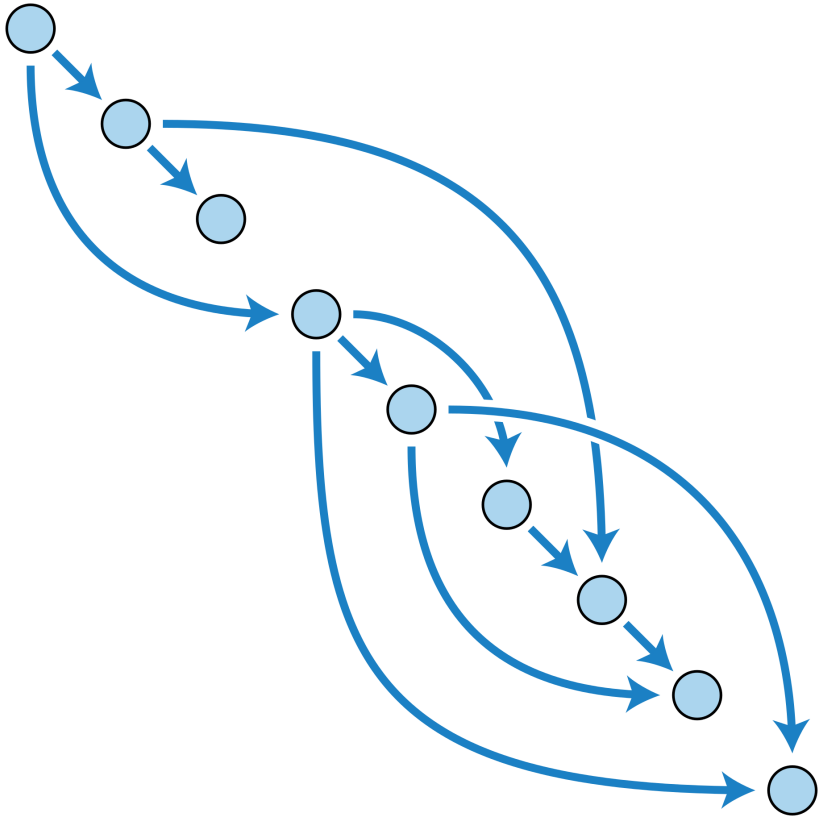
Topological Sort

Shusen Wang

Directed Acyclic Graph (DAG)

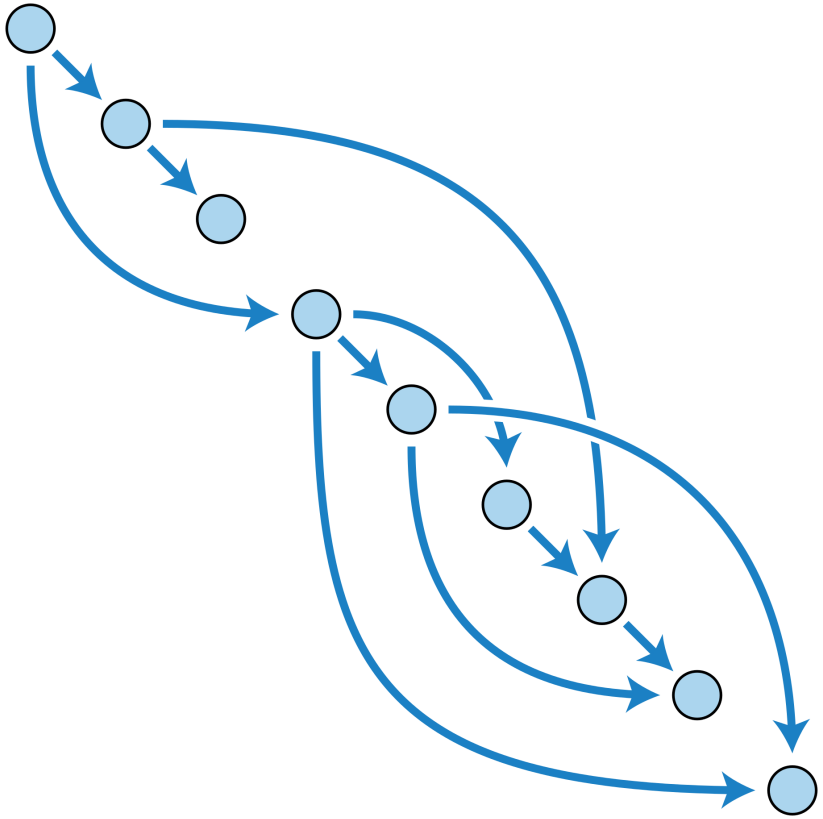


This is a DAG

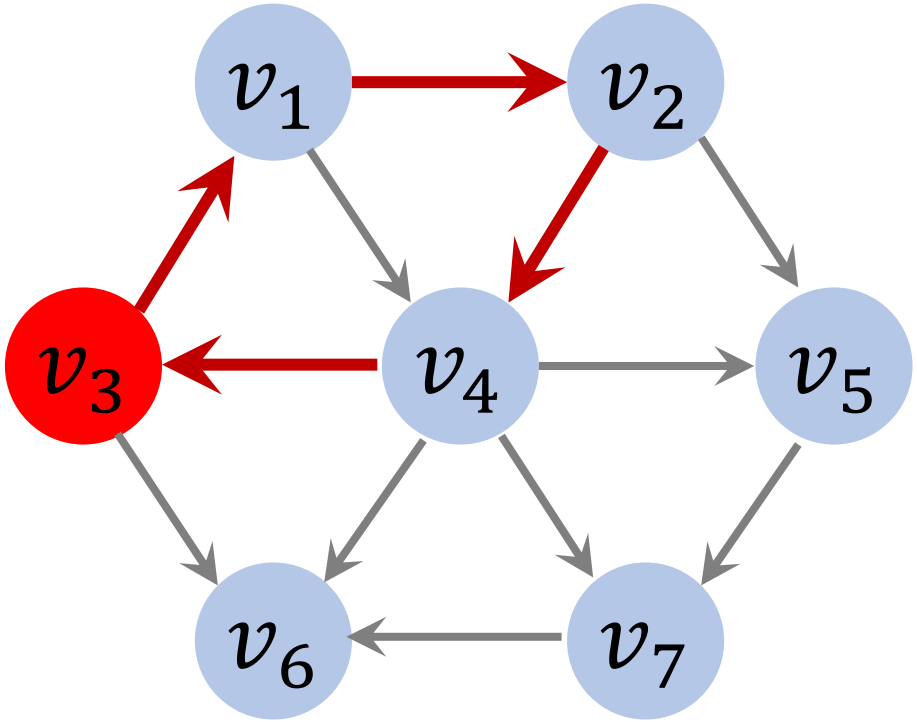
Definition of DAG

- DAG is a directed graph with no directed cycles.
- There is no way to start at any vertex v and follow a path that eventually loops back to v again.

Directed Acyclic Graph (DAG)



This is a DAG

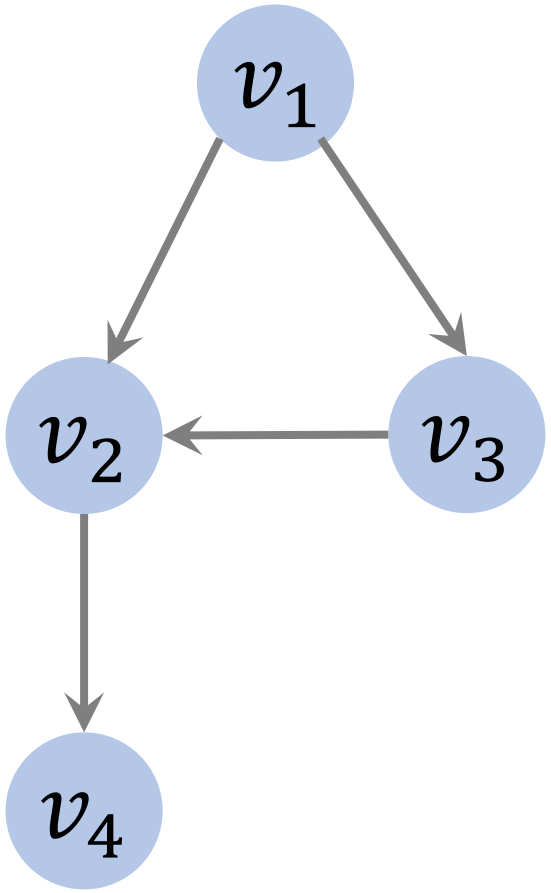


This is not a DAG

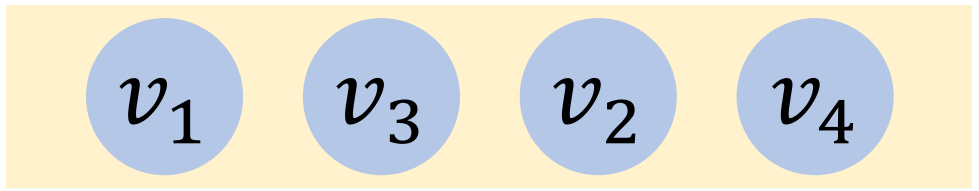
Topological Sort

- The graph must be a DAG.
- A **topological sort** is an ordering of vertices such that if there is a path from u to v, then u appears before v.

