계산이론

2022년 1학기

이은주

1장 계산 이론 개요

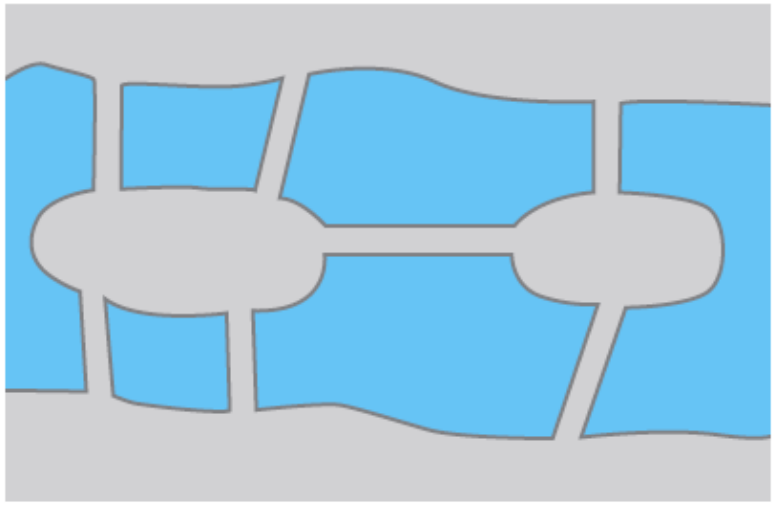
수학적 개념 및 계산이론 기초의 이해 : 그래프와 트리

그래프

- 그래프의 용어, 표현
- 그래프의 응용과 탐색
- 그래프 색칠문제

그래프 이론

- 18세기 스위스 출신의 저명한 수학자 오일러(Leonhard Euler, 1707~1783)에 의해 그래프 이론이 본격적으로 시작됨
- 그래프 이론의 대표적인 예인 쾨니히스베르크(Königsberg) 다리 문제는 두 개의 섬과 강둑 사이를 연결하는 7개의 다리가 있을 때 각 다리를 꼭 한 번씩만 건너는 경로를 찾는 문제임



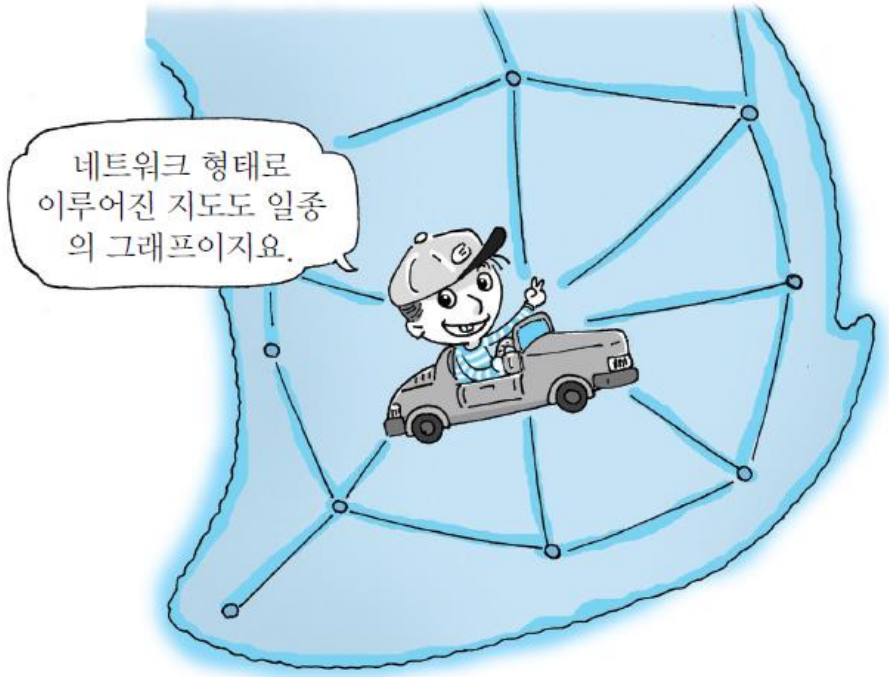
〈그림 7.1〉 쾨니히스베르크 다리 문제

여기서 잠깐!!

그래프의 응용 분야

- 통신 네트워크
- 논리 회로의 설계 및 분석
- 최단 경로 찾기나 최단 거리 순회 문제
- 시스템의 흐름도나 컴파일러에서의 최적화
- 그래프 탐색, 정렬
- 알고리즘
- 전기회로망 설계와 응용
- 순서도를 통한 작업 계획의 분석
- 분자구조식 설계와 화학결합의 표시
- 유한 상태 기계
- 유전학, 언어학, 사회과학 등

정의 7-1 그래프(graph) $G = (V, E)$는 유한한 개수의 정점(vertex) 또는 노드(node)들의 집합인 V와 연결선(edge) 또는 에지라고 불리는 정점들의 쌍들의 집합인 E로 이루어진다.

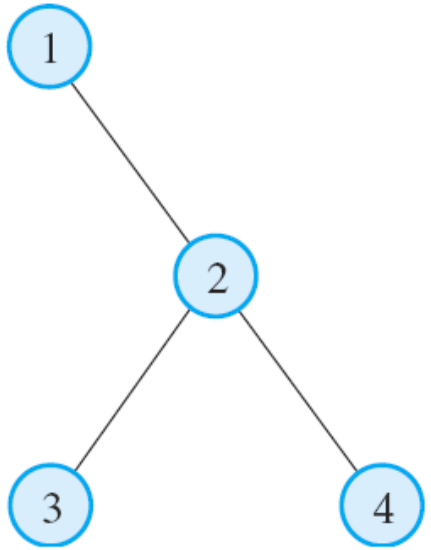


그래프의 2가지 종류

- 방향 그래프(directed graph 또는 digraph)
  - ✓ 방향이 있는 그래프임
  - ✓ 연결선을 화살표로 표시하여 방향을 나타내는 그래프임
- 방향이 없는 그래프(undirected graph)
  - ✓ 방향이 없는 그래프임
  - ✓ 그래프의 특수한 형태이므로 특별한 언급이 없는 한 그래프는 방향이 없는 그래프를 의미함

그래프의 간단한 예

V={1, 2, 3, 4}이며 E={(1, 2), (2, 3), (2, 4)}



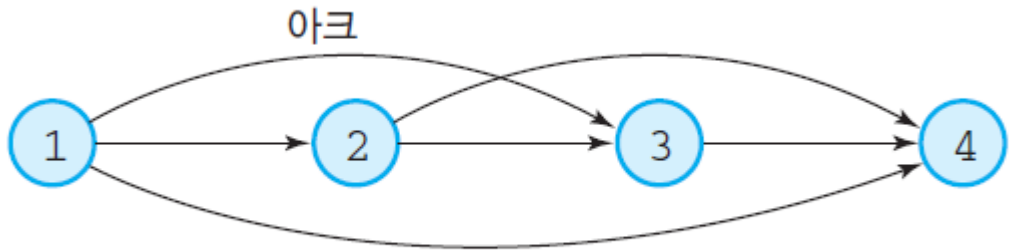
경로(path)

- 모든 $1 \le i < k$에 대해 연결선 (v_i, v_{i+1})이 존재할 때, 정점들의 열(sequence) $v_1, v_2, v_3, \cdots, v_k$라고 함
- $k \ge 1$이며 이 경로의 길이는 $k-1$임

사이클(cycle)

$v_1 = v_k$ $(k \ne 1)$이면 이러한 경로를 사이클이라고 함

정의 7-2 방향 그래프(directed graph) 또는 다이그래프(digraph)는 $G = (V, E)$로 표시된다. 여기서 V는 정점들의 집합이며, E는 정점들의 순서화된 쌍인 아크(arc) 또는 연결선들의 집합이다. 정점 v에서 정점 w로 가는 아크는 $v \rightarrow w$로 표시한다.



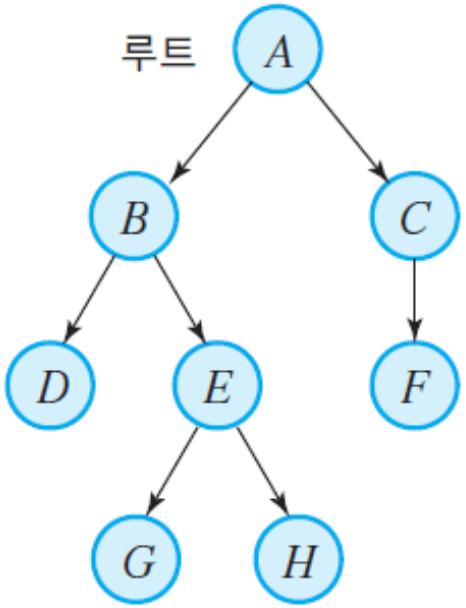
〈그림 7.3〉 방향 그래프($\{1, 2, 3, 4\}, \{v \rightarrow w \mid v < w\}$)

방향 그래프의 예

- 1→2→3→4는 정점 1에서 정점 4로 가는 경로임
- v → w인 아크에 대하여 v를 w의 선행자(predecessor)라 하며 w를 v의 후속자(successor)라고 함

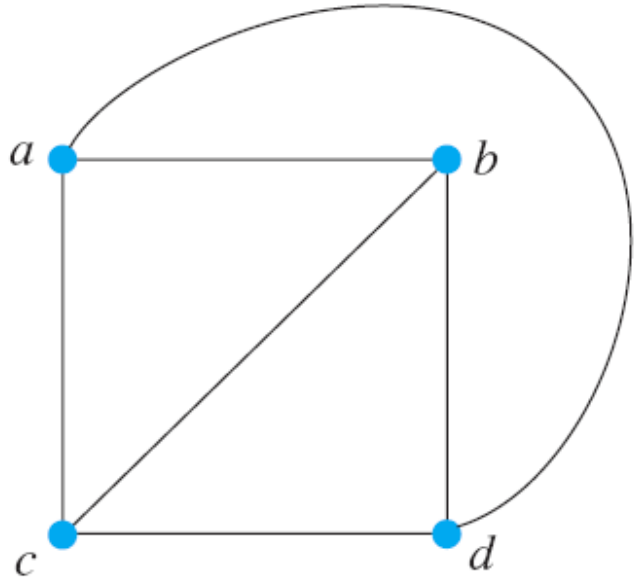
트리(Tree)

- 사이클(cycle)이 존재하지 않는 그래프임
- 루트(root)라 불리는 특별한 노드가 한 개 존재하고, 루트로부터 다른 모든 노드로 가는 경로가 항상 유일하게 존재함
- 루트로 들어오는 연결선이 없으므로 루트는 모든 트리의 출발점이 됨



정의 7-3 단순 그래프(simple graph)

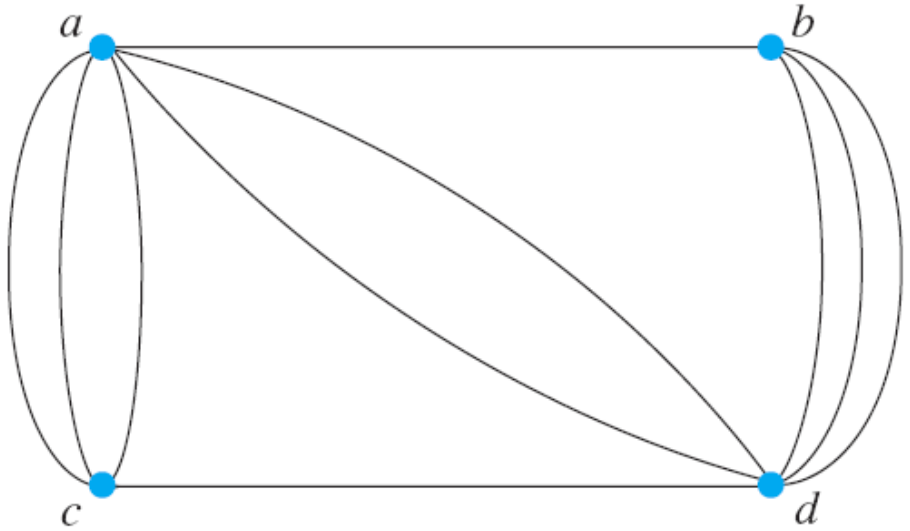
한 쌍의 정점 사이에 많아도 하나의 연결선으로 이루어진, 우리가 통상 다루는 그래프로서 자기 자신으로의 연결선이 없는 그래프이다.



〈그림 7.5〉 단순 그래프의 예

정의 7-4 멀티 그래프(multigraph)

단순 그래프의 확장으로서 한 쌍의 꼭지점 사이에 연결선의 개수의 제한이 없는 일반적인 그래프이다.



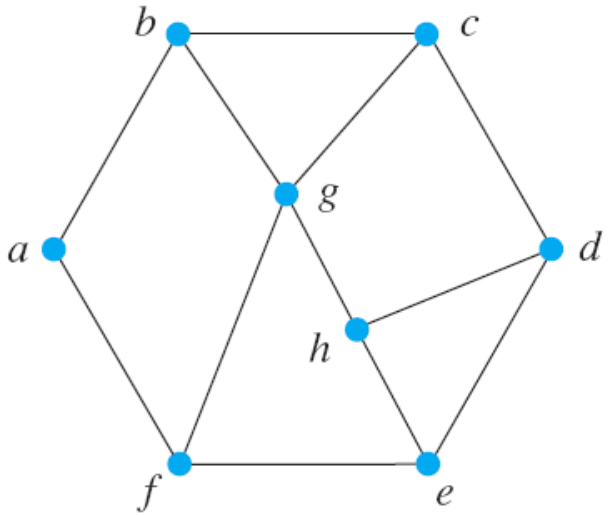
〈그림 7.6〉 멀티 그래프의 예

정의 7-5 그래프 $G = (V, E)$에서 순서화된 쌍 E를 그래프의 연결선(edge)이라고 한다. $(u, v) \in E$일 때 u와 v를 연결하는 연결선 e는 u와 v에 '접했다'(incident)라 하고 u와 v를 서로 '인접했다'(adjacent)라고 한다.

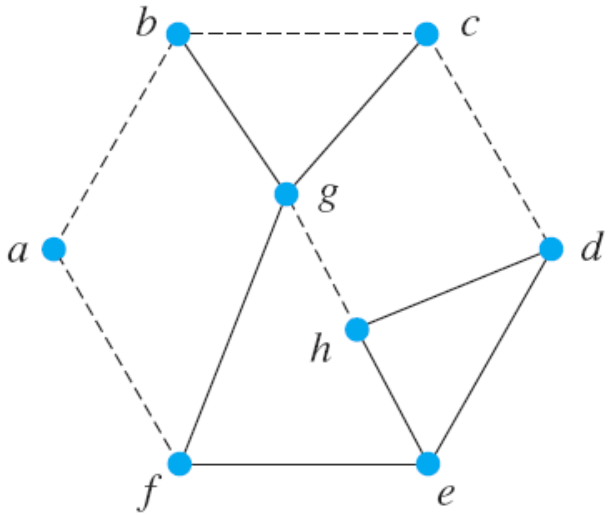
정의 7-6 $G = (V, E)$가 그래프이고 u, v가 꼭지점이라고 할 때 v의 차수(degree) $d(v)$는 v에 인접하는 연결선들의 개수를 말한다.

정의 7-7 두 개의 그래프 $G=(V, E)$, $G'=(V', E')$에서 $V' \subseteq V$, $E' \subseteq E$일 때 그래프 $G'=(V', E')$를 G의 부분 그래프(subgraph)라고 한다. 이때 $V=V'$이고 $E' \subset E$이면 G'는 G의 생성 부분 그래프(spanning subgraph)라고 한다.

예제 7-1 다음 그림의 (2)는 $V' \subseteq V$, $E' \subseteq E$이므로 (1)의 부분 그래프이다. 또한 (2)는 $V=V'$이고 $E' \subset E$이므로 (1)의 생성 부분 그래프도 된다.



(1)



(2)

정의 7-8 그래프에서의 경로는 경로, 단순 경로, 기본 경로, 사이클, 단순 사이클, 기본 사이클 등으로 구분된다.

(1) 경로(path)

- 그래프에서의 경로란 꼭지점의 열 $v_1, v_2, \cdots, v_n$에서 $(v_{k-1}, v_k) \in E, 1 \leq k < n$인 경우
- 경로의 길이(length)는 $n-1$임

(2) 단순 경로(simple path)

- 경로가 같은 연결선을 두 번 포함하지 않는 경로

(3) 기본 경로(elementary path)

- 어떤 정점들도 두 번 만나지 않는 경로

(4) 사이클(cycle) 또는 순회(circuit)

- 경로 $(v_1, v_2, , v_n)$ 에서 종점 v_n과 시점 v_1이 일치하는 경우

(5) 단순 사이클(simple cycle)

- 같은 연결선을 반복하여 방문하지 않는 사이클

(6) 기본 사이클(elementary cycle)

- 시작점을 제외한 어떠한 정점도 반복하여 방문하지 않는 사이클

정의 7-9 그래프에서 정점들 간의 연결 여부에 따라 연결 그래프, 강한 연결 그래프 등이 정의된다.

(1) 연결 그래프(connected graph)

그래프의 모든 정점들이 연결되어 있는 그래프임

(2) 강한 연결 그래프(strongly connected graph)

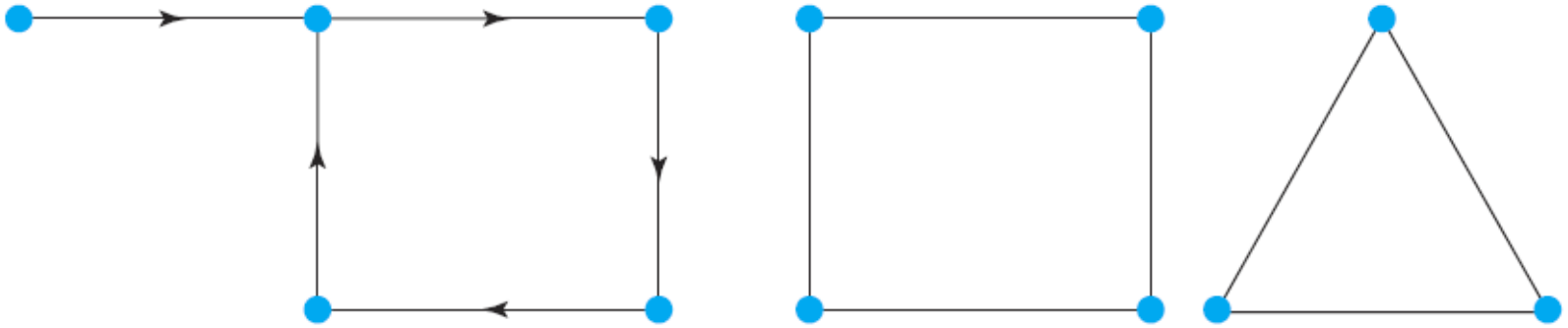
그래프에서 모든 두 정점 a와 b에 대해서 a에서 b로의 경로와 b에서 a로의 경로들이 존재하는 그래프를 말하는데, 특히 방향 그래프에서만 의미를 가짐

(3) 연결 요소(connectivity component)

그래프에서 모든 정점들이 연결되어 있는 부분

연결 수(connectivity number) : 그래프 G에서의 연결 요소의 개수

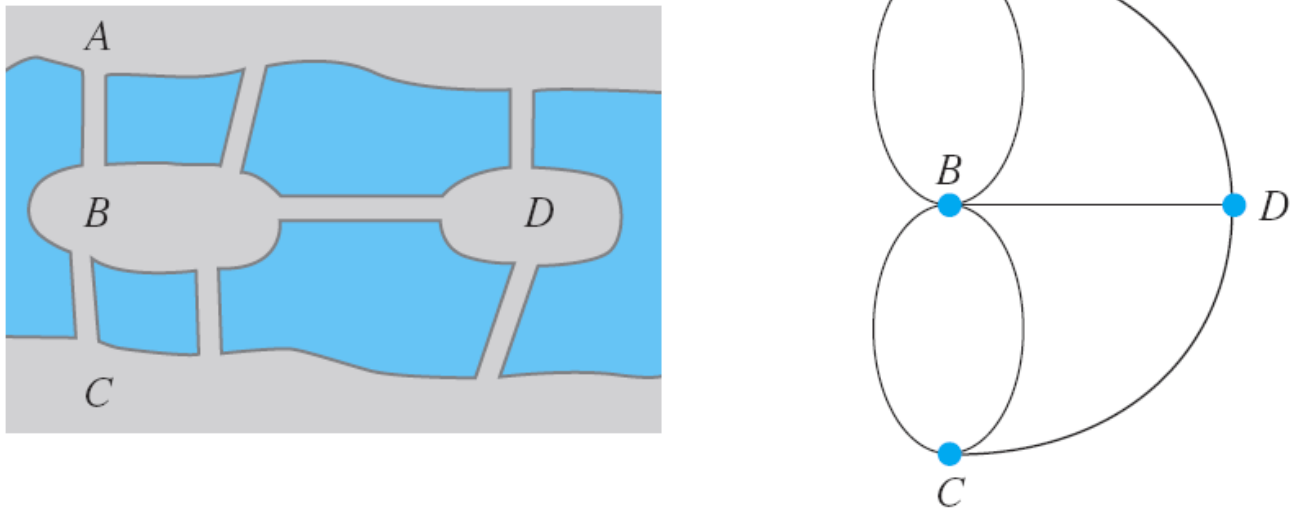
예제 7-2 다음 그림이 연결 그래프인지 강한 연결 그래프인지를 판단해보자.



(1) (2)

풀이 (1)은 연결 그래프이나 강한 연결 그래프는 아니다. 왜냐하면 연결선의 반대 방향으로의 경로가 없기 때문이다. (2)는 그래프가 서로 떨어져 있기 때문에 연결 그래프가 아니다.

- 멀티 그래프(Multi-graph)란 한 쌍의 정점 사이에 연결선의 개수의 제한이 없는 중복된 연결선을 허용하는 특별한 그래프임
- 쾨니히스베르크(K nigsberg) 다리 문제를 해결하는 방법은 멀티 그래프를 모델링하는 것임



〈그림 7.7〉 쾨니히스베르크 다리 문제의 멀티 그래프적인 표현

- 쾨니히스베르크 다리를 그래프로 나타낸 *A*, *B*, *C*, *D* 로 이름 붙인 점들은 정점을 나타내고, 정점들 사이의 선들은 연결선을 나타냄

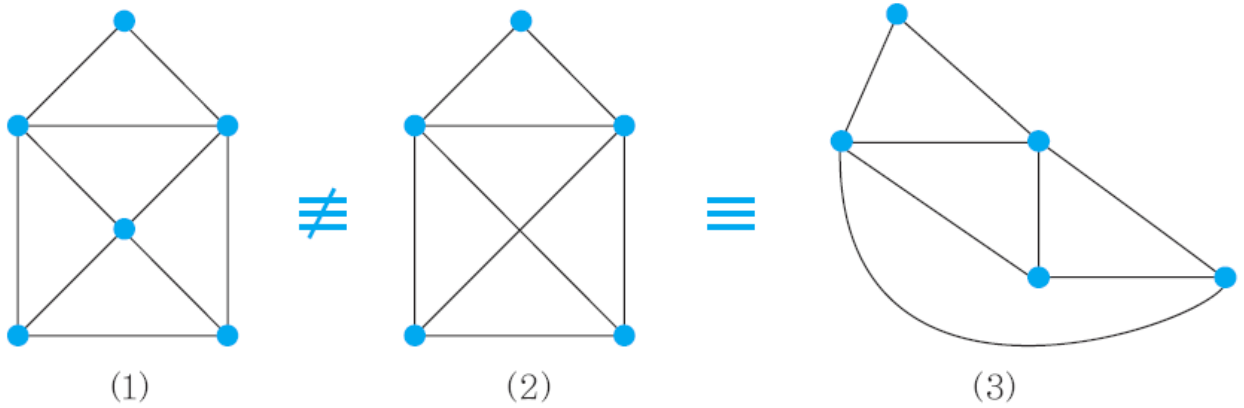
- 멀티 그래프에서 모든 연결선들을 꼭 한 번씩만 통과하는 경로를 오일러 경로(Eulerian path)라고 하는데, 쾨니히스베르크 다리 문제는 오일러 경로를 찾을 수 있는지 여부와 동치임

- 멀티 그래프에서의 오일러 경로를 판별하는 규칙은 모든 정점에서 그것과 연결된 연결선의 개수가 홀수인 정점(홀수점)의 개수가 0 또는 2개인 경우임

- 그래프에서는 *A*, *B*, *C*, *D* 4개의 정점들이 모두 홀수점을 가지므로(총 4개) 오일러 경로가 없다. 따라서 각 다리를 꼭 한 번씩만 건너는 경로가 존재하지 않음

- 정점들과 연결선으로 이루어진 도형에 대한 초기의 그래프 이론은 단순하고 직관적인 문제 해결에 국한됨
- 복잡한 모델에 대한 이론적인 연구로 발전하게 됨

- 기하학에 위상적인 개념을 접합시킨 응용으로서, 위상 기하학(Topological Geometry)적인 관계를 나타냄
- 그래프에서 점의 위치나 선의 길이 등에는 특별한 의미를 부여하지 않고 위상적인 형태에만 중점 둠
- 아래 그림의 (1)과 (2)는 위상적으로 서로 다르지만, (2)와 (3)은 위상적으로 같음



〈그림 7.8〉 위상기하학적 관계

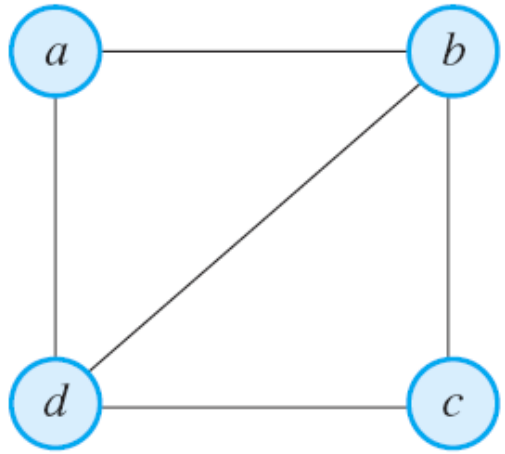
- 그래프는 그림을 이용하여 표현하는 것으로 가장 자연스럽고 이해하기에도 가장 쉬운 방법임
- 컴퓨터는 그림으로 표현된 정보를 이용할 수 없기 때문에 통상 인접 행렬이나 인접 리스트에 의해 표현됨
- 이를 통하여 컴퓨터 프로그램으로 구현됨

(1) 인접 행렬(Adjacency Matrix)

그래프 G = (V, E)에서 IVI = n일 때 G의 인접 행렬은 n × n 행렬 A로 나타내어지며, 이때 A의 각 원소 a_{ij}는 다음과 같이 정의됨

$$a_{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

예제 7-3 다음과 같은 그래프의 인접 행렬을 구해보자.

