# 2023 여름학기 | 동국대학교 SW 역량강화 캠프

## 15일차. 셋, 맵

## ● Set (셋)

▶ 중복되지 않는 값들(Key)을 모아놓은 자료구조

▶ 삽입, 삭제, 탐색의 세 연산을 지원

▶ HashSet과 TreeSet의 두 종류

| 자료구조 | HashSet | TreeSet |
|---|---|---|
| 시간복잡도 | 세 연산 모두 O(1) | 세 연산 모두 O(lgN) |
| 장점 | 빠르다 | key가 언제나 정렬되어 있다 |
| 단점 | key의 순서를 예측할 수 없다 | HashSet에 비하면 느리다. |

# Set 핵심 코드 (선언)

▶ HashSet 자료구조는 java.util.HashSet 에
TreeSet 자료구조는 java.util.TreeSet 에 정의되어 있습니다.

```
import java.util.HashSet;
import java.util.TreeSet;
```

▶ Set 자료구조는 다음과 같이 선언할 수 있습니다.

```
TreeSet<Integer> tset = new TreeSet<>();
HashSet<Integer> hset = new HashSet<>();
// 참조형 객체만 사용 가능합니다.
// TreeSet<Object> tset_name = new TreeSet<>();
// HashSet<Object> hset_name = new HashSet<>();
```

# Set 핵심 코드 (삽입, 탐색, 삭제)

▶ Set 자료구조는 add 메소드를 이용하여 set에 원소를 추가할 수 있습니다.
Boolean 자료형을 return하며, 중복된 원소일 경우 false값을 return 합니다.

```
boolean tadd = tset.add(3); // tset에 3을 집어넣는다. 이미 3이 셋 내부에 있을 경우 tadd는 false가 된다.
boolean hadd = hset.add(3); // hset에 3을 집어넣는다. 이미 3이 셋 내부에 있을 경우 hadd는 false가 된다.
```

▶ Set 자료구조는 contains 메소드를 이용하여 값이 set 내부에 존재하는지 확인할 수 있습니다.

```
boolean contain = set.contains(3); // set에 3이 있다면 true, 아니라면 false를 contain에 저장한다.
```

▶ Set 자료구조는 remove 메소드를 이용하여 set에서 값을 제거할 수 있습니다.

```
boolean remove = set.remove(3); // set에서 3을 제거합니다. 성공적으로 제거했다면 remove는 true가 됩니다.
```

## Set 핵심 코드 (메소드)

▶ Set 자료구조는 size, isEmpty, toArray 메소드를 사용할 수 있습니다.

```
int x = set.size(); // x에 set의 원소 수를 저장합니다.
boolean empty = set.isEmpty(); // empty에 set이 비어있는지 여부를 저장합니다.

Integer[] arr = set.toArray(); // set의 key 목록을 array로 변경하여 arr에 저장합니다. TreeSet의 경우 정렬되어 있습니다.
ArrayList<Integer> arr = new ArrayList<String>(set); // set의 key 목록을 ArrayList arr에 저장합니다.
// HashSet의 경우 정렬을 위해서 ArrayList에서 Collection.sort를 진행하면 됩니다.
```

▶ TreeSet 자료구조는 내부가 정렬되어 있어 pollFirst, pollLast, higher, lower 등의 메소드를 사용할 수 있습니다.

```
int tsetmin = tset.pollFirst(); // tset의 key들 중 최소값을 저장합니다. O(lgN)
int tsetmax = tset.pollLast(); // tset의 key들 중 최대값을 저장합니다. O(lgN)

int higher = tset.higher(4); // tset의 key들 중 4보다 큰 값을 가진 것들 중 최소값을 저장합니다 O(lgN)
int lower = tset.lower(4); // tset의 key들 중 4보다 작은 값을 가진 것들 중 최대값을 저장합니다 O(lgN)
```

# Python과 C++에서 set

```python
s = set()

s = set([1, 2, 3])
# s = {1, 2, 3}

s = set("Hello")
# s = {'e', 'H', 'l', 'o'}

arr = list(s)
# arr = ['e', 'H', 'l', 'o']

s.add(3)        # 값 추가
s.remove(3)     # 값 삭제
len(s)          # 원소 개수
```

```cpp
#include <iostream>
#include <set>

using namespace std;

int main()
{
    set<int> s;
    set<int>::iterator it;

    s.insert(3);                                // 값 추가
    if(s.find(3) != s.end()) s.erase(3);        // 값 삭제

    for(it=s.begin(); it!=s.end(); it++){
        cout << *it;
    }
}
```

# ● 서로 다른 수 (4677)

### 문제

n개의 정수가 들어왔을 때, 서로 다른 정수가 모두 몇 종류 들어왔는지 출력하시오
예제 입출력에서는 서로 다른 정수는 2,3,4,100의 4종류이므로 4를 출력한다.

