2021-2 알고리즘

그래프 알고리즘

한남대학교 컴퓨터공학과

그래프 알고리즘(11장~) 구성

- **트리 순회하기**

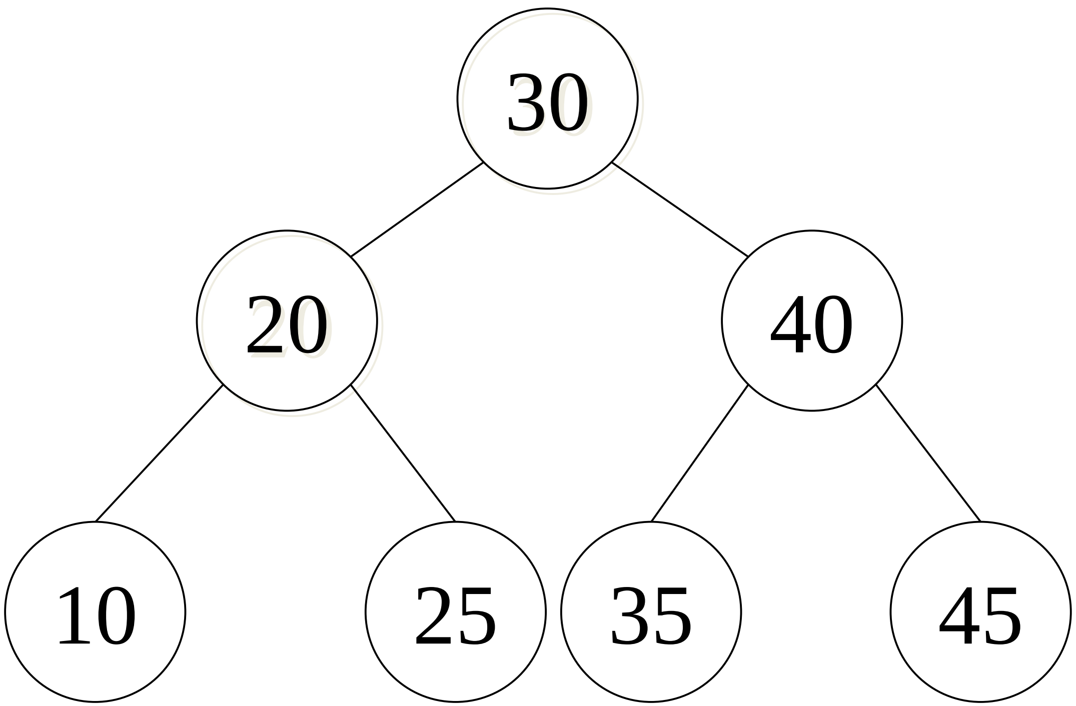
- **그래프**
  - 그래프란?
  - 그래프의 특징에 따른 분류
  - 그래프의 표현
  - nextworkx

- **그래프 탐색 알고리즘**
  - DFS, BFS
  - 최소신장트리
  - 최단 경로 알고리즘
  - 위상 정렬

트리 순회하기

트리 순회하기

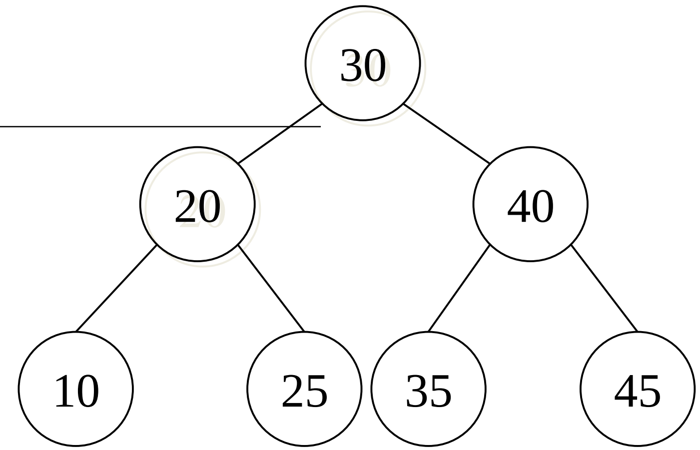
- 입력: 아래와 같이 이진검색트리가 주어진다.
- 문제: BST에 저장된 키값들을 오름차순으로 출력




```
10
20
25
30
35
40
45
```

연습문제

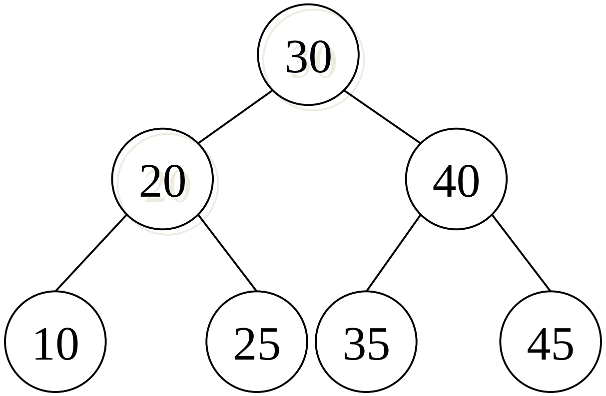
- **이진검색트리가 아래와 같이 주어진다.**




```
class BST:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None



root = BST(30)
root.left, root.right = BST(20), BST(40)
root.left.left, root.left.right = BST(10), BST(25)
root.right.left, root.right.right = BST(35), BST(45)
```

연습문제

- **키값들이 오름차순으로 출력되도록 bst_sorted() 메소드를 작성해 보자.**

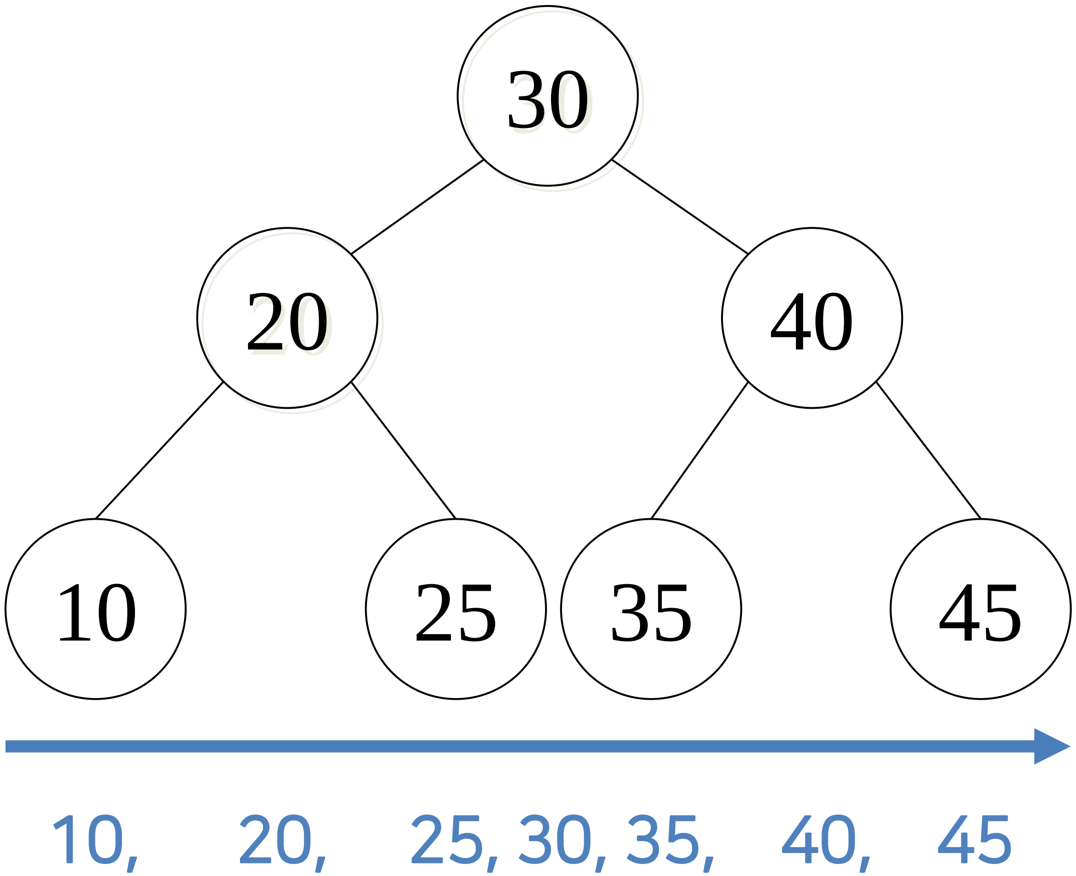



```
def bst_sorted(bst):
    pass


bst_sorted(root)
```

```
10
20
25
30
35
40
45
```

트리 순회하기

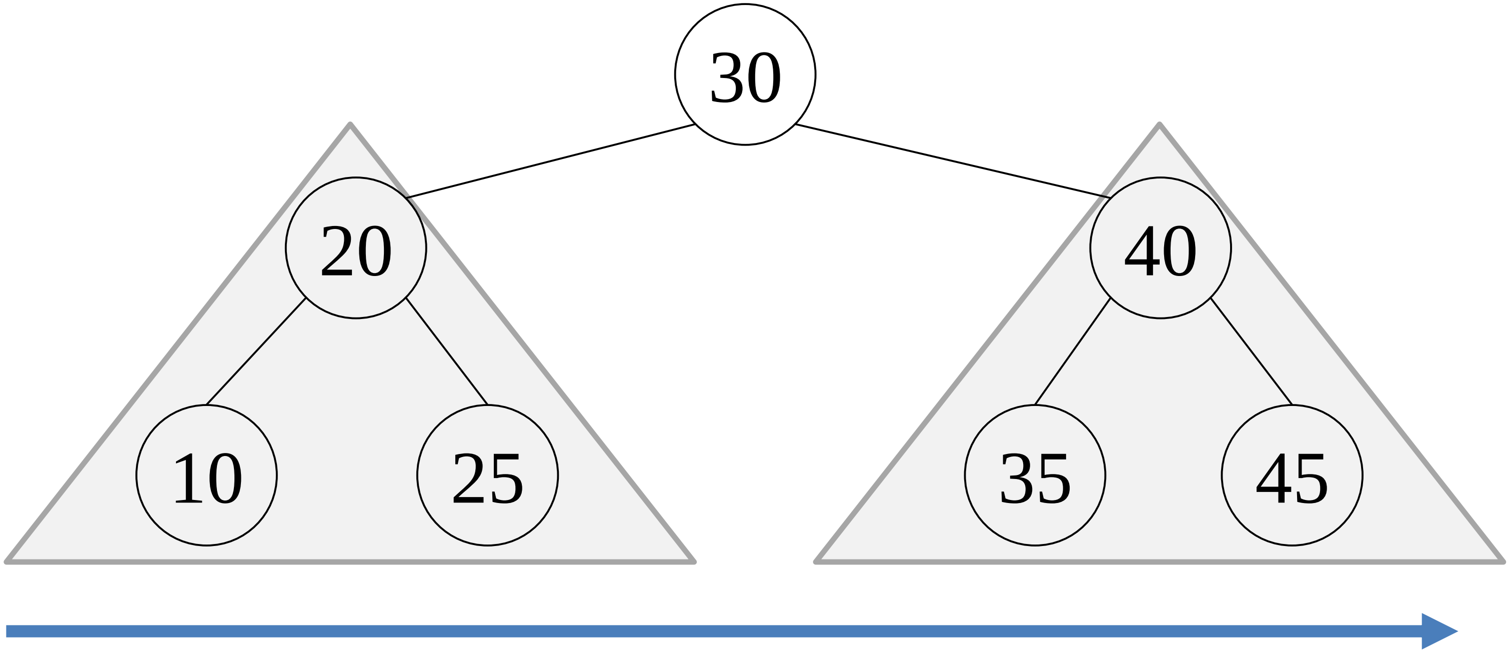
- 관찰: 왼쪽에서 오른쪽으로 출력된다.




```
10
20
25
30
35
40
45
```

트리 순회하기

- **재귀적인 문제 해석**
  - 루트 노드의 관점에서 보면:

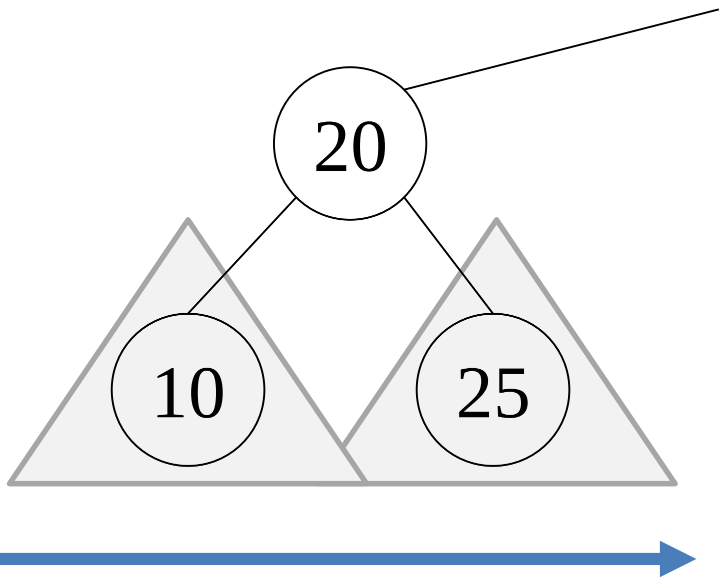


<왼쪽 서브트리>, 자기자신, <오른쪽 서브트리>

<10, 20, 25>, 30, <35, 40, 45>

트리 순회하기

- **재귀적인 문제 해석**
  – 왼쪽자식(20)을 루트로 하는 BST의 관점:



<왼쪽 서브트리>, 자기자신, <오른쪽 서브트리>

<10>, 20, <25>

트리 순회하기

- **BST에서 오름차순 출력**
  - 재귀 알고리즘

```
def bst_sorted(bst):
    if not bst:
        return

    bst_sorted(bst.left)
    print(bst.key)
    bst_sorted(bst.right)
```

연습문제

- 내림차순으로 출력해 보자.

```
def bst_sorted_desc(bst):
      pass
```

```
45
40
35
30
25
20
10
```

트리 순회하기

- **트리 순회(Tree Traversal) 문제**
  - **트리의 각 노드를 한 번씩 방문한다.**

  – 루트 노드의 위치를 기준으로
    - **전위 순회 pre-order trav.** : 루트 – 왼쪽 – 오른쪽
    - **중위 순회 in-order trav.** : **왼쪽 – 루트 - 오른쪽**
    - **후위 순회 post-order trav.** : 왼쪽 – 오른쪽 - 루트

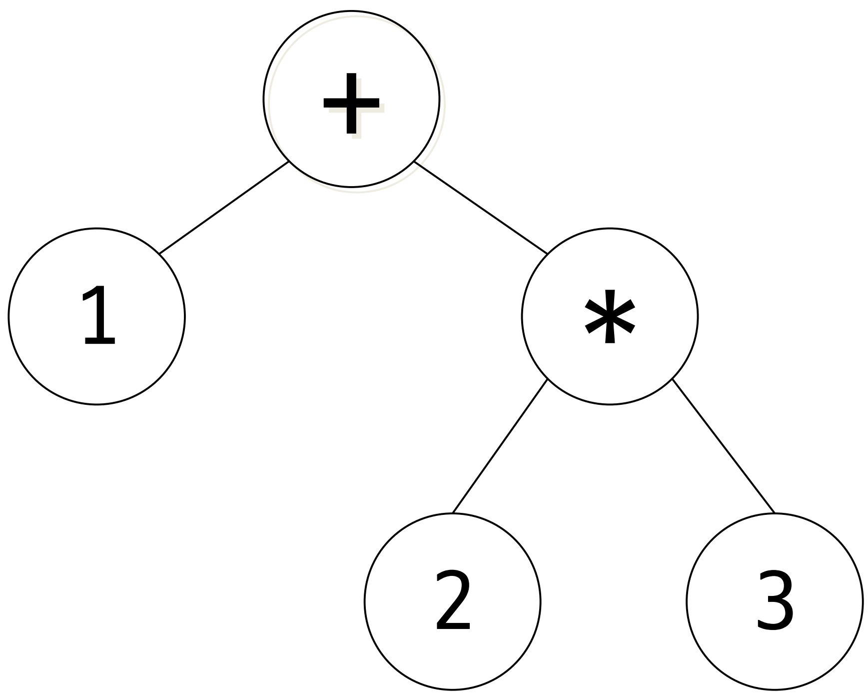
트리 순회하기

- **In-order Traversal**
  - BST가 아니라 일반적인 트리에 적용한 경우

```
def in_order_trav(t):
      if not t:
            return

      in_order_trav(t.left)
      print(t.key)
      in_order_trav(t.right)
```

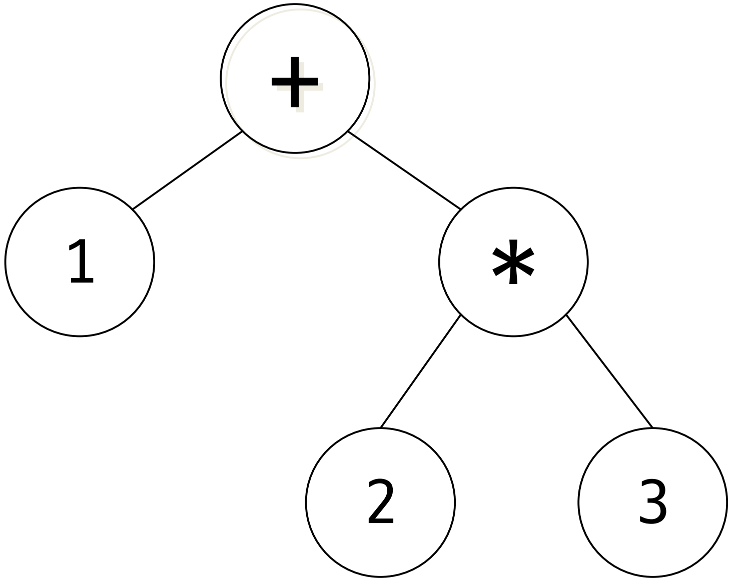
트리 순회 예시

- *exp* = 1 + 2 * 3
- *exp* 의 트리 표현



연습문제

- *exp* = 1 + 2 * 3



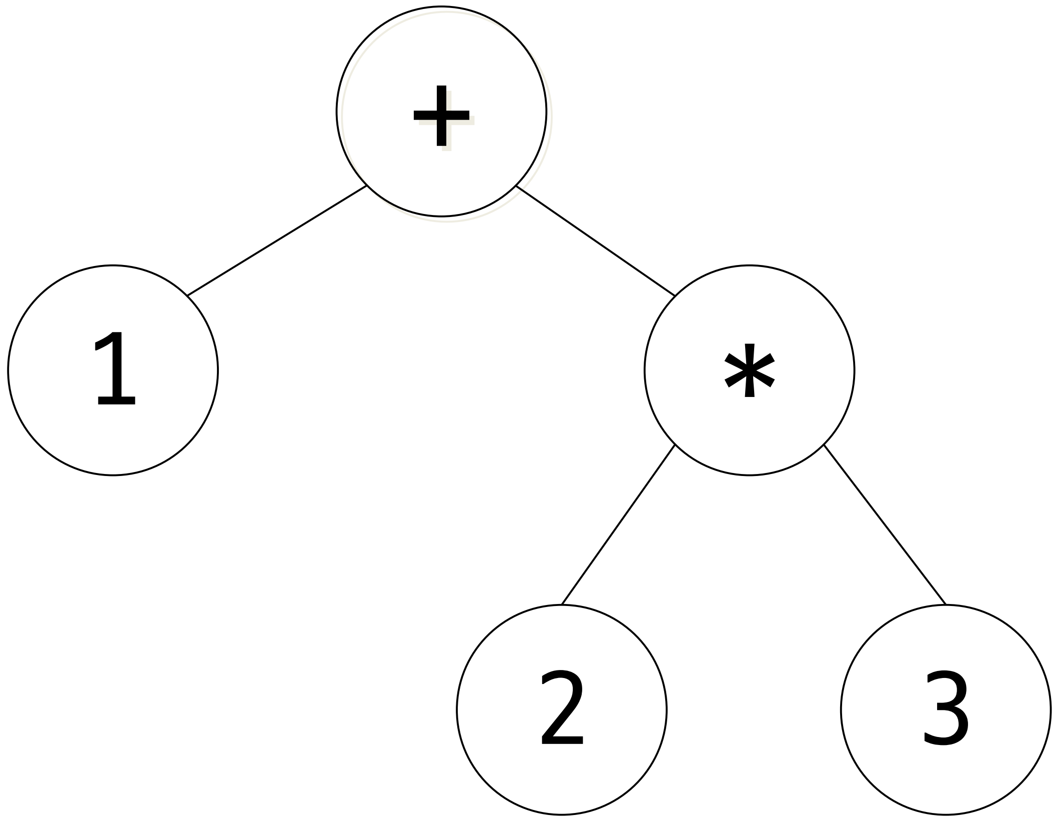
Pre-order trav.:

In-order trav.:

Post-order trav.:

트리 순회 예시

- In-order → 사람이 읽기 좋은 표현
- Post-order → 스택Stack을 써서 연산할 수 있음



그래프란?