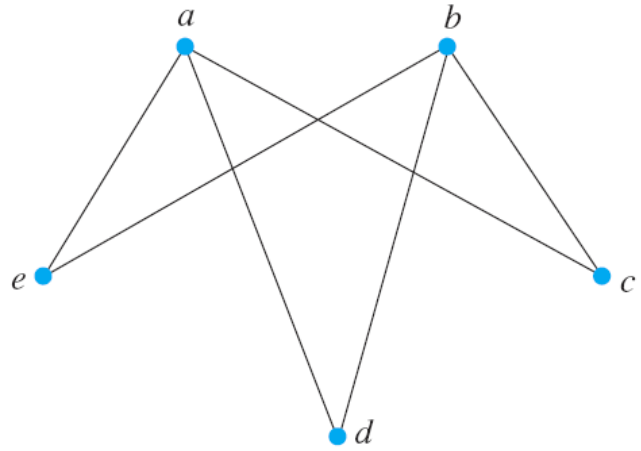


풀이 정점의 개수가 4개이므로 다음과 같은 4×4 행렬로 만들어진다.

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

예제 7-4 다음과 같은 인접 행렬을 가지는 그래프를 그려보자.

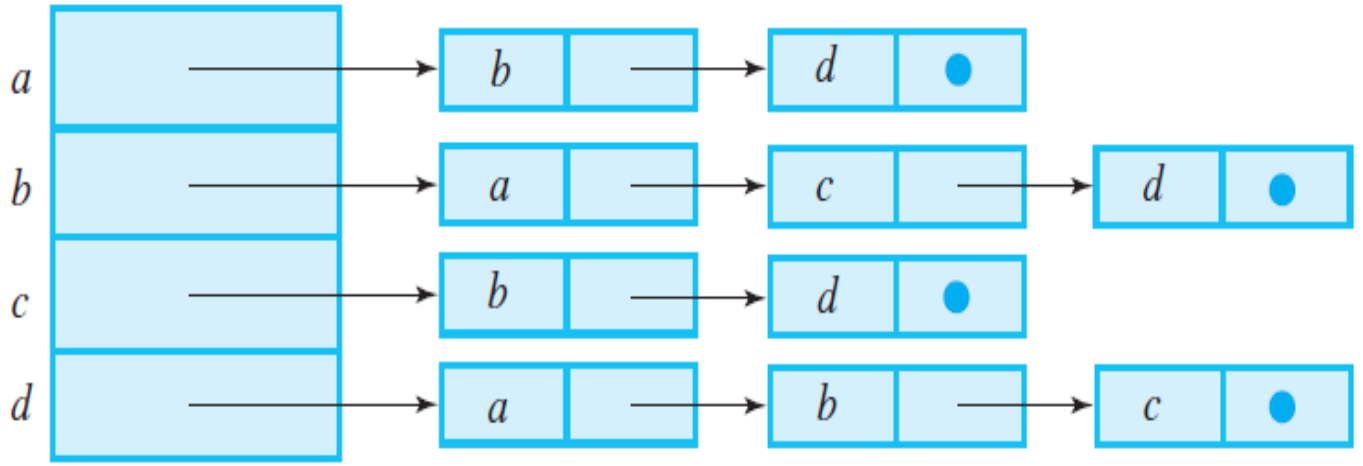
$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



풀이 인접 행렬은 그래프에서 두 꼭지점 사이에 변이 있으면 1로, 그렇지 않으면 0으로 나타내므로, 그것에 따라 다음과 같은 그래프를 그릴 수 있다.

(2) 인접 리스트(Adjacency List)

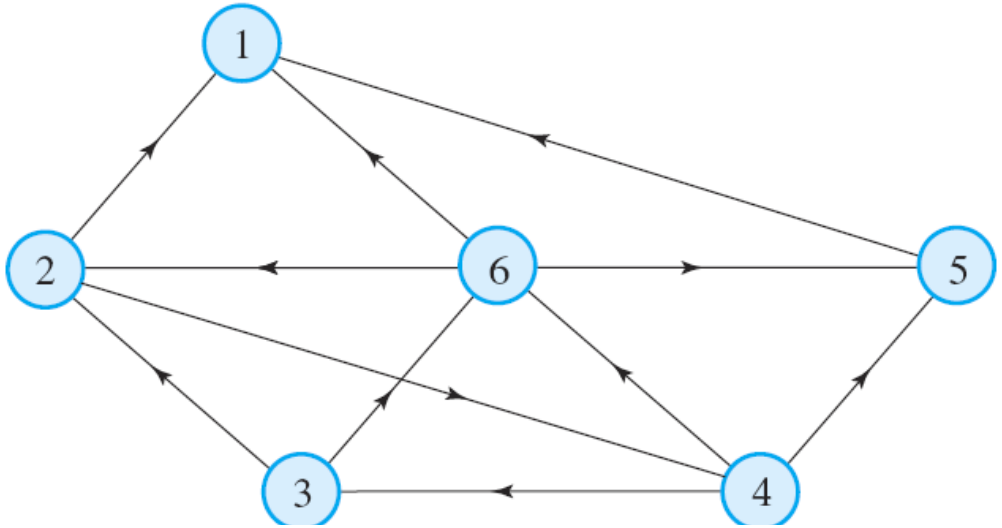
- 각 정점에 대해 포인터(pointer)가 주어지고, 그 점으로부터 연결된 정점들을 차례로 연결 리스트(linked list)로 표시함
- 같은 리스트 내에서는 순서에 관계가 없음



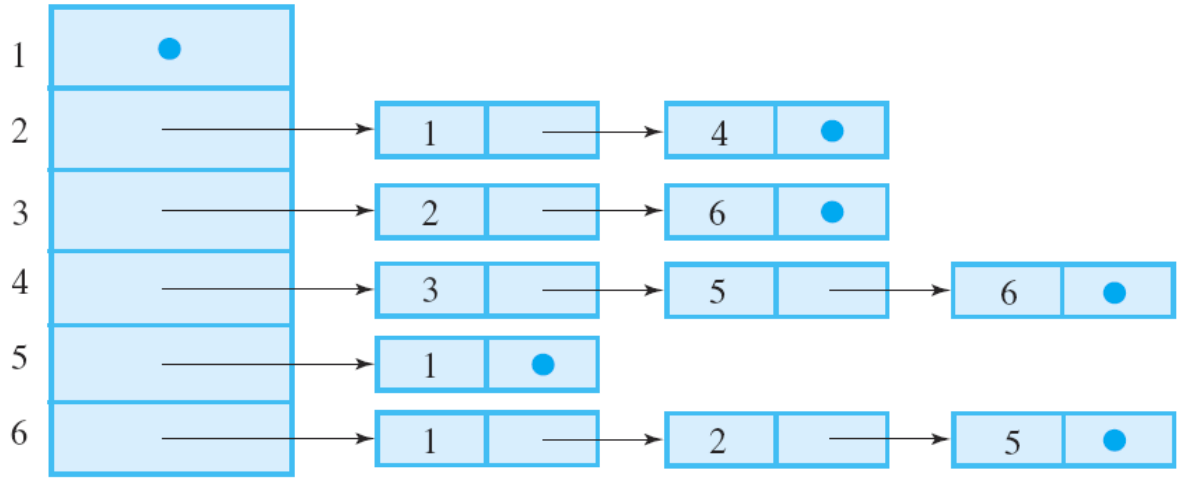
〈그림 7.9〉 인접 리스트

예제 7-5

다음의 방향 그래프에서 인접 리스트를 구해보자.



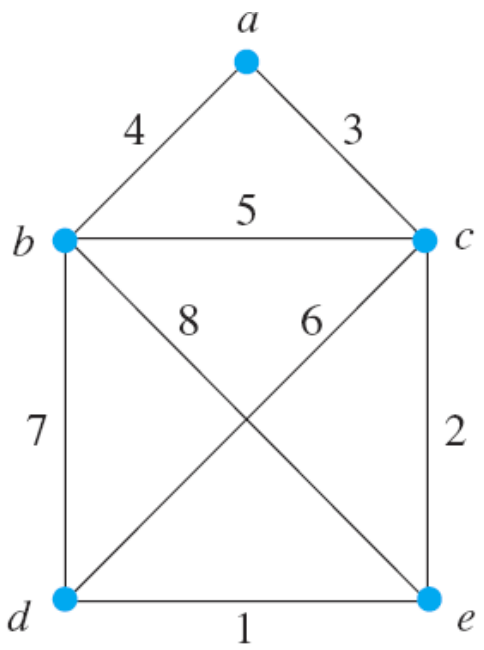
풀이 여기서는 정점 1에서 나가는 연결선이 없다.



정의 7-10 오일러의 성질을 만족하는 특수한 형태의 그래프인 오일러 경로와 오일러 순회는 다음과 같이 정의된다.

(1) 오일러 경로(Eulerian path)란 그래프에서 각 연결선을 단 한 번씩만 통과하는 경로를 말한다.

(2) 오일러 순회(Eulerian circuit)란 그래프에서 꼭지점은 여러 번 지날 수 있지만, 각 연결선을 단 한 번씩만 통과하는 순회를 말한다.

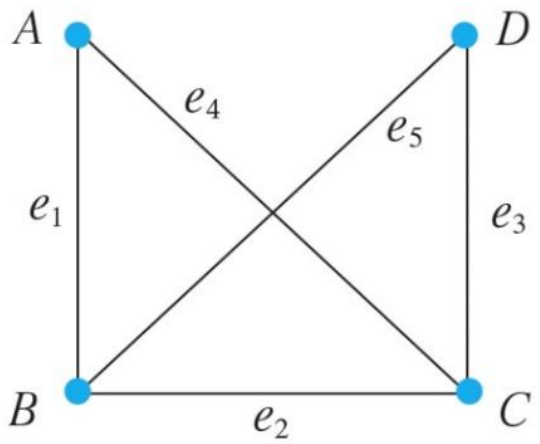


〈그림 7.10〉 오일러 경로

정리 7-1

정점 v와 연결된 연결선의 개수를 정점 v의 차수(degree)라고 하며 $\deg(v)$로 나타낸다. 이 경우 각 연결선은 그래프의 차수를 계산할 때 두 번 반복해서 계산되므로 그래프에서 모든 정점들의 차수의 합은 연결선들 개수의 2배가 된다.

예제 7-6 다음 그래프에서 모든 정점들의 차수의 합은 연결선들 개수의 2배임을 보이자.



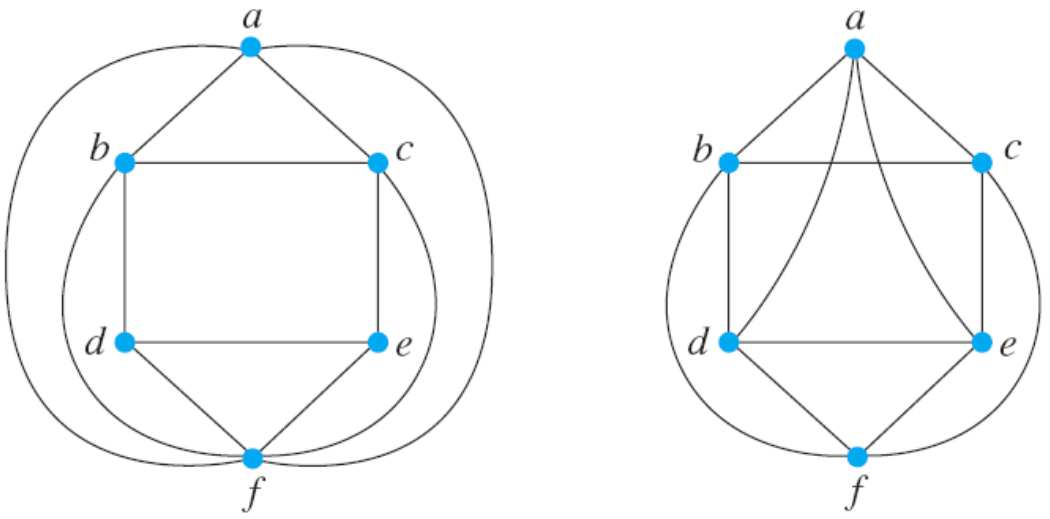
풀이 $\deg(A) = 2,\ \deg(B) = 3,\ \deg(C) = 3,\ \deg(D) = 2$

이므로, 차수의 합은 10이고 연결선은 5개이므로 차수의 합은 연결선 수의 두 배와 같다.

정리 7-2

어떤 그래프 G가 오일러 경로를 가지기 위한 필요충분조건은 G가 연결 그래프이고, 홀수 차수의 개수가 0 또는 2인 경우이다.

예제 7-7 다음과 같은 두 그래프들이 오일러 경로를 가지는지를 확인해보자.



(1) (2)

풀이 (1)의 홀수 차수인 정점은 d와 e로서 그 개수가 2이고, (2)의 경우에는 홀수 차수인 정점의 개수가 0이므로 둘 다 오일러 경로를 가진다.

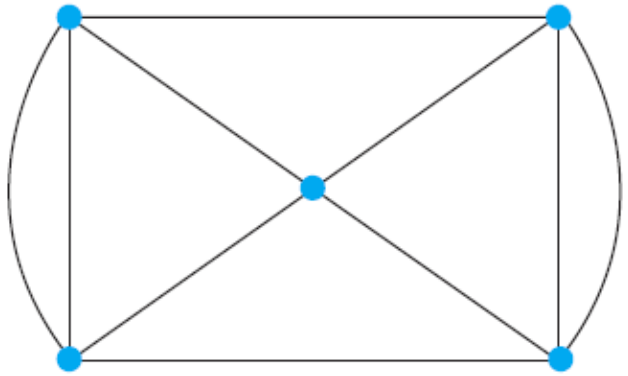
정리 7-3

어떤 그래프 G가 오일러 순회를 가지기 위한 필요충분조건은 G가 연결 그래프이고, 모든 정점들이 짝수 개의 차수를 가지는 경우이다.

정의 7-11

그래프에서 연필을 떼지 않고 모든 변을 오직 한 번만 지나는 것을 한붓그리기(traversable)라고 한다. 연결된 그래프에서 한붓그리기가 가능하려면 시작점과 끝점을 제외한 모든 꼭지점의 차수가 짝수이어야 한다.

예제 7-8 다음 그래프는 한붓그리기가 가능한지를 판별해보자.



풀이 각 정점의 차수가 모두 짝수 차수로서 홀수점이 존재하지 않는 오일러 경로이므로 한붓그리기가 가능하다.

정의 7-12 해밀턴의 성질을 만족하는 특수한 형태의 그래프인 해밀턴 경로와 해밀턴 순회는 다음과 같이 정의된다.

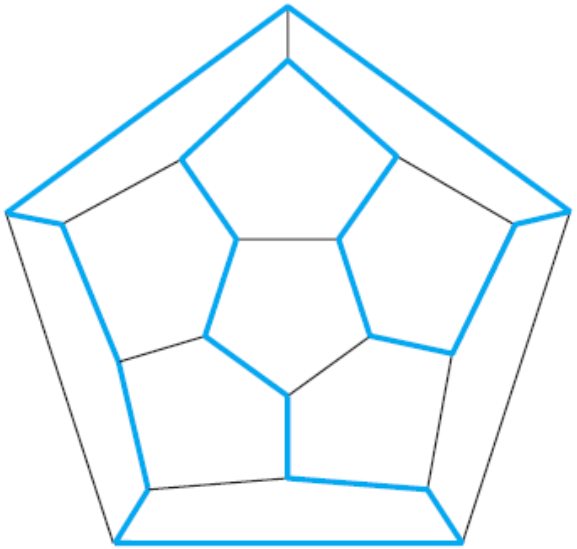
(1) 해밀턴 경로(Hamiltonian path)란 그래프에서 모든 꼭지점을 오직 한 번씩만 지나지만 시작점으로 돌아오지 않는 경로를 말한다.

(2) 해밀턴 순회(Hamitonian circuit)란 그래프에서 모든 꼭지점들을 오직 한 번씩만 지나는 순회를 말한다.

여기서 잠깐!!

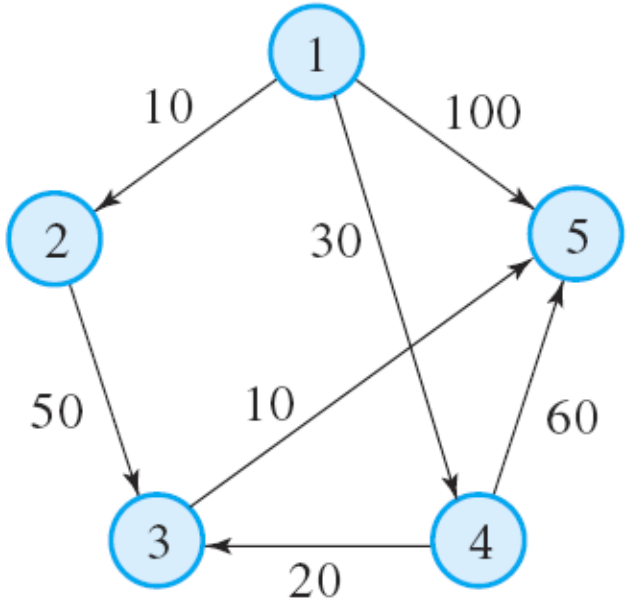
그래프의 한 꼭지점에서 이어진 변을 따라 또 다른 꼭지점으로 이동할 때, 지나간 변을 반복해서 통과하지 않을 경우에 지나온 꼭지점들을 순서대로 나열한 것을 경로라고 하고, 이때 한 꼭지점에서 출발하여 다시 출발한 꼭지점으로 돌아오는 경로를 순회라는 점에 유의하자.

예제 7-9 다음 그래프에서 굵은 색선으로 표현된 연결선은 해밀턴 순회를 나타냄을 알아보자.



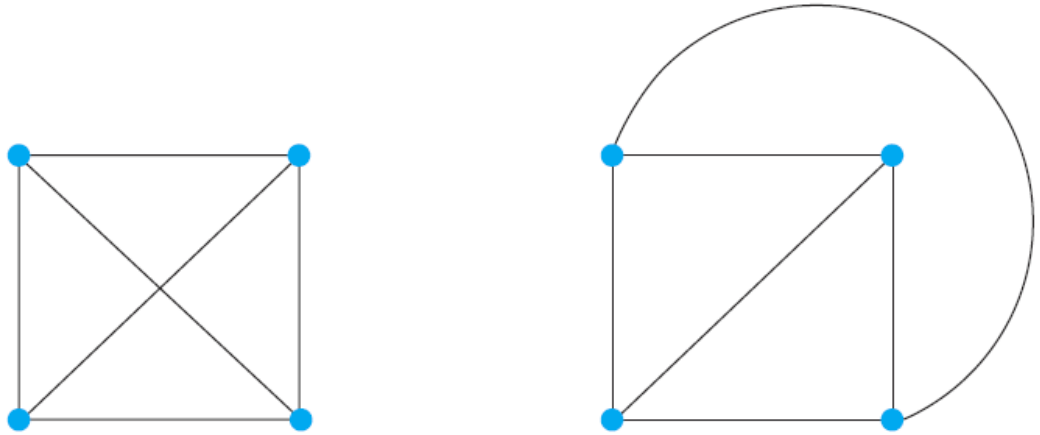
풀이 모든 정점들을 한 번씩만 거치면서 순회를 하기 때문에 해밀턴 순회이다.

정의 7-13 그래프 G의 각 연결선에 0보다 큰 수가 할당되었을 때 이 값을 가중값(weight)이라고 하며, 이와 같은 그래프를 가중 그래프(weight graph)라고 한다.



〈그림 7.11〉 가중 그래프

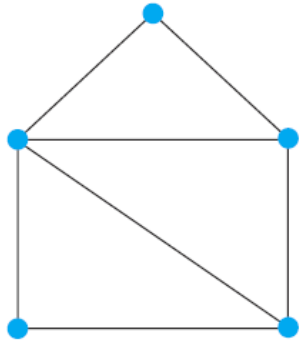
정의 7-14 그래프 $G_1=(V_1,\ E_1)$과 $G_2=(V_2,\ E_2)$가 주어졌을 때, 전단사 함수 $f:V_1\rightarrow V_2$가 존재하여 $\{u,\ v\}\in E_1\Leftrightarrow\{f(u),\ f(v)\}\in E_2$이면 f를 동형(isomorphism)이라고 하고 G_1과 G_2를 **동형 그래프(isomorphic graph)**라고 한다.



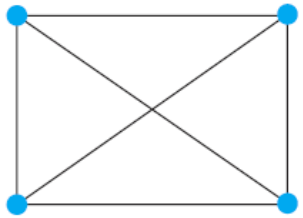
〈그림 7.12〉 동형 그래프의 예

정의 7-15 평면상에서 어떠한 연결선들도 서로 교차할 수 없도록 그려진 하나의 그래프를 **평면 그래프(planar graph)**라고 한다.

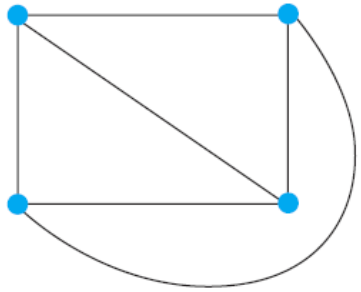
예제 7-10 다음 그래프들이 평면 그래프인지의 여부를 알아보자.



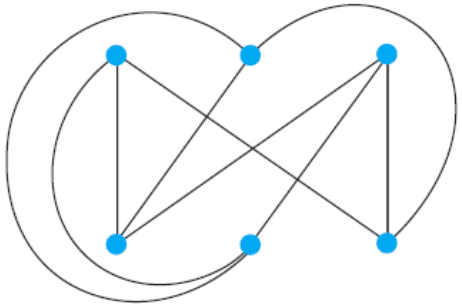
(1)



(2)



(3)



(4)

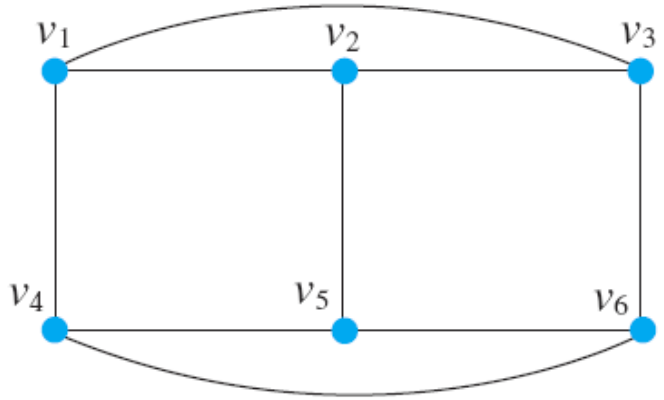
풀이 (1)은 어떤 연결선들도 서로 교차하지 않으므로 평면 그래프이다. (2)는 (3)과 같이 동형으로 변경할 수 있으므로 평면 그래프이다. 그러나 (4)는 평면 그래프가 아니다.

정리 7-4

오일러의 정리

연결된 평면 그래프에서 꼭지점의 수를 v, 연결선의 수를 e, 면의 수를 f라고 할 때 $v-e+f=2$이다.

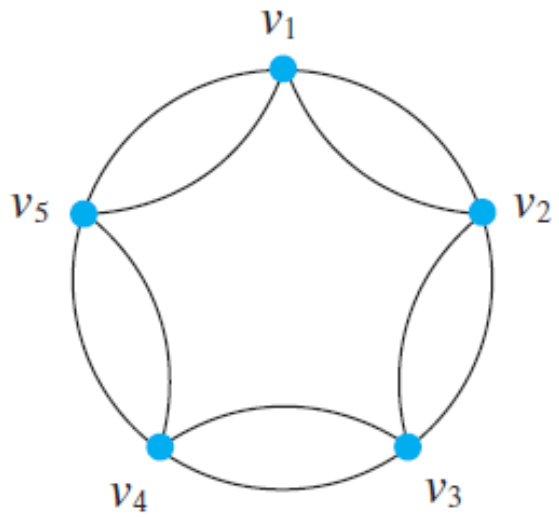
예제 7-11 다음 그래프에서 오일러의 정리가 성립하는지를 살펴보자.



풀이 이 그래프에서 꼭지점의 개수 $v = 6$, 연결선의 개수 $e = 9$, 면의 개수 $f = 5$이므로 $v - e + f = 2$인 오일러의 정리가 성립한다.

