DataStructure Seminar Chapter.11

GRAPH

WISOFT 박 대 원

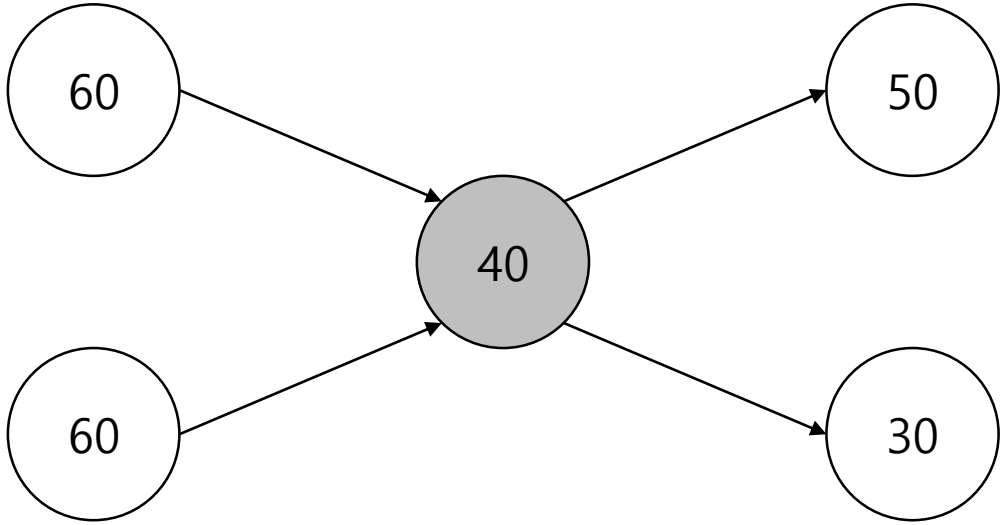
| 그래프(Graph)

노드(node)와 간선(edge)의 집합

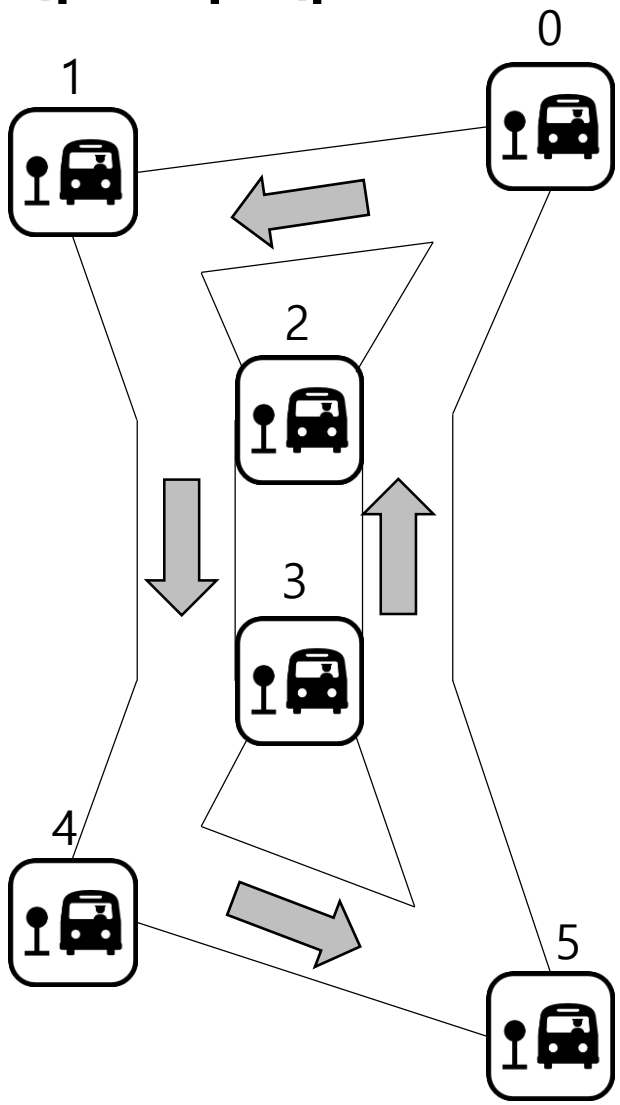
표현 능력이 강력한 비선형 자료구조

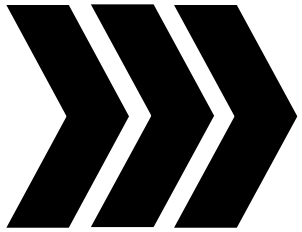
지금까지의 비선형 자료구조에서는 다음 노드가 최대 2개이고 이전 노드는 1개.

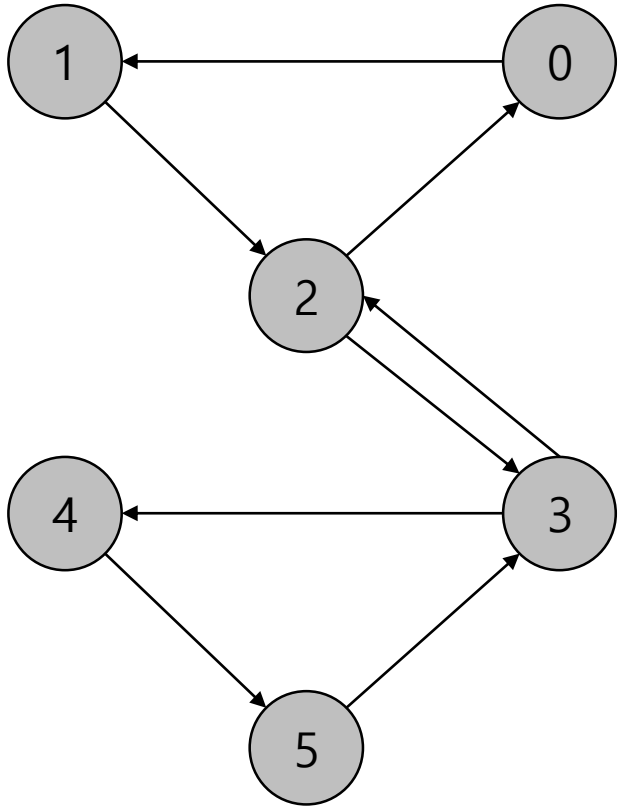
반면 그래프는 노드 사이의 관계에서 아무런 제약이 없다.



