Dictionary

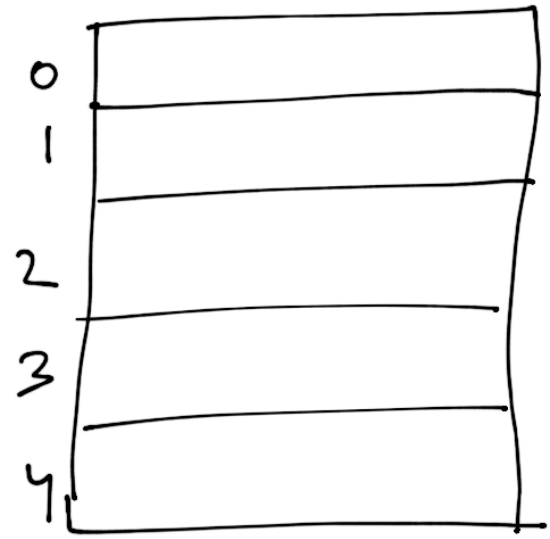
- ADT
- Maintain a set of items, each with a key
  - insert (item)
  - delete (item)
  - search (key)

Motivation

- database
- password/login
- compilers & interpreters
- network router
- String commonalities

Direct Access Table

→ Array where index can work as key



Badness

(1) Keys may not be non-negative integers

(2) gigantic memory hog

prehash : Keys to non-negative integers

Hashing

→ Searching Technique

→ O(1)

Key = 6, 26

$h(k_i) = k_i \% m$

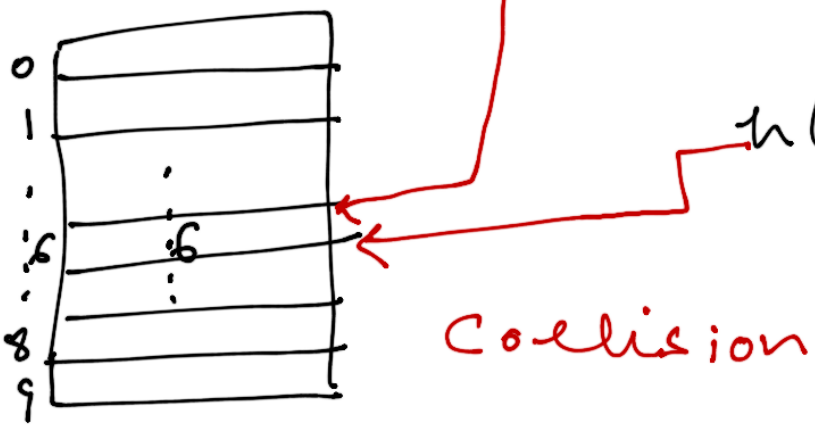
Size of Hash table (m) = 10

$h(6) = 6 \% 10$

$= 6$

$h(26) = 26 \% 10$

$= 6$



Collision

(1) Open Hashing (closed Addressing)

↳ Chaining

(2) Closed Hashing (Open Addressing)

