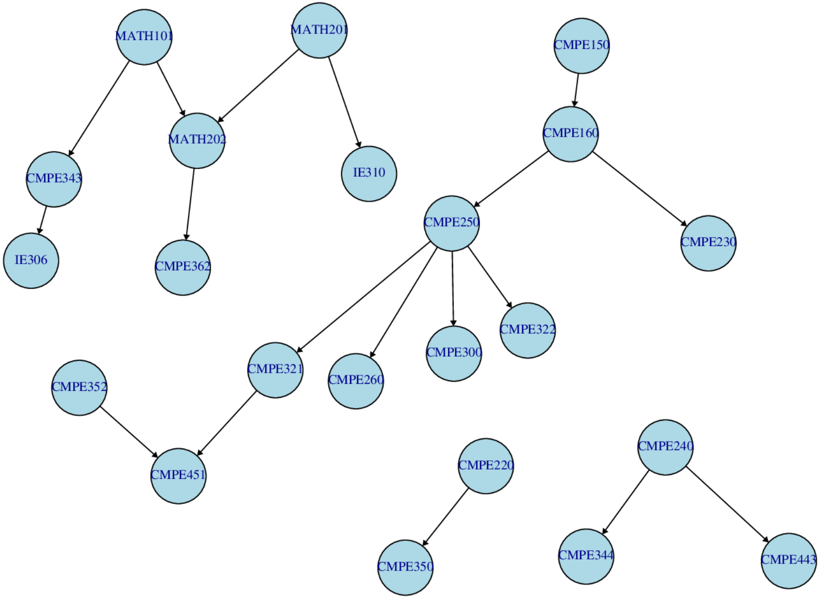
Graphs-Topological Sort

December 13, 2023

Introduction

- There are many problems involving a set of tasks in which some of the tasks must be done before others.
- For example, consider the problem of taking a course only after taking its prerequisites.
- Is there any systematic way of linearly arranging the courses in the order that they should be taken?

CMPE Course Prerequisites



Problem

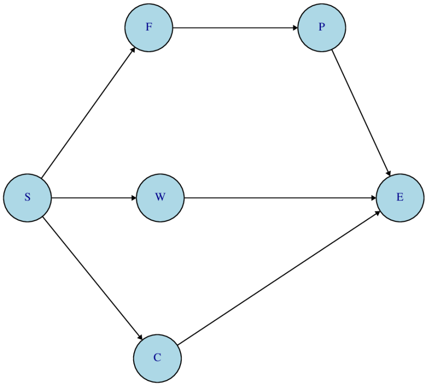
- **Input:** A directed acyclic graph $G = (V, E)$, also known as a partial order or poset.
- **Problem description:** Find a linear ordering of the vertices of V such that for each edge $(i, j) \in E$, vertex i is to the left of vertex j.

- **P**: Assemble pingpong table
- **F**: Carpet the floor
- **W**: Panel the walls
- **C**: Install ceiling
- How would you order these activities?
- Topological sorting can be used to schedule tasks under precedence constraints.

Problem Graph

S: Start **E**: End
P: Assemble pingpong table **F**: Carpet the floor
W: Panel the walls **C**: Install ceiling

Some possible orderings:
S C W F P E
S W C F P E
S F P W C E
S C F P W E



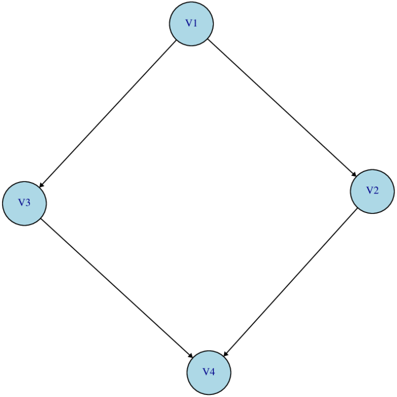
Topological Sort

- RULE:
  - If there is a path from u to v, then v appears after u in the ordering.

Graphs' Characteristics

- Directed
  - otherwise (u, v) means a path from u to v and from v to u. In that case, G cannot be ordered.
- Acyclic
  - otherwise u and v can be on a cycle, which results in u would precede v , v would precede u.
  - since any directed cycle provides an inherent contradiction to a linear order of tasks
- Each DAG has at least one topological sort
- There can be many such orderings for a given DAG
  - especially when there are few constraints

The ordering may not be unique



Possible orderings

V1, V2, V3, V4

V1, V3, V2, V4

Basic Algorithm

1. Compute the indegrees of all vertices
2. Find a vertex U with **indegree 0** and print it (store it in the ordering) **If** there is **no** such vertex then **there is a cycle** and the vertices cannot be ordered. Stop.
3. Remove U and all its edges (U, V) from the graph.
4. **Update** the indegrees of the remaining vertices.

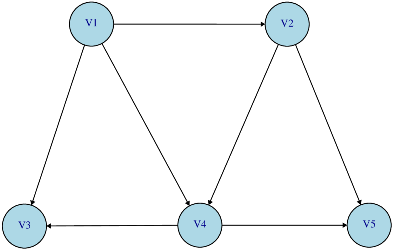
Repeat steps 2 through 4 while there are vertices to be processed.

Example

Indegrees
V1 0
V2 1
V3 2
V4 2
V5 2

First to be sorted: V1

New indegrees:
V2 0
V3 1
V4 1
V5 2



Possible sorts: V1, V2, V4, V3, V5; V1, V2, V4, V5, V3

Complexity

- $|V|$ - the number of vertices.
- To find a vertex of indegree 0 we scan all the vertices - $|V|$ operations.
- We do this for all vertices: $|V|^2$ operations
- Time Complexity: $O(|V|^2)$

Improved Algorithm

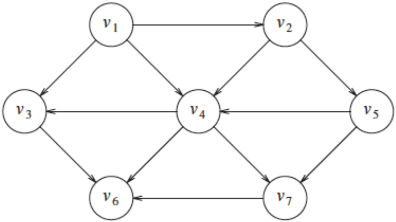
- Find a vertex of degree 0,
- Scan **only** those vertices whose indegrees have been **updated to 0**.

Improved Algorithm

1. Compute the indegrees of all vertices
2. Store all vertices with indegree 0 in a queue.
3. Dequeue vertex U from the queue.
4. For all edges (U, V) update the indegree of V, and put V in the queue if the updated indegree is 0.

Repeat steps 3 and 4 while the queue is not empty.

Example



	Indegree Before Dequeue #						
Vertex	1	2	3	4	5	6	7
v_1	0	0	0	0	0	0	0
v_2	1	0	0	0	0	0	0
v_3	2	1	1	1	0	0	0
v_4	3	2	1	0	0	0	0
v_5	1	1	0	0	0	0	0
v_6	3	3	3	3	2	1	0
v_7	2	2	2	1	0	0	0
Enqueue	v_1	v_2	v_5	v_4	v_3, v_7		v_6
Dequeue	v_1	v_2	v_5	v_4	v_3	v_7	v_6

Complexity

- $|V|$ - number of vertices,
- $|E|$ - number of edges.
- Operations needed to compute the indegrees:
  - Adjacency lists: $O(|E|)$
  - Adjacency Matrix representation: $O(|V|^2)$
- Complexity of the improved algorithm
  - Adjacency lists: $O(|E| + |V|)$,
  - Adjacency Matrix representation: $O(|V|^2)$
- Note that if the graph is complete $|E| = O(|V|^2)$