Data Structures and Algorithms

Week 10 - Graph traversal, SCCs

Subodh Sharma and Rahul Garg

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

Graph Traversal

DFS For Search

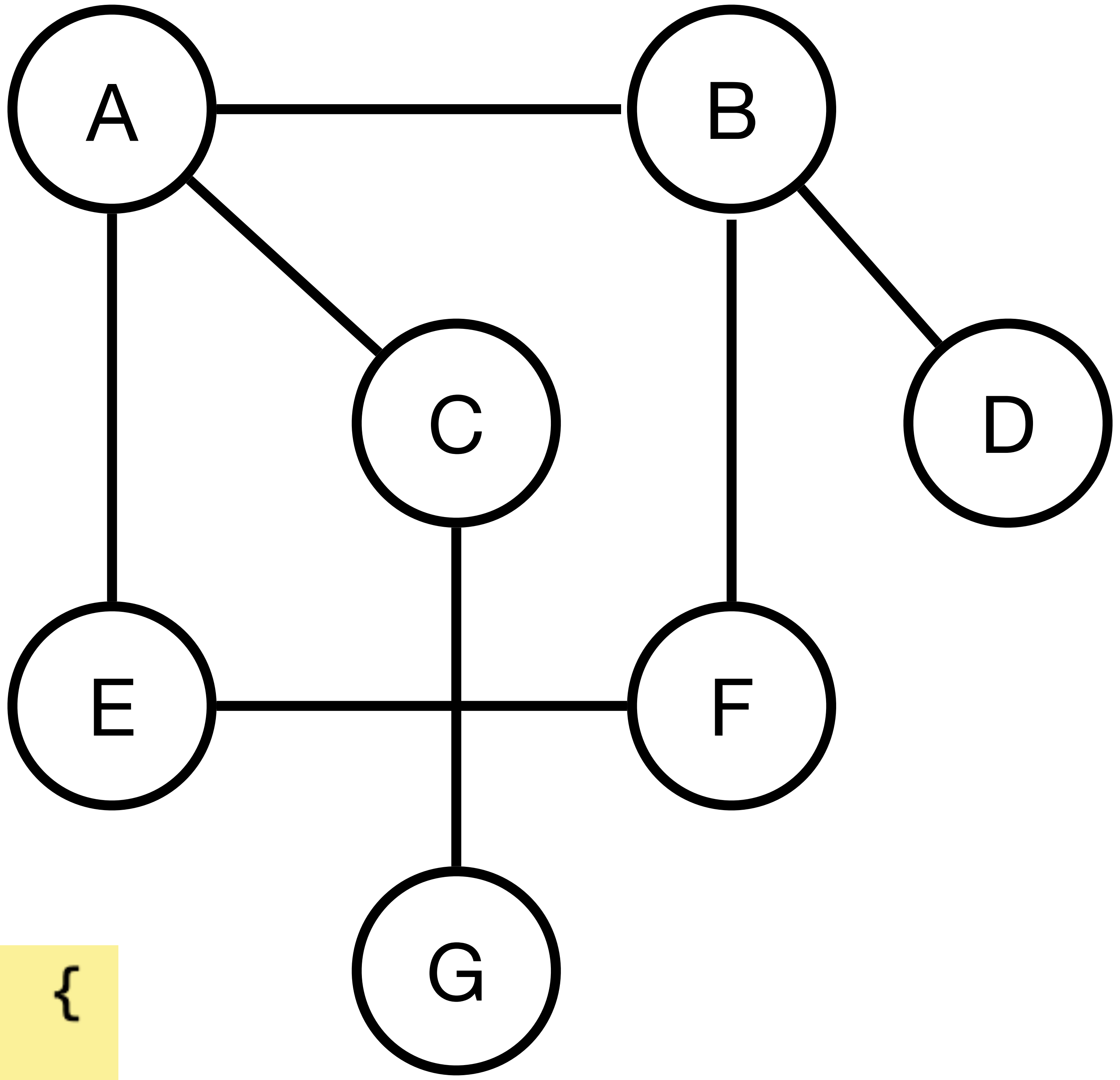
- **Depth-first Search:**
  - Idea is to visit child vertices before visiting sibling vertices of a vertex
  - Algorithm:
    - Start from chosen root vertex and iteratively visit the unvisited adjacent vertex until we cannot continue
    - Backtrack along **previously visited** vertices until unvisited vertices are found to be connected to them

