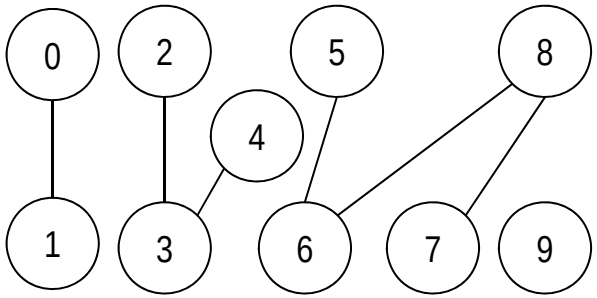
서로소집합과 union-find

서로소집합(disjoint set) 표현: 개념



5과 8이 연결되어 있는가?

{0,1}, {2,3,4}, {5,6,7,8}, {9}

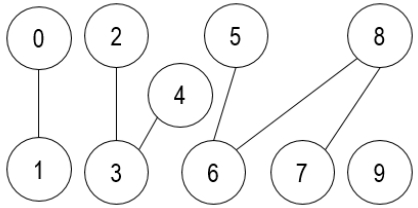
	0	1	2	3	4	5	6	7	8	9
parent	1	1	4	4	4	8	8	8	8	9

find(5)==find(8) ?

	0	1	2	3	4	5	6	7	8	9
parent	1	1	3	4	4	6	8	8	8	9

find(5)==find(8) ?

서로소집합(disjoint set) 표현: 개념

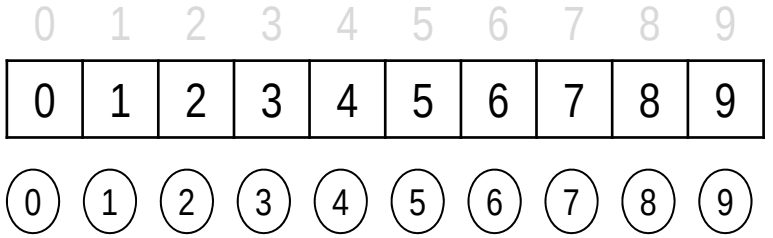


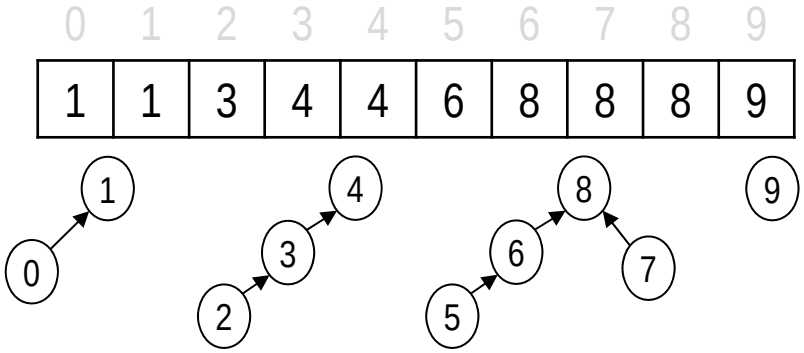
union(a, b) → a가 속한 집합과 b가 속한 집합을 합집합

{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}

union(0,1)
union(2,3), union(3,4)
union(5,6), union(7,8), union(6,8)

{0,1}, {2,3,4}, {5,6,7,8}, {9}

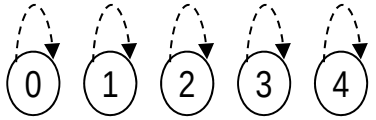




서로소집합(disjoint set) 개념: union

{0}, {1}, {2}, {3}, {4}

	0	1	2	3	4
parent	0	1	2	3	4

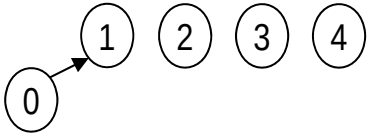


union(0,1)
0이 속한 집합과 1이 속한 집합을 합집합

노드 0의 parent는 노드 1이다.

