Topological Sort

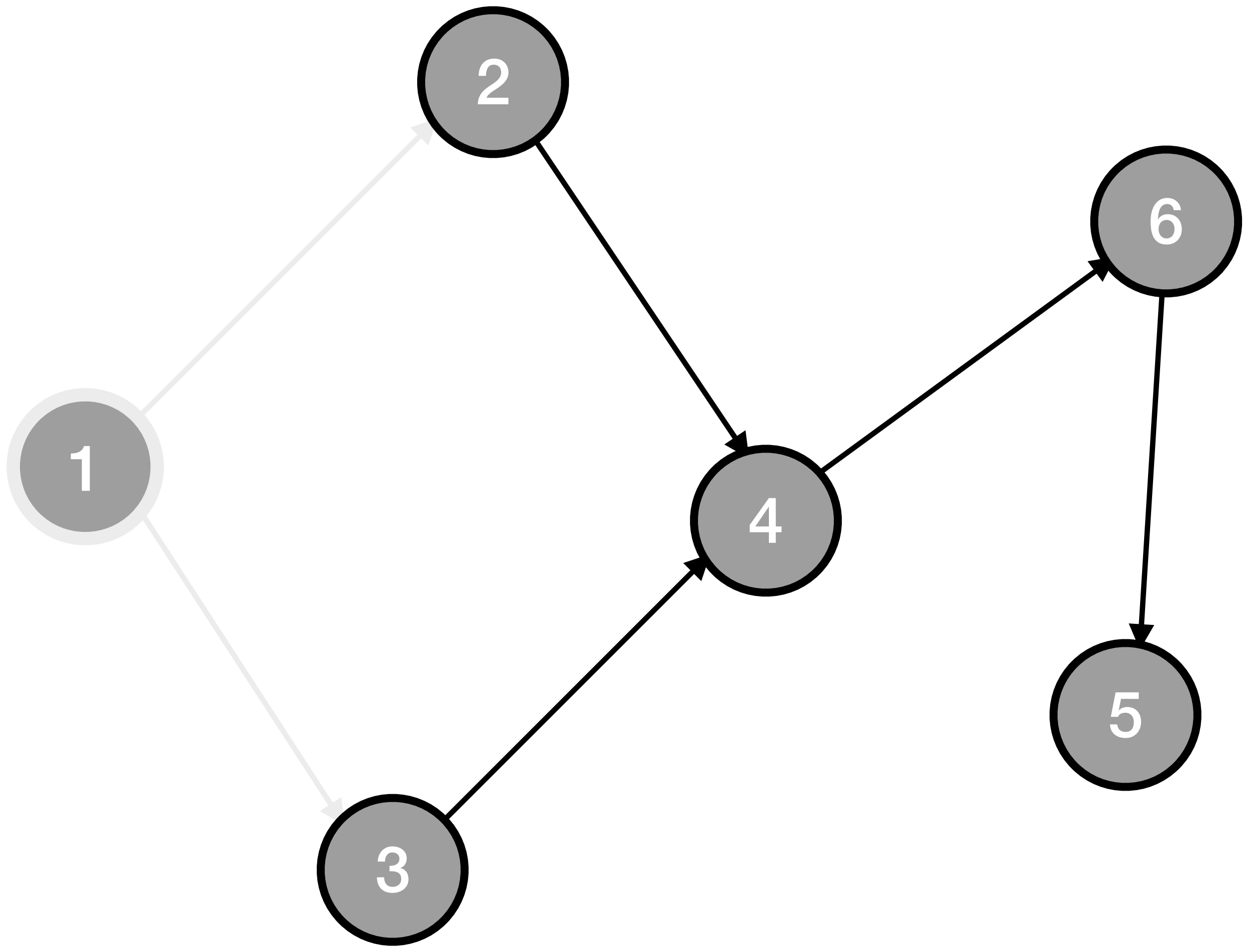
2021.02.14 10:00 KST

Topological Sort

Directed graph에서 모든 edge uv에 대해 u가 등장한 뒤에 v가 배치되도록
모든 정점을 linear하게 나열하는 것

➡ 작업의 선행 조건(a → b: a를 완료한 다음에 b 가능)에 따라 작업 순서 나열하기

➡ 게임 테크트리, 스킬트리 (이거 찍은 다음에 저거 찍고 그 담에 그거 찍고…)



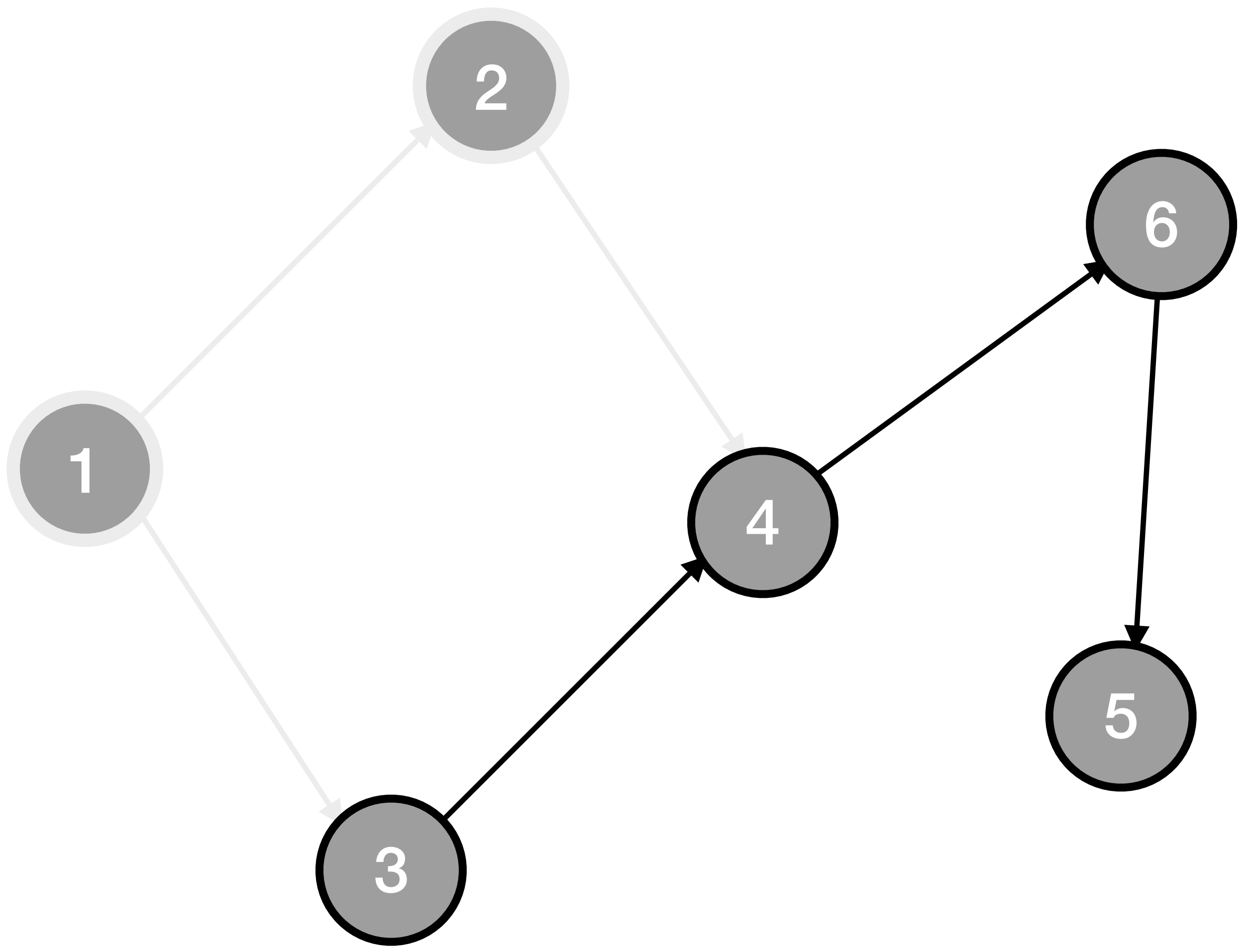
➡ 1

Topological Sort

Directed graph에서 모든 edge uv에 대해 u가 등장한 뒤에 v가 배치되도록
모든 정점을 linear하게 나열하는 것

➡ 작업의 선행 조건(a → b: a를 완료한 다음에 b 가능)에 따라 작업 순서 나열하기

➡ 게임 테크트리, 스킬트리 (이거 찍은 다음에 저거 찍고 그 담에 그거 찍고…)



