Hashing 3 : Internal Implementation

Madhan Kumar M S
Abhishek Sharma
Akansh Nirmal
amit khandelwal
Balaji S K
Bhaveshkumar
Burhan
Gagan Kumar S
Gowtham
Hemant Kumar
Ishan
Murali Mudigonda
Nikhil Pandey
Pankaj Bhanu
Purusharth A
Rajat Sharma
Rajendra
Rathna
Sanket Giri
Saurabh Ruikar
Shani Jaiswal
sharath r
Shradha Srivastava
Sneha L
Sridhar Hissaria
Subhashini
Subhranil Kundu
Sumit Adwani
Suyash Gupta
thulasi babu
Vasanth
Yugesh v

- check if given element exists in Q queries
- Issues with DTA
- collision Resolution
- Chaining
- Code Implementation walk through

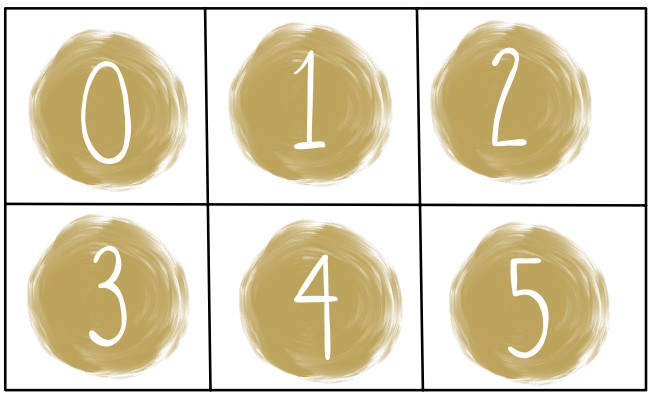
Current PSP

61 % ⟶ 70%.

Rules ⟶ Q ⟶ Q tab

⟶ answers ⟶ private

Hotel



Register

Room NO	Availabl
0	✓
1	✓
2	X
3	✓

Q> Check if any particular room is available ?

Maintain the info in a register.

↗ N

Ques > Given an int array A[] & multiple queries Q[]

	0	1	2	3	4	5	6
A =	2	4	11	6	8	9	1

Q

Queries

X = 10 ⟶ any = false

= 2 ⟶ any = true

Bruteforce

∀ X in queries . Linear search for X in A[]

TC: O(N * Q)

SC: O(1)

Idea 2

Create an array for checking if a value is present or not.

Size = max(A) + 1

∵ val of A need to be used as index.

	0	1	2	3	4	5	6
A =	2	4	11	6	8	9	1

TC: O(N) To find max.

TC: O(max(A))

	0	1	2	3	4	5	6	7	8	9	10	11
dat =		■	■		■		■		■	■		■

Direct Access Table

Queries

x = 10 ⟶ any = dat[10] ⟶ false
= 2 ⟶ any = true

} O(Q)

Per query TC : O(1)

Overall TC : O(N + max(A) + Q)

space O(max(A))

Issues with DAT { Direct Accen Table }

⟶ Wastage of Space

⟶ A[i] ⟶ 10^9 { cannot initialize }

⟶ Handling of -ve

↓

≃ 200 MB

int [10^9] ⟶ 4 * 1000 * 1000 * 1000

4 KB

4 MB

4 GB

How to overcome the issues ?

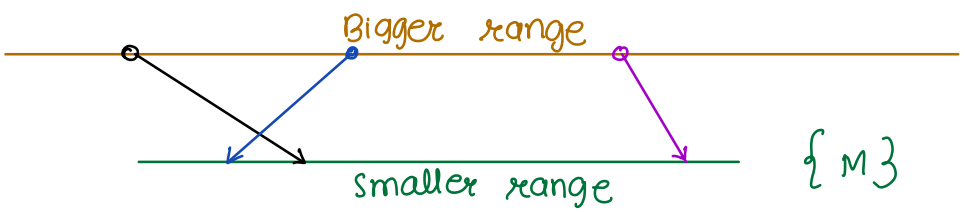
	0	1	2	3	4	5	6
A[] =	21	42	37	45	99	30	47

