

# 고 급 문 제 해 결

## <문제 9.2>

이진 트리가 대칭인지 알아보기

# Chapter 9

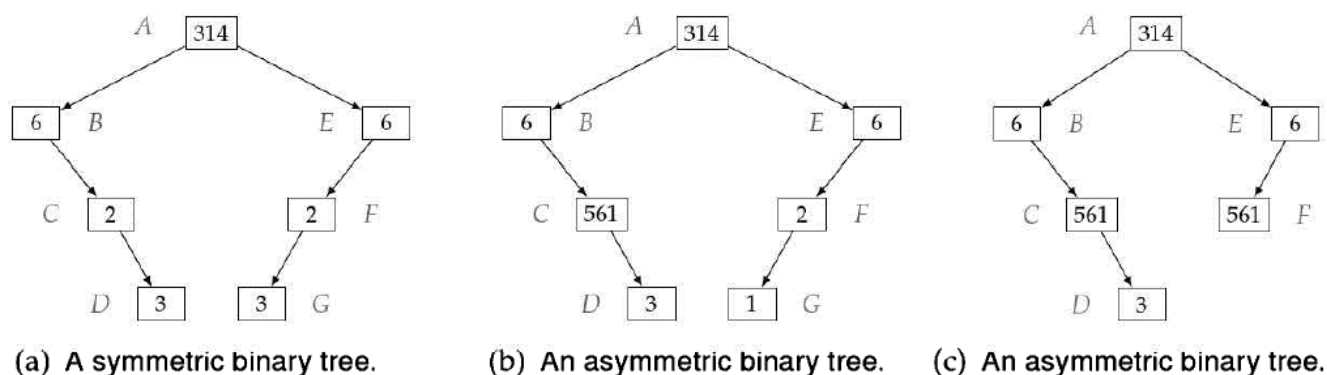
## Binary Tree

# 9.2 이진 트리가 대칭인지 알아보기

## 10.2 TEST IF A BINARY TREE IS SYMMETRIC

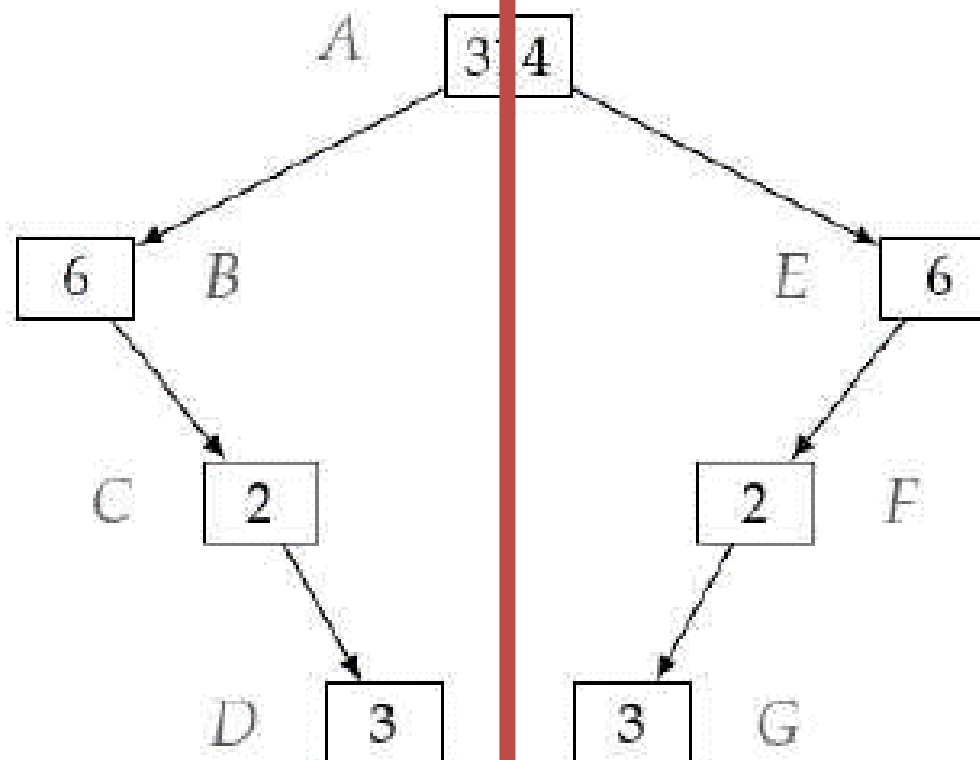
A binary tree is symmetric if you can draw a vertical line through the root and then the left subtree is the mirror image of the right subtree. The concept of a symmetric binary tree is illustrated in Figure 10.3 on the facing page.

Write a program that checks whether a binary tree is symmetric.



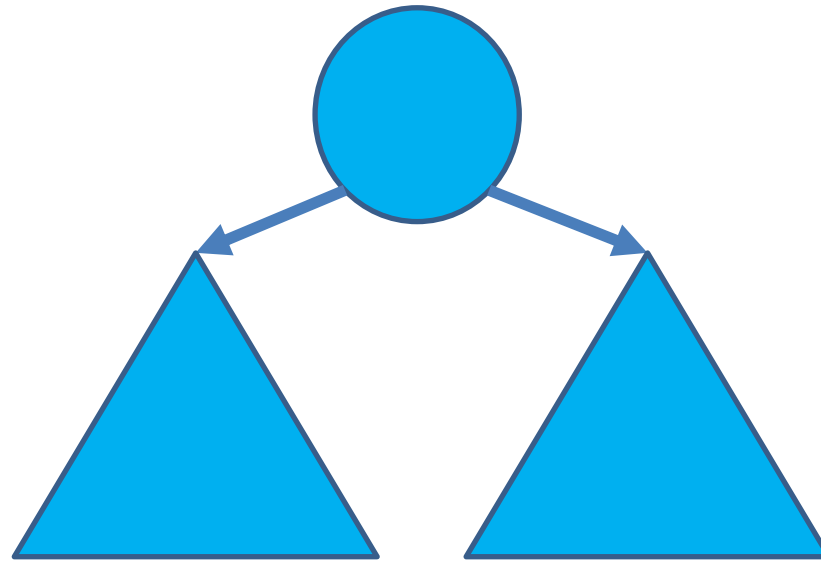
**Figure 10.3:** Symmetric and asymmetric binary trees. The tree in (a) is symmetric. The tree in (b) is structurally symmetric, but not symmetric, because symmetry requires that corresponding nodes have the same keys; here *C* and *F* as well as *D* and *G* break symmetry. The tree in (c) is asymmetric because there is no node corresponding to *D*.

