# HASHING

HASH TABLE SUPPORTS ONLY SUBSET OF OPERATIONS ALLOWED BY BINARY SEARCH TREES ( insert, search..)
NOT SUITABLE FOR OPERATIONS LIKE MIN, MAX, PRINT IN ORDER, etc.

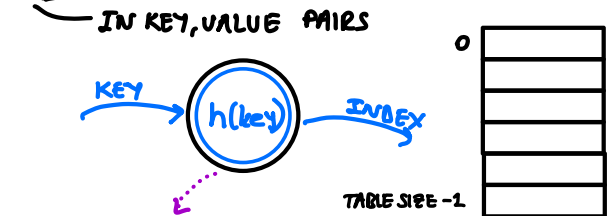
HASHING IS A TECNIQUE USED FOR PERFORMING INSERTION, DELETION AND FINDS IN CONSTANT TIME (O(1))

THE IMPLEMENTATION OF HASH TABLES IS CALLED HASHING

HASH TABLE IS AN ARRAY OF SOME FIXED SIZE, CONTAINING THE ITEMS
IN KEY, VALUE PAIRS

LOAD FACTOR = $\alpha$
- RATIO OF # OF ELEMENTS (N) IN A HASH TABLE TO THE HASH TABLE-SIZE

$\alpha$ = N / TABLE-SIZE

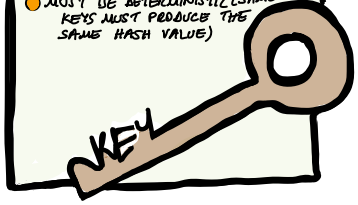



EACH KEY IS MAPPED INTO AN INDEX INTO HASH TABLE

HASH FUNCTION

- MUST BE SIMPLE
- MUST DISTRIBUTE KEYS EVENLY
- MUST BE DETERMINISTIC (SAME KEYS MUST PRODUCE THE SAME HASH VALUE)

IF WE KNOW WHICH KEYS WILL OCCUR IN ADVANCE, WE CAN WRITE PERFECT HASH FUNCTIONS BUT WE DON'T




## SOME HASH FUNCTIONS

### IF KEYS ARE NUMERIC

Key MOD N (N IS THE SIZE OF THE TABLE)

TRUNCATE: 123456789 MAP TO A TABLE OF 1000 BY PICKING THE FIRST THREE DIGITS

FOLDING: 123|456|789 ADD THEM AND TAKE MOD

:

### IF KEYS ARE STRINGS

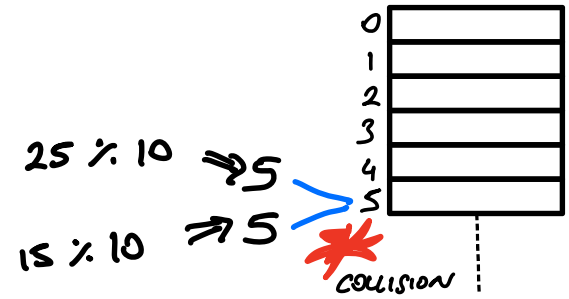
$$\left[ \text{hash(key)} = \sum_{i=0}^{KeySize-1} Key[KeySize - i - 1] \cdot 37^{i} \right]$$

Ex: key = CS

hash("CS") = $(67 \cdot 37^{0} + 83 * 37^{1})$ % TABLE SIZE

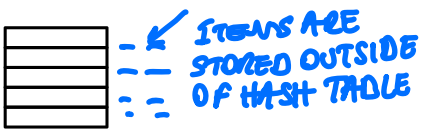
# COLLISIONS AND COLLISION RESOLUTION

IF MORE THAN ONE KEY MAPS TO THE SAME HASH VALUE A COLLISION OCCURS

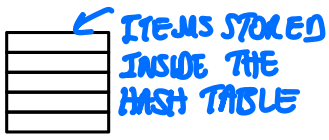



## COLLISION RESOLUTION STRATEGIES

- SEPARATE CHAINING




- OPEN ADDRESSING
  - -LINEAR PROBING
  - -QUADRATIC PROBING
  - -DOUBLE HASHING

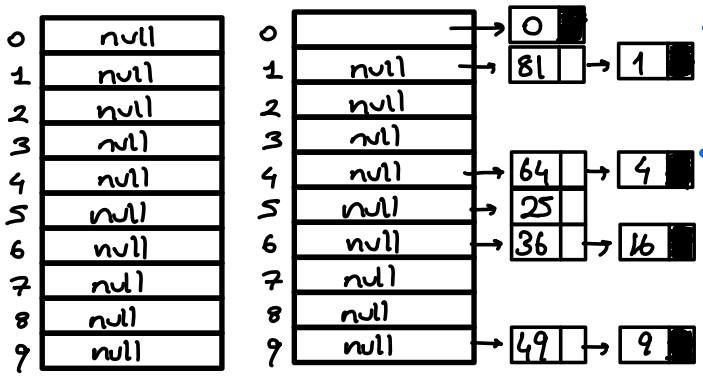



**SEPARATE CHAINING**: KEEP THE LIST OF ITEMS THAT HASH TO THE SAME VALUE

**EXAMPLE:** 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 ← KEYS, IGNORE VALUES FOR NOW

ASSUME hash(key) = key % TABLE_SIZE ← ASSUME 10.

