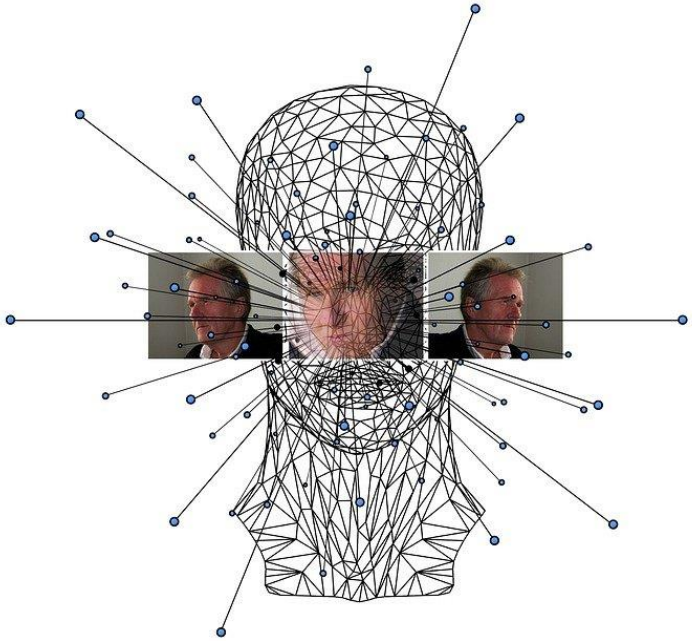


7 그래프

1. 그래프의 구조

❖ 그래프(graph)

- 연결되어 있는 원소 사이의 다:다 관계를 표현하는 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

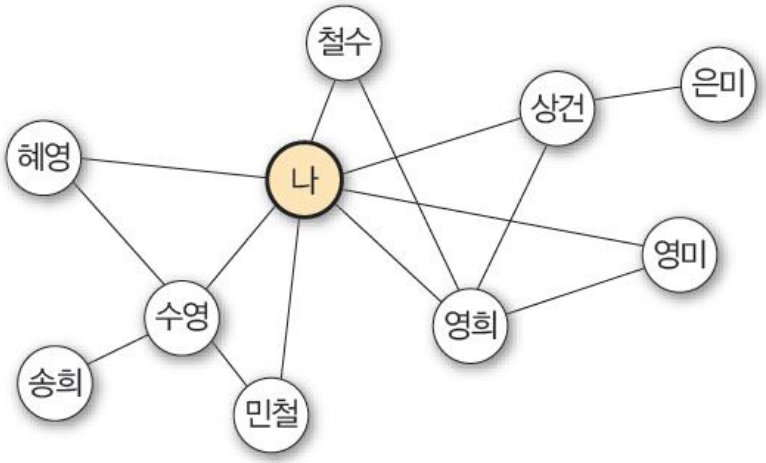
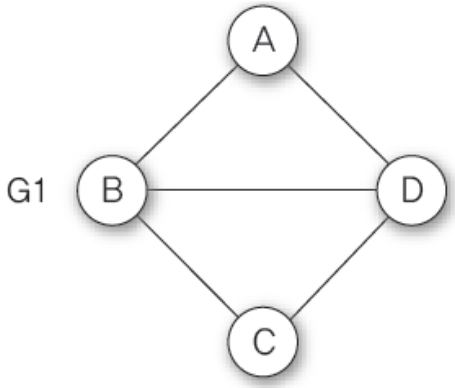


그림 8-1 그래프의 사용 예 : 버스 노선도, 인맥 지도

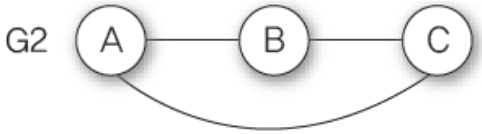
1. 그래프의 구조

❖ 그래프의 종류

- 무방향 그래프(undirected graph)
  - 두 정점을 연결하는 간선에 방향이 없는 그래프
  - 정점 V_i와 정점 V_j을 연결하는 간선을 (V_i, V_j)로 표현
    - (V_i, V_j)와 (V_j, V_i)는 같은 간선을 의미



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



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

그림 8-3 무방향 그래프의 예

1. 그래프의 구조

- 방향 그래프(directed graph) , 다이그래프(digraph)
  - 간선에 방향이 있는 그래프
  - 정점 V_i에서 정점 V_j를 연결하는 간선 즉, $V_i \rightarrow V_j$를 $<V_i, V_j>$로 표현
    - V_i를 꼬리(tail), V_j를 머리(head)라고 한다.
    - $<V_i, V_j>$와 $<V_j, V_i>$는 서로 다른 간선을 의미

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>}

그림 8-4 방향 그래프의 예

1. 그래프의 구조

- 완전 그래프(complete graph)
  - 각 정점에서 다른 모든 정점을 연결하여 최대로 많은 간선 수를 가진 그래프
  - 정점이 n개인 무방향 그래프에서 최대의 간선 수 : n(n-1)/2개
  - 정점이 n개인 방향 그래프의 최대 간선 수 : n(n-1)개
  - 완전 그래프의 예
    - G5는 정점의 개수가 4개인 무방향 그래프이므로 완전 그래프가 되려면 4(4-1)/2=6개의 간선 연결
    - G6은 정점의 개수가 4개인 방향 그래프이므로 완전 그래프가 되려면 4(4-1)=12개의 간선 연결

그림 8-5 완전 그래프의 예

1. 그래프의 구조

- 부분 그래프(subgraph)
  - 원래의 그래프에서 정점이나 간선을 일부만 제외하여 만든 그래프
  - 그래프 G와 부분 그래프 G'의 관계

$$V(G') \subseteq V(G),\ E(G') \subseteq E(G)$$

  - 그래프 G1에 대한 부분 그래프의 예

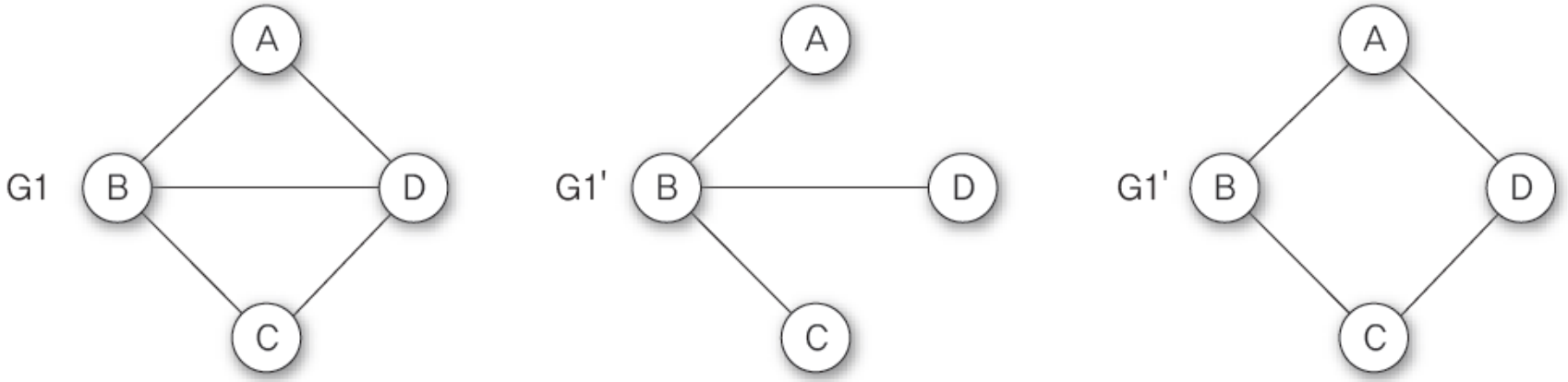


그림 8-6 그래프 G1과 부분 그래프 G1'의 예

1. 그래프의 구조

- 가중 그래프(weight graph) , 네트워크(network)
  - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프

그림 8-7 가중치 그래프의 예

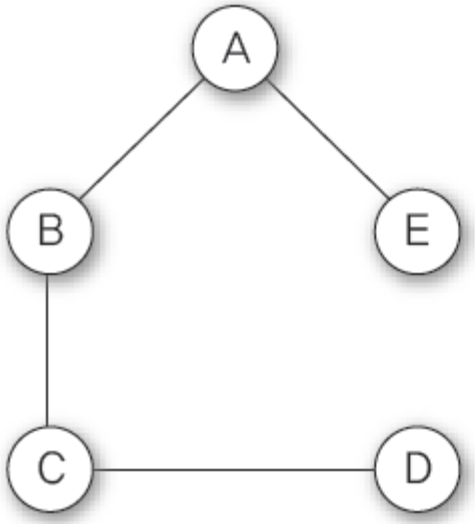
1. 그래프의 구조

❖ 그래프 관련 용어

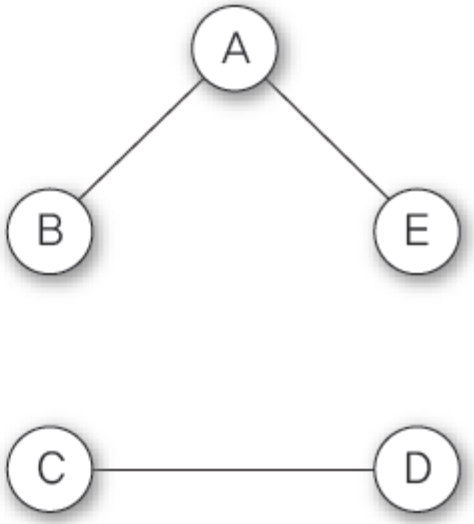
- 그래프에서 두 정점 V_i과 V_j를 연결하는 간선 (V_i, V_j)가 있을 때, 두 정점 V_i와 V_j를 인접(adjacent)되어 있다고 하고, 간선 (V_i, V_j)는 정점 V_i와 V_j에 부속(incident)되어있다고 함
  - 그래프G1에서 정점 A와 인접한 정점은 B와 D이고, 정점 A에 부속되어 있는 간선은 (A,B)와 (A,D)