## 9.2 이진 트리가 대칭인지 알아보기



(a) A symmetric binary tree.

# Recursion

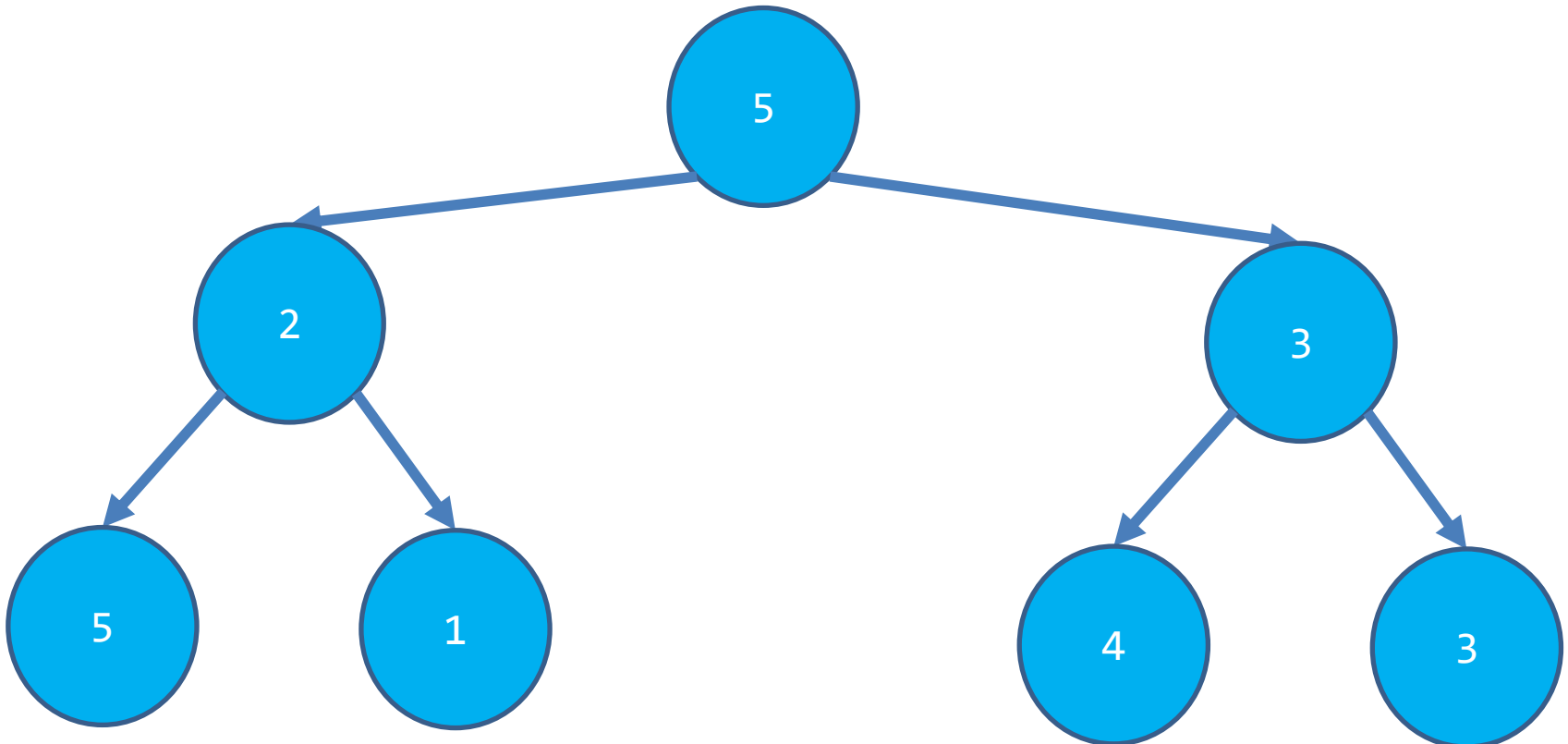


Recursion

```
def is_symmetric(T):  
    if T is None:  
        return True  
    if not T.left and not T.right:  
        return True  
    if (T.left and not T.right) or (not T.left and T.right):  
        return False  
    return is_symmetric(T.left) and is_symmetric(T.right)
```

