Hi!

Anuj Kumar

-> 2016 NITK
-> IIT Madras Internship <2014>
-> Intuit Internship <2015>
-> PTG @Adobe <2017- ...>

-> Web development, Front end, Backend, Flutter, Data structures, System Design, Crypto, Block chain, Game Development, ...
-> Freelancing & Teaching
-> Designing, Sketching, Photography, Editing, Video editing...

SURVEY-

https://forms.gle/extuzfyYMY8bQ7wk8

HashMaps in Java

Hash Tables →

Data structure which organizes data using hash functions to support quick insertion and search.

- ~~Hash set~~ ✗
- Hash Map → one of the implementations of a map data structure to store (key, value) pairs.

0 1 2
[2, 3, 4]

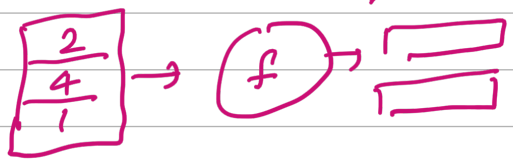
Standard template libraries in programming languages makes it easy to use them.

Choosing a proper hash function, the hash table can achieve wonderful performance in both insertion and search.

??

→ Principle of a hashtable ✓
→ Design a hashtable ✓
→ ~~Hashset~~ ✗
→ Hashmap - Aggregate information by key ✓
→ Design a proper key when using a hash table ✓

