Data Structure & Algorithms

Nilesh Ghule

Hash Table

- Associative data structure → key-value pair (association)
- Stores key-value so that for a given key, value can be searched in fastest possible time. Ideal time complexity is O(1).

↳ no collision

- Example:

	arr
0	1 | A ....
1	2 | B ...
2	
⋮	
11	12 | P ...
12	13 | Q ...
13	
⋮	
28	29 | Y ...
29	30 | Z ...

hash fn

$f(r) = r-1$

roll no = r

index = r-1

```
put(r, stud):
    arr[r-1]=stud;

get(r):
    return arr[r-1];
```

programming language terms		data structure & algorithms terms
① array	→	① table
② index	→	② slot
③ math fn to decide the index in which value to be stored	→	③ math fn of key to decide the slot of table in which value to be stored

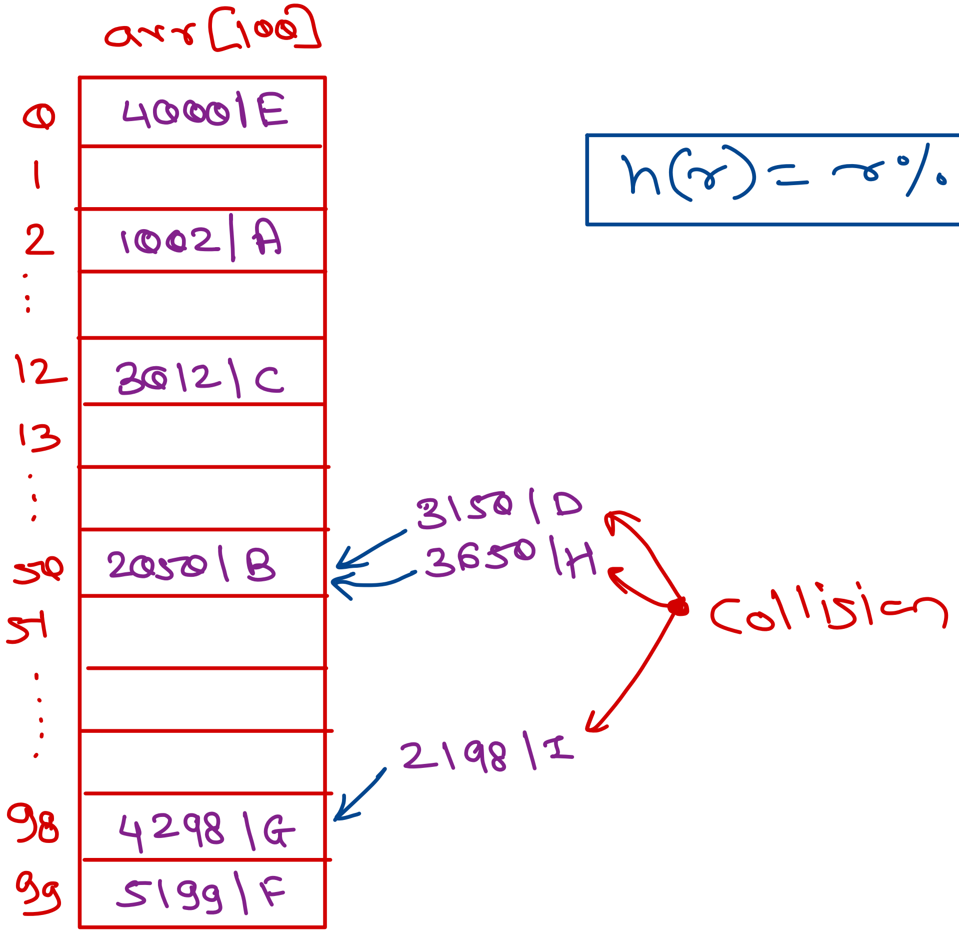
f(k)

hash fn

Hash Table

- Hash Function is math function of key, that yields slot in the table.
- If different keys resulting in same slot in the table, it is called as collision.

arr[100]