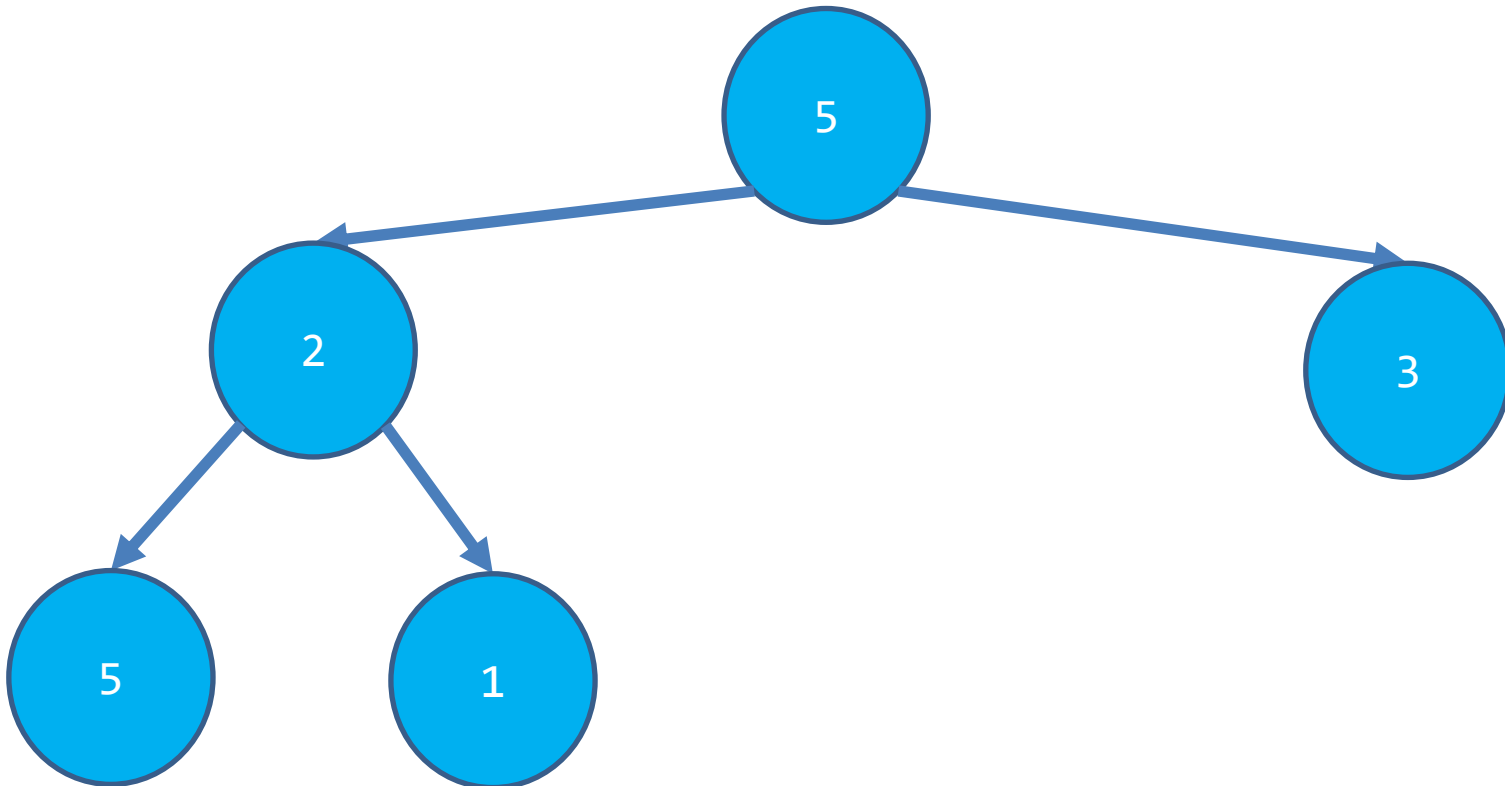
# 값이 달라도 True

```
def is_symmetric(T):  
    if T is None:  
        return True  
    if not T.left and not T.right:  
        return True  
    if (T.left and not T.right) or (not T.left and T.right):  
        return False  
    return is_symmetric(T.left) and is_symmetric(T.right)
```



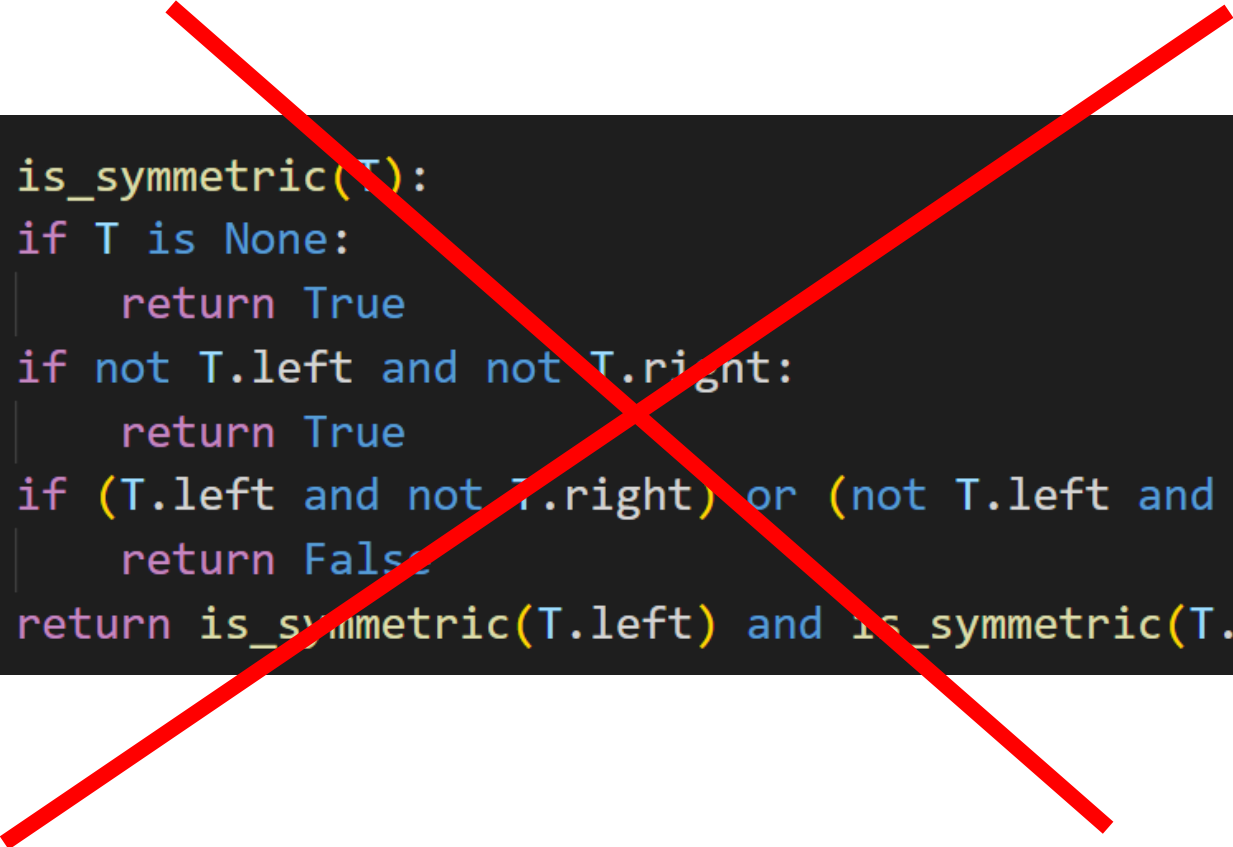
# 트리 한쪽만 대칭이어도 True

```
def is_symmetric(T):  
    if T is None:  
        return True  
    if not T.left and not T.right:  
        return True  
    if (T.left and not T.right) or (not T.left and T.right):  
        return False  
    return is_symmetric(T.left) and is_symmetric(T.right)
```



