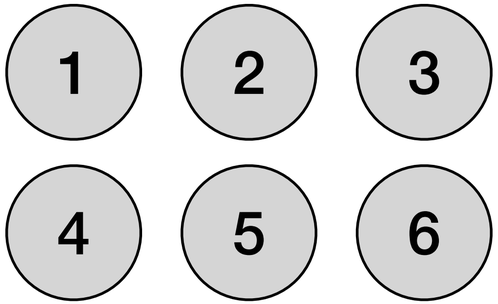
유니온 파인드

23.09.25 박송이

08-2 유니온 파인드

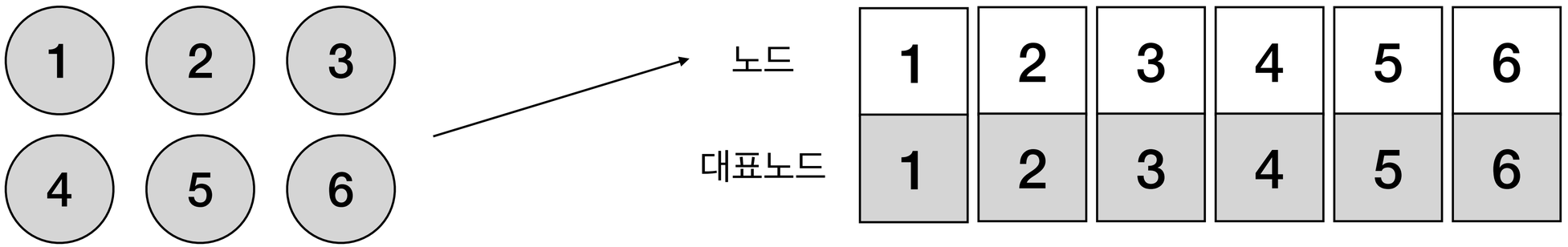
핵심 이론

- Union : 여러 노드가 있을 때 **특정 2개의 노드**를 연결해 1개의 집합으로 묶는 연산
- Find : 두 노드가 같은 집합에 속해 있는지를 확인하는 연산, 집합의 대표 노드를 찾는 연산



08-2 유니온 파인드

알고리즘 이해하기 - 유니온 파인드 표현



Union 연산 : 두개의 노드를 합치면서 대표노드의 값을 변경해주는 연산

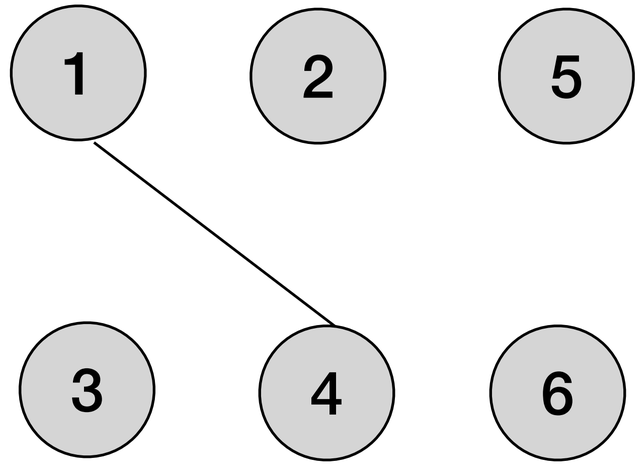
Find 연산 : union연산을 통해 바뀐 대표노드를 타고 가면서 자신의 대표노드값을 찾는 연산

08-2 유니온 파인드

알고리즘 이해하기 - 유니온 연산 수행

Union(1,**4**)