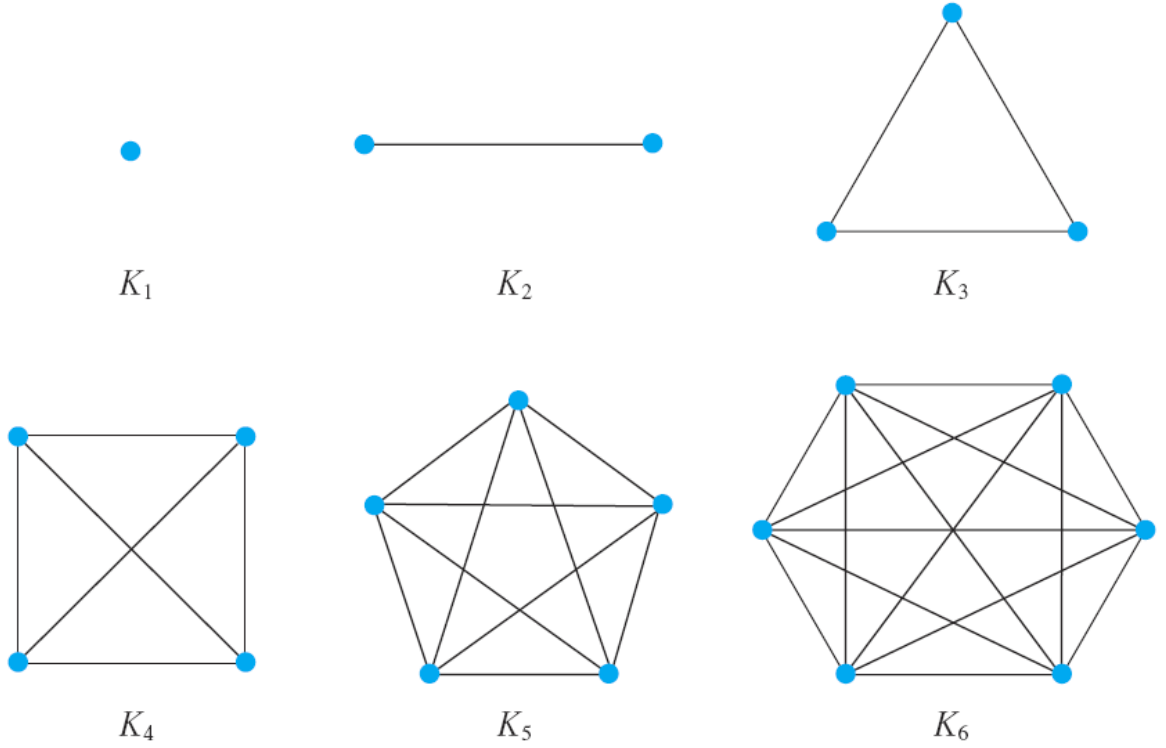
여기서 잠깐!! 평면의 개수를 셀 때 그래프 바깥에 있는 평면도 세어야 함에 유의하자.

예제 7-12 다음 그래프에서 오일러의 정리가 성립하는지를 살펴보자.



풀이 이 그래프에서 꼭지점의 개수 $v = 5$, 연결선의 개수 $e = 10$, 면의 개수 $f = 7$이므로 $v - e + f = 2$인 오일러의 정리가 성립한다.

정의 7-16 그래프 $G=(V, E)$의 모든 정점들의 쌍 사이에 연결선이 존재하면 G를 완전 그래프(complete graph)라고 한다. 즉, 각 꼭지점이 다른 모든 꼭지점들과 연결되는 그래프를 말하는데, n개의 꼭지점으로 구성된 완전 그래프는 K_n으로 표기한다.

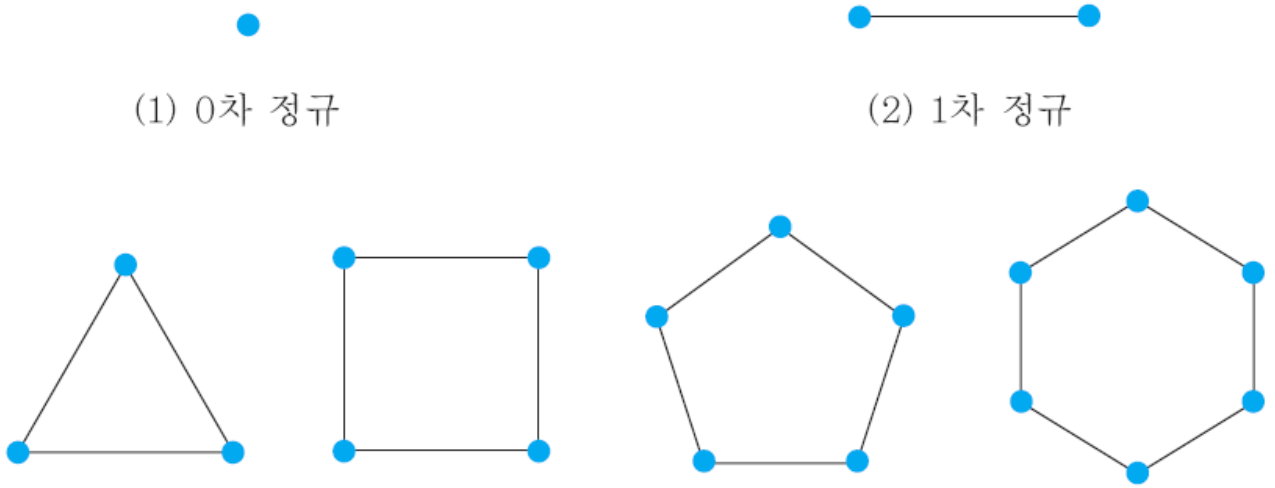


〈그림 7.13〉 완전 그래프의 예

여기서 잠깐!!

n개의 정점을 가지는 완전 그래프에서의 연결선의 개수는 $\frac{n(n-1)}{2}$개이다. 만약 $n=5$인 경우에는 $\frac{5\cdot 4}{2}=10$개의 연결선을 가진다.

정의 7-17 그래프 $G=(V, E)$의 모든 꼭지점의 차수가 k이면 G를 k차 정규 그래프라고 한다.



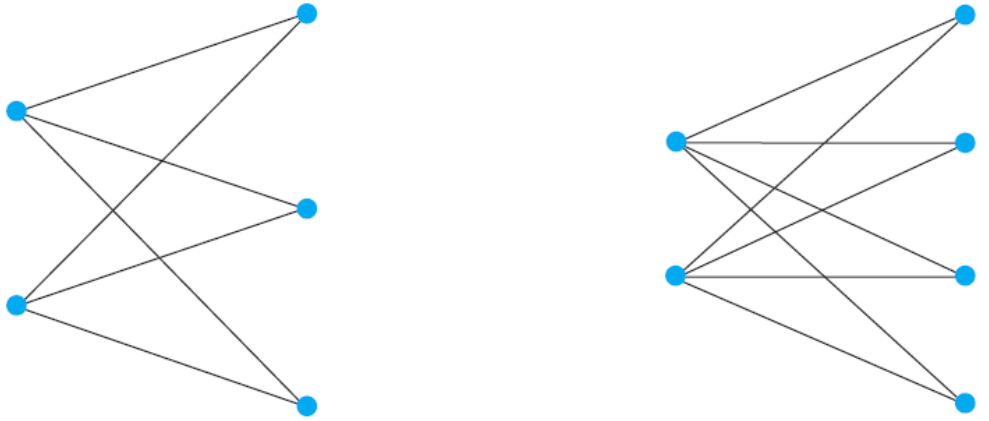
(1) 0차 정규 (2) 1차 정규

(3) 2차 정규

〈그림 7.14〉 k차 정규 그래프

정의 7-18 그래프 $G = (V, E)$에서 V가 두 부분 집합 X와 $Y = V - X$로 나누어져 각 연결선이 X 내의 정점과 Y 내의 정점의 쌍으로 연결되면 그래프 G를 이분 그래프(bipartite graph)라고 한다. 이때 X 내의 모든 정점들과 Y 내의 모든 정점들 사이에 연결선이 존재하면 G를 완전 이분 그래프(complete bipartite graph)라고 한다.

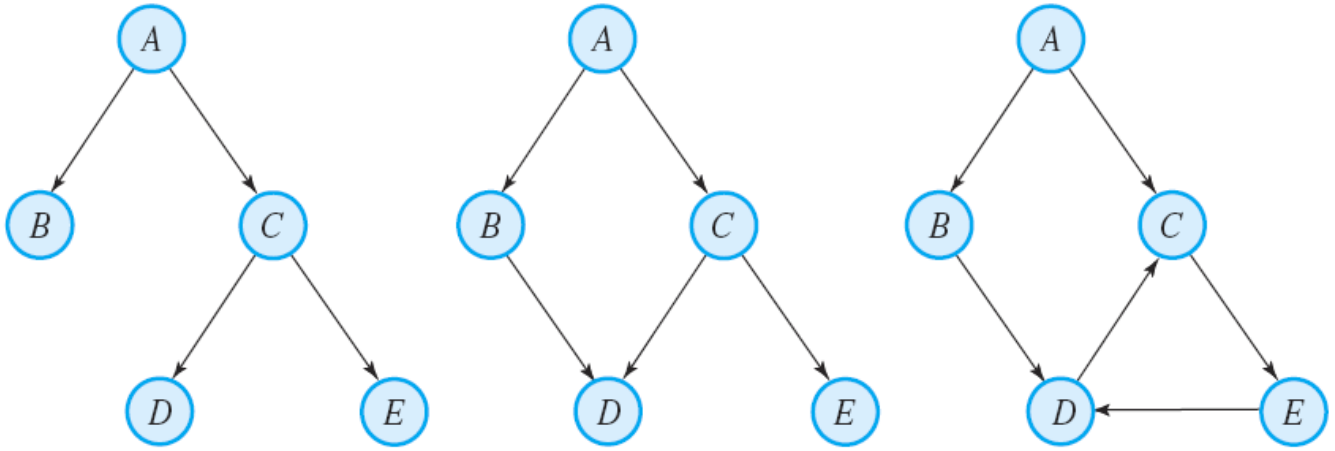
X의 정점의 개수가 m이고, Y의 정점의 개수가 n일 때 완전 이분 그래프를 $K_{m, n}$으로 나타낸다. 완전 이분 그래프는 〈그림 7.15〉에 나타나 있다.



(1) $K_{2,3}$ (2) $K_{2,4}$

〈그림 7.15〉 완전 이분 그래프의 예

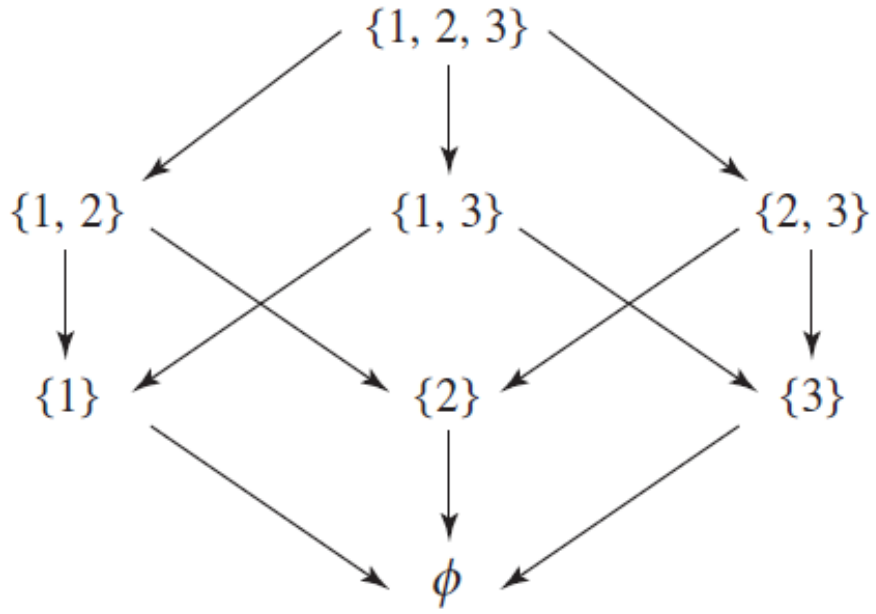
정의 7-19 방향 비사이클 그래프(Directed Acyclic Graph : DAG)는 사이클이 없는 방향 그래프를 말하는데, 트리보다는 일반적이나 임의의 방향 그래프보다는 제한적이다.



〈그림 7.16〉 트리, DAG, 사이클을 가진 방향 그래프

예제 7-13 집합 {1, 2, 3}의 진부분 집합을 DAG로 표현해보자.

풀이 집합 {1, 2, 3}의 진부분 집합 중 2개의 원소로 이루어진 것은 {1, 2}, {1, 3}, {2, 3}이고, 이들의 진부분 집합은 {1}, {2}, {3}이다. 또한 ϕ은 모든 집합의 부분 집합이므로 DAG의 가장 마지막 부분에 위치하게 된다.

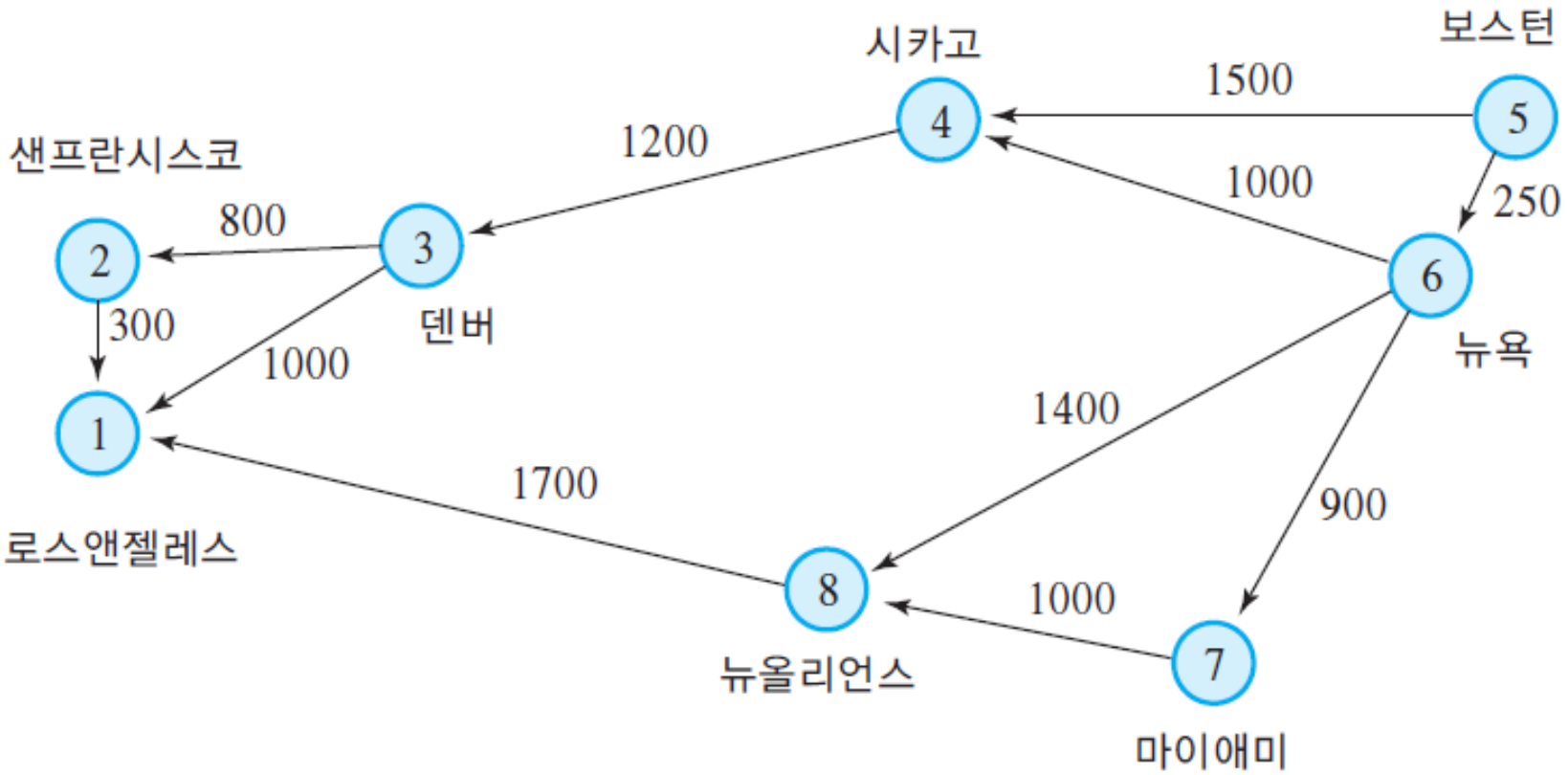


(1)최단 경로

- 그래프는 지도에서 도시를 나타내는 점과 그들 도시 간의 거리를 나타내는 연결선의 값으로 표현됨
- 도시 A에서 도시 B로 가기 위한 방법

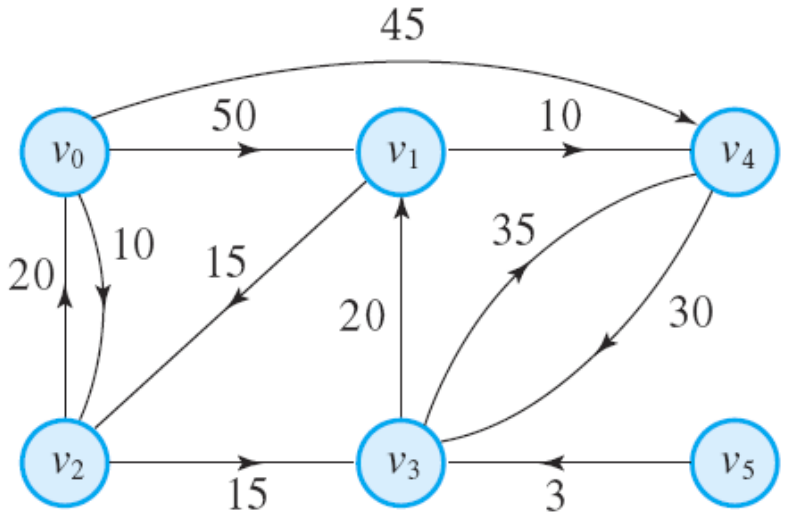
  첫째, A에서 B로 가는 경로가 있느냐는 점

  둘째, A에서 B로 가는 경로가 여러 개 있을 경우 어떤 경로로 가는 것이 가장 짧은 거리인가 하는 점
- 가장 짧은 거리의 경로를 찾는 문제 : 최단 경로 문제(shortest path problem)라 함
- 주어진 방향 그래프에서 경로의 시작점 : 출발점(source)
  - 목적지 : 도착점(destination)
- 주어진 연결선의 길이 : 0 이상인 경우를 가정함



〈그림 7.17〉 지도에서의 방향 그래프

- 연결선의 값이 두 점 사이의 거리를 나타낼 때, v_0가 출발점이라 하면 v_0에서 v_1까지의 최단 경로는 $v_0v_2v_3v_1$임
- 이때 최단 거리는 10 + 15 + 20 = 45가 됨
- 경로의 길이는 3이지만 v_0에서 v_1로 직접 가는 거리인 50보다 짧음
- v_0에서 v_1, v_2, v_3, v_4로 가는 최단 경로의 값을 나타냄



〈그림 7.18〉 가중 그래프

	경로	거리
1)	$v_0v_2v_3v_1$	45
2)	v_0v_2	10
3)	$v_0v_2v_3$	25
4)	v_0v_4	45

〈그림 7.19〉 v_0에서의 최단 경로와 거리

- 최단 경로의 거리 문제를 해결할 수 있는 방법 : 다익스트라 알고리즘(Dijkstra algorithm)
- 출발점으로부터 거리가 최소로 알려진 점들의 집합S를 유지함으로써 가장 짧은 거리를 가지는 나머지 점 v를 차례로 S에 포함시킴

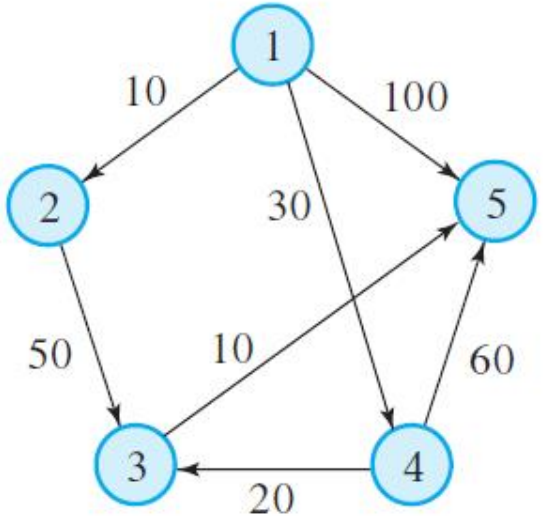
다익스트라 알고리즘

- ✓ 주어진 방향 그래프 G = (V, E)에서 v = {1, 2, ⋯, n}이고 점 {1}이 출발점이라고 가정함
- ✓ 점 i에서 j로 가는 거리를 C[i, j]로 나타내는데, 만약 i에서 j로 가는 경로가 없으면 거리는 ∞가 됨
- ✓ D[i]는 출발점에서 현재점 i에 이르는 가장 짧은 거리를 나타냄

알고리즘 1 다익스트라 알고리즘

```
void Dijkstra()
    /* 이 알고리즘은 정점 1에서 방향 그래프의 모든 정점으로의 최단 거리를 계산한다.*/
{
    S = {1};
    for( i = 2 ; i <= n ; i++ )
        D[i] = C[1, i];  /* D값을 초기화한다 */
    for( i = 1 ; i <= n-1 ; i++ )
    {
        choose a vertex w in V - S such that
            D[w] is a minimum;
        add w to S;
        for(each vertex v in V - S)
            D[v] = min (D[v], D[w] + C[w, v]);
    }
} /* Dijkstra */
```

예제 7-14 다익스트라 알고리즘을 다음과 같은 가중 그래프에 적용해보자.



풀이 처음 단계에서 $S = \{1\}$, $D[2] = 10$, $D[3] = \infty$, $D[4] = 30$, $D[5] = 100$이 된다. for 루프의 첫 번째 반복에서 $w = 2$가 최소의 D값을 가진 점으로 선택된다. 그러면 $D[3] = \min(\infty, 10 + 50) = 60$이 되고 $D[4]$와 $D[5]$는 변동이 없다. 왜냐하면 점 1에서 직접 가는 값이 점 2를 경유하는 것보다 값이 작기 때문이다. for 루프에서의 각 반복 과정 후의 D값은 다음 표와

같으며 4번째 반복 후 점 1에서 점 2, 점 3, 점 4까지의 거리가 각각 10, 50, 30, 60으로 최종 결정된다.

단계	S	w	$D[2]$	$D[3]$	$D[4]$	$D[5]$
0	{1}	—	10	∞	30	100
1	{1, 2}	2	10	60	30	100
2	{1, 2, 4}	4	10	50	30	90
3	{1, 2, 3, 4}	3	10	50	30	60
4	{1, 2, 3, 4, 5}	5	10	50	30	60

(2) 해밀턴 순회의 응용

- 해밀턴 순회의 응용 문제로는 순회판매원 문제(Traveling salesperson problem)가 있음
- 순회판매원 문제란 방문해야 할 도시들과 이들 사이의 거리가 주어졌을 경우, 순회판매원이 어떤 특정한 도시를 출발하여 어떠한 도시도 두 번 방문함이 없이 모든 도시들을 거쳐 처음 출발한 도시로 되돌아올 때, **총 여행 거리가 최소가 되는 경로를 찾는 문제임**

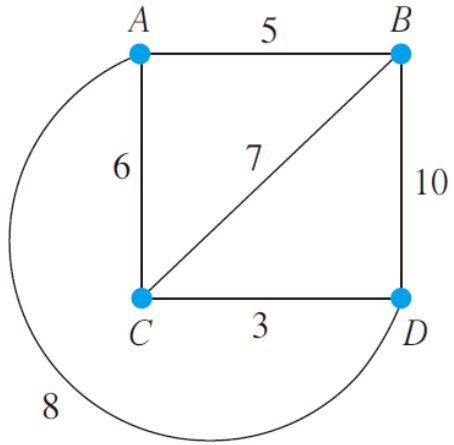
- 최소의 경로가 '최적의 경로라고 할 수 있음
- 일반적인 해결 알고리즘이 존재하지 않음
- 최근접 이웃 방법(nearest neighbor method)을 통하여 최소값은 아니더라도 근사값은 구할 수 있음
- 임의로 선택한 꼭지점에서 출발하여 그 꼭지점과 가장 가까운 꼭지점을 찾아서 연결하고 있음
- 경로를 첨가하는 과정을 반복하며 마지막에 순회를 형성하도록 하는 것임

알고리즘 2 최근접 이웃 방법 알고리즘

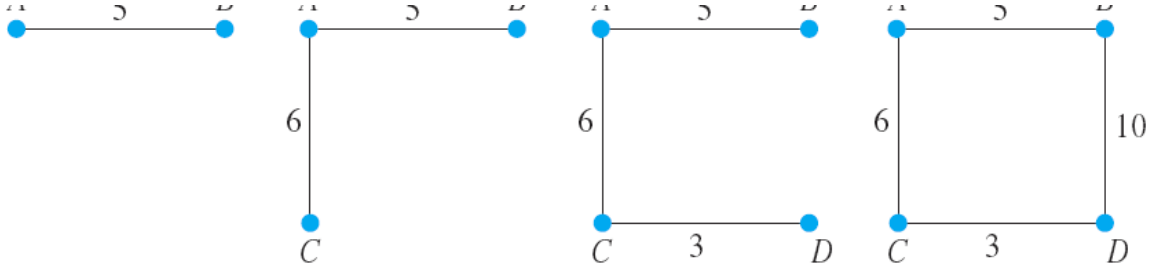
```
begin
    Choose any v1 ∈ V.
    v' ← v1
    w ← 0
    Add v' to the list of vertices in the path.
    while unmarked vertices are remained do
        begin
        Mark v'.
        Choose any unmarked vertex, u, that is closest to v'.
        Add u to the list of vertices in the path.
        w ← w + the weight of the edge (v', u)
        v' ← u
        end
    Add v1 to the list of vertices in the path.
    w ← w + the weight of the edge {v', v1}
end.
```

예제 7-15

다음의 가중 그래프에서 최근접 이웃 방법을 적용한 해밀턴 순회를 만들어 보자.



풀이 임의의 정점 B로부터 시작하자. B와 가장 가까운 방문되지 않은 정점은 A이므로 다음 그림 (1)과 같이 A를 경로에 포함시킨다. 또한 C를 포함시키고 D와 B를 연결한다.



