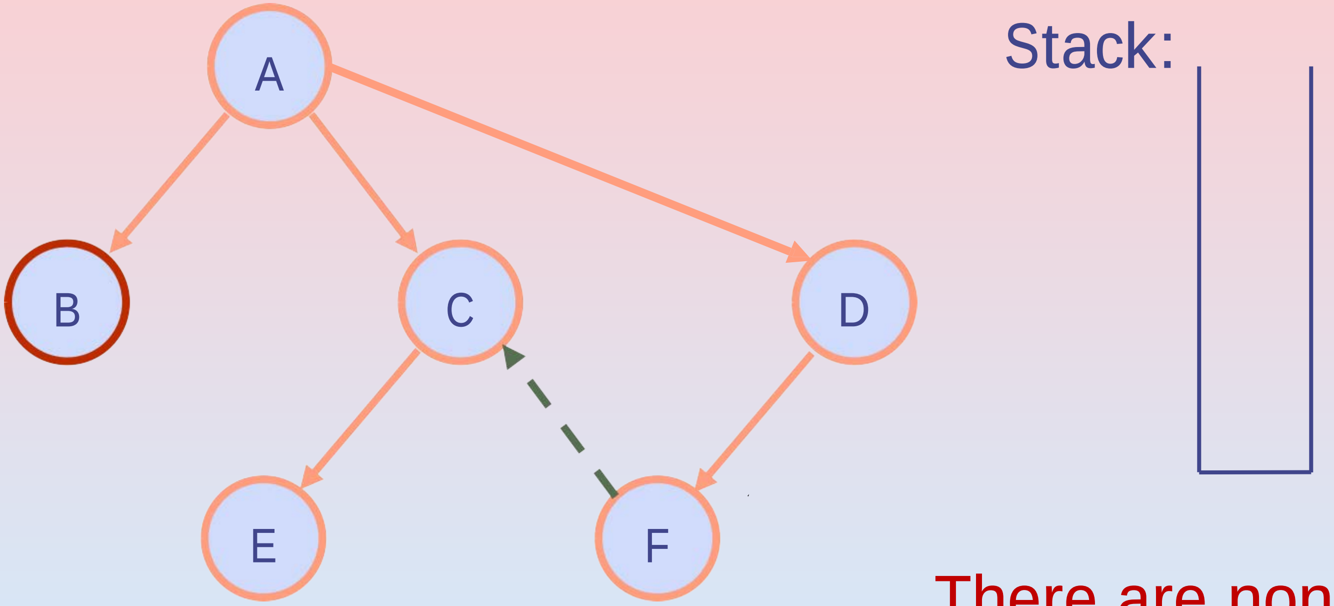
DFS implemetation



Pop a node off the stack.

A D F C E B

DFS implemetation

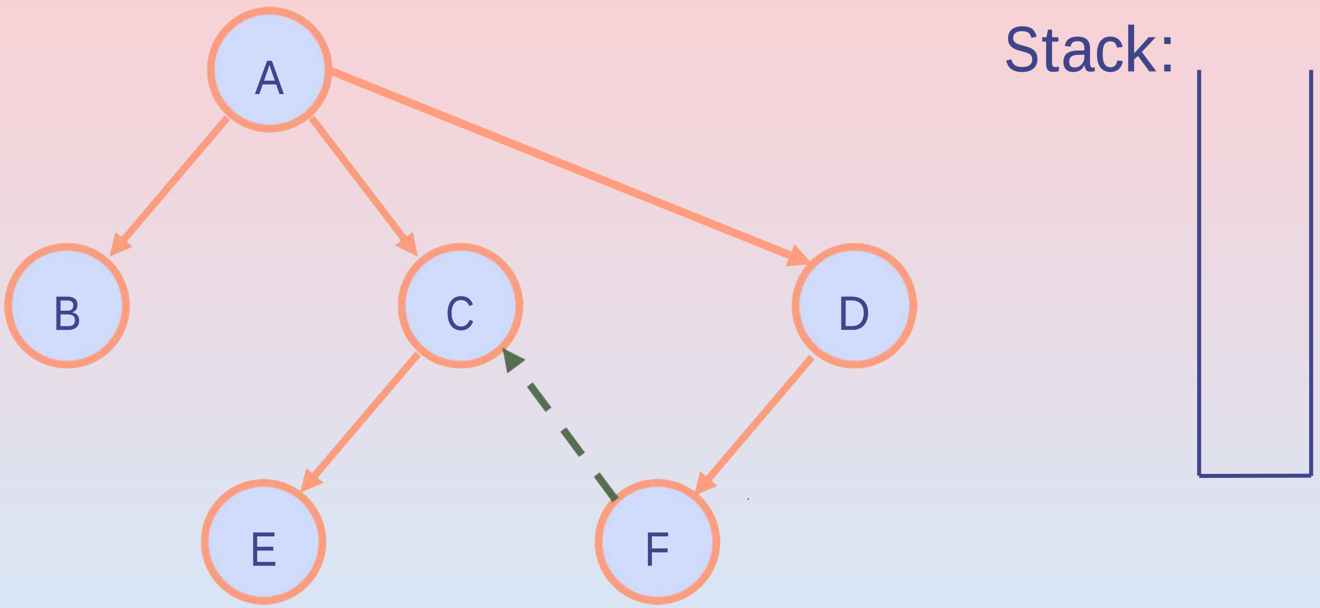


There are none!

Push the new neighbors of B onto the stack.

A D F C E B

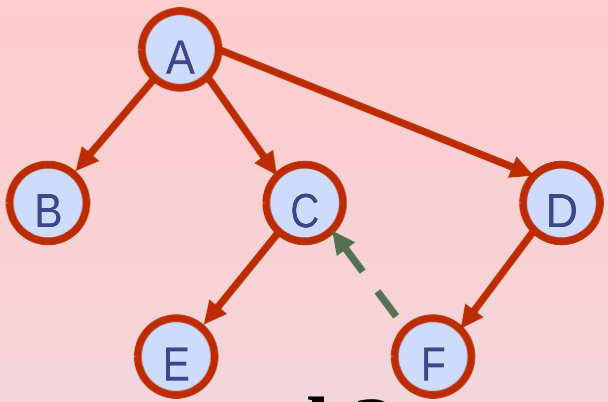
DFS implemetation



The next step would be to pop a node off the stack.

A D F C E B But since the stack is empty, we're done!

DFS implemetation

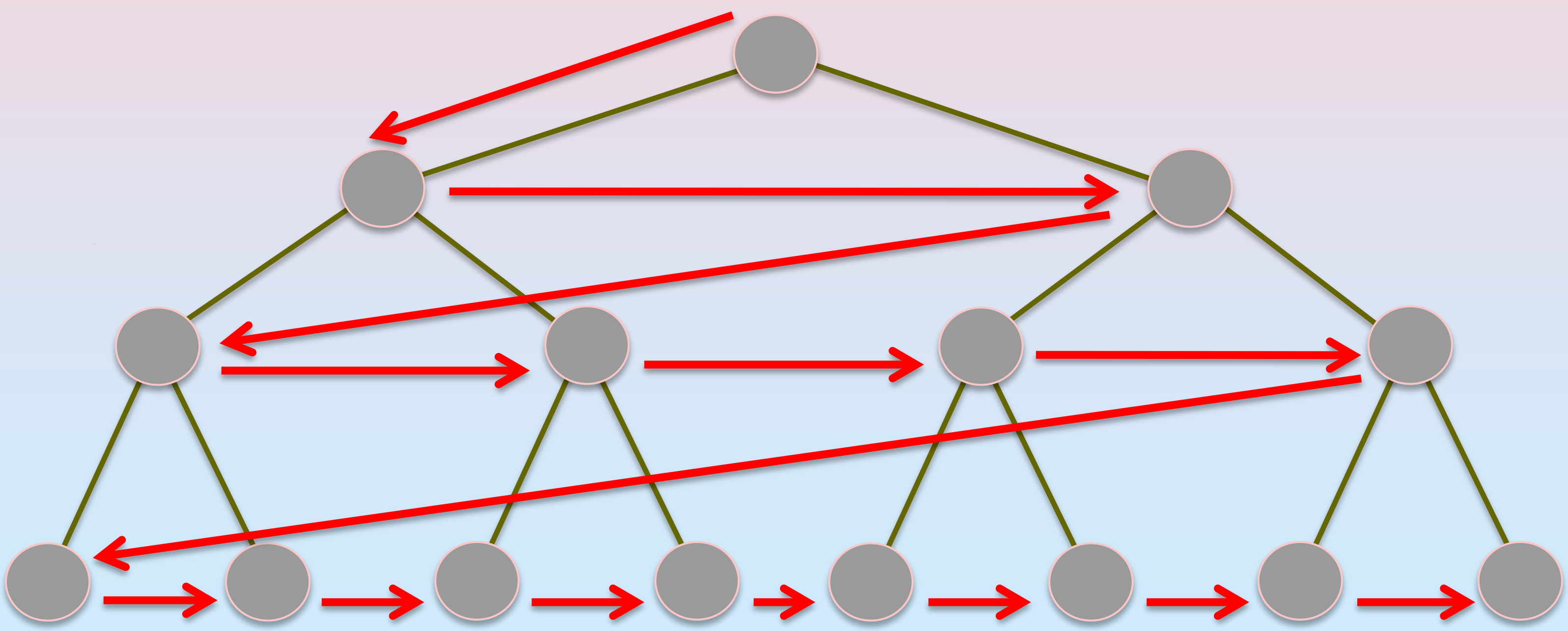


- **We got DFS A D F C E B**
- **Is it the only one DFS for the given graph?**
  - **NO**
- **Another DFS for the given graph?**
  - **A B C E F D**
- **If we want the search to end up A B C E F D, push the neighbors into the stack from right to left (e.g. D, C, B).**

