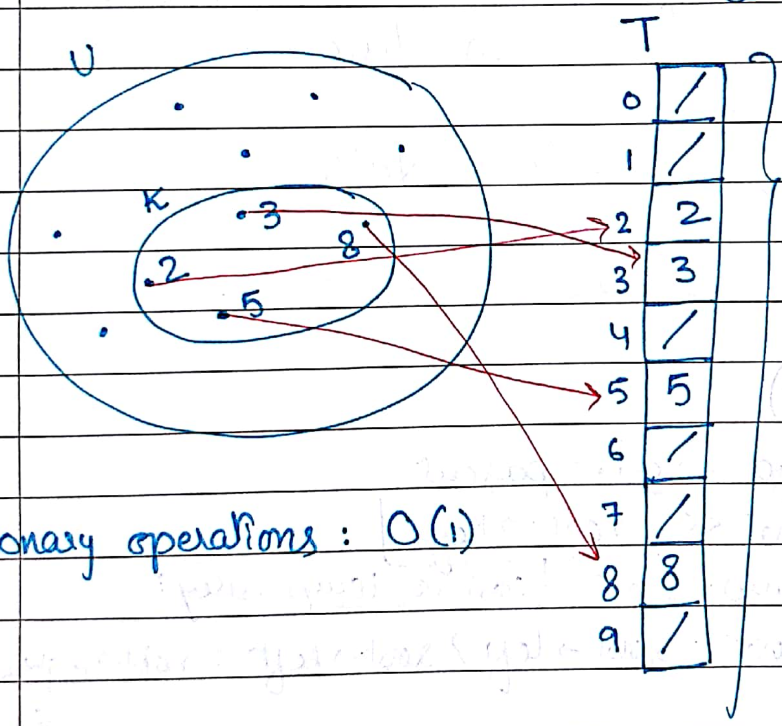
* Hashing

- Many applications need dynamic set data structure to store elements indexed using keys.
- Supports INSERT, DELETE, SEARCH
- Example: symbol table, memory management
- Dynamic set data structure can effectively be implemented using hash-tables.

* Direct address tables : Ordinary arrays

To store a key drawn from $U = \{0, 1 \ldots m-1\}$ where m is not too large and all keys are unique. (assume)