그래프 Graph?

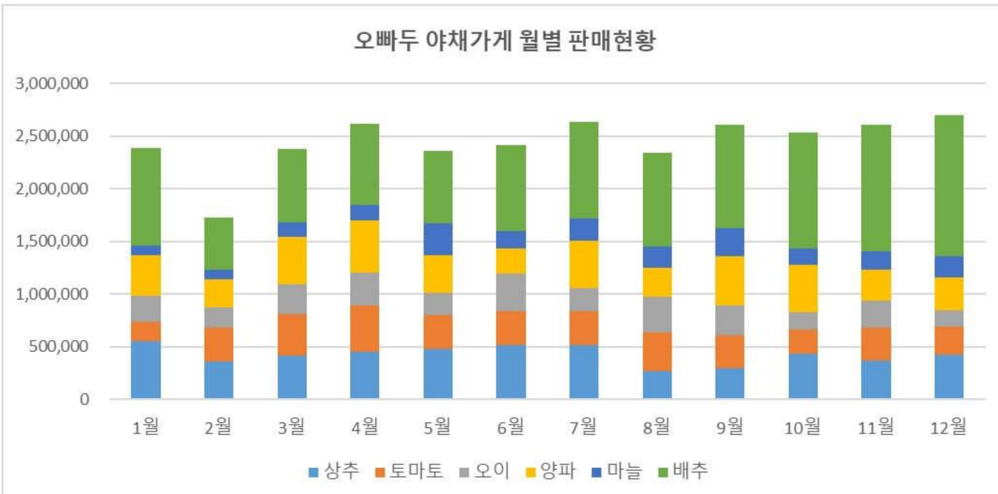


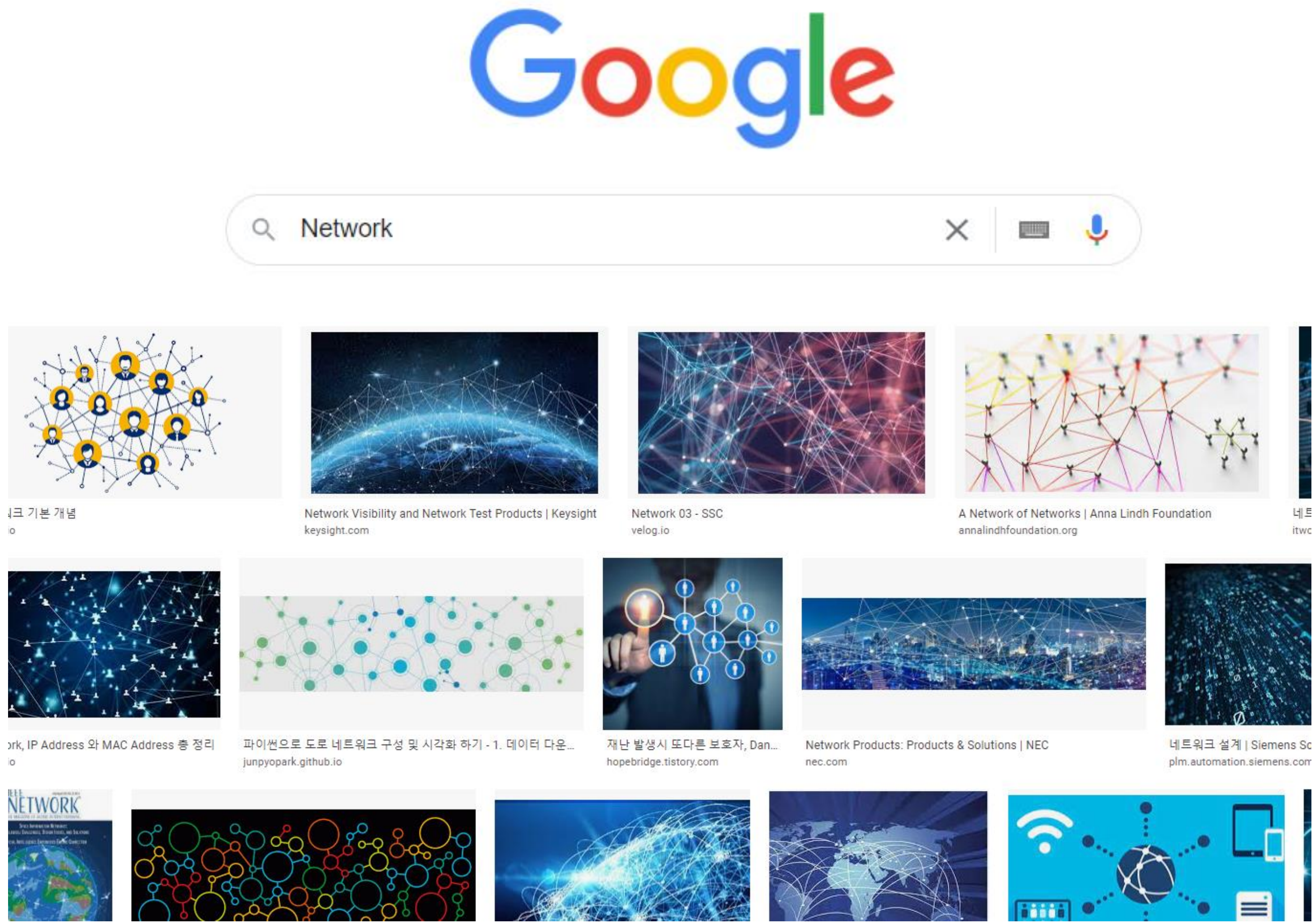
차트 chart (O)

그림 figure (O)

그래프 graph (?)

출처: oppadu.com, medium.com

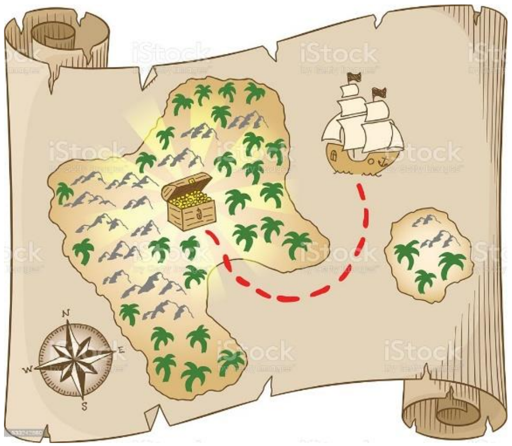
그래프 Graph?


Network
Network Visibility and Network Test Products | Keysight
keysight.com
Network 03 - SSC
velog.io
A Network of Networks | Anna Lindh Foundation
annalindhfoundation.org
IP Address 와 MAC Address 총 정리
파이썬으로 도로 네트워크 구성 및 시각화 하기 - 1. 데이터 다운...
junpyopark.github.io
재난 발생시 또다른 보호자, Dan...
hopebridge.tistory.com
Network Products: Products & Solutions | NEC
nec.com
네트워크 설계 | Siemens
NETWORK

그래프 Graph?

- 현상이나 사물을 정점vertex과 간선edge으로 표현한 것

- **현실 세계에서 그래프로 표현 가능한 현상은 무수히 많다.**
  - 망(network)이라고 부르는 것들
    - IP망, 전화망, 사설망, SNS망, …
  - 지도
  - 네비게이션
  - …



  - 그래프는 이런 현상들을 추상화한 표현

그래프

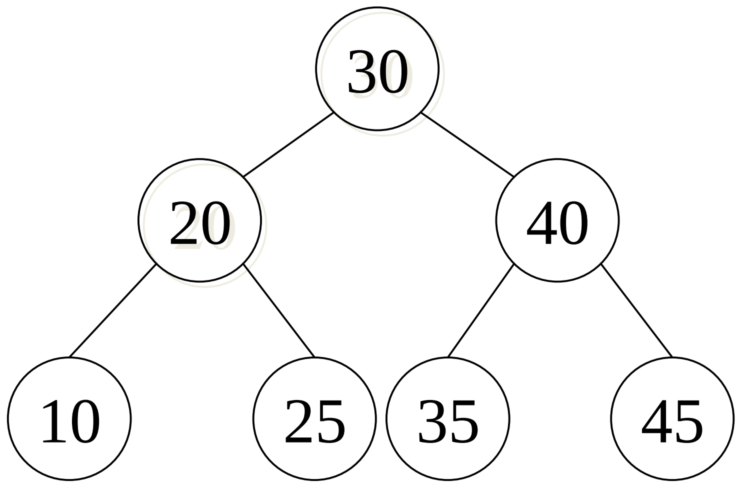
- **현상이나 사물을 정점vertex과 간선edge으로 표현한 것**

- **Graph $G = (V, E)$**
  - V: 정점 집합, E: 간선 집합

- **두 정점이 간선으로 연결되어 있으면 인접adjacent하다고 한다.**
  - 간선은 두 정점의 관계를 나타낸다.
  - 간선으로 연결된 두 정점을 서로의 **이웃(neighbor)**이라고 부른다.

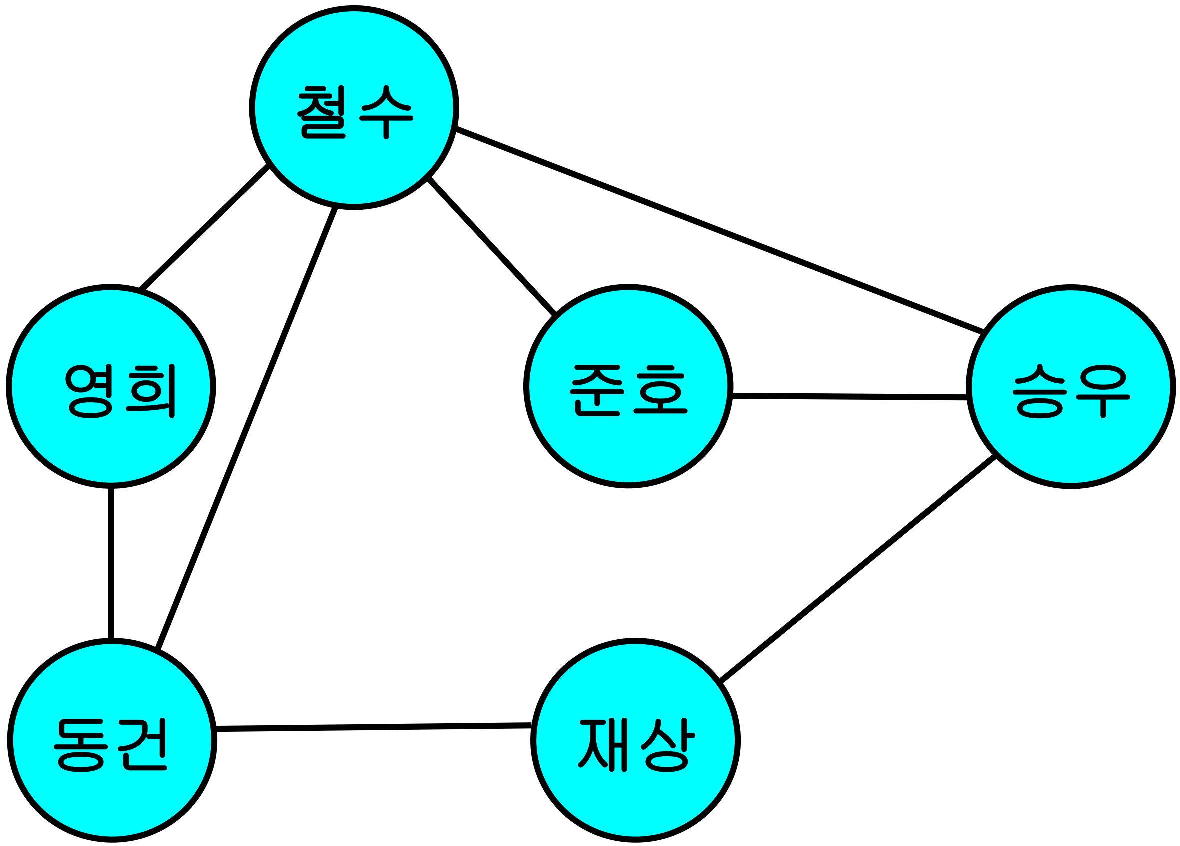
그래프

- **트리tree도 그래프의 일종**
  - 정의: 싸이클cycle이 없는 그래프
  - 특징: **|E| = |V| - 1** ← **어째서?**



