이산수학

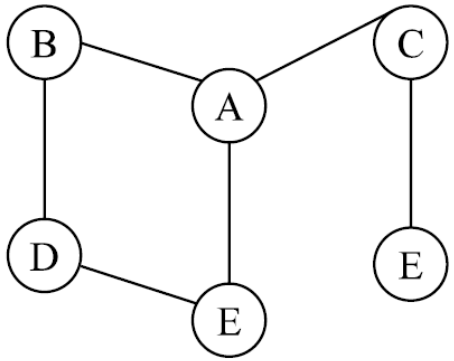
휴먼지능정보공학전공

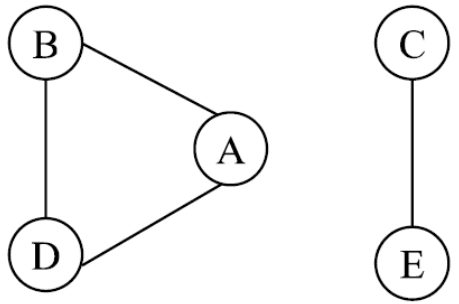
학습 내용

- 트리의 기본 개념
- 트리 구성
- 이진 트리
- 이진 트리 탐색
- 트리 활용

기본개념

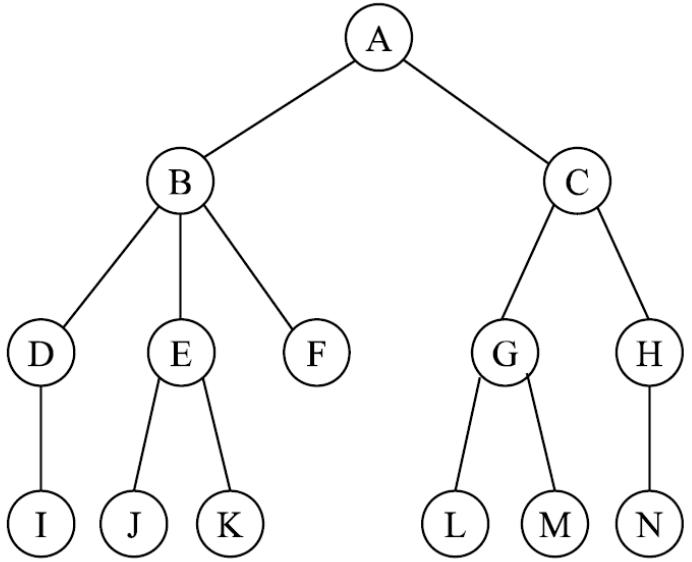
- 트리(Tree)
  - 트리 T는 비순환, 연결 그래프로 다음과 같은 특징이 있음
    - 특별한 노드인 루트(root)는 반드시 하나 있음
    - 트리 T를 구성하는 꼭짓점 v, w간에 v 에서 w 로 가는 단순 경로가 있음





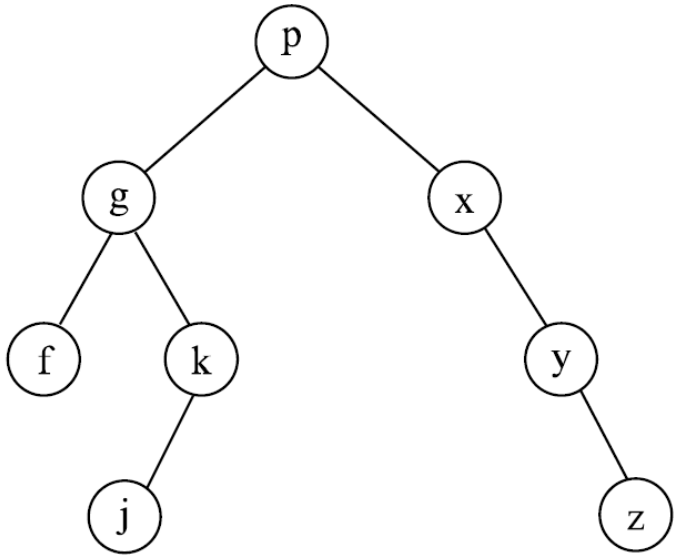
기본개념

- 서브 트리(Sub Tree)
  - T를 구성하는 꼭짓점 v를 루트로 하는 트리



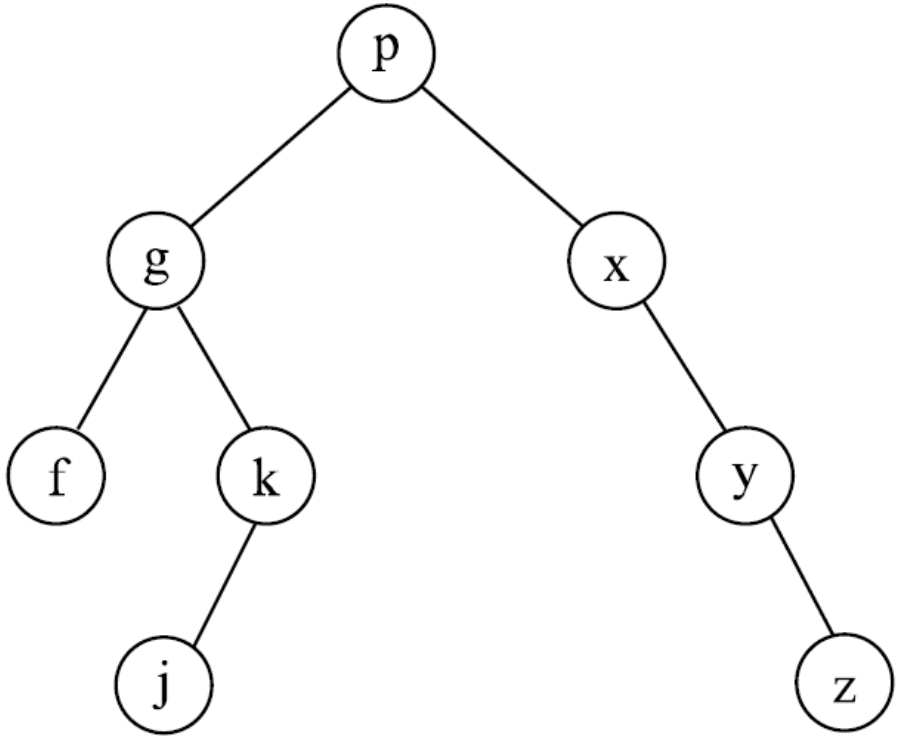
기본개념

- 노드(Node)
  - 그래프 T를 구성하는 꼭짓점
- 루트(Root)
  - 그래프 T의 시작 노드, 그래프 T의 가장 높은 곳에 위치
- 부모 노드(Parent Node)
  - 어떤 노드의 한 단계 상위 노드
- 자식 노드(Child Node)
  - 어떤 노드의 한 단계 하위 노드



