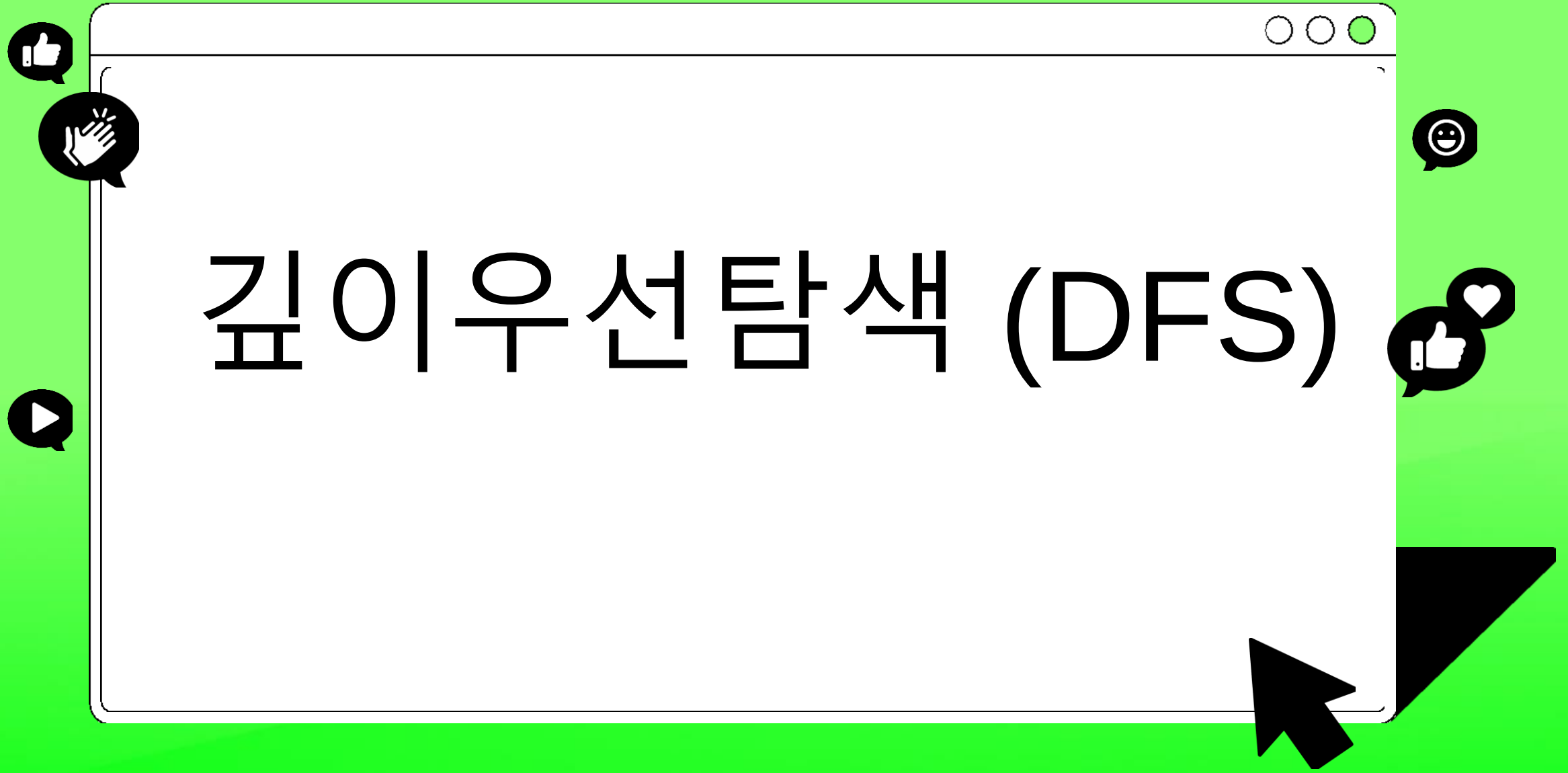
HAPPY HACKING



# 깊이우선탐색 (DFS)

1. 그래프 탐색 알고리즘
2. 깊이우선탐색(DFS)

3. DFS의 동작 과정

4. DFS의 구현 방식

5. DFS 문제 풀이

6. 이차원 격자에서의 DFS

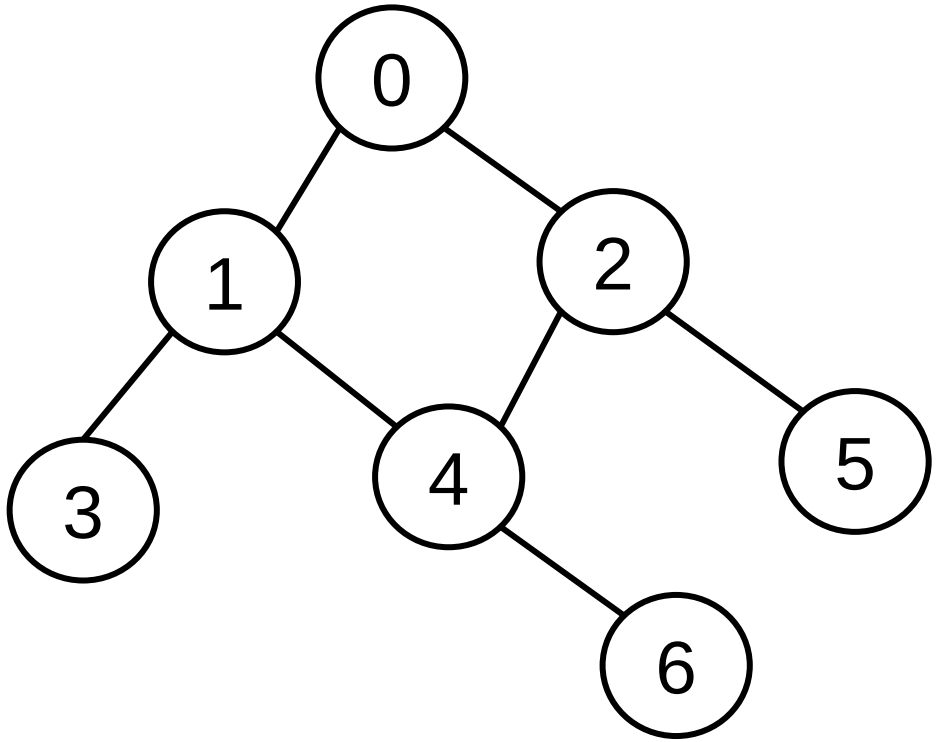
HAPPY HACKING

# 1. 그래프 탐색 알고리즘

# 1. 그래프 탐색 알고리즘

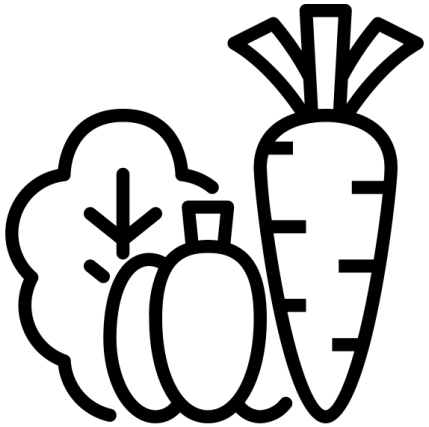
HAPPY HACKING

지난 시간에 그래프 데이터 구조에 대해 학습하였다.



HAPPY HACKING

데이터 구조는 알고리즘의 재료가 되어 문제를 해결하는데 사용된다.

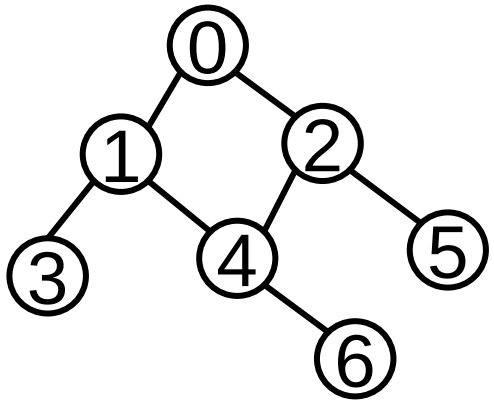


데이터 구조

알고리즘

HAPPY HACKING

그렇다면 그래프 데이터 구조는 어떤 알고리즘에 활용할 수 있을까?



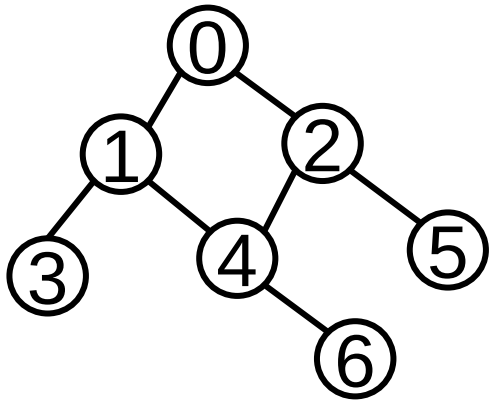

데이터 구조

?

알고리즘

# 1. 그래프 탐색 알고리즘

그래프 자료구조는 탐색 알고리즘에 활용된다!




데이터 구조

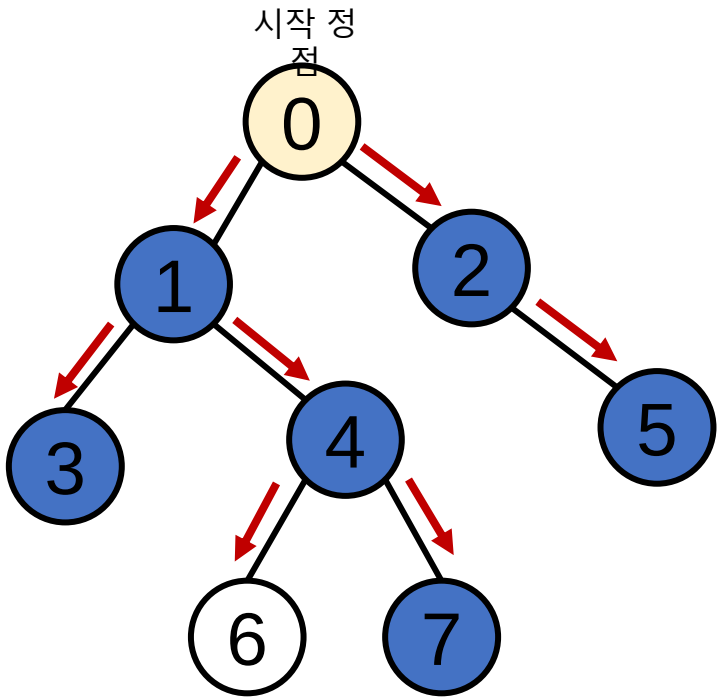
그래프 탐색

알고리즘

HAPPY HACKING

## 그래프 탐색 알고리즘이란?

시작 정점에서 간선을 타고 이동할 수 있는 모든 정점을 찾는 알고리즘



# 1. 그래프 탐색 알고리즘

HAPPY HACKING

그래프 탐색 알고리즘에는 깊이우선탐색과 너비우선탐색이 있다.

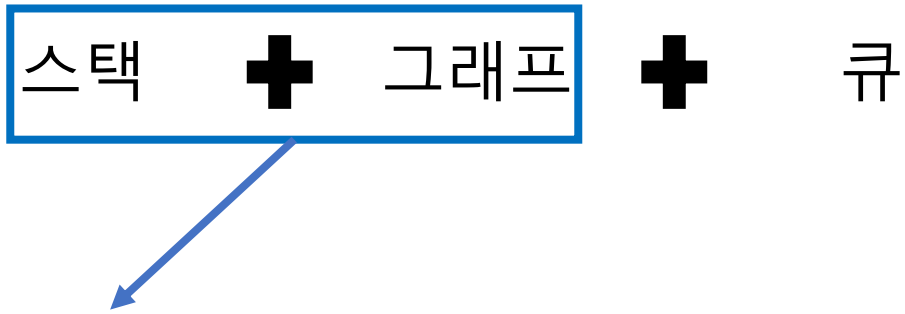
이전에 학습했던 스택과 큐 자료구조의 개념을 함께 활용한다.

스택 ✚ 그래프 ✚ 큐

# 1. 그래프 탐색 알고리즘

그래프 탐색 알고리즘에는 깊이우선탐색과 너비우선탐색이 있다.

이전에 학습했던 스택과 큐 자료구조의 개념을 함께 활용한다.



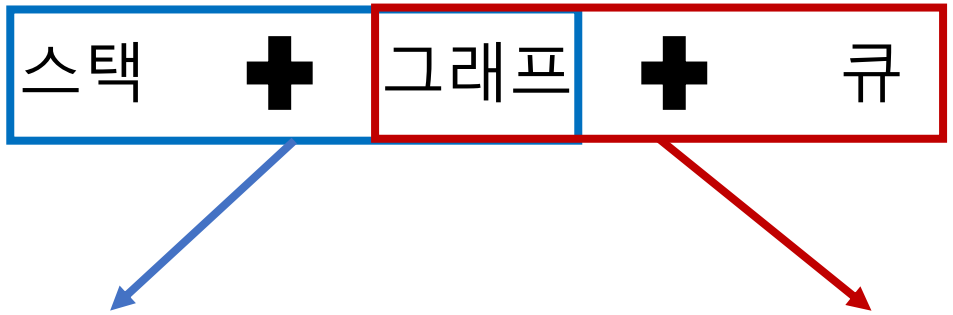

깊이우선탐색 (Depth-First Search, DFS)
그래프의 깊이를 우선으로 탐색하기 위해
스택의 개념을 활용한다.

# 1. 그래프 탐색 알고리즘

HAPPY HACKING

## 그래프 탐색 알고리즘에는 깊이우선탐색과 너비우선탐색이 있다.

이전에 학습했던 스택과 큐 자료구조의 개념을 함께 활용한다.

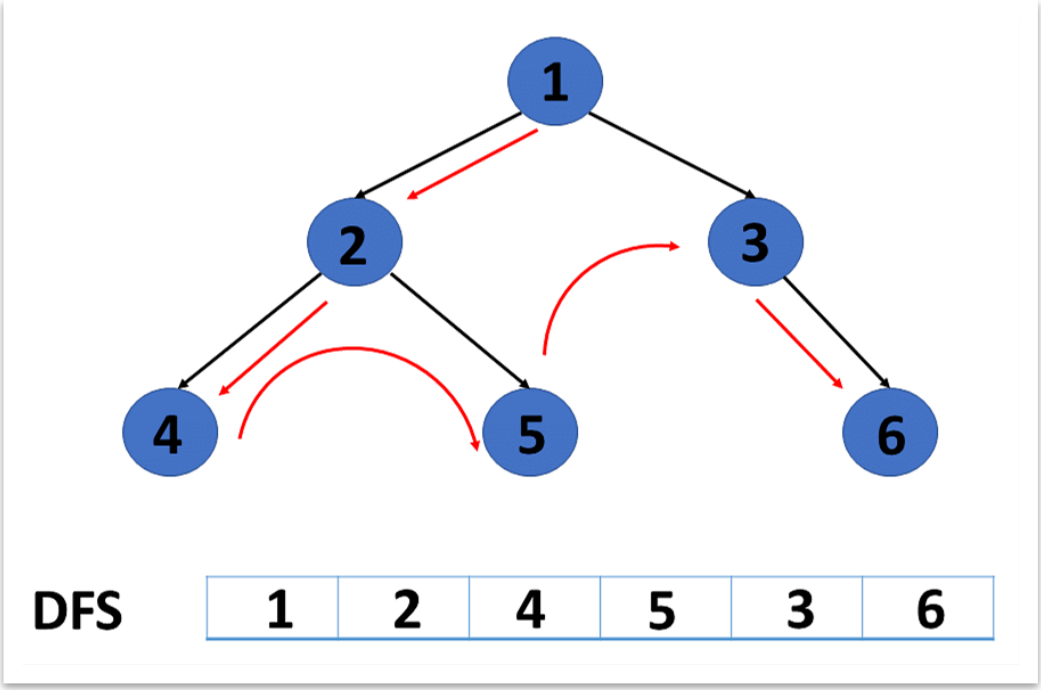


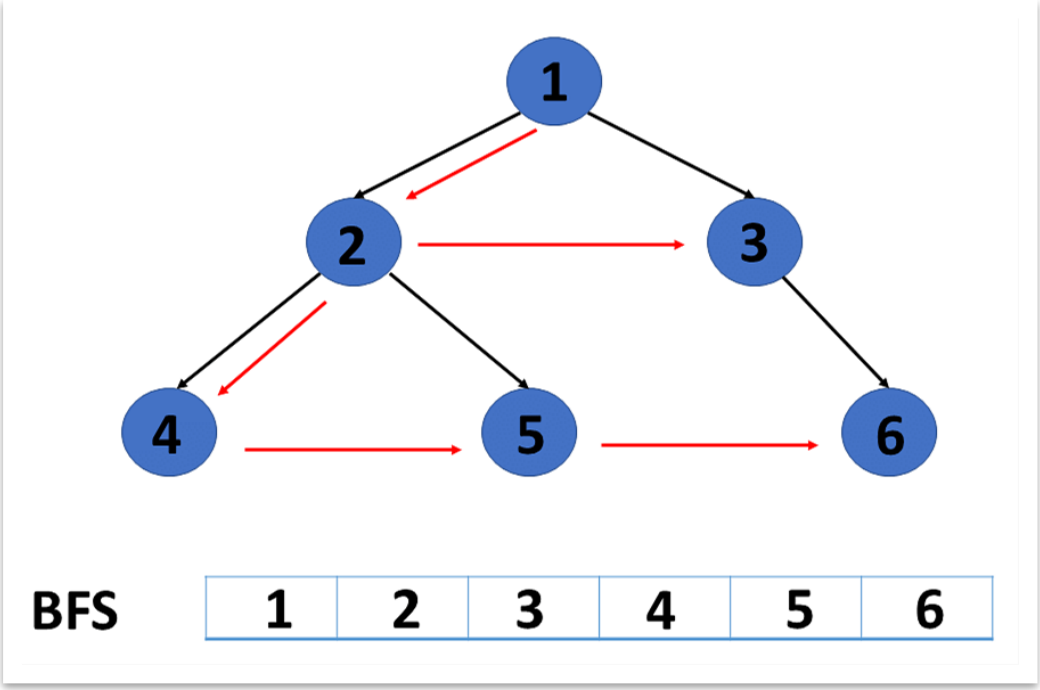

깊이우선탐색 (Depth-First Search, DFS)
그래프의 깊이를 우선으로 탐색하기 위해
스택의 개념을 활용한다.