(1) (2) (3) (4)

(1) 깊이 우선 탐색(Depth First Search : DFS)

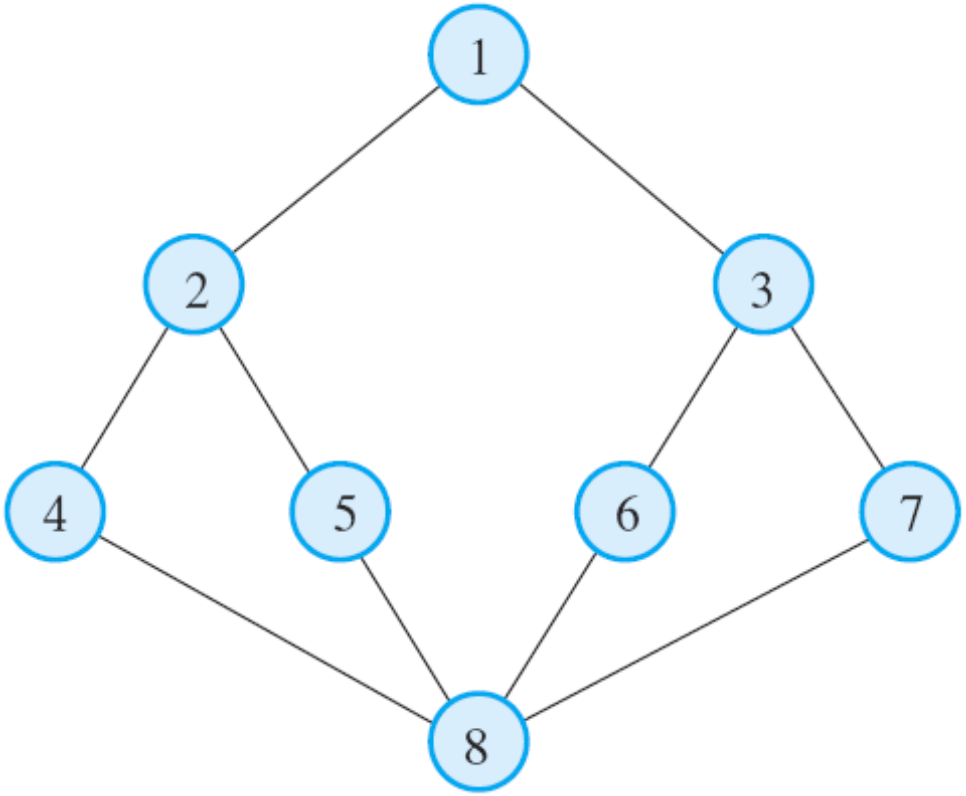
- 깊이 우선 탐색은 시작점 v부터 방문함
- v에 인접한 정점 중에서 방문하지 않은 정점 w를 방문하고 다시 w로부터 탐색을 시작함
- 어떤 정점 u를 방문하고 u에 인접한 모든 정점들을 이미 방문한 경우에는 그 전에 마지막으로 방문한 정점으로 되돌아가서 위의 과정들을 반복함
- 모든 정점들을 방문한 후 탐색을 종료함

➢ 순차적인 프로그램보다는 DFS 알고리즘을 재귀(recursive) 알고리즘으로 구현하는 것이 좋음

➢ 재귀 알고리즘으로 구현할 경우에는 스택(stack)을 사용함

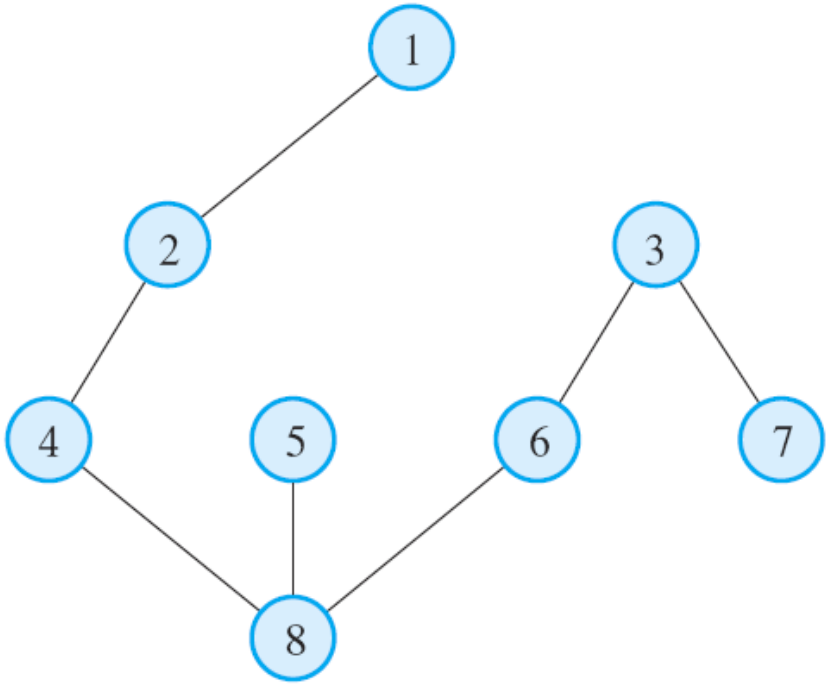
알고리즘 3 DFS

```
void dfs(int v)
    /* G = (V, E)가 n개의 정점을 가진 그래프이고 처음에는 False값으로 행렬
    visited[n]이 주어졌다고 할 때, 이 알고리즘은 정점 v로부터 도달 가능한 모든 정
    점들을 방문한다. */
{
    int w;
    visited[v] = TRUE;
    for(each vertex w adjacent to v)
        if(!visited[w]) dfs(w);
} /* dfs */
```



〈그림 7.20〉 주어진 그래프

DFS의 결과는 탐색은 1,2, 4, 8, 5, 6, 3, 7의 순으로 이루어짐



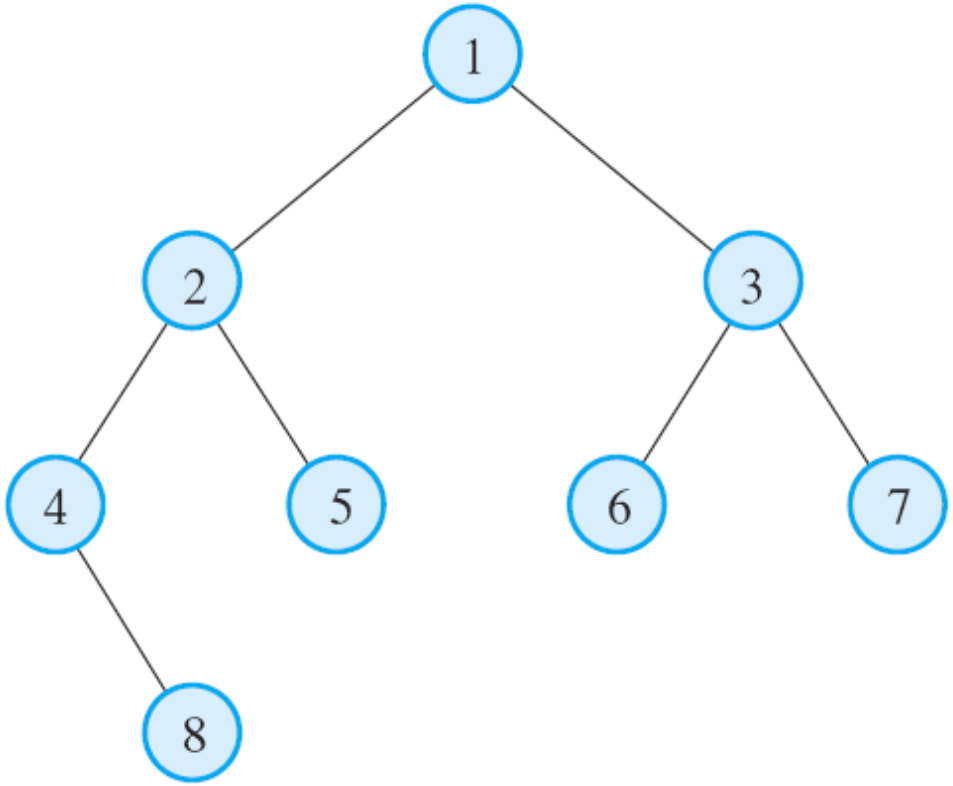
〈그림 7.21〉 DFS의 결과

(2) 너비 우선 탐색(Breadth First Search : BFS)

- 너비 우선 탐색은 처음에 방문한 정점과 인접한 정점들을 차례로 방문 : 깊이 우선 탐색과 차이
- 먼저 시작점 v를 방문한 후 v에 인접한 모든 정점들을 차례로 방문
- 더 이상 방문할 정점이 없는 경우 다시 v에 인접한 정점 가운데 맨 처음으로 방문한 정점과 인접한 정점들을 차례로 방문
- v에 인접한 정점 중 두 번째로 방문한 정점과 인접한 정점들을 차례로 방문하는 과정 반복
- 모든 정점들을 방문한 후 탐색을 종료

➢ 깊이 우선 탐색이 스택(stack)을 사용하는데 비해 너비 우선 탐색은 큐(queue)를 사용

➢ BFS의 결과는 1, 2, 3, 4, 5, 6, 7, 8의 순으로 이루어짐

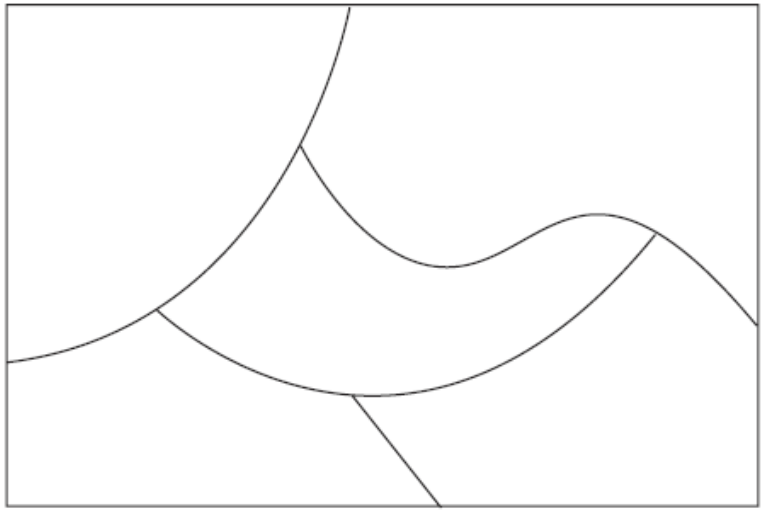


〈그림 7.22〉 BFS의 결과

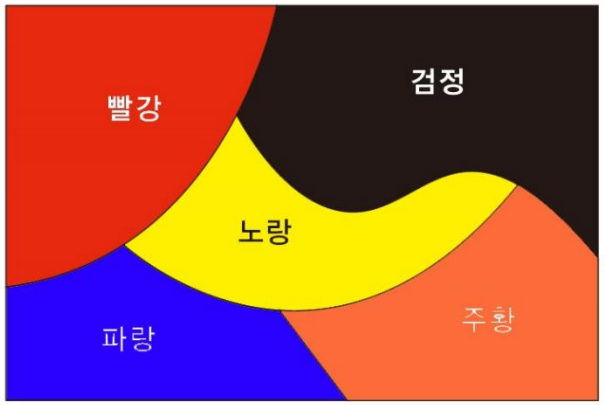
정의 7-20 어떤 주어진 그래프 G에 대해 인접한 어느 두 영역도 같은 색이 안 되도록 각 정점에 색을 칠하는 문제를 그래프 G의 색칠 문제(coloring problem)라고 한다. 그래프 G를 색칠하는 데 필요한 최소한의 색의 수를 $x(G)$로 표현하고 G의 색칠 수(chromatic number)라고 한다.

- 단순 그래프의 색칠은 각각의 정점을 서로 다른 색깔로 색칠함으로써 알 수 있음
- 대부분의 그래프에서는 정점의 수보다 적은 색깔을 사용해도 색칠이 가능함
- 그렇다면 몇 가지 색으로 모든 경계가 구별되도록 색칠할 수 있을까?

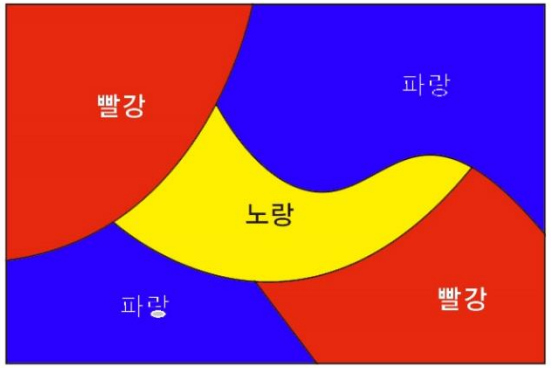
예제 ⑦-16 다음 그림과 같이 경계가 주어진 지도를 색칠해보자. 서로 경계가 만나는 부분이 있으면 서로 다른 색을 사용해야 한다.



풀이 다음 그림은 주어진 지도를 5가지의 색을 사용하여 색칠한 예이다.

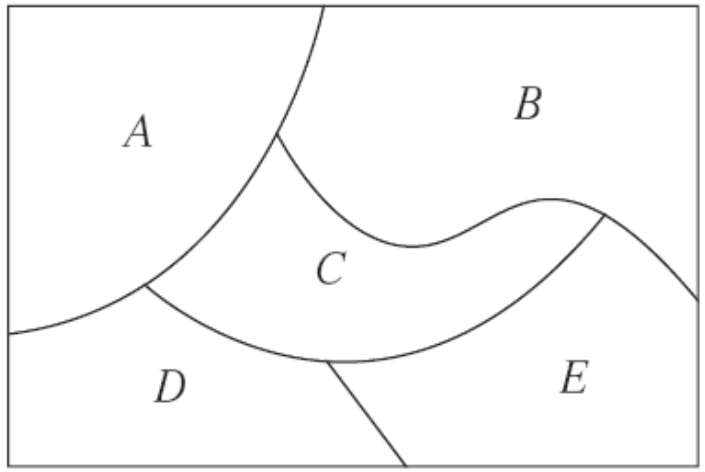


그러나 이 지도는 왼쪽 그림과 같이 4가지 색으로도 가능한 것을 알 수 있으며, 최종적으로 오른쪽 그림과 같이 3가지 색만으로도 가능하다.

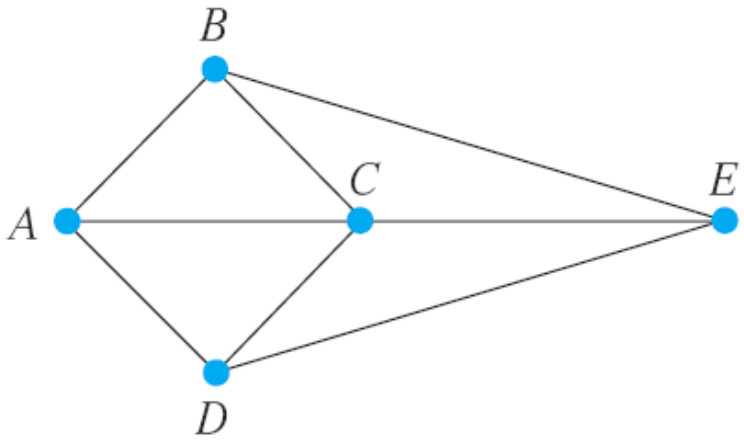


지도를 색칠하는 문제는 그래프의 색칠 문제와 연관시킬 수 있다. 예를 들어, (1)의 그래프는 (2)의 그래프와 동형이 되는데, 이것을 **쌍대 그래프**라고 한다.

- 지도를 색칠하는 문제는 그래프의 색칠 문제와 연관시킬 수 있음
- 예를 들어, (1)의 그래프와 동형이 되는 (2)의 그래프를 쌍대 그래프라고 함
- 지도의 영역을 색칠하는 문제는 쌍대 그래프의 정점들을 색칠하는 문제와 동일하므로 그래프의 인접한 어떤 정점들도 같은 색깔을 가지지 않도록 해야 함



(1)



(2)

정리 7-5

그래프 G에 대해 다음은 서로 동치이다.

(1) G는 두 가지 색으로 칠할 수 있다.

(2) G는 이분 그래프이다.

(3) G의 모든 순환은 짝수의 길이를 가진다.

정리 7-6

모든 평면 그래프는 네 가지 색으로 색칠이 가능하다. 이것을 4색 문제(four coloring problem)라고 한다.

그래프 색칠하기의 3가지 응용 분야

1) 화학물질을 창고에 저장할 때, 만일의 사고를 방지하기 위해 서로 반응하는 물질끼리는 다른 장소에 배치하는 문제
2) 방송국들의 주파수가 같아서 난시청 지역이 발생하지 않도록 하는 문제
3) 애완동물을 판매할 때 사이가 좋지 않은 동물들을 다른 전시관에 배치하는 문제

트리

- 트리의 기본 개념
- 방향 트리
- 이진 트리
- 이진 트리의 표현
- 이진 트리의 탐방
- 생성 트리와 최소비용 생성 트리
- 트리의 활용

트리(Tree)

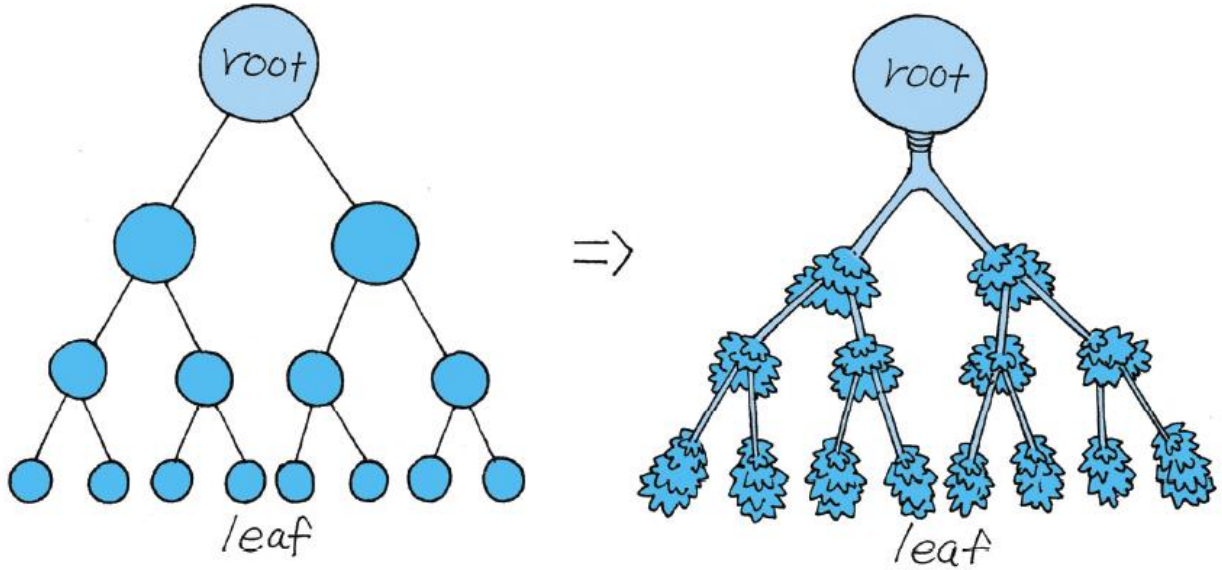
- 그래프 모양이 나무를 거꾸로 세워놓은 것처럼 생겼다고 해서 불리는 이름인데 트리(Tree) 또는 수형도(樹型圖)라고 함
- 그래프의 특별한 형태로서 컴퓨터를 통한 자료 처리와 응용에 있어서 매우 중요한 역할을 담당함
- 이진 트리의 경우에는 산술적 표현이나 자료 구조 등을 매우 간단하게 표현할 수 있는 장점이 있음
- 컴퓨터 기술의 발전과 더불어 수많은 응용 분야에 적용 가능함

트리의 기본 개념

트리(tree)는 하나 이상의 노드(node)로 구성된 유한 집합으로서 다음의 2가지 조건을 만족한다.

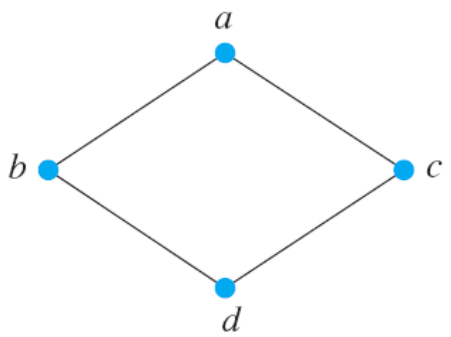
(1) 특별히 지정된 노드인 루트(root)가 있고,

(2) 나머지 노드들은 다시 각각 트리이면서 연결되지 않는(disjoint) $T_1, T_2, \cdots, T_n (N \geq 0)$으로 나누어진다. 이때 $T_1, T_2, \cdots, T_n$을 루트의 서브 트리(subtree)라고 한다.

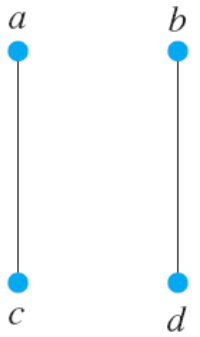


예 1 다음의 여러 그래프들이 트리에 해당되는지를 판단해보자.

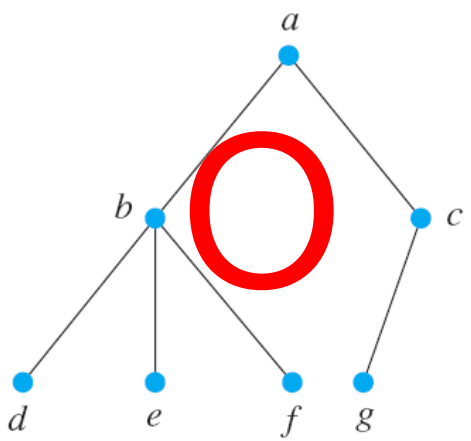
(1)



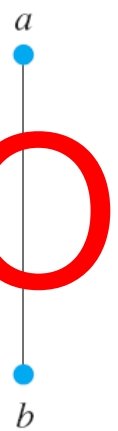
(2)



(3)



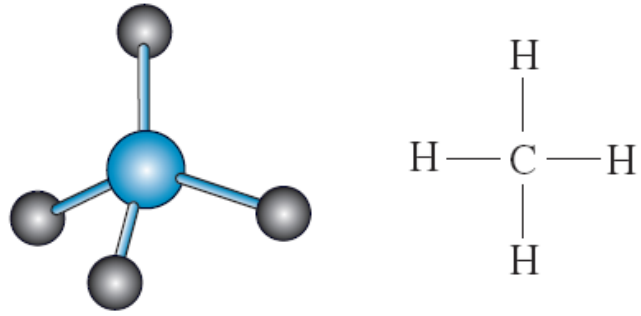
(4)



트리의 기본 개념

트리는 1847년 전기회로의 법칙으로 유명한 독일의 물리학자 키르히호프(Kirchhoff)에 의해 회로 이론의 개발에 이용되었고, 영국의 수학자 케일리(Cayley)는 1857년 탄화수소 계열의 수많은 이성체(isomer)들을 보다 쉽게 구분하기 위하여 트리 개념을 도입하였으며, 처음으로 트리란 명칭을 사용하였다.

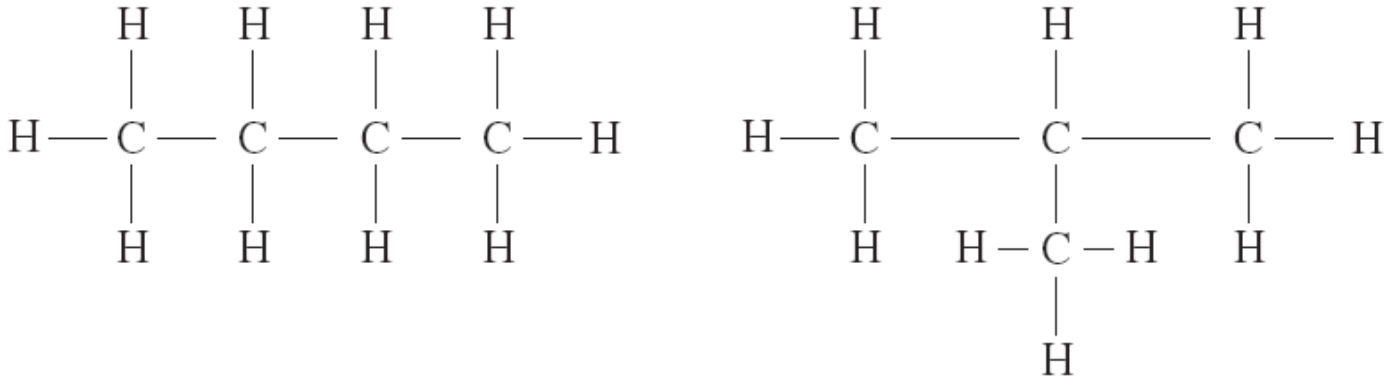
- 화합물인 포화 탄화수소는 $C_k H_{2k+2}$인 형태의 분자식을 가짐
- 탄소의 원자가는 4이고, 수소의 원자가는 1임을 고려하여 화합물 내 에 결합되어 있는 원소들을 선으로 연결하여 구조를 표현함
- CH_4를 분자식으로 가지는 메탄의 구조는 트리의 형태로 나타냄



메탄의 구조와 트리의 표현

트리의 기본 개념

- 부탄과 이소부탄은 화학식(C_4H_{10})으로는 같으나, 서로 다른 화학적 구조를 가지는 이성체
- 트리 개념의 도입은 수많은 이성체들의 분자 구조를 규명하는데 결정적인 역할
- 컴퓨터 기술의 발전과 확산에 힘입어 다양한 분야들에 적용



