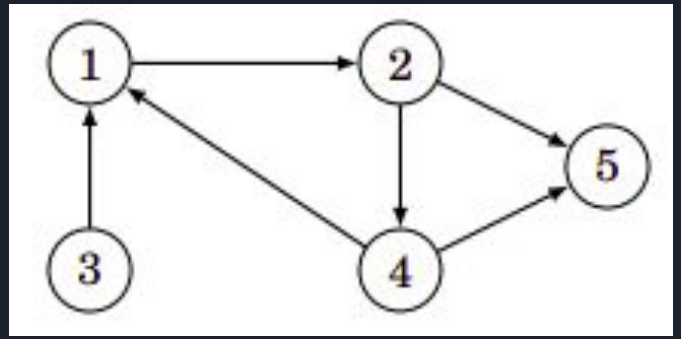
Graph Theory

Part - 03

Directed Graph

A graph is directed if the edges can be traversed in one direction only. For example, the following graph is directed:



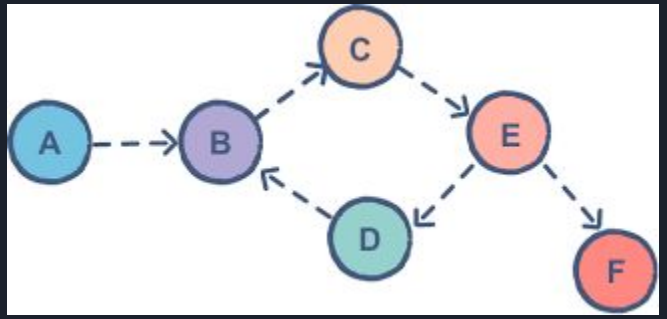
The above graph contains a path $3 \rightarrow 1 \rightarrow 2 \rightarrow 5$ from node 3 to node 5, but there is no path from node 5 to node 3.

Directed Graph

In a directed graph:

The **indegree** of a vertex V is the number of edges coming into vertex V.

The **outdegree** of a vertex V is the number of edges going out from vertex V.

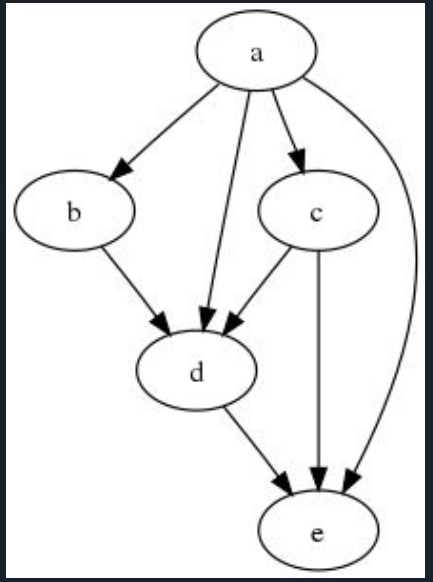


Here, the indegree of node B is 2 and outdegree of node B is 1.

Directed Acyclic Graph

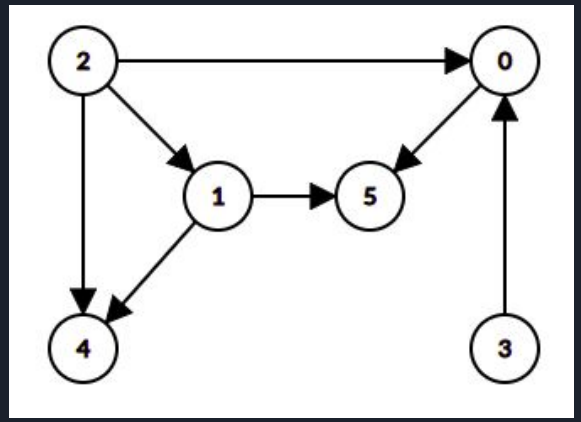
Acyclic graphs: A graph is acyclic if there are no cycles in the graph, so there is no path from any node to itself.

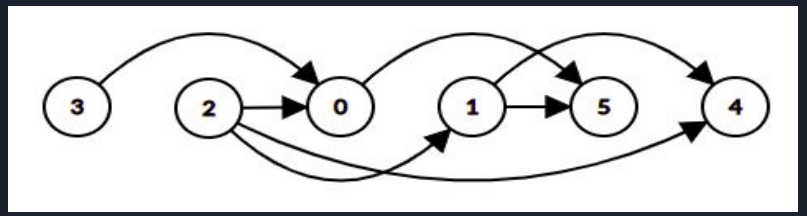
DAG: A directed acyclic graph (DAG) is simply a directed graph which contains no cycles. That is, it consists of vertices and edges, with each edge directed from one vertex to another, such that following those directions will never form a closed loop.

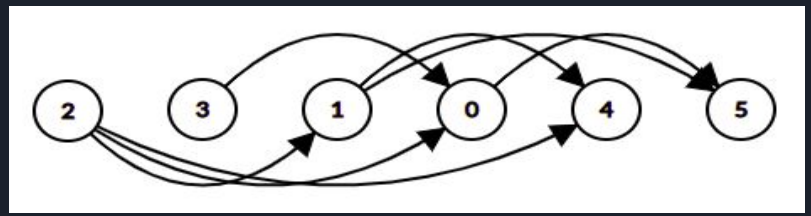


Topological Sort

A topological sort is an ordering of the nodes of a directed graph such that if there is a path from node a to node b, then node a appears before node b in the ordering.







Topological sort of the graph is [3, 2, 0, 1, 5, 4] or [2, 3, 1, 0, 4, 5]

Topological Sort

An acyclic graph always has a topological sort. If the graph contains a cycle, it is not possible to form a topological sort, because no node of the cycle can appear before the other nodes of the cycle in the ordering

Topological Sort

In a topological sorting of a DAG, which node comes first?

- The first node in a topological sorting should have the indegree equal to 0.

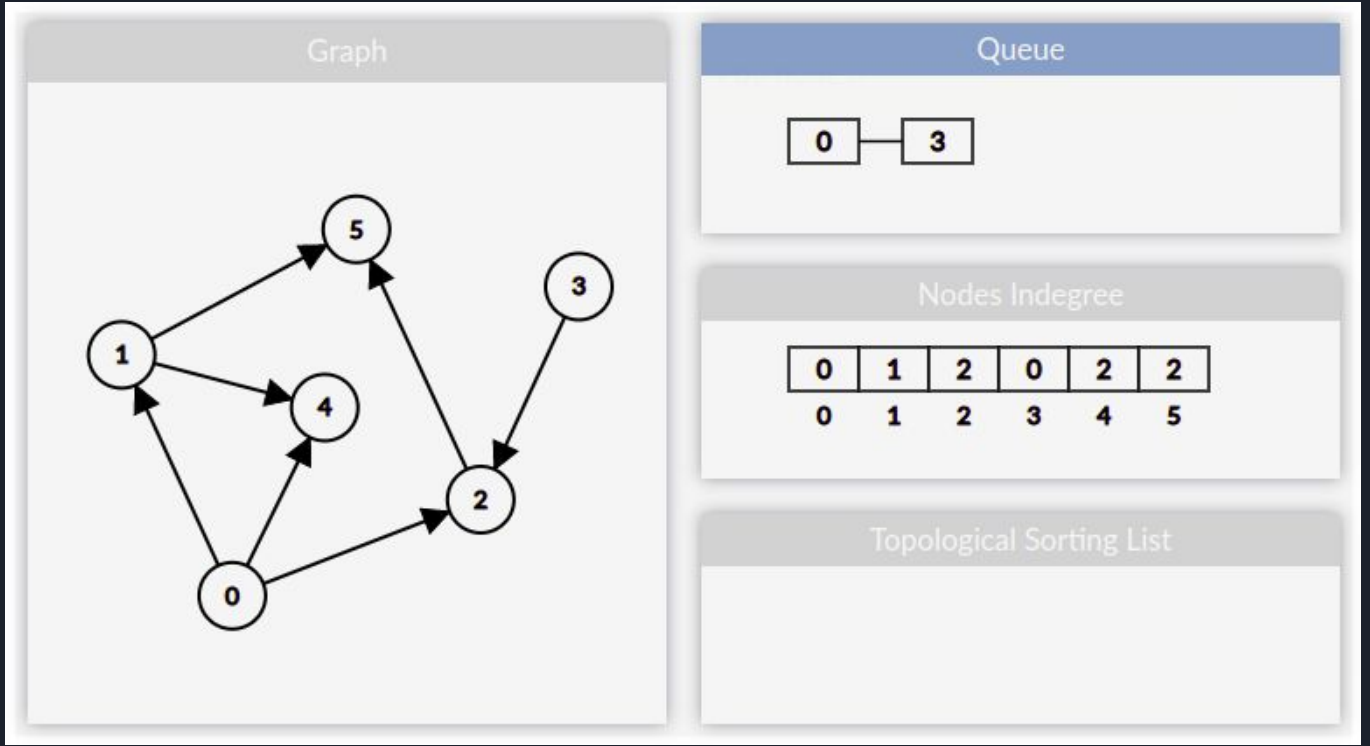
The topological sorting algorithm:

1. Find a node with indegree equal to 0 and position it the first in the topological sorting.
2. Remove that node from the graph.
3. The remaining part of the graph is also a DAG, so go back to step 1.

