Uinon-Find

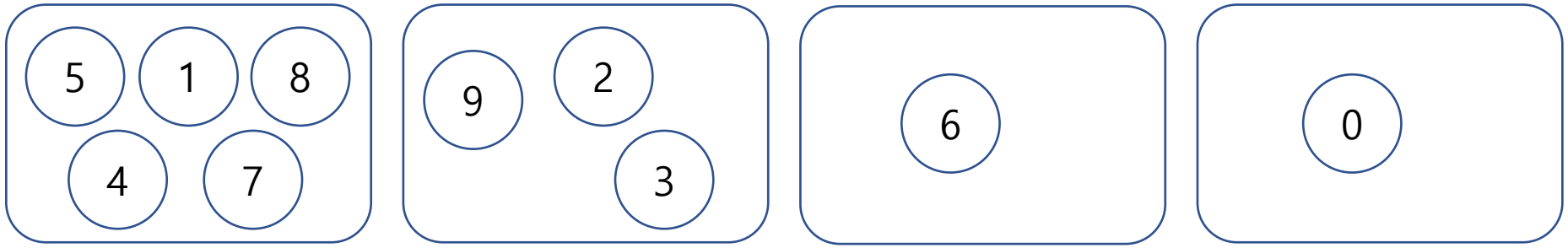
김 태 현

Union Find

Disjoint set을 표현하는 자료구조/알고리즘

Disjoint Set *서로소 집합 or 상호 배타적 집합*

: 공통 원소가 없는 부분집합



지원하는 연산

1) Init(N) : N개의 원소가 각각의 집합에 포함되도록 초기화
2) Union(a, b) : 두 원소 a,b가 속한 두 집합을 하나로 합침
3) Find(a) : a가 속한 집합 반환

표현 방법

1. sequence
2. tree

선형 자료구조로 표현 array, vector, list

1. Init(N)

- N개 원소 각각 자기 번호의 집합에 포함시킨다.
- O(N)

원소 별 집합번호

x	set id
1	1
2	2
3	3
4	4
5	5
6	6
7	7

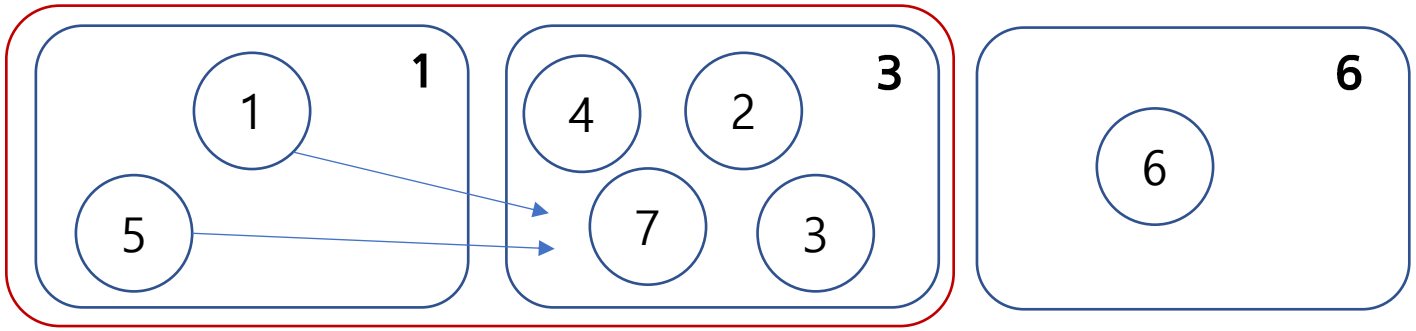
집합 별 원소 리스트

set id	x list
1	1
2	2
3	3
4	4
5	5
6	6
7	7

선형 자료구조로 표현 array, vector, list

2. Union(a,b)

- a,b 원소가 속한 집합을 합친다.
- 원소 개수가 적은 집합에서 많은 집합으로 이동
- N개의 원소가 전부 합쳐질 때 worst case : O(N log N)



union(5, 2)

3. Find(a)

- a 원소가 속한 집합 번호를 구한다.
- O(1)