- Linear Probing
- Quadratic Probing
- Double Hashing

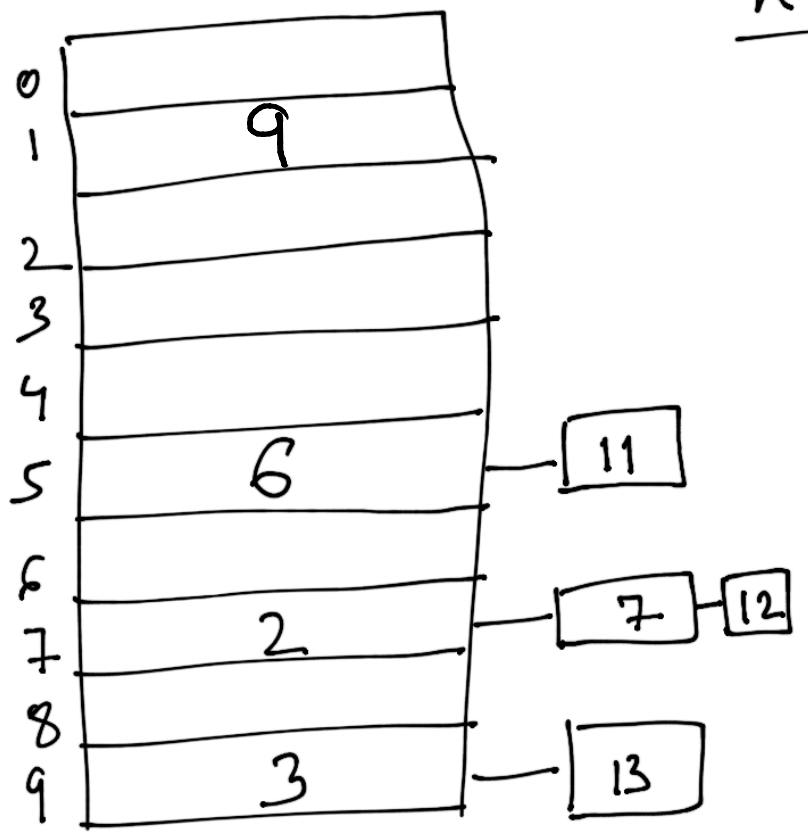
Chaining

$K = 3, 2, 9, 6, 11, 13, 7, 12$

$h(K) = 2K + 3$, $m = 10$

Use Division method $K \% m$

closed Addressing to store the value



Key	Location (K % m)
3	$(2 \times 3 + 3) \% 10 = 9$
2	$(2 \times 2 + 3) \% 10 = 7$
9	$(2 \times 9 + 3) \% 10 = 1$
6	$(2 \times 6 + 3) \% 10 = 5$
11	$(2 \times 11 + 3) \% 10 = 5$
13	$(2 \times 13 + 3) \% 10 = 9$
7	$(2 \times 7 + 3) \% 10 = 7$
12	$(2 \times 12 + 3) \% 10 = 7$

Linear Probing

$h(k) = 2k+3$, $k = 3, 2, 9, 6, 11, 13, 7, 12$

$m = 10$

Use Division method &

Open Addressing to store the value

* Insert k_i at first free location from $(h+i)\%.m$ where $i = 0, 1, \ldots, 9$

Key	Location	Probes
3	$(2\times3+3)\%.10 = 9$	1
2	$(2\times2+3)\%.10 = 7$	1
9	$(2\times9+3)\%.10 = 1$	1
6	$(2\times6+3)\%.10 = 5$	1
11	$(2\times11+3)\%.10 = 5$	2
13	$(2\times13+3)\%.10 = 9$	2
7	$(2\times7+3)\%.10 = 7$	2
12	$(2\times12+3)\%.10 = 7$	6

0	13
1	9
2	12
3	
4	
5	6
6	11
7	2
8	7
9	3

Order of Elements

13, 9, 12, —, —, 6, 11, 2, 7, 3

Quadratic Probing

Insert K_i at first free location from $(u+i^2)\%m$ where $i=0,1,\ldots,m-1$

$K = 3, 2, 9, 6, 11, 13, 7, 12$ $\quad m = 10, h(k) = 2k+3$

Use Division method & Quadratic Probing to share these values

Key	Location (u)	Probe
3	$(2\times3+3)\%10 = 9$	1
2	$(2\times2+3)\%10 = 7$	1
9	$(2\times9+3)\%10 = 1$	1
6	$(2\times6+3)\%10 = 5$	1
→ 11	$(2\times11+3)\%10 = 5$	2
13	$(2\times13+3)\%10 = 9$	2
7	$(2\times7+3)\%10 = 7$	2
12	$(2\times12+3)\%10 = 7$	5

