CS202 – Data Structures

LECTURE-19

Graphs – II

Graphs Traversals

Dr. Maryam Abdul Ghafoor
Assistant Professor
Department of Computer Science, SBASSE

Agenda

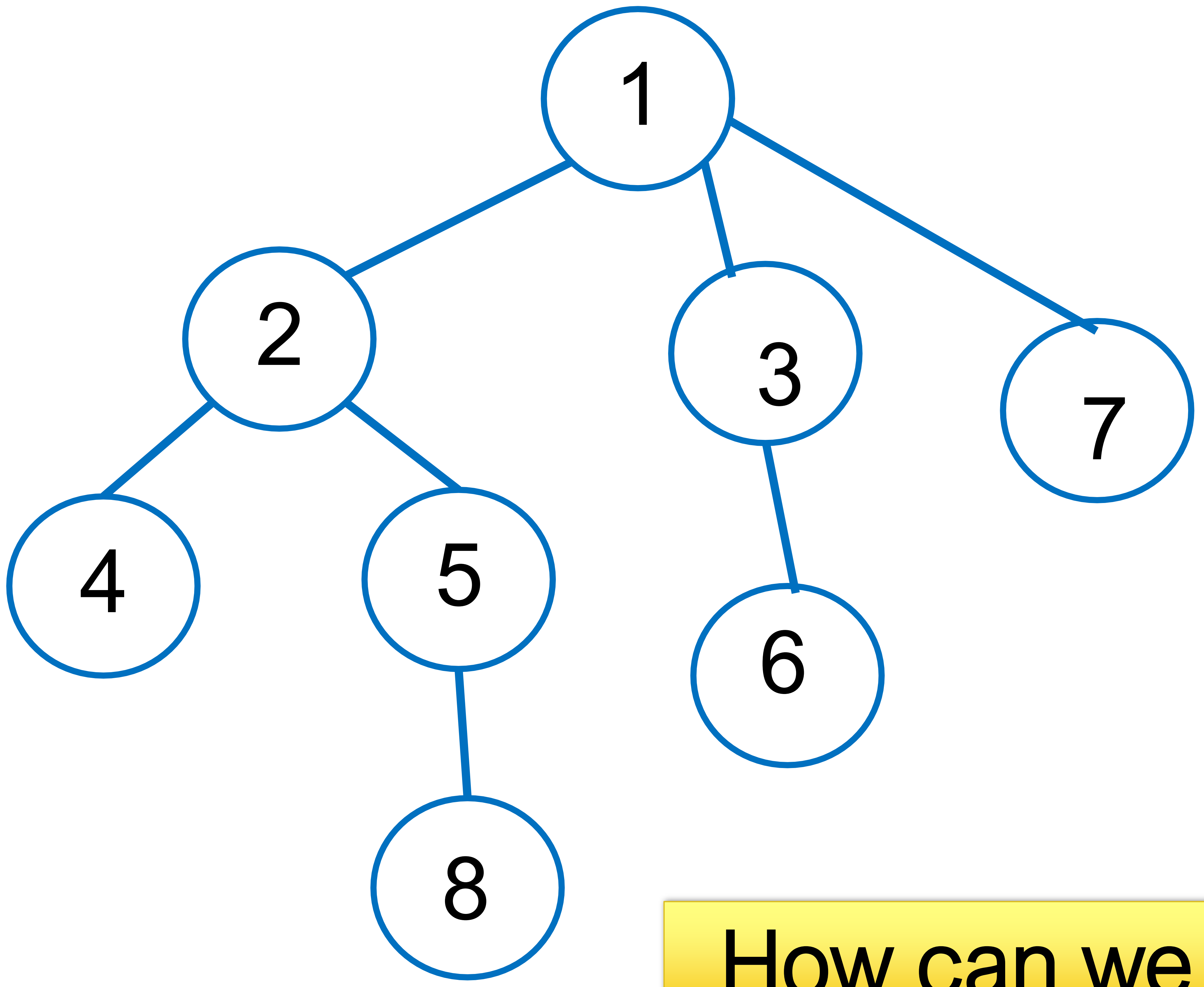
- Graph Traversals

Graphs in C++

```
Class Graph{
    unordered_map<int, vector<int>> adj;
     …..
}
```

Graph Traversal

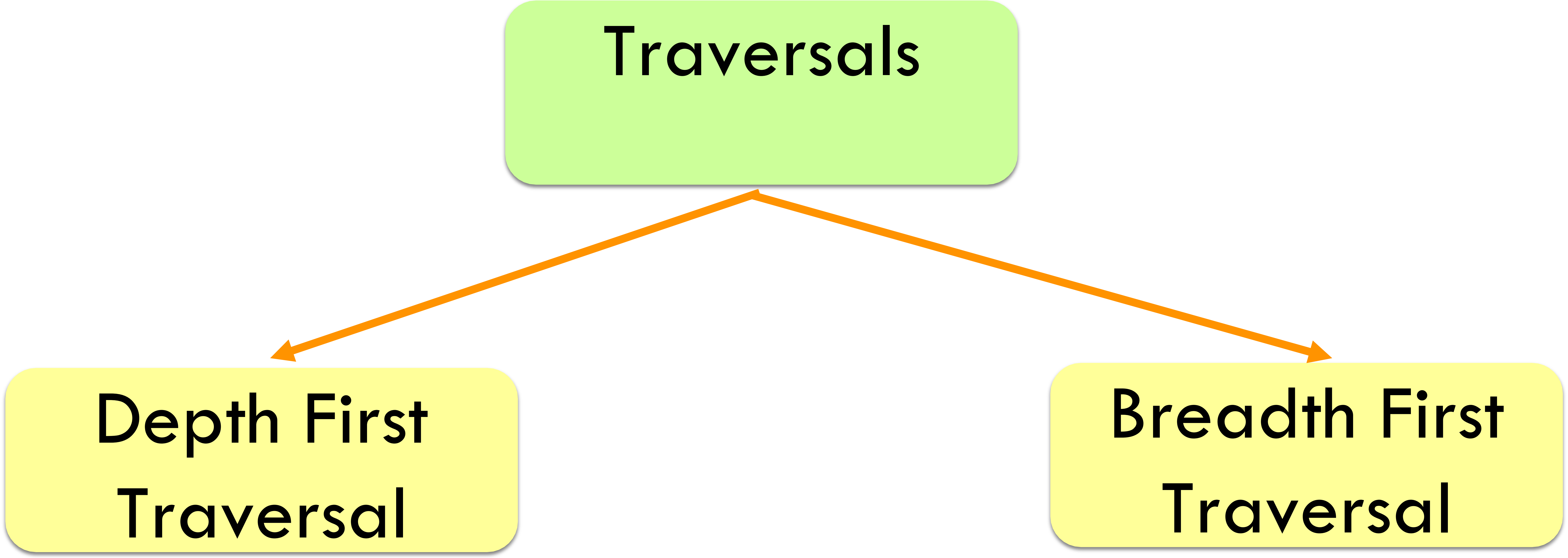
- Traversal: visiting each vertex once



How can we visit each node exactly once?

Graph Traversals

- Traversal: visiting each vertex once

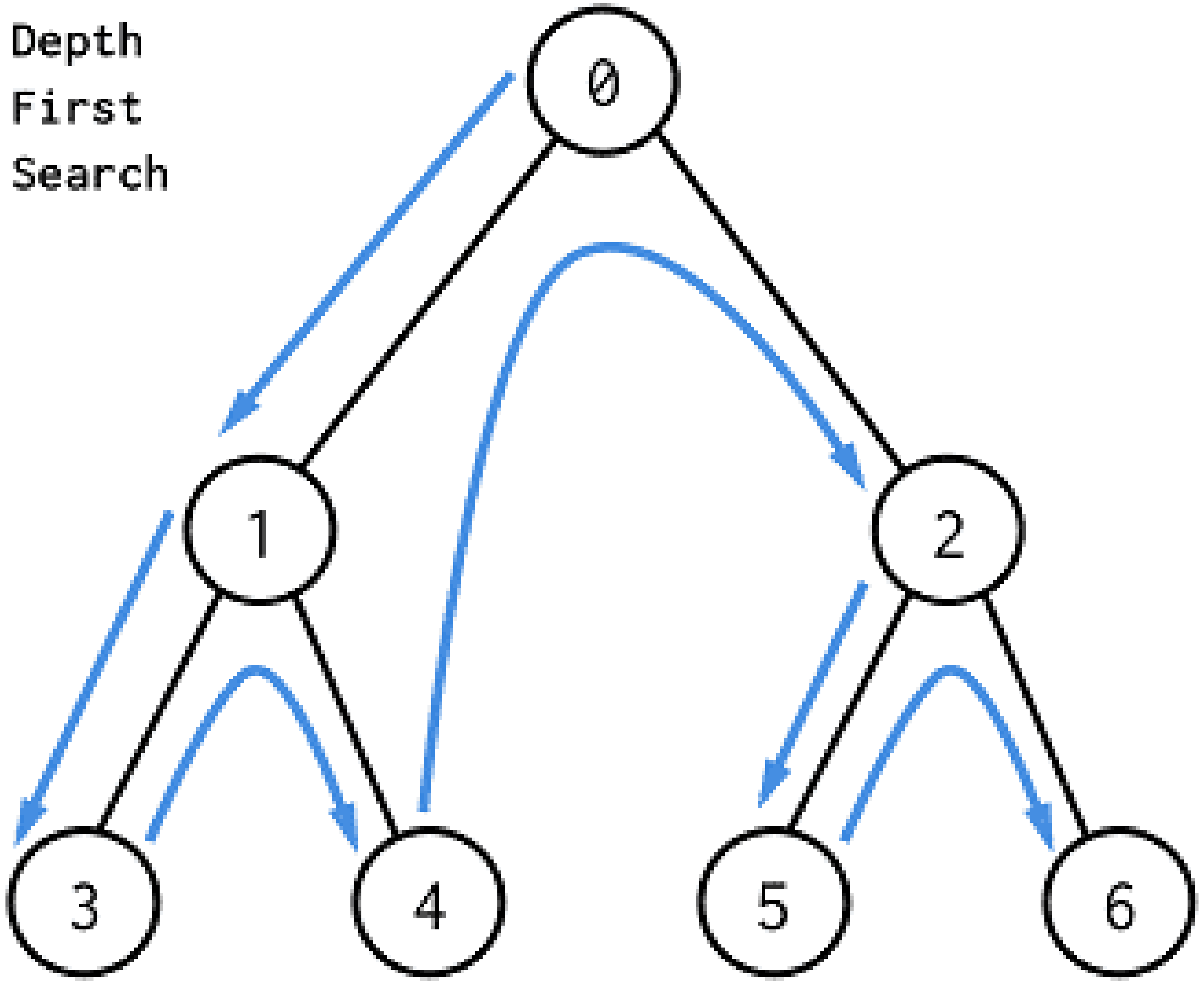


Visit nodes as deeply as possible, early on, and back track as needed

Visit nodes level by level (i.e., first nodes that are 1-hop away, then 2-hops away and so on)