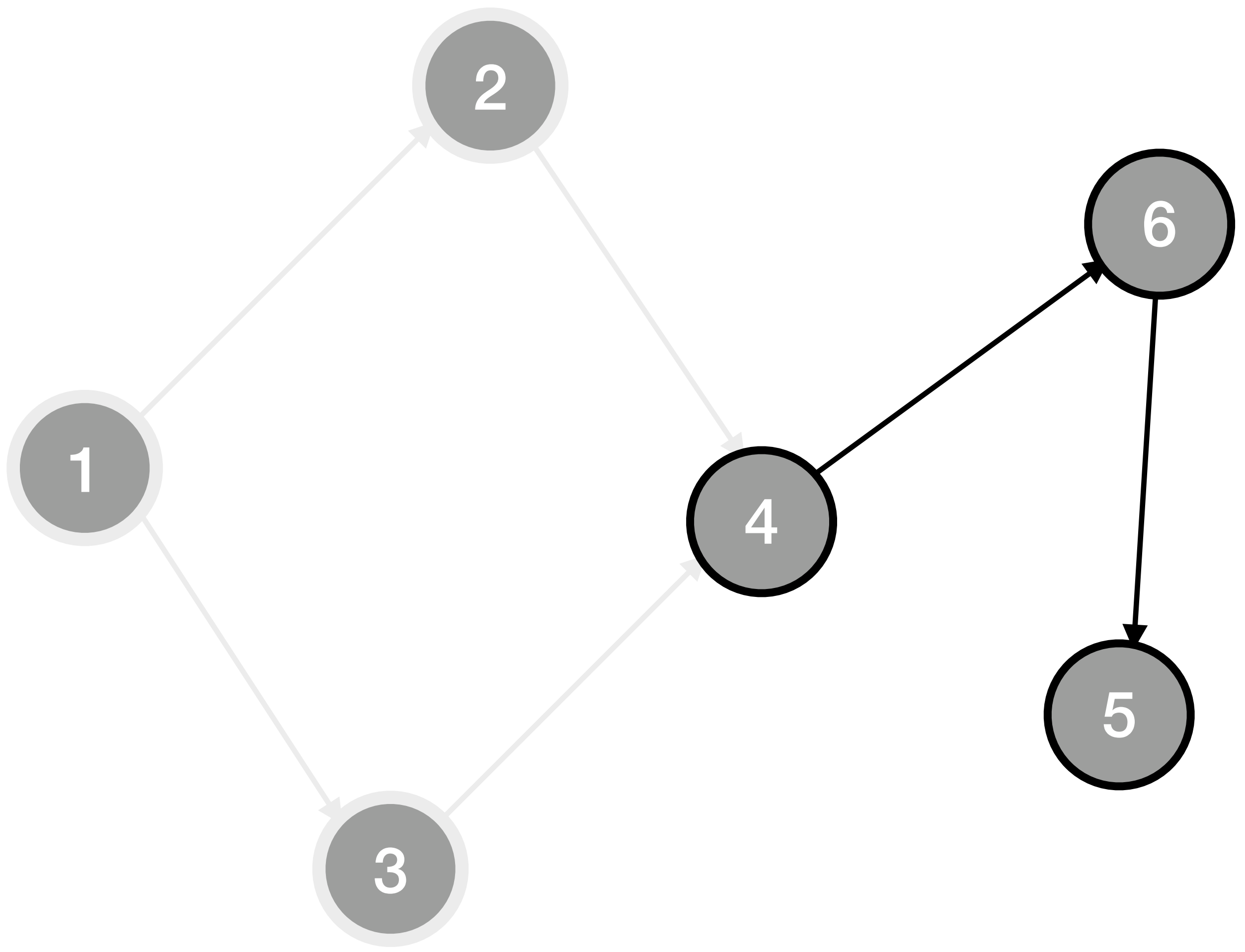
➡ 1 2

(3 먼저 가도 됨)

Topological Sort

Directed graph에서 모든 edge uv에 대해 u가 등장한 뒤에 v가 배치되도록
모든 정점을 linear하게 나열하는 것

➡ 작업의 선행 조건(a → b: a를 완료한 다음에 b 가능)에 따라 작업 순서 나열하기
➡ 게임 테크트리, 스킬트리 (이거 찍은 다음에 저거 찍고 그 담에 그거 찍고…)

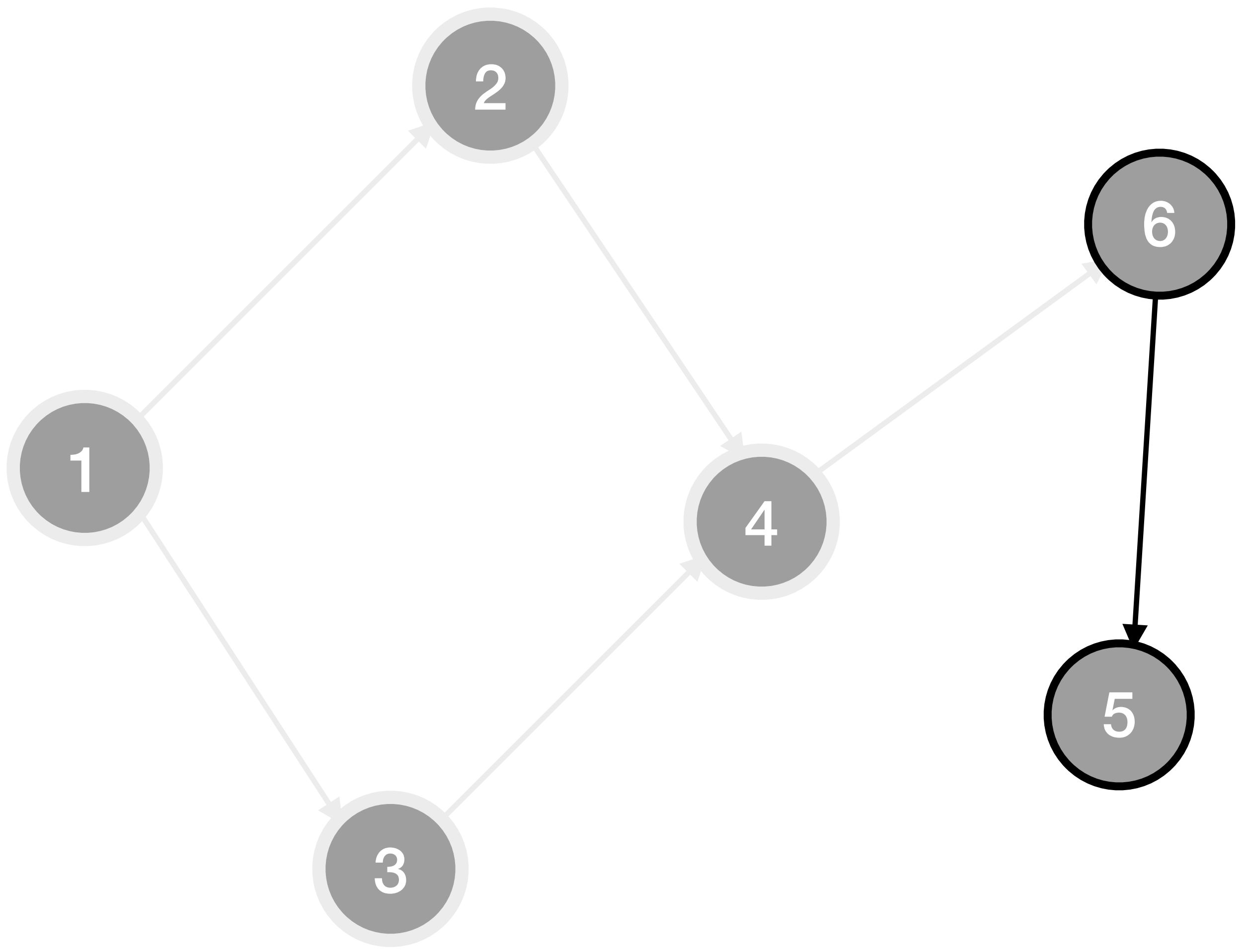


➡ 1 2 3

Topological Sort

Directed graph에서 모든 edge uv에 대해 u가 등장한 뒤에 v가 배치되도록
모든 정점을 linear하게 나열하는 것

➡ 작업의 선행 조건(a → b: a를 완료한 다음에 b 가능)에 따라 작업 순서 나열하기
➡ 게임 테크트리, 스킬트리 (이거 찍은 다음에 저거 찍고 그 담에 그거 찍고…)



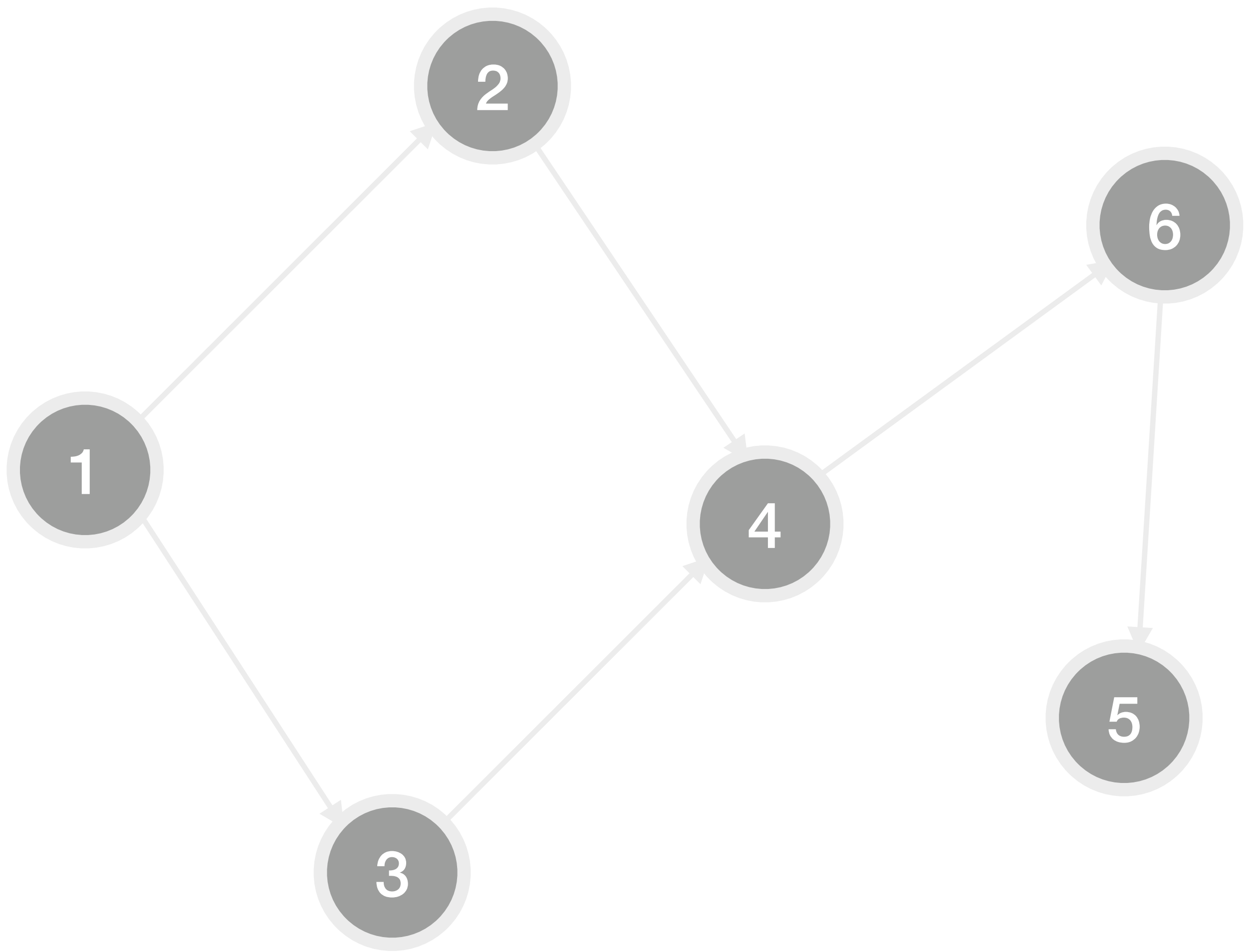
➡ 1 2 3 4

Topological Sort

Directed graph에서 모든 edge uv에 대해 u가 등장한 뒤에 v가 배치되도록
모든 정점을 linear하게 나열하는 것

➡ 작업의 선행 조건(a → b: a를 완료한 다음에 b 가능)에 따라 작업 순서 나열하기

➡ 게임 테크트리, 스킬트리 (이거 찍은 다음에 저거 찍고 그 담에 그거 찍고…)