그림 8-8 그래프 용어 설명을 위한 예

1. 그래프의 구조



(a) 연결 그래프



(b) 단절 그래프

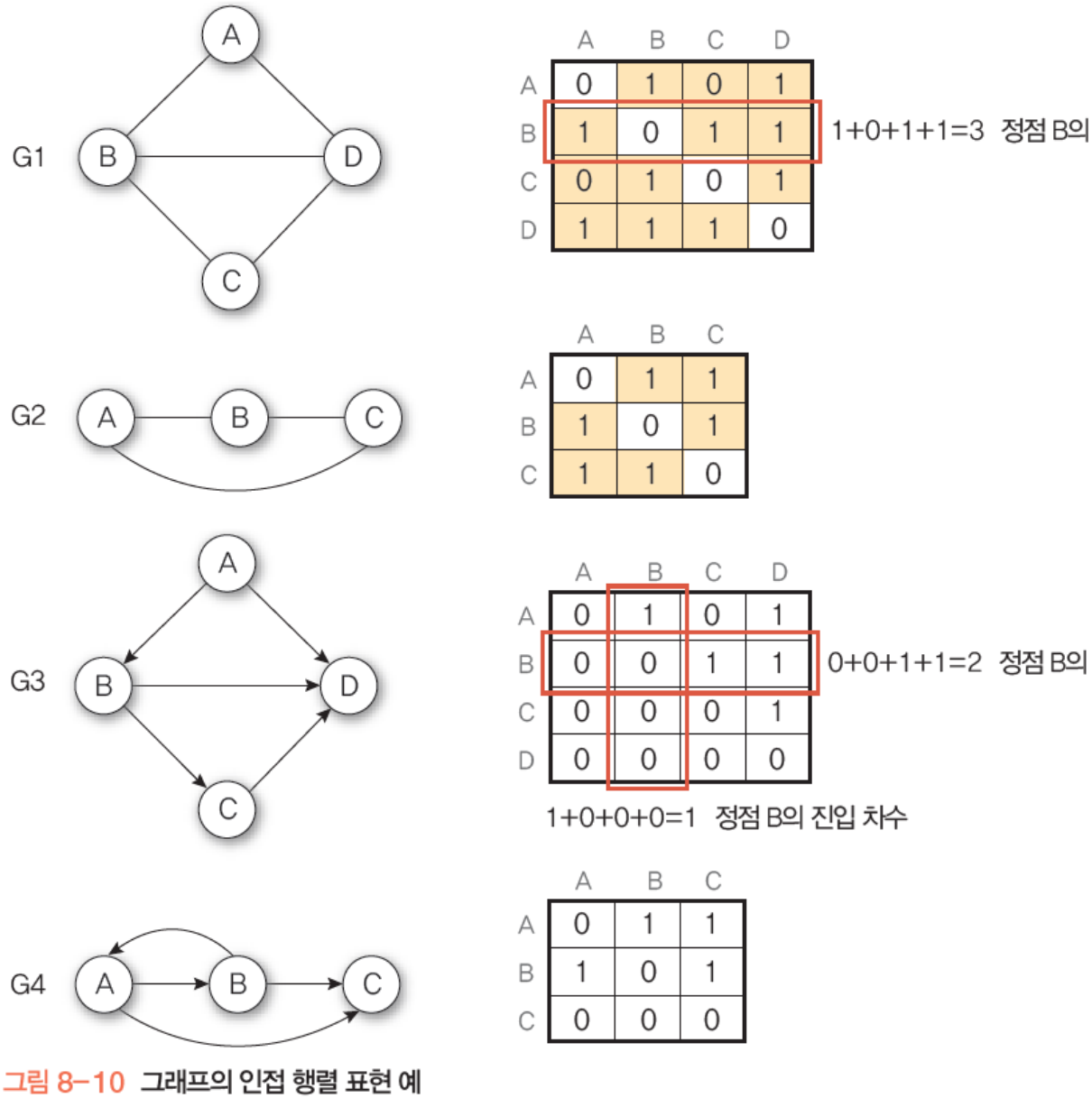
그림 8–9 연결 그래프와 단절 그래프의 예

2. 그래프의 구현

❖ 순차 자료구조를 이용한 그래프의 구현 : 인접 행렬

- 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

2. 그래프의 구현



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

1+0+1+1=3 정점 B의 차수

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

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

0+0+1+1=2 정점 B의 진출 차수

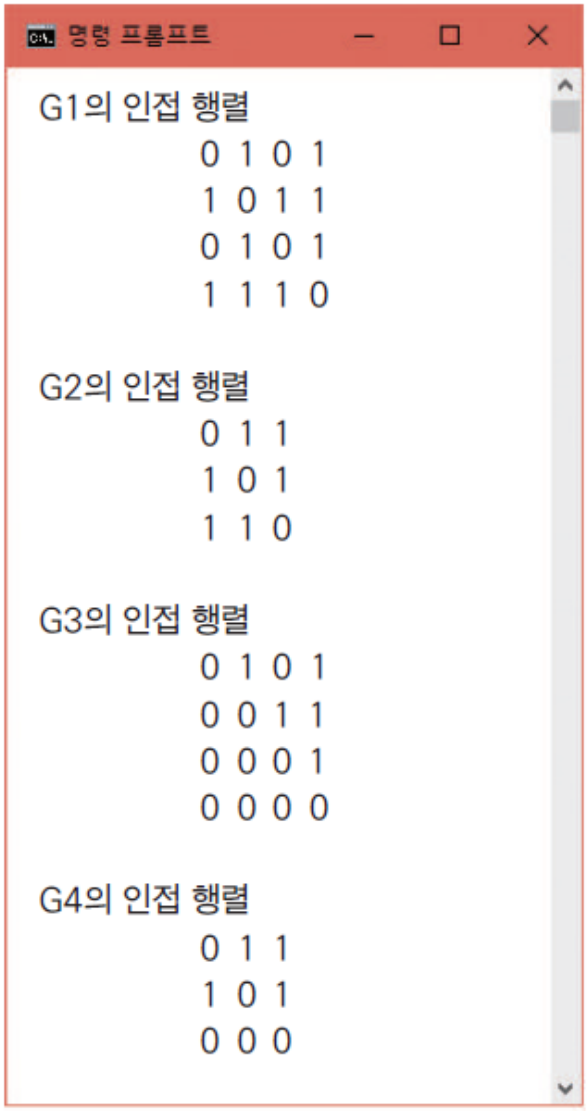
1+0+0+0=1 정점 B의 진입 차수

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

그림 8-10 그래프의 인접 행렬 표현 예

2. 그래프의 구현

- 그래프를 인접행렬로 표현하기 프로그램 : 교재 418p
- 실행 결과



2. 그래프의 구현

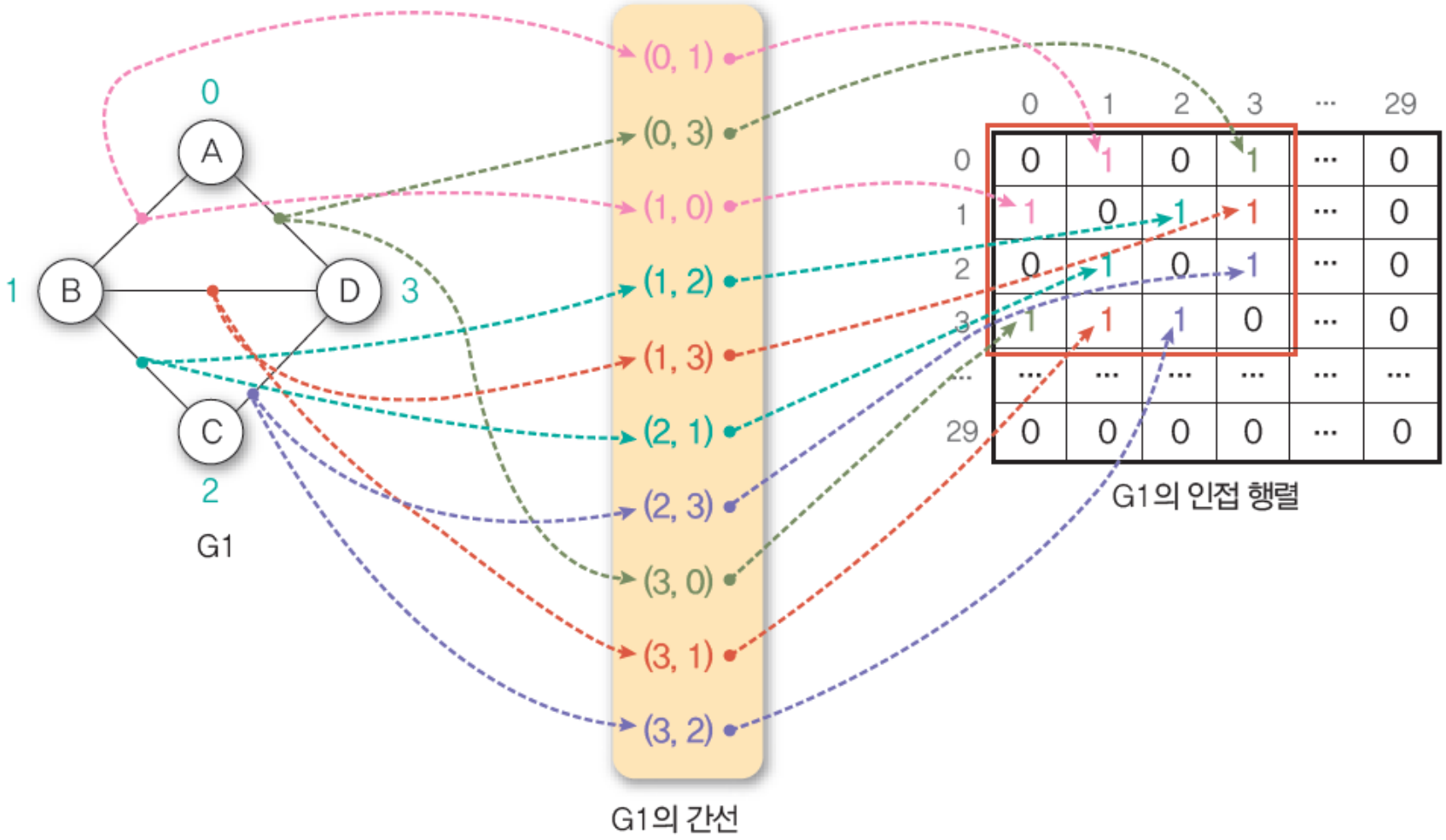
- 052~057행은 그래프 네 개인 G1, G2, G3, G4에 대해 2차원 배열의 메모리를 할당하고 0으로 초기화한 공백 그래프를 생성
- 059~073행은 그래프 G1에 정점 네 개와 간선 열 개를 삽입하여 구현한 인접 행렬을 출력
  - 060~ 061행 : 그래프 G1에 정점을 네 개 삽입