너비우선탐색 (Breadth-First Search, BFS)
그래프의 너비를 우선으로 탐색하기 위해
큐의 개념을 활용한다.

HAPPY HACKING

## 깊이우선탐색(DFS) vs 너비우선탐색(BFS)



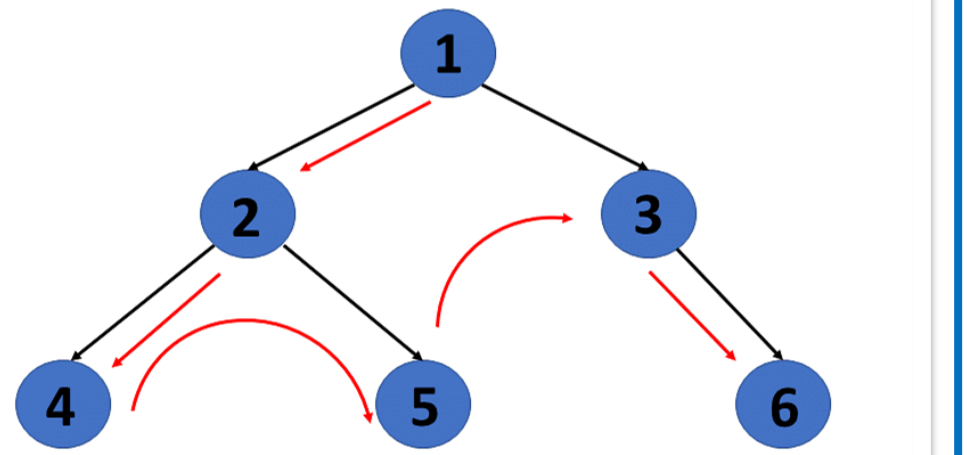

https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

# 1. 그래프 탐색 알고리즘

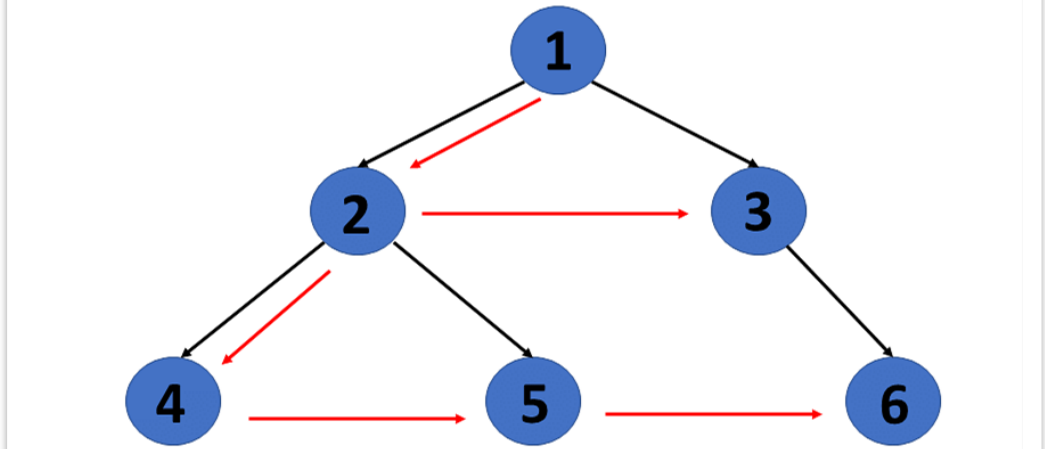
HAPPY HACKING

## 깊이우선탐색(DFS) vs 너비우선탐색(BFS)

우선 DFS에 대해 알아보자!




| DFS | 1 | 2 | 4 | 5 | 3 | 6 |
|---|---|---|---|---|---|---|




| BFS | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|

https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

HAPPY HACKING

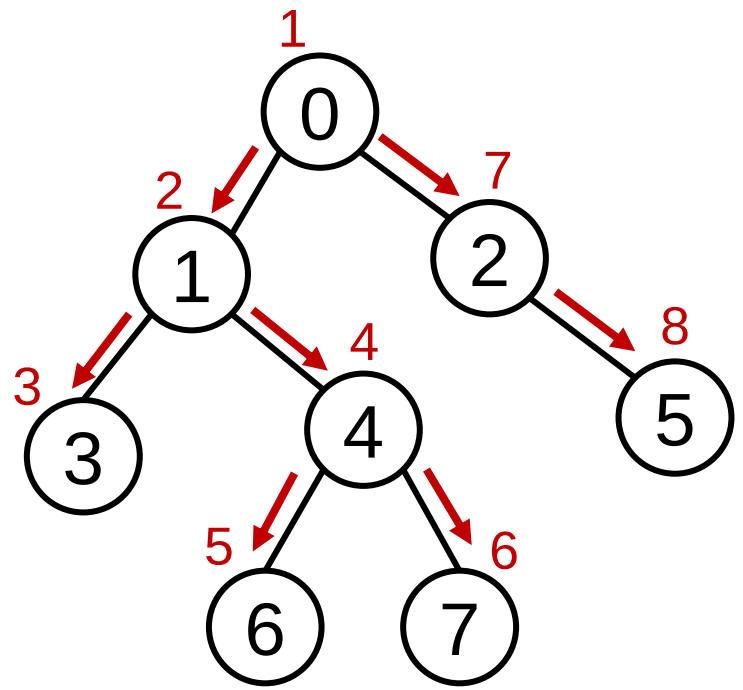
# 2. 깊이우선탐색(DFS)

HAPPY HACKING

## 깊이우선탐색(Depth-First Search, DFS)

시작 정점으로부터 갈 수 있는 하위 정점까지 가장 깊게 탐색하고,

더 이상 갈 곳이 없다면 마지막 갈림길로 돌아와서 다른 정점을 탐색하며 결국 모든 정점을 방문하는 순회 방법

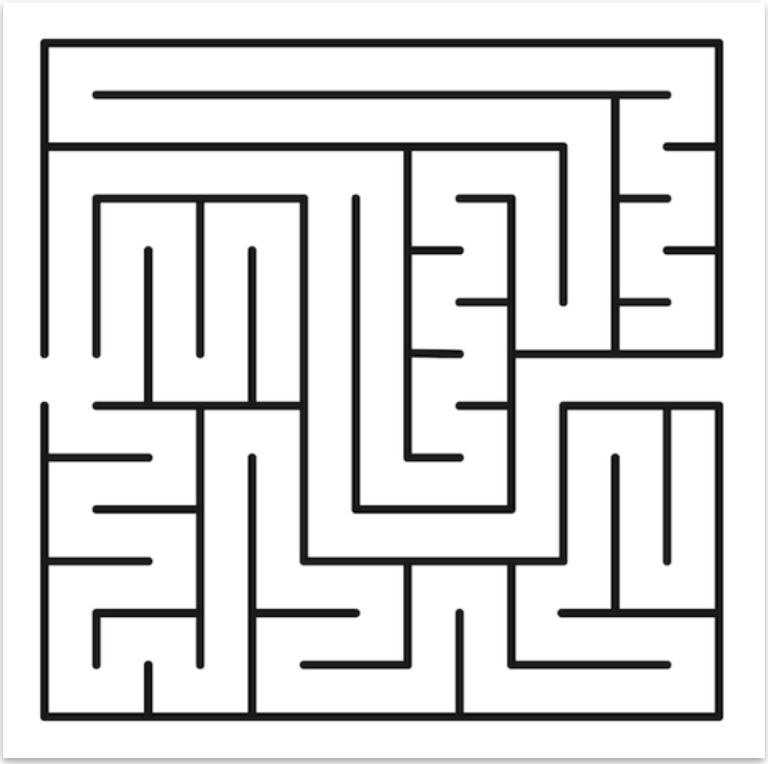


# 2. 깊이우선탐색(DFS)

HAPPY HACKING

깊이우선탐색(DFS)을 미로 탈출로 생각하면 이해하기 쉽다.

어느 한 쪽 길로 가장 깊게 들어갔다가 막히면 다시 돌아와서 다른 길을 탐색한다.

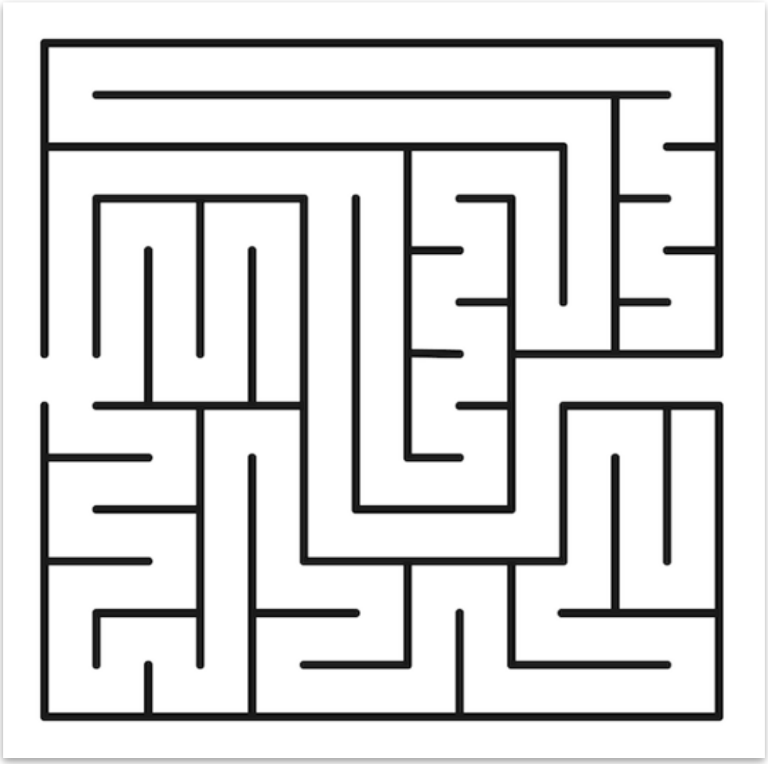


# 2. 깊이우선탐색(DFS)

HAPPY HACKING

깊이우선탐색(DFS)을 미로 탈출로 생각하면 이해하기 쉽다.

어느 한 쪽 길로 가장 깊게 들어갔다가 막히면 다시 돌아와서 다른 길을 탐색한다.

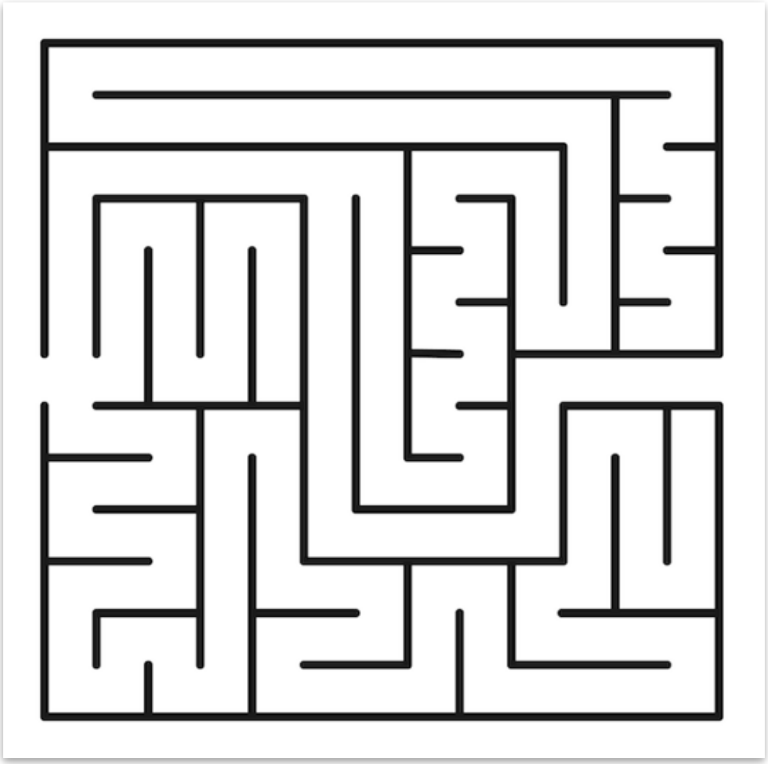


# 2. 깊이우선탐색(DFS)

HAPPY HACKING

깊이우선탐색(DFS)을 미로 탈출로 생각하면 이해하기 쉽다.

어느 한 쪽 길로 가장 깊게 들어갔다가 막히면 다시 돌아와서 다른 길을 탐색한다.

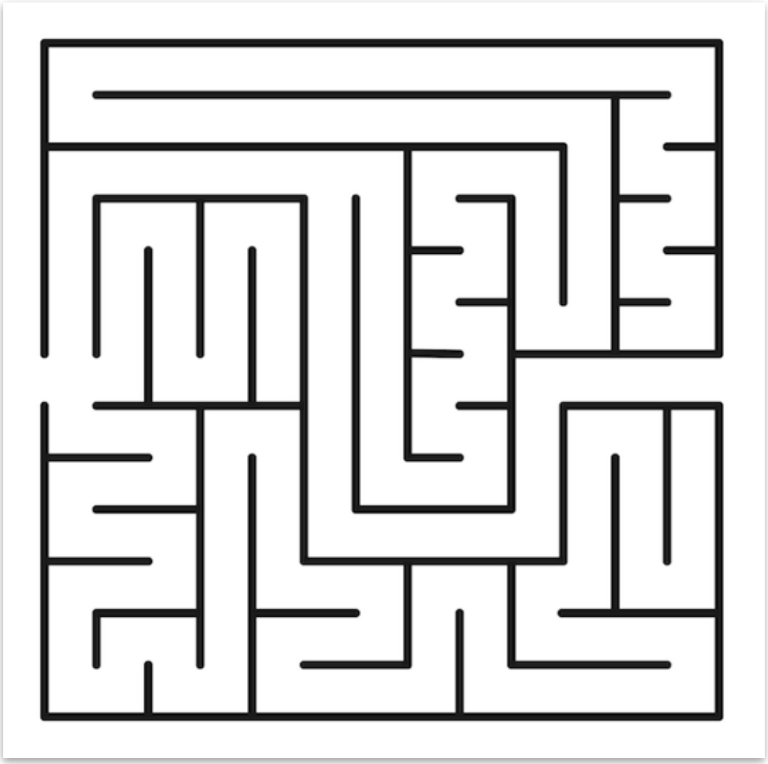


# 2. 깊이우선탐색(DFS)

HAPPY HACKING

깊이우선탐색(DFS)을 미로 탈출로 생각하면 이해하기 쉽다.

어느 한 쪽 길로 가장 깊게 들어갔다가 막히면 다시 돌아와서 다른 길을 탐색한다.



탈출 성공!

HAPPY HACKING

# 깊이우선탐색(DFS)의 특징

모든 정점을 방문할 때 유리하다. 따라서 경우의 수, 순열과 조합 문제에서 많이 사용한다.

너비우선탐색(BFS)에 비해 코드 구현이 간단하다.

단, 모든 정점을 방문할 필요가 없거나 최단 거리를 구하는 경우에는 너비우선탐색(BFS)이 유리하다.

