ALGORITHM
알고리즘

Lecture 08

그래프 (1)

컴퓨터과학과 | 이관용 교수

한국방송통신대학교

학습목차

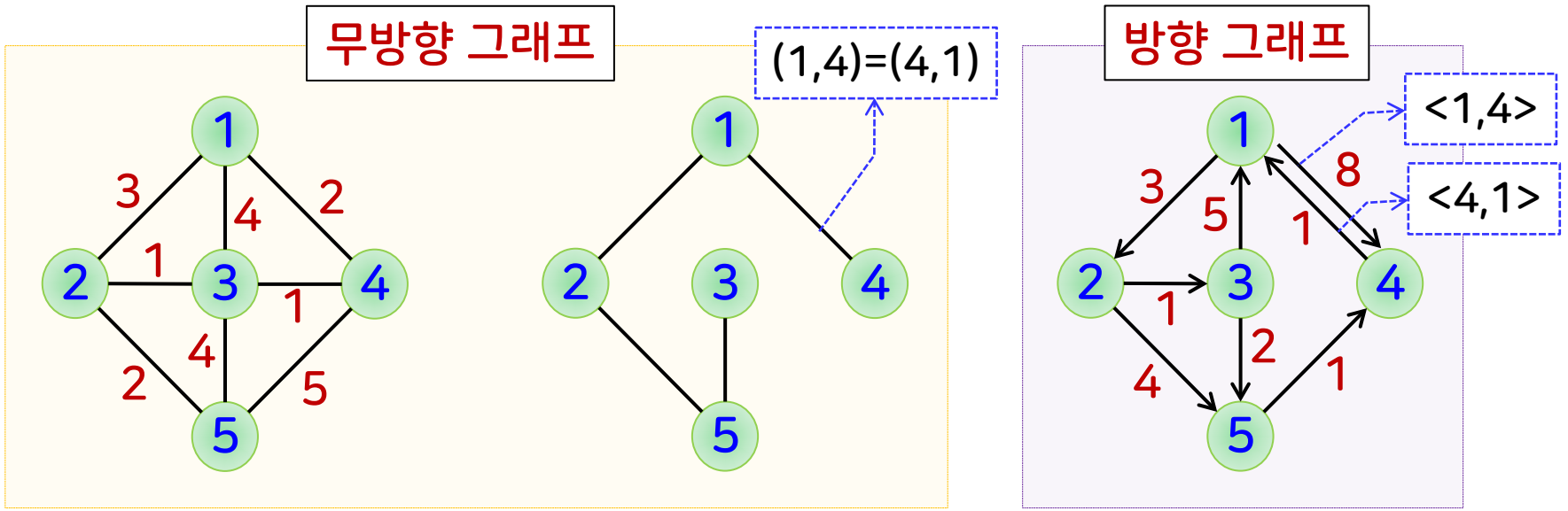
ALGORITHM
알고리즘

01. 기본 개념

그래프?

그래프 G → G=(V, E)

- V: 정점vertex의 집합, E: 간선edge의 집합



G_1 G_2 G_3

가중 그래프

$V(G_2) = \{ 1, 2, 3, 4, 5 \}$

$E(G_2) = \{ (1,2), (1,4), (2,5), (3,5) \}$

가중 그래프

트리

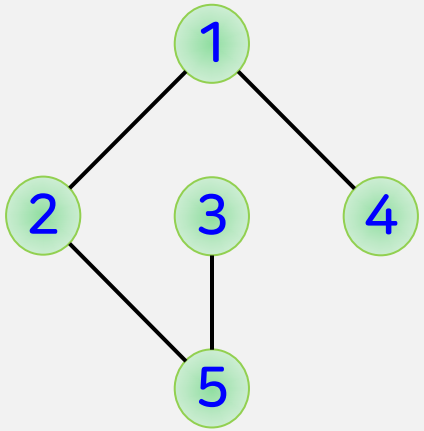
그래프의 주요 용어

- **인접** adjacent, **부수** incident
- **부분 그래프** subgraph
- **경로** path, **경로의 길이** length
- **차수** degree
  - ✓ **진입차수** in-degree, **진출차수** out-degree
- **단순 경로** simple path, **사이클** cycle, **루프** loop
- **연결** connected
- **강하게 연결** strongly connected, **약하게 연결** weakly connected

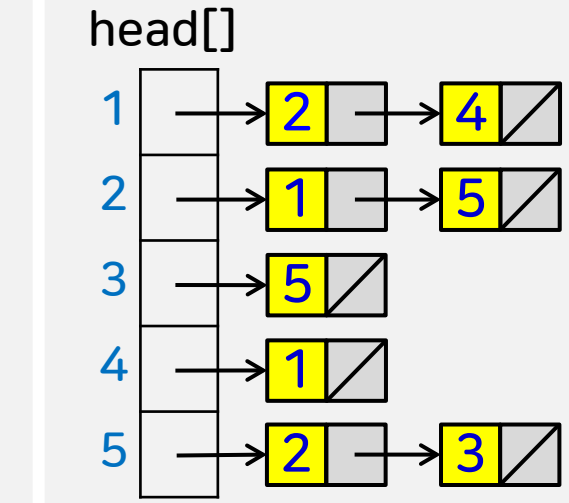
그래프의 구현 방법

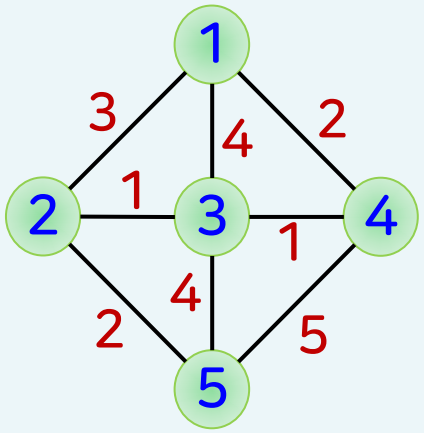
인접 행렬
adjacency matrix

인접 리스트
adjacency list

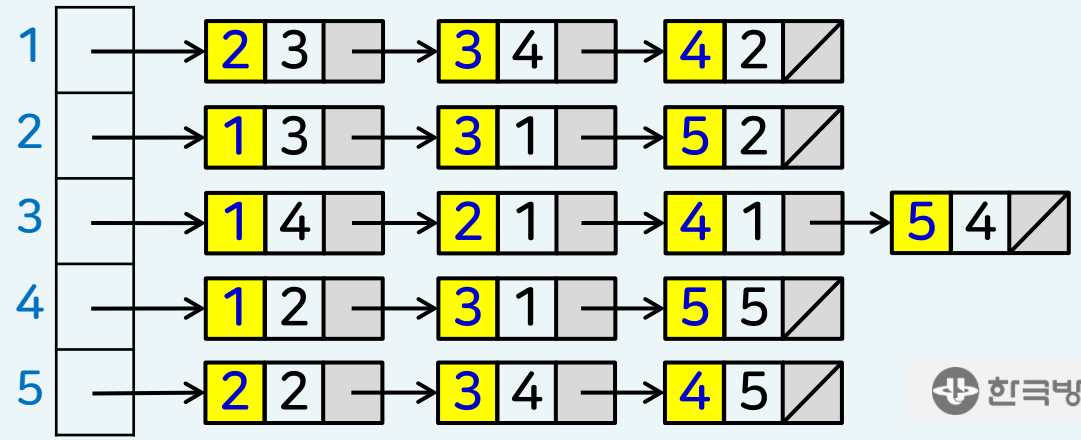


	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	0
5	0	1	1	0	0





	1	2	3	4	5
1	0	3	4	2	∞
2	3	0	1	∞	2
3	4	1	0	1	4
4	2	∞	1	0	5
5	∞	2	4	5	0



02. 그래프 순회

그래프 순회?

그래프의 모든 정점을 체계적으로 한 번씩 방문하는 것

- 그래프 탐색 방법

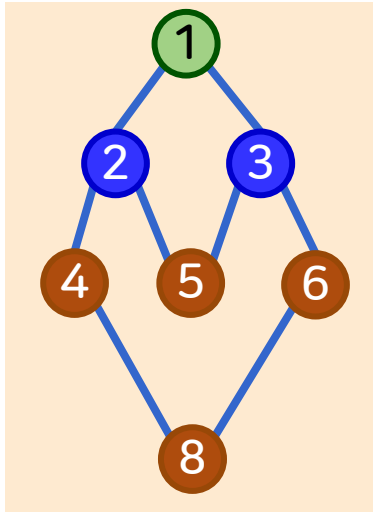
순회 방법

- 깊이 우선 탐색 DFS, Depth First Search
- 너비 우선 탐색 BFS, Breadth First Search

그래프 순회

탐색 과정에서의 정점의 구분

- **방문 정점**
  - ✓ 방문이 완료된 정점
- **주변 정점**
  - ✓ 방문 정점에 인접한 정점 중에서 아직 방문하지 않은 정점
- **미도달 정점**
  - ✓ 방문 정점도 주변 정점도 아닌 전혀 접근하지 못한 정점



깊이 우선 탐색 → 최근의 주변 정점을 우선 방문

너비 우선 탐색 → 주변 정점 중에서 오래된 것을 우선 방문

깊이 우선 탐색

한 정점을 시작으로 매번 인접한 정점 중 한 곳으로 이동하며 탐색하는 방법

→ 최근의 주변 정점을 우선으로 방문하는 탐색 방법

스택 구조를 사용해서 구현

- 현재 정점에 인접한 정점이 없어서 더 이상 탐색을 진행할 수 없으면 거꾸로 되돌아가면서 아직 탐색하지 않은 인접한 정점을 찾아서 탐색을 진행

깊이 우선 탐색

처리 과정

① 시작 정점을 스택에 삽입

② 스택의 top에 있는 정점에 대한 주변 정점이 존재하면

그중 하나의 정점을 스택에 삽입하고 방문한 정점으로 처리.

주변 정점이 없다면 스택의 top에 있는 정점을 제거

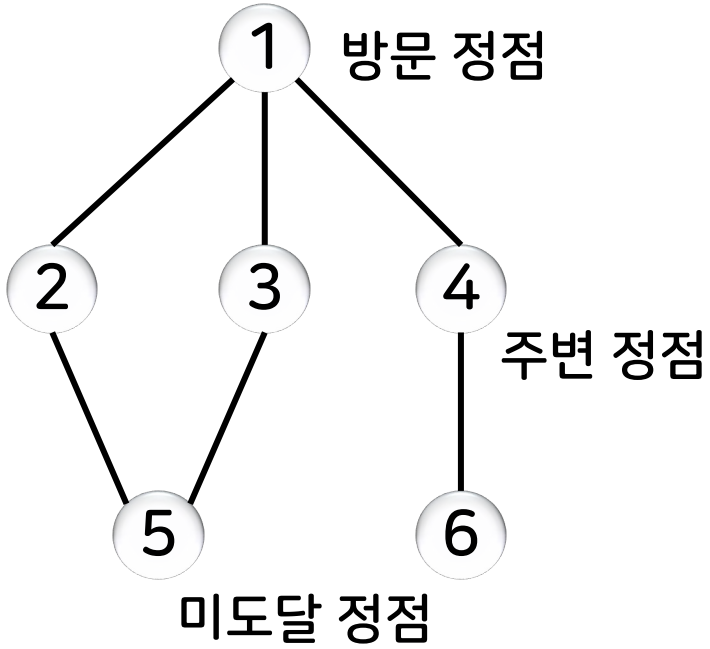
③ 스택에 더 이상의 정점이 없을 때까지 ②의 과정을 반복

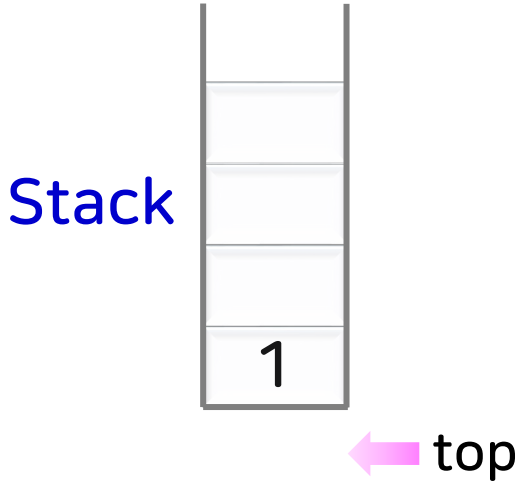
깊이 우선 탐색 알고리즘


```
DepthFirstSearch (G, s) {    // G: 입력 그래프,  s: 시작 정점
  Push(Stack, s);
  while (Stack != NULL) {
    c = Stack의 top에 있는 정점;
    c.visited = TRUE;
    정점 c를 방문 정점으로 출력;
    do {
      for (v ← c의 모든 인접한 정점) {
        if (v.visited == FALSE)
          Push(Stack, v) 후 while문으로 이동;
      }
      c = Pop(Stack);
    } while (Stack != NULL);
  }
}
```