V = {10, 20, 25, 30, 35, 40, 45}
E = {(30, 20), (30, 40), (20, 10), (20, 25), (40, 35), (40, 45)}
|V| = 7, |E| = 6

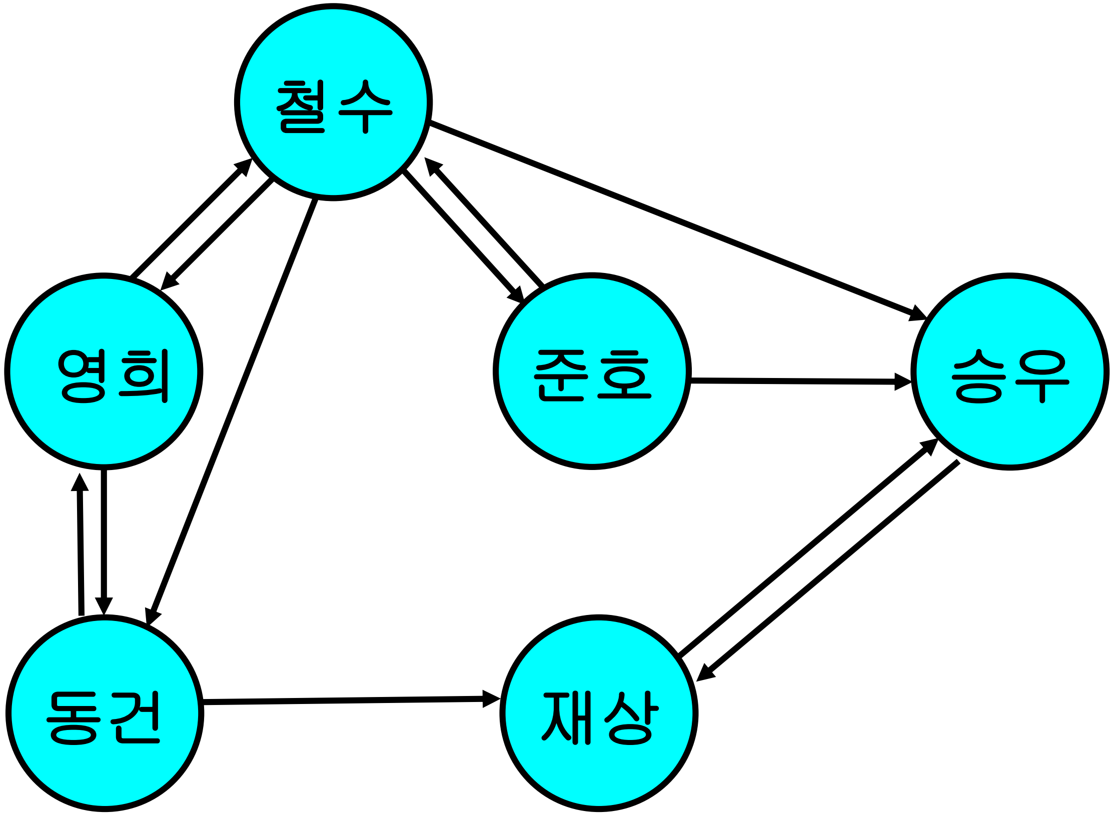
그래프의 예



사람들간의 친분 관계를 나타낸 그래프

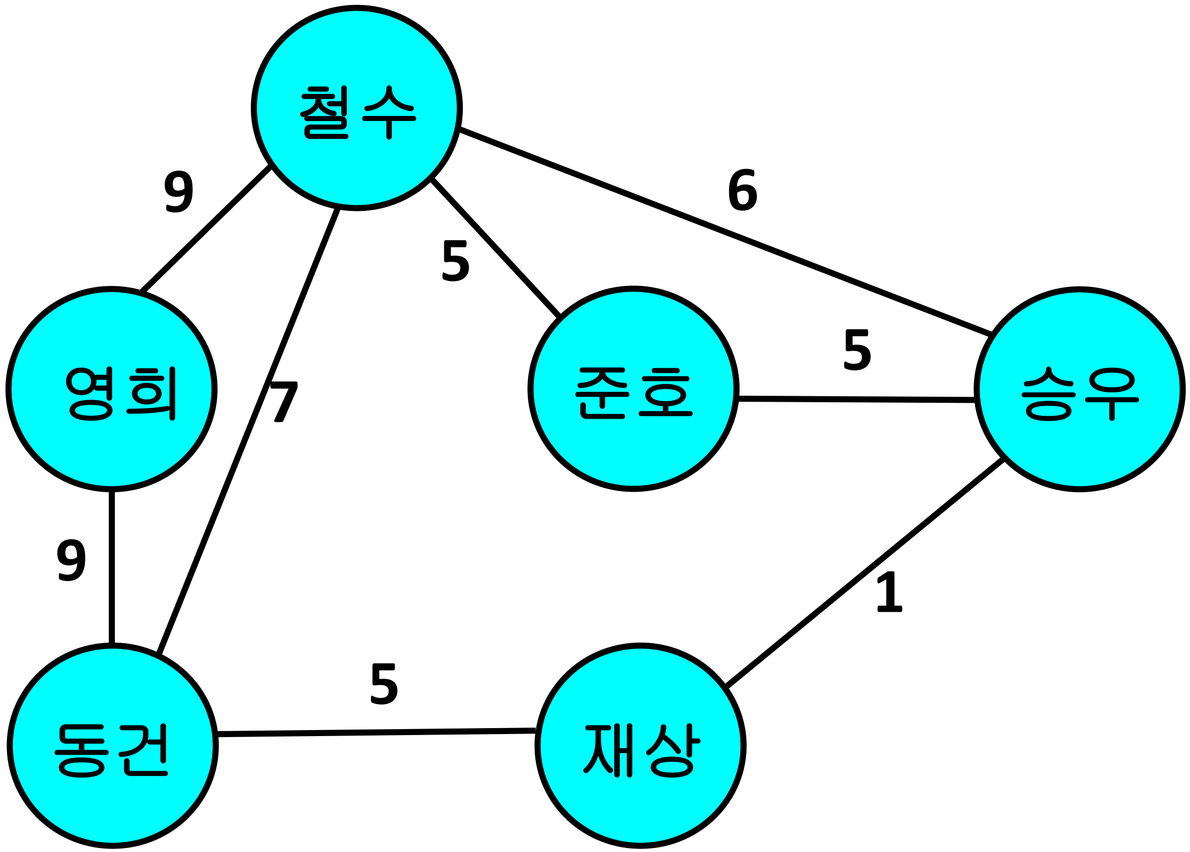
그래프의 예

유향 그래프 **directed graph, digraph**



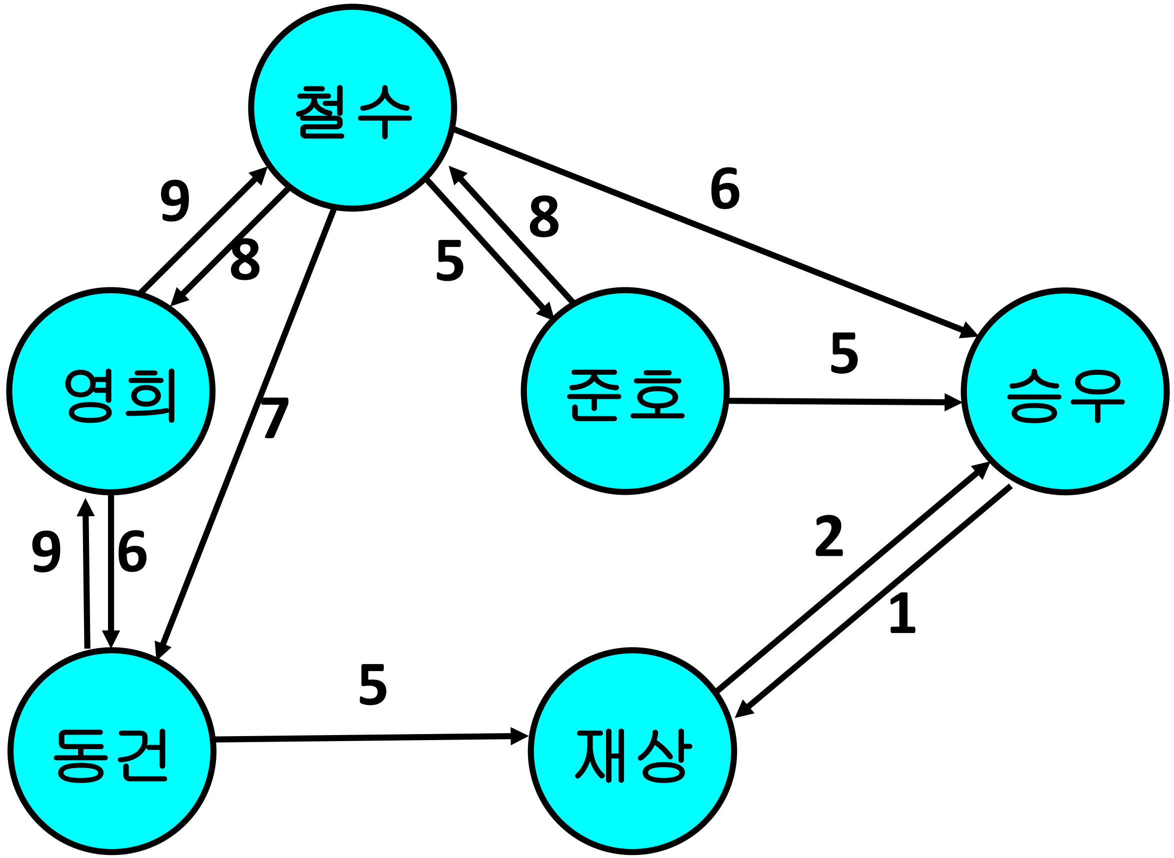
방향을 고려한 친분관계 그래프

그래프의 예



친밀도를 가중치로 나타낸 친분관계 그래프

그래프의 예



가중치를 가진 유향 그래프

그래프의 특징에 따른 분류

그래프의 특징에 따른 분류

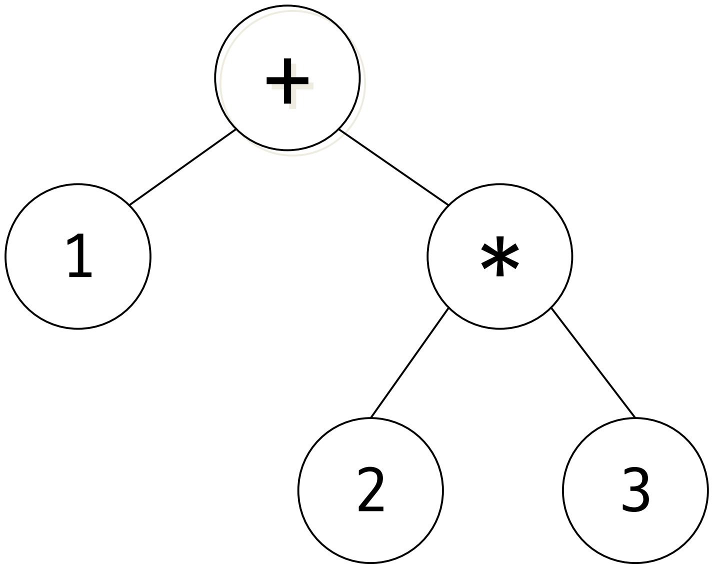
- **간선의 종류**
  - 유향(단방향) directed / 무향(양방향) undirected
- **간선의 가중치 유무**
  - weighted / unweighted
- **싸이클의 유무**
  - cyclic / acyclic**(트리tree: 싸이클이 없는 그래프)**
  - *__cycle__: 어떤 정점에서 출발해서 제자리로 돌아올 수 있음
- **연결 여부**
  - 연결connected 그래프, 비-연결disconnected 그래프

그래프의 특징에 따른 분류

- **간선의 종류**
  – directed / undirected

- **간선의 가중치 유무**
  – weighted / unweighted

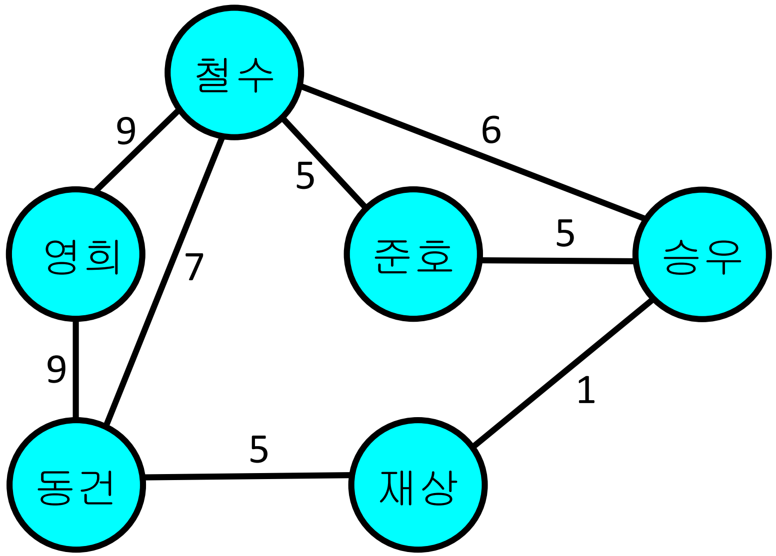
- **싸이클의 유무**
  – cyclic / acyclic



directed?
weighted?
cycle?

그래프의 특징에 따른 분류

- **간선의 종류**
  - directed / undirected
- **간선의 가중치 유무**
  - weighted / unweighted
- **싸이클의 유무**
  - cyclic / acyclic

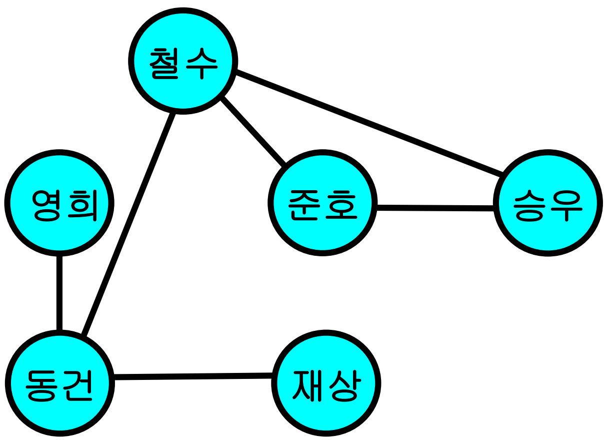


directed?
weighted?
cycle?

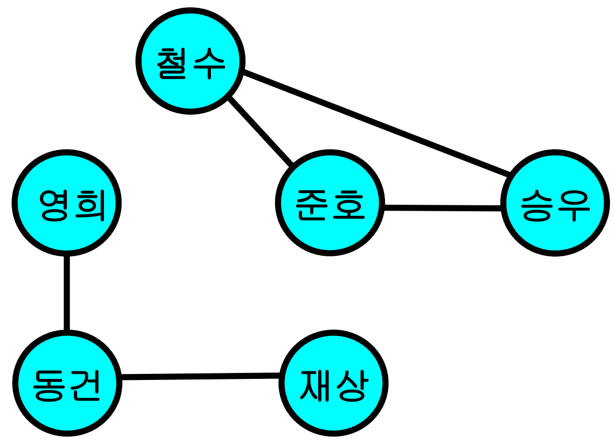
그래프의 특징에 따른 분류

- 간선의 종류, 간선의 가중치 유무, 싸이클의 유무, **연결 여부**

- **연결connected 그래프**
  - 모든 정점 쌍 (x, y)에 대해 x ~~> y 경로가 존재



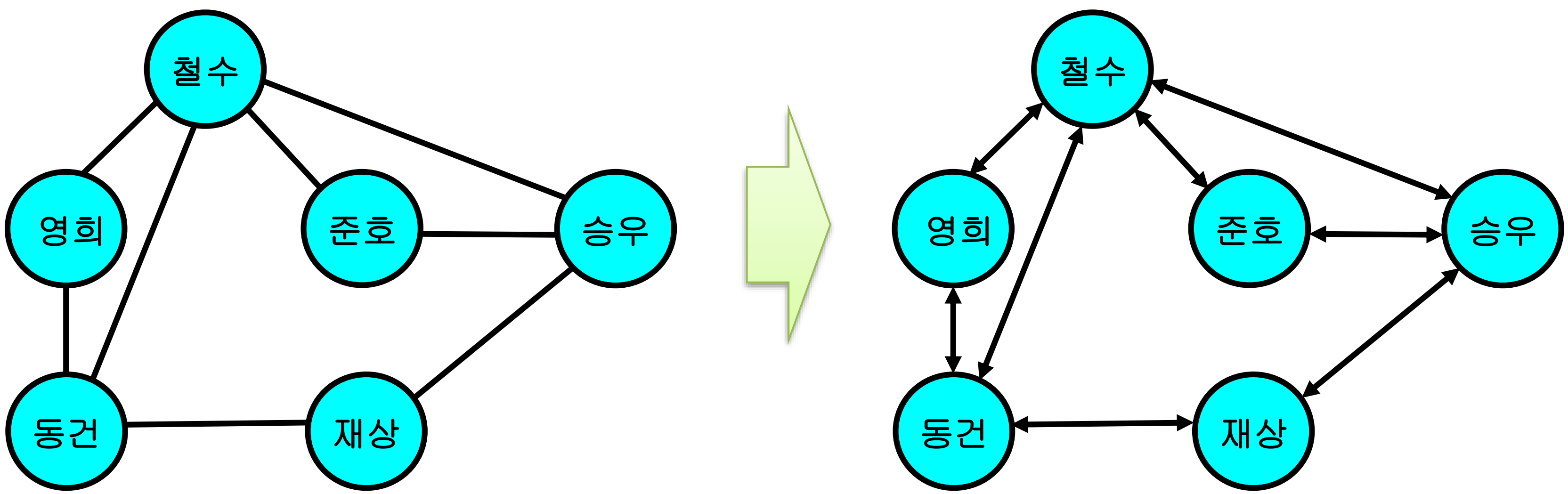
connected



disconnected graph

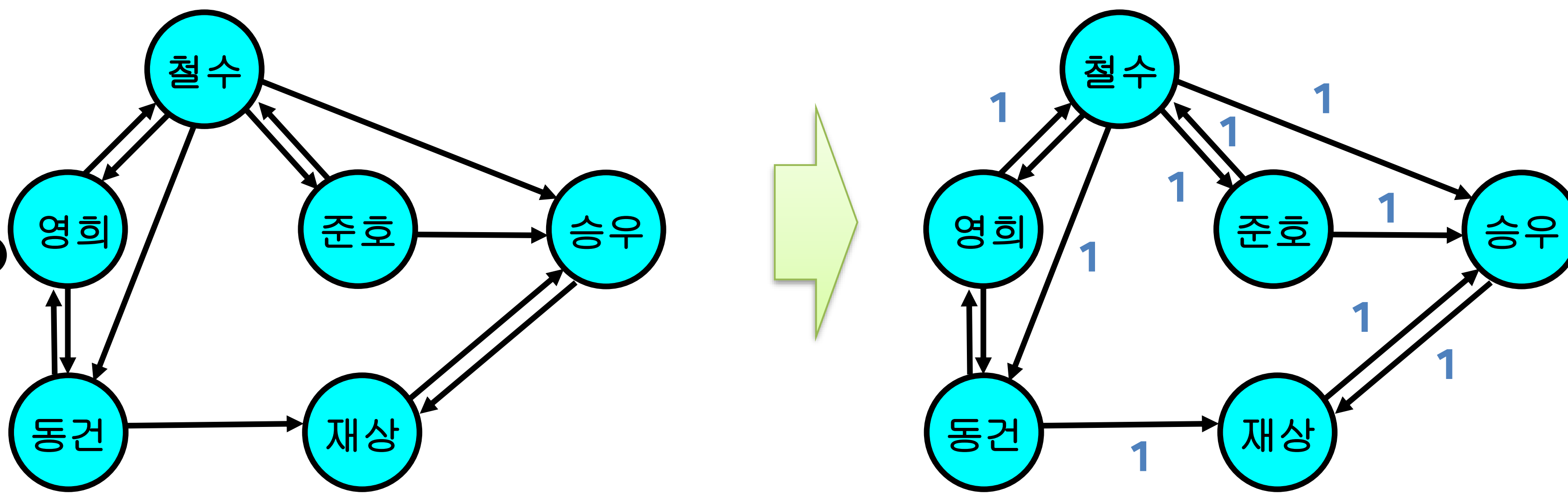
그래프의 특징에 따른 분류

- 무향 그래프는 유향 그래프로 표현 가능



그래프의 특징에 따른 분류

- 가중치 없는 그래프는
- “모든 간선의 가중치가 같은” 그래프로 표현 가능



directed graph

그래프의 분류 사용 예

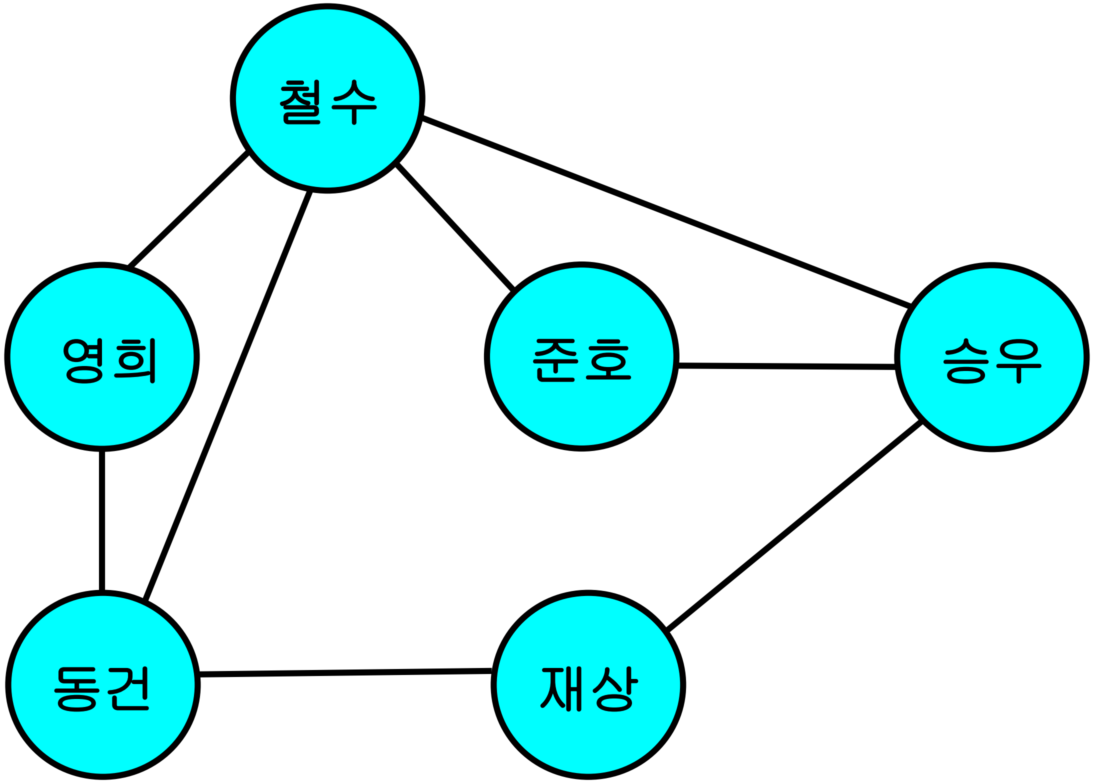
- **트리 순회(Tree Traversal) 문제**
  - 트리의 각 노드를 한 번씩만 방문한다.

