Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Linked List - competitive/interview question

- Solving a polynomial (using linked list)

$f(x) = 2x^3 - 4x^2 + 9x - 12$

$f(x) = 2x^3 - 4x^2 + 9x^1 - 12x^0$

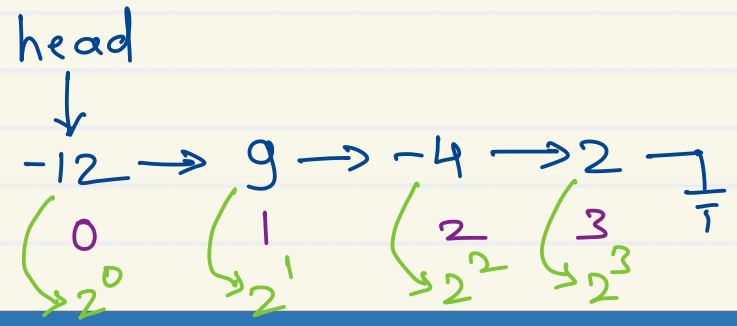
if $x = 2$

$f(2) = 2 \times 2^3 - 4 \times 2^2 + 9 \times 2^1 - 12 \times 2^0$

$= 16 - 16 + 18 - 12$

$= 6$

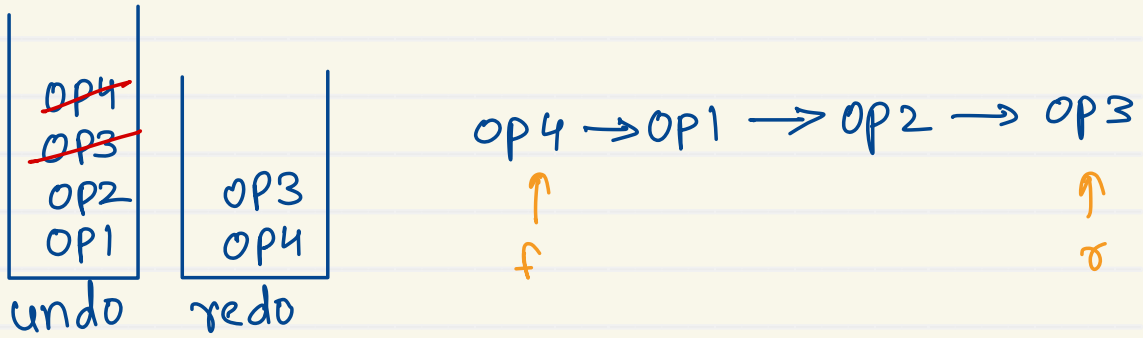
- represent polynomial in form of linked list of coefficient




```
int solvePolynomial(Node head, int x){
    res = 0;
    px = 1
    Node trav = head;
    while (trav != null) {
        coe = trav.data;
        trav = trav.next;
        res = res + coe * px;
        px = px * x;
    }
    return res;
}
```