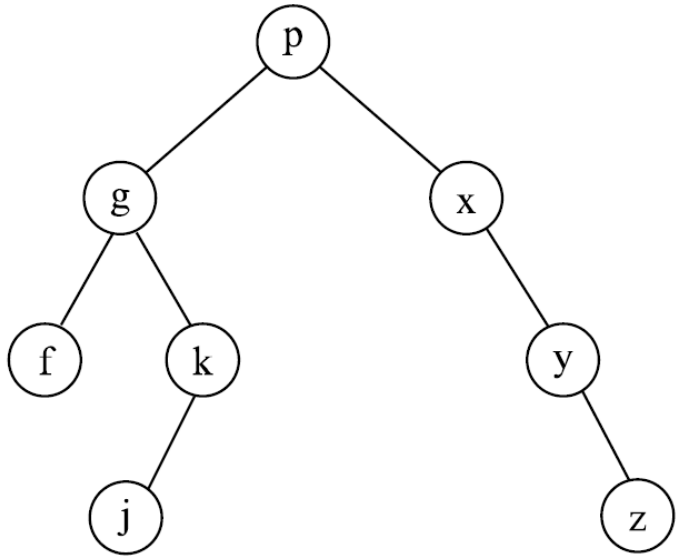
기본개념

- 조상 노드(Ancestor Node)
  - 루트 노드에서 어떤 노드에 이르는 경로에 포함된 모든 노드
- 자손 노드(Descendant Node)
  - 어떤 노드에서 잎 노드에 이르는 경로
- 차수(Degree)
  - 어떤 노드에 포함된 자식 노드의 개수



기본개념

- 레벨(Level)
  - 루트 노드를 0으로 시작하여, 자식 노드로 내려갈 때마다 하나씩 증가하는 단계
- 트리의 높이(Height) / 트리의 깊이(Depth)
  - 트리가 가지는 최대 레벨
- 숲(Forest)
  - 루트 노드를 제거하여 얻는 서브 트리의 집합



기본개념

- 트리에 대한 정리
  - n개의 꼭짓점을 갖는 연결 그래프 T에 대해 다음은 동치다.
  - (1) T 는 트리다.
  - (2) T 의 변의 수는 $n-1$ 개다.
  - (3) T 에서 변 하나를 제거하면 연결 그래프가 아니다.
  - (4) T 에 속하는 서로 다른 꼭짓점 w,v 에 대해 w에서 v로 가는 유일 경로가 존재한다.

이진트리

- *n*항 트리(*n*-ary tree)
  - 트리의 최대 차수가 *n* 인 트리

- 이진 트리(Binary Tree)
  - 트리 T를 구성하는 부모 노드가 갖는 자식 노드의 수가 최대 2개인 트리

이진트리

- 완전 이진 트리(Complete Binary Tree)
  - 높이가 h일때 레벨 1부터 h-1까지 모든 노드가 두 개씩 채워져 있고(h-2까지 자식 노드가 두개다), 레벨 h는 왼쪽부터 노드가 채워져 있는 트리
- 포화 이진 트리(Full Binary Tree)
  - 높이가 h일 때, 레벨 1에서 h까지 모든 노드가 두 개씩 채워져 있는 트리

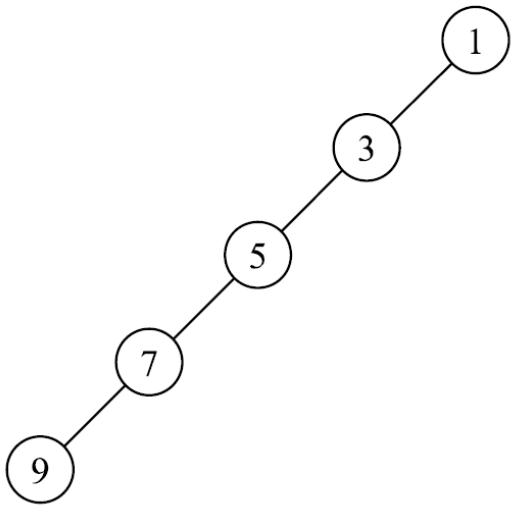
이진트리

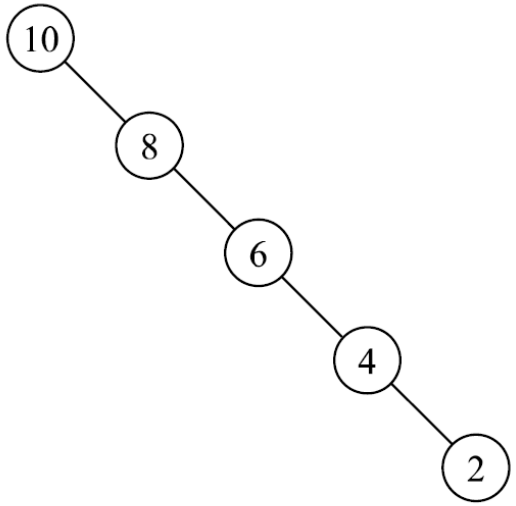
- 편향 이진 트리(Skewed Binary Tree)
  - 왼쪽이나 오른쪽 서브 트리만 가지는 트리

이진트리

문제

- 이전 값보다 큰 값은 왼쪽 노드에, 작은 값은 오른쪽 노드에 구성하려고 한다. 다음과 같은 입력을 트리로 구성하세요
- 1, 3, 5, 7, 9
- 10, 8, 6, 4, 2



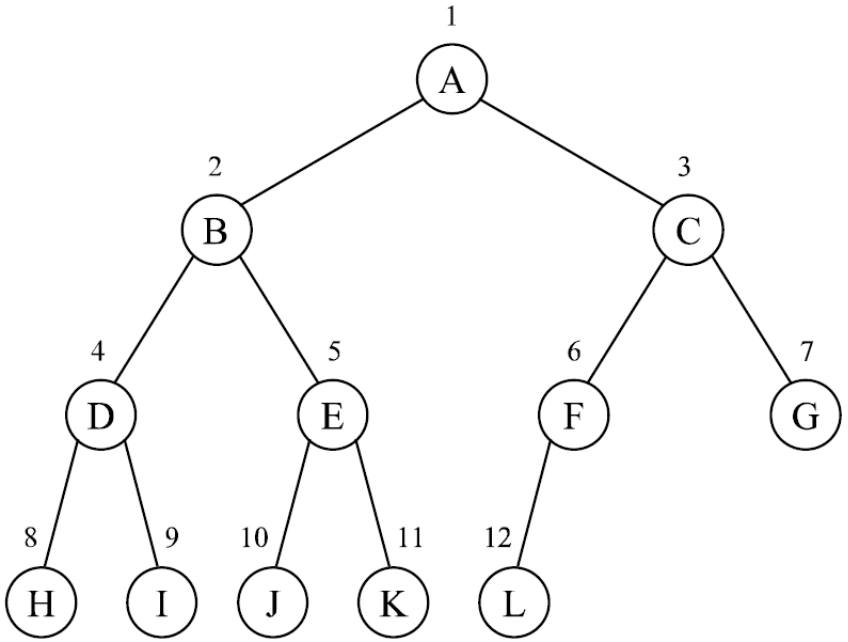


이진트리

- 이진 트리의 최대 노드 수
  - 레벨 k 에서 가질 수 있는 최대 노드 수 : 2^k 개
- 이진 트리의 최대 노드 수
  - 높이가 m 인 트리가 가질 수 있는 최대 노드 수 : $2^{(m+1)}-1$ 개
- 이진 트리의 최대 노드 수
  - 높이가 m 인 이진 트리가 가질 수 있는 최소 노드 수 : m+1개

이진트리

- 배열로 구현한 이진 트리
  - 높이가 h인 이진 트리는 각 노드 번호를 인덱스로 하여 1차원 배열로 구현할 수 있다.



