

알고리즘2 (2024-2)

3. 해시 자료구조의 활용

국립금오공과대학교 컴퓨터공학과

김 경 수

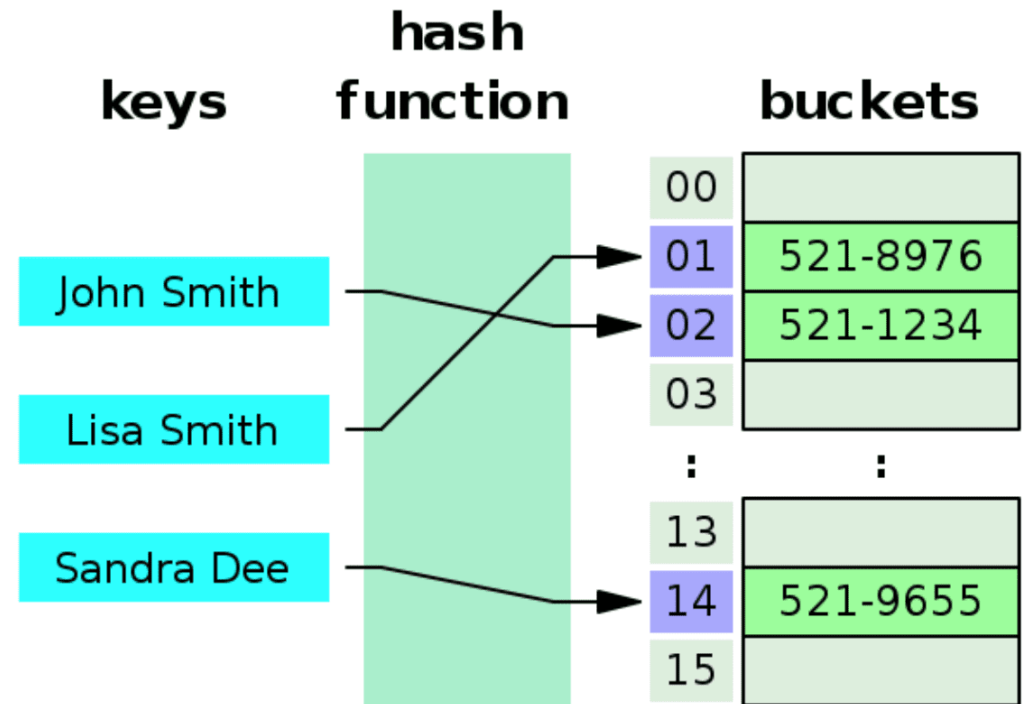
학습 목표

- ① 해시 테이블의 정의와 특성을 복습하고 코딩 테스트에서 어떤 경우에 해시 테이블을 사용하는 것이 효과적인지 이해할 수 있다.
- ② 주요 프로그래밍 언어에서 해시 테이블을 활용하는 방법을 이해하고 실습을 통해 해시 테이블의 활용 능력을 숙달할 수 있다.
- ③ 실제 코딩테스트 문제 풀이를 통해 해시 테이블이 사용되는 문제의 특성을 명확하게 이해하고 제한된 조건 하에서 효과적으로 문제를 해결할 수 있다.

해시 자료구조 리뷰

해시 테이블

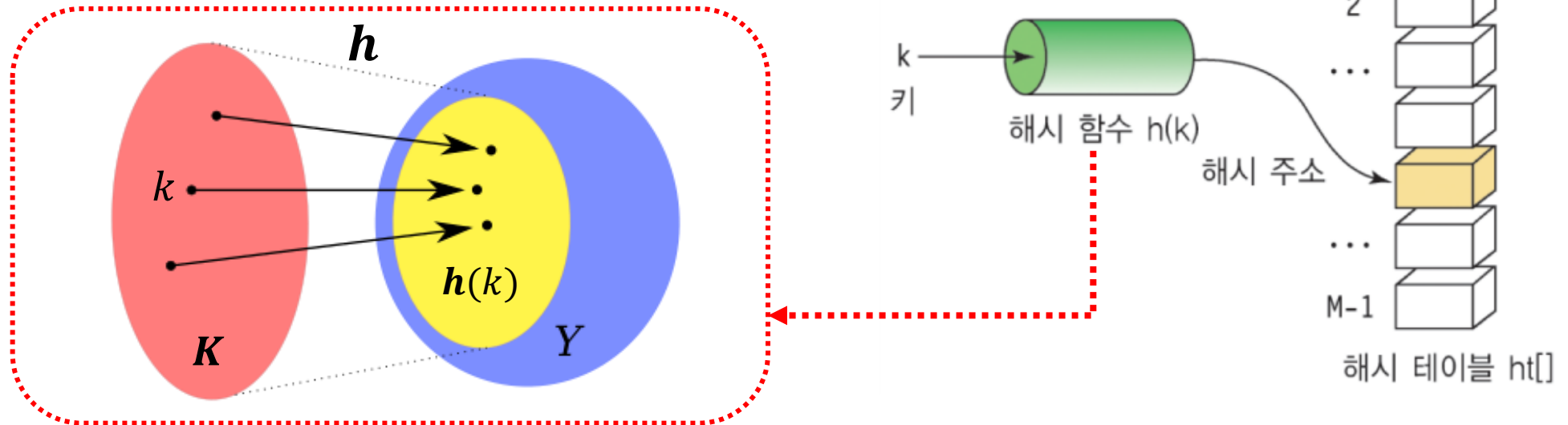
- 해시 함수(hash function)를 사용해서 변환한 값을 인덱스로 삼아 키와 값을 저장해서 빠른 데이터 탐색을 제공하는 자료구조
- 대부분의 언어에서 해시 테이블 기능을 기본적으로 제공
 - Hash table, Hash map, Hash set, 또는 Dictionary 등의 형태로 제공
- **해시 테이블(Hash table)**
 - (Key, Value)의 집합
 - 해시 테이블 내 데이터의 탐색은 키(Key)와 해시 함수를 이용하여 수행
 - Key → **해시 함수** → 해시 테이블의 인덱스



해시 함수

• 해시 함수(Hash function)

- 주어진 키를 해시 테이블의 인덱스로 변환하는 함수



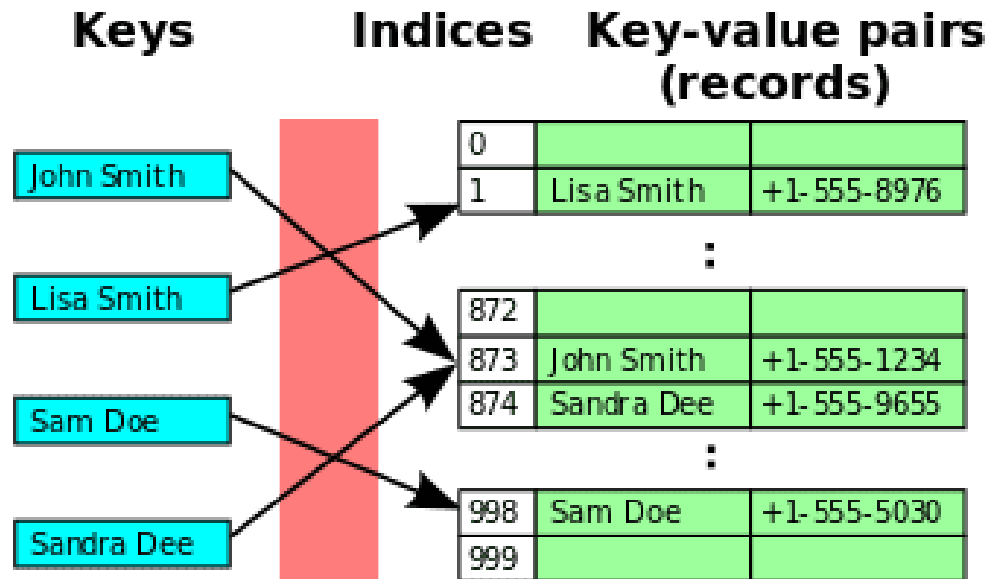
- 해시 함수는 정의역인 키 값과 치역인 해시 테이블의 인덱스가 일대일 대응이 이루어지도록 설계하지만, 모든 키 값에 대하여 일대일 대응이 이루어지기는 쉽지 않음.
- 즉, 서로 다른 키에 대해서도 같은 해시 값이 도출되는 상황이 발생할 수 있음.

