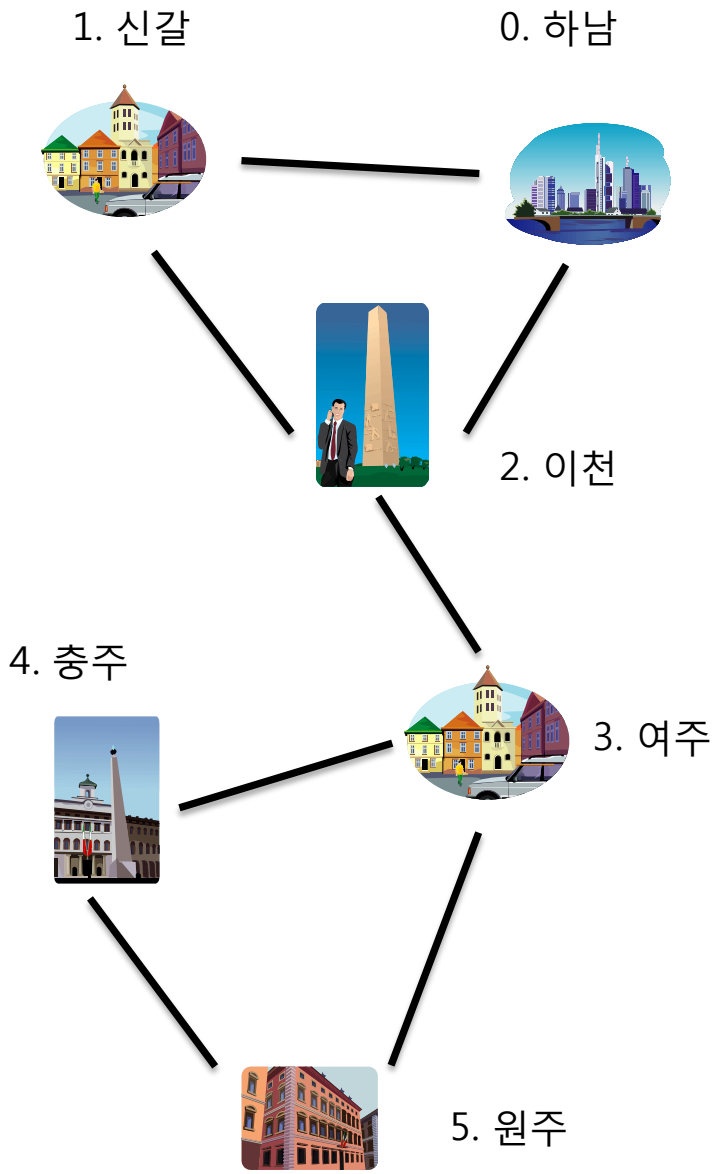
그래프

목차

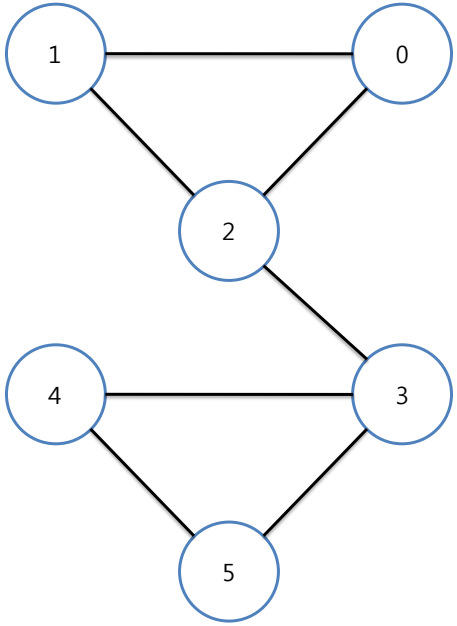
1. 그래프의 개념 (1/7)

- 그래프 = 노드 + 간선
  - Cf) 트리?

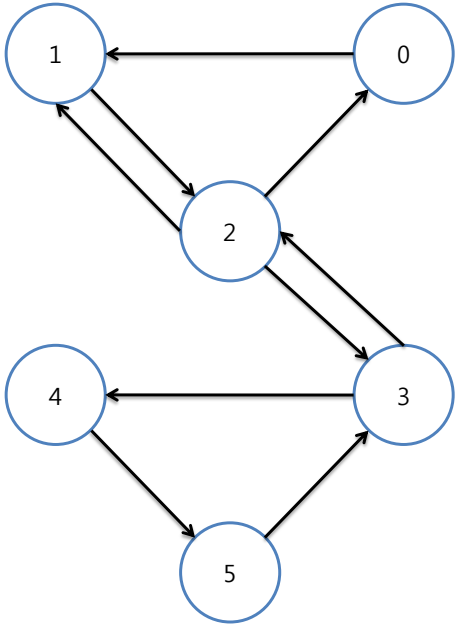


1. 그래프의 개념 (2/7)

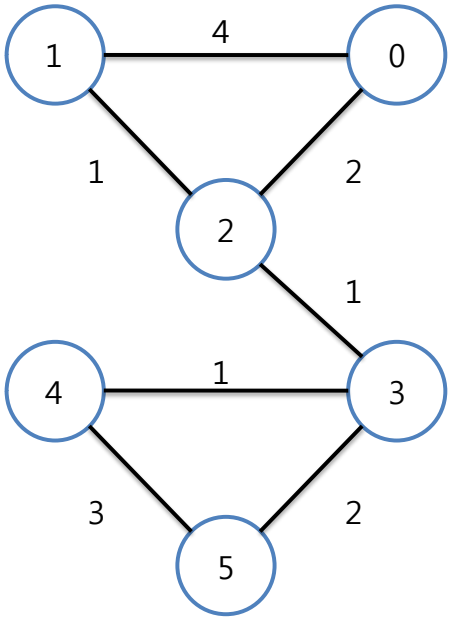
- 그래프의 종류
  - 간선의 방향성
  - 간선의 가중치



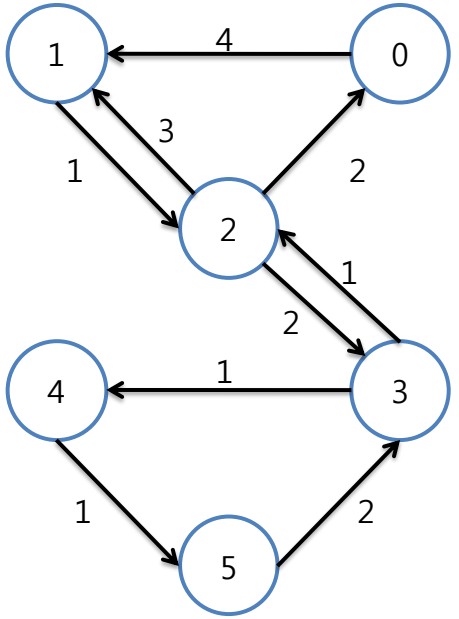
G1



G2



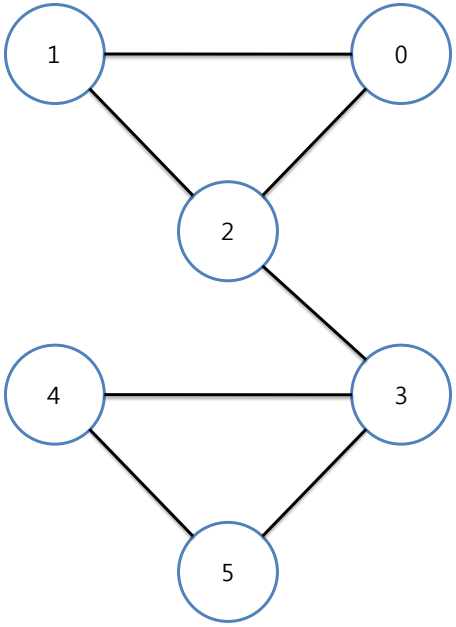
G3



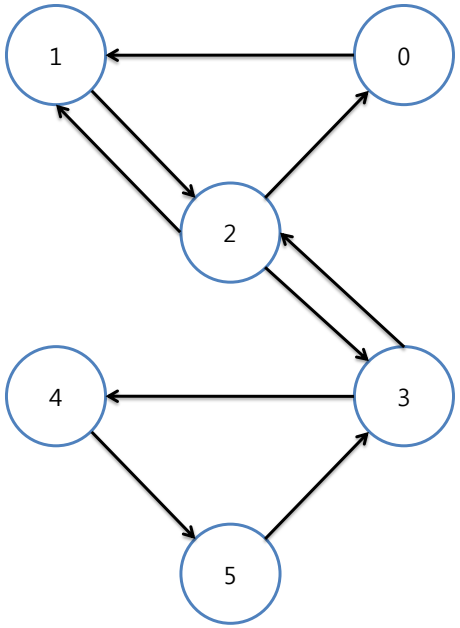
G4

1. 그래프의 개념 (3/7)

- 그래프의 식 표현



G1

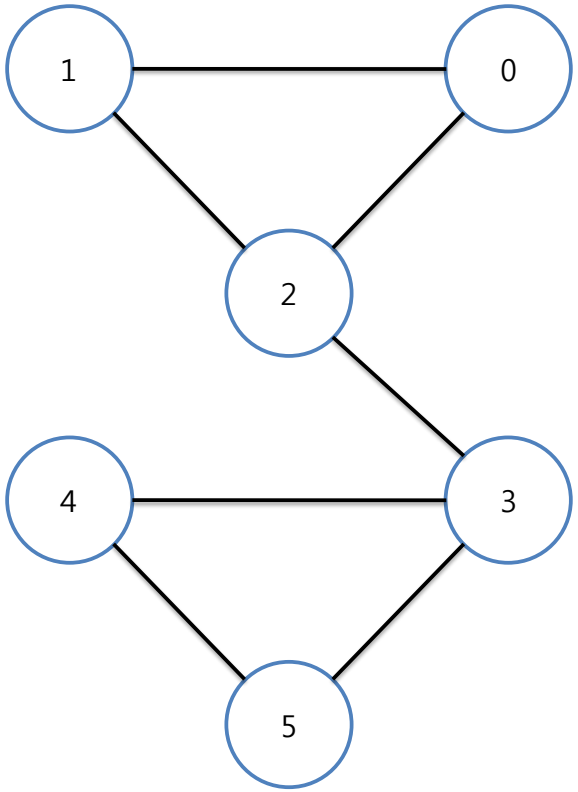


G2

- 머리-꼬리

1. 그래프의 개념 (4/7)

