해시

폰켓몬

- n마리 폰켓몬 중 n/2 마리를 선택할때 얼마나 많은 가짓수를 선택할 수 있는지
- 폰켓몬의 종류(n/2보다 작은 경우)만큼 다양한 종류의 폰켓몬을 선택할 수 있다. 포켓몬의 가짓수가 n/2보다 큰 경우 최대로 선택할 수 있는 폰켓몬의 가짓수는 n/2이다
    - [3,1,2,3] : 현재 포켓몬은 4마리이며 포켓몬 종류는 3가지(1,2,3)

      포켓몬의 가짓수가 4/2 = 2보다 크므로 최대로 선택할 수 있는 포켓몬의 가짓수는 2이다
    - [3,3,3,2,2,2] : 현재 폰켓몬은 6마리이며 폰켓몬 종류는 2가지(2,3)

      선택할 수 있는 폰켓몬 종류의 최댓값은 2이다

```
def solution(nums):
    answer = 0
    dic = {}

    for item in nums :
        if str(item) not in dic:
            dic[str(item)] = 0
        dic[str(item)] += 1

    answer = len(dic)
    if answer > len(nums)/2:
        answer = len(nums)/2

    return answer
```

▼ 파이썬

- integer to string

```
str(int_num)
```

- dictionary에 key가 있는지 확인

```
if key in dict:

if key not in dict:
```

- dictionary의 key의 개수

```
len(dict)
```

완주하지 못한 선수

- participant : 마라톤 경기에 참여한 선수 명단 리스트
- completion : 마라톤 경기를 완주한 선수 명단 리스트
  - 길이는 completion 보다 1 작음 (완주하지 못한 선수는 1명)
- 동명이인이 있을 수 있음
- 완주하지 못한 1명의 이름을 반환

```
def solution(participant, completion):
    answer = ''
    partdict={}
    for name in participant:
        if name not in partdict:
            partdict[name] = 0
        partdict[name] += 1

    for name in completion:
        partdict[name] -= 1
        if partdict[name] == 0 :
            del partdict[name]

    answer = list(partdict.keys())[0]

    return answer
```

▼ 파이썬

- dictionary에서 key-value 쌍 지우기

```
del sample_dict[key]
```

- dictionary의 key만 뽑아 리스트화

```
list(sample_dict.keys())
```

⭐ 전화번호 목록

- phone_book : 전화번호들을 담은 배열
- 어떤 한 번호가 다른 번호의 접두어인 경우가 존재한다면 return false
- phone_book을 오름차순으로 정렬하는 것이 포인트 → 인접한 인덱스끼리만 비교할 수 있게 된다
    - 문자열로 이루어진 리스트를 정렬할 경우 alphabetical order로 정렬됨

    ['12','123','1235','567','88']

```
def solution(phone_book):
    answer = True

    phone_book.sort()

    for i in range(0, len(phone_book)-1):
        if phone_book[i+1].startswith(phone_book[i]):
            return False

    return answer
```

▼ 파이썬

- sorted와 sort()의 차이
    - sort()는 리스트 원본 값을 직접 수정
    - sorted(리스트)는 리스트 원본 값을 그대로 두고 정렬값을 반환

```
list1.sort()
list1.sort(key=len)

list2 = sorted(list1)
list2 = sorted(list1, key=len)
```

⭐ 위장

- 주어진 배열 clothes 에서 서로 다르게 입을 수 있는 옷의 조합 수를 구한다
  - 종류가 같은 옷은 한번에 하나를 입을 수 있다
  - 모든 종류의 옷을 입어야 하는 것은 아니다

```
clothes = [["yellow_hat", "headgear"],
["blue_sunglasses", "eyewear"],
["green_turban", "headgear"]]

// 이 경우 다음과 같은 5개의 조합이 가능
// 1. yellow_hat
// 2. blue_sunglasses
// 3. green_turban
// 4. yellow_hat + blue_sunglasses
// 5. green_turban + blue_sunglasses
```

- 테스트케이스 1번만 통과못함 (시간초과)
  - combination 사용해 모든 조합을 다 구하는 풀이라 시간이 오래걸림

```
from itertools import combinations

def solution(clothes):
    answer = 0
    dic = {}

    for item in clothes:
        if item[1] not in dic:
            dic[item[1]] = 0
        dic[item[1]] += 1

    numofkeys = len(list(dic.keys()))
    numofclothes = len(clothes)
    answer += numofclothes
    allcombs = []

    if (numofkeys > 1):
        for i in range(2, numofkeys+1):
            for j in list(combinations(list(dic.values()),i)):
                      allcombs.append(j)

        for item in allcombs:
            temp =1
            for k in range(0, len(item)):
                temp *= item[k]
            answer += temp

    return answer
```