{0,1}, {2}, {3}, {4}

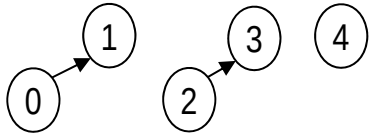
	0	1	2	3	4
parent	1	1	2	3	4



union(2,3)
2가 속한 집합과 3이 속한 집합을 합집합

{0,1}, {2,3}, {4}

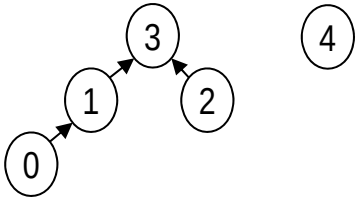
	0	1	2	3	4
parent	1	1	3	3	4



union(1,3)
1이 속한 집합과 3이 속한 집합을 합집합

{0,1,2,3}, {4}

	0	1	2	3	4
parent	1	3	3	3	4



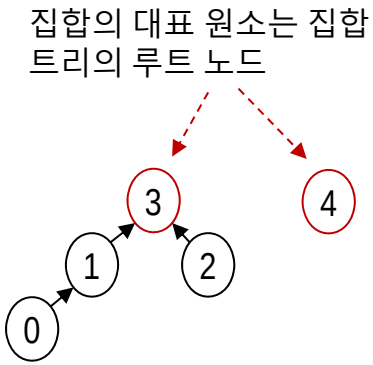
서로소집합(disjoint set) 연산: find

find(i)

- 원소 i가 속한 집합의 대표 원소 id를 반환

{0,1,2,3}, {4}

	0	1	2	3	4
parent	1	3	3	3	4



i==parent[i] 을 만족하는 i가 집합의 대표 원소 id

```
public int find(int i) {
    while( i != parent[i] ) i=parent[i];
    return i;
}
```

find(0) → 3
find(1) → 3
find(2) → 3
find(3) → 3
find(4) → 4

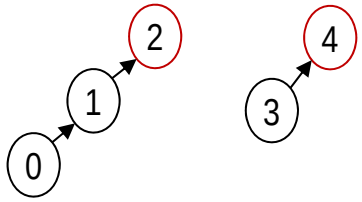
서로소집합(disjoint set) 연산: union

union(i, j)

- 원소 i가 속한 집합과 원소 j가 속한 집합을 합집합

{0,1,2}, {3,4} parent

0	1	2	3	4
1	2	2	4	4

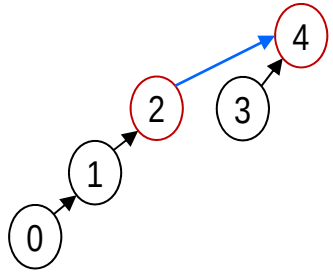



```
public int find(int i) {
    while(i!=parent[i]) i=parent[i];
    return i;
}
public void union(int i, int j) {
   i=find(i);
   j=find(j);
   if(i==j) return;
   parent[i]=j;
}
```

union(0,3)

{0,1,2,3,4} parent

0	1	2	3	4
1	2	4	4	4



서로소집합(disjoint set) 연산: find by path compression

경로 압축 기반 find 연산

```
public int find(int i) {
    if(i!=parent[i]) parent[i]=find(parent[i]);
    return parent[i];
}
```

find(0)
0!=parent[0]
parent[0]=find(parent[0])
return parent[0]

find(1)
1!=parent[1]
parent[1]=find(parent[1])
return parent[1]

4

find(2)
2!=parent[2]
parent[2]=find(parent[2])
return parent[2]

4

find(3)
3!=parent[3]
parent[3]=find(parent[3])
return parent[3]

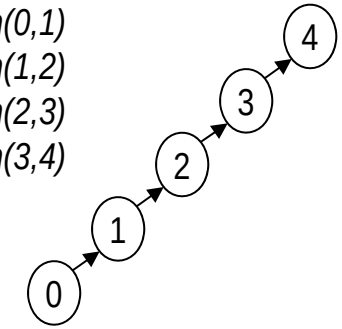
4

find(4)
4==parent[4]
return parent[4]