	0	1	2	3	···	29
0	0	0	0	0	···	0
1	0	0	0	0	···	0
2	0	0	0	0	···	0
3	0	0	0	0	···	0
···	···	···	···	···	···	···
29	0	0	0	0	···	0

G1

2. 그래프의 구현

- 062~ 071행 : 그래프 G1에 간선 열 개((0,1), (0,3), (1,0), (1,2), (1,3), (2,1), (2,3), (3,0), (3,1),(3,2)) 삽입



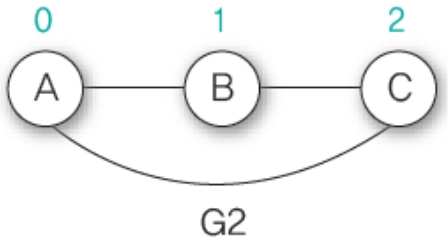
2. 그래프의 구현

- 075~085행은 그래프 G2에 정점 세 개와 간선 여섯 개를 삽입하여 구현한 인접 행렬을 출력
  - 076~ 077행 : 그래프 G2에 정점을 세 개 삽입

	0	1	2	…	29
0	0	0	0	…	0
1	0	0	0	…	0
2	0	0	0	…	0
…	…	…	…	…	0
29	0	0	0	0	…

G2

  - 078~083행 : 그래프 G2에 간선을 여섯 개((0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)) 삽입



G2

	0	1	2	…	29
0	0	1	1	…	0
1	1	0	1	…	0
2	1	1	0	…	0
…	…	…	…	…	0
29	0	0	0	0	…

G2

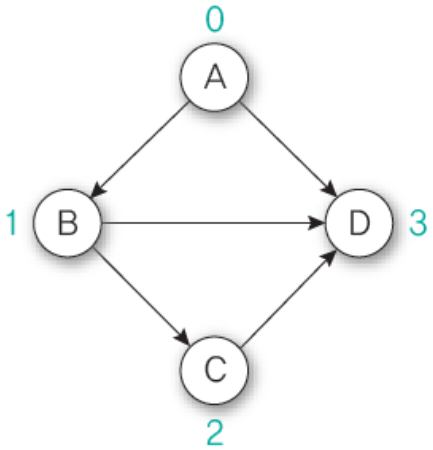
2. 그래프의 구현

- 087~096행은 그래프 G3에 정점 네 개와 간선 다섯 개를 삽입하여 구현한 인접 행렬을 출력
  - 088~089행 : 그래프 G3에 정점을 네 개 삽입

	0	1	2	3	···	29
0	0	0	0	0	···	0
1	0	0	0	0	···	0
2	0	0	0	0	···	0
3	0	0	0	0	···	0
···	···	···	···	···	···	···
29	0	0	0	0	···	0

G3

  - 090~094행 : 그래프 G3에 간선을 다섯 개(<0, 1>, <0, 3>, <1, 2>, <1, 3>, <2, 3>) 삽입



G3

	0	1	2	3	···	29
0	0	1	0	1	···	0
1	0	0	1	1	···	0
2	0	0	0	1	···	0
3	0	0	0	0	···	0
···	···	···	···	···	···	···
29	0	0	0	0	···	0

G3

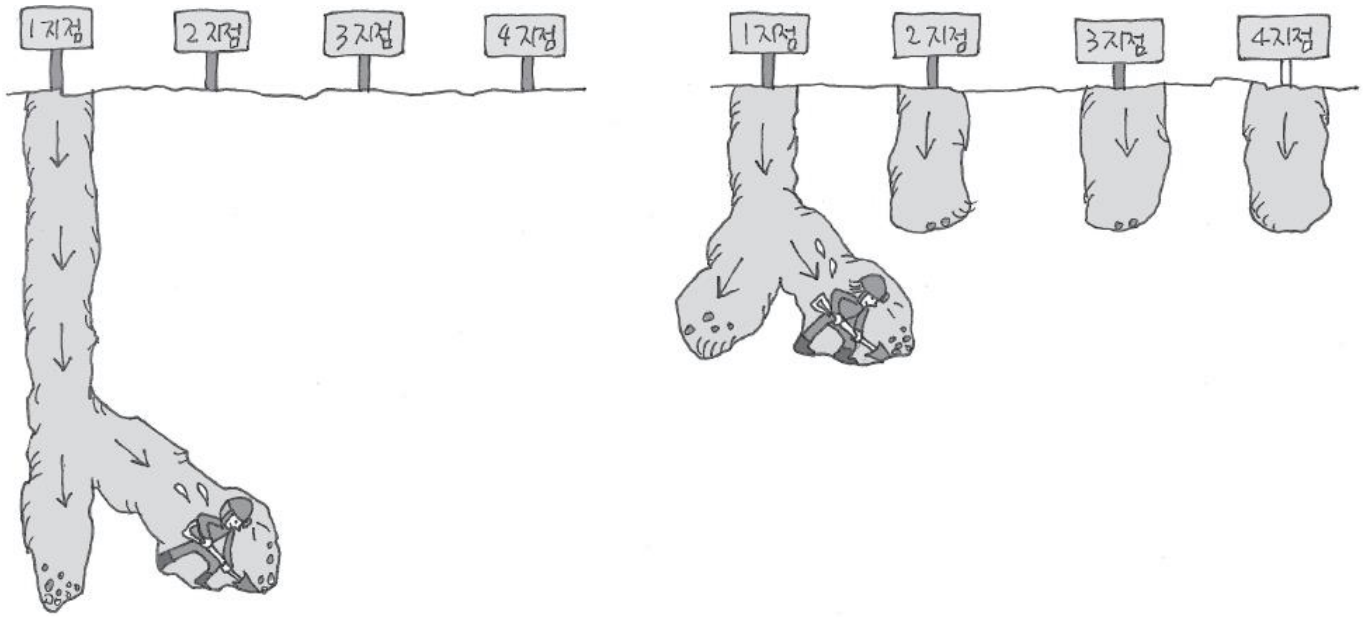
3. 그래프의 순회

❖ 그래프 순회graph traversal, 그래프 탐색graph 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)

3. 그래프의 순회

- 그래프 순회의 예) 우물 파기
  - 한 지점을 골라서 팔 수 있을 때까지 계속해서 깊게 파다가 아무리 땅을 파도 물이 나오지 않으면, 밖으로 나와 다른 지점을 골라서 다시 깊게 땅을 파는 방법 (☞ 깊이 우선 탐색)
  - 여러 지점을 고르게 파보고 물이 나오지 않으면, 파놓은 구덩이들을 다시 좀더 깊게 파는 방법 (☞ 너비 우선 탐색)



(a) 깊이 우선 탐색 예 (b) 너비 우선 탐색 예

그림 8-12 그래프 탐색 예 : 우물 파기

3. 그래프의 순회

❖ 깊이 우선 탐색

- 순회 방법
  - 시작 정점의 한 방향으로 갈 수 있는 경로가 있는 곳까지 깊이 탐색해 가다가 더 이상 갈 곳이 없으면, 가장 마지막에 만났던 갈림길 간선이 있는 정점으로 되돌아와 다른 방향의 간선으로 탐색을 계속 반복하여 결국 모든 정점을 방문하는 순회방법
  - 가장 마지막에 만났던 갈림길 간선의 정점으로 가장 먼저 되돌아가서 다시 깊이 우선 탐색을 반복해야 하므로 후입선출 구조의 스택 사용

❶ 시작 정점 v를 결정하여 방문한다.

❷ 정점 v에 인접한 정점 중에서

❷–ⓐ 방문하지 않은 정점 w가 있으면 정점 v를 스택에 push하고 w를 방문한다. 그리고 w를 v로 하여 다시 ❷를 반복한다.

❷–ⓑ 방문하지 않은 정점이 없으면 스택을 pop하여 받은 가장 마지막에 방문한 정점을 v로 설정한 뒤 다시 ❷를 수행한다.

❸ 스택이 공백이 될 때까지 ❷를 반복한다.