Assume memory limit M = 10 x % 10 = [0, 9]

	0	1	2	3	4	5	6	7	8	9
dat	30	21	42			45		37		99
								47		

37 % 10 = 7
47 % 10 = 7
} collision

Can we avoid collision ? No

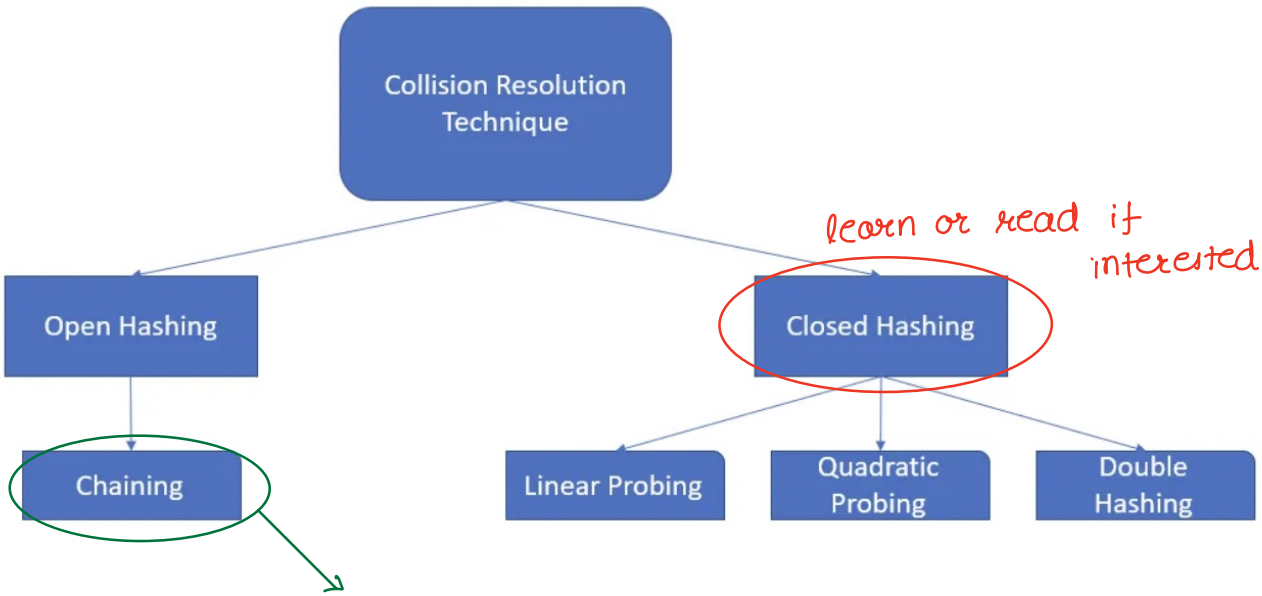


5 pigeons 4 holes

Pigeon Hole Principle

⟹ There will be one hole with > 1 pigeon inside it

Handle Collision



learn or read if interested

Asked in interviews

	0	1	2	3	4	5	6	7
A[] =	21	42	37	45	99	30	47	55

Assume memory limit M = 10 x % 10 = [0, 9]

	0	1	2	3	4	5	6	7	8	9
bucket	30	21	42			45		37		99
						55		47		

37 % 10 = 7
47 % 10 = 7
} collision

Chaining → If there is a collision keep track of values in a linear DS.

Array of ArrayList

M is fixed

elements per index is variable

Query

X = 32

steps index = X % 10 = 32 % 10 = 2

Linear search for X = 32 in AL at index 2

Worst case scenario ?

	0	1	2	3	4	5
A[]	31	41	51	61	71	81

0	1	2	3	4	5	6	7	8	9

bucket

31 → 41 → 51 → 61 → 71 → 81

TC per query ⟶ O(N)

TC per query in average case ⟶ $\frac{\#\text{elements inserted}}{\text{bucket size}}$

λ lamda = N/M

{load factor}

M = 10		λ
inserted	= 8	$\frac{8}{10} = 0.8$
	= 80	$\frac{80}{10} = 8$
	= 800	$\frac{800}{10} = 80$
	= 8000	$\frac{8000}{10} = 800$

whenever the value of λ reaches some threshold we double the value of M and re-hash all the existing values.

