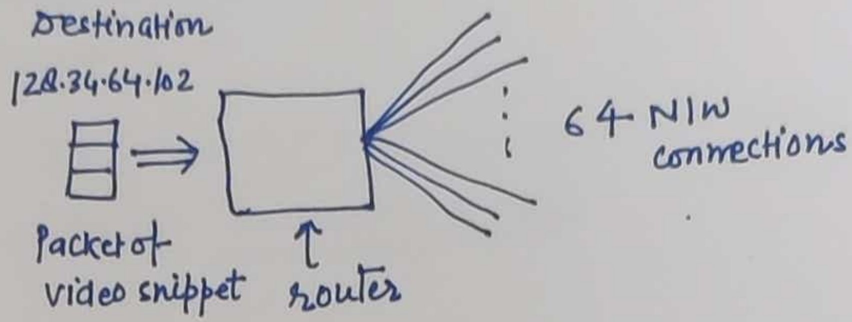
# HASHING.

## Motivation:-

Scenario:.




* Goal (in this scenario): to design a next generation router
* Router process information packets, allowing them to move through networks having a lot of interconnections.

* When a packet recieved by router from any of 64 cables, router must examine (by seeing the information at begining of packet) and decides that where to send this at remaing 63 cables.
- Delay is not allowed - at 2 Ms delay allowed.

Abstract level:-

Packet is modeled as a pair $(k, x)$ ← data in the packet

↑ Key indicating destination

To do this:.

S/W should maintain a pairs as $(k, c)$ ← cable

↑ add$^s$

Operation supported:-

put (k,c) : adds Key cable pair to collection
get (k) : return cable # for given destination key k.

Issue: one possibility — Linked list
put(k,c) — O(1) if put at head
but get (k) — O(n) times
Which is not allowed for large "n"

" BETTER OPTION IS HASH TABLE "

## IDEA for HASHTABLE DATA STRUCTURE :-

- It must allow users to assign keys to elements and then use those keys later for "look up" or "remove" the element.

  ⇑ This functionality defines a new data structure called "dictionary or map".

## Def$^n$: (MAP) -

- Map stores a set of pairs $(k, v)$ called item.
  k: key. v: value associated with key.

- Map (M) data structure supports following methods -

  get(k) : if M has an in item with key equal to k, then return that item, else return "NULL"

  put(k,v) : insert v ~~at~~ with key k. if an item (k,v') is already there then replace v' with v.

  remove(k) : ~~rete~~ If M has such item, then remove that from M, else return "NULL".

## Implementation of Map:-

- Using Lookup Tables:- - set of integers : [0, .. n-1]
  - Create an array A[n].

  put(k,v) → assign (k,v) to A[k]

  get(k) → return A[k]

  remove(k) → retun A[k] and assign "NULL" to A[k].

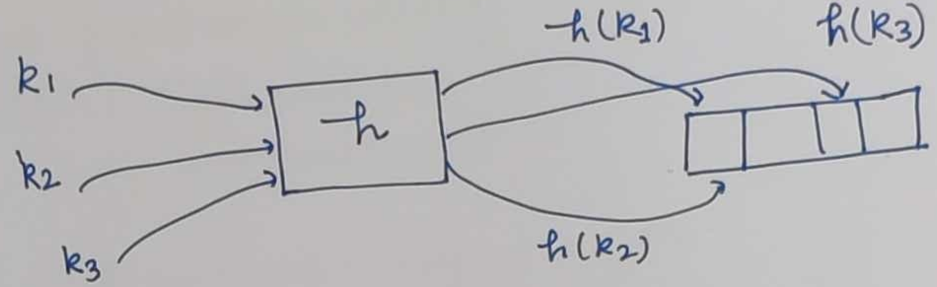
Drawback:- - space $O(n)$
- Key requires to be unique

# Hash Functions

## Hash function with lookup Table:-

**Idea:** In place of 'k', $h(k)$ will be used as an index to array A.

- store $(k, v)$ at $A[h(k)]$




**Issue:-** - A function can map two keys at same ~~look~~ location of array.

$$h(k_1) = h(k_2), \text{ for } k_1 \neq k_2$$

$\Rightarrow$ if $h(k_1) = h(k_2) =$ "p", then we say that there is collision at hash value "p".

## Properties of good hash function:-

- quick to compute
- No collision ~~or~~ avoid collision → distribute keys uniformly throughout the table.
- good hash function are rate - birthday paradox.

↓

* means: it may happen that there are various slots but hash fun. is mapping to few slots only.

## How to deal with non-iteger keys??

- first, we need an efficient way to convert it into integers.
- then apply hash function.