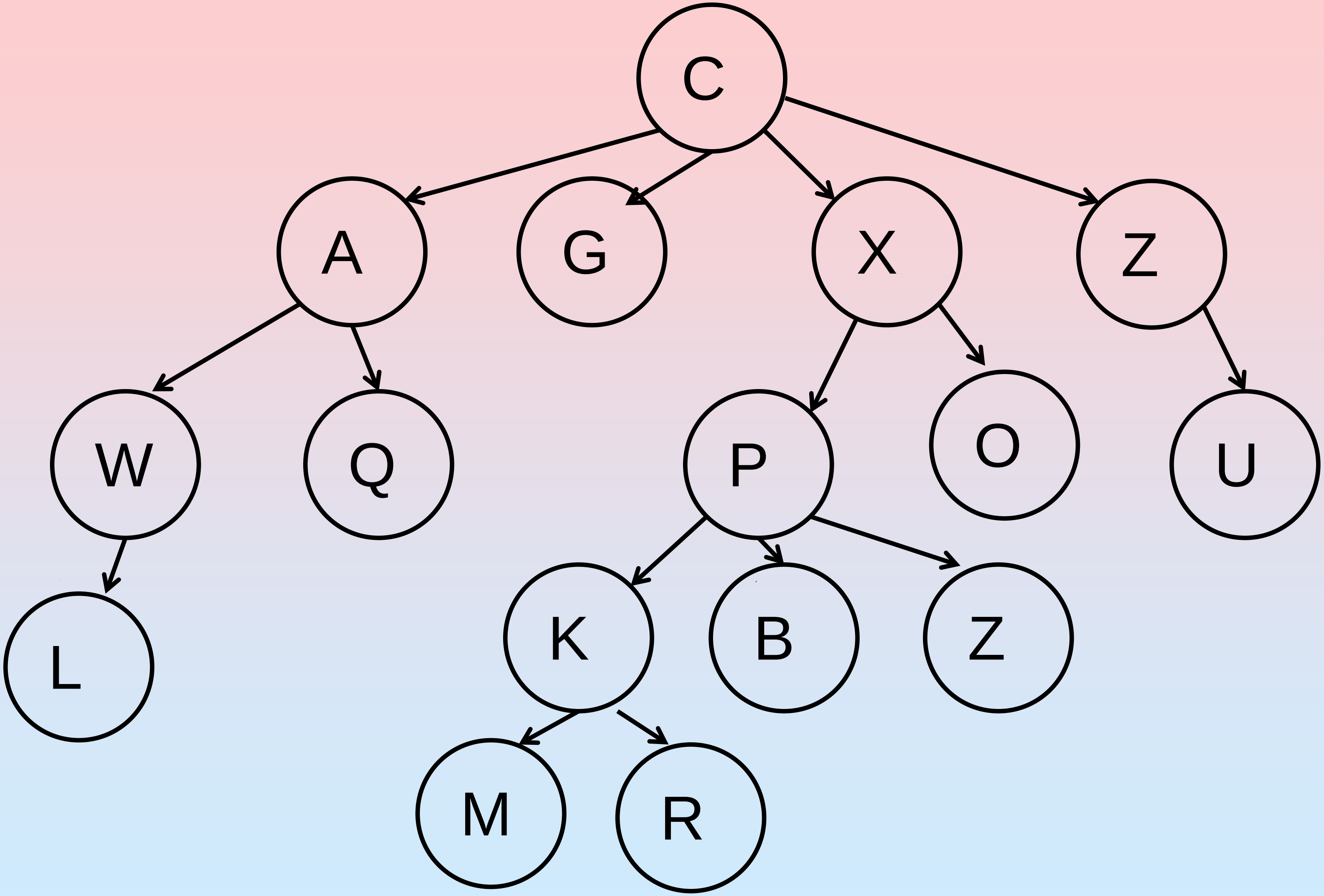
Breadth First

- 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



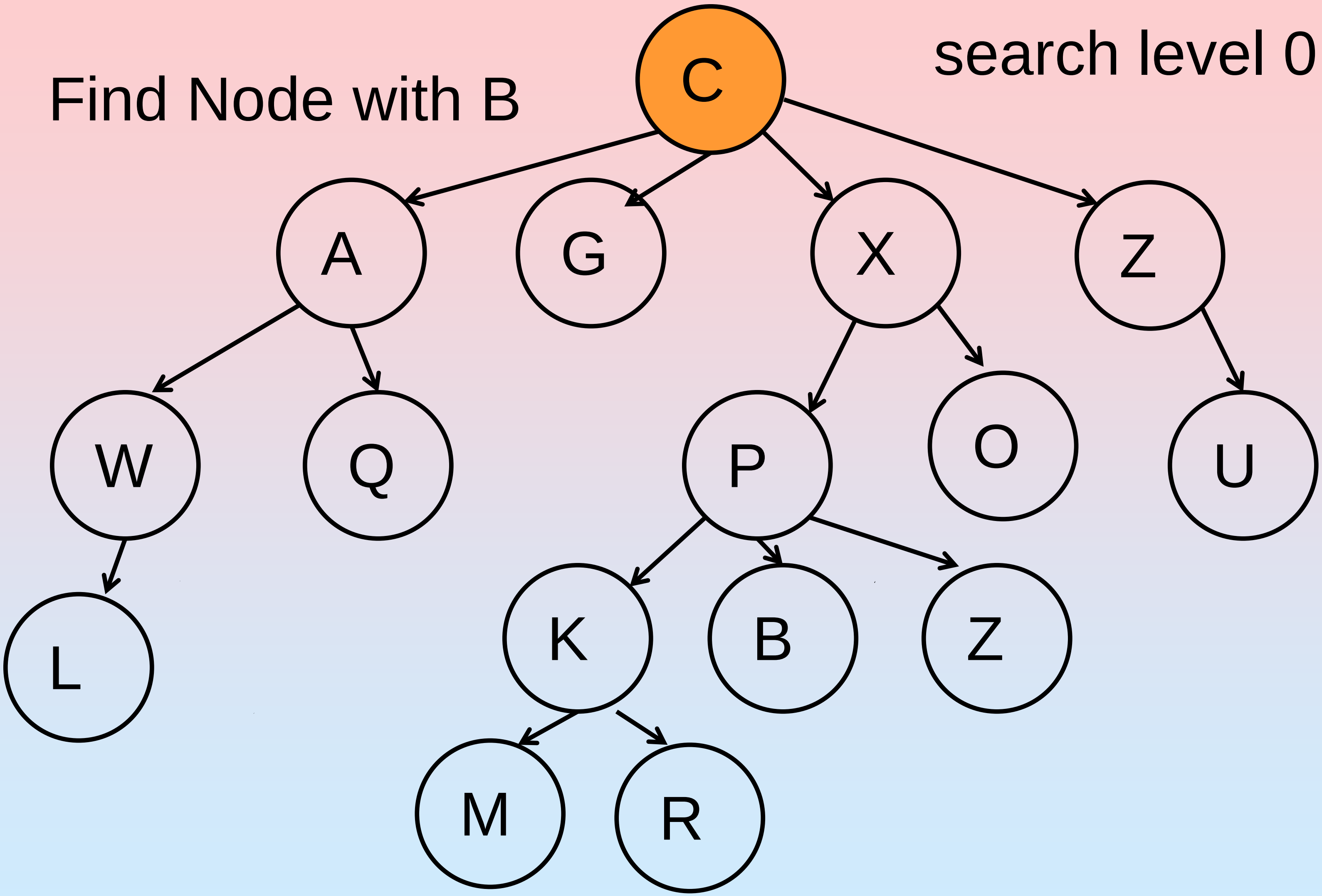
Breadth First Search of Tree



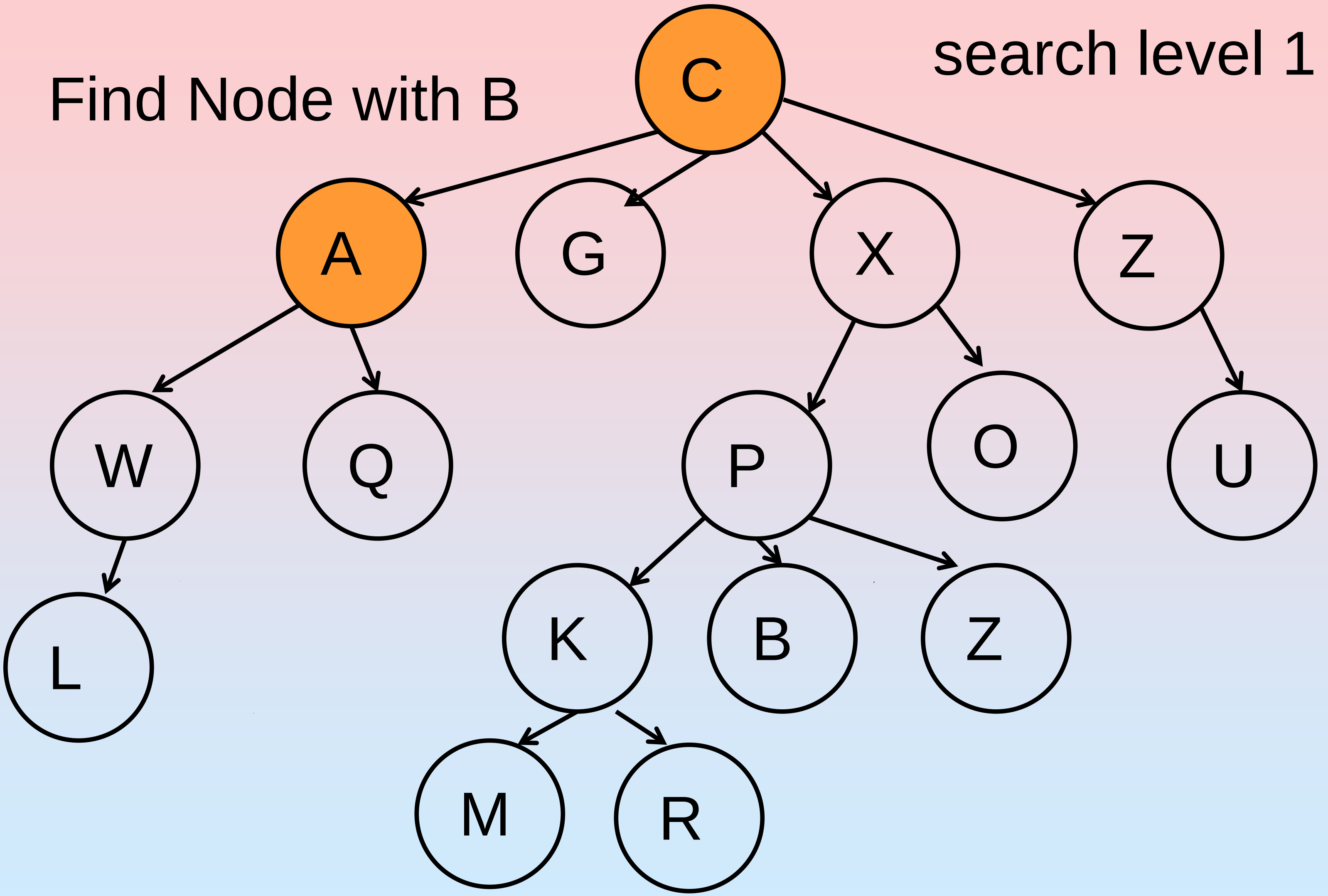
Breadth First Search of Tree

Find Node with B

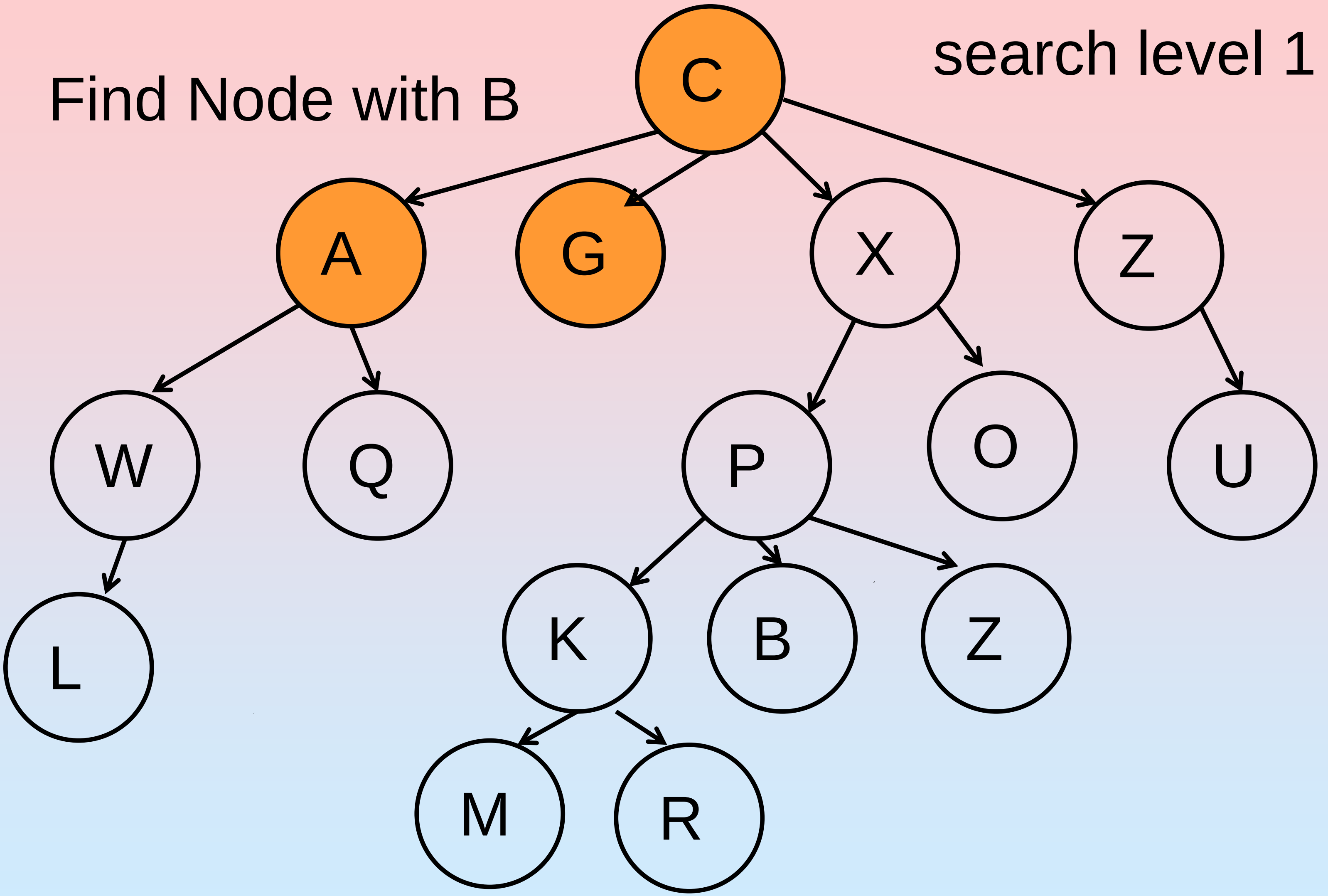
search level 0 first



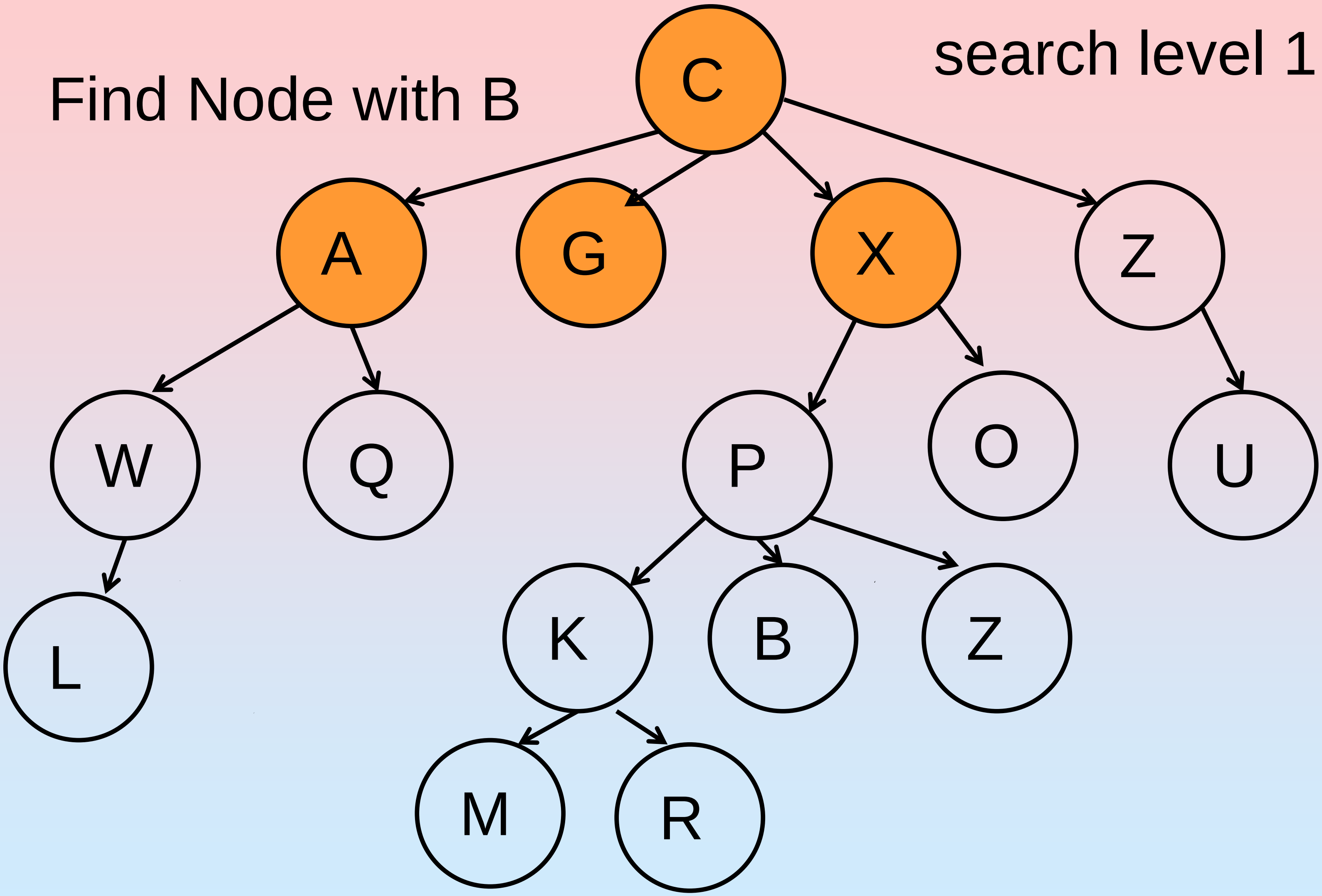
Breadth First Search of Tree



Breadth First Search of Tree



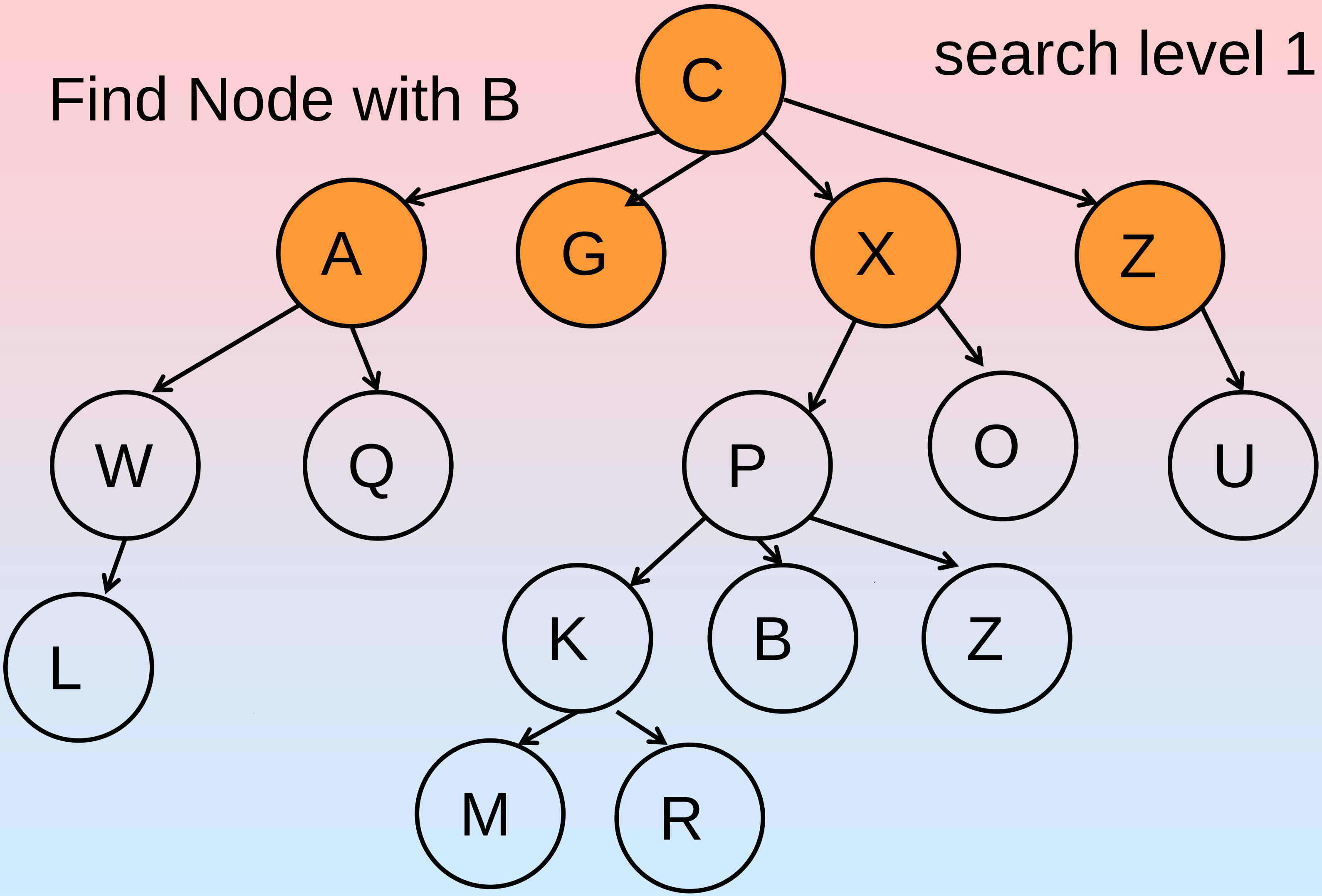
Breadth First Search of Tree