### 입력

첫 줄에 입력으로 주어지는 정수의 개수 $N$이 주어진다 $(1 \leq N \leq 200,000)$
둘째 줄에 정수 $N$개가 공백을 사이에 두고 주어진다. 이 정수들은 1 이상 10억 이하로 주어진다

### 출력

주어진 정수들 중 서로 다른 정수가 몇 종류 있는지 자연수 하나로 출력하시오.

## ● Set을 이용한 풀이

▶ 모든 수들을 Set에 add하고, Set의 size를 출력

▶ 정렬을 해 줄 필요가 없으므로 HashSet 사용

```
for(int i=0; i<N; i++) {
    int x = Integer.parseInt(st.nextToken());
    set.add(x);
}
System.out.println(set.size());
```

# ● 출퇴근 (4679)

## 문제

윌리가 일하는 회사에는 출근할 때와 퇴근할 때에 카드를 태그하여 출퇴근 기록을 기록할 수 있는 기기가 있다.
윌리는 지금 회사에 누가 일하고 있는지 궁금하여 이 기기의 로그를 전부 가지고 왔다.
이 로그는 직원의 이름과 enter/leave 의 쌍으로 이루어져있다.
enter는 출근 기록, leave는 퇴근 기록을 의미한다.
윌리를 도와 기기의 로그로부터 현재 회사에서 일을 하고 있는 사람들의 명단을 구해보자

## 입력

첫째 줄에 로그의 기록 수 $N$이 주어진다 ($N \leq 10^6$)
둘째 줄부터 N개 줄에 걸쳐 로그의 기록들이 주어진다
로그의 기록은 직원의 이름과 enter 또는 leave가 공백을 사이에 두고 주어진다.
직원의 이름은 최장 5글자의 알파벳 대소문자로 이루어진 문자열이며, 대소문자가 다른 경우 다른 직원이다.

## 출력

첫 줄에 현재 일을 하고 있는 직원의 수를 출력한다.
이후 한 줄에 한 개씩 현재 일을 하고 있는 직원들의 이름을 한 줄에 하나씩 오름차순으로 출력한다. (문자열 기본 정렬 순서)

## ● Set을 이용한 풀이

▶ set에 출근한 사람은 add, 퇴근한 사람은 remove하여 회사 내부에 있는 사람들을 set으로 관리

▶ 정렬하여 출력하여야 하므로 TreeSet 사용

▶ 또는 HashSet 사용 후, 출력 직전에 정렬

# 핵심 코드(TreeSet)

```java
int N = Integer.parseInt(br.readLine());

TreeSet<String> set = new TreeSet<>();
for(int i=0; i<N; i++){
    st = new StringTokenizer(br.readLine());

    String name = st.nextToken();
    String status = st.nextToken();

    if(status.equals("enter")) set.add(name);
    else set.remove(name);
}

System.out.println(set.size());
for(String name: set){
    System.out.println(name);
}
```

```python
N = int(input())

s = set()
for i in range(N):
    name, status = input().split()

    if status == "enter":
        s.add(name)
    else:
        s.remove(name)

s = sorted(s)

print(len(s))
for name in s:
    print(name)
```

## ● Map (맵)

▶ Key와 Value의 쌍들을 모아놓은 자료구조 (Key는 중복 X)

▶ Index에 제한이 없는 1차원 배열

▶ HashMap과 TreeMap의 두 종류

| 자료구조 | HashSet | TreeSet |
|---|---|---|
| 시간복잡도 | 삽입/key탐색/수정/삭제 O(1) | 삽입/key탐색/수정/삭제 O(lgN) |
| 장점 | 빠르다 | key가 언제나 정렬되어 있다 |
| 단점 | key의 순서를 예측할 수 없다 | HashMap에 비하면 느리다. |

# Map 핵심 코드(선언)

▶ HashMap 자료구조는 java.util.HashMap 에
TreeMap 자료구조는 java.util.TreeMap 에 정의되어 있습니다.

```
import java.util.HashMap;
import java.util.TreeMap;
```

▶ Map 자료구조는 다음과 같이 선언할 수 있습니다.

```
HashMap<String, Integer> hmap = new HashMap<>();
TreeMap<String, Integer> tmap = new TreeMap<>();
// HashMap<KeyObject, ValueObject> hmap_name = new HashMap<>();
// TreeMap<KeyObject, ValueObject> tmap_name = new TreeMap<>();
```

