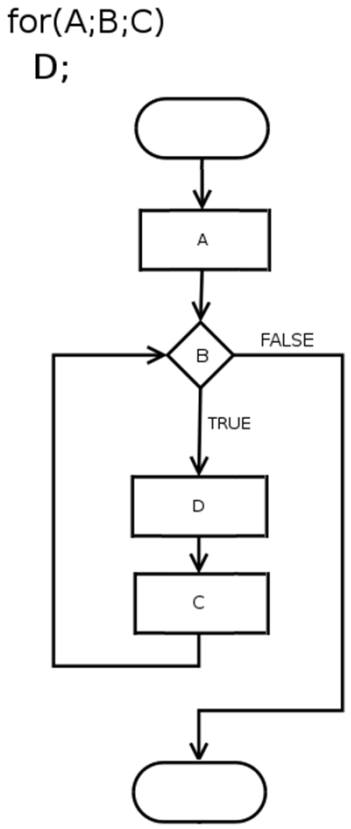
Clickseo.com

자료구조 & 알고리즘

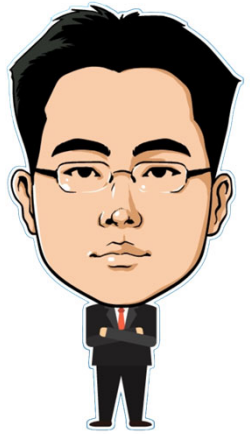


그래프
(Graph)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

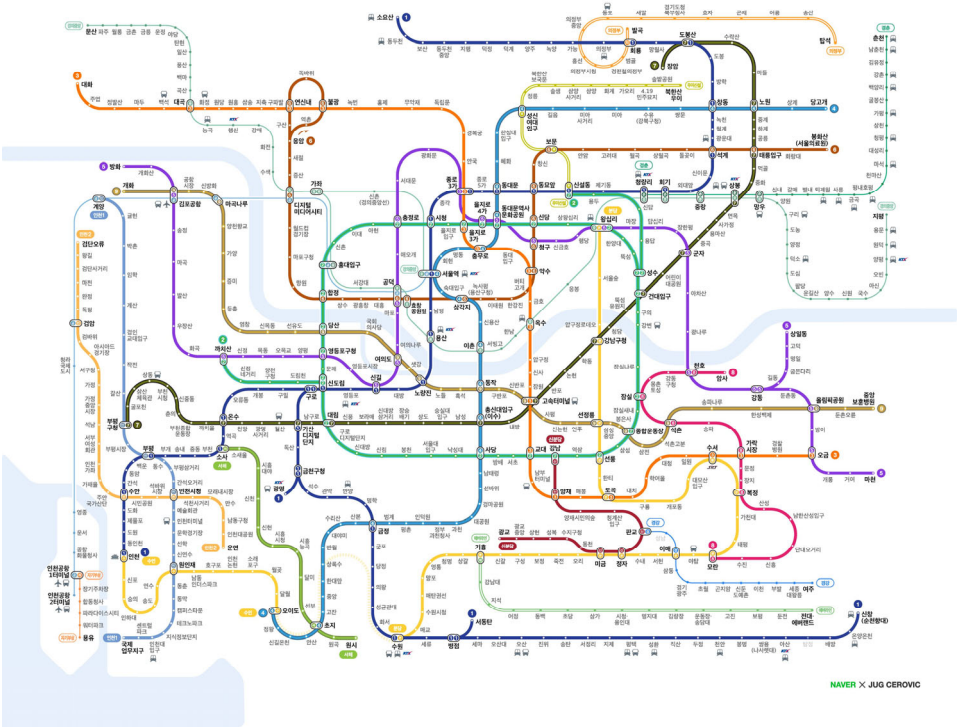
- 그래프 이해와 표현
- 그래프 순회
- 가중치 그래프

그래프 이해와 표현

- **그래프 이해와 표현**
  - 그래프의 이해
  - 그래프 종류
  - 그래프 표현
- 그래프 순회
- 가중치 그래프

그래프 이해 (1/3)

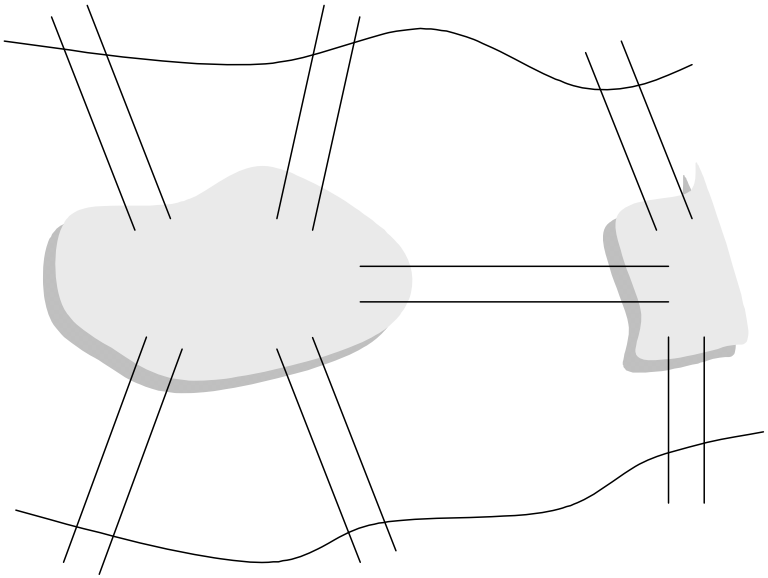
- **그래프(Graph)**
  - 연결되어 있는 원소 간의 관계를 표현하는 자료구조
    - **그래프의 예:** 인맥 지도, 수도 배관 배수 시스템, 물질의 분자 구조



[이미지 출처: "서울특별시", https://news.seoul.go.kr/]

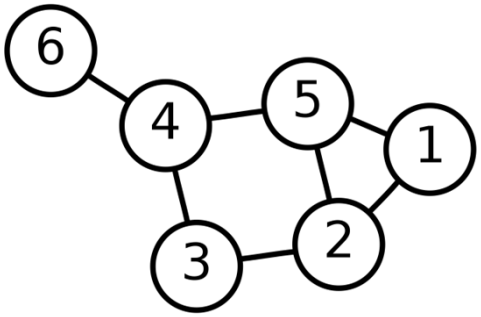
그래프 이해 (2/3)

- **케인즈버그(Koenigsberg)의 다리 문제**
  - **1736년, 오일러(Euler)가 최초로 사용한 것**
    - 섬은 **정점(vertex)**으로 놓고, 다리를 **간선(edges)**으로 나타냄.
    - **각 정점의 차수가 짝수인 경우에만 임의의 정점에서 출발하여 각 간선을 단 한 번씩만 거치고 출발한 정점으로 되돌아오는 길이 있음을 보였다.**



그래프 이해 (3/3)

- **그래프 정의**
  - 그래프 G는 집합(Set) 두 개로 구성
    - 정점(Vertex 또는 Node)
    - 간선(Edge)



$$G = (V, E)$$

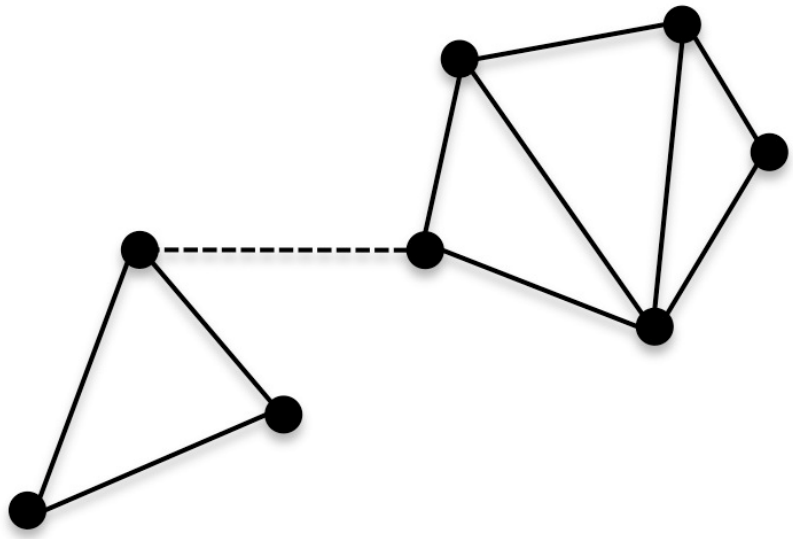
- V 는 그래프에 있는 정점들의 집합을 의미한다(대상: 대상물, 개념 등).
- E 는 정점을 연결하는 간선들의 집합을 의미한다(대상들 간의 관계).

그래프의 이해와 표현

그래프의 이해: 용어

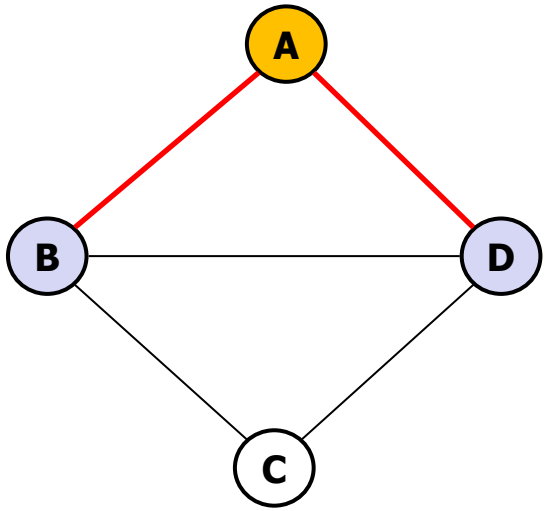
그래프의 이해: 용어 (1/6)

- **연결(Connected)**
  - 그래프에서 두 정점 V_i와 V_j까지의 경로가 있으면, 정점 V_i와 V_j가 연결되었다고 한다.
    - 연결 그래프(Connected Graph): 떨어져 있는 정점이 없는 그래프
    - 단절 그래프(Disconnected Graph): 연결되지 않은 정점이 있는 그래프



그래프의 이해: 용어 (2/6)

- **인접과 부속(Adjacent and Incident)**
  - 그래프에서 두 정점 V_i와 V_j가 연결되어 간선 (V_i, V_j)가 있을 때 두 정점 V_i와 V_j를 인접(adjacent)되어 있다고 하고, 간선 (V_i, V_j)는 정점 V_i와 V_j에 부속(incident)되어 있다고 한다.



무향 그래프

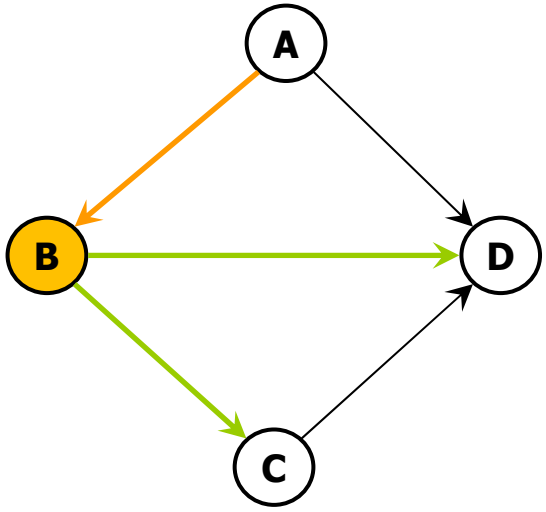
[그래프 G1]

정점 A 와 인접한 정점은 B 와 D

: 정점 A에 부속되어 있는 간선은 (A, B)와 (A, D)

그래프의 이해: 용어 (3/6)

- **차수(Degree)**
  - 정점에 부속되어 있는 간선의 수
    - 방향 그래프에서는 정점에 부속된 간선의 방향에 따라서
      - 진입 차수(in-degree)
      - 진출 차수(out-degree)
    - 방향 그래프에서의 정점의 차수는 진입 차수와 진출 차수를 합한 값이다.



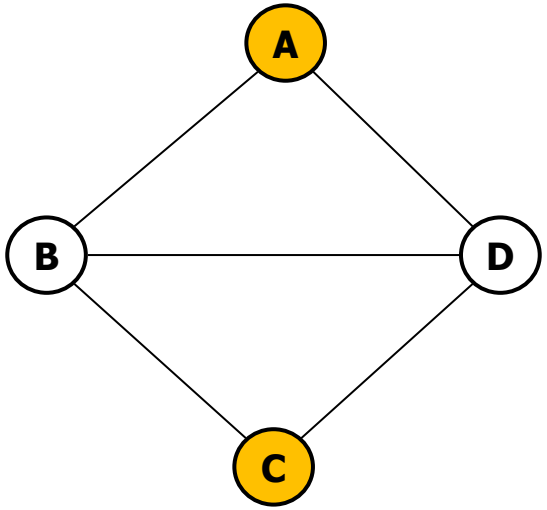
[그래프 G3]

방향그래프

정점 B 의 진입 차수는 1 이고, 진출 차수는 2 이다.
정점 B 의 전체 차수는 3 이다.

그래프의 이해: 용어 (4/6)

- **경로(Path)**
  - 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열 한 것



[그래프 G1]

정점 A 에서 정점 C 까지는 4 가지의 경로가 존재

A-B-C

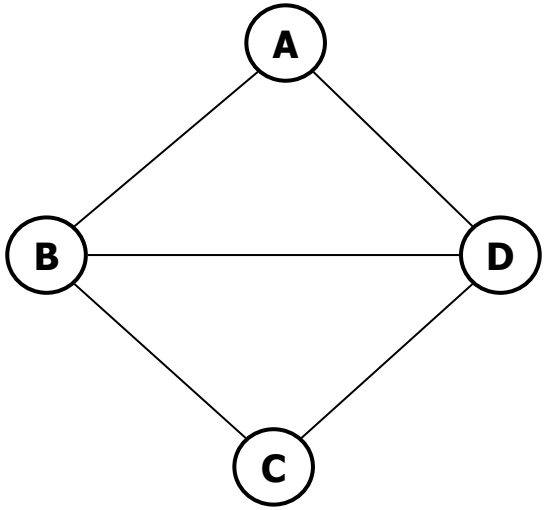
A-B-D-C

A-D-C

A-D-B-C

그래프의 이해: 용어 (5/6)

- **경로:** 경로 길이
  - 경로 길이(Path Length): 경로를 구성하는 간선의 수



[그래프 G1]

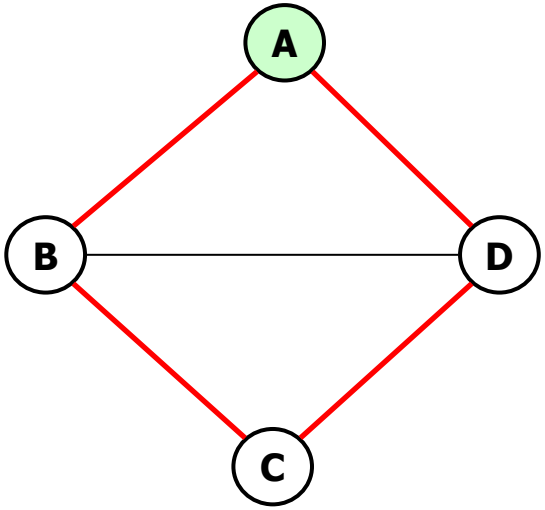
경로 A-D-B-C 는 간선 3개로 이루어진다.

(A, D)
(D, B)
(B, C)

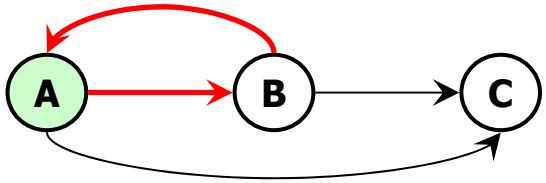
경로 길이는 3 이 된다.

그래프의 이해: 용어 (6/6)

- **경로:** 순환
  - **단순 경로:** 모두 다른 정점으로 구성된 경로
  - **순환**(Cycle)
    - 단순 경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로



[그래프 G1]



[그래프 G4]

그래프 이해와 표현

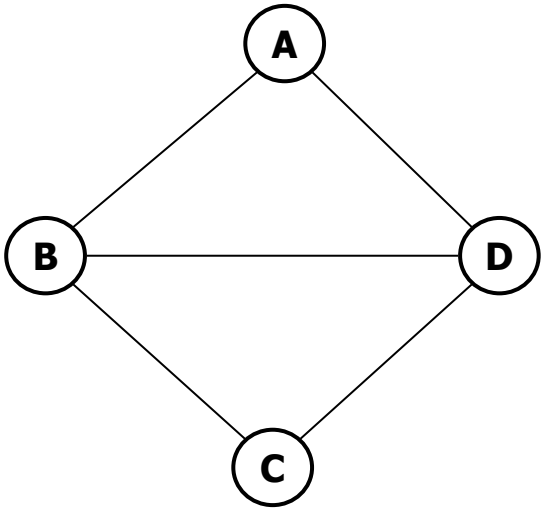
그래프 종류

그래프 종류 (1/7)

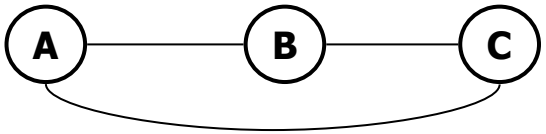
- **그래프 종류**
  - **무향 그래프**(Undirected Graph)
    - 두 정점을 연결하는 간선에 방향성이 없는 그래프
  - **방향 그래프**(Directed Graph)
    - 두 정점을 연결하는 간선에 방향성이 있는 그래프
  - **가중치 그래프**(Weight Graph)
    - **두 정점을 연결하는 간선에 가중치가 할당된 그래프**
      - 가중치는 두 정점 사이의 거리 또는 지나는 시간이 될 수도 있다.
      - 또한 음수인 경우도 존재한다.
  - **완전 그래프**(Complete Graph)
    - **모든 정점들 사이에 1:1로 직접 연결된 간선을 지닌 그래프**
  - **부분 그래프**(Subgraph)
    - **원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프**
      - 부분 그래프는 원래의 그래프에 없는 정점이나 간선을 포함하지 않는다.
  - **트리**(Tree): 순환이 없는 연결된 그래프

그래프 종류 (2/7)

- **무향 그래프**(Undirected Graph)
  - 두 정점을 연결하는 간선에 방향성이 없는 그래프



[그래프 G1]



[그래프 G2]

V(G1) = { A, B, C, D }

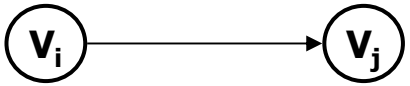
E(G1) = { (A,B), (A,D), (B,C), (B,D), (C,D) }

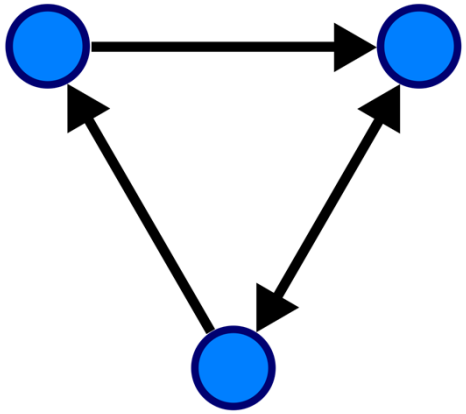
V(G2) = { A, B, C }

E(G1) = { (A,B), (A,C), (B,C)}

그래프 종류 (3/7)

- **방향 그래프**(Directed Graph)
  - 두 정점을 연결하는 간선에 방향성이 있는 그래프
    - 정점 V_i에서 정점 V_j를 연결하는 간선



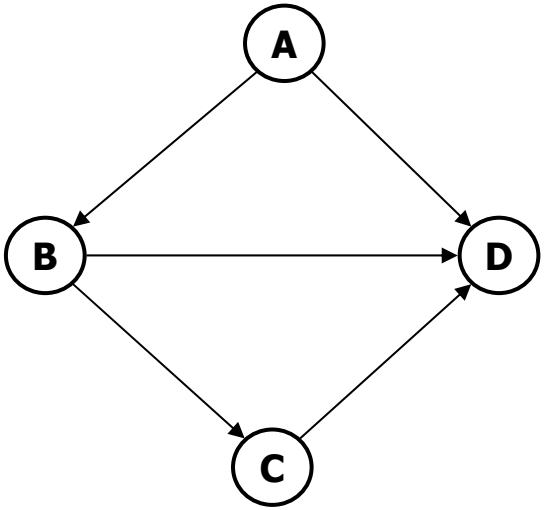


$V_i \rightarrow V_j$ 또는 $<V_i, V_j>$ 로 표현

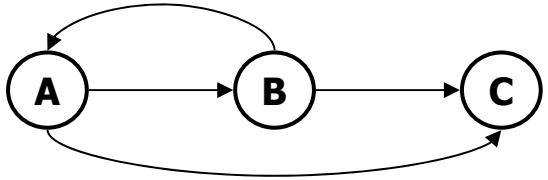
$<V_i, V_j>$와 $<V_j, V_i>$는 서로 다른 간선이 된다.

