∘ 입력

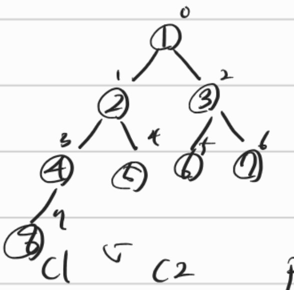
N : 노드 개수 1~26 ← int => tree의 max_size = 26 x 3.

tree = [[0]*3 for _ in range(26)]

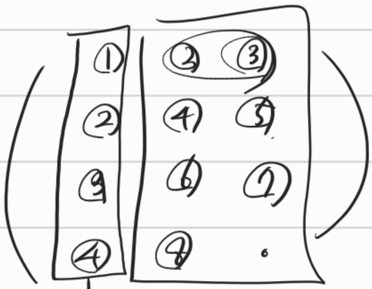
for _ in rang(N).
  ?

어떻게 표현할 것이냐? tree = ?

p cl cr.



2·N +1 lch



=> ?

i	c1	c2	p
0			
1	2	3	
2	4	5	1
3	6	7	1
4	8		2
5			2
6			3
7			3
8			4

tree[P][0]
1
2

→ tree에서 i 역할을 해줘야함
(즉 문자일 경우 해당 인덱스를 대응하는
숫자로 치환해야한다. => 아스키코드

입력이 알파벳으로 되있기 때문에 해당 알파벳을 숫자로 바꿔줘야한다
=> 아스키 코드를 활용하자. ord("A") = 65. A=1 ~ Z=26. 오류설정

모든 문자는 이미 고유의 "수"이다.

A ~ Z = 65 ~ 90. (총 26개)
a ~ z = 97 ~ 122 (소문자 / 소 어링?)

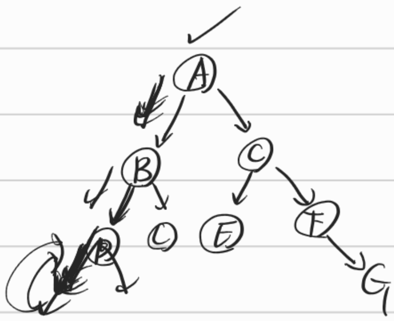
문자열 받아서 아스키코드로 치환해 tree채우기.

for _ in range(N):
  p, c1, c2 ← str, map. split() ※문자로 받음.
  ※ tree 채우기. * A=0 임을 유의해라
  tree[ord(P)-65][0] = C1.
  " [1] = C2
  tree[ord(C1)-65][3] = P.
  " C2 " = P
} tree 완성.

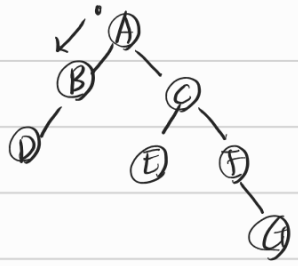
0
"

∘ 순회 일반 방식.
1) pre order_traversal
2) in order_traversal
3) post order_traversal

1) 전위 순회.
 ∘ root에서 시작
 ∘ 왼자식 탐색, 있으면 재귀
 ∘ 오른쪽 탐색. 있으면 재귀



중위 순회

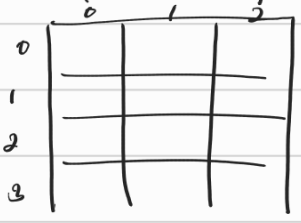


<중요 개념>

- tree

이진트리는 무조건 c1 c2 p 로 저장하자. (자른 방식도 있겠지만 일단 앞으로 계속 까지)

tree =



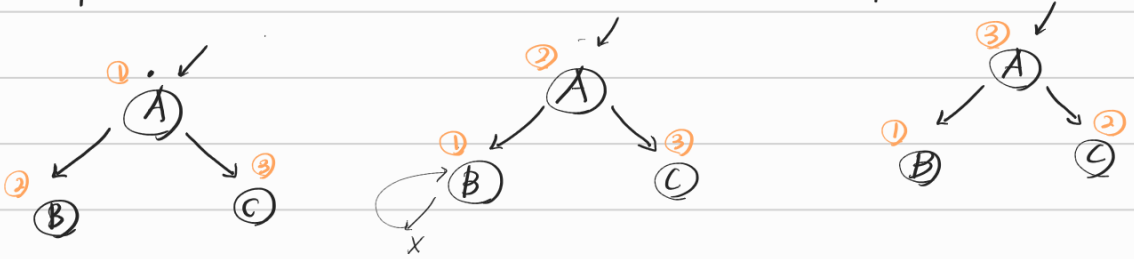
- 구체적인 절차 tip.

1. node의 Content가 숫자이면 유니코드를 활용해 "수"로 바꾼다.
2. tree[p_idx][0] = c1.
   tree[p_idx][1] = c2.
   tree[c1_idx][2] = p
   tree[c2_idx][2] = p
   } tree 한 행 채우기

<순회하기>

- 기본적 순회 형태는 재귀이다.

① preorder (전위순회) ② inorder (중위 순회) ③ postorder (후위순회)




```
def traversal(root)
    ※ print(root)  ← 전위순회
    if 왼쪽 자식이 있으면 => 재귀
    ※ print(root)  ← 중위순회
    if 오른쪽 자식이 있으면 => 재귀
    ※ print(root)  ← 후위순회
```