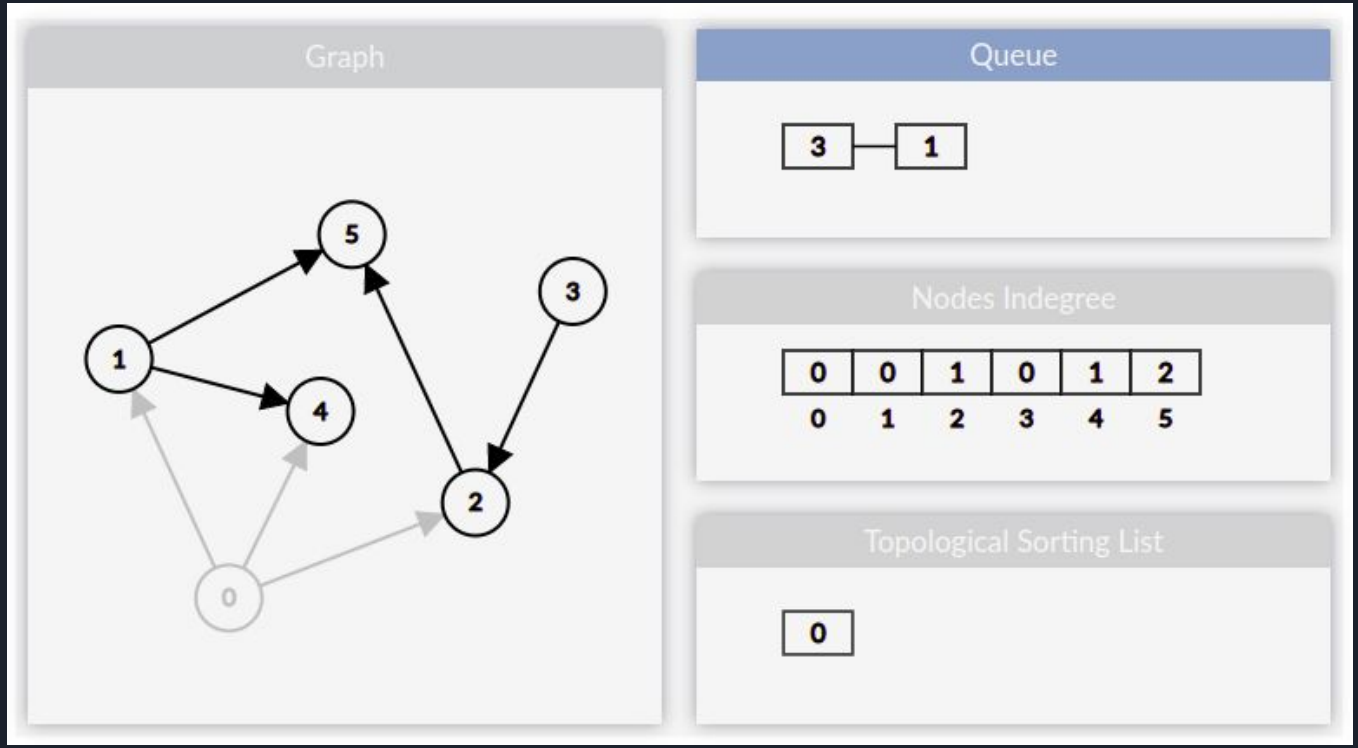
Topological Sort

We maintain a queue like bfs and an array which contains the indegree of each node. The nodes with indegree 0 is added into the queue. We take the nodes one by one from the queue and add it to the sorted list. When a node is added to the sorted list, the indegree of each adjacent node gets decreased by 1.