그림 8–13 깊이 우선 탐색 순서

3. 그래프의 순회

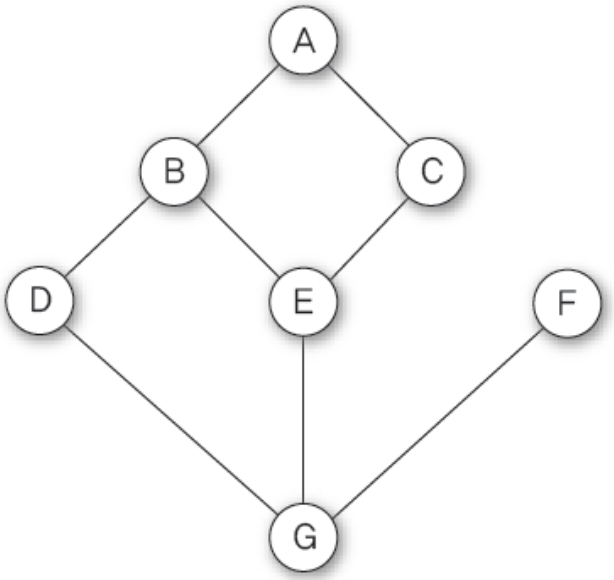
알고리즘 8-1 그래프의 깊이 우선 탐색

```
DFS(v)
    for (i ← 0; i<n; i ← i + 1) do {
        visited[i]←false;
    }
    stack ← createStack();
    visited[v] ← true;
    v 방문;
    while (not isEmpty(stack)) do {
        if (visited[v의 인접 정점 w] = false) then {
            push(stack, v);
            visited[w] ← true;
            w 방문;
            v ← w;
        }
        else v ← pop(stack);
    }
end DFS()
```

3. 그래프의 순회

- 알고리즘에 따라 그래프 G9를 깊이 우선 순회하는 과정

  ① 그래프 G9의 깊이 우선 탐색을 위한 초기 상태 : 배열v isited를 false로 초기화하고 공백 스택 생성



G9

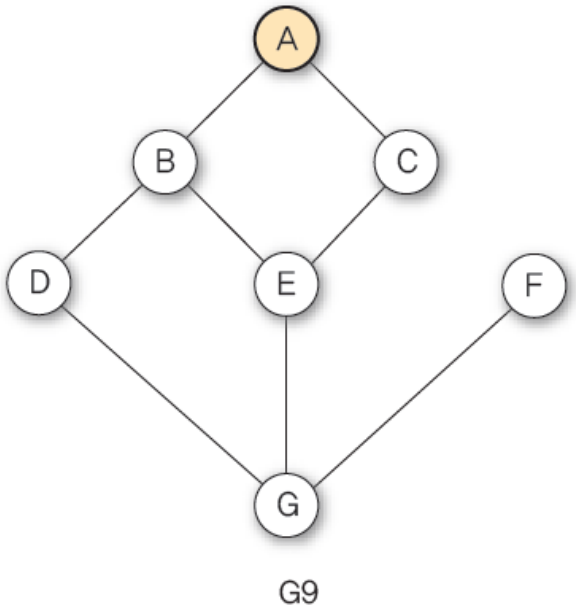
stack

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

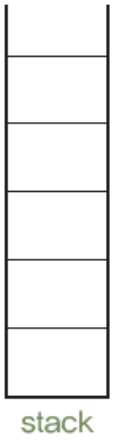
3. 그래프의 순회

② 정점 A를 시작으로 깊이 우선 탐색을 시작

```
visited[A] ← true;
A 방문;
```



G9



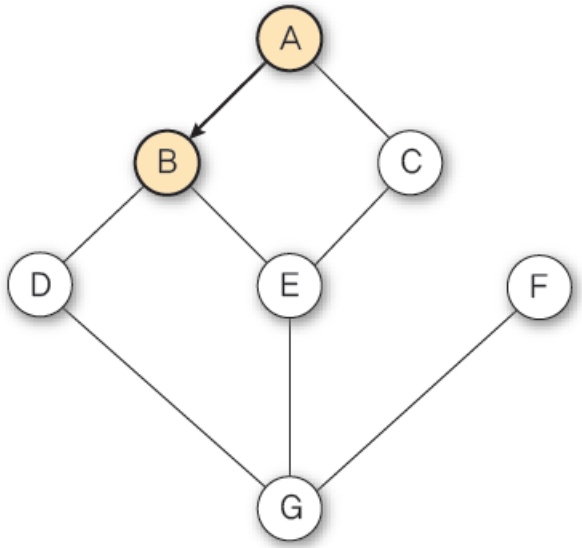
stack

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

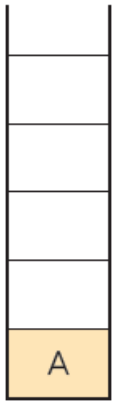
3. 그래프의 순회

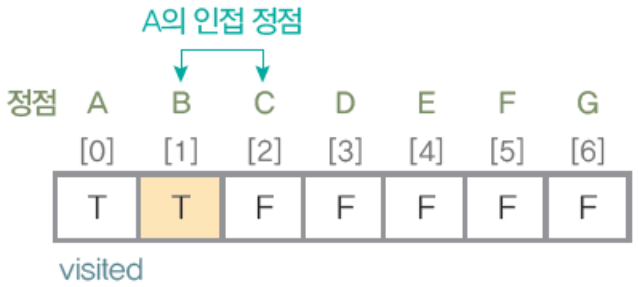
③ 정점 A에 방문하지 않은 정점 B, C가 있으므로 A를 스택에 push하고, 인접 정점 B와 C 중에서 오름차순에 따라 B를 선택하여 탐색을 계속함

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



G9

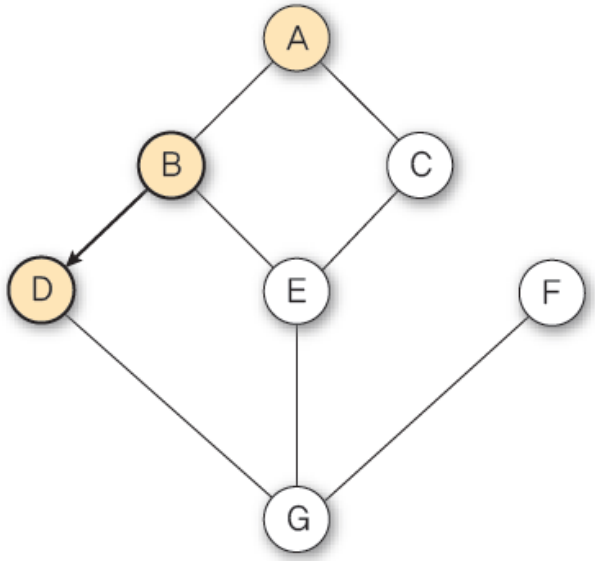


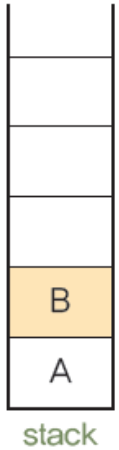


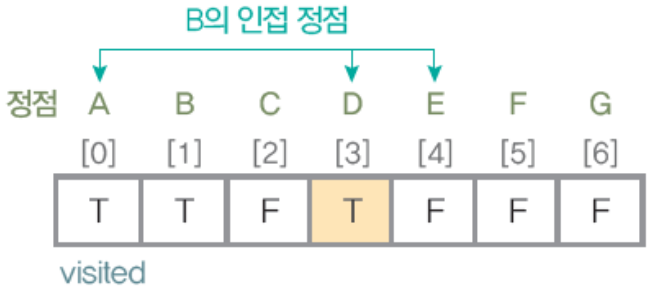
3. 그래프의 순회

④ 정점 B에 방문하지 않은 정점 D, E가 있으므로 B를 스택에 push하고, 방문하지 않은 인접 정점 D와 E 중에서 오름차순에 따라 D를 선택하여 탐색을 계속

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



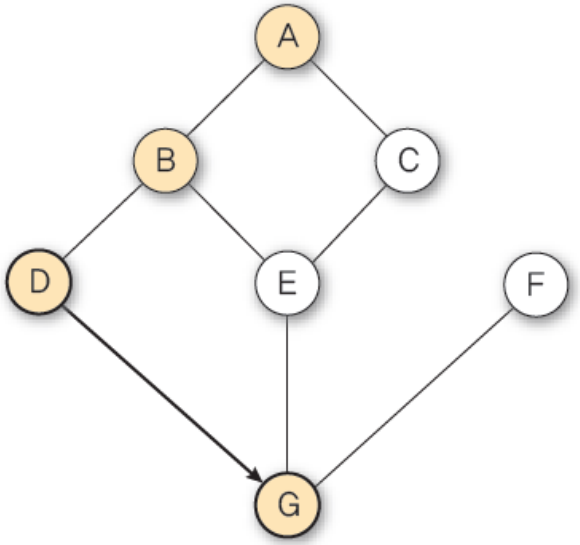


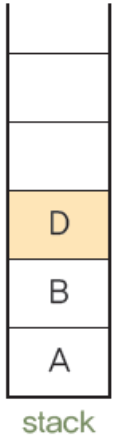


3. 그래프의 순회

⑤ 정점 D에 방문하지 않은 정점 G가 있으므로 D를 스택에 push하고, 인접 정점 G를 선택하여 탐색을 계속

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

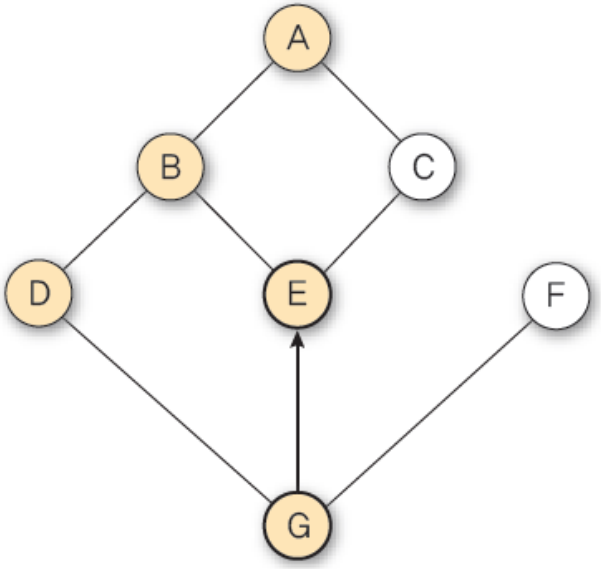


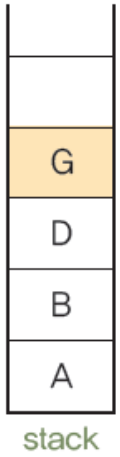


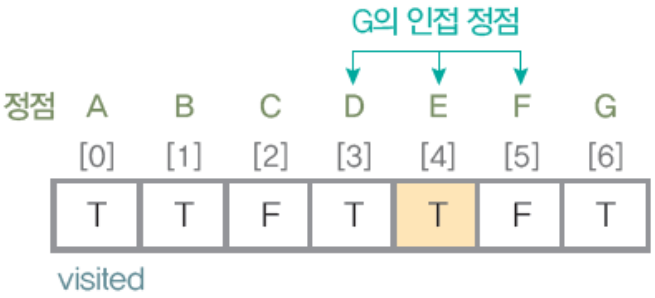
3. 그래프의 순회

⑥ 정점 G에 방문하지 않은 정점 E, F가 있으므로 G를 스택에 push하고, 방문하지 않은 인접 정점 E와 F중에서 오름차순에 따라 E를 선택하여 탐색을 계속

