Department of Electrical and Electronics Engineering
Kangwon National University

KNU KANGWON NATIONAL UNIVERSITY

Data Structure:
그래프

Prof. Jungmin Kwon

Spring, 2026

학습 목표 및 내용

- 학습목표
  - 그래프 자료구조의 개념을 이해한다.
  - 그래프를 표현하는 방법을 알아본다.
  - 그래프를 순회하는 방법과 알고리즘을 알아본다.
  - 신장 트리와 최소 비용 신장 트리를 알아본다.
  - 최소 비용 신장 트리 알고리즘을 이해한다.

- 내용

  01 그래프의 구조

  02 그래프의 구현

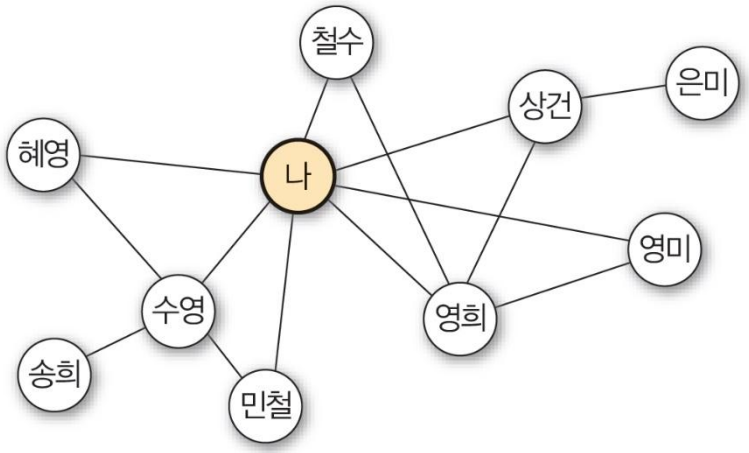
  03 그래프의 순회

  04 신장 트리와 최소 비용 신장 트리

1. 그래프의 구조

- 그래프graph
  - 연결되어 있는 원소 사이의 다:다 관계를 표현하는 자료구조
  - 그래프 G
    - 객체를 나타내는 정점vertex과 객체를 연결하는 간선edge의 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
  - 간선의 방향에 따른 분류: 무방향그래프, 방향그래프
  - 무방향 그래프undirected 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 그래프directed graph, 다이그래프digraph

- 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

- 연결형태에 따른 분류: 완전그래프, 부분그래프, 가중그래프
  - 완전 그래프complete 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 그래프subgraph

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

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

V(G) : Graph G 의 Vertex
E(G) : Graph G 의 Edge

- 그래프 G1에 대한 부분 그래프의 예

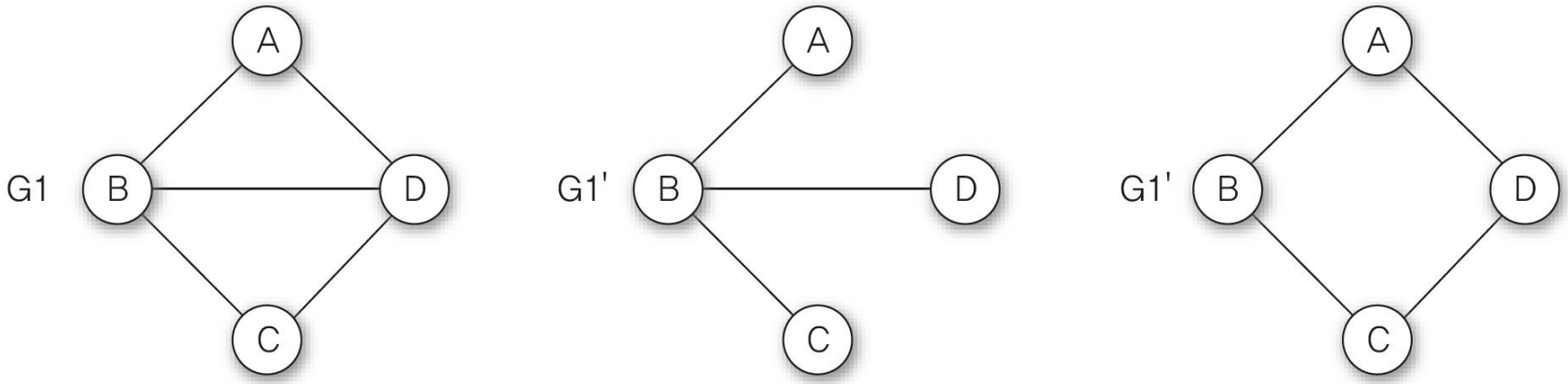


그림 8-6 그래프 G1과 부분 그래프 G1'의 예

1. 그래프의 구조

– 가중 그래프weight graph, 네트워크network
  - 정점을 연결하는 간선에 가중치weight를 할당한 그래프
  - 가중치는 연결에 대한 비용을 의미. 예: 이동 시간, 거리, 항공료

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 인접adjacent되어 있다고 하고, 간선 (V_i, V_j)는 정점 V_i와 V_j에 부속incident되어 있다고 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

- 차수degree – 정점에 부속되어있는 간선의 수
  - [그림 8-8]의 무방향 그래프 G1에서 정점 A의 차수는 2, 정점 B의 차수는 3
  - 방향 그래프의 정점의 차수 = 진입차수 + 진출차수
    - 방향 그래프의 **진입차수**(in-degree) : 정점을 머리로 하는 간선의 수
    - 방향 그래프의 **진출차수**(out-degree) : 정점을 꼬리로 하는 간선의 수
    - [그림 8-8]의 방향 그래프 G3에서 정점 B의 진입차수는 1, 진출차수는 2
      정점 B의 전체 차수는 (진입차수 + 진출차수) 이므로 3임
- 경로path
  - 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열한 것 즉, 정점 Vi에서 Vj까지 간선으로 연결된 정점을 순서대로 나열한 리스트
    - [그림 8-8]의 무방향 그래프 G1에서 정점 A에서 정점 C까지는 A-B-C 경로와 A-B-D-C 경로, A-D-C 경로, 그리고 A-D-B-C 경로가 있음
  - 경로길이path length - 경로를 구성하는 간선의 수

그림 8-8 그래프 용어 설명을 위한 예

1. 그래프의 구조

- 단순경로simple path
  - 모두 다른 정점으로 구성된 경로
    - 그래프 G1에서 정점 A에서 정점 C까지의 경로 A-B-C는 단순경로이고, 경로 A-B-D-A-B-C는 단순경로가 아님
- 사이클cycle
  - 단순경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로
    - 그래프 G1에서 단순경로 A-B-C-D-A와 그래프 G4에서 단순경로 A-B-A는 사이클이 됨
- DAGdirected acyclic graph
  - 방향 그래프이면서 사이클이 없는 그래프
- 연결 그래프connected graph
  - 서로 다른 모든 쌍의 정점들 사이에 경로가 있는 그래프 즉, 떨어져있는 정점이 없는 그래프
  - 그래프에서 두 정점 V_i에서 V_j까지의 경로가 있으면 정점 V_i와 V_j가 연결connected되었다고 함. 트리는 사이클이 없는 연결 그래프
  - 단절 그래프Disconnected Graph – 연결되지 않은 정점이 있는 그래프

1. 그래프의 구조

(a) 연결 그래프

(b) 단절 그래프

그림 8-9 연결 그래프와 단절 그래프의 예

- 그래프의 추상 자료형

ADT 8-1 그래프의 추상 자료형

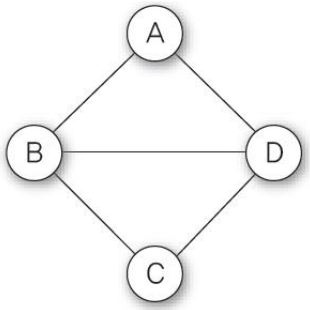
```
ADT Graph
데이터 : 공백이 아닌 정점의 집합과 간선의 집합
연산 :
   g∈Graph; u, v∈V;

   // 공백 그래프를 생성하는 연산
   createGraph() ::= create an empty Graph;

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

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

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

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

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

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

2. 그래프의 구현

- 그래프 구현 :
  - 정점의 집합과 간선의 집합을 표현해야 한다.
  - 2차원 배열을 이용하는 인접행렬 (Adjacent matrix) 방법과
  - 연결리스트를 이용하는 인접리스트(Adjacent list) 방법이 있다.
- 순차 자료구조를 이용한 그래프의 구현 : 인접 행렬
  - 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

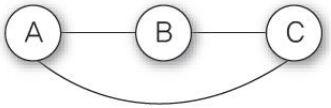
G1



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

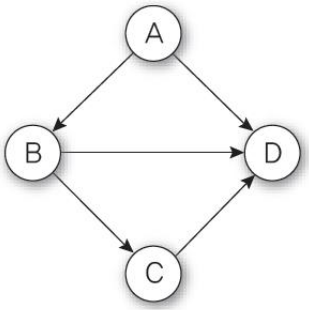
1+0+1+1=3 정점 B의 차수

G2



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

G3

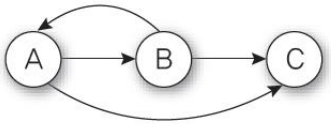


	A	B	C	D
A	0	1	0	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

0+0+1+1=2 정점 B의 진출 차수

1+0+0+0=1 정점 B의 진입 차수

G4



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

그림 8-10 그래프의 인접 행렬 표현 예

정점 자신을 가리키는 간선은 없음.

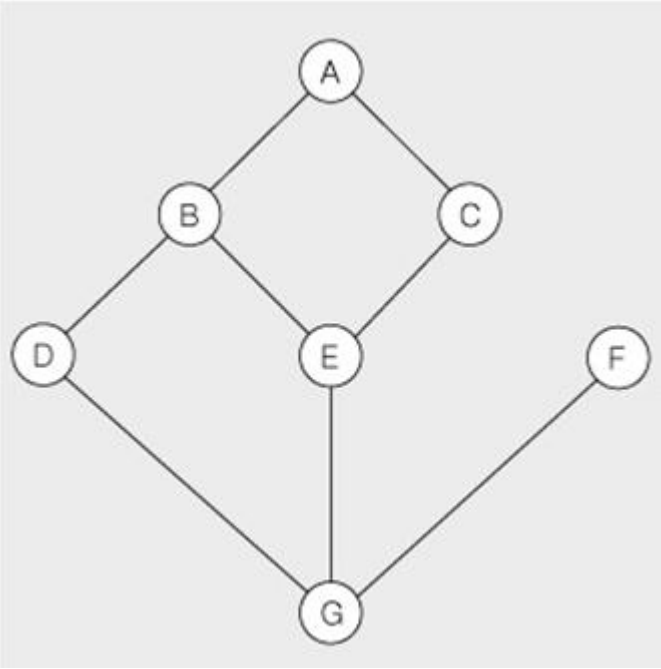
- 대각선 값은 항상 0 이다.

무방향 그래프에서,

- M → n 간선이 있으면
- N → m 간선도 있으므로
- 인접행렬은 대칭이다.

Self Test

- 다음 그래프를 인접행렬로 표현하시오.



2. 그래프의 구현

- [예제 8-1] 그래프를 인접 행렬로 표현하기 : 교재 434p
- 실행 결과

```
G1의 인접 행렬
        0 1 0 1
        1 0 1 1
        0 1 0 1
        1 1 1 0

G2의 인접 행렬
        0 1 1
        1 0 1
        1 1 0

G3의 인접 행렬
        0 1 0 1
        0 0 1 1
        0 0 0 1
        0 0 0 0

G4의 인접 행렬
        0 1 1
        1 0 1
        0 0 0
```

2. 그래프의 구현

예제 8-1 그래프를 인접 행렬로 표현하기

[adjMatrix.h]

```
#pragma once
#define MAX_VERTEX 30

// 그래프를 인접 행렬로 표현하기 위한 구조체 정의
typedef struct graphType {
    int n;                                  // 그래프의 정점 개수
    int adjMatrix[MAX_VERTEX][MAX_VERTEX]; // 그래프에 대한 30x30의 2차원 배열
} graphType;

void createGraph(graphType* g);
void insertVertex(graphType* g, int v);
void insertEdge(graphType* g, int u, int v);
void print_adjMatrix(graphType* g);
```

2. 그래프의 구현

[adjMatrix.c]

```c
#include <stdio.h>
#include <stdlib.h>
#include "adjMatrix.h"

// 공백 그래프를 생성하는 연산
void createGraph(graphType* g) {
    int i, j;
    g->n = 0;                          // 정점 개수를 0으로 초기화
    for (i = 0; i < MAX_VERTEX; i++) {
        for (j = 0; j < MAX_VERTEX; j++)
            g->adjMatrix[i][j] = 0; // 그래프 g에 대한 2차원 배열의 값을 0으로 초기화
    }
}

// 그래프 g에 정점 v를 삽입하는 연산
void insertVertex(graphType* g, int v) {
    if (((g->n) + 1) > MAX_VERTEX) {
        printf("\n 그래프 정점의 개수를 초과하였습니다!");
        return;
    }
    g->n++;                        // 그래프 정점의 개수 n을 하나 증가
}

```

2. 그래프의 구현

```
24  // 그래프 g에 간선 (u, v)를 삽입하는 연산
25  void insertEdge(graphType* g, int u, int v) {
26      // 간선 (u, v)를 삽입하기 위해 정점 u와 v가 그래프에 존재하는지 확인
27      if (u >= g->n || v >= g->n) {
28          printf("\n 그래프에 없는 정점입니다!");
29          return;
30      }
31      g->adjMatrix[u][v] = 1;     // 삽입한 간선에 대한 2차원 배열의 원소값을 1로 설정
32  }
33
34  // 그래프 g의 2차원 배열값을 순서대로 출력하는 연산
35  void print_adjMatrix(graphType* g) {
36      int i, j;
37      for (i = 0; i < (g->n); i++) {
38          printf("\n\t\t");
39          for (j = 0; j < (g->n); j++)
40              printf("%2d", g->adjMatrix[i][j]);
41      }
42  }
```

2. 그래프의 구현

[ex8_1.c]

```c
#include <stdio.h>  #include <stdlib.h>
#include "adjMatrix.h"

int main(void) {
    int i;
    graphType* G1 = (graphType*)malloc(sizeof(graphType));
    graphType* G2 = (graphType*)malloc(sizeof(graphType));
    graphType* G3 = (graphType*)malloc(sizeof(graphType));
    graphType* G4 = (graphType*)malloc(sizeof(graphType));

    // G1 구성 : 정점 u에 대한 간선 (u,v)는 v가 큰 것부터 삽입
    createGraph(G1);
    for (i = 0; i < 4; i++)
        insertVertex(G1, i);   // G1의 정점 0~3 삽입
    insertEdge(G1, 0, 3);
    insertEdge(G1, 0, 1);
    insertEdge(G1, 1, 3);
    insertEdge(G1, 1, 2);
    insertEdge(G1, 1, 0);
    insertEdge(G1, 2, 3);
    insertEdge(G1, 2, 1);
    insertEdge(G1, 3, 2);
    insertEdge(G1, 3, 1);
    insertEdge(G1, 3, 0);

```

2. 그래프의 구현

```
26      // G2 구성 : 정점 u에 대한 간선 (u,v)는 v가 큰 것부터 삽입
27      createGraph(G2);
28      for (i = 0; i < 3; i++)
29         insertVertex(G2, i);   // G2의 정점 0~2 삽입
30      insertEdge(G2, 0, 2);
31      insertEdge(G2, 0, 1);
32      insertEdge(G2, 1, 2);
33      insertEdge(G2, 1, 0);
34      insertEdge(G2, 2, 1);
35      insertEdge(G2, 2, 0);
36
37      // G3 구성 : 정점 u에 대한 간선 <u,v>는 v가 큰 것부터 삽입
38      createGraph(G3);
39      for (i = 0; i < 4; i++)
40         insertVertex(G3, i);   // G3의 정점 0~3 삽입
41      insertEdge(G3, 0, 3);
42      insertEdge(G3, 0, 1);
43      insertEdge(G3, 1, 3);
44      insertEdge(G3, 1, 2);
45      insertEdge(G3, 2, 3);
```

2. 그래프의 구현

```
46
47      // G4 구성 : 정점 u에 대한 간선 <u,v>는 v가 큰 것부터 삽입
48      createGraph(G4);
49      for (i = 0; i < 3; i++)
50         insertVertex(G4, i);   // G4의 정점 0~2 삽입
51      insertEdge(G4, 0, 2);
52      insertEdge(G4, 0, 1);
53      insertEdge(G4, 1, 2);
54      insertEdge(G4, 1, 0);
55
56      // 완성된 인접 행렬 출력
57      printf("\n G1의 인접 행렬");
58      print_adjMatrix(G1);
59
60      printf("\n\n G2의 인접 행렬");
61      print_adjMatrix(G2);
62
63      printf("\n\n G3의 인접 행렬");
64      print_adjMatrix(G3);
65
66      printf("\n\n G4의 인접 행렬");
67      print_adjMatrix(G4);
68
69      getchar();  return 0;
70   }
```

```
G1의 인접 행렬
        0 1 0 1
        1 0 1 1
        0 1 0 1
        1 1 1 0

G2의 인접 행렬
        0 1 1
        1 0 1
        1 1 0

G3의 인접 행렬
        0 1 0 1
        0 0 1 1
        0 0 0 1
        0 0 0 0

G4의 인접 행렬
        0 1 1
        1 0 1
        0 0 0
```

2. 그래프의 구현

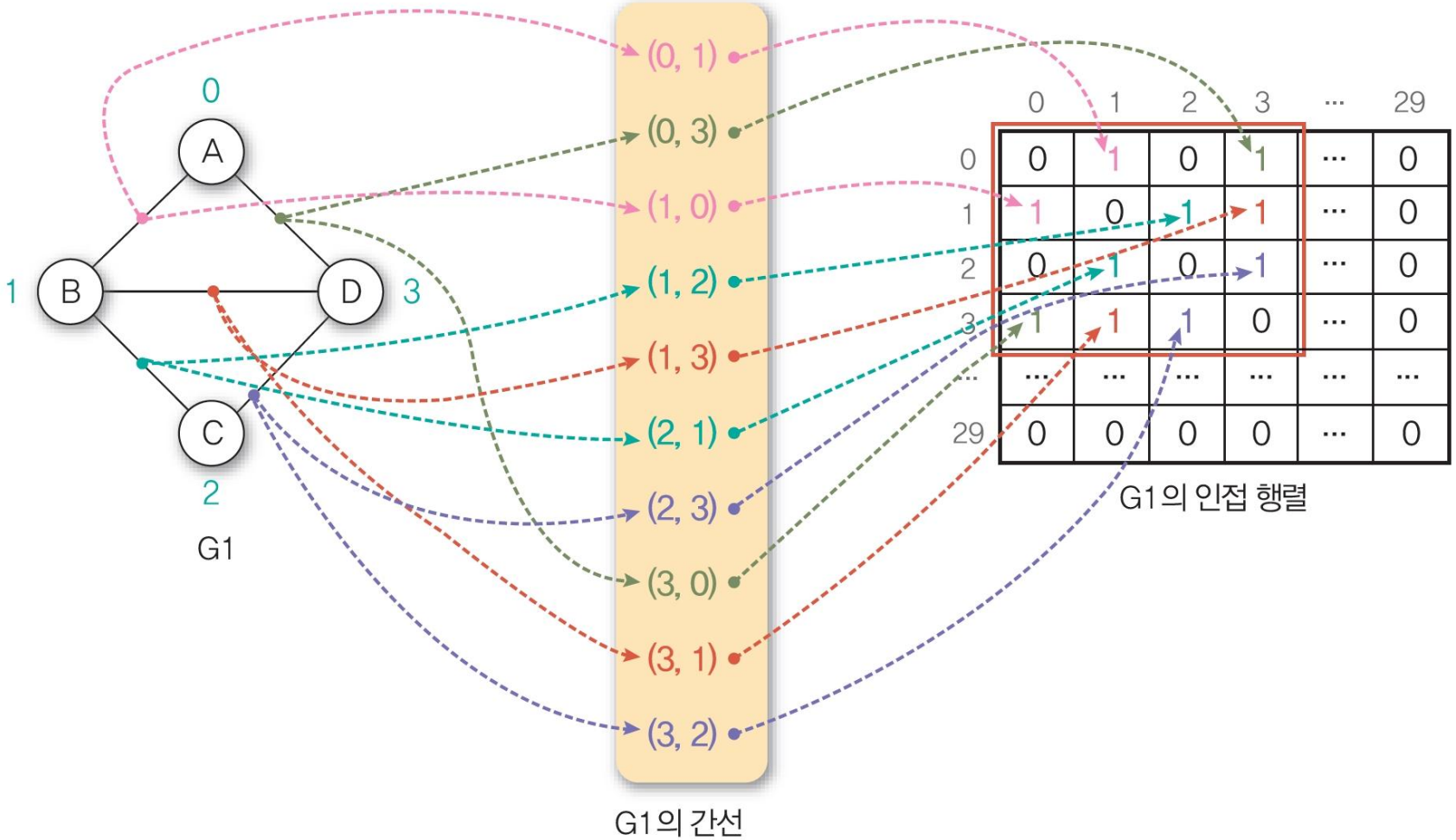
- **[ex8_1.c]**
- 12~54행은 [adjMatrix.h]에서 선언한 createGraph( ) 함수로 그래프 G1~G4에 대한 2차원 배열의 인접 행렬을 생성, insertVertex( )와 insertEdge( ) 함수로 정점과 간선을 삽입해 그래프 G1~G4에 대한 인접 행렬 완성
  - 13~14행 : 그래프 G1에 정점 네 개 삽입

	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

- 15~24행 : 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



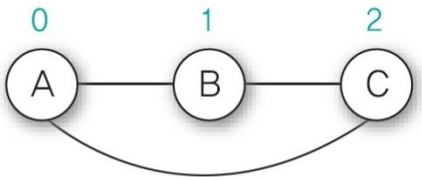
2. 그래프의 구현

- 28~29행 : 그래프 G2에 정점 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

- 30~35행 : 그래프 G2에 간선 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

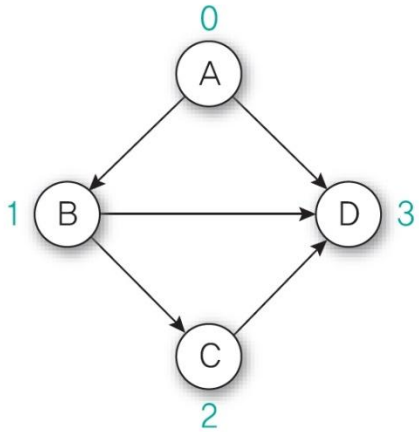
2. 그래프의 구현

- 39~40행 : 그래프 G3에 정점 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

- 41~45행 : 그래프 G3에 방향 간선 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

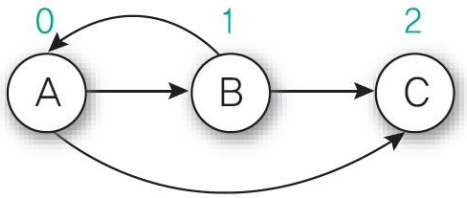
2. 그래프의 구현

- 49~50행 : 그래프 G4에 정점 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	···

G4

- 51~54행 : 그래프 G4에 방향 간선 네 개(<0,1>, <0,2>, <1,0>, <1,2>) 삽입



G4

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

G4

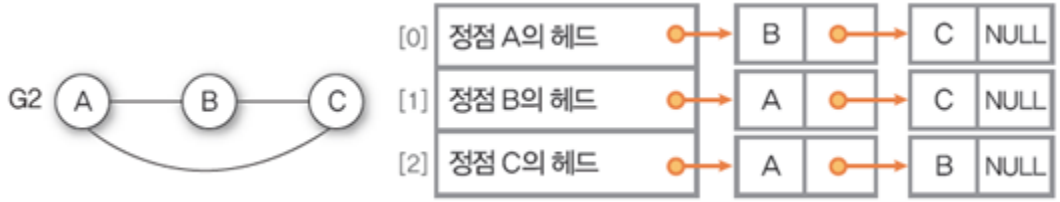
2. 그래프의 구현 - 과제

- 정점 u의 진입차수를 반환하는 함수 int inDegree(graphType* g, int u) 를 작성하고 예제 8-1의 main() 에서 테스트하시오.
  - 예제 8-1에서 G1, G2, G3, G4 그래프의 인접행렬을 출력하시오.
  - 그래프 G1~G4의 모든 정점들의 진입차수 출력하시오
  - 예시)

```
C:₩Users₩jmkwo₩Dropbox₩

G1의 인접 행렬
        0 1 0 1
        1 0 1 1
        0 1 0 1
        1 1 1 0

G2의 인접 행렬
        0 1 1
        1 0 1
        1 1 0

G3의 인접 행렬
        0 1 0 1
        0 0 1 1
        0 0 0 1
        0 0 0 0

G4의 인접 행렬
        0 1 1
        1 0 1
        0 0 0


 그래프 G1에서..
0 정점의 진입차수는 2 입니다.
1 정점의 진입차수는 3 입니다.
2 정점의 진입차수는 2 입니다.
3 정점의 진입차수는 3 입니다.
```

2. 그래프의 구현

- 연결 자료구조를 이용한 그래프의 구현 : 인접 리스트
  - 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트



  - 각 정점의 차수만큼 노드를 연결
    - 리스트 내의 노드들은 인접 정점에 대해서 오름차순으로 연결
  - 인접 리스트의 각 노드
    - 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성
    - 정점에 대한 인접 리스트의 헤드포인터는 정점 개수만큼 필요
    - 그래프는 정점의 집합이므로 각 정점에 대한 헤드 포인터를 그룹으로 묶어서 포인터 배열로 구성함

2. 그래프의 구현

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

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

2. 그래프의 구현

– 그래프 G1, G2, G3, G4에 대한 인접 리스트

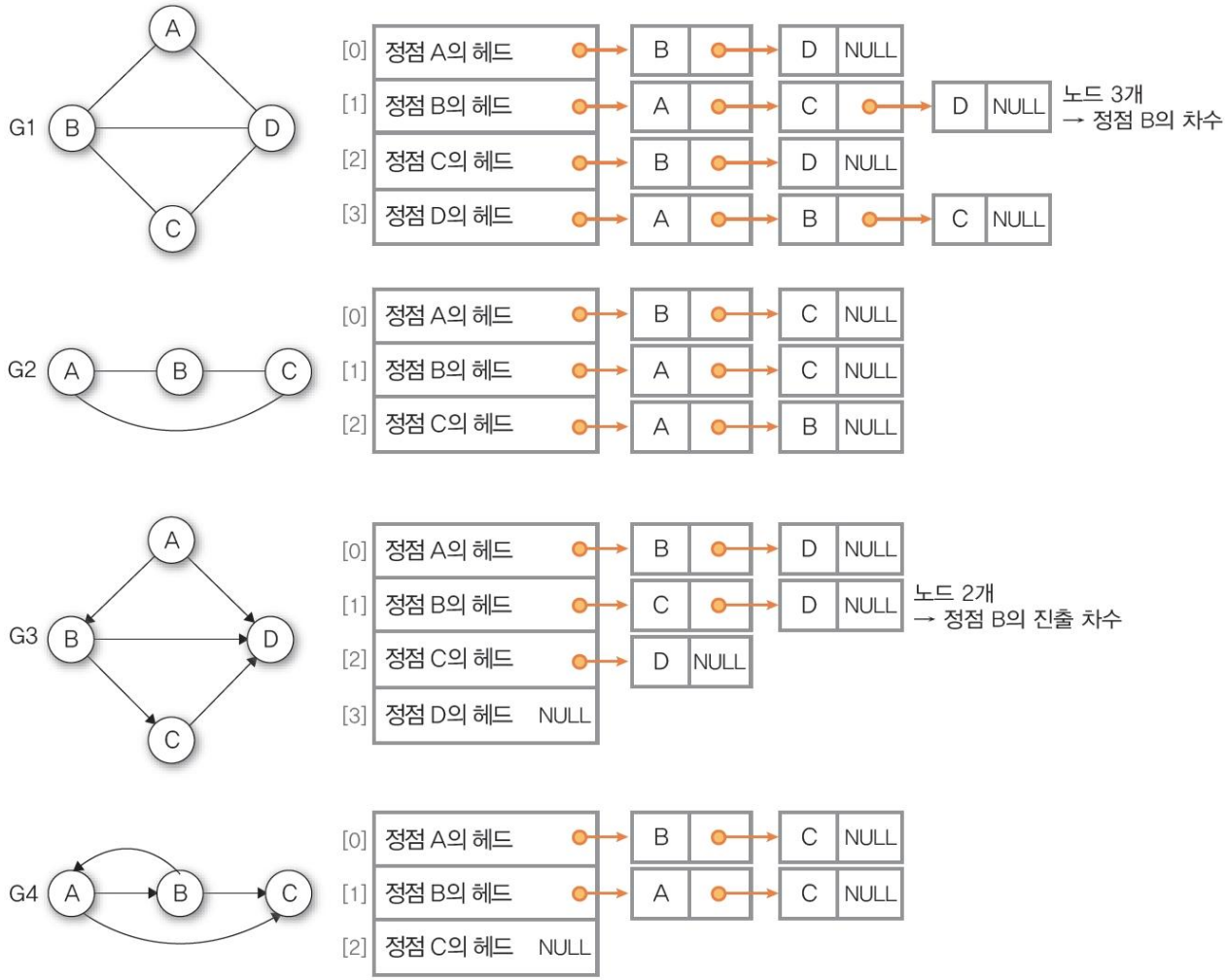
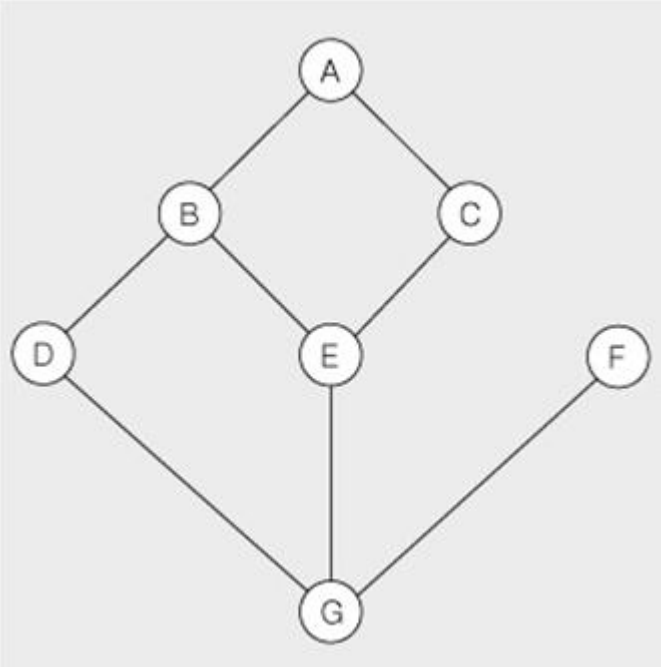


그림 8-11 그래프의 인접 리스트 표현 예

Self Test

- 다음 그래프를 인접리스트로 표현하시오.



2. 그래프의 구현

- [예제 8-2] 인접 리스트로 그래프 구현하기 : 교재 441p
- 실행 결과