Principle of a hash table -

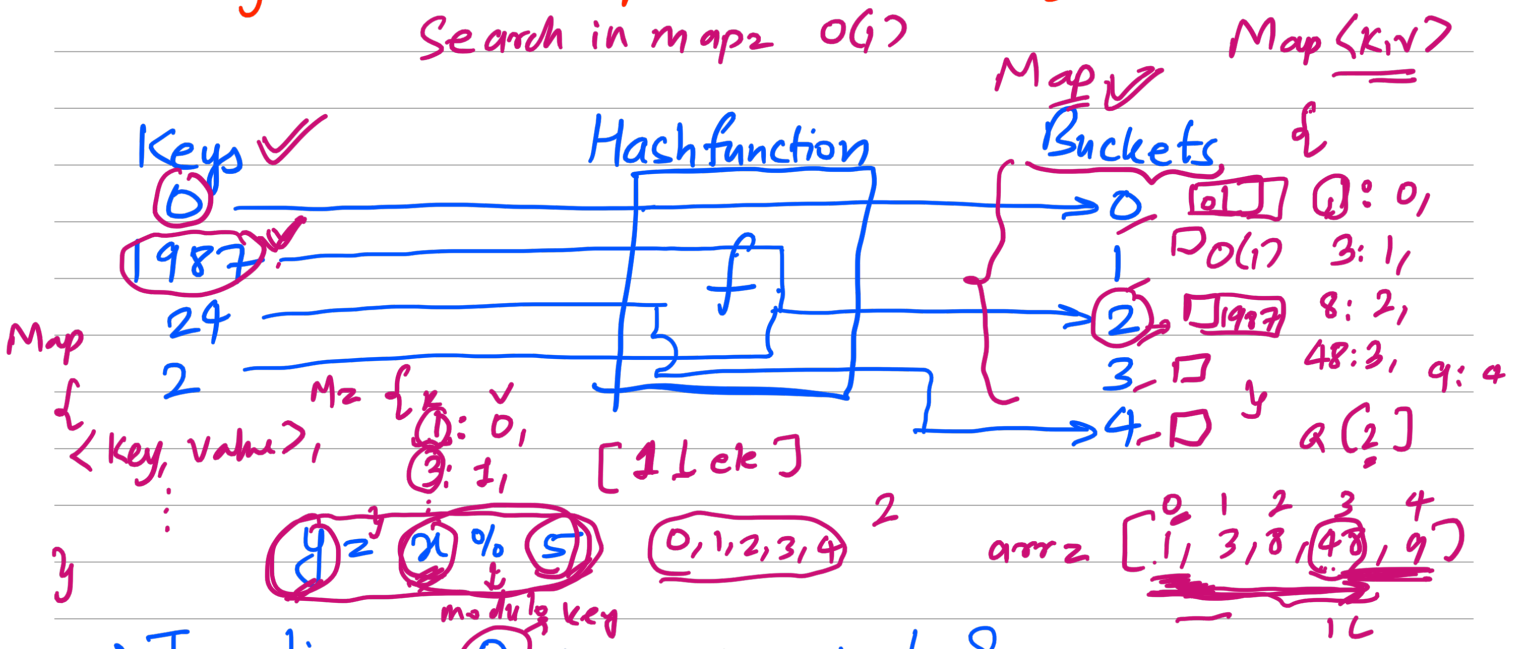


Idea → Use a hash function to map keys to buckets. ??

→ insert a new key, hash function decides which bucket the key should be assigned and the key will be stored in the corresponding bucket

→ Search for key, we use the same hashfunction to find the corresponding bucket and search only in the specified bucket

Search in map2 O(1)



→ Insertion → 0 is assigned to ?

→ Searching → Search for 1987. to be found in

1987 is found in which bucket? We search in ___ bucket and we successfully find out 1987 in that bucket.

→ Search for 23. ...

Design a hash table -

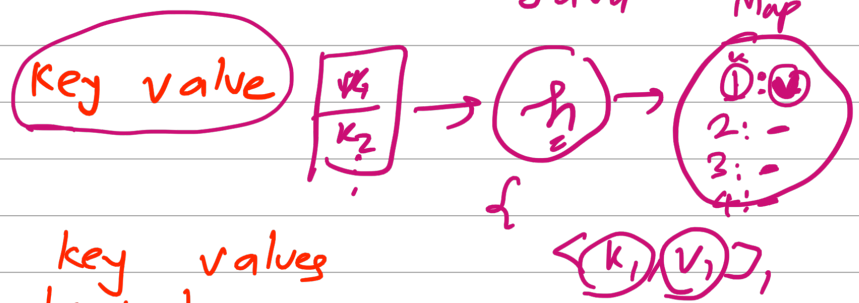
→ Hash function
- most important component
- map the key to a specific bucket //
- y = x % 5