- **→ 트리 순회 문제**
  - 입력: **Unweighted, undirected acyclic graph**
  - 출력: 각 노드를 한 번씩 방문할 때 방문 순서

그래프의 특징에 대해 따로 언급이 없으면
일반적으로 **가중치 없는 무향 그래프**

연습문제

- 아래 그래프에서 어떤 간선들을 없애야 트리 (acyclic graph)가 되는가?



그래프의 표현

그래프의 표현

- **그래프의 표현 방법**
  - **인접행렬** Adjacency Matrix
  - **인접리스트** Adjacency List
  - (인접배열)

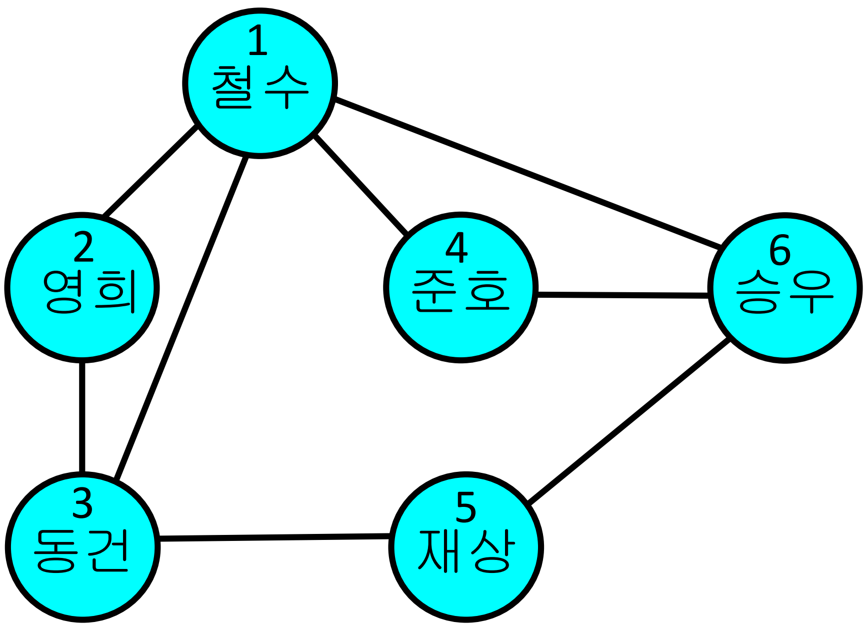
그래프의 표현 1: 인접행렬

- **인접행렬: *N* x *N* 행렬로 표현**
  - N: 정점의 총 수

  - 인접 행렬을 L이라고 하면

  - **L(*i*, *j*) = 1** : 정점 *i* 와 정점 *j* 사이에 간선이 있음
  - **L(*i*, *j*) = 0** : 정점 *i* 와 정점 *j* 사이에 간선이 없음

인접행렬

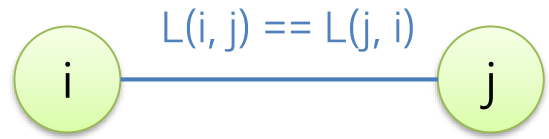


	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	1	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	1	0	0	1
6	1	0	0	1	1	0

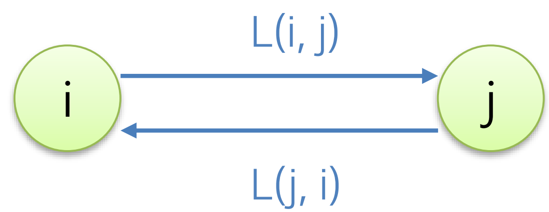
무향 그래프의 예

그래프의 표현 1: 인접행렬

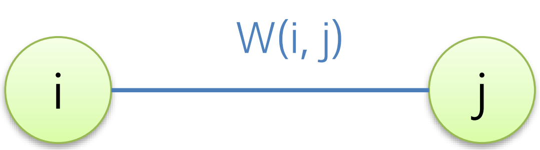
- 인접행렬: *N* x *N* 행렬로 표현
  - 무향 그래프
    - L(*i*, *j*) == L(*j*, *i*)

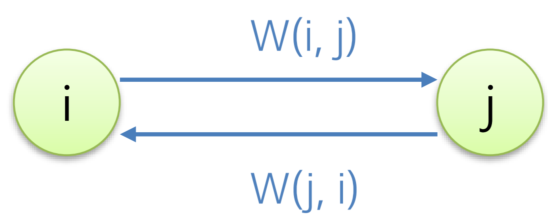


  - 유향 그래프
    - L(*i*, *j*) != L(*j*, *i*)

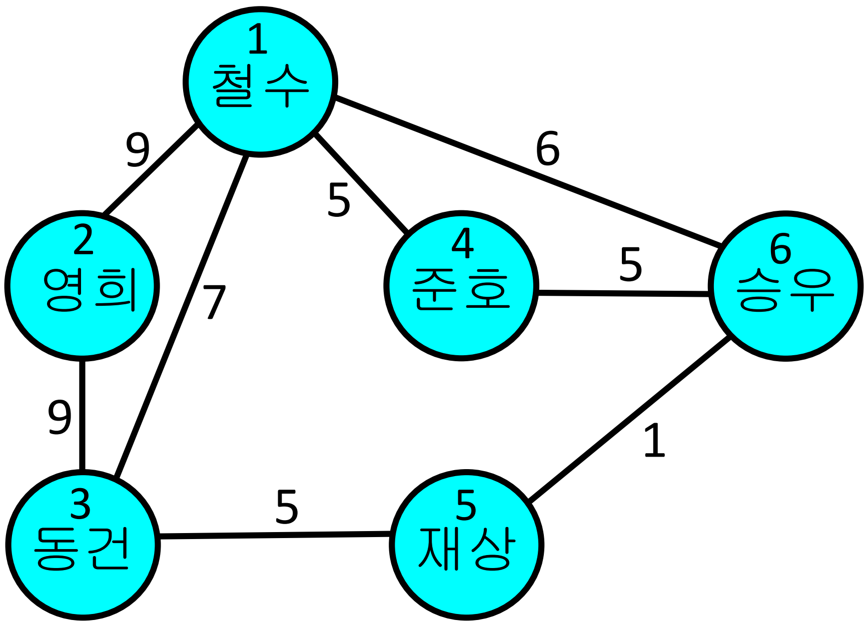


  - 가중치 있는 그래프의 경우
    - W(*i*, *j*) = 정점 *i*와 정점 *j*를 잇는 간선 $e_{i,j}$의 가중치





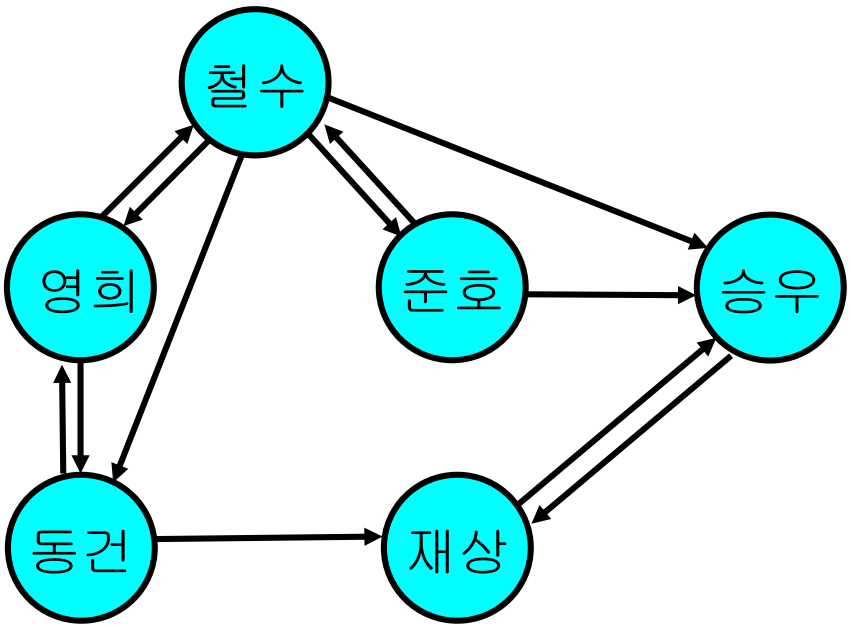
인접행렬



	1	2	3	4	5	6
1	0	9	7	5	0	6
2	9	0	9	0	0	0
3	7	9	0	0	5	0
4	5	0	0	0	0	5
5	0	0	5	0	0	1
6	6	0	0	5	1	0

가중치 있는 무향 그래프의 예

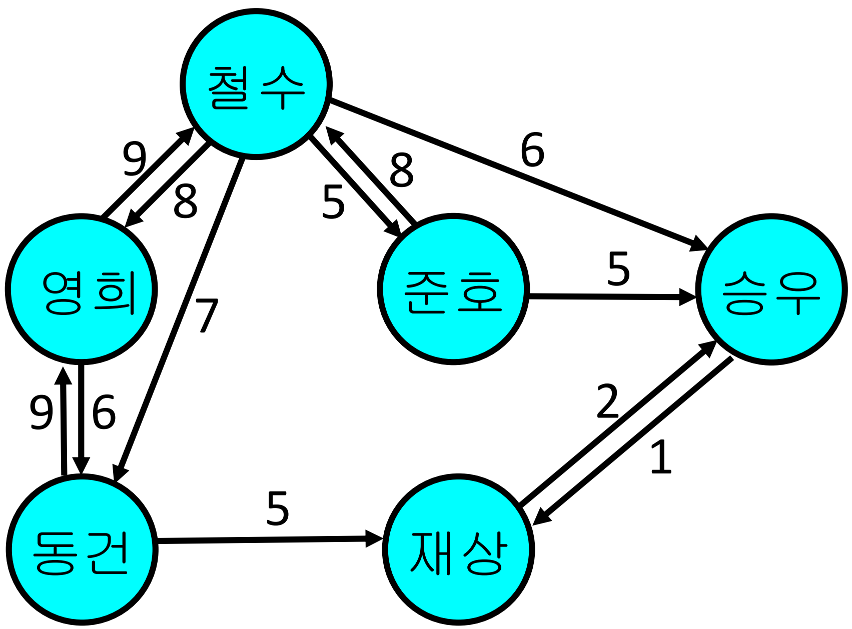
인접행렬



	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	0	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	0	0	0	1
6	0	0	0	0	1	0

유향 그래프의 예

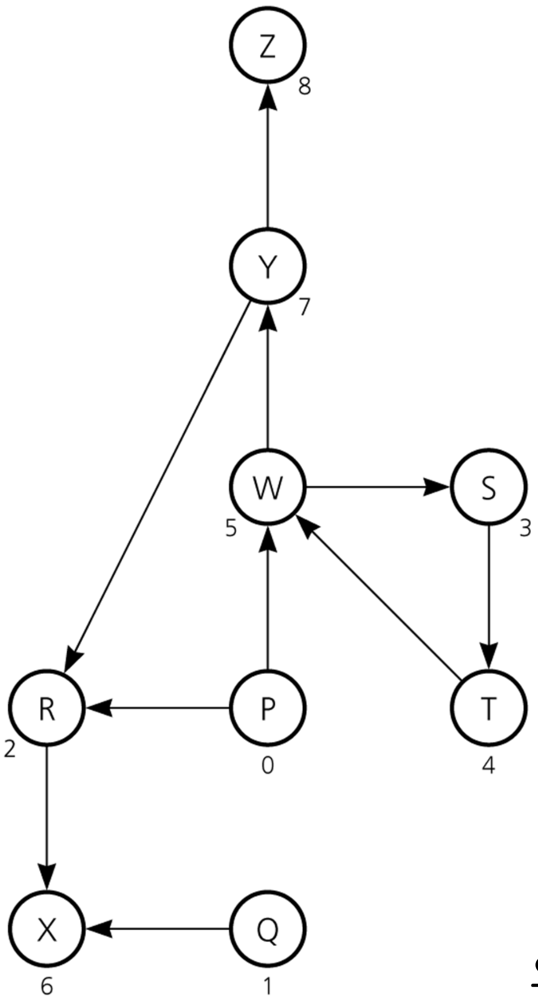
인접행렬



	1	2	3	4	5	6
1	0	8	7	5	0	6
2	9	0	6	0	0	0
3	0	9	0	0	5	0
4	8	0	0	0	0	5
5	0	0	0	0	0	2
6	0	0	0	0	1	0

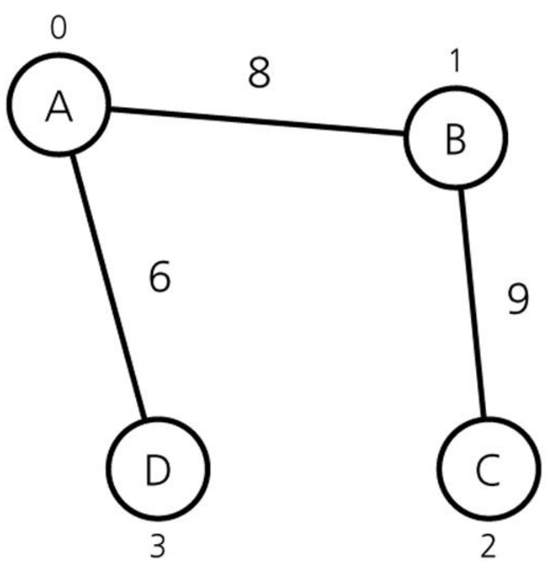
가중치 있는 유향 그래프의 예

인접행렬



		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

유향 그래프의 다른 예



		0 A	1 B	2 C	3 D
0	A	∞	8	∞	6
1	B	8	∞	9	∞
2	C	∞	9	∞	∞
3	D	6	∞	∞	∞

가중치 있는 그래프의 다른 예

연습문제

- 아래와 같은 그래프가 주어졌을 때, (리스트를 배열이라고 생각하고) 이 그래프의 인접행렬을 만들어 보자.

```
V = {0, 1, 2, 3, 4}
N = len(V)
E = {(0, 1), (0, 2), (1, 3), (1, 4), (2, 4), (3, 4)}
# E |= {(y, x) for x, y in E}   # 대칭으로 만들어줌
G = (V, E)  # undirected, unweighted

adjMatrix = [[0] * N for _ in range(N)]  # NxN matrix

# 여기에 작성


for row in adjMatrix:
    print(row)
```

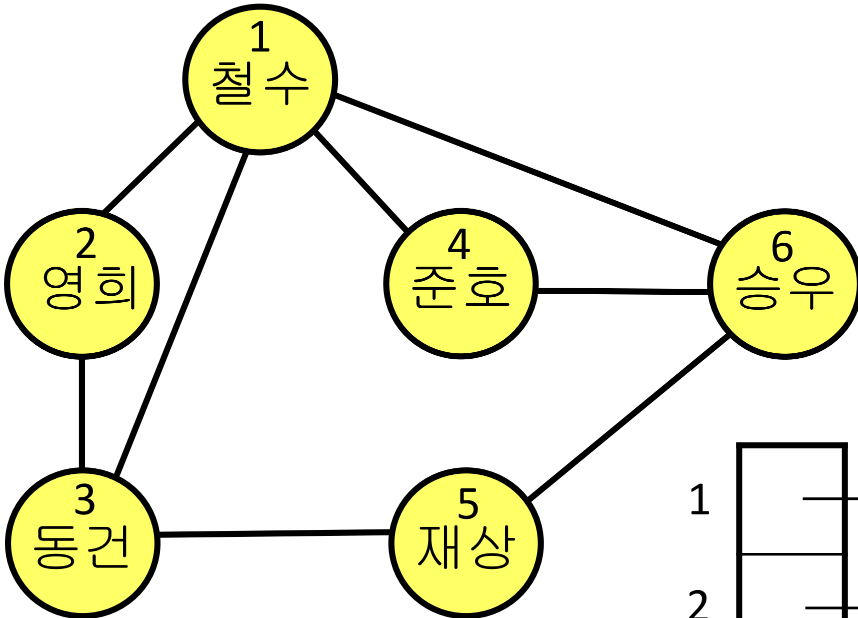
```
[0, 1, 1, 0, 0]
[1, 0, 0, 1, 1]
[1, 0, 0, 0, 1]
[0, 1, 0, 0, 1]
[0, 1, 1, 1, 0]
```

그래프의 표현 2: 인접리스트

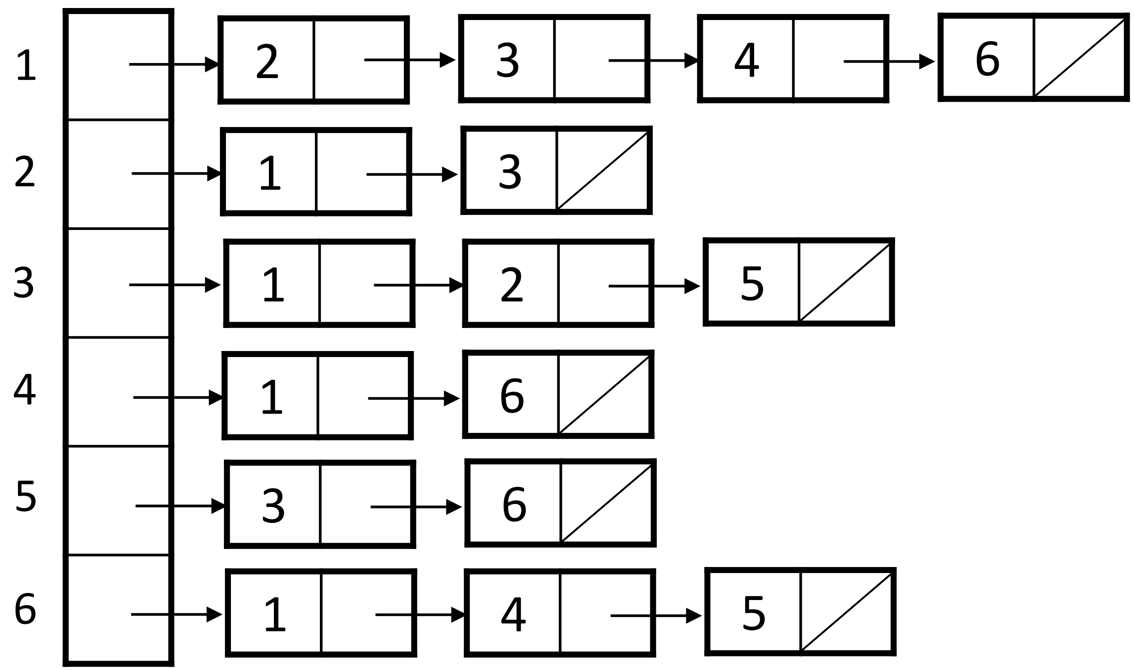
- **인접 리스트: N 개의 연결 리스트로 표현**
  - i 번째 리스트는 정점 i 에 인접한 정점들을 리스트로 연결

- **가중치 있는 그래프의 경우**
  - 리스트에 가중치도 보관한다.

