# Unit 5 - Hashing

CS 201 - Data Structures II

Spring 2023

Habib University

Syeda Saleha Raza

# Let’s talk about dictionaries (Maps)…

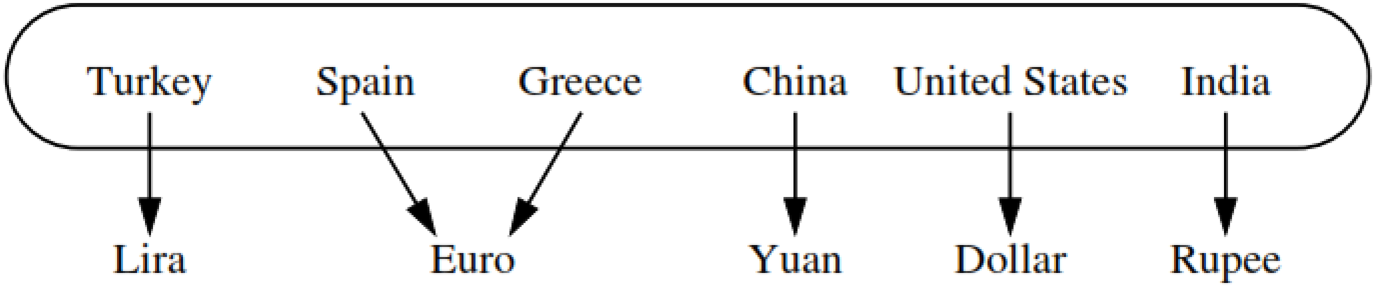
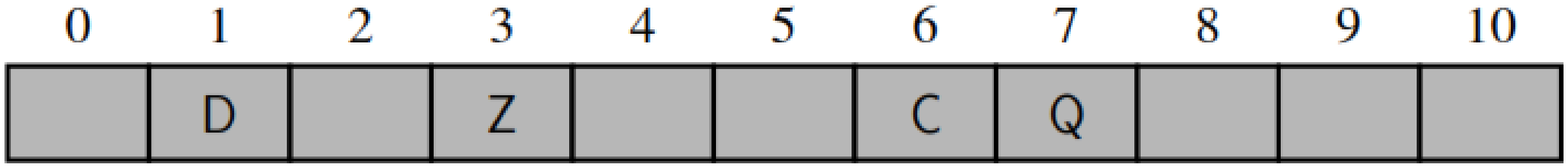


**Figure 10.1:** A map from countries (the keys) to their units of currency (the values).

# How are they implemented?<br>and<br>Why are they fast?

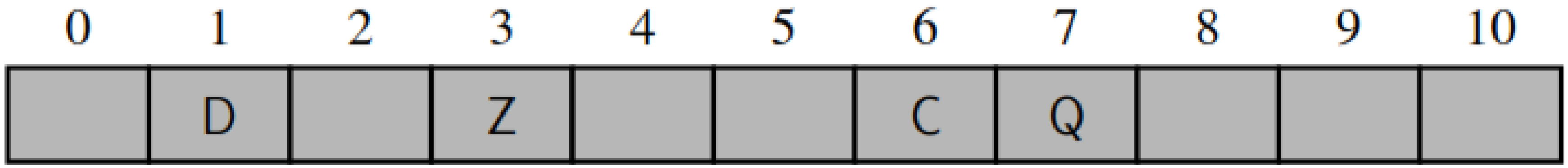
# Hash Table



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
|  | D |  | Z |  |  | C | Q |  |  |  |

**Figure 10.3:** A lookup table with length 11 for a map containing items (1,D), (3,Z), (6,C), and (7,Q).

# Hash Table



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
|  | D |  | Z |  |  | C | Q |  |  |  |

**Figure 10.3:** A lookup table with length 11 for a map containing items (1,D), (3,Z), (6,C), and (7,Q).

- What if
  - keys do not map to indices?
  - the largest key ‘N’ is grater than the number of elements ‘n’?
  - keys are not uniformly distributed?

