벤처스타트업아카데미 개념스터디 발표 자료

스터디 01
해시(Hash)

CONTENTS 목차

1. 해시와 해시함수

자료구조를 배우는 이유

해시의 기본 개념

해시함수의 종류

2. 해시 충돌

해시 충돌

해시 충돌 해결 방법

4. 예제 코드

간단한 예제 코드 및 추가 자료

3. Dictionary

Dictionary 활용 예시

Python에서 해시(Hash)를 구현하는 방법

Python에서는 **Dictionary** 라는 자료구조를 통해 해시 제공

해시를 사용하면 좋은 상황

1. 리스트를 쓸 수 없을 때

'문자열, tuple'과 같은 숫자가 아닌 다른 값으로 인덱스 값을 사용하려고 할 때 Dictionary를 사용

list[1]은 가능하지만 list['a']는 사용 불가능

2. 빠른 접근 / 탐색이 필요할 때

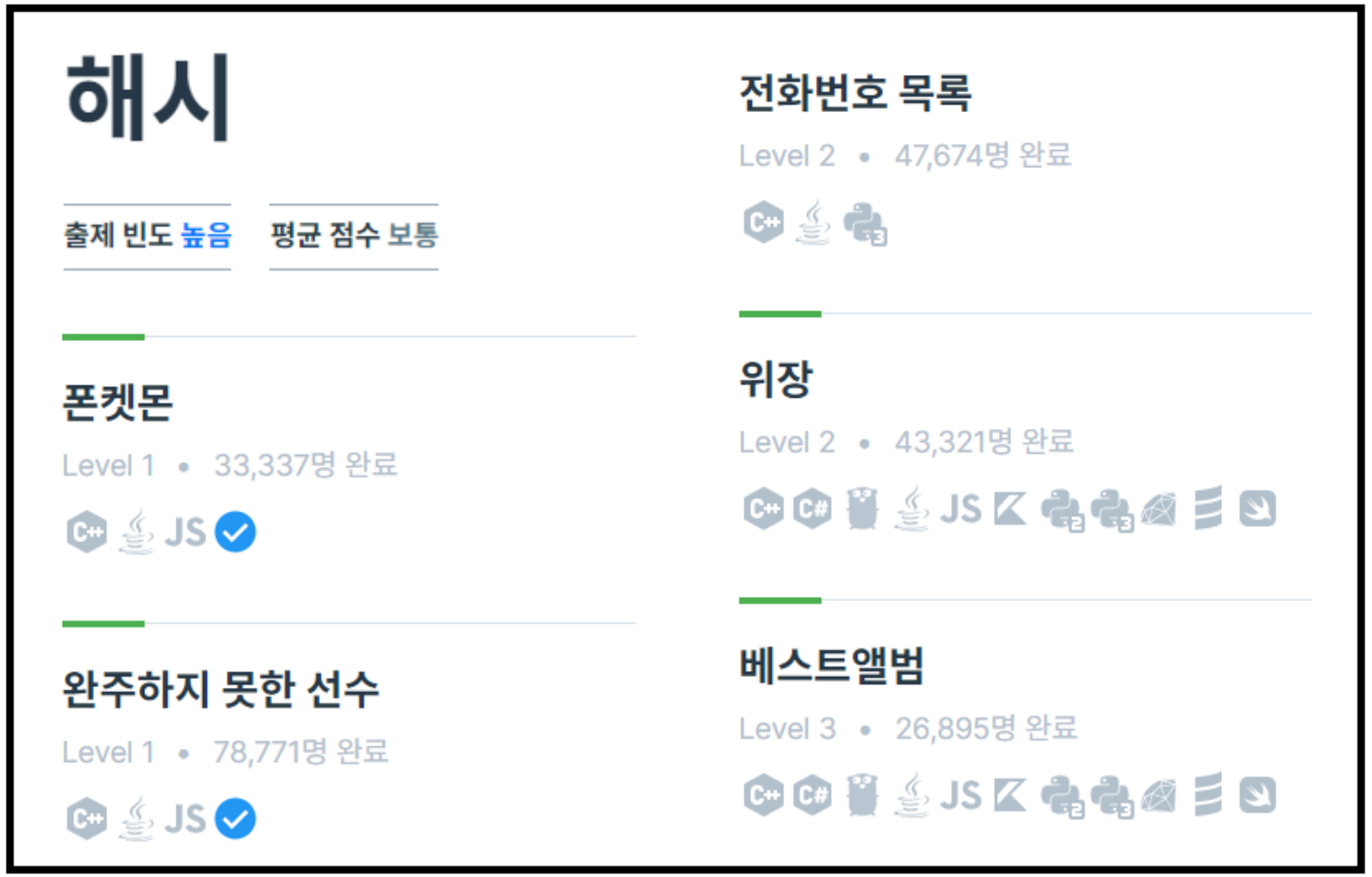
원소를 추가하거나 삭제, 찾는 일이 많을 때에는 Dictionary를 사용하는 것이 좋음.