	0	1	2	3	4	5	6	7
A[] =	21 ✓	42 ✓	37 ✓	45 ✓	99 ✓	30 ✓	47	55

M = 5 threshold for $\lambda = 1$

buckets (old)	0	1	2	3	4
	45	21	42		99
	30		37		

$$\lambda = \frac{\#\text{ elements inserted}}{M} = \frac{6}{5} > 1$$

Now rehash all existing values with M = 2 * 5 = 10

TC : O(N)

	0	1	2	3	4	5	6	7	8	9
bucket	30	21	42			45		37		99

new

$$\lambda = \frac{6}{10} = 0.6$$

TC per query $= \lambda \simeq$ constant

Break 22:30

Datatype

```java
class HMNode<K, V> {
    K key;
    V value;

    public HMNode(K key, V value) {
        this.key = key;
        this.value = value;
    }
}
```

```java
class HashMap<K, V> {

    private ArrayList<HMNode>[] buckets;
    private int size; // number of key-value pairs

    public HashMap() {
        initBuckets();
        size = 0;
    }

    private void initBuckets() {
        buckets = new ArrayList[4];
        for (int i = 0; i < 4; i++) {
            buckets[i] = new ArrayList<>();
        }
    }
```

Array of AL

M

initial M

```java
public void put(K key, V value) {
    int bi = hash(key);
    int di = getIndexWithinBucket(key, bi);

    if (di != -1) {
        // Key found, update the value
        buckets[bi].get(di).value = value;
    } else {
        // Key not found, insert new key-value pair
        HMNode newNode = new HMNode(key, value);
        buckets[bi].add(newNode);
        size++; // no. of unique keys in HM

        // Check for rehashing
        double lambda = size * 1.0 / buckets.length;
        if (lambda > 2.0) {
            rehash();
        }
    }
}
```

bucket index

data index

di == -1

N

M

double division

```java
private int hash(K key) {
    int hc = key.hashCode();
    int bi = Math.abs(hc) % buckets.length;
    return bi;
}
```

generates a big no. for that key

val % M

Linear search key inside arrayList at B[bi]

AL at index bi

```
private int getIndexWithinBucket(K key, int bi) {
    int di = 0;
    for (HMNode node : buckets[bi]) {
        if (node.key.equals(key)) {
            return di; // Key found
        }
        di++;
    }
    return -1; // Key not found
}
```

```
private void rehash() {
    ArrayList<HMNode>[] oldBuckets = buckets;
    initBuckets(2*buckets.length);
    size = 0;
                    // call put for all old nodes inserted
    for (ArrayList<HMNode> bucket : oldBuckets) {
        for (HMNode node : bucket) {
            put((K) node.key, (V) node.value);
        }
    }
}
```

bucket index

search for key in bucket[bi]

```
public boolean containsKey(K key) {
    int bi = hash(key);
    int di = getIndexWithinBucket(key, bi);

    return di != -1;
}
```

if k is present return val else null

```java
public V get(K key) {
    int bi = hash(key);
    int di = getIndexWithinBucket(key, bi);

    if (di != -1) {
        return (V) buckets[bi].get(di).value;
    } else {
        return null;
    }
}
```

```java
public V remove(K key) {
    int bi = hash(key);
    int di = getIndexWithinBucket(key, bi);

    if (di != -1) {
        // Key found, remove and return value
        size--;
        return (V) buckets[bi].remove(di).value;
    } else {
        return null; // Key not found
    }
}
```

AL

```java
public int size() {
    return size;
}
```

↗ keyset

```java
public ArrayList<K> keyset() {
    ArrayList<K> keys = new ArrayList<>();
    for (ArrayList<HMNode> bucket : buckets) {
        for (HMNode node : bucket) {
            keys.add((K) node.key);
        }
    }
    return keys;
}
```

TC: O(N)