0	13
1	9
2	
3	12
4	
5	6
6	11
7	2
8	7
9	3

for 11
u = 5
$(5+0)\%10 = 5$
$(5+1)\%10 = 6$

for 13
u = 9
$(9+0)\%10 = 9$
$(9+1)\%10 = 0$

Order: 13, 9, —, 12, —, 6, 11, 2, 7, 3

Double Hashing

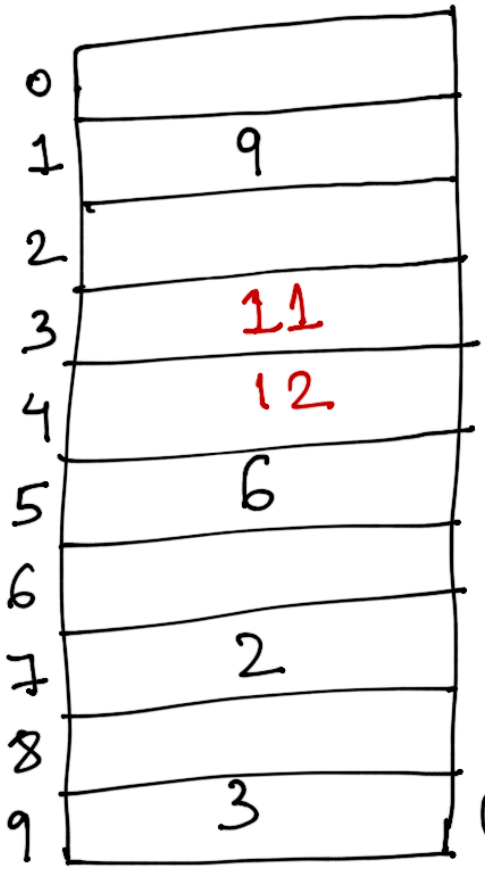
$K = 3, 2, 9, 6, 11, 13, 7, 12$

$h_1(K) = 2K+3$, $m = 10$

Use division method & double hashing to insert these elements

$h_2(K) = 3K+1$

Insert K_i at first free location from $(u + v*i) \% m$, $i = 0$ to $m-1$, $v = h_2(K) \% m$



for 11

$u = 5$

$v = (3K+1) \% m$

$= (3 \times 11 + 1) \% 10$

$= 34 \% 10$

$= 4$

$(u + v*i) \% m$

$= (5 + 4*0) \% 10$

$= 5 \% 10 = 5$

$(5 + 4*1) \% 10$

$= 9 \% 10 = 1$

$(5 + 4*2) \% 10$

$= 13 \% 10 = 3$

Key	Location(u)	v	Probe
3	$(2\times3+3) \% 10 = 9$	–	1
2	$(2\times2+3) \% 10 = 7$	–	1
9	$(2\times9+3) \% 10 = 1$	–	1
6	$(2\times6+3) \% 10 = 5$	–	1
11	$(2\times11+3) \% 10 = 5$	4	3
13	$(2\times13+3) \% 10 = 9$		X
7	$(2\times7+3) \% 10 = 7$	2	X
12	$(2\times12+3) \% 10 = 7$	7	2

Good Hash functions

1. Easy to compute $H(n)$ $O(1)$

2. Even distribution

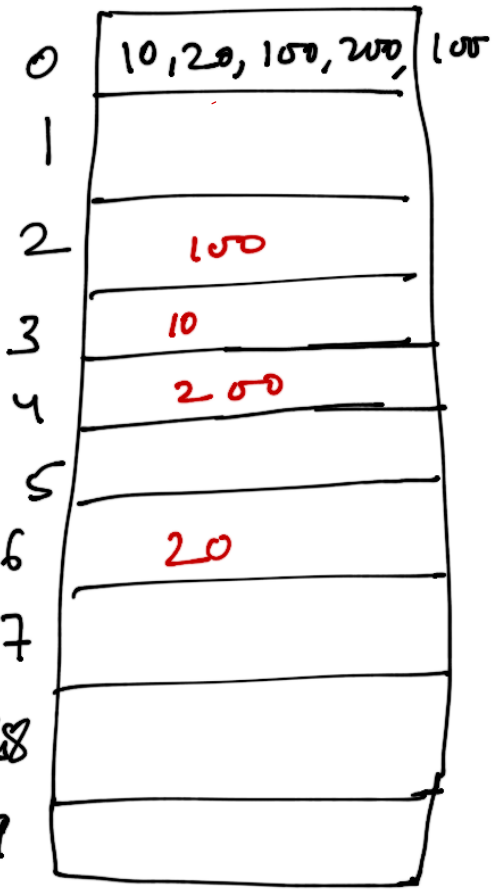
$$H(n) = n \bmod 10$$

Keys : 10, 20, 100, 200, 1000

$H(n) = n \bmod 10$

This resulting into more collisions.