부탄 이소부탄

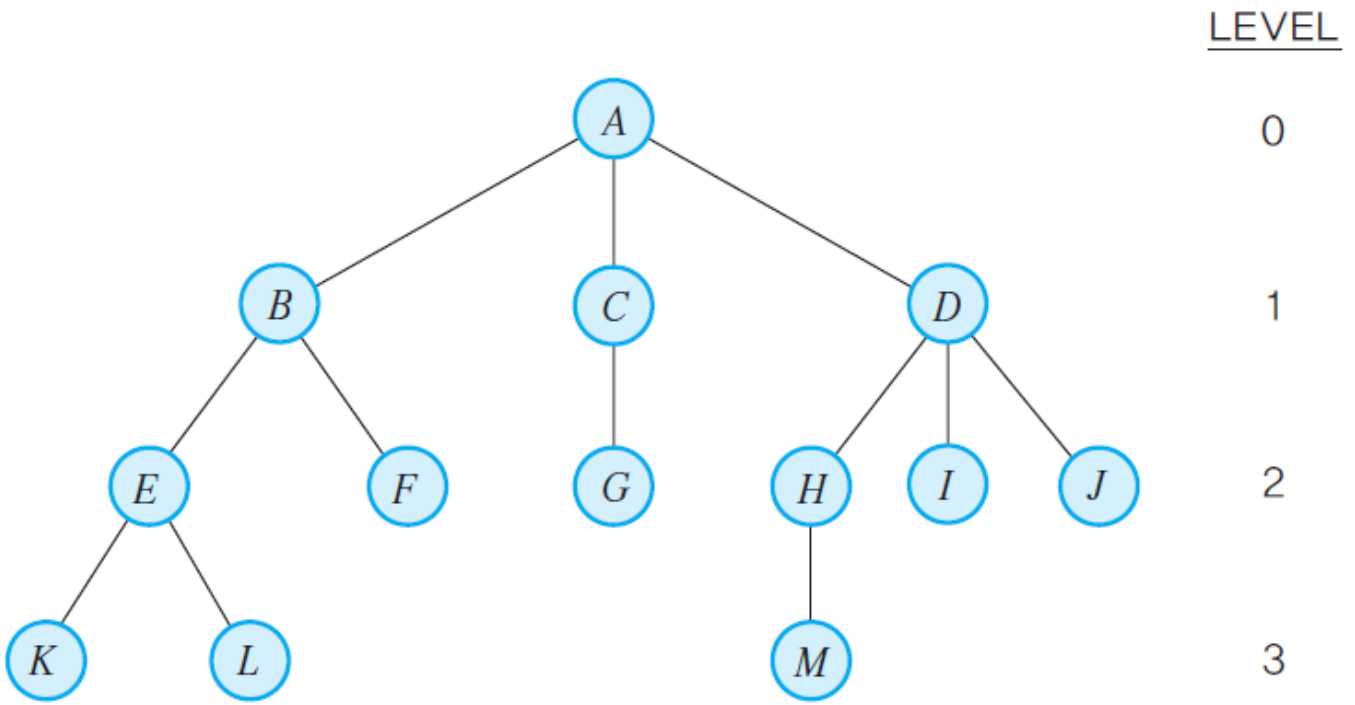
이성체의 서로 다른 트리 표현

트리의 기본 개념

트리의 응용 분야

- 최적화 문제의 해결
- 알고리즘(데이터 구조, 정렬)
- 분자구조식 설계와 화학결합의 표시, 유전학
- 언어들 간의 번역, 언어학
- 자료의 탐색, 정렬, 데이터베이스 구성
- 사회과학(조직의 분류에 있어서의 구조) 등

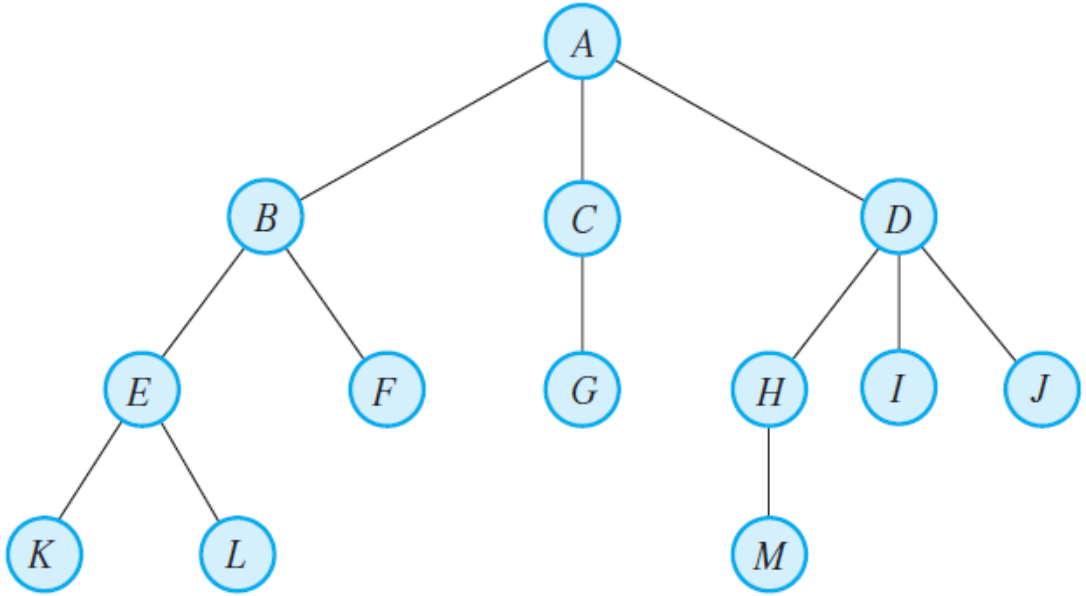
트리의 기본 개념



트리의 예

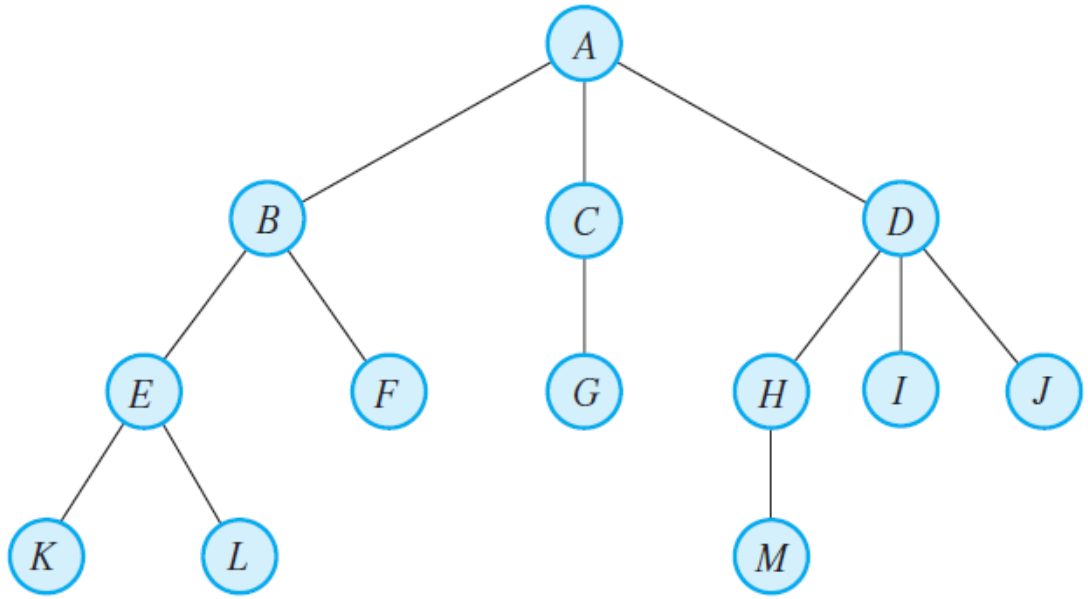
트리의 기본 개념

1) 루트(root) : 주어진 그래프의 시작 노드
   트리의 가장 높은 곳, 노드 A
2) 차수(degree) : 노드의 차수는 그 노드의 서브
   트리의 개수, 노드 A의 차수 3, B의 차수 2, F의
   차수 0
3) 잎 노드(leaf node) : 차수가 0인 노드, K, L, F, G,
   M, I, J
4) 자식 노드(children node) : 노드의 서브 트리의
   루트 노드, A의 자식 노드는 B, C, D



트리의 기본 개념

5) 부모 노드(parent node) : 자식 노드의 반대 개념, B의 부모 노드는 A, G의 부모 노드는 C
6) 형제 노드(sibling node) : 동일한 부모를 가지는 노드, B, C, D는 모두 형제 노드, K, L도 형제 노드
7) 중간 노드(internal node) : 루트도 아니고 잎 노드도 아닌 노드
8) 조상(ancestor) : 루트로부터 그 노드에 이르는 경로에 나타난 모든 노드들, F의 조상은 B와 A, M의 조상은 H, D, A



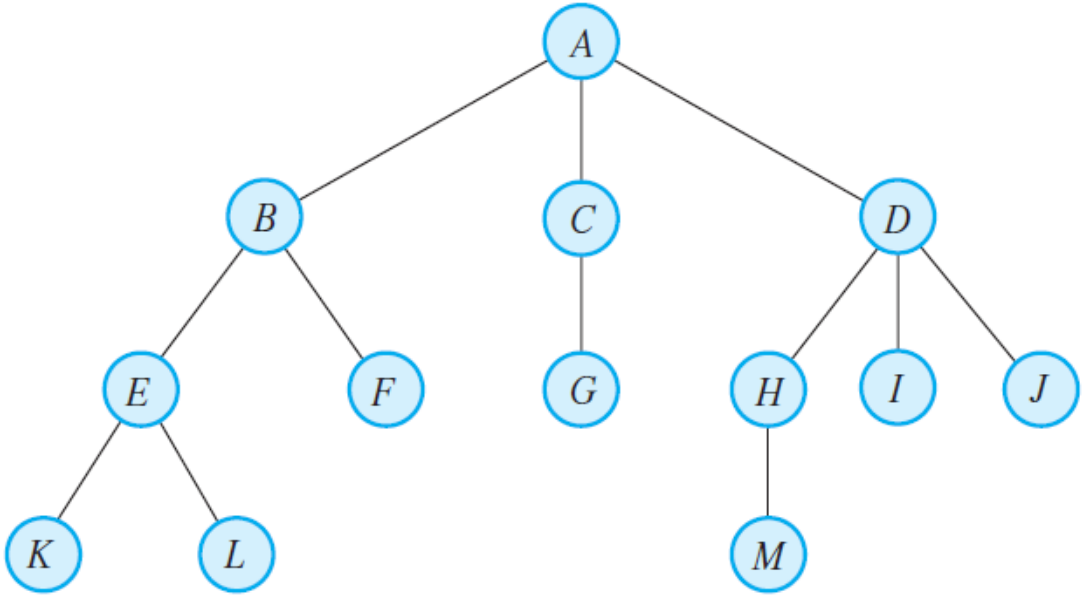
트리의 기본 개념

9) 자손(descendant) : 그 노드로 부터 잎 노드에 이르는 경로상에 나타난 모든 노드 들, B의 자손은 E, F, K, L, H의 자손은 M

10) 레벨(level) : 루트의 레벨을 0으로 놓고 자손 노드로 내려가면서 하나씩 증가, 어떤 노드의 레벨이 p라면 그것의 자식 노드의 레벨은 p+1

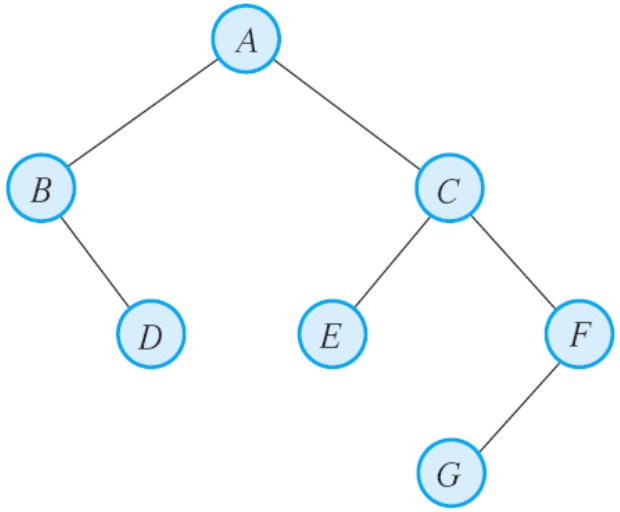
11) 트리의 높이(height) : 트리에서 노드가 가질 수 있는 최대 레벨, 트리의 깊이(depth), 이 트리의 높이는 3

12) 숲(forest) : 서로 연결되지 않는 트리들의 집합, 트리에서 루트를 제거하면 숲을 얻을 수 있음



트리의 기본 개념

예제 8-2 다음 트리에서 주어진 물음에 답해보자.



(1) 잎 노드

(2) 트리의 높이

(3) C의 차수

(4) G의 조상 노드

풀이 (1) D, E, G (2) 3 (3) 2 (4) A, C, F

트리의 기본 개념

정리 8-1

n개의 노드를 가진 트리는 $n-1$개의 연결선을 가진다.

정리 8-2

그래프 $G=(V, E)$에서 $|V|=n$이고 $|E|=m$일 때 다음의 문장들은 모두 동치이다.

(1) G는 트리이다.

(2) G는 연결되어 있고 $m=n-1$이다.

(3) G는 연결되어 있고 어느 한 연결선만을 제거하더라도 G는 연결되지 않는다.

(4) G는 사이클을 가지지 않으며 $m=n-1$이다.

(5) G는 어느 한 연결선만 첨가하더라도 사이클을 형성하게 된다.

예2 15개의 정점을 가지는 트리는 몇 개의 연결선을 가지는지 알아보자.

$m = n - 1$이므로 연결선의 개수는 $n = 14$

트리 T가 n-트리(n-ary tree)란 말은 모든 중간 노드들이 최대 n개의 자식 노드를 가질 때를 말하며, 특히 n이 2인 경우를 이진 트리(binary tree)라고 한다.

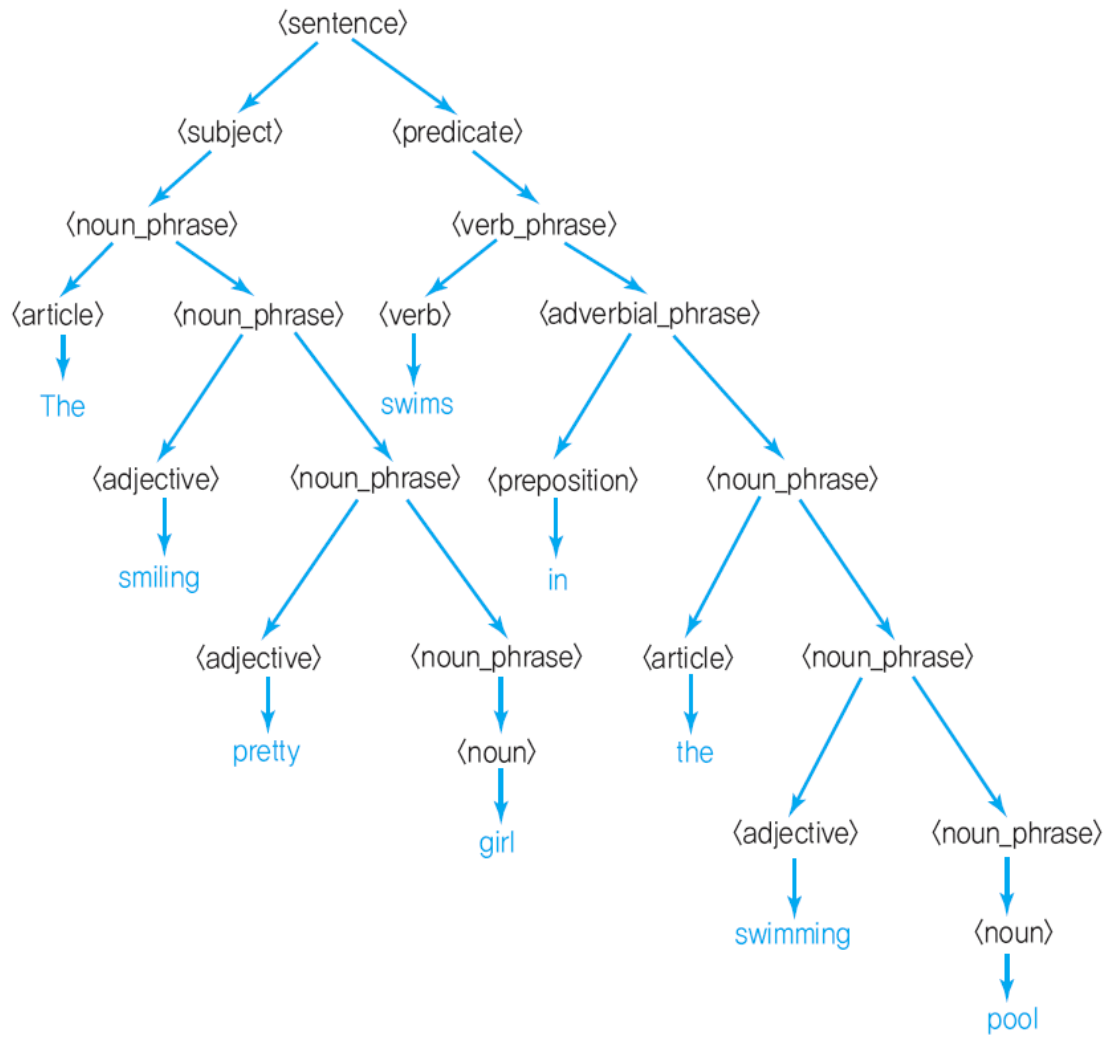
방향 트리

방향이 있고 순서화된 트리는 다음의 성질들을 만족하는 방향 그래프

1) 선행자가 없는 루트라고 불리는 노드가 하나 있으나, 이 루트에서는 모든 노드로 갈 수 있는 경로가 있음
2) 루트를 제외한 모든 노드들은 오직 하나씩만의 선행자를 가짐
3) 각 노드의 후속자들은 통상 왼쪽으로부터 순서화됨

- 어떤 방향 트리를 그릴 때 그 트리의 루트는 가장 위에 있음
- 아크(연결선)들은 밑을 향하여 그려짐

방향 트리



순서화된 트리 구조의 예

이진 트리

이진 트리(binary tree)는 노드들의 유한 집합으로서,

(1) 공집합이거나

(2) 루트와 왼쪽 서브 트리, 오른쪽 서브 트리로 이루어진다.

사향 이진 트리(skewed binary tree)

왼쪽 또는 오른쪽으로 편향된 트리의 구조

완전 이진 트리(complete binary tree)

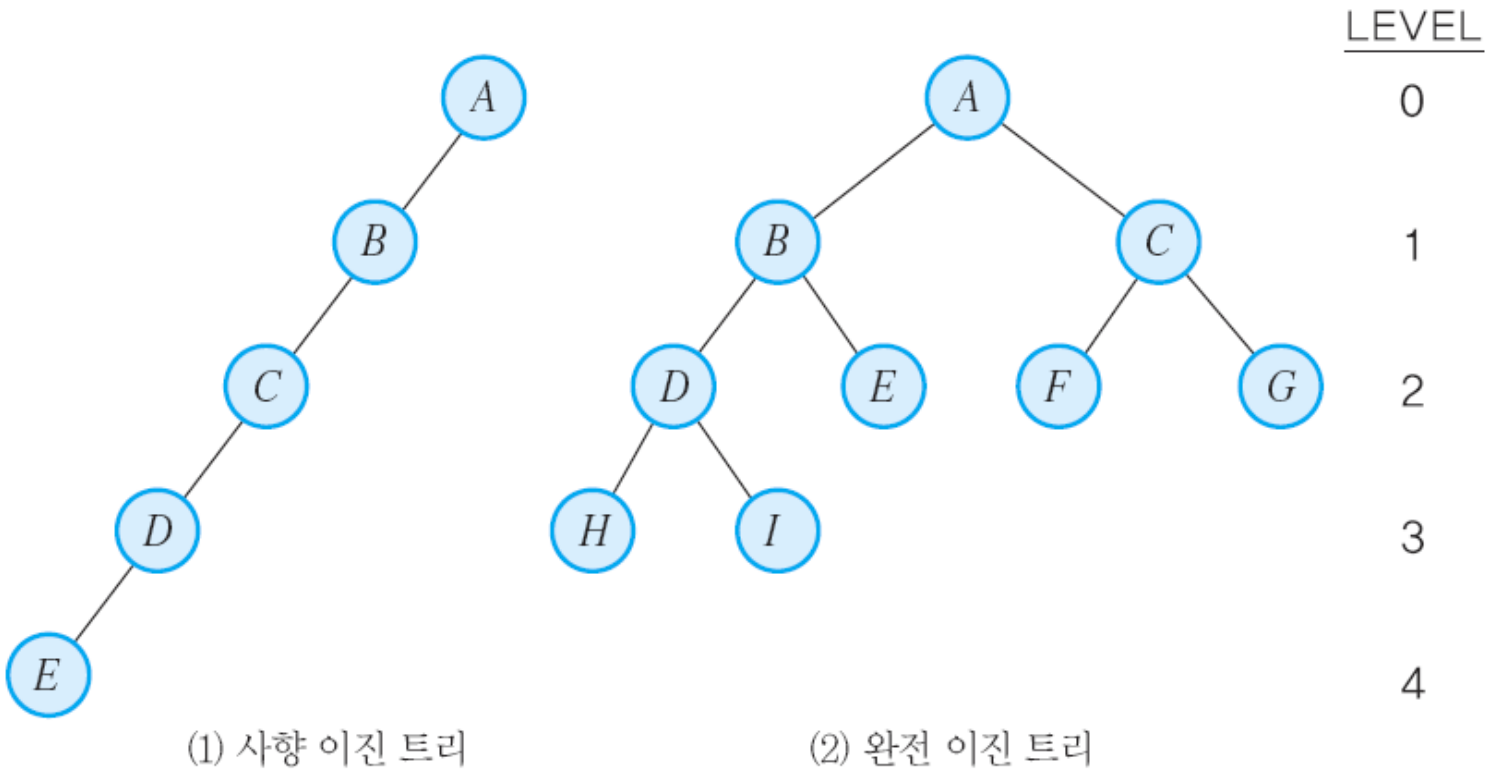
높이가 k일 때 레벨 1부터 k-1까지는 모두 차 있고 레벨 k에서는 왼쪽 노드부터 차례로 차 있는 이진 트리

포화 이진 트리(full binary tree)

잎 노드가 아닌 것들은 모두 2개씩의 자식 노드를 가지며 트리의 높이가 일정한 경우

포화 이진 트리에서 전체가 꽉 차있는 경우

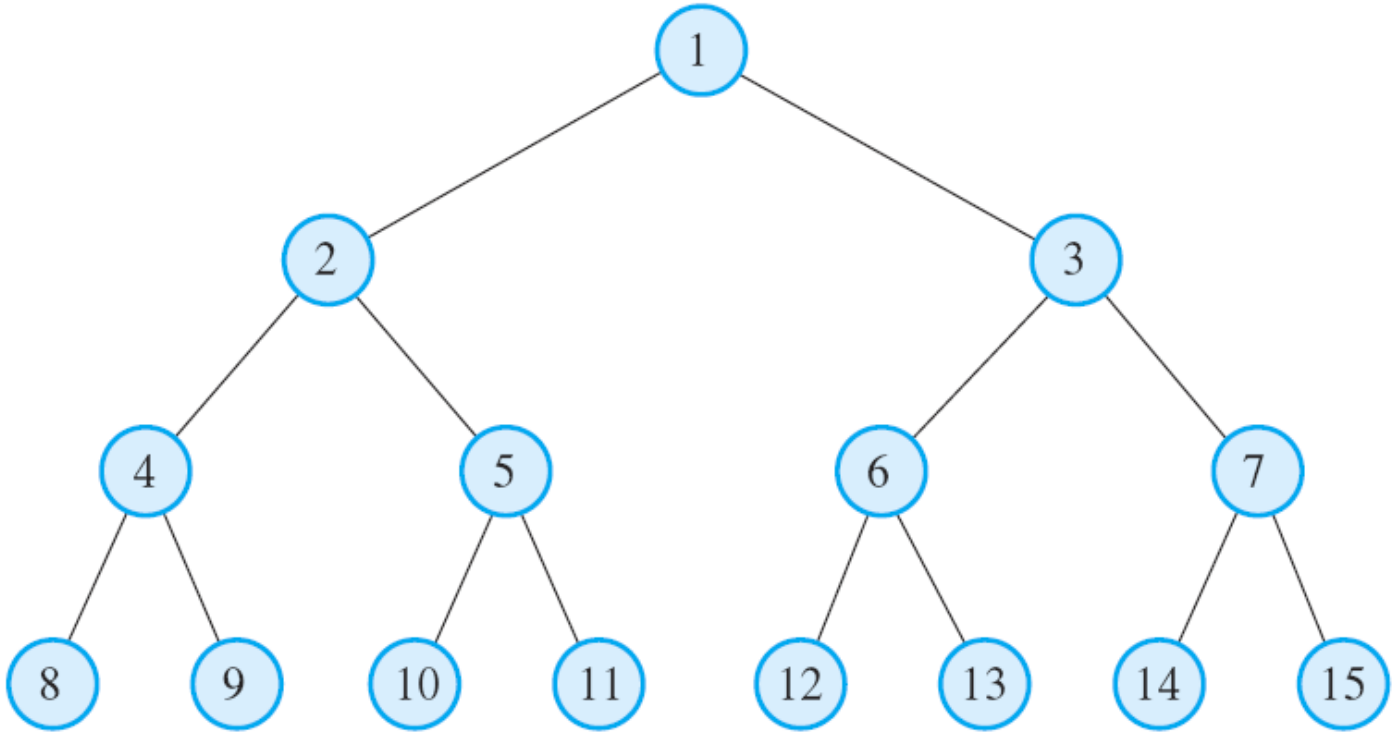
이진 트리



(1) 사향 이진 트리 (2) 완전 이진 트리

이진 트리의 예

이진 트리



포화 이진 트리의 예

이진 트리

이진 트리가 레벨 i에서 가질 수 있는 최대한의 노드 수는 2^i개이며, 높이가 k인 이진 트리가 가질 수 있는 최대한의 전체 노드 수는 $2^{k+1}-1$이다. 예를 들어, 레벨이 2인 경우에는 최대한 2^2개의 노드를 가질 수 있고, 전체적으로는 최대 $2^{k+1}-1$, 즉 7개를 가질 수 있다.

이진 트리에서 잎 노드의 개수를 n_0, 차수가 2인 노드의 개수를 n_2라고 할 때 $n_0 = n_2 + 1$의 식이 항상 성립한다.

