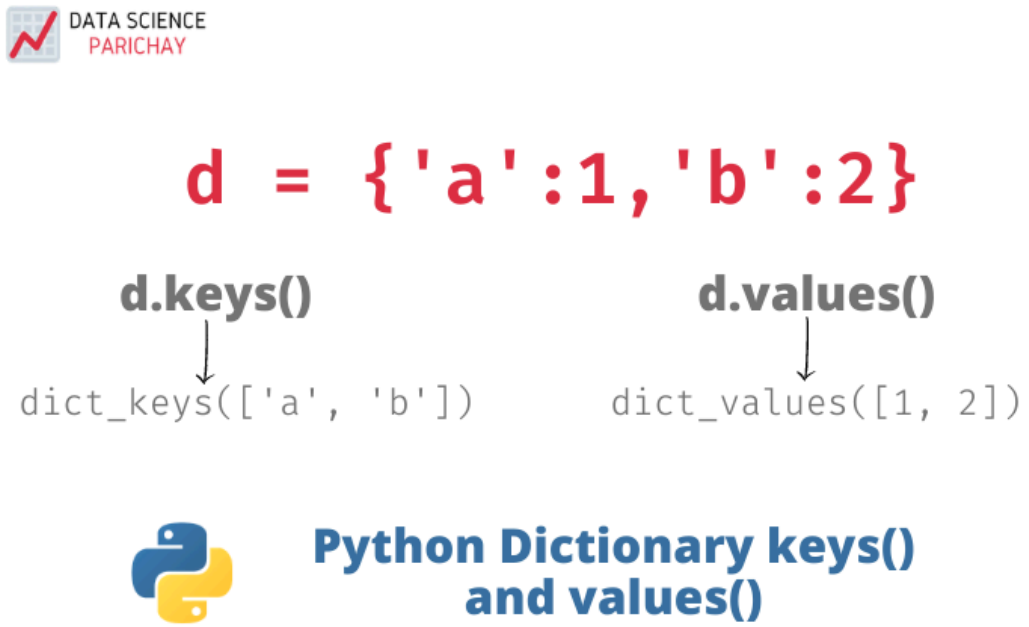
Day 5 - Dictionary & Set

개요



Dictionary와 Set에 대해 알아본다

Ditctionary 내부구조

Hash Table

해시 테이블은 key, value로 데이터를 저장하는 자료 구조 중 하나다. Python에는 크게 두 가지의 Hash Table이 존재한다.

- **Dictionary**
- **Set**

빠르게 데이터를 검색 할 수 있기에 많이 사용된다. 검색의 시간복잡도는 O(1)이다.

해싱(Hashing) 이란?

해싱

- 입력 값에 수학적 알고리즘(해시 함수)를 적용해 고정된 값을 출력하는 과정

해시 함수

- 일반적으로 입력 값을 고정된 크기의 블록으로 나누고, 수학적 연산을 적용해 최종 해시를 출력함
- 입력 데이터가 조금이라도 바뀌어도 출력 값은 크게 달라짐
- SHA-256, SHA-3, MD5 등등

