자료구조(Data Structures)

10장. 그래프(Graph)

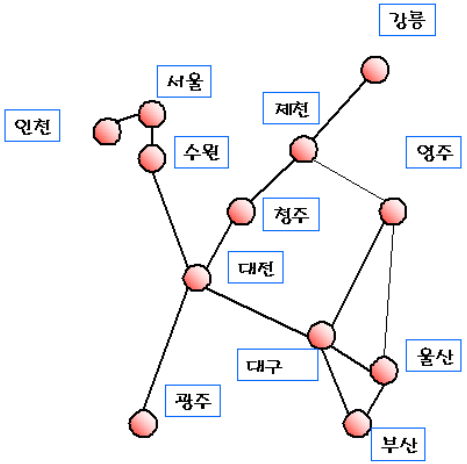
담당 교수 : 조 미경

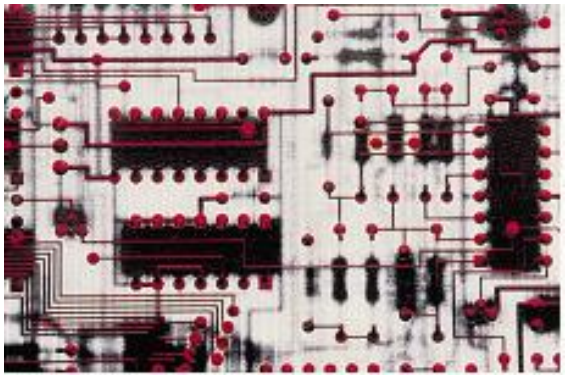
이번 장에서 학습할 내용

* 그래프란?
* 그래프 추상 데이터형
* 그래프 표현(인접행렬, 인접리스트)
* 그래프 탐색(BFS, DFS)
* 최소비용신장트리
* 최단경로

그래프(graph)

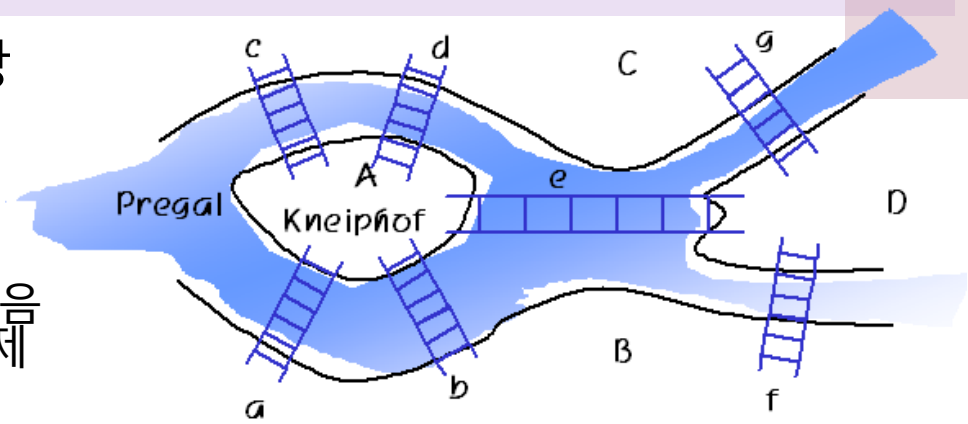
- 연결되어 있는 객체 간의 관계를 표현하는 자료구조
- 가장 일반적인 자료구조 형태
  - 우리가 배운 트리(tree)도 그래프의 특수한 경우임
  - 전기회로의 소자 간 연결 상태
  - 운영체제의 프로세스와 자원 관계
  - 큰 프로젝트에서 작은 프로젝트 간의 우선 순위
  - 지도에서 도시들의 연결 상태



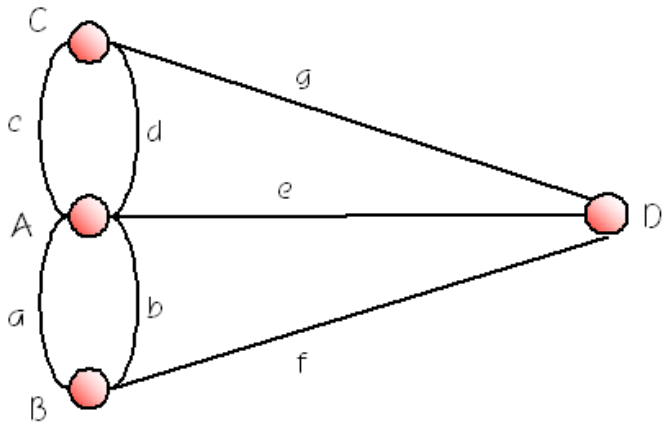


그래프 역사

- 1800년대 오일러에 의하여 창안
- 오일러 문제
  - 모든 다리를 한번만 건너서 처음 출발했던 장소로 돌아오는 문제
- A,B,C,D 지역의 연결 관계 표현
  - 위치: 정점(node)
  - 다리: 간선(edge)
- 오일러 정리
  - 모든 정점에 연결된 간선의 수가 짝수이면 오일러 경로 존재함
  - 따라서 그래프 (b)에는 오일러 경로가 존재하지 않음



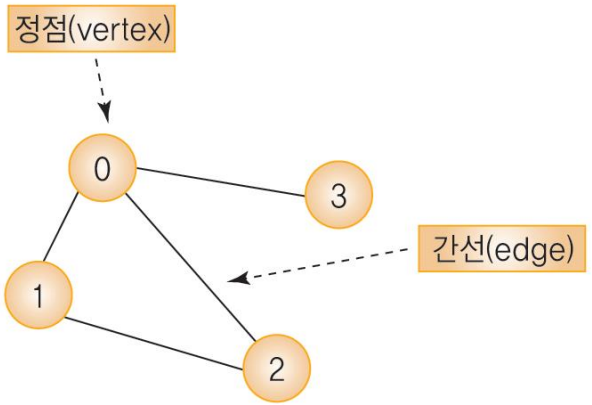
(a) 모든 다리를 한번만 건너 돌아오는 경로 문제



(b) 문제 (a)의 그래프 표현

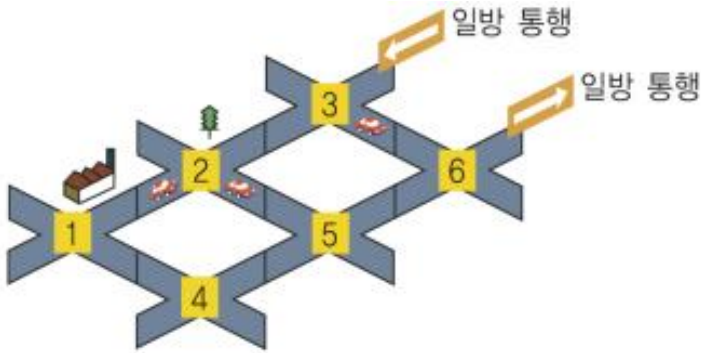
그래프 정의

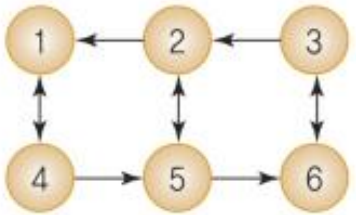
- 그래프 G는 (V, E)로 표시
- 정점(vertices)
  - 여러 가지 특성을 가질 수 있는 객체 의미
  - V(G) : 그래프 G의 정점들의 집합
  - 노드(node)라고도 불림
- 간선(edge)
  - 정점들 간의 관계 의미
  - E(G) : 그래프 G의 간선들의 집합
  - 링크(link)라고도 불림



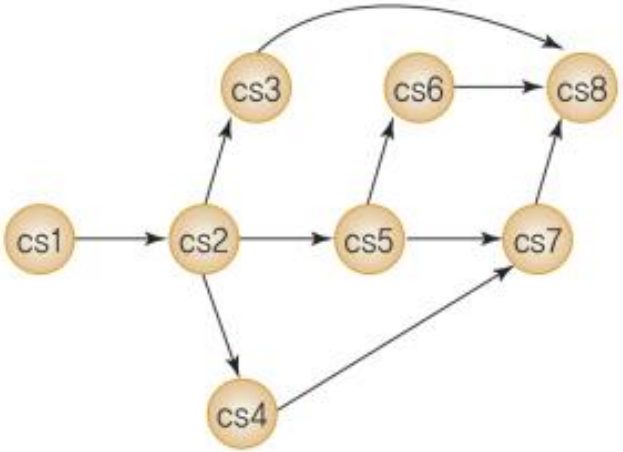
그래프로 표현하는 것들

- 도로망



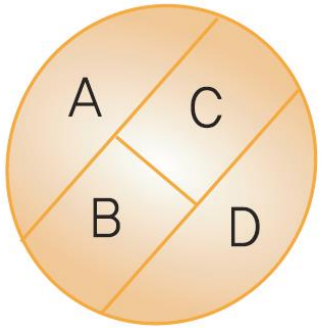


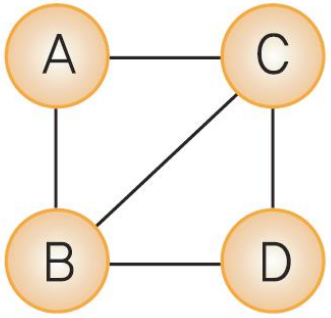
- 선수과목 관계



그래프로 표현하는 것들

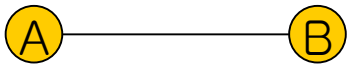
- 영역 간 인접 관계



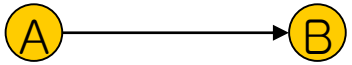


그래프의 종류

- 무방향 그래프(undirected graph)
  - 무방향 간선(undirected edge)만 사용
  - 간선을 통해서 양방향으로 갈수 있음
  - 도로의 왕복통행 길
  - (A, B)와 같이 정점의 쌍으로 표현
  - (A, B) = (B, A)