Graph Traversal

DFS For Search

- Start with A without remembering the visited nodes
  - **A -> B -> D -> F -> E -> A … and the cycle continues w/o ever visiting C or G**
- **Recursive Implementation**

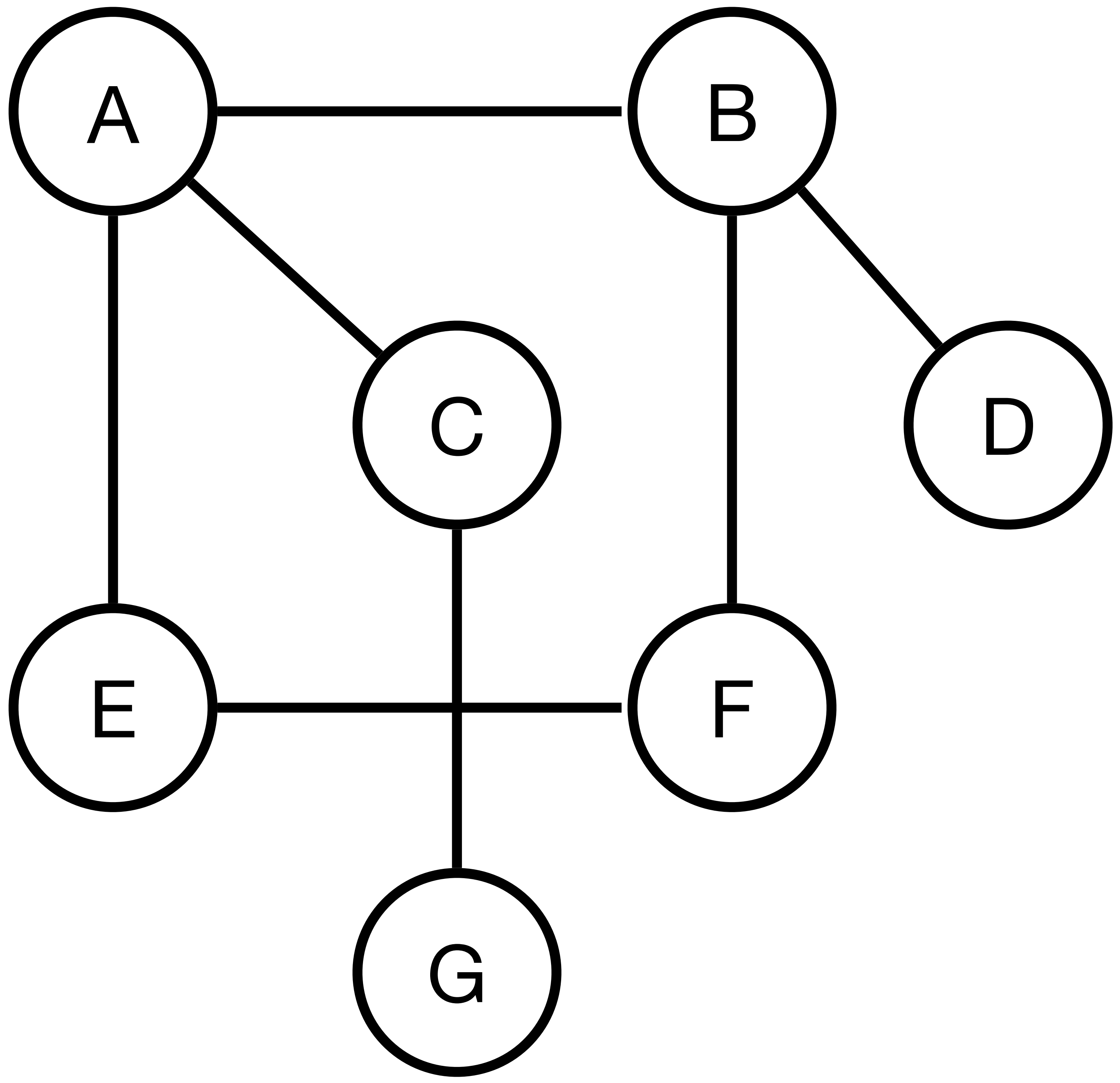
```
void DFSUtil(int vertex, std::unordered_set<int>& visited) {
  std::cout << vertex << " ";
  visited.insert(vertex);

  for (int neighbor : adjList[vertex]) {
    if (visited.find(neighbor) == visited.end()) {
      DFSUtil(neighbor, visited);
    }
  }
```