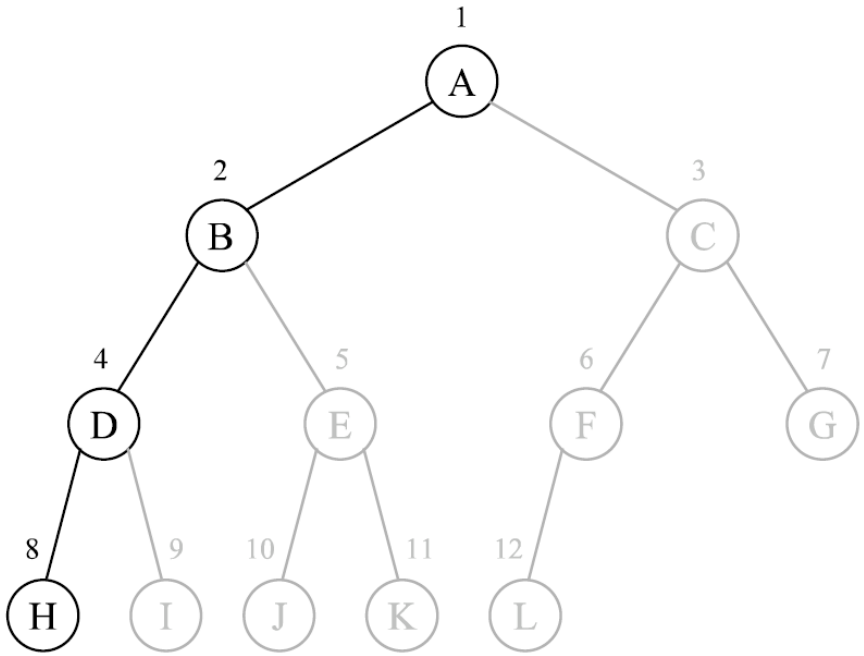
0	
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	K
12	L

이진트리

- 이진트리에 대한 배열 인덱스
  - 노드 인덱스 n의 부모 인덱스 : $\left\lfloor \frac{n}{2} \right\rfloor$
  - 노드 인덱스 n의 자식 인덱스
    - 왼쪽 자식 노드 인덱스 : $2n$
    - 오른쪽 자식 노드 인덱스 : $2n+1$

이진트리

- 이진편향트리에 대한 배열
  - 메모리 낭비 발생
  - 트리 중간에 노드 삽입, 삭제가 비효율적

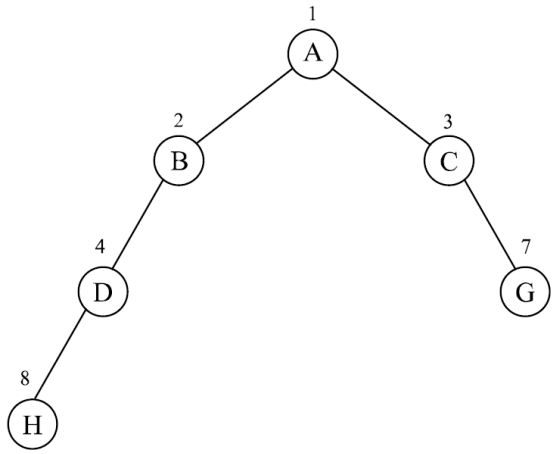


0	
1	A
2	B
3	
4	D
5	
6	
7	
8	H
9	
10	
11	
12	

이진트리

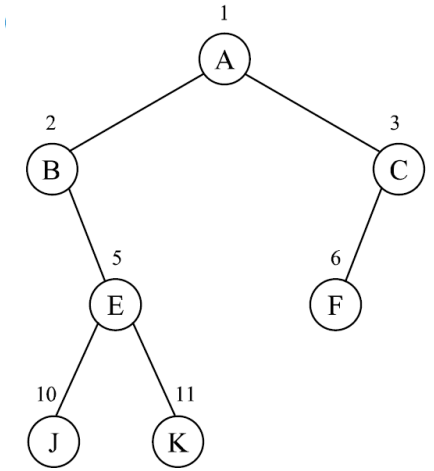
문제

- 이진 트리는 배열로 표현하고 배열은 이진 트리로 표현하세요



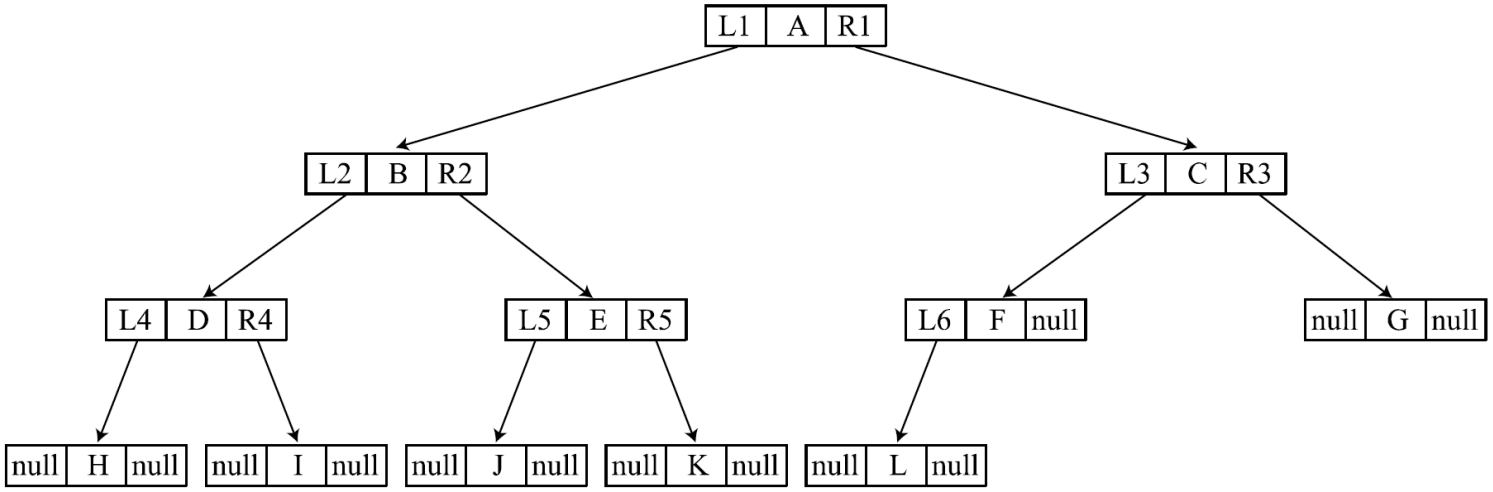
0	
1	A
2	B
3	C
4	D
5	
6	
7	G
8	H

0	
1	A
2	B
3	C
4	
5	E
6	F
7	
8	
9	
10	J
11	K



이진트리

- 연결리스트로 구현한 이진 트리
  - 메모리 낭비 해결
    - 부모 노드와 자식 노드간에 주소(포인터)로 연결
  - 삽입 삭제 용이
    - 주소 영역 변경만으로 노드 삽입 삭제 가능