- 인접
- 부속
- 차수



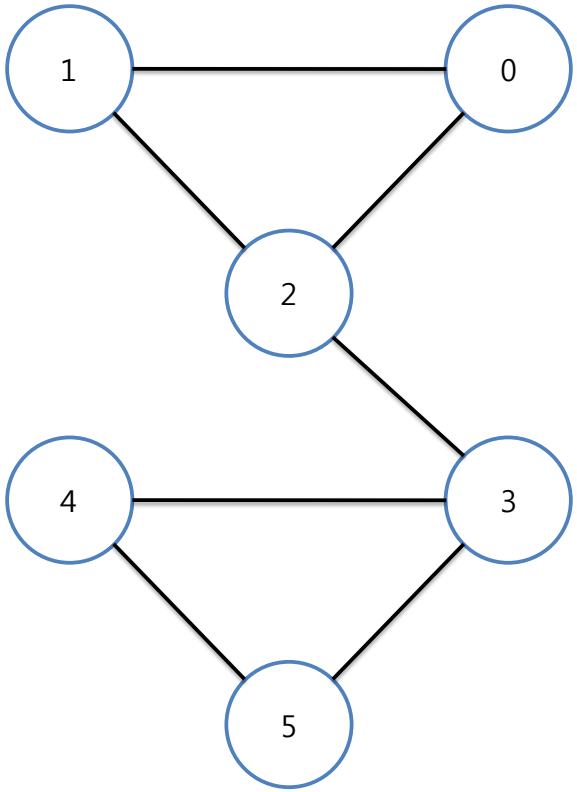
G1

1. 그래프의 개념 (5/7)

- 경로

- 단순 경로

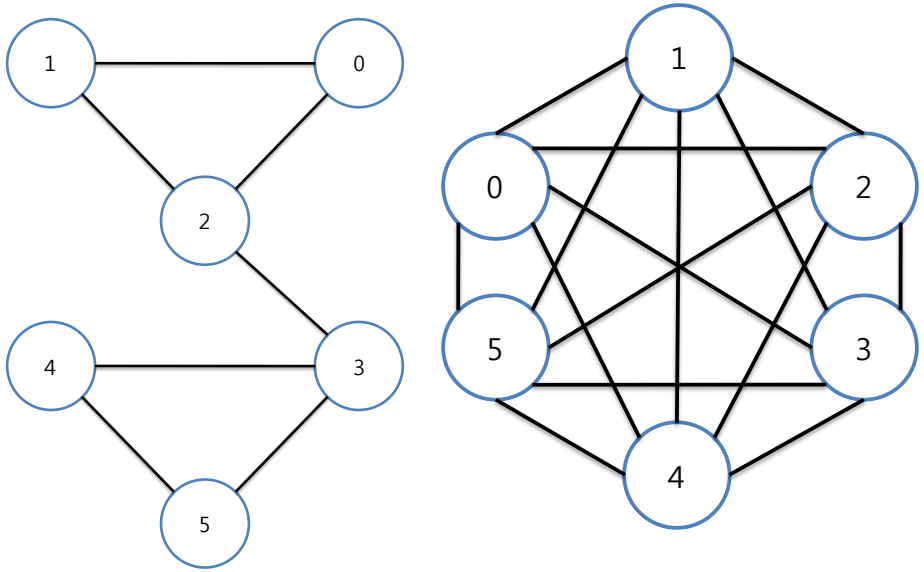
- 연결



G1

1. 그래프의 개념 (6/7)

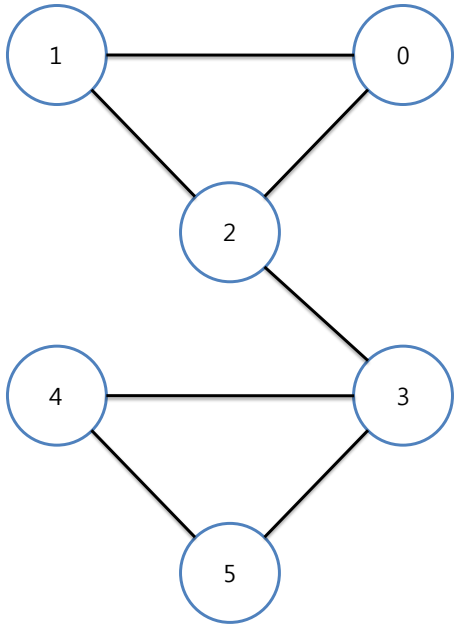
- 완전 그래프?



1. 그래프의 개념 (7/7)

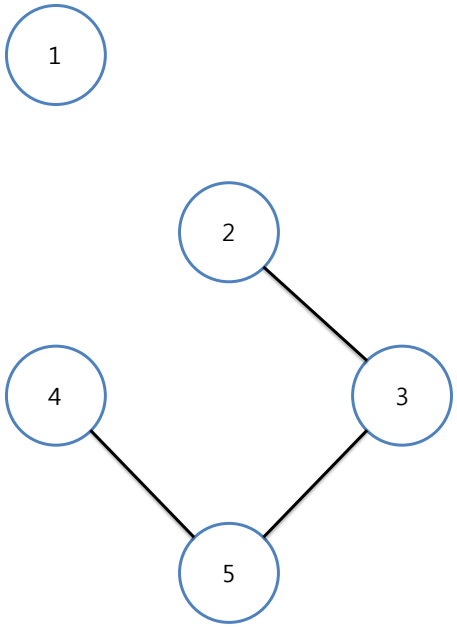
- 동일한 그래프

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

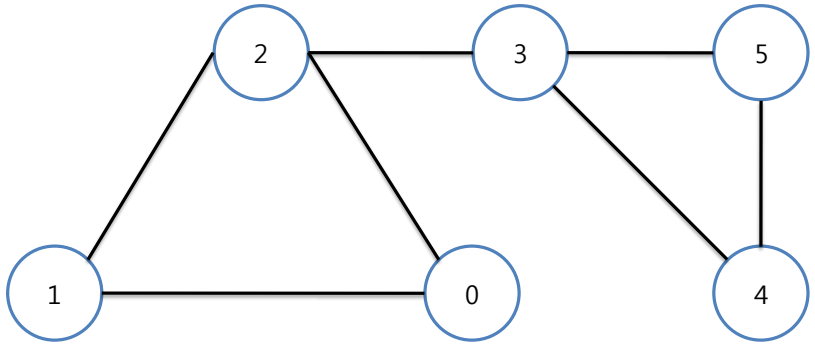


G1

- 부분 그래프



G8



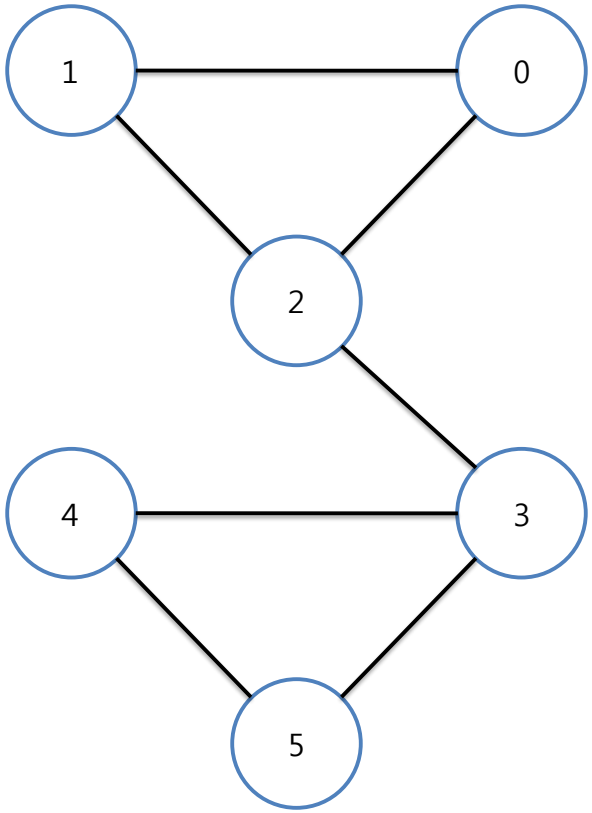
G5

2. 그래프 추상 자료형

- 그래프 생성
- 그래프 삭제
- 노드 추가
- 노드 제거
- 간선 추가
- 간선 제거
- 노드/간선 반환

3.1. 인접 행렬을 이용한 구현 (1/5)

- 간선 (i, j)

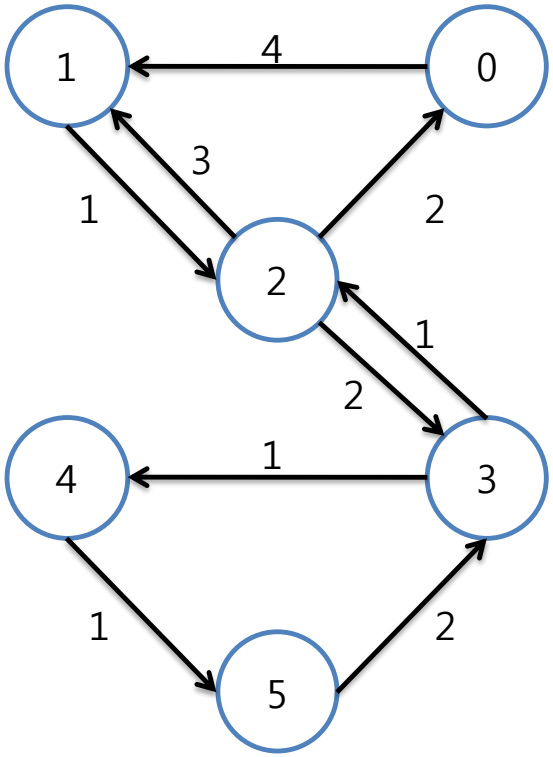


G1

	0	1	2	3	4	5	계
0		1	1				2
1	1		1				2
2	1	1		1			3
3			1		1	1	3
4				1		1	2
5				1	1		2
계	2	2	3	3	2	2	14

3.1. 인접 행렬을 이용한 구현 (2/5)

