Data Structures and Algorithms

Week 10 - More on SCCs, Topological Sort

Subodh Sharma and Rahul Garg

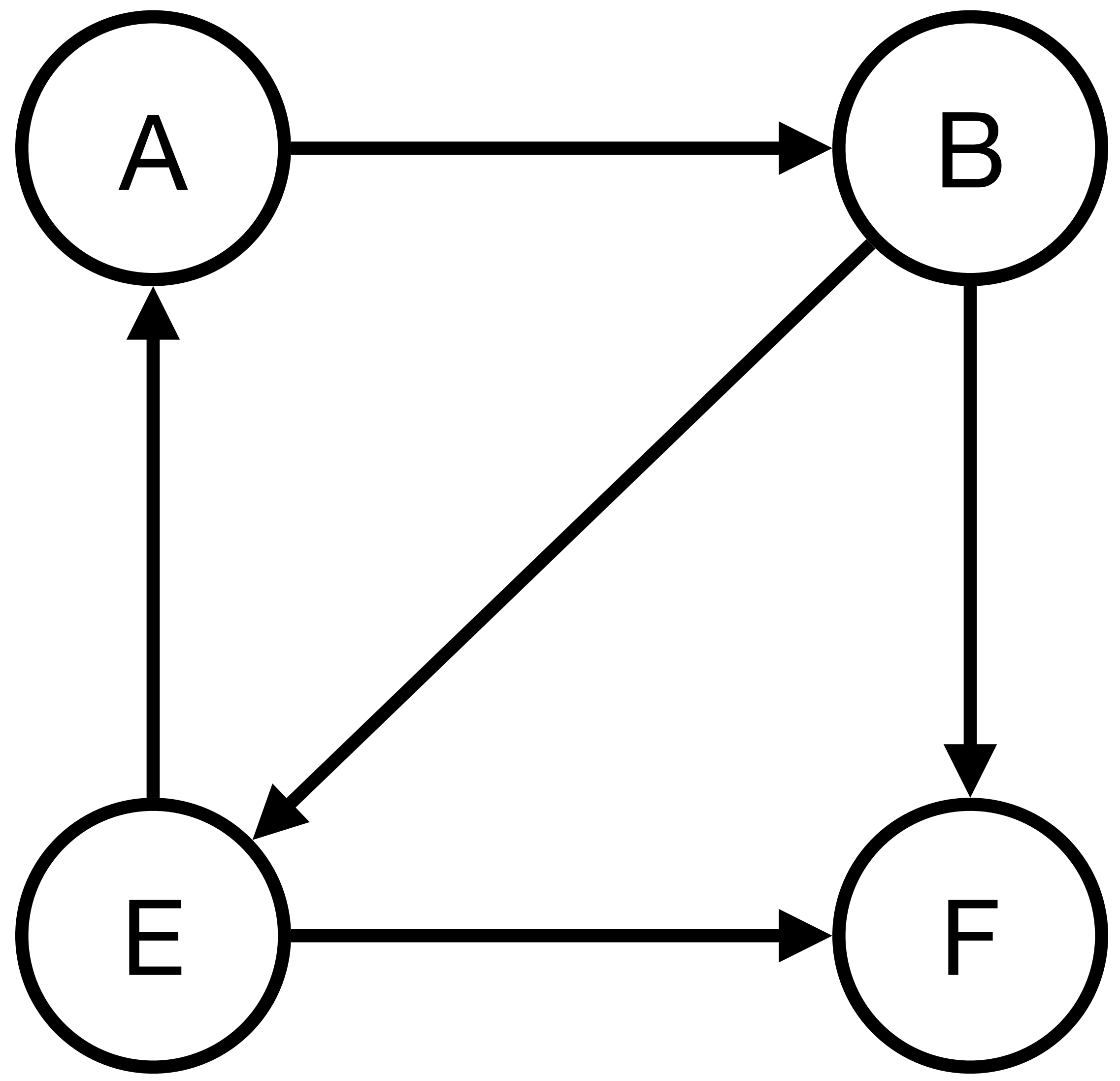
{svs,rahulgarg}@iitd.ac.in.

Strongly Connected Components

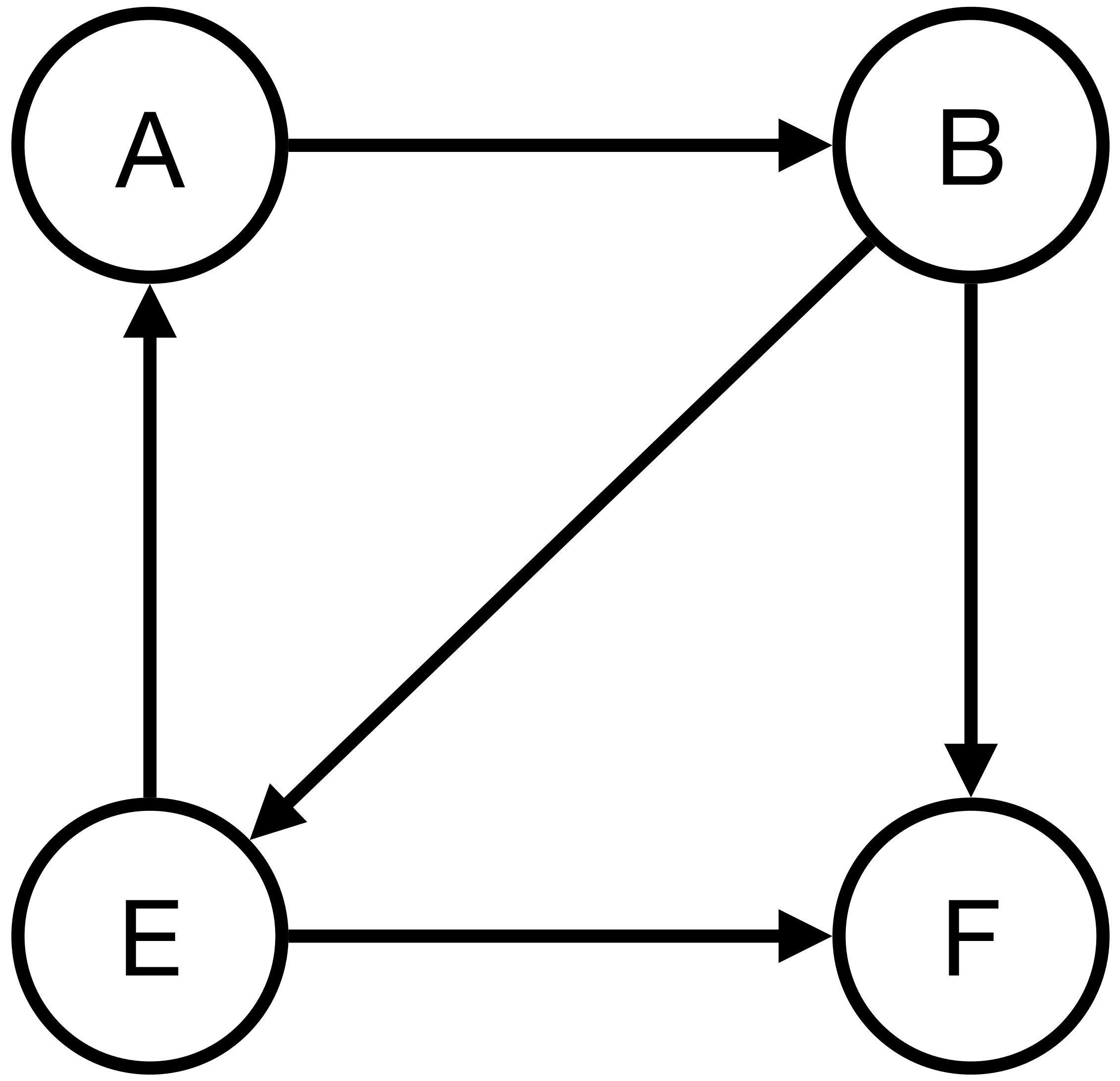
Strongly Connected Components

- A graph is said to be strongly connected - If every vertex is reachable from every other vertex
- The binary relation of being strongly connected is an **equivalence relation**
  - That is it is reflexive, symmetric and transitive
- Strongly connected component of a directed graph G is also **maximal**
- **Used in Abstractions!** SCCs in a graph can be **condensed** into single vertices leading to the formation of a **DAG**

Strongly Connected Components



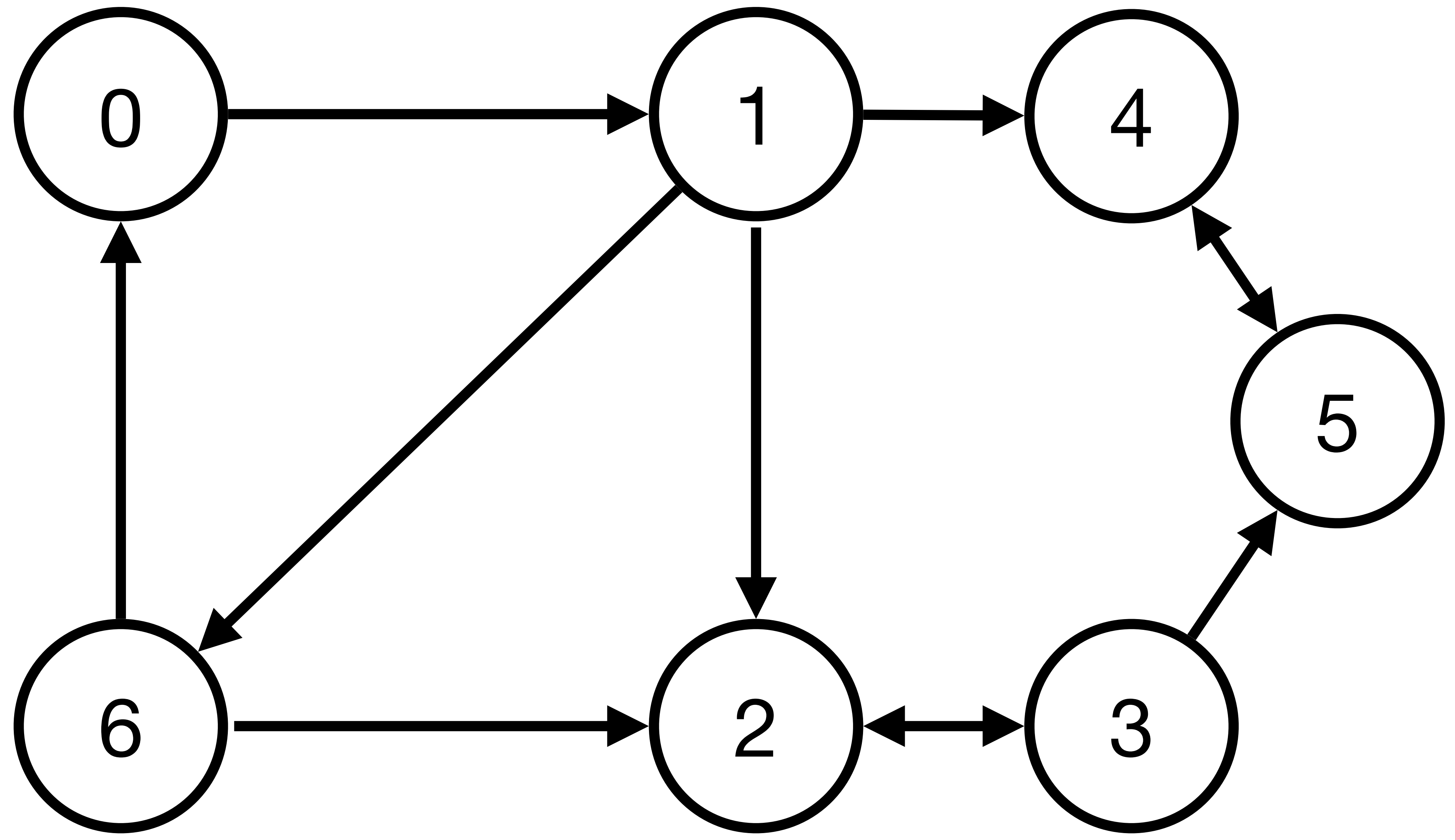
Strongly Connected Components



- SCC: ({A,B,E}. {() .. })
- Use of DFS to find SCCs — Robert Tarjan 1972 (also discovered Splay and Fibonacci Heaps)

Strongly Connected Components

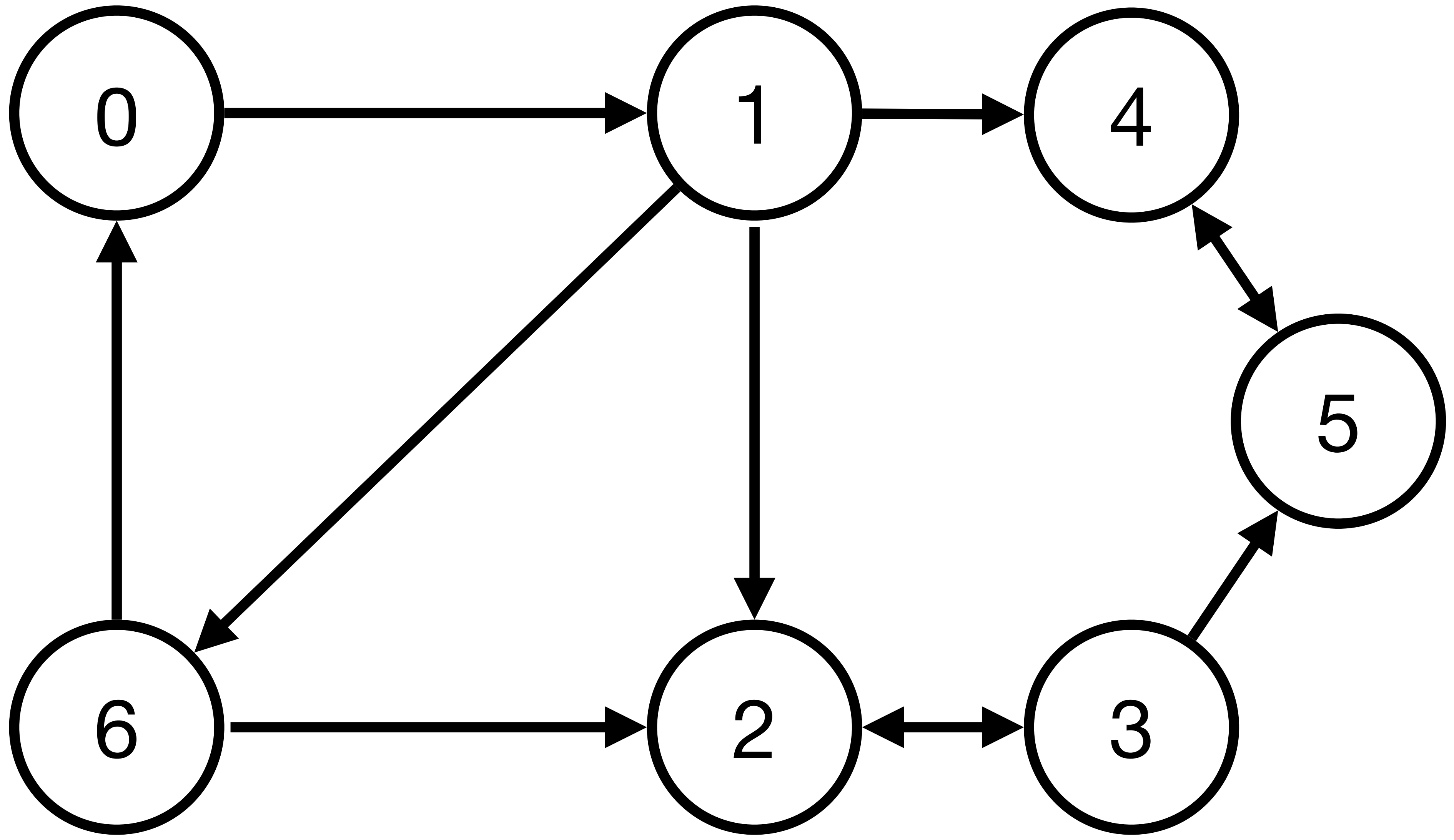
Tarjan’s Algorithm



Strongly Connected Components

Tarjan’s Algorithm

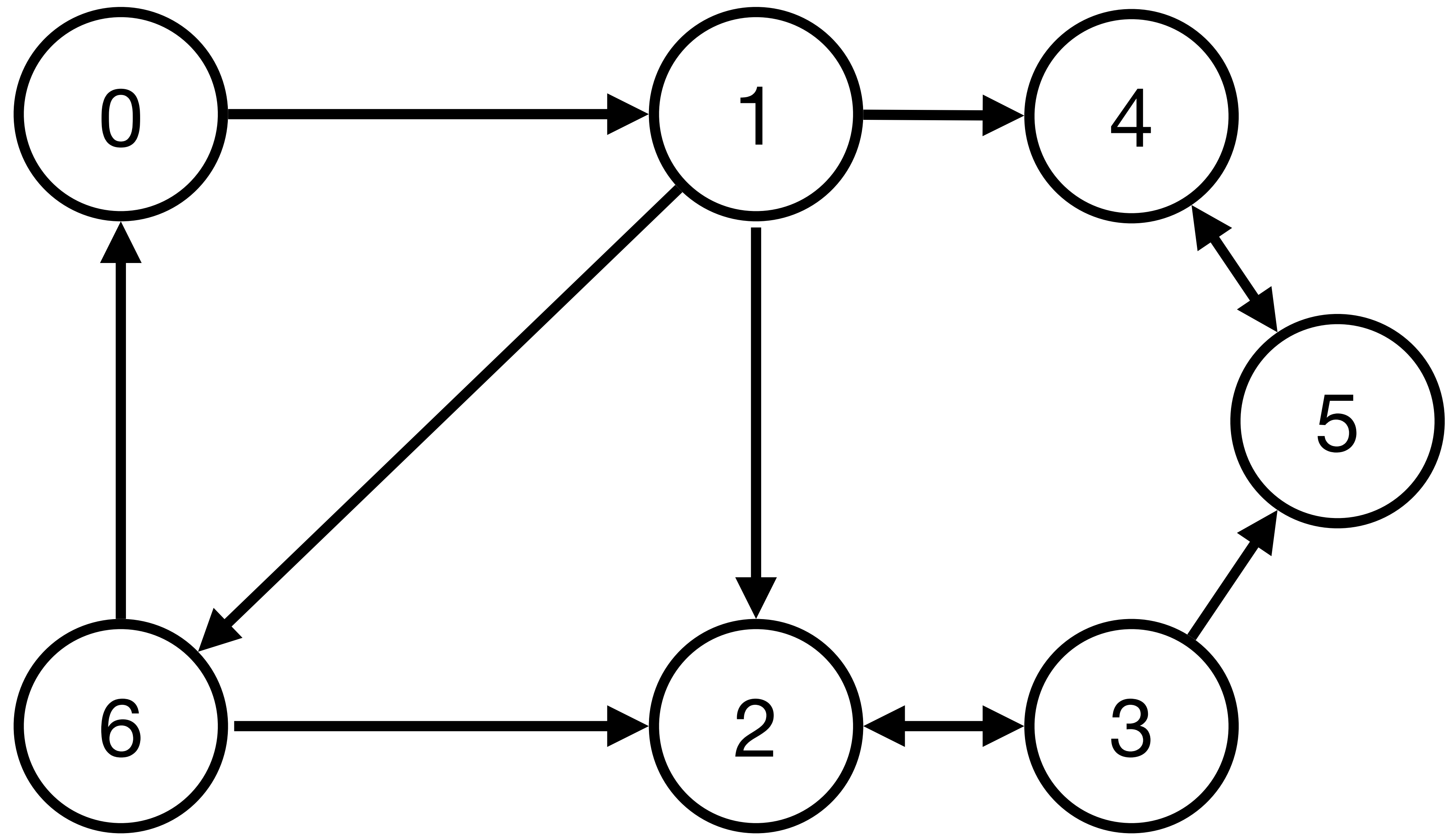
- **Key Observations:**



Strongly Connected Components

Tarjan’s Algorithm

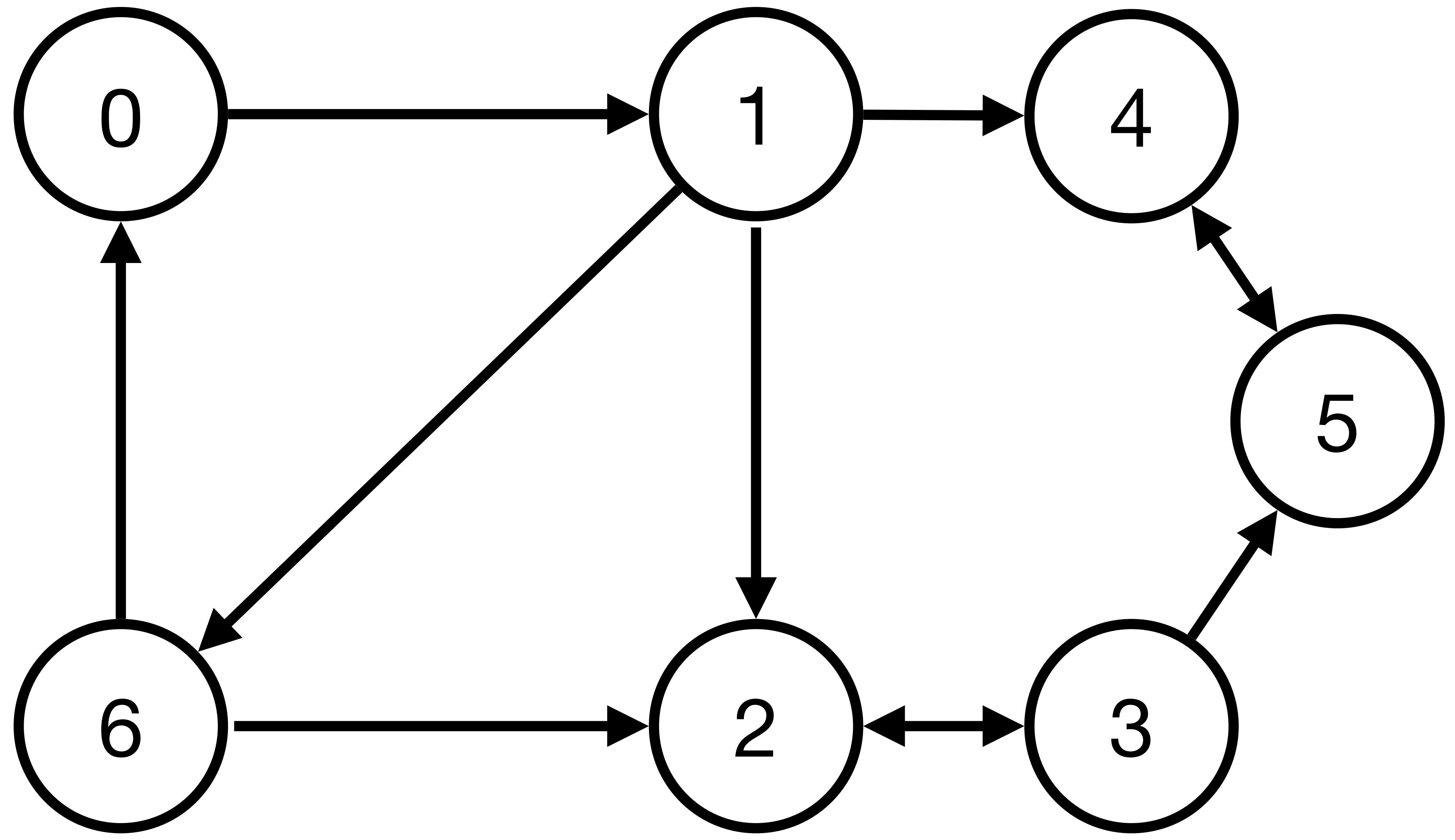
- **Key Observations:**
  - **Input:** Directed Graph G