그래프 종류 (4/7)

- **방향 그래프**
  - 그래프 G3, G4



[그래프 G3]



[그래프 G4]

V(G3) = { A, B, C, D }

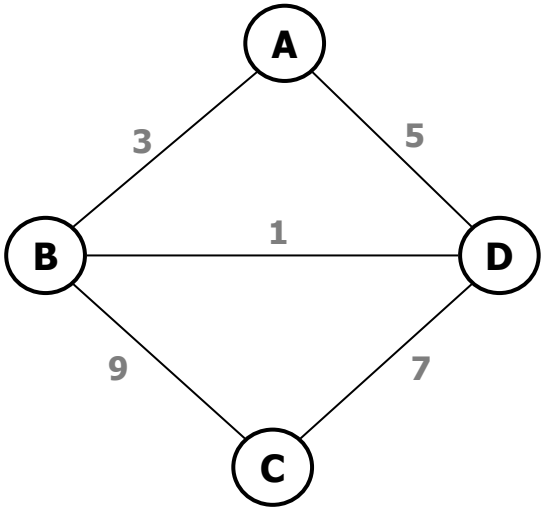
E(G3) = { <A,B>, <A,D>, <B,C>, <B,D>, <C,D> }

V(G4) = { A, B, C }

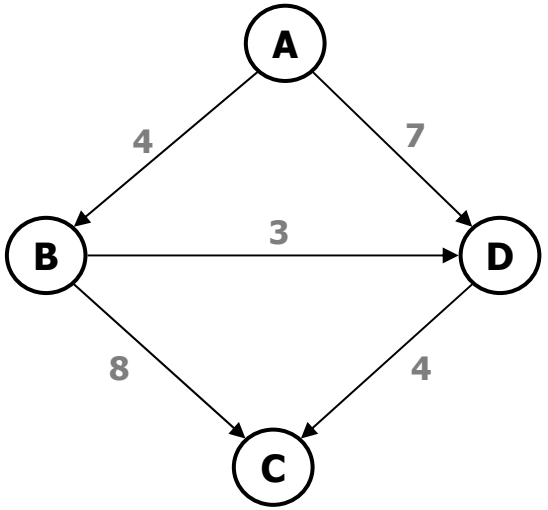
E(G4) = { <A,B>, <A,C>, <B,A>, <B,C> }

그래프 종류 (5/7)

- **가중치 그래프**(Weight Graph)
  - 두 정점을 연결하는 간선에 가중치를 할당한 그래프
    - 가중치는 두 정점 사이의 거리 또는 지나는 시간이 될 수도 있다.
    - 또한 음수인 경우도 존재한다.



[그래프 G5]

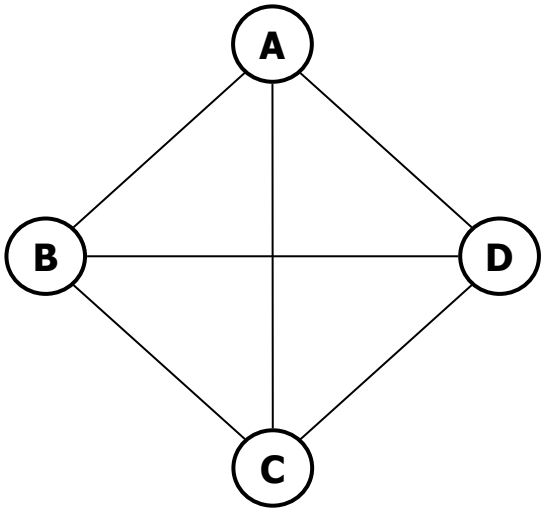


[그래프 G6]

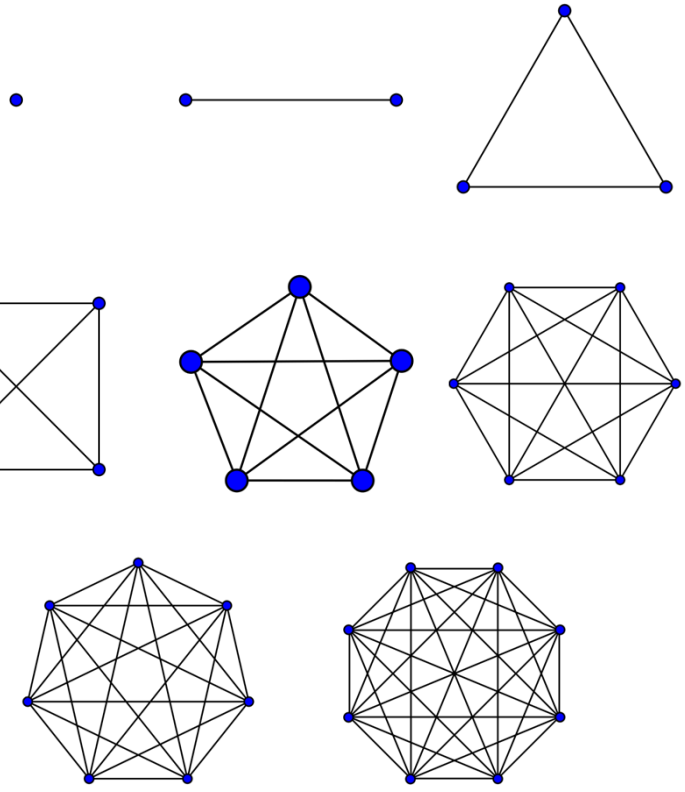
그래프 종류 (6/7)

- **완전 그래프**(Complete Graph)
  - 모든 정점들 사이에 1:1로 직접 연결된 간선을 지닌 그래프

정점이 n 개인 무방향 그래프
최대 간선 수 = n(n-1)/2

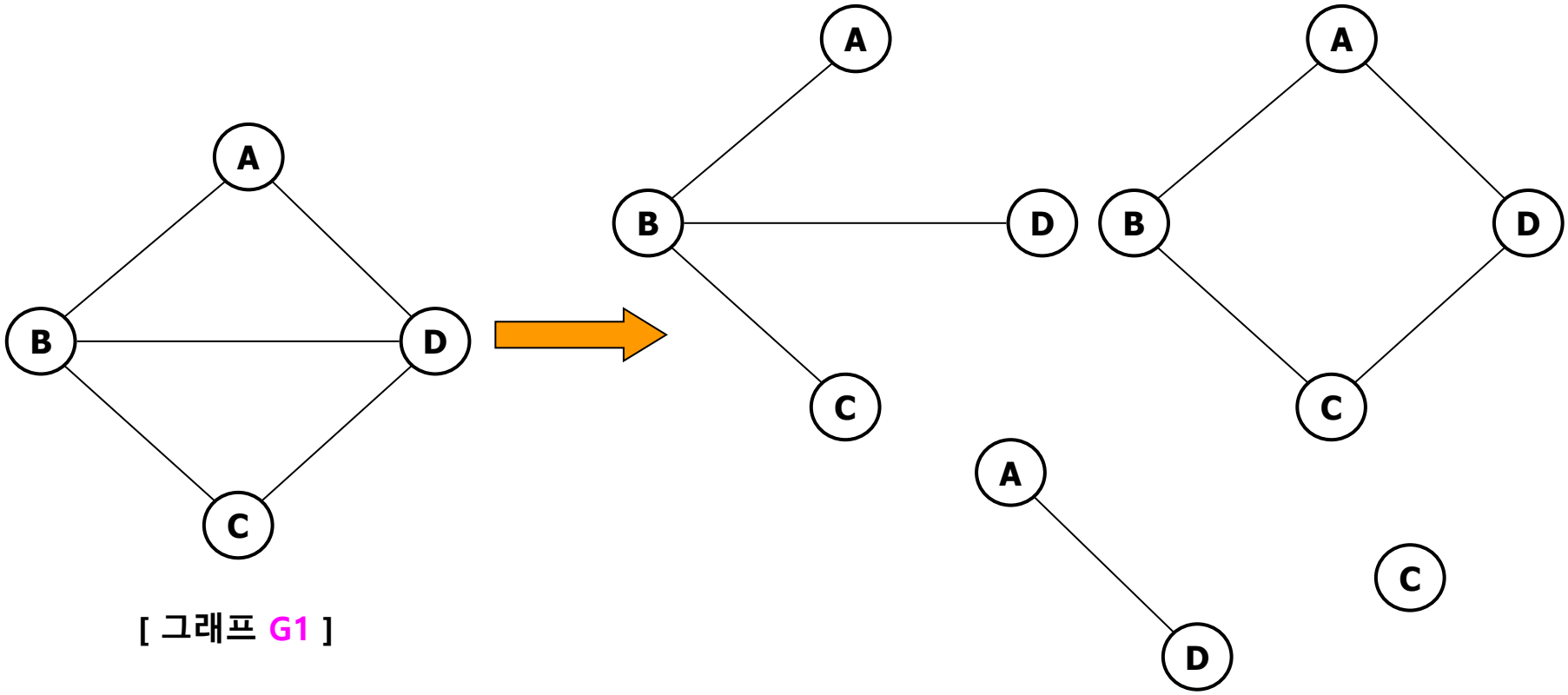


[그래프 G7]



그래프 종류 (7/7)

- **부분 그래프**(subgraph)
  - 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
    - 부분 그래프는 원래의 그래프에 없는 정점이나 간선을 포함하지 않는다.



[그래프 G1]

[그래프 G1′]

그래프 이해와 표현

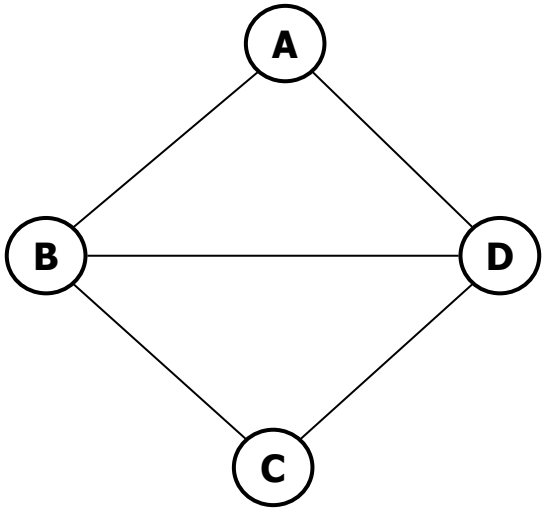
그래프 표현: 인접 행렬, 인접 리스트

그래프 표현: 인접 행렬 (1/3)

- **인접 행렬**(Adjacent Matrix)
  - 순차 자료구조를 이용하는 2차원 배열의 방법
    - **그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장**
      - N 개의 정점을 가진 그래프: **N x N 정방 행렬**
      - **행렬의 행과 열:** 그래프의 정점
      - **행렬 값: 두 정점이 인접되어 있으면 1, 인접되어 있지 않으면 0**
  - **무향 그래프의 인접 행렬**
    - 행 i 의 합 = 열 i 의 합 = 정점 i 의 차수
  - **방향 그래프의 인접 행렬**
    - 행 i 의 합 = 정점 i 의 진출 차수
    - 열 i 의 합 = 정점 i 의 진입 차수

그래프 표현: 인접 행렬 (2/3)

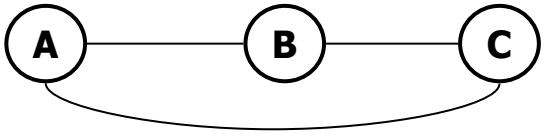
- **인접 행렬:** 2차원 배열



[그래프 G1]

	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	0

정점 B의 차수: 3 = 1+0+1+1

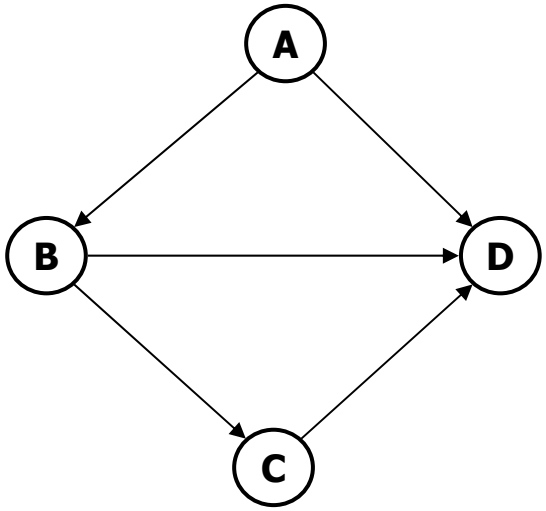


[그래프 G2]

	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

그래프 표현: 인접 행렬 (3/3)

- **인접 행렬:** 2차원 배열

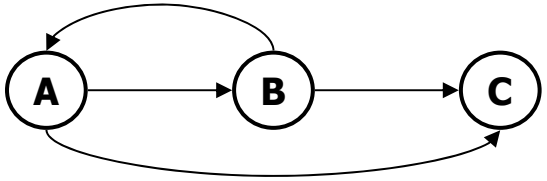


[그래프 G3]

	A	B	C	D
A	0	1	0	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

정점 B 의 진출 차수: 2 = 0+0+1+1

정점 B 의 진입 차수: 1 = 1+0+0+0



[그래프 G4]

	A	B	C
A	0	1	1
B	1	0	1
C	0	0	0

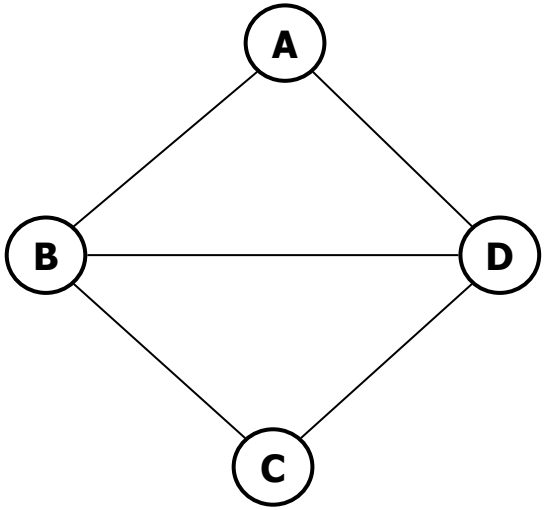
그래프 표현: 인접 리스트 (1/3)

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

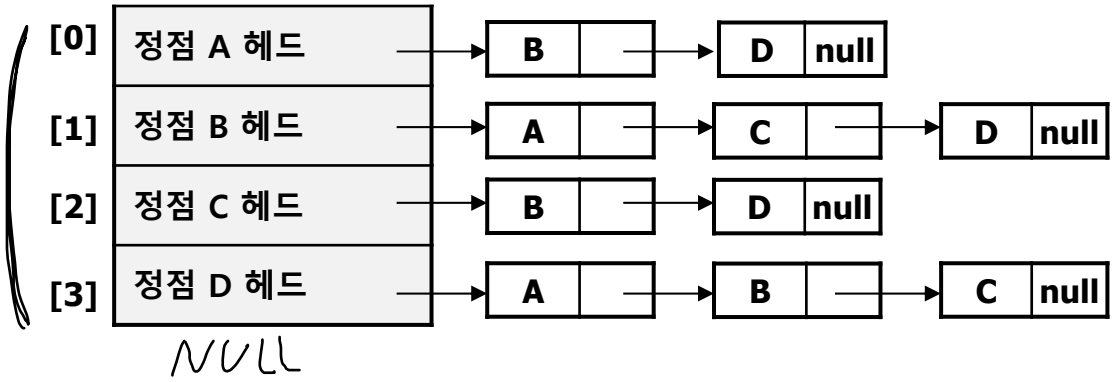
그래프 표현: 인접 리스트 (2/3)

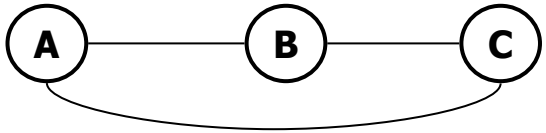
- **인접 리스트:** 무향 그래프

n 개의 정점(V)과 e 개의 간선(E)을 가진 무향 그래프
헤드 노드의 배열 크기: n
연결하는 노드의 총 수: 2e
각 정점의 헤드에 연결된 노드의 수: 정점의 차수

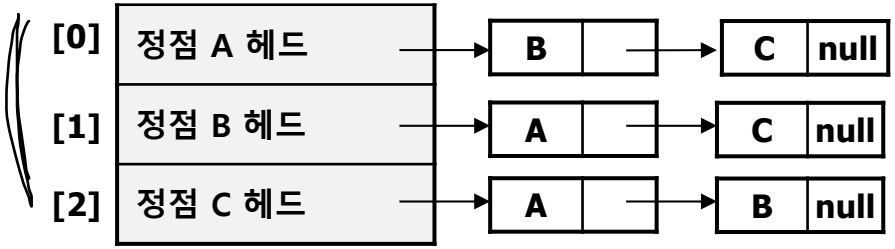


[그래프 G1]





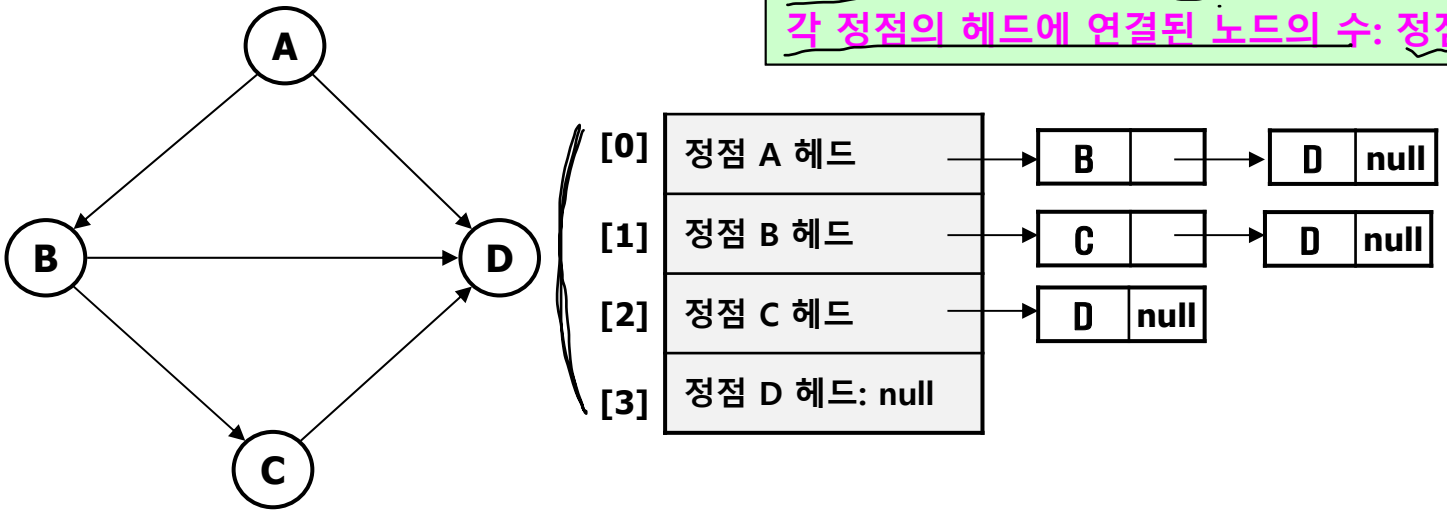
[그래프 G2]



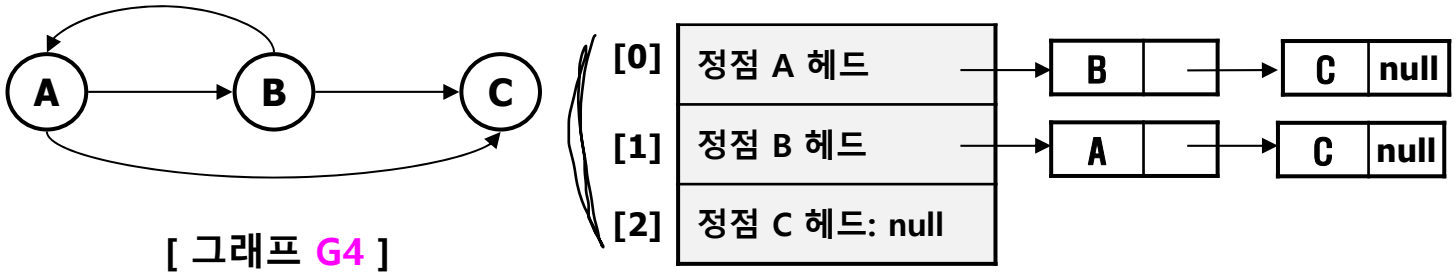
그래프 표현: 인접 리스트 (3/3)

- **인접 리스트:** 방향 그래프

n 개의 정점(V)과 e 개의 간선(E)을 가진 방향 그래프
헤드 노드의 배열 크기: n
연결하는 노드의 총 수: e
각 정점의 헤드에 연결된 노드의 수: 정점의 진출 차수



[그래프 G3]



[그래프 G4]