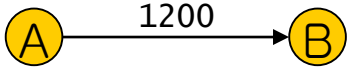


- 방향 그래프(directed graph)
  - 방향 간선(undirected edge)만 사용
  - 간선을 통해서 한쪽 방향으로만 갈 수 있음
  - 도로의 일방통행 길
  - <A, B> 와 같이 정점의 쌍으로 표현
  - <A, B> ≠ <B, A>

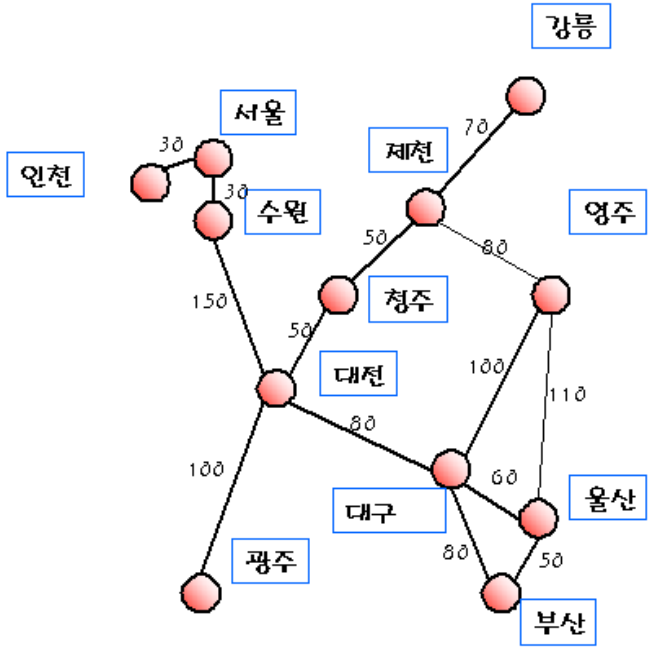


가중치 그래프

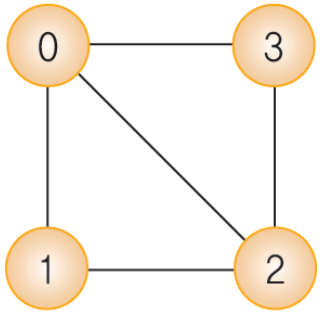
- 가중치 그래프(weighted graph)는 네트워크(network)라고도 함
- 간선에 비용(cost)이나 가중치(weight)가 할당된 그래프



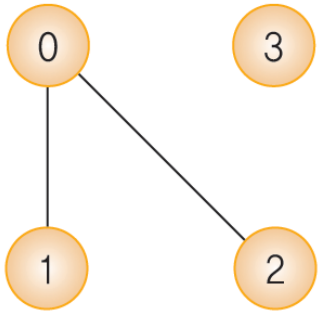
- 가중치 그래프 예
  - 정점 : 각 도시를 의미
  - 간선 : 도시를 연결하는 도로 의미
  - 가중치 : 도로의 길이



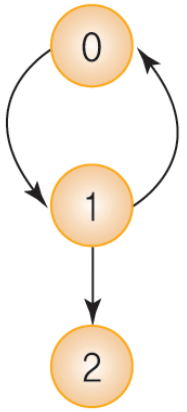
그래프 표현의 예



G1



G2



G3

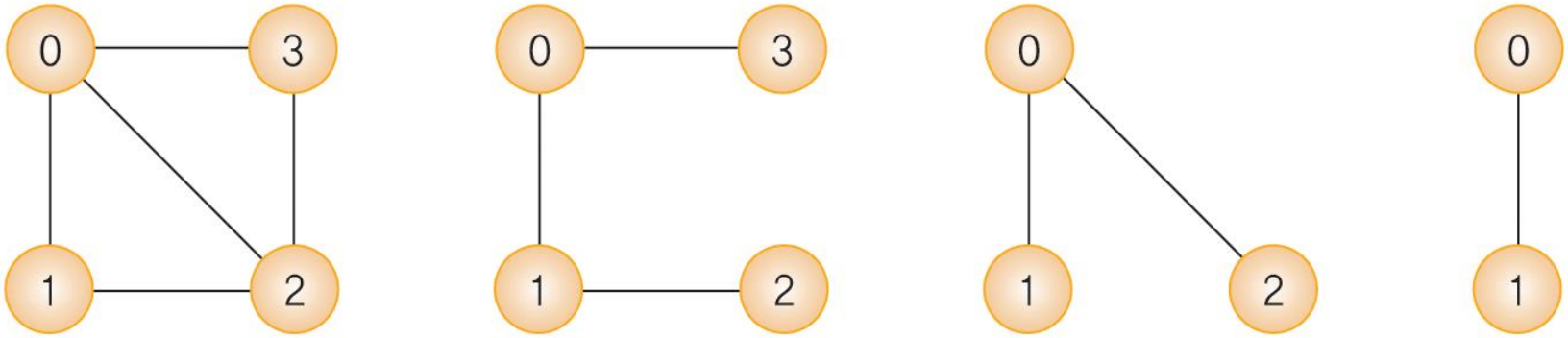
V(G1)= {0, 1, 2, 3}, E(G1)= {(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)}

V(G2)= {0, 1, 2, 3}, E(G3)= {(0, 1), (0, 2))}

V(G2)= {0, 1, 2}, E(G2)= {<0, 1>, <1, 0>, <1, 2>}

부분 그래프(subgraph)

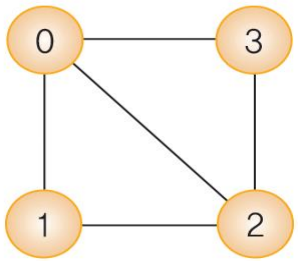
- 정점 집합 V(G)와 간선 집합 E(G)의 부분 집합으로 이루어진 그래프
- 그래프 G1의 부분 그래프들



그래프

- **인접 정점**(adjacent vertex)
  - 하나의 정점에서 간선에 의해 직접 연결된 정점
  - G1에서 정점 0의 인접 정점: 정점 1, 정점 2, 정점 3

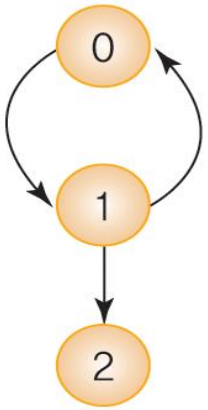
- **무방향 그래프의 차수**(degree)
  - 하나의 정점에 연결된 다른 정점의 수
  - G1에서 정점 0의 차수: 3
  - 무방향 그래프의 모든 차수의 합은 간선 수의 2배
    - G1의 차수의 합: 10
    - G1의 간선의 합: 5



G1

그래프

- **방향 그래프의 차수**(degree)
  - 진입 차수(in-degree) : 외부에서 오는 간선의 수
  - 진출 차수(out-degree) : 외부로 향하는 간선의 수
  - G3에서 정점 1의 차수: 내차수 1, 외차수 2
  - 방향 그래프의 모든 진입(진출) 차수의 합은 간선의 수
    - G3의 진입 차수의 합: 3
    - G3의 진입 차수의 합: 3
    - G3의 간선 합: 3



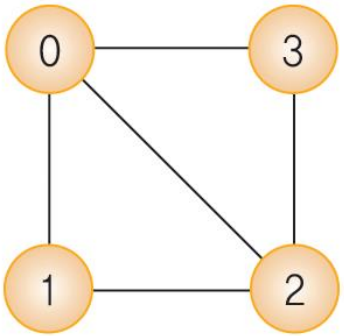
G3

그래프의 경로(path)

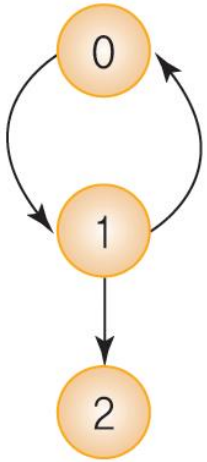
- 무방향 그래프의 정점 s로부터 정점 e까지의 경로
  - 정점의 나열 s, v1, v2, ..., vk, e
  - 나열된 정점들 간에 반드시 간선 (s, v1), (v1, v2), ... , (vk, e) 존재
- 방향 그래프의 정점 s로부터 정점 e까지의 경로
  - 정점의 나열 s, v1, v2, ..., vk, e
  - 나열된 정점들 간에 반드시 간선 <s, v1>, <v1, v2>, ... ,<vk, e> 존재
- 경로의 길이(length)
  - 경로를 구성하는데 사용된 간선의 수
- 단순 경로(simple path)
  - 경로 중에서 반복되는 간선이 없는 경로
- 사이클(cycle)
  - 단순 경로의 시작 정점과 종료 정점이 동일한 경로

그래프의 경로(path)

- G1의 0, 1, 2,3은 경로지만 0, 1, 3, 2는 경로 아님
- G1의1, 0, 2, 3은 단순경로이지만 1, 0, 2, 0은 단순경로 아님
- G1의 0, 1, 2, 0과 G3의 0, 1, 0은 사이클



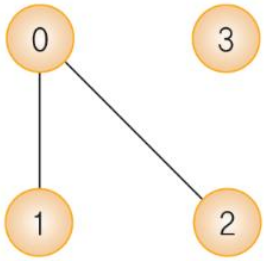
G1



G3

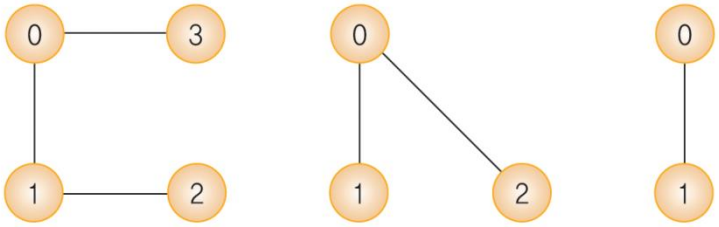
그래프의 연결정도

- 연결 그래프(connected graph)
  - 무방향 그래프 G에 있는 모든 정점쌍에 대하여 항상 경로 존재
  - G2는 비연결 그래프임



