# Hash Tables

## A Introduction to Hash Tables

Hash tables are powerful data structures that provide fast access to data using key-value pairs. They are widely used for implementing associative arrays, symbol tables, and sets. The main advantage of hash tables is their average-case constant time complexity $O(1)$ for insertion, deletion, and search operations.

- **Key-Value Mapping:** Each value is stored with a unique key.
- **Hash Function:** A function maps keys to indices in an array (the table).
- **Collision Handling:** When two keys hash to the same index, a collision occurs. Common strategies include chaining and open addressing.

## B Typical Implementation Strategies

- **Array of Buckets:** The hash table is an array where each slot (bucket) can store one or more key-value pairs.
- **Hash Functions:** Good hash functions distribute keys uniformly to minimize collisions. Examples: division method, multiplication method, universal hashing.
- **Collision Resolution:**
  - **Chaining:** Each bucket contains a linked list (or another structure) of entries that hash to the same index.
  - **Open Addressing:** All entries are stored in the array itself. On collision, probe for the next available slot (linear probing, quadratic probing, double hashing).
- **Load Factor:** The ratio of the number of entries to the number of buckets. High load factors increase collisions; resizing the table helps maintain efficiency.

## C Hash Table Operations

- **Insertion:** Compute the hash, resolve collisions, and store the key-value pair.
- **Search:** Compute the hash, resolve collisions, and find the value for a given key.
- **Deletion:** Locate the key and remove it, handling any necessary re-linking or marking in open addressing.

## D Typical Hash Functions

A hash function maps a key to an index in the hash table. Good hash functions distribute keys uniformly to minimize collisions.

- **Division Method:** $h(x) = x \bmod m$
- **Multiplication Method:** $h(x) = \lfloor m(xA \bmod 1) \rfloor$, where $0 < A < 1$
- **String Hashing:** For a string $s$, $h(s) = (s_0 a^{k-1} + s_1 a^{k-2} + ... + s_{k-1}) \bmod m$, with $a$ a constant (e.g., 31)

**Example:** Using the division method with $m = 5$:

- $h(21) = 21 \bmod 5 = 1$
- $h(31) = 31 \bmod 5 = 1$
- $h(12) = 12 \bmod 5 = 2$
- $h(43) = 43 \bmod 5 = 3$

## E Collision Management Strategies

When two keys hash to the same index, a **collision** occurs. There are two main strategies to handle collisions:

### Chaining (Separate Chaining)

Each bucket contains a linked list (or another structure) of entries. All keys that hash to the same index are stored in the list for that bucket.
**Example:** If we insert 21 and 31 into a table of size 5, both $h(21) = h(31) = 1$, so both are stored in the list at bucket 1.
**Advantages:**

- Simple to implement
- Table can store more elements than its size
- Deletion is easy (just remove from the list)

**Disadvantages:**

- Extra memory for pointers
- Performance degrades if many collisions (long lists)

### Open Addressing

All entries are stored in the array itself. On collision, probe for the next available slot.
**Common probing methods:**

- **Linear Probing:** Try slots $h(x), h(x) + 1, h(x) + 2, ...$ (modulo table size)
- **Quadratic Probing:** Try slots $h(x), h(x) + 1^2, h(x) + 2^2, ...$
- **Double Hashing:** Use a second hash function to determine the step size

**Example (Linear Probing):** Insert 21, 26, 31 into a table of size 5:

- $h(21) = 1$ (slot 1 empty, insert 21)
- $h(26) = 1$ (slot 1 full, try 2: empty, insert 26)
- $h(31) = 1$ (slots 1 and 2 full, try 3: empty, insert 31)

**Advantages:**

- No extra memory for pointers
- Good cache performance

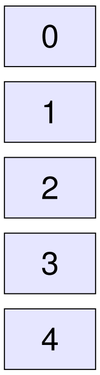
**Disadvantages:**

- Clustering: consecutive filled slots slow down operations
- Table cannot store more elements than its size
- Deletion is more complex (may require special markers)