3. 집계가 필요할 때

원소의 개수를 세는 문제는 코딩 테스트에서 많이 출제되는 문제.
이때 해시와, collections 모듈의 Counter 클래스를 사용하면 빠르게 문제 해결 가능.

프로그래머스 해시 관련 문제

https://school.programmers.co.kr/learn/courses/30/parts/12077



완주하지 못한 선수

문제 요약

- 마라톤에 참여한 선수 이름 배열 : participant
- 완주한 선수 이름 배열 : completion
- 위의 두 배열이 주어지고, 이때 완주 하지 못한 선수의 이름을 return하는 문제입니다.

테스트 케이스

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”

완주하지 못한 선수

```python
def solution(participant, completion):
    hash ={}
    for i in participant:
        if i in hash:
            hash[i] += 1
        else:
            hash[i] = 1
    for i in completion:
        if hash[i] == 1:
            del hash[i]
        else:
            hash[i] -= 1
    answer = list(hash.keys())[0]
    return answer
```

Hash라는 dictionary에 participant 이름을 하나씩 key로 사용하여, 이미 있으면 해당 key 값을 1씩 증가

1이면 동명이인이 없는 선수이며, 2이상이면 동명이인이 value값만큼 있다고 생각

hash를 다 완성하고, 완주자 명단인 completion의 선수들 이름을 하나씩 돌면서, 해당 key값의 value가 1이면 hash에서 해당 key, value를 삭제

동명이인이 아닌 완주자는 hash에서 사라짐

동명이인이 있어서, value가 2이상이면, 완주자 이름이 나올 때마다 -1

완주하지 못한 선수

```python
import collections


def solution(participant, completion):
    answer = collections.Counter(participant) - collections.Counter(completion)
    return list(answer.keys())[0]

```

Counter 클래스를 사용한 예

벤처스타트업아카데미 개념스터디 발표 자료

1. 해시(Hash)

해시와 해시함수

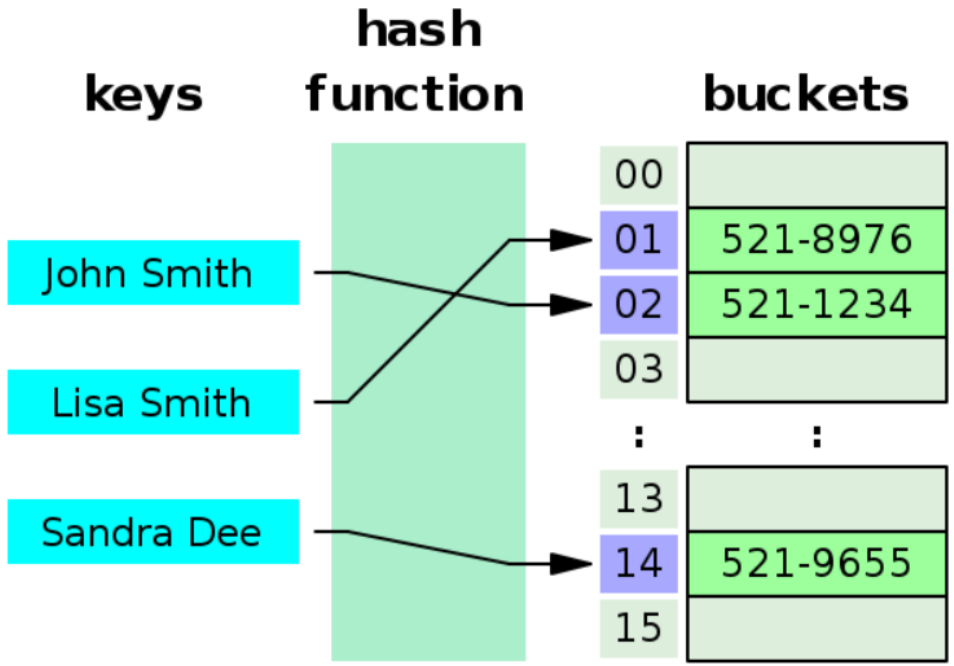
원하는 값을 최대한 효율적으로 찾을 수 있게 하기 위해서 여러가지 저장구조를 배우는 것

