HASH TABLES

Mahsa Ziraksima
ACIBADEM UNIVERSITY
Fall 2025

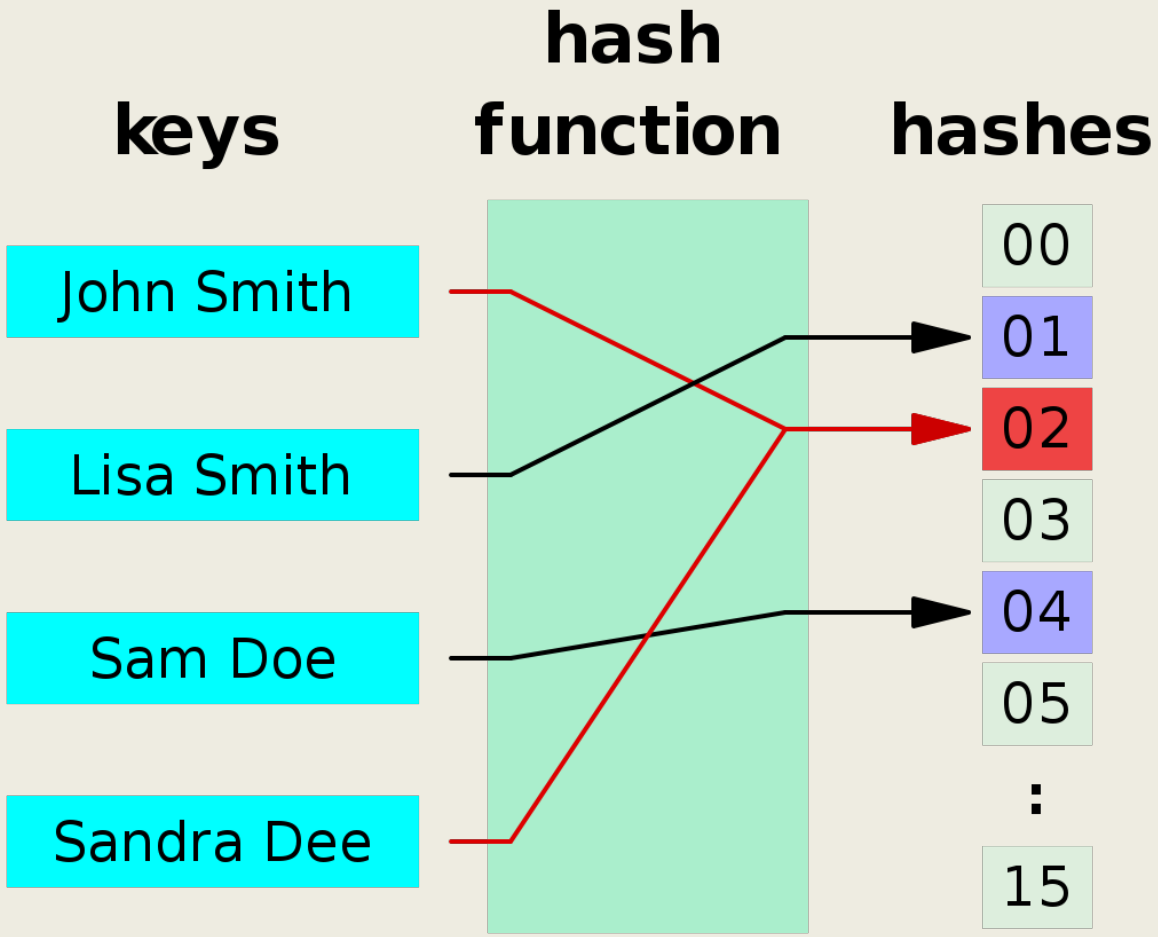
SUMMARY

- Performance
- Characters in a String
- Two Sum

HASH TABLES

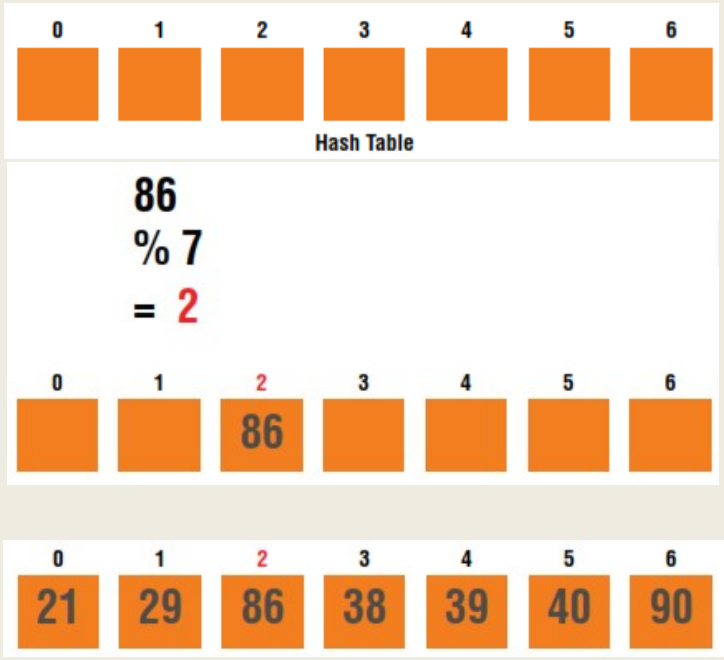
- An associative array is an abstract data type that stores key-value pairs with unique keys.
- A key-value pair consists of two pieces of data mapped together:
  - *The key is the piece of data you use to retrieve the value.*
  - *The value is the piece of data you use the key to retrieve.*
- A hash table is a linear data structure.
- You cannot store duplicate keys in a hash table, but values can be.
- A hash function is code that takes a key as input and outputs an integer you can use to map a hash table key to an array index to uses for storing the value.
- The index a hash function produces is called a hash value.

