제 9장 Trees

9.1 Introduction

9.2 Terminology and Characterizations of Trees

9.3 Spanning Trees

9.4 Minimal Spanning Trees

9.5 Binary Trees

9.6 Tree Traversals

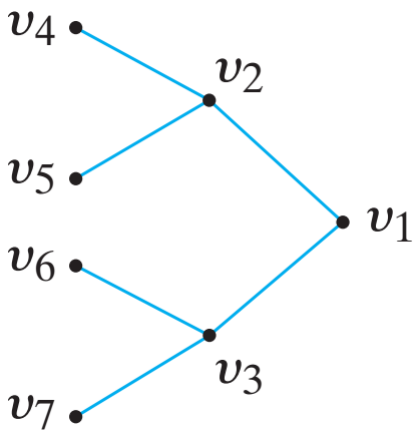
9.7 Decision Trees and the Minimum Time for Sorting

~~9.8 Insomorphisms of Trees~~

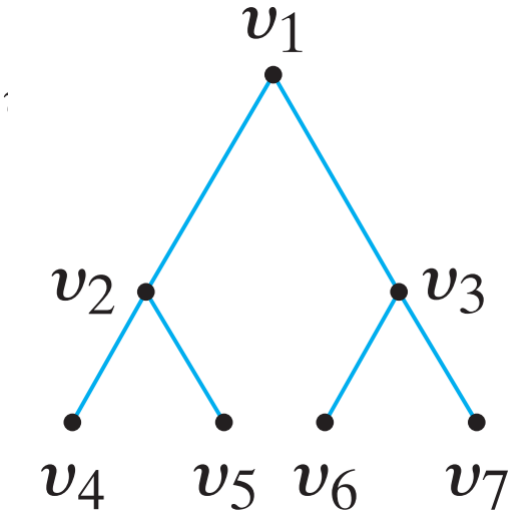
~~9.9 Game Trees~~

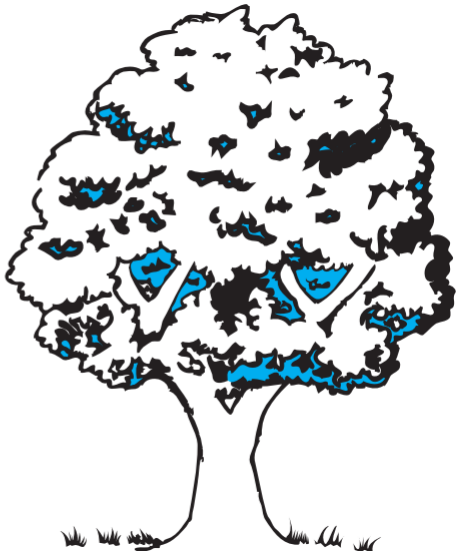
9.1 Introduction

- 토너먼트를 나타낸 트리



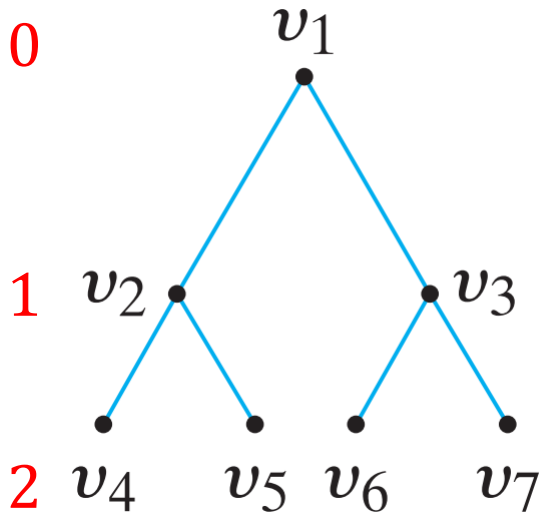
- 트리를 회전한 것과 나무와 비교





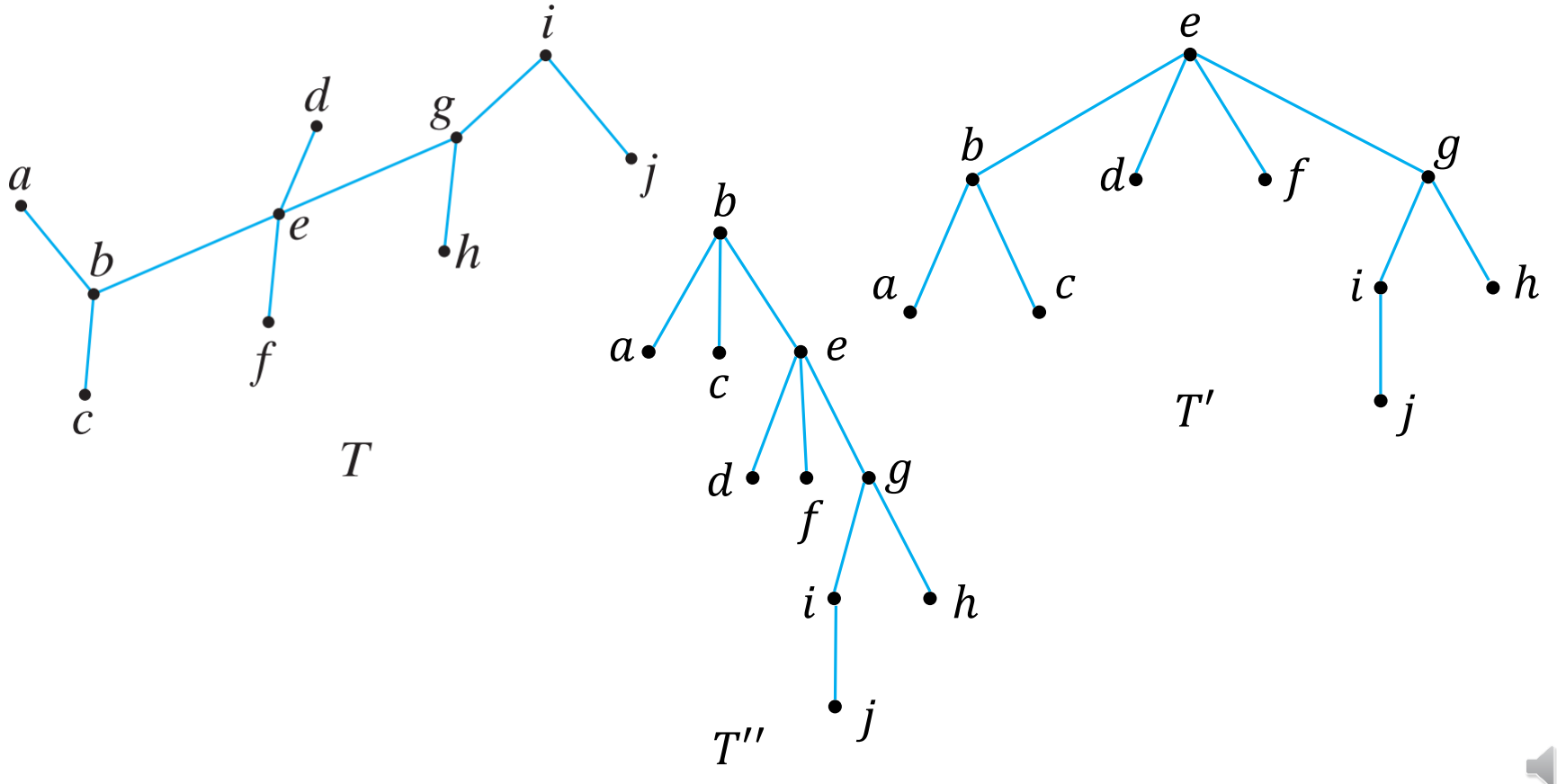
9.1 Introduction(트리, 근트리)

- 정의 9.1.1 **트리**tree T는 다음을 만족시키는 단순 그래프이다. v와 w가 T의 정점들이면, v에서 w로의 유일한 단순 경로가 존재한다.
  근 트리rooted tree는 특별한 정점이 뿌리로 지정된 트리이다.
- 어떤 정점 v의 **레벨**level은 뿌리에서 v로의 단순 경로의 길이가 된다. 근 트리의 **높이**height 는 최대 레벨 값이 된다



9.1 Introduction

- 예제 9.1.4 트리 T에서 e를 뿌리로 하면, 근 트리 T'이 된다. 정점 $a, b, c, d, e, f, g, h, i, j$의 레벨은 각각 2, 1, 2, 1, 0, 1, 1, 2, 2, 3이 된다. T'의 높이는 3 이다. 트리 T에서 b를 뿌리로 하면, 근 트리 T''이 된다.