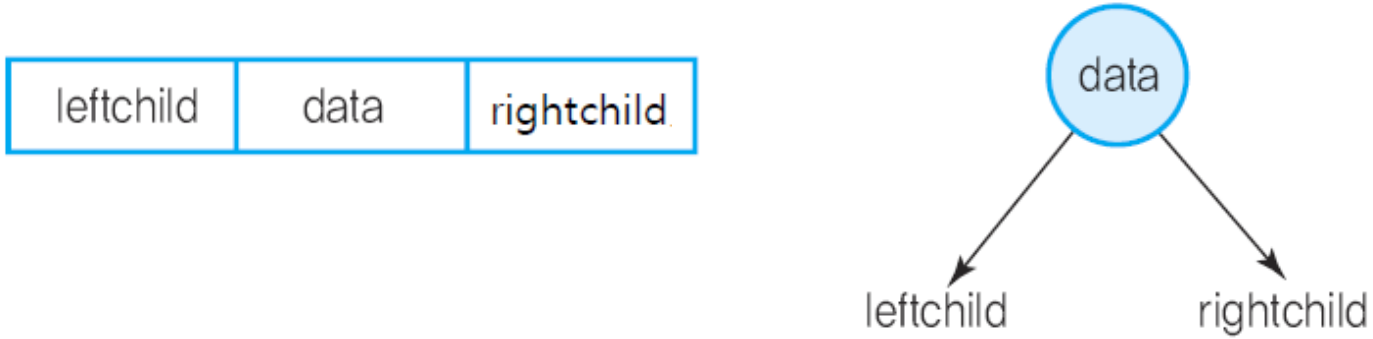
이진 트리의 표현

이진 트리를 표현하는 방법

- 배열(array)에 의한 방법, 연결 리스트(linked list)에 의한 방법
- 배열에 의한 방법 : 트리의 중간에 새로운 노드를 삽입하거나 기존의 노드를 지울 경우 비효율적임
- 연결 리스트에 의한 방법 : 일반적으로 가장 많이 쓰이고 있음

데이터(data)를 저장하는 중간 부분, 왼쪽 자식(left child), 오른쪽 자식(right child)을 각각 가리키는 포인터(pointer) 부분으로 구성

이진 트리의 표현

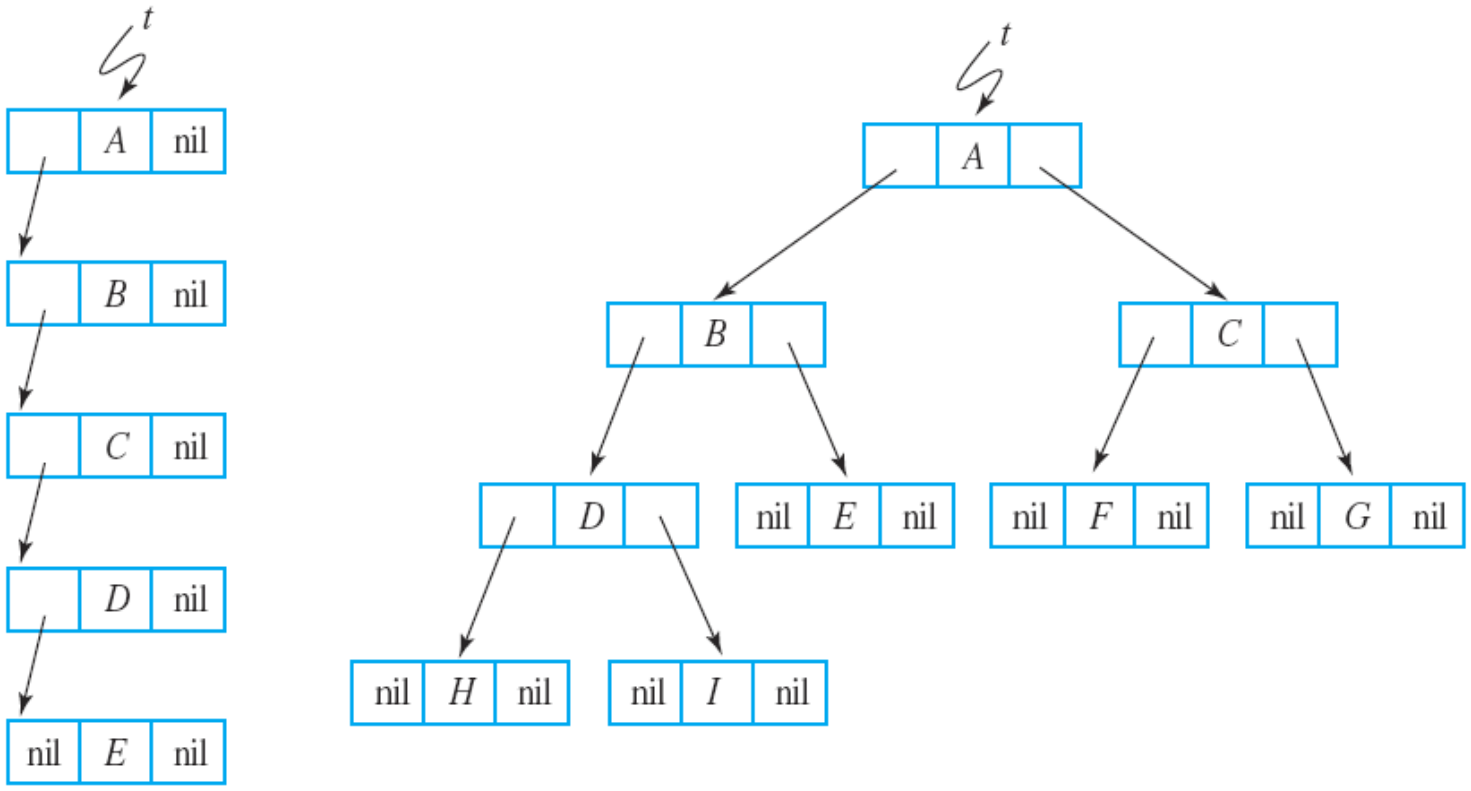


노드의 구조

```
typedef struct treenode {
    struct treenode *leftchild;
    char data;
    struct treenode *rightchild;
} TREE;
```

이진 트리의 표현

이진 트리의 연결 리스트에 의한 표현



이진 트리의 탐방

- 트리는 자료의 저장과 검색에 있어서 매우 중요한 구조 제공
- 트리에서의 연산 : 여러 가지가 있으나 가장 빈번하게 하는 것은 각 노드를 꼭 한 번씩만 방문하는 탐방(traversal)
- 탐방의 결과 각 노드에 들어 있는 데이터를 차례로 나열하게 됨
- 이진 트리를 탐방하는 것은 여러 가지 응용에 널리 쓰임
- 각 노드와 그것의 서브 트리를 같은 방법으로 탐방할 수 있음

트리 탐방의 세 가지 방법은 다음과 같다

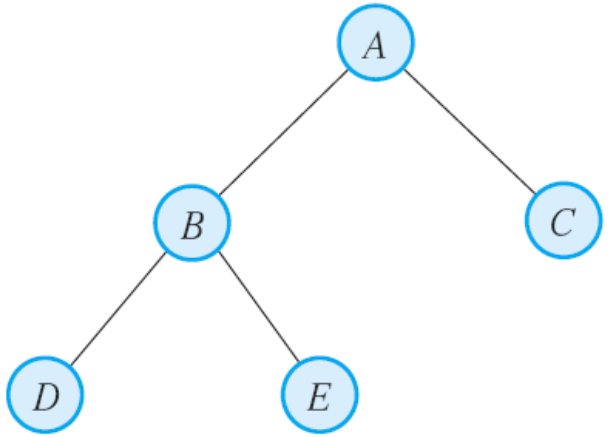
(1) 중순위 탐방

(2) 전순위 탐방

(3) 후순위 탐방

이진 트리의 탐방

- *L*, *D*, *R*을 각각 왼쪽(Left)으로의 이동, 데이터(Data) 프린트, 오른쪽(Right)으로의 이동을 나타낸다고 할 때 총 6가지의 나열 방법
- 왼쪽을 오른쪽보다 항상 먼저 방문한다고 가정 : ***LDR, DLR, LRD***의 3가지 경우
- 중순위(inorder), 전순위(preorder), 후순위(postorder)탐방이라고 함
- 수식 표현에서 중순위 표기(infix), 전순위 표기(prefix), 후순위 표기(postfix)와 각각 대응



이진 트리의 탐방

중순위 탐방(inorder traversal)은 루트에서 시작하여 트리의 각 노드에 대하여 다음과 같은 재귀적(recursive) 방법으로 정의된다.

(1) 트리의 왼쪽 서브 트리를 탐방한다.

(2) 노드를 방문하고 데이터를 프린트한다.

(3) 트리의 오른쪽 서브 트리를 탐방한다.

```
void inorder(TREE *currentnode)
{
    if(currentnode != NULL)
    {
        inorder(currentnode->leftchild);
        printf("%c", currentnode->data);
        inorder(currentnode->rightchild);
    }
}
```

이진 트리의 탐방

중순위 탐방의 재귀적 알고리즘(Recursive algorithm)

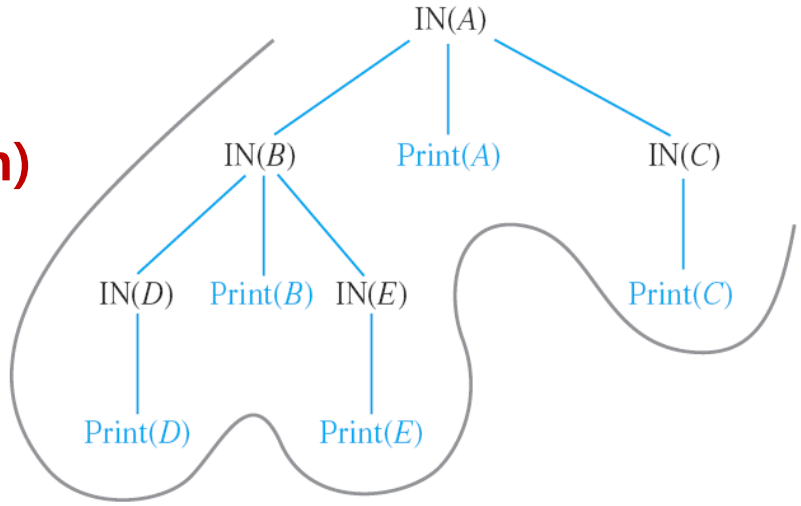
중순위 탐방은 루트로부터 시작하여

inorder(currentnode→leftchild)

printf(currentnode→data)

inorder(currentnode→rightchild)

의 **트리 형태로 계속 전개시켜 나가서 프린트된 결과를 왼쪽부터 차례로 적어나감. [가장 쉽고도 정확한 방법]**



- **이 방법을 탐방에 쓰이는 이진 트리에 적용한 결과 *D, B, E, A, C*의 순서로 프린트함**

이진 트리의 탐방

전순위 탐방(preorder traversal)은 루트에서 시작하여 트리의 각 노드에 대하여 다음과 같은 재귀적 방법으로 정의된다.

(1) 노드를 탐방하고 데이터를 프린트한다.

(2) 트리의 왼쪽 서브 트리를 탐방한다.

(3) 트리의 오른쪽 서브 트리를 탐방한다.

```
void preorder(TREE *currentnode)
{
    if(currentnode != NULL)
    {
        printf("%c", currentnode->data);
        preorder(currentnode->leftchild);
        preorder(currentnode->rightchild);
    }
}
```

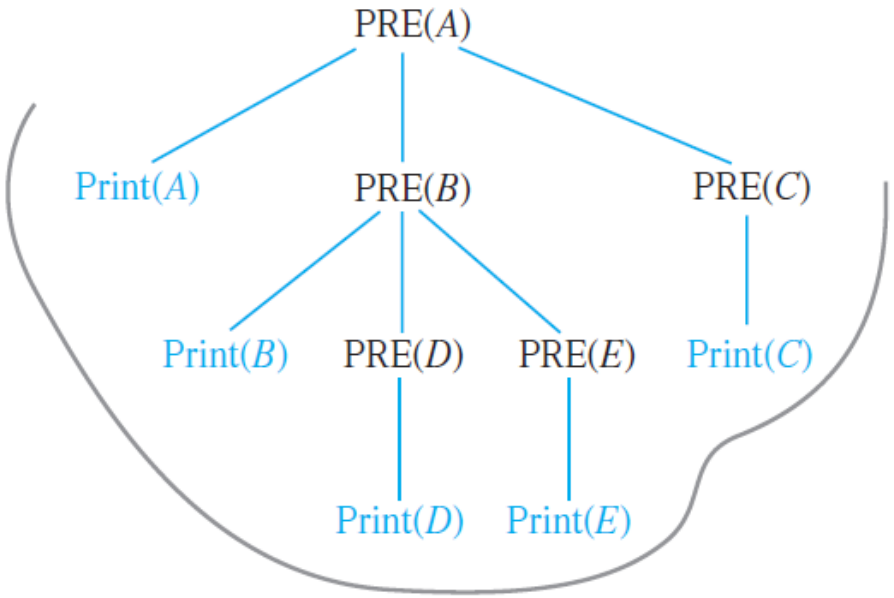
이진 트리의 탐방

전순위 탐방(D, L, R)

printf(currentnode→data)

preorder(currentnode→leftchild)

preorder(currentnode→rightchild)



탐방에 쓰이는 이진 트리를 적용한 프로그램의 작동 결과는 A, B, D, E, C

이진 트리의 탐방

후순위 탐방(postorder traversal)은 루트에서 시작하여 트리의 각 노드에 대하여 다음과 같은 재귀적 방법으로 정의된다.

(1) 트리의 왼쪽 서브 트리를 탐방한다.

(2) 트리의 오른쪽 서브 트리를 탐방한다.

(3) 노드를 탐방하고 데이터를 프린트한다.

```
void postorder(TREE *currentnode)
{
    if(currentnode != NULL)
    {
        postorder(currentnode->leftchild);
        postorder(currentnode->rightchild);
        printf("%c", currentnode->data);
    }
}
```

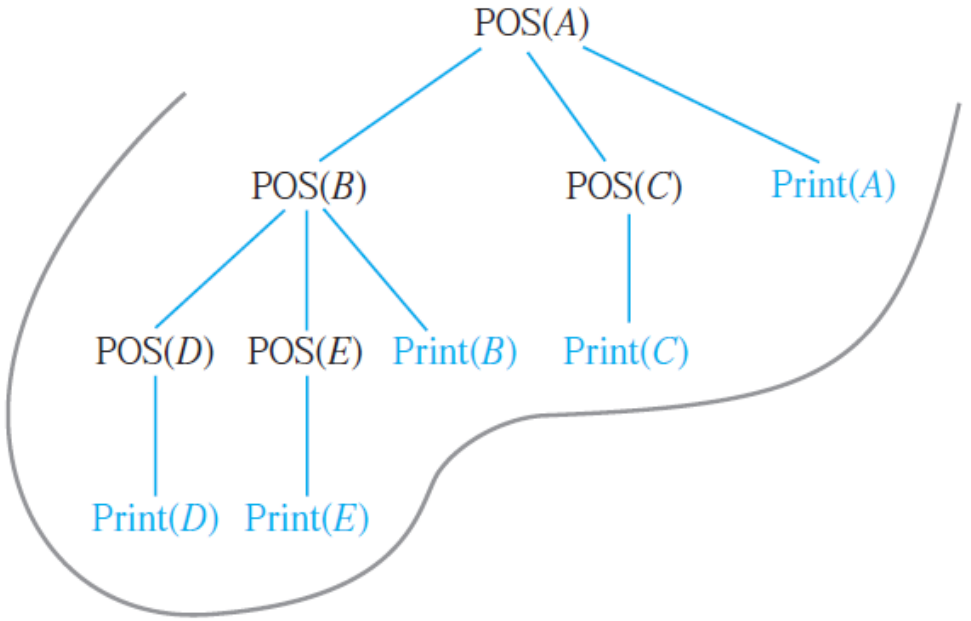
이진 트리의 탐방

후순위 탐방(L, R, D)

postorder(currentnode→leftchild)

postorder(currentnode→rightchild)

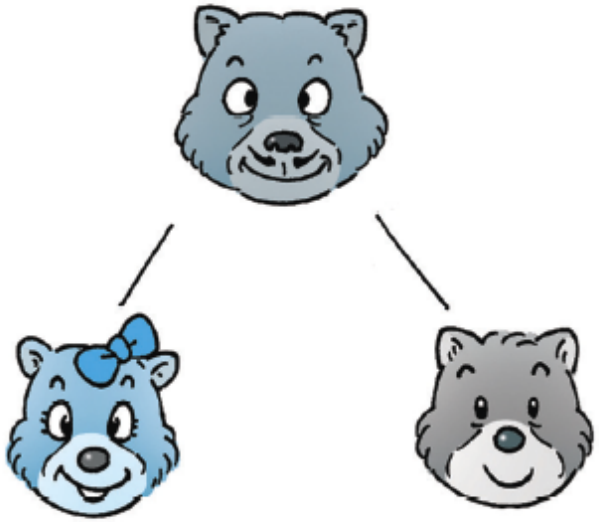
printf(currentnode→data)



탐방에 쓰이는 이진 트리를 적용한 프로그램의 작동 결과는 D, E, B, C, A

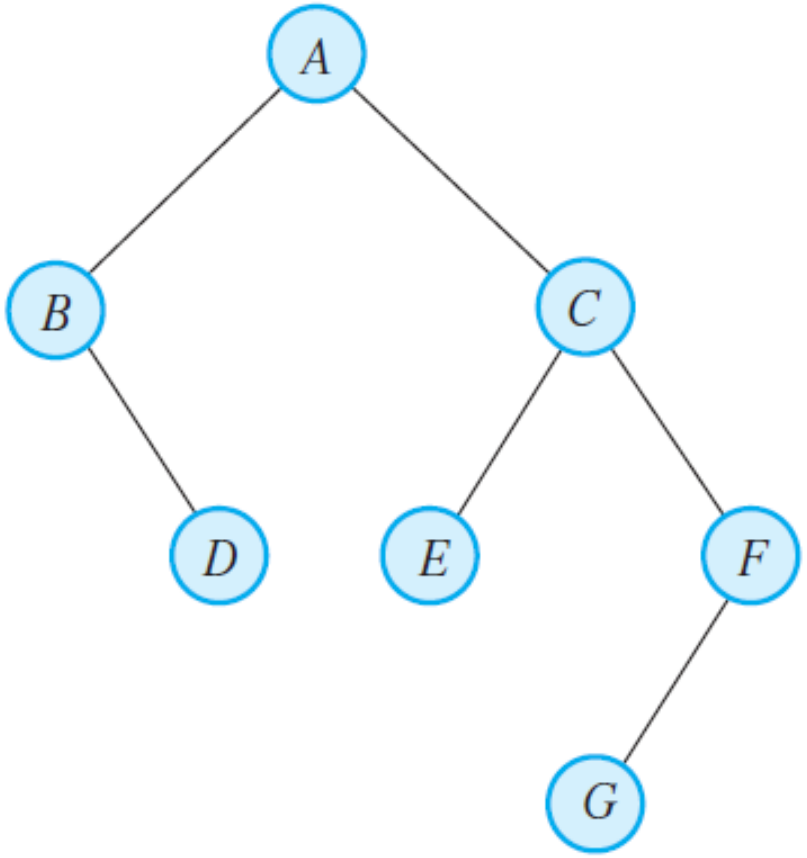
이진 트리의 탐방

중순위, 전순위, 후순위 탐방의 순서는 다음과 같다.



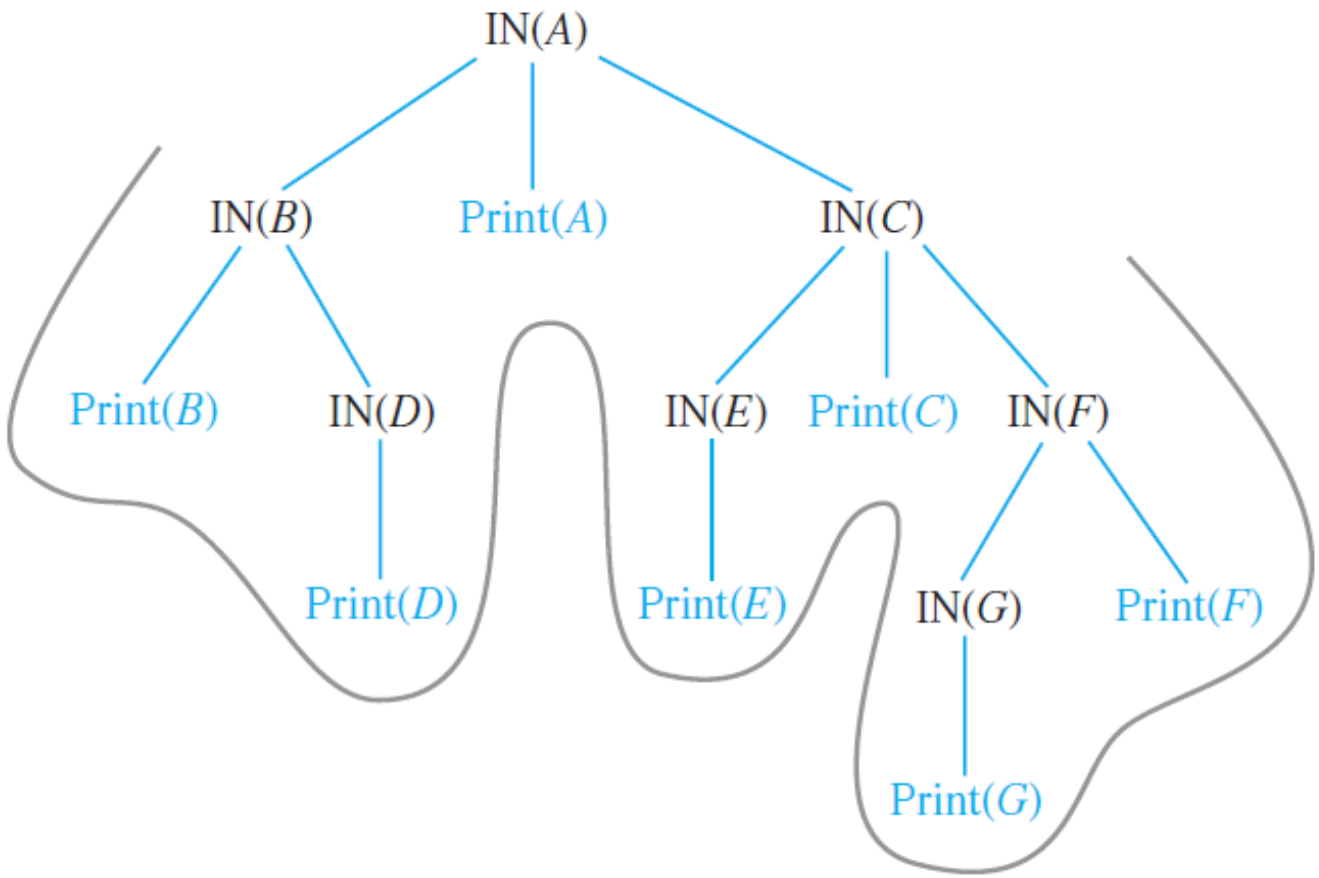
- 중순위 : 
- 전순위 : 
- 후순위 : 

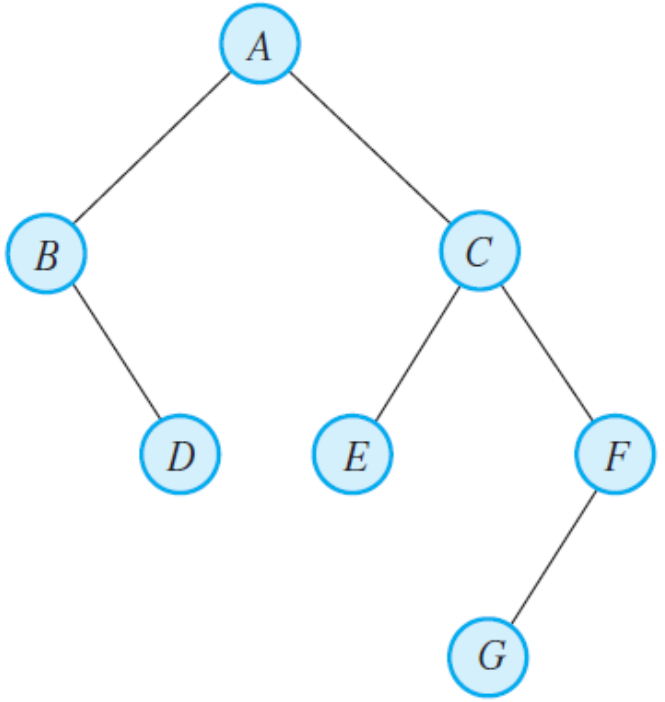
예3 다음의 이진 트리에서 중순위 탐방, 전순위 탐방, 후순위 탐방의 결과를 각각 구해보자.



이진 트리의 탐방

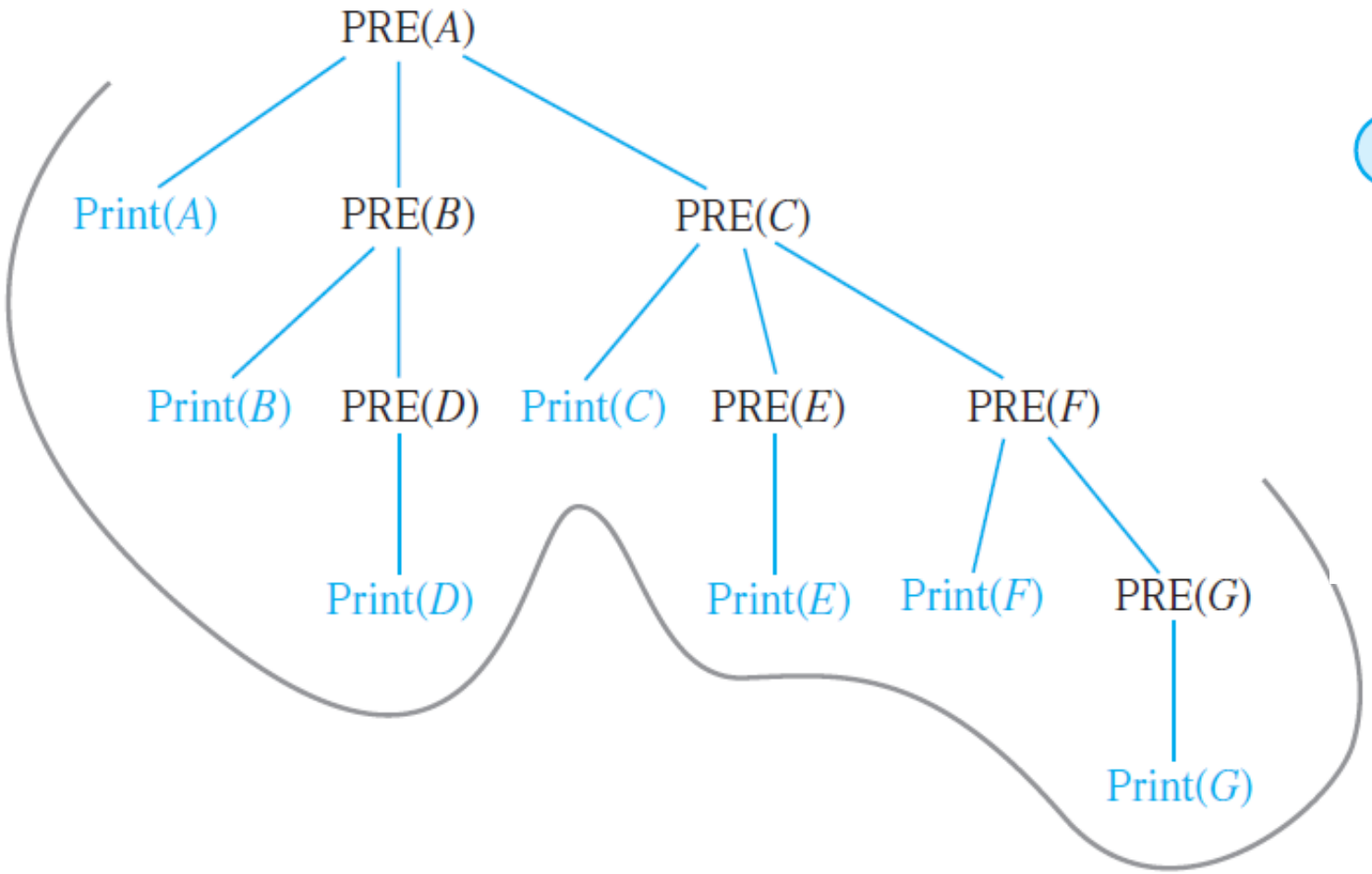
풀이 중순위 탐방의 결과는 위의 알고리즘에 따라 B, D, A, E, C, G, F이다.

