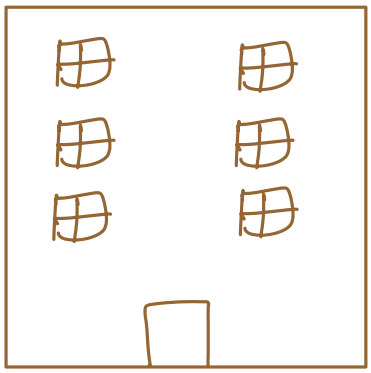
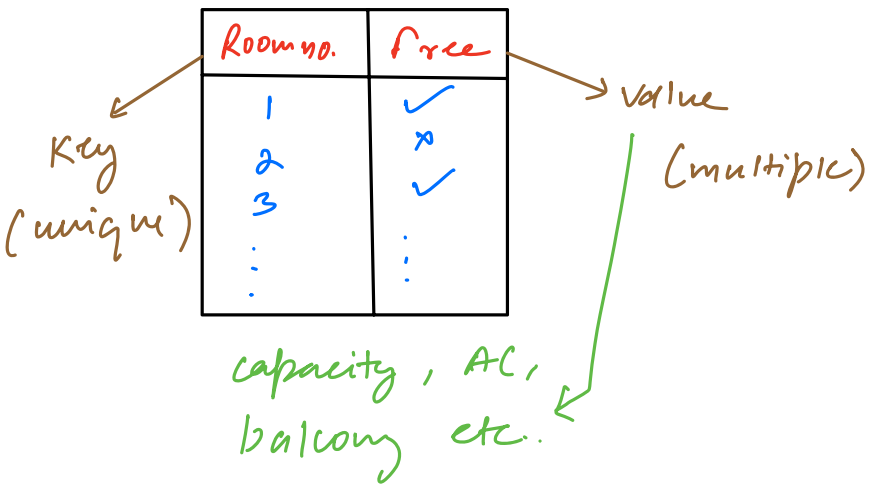
Hashing 3 : Internal Implementation & Problems



mohan

Check if any particular room is available ?



Ques → Given an integer array & multiple queries.

for every query → check if element X is present in array ?

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

Query

x = 10 ans = false

2 ans = true

Brute force → if query, travel the array & check.

TC = O(Q*N) SC = O(1)

Sol 2 → if A[i], mark them visited in a seperate array.

Direct Access Table (DAT)

data[i] → true ⇒ element 'i' is present in A
→ false ⇒ element 'i' is not present in A

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

→

data = [F T T F T F T F T T F T]
0 1 2 3 4 5 6 7 8 9 10 11

if, data[i] = false

for (i = 0 to N-1) {

    data[A[i]] = true

}

TC = O(N)

Query X → ans = data[X]

↳ TC = O(1) per query

total TC = O(N+Q)

SC = O(max(A[i]))

Max array size allowed = 10^6-10^7

10^7 → int

$10^7 \times 4B$

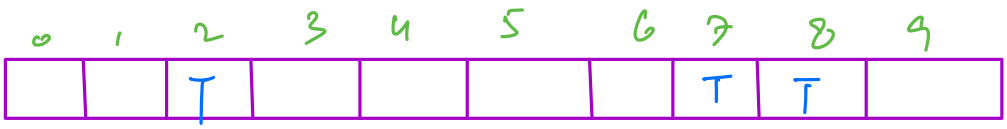
$= \frac{4 \times 10^3 \times 10^3 \times 10^1}{KB}$ $= 40\ MB$

MB

if $0 <= A[i] <= 10^9$ → MLE error

Ques → If memory limit is M, store if A[i] that it is present in available memory limit.