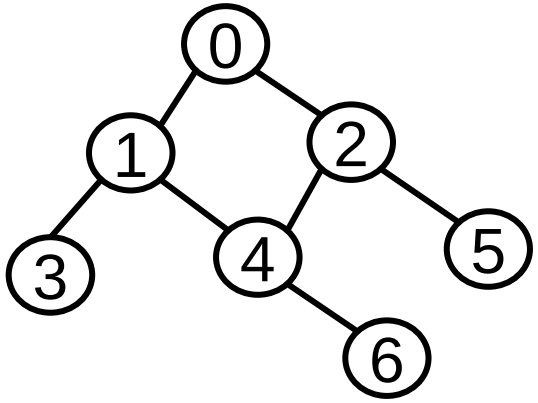
HAPPY HACKING

# 3. DFS의 동작 과정

# 3. DFS의 동작 과정

DFS를 하기 전에, 일단 탐색을 진행할 그래프가 필요하다.

그래프는 인접 행렬 혹은 인접 리스트 방식으로 표현할 수 있다.




```
graph = [
    [0, 1, 1, 0, 0, 0, 0],
    [1, 0, 0, 1, 1, 0, 0],
    [1, 0, 0, 0, 1, 1, 0],
    [0, 1, 0, 0, 0, 0, 0],
    [0, 1, 1, 0, 0, 0, 1],
    [0, 0, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0]
]
```

인접 행렬

or

```
graph = [
    [1, 2],
    [0, 3, 4],
    [0, 4, 5],
    [1],
    [1, 2, 6],
    [2],
    [4]
]
```

인접 리스트

# 3. DFS의 동작 과정

HAPPY HACKING

각 정점을 방문했는지 여부를 판별할 방문 체크 리스트가 필요하다.

사람과 달리 컴퓨터는 각 정점에 방문했는지 여부를 알 수 없다.

따라서 visited 리스트를 따로 선언하여 각 정점을 방문했는지 체크한다.

```
visited = [False] * n  # n은 정점의 개수
```

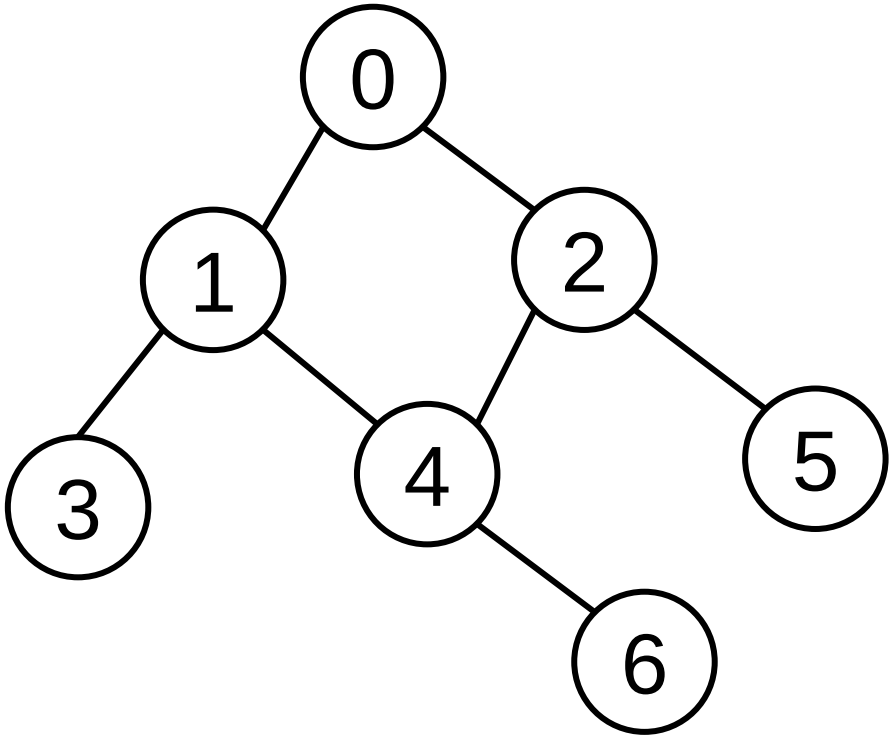
| **정점 i** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **visited[i]** | False | False | False | False | False | False | False |

인덱스는 각 정점의 번호

방문한 정점은 True, 방문하지 않은 정점은 False

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | False | False | False | False | False | False | False |

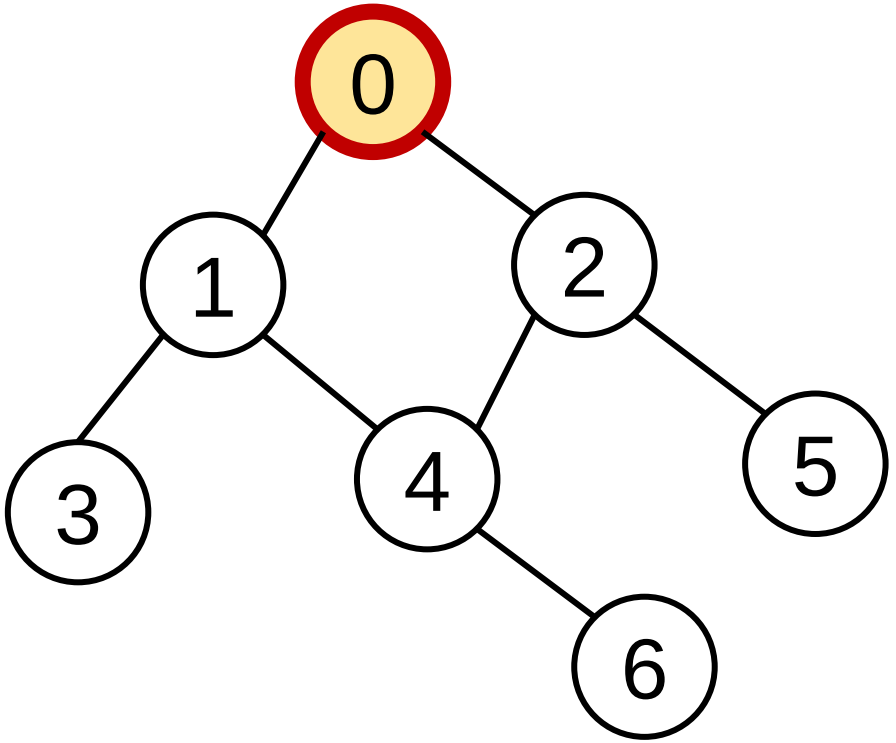



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | False | False | False | False | False | False |

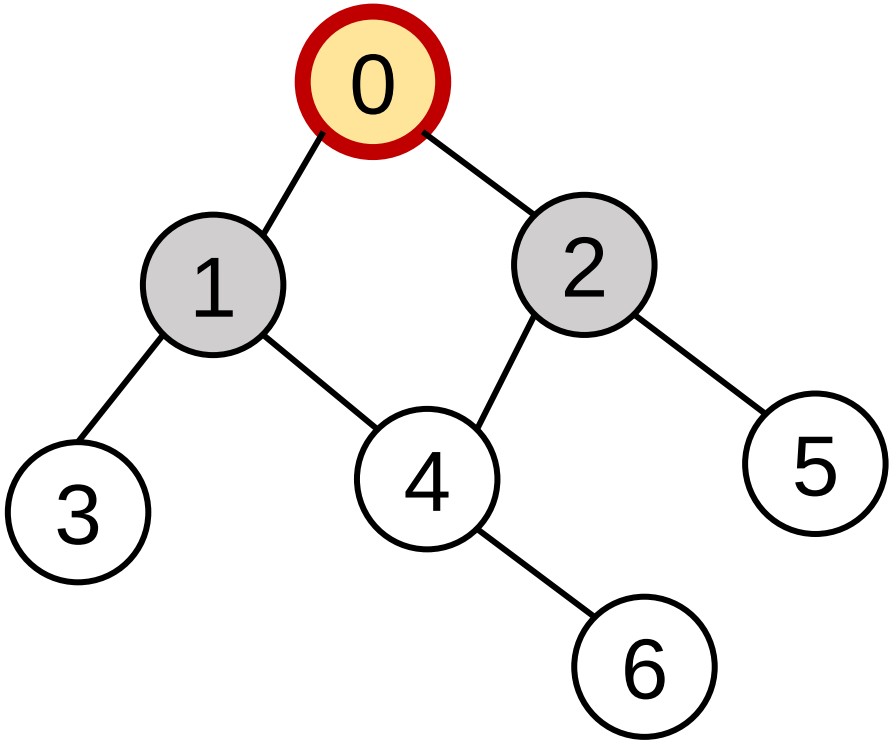



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | False | False | False | False | False | False |

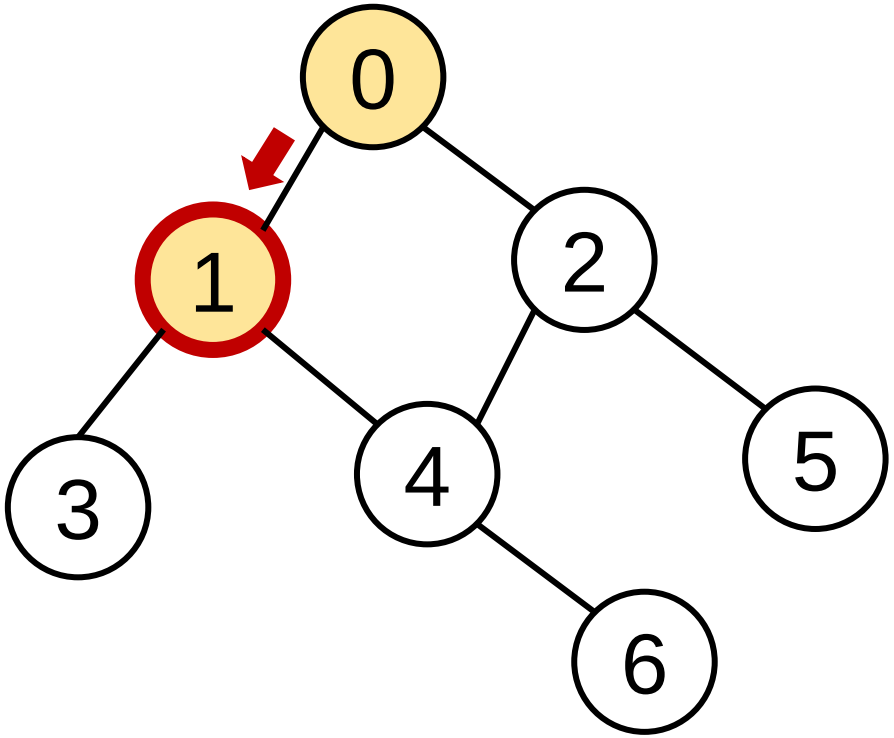



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | False | False | False | False | False | False |

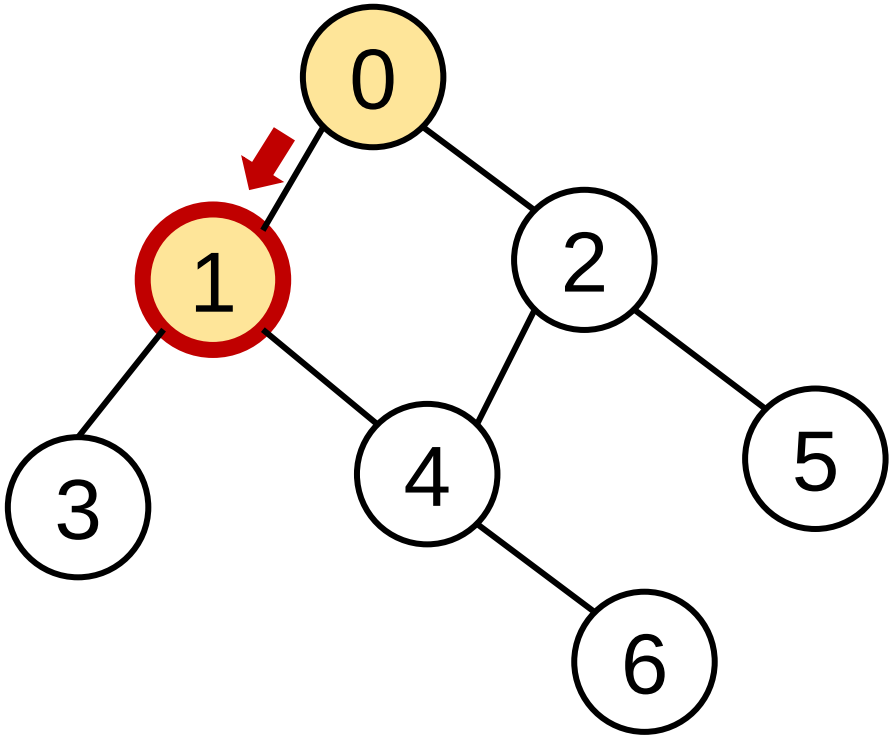



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | False | False | False | False |

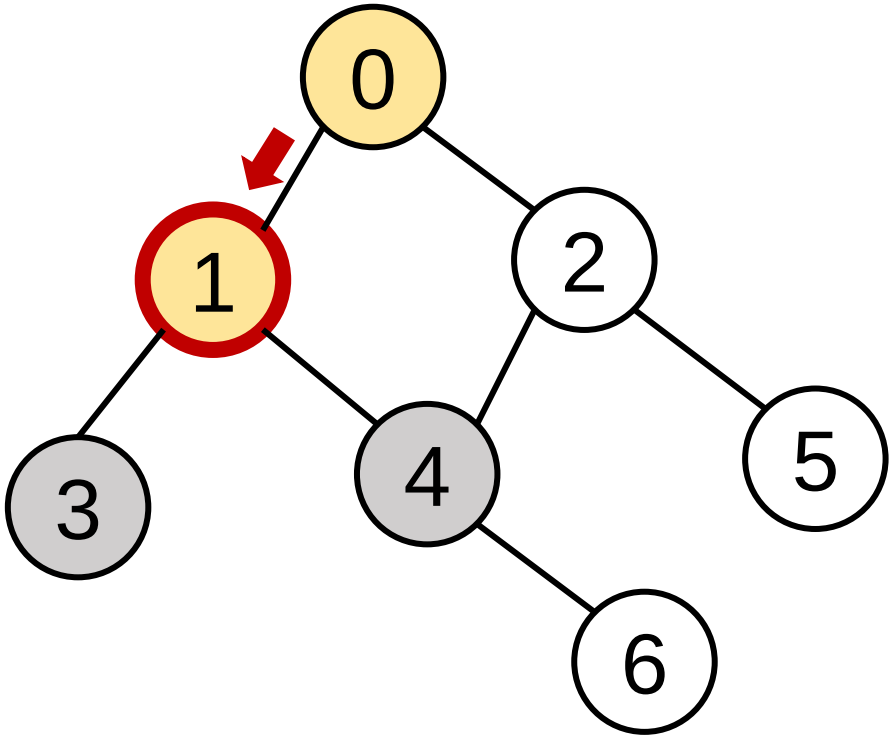



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | False | False | False | False |

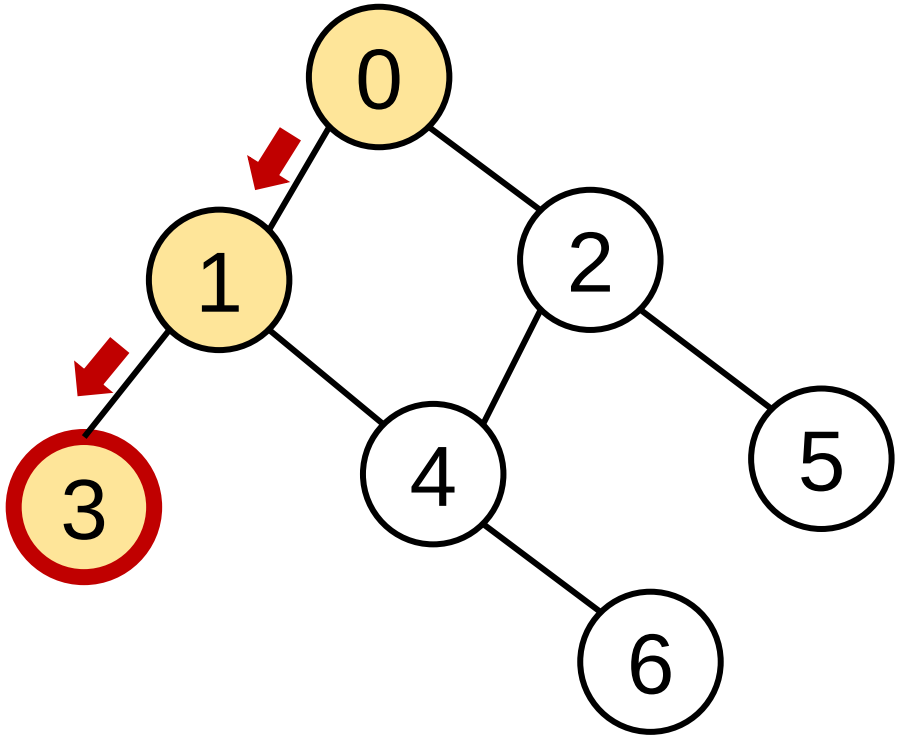



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | False | False | False | False |