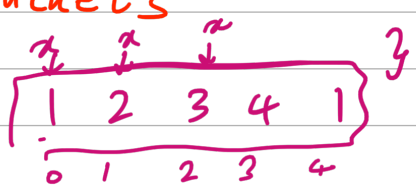
index of assigned bucket

→ range of key values

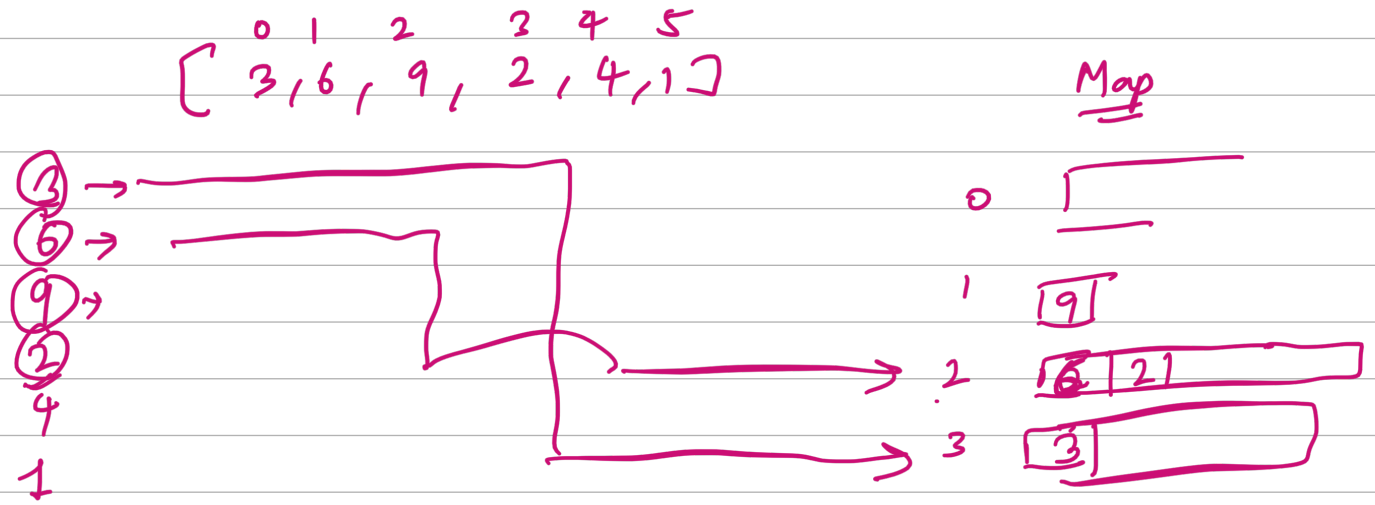
→ number of buckets

Java





Set



$y = x \% 4 \quad 2 \% 4 = 2$

Map

{

<key1: value1>

<key2: value2>

}

are [0 1 2 3
2, 4, 6, 8]

Map {

2 : 0,

4 : 1,

6 : 2,

8 : 3

}

Key type	Key Range	No. of buckets	Hash fn example
integer ✓	0 to 1,00,000	1000	yzx % 1000
char	'a' to 'z'	26	yzx - 'a'

- Open problem <design a hash function>
- idea - assign the key to bucket as uniformly as we can
- one-to-one mapping <ideal>
- tradeoff between amount of buckets and the capacity of a bucket.

→ Collision Resolution

→ Ideal - one-to-one mapping <no collisions>
→ yzx % 5

1987, 2 → ? | Collision

Collision resolution algorithm
- organize the values in the same bucket?
- what if too many values are in same bucket?
- Search for a target value in a specific bucket?

Capacity of bucket | No. of keys which might be mapped into the same bucket according to our hash function

Let's say, a bucket, it holds the maximum number of keys <N>

If N is constant and small, we can just used an array to store keys in the same bucket.

If N is variable and large, height balanced binary search tree

Exercise-

Implement a basic hashtable
- requirement
- hash function
- collision

- Insertion, Searching
- Remove

~~Design hash set~~ -

Design hash map -

- No built in hash table libraries

class NewtonHashMap -

→ NewtonHashMap() initialises the object with an empty map

→ void put(key, value) insertion. If the key already there, updak value
(key: int, value: int)

→ int get(key) → value <-1 if not found, else return value>
(key: int; value: int)
get("A") → 20

→ void remove(key) → remove the key and its value if its there
(key: int)

M { A: 20, B: 27 }

get("A") → 20 remove("A");

name: age

{ "A": 20, "B": 27 }

put("A", 20)
put("B", 27)

put(3, 0);

put(4, 1);
put(4, 3);

{ k1: v1, k2: v2 }

<k, v>

{ 3: 0, 4: 3 }

get(3);

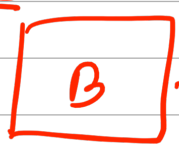
Solution →

→ array to represent hashmap
→ Each element is a bucket
→ Each bucket uses the arraylist

```
List<Pair<Int,Int>>[] map; // hashmap implemented by array
int MAX-LEN = 100000; // amount of buckets

int getIndex(int key){
    return key % MAX-LEN;
}
// search key in a given bucket
int getPos(int key, int index){
    List<Pair<Int, Int>> temp = map[index];

    if(temp == null){
        return -1;
    }

    for(int i=0; i< temp.size(); i+=1){
        if(temp.get(i).getKey() == key){
            return i;
        }
    }
    return -1;
}

NewtonHashMap(){
    map = (List<Pair<Int, Int>>[]) new ArrayList
                                    [MAX-LEN];
}
```

```
void put(key, value) {
    int index = getIndex(key);
    int pos = getPos(key, index);
//insert    if (pos < 0) {
                if (map[index] == null) {
                    map[index] = new ArrayList
                                 <Pair<Int, Int>>();
                }
                map[index].add(new Pair(key, value));
//update    } else {
                map[index].set(pos, new Pair(key, value));
        }
}

int get(key) {
    int index = getIndex(key);
    int pos = getPos(key, index);
    if (pos < 0) {
            return -1;
    } else {
            return map[index].get(pos).getValue();
    }
}

void remove(key) {
    int index = getIndex(key);
    int pos = getPos(key, index);
    if (pos >= 0) {
            map[index].remove(pos); // O(n)?
    }
}
```