4

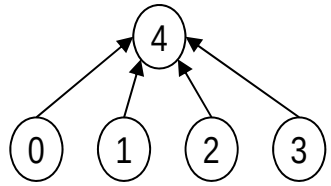
	0	1	2	3	4
parent	1	2	3	4	4

union(0,1)
union(1,2)
union(2,3)
union(3,4)



find(0)

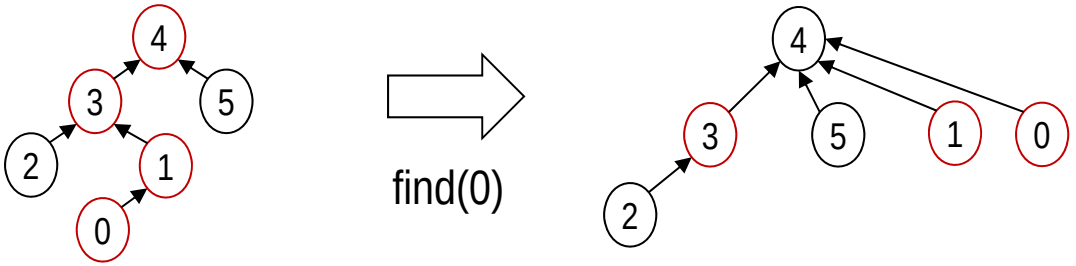
	0	1	2	3	4
parent	4	4	4	4	4



서로소집합(disjoint set) 연산: find by path compression

경로 압축 기반 find 연산

```
public int find(int i) {
        if(i!=parent[i]) parent[i]=find(parent[i]);
        return parent[i];
}
```