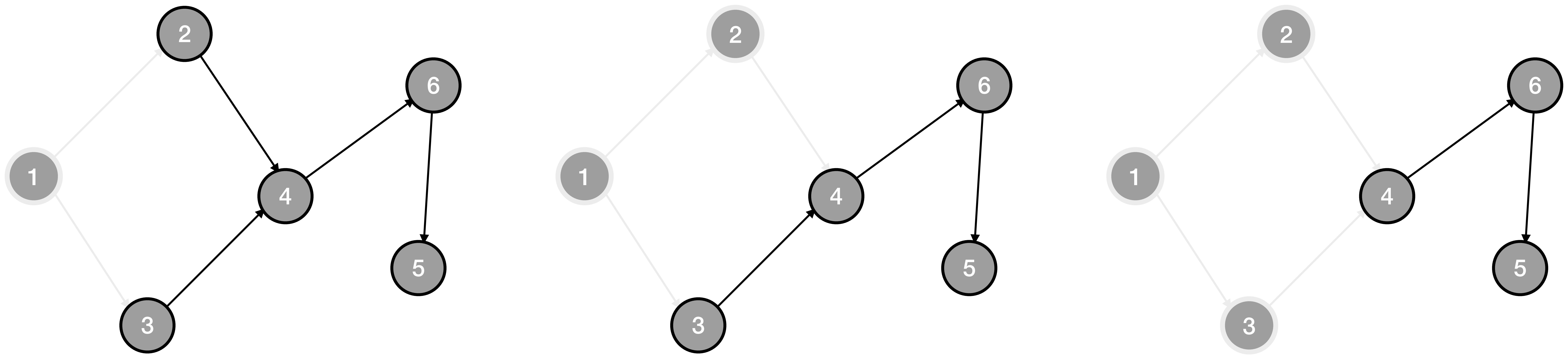


➡ 1 2 3 4 6 5

How it works

그때그때 적당히 정점을 골라서 나열하기

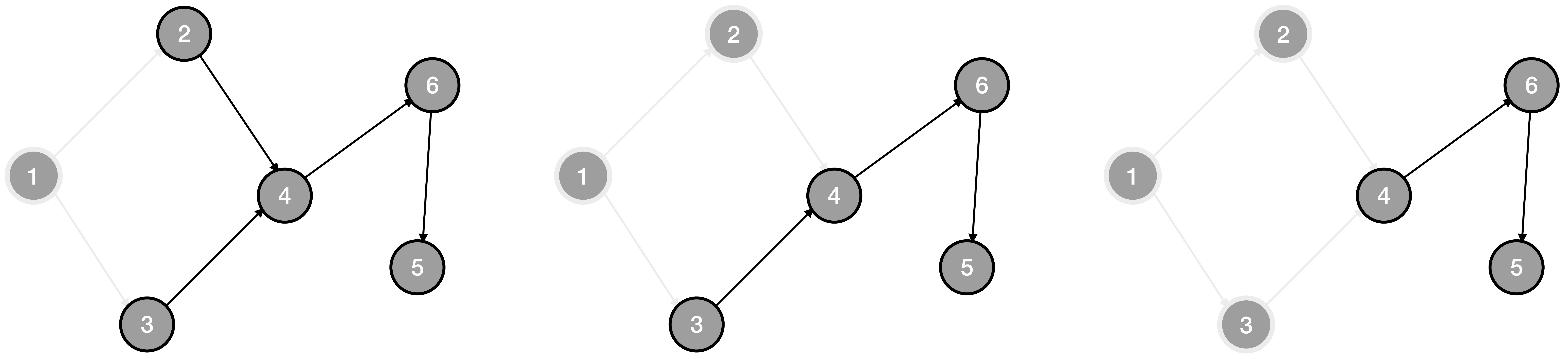
➡ 어떤 기준으로 정점을 골라야 하나요??



How it works

검은색 그래프에서 incoming edge가 없는 정점을 고르고
그 정점에서 출발하는 모든 edge를 그래프에서 제외

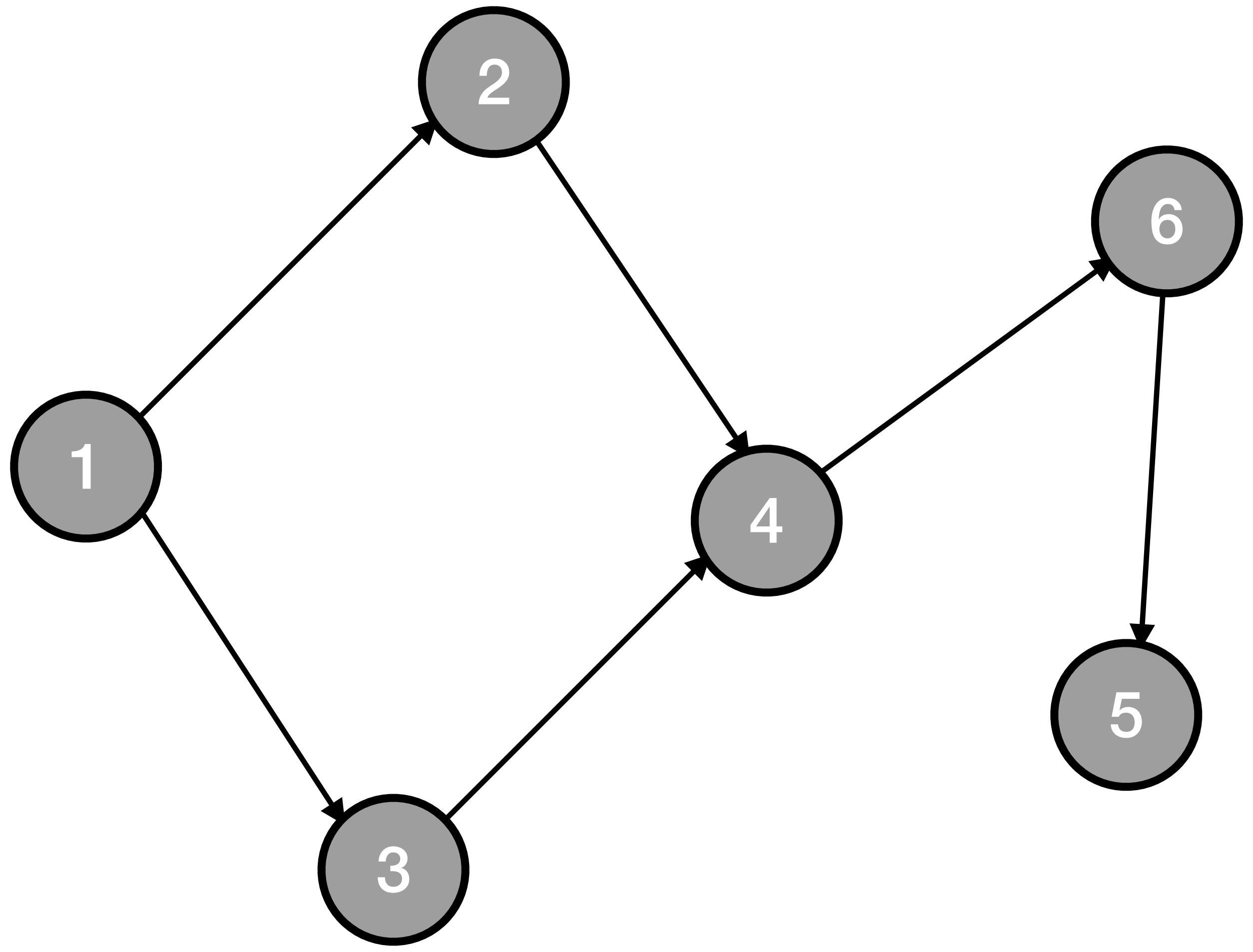
➡ 모든 정점을 나열할 때까지 반복



How it works

Algorithm 1 - Kahn's algorithm (이름은 몰라도 될 듯)

G(V, E)