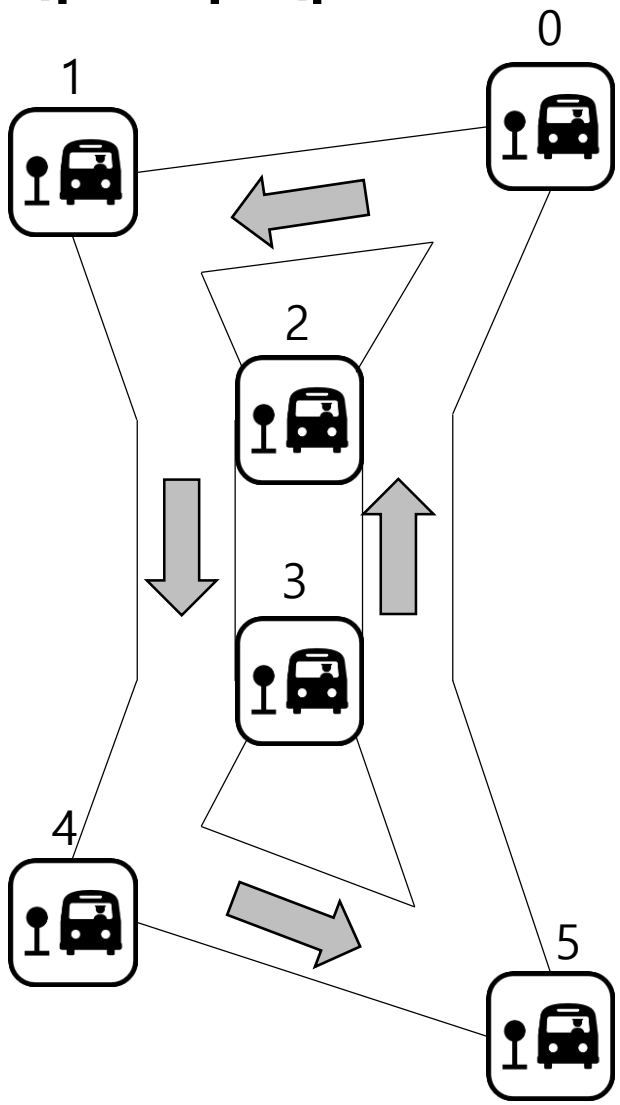
I 그래프의 예

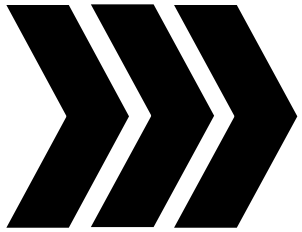


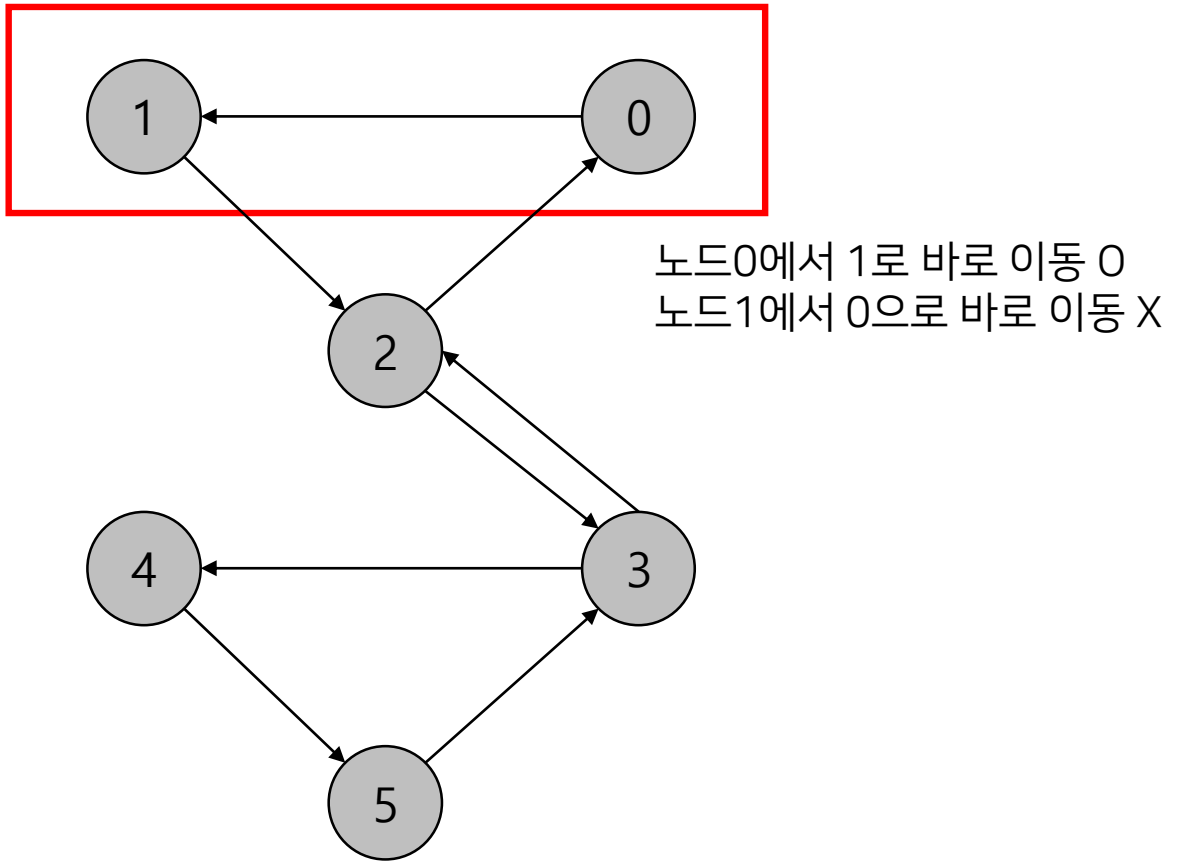




I 그래프의 예

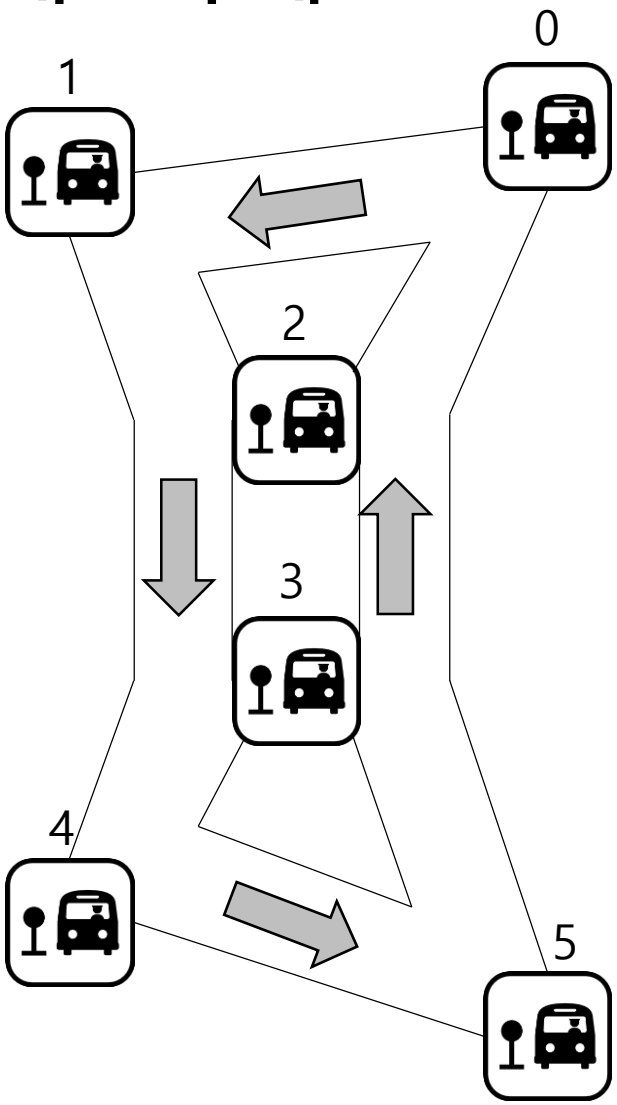


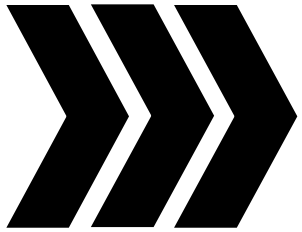


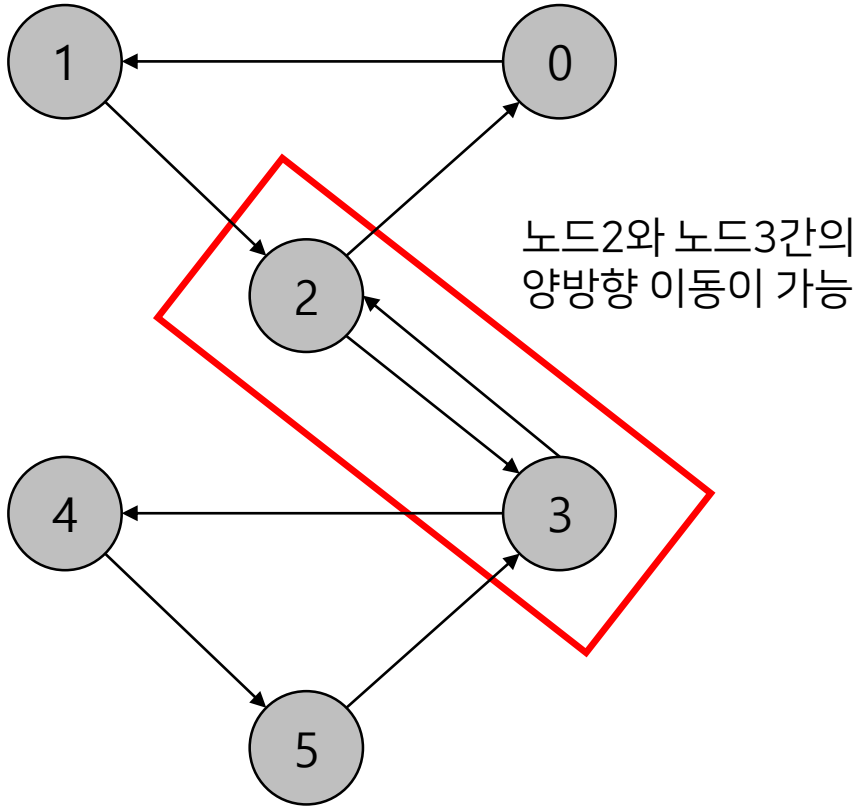


노드0에서 1로 바로 이동 O
노드1에서 0으로 바로 이동 X

I 그래프의 예







| 그래프의 수학적 정의

G = (V(G), E(G))

= 그래프 G는 노드들의 집합 V(G)와 간선들의 집합 E(G)로 구성된다.
= 그래프는 노드와 간선의 집합이다.

트리에서의 간선이 주로 노드사이의 연결 여부만을 저장했다면,

그래프에서는 연결의 방향성이나 가중치 값 등 추가적인 정보도 포함한다.