대표노드 자식노드

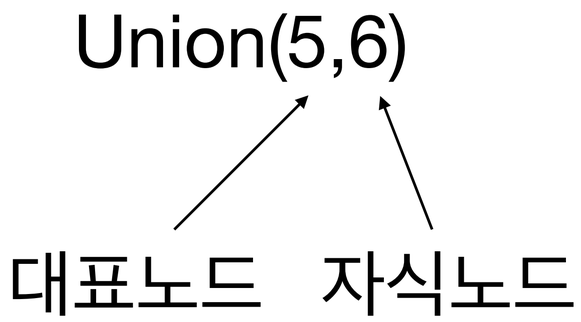


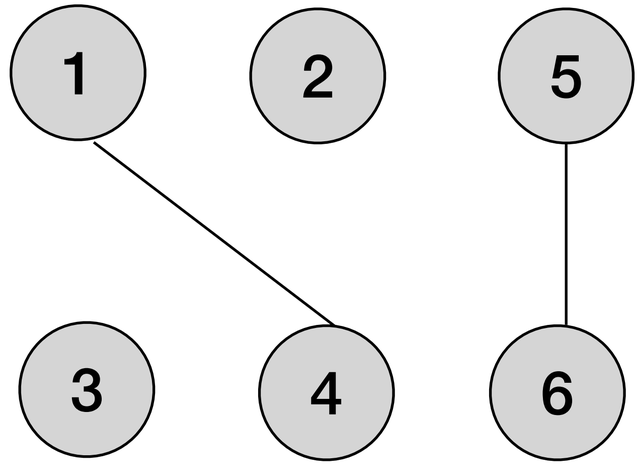
노드	1	2	3	4	5	6
대표노드	1	2	3	4	5	6

노드	1	2	3	4	5	6
대표노드	1	2	3	1	5	6

08-2 유니온 파인드

알고리즘 이해하기 - 유니온 연산 수행





노드	1	2	3	4	5	6
대표노드	1	2	3	1	5	5

08-2 유니온 파인드

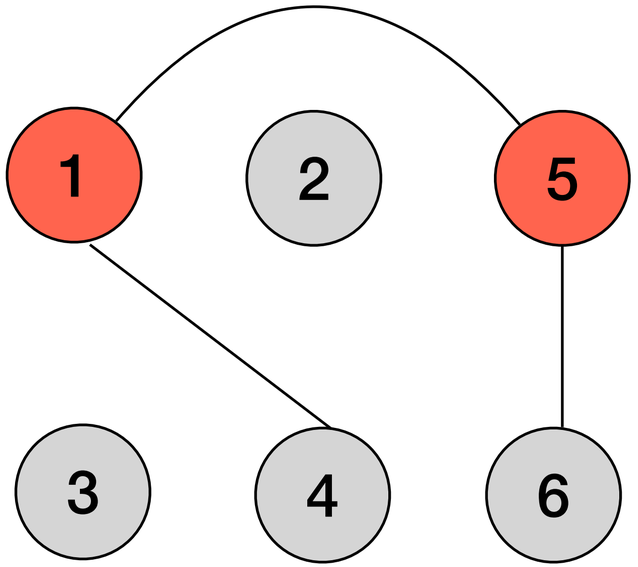
알고리즘 이해하기 - 유니온 연산 수행

Union(4,6)

자식노드 자식노드

둘 다 자식노드일 경우
대표노드를 찾아
두개를 연결해준다.

Union(1,5)



노드	1	2	3	4	5	6
대표노드	1	2	3	1	5->1	5

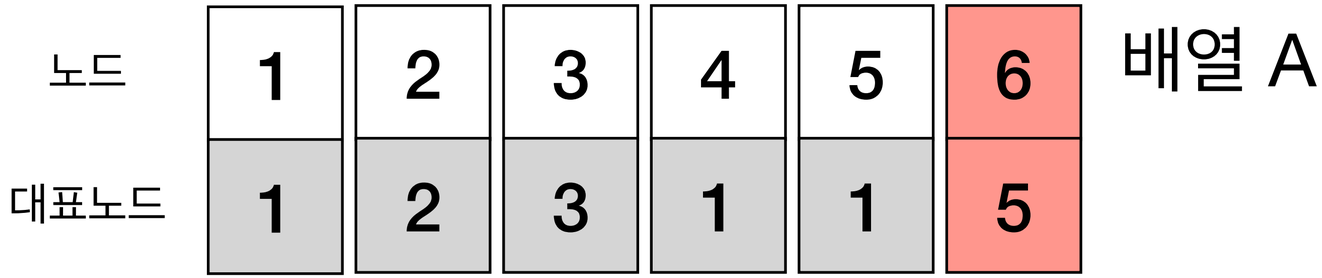
08-2 유니온 파인드

알고리즘 이해하기 - find 연산 수행

1. **자신의 속한 집합의 대표 노드를 찾는 연산**
2. **그래프를 정돈하고 시간 복잡도를 줄여준다.**

Find(6)