1. checking palindrome

2. undo / redo operations

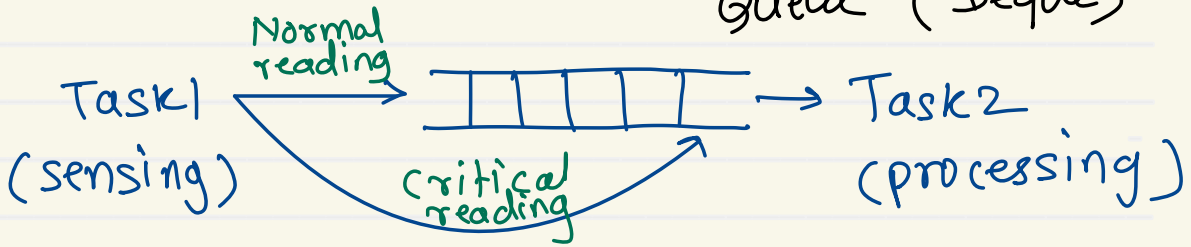


3. browser history

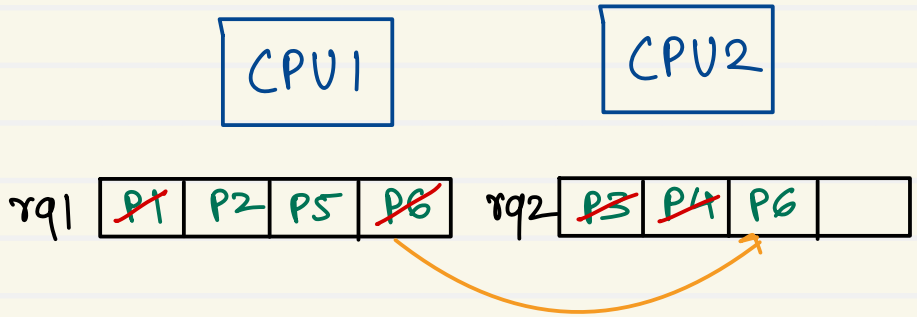
4. FreeRTOS → task communication

↓

Queue (Deque)



5. scheduling in multiprocessor systems



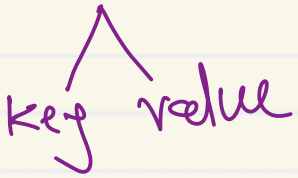
A steal Algorithm → process stealing

Hashing

Array : linear search → O(n)
binary search → O(log n)

Linked list : search → O(n)

- Associative data structure



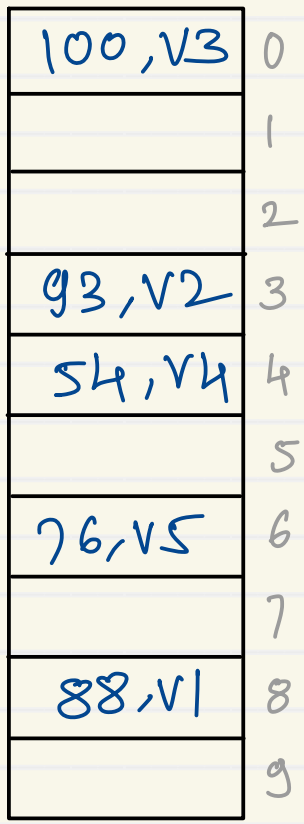
- hashing is a technique in which data can be inserted, deleted and searched in constant average time O(1)
- Implementation of hashing is known as hash table / Hashmap / Hashset / Dict
- Hash table is array of fixed size in which elements are stored in key - value pairs
  - Array - Hash table
  - Index - Slot
- In hash table only unique keys are stored
- Every key is mapped with one slot of the table and this is done with the help of mathematical function known as hash function

