Week 1. 알고리즘 스터디 - 해쉬(1)

팀장 : 라효진

📌 프로그래머스

1. 1. 완주하지 못한 선수

문제

수많은 마라톤 선수들이 마라톤에 참여하였습니다. 단 한 명의 선수를 제외하고는 모든 선수가 마라톤을 완주하였습니다.

마라톤에 참여한 선수들의 이름이 담긴 배열 participant와 완주한 선수들의 이름이 담긴 배열 completion이 주어질 때, 완주하지 못한 선수의 이름을 return 하도록 solution 함수를 작성해주세요.

- **입력**
  - participant: 마라톤에 참여한 선수들의 이름이 담긴 배열
  - completion: 완주한 선수들의 이름이 담긴 배열

- **출력**
  - 완주하지 못한 선수의 이름을 return 하도록 solution 함수

제한사항

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

예시

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"

예시 설명

- 예제 #1
  - "leo"는 참여자 명단에는 있지만, 완주자 명단에는 없기 때문에 완주하지 못했습니다.
- 예제 #2
  - "vinko"는 참여자 명단에는 있지만, 완주자 명단에는 없기 때문에 완주하지 못했습니다.
- 예제 #3

- "mislav"는 참여자 명단에는 두 명이 있지만, 완주자 명단에는 한 명밖에 없기 때문에 한명은 완주하지 못했습니다.

구현 아이디어

1. 문제에서 참여자와 완주자 명단을 이용해서 완주하지 못한 1인 찾기
2. 동명이인 고려 ⇒ 자료구조를 dict으로 key : 이름, value : 인원 수
3. 해당 자료구조로 참여 인원 증가, 완주 인원 차감
4. 결과출력에서 value가 1인 참여자 이름을 반환

코드

```
from collections import defaultdict

def solution(participant, completion):
    # 1. participant 관련 counter 딕션너리 정의
    counter = defaultdict(int)

    for p in participant:
        counter[p] += 1

    # 2. completion 선수들을 조회하면서 counter 카운터 차감
    for p in completion:
        counter[p] -= 1

    # 3. 결과 출력
    answerd = [str(k) for k, v in counter.items() if v == 1]
    return answerd[0]
```

문법 정리

- **defaultdict(default_factory=None)**
  - 새로운 사전 객체를 반환
  - default_factory : 속성의 초기 값, 기본값은 None

```
class collections.defaultdict(default_factory=None /[, ...])
```

- 예제 1. List

```
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
d = defaultdict(list)
for k, v in s:
    d[k].append(v)

sorted(d.items())

# 결과
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

- 예제 2. 문자열

```
s = 'mississippi'
d = defaultdict(int)
for k in s:
    d[k] += 1

sorted(d.items())
```

```
# 결과
[('i', 4), ('m', 1), ('p', 2), ('s', 4)]
```

- 예제 3. Set

```
s = [('red', 1), ('blue', 2), ('red', 3), ('blue', 4), ('red', 1), ('blue', 4)]
d = defaultdict(set)

for k, vin s:
  d[k].add(v)

sorted(d.items())