M = 10



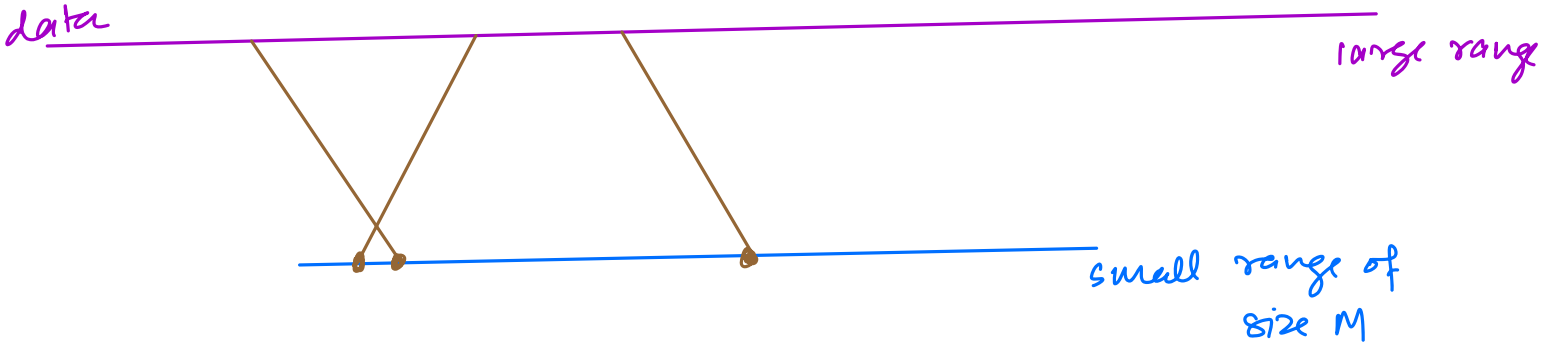
$0 <= h(A[i]) <= 9$

hash function → A[i] % 10 (↑ M)

A[i]	h(A[i])
27	27 % 10 = 7
18	18 % 10 = 8
32	32 % 10 = 2
37	37 % 10 = 7

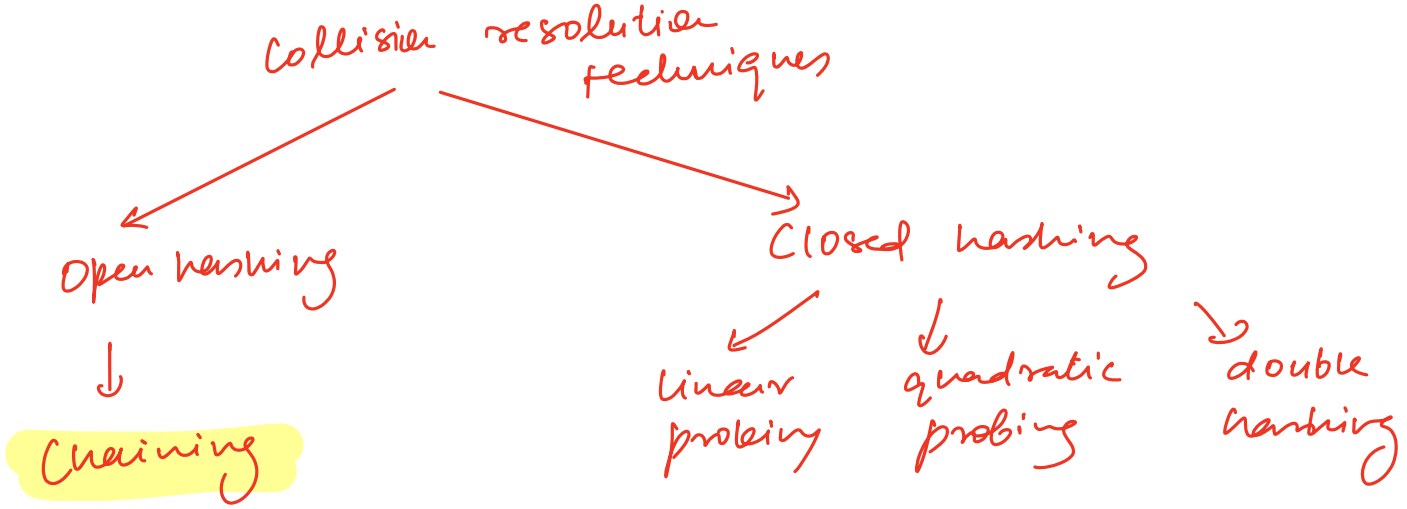
Collision

Can we avoid collision ? ⟶ NO

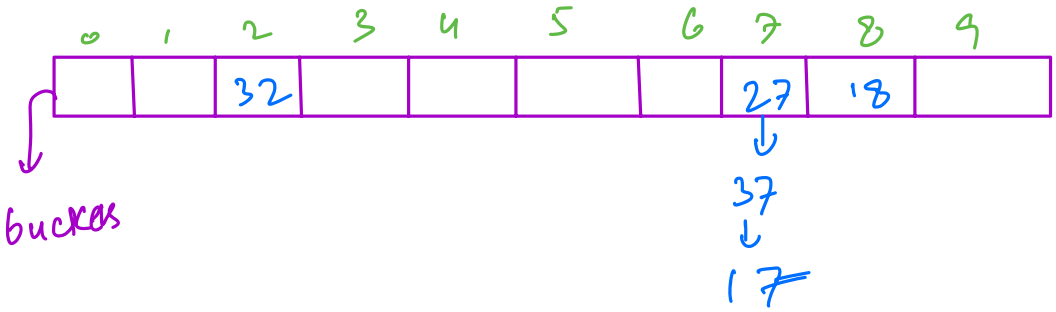


Pigeon Hole Principle → If there are N pigeons & (N-1) holes then there will be atleast 1 hole with more than 1 pigeon.