# Hashing - Bucket Array

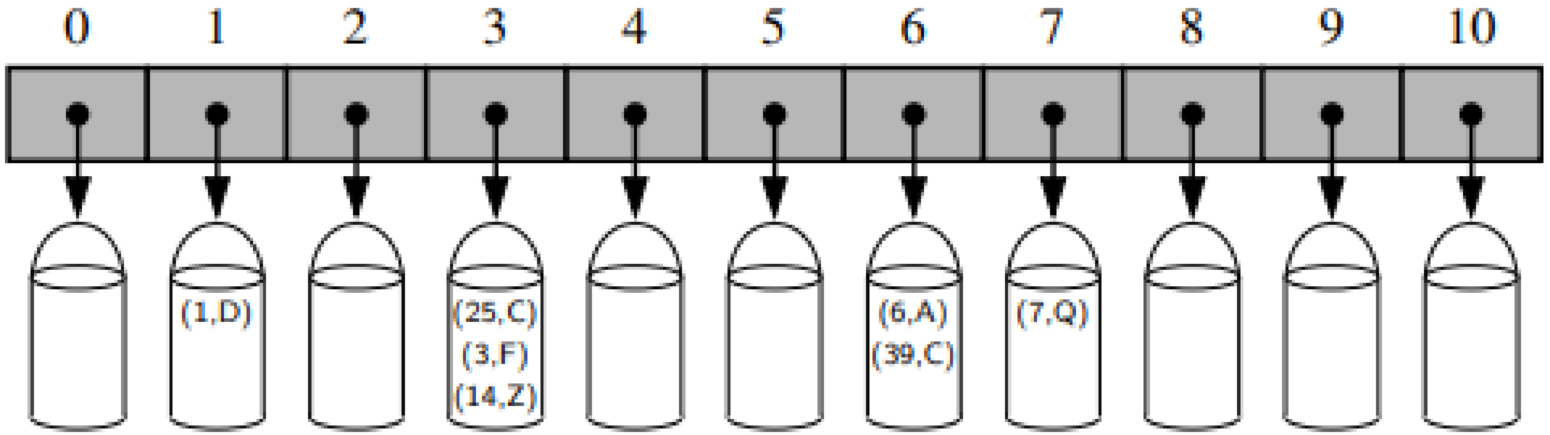


**Figure 10.4:** A bucket array of capacity 11 with items (1,D), (25,C), (3,F), (14,Z), (6,A), (39,C), and (7,Q), using a simple hash function.

# Chained Hashtable

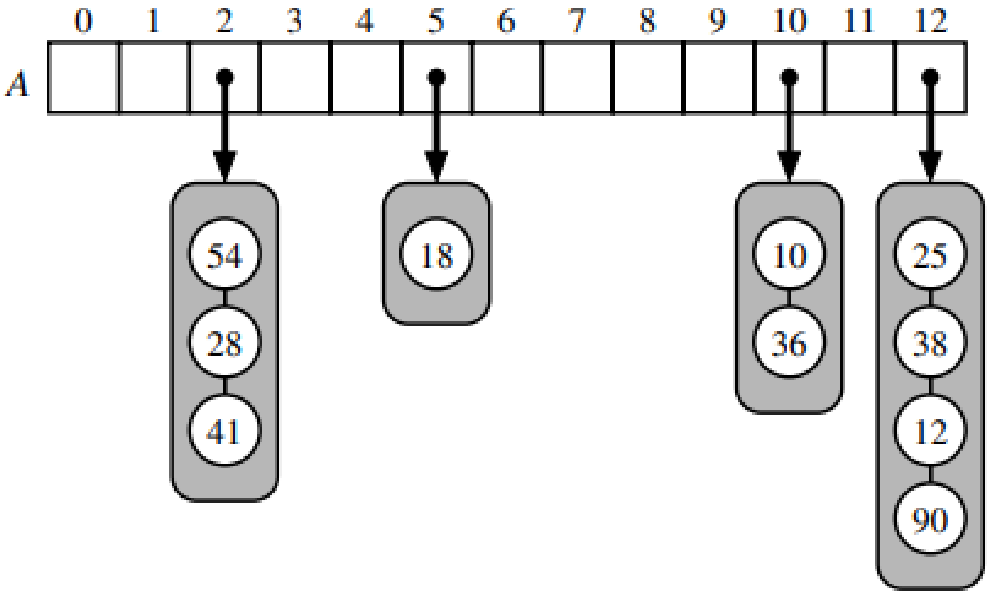


**Figure 10.6:** A hash table of size 13, storing 10 items with integer keys, with collisions resolved by separate chaining. The compression function is $h(k) = k \bmod 13$. For simplicity, we do not show the values associated with the keys.

