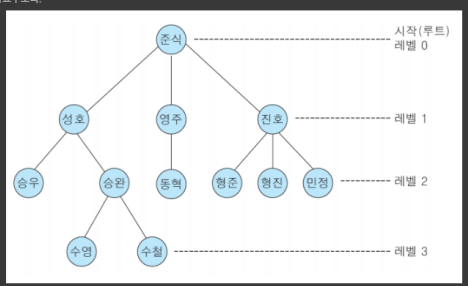
Ch 4. Tree

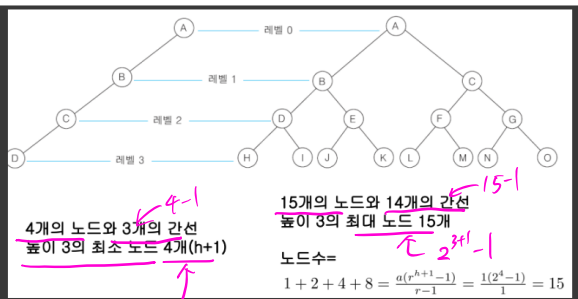
- 트리는 아래와 같은 자료구조다.



- 이진트리는 트리 중에 자식노드가 최대 두개인 트리를 말한다.
  - 자식이 없는 노드를 단말노드라고 한다.
  - 자식이 하나만 있을 수도 있다. 나머지 하나는 공백노드다.
  - 노드가 n개이면 간선은 n-1 개이다.
  - 높이가 h인 노드의 수 최소값은 $h+1$이고 최대값은 $2^{h+1}-1$이다.

모든 트리에 적용됨.

즐기선



4-1

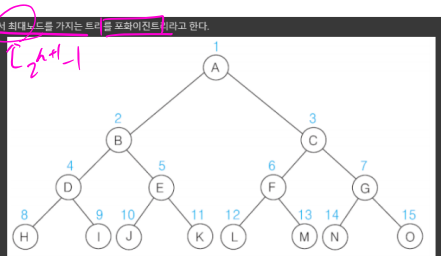
15-1

$2^{3+1}-1$

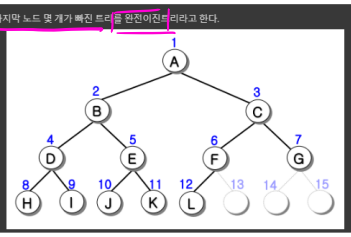
3+1

- 높이가 h인 이진트리에서 최대노드를 가지는 트리를 포화이진트리라고 한다.

$2^{h+1}-1$



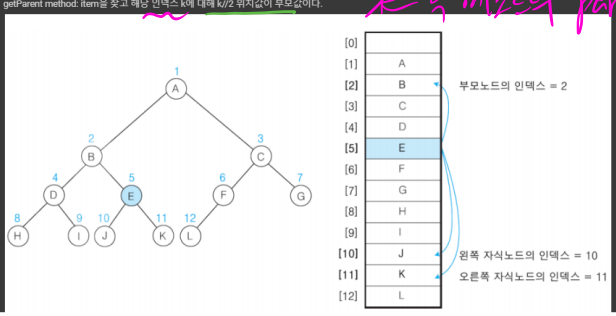
- 포화이진트리에서 아래 그림처럼 마지막 노드 몇 개가 빠진 트리를 완전이진트리라고 한다.



- 리스트를 이용한 이진트리의 ADT
  - 완전 이진트리는 순차구조이므로 배열 형태로 구현할 수 있다. 단, 루트의 인덱스는 1로 시작한다.
    - t = [None]
  - append method: 마지막 위치에서 item을 삽입한다.
  - getChild method: item 을 찾고 이 위치 인덱스 k에 대해 t[2k], t[2k+1]이 좌, 우 자식값을 리턴한다.
  - getParent method: item을 찾고 해당 인덱스 k에 대해 k//2 위치값이 부모값이다.

트리자료구조의 인덱스는 1부터 시작한다.

두 메소드의 parameter에는 특정 index 값이 들어간다.



```
class BinaryTree:
    def __init__(self):
        self.t = [None]

    def append(self, item):
        self.t.append(item)

    def size(self):
        return len(self.t) - 1

    def getChild(self, item):
        if item in self.t:
            k = self.t.index(item)
            lidx = 2 * k
            ridx = 2 * k + 1
            if lidx <= self.size():
                lnode = self.t[lidx]
            else:
                lnode = None
            if ridx <= self.size():
                rnode = self.t[ridx]
            else:
                rnode = None
            return lnode, rnode
        else:
            print('item not found~')

    def getParent(self, item):
        if item in self.t:
            k = self.t.index(item)
            pidx = k // 2
            if pidx > 0:
                return self.t[pidx]
            else:
                return None
        else:
            print('item not found~')

t = BinaryTree()
for i in range(10):
    t.append(chr(65+i))
```

```
print(t.getChild('G'))
print(t.getChild('E'))
```

```
('F', 'G')
('J', None)
```

빈 노드가 아니라, index가 0인 자리에 'None'을 미리 집어넣는다. ⇒ 트리의 index는 1부터임.

핵심!!! 리스트 내 첫번째 요소를 세면 안된다.

해당 트리 자료구조 내 트리 값들의 갯수.

핵심. (부모 노드의 인덱스를 'k'라고 할 때, 자식 노드의 인덱스는 각각 k, k+1 이다.

