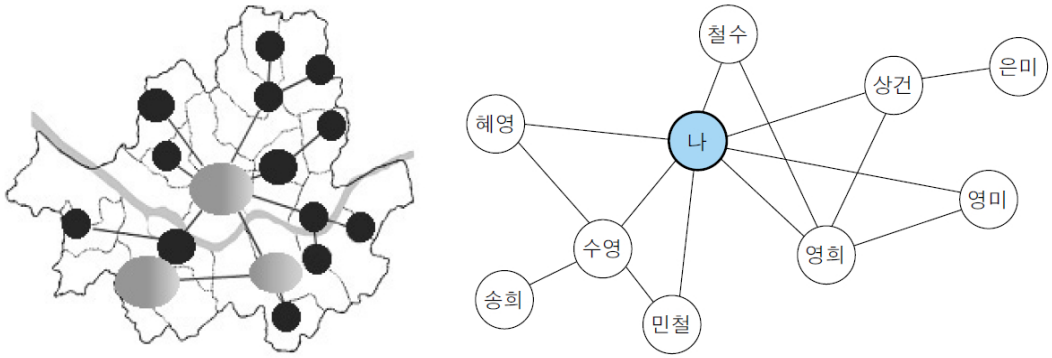
Data structure - Graph

- **그래프(graph)**
  - 선형 자료구조나 트리 자료구조로 표현하기 어려운 n:n의 관계를 가지는 원소들을 표현하기 위한 자료구조
  - 그래프 G
    - 객체를 나타내는 정점(vertex)과 객체를 연결하는 간선(edge)의 집합
    - G = (V, E)
    - V는 그래프에 있는 정점들의 집합
    - E는 정점을 연결하는 간선들의 집합



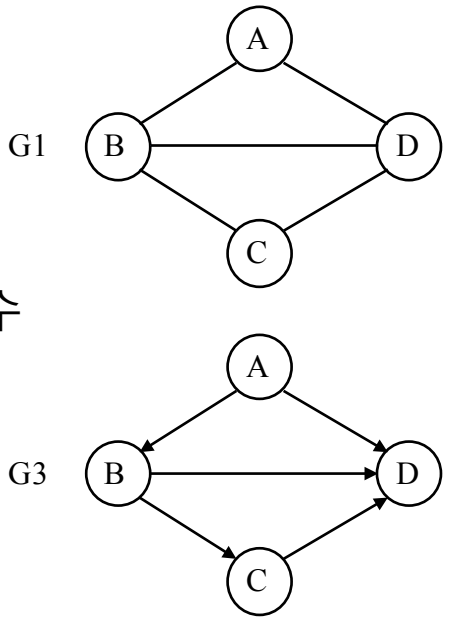
[그림 9-1] 그래프의 사용 예: 버스 노선도, 인맥 지도

Data structure - Graph

- **그래프(graph)의 종류**
  - 무방향 그래프(undirected graph)
    - 두 정점을 연결하는 간선의 방향이 없는 그래프
  - 방향 그래프(directed graph) , 다이그래프(digraph)
    - 간선이 방향을 가지고 있는 그래프
  - 완전 그래프(complete graph)
    - 각 정점에서 다른 모든 정점을 연결하여 가능한 최대의 간선 수를 가진 그래프
  - 부분 그래프(subgraph)
    - 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
  - 가중 그래프(weight graph) , 네트워크(network)
    - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프