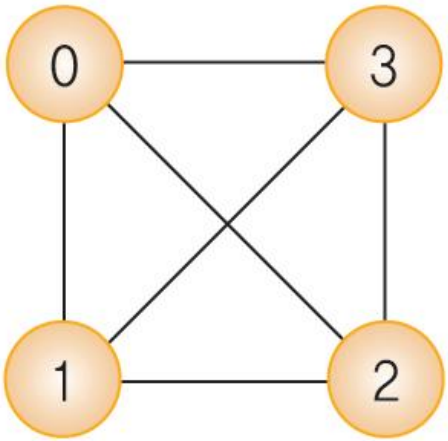
G2

- 트리(tree)
  - 그래프의 특수한 형태로서 사이클을 가지지 않는 연결 그래프
  - 트리의 예



그래프의 연결정도

- 완전 그래프(complete graph)
  - 모든 정점이 연결되어 있는 그래프
  - n개의 정점을 가진 무방향 완전그래프의 간선의 수: $n \times (n-1)/2$
  - n=4, 간선의 수 = $(4 \times 3)/2 = 6$



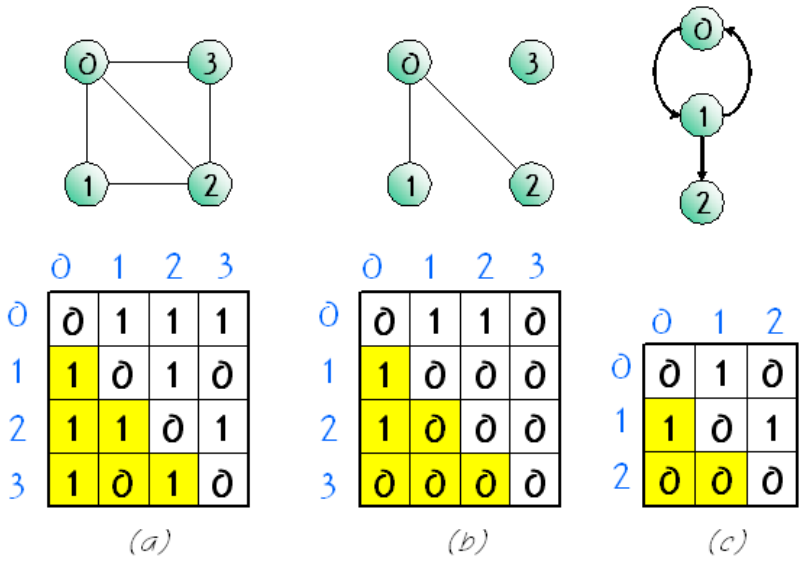
그래프 ADT

·객체: 정점의 집합과 간선의 집합
·연산:
- create_graph() ::= 그래프를 생성한다.
- init(g) ::= 그래프 g를 초기화한다.
- insert_vertex(g,v) ::= 그래프 g에 정점 v를 삽입한다.
- insert_edge(g,u,v) ::= 그래프 g에 간선 (u,v)를 삽입한다.
- delete_vertex(g,v) ::= 그래프 g의 정점 v를 삭제한다.
- delete_edge(g,u,v) ::= 그래프 g의 간선 (u,v)를 삭제한다.
- is_empty(g) ::= 그래프 g가 공백 상태인지 확인한다.
- adjacent(v) ::= 정점 v에 인접한 정점들의 리스트를 반환한다.
- destroy_graph(g) ::= 그래프 g를 제거한다.

- 그래프에 정점을 추가하려면 insert_vertex() 연산 사용
- 그래프에 간선을 추가하려면 insert_edge() 연산 사용

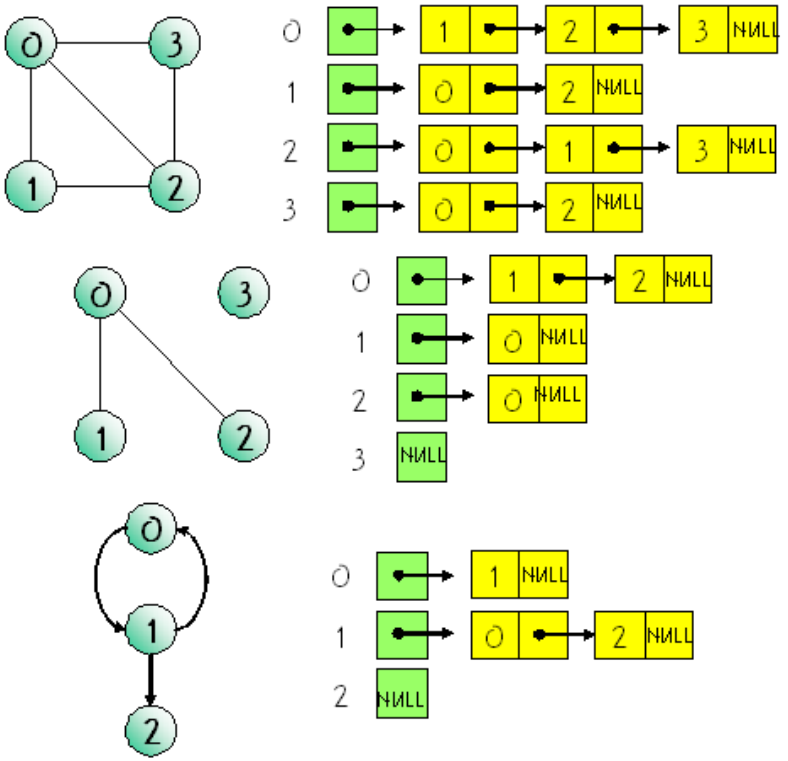
그래프 표현 방법

- 인접행렬 (adjacent matrix) 방법
  if(간선 (i, j)가 그래프에 존재) M[i][j] = 1,
  그렇지않으면 M[i][j] = 0.
- 인접 행렬의 대각선 성분은 모두 0(자체 간선 불허)
- 무방향 그래프의 인접 행렬은 대칭



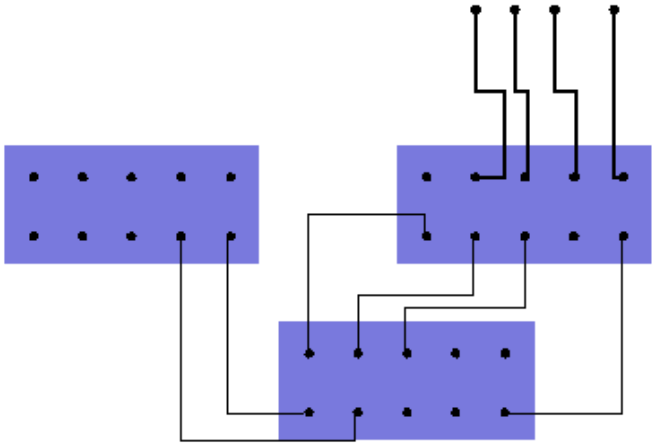
그래프 표현 방법(계속)

- 인접리스트 (adjacency list) 방법
  - 각 정점에 인접한 정점들을 연결리스트로 표현



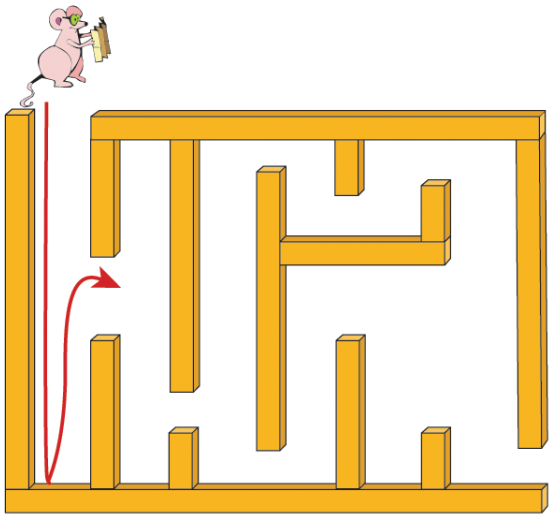
그래프 탐색

- 그래프의 가장 기본적인 연산
- 하나의 정점으로부터 시작하여 차례대로 모든 정점들을 한번씩 방문
- 많은 문제들이 단순히 그래프의 노드를 탐색하는 것으로 해결
  (예) 도로망에서 특정 도시에서 다른 도시로 갈 수 있는지 여부
  (예) 전자회로에서 특정 단자와 다른 단자가 서로 연결되어 있는지 여부