자료구조를 배우는 이유

해시 함수(Hash Function)

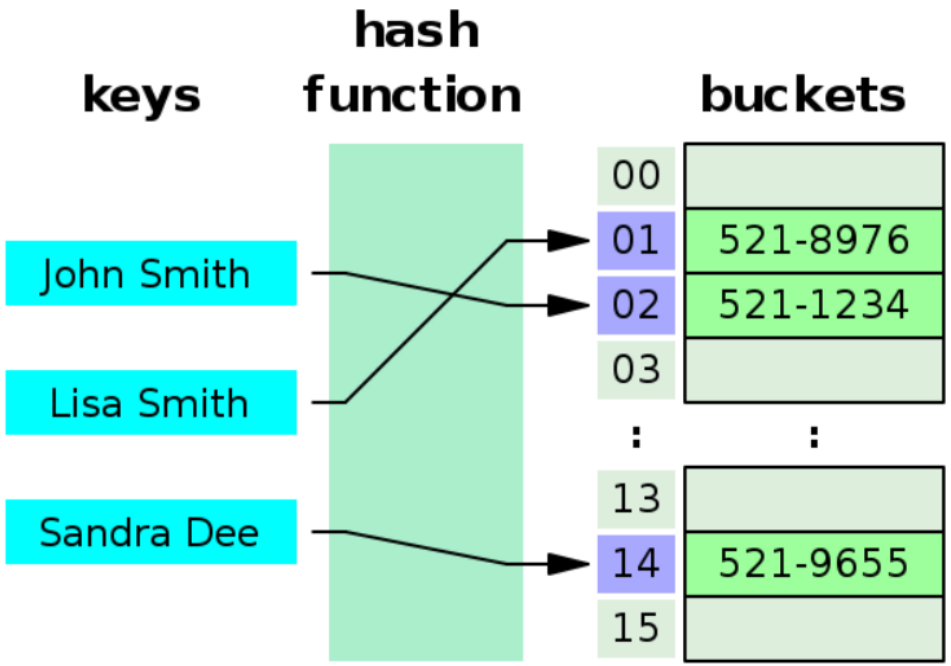
Key로 해시를 만들어내는 함수

key를 고정된 길이의 hash로 변경해주는 역할 → 해싱(Hashing)



key를 해시함수에 Input으로 넣어서 Output으로 나오는 것이 Hash(해시)

해시 테이블 구성



Key

- 고유한 값, hash function의 Input

Hash function

- key를 고정된 길이의 hash로 변경해주는 역할

- 서로 다른 key가 hashing 후 같은 hash값이 나오는 경우가 있다. 이를 해시충돌이라고 부름. 해시 충돌 발생확률이 적을 수록 좋음.

Value

- 저장소(버킷,슬롯)에 최종적으로 저장되는 값으로, hash와 매칭되어 저장

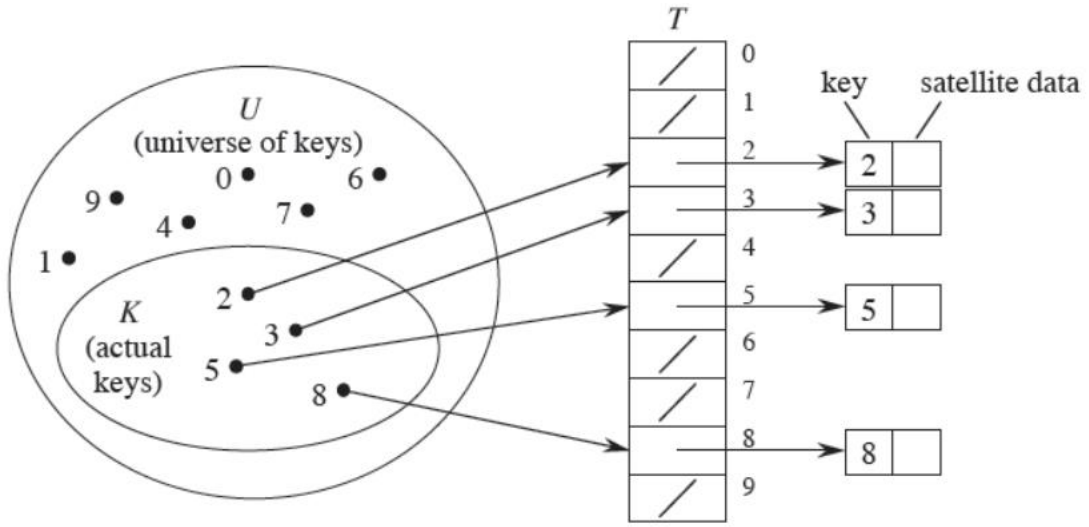
데이터가 저장되는 곳을 버킷, 슬롯이라고 함.