깊이 우선 탐색_예_1

시작 정점 1 → push



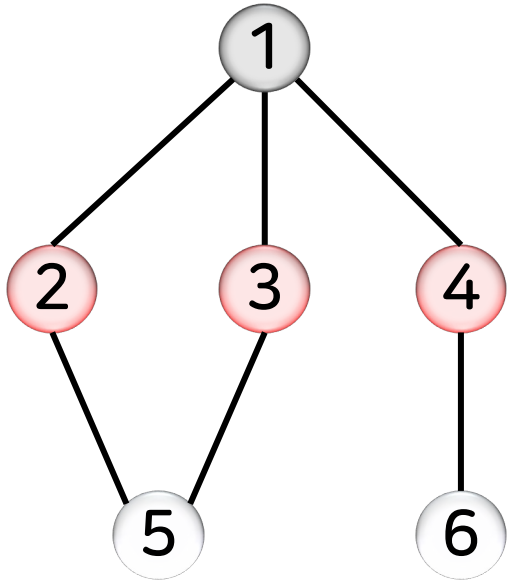


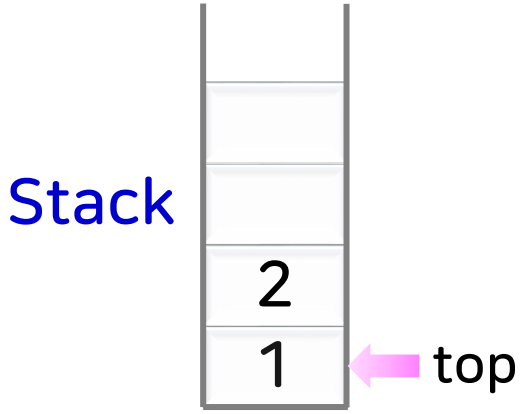
방문 순서

깊이 우선 탐색_예_1

정점 1의 주변 정점(2,3,4) 중에서 하나를 선택 → 정점 2 → push

스택의 top에 있는 정점

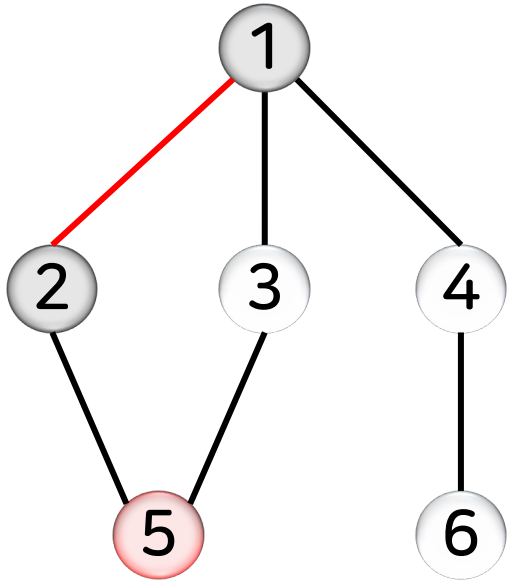


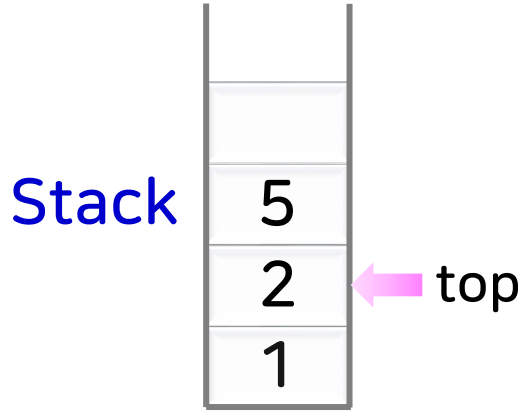


방문 순서

깊이 우선 탐색_예_1

정점 2의 주변 정점을 선택 → 정점 5 → push

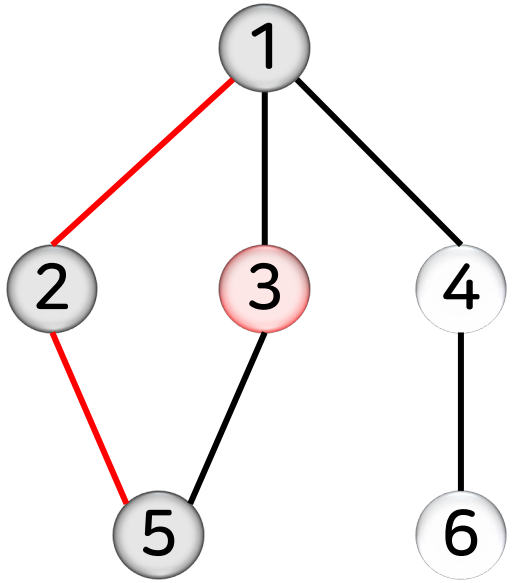


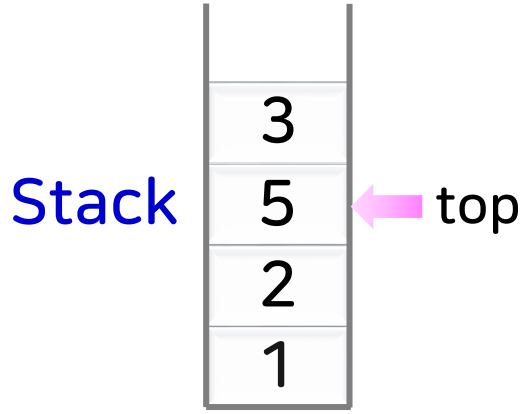


방문 순서

깊이 우선 탐색_예_1

정점 5의 주변 정점을 선택 → 정점 3 → push

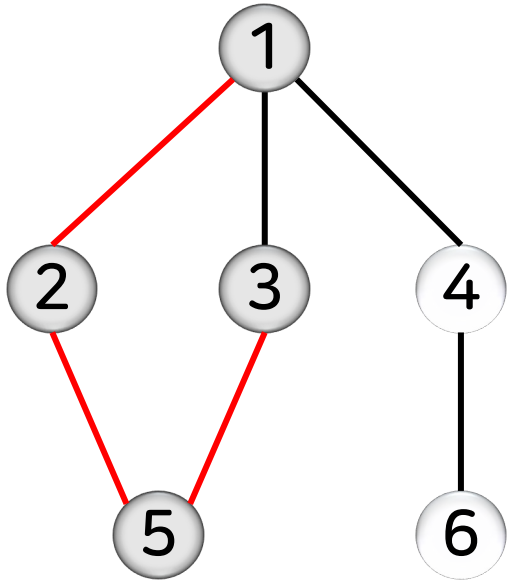


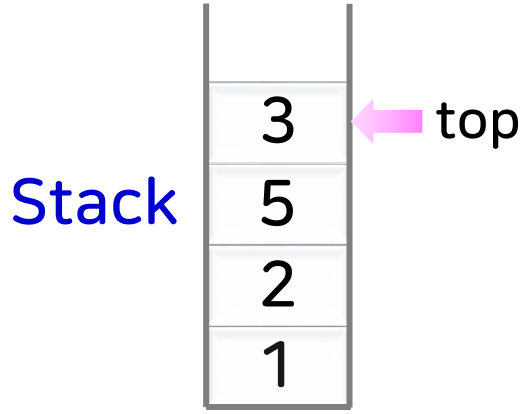


방문 순서

깊이 우선 탐색_예_1

최근 방문 정점(3)의 주변 정점이 없음 → pop → 정점 3으로 되돌아감



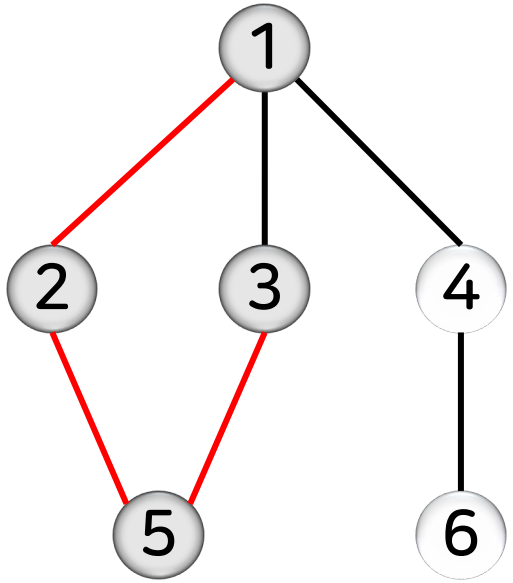


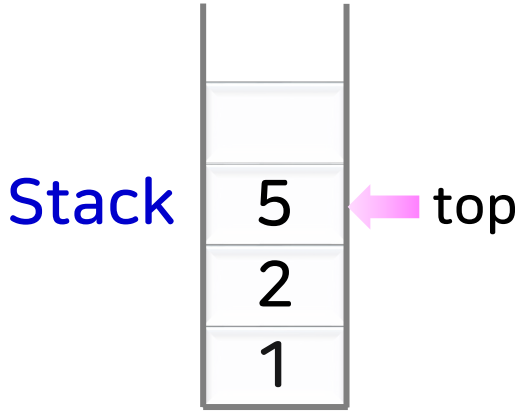
1	2	5	3		

방문 순서

깊이 우선 탐색_예_1

정점 3의 주변 정점이 없음 → pop → 정점 5으로 되돌아감

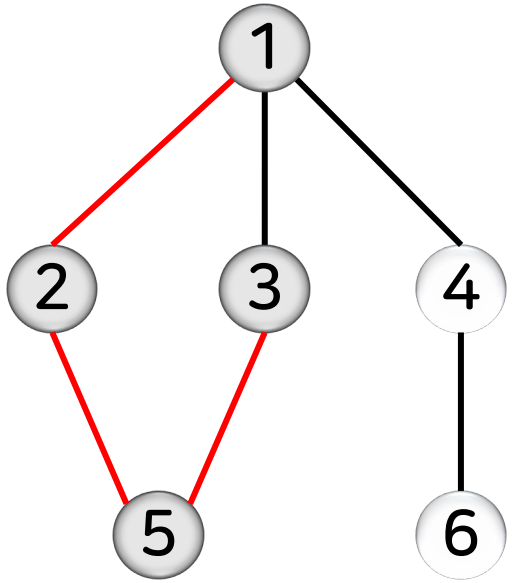


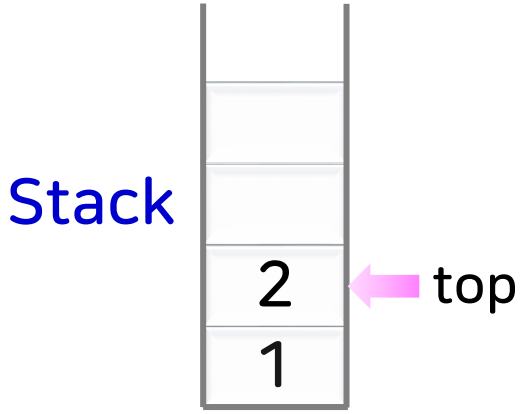


방문 순서

깊이 우선 탐색_예_1

정점 5의 주변 정점이 없음 → pop → 정점 2으로 되돌아감

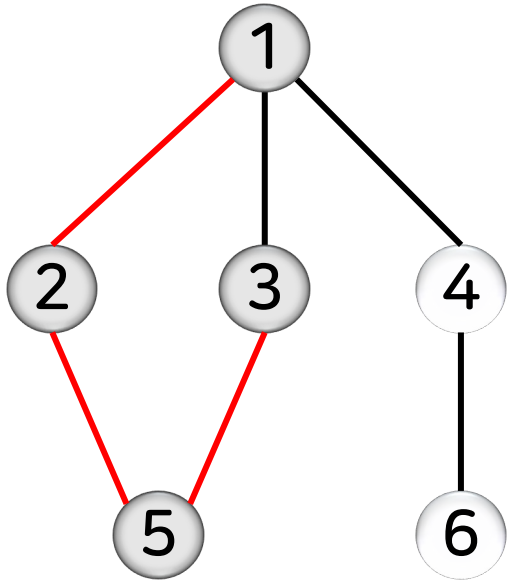


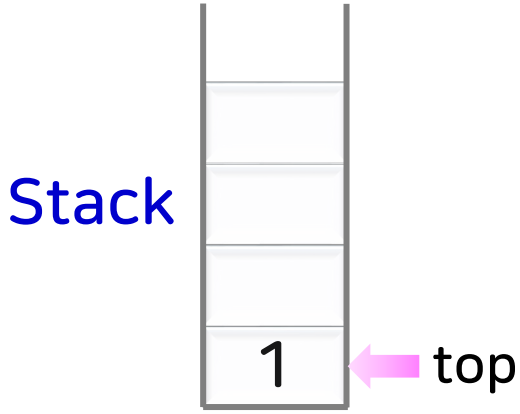


방문 순서

깊이 우선 탐색_예_1

정점 2의 주변 정점이 없음 → pop → 정점 1으로 되돌아감

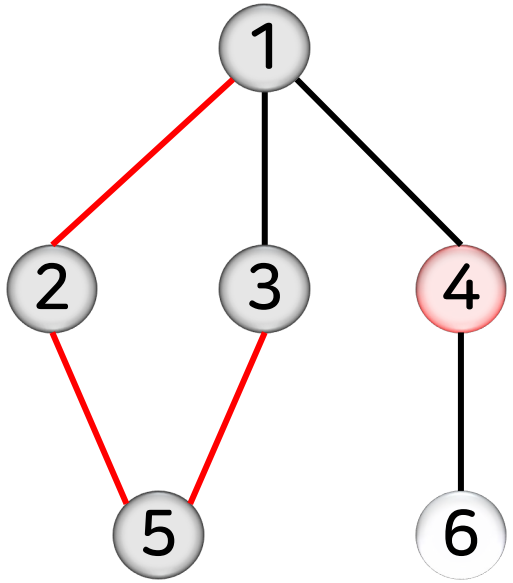


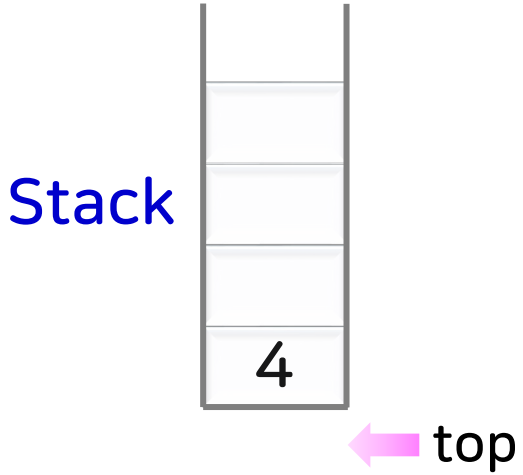


방문 순서

깊이 우선 탐색_예_1

정점 1의 주변 정점을 선택 → 정점 4 → push



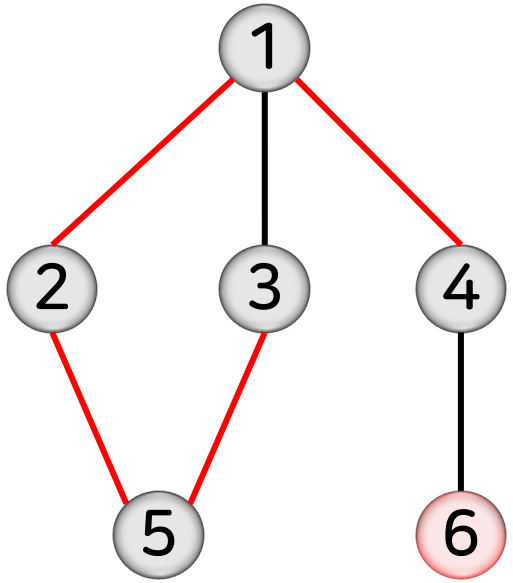


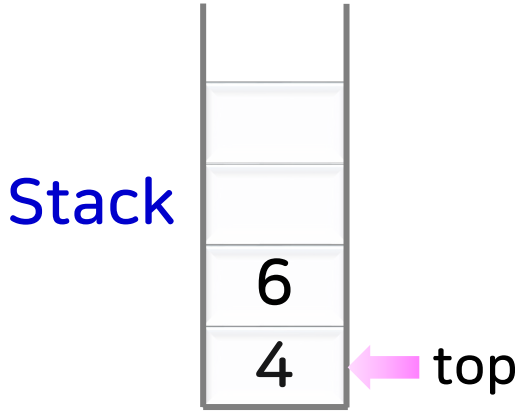
1	2	5	3	4	

방문 순서

깊이 우선 탐색_예_1

정점 4의 주변 정점을 선택 → 정점 6 → push



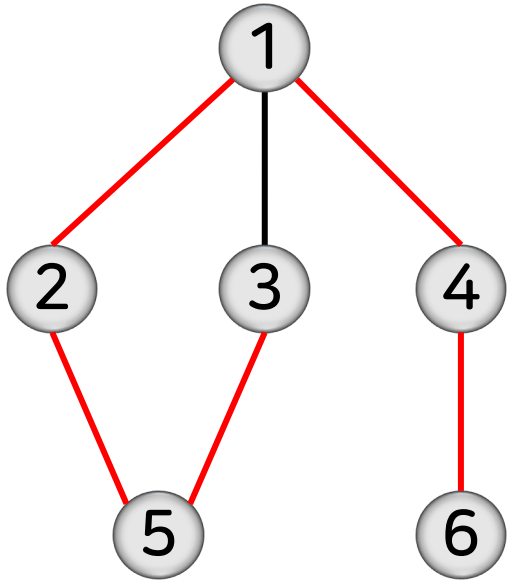


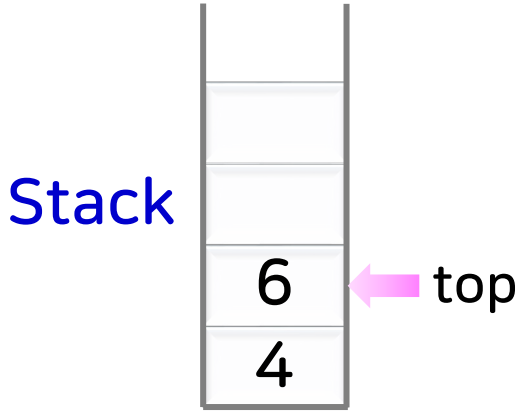
1	2	5	3	4	6

방문 순서

깊이 우선 탐색_예_1