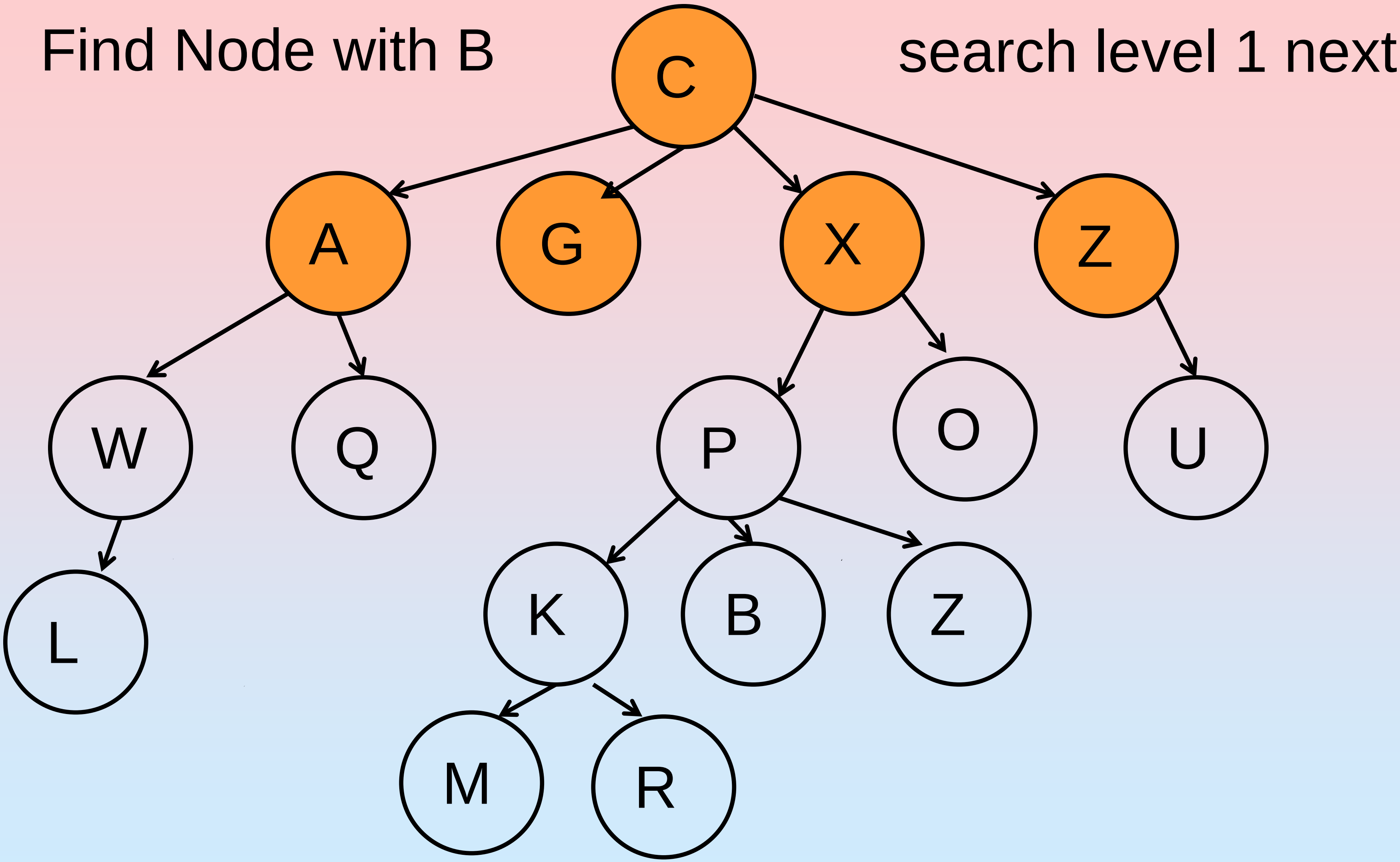
Breadth First Search of Tree

Find Node with B

search level 1



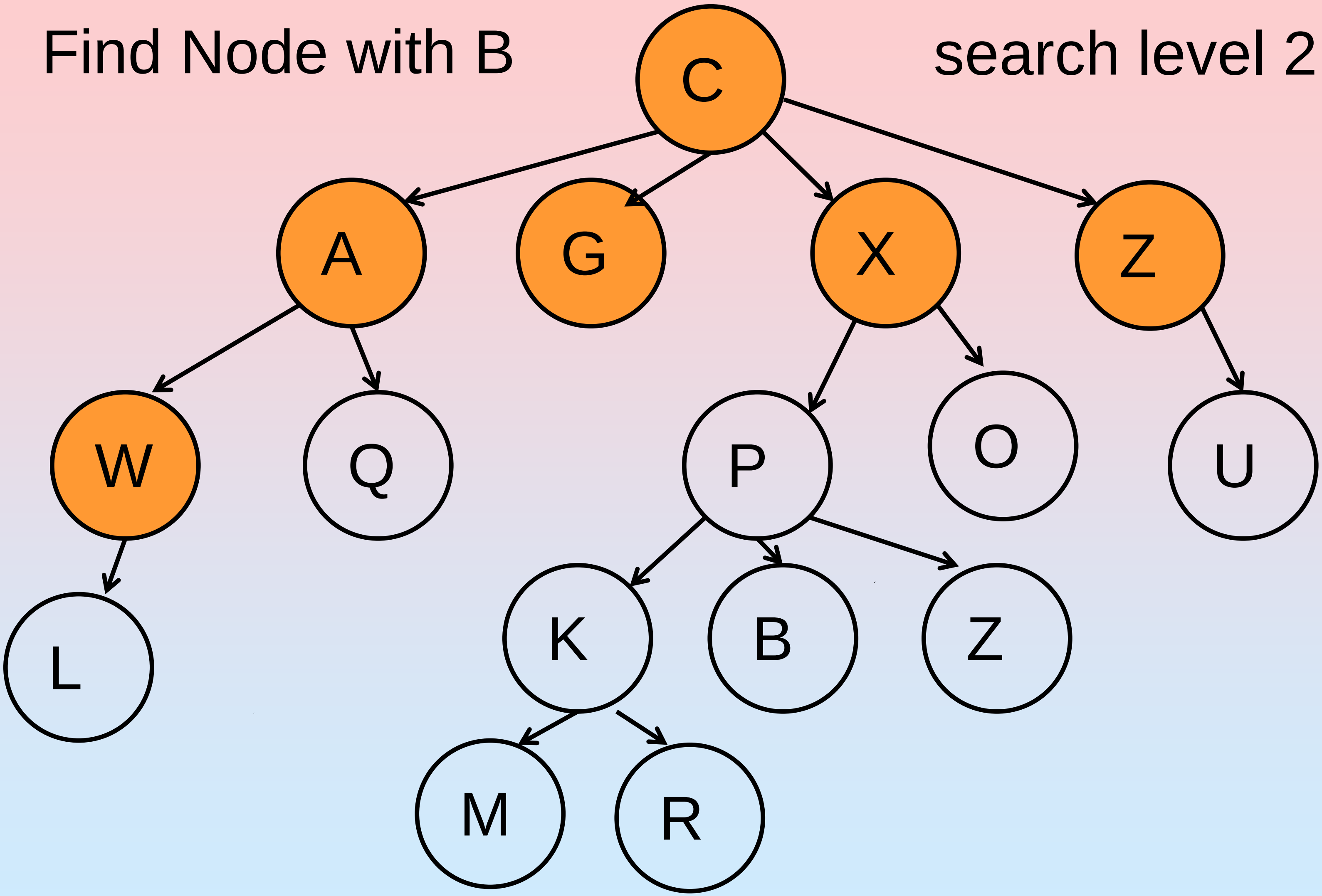
Breadth First Search of Tree



Breadth First Search of Tree

Find Node with B

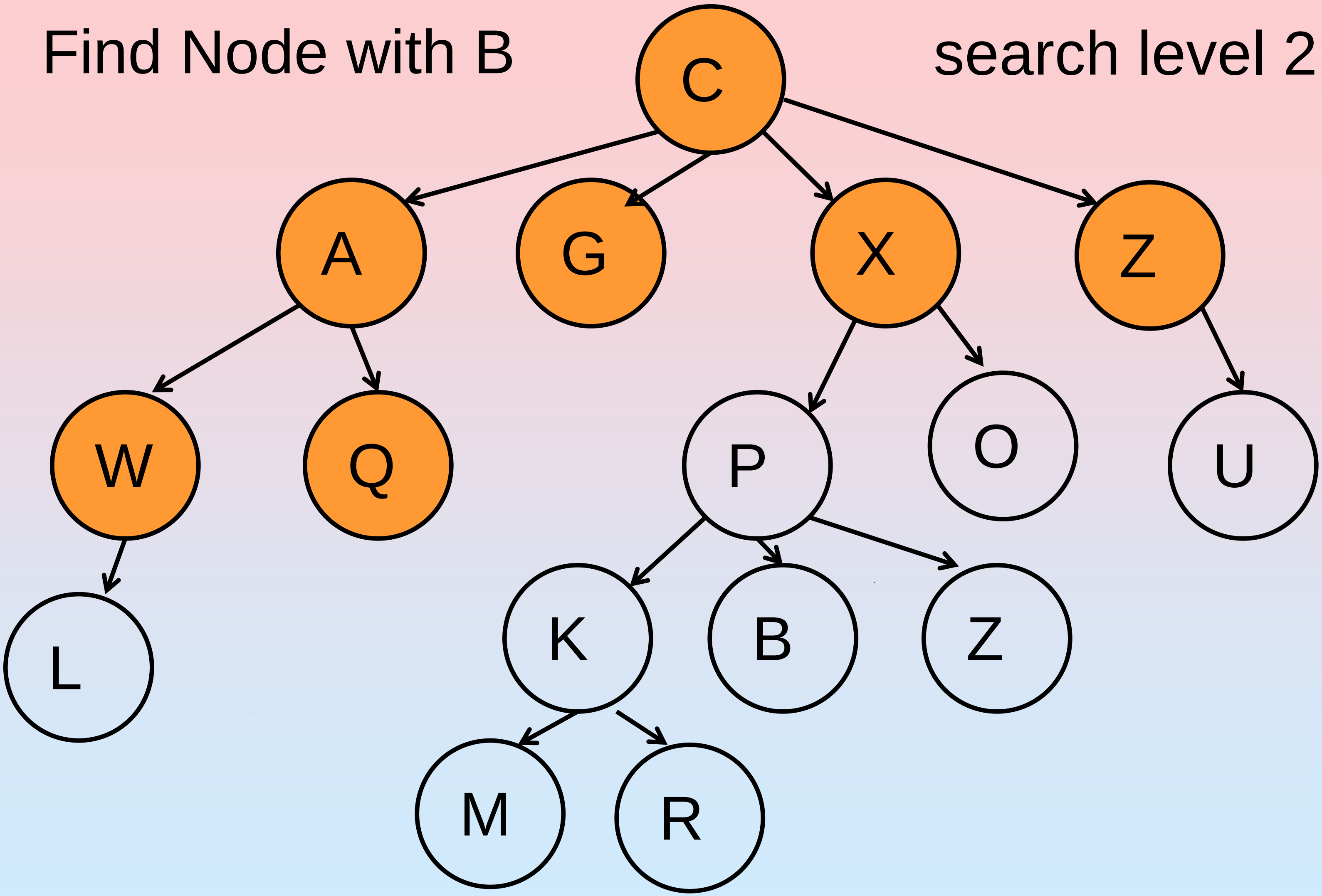
search level 2 next



Breadth First Search of Tree

Find Node with B

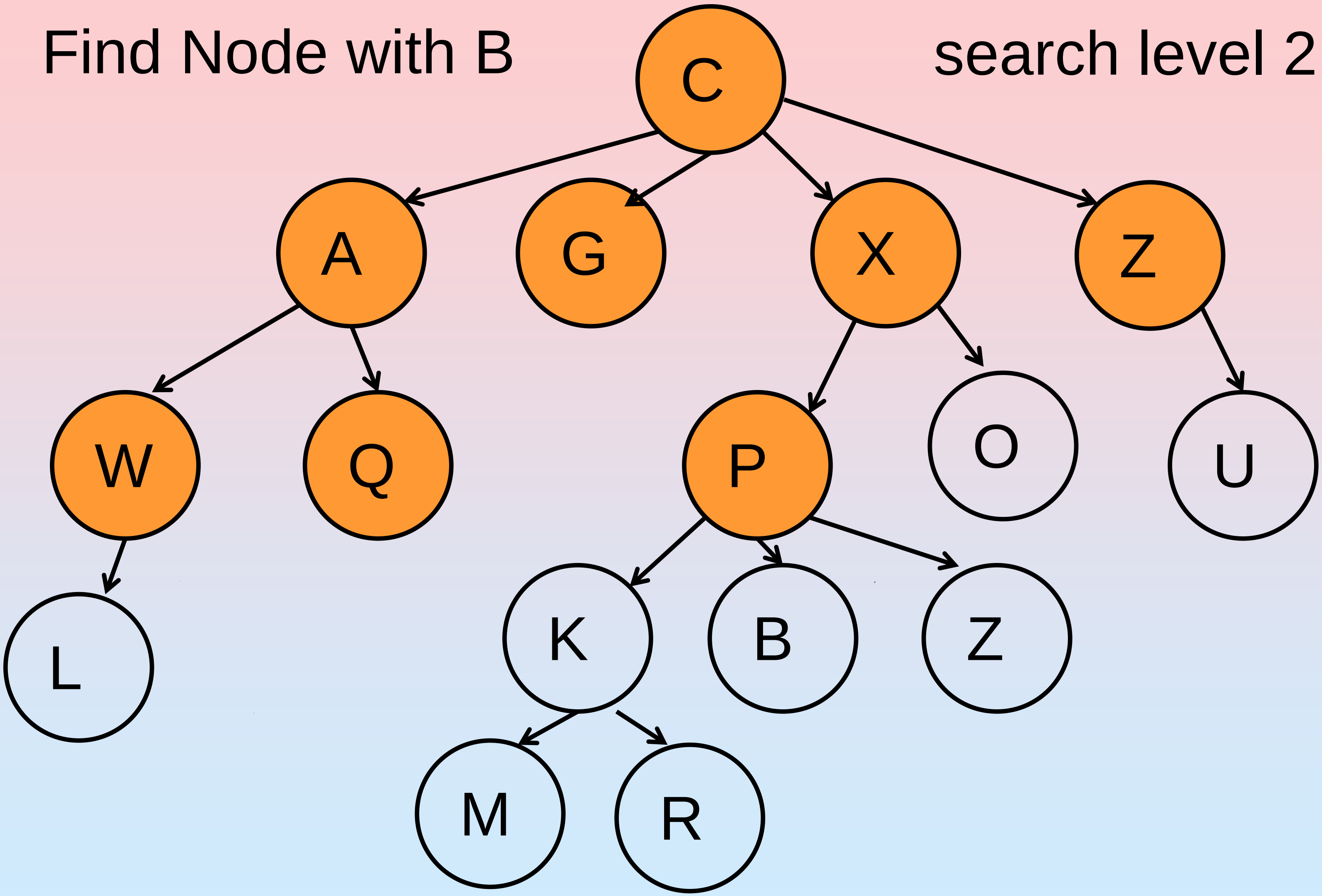
search level 2 next



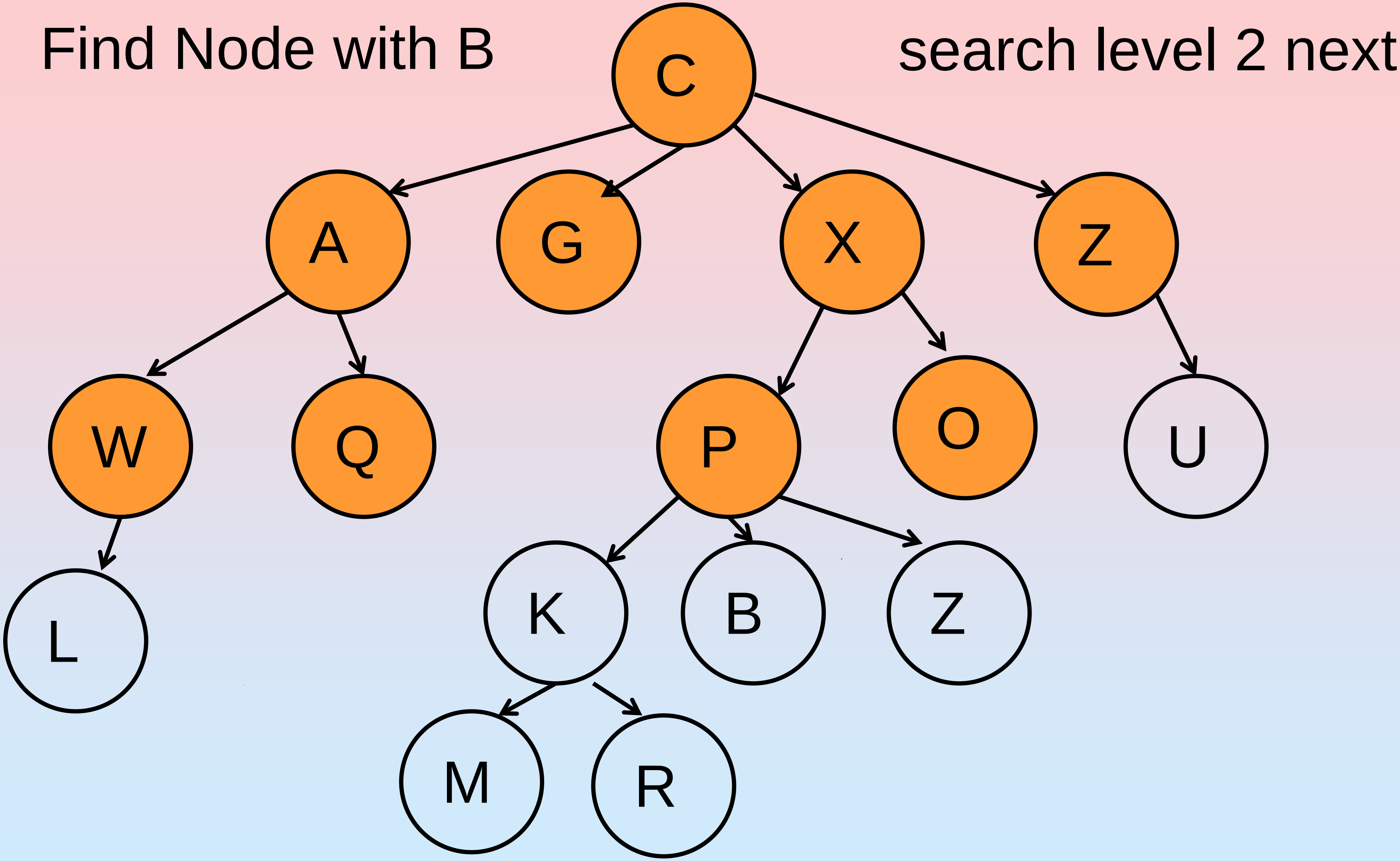
Breadth First Search of Tree

Find Node with B

search level 2 next



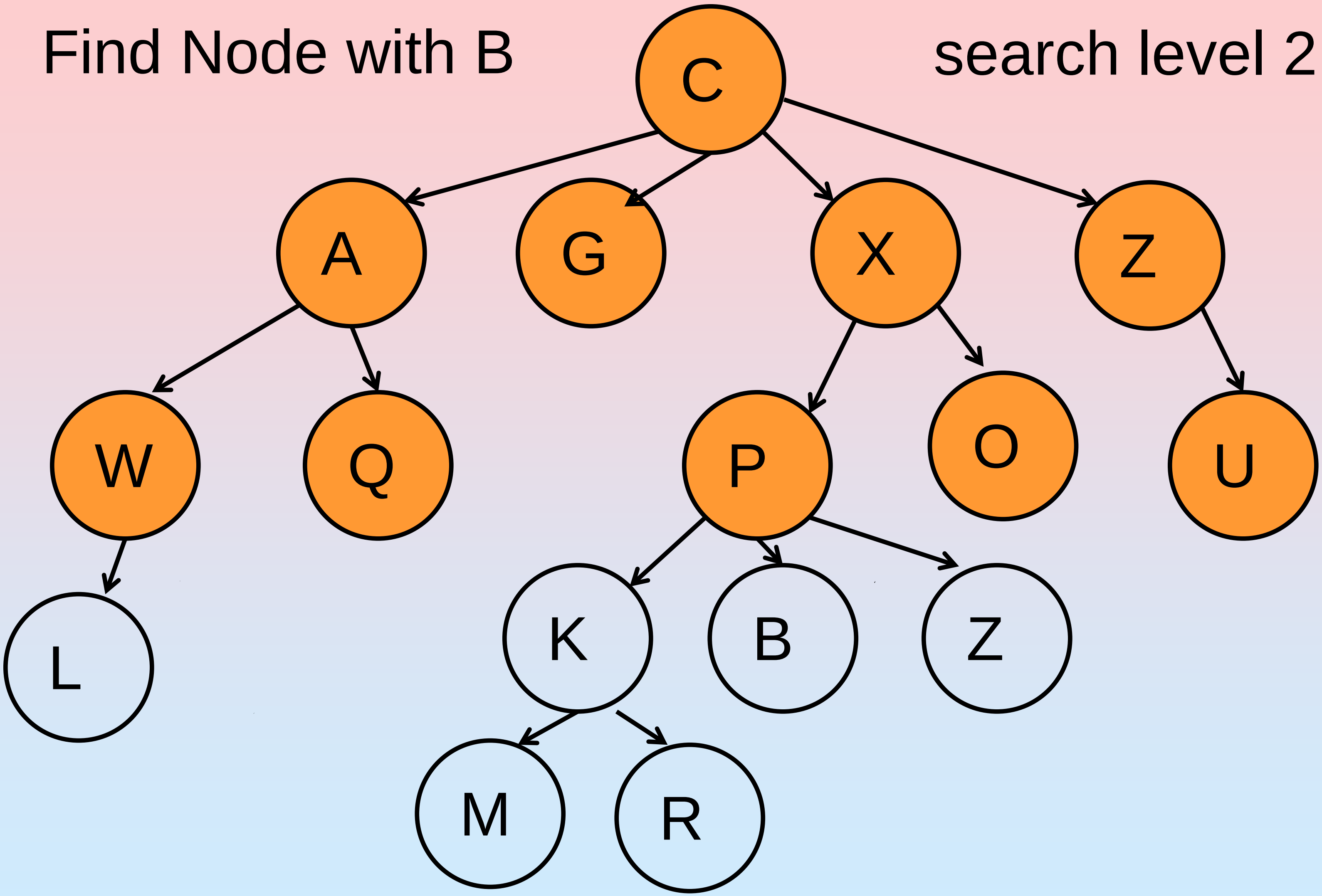
