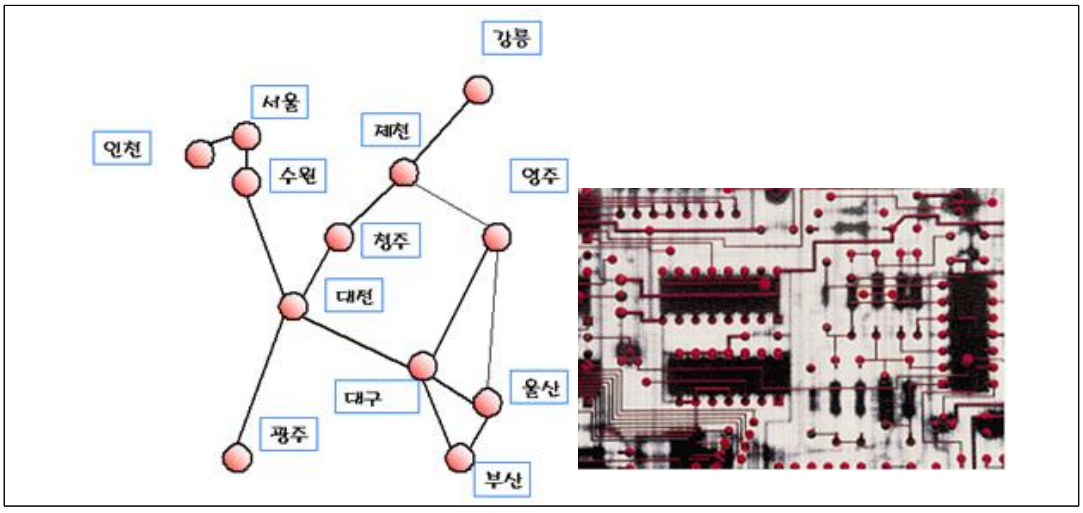
12주차 2차시 그래프 순회

【학습목표】

1. 깊이 우선 순회와 너비 우선 순회를 구분할 수 있다.
2. 그래프의 순회와 탐색을 설명할 수 있다.

학습내용1 : 그래프 순회, 그래프 탐색

* 가장 일반적인 자료구조 형태
- 전기회로의 소자 간 연결 상태
- 운영체제의 프로세스와 자원 관계
- 지도에서 도시들의 연결 상태 등

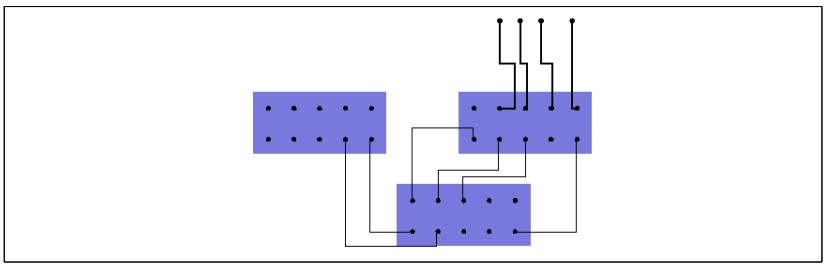


1. 그래프 순회란?

* 그래프의 가장 기본적인 연산,

* 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한 번씩 방문하여 처리하는 연산

* 많은 문제들이 단순히 그래프와 노드를 탐색하는 것으로 해결 가능
- 도로망에서 특정 도시에서 다른 도시로 갈 수 있는지 여부
- 전자회로에서 특정 단자와 다른 단자가 서로 연결되어 있는지 여부



* 그래프 탐색 방법
- 깊이 우선 탐색(Depth First Search : DFS)
- 너비 우선 탐색(Breadth First Search : BFS)

2. 그래프 순회 예제

* 한 지점을 골라서 팔 수 있을 까지 계속해서 깊에 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 않으며, 파놓은 구덩이들을 다시 좀 더 깊게 파는 방법
⇒ 너비 우선 탐색

학습내용2 : 깊이 우선 순회

1. 깊이 우선 탐색(Depth First Search : DFS)

* 순회 방법
- 시작 정점의 한 방향으로 갈 수 있는 경로가 있는 곳까지 깊이 탐색해 가다가 더 이상 갈 곳이 없게 되면, 가장 마지막에 만났던 갈림길 간선이 있는 정점으로 되돌아와서 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

* 깊이 우선 탐색의 수행 순서

(1) 시작 정점 v를 결정하여 방문한다.

(2) 정점 v에 인접한 정점 중에서

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

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

(3) 스택이 공백이 될 때까지 (2)를 반복한다.

