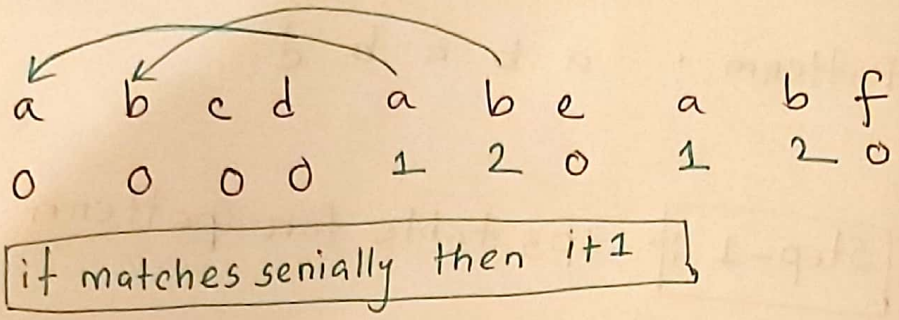
KMP

∴ Basic Lps :- table



Lps table

a b c d e (a) (b) f (a) (b) (c)

0 0 0 0 0 1 2 0 1 2 3

Lps → Longest prefix Suffix

Kmp → Knuth - Monis - Patt Algonithm

complexity → O(m+n)

m → pattern length

n → string Length

Problem-1

sting : a b a b c a b c a b a b a b d

Patterm : a b a b d

Step-1 : Lps table for pattern

i=0 (inital

	a	b	a	b	c	a	b	c	a b	a	b a	b	d
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

J=0 (inital)

	a	b	a	b	d
Indx:	0	1	2	3	4
Lps :	0	0	1	2	0

— . —

* compare string [i] = string [j]

① if match i++, j++

② if mismatch i = same

j = LPS [pattern [j-1]]

continue compare

③ if j=0 , mismatch . J = 0

i = i++

$i=0, j=0 \rightarrow$ match (1)

$i=1, j=1 \rightarrow$ match (1)

$i=2, j=2 \rightarrow$ match (1)

$i=3, j=3 \rightarrow$ match (1)

$i=4, j=4 \rightarrow$ mismatch (2)

$i=4, j=2 \rightarrow$ mismatch (2) lps[4-1] = j

$i=4, J=0 \rightarrow$ mismatch lps[2-1] = j

(3) $i=5, J=0 \rightarrow$ match

$i=6, J=1 \rightarrow$ match

$i=7, j=2 \rightarrow$ mismatch lps[2-1] = J

$i=7, J=0 \rightarrow$ mismatch

(3) $i=8, j=0 \rightarrow$ match

$i=9, j=1 \rightarrow$ match

$i=10, j=2 \rightarrow$ match

$i=11, j=3 \rightarrow$ match

$i=12, J=4 \rightarrow$ mismatch lps[4-1] = j = 2

i = 12, j = 2 → match

i = 13, j = 3 → match

i = 14, j = 4 → Full matched.

—•—

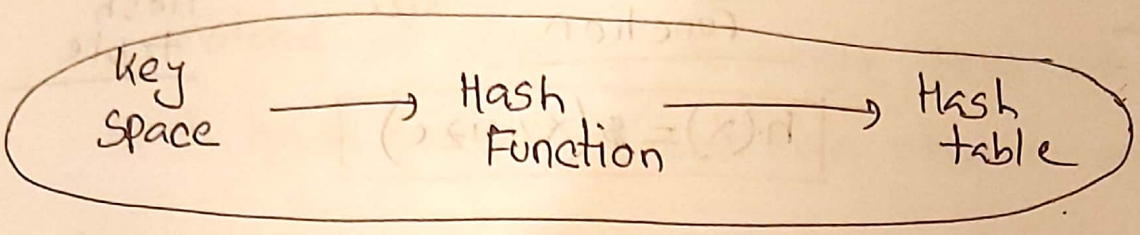
```
void kmp ( string text, string pattern)
{
    int n = text.size();          i = 0
    int m = ~~text~~ pattern.size();   j = 0

    int lps[m];
    LPPS (pattern, m, lps);

while ((N-i) ≥ (m-j)) {
match (1)  if (text[i] = pattern[j]) ) { i++, j++ }
    if (j = m) { "found" }
mismatch (2)  else if { i < N && pattern[j] != text[i] )
    { if (j!=0) j = lps[j-1]
      else i = i+1
    }
```