Depth First Traversal

- A way to traverse a graph
- Starts at the root node and explores as far as possible along each branch and then backtracking (all the way down)



Depth First Traversal (DFS)

- Uses a Stack/Recursion, LIFO

- Works by prioritizing the items that are deeper down that branch (those go on top of the stack, which we pop off first) (don't look at next layer until that whole branch is done)

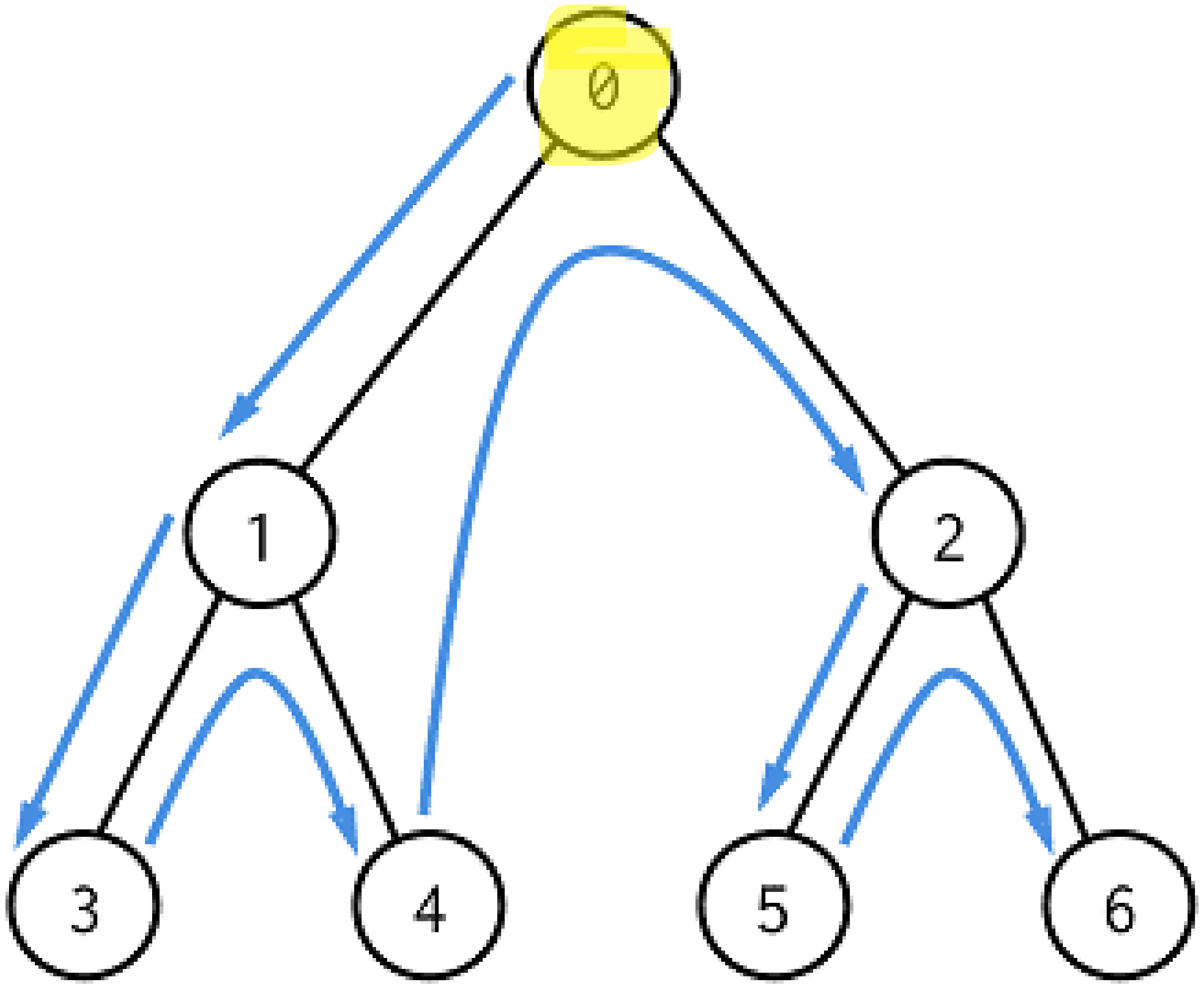
DFS

Stack:

0

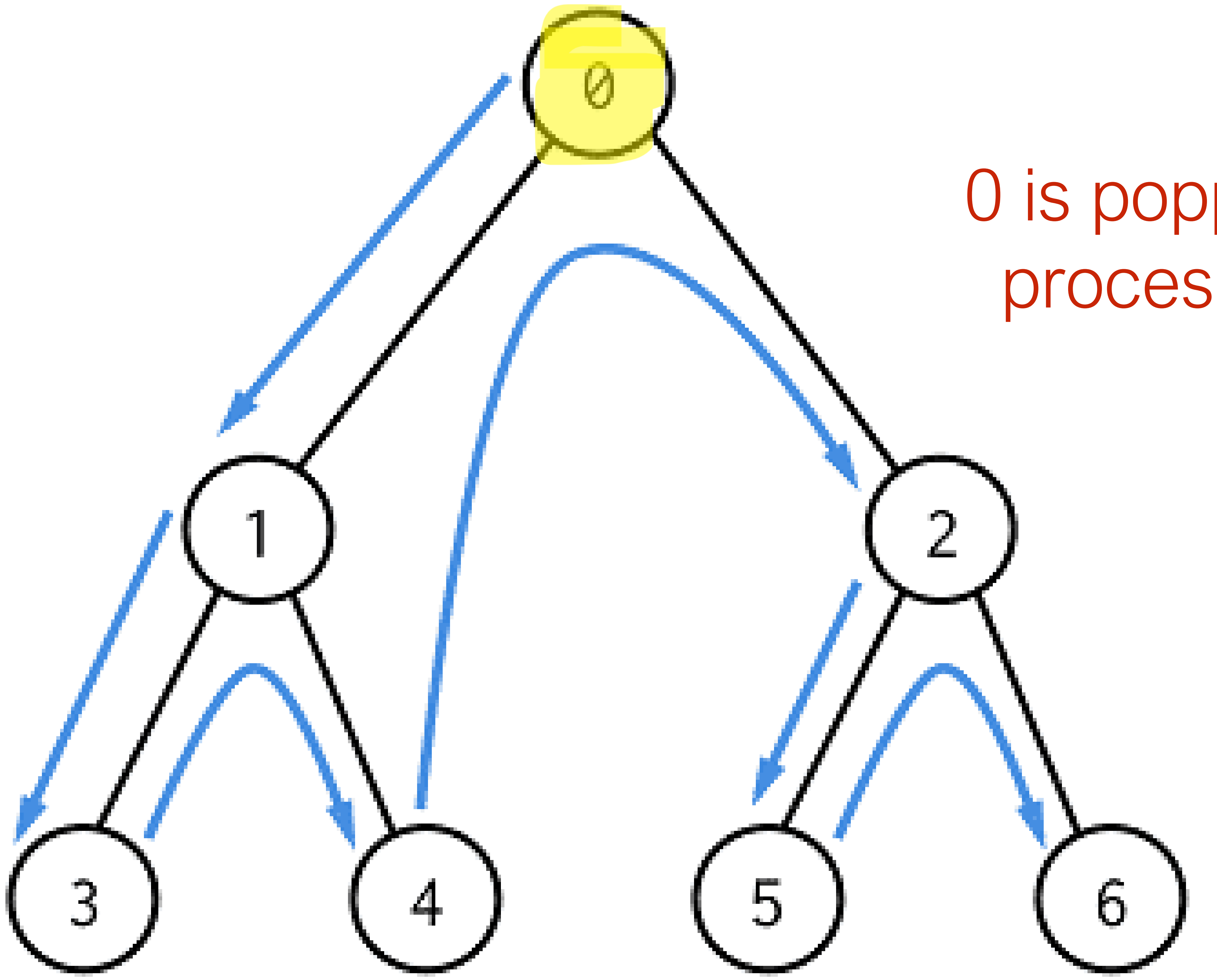
Current Node:

Visited:



DFS

Stack:



0 is popped from the stack to process its adjacent nodes

Current Node:

0

Visited:

DFS

Stack:

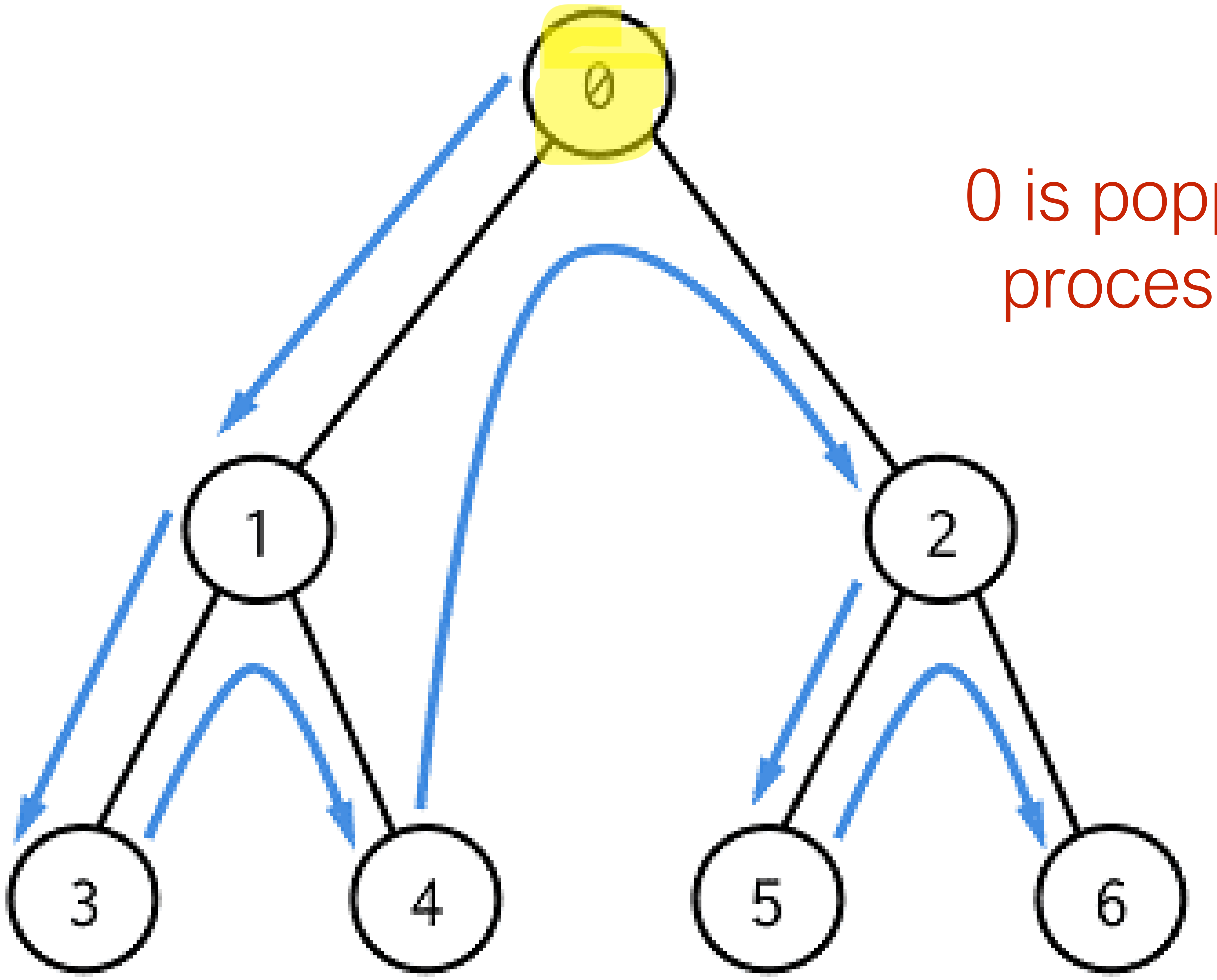
1 2

Current Node:

0

Visited:

0



0 is popped from the stack to process its adjacent nodes

Done with 0, add it to the visited set

DFS

Stack:

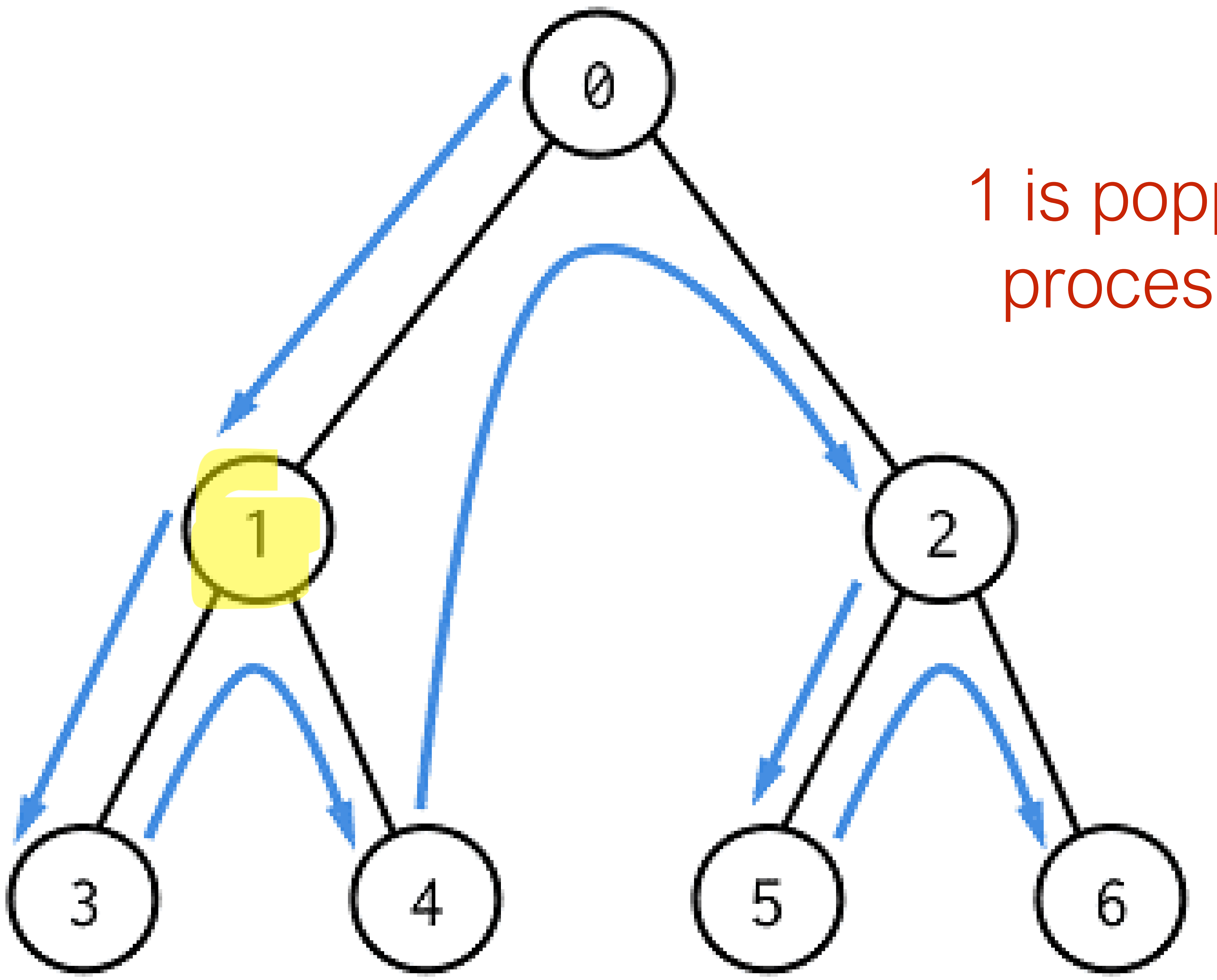
2

Current Node:

1

Visited:

0



1 is popped from the stack to process its adjacent nodes

DFS

Stack:

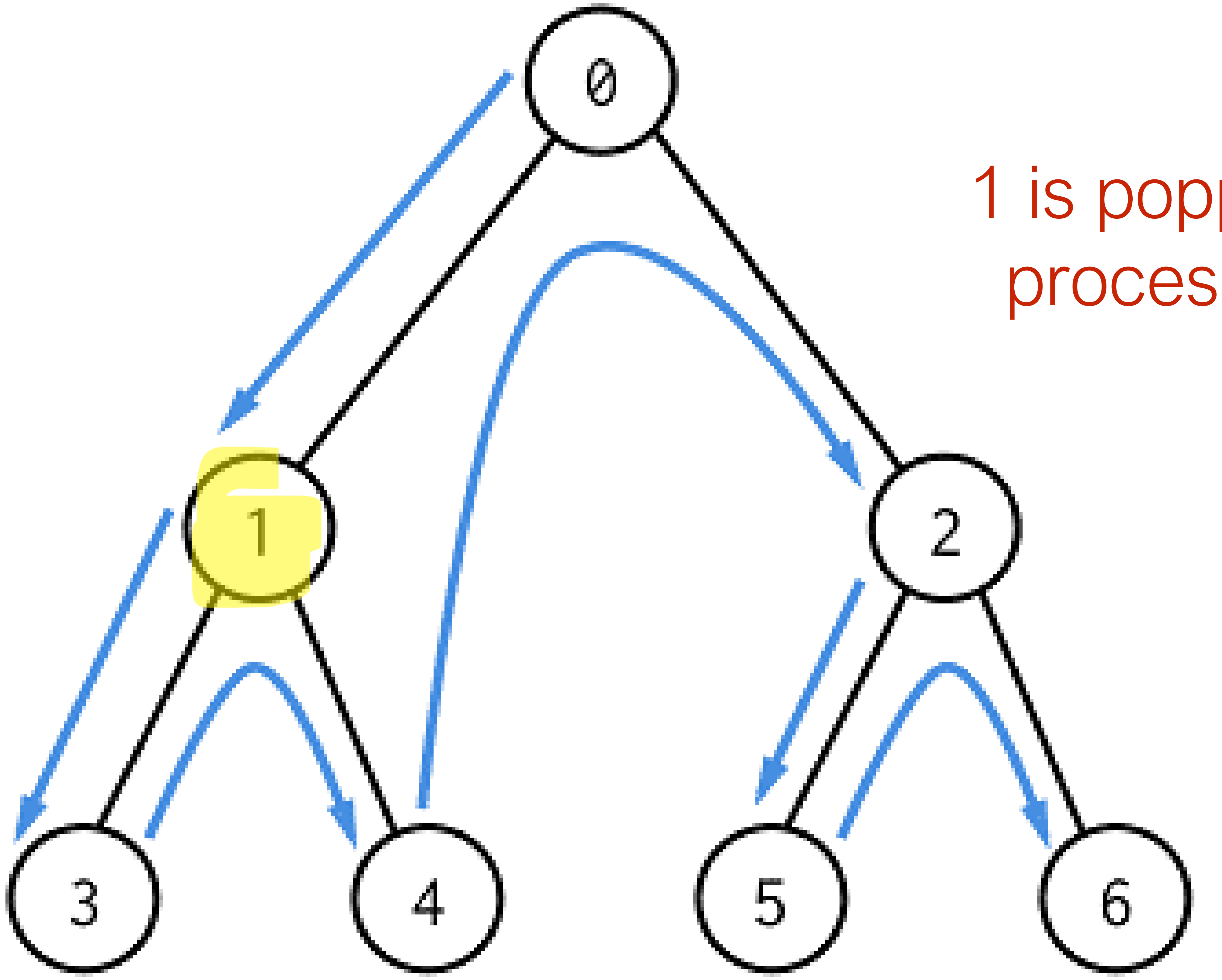
3 4 2

Current Node:

1

Visited:

0 1



1 is popped from the stack to process its adjacent nodes

Done with 1, add it to the visited set

DFS

Stack:

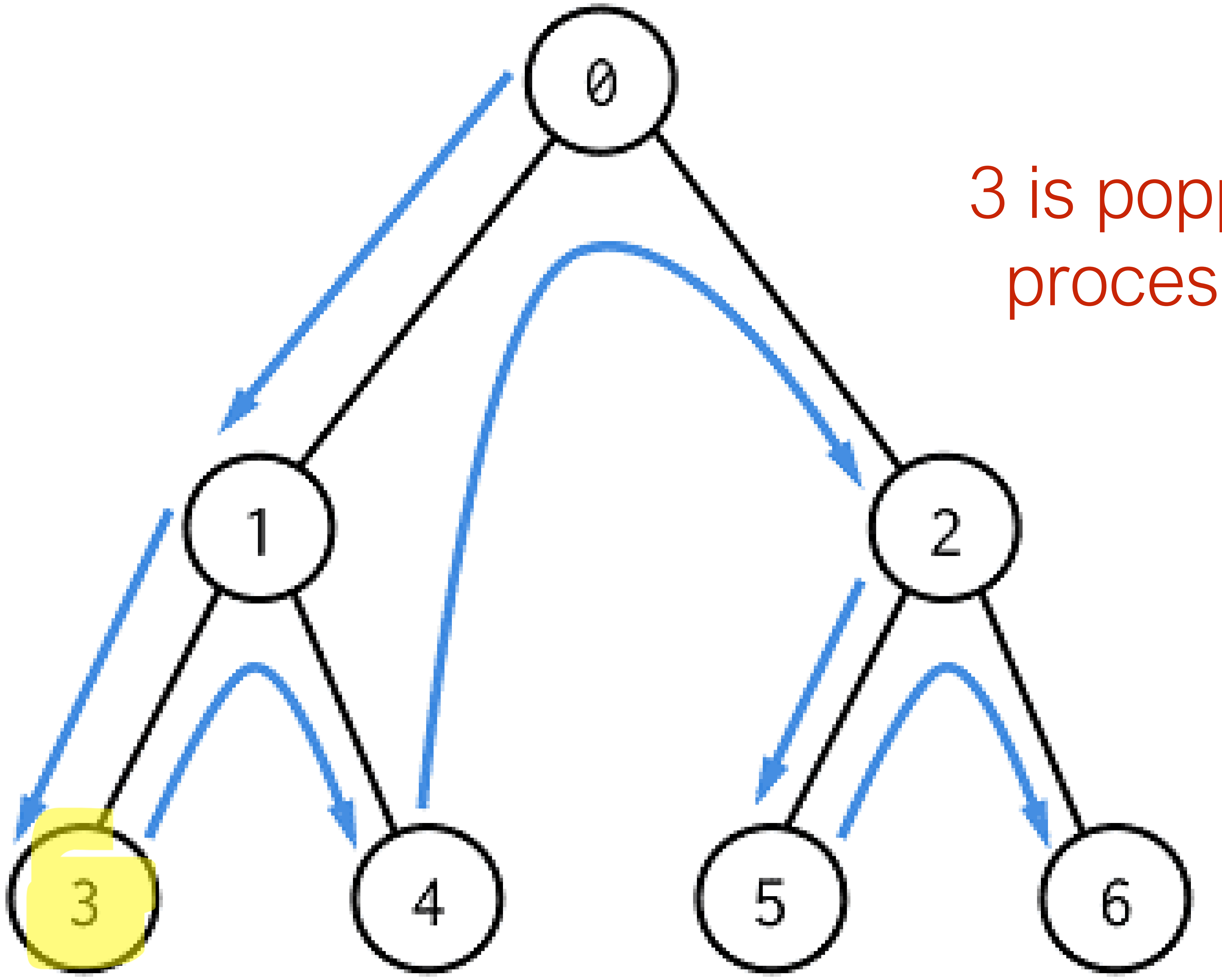
4 2

Current Node:

3

Visited:

0 1



3 is popped from the stack to process its adjacent nodes

DFS

Stack:

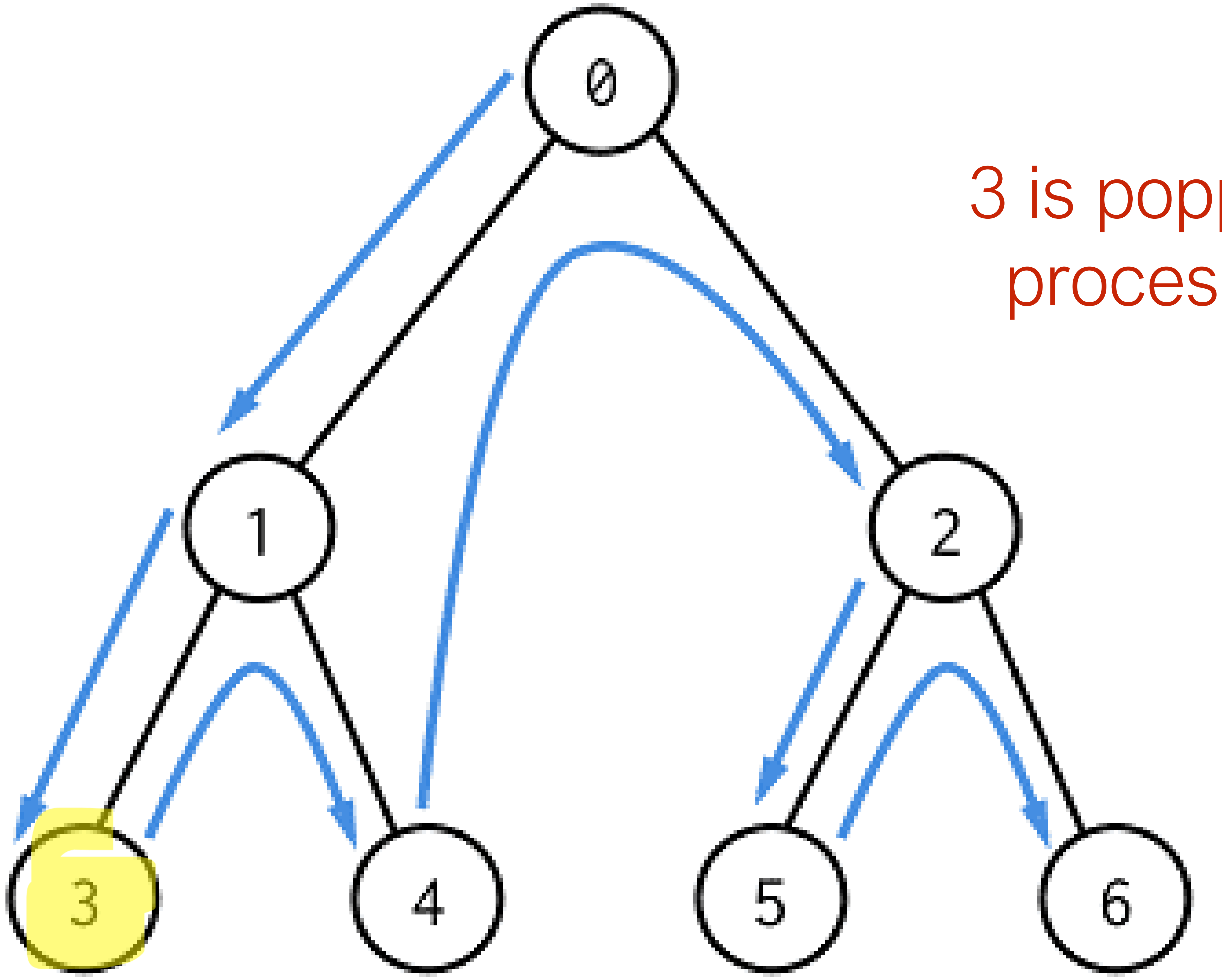
4 2

Current Node:

3

Visited:

0 1 3



3 is popped from the stack to process its adjacent nodes

Done with 3, add it to the visited set

DFS.... Last iteration

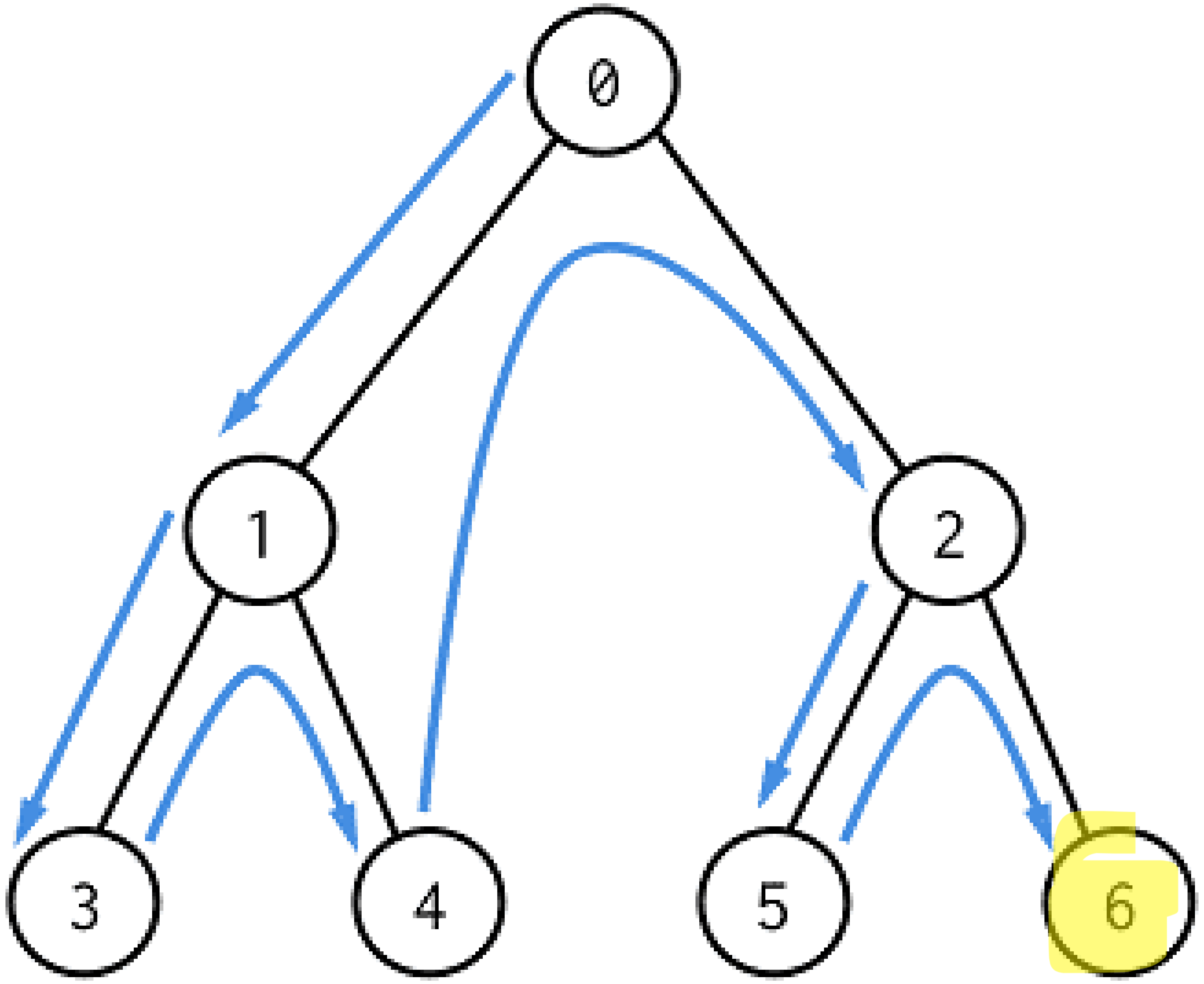
Stack:

Current Node:

6

Visited:

0 1 3 4 2 5 6



Done with 6, add it to the visited set

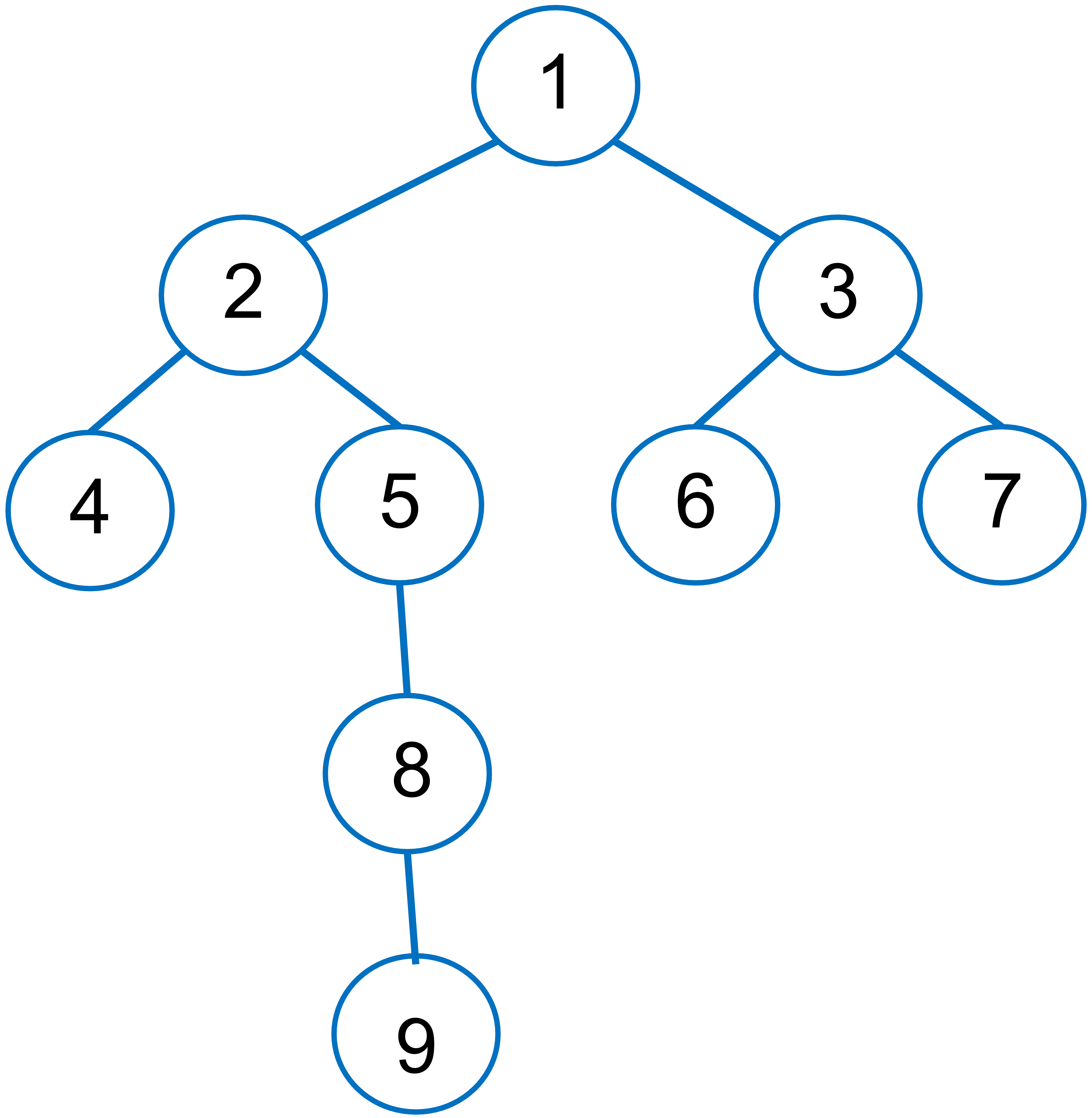
DFS implementation

- Each vertex has a boolean field "visited"

```
void dfs(vertex u){
   u.visited = true;
   for(each vertex v s.t. (u,v) is in E){
      if(!v.visited)
         dfs(v);
   }
}
```

Suppose we run DFS on a tree…

- …and always visit the left child before the right child

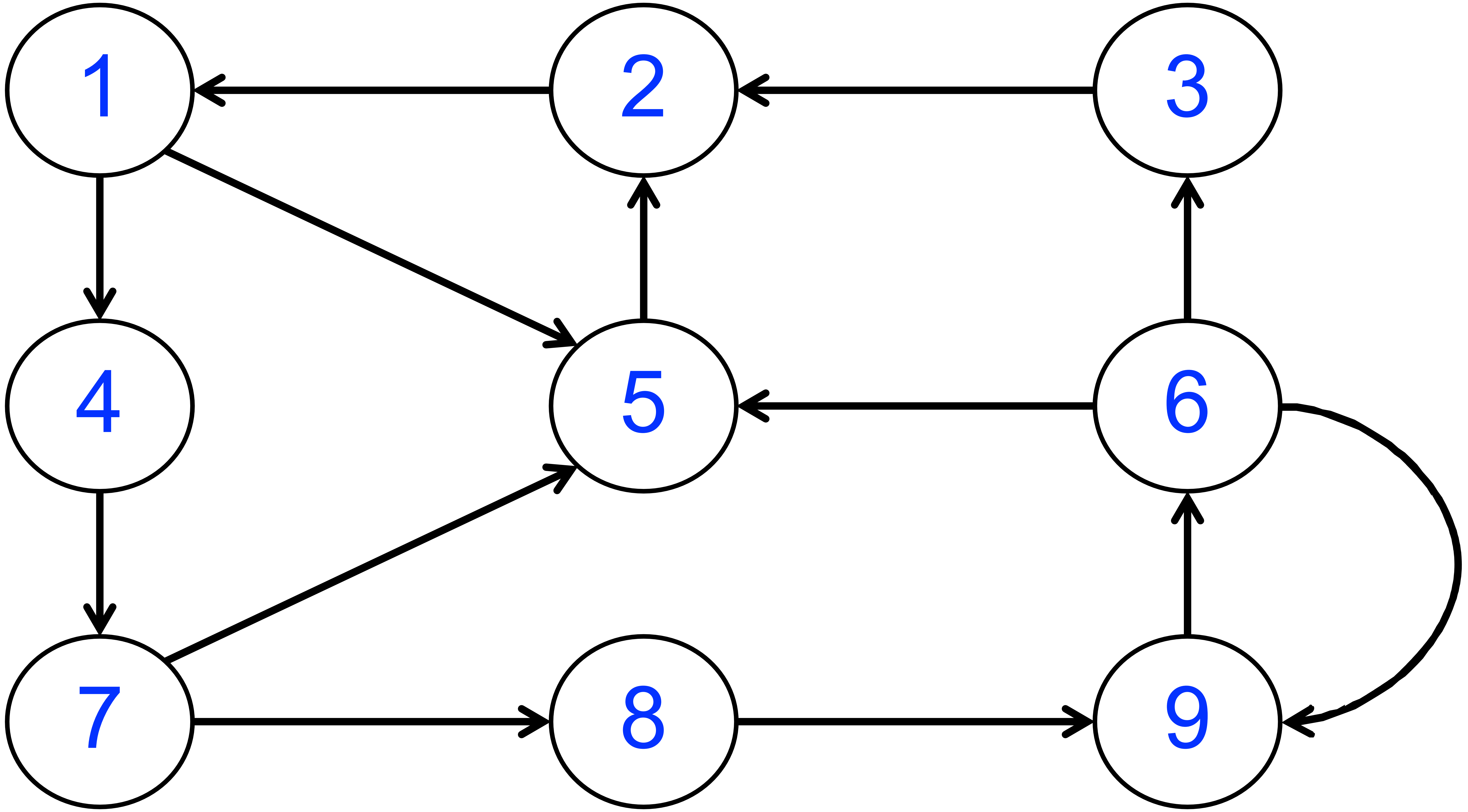


Which tree traversal does it correspond to?

Concept Check!

- What is the output of dfs(1)? (Nodes are pushed in descending order, if there are multiple paths from that node)

dfs(1):



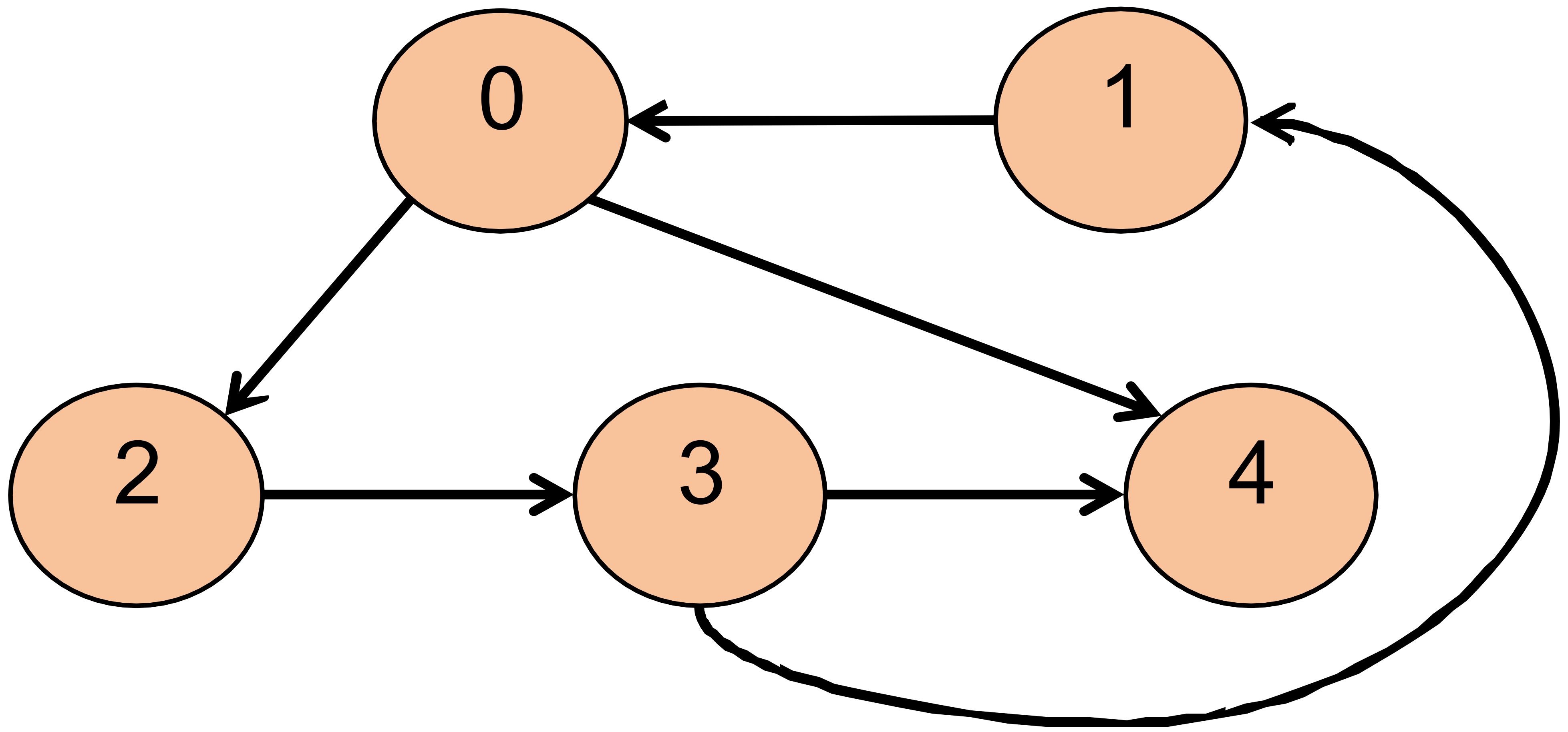
Breadth First Search (BFS)