# 결과
[('blue', {2, 4}), ('red', {1, 3})]
```

---

1. 2. 전화번호 목록

문제

전화번호부에 적힌 전화번호 중, 한 번호가 다른 번호의 접두어인 경우가 있는지 확인하려 합니다.전화번호가 다음과 같을 경우, 구조대 전화번호는 영석이의 전화번호의 접두사입니다.

- 구조대 : 119
- 박준영 : 97 674 223
- 지영석 : 11 9552 4421

전화번호부에 적힌 전화번호를 담은 배열 phone_book 이 solution 함수의 매개변수로 주어질 때, 어떤 번호가 다른 번호의 접두어인 경우가 있으면 false를 그렇지 않으면 true를 return 하도록 solution 함수를 작성해주세요.

- **입력**
  - phone_book : 전화번호부에 적힌 전화번호를 담은 배열
- **출력**
  - bool
    - true : 접두어 존재 X
    - false : 접두어 존재 O

제한사항

- phone_book의 길이는 1 이상 1,000,000 이하입니다.
  - 각 전화번호의 길이는 1 이상 20 이하입니다.
  - 같은 전화번호가 중복해서 들어있지 않습니다.

예시

phone_book	return

["119", "97674223", "1195524421"]	false
["123","456","789"]	true
["12","123","1235","567","88"]	false

예시 설명

- 예제 #1
  - 앞에서 설명한 예와 같습니다.
- 예제 #2
  - 한 번호가 다른 번호의 접두사인 경우가 없으므로, 답은 true입니다.
- 예제 #3
  - 첫 번째 전화번호, “12”가 두 번째 전화번호 “123”의 접두사입니다. 따라서 답은 false입니다.

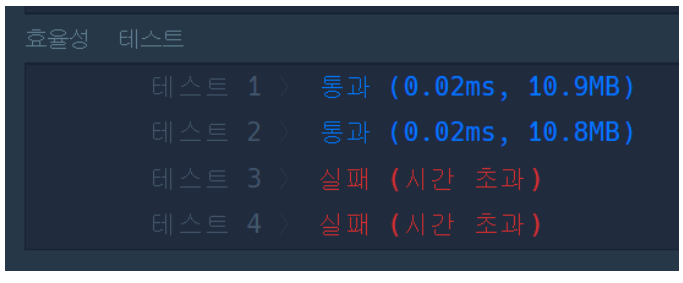
구현 아이디어

1. 리스트 데이터 두개를 비교하여 접두사가 있는지 확인한다.

실패 코드

```
# 실패 코드
def solution(phone_book):

    for i in range(len(phone_book)):
        for j in range(i + 1, len(phone_book)):
            if phone_book[i] == phone_book[j][0:len(phone_book[i])]
               or phone_book[j] == phone_book[i][0:len(phone_book[j])]:
                return False
    return True
```



개선된 코드

```
# 개선된 코드
def solution(phone_book):
    phone_book = sorted(phone_book)

    for p1, p2 in zip(phone_book, phone_book[1:]):
        if p2.startswith(p1):
            return False
    return True
```

다른 분 코드

```
from collections import defaultdict
```

```
def solution(phone_book):
    hash_map = defaultdict()

    for pb in phone_book:
         hash_map[pb]=  1

    for pb in hash_map.keys():
        text = ""
        for p in pb:
            text += p
            if text in hash_map and text != pb:
                return False
    return True
```

문법 정리

- **str.startswith(str, beg=0, end=len(string))**
  - STR : 비교 문자열
  - strbeg : 선택적 매개 변수는 문자열 검색의 시작 위치를 설정
  - strend : 선택적 매개 변수는 문자열의 끝 위치 감지를 설정

- **zip(* iterables , strict = False)**
  - 기본적으로 zip은 가장 짧은 반복 가능 항복이 소진되면 중지된다. 긴 iterable의 나머지 항목을 무시하고 결과를 가장 짧은 iterable의 길이로 잘라낸다.
  - iterable한 두 객체가 길이가 동일할 때만 해당 로직을 진행하고 싶으면 strict = True를 설정하게 되면 ValueError가 발생한다. strict=True인수가 없으면 길이가 다른 이터러블을 생성하는 모든 버그가 침묵하게 되어 프로그램의 다른 부분에서 찾기 힘든 버그로 나타날 수 있습니다.

- 예제 1.

```
list(zip(range(3), ['fee', 'fi', 'fo', 'fum']))

# 결과
[(0, 'fee'), (1, 'fi'), (2, 'fo')]
```

- 예제 2.

```
list(zip(range(3), ['fee', 'fi', 'fo', 'fum'], strict=True))