원소 별 집합번호

x	set id
1	~~1~~ 3
2	3
3	3
4	3
5	~~1~~ 3
6	6
7	3

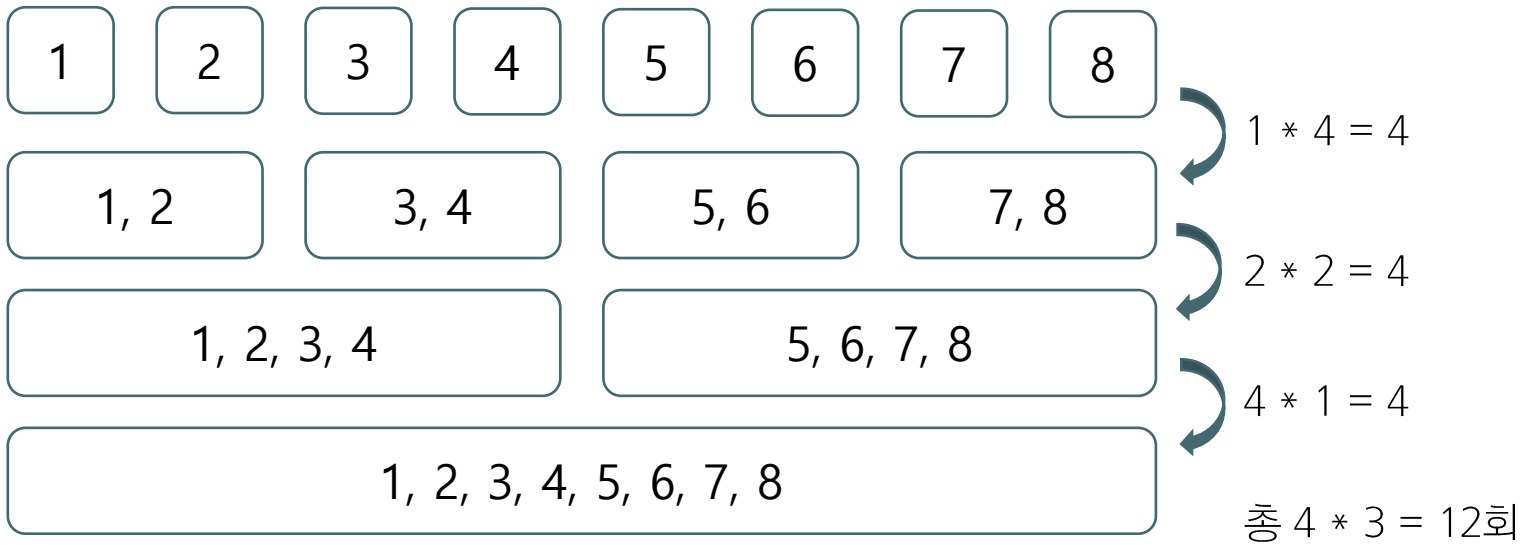
집합 별 원소 리스트

set id	x list
1	~~1, 5~~
2	
3	3, 2, 7, 4, 1, 5
4	
5	
6	6
7	

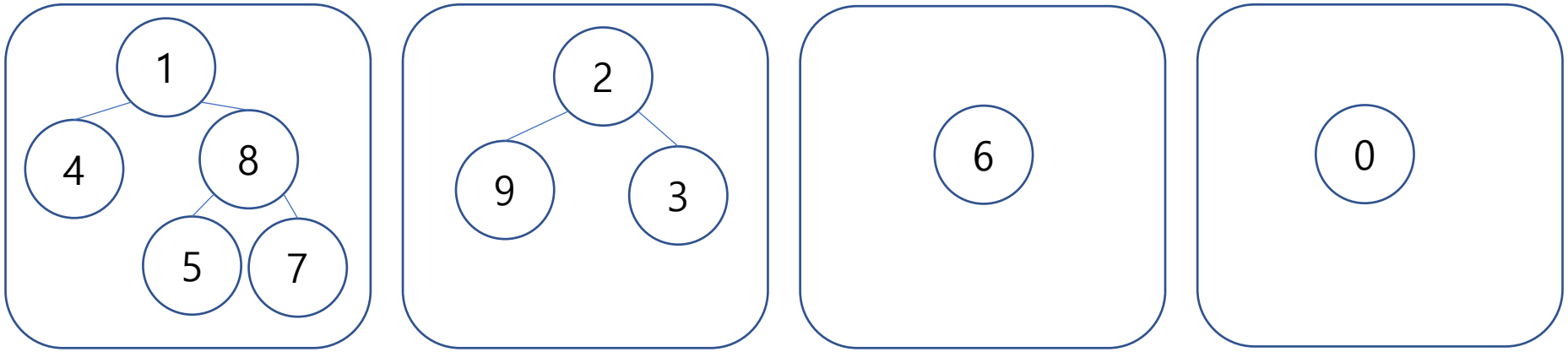
선형 자료구조로 표현 array, vector, list

Union(a,b)가 O(N log N) 인 이유

worst case 예시



트리로 표현



- 일반적으로 Union-Find 하면 트리로 표현한 방식을 의미한다.
- 각 집합은 트리로 구성되어 있어 루트 노드가 대표 노드이며, 집합의 정보는 루트 노드에 저장된다.
- 최적화 기법으로는 path compression(경로 압축)과 union-by-rank 두가지가 있다.
- 두가지 최적화를 모두 적용하면 O(*a*(N)) 으로 모든 연산이 상수시간에 수행된다.