0	4000 | E
1	
2	1002 | A
⋮	
12	3012 | C
13	
⋮	
50	2050 | B
51	
⋮	
98	4298 | G
99	5199 | F

$$h(r) = r \% 100$$

- ✓ 1002 | A
- ✓ 2050 | B
- ✓ 3012 | C
- ✓ 3150 | D
- ✓ 4000 | E
- ✓ 5199 | F
- ✓ 4298 | G
- ✓ 3650 | H
- ✓ 2198 | I

Hash Table

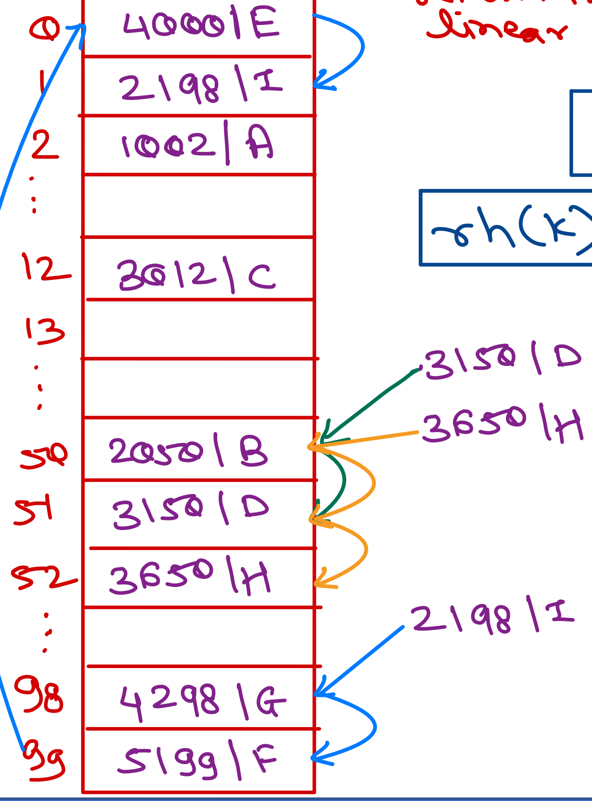
- Collision handling methods: Open addressing or Chaining
- Open addressing
  - Rehashing: Linear probing, Quadratic probing, …

h(k) → slot of table
if slot is already occupied, call another math fn to find next possible slot. (Repeat it until an empty slot is found).
Rehash fn
e.g. $rh = (h(k)+1) \% size$
or $rh = (rh(k)+1) \% size$

↳ rehash fn is math fn linear equation.

↳ rehash fn is math fn quadratic equation.

arr[100]



$$h(r) = r \% 100$$

$$rh(k) = (prev\ slot + 1) \% 100$$

- ✓ 1002|A
- ✓ 2050|B
- ✓ 3612|C
- ✓ 3150|D
- ✓ 4000|E
- ✓ 5199|F
- ✓ 4298|G
- ✓ 3650|H
- ✓ 2198|I

```
KeyValue arr[100];
 ↳ fields: key, value.

put(k,v):
    index = k % 100; //hash fn
    while(arr[index] != null)
        index = (index + 1) % 100; //rehash
    arr[index] = (k,v);

get(k):
    index = k % 100; //hash fn                 → cnt = 1;
    while(arr[index] != null) {                → if (cnt > 100)
        if (arr[index].key == k)                    return null;
            return arr[index].value;
        index = (index + 1) % 100; //rehash    → cnt ++;
    }
    return null; //no kv found.
```

Hash Table

- Load factor = Number of entries / Number of slots

key-value pairs

array size

- Cases
  - Load factor < 1
  - Load factor = 1
  - Load factor > 1

open addressing:

✓ all entries (kv) are stored in the array/table.

✓ storage is internal to the table.

✓ if occupied / not match, then find
↳ insert ↳ find
the next address (infinitely), until
empty / desired slot found,
↳ insert ↳ find