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



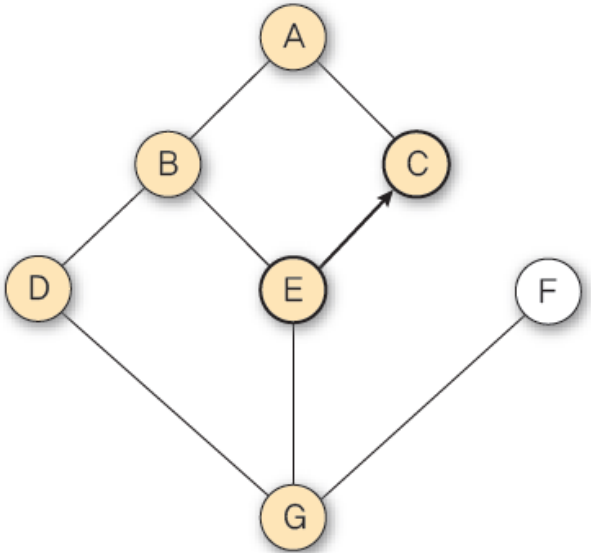


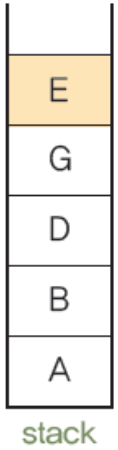


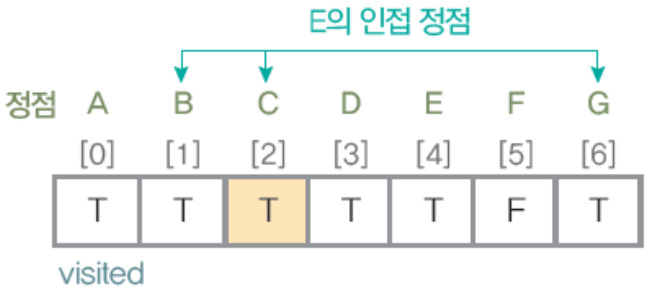
3. 그래프의 순회

⑦ 정점 E에 방문하지 않은 정점 C가 있으므로 E를 스택에 push하고, 방문하지 않은 인접 정점 C를 선택하여 탐색을 계속

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



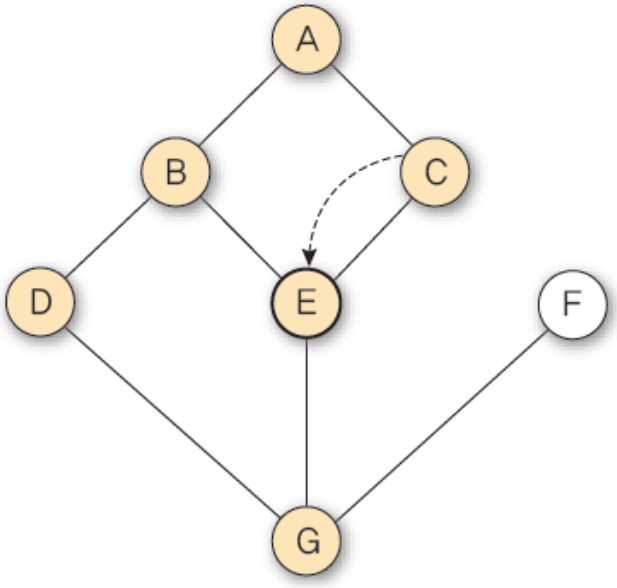


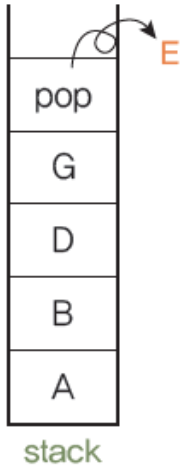


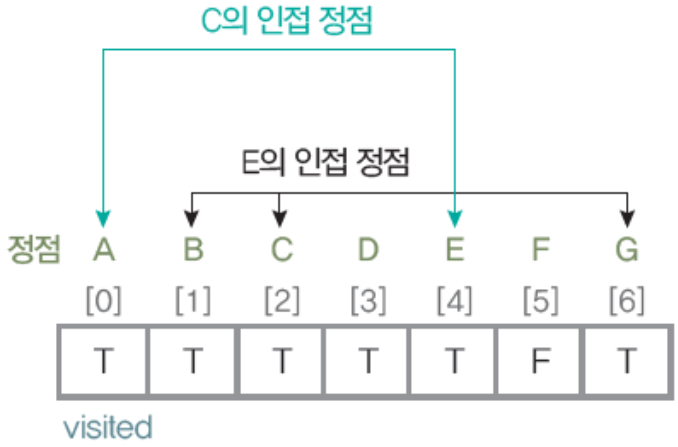
3. 그래프의 순회

⑧ 정점 C에서 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 E에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 E는 방문하지 않은 인접 정점이 없음

```
pop(stack);
```



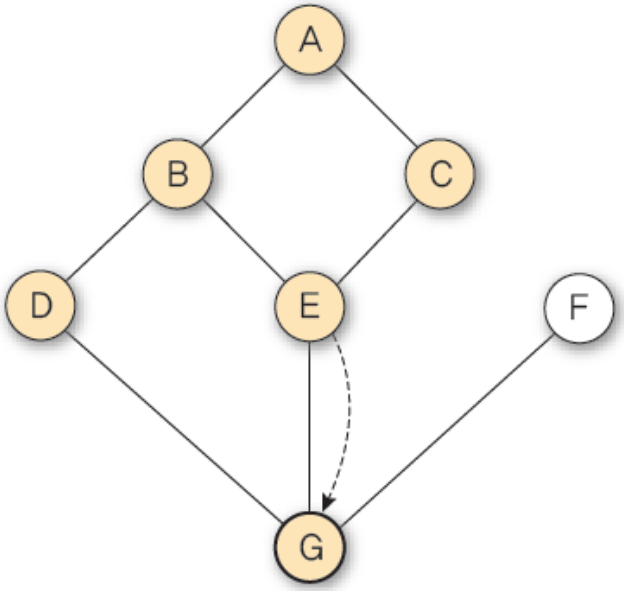


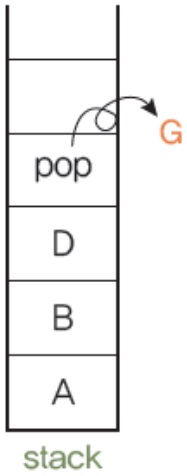


3. 그래프의 순회

⑨ 현재 정점 E에서 방문할 수 있는 인접 정점이 없으므로, 다시 스택을 pop하여 받은 정점 G에 대해서 방문하지 않은 인접 정점이 있는지 확인

```
pop(stack);
```



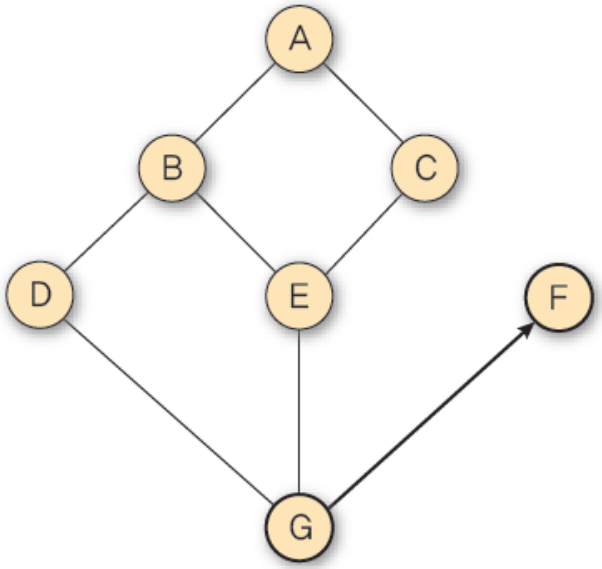


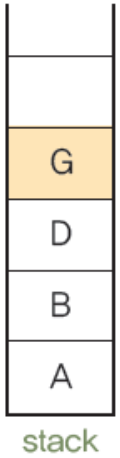
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	T	T	T	T	F	T

3. 그래프의 순회

⑩ 현재 정점 G에서 방문하지 않은 정점 F가 있으므로 G를 스택에 push하고, 인접 정점 F를 선택하여 탐색을 계속

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

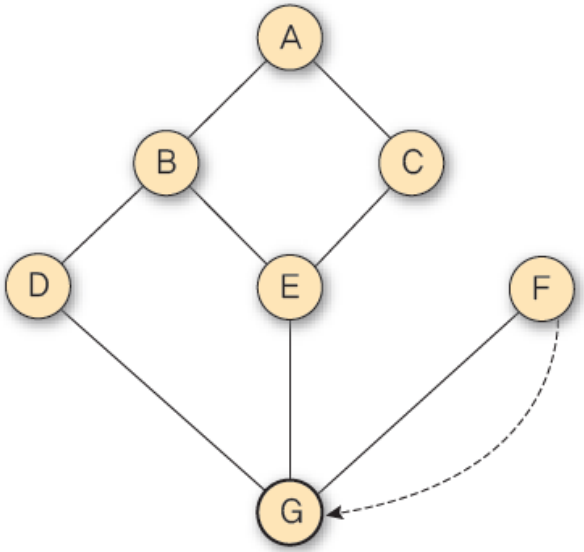


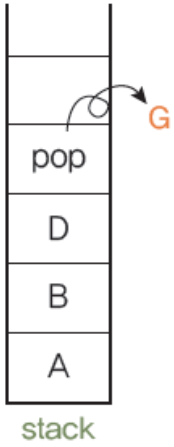


3. 그래프의 순회

⑪ 현재 정점 F에서 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 G에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 G는 방문하지 않은 인접 정점이 없음

```
pop(stack);
```

