**1. Hash Functions**

**Definition:** A hash function is a function that takes input data (keys) and converts it into a fixed-size numerical value, typically an index in a hash table. The goal is to distribute keys uniformly across the table to minimize collisions.

**Example:** Suppose you have a hash table of size 10 and you want to hash the key 123.

* **Simple Hash Function (Modulus):**

Hash(123)=123mod  10=3\text{Hash}(123) = 123 \mod 10 = 3Hash(123)=123mod10=3

The key 123 is placed at index 3.

* **Another Hash Function:**

Hash(123)=(1×100+2×10+3)mod  10=6mod  10=6\text{Hash}(123) = (1 \times 100 + 2 \times 10 + 3) \mod 10 = 6 \mod 10 = 6Hash(123)=(1×100+2×10+3)mod10=6mod10=6

The key 123 is placed at index 6.

**2. Closed Addressing**

**Definition:** Closed addressing (also known as separate chaining) handles collisions by maintaining a list (like a linked list) of all elements that hash to the same index.

**Example:** Consider a hash table of size 5 and hash function Hash(key) = key % 5.

* Insert keys: 1, 6, 11
  + Hash(1) = 1, so index 1 → [1]
  + Hash(6) = 1, so index 1 → [1, 6]
  + Hash(11) = 1, so index 1 → [1, 6, 11]

All three keys are stored in a linked list at index 1.

**3. Separate Chaining**

**Definition:** Separate chaining is a collision resolution technique under closed addressing where each bucket of the hash table contains a separate data structure (like a linked list) to store multiple elements that hash to the same index.

**Example:** Hash table size: 7  
Hash function: Hash(key) = key % 7

* Insert keys: 10, 20, 15, 7, 3
  + Hash(10) = 3 → index 3 → [10]
  + Hash(20) = 6 → index 6 → [20]
  + Hash(15) = 1 → index 1 → [15]
  + Hash(7) = 0 → index 0 → [7]
  + Hash(3) = 3 → index 3 → [10, 3]

The hash table:

less

Copy code

Index 0: [7]

Index 1: [15]

Index 3: [10, 3]

Index 6: [20]

**4. Open Addressing**

**Definition:** Open addressing is a collision resolution method where, upon a collision, the algorithm probes the hash table to find another empty slot based on a probing sequence.

**a. Linear Probing**

**Definition:** Linear probing searches for the next available slot sequentially (i.e., index + 1, index + 2, ...) when a collision occurs.

**Example:** Hash table size: 5  
Hash function: Hash(key) = key % 5

* Insert keys: 0, 5, 10
  + Hash(0) = 0 → index 0 → [0]
  + Hash(5) = 0 (collision) → try index 1 → [0, 5]
  + Hash(10) = 0 (collision) → try index 1 (occupied) → try index 2 → [0, 5, 10]

Hash table:

yaml

Copy code

Index 0: 0

Index 1: 5

Index 2: 10

Index 3: Empty

Index 4: Empty

**b. Quadratic Probing**

**Definition:** Quadratic probing uses a quadratic function to calculate the next index to probe, reducing clustering compared to linear probing. The probe sequence is typically of the form:

Hash(key)+12,Hash(key)+22,Hash(key)+32,…\text{Hash}(key) + 1^2, \text{Hash}(key) + 2^2, \text{Hash}(key) + 3^2, \dotsHash(key)+12,Hash(key)+22,Hash(key)+32,…

**Example:** Hash table size: 7  
Hash function: Hash(key) = key % 7

* Insert keys: 10, 20, 15
  + Hash(10) = 3 → index 3 → [10]
  + Hash(20) = 6 → index 6 → [20]
  + Hash(15) = 1 → index 1 → [15]

Now, inserting 22:

* Hash(22) = 1 (collision) → try 1 + 1^2 = 2 → index 2 → [22]

Hash table:

yaml

Copy code

Index 0: Empty

Index 1: 15

Index 2: 22

Index 3: 10

Index 6: 20

**c. Double Hashing**

**Definition:** Double hashing uses a second hash function to determine the step size for probing, which helps in evenly distributing keys and minimizing clustering.

**Example:** Hash table size: 11  
Primary hash function: Hash1(key) = key % 11  
Secondary hash function: Hash2(key) = 7 - (key % 7)

* Insert keys: 27, 18, 29, 28, 39

1. **Insert 27:**
   * Hash1(27) = 5
   * Place at index 5.
2. **Insert 18:**
   * Hash1(18) = 7
   * Place at index 7.
3. **Insert 29:**
   * Hash1(29) = 7 (collision with 18)
   * Hash2(29) = 7 - (29 % 7) = 7 - 1 = 6
   * Next probe: 7 + 6 = 13 → 13 % 11 = 2
   * Place at index 2.
4. **Insert 28:**
   * Hash1(28) = 6
   * Place at index 6.
5. **Insert 39:**
   * Hash1(39) = 6 (collision with 28)
   * Hash2(39) = 7 - (39 % 7) = 7 - 4 = 3
   * Next probe: 6 + 3 = 9
   * Place at index 9.

Hash table:

yaml

Copy code

Index 2: 29

Index 5: 27

Index 6: 28

Index 7: 18

Index 9: 39

**5. Rehashing**

**Definition:** Rehashing is the process of resizing the hash table and applying a new hash function when the load factor (ratio of number of elements to table size) exceeds a certain threshold. This helps in maintaining efficient operations by reducing collisions.

**Example:** Initial hash table size: 5  
Load factor threshold: 0.6

* Current elements: 3
* Load factor: 35=0.6\frac{3}{5} = 0.653​=0.6 → At threshold, trigger rehashing.

**Rehashing Steps:**

1. **Resize the table:** Double the size to 10.
2. **Choose a new hash function:** For simplicity, use Hash(key) = key % 10.
3. **Reinsert existing elements:** Insert the 3 elements into the new table based on the new hash function.

After rehashing, collisions are likely reduced due to the larger table size.

**Summary**

* **Hash Functions** convert keys to indices.
* **Closed Addressing** handles collisions by storing multiple elements in the same bucket.
* **Separate Chaining** is a type of closed addressing using linked lists.
* **Open Addressing** resolves collisions by finding another empty slot using probing:
  + **Linear Probing** searches sequentially.
  + **Quadratic Probing** uses a quadratic formula for probing.
  + **Double Hashing** uses a second hash function for probing.
* **Rehashing** resizes the table and rehashes all keys to maintain efficiency.