```
G1의 인접 리스트
        정점 A의 인접 리스트 -> B -> D
        정점 B의 인접 리스트 -> A -> C -> D
        정점 C의 인접 리스트 -> B -> D
        정점 D의 인접 리스트 -> A -> B -> C

G2의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> C
        정점 C의 인접 리스트 -> A -> B

G3의 인접 리스트
        정점 A의 인접 리스트 -> B -> D
        정점 B의 인접 리스트 -> C -> D
        정점 C의 인접 리스트 -> D
        정점 D의 인접 리스트

G4의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> C
        정점 C의 인접 리스트
```

2. 그래프의 구현

예제 8-2 인접 리스트로 그래프 구현하기

[adjList.h]

```
#pragma once
#define MAX_VERTEX 30                    // 헤드 포인터 배열의 최대 크기

// 인접 리스트의 노드 구조를 구조체로 정의
typedef struct graphNode {
    int vertex;                          // 정점을 나타내는 데이터 필드
    struct graphNode* link;              // 다음 인접 정점을 연결하는 링크 필드
} graphNode;

// 그래프를 인접 리스트로 표현하기 위한 구조체 정의
typedef struct graphType {
    int n;                               // 그래프의 정점 개수
    graphNode* adjList_H[MAX_VERTEX];    // 그래프 정점에 대한 헤드 포인터 배열
} graphType;

void createGraph(graphType* g);
void insertVertex(graphType* g, int v);
void insertEdge(graphType* g, int u, int v);
void print_adjList(graphType* g);
```

2. 그래프의 구현

[adjList.c]

```
#include <stdio.h>
#include "adjList.h"

// 공백 그래프를 생성하는 연산
void createGraph(graphType* g) {
    int v;
    g->n = 0;                          // 그래프의 정점 개수를 0으로 초기화
    for (v = 0; v < MAX_VERTEX; v++)
        g->adjList_H[v] = NULL;        // 그래프의 정점에 대한 헤드 포인터 배열을 NULL로 초기화
}

// 그래프 g에 정점 v를 삽입하는 연산
void insertVertex(graphType* g, int v) {
    if (((g->n) + 1) > MAX_VERTEX) {
        printf("\n 그래프 정점의 개수를 초과하였습니다!");
        return;
    }
    g->n++;                            // 그래프의 정점 개수 n을 하나 증가
}

```

#include <stdlib.h>

2. 그래프의 구현

```c
// 그래프 g에 간선 (u, v)를 삽입하는 연산
void insertEdge(graphType* g, int u, int v) {
    graphNode* node;

    // 간선 (u, v)를 삽입하기 위해 정점 u와 정점 v가 현재 그래프에 있는지 확인
    if (u >= g->n || v >= g->n) {
        printf("\n 그래프에 없는 정점입니다!");
        return;
    }
    node = (graphNode*)malloc(sizeof(graphNode));
    node->vertex = v;
    node->link = g->adjList_H[u]; // 삽입 간선에 대한 노드를 리스트의 첫 번째 노드로 연결
    g->adjList_H[u] = node;
}

// 그래프 g의 각 정점에 대한 인접 리스트를 출력하는 연산
void print_adjList(graphType* g) {
    int i;
    graphNode* p;
    for (i = 0; i < g->n; i++) {
        printf("\n\t\t정점 %c의 인접 리스트", i + 65);
        p = g->adjList_H[i];
        while (p) {
            printf(" -> %c", p->vertex + 65); // 정점 0~3을 A~D로 출력
            p = p->link;
        }
    }
}
```

2. 그래프의 구현

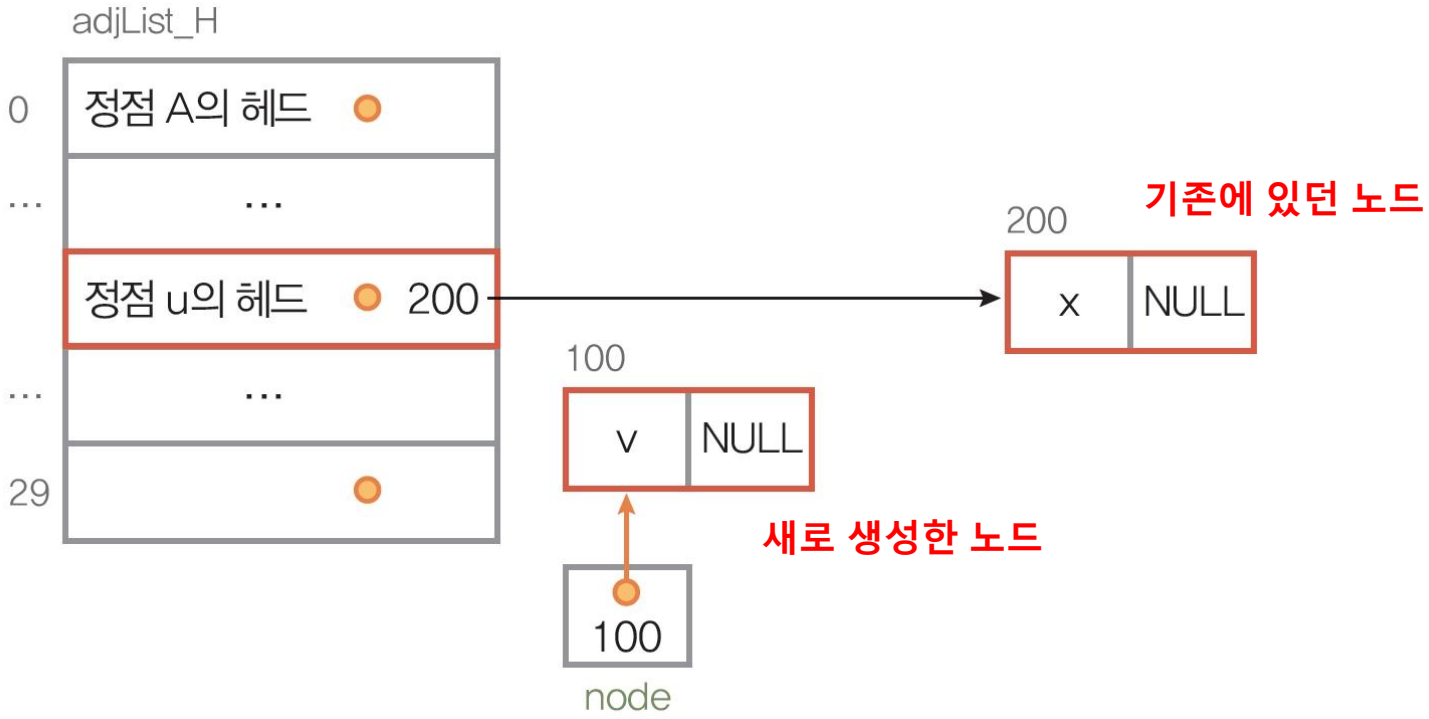
[ex8_2.c]

```c
#include <stdio.h>
#include "adjList.h"

int main(void) {
   // [예제 8-1]의 [ex8_1.c]의 05 ~ 54행과 동일

    // 완성된 인접 리스트 출력
    printf("\n G1의 인접 리스트");
    print_adjList(G1);

    printf("\n\n G2의 인접 리스트");
    print_adjList(G2);

    printf("\n\n G3의 인접 리스트");
    print_adjList(G3);

    printf("\n\n G4의 인접 리스트");
    print_adjList(G4);

    getchar();  return 0;
}
```

```
G1의 인접 리스트
        정점 A의 인접 리스트 -> B -> D
        정점 B의 인접 리스트 -> A -> C -> D
        정점 C의 인접 리스트 -> B -> D
        정점 D의 인접 리스트 -> A -> B -> C

G2의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> C
        정점 C의 인접 리스트 -> A -> B

G3의 인접 리스트
        정점 A의 인접 리스트 -> B -> D
        정점 B의 인접 리스트 -> C -> D
        정점 C의 인접 리스트 -> D
        정점 D의 인접 리스트

G4의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> C
        정점 C의 인접 리스트
```

2. 그래프의 구현

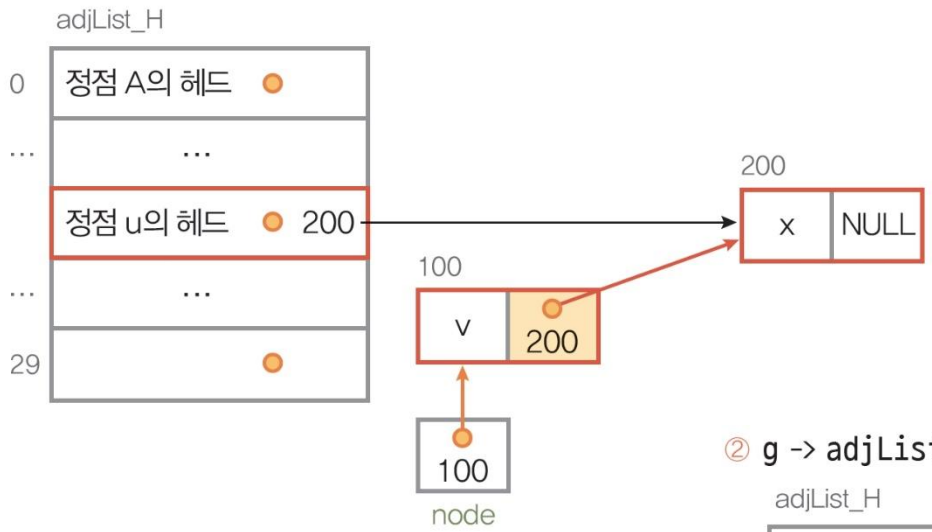
- **[adjList.c]**
- 22~34행의 insertEdge()는 그래프 g에 새로운 간선 (u, v)를 삽입하는 연산 수행
  - 31행 : 다음과 같이 새 노드의 데이터 필드 vertex에 인접 정점 v 저장



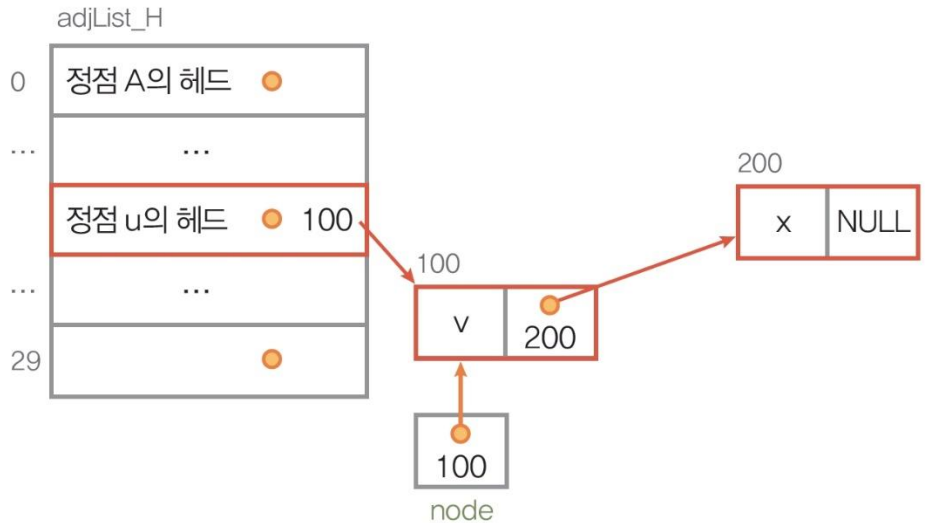
2. 그래프의 구현

- 32~33행 : 새 노드를 정점 u의 인접 리스트에 첫 번째 노드로 연결

① `node -> link = g -> adjList_H[u];`



② `g -> adjList_H[u] = node;`

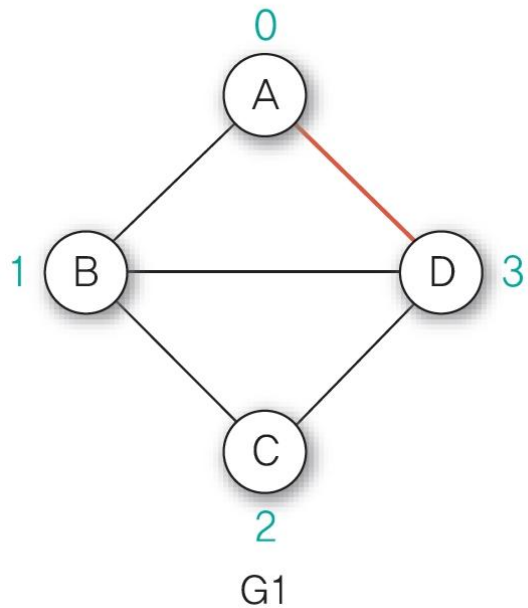


2. 그래프의 구현

- **[ex8_2.c]**
- 05~54행은 [예제 8-1]의 [ex8_1.c]의 05~54행을 그대로 사용하여 그래프 G1~G4 구성.
- 다만, [ex8_2.c]의 graphType/createGraph()/insertVertex()/insertEdge() 함수는 [adjList.h]에 선언된 인접 리스트 함수를 사용하므로 실제 구현되는 것은 인접 리스트 그래프임.
  - 12~24행 : 그래프 G1 생성, 정점과 간선을 삽입하여 G1 인접 리스트 구성
  - 13~14행 : 그래프 G1에 정점 네 개 삽입

2. 그래프의 구현

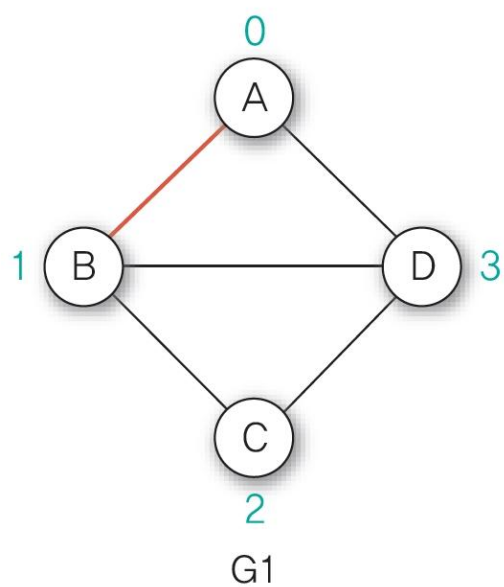
- 15행 : 그래프 G1에 간선 (0, 3) 삽입

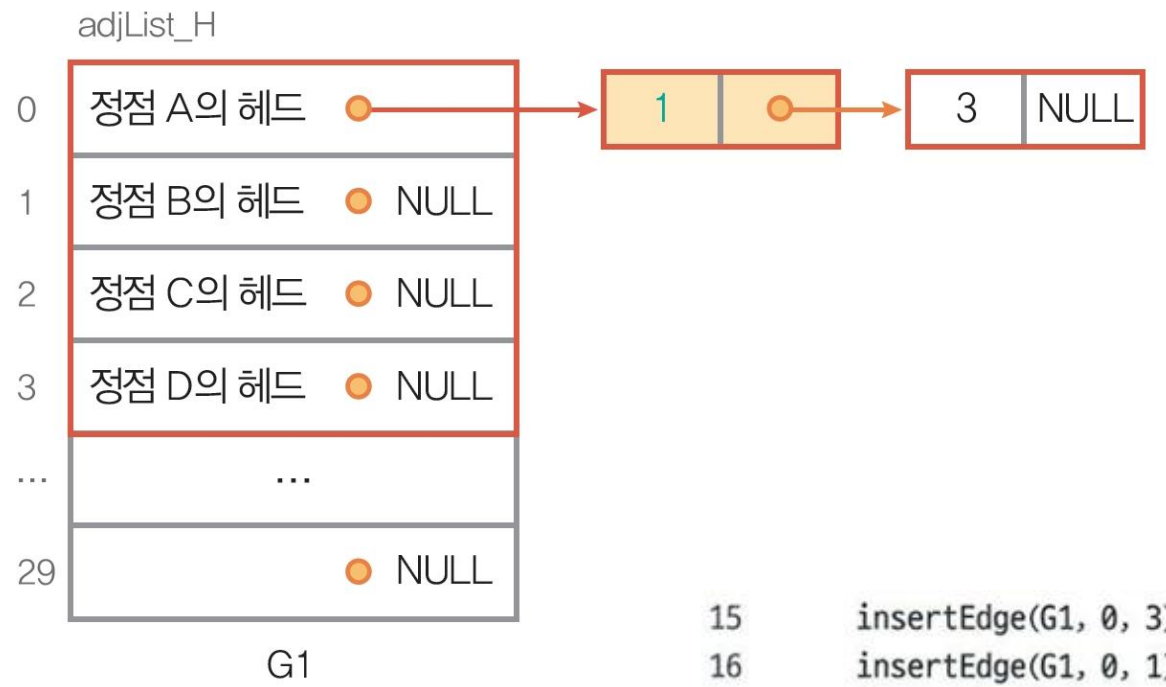


```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 16행 : 그래프 G1에 간선 (0, 1) 삽입

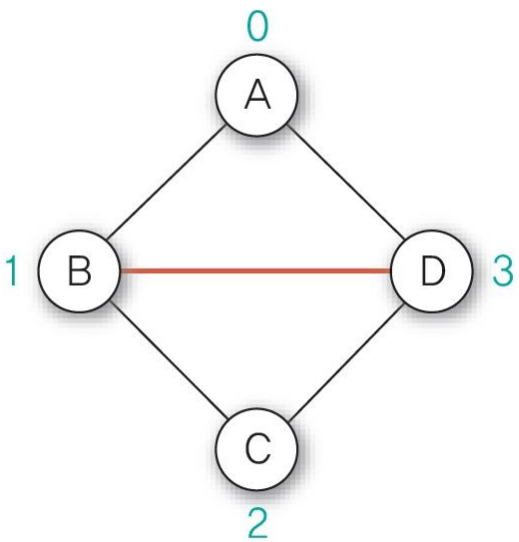




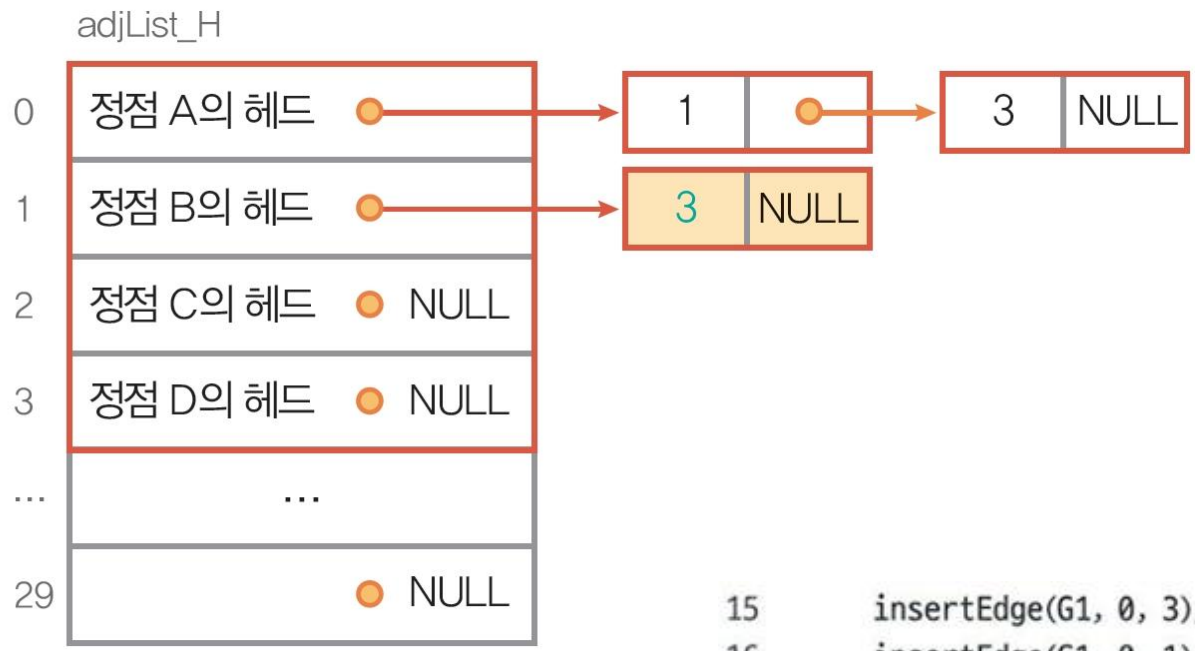

```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 17행 : 그래프 G1에 간선 (1, 3) 삽입



G1

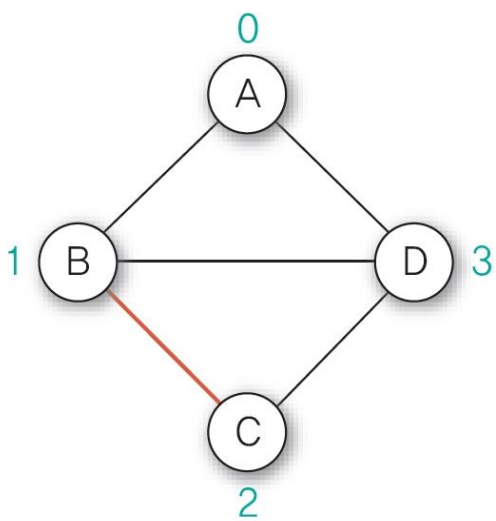


G1

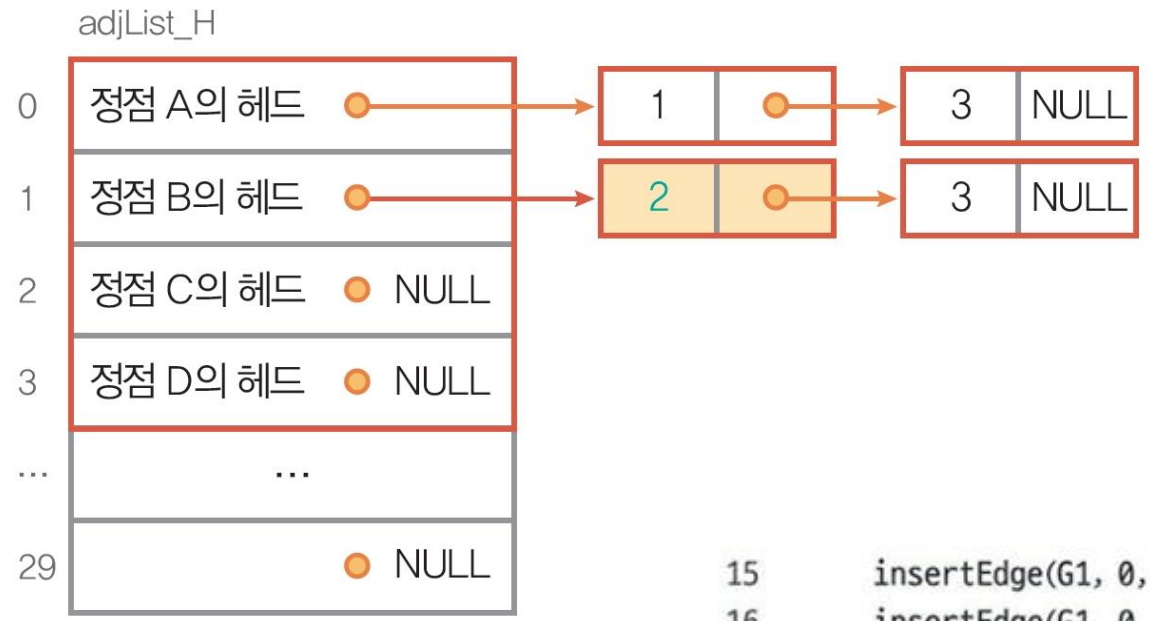
```
15    insertEdge(G1, 0, 3);
16    insertEdge(G1, 0, 1);
17    insertEdge(G1, 1, 3);
18    insertEdge(G1, 1, 2);
19    insertEdge(G1, 1, 0);
20    insertEdge(G1, 2, 3);
21    insertEdge(G1, 2, 1);
22    insertEdge(G1, 3, 2);
23    insertEdge(G1, 3, 1);
24    insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 18행 : 그래프 G1에 간선 (1, 2) 삽입