Breadth First Search of Tree



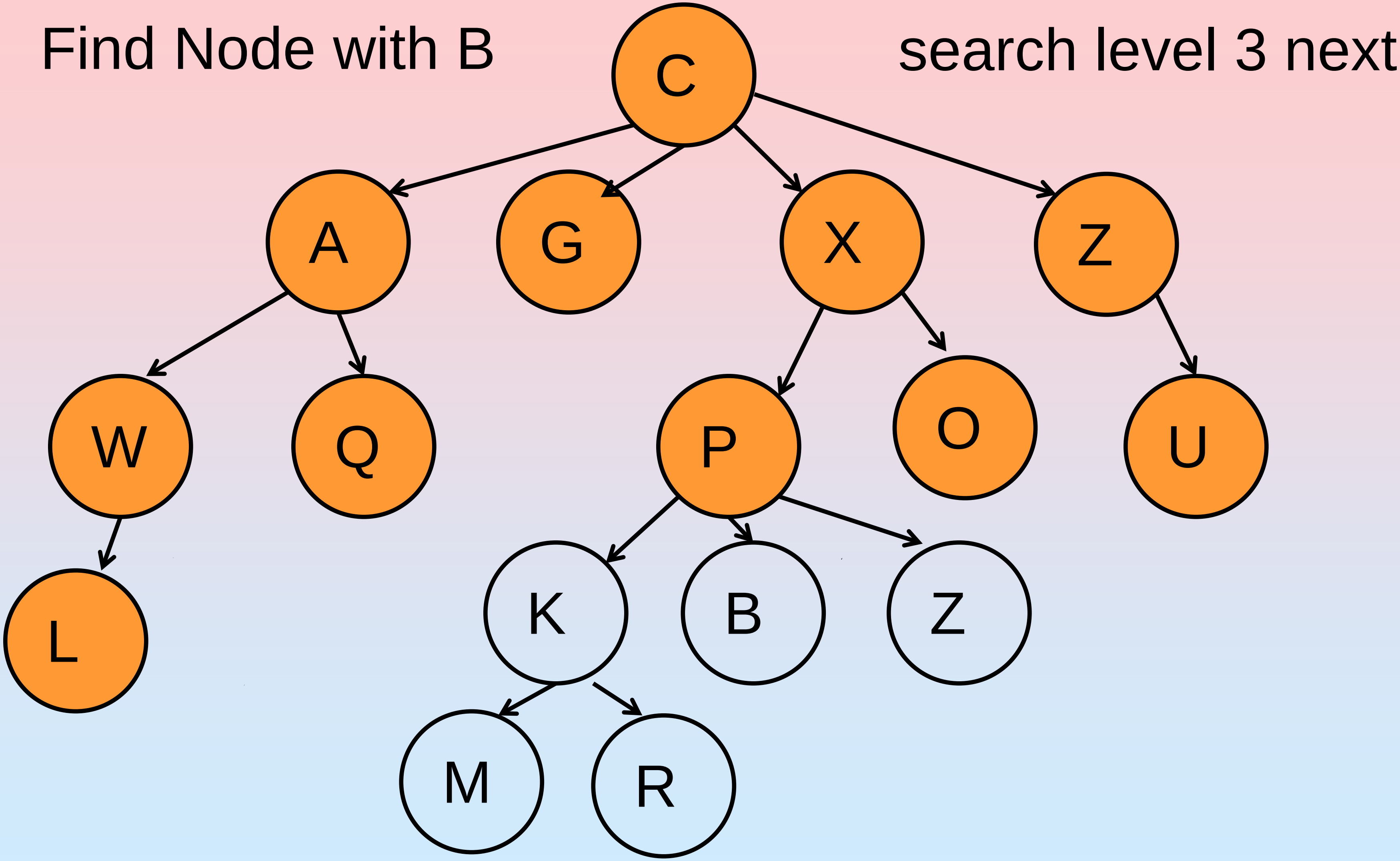
Breadth First Search of Tree

Find Node with B

search level 2 next



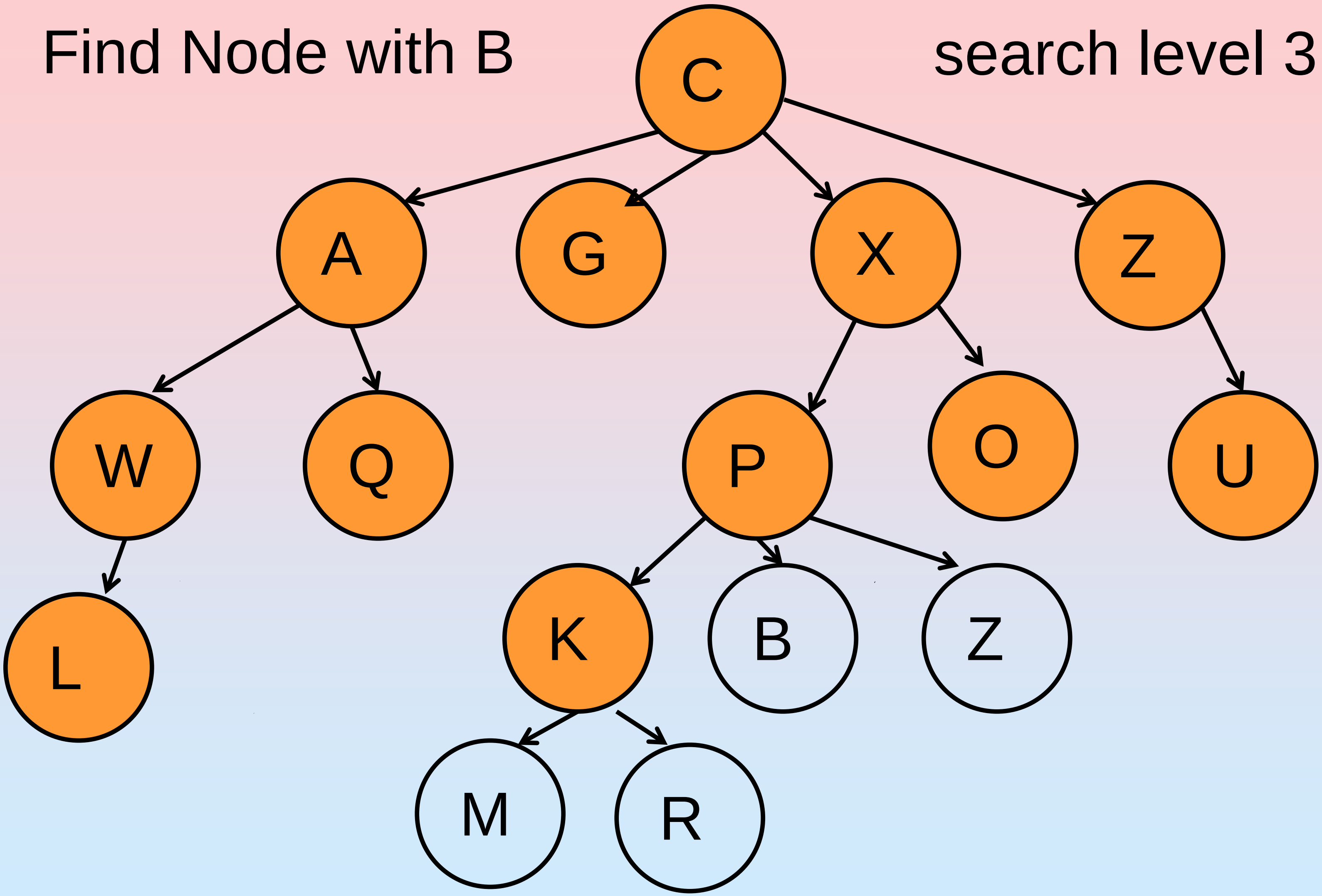
Breadth First Search of Tree



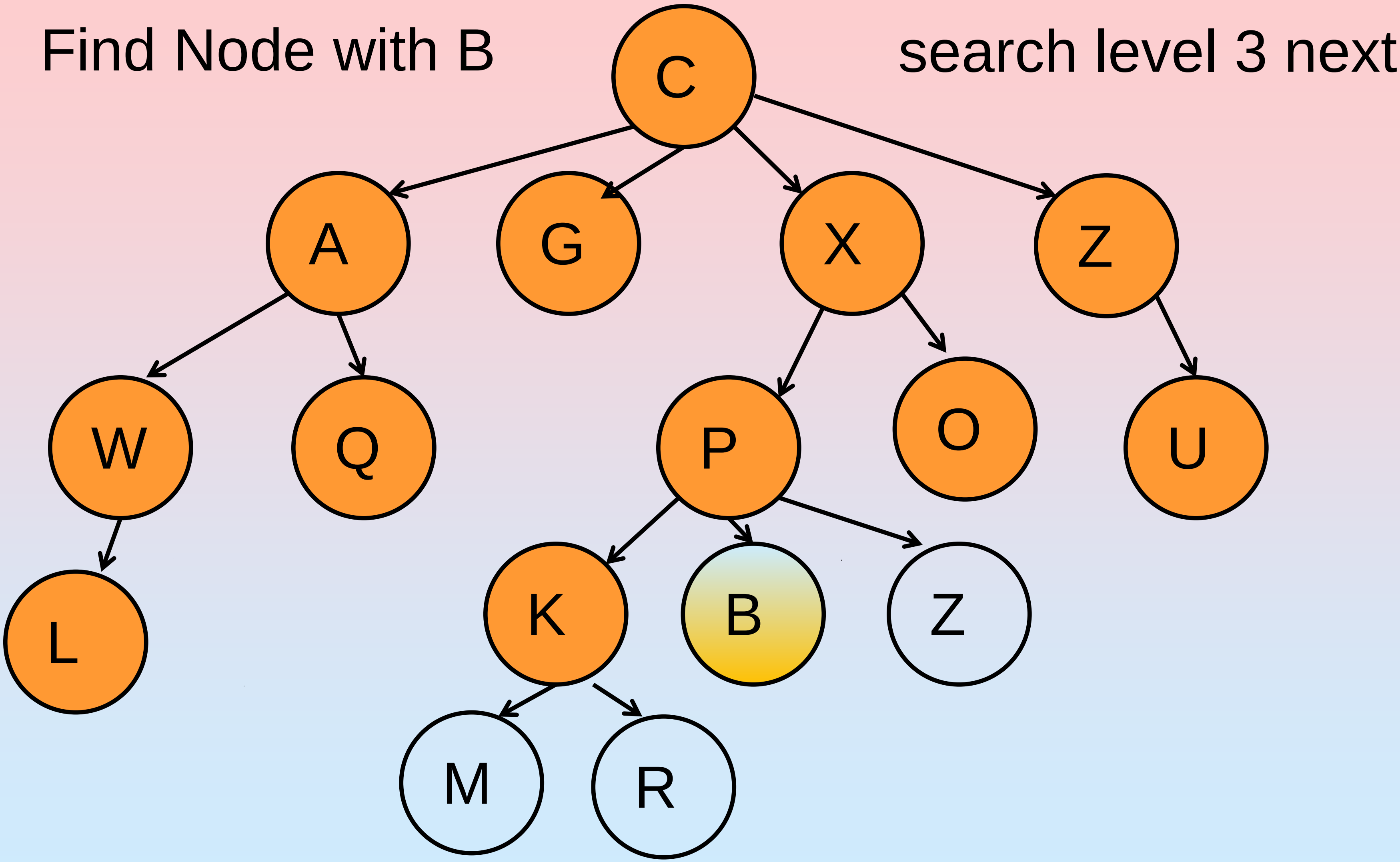
Breadth First Search of Tree

Find Node with B

search level 3 next



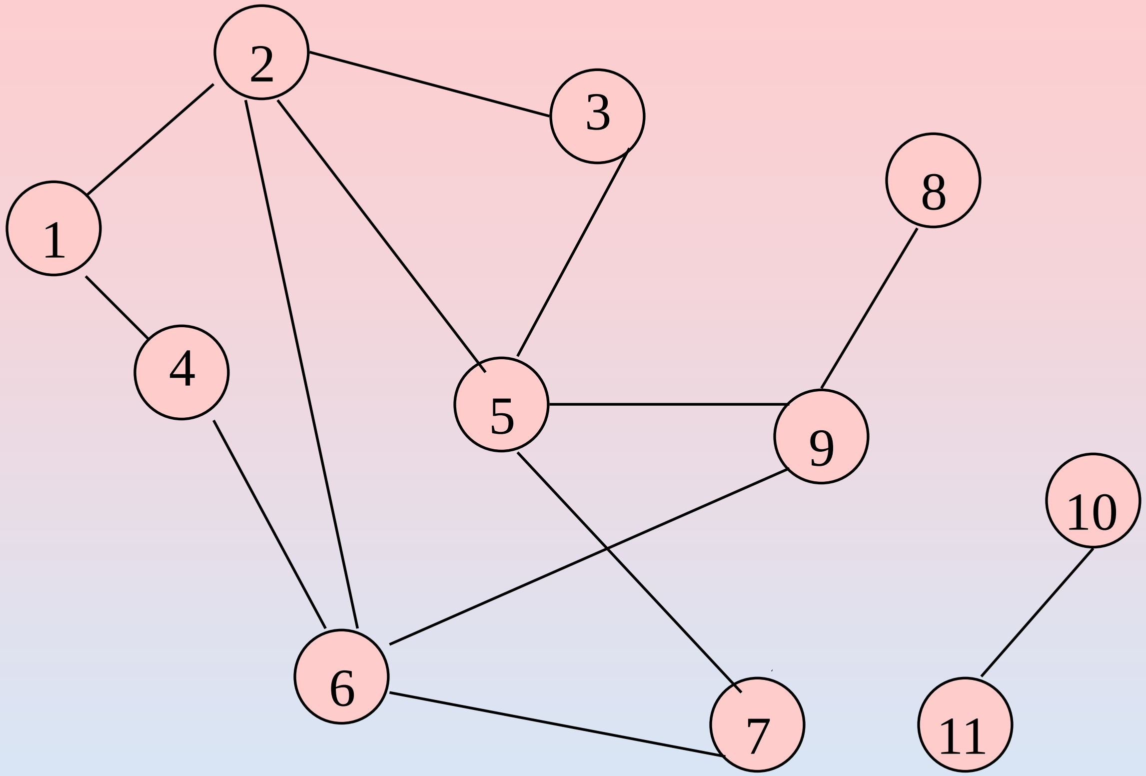
Breadth First Search of Tree



Breadth-First Search Implementaion

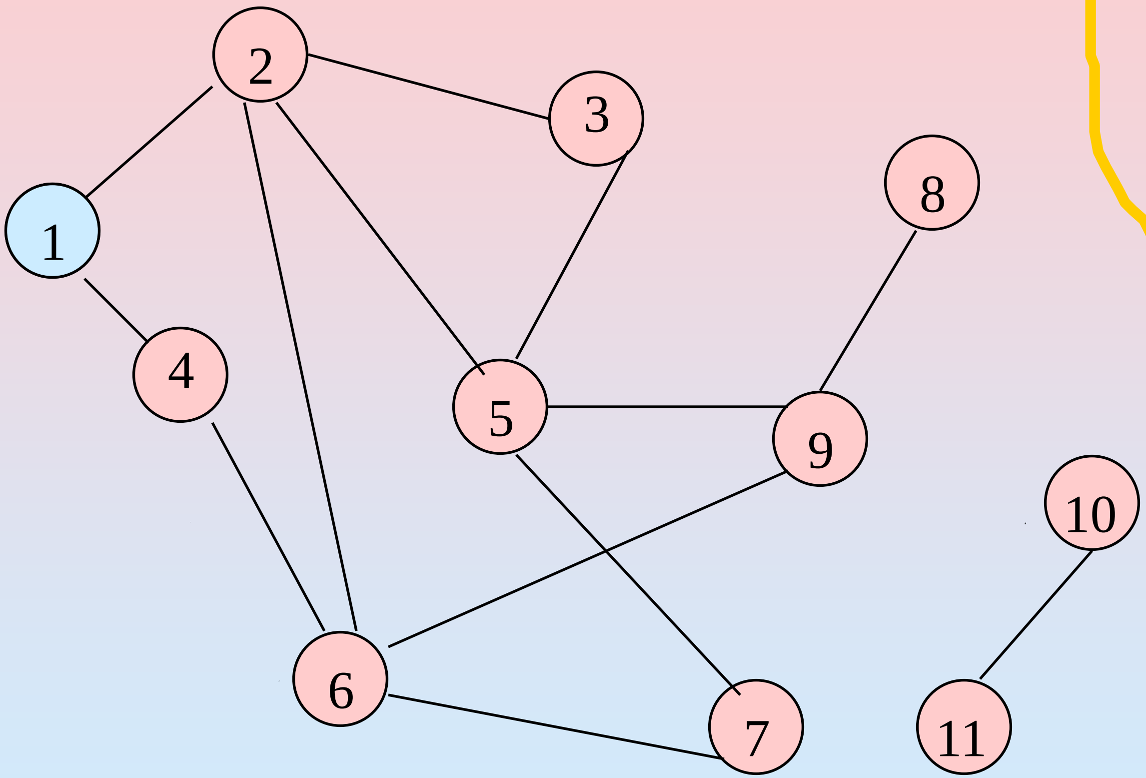
- **Visit start vertex and put into a FIFO queue.**