index



배열 A

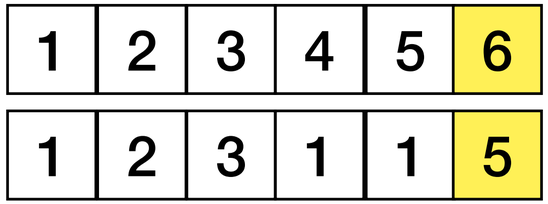
노드와 대표노드의 값 비교하기

같으면 -> 값 리턴

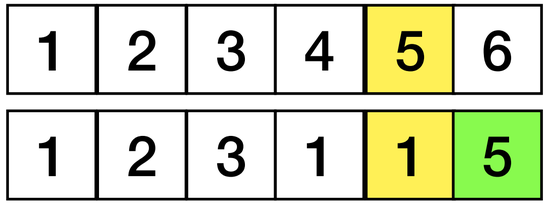
다르면 -> 대표노드값이 가리키는 노드로 이동 후 다시 비교

08-2 유니온 파인드

알고리즘 이해하기 - find 연산 수행

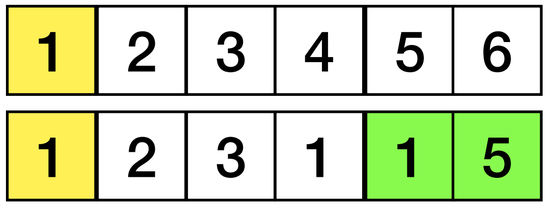


노드와 대표노드의 값이 다르면 이동

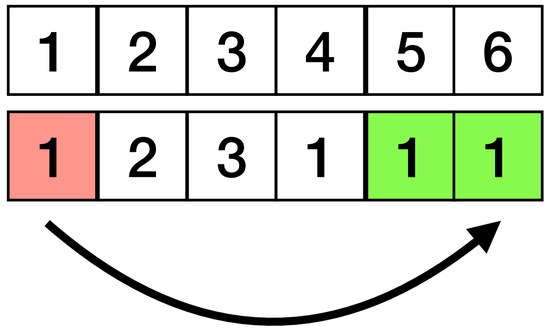


노드와 대표노드의 값이 다르면 이동

재귀함수를 거치면서 그동안 거친
노드의 value값을 1번
노드의 value값으로 모두 변경

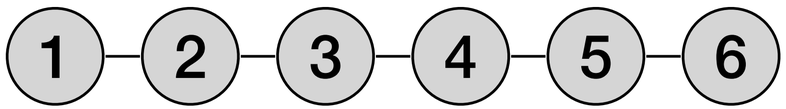


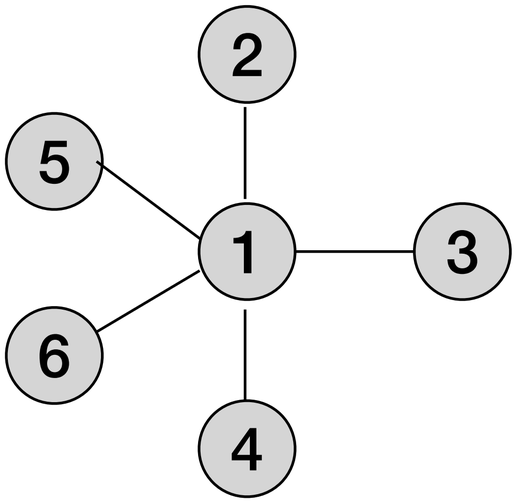
노드와 대표노드의 값이 같다면 ->



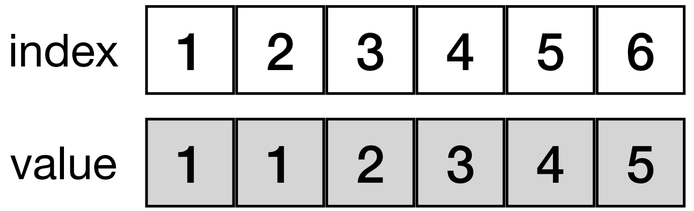
08-2 유니온 파인드

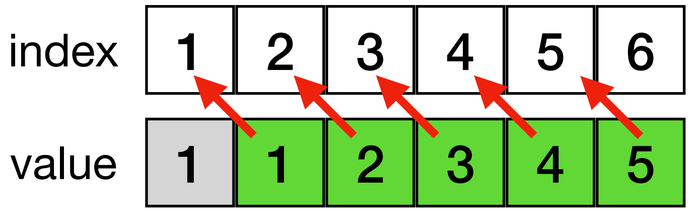
알고리즘 이해하기 - 어떻게 해서 시간복잡도를 줄여주는가?