0↓
h(0) = 0 % 10 = 0




ITEMS ARE STORED OUTSIDE OF HASH TABLE

INSTEAD OF LINKED LISTS, BSTS CAN ALSO BE USED

## OPERATIONS

- **INITIALIZATION:**

  ALL ENTRIES ARE SET TO NULL

  ```
  LinkedList[] hashTable = new LinkedList[10];
  ```

- **SEARCH**

  LOCATE THE CELL USING HASH FUNCTION
  SEQUENTIAL SEARCH ON THE LINKED LIST IN THAT INDEX

- **INSERT**

  LOCATE THE CELL USING HASH FUNCTION
  INSERT FIRST INTO LINKED LIST (IF ITEM DOES NOT EXIST)

- **DELETE**

  LOCATE THE CELL USING HASH FUNCTION
  DELETE THE ITEM FROM THE LINKEDLIST

## ALGORITHM ANALYSIS

★ COLLISIONS ARE VERY LIKELY

★ SEPARATE CHAINING: $\alpha$ IS NOT BOUND BY 1; IT CAN BE $> 1$.

★ COST = CONSTANT TIME TO EVALUATE HASH FUNCTION + TIME TO TRAVERSE THE LIST
$

★ UNSUCCESSFUL SEARCH: TRAVERSE THE ENTIRE LIST, SO WE NEED TO COMPARE $\alpha$ NODES ON THE AVERAGE

★ SUCCESSFUL SEARCH: AVERAGE SEARCH COST $= 1 + \frac{\alpha}{2}$

ON THE AVERAGE, WE NEED TO CHECK HALF OF THE OTHER NODES

LOAD FACTOR IS MORE IMPORTANT THAN TABLE SIZE.

KEEP LOAD FACTOR AROUND 1

# OPEN ADDRESSING

- LINEAR PROBING
- QUADRATIC PROBING
- DOUBLE HASHING

$$hash_i(key) = [h(key) + f(i)] \% TABLE\_SIZE$$

$f(i) = i$ → LINEAR PROBING

$f(i) = i^2$ → QUADRATIC PROBING

$f(i) = i \cdot h_2(key)$ → DOUBLE HASHING

## LINEAR PROBING $f(i) = i$

**EXAMPLE:** 0, 1, 4, 9, 16, 25, 36, 49, 64, 81

ASSUME $h(key) = key \% TABLE\_SIZE$

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 49 |
| 3 | 81 |
| 4 | 4 |
| 5 | 25 |
| 6 | 16 |
| 7 | 36 |
| 8 | 64 |
| 9 | 9 |

$hash_0(0) = (h(0) + f(0)) \% 10 = 0$

$hash_0(1) = (h(1) + f(0)) \% 10 = 1$

$hash_0(4) = (h(4) + f(0)) \% 10 = 4$

$hash_0(9) = (h(9) + f(0)) \% 10 = 9$

$hash_0(16) = (h(16) + f(0)) \% 10 = 6$

$hash_0(25) = (h(25) + f(0)) \% 10 = 5$

$hash_0(36) = (h(36) + f(0)) \% 10 = 6$ ✱ collision

$hash_1(36) = (h(36) + f(1)) \% 10 = 7$

$hash_0(49) = (h(49) + f(0)) \% 10 = 9$ ✱ collision

$hash_1(49) = (h(49) + f(1)) \% 10 = 0$ ✱ collision

$hash_2(49) = (h(49) + f(2)) \% 10 = 1$ ✱ collision

$hash_3(49) = (h(49) + f(3)) \% 10 = 2$

$hash_0(64) = (h(64) + f(0)) \% 10 = 4$ ✱ collision

$hash_1(64) = (h(64) + f(1)) \% 10 = 5$ ✱ collision

$hash_2(64) = (h(64) + f(2)) \% 10 = 6$ ✱ collision

$hash_3(64) = (h(64) + f(3)) \% 10 = 7$ ✱ collision

$hash_4(64) = (h(64) + f(4)) \% 10 = 8$

$hash_0(81) = (h(81) + f(0)) \% 10 = 1$ ✱ collision

$hash_1(81) = (h(81) + f(1)) \% 10 = 2$ ✱ collision

$hash_2(81) = (h(81) + f(2)) \% 10 = 3$

# QUADRATIC PROBING $f(i)=i^2$

**EXAMPLE:** 0, 1, 4, 9, 16, 25, 36, 49, 64, 81

ASSUME $h(key) = key \% TABLE\_SIZE$

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 81 |
| 3 | 49 |
| 4 | 4 |
| 5 | 25 |
| 6 | 16 |
| 7 | 36 |
| 8 | 64 |
| 9 | 9 |