- **Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue.**

Breadth-First Search of Graph



Start search at vertex 1.

Breadth-First Search of Graph

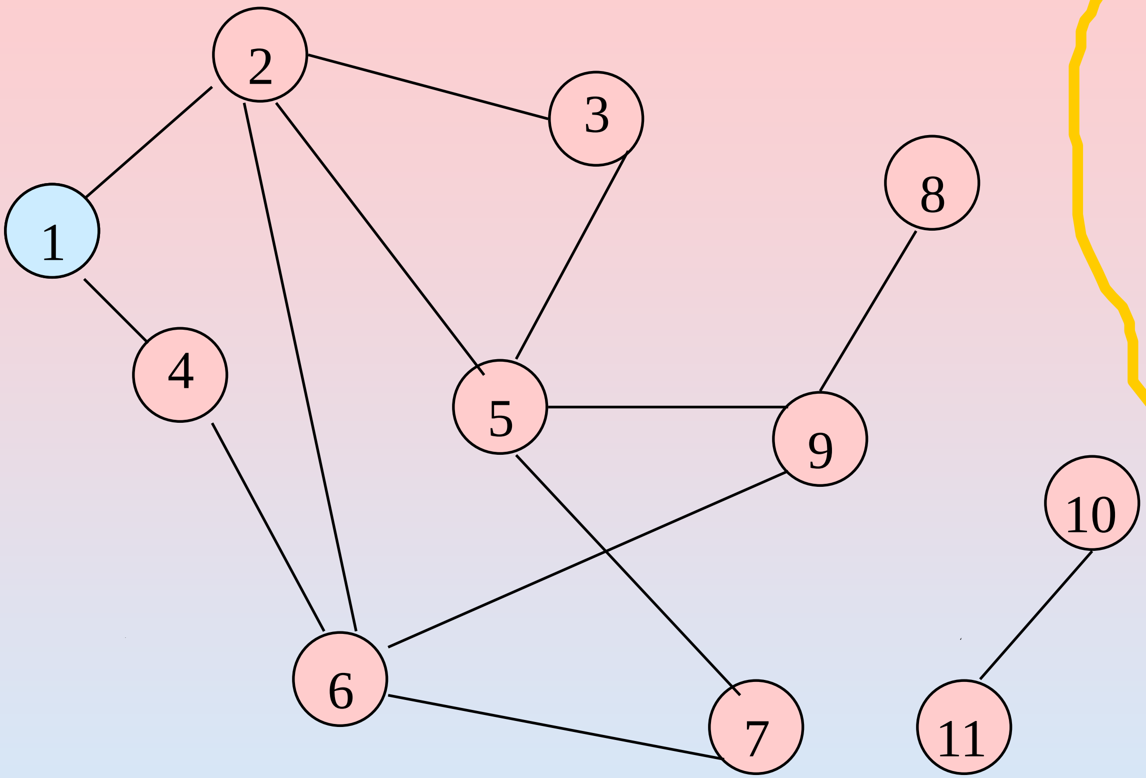


FIFO Queue

1

Visit/mark/label start vertex and put in a FIFO queue.

Breadth-First Search of Graph

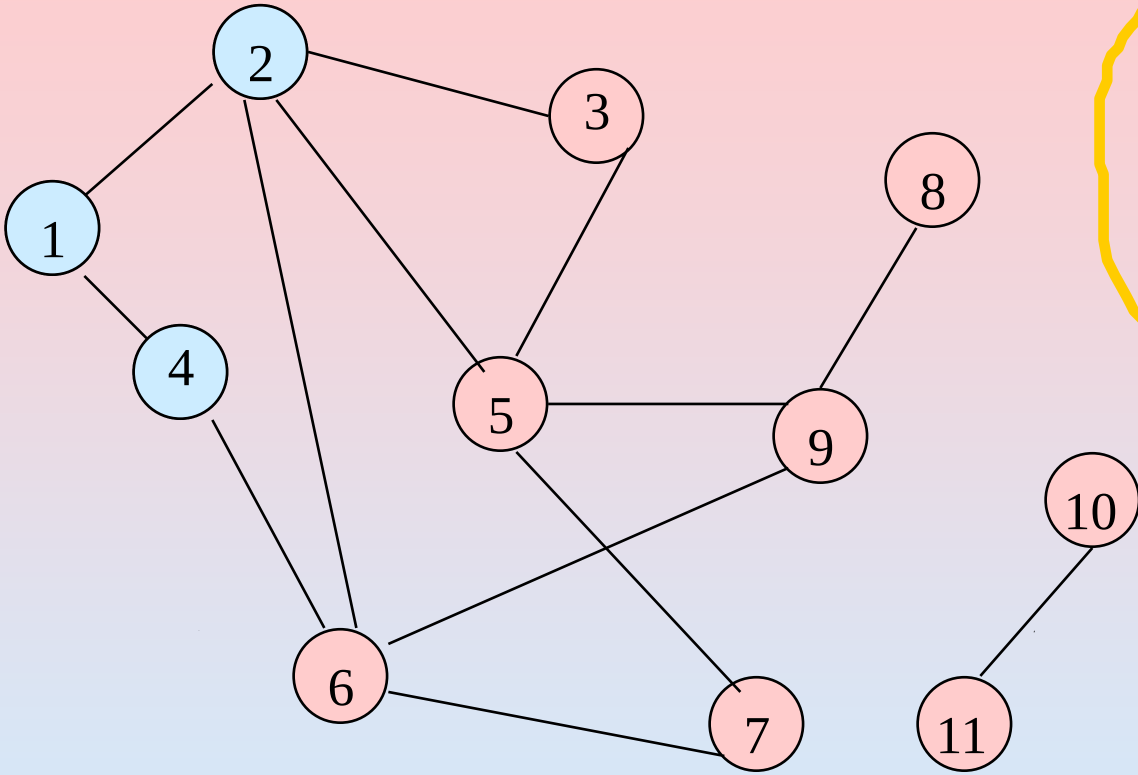


FIFO Queue

1

Remove 1 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



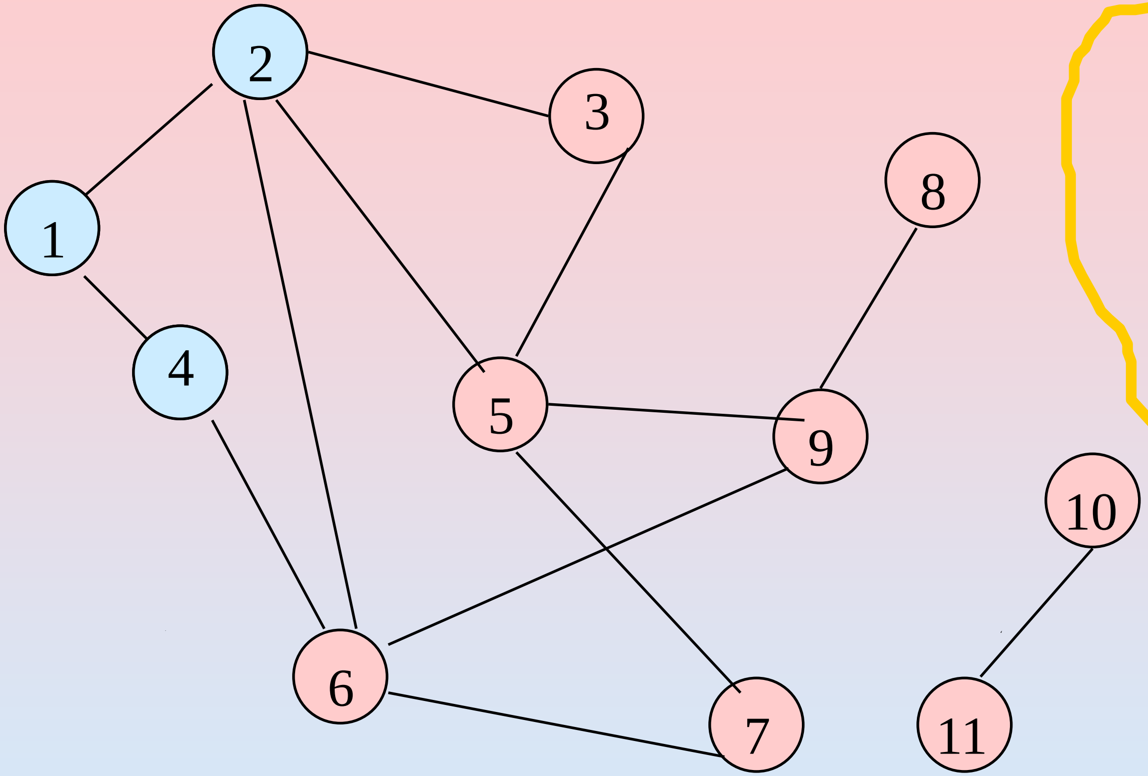
FIFO Queue

2 4

Remove 1 from Queue; visit adjacent unvisited vertices;

put in Queue.