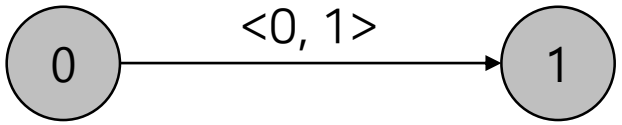
그래프가 가지는 강력한 점은 이처럼 간선에 저장되는 정보가 풍부하다는 점이다.

| 간선의 특성에 따른 그래프의 종류

1. 방향 그래프(directed graph)

두 노드를 연결하는 간선에 방향이 있는 그래프.
다이그래프(digraph)라고도 함.

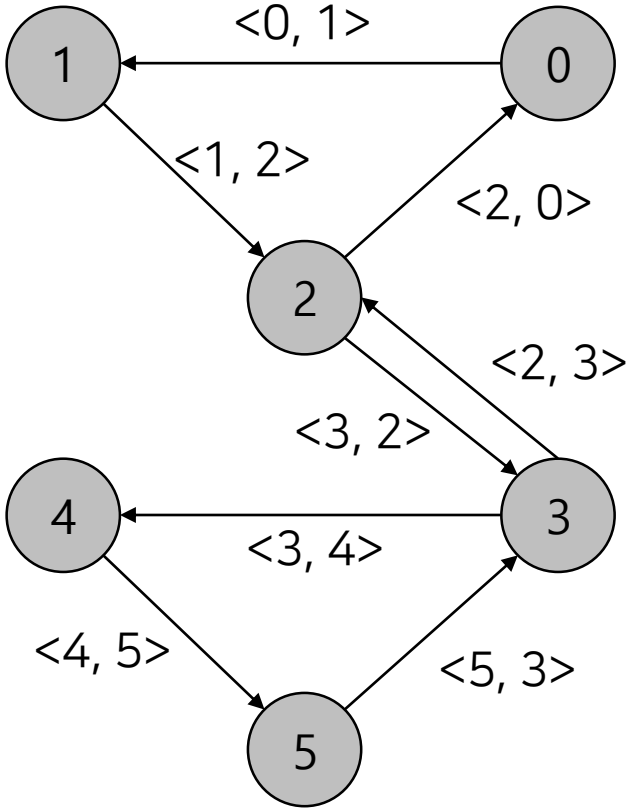
방향 그래프의 간선의 표현



꼬리(Tail) 머리(Head)

꼬리(Tail) : 방향 그래프의 간선에서 시작 노드.
머리(head) : 간선의 종료 노드.

G1의 노드: V(G1) = {0, 1, 2, 3, 4, 5}
G1의 간선: E(G1) = {<0, 1>, <1, 2>, <2, 0>, <2, 3> ,<3, 2>, <3, 4>, <4, 5>, <5, 3>}

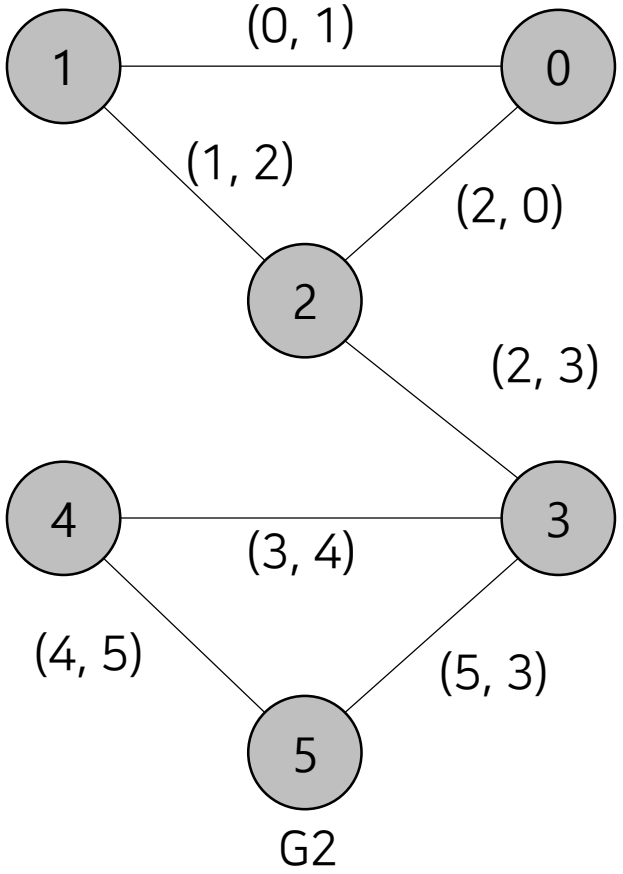


G1

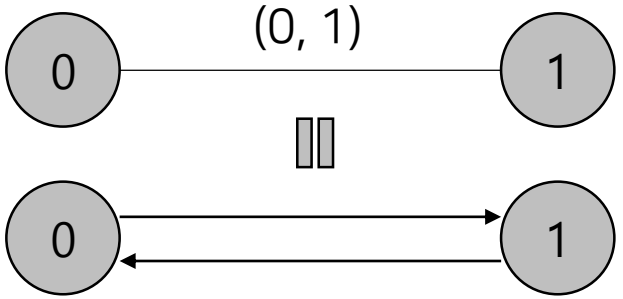
2. 무방향 그래프(undirected graph)

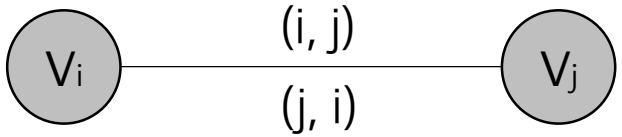
두 노드를 연결하는 간선에 방향이 없는 그래프.
연결 관계가 대칭적이다.

G2의 노드: V(G2) = {0, 1, 2, 3, 4, 5}
G2의 간선: E(G2) ={(0, 1), (1, 2), (2, 0), (2, 3), (3, 4), (4, 5), (5, 3)}



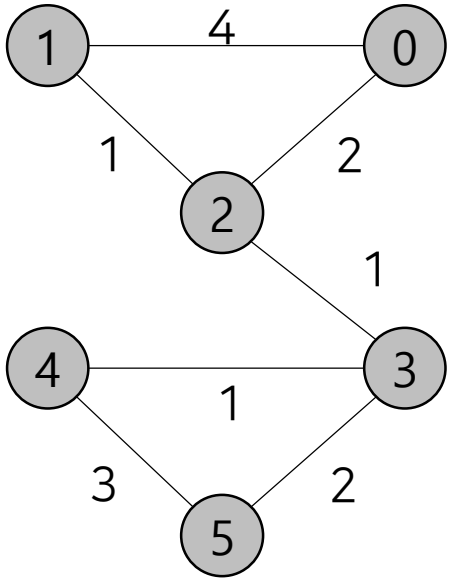
무방향 그래프의 간선의 표현



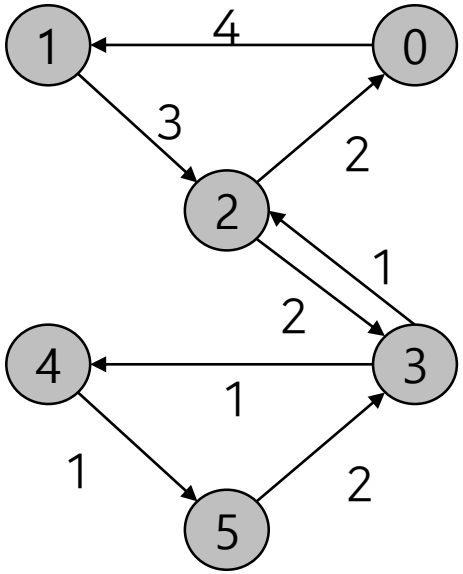


3. 가중 그래프(weighted graph)

노드를 연결하는 간선에 가중치(weight)가 있는 그래프
가중치는 간선 사이의 비용(cost), 거리(distance) 등이 될 수 있다.
무방향 그래프에 가중치가 존재할 때 무방향 가중 그래프(undirected weighted graph),
방향 그래프에 가중치가 존재할 때 방향 가중 그래프(directed weighted graph)라고 한다.
방향 가중 그래프를 네트워크(network)라고도 부른다.



