그래프(Graph)

1.1 그래프의 기본 개요

- Koenigsberg 다리 문제를 해결하기 위해 Euler가 처음 사용
- 선형 자료구조나 트리 자료구조로 표현하기 어려운 多:多의 관계를 가지는 원소들을 표현하기 위한 자료구조
- 자료객체(data object)들 사이의 관계를 다이어그램으로 나타내는 비선형적인 자료구조로서 전산학, 수학, 공학, 언어학, 사회과학 등 여러 분야에서 응용
- 특히 전기 회로의 분석, 최단거리 검색, 컴퓨터 network, 인공지능 등에 광범위하게 이용

1

그래프의 기본 개요

그래프는 1736년 수학자 Euler가 동부 프러시아에 있는 kneiphof라는
2개의 섬과 n개의 다리 문제를 해결하는데 처음 사용.

철학자 칸트가 여생을 보낸 Koenigsberg시에는 Pregal강이
2개의 섬 주위를 흐른다. 이 강은 4개의 지역(A~D)과 접해 있다.
a~g의 7개의 다리로 연결.

오일러는 그래프를 사용하여 모든 다리를 한번씩만 거쳐서 원래의 위치로
되돌아 올 수 없다는 것을 증명.

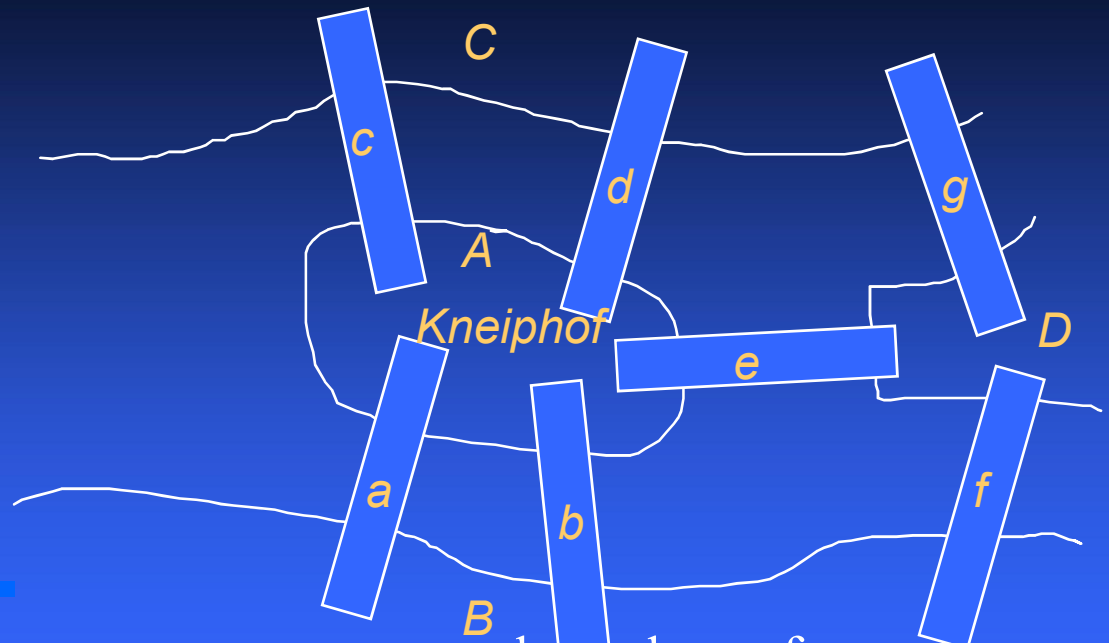
Eulerian Walk.:

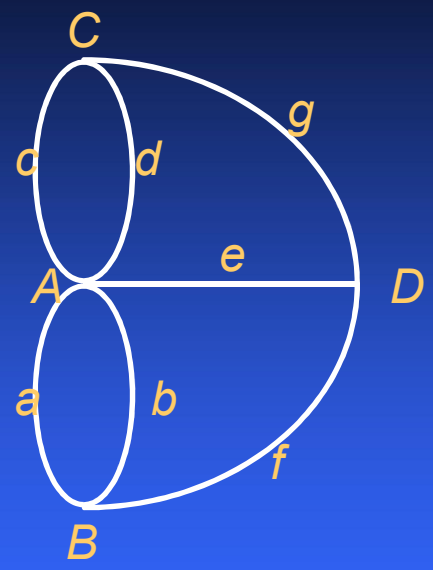
간선의 갯수가 짝수인 경우만 임의의 정점을에서 출발하여
각 간선을 한번씩만 거치고 출발한 정점으로 되돌아 오는길:

1

그래프(Graph)

각 정점의 차수가 짝수일 경우에만 임의의 정점을 출발하여 모든 다리를 한번씩 만 거쳐서 다시 제자리로 되돌아 올 수 있음



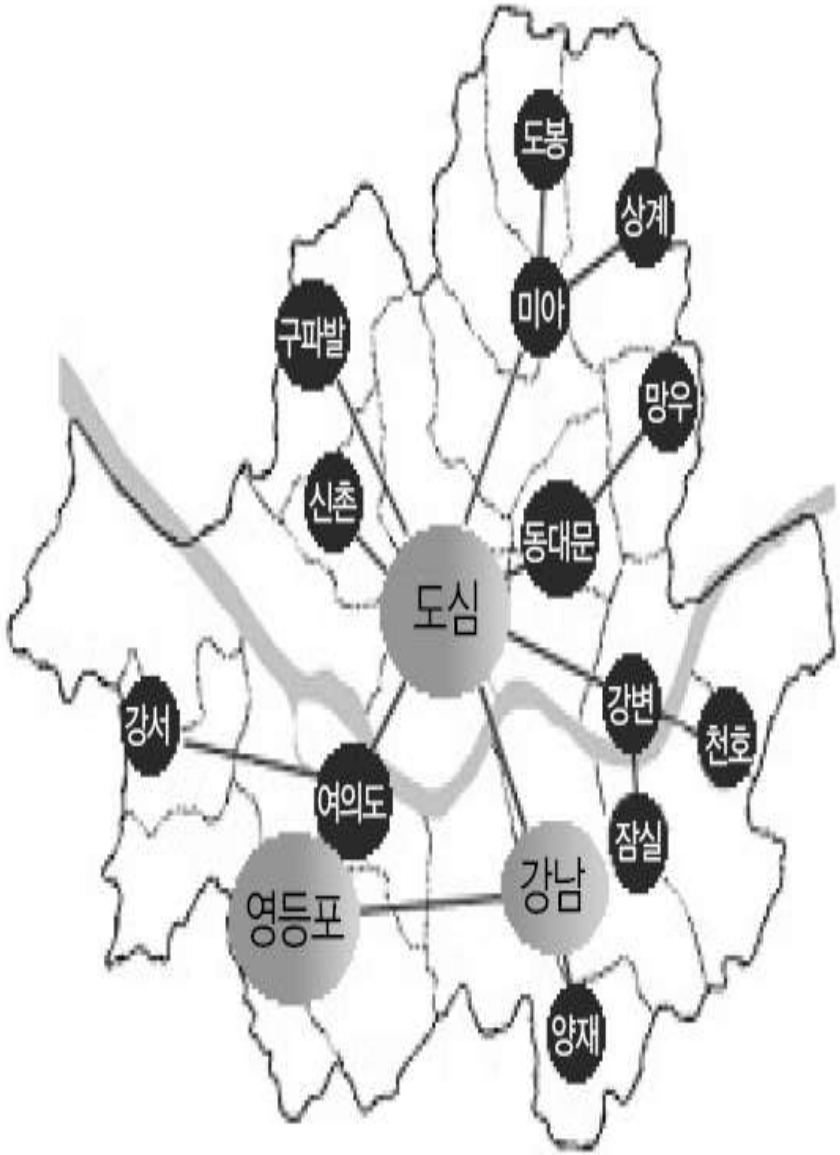


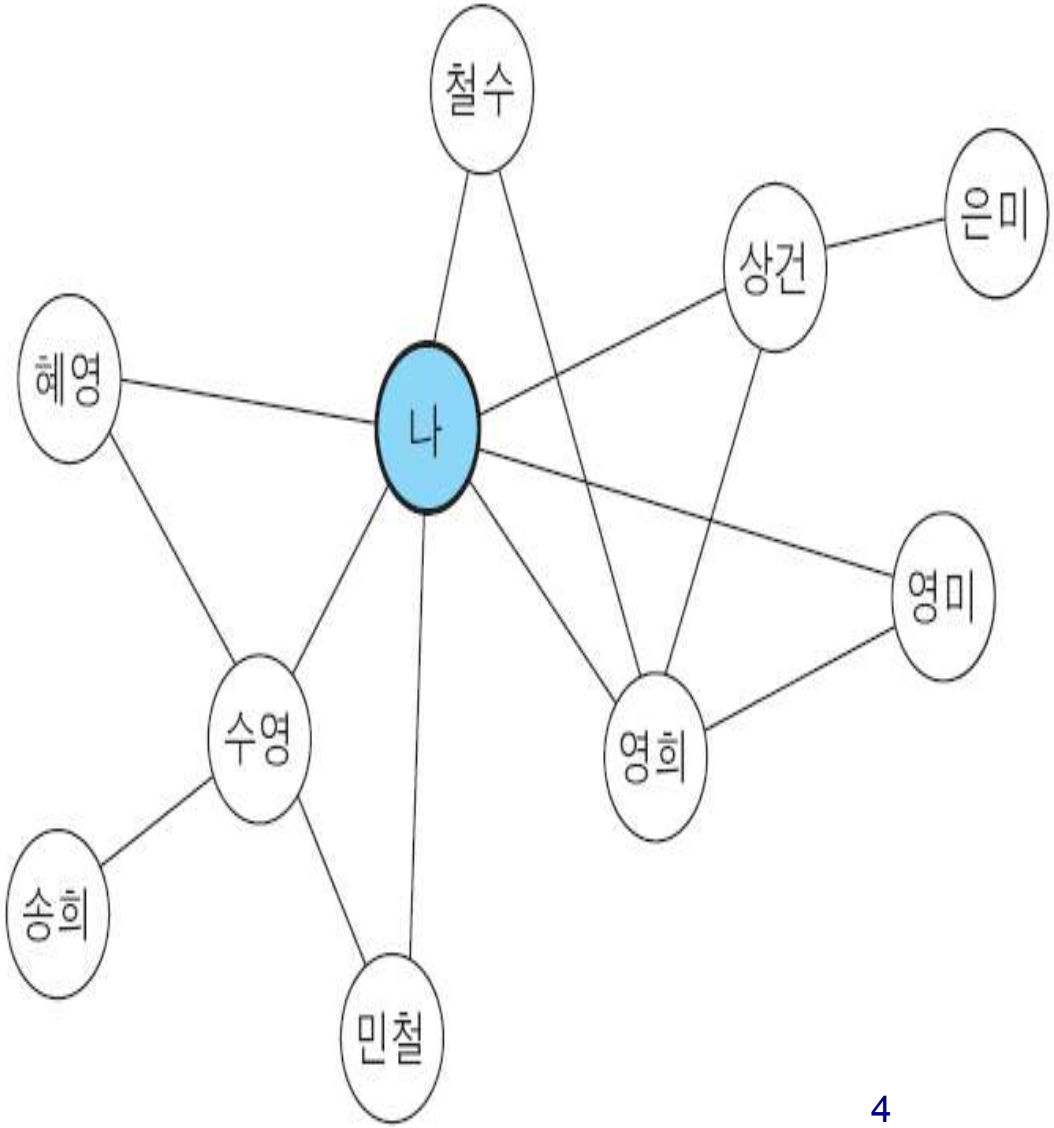
a e g d b f

B → A → D → C → A → B → D

- B에서 출발하여 D에와 있으므로 원래 위치로 돌아오지 못하고
- 모든 다리를 다 거치지도 못함.

그래프의 예

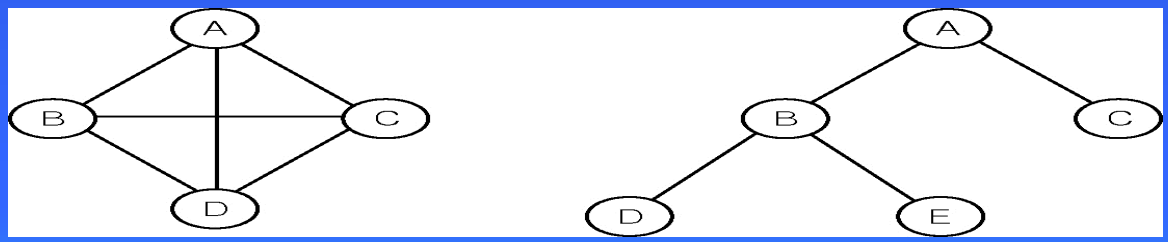

도봉
상계
미아
구파발
망우
신촌
동대문
도심
강서
강변
천호
여의도
잠실
영등포
강남
양재


철수
은미
상건
혜영
나
영미
수영
영희
송희
민철

1.2 그래프의 정의

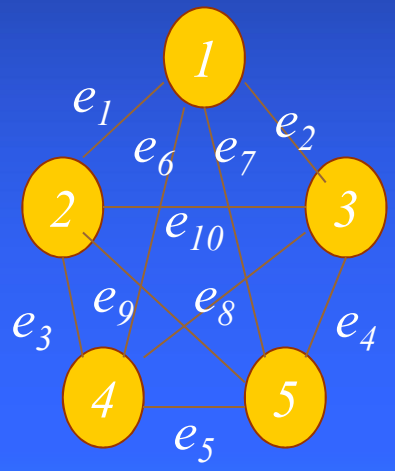
- G = (V, E)
- V(G) : 공집합이 아닌 정점들의 유한집합
- E(G) : 간선들의 집합
- 방향그래프(digraph, directed graph)에서의 간선의 표기
  - $<v_1, v_2>$ V1은 간선의 시작이고 V2는 간선의 끝

- 무방향 그래프(undirected graph)에서의 간선의 표기
  - (V1, V2)와 (V2, V1)은 같은 간선

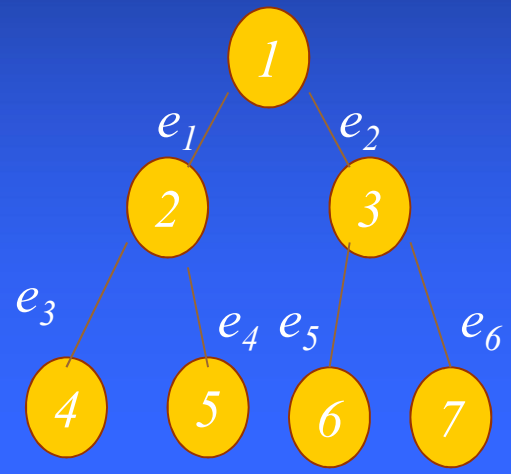


그래프의 정의

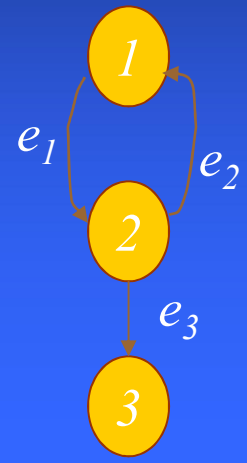
V(G1) = {1, 2, 3, 4, 5}
E(G1) = {(1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5)}
V(G2) = {1, 2, 3, 4, 5, 6, 7}
E(G2) = ((1,2), (1,3), (2,4), (2,5), (3,6), (3,7)}
V(G3) = {1,2,3}
E(G3) = {<1,2>, <2,1>, <2,3>}



G1 : 완전한 그래프



G2 : 트리



G3 : 방향그래프

1.3 그래프의 용어

- **인접(adjacent)**

  간선(v_1, v_2)가 집합 E(G)에 속한다면,

  * 정점 v_1과 정점 v_2는 인접했다고 한다

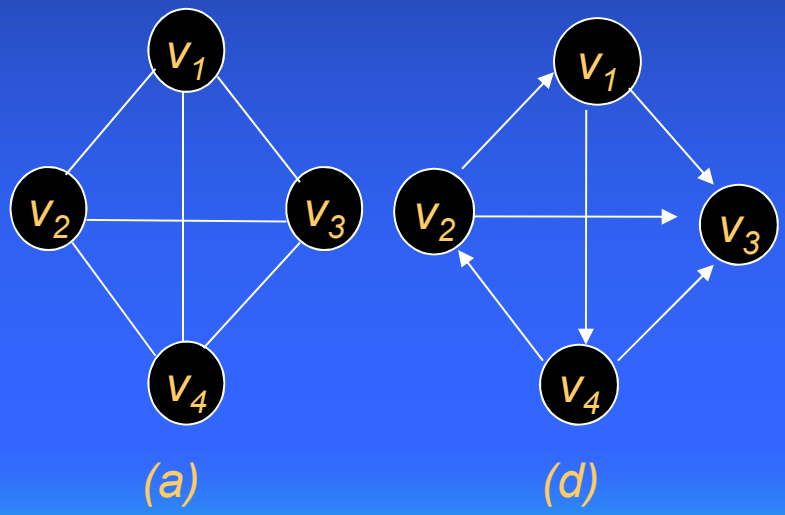
- 부속(incident)

  v_1과 v_2가 인접하였을때 간선 (v_1, v_2)는 정점 v_1과 v_2에 부속.

  즉, 정점 사이에 연결된 간선

그래프의 용어

- 경로: 임의의 정점에서 다른 정점에 이르는 정점들의 순서
- 경로의 길이 : 경로상에 포함된 간선의수
- 사이클 :경로의 시작 정점과 도착정점이 같은 단순경로
- 비사이클 그래프 또는 트리 : 사이클이 없는 그래프
- 대그(dag): 방향 그래프의 비사이클 그래프

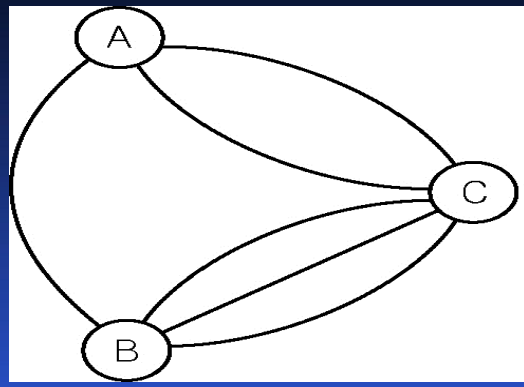


- (a) v_1에서 v_3에 이르는 경로
  : $v_1 v_2 v_4 v_3$ (길이는 3)
- (d) v_4에서 v_3에 이르는 경로
  : $v_4 v_2 v_1 v_3$ (길이는 3)
- $v_1 v_2 v_3 v_1$: (a)의 사이클
- $v_2 v_1 v_4 v_2$: (d)의 사이클

그래프의 용어(계속)

- 다중그래프(multi-graph)

  두 정점 사이에 두개 이상의 간선이 존재하는 그래프



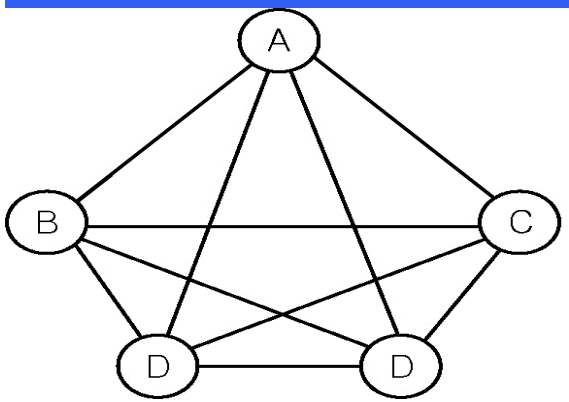
정점 A와 C사이에 간선이 2개

정점 B와 C사이에 간선이 3개

- 완전그래프(complete graph)

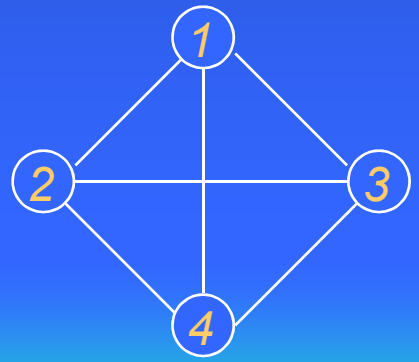
  n개의 정점으로 구성된 무방향 그래프에서 간선 수가 최대인 그래프

  ☞최대 간선수 : (n-1)+(n-2)+…+1 = n(n - 1)/2



정점이 5인 무방향 그래프의

최대 간선수는 5(5−1)/2=10개이다.