최근 방문 정점의 주변 정점이 없음 → pop으로 정점 6으로 되돌아감



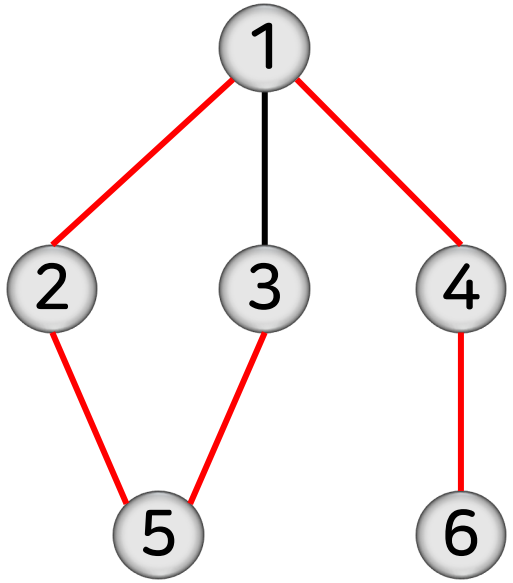


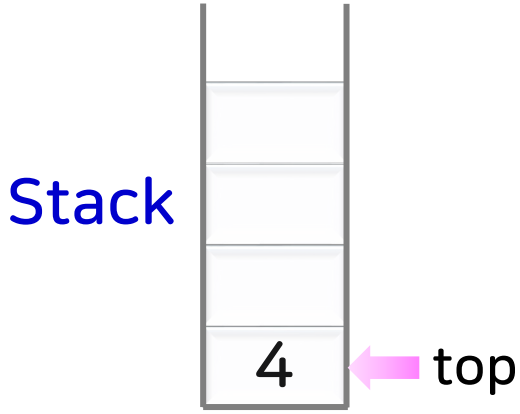
1	2	5	3	4	6

방문 순서

깊이 우선 탐색_예_1

정점 6의 주변 정점이 없음 → pop으로 정점 4으로 되돌아감



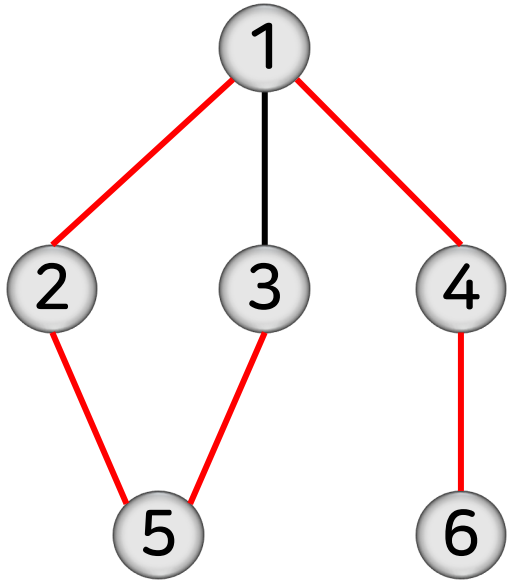


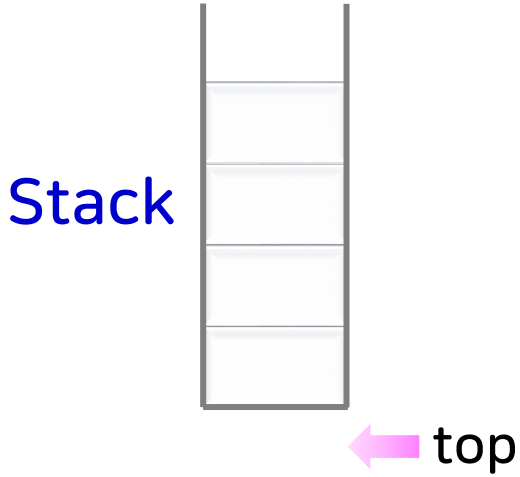
1	2	5	3	4	6

방문 순서

깊이 우선 탐색_예_1

스택 = NULL → 종료





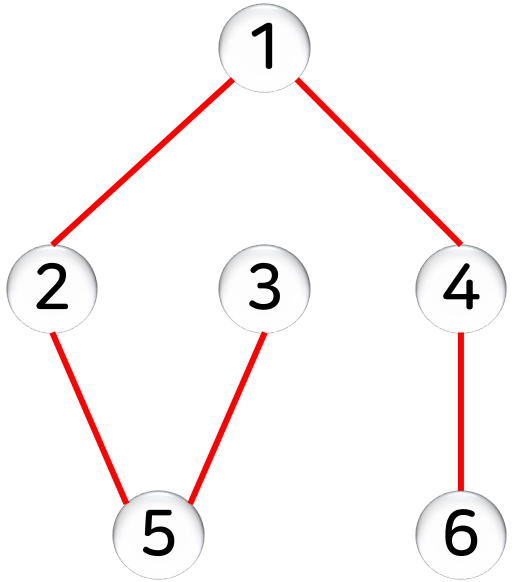
1	2	5	3	4	6

방문 순서

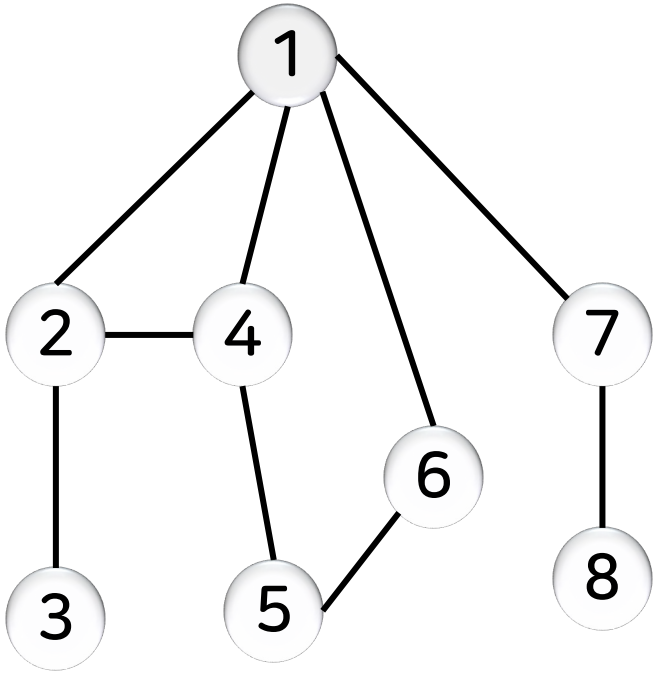
깊이 우선 탐색

깊이 우선 트리

- DFS를 수행하는 과정에서 방문한 정점과 그때 사용된 간선으로 구성되는 트리



깊이 우선 탐색_예_2



방문 순서?

1,2,3,4,5,6,7,8

1,2,4,5,6,3,7,8

1,4,2,3,5,6,7,8

1,4,5,6,2,3,7,8

1,6,5,4,2,3,7,8

1,7,8,2,3,4,5,6

1,7,8,2,4,5,6,3

1,7,8,4,2,3,5,6

1,7,8,4,5,6,2,3

1,7,8,6,5,4,2,3

깊이 우선 탐색의 성능과 특징

인접 리스트 → O(|V|+|E|), 인접 행렬 → O($|V|^2$)

순환 알고리즘

```
DFS_recursion (G, current) {
  current.visited = TRUE;
  방문 정점으로 current 출력;
  for (next ← current의 모든 인접한 정점) {
     if (next.visited == FALSE)
        DFS_recursion(G, next);
  }
}
```

너비 우선 탐색

시작 정점을 기준으로 거리가 가장 가깝게 인접한 정점을 우선으로 모두 방문한 후 시작 정점과의 거리가 점점 멀어지는 순서로 인접 정점들을 탐색하는 방법

- "거리" → 시작 정점으로부터의 경로의 길이

→ 주변 정점 중에서 가장 오래된 것부터 우선 방문하는 방법

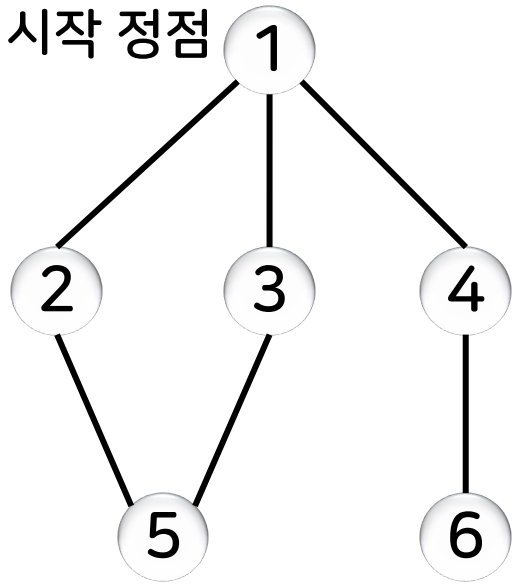
큐를 사용하여 주변 정점을 관리

너비 우선 탐색 알고리즘


```
BreadthFirstSearch (G, s)
{
  Enqueue(Queue, s);
  s.flag = TRUE;
  while (Queue != NULL) {
    c = Dequeue(Queue);
    정점 c를 방문 정점으로 출력;
    for (v ← c의 모든 인접한 정점)
      if (v.flag == FALSE) {
        Enqueue(Queue, v);
        v.flag = TRUE;
      }
  }
}
```