→ 충돌(Collision)

해시 함수의 충돌 해결법

• 개방 주소법(Open addressing)

- 충돌이 발생한 키 값을 테이블 내 다른 빈 공간(bucket)을 찾아서 저장하는 방법
- 주어진 키 값에 대한 해시 값에 일정한 상수를 더해서 다른 주소에 할당
- 다른 해시 값에 대해서도 연속적으로 충돌이 발생하는 문제가 발생할 수 있음



Hash function

해시 함수의 충돌 해결법

• 개방 주소법의 종류

➤ 선형 조사법(Linear probing)

- 충돌이 발생한 위치에서 1씩 더하면서 할당할 위치를 지정 (m 은 해시 테이블의 최대 크기)

$$h(k, i) = (h(k) + i) \bmod m$$

➤ 이차 조사법(Quadratic probing)

- 충돌이 발생한 위치에서 제곱 수를 더하여 할당할 위치를 결정

$$h(k, i) = (h(k) + i^2) \bmod m$$

➤ 이중 해시법(Dobule probing)

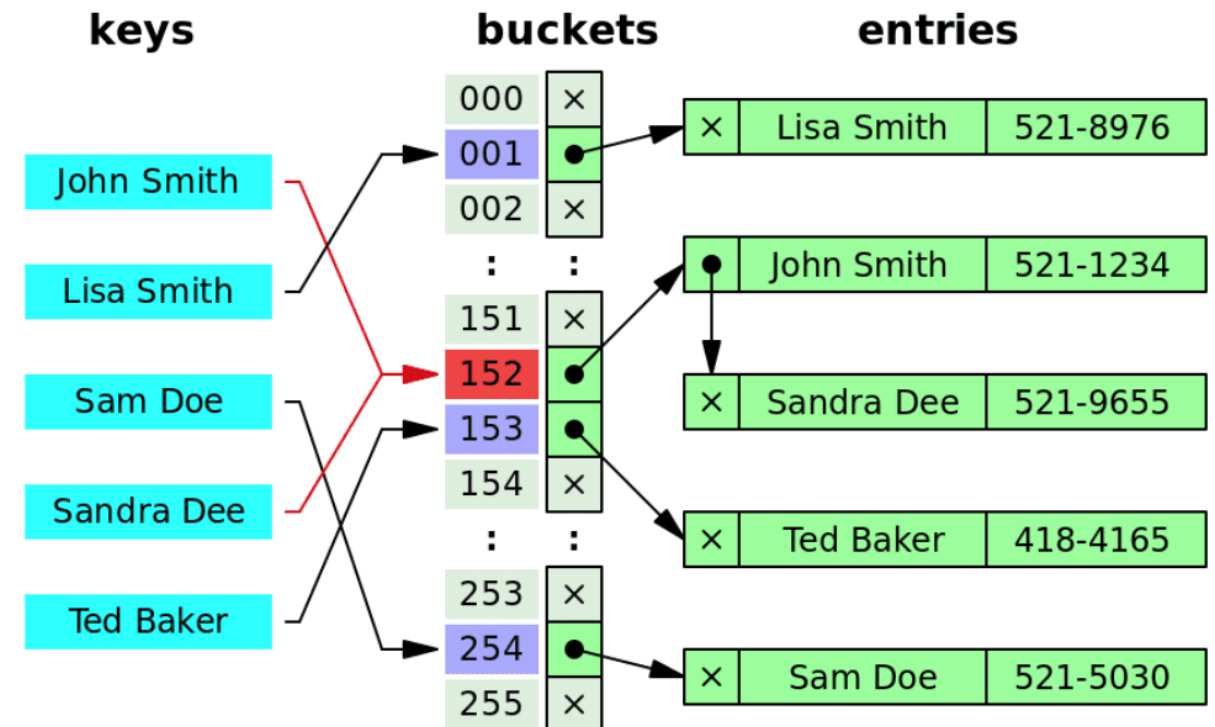
- 충돌이 발생하는 경우 기존의 해시 함수와 다른 해시 함수를 사용

$$h(k, i) = (h_1(k) + i \times h_2(k)) \bmod m$$

해시 함수의 충돌 해결법

• 체이닝(Chaining)

- 해시 값에 해당하는 위치에 다른 value가 포함되어 있다면, 해당 위치에 연결 리스트(linked-list)의 형태로 해당 value를 저장하는 방법
- 충돌이 발생한 위치에 많은 수의 value가 저장되어 있는 경우 탐색에 소요되는 시간이 증가하는 문제점이 존재함



Python에서 해시 테이블 활용 방법 (Dictionary)

작업	코드
해시 테이블 선언/초기화	<pre>keys = ['kim', 'lee', 'park', 'jung', 'choi', 'kwon'] values = [50, 45, 40, 30, 20, 15] htable = dict(zip(keys, values))</pre>
(Key, Value) 삽입/변경	<pre>htable['song'] = 12</pre>
(Key, Value) 탐색	<pre>htable['park'] 또는 htable.get('park')</pre>
(Key, Value) 삭제	<pre>htable.pop('lee') 또는 del htable['lee']</pre>
Key list 반환	<pre>htable.keys()</pre>
Value list 반환	<pre>htable.values()</pre>
Key 존재 여부 확인	<pre>if 'kim' in htable: ... else: ...</pre>

언제 해시 테이블을 활용해야 하는가?

• 코딩 테스트에서 해시 테이블을 사용하는 경우

- ① 문제의 해결 방법 과정에서 데이터를 일련의 **키(key)와 값(value)의 쌍(pair)**으로 표현할 수 있는 경우
- ② “문자열”을 활용하여 데이터를 인덱싱하거나 검색해야 하는 경우
 - 특히, 문자열이 키(key)가 되고 각 문자열에 대한 값(value)들이 주어지는 경우
- ③ 특정 값이나 정보를 기준으로 빈번한 검색을 수행해야 하는 경우
- ④ 특정 정보와 매핑되는 값 사이의 관계를 확인해야 하는 경우

연습문제 1. 완주하지 못한 선수

문제 1. 완주하지 못한 선수 (1/2)

제한시간: 20분

• 문제 설명

- 수많은 마라톤 선수들이 마라톤에 참여하였다. 단 한 명의 선수를 제외하고는 모든 선수가 마라톤을 완주하였다.
- 마라톤에 참여한 선수들의 이름이 담긴 배열 participant와 완주한 선수들의 이름이 담긴 배열 completion이 주어질 때, 완주하지 못한 선수의 이름을 반환하도록 solution1 함수를 작성하시오.

• 제한사항

- 마라톤 경기에 참여한 선수의 수는 1명 이상 100,000명 이하이다.
- completion의 길이는 participant의 길이보다 1 작다.
- 참가자의 이름은 1개 이상 20개 이하의 알파벳 소문자로 이루어져 있다.
- 참가자 중에는 동명이인이 있을 수 있다.

문제 1. 완주하지 못한 선수 (2/2)

제한시간: 20분

• 입출력 예제

participant	completion	return
["leo", "kiki", "eden"]	["eden", "kiki"]	"leo"
["marina", "josipa", "nikola", "vinko", "filipa"]	["josipa", "filipa", "marina", "nikola"]	"vinko"
["mislav", "stanko", "mislav", "ana"]	["stanko", "ana", "mislav"]	"mislav"

• 입출력 예제 설명

- 예제 1: " leo " 는 참여자 명단에는 있지만, 완주자 명단에는 없기 때문에 완주하지 못함.
- 예제 2: " vinko " 는 참여자 명단에는 있지만, 완주자 명단에는 없으므로 완주하지 못함.
- 예제 3: " mislav " 는 참여자 명단에는 두 명이 있지만, 완주자 명단에는 한 명밖에 없기 때문에 한명은 완주하지 못함.

문제 1. 문제 분석

- 두 입력 리스트를 서로 비교하는 것이 핵심이다.
 - Participant 리스트에는 존재하나 completion 리스트에는 존재하지 않는 원소를 찾는 것이 본 문제의 목표이다.
- 입력 데이터는 텍스트로 구성된 리스트
 - 문자열 매칭 or 해시 함수를 사용
- 입력 리스트의 길이는 최대 약 200,000명이다.
 - 두 입력 리스트를 비교할 때 $O(n^2)$ 의 시간 복잡도는 허용되지 않음
 - 시간 복잡도를 $O(n \log n)$ 이하로 줄이는 것이 바람직함
- 동명 이인이 있을 수 있다.

participant	completion	return
["leo", "kiki", "eden"]	["eden", "kiki"]	"leo"
["marina", "josipa", "nikola", "vinko", "filipa"]	["josipa", "filipa", "marina", "nikola"]	"vinko"

문제 1. 문제 해결 전략 수립

- 예제 리스트

participant = [marina, josipa, nikola, vinko, filipa]

completion = [josipa, filipa, marina, nikola]

- 두 리스트 내 단어 간의 일대일 매칭이 필요한 상황

- 각 단어를 고유의 숫자로 바꾸는 것이 더욱 용이함 ➔ **해시 테이블 사용**

- 해시 테이블을 활용하는 방법

- 두 입력 리스트 중 participant를 해시 테이블에 추가한다.

- 이때 Key는 participant의 원소(참가자명), Value는 해당 참가자의 수로 정한다.