## Set 핵심 코드 (삽입, 탐색, 삭제, 수정)

▶ Map 자료구조는 put 메소드를 이용하여 Map에 원소를 추가할 수 있습니다.

```
map.put("Alice", 3); // map에 key = "Alice", value = 3 인 key-value 쌍을 매핑한다.
```

▶ Map 자료구조는 containsKey/Value, get 메소드를 이용하여 Map 내부 값을 탐색할 수 있습니다.

```
boolean conkey = map.containsKey("Alice"); // map에 key="Alice" 인 매핑 정보가 있는지 여부를 conkey에 저장한다.
//HashMap O(1)  // TreeMap O(lgN)
boolean conval = map.containsValue(3); // map에 value = 3인 매핑 정보가 있는지 여부를 conval에 저장한다.
//HashMap O(N) // TreeMap O(N)
int x = map.get("Alice"); // x에 key = "Alice"인 매핑정보에 대한 value를 저장함.
```

▶ Map 자료구조는 remove 메소드를 이용하여 Map에서 값을 제거할 수 있습니다.

```
map.remove("Alice"); // map에서 key="Alice"인 매핑 정보를 제거합니다.
```

▶ Map 자료구조는 replace 메소드를 이용하여 Map에서 Value 값을 수정할 수 있습니다.

```
map.replace("Alice", 5); // map에서 key="Alice"인 매핑 정보의 value를 5로 수정합니다.
```

# Map 핵심 코드 (메소드)

▶ Map 자료구조는 size, isEmpty, KeySet 메소드를 사용할 수 있습니다.

```
int size = map.size(); // size에 map에 저장된 매핑 정보의 개수를 저장합니다.
Boolean empty = map.isEmpty(); // map이 비어있는 지 여부를 empty에 저장합니다.
Set<String> set = map.keySet(); // map에 저장된 매핑정보들의 key를 Set 자료형으로 set에 저장합니다.
```

▶ TreeMap 자료구조는 key들이 정렬되어 있어 key들 간 대소비교 등의 다양한 메소드를 사용할 수 있습니다.

# Python과 C++에서 map

```python
dic = {}
dic = {'a': 1, 'b': 2}

dic['c'] = 3        # 값 추가
dic['a'] = 4        # 값 수정

# 탐색
for key in dic:
for val in dic.values():
for key, val in dic.items():

if 'b' in dic:      # key값으로만 찾을 수 있음

del dic['b']        # 삭제
```

```cpp
#include <iostream>
#include <map>

using namespace std;

int main()
{
    map<string, int> Map;
    map<string, int>::iterator it;

    Map["abc"] = 1;
    Map["abc"]++;

    for(it=Map.begin(); it!=Map.end(); it++){
        cout << it->first << ' ' << it->second;
    }
}
```

# ● 도서관 (4680)

### 문제

윌리는 도서관에서 사서로 일한다.
윌리는 오늘 하루 동안 가장 많이 대여된 책을 오늘의 추천 도서로 선정하려고 한다.
오늘 대여된 책의 이름 N개가 주어질 때, 가장 많이 대여된 책의 이름을 출력해보자.

### 입력

첫 줄에 대여된 책의 권 수 $N$이 주어진다.($N \leq 100,000$)
이후 N개 줄에 걸쳐 책의 이름이 주어집니다. 책의 이름은 20자 이하의 소문자로만 이루어진 문자열입니다.

### 출력

가장 많이 대여된 책의 이름을 출력합니다. 만약 가장 많이 대여된 책이 여러 권이라면 가장 사전순으로 빠른 책의 이름을 출력합니다.

## ● Map을 이용한 풀이

▶ Map에 {key = 책 이름, value = 대여 횟수} 로 정보를 매핑시킨다.

▶ 모든 매핑이 끝난 이후 value가 가장 큰 매핑 정보의 key를 출력시킨다.

▶ TreeMap을 이용하면, 마지막에 Key를 탐색할 때 사전순으로 빠른 순으로 탐색할 수 있다.

## 핵심 코드

```java
TreeMap<String, Integer> map = new TreeMap<>();
for(int i=0; i<N; i++){
    String book = br.readLine();

    if(!map.containsKey(book)) map.put(book, 0);
    map.replace(book, map.get(book)+1);
}

int max=0;
String ans = new String();
for(String book: map.keySet()){
    if(max < map.get(book)){
        max = map.get(book);
        ans = book;
    }
}

System.out.print(ans);
```

```python
N = int(input())

dic = {}
for i in range(N):
    book = input()

    if book not in dic: dic[book] = 0
    dic[book] += 1

max = 0
book = ""
for key, val in dic.items():
    if max < val or (max == val and book > key):
        max = val
        book = key

print(book)
```