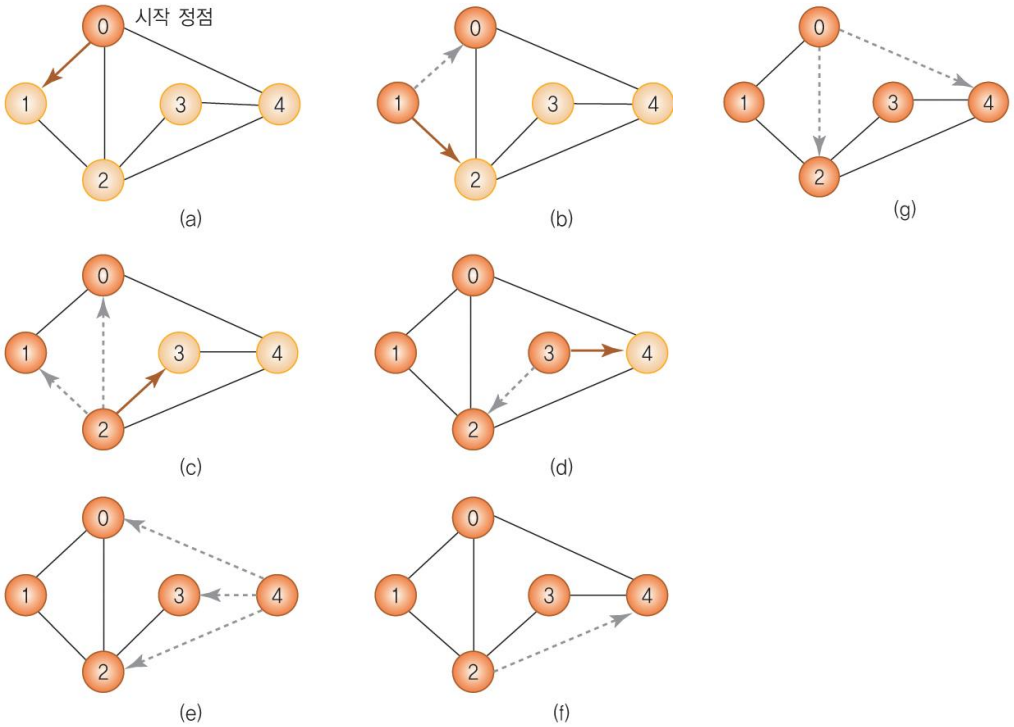
깊이 우선 탐색(DFS)

- **깊이 우선 탐색** (DFS: depth-first search)
  - 한 방향으로 갈 수 있을 때까지 가다가 더 이상 갈 수 없게 되면 가장 가까운 갈림길로 돌아와서 이 곳으로부터 다른 방향으로 다시 탐색 진행
  - 되돌아가기 위해서는 스택 필요(순환함수 호출로 묵시적인 스택 이용 가능)



DFS 알고리즘

```
depth_first_search(v)
        v를 방문되었다고 표시;
        for all u ∈ (v에 인접한 정점) do
          if (u가 아직 방문되지 않았으면)then depth_first_search(u)
```



DFS 프로그램

```
// 인접 행렬로 표현된 그래프에 대한 깊이 우선 탐색
void dfs_mat(GraphType *g, int v)
{
   int w;
   visited[v] = TRUE;                    // 정점 v의 방문 표시
   printf("%d ", v);                     // 방문한 정점 출력
   for(w=0; w<g->n; w++)                                  // 인접 정점 탐색
      if( g->adj_mat[v][w] && !visited[w] ) dfs_mat(g, w);          //정점 w에서 DFS 새로시작
}
```

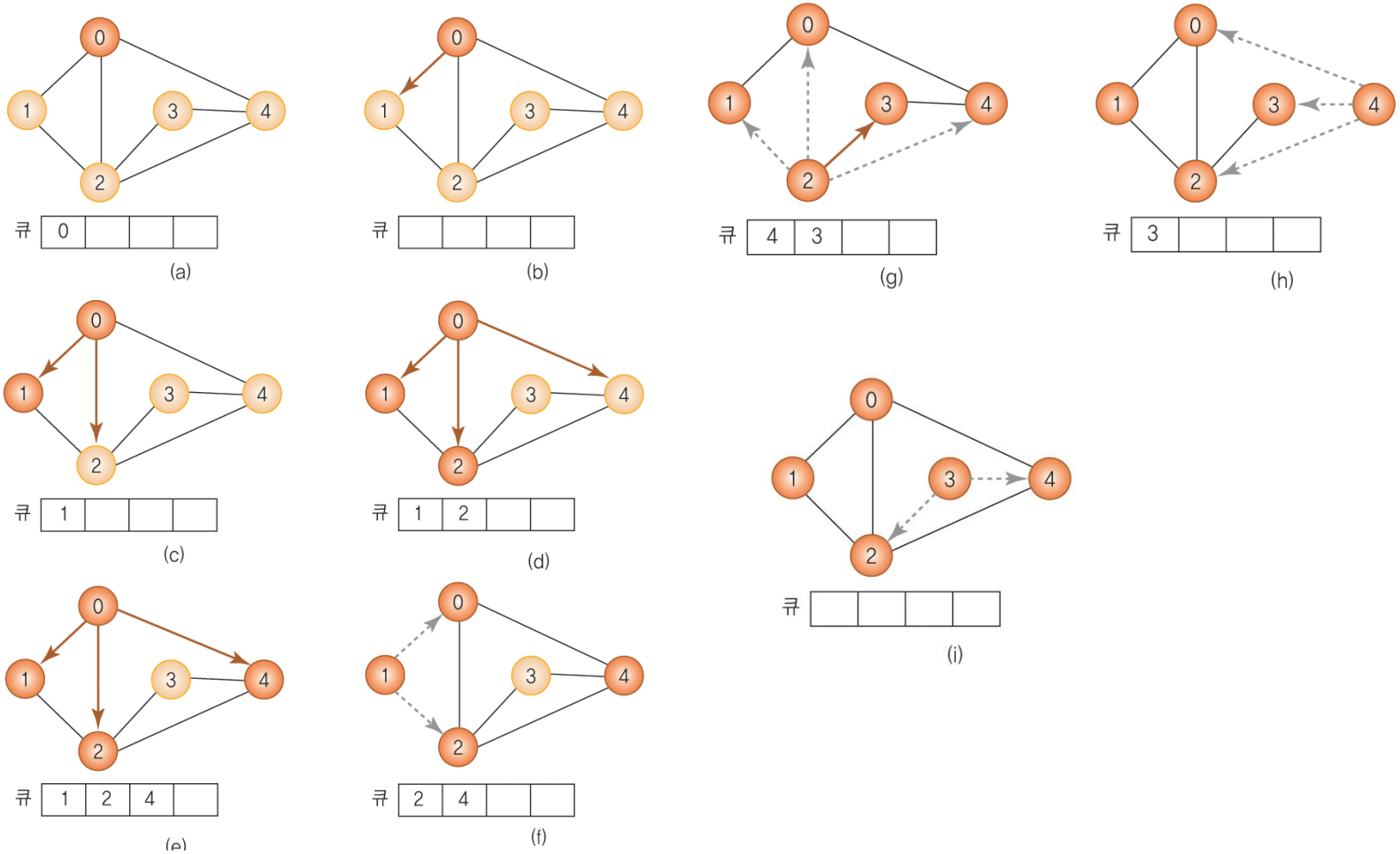
```
// 인접 리스트로 표현된 그래프에 대한 깊이 우선 탐색
void dfs_list(GraphType *g, int v)
{
   GraphNode *w;
   visited[v] = TRUE;                    // 정점 v의 방문 표시
   printf("%d ", v);                     // 방문한 정점 출력
   for(w=g->adj_list[v]; w; w=w->link)   // 인접 정점 탐색
      if(!visited[w->vertex]) dfs_list(g, w->vertex); //정점 w에서 DFS 새로시작
}
```

너비 우선 탐색(BFS)

- **너비 우선 탐색(BFS: breadth-first search)**
  - 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법
  - 큐를 사용하여 구현됨
- 너비우선탐색 알고리즘

```
breadth_first_search(v)
v를 방문되었다고 표시;
큐 Q에 정점 v를 삽입;
while (not is_empty(Q)) do
        Q에서 정점 w를 삭제;
        for all u ∈ (w에 인접한 정점) do
                if (u가 아직 방문되지 않았으면) then
                        u를 큐에 삽입;
                        u를 방문되었다고 표시;
```

너비우선 탐색(BFS)



BFS 프로그램(인접행렬)

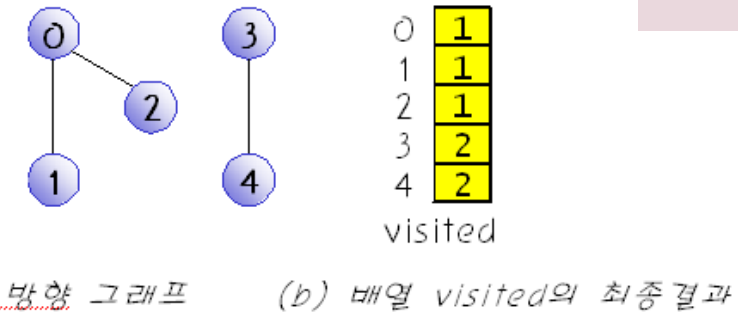
```
void bfs_mat(GraphType *g, int v)
{
        int w;
        QueueType q;
        init(&q);                               // 큐 초기화
        visited[v] = TRUE;      // 정점 v 방문 표시
        printf("%d ", v);       // 정점 출력
        enqueue(&q, v);                         // 시작 정점을 큐에 저장
        while(!is_empty(&q)){
        v = dequeue(&q);                        // 큐에 정점 추출
        for(w=0; w<g->n; w++)                   // 인접 정점 탐색
                if(g->adj_mat[v][w] && !visited[w]){
                        visited[w] = TRUE;   // 방문 표시
                        printf("%d ", w);               // 정점 출력
                        enqueue(&q, w);         // 방문한 정점을 큐에 저장
                }
        }
}
```

BFS 프로그램(인접리스트)

```
void bfs_list(GraphType *g, int v)
{
        GraphNode *w;
        QueueType q;
        init(&q);                   // 큐 초기화
        visited[v] = TRUE;          // 정점 v 방문 표시
        printf("%d ", v);           // 정점 v 출력
        enqueue(&q, v);                     // 시작정점을 큐에 저장
        while(!is_empty(&q)){
                v = dequeue(&q);                                    // 큐에서 정점 추출
                for(w=g->adj_list[v]; w; w = w->link) //인접 정점 탐색
                        if(!visited[w->vertex]){     // 미방문 정점 탐색
                        visited[w->vertex] = TRUE;   // 방문 표시
                        printf("%d ", w->vertex);    // 정점 출력
                        enqueue(&q, w->vertex); // 방문한 정점을 큐에 삽입
                }
        }
}
```