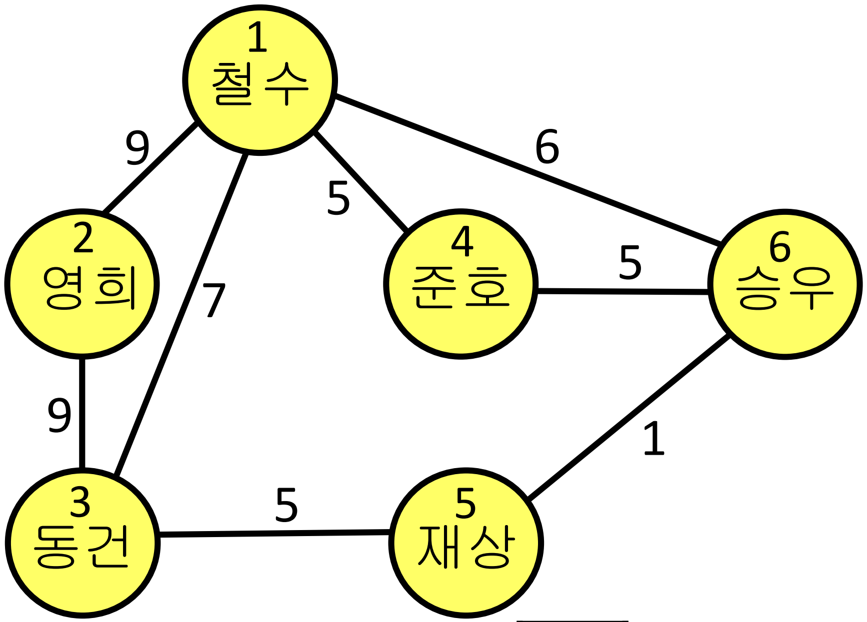
인접 리스트



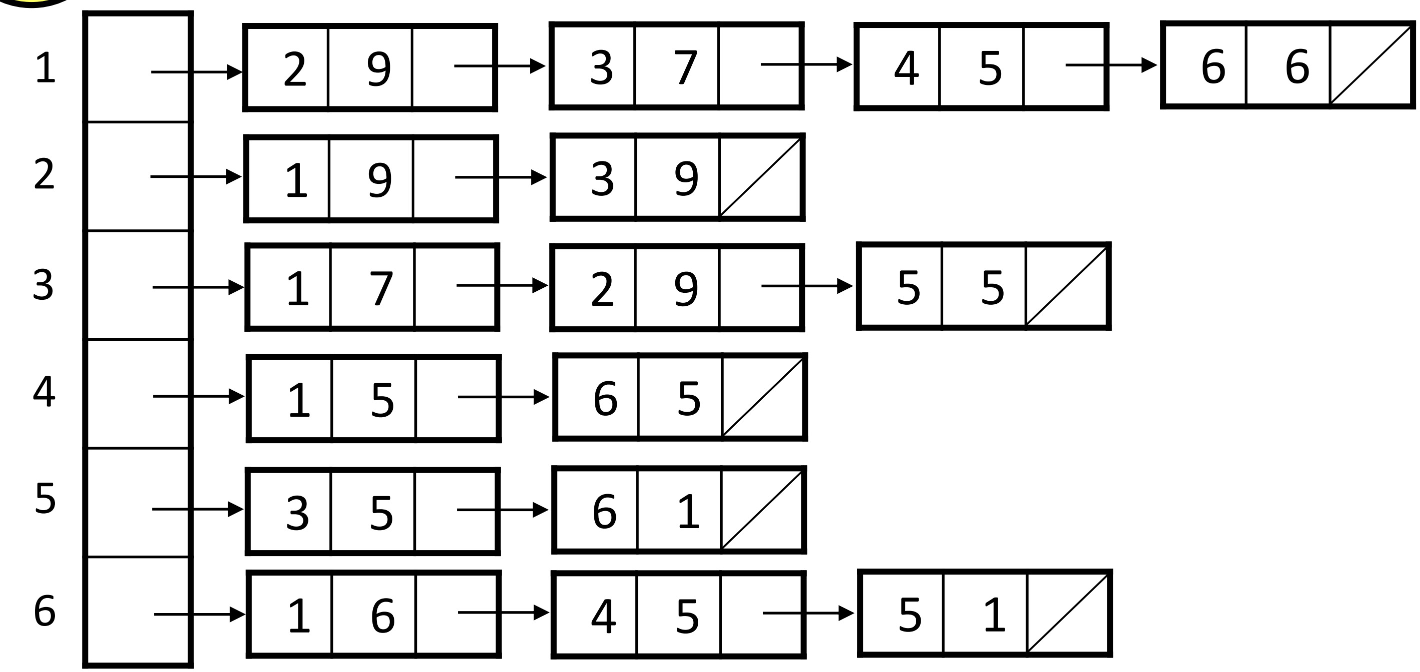
무향 그래프의 예



인접 리스트



가중치 있는 그래프의 예



연습문제

- **주어진 그래프로 인접리스트를 만들어 보자.**

```
V = {0, 1, 2, 3, 4}
N = len(V)
E = {(0, 1), (0, 2), (1, 3), (1, 4), (2, 4), (3, 4)}
# E |= {(y, x) for (x, y) in E}
G = (V, E)  # 가중치 없는 무향 그래프

adjList = {v:[] for v in V}

# 여기에 작성


for v in V:
    print(v, '->', adjList[v])
```

```
0 -> [1, 2]
1 -> [4, 0, 3]
2 -> [4, 0]
3 -> [4, 1]
4 -> [3, 1, 2]
```

연습문제

- **아래와 같이 <가중치 있는 유향 그래프>로 바뀐다면?**

```
V = {0, 1, 2, 3, 4}; N = len(V)
# (시작정점, 끝정점, 가중치) 들의 집합
E = {(0, 1, 2), (0, 2, 5), (1, 3, 3), (1, 4, 7), (2, 4, 4), (3, 4, 3)}
G = (V, E)  # 가중치 있는 유향 그래프

adjList = {v:[] for v in V}
# 여기에 작성


print("vertex -> [(neighbor1, weight1), (neighbor2, weight2), ...]")
for v in V:
    print(v, '->', adjList[v])
```

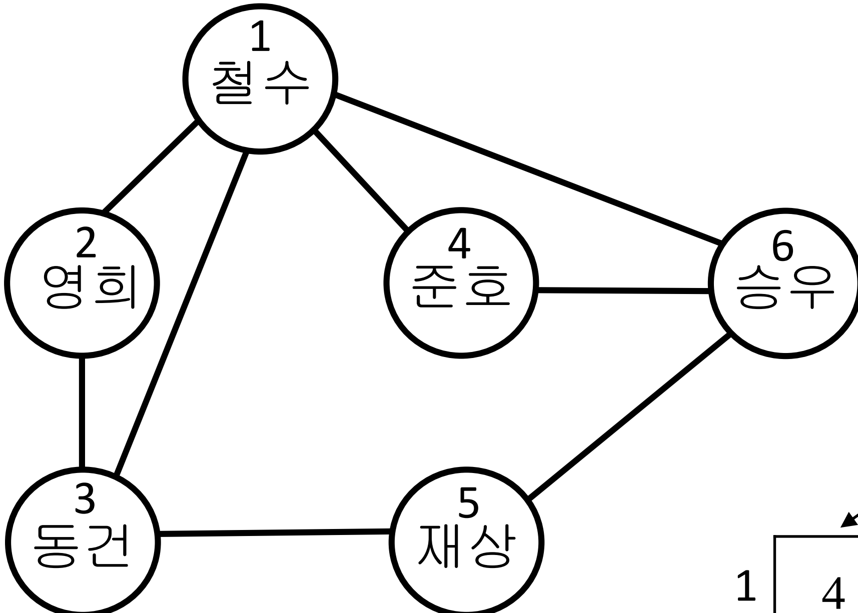
```
vertex -> [(neighbor1, weight1), (neighbor2, weight2), ...]
0 -> [(1, 2), (2, 5)]
1 -> [(3, 3), (4, 7)]
2 -> [(4, 4)]
3 -> [(4, 3)]
4 -> []
```

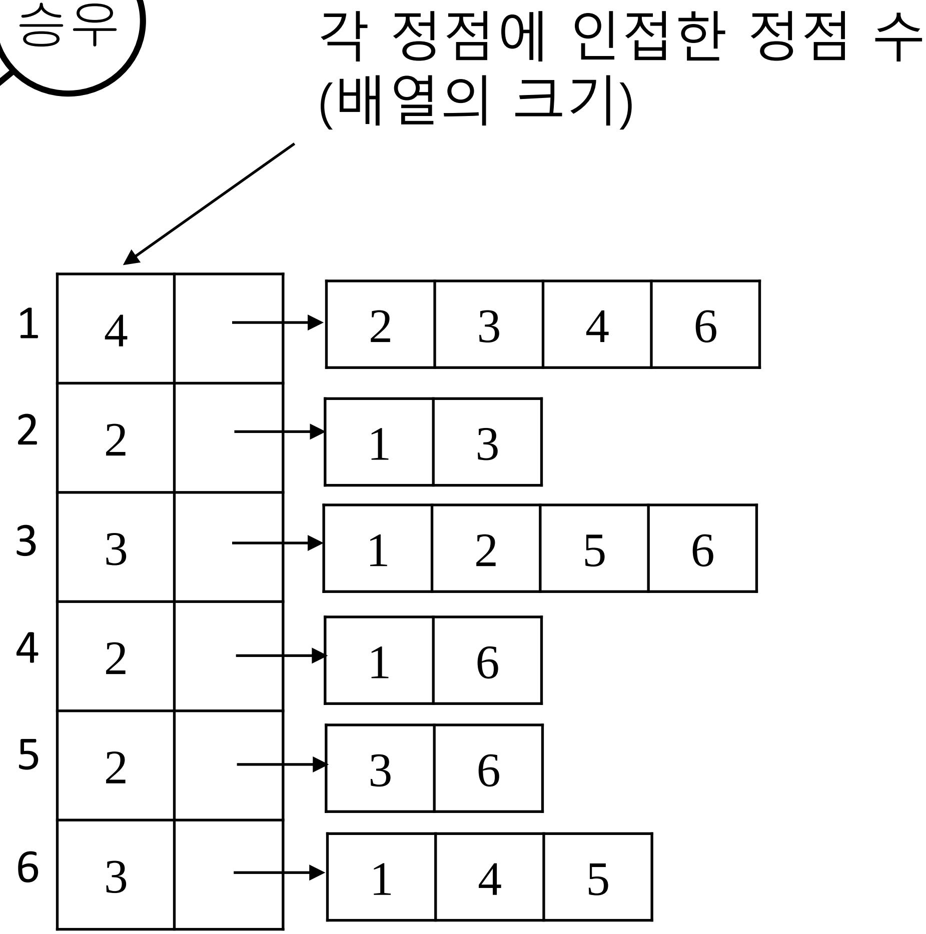
그래프의 표현 3: 인접 배열

- **인접 배열**
  - N 개의 연결 배열로 표현
  - i 번째 배열은 정점 i 에 인접한 정점들을 집합

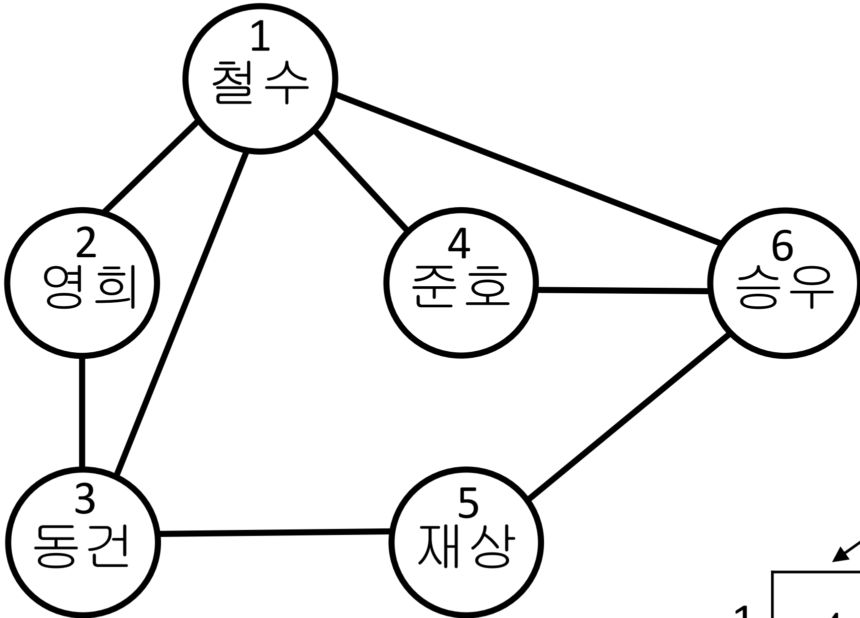
- **가중치 있는 그래프의 경우**
  - 리스트에 가중치도 보관한다.

인접 배열

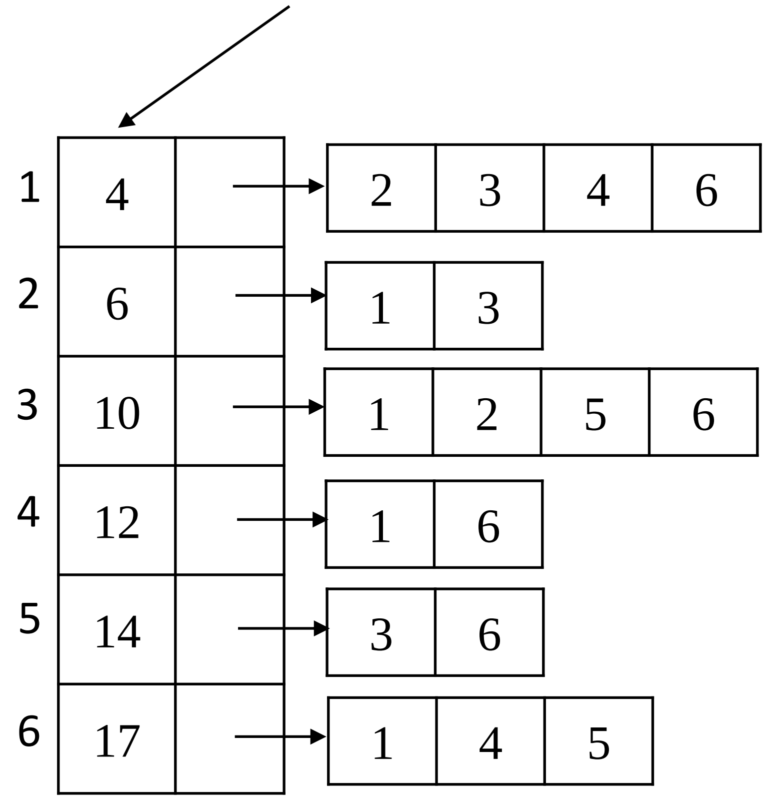

1
철수
2
영희
3
동건
4
준호
5
재상
6
승우


각 정점에 인접한 정점 수
(배열의 크기)
1 4 → 2 3 4 6
2 2 → 1 3
3 3 → 1 2 5 6
4 2 → 1 6
5 2 → 3 6
6 3 → 1 4 5

인접 배열



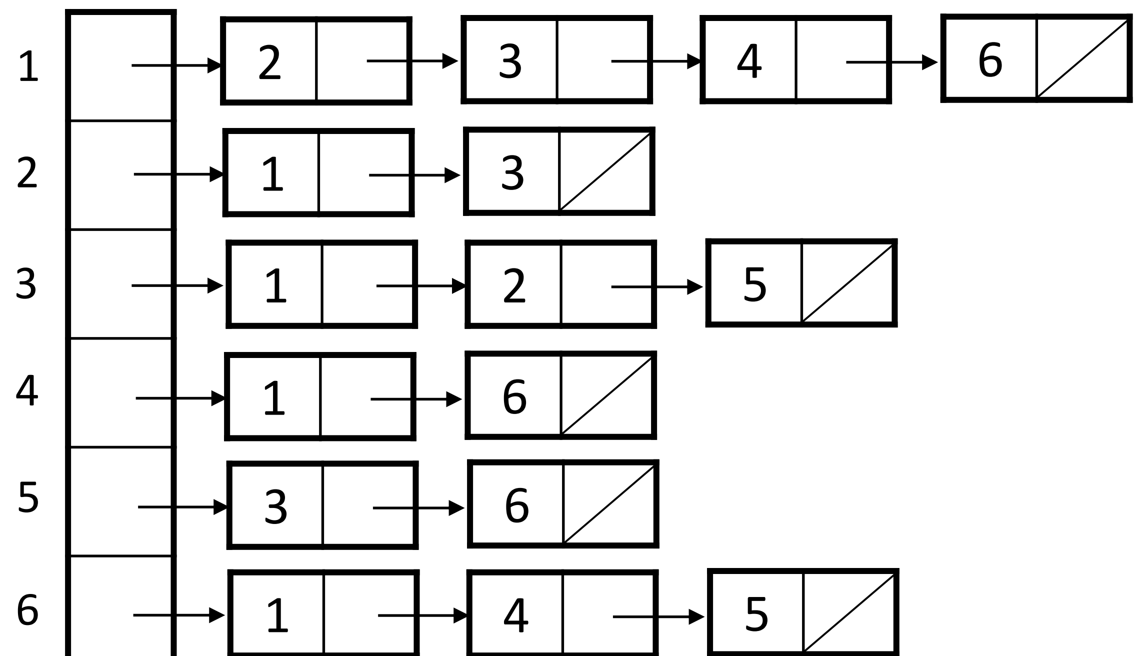
배열에서 각 정점에 인접한
정점 목록의 끝자리



생각해 볼 문제

	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	1	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	1	0	0	1
6	1	0	0	1	1	0

- 인접행렬 표현과 인접리스트 표현을 비교했을 때 장단점은?