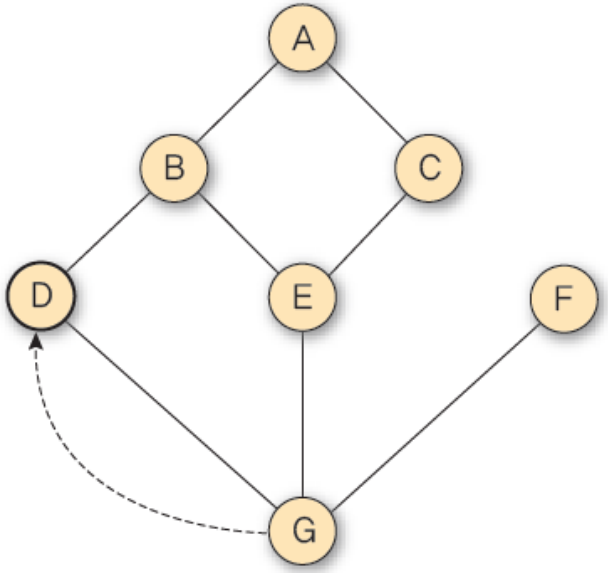


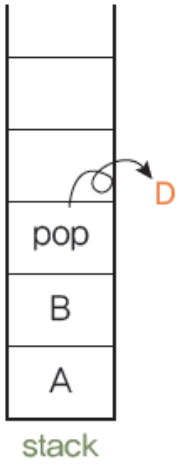


3. 그래프의 순회

⑫ 현재 정점 G에서 방문하지 않은 인접 정점이 없으므로 다시 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 D에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 D는 방문하지 않은 인접 정점이 없음

```
pop(stack);
```

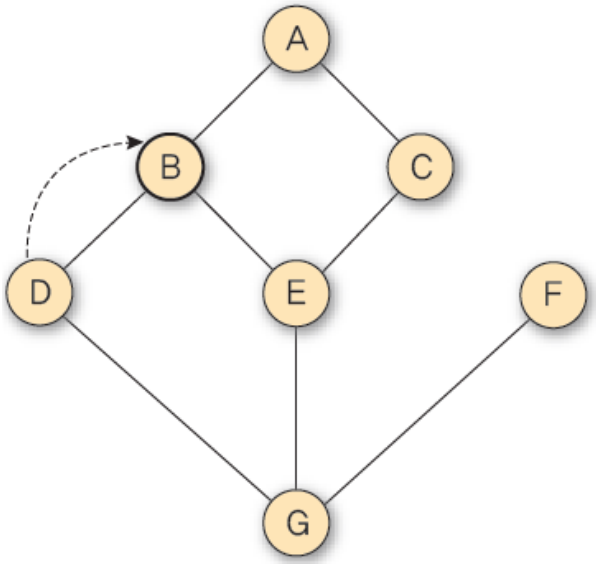


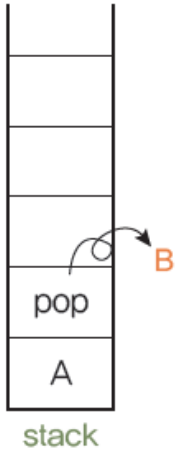


3. 그래프의 순회

⑬ 현재 정점 D에서 방문하지 않은 인접 정점이 없으므로 다시 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 B에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 B는 방문하지 않은 인접 정점이 없음

```
pop(stack);
```

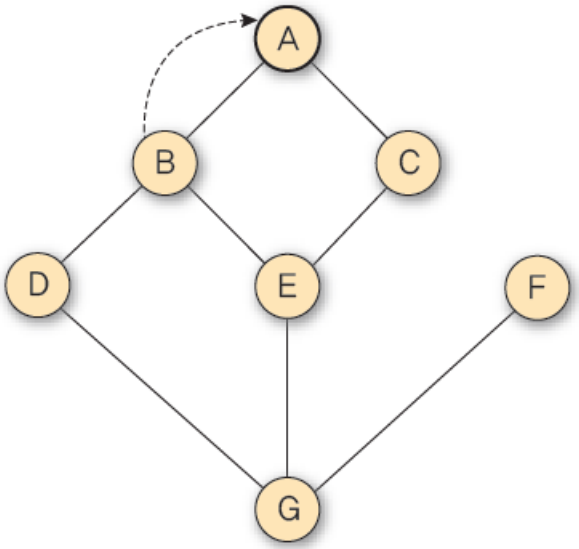


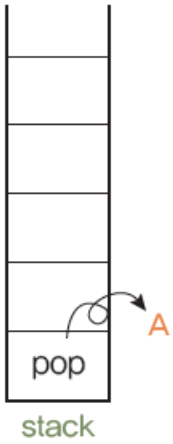


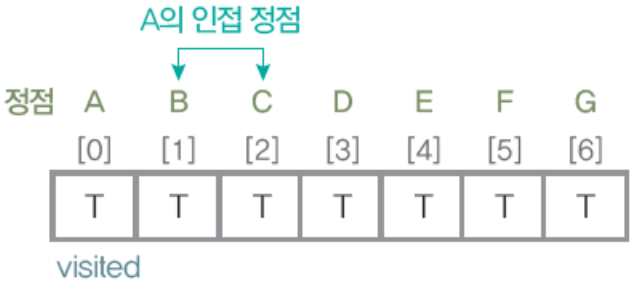
3. 그래프의 순회

⑭ 현재 정점 B에서 방문하지 않은 인접 정점이 없으므로 다시 마지막 정점으로 돌아가기 위해 스택을 pop. 그리고 받은 정점 A에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 A는 방문 하지 않은 인접 정점이 없음

```
pop(stack);
```







⑮현재 정점 A에서 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop. 스택이 공백이므로 깊이 우선 탐색을 종료

3. 그래프의 순회

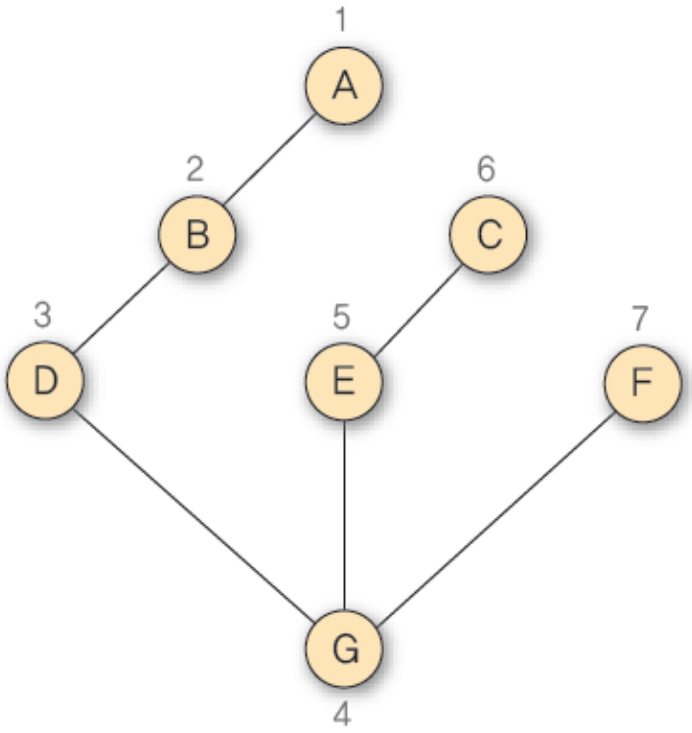
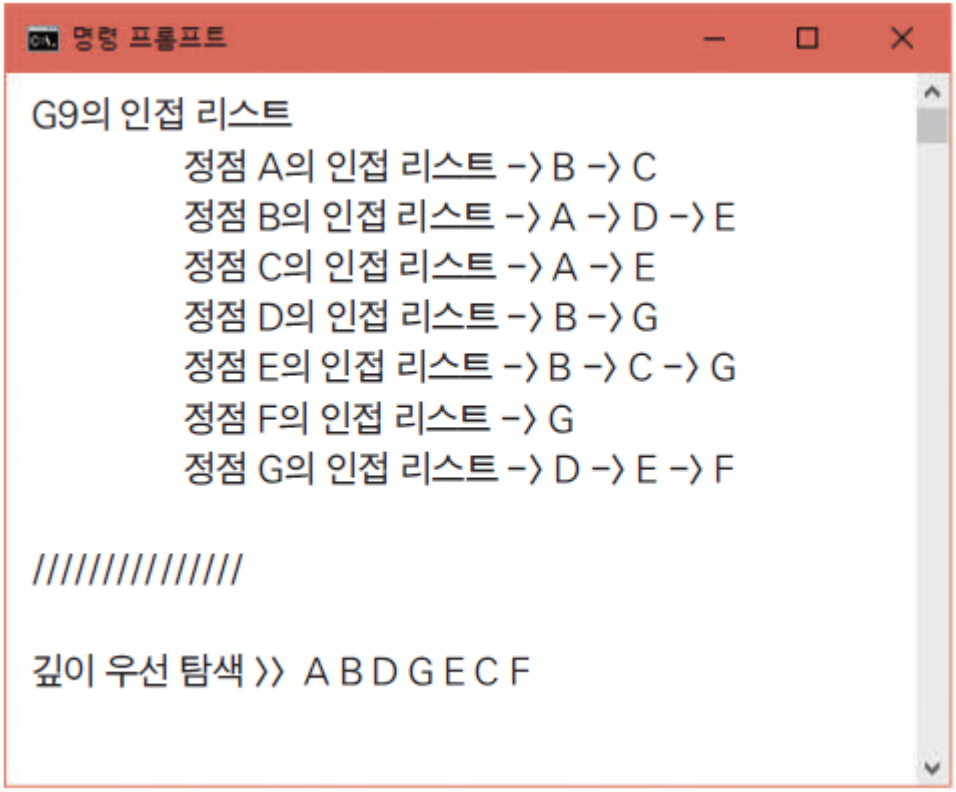