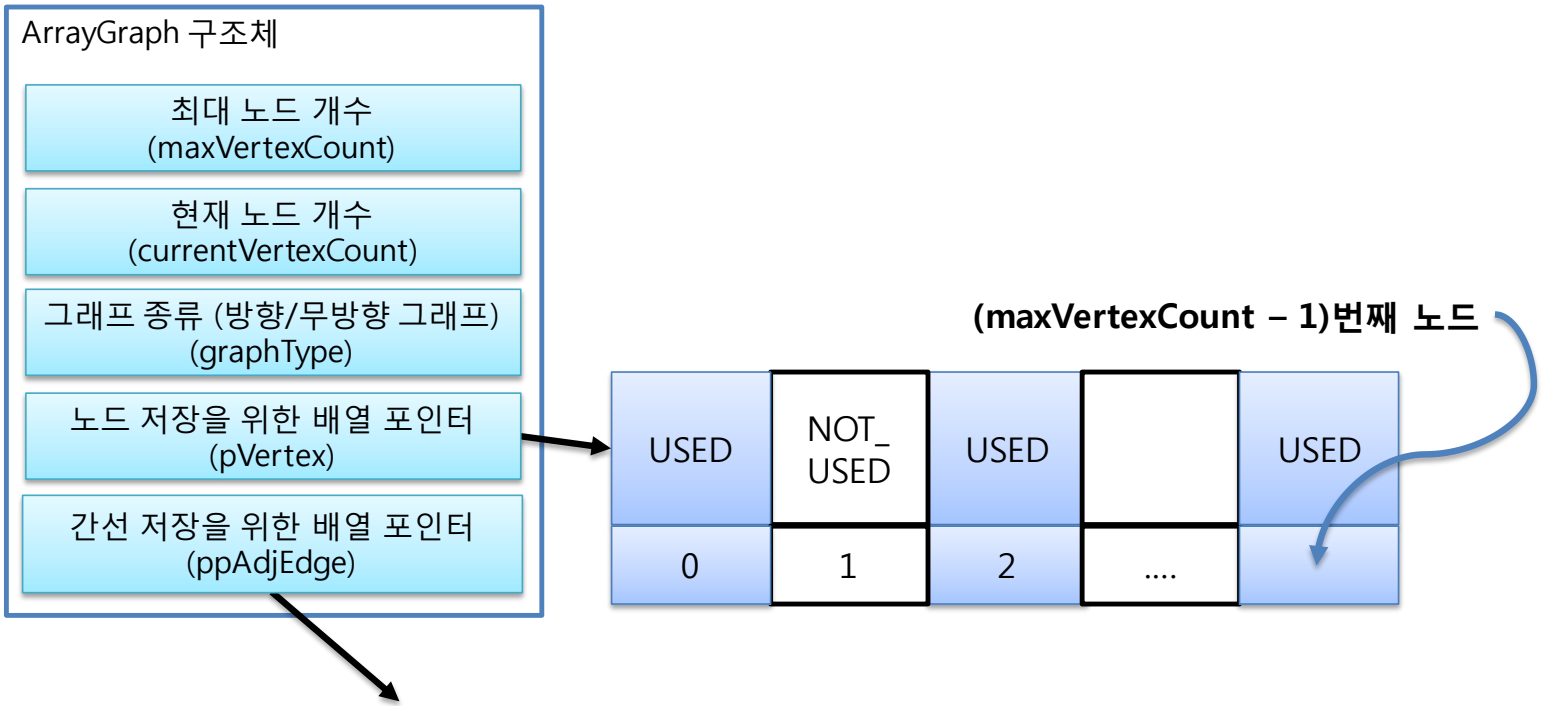
- 간선 <i, j>



G4

	0	1	2	3	4	5	계
0	0	**4**	**0**	0	0	0	4
1	**0**	0	**1**	0	0	0	1
2	**2**	**3**	0	**2**	**0**	0	7
3	0	0	**1**	0	**1**	0	2
4	0	0	0	0	0	**1**	1
5	0	0	0	**2**	0	0	2
계	2	7	2	4	1	1	17

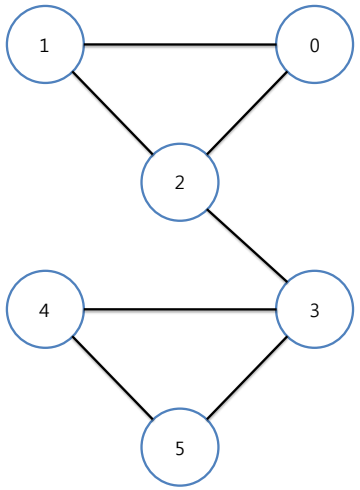
3.1. 인접 행렬을 이용한 구현 (4/5)



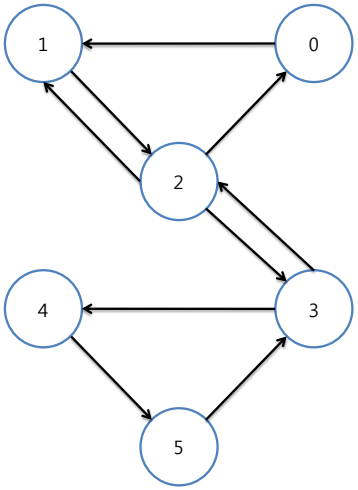
	0	1	2	...	maxVertexCount – 1
0			1		1
1					
2	1				
....					
maxVertexCount – 1	1				

3.1. 인접 행렬을 이용한 구현 (5/5)

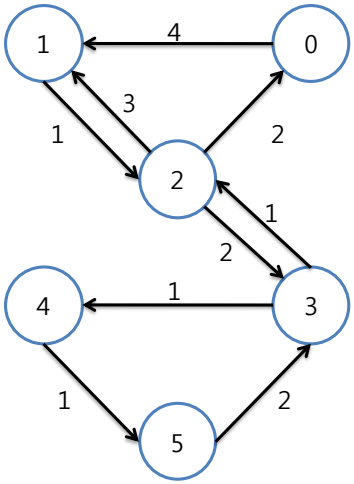
- 그래프의 생성
- 노드 추가
- 간선 추가
- 노드와 간선의 제거
- 기타
- 예제 프로그램