- Works by prioritizing the items that are "siblings", or in the same graph layer (those go in the queue first, which we take off first
- Hard to code 'recursively'
- Uses a queue (FIFO), as we do in level-order traversal on a tree
  - Queue holds "nodes to be visited"

BFS Pseudocode

```
void bfs(vertex u){
   u.visited = true;
   q = new Queue();
   q.enqueue(u);

   while(q is not empty){
      v = q.dequeue();
      for(each e s.t.(v,e) ∈ E){
         if(!e.visited){
            e.visited = true;
            q.enqueue(e);
         }
   }}}
```

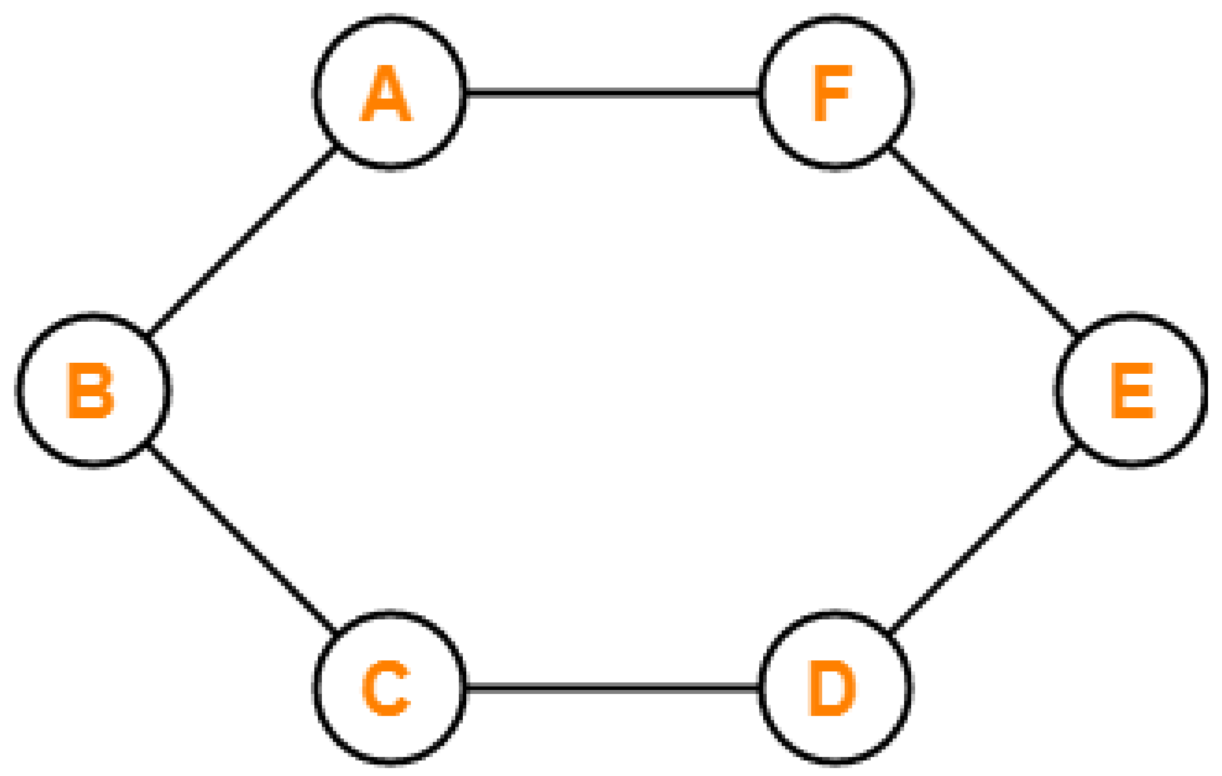


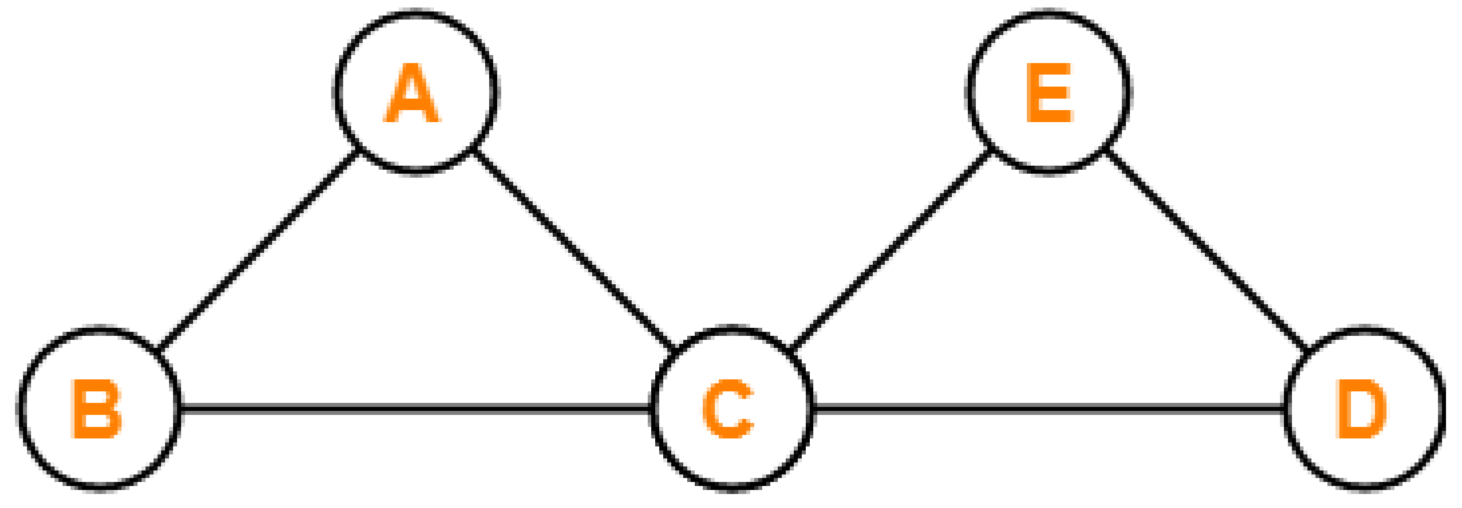
Traversals – Time Complexity

- AL: $O(|V| + |E|)$
- AM: $O(|V|^2)$

Many Interesting Graph Problems

- There are lots of interesting questions we can ask about a graph
  - Are there cycles in the graph?
  - What is the shortest route from A to E?
  - What is the longest path without cycles?
  - Is there a tour you can take that only uses each vertex exactly once? (Hamilton tour)
  - Is there a tour that uses each edge exactly once? (Euler Tour)





Let’s try to solve some problems

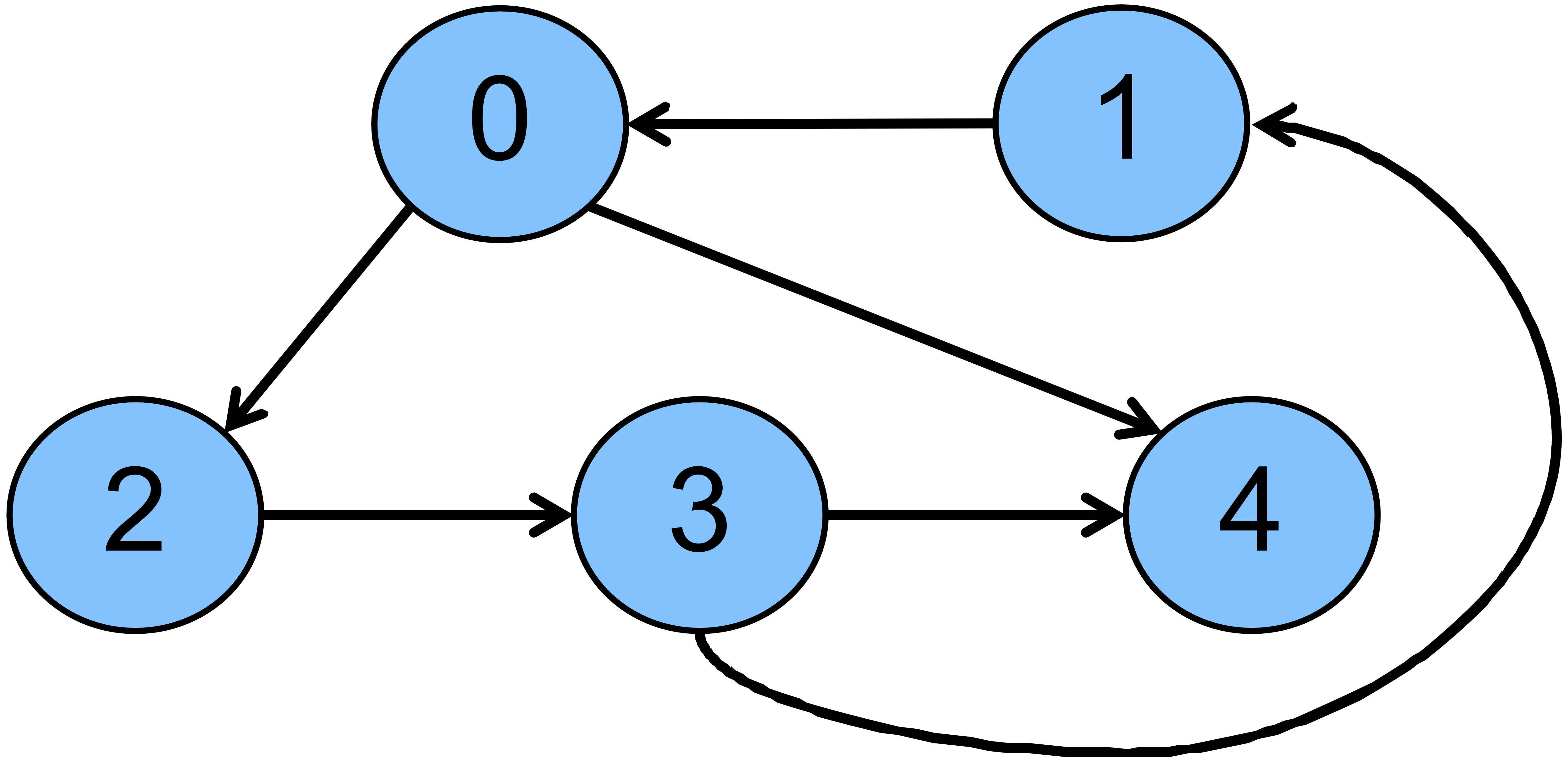
- Problem (Path detection): “Is there a path from vertex s to vertex t?”