Remove -



→ n values

① → move $n-1$ values to left

② → $n-2$

② → $n-3$

⋮

$$\frac{(n-1)+(n-2)+\ldots+1+0}{n}$$

$$\Rightarrow \frac{\frac{n}{2}(2.0+(n-1)1)}{n} \qquad S_n = \frac{n}{2}(2.a+(n-1)d)$$

$$\Rightarrow \frac{n-1}{2}$$

$O(n)$

Can we do better?

- Swap the element which we want to remove with last element in bucket. Then remove the last element
- Linked List... remove the element in O(1) time complexity...

Complexity Analysis -

Space

→ M keys, O(M) space

Time

→ design

→ 1. bucket size is small and constant;

Insertion → O(1)

Search → O(1)

2. Maximum bucket size N

Insertion → O(1)

Search → O(N)

Few points to ponder on the inbuilt HashMap-

↳ Key → hashcode → index of bucket

↳ Each bucket contains an array to store all values in same bucket initially

↳ If there are too many values in the same bucket, it is maintained in a height balanced BST.

Avg. time complexity
- Insertion → O(1)
- Search → O(1)

Worst time complexity
- Insertion → O(log N)
- Search → O(log N)

So, it is a tradeoff between insertion and search.

HashMap → JAVA →

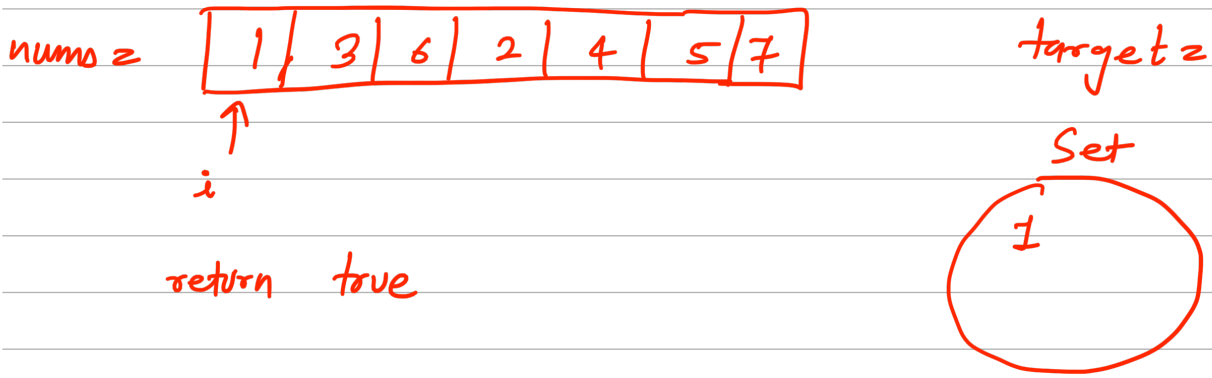
```
// initialize
Map<Int, Int> hashmap = new HashMap<>();

// insert a new (key, value) pair
hashmap.putIfAbsent(0,0);
hashmap.putIfAbsent(2, 3);

// insert a new (key, value) pair or update the value of existed key
hashmap.put(1,1);
hashmap.put(1, 2);

// get the value of specific key
System.out.println("The value of key 1 is: " +
                   hashmap.get(1));
```

```
// delete a key
hashmap.remove(2);

// check if a key is in the hash map
if(!hashmap.containsKey(2)) {
        System.out.println("key 2 is not in the
                            hashmap");
}

// get the size of hashmap
System.out.println("The size of hashmap is: " +
                   hashmap.size());

// Iterate the hashmap
for(Map.Entry<Int, Int> entry: hashmap.entrySet()) {

        System.out.println("(" + entry.getKey() +
                           ", " + entoy.getValue() + ")");
}
System.out.println("are in the hashmap.");

// clear the hashmap
hashmap.clear();

// check if the hashmap is empty
if(hashmap.isEmpty()) {
        System.out.println("hash map is empty now!");
}
```