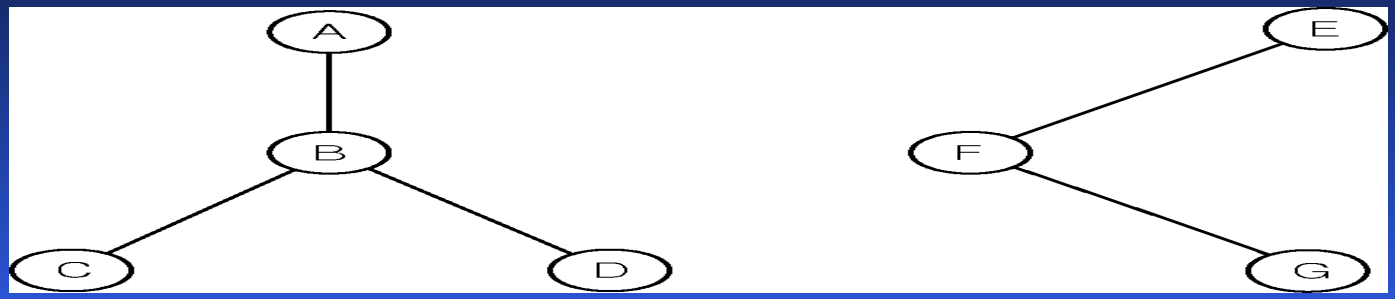


그래프의 용어(계속)

- 연결그래프(connected graph)

  그래프 G가 속하는 모든 정점들이 연결되어 있어서 모든 정점들의 쌍에 대하여 경로가 존재하는 그래프

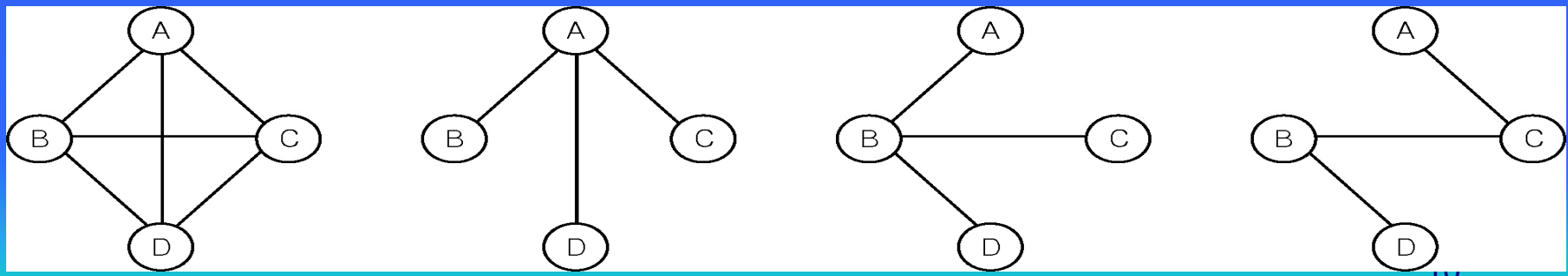
- 단절그래프(disconnected graph)



- 연결요소(**connected component**)

  어떤 정점으로부터 다른 정점으로 갈 수 있는 경로들의 집합

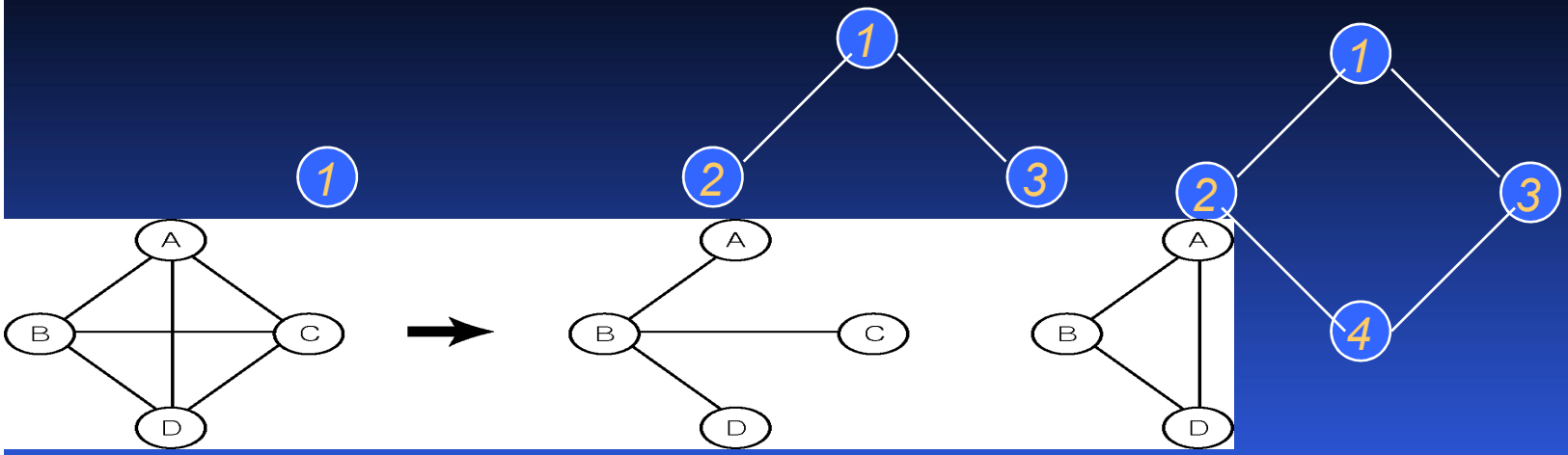
  무방향 그래프 **G**에서 최대 연결 부그래프



그래프의 용어(계속)

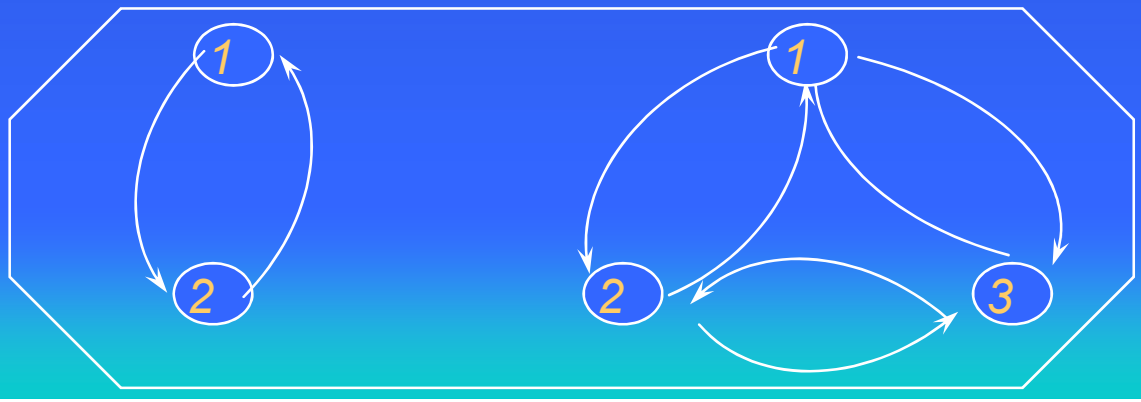
- 부 그래프(subgraph)

  V(G') ⊆ V(G)이고, E(G') ⊆ E(G)인 그래프 G'를 그래프 G의 부그래프



- 강력연결 그래프(strongly connected graph)

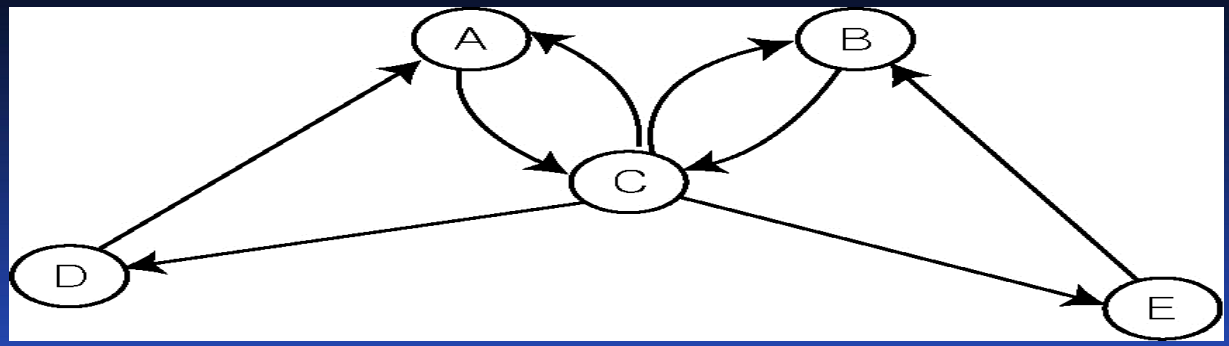
  방향그래프 G에서 V(G)에 속한 상이한 두 정점 V1, V2의 모든 쌍에 대하여 V1에서 V2로, 또한 V3에서 V2로의 방향경로(directed path)가 존재하면, G를 강력 연결그래프



그래프의 용어(계속)

- 차수

  정점에 부속된 간선들의 수



- 진입차수(in-degree)

  방향그래프 G에서 임의 정점 v가 머리(head)가 되는 간선들의 수

  즉, 화살표가 들어오는 간선의 수

- 진출차수(out-degree)

  방향그래프 G에서 임의 정점 v가 꼬리(tail)가 되는 간선들의 수

  즉, 화살표가 밖으로 나가는 간선의 수

- 연결(connected)

  무방향 그래프 G에서 정점 v_1에서 부터 v_2에 이르는 경로가 존재하면

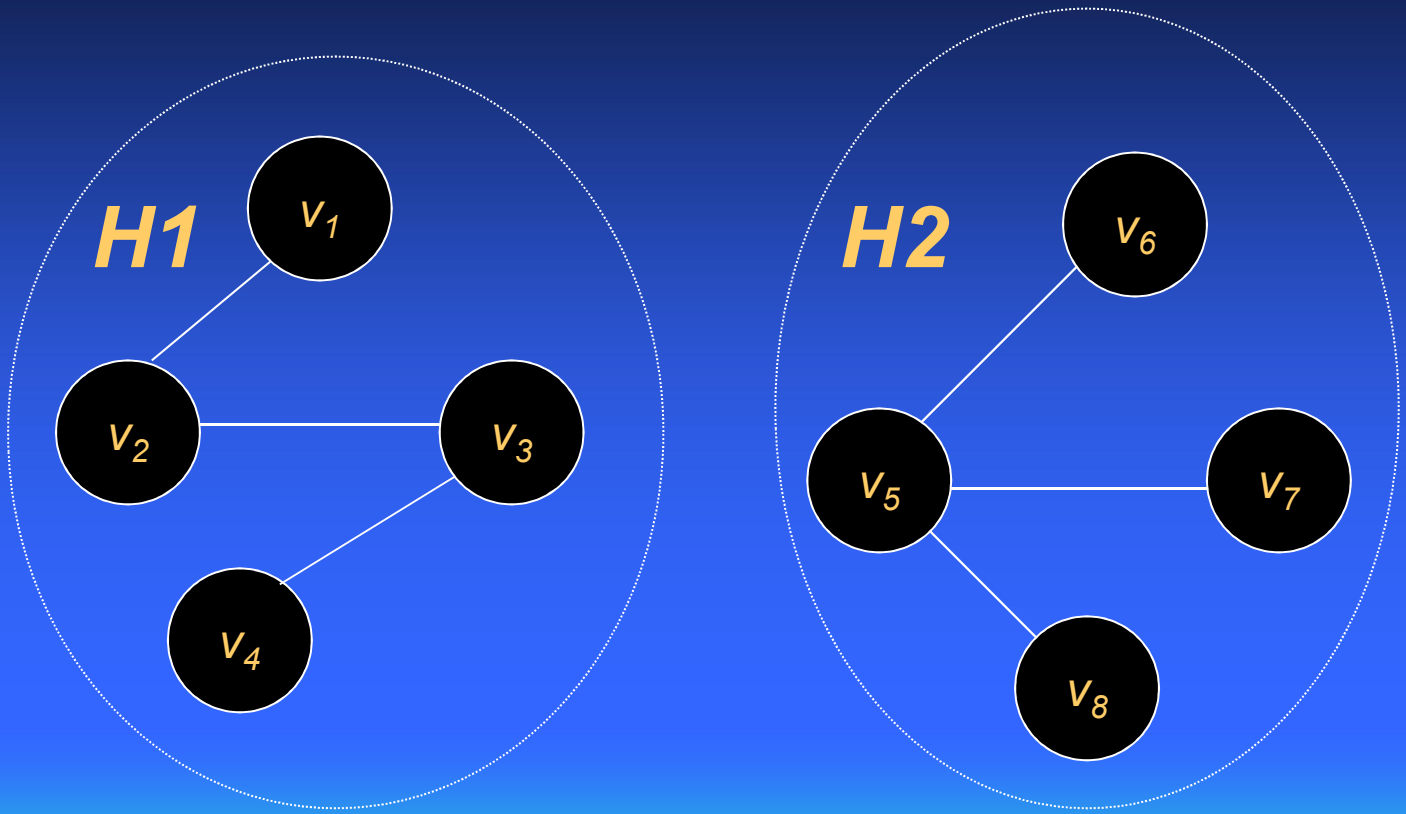
  v_1과 v_2는 연결.

그래프의 용어(계속)

❑ 연결성

무방향 그래프에서 모든 정점들의 쌍에 대하여 경로가 존재하면

그 그래프는 연결되었다 라고 한다.



연결되지 않은 그래프 G5

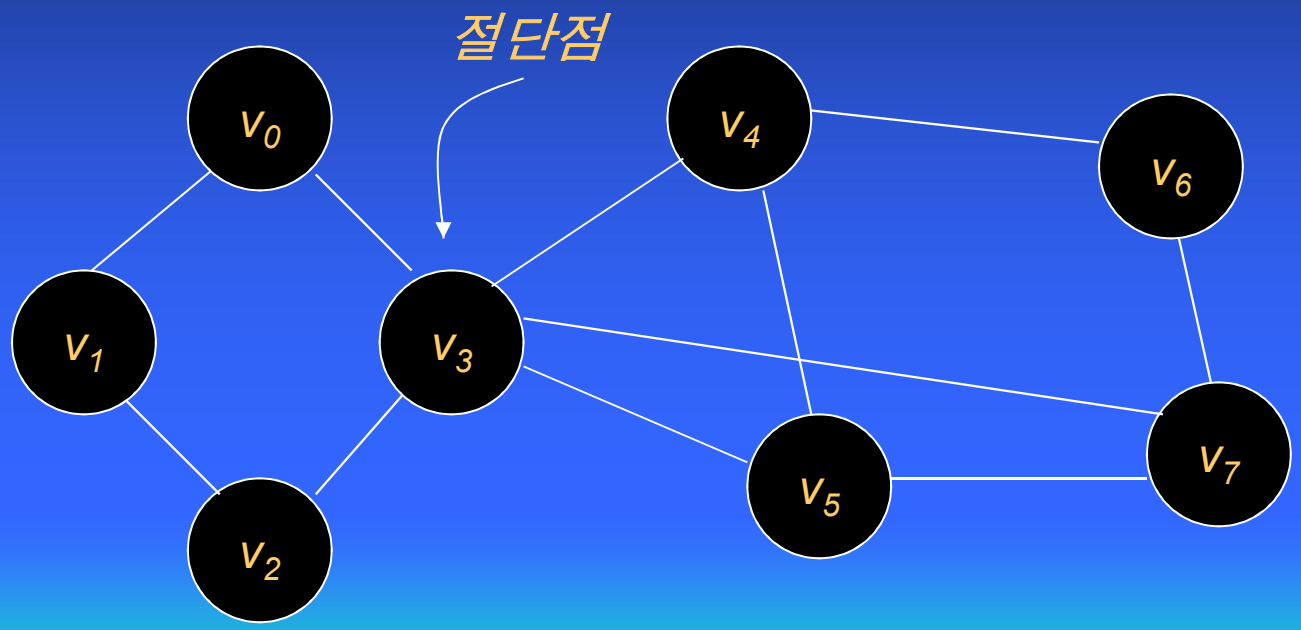
그래프의 용어(계속)

❑연결 요소(connected component)

연결 요소란 무방향성 그래프에서 최대로 연결된 부분 그래프를 말한다.

❑절단점

정점 v와 v에 부속된 모든 간선들을 모두 제거하면 그래프 G가 두 개 이상 연결 요소로 분리되는 정점 v를 절단점이라 한다.
이 때 절단점을 갖지 않는 연결 그래프를 이중 연결 그래프라 한다.