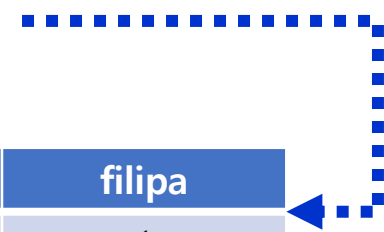
(Why? 동명이인 참가자가 존재할 수 있기 때문)

문제 1. 문제 해결 방법 기술

• 예제 리스트와 해시 테이블

participant = [marina, josipa, nikola, vinko, filipa]
 completion = [josipa, filipa, marina, nikola]

Keys	marina	josipa	nikola	vinko	filipa
Values	1	1	1	1	1



• 문제 해결 방법 기술

- ① 입력 리스트 participant를 이용하여 hash table을 생성한다.
 - Key: 참가자의 이름, Value: 동일 이름을 가진 사람의 수
- ② 리스트 completion의 각 원소(= 참가자 이름)가 hash table 내에 존재하는지 확인한다.
 - 만약 hash table에 참가자의 이름이 존재한다면, 해당 키에 대응되는 value를 1 감소시킨다.
- ③ Hash table에서 value가 1 이상인 key 리스트를 반환한다.
 - 완주한 사람(key)의 value는 0이 되므로, 이는 곧 완주하지 않은 사람들의 리스트를 반환하는 것과 같다.

문제 1. 구현 및 검증

```
def solution1(participant, completion):
    dic = {}

    # 참가자들을 딕셔너리에 추가
    # key: 참가자 이름, value: 동명이인 수
    for p in participant:
        if p in dic:
            dic[p] += 1
        else:
            dic[p] = 1

    # 딕셔너리에서 완주한 사람(key)의 value를 1 감소
    for c in completion:
        dic[c] -= 1

    # 딕셔너리의 value가 1 이상이면 미완주자
    for key in dic.keys():
        if dic[key] > 0:
            return key

    return [] # 모두 완주한 경우 빈 리스트 반환
```

테스트 케이스

```
# Test
participant = ["marina", "josipa", "nikola",
               "vinko", "filipa"]
completion = ["josipa", "filipa", "marina",
              "nikola"]

print(solution1(participant, completion))
```

연습문제 2. 베스트앨범

문제 2. 베스트앨범 (1/3)

제한시간: 50분

• 문제설명

- 스트리밍 사이트에서 장르 별로 가장 많이 재생된 노래를 두 개씩 모아 베스트 앨범을 출시하려 한다. 노래는 고유 번호로 구분하며, 노래를 수록하는 기준은 다음과 같다.
 - 속한 노래가 많이 재생된 장르를 먼저 수록한다.
 - 장르 내에서 많이 재생된 노래를 먼저 수록한다.
 - 장르 내에서 재생 횟수가 같은 노래 중 고유 번호가 낮은 노래를 먼저 수록한다.
- 노래의 장르를 나타내는 문자열 배열 genres와 노래별 재생 횟수를 나타내는 정수 배열 plays가 주어질 때, 베스트 앨범에 들어갈 노래의 고유 번호를 순서대로 반환하도록 solution2 함수를 완성하시오.

문제 2. 베스트앨범 (2/3)

제한시간: 50분

• 제한사항

- genres[i]는 고유번호가 i인 노래의 장르이다.
- plays[i]는 고유번호가 i인 노래가 재생된 횟수이다.
- genres와 plays의 길이는 같으며, 이는 1 이상 10,000 이하이다.
- 장르 종류는 100개 미만이다.
- 장르에 속한 곡이 하나라면, 하나의 곡만 선택한다.
- 모든 장르는 재생된 횟수가 다르다.

문제 2. 베스트앨범 (3/3)

제한시간: 50분

• 입출력 예제

genres	plays	return
["classic", "pop", "classic", "classic", "pop"]	[500, 600, 150, 800, 2500]	[4, 1, 3, 0]

• 입출력 예제 설명

- classic 장르는 1,450회 재생되었으며, classic 노래는 다음과 같다.
 - 고유 번호 3: 800회 재생
 - 고유 번호 0: 500회 재생
 - 고유 번호 2: 150회 재생
- pop 장르는 3,100회 재생되었으며, pop 노래는 다음과 같다.
 - 고유 번호 4: 2,500회 재생
 - 고유 번호 1: 600회 재생
- 따라서 pop 장르의 [4, 1]번 노래를 먼저, classic 장르의 [3, 0]번 노래를 그다음에 수록한다.
- 장르 별로 가장 많이 재생된 노래를 최대 두 개까지 모아 베스트 앨범을 출시하므로 2번 노래는 수록되지 않는다.

★ 문제 출처: <https://school.programmers.co.kr/learn/courses/30/lessons/42579>

문제 2. 문제 분석

• 입력 데이터의 의미 분석

- 첫 번째 입력 리스트: i 번째 노래의 장르를 의미한다.
- 두 번째 입력 리스트: i 번째 노래의 재생 횟수를 의미한다.

• 문제의 목표 분석

- 장르별로 가장 많이 재생된 노래 2곡 뽑는다.
- 노래 리스트를 만들기 위해서 아래의 기준으로 노래들을 정렬한다.
 - 장르의 정렬 기준 → 장르 내 속한 노래들의 총 재생 횟수를 기준으로 내림차순
 - 같은 장르 내 노래의 정렬 기준 → 각 노래의 재생 횟수를 기준으로 내림차순
 - 재생횟수가 같은 노래들의 정렬 기준 → 노래의 ID 기준으로 오름차순

문제 2. 문제 해결 전략 수립

• 어떤 자료구조가 가장 적합한가?

- 각 장르가 문자열로 주어짐.
- 각 장르에 대하여, <장르 - 총 재생 횟수> 및 <장르 - 소속된 노래 및 재생횟수> 데이터가 필요함.
- 따라서, "장르"를 키(key)로 갖는 해시 테이블(또는 딕셔너리)이 필요함.

• 해시 테이블 모델링

- 총 두 개의 해시 테이블이 필요함.
- 해시 테이블 1 → Key: 장르, Value: 총 재생 횟수
- 해시 테이블 2 → Key: 장르, Value: (노래 ID, 재생 횟수)

genres	plays	return
["classic", "pop", "classic", "classic", "pop"]	[500, 600, 150, 800, 2500]	[4, 1, 3, 0]

Key	Value
classic	1450
pop	3100

play_dict

Key	Value
classic	[(0, 500), (2, 150), (3, 800)]
pop	[(1, 600), (4, 2500)]

genres_dict

문제 2. 문제 해결 방법 기술

- 문제 해결 방법을 구체적으로 기술하면 다음과 같다.
 - ① 입력 리스트로부터 해시 테이블 1과 2 구축
 - **play_dict** → Key: 장르, Value: 총 재생 횟수
 - **genres_dict** → Key: 장르, Value: (노래 ID, 재생 횟수)
 - ② 구축된 **play_dict**을 이용하여 총 재생 횟수가 많은 순서대로 장르 리스트를 정렬
 - ③ 정렬된 장르 리스트 내 각각의 장르에 대하여 **genres_dict**을 참조하여 아래와 같이 장르 내 노래에서 가장 많이 재생된 두 곡을 선택
 - 장르 리스트의 노래 리스트를 재생 횟수를 기준으로 내림차순으로 정렬
 - 정렬된 노래 리스트에서 최대 2곡을 선택하여 선곡 리스트에 저장
 - ④ 선곡 리스트를 반환

문제 2. 구현 및 검증

테스트 케이스

```
# Test
genres = ["classic", "pop", "classic", "classic", "pop"]
plays = [500, 600, 150, 800, 2500]

answers = solution2(genres, plays)
print(answers)
```

```
def solution2(genres, plays):
    answer = []
    genres_dict = {} # 장르 별 노래 리스트(id, 재생 횟수) 저장
    play_dict = {}   # 장르 별 총 재생 횟수 저장

    # 1. 해시 테이블에 장르별 총 재생 횟수와 각 곡의 재생 횟수를 저장
    for i in range(len(genres)):
        genre = genres[i]
        play = plays[i]

        if genre not in genres_dict:
            genres_dict[genre] = []
            play_dict[genre] = 0
        genres_dict[genre].append((i, play))
        play_dict[genre] += play

    # 2. 총 재생 횟수가 많은 장르순으로 정렬
    sorted_genres = sorted(play_dict.items(), key = lambda x: x[1], reverse=True)

    # 3. 각 장르 내에서 노래를 재생 횟수 순으로 정렬해 최대 2곡까지 선택
    for genre, _ in sorted_genres:
        sorted_songs = sorted(genres_dict[genre], key=lambda x: (-x[1], x[0]))
        answer.extend([idx for idx, _ in sorted_songs[:2]])

    return answer
```

연습문제 3. 신고 결과 받기

문제 3. 신고 결과 받기 (1/5)

제한시간: 70분

• 문제설명

- 신입사원 무지는 게시판 불량 이용자를 신고하고 처리 결과를 메일로 발송하는 시스템을 개발하려 한다. 무지가 개발하려는 시스템은 다음과 같다.
- 각 유저는 한 번에 한 명의 유저를 신고할 수 있다.
 - 신고 횟수에 제한은 없다. 서로 다른 유저를 계속해서 신고할 수 있다.
 - 한 유저를 여러 번 신고할 수도 있지만, 동일한 유저에 대한 신고 횟수는 1회로 처리된다.
- k번 이상 신고된 유저는 게시판 이용이 정지되며, 해당 유저를 신고한 모든 유저에게 정지 사실을 메일로 발송한다.
 - 유저가 신고한 모든 내용을 취합하여 마지막에 한꺼번에 게시판 이용 정지를 시키면서 정지 메일을 발송한다.