Strongly Connected Components

Tarjan’s Algorithm

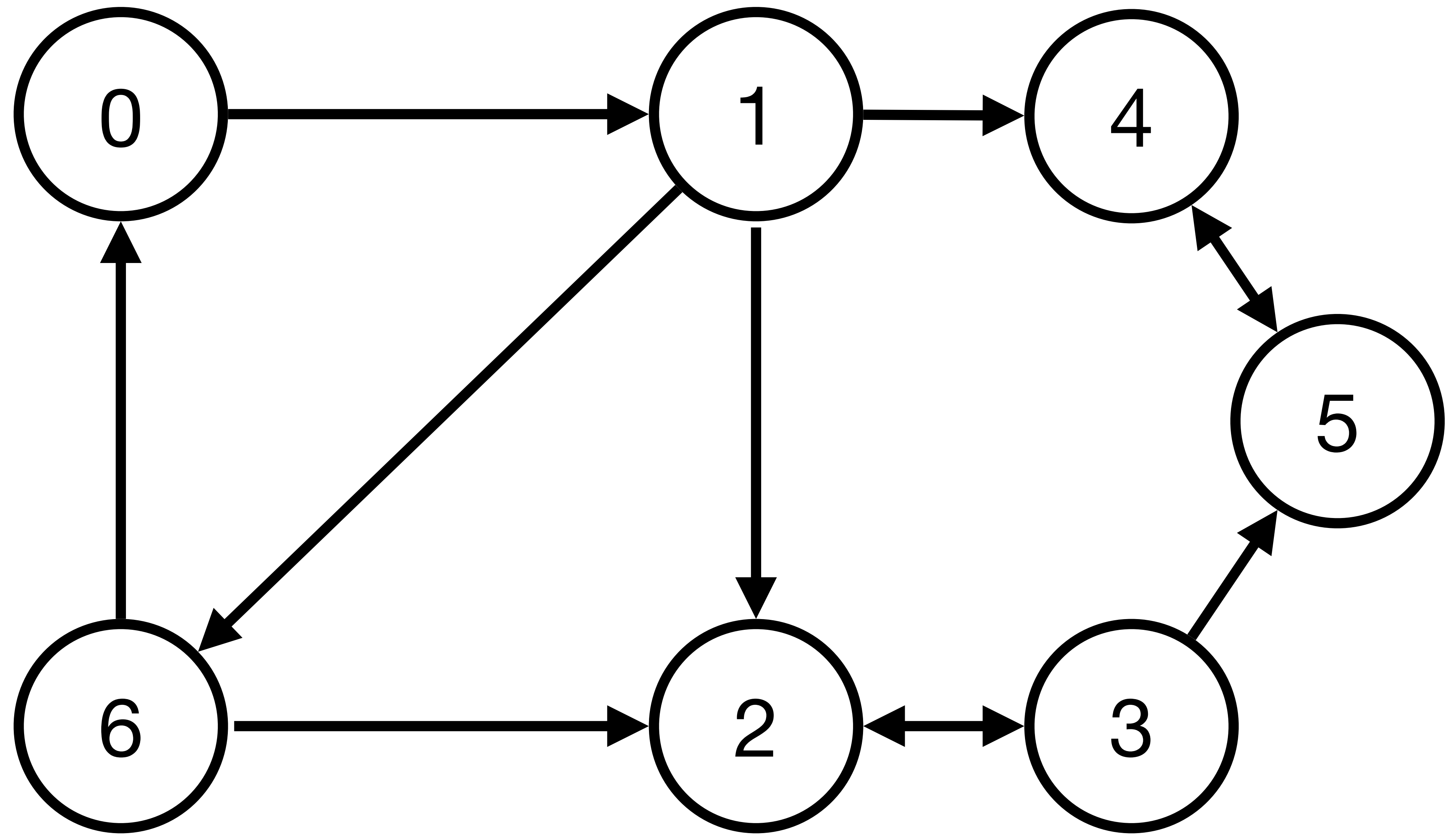
- **Key Observations:**
  - **Input:** Directed Graph G
  - **Output:** subgraph with vertices of SCC



Strongly Connected Components

Tarjan's Algorithm

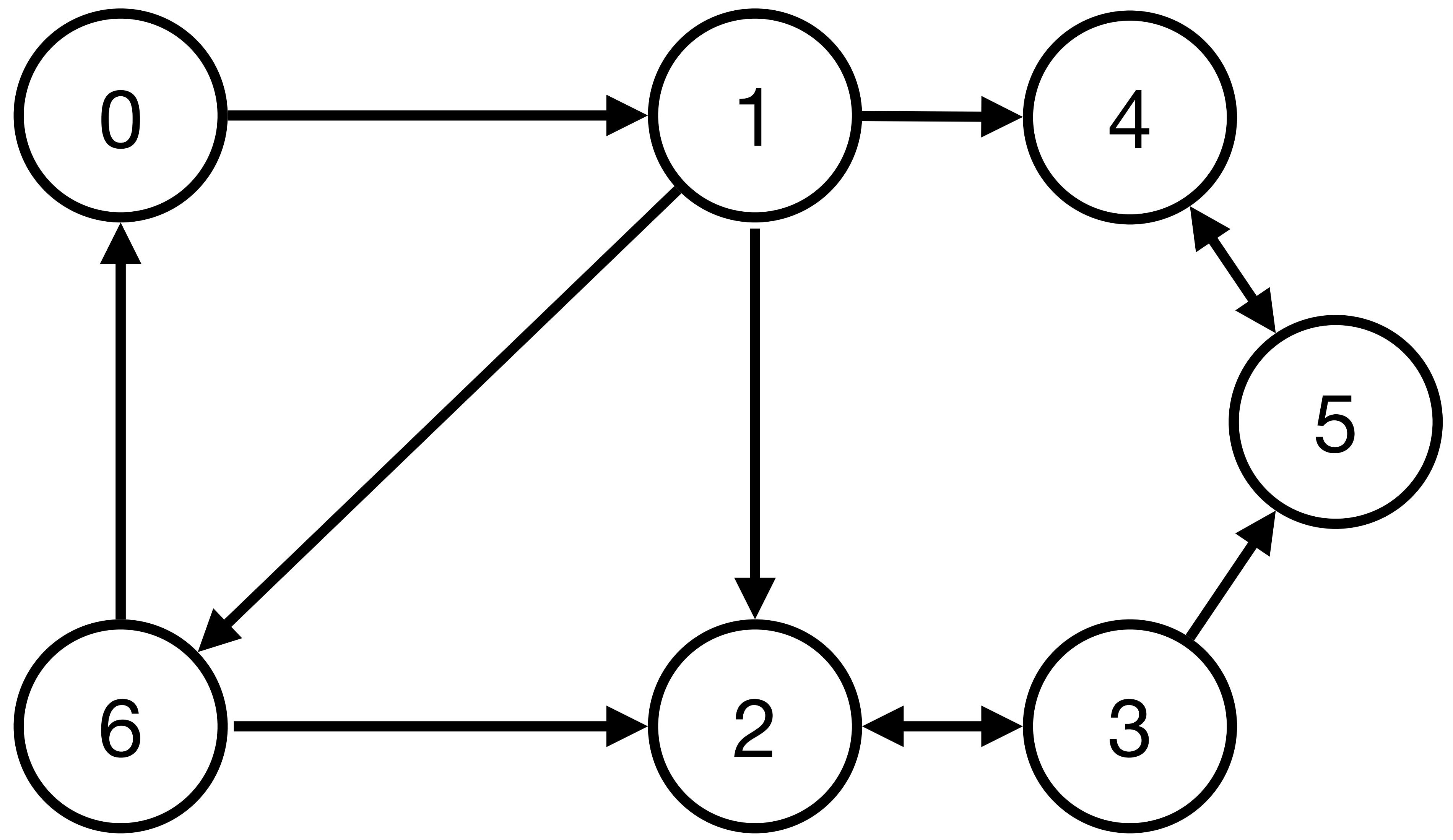
- **Key Observations:**
  - **Input:** Directed Graph G
  - **Output:** subgraph with vertices of SCC
  - Each vertex appears in exactly one SCC of the graph



Strongly Connected Components

Tarjan's Algorithm

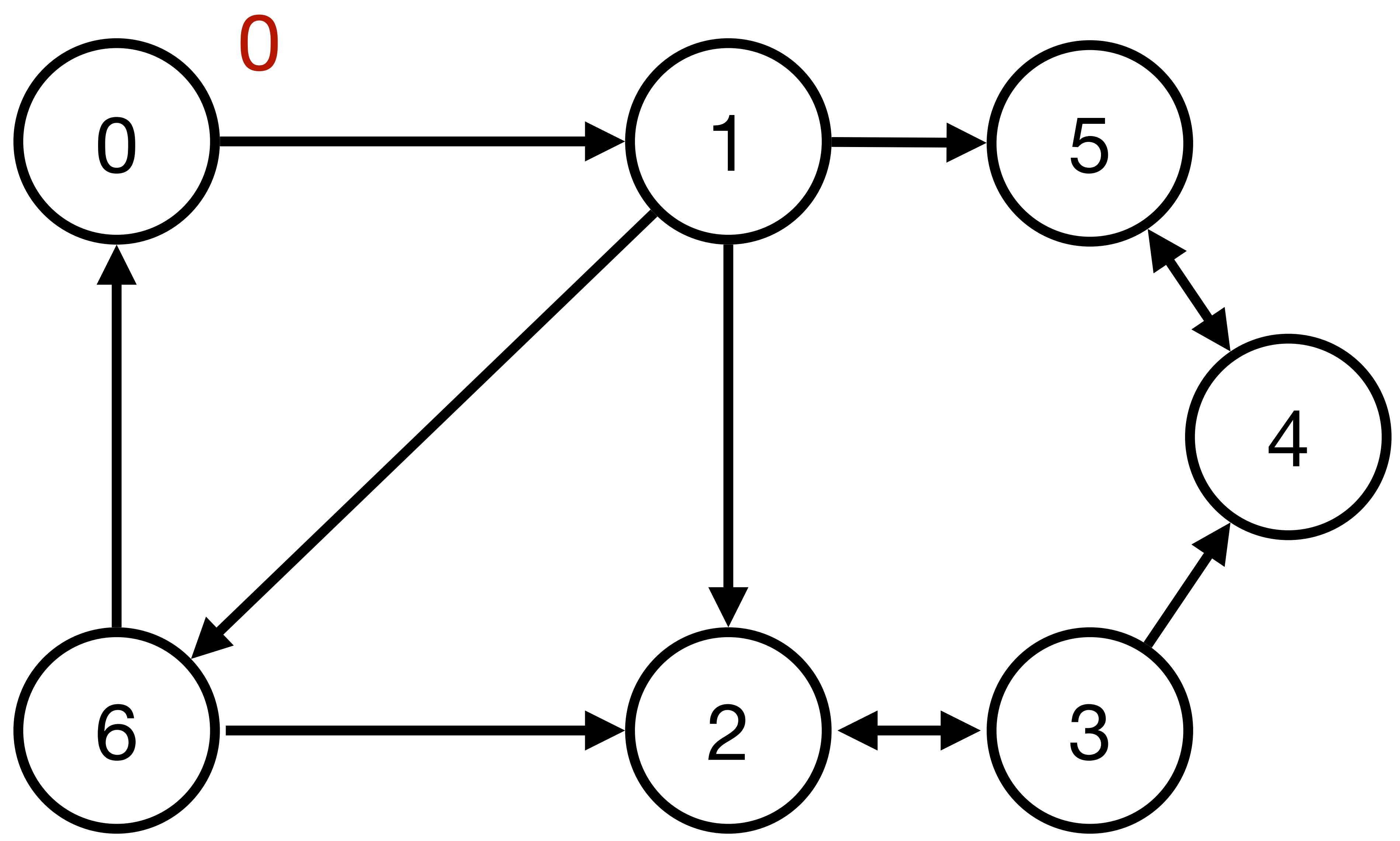
- **Key Observations:**
  - **Input:** Directed Graph G
  - **Output:** subgraph with vertices of SCC
  - Each vertex appears in exactly one SCC of the graph
  - Use of DFS + idea of **low-link values**



Strongly Connected Components

Tarjan's Algorithm

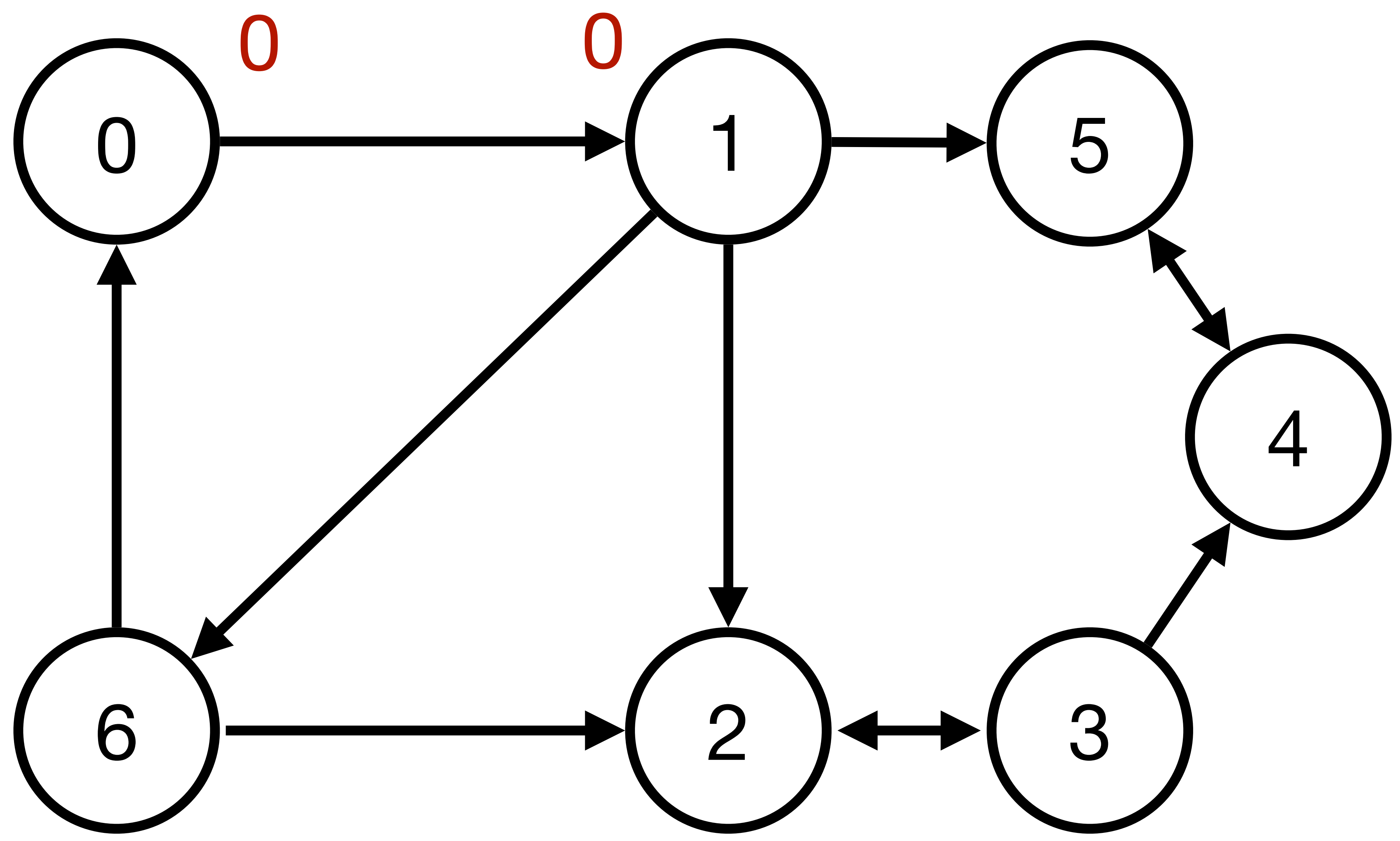
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**



Strongly Connected Components

Tarjan’s Algorithm

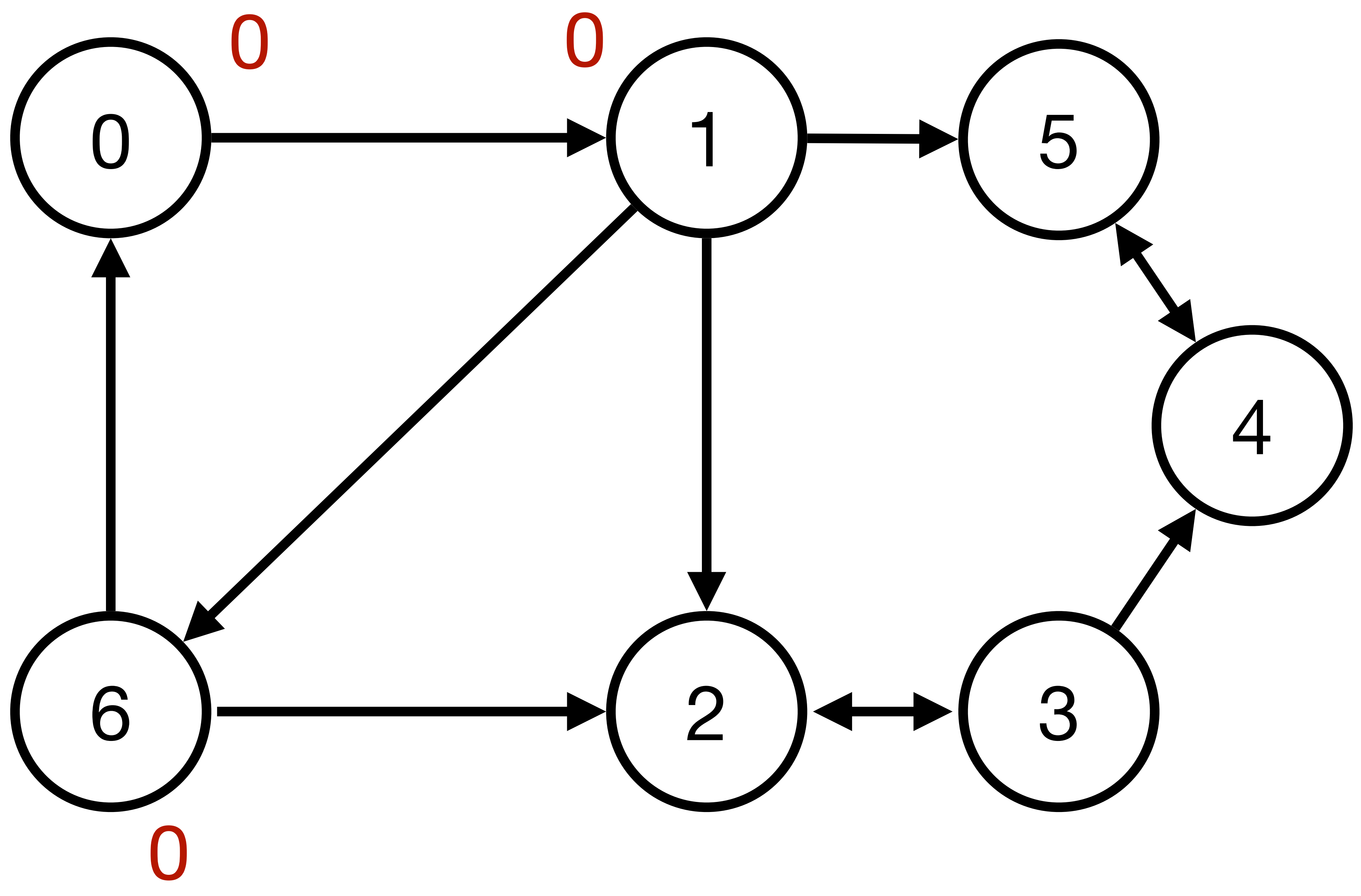
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan's Algorithm

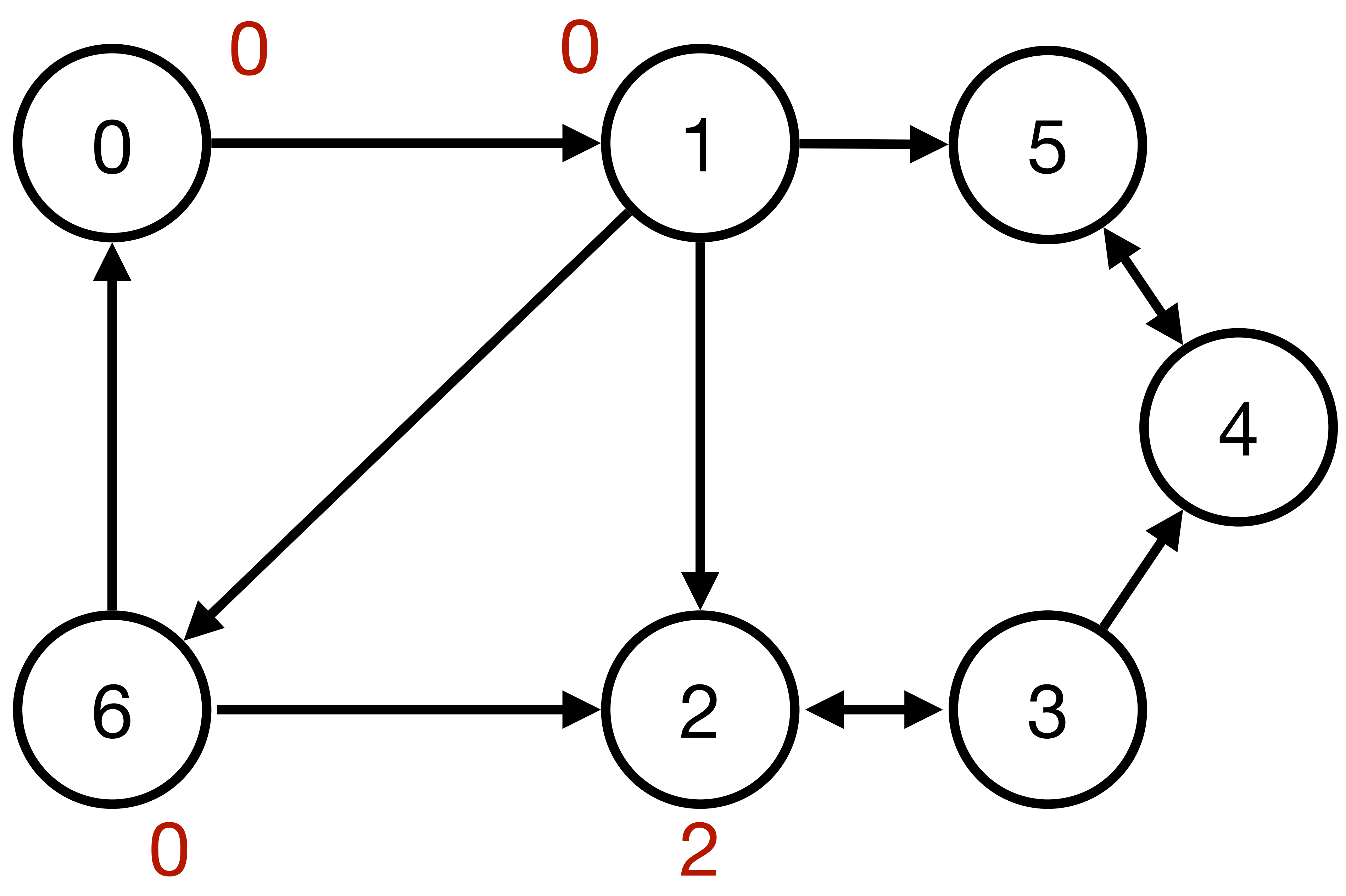
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan’s Algorithm