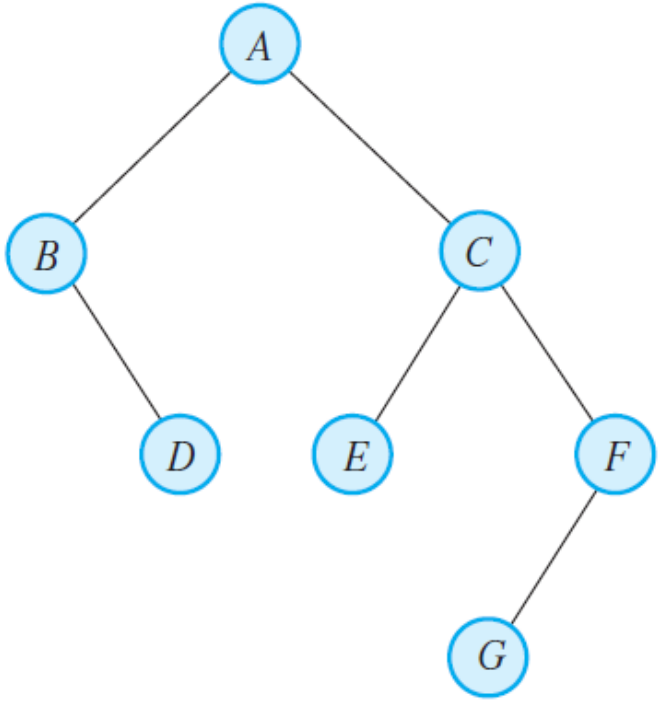




이진 트리의 탐방

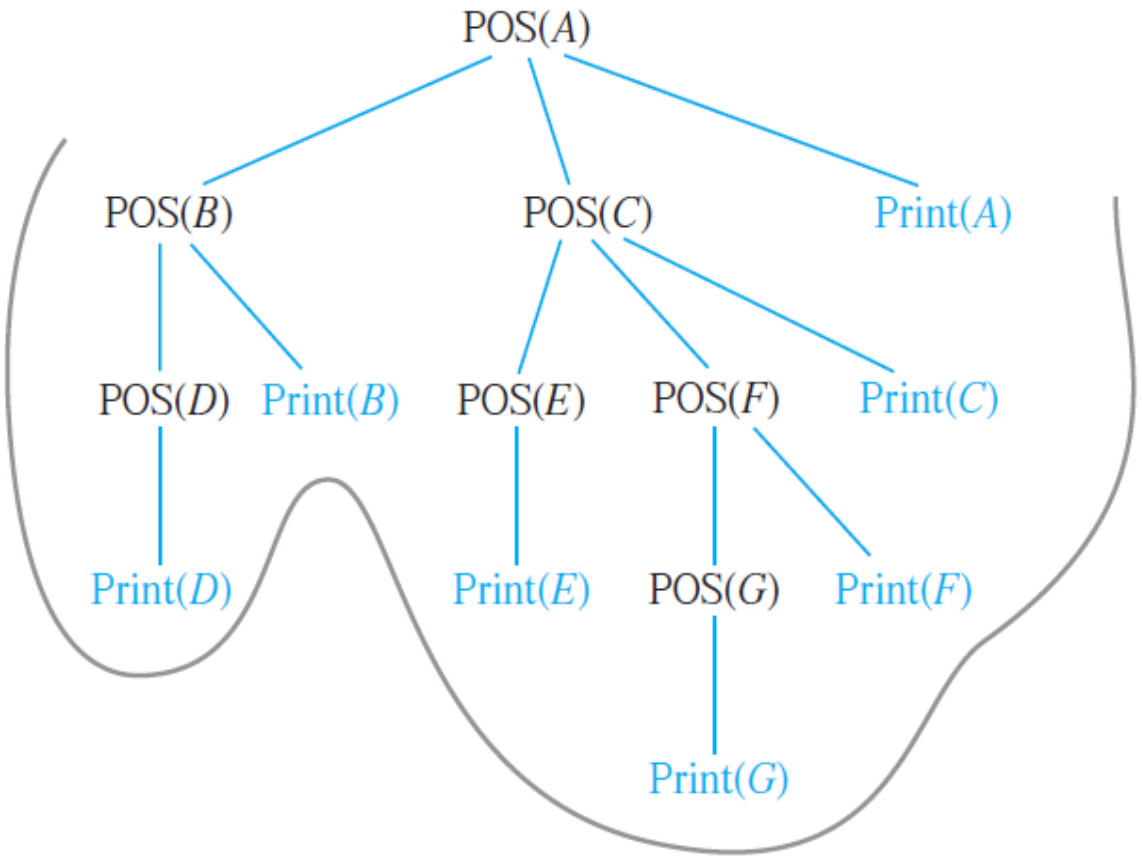
전순위 탐방의 결과는 위의 알고리즘에 따라 A, B, D, C, E, F, G이다.

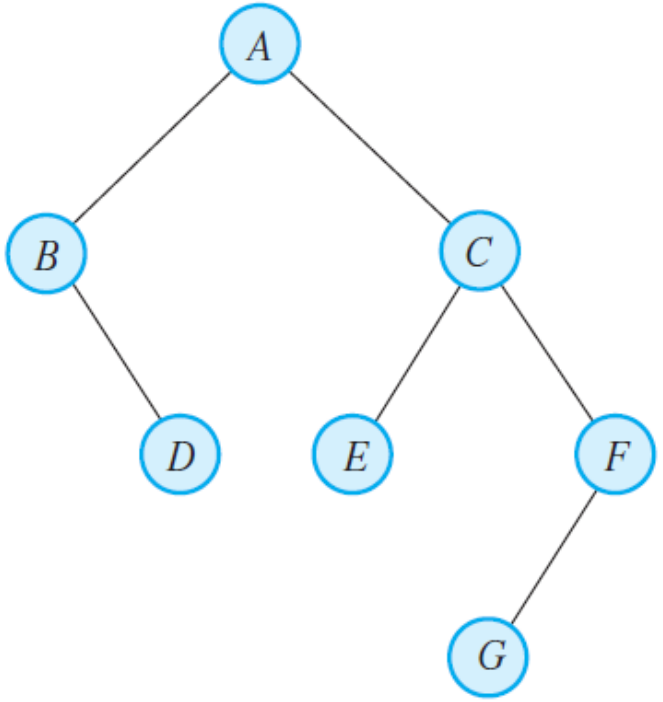




이진 트리의 탐방

후순위 탐방의 결과는 위의 알고리즘에 따라 D, B, E, G, F, C, A이다.

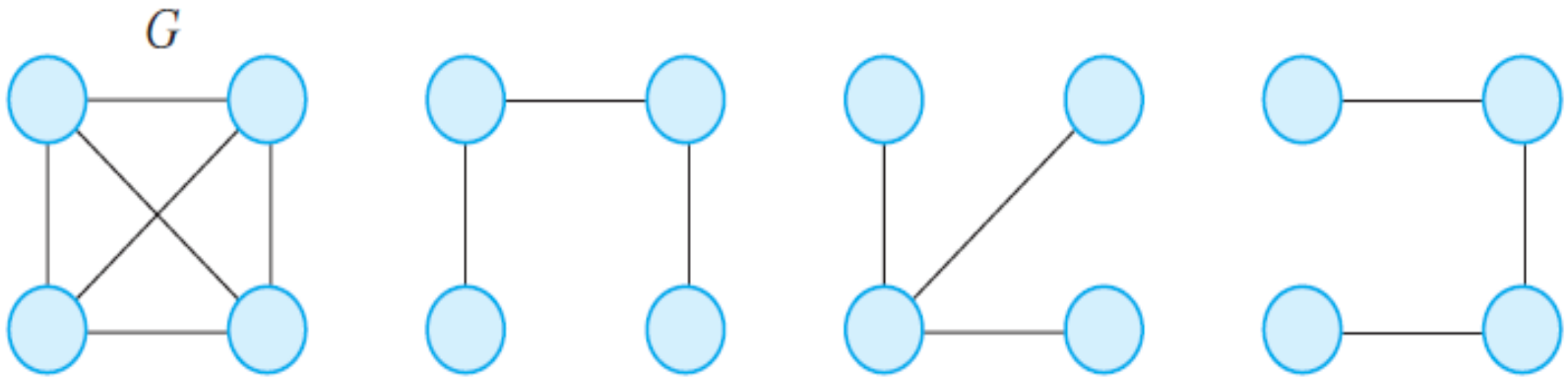




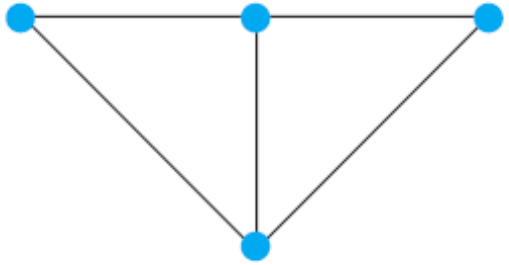
생성 트리와 최소비용 생성 트리

어떤 그래프 G에서 모든 노드들을 포함하는 트리를 생성 트리(spanning tree)라고 한다.

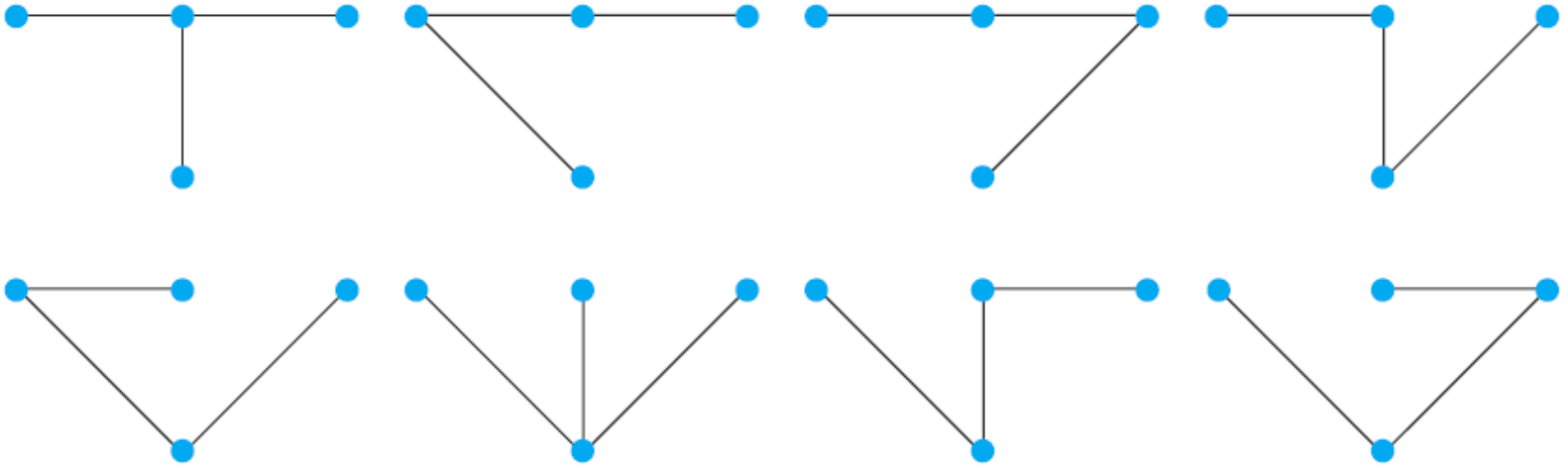
주어진 그래프 G와 그것의 3가지 생성 트리들



예 4 다음과 같은 그래프 G의 생성 트리를 모두 구해보자.



풀이 G의 생성 트리는 모두 8가지인데 다음과 같다.



생성 트리와 최소비용 생성 트리

생성 트리의 비용(cost)은 트리 연결선의 값을 모두 합한 값이다.

최소 비용 생성 트리(Minimum Spanning Tree : MST)란 최소한의 비용을 가지는 생성 트리를 말한다.

- 최소 비용 생성 트리의 대표적인 예는 통신 네트워크의 연결임
- 각 도시들을 연결하는 데 있어서 최소 비용으로 연결하는 방법을 찾는 것임
- 최소 비용 생성 트리의 대표적인 2가지 방법은 프림(Prim)의 알고리즘과 크루스칼(Kruskal)의 알고리즘임

생성 트리와 최소비용 생성 트리

알고리즘 1 프림의 알고리즘(Prim's algorithm)

주어진 가중 그래프 $G = (V, E)$에서 $V = \{1, 2, \cdots, n\}$이라고 하자.

(1) 노드의 집합 U를 1로 시작한다.

(2) $u \in U$, $v \in V - U$일 때 U와 $V - U$를 연결하는 가장 짧은 연결선인 (u, v)를 찾아서 v를 U에 포함시킨다. 이때 (u, v)는 사이클(cycle)을 형성하지 않는 것이라야 한다.

(3) (2)의 과정을 $U = V$ 때까지 반복한다.

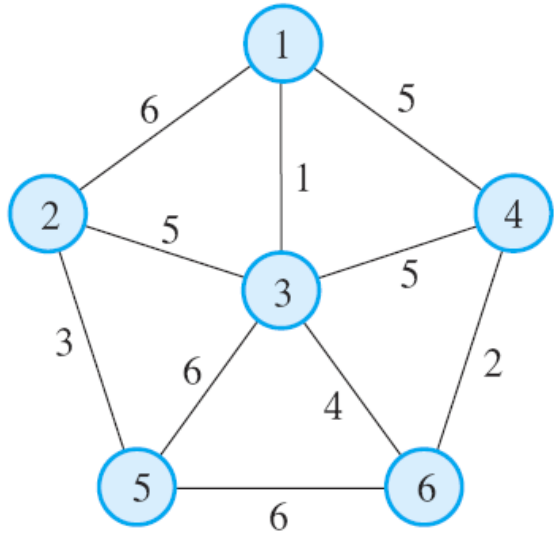
생성 트리와 최소비용 생성 트리

```
void Prim(graph G: set_of_edges T)
    /* 프림의 알고리즘은 G에 대한 최소 비용 생성 트리를 만든다. */
{
    set_of_vertices U;
    vertex u, v;
    T = NULL
    U = {1};
    while( U != V )
    {
        let (u, v) be a lowest cost edge such that u is in U and v is in V - U;
        T = T ∪ {(u, v)};
        U = U ∪ {v};
    }
} /* Prim */
```

생성 트리와 최소비용 생성 트리

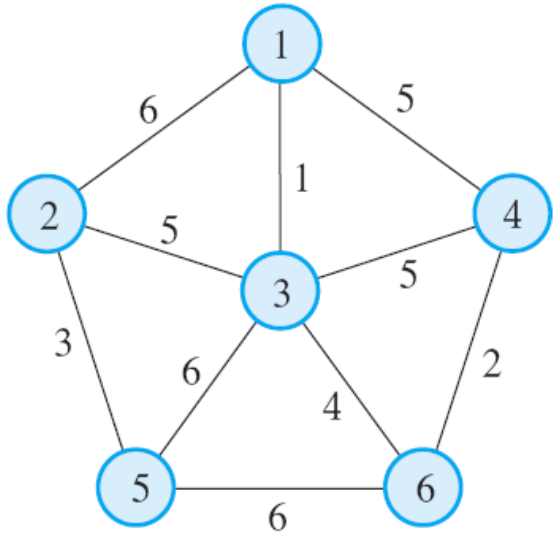
크루스칼 알고리즘의 쉬운 해설

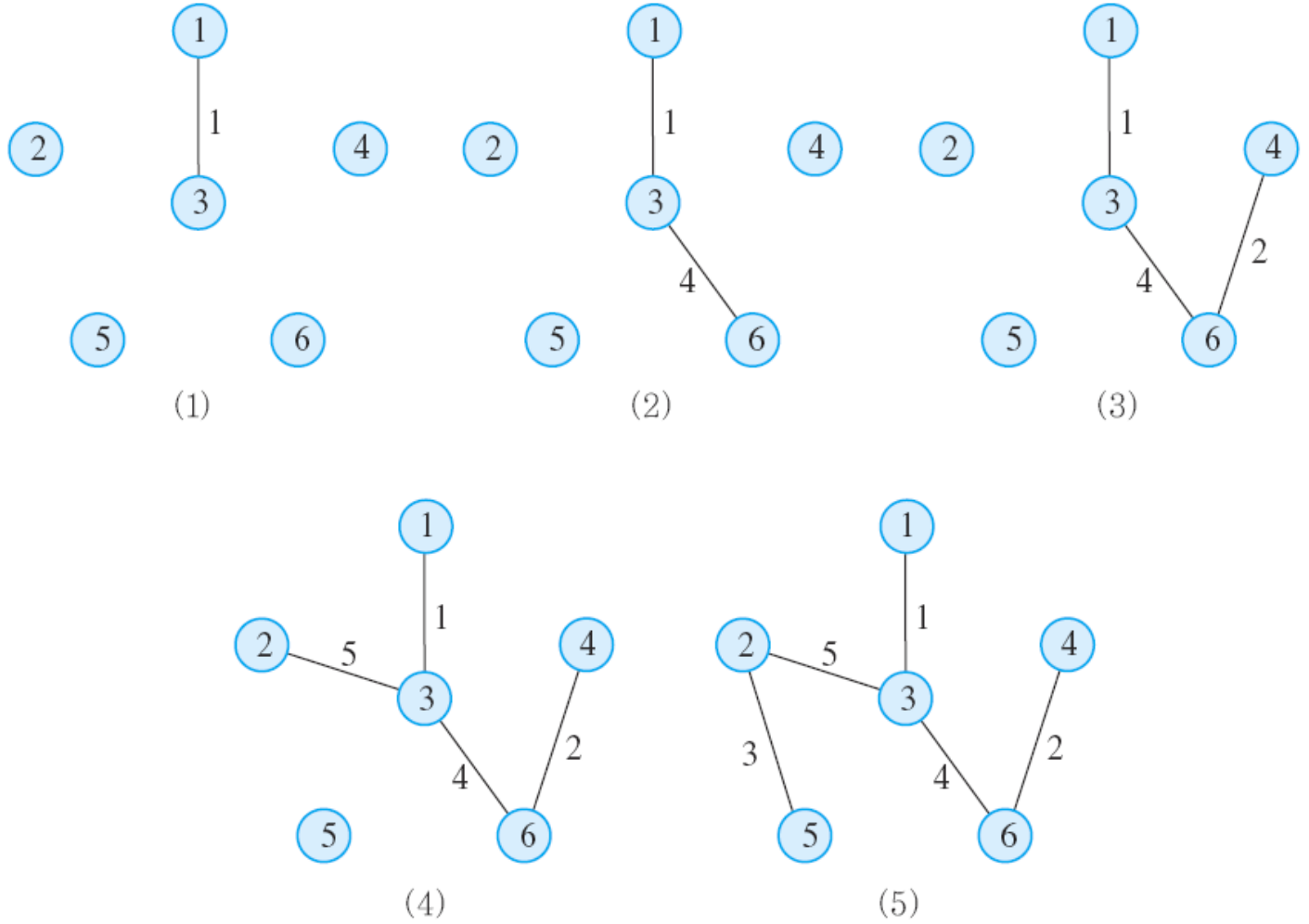
먼저 연결선들을 비용이 적은 순서대로 나열한다. 그 후 연결선을 제거한 그래프에서 연결선의 비용이 적은 연결선의 순서로 사이클이 생기지 않도록 연결해 나가면 된다. 만약 비용이 같은 경우에는 어떤 순서로 적용해도 무방하다.



가중 그래프

생성 트리와 최소비용 생성 트리





프림의 알고리즘에 의한 MST의 생성 과정

생성 트리와 최소비용 생성 트리

알고리즘 2 크루스칼의 알고리즘(Kruskal's algorithm)

주어진 가중 그래프 $G = (V, E)$에서 $V = \{1, 2, \cdots, n\}$이라고 하고 T를 연결선의 집합이라고 하자.

(1) T를 ϕ으로 놓는다.

(2) 연결선의 집합 E를 비용이 적은 순서로 정렬한다.

(3) 가장 최소값을 가진 연결선 (u, v)를 차례로 찾아서 (u, v)가 사이클을 이루지 않으면 (u, v)를 T에 포함시킨다.

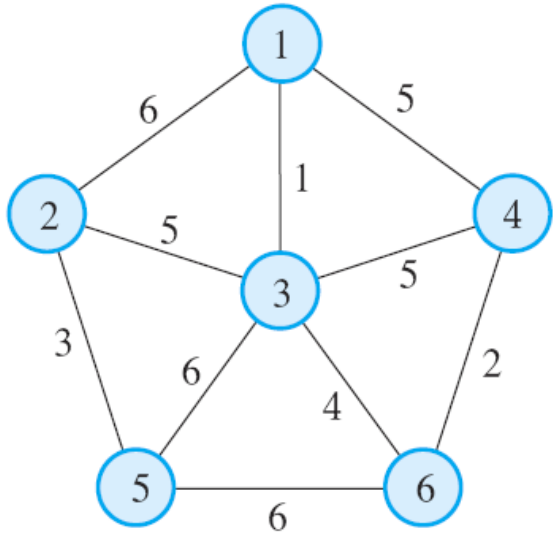
(4) (3)의 과정을 $|T| = |V| - 1$일 때까지 반복한다.