G3



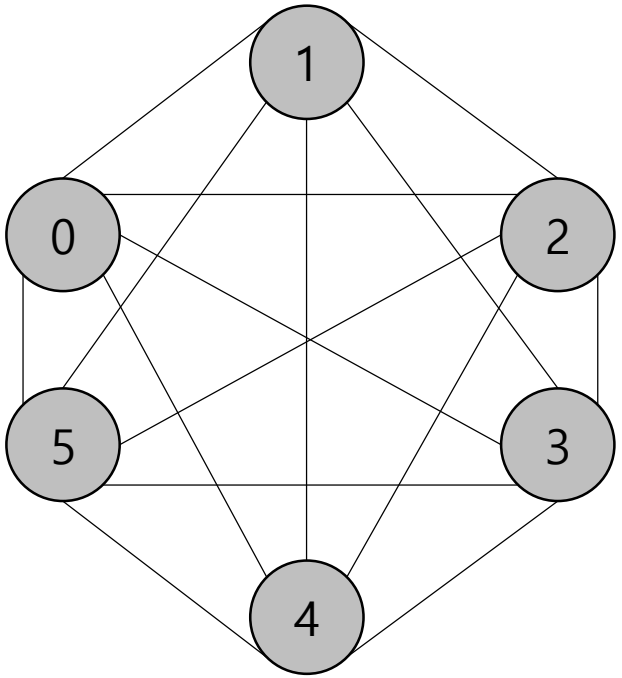
G4

| 구조적 특성에 따른 그래프의 종류

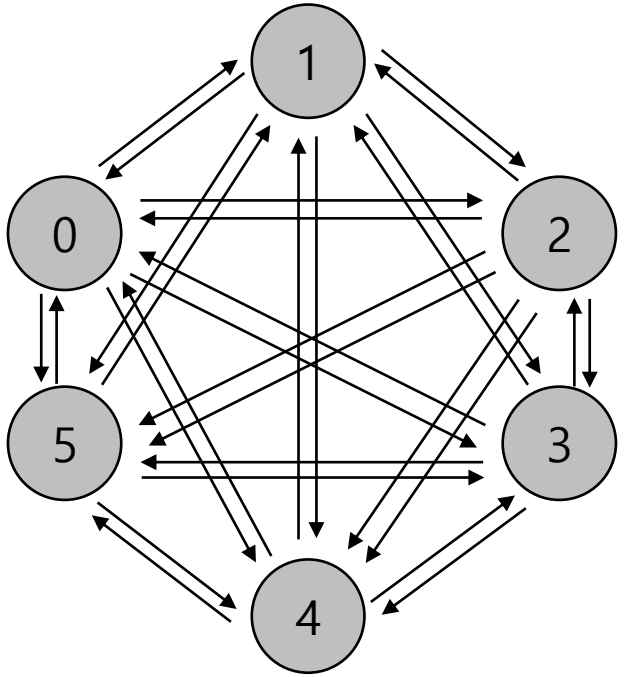
1. 완전 그래프(compete graph)

그래프 내의 모든 노드가 간선으로 연결된 그래프 = **연결 가능한 최대 간선 수를 가진 그래프**

무방향 그래프의 간선 수	*m = n(n-1)/2*
방향 그래프의 간선 수	*m = n(n-1)*



G6



G7

2. 부분 그래프(sub graph)

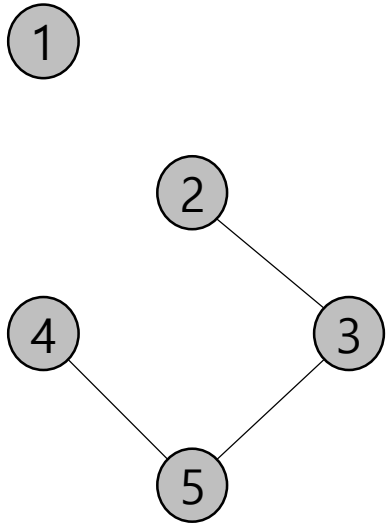
원래의 그래프에서 일부의 노드나 간선을 제외하여 만든 그래프

G = (V(G), E(G))

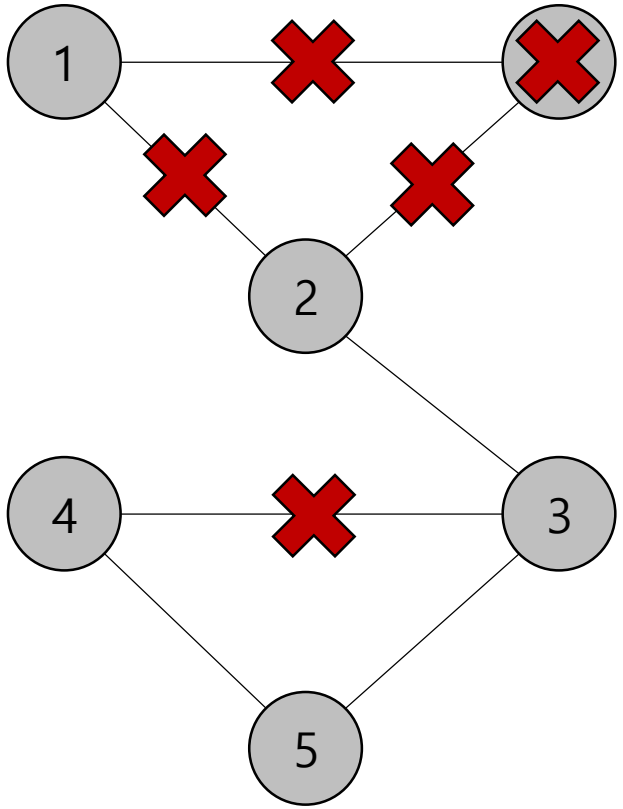
G′ = (V(G′), E(G′))

V(G′) ⊆ V(G)

E(G′) ⊆ E(G)



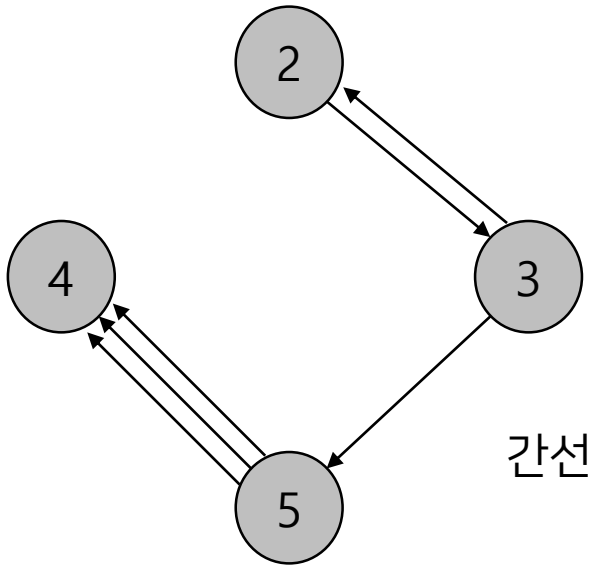
G8



G2

2. 다중 그래프(multi graph)

중복된 간선을 포함하는 그래프
무방향 그래프라 할지라도 2개 이상, 여러개의 간선이 올 수 있다.



G10

간선 <5, 4>가 3개 있으므로 같은 간선이 여러 개 있다는 점에서 다중그래프.

| 그래프 관련 용어

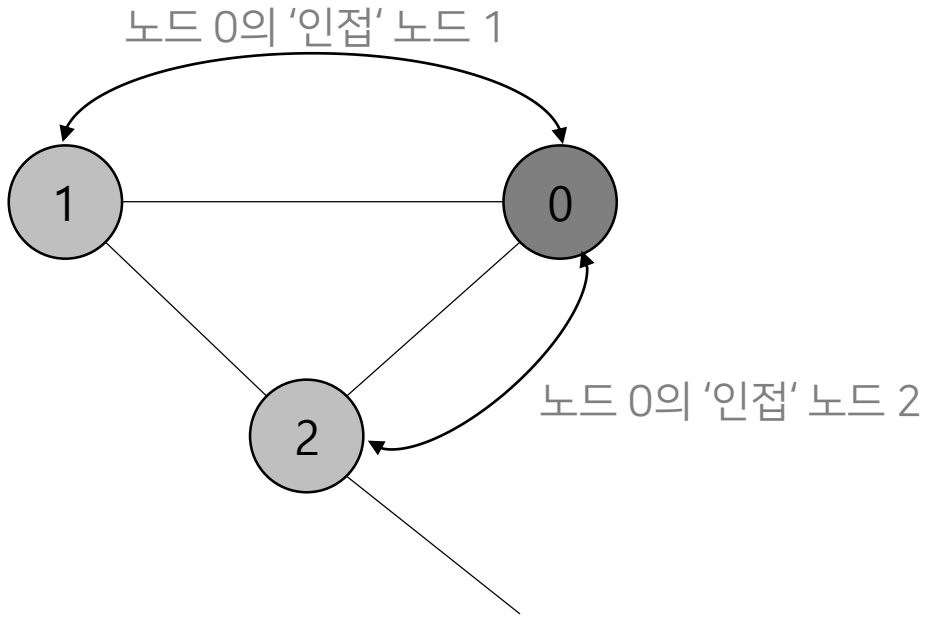
1.인접(adjacent)

연결과 같은 의미로 사용된다.