문제 3. 신고 결과 받기 (2/5)

제한시간: 70분

- 아래 왼쪽 표는 전체 유저 목록이 ["muzi", "frodo", "apeach", "neo"]이고, $k = 2$ (즉, 2번 이상 신고당하면 이용 정지)인 경우의 예시이고, 오른쪽 표는 각 유저별로 신고당한 횟수이다.


유저 ID	유저가 신고한 ID	설명
"muzi"	"frodo"	"muzi"가 "frodo"를 신고했습니다.
"apeach"	"frodo"	"apeach"가 "frodo"를 신고했습니다.
"frodo"	"neo"	"frodo"가 "neo"를 신고했습니다.
"muzi"	"neo"	"muzi"가 "neo"를 신고했습니다.
"apeach"	"muzi"	"apeach"가 "muzi"를 신고했습니다.

유저 ID	신고당한 횟수
"muzi"	1
"frodo"	2
"apeach"	0
"neo"	2

문제 3. 신고 결과 받기 (3/5)

제한시간: 70분

- 아래 왼쪽 예시에서는 2번 이상 신고당한 "frodo"와 "neo"의 게시판 이용이 정지된다. 이때, 각 유저별로 신고한 아이디와 정지된 아이디를 정리하면 다음과 같다.
- 따라서 "muzi"는 처리 결과 메일을 2회, "frodo"와 "apeach"는 각각 처리 결과 메일을 1회 받는다.

유저 ID	신고당한 횟수		유저 ID	유저가 신고한 ID	정지된 ID
"muzi"	1		"muzi"	["frodo", "neo"]	["frodo", "neo"]
"frodo"	2		"frodo"	["neo"]	["neo"]
"apeach"	0		"apeach"	["muzi", "frodo"]	["frodo"]
"neo"	2		"neo"	없음	없음

- 이용자의 ID가 담긴 문자열 배열 id_list, 각 이용자가 신고한 이용자의 ID 정보가 담긴 문자열 배열 report, 정지 기준이 되는 신고 횟수 k가 매개변수로 주어질 때, 각 유저별로 처리 결과 메일을 받은 횟수를 배열에 담아 반환하도록 solution3 함수를 완성하시오.

문제 3. 신고 결과 받기 (4/5)

제한시간: 70분

• 제한사항

- $2 \leq \text{id_list의 길이} \leq 1,000$
 - $1 \leq \text{id_list의 원소 길이} \leq 10$
 - id_list의 원소는 이용자의 id를 나타내는 문자열이며 알파벳 소문자로만 이루어져 있다.
 - id_list에는 같은 아이디가 중복해서 들어있지 않다.
- $1 \leq \text{report의 길이} \leq 200,000$
 - $3 \leq \text{report의 원소 길이} \leq 21$
 - report의 원소는 "이용자id 신고한id"형태의 문자열이다.
 - 예를 들어 " muzi frodo " 의 경우 " muzi " 가 " frodo " 를 신고했다는 의미이다.
 - id는 알파벳 소문자로만 이루어져 있다.
 - 이용자id와 신고한id는 공백(스페이스)하나로 구분되어 있다.
 - 자기 자신을 신고하는 경우는 없다.
- $1 \leq k \leq 200$, k는 자연수이다.
- return 하는 배열은 id_list에 담긴 id 순서대로 각 유저가 받은 결과 메일 수를 담는다.

문제 3. 신고 결과 받기 (5/5)

제한시간: 70분

• 입출력 예제

id_list	report	k	result
["muzi", "frodo", "apeach", "neo"]	["muzi frodo", "apeach frodo", "frodo neo", "muzi neo", "apeach muzi"]	2	[2,1,1,0]
["con", "ryan"]	["ryan con", "ryan con", "ryan con", "ryan con"]	3	[0,0]

• 입출력 예제 설명

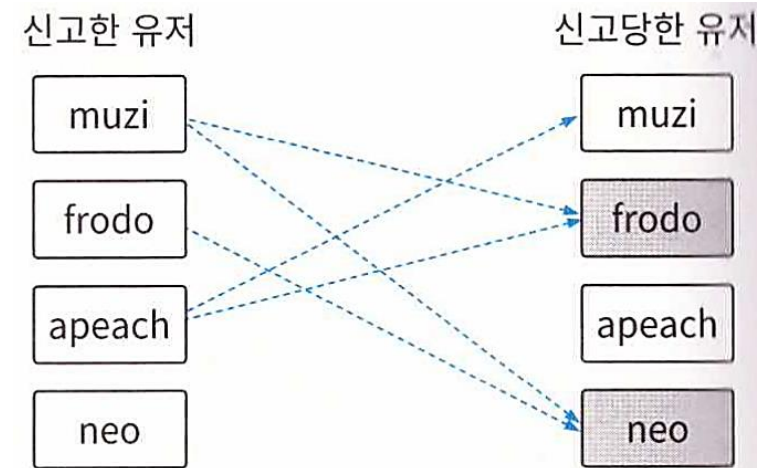
• 예제 1: 문제의 예시와 같다.

• 예제 2:

- "ryan"이 "con"을 4번 신고했으나, 주어진 조건에 따라 한 유저가 같은 유저를 여러 번 신고한 경우는 신고 횟수 1회로 처리한다. 따라서 "con"은 1회 신고당했다.
- 3번 이상 신고당한 이용자는 없으며, "con"과 "ryan"은 결과 메일을 받지 않는다. 따라서 [0, 0]을 return 한다.

문제 3. 문제 분석

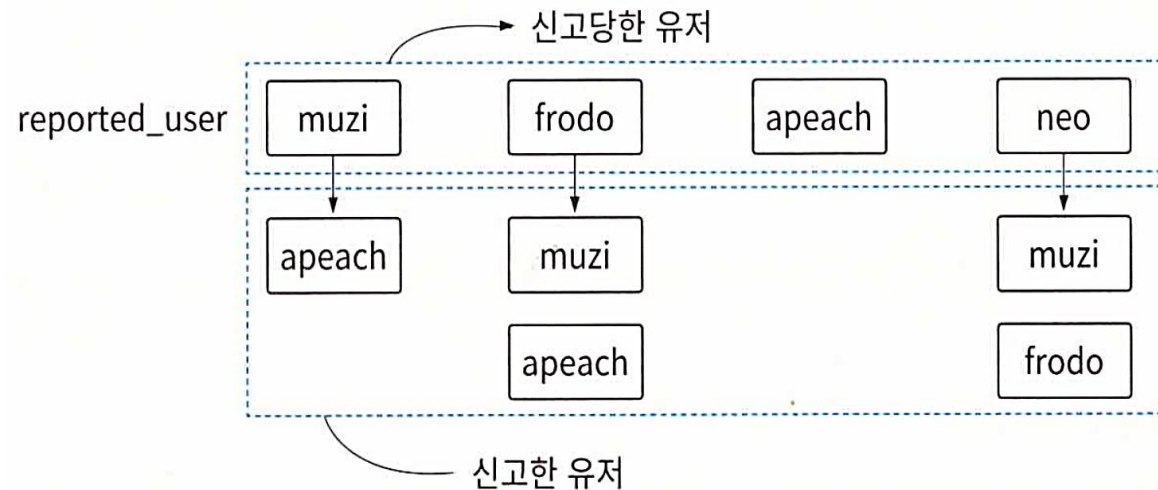
- 문제 해결을 위해 우리가 알아야 할 정보
 - ① 신고한 유저와 신고를 당한 유저의 관계
 - ② 각 유저 별 신고를 당한 횟수
 - ③ 각 유저 별 신고를 한 횟수
- 어떤 자료구조를 활용해야 하는가?
 - 신고한 유저와 신고를 당한 유저 모두 ID가 문자열 형태로 주어짐
 - 누가 어떤 유저를 신고했는지와 누가 몇 번 신고를 당했는가를 빠르게 파악해야 함
 - 즉, 입력 데이터로부터 "신고한 유저 – 신고당한 유저" 및 "신고당한 유저 – 신고당한 횟수" 관계를 파악하여 저장해야 함. (★ Key – Value 관계로 모델링 가능)
- 따라서, 해시 테이블을 이용하여 위 정보를 저장하고 활용할 수 있다.



문제 3. 문제 해결 전략 수립

• 문제 해결을 위한 자료구조 설계

	Name	Key	Value
신고를 당한 유저와 신고를 한 유저	reported_user	신고된 유저 ID	신고한 유저 ID 집합
각 유저 별 신고를 한 횟수	count	신고한 유저 ID	신고 횟수



- 딕셔너리 "**reported_user**"를 활용하면 신고를 당한 각 유저 별로 어떤 유저가 신고했는지, 신고를 당한 횟수를 확인할 수 있다.
- 그러나 우리의 목표는 "정지당한 유저가 있을 때 이를 신고한 유저에게 이메일을 보낸 횟수"를 반환하는 것
- 따라서 신고를 한 유저가, 자신이 신고한 유저 중 실제 k회 이상 신고 당한 유저는 몇 명인지를 나타내는 또 다른 딕셔너리 "**count**"를 활용한다.

