[읽을거리] 4. 집합

시간복잡도

Set의 종류별 시간복잡도는 다음과 같습니다.

	add()	contains()	next()
HashSet	O(1)	O(1)	O(h/n)
LinkedHashSet	O(1)	O(1)	O(1)
EnumSet	O(1)	O(1)	O(1)
TreeSet	O(log n)	O(log n)	O(log n)
CopyOnWriteArraySet	O(n)	O(n)	O(1)
ConcurrentSkipList	O(log n)	O(log n)	O(1)

HashSet의 경우 next() 는 O(h/n) 을 갖습니다. (h는 bucket의 크기, n은 데이터의 수)

n이 적을 때는 대부분의 bucket이 비어있게 되고, 다음값이 저장된 bucket을 찾기위해 탐색하게 됩니다. n이 증가하여 bucket의 대부분이 차있을 경우 다음값이 저장된 bucket을 찾는 시간 복잡도는 O(1)에 수렴하게 됩니다.

합집합/차집합/교집합

합집합은 `addAll`, 차집합은 `removeAll`, 교집합은 `retainAll` 을 사용해서 각 집합을 구하게 됩니다. 이 과정에서 `contains` 를 통해 중복되는 요소를 찾게 됩니다.

HashSet을 사용하게 된다면 O(1) 의 시간복잡도를 갖게되고, TreeSet을 사용한다면 O(log n) 의 시간복잡도를 갖게 될것입니다.

List의 중복요소 제거하기

addAll 을 사용하는 방법

```
List<String> list = Arrays.asList("A", "B", "C", "A", "B", "C");
Set<String> set = new HashSet<>(list);
System.out.println(set);  // [A, B, C]
```

단순히 Set을 만들고 List를 addAll 하여 중복을 제거할 수 있습니다.

```
List<String> list = Arrays.asList("A", "B", "C", "A", "B", "C");
Set<String> set = new HashSet<>(list);
List<String> list2 = new LinkedList<>(set);
System.out.println(list2);  // [A, B, C]
```

Set을 다시 List로 만드려면 List를 생성하고 addAll로 넣어주면 됩니다.

stream을 사용하는 방법

```
List<String> list = Arrays.asList("A", "B", "C", "A", "B", "C");
List<String> list2 = list.stream().distinct().collect(Collectors.toList());
System.out.println(list2);  // [A, B, C]
```

stream의 `distinct` 를 사용해서 중복을 제거할 수 있습니다. stream을 다시 List로 만들기 위해 collect 를 사용하면 됩니다.

리스트로 Set 효과 내기

합집합 만들기

```
<T> List<T> union(List<T> list1, List<T> list2) {
    List<T> union = new LinkedList<>();
    union.addAll(list1);
    for (T t : list2) if (!union.contains(t)) union.add(t);
    return union;
}
```

차집합 만들기

```
<T> List<T> difference(List<T> list1, List<T> list2) {
    List<T> difference = new LinkedList<>();
    difference.addAll(list1);
    for (T t : list2) difference.remove(t);
    return difference;
}
```

교집합 만들기

```
<T> List<T> intersection(List<T> list1, List<T> list2) {
    List<T> intersection = new LinkedList<>();
    for (T t : list1) if(list2.contains(t)) intersection.add(t);
    return intersection;
}
```