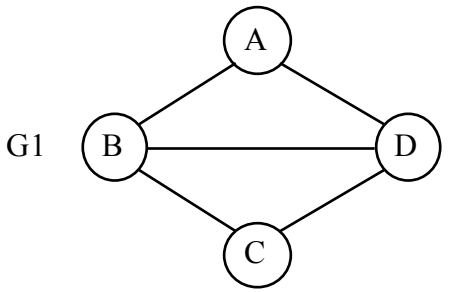
Data structure - Graph

- 그래프 관련 용어
  - 그래프에서 두 정점 Vi과 Vj를 연결하는 간선 (Vi, Vj)가 있을 때, 두 정점 Vi와 Vj를 인접(Adjacent)되어 있다고 하고, 간선 (Vi, Vj)는 정점 Vi와 Vj에 부속(Incident)되어있다.
    - 그래프G1에서 정점 A와 인접한 정점은 B와 D이고, 정점 A에 부속되어 있는 간선은 (A,B)와 (A,D)
  - 차수(Degree) – 정점에 부속되어있는 간선의 수
    - 무방향 그래프 G1에서 정점 A의 차수는 2, 정점 B의 차수는 3
    - 방향 그래프의 정점의 차수 = 진입차수 + 진출차수
      - 방향 그래프의 **진입차수**(in-degree) : 정점을 머리로 하는 간선의 수
      - 방향 그래프의 **진출차수**(out-degree) : 정점을 꼬리로 하는 간선의 수
      - 방향 그래프 G3에서 정점 B의 진입차수는 1, 진출차수는 2
        정점 B의 전체 차수는 (진입차수 + 진출차수) 이므로 3이 된다.



Data structure - Graph

- **경로(Path)**
  - 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열한 것 즉, 정점 Vi에서 Vj까지 간선으로 연결된 정점을 순서대로 나열한 리스트
    - 그래프 G1에서 정점 A에서 정점 C까지는 A-B-C 경로와 A-B-D-C 경로, A-D-C 경로, 그리고 A-D-B-C 경로가 있다.
- **경로길이(Path length)**
  - 경로를 구성하는 간선의 수
- **단순경로(Simple path)**
  - 모두 다른 정점으로 구성된 경로
    - 그래프 G1에서 정점 A에서 정점 C까지의 경로 A-B-C는 단순경로이고, 경로 A-B-D-A-B-C는 단순경로가 아니다.
- **사이클(Cycle)**
  - 단순경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로
    - 그래프 G1에서 단순경로 A-B-C-D-A와 그래프 G4에서 단순경로 A-B-A는 사이클이 된다.



Data structure - Graph

- **DAG(directed acyclic graph)**
  - 방향 그래프이면서 사이클이이 없는 그래프
- **연결 그래프(connected graph)**
  - 서로 다른 모든 쌍의 정점들 사이에 경로가 있는 그래프 즉, 떨어져 있는 정점이 없는 그래프
  - 그래프에서 두 정점 Vi에서 Vj까지의 경로가 있으면 정점 Vi와 Vj가 연결(connected) 되었다고 한다.
  - 트리는 사이클이 없는 연결 그래프이다.

Data structure - Graph

- **추상 자료형 그래프**

```
ADT  Graph
  데이터 : 공백이 아닌 정점의 집합과 간선의 집합
  연산 :
      g∈Graph;  u,v∈V;
      createGraph() ::= create an empty Graph;
            // 공백 그래프의 생성 연산

      isEmpty(g) ::= if (g have no vertex)  then return true;
                     else return false;
            // 그래프 g가 정점이 없는 공백 그래프인지를 검사하는 연산

      insertVertex(g, v) ::= insert vertex v into g;
            // 그래프 g에 정점 v를 삽입하는 연산

      insertEdge(g, u, v) ::= insert edge (u,v) into g;
            // 그래프 g에 간선 (u,v)를 삽입하는 연산

      deleteVertex(g, v) ::= delete vertex v and all edges incident on v from g;
            // 그래프 g에서 정점 v를 삭제하고 그에 부속된 모든 간선을 삭제하는 연산

      deleteEdge(g, u, v) ::= delete edges (u,v) from  g;
            // 그래프 g에서 간선 (u,v)를 삭제하는 연산

      adjacent(g, v) ::= return set of all vertices adjacent to v;
            // 정점 v에 인접한 모든 정점을 반환하는 연산
End Graph
```

Data structure - Graph

- **인접 행렬(adjacent matrix)**
  - 행렬에 대한 2차원 배열을 사용하는 순차 자료구조 방법
  - 그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장
    - n개의 정점을 가진 그래프 : n x n 정방행렬
    - 행렬의 행번호와 열번호 : 그래프의 정점
    - 행렬 값 : 두 정점이 인접되어있으면 1, 인접되어있지 않으면 0
  - 무방향 그래프의 인접 행렬
    - 행 i의 합 = 열 i의 합 = 정점 i의 차수
  - 방향 그래프의 인접 행렬
    - 행 i의 합 = 정점 i의 진출차수
    - 열 i의 합 = 정점 i의 진입차수

Data structure - Graph

- **인접 행렬 표현의 단점**
  - n개의 정점을 가지는 그래프를 항상 n x n개의 메모리 사용
  - 정점의 개수에 비해서 간선의 개수가 적은 희소 그래프에 대한 인접 행렬은 희소 행렬이 되므로 메모리의 낭비 발생