Direct address table

$T[\text{Key}] = \text{Key}$

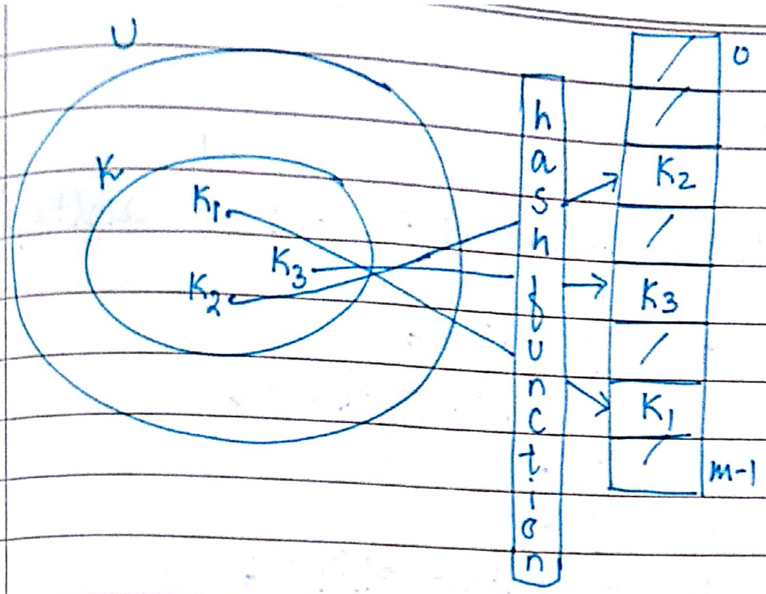
Dictionary operations : $O(1)$

* Hash-tables

$|K| \ll |U| \rightarrow$ Not practical : memory wastage

$h : U \rightarrow \{0, 1, \ldots m-1\}$ | Key hashes to $h(\text{Key})$

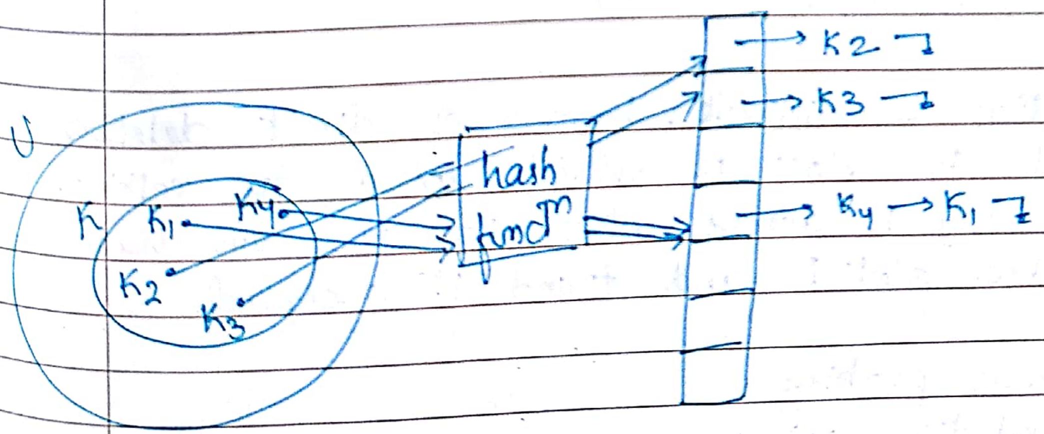
$T[h(\text{Key})] = \text{Key}$ | $h(\text{Key})$ is the hash value of Key



* Hash function
  - ↳ simple to compute.
  - ↳ must distribute keys evenly among cells.
  - ↳ perfect hash functions can be designed if keys are known in advance.

collisions : two keys hash to the same slot.
↳ can't be avoided but can be resolved.

* Collision Resolution Techniques
  - ↳ Chaining
  - ↳ Open Addressing



INSERT : $O(1)$ → assuming element is not present already.

SEARCH : O(length of the list). : $\Theta(1+\alpha)$

DELETE : $O($ → singly linked list

$O(1)$ → doubly linked list

α = load factor = n/m : hash table T has m slots to store n elements.

Example : Chaining

$K = \{0, 1, 4, 9, 16, 25, 36, 49, 64, 81\}$ | $hash(Key) = Key \% 10$

0: 0%10 = 0	0		→ [0] → ⊥
1: 1%10 = 1	1		→ [81] → [1] → ⊥
4: 4%10 = 4	2		⊥
9: 9%10 = 9	3		⊥
16: 16%10 = 6	4		→ [64] → [4] → ⊥
25: 25%10 = 5	5		→ [25] → ⊥
36: 36%10 = 6	6		→ [36] → [16] → ⊥
49: 49%10 = 9	7		⊥
64: 64%10 = 4	8		⊥
81: 81%10 = 1	9		→ [49] → [9] → ⊥

* Open addressing

↳ No lists. All elements occupy hash table itself

↳ Idea is to successively examine or probe the hash table till an empty slot is found.

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

$T[h(Key, i)] = Key$

Deletion is difficult. Instead of direct deletion mark the slot as deleted so as to retrieve any key k during whose insertion we had probed slot i and found it occupied.

① Linear probing
② Quadratic probing
③ Double hashing

* Linear Probing

$h(k,i) = (h'(k) + i) \mod m$

where $i = \{0, 1 \ldots m-1\}$

$T[h(k,i)] = k$

Probe sequence:

$T[h'(k)], T[h'(k)+1], \ldots$

- There are only m distinct probe sequences.
- Easy to implement.
- Problem of primary clustering, increase in average search time

Example: Linear Probing

$K = \{89, 18, 49, 58, 9\}$ $m = 10$

$h(k,i) = (h'(k) + i) \mod m$

where $h'(k) = k \mod m$

89: $h(89,0) = (89\%10 + 0)\%10$
$= 9\%10 = 9$

$T[9] = 89$ ✓

18: $h(18,0) = (18\%10 + 0)\%10$
$= 8\%10 = 8$

$T[8] = 18$ ✓

49: $h(49,0) = (49\%10 + 0)\%10$
$= 9\%10 = 9$

$T[9]$ = occupied.

