Directed Graphs & Topo Sort

- Raghav Goel

Goal

Pre requisites → • Basics of graphs

• DFS

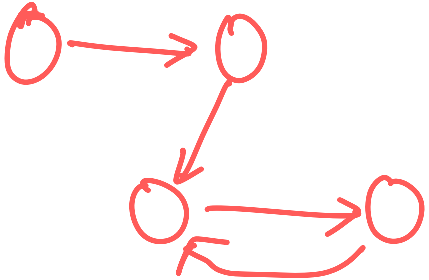
• BFS

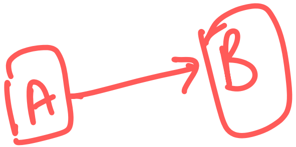
To understand

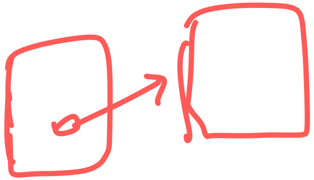
- Directed Graphs & Directed Acyclic Graphs (DAGs)
- Cycle Detection
- Topological Sorting

Directed Graphs

- A directed graph (digraph) is a set of vertices connected by directed edges. Each edge has a direction, meaning it goes from one vertex to another.
- Examples:
  - Web Links (A web page linking to another).
  - Social Media (Followers on Twitter/X).





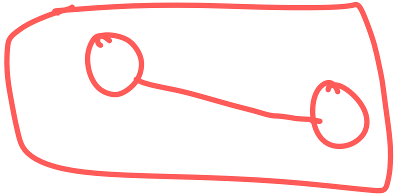


Directed Acyclic Graphs (DAGs)

- A Directed Acyclic Graph (DAG) is a directed graph with no cycles.
- You can never return to the same node once you follow a path.
- Allows Topological Sorting (A linear ordering of nodes).
- Applications:
  - Task Scheduling (e.g., CPU scheduling, project management)

$$P_1 \rightarrow P_2 \rightarrow P_3$$

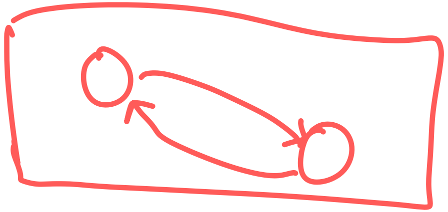
Undirected Graphs

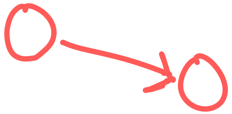


DFS

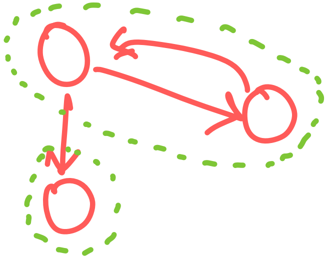
components

Directed Graphs





from node = 1 to n
dfs(node)



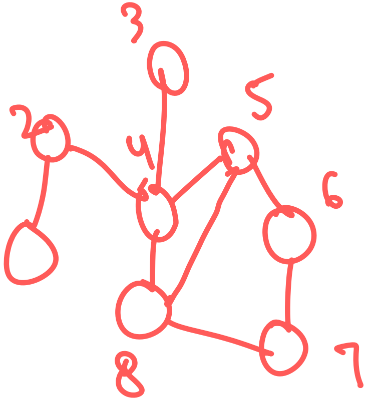
Strongly connected components

Cycle Detection in Directed Graphs

- Maintain two arrays:
  - visited → To track visited nodes.
  - inStack → To track nodes in the current DFS call.
- For each unvisited node, perform DFS.
- If a node is already in the recursion stack, a cycle is detected.
- If DFS completes without detecting a cycle, the graph is DAG.
- Time Complexity: O(V + E)

Cycle Detection in Directed Graph

undirected

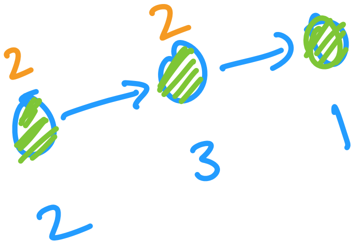


DFS

vis[2] = 2 , vis[3] = 2

1 → 1

2 → 2, 3, 1



vis[i] = true/false

vis[i] → 0, 1, 2, 3 . . . .