L   위상 정렬 결과를 저장할 배열

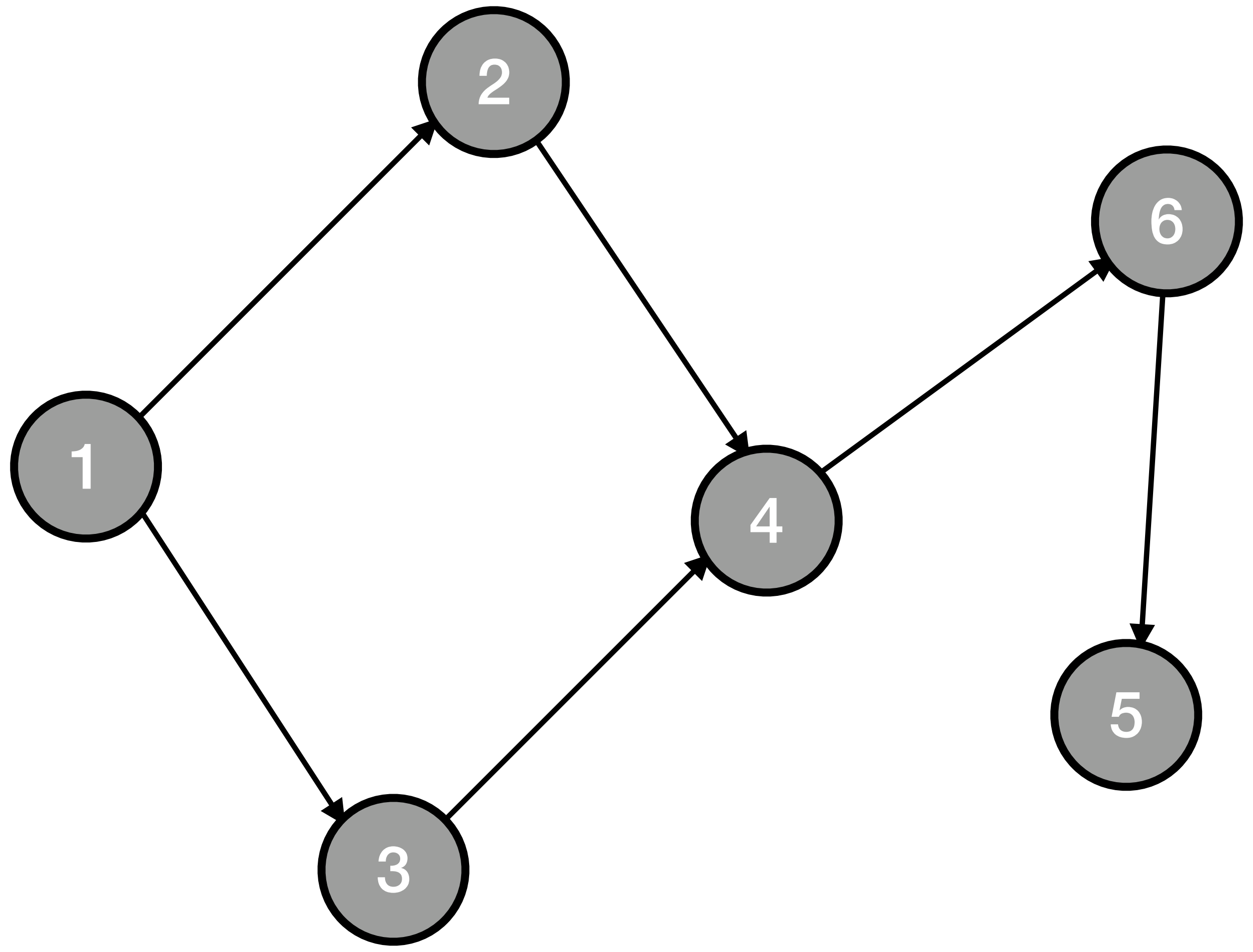
S   incoming edge가 없는 정점을 넣을 set

1) 현재 incoming edge가 없는 정점을 모두 S에 넣는다
2) S가 비어있지 않으면, 한 정점을 꺼내 이 정점을 u라고 하자.
   먼저 u를 L의 뒤에 추가하고,
   u에서 출발하는 모든 edge uv에 대해 이 edge를 G에서 제외한다.
   이때 정점 v의 incoming edge가 하나도 없다면 v를 S에 넣는다.
   S가 빌 때까지 이를 반복한다.
3) 알고리즘이 종료 후 L는 G의 위상 정렬 결과이다.

How it works

Algorithm 1 - Kahn's algorithm (이름은 몰라도 될 듯)

G(V, E)



1) 현재 incoming edge가 없는 정점을 모두 S에 넣는다
2) S가 비어있지 않으면, 한 정점을 꺼내 이 정점을 u라고 하자.
   먼저 u를 L의 뒤에 추가하고,
   u에서 출발하는 모든 edge uv에 대해 이 edge를 G에서 제외한다.
   이때 정점 v의 incoming edge가 하나도 없다면 v를 S에 넣는다.
   S가 빌 때까지 이를 반복한다.
3) 알고리즘이 종료 후 L는 G의 위상 정렬 결과이다.

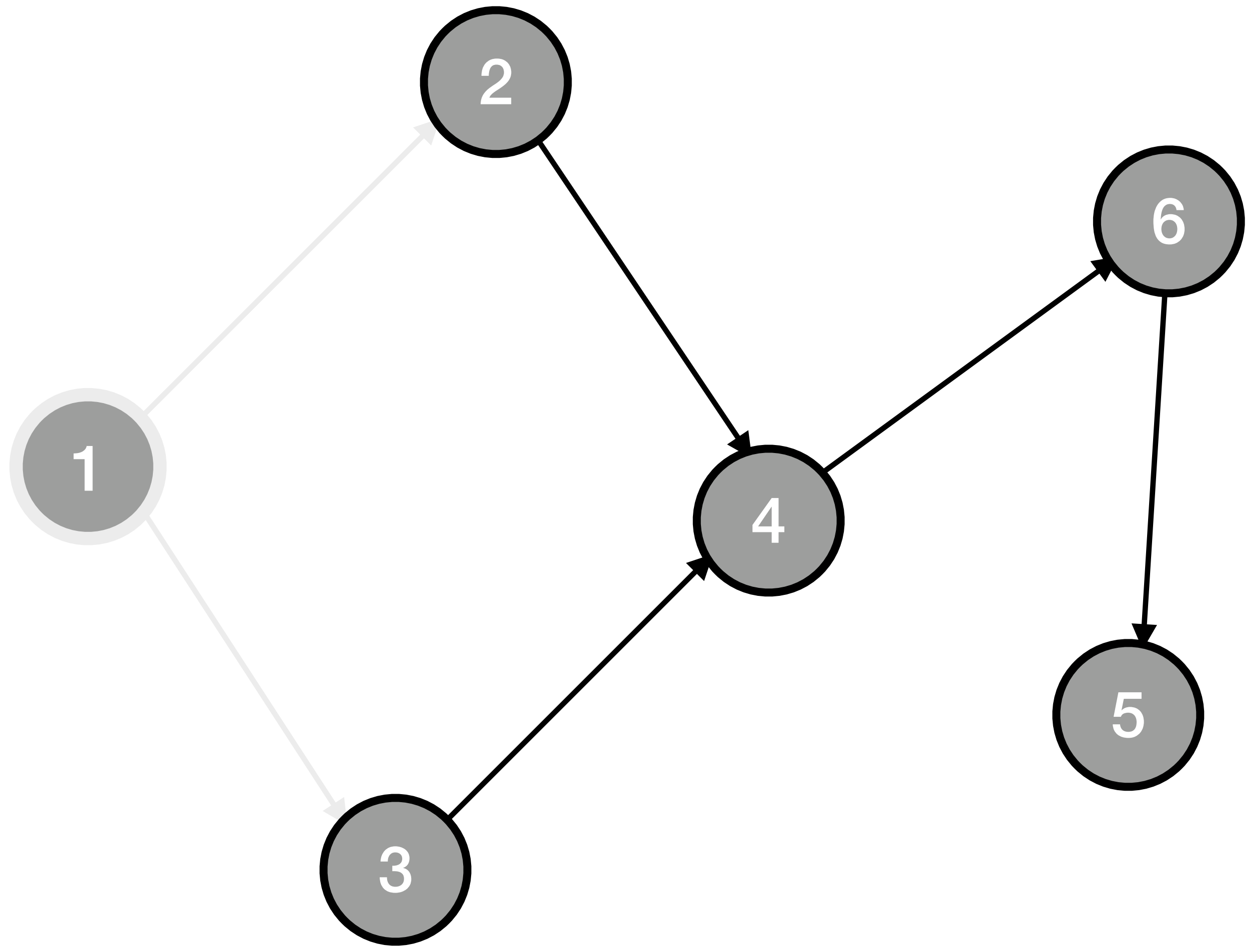
L

S 1

How it works

Algorithm 1 - Kahn's algorithm (이름은 몰라도 될 듯)

G(V, E)



