


Hash

# Index	0
📅 CreatedAt	@September 28, 2022
👤 Person	 Ally Hyeseong Kim
⋮ Status	In Progress
⋮ Tags	Hash Java Python
📅 UpdatedAt	@September 28, 2022

References

파이썬 알고리즘 인터뷰

2021 세종도서 학술부문 선정작. 현업과 실무에 유용한 주요 알고리즘 이론을 깊숙이 이해하고, 파이썬의 핵심 기능과 문법까지 상세하게 이해할 수 있는 취업용 코딩 테스트를 위한 완벽 가이드다. 200여 개가 넘는...

 <https://www.aladin.co.kr/shop/wproduct.aspx?ItemId=245495826>



References

1. Hash
2. 충돌의 발생: Birthday Problem
3. 충돌의 원리: Pigeon Principle
4. Load Factor
5. Hash Function: Modulo-Division Method
6. Hash Table 충돌 해결 방법
7. 언어별 Hash Table 구현 방법

Hash Table 혹은 **HashMap**은 key를 value에 매핑할 수 있는 구조인, 연관 배열 (Associative Array) 추상 자료형(Abstract Data Type, ADT)을 구현하는 자료구조이다. **Hash Table**은 대부분의 연산이 시간복잡도가 $O(1)$ 이다.

1. Hash

Hash Function은 임의 크기 데이터를 고정 크기 값으로 매핑하는데 사용할 수 있는 함수이다.

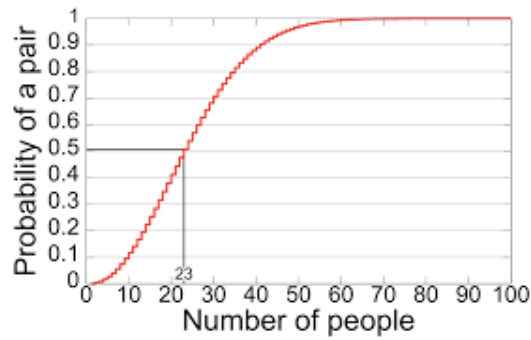
- **Hashing**: **Hash Table**을 인덱싱하기 위해 **Hash Function**을 사용하는 것



Hash Function 고려사항

- **Hash Function** 값 충돌 최소화
- 쉽고 빠른 연산
- **Hash Table** 전체에 **Hash** 값이 균일하게 분포
- 사용할 key의 모든 정보를 이용하여 **Hashing**
- **Hash Table** 사용 효율이 높은 것

2. 충돌의 발생: Birthday Problem



모인 사람 수에 따라 생일이 같은 2명이 존재할 확률

- 충돌은 생각보다 쉽게 일어나므로 충돌을 최소화하는 것은 중요하다.

3. 충돌의 원리: Pigeon Principle

Pigeon Principle 은 n 개의 아이템을 m 개의 컨테이너에 넣을 때, $n > m$ 이라면 적어도 하나의 컨테이너에는 반드시 2개 이상의 아이템이 들어있다는 원리이다.

- **Hash Function** 값이 충돌하면 추가 연산이 필요하므로 가급적 충돌을 최소화해야 한다.

4. Load Factor

Load Factor 는 **Hash Table** 에 저장된 데이터 개수 n 을 **Bucket**의 개수 k 로 나눈 것이다.

$$loadfactor = \frac{n}{k}$$

- **Load Factor** 비율에 따라 **Hash Table**의 크기를 조정할지 결정한다.
 - **Load Factor**가 증가할 수록 **Hash Table**의 성능이 점점 감소한다.
- **Load Factor**를 사용하여 **Hash Function**이 key를 잘 분산하는지에 대한 효율성을 측정할 수 있다.
- **Java**에서 **HashMap**의 **default Load Factor** 값은 0.75이다.
 - 이를 넘어갈 경우 동적 배열처럼 **Hash Table**의 공간을 재할당한다.

5. Hash Function: Modulo-Division Method

$$h(x) = x \mod m$$

- m : **Hash Table** 크기
- x : 간단한 규칙을 통해 만들어낸 랜덤한 상태의 key 값
 - *Joshua Bloch, Effective Java*

$$P(X) = S[0] + X^{(n-1)} + S[1] + X^{(n-2)} + \dots + S[n-1]$$

- *The C Programming Language*

$$x = P(31)$$

```
/* hash: 문자열 s에 대한 해시 값 구성 */
unsigned hash(char *s) {
    unsigned hashval;

    for (hashval = 0; *s != '\0'; s++) {
        hashval = *s + 31 * hashval;
    }
    return hashval % HASHSIZE;
}
```

```
hashCode = 31 * hashCode + (e == null ? 0 : e.hashCode());
```

6. Hash Table 충돌 해결 방법

6.1. Separate Chaining

Separate Chaining 은 **Hash Table** 의 기본 방식으로 충돌 발생 시 **Linked List**로 연결한다.

