알고리즘 Quiz 1

* 충남대학교 컴퓨터공학과
* 분반: 알고리즘 04분반
* 학번: 201701975
* 이름: 구건모

>> 문제 1번

<Array>

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	9	6	1	5	11	11	$\frac{1}{7}$	4	3	6	2	$\frac{1}{6}$	6

문제의 요구사항에 따라 배열의 index 는 각 트리 노드의 번호이며 root 는 트리의 노드 개수의 역수 형태로 저장하도록 하였습니다.

root 가 6 인 Tree 는 노드 개수가 7 개 이므로, Index 6 의 값은 노드 개수의 역수인 $\frac{1}{7}$ 이 들어가게 됩니다. root 가 11 인 Tree 는 노드 개수가 6 개 이므로 Index 11 번의 값은 노드 개수의 역수인 $\frac{1}{6}$ 이 들어가게 됩니다.

또한 root 가 아닌 노드들은 자신의 값을 각각 자신의 parent 노드 번호를 배열값으로 가지게 하므로써 Tree 의 구조를 배열에 담을 수 있습니다. 예를 들면 0 번 노드는 자신의 Parent 가 9 번 노드이므로 배열 Index 0 번의 값은 9 가 됩니다. 9 번 노드의 경우 자신의 parent 가 6 번 노드이므로 배열의 Index 9 번 값은 6 이 할당되는등의 방식으로 모든 Tree 의 정보를 Array 에 담을 수 있습니다.

#[1] 정답

<Array>

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	9	6	1	5	11	11	$\frac{1}{7}$	4	3	6	2	$\frac{1}{6}$	6

>> 문제 2번

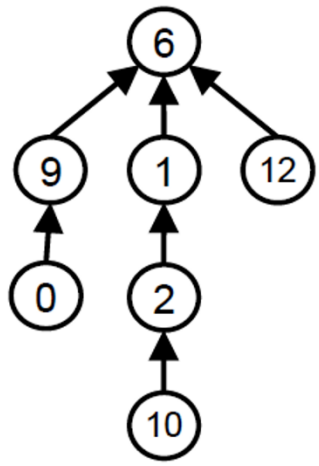
Union(Find(2),Find(8)) 부터 살펴보면, 우선 Union 이 수행되기 전에 2 번 Node 와 8 번 Node 에 대해서 각각 Find 가 수행되고 있음을 확인할 수 있는데 해당 문제에서 collapsing Rule 을 적용하여 Find 를 수행한다고 가정하였으므로 Find 과정을 수행할 때 Tree 의 변경을 가져오게 됩니다.

따라서 Union(Find(2),Find(8))에서 Union 내부의 Find(2), Find(8)이 각각 수행됨에 따른 변화를 살펴보면, 다음과 같습니다.

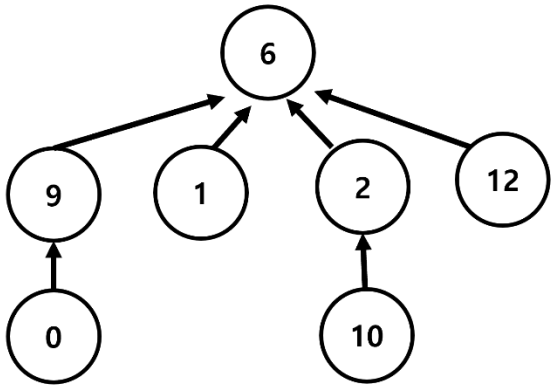
<Find(2)를 수행하였을 때의 변화>

Find(2) 를 수행한 결과

수행 전 Tree



Find(2) 수행이후 Collasping된 Tree



<변경된 배열값>

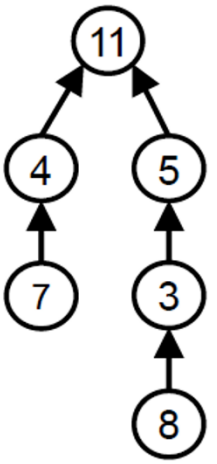
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	9	6	6	5	11	11	$\frac{1}{7}$	4	3	6	2	$\frac{1}{6}$	6

Find(2)를 수행하는 과정에서 노드 2 와 노드 1 을 거치므로 두 노드들의 parent 가 6 번 노드로 변경되게 됩니다. 사실상 1 번 노드는 원래 parent 가 6 번 노드이므로 변경사항은 없습니다.