L 1

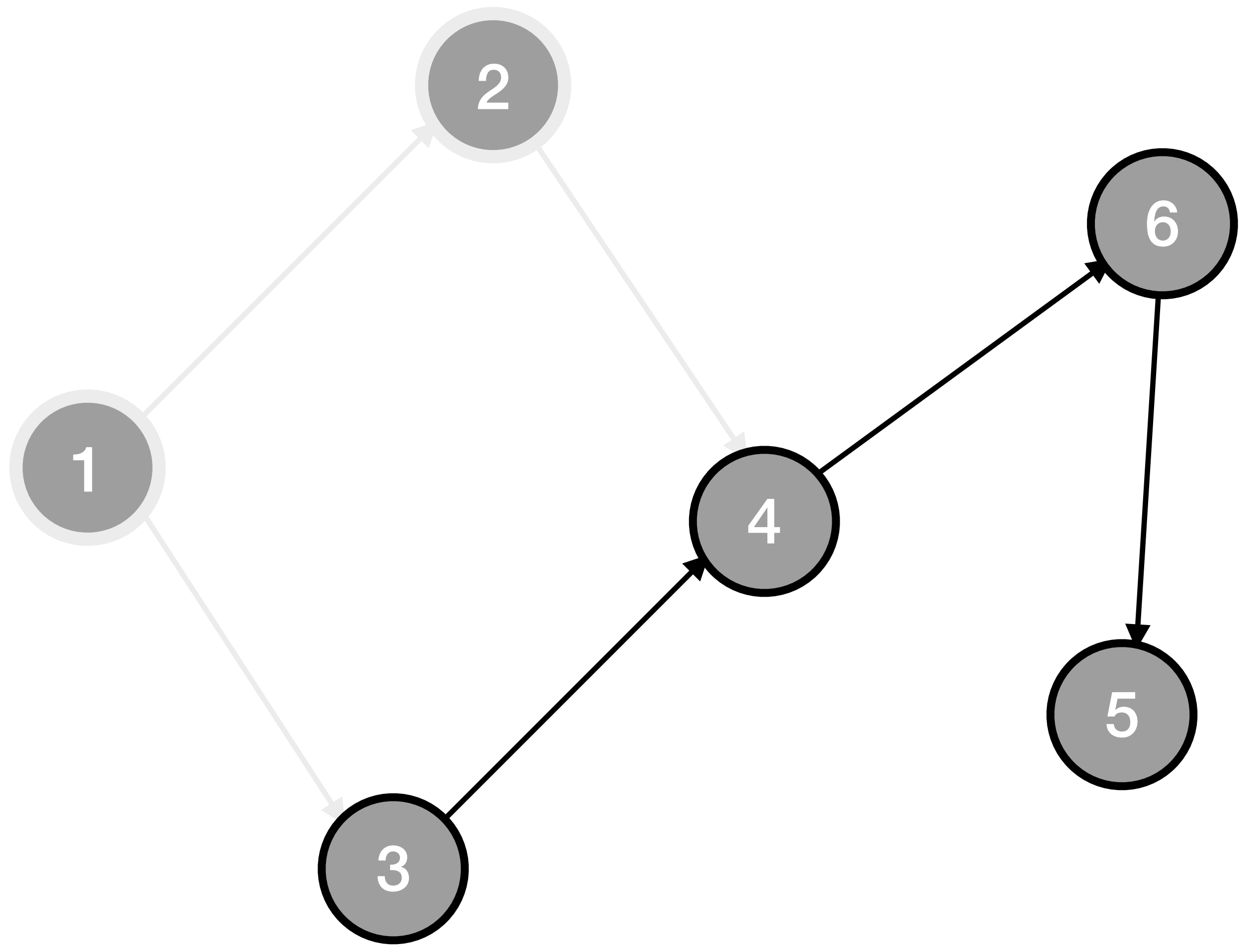
S 2 3

1) 현재 incoming edge가 없는 정점을 모두 S에 넣는다
2) S가 비어있지 않으면, 한 정점을 꺼내 이 정점을 u라고 하자.
   먼저 u를 L의 뒤에 추가하고,
   u에서 출발하는 모든 edge uv에 대해 이 edge를 G에서 제외한다.
   이때 정점 v의 incoming edge가 하나도 없다면 v를 S에 넣는다.
   S가 빌 때까지 이를 반복한다.
3) 알고리즘이 종료 후 L는 G의 위상 정렬 결과이다.

How it works

Algorithm 1 - Kahn's algorithm (이름은 몰라도 될 듯)

G(V, E)



L 1 2

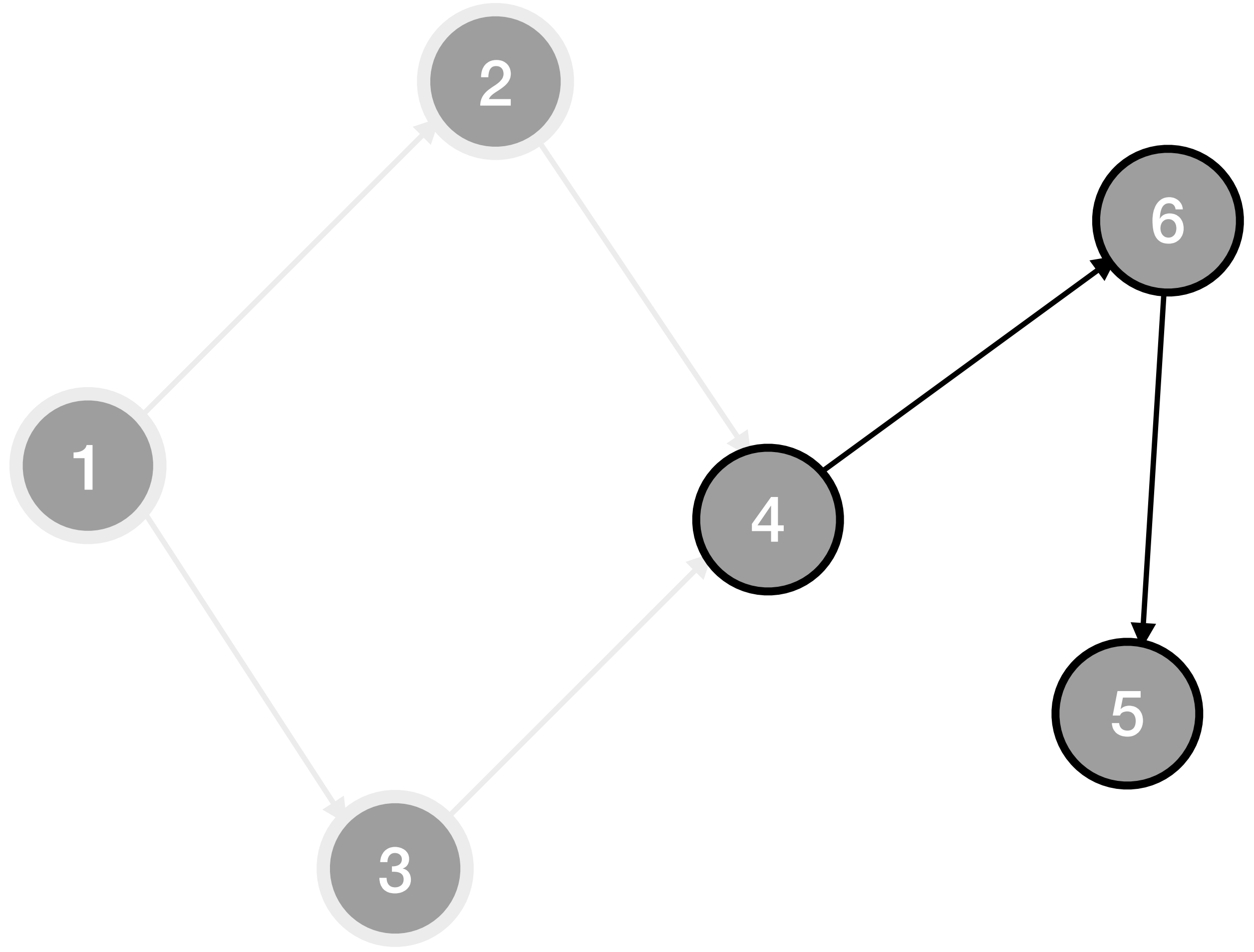
S 3

1) 현재 incoming edge가 없는 정점을 모두 S에 넣는다
2) S가 비어있지 않으면, 한 정점을 꺼내 이 정점을 u라고 하자.
   먼저 u를 L의 뒤에 추가하고,
   u에서 출발하는 모든 edge uv에 대해 이 edge를 G에서 제외한다.
   이때 정점 v의 incoming edge가 하나도 없다면 v를 S에 넣는다.
   S가 빌 때까지 이를 반복한다.
3) 알고리즘이 종료 후 L는 G의 위상 정렬 결과이다.

How it works

Algorithm 1 - Kahn's algorithm (이름은 몰라도 될 듯)

G(V, E)



L 1 2 3

S 4

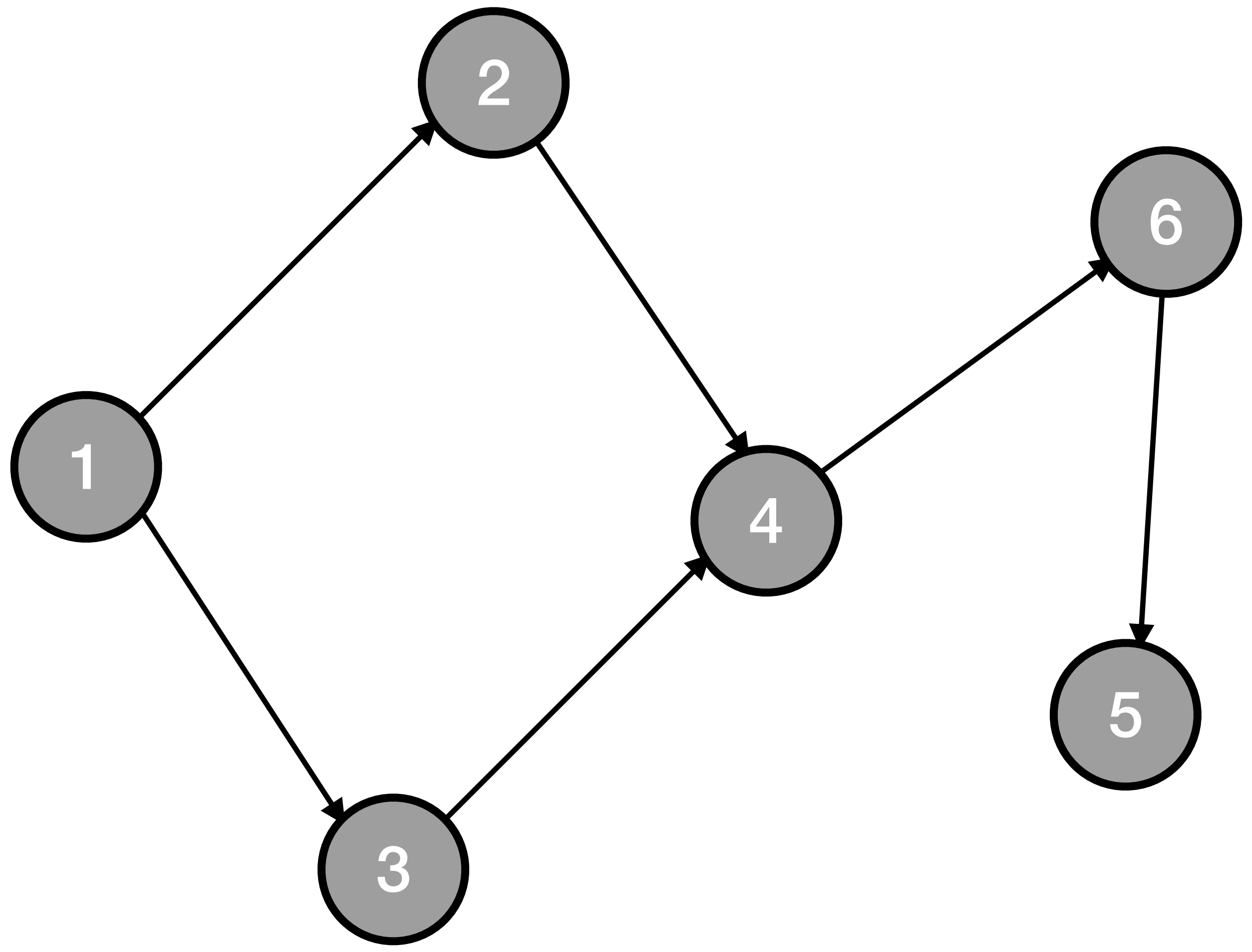
(이하 생략)

1) 현재 incoming edge가 없는 정점을 모두 S에 넣는다
2) S가 비어있지 않으면, 한 정점을 꺼내 이 정점을 u라고 하자.
   먼저 u를 L의 뒤에 추가하고,
   u에서 출발하는 모든 edge uv에 대해 이 edge를 G에서 제외한다.
   이때 정점 v의 incoming edge가 하나도 없다면 v를 S에 넣는다.
   S가 빌 때까지 이를 반복한다.
3) 알고리즘이 종료 후 L는 G의 위상 정렬 결과이다.