  a(N) <= 4 : 아커만 함수

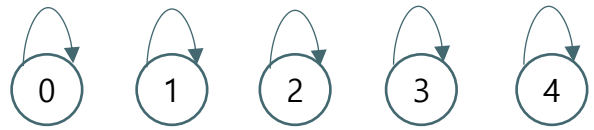
1. Init(N)

- 모든 노드를 *parent[x] = x, rank[x] = 0* 로 초기화
- parent[x]가 x인 경우, 트리의 루트이며 집합의 대표 노드가 된다.

```
const int LM = 100003;
int parent[LM], rank[LM];

void Init(int N) {
    for (int i = 0; i < N; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}
```

```
int Find(int x) {
    if (parent[x] == x) return x;
    return parent[x] = Find(parent[x]);
}

void Union(int a, int b) {
    a = Find(a), b = Find(b);
    if (a == b) return;
    if (rank[a] < rank[b]) swap(a, b);

    parent[b] = a;
    if (rank[a] == rank[b]) rank[a]++;
}
```

	0	1	2	3	4
parent	0	1	2	3	4
rank	0	0	0	0	0

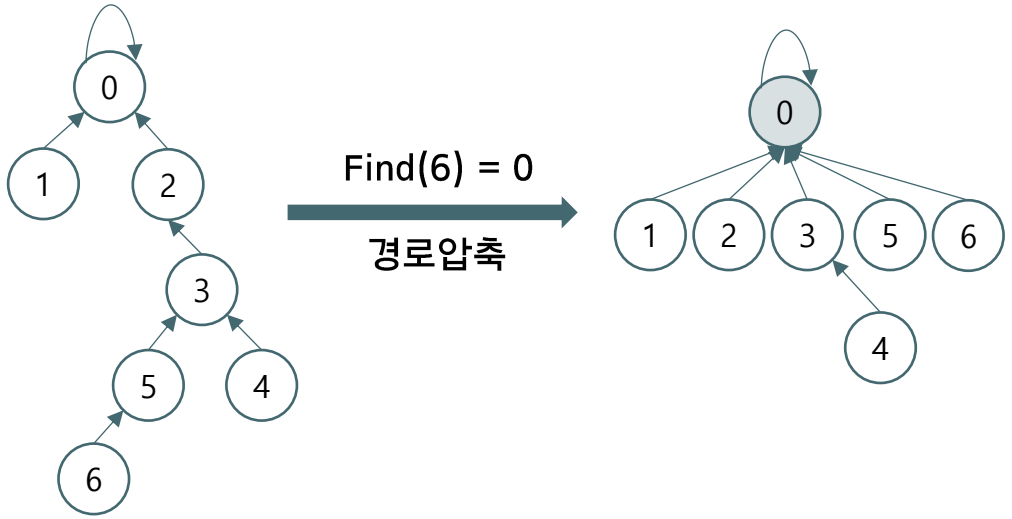


2. Find(x)

- x의 root 노드 반환
- 경로 압축: x와 root사이 모든 노드의 부모 노드를 root로 변경

```
const int LM = 100003;
int parent[LM], rank[LM];

void Init(int N) {
    for (int i = 0; i < N; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

int Find(int x) {
    if (parent[x] == x) return x;
    return parent[x] = Find(parent[x]);
}

void Union(int a, int b) {
    a = Find(a), b = Find(b);
    if (a == b) return;

    if (rank[a] < rank[b]) swap(a, b);

    parent[b] = a;
    if (rank[a] == rank[b]) rank[a]++;
}
```

	0	1	2	3	4	5	6
parent	0	0	0	~~2~~ 0	3	~~3~~ 0	~~5~~ 0
rank	4						

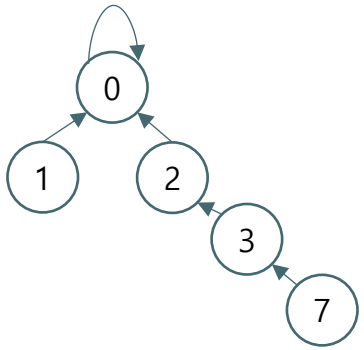


3. Union(a, b)

- a, b의 집합을 구하고 같은 집합에 속한 경우 return
- union by rank : rank는 높이를 나타내며, rank가 낮은쪽(b)에서 높은쪽(a)으로 트리를 연결(parent[b] = a)
  rank가 같았다면 1 증가
  경로압축을 통해 변경되는 높이는 rank에 반영되지 않는다.