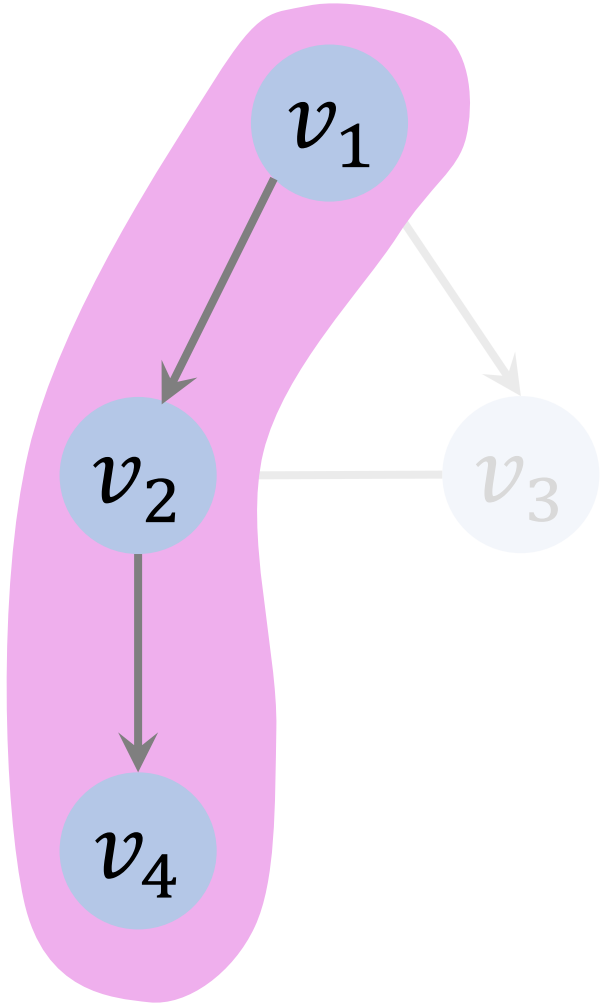
Example 1



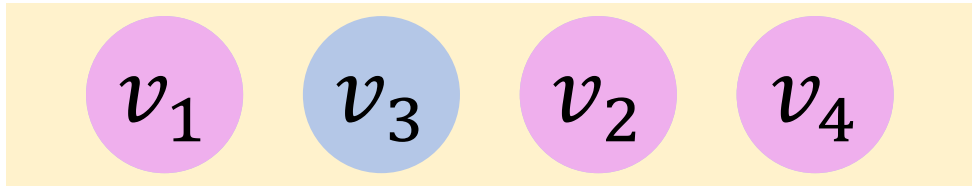
Sort



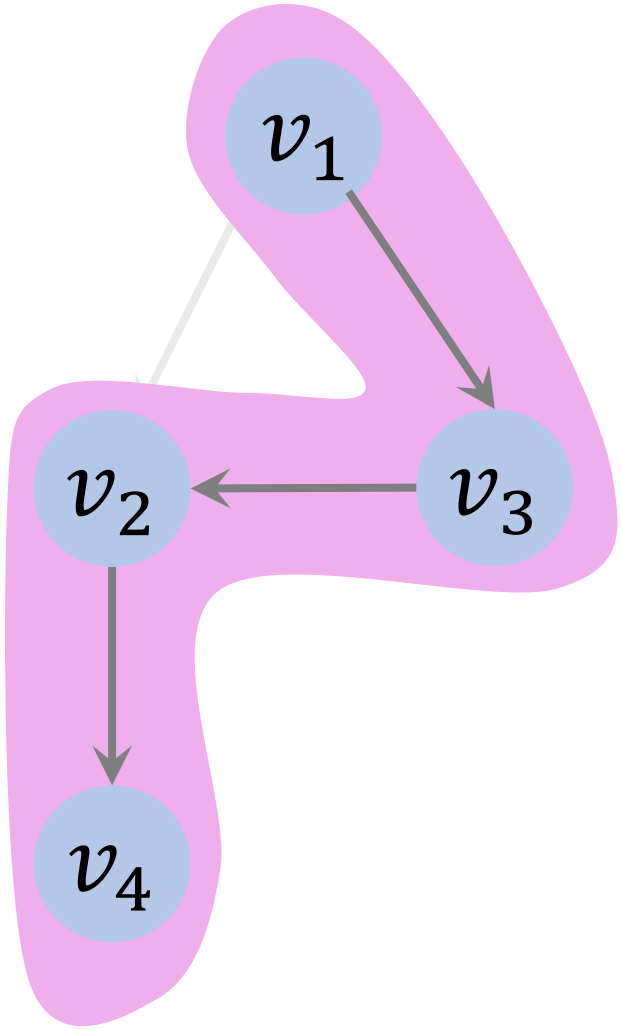
Example 1



Sort



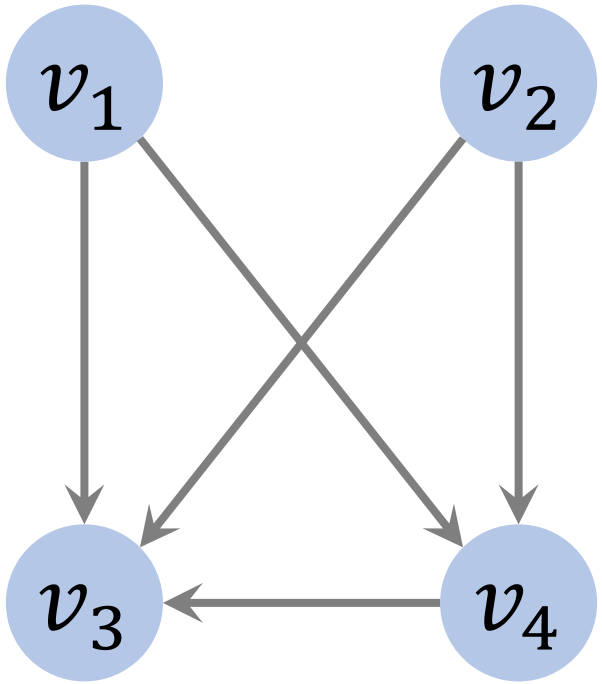
Example 1



Sort

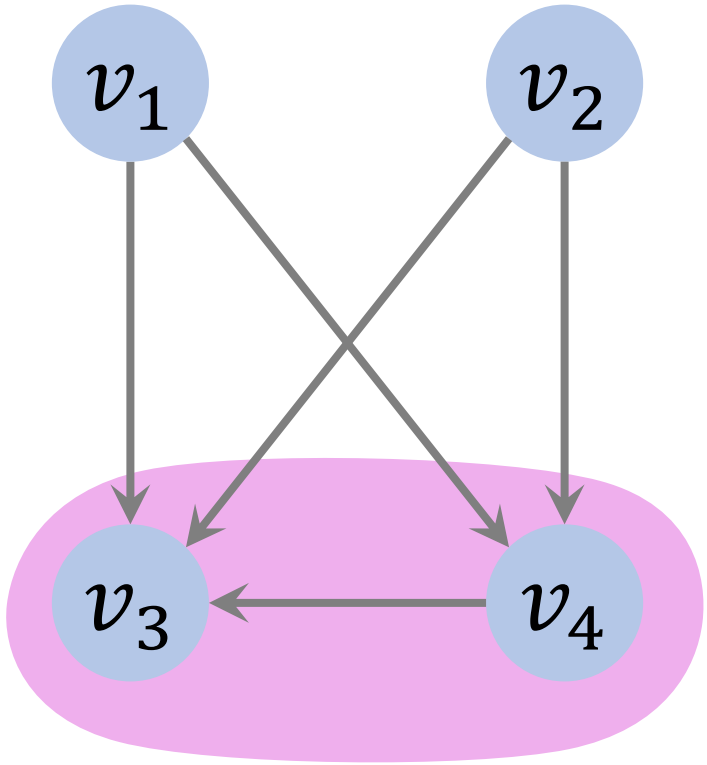
v_1 v_3 v_2 v_4

Example 2

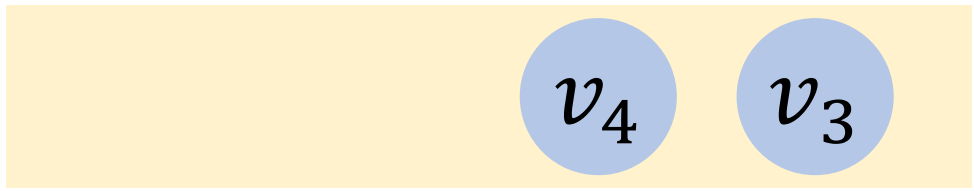


Sort

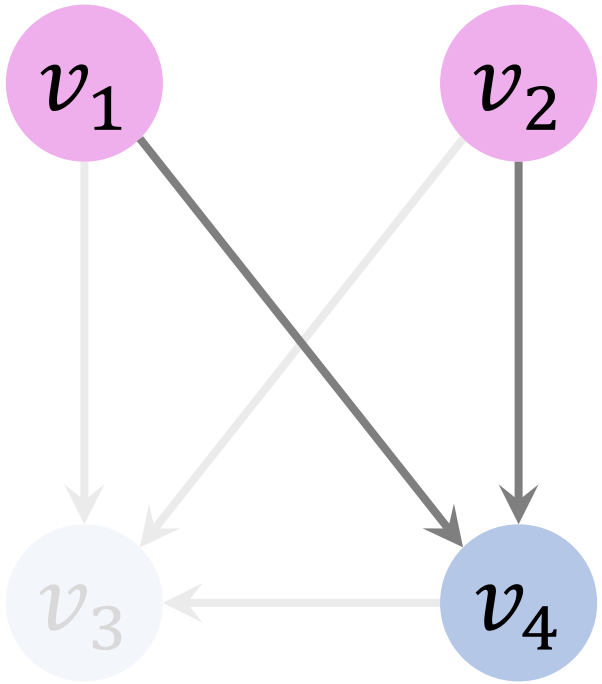
Example 2



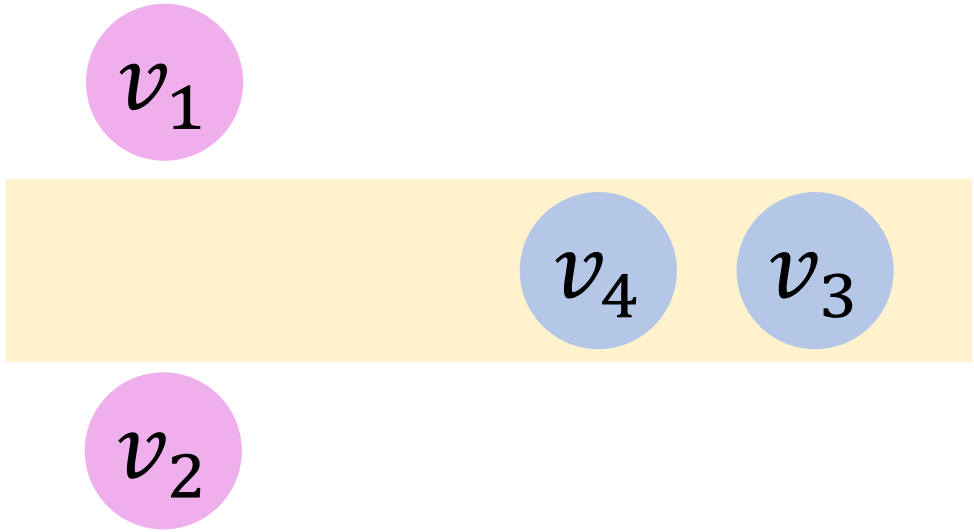
Sort



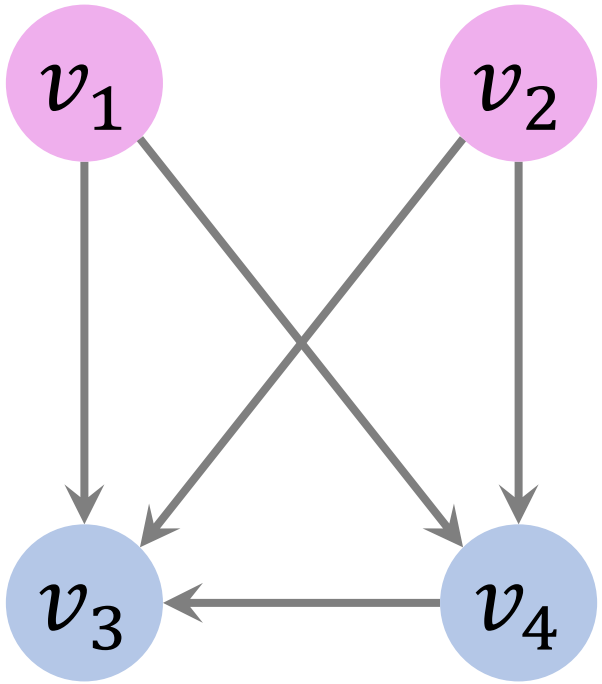
Example 2



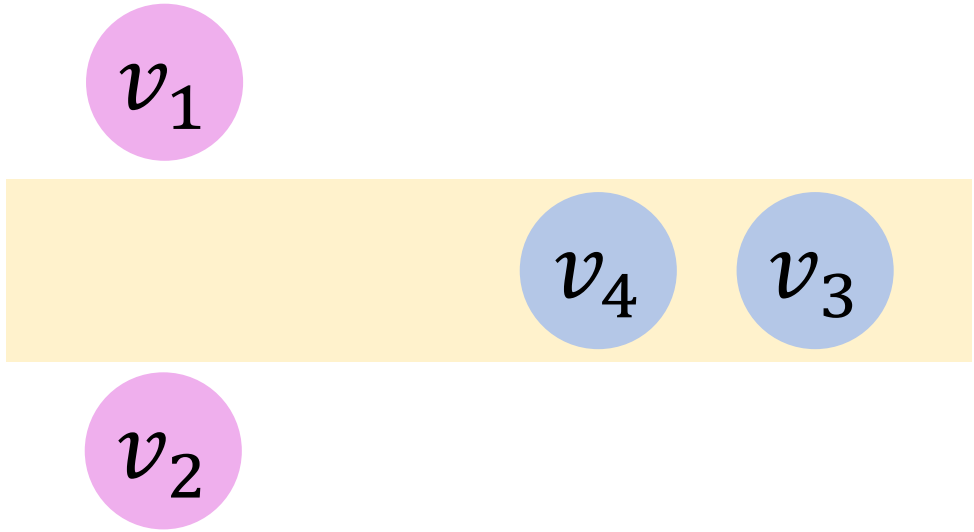
Sort



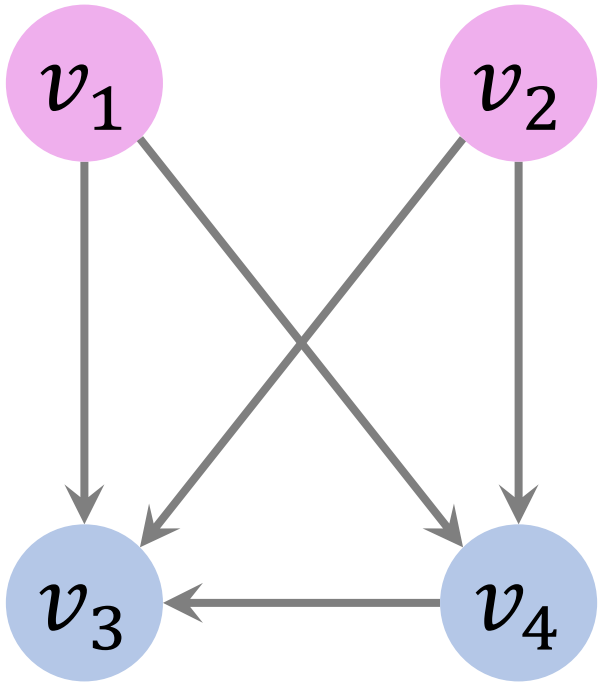
Example 2



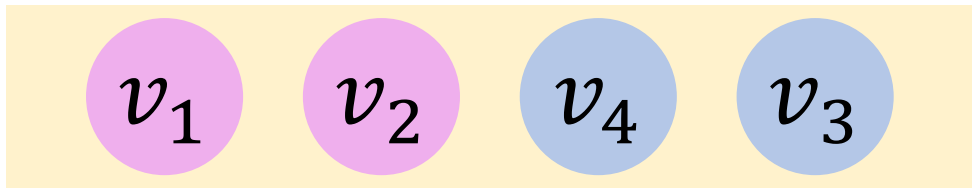
Sort



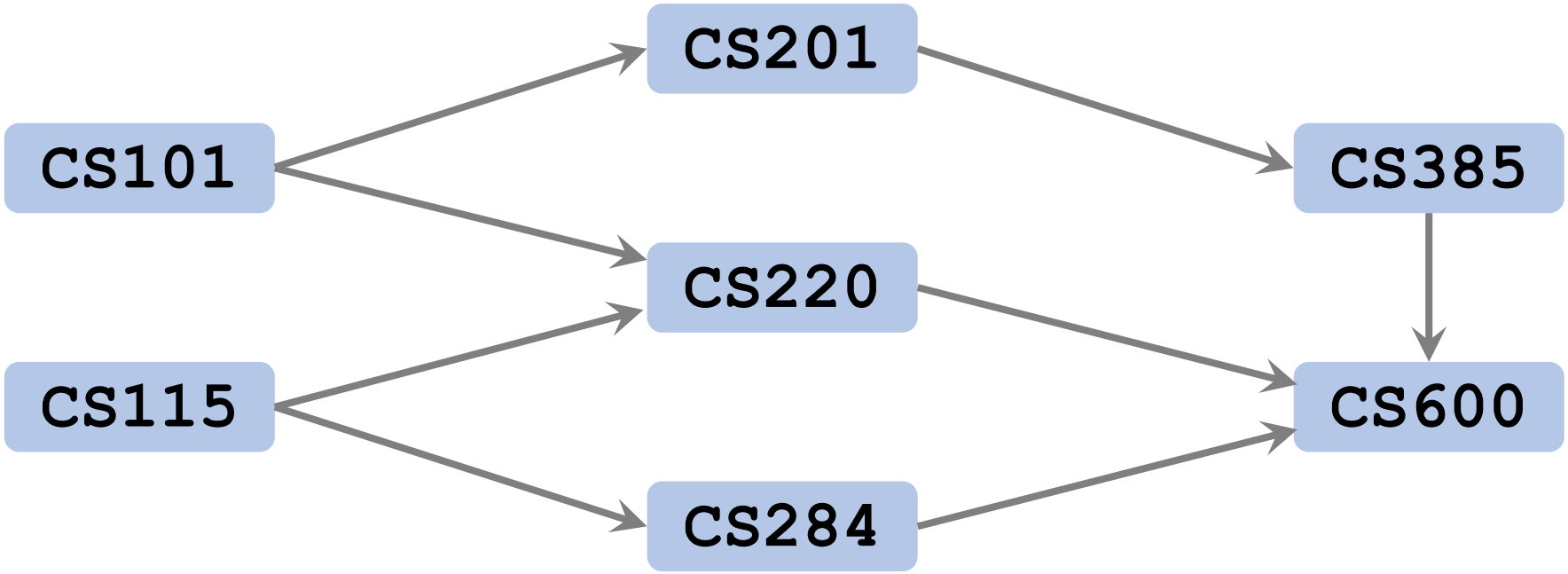
Example 2



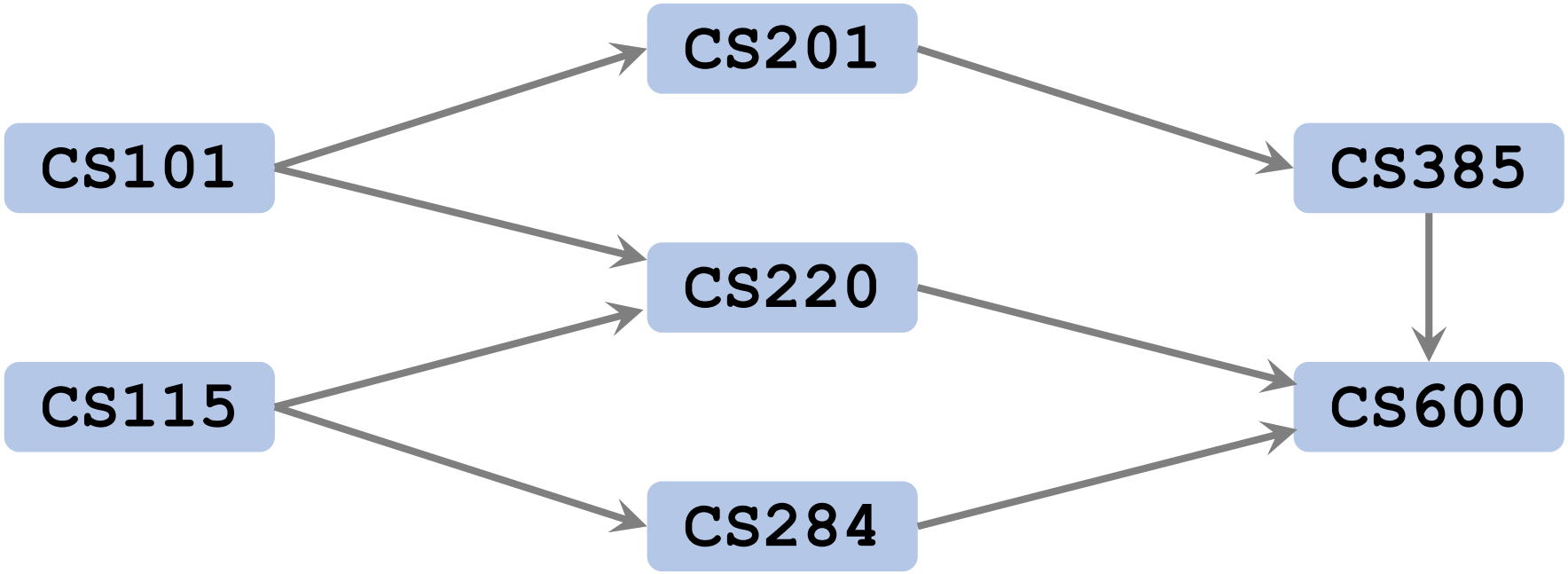
Sort



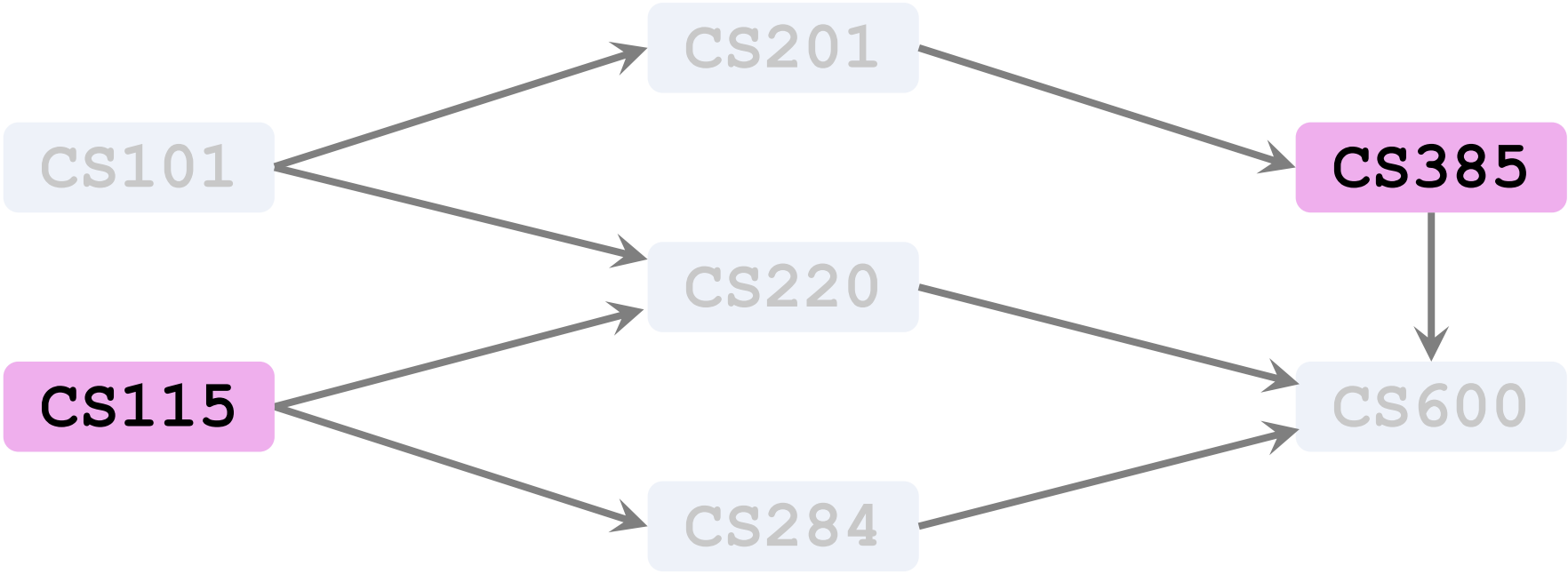
Example 3



Example 3

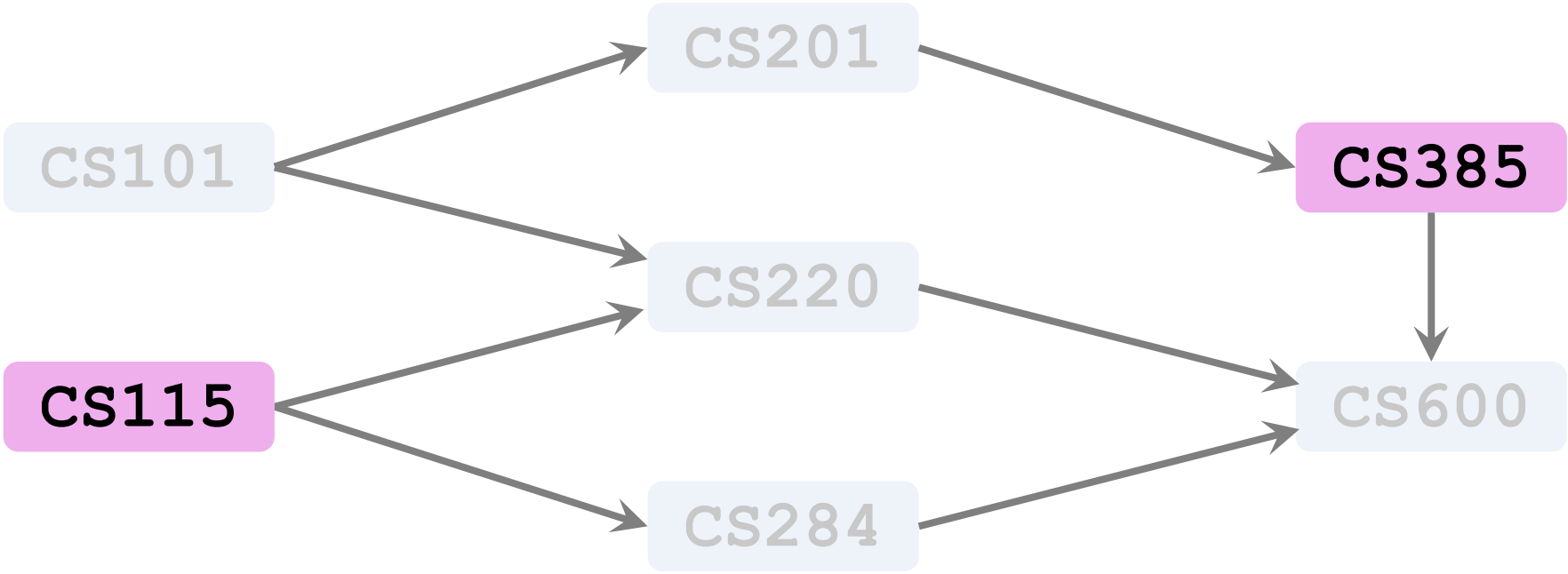


Example 3



CS101 CS115 CS201 CS220 CS284 CS385 CS600

Example 3



CS101 CS115 CS201 CS220 CS284 CS385 CS600

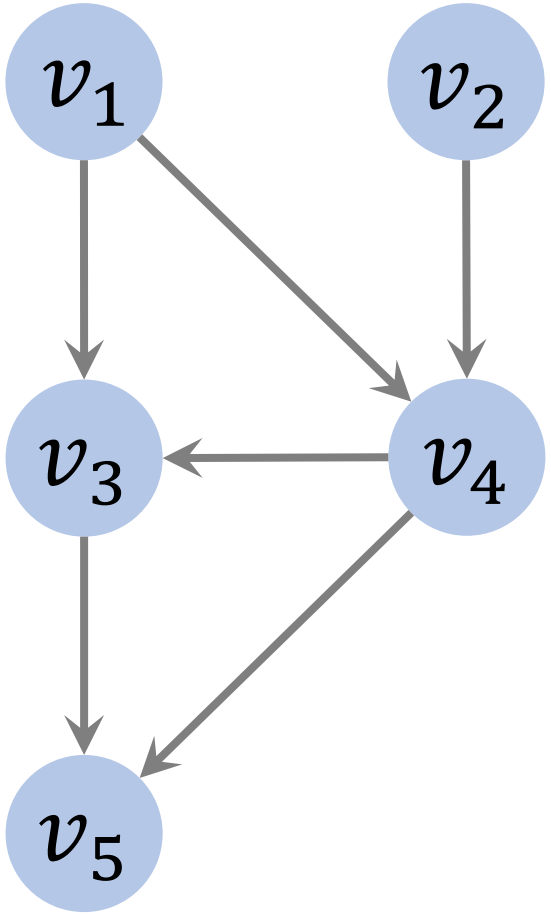