Graph Traversal

DFS For Search

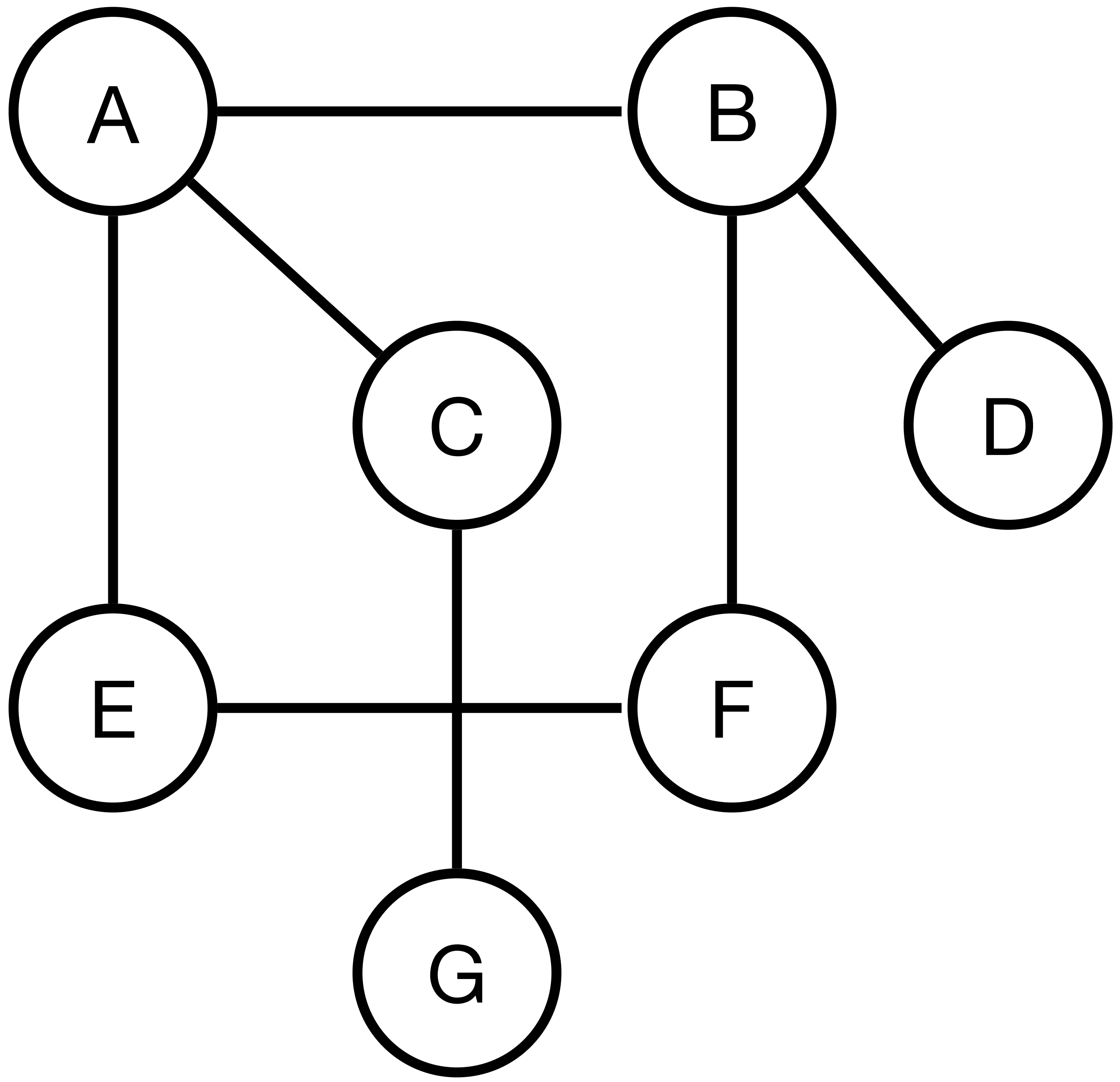
- Iterative **Implementation**
  - Put the currentVertex in stack
  - While stack is not empty
    - Pop the top
    - If top is not visited, add it to the set of visited nodes
      - Add **all** the neighbours of top to the stack.