Traceback (most recent call last):
  ...
ValueError: zip() argument 2 is longer than argument 1
```

---

1. 3. 위장

문제

스파이들은 매일 다른 옷을 조합하여 입어 자신을 위장합니다.

예를 들어 스파이가 가진 옷이 아래와 같고 오늘 스파이가 동그란 안경, 긴 코트, 파란색 티셔츠를 입었다면 다음날은 청바지를 추가로 입거나 동그란 안경 대신 검정 선글라스를 착용하거나 해야 합니다.

종류	이름
얼굴	동그란 안경, 검정 선글라스
상의	파란색 티셔츠
하의	청바지
겉옷	긴 코트

스파이가 가진 의상들이 담긴 2차원 배열 clothes가 주어질 때 서로 다른 옷의 조합의 수를 return 하도록 solution 함수를 작성해주세요.

- **입력**
  - clothes : 스파이가 가진 의상들이 담긴 2차원 배열

- **출력**
  - 서로 다른 옷의 조합의 수

제한사항

- clothes의 각 행은 [의상의 이름, 의상의 종류]로 이루어져 있습니다.
- 스파이가 가진 의상의 수는 1개 이상 30개 이하입니다.
- 같은 이름을 가진 의상은 존재하지 않습니다.
- clothes의 모든 원소는 문자열로 이루어져 있습니다.
- 모든 문자열의 길이는 1 이상 20 이하인 자연수이고 알파벳 소문자 또는 '_' 로만 이루어져 있습니다.
- 스파이는 하루에 최소 한 개의 의상은 입습니다.

예시

clothes	return
[["yellowhat", "headgear"], ["bluesunglasses", "eyewear"], ["green_turban", "headgear"]]	5
[["crowmask", "face"], ["bluesunglasses", "face"], ["smoky_makeup", "face"]]	3

예시 설명

- 예제 #1
  - headgear에 해당하는 의상이 yellow_hat, green_turban이고 eyewear에 해당하는 의상이 blue_sunglasses이므로 아래와 같이 5개의 조합이 가능합니다.

```
1. yellow_hat
2. blue_sunglasses
3. green_turban
4. yellow_hat + blue_sunglasses
5. green_turban + blue_sunglasses
```

- 예제 #2
  - face에 해당하는 의상이 crow_mask, blue_sunglasses, smoky_makeup이므로 아래와 같이 3개의 조합이 가능합니다.

```
1. crow_mask
2. blue_sunglasses
3. smoky_makeup
```

구현 아이디어

1. 조합 개수의 합을 찾는 것
2. 전체 옷의 수 + 1(없음)을 하여 곱으로 전체 가지 수 찾음

코드

```
from collections import defaultdict

def solution(clothes):
    answer = 1
    my_clothe = defaultdict(int)

    for v, k in clothes:
        my_clothe[k] += 1

    for k, v in my_clothe.items():
        my_clothe[k] += 1
        answer *= my_clothe[k]

    return answer - 1
```

---

1. 4. 베스트 앨범

문제

스트리밍 사이트에서 장르 별로 가장 많이 재생된 노래를 두 개씩 모아 베스트 앨범을 출시하려 합니다. 노래는 고유 번호로 구분하며, 노래를 수록하는 기준은 다음과 같습니다.

1. 속한 노래가 많이 재생된 장르를 먼저 수록합니다.
2. 장르 내에서 많이 재생된 노래를 먼저 수록합니다.
3. 장르 내에서 재생 횟수가 같은 노래 중에서는 고유 번호가 낮은 노래를 먼저 수록합니다.

노래의 장르를 나타내는 문자열 배열 genres와 노래별 재생 횟수를 나타내는 정수 배열 plays가 주어질 때, 베스트 앨범에 들어갈 노래의 고유 번호를 순서대로 return 하도록 solution 함수를 완성하세요.

- **입력**
  - genres : 노래의 장르를 나타내는 문자열 배열
  - plays : 노래별 재생 횟수를 나타내는 정수 배열
- **출력**
  - 베스트 앨범에 들어갈 노래의 고유 번호를 순서

제한사항

- genres[i]는 고유번호가 i인 노래의 장르입니다.
- plays[i]는 고유번호가 i인 노래가 재생된 횟수입니다.
- genres와 plays의 길이는 같으며, 이는 1 이상 10,000 이하입니다.
- 장르 종류는 100개 미만입니다.
- 장르에 속한 곡이 하나라면, 하나의 곡만 선택합니다.
- 모든 장르는 재생된 횟수가 다릅니다.

예시

genres	plays
["classic", "pop", "classic", "classic", "pop"]	[500, 600, 150, 800, 2500]

예시 설명

- 예제 #1
  - classic 장르는 1,450회 재생되었으며, classic 노래는 다음과 같습니다.
    - 고유 번호 3: 800회 재생
    - 고유 번호 0: 500회 재생
    - 고유 번호 2: 150회 재생
  - pop 장르는 3,100회 재생되었으며, pop 노래는 다음과 같습니다.
    - 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 수록합니다.

구현 아이디어

1. { key : [(idx, value), (idx, value), (idx, value)]} 형태의 자료구조를 택함
2. value의 합으로 정렬
3. 하나의 key안의 리스트의 값들을 value 기준으로 정렬
4. 정렬한 후 첫 리스트 item을 출력
5. 이후 2개 이상일 경우 item 하나 더 출력

코드