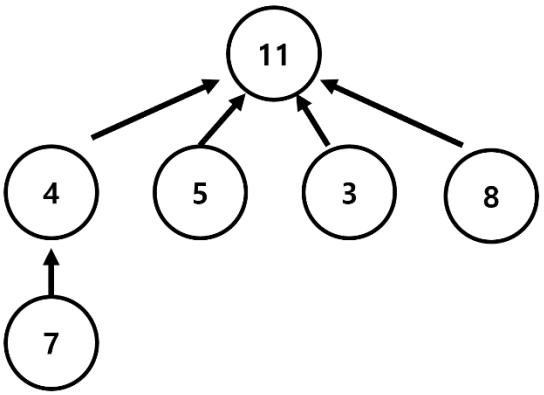
<Find(2) 수행 이후, Find(8)을 수행하였을 때의 변화>

Find(8) 를 수행한 결과

수행 전 Tree



Find(8) 수행이후 Collasping된 Tree



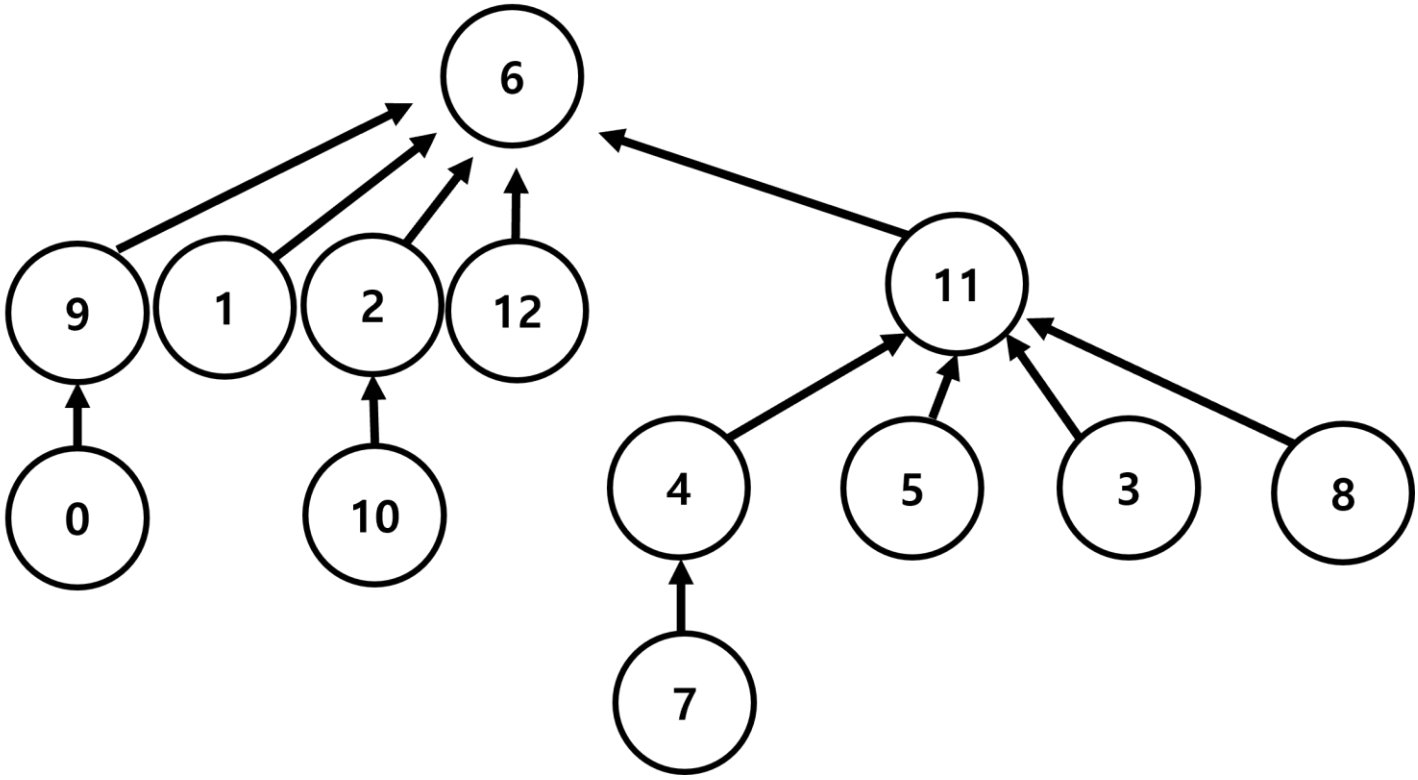
<변경된 배열값>

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	9	6	6	11	11	11	$\frac{1}{7}$	4	11	6	2	$\frac{1}{6}$	6

<Union 수행 하였을 때의 변화>

Find(2) = 6, Find(8) = 11 이고 앞의 과정과 같이 Find가 수행되었을 때, 앞의 Find 과정에서 변경된 Tree 상태에서 Union이 진행되게 되는데 Weighting Rule을 적용하여 Union 하므로 트리의 원소 개수가 더 많은 트리의 밑으로 상대적으로 적은 원소를 가지고 있는 Tree가 Union 되므로, 상대적으로 원소수가 많은, 6번 노드를 root로 하는 tree의 밑으로 11번 노드를 root로 하는 tree가 union 됩니다. 따라서 11번 노드의 Parent가 6번 노드가 되므로 배열의 index 11 값이 6으로 변경됩니다. 트리가 밑으로 들어가면서 6번 노드를 root로 하는 Tree의 전체 원소수도 달라지게 되므로 배열의 값이 변경됩니다.