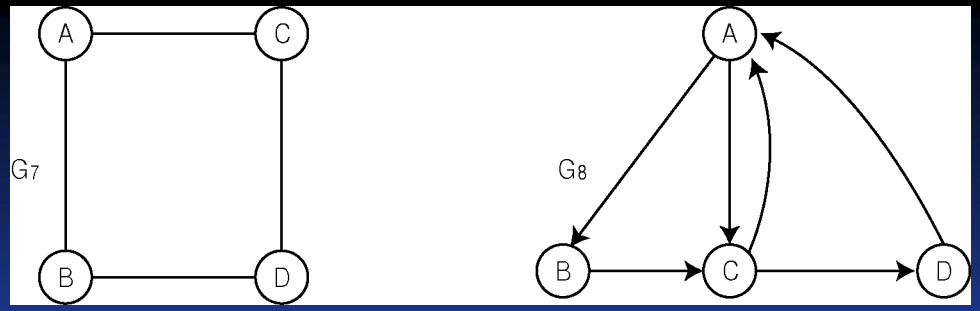


2. 그래프의 표현

2.1그래프 표현의 개념

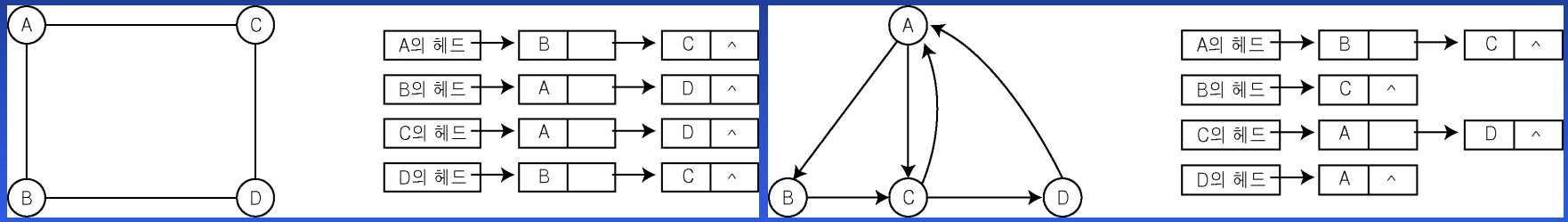
❑ 인접 행렬

그래프를 컴퓨터에서 표현하는 방법



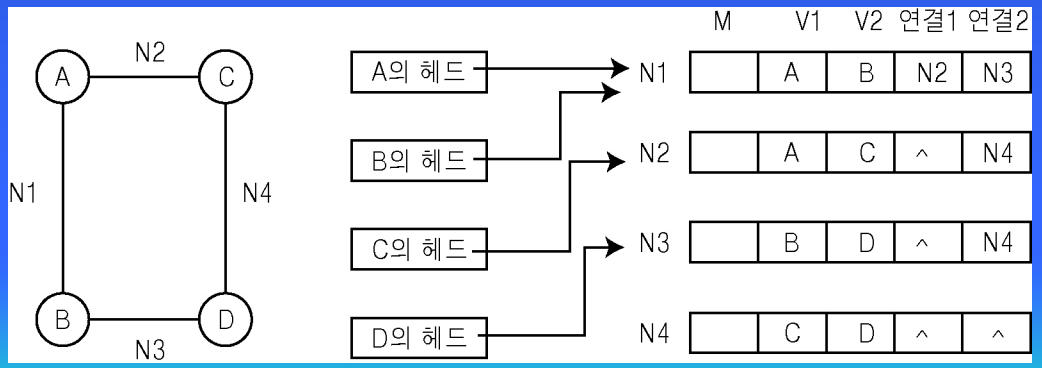
❑ 인접 리스트

각 정점에 인접한 간선들을 연결 리스트로 표현하는 방법



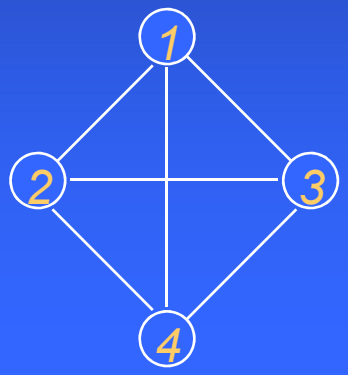
❑ 인접 다중 리스트

각 간선에 대해 하나의 노드만을 사용하고 이 노드들을 여러 개의 헤드 노드가 가리키도록 만든 리스트



2.2 인접행렬 표현법

- 정점 수만큼의 행과 열로 구성된 정방 행렬
- 정점 i와 j사이에 간선이 있으면 $a_{ij} = 1$, 없으면 $a_{ij} = 0$
- 무방향그래프에서 정점 i의 차수는 그 행의 합이다.
- 방향그래프에서 간선이 있으면 $a_{ij} = 1$, , 없으면 $a_{ij} = 0$
- 방향그래프에서 정점 i의
  - 진출차수는 그 행의 합
  - 집입차수는 그 열의 합



G_1

	열1	열2	열3	열4
행1	0	1	1	1
행2	1	0	1	1
행3	1	1	0	1
행4	1	1	1	0

G_1의 인접행렬

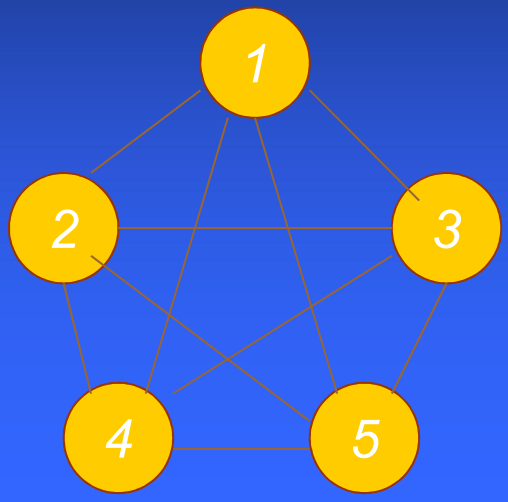
	열1	열2	열3
행1	0	1	0
행2	1	0	1
행3	0	0	0

G_3의 인접행렬

G_3

2.2 인접행렬 표현법(계속)

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



G2 : 완전한 그래프

	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	1	11
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	0

G2의 인접 행렬

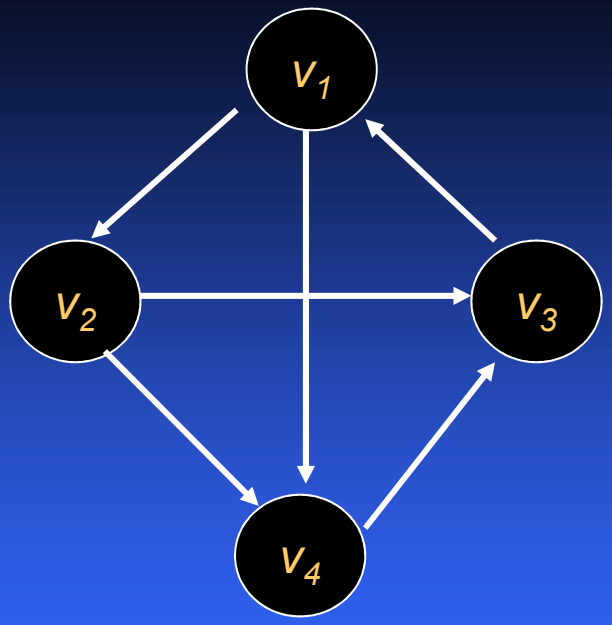
2.3 인접 리스트 표현법

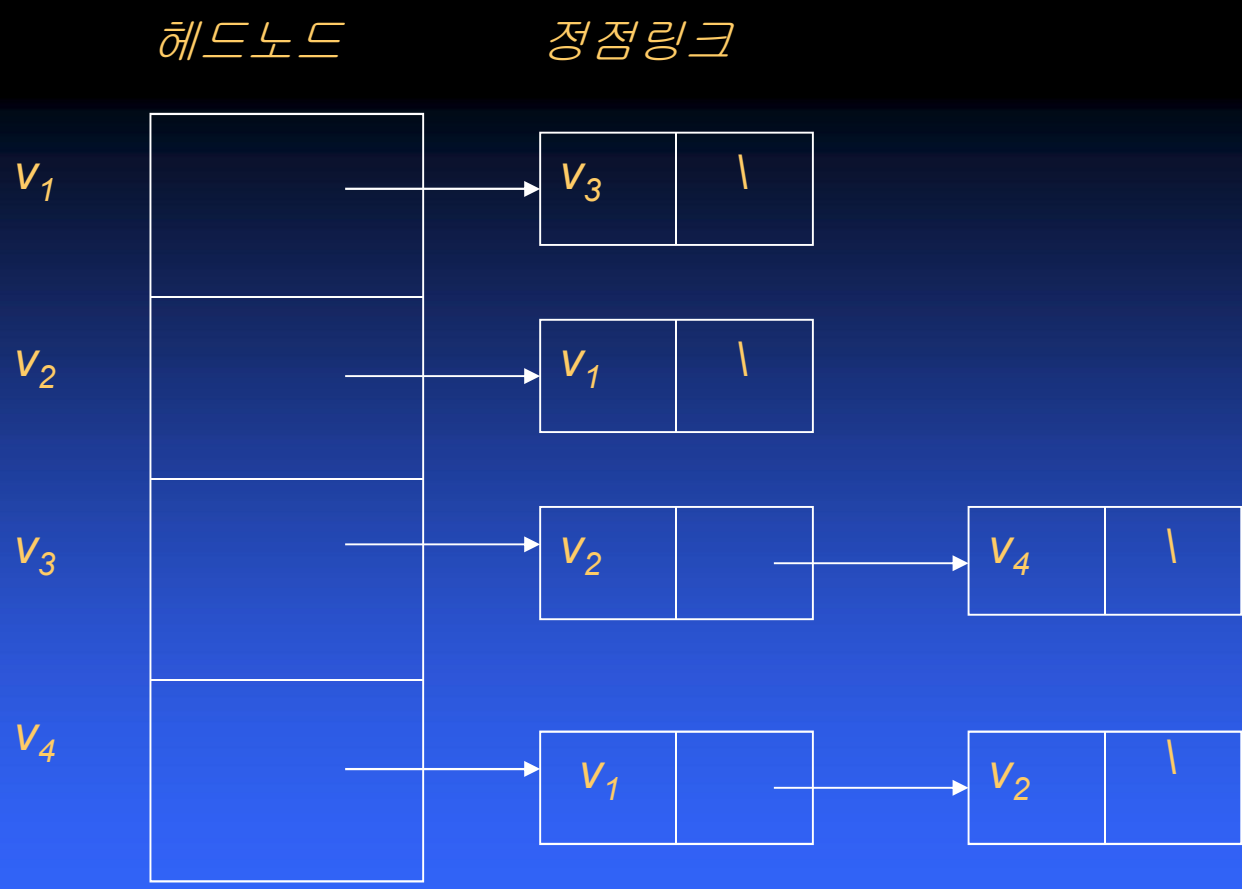
- 그래프의 각각의 정점에 대해 인접한 정점들을 연결 리스트로 표현
- 그래프를 연결된 정점들만을 이용하여 표현하므로서 기억장소의 낭비를 줄일 수 있다.
- 반대로 간선의 수가 상대적으로 많은 경우 인접행렬을 사용할 때보다 기억장소 낭비가 심해질 수 있다.
- 정점에 연결된 노드의 수를 이용하여 쉽게 정점의 차수를 구할 수 있으나, 방향그래프의 경우 진입 차수를 구하기 매우 어렵다는 단점을 갖는다.

  (역 인접리스트 이용)

- 진입 차수를 구하기 어려운 문제를 해결하기 위해서, 방향성 그래프의 각 정점으로 들어오는 간선과 인접한 정점으로 구성한 것이 역 인접 리스트이다.

2.3 인접 리스트 표현법

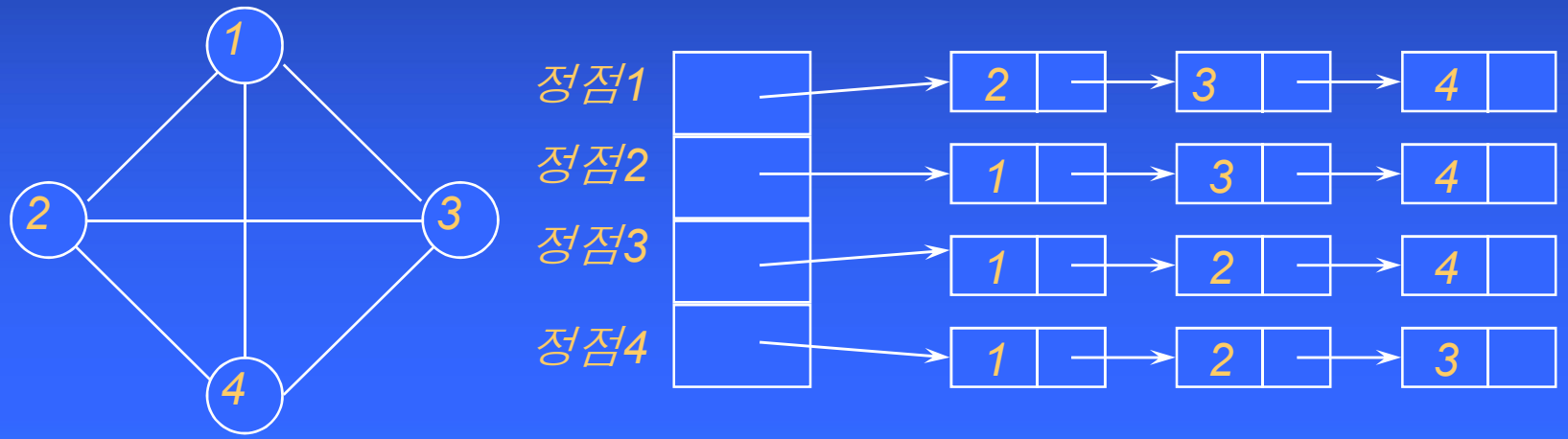




그래프 G6의 역 인접 리스트 표현

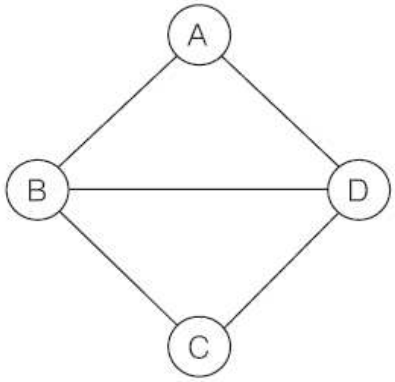
2.3 인접 리스트 표현법

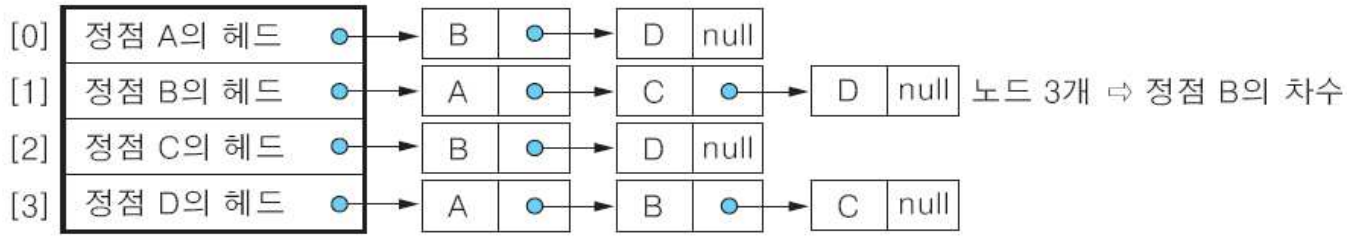
- 인접행렬의 n개의 행렬들을 n개의 연결리스트로 표현
- 그래프의 각 정점에 대해 한개의 연결리스트 존재
- 진출차수(out-degree) : 인접리스트의 노드의 수
- 진입차수(in-degree) : 역인접리스트의 노드의 수



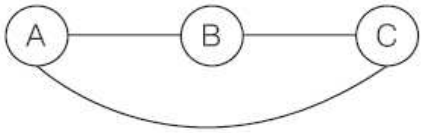
2.3 인접 리스트 표현법(계속)

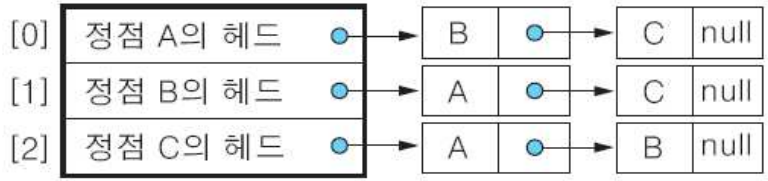
G1



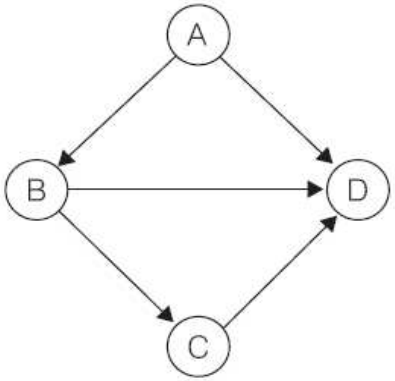


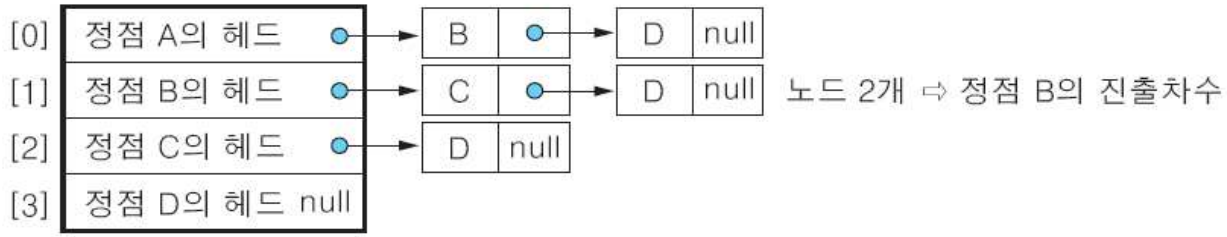
G2



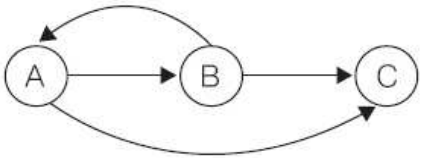


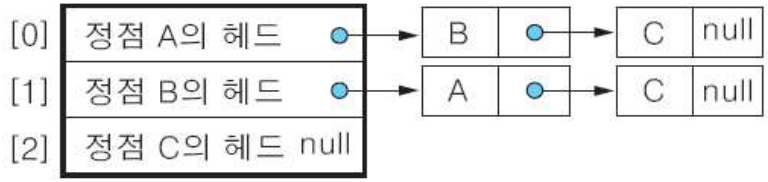
G3





G4

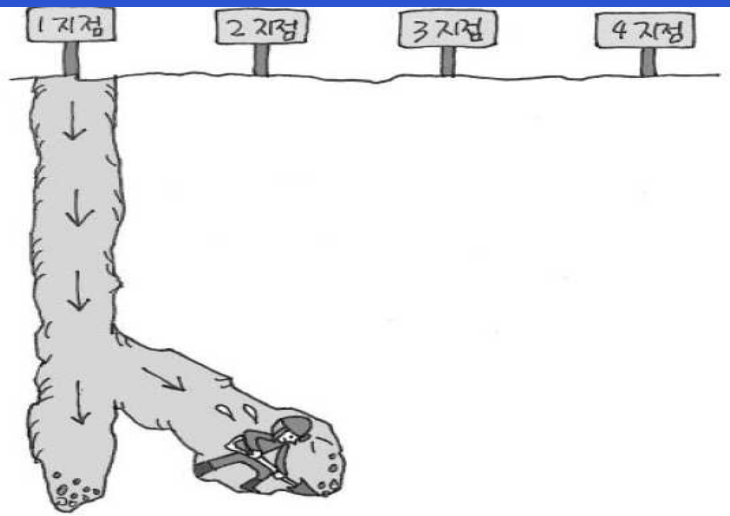




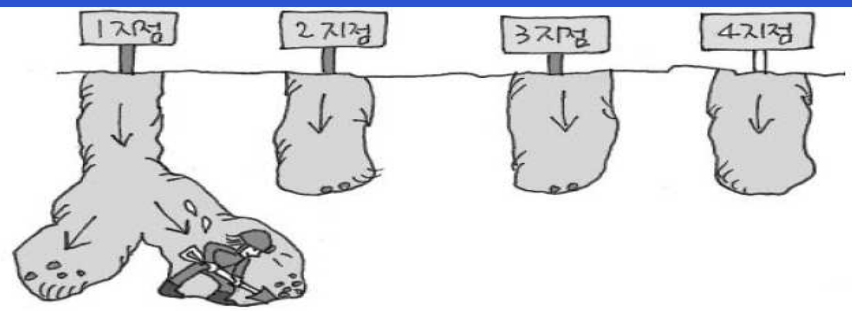
3. 그래프의 운행

3.1 그래프 운행의 개념

- 그래프 순회(graph traversal), 그래프 탐색(graph search)
  - 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문하여 처리하는 연산
  - 그래프 운행의 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 검색) DFS
    - 여러 지점을 고르게 파보고 물이 나오지 않으면, 파놓은 구덩이들을 다시 좀더 깊게 파는 방법 (☞ 너비 우선 탐색) BFS



(a) 깊이 우선 탐색의 예



(b) 너비 우선 탐색의 예

3.2 깊이 우선 검색(DFS)

- 깊이 우선 검색 (Depth First Search)

  (1) 시작 정점 v를 결정하여 방문한다.