Basic Idea

Basic Idea

Repeat until the graph is empty:

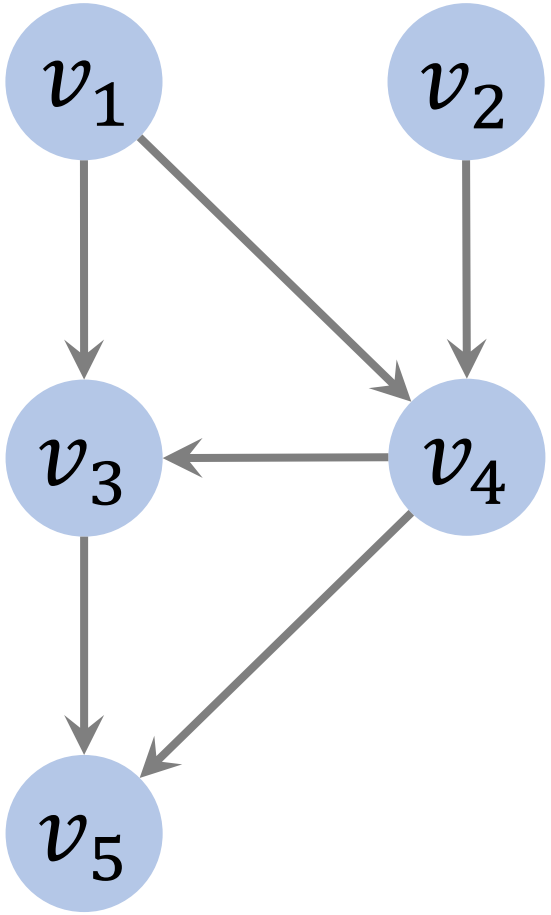
1. Identify a vertex with no incoming edges.
2. Add the vertex to the ordering.
3. Remove the vertex from the graph.

Initial State



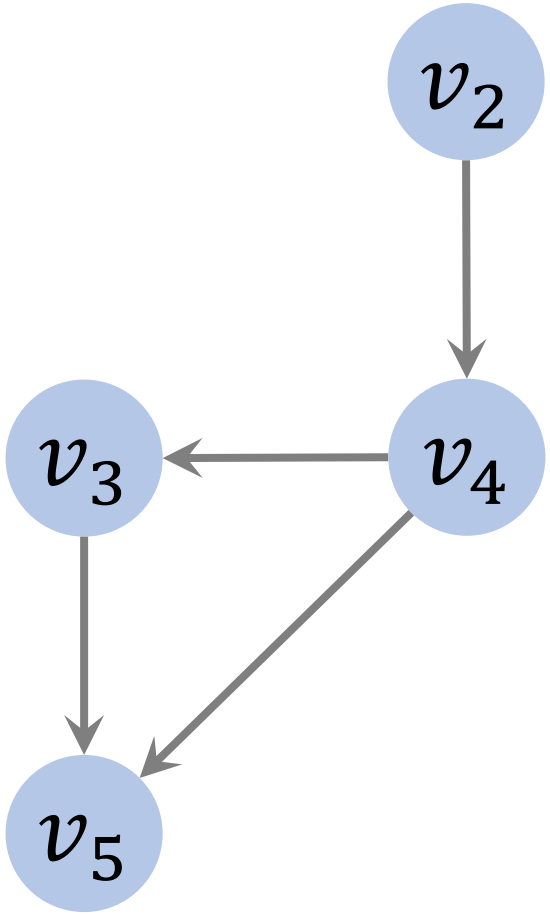
Ordering:

Iteration 1



Ordering:

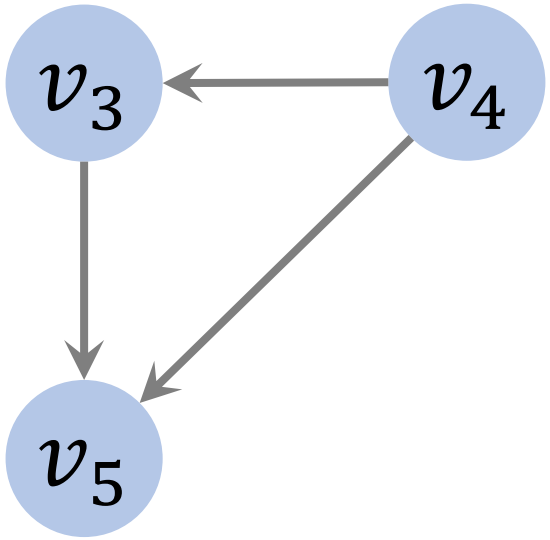
Iteration 2



Ordering:

v_1

Iteration 3



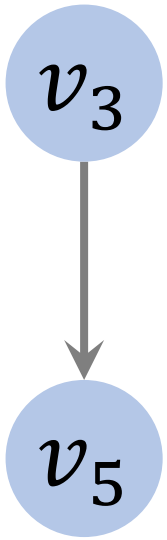
Ordering:

v_1 v_2

Iteration 4

Ordering:

v_1 v_2 v_4



Iteration 5

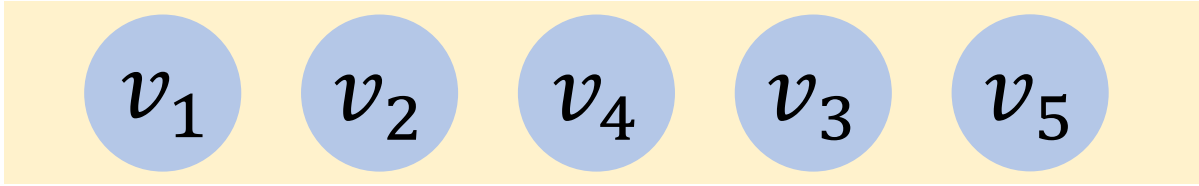
Ordering:

v_1 v_2 v_4 v_3

v_5

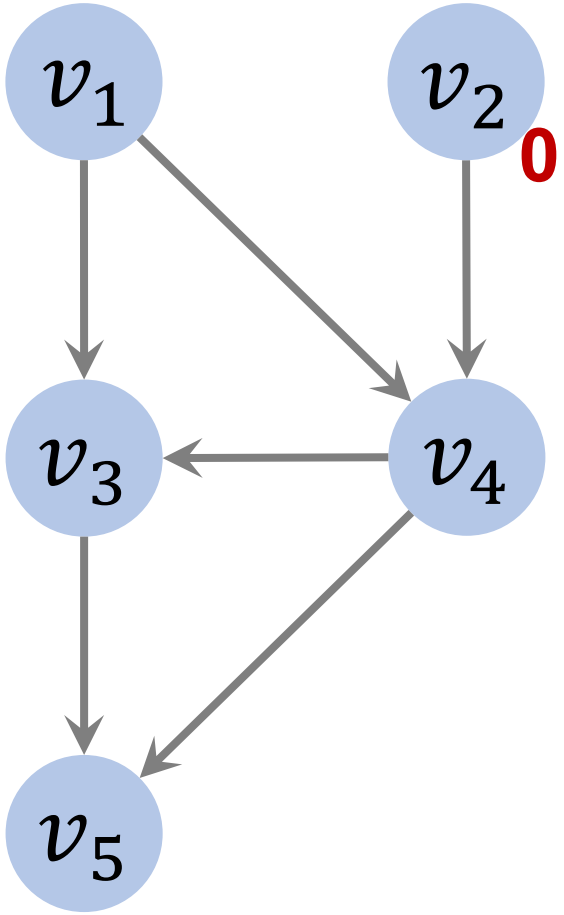
End of Procedure

Ordering:

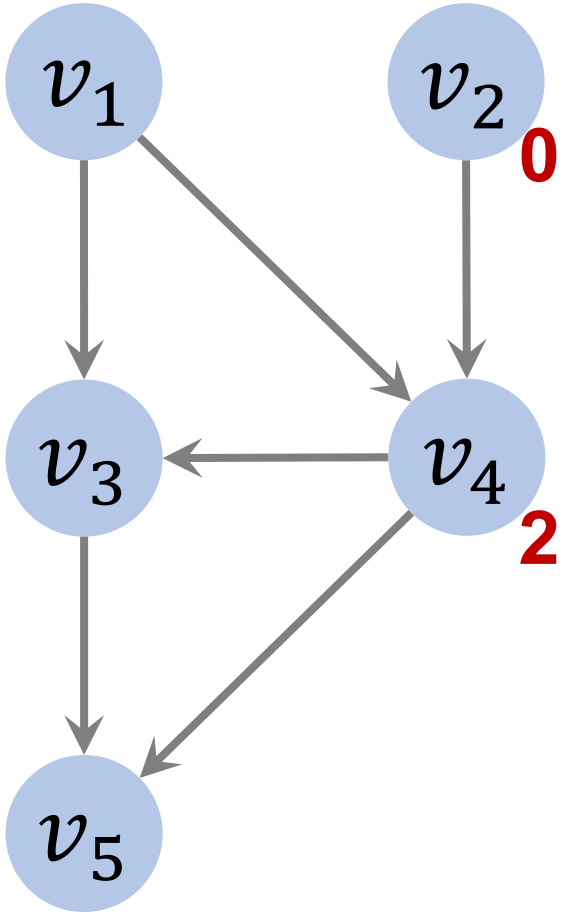


Algorithm

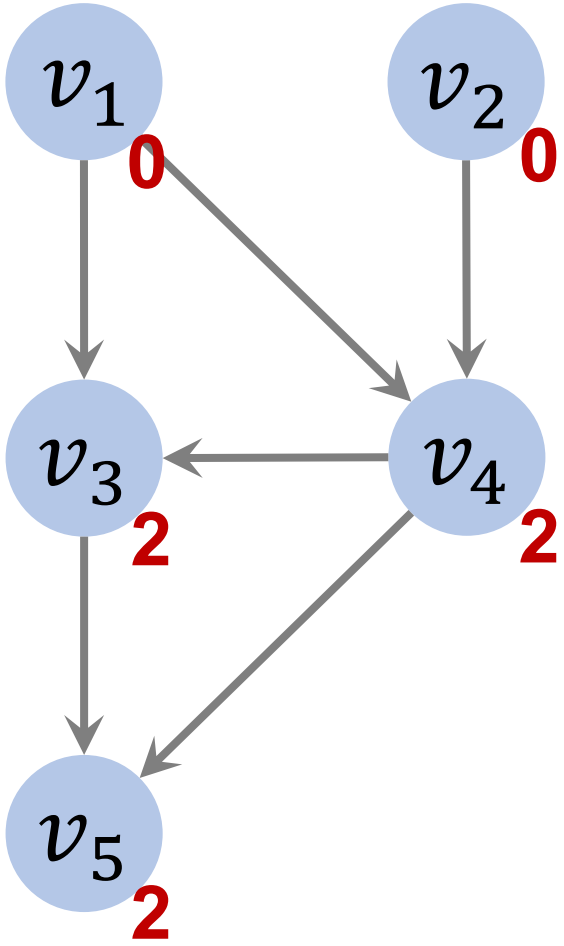
Indegree: number of incoming edges



Indegree: number of incoming edges

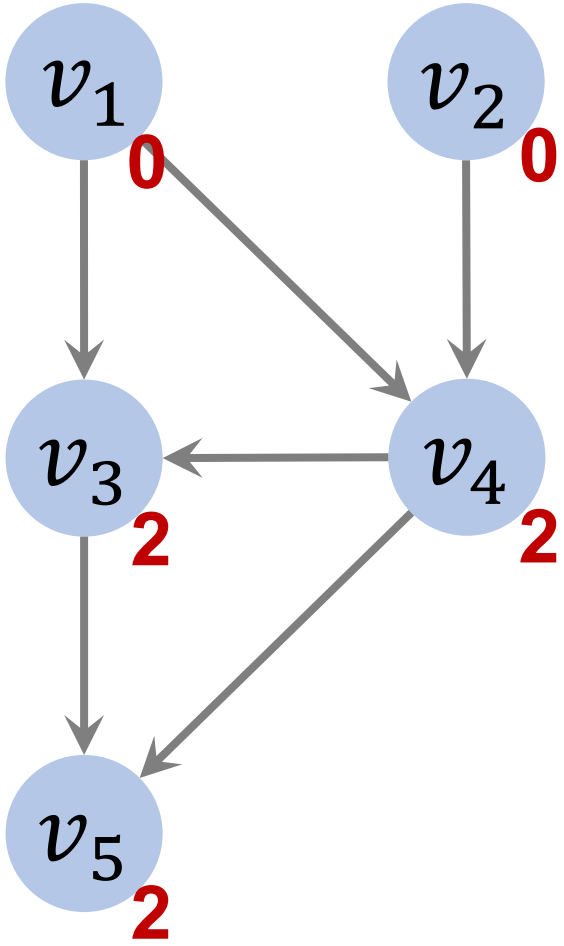


Indegree: number of incoming edges



- m: number of edges.
- Time complexity of counting all the indegrees is $O(m)$.