# 결론

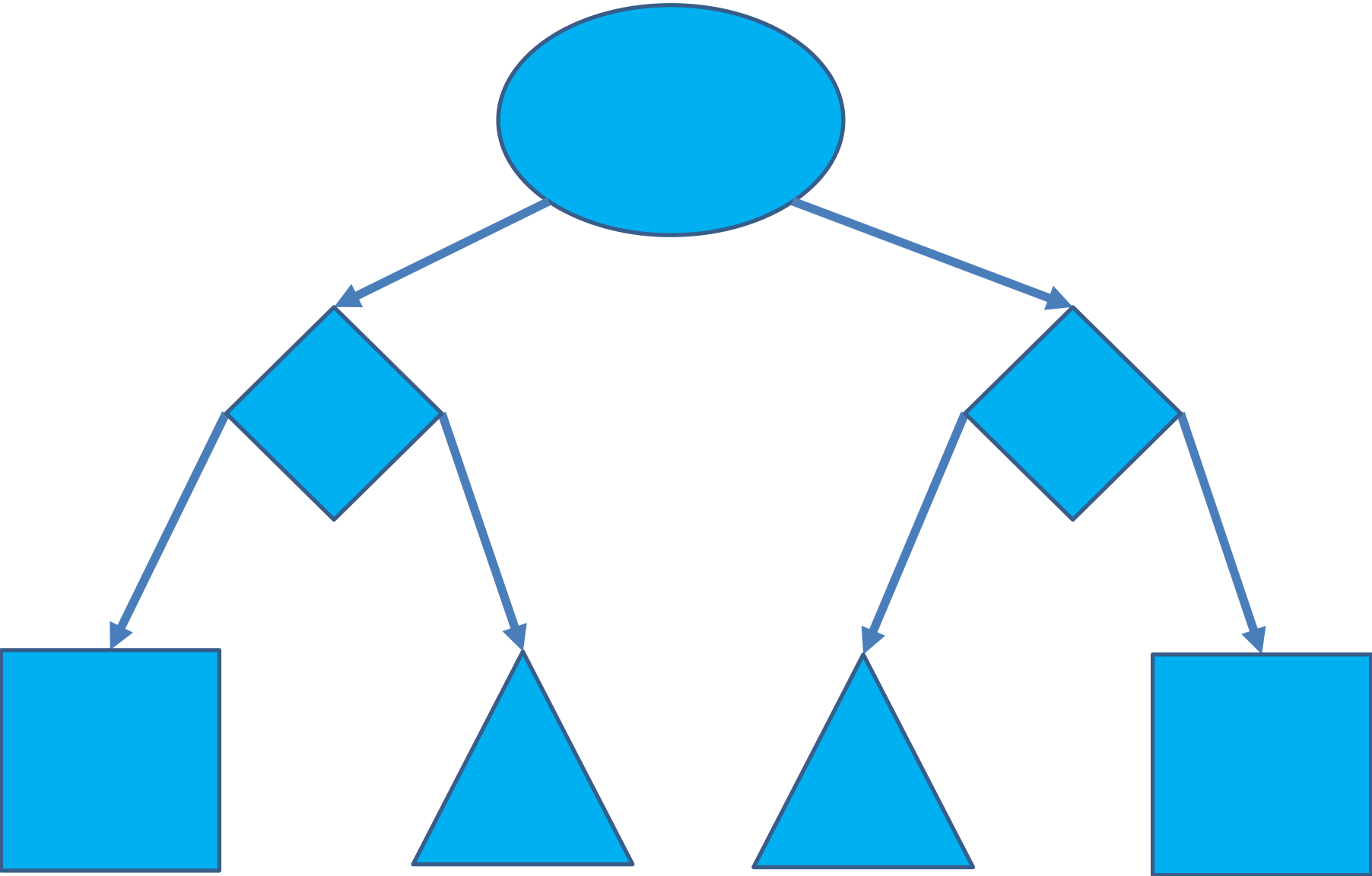
해당 코드는 결론적으로,  
Child node가 0개 또는 2개라면 True를 반환



```
def is_symmetric(T):  
    if T is None:  
        return True  
    if not T.left and not T.right:  
        return True  
    if (T.left and not T.right) or (not T.left and T.right):  
        return False  
    return is_symmetric(T.left) and is_symmetric(T.right)
```