- 나머지 연산도 해싱 함수로 사용할 수 있다

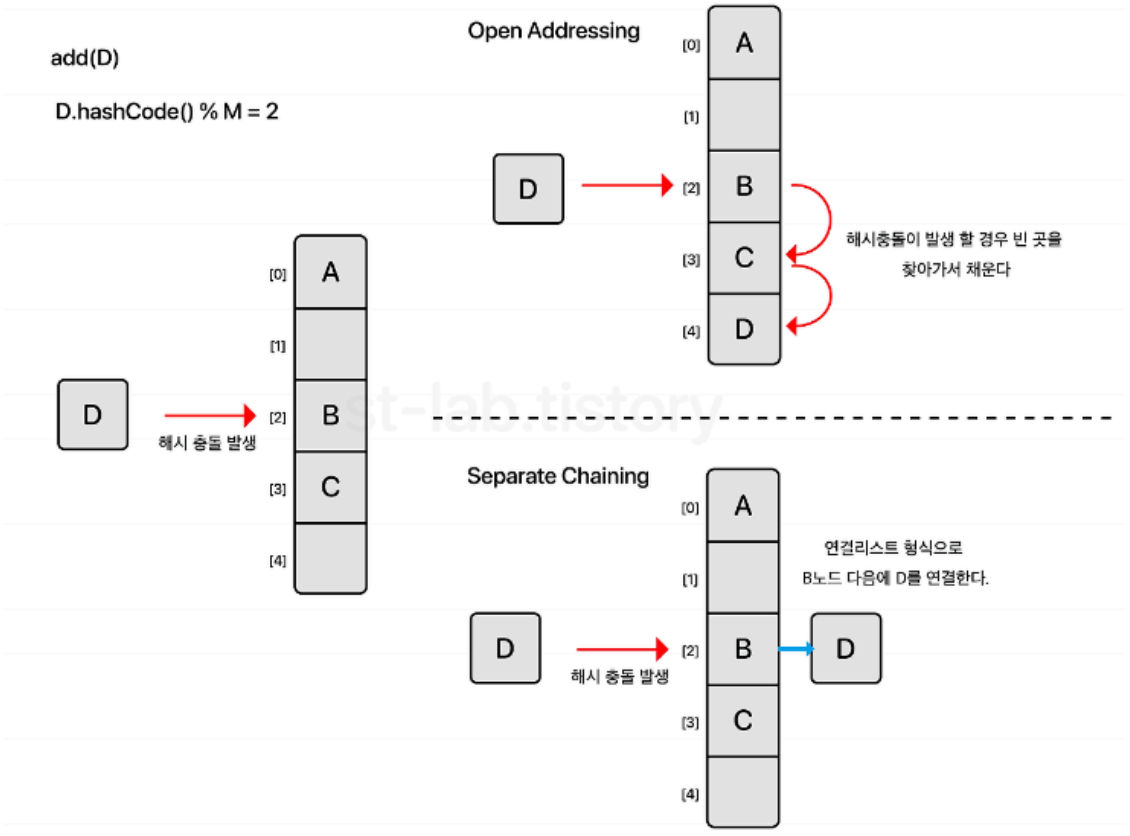
충돌 (Collition)

해시 함수에 따라서는 여러 키 값이 하나의 키로 매핑 될 수 있다. 이런 경우를 **해싱 충돌**이라고 한다.

```
def hashing(v: int):
    return v % 7

if __name__ == "__main__":
    print(hashing(1)) # 1
    print(hashing(8)) # 1
```

해싱 충돌을 해결하는 방법에는 크게 두 가지가 있다



Open Addressing (linear probing)

- close hasing이라고도 한다
- 이어진 bucket을 찾아 값을 저장한다
- bucket이 다 채워질 경우 더 많은 bucket을 가진 테이블이 새로 생성된다 (rehashing)
- 파이썬에서는 이 방법을 우선적으로 사용한다 (구현 오버헤드 때문)

```
hash_table = [0 for i in range(11)]

def save(key, value):
    index = hash(key) % len(hash_table)

    #index(key의 주소)부터 hash_table끝까지 선형 탐색
    for i in range(index, len(hash_table)):

        #빈 공간이면 데이터 저장
        if hash_table[i] == 0:
            hash_table[i] = [key, value]
            return

        #빈 공간이 아니고, 데이터의 key값이 같다면, 업데이트
        elif hash_table[i][0] == key:
            hash_table[i][1] = value
            return

#테스트 코드

key_candidates = ['a','b','c','d','e','f','g','h']

for i in range(7):
    save_oa(key_candidates[i], hash(key_candidates[i]) % len(hash_table))

print(hash_table)
```

```
# 결과
[['c', 0], ['d', 0], ['a', 2], ['f', 3], 0, ['e', 5], ['g', 6], 0, 0, 0, ['b', 10]]
```