G1

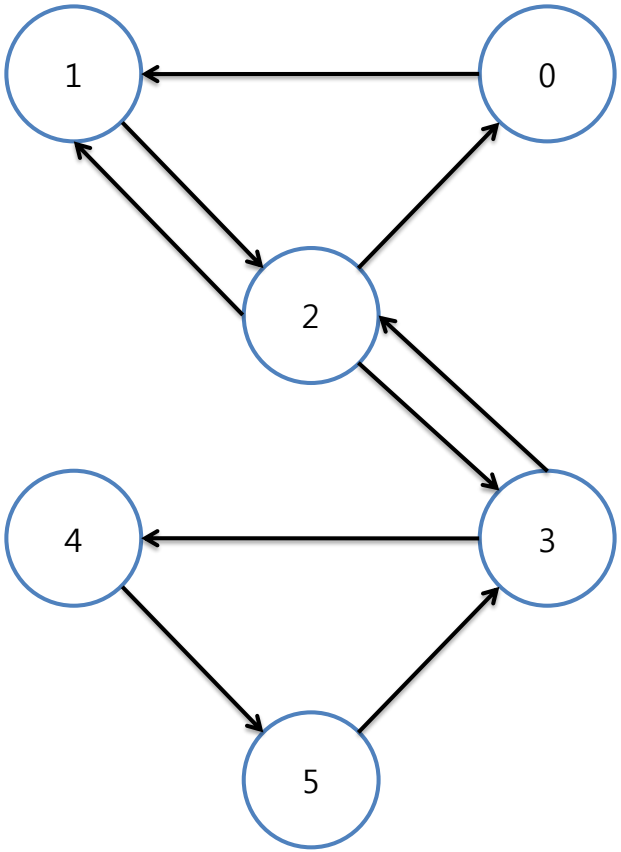


G2

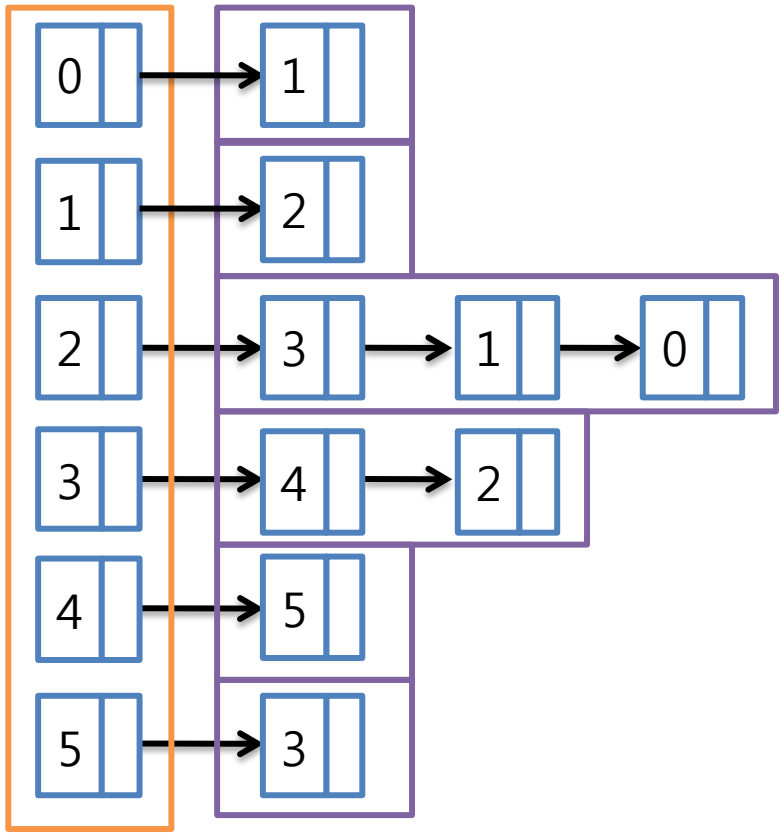


G4

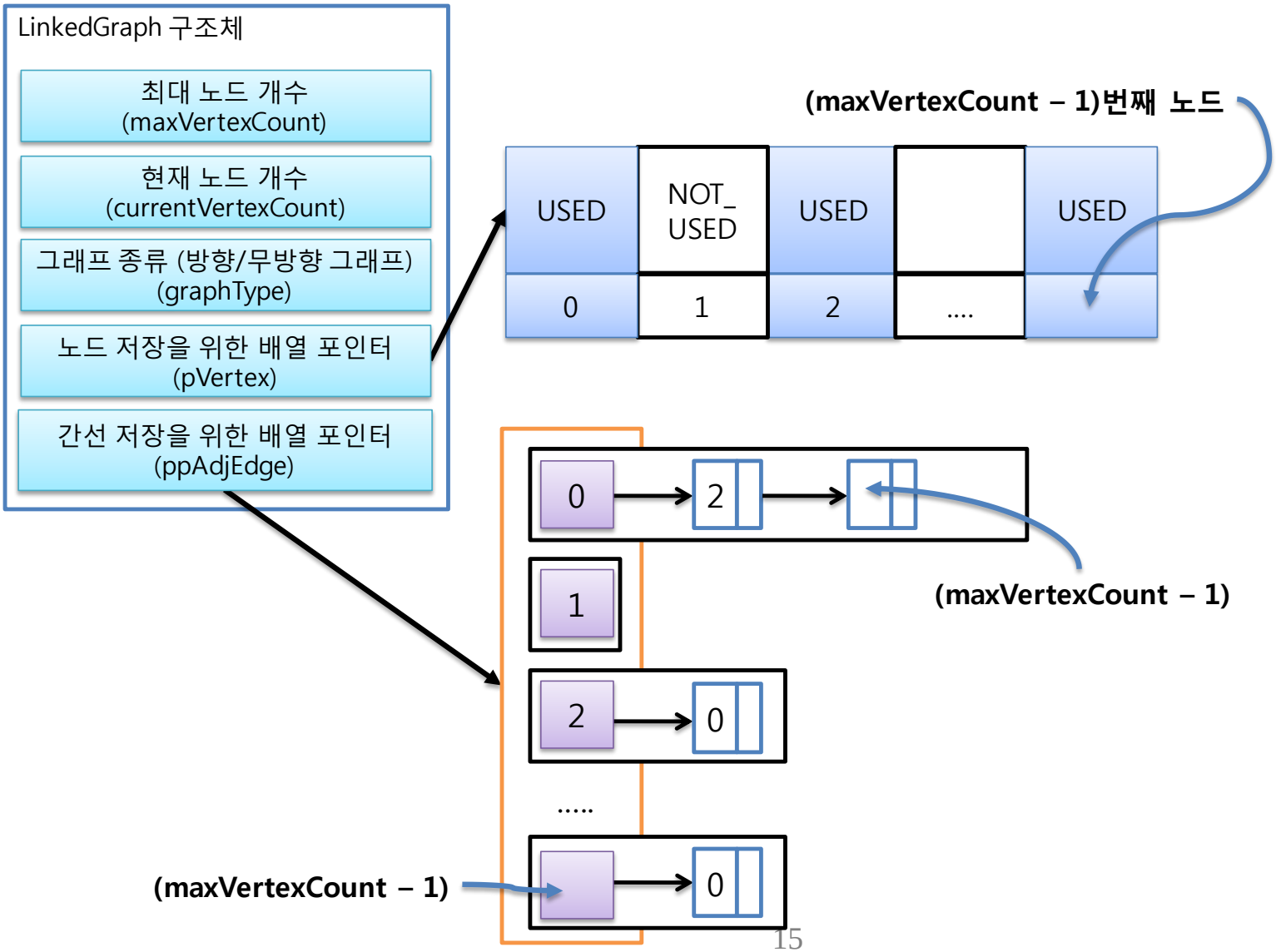
3.2. 인접 리스트를 이용한 구현 (1/5)



G2

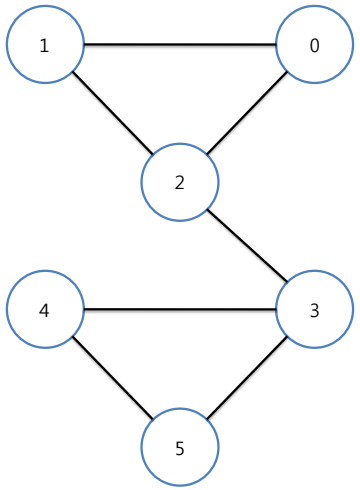


3.2. 인접 리스트를 이용한 구현 (4/5)

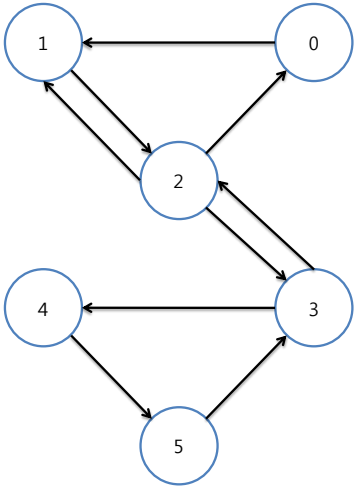


3.2. 인접 리스트를 이용한 구현 (5/5)

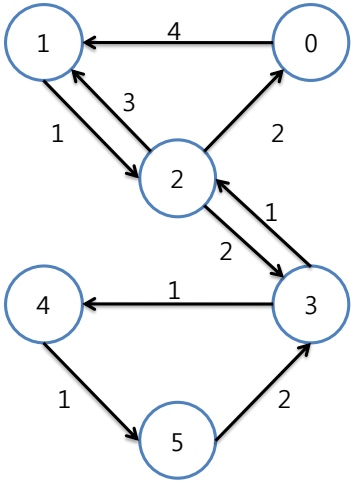
- 그래프의 생성
- 노드 추가
- 간선 추가
- 노드와 간선의 제거
- 기타
- 예제 프로그램



G1



G2

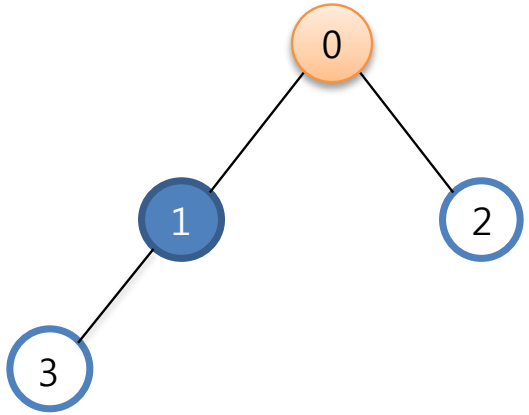


G4

4. 그래프 탐색

- 탐색이란?

- 2가지 종류
  - [깊이 우선 탐색]-DFS(Depth First Search)
  - [넓이 우선 탐색]-BFS(Breadth First Search)



4.1. 깊이-우선 탐색 (1/4)

- 재귀 호출에 의한 구현 (의사 코드)

```
traversal_DFS (node v) {
        v ← 방문;
        for( all u ∈ (v의 인접 노드들) ) {
                if (u != 방문) {
                        traversal_DFS ( u );
                }
        }
}
```

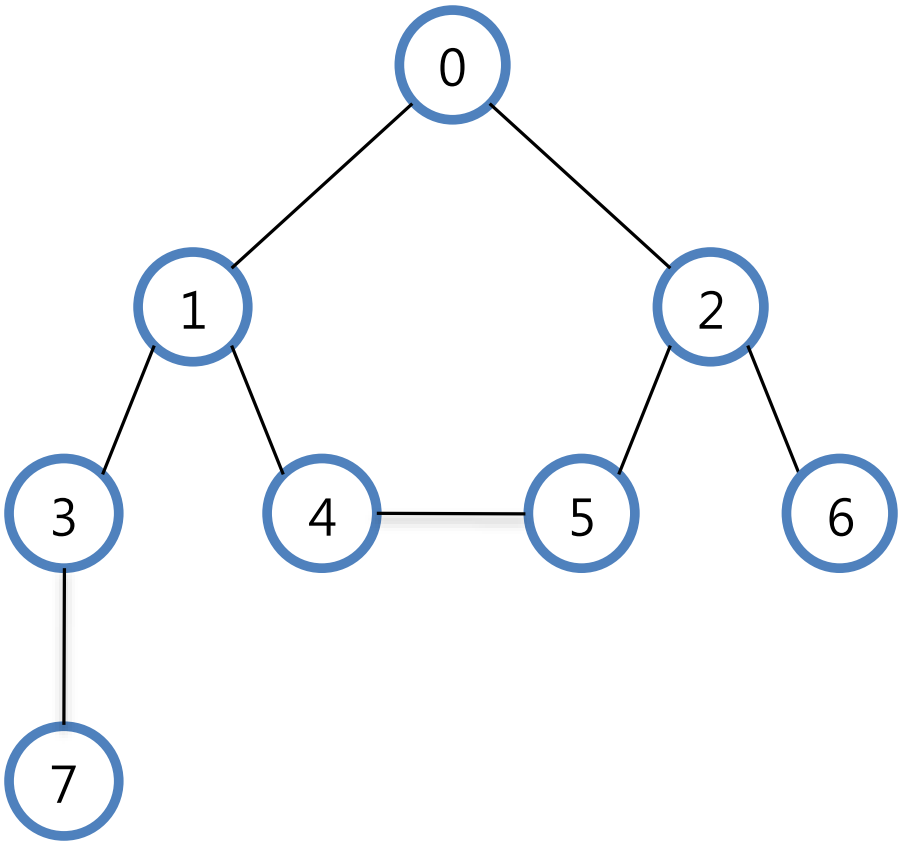
4.1. 깊이-우선 탐색 (2/4)

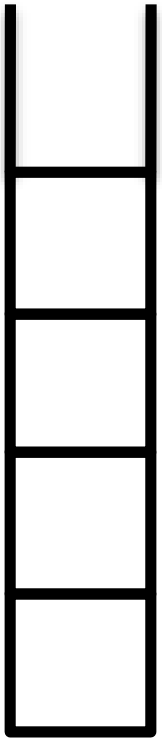
- 반복 호출에 의한 구현

```
traversal_DFS (node v) {
        스택 S;

        v ← 방문;
        push(S, v);
        while( not is_empty(S) ) {
                u ← pop(S);
                for( all w ∈ (u의 인접 노드들) ) {
                        if (w != 방문) {
                                w ← 방문;
                                push(S, w);
                        }
                }
        }
}
```

4.1. 깊이-우선 탐색 (3/4)