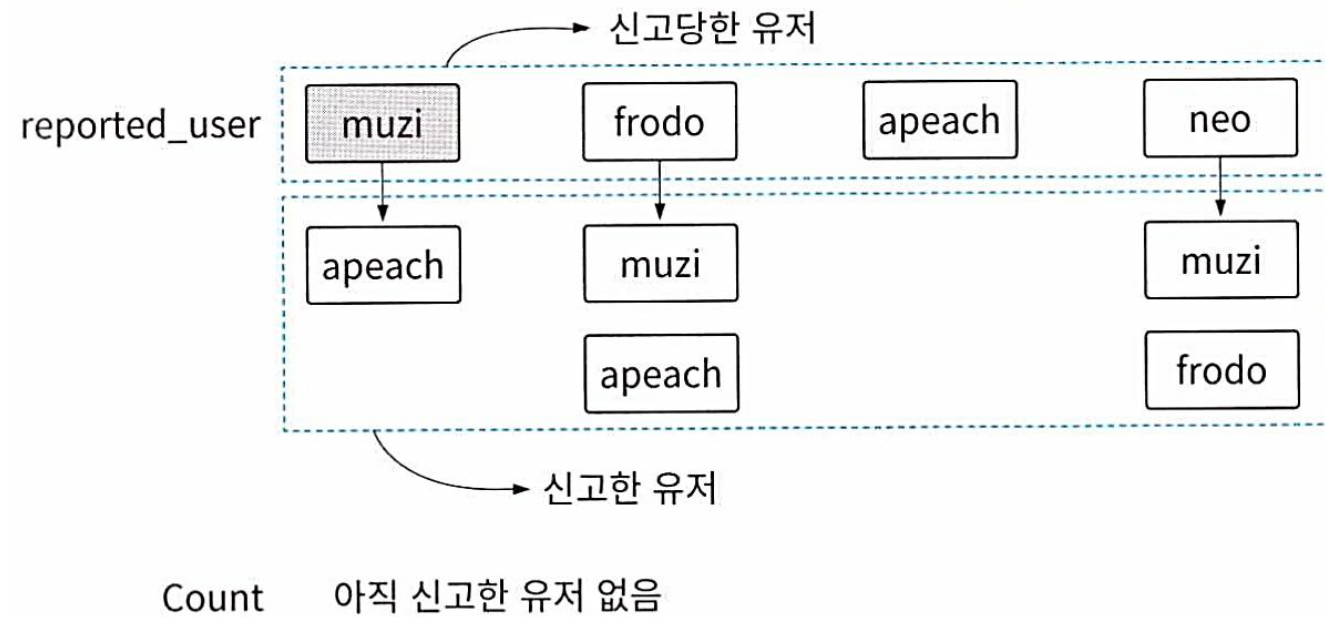
문제 3. 문제 해결 방법 기술

- ① 입력 리스트 report로부터 신고를 당한 유저와 신고를 한 유저에 대한 정보를 읽어들이고 후 "reported_user"에 Key와 Value로 저장한다.
 - Key: 신고를 당한 유저의 ID, Value: 신고를 한 유저의 ID
 - 이때, 한 유저에 대하여 여러 유저가 신고한 경우도 있으므로, Value는 집합(Set)으로 처리한다.
- ② 딕셔너리 "reported_user"를 순회하면서 신고를 당한 각 유저에 대해서 어떤 유저들이 몇 번 신고했는지 확인한다.
 - 신고를 한 유저가 k명 미만인 경우 메일 발송이 안되므로 카운트하지 않고 그냥 통과한다.
 - 신고를 한 유저가 k명 이상인 경우 메일 발송이 이루어지므로 딕셔너리 "count" 내 신고한 유저의 Value를 1 증가시킨다.
- ③ 딕셔너리 "count"를 참고하여 신고를 받는 사람들의 리스트를 반환한다.
 - 문제의 최종 목표는 신고를 수행한 유저별로 이메일을 받는 횟수를 리스트 형태로 반환하는 것.
 - 따라서 딕셔너리 "count"를 참고하여 각 유저 별 카운트 횟수(Value)를 리스트로 반환한다.

문제 3. 문제 해결 예제

• 1단계

- 신고를 당한 사람: **muzi**
- muzi를 신고한 유저는 apeach 1명이므로 아무것도 수행하지 않는다.

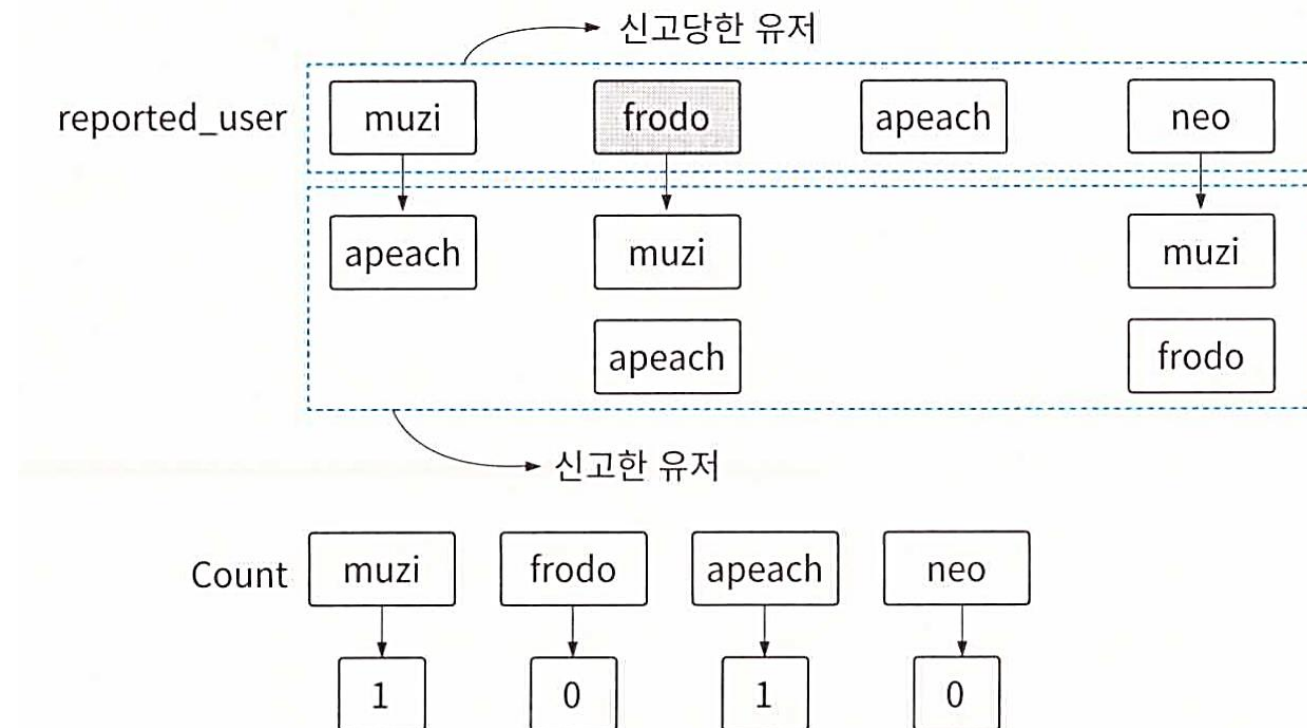


문제 3. 문제 해결 예제

• 2단계

➤ 신고를 당한 사람: **frodo**

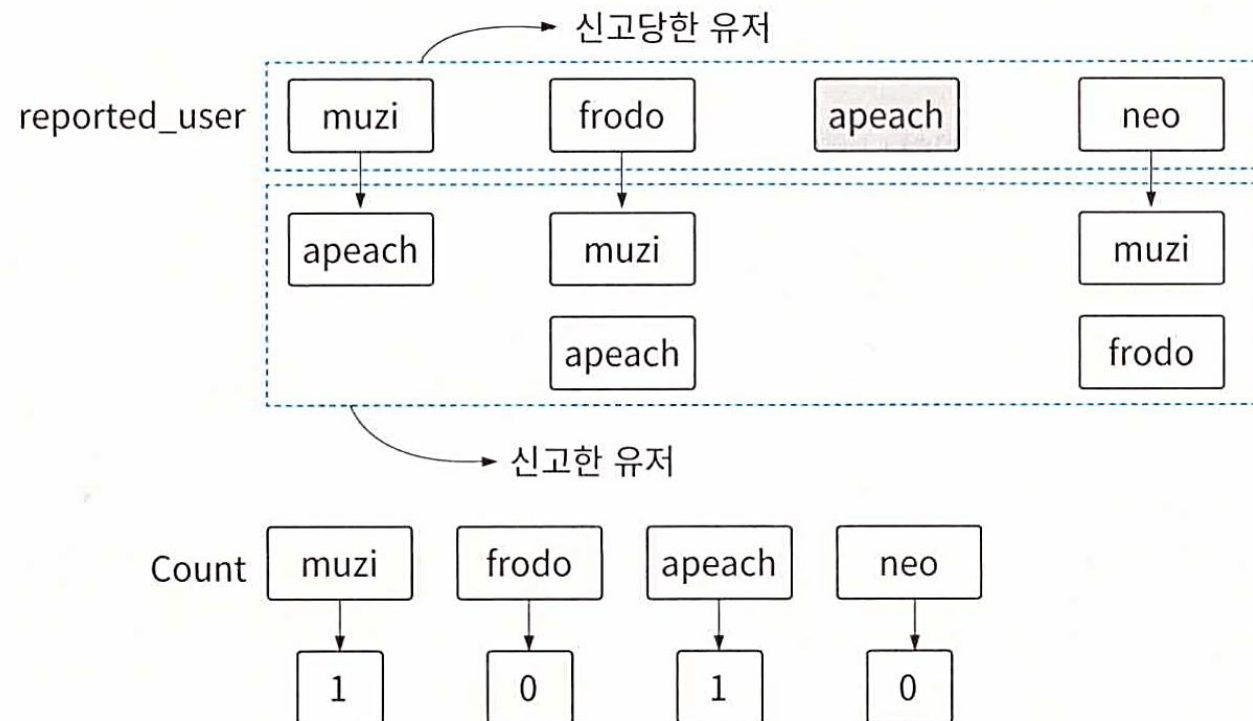
➤ frodo를 신고한 유저는 muzi와 apeach 2명이므로, 이들에 대한 count를 1 증가시킨다.