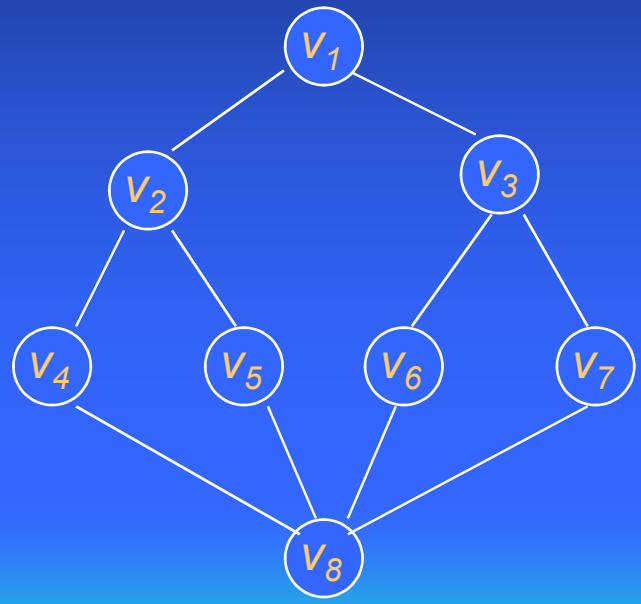
  (2) v에 인접된 정점 가운데 방문되지 않은 정점w를 선택하여 DFS 방식을 시작한다.

  (3) 모든 인접 정점을 방문한 정점 v를 만나면 방문되지 않은 인접 정점을 가졌던 마지막 정점으로 되돌아가서 시작한다.

  (4) 더이상 방문할 정점이 없으면 DFS는 끝이 난다.

- 알고리즘

```
procedure DFS(v)
   VISITED(v) = 1
   for ( v에 인접한 정점 w ) do
      if ( VISITED(w) = 0 )
            call DFS(w)
   end
end DFS
```



- $v_1, v_2, v_4, v_8, v_5, v_6, v_3, v_7$

3.2 깊이 우선 검색(DFS)

- 깊이 우선 검색의 수행 순서

(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.

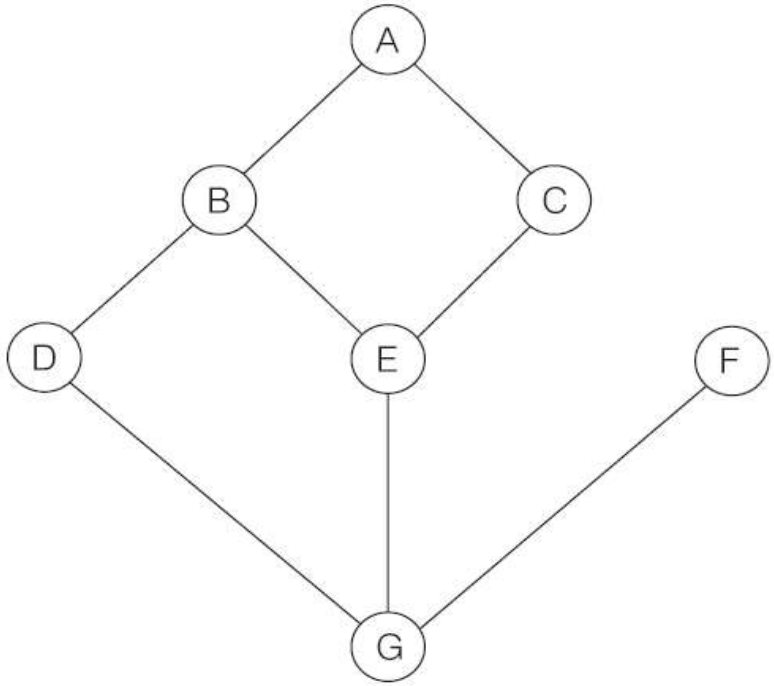
② 방문하지 않은 정점이 없으면, 탐색의 방향을 바꾸기 위해서 스택을 pop하여 받은 가장 마지막 방문 정점을 v로 하여 다시 (2)를 수행한다.

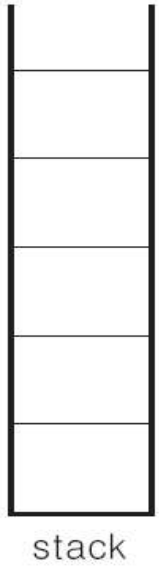
(3) 스택이 공백이 될 때까지 (2)를 반복한다.

3.2 깊이 우선 검색(DFS)

- 깊이 우선 검색
  - 예) 그래프 G9에 대한 깊이 우선 탐색
  - 초기상태 : 배열 visited를 False로 초기화하고, 공백 스택을 생성

초기상태 : 배열 visited를 False로 초기화하고 공백 스택을 생성한다.



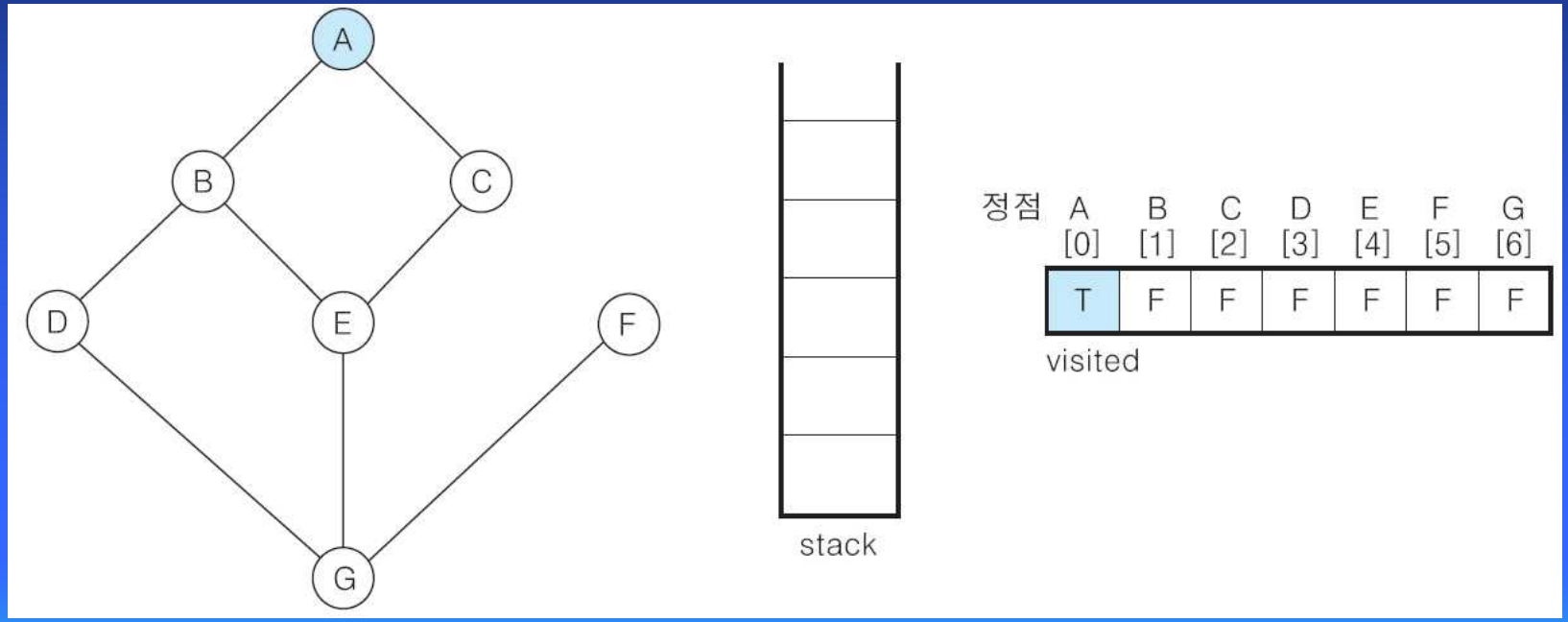


정점	A [0]	B [1]	C [2]	D [3]	E [4]	F [5]	G [6]
visited	F	F	F	F	F	F	F

3.2 깊이 우선 검색(DFS)

① 정점 **A**를 시작으로 깊이 우선 검색을 시작

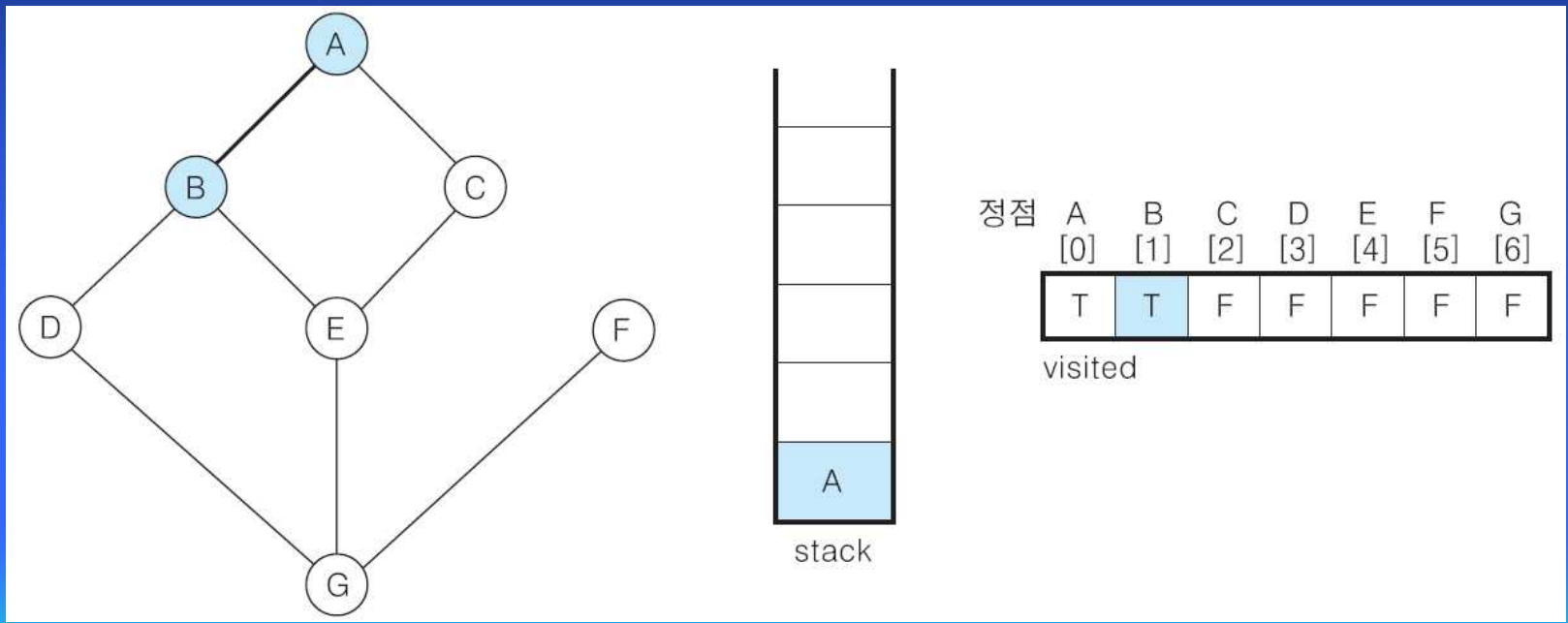
visited[A]□true;
A 방문;



3.2 깊이 우선 검색(DFS)

② 정점 A에 방문하지 않은 정점 B, C가 있으므로 A를 스택에 push 하고, 인접정점 B와 C 중에서 오름차순에 따라 B를 선택하여 탐색 진행

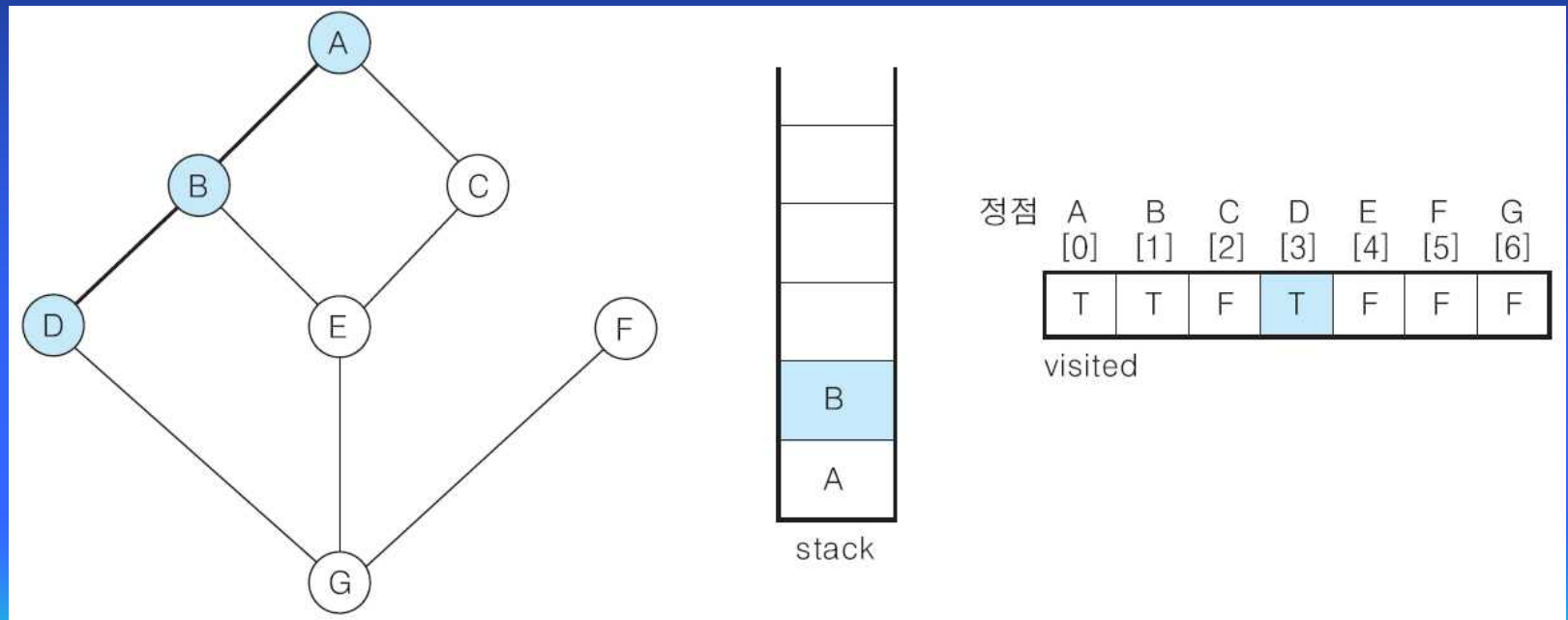
```
push(stack, A);
 visited[B]□true;
 B 방문;
```



3.2 깊이 우선 검색(DFS)

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

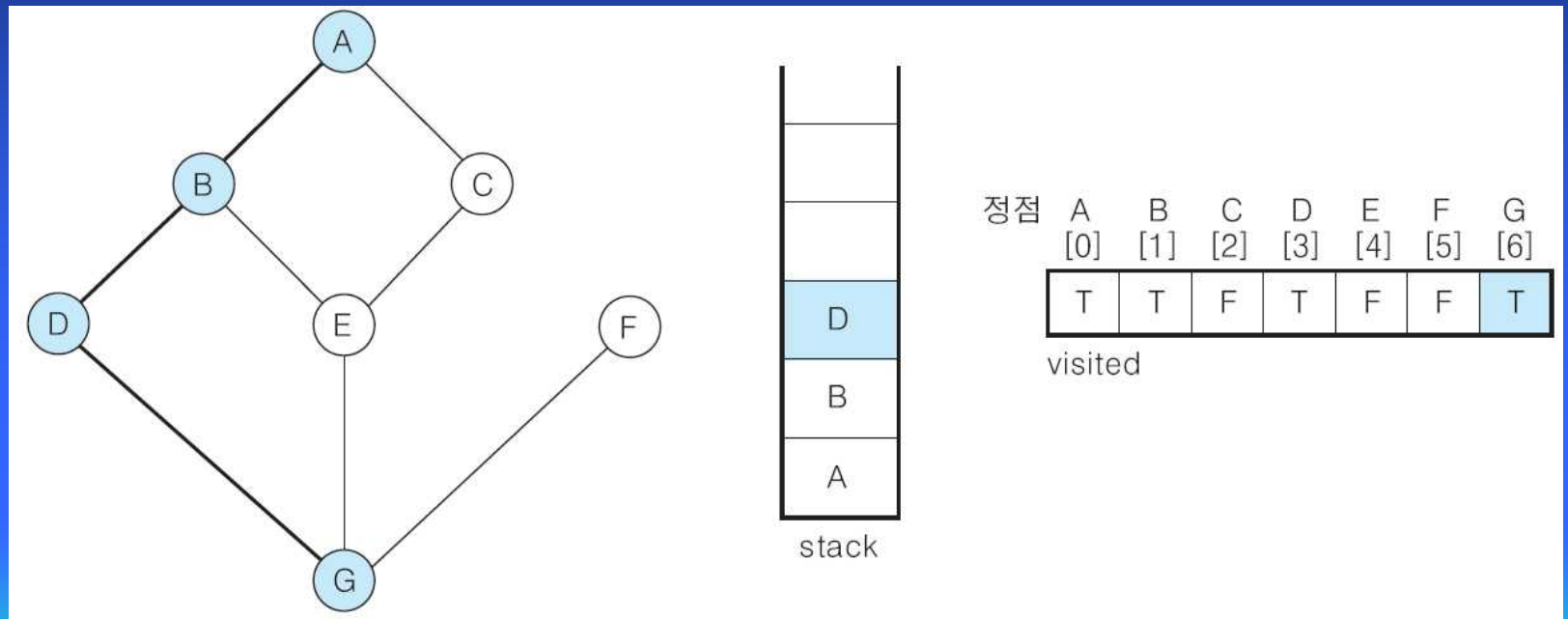
```
push(stack, B);
visited[D]□true;
D 방문;
```



3.2 깊이 우선 검색(DFS)

④ 정점 D에 방문하지 않은 정점 G가 있으므로 D를 스택에 push 하고, 인접정점 G를 선택하여 탐색을 계속한다.