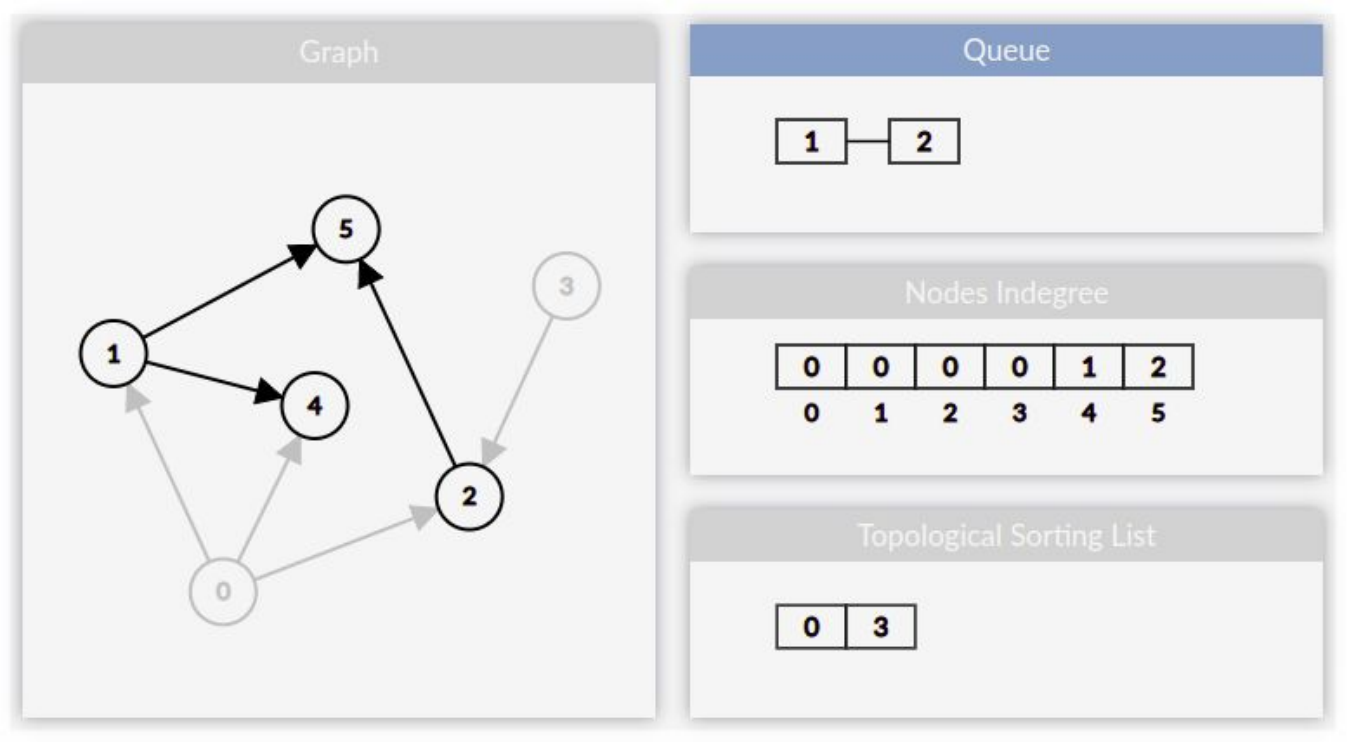
Topological Sort



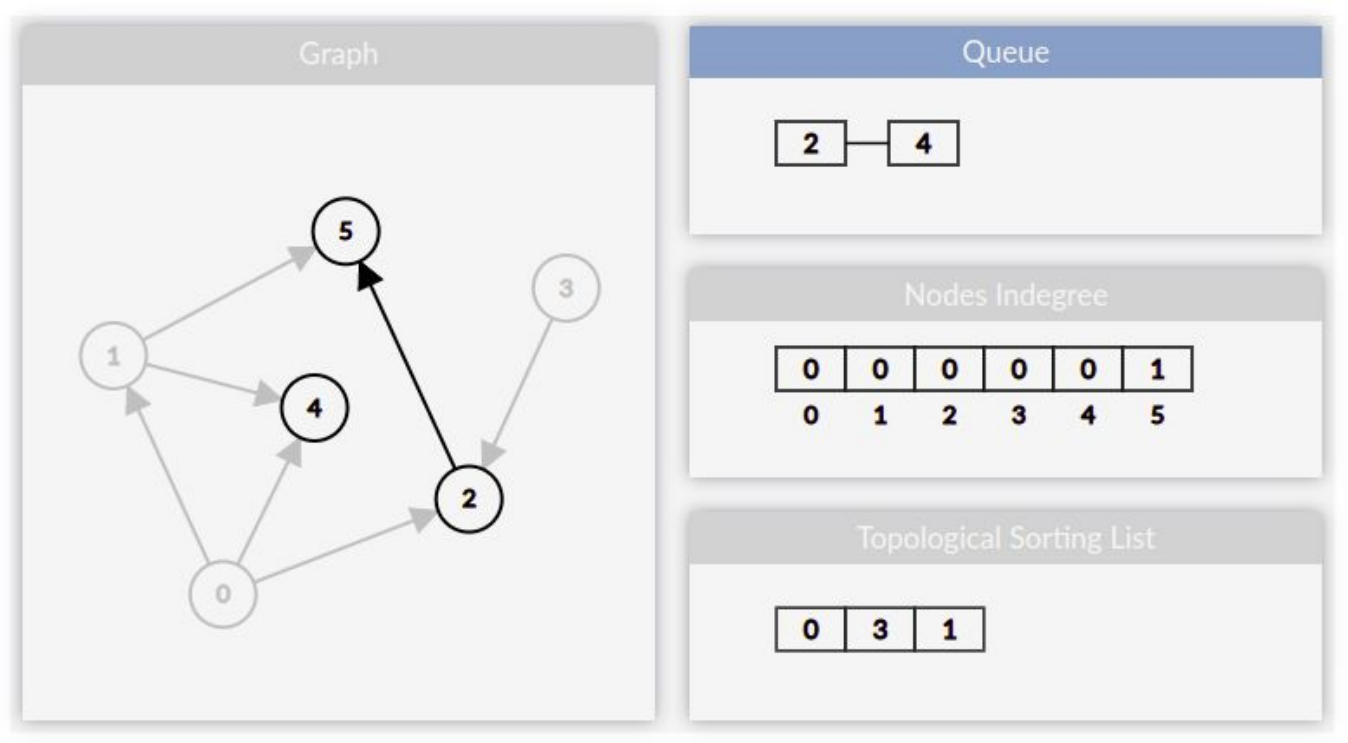
Topological Sort



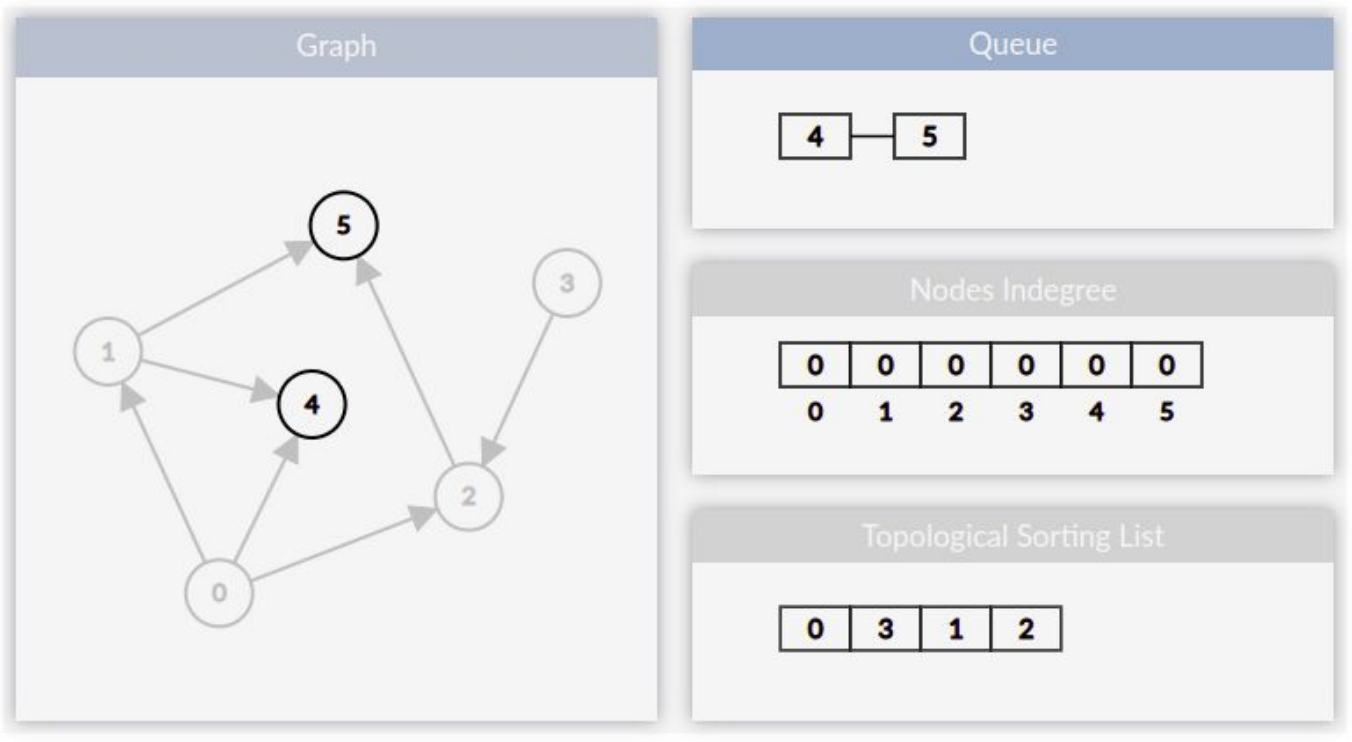
Topological Sort



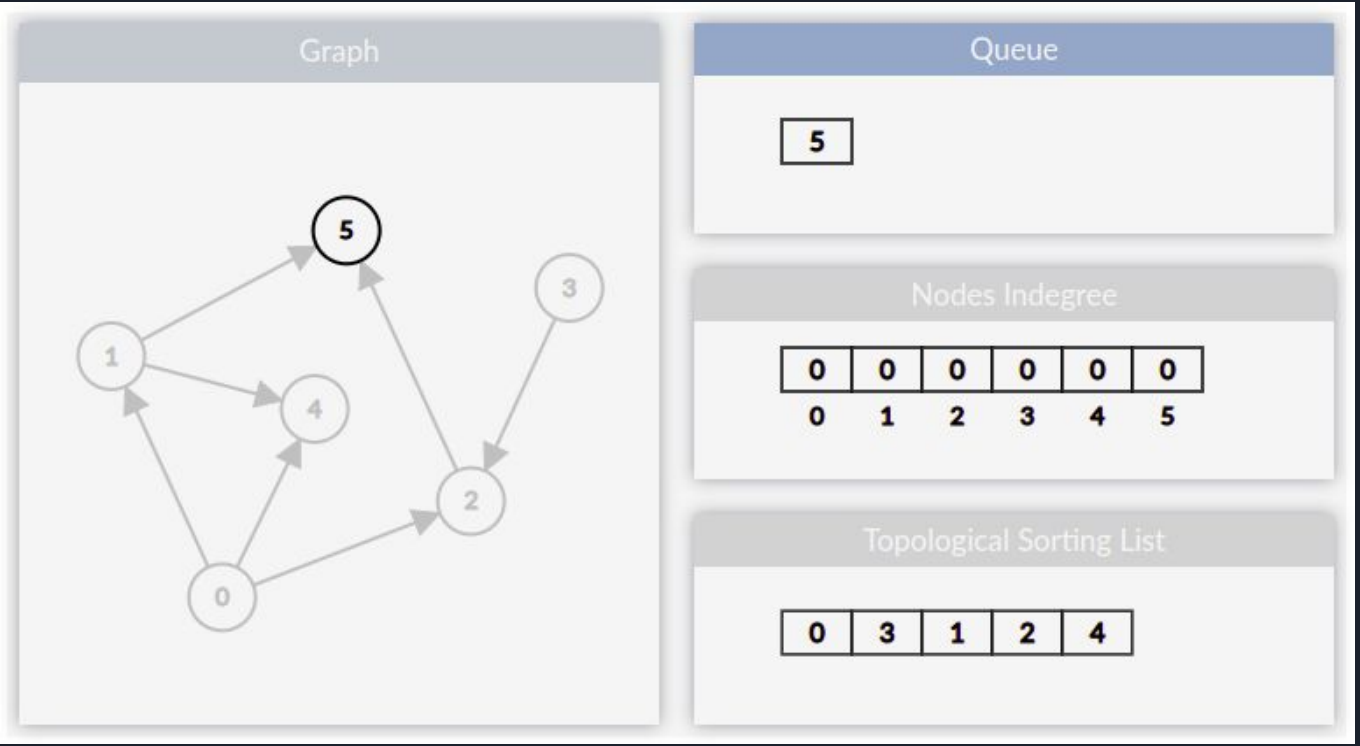
Topological Sort



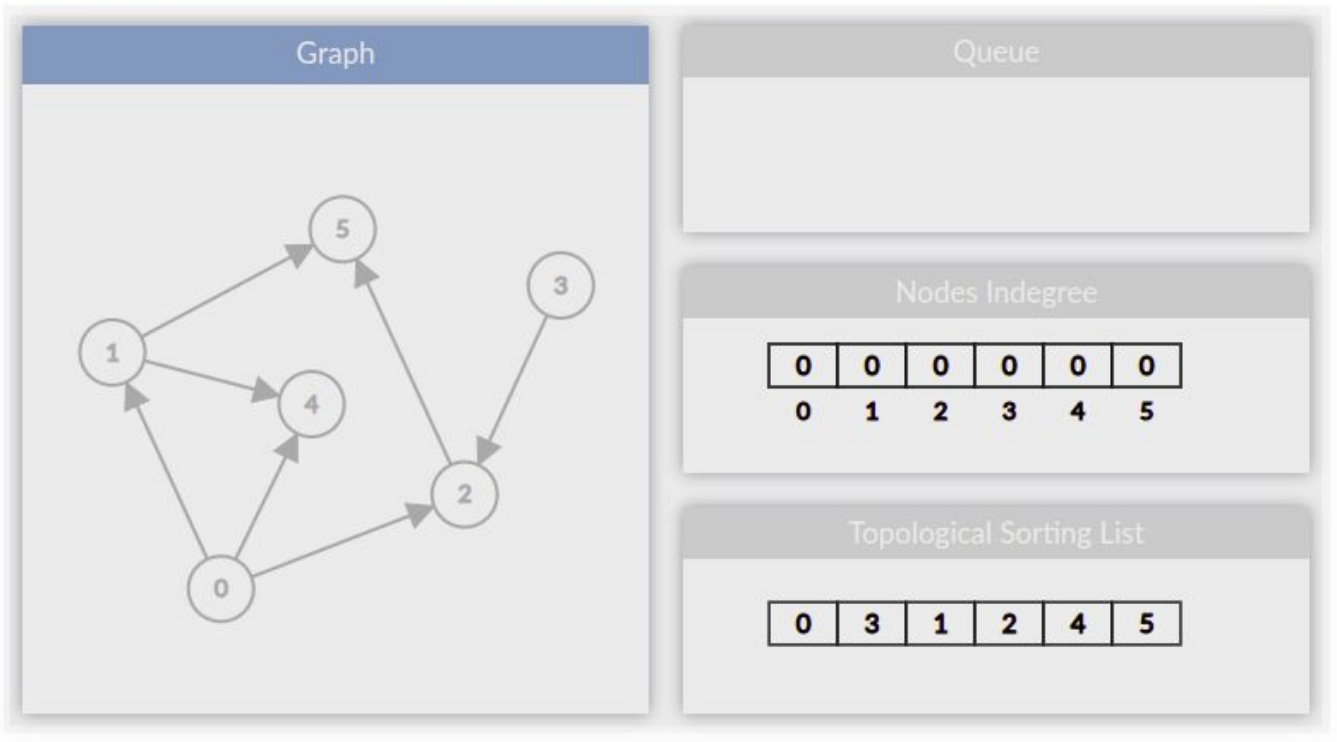
Topological Sort



Topological Sort



Topological Sort



Topological Sort (BFS Implementation)

```cpp
vector<int> adj[N];
int indegree[N];
vector<int> ans;

void topological_sort(){
    queue<int> q;
    for(int i = 1; i <= n; i++)
        if(indegree[i]==0) q.push(i);

    while(!q.empty()){
        int u = q.front();
        q.pop();
        ans.push_back(u);
        for(int v: adj[u]){
            indegree[v]--;
            if(indegree[v]==0) q.push(v);
        }
    }
}
```