$h(49,1) = (49\%10 + 1)\%10$
$= (9+1)\%10 = 10\%10 = 0$

$T[0] = 49$ ✓

0	49	2
1	58	4
2	9	4
3		
4		
5		
6		
7		
8	18	1
9	89	1

58: $h(58,0)$
$= (58\%10 + 0)\%10$
$= 8\%10 = 8$

$T[8]$ = occupied

$h(58,1) = (58\%10 + 1)\%10$
$= 9\%10 = 9$

$T[9]$ = occupied

$h[58,2] = (58\%10 + 2)\%10$
$= 10\%10 = 0$

$T[0]$ = occupied

$h(58,3) = (58\%10 + 3)\%10$
$= 11\%10 = 1$

$T[1] = 58$ ✓

9: Probe sequence followed is 9, 0, 1, 2

$T[2] = 9$ ✓

PAGE

classmate

Example: Linear Probing

K = {34, 55, 12, 8, 45, 37, 32, 88, 98, 54, 21, 42, 56, 74, 52, 33, 16}

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Value	74	21	42	52	33	45	16	/	8	88	/	/	12	32	34	55	54	37	98	56
#Probes	7	1	1	12	12	1	11	/	1	2	/	/	1	2	1	1	3	1	1	4

34: 34 % 20 = 14 ✓
55: 55 % 20 = 15 ✓
12: 12 % 20 = 12 ✓
8: 8 % 20 = 8 ✓
45: 45 % 20 = 5 ✓
37: 37 % 20 = 17 ✓
32: 32 % 20 = 12 x, 13 ✓
88: 88 % 20 = 8 x, 9 ✓
98: 98 % 20 = 18 ✓
54: 54 % 20 = 14 x, 15 x, 16 ✓
21: 21 % 20 = 1 ✓
42: 42 % 20 = 2 ✓
56: 56 % 20 = 16 x, 17 x, 18 x, 19 ✓
74: 74 % 20 = 14 x, 15 x, 16 x, 17 x, 18 x, 19 x, 0 ✓
52: 52 % 20 = 12 x, 13 x, 14 x, 15 x, 16 x, 17 x, 18 x, 19 x, 0 x, 1 x, 2 x, 3 ✓
33: 33 % 20 = 13 x, 14 x, 15 x, 16 x, 17 x, 18 x, 19 x, 0 x, 1 x, 2 x, 3 x, 4 ✓
16: 16 % 20 = 16 x, 17 x, 18 x, 19 x, 0 x, 1 x, 2 x, 3 x, 4 x, 5 x, 6 ✓

* Quadratic Probing

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

leads to milder form of clustering, known as secondary clustering

Scanned by CamScanner

Example: Quadractic Probing

$k = \{89, 18, 49, 58, 9\}$ $m = 10$

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

where $h'(k) = k \bmod m$

	value	#probes
0	49	2
1		
2	58	3
3	9	3
4		
5		
6		
7		
8	18	1
9	89	1

89: $(89\%10 + 0^2) \bmod 10$
$= 9\%10 = 9$ ✓

18: $(18\%10 + 0^2)\%10$
$= 8\%10 = 8$ ✓

49: $(49\%10 + 0^2)\%10$
$= 9\%10 = 9$ ✗

$(49\%10 + 1^2)\%10$
$= (9+1)\%10 = 0$ ✓

58: $(58\%10 + 0^2)\%10$
$= 8\%10 = 8$ ✗

$(58\%10 + 1^2)\%10$
$= (8+1)\%10 = 9$ ✗

$(58\%10 + 2^2)\%10$
$= (8+4)\%10 = 2$ ✓

9: $(9\%10 + 0^2)\%10$
$= 9\%10 = 9$ ✗

$(9\%10 + 1^2)\%10$
$= (9+1)\%10 = 0$ ✗

$(9\%10 + 2^2)\%10$
$= (9+4)\%10 = 3$ ✓

Problem: Not sure if all the available slots are probed.