Two Scenarios in Hashmap -

1. → Need more information rather than only the key. Then we can build a mapping relationship between key and information by hashmap.

Given an array of integers, return indices of the two numbers such that they add a specific target



But,
we are asked to return more information which means we not only care about the value but also about index as the value.
Therefore,
we should use a hashmap.

- Some cases, we need more information not just to return more information but also to help us with our decisions

- In above, we found it in map, we return that information immediately. But sometimes, we might want to check if the value of the key is accepted first.

Template to solve for these kinds of problems

```
ReturnType solve(List<Type> keys) {
    Map<Type, InfoType> hashmap = new HashMap();
    for(Type key: keys) {
        if(hashmap.containsKey(key)) {
```

```
            if (hashmap.get(key) satisfies
                the req.) {
                return needed_info;
            }
        }

        hashmap.put(key, value);
    }
    return needed_info;
}
```

Problems-

① → Two Sum →

nums = [2, 7, 11, 15] target = 9
→ [0, 1]

nums = [3, 2, 4] target = 6
→ [1, 2]

nums = [3, 3] target = 6
→ [0, 1]

```
int[] twoSum(int[] nums, int target) {
    Map<Int, Int> hashmap = new HashMap();

    for(int idx=0; idx < nums.length; idx+=1) {

        int rem = target - nums[idx];

        if (map.containsKey(rem)) {
            return new int[] {
                map.get(rem),
                idx
            };
        }
    }
```

```
            map.put(nums[idx], idx);
        }

        return new int[] {
            -1, -1
        };
    }
```

T: O(n), S: O(n)

② Isomorphic strings-

s: "egg" t: "add"
→ true

s: "foo" t: "bar"
→ false

s: "paper" t: "title"
→ true

```
boolean isIsomorphic(s, t) {

    int[] mapDictSToT = new int[256];
    int[] mapDictTToS = new int[256];

    Arrays.fill(mapDictSToT, -1);
    Arrays.fill(mapDictTToS, -1);

    for(int i=0; i< s.length(); i+=1) {
        char c1 = s.charAt(i);
        char c2 = s.charAt(i);
        //1. No mapping exists
        if(mapDictSToT[c1] == -1 &&
           mapDictTToS[c2] == -1) {
            mapDictSToT[c1] = c2;
            mapDictTToS[c2] = c1;
        }
```

```
else if(!(mapDictSToT[c1] == c2 &&
          mapDictTToS[c2] == c1)) {
    return false;
}
}
return true;
}
```

T = O(n)
S = O(1)

s = paper
t = title

p → t
a → i
e → l
r → e

b a d c
b a b a

b → b
a → b
d → b

b → b
a → a

③ → Minimum index sum of two lists –

list1 = ["Shogun"(0), "Tapioca Express"(1), "Burger King"(2), "KFC"(3)]
list2 = ["Piatti"(0), "The Grill at Torrey Pines"(1), "Shogun"(2), "Hungry Hunter Steakhouse"(3)]
→ ["Shogun"]

list1 = ["Shogun"(0), "Tapioca Express"(1), "Burger King"(2), "KFC"(3)]
list2 = ["KFC"(0), "Shogun"(1), "Burger King"(2)]

→ ["Shogun"] 0+1 = 1