Data structure - Graph

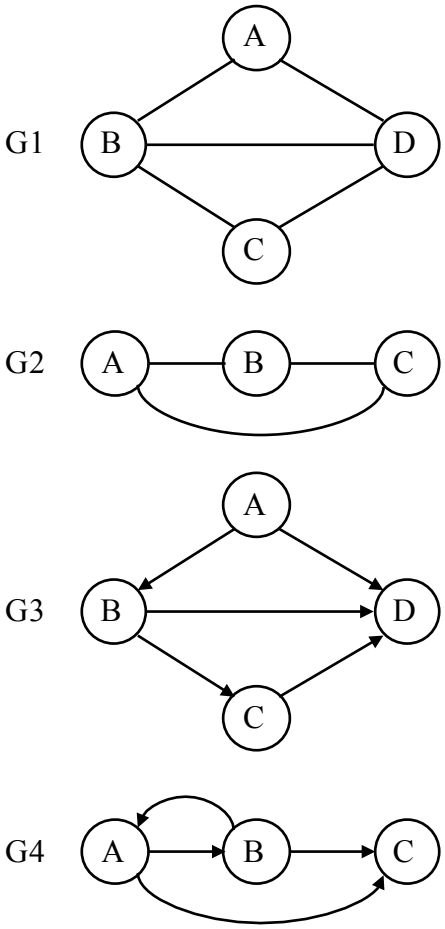
- **인접 리스트(adjacent list)**
  - 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트
  - 각 정점의 차수만큼 노드를 연결
    - 리스트 내의 노드들은 인접 정점에 대해서 오름차순으로 연결
  - 인접 리스트의 각 노드
    - 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성
  - 정점의 헤드 노드
    - 정점에 대한 리스트의 시작을 표현