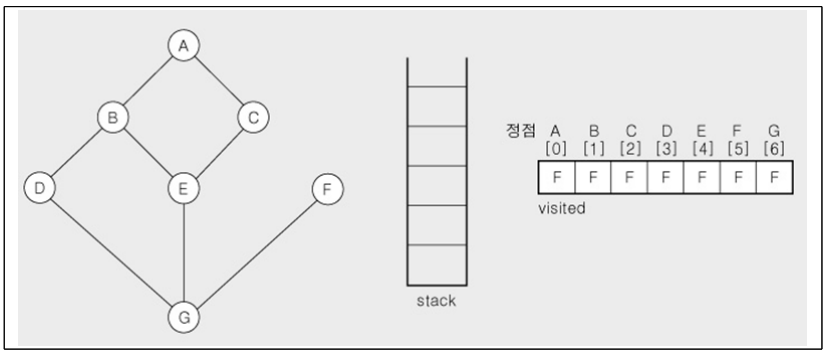
* 깊이 우선 탐색 알고리즘

알고리즘 9-1 깊이 우선 탐색 알고리즘

```
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()
```

2. 깊이 우선 탐색 예 - 그래프 G9에 대한 깊이 우선 탐색

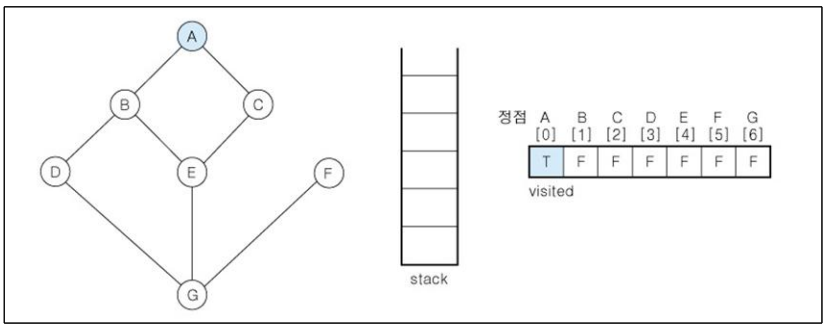
* 초기 상태 : 배열 visited를 False로 초기화하고, 공백 스택을 생성



① 정점 A를 시작으로 깊이 우선 탐색을 시작

visited[A] ←true;

A 방문

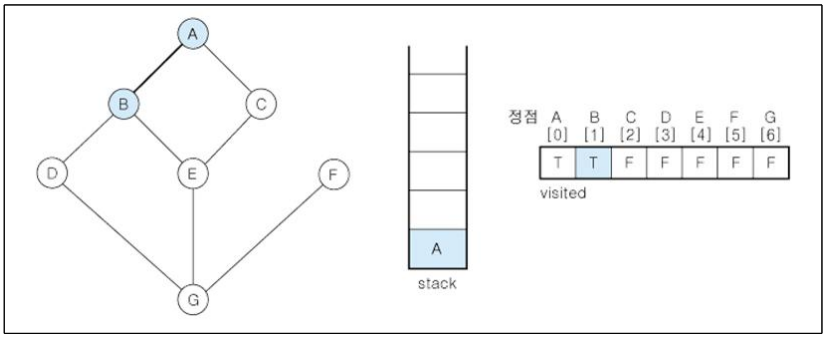


② 정점 A에 방문하지 않은 정점 B, C가 있으므로 A를 스택에 push 하고, 인접정점 B와 C 중에서 오름차순에 따라 B를 선택하여 탐색을 계속한다

push(stack, A);

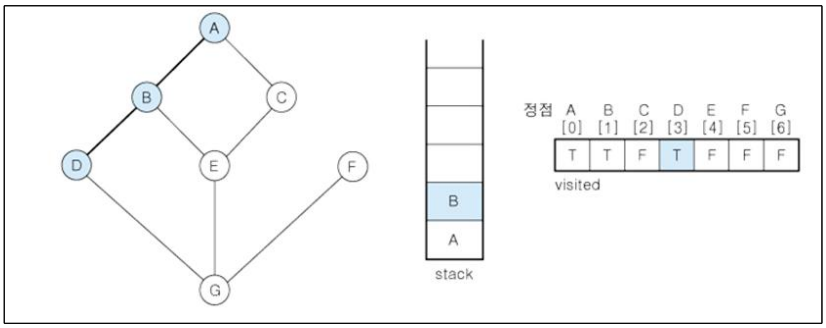
visited[B]←true;

B 방문;



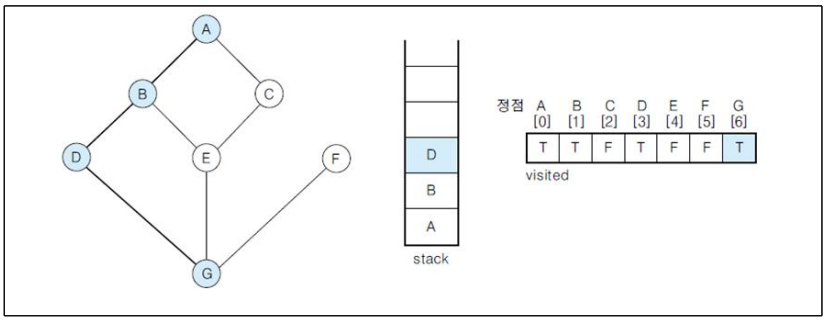
③ 정점 B에 방문하지 않은 정점 D, E가 있으므로 B를 스택에 push 하고, 인접정점 D와 E 중에서 오름차순에 따라 D를 선택하여 탐색을 계속한다.