이진트리

- 이진트리 순회의 규칙
  - 항상 루트에서 시작한다. 즉 트리의 레벨 0에서 시작한다.
  - 서브 트리에 대한 순회의 순서는 항상 왼쪽에서 오른쪽으로 이루어진다.
  - 데이터를 읽기 전에 왼쪽, 혹은 오른쪽 노드가 있는지 확인하는 작업을 한다.

이진트리

- 전위순회(Preorder Traversal)
  - 루트 노드 - 왼쪽 노드 - 오른쪽 노드 순으로 방문하는 순회 방식
    - 노드 1에 방문 : 데이터 A를 읽고(P) 왼쪽 노드가 있는지 확인
    - 노드 2에 방문 : 데이터 B를 읽고(L) 왼쪽 노드가 있는지 확인
    - 왼쪽 노드가 없으므로 오른쪽 노드가 있는지 확인
    - 오른쪽 노드도 없으므로, 다시 노드 1에 방문 : 오른쪽노드가 있는지 확인
    - 노드 3에 방문 : 데이터 C를 읽고(R) 왼쪽노드가 있는지 확인
    - 왼쪽 노드가 없으므로 오른쪽 노드가 있는지 확인
    - 오른쪽 노드도 없으므로 다시 노드 1에 방문 : 순회 종료

이진트리

- 중위순회(Inorder Traversal)
  - 왼쪽 노드 - 루트 노드 - 오른쪽 노드 순으로 방문하는 순회 방식
    - 노드 1에 방문 : 왼쪽 노드가 있는지 확인
    - 노드 2에 방문 : 왼쪽 노드가 있는지 확인
    - 왼쪽 노드가 없으므로, 노드 2의 데이터 B를 읽는다(L).
    - 노드 2의 오른쪽 노드가 있는지 확인 : 오른쪽 노드도 없다.
    - 다시 노드 1에 방문 : 데이터 A를 읽는다(P).
    - 노드 1의 오른쪽 노드가 있는지 확인
    - 노드 3에 방문 : 왼쪽 노드가 있는지 확인
    - 왼쪽 노드가 없으므로, 노드 3의 데이터 C를 읽는다(R)
    - 노드 3의 오른쪽 노드가 있는지 확인 : 오른쪽 노드도 없다.
    - 다시 노드 1에 방문 : 순회 종료

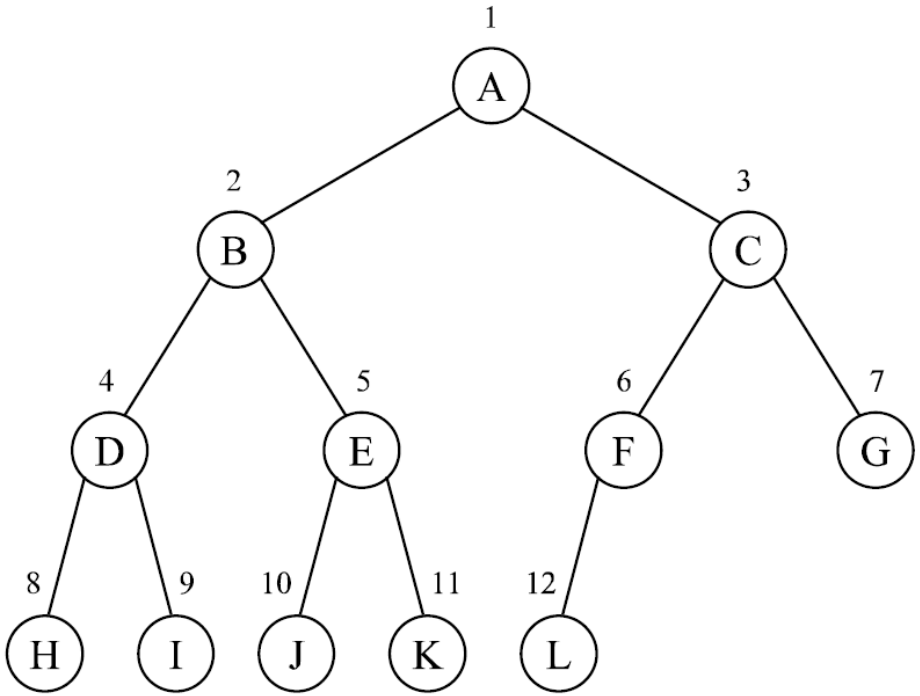
이진트리

- 후위순회(Inorder Traversal)
  - 왼쪽 노드 - 오른쪽 노드 - 루트 노드 순으로 방문하는 순회 방식
    - 노드 1에 방문 : 왼쪽 노드가 있는지 확인
    - 노드 2에 방문 : 왼쪽 노드가 있는지 확인
    - 왼쪽 노드가 없으므로, 노드 2의 오른쪽 노드가 있는지 확인
    - 오른쪽 노드도 없으므로, 노드 2의 데이터 B를 읽는다(L).
    - 다시 노드 1에 방문 : 노드 1의 오른쪽 노드가 있는지 확인
    - 노드 3에 방문 : 왼쪽 노드가 있는지 확인
    - 왼쪽 노드가 없으므로, 노드 3의 오른쪽 노드가 있는지 확인
    - 오른쪽 노드도 없으므로, 노드 3의 데이터 C를 읽는다(R).
    - 다시 노드 1에 방문 : 데이터 A를 읽고(P) 순회 종료