Topological Sort (DFS Implementation)

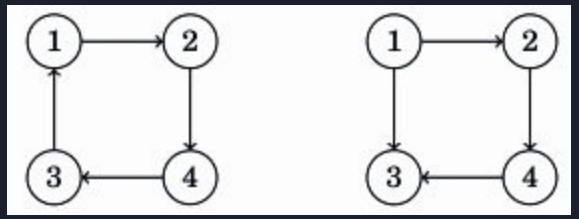
```cpp
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<int> visited, ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

Strongly Connected Component

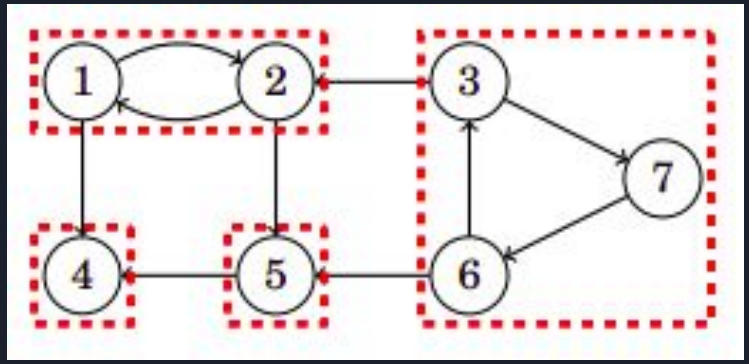
A graph is strongly connected if there is a path from any node to all other nodes in the graph.



The left graph is strongly connected but the right graph is not.

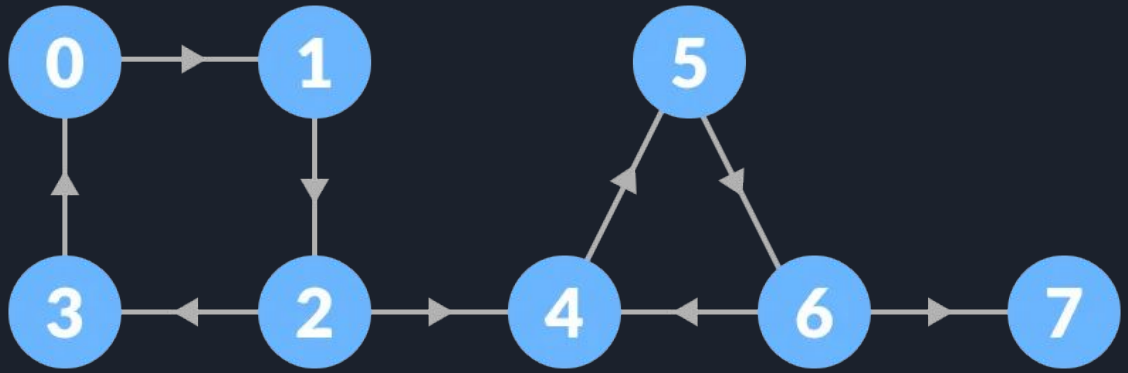
Strongly Connected Component

The strongly connected components of a graph divide the graph into strongly connected parts that are as large as possible.

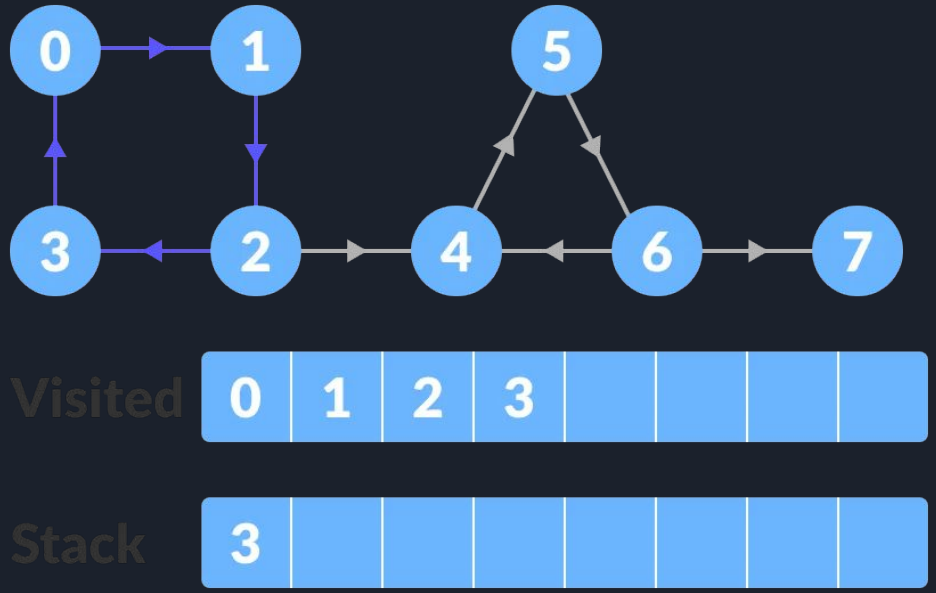


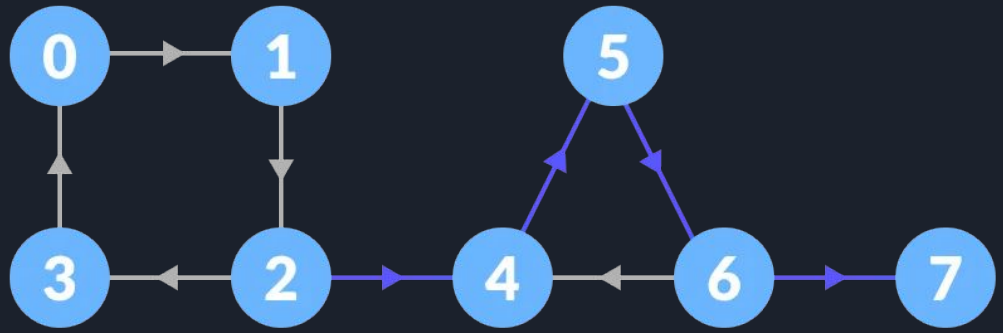
Strongly Connected Component

Given Graph:



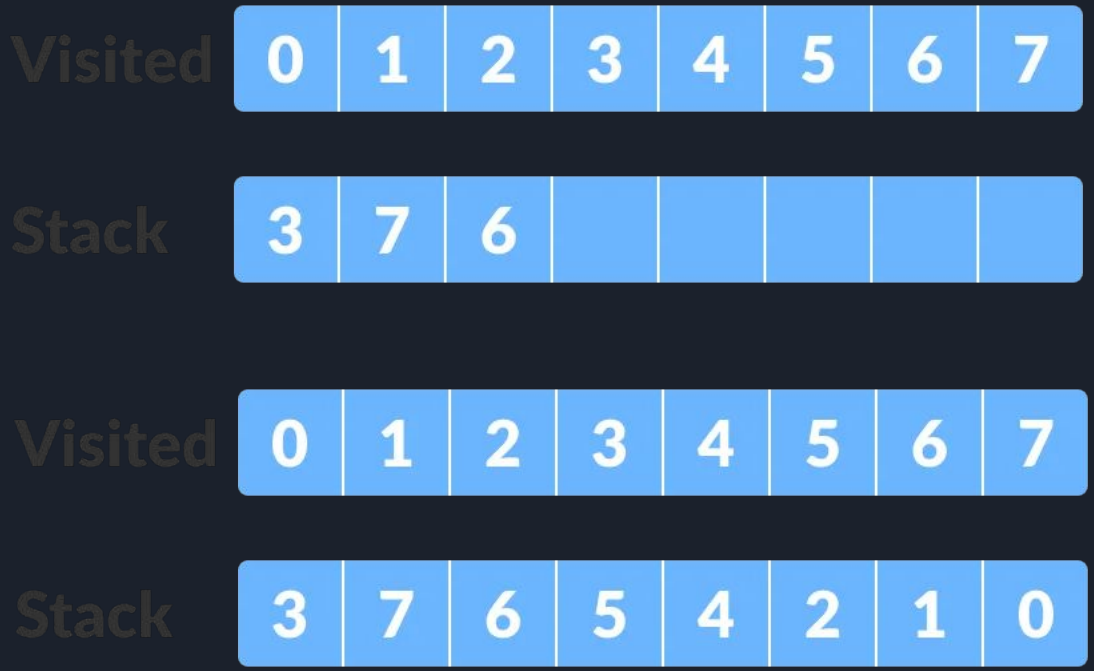
Step - 1:



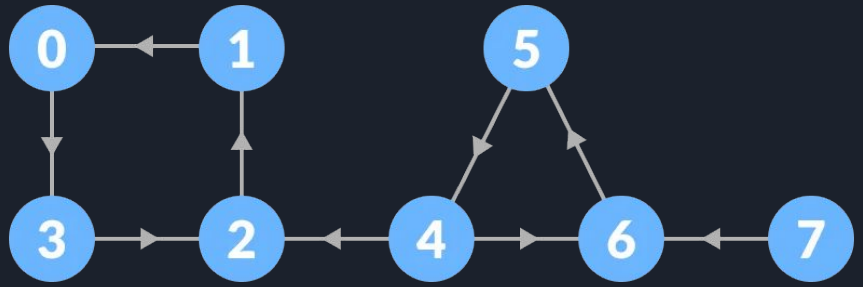


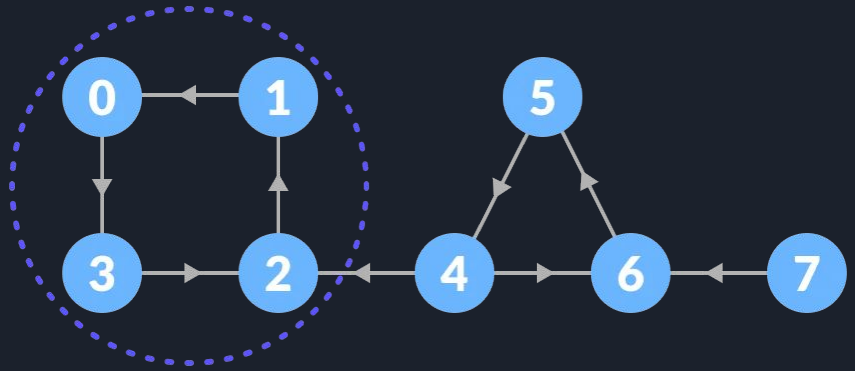
Visited	0	1	2	3	4	5	6	7

Stack	3	7						

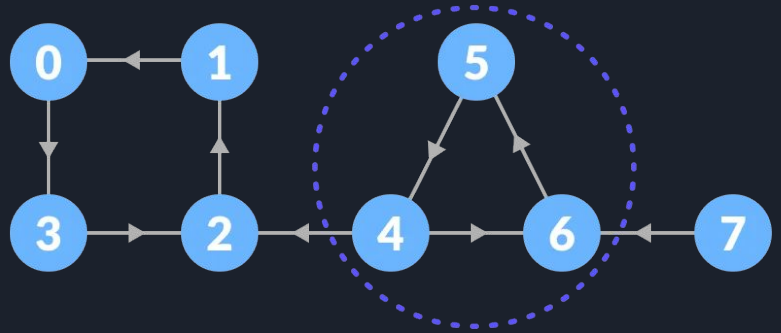

Visited
0 1 2 3 4 5 6 7
Stack
3 7 6
Visited
0 1 2 3 4 5 6 7
Stack
3 7 6 5 4 2 1 0

Step - 2:

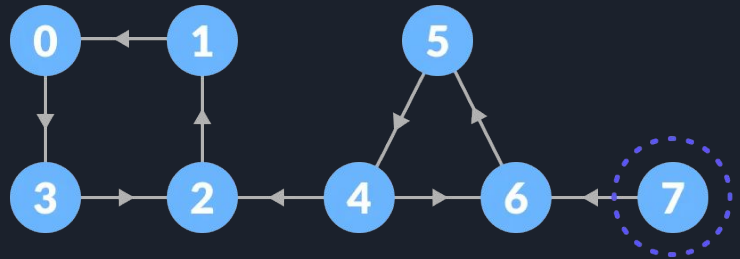




Visited	0	1	2	3				
