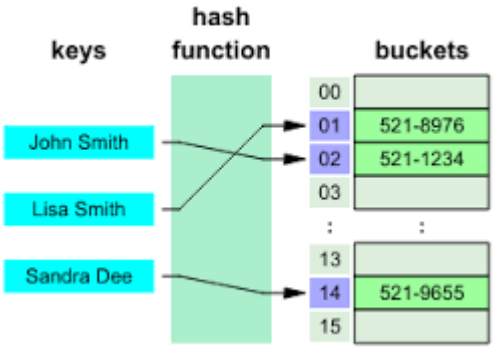
Hash

👋 Notion에 오신 것을 환영합니다!

<4주차> HASH

Hash Table



- Key, Value 쌍을 저장하고 효율적으로 검색할 수 있는 자료구조

Hash Function

- keys를 입력받아 Hash 값(Hash Table의 Index)을 출력하는 함수
- Python의 `hash()` 함수는 주어진 객체의 내용을 기반으로 한 해시 값을 반환. 동일한 객체는 동일한 실행 세션 내에서 항상 같은 해시 값을 가지지만, 다른 세션에서는 해시 랜덤화로 인해 다른 해시 값을 가짐
-

Buckets

- 해시 테이블에서 특정 해시 값에 해당하는 위치

- 해시 함수에 의해 선택된 배열의 한 슬롯으로 각 버킷은 하나 이상의 key-value 쌍을 저장 가능

Hash Table의 장점

속도

- 삽입, 삭제, 검색에 O(1)소모, 데이터 크기가 클 경우와 삽입 삭제가 빈번할 경우 유리

메모리 사용의 효율성

- Hash Table의 동적 할당을 통해 유연한 메모리 사용 가능

간단함

- 구현이 간단하며 대다수의 언어에서 표준 라이브러리로 제공됨

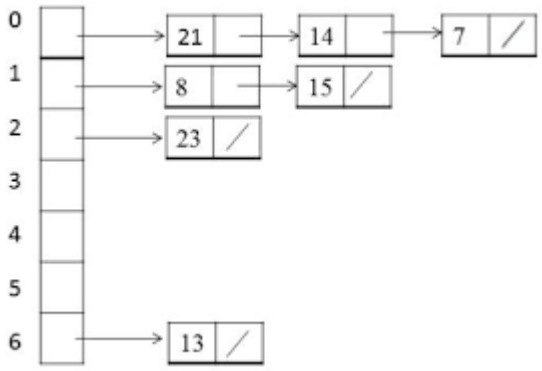
Hash Table 코드 구현 (Hash Function→ Mod)

```python
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size

    def hash_function(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash_function(key)
        if self.table[index] is None:
            self.table[index] = []
        self.table[index].append((key, value))

    def search(self, key):
        index = self.hash_function(key)
        if self.table[index] is None:
            return None
        for (k, v) in self.table[index]:
            if k == key:
                return v
        return None

    def delete(self, key):
        index = self.hash_function(key)
        if self.table[index] is None:
            return False
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                return True
        return False
```

Hash Collision

- 두 개 이상의 keys 가 동일한 해쉬 값(index)을 가질 경우 발생하는 문제
- 충돌 해결 방법

1. **Chaining**



▼ 삽입하고자 하는 위치(Bucket)에 데이터가 이미 존재할 경우, Linked List를 활용해 새로운 key-value 쌍을 삽입.

▼ Hash Table의 크기 고정, index 불변 → Clustering 문제 발생 X

2. **Open Addressing**

▼ Collision을 방지하기 위해 다른 빈 Bucket을 찾아 데이터를 저장

1. Linear Probing

- 선형적으로 빈 칸을 탐색하는 방식의 Hash Table Collison 회피 기법
- 데이터 삭제 후 선형적 탐색 불가 → 삭제한 흔적 남겨서 해결
- Clustering

```
def linear_probing(hash_table, key, value):
    index = hash_function(key)
    while hash_table[index] is not None:
        index = (index + 1) % len(hash_table)
    hash_table[index] = (key, value)
```

1. Quadratic Probing

- Collision이 발생했을 때 다음 index를 검사하는 간격이 제곱수로 증가하는 방식
- 같은 index를 가지는 keys들의 경우 간격이 동일함→ 충돌 해결 X

1. Double Hashing

```python
def double_hashing(hash_table, key, value):
    index = hash_function(key)
    step_size = second_hash_function(key)
    while hash_table[index] is not None:
        index = (index + step_size) % len(hash_table)
    hash_table[index] = (key, value)
```

- 두 개의 해시 함수를 사용하는 방식