그래프 순회

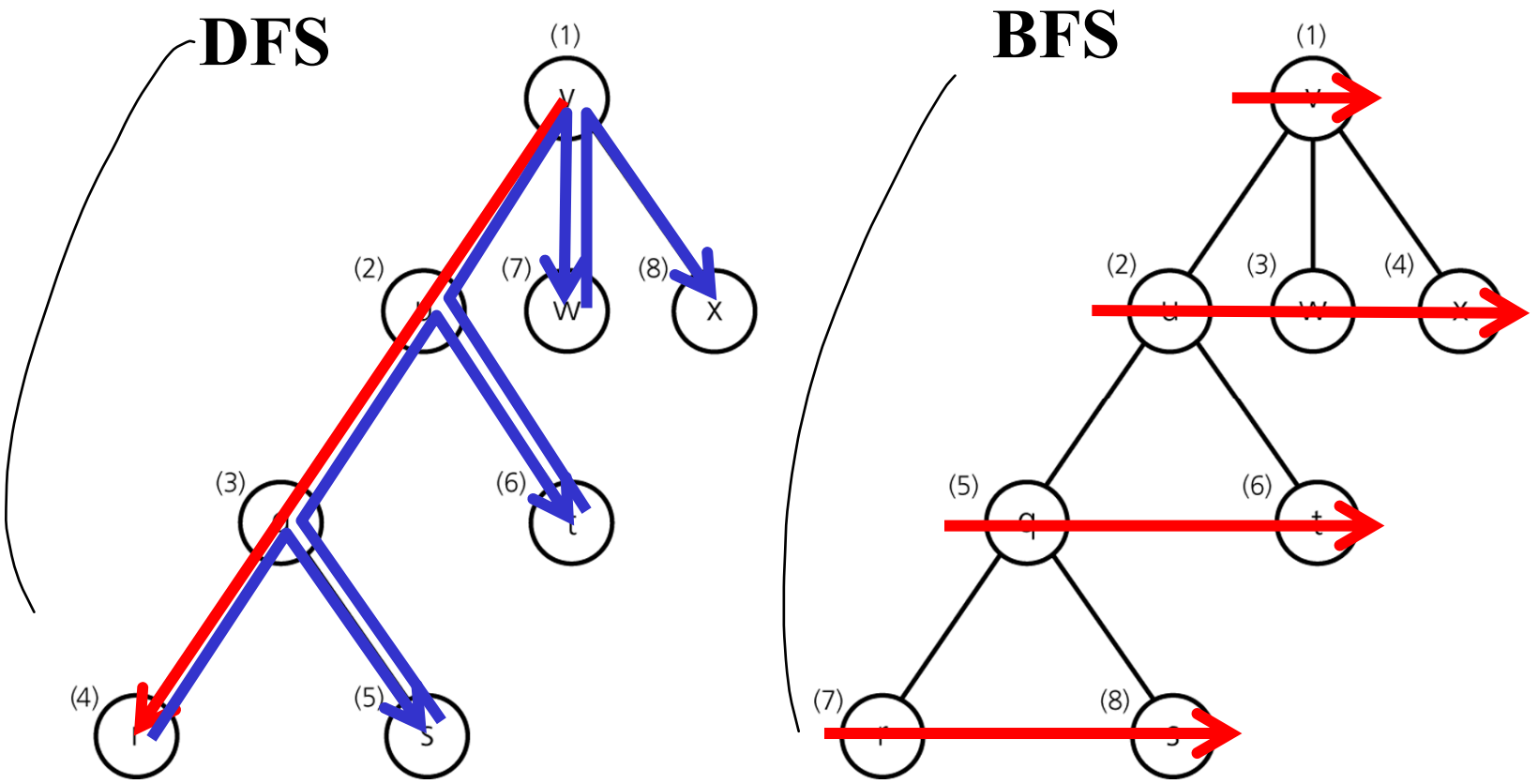
- 그래프의 이해와 표현
- **그래프 순회**
  - 깊이 우선 순회
  - 높이 우선 순회
- 가중치 그래프

그래프 순회 (1/2)

- **그래프 순회**(Graph Traversal)
  - 그래프 탐색(Graph Search)
    - 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문한다.
  - 그래프 순회의 두 가지 방법
    - 깊이 우선 탐색(DFS, Depth First Search)
    - 너비 우선 탐색(BFS, Breadth First Search)

그래프 순회 (2/2)

- 이진 트리의 순회
  - 깊이 우선 순회(DFS)와 너비 우선 순회(BFS)



그래프 순회

깊이 우선 탐색(DFS)

깊이 우선 탐색

● 깊이 우선 탐색

❍ DFS (Depth First Search)

- 시작 정점의 한 방향으로 갈 수 있는 경로가 있는 곳까지 깊이 탐색해가다가 더 이상 갈 곳이 없으면 가장 마지막에 만났던 갈림길 간선이 있는 정점으로 되돌아와서 다른 방향의 간선으로 탐색을 계속하여 결국 모든 정점을 방문하는 순회 방법

(1) 시작 정점 V를 결정하여 방문한다.

(2) 정점 v에 인접한 정점 중에서

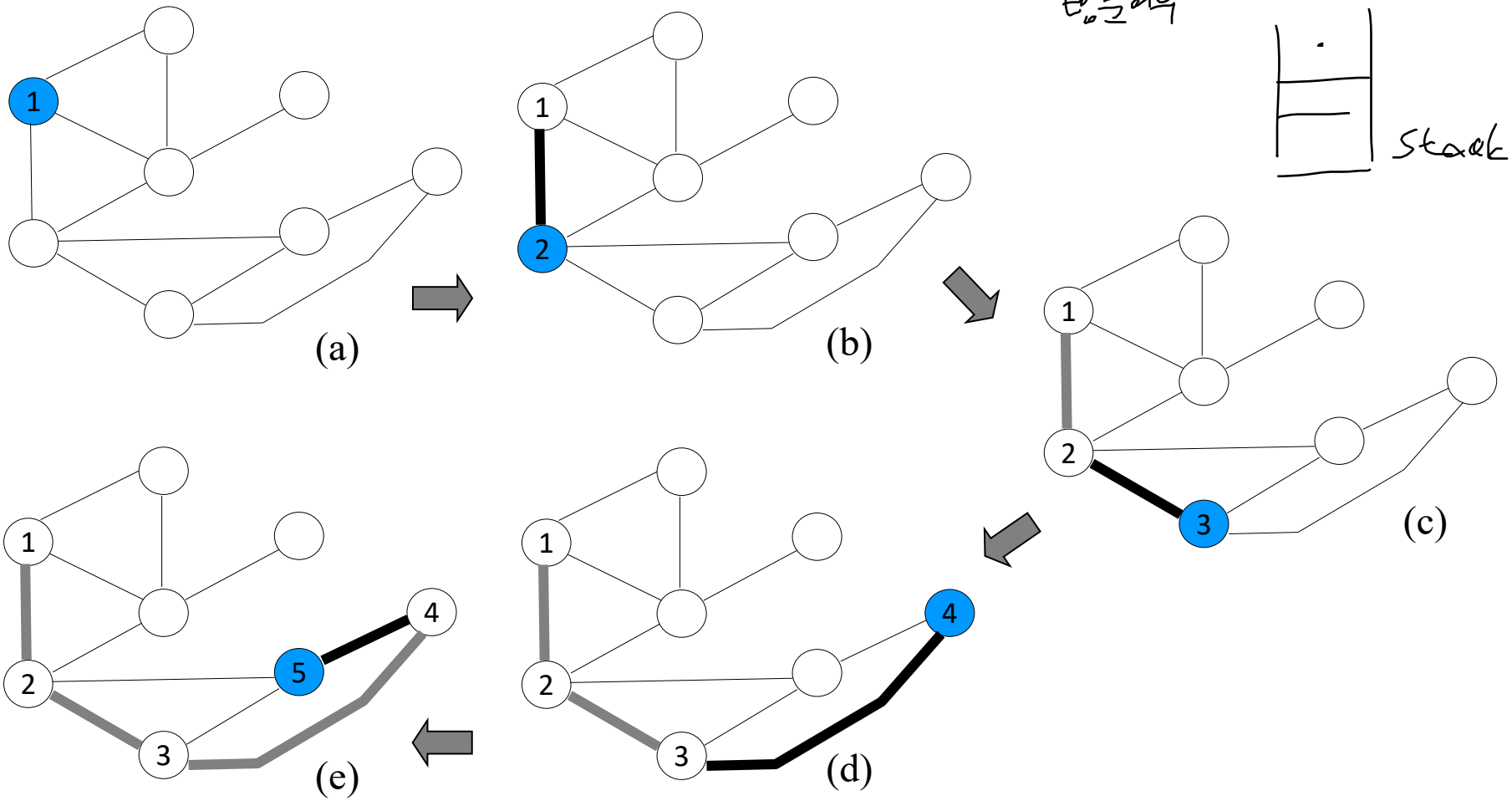
① 방문하지 않은 정점 w가 있으면 정점 v를 스택에 push하고 w를 방문한다.
그리고 w를 v로 하여 다시 (2)를 반복한다.

② 방문하지 않은 정점이 없으면 스택을 pop하여 받은 가장 마지막 방문 정점을 v로 설정한 뒤 다시 (2)를 수행한다.

(3) 스택이 공백이 될 때까지 (2)를 반복한다.

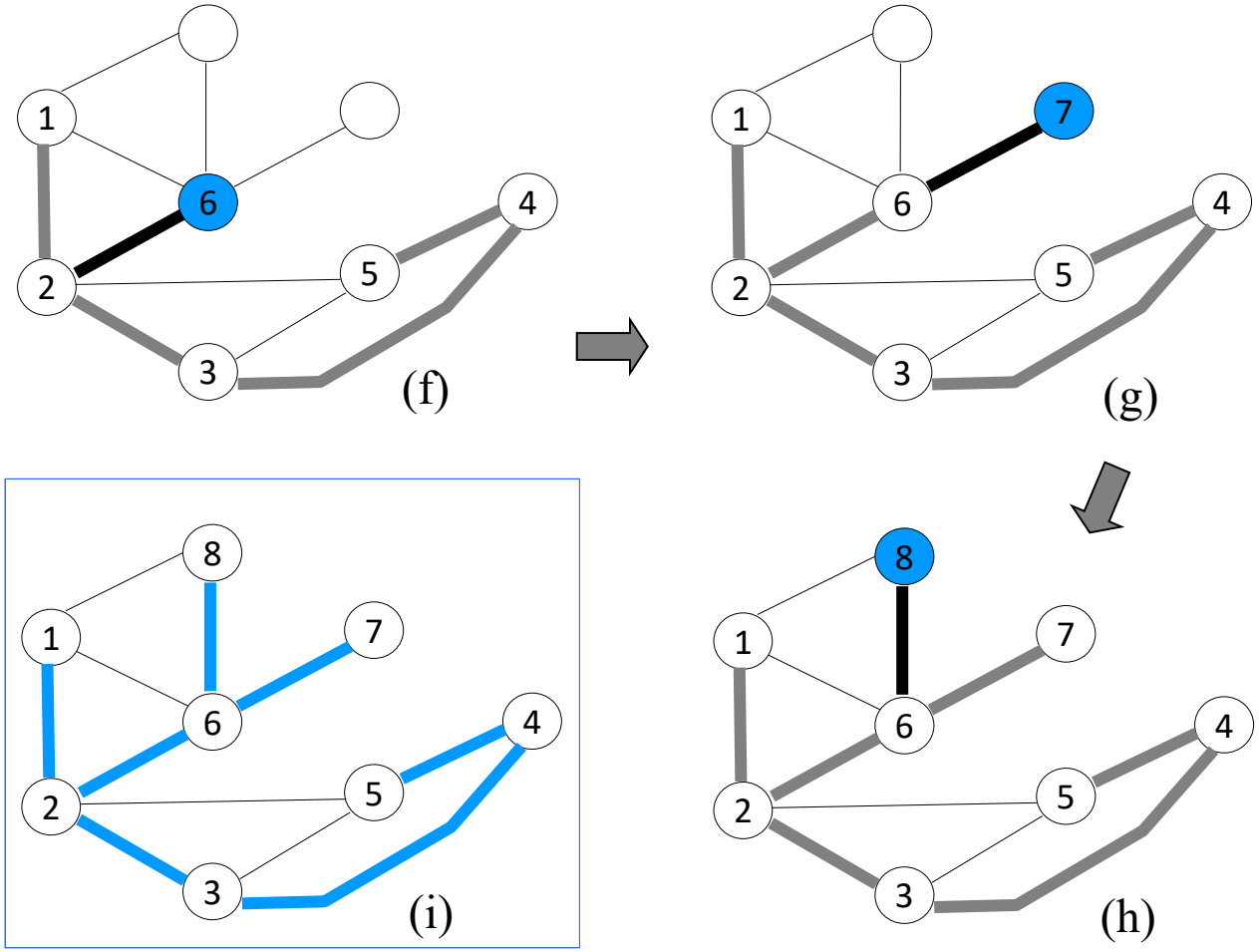
깊이 우선 탐색 (cont'd)

- 깊이 우선 탐색(DFS)의 작동 예 #1



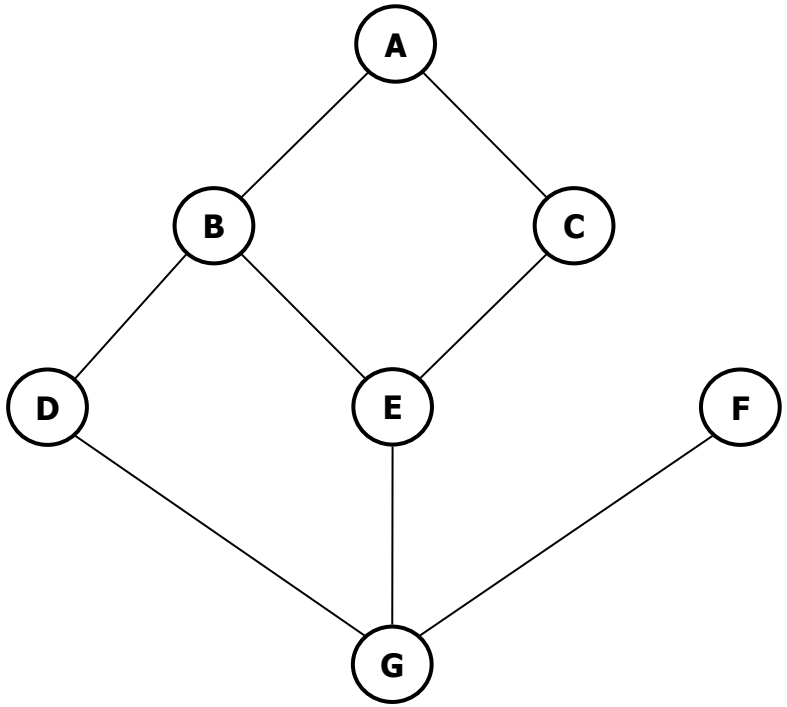
깊이 우선 탐색 (cont'd)

- 깊이 우선 탐색(DFS)의 작동 예 #2

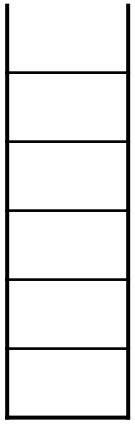


깊이 우선 탐색 (cont'd)

- 깊이 우선 탐색 과정
  - 초기상태
    - 배열 visited를 False로 초기화하고 공백 스택을 생성한다.



〔그래프 G9〕



stack

정점 A B C D E F G

〔0〕〔1〕〔2〕〔3〕〔4〕〔5〕〔6〕

F	F	F	F	F	F	F

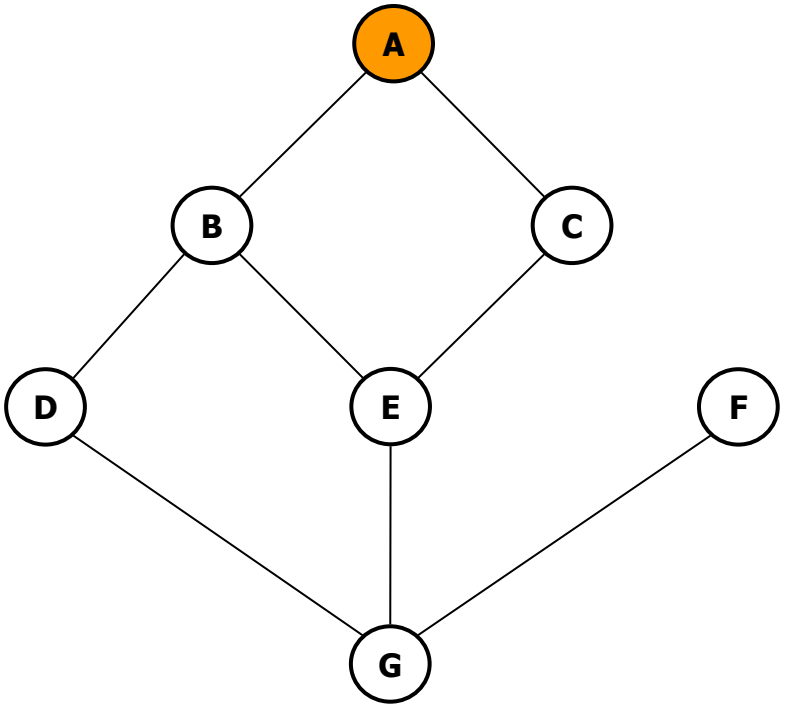
visited

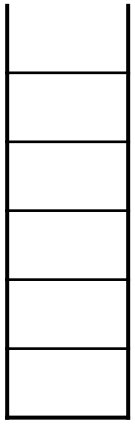
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  1. 정점 A를 시작으로 깊이 우선 탐색 시작

```
visited[A] ← true;
A 방문;
```





stack

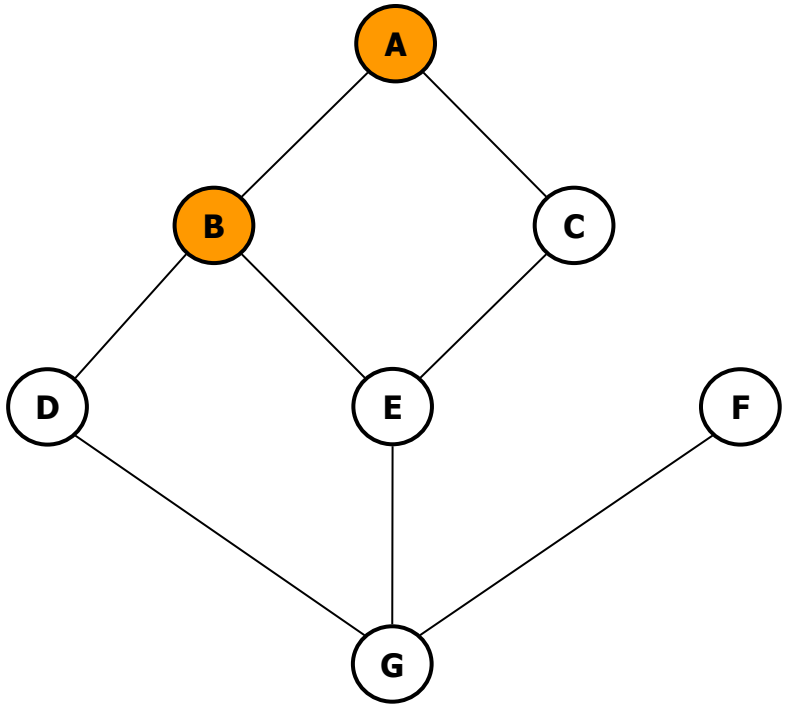
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	F	F	F	F	F	F

visited

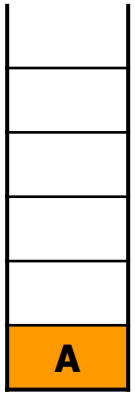
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  2. 정점 A에 방문하지 않은 정점 B,C가 있으므로...
     - A를 스택에 push
     - 인접 정점 B와 C 중에서 오름차순에 따라 B를 선택하여 탐색을 계속한다.




```
push(stack, A);
visited[B] ← true;
B 방문;
```



stack

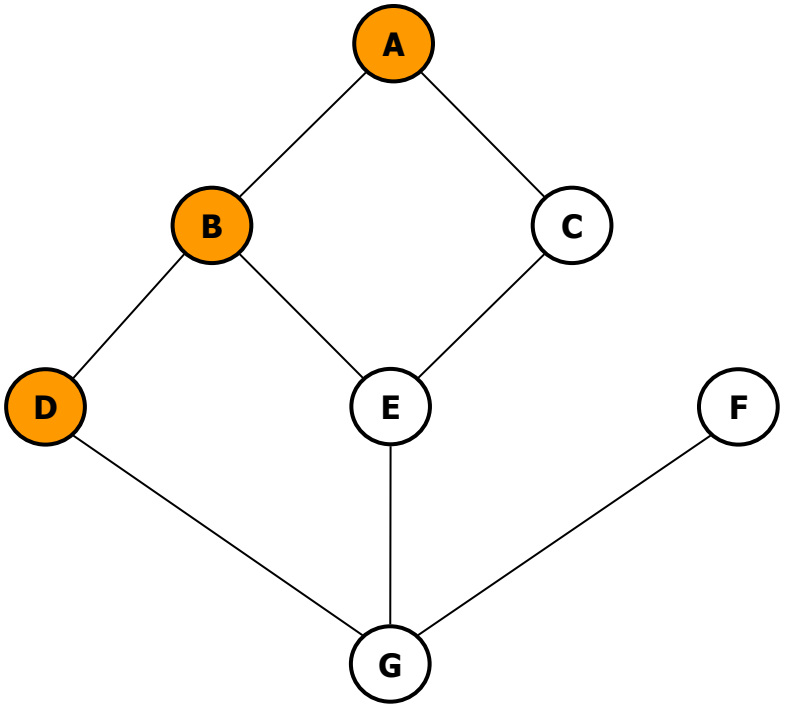
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	F	F	F	F	F