Graph Traversal

BFS For Search

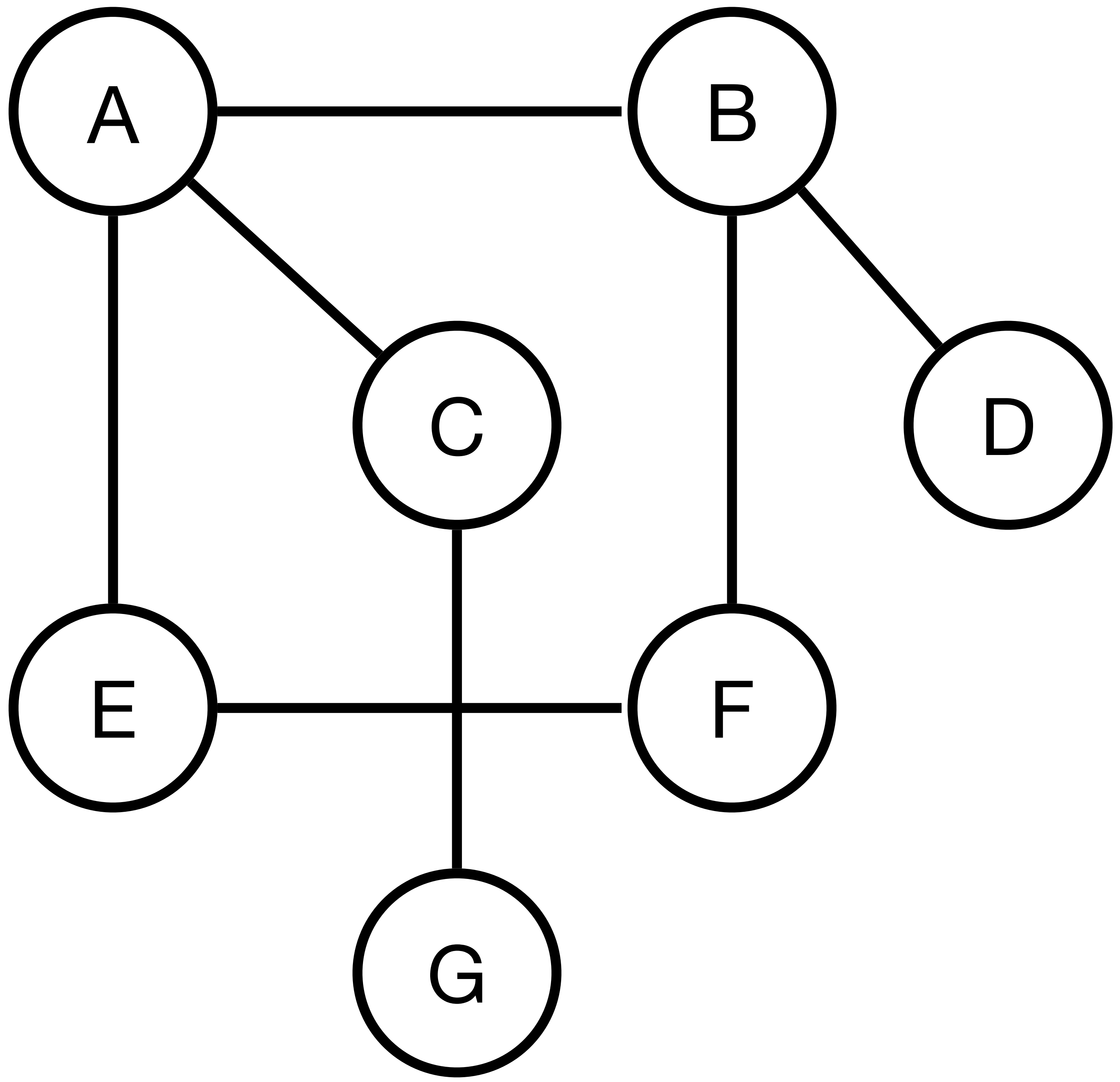
- As the name suggests, explores all the nodes at the present level before moving to the nodes of the next level
- **A -> B, E, C -> F,D; G**
- **Algorithm steps:**
  - Enqueue the root (or starting vertex)
  - While the queue is not empty:
    - Dequeue the vertex from front; add to the visited sets
    - Add all unvisited neighbours of the vertex to the queue