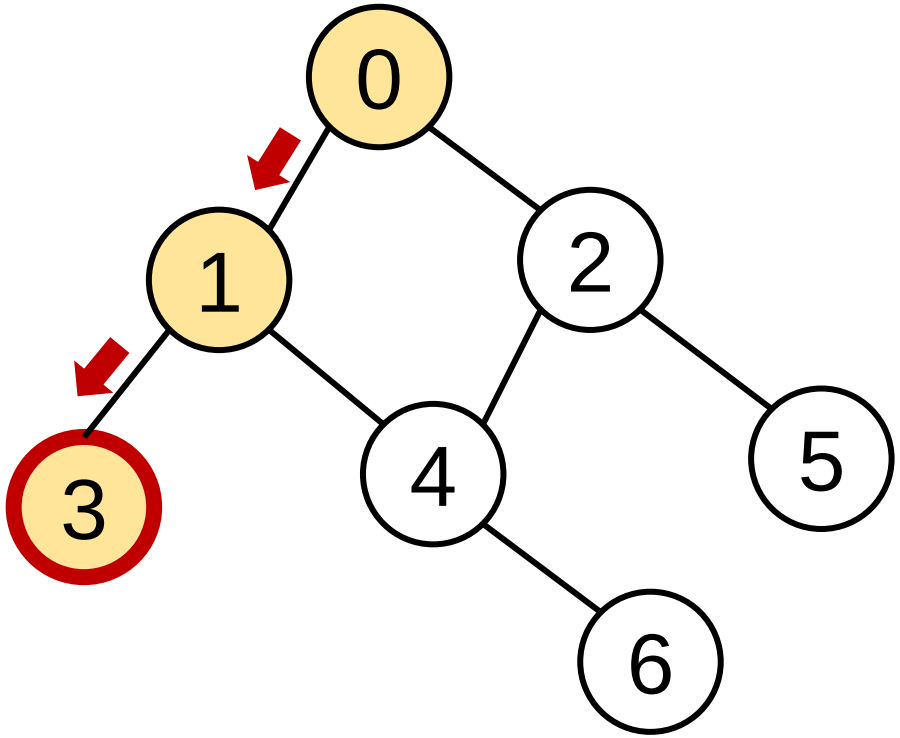



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | False | False | False |



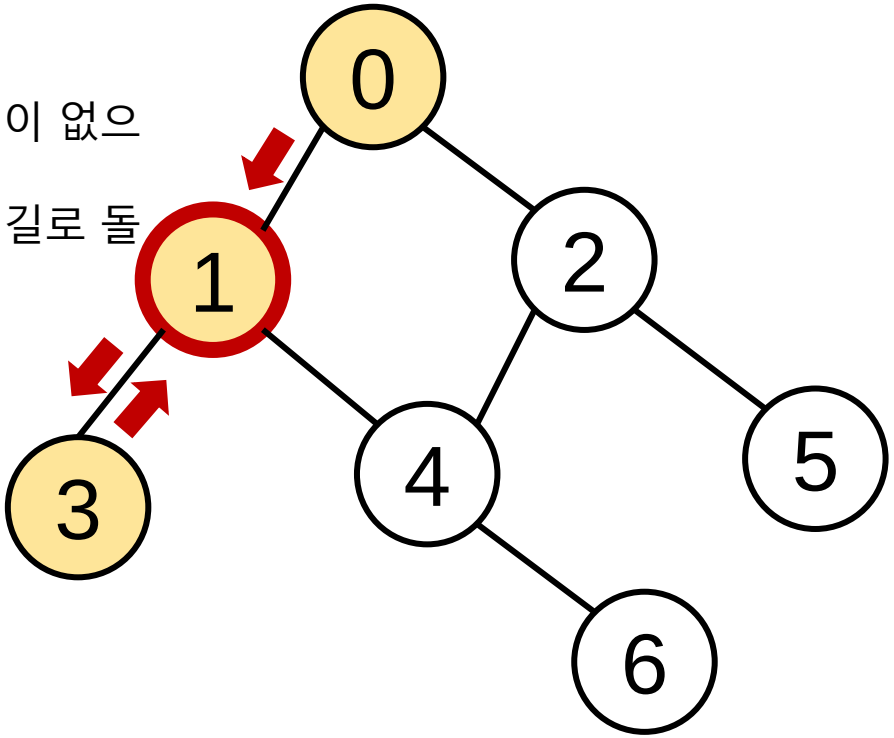

[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | False | False | False |

이동할 정점이 없으므로
마지막 갈림길로 돌아감

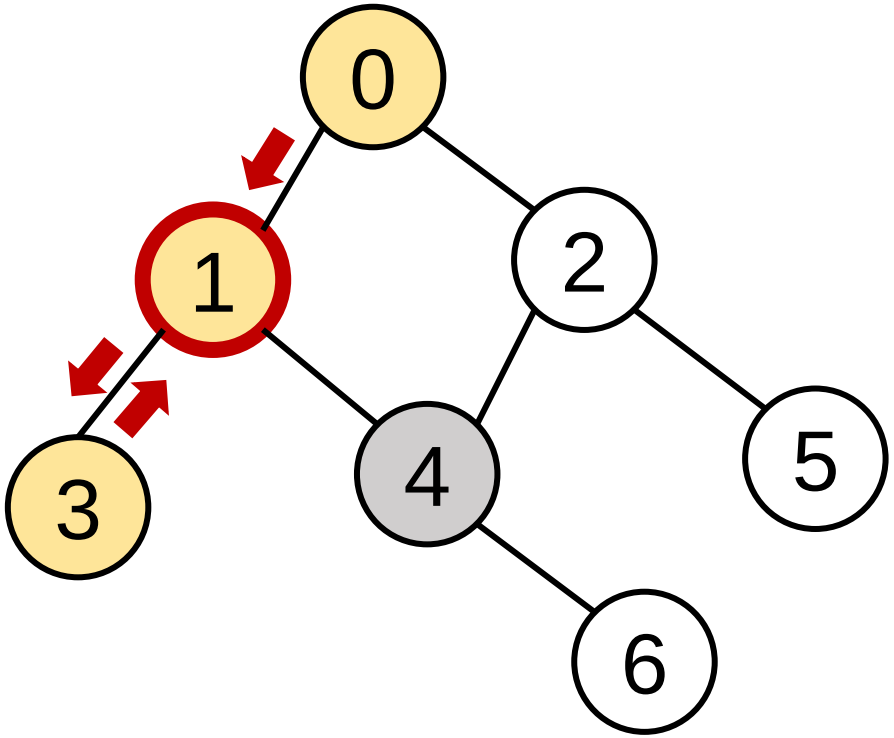



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | False | False | False |

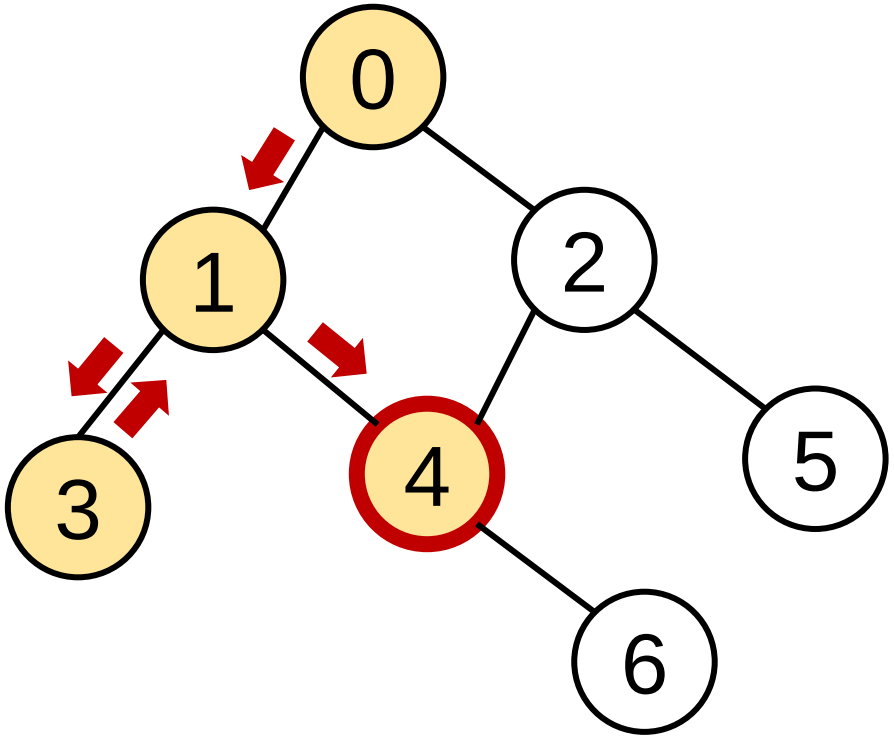



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | False | False | False |

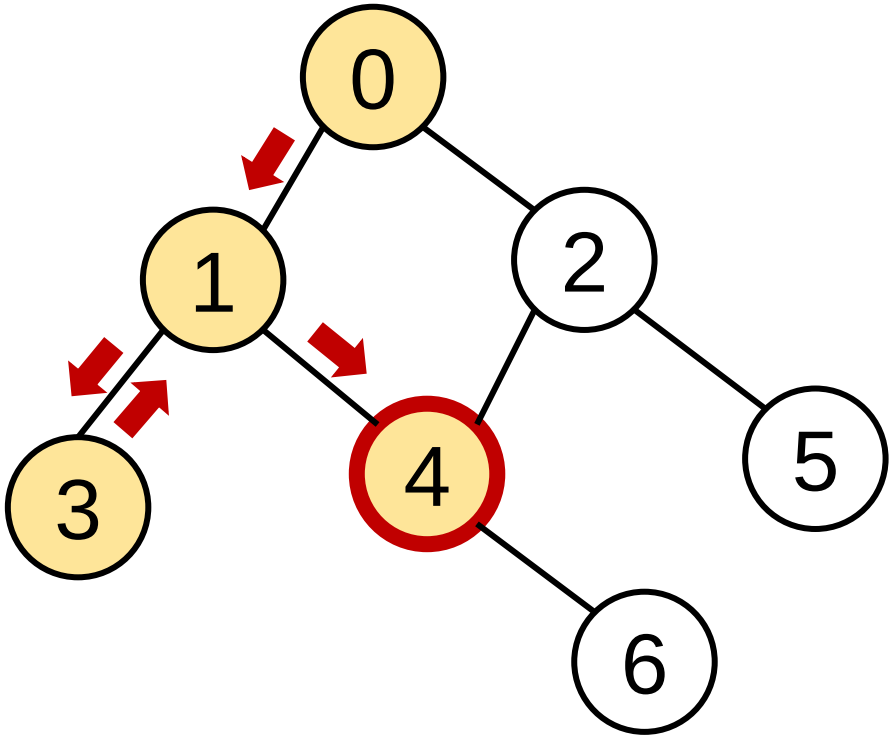



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | True | False | False |

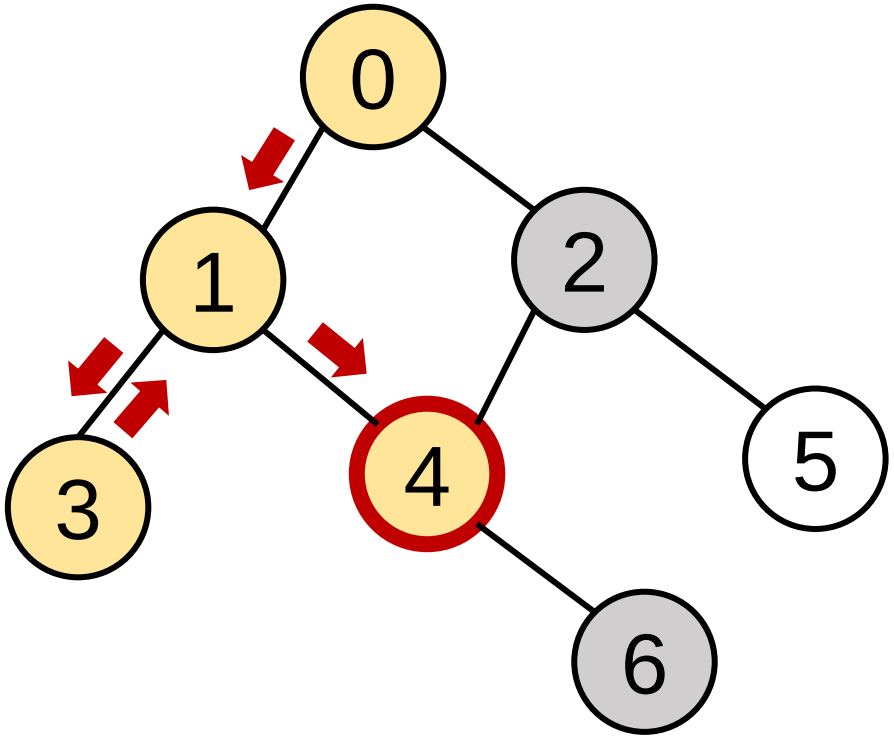



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | True | False | False |

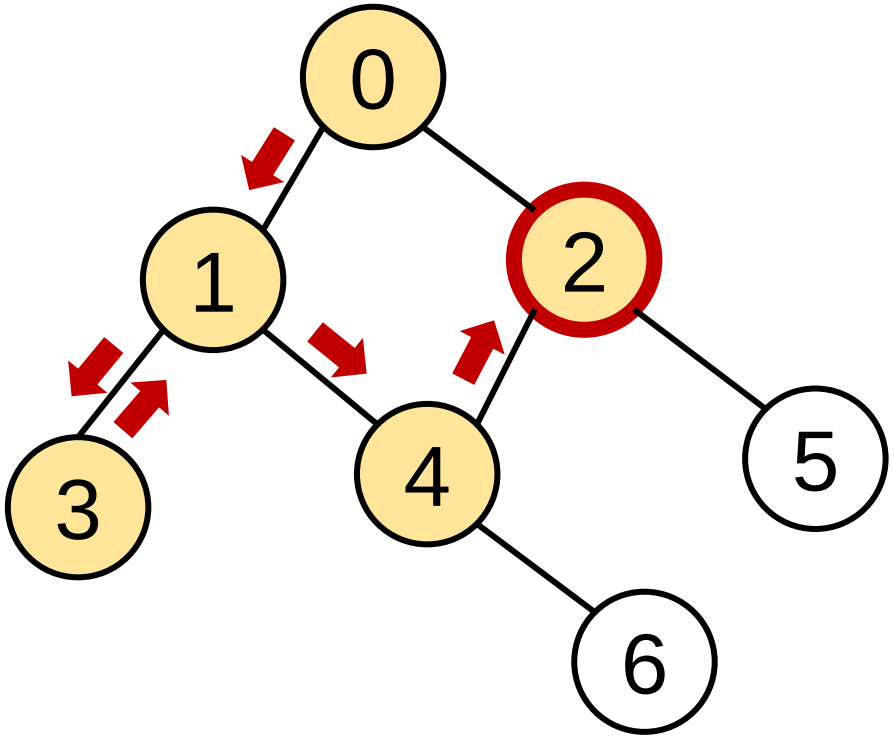



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | True | True | False | False |