Seperate Chaining

- 각 버킷을 링크드 리스트로 연결한다
- C++이나 자바 같은 전통적인 언어는 이 기법을 주로 사용한다

```
hash_table = [0 for i in range(11)]

def save(key, value):
    #임의의 해시함수
    index = hash(key) % len(hash_table)

    #만약 해시함수값(index)의 버킷이 이미 찼다면
    if hash_table[index] != 0:

        #해당 버킷에 링크드 리스트가 구현되어 있을 수도 있고,
        #그 안에 해당 key값이 이미 저장되어 있을 수 있으니 찾아본다.
        for data in hash_table[index]:
            if data[0] == key:
                data[1] = value #이미 저장되어 있다면 업데이트
                return

        #처음 저장하는 데이터라면 뒤에 붙여준다.
        return hash_table[index].append([key, value])

    #애초에 빈 버킷이었다면 2차원 리스트(링크드 리스트 흉내)로 버킷 업데이트
    hash_table[index] = [[key, value]]
    return

#테스트 코드

import random

key_candidates = ['a','b','c','d','e','f','g','h']
```

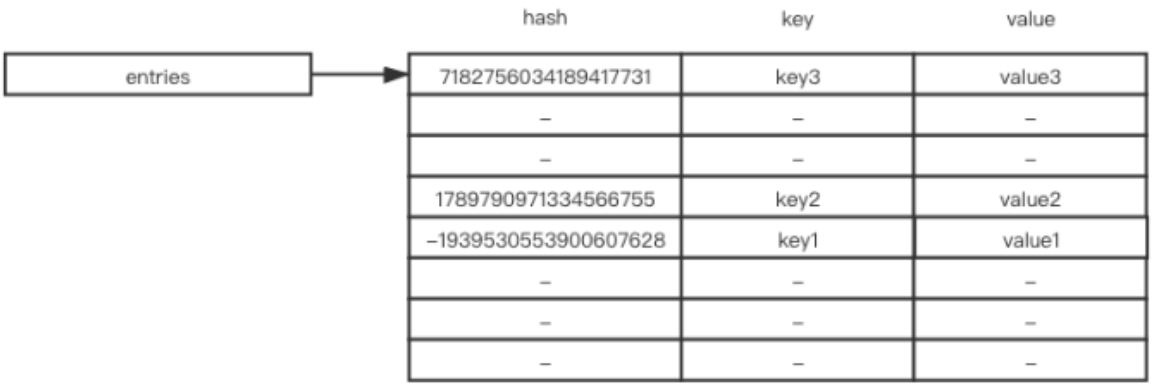
```
for i in range(7):
    save(key_candidates[i], random.randint(1,11))

print(hash_table)
```

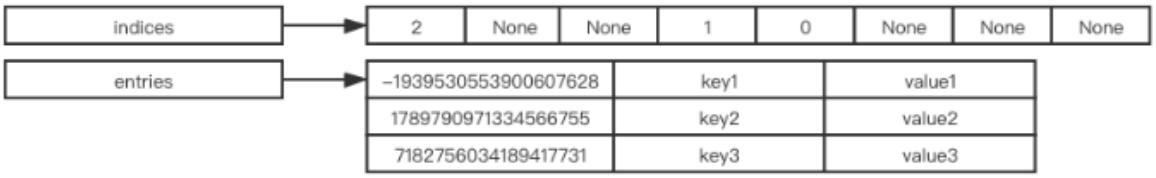
```
# 결과
[
    [['c', 0], ['d', 0]],
    0,
    [['a', 2]],
    [['f', 3]],
    0,
    [['e', 5]],
    [['g', 6]],
    0,
    0,
    0,
    [['b', 10]]
]
```

파이썬 내부 구현

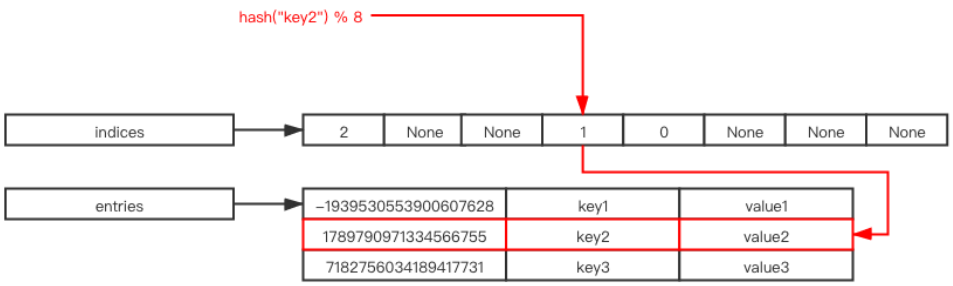
3.6 버전 이전의 CPython은 아래와 같은 구조로 dictionary를 구현한다-