G1

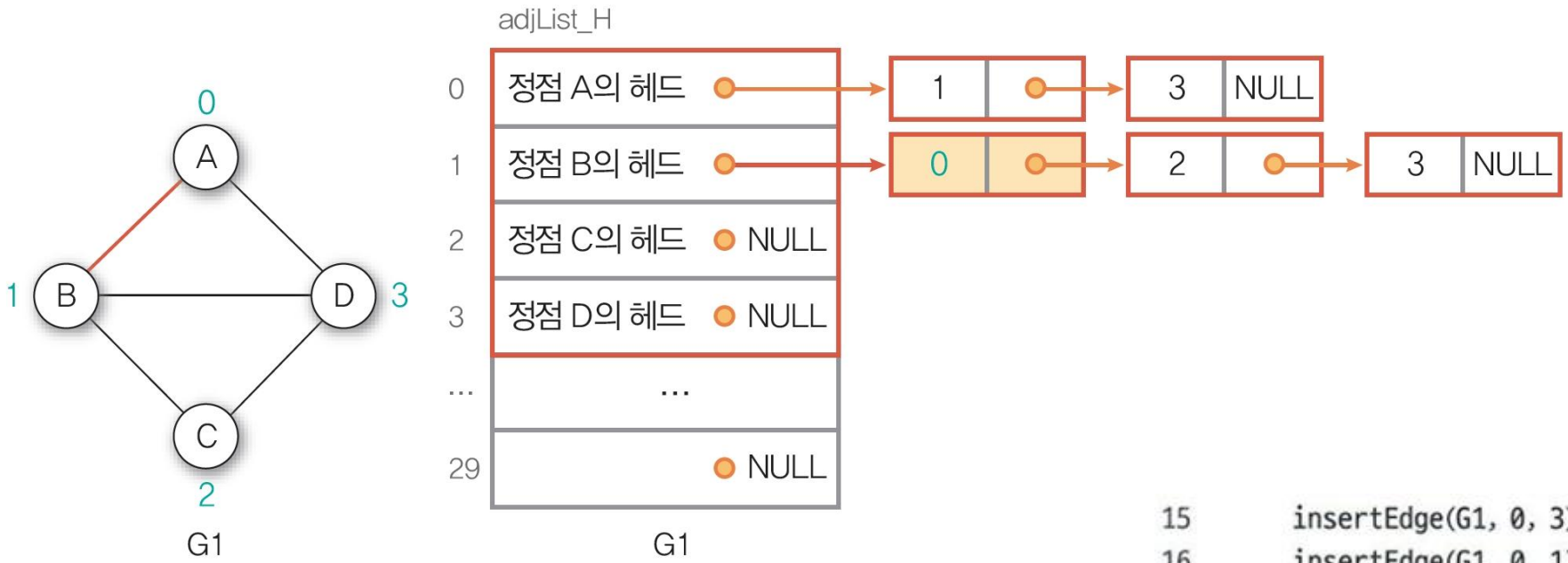


G1

```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

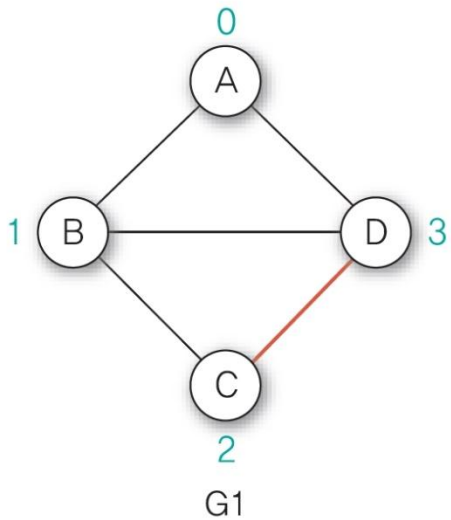
- 19행 : 그래프 G1에 간선 (1, 0) 삽입

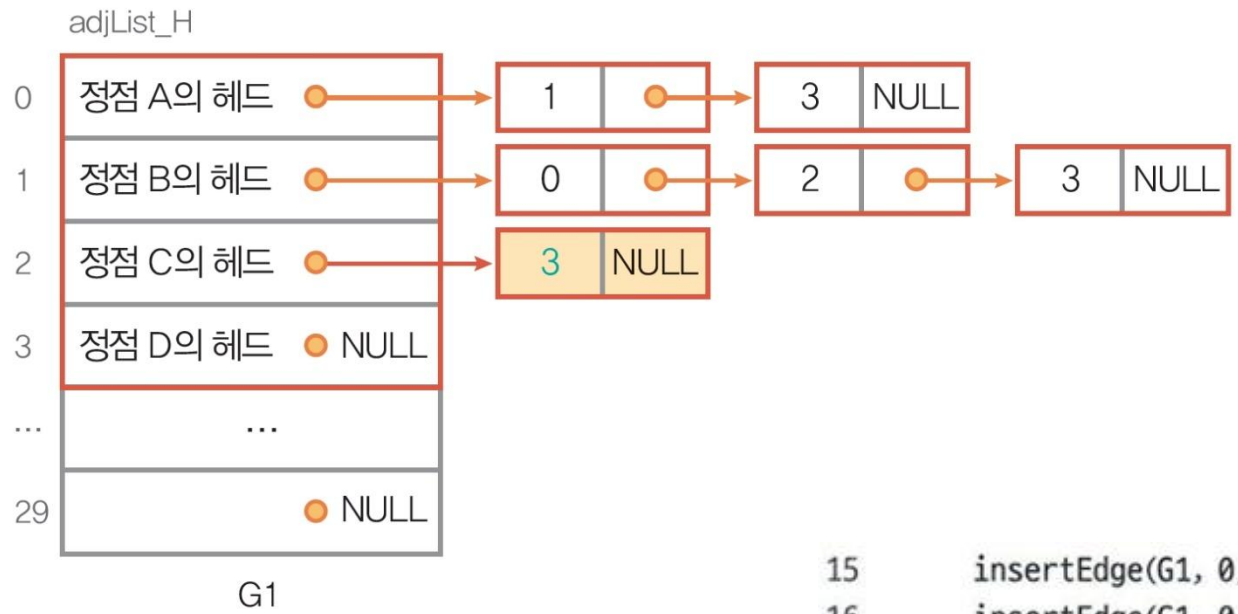



```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 20행 : 그래프 G1에 간선 (2, 3) 삽입

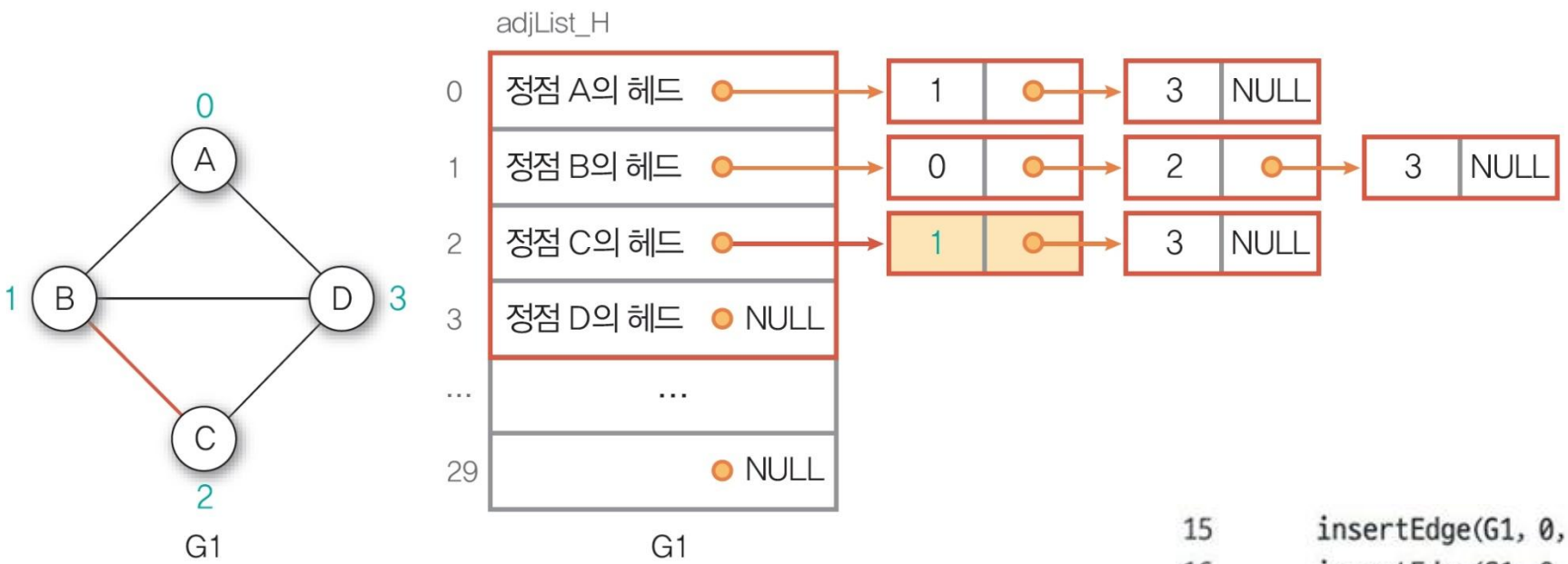





```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

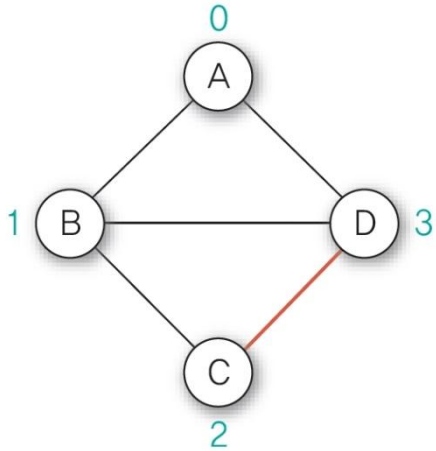
- 21행 : 그래프 G1에 간선 (2, 1) 삽입



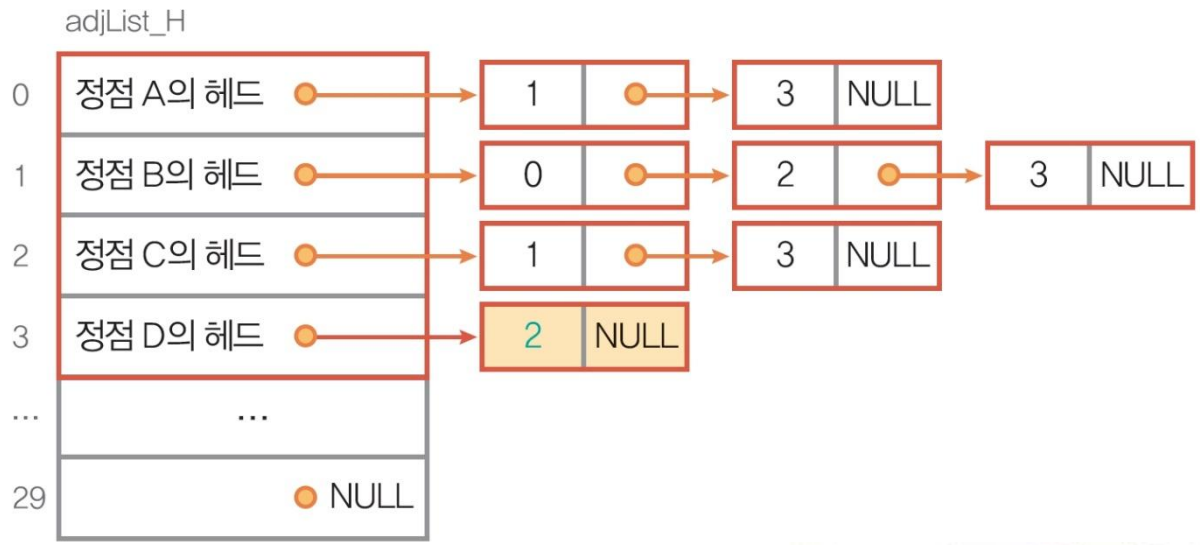

```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 22행 : 그래프 G1에 간선 (3, 2) 삽입



G1

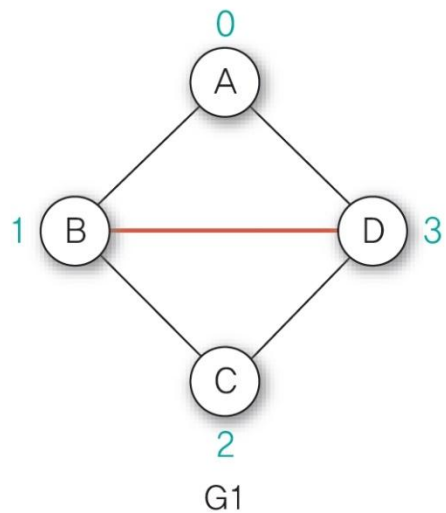


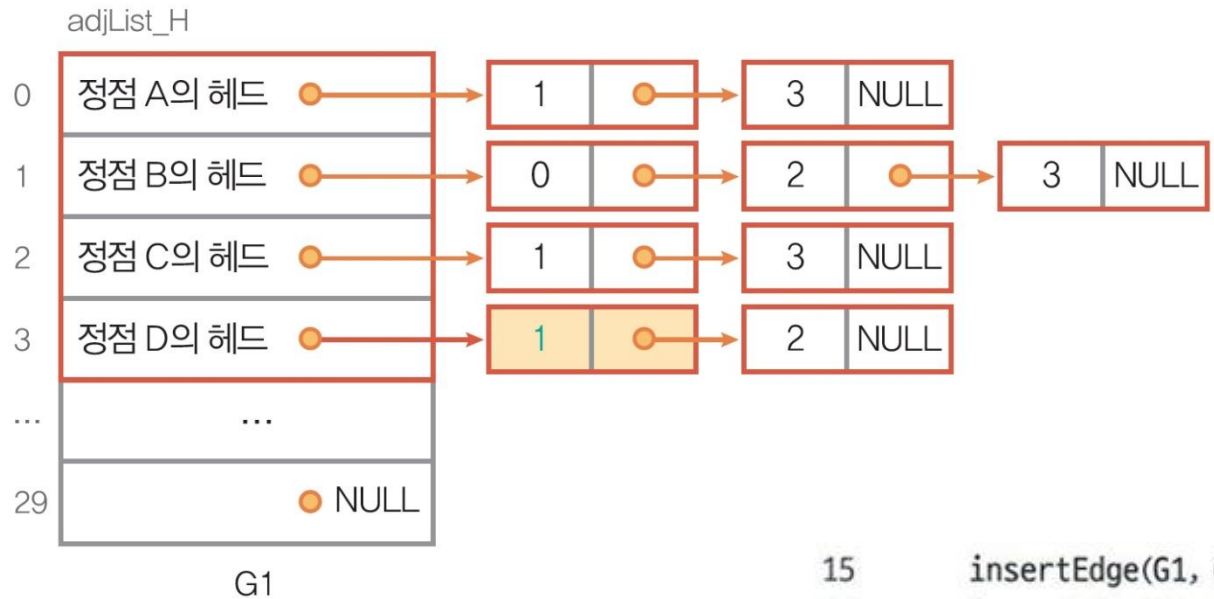
G1

```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 23행 : 그래프 G1에 간선 (3, 1) 삽입

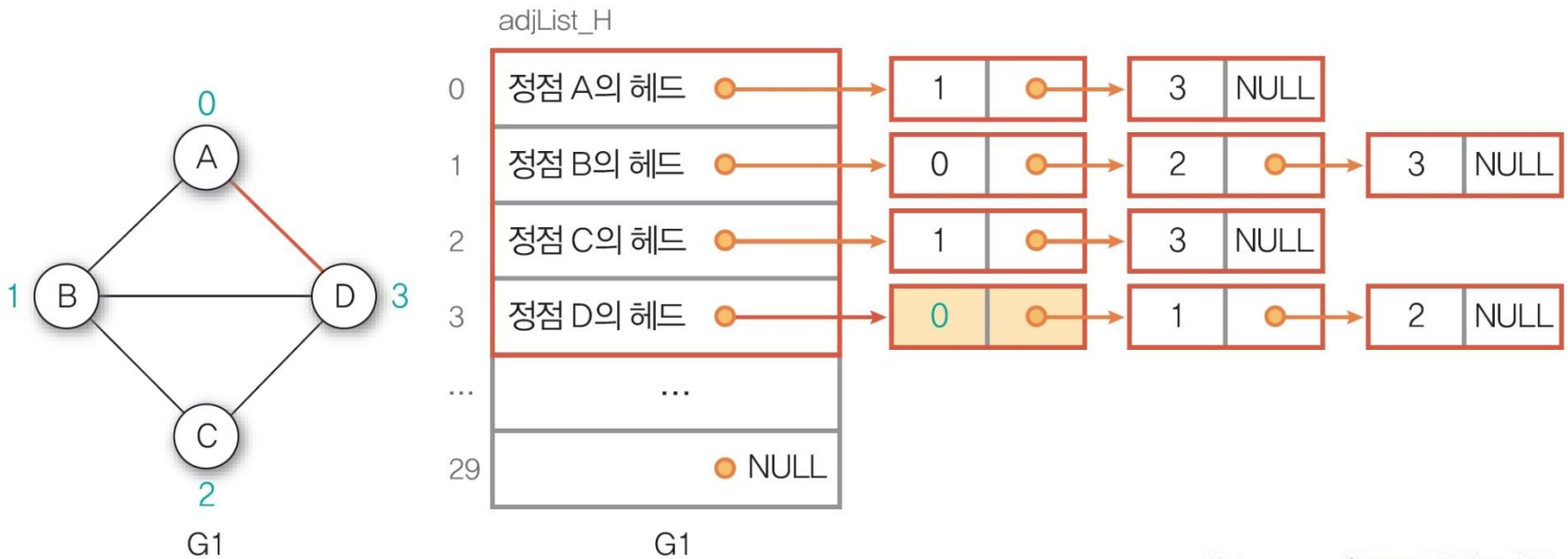





```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

- 24행 : 그래프 G1에 간선 (3, 0) 삽입



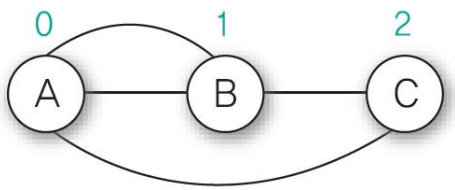

```
15      insertEdge(G1, 0, 3);
16      insertEdge(G1, 0, 1);
17      insertEdge(G1, 1, 3);
18      insertEdge(G1, 1, 2);
19      insertEdge(G1, 1, 0);
20      insertEdge(G1, 2, 3);
21      insertEdge(G1, 2, 1);
22      insertEdge(G1, 3, 2);
23      insertEdge(G1, 3, 1);
24      insertEdge(G1, 3, 0);
```

2. 그래프의 구현

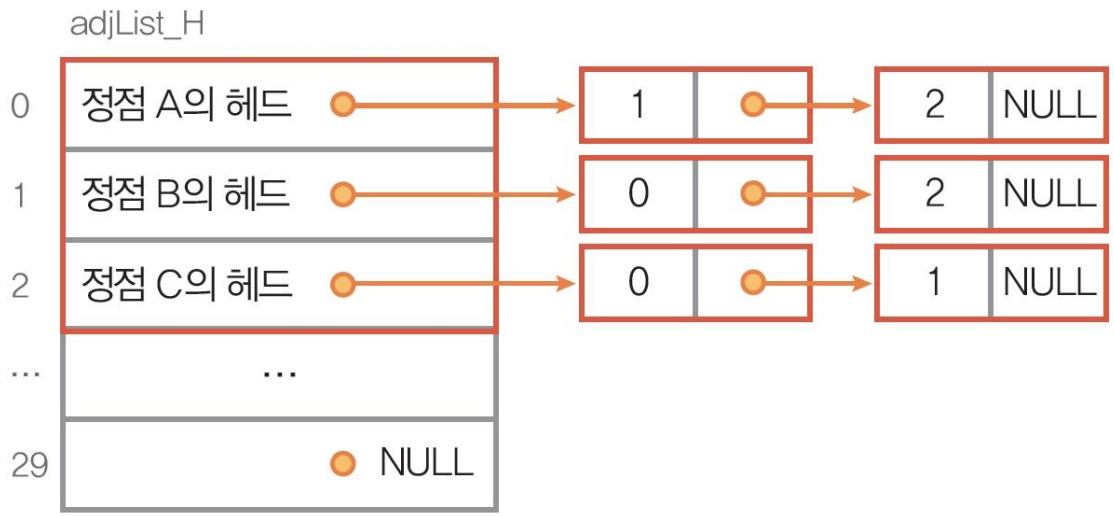
- 27~35행은 그래프 G2 생성, 정점과 간선 삽입하여 G2 인접 리스트 구성
  - 28~29행 : 그래프 G2에 정점 세 개 삽입

2. 그래프의 구현

- 30~35행 : 그래프 G2에 간선 여섯 개((0,2), (0,1), (1,2), (1,0), (2,1), (2,0)) 삽입



G2



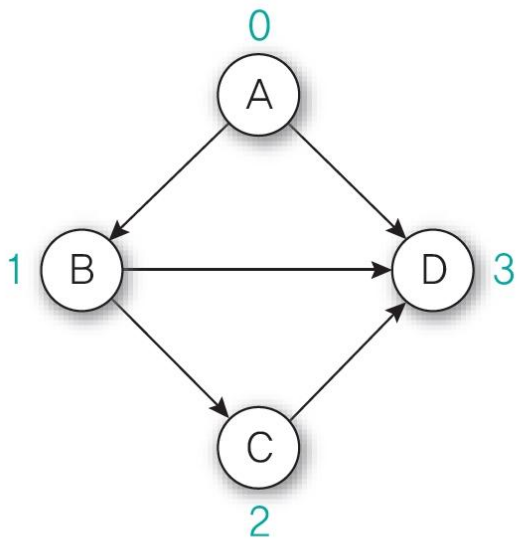
G2

2. 그래프의 구현

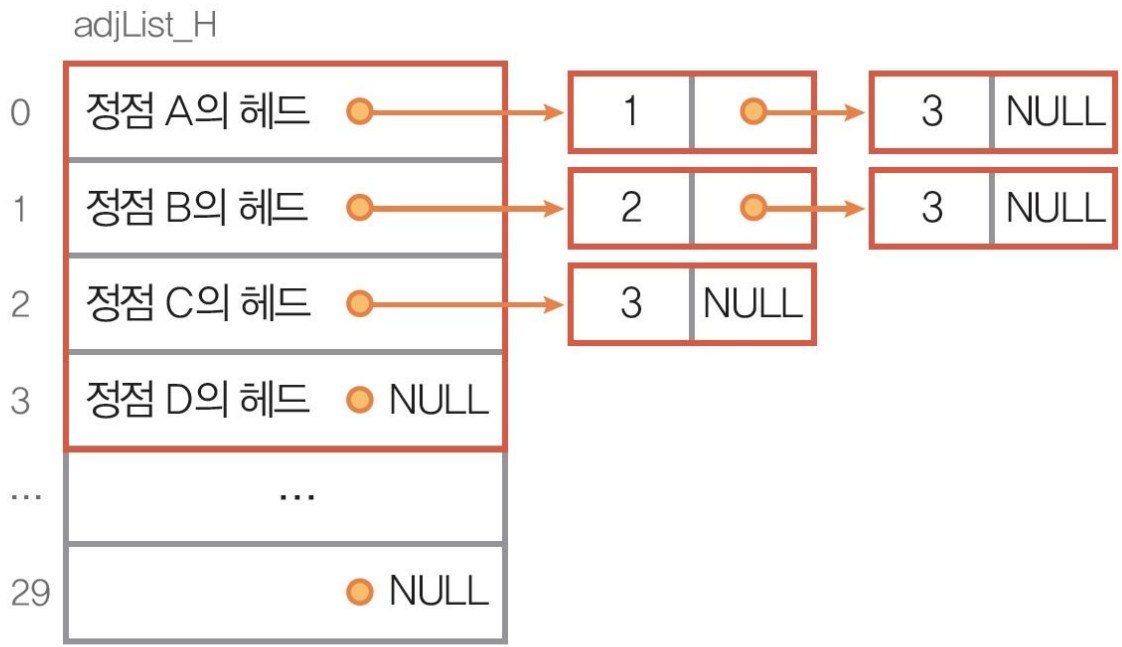
- 38~45행은 그래프 G3 생성, 정점과 간선을 삽입하여 인접 리스트 구성
  - 39~40행 : 그래프 G3에 정점 네 개 삽입

2. 그래프의 구현

- 41~45행 : 그래프 G3에 방향 간선 다섯 개(<0,3>, <0,1>, <1,3>, <1,2>, <2,3>) 삽입



G3



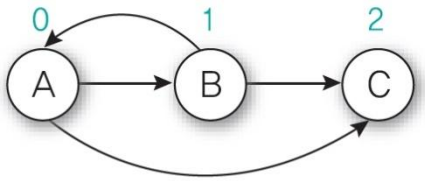
G3

2. 그래프의 구현

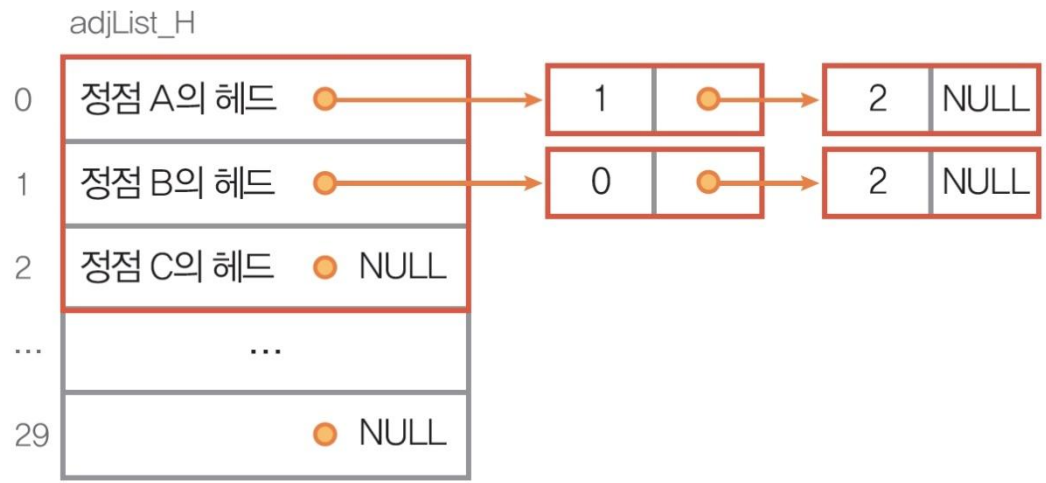
- 48~54행은 그래프 G4 생성, 정점과 간선을 삽입하여 G4 인접 리스트 구성
  - 49~50행 : 그래프 G4에 정점 세 개 삽입

2. 그래프의 구현

- 51~54행 : 그래프 G4에 방향 간선 네 개(<0,2>, <0,1>, <1,2>, <1,0>) 삽입



G4



G4

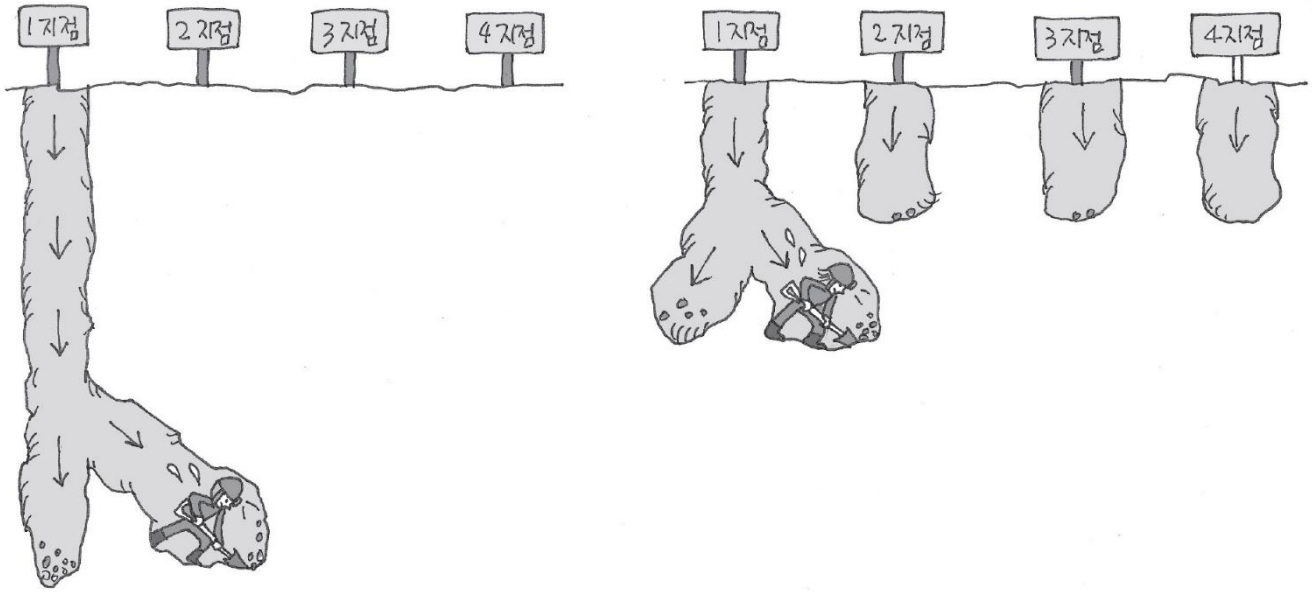
3. 그래프의 순회

- 그래프 순회graph traversal, 그래프 탐색graph search
  - 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문하여 처리하는 연산
  - 그래프 탐색 방법
    - 깊이 우선 탐색depth first search : DFS
    - 너비 우선 탐색breadth 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

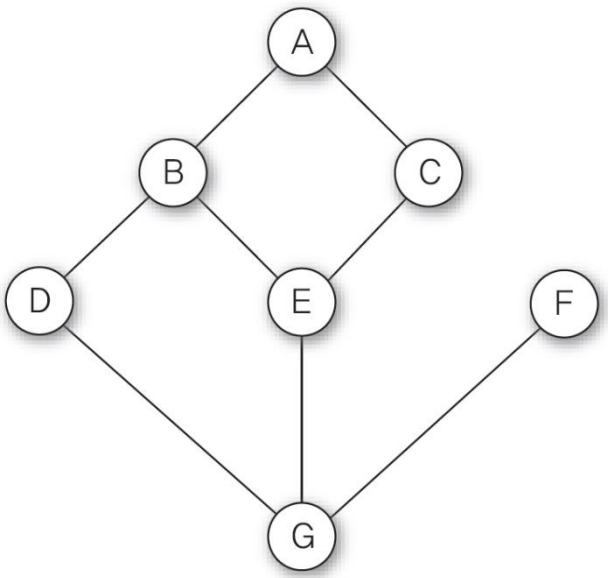
알고리즘 8-1 그래프의 깊이 우선 탐색

```
DFS(v)
    for (i ← 0; i<n; i ← i + 1) do {   모든 정점을 방문하지 않은 상태로 초기화
        visited[i]←false;
    }
    stack ← createStack();
    visited[v] ← true;
    v 방문; push (stack, v)
    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()
```