- Solution:
  - Run DFS start from vertex s
  - If vertex k is visited → there is a path from j to k
  - Time complexity?

Problem: Cycle detection

"Does the graph G contain a cycle?"

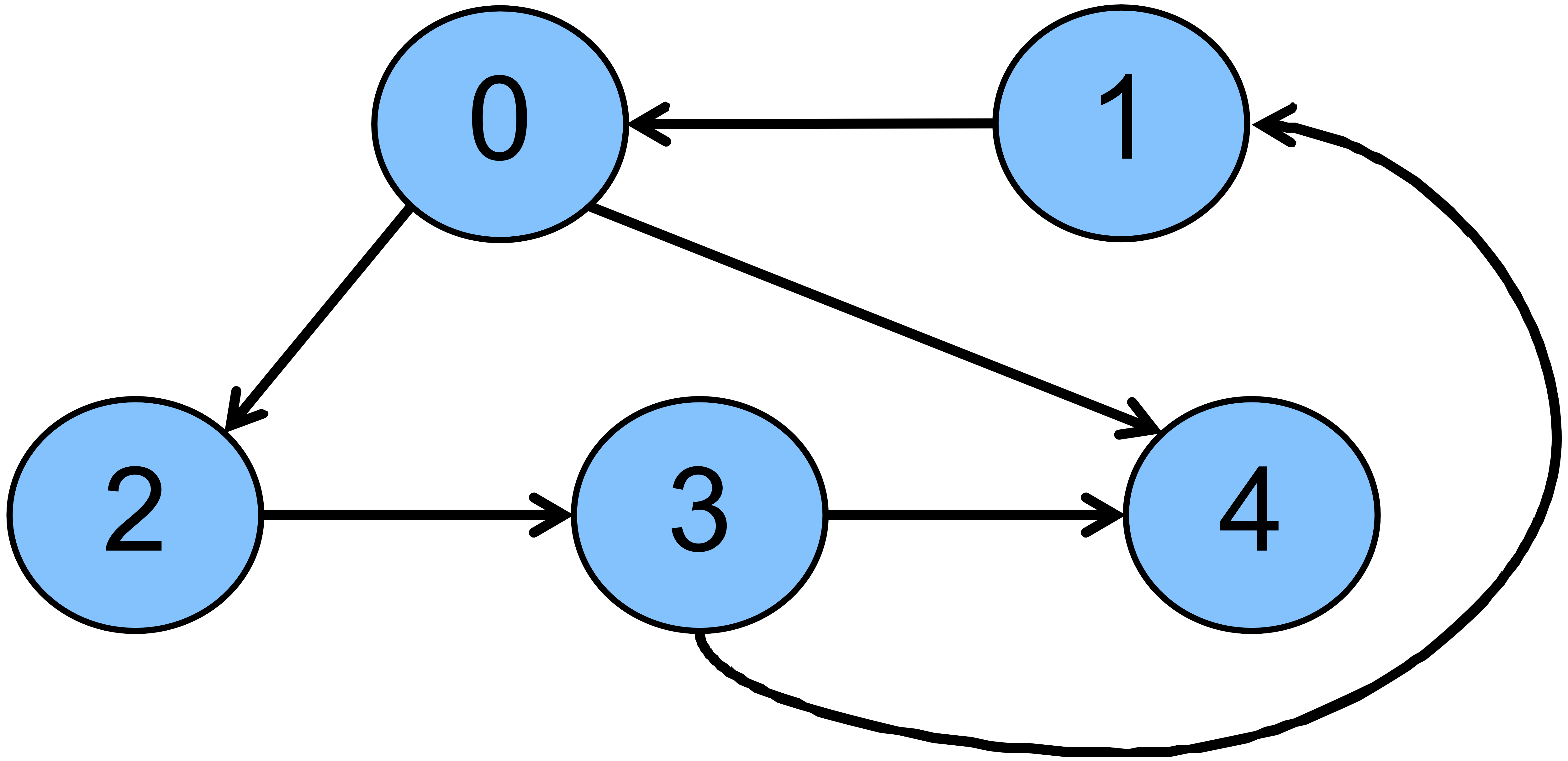
Cycle Detection




```
Cycle:
{0,2,3,1,0}
```

Cycle Detection

Cycle:
{0,2,3,1,0}

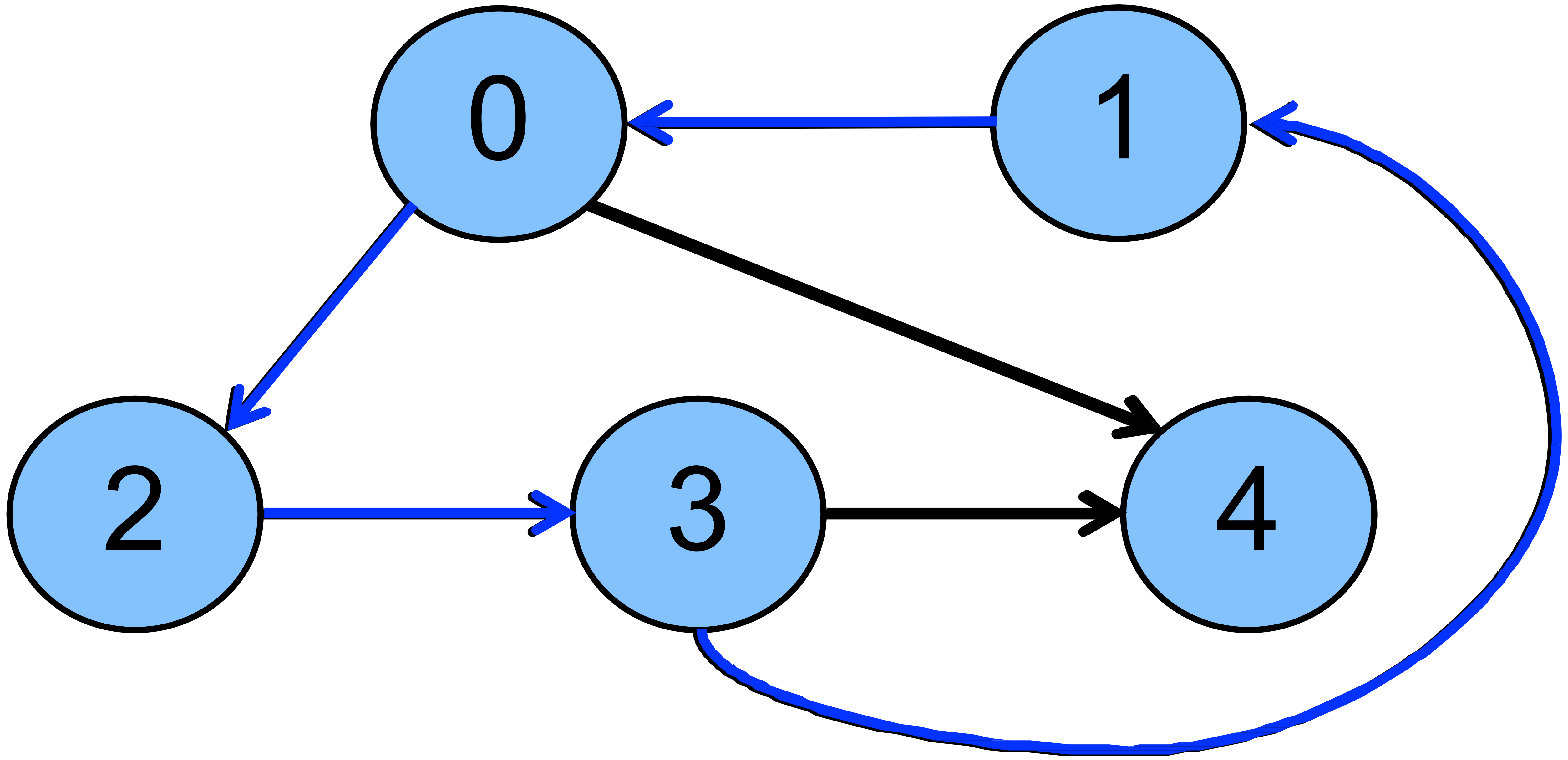


Approach 1: Run DFS, if you encounter a vertex that is already visited then return "there is a cycle"