9.1 Introduction

- 예제 9.1.8 허프만 코드Huffman codes 문자를 가변 길이의 비트 문자열로 표현한다. 아이디어는 많이 사용되는 문자들을 표현하기 위해서는 짧은 길이의 비트 문자열을 사용하고, 적게 사용되는 글자들에 대해서는 긴 길이의 비트 문자열을 사용하는 것이다.
- 허프만 코드는 근 트리에 의해 쉽게 정의된다. 비트 문자열을 디코딩하기 위해서는 뿌리에서 시작하여문자를 만날 때까지 아래로 내려간다.
- 예, 01010111는 다음으로 해석:

$$RAT$$

- 트리에 의해 정의된 허프만 코드에서, A 는 1, O는 00, R는 010, T는 0111, S는 0110로 표현된다.

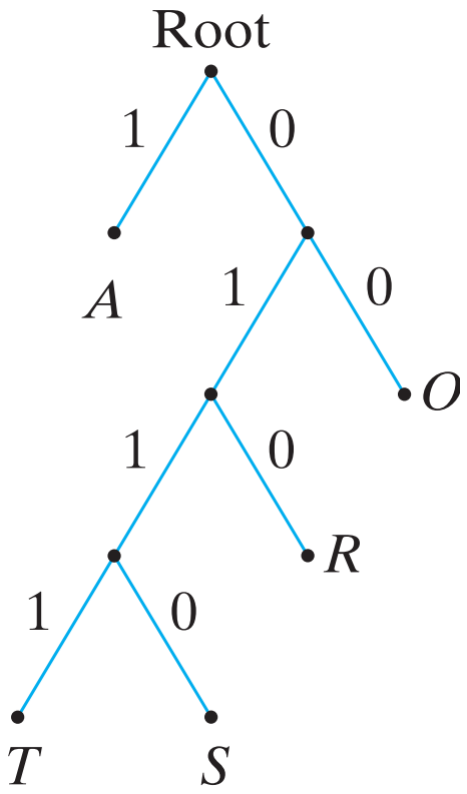


그림 1.10

9.1 Introduction

- Algorithm 9.1.9 최적의 허프만 코드 구성
  이 알고리즘은 표현되어야 하는 문자들의 발생 빈도가 나와 있는 표로부터 최적의 허프만 코드를 구성한다.
  출력은 그림 1.10과 같이 최하위 레벨에 빈도가 표기된 정점들이 있고, 비트들로 표기된 간선들이 있는 근 트리이다.
  코딩 트리 는 각각의 발생 빈도를 그 발생 빈도를 갖는 문자들로 바꿈으로써 얻을 수 있다.
- 표에 있는 문자들의 빈도와 동일한 문자 빈도를 갖는다면,
  이 알고리즘에 의해 구성되는 코드는 최소 공간으로 문자열들을 표현한다.

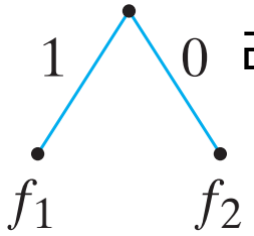
9.1 Introduction

- Algorithm 9.1.9 최적의 허프만 코드
- Input: n개의 빈도에 대한 수열, $n \geq 2$

  Output: 최적의 허프만 코드를 정의한 근트리

  $huffman(f, n)$ {

  if $(n == 2)$ {

  f_1과 f_2는 빈도를 나타내고, T를 

라 하자.

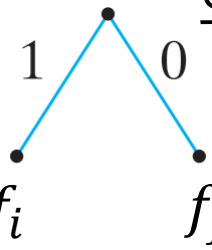
  return T

  }

  f_i와 f_j가 가장 작은 빈도를 나타낸다고 하자.

  리스트 f에서 f_i와 f_j를 $f_i + f_j$로 표기를 바꾼다.

  $T' = huffman(f, n - 1)$

  T'에서 $f_i + f_j$로 표기된 정점을 

와 같은 트리로 바꾸어 트리 T 를 구한다.

  return T

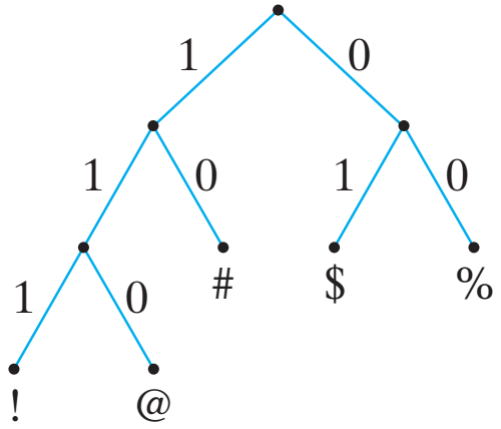
  }

9.1 Introduction

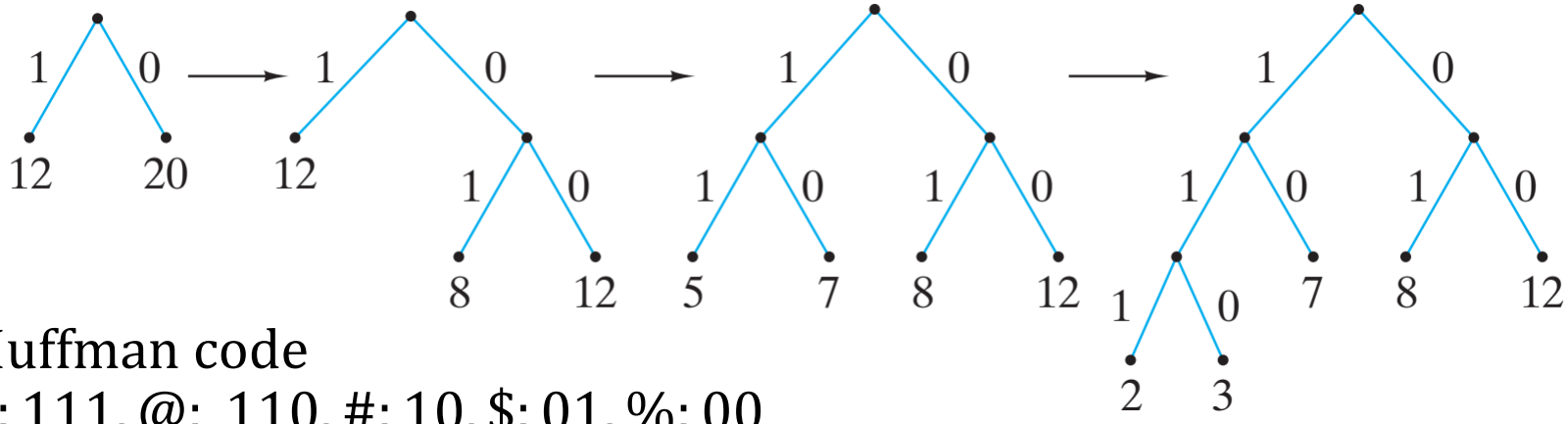
- 예제 9.1.10 표를 사용하여 최적의 허프만 코드를 구성
- 풀이

문자	빈도
!	2
@	3
#	7
$	8
%	12

$$2, 3, 7, 8, 12 \rightarrow 2 + 3, 7, 8, 12$$
$$5, 7, 8, 12 \rightarrow 5 + 7, 8, 12$$
$$8, 12, 12 \rightarrow 8 + 12, 12$$
$$12, 20$$



- Tree



- Huffman code
  !: 111, @: 110, #: 10, $: 01, %: 00

9.2 Terminology and Characterizations of Trees

- 정의 9.2.1 T를 뿌리 v_0를 가진 트리라고 하자. x, y, z 는 T의 정점들이고, $(v_0, v_1, \dots, v_n)$은 T에서의 단순 경로라 하자.