key → Hash function → slot

Hashing

key → 88-V1
93-V2
100-V3
54-V4
76-V5

size = 10



Hash Table

h(k) = k % size

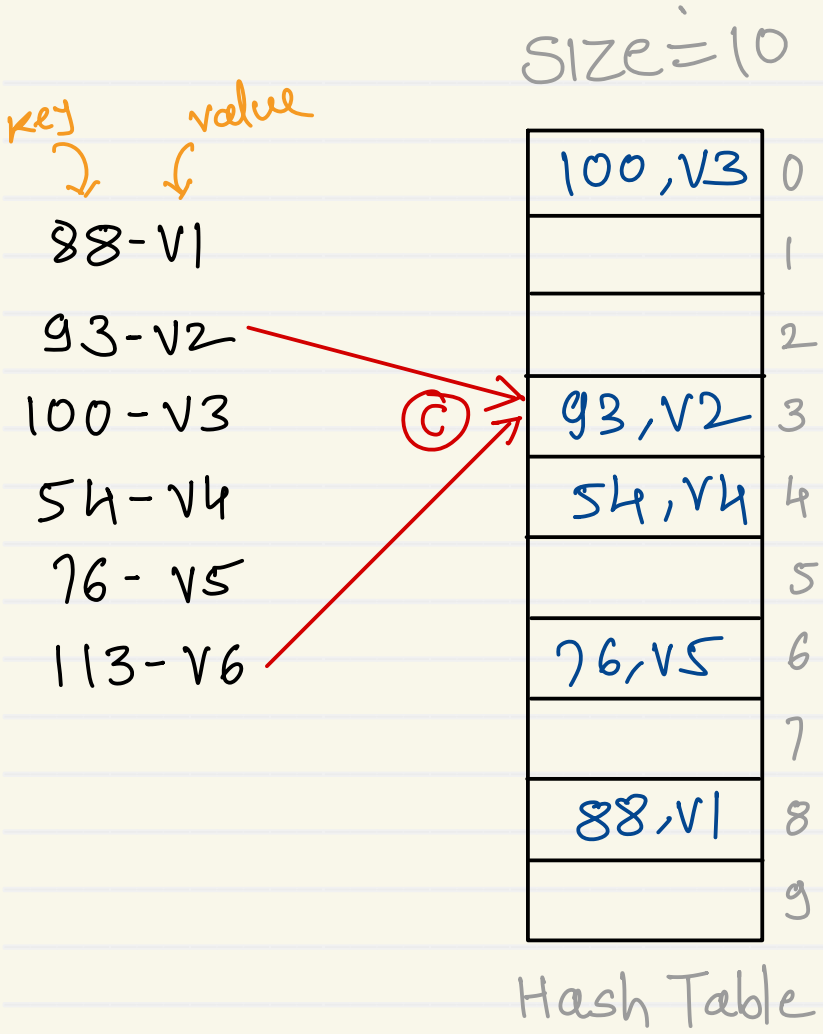
↑ key

h(88) = 88 % 10 = 8
h(93) = 93 % 10 = 3
h(100) = 100 % 10 = 0
h(54) = 54 % 10 = 4
h(76) = 76 % 10 = 6