이진트리

문제

- 다음 트리의 전위/중위/후위 순회 값을 구하세요



A - B - D - H - I - E - J - K - C - F - L - G

H - D - I - B - J - E - K - A - L - F - C - G

H - I - D - J - K - E - B - L - F - G - C - A

이진트리

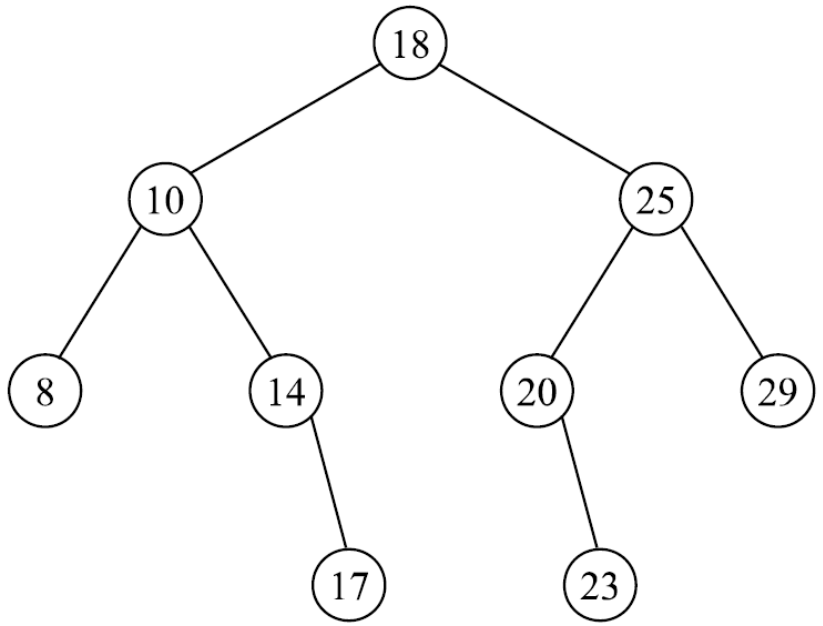
- 전위표기
  - 수식에서의 연산자가 피연산자보다 앞에 작성되는 표기법
  - 연산자 - 피연산자1 - 피연산자2
- 중위표기
  - 수식에서의 연산자가 피연산자들의 중간에 작성되는 표기법
  - 피연산자1 - 연산자 - 피연산자2
- 후위표기
  - 수식에서의 연산자가 피연산자들의 뒤에 작성되는 표기법
  - 피연산자1 - 피연산자2 - 연산자

이진트리탐색

- 탐색 트리(Binary Search Tree)
  - 노드가 가지는 데이터의 크기에 따라 노드의 위치를 탐색할 수 있는 트리
    - 트리에서 탐색되는 모든 원소는 서로 다른 유일키를 갖는다.
    - 왼쪽 서브 트리에 있는 원소의 키들은 그 루트의 키보다 작다.
    - 오른쪽 서브 트리에 있는 원소들의 키들은 그 루트의 키보다 크다.

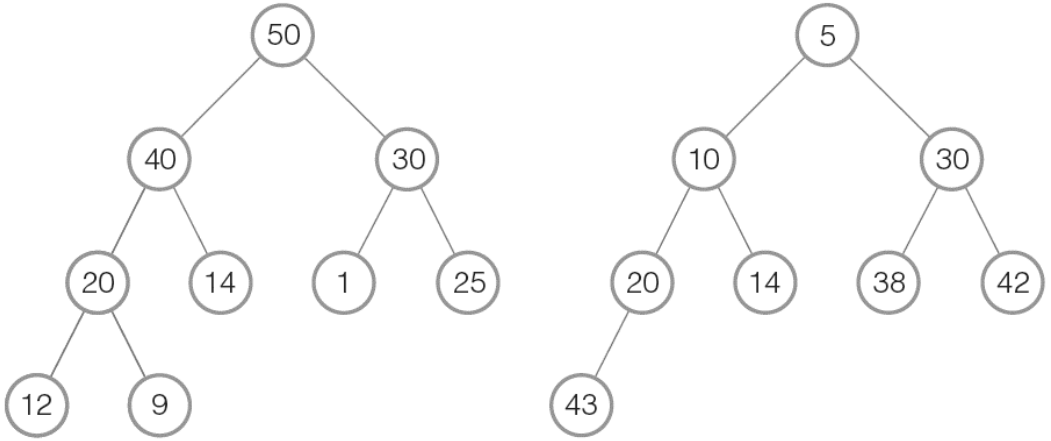
이진트리탐색

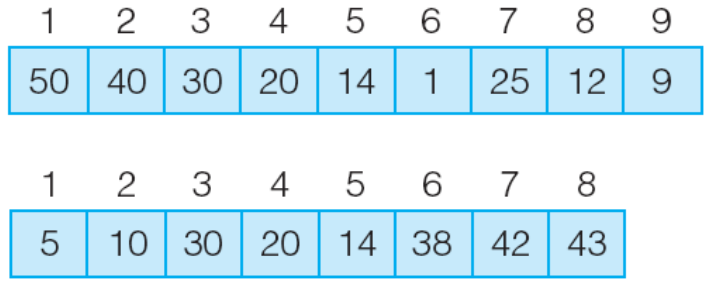
- 탐색 트리(Binary Search Tree)
  - 노드가 가지는 데이터의 크기에 따라 노드의 위치를 탐색할 수 있는 트리
  - 18, 10, 25, 20, 14, 8, 23, 17, 29



이진트리탐색

- 최대 힙과 최소 힙
  - 최대 힙(max heap)부모 노드의 키 값이 자식 노드의 키 값보다 크거나 같은 완전 이진 트리
    - key(부모 노드) >= key(자식 노드)
  - 최소 힙(min heap)부모 노드의 키 값이 자식 노드의 키 값보다 작거나 같은 완전 이진 트리
    - key(부모 노드) <= key(자식 노드)