Stack	3	7	6	5	4	2	1	
SCC	0	1	2	3				



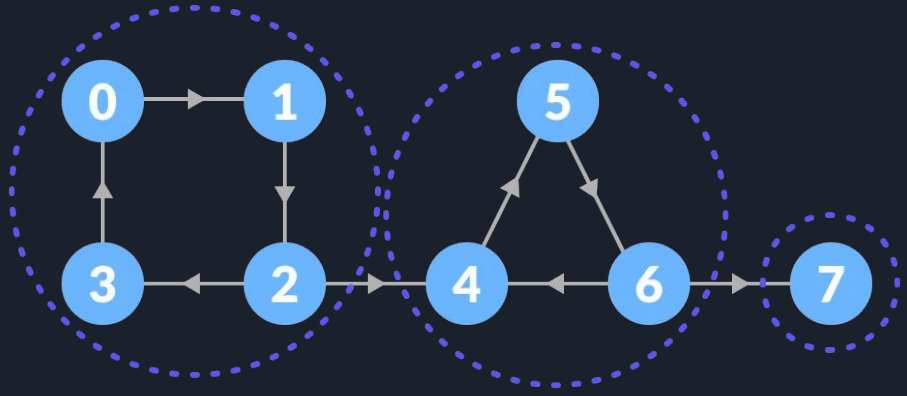
Visited	0	1	2	3	4	5	6	
Stack	3	7	6	5				
SCC	4	5	6					



Visited	0	1	2	3	4	5	6	

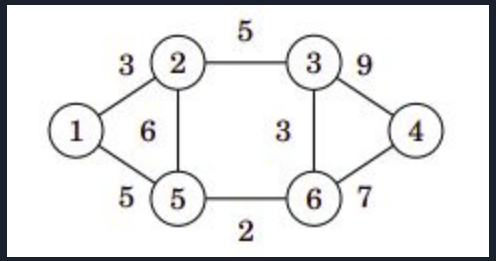
Stack								

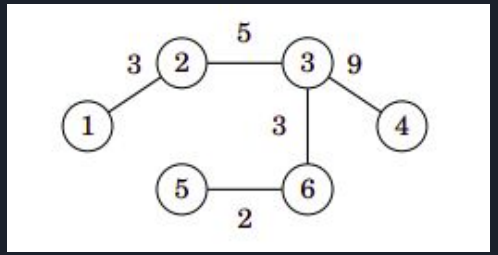
SCC	7							


0
1
5
3
2
4
6
7

Spanning Trees

A spanning tree of a graph consists of all nodes of the graph and some of the edges of the graph so that there is exactly one path between any two nodes.

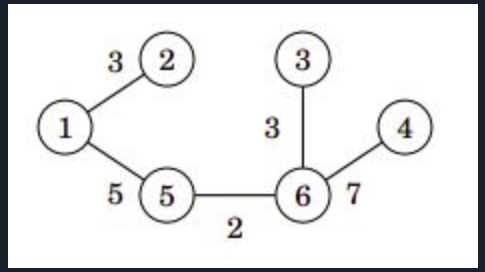




The weight of a spanning tree is the sum of its edge weights. For example, the weight of the above (right) spanning tree is 3+5+9+3+2 = 22.

Minimum Spanning Tree

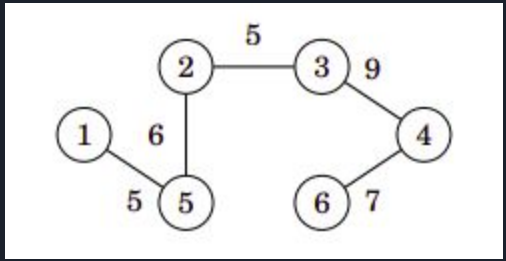
A minimum spanning tree is a spanning tree whose weight is as small as possible.



The weight of the minimum spanning tree for the example graph is 20.

Maximum Spanning Tree

A maximum spanning tree is a spanning tree whose weight is as large as possible.



The weight of a maximum spanning tree for the example graph is 32.

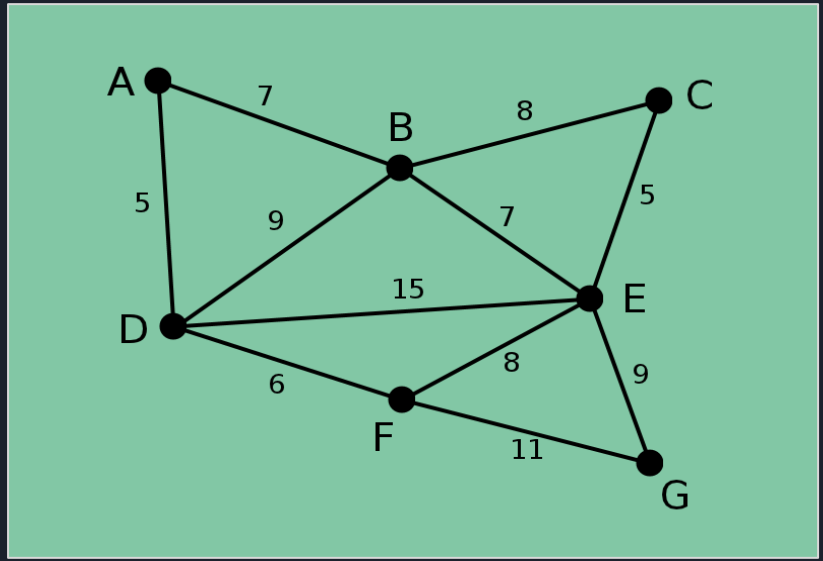
Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm to find the minimum spanning tree of a graph. The algorithm works as follows:

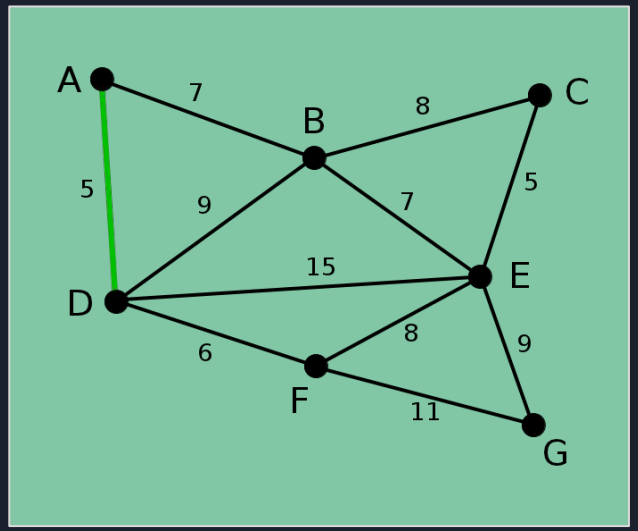
1. Sort all the edges from low weight to high.
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

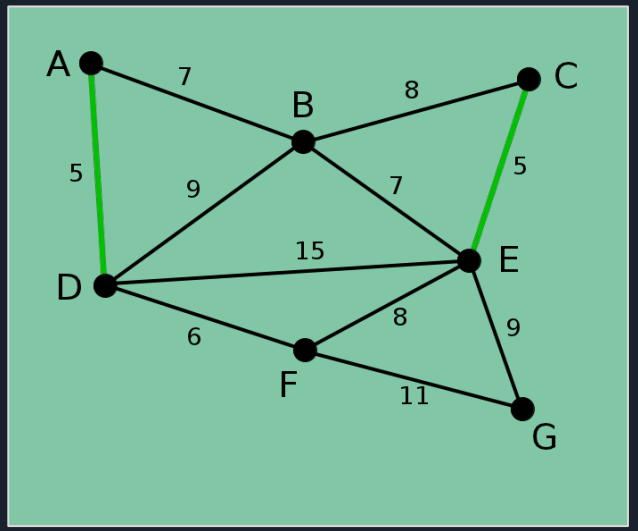
Kruskal’s Algorithm

Consider the given graph:

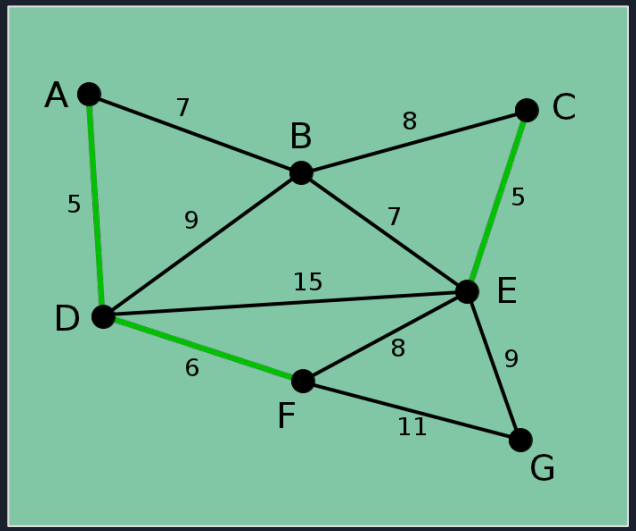


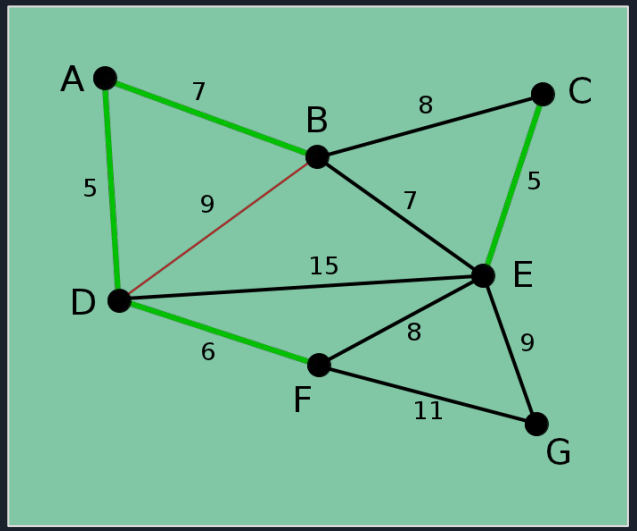
Kruskal’s Algorithm



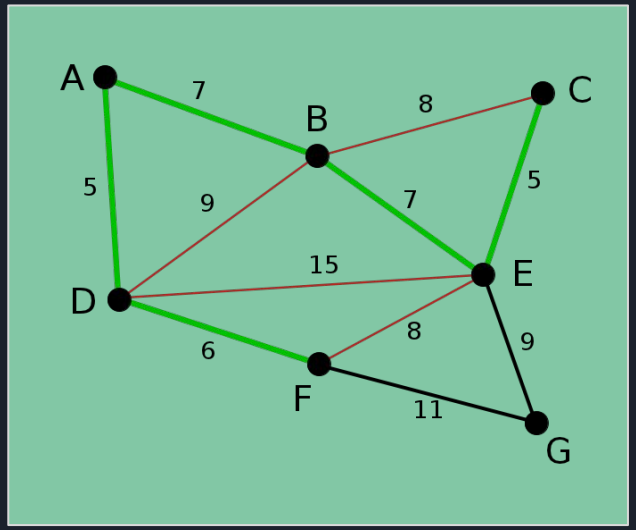


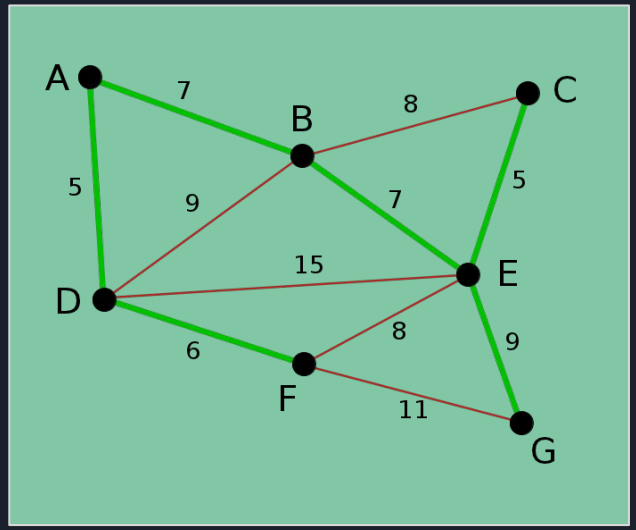
Kruskal’s Algorithm





Kruskal’s Algorithm





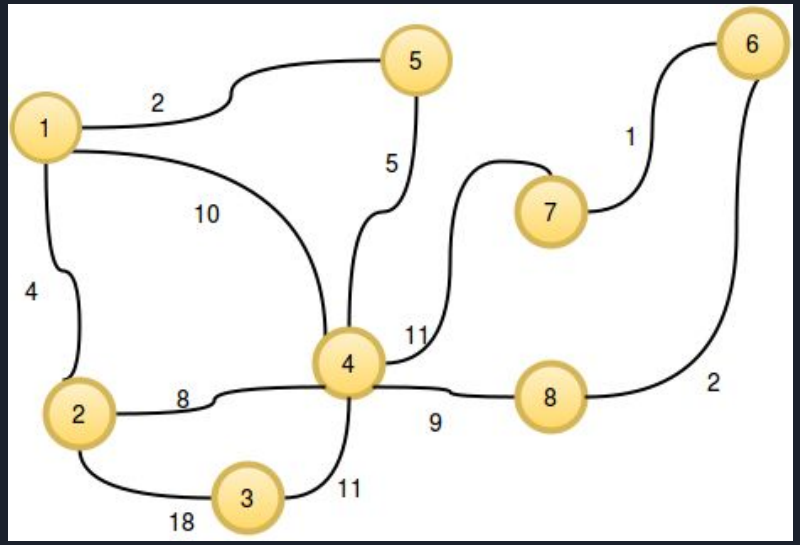
Prim’s Algorithm

Kruskal’s algorithm is also a greedy algorithm to find the minimum spanning tree of a graph. The algorithm works as follows:

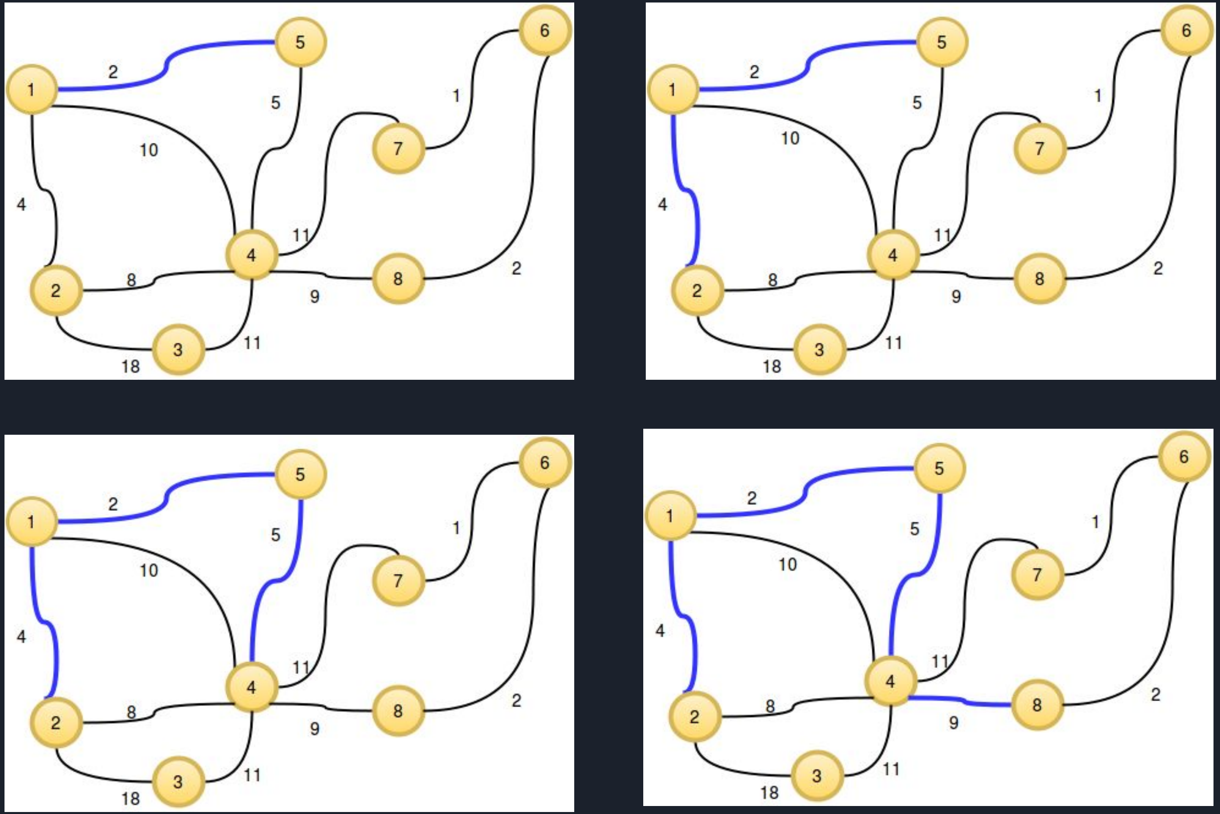
1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.
3. Keep repeating step 2 until we get a minimum spanning tree.

Prim’s Algorithm

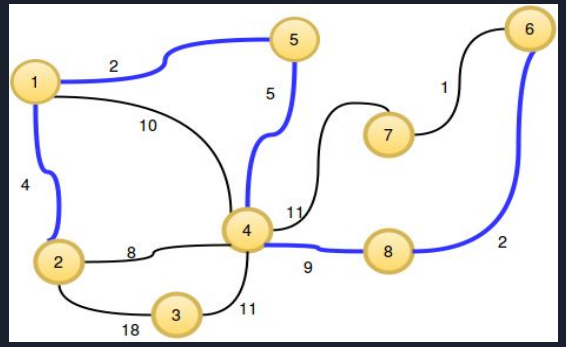
Consider the given graph:

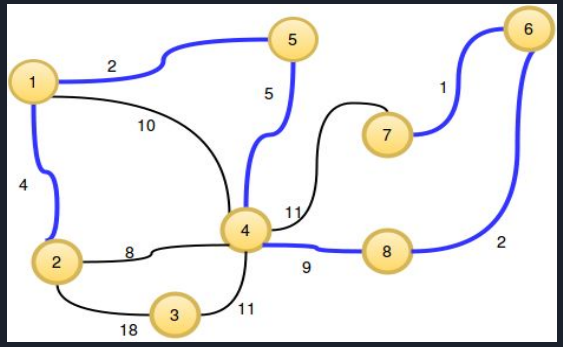


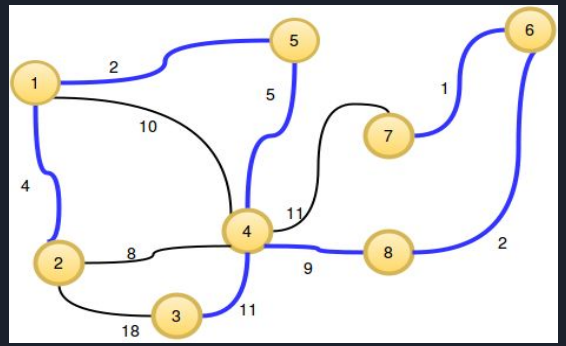
Prim’s Algorithm

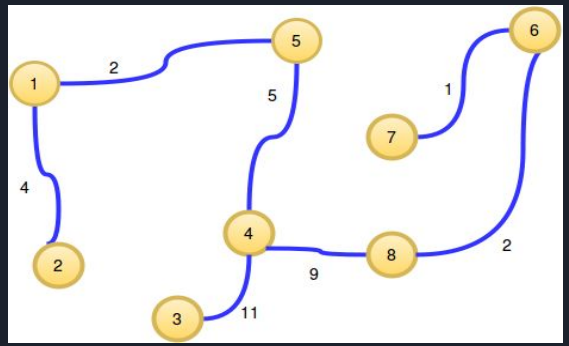


Prim’s Algorithm