노드 0과 노드 1은 간선 (0, 1)로 연결 되어있고,

노드 0과 노드 2도 간선 0, 2)로 연결 되어있다.

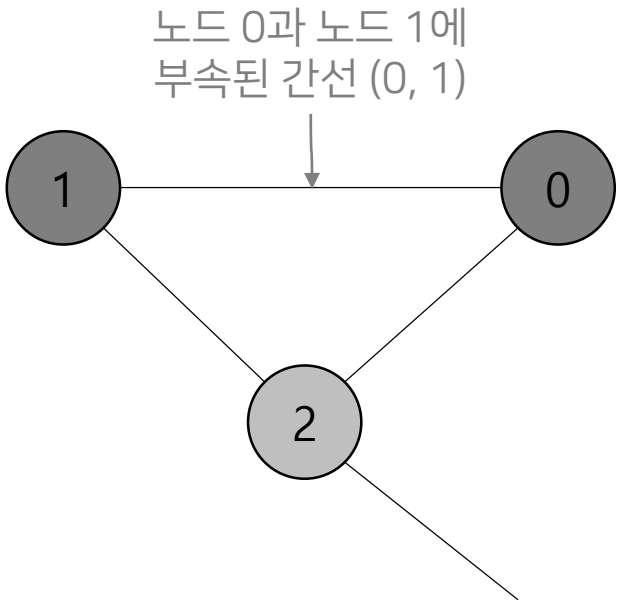
즉 노드 0의 인접 노드는 노드 1과 2이다.



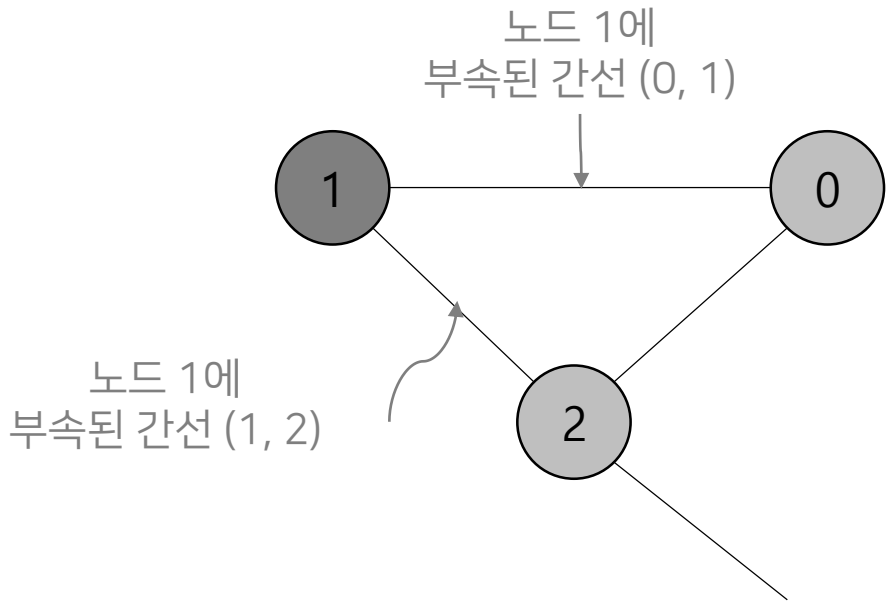
I 그래프 관련 용어

2. 부속(incident)

두개의 노드를 연결하는 간선이 존재하는 경우 이 간선은 두 노드에 각각 부속되었다고 한다.



간선 관점

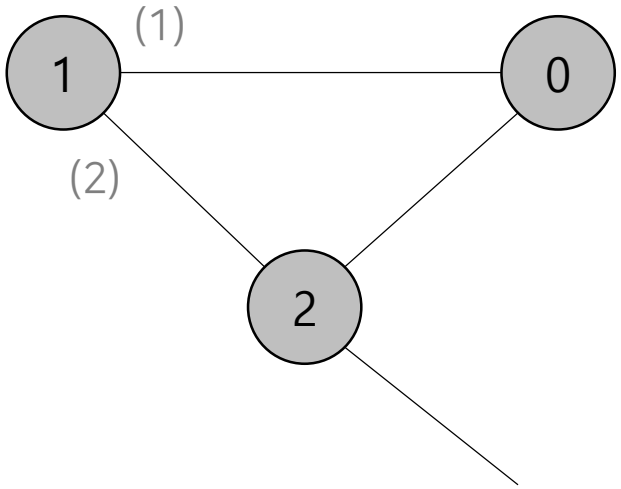


노드 관점

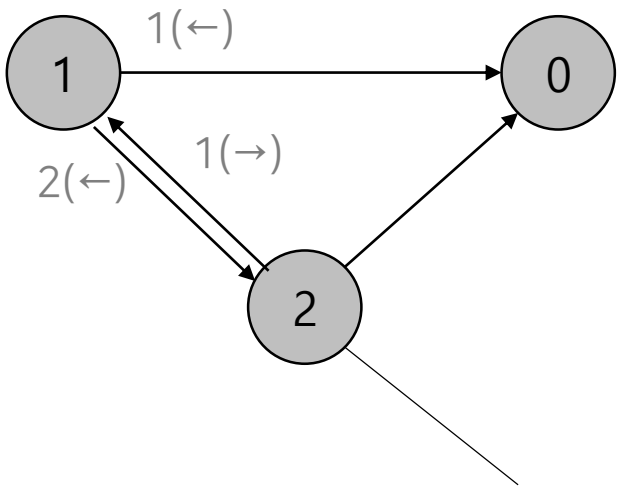
| 그래프 관련 용어

3. 차수(degree)

노드에 부속된 간선의 개수를 차수라고 한다.



무방향 그래프
노드 1의 차수 = 2

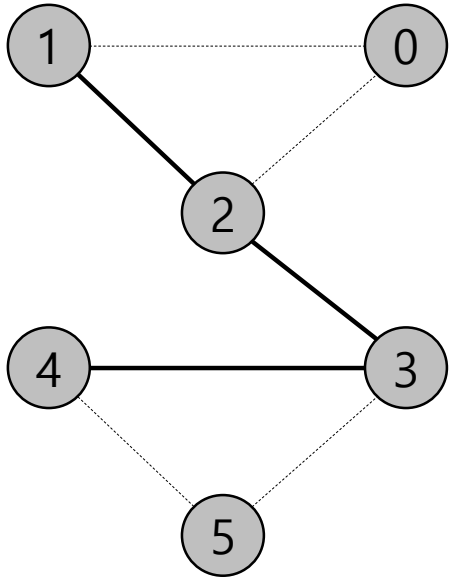


방향 그래프
노드 1의 진입차수 = 1
노드 1의 진출차수 = 2

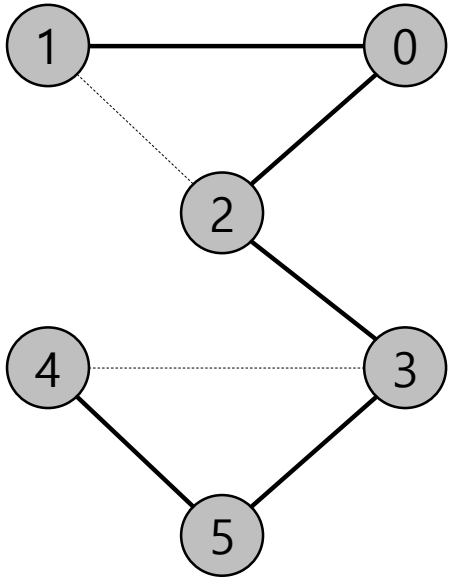
방향 그래프의 경우
간선의 방향성이 있기 때문에
노드로 들어오는 간선의 개수를
진입차수(in-degree)
노드에서 나가는 간선의 개수를
진출차수(out-degree)
라고 한다

| 그래프 관련 용어

4. 경로(path)
노드 A에서 노드 B까지 통과하는 길.
정확하게는 **노드 A부터 노드 B에 이르는 간선들의 인접 노드를 순서대로 나열한 목록**을 말한다.



① 1-2-3-4



② 1-0-2-3-5-4

경로 길이 : 경로를 구성하는 간선의 개수.

단순 경로 : 경로에 노드들을 모두 1번씩만 사용한 경로
방향 그래프에서는 단순 방향 경로.

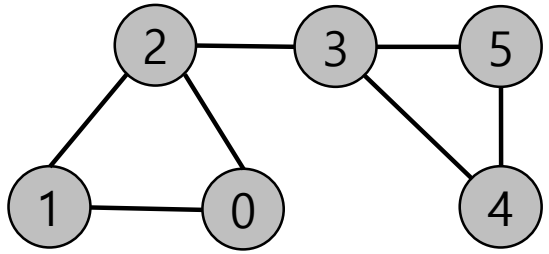
사이클 : 경로의 시작노드와 마지막 노드가 같은 경로.