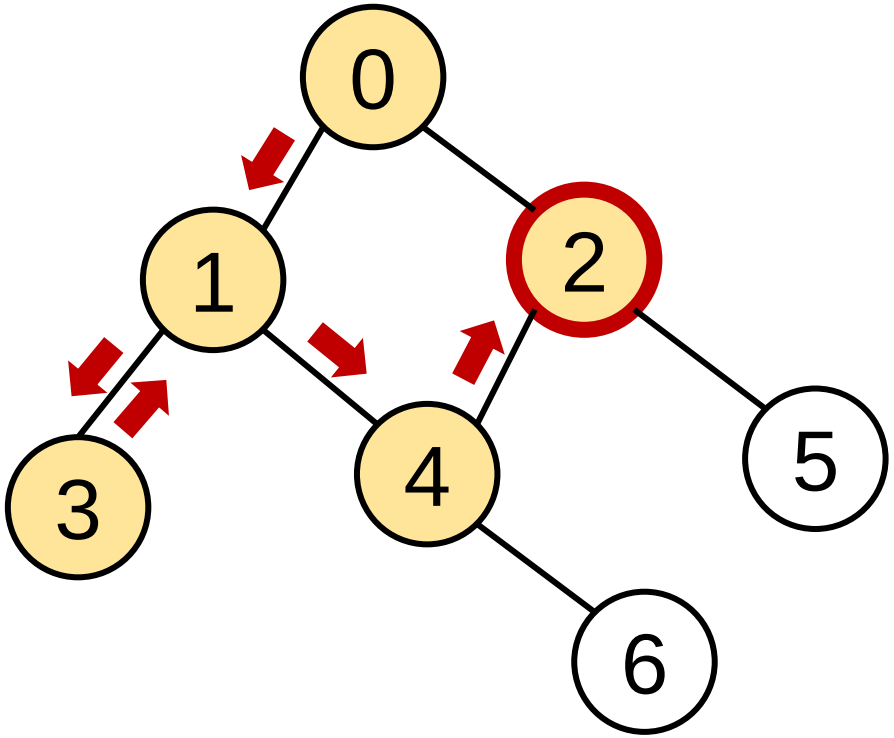



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | False | False |

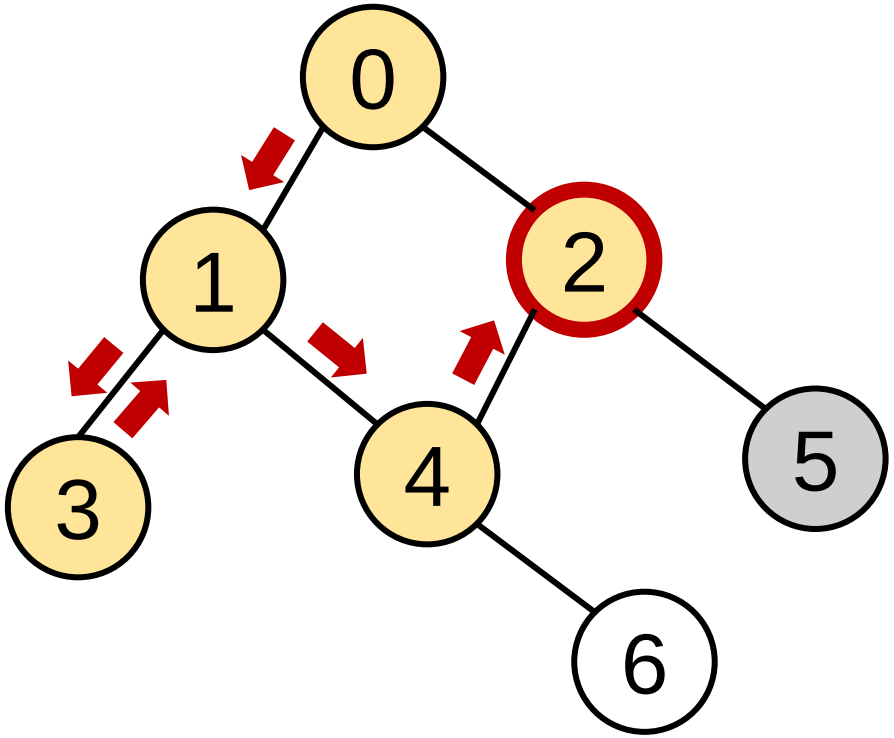



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | False | False |

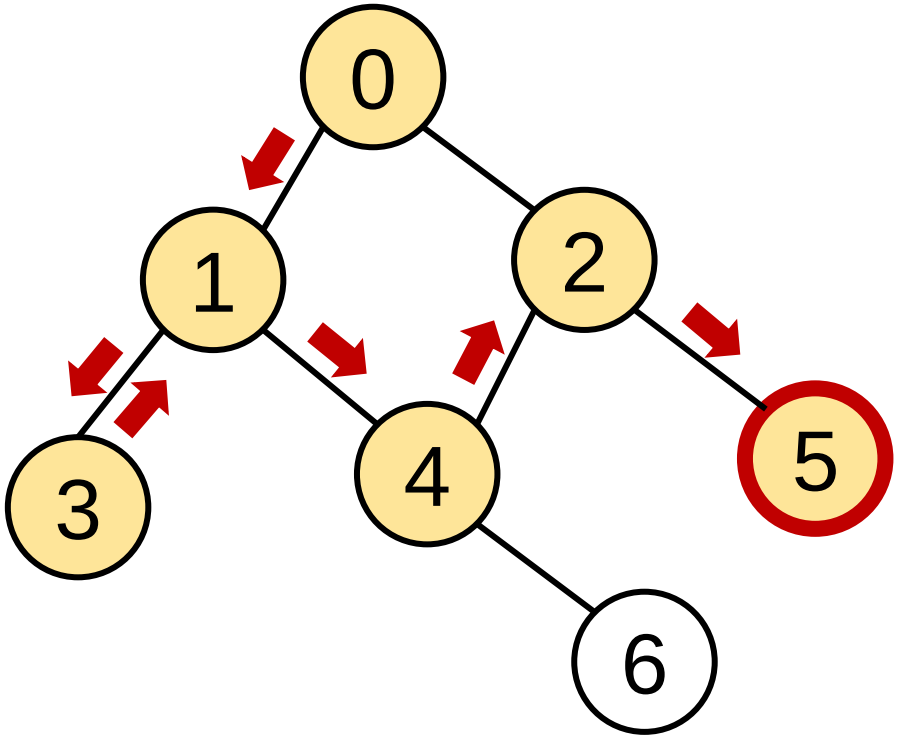



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | False | False |

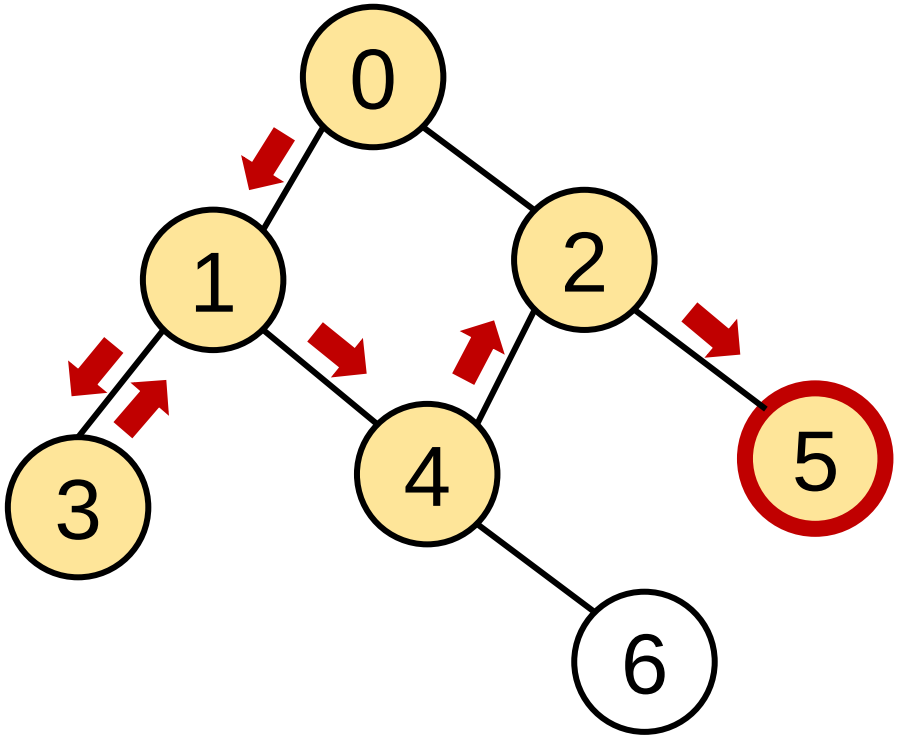



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | False |

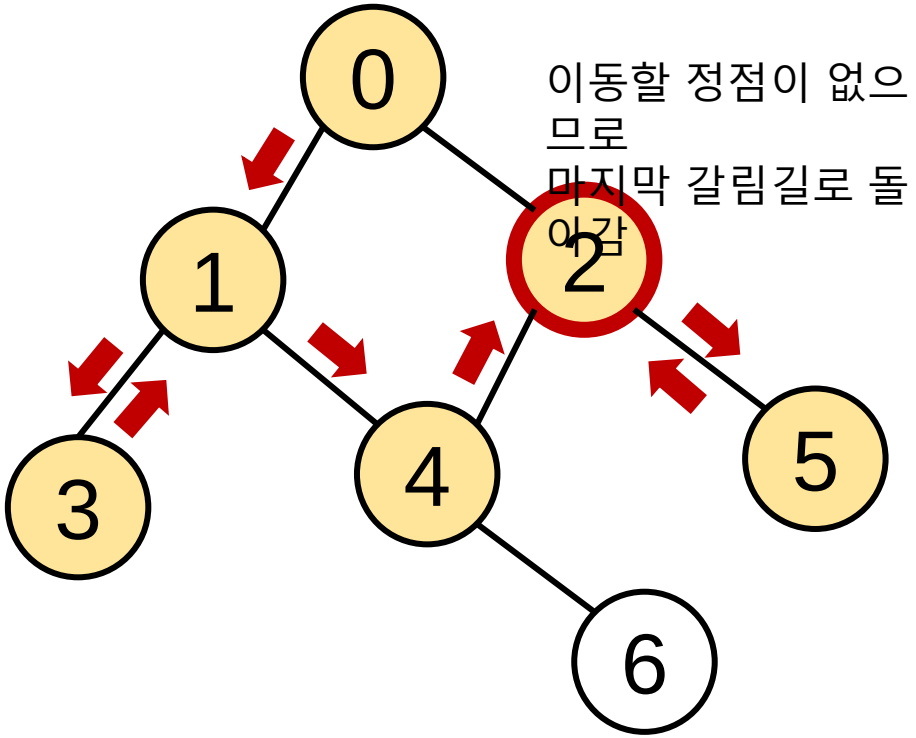



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | False |

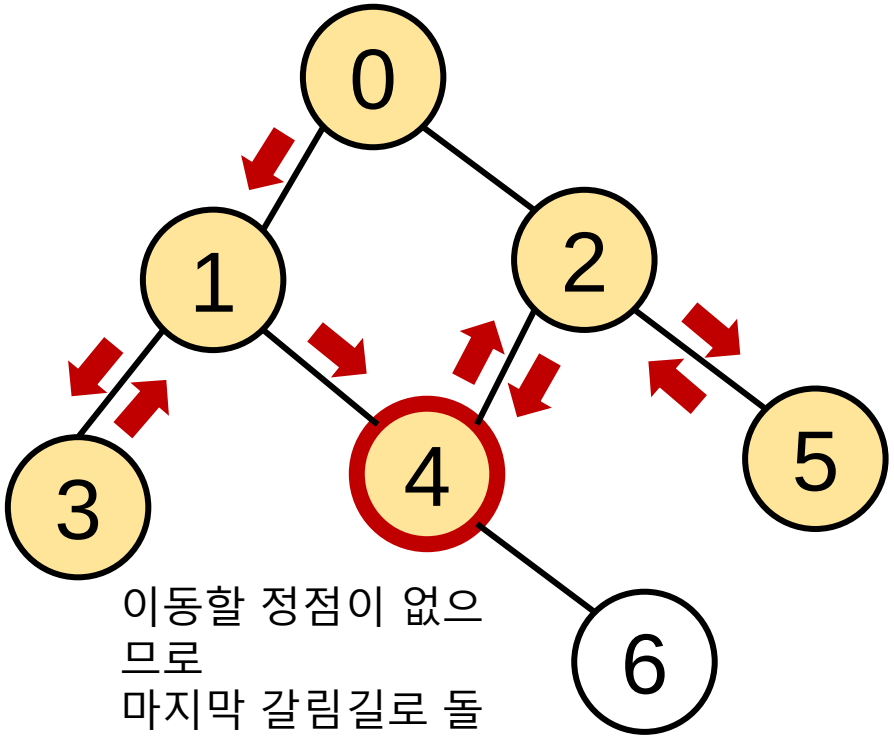



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | False |

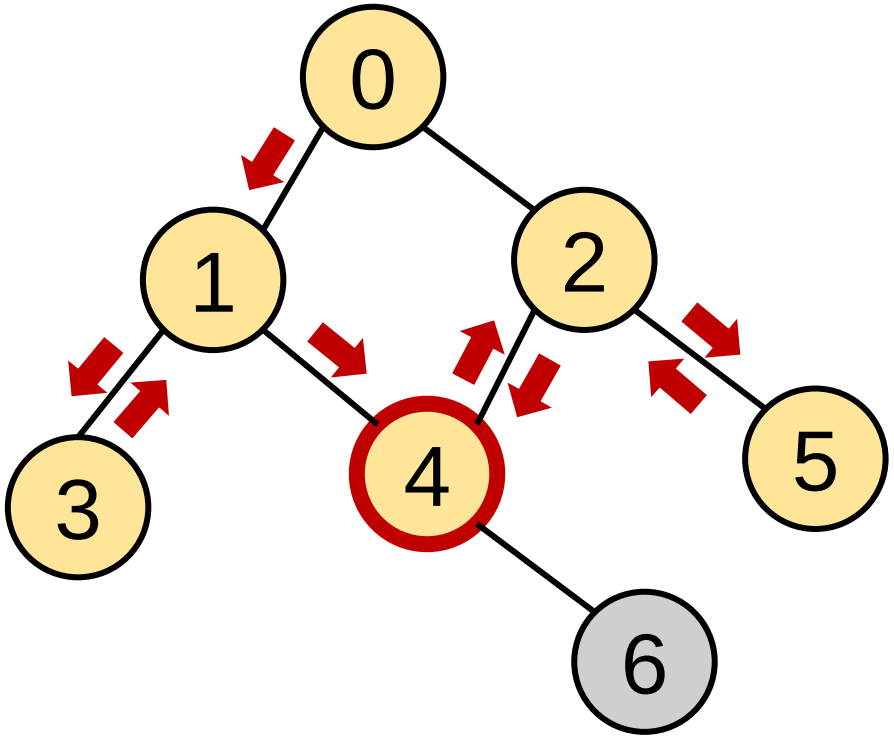



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 2. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | False |

