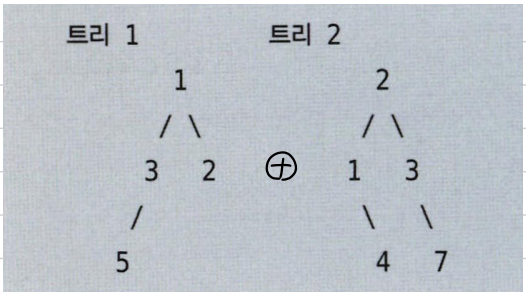
① 이해

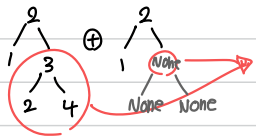


- 두 트리의 같은 자리끼리의 합을 노드로 하는 트리를 구하여라.

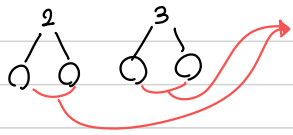
② 계획

BFS

- 위 → 아래로 내려가며 ①, ② 트리의 동일한 위치의 노드를 큐에 넣어라 빼며 노드끼리의 합 계산.
  - 동일한 위치에 있기 불가능한 경우로 존재 (ex) 한 쪽 트리의 노드만 None 인 경우)



이 경우 계산을 어떻게 ...? → 자식 노드가 없는 경우에는 None을 큐에 append하여 동일한 위치로 맞춤.



- 있는 경우 / 없는 경우 케이스 분류해서 해결 → 케이스가 너무 많은데 ...
- 자식 간의 관계가 중요하기에 사용시 노드 하위에서는 합만 처리
  - 노드 자식 하위에서 큐에 append 할지 결정.

③ 풀이 코드 (1)

1. BFS

- 내가 짠 코드.

```
class Solution:
    def mergeTrees(self, root1: Optional[TreeNode], root2: Optional[TreeNode]) -> Optional[T
        if not root1 : return root2
        if not root2 : return root1

        queue1 = collections.deque([root1])
        queue2 = collections.deque([root2])

        while queue1 or queue2 :
            node1 = queue1.popleft()
            node2 = queue2.popleft()

            node1.val = node1.val + node2.val

            if node1.left :
                if node2.left :
                    queue1.append(node1.left)
                    queue2.append(node2.left)
                else :
                    pass
            else :
                if node2.left :
                    node1.left = node2.left
                else : pass

            if node1.right :
                if node2.right :
                    queue1.append(node1.right)
                    queue2.append(node2.right)
                else :
                    pass
            else :
                if node2.right :
                    node1.right = node2.right
                else : pass

        return root1
```

BFS를 위한 큐

계산 결과를 node1에 다시 합쳐 넣어 위 → 아래로 node1 → 최종 트리의 노드를 교체.

왼쪽 자식 노드

오른쪽 자식 노드 (왼쪽 자식의 계산과 동일)

① node 1, 2 자식 노드가 모두 존재
: 자식도 계산을 위해 큐에 append

② node 1 의 자식만 있는 경우
: node 1의 자식을 타고 내려가도 node 2는 계속 None 이기에 뒤의 계산이 필요X (큐에 append X)
- 최종 트리의 노드 == node 1

③ node 2의 자식만 있는 경우
∴ ②와 동일한 이유로 큐에 append X
- 최종 트리의 노드 == node 2

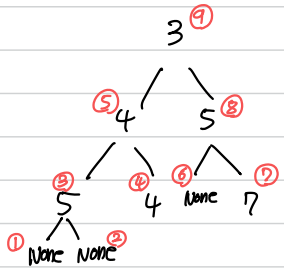
④ 둘 다 자식이 없는 경우: 최종 트리의 리프노드.

③풀이코드(2)

재귀,,

```
class Solution:
    def mergeTrees(self, root1: Optional[TreeNode], root2: Optional[TreeNode]) -> Optional[Tre
        if root1 and root2 :
            Node = TreeNode(root1.val + root2.val)
            Node.left = (self.mergeTrees(root1.left, root2.left))
            Node.right = (self.mergeTrees(root1.right, root2.right))
            return Node
        else : return root1 or root2
```

- 노드 합 계산
- 자식 노드 계산
- 밑에서 올라가며 노드를 리턴 → 제일 마지막에 최종 트리를 return



- 맥트래킹 되는 순서,

둘 중 하나만 존재할 경우 : 하나의 node만 return
둘 다 None인 경우 : None return
최종 노드의 리프 노드,,

비고

- return 객체 1 or 객체 2

⇒ 위 코드는 · 객체 1,2 중에 빈 컨테이너가 아니거나 None이 아닌 객체를 return
· 둘 다 빈 컨테이너거나 None인 경우 빈 컨테이너를 return.

ex) · return [] or [1,2,3] => [1,2,3] return
· return [] or [] => 빈 컨테이너 return
가정을에서 거짓일 경우

· 참고: 만약 둘 다 빈 컨테이너가 아닐 경우 "객체 1"을 return,,