How it works

Algorithm 2 - DFS

G(V, E)



모든 미방문 정점 u에서 dfs를 돌린다.

u의 모든 child를 방문하고 dfs가 종료될 때, **L의 앞**에 u를 추가한다.

(L을 역순으로 완성한다고 보면 됨)

알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

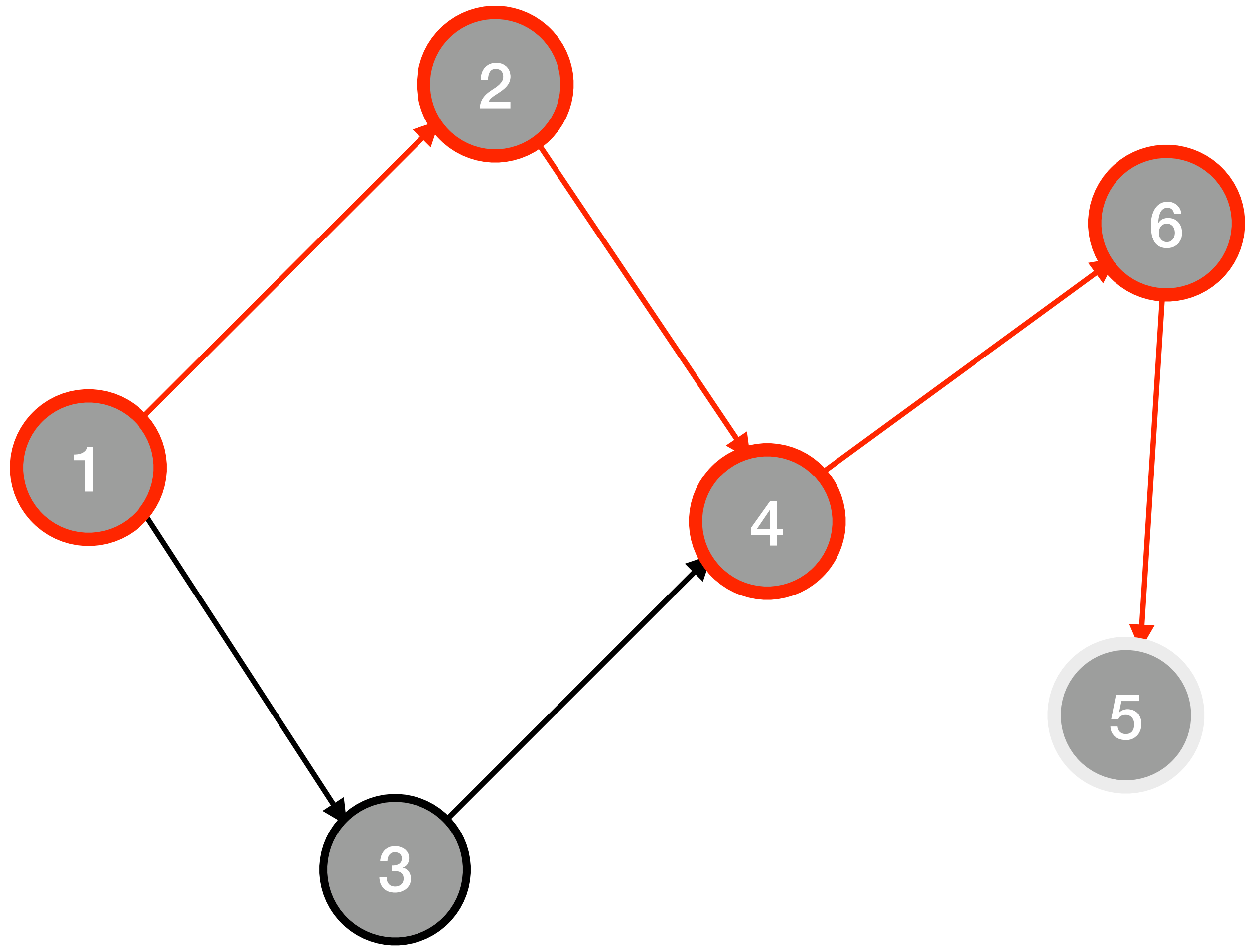
L

위상 정렬 결과를 저장할 배열

How it works

Algorithm 2 - DFS

G(V, E)



L 5

모든 미방문 정점 u에서 dfs를 돌린다.

u의 모든 child를 방문하고 dfs가 종료될 때, **L의 앞**에 u를 추가한다.

(L을 역순으로 완성한다고 보면 됨)

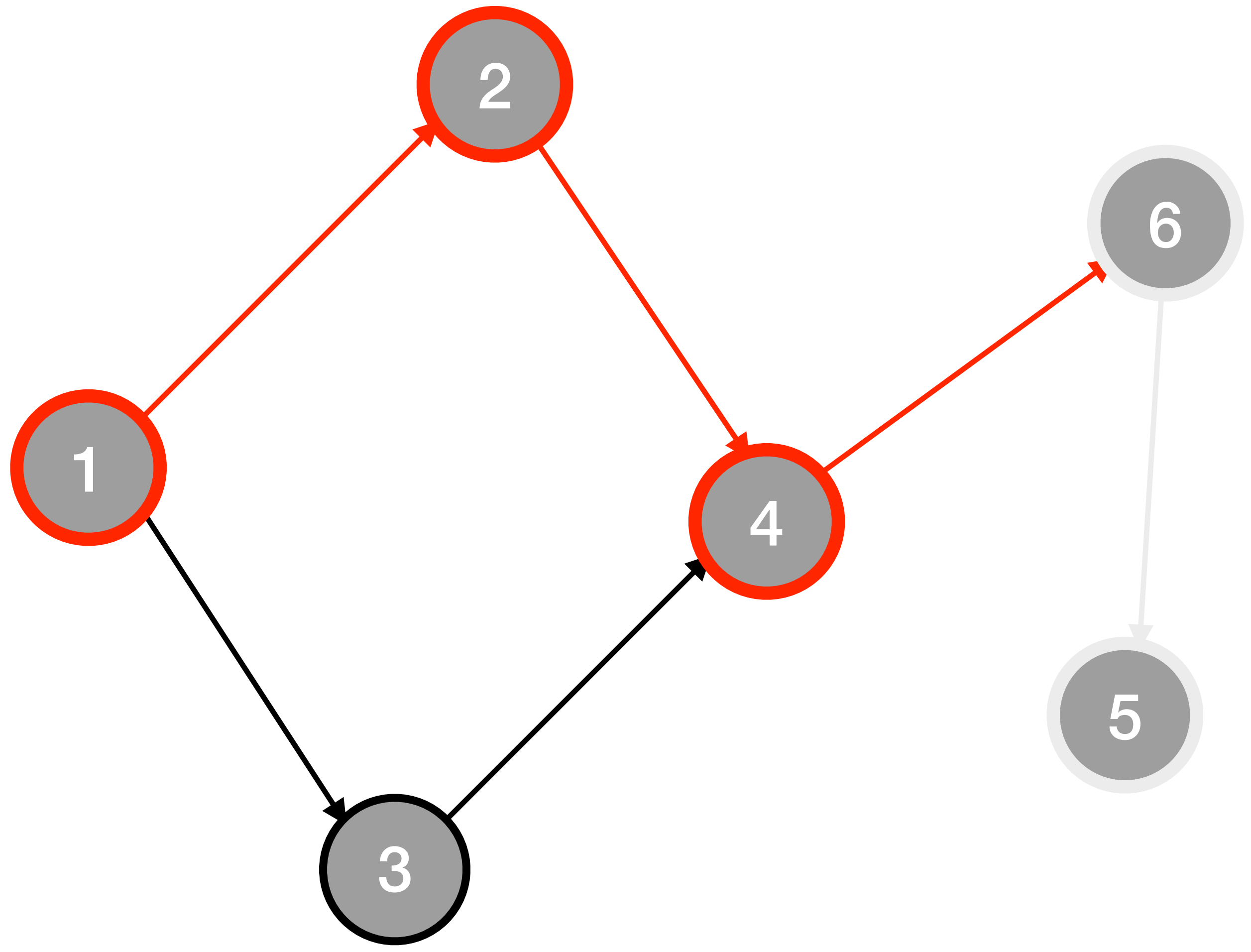
알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

How it works

Algorithm 2 - DFS

G(V, E)