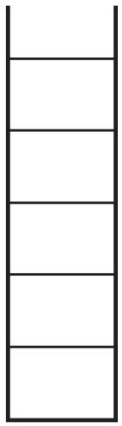
3. 그래프의 순회

- 알고리즘에 따라 그래프 G9를 깊이 우선 순회하는 과정

  ⓪ 그래프 G9의 깊이 우선 탐색을 위한 초기 상태 : 배열 visited를 false로 초기화하고 공백 스택 생성



G9



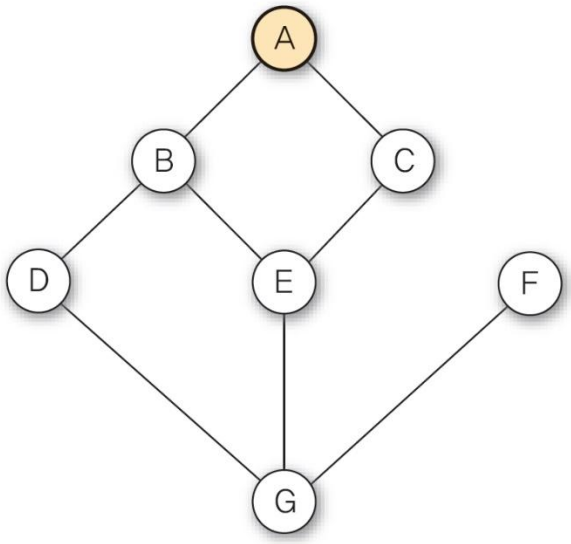
stack

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

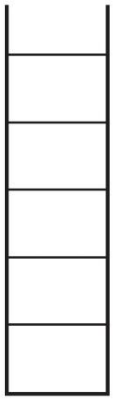
3. 그래프의 순회

① 정점 A를 시작으로 깊이 우선 탐색을 시작

```
visited[A] ← true;
A 방문;
```



G9

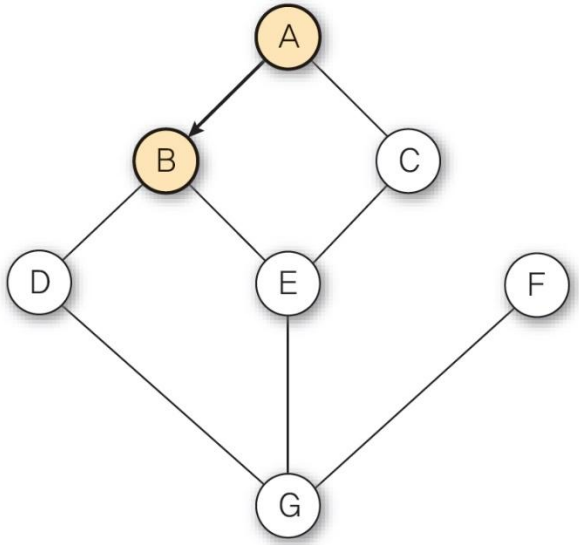


stack

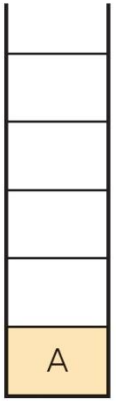
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	F	F	F	F	F	F

② 정점 A에 방문하지 않은 정점 B, C가 있으므로 A를 스택에 push하고, 인접 정점 B와 C 중에서 오름차순에 따라 B를 선택하여 탐색을 계속함

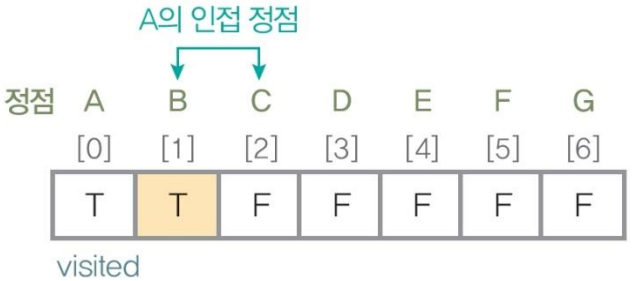
```
push(stack, A);
visited[B] ← true;
B 방문;'
```



G9



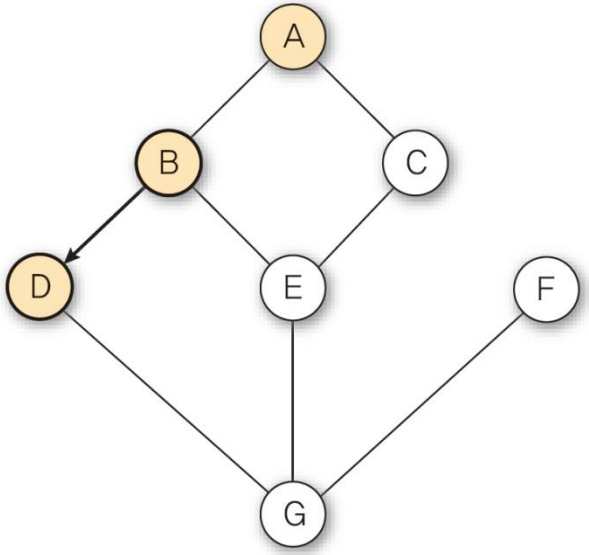
stack

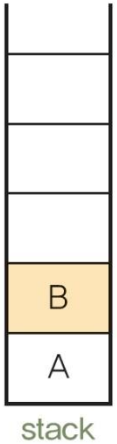


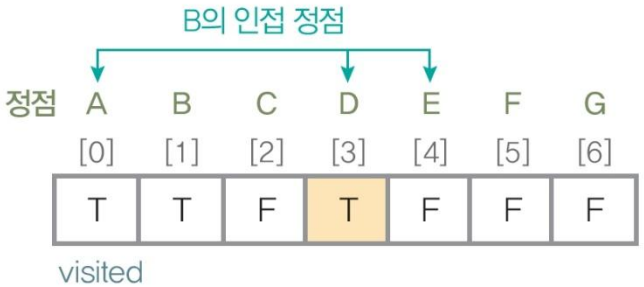
3. 그래프의 순회

③ 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;
```



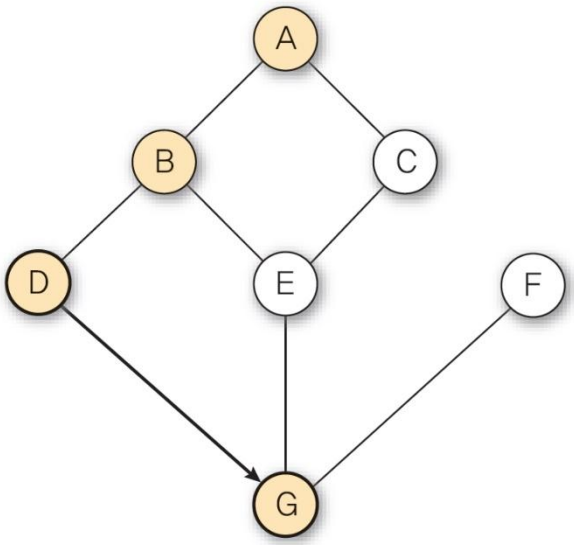


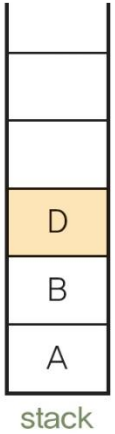


3. 그래프의 순회

④ 정점 D에 방문하지 않은 정점 G가 있으므로 D를 스택에 push하고, 인접 정점 G를 선택하여 탐색을 계속

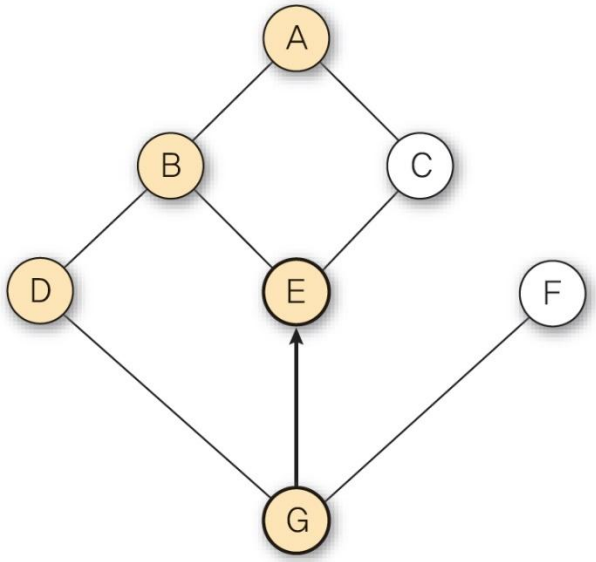
```
push(stack, D);
visited[G] ← true;
G 방문;
```

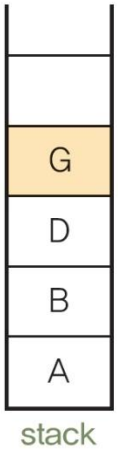




⑤ 정점 G에 방문하지 않은 정점 E, F가 있으므로 G를 스택에 push하고, 방문하지 않은 인접 정점 E와 F중에서 오름차순에 따라 E를 선택하여 탐색을 계속

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



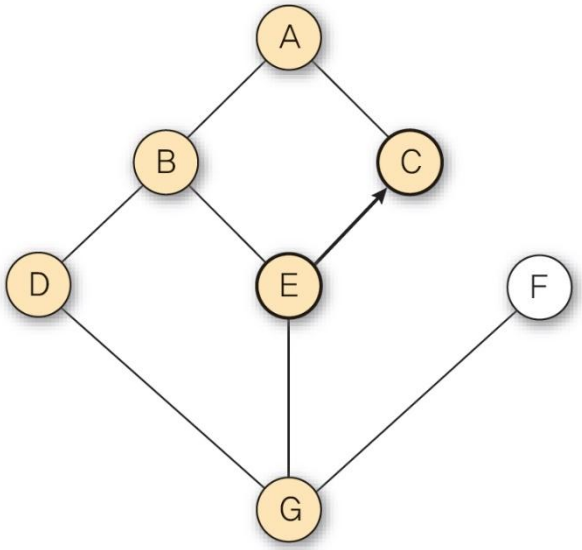


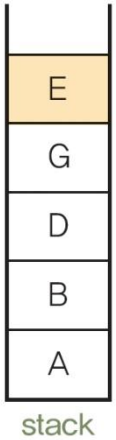
G의 인접 정점

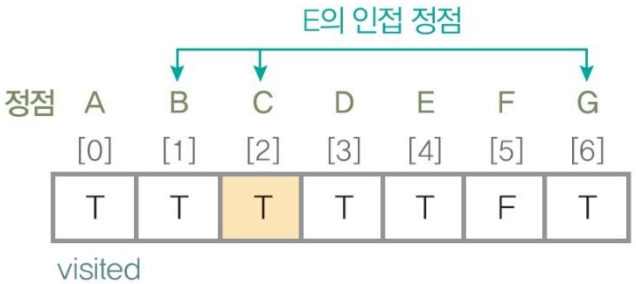
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	T	F	T	T	F	T

⑥ 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;
```

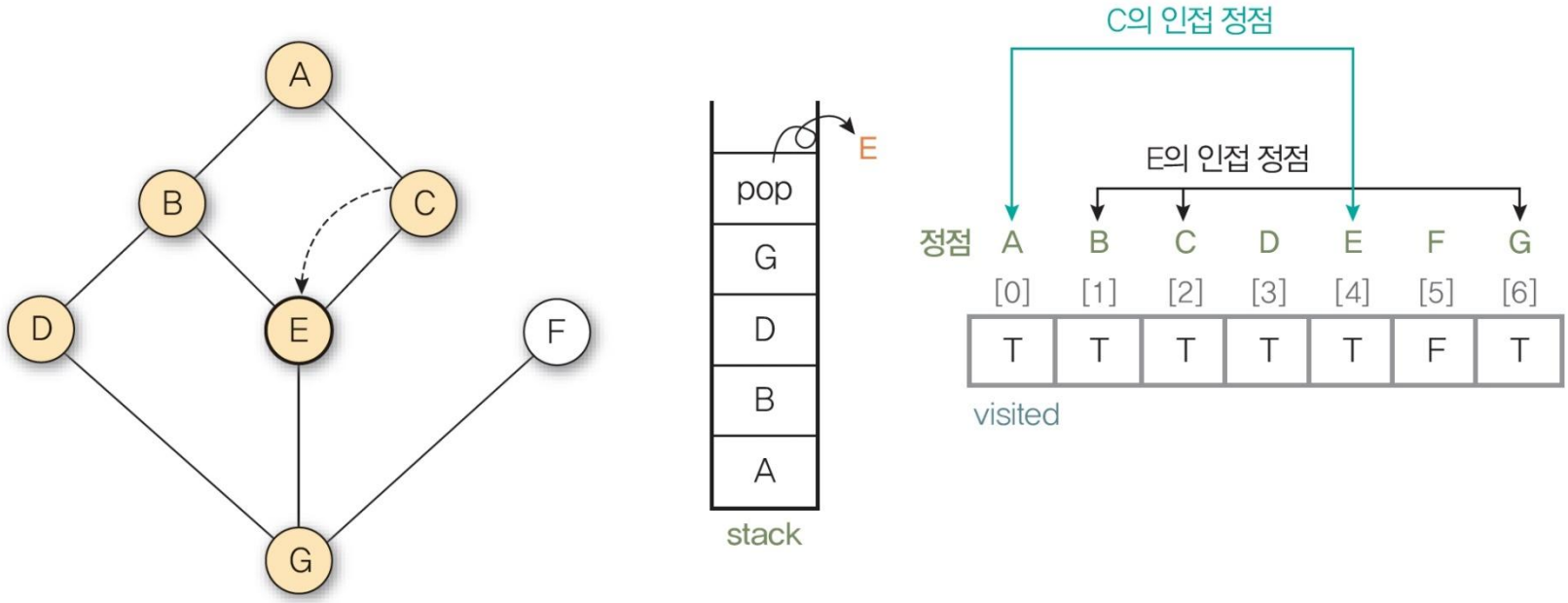






⑦ 정점 C에서 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 E에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 E는 방문하지 않은 인접 정점이 없음

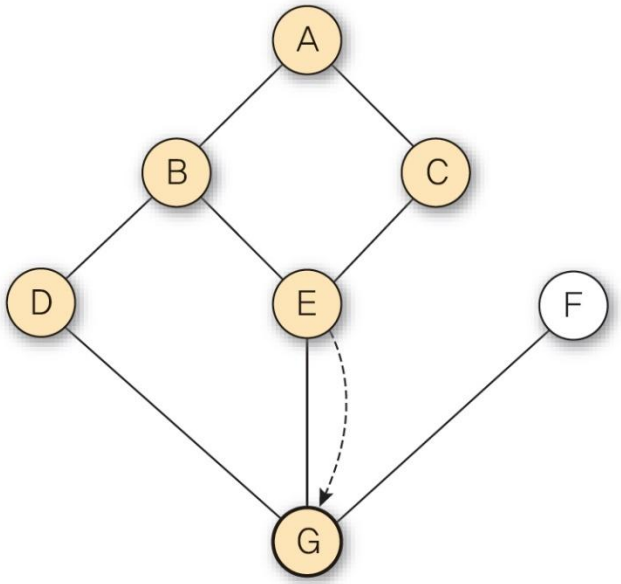
```
pop(stack);
```

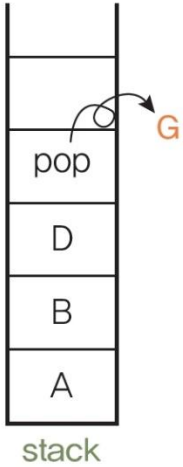


3. 그래프의 순회

⑧ 현재 정점 E에서 방문할 수 있는 인접 정점이 없으므로, 다시 스택을 pop하여 받은 정점 G에 대해서 방문하지 않은 인접 정점이 있는지 확인

```
pop(stack);
```

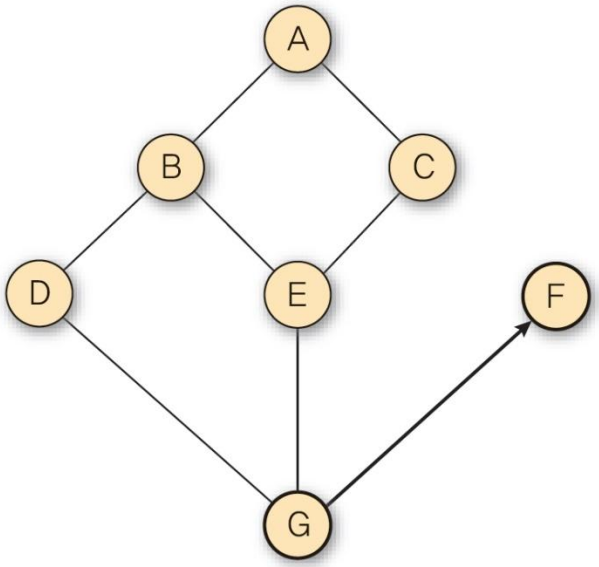


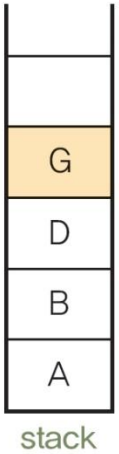


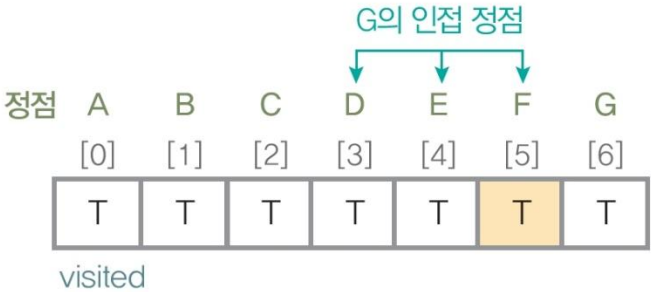
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	T	T	T	T	F	T

⑨ 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;
```

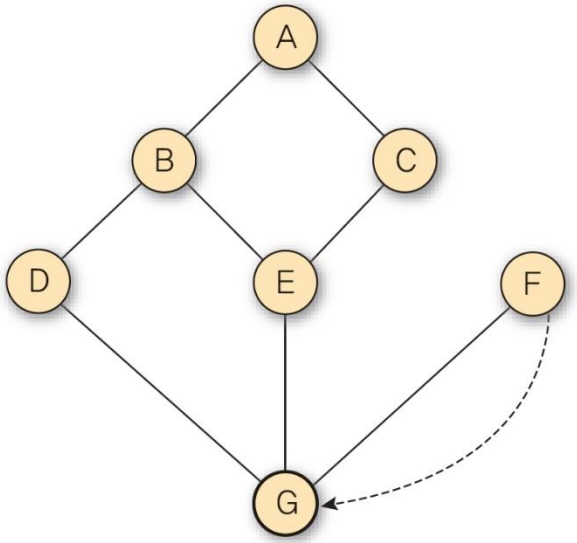


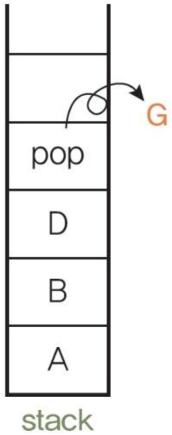


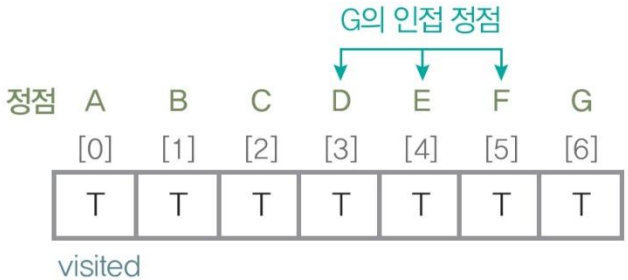


⑩ 현재 정점 F에서 방문하지 않은 인접 정점이 없으므로 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 G에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 G는 방문하지 않은 인접 정점이 없음

```
pop(stack);
```



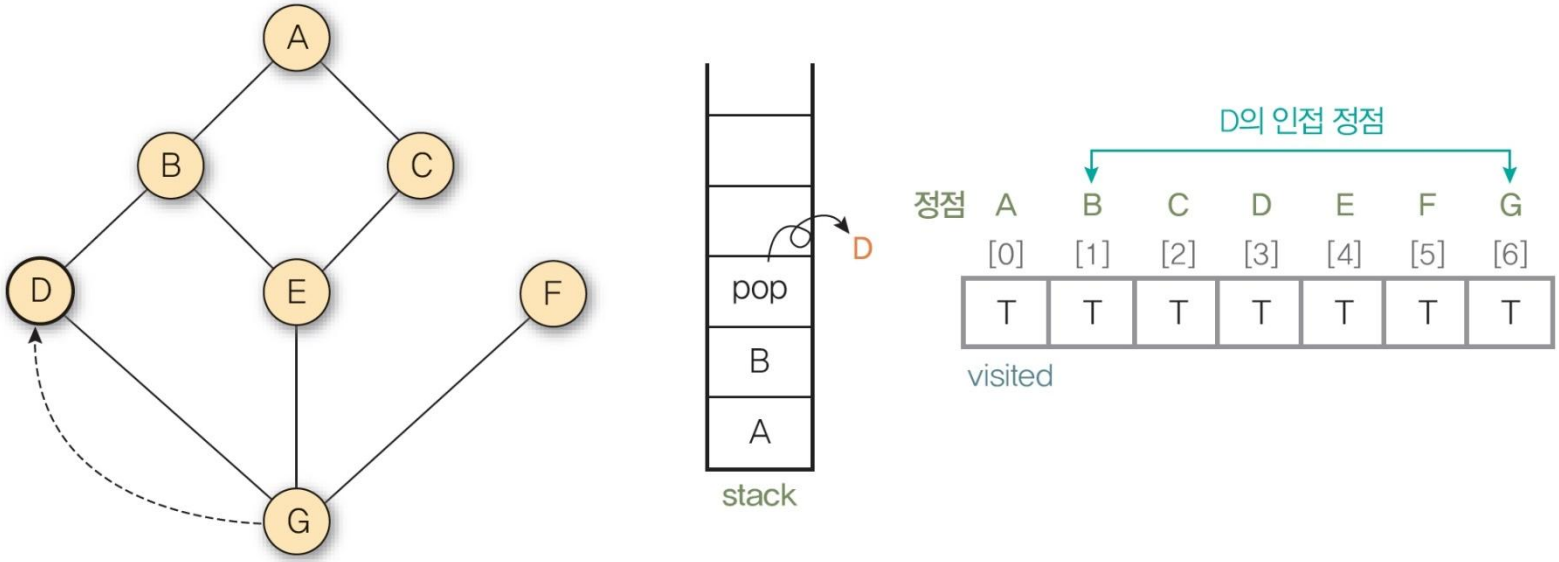




3. 그래프의 순회

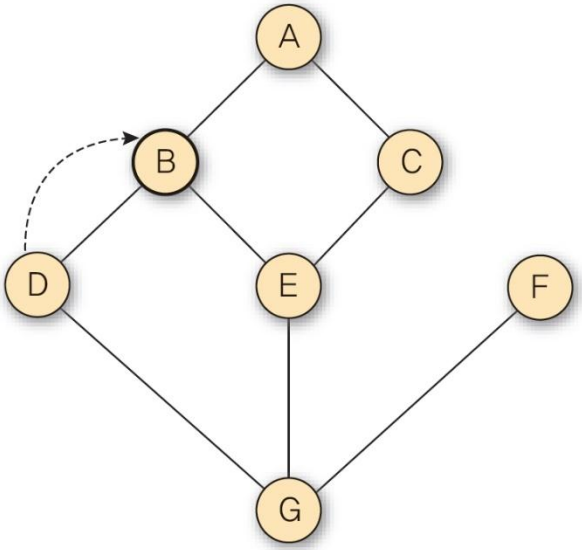
⑪ 현재 정점 G에서 방문하지 않은 인접 정점이 없으므로 다시 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 D에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 D는 방문하지 않은 인접 정점이 없음

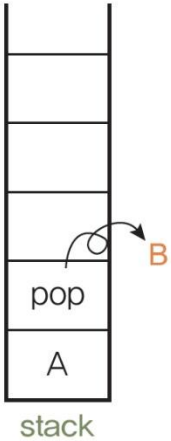
```
pop(stack);
```

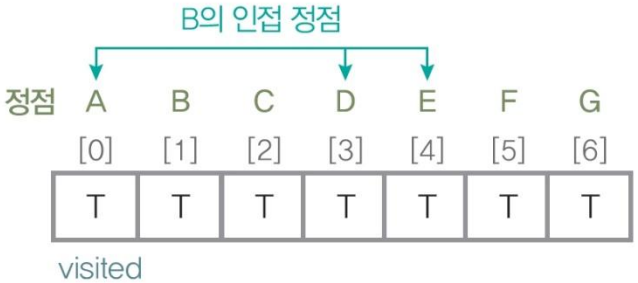


⑫ 현재 정점 D에서 방문하지 않은 인접 정점이 없으므로 다시 마지막 정점으로 돌아가기 위해 스택을 pop하여 받은 정점 B에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 B는 방문하지 않은 인접 정점이 없음

```
pop(stack);
```

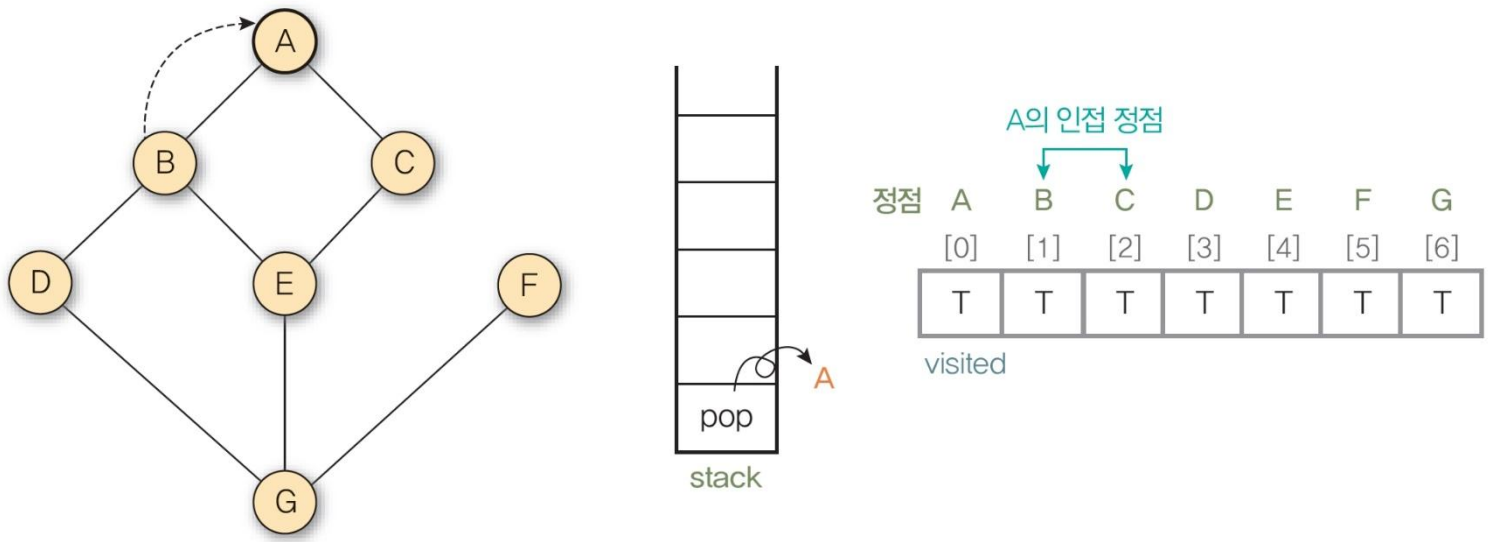






⑬ 현재 정점 B에서 방문하지 않은 인접 정점이 없으므로 다시 마지막 정점으로 돌아가기 위해 스택을 pop. 그리고 받은 정점 A에 대해서 방문하지 않은 인접 정점이 있는지 확인. 정점 A는 방문 하지 않은 인접 정점이 없음

```
pop(stack);
```



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

3. 그래프의 순회

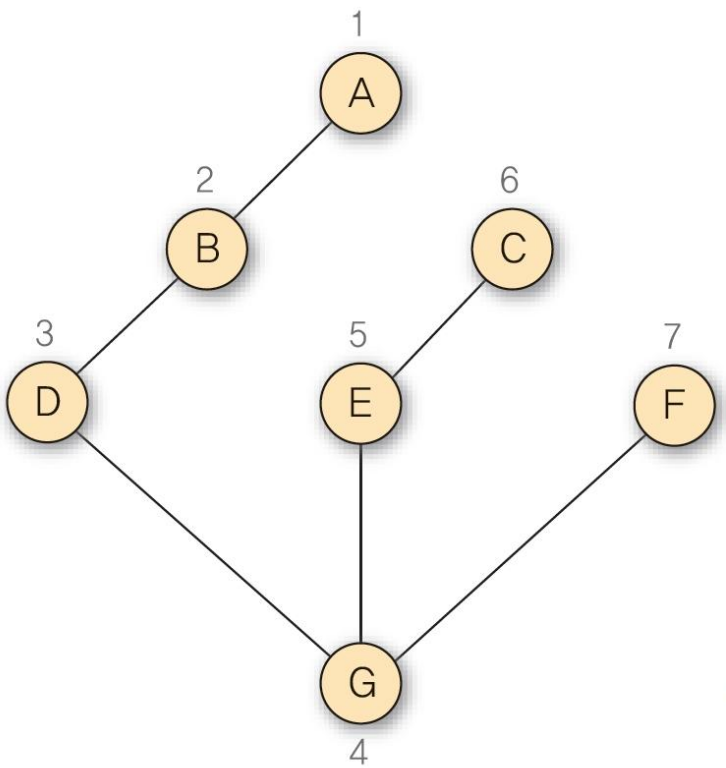


그림 8-14 그래프 G9의 깊이 우선 탐색 경로 : A-B-D-G-E-C-F

3. 그래프의 순회

- [예제 8-3] 그래프를 깊이 우선 탐색하기 : 교재 459p
- 실행 결과

```
G9의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> D -> E
        정점 C의 인접 리스트 -> A -> E
        정점 D의 인접 리스트 -> B -> G
        정점 E의 인접 리스트 -> B -> C -> G
        정점 F의 인접 리스트 -> G
        정점 G의 인접 리스트 -> D -> E -> F

///////////////

깊이 우선 탐색 >> A B D G E C F
```

3. 그래프의 순회

예제 8-3 그래프를 깊이 우선 탐색하기

[stackL.h]

```
// [예제 5-2]의 [stackL.h] 사용
```

[stackL.c]

```
// [예제 5-2]의 [stackL.c] 사용
```

[adjList.h]

```
// [예제 8-2]의 [adjList.h]를 수정하여 사용
#pragma once
#define MAX_VERTEX 30          // 헤드 포인터 배열의 최대 크기
#define FALSE 0                //추가!
#define TRUE 1                 //추가!