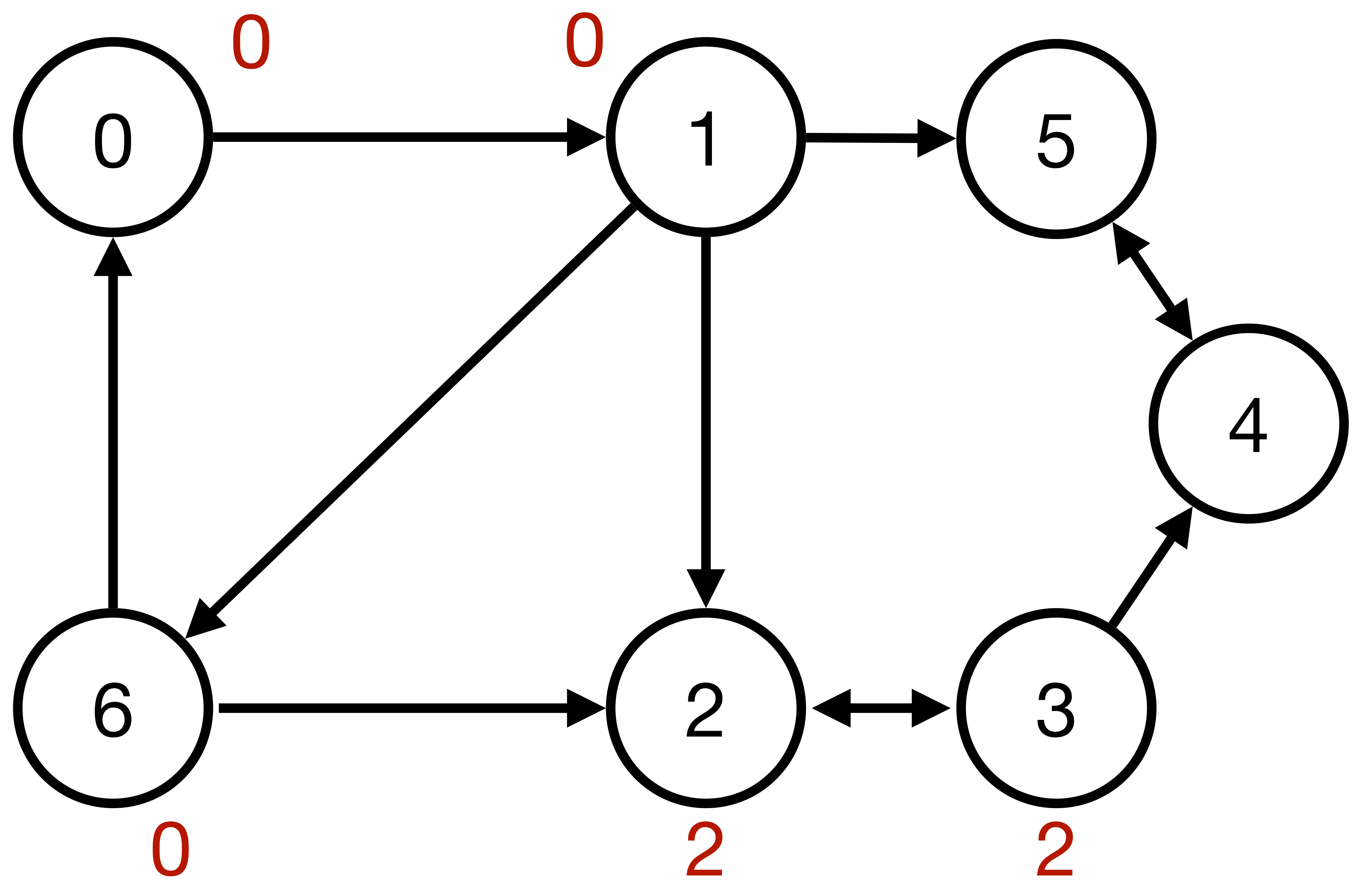
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan's Algorithm

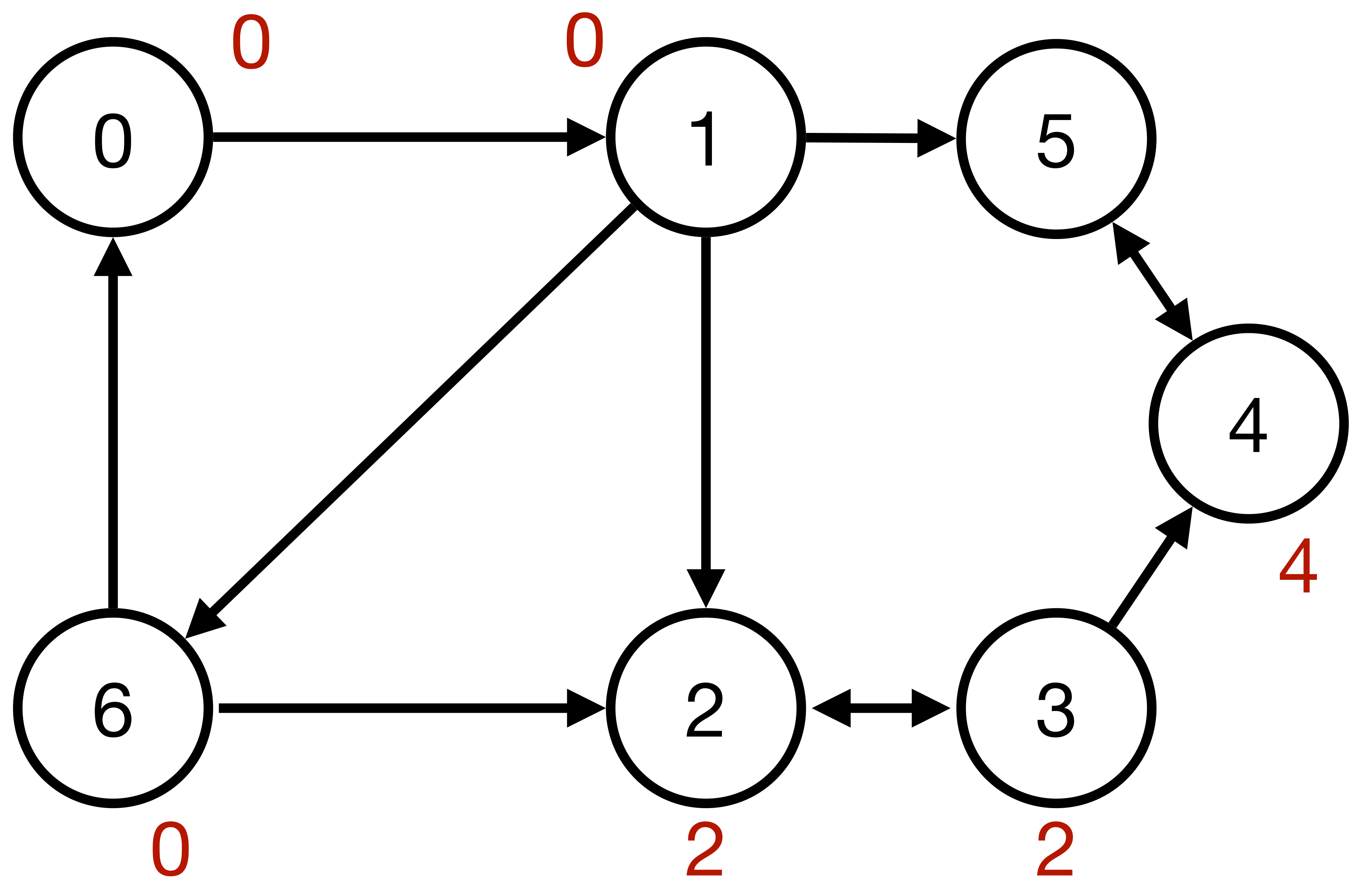
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**



Strongly Connected Components

Tarjan's Algorithm

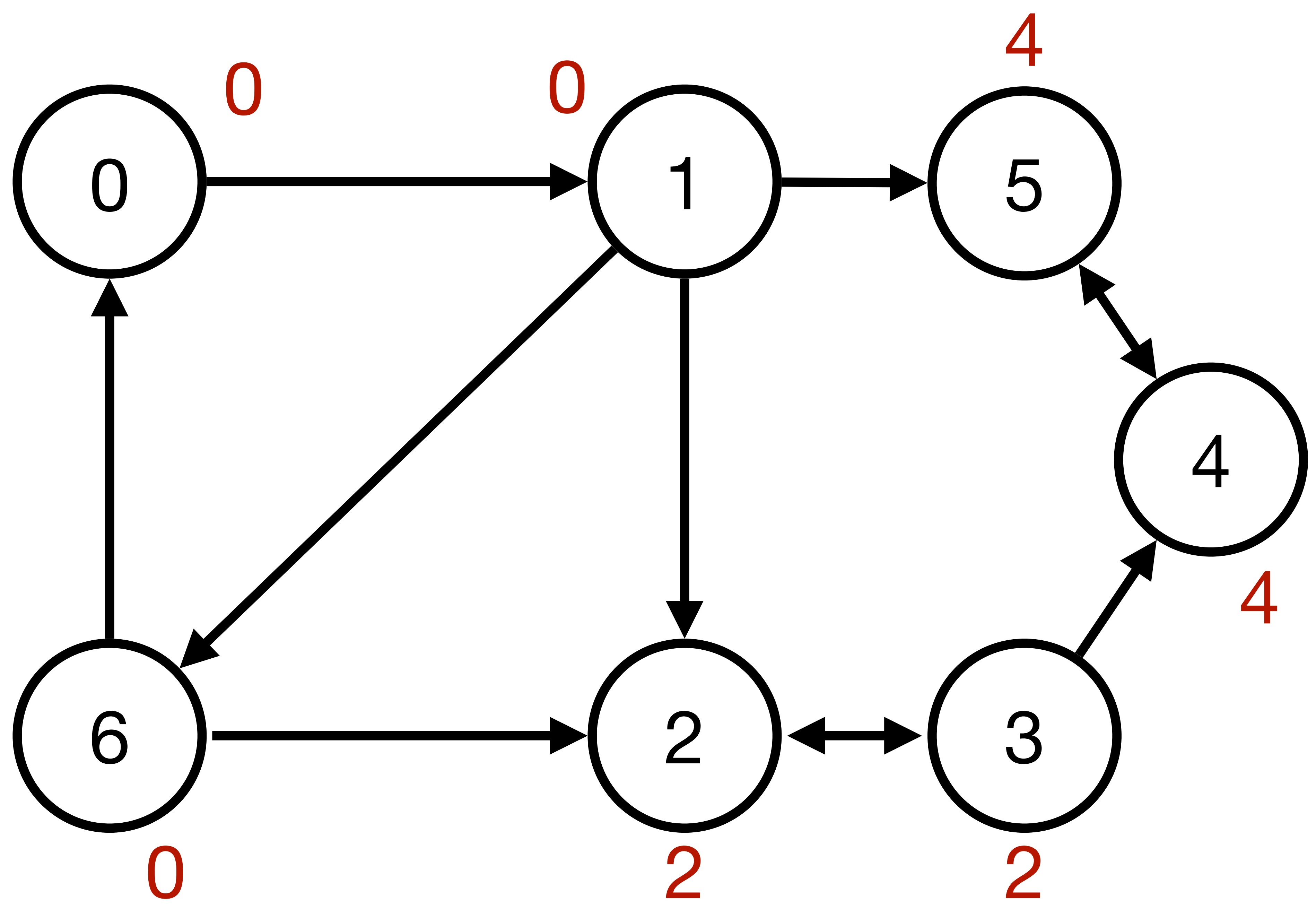
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan’s Algorithm

- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**

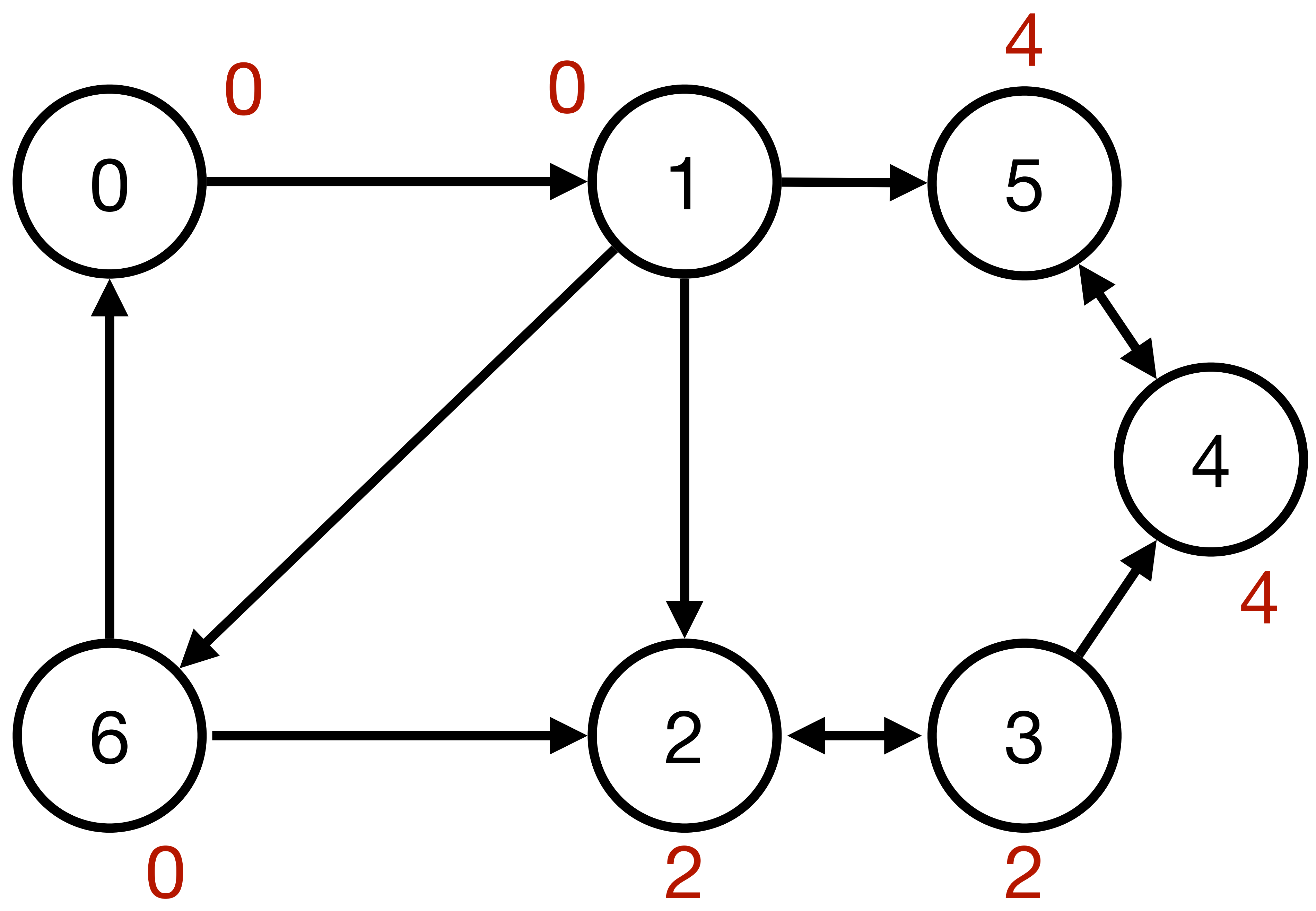


Strongly Connected Components

Tarjan’s Algorithm

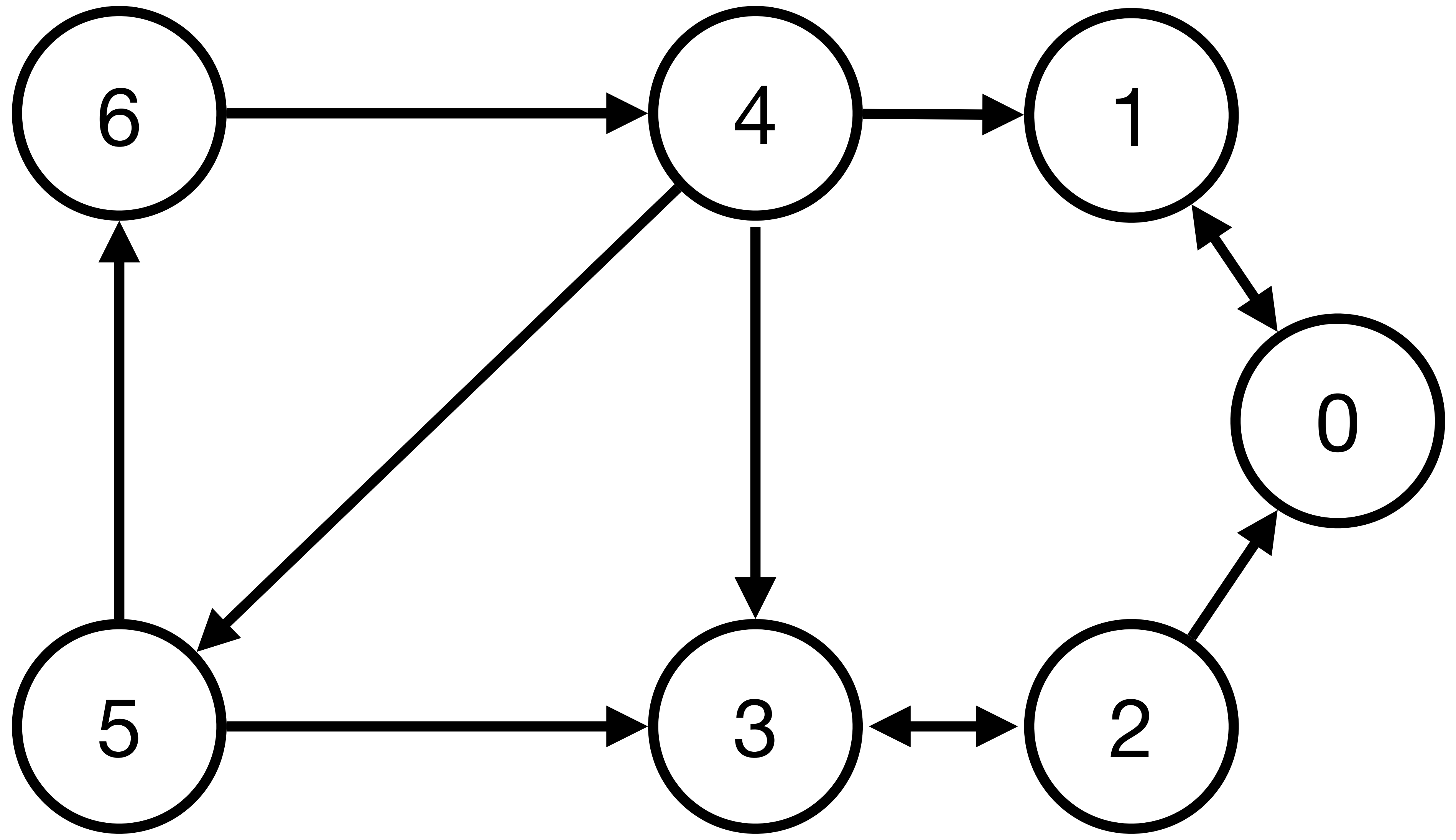
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).