Problem (MST)

You are given an n×n distance matrix. (The cell of i-th row and j-th column represents the distance between node i and j). Determine if it is the distance matrix of a weighted tree (all weights must be positive integers).

Constraints:

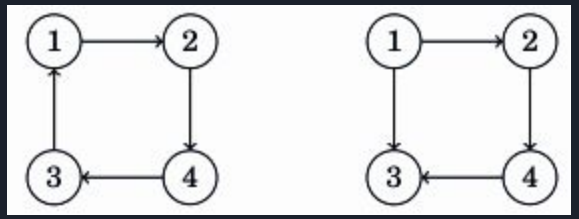
$1 \le n \le 2000$

$0 \le d[i][j] \le 1e9$

Problem Source: CF-472D

Strongly Connected Component

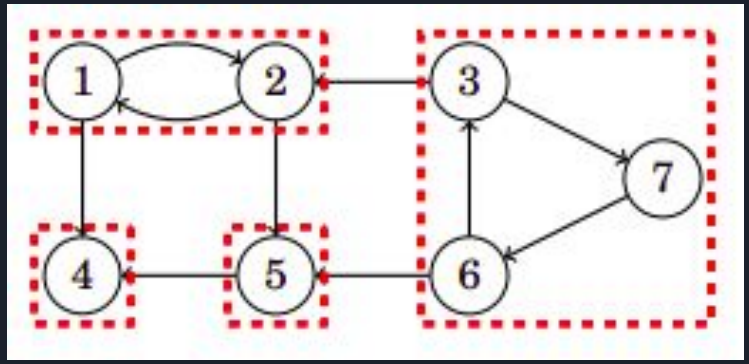
A graph is strongly connected if there is a path from any node to all other nodes in the graph.



The left graph is strongly connected but the right graph is not.

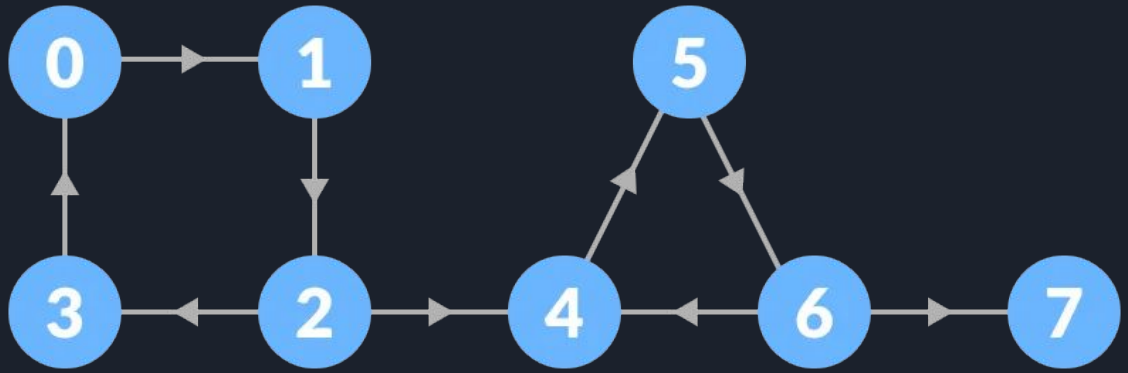
Strongly Connected Component

The strongly connected components of a graph divide the graph into strongly connected parts that are as large as possible.



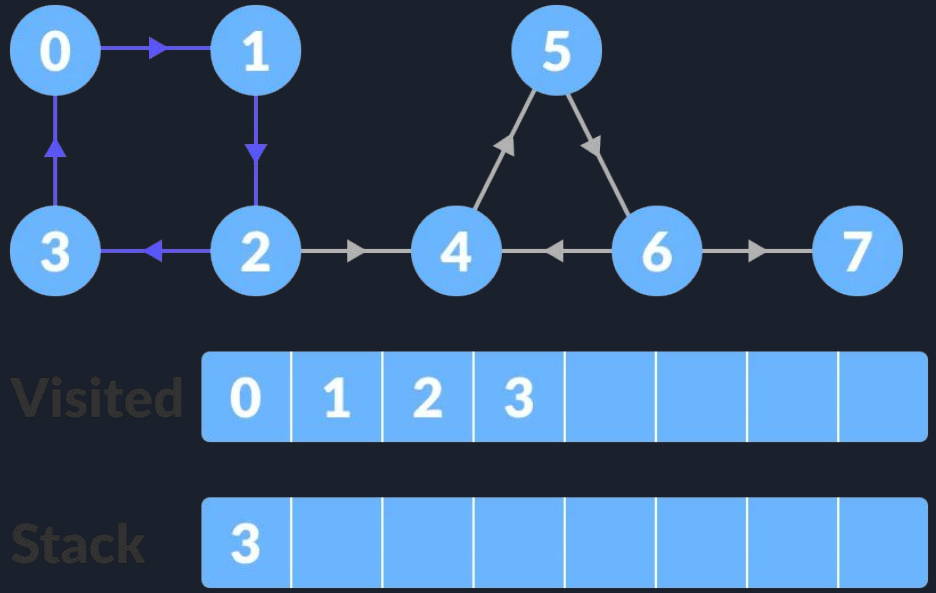
Kosaraju’s Algorithm

Given Graph:

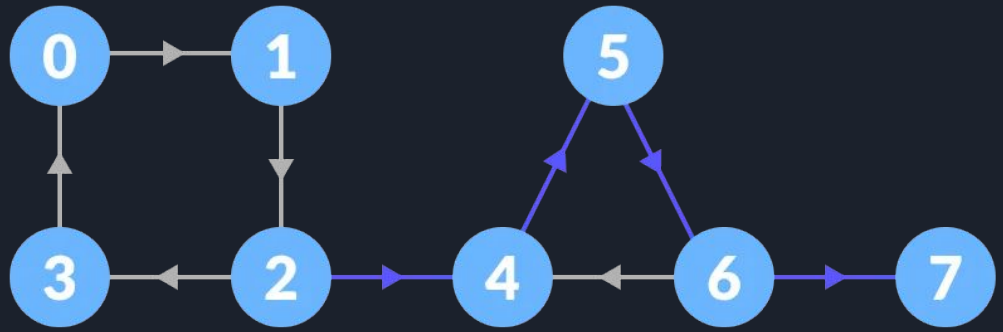


Kosaraju’s Algorithm

Step - 1:



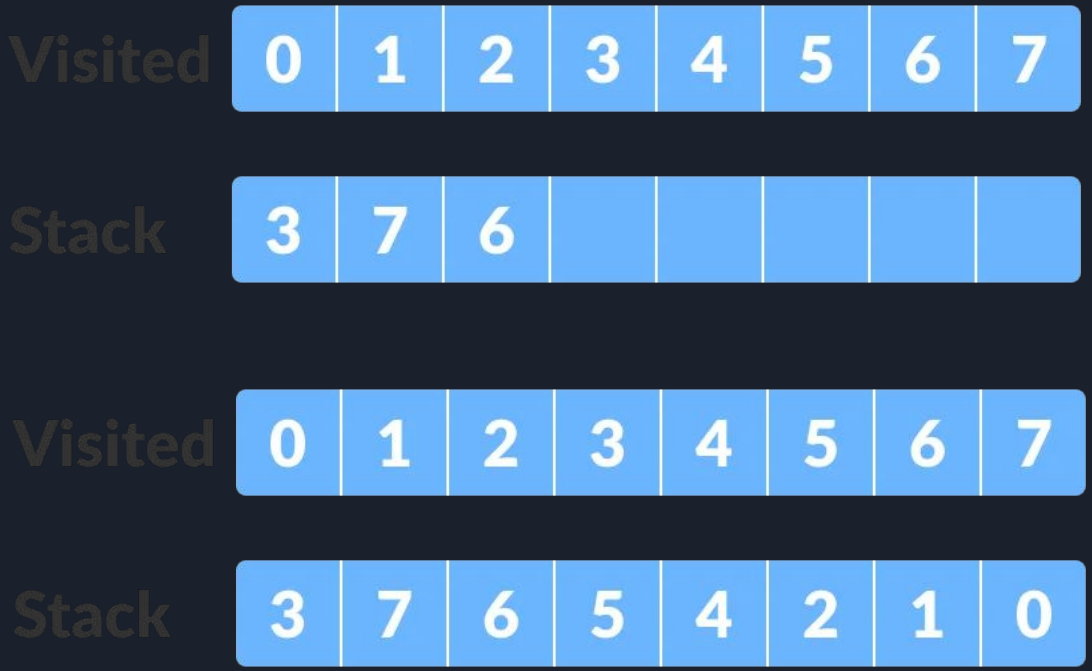
Kosaraju’s Algorithm



Visited	0	1	2	3	4	5	6	7

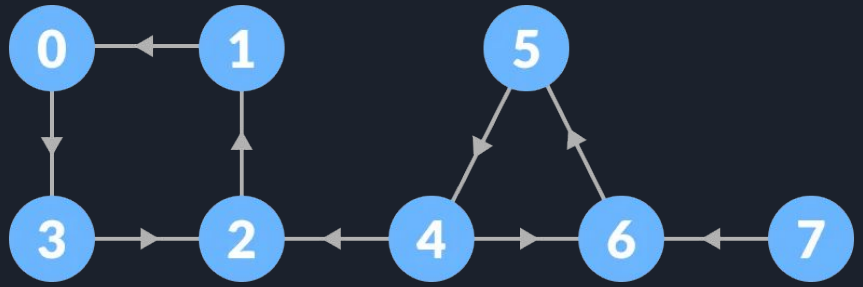
Stack	3	7						

Kosaraju’s Algorithm