연결 성분

- 최대로 연결된 부분 그래프들
- DFS 또는 BFS 반복 이용
  - DFS 또는 BFS 탐색 프로그램의 visited[v]=TRUE; 를 visited[v]=count; 로 교체



(a) 무방향 그래프 (b) 배열 visited의 최종결과

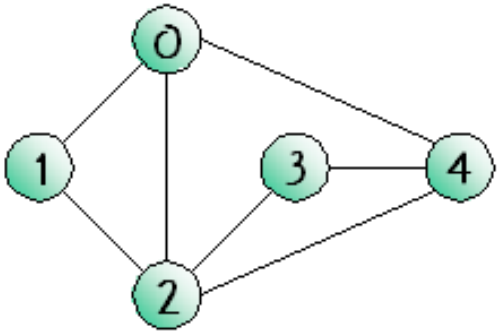
```
void find_connected_component(GraphType *g)
{           int i;
            count = 0;
            for(i=0; i<g->n; i++)
                        if(!visited[i]){    // 방문되지 않았으면
                                    count++;
                                    dfs_mat(g, i);
                        }
}
```

신장 트리(spanning tree)

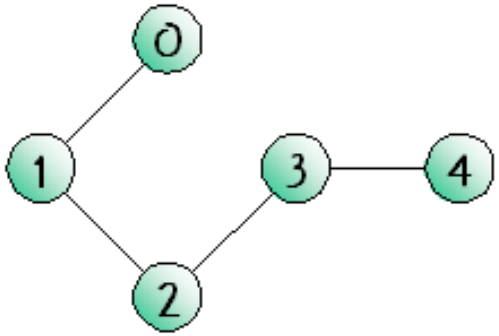
- 그래프내의 모든 정점을 포함하는 트리
- 모든 정점들이 연결되어 있어야 하고 사이클을 포함해서는 안됨
- n개의 정점을 가지는 그래프의 신장트리는 n-1개의 간선을 가짐
- 최소의 링크를 사용하는 네트워크 구축 시 사용
  - 통신망, 도로망, 유통망 등
- 신장트리 알고리즘

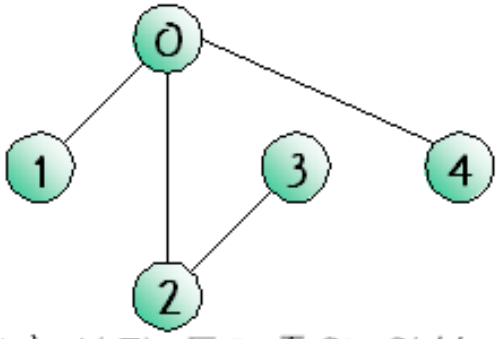
```
depth_first_search(v)

v를 방문되었다고 표시;
for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면)  then
                (v,u)를 신장트리 간선이라고 표시;
                depth_first_search(u);
```

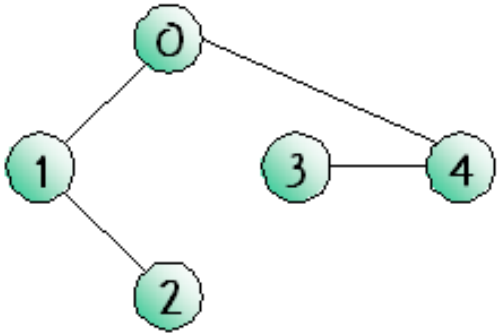
신장 트리



(a) 연결 그래프



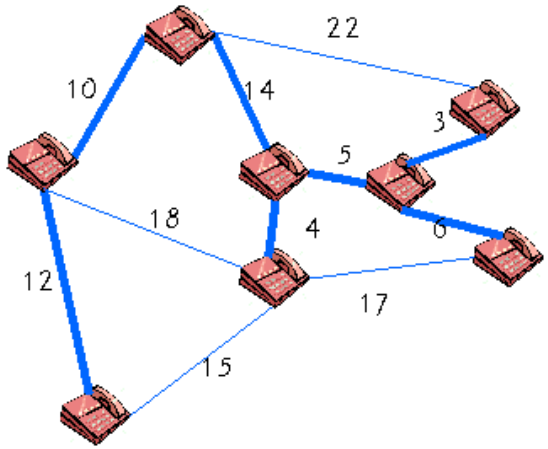




(b) 신장 트리중의 일부

최소비용 신장트리 (MST: minimum spanning tree)

- 네트워크에 있는 모든 정점들을 가장 적은 수의 간선과 비용으로 연결
- MST의 응용
  - 도로 건설 - 도시들을 모두 연결하면서 도로의 길이를 최소가 되도록 하는 문제
  - 전기 회로 - 단자들을 모두 연결하면서 전선의 길이를 가장 최소로 하는 문제
  - 통신 - 전화선의 길이가 최소가 되도록 전화 케이블 망을 구성하는 문제
  - 배관 - 파이프를 모두 연결하면서 파이프의 총 길이를 최소로 하는 문제



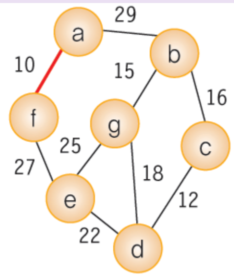
Kruskal의 MST 알고리즘

- 탐욕적인 방법(greedy method)
  - 주요 알고리즘 설계 기법
  - 각 단계에서 최선의 답을 선택하는 과정을 반복함으로써 최종적인 해답에 도달
  - 탐욕적인 방법은 항상 최적의 해답을 주는지 검증 필요
  - Kruskal MST 알고리즘은 최적의 해답임이 증명됨

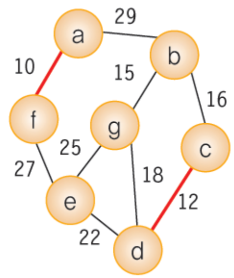
Kruskal의 MST 알고리즘

- MST는 최소 비용의 간선으로 구성됨과 동시에 사이클을 포함하지 않아야 함
- 각 단계에서 사이클을 이루지 않는 최소 비용 간선 선택
  - 그래프의 간선들을 가중치의 오름차순으로 정렬
  - 정렬된 간선 중에서 사이클을 형성하지 않는 간선을 현재의 MST 집합에 추가
  - 만약 사이클을 형성하면 그 간선은 제외

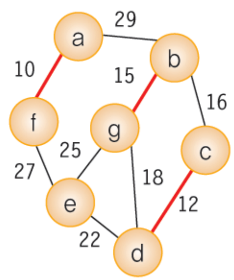
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



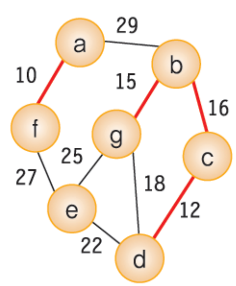
af	**cd**	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



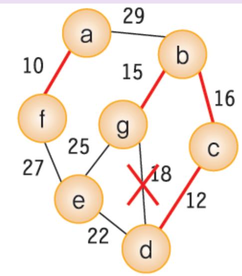
af	cd	**bg**	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



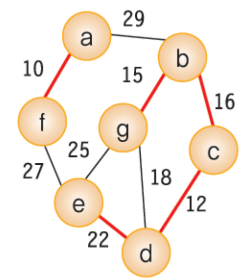
af	cd	bg	**bc**	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



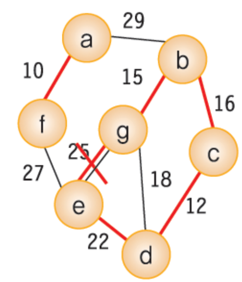
af	cd	bg	bc	**dg**	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



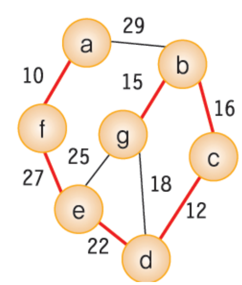
af	cd	bg	bc	dg	**de**	eg	ef	ab
10	12	15	16	18	22	25	27	29



af	cd	bg	bc	dg	de	**eg**	ef	ab
10	12	15	16	18	22	25	27	29

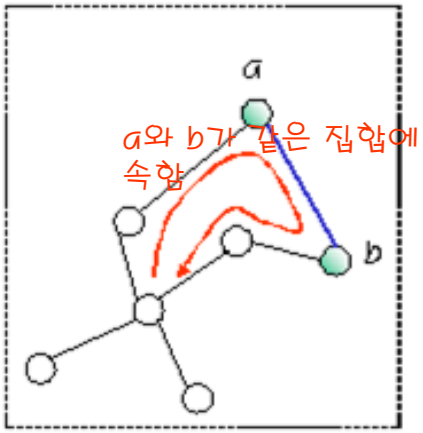


af	cd	bg	bc	dg	de	eg	**ef**	ab
10	12	15	16	18	22	25	27	29

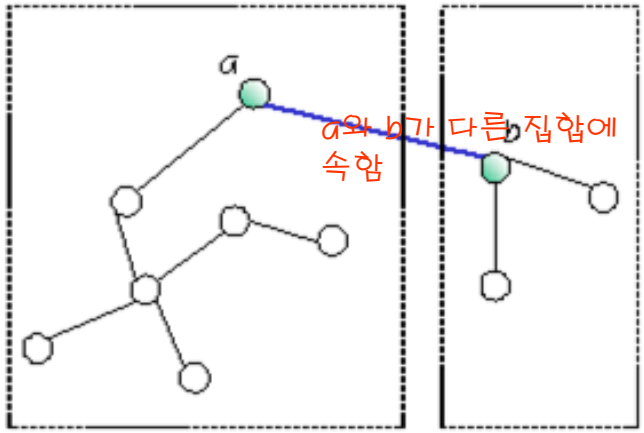


Kruskal의 MST 알고리즘

- union-find 알고리즘
  - 두 집합들의 합집합 만듬
  - 원소가 어떤 집합에 속하는지 알아냄
  - Kruskal의 MST 알고리즘에서 사이클 검사에 사용