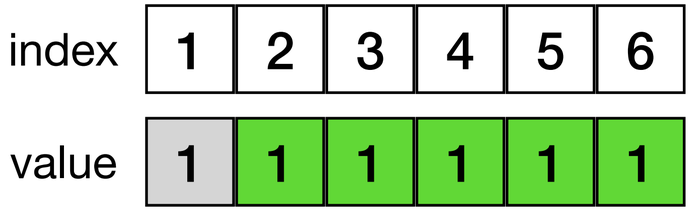




find(6)







한번의 find(6) 연산으로 경로 압축의 효과를 볼 수 있다.

문제 50 - 집합 표현하기

입력	출력
7 8 0 1 3 1 1 7 0 7 6 1 7 1 0 3 7 0 4 2 0 1 1 1 1 1	NO YES YES

n <= 1,000,000
m <= 100,000

합집합 연산 - 0 a b 형태로 입력이 주어진다.

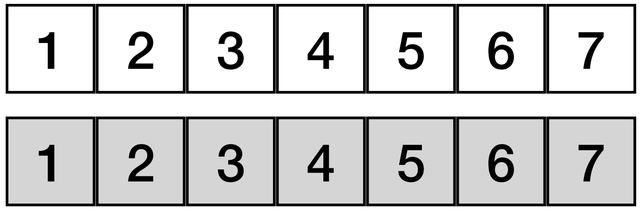
두 원소가 같은 집합에 포함되어 있는가? - 1 a b 형태

1로 시작하는 입력에 1줄에 1개씩 YES, NO를 표시한다.

최대 원소의 개수가 1,000,000
질의 개수가 100,000이므로 경로단축이 필요한 전형적인 유니온 파인드 문제이다.

문제 50 - 집합 표현하기

처음에는 노드가 연결돼 있지 않으므로
각 노드의 대표노드는 자기 자신이 된다.
각 노드의 값을 자기 인덱스값으로 초기화한다.



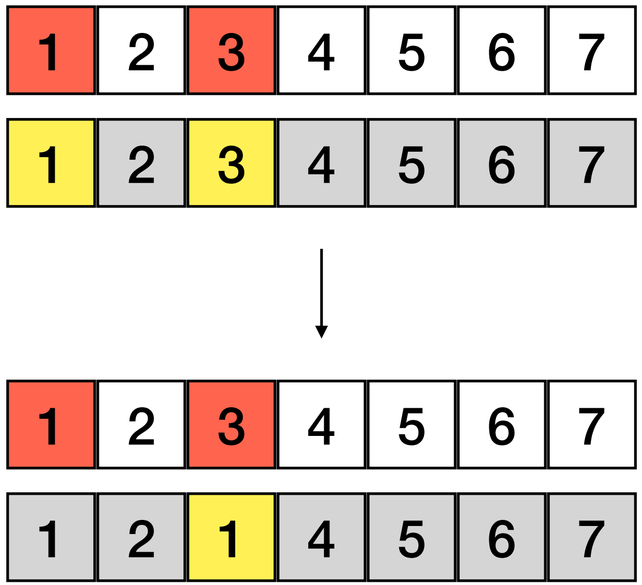

```
parent = [0] * (N+1)

for i in range(0,N+1):

    parent[i] = i
```

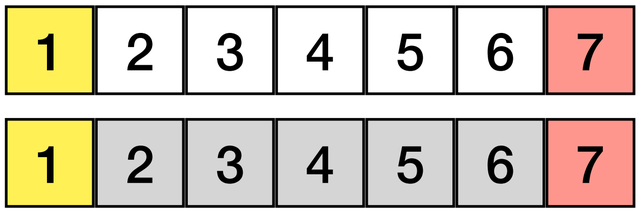
UNION 연산

0,1,3 -> union(1,3)



FIND 연산

1,1,7 -> find(1) == find(7)



find(1)== 1

!=

find(7)==7

-> NO 출력

문제 50 - 집합 표현하기

```python
def union(a,b):
    a = find(a)
    b = find(b)
    if a !=b:
        parent[b] = a
```

```python
def find(a):
    if a == parent[a]:
        return a
    else:
        parent[a] = find(parent[a])
        return parent[a]
```

1. find로 각 노드의 대표 노드를 찾는다.
2. 그리고 두 노드의 각 대표노드가 같지 않으면 b노드의 대표노드를 a의 대표노드로 해준다.

a노드값 자체와 a의 대표노드값이 같은지 비교하기 -> 같으면 a 자체를 반환하기

같지 않다면 재귀적으로 parent[a]를 넣어 다시 호출하기

```python
def checkSame(a,b):
    a = find(a)
    b = find(b)
    if a==b:
        return True
    else:
        return False
```

checkSame함수에서 인자 a,b를 받아
이 안에서 find함수를 호출해 같은지 비교하기

문제 51 - 여행 계획 짜기

입력	출력
3 #N 3 #M 0 1 0 1 0 1 0 1 0 1 2 3 #여행계획	YES

n <= 200
m <= 1000

도시 개수 N개 , 여행경로 데이터 M
i번째 줄의 j번째 수는 i번 도시와 j번 도시의 연결 정보
1 : 연결 됐음
0 : 연결 없음.
A와 B가 연결됐으면 B와 A도 연결됐음.