연결되다 : 그래프 내의 모든 노드 사이에 경로가
있을 경우 연결되었다(connected)라고 함.

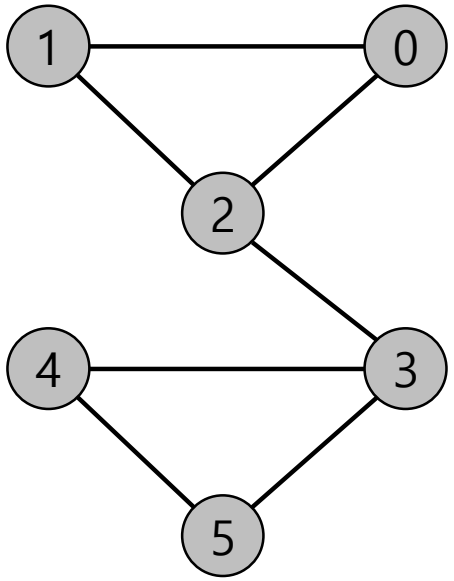
| 그래프 관련 용어

5. 그래프의 동일성

그래프에서 노드와 간선의 집합이 서로 같으면 같은 그래프이다.



G5



G1

G5의 노드: V(G5) = {0, 1, 2, 3, 4, 5}

G5의 간선: E(G5) ={(0, 1), (1, 2), (2, 0), (2, 3), (3, 4), (4, 5), (5, 3)}

G1의 노드: V(G1) = {0, 1, 2, 3, 4, 5}

= V(G5)

G1의 간선: E(G1) ={(0, 1), (1, 2), (2, 0), (2, 3), (3, 4), (4, 5), (5, 3)}

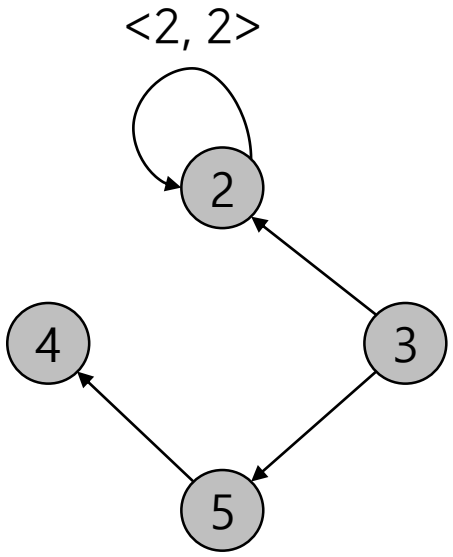
= E(G5)

= 서로 같은 그래프

| 그래프 관련 용어

6. 루프(loop)

그래프의 한 노드에서 자기 자신으로 이어지는 간선이 있다면 루프 라고 한다.



G9

| 그래프의 종류

구분	종류	설명
간선의 방향성	무방향 그래프	간선에 방향이 없는 그래프
	방향 그래프	간선에 방향이 있는 그래프
간선의 가중치	가중 그래프	간선에 가중치가 할당된 그래프
구조적 특징	완전 그래프	연결 가능한 최대 간선 수를 가진 그래프
	부분 그래프	원래의 그래프에서 일부의 노드나 간선을 제외하여 만든 그래프
	다중 그래프	중복된 간선을 포함하는 그래프

I 그래프의 추상 자료형

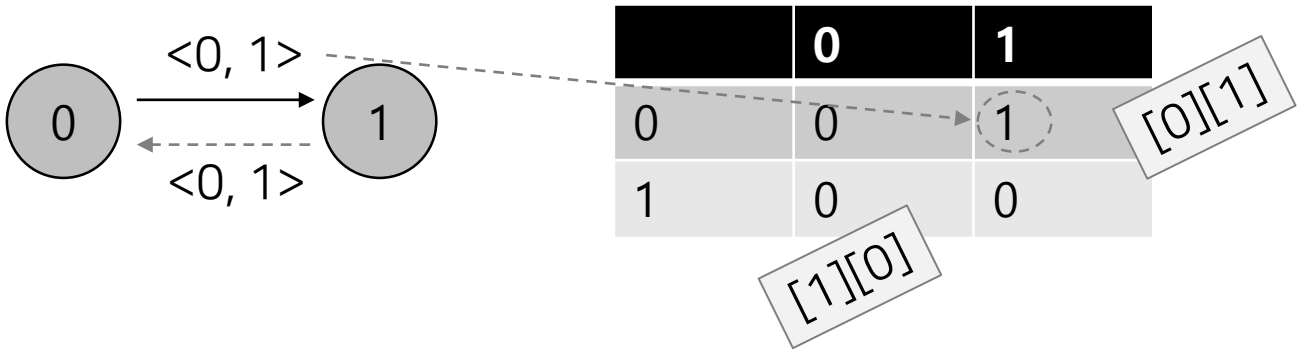
그래프의 구현에 필요한 기본 연산을 정리한 추상 자료형

이름		입력	출력	설명
그래프 생성	`createGraph()`	최대 노드 개수 n	그래프 g	n 개의 노드를 가지는 공백(empty) 그래프 g를 생성
그래프 삭제	`deleteGraph()`	그래프 g	N/A	그래프 g의 모든 노드 V와 간선 E를 제거하고, 그래프g를 제거
간선 추가	`addEdge()`	그래프 g 노드 u 노드 v	N/A	그래프 g에 노드 u와 노드 v를 연결하는 새로운 간선 e를 추가
간선 제거	`removeEdge()`	그래프 g 노드 u 노드 v	N/A	그래프 g의 간선 (u, v) 혹은 <u, v>를 제거

간선 추가의 입력 파라미터는 간선이 부속된 2개의 노드 u와 v를 입력으로 한다.
즉, 간선의 시작 혹은 종료노드를 통해서 어떤 간선이 추가 혹은 제거되는지 알려준다.
무방향 그래프라면 노드 u와 노드 v의 순서에 상관없이 같은 간선.
하지만 방향 그래프라면 노드 u와 노드 v의 순서가 다르면 서로 다른 간선으로 여겨진다.
따라서 방향그래프에서 간선을 추가 혹은 제거 할 때는 간선의 방향성에 주의.

| 인접 행렬로 구현한 그래프

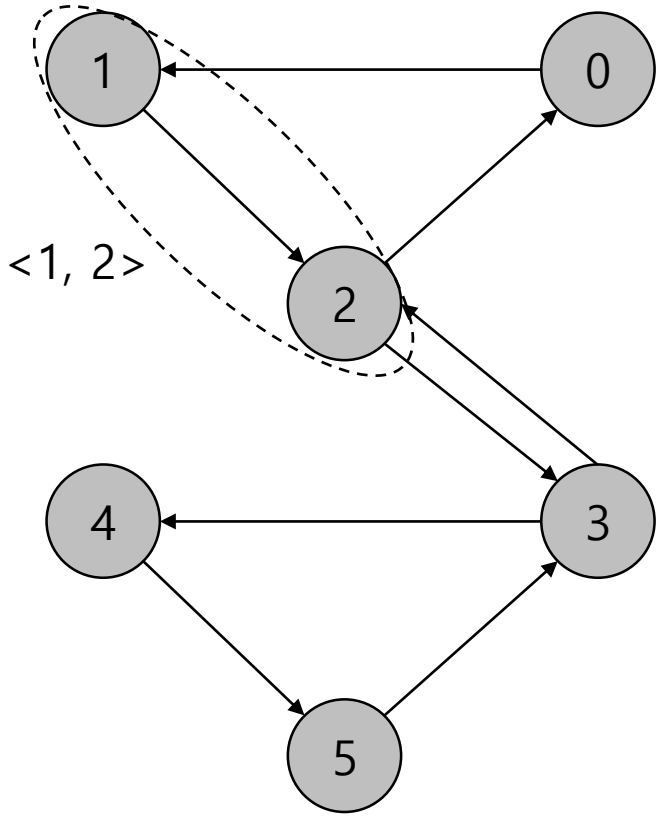
인접 행렬이란?
노드를 연결하는 간선의 정보를 저장하는 2차원 배열.



행(Row)은 시작 노드를 의미하고 열(Column)이 종료 노드를 뜻한다.