Breadth-First Search of Graph



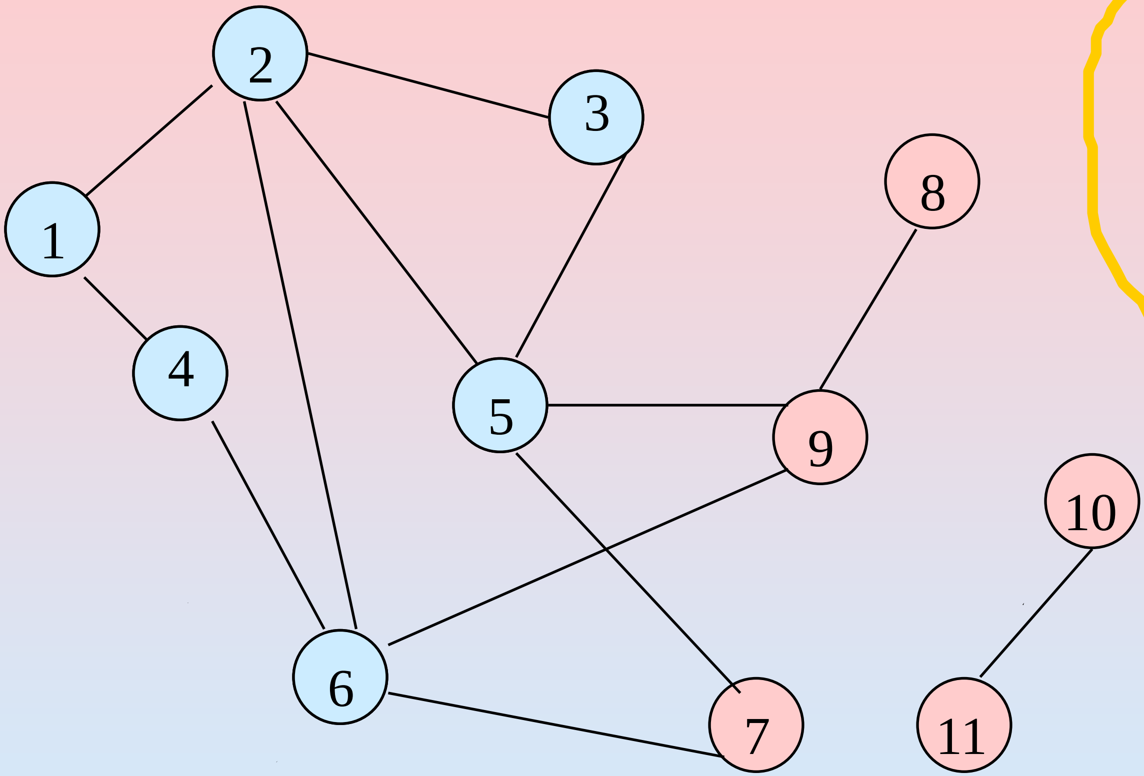
FIFO Queue

2 4

Remove 2 from Queue; visit adjacent unvisited vertices;

put in Queue.

Breadth-First Search of Graph

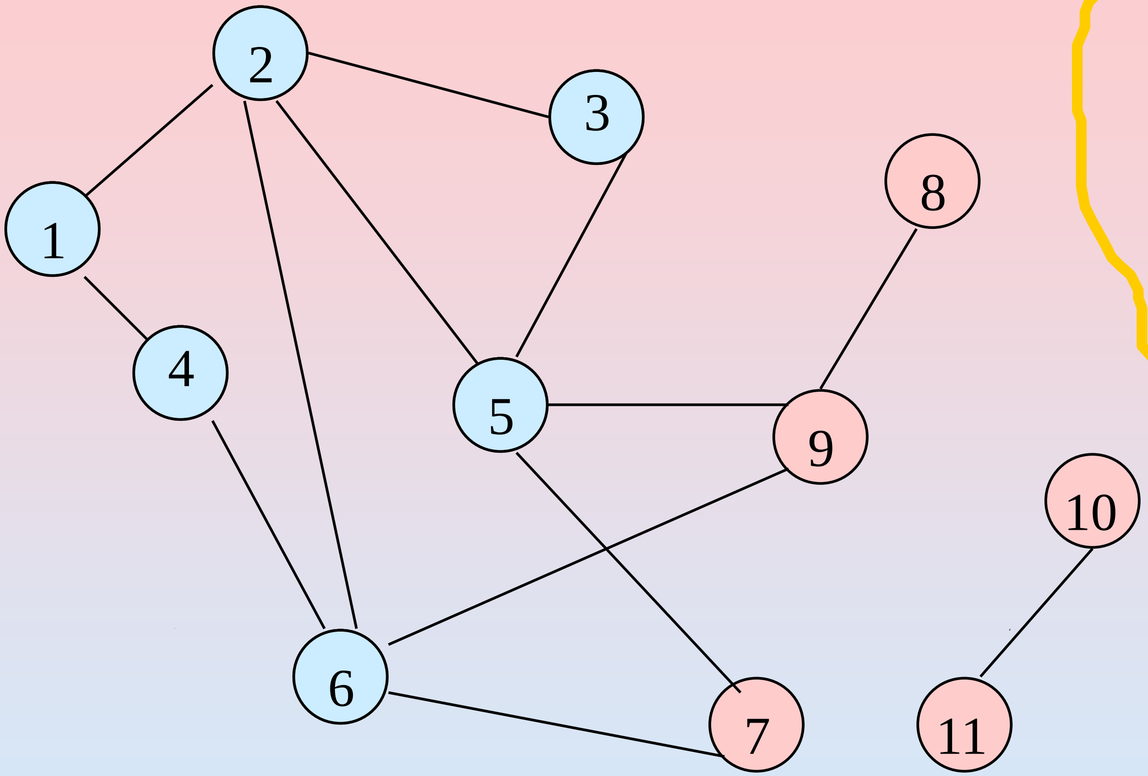


FIFO Queue

4 5 3 6

Remove 2 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph

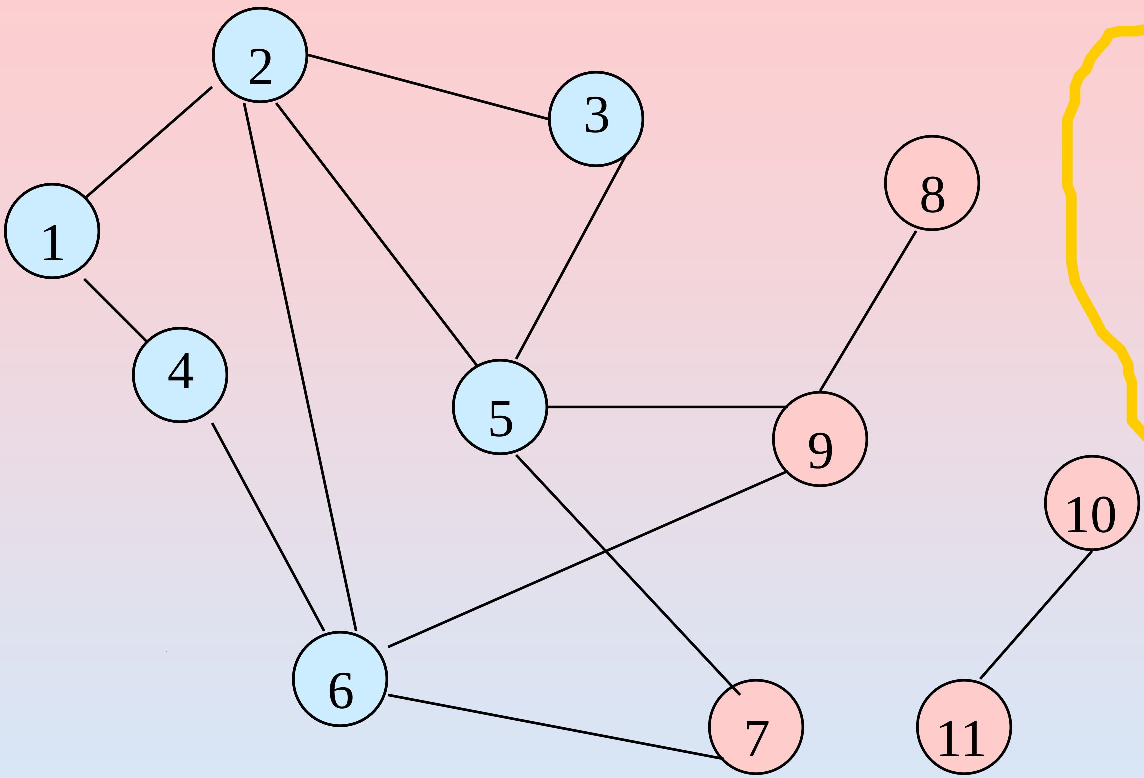


FIFO Queue

4 5 3 6

Remove 4 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph

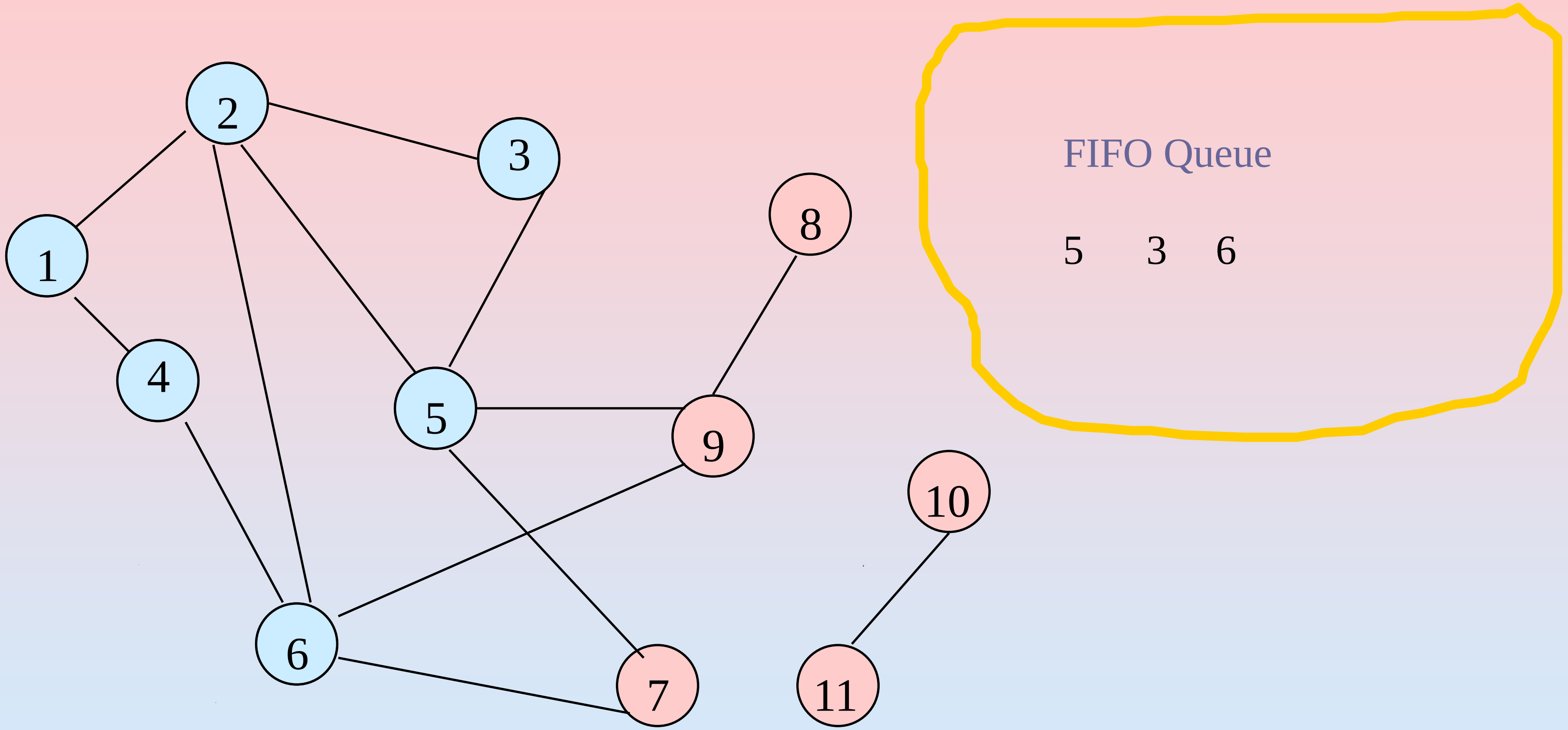


FIFO Queue

5 3 6

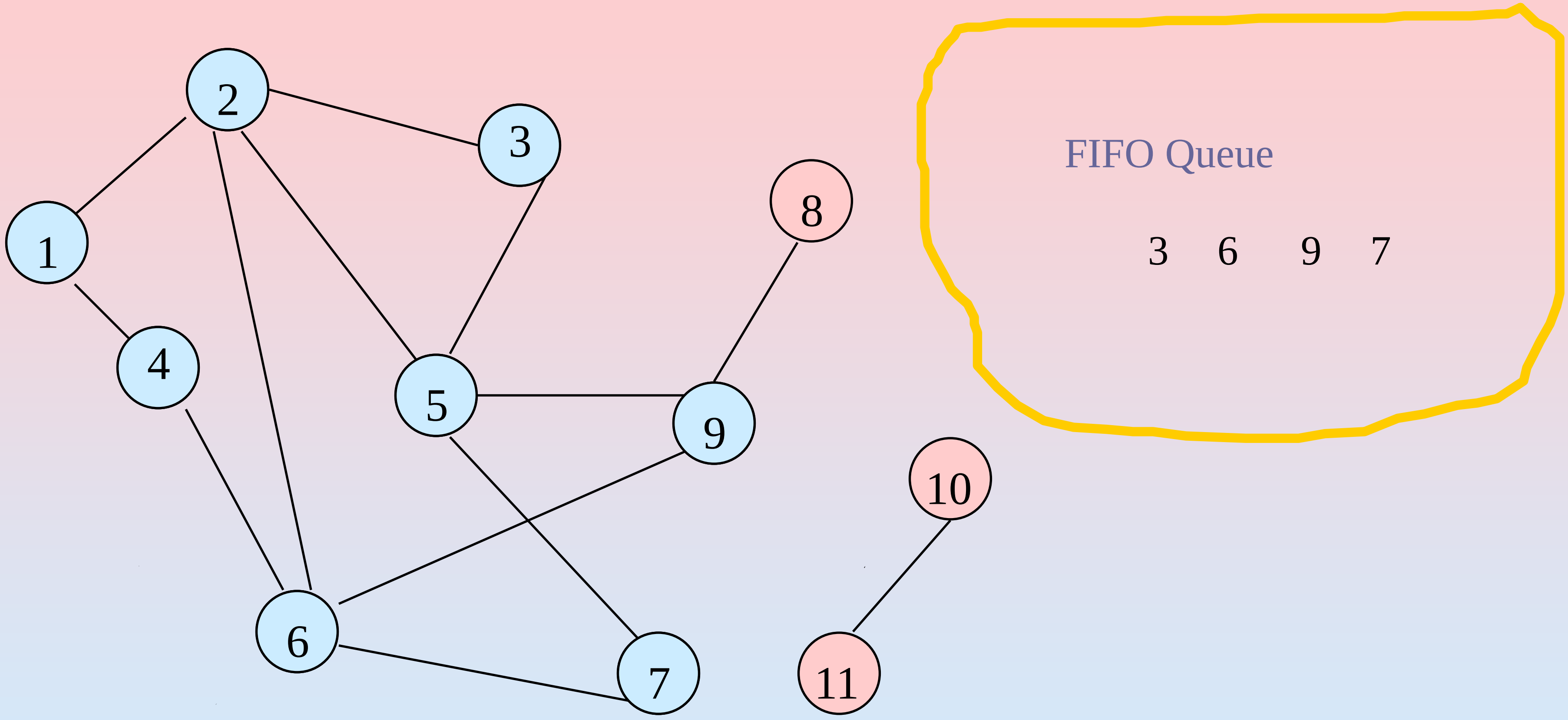
Remove 4 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