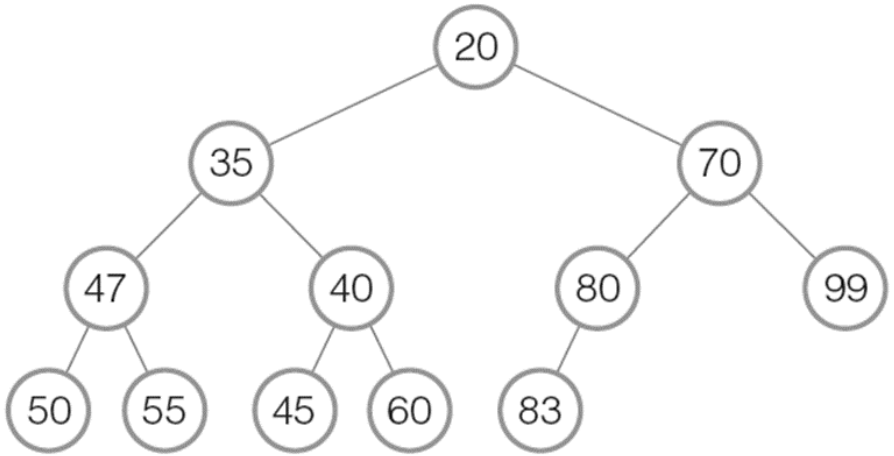




이진트리탐색

문제

- 다음과 같은 최소힙을 1차원 배열로 나타내고 인덱스 5에 해당하는 값의 부모, 왼쪽 자식, 오른쪽 자식의 위치와 그 값을 각각 구하세요

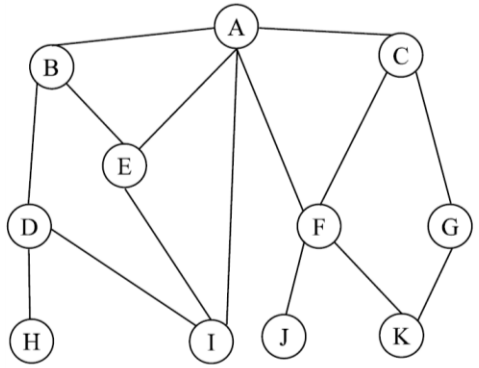


1	2	3	4	5	6	7	8	9	10	11	12
20	35	70	47	40	80	99	50	55	45	60	83

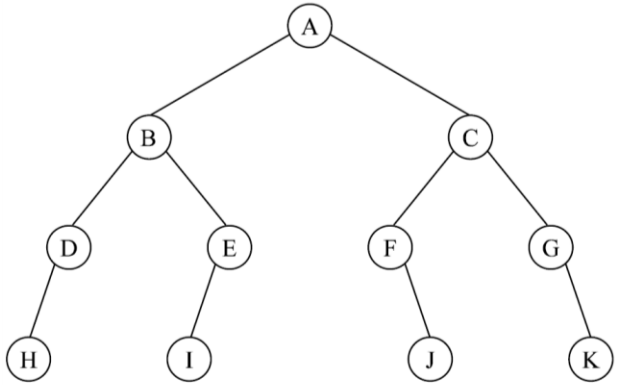
인덱스 5에 해당하는 값인 40의 부모의 위치는 $\left\lfloor \frac{5}{2} \right\rfloor = 2$므로 부모는 인덱스 2에 해당하는 값 35다. 또한 왼쪽 자식의 위치는 $2 \cdot 5 = 10$이므로 45가 왼쪽 자식이 되고, 오른쪽 자식의 위치는 $2 \cdot 5 + 1 = 11$이므로 60이 오른쪽 자식이 된다.