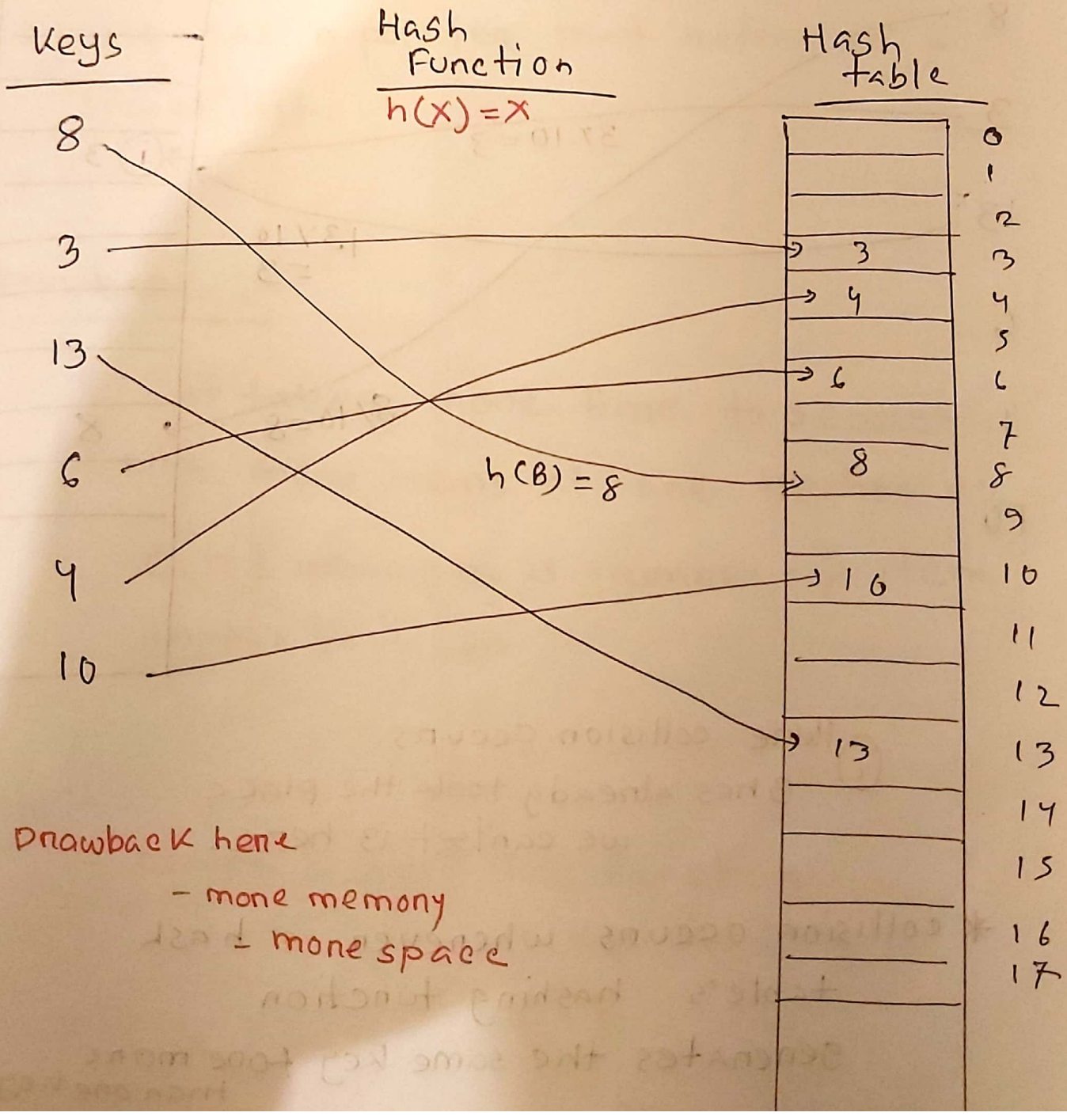
Lps

```
LPS( int m, string pattern, int lps[])
{
    int Len = 0;
    lps[0] = 0;
    int i = 1;
while(i<M){
        if (pat[i] = pat(len)
        { len++,
          lps[i]=len
            i++
        }

    else    if (len==0)
        { lps [i]=0
            i++
        else
        <
        len = lps[len-1]
        }

    }
```