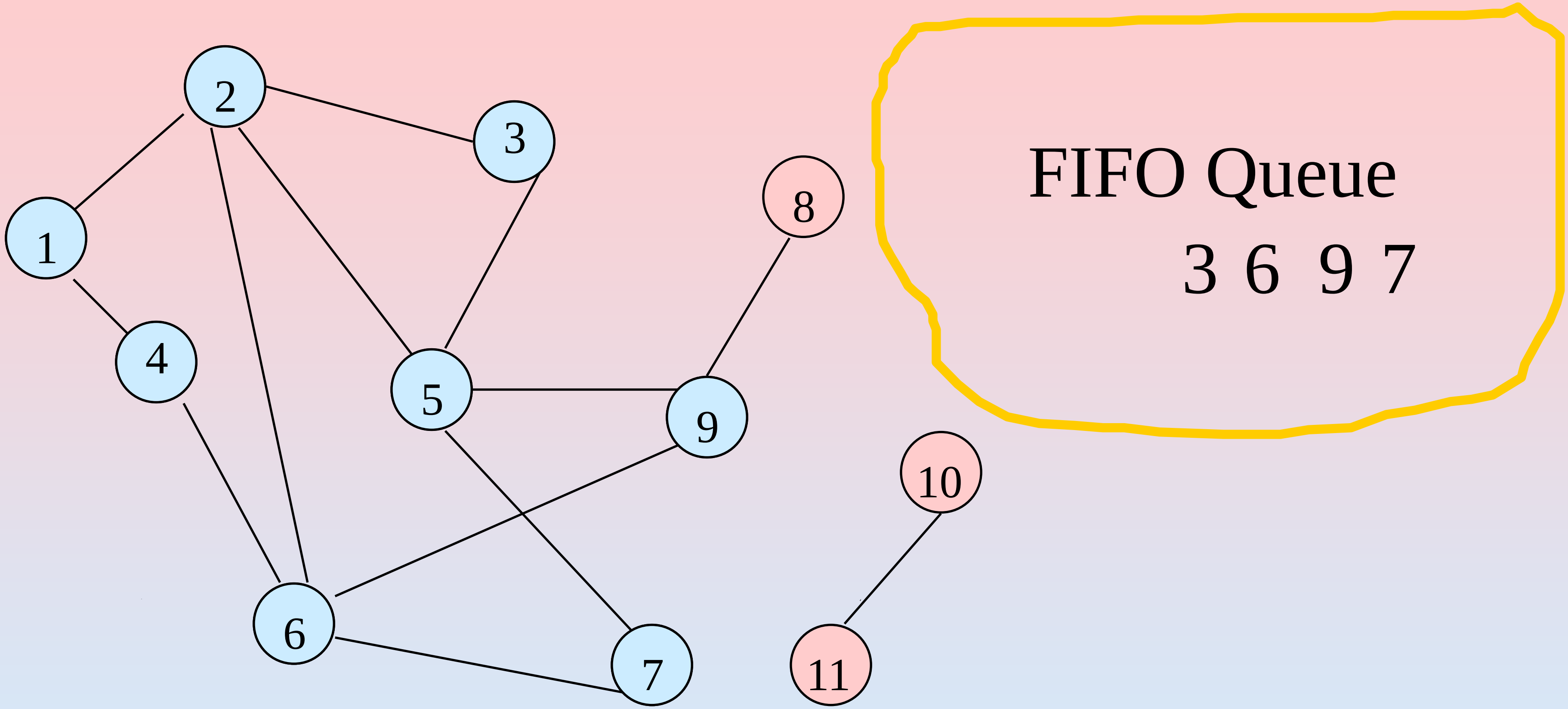
Remove **5** **from** **Queue; visit adjacent unvisited vertices; put in Queue.**

Breadth-First Search of Graph



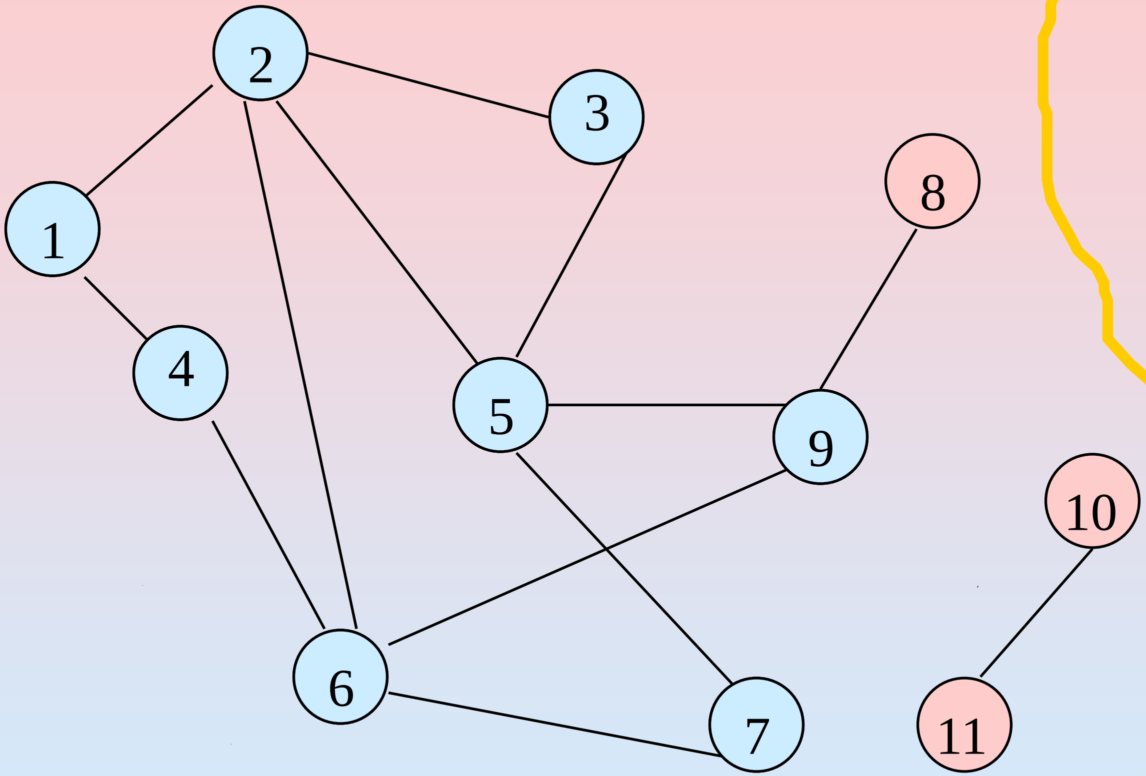
Remove 5 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



Remove 3 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph

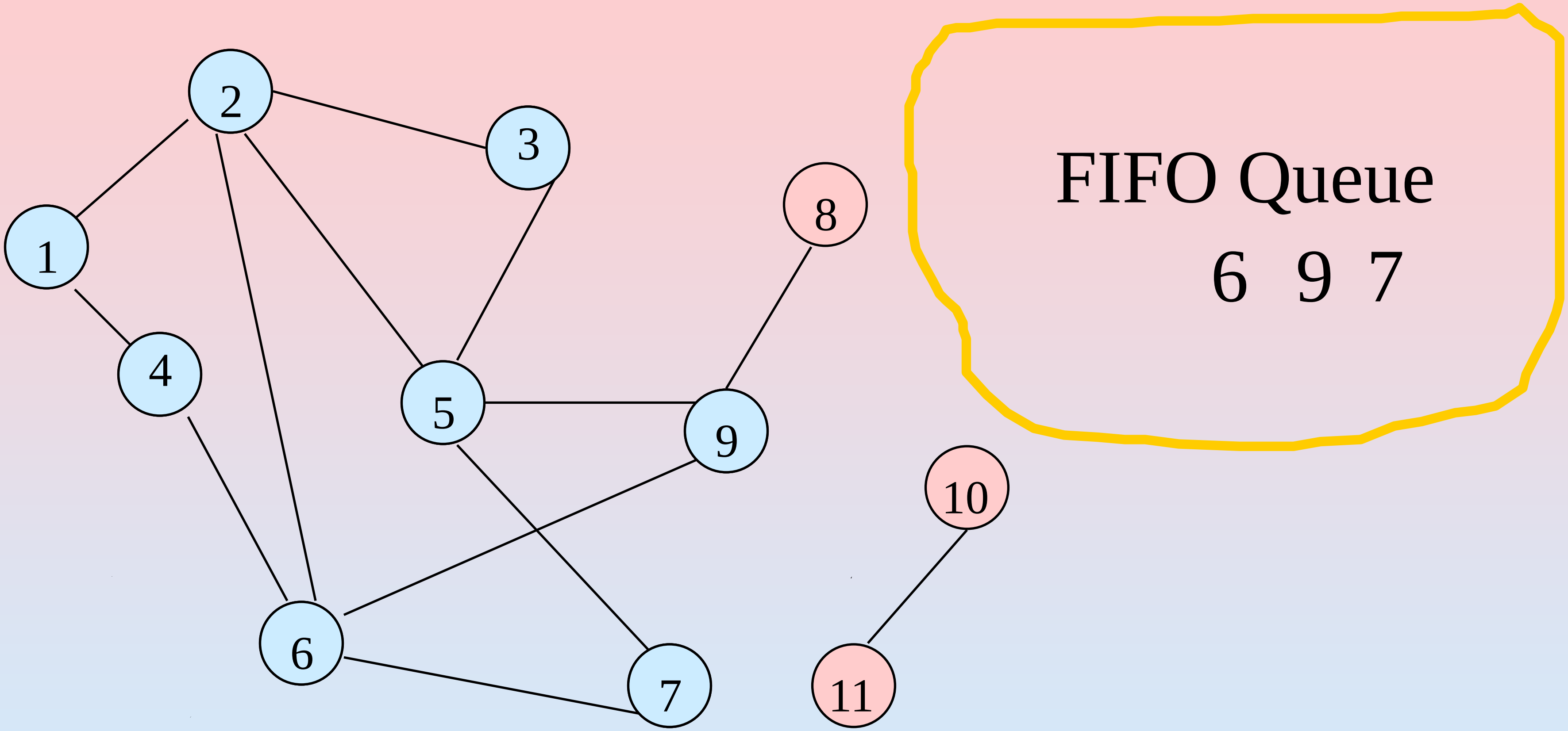


FIFO Queue

6 9 7

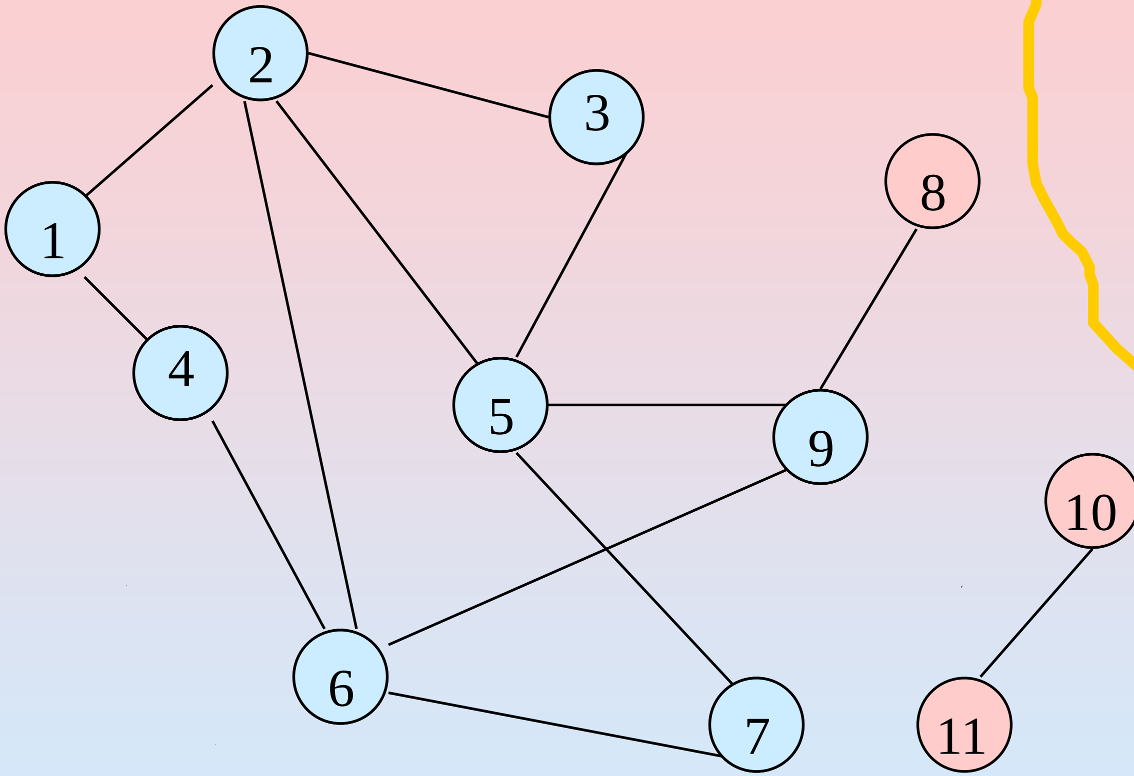
Remove 3 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