visited

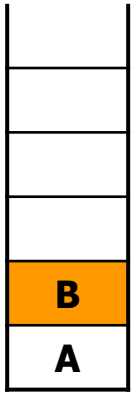
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  3. 정점 B에 방문하지 않은 정점 D,E가 있으므로...
     - B를 스택에 push
     - 인접 정점 D와 E 중에서 오름차순에 따라 D를 선택하여 탐색을 계속한다.




```
push(stack, B);
visited[D] ← true;
B 방문;
```



stack

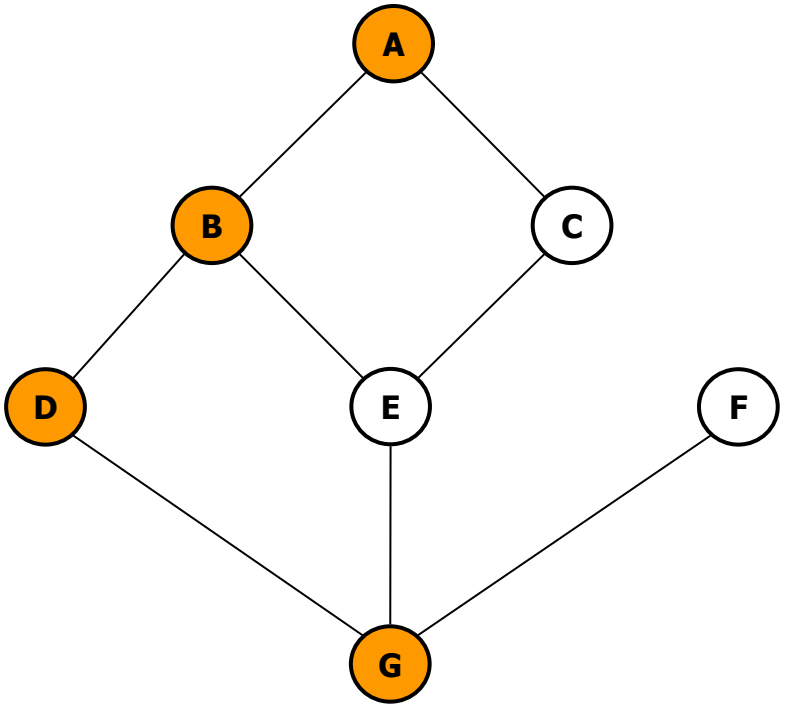
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	F	T	F	F	F

visited

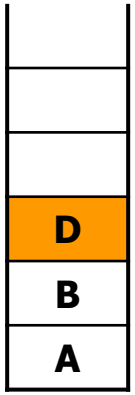
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  4. 정점 D에 방문하지 않은 정점 G가 있으므로...
     - D를 스택에 push
     - 인접 정점 G를 선택하여 탐색을 계속한다.




```
push(stack, D);
visited[G] ← true;
G 방문;
```



stack

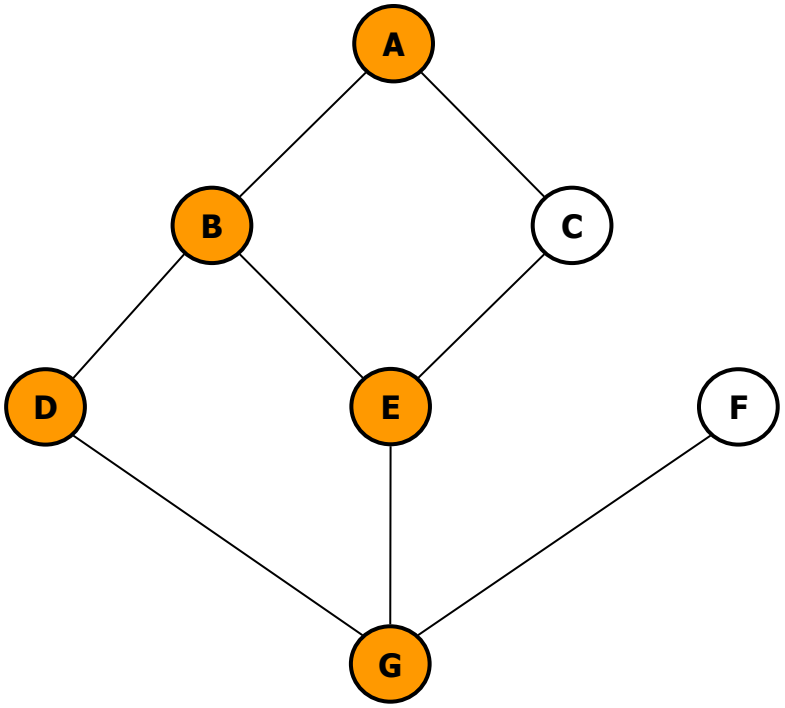
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	F	T	F	F	T

visited

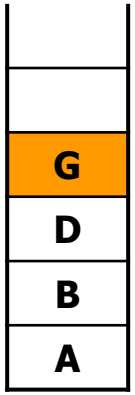
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  5. 정점 G에 방문하지 않은 정점 E,F가 있으므로...
     - G를 스택에 push
     - 인접 정점 E와 F중에서 오름차순에 따라 E를 선택하여 탐색을 계속한다.




```
push(stack, G);
visited[E] ← true;
E 방문;
```



stack

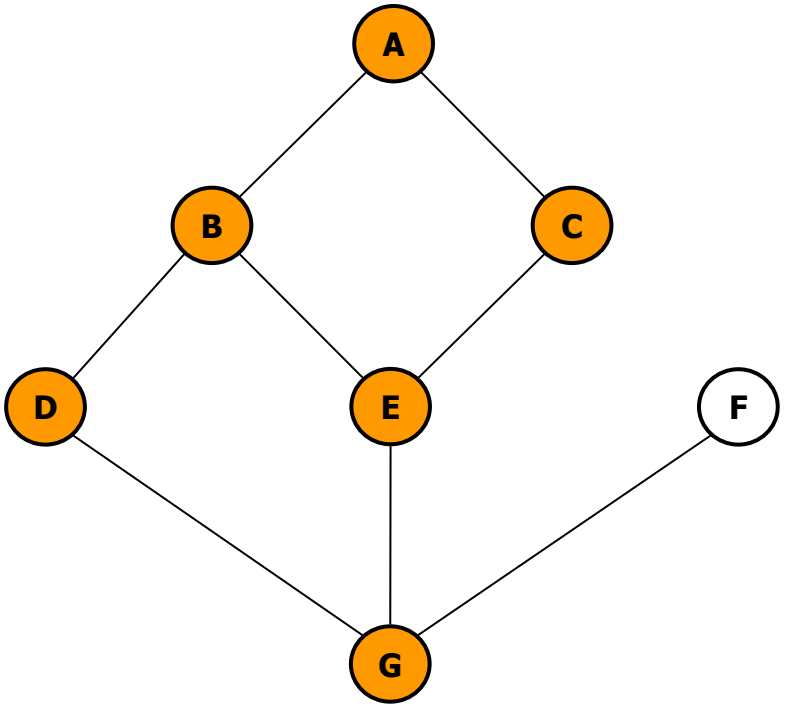
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	F	T	T	F	T

visited

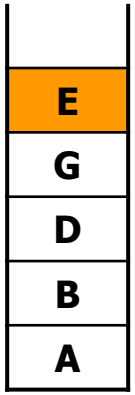
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  6. 정점 E에 방문하지 않은 정점 C가 있으므로...
     - E를 스택에 push
     - 인접 정점 C를 선택하여 탐색을 계속한다.




```
push(stack, E);
visited[C] ← true;
C 방문;
```



stack

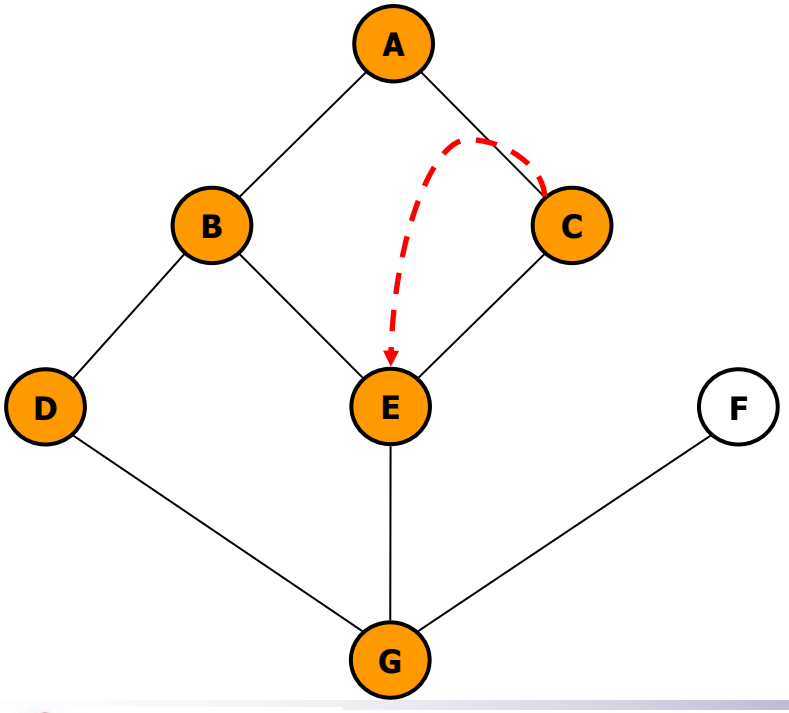
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	F	T

visited

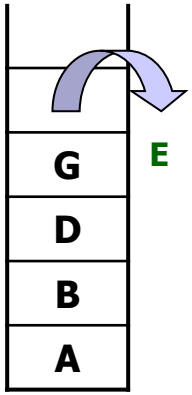
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  7. 정점 C에 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 정점 E에 대해서 방문하지 않은 정점이 있는 확인한다.
     - 정점 E는 방문하지 않은 인접 정점이 없다.




```
pop(stack);
```



stack

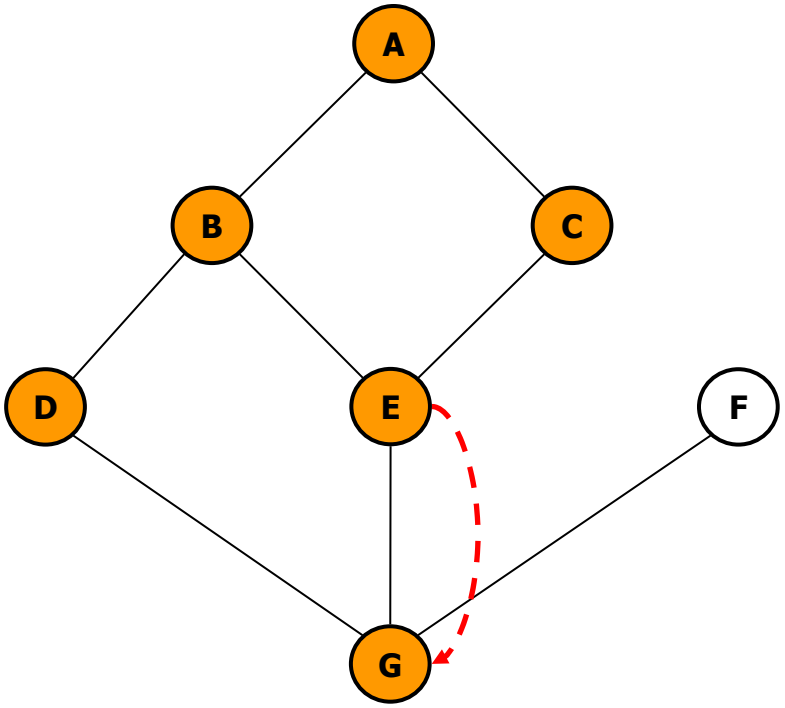
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	F	T

visited

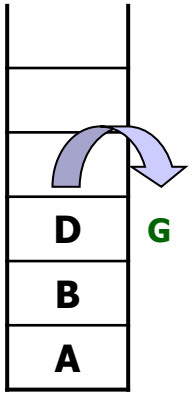
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  8. 다시 스택을 pop하여 받은 정점 G에 대해서 방문하지 않은 인접 정점이 있는지 확인한다.




```
pop(stack);
```



stack

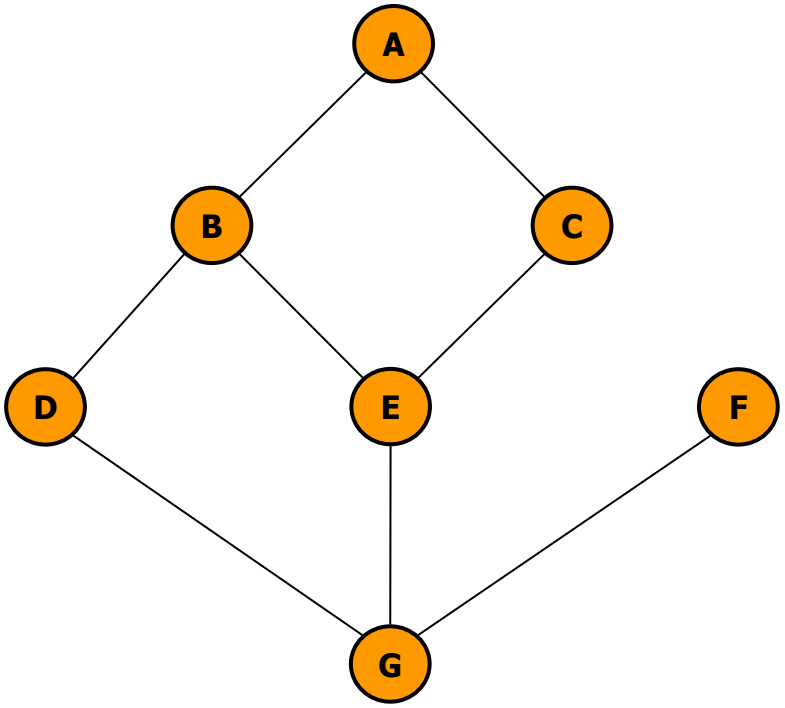
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	F	T

visited

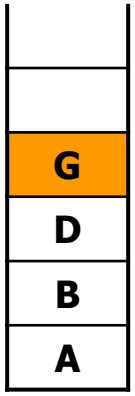
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  9. 정점 G에 방문하지 않은 F가 있으므로 스택에 push하고, 인접 정점 F를 선택하여 탐색을 계속한다.




```
push(stack, G)
visited[F] ← true;
F 방문;
```



stack

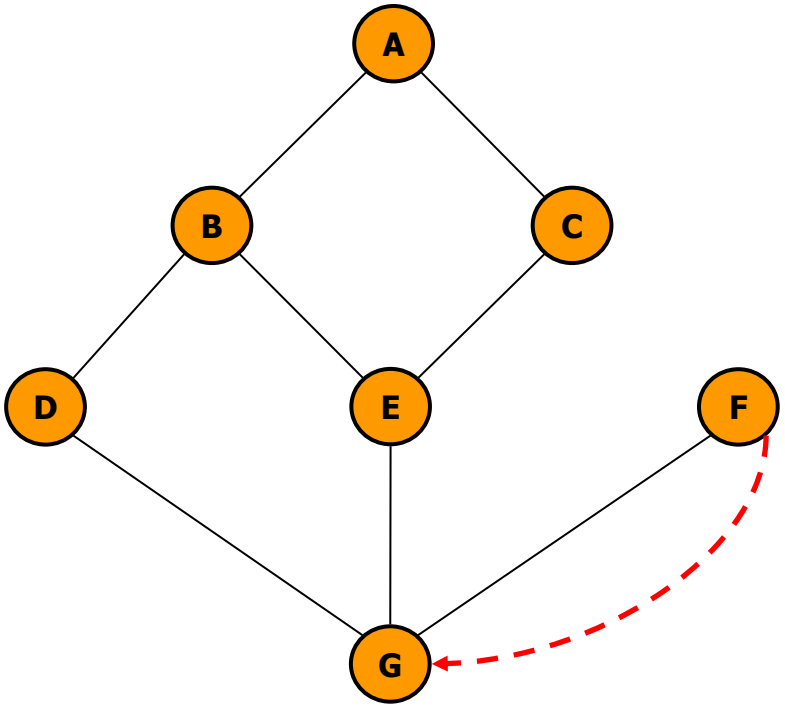
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	T	T

visited

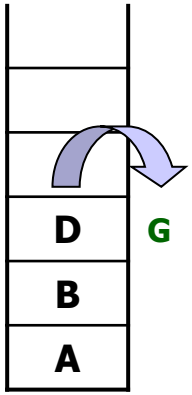
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  10. 정점 F에 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 G에 대해서 방문하지 않은 인접 정점이 있는지 확인한다.




```
pop(stack);
```



stack

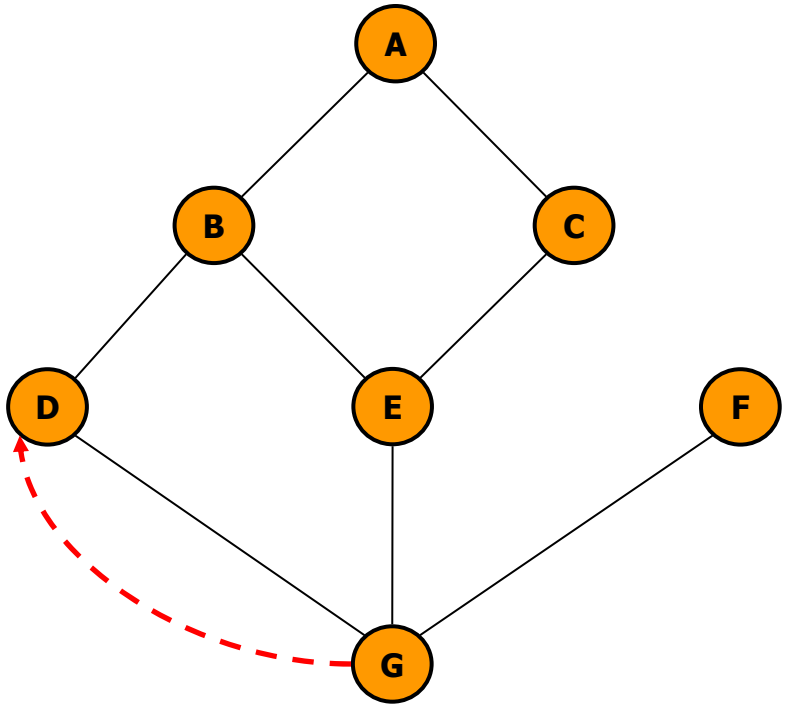
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	T	T

visited

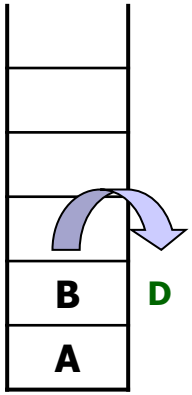
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  11. 정점 G에 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 D에 대해서 방문하지 않은 인접 정점이 있는지 확인한다.




```
pop(stack);
```



stack

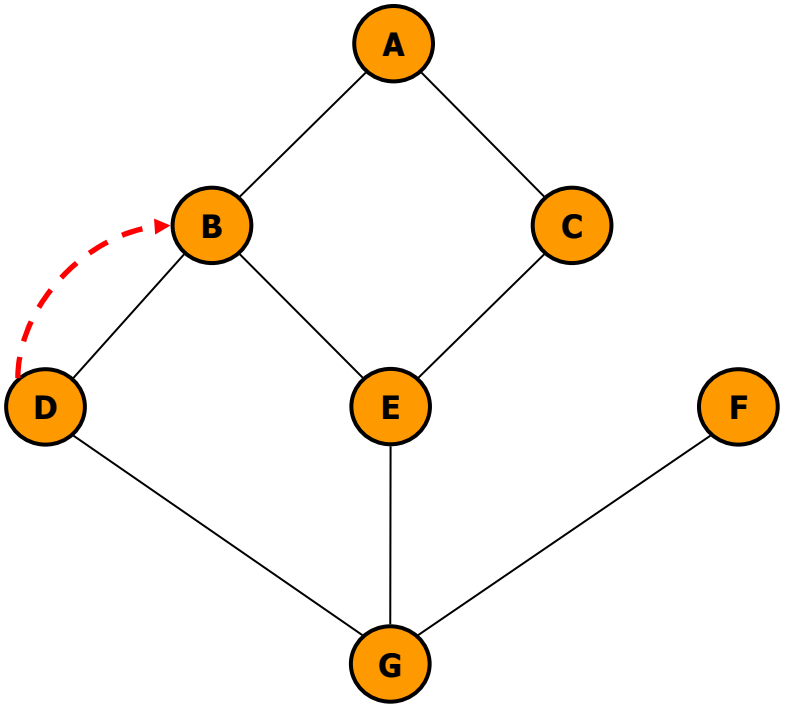
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	T	T

visited

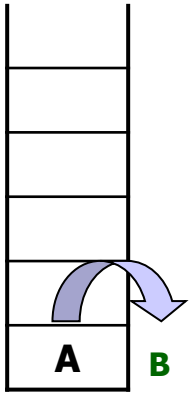
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  12. 정점 D에 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 B에 대해서 방문하지 않은 인접 정점이 있는지 확인한다.




```
pop(stack);
```



stack

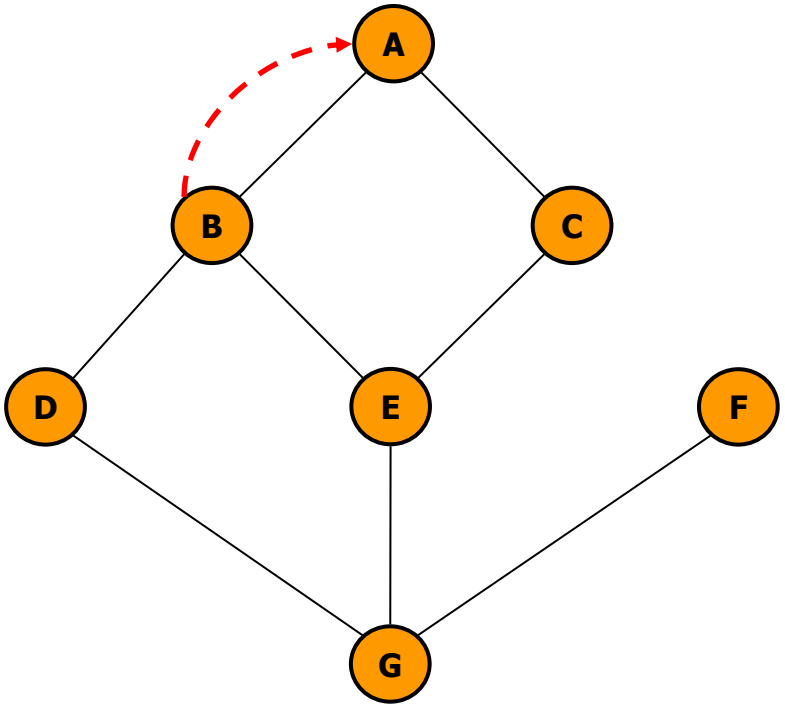
A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	T	T

visited

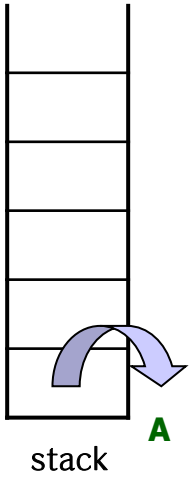
깊이 우선 탐색 (cont'd)