그림 8-14 그래프 G9의 깊이 우선 탐색 경로 : A-B-D-G-E-C-F

3. 그래프의 순회

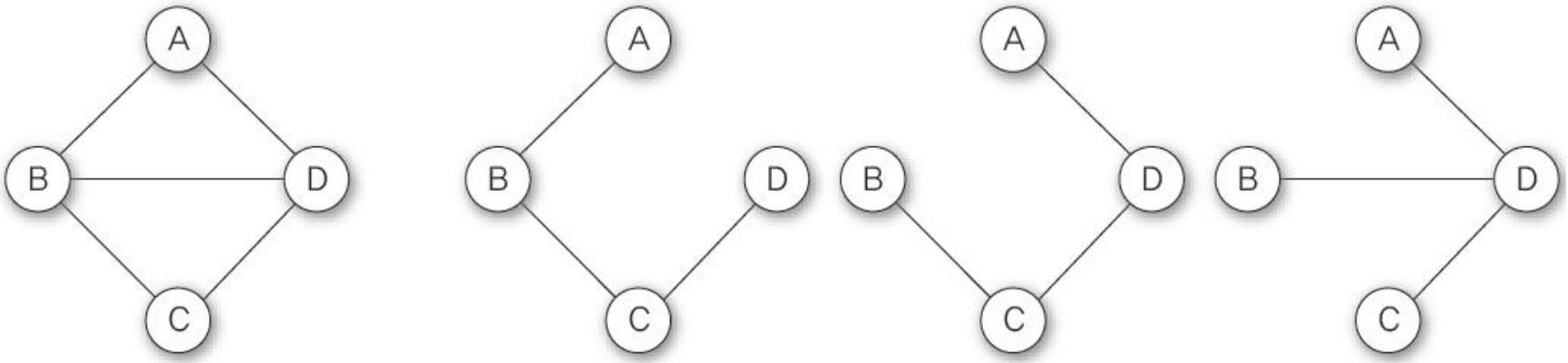
- 그래프 깊이 우선 탐색하기 프로그램 : 교재 444p
- 실행 결과



4. 신장 트리와 최소 비용 신장 트리

❖ 신장 트리(spanning tree)

- n개의 정점으로 이루어진 무방향 그래프 G에서 n개의 모든 정점과 n-1개의 간선으로 만들어진 트리

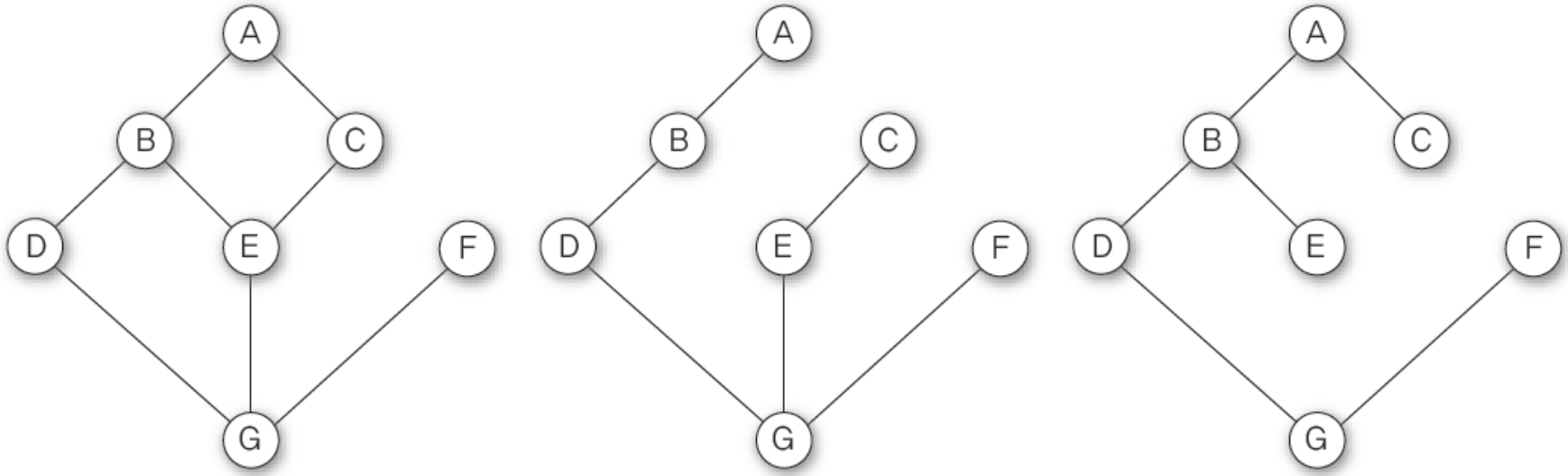


(a) G1 (b) G1의 신장 트리

그림 8-17 그래프 G1과 신장 트리의 예

4. 신장 트리와 최소 비용 신장 트리

- 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



(a) G9 (b) G9의 깊이 우선 신장 트리 (c) G9의 너비 우선 신장 트리

그림 8–18 그래프 G9와 신장 트리의 예

4. 신장 트리와 최소 비용 신장 트리

❖ 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
  - 크루스칼Kruskal의 알고리즘
  - 프림Prime의알고리즘

4. 신장 트리와 최소 비용 신장 트리

❖ 크루스칼 알고리즘 I

- 가중치가 높은 간선을 제거하면서 최소 비용 신장 트리를 만드는 방법
- 순서

❶ 그래프 G의 모든 간선을 가중치에 따라 내림차순으로 정렬한다.

❷ 그래프 G에서 가중치가 가장 높은 간선을 제거한다. 단, 이때 정점을 그래프에서 분리시키는 간선은 제거할 수 없으므로 이런 경우에는 그 다음으로 가중치가 높은 간선을 제거한다.

❸ 그래프 G에 간선이 n–1개만 남을 때까지 ❷를 반복한다.

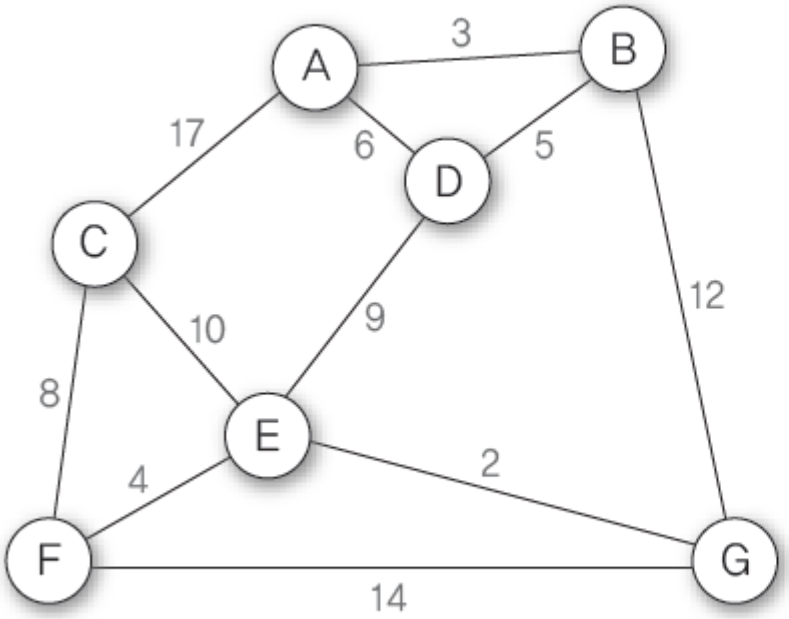
❹ 그래프에 간선이 n–1개만 남으면 최소 비용 신장 트리가 완성된다.

그림 8–19 크루스칼 알고리즘 I 의 순서

4. 신장 트리와 최소 비용 신장 트리

- 크루스칼 알고리즘 I 을 이용하여 G10의 최소 비용 신장 트리 만들기

  0. 초기 상태 : 그래프 G10의 간선을 가중치에 따라서 내림차순 정렬



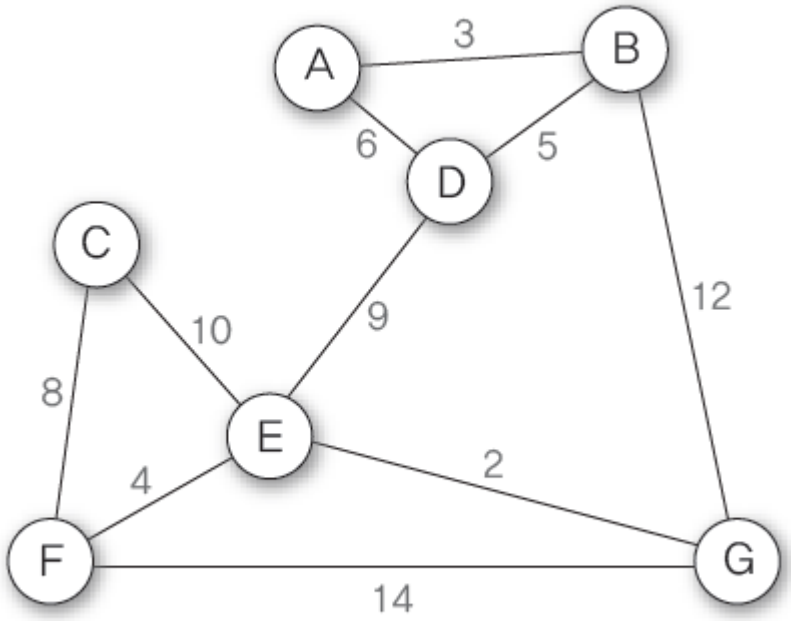
G10

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)