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



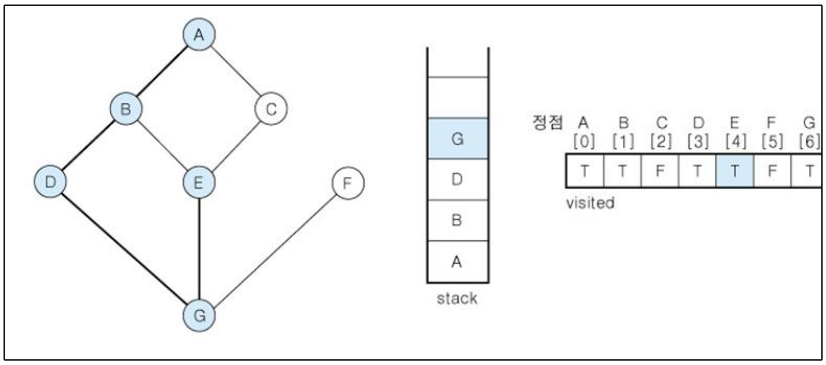
④ 정점 D에 방문하지 않은 정점 G가 있으므로 D를 스택에 push 하고, 인접정점 G를 선택하여 탐색을 계속한다.

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



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

push(stack, G);
visited[E]←true;
E 방문

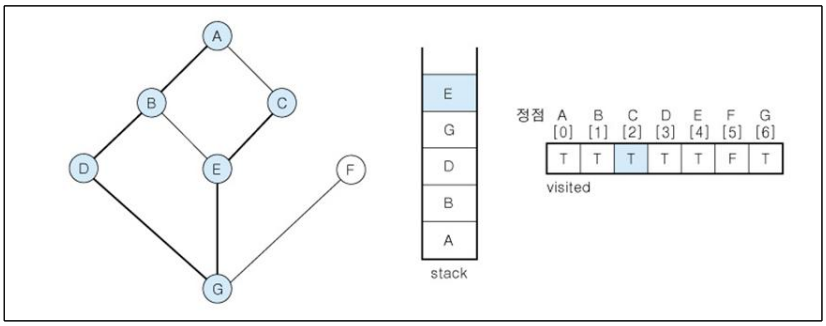


⑥ 정점 E에 방문하지 않은 정점 C가 있으므로 E를 스택에 push 하고, 인접정점 C를 선택하여 탐색을 계속한다.

push(stack, E);

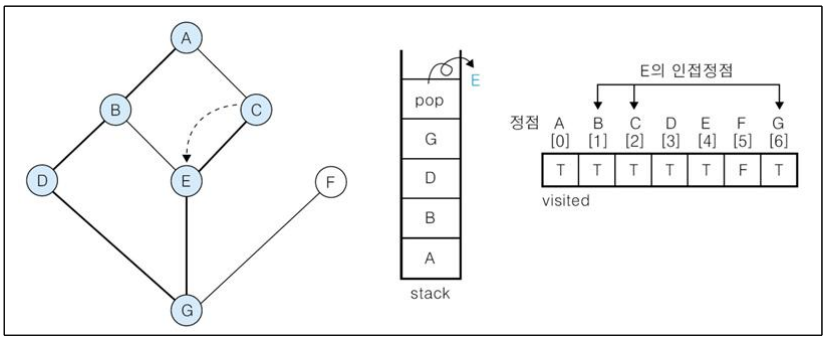
visited[C]←true;

C 방문



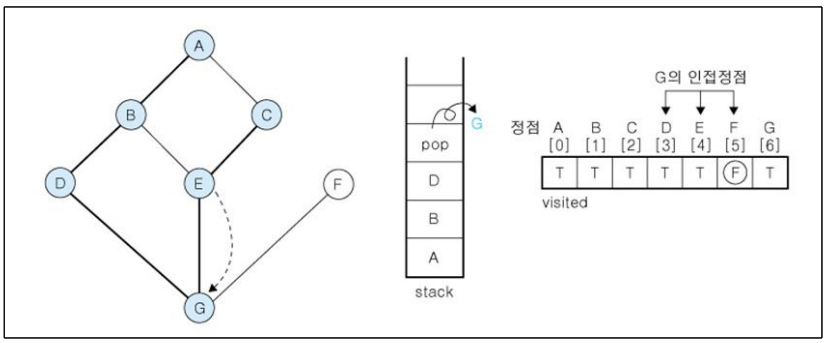
⑦ 정점 C에서 방문하지 않은 인접정점이 없으므로, 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 E에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

pop(stack)