장단점

장점

해시테이블은 key-value가 1:1로 mapping되어 있기 때문에 삽입, 삭제, 검색의 과정에서 모두 평균적으로O(1)의 시간 복잡도(Big-O)를 가지고 있다.

*시간 복잡도 Big-O: 알고리즘 성능을 수학적으로 표기해주는 표기법



단점

- 해시 충돌이 발생(개방 주소법, 체이닝 과 같은 기법으로 해결해 줘야 한다.)
- 순서/관계가 있는 배열에는 어울리지 않는다.
- 공간 효율성이 떨어진다. 데이터가 저장되기 전에 저장공간을 미리 만들어야 한다. 공간을 만들었지만 공간에 채워지지 않는 경우가 발생한다.
- hash function의 의존도가 높다. 해시함수가 복잡하다면 hash를 만들어 내는데 오래 걸릴 것이다.

해시함수 예시

division method

- 가장 기본적인 해시 함수이다. 숫자로 된 키를 해시테이블 크기 m으로 나눈 나머지를 해시값으로 변환한다. 간단하면서도 빠른 연산이 가능한 것이 장점이다.
- 해시의 중복을 방지하기 위해 테이블의 크기 m은 소수로 지정해서 사용하는것이 좋다. 하지만 남는 공간이 발생해 메모리상으로 비효율적이다.

multiplication method

- 숫자 키 k, A는 0<A<1 사이의 실수 일때 `h(k) = (ka mod 1)*m 으로 계산한다.
- 2진수 연산에 최적화된 컴퓨터구조를 고려한 해시함수이다.

multiplication method

- 여러개의 해시함수를 만들고, 이 해사함수의 집합 H에서 무작위로 해시함수를 선택해 해시값을 만드는 기법.
- 서로 다른 해시함수가 서로 다른 해시값을 만들어내기 때문에 같은 공간에 매핑할 확률을 줄이는 것이 목적

자바에서 사용하는 Hash

`HashTable` 이란 JDK 1.0부터 있던 Java의 API이고, `HashMap` 은 Java2에서 처음 선보인 Java Collections Framework에 속한 API이다.

HashTable 또한 Map 인터페이스를 구현하고 있기때문에 HashMap과 HashTable이 제공하는 기능은 같다.

차이점

- 보조 해시 함수
  - HashMap은 보조 해시함수를 사용하기 때문에 보조 해시 함수를 사용하지 않는 HashTable에 비하여 해시 충돌(hash collision)이 덜 발생할 수 있어 상대적으로 성능상 이점이 있다.

- 동기화
  - HashMap의 경우 동기화를 지원하지 않는다. 그래서 Hash Table은 동기화 처리라는 비용때문에 HashMap에 비해 더 느리다고 한다. 프로그래밍상의 편의성 때문에 멀티쓰레드 환경에서도 HashTable을 쓰기보다는 HashMap을 다시 감싸서 `Map m == Collections.syschronizedMap(new HashpMap());` 과 같은 형태가 더 선호된다.

*참고 자료: https://go-coding.tistory.com/30

요약

- 해시(hash)는?

> 색인 또는 인덱스

- 해시 함수(hash function)는?

> Key를 hash로 만들어 주는 함수

- 해시테이블?