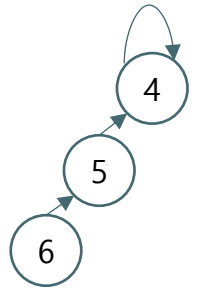
```
const int LM = 100003;
int parent[LM], rank[LM];

void Init(int N) {
    for (int i = 0; i < N; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

int Find(int x) {
    if (parent[x] == x) return x;
    return parent[x] = Find(parent[x]);
}

void Union(int a, int b) {
    a = Find(a), b = Find(b);
    if (a == b) return;

    if (rank[a] < rank[b]) swap(a, b);

    parent[b] = a;
    if (rank[a] == rank[b]) rank[a]++;
}
```

	0	1	2	3	4	5	6	7
parent	0	0	0	~~2~~ 0	4	4	~~5~~ 4	~~3~~ 0
rank	3				2			

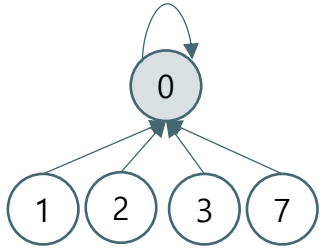


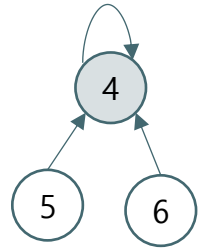
x = Find(7) = 0

Union(7, 6)



y = Find(6) = 4



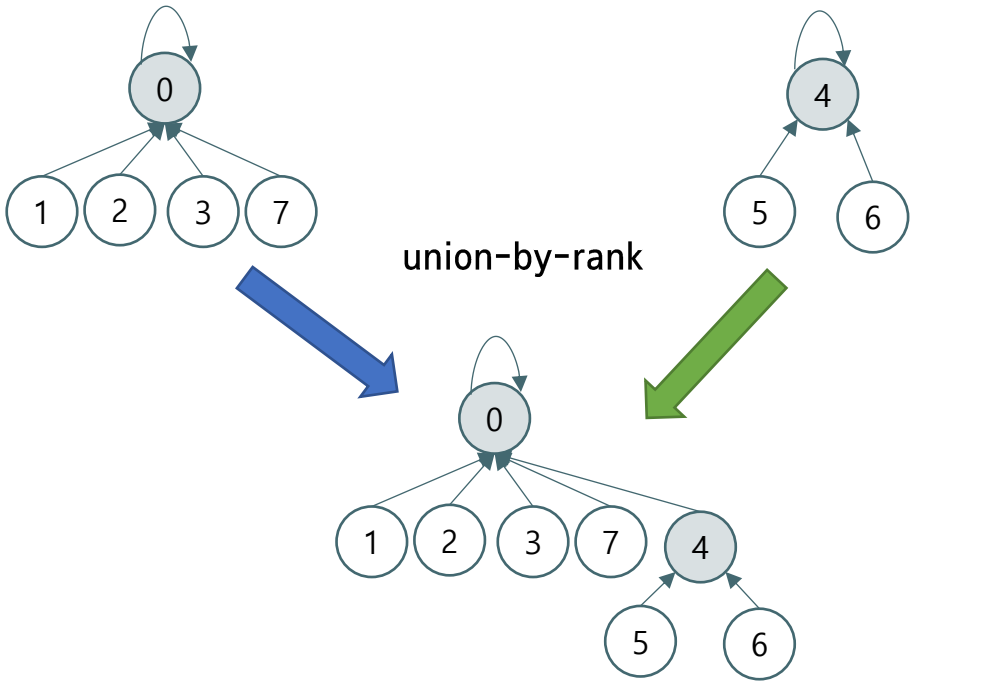


3. Union(a, b)

- a, b의 집합을 구하고 같은 집합에 속한 경우 return
- union by rank : rank는 높이를 나타내며, rank가 낮은쪽(b)에서 높은쪽(a)으로 트리를 연결(parent[b] = a)
  rank가 같았다면 1 증가
  경로압축을 통해 변경되는 높이는 rank에 반영되지 않는다.

```
const int LM = 100003;
int parent[LM], rank[LM];

void Init(int N) {
    for (int i = 0; i < N; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

int Find(int x) {
    if (parent[x] == x) return x;
    return parent[x] = Find(parent[x]);
}

void Union(int a, int b) {
    a = Find(a), b = Find(b);
    if (a == b) return;

    if (rank[a] < rank[b]) swap(a, b);

    parent[b] = a;
    if (rank[a] == rank[b]) rank[a]++;
}
```

	0	1	2	3	4	5	6	7
parent	0	0	0	0	~~4~~ 0	4	4	0
rank	3				~~2~~			




감사합니다