⑧ 정점 E는 방문하지 않은 인접정점이 없으므로, 다시 스택을 pop 하여 받은 정점 G에 대해서 방문하지 않은 인접정점이 있는지 확인한다

pop(stack);

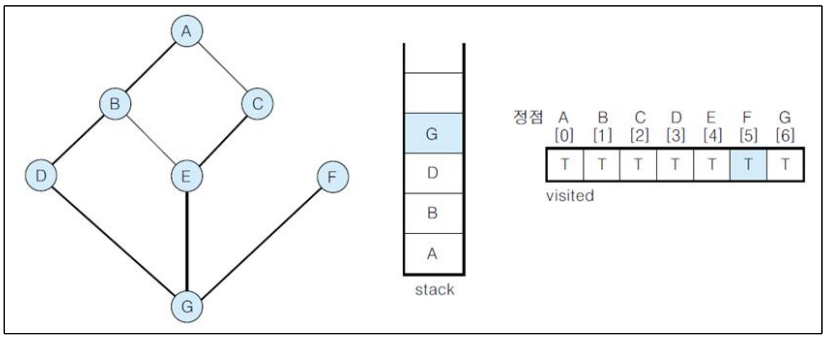


⑨ 정점 G에 방문하지 않은 정점 F가 있으므로 G를 스택에 push 하고, 인접정점 F를 선택하여 탐색을 계속한다.

push(stack, G);

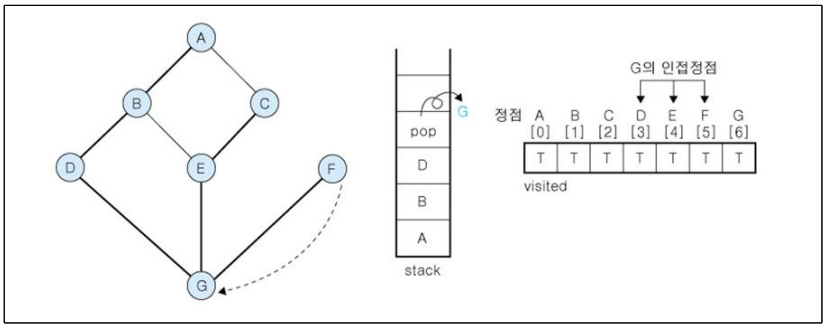
visited[F]←true;

F 방문



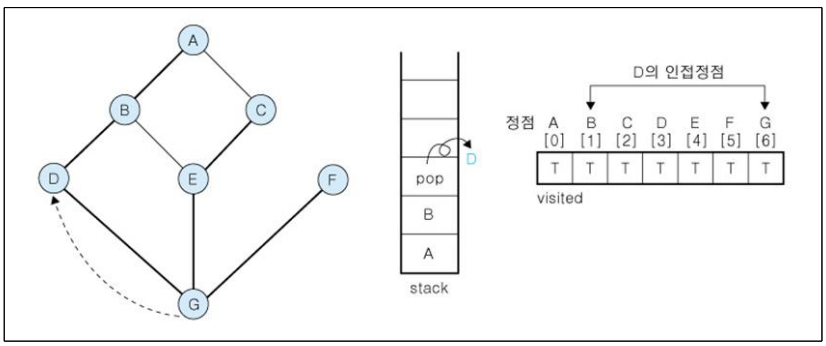
⑩ 정점 F에서 방문하지 않은 인접정점이 없으므로, 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 G에 대해서 방문하지 않은 인접정점이 있는지 확인한다

pop(stack);



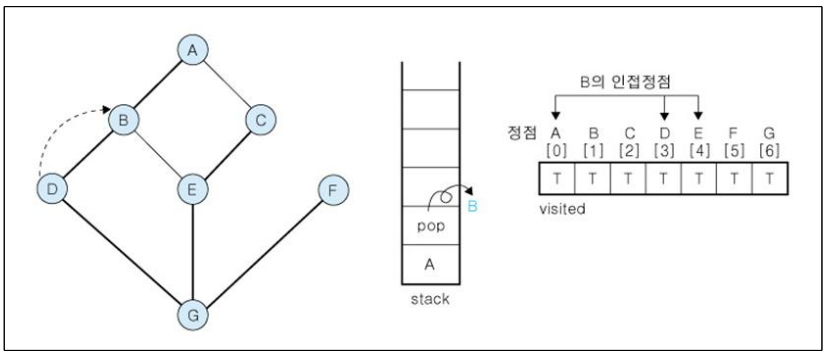
⑪ 정점 G에서 방문하지 않은 인접정점이 없으므로, 다시 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 D에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

pop(stack);