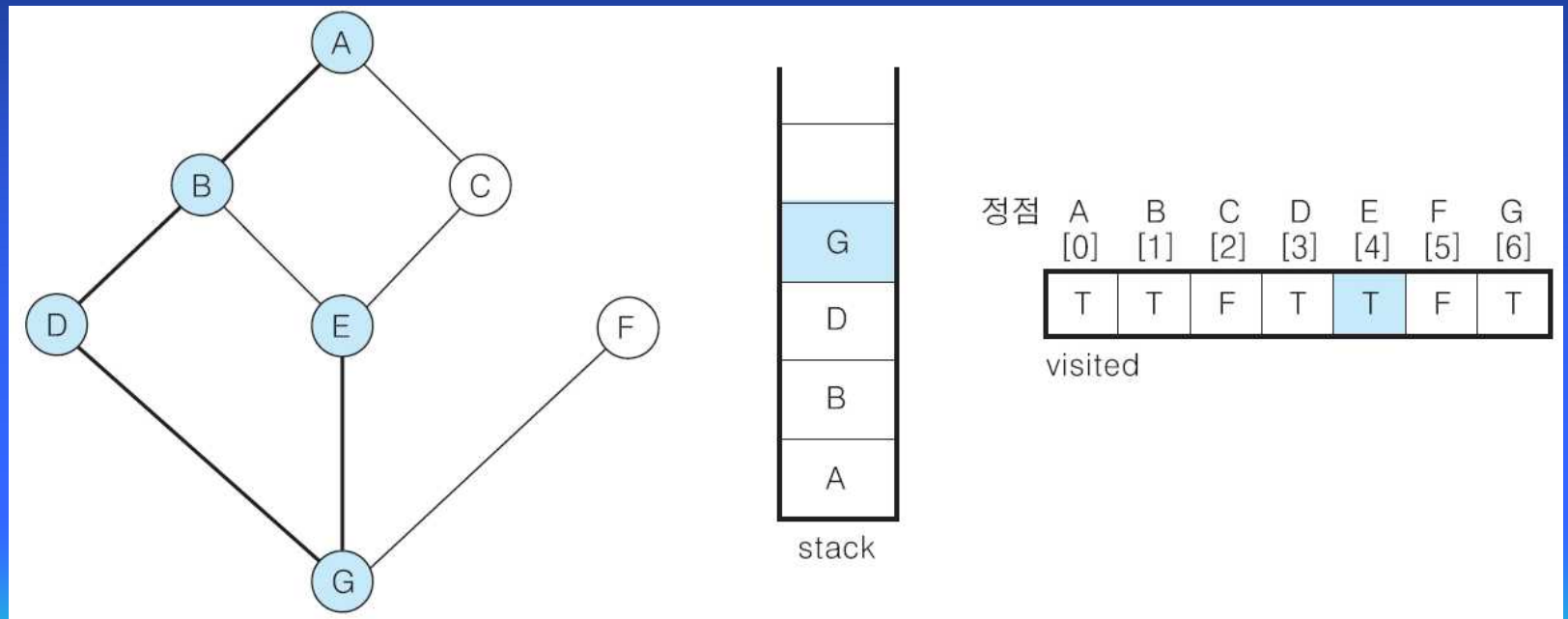
```
push(stack, D);
visited[G]□true;
G 방문;
```



3.2 깊이 우선 검색(DFS)

⑤ 정점 G에 방문하지 않은 정점 E, F가 있으므로 G를 스택에 push 하고, 인접정점 E와 F 중에서 오름차순에 따라 E를 선택하여 탐색을 계속.

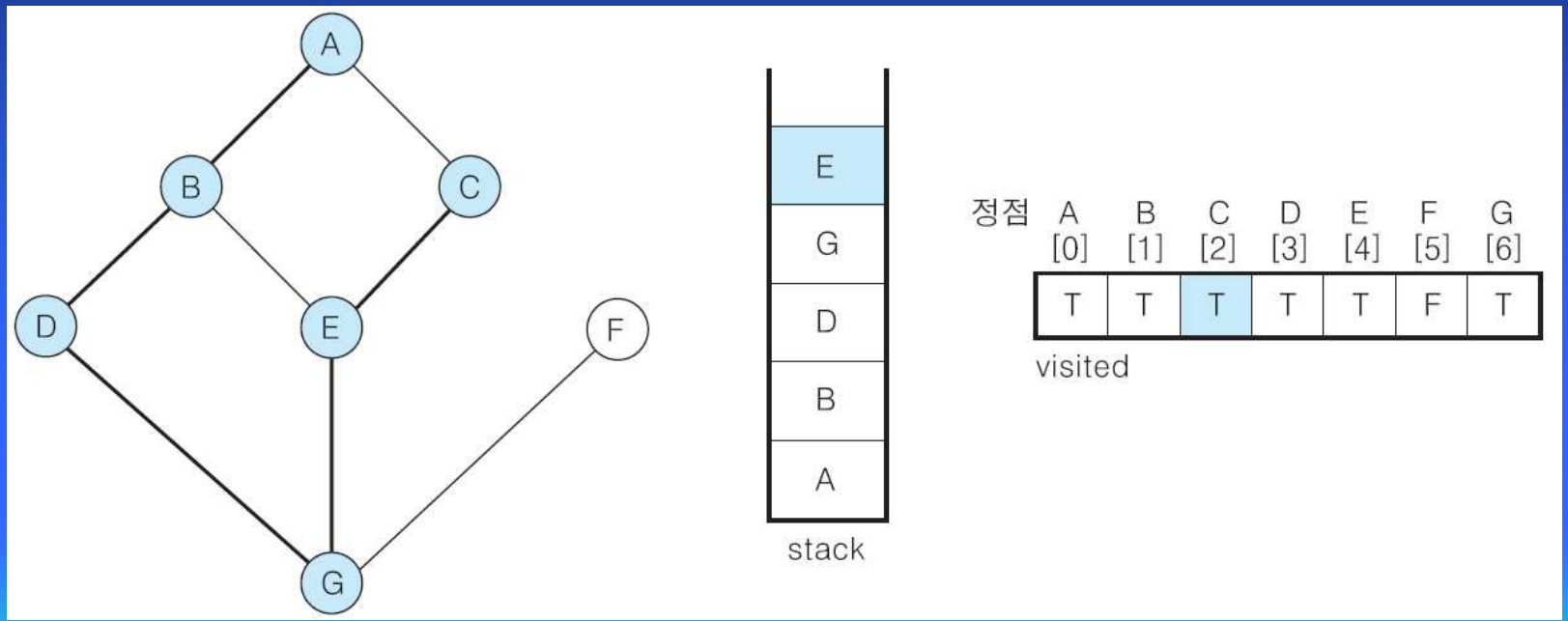
```
push(stack, G);
visited[E]□true;
E 방문;
```



3.2 깊이 우선 검색(DFS)

⑥ 정점 E에 방문하지 않은 정점 C가 있으므로 E를 스택에 push 하고, 인접정점 C를 선택하여 탐색을 계속한다.

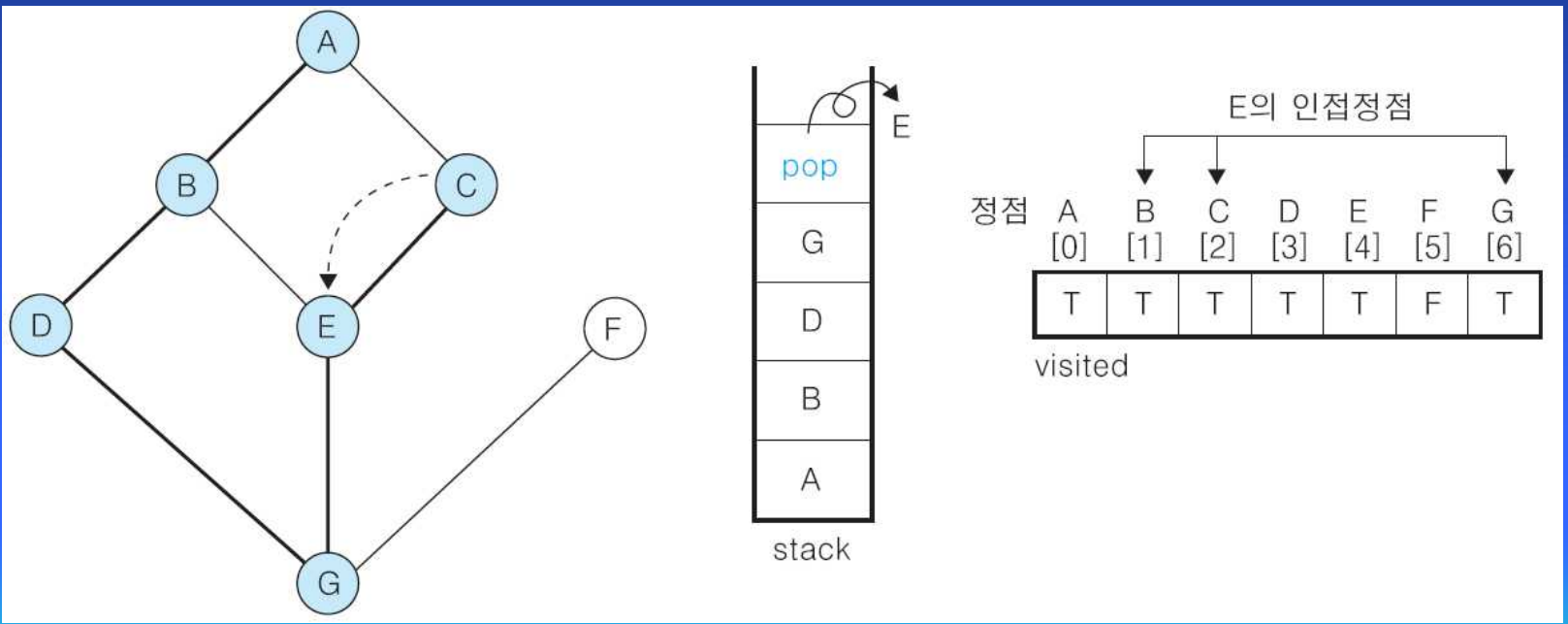
```
push(stack, E);
 visited[C]□true;
 C 방문;
```



3.2 깊이 우선 검색(DFS)

⑦ 정점 C에서 방문하지 않은 인접정점이 없으므로, 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 E에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

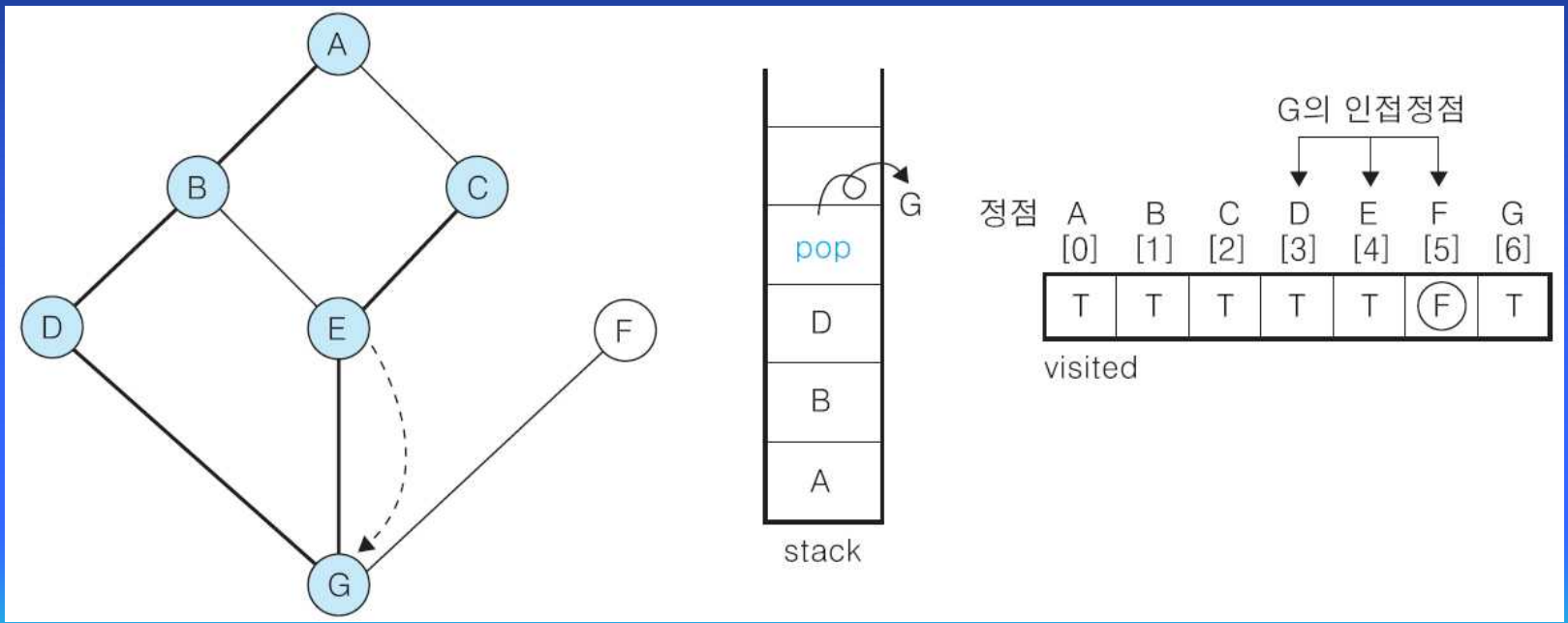
```
pop(stack);
```



3.2 깊이 우선 검색(DFS)

⑧ 정점 **E**는 방문하지 않은 인접정점이 없으므로, 다시 스택을 pop 하여 받은 정점 G에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

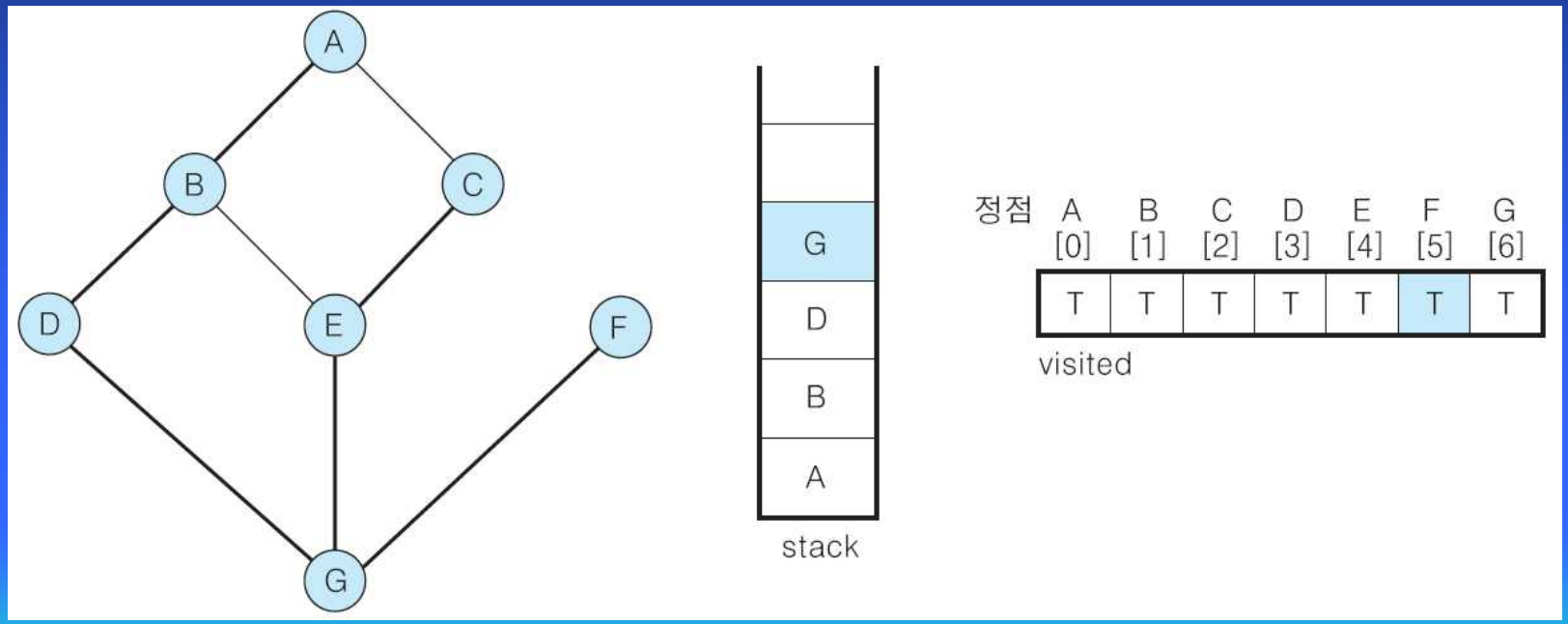
```
pop(stack);
```



3.2 깊이 우선 검색(DFS)

⑨ 정점 G에 방문하지 않은 정점 F가 있으므로 G를 스택에 push 하고, 인접정점 F를 선택하여 탐색을 계속한다.

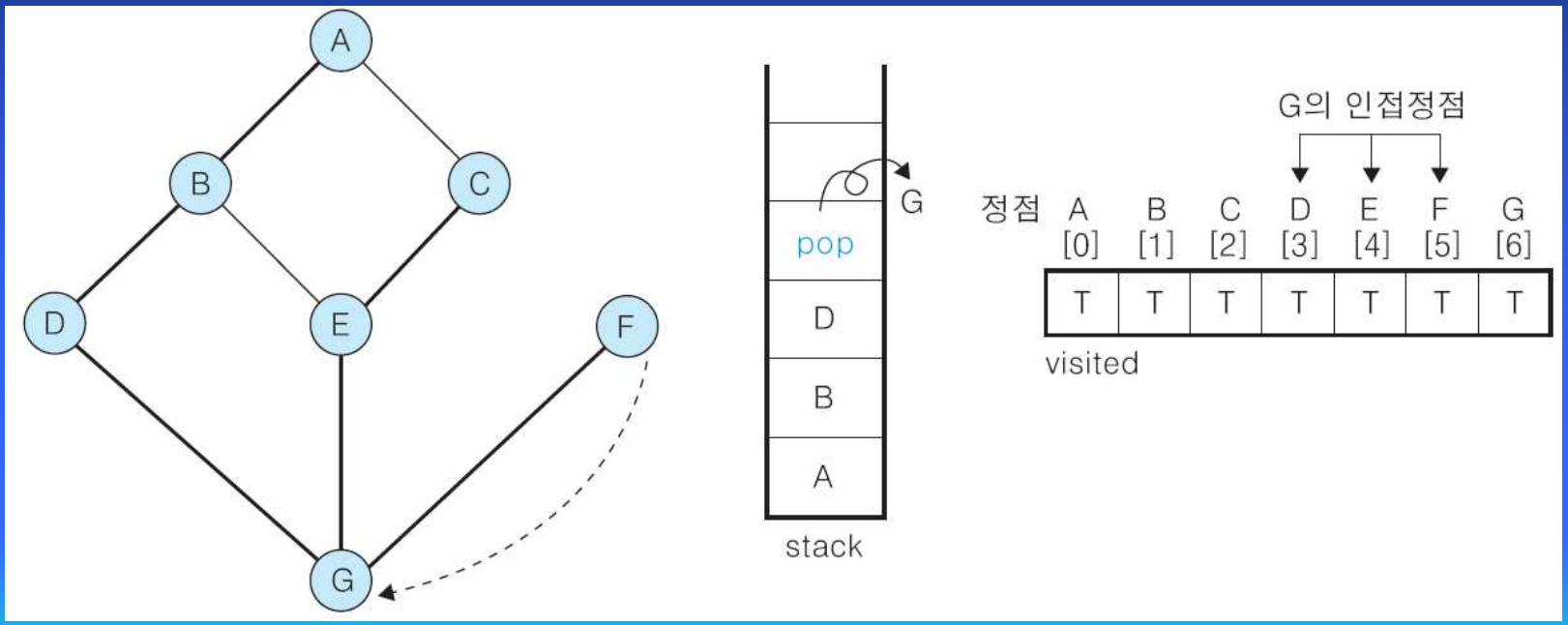
```
push(stack, G);
 visited[F]□true;
 F 방문;
```