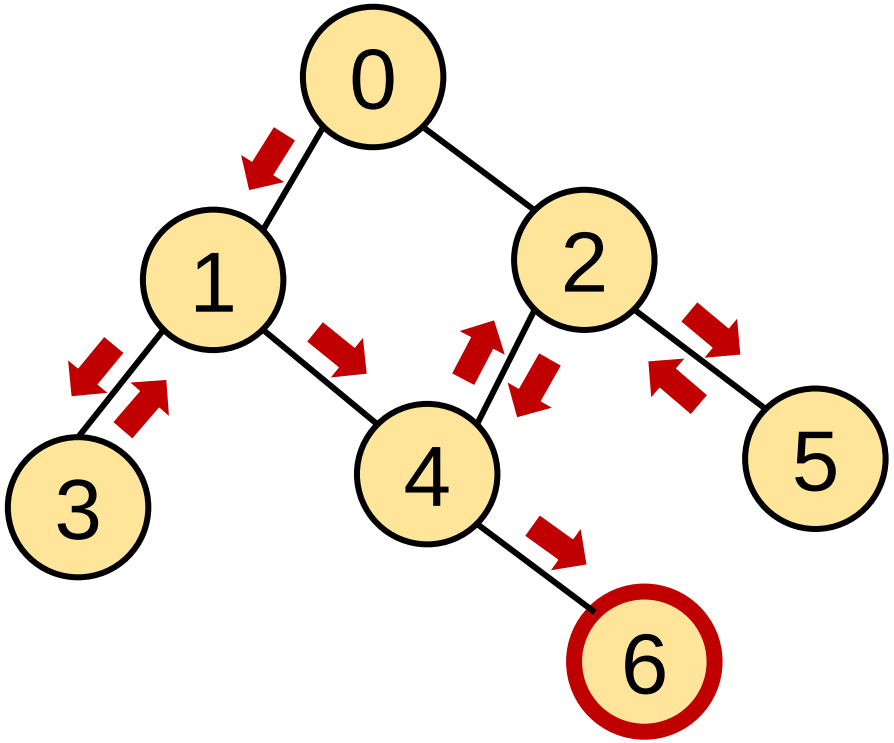



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | False |

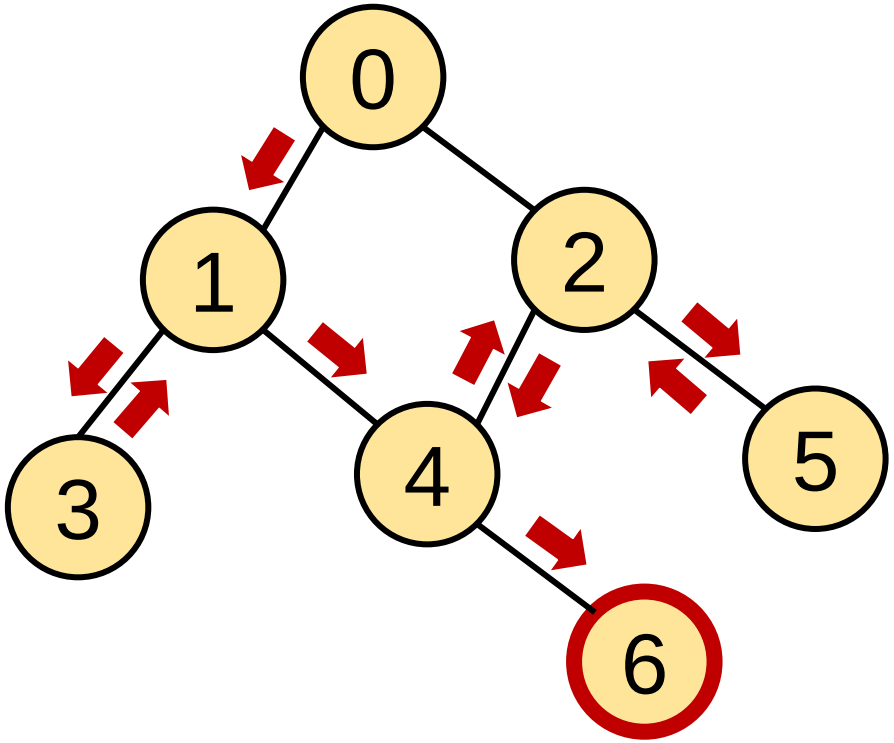



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | True |



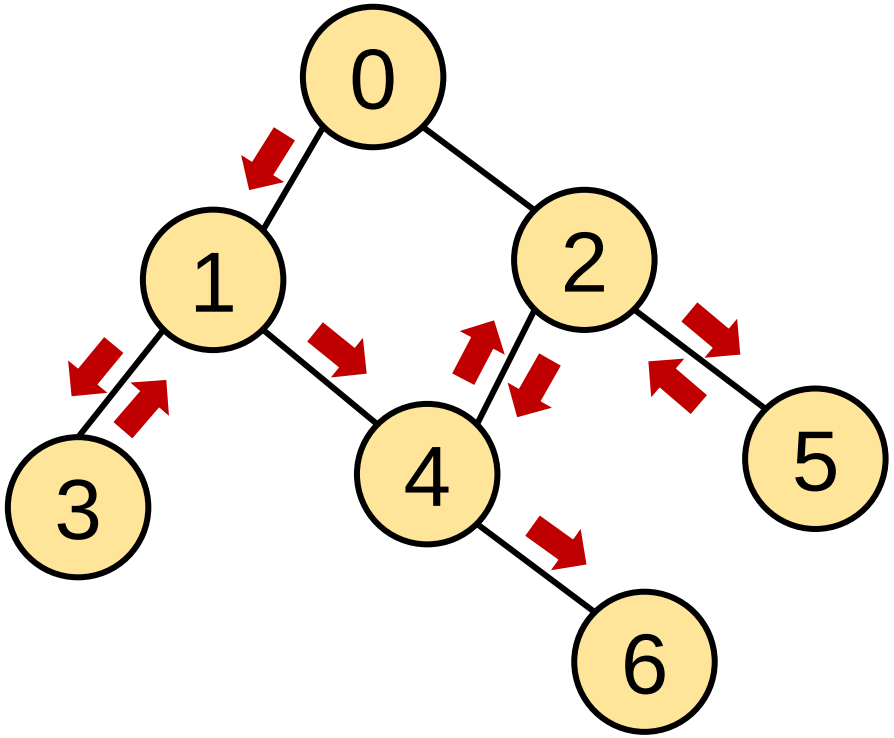

[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | True |

모든 정점을 방문했으므로 탐색 종료

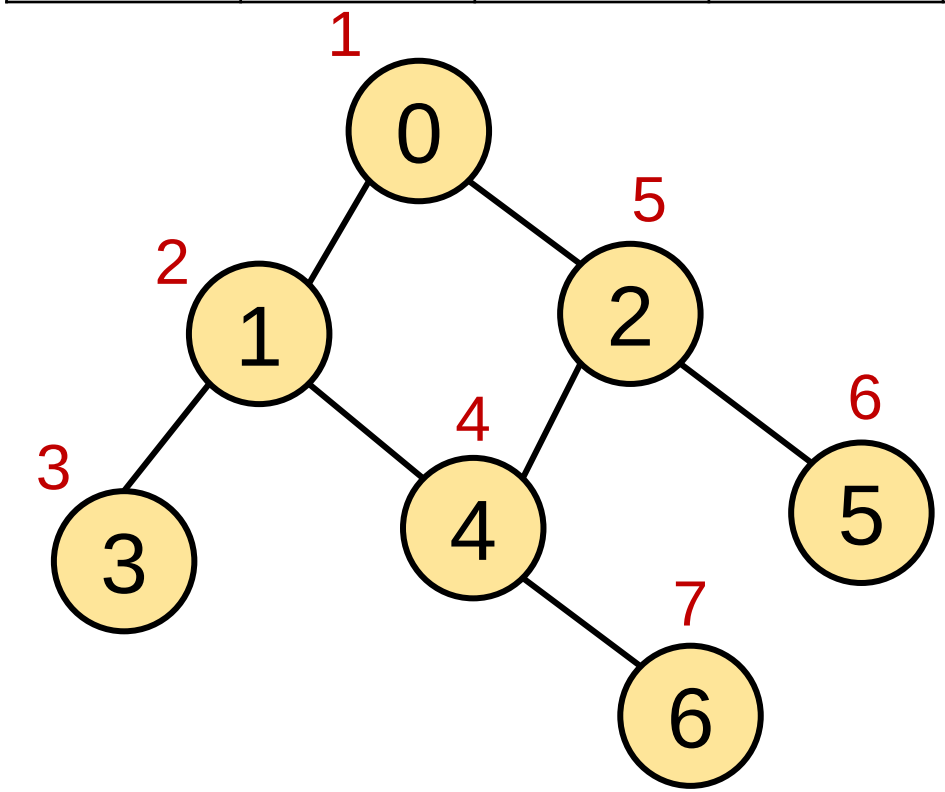



[DFS의 사이클]

1. 현재 정점 방문처리
2. 인접한 모든 정점 확인
3. 방문하지 않은 인접 정점 이동

# 3. DFS의 동작 과정

| 정점 i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | True | True | True | True |




방문 정점 순서

0 – 1 – 3 – 4 – 2 – 5 – 6

HAPPY HACKING

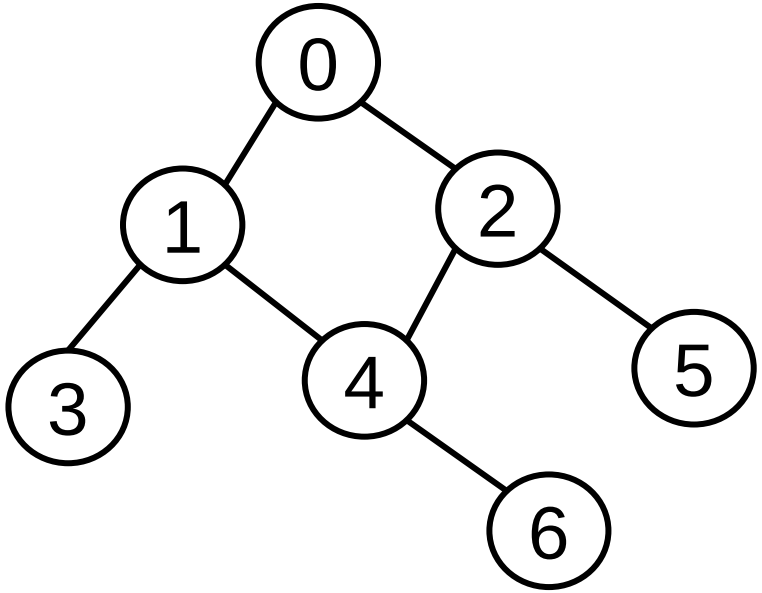
# 4. DFS의 구현 방식

# 4. DFS의 구현 방식

HAPPY HACKING

지금까지 살펴본 DFS를 코드로 구현해보자.

여기에서는 인접 리스트로 표현한 그래프를 기준으로 설명한다.




```
graph = [
    [1, 2],
    [0, 3, 4],
    [0, 4, 5],
    [1],
    [1, 2, 6],
    [2],
    [4]
]
```

# 4. DFS의 구현 방식

HAPPY HACKING

## 반복문을 이용한 DFS

DFS는 직전에 방문한 정점으로 차례로 돌아가야 하므로, 후입선출(LIFO)구조의 스택을 사용한다.

```
graph = [
    [1, 2],
    [0, 3, 4],
    [0, 4, 5],
    [1],
    [1, 2, 6],
    [2],
    [4]
]
```

```
visited = [False] * n  # 방문 처리 리스트 만들기

def dfs(start):
    stack = [start]  # 돌아갈 곳을 기록
    visited[start] = True  # 시작 정점 방문 처리

    while stack:  # 스택이 빌 때까지(돌아갈 곳이 없을때까지) 반복
        cur = stack.pop()  # 현재 방문 정점(후입선출)

        for adj in graph[cur]:  # 인접한 모든 정점에 대해
            if not visited[adj]:  # 아직 방문하지 않았다면
                visited[adj] = True  # 방문 처리
                stack.append(adj)  # 스택에 넣기

dfs(0)  # 0번 정점에서 시작
```

HAPPY HACKING

# 5. DFS 문제 풀이

# 5. DFS 문제 풀이

HAPPY HACKING

대표 예제를 통해 재귀를 이용한 DFS를 이해해보자.

| 문제 번호 | 문제 | 링크 |
| --- | --- | --- |
| BOJ 2606 | 바이러스 | https://www.acmicpc.net/problem/2606 |

문제를 풀지 않고, 단순히 DFS를 이용한 해결 방법만 고민해보자.

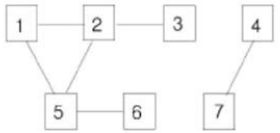
# 5. DFS 문제 풀이

HAPPY HACKING

## 문제

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.




< 그림 1 >

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

### 예제 입력 1 복사

```
7
6
1 2
2 3
1 5
5 2
5 6
4 7
```

### 예제 출력 1 복사

```
4
```

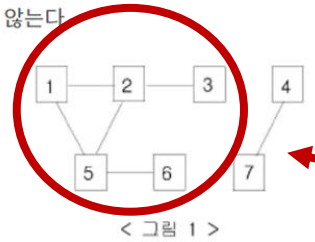
# 5. DFS 문제 풀이

HAPPY HACKING

## 문제

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



< 그림 1 >

[단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

== 1번 컴퓨터에서 방문 가능한 컴퓨터의 수

### 예제 입력 1 복사

```
7
6
1 2
2 3
1 5
5 2
5 6
4 7
```

[단계 1] 입력 값을 받아 인접 리스트 생성

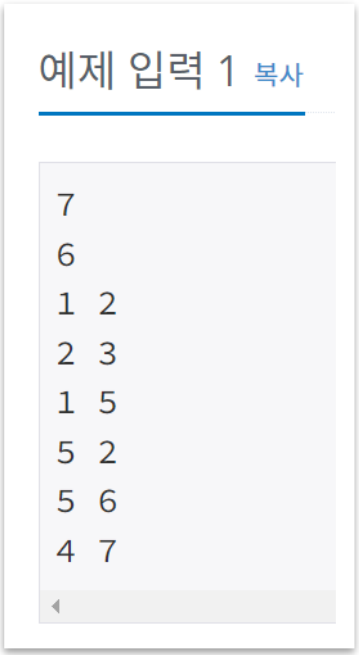
### 예제 출력 1 복사

```
4
```

# 5. DFS 문제 풀이

HAPPY HACKING

## [단계 1] 입력 값을 받아 인접 리스트를 생성

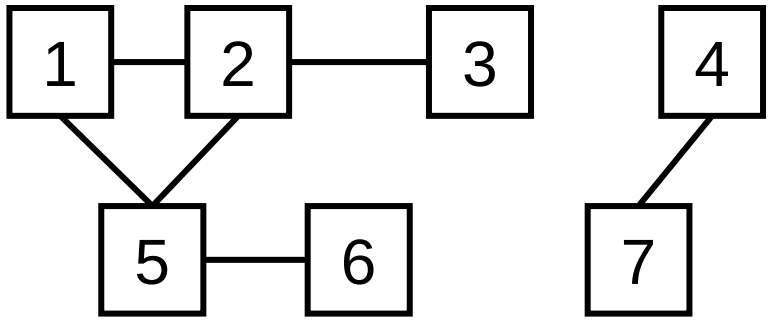



```python
n = int(input())  # 정점 개수(컴퓨터)
m = int(input())  # 간선 개수(네트워크)
graph = [[] for _ in range(n + 1)]
visited = [False] * (n + 1)
total = 0

# 인접 리스트 만들기
for _ in range(m):
    v1, v2 = map(int, input().split())
    graph[v1].append(v2)
    graph[v2].append(v1)
```

HAPPY HACKING

## [단계 1] 입력 값을 받아 인접 리스트를 생성

“예제 입력 1” 그래프의 모습과 인접 리스트의 생성 결과




그래프

```
graph = [
    [],
    [2, 5],
    [1, 3, 5],
    [2], [7],
    [1, 2, 6],
    [5],
    [4]
]
```

인접 리스트

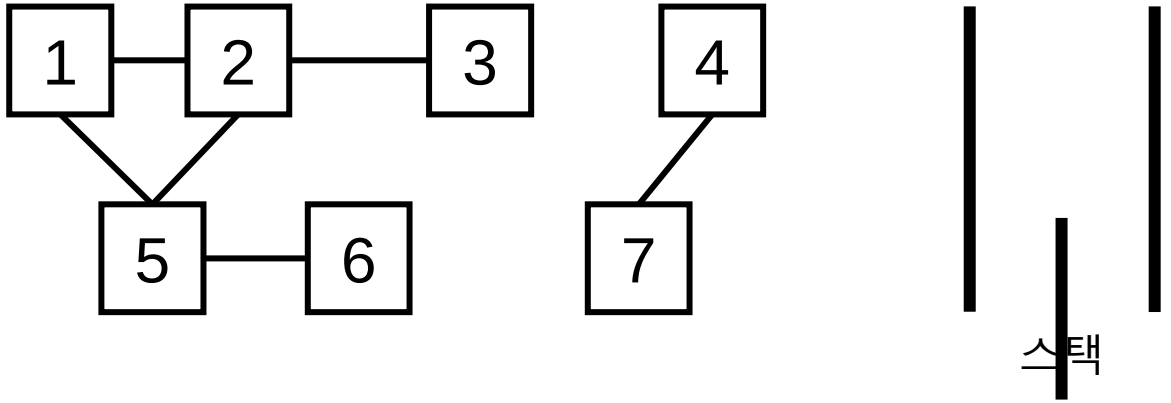
HAPPY HACKING

## [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

단계 2를 세분화하여 DFS의 자세한 동작 과정을 알아보자.

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | False | False | False | False | False | False | False |



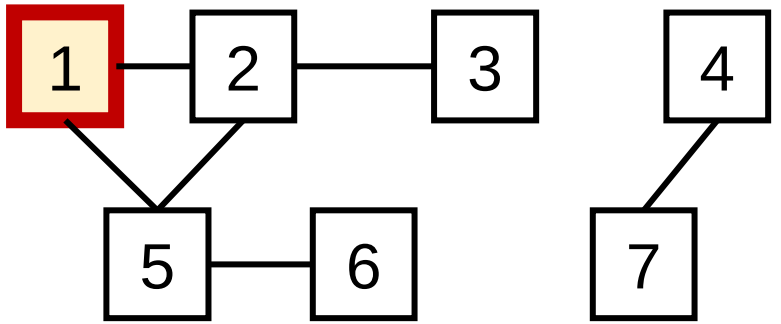
HAPPY HACKING

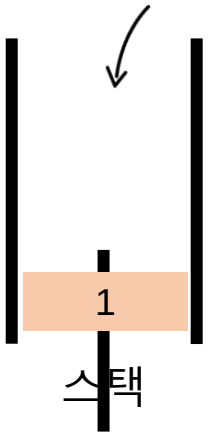
# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-1] dfs(1)을 호출하며 1번 컴퓨터에서 DFS 시작

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | False | False | False | False | False | False | False |



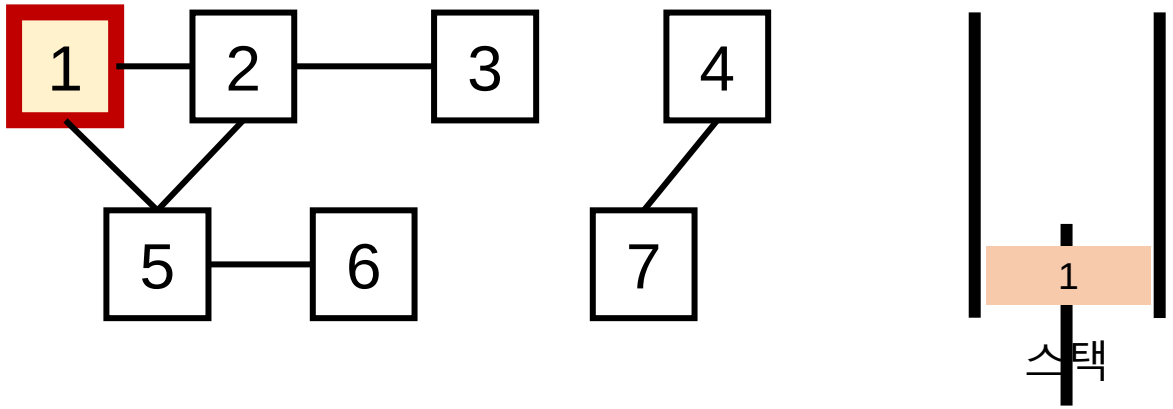
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