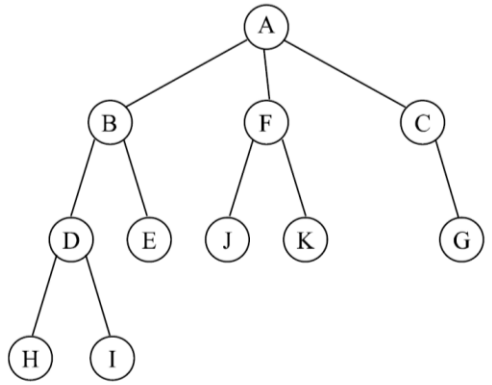
이진트리탐색

- 신장트리
  - 그래프 G의 꼭짓점을 모두 노드로 포함하는 트리 T



그래프 G



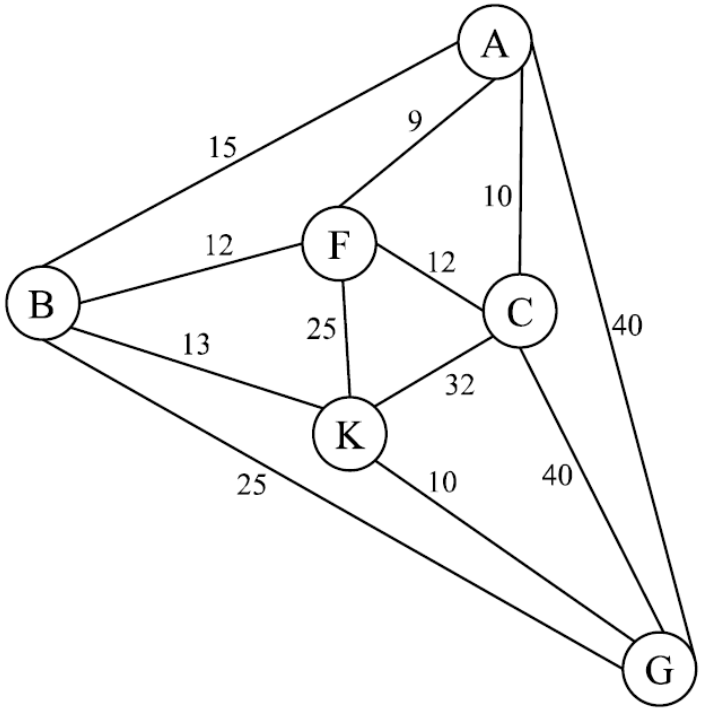


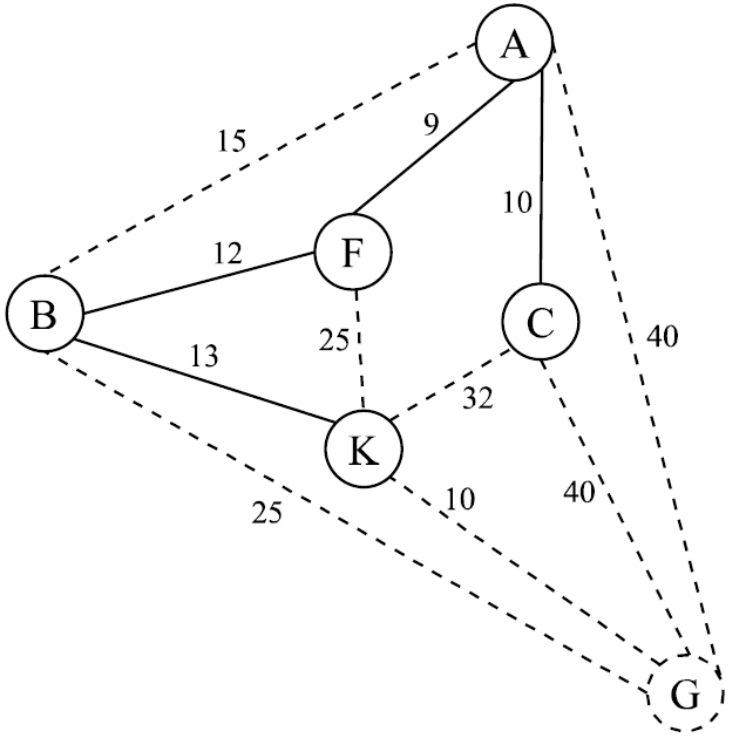
이진트리탐색

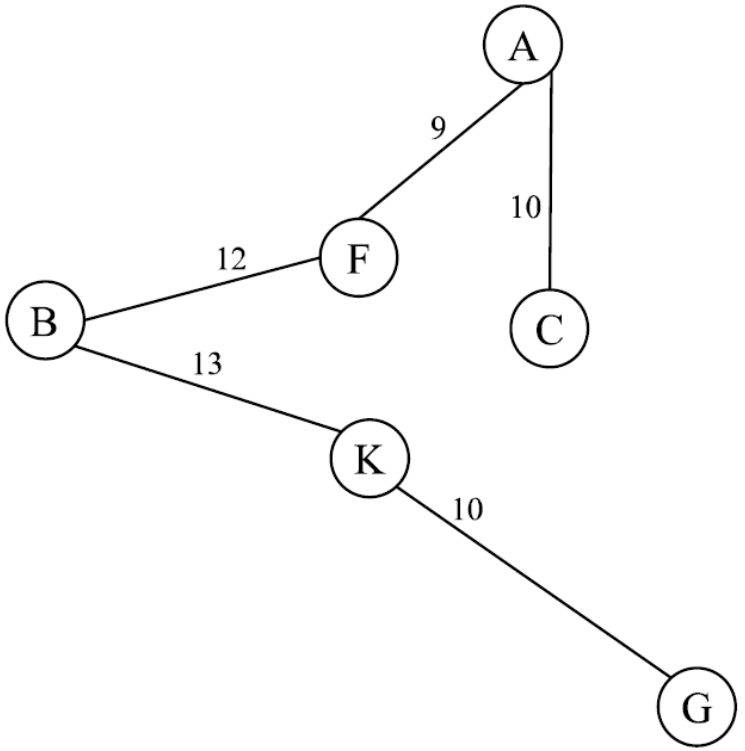
- 최소신장 트리(Minimal Spanning Tree)
  - 그래프 G의 꼭짓점을 모두 노드로 포함하면서 비용을 최소로 하는 트리 T
- 비용을 최소로 하는 알고리즘
  - 그래프 G의 변 중 비용이 가장 낮은 변들로 트리를 구성하는 알고리즘
    - 프림 알고리즘(Prim Algorithm)
      - 가중치가 가장 작은 변을 선택
      - 연결된 꼭짓점들과 연결된 모든 변들 중 가중치가 가장 작은 변을 선택
      - 가중치가 같은 변은 임의로 선택
      - 선택된 변에 의해 순환이 형성되는 경우는 선택하지 않음
      - n개의 꼭짓점에 대하여 n-1개의 변이 연결되면 종료
    - 크루스칼 알고리즘(Kruskal Algorithm)
      - 가중치가 가장 작은 변을 차례로 선택하여 노드들을 연결
      - 가중치가 같은 변은 임의로 선택
      - 선택된 변에 의해 회로가 형성되는 경우는 선택하지 않음
      - n개의 꼭짓점에 대하여 n-1개의 변이 연결되면 종료

이진트리탐색

- 프림 알고리즘

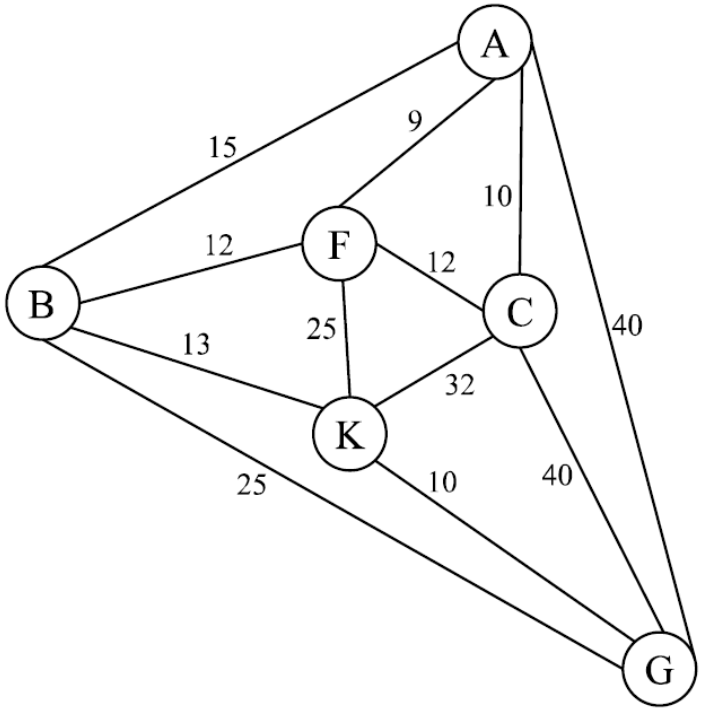




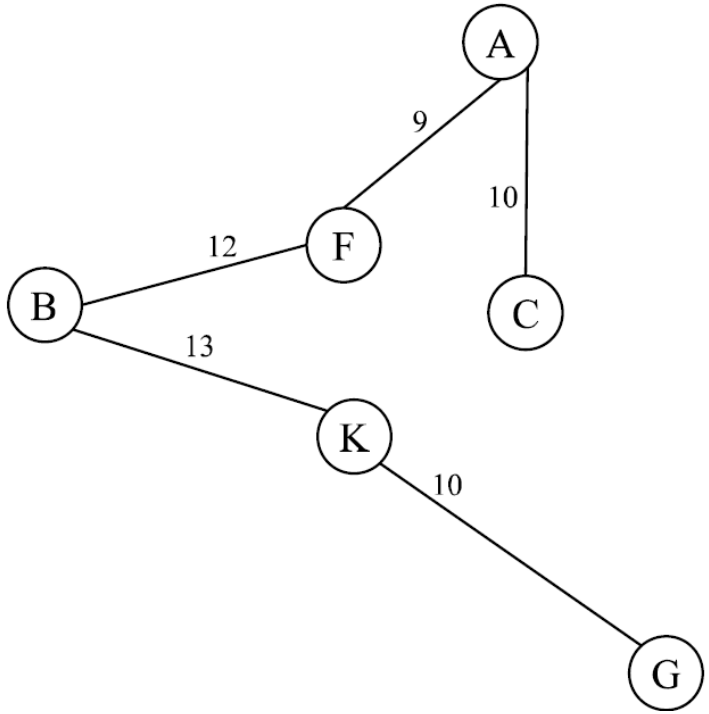


이진트리탐색

- 크루스칼 알고리즘



노드	연결 비용	노드	연결 비용
A-B	15	A-C	10
A-F	9	A-G	40
B-F	12	B-K	13
B-G	25	F-K	25
F-C	12	C-K	32
C-G	40	K-G	10