- 조합을 다 구하기보다 이렇게 하는게 빠름
  - 각 옷 종류마다 (n개 중 하나를 입는 경우 + 하나도 안 입는 경우) 의 가능성이 있음
  - 즉 하의가 2개라면 하의를 안 입는 경우를 포함해 총 3가지 경우가 가능
  - 따라서 (하의 2개 + 안입은 경우) * (상의 2개 + 안입은 경우) * (모자 4개 + 안입은 경우) 를 구한 후 마지막에 세 종류 다 안입는 경우의 수를 빼면 됨(하나도 안걸칠 순 없음)
  - 완전 쉬운문제였음

```
def solution(clothes):
    answer = 0
    dic = {}

    for item in clothes:
        if item[1] not in dic:
            dic[item[1]] = 0
        dic[item[1]] += 1


    temp = 1
    for stock in list(dic.values()):
        temp *= stock+1

    answer = temp - 1

    return answer
```

▼ 파이썬

- 리스트 내 항목들의 조합 구하기
  - permutations : 순서도 고려 ((1,2) 와 (2,1)은 다름)
  - combinations : 순서 고려하지 않음

```
from itertools import permutations
from itertools import combinations

a = []
b = []
items = [1,2,3,4,5]

# permutations
for i in list(permutations(items,2)): # permutation으로 2개 뽑음
  a.append(i)

# combinations
for i in list(combinations(items,2)):
  b.append(i)
```

베스트앨범

- 장르별로 가장 많이 재생된 노래 **2개**를 베스트앨범에 넣는다
- 노래 수록 기준
  1. 속한 노래의 재생수 총합이 가장 큰 장르를 먼저 수록
  2. 장르 내에서 많이 재생된 노래를 먼저 수록
  3. 장르 내에서 재생횟수가 같은 노래 중에는 고유번호(인덱스)가 낮은 노래를 먼저 수록
- 장르에 속한 곡이 하나라면 하나의 곡만 수록

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

1. 가장 많이 재생된 장르 pop
  (1) pop 에서 가장 많이 재생된 노래 고유번호 4 -> 1
2. 두번째로 많이 재생된 장르 classic
  (1) classic에서 가장 많이 재생된 노래 고유번호 3 -> 0
```

```
def solution(genres, plays):
    answer = []
    playsPerGenre = {}
    for i in range(0, len(plays)):
        if genres[i] not in playsPerGenre:
            playsPerGenre[genres[i]] = 0
        playsPerGenre[genres[i]] += plays[i]

    sortedDict = sorted(playsPerGenre.items(), key= lambda item:item[1], reverse=True)
    for genre, totalPlays in sortedDict:
        crntGenrePlays = []
        for i in range(0, len(plays)):
            if genres[i] == genre:
                crntGenrePlays.append((i, plays[i]))
        crntGenrePlays = sorted(crntGenrePlays, key=lambda item:item[1], reverse=True)

        for i in range(0, 2):
            if i >= len(crntGenrePlays):
                break
            answer.append(crntGenrePlays[i][0])

    return answer
```

▼ 파이썬 람다함수에 대해 (lambda expression)

- 함수를 간편하게 작성할 수 있음

```
# 일반적인 함수 선언 및 호출
def plus_ten(x):
  return x + 10

plus_ten(1)

# 람다 함수 선언 및 호출
plus_ten = lambda x : x + 10
plus_ten(1)

# 또는
(lambda x : x + 10)(1)
```

- sort에 사용
  - sort 기준을 명시하는 key에 매개변수로 람다함수 작성 가능

```
data_list = ["but","i","wont","hesitate"]

# data_list의 각 원소 x에 대해 x의 길이를 기준으로 data_list를 정렬한다
data_list.sort(key=lambda x : len(x))

print(data_list) # ["i","but","wont","hesitate"]
```

- map에 사용

```
# map(적용할 함수, 리스트)
# 기존 방식
def plus_ten(x) : return x + 10

list(map(plus_ten, [1,2,3])) # [11,12,13]

# 람다 함수 사용
list(map(lambda x : x + 10, [1,2,3]))  # [11,12,13]
```