CHAPTER 3
DATA STRUCTURES: HASHING

3.2 Hashing

- **Basic Principle:**
  - Derive a **number** from a given element, and use that **number** to store & access the given element.
  - Here, the given **element** is referred to as **Key**, and the derived **number** is referred to as **Index.**
- **Definition:**
  - "Hashing is a search technique which **directly** finds out the **location** of a given data **element** in a **constant** search time."

3.2.1 Hash Function

- **Definition:**
  - "Hashing uses some **function** to find out the **location** of a given data **element** using that element. Such functions are referred to as Hash Functions."
- **Various Hash Functions:**
  - Some of the known Hash Functions are given below:

1. **Division Method:**
   - Divide key by size, and use the remainder as an Index.
   - **Index = Key % Size** where, Key = given data element, Size = total capacity of storage, Index = location of given data element.
   - **Example:** Suppose we have to store integer value '15' in an array of size '10'. Then, we can use –

     Index = 15 % 10 = 5

2. **Mid-square Method:**
   - Take the square of key, and use the middle digits of square value as an Index.
   - **Example:** Suppose, key = 15 → $key^2 = 15^2 = 225$ → **2**
3. **Folding Method:**
   - Consider that the key is in binary form.
   - Take EX-OR operation among lower & upper bits.
   - **Example:** Suppose, key = 39 → 100111 → 100 ^ 111 → **011** → **3**

4. **Multiplicative Method:**
   - Multiply key with some constant value which is between 0 and 1
   - Extract fractional part
   - Multiply it by size
   - Use integer value of this multiplication as an index
   - index = floor (Size * (C * Key % 1))

     Where c is a constant value between 0 and 1
   - Example: Consider Key = 15, Size = 10, C = 0.75

     Index = floor (Size * (C * Key % 1))
     = floor (10 * (0.75 * 15 % 1))
     = floor (10 * (11.25 % 1))
     = floor (10 * (0.25))
     = floor (2.5)
     = 2

3.2.2 Hash Collision

- **Definition:**
  - "A **situation** in which –
    - **Two** different '**key**' values produce the **same** index, and
    - Tries to **occupy** the same location."
- **Collision Resolution Techniques:**

  A. Open Addressing

  1. **Linear Probing:**
     - Store colliding element into the next available space.
     - **index = (key + n) % size**

       Where n is no. of times collision occurs.
     - Example: Collision between 15 and 25

       Index = 15 % 10 = 5

       Index = 25 % 10 = 5 # Collision at index 5

       Index = (25 + 1) % 10 = 6

       Index = (25 + 2) % 10 = 7

       Index = (25 + 3) % 10 = 8 and so on until empty location is found
     - **Disadvantage:** Tends to clustering.

2. **Quadratic Probing**
   - Avoids the problem of Linear Probing of Clustering.
   - **index = (key + n^2) % size**

     Where n is no. of times collision occurs.
   - Example: Collision between 15 and 25

     Index = 15 % 10 = 5

     Index = 25 % 10 = 5 # Collision at index 5
     Index = (25 + 1^2) % 10 = 6
     Index = (25 + 2^2) % 10 = 9
     Index = (25 + 3^2) % 10 = 4 and so on until empty location is found

3. **Re-hashing / Double Hashing**
   - Use colliding index as key, and Generate a new index.
   - Example: Collision between 15 and 25

     Index1 = 15 % 10 = 5

     Index1 = 25 % 10 = 5 # Collision at index 5

     Offset = Prime – (Key2 % Prime) # Key2 = Colliding Index
     = 7 – (5%7) = 2 # 7 is prime number < Size

     Index2 = (Index1 + (n * Offset)) % Size # Collision number

     → (5 + (1 * 2)) % 10 = 7
     → (5 + (2 * 2)) % 10 = 9 and so on until empty location is found

4. **Random Probing**
   - Select any other index randomly until the free location is found.
   - Example: Collision between 15 and 25

     Index = 15 % 10 = 5

     Index = 25 % 10 = 5 # Collision at index 5

     Index = RandomNumber % Size

     Index = 7 % 10 = 7

B. Chaining

- Use an array of **Singly Linked List.**
- Use an **index** to point to a particular SLL from an array.
- Then, **search** in that particular SLL.

● ● ●