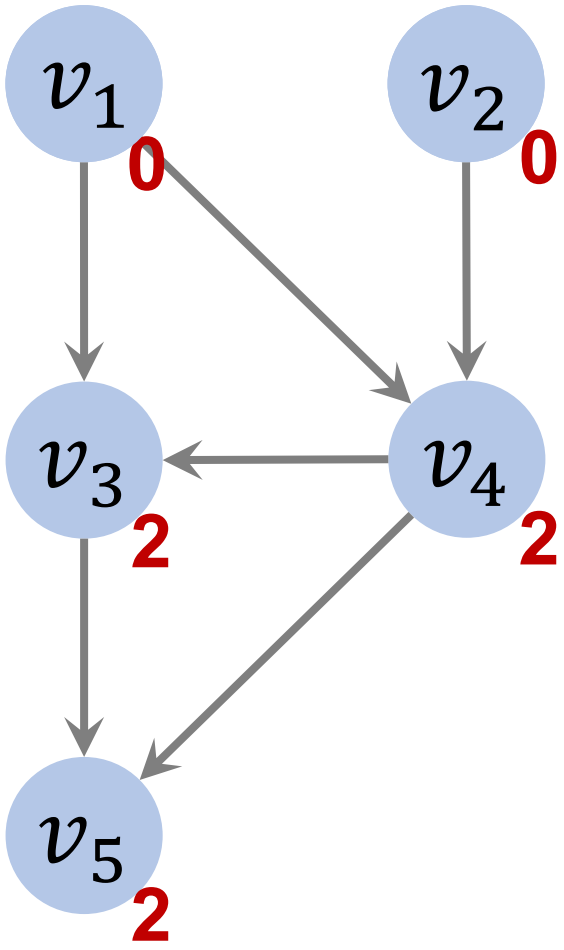
Initial State



Queue:

Ordering:

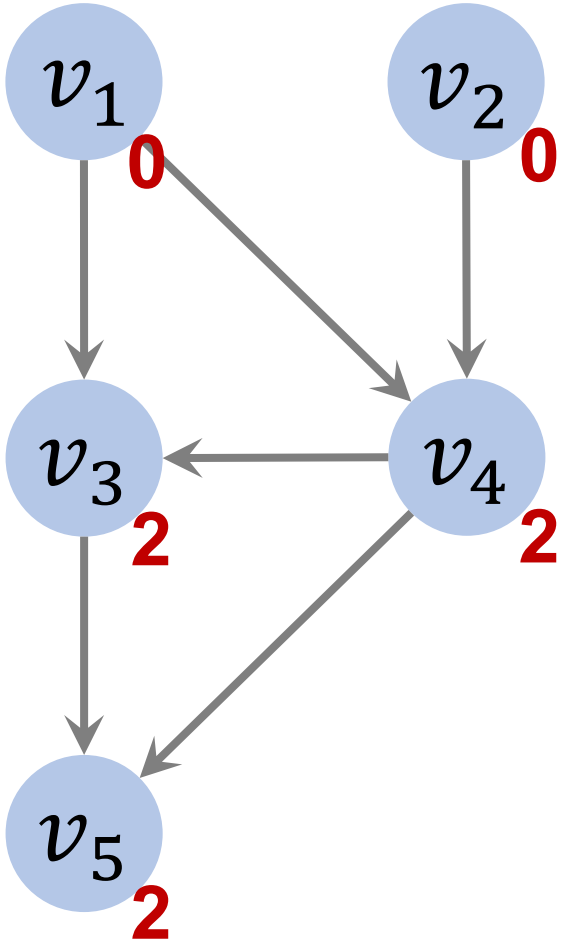
Enqueue the vertices with zero indegree



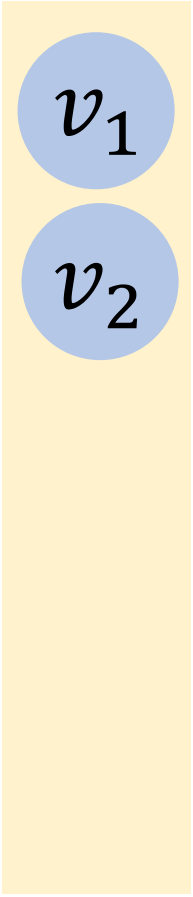
Queue:

Ordering:

Iteration 1

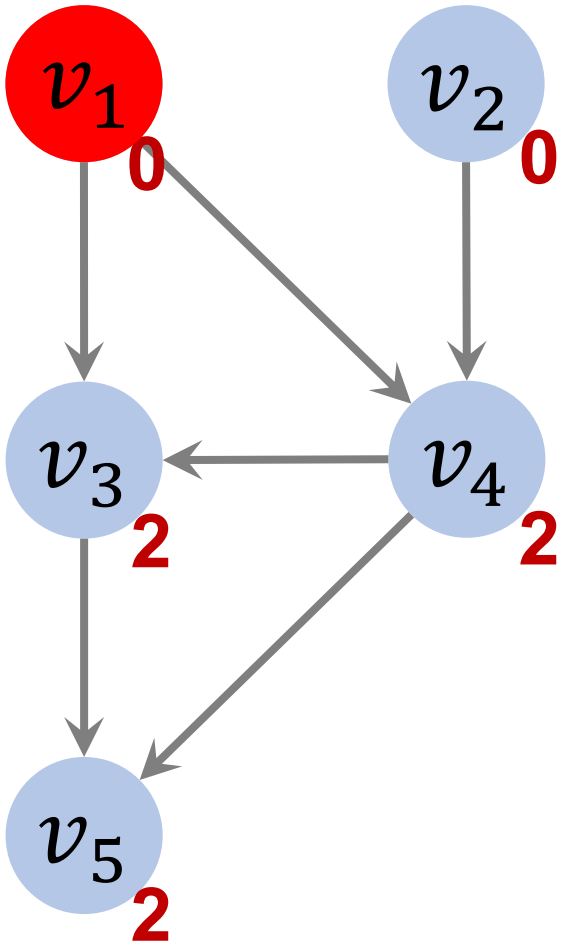


Queue:

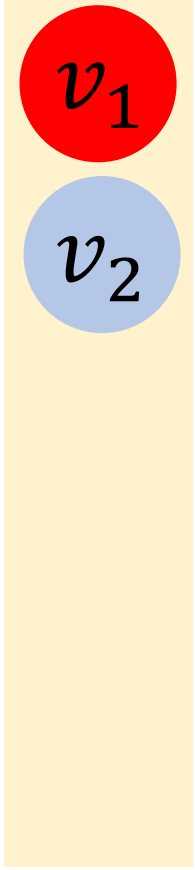


Ordering:

Iteration 1

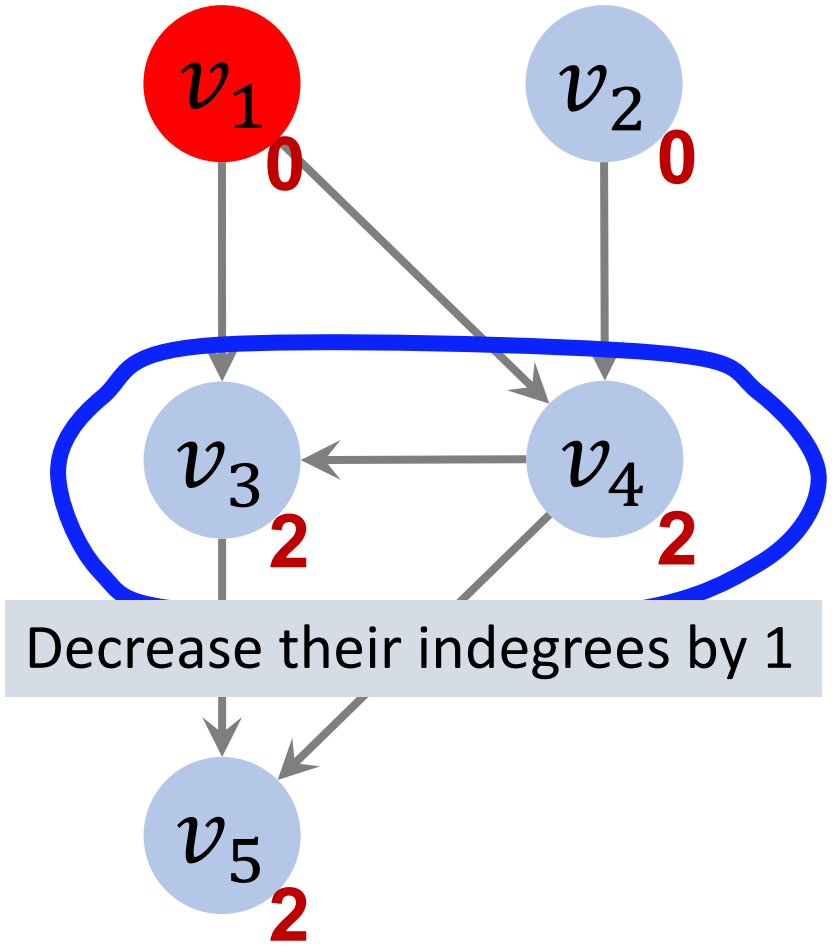


Queue:



Ordering:

Iteration 1



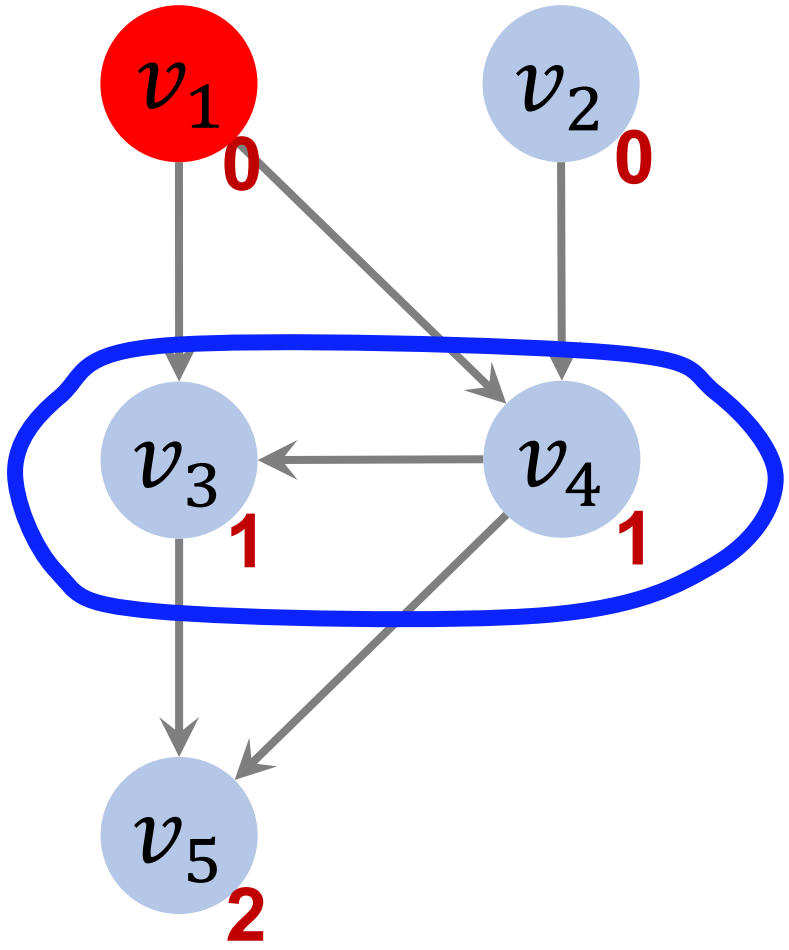
Queue:

v_2

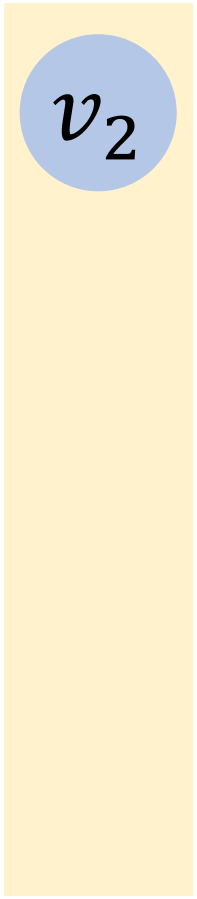
Ordering:

v_1

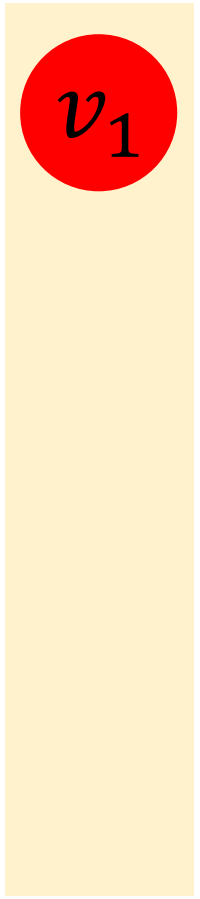
Iteration 1



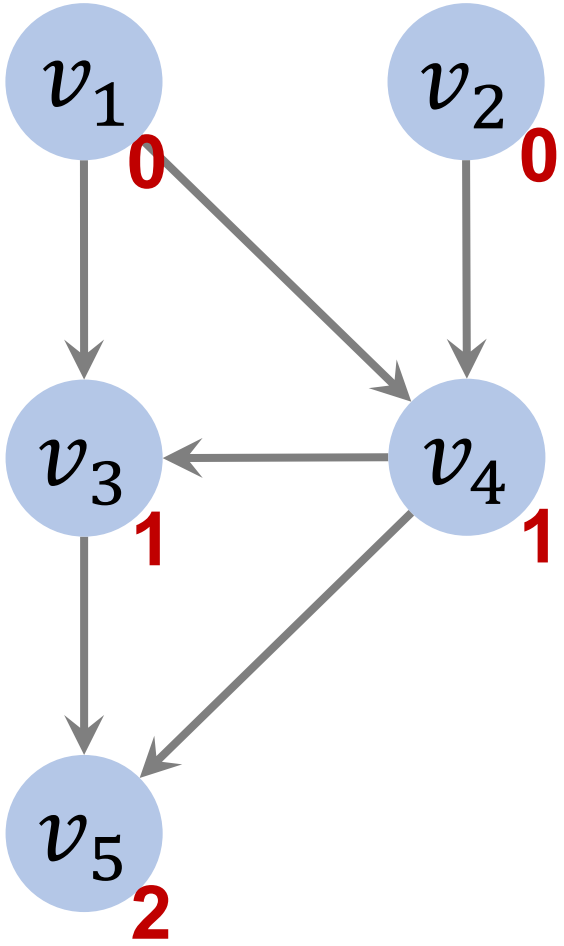
Queue:



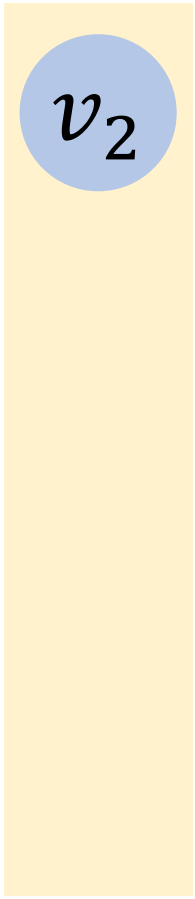
Ordering:



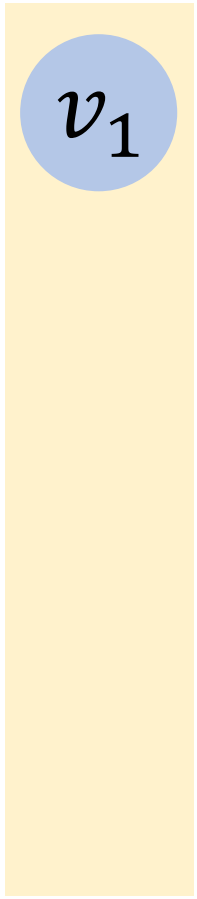
Iteration 2



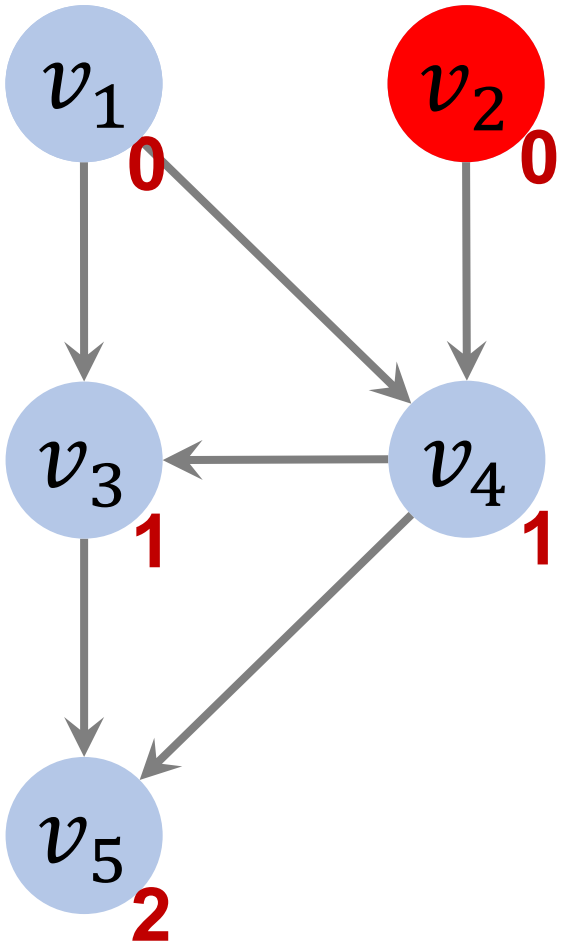
Queue:



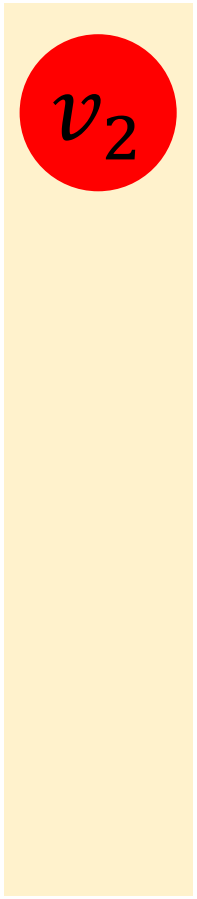
Ordering:



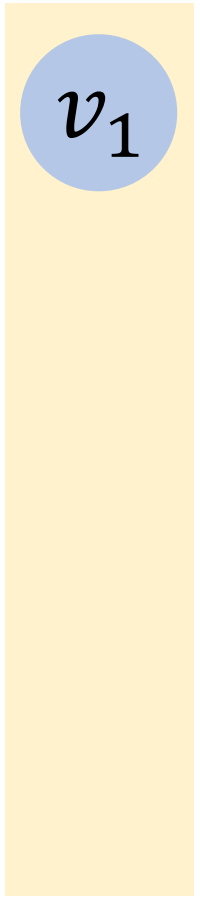
Iteration 2



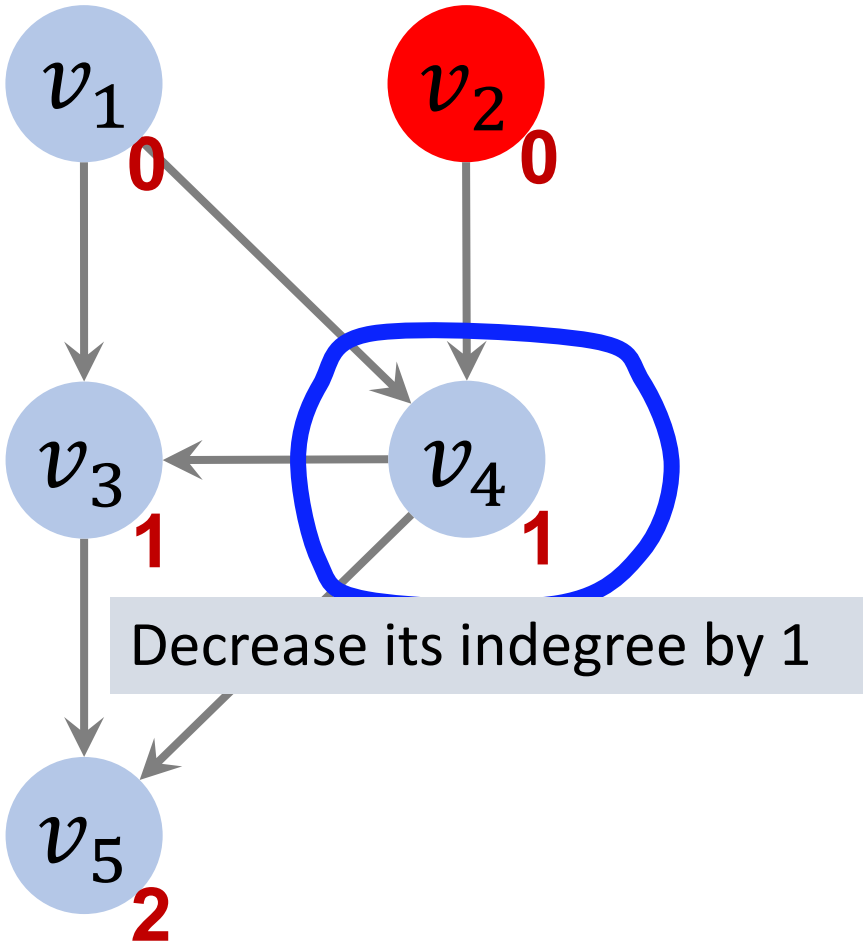
Queue:



Ordering:



Iteration 2



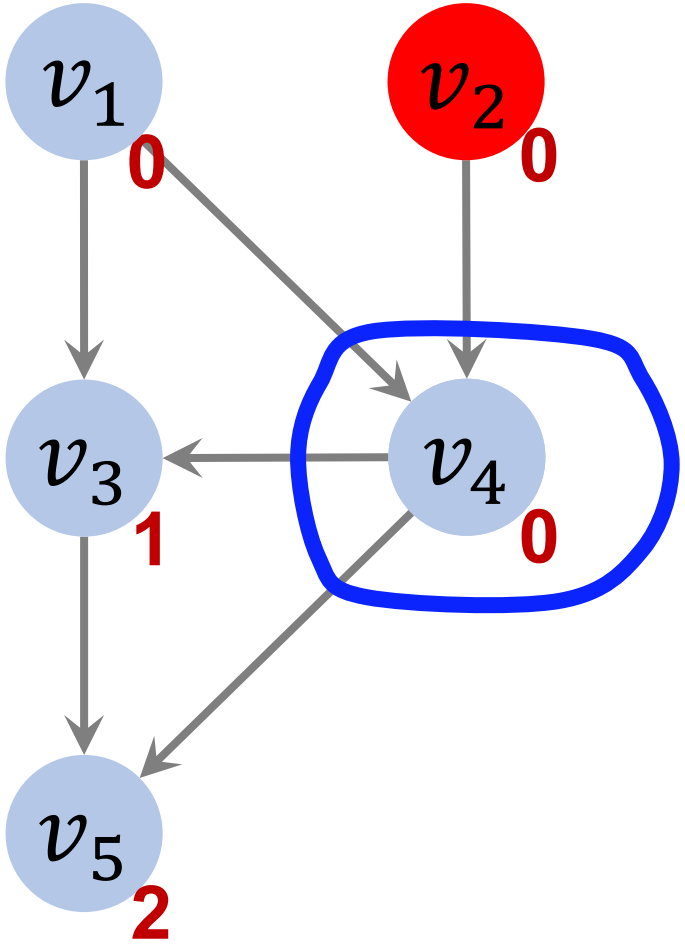
Queue:

Ordering:

v_1

v_2

Iteration 2



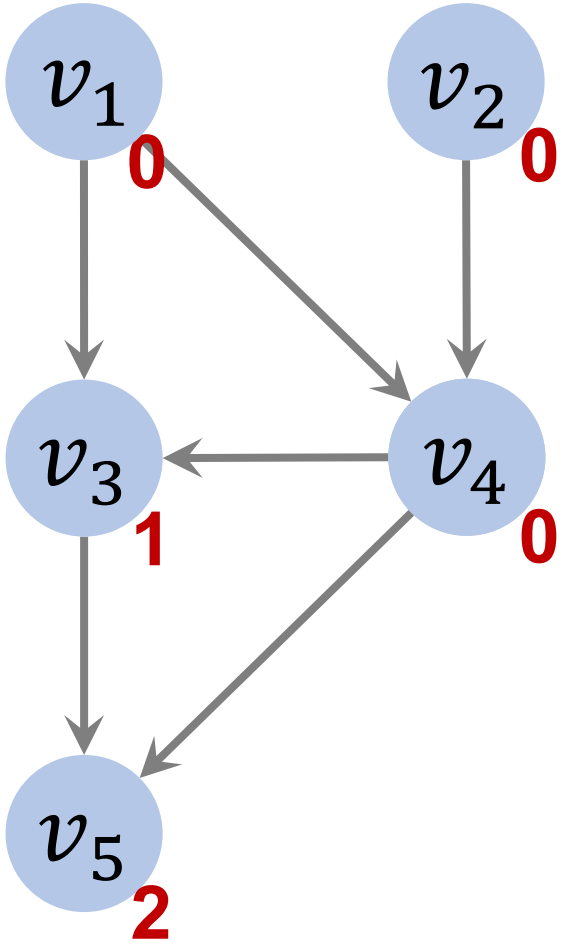
Queue:

Ordering:

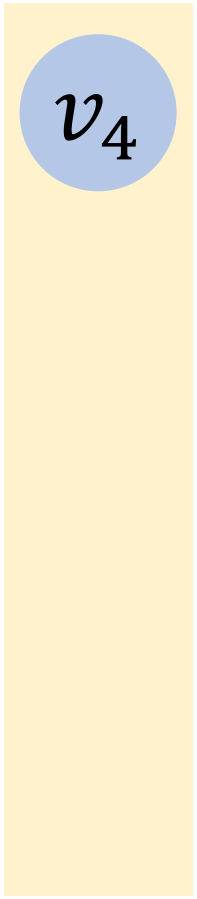
v_1

v_2

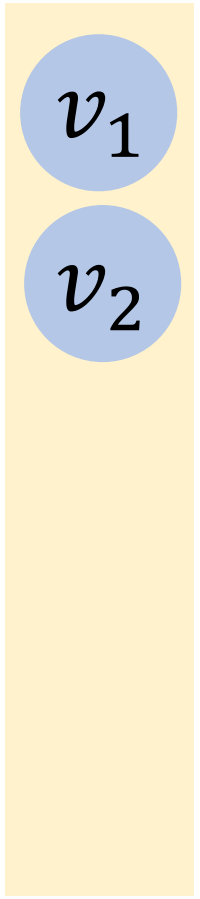
Iteration 3



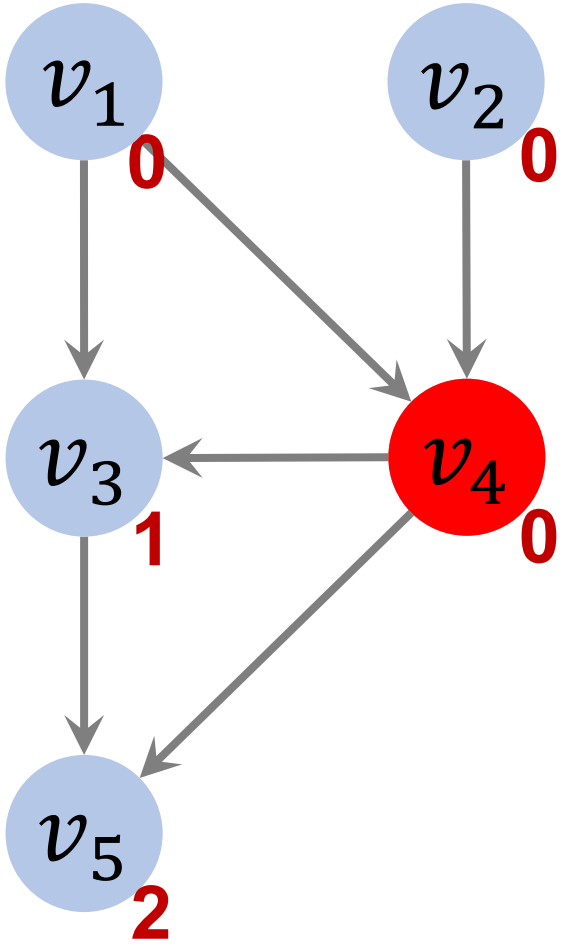
Queue:



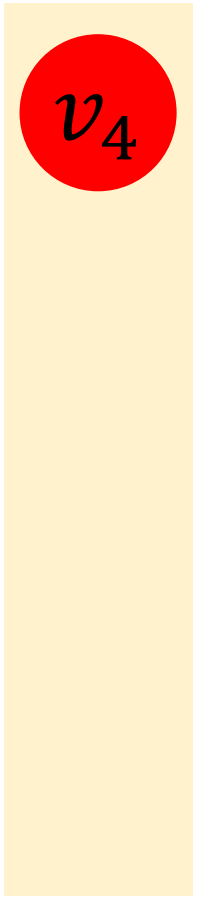
Ordering:



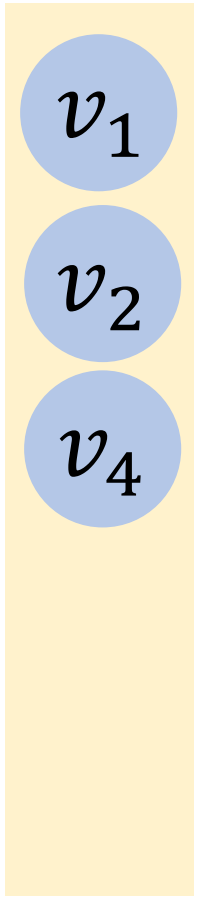
Iteration 3



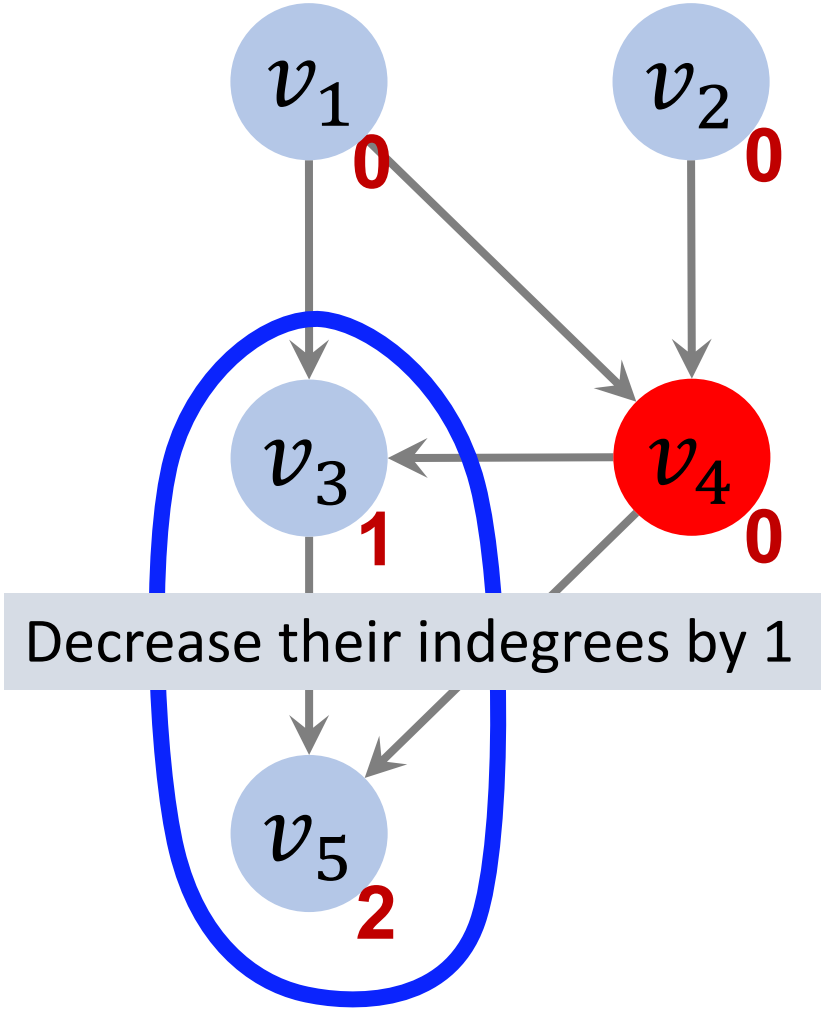
Queue:



Ordering:



Iteration 3



Queue:

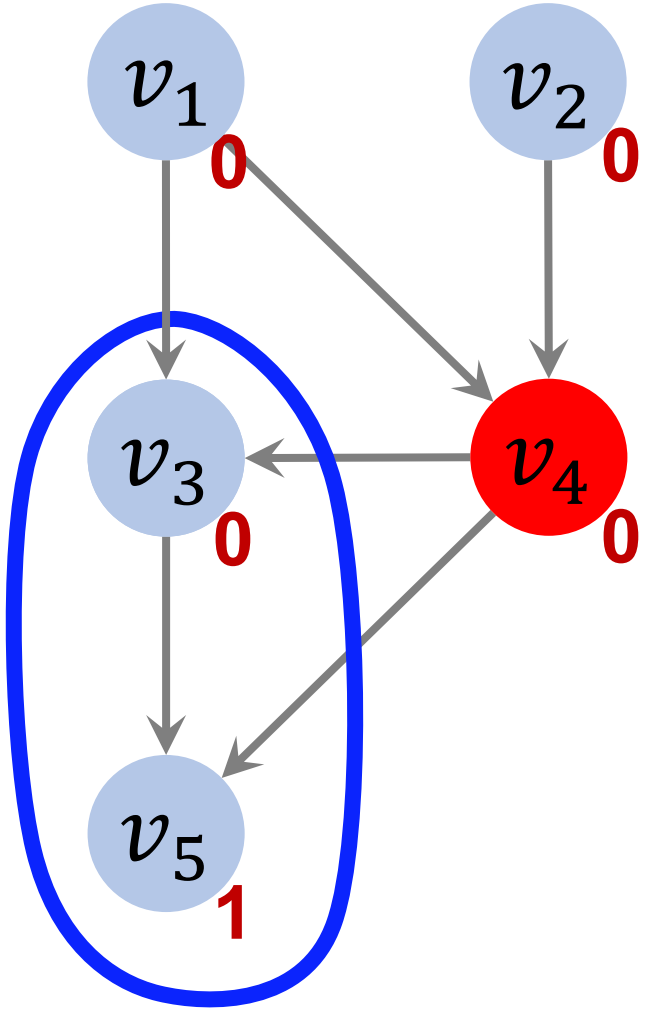
Ordering:

v_1

v_2

v_4

Iteration 3



Queue:

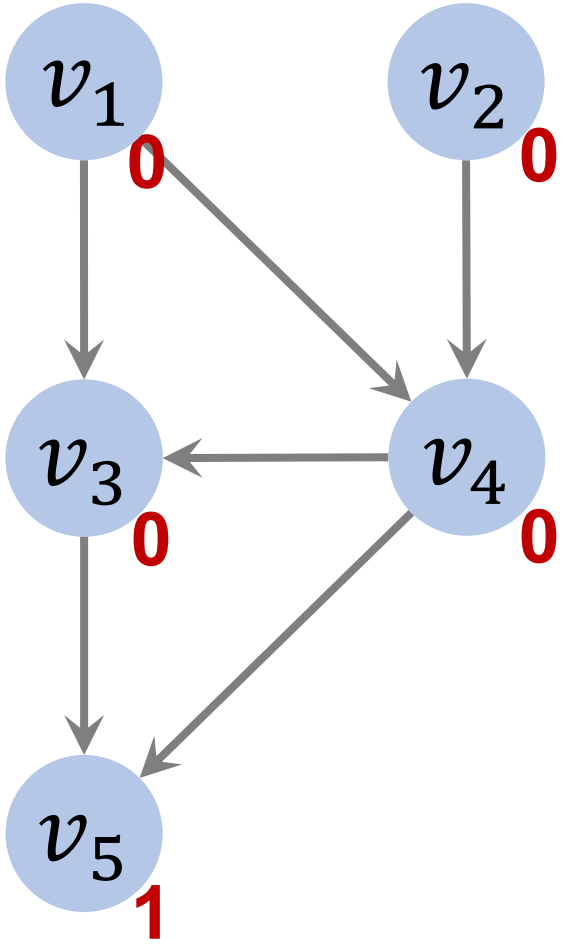
Ordering:

v_1

v_2

v_4

Iteration 4



Queue:

v_3

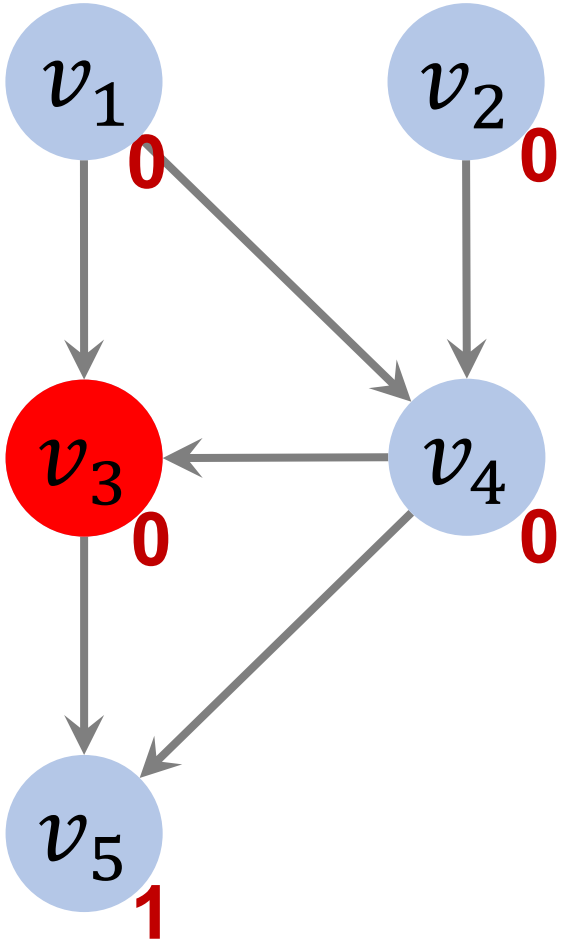
Ordering:

v_1

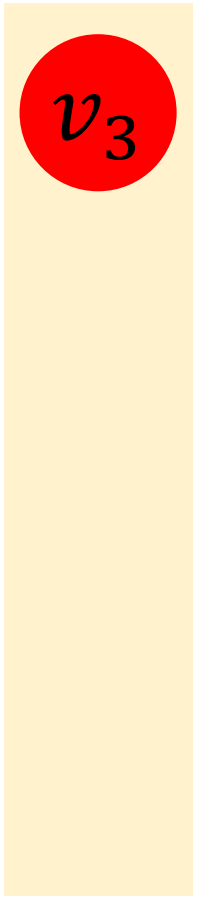
v_2

v_4

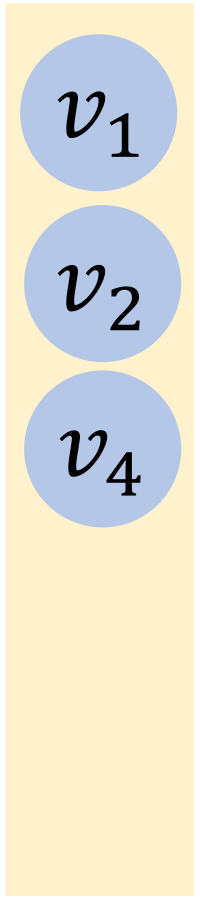
Iteration 4



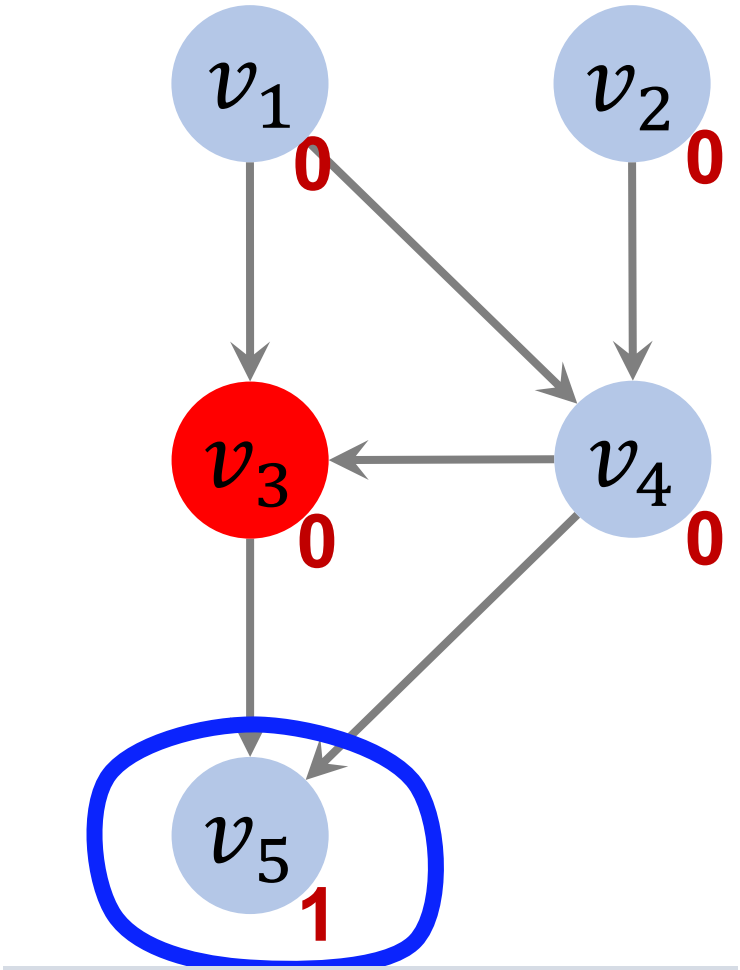
Queue:



Ordering:



Iteration 4



Decrease its indegree by 1

Queue:

Ordering:

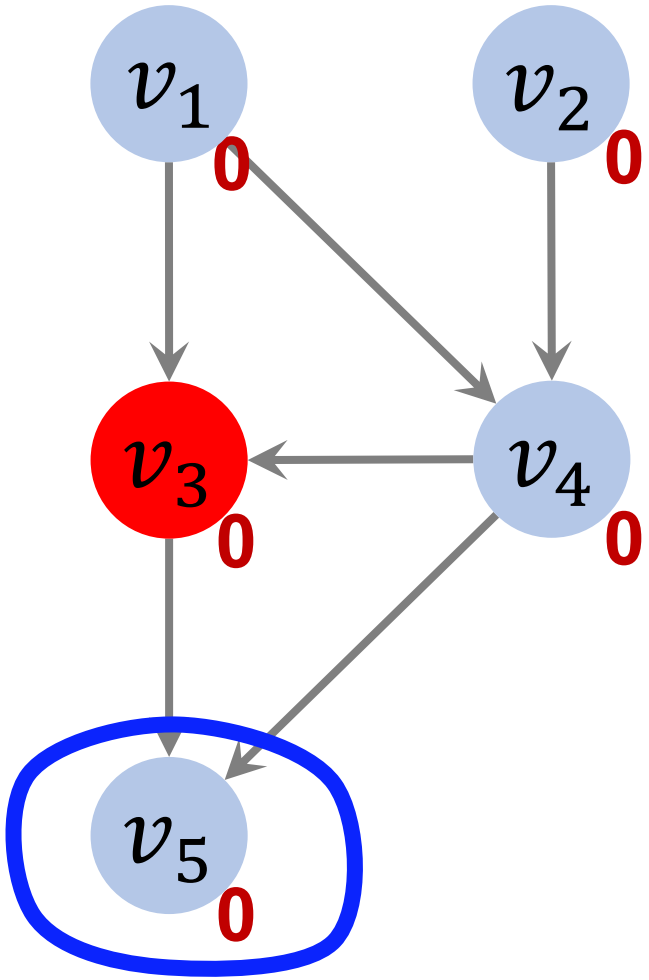
v_1

v_2

v_4

v_3

Iteration 4



Queue:

Ordering:

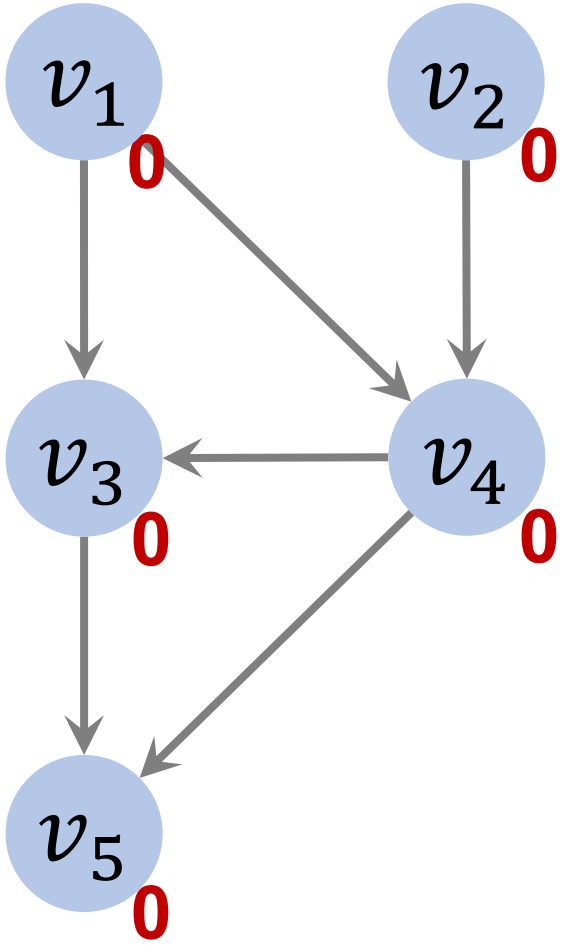
v_1

v_2

v_4

v_3

Iteration 5



Queue:

v_5

Ordering:

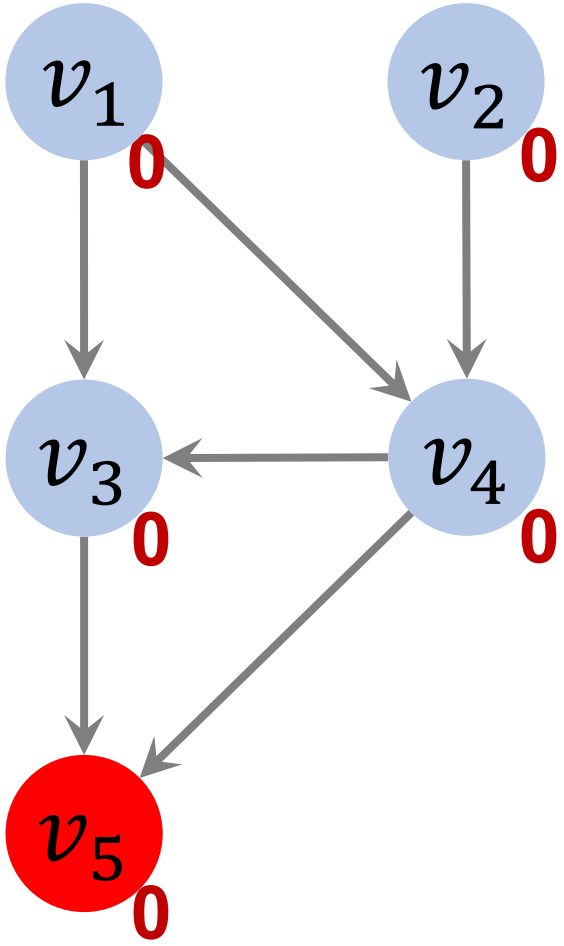
v_1

v_2

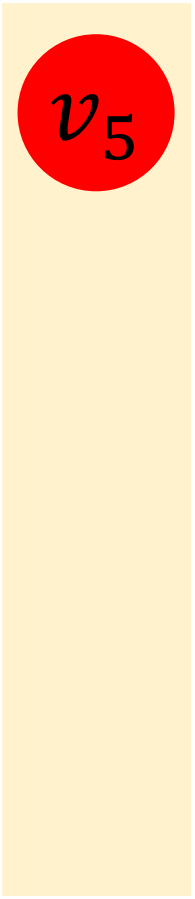
v_4

v_3

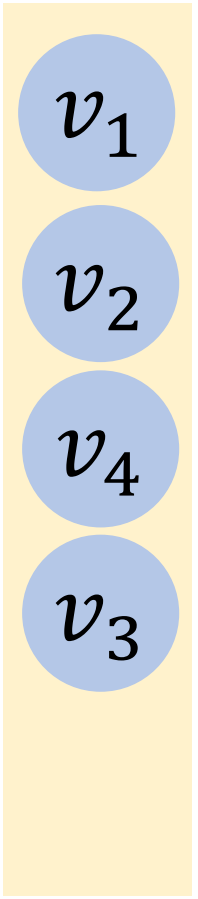
Iteration 5



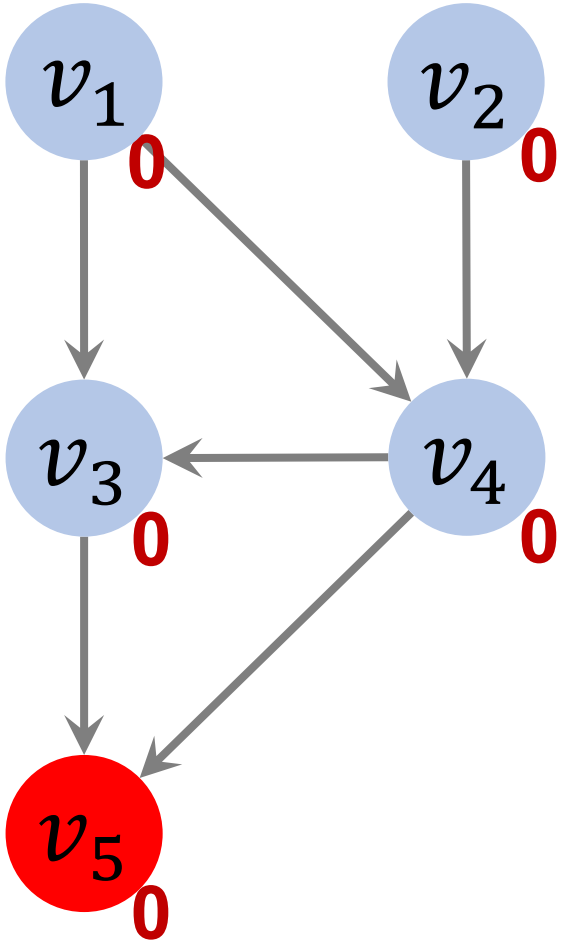
Queue:



Ordering:



Iteration 5



Queue:

Ordering:

v_1

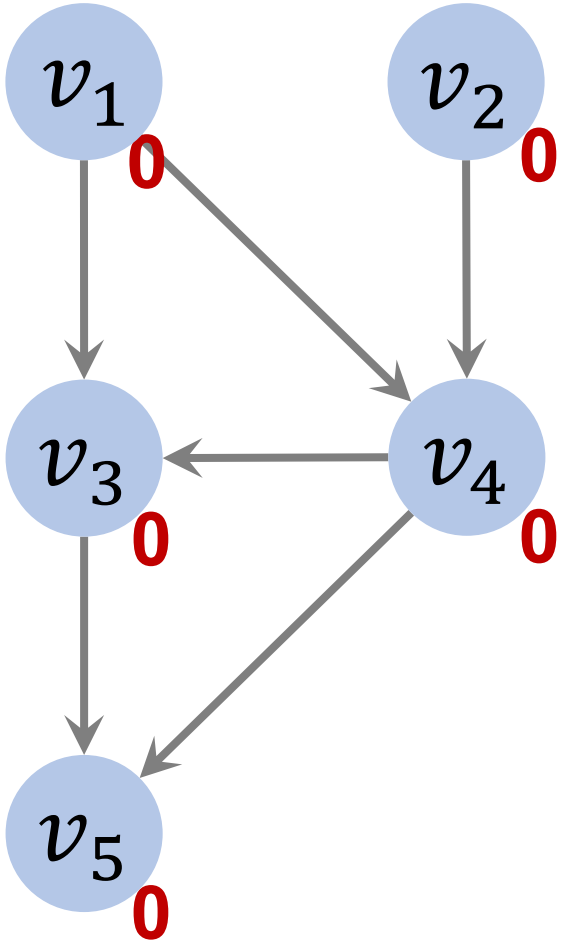
v_2

v_4

v_3

v_5

End of Procedure



Queue:

Ordering:

v_1

v_2

v_4

v_3

v_5

Dealing with Cyclic Graphs

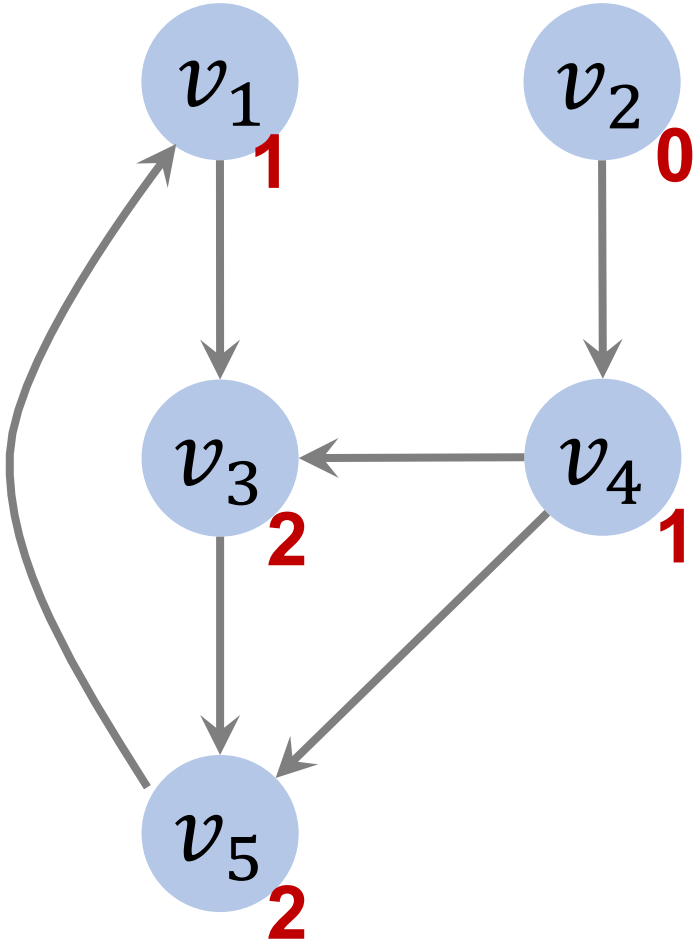
What if the graph has cycles?

- After emptying the queue, we check if

$$\#\text{Iterations} = \#\text{Vertices}.$$

- Equal ➔ No cycle.
- Unequal ➔ Cycle.