Union(Find(2),Find(8))로 변경된 Tree



Union(Find(2),Find(8))로 변경된 Array

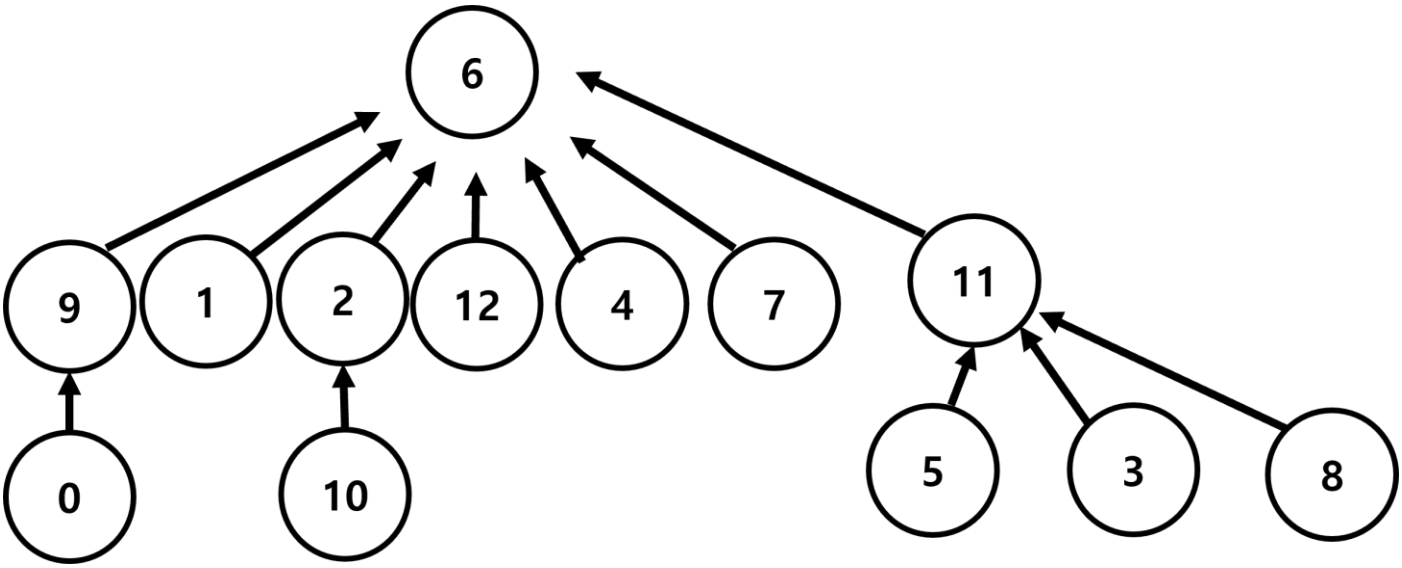
<변경된 배열값>

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	9	6	6	11	11	11	$\frac{1}{13}$	4	11	6	2	**6**	6

11번 노드를 루트로 하는 트리가 6번 노드를 루트로 하는 트리에 union 되면서 6번 노드를 루트로 하는 트리의 전체 원소수가 변경되었고 11번 노드의 parent 가 6번으로 변경되게 됩니다.

<Union 수행한 상태에서, Find(7) 수행 시 결과>