## Approach for dealing with non-integer:-

A hash function is usually the composition of two maps : hash code map
compression map

$$\left[\begin{array}{l} \text{hash code map: Key} \rightarrow \text{integer} \\ \text{Compression map: iteger} \rightarrow [0, \cdots, p-1] \end{array}\right]$$

### 1. Summing components:-

Key 'k' is a d-tuple
$(x_1, \cdots, x_d)$

$$\boxed{h(k) = \sum_{i=1}^{d} x_i}$$ : Hash code mapping

if $h(k) \gg p$, take $h(k)$ with modulo $p$ ] compression map.

- Issue:- ~~STOP~~ SPOT & TOPS
$\Downarrow$ $\Downarrow$
same some in terms of ASCII code
$\Downarrow$ collision.

## (ii) Polynominal Evalution method:-

For string of natural language, combine the character values $k = (x_1, x_2, \cdots, x_d)$

use $a \neq 1$ and hash function is:

$$h(k) = x_1 a^{d-1} + x_2 a^{d-2} + \cdots + x_{d-1} a + x_d$$

By Horner's rule - it can be written as -

$$h(k) = x_d + a(x_{d-1} + a(x_{d-2} + \cdots + a(x_3 + a(x_2 + a x_1)) \cdots$$

$x_1, \cdots, x_d$: coefficient of $(d-1)$ - degree polynomial.

**Fact**:- Experimental study suggest that,

$a = 33, 37, 39, 41$ : good choices for 'a' for english words

$\Rightarrow$ for a dictionary of 50,000 words - in each case # collision can be less than 7.

## Compression Maps Approaches :-

### (i) Modular Division:-

$$\boxed{h(k) = k \bmod n}$$

**Issue**:- (i) Key = $\{200, 205, 210, 215, \cdots, 600\}$

$n = 100$ } then each hash code collide with three others.

if $n = 101$: no collision.

$\boxed{\text{Try to choose "n" as prime number}}$

(ii) Random linear function:

$$\boxed{h(k) = (ak+b) \bmod n}$$

$$\boxed{\begin{matrix} 0 < a < n \\ 0 \le b < n \end{matrix}}$$

* eliminates collision - but 'a' should not be multiple of 'n'.

# Collision Resolution

→ separate chaing.
→ Open addressing

# of Keys

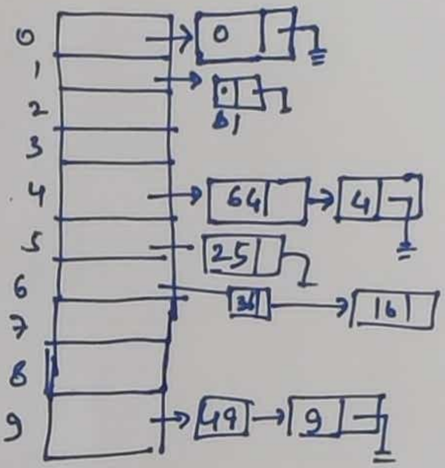
$\alpha : \frac{n}{m}$ ← # available slots in Table

Load factor

**Chaing:** linked list of colliding elements in each slot of hash table.

Ex:- $K = \{0, 1, 4, 9, 16, 25, 36, 49, 64, 81\}$

$h(k) = k \bmod 10$




worst-case: search & Delete $O(n)$

get() remove.

**Open Addressing:**

- No list: All elements occupy hash table it self
  : Idea is to ~~successful~~ successively examine or probe the hash table till an empty slot is found

## Linear probing :-

$$h : U \times \{0,1,\ldots,m\} \rightarrow \{0,1,\ldots,m-1\}$$

→ Probe sequence.

$(h(k,0), h(k,1); \ldots h(k, m-1)) \rightarrow$ a permutation of $\{0,1,\ldots,m-1\}$

- Put (k, v)

$i \leftarrow h(k)$

while( $A[i] \neq$ NULL)

$i \leftarrow (i+1) \bmod n$

$A[i] \leftarrow v$

Ex:- $K = \{89, 18, 49, 58, 9\}$

$n = 10$

89 : $h(89,0) = 9$

$A[9] = 89$

18 : $h(18,0) = 8$

49 : → $A[0]$

58 : → $A[1]$

2 : 9 → $A[2]$

* Problem of Primary clustering
increase Avg search time

## Quadratic Probing →

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

Ex:- $K = \{89, 18, 49, 58, 9\}$

$$h(k,i) = (h'(k) + i^2) \bmod 10$$

$c_1 = 0$
$c_2 = 1$
$m = 10$.