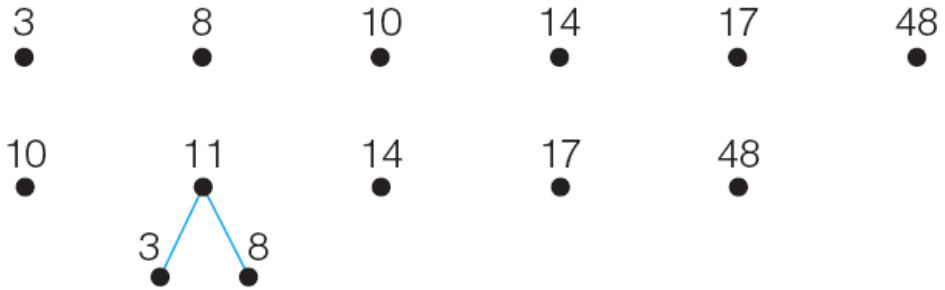


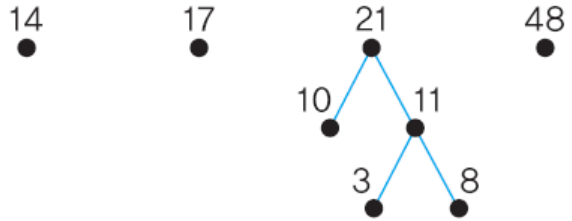
트리활용

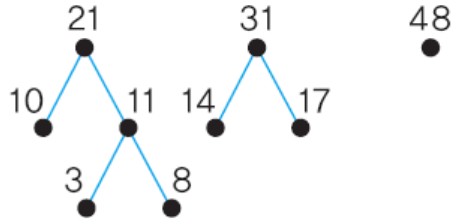
- 허프만 알고리즘(Huffman Algorithm)
  - 발생 빈도가 높은 문자에는 적은 비트를 할당하고 발생 빈도가 낮은 문자에는 많은 비트를 할당하기위한 알고리즘
    - 발생 빈도가 가장 낮은 두 문자를 선택하여 하나의 이진 트리로 연결
      - 왼쪽 노드에는 빈도수가 낮은 문자, 오른쪽 노드에는 빈도가 높은 문자가 오도록 함
      - 그 두 문자 노드의 루트는 두 문자의 빈도의 합으로 함
      - 문자들을 이진 트리로 연결하는 것을 최우선적으로 작업하고, 그 후에 이진 트리들을 연결하도록 함
    - 위의 과정을 모든 문자가 하나의 이진 트리로 묶일 때까지 반복
    - 생성된 이진 트리의 왼쪽 노드에는 0, 오른쪽 노드에는 1을 부여하여 루트부터 해당 문자까지의 0 또는 1을 순서대로 나열한 것이 해당 문자의 허프만 코드가 됨

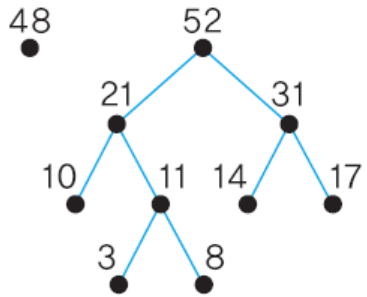
트리활용

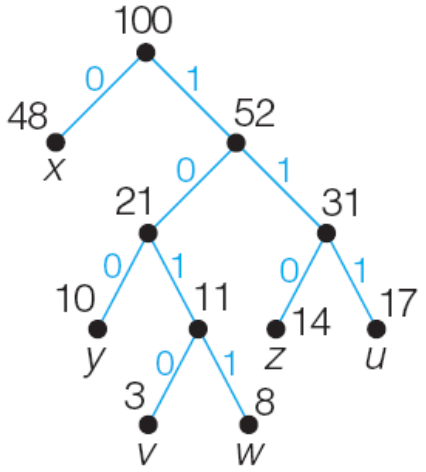
- 허프만 알고리즘(Huffman Algorithm)
  - u, v, w, x, y, z
  - 17, 3, 8, 48, 10, 14











트리활용

문제

- (1) 허프만 알고리즘을 따라 허프만 트리를 만들어라.
- (2) 각 문자에 대한 허프만 코드를 작성하라.
- (3) (2)의 허프만 코드를 이용해 문자열 uvwxyz를 코드로 작성하라.
- (4) (2)의 허프만 코드를 이용해 1111010101110100110 코드에 대응되는 문자열을 작성하라.

이산수학 – 문제해결 - 파이썬코딩

```
#이분탐색
def binsearch(a,x):
    start = 0
    end = len(a) -1
    while start<=end:
        mid = (start + end)//2
        if x ==a[mid]:
            return mid
        elif x > a[mid]:
            start = mid +1
        else:
            end = mid -1
    return -1
d = [2,4,5,16,21,34,48,56,72,82,90]
print(binsearch(d, 72))
```

```
# 출력결과

8
```

이산수학 – 문제해결 - 파이썬코딩

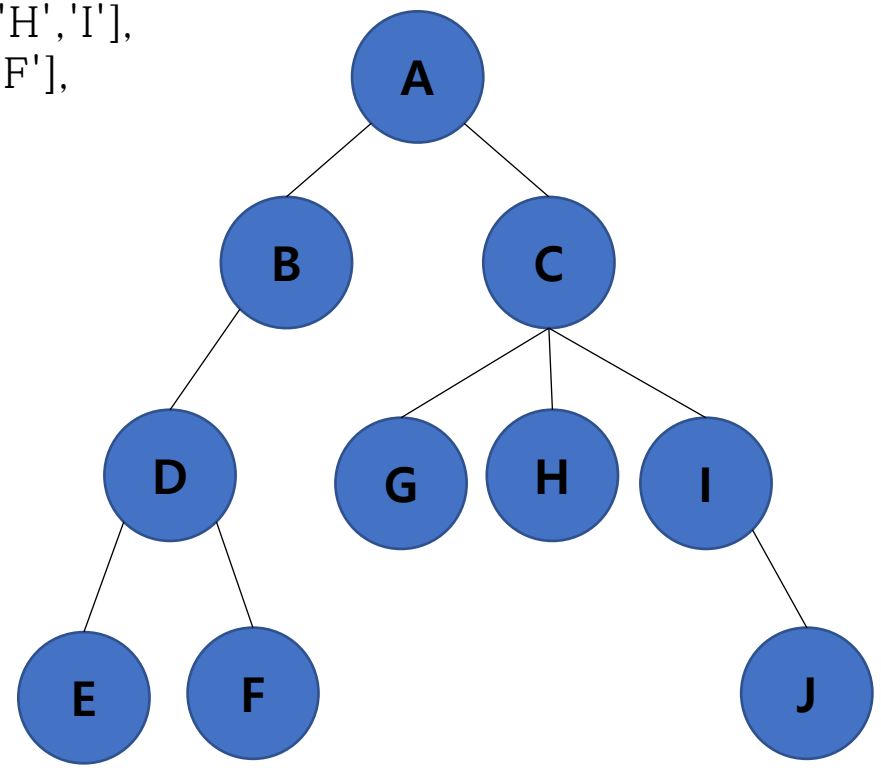
```
#순차탐색
def searchlist(a,x):
    n = len(a)
    for i in range(0,n):
        if x==a[i]:
            return i;
    return -1
v = [16,4,99,21,15,34,7,33]
print(searchlist(v, 33))
```

```
# 출력결과

7
```

이산수학 – 문제해결 - 파이썬코딩

```
#깊이우선 / 너비우선탐색
graph = {
    'A':['B','C'],
    'B':['A','D'],
    'C':['A','G','H','I'],
    'D':['B','E','F'],
    'E':['D'],
    'F':['D'],
    'G':['C'],
    'H':['C'],
    'I':['C','J'],
    'J':['I']}
```




```
def my_dfs(graph, start):
    visited = list()
    need_visit = list()
    need_visit.append(start)
    while need_visit:
        node = need_visit.pop()
        if node not in visited:
            visited.append(node)
            need_visit.extend(graph[node])
    return visited

def my_bfs(graph, start):
    visited = list()
    need_visit = list()
    need_visit.append(start)
    while need_visit:
        node = need_visit.pop(0)
        if node not in visited:
            visited.append(node)
            need_visit.extend(graph[node])
    return visited
print("DFS",my_dfs(graph,'A'))
print("BFS",my_bfs(graph,'A'))
```

학습 내용 요약

- 트리의 기본 개념
- 트리 구성
- 이진 트리
- 이진 트리 탐색
- 트리 활용