Theorem: $m \to$ prime
$\alpha \leq 0.5$
the all probes will be to different locations.

PAGE

Example: Quadratic Probing

K = { 34, 55, 12, 8, 45, 37, 32, 88, 98, 54, 21, 42, 56, 74, 52, 33, 16 }

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

34: 34 % 20 = 14 ✓

55: 55 % 20 = 15 ✓

12: 12 % 20 = 12 ✓

8: 8 % 20 = 8 ✓

45: 45 % 20 = 5 ✓

37: 37 % 20 = 17 ✓

32: 32 % 20 = 12 ×, $12 + 1^2 = 13$ ✓

88: 88 % 20 = 8 ×, $8 + 1^2 = 9$ ✓

98: 98 % 20 = 18 ✓

54: 54 % 20 = 14 ×, $14 + 1^2 = 15$ ×, $14 + 2^2 = 18$ ×, $14 + 3^2 = 23$ % 20 = 3 ✓

21: 21 % 20 = 1 ✓

42: 42 % 20 = 2 ✓

56: 56 % 20 = 16 ✓

74: 74 % 20 = 14 ×, $14 + 1^2 = 15$ ×, $14 + 2^2 = 18$ ×, $14 + 3^2 = 23$ % 20 = 3 ×
$14 + 4^2 = 30$ % 20 = 10 ✓

52: 52 % 20 = 12 ×, $12 + 1^2 = 13$ ×, $12 + 2^2 = 16$ ×, $12 + 3^2 = 21$ % 20 = 1 ×
$12 + 4^2 = 28$ % 20 = 8 ×, $12 + 5^2 = 37$ % 20 = 17 ×
$12 + 6^2 = 48$ % 20 = 8 ×, $12 + 7^2 = 61$ % 20 = 1 ×
$12 + 8^2 = 76$ % 20 = 16 ×, $12 + 9^2 = 93$ % 20 = 13 ×
$12 + 10^2 = 112$ % 20 = 12 ×, $12 + 11^2 = 133$ % 20 = 13 ×
$12 + 12^2 = 156$ % 20 = 16 ×, $12 + 13^2 = 181$ % 20 = 1 ×
$12 + 14^2 = 208$ % 20 = 8 ×, $12 + 15^2 = 237$ % 20 = 17 ×
$12 + 16^2 = 268$ % 20 = 8 ×, $12 + 17^2 = 301$ % 20 = 1 ×
$12 + 18^2 = 336$ % 20 = 16 ×, $12 + 19^2 = 373$ % 20 = 13 ×

⇒ 52 cannot be inserted.

Similarly, 33 cannot be inserted.

16: 16 % 20 = 16 ×, $16 + 1^2 = 17$ ×
$16 + 2^2 = 20$ % 20 = 0 ✓

* **Double hashing**

↳ One of the best methods for open addressing

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

Example:

$k = \{34, 55, 12, 8, 45, 37, 32, 88, 98, 54, 21, 42, 56, 74, 52, 33, 16\}$

$h_1(k) = k \% 20$; $h_2(k) = k \% 6 + 1$

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 × 56%6+1 = 3

= 16+3 = 19 ✓

74: 74%20 = 14 × 74%6+1 = 3

= 14+3 = 17 × , 14+2×3 = 20% 20 = 0 ✓

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

(12+2×3)%20 = 18%20 = 18 ✓

88: 88% 20 = 8 × 88%6 +1 = 5

8+5 = 13 ✓

98: 98% 20 = 18 × 98%6+1 = 3.

18+3 = 21%20 = 1 ✓

54: 54%20 = 14 × 54%6+1 = 1

14+1 = 15 × , 14+2×1 = 16 ✓

21: 21%20 = 1 × 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 ✓