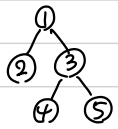
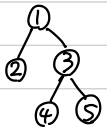
① 이해

· Tree를 직렬화하고, 다시 직렬화한 결과를 역직렬화 하여라. (직렬화의 결과는 문자열로...)

ex)



=> [1,2,3,null,null, 4,5] =>



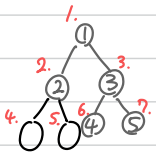
② 계획 (1)

1. 직렬화

=> BFS..

· BFS 가 아닌 DFS를 사용하여 직렬화 하여도 되지만, 'BFS의 탐색 순서 == 직렬화 순서'이기 때문에 BFS 선택..

ex)



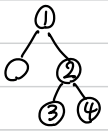
1 2 3 4 5 6 7

=> [1,2,3,null, null, 4,5]

· 큐에서 현재 노드를 빼냄과 동시에 자식 노드들을 append 하면 BFS로 트리 탐색 가능..

· null 값도 직렬화 결과에 포함돼야 하므로 "#"를 대신 직렬화 결과에 append..

테스트 케이스..



=> [1, #, 2, 3,4] => "1 # 2 3 4"

· 순서 :

Q	result
1	
null, 2	1
2	1, #
3, 4	1, #, 2
4, null, null	1, #, 2, 3
null x4	1, #, 2, 3, 4
⋮	
	1, #, 2, 3, 4, #, #, #, #

② 계획 (1)

2. 역직렬화

"큐 이용"

(BFS와 유사하게)

- 제일 위의 노드를 복사 (노드를 내려가며 생성할거기 때문에 제일 위 노드를 저장하면 2 노드는 트리의 루트 노드가 됨)

- 직렬화해서 얻은 data를 큐 형태로 저장 (data_list)
- 큐를 생성

전체적인 순서 : ① 큐에 data_list를 pop해서 노드로 만들어 append

② 큐에서 pop ⇒ 현재 노드

③ 왼쪽 노드 ⇒ data_list에서 pop해서 노드로
왼쪽 자식 · #일 경우 null

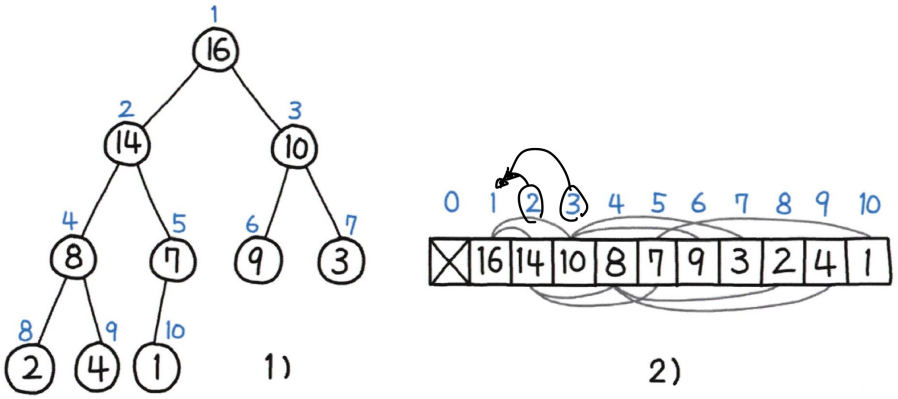
④ 오른쪽 노드 : 〃
오른쪽 자식

⑤ 왼쪽 자식 노드, 오른쪽 〃 을 큐에 append

②~⑤ 큐가 빌 때까지 반복

② 계획 (2)

"인덱스 이용"



*위 그림처럼 해당 노드의 인덱스를 알면 → 깊이, 부모/자식의 인덱스를 특정할 수 있다.
(트리의 배열 표현의 경우 계산상의 편의성을 위해 1부터 시작)

- 해당 노드의 인덱스를 i라고 할 경우 → "부모 [$\frac{i}{2}$], 왼쪽 자식 2i, 오른쪽 자식 2i+1" 이런 식으로 특정이 가능하다.
(위의 그림처럼 굳이 완전 이진 트리의 형태가 아니더라도 비어 있는 위치는 널(Null)로 표현 가능하기 때문에,
※ 사실상 모든 트리는 배열의 형태로 나타낼 수 있다.)