NETWORKX

networkx

- **파이썬 실행 환경에 network, matplotlib를 설치한다.**

```
import networkx as nx
from matplotlib import pyplot as plt
```

networkx

- **설치가 끝났으면 아래 예제를 실행해 보자.**

```
import networkx as nx
from matplotlib import pyplot as plt


t = nx.Graph()
t.add_node(30)
t.remove_node(30)
t.add_nodes_from([30, 20, 40, 10, 25, 35, 45])
t.add_edge(30, 20)
# t.remove_edge(30, 20)
t.add_edges_from([(30, 40), (20, 10), (20, 25), (40, 35), (40, 45)])

print("nodes =", t.nodes); print("edges =", t.edges)
print("30's neighbors =", list(t.neighbors(30)))
print(nx.info(t))

nx.draw(t)
# nx.draw(t, with_labels=True, node_size=500, font_color="white")
plt.show()
```

networkx

- networkx: 그래프 시각화 라이브러리
  - 무향/유향 그래프 객체 생성

```
t = nx.Graph()
```

```
digraph = nx.DiGraph()
```

  - 정점, 간선 삽입 삭제

```
t.add_node(30)
t.remove_node(30)
t.add_nodes_from([30, 20, 40, 10, 25, 35, 45])
```

```
t.add_edge(30, 20)
# t.remove_edge(30, 20)
t.add_edges_from([(30, 40), (20, 10), (20, 25), (40,
35), (40, 45)])
```

networkx

– 그래프 정보 출력하기

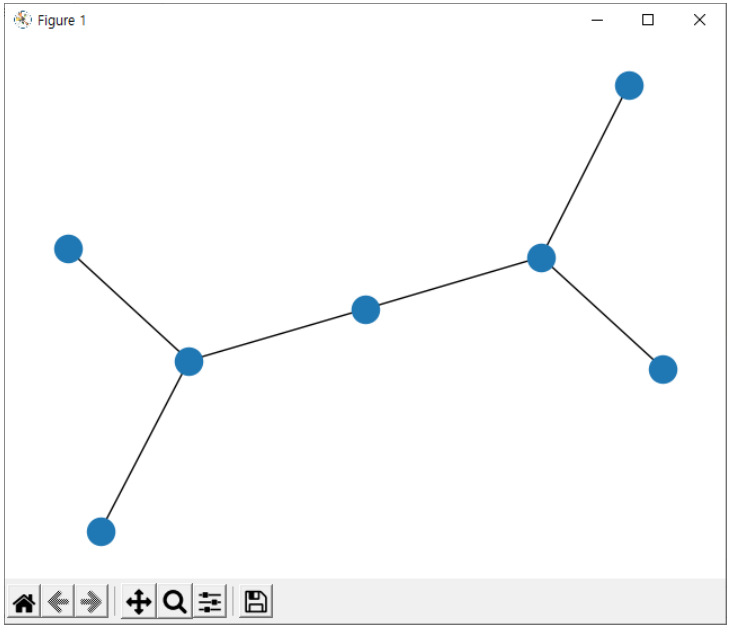
```
print("nodes =", t.nodes)
print("edges =", t.edges)
print("30's neighbors =", list(t.neighbors(30)))
print(nx.info(t))
```

```
nodes = [30, 20, 40, 10, 25, 35, 45]
edges = [(30, 20), (30, 40), (20, 10), (20, 25), (40, 35), (40, 45)]
30's neighbors = [20, 40]
Graph with 7 nodes and 6 edges
```

networkx

– pyplot으로 그래프 그리기

```
nx.draw(t)
# nx.draw(t, with_labels=True, node_size=500,
font_color="white")
plt.show()
```



연습문제

- 앞의 연습문제에서 주어졌던 <가중치 없는 무향 그래프>이다.
- 이 그래프를 networkx를 사용해서 그려 보자.

```
V = {0, 1, 2, 3, 4}
N = len(V)
E = {(0, 1), (0, 2), (1, 3), (1, 4), (2, 4), (3, 4)}
E |= {(y, x) for (x, y) in E}
G = (V, E)
```

연습문제

- **앞의 연습문제에서 주어졌던 <가충치 있는 유향 그래프>이다.**
- **이 그래프를 networkx를 사용해서 그려 보자.**

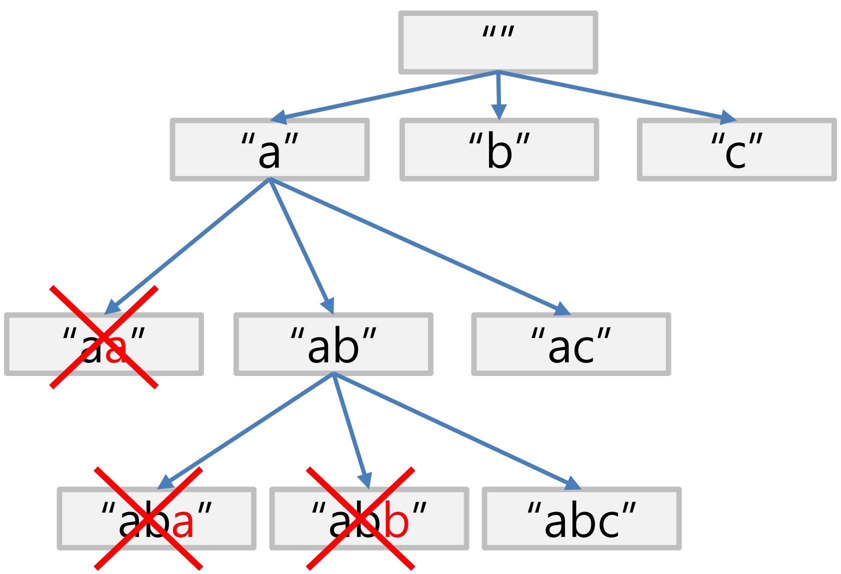
  – networkx Graph 객체의 **add_weighted_edges_from()** 메소드 사용

```
V = {0, 1, 2, 3, 4}; N = len(V)
# (시작정점, 끝정점, 가중치) 들의 집합
E = {(0, 1, 2), (0, 2, 5), (1, 3, 3),
     (1, 4, 7), (2, 4, 4), (3, 4, 3)}
G = (V, E)
```

DFS, BFS

Review: 문제 공간 탐색

- ‘a’, ‘b’, ‘c’를 한 번씩 써서 문자열 만들기



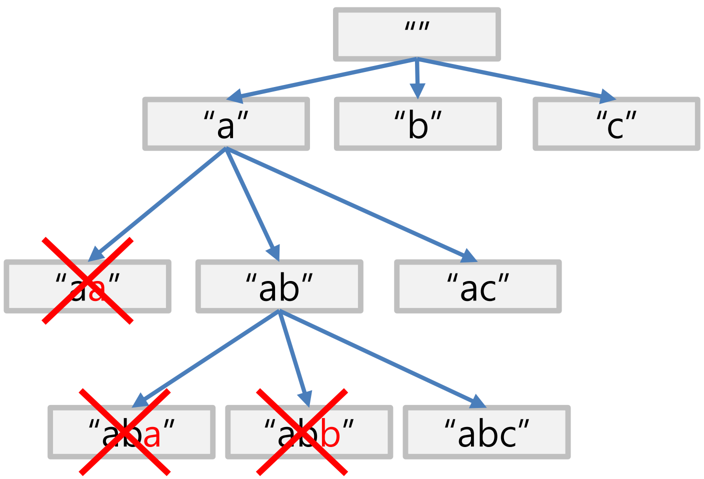

```python
def enum_dict_order(s, used_chars):
    if len(s) == len(char_set):
        print(s)
        return

    for char in char_set:
        if char not in used_chars:
            enum_dict_order(s + char,
                   used_chars | {char})


char_set = list("abc")
enum_dict_order("", set())
```

그래프에서 모든 정점 방문하기

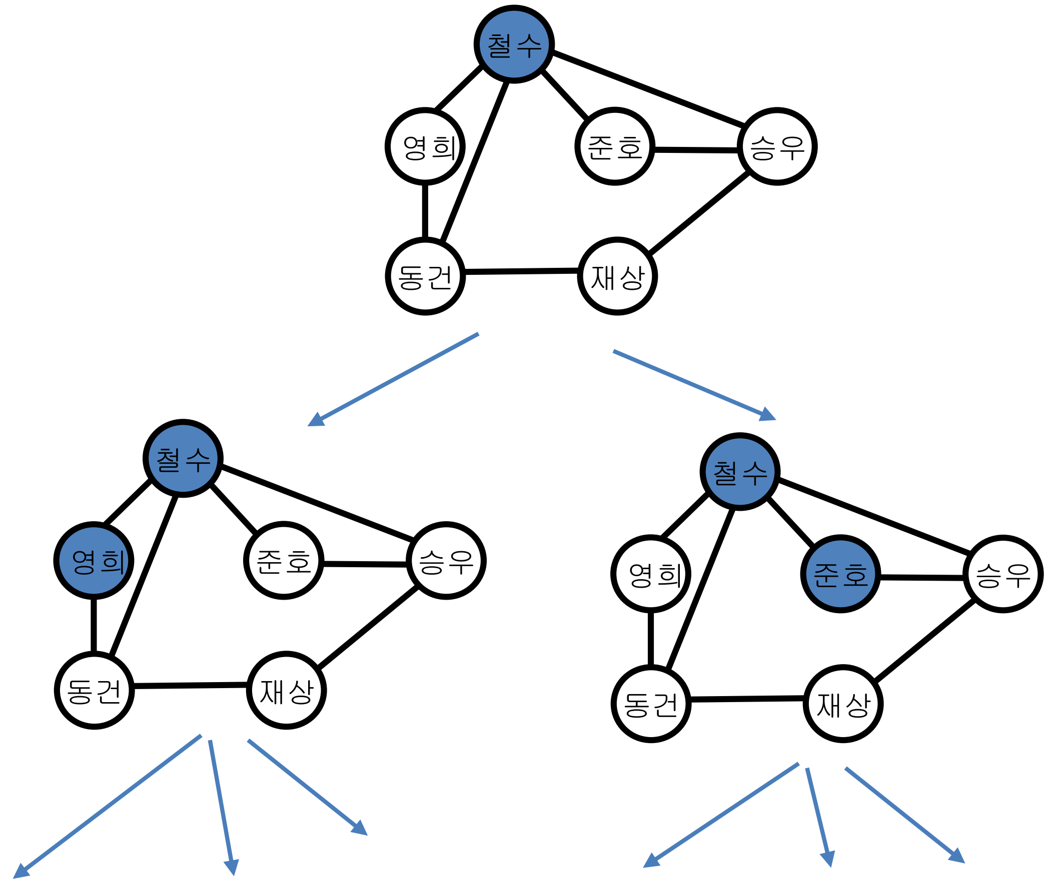
- **<문제>**
- **입력:**
  – 가중치 없는 무향 그래프 $G = (V, E)$
- **출력:**
  – 임의의 정점 s 에서 출발해서 모든 정점을 한 번씩 방문한다.
  – 방문한 순서대로 출력

비교) 문제 공간 == 경우의 수

입력: {'a', 'b', 'c'}
<문제 공간>



입력: G = (V, E)
<문제 공간>



그래프 알고리즘의 기본

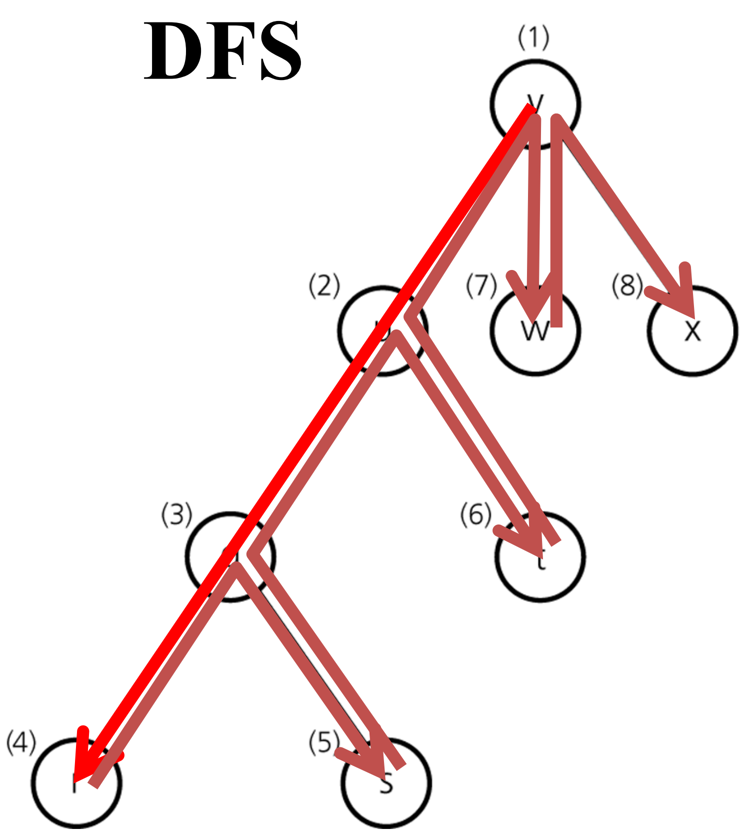
–깊이우선탐색(DFS)

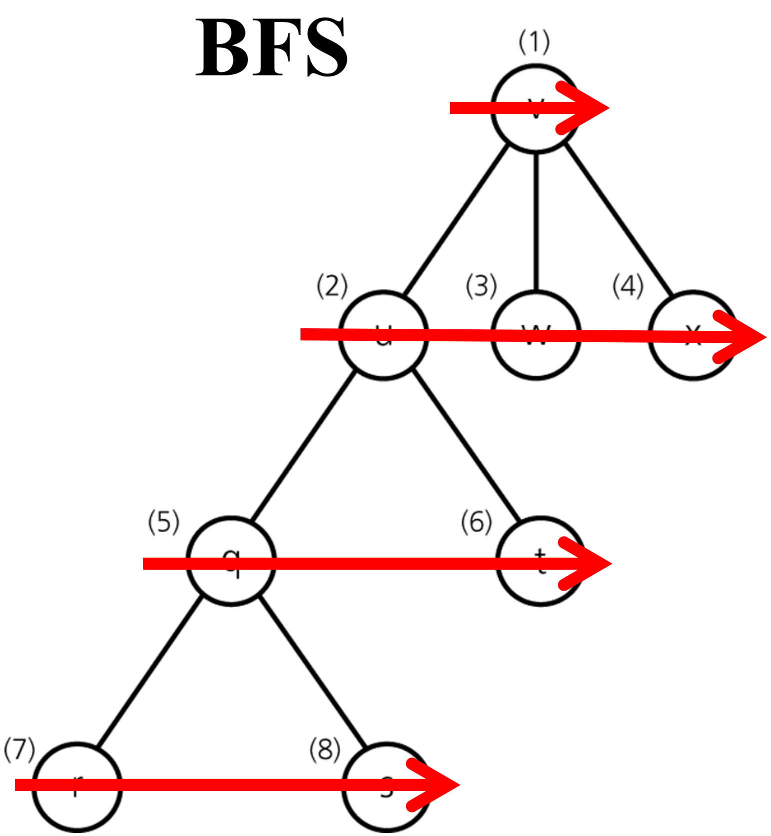
- Depth-First Search

–너비우선탐색(BFS)

- Breadth-First Search

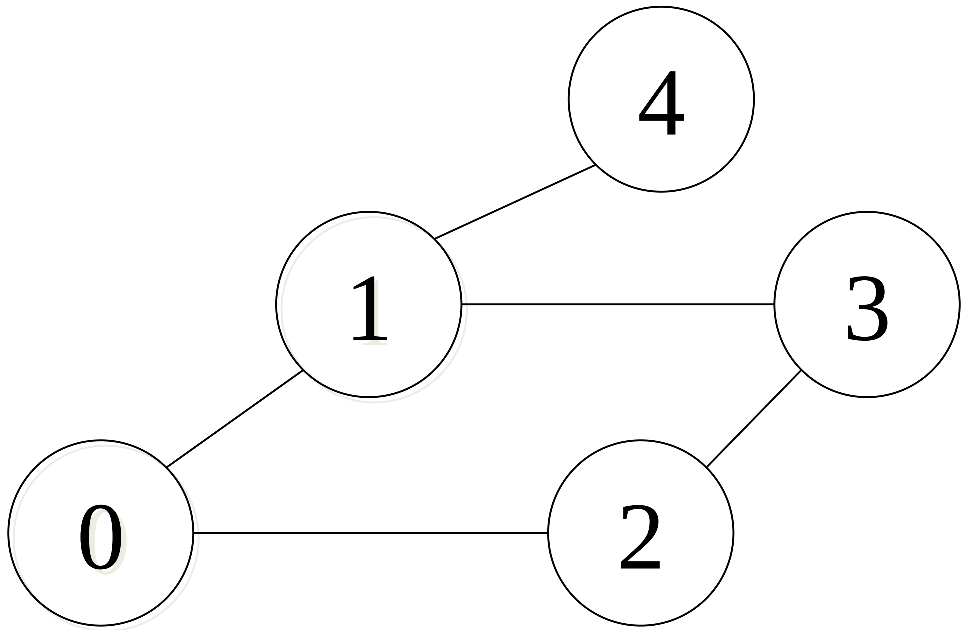
동일한 트리를 각각 DFS/BFS로 방문하기





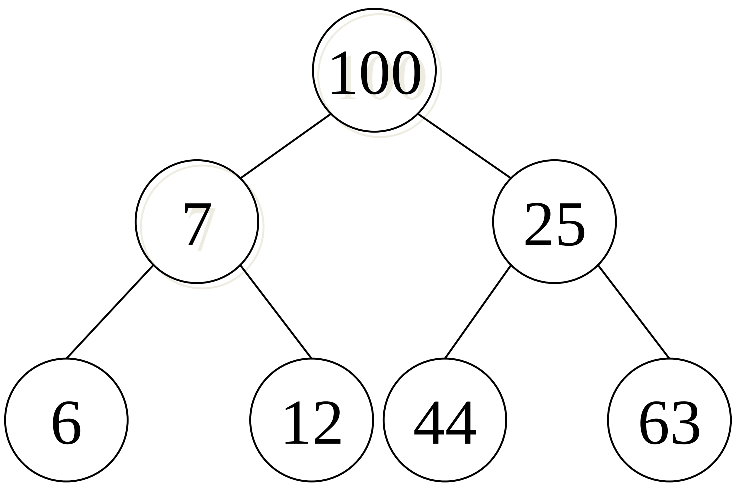
연습문제

- 아래 그래프를 0에서 시작해서 DFS, BFS로 방문하는 순서를 적어 보자.



트리에서 DFS

- **이진트리에서 DFS: 전위/중위/후위 순회**
  - 전위 순회한 경우

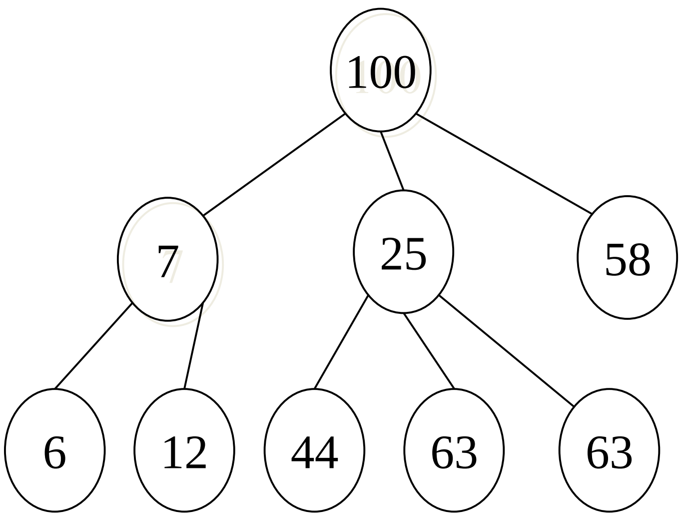



```
binaryTreeDfs(t):
        if not t: return

        print(t.key)
        binaryTreeDfs(t.left)
        binaryTreeDfs(t.right)
```

100 7 6 12 25 44 63

트리에서 DFS

- **다진 트리(N-ary Tree)에서 DFS:**
  - 자식들을 차례로 방문



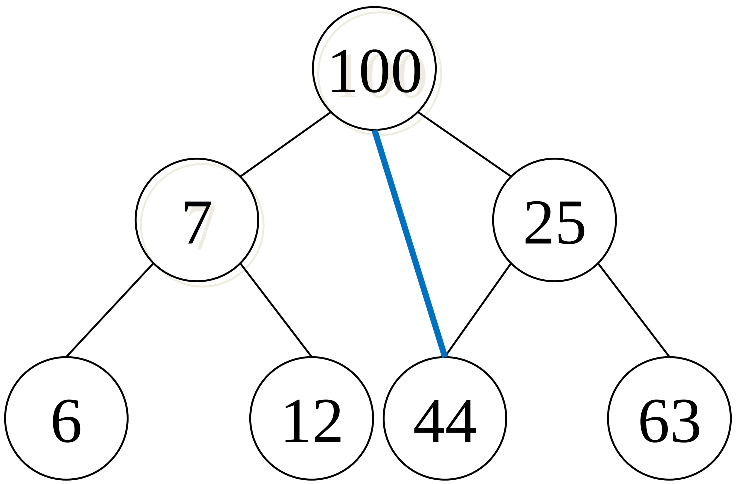
이 경우 경계조건은 필요 없음

```
treeDfs(t):
        print(t.key)
        for child in t.children:
                treeDfs(child)
```

?????