## [2-2] 1번 컴퓨터 방문처리

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[ i] | True | False | False | False | False | False | False |



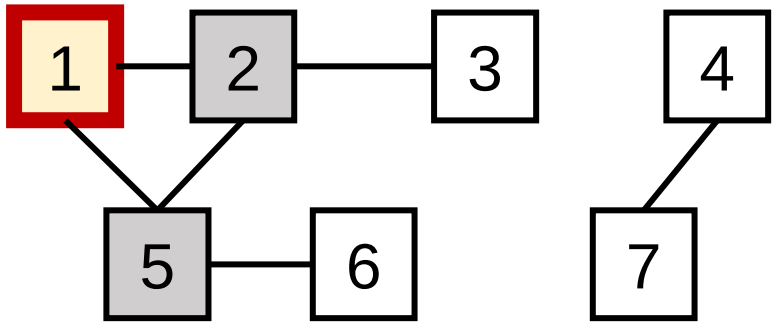
HAPPY HACKING

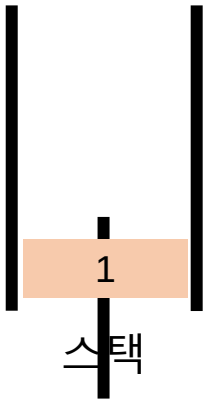
# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-3] 1번 컴퓨터와 인접한 컴퓨터 중 아직 방문하지 않은 곳 조회

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | False | False | False | False | False | False |



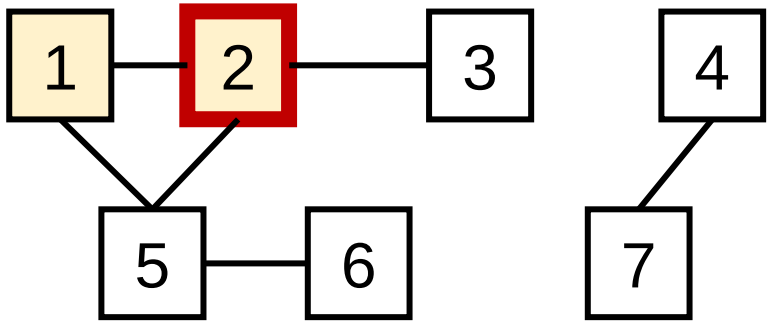
HAPPY HACKING

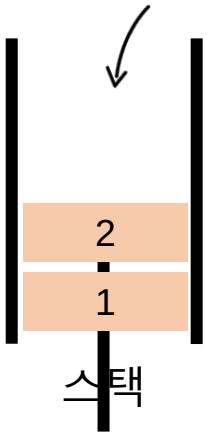
# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-4] 2번 컴퓨터로 이동

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | False | False | False | False | False | False |



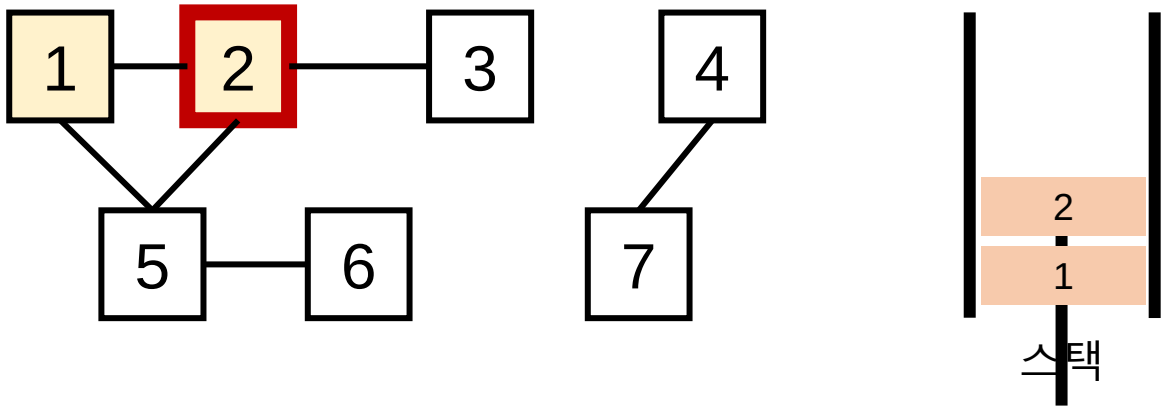
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-5] 2번 컴퓨터 방문처리

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | False | False | False | False |



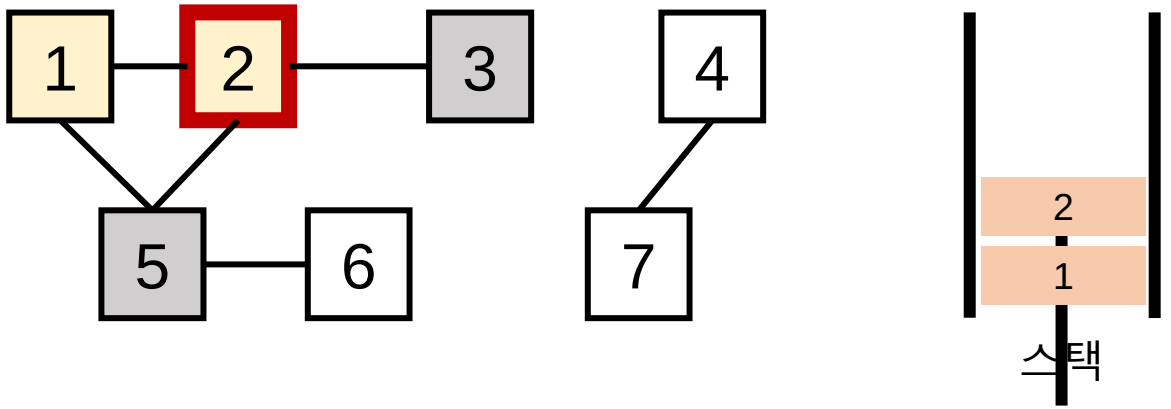
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-6] 2번 컴퓨터와 인접한 컴퓨터 중 아직 방문하지 않은 곳 조회

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | False | False | False | False |



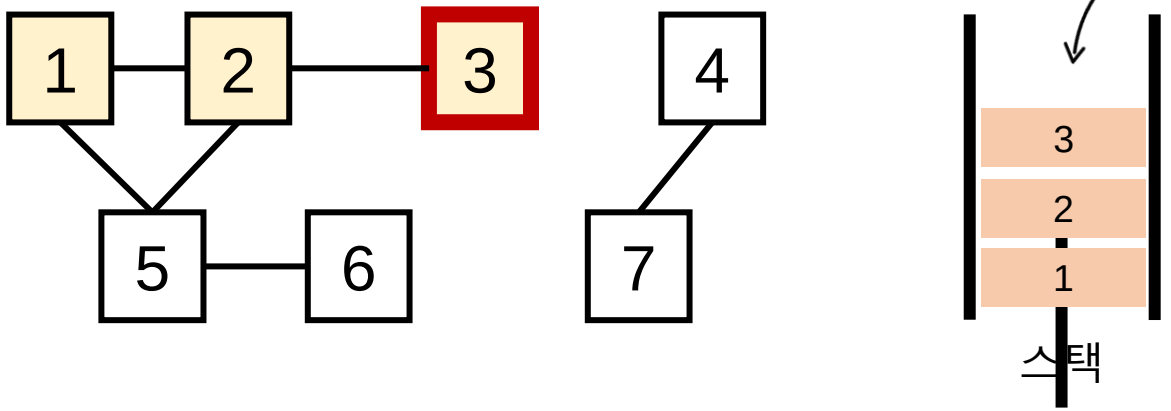
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-7] 3번 컴퓨터로 이동

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | False | False | False | False | False |



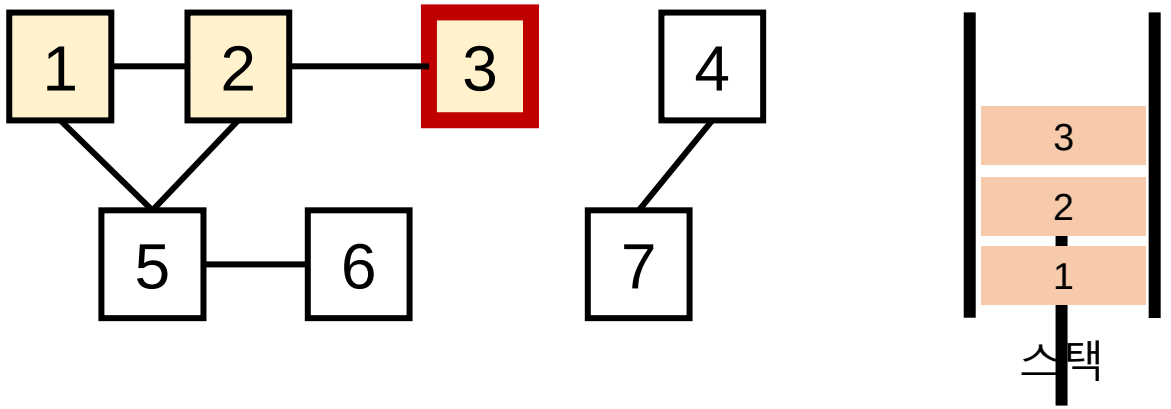
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

## [2-8] 3번 컴퓨터 방문처리

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | False | False | False |



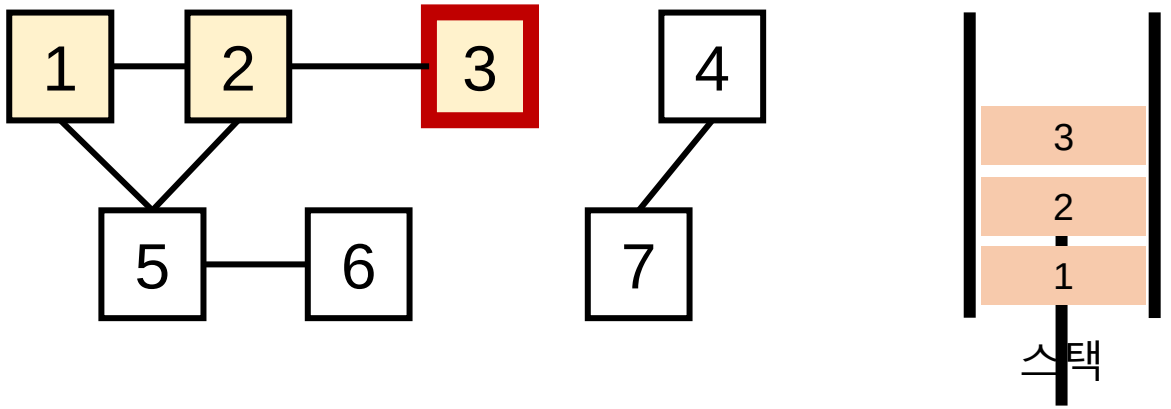
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-9] 3번 컴퓨터와 인접한 컴퓨터 중 아직 방문하지 않은 곳 조회

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | False | False | False |



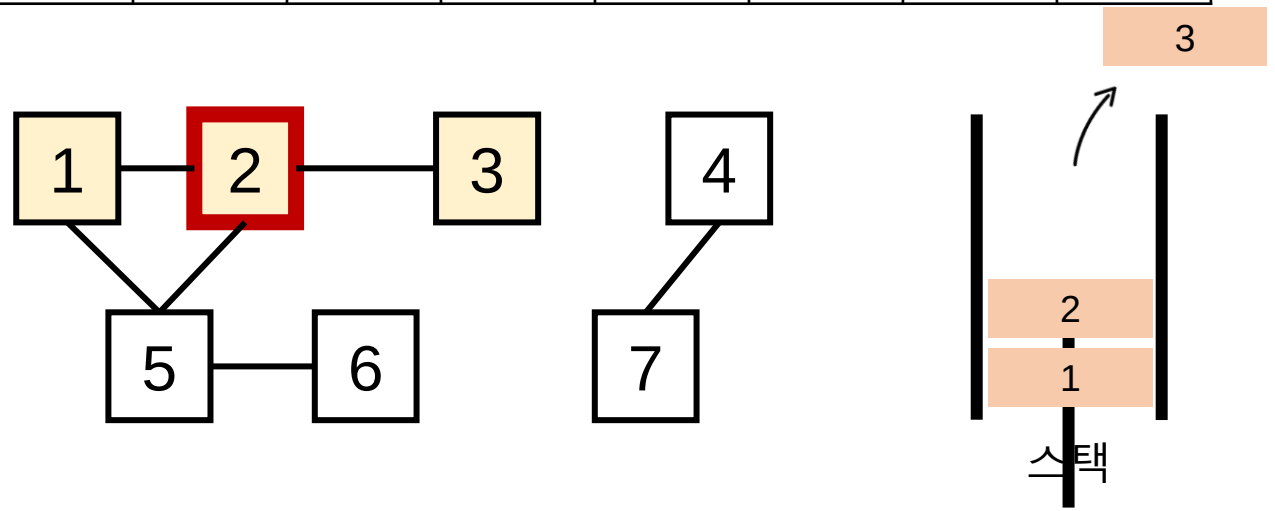
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-10] 3번 컴퓨터에서 갈 곳이 없으므로 이전 정점으로 돌아감

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | False | False | False |



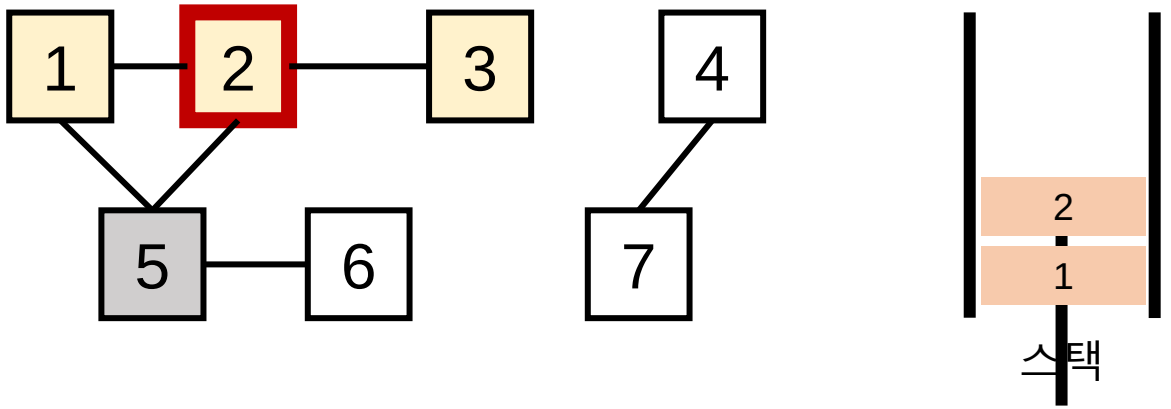
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-11] 방문하지 않은 인접 컴퓨터 조회

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | False | False | False |



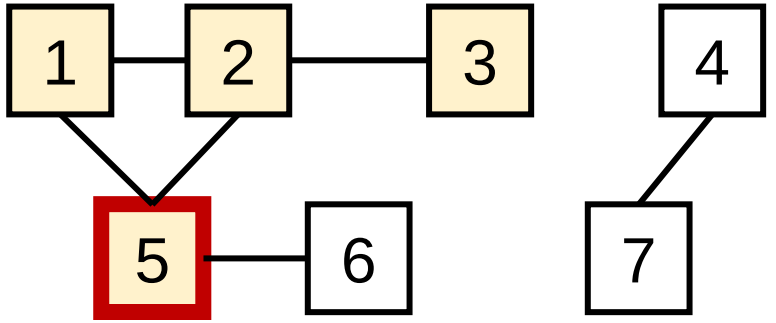
HAPPY HACKING

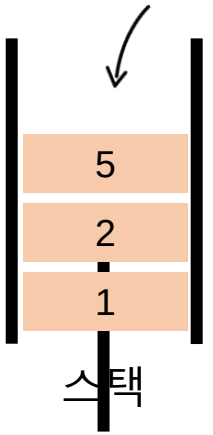
# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-12] 5번 컴퓨터로 이동