- **깊이 우선 탐색 과정 (cont'd)**

  13. 정점 B에 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 A에 대해서 방문하지 않은 인접 정점이 있는지 확인한다.




```
pop(stack);
```

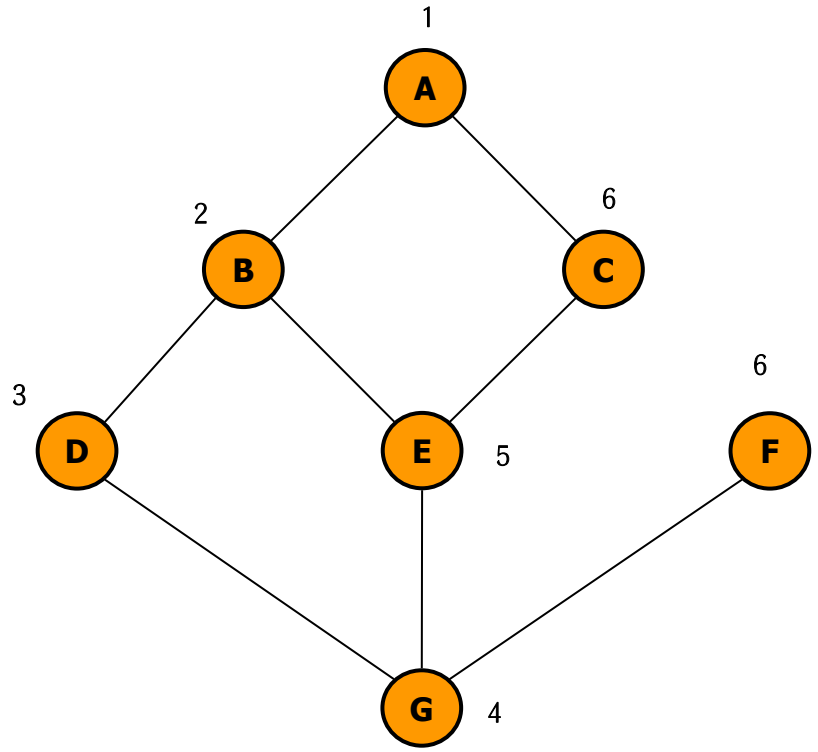


A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	T	T

visited

깊이 우선 탐색 (cont'd)

- 깊이 우선 탐색 과정 (cont'd)
  - 정점 A에 방문하지 않은 인접정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하는데, 스택이 공백이므로 깊이 우선 탐색을 종료한다.



[그래프 G9]

그래프 순회

너비 우선 탐색(BFS)

너비 우선 탐색

너비 우선 탐색

BFS (Breadth First Search)

- 시작 정점으로부터 인접한 정점들을 모두 차례로 방문하고 나서 방문했던 정점을 시작으로 다시 인접한 정점들을 차례로 방문하여 가까운 정점들을 먼저 방문하고 멀리 있는 정점들은 나중에 방문하는 순회 방법

(1) 시작 정점 V를 결정하여 방문한다.

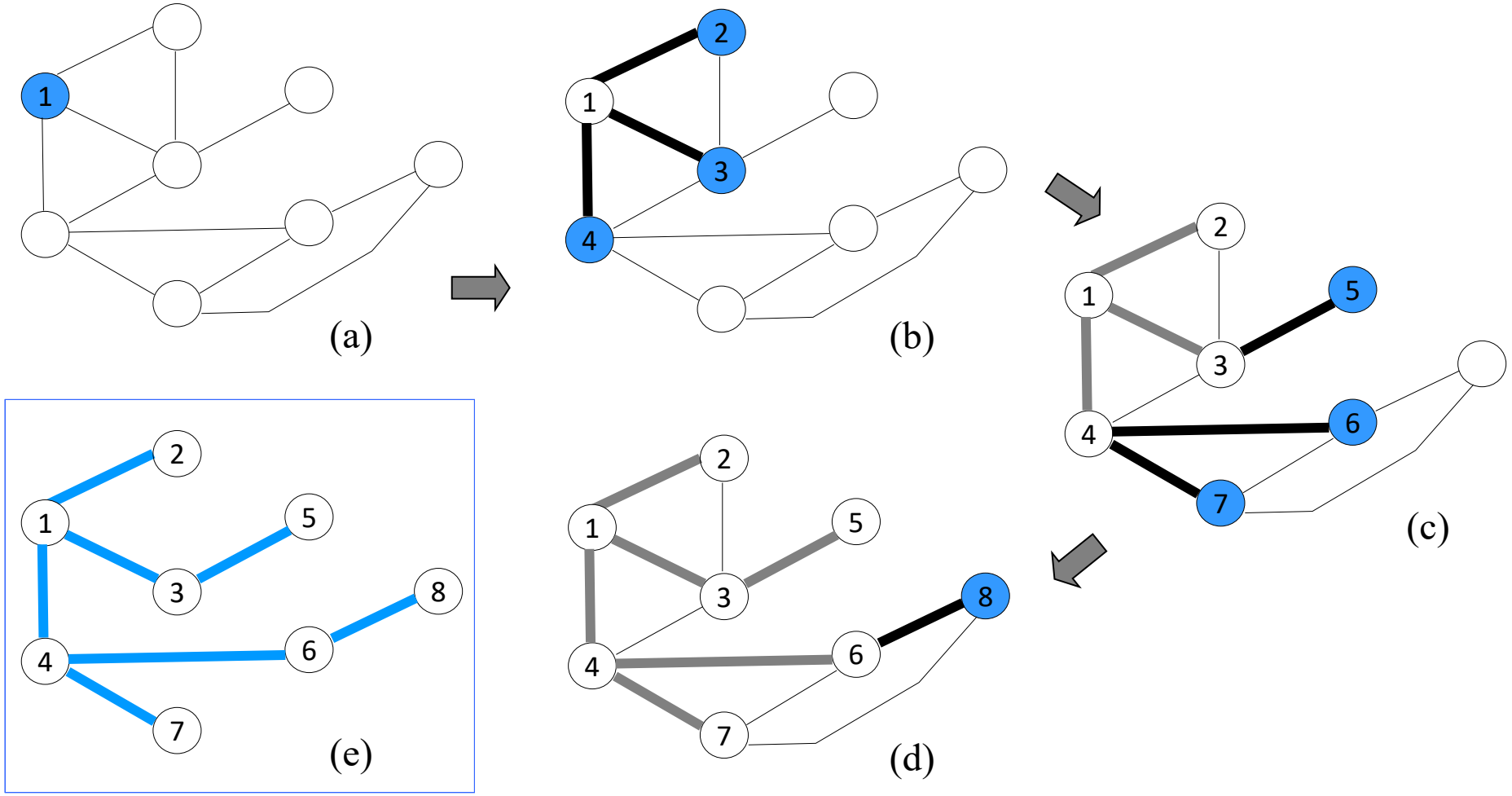
(2) 정점 v에 인접한 정점들 중에서 방문하지 않은 정점을 차례로 방문하면서 큐에 enQueue 한다.

(3) 방문하지 않은 인접한 정점이 없으면 방문했던 정점에서 인접한 정점들을 다시 차례로 방문하기 위해 큐에서 deQueue하여 (2)를 반복한다.

(4) 큐가 공백이 될 때까지 (2)~(3)을 반복한다.

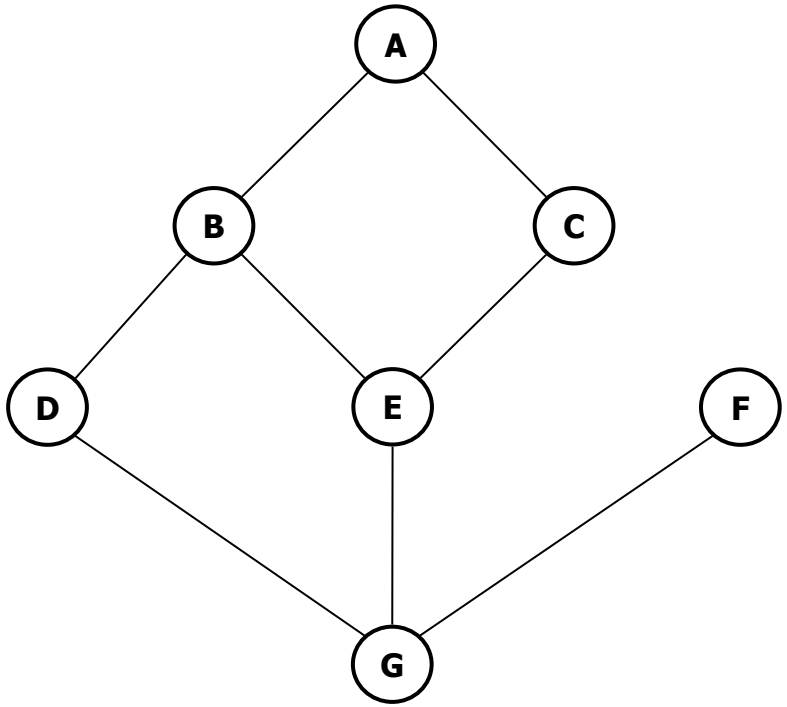
너비 우선 탐색 (cont'd)

- 너비 우선 탐색(BFS)의 작동 예



너비 우선 탐색 (cont'd)

- 너비 우선 탐색 과정
  - 초기상태
    - 배열 visited를 False로 초기화하고 공백 큐를 생성한다.



정점

A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
F	F	F	F	F	F	F

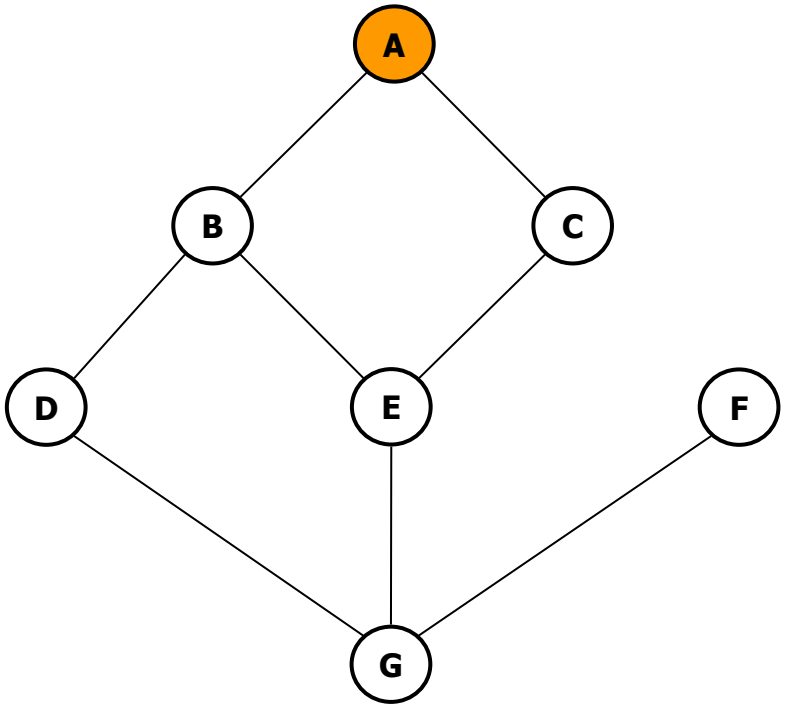
visited

Q

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  1. 정점 A를 시작으로 너비 우선 탐색을 시작한다.




```
visited[A] ← true;
A 방문;
```

정점 A B C D E F G

[0] [1] [2] [3] [4] [5] [6]

T	F	F	F	F	F	F

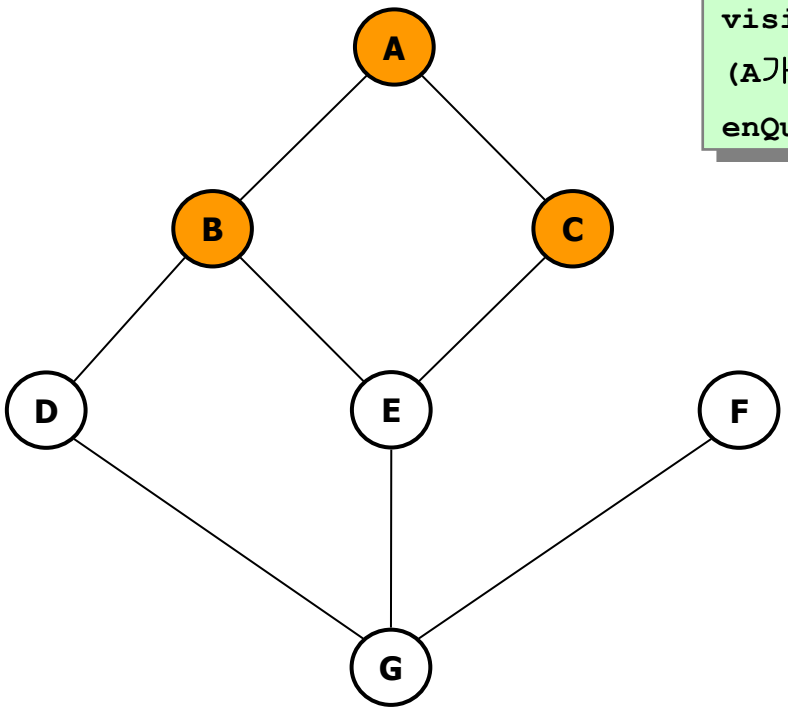
visited

Q

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  2. 정점 A가 방문하지 않은 모든 인접 정점 B, C를 방문하고 큐에 enQueue 한다.




```
visited[(A가 방문하지 않은 인접 정점 B와 C)] ← true;
(A가 방문하지 않은 인접 정점 B와 C) 방문;
enQueue(Q, (A가 방문하지 않은 인접 정점 B와 C));
```

정점 A B C D E F G

[0] [1] [2] [3] [4] [5] [6]

T	T	T	F	F	F	F

visited

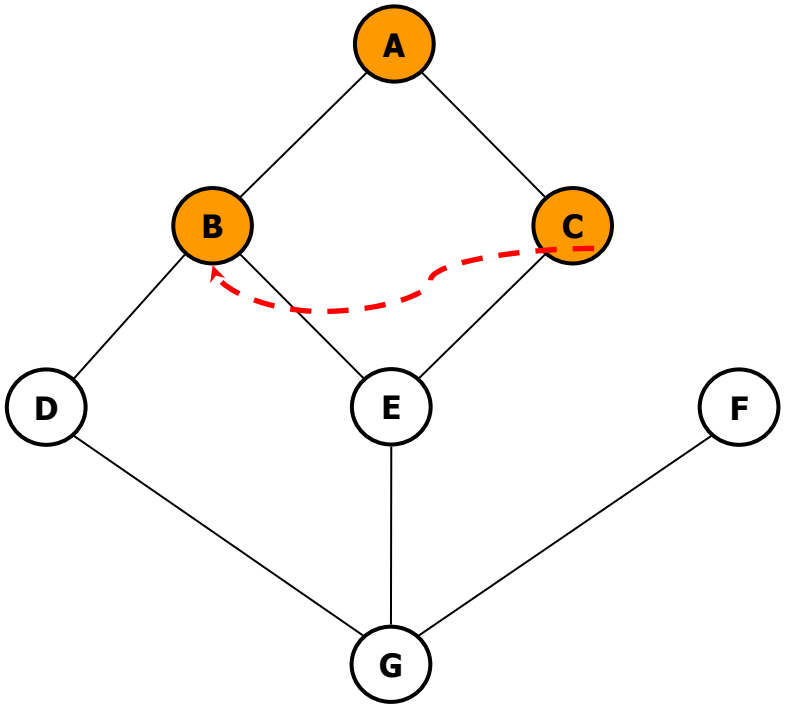
Q

	B	C					

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  3. 정점 A에 대한 인접 정점들을 처리했으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 B를 받는다.




```
v ← deQueue(Q);
```

정점 A B C D E F G

[0] [1] [2] [3] [4] [5] [6]

T	T	T	F	F	F	F

visited

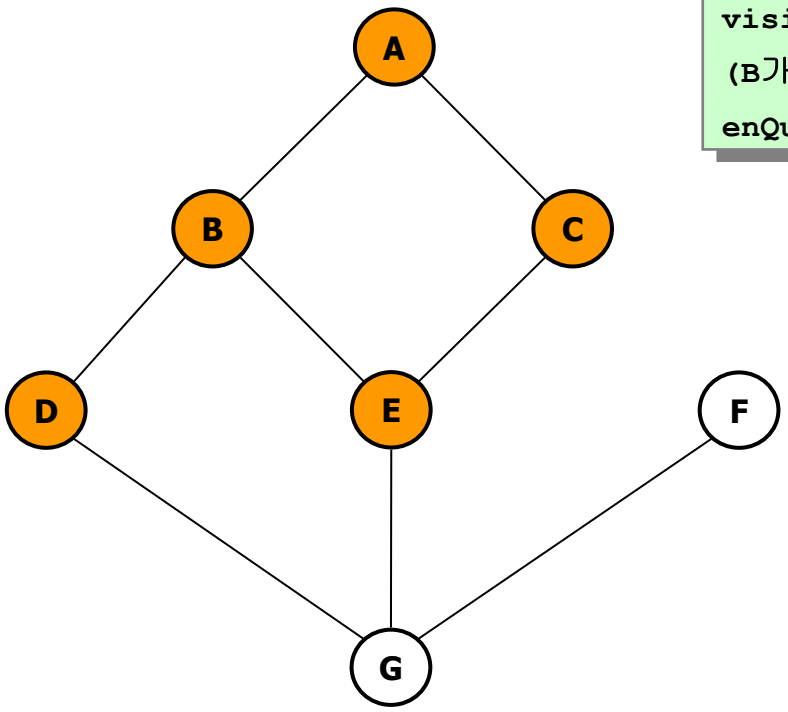
Q

B

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  4. 정점 B가 방문하지 않은 모든 인접 정점 D, E를 방문하고 큐에 enQueue한다.




```
visited[(B가 방문하지 않은 인접 정점 D와 E)] ← true;
(B가 방문하지 않은 인접 정점 D와 E) 방문;
enQueue(Q, (B가 방문하지 않은 인접 정점 D와 E));
```