## Double Hashing

- Two hash functions- $h_1$ & $h_2$
- $h_1(k)$: position where we should check first
- $h_2(k)$: will gives location we should look again for key.
- In linear $h_2(k)$ is always.

$$h(k,i) = (h_1(k) + i\,h_2(k)) \bmod m$$
$$\Downarrow$$
$$(h_1(k) + i\,h_2(k))$$

Code:-

```
i ← h1(k)
P ← h2(k)
While A[i] ≠ NULL
    i ← (i+P) mod m
A[i] ← k.
```

Ex:- $h_1 = k \bmod 13$, $h_2(k) = 8 - k \bmod m$

$\langle 18, 41, 22, 44, 59, 32, 31, 73 \rangle$

↓

for 44. → initially 5 will be occupied will go 4 location ahead, again occupied, then go to next

* Double hashing

↳ One of the best methods for open addressing

$$h(k,i) = (h_1(k) + i\,h_2(k)) \bmod m$$

Example:

- $k = \{34, 55, 12, 8, 45, 37, 32, 88, 98, 54, 21, 42, 56, 74, 52, 33, 16\}$ $\quad h_1(k) = k \% 20$ ; $h_2(k) = k \% 6 + 1$

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| values | 74 | 98 | 42 | / | / | 45 | 16 | 52 | 8 | 21 | / | / | 12 | 88 | 34 | 55 | 54 | 37 | 32 | 56 |
| #Probes | 3 | 2 | 1 | / | / | 1 | 3 | 4 | 1 | 3 | / | / | 1 | 2 | 1 | 1 | 3 | 1 | 3 | 2 |

34: 34 % 20 = 14 ✓

55: 55 % 20 = 15 ✓

12: 12 % 20 = 12 ✓

8: 8 % 20 = 8 ✓

45: 45 % 20 = 5 ✓

37: 37 % 20 = 17 ✓

42: 42 % 20 = 2 ✓

56: 56 % 20 = 16 × $\quad$ 56 % 6 + 1 = 3

= 16 + 3 = 19 ✓

74: 74 % 20 = 14 × $\quad$ 74 % 6 + 1 = 3

= 14 + 3 = 17 ×, 14 + 2×3 = 20 % 20 = 0 ✓

32: 32 % 20 = 12 ×, $\underbrace{32 \% 20}_{12} + 1 \times (\underbrace{32 \% 6 + 1}_{2+1=3}) = 12 + 3 = 15$ ×

(12 + 2×3) % 20 = 18 % 20 = 18 ✓

88: 88 % 20 = 8 × $\quad$ 88 % 6 + 1 = 5

8 + 5 = 13 ✓

98: 98 % 20 = 18 × $\quad$ 98 % 6 + 1 = 3

18 + 3 = 21 % 20 = 1 ✓

54: 54 % 20 = 14 × $\quad$ 54 % 6 + 1 = 1

14 + 1 = 15 ×, 14 + 2×1 = 16 ✓

21: 21 % 20 = 1 × $\quad$ 21 % 6 + 1 = 4

= 1 + 4 = 5 ×, 1 + 2×4 = 9 ✓

52: 52 % 20 = 12 ×

=> 52 % 6 + 1 = 5

= 12 + 5 = 17 ×, 12 + 2×5 = 22 % 20 = 2 ×

= 12 + 3×5 = 27 % 20 = 7 ✓

~~33: 33 % 20 = 13 ×~~

=> 33 % 6 + 1 = 4

= 13 + 4 = 17 ×

= 13 + 2×4 = 21 % 20 = 1 ×

~~= 13 + 3×4 = 25 % 20 = 5 ×~~

= 13 + 4×4 = 29 % 20 = 9 ×

= 13 + 5×4 = 33 % 20 = 13 ×

= 13 + 6×4 = 37 % 20 = 17 ×

= 13 + 7×4 = 41 % 20 = 1 ×

33 → cannot be inserted

16: 16 % 20 = 16 ×, 16 % 6 + 1 = 5

= 16 + 5 = 21 % 20 = 1 ×

16 + 2×5 = 26 % 20 = 6 ✓

# Linear Probing - (Implementation Point of View)

## Put(k,v) :-

$i \leftarrow h(k)$

While $A[i] \neq NULL$

  if $A[i].Key = k$

    $A[i] \leftarrow (k, v)$

  $i \leftarrow (i+1) \bmod m$

$A[i] \leftarrow (k, v)$

## Get(k):

$i \leftarrow h(k)$

While $A[i] \neq NULL$

  if $A[i].Key = k$

    return $A[i]$

  $i \leftarrow (i+1) \bmod m$

return NULL.

## Remove(k)

$i \leftarrow h(k)$

While $A[i] \neq NULL$

  if $A[i].Key = k$

    $temp \leftarrow A[i]$

    $A[i] \leftarrow NULL$

    Shift(i)

    return temp

  $i \leftarrow (i+1) \bmod m$

return NULL

## Shift(i)

$s \leftarrow 1$

While $A[(i+s) \bmod m] \neq NULL$

  $J \leftarrow h(A[(i+s) \bmod m].Key)$

  if $J \notin (i, i+s] \bmod n$

Fill the Hole → $A[i] \leftarrow A[(i+s) \bmod m]$

Move the hole → $A[(i+s) \bmod N] \leftarrow NULL$

    $i \leftarrow (i+s) \bmod N$

    $s \leftarrow 1$

  else

    $s \leftarrow s+1$