L 6 5

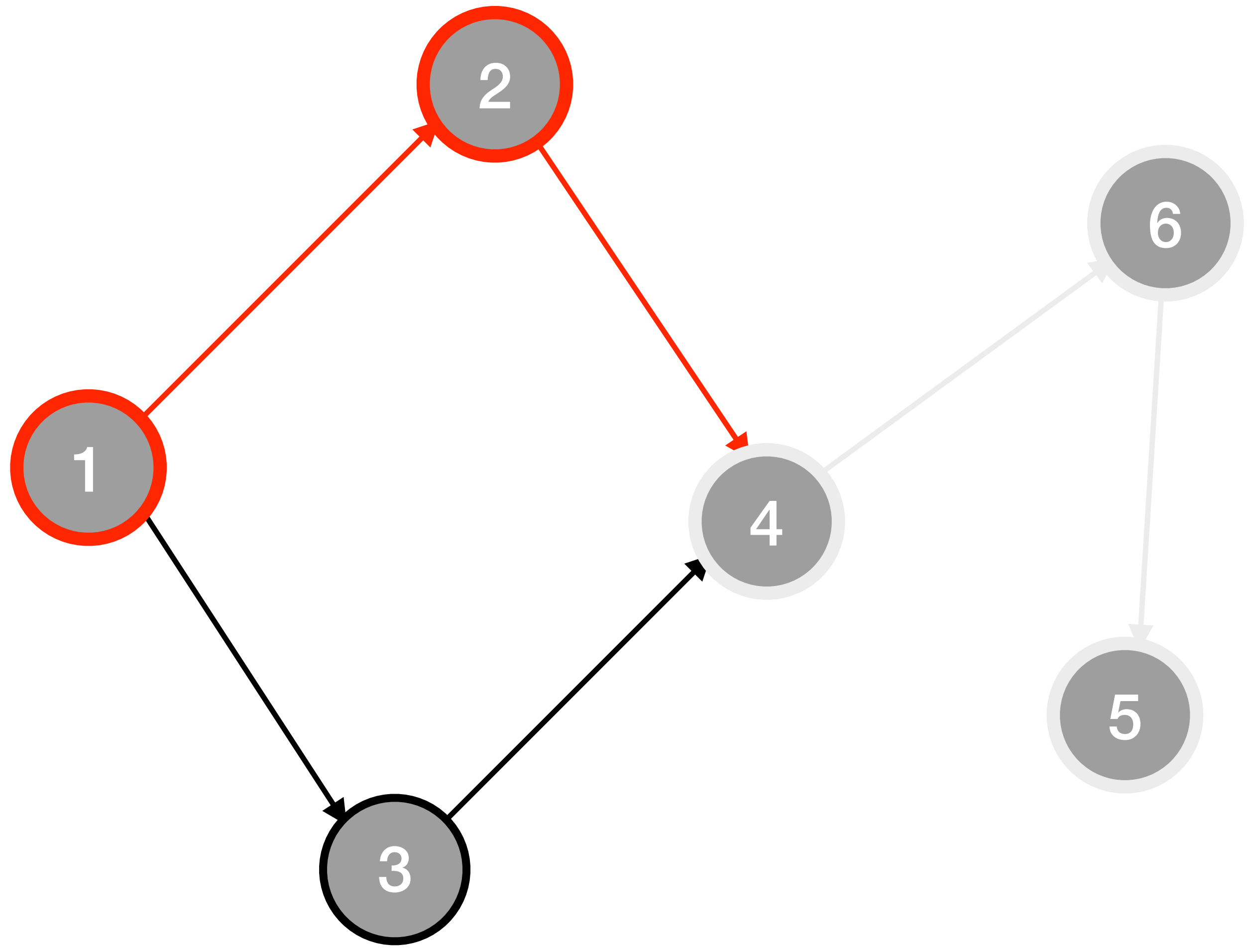
모든 미방문 정점 u에서 dfs를 돌린다.
u의 모든 child를 방문하고 dfs가 종료될 때, **L의 앞**에 u를 추가한다.
(L을 역순으로 완성한다고 보면 됨)
알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

How it works

Algorithm 2 - DFS

G(V, E)



L 4 6 5

모든 미방문 정점 u에서 dfs를 돌린다.

u의 모든 child를 방문하고 dfs가 종료될 때, **<u>L의 앞</u>**에 u를 추가한다.

(L을 역순으로 완성한다고 보면 됨)

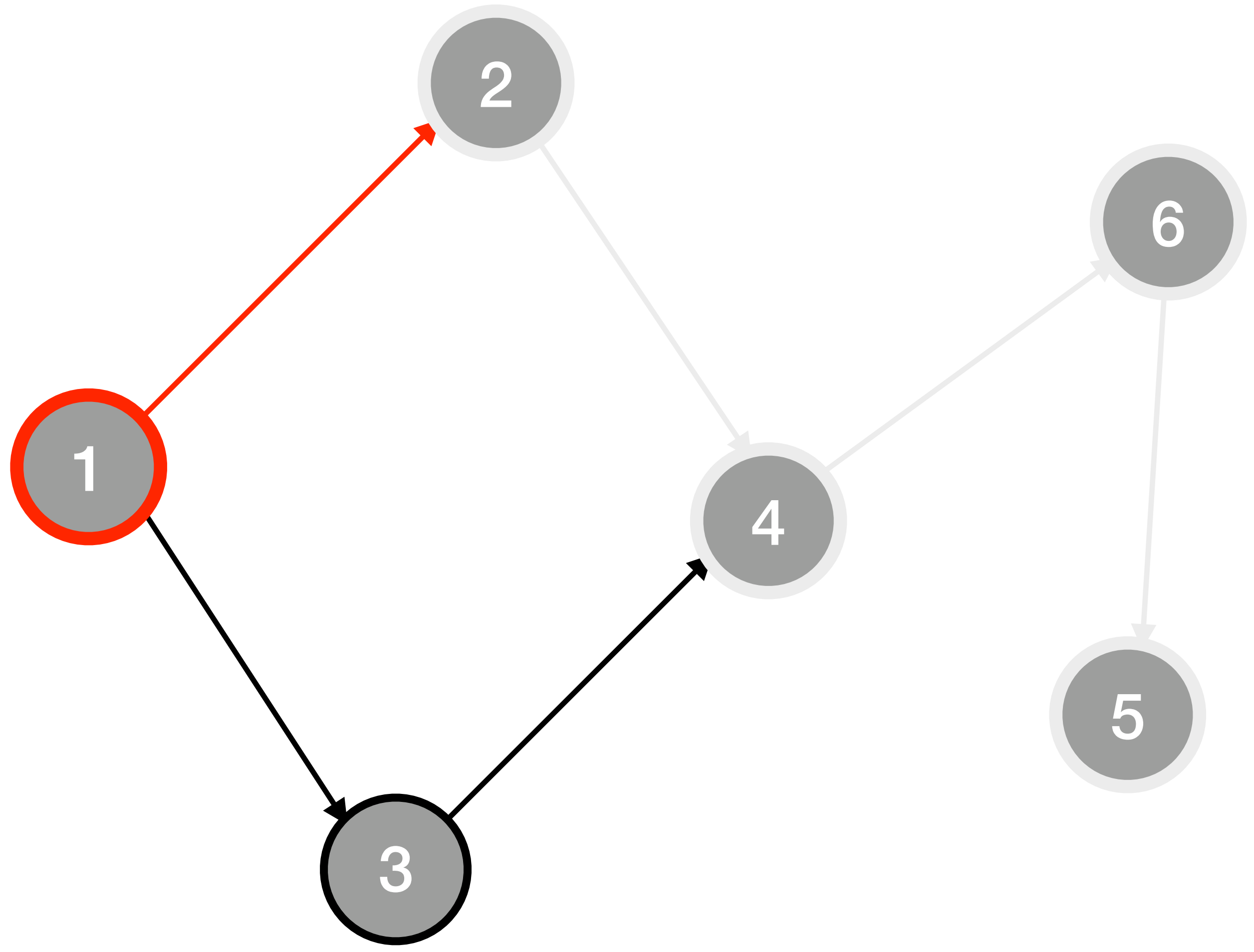
알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

How it works

Algorithm 2 - DFS

G(V, E)



L 2 4 6 5

모든 미방문 정점 u에서 dfs를 돌린다.

u의 모든 child를 방문하고 dfs가 종료될 때, **<u>L의 앞</u>**에 u를 추가한다.

(L을 역순으로 완성한다고 보면 됨)

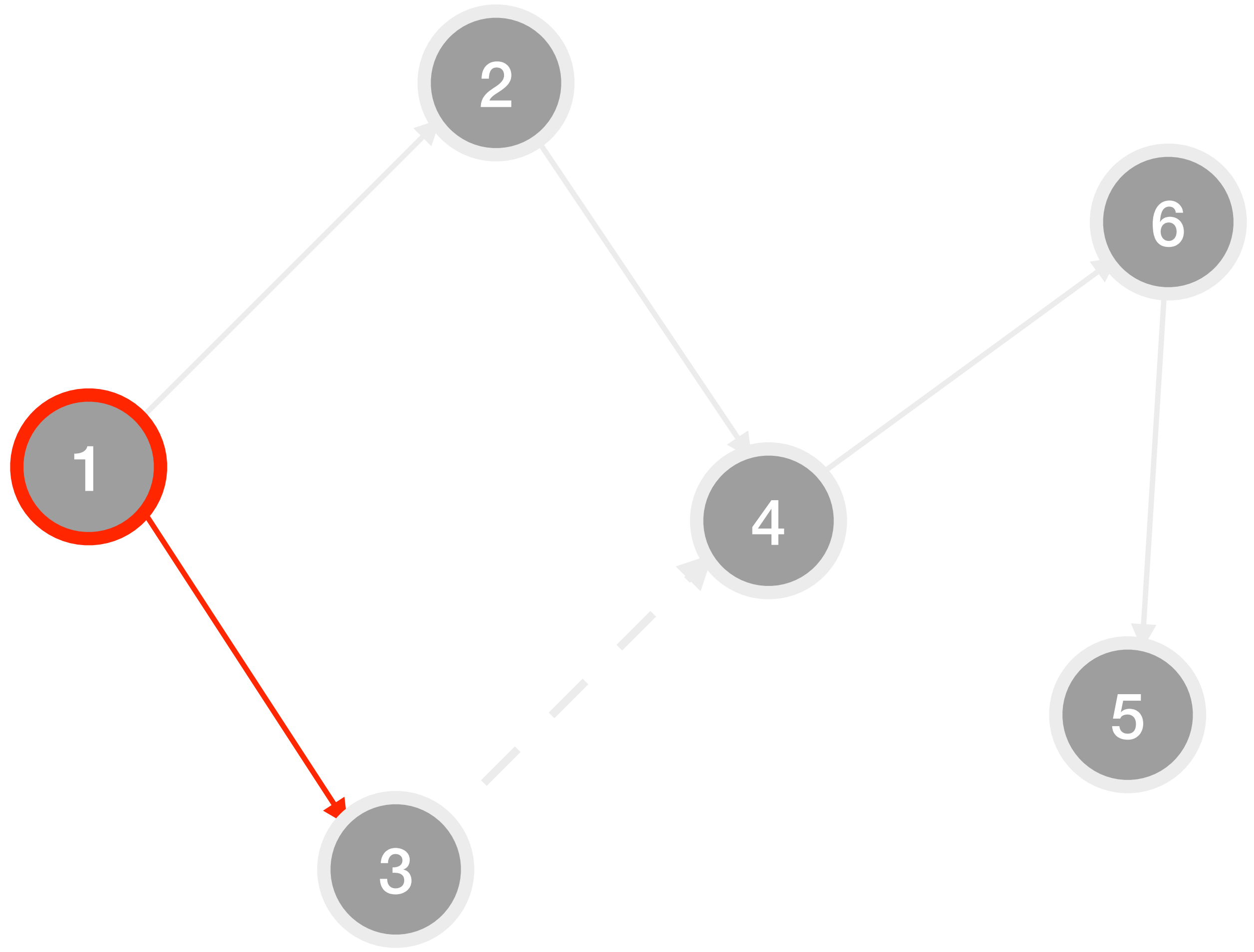
알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