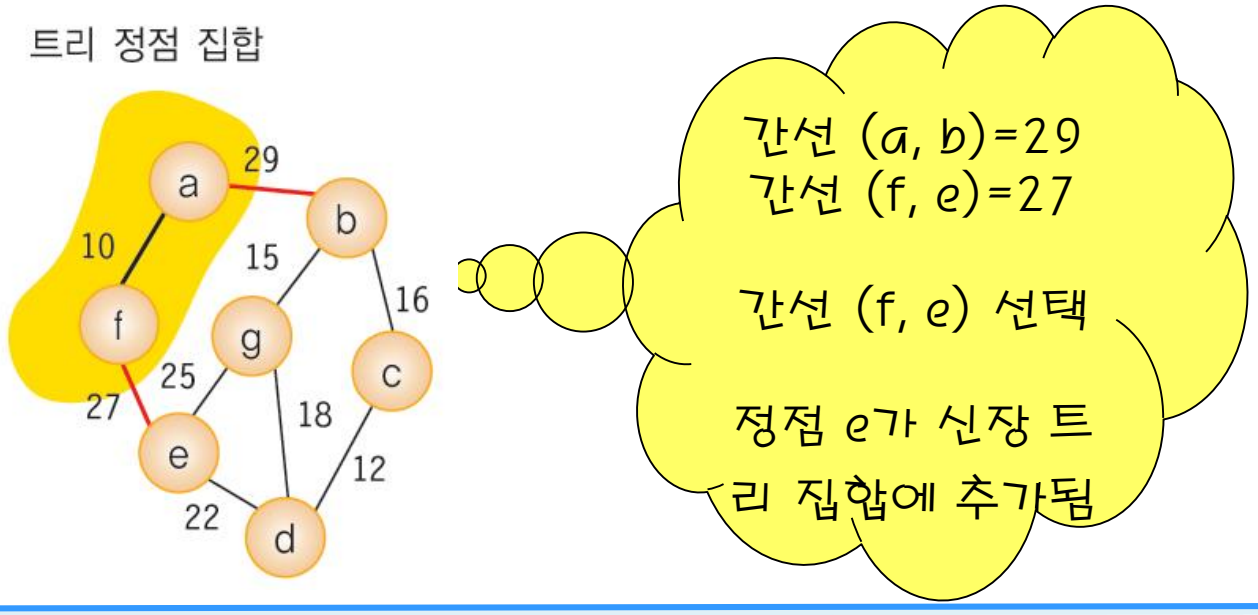
(a) 사이클 형성

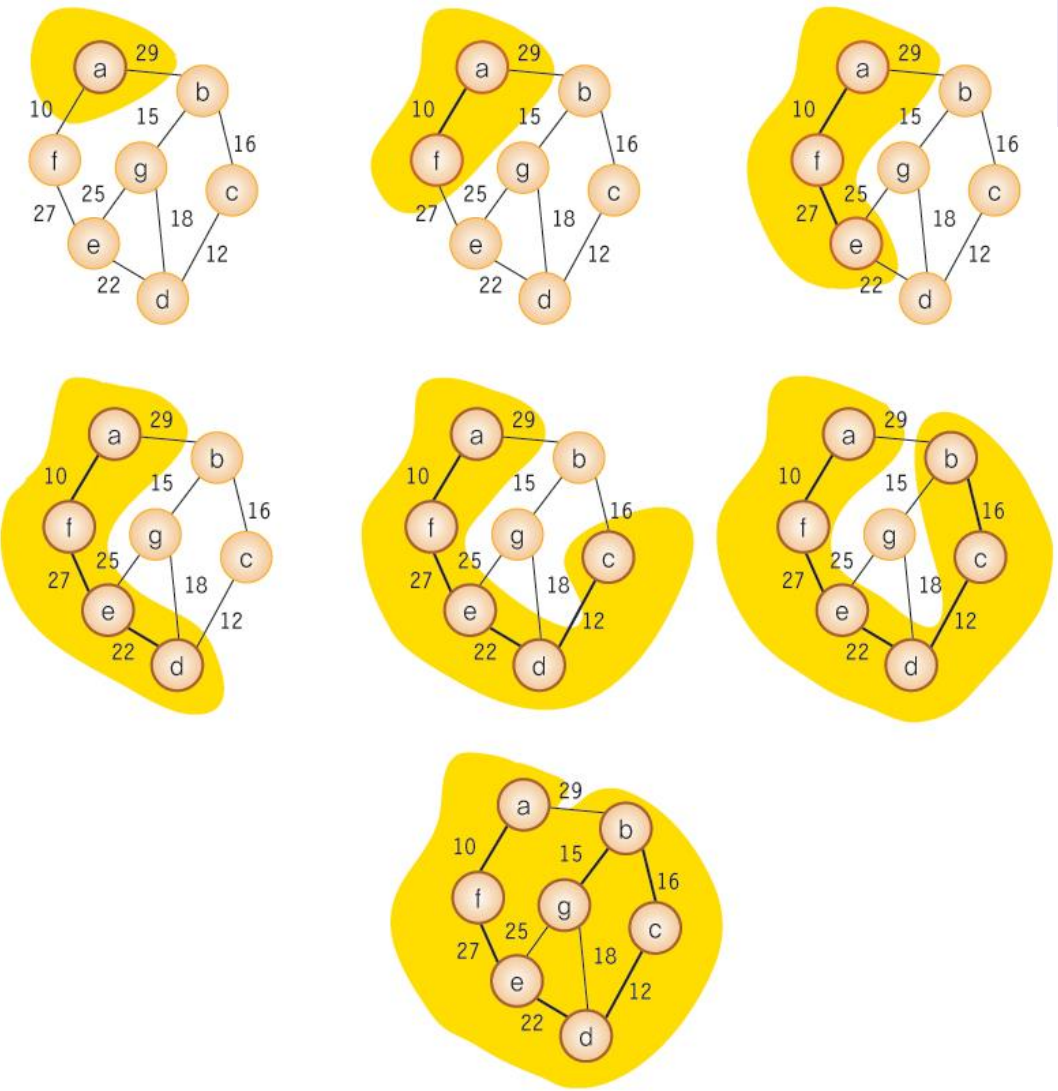


(b) 사이클 형성되지 않음

Prim의 MST 알고리즘

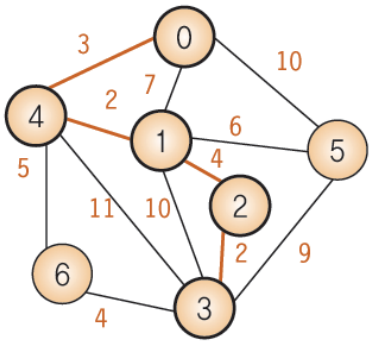
- 시작 정점에서부터 출발하여 신장 트리 집합을 단계적으로 확장해나감
  - 시작 단계에서는 시작 정점만이 신장 트리 집합에 포함됨
- 신장 트리 집합에 인접한 정점 중에서 최저 간선으로 연결된 정점 선택하여 신장 트리 집합에 추가함
- 이 과정은 신장 트리 집합이 n-1개의 간선을 가질 때까지 반복




a
b
c
d
e
f
g
29
10
15
16
27
25
18
12
22

최단 경로(shortest path)

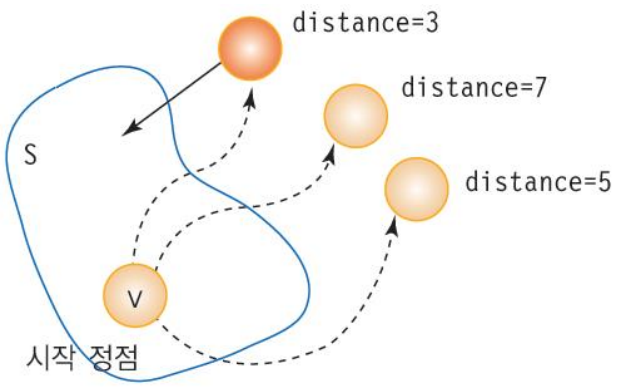
- 네트워크에서 정점 u와 정점 v를 연결하는 경로 중에서 간선들의 가중치 합이 최소가 되는 경로
- 간선의 가중치는 비용, 거리, 시간 등
- 정점 0에서 정점 3으로 가는 최단 경로 문제
  - 인접행렬에서 간선이 없는 노드쌍의 가중치는 ∞ 임
  - 0,4,1,2,3이 최단 경로
  - 최단경로 길이는 3+2+4+2=11



	0	1	2	3	4	5	6
0	0	7	∞	∞	3	10	∞
1	7	0	4	10	2	6	∞
2	∞	4	0	2	∞	∞	∞
3	∞	10	2	0	11	9	4
4	3	2	∞	11	0	∞	5
5	10	6	∞	9	∞	0	∞
6	∞	∞	∞	4	5	∞	0