③ 풀이코드 (1)

직렬화 : BFS

```
class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """
        queue = collections.deque([root])
        result = []

        while queue :
            node = queue.popleft()

            if node :
                queue.append(node.left)
                queue.append(node.right)
                result.append(str(node.val))
            else :
                result.append("#")

        print(result)
        return ' '.join(result)
```

위 → 아래로 내려가면서 · node 존재 → 문자로 변환해 저장
· " 존재X → # 저장

return 타입이 string 이기에 join 메서드로 문자열 변환..

역직렬화 : 큐 이용 (BFS와 유사)

```
def deserialize(self, data):
    """Decodes your encoded data to tree.

    :type data: str
    :rtype: TreeNode
    """
    if data[0] == "#" : return None

    data = data.split()
    data_queue = collections.deque(data)

    queue = collections.deque([TreeNode(int(data_queue.popleft()))])
    tree = TreeNode()

    i = 0
    while queue and data_queue :
        node = queue.popleft()

        if i == 0 : tree = node

        if not node : continue
        else :
            value = data_queue.popleft()
            if value == "#" : node.left = None
            else :
                node.left = TreeNode(int(value))
                queue.append(node.left)

            value = data_queue.popleft()
            if value == "#" : node.right = None
            else :
                node.right = TreeNode(int(value))
                queue.append(node.right)
        i += 1

    return tree
```

예외 처리 (빈 트리)

계산을 위해 문자열 → 노드.value / None 이 담긴 큐로...

계산을 위한 큐 생성

루트 노드 저장 노드

루트 노드 저장

노드가 None 일 경우 아무 처리 X (?)

노드.value는 data_큐 에 저장되어 있기에 pop해서 사용..

왼쪽 자식

값 존재 X → 왼쪽 노드 = None..

값이 존재할 경우 · 왼쪽 노드 생성
· 생성 노드를 다시 큐에 append

오른쪽 자식

* data_queue.popleft() 값이 '#'일 경우에는 왜 큐에 대입하지 않는가?

A. 2
3 ≠ [2, #, 3, #, #, 1, 2]
1 2 = [2, #, 3, 1, 2] 로 처리되기 때문에 '#'이 pop 되었을 경우 해당 노드만 None로 처리하면
그 다음 이어지는 자식노드가 없기 때문에 큐에 넣을 필요가 없다.

③ 풀이코드 (2)
: index 이용

직렬화
: BFS 이용

· 직렬화의 경우 거의 동일하고, 트리를 배열화 했을 때 index는 1부터 시작하는게 편하기 때문에 풀이코드(1) 에서 "result=["#"]" 로만 수정해주면 된다.

역직렬화
: index 이용

```
def deserialize(self, data):
    """Decodes your encoded data to tree.

    :type data: str
    :rtype: TreeNode
    """
    if data == "# #" : return None

    data_list = data.split()

    root = TreeNode(data_list[1])   → 당연 제출을 위한 루트 노드 저장
    queue = collections.deque([root])

    index = 2   → 자식의 index를 처리하기 위해 현재 계산 중인 노드의 index 모두 뒤로 설정

    while queue :
        node = queue.popleft()   → 자식 노드 append → " 를 기준으로 자식노드 append ...

        if data_list[index] == '#' :   → '#'인 경우 어떠한 처리 X
            pass
        else :
            node.left = TreeNode(data_list[index])
            queue.append(node.left)
        index += 1

        if data_list[index] == '#' :
            pass
        else :
            node.right = TreeNode(data_list[index])
            queue.append(node.right)
        index += 1

    return root
```

왼쪽 자식

오른쪽 자식

④ 비고,,

· None로 처리할 경우 "노드 = None" 을 굳이 할 필요x
아무 시행을 하지 않아도 None로 간주

```
if value == "#" : node.left = None
```

→

```
if data_list[index] == '#'
    pass
```