How it works

Algorithm 2 - DFS

G(V, E)



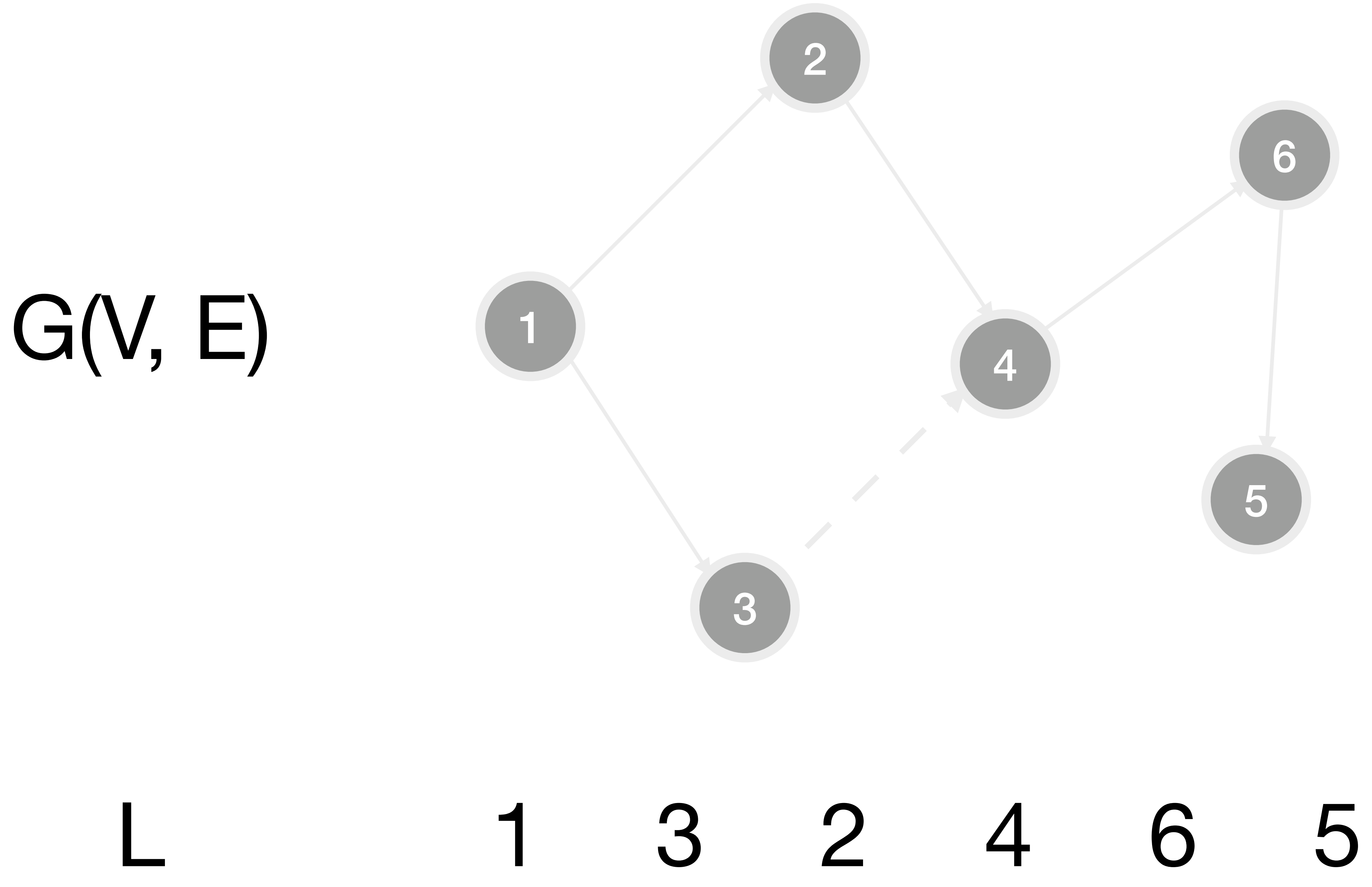
모든 미방문 정점 u에서 dfs를 돌린다.
u의 모든 child를 방문하고 dfs가 종료될 때, **<u>L의 앞</u>**에 u를 추가한다.
(L을 역순으로 완성한다고 보면 됨)
알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

L 3 2 4 6 5

How it works

Algorithm 2 - DFS



모든 미방문 정점 u에서 dfs를 돌린다.

u의 모든 child를 방문하고 dfs가 종료될 때, **L의 앞**에 u를 추가한다.

(L을 역순으로 완성한다고 보면 됨)

알고리즘 종료 후 L은 G의 위상 정렬 결과이다.

➡ 🤔 이게 정말 되나요?

How it works

Time complexity

두 방법 모두 각 정점과 그 정점에 연결된 edge를 순회하므로

시간복잡도는 $O(|V| + |E|)$

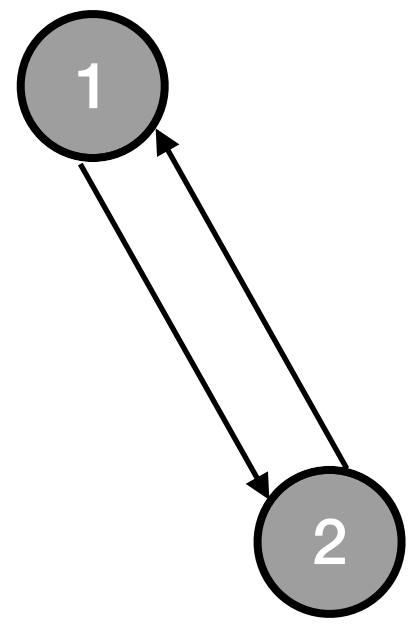
Does it work?

단점

1) 현재 **incoming edge가 없는 정점**을 모두 S에 넣는다
2) …

➡ 모든 정점을 다 순회하지 않았는데 incoming edge가 없는 정점이 없으면 어떡하나요??

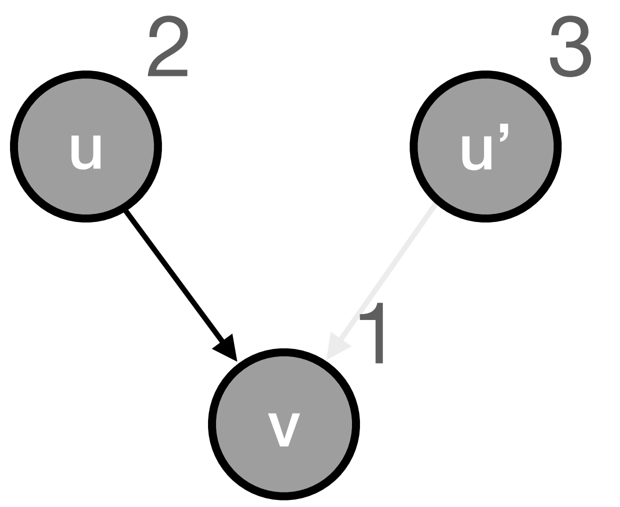
➡ 사이클이 없는 **directed acyclic graph (DAG)**여야 위상 정렬 가능



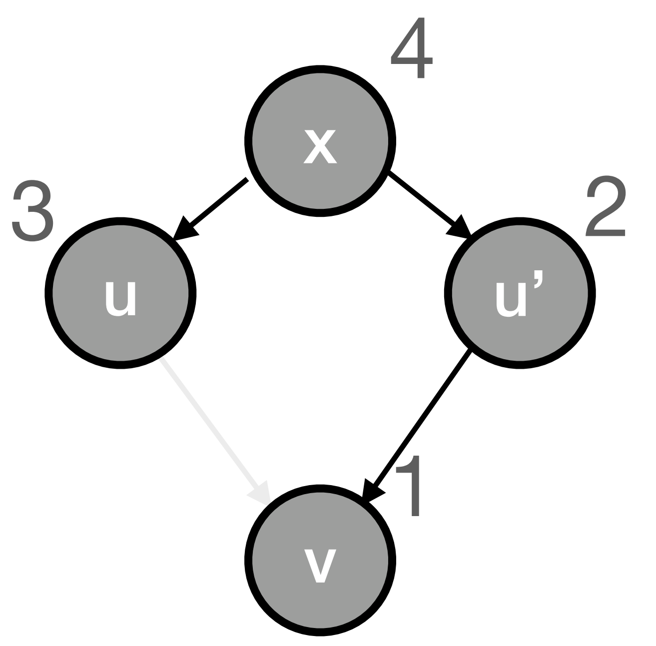
Does it work?

Algorithm 2 - DFS

1)



2)



얼레벌레 $proof$) (맞는지는 모름 ㅎ)

어떤 edge uv에 대해 배열 L에서 u가 항상 v보다 앞선다는 것을 증명하면 된다.

1) dfs(u)가 dfs(v)보다 먼저 호출되는 경우
   이는 dfs(u)가 호출되는 시점에 아직 v를 방문하지 않았음을 의미한다.
   그러므로 dfs(u)가 종료되기 전에 v를 방문하게 된다.
   따라서 dfs(v)는 dfs(u)보다 먼저 종료되므로 u는 v보다 앞선다.
2) dfs(v)가 dfs(u)보다 먼저 호출되는 경우
   이는 어떤 정점 u’(v 포함)에서 재귀적으로 dfs를 수행한 결과
   u를 거치지 않고 v를 먼저 방문하였음을 의미한다.
   위상 정렬을 수행할 때 G는 DAG임을 가정하기에, edge uv가 이미 존재하는
   이상 v에서 u로 가는 경로는 존재하지 않는다.
   즉, v를 방문하면 u를 방문하기 전에 먼저 dfs(v)가 종료된다.
   따라서 dfs(v)는 dfs(u)보다 먼저 종료되므로 u는 v보다 앞선다.

그러므로 배열 L에서 u는 항상 v보다 앞선다카더라.