// 인접 리스트의 노드 구조를 구조체로 정의
typedef struct graphNode {
    int vertex;                // 정점을 나타내는 데이터 필드
    struct graphNode* link;    // 다음 인접 정점을 연결하는 링크 필드
} graphNode;

// 그래프를 인접 리스트로 표현하기 위한 구조체 정의
typedef struct graphType {
    int n;                                  // 그래프의 정점 개수
    graphNode* adjList_H[MAX_VERTEX];       // 그래프 정점에 대한 헤드 포인터 배열
    int visited[MAX_VERTEX];                // 정점에 대한 방문 표시를 위한 배열 추가!
} graphType;
```

3. 그래프의 순회

```
19
20  void createGraph(graphType* g);
21  void insertVertex(graphType* g, int v);
22  void insertEdge(graphType* g, int u, int v);
23  void print_adjList(graphType* g);
```

[adjList.c]

```
01  #include <stdio.h>
02  #include "adjList.h"
03
04  // 공백 그래프를 생성하는 연산
05  void createGraph(graphType* g) {
06      int v;
07      g->n = 0;                    // 그래프의 정점 개수를 0으로 초기화
08      for (v = 0; v < MAX_VERTEX; v++) {
09          g->adjList_H[v] = NULL; // 그래프의 정점에 대한 헤드 포인터 배열을 NULL로 초기화
10          g->visited[v] = FALSE;  // 그래프의 정점에 대한 배열 visited를 FALSE로 초기화 추가!
11      }
12  }
…   // 나머지는 [예제 8-2]의 [adjList.c]의 12~48행과 동일
```

3. 그래프의 순회

[DFS.h]

```
#pragma once
#include "adjList.h"
void DFS_adjList(graphType* g, int v);
```

[DFS.c]

```
#include <stdio.h>
#include "adjList.h"
#include "stackL.h"
#include "DFS.h"
```

3. 그래프의 순회

[알고리즘 8-1] 구현

```c
// 그래프 g에서 정점 v에 대한 깊이 우선 탐색 연산
void DFS_adjList(graphType* g, int v) {
    graphNode* w;
    top = NULL;             // 스택의 top 설정
    push(v);                // 깊이 우선 탐색을 시작하는 정점 v를 스택에 push
    g->visited[v] = TRUE;   // 정점 v를 방문했으므로 해당 배열값을 TRUE로 설정
    printf(" %c", v + 65);

    // 스택이 공백이 아닌 동안 깊이 우선 탐색 반복
    while (!isStackEmpty()) {
        w = g->adjList_H[v];
        // 인접 정점이 있는 동안 수행
        while (w) {
            // 현재 정점 w를 방문하지 않은 경우
            if (!g->visited[w->vertex]) {
                push(w->vertex);                // 현재 정점 W를 스택에 push
                g->visited[w->vertex] = TRUE;   // 정점 w에 대한 배열값을 TRUE로 설정
                printf(" %c", w->vertex + 65);  // 정점 0~6을 A~G로 바꾸어서 출력
                v = w->vertex;
                w = g->adjList_H[v];
            }
            // 현재 정점 w가 이미 방문된 경우
            else w = w->link;
        }
        v = pop();    // 현재 정점에서 순회를 진행할 인접 정점이 더 없는 경우에 스택 pop!
    } // 스택이 공백이면 깊이 우선 탐색 종료
}
```

v : 현재 정점
w : 다음 정점

– 인접 리스트 예: 그래프 G1, G2, G3, G4

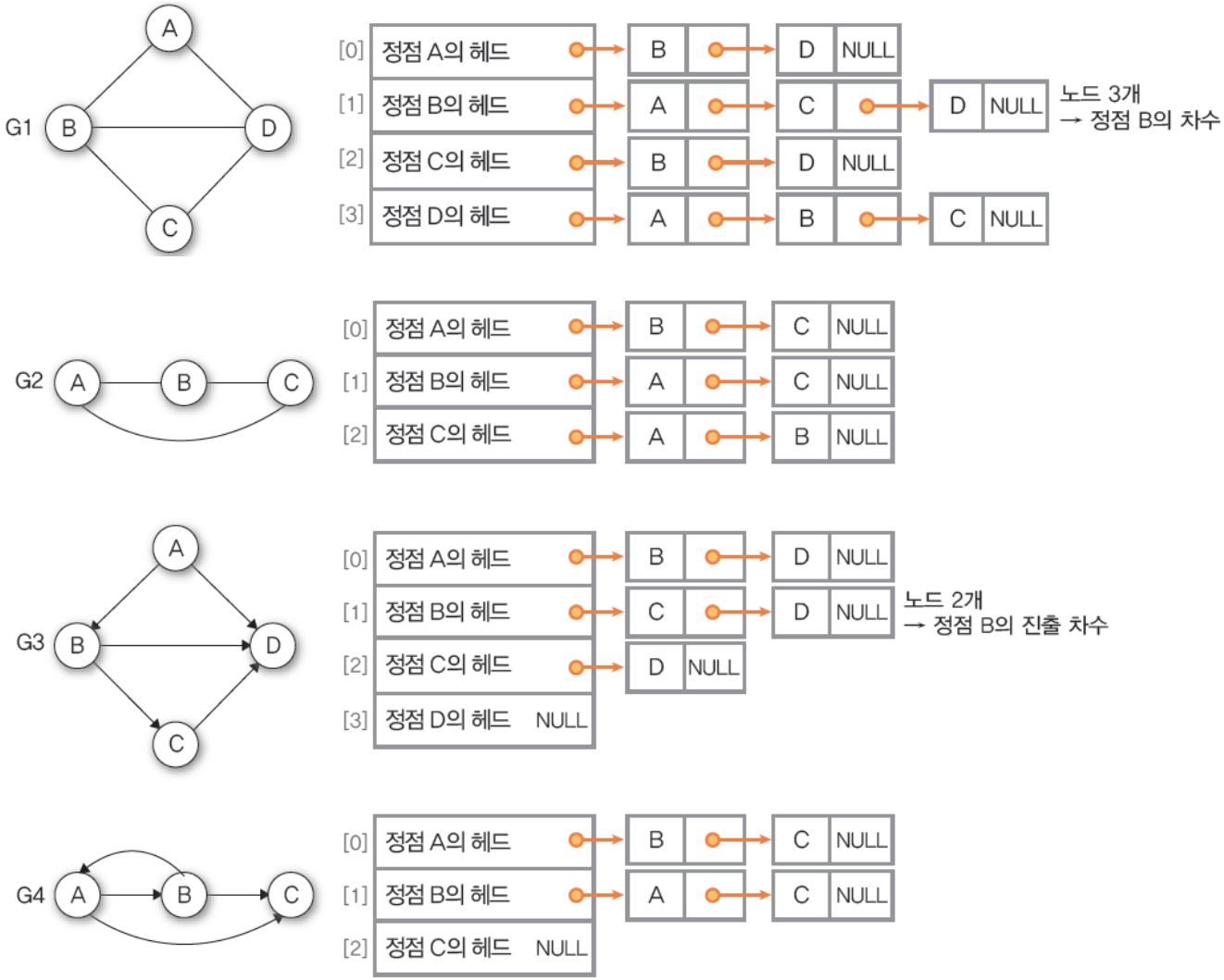
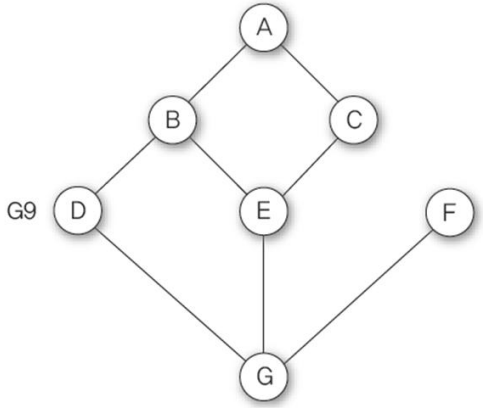


그림 8-11 그래프의 인접 리스트 표현 예

3. 그래프의 순회

[ex8_3.c]

```c
#include <stdio.h>
#include "DFS.h"

 int main(void) {
    int i;
    graphType* G9;
    G9 = (graphType*)malloc(sizeof(graphType));
    createGraph(G9);
    // 그래프 G9 구성 : 정점 u에 대한 간선 (u,v)는 v가 큰 것부터 삽입
    for (i = 0; i < 7; i++)
        insertVertex(G9, i);
    insertEdge(G9, 0, 2);
    insertEdge(G9, 0, 1);
    insertEdge(G9, 1, 4);
    insertEdge(G9, 1, 3);
    insertEdge(G9, 1, 0);
    insertEdge(G9, 2, 4);
    insertEdge(G9, 2, 0);
    insertEdge(G9, 3, 6);
    insertEdge(G9, 3, 1);
    insertEdge(G9, 4, 6);
    insertEdge(G9, 4, 2);
    insertEdge(G9, 4, 1);
```



G9의 인접 리스트

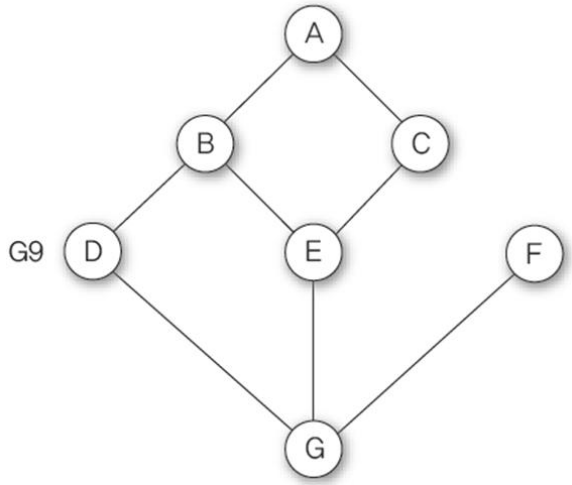
정점 A의 인접 리스트 –> B –> C
정점 B의 인접 리스트 –> A –> D –> E
정점 C의 인접 리스트 –> A –> E
정점 D의 인접 리스트 –> B –> G
정점 E의 인접 리스트 –> B –> C –> G
정점 F의 인접 리스트 –> G
정점 G의 인접 리스트 –> D –> E –> F

//////////////

깊이 우선 탐색 》 A B D G E C F

3. 그래프의 순회

```
24        insertEdge(G9, 5, 6);
25        insertEdge(G9, 6, 5);
26        insertEdge(G9, 6, 4);
27        insertEdge(G9, 6, 3);
28        printf("\n G9의 인접 리스트 ");
29        print_adjList(G9);          // G9의 인접 리스트를 확인용 출력
30
31        printf("\n\n//////////////\n\n깊이 우선 탐색 >> ");
32        DFS_adjList(G9, 0);         // 0번 정점인 정점 A에서 깊이 우선 탐색 시작
33
34        getchar();   return 0;
35    }
```




```
G9의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> D -> E
        정점 C의 인접 리스트 -> A -> E
        정점 D의 인접 리스트 -> B -> G
        정점 E의 인접 리스트 -> B -> C -> G
        정점 F의 인접 리스트 -> G
        정점 G의 인접 리스트 -> D -> E -> F

//////////////

깊이 우선 탐색 >> A B D G E C F
```

- 너비 우선 탐색breadth first search : BFS
  - 탐색 순서

❶ 시작 정점 v를 결정하여 방문한다.

❷ 정점 v에 인접한 정점 중에서 방문하지 않은 정점을 차례로 방문하면서 큐에 enQueue한다.

❸ 방문하지 않은 인접한 정점이 없으면, 방문했던 정점에서 인접한 정점을 다시 차례로 방문하기 위해 큐에서 deQueue 하여 받은 정점을 v로 설정하고 ❷를 반복한다.

❹ 큐가 공백이 될 때까지 ❷~❸을 반복한다.

그림 8-15 너비 우선 탐색 순서

3. 그래프의 순회

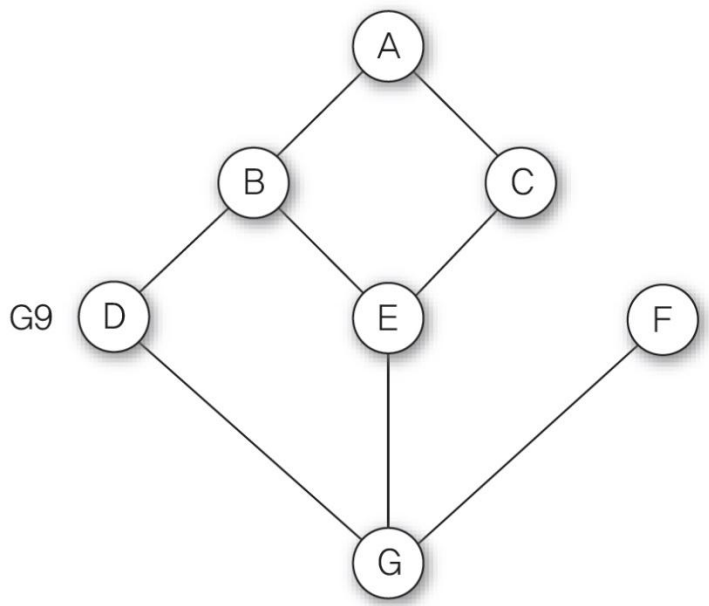
알고리즘 8-2 그래프의 너비 우선 탐색

```
BFS(v)
    for (i ← 0; i<n; i ← i + 1) do {   모든 정점을 방문하지 않은 상태로 초기화
        visited[i] ← false;
    }
    Q ← createQueue();
    visited[v] ← true;
    v 방문;    enQueue (Q, v);
    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()
```

3. 그래프의 순회

– 알고리즘에 따른 그래프 G9의 너비 우선 탐색 과정

⓪ 초기 상태 : 배열 visited를 False로 초기화, 공백 큐 생성



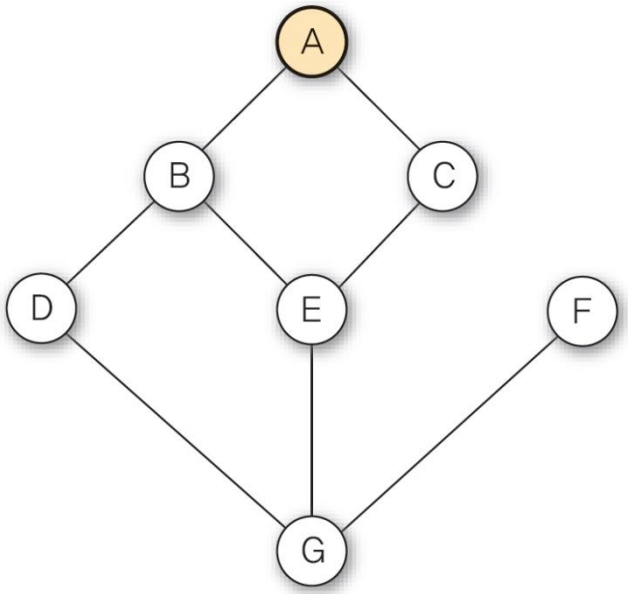
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	F	F	F	F	F	F	F

Q

3. 그래프의 순회

① 정점 A를 시작으로 너비 우선 탐색을 시작

```
visited[A] ← true;
A 방문;
```

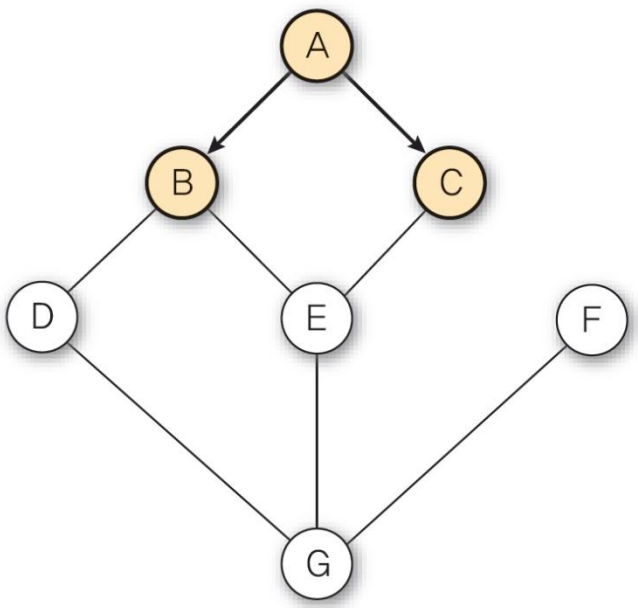


정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	F	F	F	F	F	F

Q

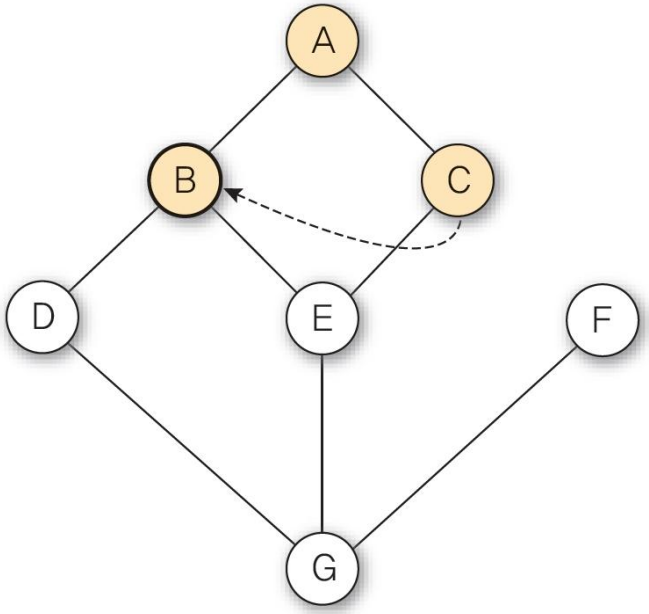
② 정점 A에서 방문하지 않은 모든 인접 정점 B, C를 방문하고 큐에 enQueue

```
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));
```

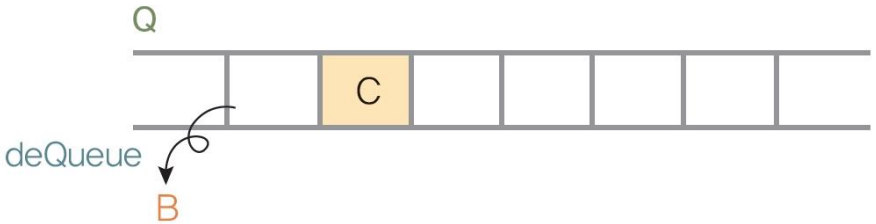


③ 정점 A에 대한 인접 정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 B를 받음

```
v ← deQueue(Q);
```

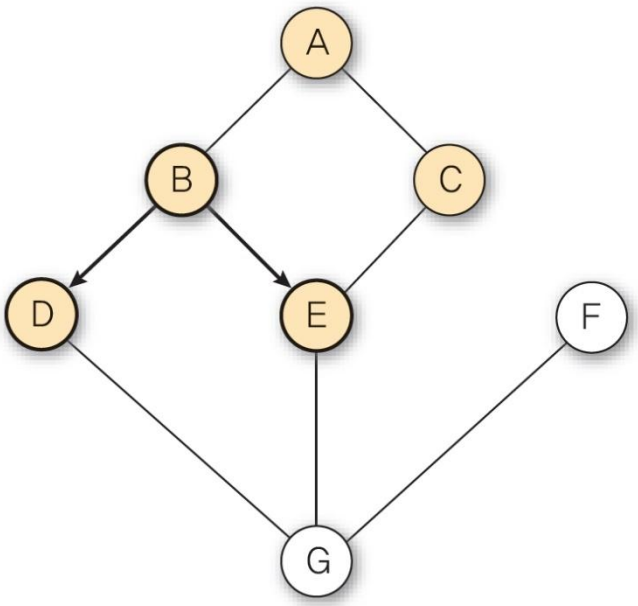


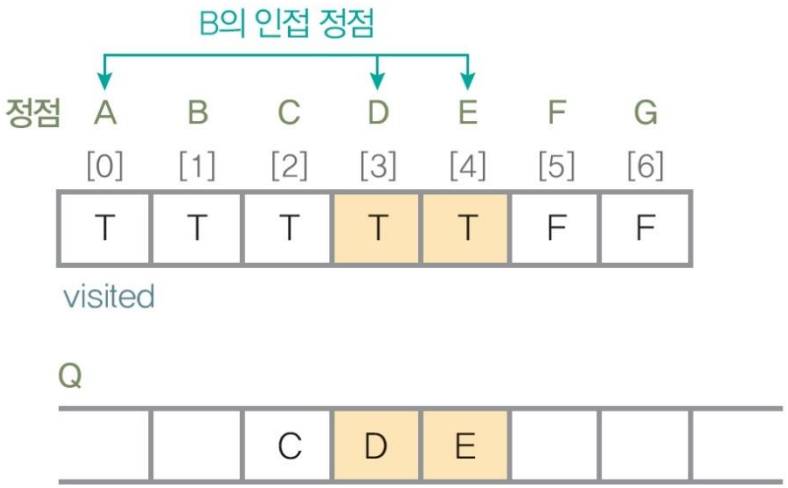
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	T	T	F	F	F	F



④ 정점 B에서 방문하지 않은 모든 인접 정점 D, E를 방문하고 큐에 enQueue

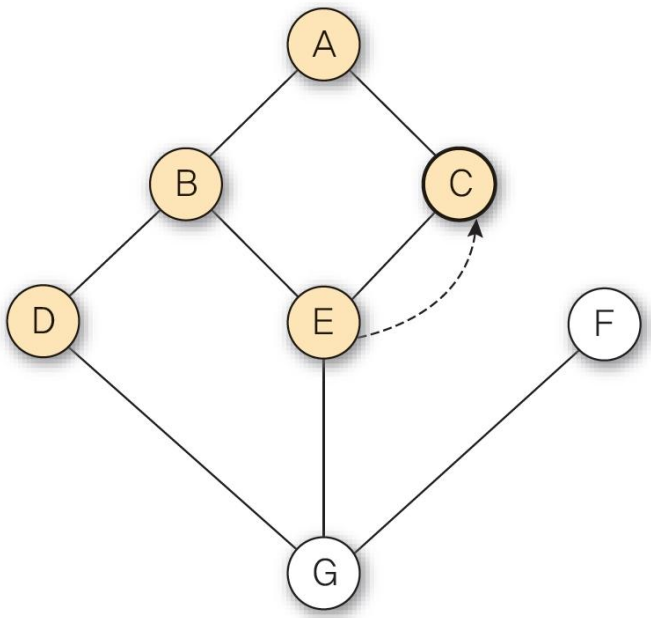
```
visited[(B가 방문하지 않은 인접 정점 D와 E)] ← true;
(B가 방문하지 않은 인접 정점 D와 E) 방문;
enQueue(Q, (B가 방문하지 않은 인접 정점 D와 E));
```

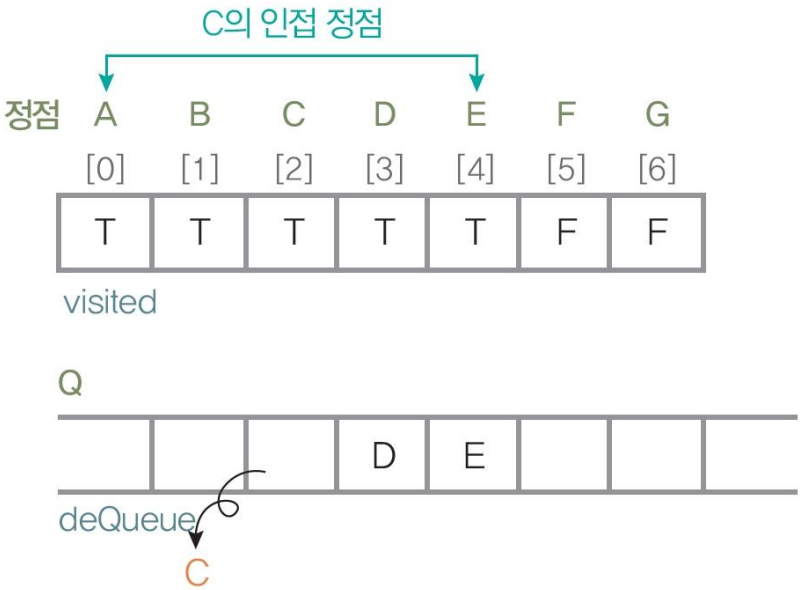




⑤ 정점 B에 대한 인접 정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 C를 받음

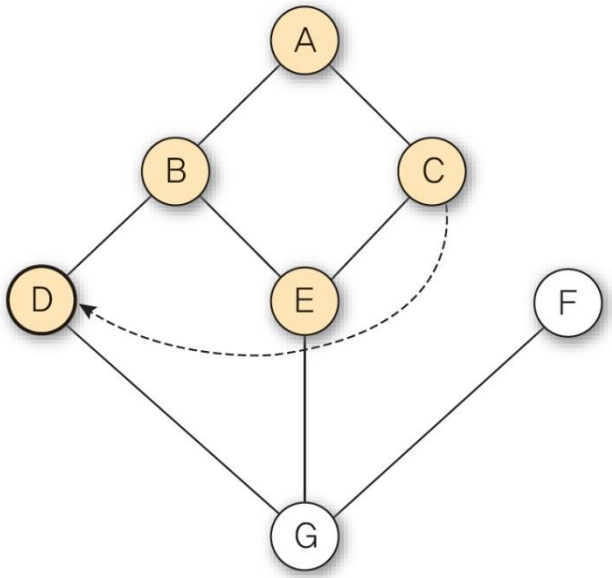
```
v ← deQueue(Q);
```



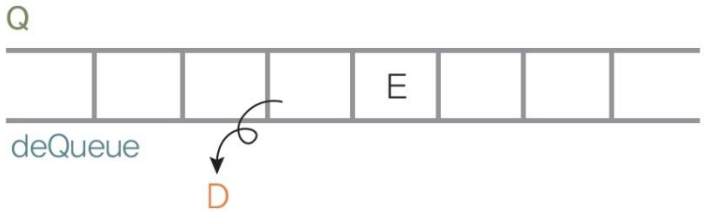


⑥ 정점 C에는 방문하지 않은 인접 정점이 없으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 D를 받음