> Hash를 주소로 삼아 데이터를 저장하는 자료구조

벤처스타트업아카데미 개념스터디 발표 자료

2. 해시 충돌

Hash Collision 및 해결 방법

해시 충돌

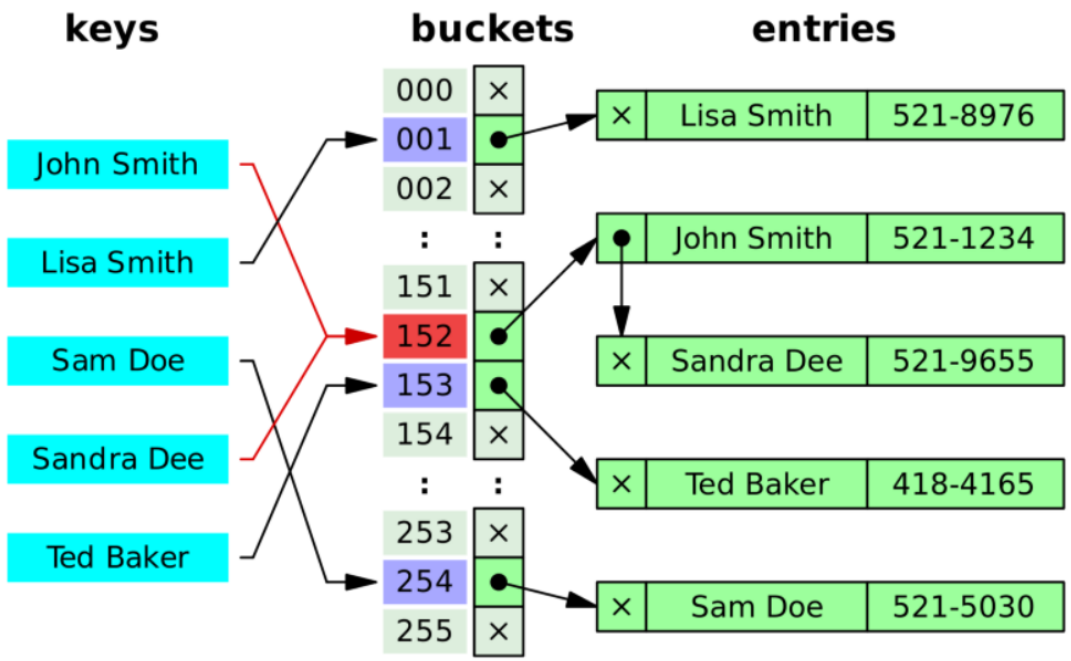
서로 다른 key 값을 사용하여 같은 hash 값이 얻어진 상황

해시 함수로 해시를 만드는 과정에서 서로 다른 key가 같은 해시로 변경되면 같은 공간에 2개의 value가 저장되므로 key-value가 1:1로 mapping되어야 하는 해시 테이블의 특성에 위배

해시 충돌은 필연적으로 나타날 수 밖에 없다.

Chaining

저장소(Bucket)에서 충돌이 일어나면 기존 값과 새로운 값을 연결리스트로 연결하는 방법



장점

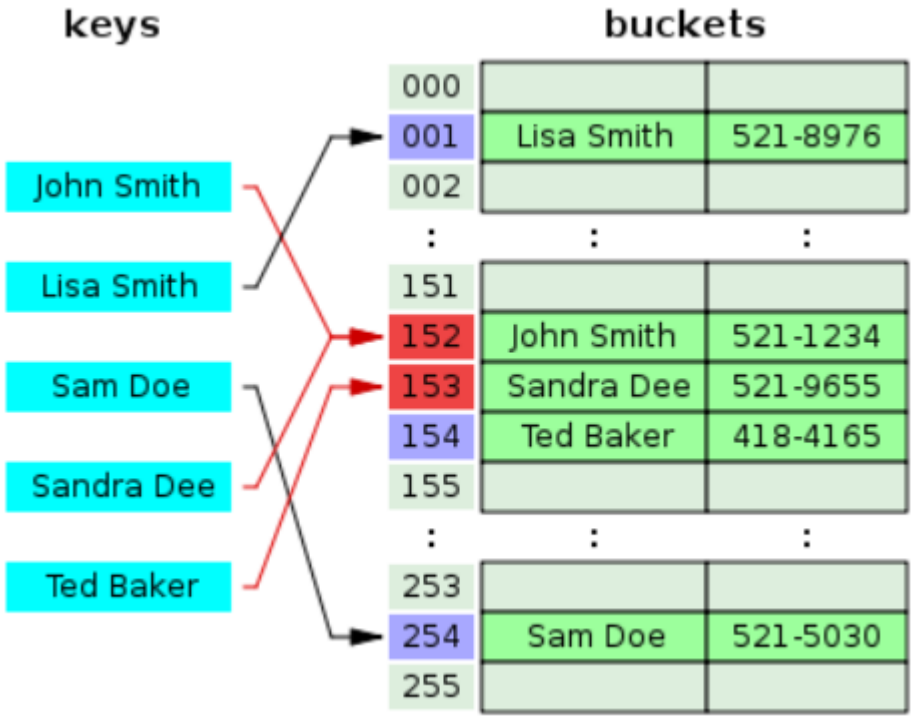
미리 충돌을 대비해서 공간을 많이 잡아 놓을 필요가 없음
(충돌이 나면 그때 공간을 만들어서 연결)

단점

같은 hash에 자료들이 많이 연결되면
검색 시 효율이 낮아진다.