Cycle Detection

- dfs(0) = {0,2,3,1,0}

```
Cycle:
{0,2,3,1,0}
```

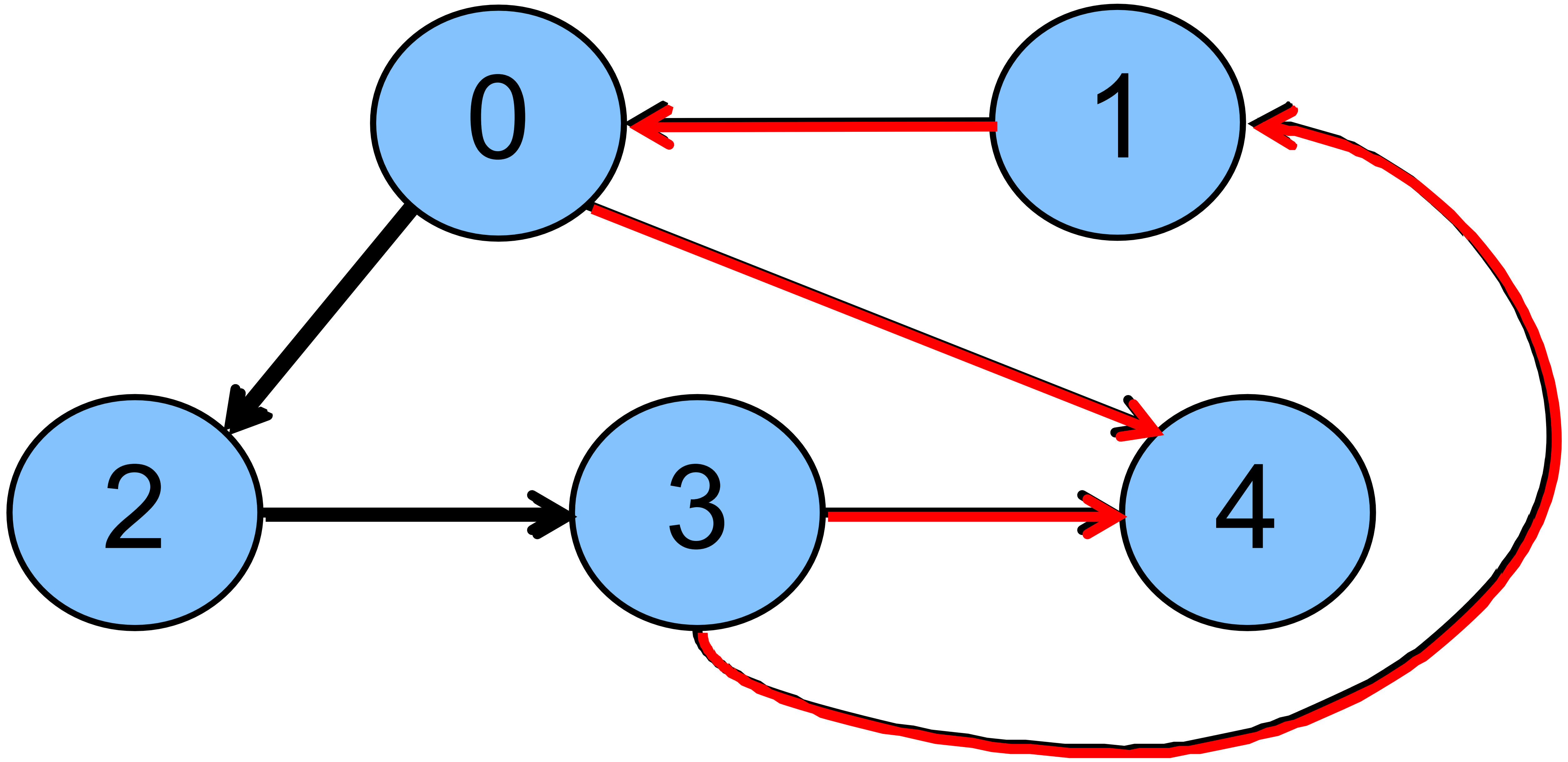


Approach 1: Run DFS, if you encounter a vertex that is already visited then return "there is a cycle"

Cycle Detection

- dfs(0) = {0,2,3,1,0}
- dfs(3) ={3,4,1,0,4*} NOT a cycle!

Cycle:
{0,2,3,1,0}

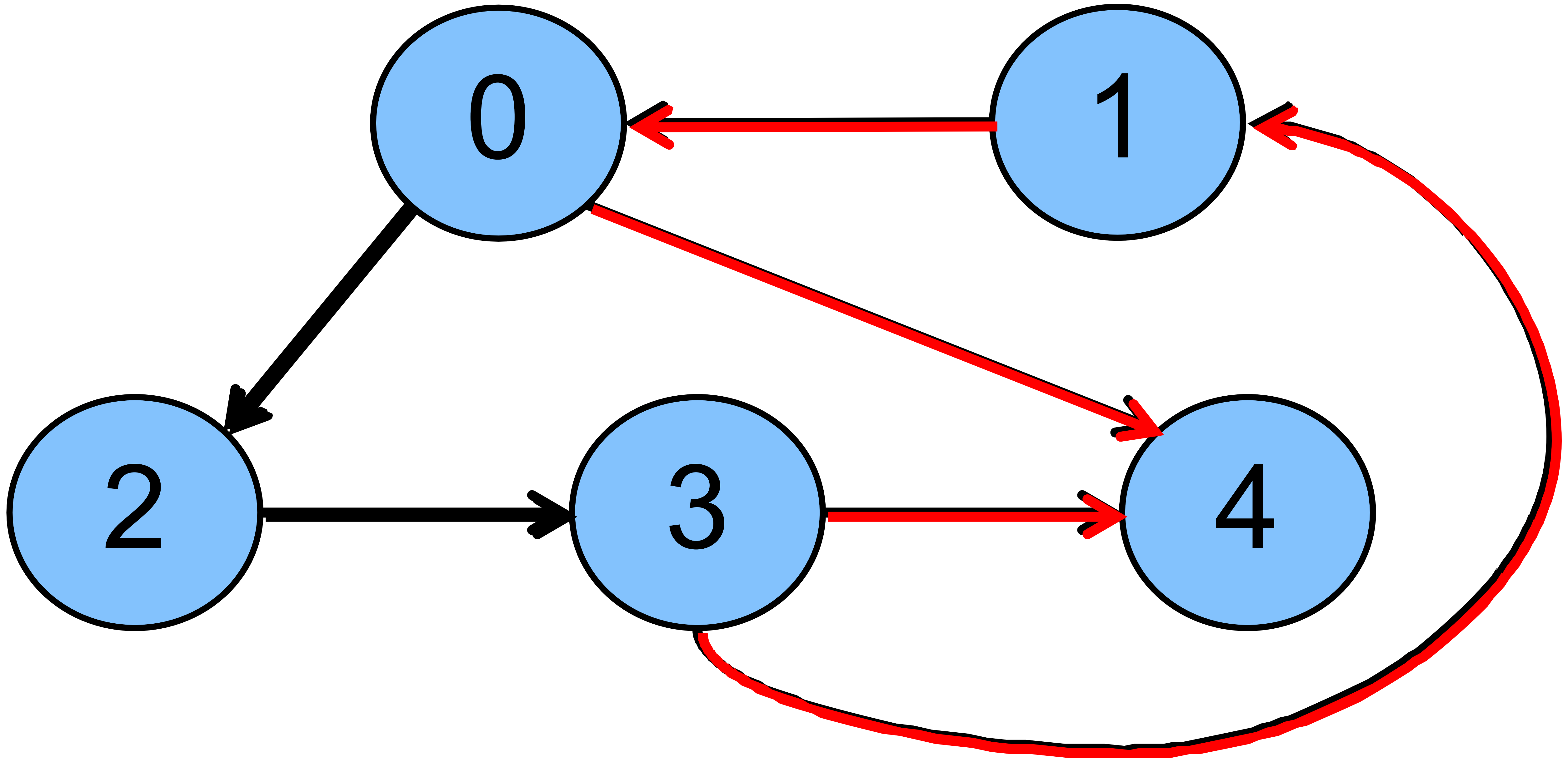


Approach 1: Run DFS, if you encounter a vertex that is already visited then return "there is a cycle"

Cycle Detection

- dfs(0) = {0,2,3,1,0}
- dfs(3) ={3,4,1,0,4*} NOT a cycle!

Cycle:
{0,2,3,1,0}

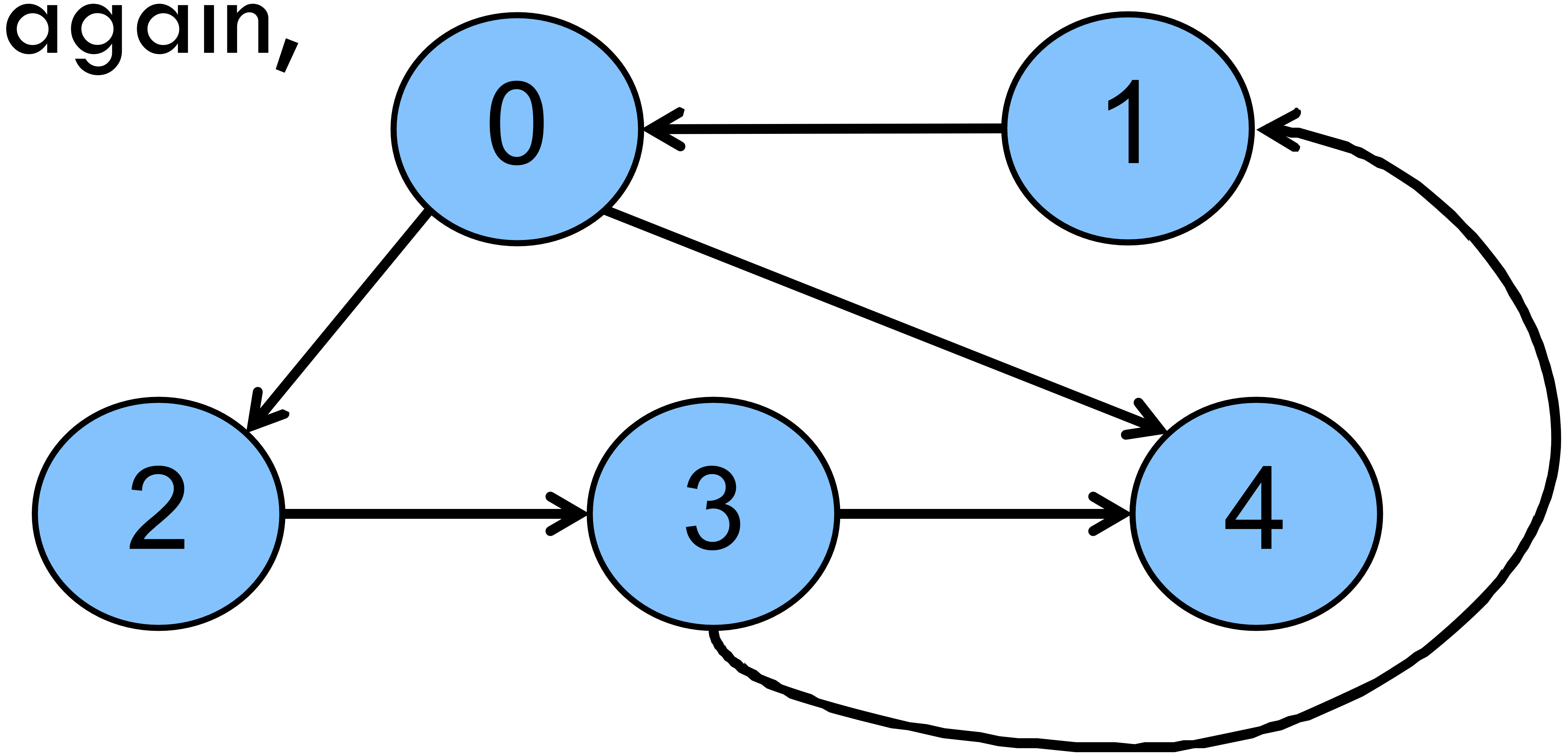


Approach 1: Run DFS, if you encounter a vertex that is already visited then return "there is a cycle"

Cycle Detection – Observations

- Case-1: when we visited vertex 0 again, dfs(0) was still active!
- Case-2: when we visited 4 again, dfs(4) was already done

```
Cycle:
{0,2,3,1,0}
```



Approach 2: Keep track of when a vertex is "inprogress"
Use a 3-state field to mark progress: (unvisited, inprogress, done)

Cycle Detection – Observations

- **Approach 2:** Keep track of when a vertex is "inprogress"
- Use a 3-state field to mark progress: (unvisited, inprogress, done)

  1. Initially, all nodes are unvisited
  2. When a node is first visited, we mark it as "inprogress"
  3. Once all successor nodes are visited, we mark it as done
  4. There is a cyclic path reachable from vertex i iff some node's successor is found to be marked "inProgress" during dfs(i)

Time Complexity

AL: O (|V| + |E|)

AM:O ($|V|^2$)

Questions