Design HashMap

# Index	706
CreatedAt	@September 28, 2022
Person	A Ally Hyeseong Kim
Status	In Progress
Tags	Hash Java Python
UpdatedAt	@September 28, 2022

References

Design HashMap - LeetCode

Design HashMap - Design a HashMap without using any built-in hash table libraries. Implement the MyHashMap class: * MyHashMap() initializes

https://leetcode.com/problems/design-hashmap/

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

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

1. Built-in HashTable Library

1.1. Python Dictionary

```
class MyHashMap:

    def __init__(self):
        self.my_hash_map = dict()
    def put(self, key: int, value: int) -> None:
        self.my_hash_map[key] = value

    def get(self, key: int) -> int:
        return self.my_hash_map.get(key, -1)

    def remove(self, key: int) -> None:
        if self.my_hash_map.get(key, -1) > -1:
            del self.my_hash_map[key]
```

- `del dict[key]`: remove `(key, value)` in `dict`

1.2. Java HashMap

```
class MyHashMap {

    Map<Integer, Integer> my_hash_map;

    public MyHashMap() {
        this.my_hash_map = new HashMap<>();
    }

    public void put(int key, int value) {
        this.my_hash_map.put(key, value);
    }

    public int get(int key) {
        return this.my_hash_map.getOrDefault(key, -1);
    }

    public void remove(int key) {
        if (this.my_hash_map.getOrDefault(key, -1) > -1) {
            this.my_hash_map.remove(key);
        }
    }
}
```

- `map.getOrDefault(key, defaultValue)`: return `defaultValue` if `key` is not in `map`

2. Separate Chaining

```
class MyHashMap {

    def __init__(self):
        self.size = 1000
        self.table = collections.defaultdict(ListNode)

    def put(self, key: int, value: int) -> None:
        index = key % self.size
        if self.table[index].value is None:
            self.table[index] = ListNode(key, value)
            return
        p = self.table[index]
        while p:
            if p.key == key:
                p.value = value
                return
            if p.next is None:
                break
            p = p.next
        p.next = ListNode(key, value)

    def get(self, key: int) -> int:
        index = key % self.size
        if self.table[index].value is None:
            return -1
        p = self.table[index]
        while p:
            if p.key == key:
                return p.value
            p = p.next
        return -1

    def remove(self, key: int) -> None:
        index = key % self.size
        if self.table[index].value is None:
            return
        p = self.table[index]
        if p.key == key:
            self.table[index] = ListNode() if p.next is None else p.next
            return
        prev = p
        while p:
            if p.key == key:
                prev.next = p.next
                return
            prev, p = p, p.next
```