간선 수 : 11개

4. 신장 트리와 최소 비용 신장 트리

① 가중치가 가장 높은 간선 (A, C)를 제거 → 남은 간선 수는 열 개

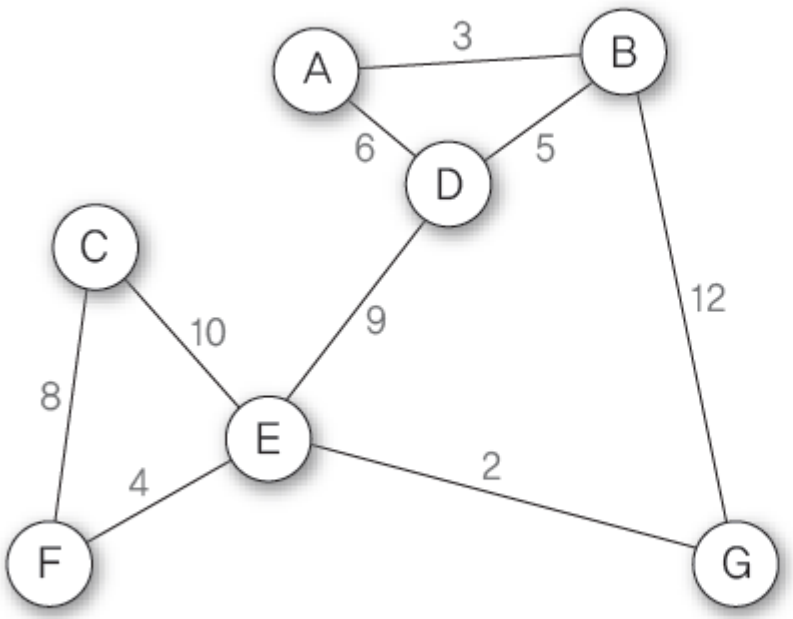


가중치	간선
~~17~~	~~(A, C)~~
14	(F, G)
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

② 남은 간선 중에서 가중치가 가장 높은 간선 (F, G)를 제거

→ 남은 간선 수는 아홉 개

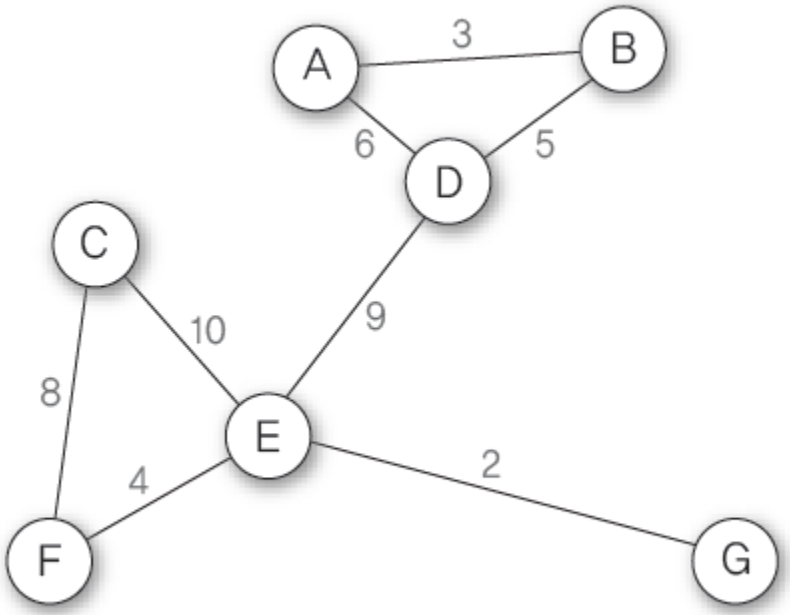


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

③ 남은 간선 중에서 가중치가 가장 높은 간선 (B, G)를 제거

→ 남은 간선 수는 여덟 개

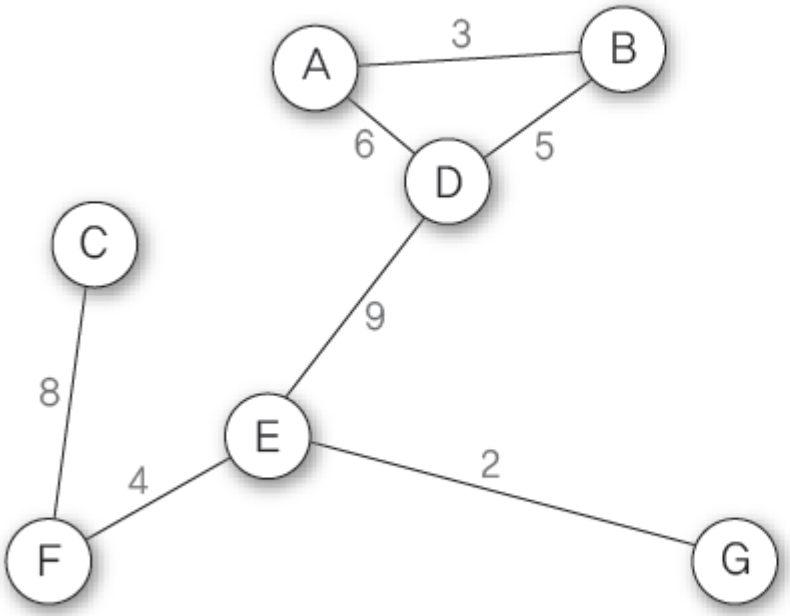


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

④ 남은 간선 중에서 가중치가 가장 높은 간선 (C, E)를 제거

→ 남은 간선 수는 일곱 개

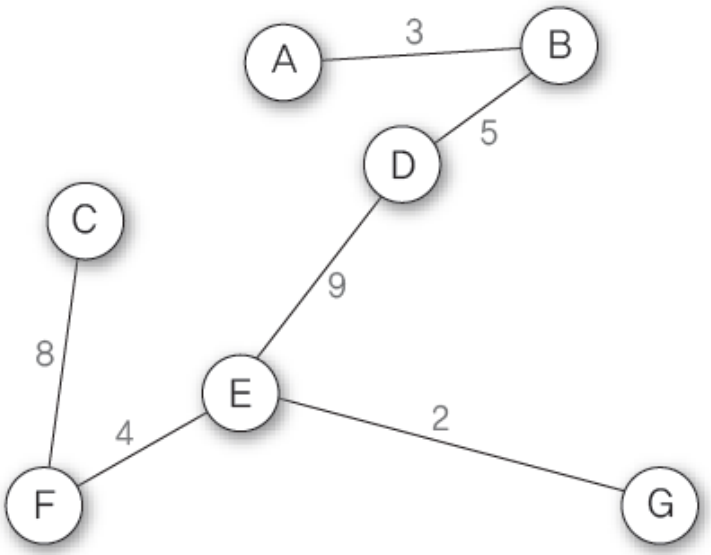


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
~~10~~	~~(C, E)~~
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

⑤ 남은 간선 중에서 가중치가 가장 높은 간선 (D, E)를 제거하면 그래프가 분리되어 단절 그래프가 되므로 그 다음으로 가중치가 높은 간선 (C, F)를 제거. 그런데 간선 (C, F)를 제거하면 정점 C가 그래프에서 분리되므로 제거 불가능. 따라서 그 다음으로 가중치가 높은 간선 (A, D)를 제거

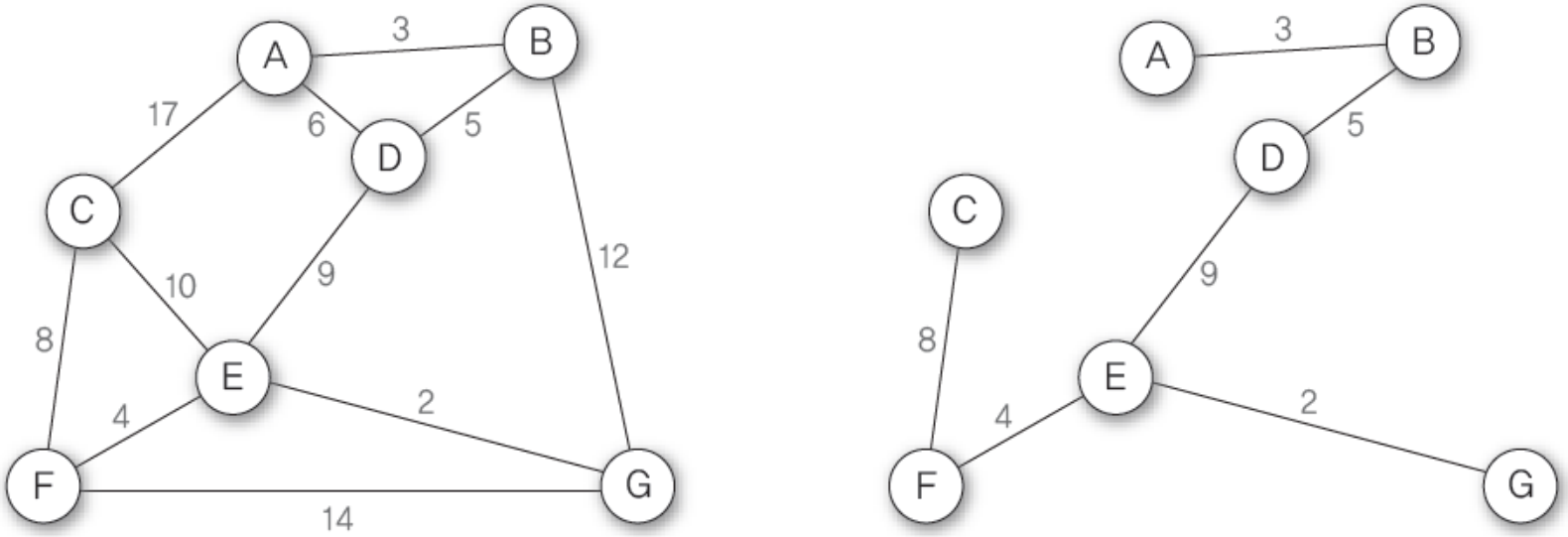
→ 남은 간선 수는 여섯 개



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)

4. 신장 트리와 최소 비용 신장 트리

⑥ 현재 남은 간선 수가 여섯 개이므로 알고리즘 수행을 종료하면 신장 트리 완성



(a) G10

(b) 크루스칼 알고리즘 I을 적용한 G10의 최소 비용 신장 트리

그림 8-20 그래프 G10을 크루스칼 알고리즘 I으로 완성한 최소 비용 신장 트리

Thank You