- **Time Complexity:** $O(V\,.\,(V+E))$



Strongly Connected Components

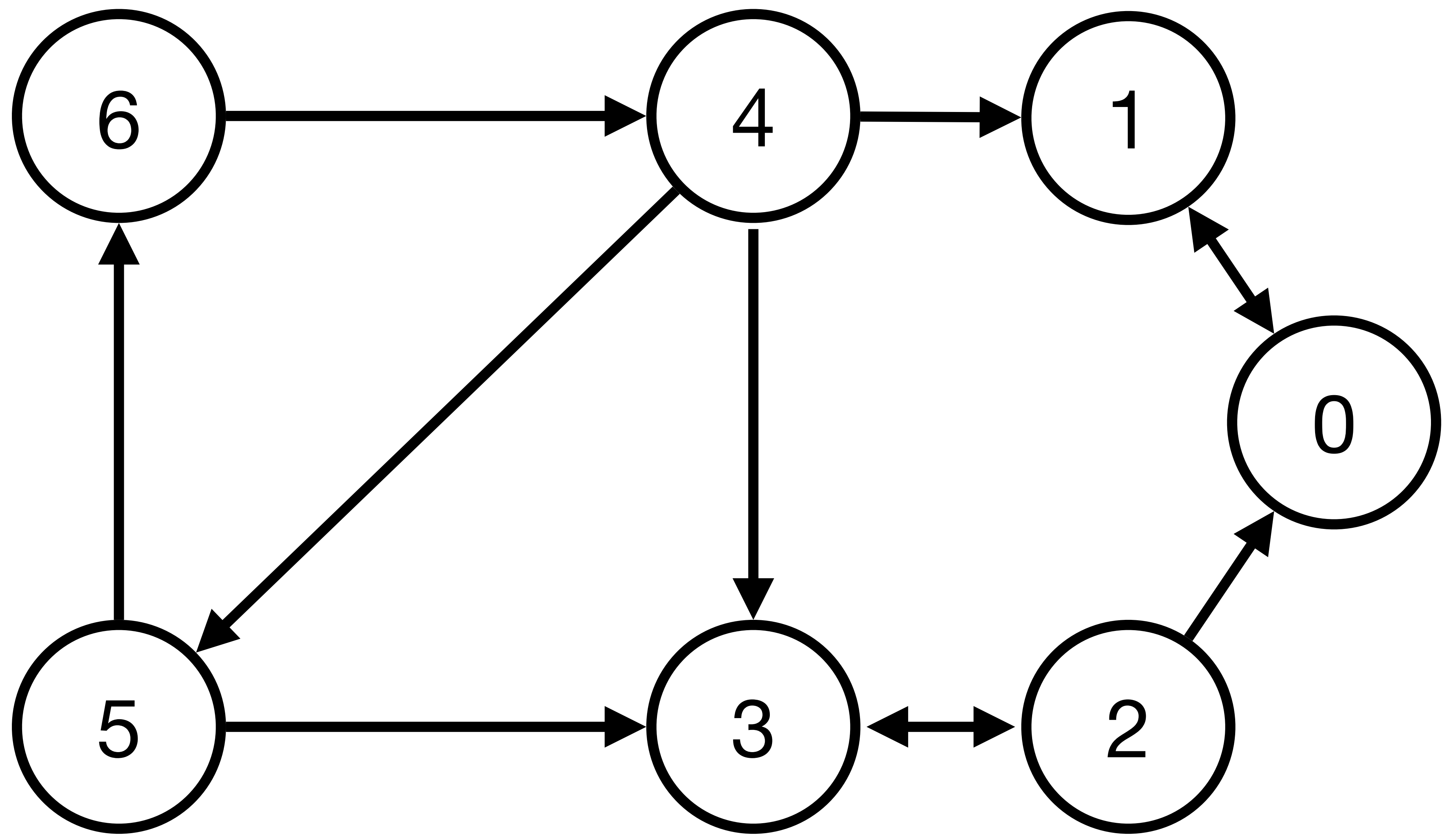
Tarjan's Algorithm



Strongly Connected Components

Tarjan’s Algorithm

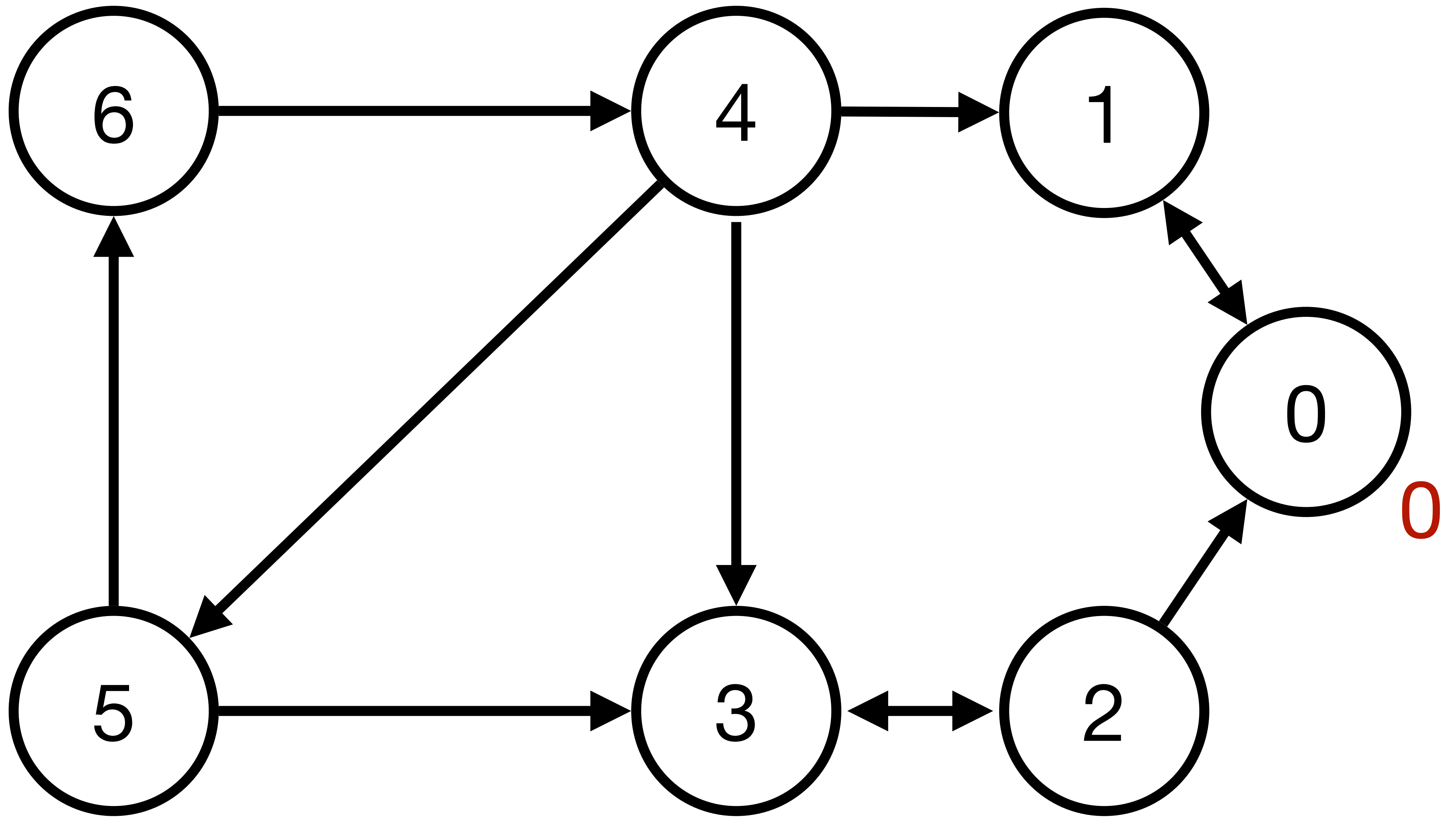
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**



Strongly Connected Components

Tarjan's Algorithm

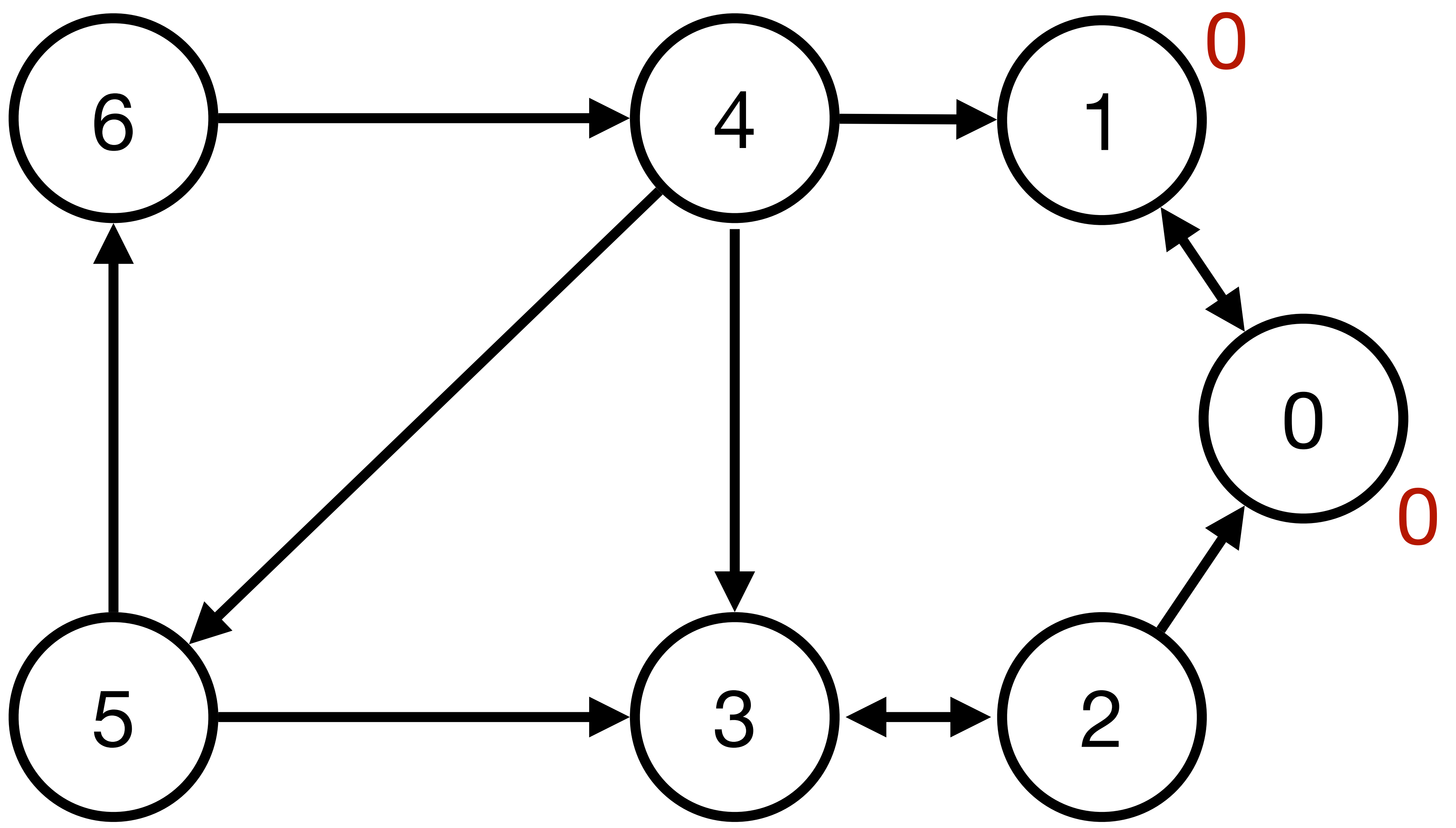
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan's Algorithm

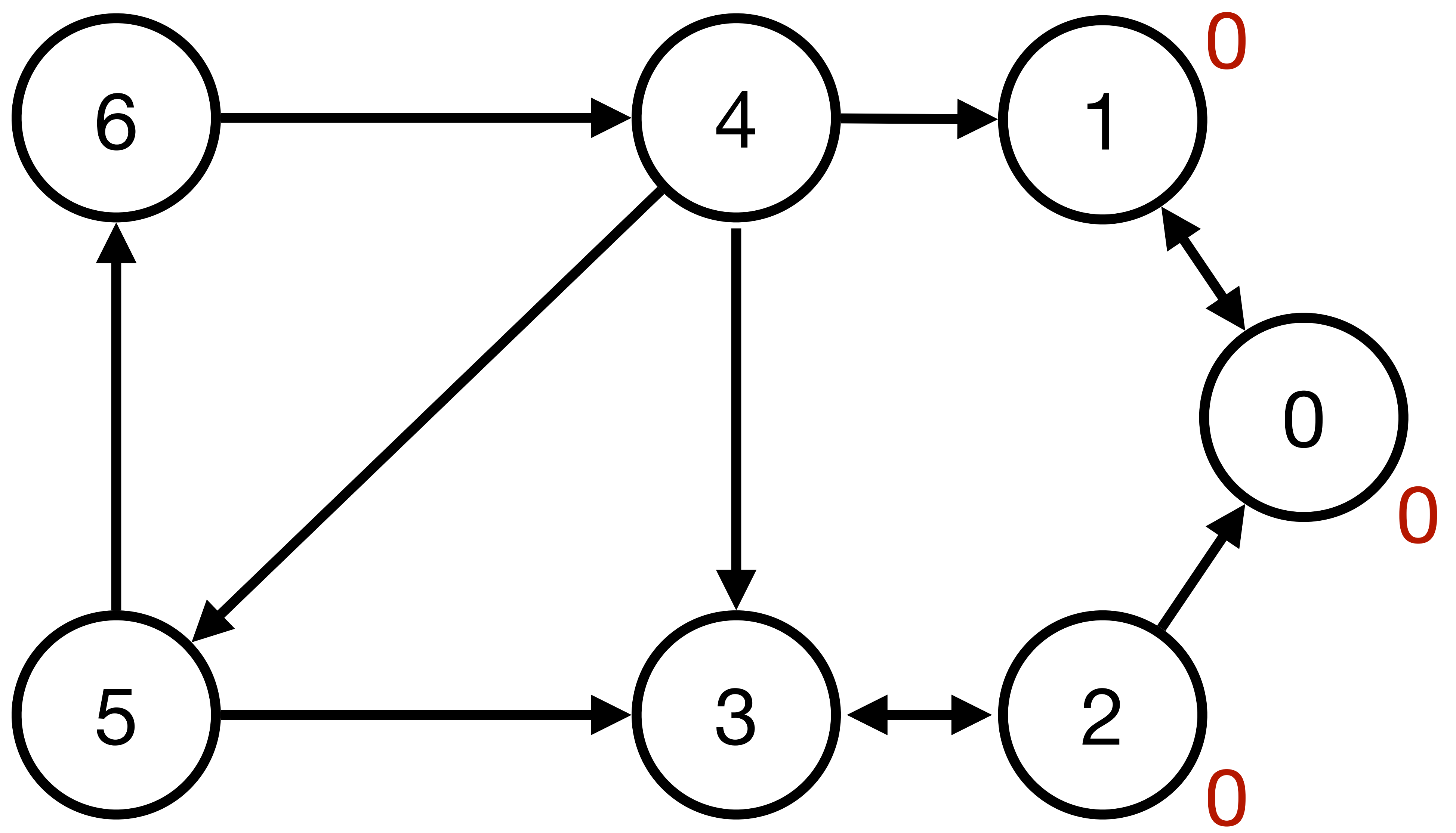
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**



Strongly Connected Components

Tarjan's Algorithm

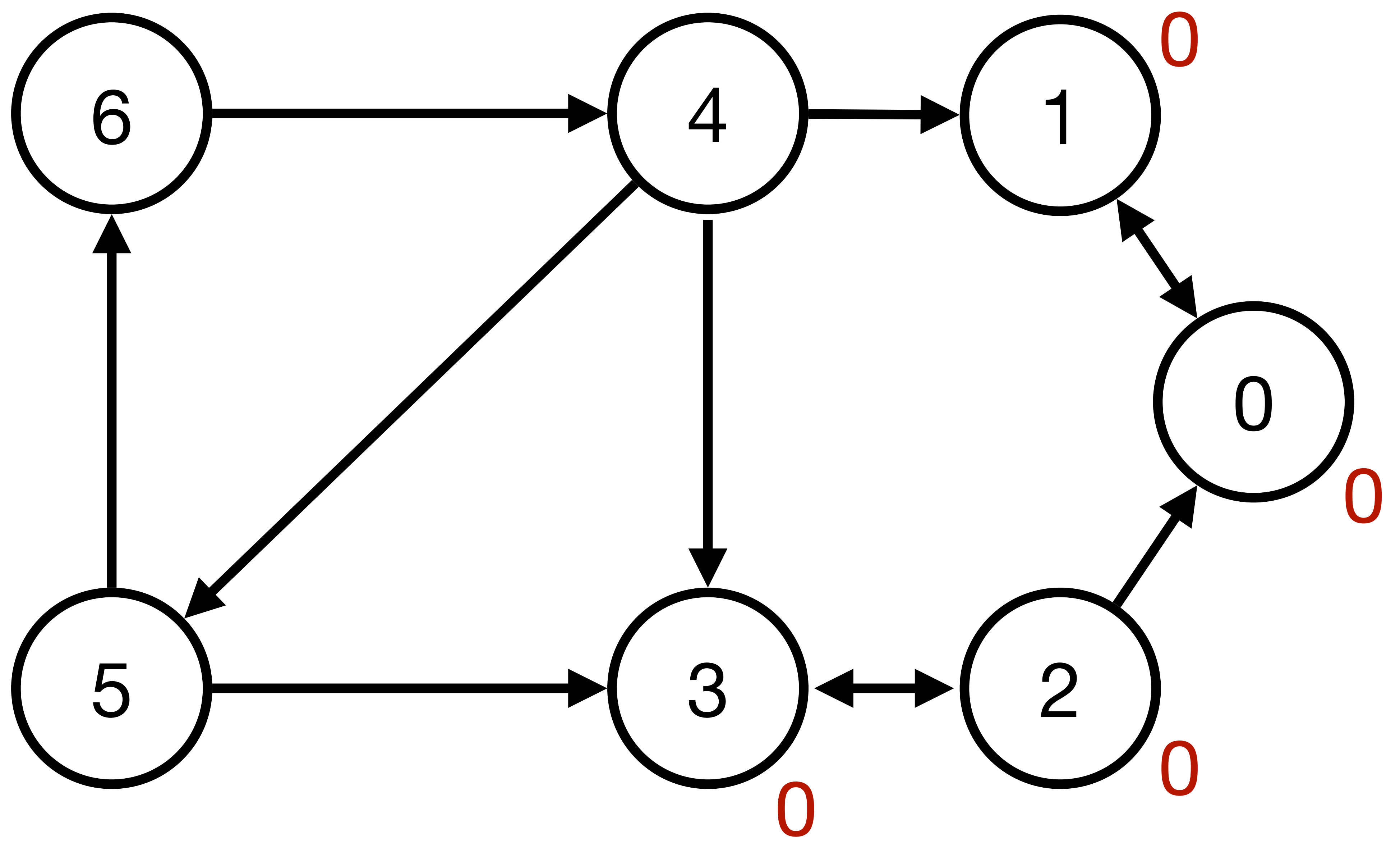
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan’s Algorithm

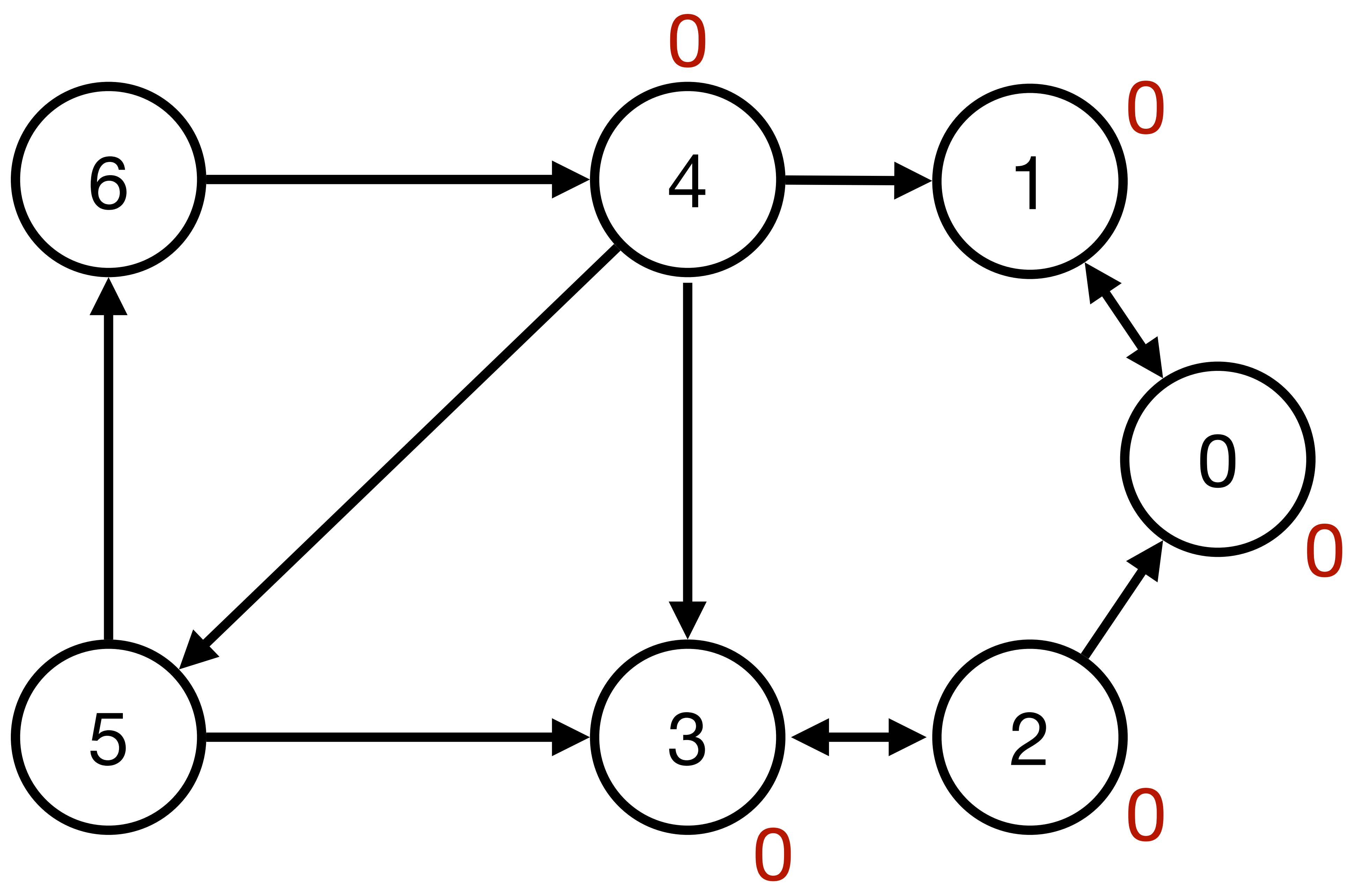
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan’s Algorithm

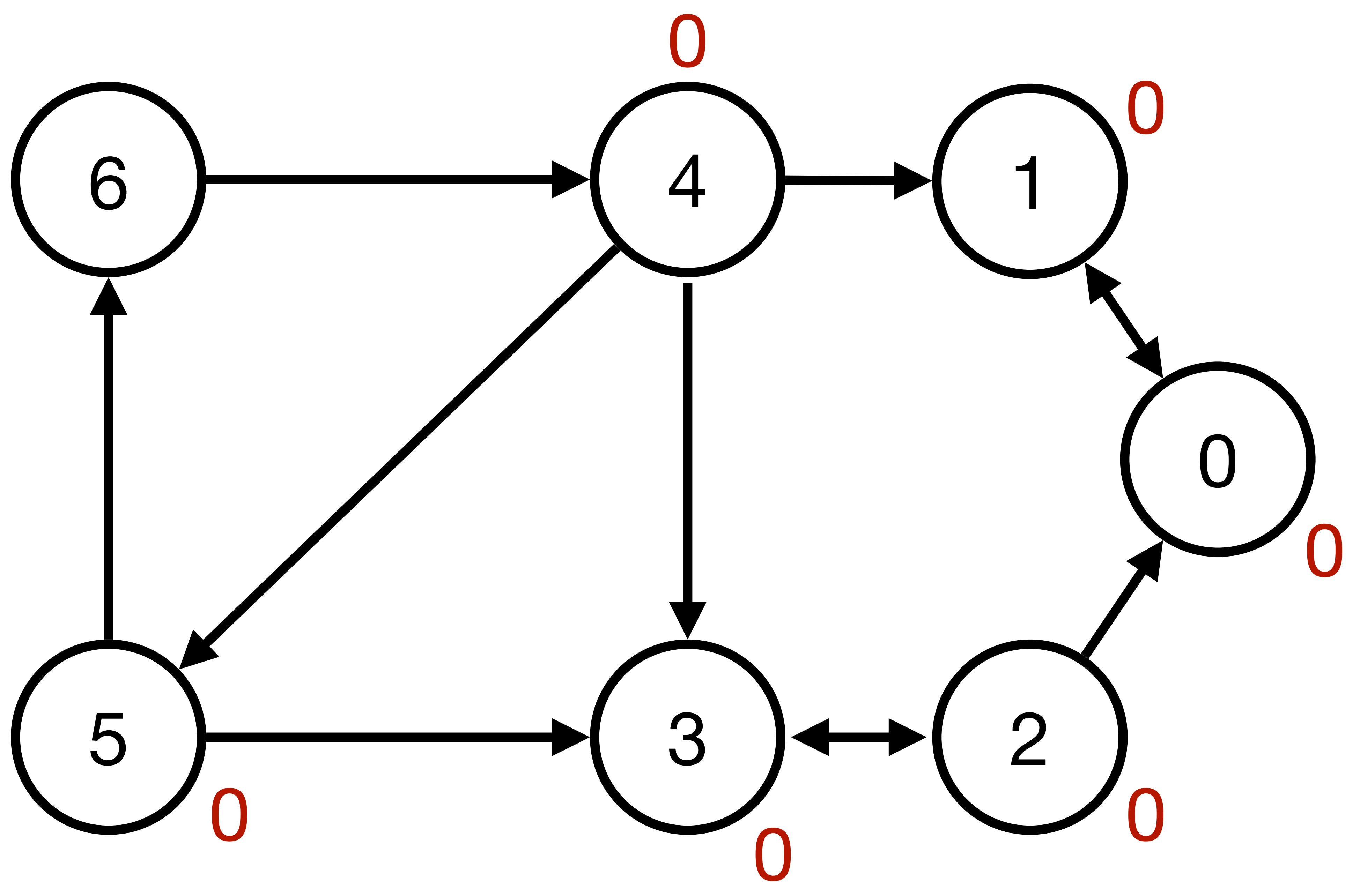
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan's Algorithm