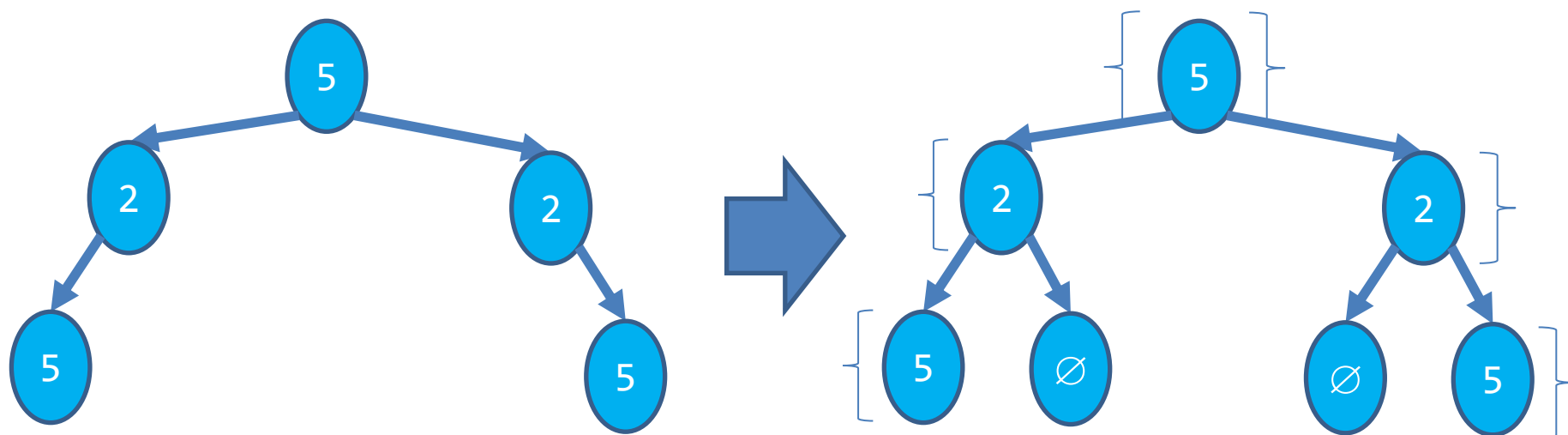


# Recursion



# Iteration #1

- 핵심 Idea:
  - ✓ Binary Tree -> FBT로 변형
  - ✓ 각 Level 단위로 List를 만들어 보아서
  - ✓ 해당 Level에 해당하는 값이 palindrome인지 확인하기



# Code

```
def isSymmetric1(self, root: Optional[TreeNode]) -> bool:

    if root is None:
        return True

    def height(T):
        if T is None:
            return 0
        return max(height(T.left), height(T.right)) + 1

    Q = deque([(root, 1)])
    L = []
    h = height(root)
    prev_lv = 1
    is_sym = True

    while Q:
        x, lv = Q.popleft()
        if lv != prev_lv:
            print(L, prev_lv)
            if L != L[::-1]:
                is_sym = False
                break
            L = []
            prev_lv = lv

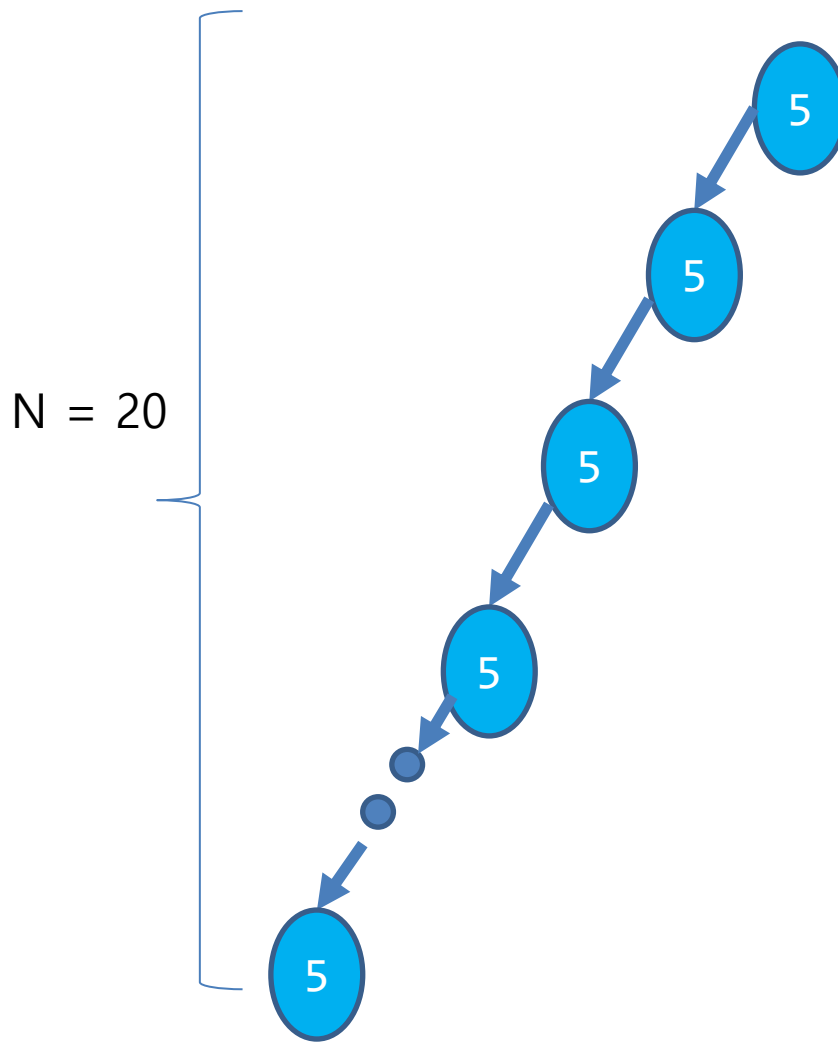
        if lv <= h:
            if x:
                Q.append((x.left, lv + 1))
                Q.append((x.right, lv + 1))
                L.append(str(x.val))
            else:
                Q.append((None, lv + 1))
                Q.append((None, lv + 1))

                L.append('None')

    return is_sym
```