이런 경우 메모리의 낭비가 심할 수 있다. 이런 문제를 해결하기 위해 hash table 내 메모리 주소와 인덱스를 분리하도록 개선되었다. (3.6 이후)

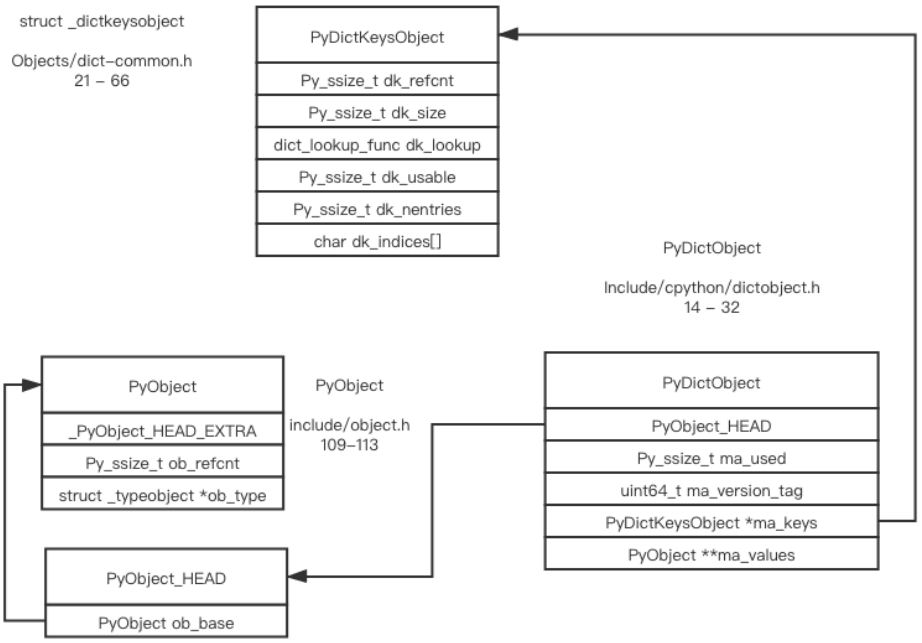


indices는 더 간단한 버전의 hash table이라고 생각하면 편하다. entries는 array며, 기존의 hash value, 키, 값을 각 요소로 저장한다. 탐색이 일어날 때, 가정 먼저 indices를 탐색하고, entires의 인덱스 값을 확인하게 된다



파이썬 구현체를 확인해보면, 다음과 같은 memory layout을 가지는 것을 알 수 있다시간 복잡도

.



시간 복잡도

Operation	Average Case	Amortized Worst Case
k in d	O(1)	O(n)
Copy[3]	O(n)	O(n)
Get Item	O(1)	O(n)
Set Item[1]	O(1)	O(n)
Delete Item	O(1)	O(n)
Iteration[3]	O(n)	O(n)

- 대체적으로 O(1)의 시간 복잡도를 지닌다 (바로 명령이 실행된다는 뜻)
- 최악의 경우, O(n)인데, 이 경우 키가 고르지 않게 분포되었을 경우다.

Ditctionary 사용하기

초기화

```
# 딕셔너리 기본형태
# {키:값, 키:값, 키:값}으로 키-값을 구분해주는 것은 콤마(,)
X = {'a':4, 'b':5, 'c':6}

# 혹은 다음과 같이 빈 딕셔너리를 선언할 수도 있다
X = {}
X = dict()
```

그렇다면 이미 존재하는 값으로부터 dictionary를 초기회 하는 방법은?

```
a = [('동해',1), ('물과',2),('백두',3),('산이',4)]
dic_1, dic_2 = {}, {}

# 1. for 문을 사용하는 방법
for i,j in a:
    dic_1[i] = j

# 2. dictcomp를 이용하는 방법 (더 빠르다)
dic_2 = {i:j for i,j in a}
```