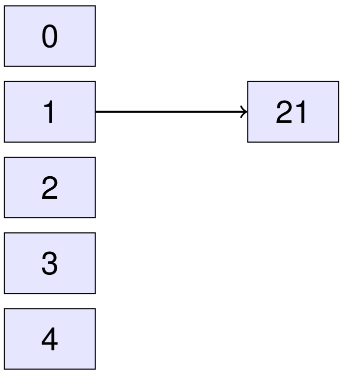
## F Step-by-Step Insertion Example (Chaining)

Suppose we insert 21, 31, 12, and 43 into a hash table with 5 buckets using $h(x) = x \bmod 5$ and **chaining** for collision resolution:

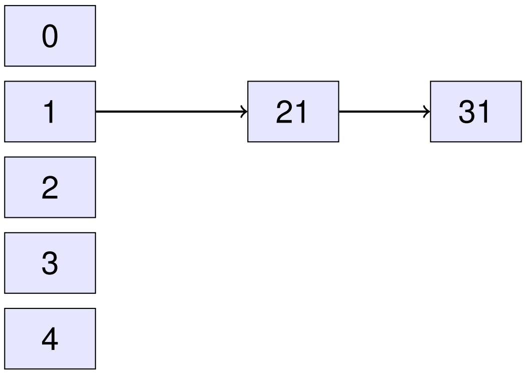
**Step 0: Empty hash table with 5 buckets.**



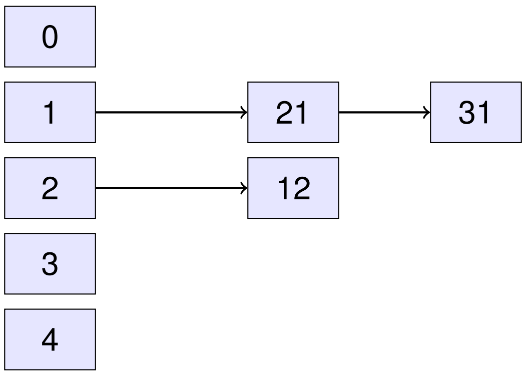
**Step 1: Insert 21.** $h(21) = 1$**; 21 goes in bucket 1.**



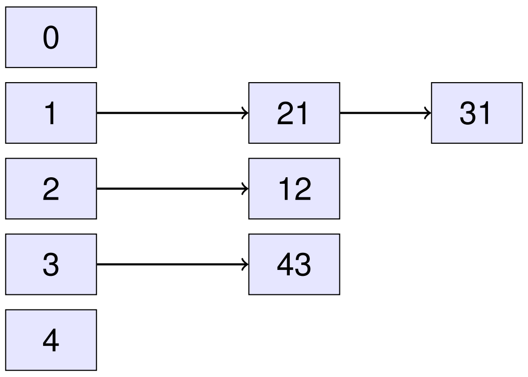
**Step 2: Insert 31.** $h(31) = 1$**; 31 chains after 21 in bucket 1.**



**Step 3: Insert 12.** $h(12) = 2$**; 12 goes in bucket 2.**



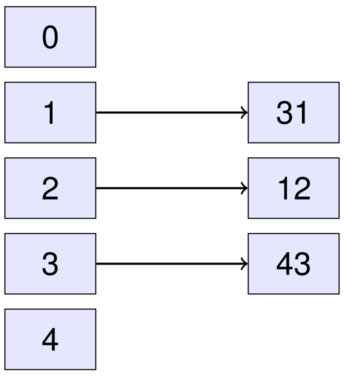
**Step 4: Insert 43.** $h(43) = 3$**; 43 goes in bucket 3.**



## G Step-by-Step Deletion Example (Chaining)

Suppose we now delete 21 from the table:

**After deleting 21: 31 becomes the first in bucket 1.**



## H Practice

**Practice Exercise:**

Suppose you have a hash table of size 7 and use the hash function $h(x) = x \bmod 7$. Insert the keys 10, 24, 15, and 31 using linear probing for collision resolution. Show the final state of the table and explain each step.