- Find(7) 수행 이후 Tree



Find(7) 수행 시 7번 노드부터 루트가 나올 때까지 Parent를 타고 올라가는데 올라가는 Collapsing Rule을 적용하였으므로 Find(7)을 수행하는 동안 거친 모든 노드들, 즉 7번과 4번, 11번 노드가 root 노드인 6번 노드를 parent로 삼게 됩니다. 따라서 이에 따라 배열 값도 변경됩니다.

<Find(7)로 변경된 Array>

<변경된 배열값>

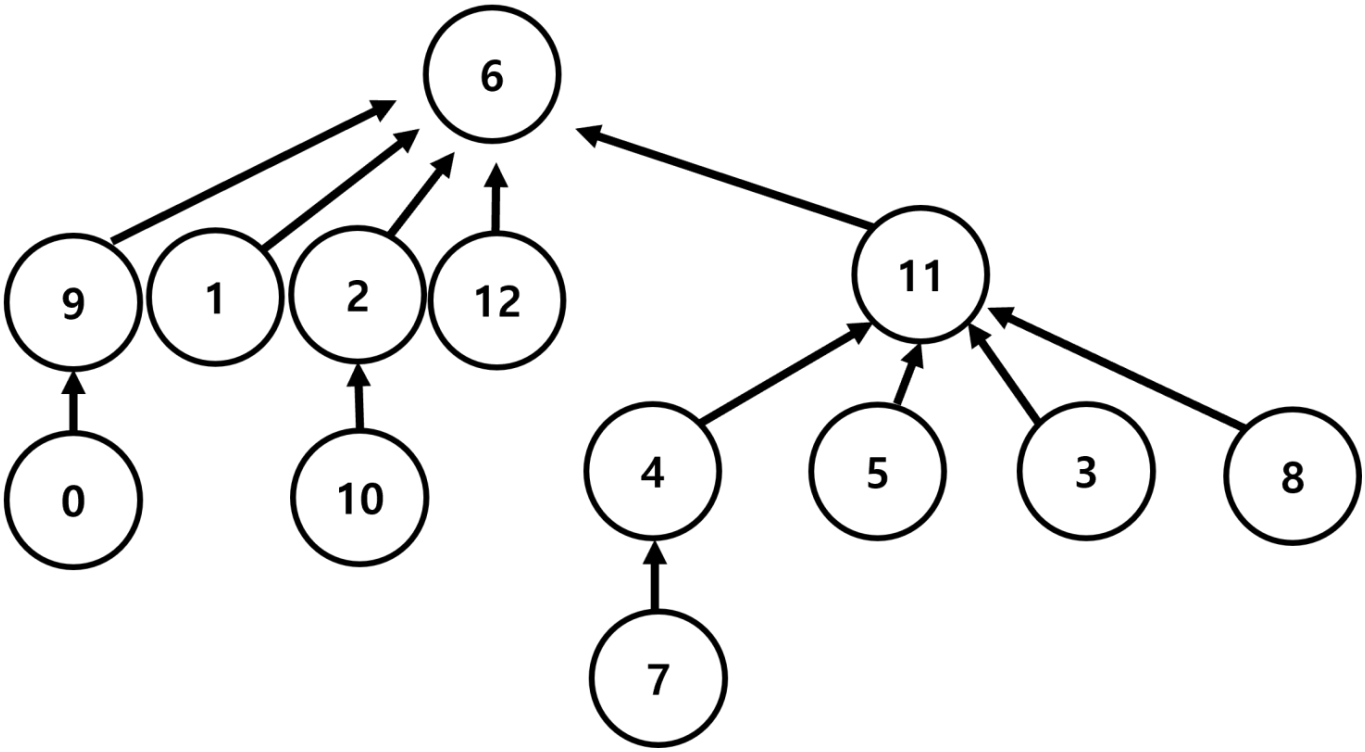
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	9	6	6	11	6	11	$\frac{1}{13}$	6	11	6	2	6	6