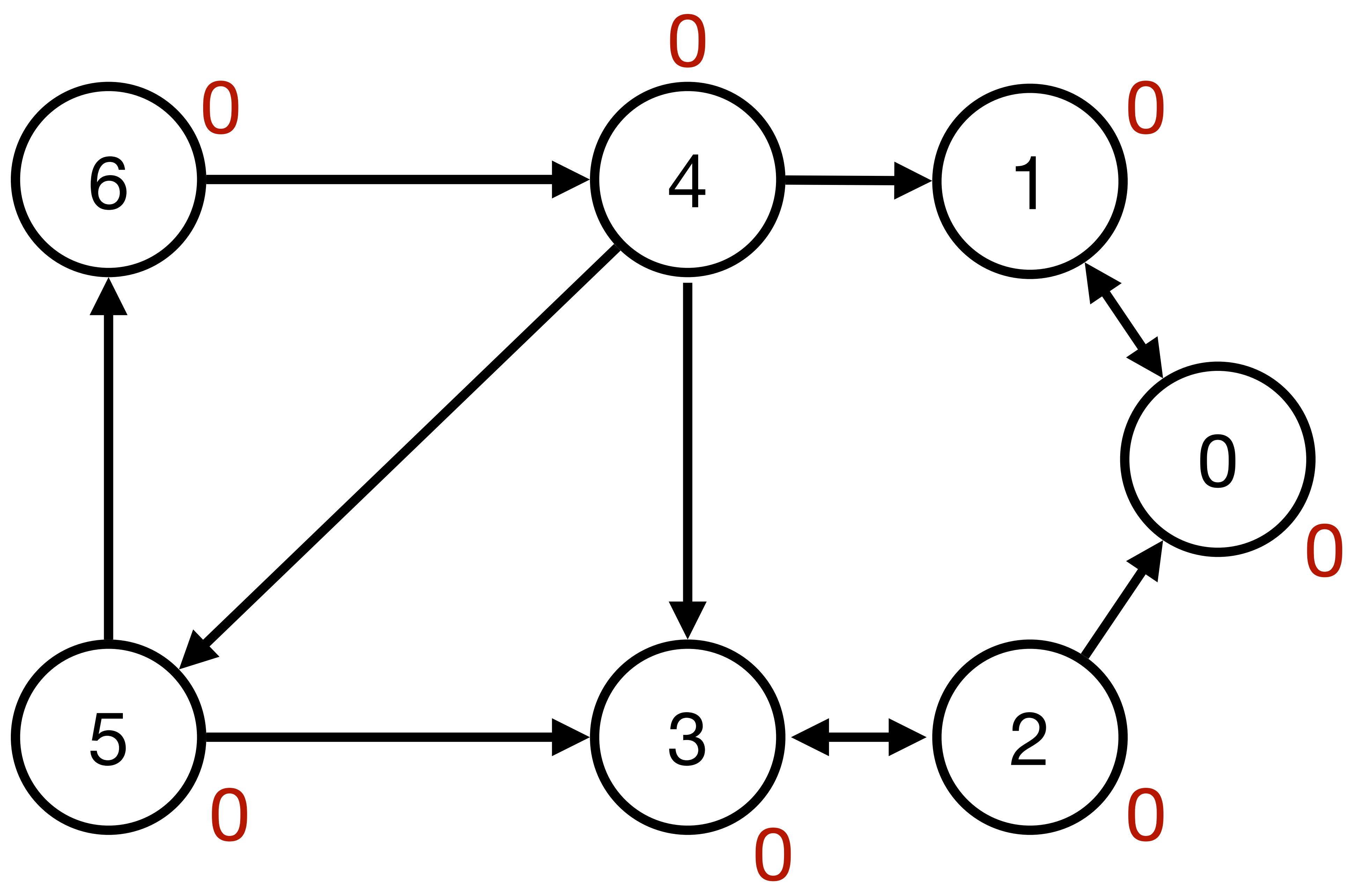
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan’s Algorithm

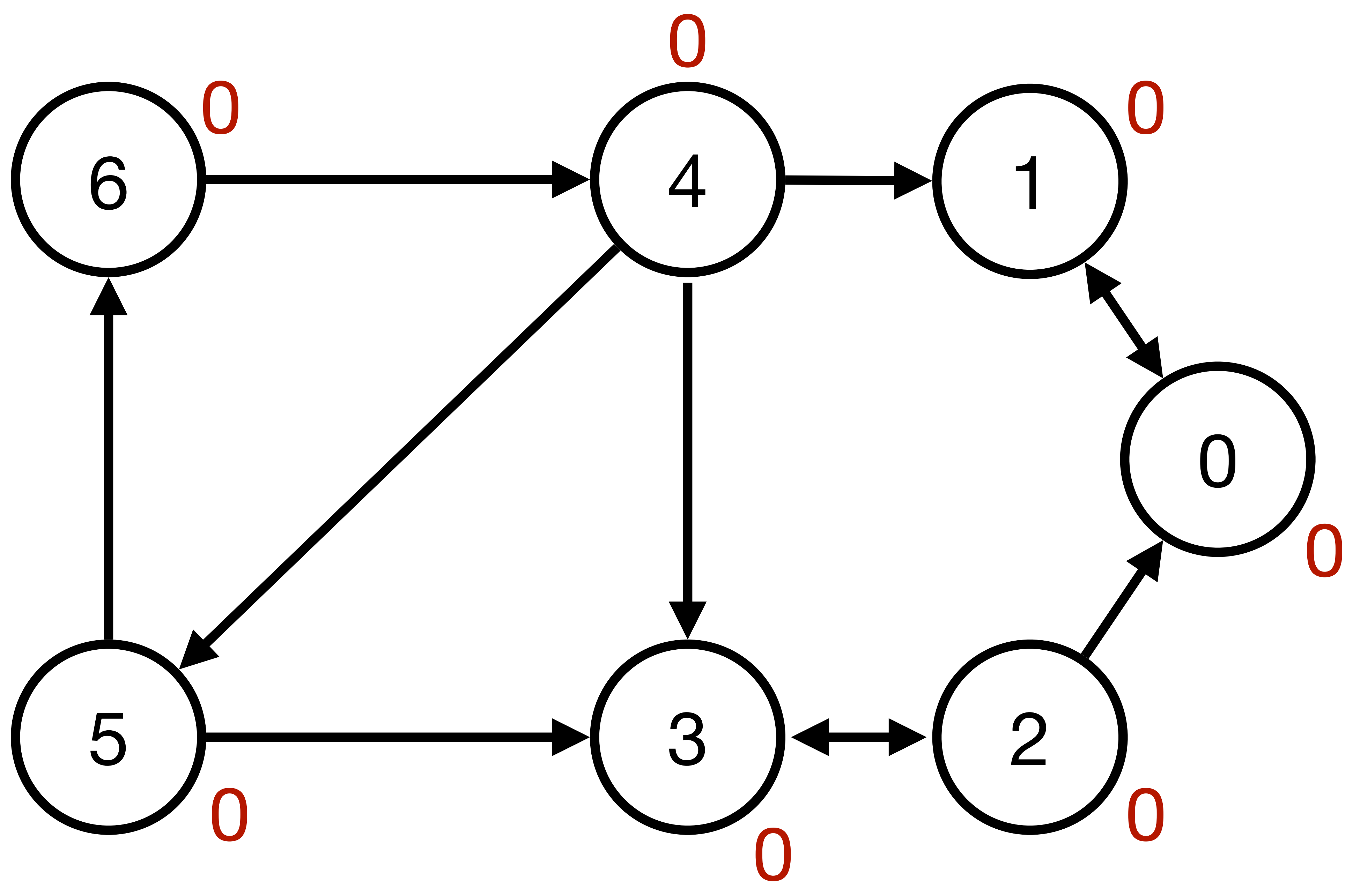
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself).



Strongly Connected Components

Tarjan’s Algorithm

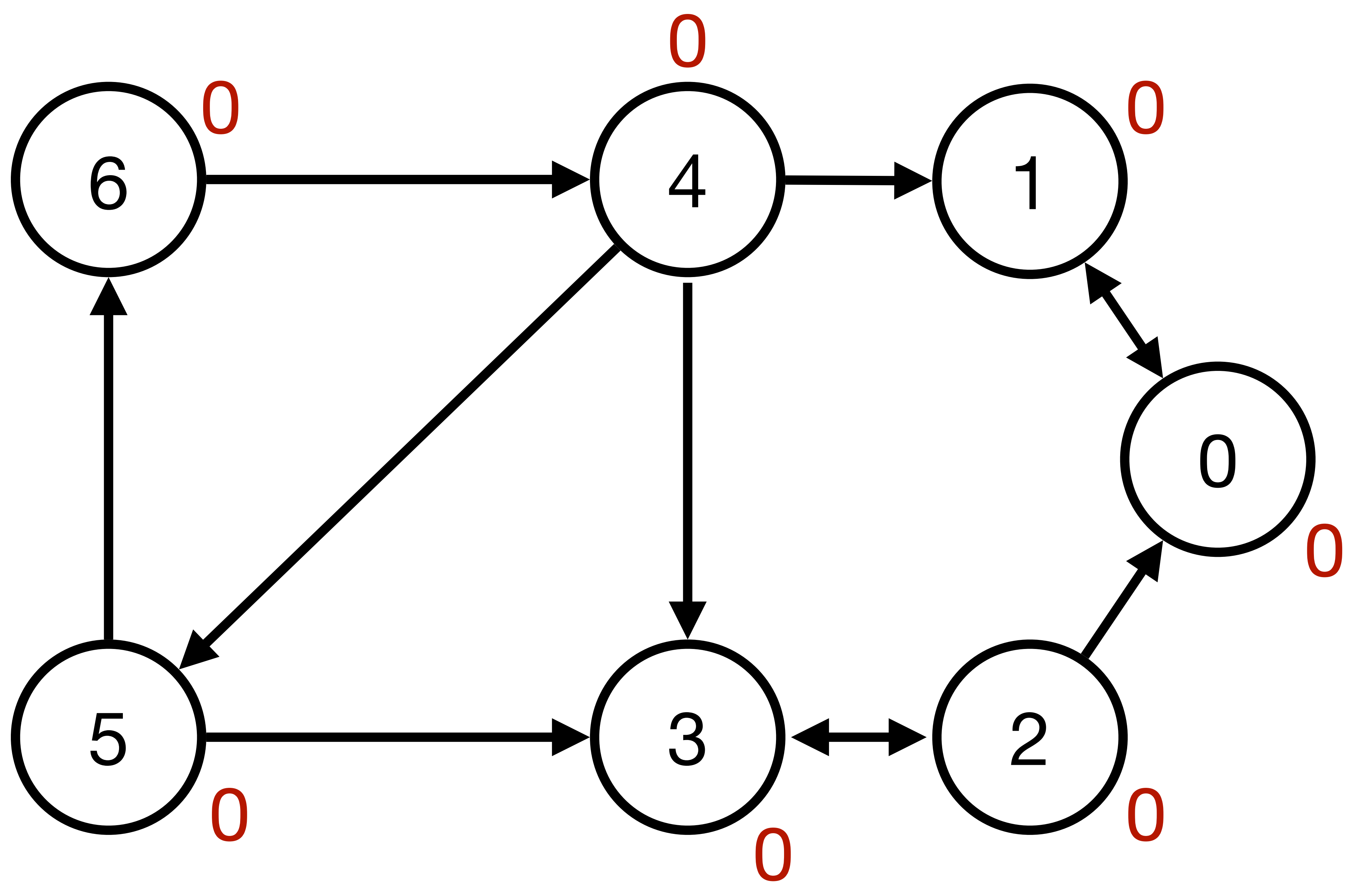
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**
  - **CAUTION:** LL values are dependent in the order of exploration



Strongly Connected Components

Tarjan's Algorithm

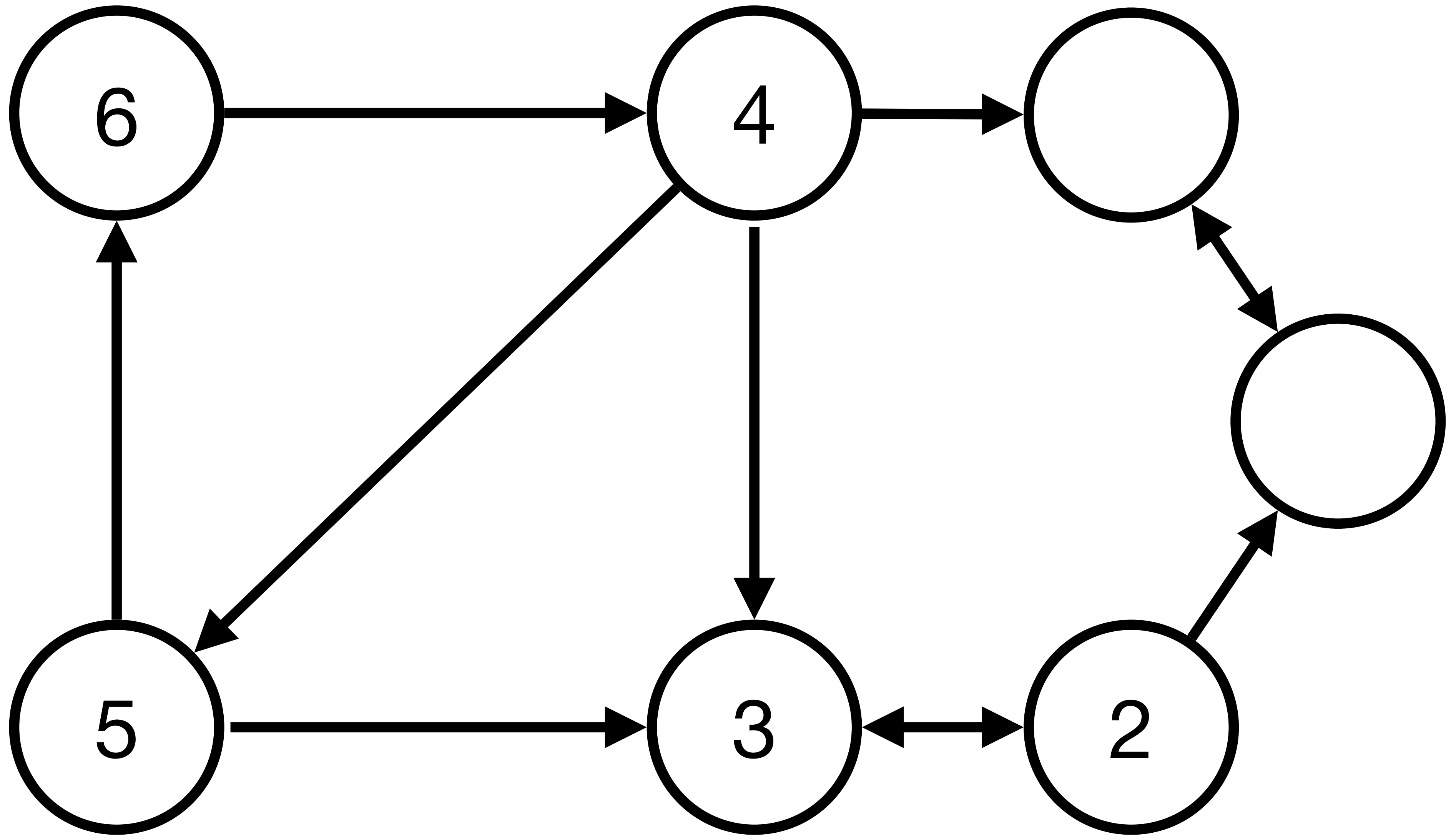
- **Low-link values:** An LL value of a node is the **smallest** node id reachable from that node (including itself)**.**
  - **CAUTION:** LL values are dependent in the order of exploration



- **Incorrect SCC was computed**

Strongly Connected Components

Tarjan's Algorithm

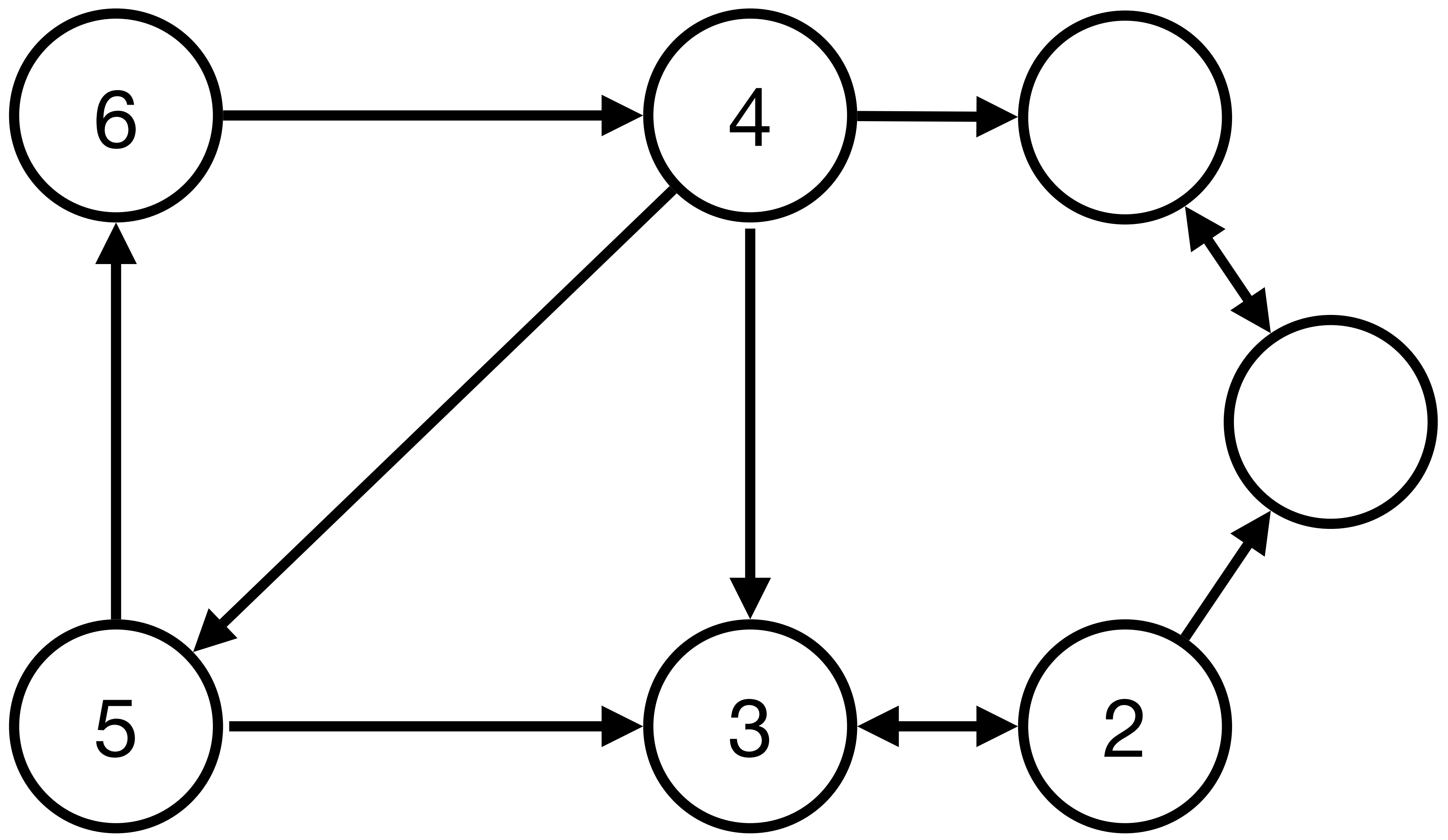


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:

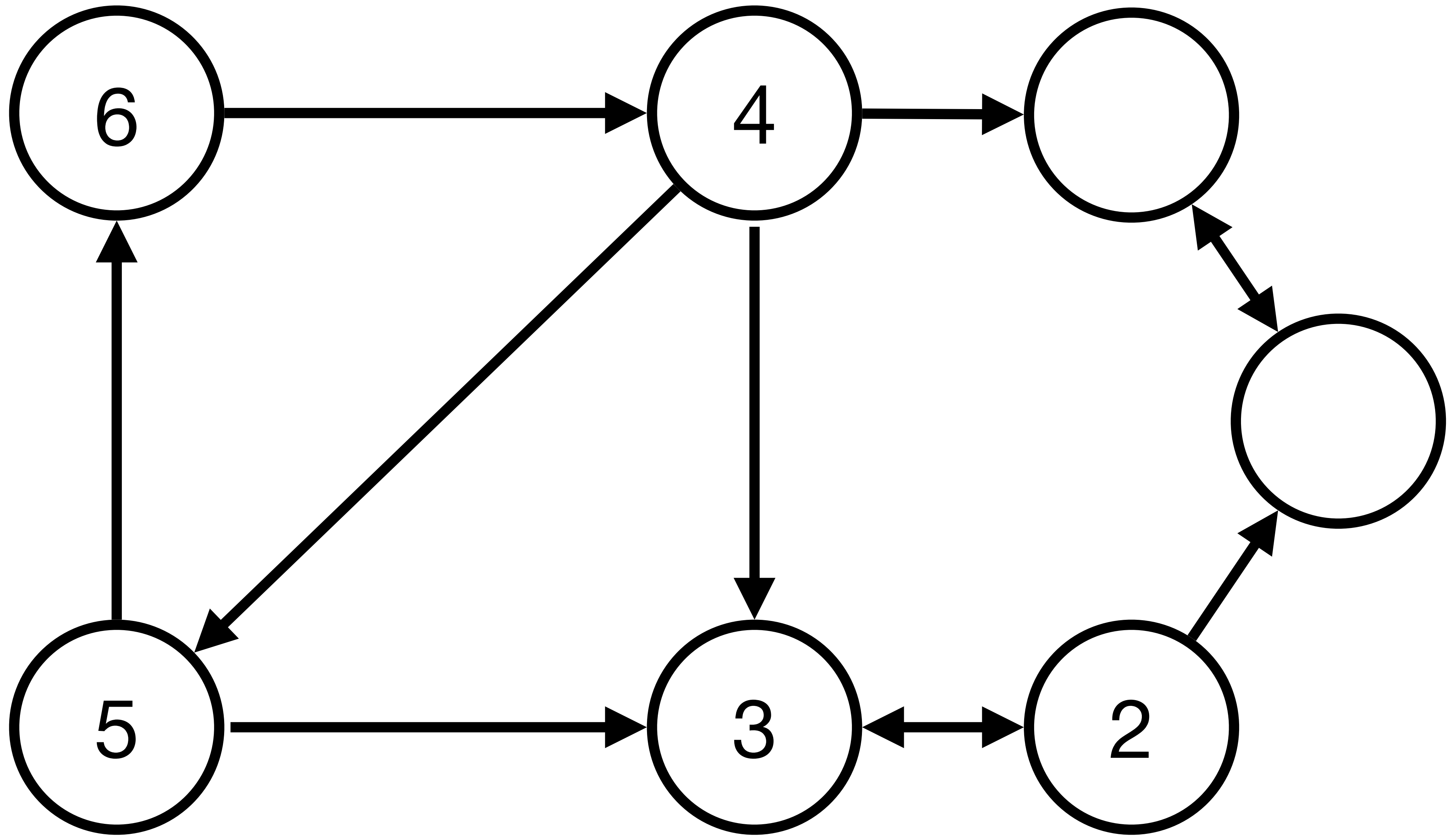


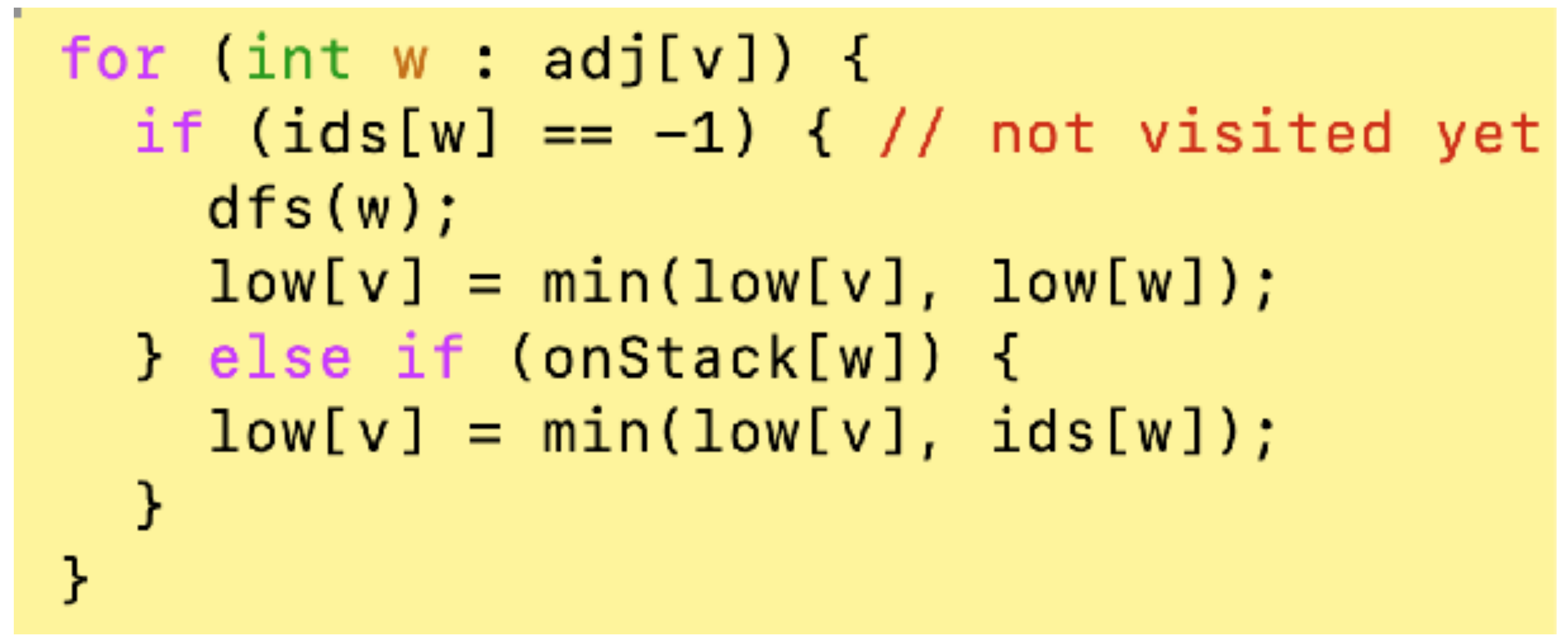
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
    - Start DFS from a node