너비 우선 탐색_예_1

시작 정점 1 → 큐에 삽입하고 탐색 시작

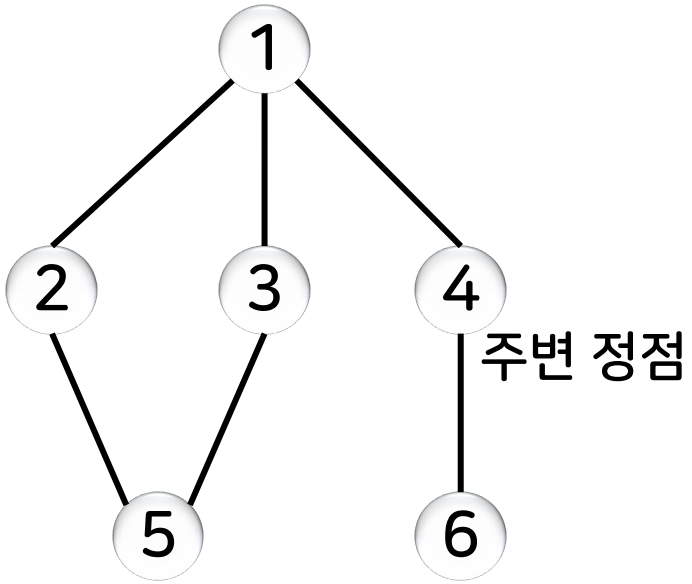


Queue

방문 순서

너비 우선 탐색_예_1

큐에서 삭제한 정점 1를 방문 → 주변 정점(2,3,4)을 큐에 삽입

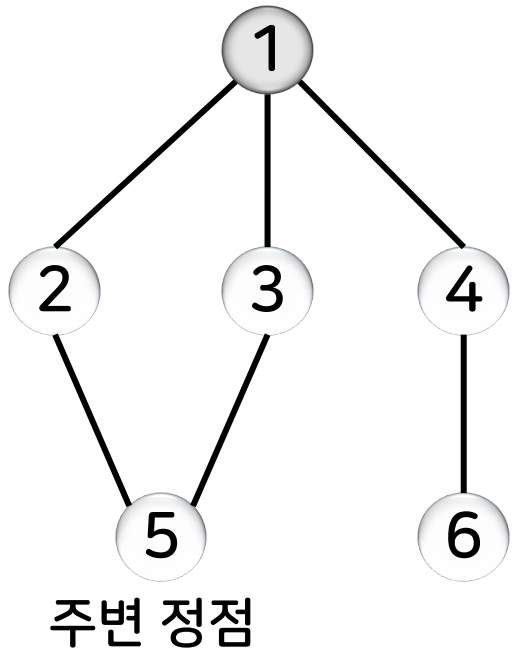


Queue

방문 순서

너비 우선 탐색_예_1

큐에서 삭제한 정점 2를 방문 → 주변 정점(5)을 큐에 삽입

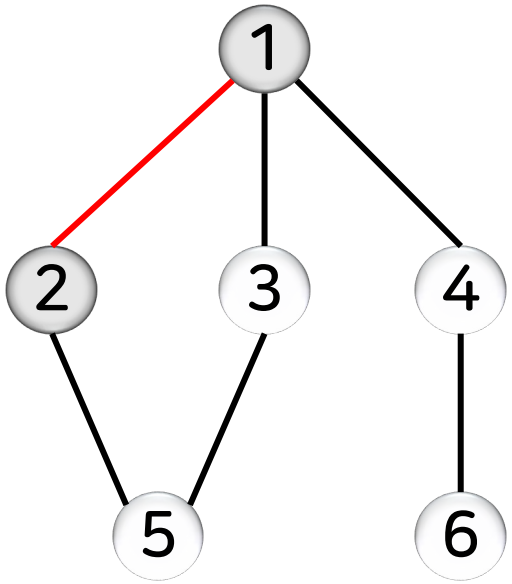


Queue

방문 순서

너비 우선 탐색_예_1

큐에서 삭제한 정점 3을 방문 → 주변 정점 없음

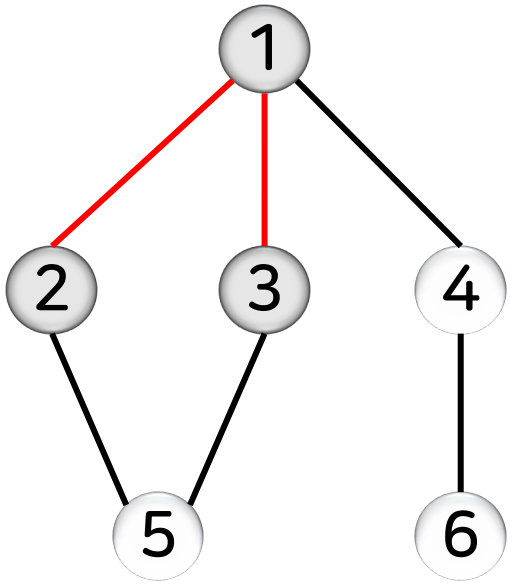


Queue

방문 순서

너비 우선 탐색_예_1

큐에서 삭제한 정점 4를 방문 → 주변 정점(6)을 큐에 삽입

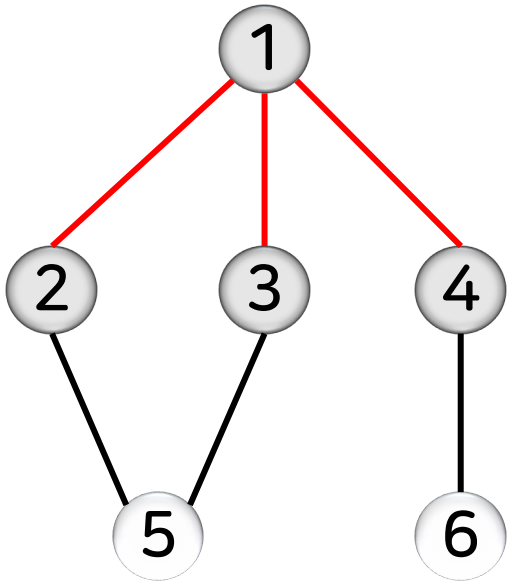


Queue

방문 순서

너비 우선 탐색_예_1

큐에서 삭제한 정점 5를 방문 → 주변 정점 없음

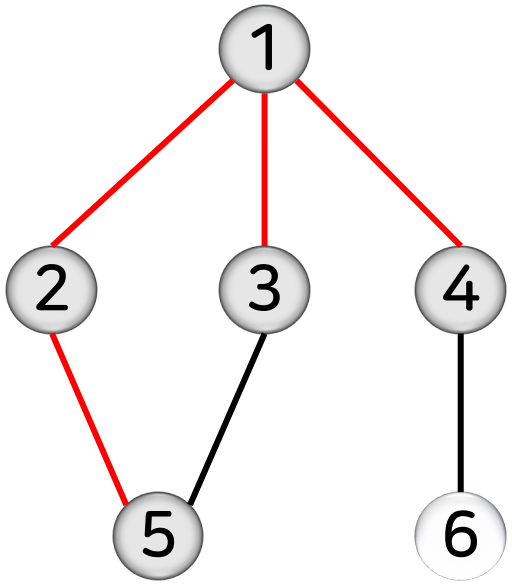


Queue

방문 순서

너비 우선 탐색_예_1

큐에서 삭제한 정점 6을 방문 → 주변 정점 없음 → 탐색 완료

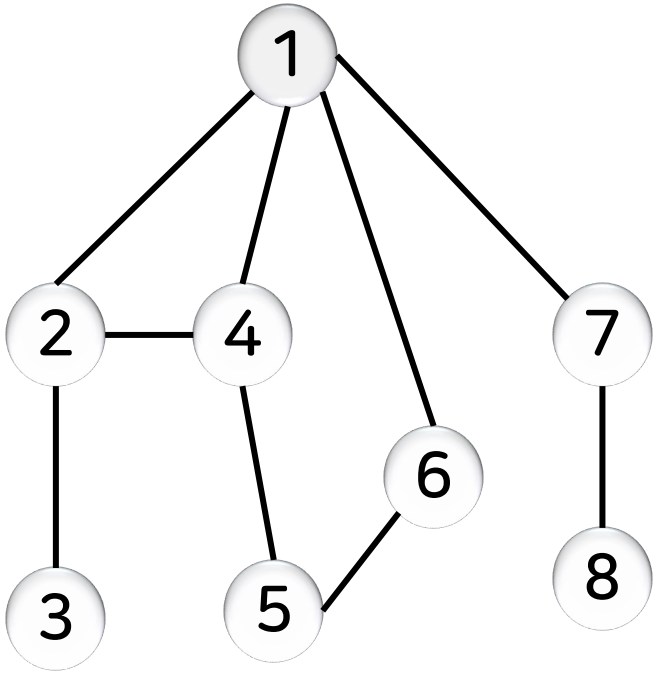


Queue

방문 순서

1	2	3	4	5	6

너비 우선 탐색_예__2



방문 순서?

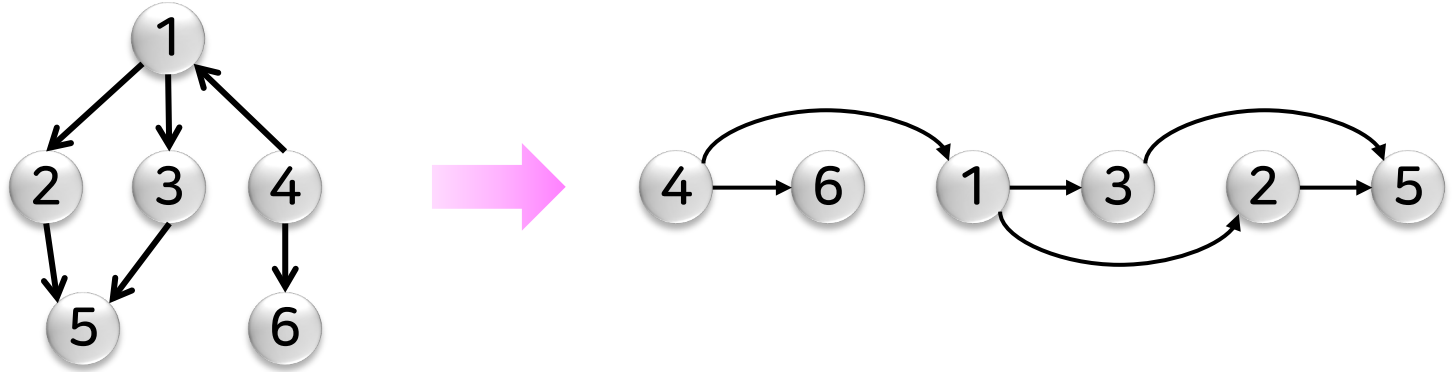