HASH TABLES


keys
hash function
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

HASH TABLES

- Hashing function of integer data types:
  - *Division hashing*
    - A standard technique is to use a modulo function on the key, by selecting a divisor M which is a prime number close to the table size, so h (K) = K mod M.



21 % 7 = 0

29 % 7 = 1

38 % 7 = 3

39 % 7 = 4

40 % 7 = 5

HASH TABLES

- Hashing function of integer data types:
  - *Mid-squares*
    - A mid-squares hash code is produced by squaring the input and extracting an appropriate number of middle digits or bits.
    - For example, if the input is 123,456,789 and the hash table size 10,000, squaring the key produces 1.524157875019e16, so the hash code is taken as the middle 4 digits of the 17-digit number (ignoring the high

$h(31) = 31^2 = 961$

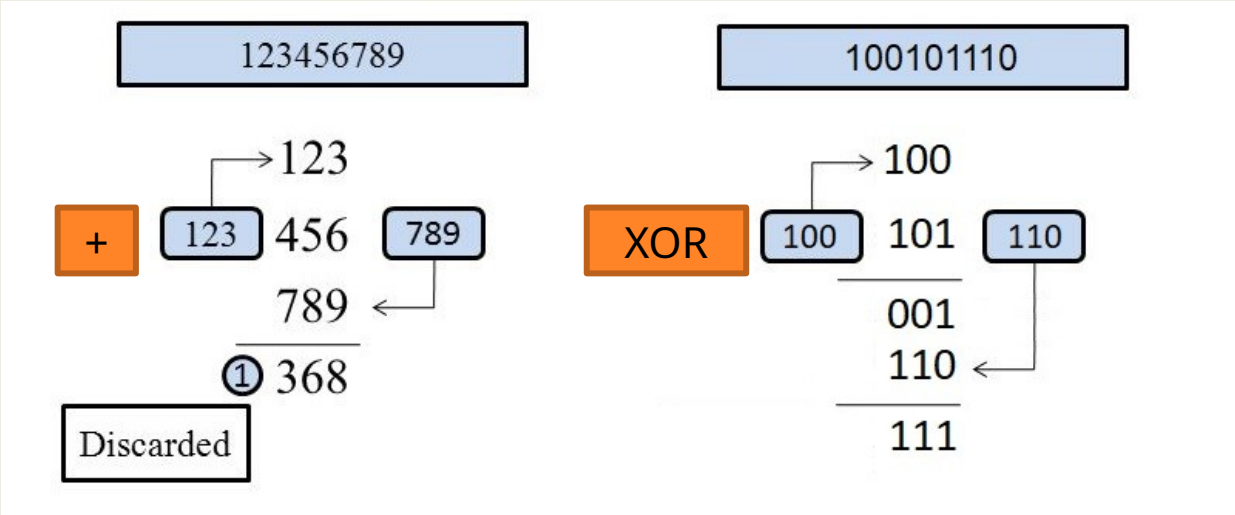
$h(15) = 15^2 = 255$

$h(22) = 22^2 = 484$

Index	Value
5	15
6	31
7	
8	22

HASH TABLES

- Hashing function of integer data types:
  - *Folding*
    - A folding hash code is produced by dividing the input into n sections of m bits, where 2^m is the table size, and using a parity-preserving bitwise operation like ADD or XOR, to combine the sections.



HASH TABLES

26
% 7
= 2

- Two numbers hashing to the same slot is a **collision**.
- Solution: Put it in the next empty slot, adding time complexity.
- your goal when creating a hash table is to use the correct number of slots and a hash function that produces the fewest collisions.

HASH TABLES

- Searching for data in a hash table is O(1)
- Inserting and deleting data in a hash table is O(1)
- Collisions can erode the efficiency of hash tables, making searching, insertion, and deletion O(n) in the worst- case scenario.

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)

HASH TABLES

- Challenge: count all the characters in a string.

```
def count_characters(s):
    freq = {}
    for ch in s:
        freq[ch] = freq.get(ch, 0) + 1
    return freq

# Example
print(count_characters("hello"))
```

```
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

This dictionary acts as a **hash table**, where:

- the **key** is a character
- the **value** is the number of times that character appears

HASH TABLES

- Challenge: two sum problem

return the two numbers' indexes in an unsorted list that adds up to a target value.

```
def two_sum(nums, target):
    seen = {}  # value → index

    for i, num in enumerate(nums):
        complement = target - num
        if complement in seen:
            return [seen[complement], i]
        seen[num] = i

nums = [-1, 2, 3, 4, 7]
target = 6
print(two_sum(nums, target))
```

```
[1, 3]
```

- Solution: using brute force, O(n^2)