11번의 parent는 원래 6번 노드이기 때문에 변경이 없고 4번 7번에 노드에 대하여 parent가 6으로 바뀌게 됩니다.

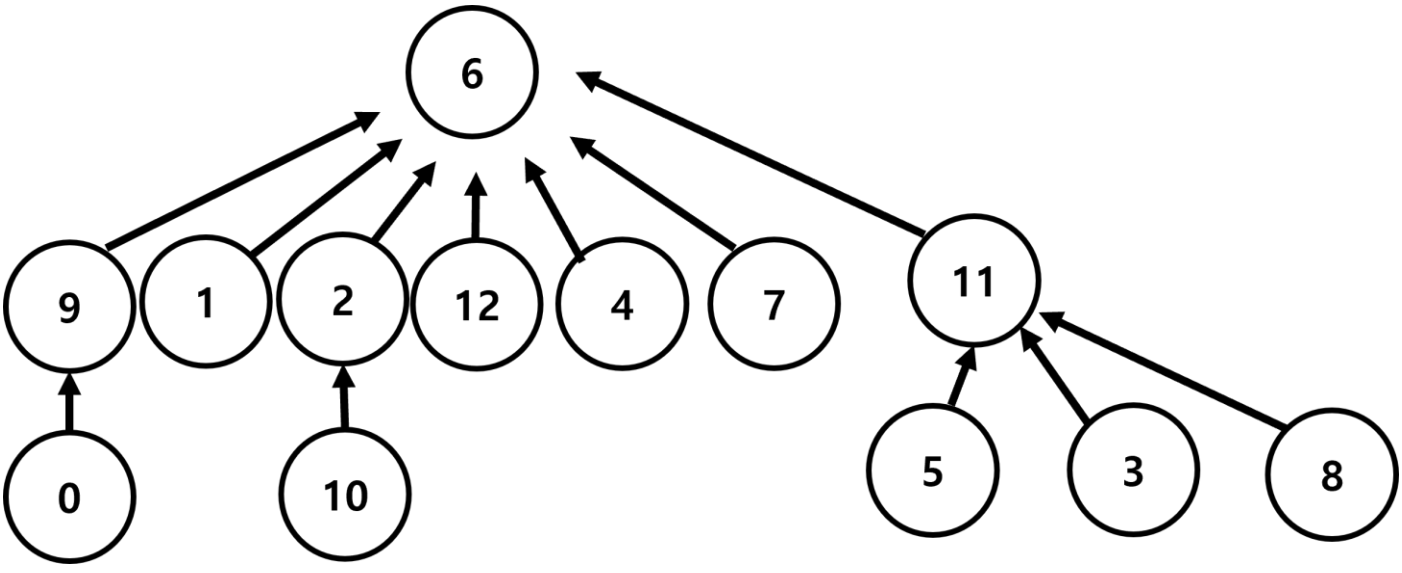
#[2] 정답

Union(Find(2),Find(8)) -> Find(7) 수행시 Tree 변화 과정

Union(Find(2),Find(8))로 변경된 Tree



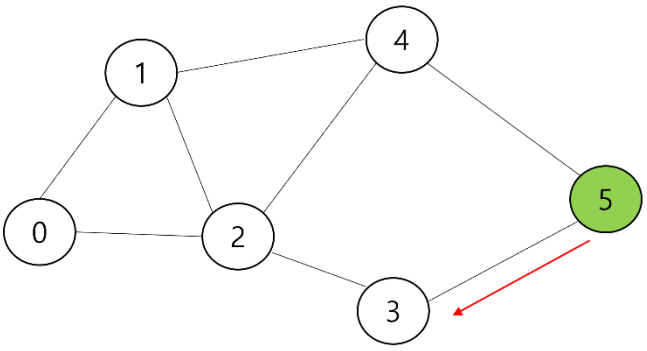
- Find(7) 수행 후 Tree



>> 문제 3번

#[3] -과정1

current vertex = V_5



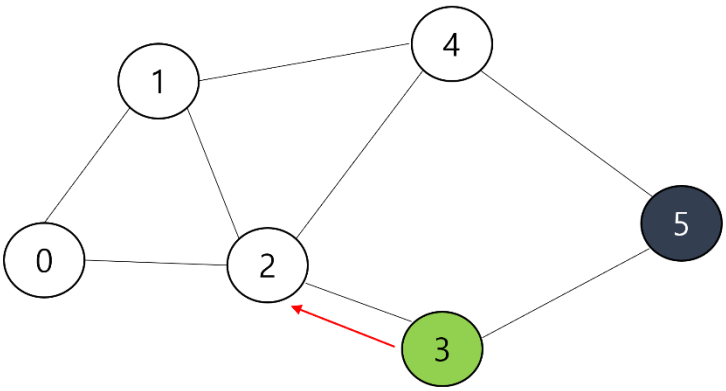
> Current: 5