1, [2/4/6/7], [3/5/8]

정점 1의 주변 정점 2,4,6,7를
방문하는 순서에 따라 24가지의 경우가 존재

03. 그래프 순회의 응용

위상 정렬

무사이클 방향 그래프(DAG, Directed Acyclic Graph)에서 모든 간선이 한 방향으로만 향하도록 정점을 한 줄로 나열하는 것

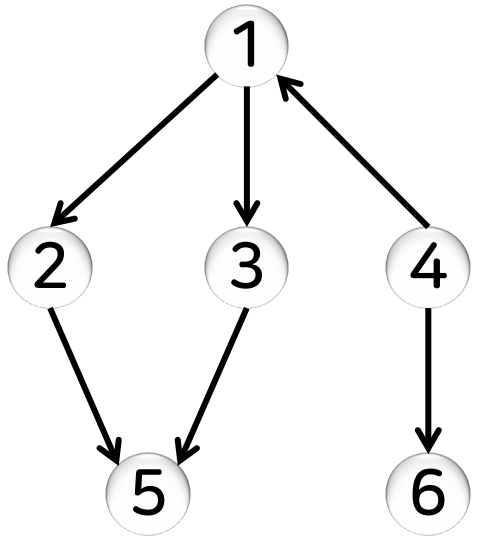


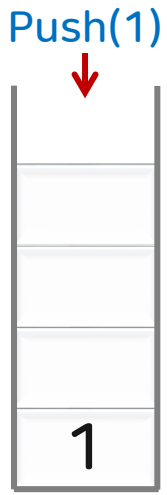
깊이 우선 탐색을 활용하여 구함

- DFS를 수행하다가 더 이상 주변 정점이 없어서 되돌아갈 때, 스택에서 삭제되는 정점을 역순으로 나열하면 됨

위상 정렬_예

시작 정점 1 → Push(1)



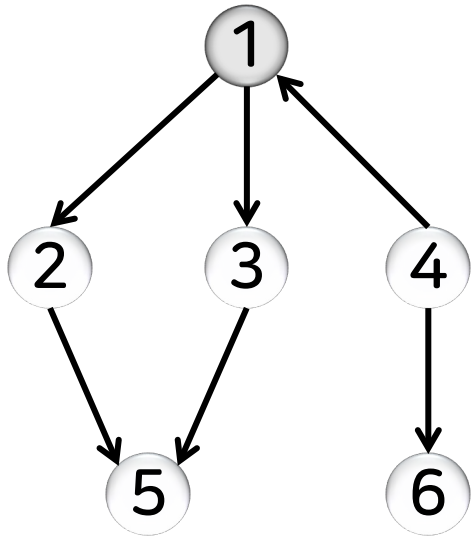


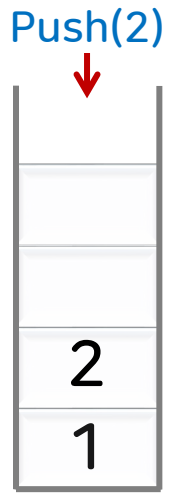
Stack

정렬 순서

위상 정렬_예

스택 탑에 있는 정점의 주변 정점 선택 → Push(2)



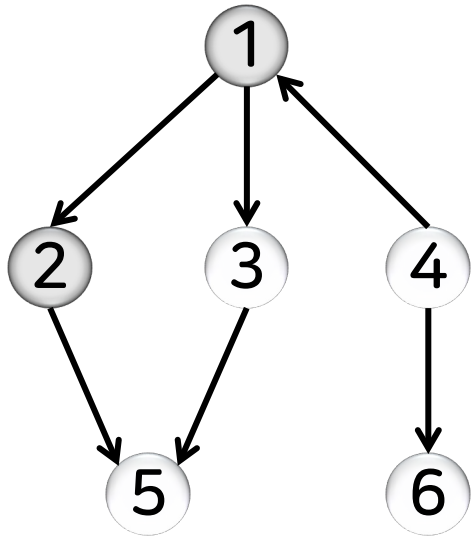


Stack

정렬 순서

위상 정렬_예

스택 탑에 있는 정점의 주변 정점 선택 → Push(5)

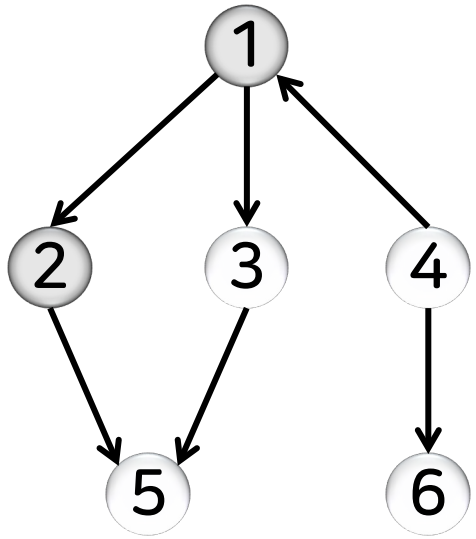


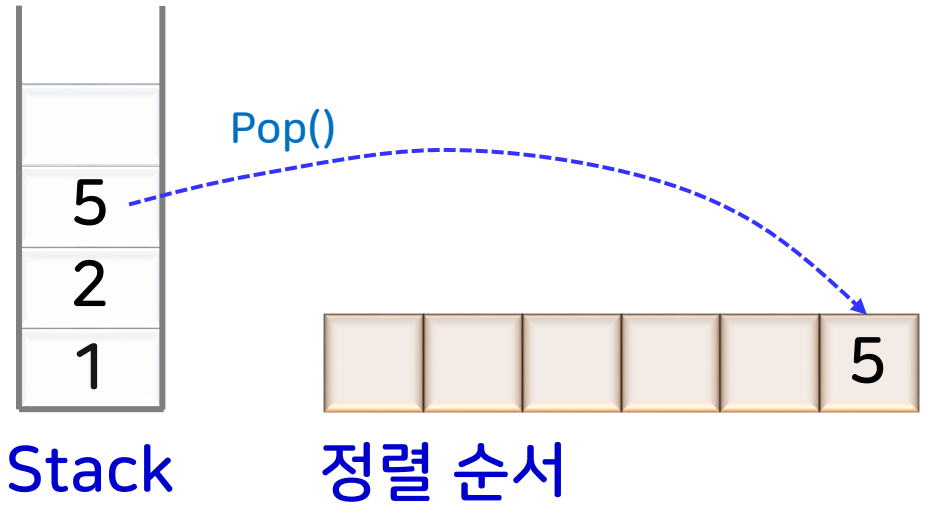
Stack

정렬 순서

위상 정렬_예

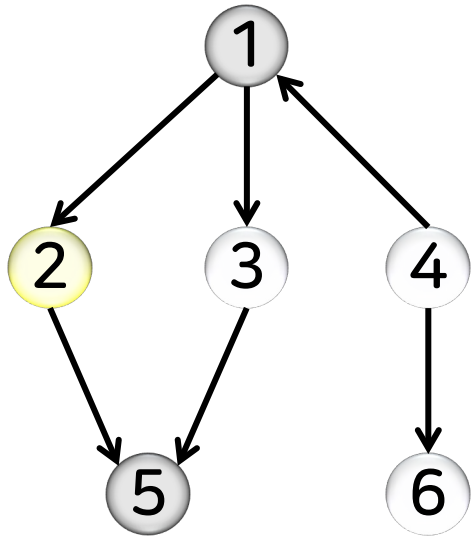
스택 탑에 있는 정점의 주변 정점 선택 → 없음 → Pop()