arr[100]
kv pairs = 75
entries < arr size
L.F. < 1

arr[100]
kv pairs = 100
entries == arr size
L.F. = 1

arr[100]
kv pairs = 120
entries > arr size
L.F. > 1

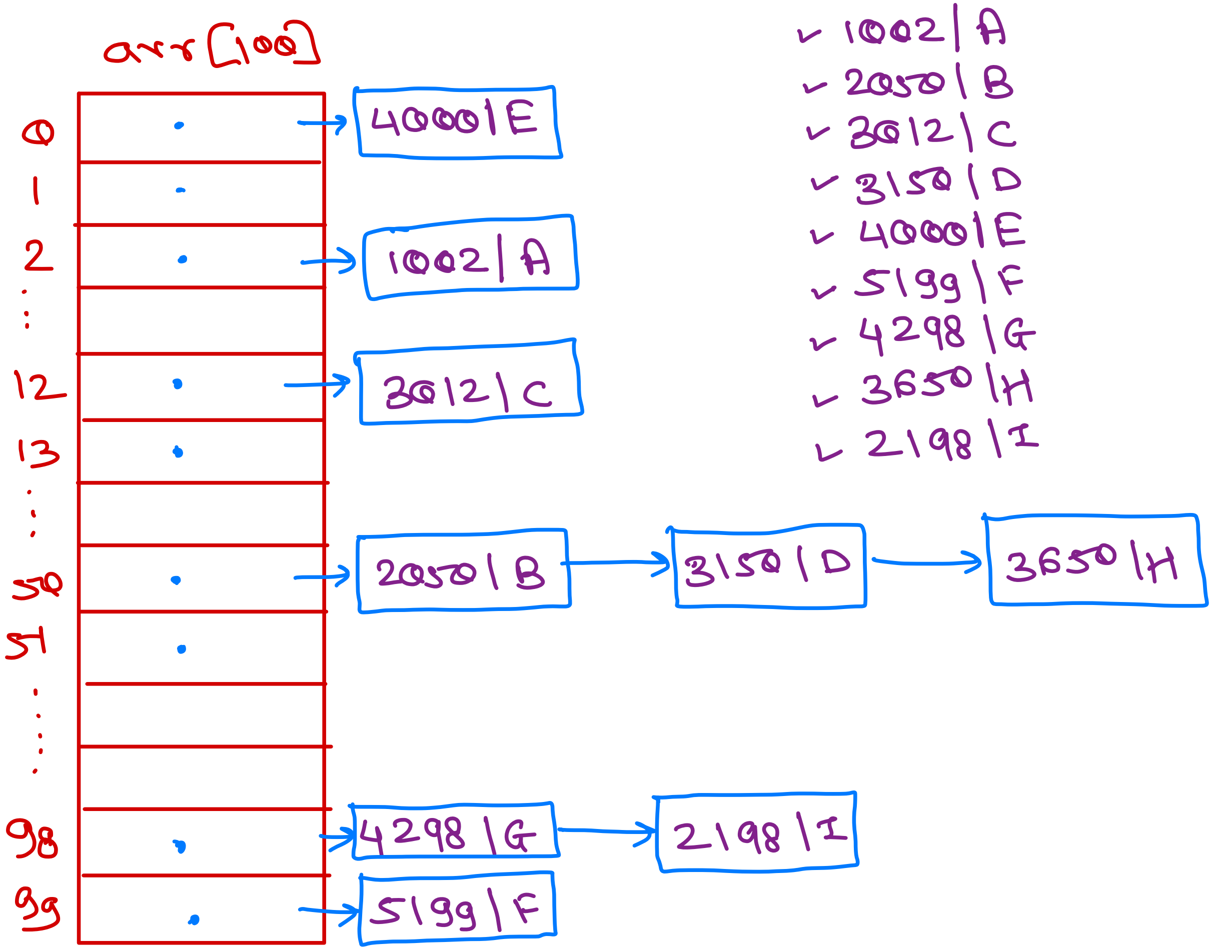
open addressing
or chaining

chaining

Hash Table

- Chaining

$\boxed{h(r) = r \% 100}$



- ✓ 1002 | A
- ✓ 2050 | B
- ✓ 3012 | C
- ✓ 3150 | D
- ✓ 4000 | E
- ✓ 5199 | F
- ✓ 4298 | G
- ✓ 3650 | H
- ✓ 2198 | I

✓ each slot in the table hold a collection of key-value pairs/entries called as buckets.