정점

A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	F	F

visited

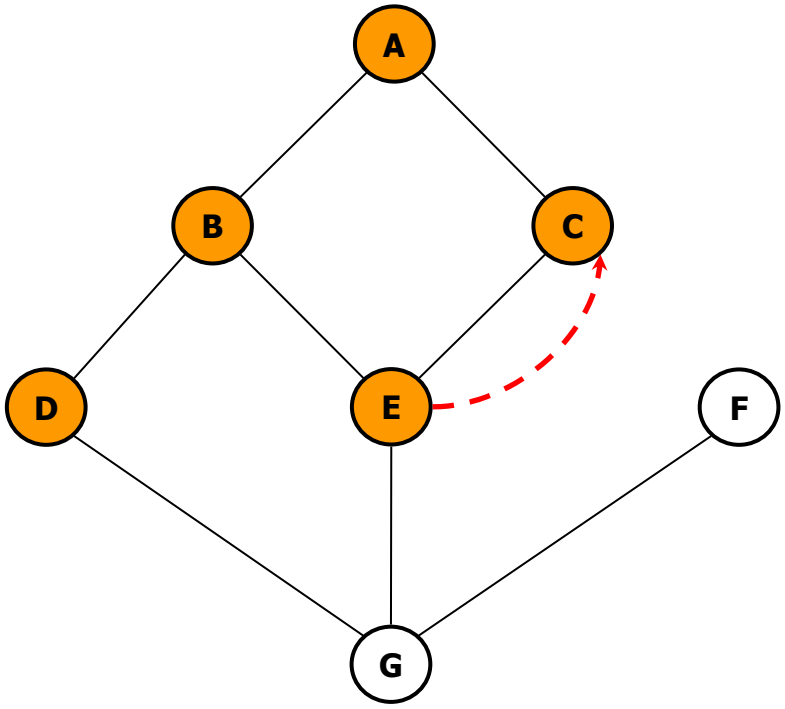
Q

		C	D	E			

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  5. 정점 B에 대한 인접 정점들을 처리했으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 C를 받는다.




```
v ← deQueue(Q);
```

정점 A B C D E F G

[0] [1] [2] [3] [4] [5] [6]

T	T	T	T	T	F	F

visited

Q

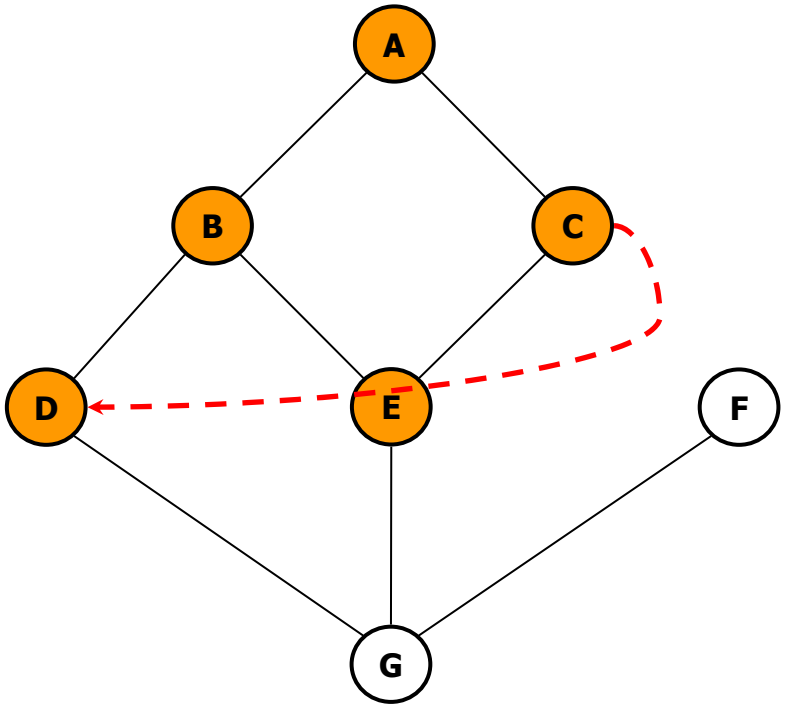
			D	E			

C

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  6. 정점 C에는 방문하지 않은 인접 정점이 없으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 D를 받는다.



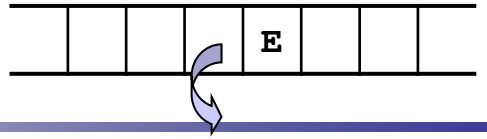

```
v ← deQueue(Q);
```

정점

A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	F	F

visited

Q

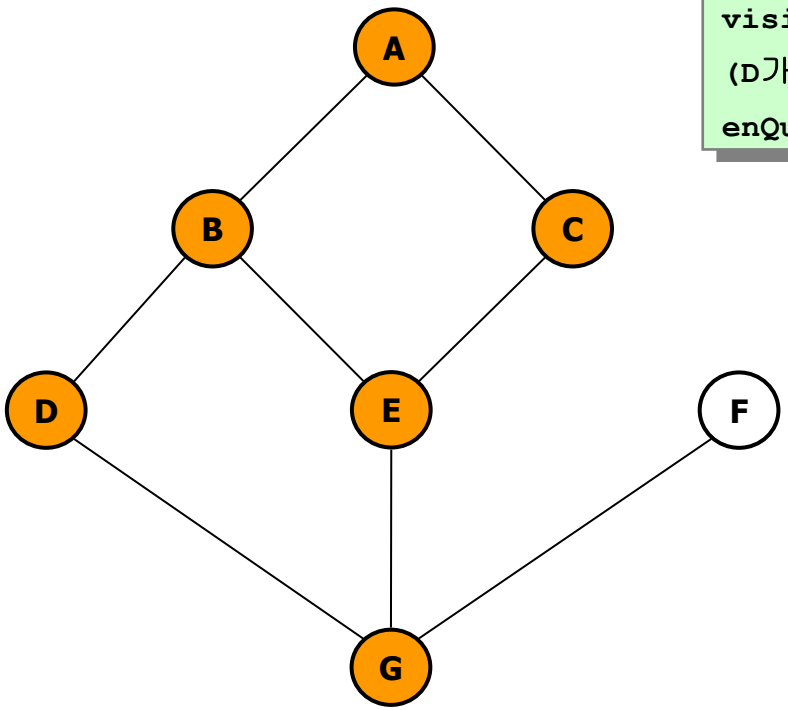


D

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  7. 정점 D가 방문하지 않은 인접 정점 G를 방문하고 큐에 enQueue 한다.




```
visited[(D가 방문하지 않은 인접 정점 G)] ← true;
(D가 방문하지 않은 인접 정점 G) 방문;
enQueue(Q, (D가 방문하지 않은 인접 정점 G));
```

정점 A B C D E F G

[0] [1] [2] [3] [4] [5] [6]

T	T	T	T	T	F	T

visited

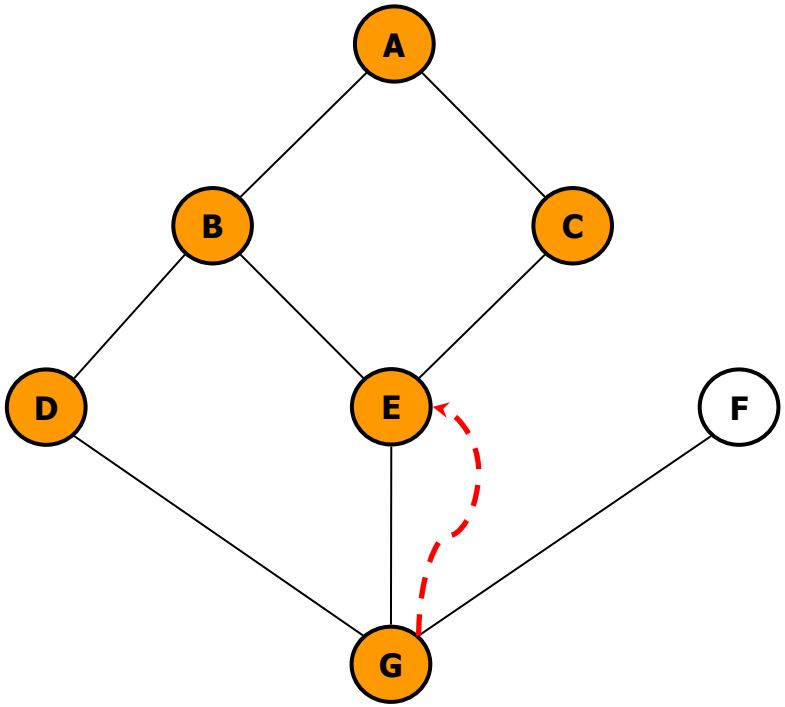
Q

				E	G		

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  8. 정점 D에 대한 인접 정점들을 처리했으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 E를 받는다.




```
v ← deQueue(Q);
```

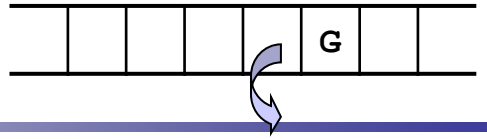
정점 A B C D E F G

[0] [1] [2] [3] [4] [5] [6]

T	T	T	T	T	F	T

visited

Q

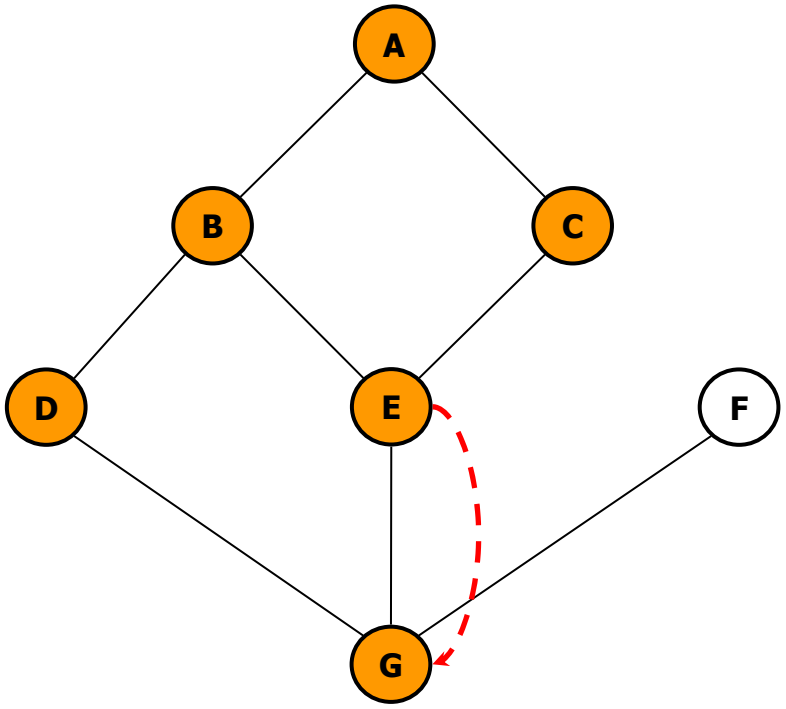


E

너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  9. 정점 E에는 방문하지 않은 인접 정점이 없으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 G를 받는다.




```
v ← deQueue(Q);
```

정점

A	B	C	D	E	F	G
[0]	[1]	[2]	[3]	[4]	[5]	[6]
T	T	T	T	T	F	T

visited

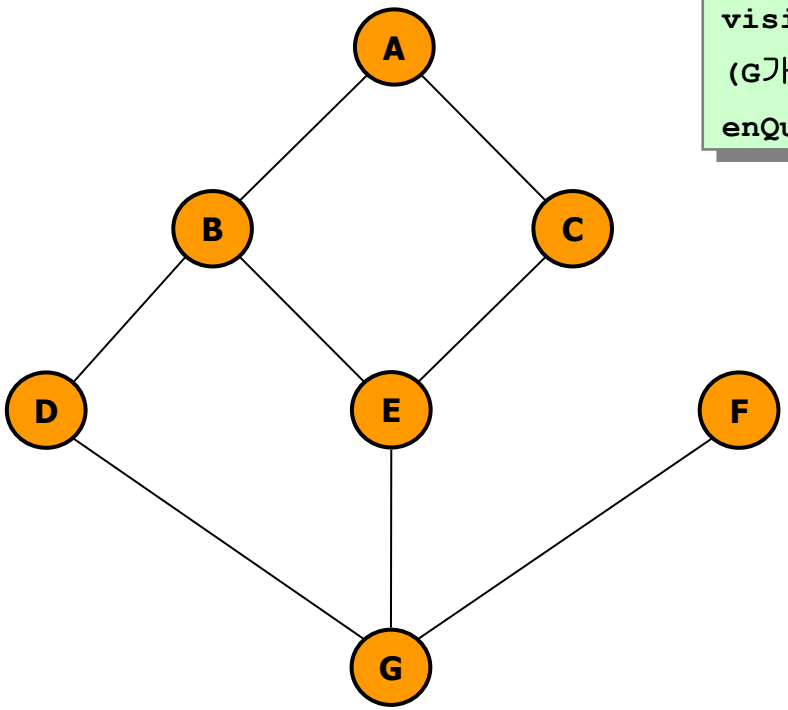
Q

G

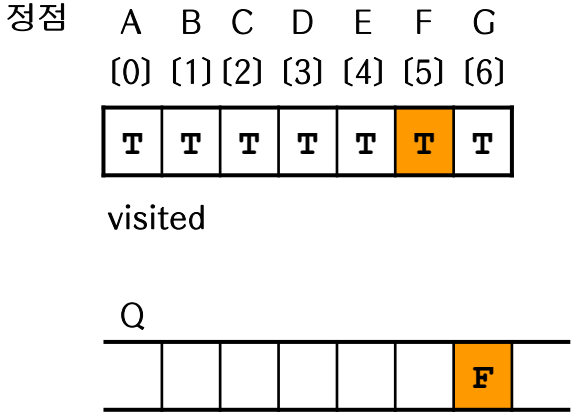
너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  10. 정점 G가 방문하지 않은 인접 정점 F를 방문하고 큐에 enQueue한다.



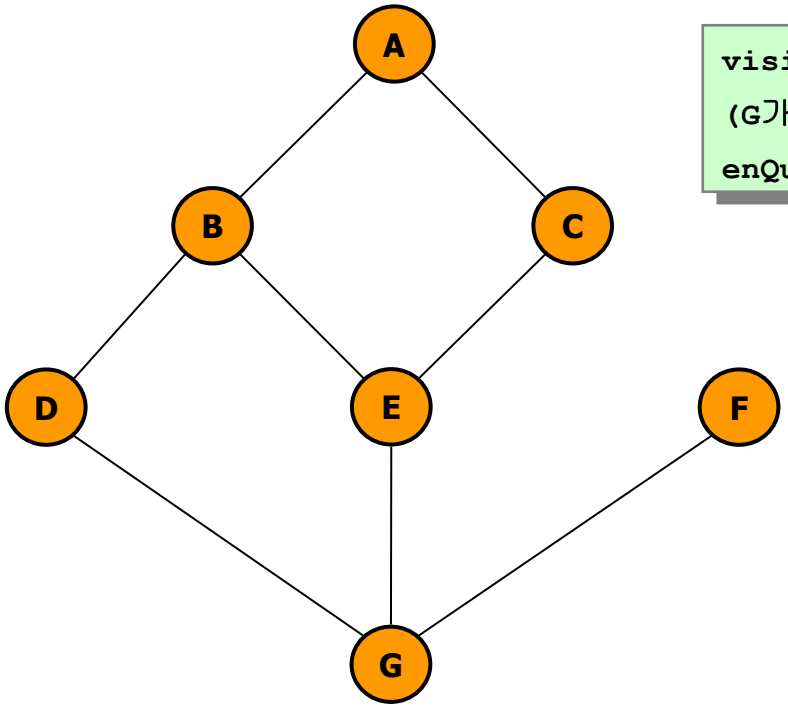

```
visited[(G가 방문하지 않은 인접 정점 F)] ← true;
(G가 방문하지 않은 인접 정점 F) 방문;
enQueue(Q, (G가 방문하지 않은 인접 정점 F));
```



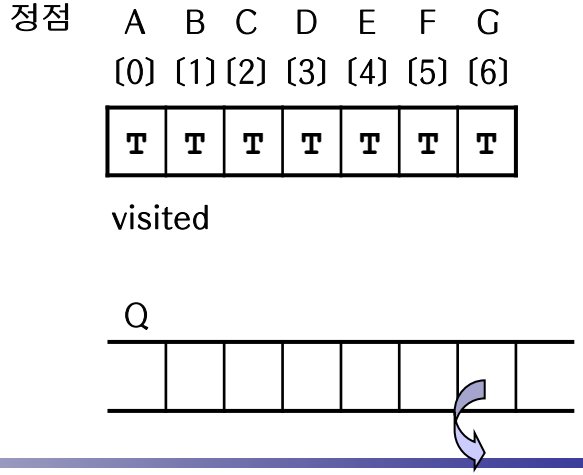
너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**

  11. 정점 G에 대한 인접 정점들을 처리했으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 F를 받는다.

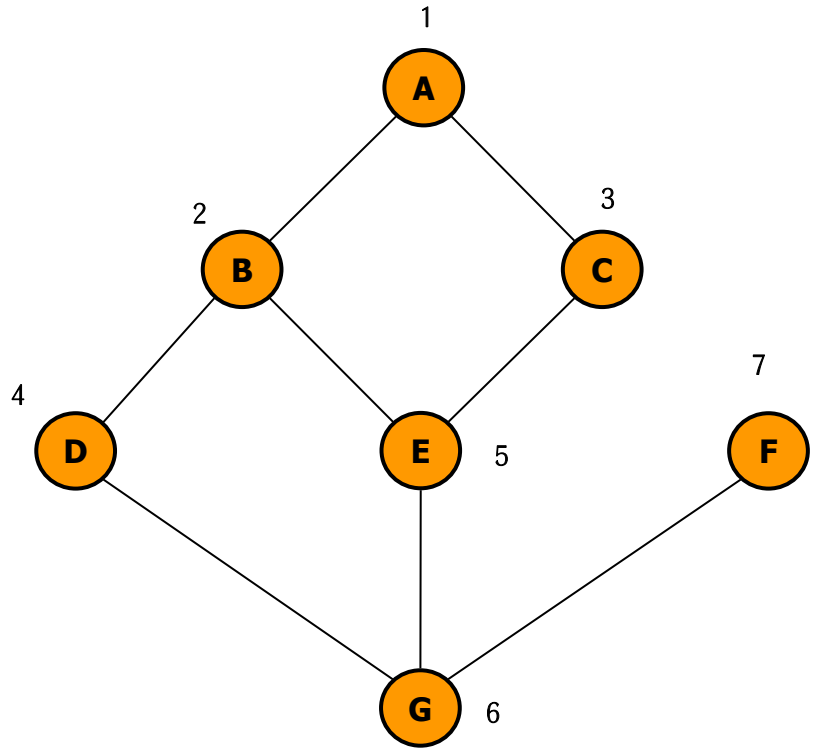



```
visited[(G가 방문하지 않은 인접 정점 F)] ← true;
(G가 방문하지 않은 인접 정점 F) 방문;
enQueue(Q, (G가 방문하지 않은 인접 정점 F));
```



너비 우선 탐색 (cont'd)

- **너비 우선 탐색 과정 (cont'd)**
  - 정점 F는 모든 인접 정점을 방문했으므로 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하는데, 큐가 공백이므로 너비 우선 탐색을 종료한다.