```
v ← deQueue(Q);
```

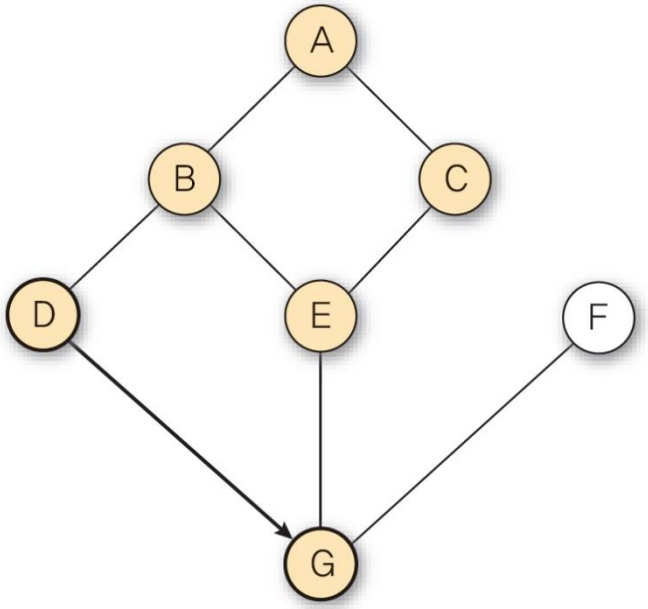


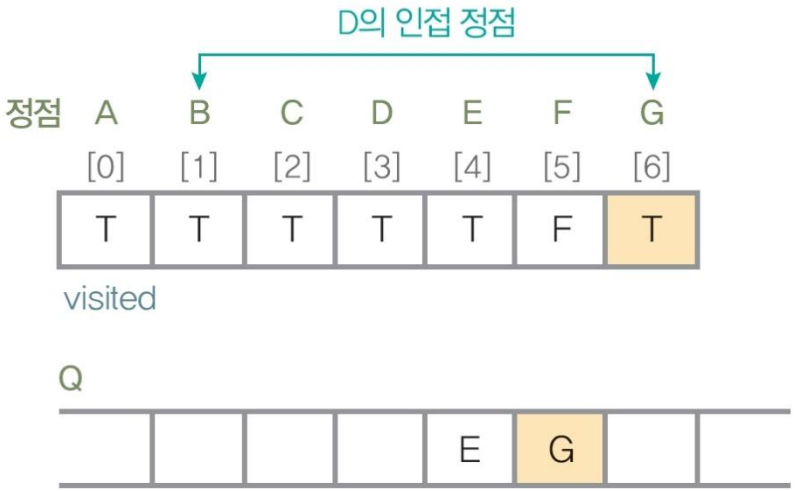
정점	A	B	C	D	E	F	G
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
visited	T	T	T	T	T	F	F



⑦ 정점 D에서 방문하지 않은 인접 정점 G를 방문하고 큐에 enQueue

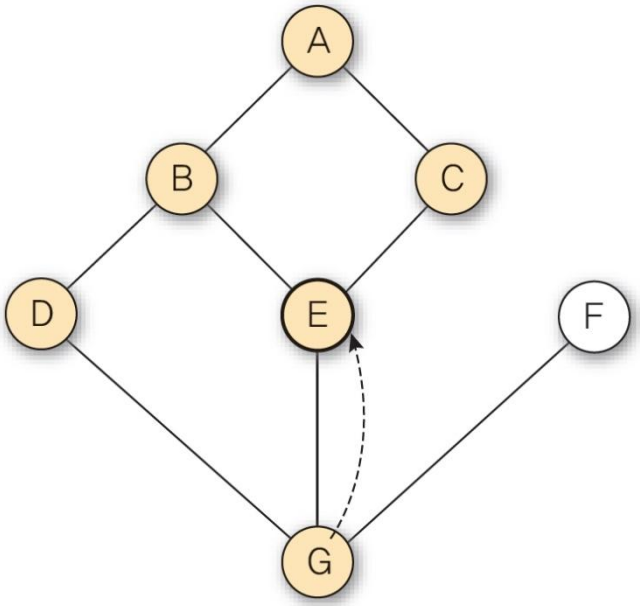
```
visited[(D가 방문하지 않은 인접 정점 G)] ← true;
(D가 방문하지 않은 인접 정점 G) 방문;
enQueue(Q, (D가 방문하지 않은 인접 정점 G));
```

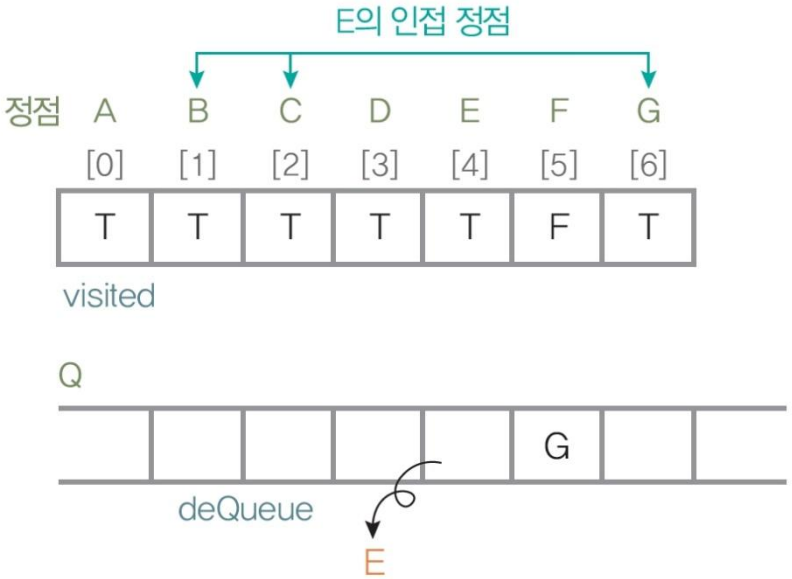




⑧ 정점 D에 대한 인접 정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 E를 받음

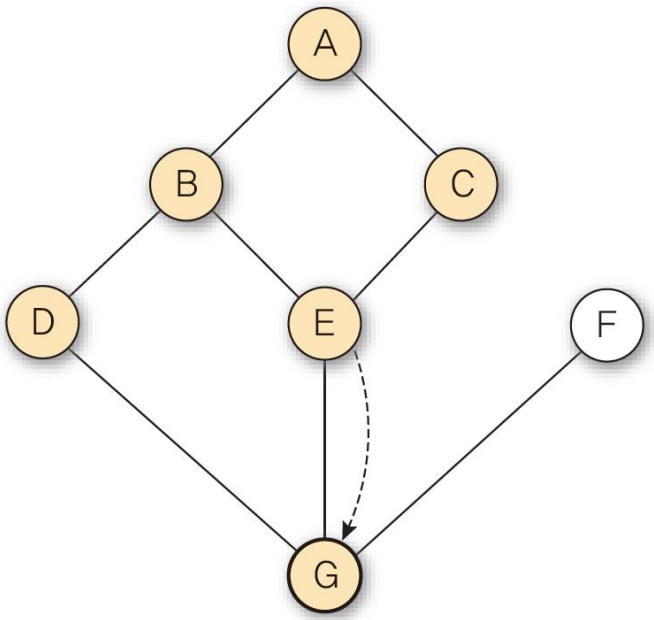
```
v ← deQueue(Q);
```

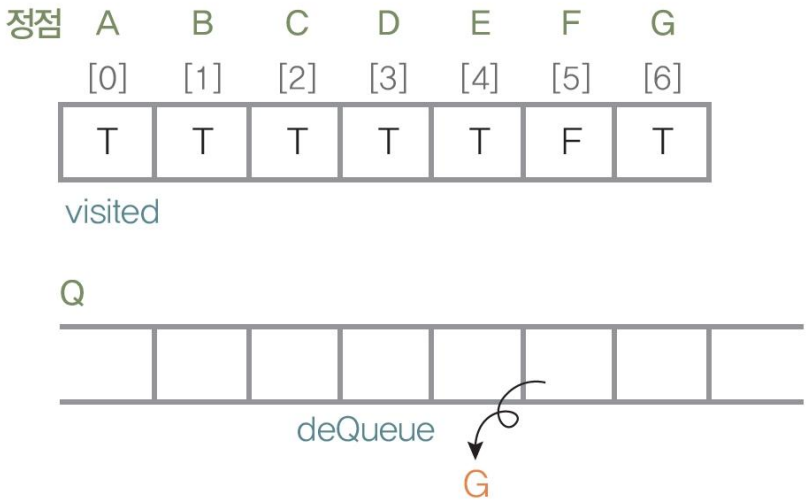




⑨ 정점 E에는 방문하지 않은 인접 정점이 없으므로, 너비 우선 탐색을 계속 할 다음 정점을 찾기 위해 큐를 deQueue하여 G를 받음

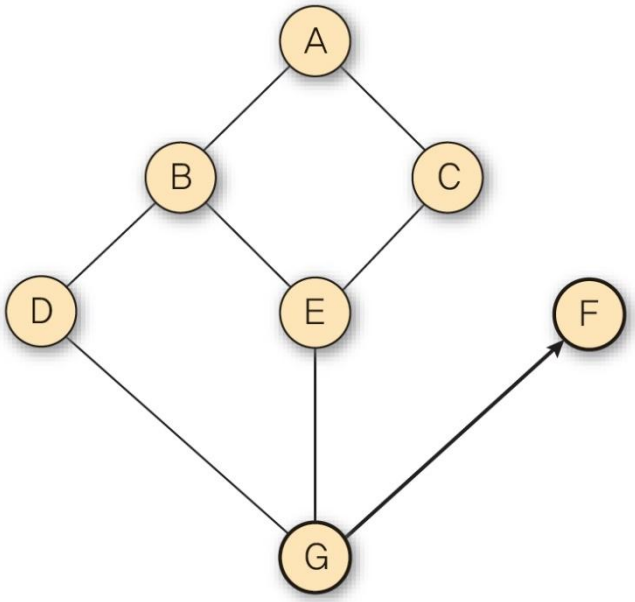
```
v ← deQueue(Q);
```

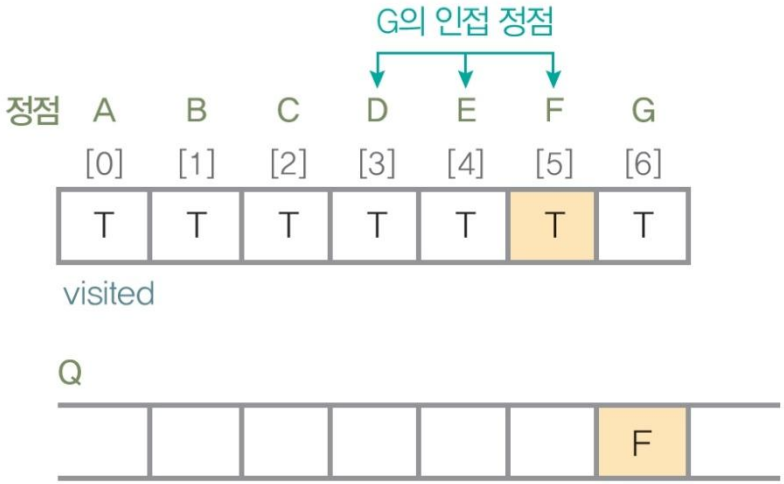




⑩ 정점 G에서 방문하지 않은 인접 정점 F를 방문하고 큐에 enQueue

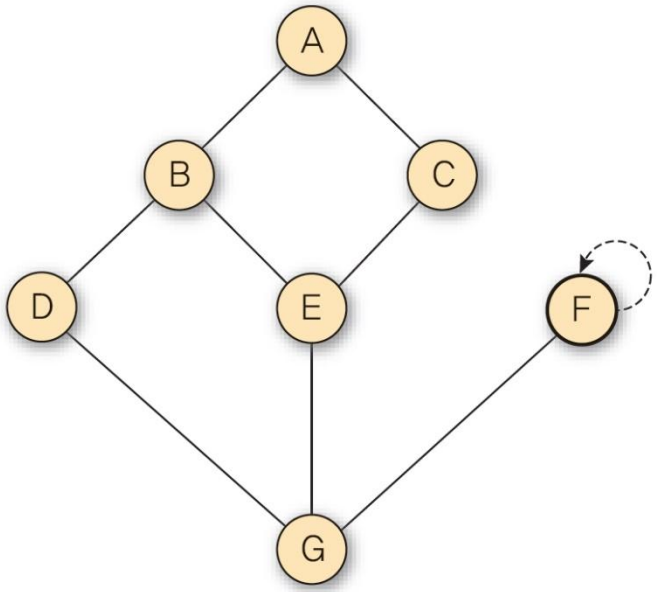
```
visited[(G가 방문하지 않은 인접 정점 F)] ← true;
(G가 방문하지 않은 인접 정점 F) 방문;
enQueue(Q, (G가 방문하지 않은 인접 정점 F));
```

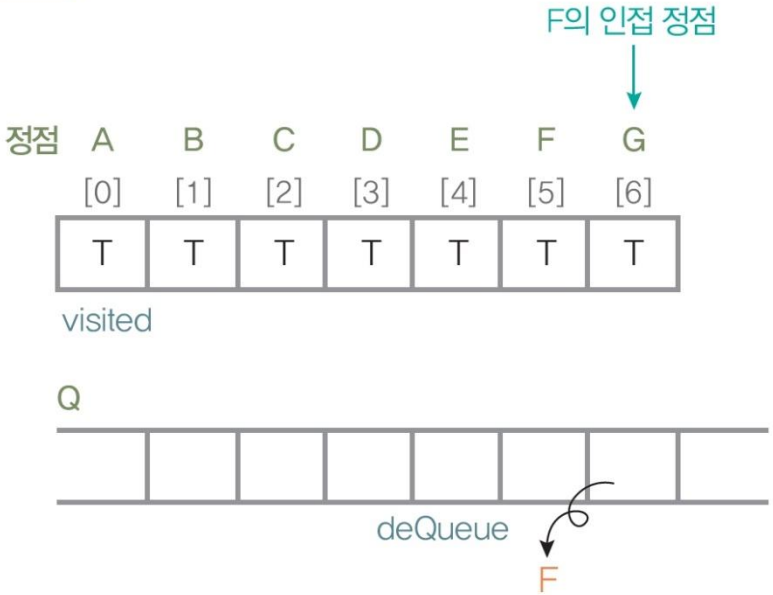




⑪ 정점 G에 대한 인접 정점들을 처리했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue하여 F를 받음

```
v ← deQueue(Q);
```





⑫ 정점 F는 모든 인접 정점을 방문했으므로, 너비 우선 탐색을 계속할 다음 정점을 찾기 위해 큐를 deQueue. 큐가 공백이므로 너비 우선 탐색을 종료

3. 그래프의 순회

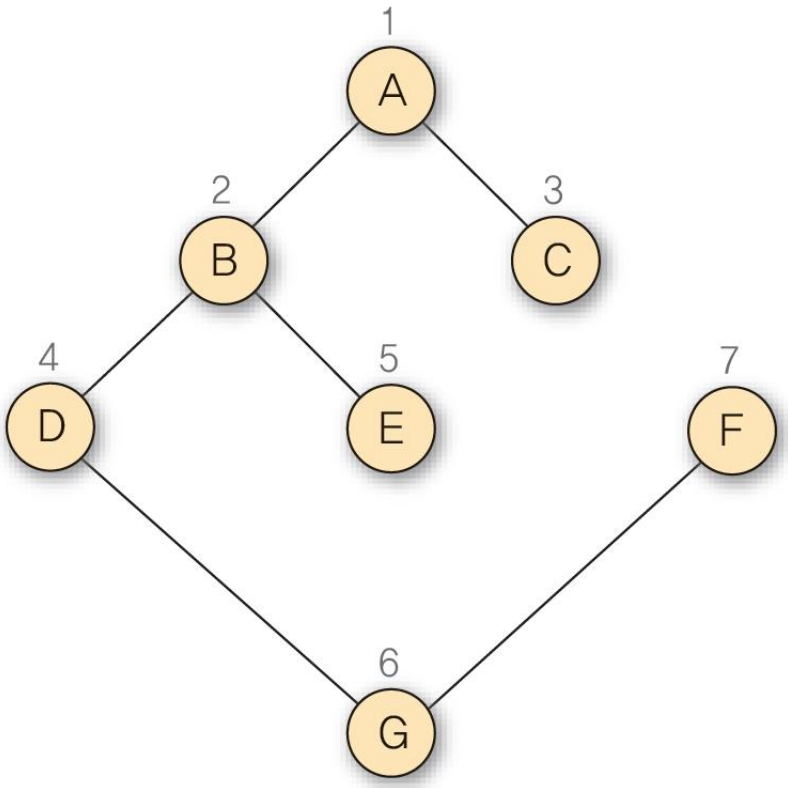


그림 8-16 그래프 G9의 너비 우선 탐색 경로 : A-B-C-D-E-G-F

3. 그래프의 순회

- [예제 8-4] 그래프를 너비 우선 탐색하기 : 교재 469p
- 실행 결과

```
G9의 인접 리스트
        정점 A의 인접 리스트 -> B -> C
        정점 B의 인접 리스트 -> A -> D -> E
        정점 C의 인접 리스트 -> A -> E
        정점 D의 인접 리스트 -> B -> G
        정점 E의 인접 리스트 -> B -> C -> G
        정점 F의 인접 리스트 -> G
        정점 G의 인접 리스트 -> D -> E -> F

///////////////

너비 우선 탐색 >> A B C D E G F
```

예제 8-4 그래프를 너비 우선 탐색하기

[LinkedQueue.h]

```
01    #pragma once
02    typedef int element;     // 연결 큐 원소(element)의 자료형을 int로 수정!!
…     // 나머지는 [예제 6-3]의 [LinkedQueue.h]와 동일
```

[LinkedQueue.c]

```
01    // [예제 6-3]의 [LinkedQueue.c]와 동일. 17행과 68행만 다음과 같이 수정
…        ……
17    // printf(" Linked Queue is empty! "); // 출력문 주석 처리!
…        ……
68    printf("%3d", temp->data); // 출력 형식 수정
…         ……
```

[adjList.h]

```
01    // [예제 8-3]에서 수정한 [adjList.h] 사용
```

[adjList.c]

```
01    // [예제 8-3]에서 수정한 [adjList.c] 사용
```

3. 그래프의 순회

[BFS.h]

```
#pragma once
#include "adjList.h"
void BFS_adjList(graphType* g, int v);
```

[BFS.c]

```
#include "LinkedQueue.h"
#include "adjList.h"
#include "BFS.h"

// 그래프 g에서 정점 v에 대한 너비 우선 탐색 연산                    [알고리즘 8-2] 구현
void BFS_adjList(graphType* g, int v) {
    graphNode* w;
    LQueueType* Q;                  // 큐
    Q = createLinkedQueue();        // 큐 생성
    g->visited[v] = TRUE;           // 시작 정점 v에 대한 배열값을 TRUE로 설정
    printf(" %c", v + 65);
    enLQueue(Q, v);                 // 현재 정점 v를 큐에 enQueue
```

3. 그래프의 순회

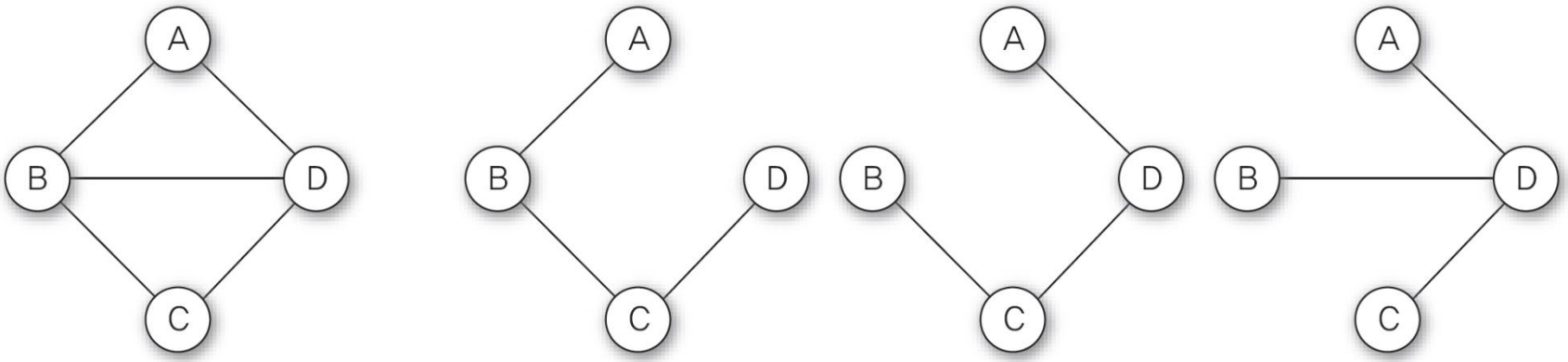
```
    // 큐가 공백인 아닌 동안 너비 우선 탐색 수행
    while (!isLQEmpty(Q)) {
        v = deLQueue(Q);
        // 현재 정점 w를 방문하지 않은 경우
        for (w = g->adjList_H[v]; w; w = w->link)    // 인접 정점이 있는 동안 수행
            if (!g->visited[w->vertex]) {            // 정점 w가 방문하지 않은 정점인 경우
                g->visited[w->vertex] = TRUE;
                printf(" %c", w->vertex + 65);       // 정점 0~6을 A~G로 바꾸어 출력
                enLQueue(Q, w->vertex);
            }
    } // 큐가 공백이면 너비 우선 탐색 종료
}
```

[ex8_4.c]