주어진 여행계획으로 여행이 가능하면
YES, 아니면 NO 출력

문제 51 - 여행 계획 짜기

```
3 #N
3 #M
0 1 0
1 0 1
0 1 0
1 2 3 #여행계획
```

입력의 형태가 **인접행렬**로 주어졌기 때문에
인접 행렬을 탐색하면서 연결될때마다 union 연산을 수행하는 방식으로 접근

parent

1	2	3
1	2	3

```
parent = [0] * (N+1)
```

dosi

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

```
dosi = [[0] for j
in range(N+1)] for i
in range(N+1)]
```

route

1	2	3

```
route = list(map(int,input().split()))
route.insert(0,0)
```

문제 51 - 여행 계획 짜기

dosi

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

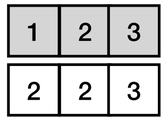
인접행렬을 돌면서
값이 1일때만 union연산수행

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

union(1,2)

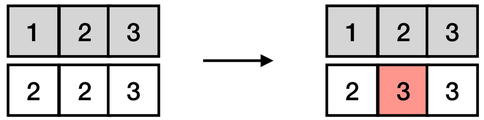
큰 도시가 대표가 되도록 한다.

union(2,1)



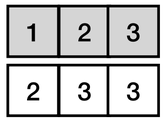
대표노드가 같아 연산 불필요

union(2,3)



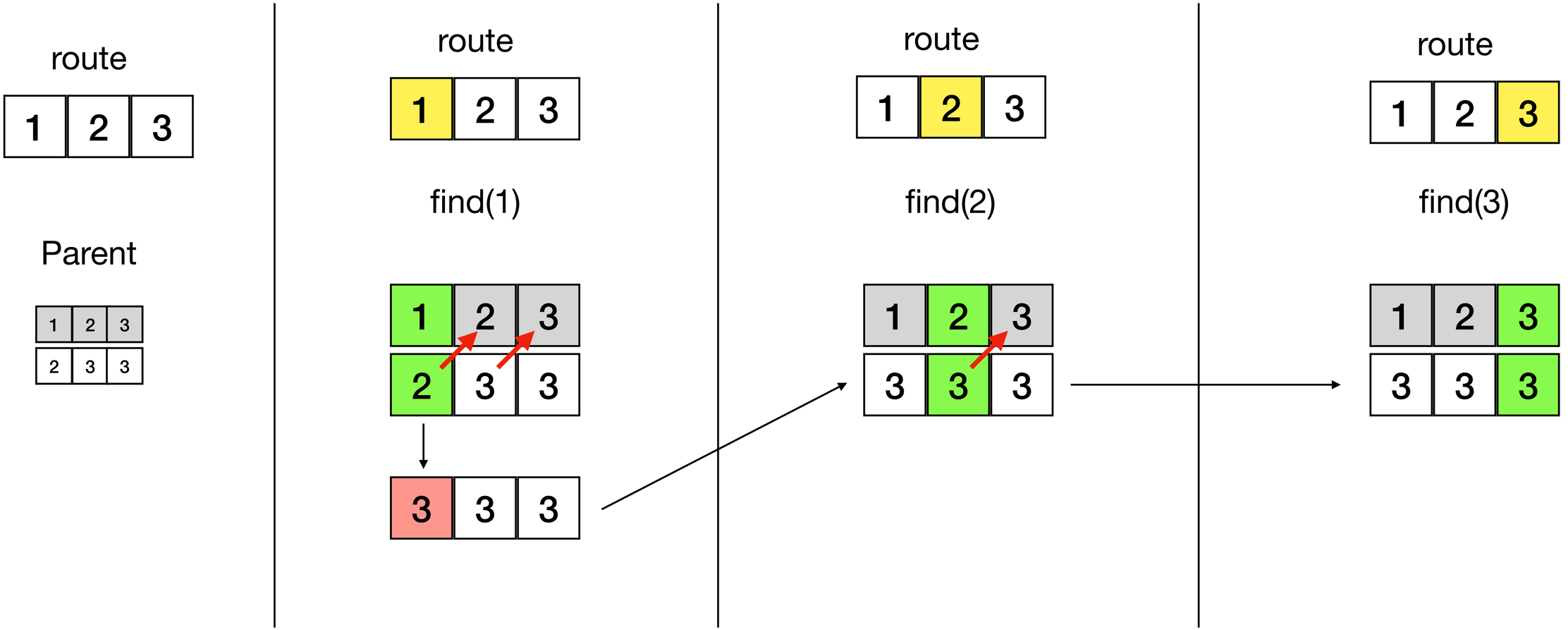
큰 도시가 대표가 되도록 한다.