3.2 깊이 우선 검색(DFS)

⑩ 정점 F에서 방문하지 않은 인접정점이 없으므로, 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 G에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

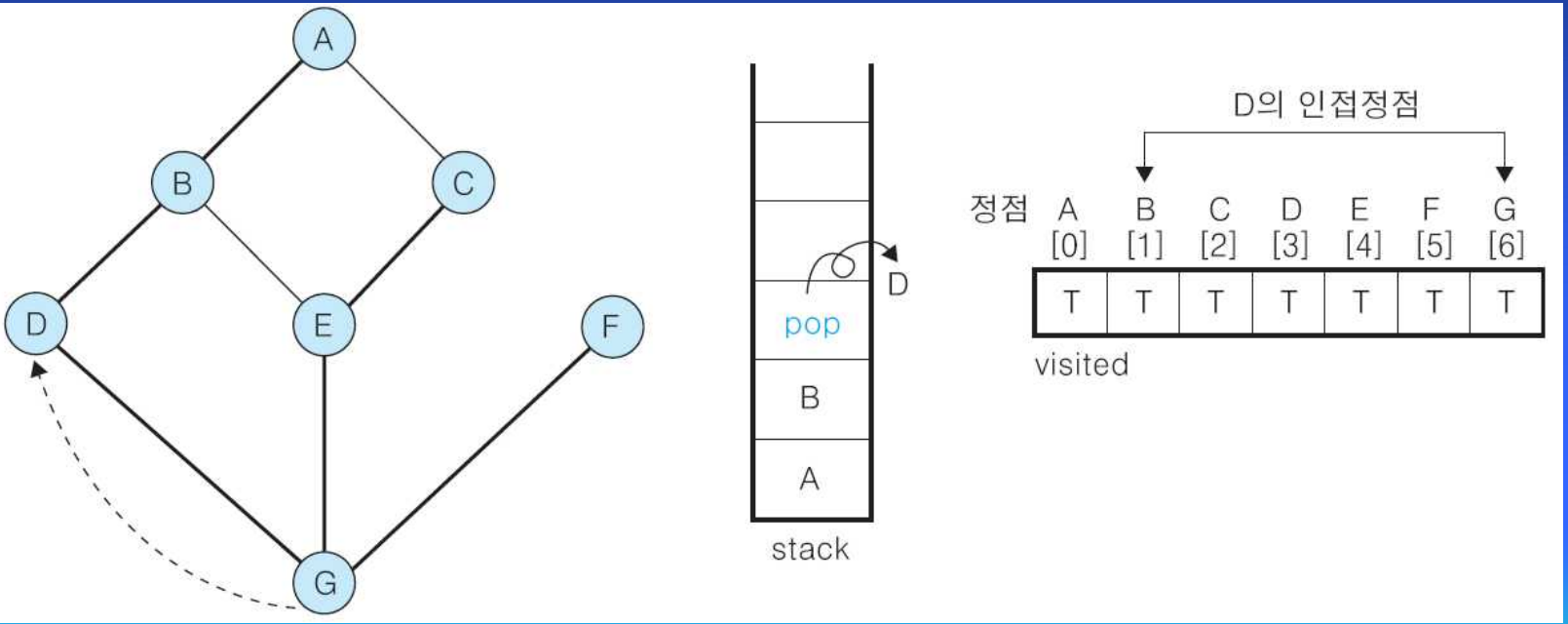
```
pop(stack);
```



3.2 깊이 우선 검색(DFS)

⑪ 정점 G에서 방문하지 않은 인접정점이 없으므로, 다시 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 D에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

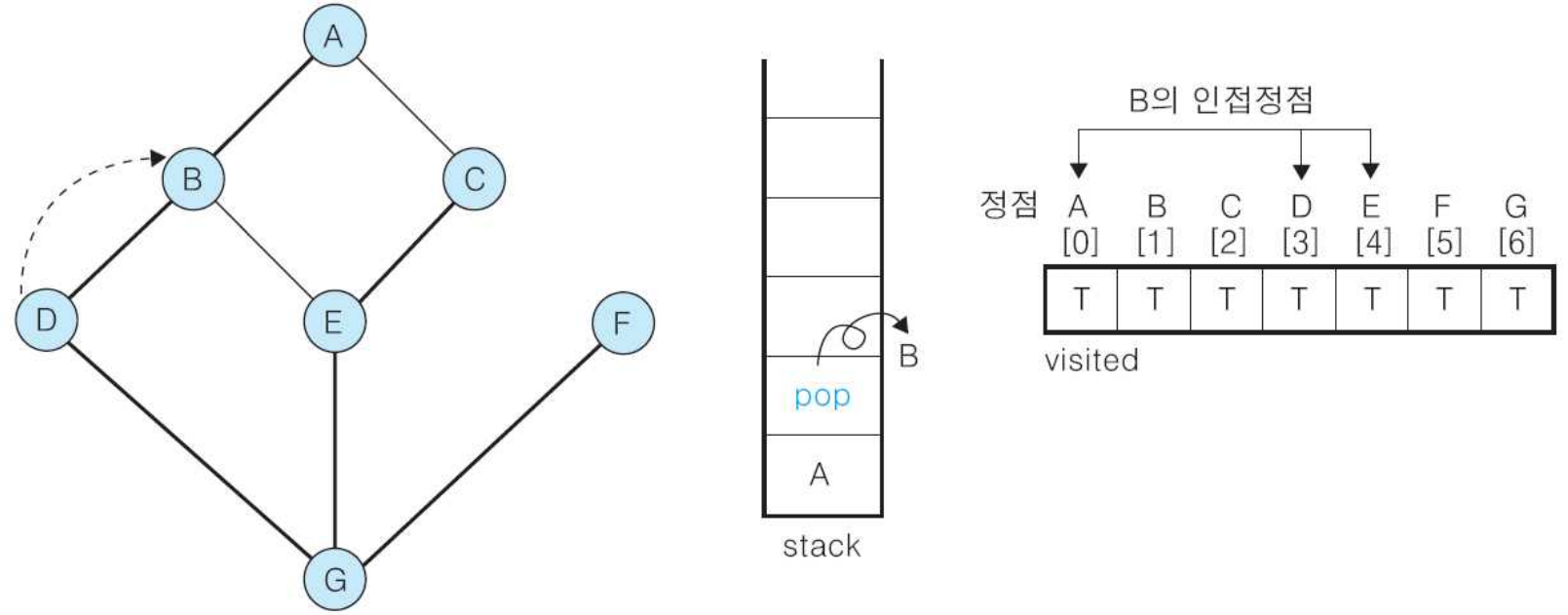
```
pop(stack);
```



3.2 깊이 우선 검색(DFS)

⑫ 정점 D에서 방문하지 않은 인접정점이 없으므로, 다시 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 B에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

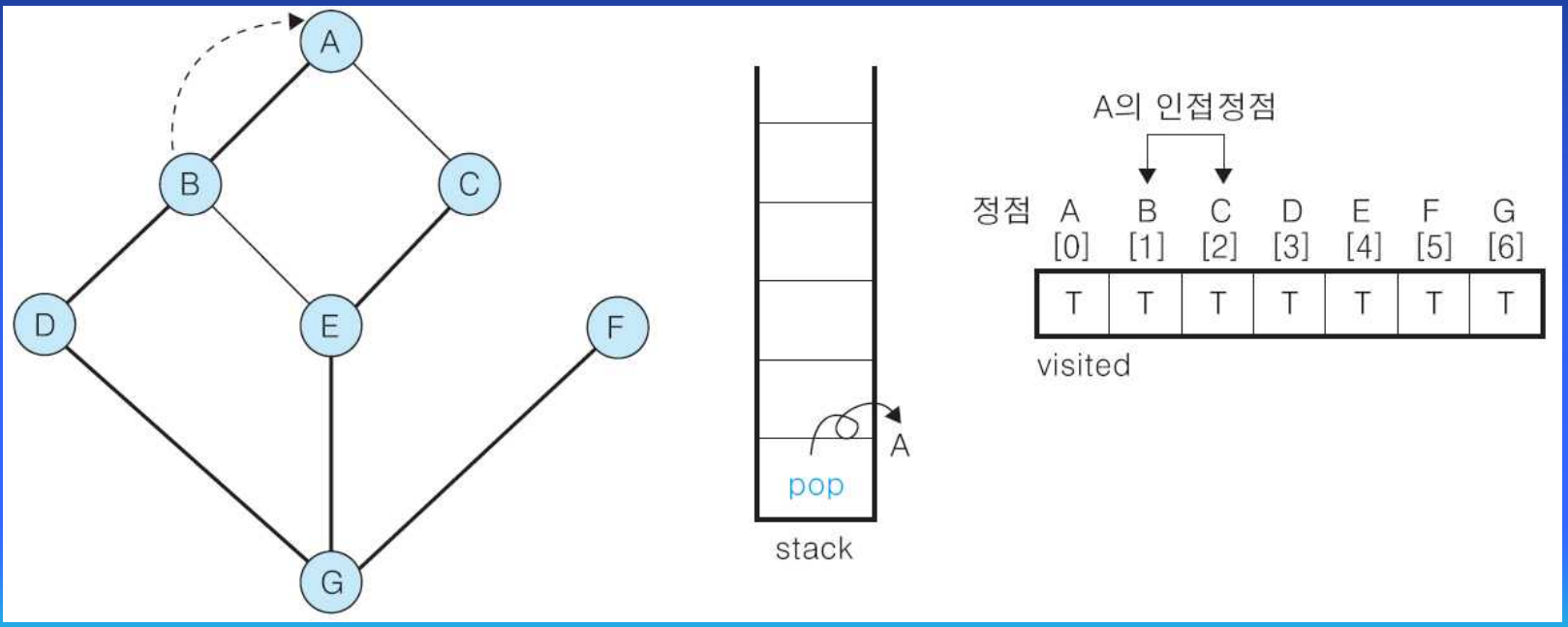
```
pop(stack);
```



3.2 깊이 우선 검색(DFS)

⑬ 정점 B에서 방문하지 않은 인접정점이 없으므로, 다시 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 A에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

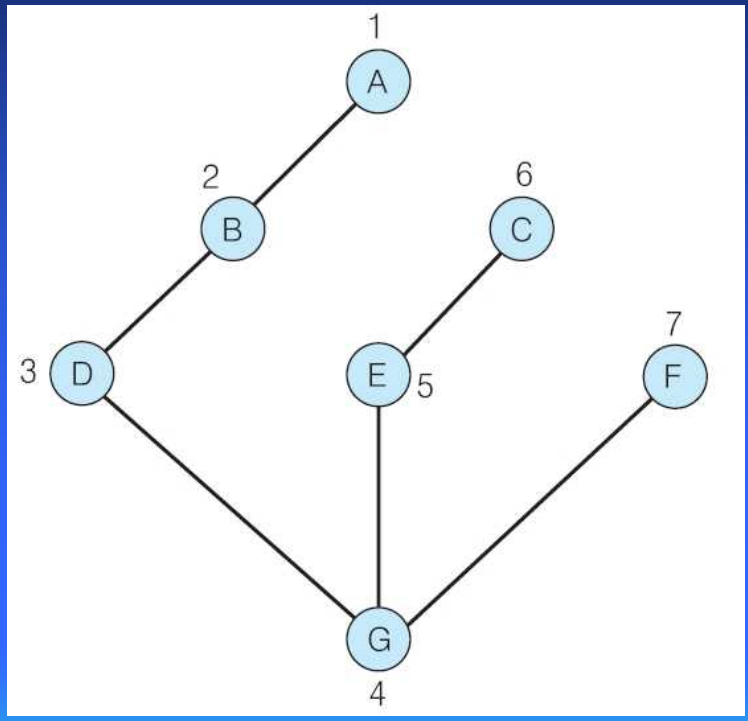
```
pop(stack);
```



3.2 깊이 우선 검색(DFS)

⑭ 정점 A에서 방문하지 않은 인접정점이 없으므로, 마지막 정점으로 돌아가기 위해 스택을 pop 하는데 스택이 공백이므로 깊이 우선 탐색을 종료한다.

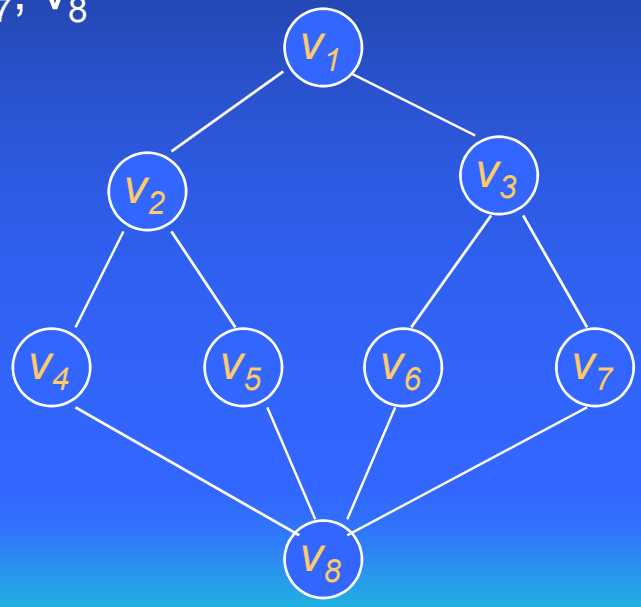
☞그래프 G9의 깊이 우선 탐색 경로 : A-B-D-G-E-C-F



3.3 너비 우선 검색(BFS)

- 너비우선 검색 (Breath First Search)
  - (1) 시작되는 정점 v를 결정하여 방문한다.
  - (2) 정점 v에 인접하고 방문하지 않은 모든 정점을 방문하고 다시 이 정점에 인접하고 방문하지 않은 모든 정점에 대해 BFS를 계속한다.
  - (3) 더이상 방문할 정점이 없을때 BFS는 끝이 난다.

- V_1, V_2, V_3, V_4,V_5, V_6, V_7, V_8



3.3 너비 우선 검색(BFS)

- 너비 우선 검색(breadth first search : BFS)
  - 순회 방법
    - 시작 정점으로부터 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 선입선출의 구조를 갖는 큐를 사용

3.3 너비 우선 검색(BFS)

- 너비 우선 검색의 수행 순서

(1) 시작 정점 v를 결정하여 방문한다.

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

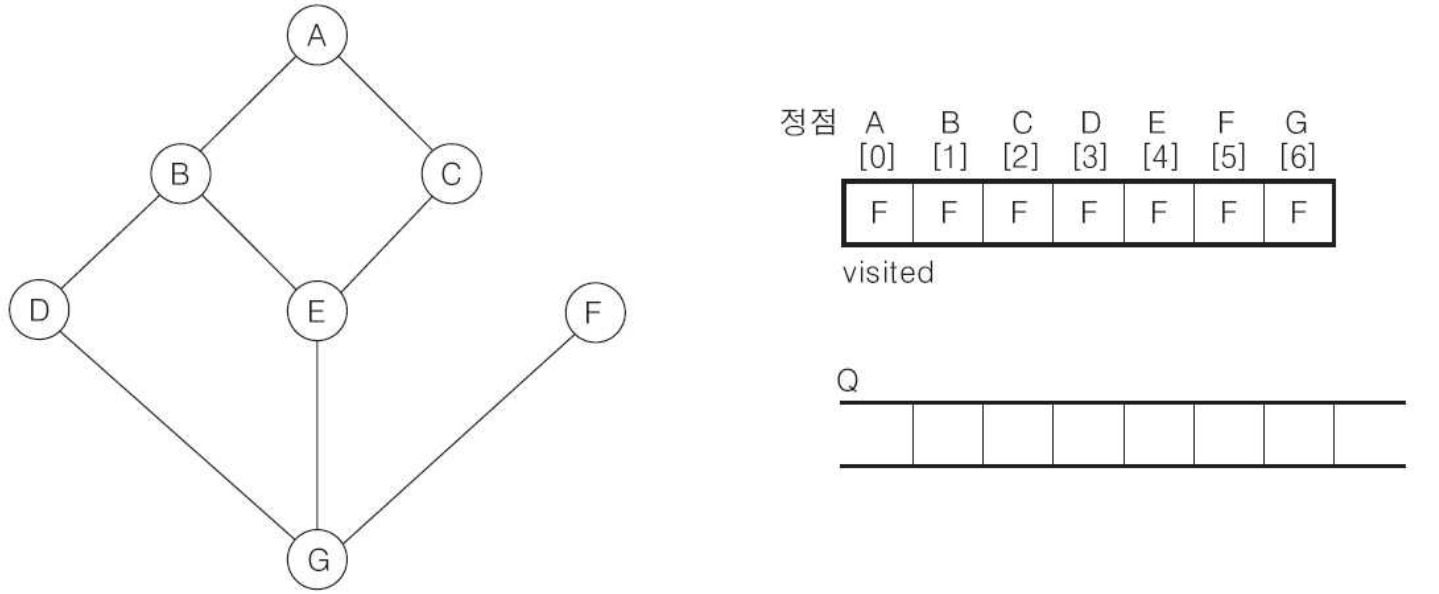
(3) 방문하지 않은 인접한 정점이 없으면, 방문했던 정점에서 인접한 정점들을 다시 차례로 방문하기 위해서 큐에서 deQueue하여 구한 정점에서 (2)를 반복한다.

(4) 큐가 공백이 될 때까지 (2)~(3)을 반복한다.

3.3 너비 우선 검색(BFS)

- 너비 우선 검색
  - 예) 그래프 G9에 대한 너비 우선 탐색
  - 초기상태 : 배열 visited를 False로 초기화하고, 공백 큐를 생성

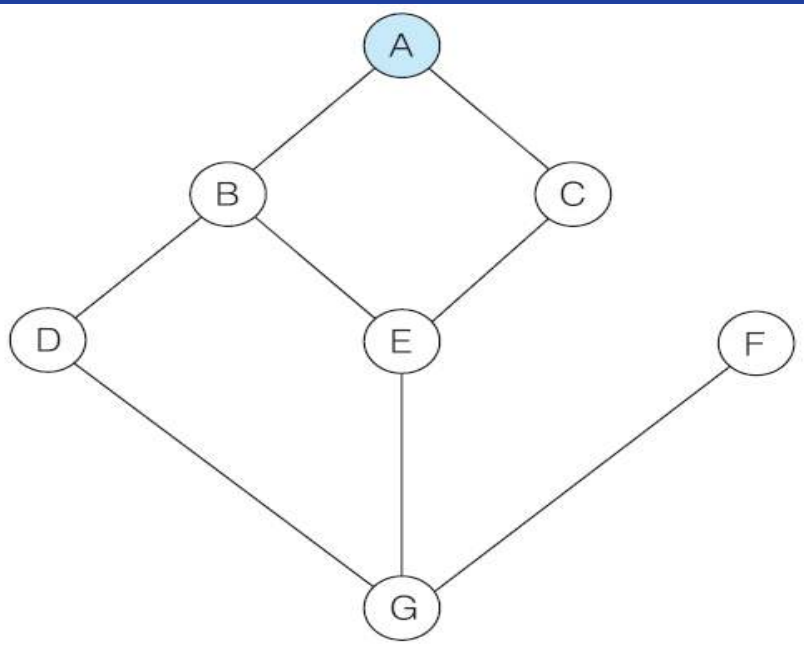
초기 상태 : 배열 visited를 False로 초기화하고 공백 큐를 생성한다.