Graph Traversal

BFS For Search

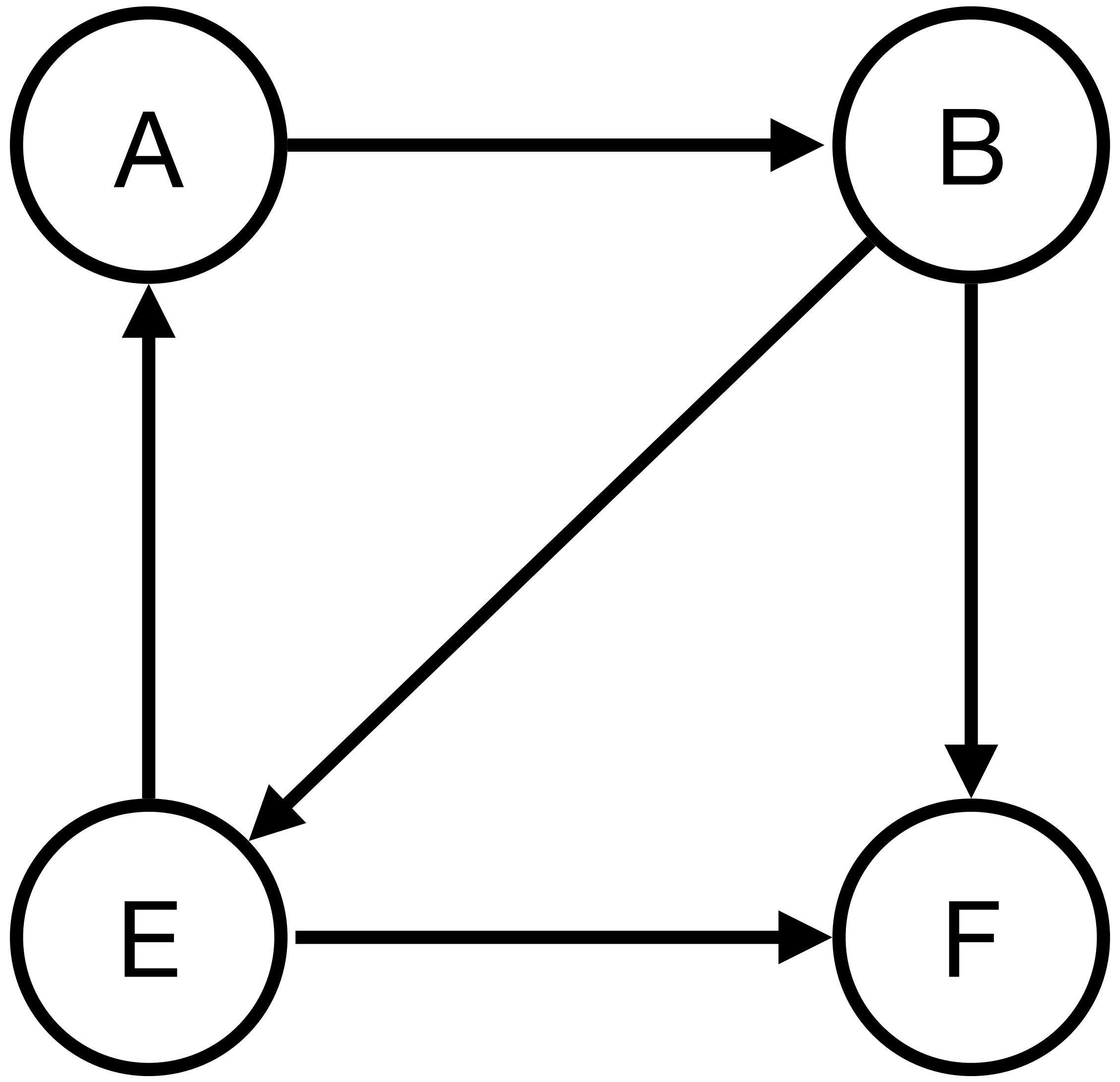
- A -> B, E, C -> F,D; G
- Show the code!