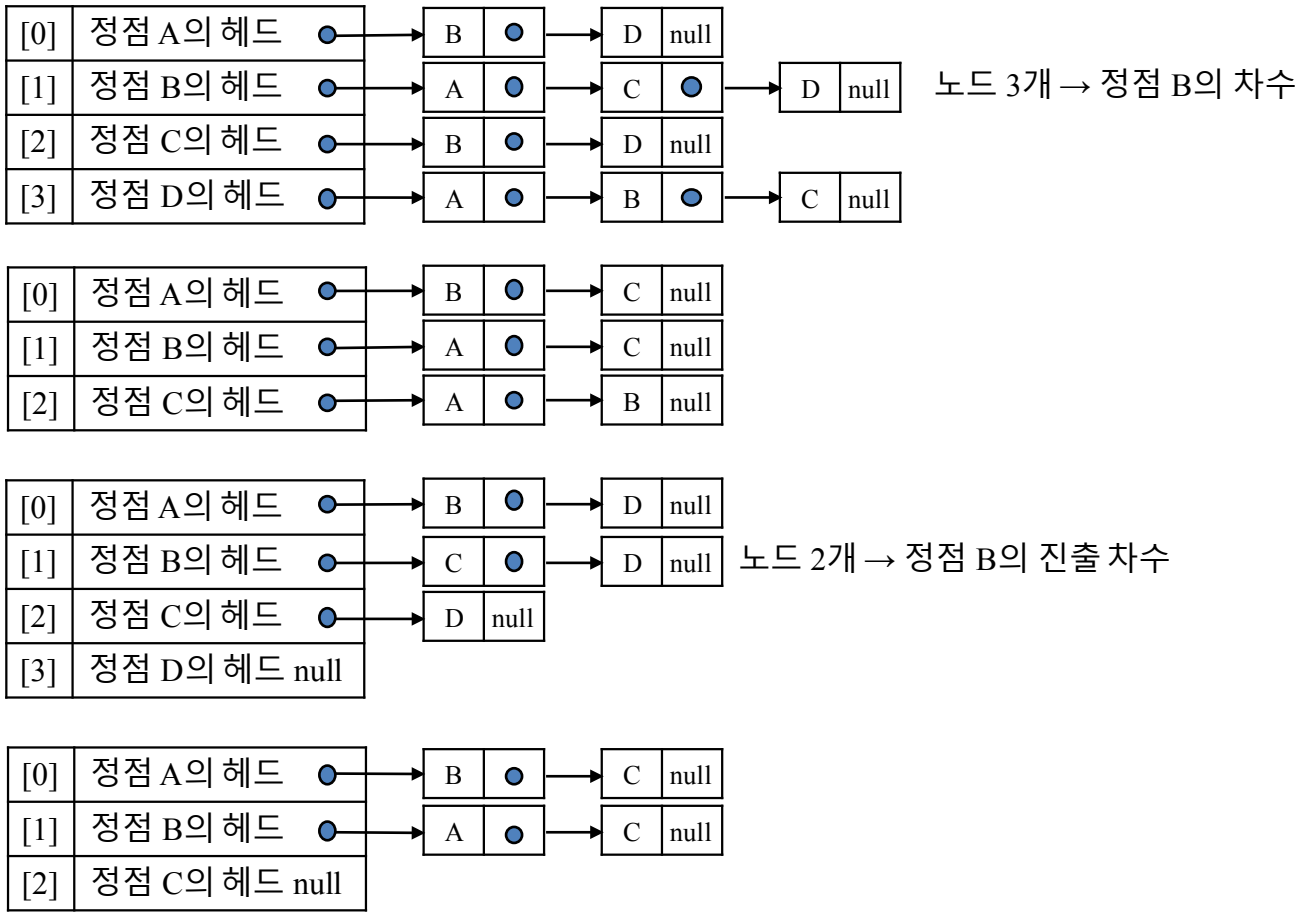
Data structure - Graph

- n개의 정점과 e개의 간선을 가진 무방향 그래프의 인접 리스트
- 헤드 노드 배열의 크기 : n
- 연결하는 노드의 수 : 2e
- 각 정점의 헤드에 연결된 노드의 수 : 정점의 차수

- n개의 정점과 e개의 간선을 가진 방향 그래프의 인접 리스트
- 헤드 노드 배열의 크기 : n
- 연결하는 노드의 수 : e
- 각 정점의 헤드에 연결된 노드의 수 : 정점의 진출 차수

Data structure - Graph





Data structure - Graph

- **그래프 순회(graph traversal), 그래프 탐색(graph search)**
  - 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문하여 처리하는 연산
  - 그래프 탐색방법
    - 깊이 우선 탐색(depth first search : DFS)
    - 너비 우선 탐색(breadth first search : BFS)

Data structure - Graph

- **깊이 우선 탐색의 수행 순서**

⑴ 시작 정점 v를 결정하여 방문한다.

⑵ 정점 v에 인접한 정점 중에서

① **방문하지 않은 정점 w가 있으면, 정점 v를 스택에 push**하고 정점 w를 방문한다. 그리고 w를 v로 하여 다시 ⑵를 반복한다.

② **방문하지 않은 정점이 없으면**, 탐색의 방향을 바꾸기 위해서 **스택을 pop**하여 받은 가장 마지막 방문 정점을 v로 하여 다시 ⑵를 반복한다.

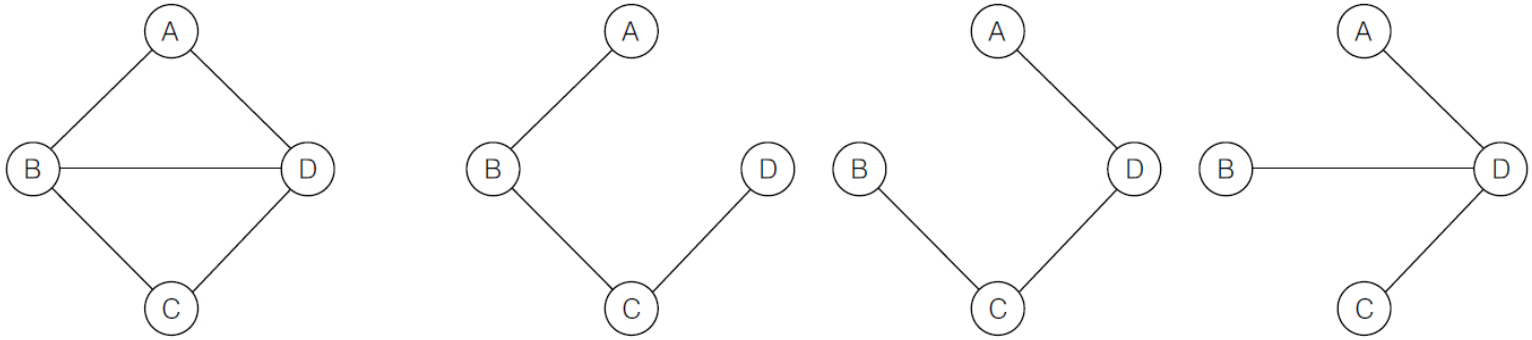
⑶ 스택이 공백이 될 때까지 ⑵를 반복한다.