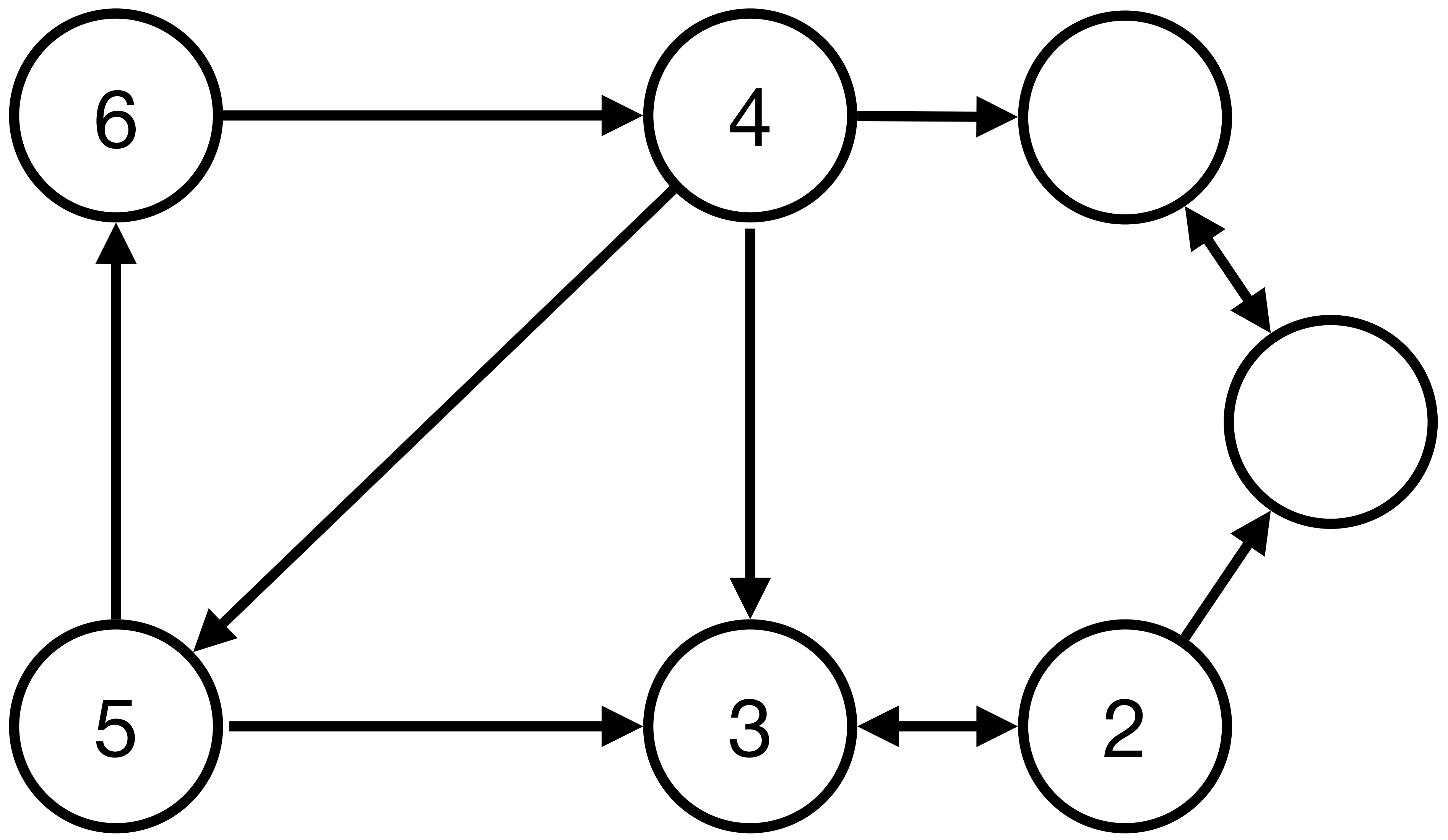


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value

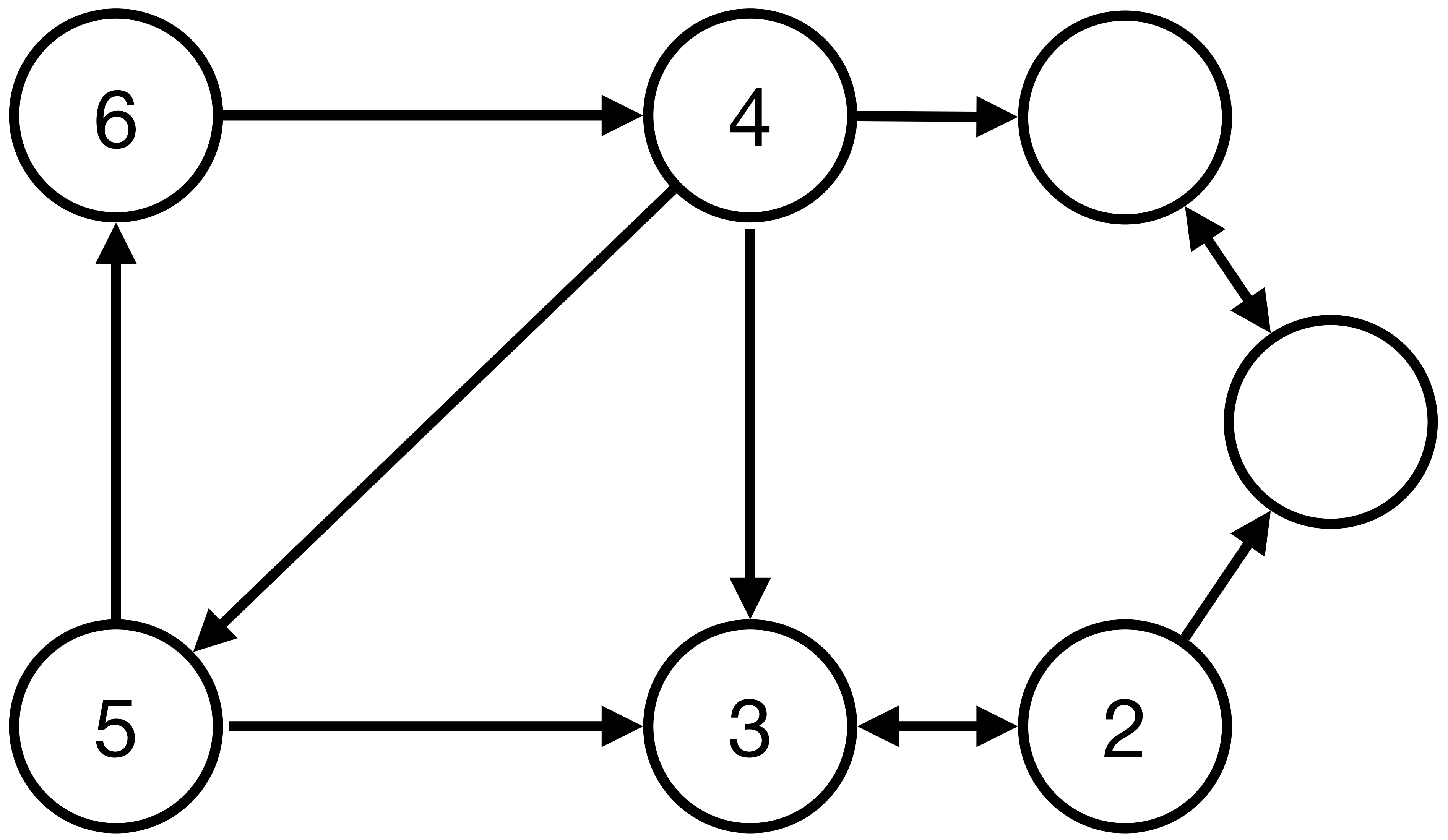


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes

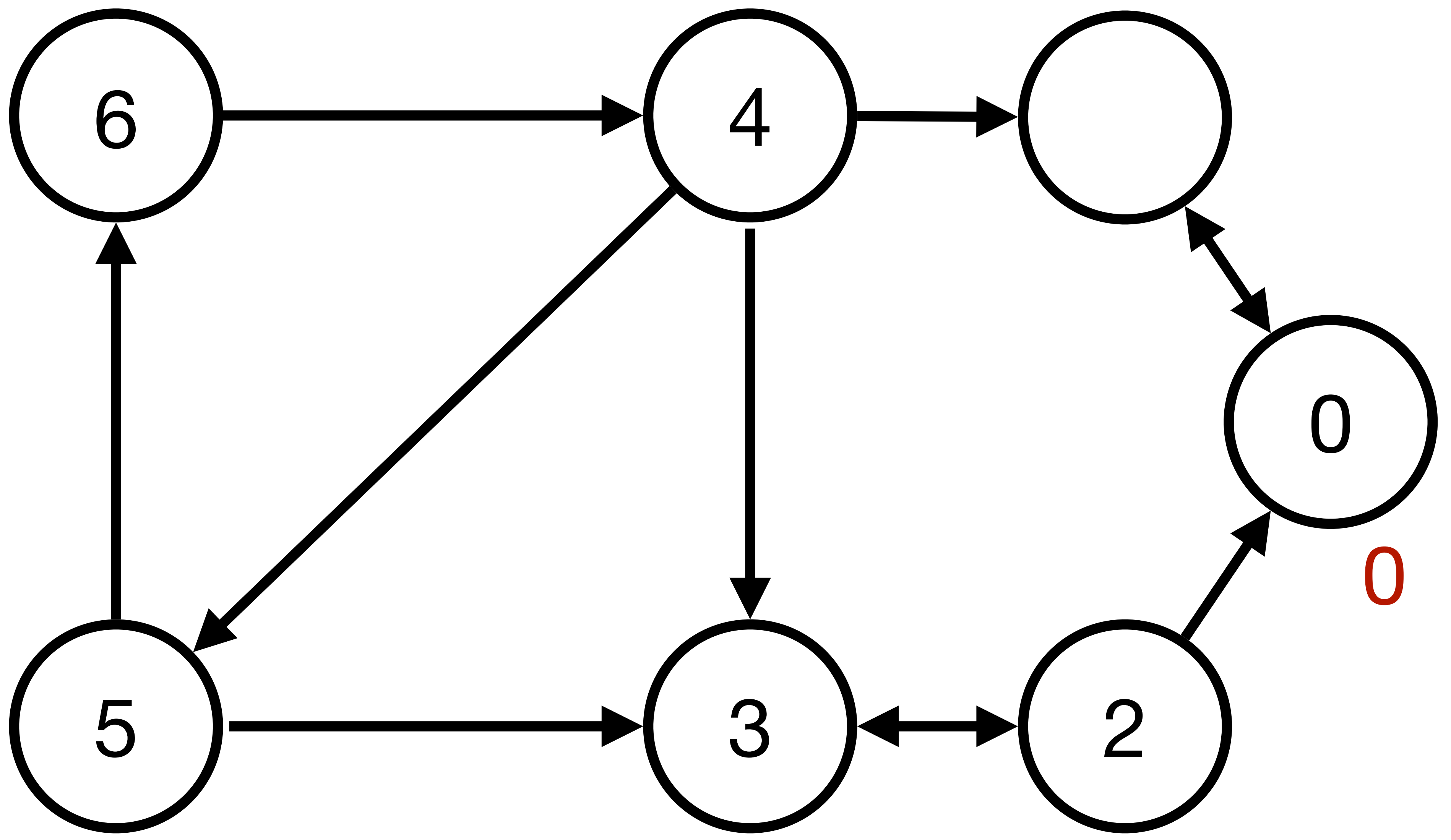


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes

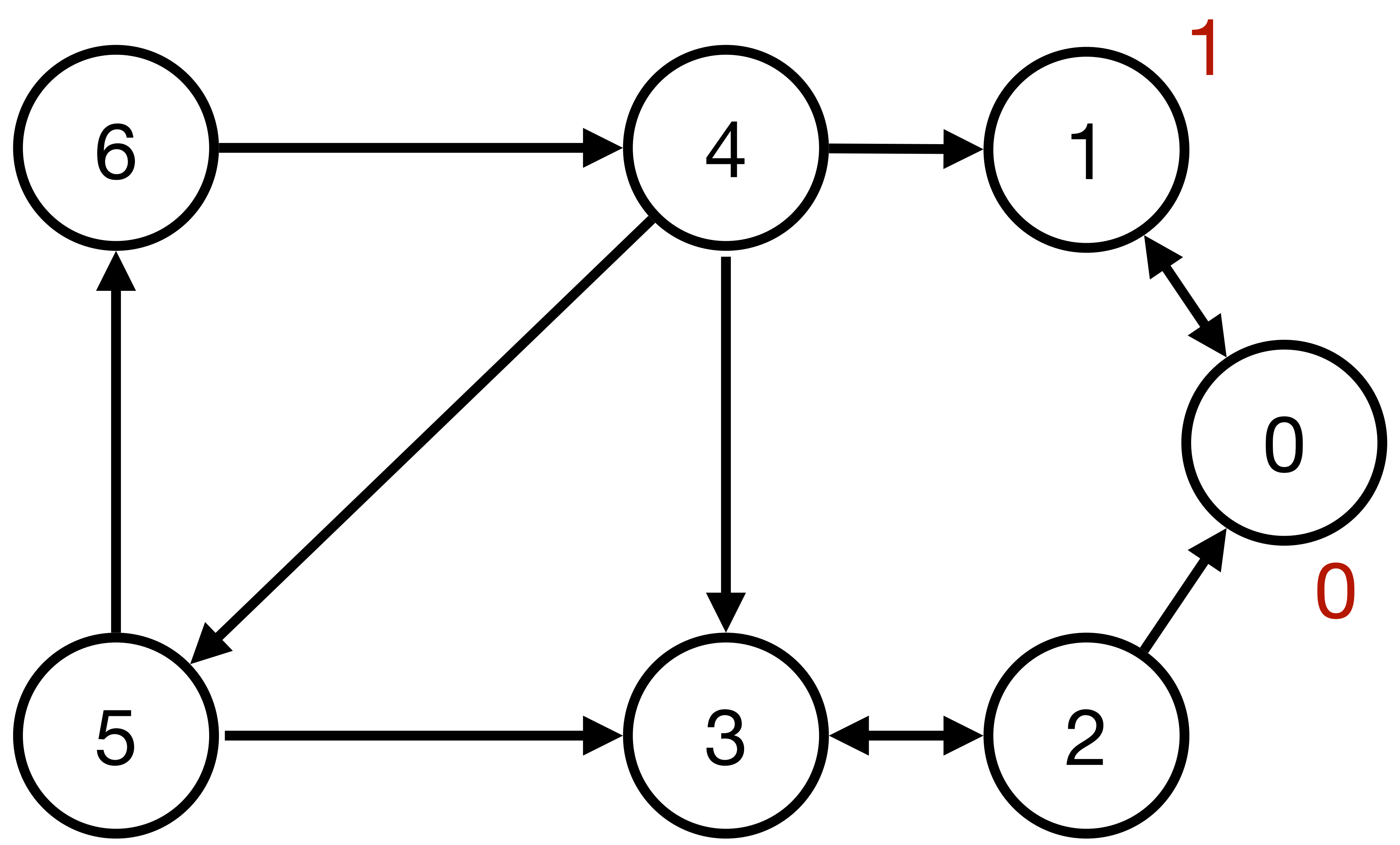


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan’s Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes

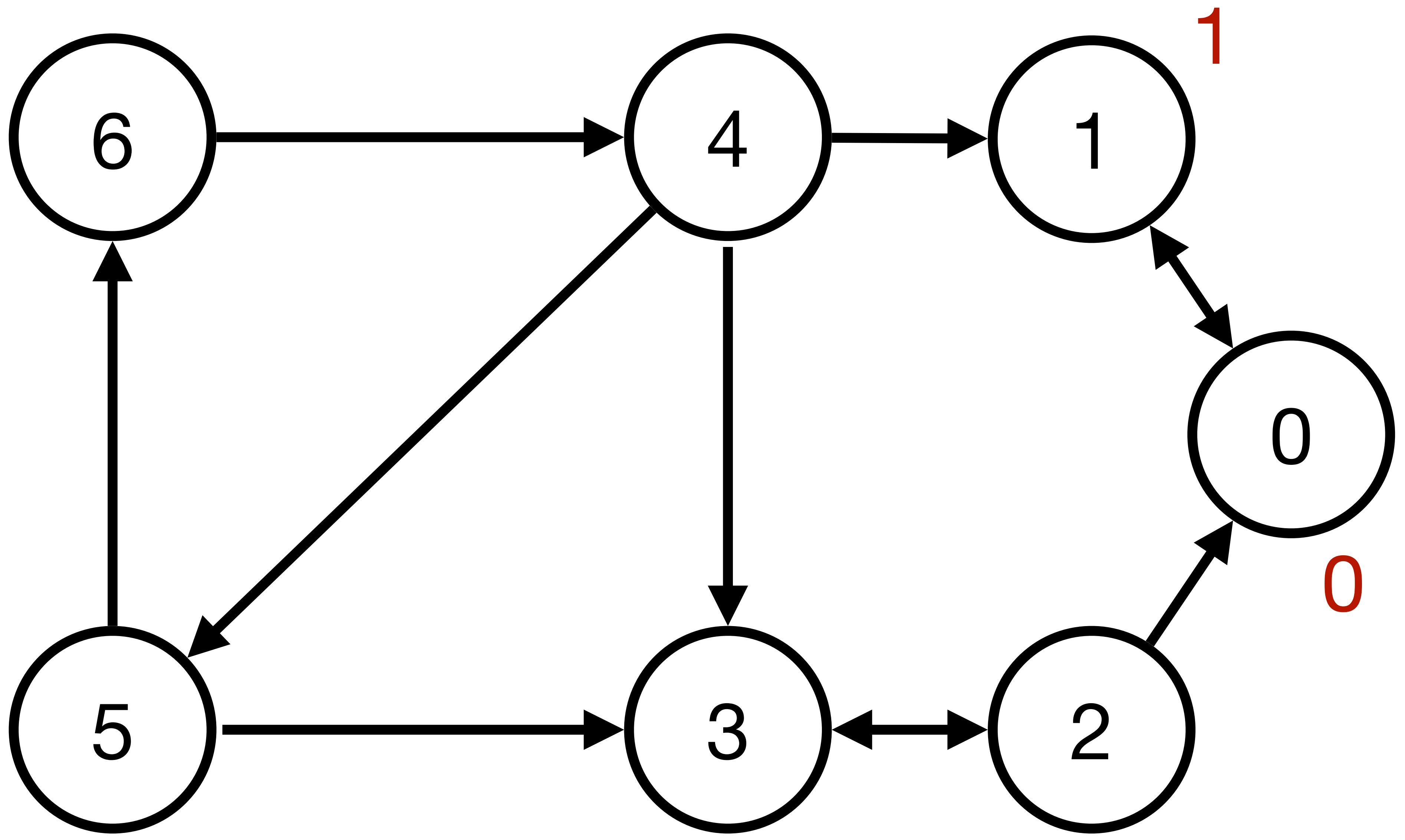


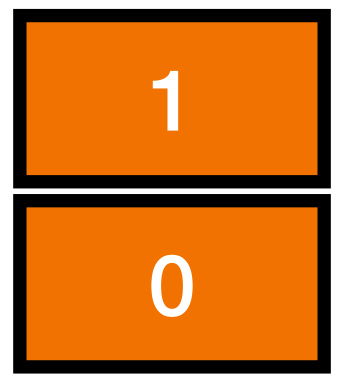
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan’s Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes



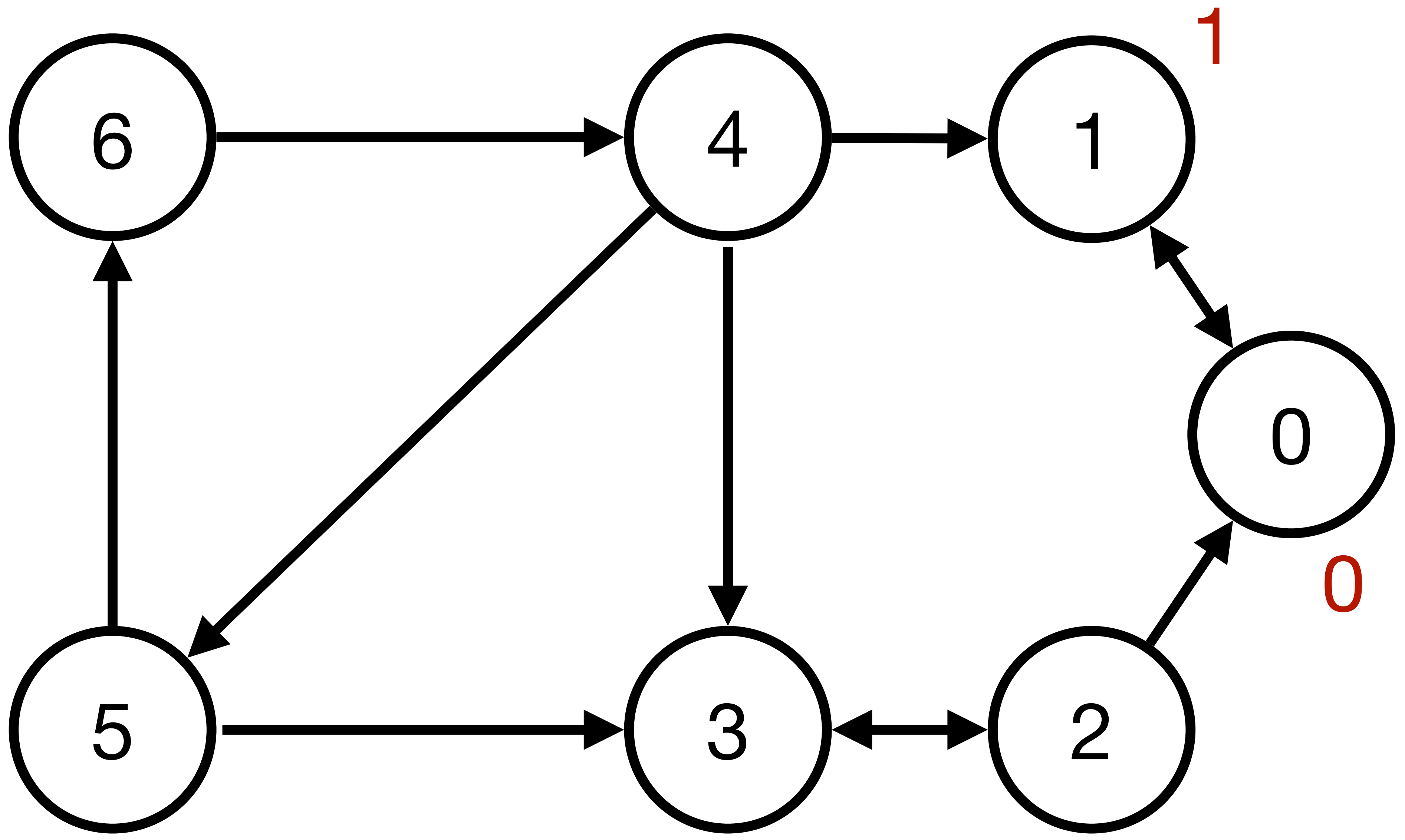


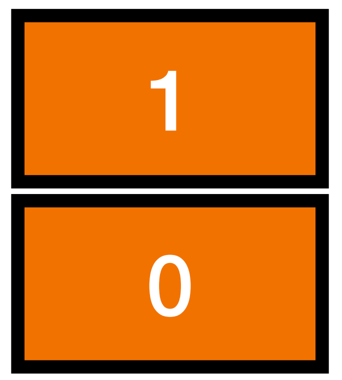
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan’s Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node’s and next node’s LL value



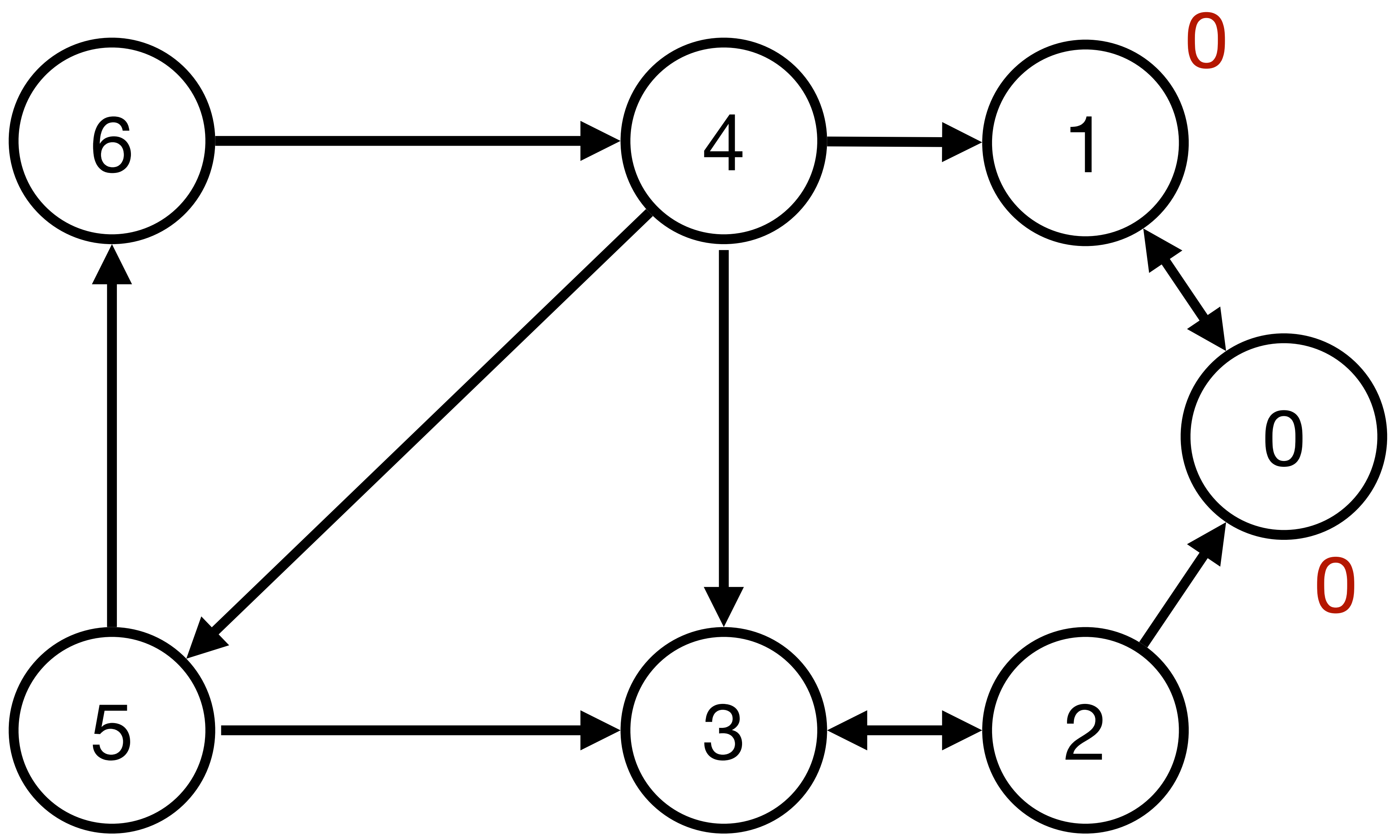


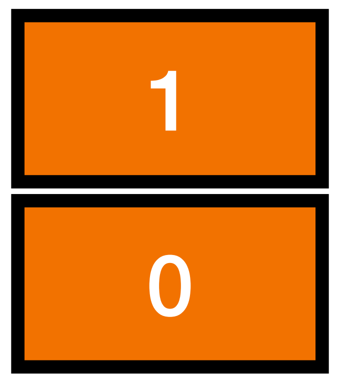
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan’s Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node’s and next node’s LL value



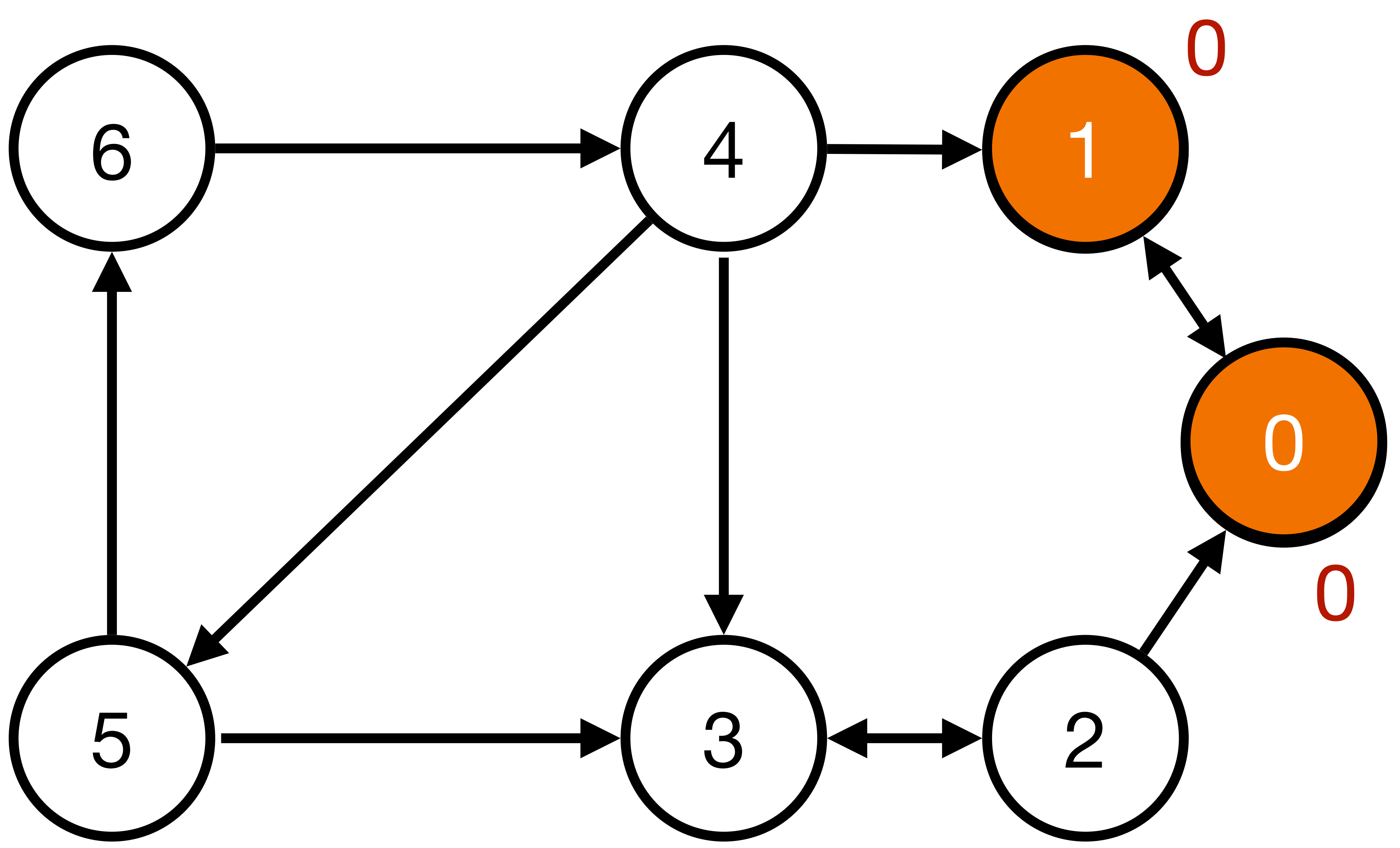


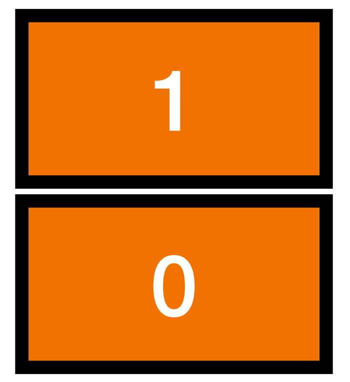
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node's and next node's LL value



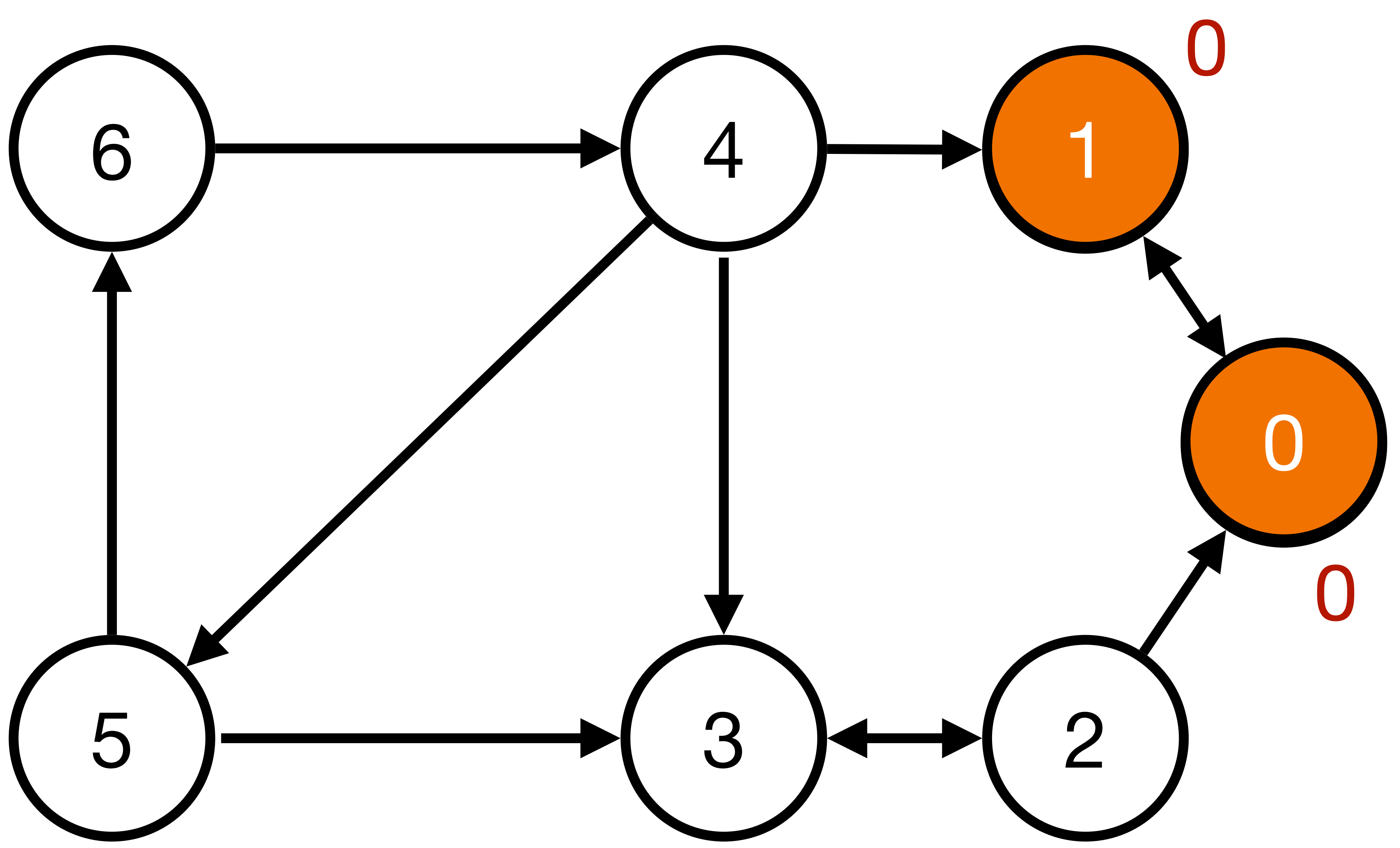


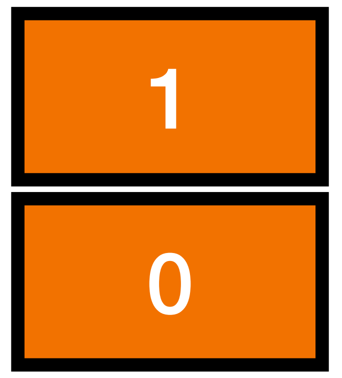
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node's and next node's LL value
    - Allows LL values to propagate through cycles



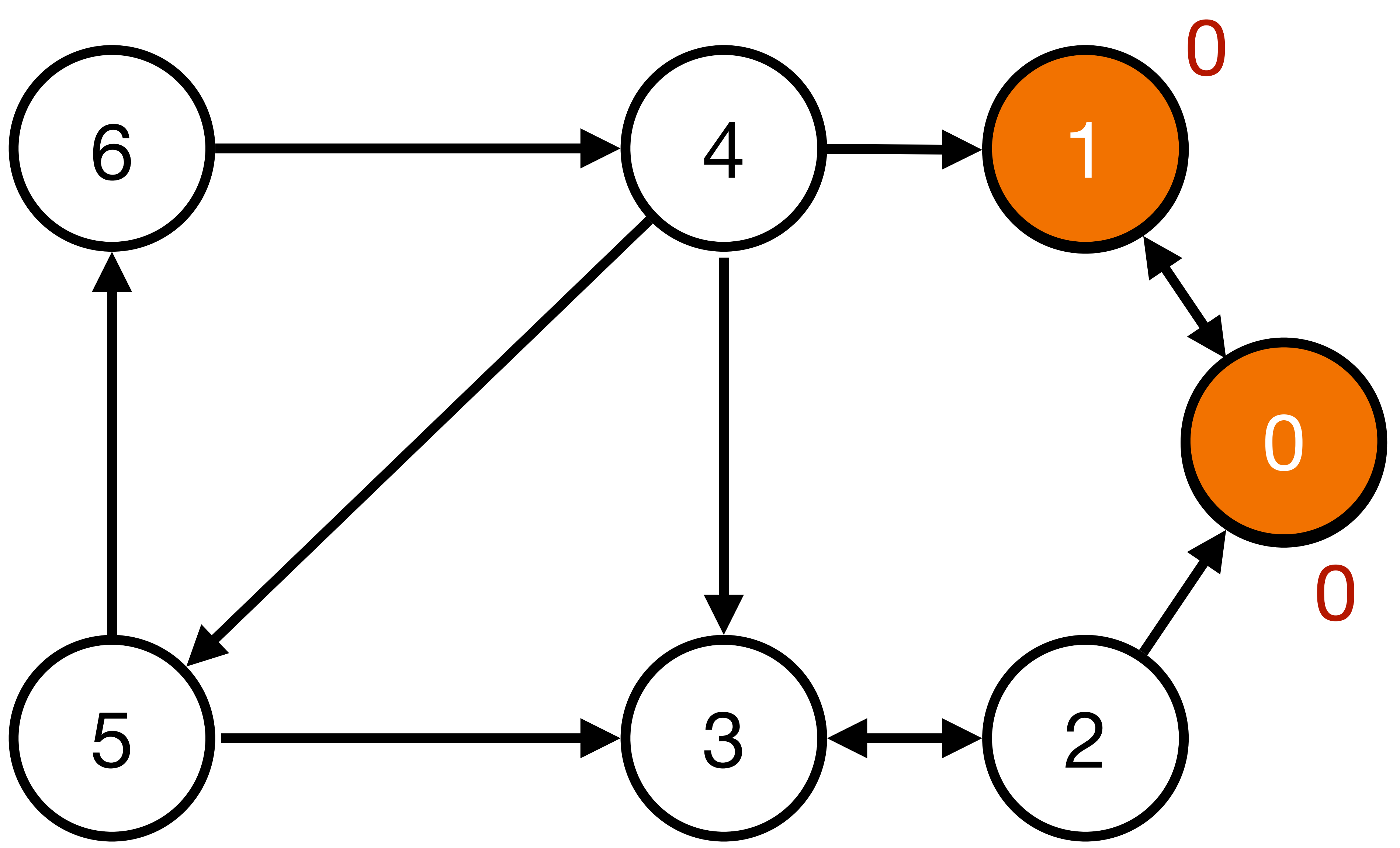


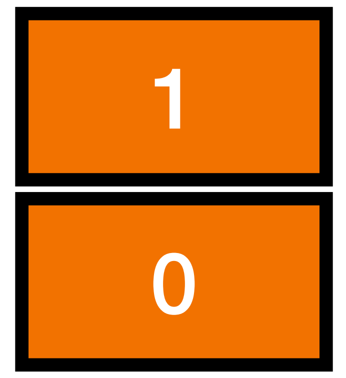
```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node's and next node's LL value
    - Allows LL values to propagate through cycles
  - If all nodes are visited and the current node starts an SCC then pop nodes of the stack until the **current** node



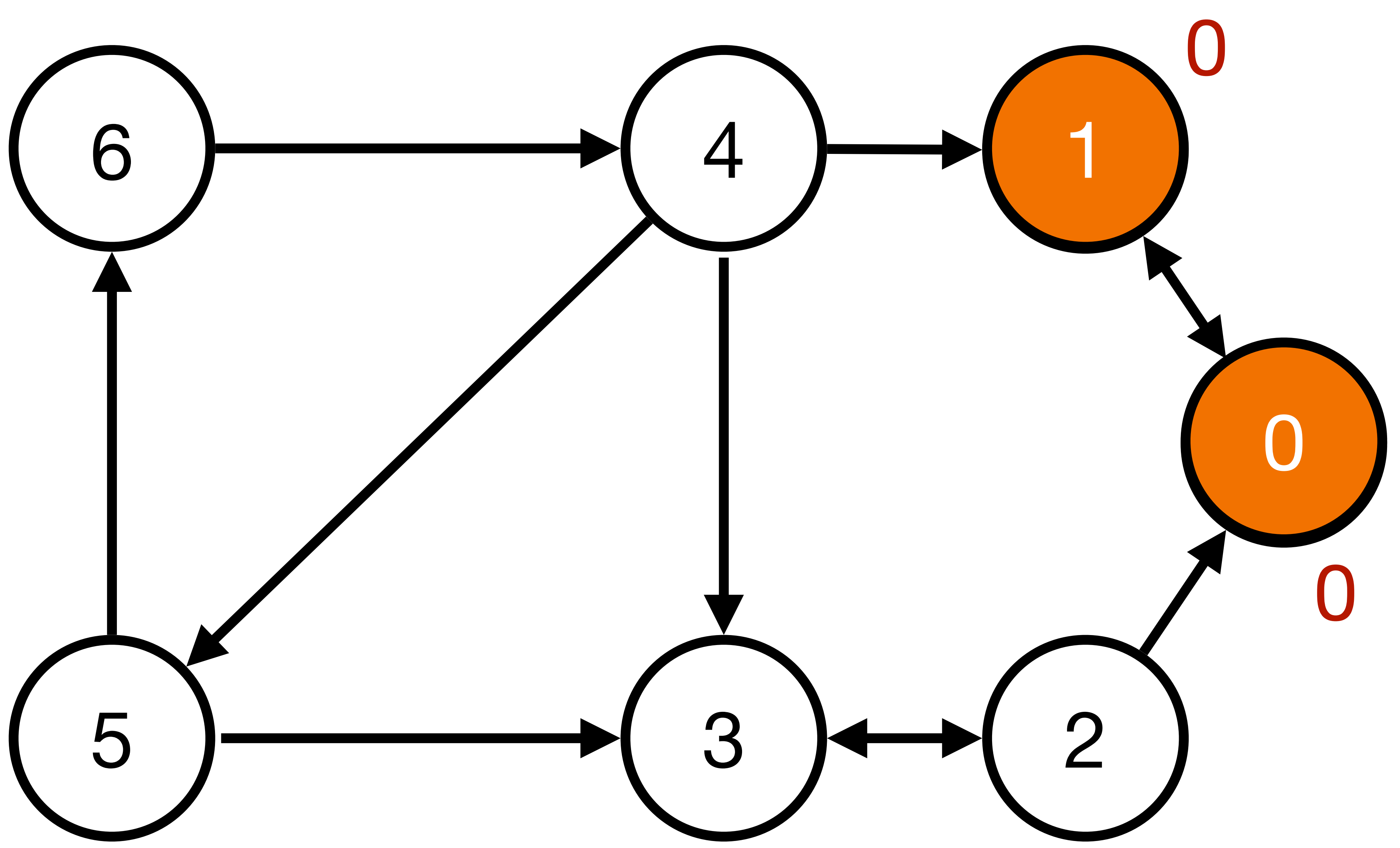


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node's and next node's LL value
    - Allows LL values to propagate through cycles
  - If all nodes are visited and the current node starts an SCC then pop nodes of the stack until the **current** node

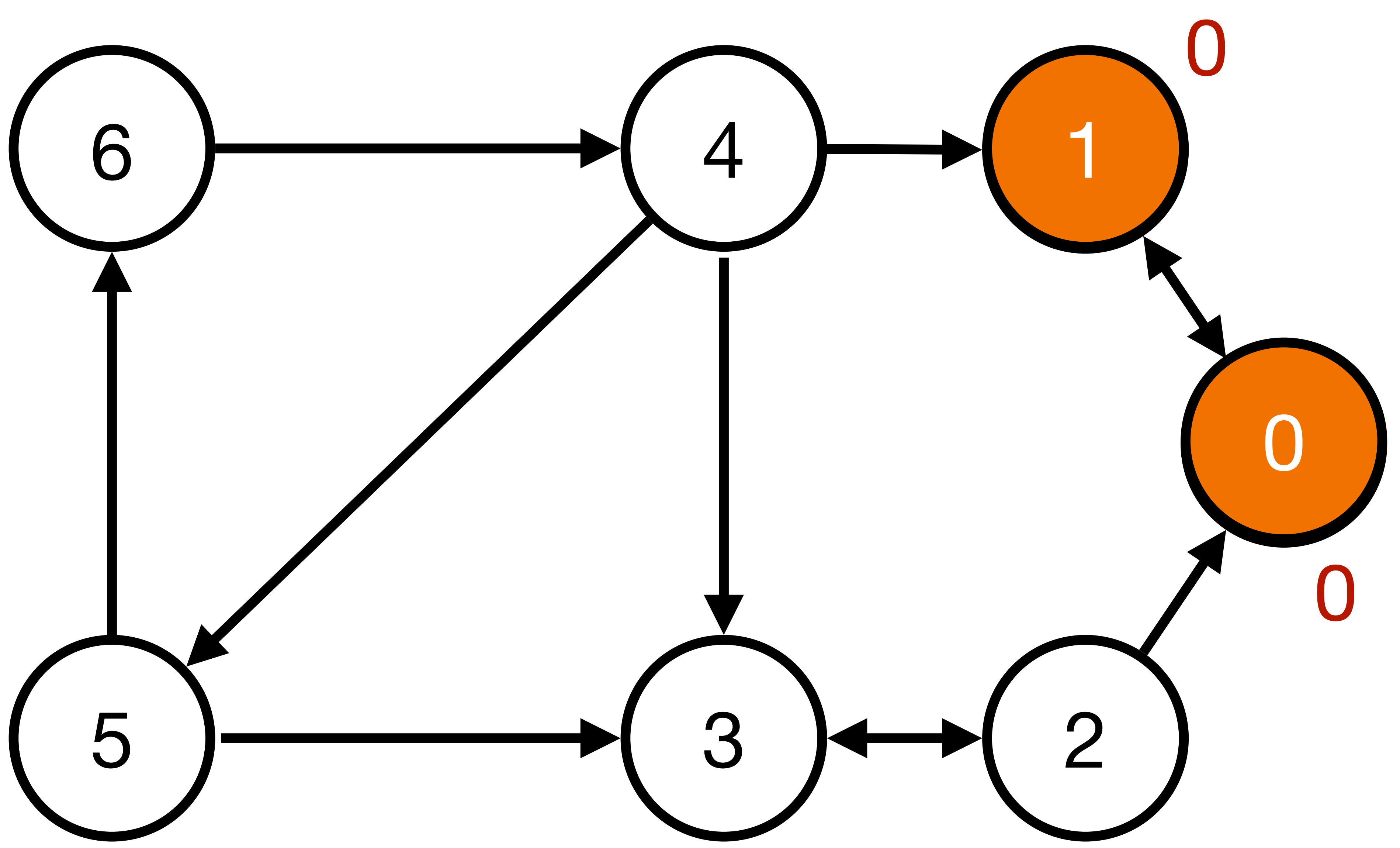