# 문제점 – 시간 초과

- 시간 초과!
- 왜 시간 초과가 일어날까?
  - ✓ EX ) Skewed binary tree



FBT 노드 개수 =  $2^h - 1$

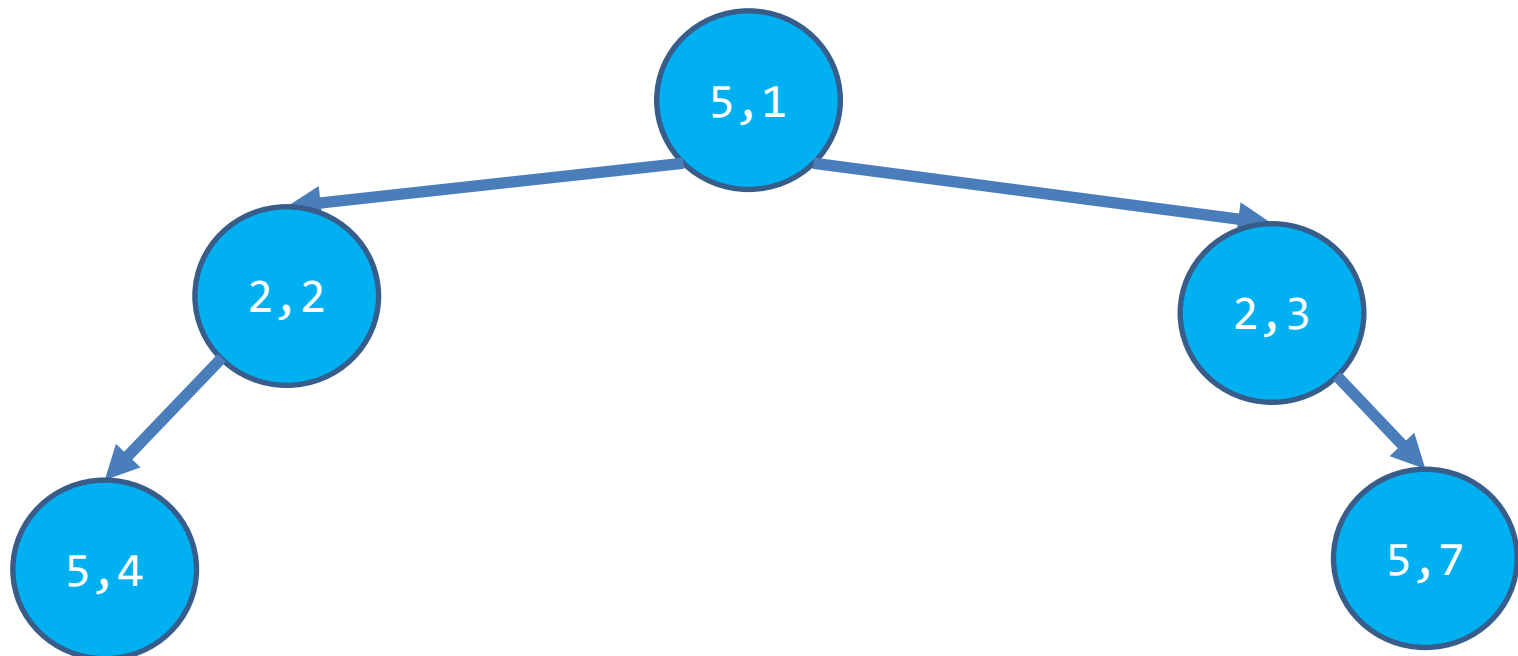
n = 20 일 때,  $2^{20} - 1 \approx 10^6$

Time Limit Exceeded!