싸이클이 있는 그래프에서 DFS

- 싸이클이 있는 그래프를 트리와 같은 알고리즘으로 방문하면 어떤 문제가 생길까?



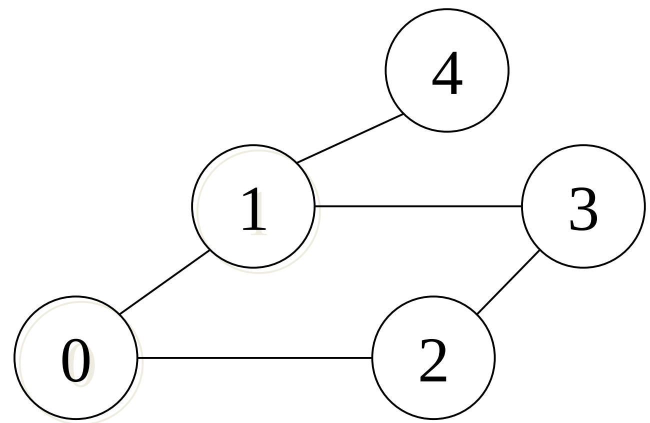

```
treeDfs(t):
        print(t.key)
        for child in t.children:
                treeDfs(child)
```

싸이클이 있는 그래프에서 DFS

- **아래 프로그램을 실행해 보자.**
  - 주어진 그래프로 인접리스트를 만든다.

```
V = {0, 1, 2, 3, 4}
N = len(V)
E = {(0, 1), (0, 2), (1, 4), (2, 3),
(1, 4)}
G = (V, E)

L = {v:[] for v in V}
for x, y in E:
    L[x].append(y)
    L[y].append(x)

for v in V:
    print(v, '->’, L[v])
```

L[x]: x와 이웃link한
정점들의 리스트




```
0 -> [1, 2]
1 -> [0, 4]
2 -> [0, 3]
3 -> [2]
4 -> [1]
```

싸이클이 있는 그래프에서 DFS

- **앞의 프로그램에 추가하고 실행해 본다.**
  - treeDFS()와 같은 알고리즘

v와 연결된 정점들(L[v])을
차례로 방문(깊이우선탐색)한다.

<추가할 코드>

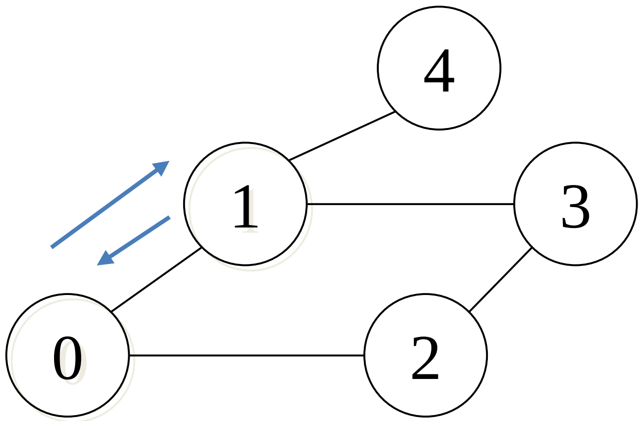
```
def dfs(v):
    print(v)
    for w in L[v]:
        dfs(w)



dfs(0)
```

<비교>

```
treeDfs(t):
        print(t.key)
        for child in t.children:
                treeDfs(child)
```

싸이클이 있는 그래프에서 DFS

- 0, 1이 무한 반복된다!




```
1
0
1
0
1
0
1
```

```
  [Previous line repeated 993 more times]
  File "C:\Users\jhjang\Google 드라이브\code\testproject\main.py", line 16, in dfs
    print(v)
RecursionError: maximum recursion depth exceeded while calling a Python object
```

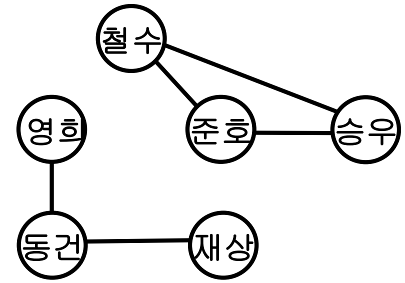
싸이클이 있는 그래프에서 DFS

- 한번 방문한 정점을 기억할 필요가 있음

```
def dfs(v):
    print(v)
    for w in L[v]:
        dfs(w, L)


dfs(0)
```

```
def dfs(v):
    visited.add(v)
    print(v)
    for w in L[v]:
        if w not in visited:
            dfs(w)


visited = set()
dfs(0)
```

DFS 깊이우선탐색



DFS(*G*) ← disconnected graph인 경우 필요

{

  for each $v \in V$

    visited[v] ← NO;

  for each $v \in V$

    if (visited[v] = NO) **then** aDFS(v);

}

aDFS (*v*)

{

  visited[v] ← YES;

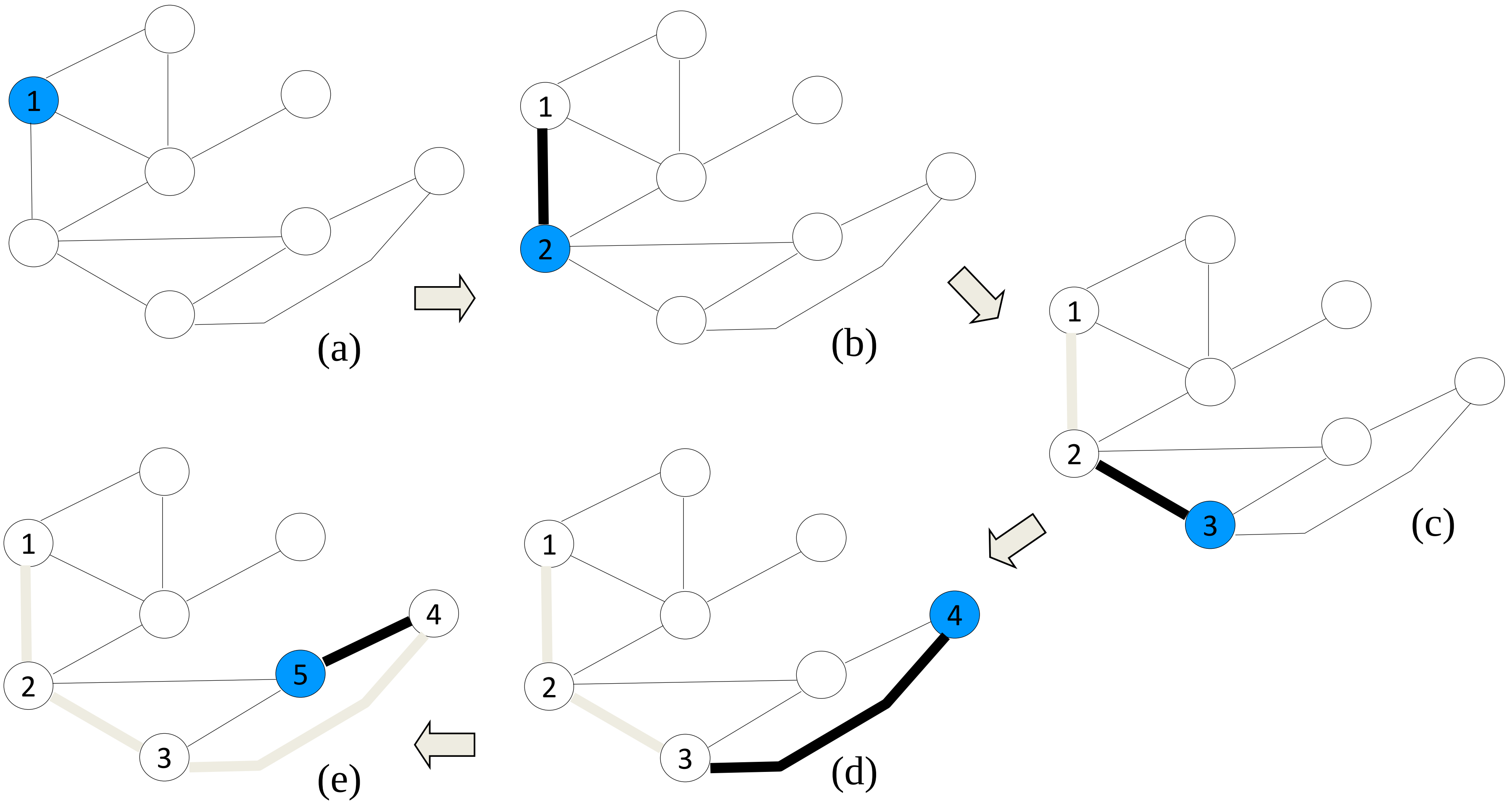
  for each $x \in L(v)$ ▷ $L(v)$: 정점 v의 인접 리스트

    if (visited[x] = NO) **then** aDFS(x);

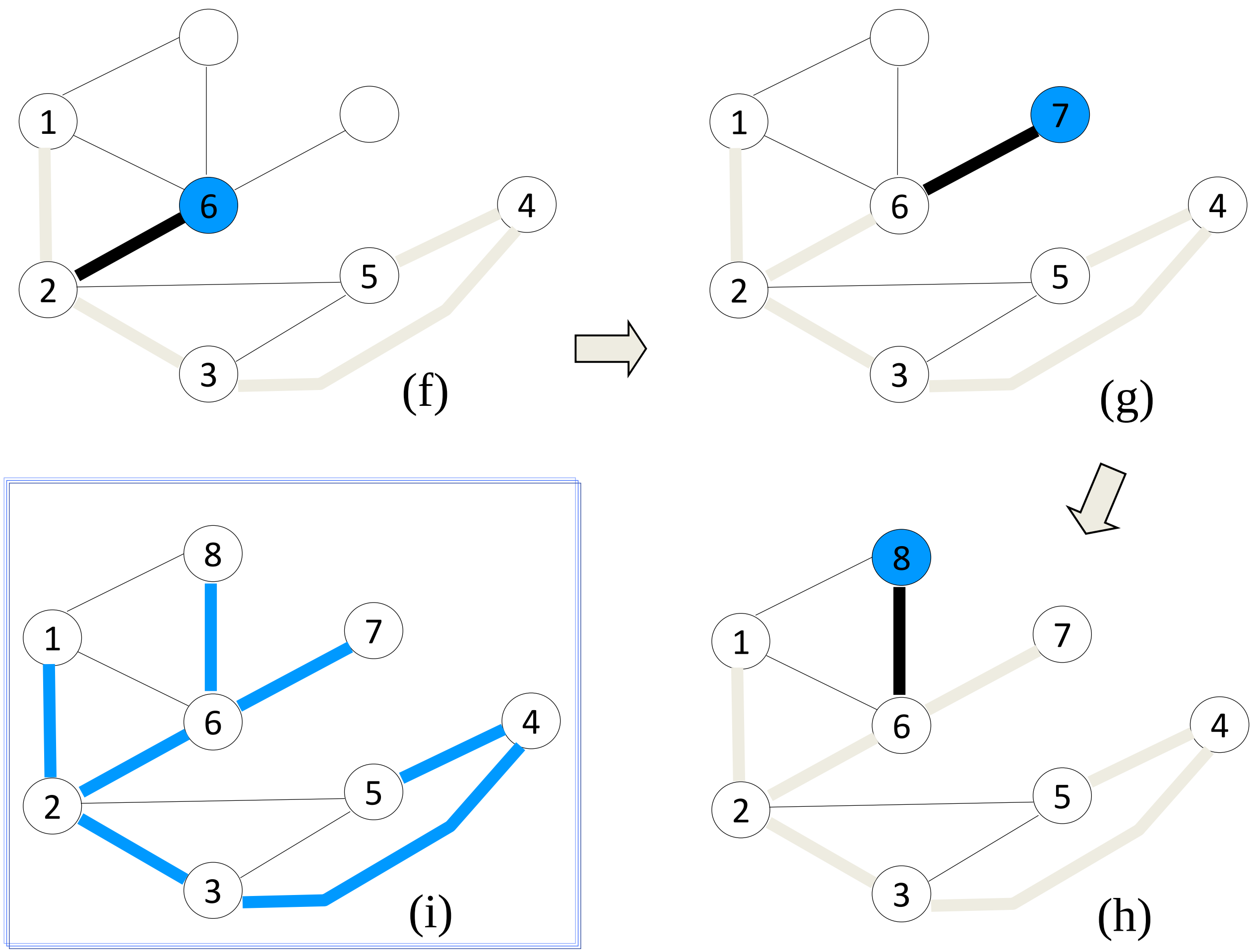
}

✓수행 시간: $\Theta(|V|+|E|)$

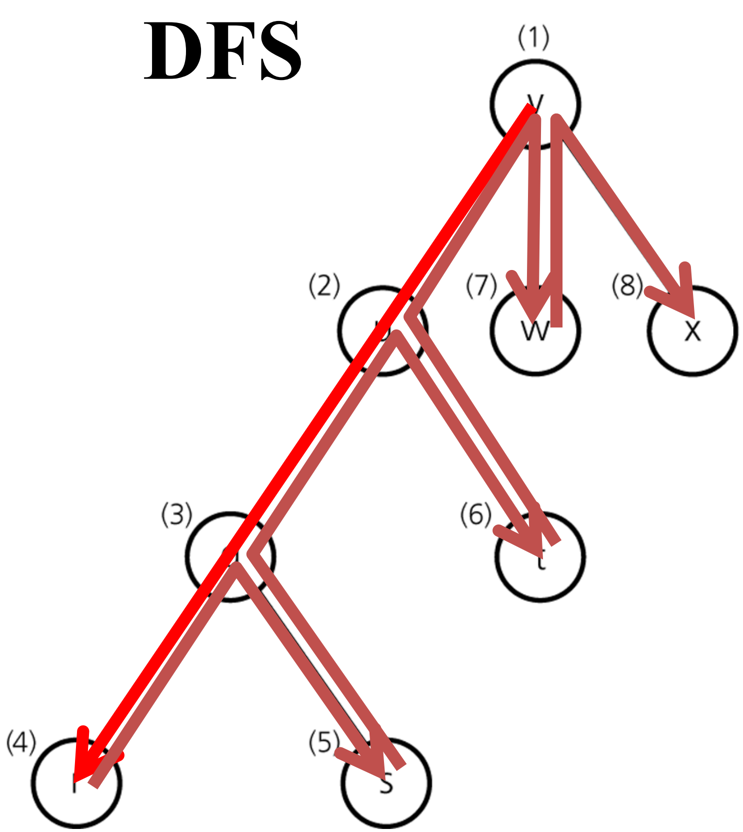
DFS의 작동 예

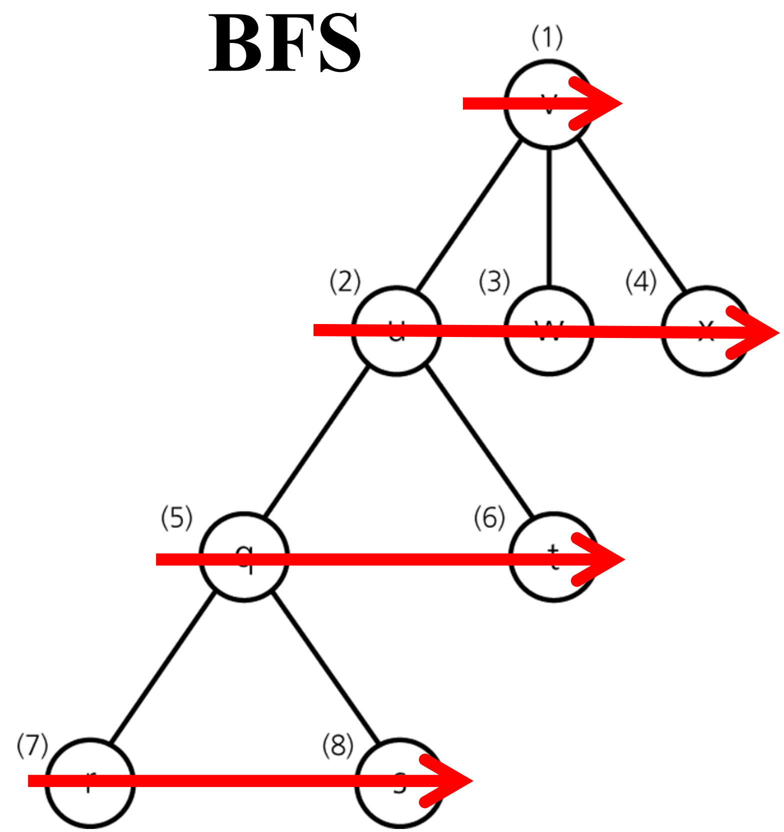


DFS의 작동 예 (계속)