Remove 6 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph

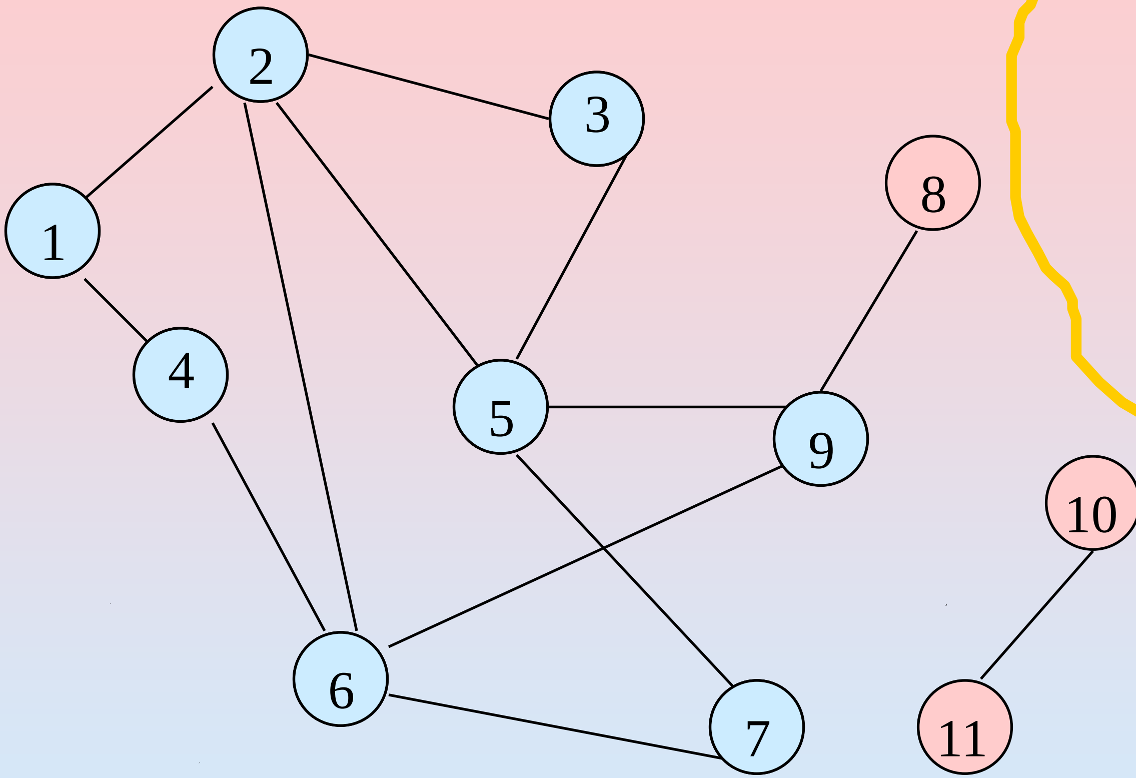


FIFO Queue

9 7

Remove 6 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph

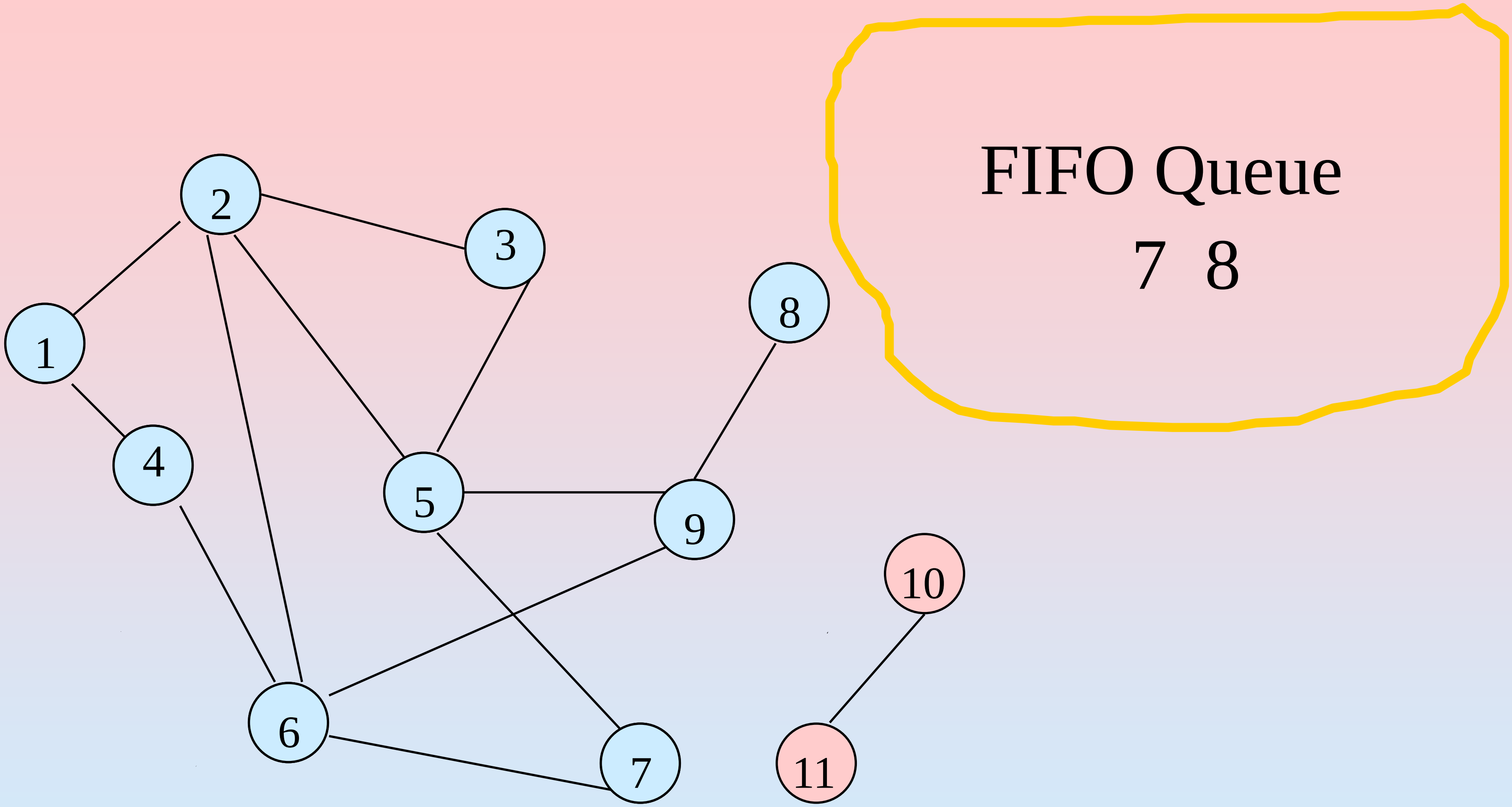


FIFO Queue

9 7

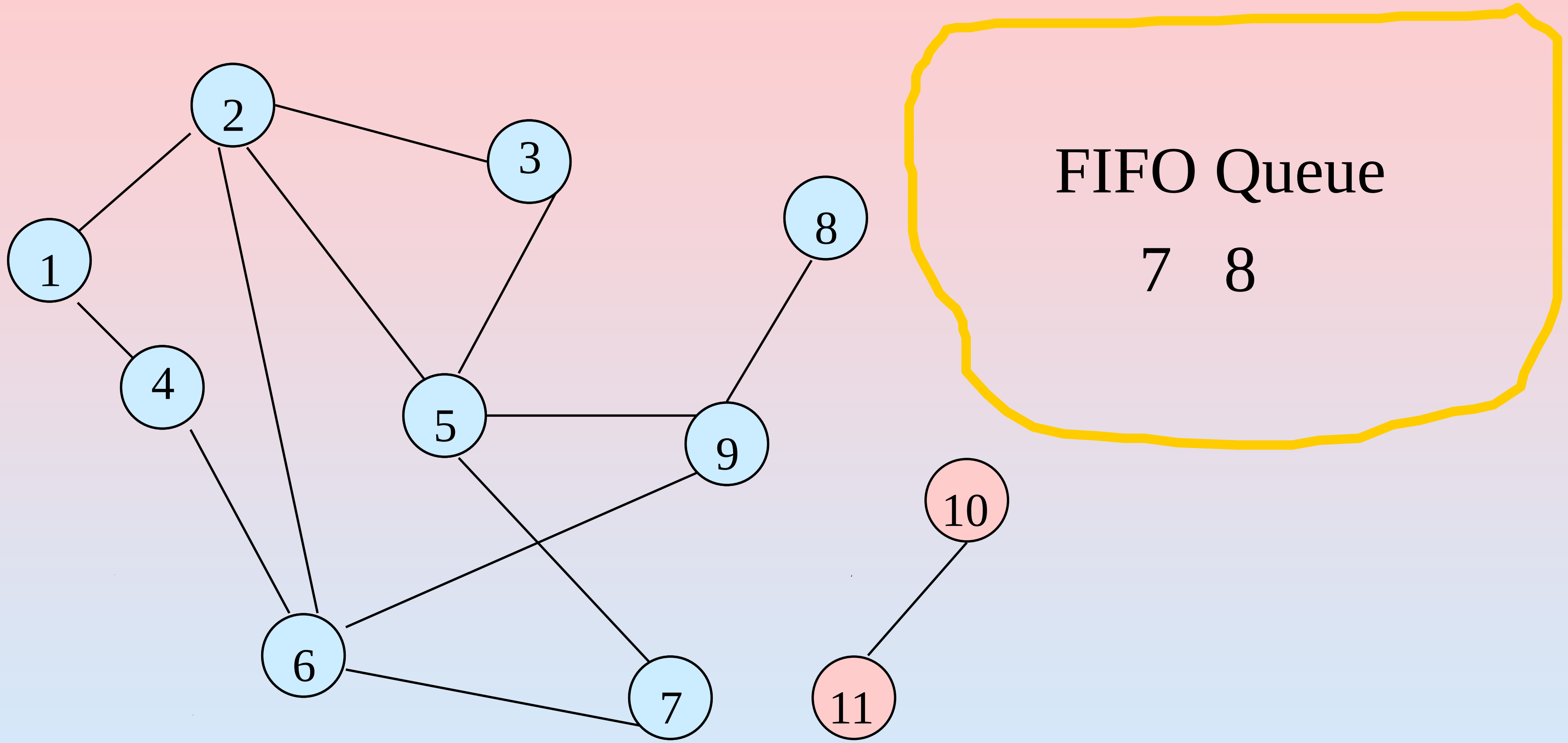
Remove 9 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



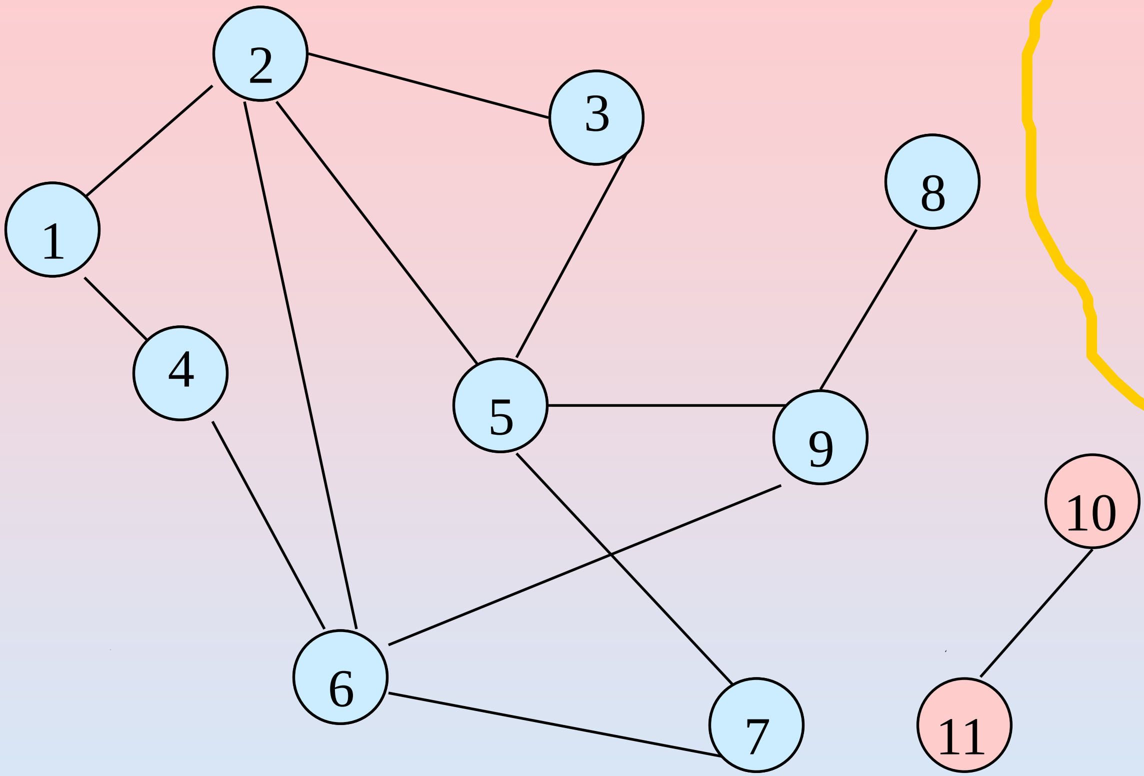
Remove 9 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



Remove 7 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph

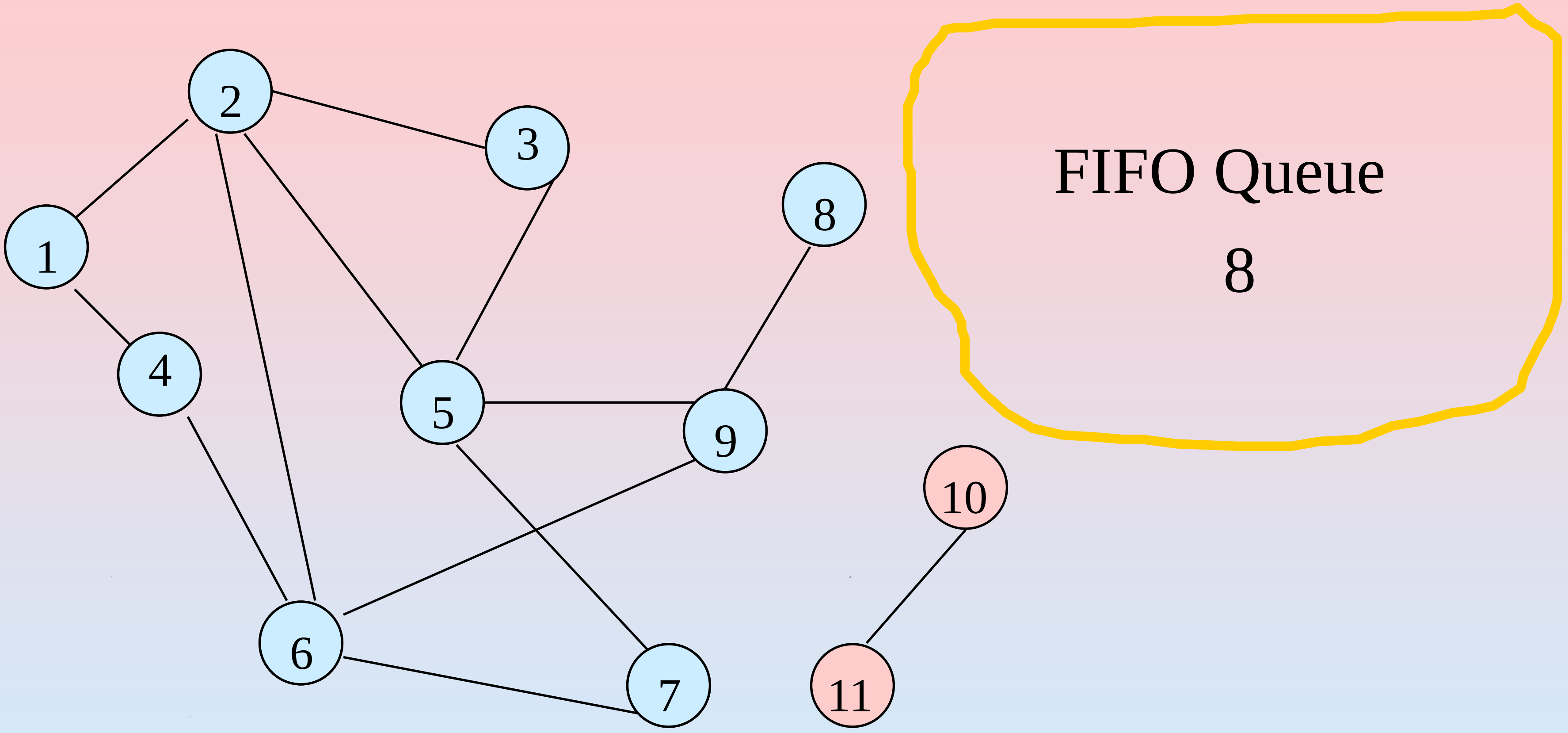


FIFO Queue

8

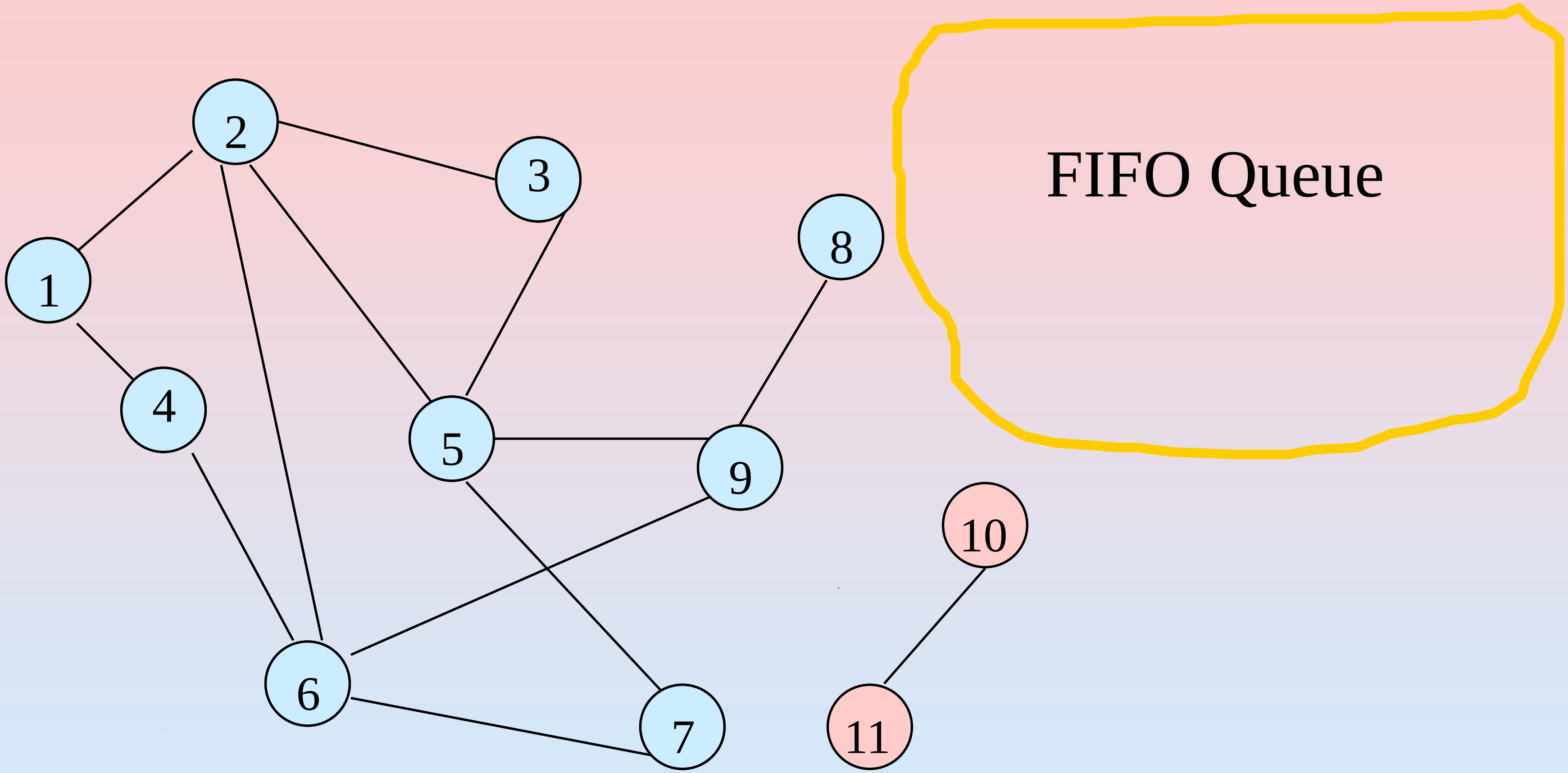
Remove 7 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



Remove 8 from Queue; visit adjacent unvisited vertices; put in Queue.

Breadth-First Search of Graph



Queue is empty. Search terminates.

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**