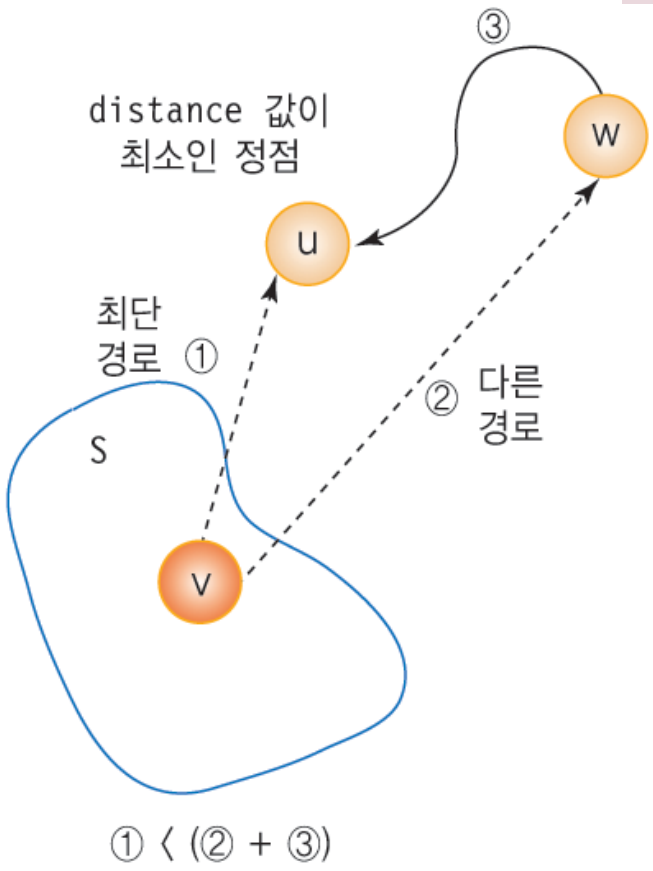
Dijkstra의 최단경로 알고리즘

- 하나의 시작 정점으로부터 모든 다른 정점까지의 최단 경로 찾음
- 집합 S
  - 시작 정점 v로부터의 최단경로가 이미 발견된 정점들의 집합
- distance 배열
  - 최단경로가 알려진 정점들만을 이용한 다른 정점들까지의 최단경로 길이
  - distance 배열의 초기값(시작 정점 v)
    - distance[v] = 0
    - 다른 정점에 대한 distance 값은 시작정점과 해당 정점간의 가중치 값
- 매 단계에서 가장 distance 값이 작은 정점을 S에 추가



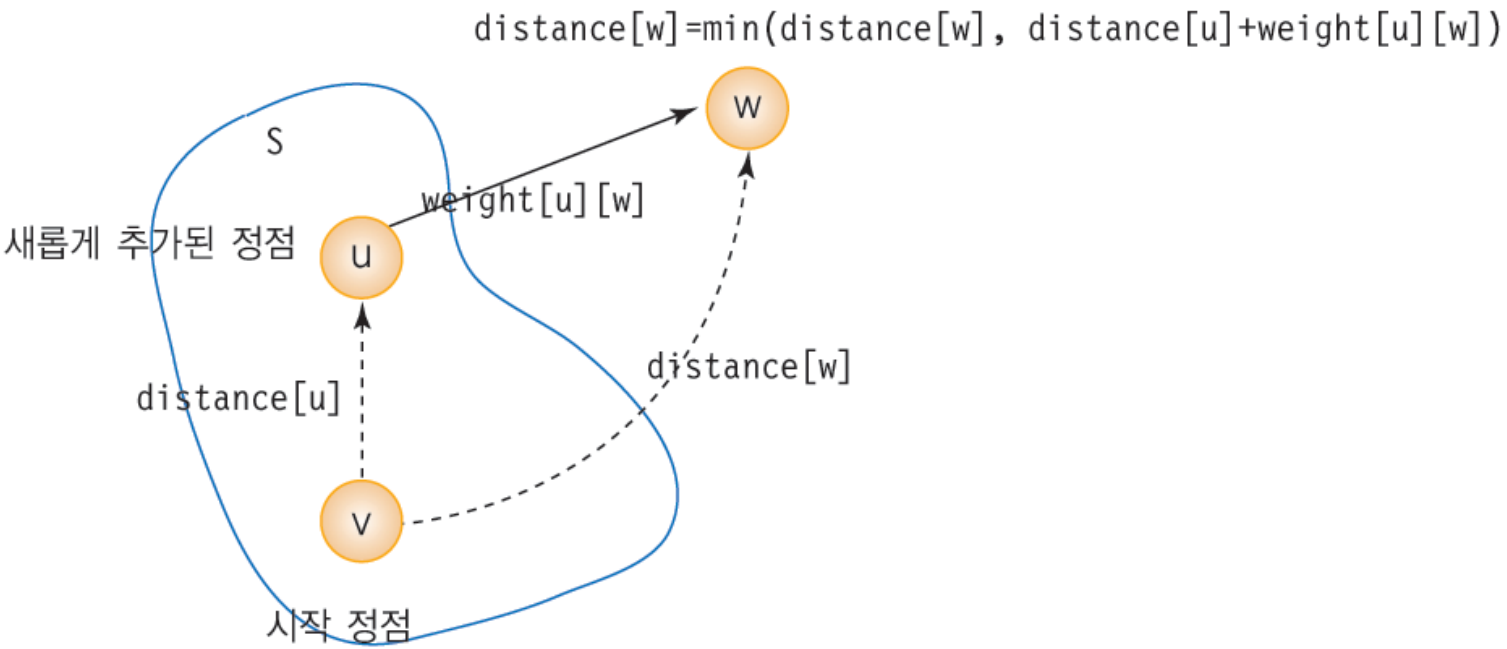
Dijkstra의 최단경로 알고리즘

- distance 값이 가장 작은 정점을 u라고 하자. 그러면 시작 정점 v에서 정점 u까지의 최단거리는 경로 ①이 된다.
- 정점 w를 거쳐서 정점 u로 가는 가상적인 더 짧은 경로가 있다고 가정해보자. 그러면 정점 v에서 정점 u까지의 거리는 정점 v에서 정점 w까지의 거리 ②와 정점 w에서 정점 u로 가는 거리③을 합한 값이 된다.
- 그러나 경로 ②는 경로 ①보다 항상 길 수 밖에 없다. 왜냐하면 현재 distance 값이 가장 작은 정점은 u이기 때문이다.
- 따라서 매 단계에서 distance 값이 가장 작은 정점들을 추가해나가면 시작 정점에서 모든 정점까지의 최단거리를 구할 수 있다.

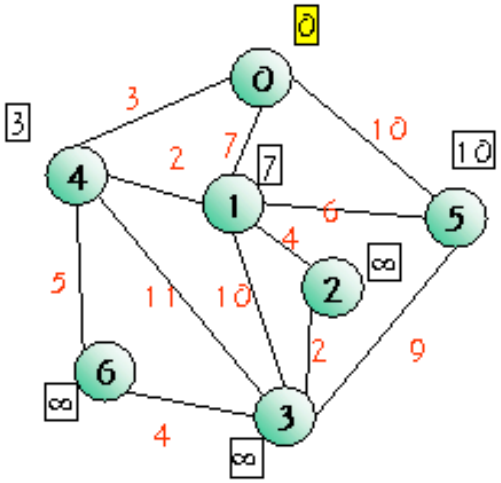


Dijkstra의 최단경로 알고리즘

- 새로운 정점이 S에 추가되면 distance값 갱신

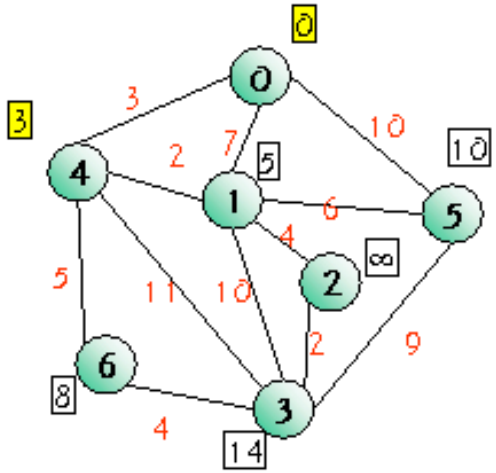


Dijkstra의 최단경로 알고리즘



S={0}

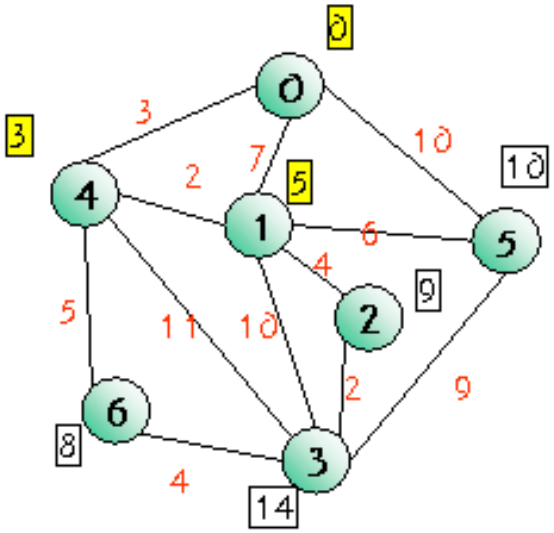
	0	1	2	3	4	5	6
distance[]=	0	7	∞	∞	3	10	∞



S={0, 4}

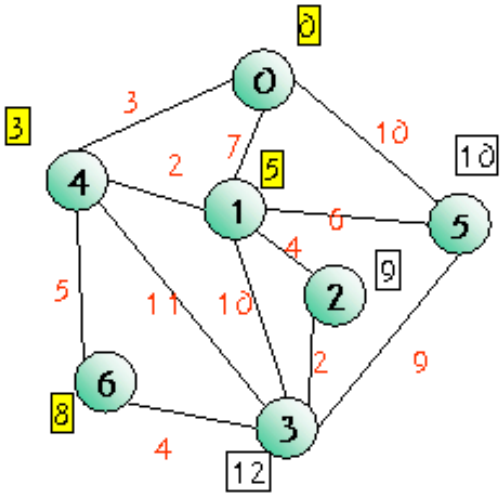
	0	1	2	3	4	5	6
distance[]=	0	5	∞	14	3	10	8