```
from collections import defaultdict

def solution(genres, plays):
    # genres[i] : 고유 번호가 i인 노래 장르
    # plays[i] : 고유 번호가 i인 노래가 재생된 횟수
    answer = []

    playlist = defaultdict(list)
    for i, (g, p) in enumerate(zip(genres, plays)):
        playlist[g].append((i, p))

    playlist = dict(sorted(playlist.items(), key=lambda x: sum([_x[1] for _x in x[1]]), reverse=True))

    for key, value in playlist.items():
        value = sorted(value, key=lambda x: x[1], reverse=True)

        answer.append(value[0][0])

        if len(value) >= 2:
            answer.append(value[1][0])

    return answer
```

문법 정리

- **sorted(iterable, */*, *, key=None, reverse=False)**
  - 새로운 사전 객체를 반환
  - key : *iterable의* 각 요소에서 비교 키를 추출하는 데 사용되는 하나의 인수 함수

    - reverse *:* bool값으로 내림차순의 유무를 선택

- 예제 1.

```
str_list = ['abc','baa','abcde','efgh']
print(sorted(str_list, key=len))  # 함수

# 결과
['abc', 'baa', 'efgh', 'abcde']
```

- 예제 2.

```
str_list = ['abc','baa','abcde','efgh']
print(sorted(str_list, key=lambda x : -x[1]))

# 결과
['baa', 'abc', 'abcde', 'efgh']
```

---

1. 5. 소수 만들기

문제

주어진 숫자 중 3개의 수를 더했을 때 소수가 되는 경우의 개수를 구하려고 합니다. 숫자들이 들어있는 배열 nums가 매개변수로 주어질 때, nums에 있는 숫자들 중 서로 다른 3개를 골라 더했을 때 소수가 되는 경우의 개수를 return 하도록 solution 함수를 완성해주세요.

- **입력**
    - nums : 숫자들이 들어있는 배열

- **출력**
    - 소수가 되는 경우의 개수

제한사항

- nums에 들어있는 숫자의 개수는 3개 이상 50개 이하입니다.
- nums의 각 원소는 1 이상 1,000 이하의 자연수이며, 중복된 숫자가 들어있지 않습니다.

예시

nums	result
[1,2,3,4]	1
[1,2,7,6,4]	4

예시 설명

- 예제 #1
    - [1,2,4]를 이용해서 7을 만들 수 있습니다.
- 예제 #2

- [1,2,4]를 이용해서 7을 만들 수 있습니다.
- [1,4,6]을 이용해서 11을 만들 수 있습니다.
- [2,4,7]을 이용해서 13을 만들 수 있습니다.
- [4,6,7]을 이용해서 17을 만들 수 있습니다.

구현 아이디어

1. 에라토스테네스의 체를 이용하여 소수 판별하는 함수 만들기
2. 조합 함수를 이용하여 3가지을 뽑아 합을 만들 수 있는 모든 경우 탐색

코드

```
from math import sqrt
from itertools import combinations

def is_prime_num(n):
    for i in range(2, int(sqrt(n))+1):
        if n % i == 0:
            return False
    return True


def solution(nums):
    answer = 0
    rep = list(combinations(nums, r = 3))

    for i in rep:
        if is_prime_num(sum(i)):
            answer += 1

    return answer
```

문법 정리

- **itertools.combinations(iterable, r)**
  - 입력 *iterable* 에서 요소의 *r* 길이 부분 조합 리스트를 반환
  - 조합 튜플은 *iterable* 입력의 순서에 따라 사전순으로 방출
  - r : 몇 개의 조합으로 출력할 지정

- 예제 1.

```
from itertools import combinations

a = [1, 2, 3]
print(list(combinations(a, r=2)))

# 결과
[(1, 2), (1, 3), (2, 3)]
```

- 예제 2.

```
from itertools import combinations

a = [1, 2, 3, 4, 5, 6]
print(list(combinations(a, r=5)))

# 결과
[(1, 2, 3, 4, 5), (1, 2, 3, 4, 6), (1, 2, 3, 5, 6), (1, 2, 4, 5, 6),
 (1, 3, 4, 5, 6), (2, 3, 4, 5, 6)]
```