$a)$ v_{n-1}은 v_n의 **부모**parent

$b)$ $v_0, \dots, v_{n-1}$은 v_n의 **조상**ancestor

$c)$ v_n은 v_{n-1}의 **자식**child

$d)$ 만약 x가 y의 조상이라면, y는 x의 **후손**descendant

$e)$ 만약 x와 y가 z의 자식이라면, x와 y는 **형제**sibling

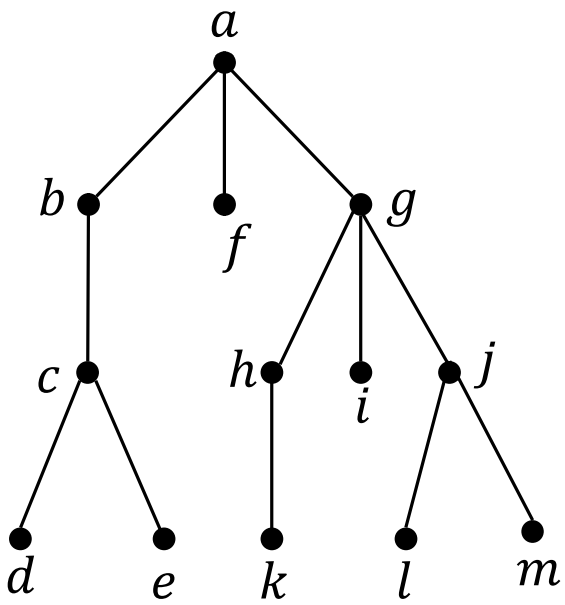
$f)$ 만약 x가 자식을 갖지 않았다면, x는 **말단 정점**terminal vertex, leaf

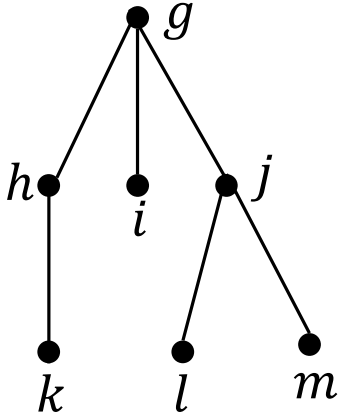
$g)$ 만약 x가 말단 정점이 아니면, x는 **내부 정점**internal vertex

$h)$ x를 뿌리로 하는 T의 **부분 트리**subtree는 정점의 집합 V와 간선의 집합 E를 갖는 그래프이다.
V는 x와 x의 후손들이다.
$E = \{e | e$는 x에서 V의 어떤 정점까지의 단순 경로 상의 간선$\}$

9.2 Terminology and Characterizations of Trees

예제





뿌리가 g인 부분 트리

- c의 부모는 b.
  g의 자식은 h, i, j.
  h의 형제는 $i,\ j$.
  e의 조상은 c, b, a.
  b의 후손은 c, d, e.
- 내부 정점은 a, b, c, g, h, j.
  말단 정점은 $d, e, f\ , i, k, l, m$.

9.2 Terminology and Characterizations of Trees

- *cycle*: nonzero length path v to v, no repeated edges.
- *simple path*: no repeated vertices.
- *simple cycle*: cycle from v to v except for the beginning and ending vertices, no repeated vertices
- *simple graph*: neither loops nor parallel edges
- *connected graph*: $\forall v, w \in V$, there is a path from v to w
- *tree*: simple graph, $\forall v, w \in V$, there is a unique simple path from v to w
- *component of G containing v*: The subgraph of G consisting of all edges and vertices in G that are contained in some path beginning at v
- A graph is connected iff it has exactly one component
- A graph with no cycles is called an **acyclic graph**.

9.2 Terminology and Characterizations of Trees

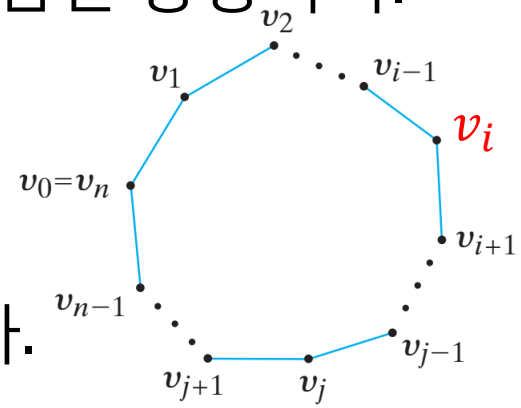
- 정리 9.2.3 T를 n개의 정점을 가진 그래프. 다음은 동등하다.

(a) T는 트리이다.

(b) T는 연결되어 있고 비순환적이다.

(c) T는 연결되고 $n-1$개의 간선을 갖는다.

(d) T는 비순환적이고 $n-1$개의 간선을 갖는다.



증명 $(a) \Rightarrow (b)$

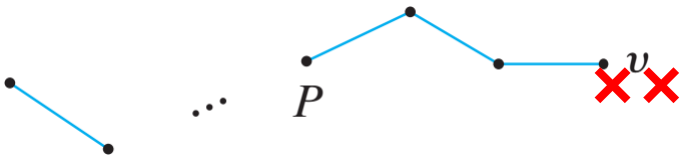
- 트리에서는 임의의 정점에서 다른 모든 정점으로의 단순 경로가 있으므로, T가 연결되어 있다.
- T가 사이클 C'을 포함한다고 하자. 정리 8.2.24 에 의해서 T는 $v_0 = v_n$인 단순 사이클 $C = (v_0, \dots, v_n)$을 포함한다(그림↑). T는 단순 그래프이므로 C는 루프가 될 수 없다. 그러므로 C는 적어도 $i < j$ 인 2개의 구별되는 정점 v_i 와 v_j를 포함한다. 이제

$$(v_i, v_{i+1}, \dots, v_j), \qquad (v_i, v_{i-1}, \dots, v_0, v_{n-1}, \dots, v_j)$$

는 v_i에서 v_j로의 서로 다른 단순 경로들이다. 이것은 트리의 정의에 모순 된다. 그러므로 트리는 사이클을 포함할 수 없다.

9.2 Terminology and Characterizations of Trees

- $(b) \Rightarrow (c)$ T가 연결되어 있고 비순환적이라고 가정한다. T가 $n-1$개의 간선을 갖는다는 것을 정점의 수 n에 대한 귀납법으로 증명한다.
- 만약 $n = 1$이면, T는 간선은 없고 단지 하나의 정점으로 구성된다. 그러므로 $n = 1$인 경우에 성립한다.
- n개의 정점을 가진 연결된 비순환 그래프에서 성립함을 가정. T를 $n+1$개의 정점을 갖는 연결된 비순환 그래프라고 하자. 간선을 반복하지 않는 최대 길이의 경로 P를 선택한다. T가 비순환적이므로 P는 사이클이 없다. 그러므로 P는 차수가 1인 정점 v를 포함한다(그림).