Dijkstra의 최단경로 알고리즘



S={0, 4, 1}

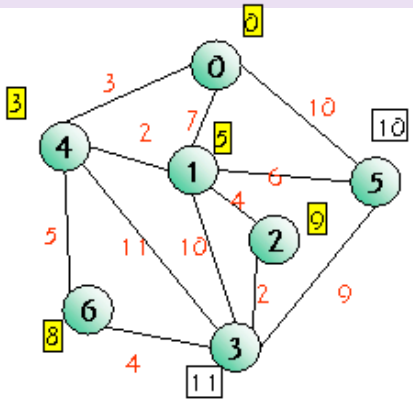
	0	1	2	3	4	5	6
distance[]=	0	5	9	14	3	10	8



S={0, 4, 1, 6}

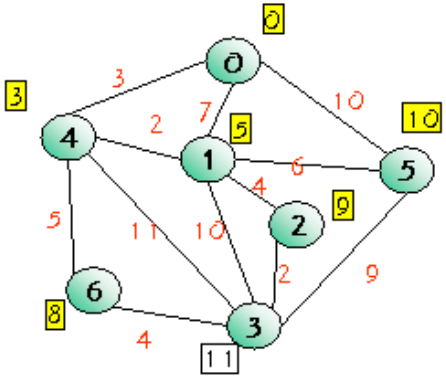
	0	1	2	3	4	5	6
distance[]=	0	5	9	12	3	10	8

Dijkstra의 최단경로 알고리즘



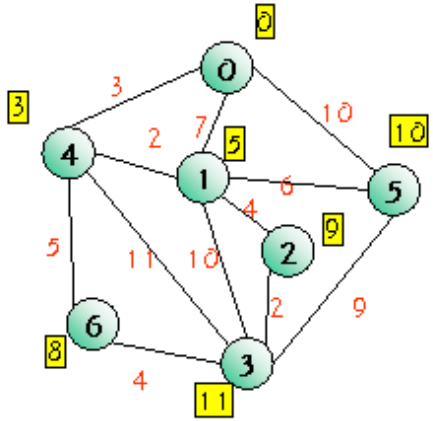
S={0, 4, 1, 6, 2}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5, 3}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8

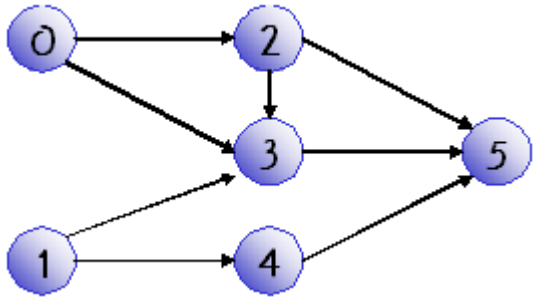
위상정렬(topological sort)

- 방향 그래프에서 간선 <u, v>가 있다면 정점 u는 정점 v를 선행함
- 위상 정렬이란 방향 그래프 정점들의 선행 순서를 위배하지 않으면서 모든 정점을 나열하는 것

위상정렬(topological sort)

- 선수 과목은 과목들의 선행 관계 표현함

과목번호	과목명	선수과목
0	컴퓨터개론	없음
1	이산수학	없음
2	C언어	0
3	자료구조	0, 1, 2
4	확률	1
5	알고리즘	2, 3, 4

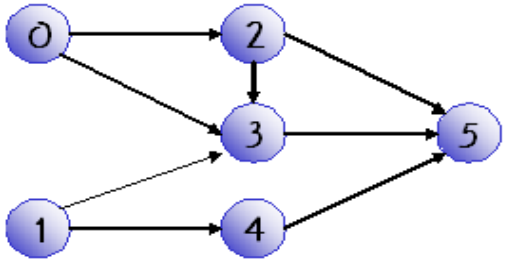


- 위상 순서(topological order)
  - (0,1,2,3,4,5) , (1,0,2,3,4,5)
- (2,0,1,3,4,5)는 위상 순서가 아님
  - 왜냐하면 2번 정점이 0번 정점 앞에 오기 때문

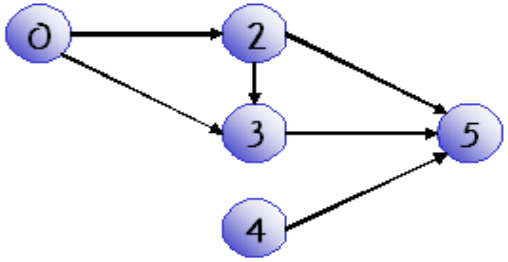
위상정렬 알고리즘

```
Input: 그래프 G=(V,E)
Output: 위상 정렬 순서

topo_sort(G)

for i←0 to n-1 do
        if( 모든 정점이 선행 정점을 가지면 )
                then 사이클이 존재하고 위상 정렬 불가;
        선행 정점을 가지지 않는 정점 v 선택;
        v를 출력;
        v와 v에서 나온 모든 간선들을 그래프에서 삭제;
```

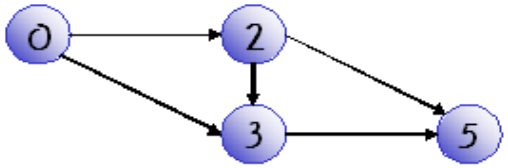
위상정렬의 예



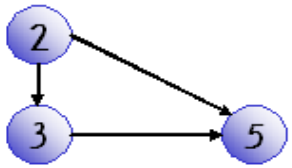
(a) 초기상태



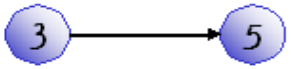
(b) 1 제거



(a) 4 제거



(b) 0 제거

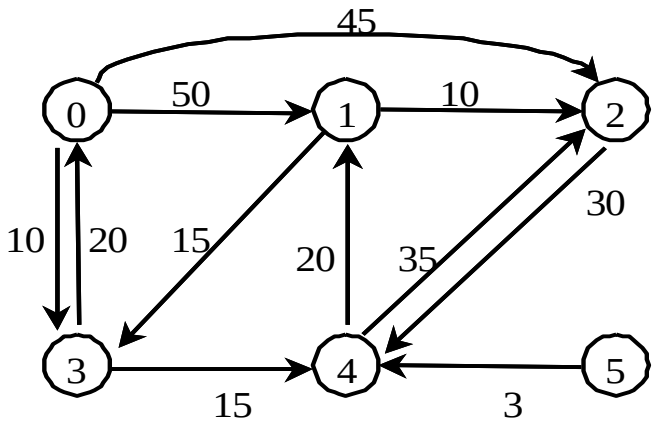


(a) 2 제거

(b) 3 제거

연습문제 1

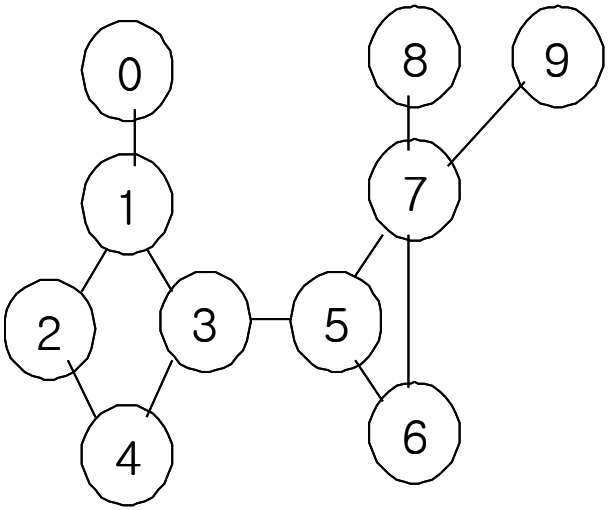
- 다음의 방향 그래프에 대하여 다음 질문에 답하시오.
  - 각 정점의 진입차수와 진출차수
  - 인접 행렬 표현
  - 인접 리스트 표현
  - 모든 사이클과 그 길이



(a) graph

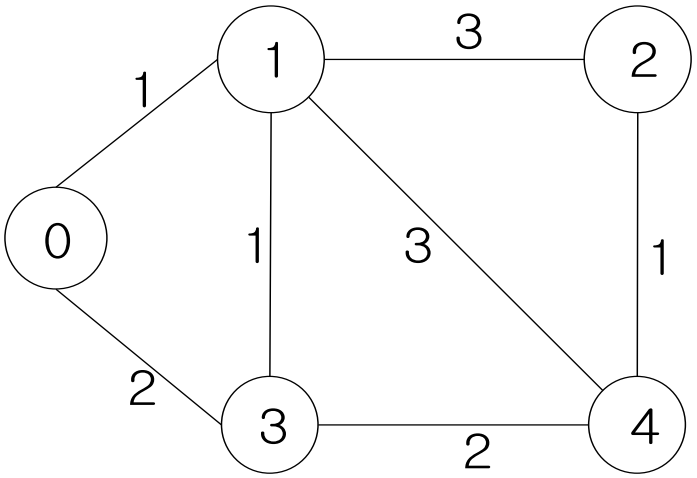
연습문제 2

- 다음 그래프에 대하여 답하시오. 그래프는 인접 행렬로 표현되어 있다고 가정한다.
  - 정점 3에서 출발하여 깊이 우선 탐색했을 경우의 방문순서
  - 정점 6에서 출발하여 깊이 우선 탐색했을 경우의 방문순서
  - 정점 3에서 출발하여 너비 우선 탐색했을 경우의 방문순서
  - 정점 6에서 출발하여 너비 우선 탐색했을 경우의 방문순서



연습문제 3

- 아래의 네트워크에 대하여 Kruskal의 MST 알고리즘을 이용해서 최소 비용 신장 트리가 구성되는 과정을 보여라.



연습문제 4

- 다음의 그래프에 대하여 위상 정렬을 적용하고 그 결과를 구하시오.