스택

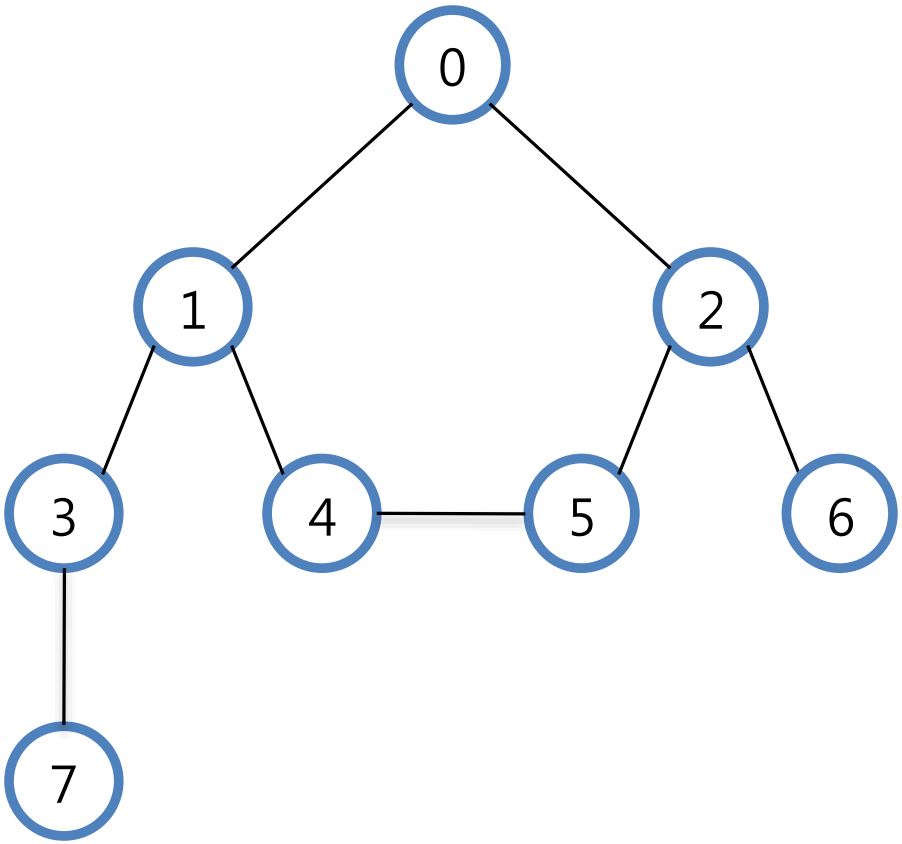
4.2. 넓이-우선 탐색 (1/3)

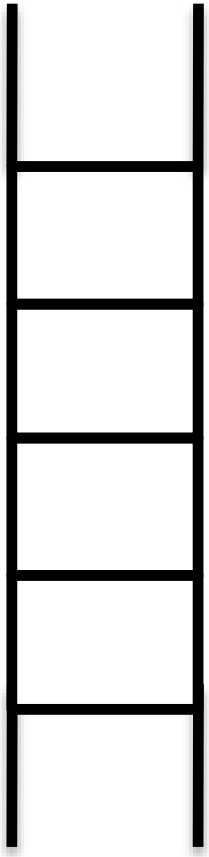
- 반복 호출에 의한 구현

```
traversal_BFS (node v) {
        큐 Q;

        v ← 방문;
        enqueue(Q, v);
        while( not is_empty(Q) ) {
                u ← dequeue(Q);
                for( all w ∈ (u의 인접 노드들) ) {
                        if (w != 방문) {
                                w ← 방문;
                                enqueue(Q, w);
                        }
                }
        }
}
```

4.2. 넓이-우선 탐색 (2/3)