- 무한히 저장할 수 있다.



Separate Chaining 원리

1. key의 **Hash** 값을 계산한다.
2. **Hash** 값을 이용해 배열의 인덱스를 구한다.
3. 같은 인덱스가 있으면 **Linked List**로 연결한다.

- 탐색에서 best case는 $O(1)$, worst case는 $O(n)$ 이다.
- Java 8에서는 **Linked List**의 구조를 최적화하여 데이터 개수가 많아지면 **Red-Black Tree**에 저장하여 worst case에서 $O(\log n)$ 으로 탐색할 수 있다.

jdk9u/HashMap.java at b202236a513fd5c710733ef3ca2db720f03f67d2 · openjdk/jdk9u
https://openjdk.org/projects/jdk-updates last released 2018-01-16 - jdk9u/HashMap.java
at b202236a513fd5c710733ef3ca2db720f03f67d2 · openjdk/jdk9u
https://github.com/openjdk/jdk9u/blob/b202236a513fd5c710733ef3ca2db720f03f67d2/jdk/src/java.base/share/classes/java/util/HashMap.java

openjdk/jdk9u

https://openjdk.org/projects/jdk-updates last released 2018-01-16

343 Contributors 0 Issues 2 Stars 5 Forks

6.2. Open Addressing

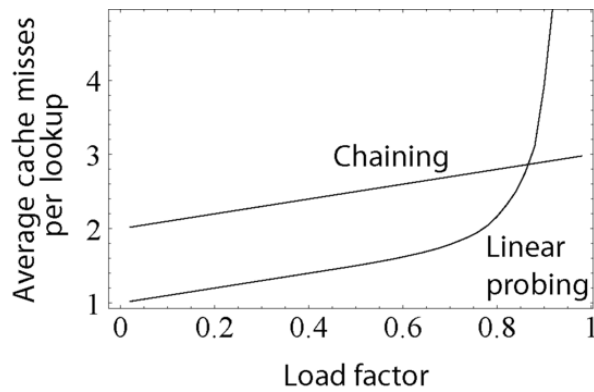
Open Addressing 은 충돌 발생 시 **Probing**을 통해 빈 공간을 찾아나선다.

- Bucket 사이즈보다 크면 삽입할 수 없다.
 - 기준이 되는 **Load Factor** 비율을 넘으면, **Rehashing** 작업이 일어난다.
 - **Rehashing** : **Growth Factor**의 비율에 따라 더 큰 크기의 또 다른 Bucket을 생성하여 새롭게 복사하는 작업
- 모든 원소가 자신의 **Hash** 값과 일치하는 주소에 저장된다는 보장이 없다.

Linear Probing 은 **Open Addressing** 방식 중에서 가장 간단한 방식이다. 충돌 발생 시 해당 위치부터 순차적으로 하나씩 **Probing**하고 비어 있는 공간에 삽입한다. (가장 가까운 빈 공간에 삽입된다.)

- **Hash Table**에 저장되는 데이터들이 고르게 분포되지 않고 뭉치는 경향이 있다.
 - **Clustering** : **Hash Table** 여기저기에 연속된 데이터 그룹이 생기는 현상
 - **Probing** 시간을 오래 걸리게 하고, 전체적으로 **Hashing** 효율을 떨어뜨린다.

7. 언어별 Hash Table 구현 방법



Chaining과 Linear Probing 방식의 Load Factor에 따른 성능 비교

- 빈 공간을 Probing하는 **Linear Probing** 방식은 공간이 찰수록 Probing 하는데 더 오랜 시간이 걸리고, 가득 찰 경우 더 이상 빈 공간을 찾을 수 없어 **Load Factor** 1 이상은 저장할 수 없다.
 - Modern Language (ex. Ruby, Python)는 **Open Addressing** 방식으로 성능을 높이고, **Load Factor**를 낮게 설정하여 성능 저하 문제를 해결한다.

7.1. C++(GCC libstdc++): Separate Chaining

- **Load Factor**는 1이다.

7.2. Java: Separate Chaining

- **Load Factor**는 0.75이다.

7.3. Go: Separate Chaining

- **Load Factor**는 6.5이다.

7.4. Ruby: Open Addressing

- **Load Factor**는 0.5이다.

7.5. Python: Open Addressing

- **Dictionary** 자료형은 **Hash Table**로 구현되어 있다.
- **Hash Table** 충돌 시 **Open Addressing** 방식으로 해결한다.
 - Chaining 시 malloc으로 메모리를 할당하는 오버헤드가 높다. *CPython*
- **Load Factor**는 0.66이다.