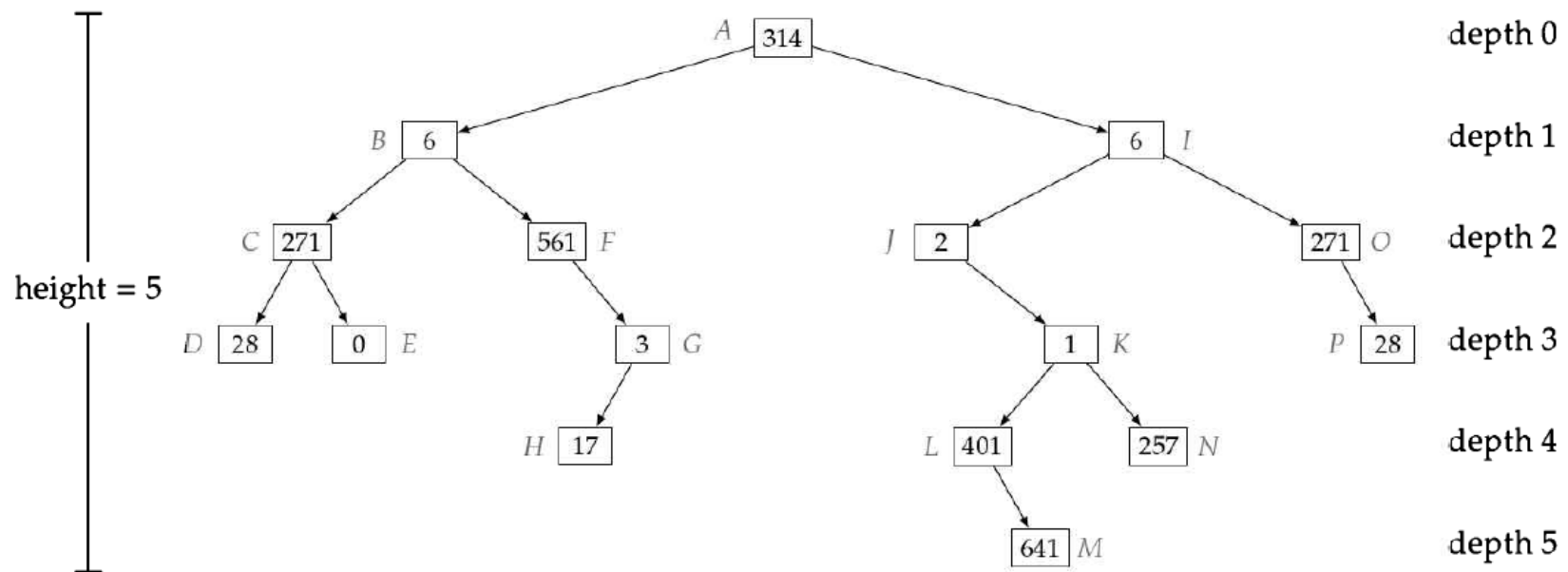
# Iteration #2

- 핵심 Idea:

- ✓ Binary Tree => FBT 처럼, 그러나 Null이 아닌 Node들만!
  - 필요한 Node들만 보되, FBT기준 몇 번째 노드인지를 계산
  - 몇 번째 노드인지 계산하는 방법은, Heap index계산할때와 같음
  - 1.값 일치 2. 몇 번째 노드인가? -> 대칭성을 사용해서 판단

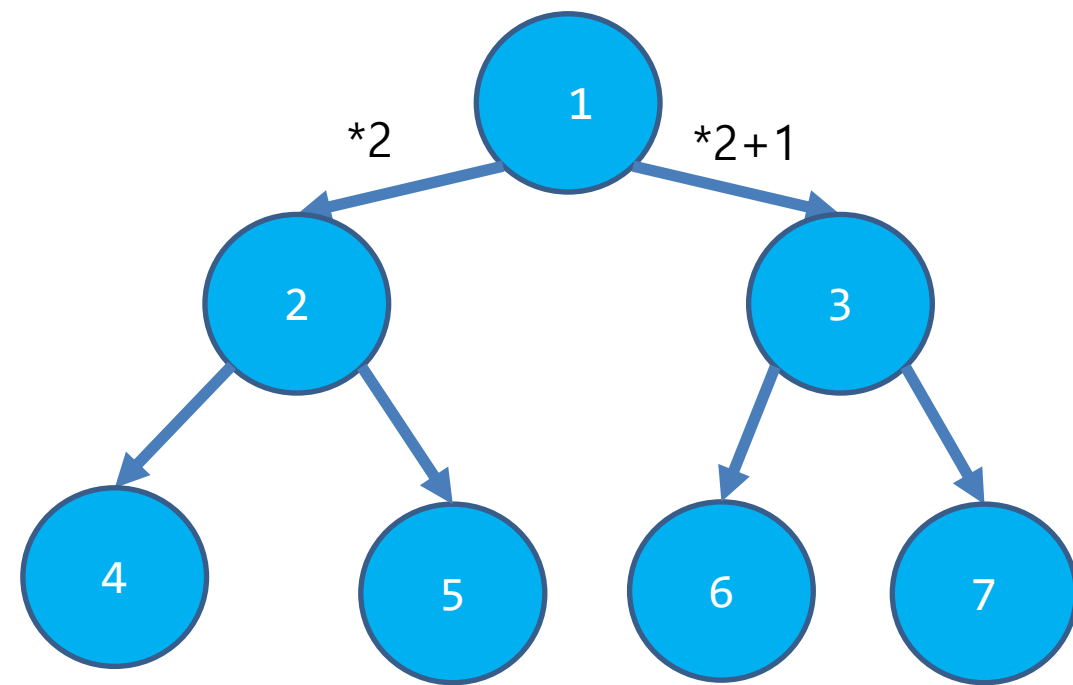


# Level order traverse



**Variant:** Write a program which takes as input a binary tree and returns the keys in top down, alternating left-to-right and right-to-left order, starting from left-to-right. For example, if the input is the tree in Figure 10.1 on Page 150, your program should return  $\langle\langle 314 \rangle, \langle 6, 6 \rangle, \langle 271, 561, 2, 271 \rangle, \langle 28, 1, 3, 0, 28 \rangle, \langle 17, 401, 257 \rangle, \langle 641 \rangle\rangle$ .