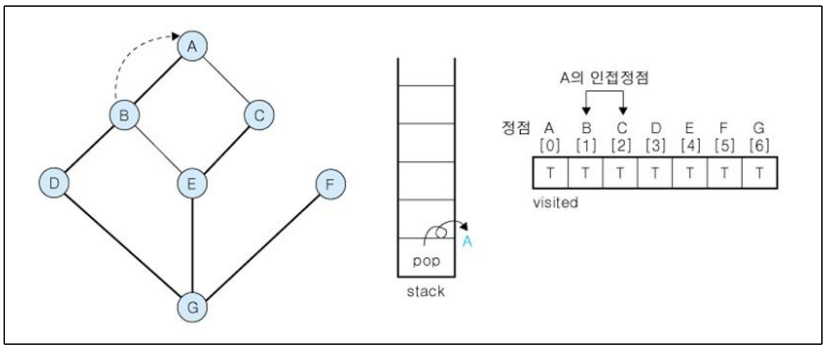
⑫ 정점 D에서 방문하지 않은 인접정점이 없으므로, 다시 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 B에 대해서 방문하지 않은 인접정점이 있는지 확인한다

pop(stack);

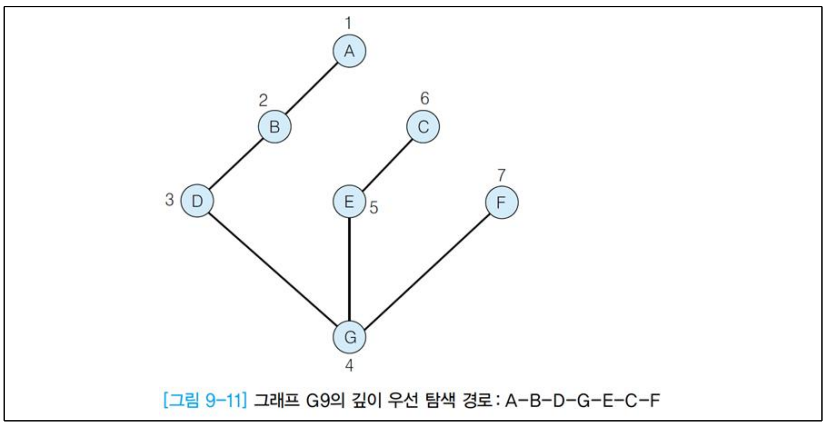


⑬ 정점 B에서 방문하지 않은 인접정점이 없으므로, 다시 마지막 정점으로 돌아가기 위해 스택을 pop 하여 받은 정점 A에 대해서 방문하지 않은 인접정점이 있는지 확인한다.

pop(stack);



⑭ 현재 정점 A에서 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하는데, 스택이 공백이므로 깊이 우선 탐색을 종료한다.



[그림 9-11] 그래프 G9의 깊이 우선 탐색 경로 : A-B-D-G-E-C-F

학습내용3 : 너비 우선 순회

1. 너비 우선 탐색(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

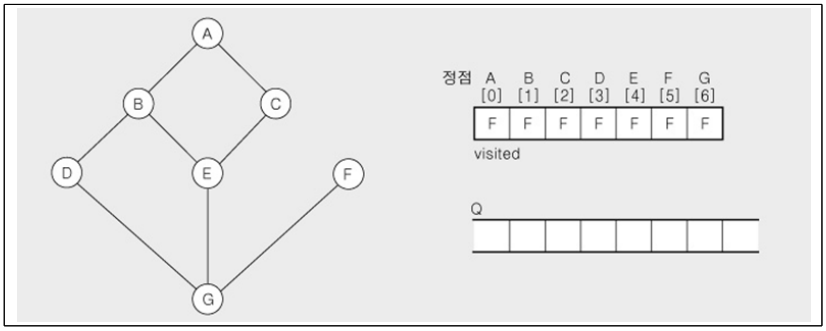
* 너비 우선 탐색 알고리즘

알고리즘 9-2 너비 우선 탐색 알고리즘

```
BFS(v)
    for (i←0; i<n; i←i+1) do {
        visited[i]←false;
    }
    Q←createQueue();
    visited[v]←true;
    v 방문;
    while (not isEmpty(Q)) do {
        while (visited[v의 인접 정점 w]=false) do {
            visited[w]←true;
            w 방문;
            enQueue(Q, w);
        }
        v←deQueue(Q);
    }
end BFS()
```