```
01    // [예제 8-3]에서 작성한 [ex8_3.c]의 02행과 32행을 다음과 같이 수정
02    #include "BFS.h"
…
32    BFS_adjList(G9, 0); // 0번 정점인 정점 A에서 너비 우선 탐색 시작
…
```

4. 신장 트리와 최소 비용 신장 트리

- 신장 트리spanning tree
  - n개의 정점으로 이루어진 무방향 그래프 G에서 n개의 모든 정점과 n-1개의 간선으로 만들어진 트리(트리: 싸이클이 없다)
  - 최소의 간선으로 모든 정점을 연결한다.
  - 연결그래프에서 순회를 하면 n-1 개의 간선을 이동하면서 모든 정점을 방문 => 신장 트리를 생성하게 된다.

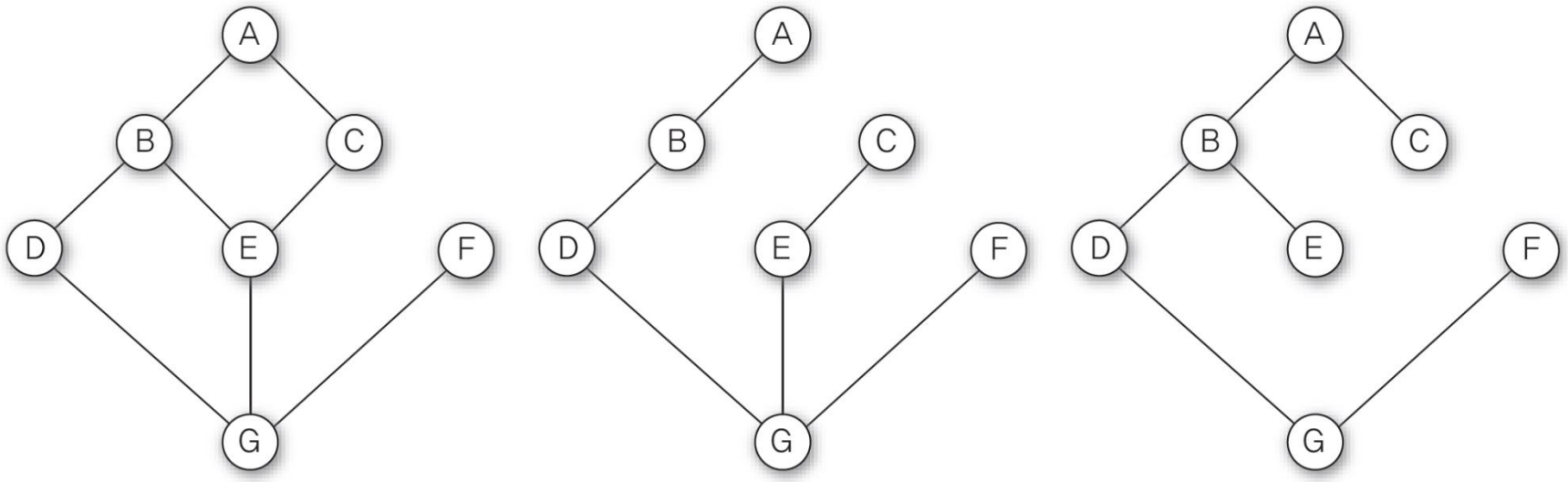


(a) G1 (b) G1의 신장 트리

그림 8-17 그래프 G1과 신장 트리의 예

4. 신장 트리와 최소 비용 신장 트리

- 깊이 우선 신장 트리depth first spanning tree
  - 깊이 우선 탐색을 이용하여 생성된 신장 트리
- 너비 우선 신장 트리breadth 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 트리minimum 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
  - 응용분야 : 최소비용으로 각 노드를 연결하는 방법. 네트워크 설계
    - Computer network, telecommunication network, transportation network
  - 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.

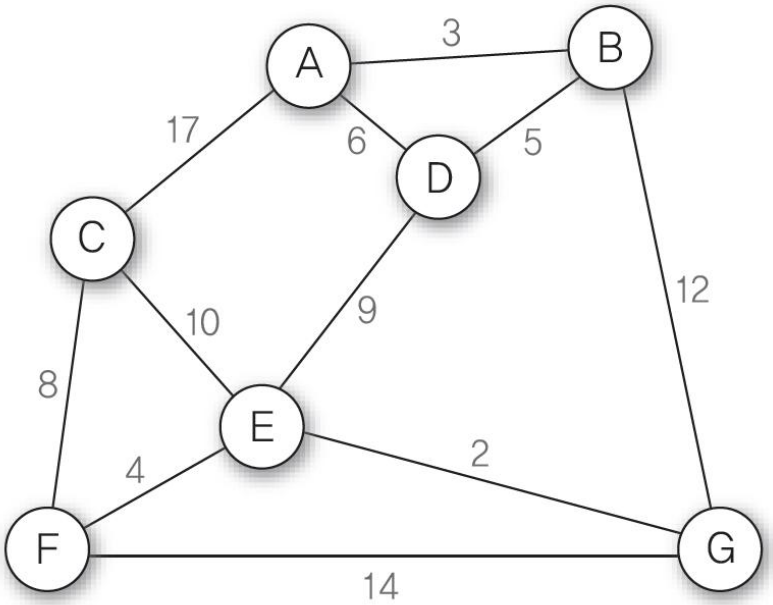
❹ 그래프에 간선이 n–1개만 남으면 최소 비용 신장 트리가 완성된다.

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



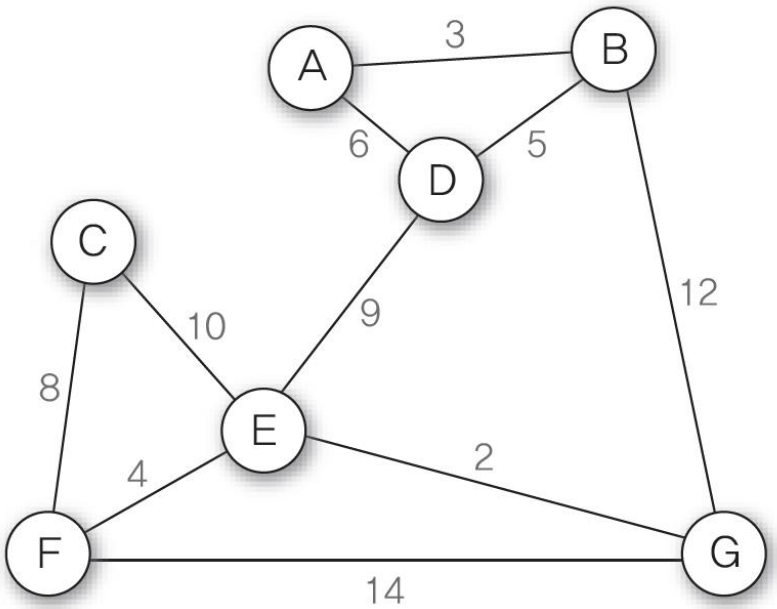
G10

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

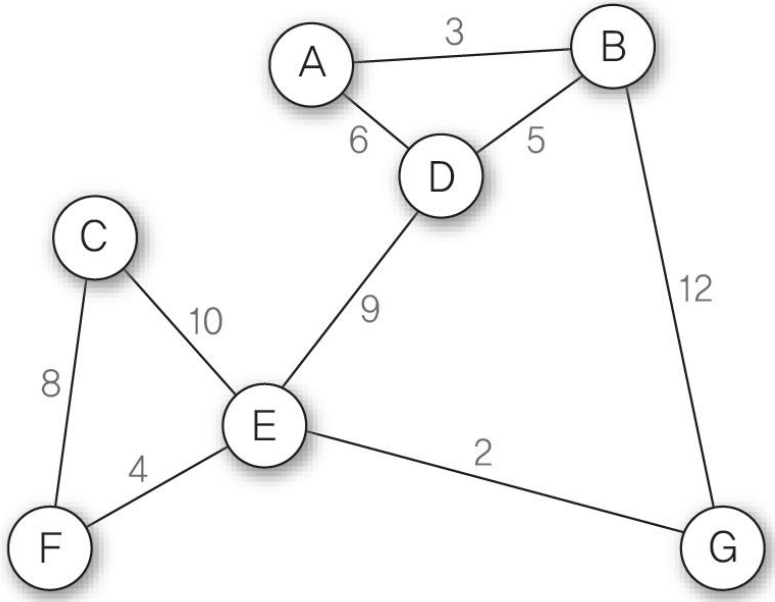


가중치	간선
~~17~~	~~(A, C)~~
14	(F, G)
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

② 남은 간선 중에서 가중치가 가장 높은 간선 (F, G)를 제거

→ 남은 간선 수는 아홉 개

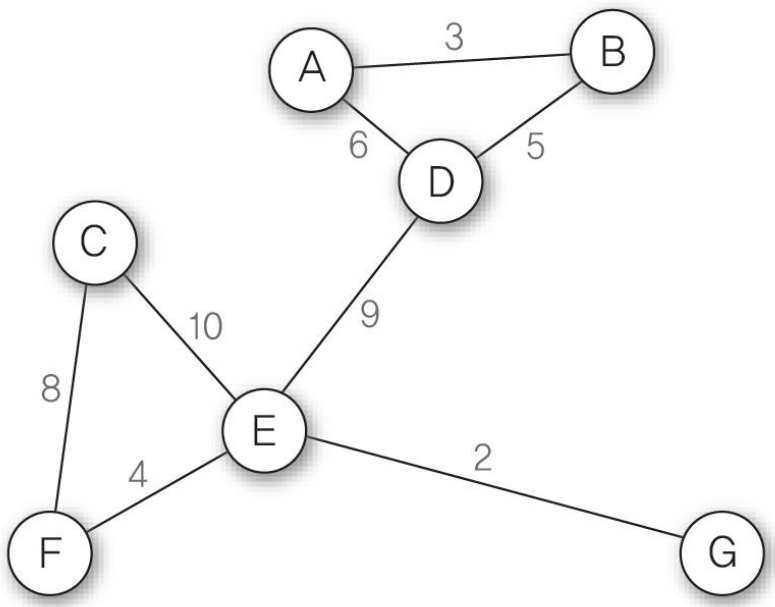


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
12	(B, G)
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

③ 남은 간선 중에서 가중치가 가장 높은 간선 (B, G)를 제거

→ 남은 간선 수는 여덟 개

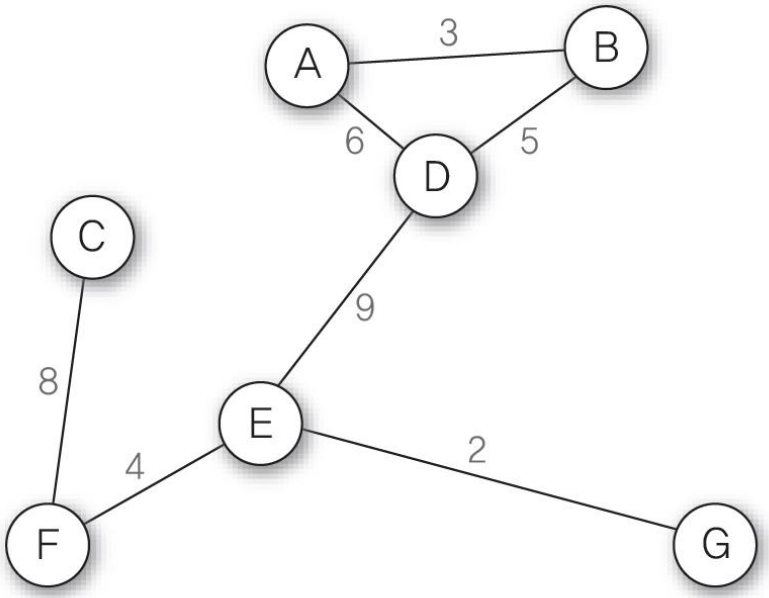


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
10	(C, E)
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

④ 남은 간선 중에서 가중치가 가장 높은 간선 (C, E)를 제거

→ 남은 간선 수는 일곱 개

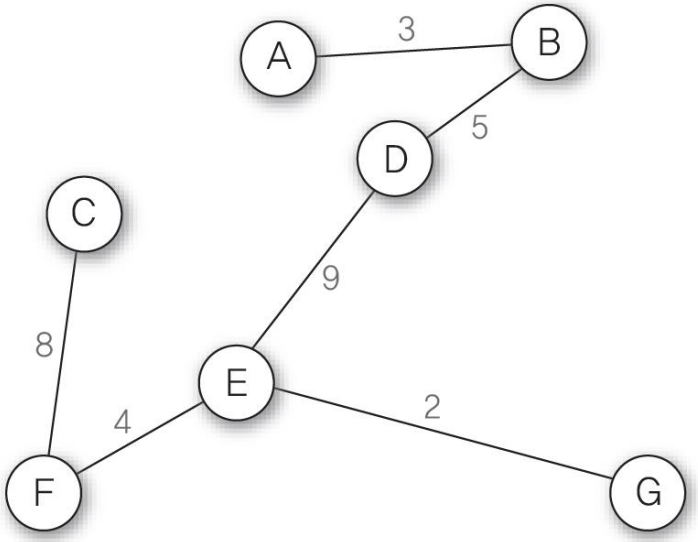


가중치	간선
~~17~~	~~(A, C)~~
~~14~~	~~(F, G)~~
~~12~~	~~(B, G)~~
~~10~~	~~(C, E)~~
9	(D, E)
8	(C, F)
6	(A, D)
5	(B, D)
4	(E, F)
3	(A, B)
2	(E, G)

4. 신장 트리와 최소 비용 신장 트리

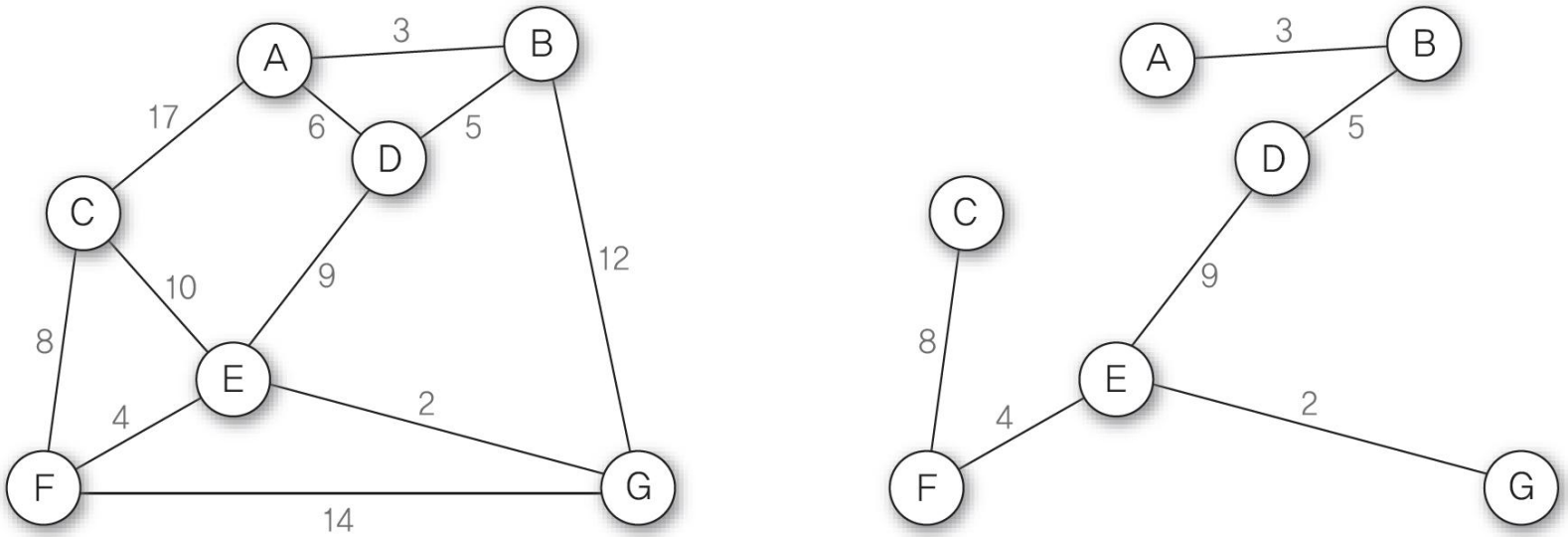
⑤ 남은 간선 중에서 가중치가 가장 높은 간선 (D, E)를 제거하면 그래프가 분리되어 단절 그래프가 되므로 그 다음으로 가중치가 높은 간선 (C, F)를 제거. 그런데 간선 (C, F)를 제거하면 정점 C가 그래프에서 분리되므로 제거 불가능. 따라서 그 다음으로 가중치가 높은 간선 (A, D)를 제거

→ 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)

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

4. 신장 트리와 최소 비용 신장 트리

- 크루스칼 알고리즘 II
  - 가중치가 낮은 간선을 삽입하면서 최소 비용 신장 트리를 만드는 방법
  - 순서

❶ 그래프 G의 모든 간선을 가중치에 따라 오름차순으로 정리한다.

❷ 그래프 G에 가중치가 가장 낮은 간선을 삽입한다. 단, 이때 사이클을 형성하는 간선은 삽입할 수 없으므로 그 다음으로 가중치가 낮은 간선을 삽입한다.

❸ 그래프 G에 간선이 n–1개 삽입될 때까지 ❷를 반복한다.

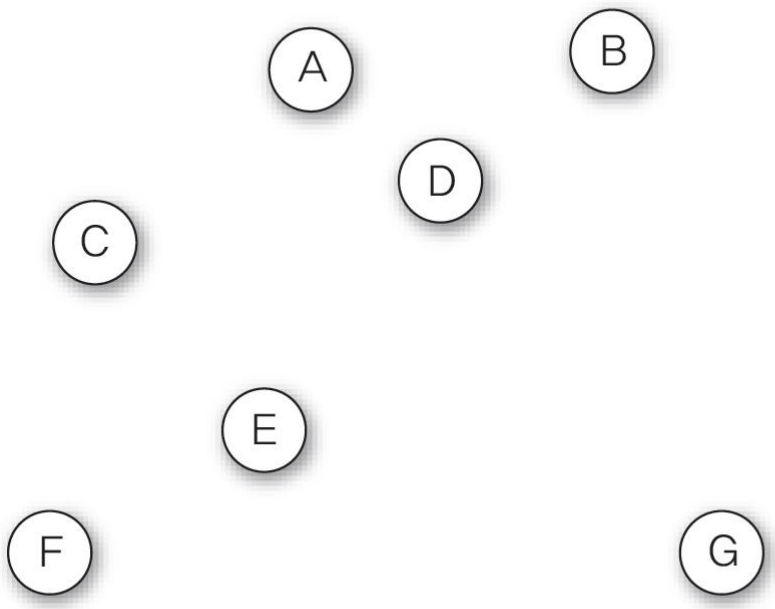
❹ 그래프 G의 간선이 n–1개가 되면 최소 비용 신장 트리가 완성된다.

그림 8-21 크루스칼 알고리즘 II의 순서

4. 신장 트리와 최소 비용 신장 트리

– 크루스칼 알고리즘Ⅱ를 이용하여 G10의 최소 비용 신장 트리 만들기

⓪ 초기 상태 : 그래프 G10의 간선 가중치에 따라 오름차순으로 정렬되어 있고, 그래프 G10은 정점만 있음



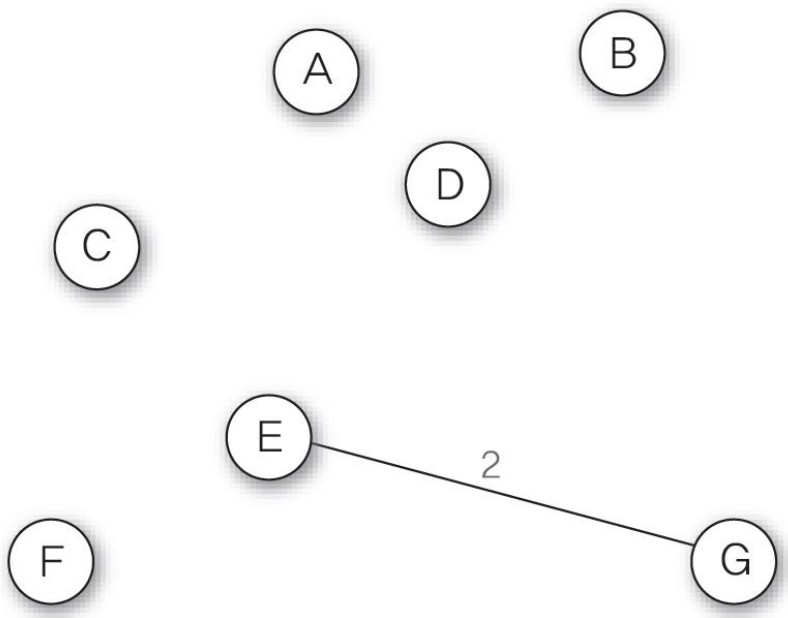
G10

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)

간선 수 : 11개

4. 신장 트리와 최소 비용 신장 트리

① 가중치가 가장 낮은 간선 (E, G)를 삽입 → 삽입한 간선 수는 한 개

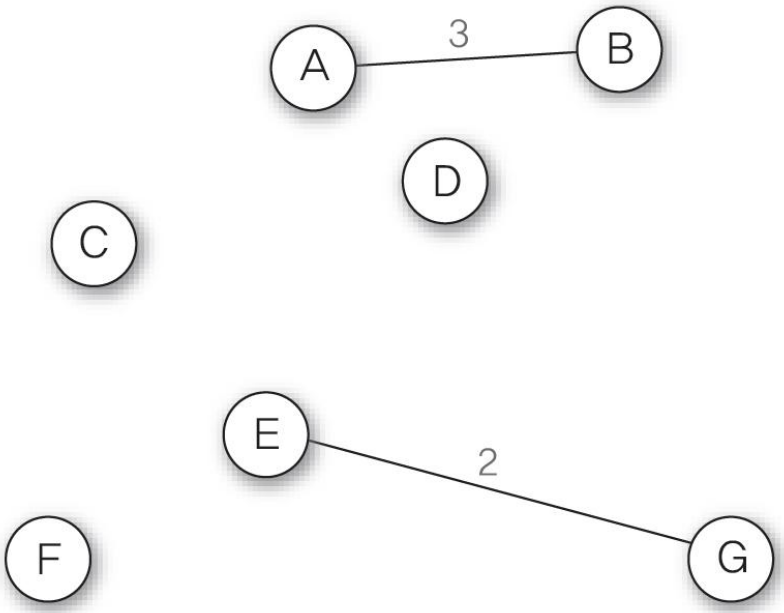


가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

4. 신장 트리와 최소 비용 신장 트리

② 나머지 간선 중에서 가중치가 가장 낮은 간선 (A, B)를 삽입

→ 삽입한 간선 수는 두 개

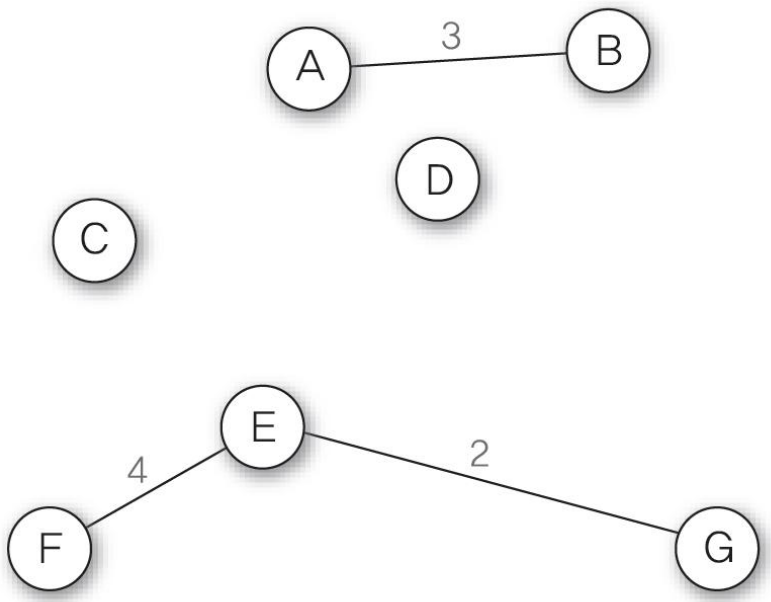


가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

4. 신장 트리와 최소 비용 신장 트리

③ 나머지 간선 중에서 가중치가 가장 낮은 간선 (E, F)를 삽입

→ 삽입한 간선 수는 세 개

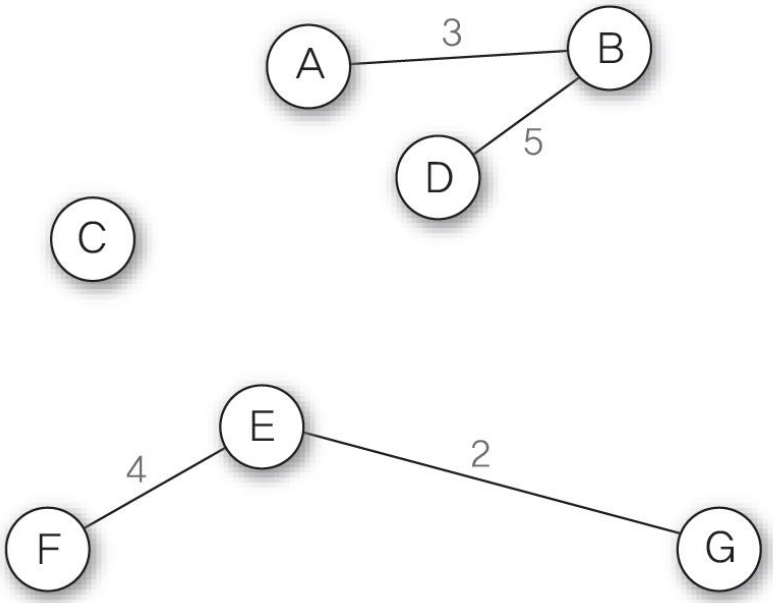


가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

4. 신장 트리와 최소 비용 신장 트리

④ 나머지 간선 중에서 가중치가 가장 낮은 간선 (B, D)를 삽입

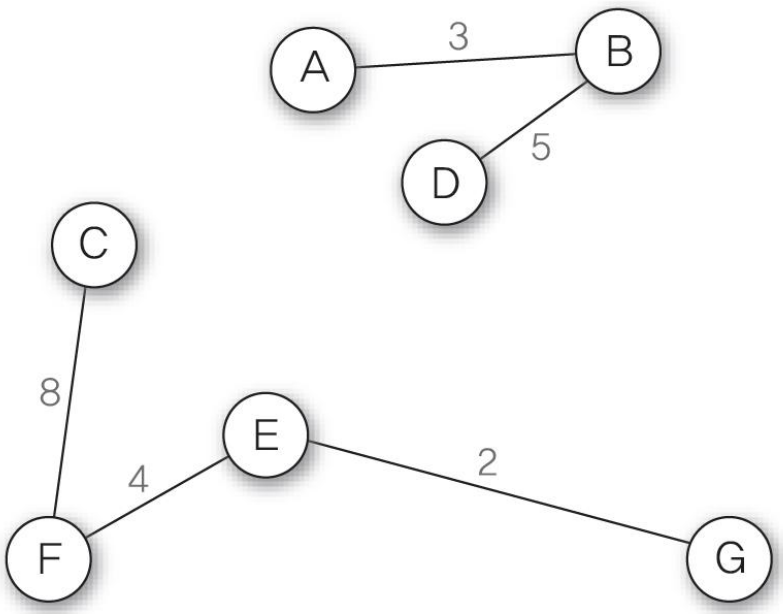
→ 삽입한 간선 수는 네 개



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)

4. 신장 트리와 최소 비용 신장 트리

⑤ 나머지 간선 중에서 가중치가 가장 낮은 간선 (A, D)를 삽입하면 A-B-D-A의 사이클이 생성되므로 삽입할 수 없음. 이런 경우에는 그 다음으로 가중치가 낮은 간선 (C, F)를 삽입 → 삽입한 간선 수는 다섯 개

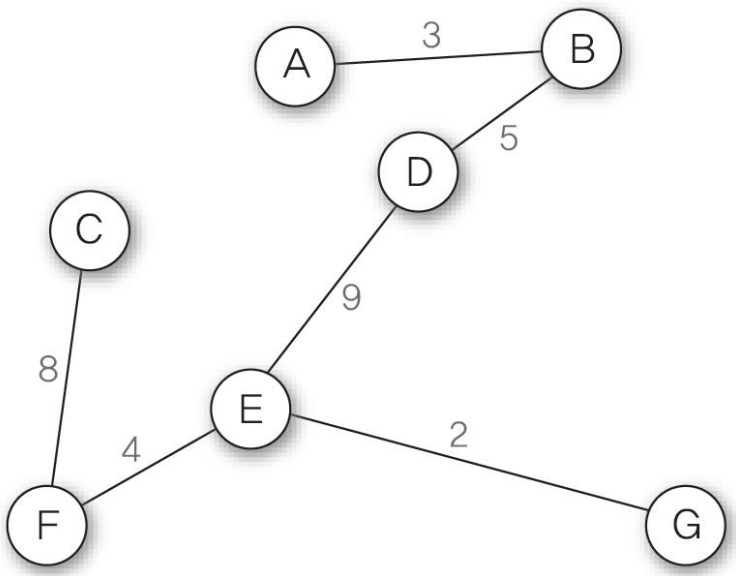


가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

4. 신장 트리와 최소 비용 신장 트리

⑥ 나머지 간선 중에서 가중치가 가장 낮은 간선 (D, E)를 삽입

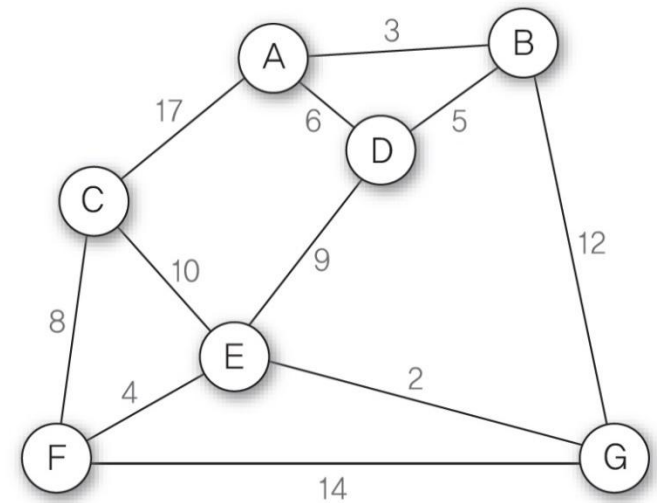
→ 삽입한 간선 수는 여섯 개



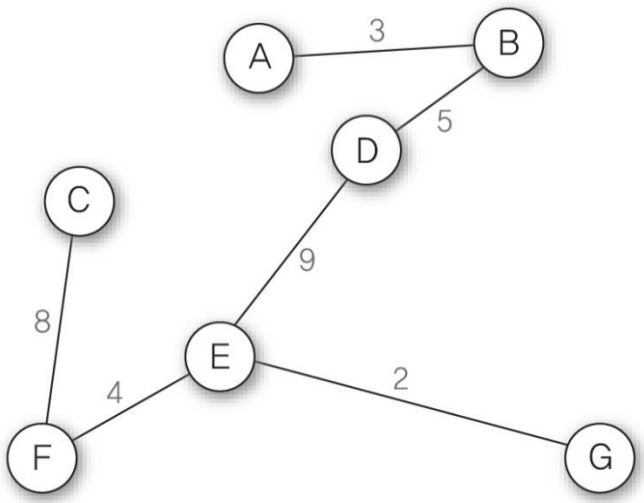
가중치	간선
2	(E, G)
3	(A, B)
4	(E, F)
5	(B, D)
6	(A, D)
8	(C, F)
9	(D, E)
10	(C, E)
12	(B, G)
14	(F, G)
17	(A, C)

⑦ 현재 삽입한 간선이 여섯 개이므로 알고리즘 수행을 종료하면 신장 트리 완성

4. 신장 트리와 최소 비용 신장 트리



(a) G10



(b) 크루스칼 알고리즘 II를 적용한 G10의 최소 비용 신장 트리

그림 8-22 그래프 G10을 크루스칼 알고리즘 II로 완성한 최소 비용 신장 트리

4. 신장 트리와 최소 비용 신장 트리

- 프림 알고리즘
  - 간선을 정렬하지 않고 하나의 정점에서 시작하여 트리를 확장해 나가는 방법
  - 순서

❶ 그래프 G에서 시작 정점을 선택한다.

❷ 선택한 정점에 부속된 모든 간선 중에서 가중치가 가장 낮은 간선을 연결하여 트리를 확장한다.

❸ 이전에 선택한 정점과 새로 확장된 정점에 부속된 모든 간선 중에서 가중치가 가장 낮은 간선을 삽입한다.
단, 사이클을 형성하는 간선은 삽입할 수 없으므로 이런 경우에는 그 다음으로 가중치가 낮은 간선을 선택한다.

❹ 그래프 G에 간선이 n–1개 삽입될 때까지 ❸을 반복한다.

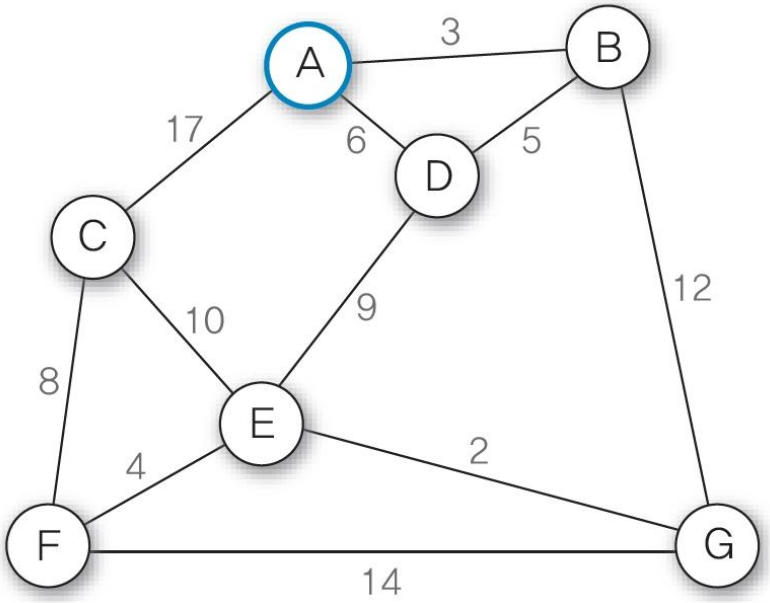
❺ 그래프 G의 간선이 n–1개가 되면 최소 비용 신장 트리가 완성된다.

그림 8–23 프림 알고리즘의 순서

4. 신장 트리와 최소 비용 신장 트리

– 프림 알고리즘을 이용하여 그래프 G10의 최소 비용 신장 트리 만들기

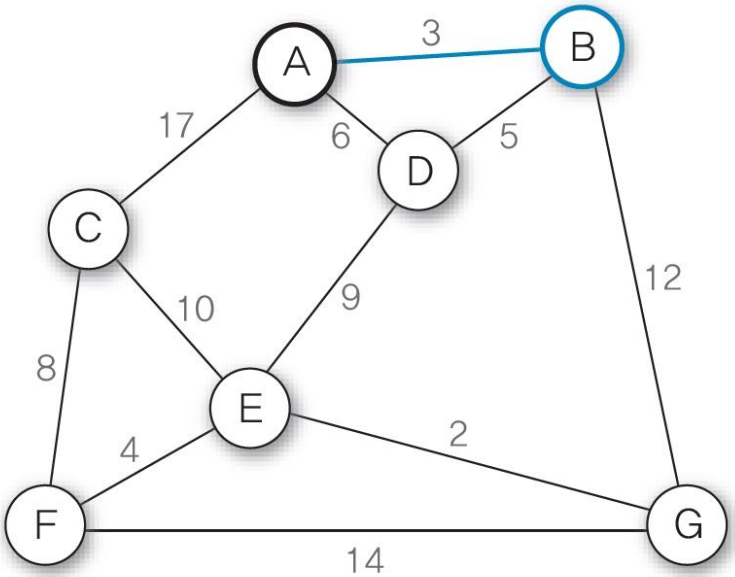
⓪ 초기 상태 : 그래프 G10의 정점 중에서 정점 A를 시작 정점으로 선택



G10

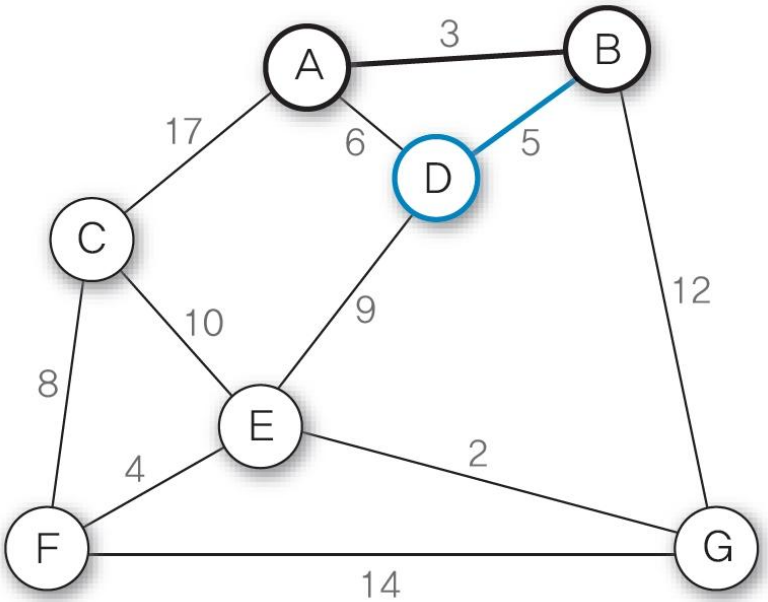
4. 신장 트리와 최소 비용 신장 트리

① 정점 A에 부속된 간선 중에서 가중치가 가장 낮은 간선 (A, B)를 삽입하여 리를 확장 → 삽입한 간선 수는 한 개



4. 신장 트리와 최소 비용 신장 트리

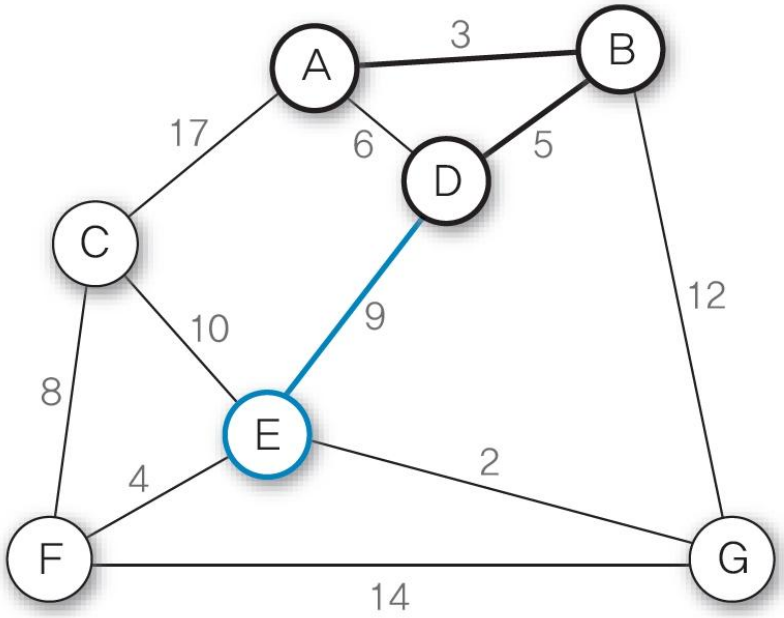
② 현재 확장된 트리의 정점 A, B에 부속된 간선 중에서 가중치가 가장 낮은 간선 (B, D)를 삽입하여 트리를 확장 → 삽입한 간선 수는 두 개



4. 신장 트리와 최소 비용 신장 트리

③ 현재 확장된 트리의 정점 A, B, D에 부속된 간선 중에서 가중치가 가장 낮은 간선인 (A, D)를 삽입하면 A-B-D-A의 사이클이 생성되므로 삽입할 수 없음. 따라서 그 다음으로 가중치가 낮은 간선 (D,E)를 삽입

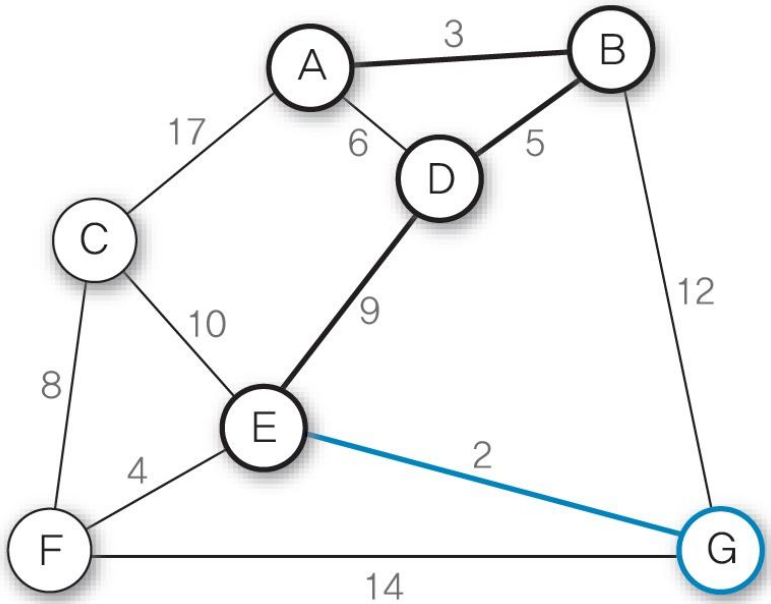
→ 삽입한 간선 수는 세 개, 삽입 불가능한 간선은 (A, D)



4. 신장 트리와 최소 비용 신장 트리

④ 현재 확장된 트리의 정점 A, B, D, E에 부속된 간선 중에서 가중치가 가장 낮은 간선 (E, G)를 삽입하여 트리를 확장

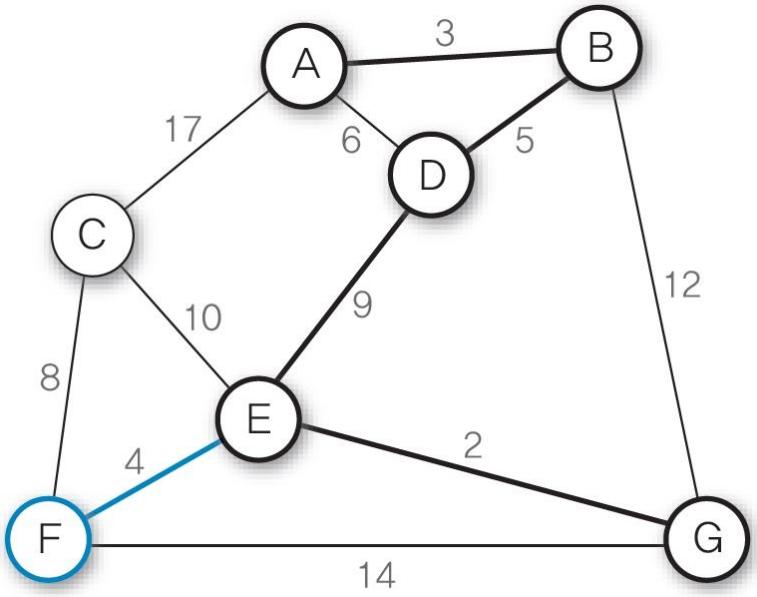
→ 삽입한 간선 수는 네 개, 삽입 불가능한 간선은 (A, D)



4. 신장 트리와 최소 비용 신장 트리

⑤ 현재 확장된 트리의 정점 A, B, D, E, G에 부속된 간선 중에서 가중치가 가장 낮은 간선 (E, F)를 삽입하여 트리를 확장

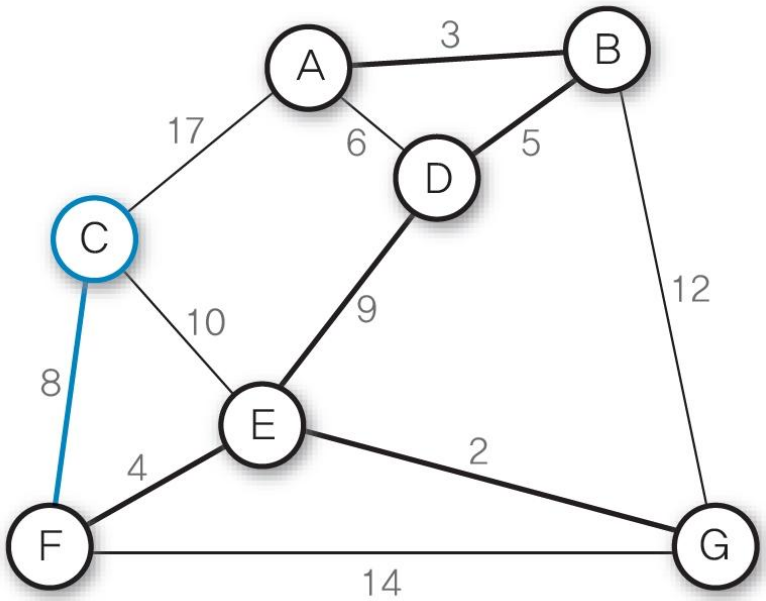
→ 삽입한 간선 수는 다섯 개, 삽입 불가능한 간선은 (A, D)



4. 신장 트리와 최소 비용 신장 트리

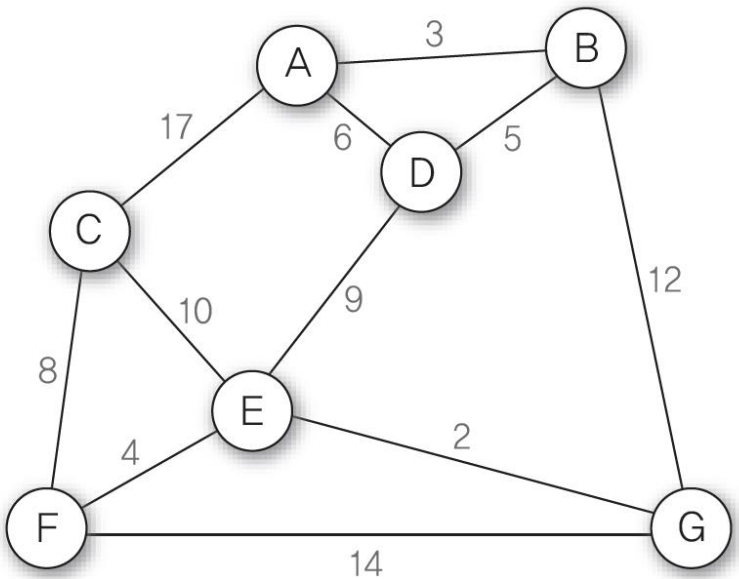
⑥ 현재 확장된 트리의 정점 A, B, D, E, F, G에 부속된 간선 중에서 가중치가 가장 낮은 간선 (C, F)를 삽입하여 트리를 확장

→ 삽입한 간선 수는 여섯 개, 삽입 불가능한 간선은 (A, D)

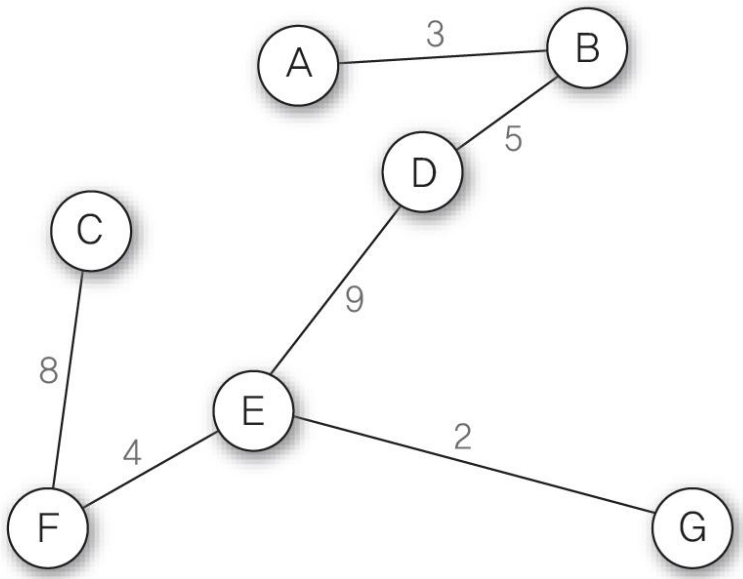


⑦ 현재 삽입한 간선 수가 여섯 개이므로 알고리즘 수행을 종료하면 신장 트리 완성

4. 신장 트리와 최소 비용 신장 트리



(a) G10



(b) 프림 알고리즘을 적용한 G10의 최소 비용 신장 트리

그림 8-24 그래프 G10과 프림 알고리즘에 의해 완성한 최소 비용 신장 트리

4. 신장 트리와 최소 비용 신장 트리

- 최단 경로Shortest Path
  - 신장 트리가 아닌 가중치 그래프, 즉 네트워크Network에서 정점 u와 정점 v를 연결하는 경로 중에서 가중치의 합이 최소인 경로

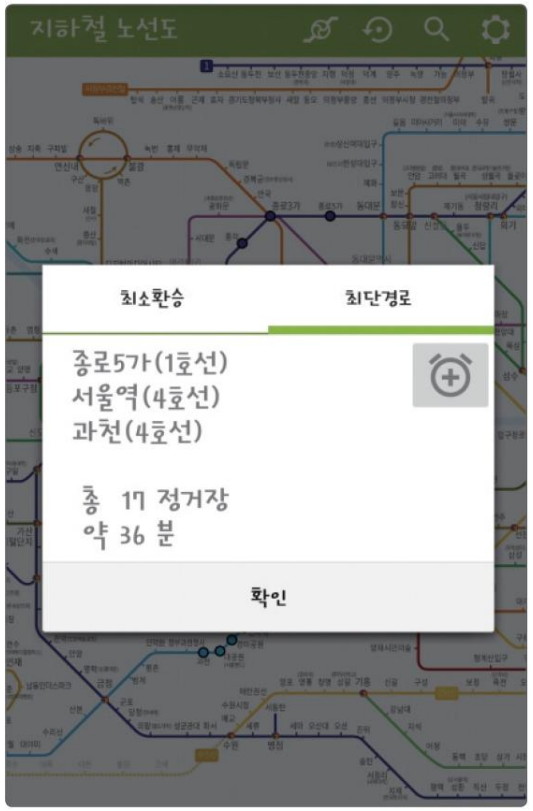
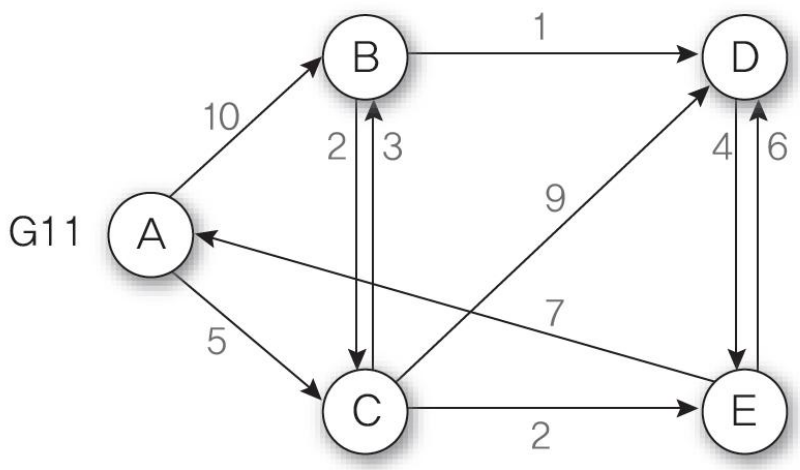


그림 8-25 최단 경로를 응용한 예

4. 신장 트리와 최소 비용 신장 트리

– 가중치 인접 행렬Weight Adjacent Matrix

- 최단 경로를 구하려는 가중치 그래프의 가중치를 저장
- 그래프에서 사용한 인접 행렬과 같은 형태의 2차원 배열, 두 정점 사이에 간선이 없으면 0이 아니라 ∞(무한대)값이 저장되어 있다고 가정.
- 각 정점은 자기 자신과 이어진 간선을 허용하지 않으므로 가중치 인접 행렬에서 대각선 값은 0



Weight 행렬

	A	B	C	D	E
A	0	10	5	∞	∞
B	∞	0	2	1	∞
C	∞	3	0	9	2
D	∞	∞	∞	0	4
E	7	∞	∞	6	0

그림 8–26 가중치 방향 그래프와 가중치 인접 행렬의 예

4. 신장 트리와 최소 비용 신장 트리

– 다익스트라Dijkstra 최단 경로 알고리즘

- 하나의 시작 정점에서 다른 모든 정점까지의 최단 경로를 구함
- 단일점에서의 최단 경로One to All Shortest Path 알고리즘 중 가장 많이 사용.
- 무방향 그래프나 방향 그래프에 모두 적용 가능

- 순서

시작정점을

❶ **경로 길이를 저장할 배열 distance 준비 :** 시작 정점으로부터 각 정점에 이르는 경로의 길이를 저장하기 위한 배열 distance를 ∞(무한대)로 초기화한다.

❷ **시작 정점 초기화 :** 시작 정점의 distance를 0으로 초기화하고 집합 S에 추가한다.

❸ **최단 거리 수정 :** 집합 S에 속하지 않은 정점 중에서 distance가 최소인 정점 u를 찾아 집합 S에 추가한다. 새로운 정점 u가 추가되면, u에 인접하고 집합 S에 포함되지 않은 정점 w의 distance값을 다음 식에 따라 수정한다. 집합 S에 모든 정점이 추가될 때까지 ❸을 반복한다.

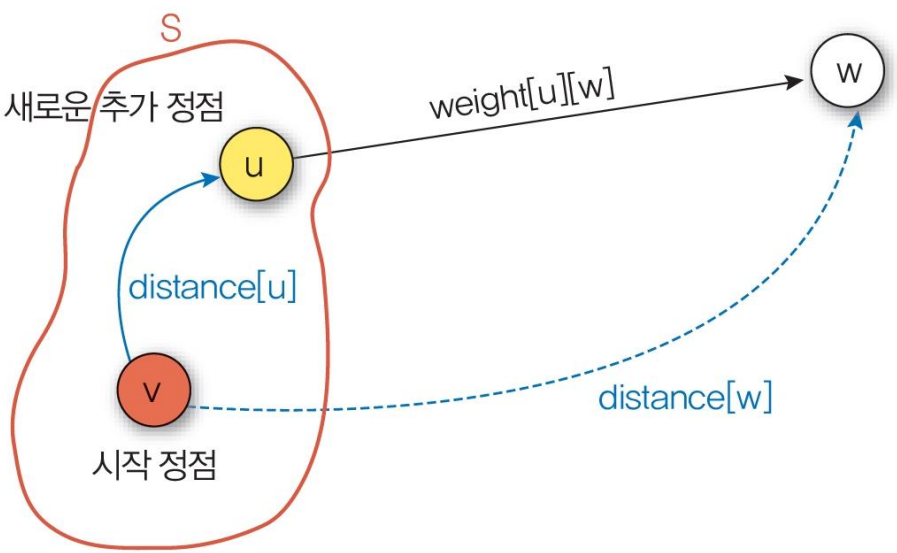
$$\text{distance}[w] \leftarrow \min(\text{distance}[w],\ \text{distance}[u]+\text{weight}[u][w])$$

그림 8-28 다익스트라 최단 경로 알고리즘에서 최단 경로 찾기

4. 신장 트리와 최소 비용 신장 트리

- 시작정점 v 에서 w 까지의 경로비용를 알고 있을 때
- 시작 정점 v에서 정점u까지의 최단 경로 distance[u]를 구해 정점 u가 집합 S에 추가되면 새로 추가된 정점 u에 의해 단축되는 경로가 있는지를 확인함.
  - 즉, 현재 알고 있는 distance[w]를 새로 추가된 정점 u를 거쳐서 가는 distance[u]+weight[u][w]를 계산한 경로와 비교해 경로가 단축되면 distance[w]를 단축된 경로값으로 수정함으로써 최단 경로를 찾음

$$\text{distance}[w] \leftarrow \mathbf{min}(\text{distance}[w],\ \text{distance}[u]+\text{weight}[u][w])$$



- S : 추가된 정점을 관리하는 집합
- weight : 가중치 인접 행렬

그림 8-27 다익스트라 최단 경로 알고리즘에서 최단 경로 구하기의 기본 개념

알고리즘 8-3 다익스트라 최단 경로