> Visited: 5

V_5 에서부터 DFS 시작.
V_5 에서 시작하므로 V_5 Visited 처리
V_5 의 인접 vertices 중에서 번호가 작은 vertex를 방문한다고 문제에서 정의되었으므로 V_3 선택, V_3이 Visited 상태가 아니므로 $V_5 \rightarrow V_3$ 로 이동.

#[3] -과정2

current vertex = V_3



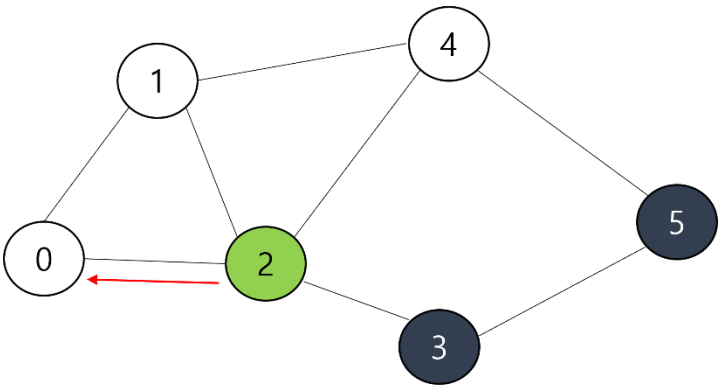
> Current: 3

> Visited: 5 3

V_3로 이동 하였으므로 V_3 Visited 처리
V_3 의 인접 vertices 중에서,
번호가 작은 vertex인 V_2 선택
V_2이 Visited 상태가 아니므로 V_2으로 이동