2. 너비 우선 탐색 예 - 그래프 G9에 대한 너비 우선 탐색

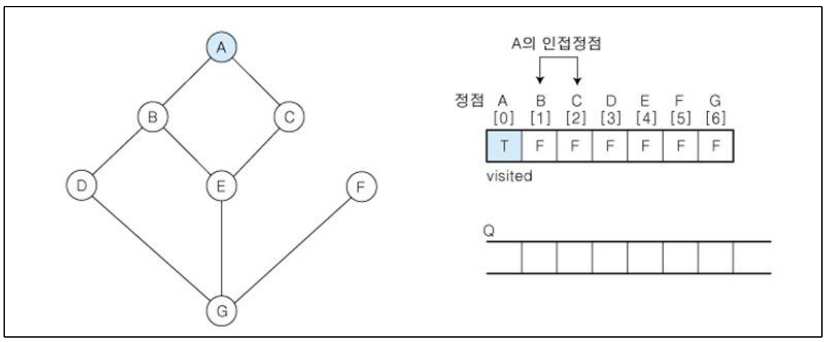
* 초기상태 : 배열 visited를 False로 초기화하고, 공백 큐를 생성



① 정점 A를 시작으로 너비 우선 탐색을 시작한다

visited[A]←true;

A 방문;

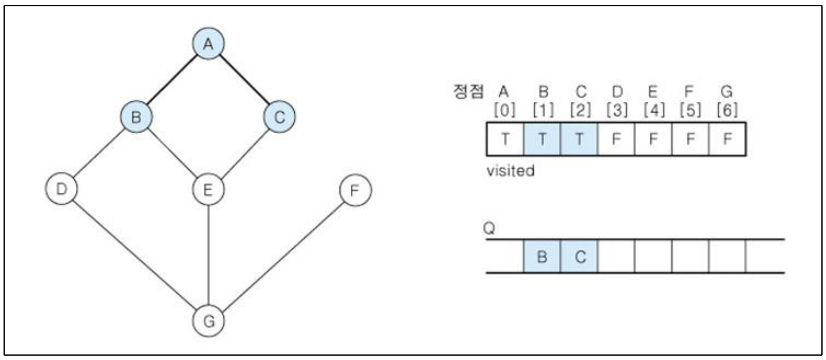


② 정점 A의 방문 안한 모든 인접정점 B, C를 방문하고, 큐에 enQueue 한다

visited[(A의 방문 안한 인접정점 B와 C)]←true;

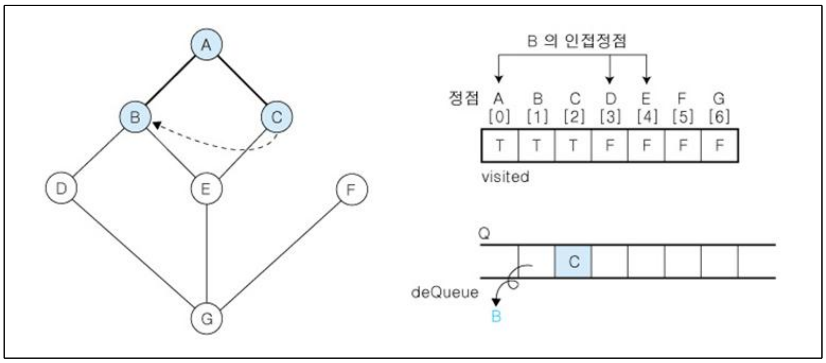
(A의 방문 안한 인접정점 B와 C) 방문

enQueue(Q, (A의 방문 안한 인접정점 B와 C));



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

v ← deQueue(Q);

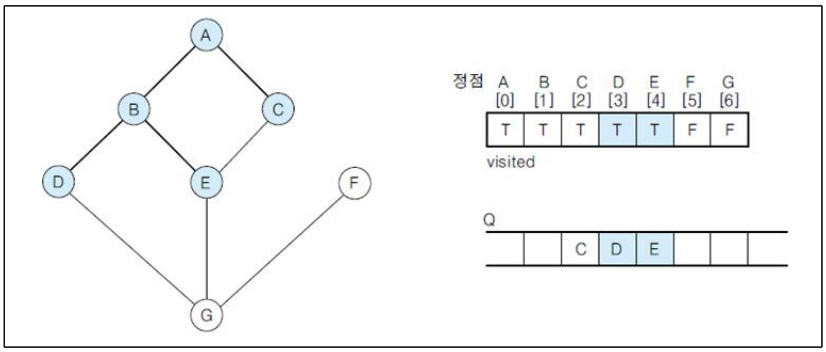


④ 정점 B의 방문 안한 모든 인접정점 D, E를 방문하고 큐에 enQueue 한다

visited[(B의 방문 안한 인접정점 D와 E)]←true;