Data structure - Graph

- **너비 우선 탐색(Breath first search : BFS)**
  - 순회 방법
    - 시작 정점으로 부터 인접한 정점들을 모두 차례로 방문하고 나서, 방문했던 정점을 시작으로 하여 다시 인접한 정점들을 차례로 방문하는 방식
    - 가까운 정점들을 먼저 방문하고 멀리 있는 정점들은 나중에 방문하는 순회방법
    - 인접한 정점들에 대해서 차례로 다시 너비 우선 탐색을 반복해야 하므로 선입선출 구조를 갖는 큐 사용

Data structure - Graph

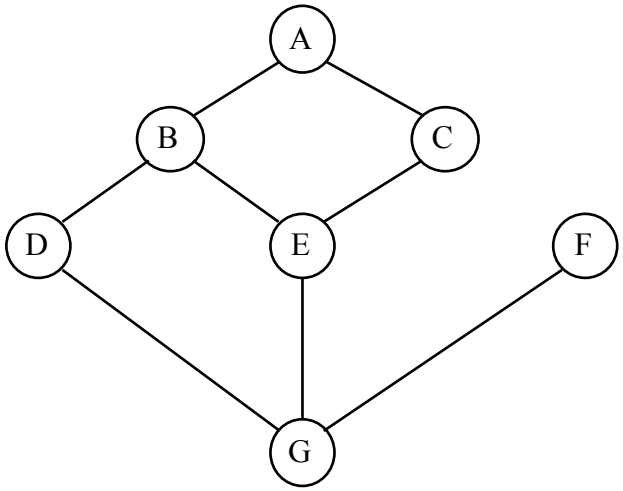
- **신장 트리(spanning tree)**
  - n개의 정점으로 이루어진 무방향 그래프 G에서 n개의 모든 정점과 n-1개의 간선으로 만들어진 트리
  - 그래프 G1과 신장 트리의 예

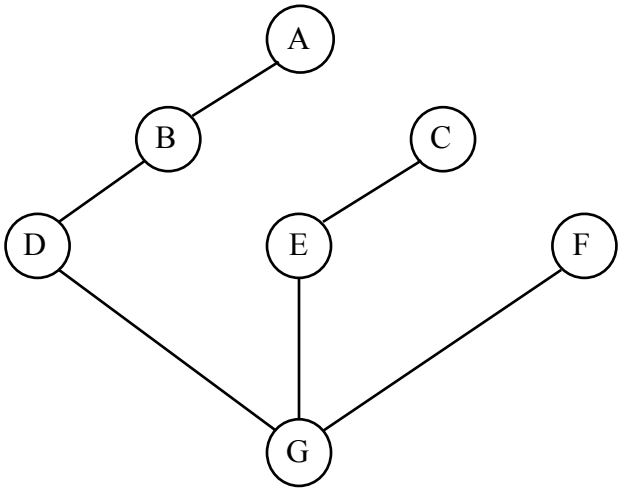


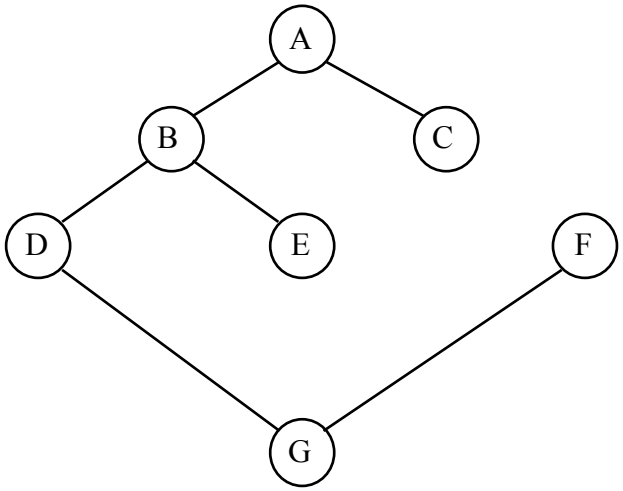
[그림 9-13] 그래프 G1과 신장 트리의 예

Data structure - Graph

- 깊이 우선 신장 트리(depth first spanning tree)
  - 깊이 우선 탐색을 이용하여 생성된 신장 트리
- 너비 우선 신장 트리(breadth first spanning tree)
  - 너비 우선 탐색을 이용하여 생성된 신장 트리
- 그래프 G1의 깊이 우선 신장 트리와 너비 우선 신장 트리