조회하기

```
a = [('동해',1), ('물과',2),('백두',3),('산이',4)]
my_dict = {i:j for i,j in a}

my_dict['동해'] # 1
my_dict['APT'] # 에러

"""
이런 에러가 난다

Traceback (most recent call last):
  File "/home/ubuntu/workspace/study-python/Lectures/Day-5/code/test2.py'
    dic_2['APT']
KeyError: 'APT'
"""
```

```
# 특정 요소가 포함되어있는지 확인을 위해 'in'을 사용할 수 있다
print('산이' in my_dict)
>>> True
```

이런 문제 때문에 get을 사용하기도 한다

get(key[,default])

- key 가 딕셔너리에 있는 경우 key 에 대응하는 값을 돌려주고, 그렇지 않으면 default 를 돌려준다
- default가 없으면 None이 적용된다

```
>>> d = {"one": 1, "two": 2, "three": 3}
>>> d.get("one")
1


# four 라는 key 가 없으므로, default 값인 4를 반환함
>>> d.get("four", 4)
4

# 자료에는 변화가 없음
>>> print(d)
{"one": 1, "two": 2, "three": 3}
```

업데이트

```
d = {"one": 1, "two": 2, "three": 3}

# 같은 키를 쓰면 값을 업데이트 할 수 있다
d["one"]=11 # 11로 업데이트

# update를 사용해 한번에 업데이트가 될 수 있다 (없으면 추가)
d.update({'two': 22, 'three': 33, 'four': 44})
```

```
print(d)
>>> {'one': 11, 'two': 22, 'three': 33, 'four': 44}
```

삭제하기

```
# del이나 pop을 사용해 삭제할 수 있다
d = {"one": 1, "two": 2, "three": 3}

del d["one"] # {'two': 2, 'three': 3}

# pop을 사용하면 값을 반환받을 수 있다
print(d.pop("two")) # 2
print(d)            # {'three': 3}
```

순회하기

```
# items 는 key, value에 대한 iterator를 반환한다
d = {"one": 1, "two": 2, "three": 3}

for k,v in d.items():
    print(k, v)

>>> one 1
>>> two 2
>>> three 3

# keys()는 모든 키값을, values()는 값을 반환한다
print(d.keys(), d.values())
>>> dict_keys(['one', 'two', 'three'])
>>> dict_values([1, 2, 3])
```

Set 사용하기

Set 은 Dictionary에 value 값이 없는 버전이라고 생각하면 이해하기 편하다

- 순서가 없고
- 중복을 허용하지 않는다

이런 특성때문에 중복을 제거하기 위한 필터로 많이 사용한다

```
s = set("Hello")
print(s)

>>> {'e','o','l','H'}
```

이를 사용하여 교집합, 차집합, 합집합 등을 구현할 수 있다.

```
set1 = {1,2}
set2 = {2,3}

# 교집합
print(set1 & set2)
>>> {2}

# 합집합
print(set1 | set2)
>>> {1, 2, 3}

# 차집합
print(set1 - set2)
>>> {1}
```

더보기

여러가지 변형 Dictionary, Set 들이 존재한다

- defaultdict : 기본 값을 지정할 수 있는 dictionary
- orderdict : 삽입 순서대로 조회 가능한 dictionary
- MappingProxyType : 읽기 전용 dict를 만들기 위한 wrapper

- FrozenSet : 불변으로 생성한 set

이 중 defaultdict는 꽤 유용하다

```
# default dict를 사용하지 않는 버전
def count_letters(words):
  counter = {}
  for letter in words:
      counter.setdefault(letter,0) # key 값이 없으면 0으로 초기화
      counter[letter] += 1
    return counter
count_letters("apple")

# default_dict를 사용한 버전
from collections import defaultdict

def count_letters_default(words):
    counter = defaultdict(int) # 값을 지정하지 않은 키는 그 값이 0으로 지정
    for letter in words:
        counter[letter] += 1 # 기본 값을 넣는 과정이 없어졌다
    return counter
count_letters_default("apple")
```

레퍼런스

- https://wikidocs.net/234538
- https://wiki.python.org/moin/TimeComplexity