1. (서로소집합 개념) 다음은 union-find 연산에 기반한 서로소집합 표현을 소개하는 예시 코드이다. 이 코드를 입력하고 실행하면서 서로소집합 표현법을 학습하시오. 아래 코드의 union-find 구현 방식은 비효율적이므로 개선이 필요하다. (Reference: https://en.wikipedia.org/wiki/Disjoint-set_data_structure, CC-BY-SA)

```
public class Test {
	public static void main(String[] args) {
		// union-find 연산을 이용한 <서로소집합>의 표현방법: {0,1}, {2,3,4}, {5,6,7,8}, {9}
		// 모든 단일원소집합들을 만든 후 union(합집합) 연산을 적용
		int	N=10;
		UF	uf=new UF(N); // 단일원소집합들 생성: {0},{1},{2},{3},...,{N}
		System.out.println(uf);
		uf.union(0,1); // 0->1
		uf.union(2,3); // 2->3
		uf.union(3,4); // 2->3->4
		uf.union(5,6); // 5->6
		uf.union(6,7); // 5->6->7
		uf.union(7,8); // 5->6->7->8
		System.out.println(uf);

		System.out.println(uf.find(5)==uf.find(7)); // find 연산 통해 노드 연결 여부 검사
		System.out.println(uf.find(5)==uf.find(2)); // find 연산 통해 노드 연결 여부 검사
		uf.union(4, 5);
		System.out.println(uf);
	}
}
public class UF {
	int	parent[];
	public UF(int N) {
		parent=new int[N];
		for (int i = 0; i < parent.length; i++) parent[i]=i;
	}
	public void union(int i, int j) {
		i=find(i);
		j=find(j);
		if(i==j) return;
		parent[i]=j;
	}
	public int find(int i) {
		while(i!=parent[i]) i=parent[i];
		return i;
	}
	@Override
	public String toString() {
		return Arrays.toString(parent);
	}
}
```

2. (**실습**: find by path compression & union by rank) 다음은 경로압축을 병행하는 find 연산과 rank에 기반한 union 연산을 활용하여 서로소집합을 효율적으로 구현한 코드이다. 아래 코드를 입력하고 실행하면서 서로소집합을 위한 union-find 연산 구현법을 학습하시오. (Reference: https://algs4.cs.princeton.edu/15uf/UF.java.html, GPLv3, Reference: https://en.wikipedia.org/wiki/Disjoint-set_data_structure, CC-BY-SA)

- **실습** #1: 아래 find 함수는 재귀적으로 구현되었다. 이를 비재귀적으로 동작하도록 다시 작성하시오.

- **실습** #2: 아래 main 함수의 마지막 문장은 그래프 내 연결요소의 총 개수 uf.count를 출력한다. 이 문장이 정상 동작하도록 UF 클래스를 수정하시오. 힌트: 최초 연결요소의 수는 그래프 내 전체 노드의 수와 같으며, 이후 합집합 연산이 1회 발생할 때마다 연결요소의 수는 1씩 감소한다.

```
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);
	}
}
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) { // path compression
		if(i!=parent[i]) parent[i]=find(parent[i]);
		return parent[i];
	}
	@Override
	public String toString() {
		return Arrays.toString(parent);
	}
}
```

3. (**실습**: 서로소집합 응용) 다음은 최초 V명의 사용자를 갖는 어떤 SNS 서비스에서 임의의 두 사용자 간에 친구 맺기를 V/2회 수행한 후, 총 V명 사용자들 내에 형성된 연결요소(친구그룹)의 개수와 임의 선택된 일부 사용자 쌍의 친구관계여부를 출력하는 코드이다. A와 B가 친구 관계에 있고, B와 C가 친구 관계에 있으면 A와 C도 친구 관계에 있다고 가정하며 친구그룹은 친구관계에 있는 모든 사용자들의 모음으로 정의한다.

- **실습** #1: V를 1000000으로 변경한 후 아래 코드의 실행 시간을 측정해 보시오.

- **실습** #2: 친구그룹의 크기를 그룹 내 사용자의 수라고 정의할 때, 가장 큰 친구그룹의 크기를 출력하는 코드를 작성하시오.

```
public class Test {
	public static void main(String[] args) {
			int		V=1000;
			UF		uf=new UF(V);
			Random	random=new Random();
			for (int i = 0; i < V/2; i++){
				int		u=random.nextInt(V);
				int		v=random.nextInt(V);
				uf.union(u, v);
			}
			System.out.println("연결요소 개수="+uf.count);
			for (int i = 0; i < 5; i++) {
				int		u=random.nextInt(V);
				int		v=random.nextInt(V);
				System.out.println("connected("+u+","+v+") ? => "+(uf.find(u)==uf.find(v)));
			}
	}
}
```

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍 미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.
- 남궁성. 자바의 정석. 도우출판.
- 김윤명. (2010). 뇌를 자극하는 JSP & Servlet. 한빛미디어.