# FBT 일때 index 값 계산



$$depth = 0 \rightarrow [2^0, 2^{0+1})$$

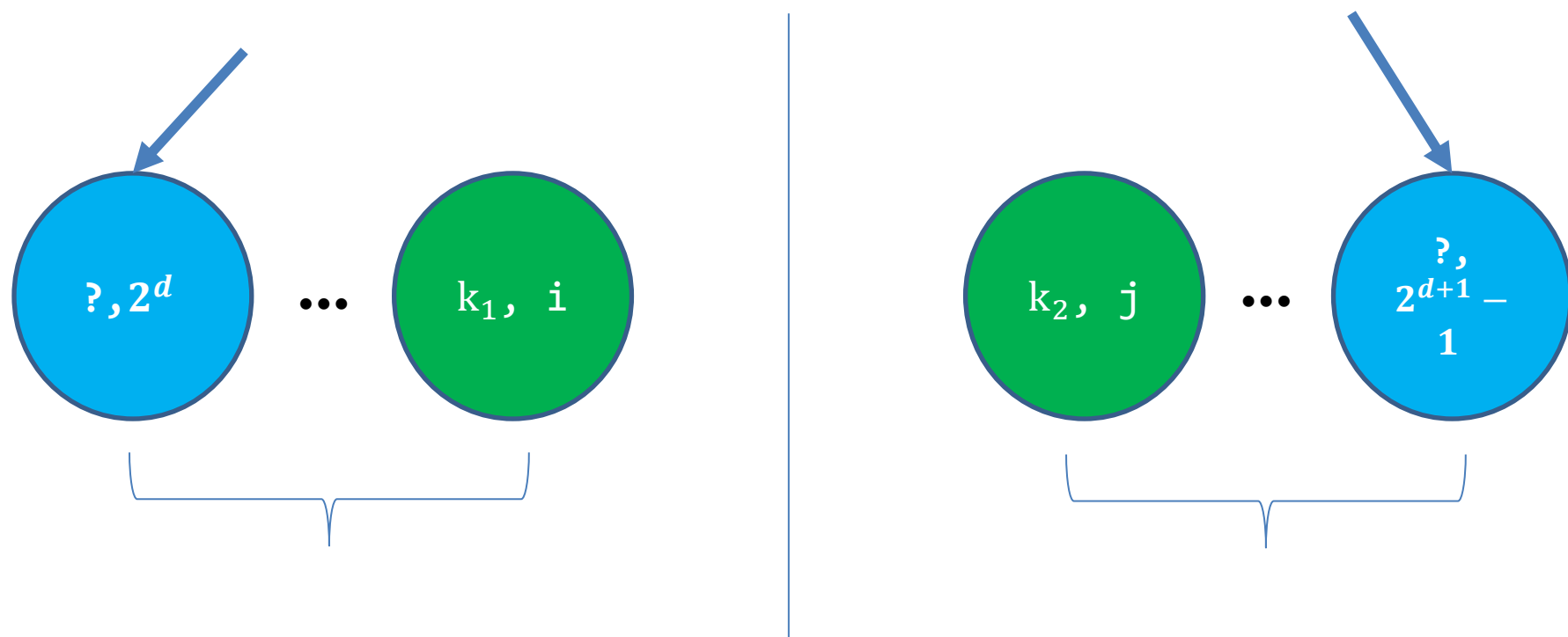
$$depth = 1 \rightarrow [2^1, 2^{1+1})$$

$$depth = 2 \rightarrow [2^2, 2^{2+1})$$

$$depth = d \rightarrow [2^d, 2^{d+1})$$

$$depth = d \rightarrow \text{맨 왼쪽: } 2^d, \text{ 맨 오른쪽: } 2^{d+1} - 1$$

# 각 depth마다 끝 노드로부터 얼마나 떨어져 있는가 계산



$depth = d \rightarrow$  맨 왼쪽:  $2^d$  , 맨 오른쪽 :  $2^{d+1} - 1$

$$i - 2^d = 2^{d+1} - 1 - j \text{ and } (k_1 = k_2)$$



# Code

```
def isSymmetric2(self, root: Optional[TreeNode]) -> bool:
```

```
    if root is None:
```

```
        return True
```

```
    Q = deque([(root, 1)])
```

```
    lv = 0
```

```
    lower = 2 ** (lv)
```

```
    upper = 2 ** (lv + 1)
```

```
    mid = (lower + upper - 1) // 2
```

```
    L = []
```

```
    level = []
```

```
    while Q:
```

```
        node, num = Q.popleft()
```

```
        if not (lower <= num < upper):
```

```
            L.append(list(level))
```

```
            level = []
```

```
            lv += 1
```

```
            lower = 2 ** (lv)
```

```
            upper = 2 ** (lv + 1)
```

```
        if node.left:
```

```
            Q.append((node.left, num * 2))
```

```
        if node.right:
```

```
            Q.append((node.right, num * 2 + 1))
```

```
        level.append((node, num))
```

```
    L.append(list(level))
```

Level order traverse

# Code

```
for lv in range(1, len(L)):
    lower = 2 ** (lv)
    upper = 2 ** (lv + 1)
    mid = (lower + upper - 1) // 2
    left_idx, right_idx = 0, len(L[lv]) - 1
```

```
    if len(L[lv]) % 2 == 1:
        return False
```

```
    while left_idx < right_idx:
```

```
        if left_idx > mid:
            return False
```

```
    left = L[lv][left_idx]
```

```
    left_val, left_num = left[0].val, left[1]
```

```
    right = L[lv][right_idx]
```

```
    right_val, right_num = right[0].val, right[1]
```

```
    offset = left_num - lower
```

```
    if (upper - 1 - right_num != offset) or (left_val != right_val):
        return False
```

```
    left_idx += 1
```

```
    right_idx -= 1
```

```
return True
```

# Summary

- 문제 9.2:이진 트리가 대칭인지 알아보기
- Recursion
- Iteration #1 - FBT를 진짜로 만들었음 (TLE)
- Iteration #2 - 필요한 Node들에 대해 FBT기준으로 몇 번째 Node인지 계산하였음
  - ✓ Queue를 이용한 Level order traversal
  - ✓ Full Binary Tree depth, 각 level당 node갯수, 등 정의 사용
  - ✓ 수식을 이용하여 논리적 전개
- 사실 재귀의 정의를 그대로 사용해서 Queue 2개로 풀이가능

들어 주셔서 감사합니다