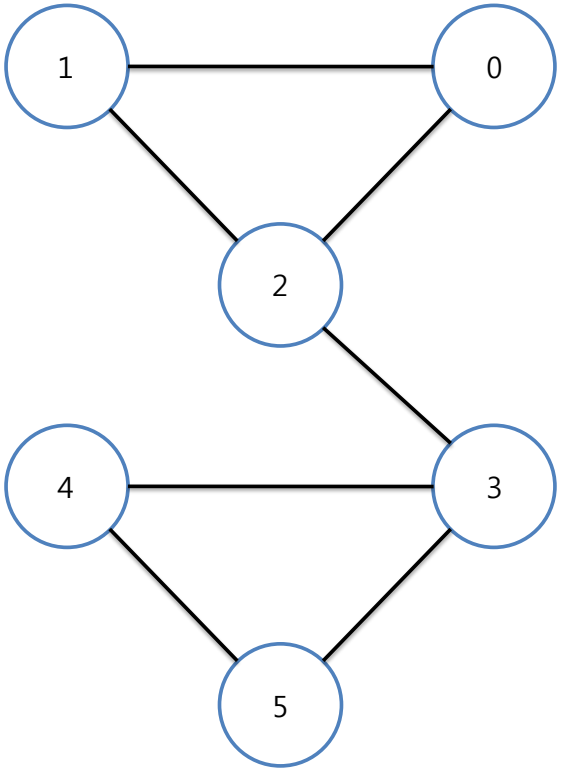




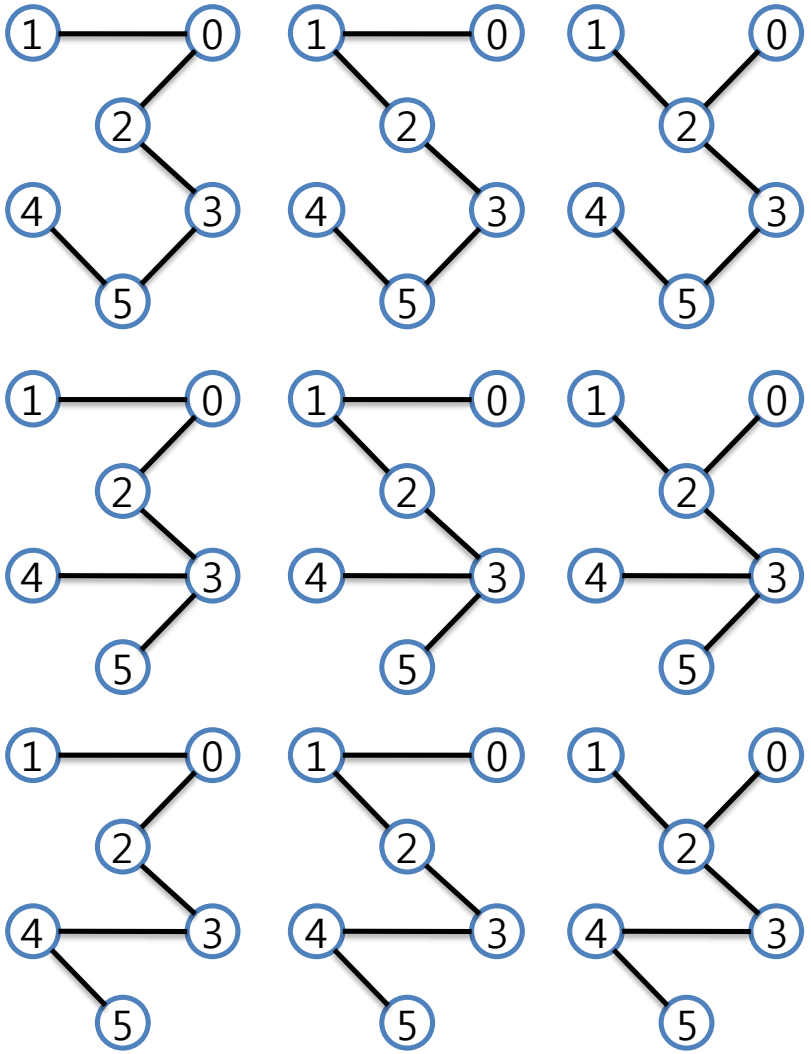
5. 신장 트리와 최소 비용 신장 트리

- 신장 트리

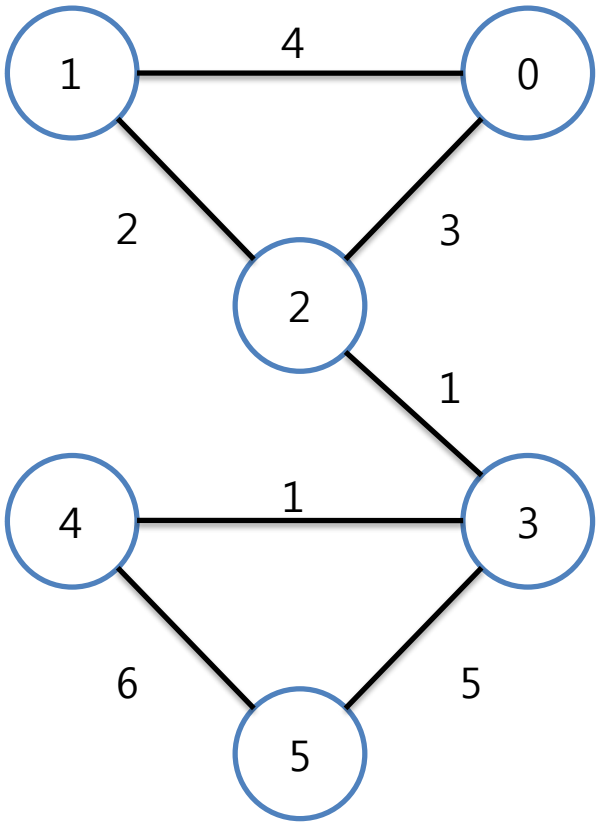
$G = (V, E)$



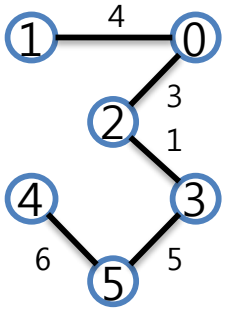
G1



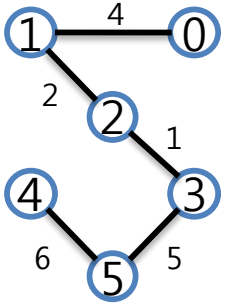
5.1. 최소 비용 신장 트리



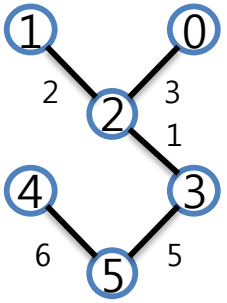
G3



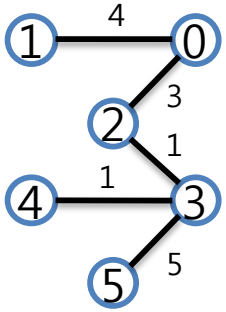
19



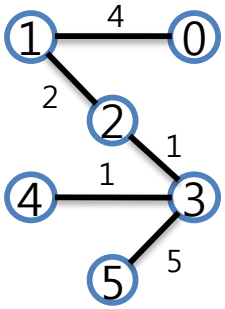
18



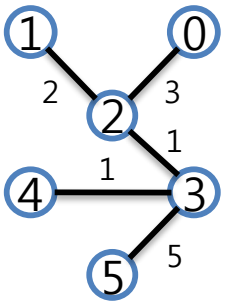
17



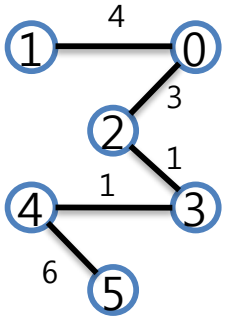
14



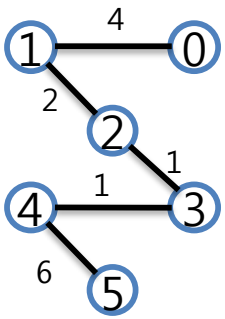
13



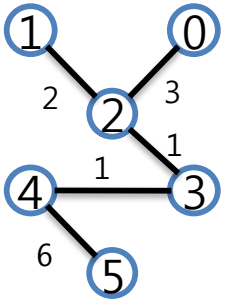
12



15



14



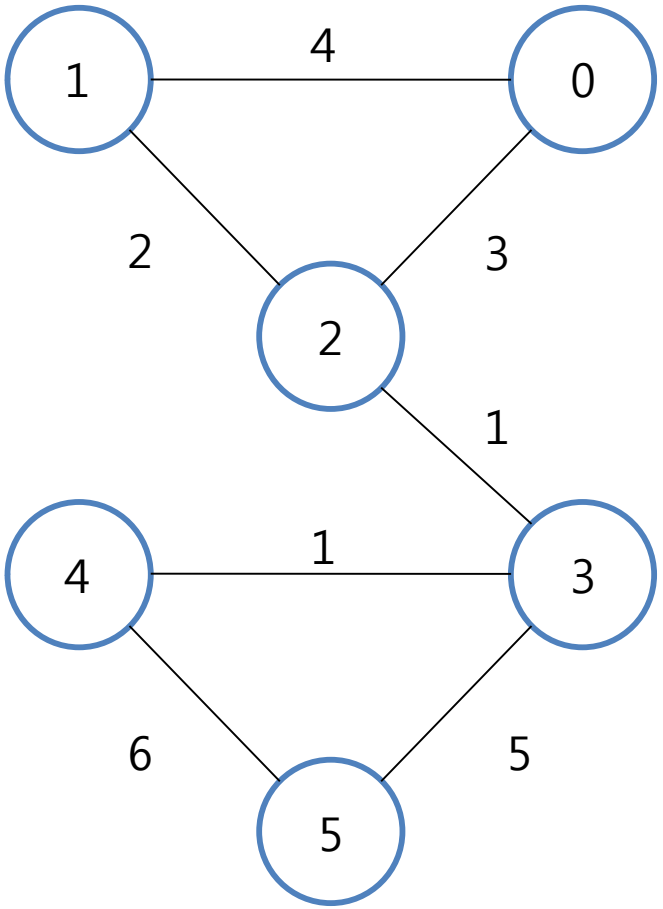
13

5.2. Kruskal 알고리즘 (1/3)

- 의사 코드

<table>
<tr><td colspan="2">초기화</td><td>(1) 그래프 G, H
H = (V(G), E(H))
단, E(H) =ø
(2)그래프 G의 모든 (간선)을 (가중치) 값에 따라
(오름차)순으로 정렬한다.</td></tr>
<tr><td rowspan="2">루프</td><td>Step-1</td><td>정렬된 간선들인 중에서
(1) 가장 가중치가 작으면서
(2) 순환을 발생시키지 않는
간선 e를 추출</td></tr>
<tr><td>Step-2</td><td>그래프 G의 모든 노드가 연결될 때까지 Step-1을 반복</td></tr>
</table>

5.2. Kruskal 알고리즘 (2/3)



간선	가중치
(2, 3)	1
(3, 4)	1
(1, 2)	2
(0, 2)	3
(0, 1)	4
(3, 5)	5
(4, 5)	6

5.3. Prim 알고리즘 (1/3)

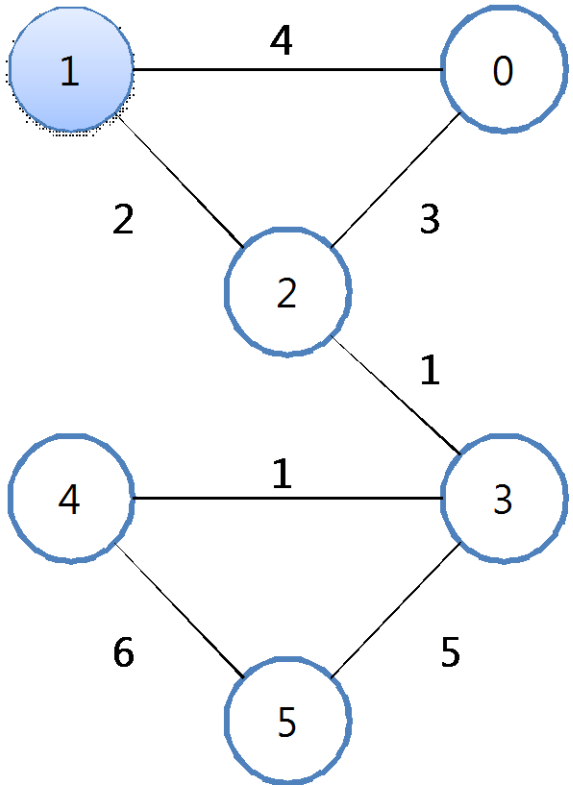
- 의사 코드

- 의사 코드

<table>
<tr><td colspan="2">초기화</td><td>V(T) = { r }
E(T) = ø</td></tr>
<tr><td rowspan="2">루프</td><td>Step-1</td><td>V(T)와 (연결)된 모든 간선들 중에서
(1) 가장 가중치가 작으면서
(2) 순환을 발생시키지 않는 간선을 선택

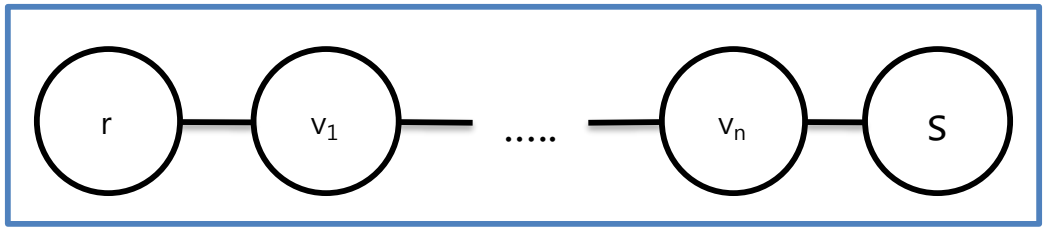
E(T) = E(T) ∪ { e }, e: 추가되는 (최소 가중치) 간선
V(T) = V(T) ∪ { w }, w: (e와 부속된로 V(T)에 포함되지 않던) 노드</td></tr>
<tr><td>Step-2</td><td>그래프 G의 모든 노드가 V(T)에 포함될 때까지 Step-1을 반복</td></tr>
</table>

5.3. Prim 알고리즘 (2/3)

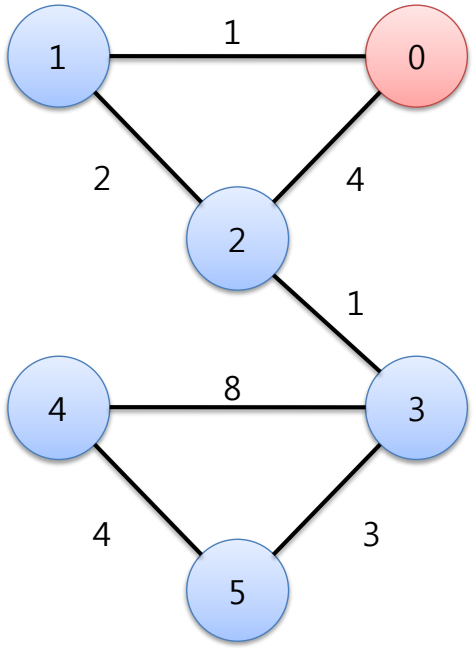


6. 최단 경로

- 최단 경로 문제란?

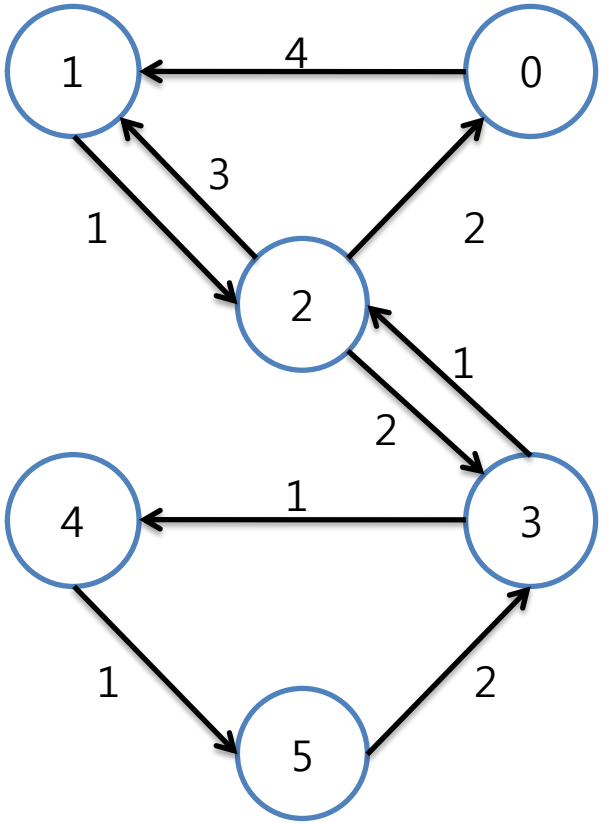


- 종류
  1)
  2)
  3)



G4

6.1. 단일 시작점에서: Dijkstra 알고리즘 (1)