Add: O(1)
1. find slot
2. arr[slot] = data

search: O(1)
1. find slot
2. return arr[slot].value

Delete: O(1)
1. find slot
2. arr[slot] = null



h(k) = k % size

↑ key

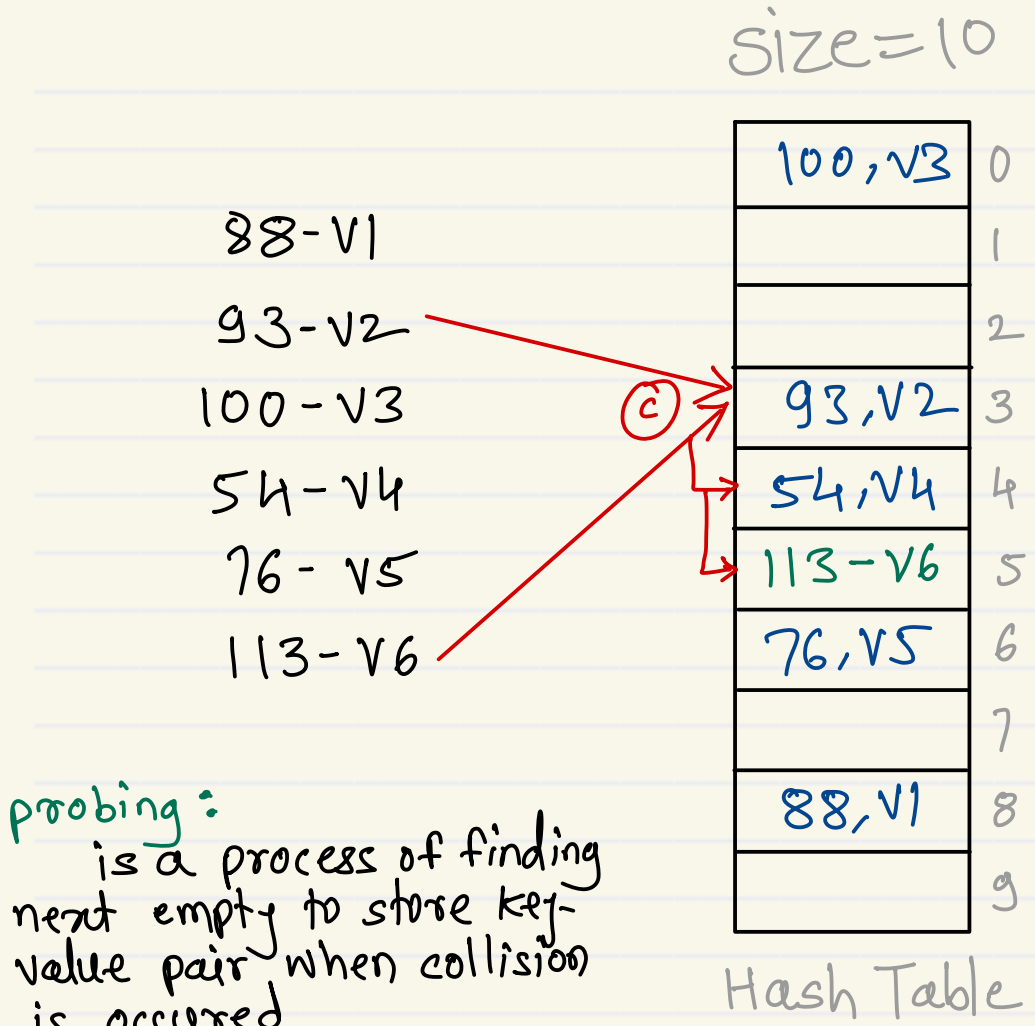
h(88) = 88 % 10 = 8
h(93) = 93 % 10 = 3
h(100) = 100 % 10 = 0
h(54) = 54 % 10 = 4
h(76) = 76 % 10 = 6
h(113) = 113 % 10 = 3

Collision :
- when two distinct keys yield/give same slot of the table

Collision handling / resolution techinques