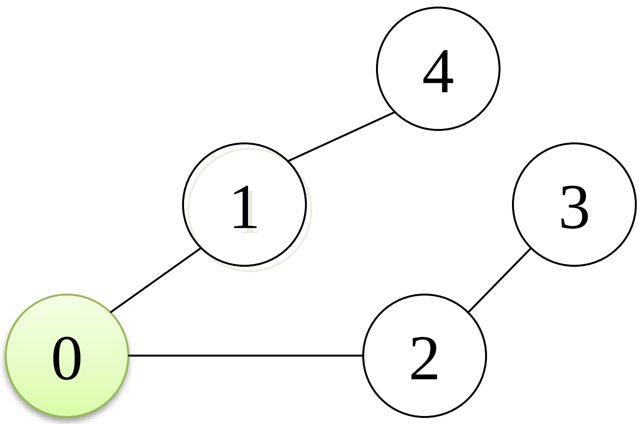
동일한 트리를 각각 DFS/BFS로 방문하기



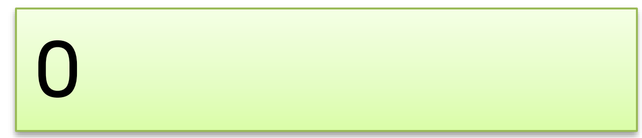


BFS

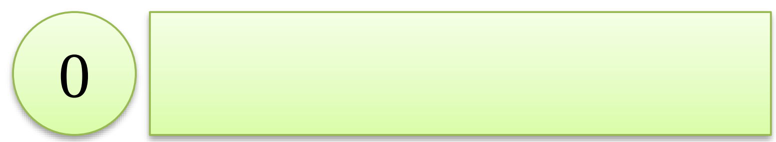
- **큐(Queue): 대기열 == 앞으로 방문할 순서**

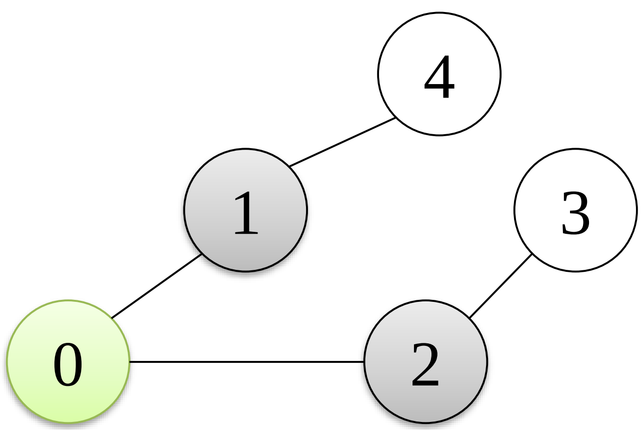


큐에 시작 정점을 넣고 시작

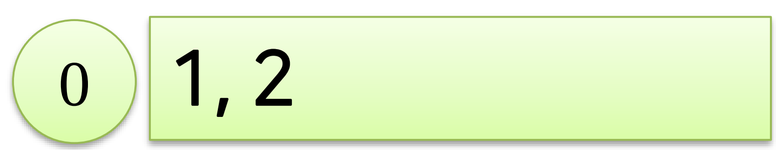


0을 꺼낸다(0을 방문한다)



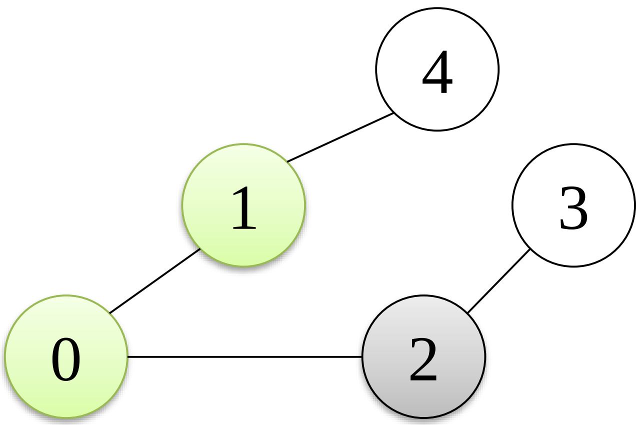


0의 자식 중 **방문하지 않은** 정점들을 큐에 넣는다.

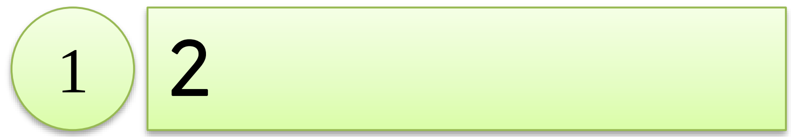


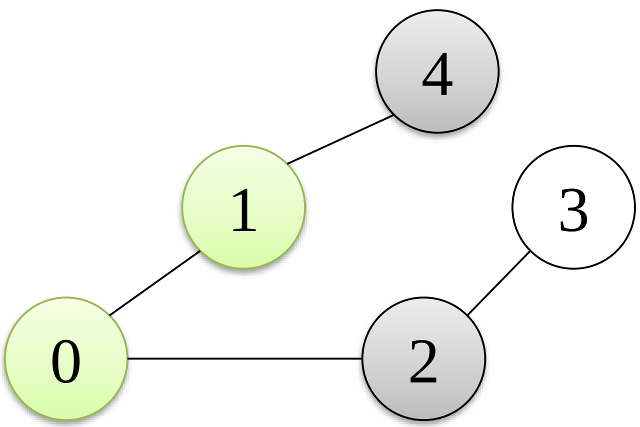
BFS

- **Queue: 대기열 == 앞으로 방문할 순서**

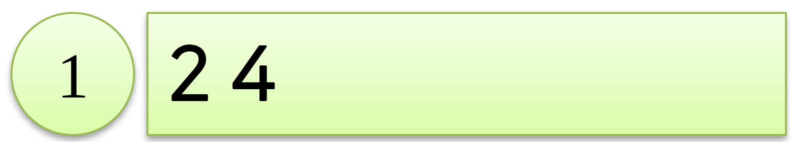


큐에서 첫 번째 원소를 꺼낸다(방문한다).



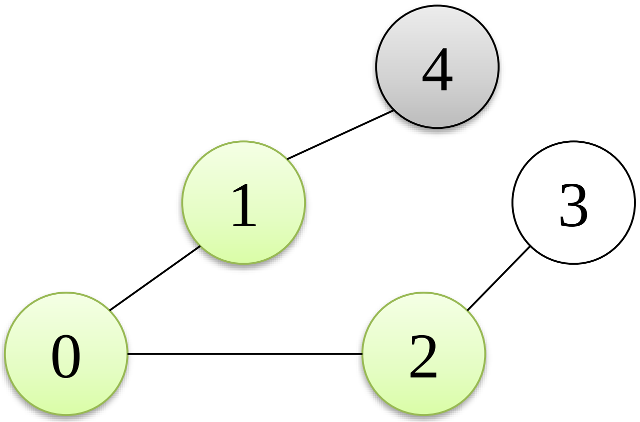


1의 자식 중 방문하지 않은 정점들을 큐에 넣는다.

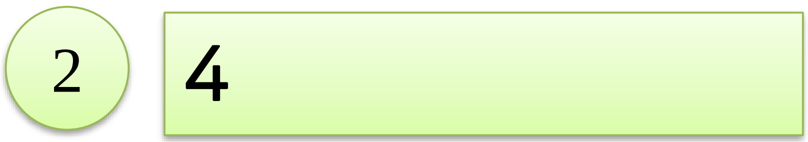


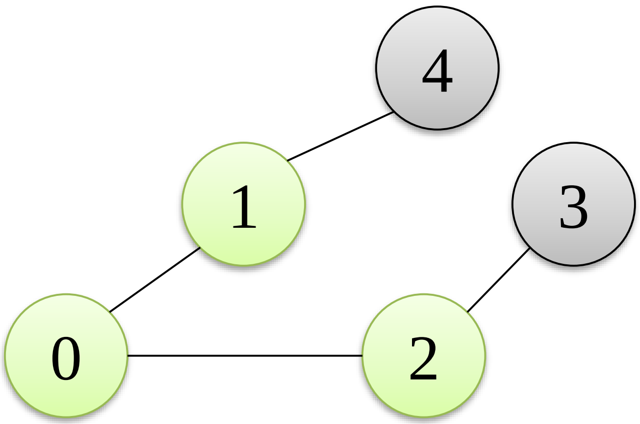
BFS

- **Queue: 대기열 == 앞으로 방문할 순서**

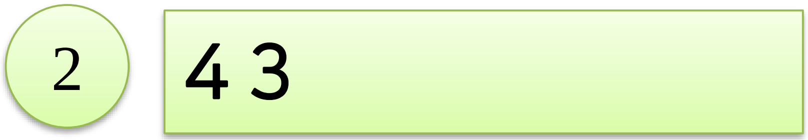


큐에서 첫 번째 원소를 꺼낸다(방문한다).



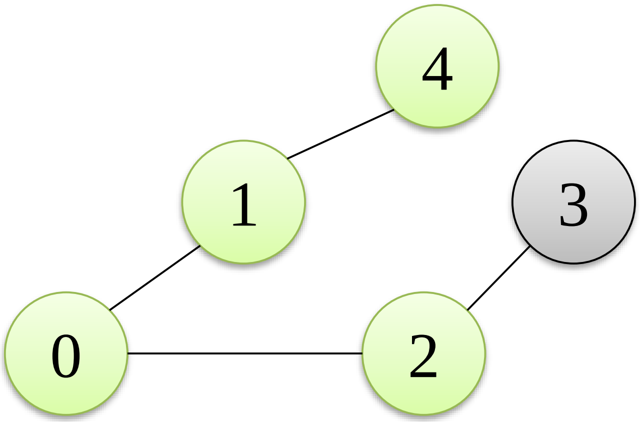


2의 자식 중 방문하지 않은 정점들을 큐에 넣는다.

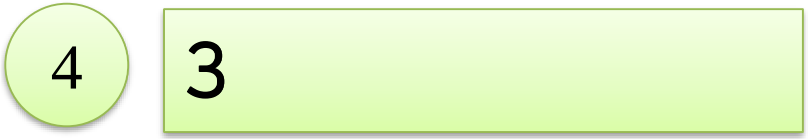


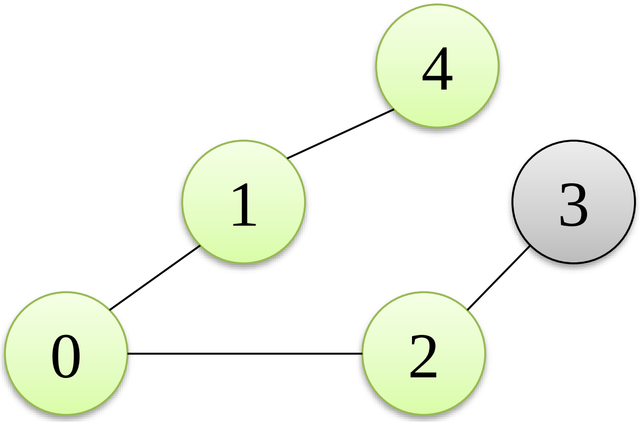
BFS

- **Queue: 대기열 == 앞으로 방문할 순서**

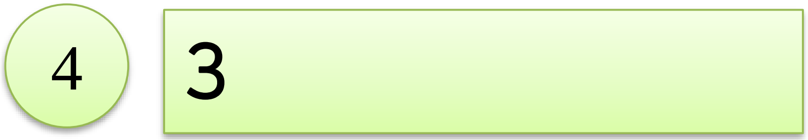


큐에서 첫 번째 원소를 꺼낸다(방문한다).



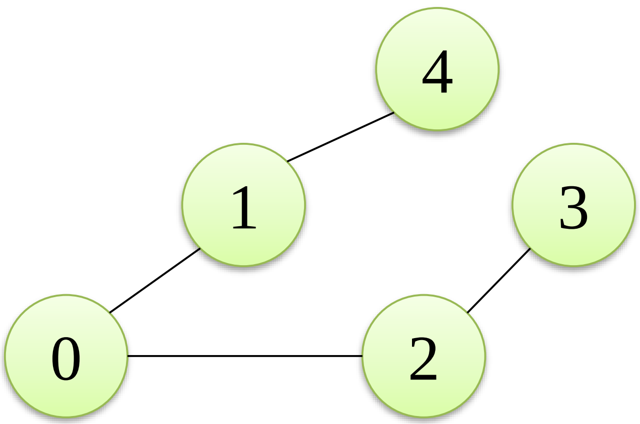


4의 자식 중 방문하지 않은 정점들을 큐에 넣는다.

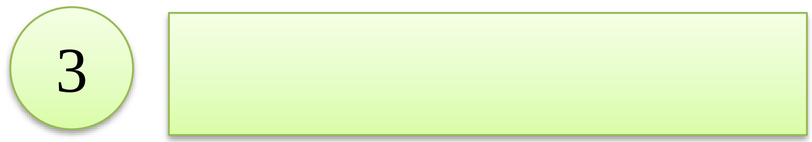


BFS

- **Queue: 대기열 == 앞으로 방문할 순서**



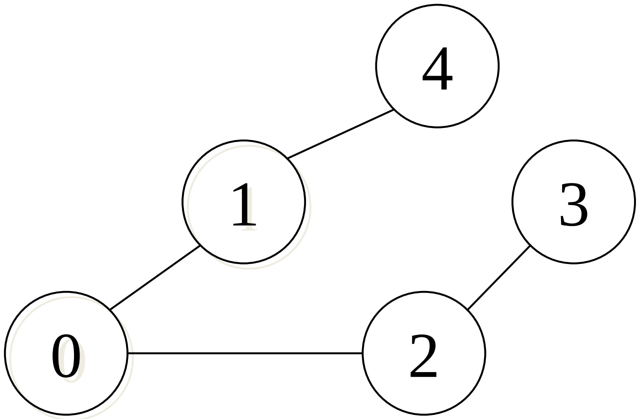
큐에서 첫 번째 원소를 꺼낸다(방문한다).

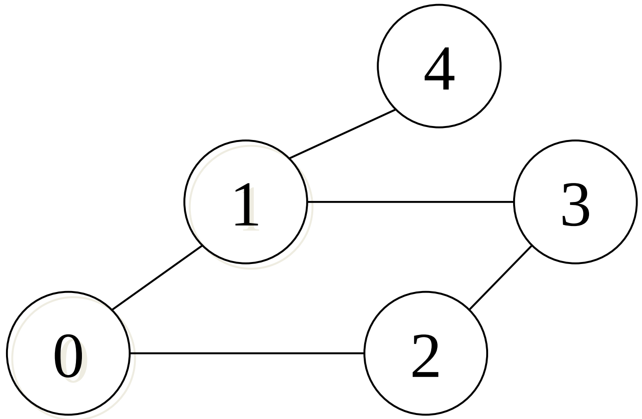


큐가 비었으면 BFS를 종료한다.

BFS

- **트리가 아닌 그래프에서도 같은 알고리즘을 적용**
  - “자식” 대신 “이웃한 정점”을 방문





BFS

```
def bfs(v):
    Q = [v]
    while Q:
        v = Q.pop(0)
        visited.add(v)
        print(v)

        for w in L[v]:
            if w not in visited:
                Q.append(w)


visited = set()
bfs(0)
```

- 시작 정점을 큐에 넣고 시작
- 큐가 빌 때까지 반복한다.

큐의 첫 번째 원소를 꺼내서 방문

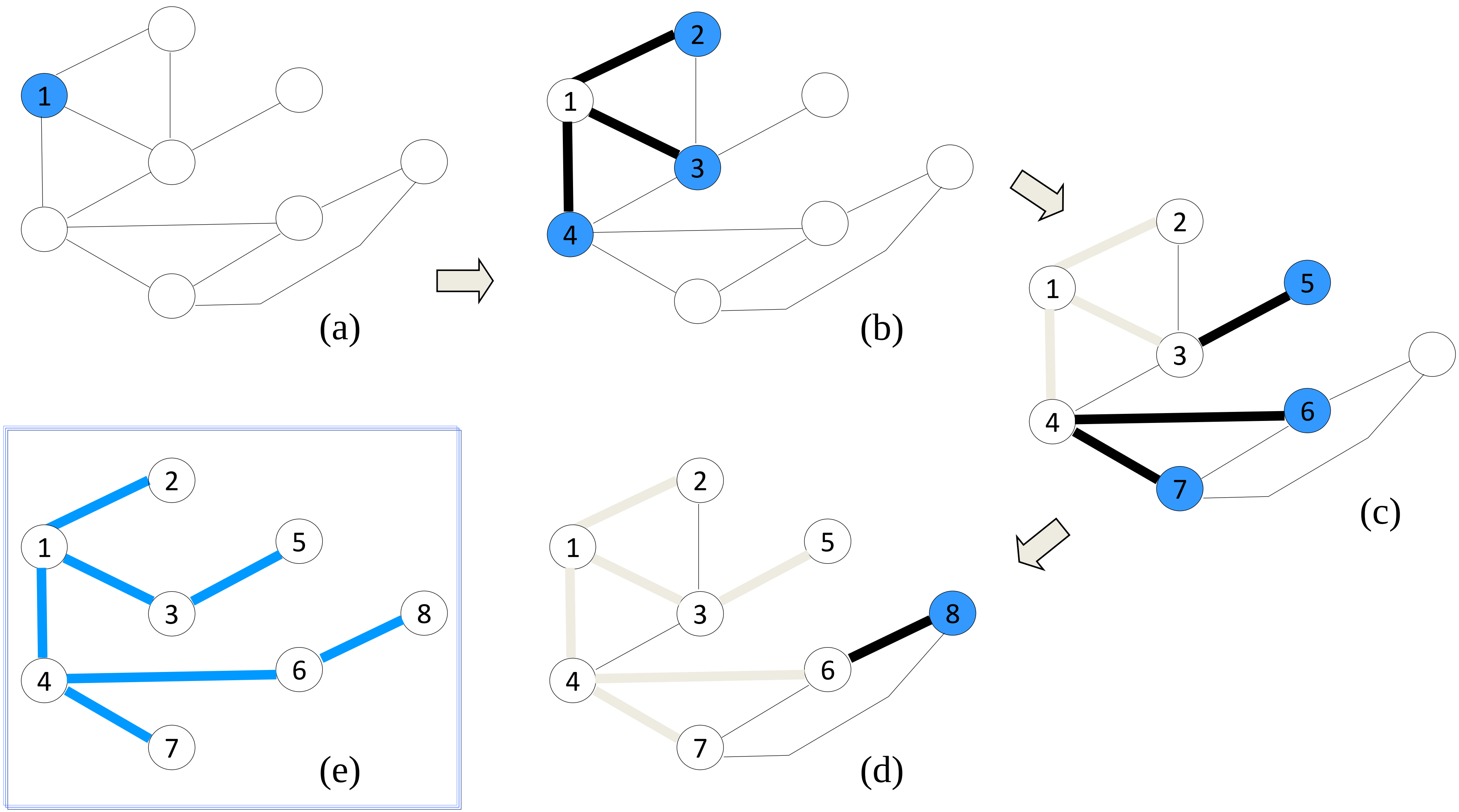
v의 이웃 중, 아직 방문하지 않은 정점들을 대기열에 넣는다.

BFS너비우선탐색

```
BFS(G, v)
{
        for each v ∈ V−{s}
                visited[v] ← NO;
        visited[s] ← YES;                ▷ s: 시작 정점
        enqueue(Q, s);                   ▷ Q: 큐
        while (Q ≠ φ) {
                u ← dequeue(Q);
                for each v ∈ L(u)                ▷ L(u): 정점 u의 인접 리스트
                        if (visited[v] = NO) then
                                visited[u] ← YES;
                                enqueue(Q, v);
                        }

        }
}
```

✓수행 시간: $\Theta(|V|+|E|)$

BFS의 작동 예



연습문제

- dfs(), bfs() 함수를 지운 후 다시 작성해 보자.