T^*를 T에서 정점 v와 v에 결합된 간선을 제거한 것이라고 하자. 그러면 T^*는 연결되어 있고 비순환적이다.
또한 T^*는 n개의 정점을 가지므로 가정에 의해 T^*는 $n-1$개의 간선을 갖는다. 그러므로 T는 n개의 간선을 갖는다.

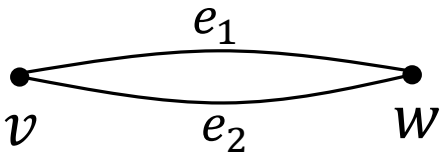
9.2 Terminology and Characterizations of Trees

- $(c) \Rightarrow (d)$ T가 연결되어 있고 $n-1$개 의 간선을 갖는다고 가정. T가 비순환적이라는 것을 보여야 한다.
- T가 적어도 하나의 사이클을 갖는다고 가정하자.
  사이클에서 하나의 간선을 제거하더라도 그래프는 계속 연결되어 있으므로, 결과 그래프인 T^*가 연결된 비순환 그래프가 될 때까지 T에서 사이클을 구성하는 간선들을 (정점들은 제거하지 않고) 제거할 수 있다.
  이제 T^* 는 n개의 정점을 가진 연결된 비순환 그래프이다.
  $(b) \Rightarrow (c)$에 의해, T^*가 $n-1$개의 간선을 갖는다.
  그러나 이제는 T가 $n-1$개보다 많은 수의 간선을 갖는다.
  이것은 모순이다. 그러므로 T는 비순환적이다.

(a) T는 트리이다
(b) T는 연결되어 있고 비순환적이다
(c) T는 연결되어 있고 $n-1$개의 간선을 갖는다
(d) T는 비순환적이고 $n-1$개의 간선을 갖는다

9.2 Terminology and Characterizations of Trees

- $(d) \Rightarrow (a)$ T가 비순환적이고 $n-1$개의 간선을 갖는다고 가정. T가 트리임을 보여야 한다. (T가 단순 그래프이고, T는 임의의 정점에서 다른 정점으로의 유일한 단순 경로를 갖는다.)
- 루프는 사이클이고 T는 비순환적이라고 했으므로, 그래프 T는 어떠한 루프도 포함할 수 없다. 마찬가지로 T는 v와 w에 결합된 서로 다른 간선 e_1과 e_2를 포함할 수 없다. 그렇게 된다면 사이클 (v, e_1, w, e_2, v)를 가지게 되기 때문이다. 그러므로 T는 단순 그래프이다.



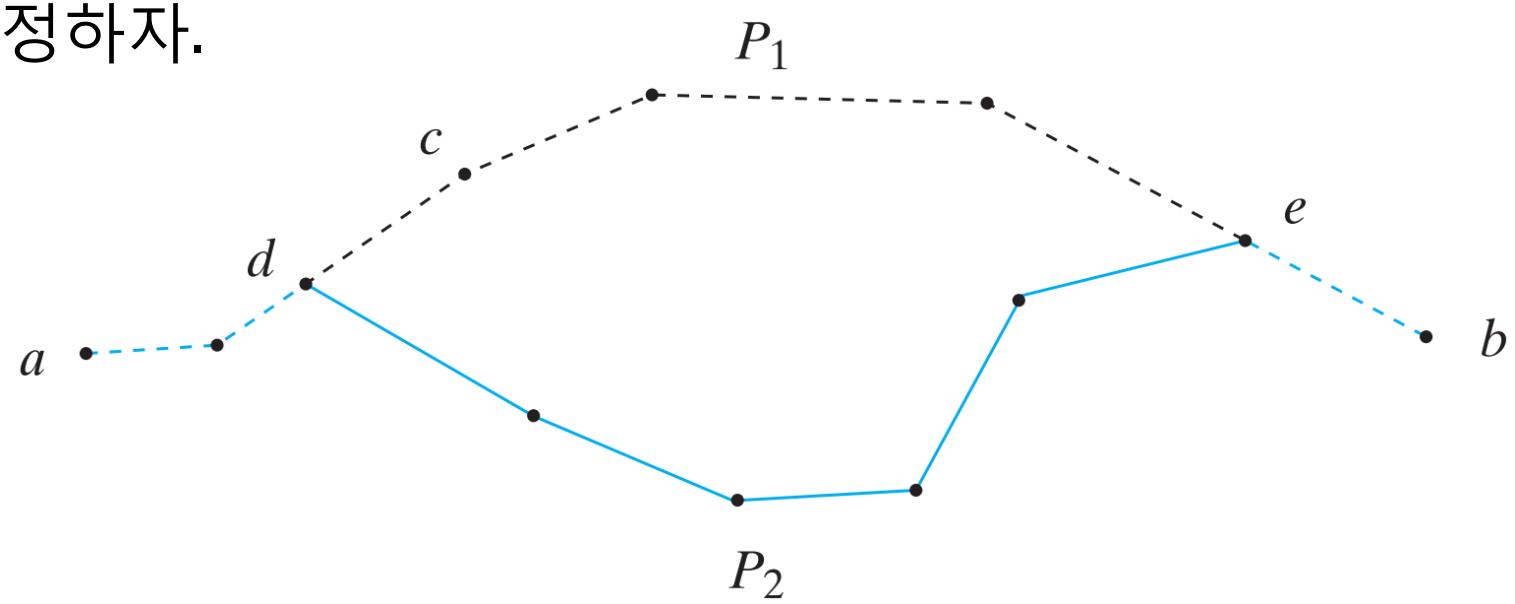
- T가 연결되어 있지 않다고 가정하자.
  $T_1, T_2, \ldots, T_k$를 T의 요소들이라고 하자. T가 연결되어 있지 않으므로, $k > 1$이다. T_i가 n_i개의 정점을 갖는다고 하자.
  각각의 T_i는 연결되어 있고 비순환적이므로, $(b) \Rightarrow (c)$에 의해 T_i는 $n_i - 1$개의 간선을 갖는다.

$$n - 1 = (n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) \quad \text{(간수의 수)}$$
$$= n_1 + n_2 + \cdots + n_k - k = n - k \quad \text{(정점의 수)}$$

위 식은 $k > 1$에 모순!! 그러므로 T는 연결되어 있다.

9.2 Terminology and Characterizations of Trees

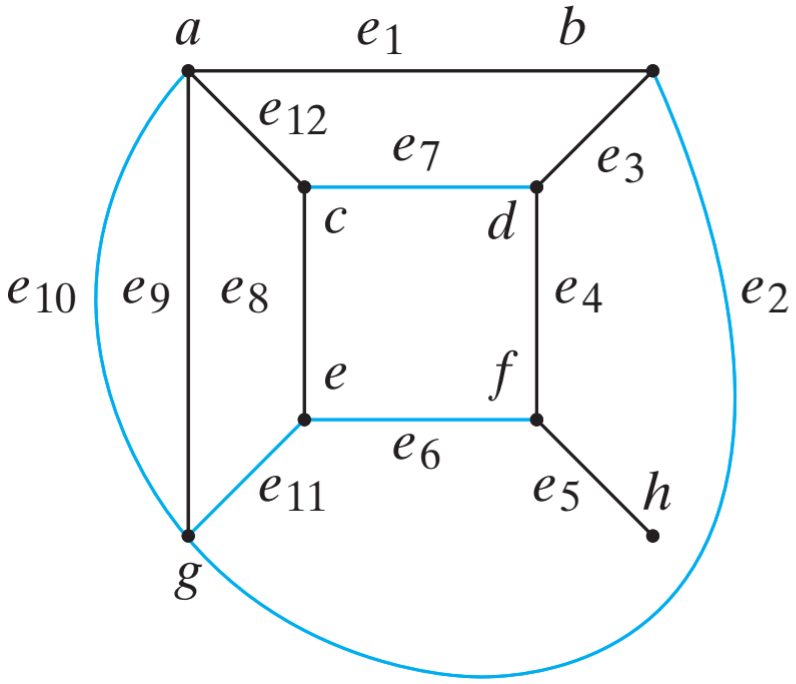
- T에서 a에서 b로의 서로 다른 단순 경로 P_1 과 P_2가 존재한다고 가정하자.