DFS O(V+E)

O(V)

CP Algorithms

whether the node is visited in the same dfs traversal or not

~~int vis[]~~

bool vis[ ]   vis[i] → true /false

bool instack[ ]

dfs(i) {
instack[i] → true

instack[i] → false
}

Implementation of Cycle Detection

dfs

```
bool checkCycle(int node) {
  visited[node] = inStack[node] = true;
  for (int child : adj[node]) {
    if (inStack[child]) return true;
    if (!visited[child] && checkCycle(child)) return true;
  }
  inStack[node] = false;
  return false;
}
```

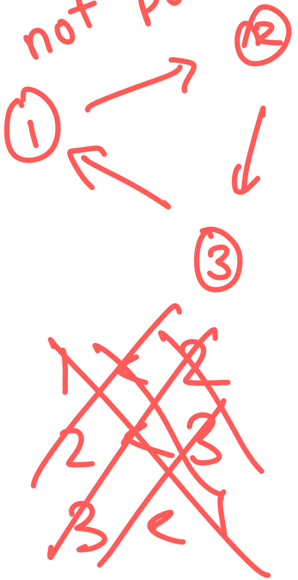
TC → O(V+E)

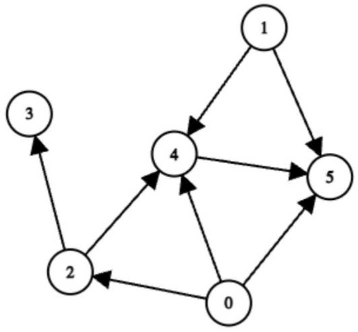
SC → O(V)

Topological Sorting/Ordering

- A linear ordering of vertices such that for every directed edge $u \longrightarrow v$, u appears before v in the ordering.
- Only possible for Directed Acyclic Graphs (DAGs).
- Every DAG has at least one valid topological order.

not possible





```
0 -> 2
0 -> 4
0 -> 5
1 -> 4
1 -> 5
2 -> 3
2 -> 4
4 -> 5
```

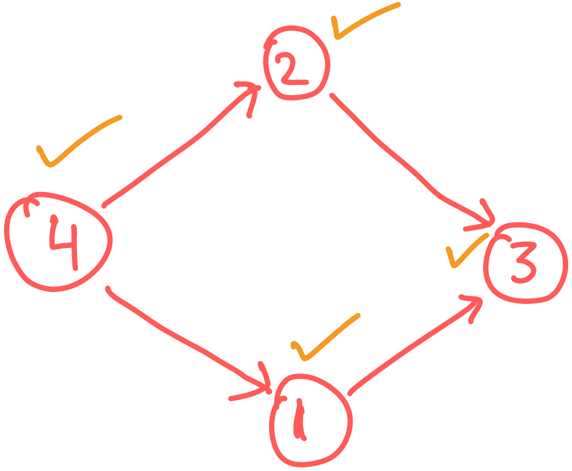
Topological Ordering: 0 2 3 1 4 5

Task : Find any valid topological ordering

1. DFS

2. BFS

DFS



1 → 1, 3

2 → 2, 3

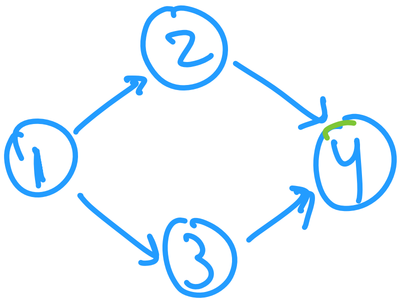
3 → 3

4 → 4 → 1, 2

dfs (1)
dfs(2)
dfs(3)
dfs(4)

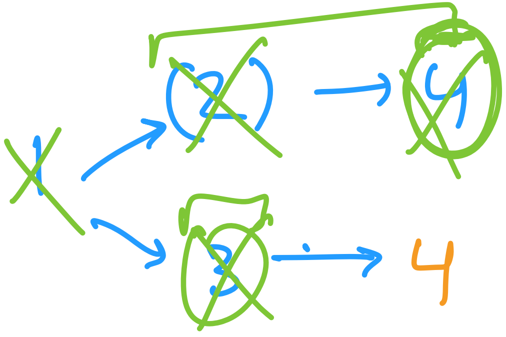
Topo ordering → 4 2 1 3

→ dfs(4) dfs(3) dfs(2) dfs(1)



dfs(1)

dfs(2) dfs(3)