1. Open addressing
   i. linear probing
   ii. quadratic probing
   iii. double hashing
2. closed addressing (chaining)

Open addressing - Linear probing



probing :
is a process of finding next empty to store key-value pair when collision is occured

h(k) = k % size

h(k, i) = [h(k) + f(i)] % size

f(i) = i ← probe number

where i = 1, 2, 3, …

h(113) = 113 % 10 = 3 (c)

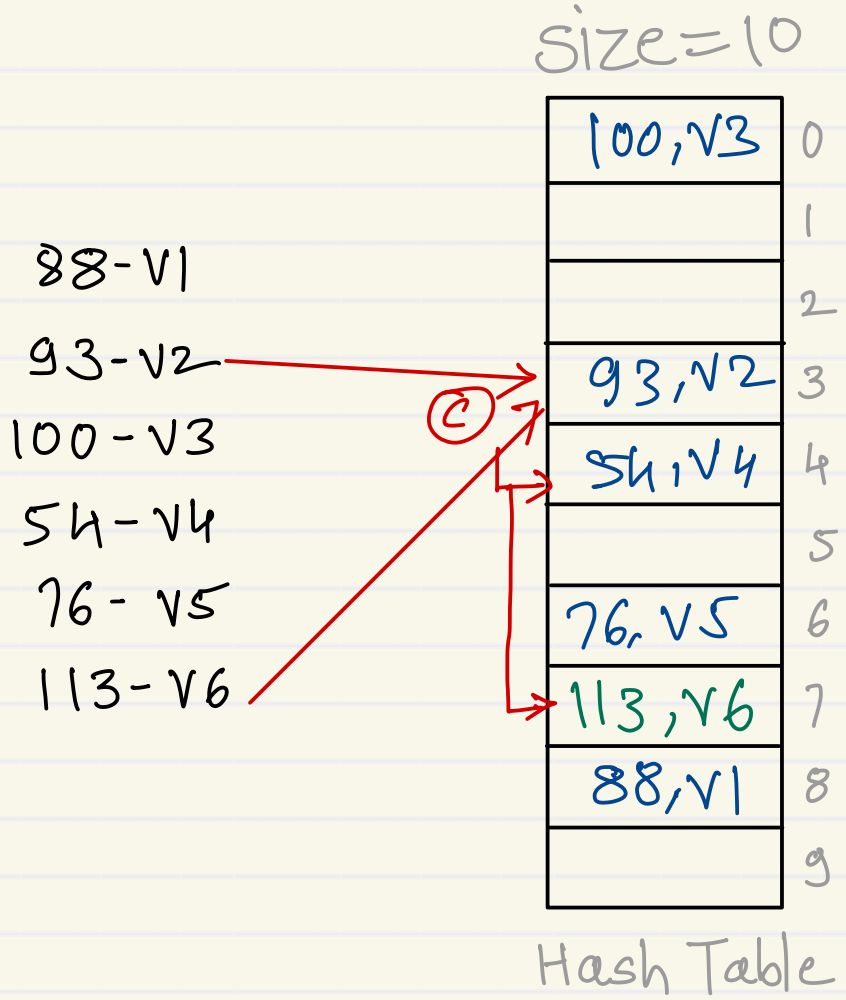
1st probe : h(113, 1) = [3 + 1] % 10 = 4 (c)