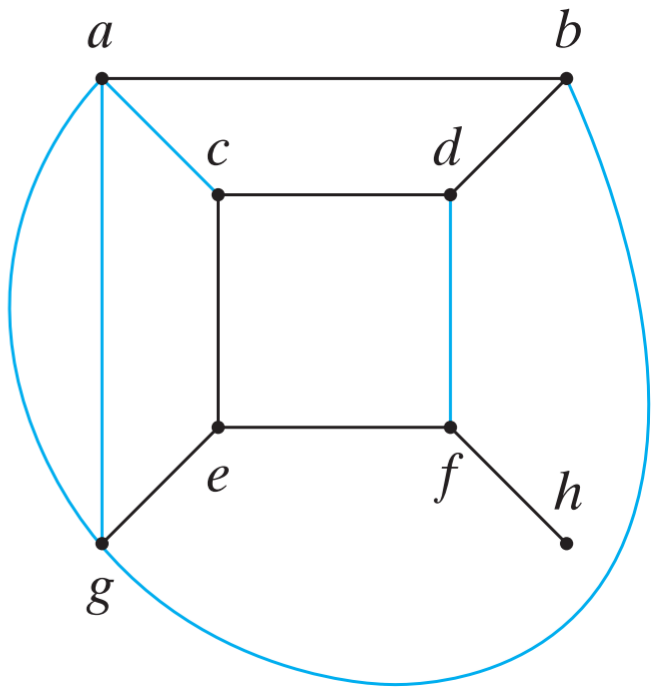


- c를 a 다음에 있으면서 P_1상에는 있고 P_2상에는 없는 첫 번째 정점이라고 하자. d를 P_1 상에서 c 바로 앞의 정점이라고 하자. 그리고 e를 d 다음에 있으면서 P_1과 P_2 상에 모두 있는 첫 번째 정점이라고 하자.
  그림과 같이 사이클이 존재하여, 모순이 발생한다. 따라서 T는 임의의 정점에서 다른 정점으로의 유일한 단순 경로를 갖는다. 그러므로 T는 트리이다.

9.3 Spanning Trees

- 정의 9.3.1 T가 그래프 G의 모든 정점을 포함하는 부분 그래프로 트리이면, T는 G의 **신장 트리**spanning tree이다.
- 예제 일반적으로 그래프는 여러 개의 신장 트리를 갖는다. 그림의 그래프 G의 신장 트리는 검은 선으로 표시한 것 이다.



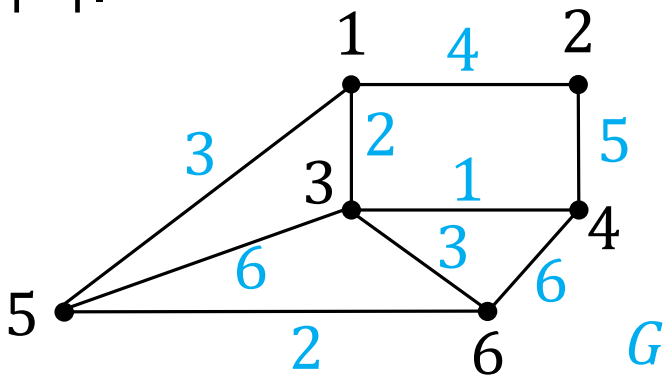


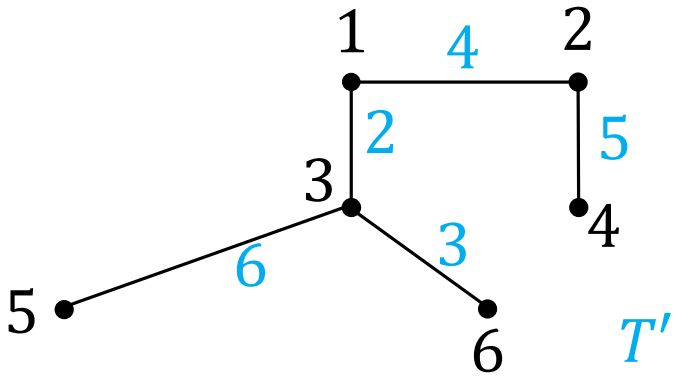
9.3 Spanning Trees

- 정리 9.3.4
  그래프 G가 신장 트리 T를 갖는다 $\Leftrightarrow$ G가 연결되어 있다
- 증명 $\Rightarrow$ 그래프 G가 신장 트리 T를 갖는다고 가정하고,
  a와 b를 G의 정점이라고 하자.
  그러면 a 와 b는 T의 정점이고, T는 트리이므로 a에서 b로의 경로 P가 존재한다. 그러나 P는 G 에서도 a에서 b로의 경로의 역할을 한다. 따라서 G는 연결되어 있다.
- $\Leftarrow$ G가 연결되어 있다고 가정하자.
  만약 G가 비순환적이면 정리 9.2.3에 의해 G는 트리이다.
  G가 사이클을 포함한다면, 이 사이클에서 간선 하나(정점은 그대로 둔다)를 제거한다. 결과 그래프는 연결은 되어 있다. 이 과정을 반복한다.
  결국, 비순환적이고 연결된 부분 그래프 T를 생성할 수 있다.
  정리 9.2.3 에 의해 T는 트리이다. 또한 T가 G의 모든 정점을 포함하므로 T는 G의 신장 트리이다.

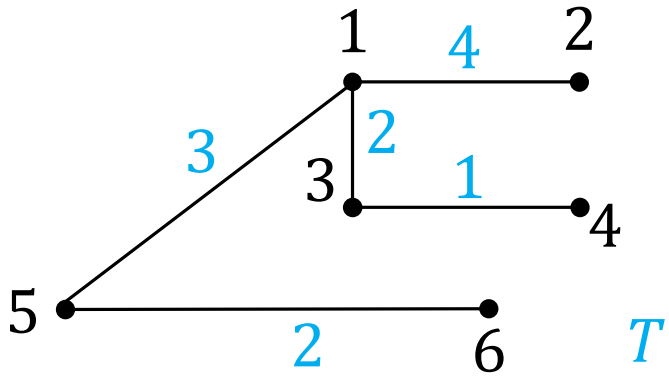
9.4 Minimal Spanning Trees

- 정의 9.4.1 G를 가중치 그래프라고 하자.
  G의 **최소 신장 트리**minimal spanning tree는 최소 가중치를 갖는 G의 신장 트리이다.
- 예제 9.4.2





가중치 20



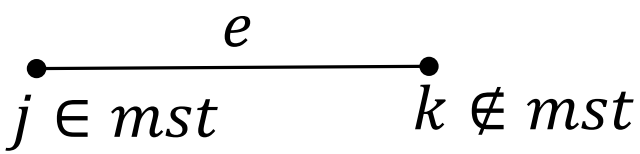
가중치 12

9.4 Minimal Spanning Trees(프림의 알고리즘)

- 프림의 알고리즘은 최소 신장 트리가 구해질 때까지 간선들을 반복적으로 추가함으로써 트리를 구축한다. 이 알고리즘은 하나의 정점을 가지고 시작한다. 각 반복에서 사이클을 형성하지 않는 최소 가중치를 가진 간선들을 현재의 트리에 추가한다.
- Algorithm 9.4.3 프림의 알고리즘Prim's Algorithm
  이 알고리즘은 연결된 가중치가 있는 그래프에서 최소 신장 트리minimal spanning tree를 찾는다.
- Input: A connected, weighted graph with vertices $1, \ldots, n$ and start vertex s.
  If (i, j) is an edge, $w(i, j)$ is the weight of (i, j);
  if (i, j) is not an edge, $w(i, j)$ is equal to ∞.
- Output: The set of edges E in a minimal spanning tree.