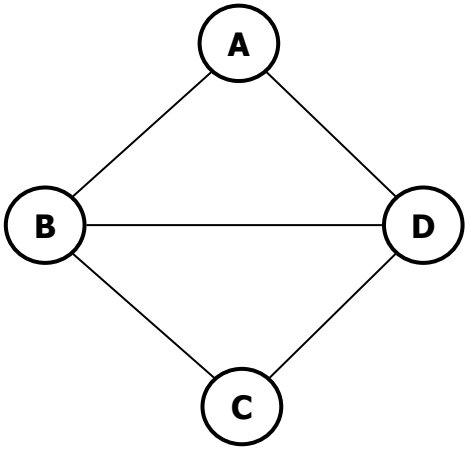
〔 그래프 G9 〕

가중치 그래프

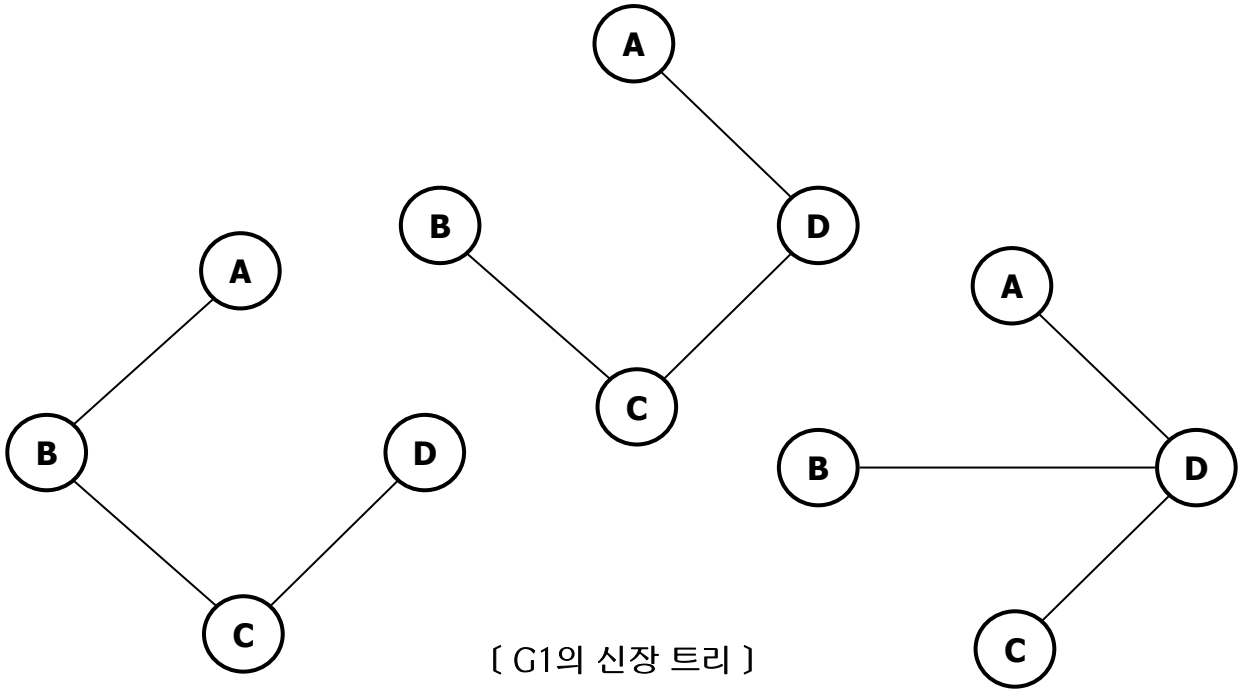
- 그래프의 이해와 표현
- 그래프 순회
- **가중치 그래프**
  - 최소 신장 트리
  - 최단 경로

신장 트리 (1/3)

- **신장 트리**(spanning tree)
  - n개의 정점으로 이루어진 무방향 그래프 G에서 n개의 모든 정점과 n-1개의 간선으로 만들어진 트리
    - 그래프의 관점에서 트리는 사이클이 없는 단순 연결 그래프



〔 그래프 G1 〕

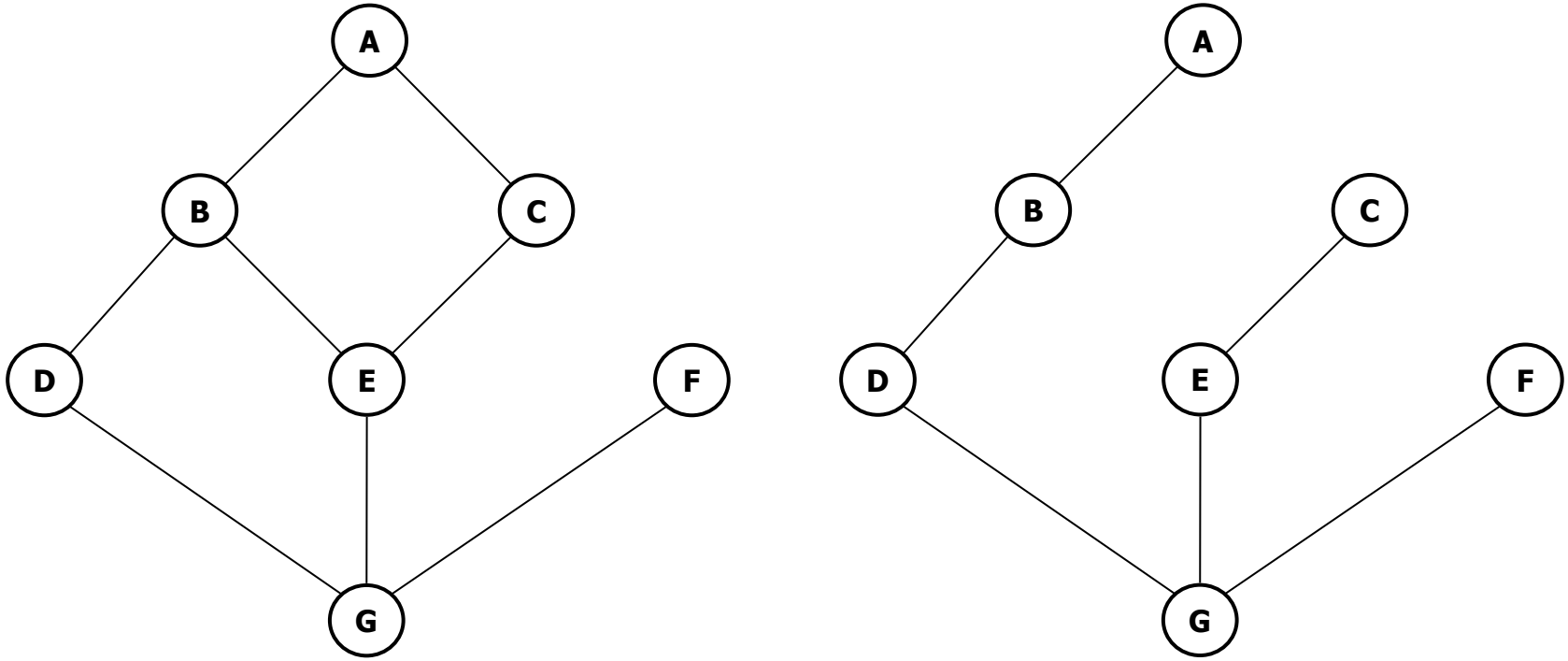


〔 G1의 신장 트리 〕

〔 그래프 G1과 신장 트리의 예 〕

신장 트리 (2/3)

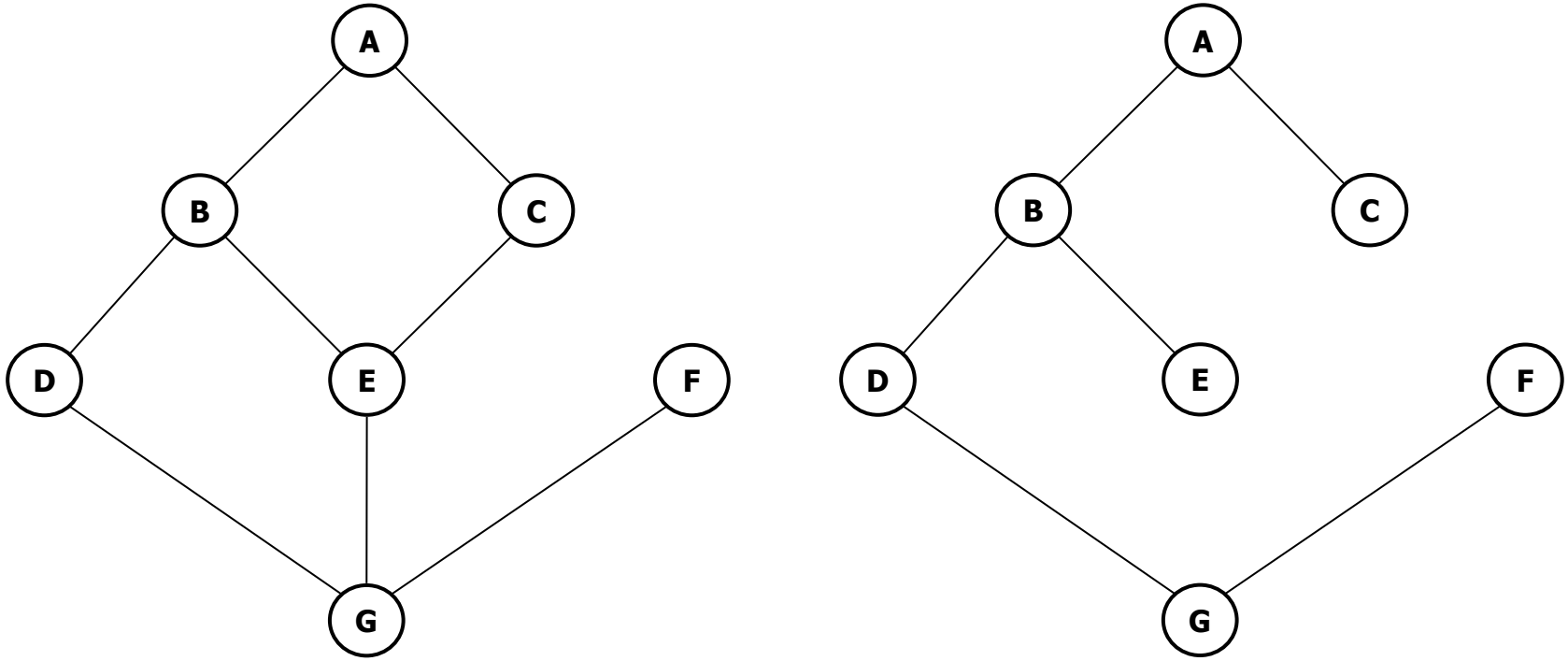
- **신장 트리** : 깊이 우선 신장 트리
  - 깊이 우선 신장 트리(depth first spanning tree)
    - 깊이 우선 탐색을 이용하여 생성된 신장 트리



〔 그래프 G9와 깊이 우선 신장 트리 〕

신장 트리 (3/3)

- **신장 트리** : 너비 우선 신장 트리
  - 너비 우선 신장 트리(breadth first spanning tree)
    - 너비 우선 탐색을 이용하여 생성된 신장 트리



〔 그래프 G9와 너비 우선 신장 트리 〕

최소 신장 트리

- **최소 신장 트리(minimum 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 신장 트리를 만드는 알고리즘
    - Kruskal 알고리즘
    - Prim 알고리즘

신장 트리

최소 신장 트리

Prim 알고리즘

최소 신장 트리 (cont'd)

- **프림(Prim) 알고리즘**
  - 프림 알고리즘은 간선을 정렬하지 않고 하나의 정점에서 시작하여 트리를 확장해 나가는 방법

(1) 그래프 G에서 시작 정점을 선택한다.

(2) 선택한 정점에 부속된 모든 간선 중에서 가중치가 가장 작은 간선을 연결하여 트리를 확장한다.

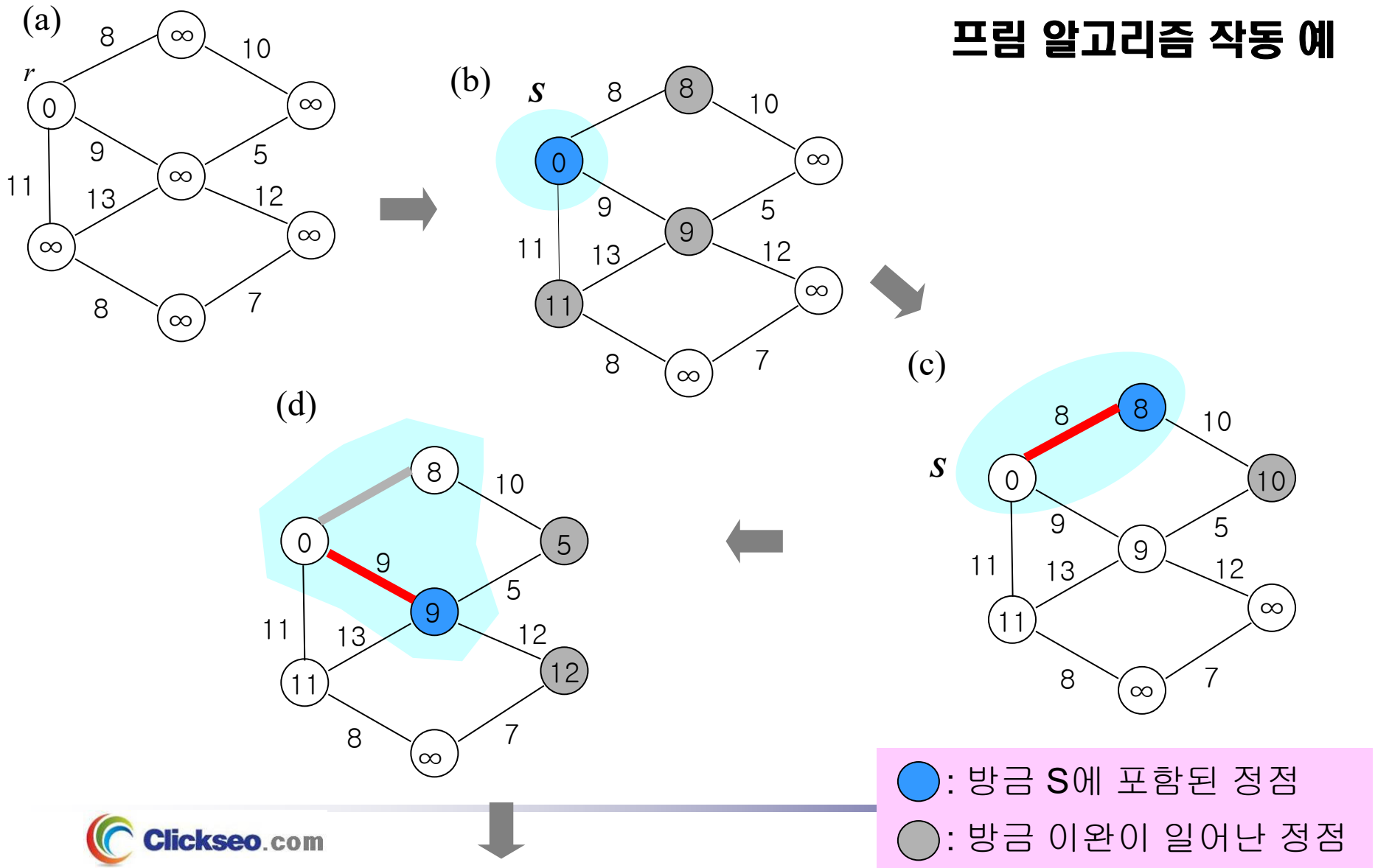
(3) 이전에 선택한 정점과 새로 선택한 정점에 부속된 모든 간선 중에서 가중치가 가장 작은 간선을 삽입하는데, 사이클을 형성하는 간선은 삽입할 수 없으므로 그 다음으로 가중치가 작은 간선을 선택한다.

(4) 그래프 G에 n-1개의 간선을 삽입할 때까지 (3)을 반복한다.

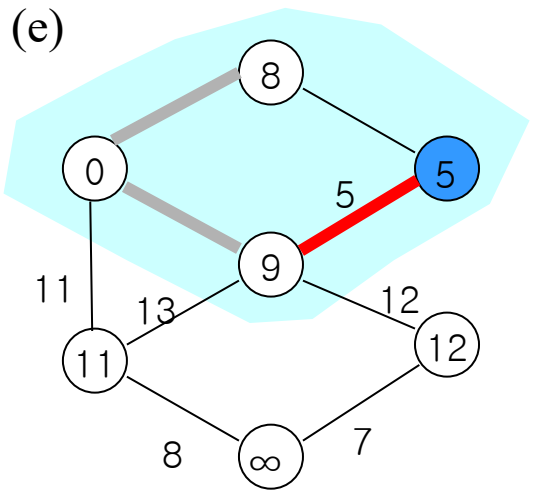
(5) 그래프 G에 간선이 n-1개가 되면 최소 비용 신장 트리가 완성된다.

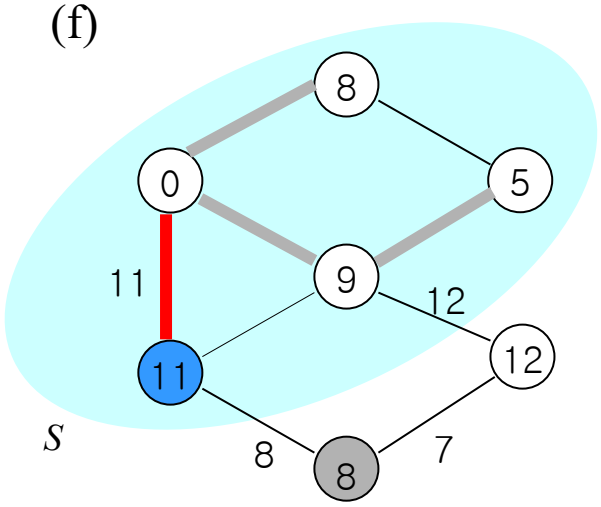
최소 신장 트리 (cont'd)

프림 알고리즘 작동 예

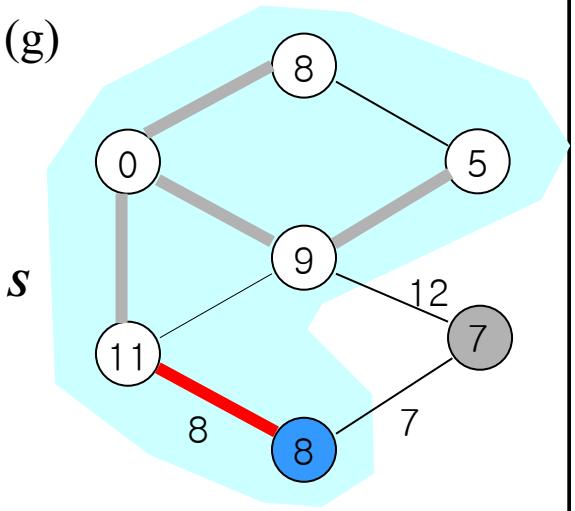


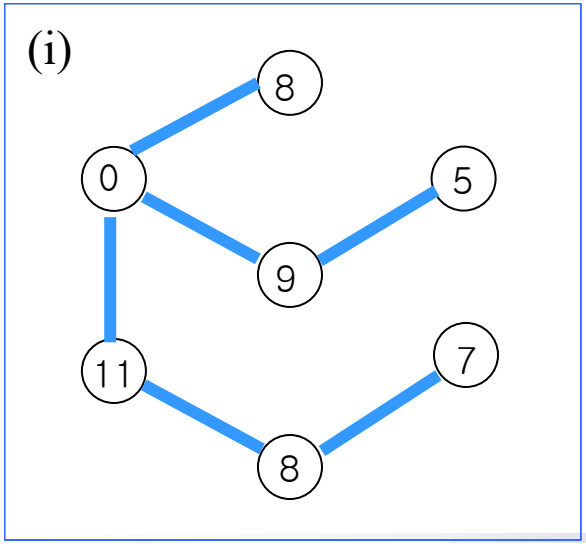
최소 신장 트리 (cont'd)

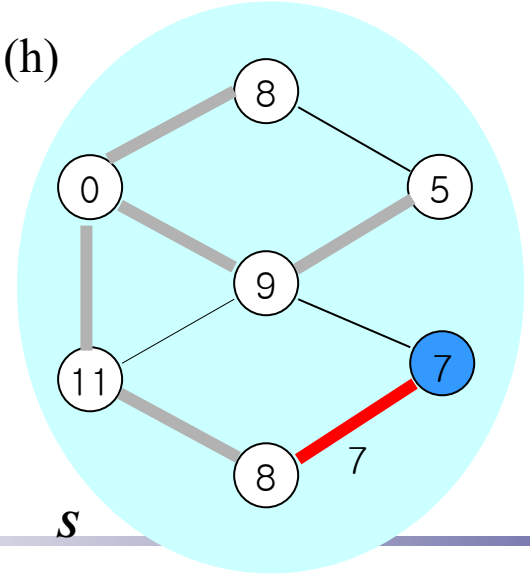




프림 알고리즘 작동 예







신장 트리

최소 신장 트리

Kruskal 알고리즘

최소 신장 트리 (cont'd)

- **크루스칼 (Kruskal) 알고리즘 1**
  - 가중치가 높은 간선을 제거하면서 최소 비용 신장 트리를 만드는 방법

(1) 그래프 G의 모든 간선을 가중치에 따라 내림차순으로 정리한다.

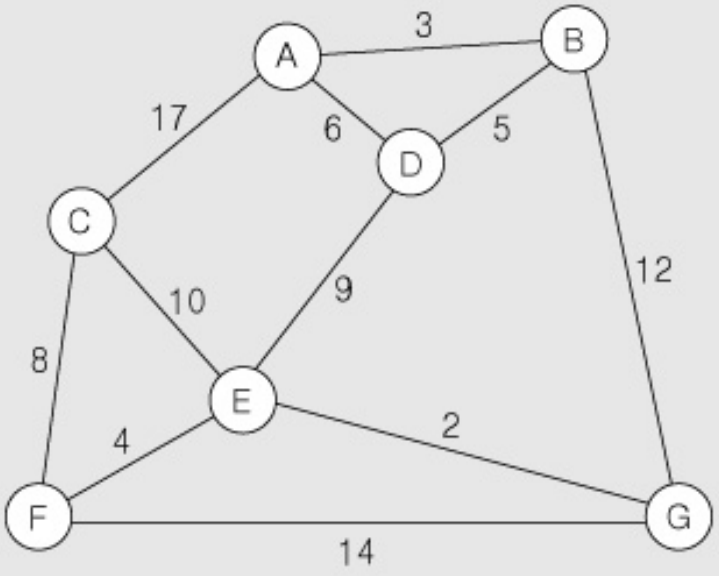
(2) 그래프 G에서 가중치가 가장 높은 간선을 제거하는데, 이때 정점을 그래프에서 분리시키는 간선은 제거할 수 없으므로 그 다음으로 가중치가 높은 간선을 제거한다.

(3) 그래프 G에 n-1 개의 간선만 남을 때까지 (2)를 반복한다.

(4) 그래프 G에 n-1 개의 간선이 남게 되면 최소 비용 신장 트리가 완성된다.