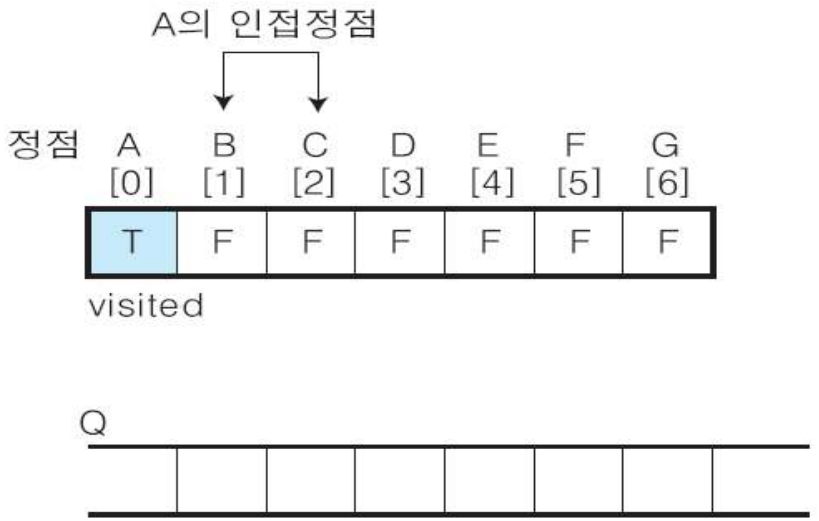


3.3 너비 우선 검색(BFS)

① 정점 A를 시작으로 너비 우선 탐색을 시작한다.

```
visited[A]←true;
A 방문;
```

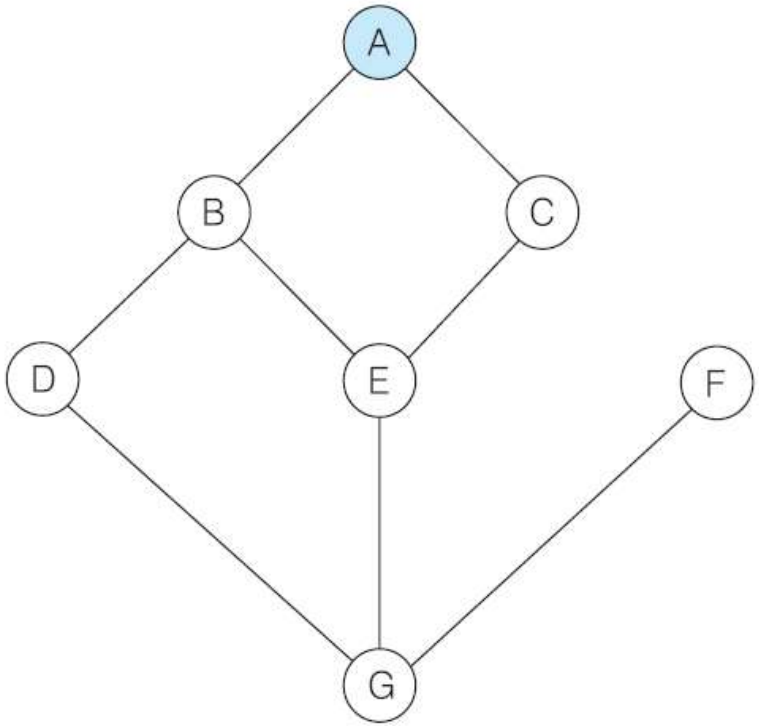


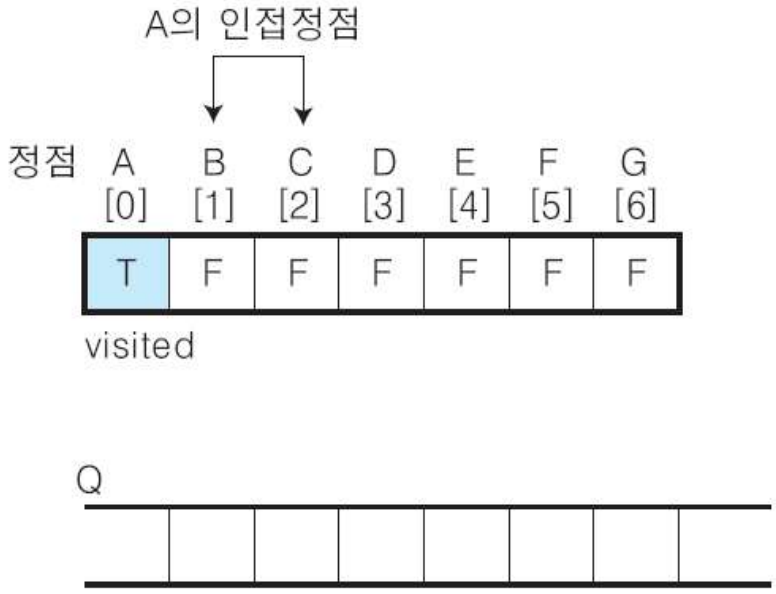


3.3 너비 우선 검색(BFS)

② 정점 A의 방문 안한 모든 인접정점 B, C를 방문하고, 큐에 enQueue 한다.

```
visited[(A의 방문 안한 인접정점 B와 C)]←true;
  (A의 방문 안한 인접정점 B와 C) 방문;
  enQueue(Q, (A의 방문 안한 인접정점 B와 C));
```

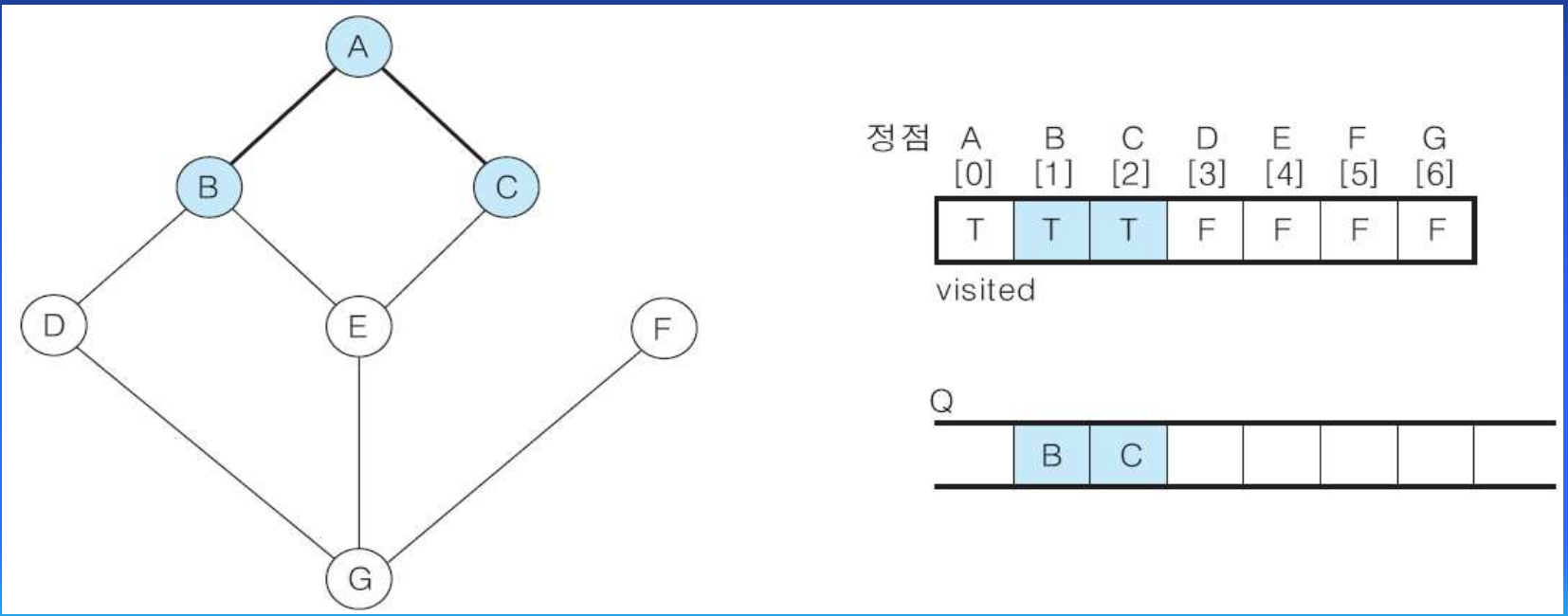




3.3 너비 우선 검색(BFS)

③ 정점 A에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 B를 구한다.

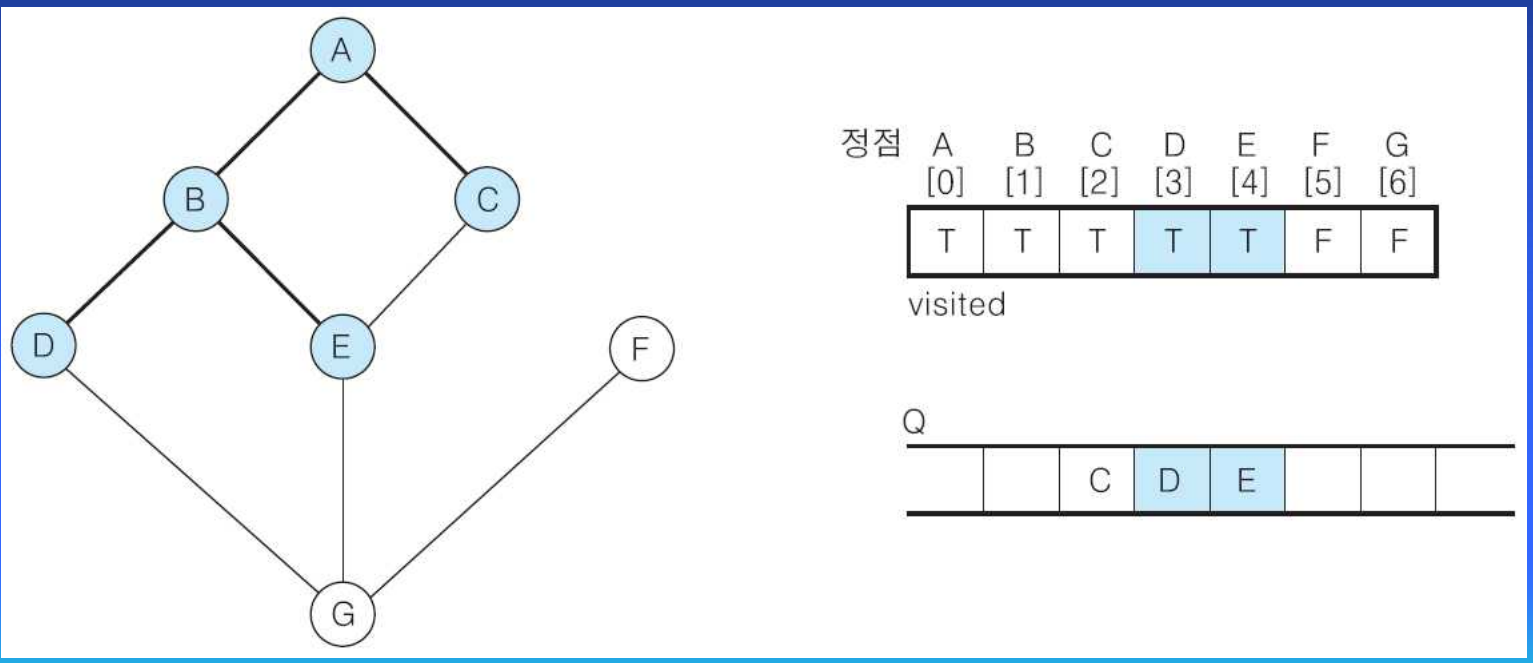
```
v ← deQueue(Q);
```



3.3 너비 우선 검색(BFS)

④ 정점 B의 방문 안한 모든 인접정점 D, E를 방문하고 큐에 enQueue 한다.

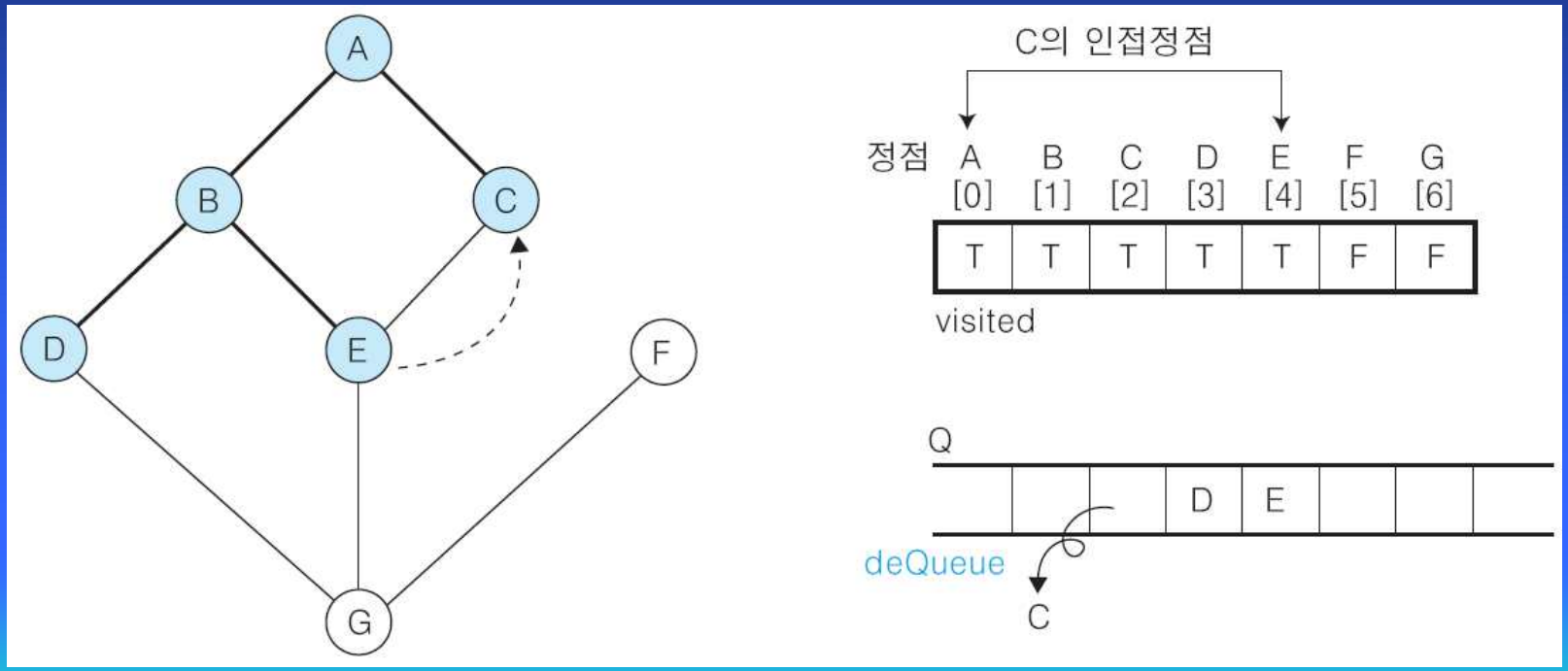
visited[(B의 방문 안한 인접정점 D와 E)]←true;
(B의 방문 안한 인접정점 D와 E) 방문;
enQueue(Q, (B의 방문 안한 인접정점 D와 E));



3.3 너비 우선 검색(BFS)

⑤ 정점 B에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 C를 구한다.

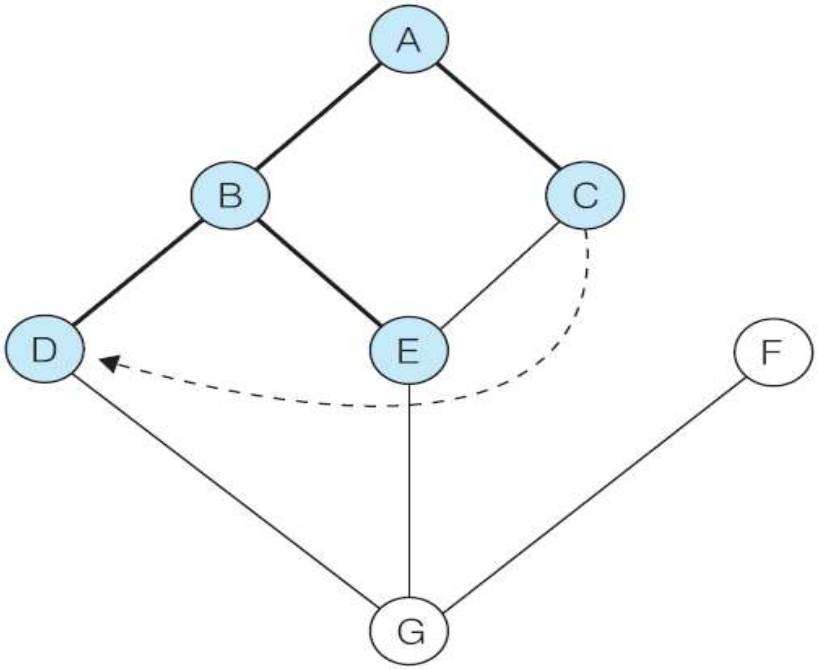
```
v ← deQueue(Q);
```

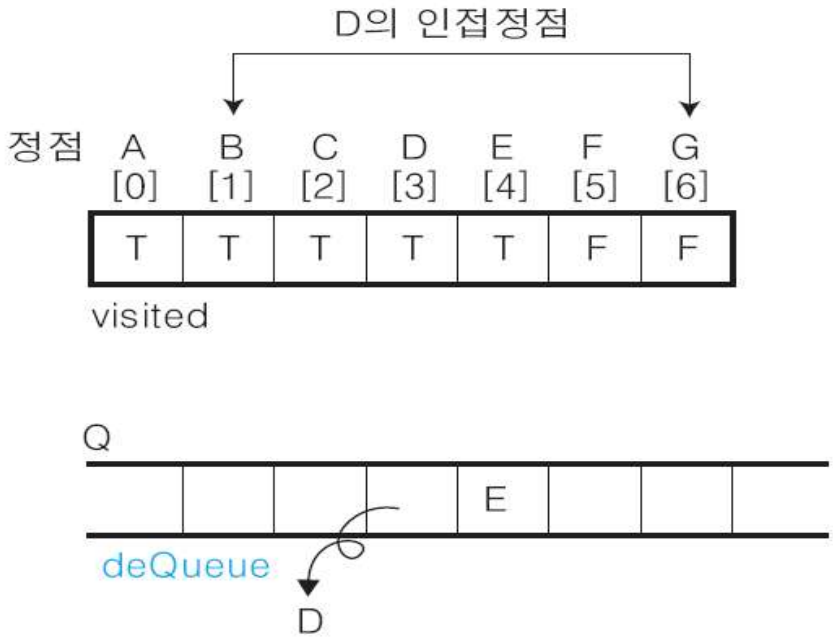


3.3 너비 우선 검색(BFS)

⑥ 정점 C에는 방문 안한 인접정점이 없으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 D를 구한다.

```
v ← deQueue(Q);
```