자식들의 인덱스 넘버가 트리에 있는 모든 노드의 갯수보다 커서는 안된다.

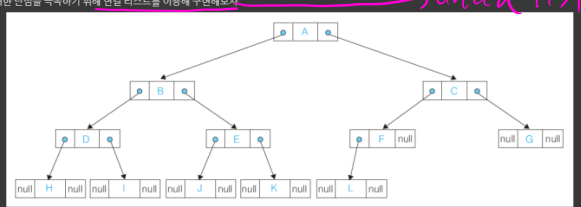
핵심!!! (자식노드의 인덱스를 'i'라고 할 때, 부모노드의 인덱스는 i//2 이다.

부모의 인덱스 넘버는 당연히 '0'보다 크다.

연결 리스트를 이용한 이진트리

- 위에서 구현한 리스트를 이용한 이진트리는 편향트리와 같이 중간이 비어있는 구조에는 적합치 않다.
- 이러한 단점을 극복하기 위해 연결 리스트를 이용해 구현해보자.

Linked list Tree는 중간에 노드들이 비어있는 구조에 적합하다.




```
class BNode:
    def __init__(self, item):
        self.item=item
        self.left = None
        self.right = None
    def setLeft(self, node):
        self.left = node
    def setRight(self, node):
        self.right = node

class BinaryTree:
    def __init__(self,root):
        self.root = root

a = BNode('A')
b = BNode('B')
c = BNode('C')
d = BNode('D')
e = BNode('E')
f = BNode('F')
g = BNode('G')
h = BNode('H')
i = BNode('I')
j = BNode('J')
k = BNode('K')

a.setLeft(b)
a.setRight(c)
b.setLeft(d)
b.setRight(e)
c.setLeft(f)
c.setRight(g)
d.setLeft(h)
e.setLeft(i)
e.setRight(j)
g.setRight(k)

t = BinaryTree(a)
```

노드의 왼쪽 오른쪽 설정.

해당 메소드들의 파라미터에 들어가는 객체는 Node class type의 객체가 들어간다.

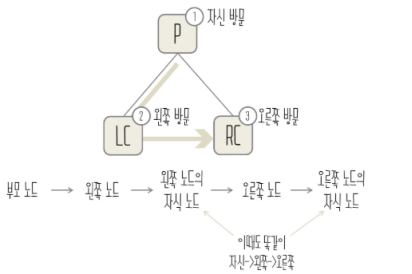
맨 꼭대기에 위치한 노드

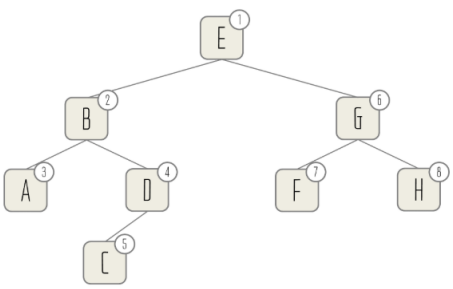
해당 파라미터에 리얼 최초 생성했던 BNode class type 객체를 self.root에 바인딩만 하면 끝.

※ 해당 자료구조들 내부에 Node class type의 객체가 원소로써 들어가는 형태임.

※ 트리 자료구조는 '비선형' 자료구조이다. 그래서 트리 자료만의 '순회 경로' 세 가지가 존재한다.

전위

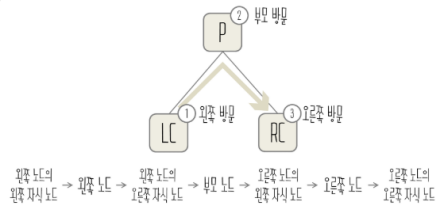


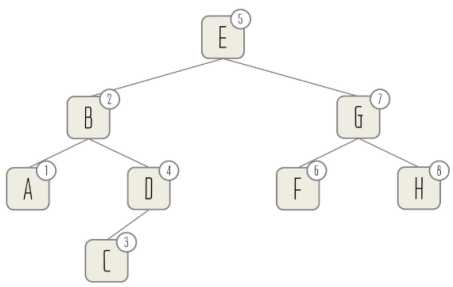


전위 순회 결과 : E -> B -> A -> D -> C -> G -> F -> H

▪ 중위 순회

중위

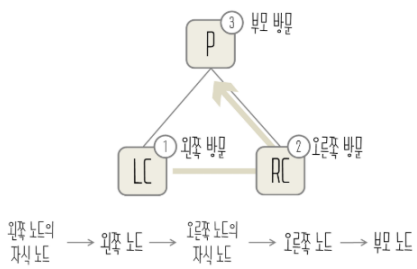


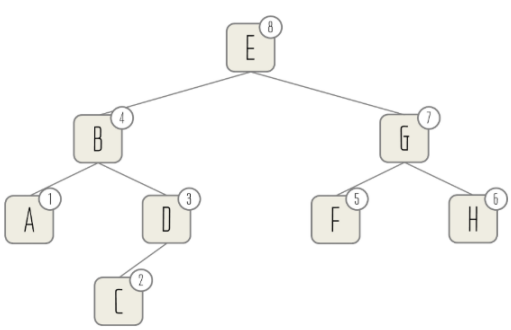


중위 순회 결과 : A -> B -> C -> D ->E -> F -> G -> H

▪ 후위 순회

후위





후위 순회 결과 : A -> C -> D -> B ->F -> H -> G -> E