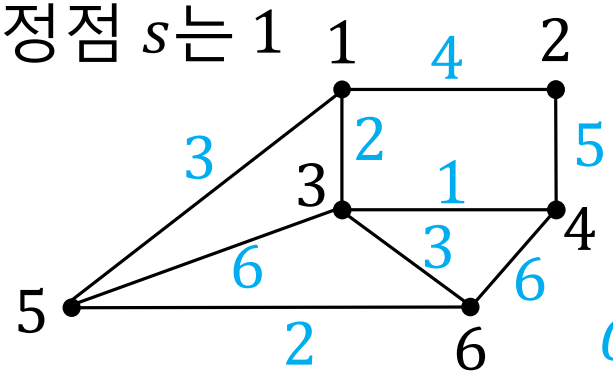
9.4 Minimal Spanning Trees(프림의 알고리즘)

```
    prim(w, n, s) {
        // v(i) = 1 if vertex i has been added to mst
 1.     for i = 1 to n
 2.         v(i) = 0
 3.     v(s) = 1  // add start vertex to mst
 4.     E = ∅     // begin with an empty edge set
        // put n − 1 edges in the minimal spanning tree
 5.     for i = 1 to n − 1 {
            // add edge of minimum weight with one vertex in mst
            // and one vertex not in mst
 6.         min = ∞
 7.         for j = 1 to n
 8.             if (v(j) == 1)  // if j is a vertex in mst
 9.                 for k = 1 to n
10.                     if (v(k) == 0 ∧ w(j, k) < min ) {
11.                         add_vertex = k
12.                         e = (j, k)
13.                         min = w(j, k)
14.                     }
            // put vertex and edge in mst
15.         v(add_vertex) = 1
16.         E = E ∪ {e}
17.     }
18.  return E
19.}
```

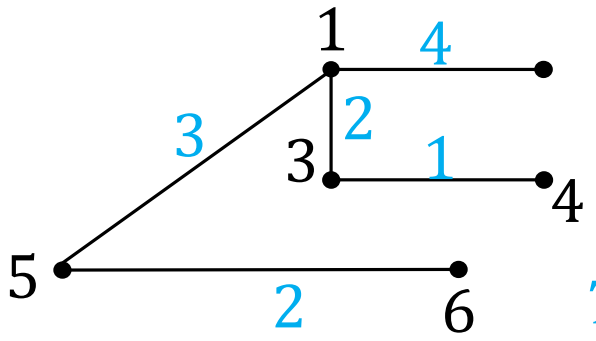


9.4 Minimal Spanning Trees

□ 예제 9.4.4 시작 정점 s는 1



G



T

①

Edge	Weight
(1,2)	4
(1, 3)	**2**
(1,5)	3

②

Edge	Weight
(1,2)	4
(1,5)	3
(3, 4)	**1**
(3,5)	6
(3,6)	3

③

Edge	Weight
(1,2)	4
(1, 5)	**3**
(2,4)	5
(3,5)	6
(3,6)	3
(4,6)	6

④

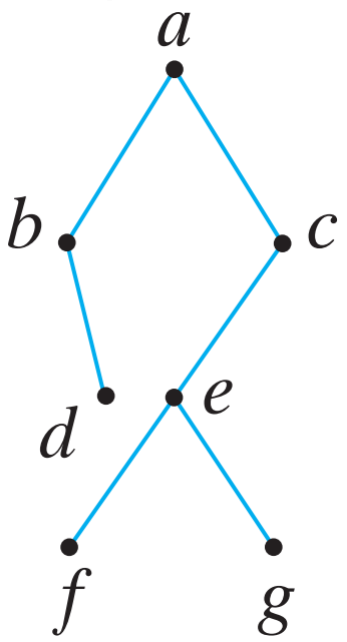
Edge	Weight
(1,2)	4
(2,4)	5
(3,6)	3
(4,6)	6
(5, 6)	**2**

⑤

Edge	Weight
(1, 2)	**4**
(2,4)	5

9.5 Binary Trees

- 정의 9.5.1 **이진 트리**는 각 정점이 자식이 없거나, 1 개 또는 2개의 자식을 가지는 근 트리이다.
  만약 어떤 정점이 하나의 자식을 가지면, 그 자식은 왼쪽 자식 또는 오른쪽 자식으로 지정된다.
  만약 어떤 정점이 2개의 자식을 가지면, 자식 하나는 왼쪽 자식으로, 그리고 다른 자식은 오른쪽 자식으로 지정된다.
- 예제 9.5.2 그림의 이진 트리에서,
  정점 b는 정점 a의 왼쪽 자식이다.
  정점 c는 정점 a의 오른쪽 자식이다.
  정점 d는 정점 b의 오른쪽 자식이다.
  정점 b는 왼쪽 자식이 없다.
  정점 e는 정점 c의 왼쪽자식이다
  정점 c는오른쪽 자식이 없다.



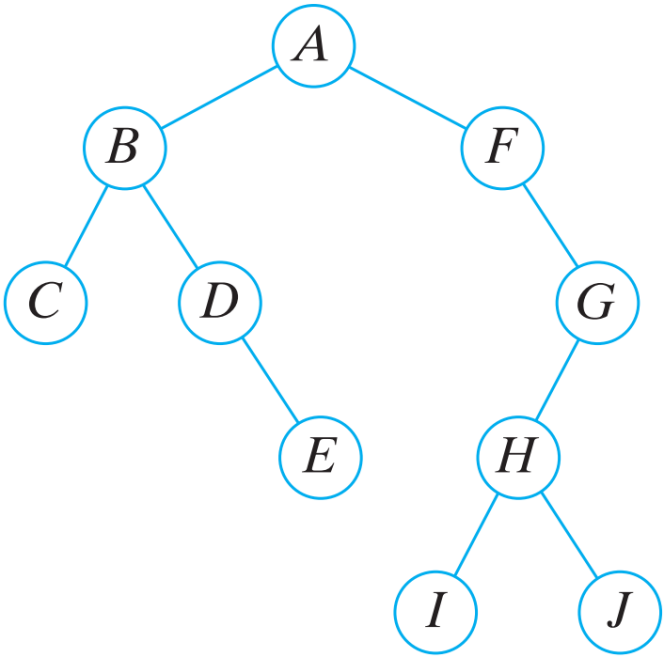
9.6 Tree Traversals

- 트리의 각 정점을 정확히 한 번씩만 방문하는 체계적인 방법으로 트리를 운행
- 이진 트리의 운행법
  - 전위 운행법preorder traversal
  - 중위 운행법inorder traversal
  - 후위 운행법postorder traversal
- “전위”, “중위”, “후위”는 운행 중 뿌리의 위치를 말한다
  - “전위”는 뿌리를 먼저 처리
  - “중위”는 뿌리를 중간에 처리 (두 자식 사이)
  - “후위”는 뿌리를 마지막에 처리

9.6 Tree Traversals

- Algorithm 9.6.1 전위 운행법Preorder Traversal
  이 재귀 알고리즘은 전위 운행법을 사용하여 이진 트리의 정점들을 처리한다.
- Input: PT, 이진 트리의 뿌리, 또는 트리가 비어 있음을 나타내는 특별한 값 null
- Output: "process" 에 따라 종속적이다.

