1. 해시테이블

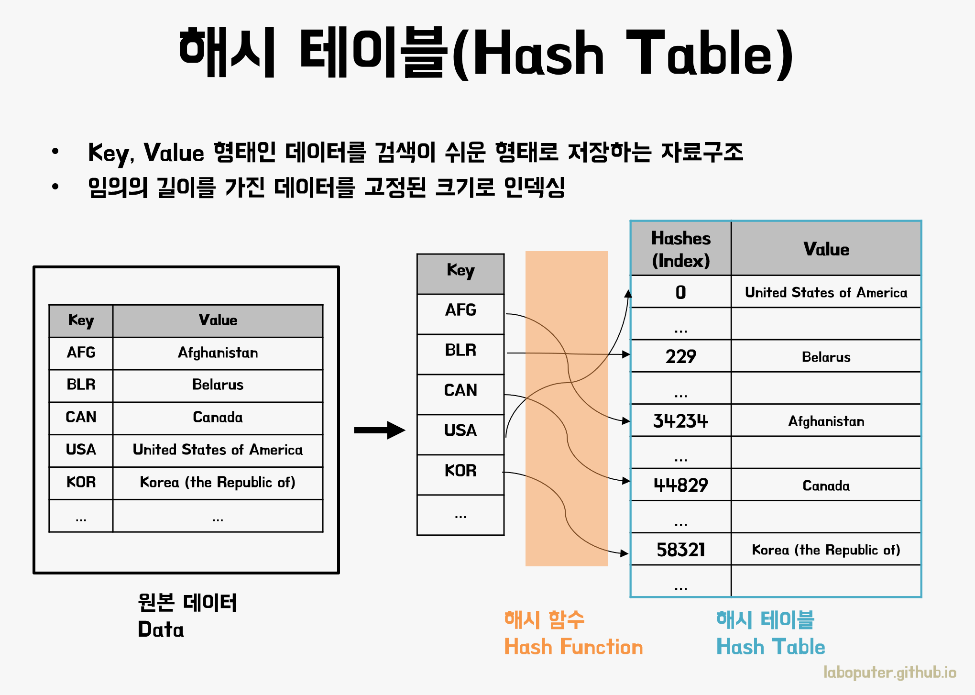
탐색 : O(1) (평균)

삽입 : O(1)

삭제 : O(1)

**키 값에 매핑할 수 있는 구조인, 연관 배열 추상 자료형을**

**구현하는** [**자료 구조**](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%A3%8C_%EA%B5%AC%EC%A1%B0)



2. 특징 및 장단점

분할상환분석에 따른 시간복잡도 O(1),

데이터 양에 관계없이 쉽고, 빠른 성능

해시 함수 값 충돌의 최소화

키의 모든 정보를 이용하여 해싱

(응용) 체크섬, 손실압축, 무작위화 함수, 암호, 파이썬 딕셔너리

3. 용어 설명

충돌 : 서로 같은 키들이 동일한 해시값

해싱 (hashing) : 키를 간단한 함수를 사용해 변환한 값을 리스트의 인덱스로 하여 항목을 저장하는 것

해시함수 (hash function) : 해싱에 사용되는 함수

해시값 (hash value, 해시 주소) : 해시함수가 계산한 값

해시테이블 (hash table) : 저장되는 1차원 리스트

\* 비둘기집 원리 : n개 item, m개 컨테이너 -> n>m이면 적어도 하나의 컨테이너에 반드시 2개

이상의 아이템이 들어있다는 원리

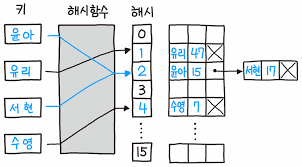
\* 로드 팩터 (load factor) : 해시 테이블에 저장된 데이터 개수 n을 버킷의 개수 k로 나눈 것

**Load factor = n / k**  -> 비율에 따라서 해시함수를 재작성 해야 될지,

해시테이블 크기를 조정해야 할지 결정

시간과 공간 비용의 적절한 절충안, 일반적으로 로드팩터 ↑, 테이블 성능 ↓

\* 해시 함수 : 다양한 알고리즘 有, \* 구글 : 딥러닝 이용 충돌 최소화 논문 발표

4. 종류 (충돌 해결)

가. 개별 체이닝 (separate chaining)

1) 충돌 발생 시 연결리스트로 연결하는 방식

2) 전통적 방식, 알고리즘 간단 -> 인기

가) 키 해시 값 계산 -> 해시 값 이용 배열 인덱스 구함 -> 같은 인덱스 연결리스트로 연결

3) 대부분의 탐색 O(1), 최악의 경우 (모든 해시 충돌 발생) O(n)

나. 개방주소방식 (open addressing)

1) 충돌된 키들을 해시테이블 전체를 열린 공간으로 여겨 비어있는 곳을 찾아 (probing)

항목을 저장하는 방식

2) 모든 원소가 반드시 자신의 해시 값과 일치하는 주소에 저장된다는 보장 없음

3) 단점 : 선형탐사의 경우 해시 테이블에 저장되는 데이터들이 고르게 분포되지 않고

뭉치는 경향. 테이블 여기저기에 연속된 데이터 그룹이 생기는 현상(클러스터링)

4) 일정 이상 채워지면 (즉, 기준이 되는 로드 팩터 비율을 넘어서면) 그로스 팩터(growth

factor)의 비율에 따라 더 큰 크기의 또 다른 버킷 생성 후 리해싱(rehashing) 작업

동적 배열에서 공간이 가득 찰 경우 더블링으로 새롭게 복사해 옮겨가는 과정과 유사

다. 폐쇄주소방식 (closed addressing)

1) 해시값에 대응되는 해시테이블 원소에 반드시 키를 저장 (충돌 시 동일한 곳 저장)

5. 어디에 쓸까?

가. 딕셔너리 사용하면 좋을 때

\* 검색을 많이 하는 구조

\* 상황에 맞는 해시 함수가 있을 때

나. 사용하면 안 좋을 때

\* 자료의 순서가 중요할 때

\* 공간의 효율성이 중요할 때