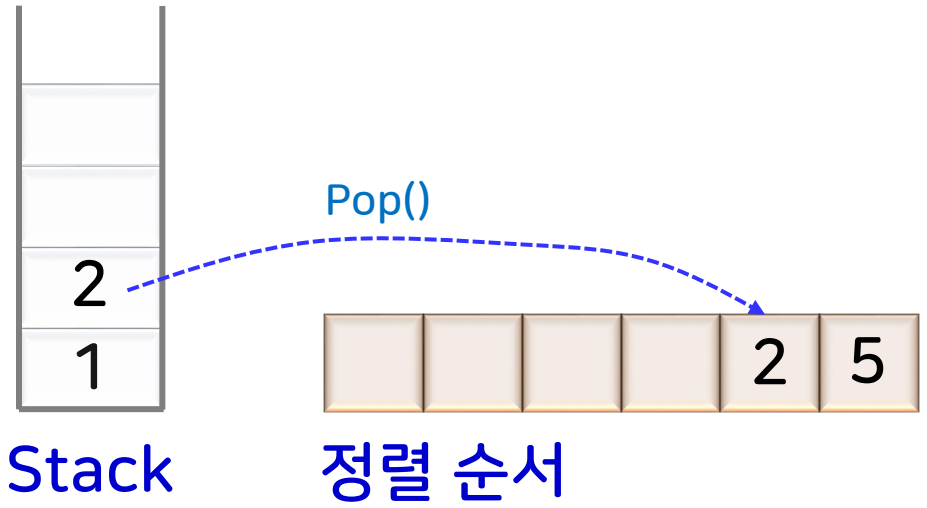




위상 정렬_예

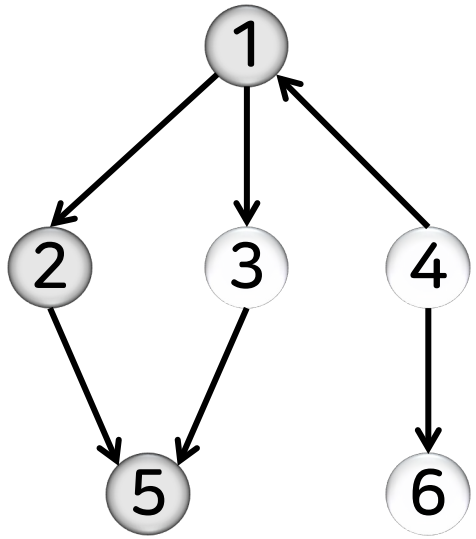
스택 탑에 있는 정점의 주변 정점 선택 → 없음 → Pop()

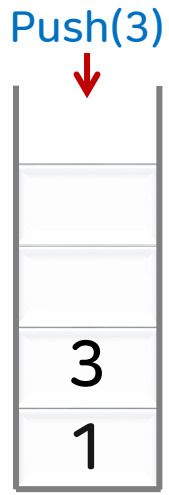




위상 정렬_예

스택 탑에 있는 정점의 주변 정점 선택 → Push(3)



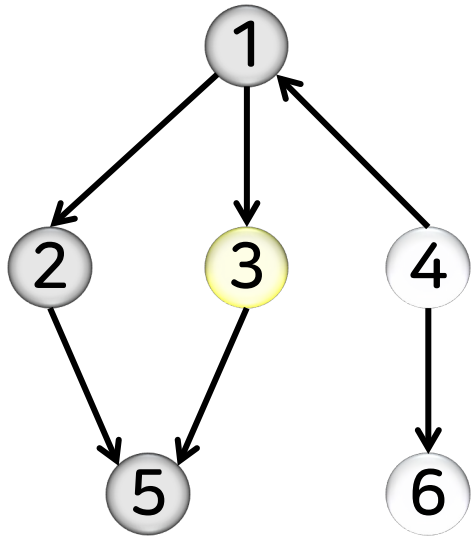


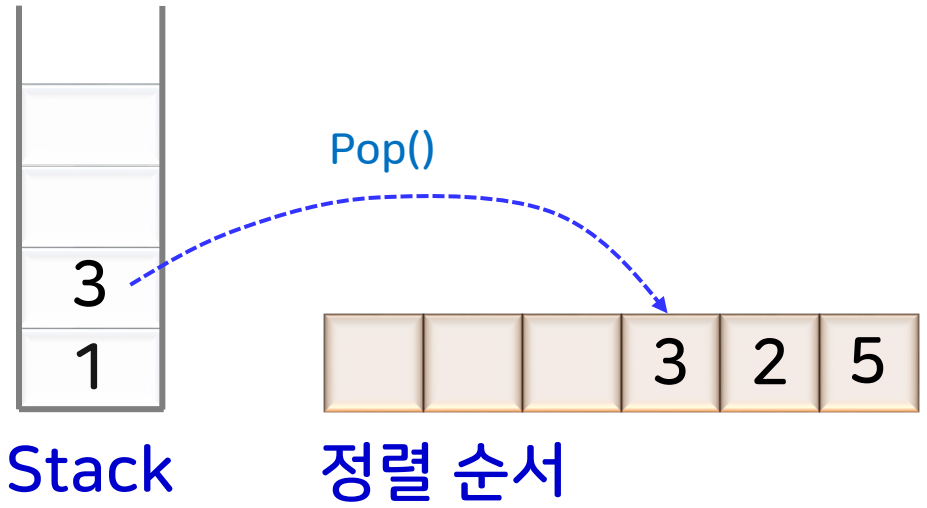
Stack

정렬 순서

위상 정렬_예

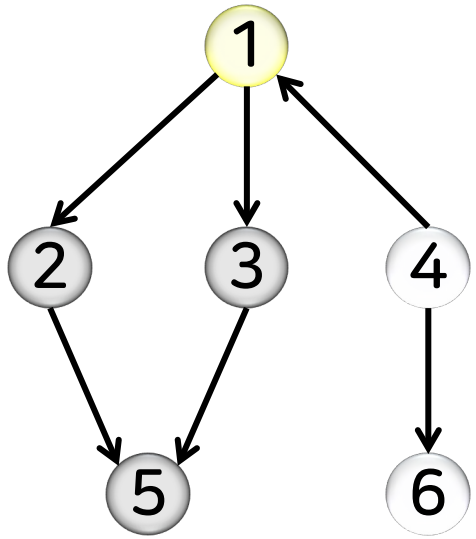
스택 탑에 있는 정점의 주변 정점 선택 → 없음 → Pop()

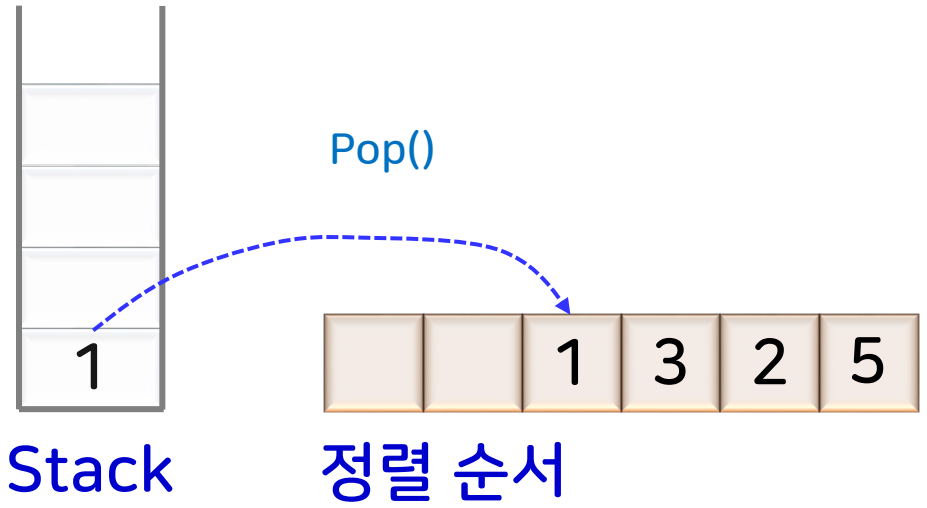




위상 정렬_예

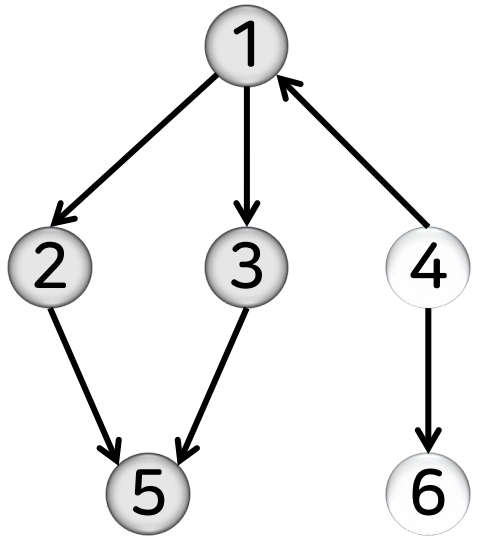
스택 탑에 있는 정점의 주변 정점 선택 → 없음 → Pop()

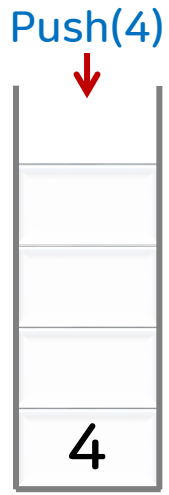




위상 정렬_예

스택이 비었으므로 아직 탐색하지 않은 정점을 찾음 → Push(4)



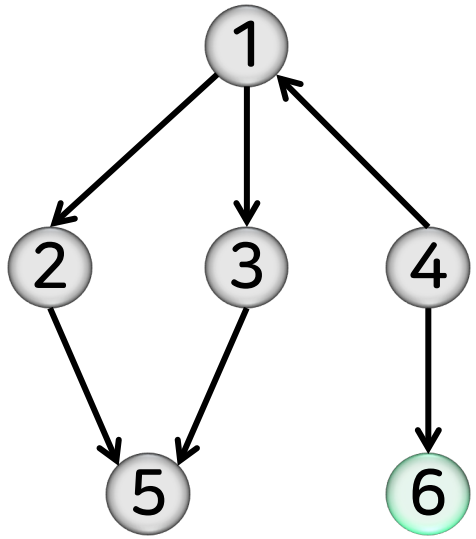


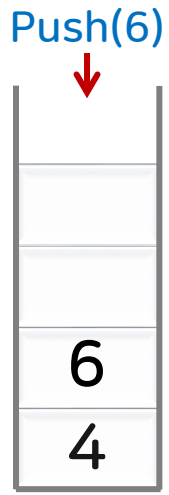
Stack

정렬 순서

위상 정렬_예

스택 탑에 있는 정점의 주변 정점 선택 → Push(6)