(해결 방법 2) Open Addressing(개방주소법)개방 주소법은 충돌이 일어나면 비어 있는 hash에 데이터를 저장하는 방법. 개방주소법의 해시 테이블은 hash와 value가 1:1관계를 유지.

Open Addressing 충돌이 발생할 경우 비어 있는 hash를 찾아 저장하는 방법



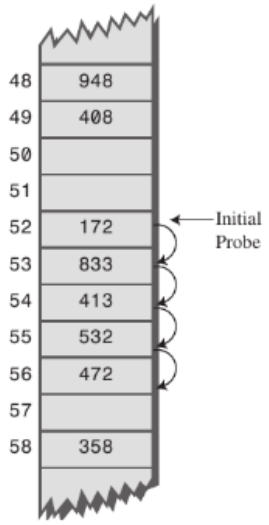
옆의 그림에서 John과 Sandra의 hash가 동일해 충돌이 일어난다. 이때 Sandra는 바로 그 다음 비어 있던 153 hash에 값을 저장한다. 그 다음 Ted가 테이블에 저장을 하려 했으나 본인의 hash에 이미 Sandra로 채워져 있어 Ted도 Sandra처럼 바로 다음 비어 있던 154 hash에 값을 저장한다.

Open Addressing(개방주소법)개방 주소법은

충돌이 일어나면 비어 있는 hash에 데이터를 저장하는 방법.

개방주소법의 해시 테이블은 hash와 value가 1:1관계를 유지.

비어 있는 hash를 찾아가는 방법

- 선형탐색
  - 해시값에서 고정폭으로 건너 뛰면서 비어있는 해시가 나오면 저장하게 된다.
  - 

    52번 해시에서 충복이 일어나 고정폭(1) 씩 건너뛰면서 빈 해시를 찾으려면 57까지 5번의 연산을 해야한다. 특정 해시값 주변 버킷이 모두 채워져 있는 primary clustring 문제에 취약하다.

- 제곱 탐색
  - 고정폭이 아닌 1칸 -> 4칸 -> 9칸 -> 16칸 씩 건너 뛰면서 빈 칸은 찾는다. 해시값이 같은 해시들이 들어오면 공간을 많이 확보해놔야 한다.

벤처스타트업아카데미 개념스터디 발표 자료

3. Dictionary

Key 와 Value를 한 쌍으로 갖는 자료형

Dictionary

참고 자료 https://wikidocs.net/16
https://yunaaaas.tistory.com/46

List 나 Tuple처럼 순차적으로(sequential) 해당 요소의 값을 구하지 않고 Key를 통해 Value를 얻는다.

기본 형태

{Key1: Value1, Key2: Value2, Key3: Value3, …}

활용

```
# 특정 key-value쌍을 가진 dictionary 선언

Dog = {
    'name': '동동이',
    'weight': 4,
    'height': 100,
}
```

```
# dictionary를 value로 가지는 dictionary 선언

Animals = {
    'Dog': {
        'name': '동동이',
        'age': '5'
    },
    'Cat': {
        'name': '야옹이',
        'weight': 3
    }
}
```

Dictionary 요소 추가

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
```

{1: 'a'} 딕셔너리에 a[2] = 'b'와 같이 입력하면 딕셔너리 a에 Key와 Value가 각각 2와 'b'인 `{2 : 'b'}` 딕셔너리 쌍이 추가된다.

```
>>> a['name'] = 'pey'
>>> a
{1: 'a', 2: 'b', 'name': 'pey'}
```

딕셔너리 a에 `{'name': 'pey'}` 쌍이 추가되었다.

```
>>> a[3] = [1,2,3]
>>> a
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

Key는 3, Value는 [1, 2, 3]을 가지는 한 쌍이 또 추가되었다.

Dictionary 요소 추가

값을 집어넣거나, 값을 업데이트 할 때 [ ] 를 사용

```
# 값 집어넣기

dict = {'김철수': 300, 'Anna': 180}
dict['홍길동'] = 100
dict #{'Anna': 180, '김철수': 300, '홍길동': 100}
```

```
# 값 수정하기1

dict = {'김철수': 300, 'Anna': 180}
dict['김철수'] = 500
dict # {'Anna': 180, '김철수': 500}
```

```
# 값 수정하기2

dict = {'김철수': 300, 'Anna': 180}
dict['김철수'] += 500
dict # {'Anna': 180, '김철수': 800}
```

Dictionary 요소 삭제

del dict_obj[key]

del은 키워드(예약어)로써, 만약 Dictionary에 key가 없다면 keyError 발생

```
# del 이용하기 - 키가 있는 경우
dict = {'김철수': 300, 'Anna': 180}
del dict['김철수']

dict #{'Anna': 180}
```

```
# del 이용하기 - 키가 없는 경우 raise KeyError
my_dict = {'김철수': 300, 'Anna': 180}
del my_dict['홍길동']
'''
keyError 발생!
'''
```

Dictionary 요소 삭제

pop(key[,default])

pop은 메소드로써, pop메소드는 key 값에 해당하는 value를 리턴.
key가 없다면 두번째 파라미터인 default를 리턴.