# Chaining - Example

- x: 5 , 9 , 16, 4, 12, 7, 8 , 3
- Hash function: (x-2) % 5

# ChainedHashTable - Exercise

- X: 4, 20, 39, 17, 29, 34, 11,60, 45, 58, 48, 12,
- Hash function: (x-3) % 11

# Collision Handling – Open Addressing

- Linear Probing
- Quadratic Probing
- Rehashing

# Collision Handling - Linear Probing

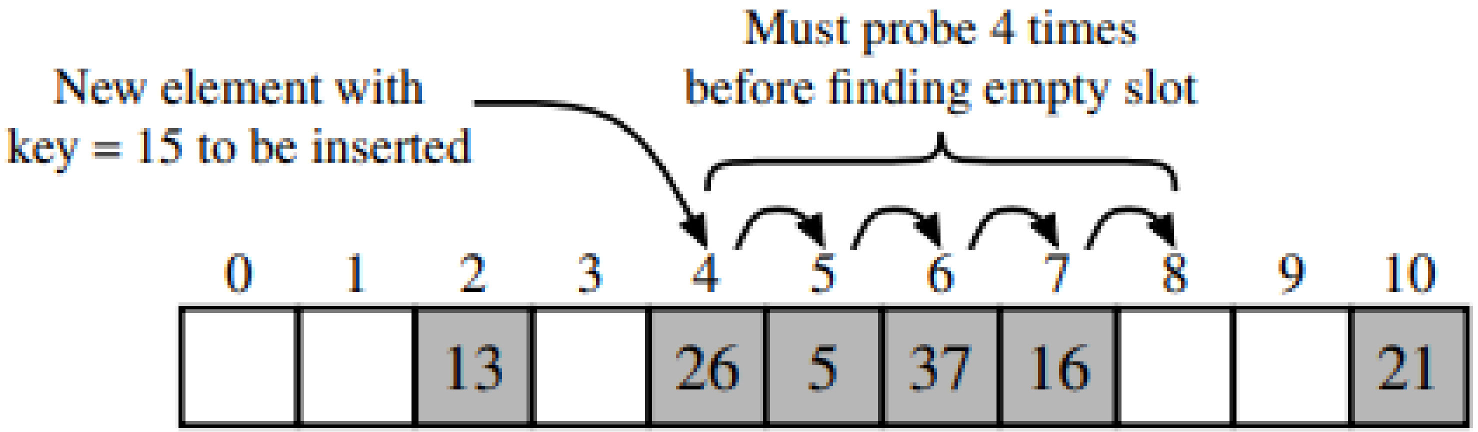


**Figure 10.7:** Insertion into a hash table with integer keys using linear probing. The hash function is $h(k) = k \bmod 11$. Values associated with keys are not shown.

# Linear Probing - Exercise

- 4, 20, 39, 17, 29, 34, 11,60, 45, 58, 48, 12,
- Hash function: (x-3) % 11

# Open addressing

- Linear probing
- Quadratic probing
- Double hashing

# Resizing a hashtable

# Desired properties of Hash Function

- Uniformly distributed (will use all of the range evenly)
- Low probability of collision (related partly to the previous)
- Computationally fast

# Hashing Exercise

- Numbers to be inserted in a hashtable of size 10:

{49, 64, 38, 79, 41, 44, 15, 34}

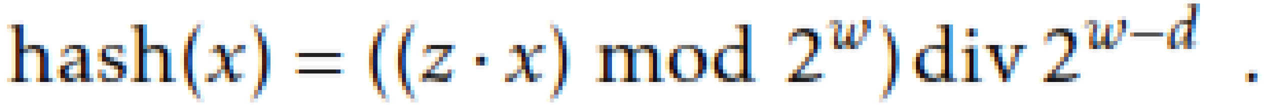
h(x) = x % 10

1. Using chaining
2. using linear probing
3. using quadratic probing
4. using double-hashing

   h`(x) = div(x,2)

5. Let’s resize this hashtable obtained in part (2)

# Hash function being fast



$$\mathrm{hash}(x) = ((z \cdot x) \bmod 2^{w}) \operatorname{div} 2^{w-d} \quad .$$

- Many hash functions work on table sizes in powers of 2.
- X % $2^r$ = X & ($2^r$ -1)- in binary representation

# Implementing USet interface

# When NOT to use Hash tables?

# Resources

- Open Data Structures (pseudocode edition), by Pat Morin. Available online at http://opendatastructures.org
- Data Structures and Algorithms in Python, by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. 2013. (1st. ed.). Wiley Publishing

# Thanks