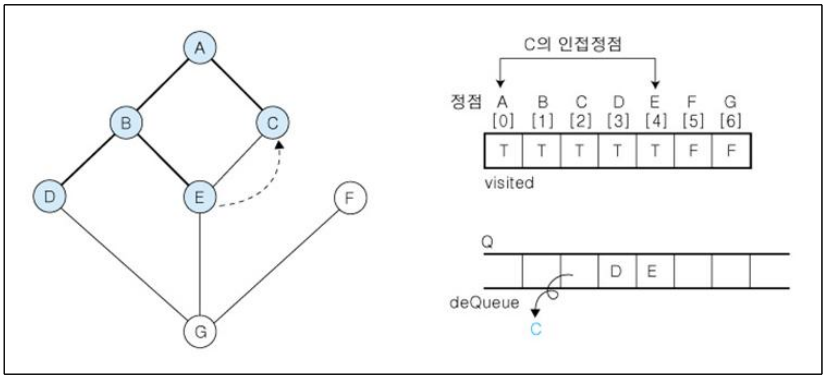
(B의 방문 안한 인접정점 D와 E) 방문

enQueue(Q, (B의 방문 안한 인접정점 D와 E));



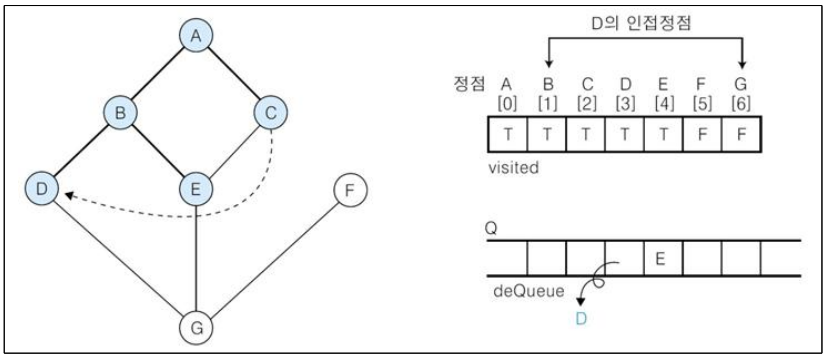
⑤ 정점 B에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 C를 구한다.

v ← deQueue(Q);



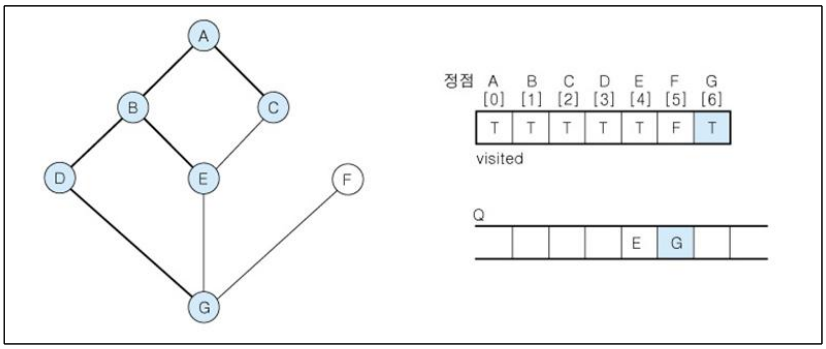
⑥ 정점 C에는 방문 안한 인접정점이 없으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 D를 구한다.

v ← deQueue(Q);



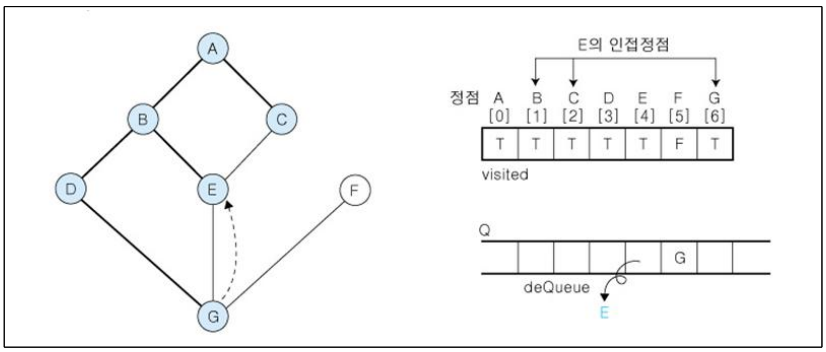
⑦ 정점 D의 방문 안한 인접정점 G를 방문하고 큐에 enQueue 한다.