만약 default 설정하지 않았을 시 keyError가 발생

```
# pop 이용하기 - 키가 있는 경우 대응하는 value 리턴
my_dict = {'김철수': 300, 'Anna': 180}
my_dict.pop('김철수', 180) # 300
```

```
# pop 이용하기 - 키가 없는 경우 대응하는 default 리턴
my_dict = {'김철수': 300, 'Anna': 180}
my_dict.pop('홍길동', 180) # 180
```

기타 자료

del은 예약어, pop은 메서드(파이썬에서는/함수)

예약어

예약어 (또는 키워드)란 파이썬에서 이미 문법적인 용도로 사용되고 있기 때문에 변수명 등의 식별자로 사용하면 안 되는 단어들을 말한다. 예약어는 Reserved(예약된) Words 또는 키워드라고 한다. 파이썬에서 이미 사용되고 있는(용도가 예약된) 단어들을 지칭하며 이미 문법적인 용도로 사용(이를 신택스(Syntax)라고 부른다)되기 때문에 변수명 등의 식별자로 사용하면 안 된다. 예약어를 변수에 활용하면 문제가 발생한다. 에러가 발생하는 것이 아닌 예약어 고유 기능이 사라지게 되는 것이다.

내장함수

별도의 모듈(Module)의 추가 없이 기본적으로 제공되는 함수들을 말한다.

https://blog.naver.com/lool2389/220649062093

Dictionary 요소 삭제

딕셔너리 요소 삭제하기

```
>>> del a[1]
>>> a
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

위 예제는 딕셔너리 요소를 지우는 방법을 보여 준다. del 함수를 사용해서 del a[key]처럼 입력하면 지정한 Key에 해당하는 {key : value} 쌍이 삭제된다.

Del 함수 이용

Iterate

딕셔너리를 for문을 이용하여 조회하는 방법

Key 로만 순회하는 경우

```
# key로만 순회
dict = {'김철수': 300, 'Anna': 180}
for key in dict:
    print(key)
    # 이 경우 value를 찾고 싶으면 dict[key] 와 같이 접근을 따로 해주어야.

'''
김철수
Anna
'''
```

Iterate

key, value 동시 순회 (items() 사용)

```
# key-value 동시 순회

dict = {'김철수': 300, 'Anna': 180}
for key, value in dict.items():
    print(key, value)

'''
김철수 300
Anna 180
'''
```

기타 활용 예시

특정 key가 딕셔너리에 있는지 없는지 조회 In 을 이용

```
dict = {'김철수': 300, 'Anna': 180}
print("김철수" in dict) #True
print("김철수" not in dict) # False
```

과제 1번. 폰켓몬

```
def solution(nums):
    arr=[]
    for i in nums:
        if i not in arr:
            arr.append(i)
    return len(nums)/2 if len(nums)/2 < len(arr) else len(arr)
```

기타 활용 예시

key 또는 value만 뽑아내는 방법

1. key 만 : keys()

```
# key를 extract - keys 사용


my_dict = {'김철수': 300, 'Anna': 180}
my_dict.keys() # dict_keys(['김철수', 'Anna'])
```

2. value만 : values()

```
# value를 extract - values 사용


my_dict = {'김철수': 300, 'Anna': 180}
my_dict.values() # dict_values([300, 180])
```

기타 활용 예시

key 또는 value만 뽑아내는 방법

3. key - value 모두 : items()

```
# key, value 쌍을 extract - items 사용


my_dict = {'김철수': 300, 'Anna': 180}
my_dict.items() # dict_items([('김철수', 300), ('Anna', 180)])
```

기타 활용 예시 - Counter

파이썬 collections 모듈의 Counter 사용법

```
from collections import Counter
```

Counter 생성자에 중복된 데이터가 저장된 배열을 인자로 주면, 각 원소가 몇 번씩 나오는지가 저장된 객체를 얻을 수 있음.