문제 3. 문제 해결 예제

• 3단계

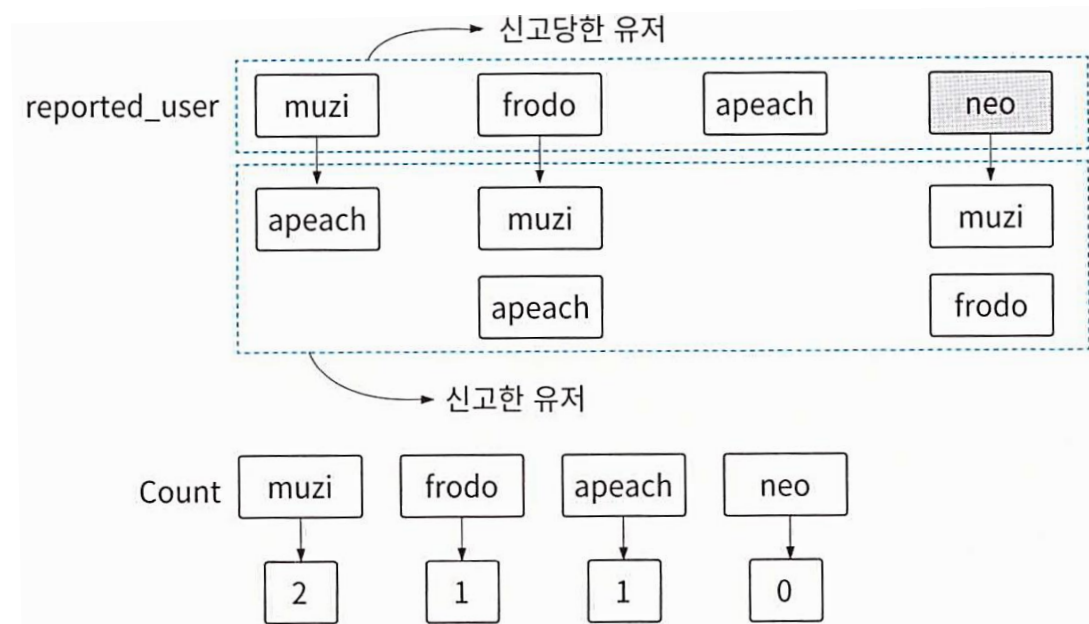
- 신고를 당한 사람: **apeach**
- apeach를 신고한 유저는 없으므로 아무것도 수행하지 않는다.



문제 3. 문제 해결 예제

• 4단계

- 신고를 당한 사람: **neo**
- neo를 신고한 유저는 muzi와 frodo 2명이므로, 이들에 대한 count를 1 증가시킨다.



- reported_user 내에 존재하는 신고 당한 유저들을 모두 확인하였으므로, 각 유저별로 신고 횟수를 저장한 리스트를 딕셔너리 count를 참조하여 반환한다.

문제 3. 구현 및 검증

입출력 예제

id_list	report	k	result
["muzi", "frodo", "apeach", "neo"]	["muzi frodo", "apeach frodo", "frodo neo", "muzi neo", "apeach muzi"]	2	[2,1,1,0]
["con", "ryan"]	["ryan con", "ryan con", "ryan con", "ryan con"]	3	[0,0]

테스트 케이스

```
# Test
id_list = ["muzi", "frodo", "apeach", "neo"]
report = ["muzi frodo", "apeach frodo", "frodo neo", "muzi neo", "apeach muzi"]
k = 2

result = solution3(id_list, report, k)
print(result)
```



```
def solution3(id_list, report, k):
    reported_user = {}
    count = {}
    # 유저들의 신고 기록을 순회하면서 해시테이블 구축
    for r in report:
        user_id, reported_id = r.split()
        if reported_id not in reported_user: # 신고당한 기록이 없는 경우
            reported_user[reported_id] = set()
        # 신고한 사람의 아이디를 집합에 담아 딕셔너리에 저장
        reported_user[reported_id].add(user_id)

    for reported_id, user_id_lst in reported_user.items():
        if len(user_id_lst) >= k: # 신고한 유저의 수를 확인(k명 이상이면 정지당함)
            for uid in user_id_lst: # 딕셔너리를 순회하며 count 계산
                if uid not in count:
                    count[uid] = 1
                else:
                    count[uid] += 1

    answer = []

    # 각 아이디별 메일을 받은 횟수를 순서대로 정리
    for i in range(len(id_list)):
        if id_list[i] not in count:
            answer.append(0)
        else:
            answer.append(count[id_list[i]])

    return answer
```

연습문제 4. 오픈채팅방

문제 4. 오픈채팅방 (1/5)

제한시간: 50분

- 카카오톡 오픈채팅방에서는 친구가 아닌 사람들과 대화를 할 수 있는데, 본래 닉네임이 아닌 가상의 닉네임을 사용하여 채팅방에 들어갈 수 있다.
- 신입사원인 김크루는 카카오톡 오픈 채팅방을 개설한 사람을 위해, 다양한 사람들이 들어오고, 나가는 것을 지켜볼 수 있는 관리자창을 만들기로 했다. 채팅방에 누군가 들어오면 다음 메시지가 출력된다.
"[닉네임]님이 들어왔습니다."
- 채팅방에서 누군가 나가면 다음 메시지가 출력된다.
"[닉네임]님이 나갔습니다."
- 채팅방에서 닉네임을 변경하는 방법은 다음과 같이 두 가지이다.
 - ① 채팅방을 나간 후, 새로운 닉네임으로 다시 들어간다.
 - ② 채팅방에서 닉네임을 변경한다.
- 닉네임을 변경할 때는 기존에 채팅방에 출력되어 있던 메시지의 닉네임도 전부 변경된다.

문제 4. 오픈채팅방 (2/5)

제한시간: 50분

- 예를 들어, 채팅방에 "Muzi"와 "Prodo"라는 닉네임을 사용하는 사람이 순서대로 들어오면 채팅방에는 다음과 같이 메시지가 출력된다.

"Muzi님이 들어왔습니다."

"Prodo님이 들어왔습니다."

- 채팅방에 있던 사람이 나가면 채팅방에는 다음과 같이 메시지가 남는다.

"Muzi님이 들어왔습니다."

"Prodo님이 들어왔습니다."

"Muzi님이 나갔습니다."

- Muzi가 나간후 다시 들어올 때, Prodo 라는 닉네임으로 들어올 경우 기존에 채팅방에 남아있던 Muzi도 Prodo로 다음과 같이 변경된다.

"Prodo님이 들어왔습니다."

"Prodo님이 들어왔습니다."

"Prodo님이 나갔습니다."

"Prodo님이 들어왔습니다."

문제 4. 오픈채팅방 (3/5)

제한시간: 50분

- 채팅방은 중복 닉네임을 허용하기 때문에, 현재 채팅방에는 Prodo라는 닉네임을 사용하는 사람이 두 명이 있다. 이제, 채팅방에 두 번째로 들어왔던 Prodo가 Ryan으로 닉네임을 변경하면 채팅방 메시지는 다음과 같이 변경된다.

"Prodo님이 들어왔습니다."

"Ryan님이 들어왔습니다."

"Prodo님이 나갔습니다."

"Prodo님이 들어왔습니다."

- 채팅방에 들어오고 나가거나, 닉네임을 변경한 기록이 담긴 문자열 배열 record가 매개변수로 주어질 때, 모든 기록이 처리된 후, 최종적으로 방을 개설한 사람이 보게 되는 메시지를 문자열 배열 형태로 return 하도록 **solution4** 함수를 완성하시오.