Hashing

Basic Hashing





Drawback here

- more memory
- more space

Optimize of basic hashing

(change hash function)

Here size = 10

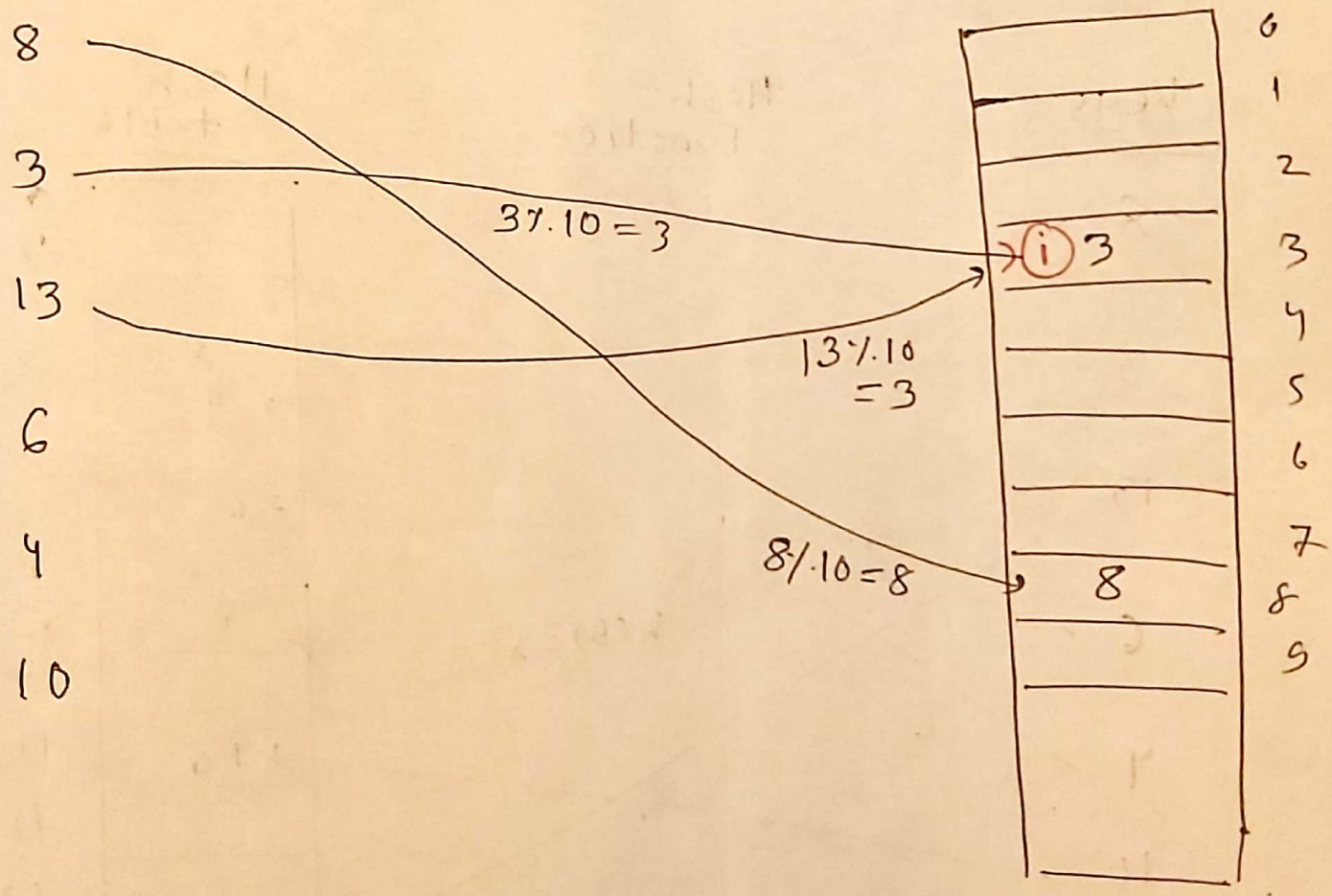
Keys

Hash function

size = table size

Hash table

$$h(x) = x \% size$$



(i) Here collision occurs
3 has already took the place
we can't set 13 here.

* collision occurs whenever a hash table's hashing function genenates the same key for one more than one key.

ways to solve collision

(1) Open hashing / chaining :-