union(3,2)



대표노드가 같아 연산 불필요

문제 51 - 여행 계획 짜기



여행경로에 있는 모든 도시의 대표 도시가 3으로 같으므로 YES 출력

문제 52 - 거짓말쟁이가 되긴 싫어. - 문제

입력	출력
4 3 #사람수, #파티 수 0 #진실을 아는 사람의 정보 2 1 2 #파티 정보 1 3 3 2 3 4 #2,3,4 3명 참여	3

n,m <= 50
0<= 진실을 아는 사람 수, 각 파티별 오는 사람의 수 <=50

지민이가 거짓말쟁이로 알려지지 않으면서
과장된 이야기를 할 수 있는 파티의 최댓값을 구하기

문제 52 - 거짓말쟁이가 되긴 싫어. - 솔루션

각각의 파티마다 union연산을 이용해서 대표 노드를 같게 만들어주기

각 파티의 대표 노드와 진실을 알고 있는 사람들의 각 대표 노드가 동일한지 find연산을 통해 계산

만약 같지 않다면 해당 파티에서는 거짓말을 할 수 있다. -> count 해주기

문제 52 - 거짓말쟁이가 되긴 싫어.

초기화

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

8 5

3 1 2 7 -> 진실을 아는 사람

2 3 4

1 5

2 5 6

2 6 8

1 8

2 3 4 => union(3,4)

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

1	2	3	4	5	6	7	8
1	2	3	3	5	6	7	8

2 5 6 => union(5,6)

1	2	3	4	5	6	7	8
1	2	3	3	5	6	7	8

1	2	3	4	5	6	7	8
1	2	3	3	5	5	7	8

2 6 8 => union(6,8)

1	2	3	4	5	6	7	8
1	2	3	4	5	5	7	8

1	2	3	4	5	6	7	8
1	2	3	3	5	5	7	5

문제 52 - 거짓말쟁이가 되긴 싫어.

각 파티별 대표 노드

1	2	3	4	5	6	7	8
1	2	3	3	5	6	7	5

진실을 아는 사람들의 대표 노드 -> 1,2,7

find(1) -> **1**
find(2) -> **2**
find(3) -> **7**

입력

8 5
3 1 2 7
2 3 4
1 5
2 5 6
2 6 8
1 8

1번 파티 (3,4) -> find(3) -> 3
2번 파티 (5) -> find(5) -> 5
3번 파티 (5,6) -> find(5) -> 5
4번 파티 (6,8) -> find(6) -> 6
5번 파티 (8) -> find(8) -> 5

과장할 수 있음
과장할 수 있음
과장할 수 있음
과장할 수 있음
과장할 수 있음

출력

5

문제 52 - 거짓말쟁이가 되긴 싫어. - 코드

파티에 참여한 사람들을 하나의 그룹으로 만들어주기

```
for i in range(M):
    firstPeople = party[i][0]
    for j in range(1,len(party[i])):
        union(firstPeople,party[i][j])
```

예를 들어,
파티 정보가
4 1 2 3 4로 주어진다면
파티에 참여한 사람은 4명이 되고
각 1,2,3,4를 union해야 한다.
그렇기에 여기서는 제일 앞에 있는
1번 사람을 firstPeople이라고 지정하고
firstPeople과 2,3,4를 각각 union해준다.

각 파티의 대표 노드와 진실을 아는 사람들을 비교하기

```
for i in range(M):
    isPossible = True
    firstPeople = party[i][0]
    for j in range(len(trueP)):
        if find(firstPeople) == find(trueP[j]):
            isPossible = False
            break
    if isPossible:
        result +=1

print(result)
```

처음에는 isPossible를 True로 해두고
진실을 아는 사람들의 배열인 trueP만큼 루프를 돌면서
만약 firstPeople의 대표노드와 trueP의 대표노드가 같다면
isPossible을 False로 바꿔준다.