Data structure - Graph

- **최소 비용 신장 트리(minimum cost spanning tree)**
  - 무방향 가중치 그래프에서 신장 트리를 구성하는 간선들의 가중치 합이 최소인 신장 트리
    - 가중치 그래프의 간선에 주어진 가중치
      - **비용이나 거리, 시간을 의미하는 값**
  - 최소 비용 신장 트리를 만드는 알고리즘
    - Kruskal 알고리즘 I, II
    - Prime 알고리즘

Data structure - Graph

- **Kruskal 알고리즘 I**
  - 가중치가 높은 간선을 제거하면서 최소 비용 신장 트리를 만드는 방법
  - Kruskal 알고리즘 I

⑴ 그래프 G의 **모든 간선을 가중치에 따라 내림차순으로 정리한다**.

⑵ 그래프 G에서 **가중치가 가장 높은 간선을 제거한다.**

이때 정점을 그래프에서 분리시키는 간선은 제거할 수 없으므로

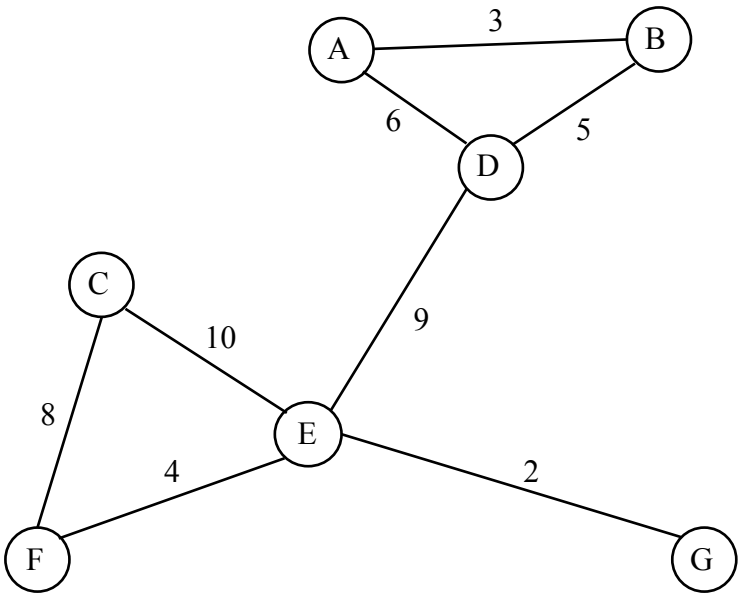
이런 경우에는 그 다음으로 가중치가 높은 간선을 제거한다.

⑶ 그래프 G에 n-1개의 간선만 남을 때까지 ⑵를 반복한다.

⑷ 그래프에 n-1개의 간선이 남게 되면 최소 비용 신장 트리가 완성된다.

Data structure - Graph

③ 남은 간선 중에서 가중치가 가장 큰 간선 (B,G) 제거. (현재 남은 간선의 수 : 8개)