간선 <i, j>가 존재	[i][j] = 1
간선 <i, j>가 존재하지 않음	[i][j] = 0

| 인접 행렬로 구현한 그래프



[1][2]

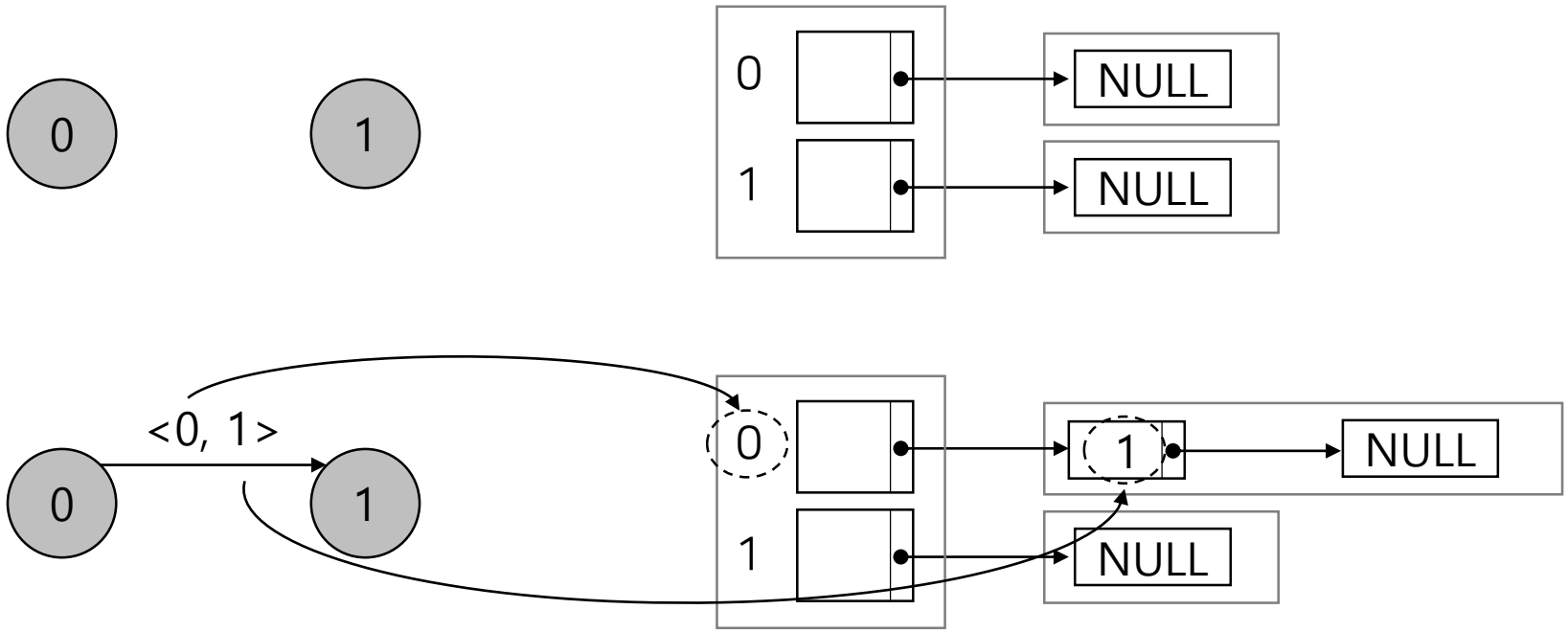
[2][1]

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

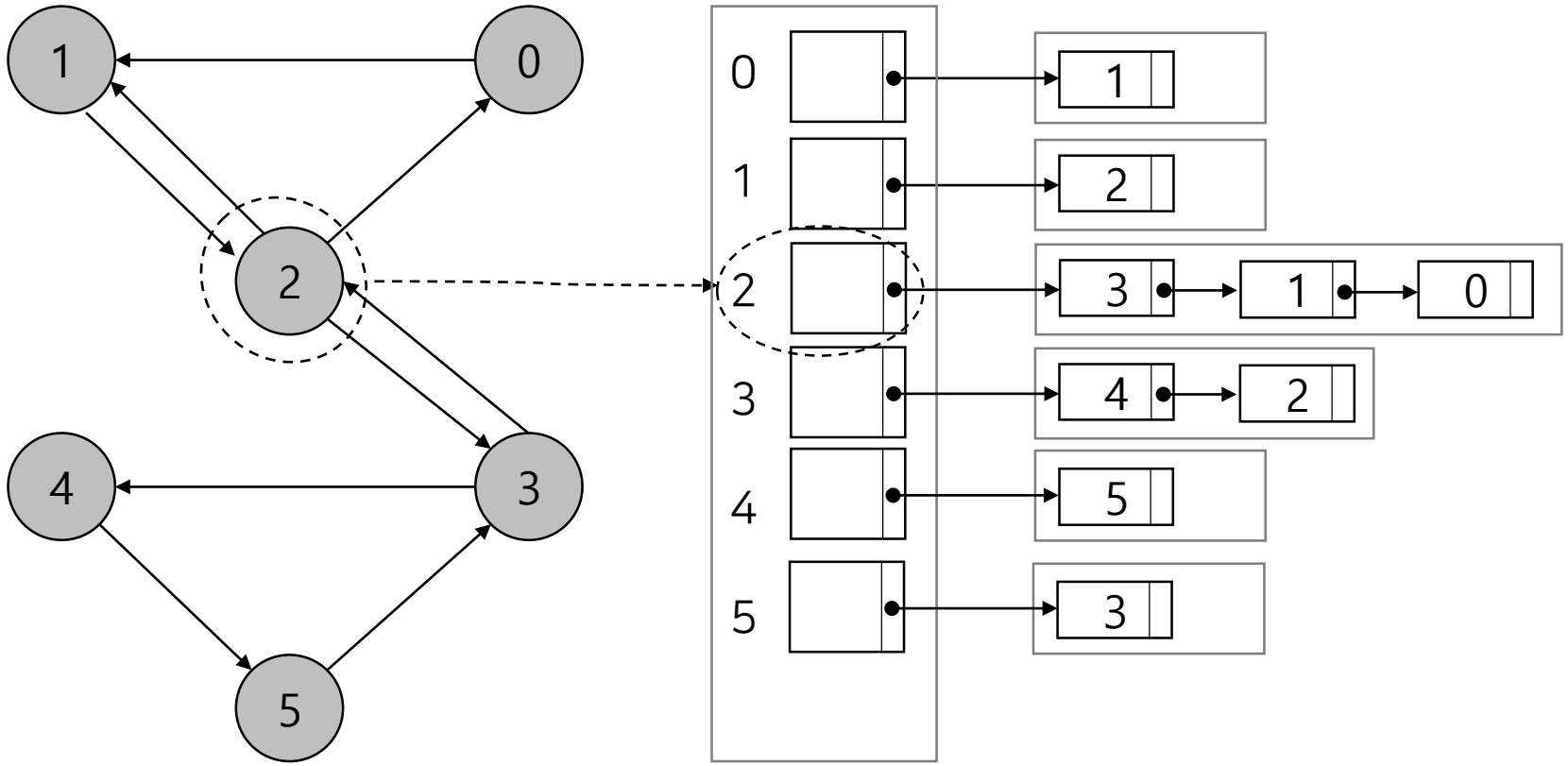
노드가 6개이기 때문에 원소의 개수는 36(6x6)개다.
반면, 간선은 모두 8개로 전체 배열의 22%(8/36)를 사용.
따라서 2차원 배열로 구현할 경우 메모리의 낭비가 크다.

| 인접 리스트로 구현한 그래프

인접 리스트란?
노드의 개수가 동적으로 정해진다면 인접 리스트는 보통 연결 리스트로 구현된다.



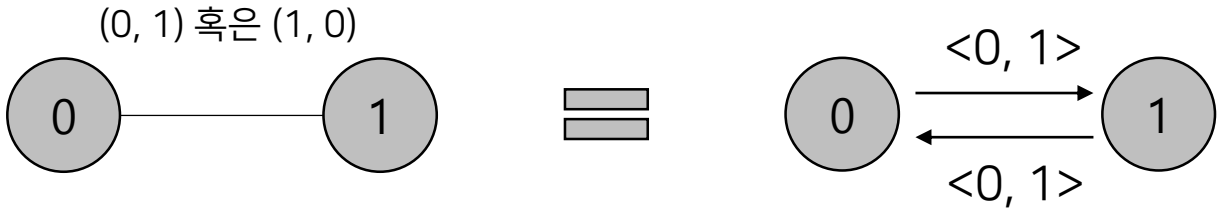
| 인접 리스트로 구현한 그래프



[2][1]

| 무방향 그래프의 구현

무방향 그래프는 방향 그래프와 달리 **간선의 대칭성(symmetry)**이라는 고유한 특성이 있다.
간선의 대칭성이란 무방향 그래프에서는 시작노드와 종료노드가 서로 반대 방향인 간선이 존재한다는 의미이다.



