

① 이해

노드 개수, 무방향 그래프를 입력 받아서 트리가 최소 높이가 되는 루트 목록을 반환.

ex) $n=4$, $edge = [[1,0], [1,2], [1,3]]$

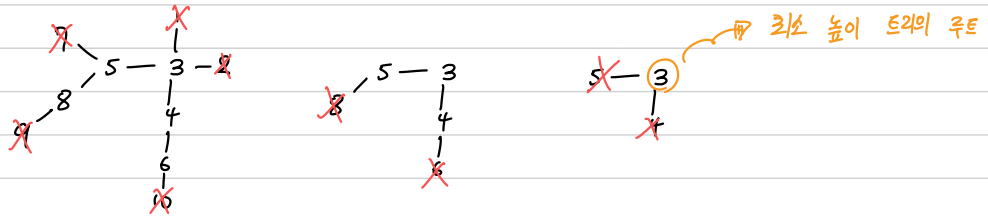


$A = [1]$

② 계획
단계별 노드
제거

· 최소 높이의 트리를 만들기 위해서 무방향 노드를 이어서 제일 끝에 있는 노드를 반복하여 제거하면 된다.
(마지막 트리에서 아래 \rightarrow 위 노드를 제거해 올라가는 식)

ex) $n=10$, $edge = [[1,3], [2,3], [3,4], [3,5], [4,6], [6,10], [5,7], [5,8], [8,9]]$

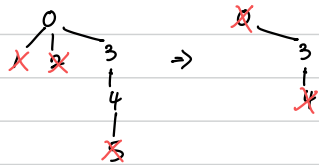


1. 간선들을 defaultdict에 저장

2. defaultdict의 키들을 돌면서 간선이 하나인 (key의 value가 하나 / 제일 끝 노드를 의미) 키들을 제거

3. 남은 키가 2이하가 될 때까지 반복 (최소 높이 트리의 루트는 2개까지 존재 가능)

ex) $n=6$, $edge = [[0,1], [0,2], [0,3], [3,4], [4,5]]$



③ 풀이 코드 (1)

단순 반복

직관적 풀이

↓

비효율적...

```
class Solution:
```

```
    def findMinHeightTrees(self, n: int, edges: List[List[int]]) -> List[int]:
```

```
        if n == 1 : return [0] ~ 예외 처리
```

```
        dic = collections.defaultdict(list)
```

```
        for i in edges :
```

```
            a,b = i
```

```
            dic[a].append(b)
```

```
            dic[b].append(a)
```

→ Graph를 dictionary에 저장

```
        nodes = dic.keys()
```

```
        while n > 2 :
```

```
            node = []
```

```
            leave_node = []
```

```
            for i in nodes :
```

```
                if len(dic[i]) == 1 :
```

```
                    leave_node.append(i)
```

```
                else : node.append(i)
```

→ dictionary의 value 길이가 1인 키 (리프노드)와 아닌 것을 구분

```
            for i in leave_node :
```

```
                dic[dic[i][0]].remove(i)
```

→ 리프노드 제거

```
            nodes = node
```

```
            n = len(nodes)
```

→ 다음 반복 시에 리프노드를 제외하고 반복,

```
        return nodes
```

③ 풀이코드 (2)

리프노드에

집중해서 반복

```
class Solution:
```

```
def findMinHeightTrees(self, n: int, edges: List[List[int]]) -> List[int]:
```

```
    if n == 1 : return [0]
```

```
    dic = collections.defaultdict(list)
```

```
    for i in edges :
```

```
        a,b = i
```

```
        dic[a].append(b)
```

```
        dic[b].append(a)
```

(1)과 동일

```
    leaves = []
```

```
    for i in range(n+1) :
```

```
        if len(dic[i]) == 1 : leaves.append(i)
```

문제 조건에 0~n-1까지의 노드가 존재한다는 정보 활용

```
    while n > 2 :
```

```
        n -= len(leaves) ↖ 전 행의 노드 개수에서 리프 노드 개수만큼 제거,,
```

```
        new_leaves = []
```

```
        for leaf in leaves :
```

```
            node = dic[leaf].pop()
```

```
            dic[node].remove(leaf)
```

리프노드 제거

```
            if len(dic[node]) == 1 : new_leaves.append(node)
```

리프 삭제 후 다음 리프노드 저장

```
        leaves = new_leaves → 다음 리프노드,,
```

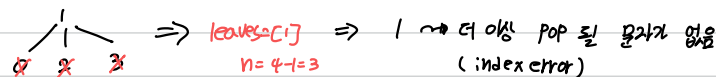
```
    return leaves
```

④ 의문

Q. 왜 n을 while 처음에 업데이트 하나?

A. "leaves = new_leaves" 코드 전에만 업데이트 되면 됨,,

ex) " 이후에 업데이트



⑤ 비교,,

※ 항상 문제 조건을 잘 보자!!

• 전체 노드를 본 필요 없이 필요에 따라 특정 노드에만 집중!!,,

(이 문제의 경우 리프 노드에만 집중),