$ABCDEFGHIJ$



$preorder(PT)$ {

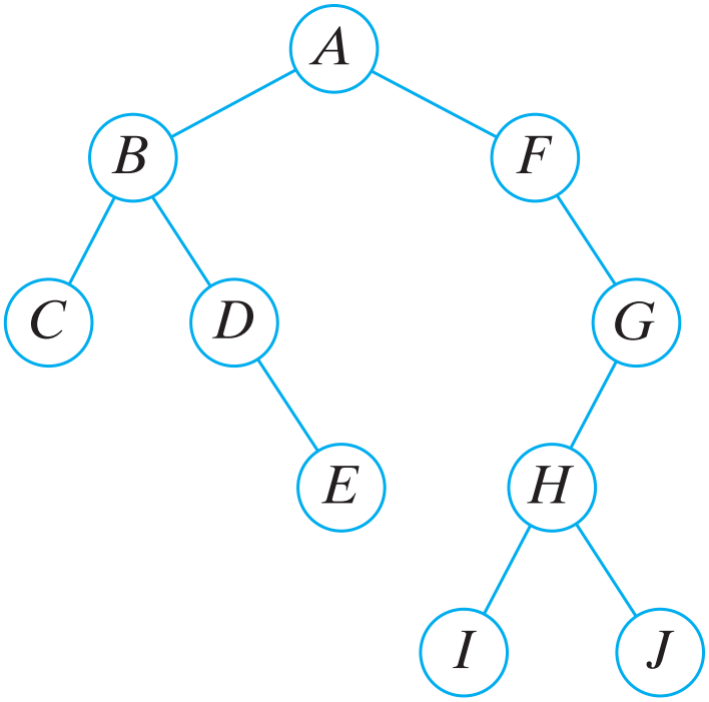
1. if $(PT == null)$
2. return
3. process PT
4. $l =$ left child of PT
5. $preorder(l)$
6. $r =$ right child of PT
7. $preorder(r)$

}

9.6 Tree Traversals

- Algorithm 9.6.3 중위 운행법Inorder Traversal
  이 재귀 알고리즘은 중위 운행법을 사용하여 이진 트리의 정점들을 처리한다.
- Input: PT, 이진 트리의 뿌리, 또는 트리가 비어 있음을 나타내는 특별한 값 null
- Output: "process" 에 따라 종속적이다.

$CBDEAFIHJG$



$inorder(PT)$ {

1. if (PT == $null$)
2. return
3. l = left child of PT
4. $inorder(l)$
5. process PT
6. r = right child of PT
7. $inorder(r)$

}

9.6 Tree Traversals

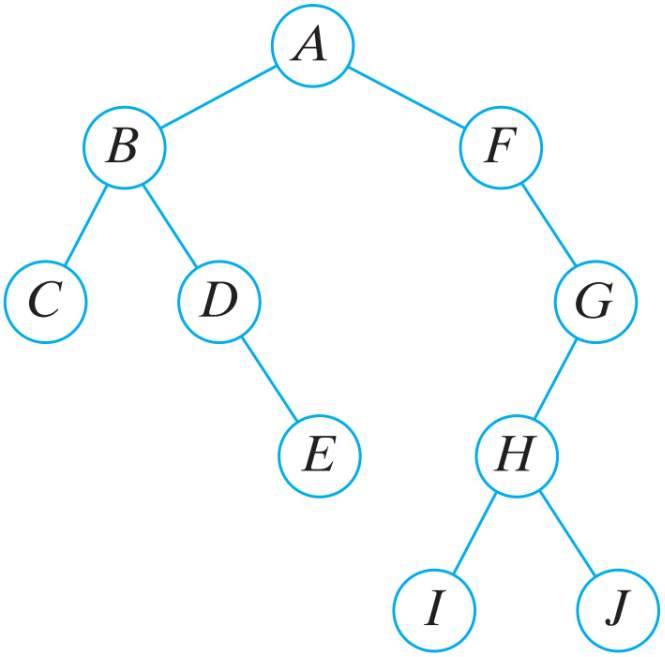
- Algorithm 9.6.5 후위 운행법Postorder Traversal
  이 재귀 알고리즘은 후위 운행법을 사용하여 이진 트리의 정점들을 처리한다.
- Input: PT, 이진 트리의 뿌리, 또는 트리가 비어 있음을 나타내는 특별한 값 null
- Output: "process" 에 따라 종속적이다.

$CEDBIJHGFA$

$postorder(PT)$ {

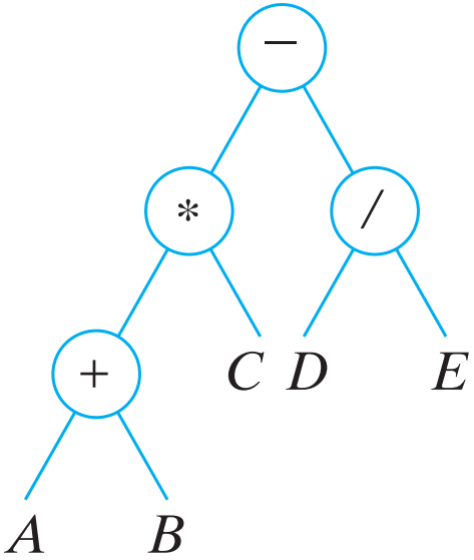
1. if (PT == $null$)
2. return
3. l = left child of PT
4. $postorder(l)$
5. r = right child of PT
6. $postorder(r)$
7. process PT

}



9.6 Tree Traversals

- 산술 수식을 이진 트리로 표현하는 방법
  - 말단 정점: 피연산자operands
  - 중간 정점: 연산자operators
- 예, $(A + B) * C - D/E$