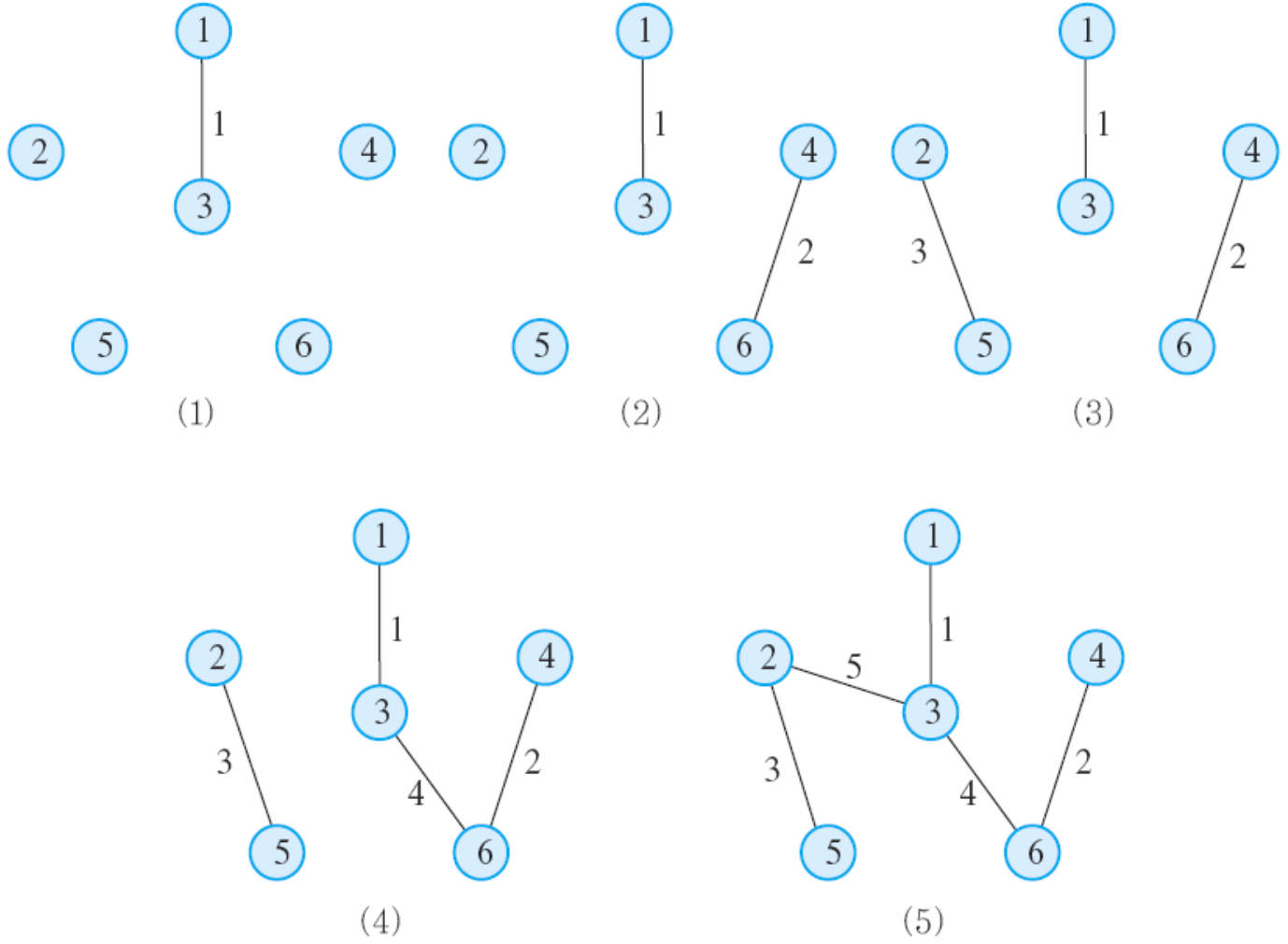
생성 트리와 최소비용 생성 트리

```
T = NULL;
while(T contains less than n-1 edges and E not empty)
{
    choose an edge (v, w) from E of lowest cost;
    delete (v, w) from E;
    if ((v, w) does not create a cycle in T)
        add (v, w) to T;
    else discard (v, w);
}
if(T constains fewer than n-1 deges) printf("no spanning tree\n");
```

크루스칼의 알고리즘

생성 트리와 최소비용 생성 트리





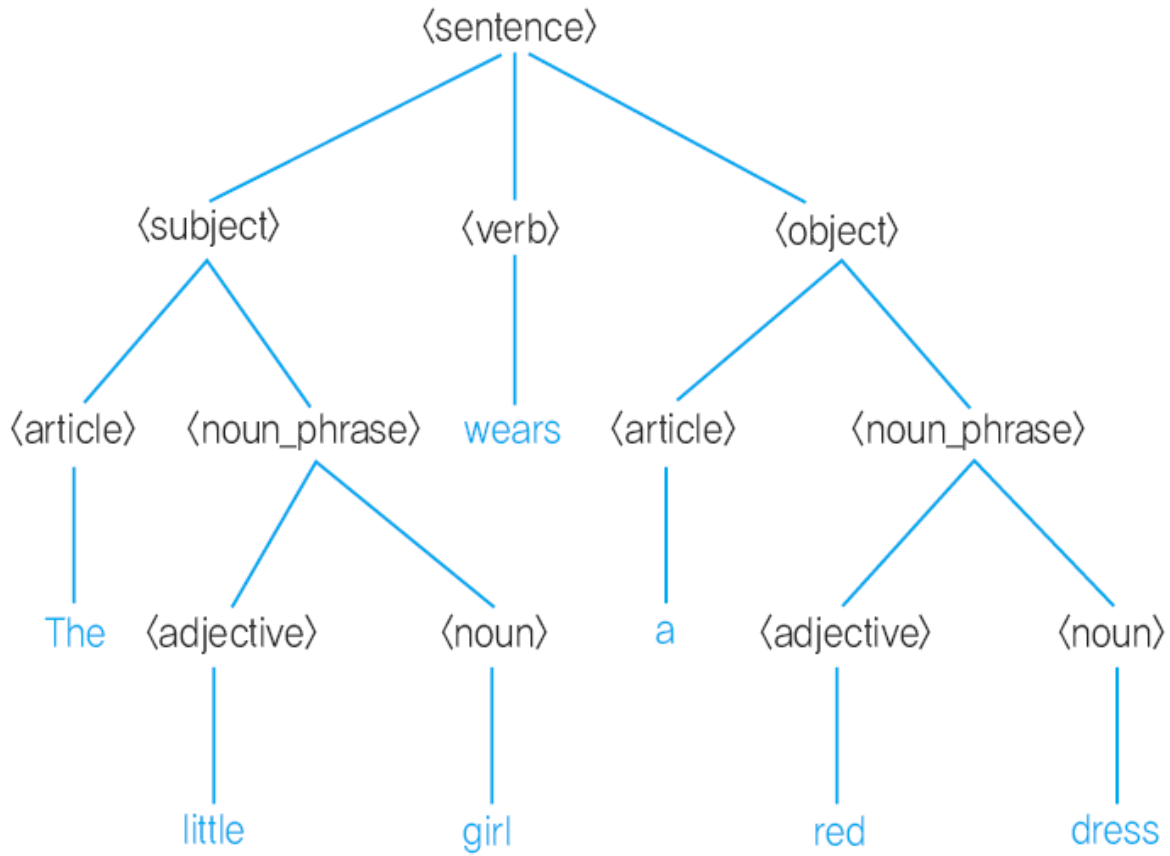
크루스칼의 알고리즘에 의한 MST의 생성 과정

트리의 활용

(1)문법의 파싱(parsing)

- 트리는 문법의 파싱을 통하여 자연어 처리와 컴파일러(compiler) 등에 활용됨
- 문장 트리에서 문장은 주어, 동사, 목적어로 이루어짐
- 주어는 관사와 명사구로 나누어지고 목적어는 관사와 명사구로 나누어짐
- 명사구는 형용사와 명사로 다시 나누어지는 파싱 과정을 반복함

트리의 활용


⟨sentence⟩
⟨subject⟩
⟨verb⟩
⟨object⟩
⟨article⟩
⟨noun_phrase⟩
wears
⟨article⟩
⟨noun_phrase⟩
The
⟨adjective⟩
⟨noun⟩
a
⟨adjective⟩
⟨noun⟩
little
girl
red
dress

문장 트리

트리의 활용

(2) 허프만 코드(Huffman code)

알파벳의 열(sequence)로서 이루어진 메시지가 있고, 각 메시지의 영문자가 각각 독립적이고 위치에 관계없이 어떤 정해진 확률로 나타남

예 : 5개의 영문자 a, b, c, d, e가 나타날 확률이 각각 0.12, 0.4, 0.15, 0.08, 0.25라고 할 때, 이 영문자를 각각 0과 1의 열로서 코드화하고자 하면, 어느 영문자의 코드도 다른 어떤 영문자의 코드의 접두어로 표현되어서는 안됨

a가 01이고 b가 010일 때 a는 b의 접두어가 되므로 적합하지 않음

트리의 활용

(3) 결정 트리(decision tree)

- 트리를 이용할 때 매우 유용하게 쓰이는 것은 결정 트리임
- 가능성 있는 경우의 수가 너무나 많기 때문에 모든 면에서 입증하기가 매우 어려운 문제를 만날 수가 있음
- 결정 트리를 활용하면 주어진 문제를 일목요연하게 입증할 수 있음

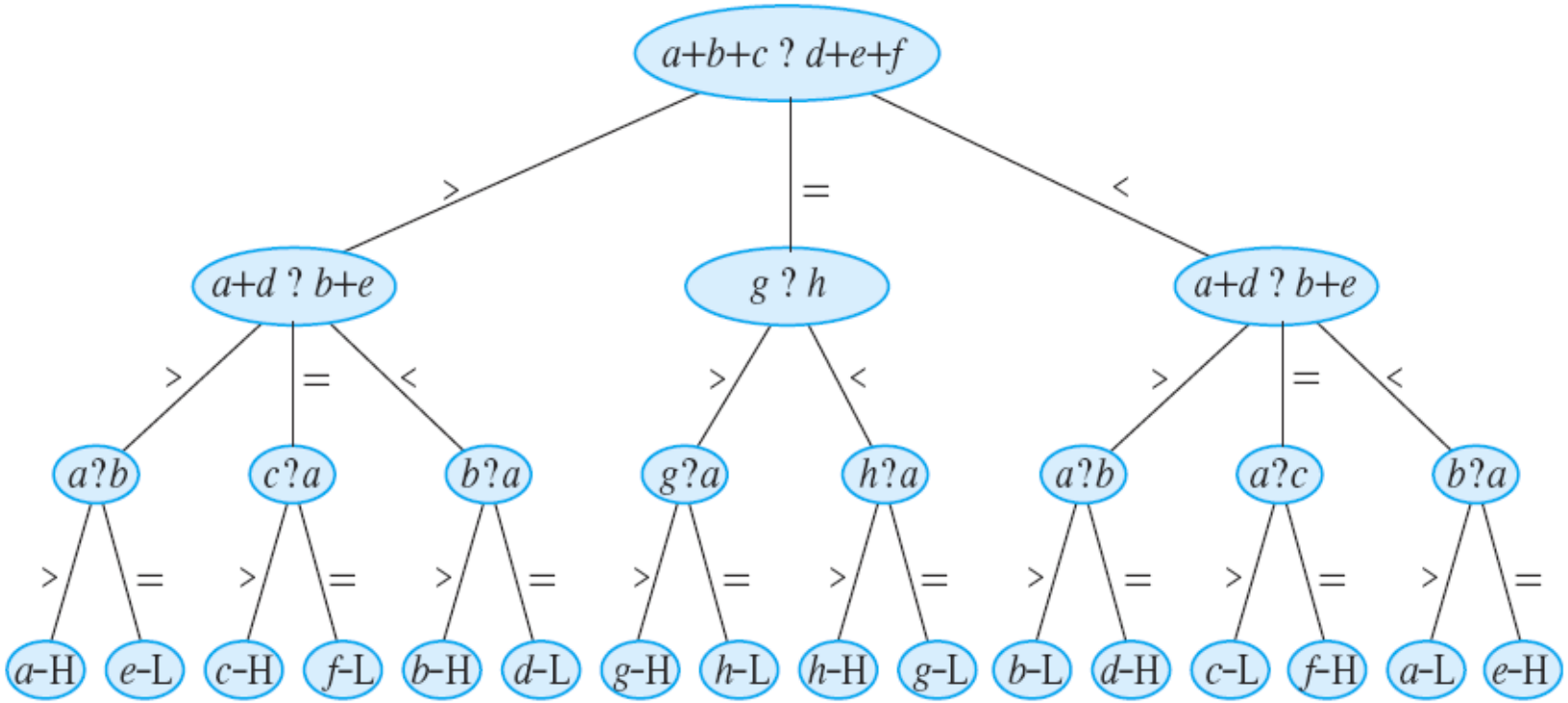
8개의 동전 문제(eight-coins problem)

- 결정 트리의 예로 잘 알려져 있음
- 크기와 색깔이 똑같은 8개의 동전 *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h* 중에서 한 개는 불량품이어서 다른 동전들과는 다른 무게를 가짐
- 무게가 같거나 크고 작은 것만을 판단할 수 있는 하나의 천칭 저울을 사용하여 단 세 번만의 계량으로 어느 동전이 불량품이고, 다른 동전보다 무겁거나 가벼운지를 동시에 판단하고자 함

트리의 활용

- a+b+c < d+e+f인 경우 불량품이 6개 가운데 있으며 이때의 g와 h는 정상품이라는 것을 알 수 있음
- 다음 단계에서 d와 b를 양 천칭 저울에서 바꾸어 계량한 결과 a+d < b+e로 부등호에 변화가 없다면 다음의 2가지 가능성을 말해줌
- a가 불량품이거나 e가 불량품인 경우임
- 이런 경우는 정상품과 비교하여 H나 L을 판단할 수 있음
- 만약 a+d = b+e인 경우에는 c나 f가 불량품임
- 또한 b나 d가 불량품인 경우에는 다른 정상품과 비교하여 b 또는 d가 H나 L이 되게 결정됨

트리의 활용



8개의 동전 문제 결정 트리

트리의 활용

- 최종 판단을 위한 프로시저인 comp의 프로그램임
- 프로시저 eight-coins는 앞의 결정 트리의 판단과 같은 기능을 수행함

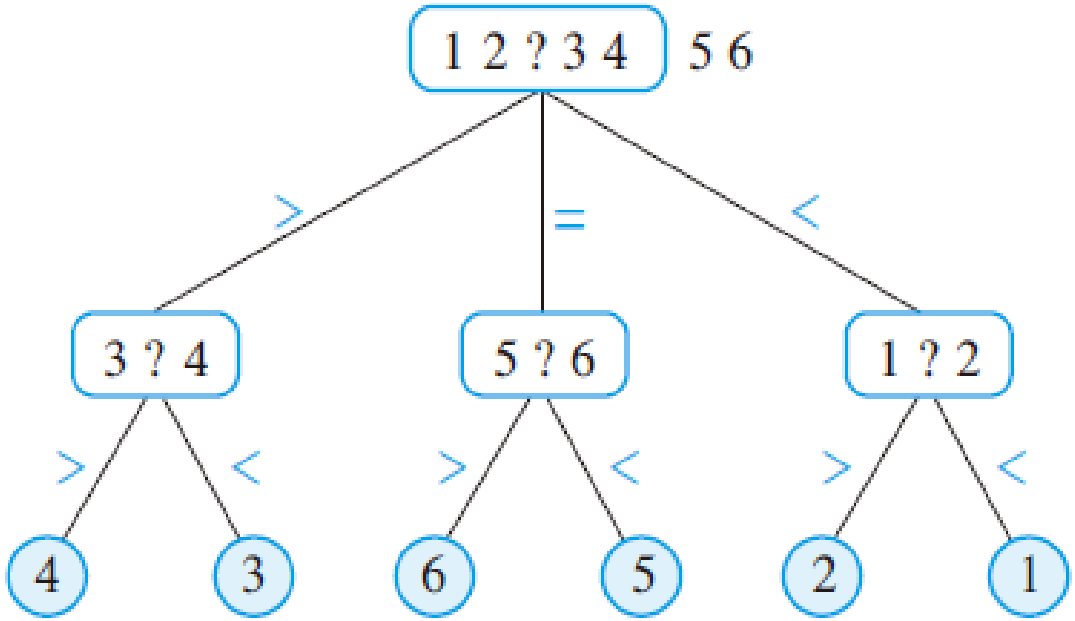
```
void comp(int x, int y, int z)
    /* x를 정상품인 z와 비교한다 */
{
    if (x > z) printf("%d heavy \n", x);
    else printf("%d light \n", y);
}  /* comp */
```

트리의 활용

```
void eight-coins()
   /* 8개의 동전 무게가 입력되면 단 세 번의 비교로써 비정상적인 동전을 발견할 수 있다. */
{
   int a, b, c, d, e, f, g, h;
   scanf("%d %d %d %d %d %d %d %d", &a, &b, &c, &d, &e, &f, &g, &h);
   switch(compare(a+b+c, d+e+f)) {
      case '=' : if(g > h) comp(g, h, a);
                 else comp(h, g, a);
                 break;
      case '>' : switch(compare(a+d, b+e)) {
                 case '=' : comp(c, f, a);
                 case '>' : comp(a, e, b);
                 case '<' : comp(b, d, a);
                }
                break;
      case '<' : switch(compare(a+d, b+e)) {
                 case '=' : comp(f, c, a);
                 case '>' : comp(d, b, a);
                 case '<' : comp(e, a, b);
                }
   }  /* switch */
}  /* eightcoins */
```

예 5

크기와 모양이 똑같은 6개의 공이 있는데 이 중 5개의 무게는 같고 한 개는 약간 더 가볍다고 한다. 천칭 저울을 이용하여 최소 몇 번의 측정으로 1개의 가벼운 공을 찾아낼 수 있을까? 게임 트리를 만들어 찾아내시오.



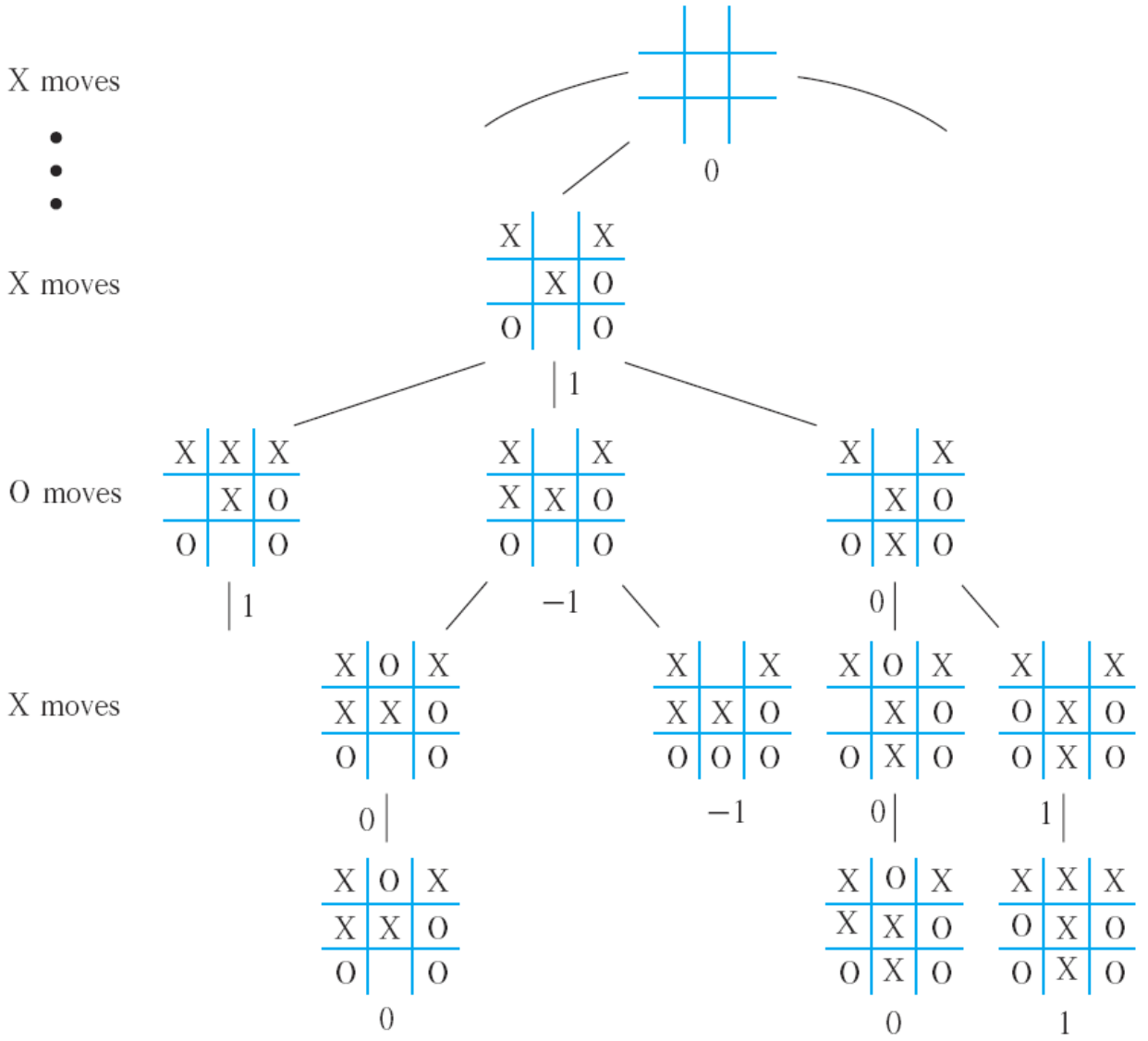
가벼운 공을 찾아내는 결정 트리

트리의 활용

(4) 게임(game)

- 트리는 체스, 틱텍토(tic-tac-toe), 장기, 바둑 등 게임에 있어서의 진행과 전략을 구사할 수 있는 게임 트리로도 활용됨
- tic-tac-toe 게임 : 가로, 세로, 대각선으로 연속된 세 개를 놓으면 이기는 게임

트리의 활용



tic-tac-toe 게임 트리

트리의 활용

게임 트리를 이용하는 또 다른 응용으로는 인공지능 문제를 해결하는 8-puzzle 게임이 있다. 이 게임은 〈그림 8.21〉과 같이 3×3 크기의 박스 안에서 인접한 숫자를 빈 곳으로 계속적으로 움직여서 목표 상태(goal)로 만드는 서양식 게임이다.

Start

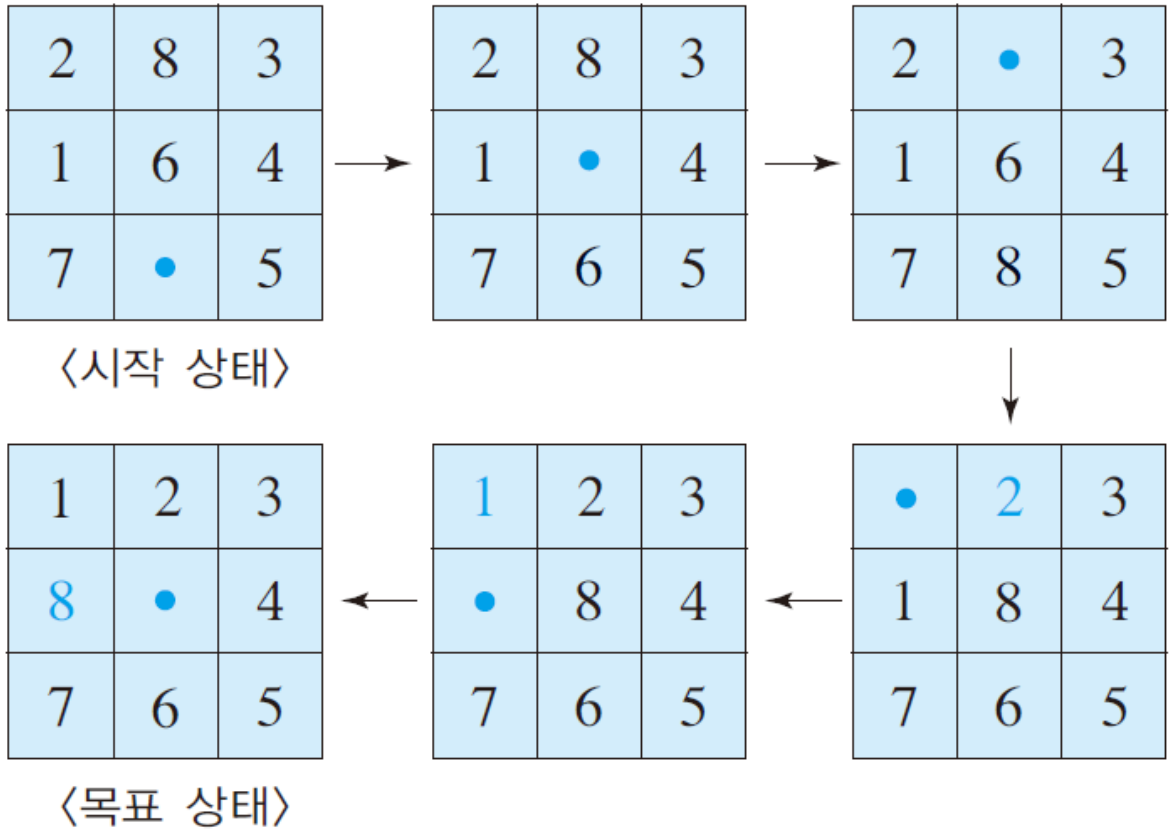
2	8	3
1	6	4
7	•	5

Goal

1	2	3
8	•	4
7	6	5

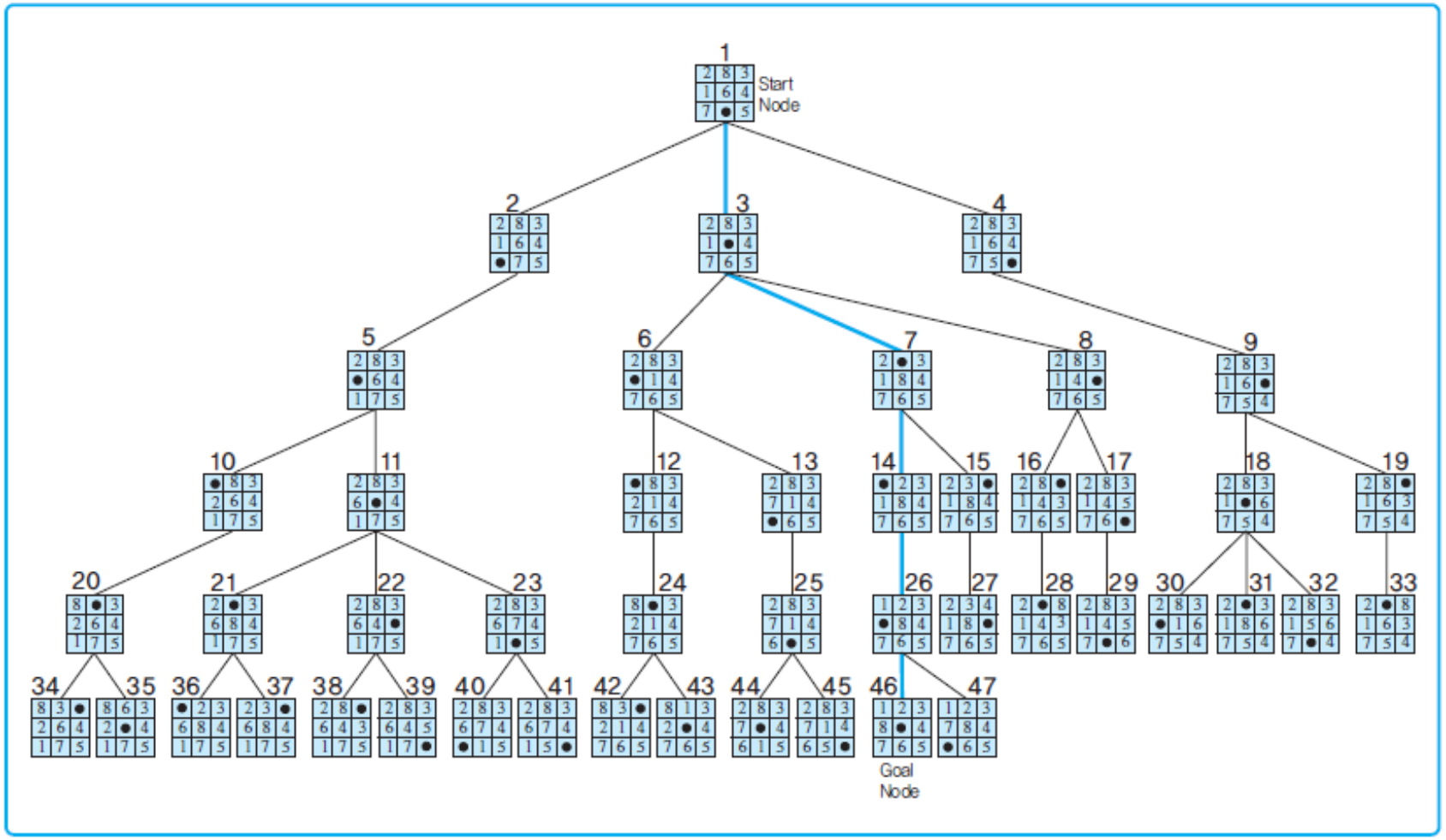
시작 상태와 목표 상태

트리의 활용



시작 상태에서 목표 상태까지의 중간 상태 이동

트리의 활용



8-puzzle 트리