2nd probe : h(113, 2) = [3 + 2] % 10 = 5

Primary clustering :

- near key position table becomes crowded / bulky.
- need to take long run of filled slots to find next empty slot "near" key position.

Open addressing - Quadratic probing



h(k) = k % size

h(k, i) = [h(k) + f(i)] % size

f(i) = i^2

where i = 1, 2, 3, …

h(113) = 113 % 10 = 3 (C)

1st probe : h(113, 1) = [3 + 1] % 10 = 4 (C)

2nd probe : h(113, 2) = [3 + 4] % 10 = 7

- problem of primary clustering is solved.
- no guarantee of getting empty slot for key-value pair

Secondary clustering :
- need to take long run of filled slots to find next empty slot "away" key position.

Open addressing - Quadratic probing

88-V1
93-V2
100-V3
54-V4
76-V5
113-V6
23,V7
33,V8

size=10

100,V3	0
	1
23,V7	2
93,V2	3
54,V4	4
	5
76,V5	6
113,V6	7
88,V1	8
33,V8	9

Hash Table

h(k) = k % size

h(k, i) = [h(k) + f(i)] % size

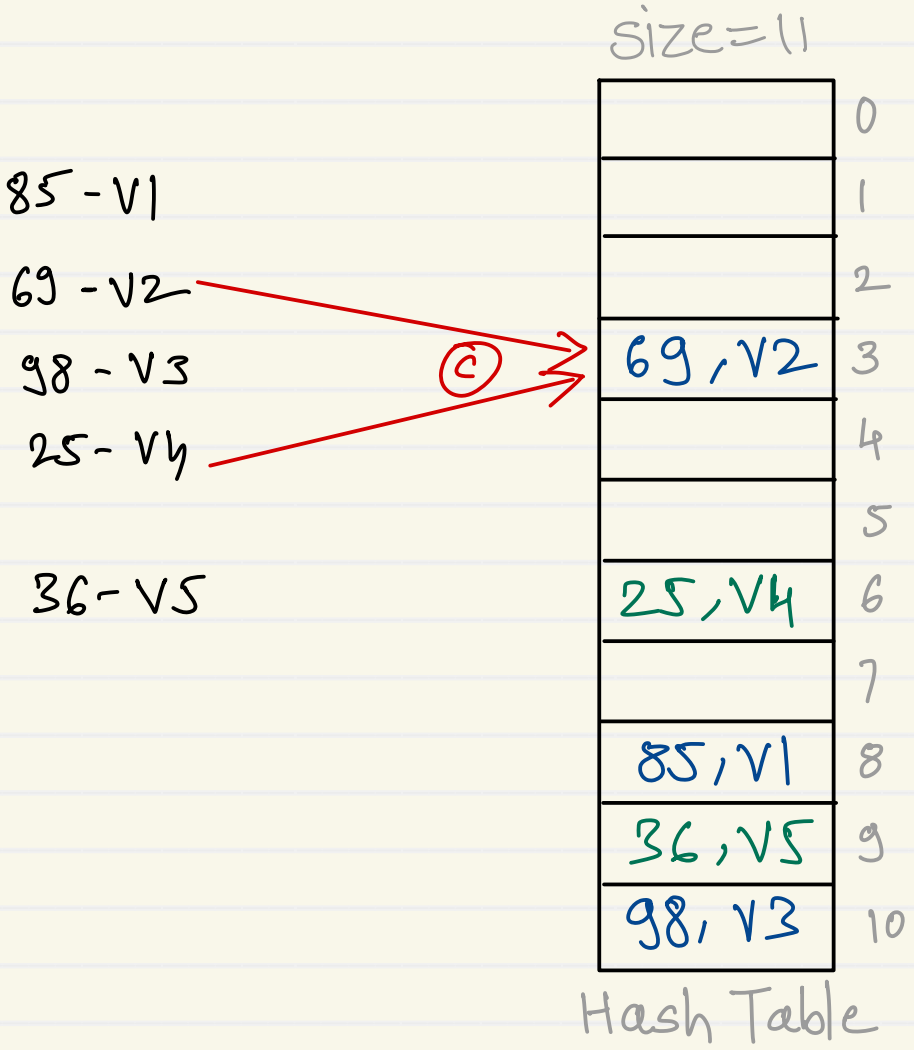
f(i) = i^2

where i = 1, 2, 3, …

h(23) = 23 %. 10 = 3 (c)
1st: h(23,1) = [3+1] %. 10 = 4 (c)
2nd: h(23,2) = [3+4] %. 10 = 7 (c)
3rd: h(23,3) = [3+9] %. 10 = 2

h(33) = 23 %. 10 = 3 (c)
1st: h(33,1) = [3+1] %. 10 = 4 (c)
2nd: h(33,2) = [3+4] %. 10 = 7 (c)
3rd: h(33,3) = [3+9] %. 10 = 2 (c)
4th: h(33,4) = [3+16] %. 10 = 9

Open addressing - Double hashing



h1(k) = k % size

h2(k) = 7 - (k % 7)

h(k, i) = [h1(k) + i * h2(k)] % size

h1(85) = 85 % 11 = 8

h1(69) = 69 % 11 = 3

h1(98) = 98 % 11 = 10

h1(25) = 25 % 11 = 3 (c)