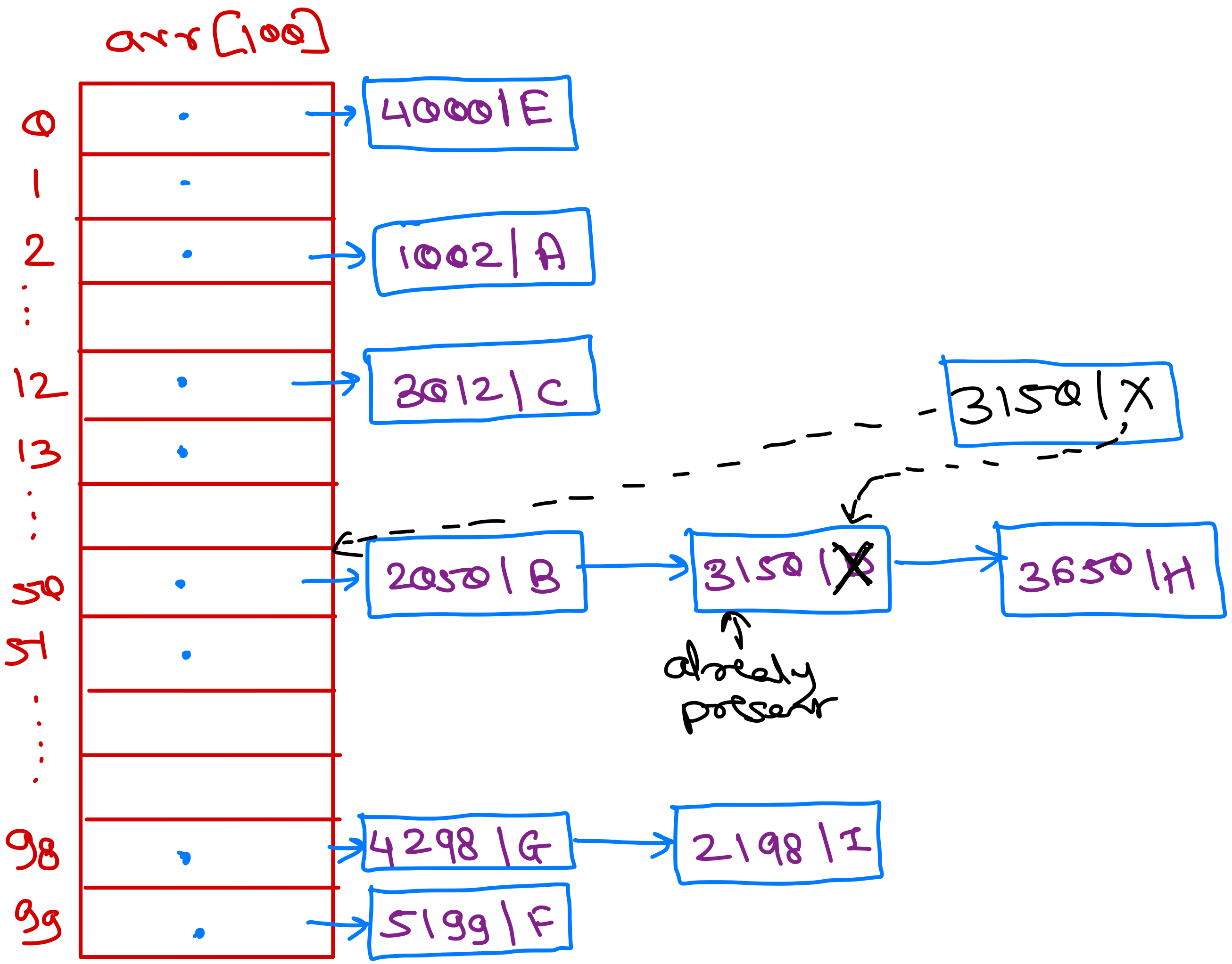
✓ Note that, actual kv pairs/entries are not stored in table. The storage is external to the table.

✓ if multiple entries collide (same slot in table), then they will be added in same bucket..

✓ num of entries can be greater, equal or lesser than table size (num of buckets).

✓ generic impl of hashtable (irrespective of load factor). So C++ STL map<>, Java HashMap/HashTable, Python/C# Dictionary internally follow chaining.

Hash Table



```
List arr[100]; // arr[i] is a list.
               // each entry in list
               //   is KeyValue pair.

put(k, v):
    index = k % 100; //hash fn
    for (kv : arr[index]) {
        if (kv.key == k) {
            kv.value = v;
            return;
        }
    }
    arr[index].add(new kv);

get(k):
    index = k % 100; //hash fn
    for(kv : arr[index]) {
        if (kv.key == k)
            return kv.value;
    }
    return null;
```

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>