$hash_0(0) = (h(0) + f(0)) \% 10 = 0$

$hash_0(1) = (h(1) + f(0)) \% 10 = 1$

$hash_0(4) = (h(4) + f(0)) \% 10 = 4$

$hash_0(9) = (h(9) + f(0)) \% 10 = 9$

$hash_0(16) = (h(16) + f(0)) \% 10 = 6$

$hash_0(25) = (h(25) + f(0)) \% 10 = 5$

$hash_0(36) = (h(36) + f(0)) \% 10 = 6$ ✱ collision

$hash_1(36) = (h(36) + f(1)) \% 10 = 7$

$hash_0(49) = (h(49) + f(0)) \% 10 = 9$ ✱ collision

$hash_1(49) = (h(49) + f(1)) \% 10 = 0$ ✱ collision

$hash_2(49) = (h(49) + f(2)) \% 10 = 3$

$hash_0(64) = (h(64) + f(0)) \% 10 = 4$ ✱ collision

$hash_1(64) = (h(64) + f(1)) \% 10 = 5$ ✱ collision

$hash_2(64) = (h(64) + f(2)) \% 10 = 8$

$hash_0(81) = (h(81) + f(0)) \% 10 = 1$ ✱ collision

$hash_1(81) = (h(81) + f(1)) \% 10 = 2$ ✱ collision

$hash_2(81) = (h(81) + f(2)) \% 10 = 5$ ✱ collision

$hash_3(81) = (h(81) + f(3)) \% 10 = 0$ ✱ collision

$hash_4(81) = (h(81) + f(4)) \% 10 = 7$ ✱ collision

$hash_5(81) = (h(81) + f(5)) \% 10 = 6$ ✱ collision

$hash_6(81) = (h(81) + f(6)) \% 10 = 7$ ✱ collision

$hash_7(81) = (h(81) + f(7)) \% 10 = 0$ ✱ collision

$hash_8(81) = (h(81) + f(8)) \% 10 = 5$ ✱ collision

$hash_9(81) = (h(81) + f(9)) \% 10 = 2$

# QUADRATIC PROBING $f(i) = i \cdot h_2(key)$

**EXAMPLE:** 0, 1, 4, 9, 16, 25, 36, 49, 64, 81

ASSUME $h_1(key) = key \ \% \ TABLE\_SIZE$ $h_2 = 5 - (key \ \% 5)$

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 49 |
| 3 | 81 |
| 4 | 4 |
| 5 | 25 |
| 6 | 16 |
| 7 | 64 |
| 8 | 36 |
| 9 | 9 |

$hash_0(0) = (h(0) + f(0)) \% 10 = 0$

$hash_0(1) = (h(1) + f(0)) \% 10 = 1$

$hash_0(4) = (h(4) + f(0)) \% 10 = 4$

$hash_0(9) = (h(9) + f(0)) \% 10 = 9$

$hash_0(16) = (h(16) + f(0)) \% 10 = 6$

$hash_0(25) = (h(25) + f(0)) \% 10 = 5$

$hash_0(36) = (h(36) + f(0)) \% 10 = 6$ *
$hash_1(36) = (h(36) + f(1)) \% 10 = 0$ * 1. $(5 - (6 \% 5))$
$hash_2(36) = (h(36) + f(2)) \% 10 = 4$ * 2. $(5 - (6 \% 5))$
$hash_3(36) = (h(36) + f(3)) \% 10 = 8$ 3. $(5 - (6 \% 5))$

$hash_0(49) = (h(49) + f(0)) \% 10 = 9$ *
$hash_1(49) = (h(49) + f(1)) \% 10 = 0$ * 1. $(5 - (49 \% 5))$
$hash_2(49) = (h(49) + f(2)) \% 10 = 1$ * 2. $(5 - (49 \% 5))$
$hash_3(49) = (h(49) + f(3)) \% 10 = 2$ 3. $(5 - (49 \% 5))$

$hash_0(64) = (h(64) + f(0)) \% 10 = 4$ *
$hash_1(64) = (h(64) + f(1)) \% 10 = 5$ * 1. $(5 - (64 \% 5))$
$hash_2(64) = (h(64) + f(2)) \% 10 = 8$ * 2. $(5 - (64 \% 5))$
$hash_3(64) = (h(64) + f(3)) \% 10 = 7$ 3. $(5 - (64 \% 5))$

$hash_0(81) = (h(81) + f(0)) \% 10 = 1$ *
$hash_1(81) = (h(81) + f(1)) \% 10 = 5$ * 1. $(5 - (81 \% 5))$
$hash_2(81) = (h(81) + f(2)) \% 10 = 9$ * 2. $(5 - (81 \% 5))$
$hash_3(81) = (h(81) + f(3)) \% 10 = 3$ 3. $(5 - (81 \% 5))$