#[3] -과정3

current vertex = V_2



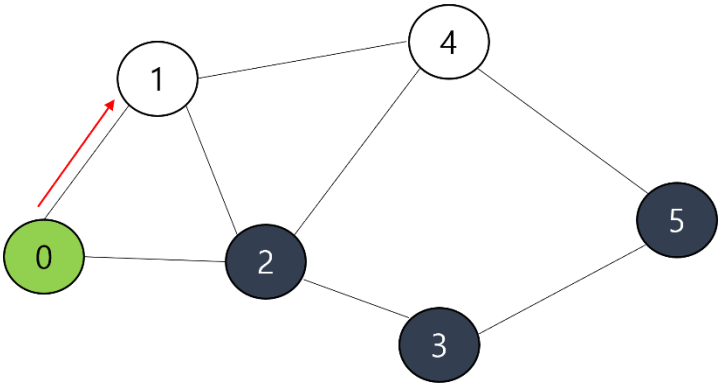
> Current: 2

> Visited: 5 3 2

V_2로 이동 하였으므로 V_2 를 Visited 처리
V_2 의 인접 vertices 중에서,
번호가 작은 vertex인 V_0 선택
V_0이 Visited 상태가 아니므로 V_0으로 이동

#[3] -과정4

current vertex = V_0



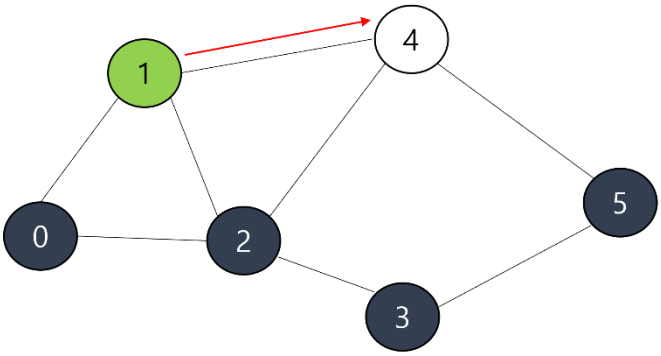
> Current: 0

> Visited: 5 3 2 0

V_0로 이동 하였으므로 V_0 를 Visited 처리
V_0 의 인접 vertices 중에서,
번호가 작은 vertex인 V_1 선택
V_1이 Visited 상태가 아니므로 V_1으로 이동

#[3] -과정5

current vertex = V_1



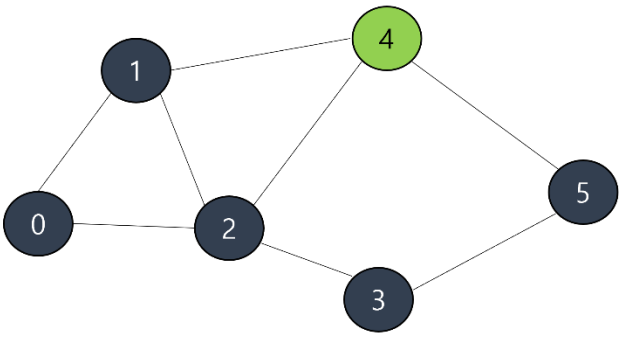
> Current: 1

> Visited: 5 3 2 0 1

V_1로 이동 하였으므로 V_1 를 Visited 처리
V_1 의 인접 vertices 중에서,
번호가 작은 vertex인 V_0 은 visited 상태이고,
그 다음으로 작은 vertex인 V_2도 Visited 상태이므로
V_4 선택.
V_4 가 Visited 상태가 아니므로 V_4로 이동