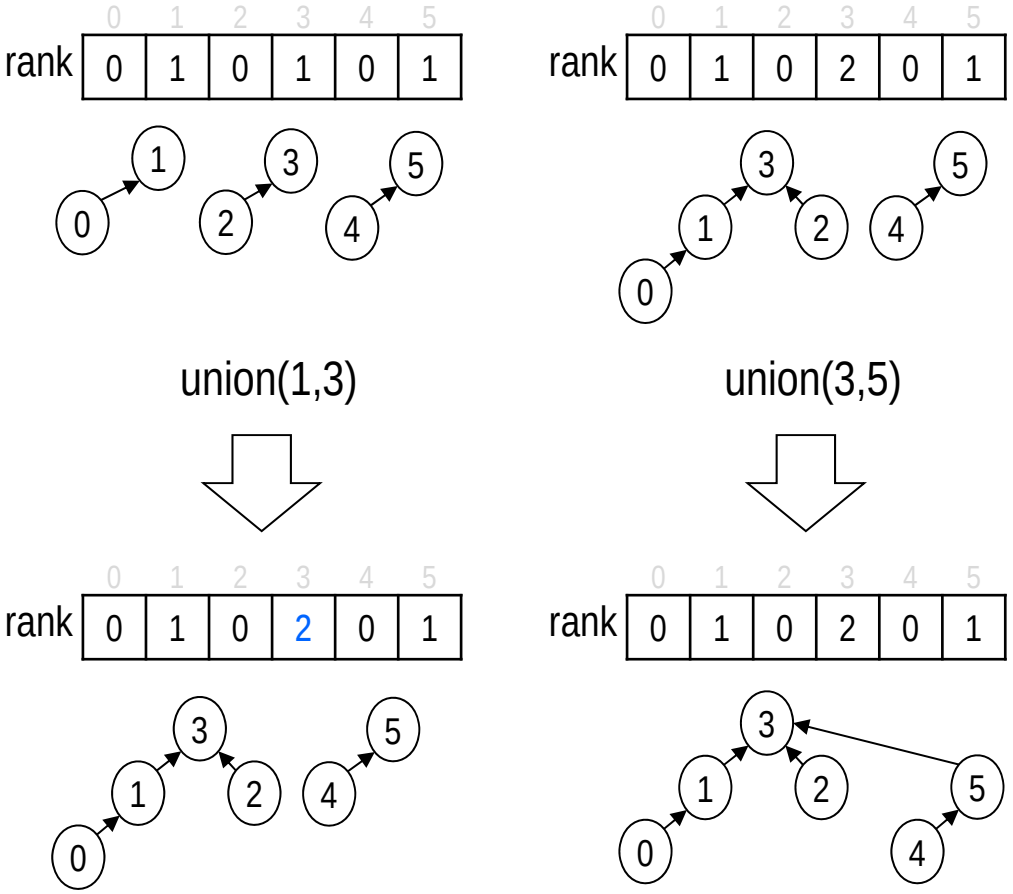


서로소집합(disjoint set) 연산: union by rank

rank → 특정 집합에 대응하는 트리의 높이와 유사

rank 기반 union

```
public void union(int i, int j) {
        i=find(i);
        j=find(j);
        if(i==j) return;
        if(rank[i]<rank[j]) parent[i]=j;
        else if(rank[i]>rank[j]) parent[j]=i;
        else{
                parent[i]=j;
                rank[j]++;
        }
}
```



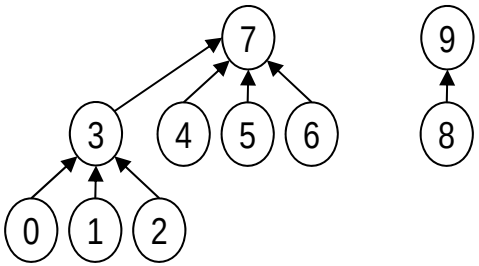
Union-find 자료구조

```
public class UF {
	int	parent[], rank[], count;
	public UF(int N) {
		parent=new int[N];
		rank=new int[N];
		for (int i = 0; i < parent.length; i++){
			parent[i]=i;
		}
	}
	public void union(int i, int j) { // union by rank
		i=find(i);
		j=find(j);
		if(i==j) return;
		if(rank[i]<rank[j]) parent[i]=j;
		else if(rank[i]>rank[j]) parent[j]=i;
		else{
			parent[i]=j;
			rank[j]++;
		}
	}
	public int find(int i) { // find by path compression
		if(i!=parent[i]) parent[i]=find(parent[i]);
		return parent[i];
	}
	@Override
	public String toString() { return Arrays.toString(parent); }
}
```

실습: 그래프 내 연결요소의 개수가 count에 저장되도록 클래스 UF를 수정하시오

실습: find 함수를 비재귀적으로 구현하시오

```
public class Test {
	public static void main(String[] args) {
		int	N=10;
		UF	uf=new UF(N);
		System.out.println(uf);
		uf.union(0,1);
		uf.union(2,3);
		uf.union(4,5);
		uf.union(6,7);
		uf.union(8,9);
		uf.union(0,2);
		uf.union(4,6);
		uf.union(0,4);
		System.out.println(uf);
		System.out.println(uf.find(1)==uf.find(6));
		//System.out.println("연결요소 개수 = "+uf.count);
	}
}
```



Reference: https://algs4.cs.princeton.edu/15uf/UF.java.html, GPLv3
Reference: Reference: https://en.wikipedia.org/wiki/Disjoint-set_data_structure, CC-BY-SA

서로소집합과 union-find

- Union-find 시간복잡도
  - Find
    - O(α(n))
  - Union
    - O(α(n))

α(n)
- inverse Ackermann function
- 대부분의 경우 $\alpha(n)<5$
- Ackermann function $A(4,4) \approx 2^{2^{2^{65536}}}$

- Union-find 응용
  - 동적으로 변하는 비방향 그래프에서 연결요소들 표현
  - 그래프 내 임의의 두 노드 간 연결 경로 존재 여부 판단
  - Kruskal의 최소신장트리 알고리즘에서 활용

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 프로그래밍 콘테스트 챌린징, Akiba 등 공저, 로드북, 2011.
- https://introcs.cs.princeton.edu/
- Introduction to Algorithms, Cormen et al., 3rd Edition (The MIT Press)