Instead of storing a single item at each hash bucket, we can chain multiple elements together so that each key of the table has a pointer that reference a linked list.

Drawback :-

it takes more time to search with more items at one location or $O(n)$ where n is number of item in the bucket.

Solu :-

write a good hash function so that n in not too long

$\therefore O(1)$

chaining

Key

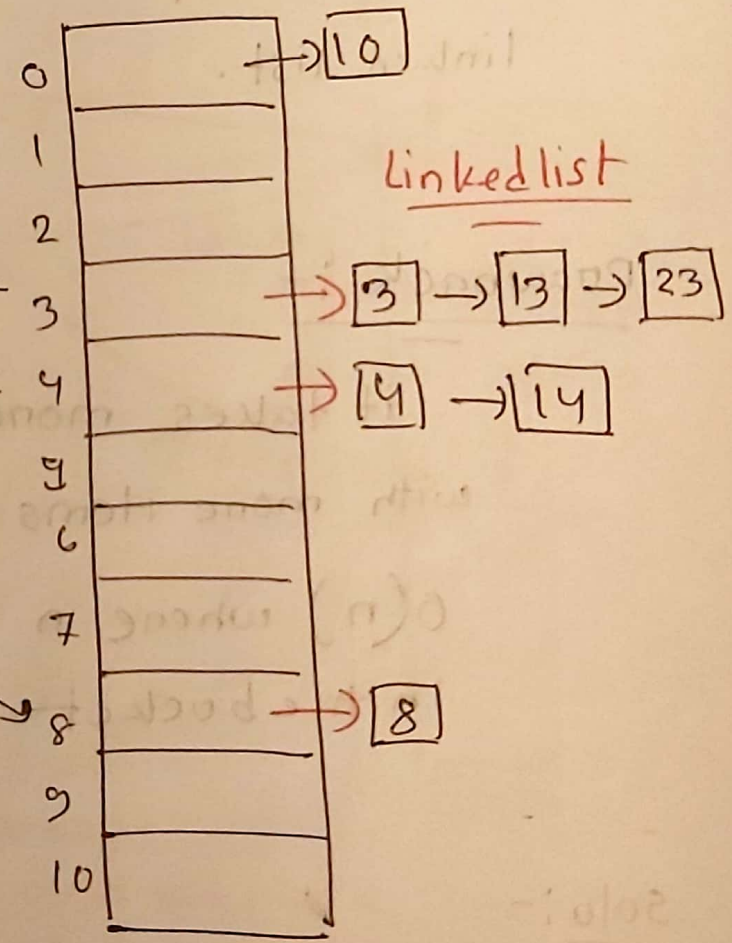
8
3
13
23
6
4
14
10

hash function

$h(x) = x \% size$

size = 10

Hash table



Linkedlist

② (i) closEd hashing :-

openaddnessing

(Linear Probing)