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

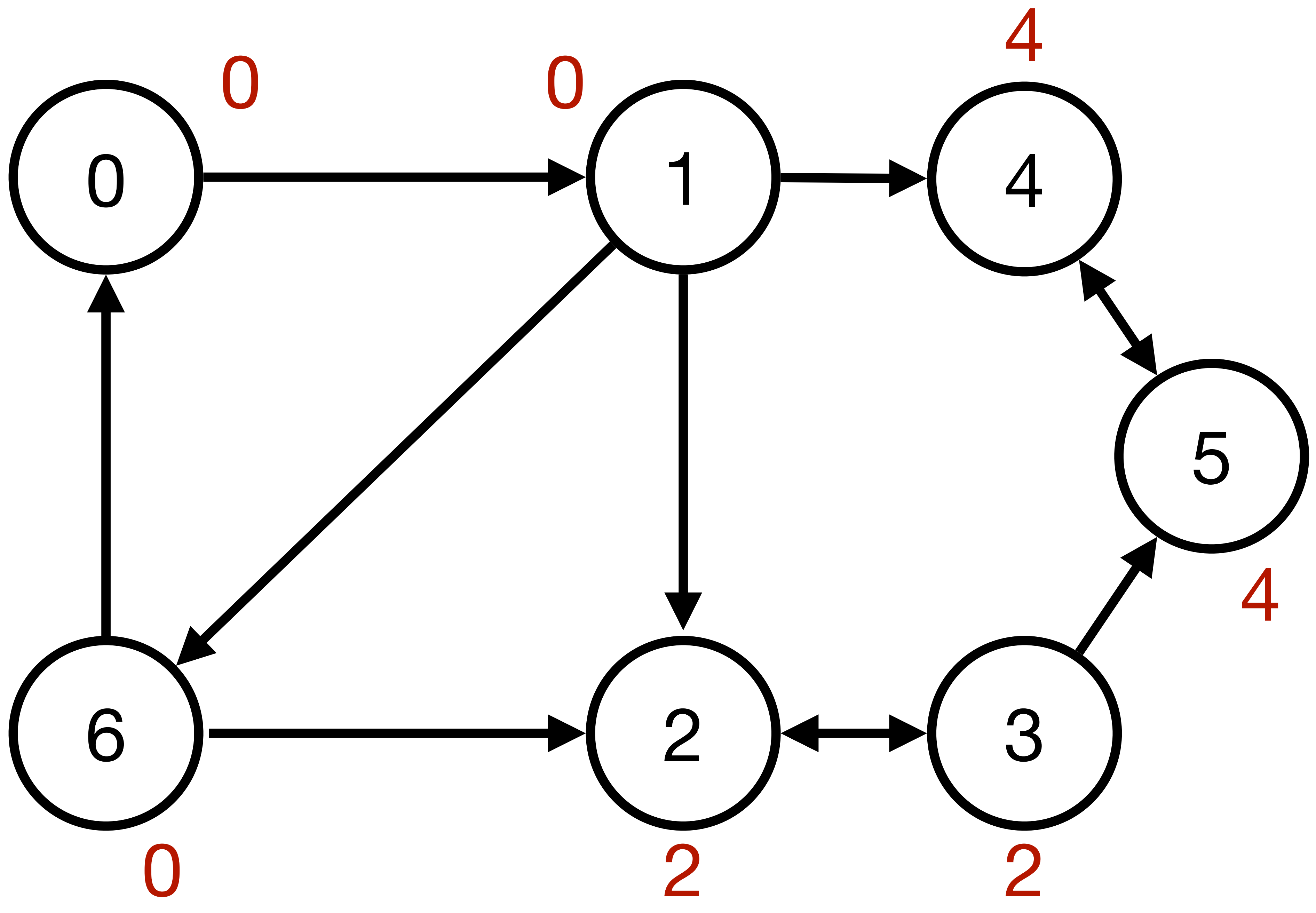


- SCC: ({A,B,E}. {() .. })
- Use of DFS to find SCCs — Robert Tarjan 1972 (also discovered Splay and Fibonacci Heaps)