$$H'(n) = n \bmod 7$$



3. Maintain loadfactor less

λ = Load factor

$= \frac{N}{T}$ ← No. of Elements in the hashtable

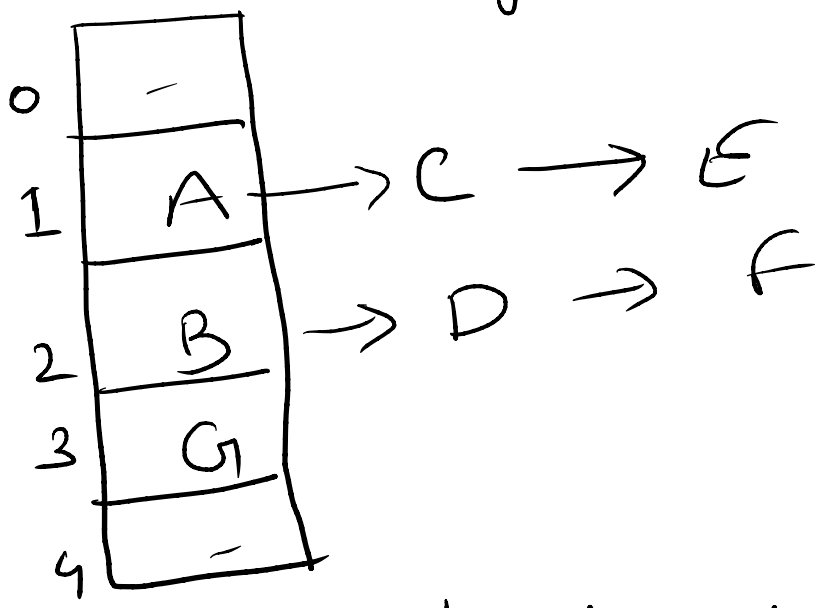
T ← Table size ?

→ Low 'λ' factor means less collisions

Eg: Given a hash table with 25 slots that stores 2000 elements,

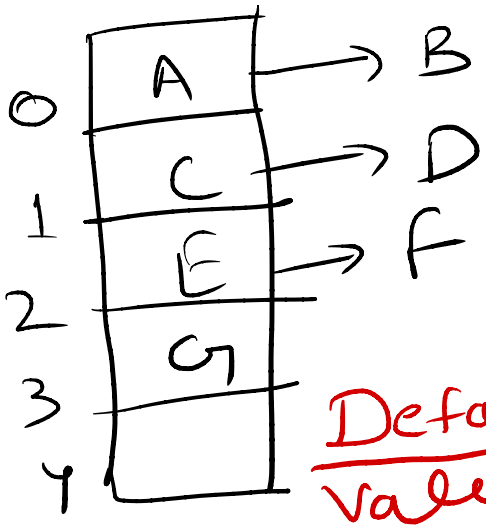
Load factor (λ) $= \frac{2000}{25} = 80$

Key values: A, B, C, D, E, F, G



$$\text{Load factor} = \frac{\text{Total stored elements}}{\text{Size of the table}}$$

$$= \frac{7}{5} = 1.4$$



Load factor $= \frac{7}{5} = 1.4$

Default value : preferable to keep Load factor < 0.75

Hashmap	Hashtable
* Non-synchronized Not thread-safe & can't be shared b/w multiple threads.	* Synchronized Thred-safe & can be shared b/w many threads.
* It allows null Key and value.	* It doesn't allow null key or value.
* It is fast.	* It is comparatively slow.