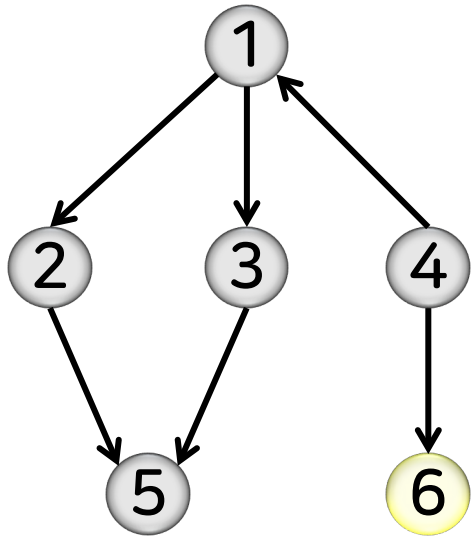


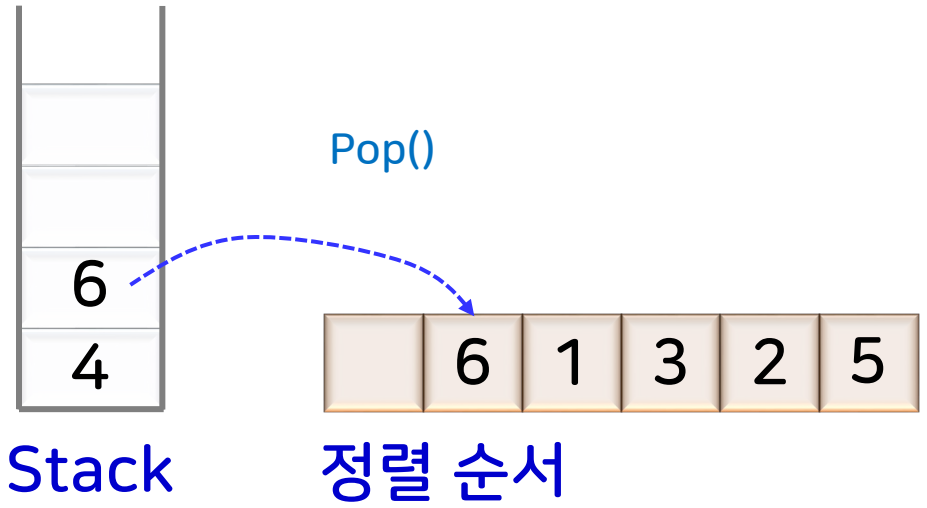
Stack

정렬 순서

위상 정렬_예

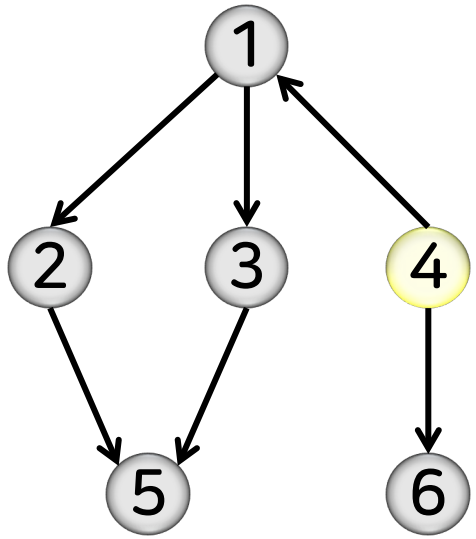
스택 탑에 있는 정점의 주변 정점 선택 → 없음 → Pop()

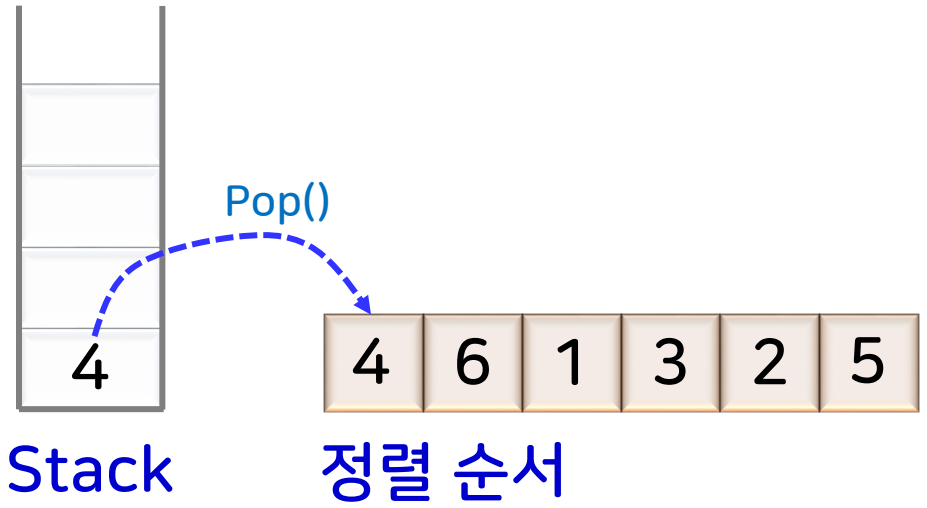




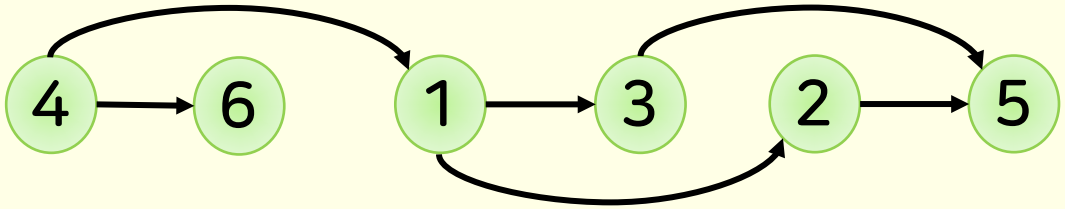
위상 정렬_예

스택 탑에 있는 정점의 주변 정점 선택 → 없음 → Pop()





위상 정렬 결과



그래프의 연결성

정점 간의 연결 관계를 다루는 것

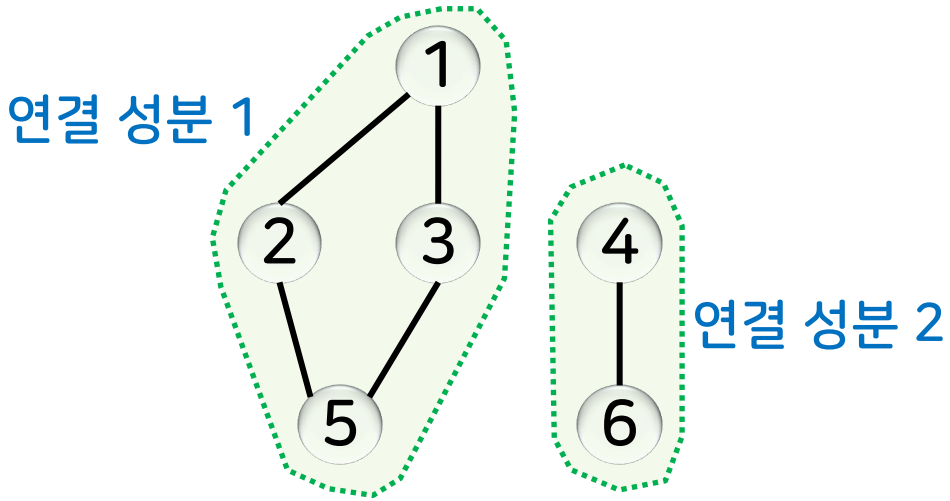
- 연결 성분
  - ✓ connected component
- 강연결 성분
  - ✓ strongly connected component