h2(25) = 7 - (25 % 7) = 3

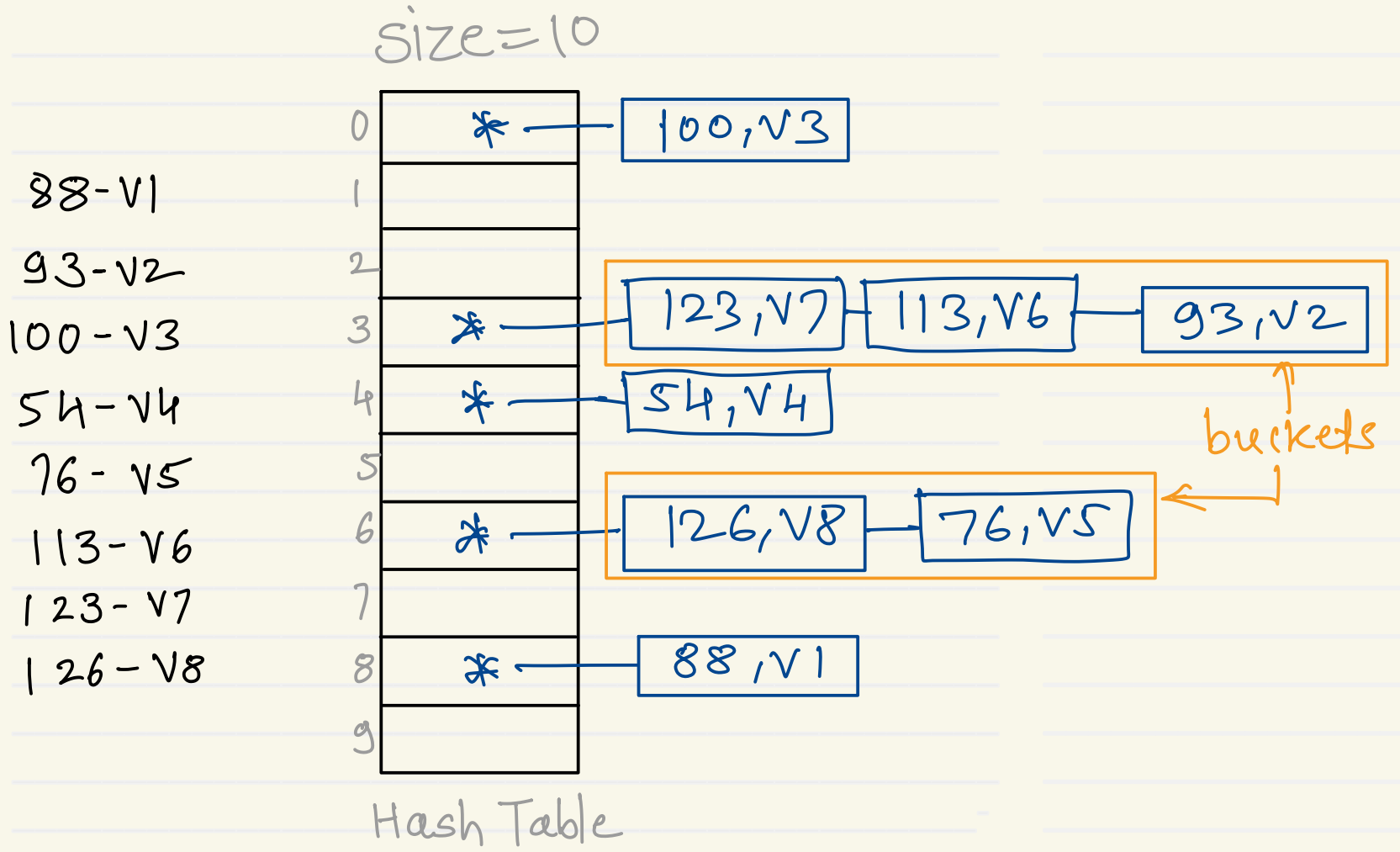
1st : h(25, 1) = [3 + 1 * 3] % 11 = 6

h1(36) = 36 % 11 = 3 (c)

h2(36) = 7 - (36 % 7) = 6

1st : h(36, 1) = [3 + 1 * 6] % 11 = 9

Closed Addressing / Chaining / Separate Chaining



h(k) = k % size

Advantage :
can store multiple key value pairs

Disadvantages :
i. key value pair are stored outside the table
ii. space requirement is more
iii. worst case time complexity is O(n).
(if maximum key's yeild same slot)

Rehashing

$$\text{Load factor} = \frac{n}{N}$$

e.g. N = 10
n = 7
load factor = $\frac{n}{N} = \frac{7}{10} = 0.7$
Hash table is 70% filled

n - number of elements (key-value) present in hash table
N - number of total slots in hash table

- Load factor ranges from 0 to 1.
- If n < N Load factor < 1 - free slots are available
- If n = N Load factor = 1 - free slots are not available
  if n > N Load factor > 1

{ open addr can be used
{ open addr can not be used
{ chaining

- In rehashing, whenever hash table will be filled more than 60 or 70 % size of hash table is increased by twice
- Existing key value pairs are remapped according to new size

open addr => 0.6 to 0.7
closed addr => 0.75 to 0.9