```
>>> Counter(["hi", "hey", "hi", "hi", "hello", "hey"])
Counter({'hi': 3, 'hey': 2, 'hello': 1})
```

```
>>> Counter("hello world")
Counter({'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1})
```

기타 활용 예시 - Counter

파이썬 collections 모듈의 Counter 사용법

대괄호를 사용하여 키 값을 읽을 수 있다.

```
counter = Counter("hello world")
counter["o"], counter["l"]
```

```
(2, 3)
```

기타 활용 예시 - Counter

파이썬 collections 모듈의 Counter 사용법

특정 키 값을 갱신할 수 있음

```
>>> Counter("hello world")
Counter({'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1})
```

```
counter["l"] += 1
counter["h"] -= 1
counter
```

```
Counter({'h': 0, 'e': 1, 'l': 4, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1})
```

기타 활용 예시 - Counter

가장 많이 나온 데이터 찾기 .most_common()

```
from collections import Counter

Counter('hello world').most_common()
```

```
[('l', 3), ('o', 2), ('h', 1), ('e', 1), (' ', 1), ('w', 1), ('r', 1), ('d', 1)]
```

```
from collections import Counter

Counter('hello world').most_common(1)
```

```
[('l', 3)]
```

기타 활용 예시 - Counter

산술 연산자 활용

예를 들어, 아래와 같이 2개의 카운터 객체가 있을 때,

```
counter1 = Counter(["A", "A", "B"])
counter2 = Counter(["A", "B", "B"])
```

```
counter1 + counter2
```

```
Counter({'A': 3, 'B': 3})
```

```
counter1 - counter2
```

```
Counter({'A': 1})
```

뺄셈의 결과로 0이나 음수가 나온 경우에는
최종 카운터 객체에서 제외

4. 예제 코드

추가 자료

전화 번호 목록 (Level 2)

https://programmers.co.kr/learn/courses/30/lessons/42577

문제 요약

- phone_book : 전화번호부 배열이 주어집니다.
- 한 번호가 전화번호부의 다른 번호의 접두어인 경우가 존재하면 false, 없으면 true를 return합니다.

테스트 케이스

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

전화 번호 목록 (Level 2)

https://programmers.co.kr/learn/courses/30/lessons/42577

해결 예시

```python
def solution(phone_book):
    answer = True
    finish = False
    phone_book = sorted(phone_book, key=len)
    for i in range(0, len(phone_book)):
        if finish:
            break
        current = phone_book[i]
        for j in range(i+1, len(phone_book)):
            comp = phone_book[j]
            if len(current)<len(comp) and current == comp[0:len(current)]:
                answer = False
                finish = True
                break
    return answer
```

베스트 앨범 (Level 3)

https://programmers.co.kr/learn/courses/30/lessons/42579

문제 요약

- genres[i] : 고유번호가 i인 노래의 장르
- plays[i] : 고유번호가 i인 노래의 재생 횟수
- 베스트 앨범은 아래와 같은 규칙으로 선정합니다.

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.
4. 위의 규칙으로 장르당 최대 2곡을 선정하여 앨범에 수록합니다.

테스트 케이스

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

베스트 앨범 (Level 3)

해결 예시

```
def solution(genres, plays):
    song = dict()
    sum_play = dict()
    n = len(genres)
    answer = []

    for i in range(n):
        genre = genres[i]
        num_play = plays[i]
        if genre in song:
            song[genre].append([(-1)*num_play, i])
            song[genre].sort()
        else:
            song[genre] = [[(-1)*num_play, i]]

        if genre in sum_play:
            sum_play[genre] += num_play
        else:
            sum_play[genre] = num_play
    sorted_list = []
    for k in sum_play.keys():
        sorted_list.append([sum_play[k], k])
    sorted_list.sort(reverse=True)

    for _, g in sorted_list:
        for i in range(2):
            answer.append(song[g][i][1])
            if len(song[g]) == 1:
                break
    return answer
```

베스트 앨범 (Level 3)

song

```
if genre in song:
    song[genre].append([(-1)*num_play, i])
    song[genre].sort()
else:
    song[genre] = [[(-1)*num_play, i]]
```

Sum_play

```
if genre in sum_play:
    sum_play[genre] += num_play
else:
    sum_play[genre] = num_play
```

베스트 앨범 (Level 3)

```
sorted_list = []
for k in sum_play.keys():
    sorted_list.append([sum_play[k], k])
sorted_list.sort(reverse=True)
```

```
for _, g in sorted_list:
    for i in range(2):
        answer.append(song[g][i][1])
        if len(song[g]) == 1:
            break
```