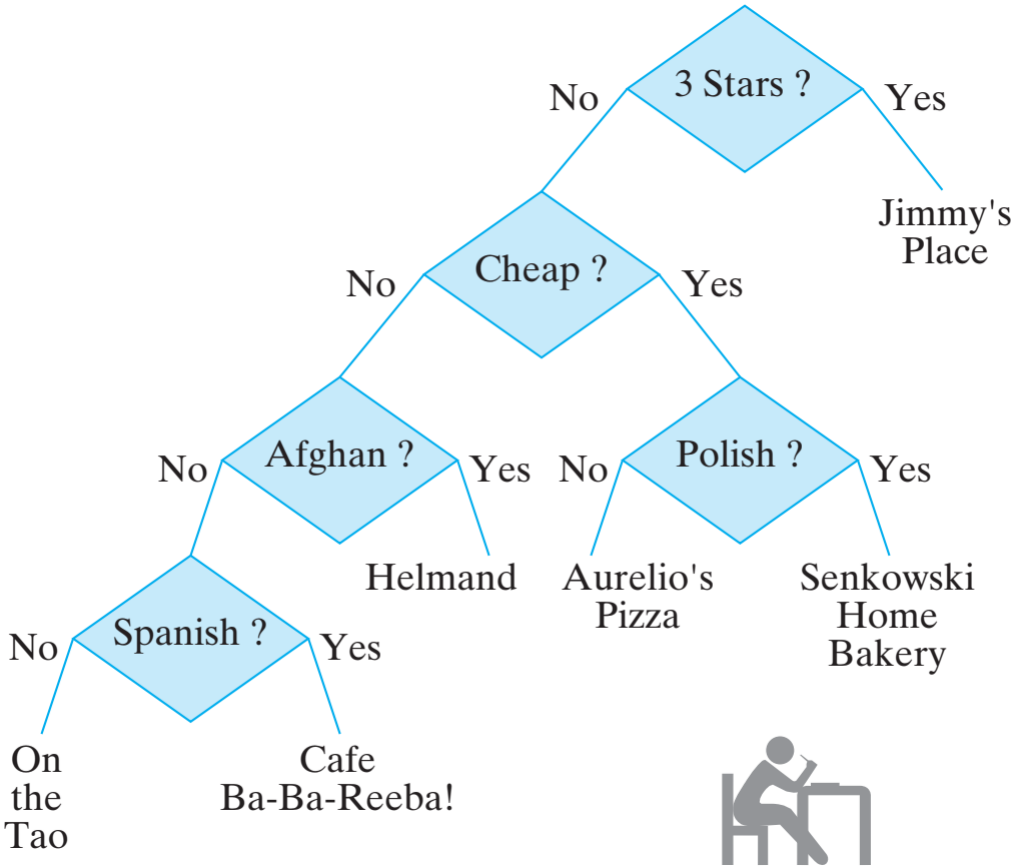
- 그림의 이진 트리를 후위 운행법으로 방문하면

$$AB + C * DE/-$$

- 수식의 후위 표기법
  - 연산자가 피연산자보다 뒤에 나온다
  - 괄호가 필요하지 않다

9.7 Decision Trees and the Min. Time for Sorting

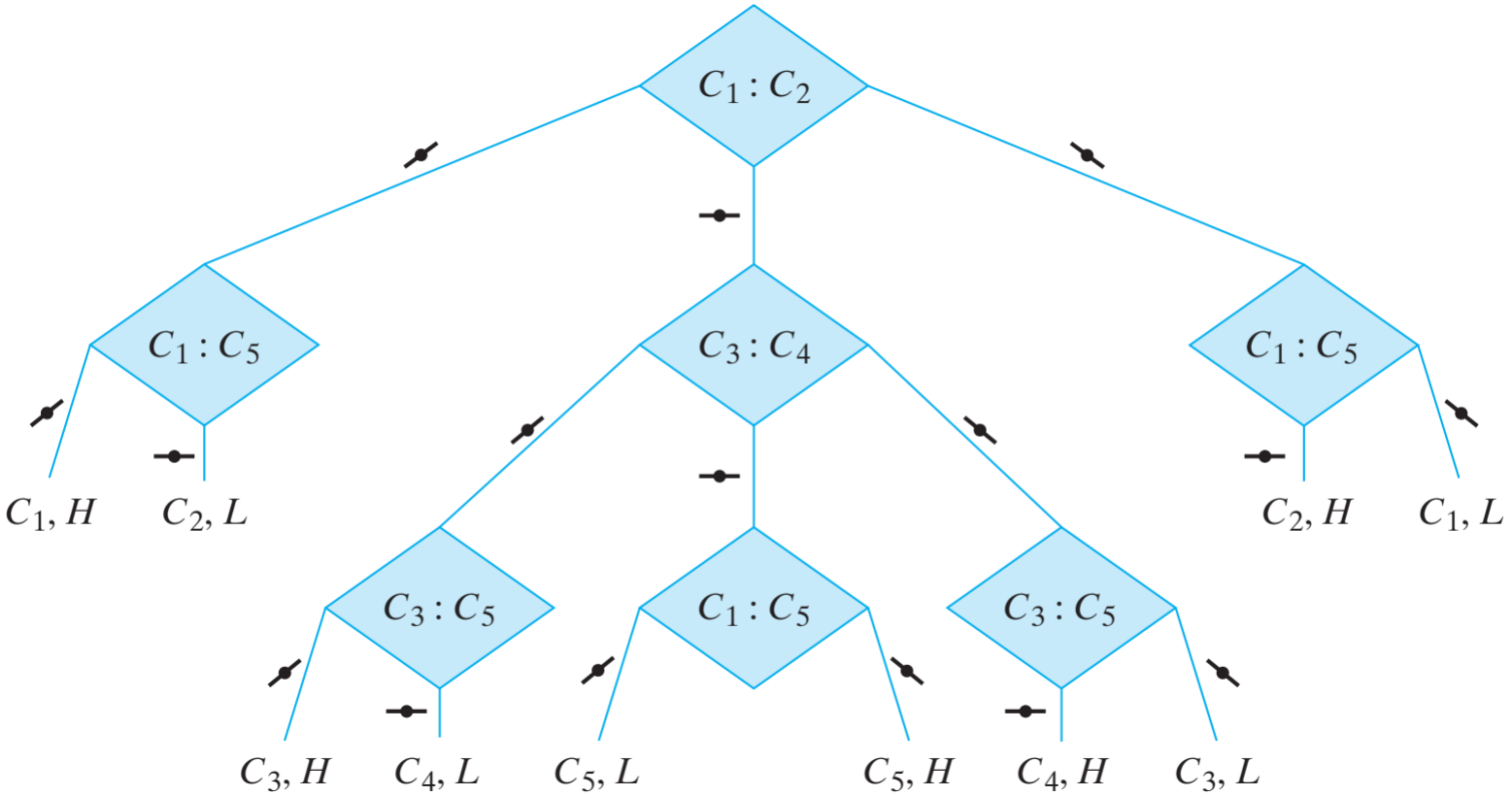
- 다음 이진 트리는 레스토랑을 선택하기 위한 알고리즘을 제시한다. 결정 트리decision tree라고 한다.



각 중간 정점은 질문. 뿌리에서 시작해서 각각의 질문에 답하고 적절한 간선을 따라가면, 레스토랑을 나타내는 말단 정점에 도착할 것이다.

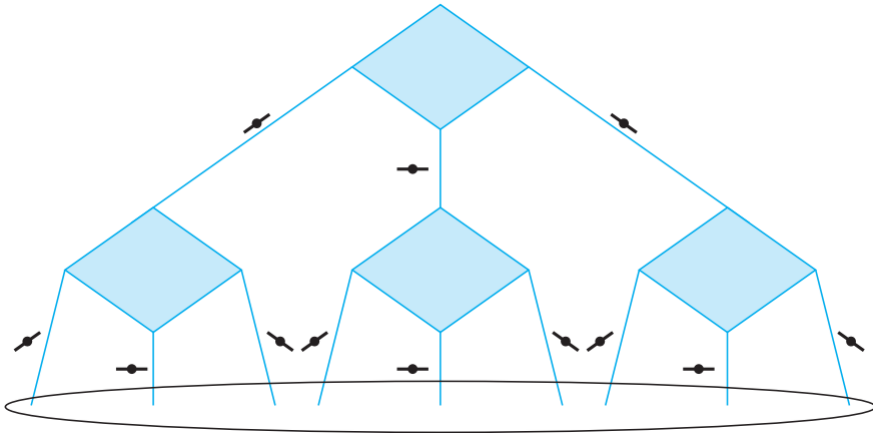
9.7 Decision Trees and the Min. Time for Sorting

- 예제 9.7.1 5개의 동전 중 하나만 가볍거나 무겁다. 천칭 사용.
- 최악의 경우 시간을 최악의 경우에 필요한 무게 비교의 수로 정의한다면, 최악의 경우 시간은 결정 트리의 높이인 3 이다.



9.7 Decision Trees and the Min. Time for Sorting

- 5개의 동전 문제를 최악의 경우에도 2 이하의 시간에 해결하는 알고리즘이 있다고 가정하자.
- 높이가 2인 결정 트리로 기술될 수 있다. 이러한 트리는 말단 정점이 많아야 9개이다.
- 5개의 동전 문제는 $10(=5\times 2)$개의 결과를 갖는다. 모순!
- 결정 트리가 어떤 문제를 해결하기 위해 필요한 최악의 경우 시간에 대한 하한을 제시하는 데 사용될 수 있다.



9.7 Decision Trees and the Min. Time for Sorting

- 아래 결정 트리에 의해 제시된 알고리즘의 최악의 경우 비교의 수는 3이다. 3개를 나열할 수 있는 경우의 수는 $3! = 6$이므로 높이가 2인 결정 트리(4개의 말단 정점)로는 기술 불가.
- n개를 정렬하는 데 필요한 최악의 경우 비교의 수는 $\Omega(n \lg n)$ 이다.