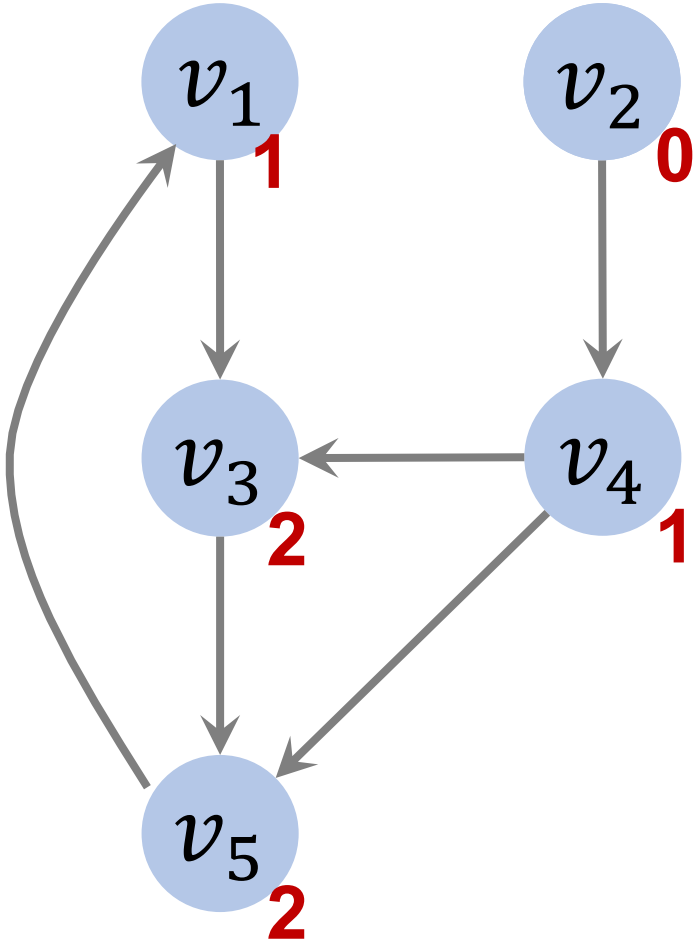
Example



Queue:

Ordering:

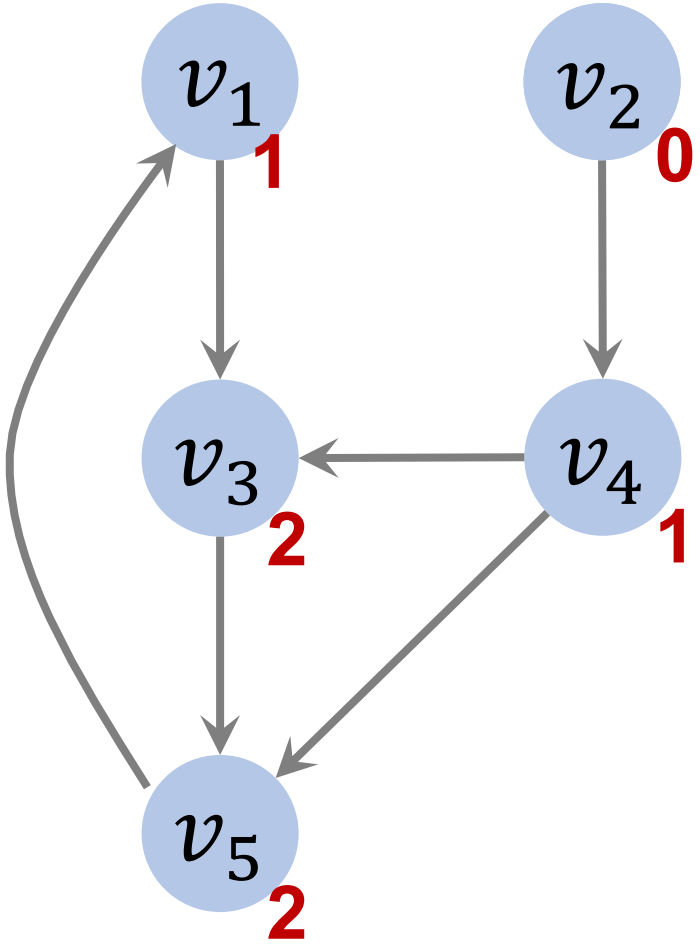
Initialization



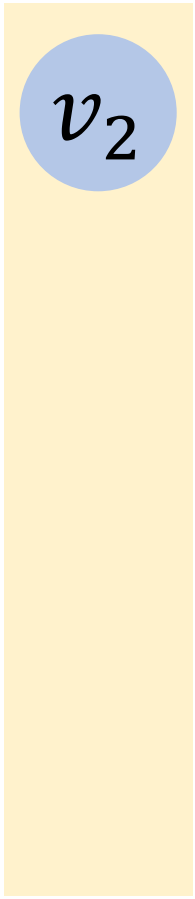
Queue:

Ordering:

Iteration 1

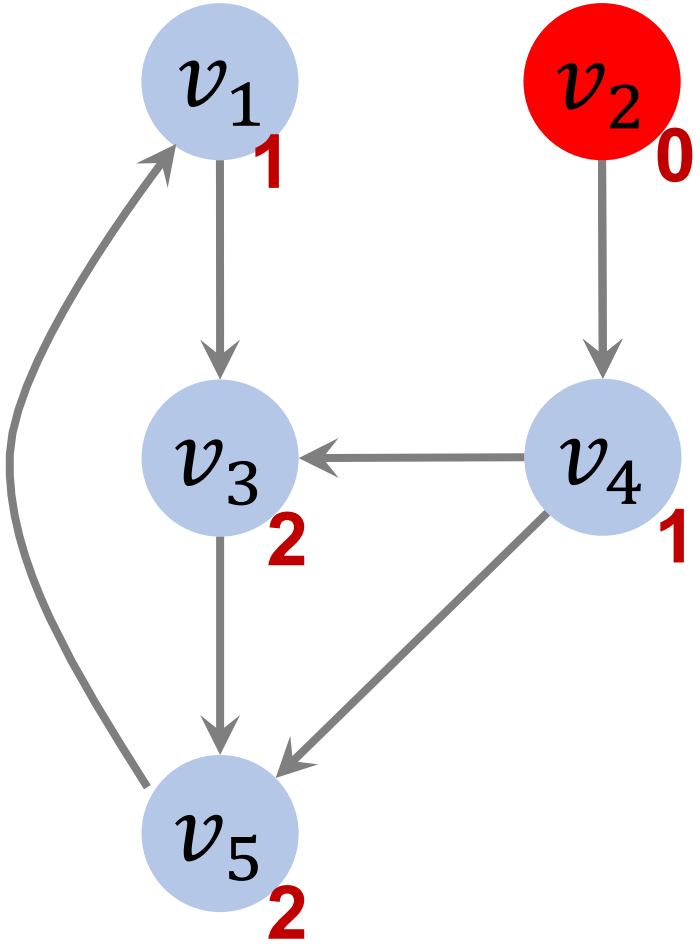


Queue:

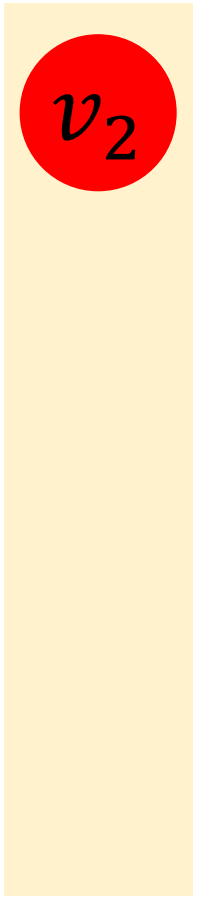


Ordering:

Iteration 1

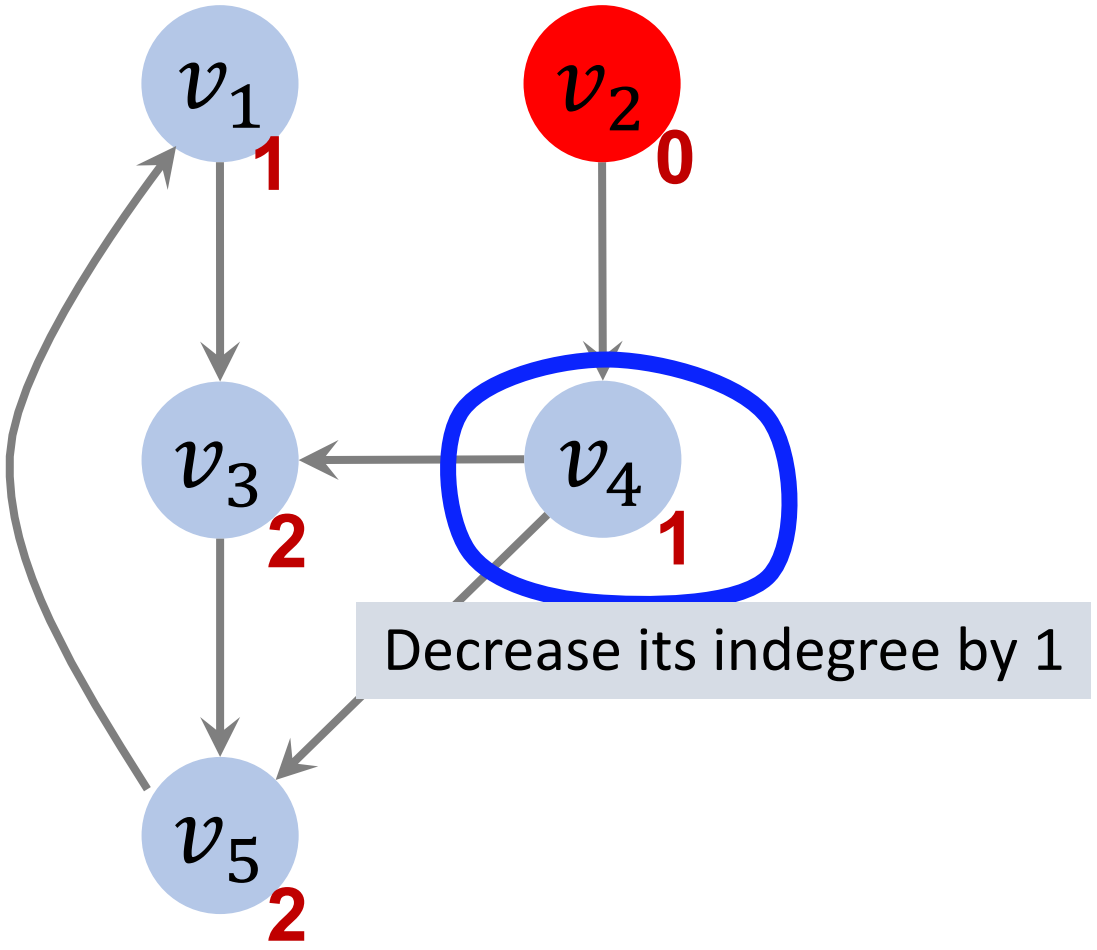


Queue:



Ordering:

Iteration 1

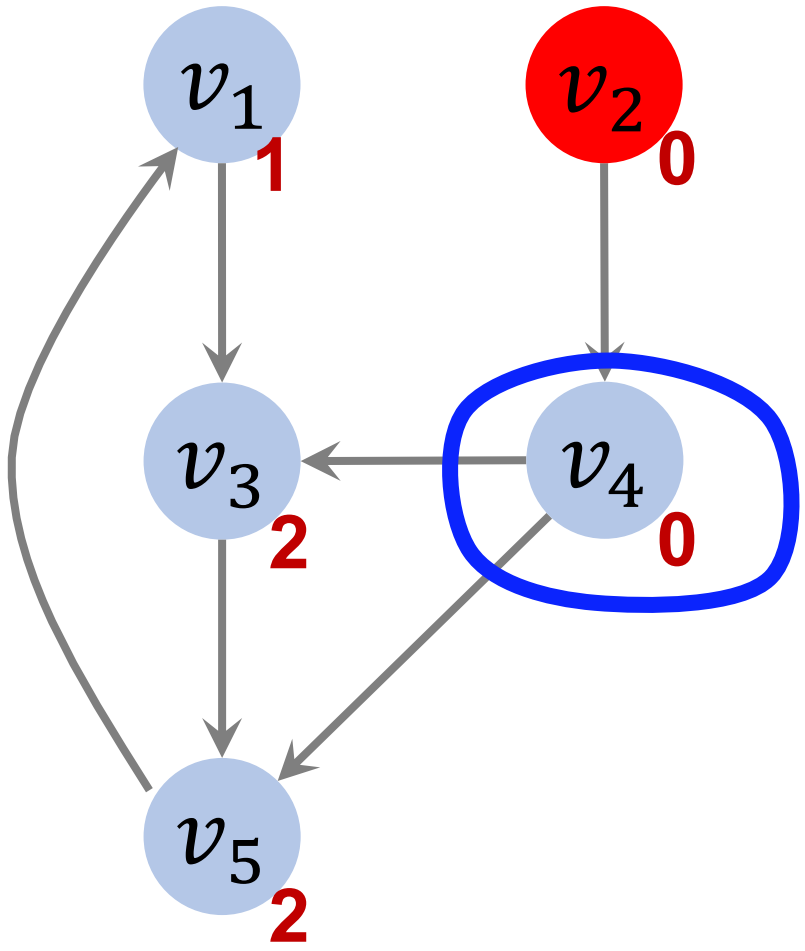


Queue:

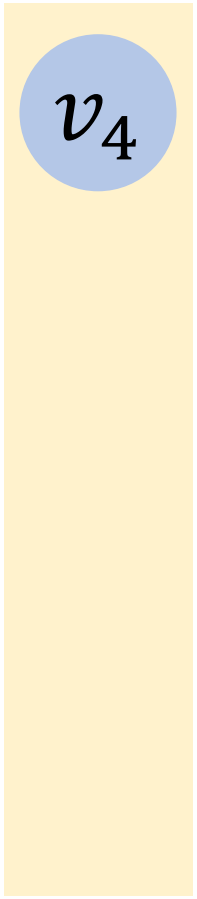
Ordering:

v_2

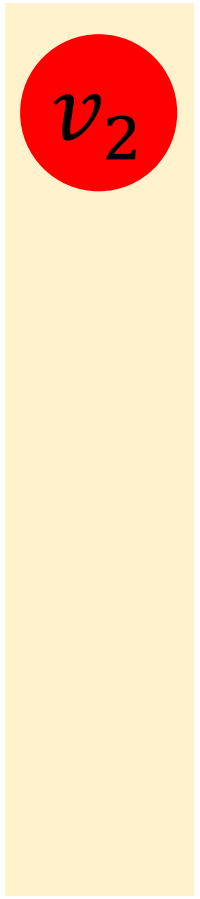
Iteration 1



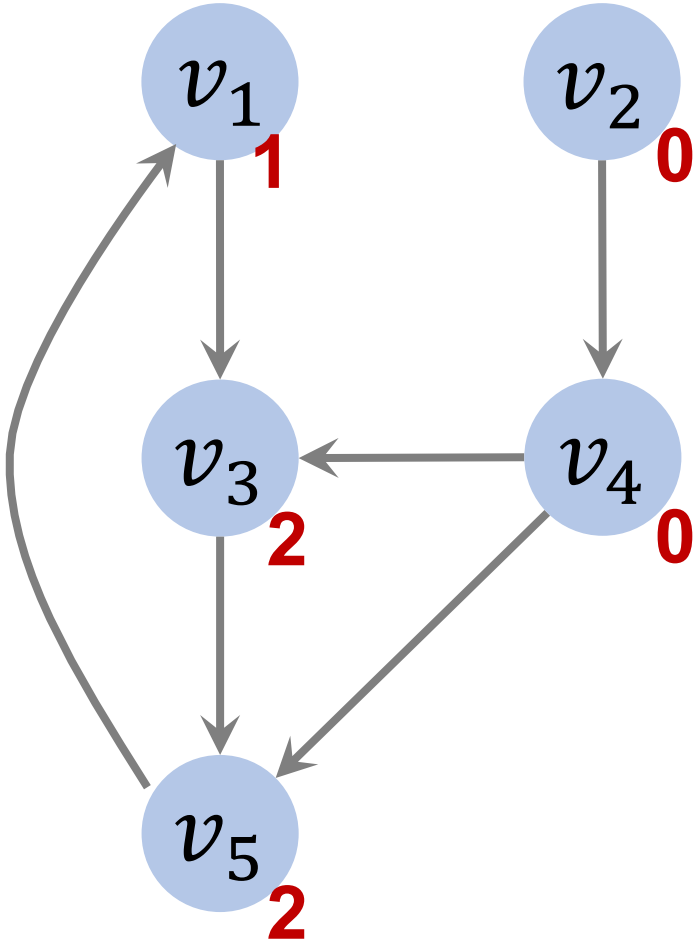
Queue:



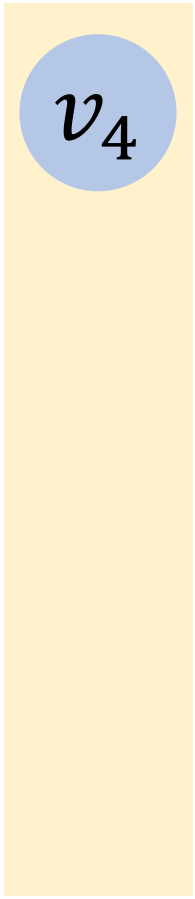
Ordering:



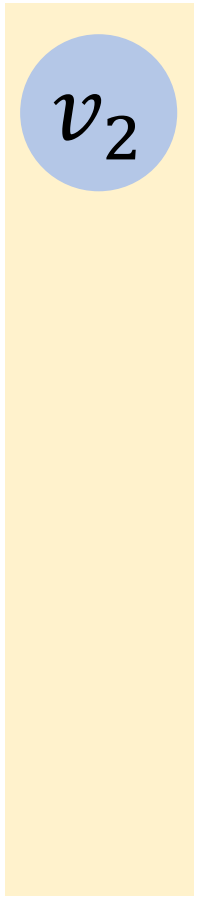
Iteration 2



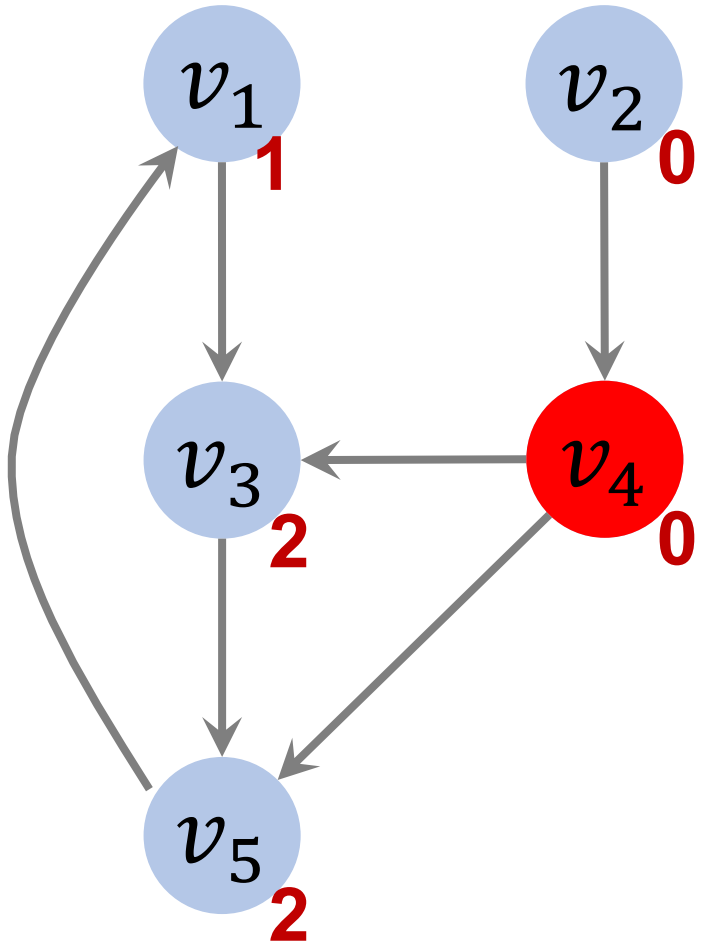
Queue:



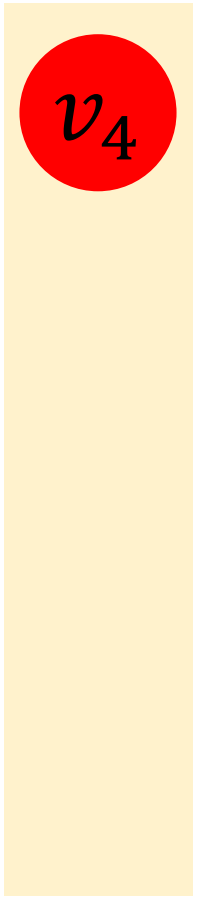
Ordering:



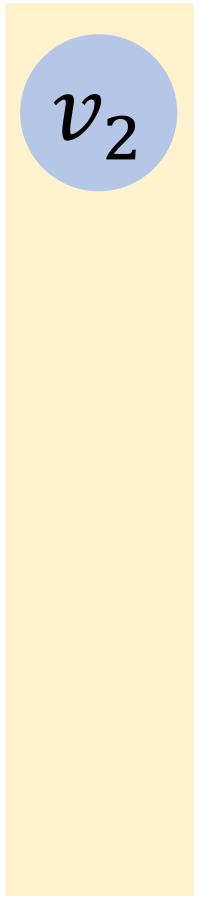
Iteration 2



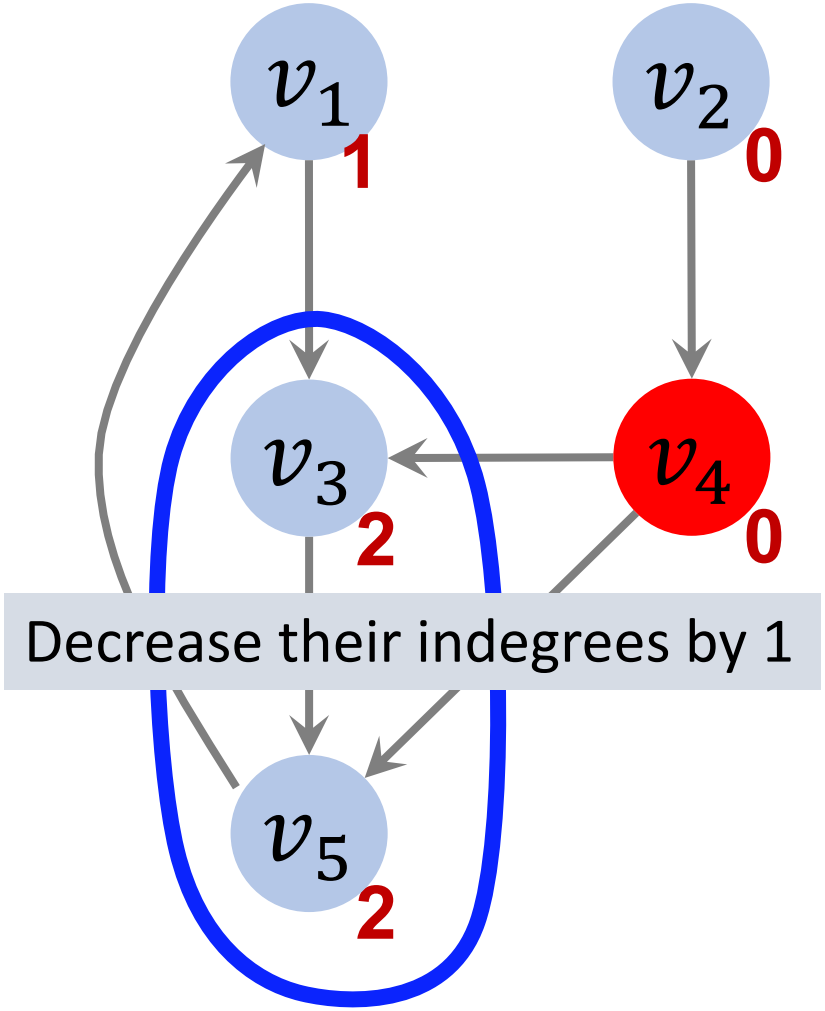
Queue:



Ordering:



Iteration 2



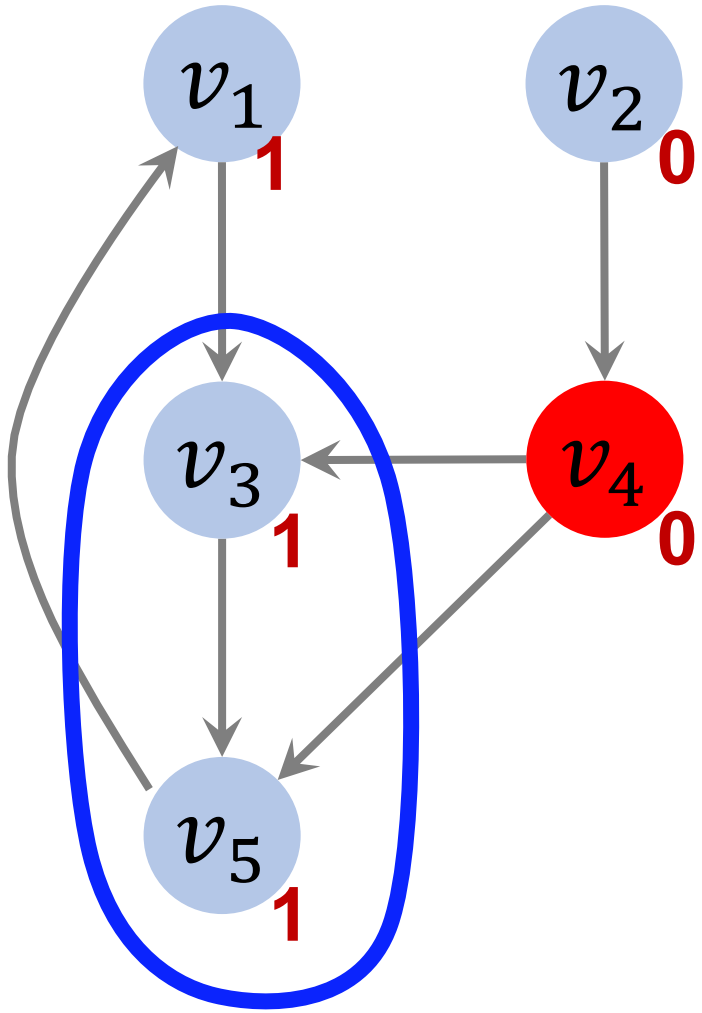
Queue:

Ordering:

v_2

v_4

Iteration 2



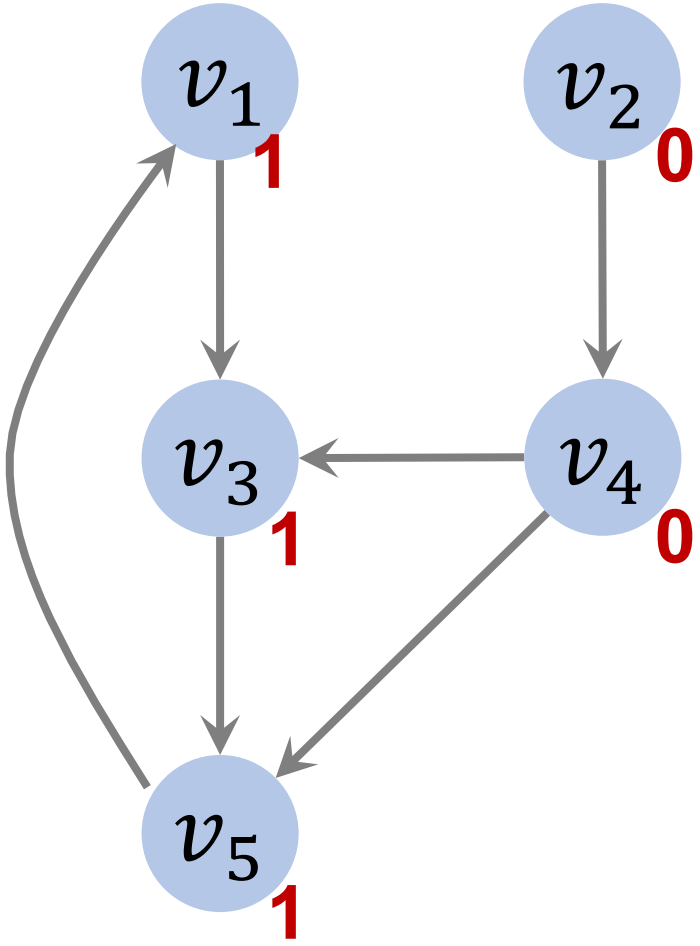
Queue:

Ordering:

v_2

v_4

End of Procedure



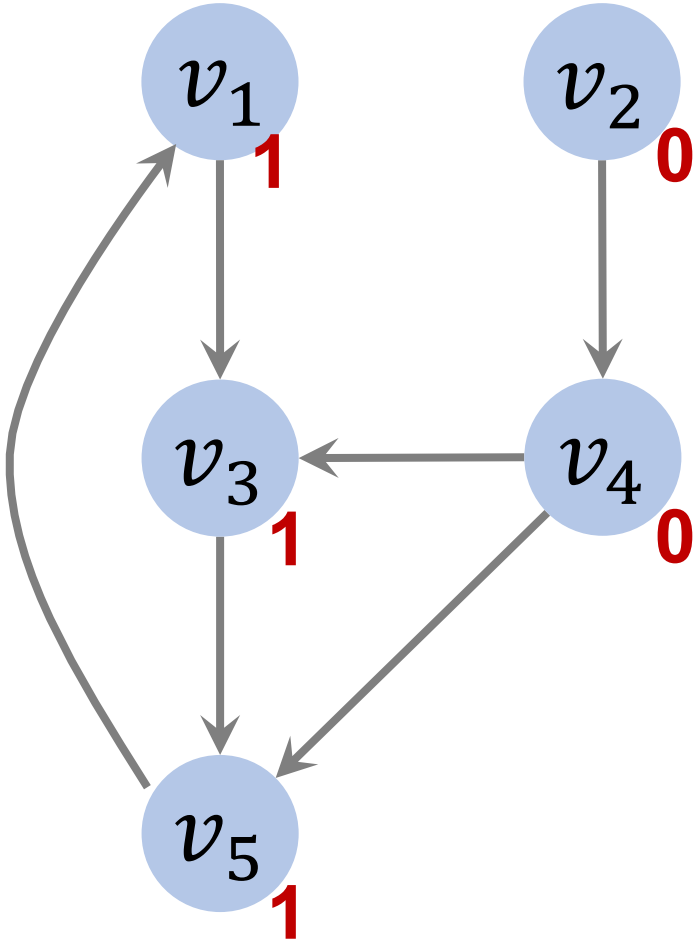
Queue:

Ordering:

v_2

v_4

End of Procedure



- #Iterations is 2.
- #Vertices is 5.
- #Iterations $\neq$ #Vertices.
- There is at least one cycle.
- The vertices cannot be sorted.

Time Complexity

Time Complexity

- m: number of edges.
- n: number of vertices.

Time Complexity

- m: number of edges.
- n: number of vertices.
- Calculating all the indegrees. ➔ $O(m)$ time .

Time Complexity

- m: number of edges.
- n: number of vertices.
- Calculating all the indegrees. ➔ $O(m)$ time .
- Every vertex is enqueued and dequeue once. ➔ $O(n)$ time.
- Every edge is touched only once (decrease it neighbors' indegrees.) ➔ $O(m)$ time.

Overall time complexity: $O(m + n)$.

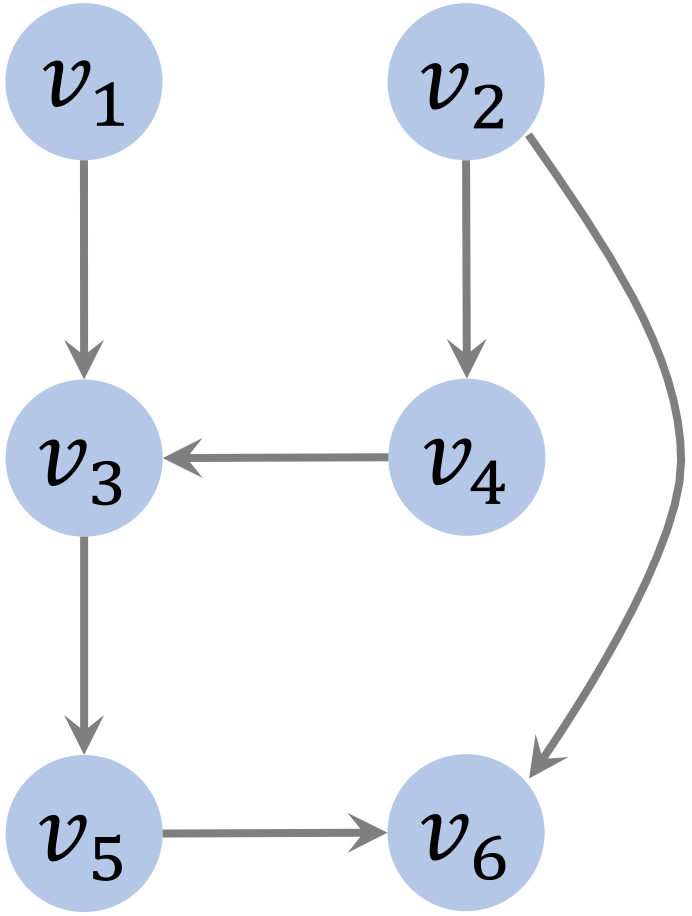
Time Complexity

- m: number of edges.
- n: number of vertices.
- Calculating all the indegrees. ➔ $O(m)$ time .
- Every vertex is enqueued and dequeue once. ➔ $O(n)$ time.
- Every edge is touched only once (decrease it neighbors' indegrees.) ➔ $O(m)$ time.

Overall time complexity: $O(m + n)$.

Question

Question 1



Which can the results of topological sort?
(There may be multiple correct orderings.)

A. $v_1, v_2, v_3, v_4, v_5, v_6.$

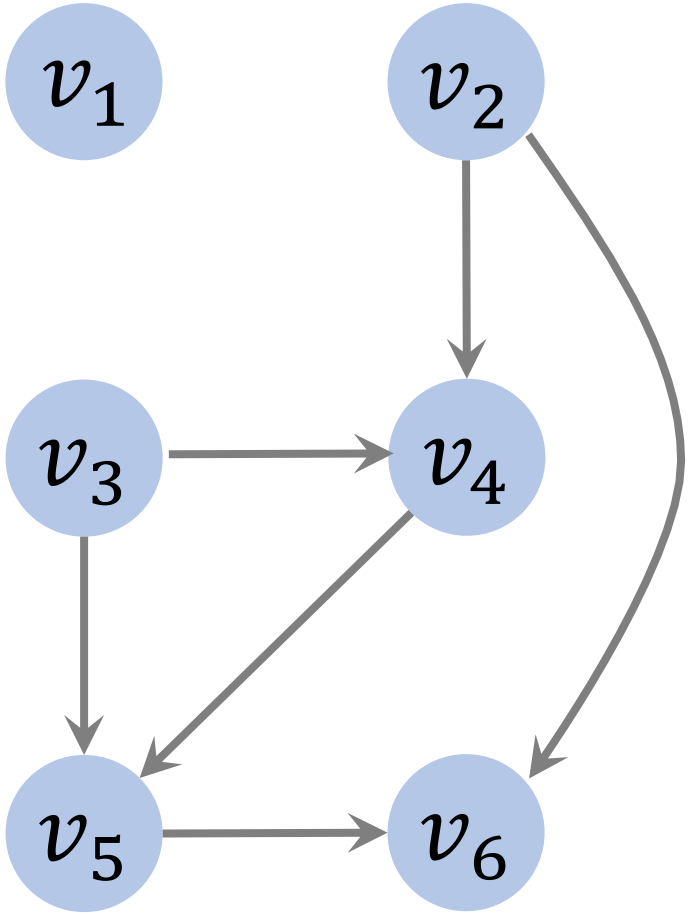
B. $v_1, v_2, v_4, v_3, v_5, v_6.$

C. $v_1, v_3, v_2, v_4, v_5, v_6.$

D. $v_2, v_1, v_4, v_3, v_5, v_6.$

E. $v_2, v_4, v_1, v_3, v_5, v_6.$

Question 2



Which can the results of topological sort?
(There may be multiple correct orderings.)

A. $v_1, v_2, v_3, v_4, v_5, v_6.$

B. $v_1, v_2, v_4, v_3, v_5, v_6.$

C. $v_1, v_3, v_2, v_4, v_5, v_6.$

D. $v_2, v_1, v_4, v_3, v_5, v_6.$

E. $v_3, v_1, v_2, v_4, v_5, v_6.$

Thank You!