문제 4. 오픈채팅방 (4/5)

제한시간: 50분

• 제한사항

- record는 다음과 같은 문자열이 담긴 배열이며, 길이는 1 이상 100,000 이하이다.
- 다음은 record에 담긴 문자열에 대한 설명이다.
 - 모든 유저는 [유저 아이디]로 구분한다.
 - [유저 아이디] 사용자가 [닉네임]으로 채팅방에 입장 - "Enter [유저 아이디] [닉네임]"
(ex. "Enter uid1234 Muzi")
 - [유저 아이디] 사용자가 채팅방에서 퇴장 - "Leave [유저 아이디]"
(ex. "Leave uid1234")
 - [유저 아이디] 사용자가 닉네임을 [닉네임]으로 변경 - "Change [유저 아이디] [닉네임]"
(ex. "Change uid1234 Muzi")
 - 첫 단어는 Enter, Leave, Change 중 하나이다.
 - 각 단어는 공백으로 구분되어 있으며, 알파벳 대문자, 소문자, 숫자로만 이루어져있다.
 - 유저 아이디와 닉네임은 알파벳 대문자, 소문자를 구별한다.
 - 유저 아이디와 닉네임의 길이는 1 이상 10 이하이다.
 - 채팅방에서 나간 유저가 닉네임을 변경하는 등 잘못 된 입력은 주어지지 않는다.

문제 4. 오픈채팅방 (5/5)

제한시간: 50분

- 입출력 예제

입력: record

["Enter uid1234 Muzi", "Enter uid4567 Prodo", "Leave uid1234", "Enter uid1234 Prodo",
"Change uid4567 Ryan"]

출력: result

["Prodo님이 들어왔습니다.", "Ryan님이 들어왔습니다.", "Prodo님이 나갔습니다.",
"Prodo님이 들어왔습니다."]

문제 4. 문제 분석

- 문제의 목표 및 분석

- 방 개설자 기준으로 최종으로 채팅방에서 보일 메시지를 반환하는 문제

- 문제로부터 파악할 수 있는 정보

- 채팅방 내 유저의 식별: 유저의 아이디, 닉네임
- 채팅방 내 유저의 동작 정보:
 - ① **Enter**: 새로운 유저가 채팅방에 들어옴
 - ② **Leave**: 채팅방 내 유저가 채팅방에서 나감
 - ③ **Change**: 채팅방 내 유저가 자신의 닉네임을 변경함
- 채팅방에서 각 유저는 "닉네임"으로만 표기되며 중간에 변동될 수 있음
- 유저의 아이디는 변동되지 않음

문제 4. 문제 해결 전략 수립

• 문제 해결을 위한 자료구조 설계

- 유저의 아이디와 닉네임 관계 (즉, 아이디 – 닉네임)
 - 채팅방에는 닉네임만 등장함
 - 입력 레코드에는 유저의 아이디를 사용함
 - 단, 아이디는 고정된 값이고 닉네임은 중간에 변동될 수 있음

해시테이블 구축

- Key: **아이디**
- Value: **닉네임**

- 특정 유저가 현재 채팅방에 존재하는가 여부
 - 채팅방에 처음 들어올 때(Enter) 해시 테이블에 아이디와 닉네임을 추가함
 - 유저가 채팅방에서 나가면(Leave) 별도의 작업을 수행하지 않음
 - "Change" 행동의 경우 해시 테이블 내 닉네임만 변경하면 됨

별도의 해시테이블을
구축할 필요 없음

문제 4. 문제 해결 전략 수립

• 문제 해결 아이디어

- 입력 리스트 내 값의 구조: 동작 아이디 닉네임
 - 출력되는 내용: 입력 리스트와 일대일로 대응되어 출력
-
- 입력 리스트에서 문자열을 하나씩 읽어들이 때마다 아래의 작업을 수행한다.
 - 입력 문자열로부터 동작정보와 유저의 아이디, 닉네임을 추출한다.
 - 어떤 종류의 동작인지 파악하고, 동작의 종류에 따라 아래의 작업 중 하나를 수행한다.
 - ① **Enter**: 유저의 아이디와 닉네임을 해시 테이블에 추가함.
 - ② **Leave**: 유저가 채팅방에서 떠났다는 의미이므로 아무 작업도 수행하지 않음
 - ③ **Change**: 해시 테이블 내에서 해당 아이디를 갖는 유저의 닉네임을 변경함

입력: record

```
["Enter uid1234 Muzi", "Enter uid4567 Prodo", "Leave uid1234", "Enter uid1234 Prodo",  
"Change uid4567 Ryan"]
```

출력: result

```
["Prodo님이 들어왔습니다.", "Ryan님이 들어왔습니다.", "Prodo님이 나갔습니다.",  
"Prodo님이 들어왔습니다."]
```

문제 4. 문제 해결 전략 수립

• 문제 해결 전략 수립 시 유의사항

- ① 닉네임의 중복이 허용됨
 - 해시 테이블의 Key는 유저의 아이디, Value는 닉네임이 되어야 함
- ② 닉네임을 중간에 변경할 수 있음
 - 닉네임을 중간에 변경하면 이전에 출력된 닉네임은 어떻게 처리해야 하는가?
 - 이에 대한 조건이 문제에 존재함 → “닉네임을 변경할 때는 기존에 채팅방에 출력되어 있던 메시지의 닉네임도 전부 변경됨”
- ③ 출력 메시지의 종류에 유의
 - “Change” 동작의 경우 별도의 출력 메시지가 없음에 유의

문제 4. 문제 해결 전략 수립

- ④ 닉네임을 변경하는 방법이 두 가지임에 유의
- 특히, 채팅방을 나간 후("Leave") 다른 닉네임으로 다시 들어오는 경우
 - **파이썬 딕셔너리**를 사용하는 경우 채팅방에 새로 들어오는 동작("Enter")과 동일하게 구현할 수 있음
 - 단, 파이썬의 딕셔너리가 아닌 **C++, Java 해시 테이블**을 사용하는 경우 해시 테이블에 해당 유저의 아이디(Key)가 존재하는지 우선 확인한 후, 존재한다면 기존의 Value를 변경하는 코드로 구현할 수 있음

문제 4. 문제 해결 방법 기술

- ① 딕셔너리 uid를 초기화한다.
- ② 입력 리스트를 순회하면서 딕셔너리 uid를 구축한다.
 - 입력 리스트에서 문자열 s를 읽어들인다.
 - 문자열 s를 공백을 기준으로 분리하여 동작정보, 아이디, 닉네임을 추출한다.
 - 동작 정보가 "Enter"인 경우 <아이디, 닉네임>을 딕셔너리 uid에 추가한다.
 - 동작 정보가 "Leave"인 경우 아무 작업도 수행하지 않는다.
 - 동작 정보가 "Change"인 경우 딕셔너리 uid에서 주어진 유저의 아이디를 Key로 갖는 Value를 변경한다.
- ③ 입력 리스트 record의 값을 순회하면서 양식에 따라 출력 메시지를 생성하고 이를 출력 리스트 result에 추가한다.
 - 동작 정보가 "Enter"인 경우 출력 메시지 양식 → "**uid[아이디]**님이 들어왔습니다."
 - 동작 정보가 "Leave"인 경우 출력 메시지 양식 → "**uid[아이디]**님이 나갔습니다."

문제 4. 문제 해결 예제

• 1단계

- 입출력에 존재하는 record의 초기 구조는 오른쪽 그림과 같다.

record		
Enter	uid1234	Muzi
Enter	uid4567	Prodo
Leave	uid1234	
Enter	uid1234	Prodo
Change	uid4567	Ryan

• 2단계

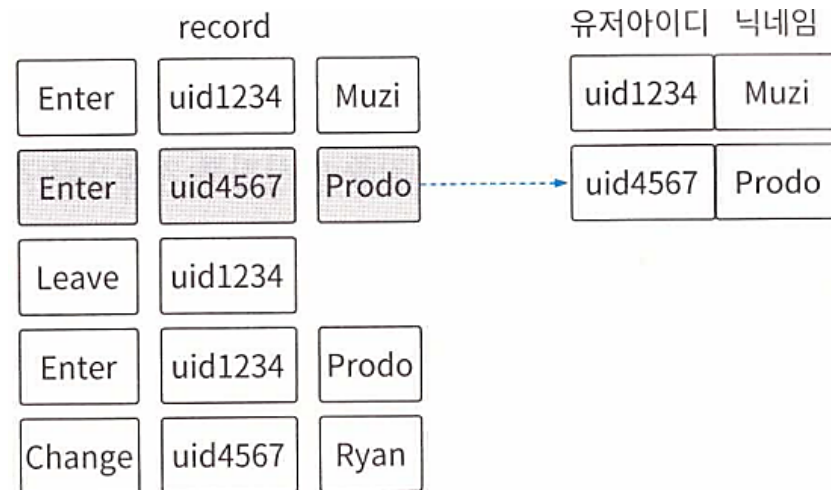
- 첫 입력이 "Enter"이므로 유저 아이디와 닉네임을 딕셔너리에 저장한다.
- 유저 "uid1234"는 새로 추가되는 유저이므로 딕셔너리에 그대로 추가한다.