	0	1
0	0	1
1	1	0

[0][1]
[1][0]
대칭성

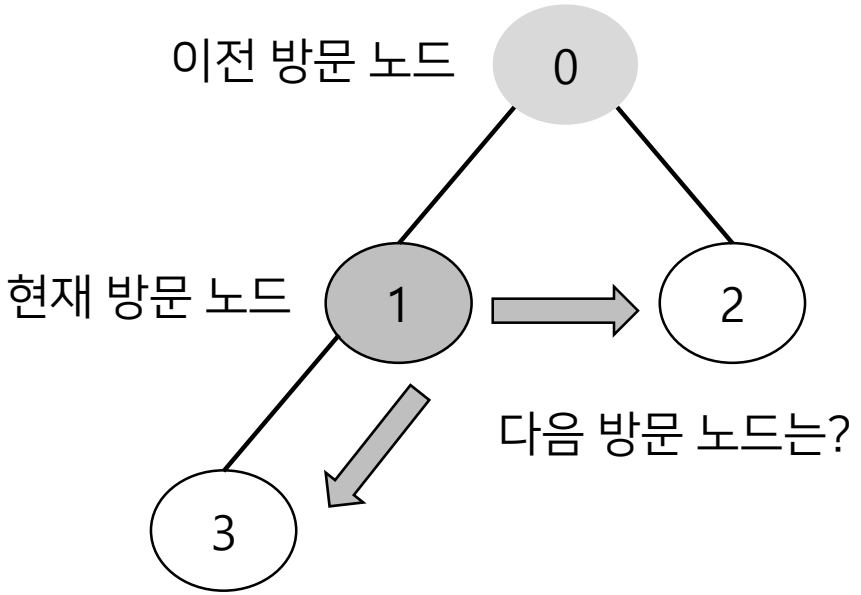
[1][0] 간선과 [0][1] 간선은 서로 행과 열의 위치가 바뀐 간선.
행과 열이 서로 바뀌었다는 점에서 대칭이다.

구현 관점에서 보자면 무방향 그래프는 간선의
대칭성을 위해서 추가되는 로직이 필요하다.

| 그래프의 탐색

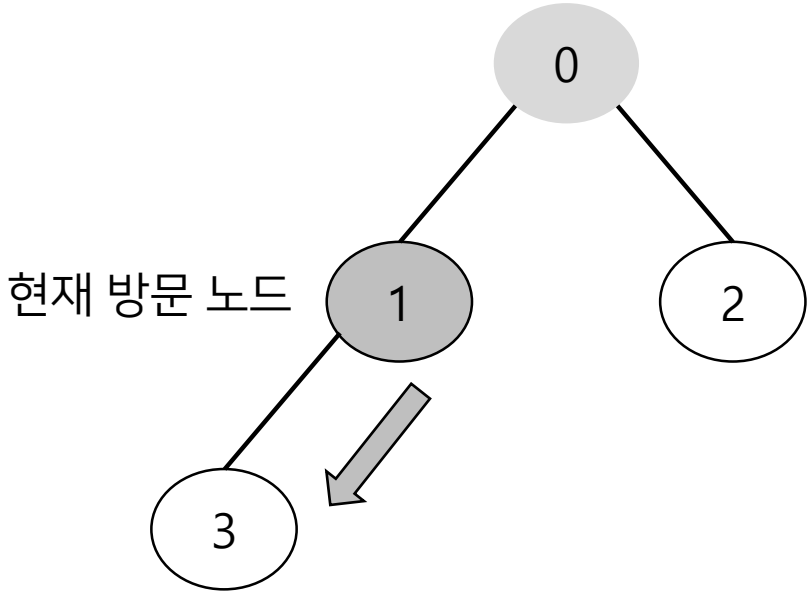
그래프 탐색은 그래프 상의 모든 노드를 한 번씩 방문하는 것.
깊이-우선탐색(DFS: depth-first search)와 **넓이-우선탐색**(BFS: breadth-first search)가 있다.

구분	다음 노드 선택 기준	
깊이-우선 탐색	더 깊은 노드	현재 방문 노드 기준
넓이-우선 탐색	더 넓은 노드	이전 방문 노드 기준



| 깊이-우선탐색(DFS: depth-first search)

깊이-우선탐색에서는 현재 선택된 노드와 연결된 노드를 먼저 선택.
즉, 현재 노드와 연결된 노드 중에서 아직 방문하지 않은 노드가 선택된다.

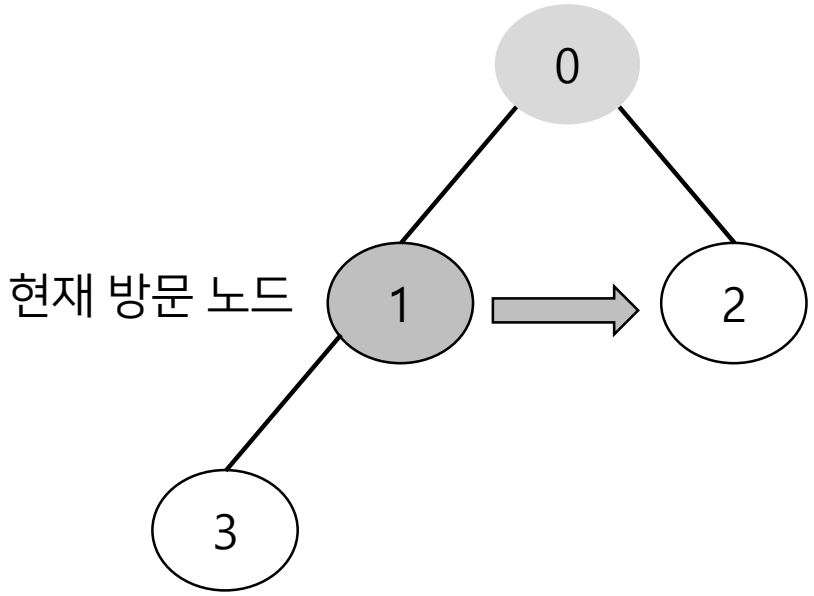



```
traversalDFS (node startNode) {
        startNode ← 방문;
        for ( all next Node ∈ (startNode의 인접 노드들) ) {
                if (nextNode != 방문) {
                        traversalDFS ( nextNode );
                }
        }
}
```

DFS : [0] → [1] → [3] → [2]

| 넓이-우선탐색(BFS: breadth-first search)

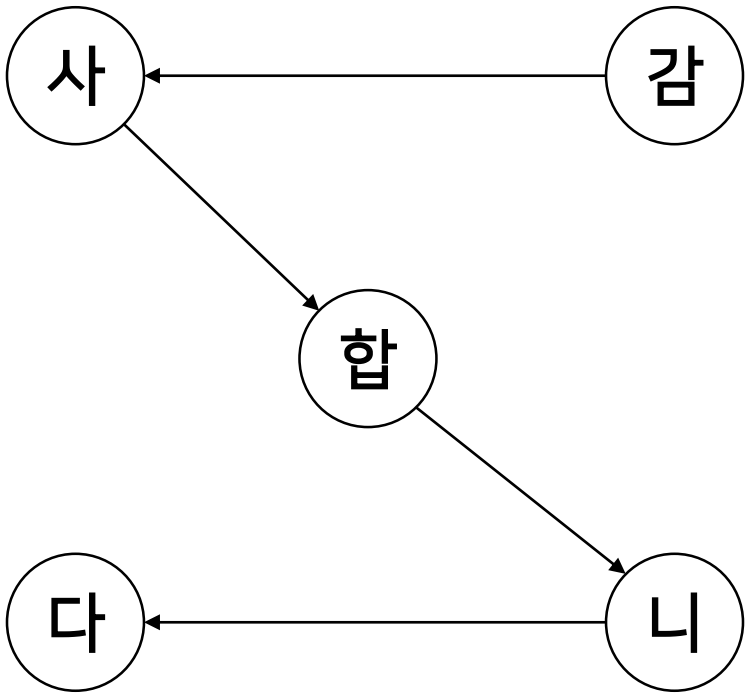
넓이-우선 탐색은 다음 방문할 노드를 선택할 때 현재 방문한 노드가 아니라 이전에 방문했던 노드와 연결된 노드를 먼저 선택하는 방법.
일반적으로 BFS는 큐(queue)를 통하여 구현될 수 있다.



BFS : [0] → [1] → [2] → [3]

```
traversalBFS (node startNode) {
        큐 Q;

        startNode ← 방문;
        enqueue(Q, v);                  //v = 현재 방문 노드
        while( not is_empty(Q) ) {
                u ← dequeue(Q);         //u = 이전에 방문한 노드
                for ( all w ∈ (startNode의 인접 노드들) ) {
                if (w != 방문) {
                        w ← 방문;       //w = 인접 노드
                        enqueue(Q, w);
                }
        }
}
```


사
감
합
다
니