#[3] -과정6

current vertex = V_4



> Current: 4

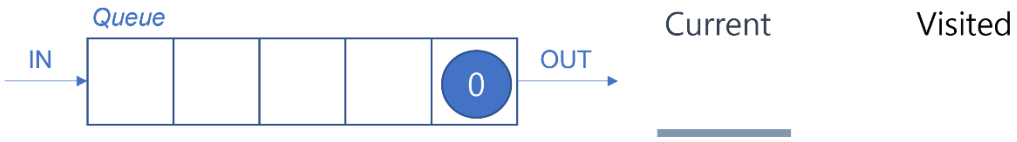
> Visited: 5 3 2 0 1 4

따라서 DFS에 따른 방문순서는 5 → 3 → 1 → 0 → 1 → 4.

#[3] 정답 5 → 3 → 1 → 0 → 1 → 4

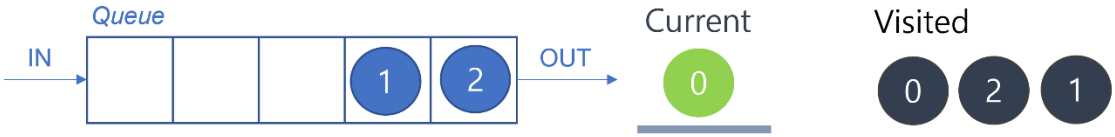
>> 문제 4번

#[4] -과정1



V_0 번 부터 BFS를 실시하므로 시작 vertex인 0번 vertex(V_0)를 Queue에 넣어줌.

#[4] -과정2



Queue가 empty가 아니므로 V_0을 DeQueue하고 방문 처리함.
V_0의 인접 vertex들을 weight가 작은 순서대로(V_2, V_1) InQueue 하고 Visited 처리함.

#[4] -과정3

Queue가 empty가 아니므로 하나(V_2) 를 DeQueue 하고 visited 처리 이제 V_2 가 current
V_2 의 인접 Vertex들을 weight가 작은 순서에 따라 InQueue, V_4, V_3 순서대로 InQueue 하고 Visited 처리함.
V_0, V_1은 이미 Visited 상태이므로 InQueue 하지 않음

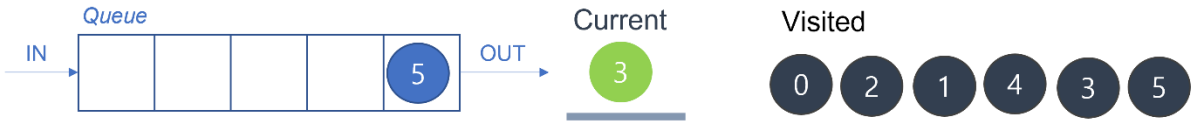
#[4] -과정4

Queue가 empty가 아니므로 하나(V_1) 를 DeQueue 하고 visited 처리 이제 (V_1)이 current
V_1 의 인접 vertex 들은 전부 Visited 상태이므로 어떤 vertex도 InQueue 되지 않음

#[4] -과정5

Queue가 empty가 아니므로 하나(V_4) 를 DeQueue 하고 visited 처리 이제 V_4 가 current
V_4 의 인접 vertex들을 weight가 작은 순서대로 InQueue, V_5가 InQueue 되고 Visited 처리됨
V_1 V_2 V_3 은 이미 Visited 상태이므로 InQueue 하지 않음

#[4] -과정6



Queue가 empty가 아니므로 하나(V_3) 를 DeQueue 하고 visited 처리 이제 V_3가 current
V_3 의 인접 vertex들이 모두 Visited 상태이므로 InQueue하지 않음

#[4] -과정7

Queue가 empty가 아니므로 하나(V_5) 를 DeQueue 하고 visited 처리 이제 V_5가 current
V_5 의 인접 vertex들이 모두 Visited 상태이므로 InQueue하지 않음

#[4] -과정8

Queue가 empty 이므로 BFS가 종료됨.

결과로 도출된 방문 순서: 0 → 2 → 1 → 4 → 3 → 5

#[4] 정답 0 → 2 → 1 → 4 → 3 → 5