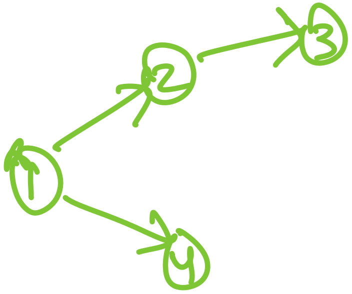


dfs(4) → 1 3 2 4

Theoretical

timein time_out

tin tout



u v

tout[u] > tout[v]

u=1 v=3

u should come before v

Simpler visualization

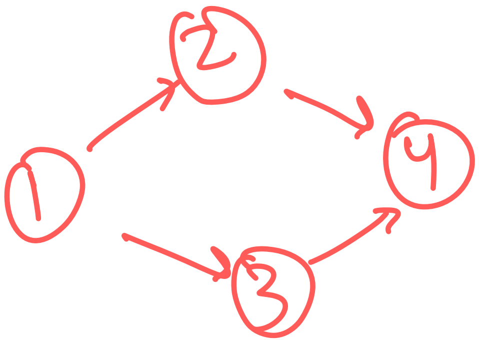
u

dfs(u) →

→

Topo → u - - - - - - -

1 3 2 4



~~dfs(1)~~ → ~~dfs(2)~~ → ~~dfs(4)~~

~~dfs(1)~~ → ~~dfs(3)~~ → ~~dfs(4)~~

Topo Sort using DFS

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

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

$\rightarrow O(n)$ TC

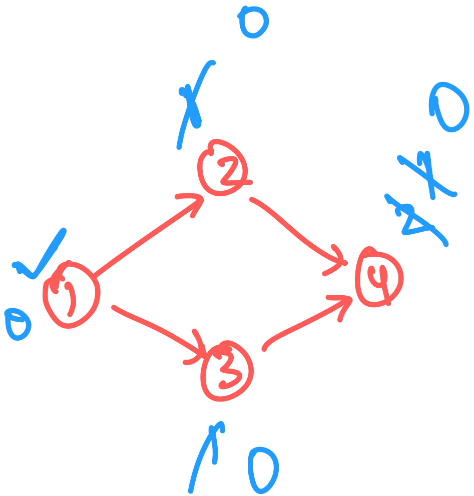
$\rightarrow O(n)$

Overall TC

$\rightarrow O(n)$

SC $\rightarrow O(n)$

BFS



there must exist atleast one node with indegree = 0

1 2 3 4

in[ ]

Topo Sort using BFS (Kahn's Algorithm)

```
vector<int> kahns_algorithm(vector<vector<int>> &adj) {
  int n = adj.size();
  vector<int> indegree(n, 0);
  for (int u = 0; u < n; u++)
    for (int v : adj[u]) indegree[v]++;
  queue<int> q;
  for (int u = 0; u < n; u++)
    if (indegree[u] == 0) q.push(u);
  vector<int> topo;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    topo.push_back(u);
    for (int v : adj[u]) {
      indegree[v]--;
      if (indegree[v] == 0) q.push(v);
    }
  }
  return topo;
}
```

TC $\rightarrow$ $O(V+E)$

SC $\rightarrow$ $O(V)$

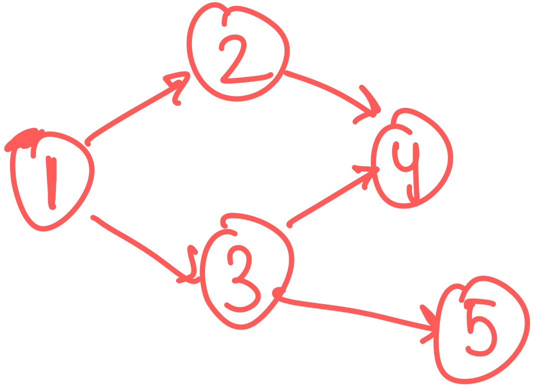
Applications of Topo Sort

- To check if a Directed Graph is a DAG or not.
- Task Scheduling
- Compilation Order in Programming
- Resolving Dependencies (e.g., Package Installation)

if Kahn's algo gives empty vector then graph is not a DAG

Problem

Given a DAG, find the Lexicographically Smallest Topological Sorting/Ordering



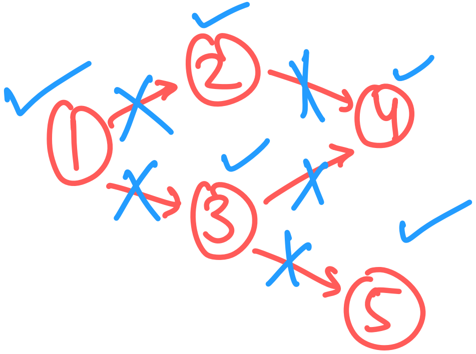
1, 2, 3, 4, 5

1 3 5 2 4

1 3 2 5 4

Kahn's Algo → 1, 2, 3, 4, 5

priority queue → ~~1~~ ~~3~~ ~~2~~ ~~4~~ ~~5~~ ~~4~~



Answer → use priority queue instead of queue