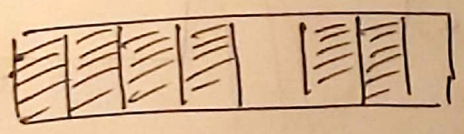
if collision occurs at spot on hash table, a hash function can simply go to the next empty bucket over and add element here

— Linear Probing

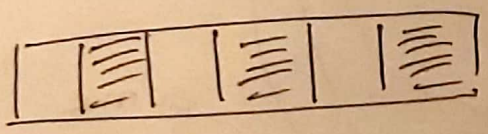
(probe through the table untill it finds an empty space, cycle back)

drawbacks :-

The tendacy of clustering



→ a clustered table



→ a well spaced table

Linear Probing

size = 10

keys

Hash function

$h(x) = x \% size$

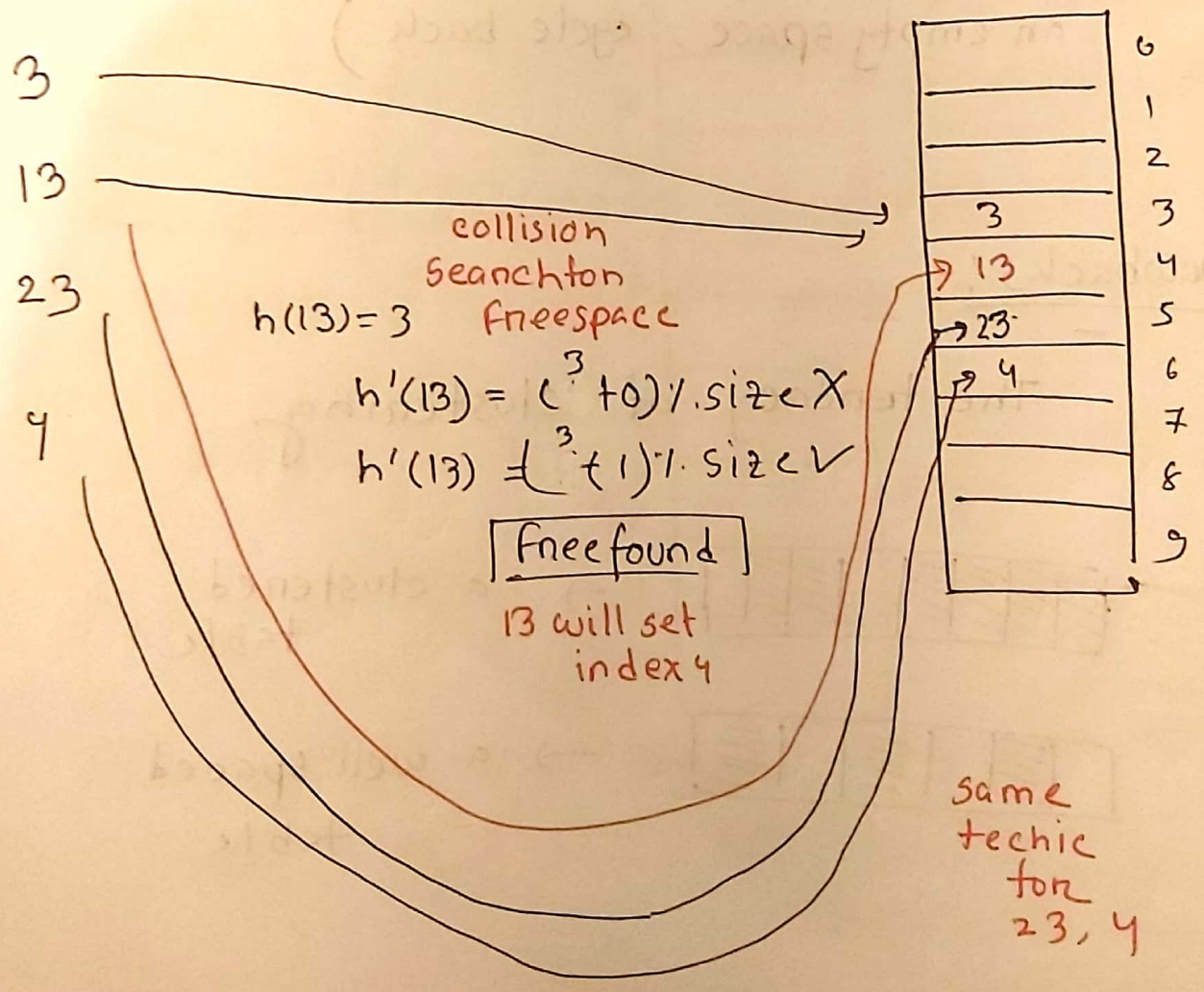
Hash table

$\therefore h'(x) = \{h(x) + f(i)\} \% size$

$\therefore f(i) = i$

$i = 0, 1, 2, 3 \ldots$

Search for Free space



Quadratic Probing

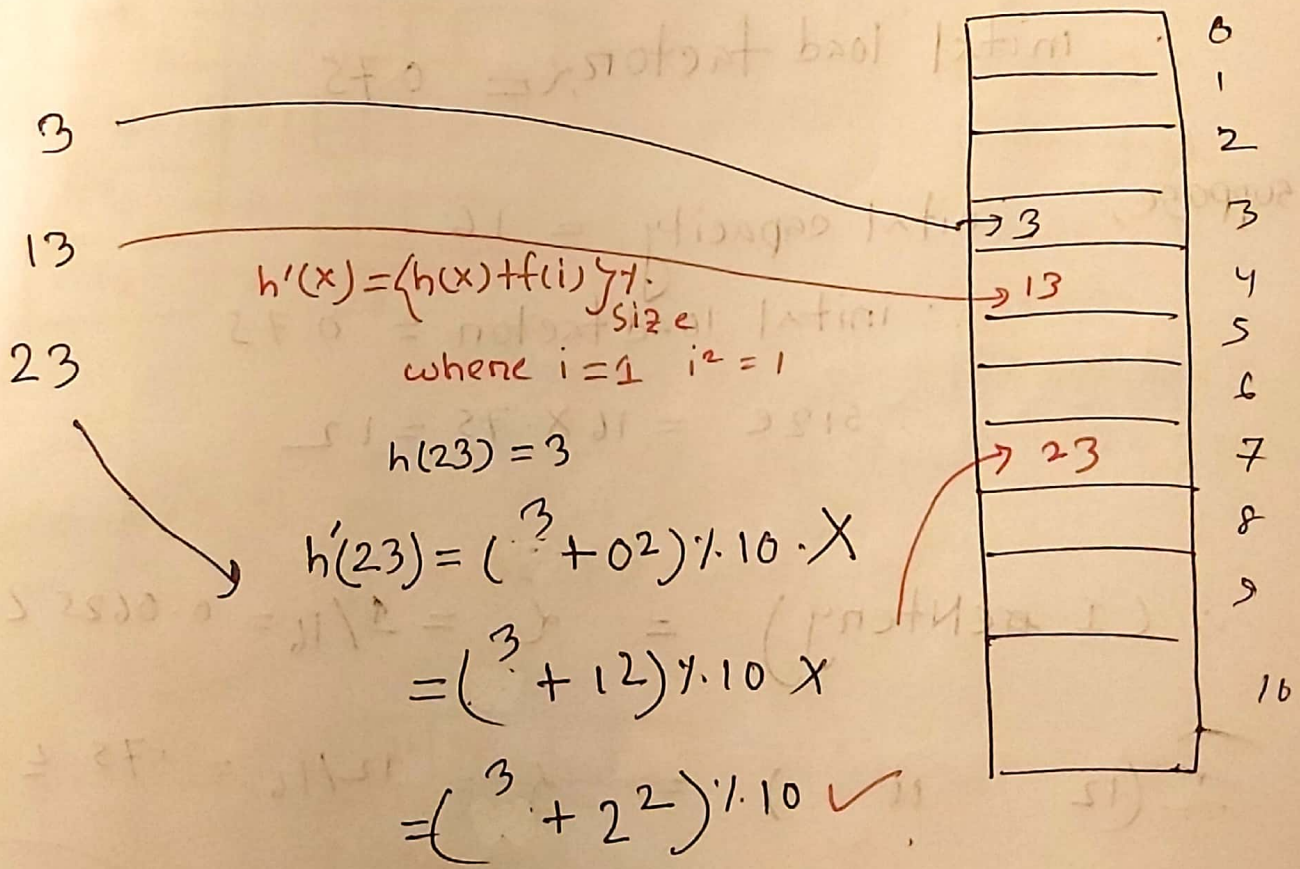
Key — Hash function — Hash table

$\therefore h(x) = x \% \text{size}$

$\therefore h'(x) = \{h(x) + f(i)\} \% \text{size}$

$f(i) = i^2$

$i = 0, 1, 2, 3 \ldots$



Free space finding using Quadric

what is load factor in hashmap? (α)

no. key values pe... slot =

→ load factor is the measure that decide when to increase the capacity of the HashMap

$$\therefore \text{load factor} = \alpha = \frac{n \text{ (no. keys)} \quad \text{(no. enteries in map)}}{m \text{ (table size)} \quad \text{(total size of hashmap)}}$$

$\therefore$ initial load factor, $\alpha = 0.75$

suppose, Initial capacity = 16

$\therefore$ initial load factor = 0.75

$\therefore$ size = $16 \times .75 = 12$

$\therefore$ (1 entery) = $\alpha = 1/16 = 0.0625 < .75$

$\therefore$ (12 ") = $\alpha = 12/16 = .75 \leq .75$

$\therefore$ (13 ") = $\alpha = 13/16 = .8125 > .75$

need to increase the hashmap size

what is perfect hashing?