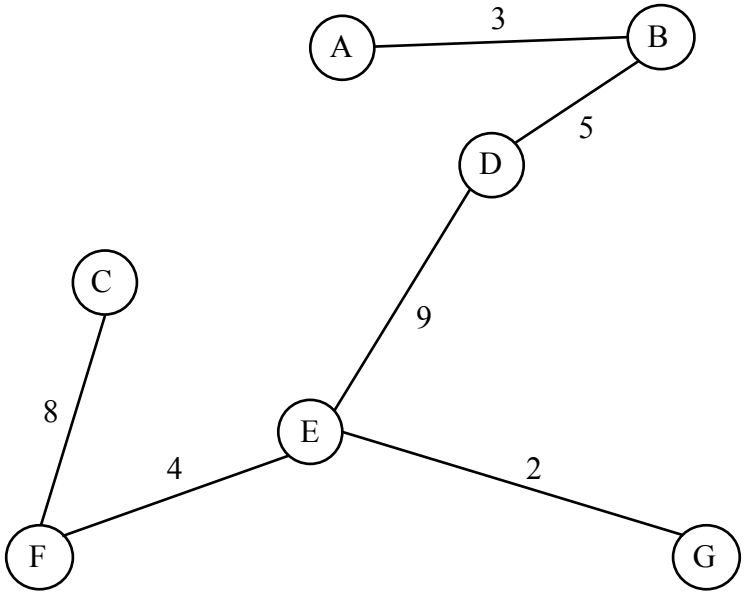


간선의 수: 8개

가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

Data structure - Graph

- G10의 최소 비용 신장 트리
  - 현재 남은 간선의 수가 6개 이므로 알고리즘 수행을 종료하고 신장 트리 완성



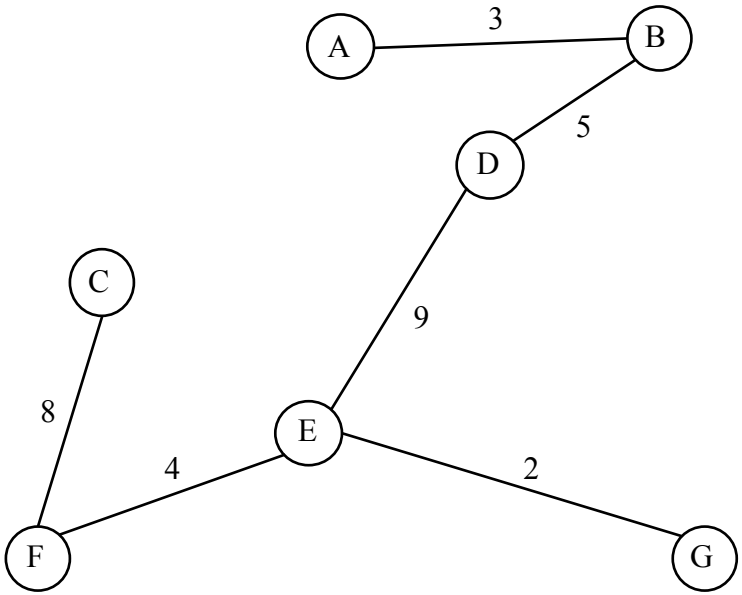
Data structure - Graph

- **Kruskal 알고리즘 Ⅱ**
  - 가중치가 낮은 간선을 삽입하면서 최소 비용 신장 트리를 만드는 방법
  - Kruskal 알고리즘 Ⅱ

> ⑴ 그래프 G의 **모든 간선을 가중치에 따라 오름차순으로 정리한다**.
>
> ⑵ 그래프 G에 **가중치가 가장 작은 간선을 삽입한다.**
>
> 이때 사이클을 형성하는 간선은 삽입할 수 없으므로
>
> 이런 경우에는 그 다음으로 가중치가 작은 간선을 삽입한다.
>
> ⑶ 그래프 G에 n-1개의 간선을 삽입할 때까지 ⑵를 반복한다.
>
> ⑷ 그래프 G의 간선이 n-1개가 되면 최소 비용 신장 트리가 완성된다.

Data structure - Graph

- **G10의 최소 비용 신장 트리**
  - 현재 삽입한 간선의 수가 6개이므로 알고리즘 수행을 종료하고 신장 트리 완성



Data structure - Graph

- **Prime 알고리즘**
  - 간선을 정렬하지 않고 하나의 정점에서 시작하여 트리를 확장하는 방법

⑴ 그래프 G에서 시작 정점을 선택한다.

⑵ **선택한 정점에 부속된 모든 간선 중에서 가중치가 가장 작은 간선을 연결하여 트리를 확장한다.**

⑶ 이전에 선택한 정점과 새로 확장된 정점에 부속된 모든 간선 중에서 가중치가 가장 작은 간선을 삽입한다.

이때 사이클을 형성하는 간선은 삽입할 수 없으므로

그 다음으로 가중치가 작은 간선을 선택한다.

⑷ 그래프 G에 n-1개의 간선을 삽입할 때까지 ⑶을 반복한다.

⑸ 그래프 G의 간선이 n-1개가 되면 최소 비용 신장 트리가 완성된다.