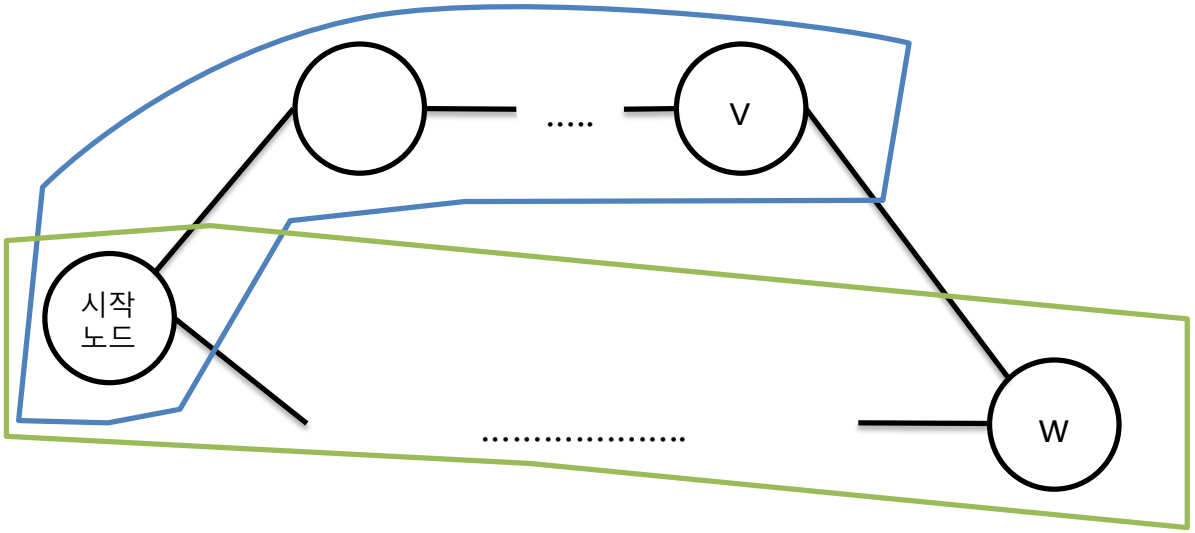


G4

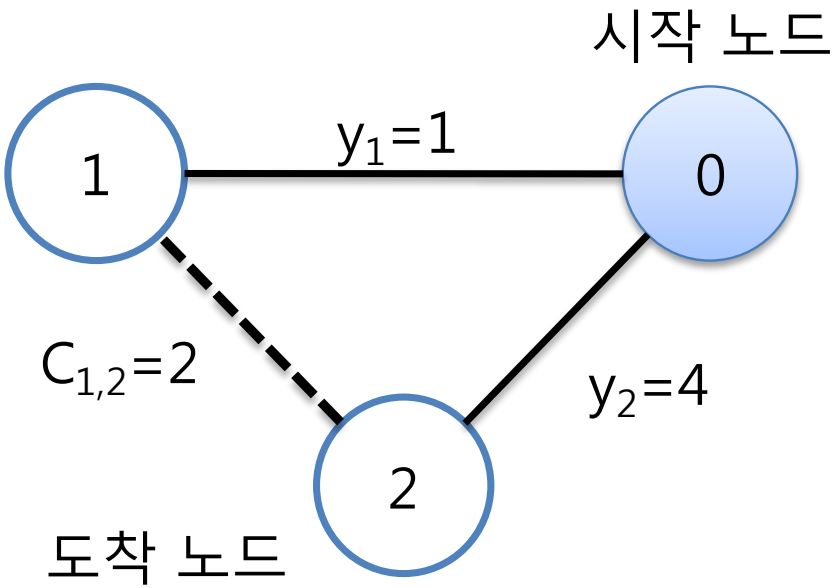
시작 노드	도착 노드
0	1
0	2
0	3
0	4
0	5

6.1. 단일 시작점에서: Dijkstra 알고리즘 (2)

- 기본 개념
  - (1) 노드 v
  - (2) 노드 w
  - (3) 개선 가능성 점검



6.1. 단일 시작점에서: Dijkstra 알고리즘 (3)



6.1. 단일 시작점에서: Dijkstra 알고리즘 (4)

- 의사 코드

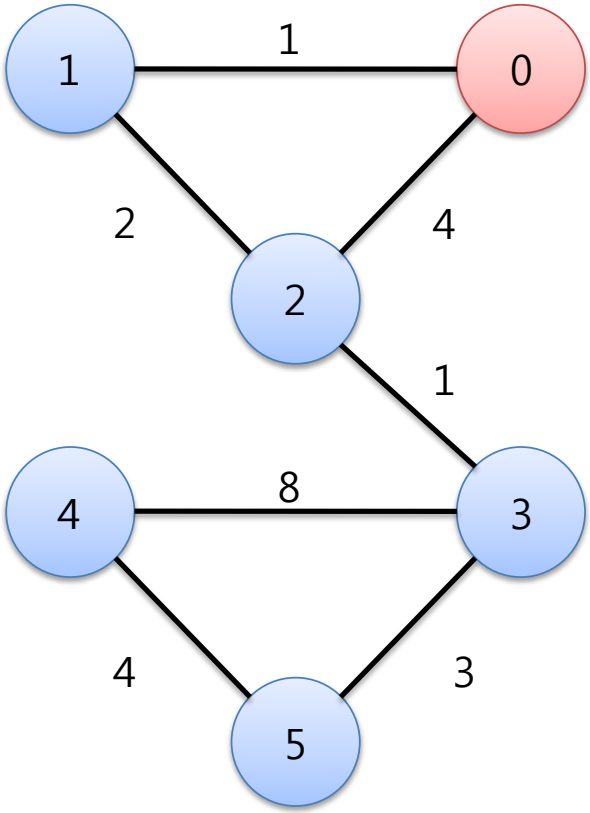
<table>
<tr><td colspan="2">초기화</td><td>1) 노드 집합 S 초기화
S = V(G)

2) 거리 초기화: 시작 노드 r, 다른 노드들 w</td></tr>
<tr><td rowspan="2">루프</td><td>Step-1</td><td>1) 집합 S 중에서 최단 거리를 가진 노드 v를 찾아서 제거

2) 노드 v에 인접한 노드 w에 대해서 다음 조건 검사를 실시

$y_w = y_v + c_{vw}$, if $y_v + c_{vw} < y_w$</td></tr>
<tr><td>Step 2</td><td>집합 S에 모든 노드가 제거될 때까지 Step 1을 반복</td></tr>
</table>

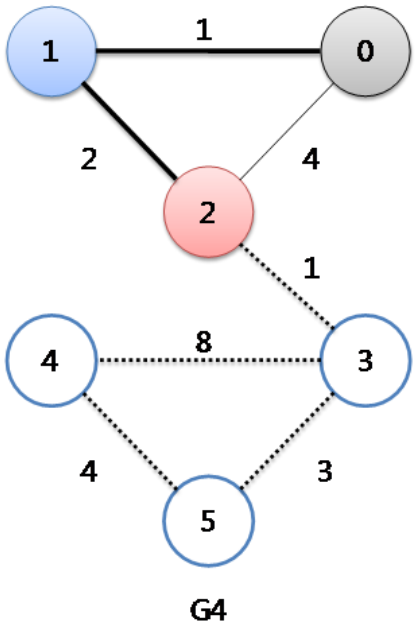
6.1. 단일 시작점에서: Dijkstra 알고리즘 (5)