```
Dijkstra_shortestPath(G,v)
    S ← {v};
    for (i ← 0; i∈V(G); i ← i+1) do
        distance[i] ← weight[v][i];
    for(i ← 0; S != V(G); i ← i+1 ) do {  // 집합 S와 정점의 집합 V(G)가 같지 않은 동안
        u ← S에 속하지 않은 정점 중에서 distance[]가 최소인 정점;
        S ← S ∪ {u};
        for (w ← 0; w∈V(G); w ← w+1) do
            if(w ∉ S) then
                distance[w] ← min(distance[w], distance[u]+weight[u][w]);
    }
end Dijkstra_shortestPath()
```

i 는 vertex 번호.
시작 정점 v 에서 다른 모든 정점들 까지의 거리를 distance 에 저장.

S 에 모든 정점이 포함될때 까지

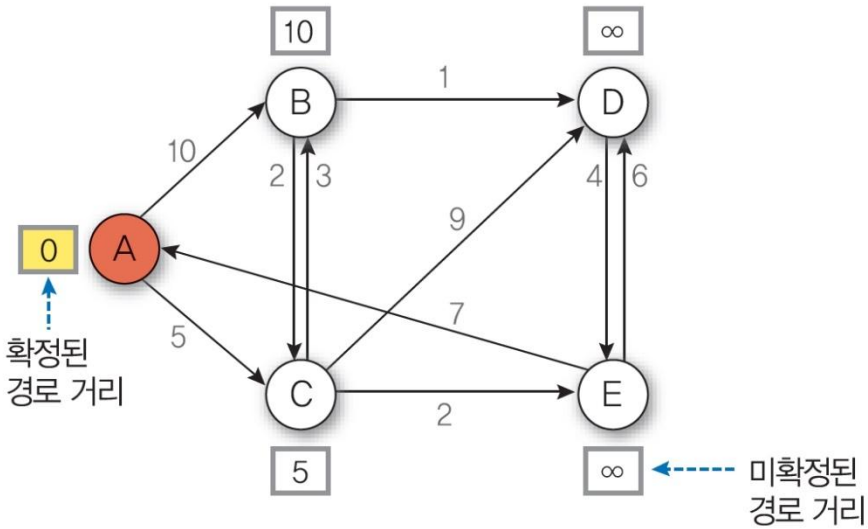
4. 신장 트리와 최소 비용 신장 트리

– [그림 8-26]의 그래프 G11에 [알고리즘 8-3]을 적용해 최단 경로를 구하는 과정

- 시작 정점은 A,
- 각 정점 옆에 붙은 숫자는 시작 정점 A에서 각 정점까지 도달하는 거리를 의미.
- 집합 S에 포함되는 추가 경로 거리는 노란색, 집합 S의 정점은 빨간색, 현재 단계에서 새로 추가된 정점과 그로 인해 수정된 distance값은 파란색으로 표시

4. 신장 트리와 최소 비용 신장 트리

⓪ 초기 상태 : 시작 정점 A를 집합 S의 초깃값으로 설정, 시작 정점 A에서의 인접 정점에 대한 가중치값을 인접 행렬 weight에서 구해 배열 distance의 초깃값으로 설정



	A	B	C	D	E
A	0	10	5	∞	∞
B	∞	0	2	1	∞
C	∞	3	0	9	2
D	∞	∞	∞	0	4
E	7	∞	∞	6	0

S = {A}

	A	B	C	D	E
distance	0	10	5	∞	∞

집합 S에 포함되지 않은 정점 중 최솟값

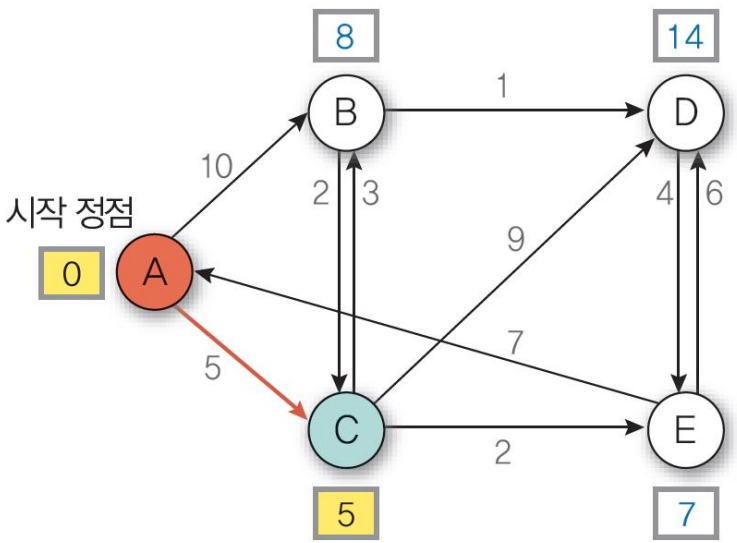
4. 신장 트리와 최소 비용 신장 트리

① 시작 정점 A의 진출 간선과 연결된 인접 정점 B와 C 중에서 최단 경로인 C를 집합 S에 추가. 새로 추가된 정점 C에 의해 단축되는 경로가 있는지 확인하여 배열 distance값을 수정

distance[B] ← min(distance[B], distance[C]+weight[C][B]) = min(10, 5+3) = 8

distance[D] ← min(distance[D], distance[C]+weight[C][D]) = min(∞, 5+9) = 14

distance[E] ← min(distance[E], distance[C]+weight[C][E]) = min(∞, 5+2) = 7



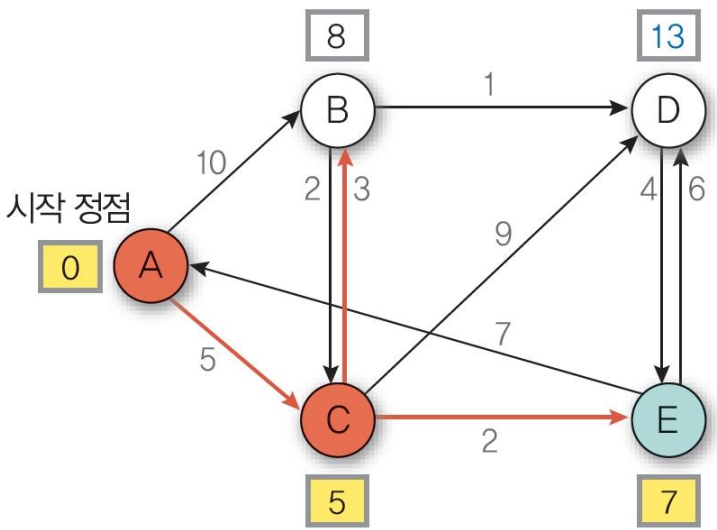
S = {A, C}

	A	B	C	D	E
distance	0	8	5	14	7

4. 신장 트리와 최소 비용 신장 트리

② 집합 S에 포함되지 않은 정점 중에서 최단 경로인 E를 집합 S에 추가. 새로 추가된 정점 E에 의해 단축되는 경로가 있는지 확인하여 배열 distance값을 수정

distance[D] ← min(distance[D], distance[E]+weight[E][D]) = min(14, 7+6) = 13



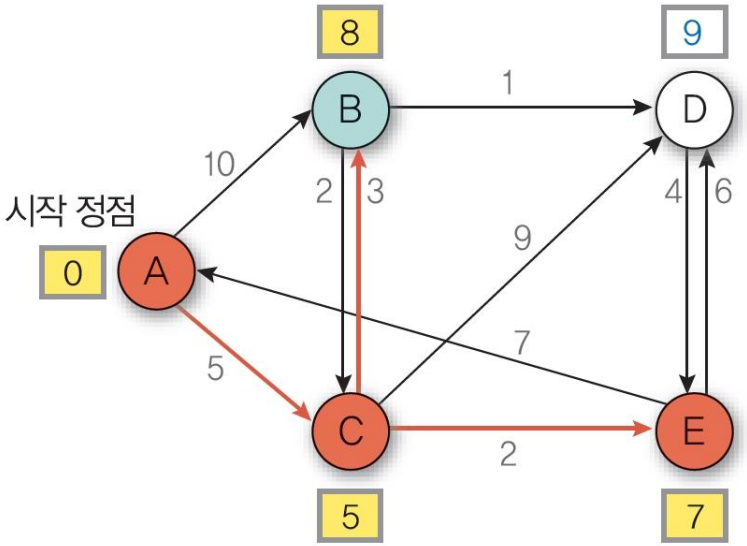
S = {A, C, E}

	A	B	C	D	E
distance	0	8	5	13	7

최소값

③ 집합 S에 포함되지 않은 정점 중에서 최단 경로인 B를 집합 S에 추가. 새로 추가된 정점 B에 의해 단축되는 경로가 있는지 확인하여 배열 distance값을 수정

$$\text{distance}[D] \leftarrow \min(\text{distance}[D],\ \text{distance}[B]+\text{weight}[B][D]) = \min(13,\ 8+1) = 9$$

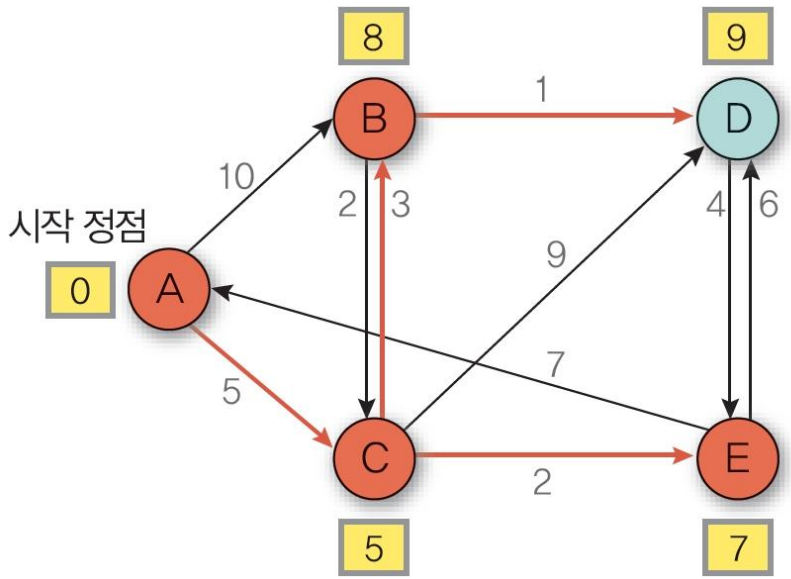


S = {A, B, C, E}

	A	B	C	D	E
distance	0	8	5	9	7

4. 신장 트리와 최소 비용 신장 트리

④ 집합 S에 포함되지 않은 정점 중에서 최단 경로인 D를 집합 S에 추가.
이로써 모든 정점이 집합 S에 추가되었으므로 최단 경로가 완성됨



S = {A, B, C, D, E}

	A	B	C	D	E
distance	0	8	5	9	7