Strongly Connected Components

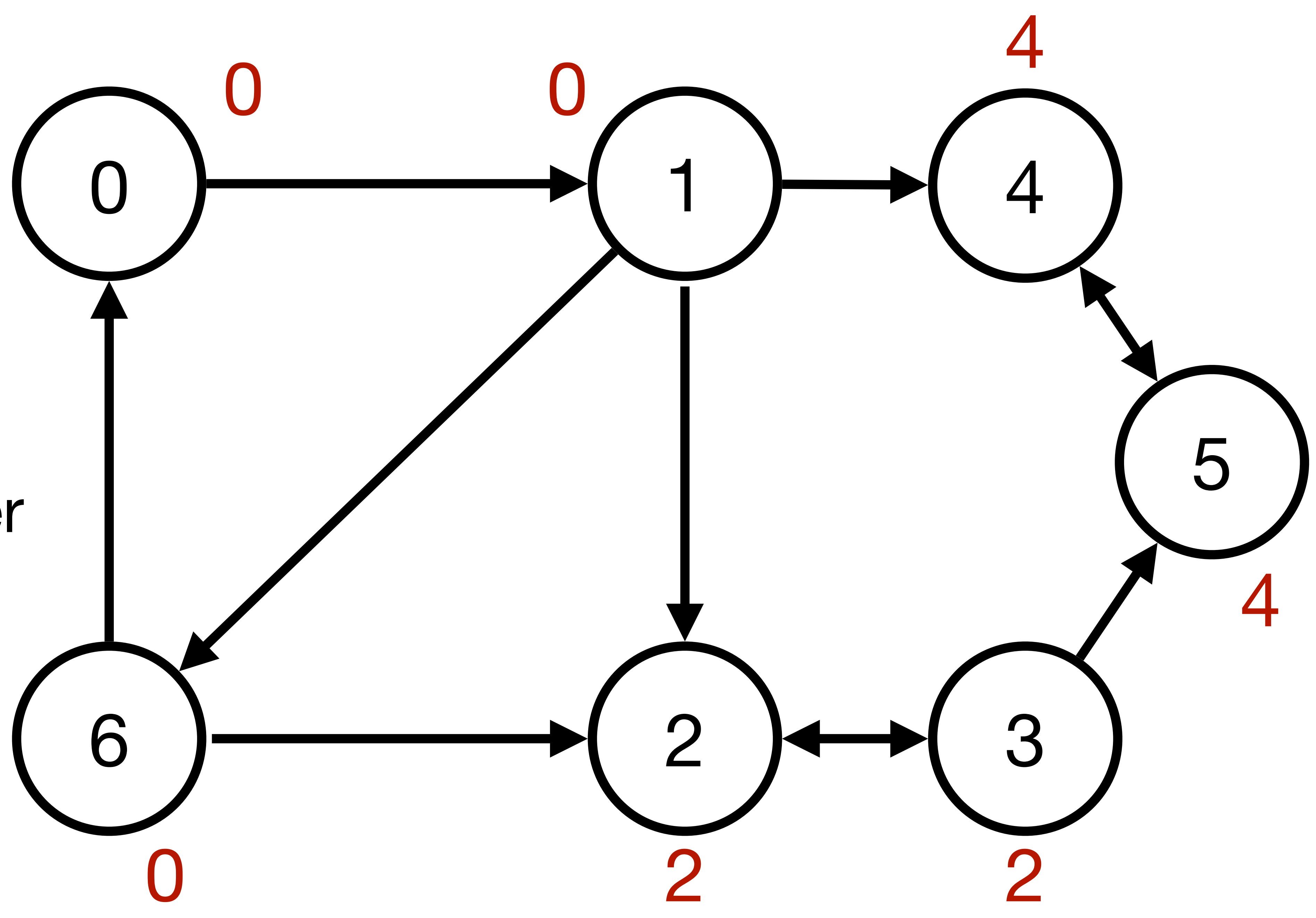
Tarjan's Algorithm

- **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**
- **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



- **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
- 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 callback, if the prev node is on the stack update the LL value of it to the last 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

Tarjan’s SCC Code