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | False | False | False |



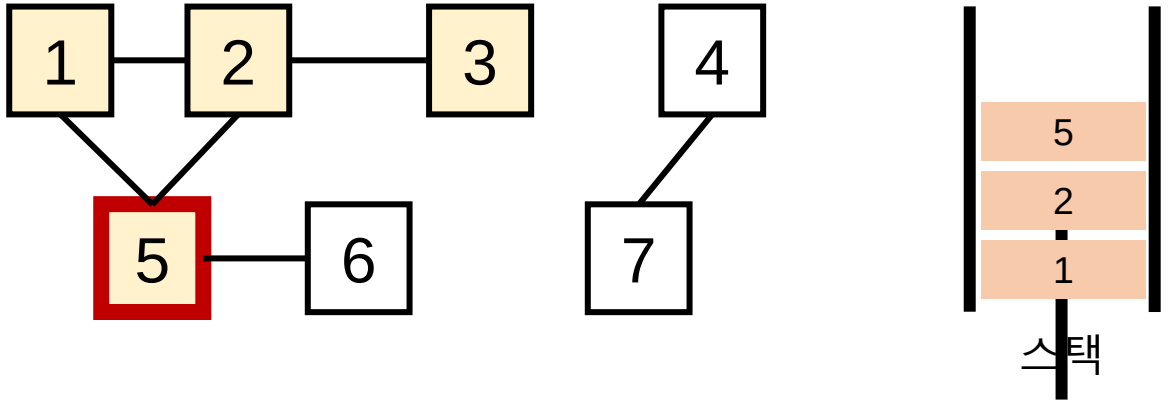
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

## [2-13] 5번 컴퓨터 방문처리

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | False | False |



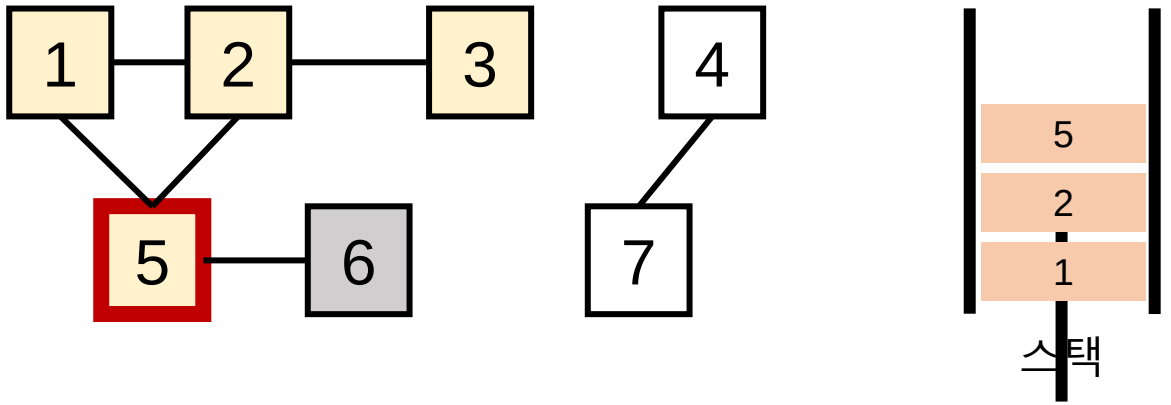
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-14] 5번 컴퓨터와 인접한 컴퓨터 중 아직 방문하지 않은 곳 조회

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | False | False |



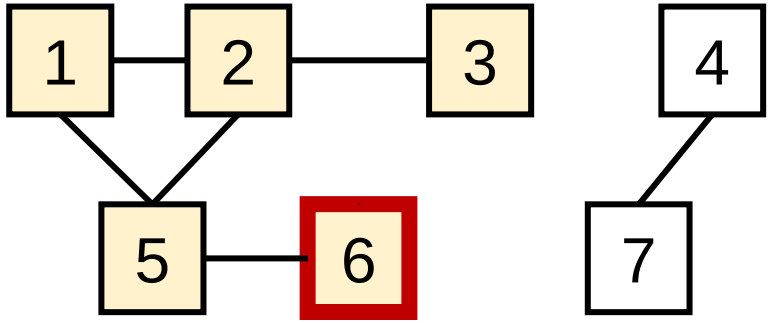
HAPPY HACKING

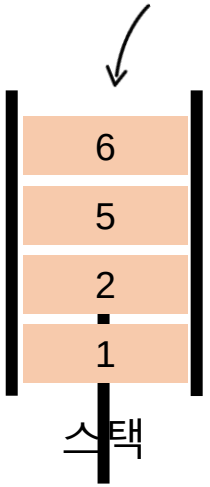
# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-15] 6번 컴퓨터로 이동

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | False | False |



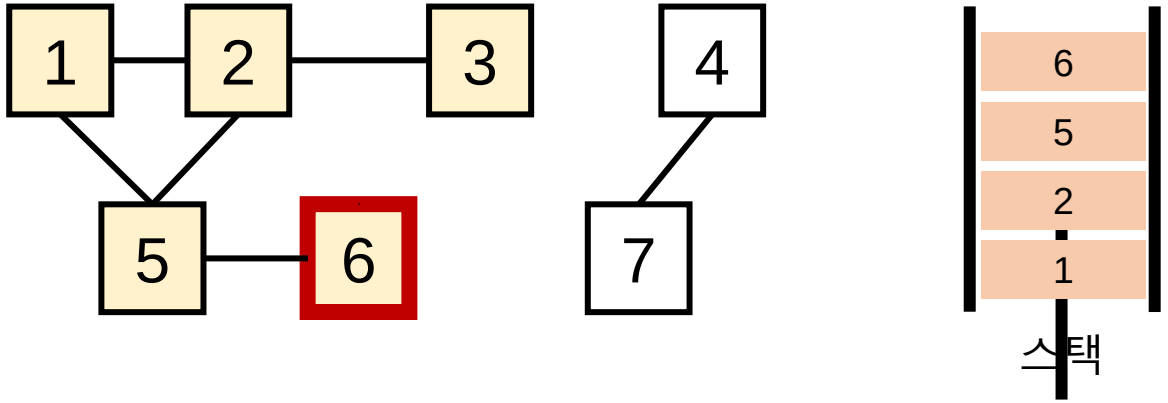
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-16] 6번 컴퓨터 방문처리

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |



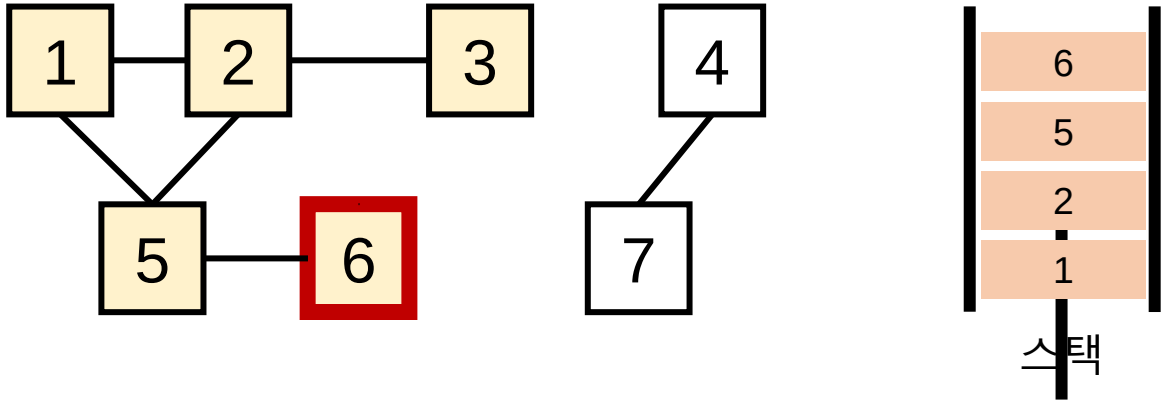
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-17] 6번 컴퓨터와 인접한 컴퓨터 중 아직 방문하지 않은 곳 조회

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |



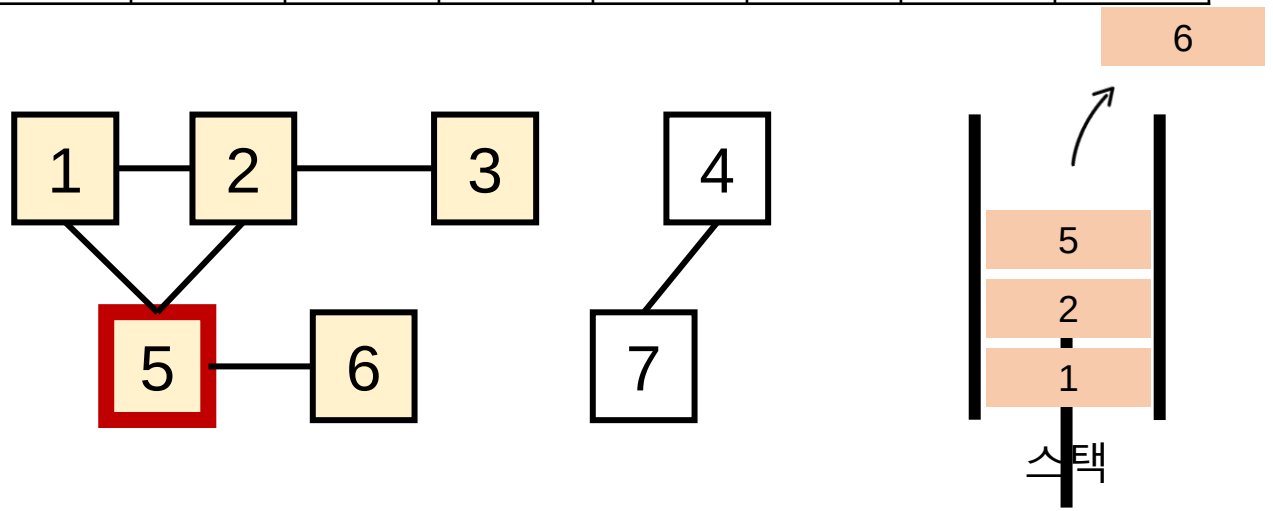
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-18] 6번 컴퓨터에서 갈 곳이 없으므로 이전 정점으로 돌아감

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |



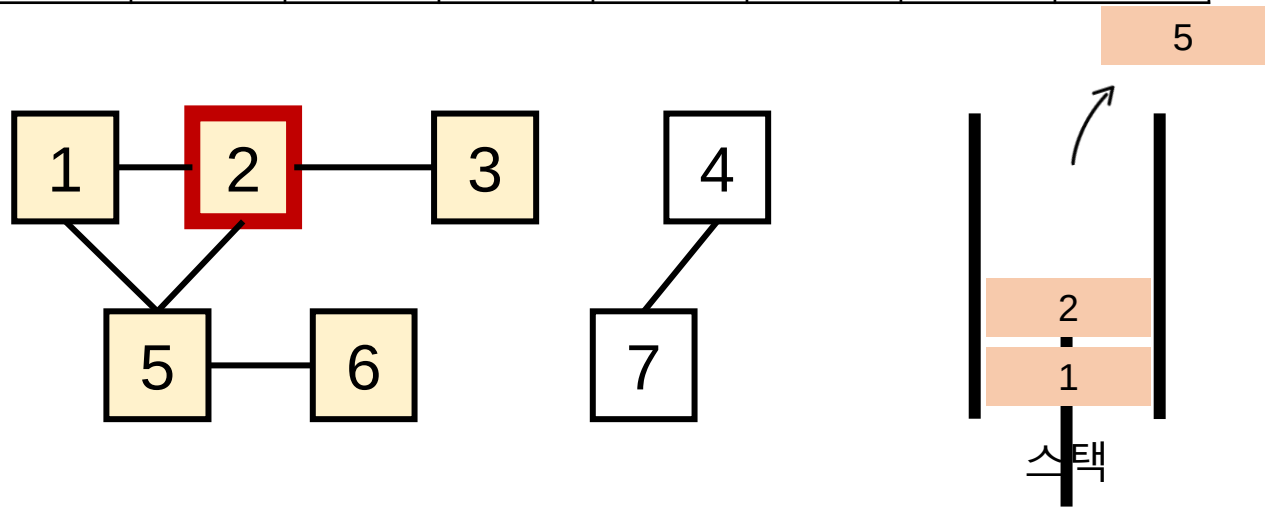
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-19] 5번 컴퓨터에서 갈 곳이 없으므로 이전 정점으로 돌아감

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |



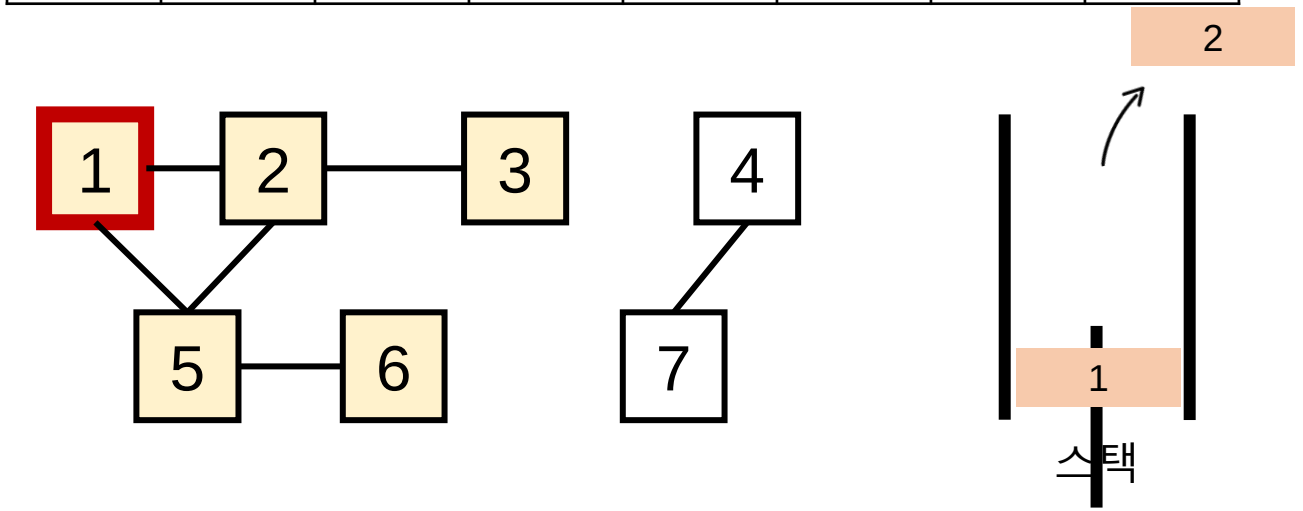
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-20] 2번 컴퓨터에서 갈 곳이 없으므로 이전 정점으로 돌아감

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |



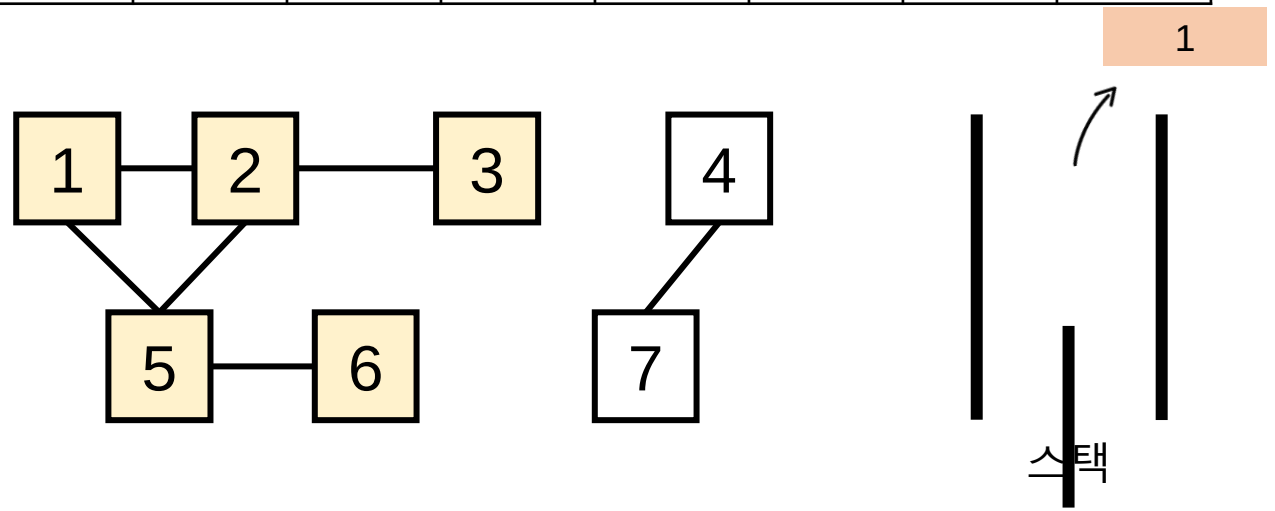
HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-21] 1번 컴퓨터에서 갈 곳이 없으므로 dfs(1)이 종료되고 탐색 종료

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |



HAPPY HACKING

# [단계 2] 1번 컴퓨터를 시작 정점으로 DFS 진행

[2-22] 최종적으로 1번 컴퓨터에 의해 감염되는 컴퓨터는 2, 3, 5, 6 (총 4대)

```
visited = [False] * n

def dfs(start):
    stack = [start]
    visited[start] = True

    while stack:
        cur = stack.pop()

        for adj in graph[cur]:
            if not visited[adj]:
                visited[adj] = True
                stack.append(adj)
dfs(1)  # 1번 정점에서 시작
```

| 정점 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| visited[i] | True | True | True | False | True | True | False |