record			유저아이디	닉네임
Enter	uid1234	Muzi	uid1234	Muzi
Enter	uid4567	Prodo		
Leave	uid1234			
Enter	uid1234	Prodo		
Change	uid4567	Ryan		

문제 4. 문제 해결 예제

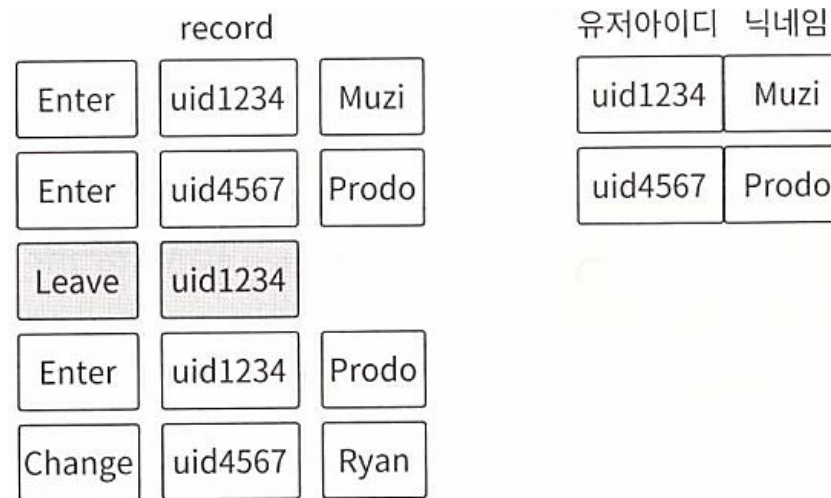
• 3단계

- 첫 입력이 "Enter"이므로 유저 아이디와 닉네임을 딕셔너리에 저장해야 한다.
- 유저 "uid4567"은 기존에 딕셔너리에 존재하지 않는 새로운 유저이므로 딕셔너리에 새롭게 추가한다.



• 4단계

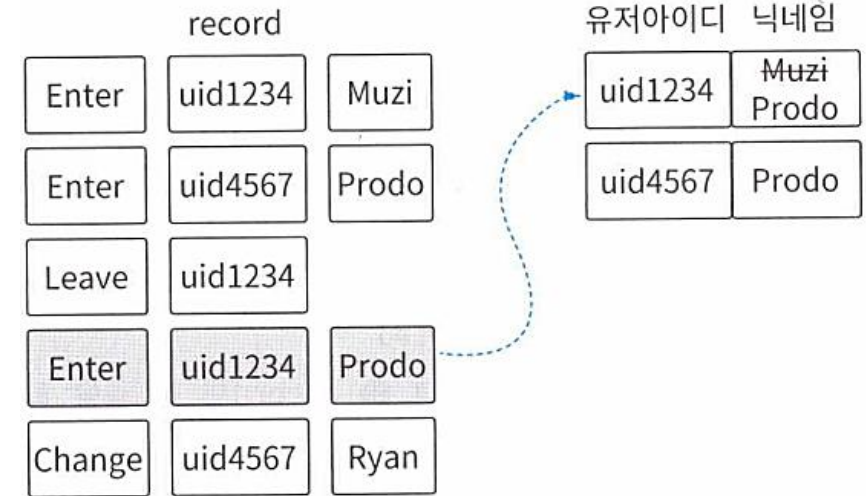
- 첫 입력이 "Leave"이다.
- 이 경우, 실제 메시지를 수정하는 것이 아니라 유저 아이디에 대한 닉네임만 수정하는 작업이다.
- 따라서, "Leave"에서는 딕셔너리에 별도의 데이터를 추가하지 않는다.



문제 4. 문제 해결 예제

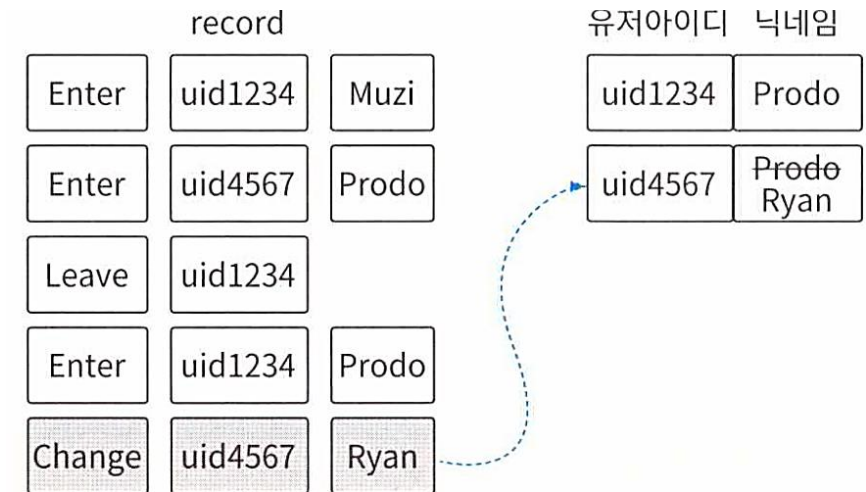
• 5단계

- 첫 입력이 "Enter"이므로 유저 아이디와 닉네임을 딕셔너리에 저장해야 한다.
- 그러나, 지금 저장하려는 아이디 "uid1234"는 이미 딕셔너리에 존재하는 아이디이므로, 딕셔너리에서 해당 아이디의 닉네임만 수정한다.



• 6단계

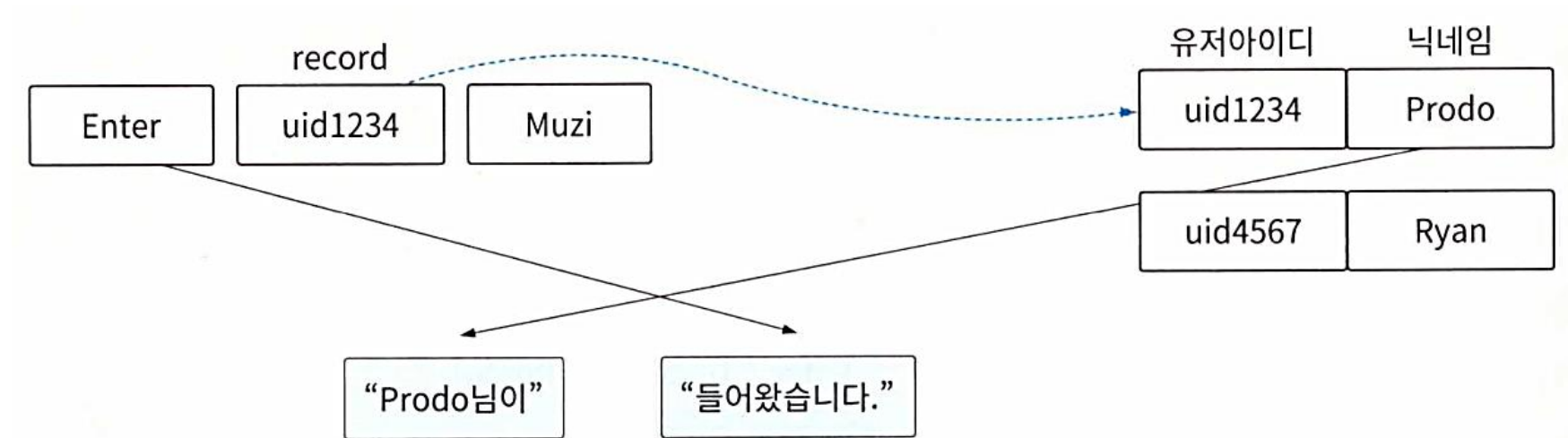
- 첫 입력이 "Change"이므로 유저 아이디 "uid4567"은 이미 딕셔너리에 존재하는 상황이다.
- 따라서 딕셔너리에서 유저 "uid4567"의 닉네임만 수정한다.



문제 4. 문제 해결 예제

• 7단계

- 앞 단계에서 구축된 딕셔너리와 기존의 record를 조합하여 출력 메시지 양식을 구성하고, 이를 출력한다.



문제 4. 구현 및 검증

```
def solution4(record):  
    answer = []  
    uid = {}  
    for line in record:  
        cmd = line.split(" ")  
        if cmd[0] != "Leave":  
            uid[cmd[1]] = cmd[2]  
  
    for line in record:  
        cmd = line.split(" ")  
        if cmd[0] == "Enter":  
            answer.append("%s님이 들어왔습니다." % uid[cmd[1]])  
        elif cmd[0] == "Change":  
            pass  
        else:  
            answer.append("%s님이 나갔습니다." % uid[cmd[1]])  
  
    return answer
```

문제 4. 구현 및 검증

테스트 케이스

```
record = ["Enter uid1234 Muzi", "Enter uid4567 Prodo", "Leave uid1234",  
          "Enter uid1234 Prodo", "Change uid4567 Ryan"]  
answer = solution4(record)  
print(answer)
```

['Prodo님이 들어왔습니다.', 'Ryan님이 들어왔습니다.', 'Prodo님이 나갔습니다.', 'Prodo님이 들어왔습니다.']

Q & A