연결 성분

무방향 그래프에서 임의의 두 정점 간의 경로가 존재하는 최대 부분 그래프

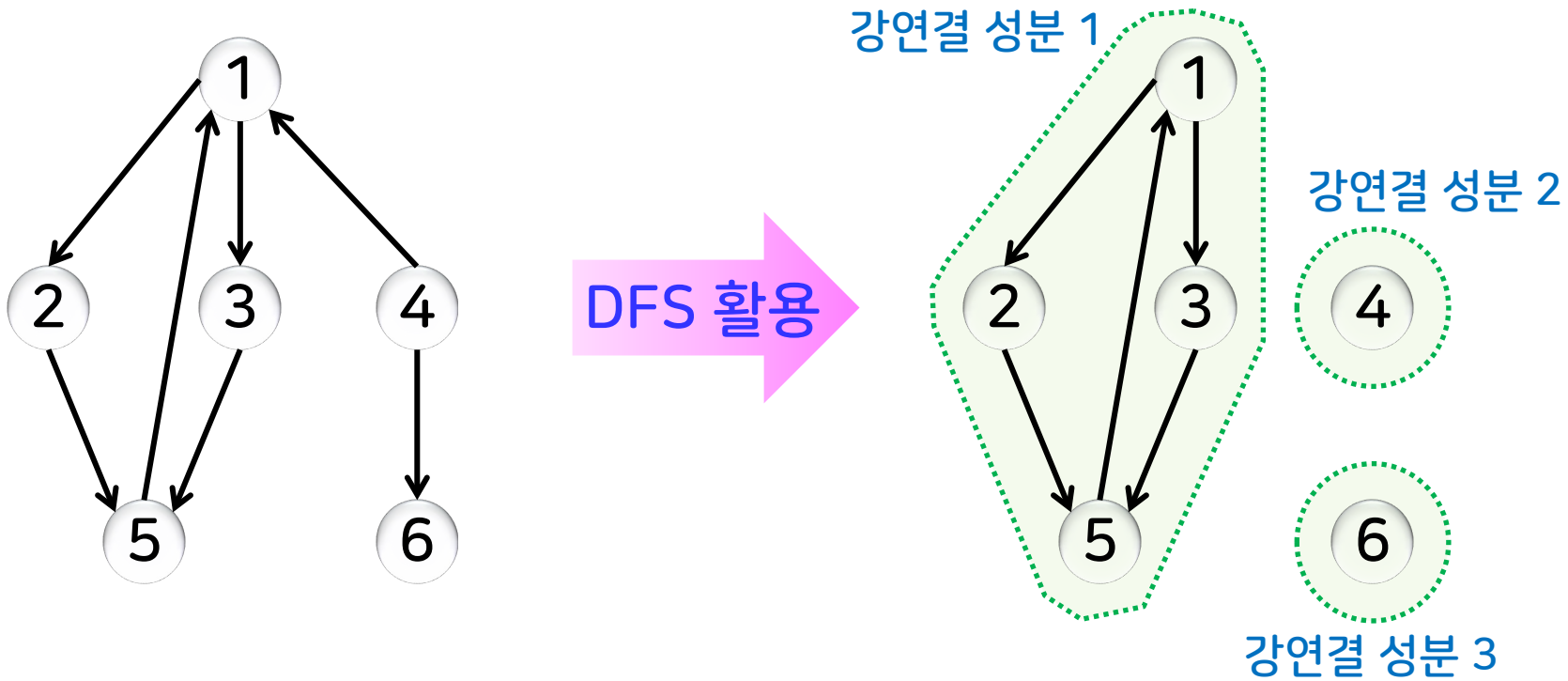
- 너비 우선 탐색 또는 깊이 우선 탐색을 활용하여 구함

✓ while (아직 탐색하지 않은 정점이 존재)
탐색을 수행하다가 큐/스택이 비게 되면
그때까지 탐색한 정점들을 하나의 연결 성분으로 구성



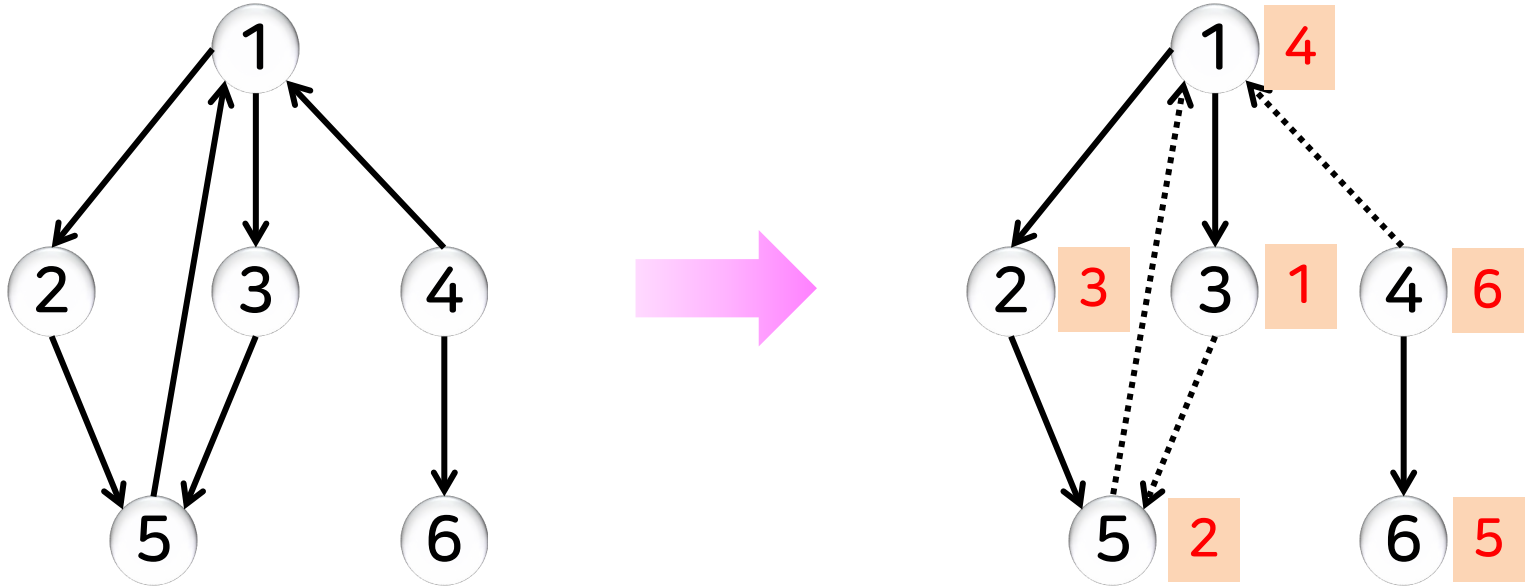
강연결 성분

방향 그래프에서 임의의 두 정점 사이에 양방향의 경로가 존재하는 최대 부분 그래프



강연결 성분_예

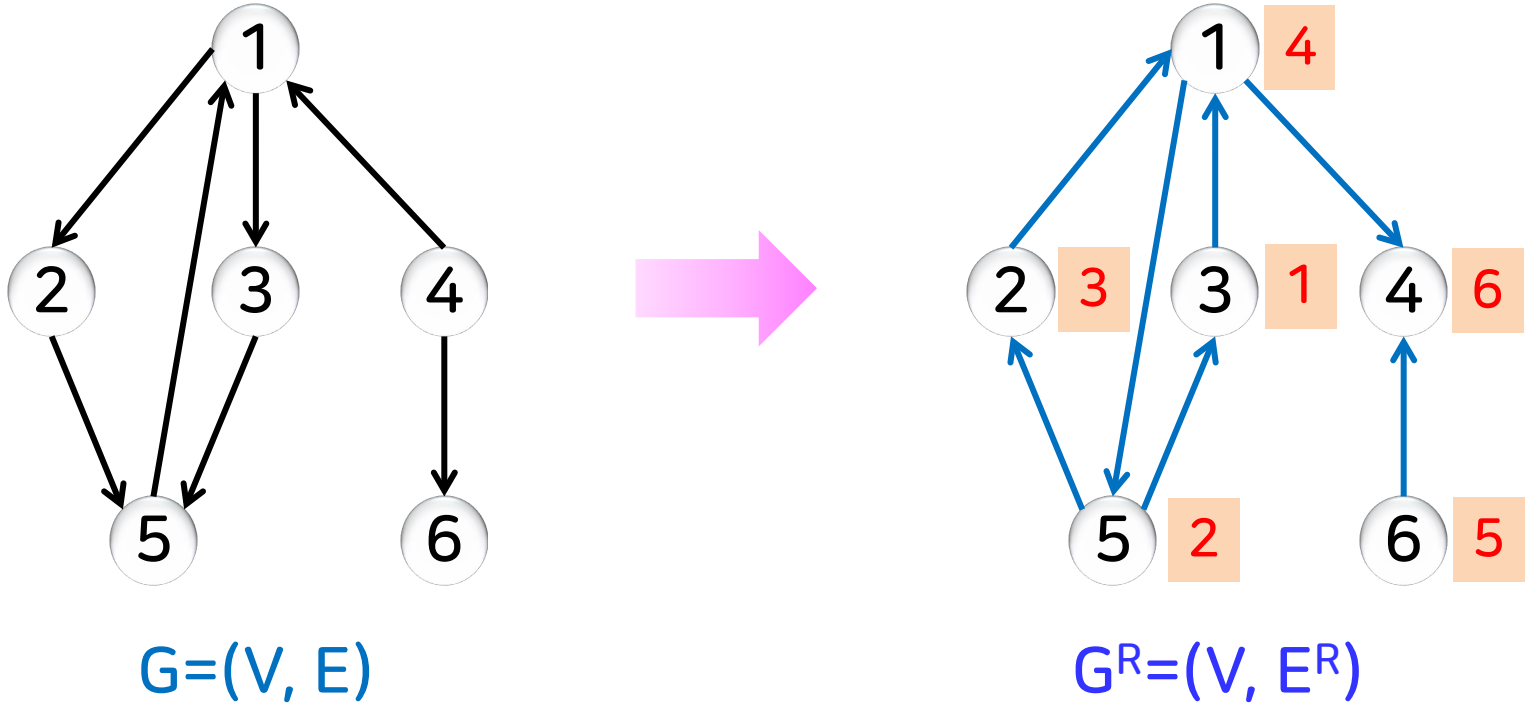
① 깊이 우선 탐색을 적용하여 정점의 방문 완료 순서(방문 순서의 역순)를 구함



G=(V, E)

강연결 성분_예

② G=(V,E)에 대해 E^R={<v,u>|<u,v>∈E}을 사용한 G^R=(V, E^R)을 구함

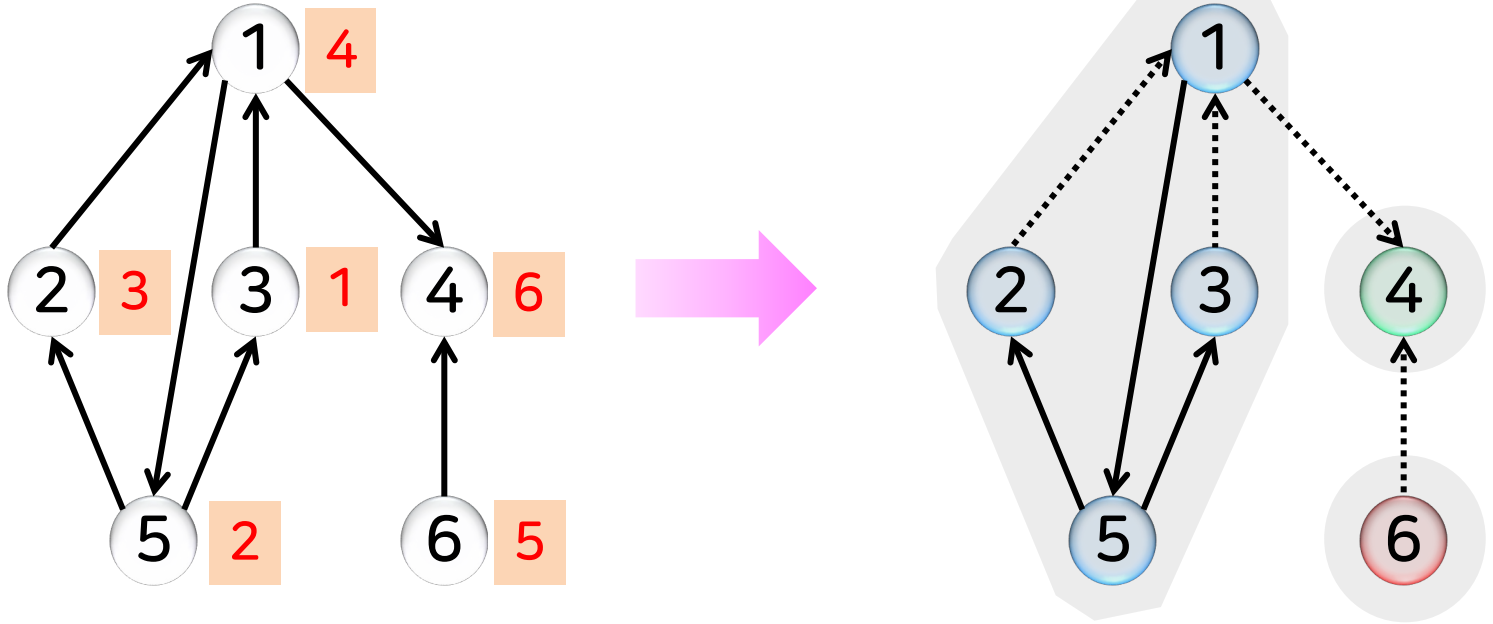


G=(V, E)

G^R=(V, E^R)

강연결 성분_예

③ G^R에 대해서 방문 완료 번호가 큰 것부터
DFS를 수행하여 갈 수 있는 정점들의 각 리스트가 강연결 성분이 됨



G^R

강연결 성분 → {1, 2, 3, 5}, {4}, {6}

정리하기

1. 기본 개념

- G=(V, E), 무방향 그래프 vs 방향 그래프, 가중 그래프
- 인접, 부수, 부분 그래프, 경로, 경로의 길이, 차수, 단순경로, 사이클, 연결 등
- 구현 방법 → 인접 행렬, 인접 리스트

2. 그래프 순회

- **깊이 우선 탐색** → 최근의 주변 정점을 우선으로 방문하는 방법, 스택
- **너비 우선 탐색** → 주변 정점 중에서 가장 오래된 것부터 우선 방문, 큐

3. 그래프 순회의 응용

- **위상 정렬** → 무사이클 방향 그래프에서 모든 간선의 방향이 한 방향으로만 향하도록 정점을 한 줄로 나열하는 것 → 깊이 우선 탐색 활용
- **연결 성분** → 무방향 그래프에서 임의의 두 정점 간의 경로가 존재하는 최대 부분 그래프
- **강연결 성분** → 방향 그래프에서 임의의 두 정점 사이에 양방향의 경로가 존재하는 최대 부분 그래프 → $G^R=(V,E^R)$에 대해 G의 방문 완료 번호가 큰 것부터 DFS를 적용

ALGORITHM
알고리즘

다음시간에는

Lecture 09

그래프 (2)

컴퓨터과학과 | 이관용 교수