G4

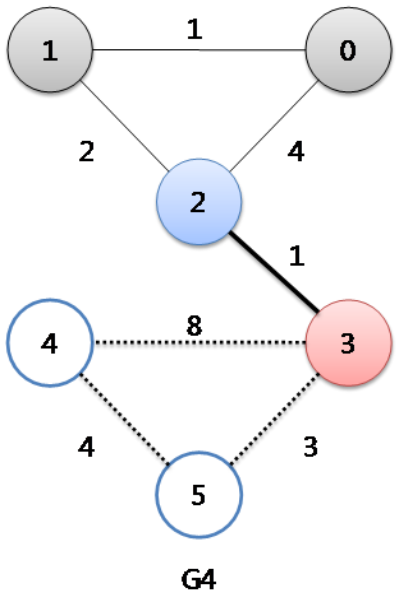
집합 (S)
1
2
3
4
5

도착 노드 (v)	거리(y_v)
1	1
2	2
3	∞
4	∞
5	∞



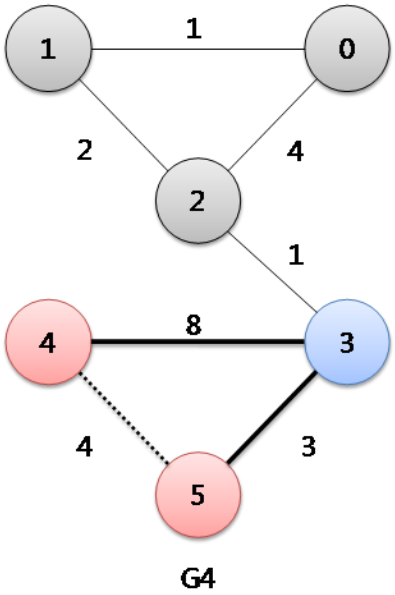
집합 (S)
~~1~~
2
3
4
5

도착 노드 (v)	거리(y_v)	단축거리 ($y_{v'}$)
1	1	-
2	4	3
3	∞	-
4	∞	-
5	∞	-



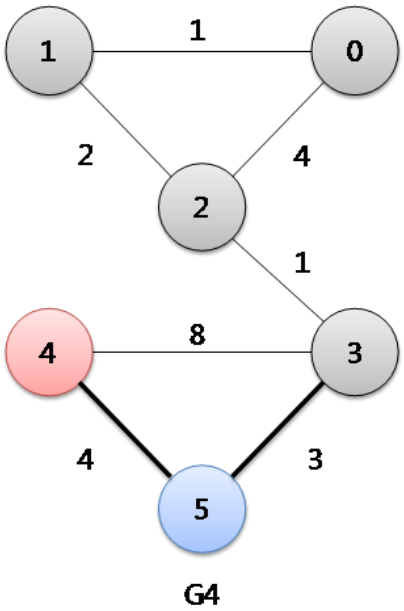
집합 (S)
~~2~~
3
4
5

도착 노드 (v)	거리(y_v)	단축거리 ($y_{v'}$)
1	1	-
2	3	-
3	∞	4
4	∞	-
5	∞	-



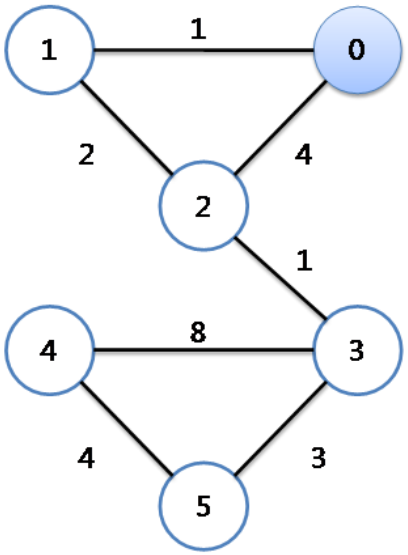
집합 (S)
~~3~~
4
5

도착 노드 (v)	거리(y_v)	단축거리 ($y_{v'}$)
1	1	-
2	3	-
3	4	-
4	∞	12
5	∞	7



집합 (S)
4
~~5~~

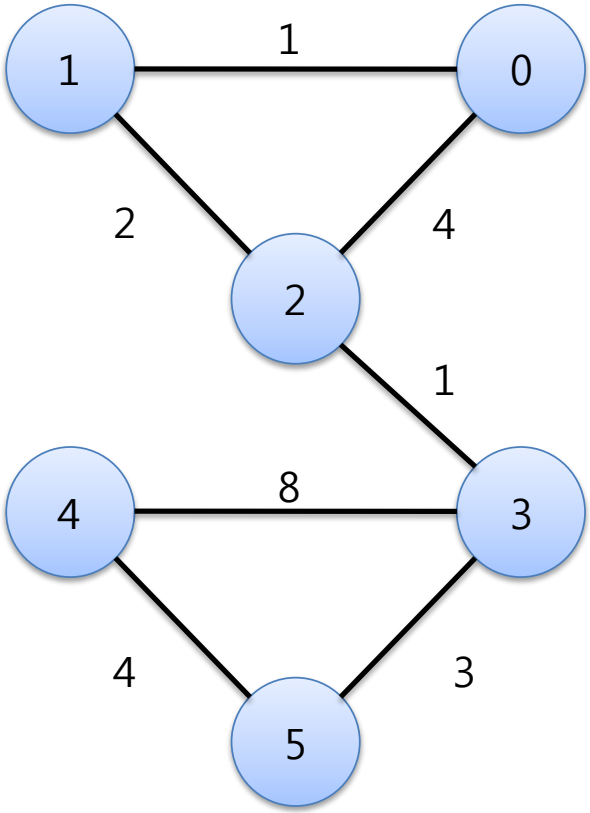
도착 노드 (v)	거리(y_v)	단축거리 ($y_{v'}$)
1	1	-
2	3	-
3	4	-
4	12	11
5	7	-



G4

도착 노드(v)	거리(y_v)	이전 연결 노드
1	1	노드 0
2	3	노드 1
3	4	노드 3
4	11	노드 5
5	7	노드 3

6.2. 모든 최단 경로 구하기: Floyd 알고리즘 (1)



G4

시작 노드	도착 노드					
0	–	1	2	3	4	5
1	0	–	2	3	4	5
2	0	1	–	3	4	5
3	0	1	2	–	4	5
4	0	1	2	3	–	5
5	0	1	2	3	4	–

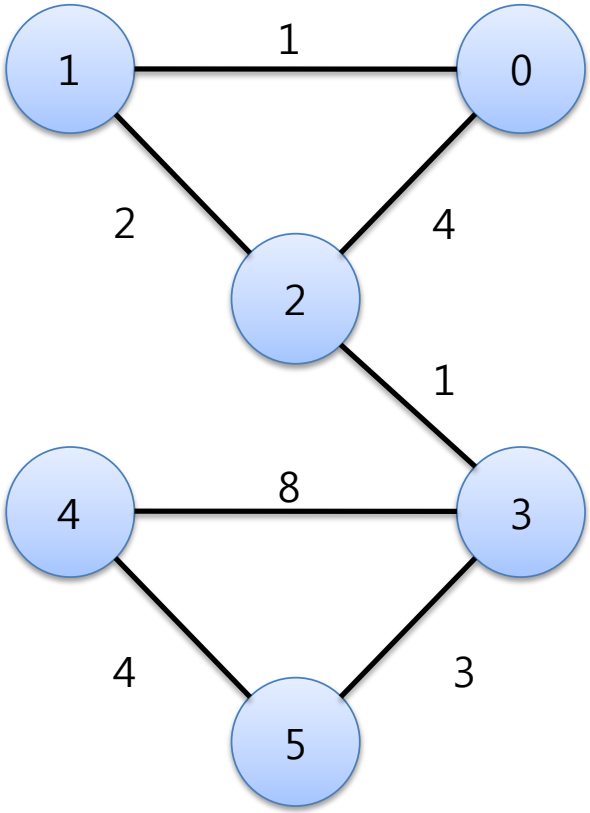
6.2. 모든 최단 경로 구하기: Floyd 알고리즘 (2)

- 의사 코드

Step 0	그래프 G의 모든 노드 r, s에 대해 2차원 배열 A 초기화 $A^{-1}[r][s] = c_{r,s}$, r과 s 사이에 간선이 존재 $A^{-1}[r][s]=\infty$, r과 s 사이에 간선이 존재하지 않음

Step 1	for (v : 그래프 G의 모든 노드)
	for (r: 그래프 G의 모든 노드)
	for (s: 그래프 G의 모든 노드)
	if ($A^{v-1}[r][v] + A^{v-1}[v][s] < A^{v-1}[r][s]$)
	$A^{v}[r][s] = A^{v-1}[r][v] + A^{v-1}[v][s]$

6.2. 모든 최단 경로 구하기: Floyd 알고리즘 (3)



G4

		도착 노드					
		0	1	2	3	4	5
시작 정점	0	0	1	4	∞	∞	∞
	1	1	0	2	∞	∞	∞
	2	4	2	0	1	∞	∞
	3	∞	∞	1	0	8	3
	4	∞	∞	∞	8	0	4
	5	∞	∞	∞	3	4	0

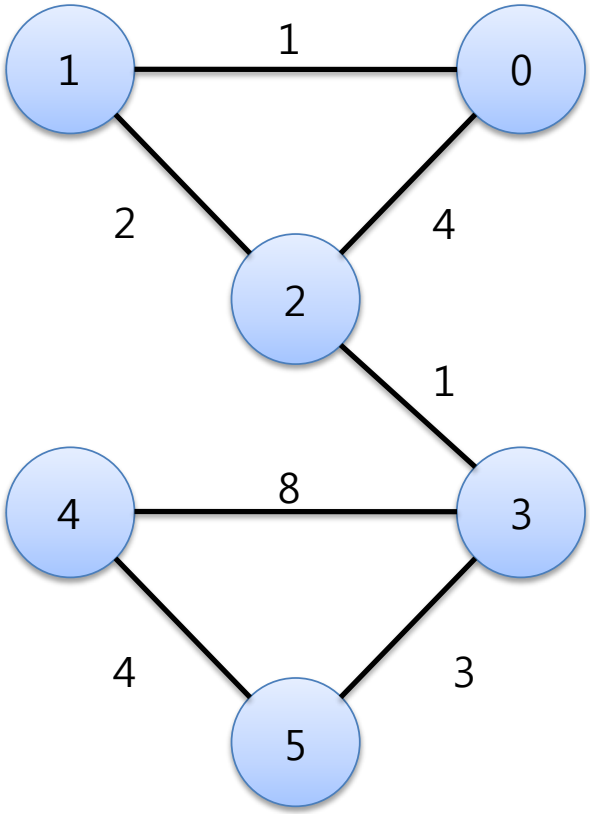
6.3. 도달 가능성 구하기

- 의사 코드

초기화	그래프 G의 모든 노드 r, s에 대해 2차원 배열 A 초기화 $A^{-1}[r][s] = 1$, r과 s 사이에 간선이 존재 $A^{-1}[r][s] = 0$, r과 s 사이에 간선이 존재하지 않음

루프	for (v : 그래프 G의 모든 노드)
	for (r: 그래프 G의 모든 노드)
	if ($A^{v-1}[r][v] > 0$)
	for (s: 그래프 G의 모든 노드)
	if ($A^{v-1}[v][s] > 0$)
	$A^{v}[r][s] = 1$

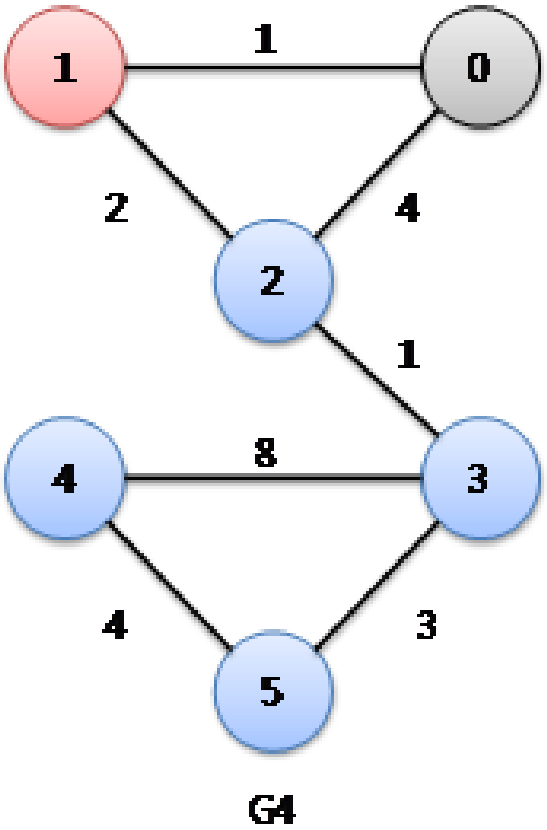
- 첫 번째 루프
  - 중간 노드 v=0인 경우
    - $A^0[1][2] = 2 < A^{-1}[1][0] + A^{-1}[0][2] = 1+4 = 5$



G4

		도착 노드					
		0	1	2	3	4	5
시작 정점	0	0	1	4	∞	∞	∞
	1	1	0	2	∞	∞	∞
	2	4	2	0	1	∞	∞
	3	∞	∞	1	0	8	3
	4	∞	∞	∞	8	0	4
	5	∞	∞	∞	3	4	0

- (두 번째 루프) 중간 노드 v가 노드 1인 경우
  - $A^1[0][2] = 4 > A^0[0][1] + A^0[1][2]=3$



G4

		도착 노드					
		0	1	2	3	4	5
시작정점	0	0	1	4⇒3	∞	∞	∞
	1	1	0	2	∞	∞	∞
	2	4⇒3	2	0	1	∞	∞
	3	∞	∞	1	0	8	3
	4	∞	∞	∞	8	0	4
	5	∞	∞	∞	3	4	0

이번 장에서는

- 그래프의 개념
- 그래프 추상 자료형
- 그래프 구현
- 그래프 탐색
- 신장 트리와 최소 비용 신장 트리
- 최단 경로