```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node's and next node's LL value
    - Allows LL values to propagate through cycles
  - If all nodes are visited and the current node starts an SCC then pop nodes of the stack until the **current** node



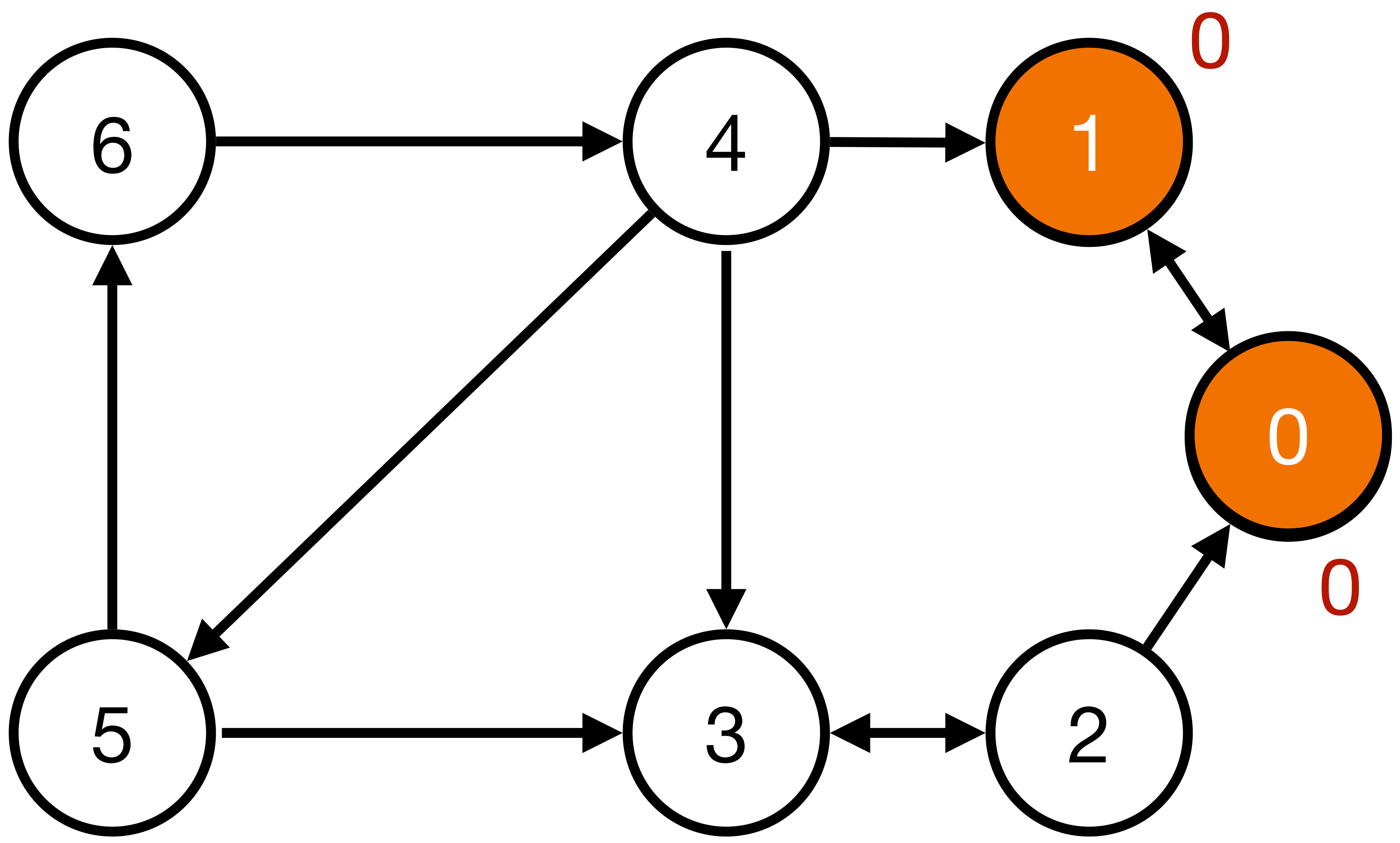
low[1] = min(low[1], ids[0])

```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

Strongly Connected Components

Tarjan's Algorithm

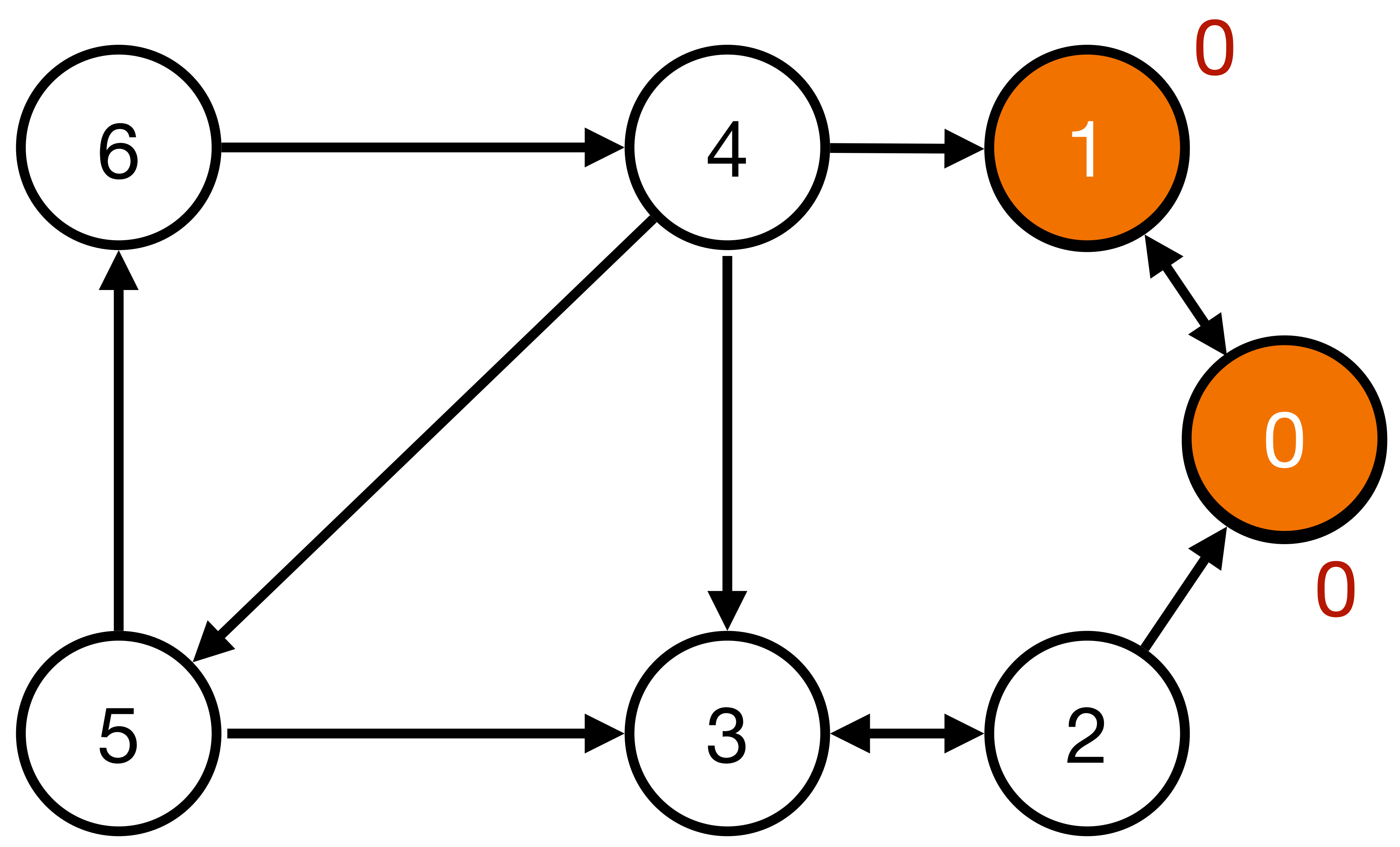
- Algorithm:
  - Start DFS from a node
  - Upon visiting a node assign it a unique integer id and an LL value
    - Mark the node visited and them to the stack of seen nodes
  - On DFS backtrack, if the **next node** is on the stack update the LL value of the **current node** to the minimum of the current node's and next node's LL value
    - Allows LL values to propagate through cycles
  - If all nodes are visited and the current node starts an SCC then pop nodes of the stack until the **current** node



```
for (int w : adj[v]) {
  if (ids[w] == -1) { // not visited yet
    dfs(w);
    low[v] = min(low[v], low[w]);
  } else if (onStack[w]) {
    low[v] = min(low[v], ids[w]);
  }
}
```

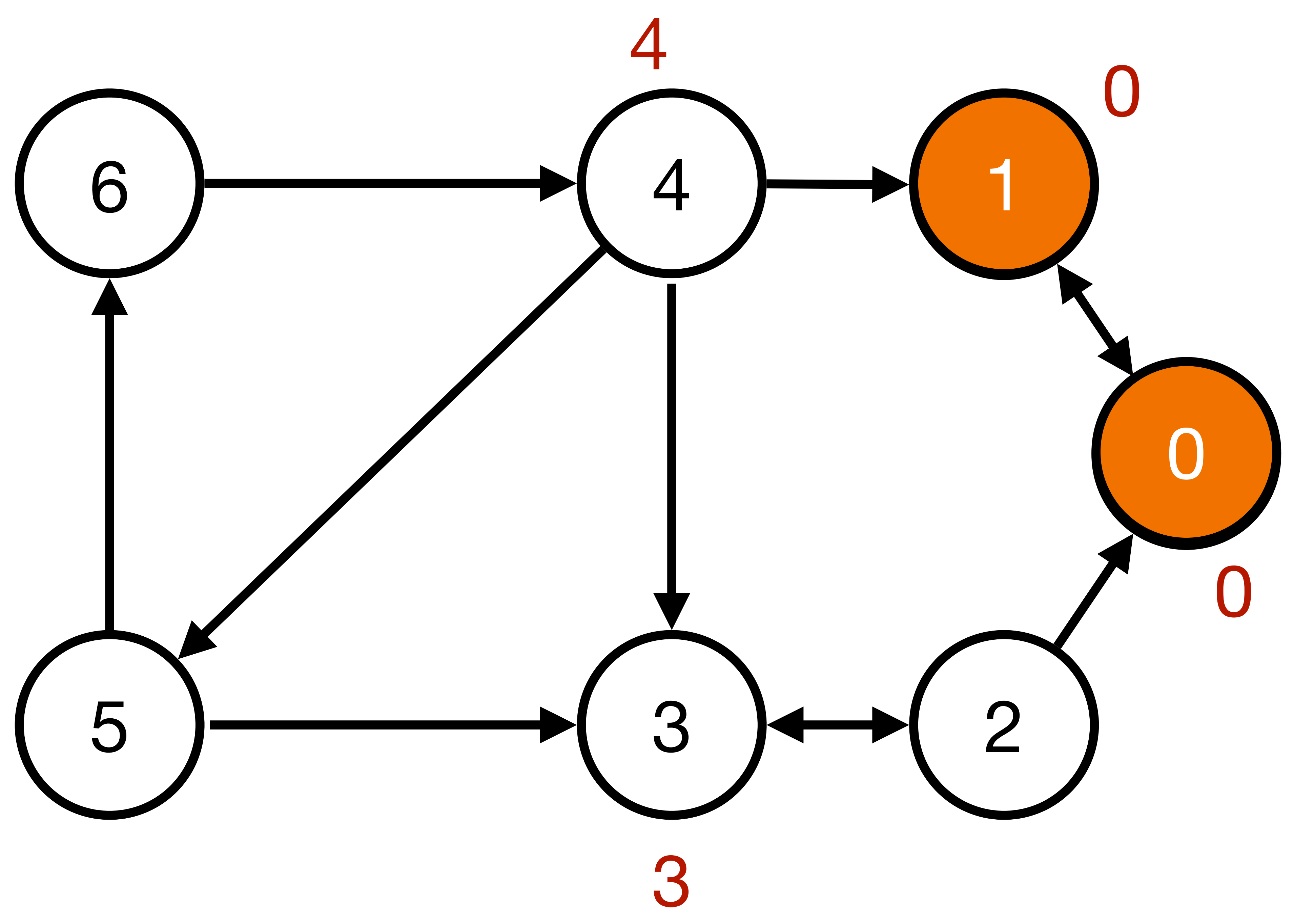
Strongly Connected Components

Tarjan's Algorithm



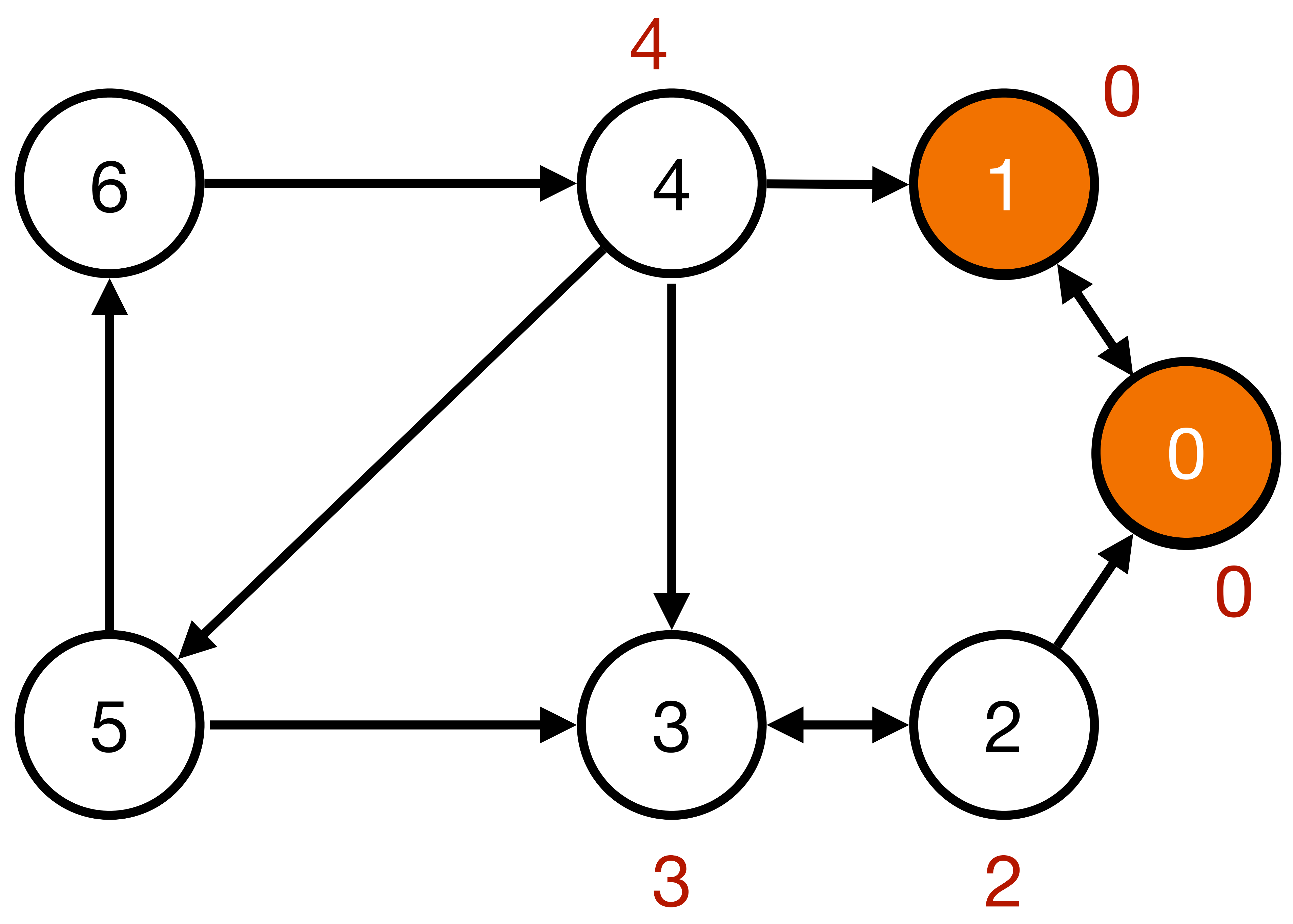
Strongly Connected Components

Tarjan's Algorithm



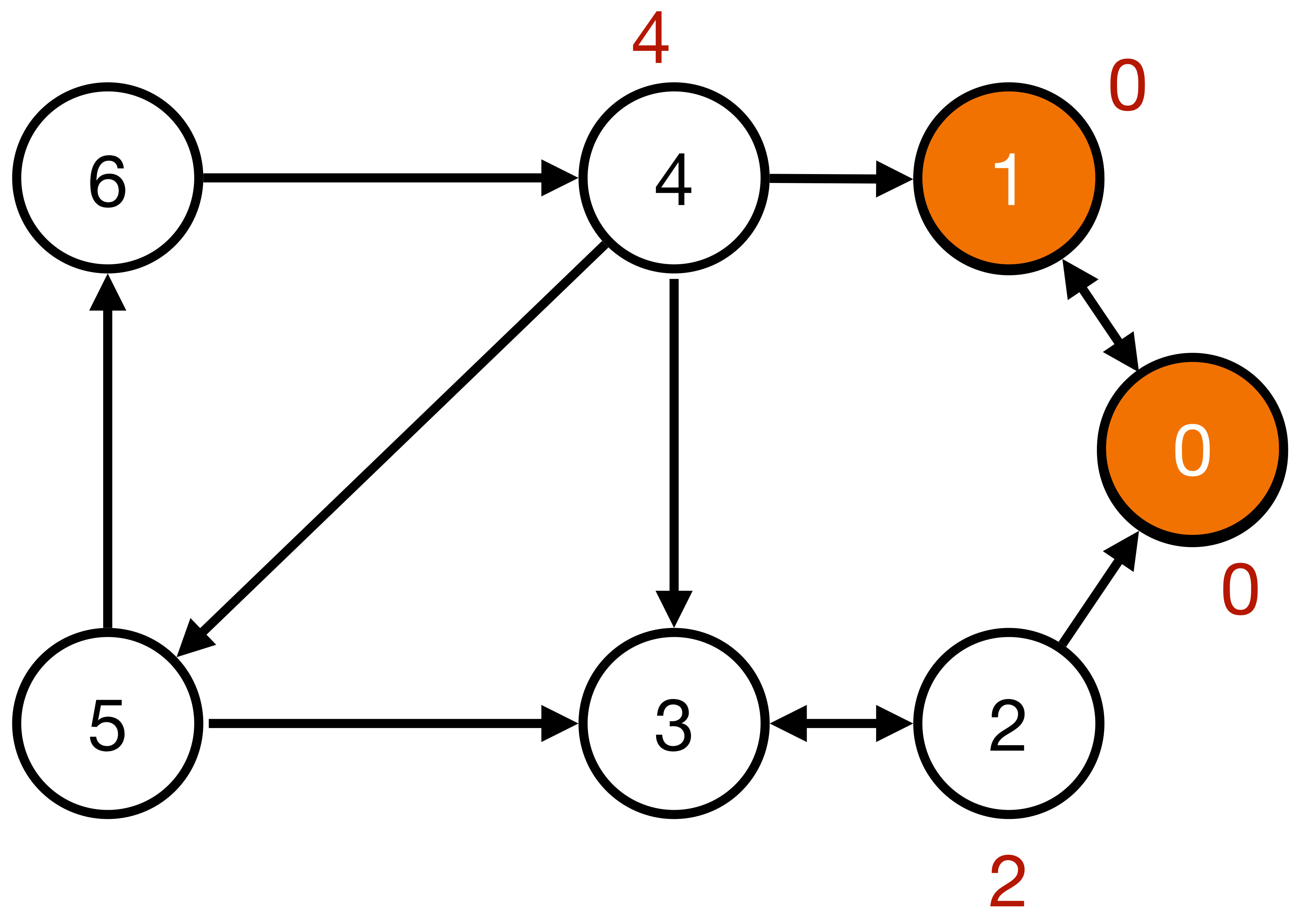
Strongly Connected Components

Tarjan's Algorithm



Strongly Connected Components

Tarjan's Algorithm



Strongly Connected Components

Tarjan’s Algorithm

- **Invariant of Tarjan’s Alg:** A node remains on the stack **iff** there exists a path from it to a node on the stack
    - Prevents the LL values of different SCCs from interfering with each other