Can we handle collision ? → Yes



Chaining



A[i]	h(A[i])
27	27%10 = 7
18	18%10 = 8
32	32%10 = 2
37	37%10 = 7
17	17%10 = 7

TC per query > O(1)

(worst case TC = O(N))

TC on average <= λ (lambda)

(load factor) λ = # elements inserted → 8 / size of data array (M) → 10

$\lambda = \frac{8}{10} = 0.8$

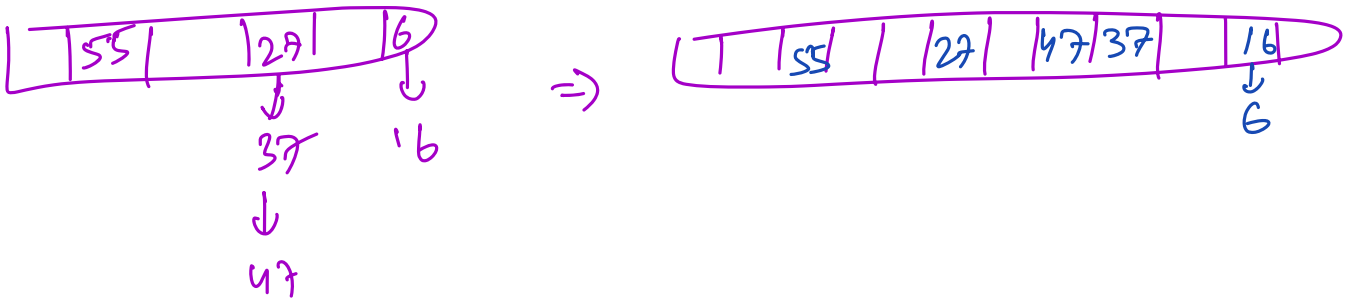
80/10 = 8

800/10 = 80

There is a predifined threshold for λ = 0.7

If λ becomes greater than threshold =>

Redistribute (TC = O(N)) the existing elements on a new data array with double size. λ → half



Code Implementation

```
class HashMap<K,V> {

    private class HMNode {
        K key;
        V value;
        public HMNode(key, val){
            this.key = key
            this.value = val
        }
    }

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

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

```
private void initBuckets() {
    buckets = new ArrayList[4]; ← initial size of hashmap
    for (i=0 to 3)
        buckets[i] = new ArrayList<>();
  }
}
```

---

put(2,3) => 2→3
put(2,4) => 2→4

Insertion

```
void put(K key, V value) {
    int bi = hash(key); → index of bucket
    int di = getIndexWithinBucket(key, bi);

    if (di != -1) { → key is present
        buckets[bi].get(di).value = value

    }
    else {
        HMNode temp = new HMNode(key, value)
        buckets[bi].add(temp);
        size++;
                                  for decimal
        lambda = size * (1.0) / buckets.length
        if (lambda > 0.7)  // rehashing
            rehash();
    }
```

```
    }
}
```

```
private int hash(K key) {
    int hc = key.hashCode()
    return hc % buckets.length
}
```

```
private int getIndexWithBucket(K key, int bi) {
                                  ↳ TC = O(len of arraylist)
    int di = 0
    for (HMNode node : buckets[bi]) {
        if (node.key.equals(key))
            return di;
        }
        di++
    }
    return -1    // key not found
}
```

```
private void rehash() {
    ArrayList<HMNode>[] oldBuckets = buckets.
    buckets = new ArrayList[oldBuckets.length*2]
    // copy old bucket to new bucket
}
```

```
public V get(K key) {
    bi: hash(key)
    di: getIndexWithinBucket(key, bc);
    if (di == -1) {
        return null
    }
    else {
        return buckets[bi].get(di).value
    }
}
```

---

public bool containKey() ⇒ TODO

```
public V remove(K key) {
    bi: hash(key)
    di: getIndexWithinBucket(key, bc);
    if (di == -1) {
        return null
    }
    else {
        size--
```

```
        return buckets[bi].remove(di).value;
    }
}
```

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

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