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



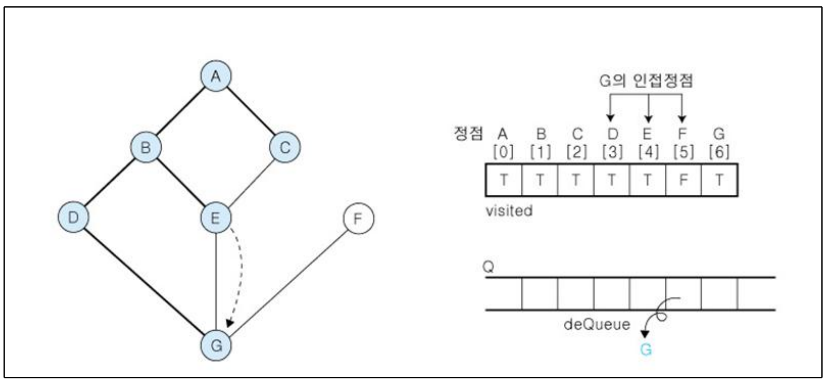
⑧ 정점 D에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 E를 구한다

v ← deQueue(Q);



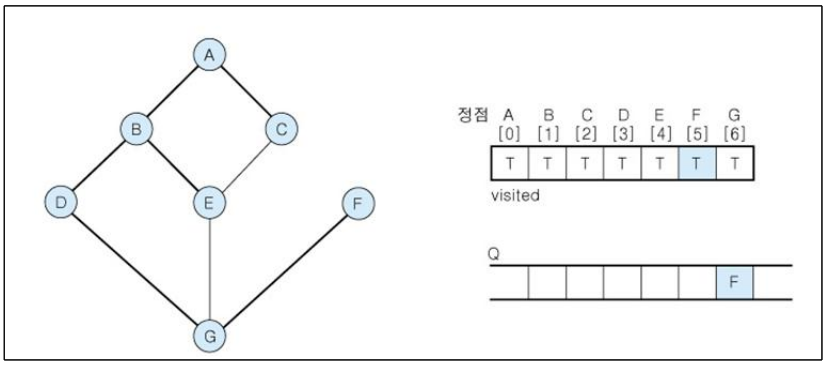
⑨ 정점 E에는 방문 안한 인접정점이 없으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 G를 구한다

v ← deQueue(Q);



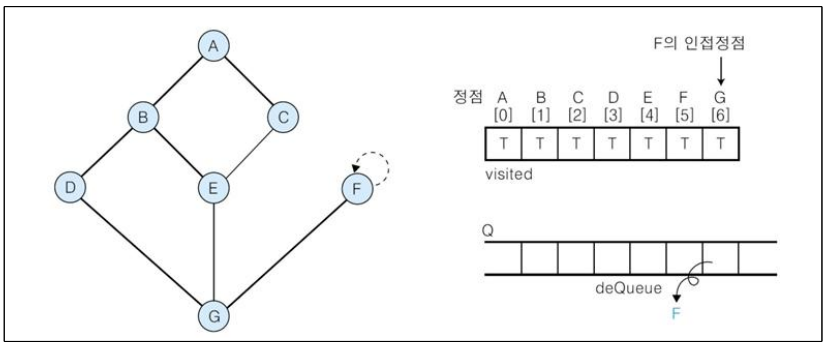
⑩ 정점 G의 방문 안한 인접정점 F를 방문하고 큐에 enQueue 한다.

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



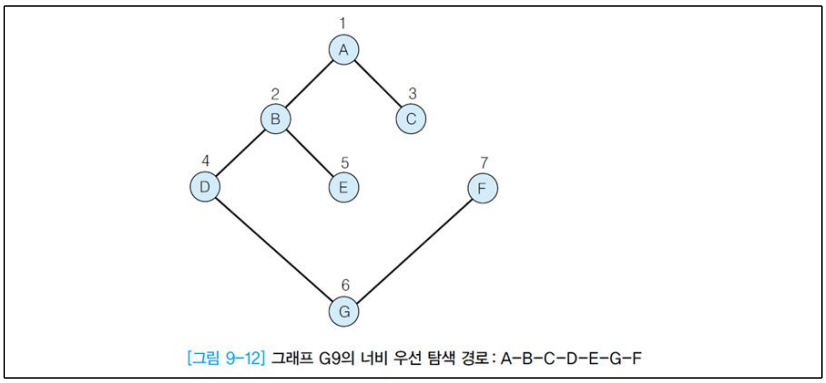
⑪ 정점 G에 대한 인접정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 정점 F를 구한다 .

v ← deQueue(Q);



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



[그림 9-12] 그래프 G9의 너비 우선 탐색 경로: A-B-C-D-E-G-F

【학습정리】

1. 깊이 우선 탐색은 시작 정점의 한 방향으로 갈 수 있는 경로가 있는 곳까지 깊이 탐색하다가 더 이상 갈 곳이 없으면 가장 마지막에 만났던 갈림길 간선이 있는 정점으로 되돌아와서 다른 방향의 간선으로 탐색을 계속하여 결국 모든 정점을 방문하는 순회방법이다.
- 깊이 우선 탐색은 스택을 사용한다.

2. 너비 우선 탐색은 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 방법이다.
- 너비 우선 탐색은 큐를 사용한다.