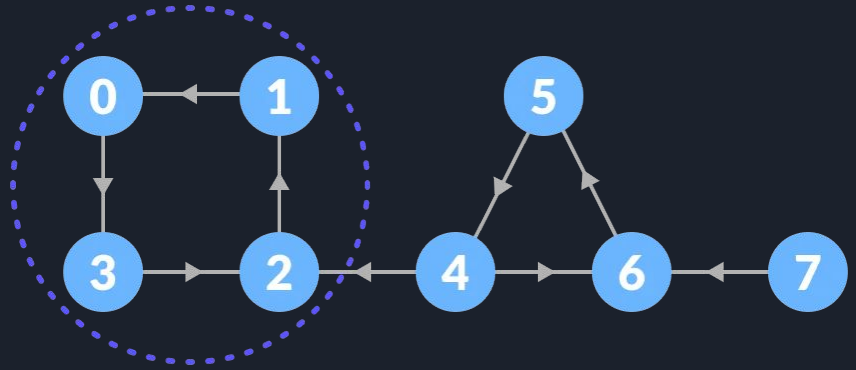


Kosaraju’s Algorithm

Step - 2:

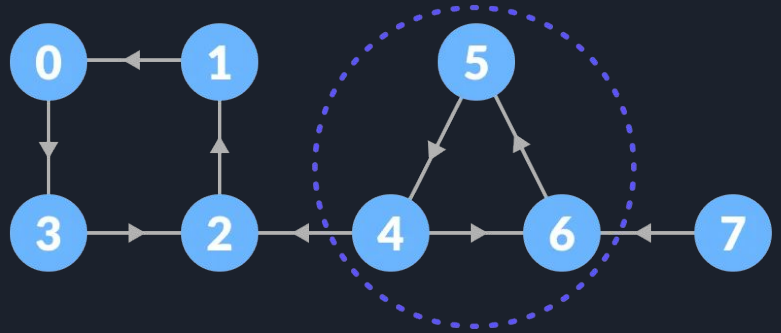


Kosaraju’s Algorithm



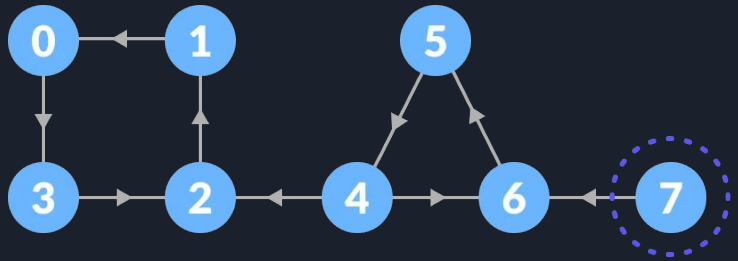
Visited	0	1	2	3				
Stack	3	7	6	5	4	2	1	
SCC	0	1	2	3				

Kosaraju’s Algorithm



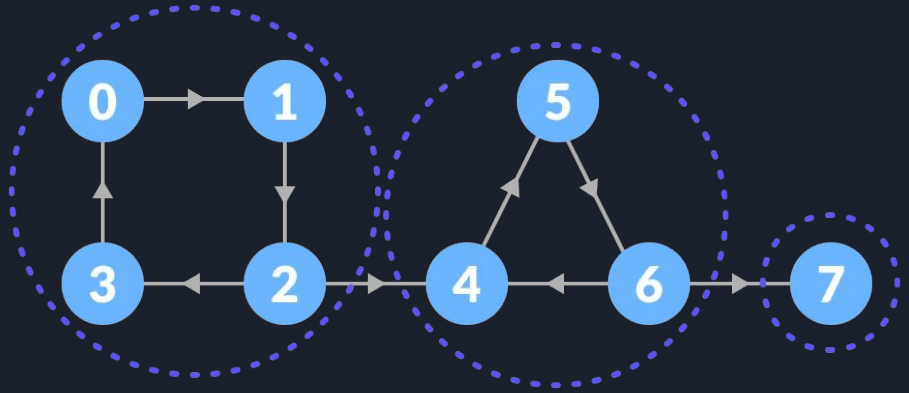
Visited	0	1	2	3	4	5	6	
Stack	3	7	6	5				
SCC	4	5	6					

Kosaraju’s Algorithm



Visited	0	1	2	3	4	5	6	
Stack								
SCC	7							

Kosaraju’s Algorithm



Problem (SCC)

A game has n rooms and m tunnels between them. Each room has a certain number of coins. What is the maximum number of coins you can collect while moving through the tunnels when you can freely choose your starting and ending room?

Input:

The first input line has two integers n and m: the number of rooms and tunnels. The rooms are numbered 1,2,…,n.

Then, there are n integers k1,k2,…,kn: the number of coins in each room.

Finally, there are m lines describing the tunnels. Each line has two integers a and b: there is a tunnel from room a to room b. Each tunnel is a one-way tunnel.

Output:

Print one integer: the maximum number of coins you can collect.

Problem Source: CSES-1686