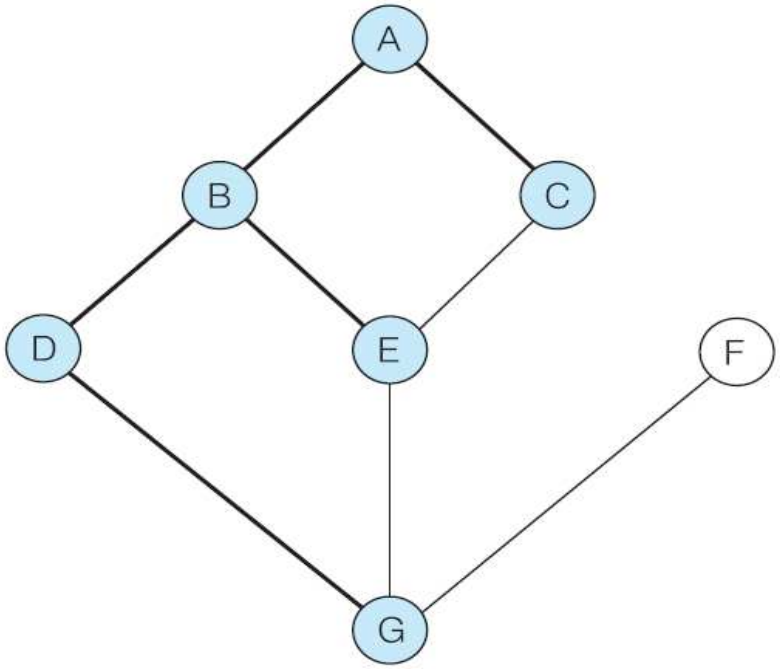


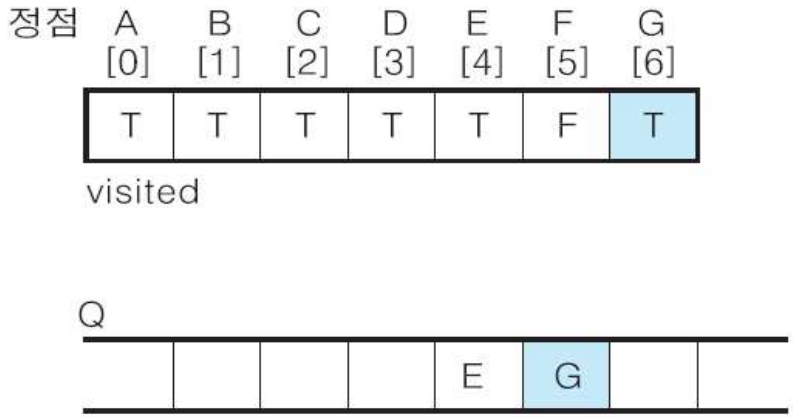


3.3 너비 우선 검색(BFS)

⑦ 정점 D의 방문 안한 인접정점 G를 방문하고 큐에 enQueue 한다.

visited[(D의 방문 안한 인접정점 G)]←true;
(D의 방문 안한 인접정점 G) 방문;
enQueue(Q, (D의 방문 안한 인접정점 G));

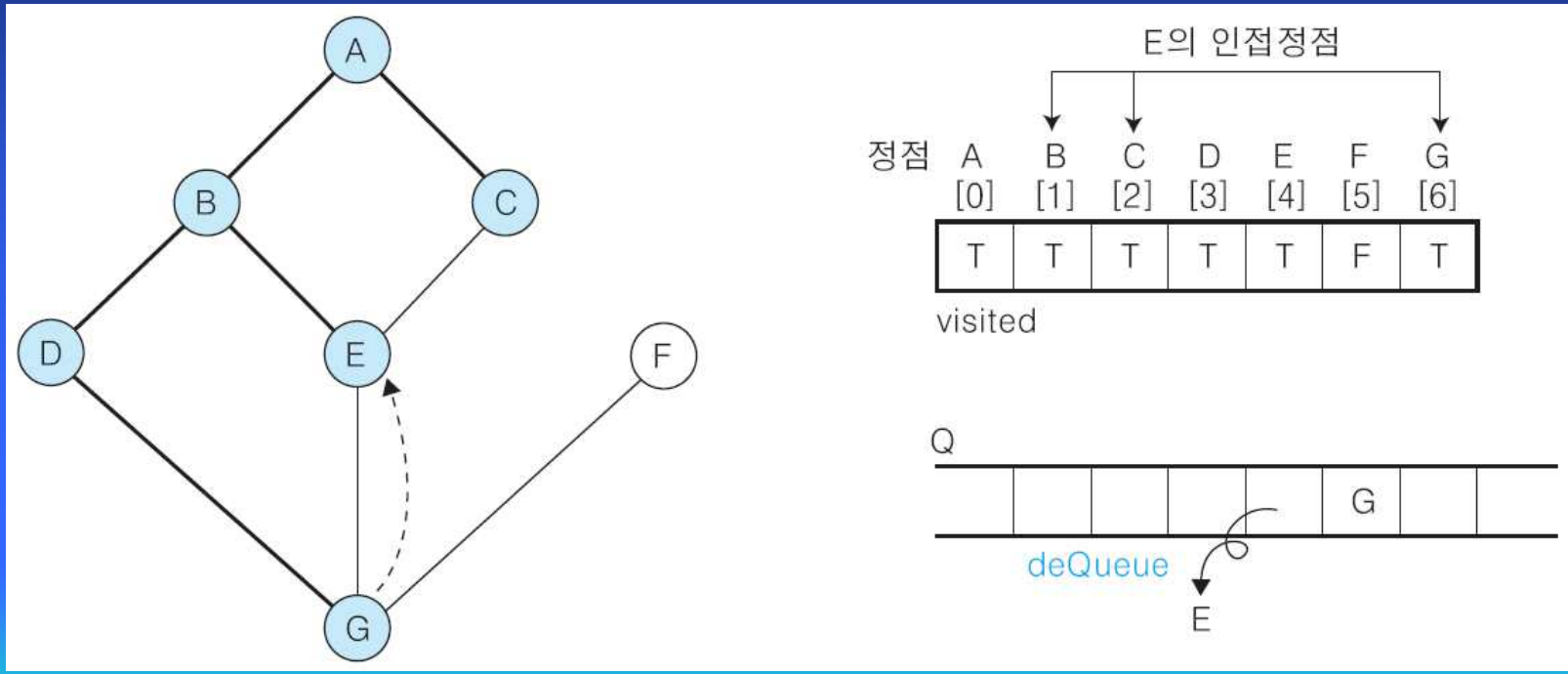




3.3 너비 우선 검색(BFS)

⑧ 정점 D에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 E를 구한다.

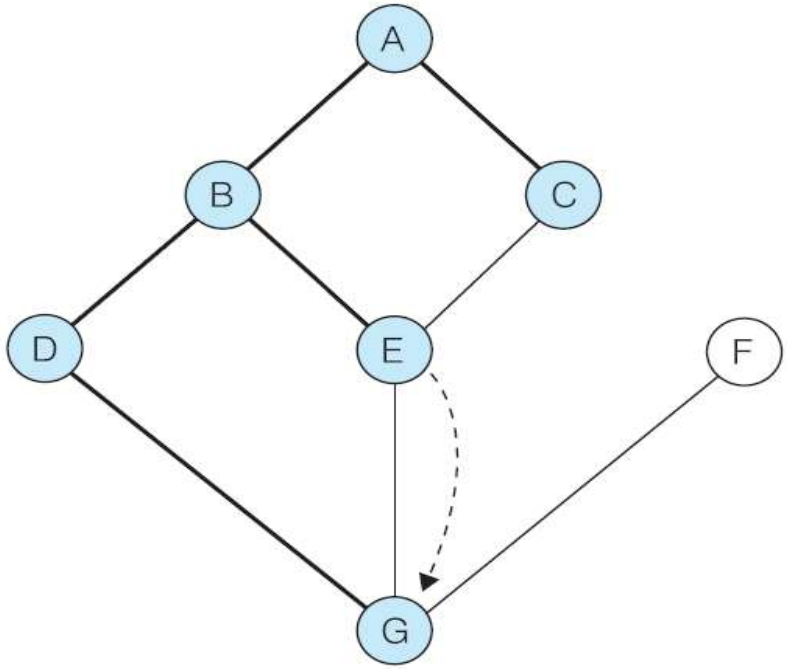
```
v ← deQueue(Q);
```

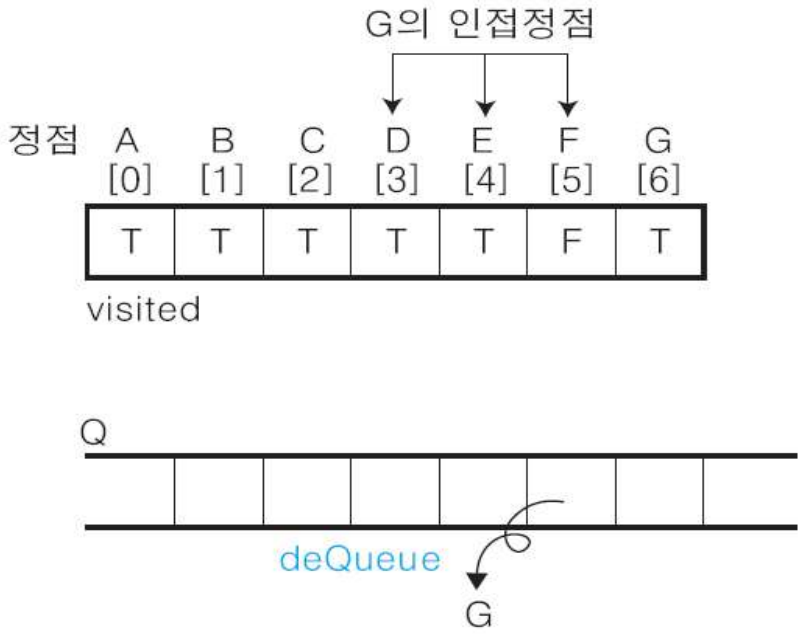


3.3 너비 우선 검색(BFS)

⑨ 정점 **E**에는 방문 안한 인접정점이 없으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 G를 구한다.

```
v ← deQueue(Q);
```

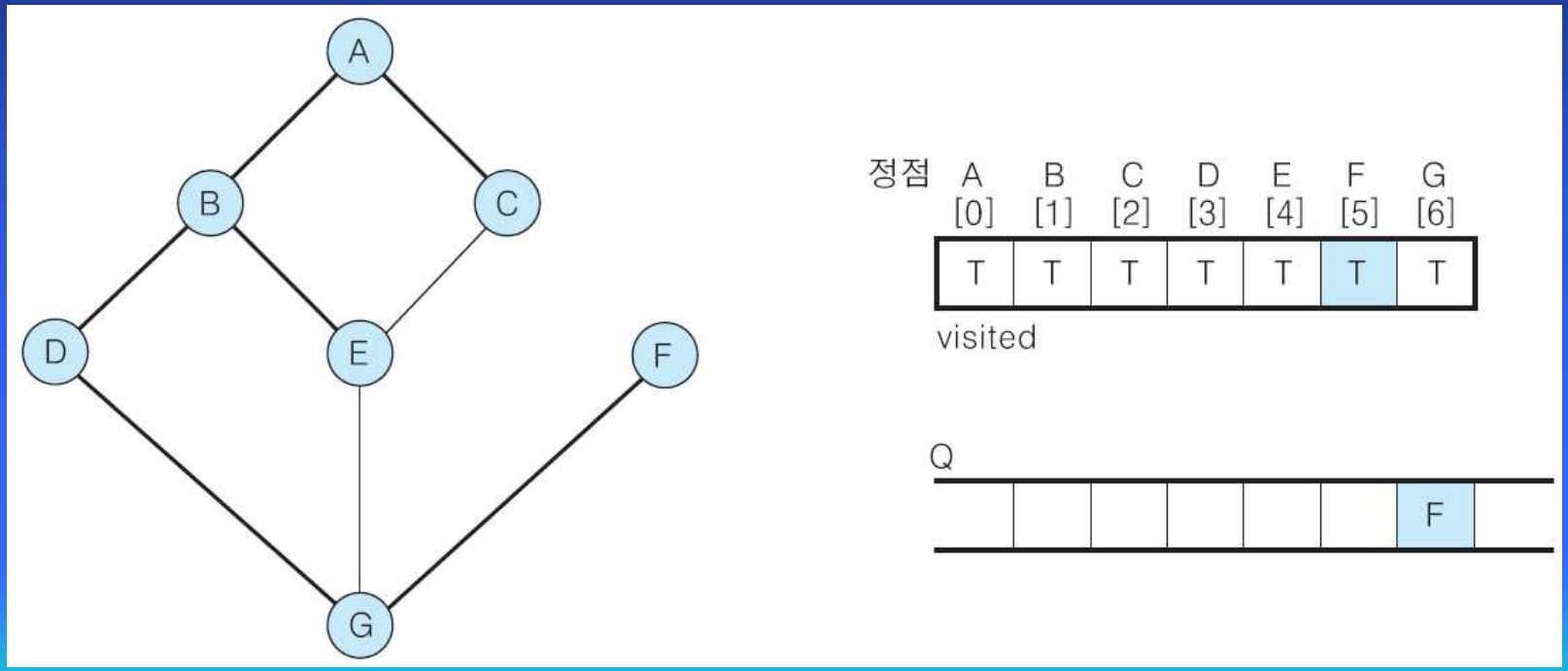




3.3 너비 우선 검색(BFS)

⑩ 정점 G의 방문 안한 인접정점 F를 방문하고 큐에 enQueue 한다.

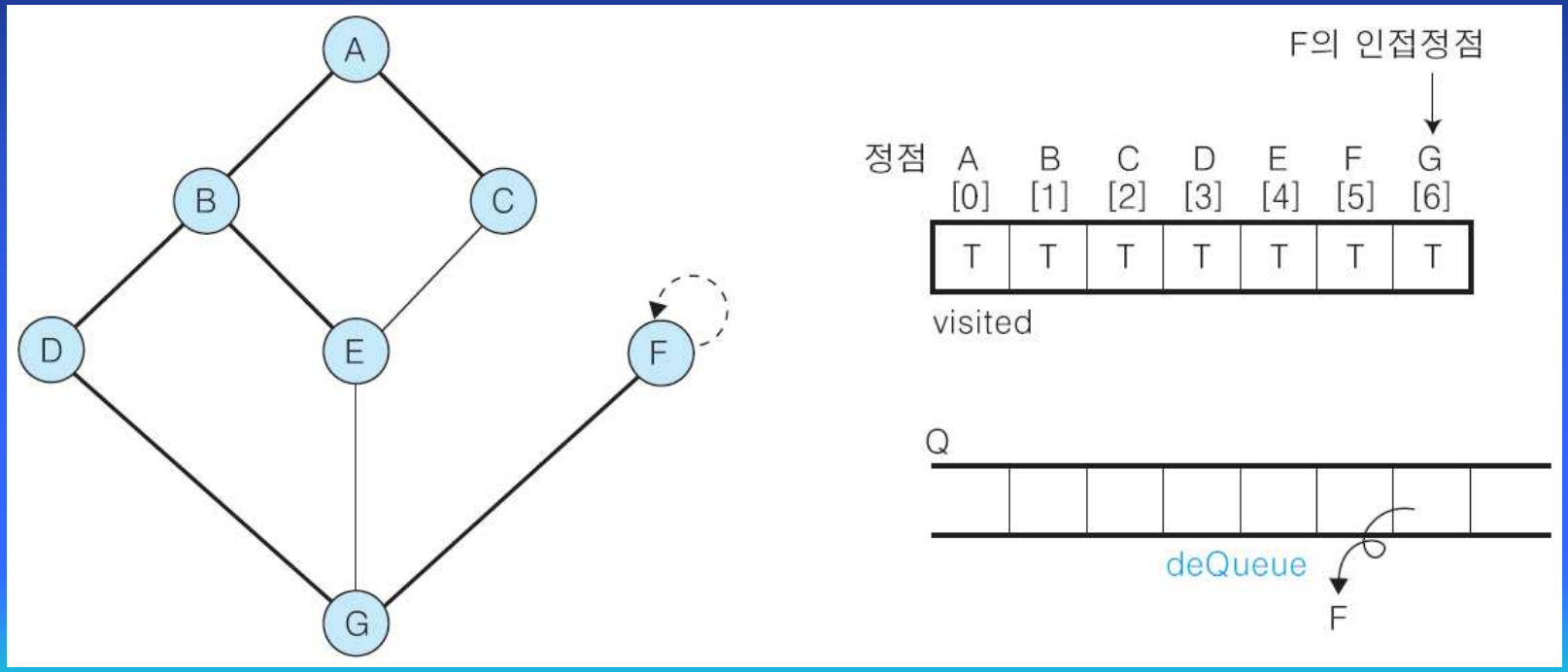
```
visited[(G의 방문 안한 인접정점 F)]←true;
 (G의 방문 안한 인접정점 F) 방문;
 enQueue(Q, (G의 방문 안한 인접정점 F));
```



3.3 너비 우선 검색(BFS)

⑪ 정점 G에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 F를 구한다 .

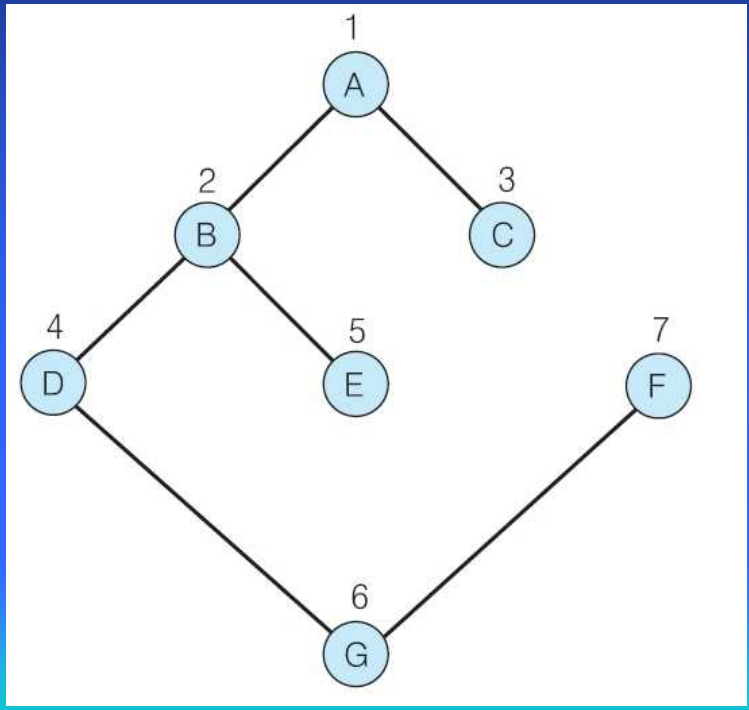
```
v ← deQueue(Q);
```



3.3 너비 우선 검색(BFS)

⑫ 정점 F에는 방문 안한 인접정점이 없으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하는데 큐가 공백이므로 너비 우선 탐색을 종료한다 .

☞그래프 G9의 너비 우선 탐색 경로 : A-B-C-D-E-G-F



3.3 너비 우선 검색(BFS)

- 그래프 G9를 너비 우선 탐색하는 프로그램
  - 그래프 **G9**를 인접 리스트로 표현한다.
  - 정점 **A~G** 대신에 **0~6**의 번호를 사용하여 연산하고, 출력할 때에는 **A~G** 문자로 바꾸어 표시한다.
  - 너비 우선 탐색을 위해서 큐 프로그램을 사용한다.
  - 실행 결과 **>**