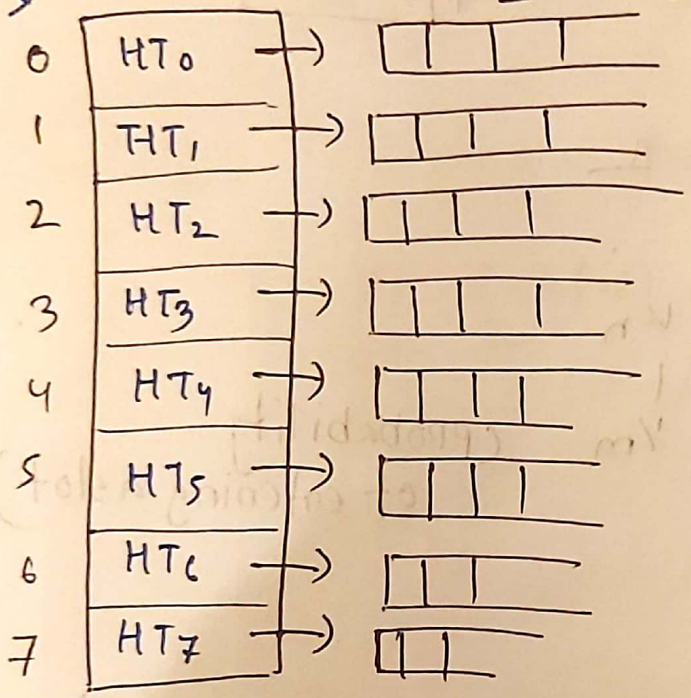
2 table

- Pimany PHT hashtable
- Secondany hash table HT

worst
→ search ~~eas~~ time — O(1)
→ O(n) — worst space

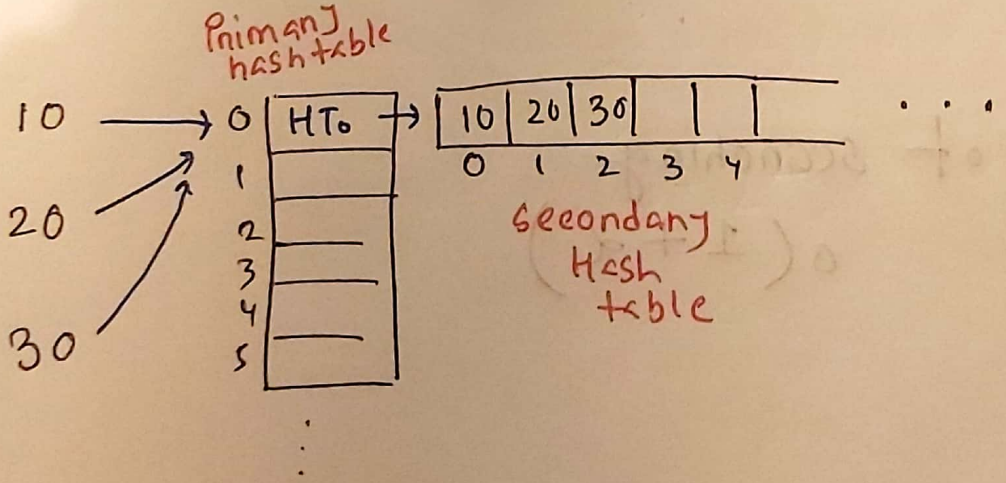
(Pimany hash table)

PHT



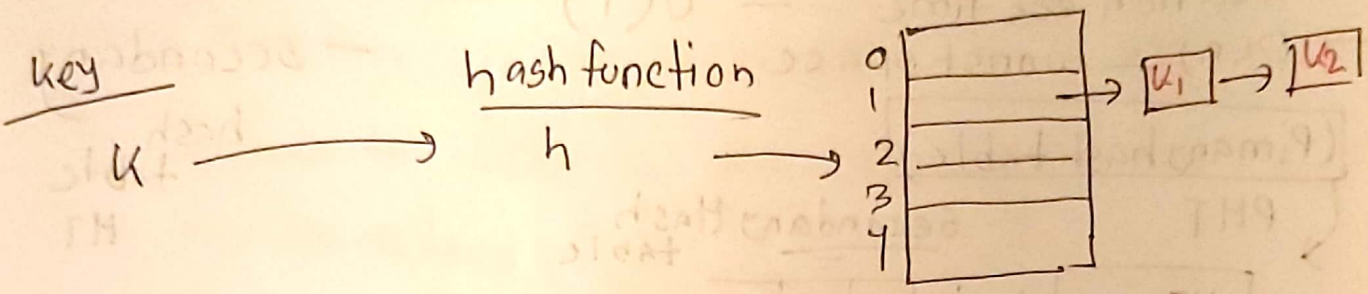
Hash table এর Each Hash table So that সবচেয়ে কম collision হয়।

How it works?



Uniform hashing

∴ n = no. keys
∴ m = no. slots



$\therefore h(k_1) = h(k_2) = i = 1$

k_1	k_2	....	k_n	
1	1		1	
1/m	1/m		1/m	(probability of entering a slot)

average length of chain in each of the slots =

$\therefore$ loadfaeton $\alpha = \frac{n}{m}$

time complexity of seaching

$o(1+\alpha)$

simple uniform hashing

$$\Pr_{k_1 \neq k_2} \{ h(k_1) = h(k_2) \} = \frac{1}{m}$$

Universal hashing

a random hash function

$$\therefore \Pr_{h \in H} \{ h(k) = h(k') \} \leq \frac{1}{m} \quad \text{for all } k \neq k'$$

Pobability of collision in Hashing =

Double Hashing

(a method of collision Handling)

Key

Hash function

$h(k) = h_1(k) + j\,h_2(k)$

$\therefore h_1(k) = k \bmod 13$

$\therefore h_2(k) = 7 - k \bmod 7$

$\therefore j = 0, 1, 2, 3$

Hash table

18

22

44

$h(18) = h_1(18) + 0 \times h_2(18)$

$h(22) = h_1(22) + 0\,h_2(22)$

$h(44) = h_1(44) + 0\,h_2(44)$

No space

$h(44) = h_1(44) + 1\,h_2(44)$