① 이해

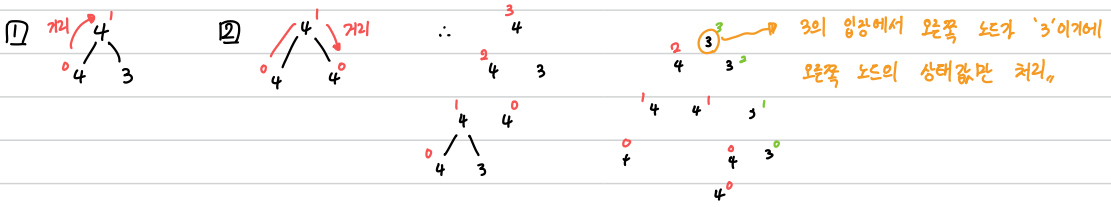
트리 상에서 동일한 값으로 연결된 가장 긴 길이를 구하라

② 계획

* 알고리즘 : DFS ⇝ 재귀

• 자식 노드와 같으면 '상태값++' 하는 식으로 상태값을 업데이트하며 길이도 업데이트

ex)



• 재귀를 사용하여 DFS로 리프노드까지 탐색 후 거슬러 올라오면서 상태값, 거리 계산,,

• 4 ⇝ 이런 경우는 'is not node.left 로 처리,,
and not node.right'

4 ⇝ 이 경우는 어떻게 처리??

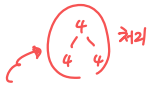
왼/오른쪽에만 노드가 있는 경우를
따로 처리?

③최종 코드

한 쪽 노드만 있는 경우를 처리하기 위해서 모든 경우를 고려해서 코드를 짬

코드가 너무 복잡,,

```
class Solution:
    longest = 0
    def longestUnivaluePath(self, root: Optional[TreeNode]) -> int:
        def dfs(node) :
            if not node : return -1                                  # 빈 트리 처리
            if not node.left and not node.right : return 0           # 자식 노드가 없을 경우
            elif node.left and not node.right :                      # 왼쪽 자식 노드만 있을 경우
                left = dfs(node.left)

                if node.val == node.left.val :
                    distance = left + 1
                    if self.longest < distance : self.longest = distance
                    return left + 1
                else : return 0
            elif node.right and not node.left :                      # 오른쪽 자식 노드만 있을 경우,,
                right = dfs(node.right)

                if node.val == node.right.val :
                    distance = right + 1
                    if self.longest < distance : self.longest = distance
                    return right + 1
                else : return 0
            else :                                                   # 왼/오른쪽 자식 모두 있을 경우,,
                left = dfs(node.left)
                right = dfs(node.right)

                if node.val == node.left.val and node.val == node.right.val :
                    distance = left + right + 2
                    if self.longest < distance : self.longest = distance
                    return max(left,right) + 1

                elif node.val == node.left.val :
                    distance = left + 1
                    if self.longest < distance : self.longest = distance
                    return left + 1

                elif node.val == node.right.val :
                    distance = right + 1
                    if self.longest < distance : self.longest = distance
                    return right + 1

                else : return 0

        dfs(root)
        return self.longest
```



④ 교재 코드 „

- 계산하는 부분은 비슷
- 왼/오른쪽 노드 처리 방법의 차이 „

```python
class Solution:
    result: int = 0

    def longestUnivaluePath(self, root: TreeNode) -> int:
        def dfs(node: TreeNode):
            if node is None:
                return 0

            # 존재하지 않는 노드까지 DFS 재귀 탐색
            left = dfs(node.left)
            right = dfs(node.right)

            # 현재 노드가 자식 노드와 동일한 경우 거리 1 증가
            if node.left and node.left.val == node.val:
                left += 1
            else:
                left = 0
            if node.right and node.right.val == node.val:
                right += 1
            else:
                right = 0

            # 왼쪽과 오른쪽 자식 노드 간 거리의 합 최댓값이 결과
            self.result = max(self.result, left + right)
            # 자식 노드 상태값 중 큰 값 리턴
            return max(left, right)

        dfs(root)
        return self.result
```

: if 문의 앞쪽에 노드 존재 여부를 두어 없을 경우 쉽게 넘어가도록 처리 „

같을 경우 '+1' 계산을 하며 해당 노드를 기준으로 왼/오른쪽 연결하여 거리를 계산

⑤ 비고 „

* 이 문제의 코드를 처음 짤 때, None(존재하지 않는 노드)의 value (속성)을 참고하며 "AttributeError"가 발생

↓ 해결하기 위해 모든 케이스(왼쪽 노드만 있는 경우/오른쪽 " / 둘 다 " / 둘 다 없는 경우)를 고려

↓ 식이 복잡 „

☆ 교재 코드에서는 "Short Circuit evaluation"을 통한 장점으로 AttributeError를 피해감 „

: 논리연산자 and/or 연산자의 경우 결과가 확정되면 뒤 연산은 아니옴.

ex) if node.left and node.left.value == node.value :

① 존재하지 않으면 (거짓) → 이 방식으로 AttributeError를 피해감 „

② 존재하면 "node.left.value == node.value" 연산