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**
  - **초기 상태**
    - 그래프 G10의 간선을 가중치에 따라서 내림차순 정렬



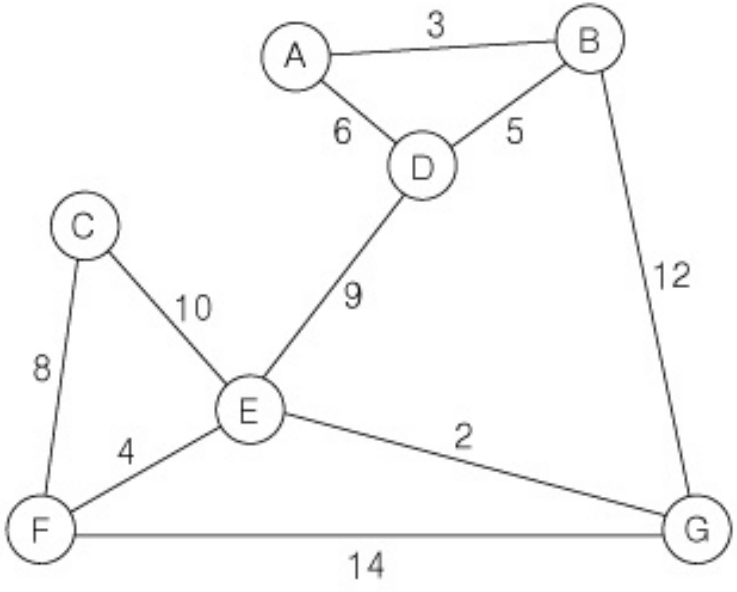
가중치	간선
17	(A, C)
14	(F, G)
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

간선의 수 : 11개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**

  1. 가중치가 가장 큰 간선 (A,C) 제거



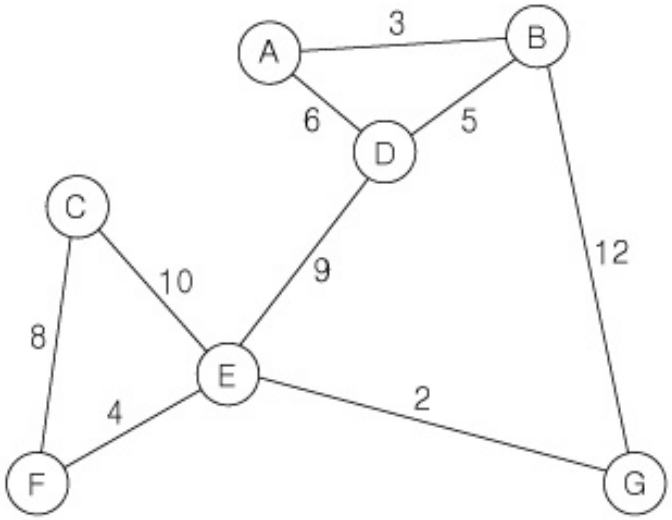
가중치	간선
~~17~~	~~(A, C)~~
14	(F, G)
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

간선의 수 : 10개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**

  2. 남은 간선 중에서 가중치가 가장 큰 간선 (F,G) 제거



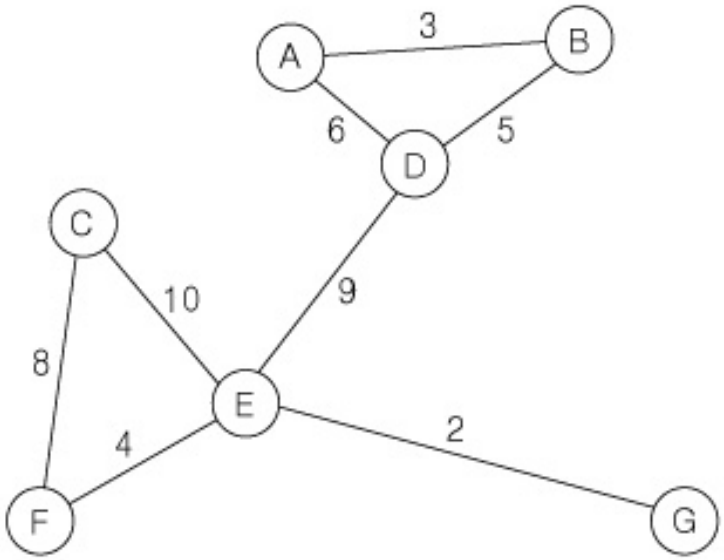
가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

간선의 수 : 9개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**

  3. 남은 간선 중에서 가중치가 가장 큰 간선 (B,G) 제거



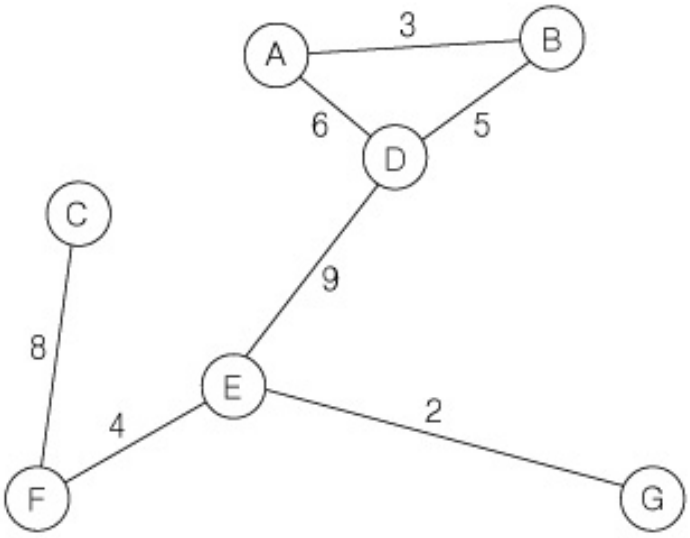
가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

간선의 수 : 8개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**

  4. 남은 간선 중에서 가중치가 가장 큰 간선 (B,G) 제거



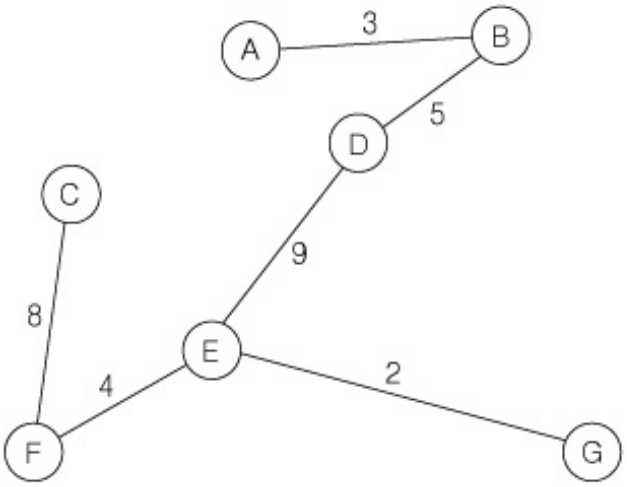
가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
~~10~~	~~(C, E)~~
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

간선의 수 : 7개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**

  5. 남은 간선 중에서 가중치가 가장 큰 간선 (D,E)를 제거하면, 그래프가 분리되어 단절 그래프가 되므로, 그 다음으로 가중치가 큰 간선 (C,F)를 제거해야 한다. 그런데 간선 (C,F)를 제거하면 정점 C가 분리되므로 제거할 수 없으므로, 다시 그 다음으로 가중치가 큰 간선 (A,D)를 제거한다.

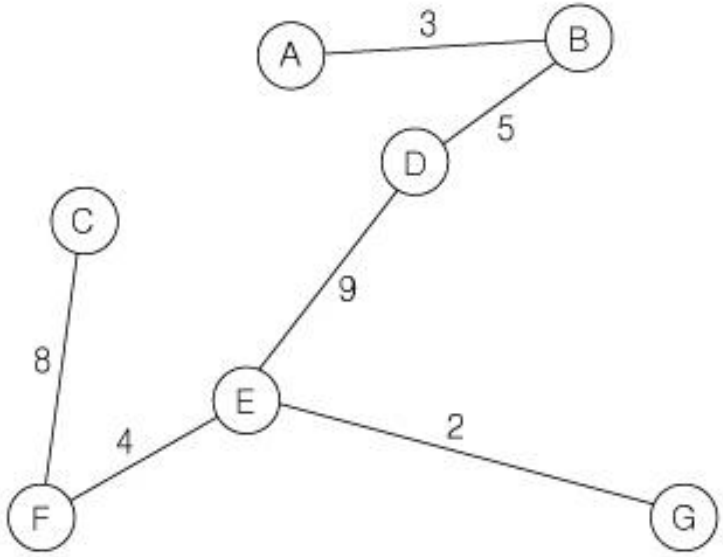


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
~~10~~	~~(C, E)~~
9	(D, E)
8	(C, F)
~~6~~	~~(A, D)~~
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

간선의 수 : 6개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 1 작동 예 (cont'd)**
  - Kruskal 알고리즘 1 을 이용하여 완성된 G10의 최소 비용 신장 트리



최소 신장 트리 (cont'd)

- **크루스칼 (Kruskal) 알고리즘 2**
  - 가중치가 낮은 간선을 삽입하면서 최소 비용 신장 트리를 만드는 방법

(1) 그래프 G의 모든 간선을 가중치에 따라 오름차순으로 정리한다.

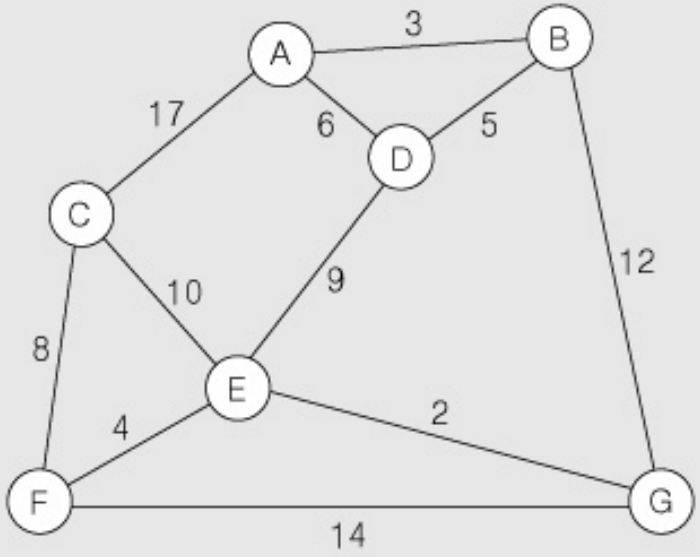
(2) 그래프 G에서 가중치가 가장 작은 간선을 삽입하는데, 이때 사이클을 형성하는 간선은 삽입할 수 없으므로 그 다음으로 가중치가 작은 간선을 삽입한다.

(3) 그래프 G에 n-1 개의 간선만 삽입할 때까지 (2)를 반복한다.

(4) 그래프 G에 간선이 n-1 개가 되면 최소 비용 신장 트리가 완성된다.

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**
  - **초기 상태**
    - 그래프 G10의 간선을 가중치에 따라서 오름차순 정렬



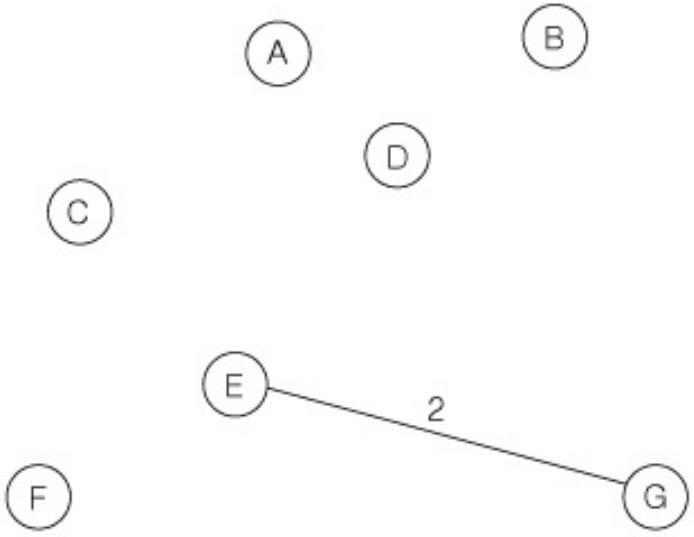
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

간선의 수 : 11개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**

  1. 가중치가 가장 작은 간선 (E,G) 삽입



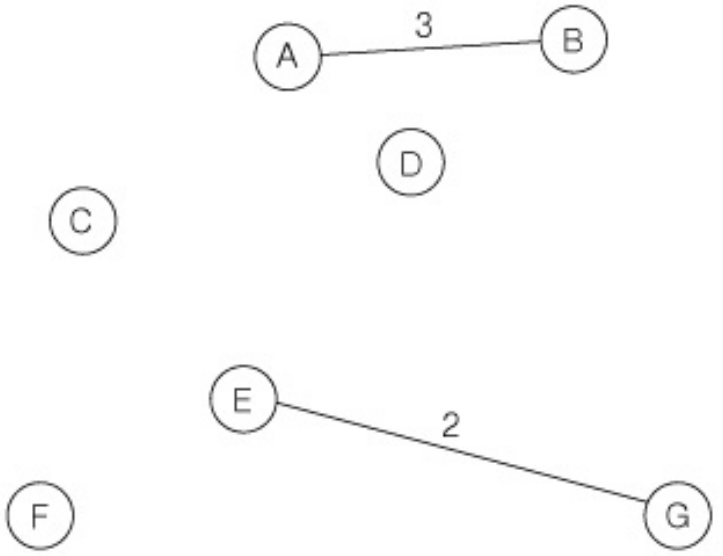
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

삽입한 간선의 수 : 1개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**

  2. 나머지 간선 중에서 가중치가 가장 작은 간선 (A,B) 삽입



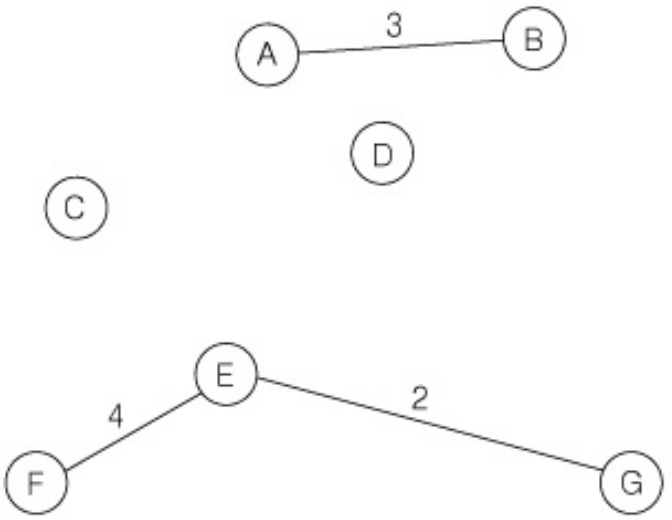
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

삽입한 간선의 수 : 2개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**

  3. 나머지 간선 중에서 가중치가 가장 작은 간선 (E,F) 삽입



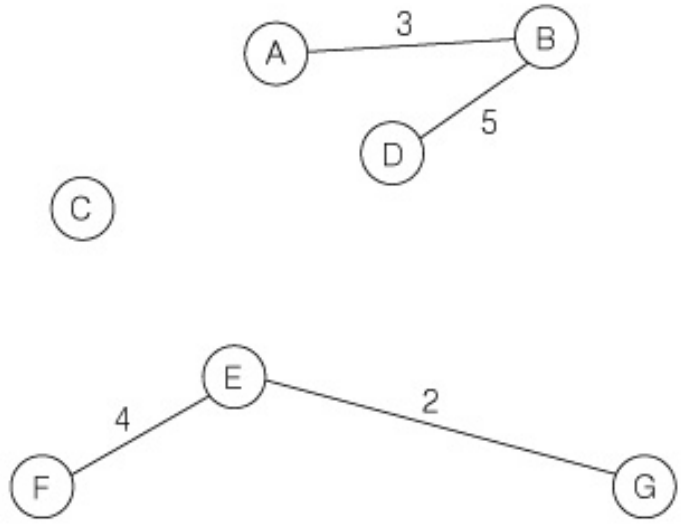
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

삽입한 간선의 수 : 3개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**

  4. 나머지 간선 중에서 가중치가 가장 작은 간선 (B,D) 삽입



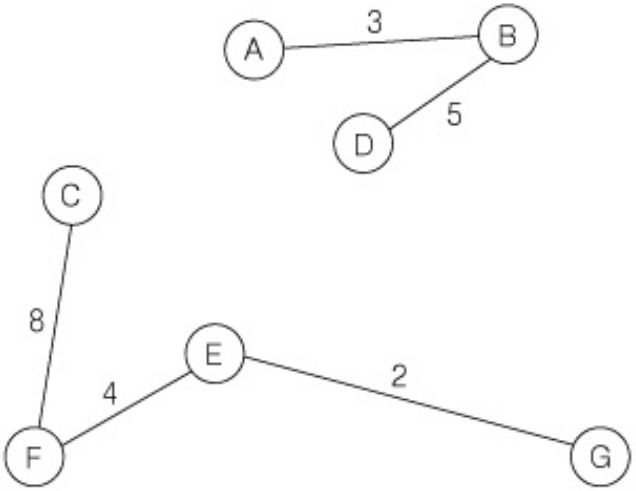
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

삽입한 간선의 수 : 4개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**

  5. 나머지 간선 중에서 가중치가 가장 작은 간선 (A,D)를 삽입하면 A-B-D의 사이클이 생성되므로 삽입할 수 없다. 그 다음으로 가중치가 가장 작은 간선 (C,F) 삽입



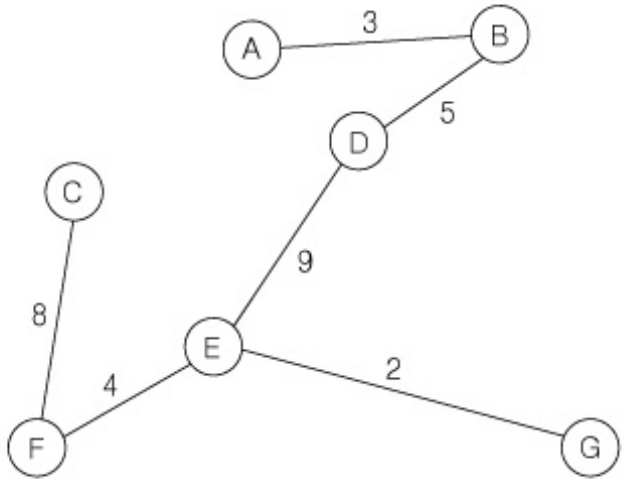
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

삽입한 간선의 수 : 5개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**

  6. 나머지 간선 중에서 가중치가 가장 작은 간선 (D,E) 삽입
     - 현재 삽입한 간선의 수가 6개 이므로 알고리즘 수행을 종료하고 신장 트리 완성

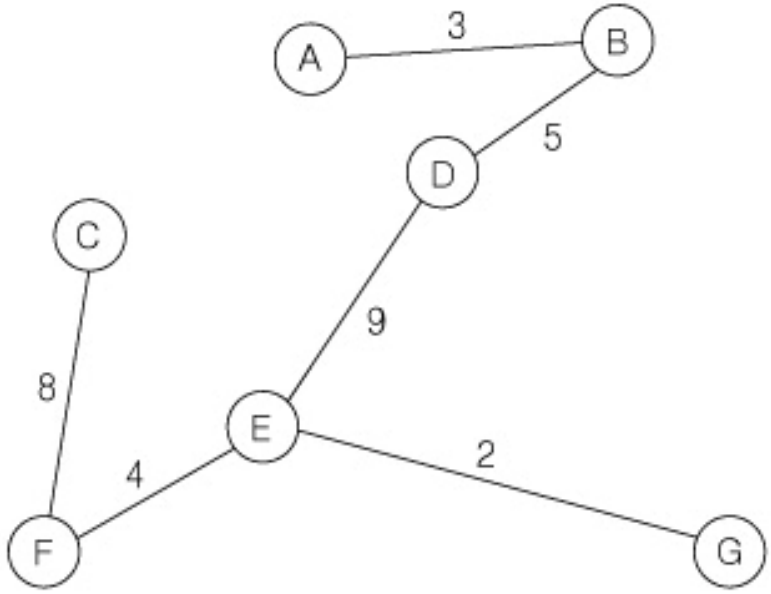


가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

삽입한 간선의 수 : 6개

최소 신장 트리 (cont'd)

- **크루스칼 알고리즘 2 작동 예 (cont'd)**
  - Kruskal 알고리즘 2 를 이용하여 완성된 G10의 최소 비용 신장 트리



참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, “C++로 구현하는 자료구조와 알고리즘”, 한티에듀, 2020.

[2] 주우석, “IT CookBook, C・C++ 로 배우는 자료구조론”, 한빛아카데미, 2019.

[3] 이지영, “C 로 배우는 쉬운 자료구조”, 한빛아카데미, 2022.

[4] 문병로, “IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법”(개정판), 개정판, 한빛아카데미, 2018.

[5] Richard E. Neapolitan, 도경구 역, “알고리즘 기초”, 도서출판 홍릉, 2017.

[6] “프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문”, 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

[7] “IT CookBook, 쉽게 배우는 자료구조 with 파이썬”, 문병로, 한빛아카데미, 2022.

[8] “이것이 취업을 위한 코딩 테스트다 with 파이썬”, 나동빈, 한빛미디어, 2020.