Marwadi University

Marwadi University

01CE1301 – Data Structure

Unit - 5
Hashing & Collision

Prof. Chirag Bhalodia
Department of Computer Engineering

Outline

- Hashing Concepts and methods
- Hash Table Methods
- Introduction of Hash Functions
- Collision in Hashing
- Collision-Resolution Techniques

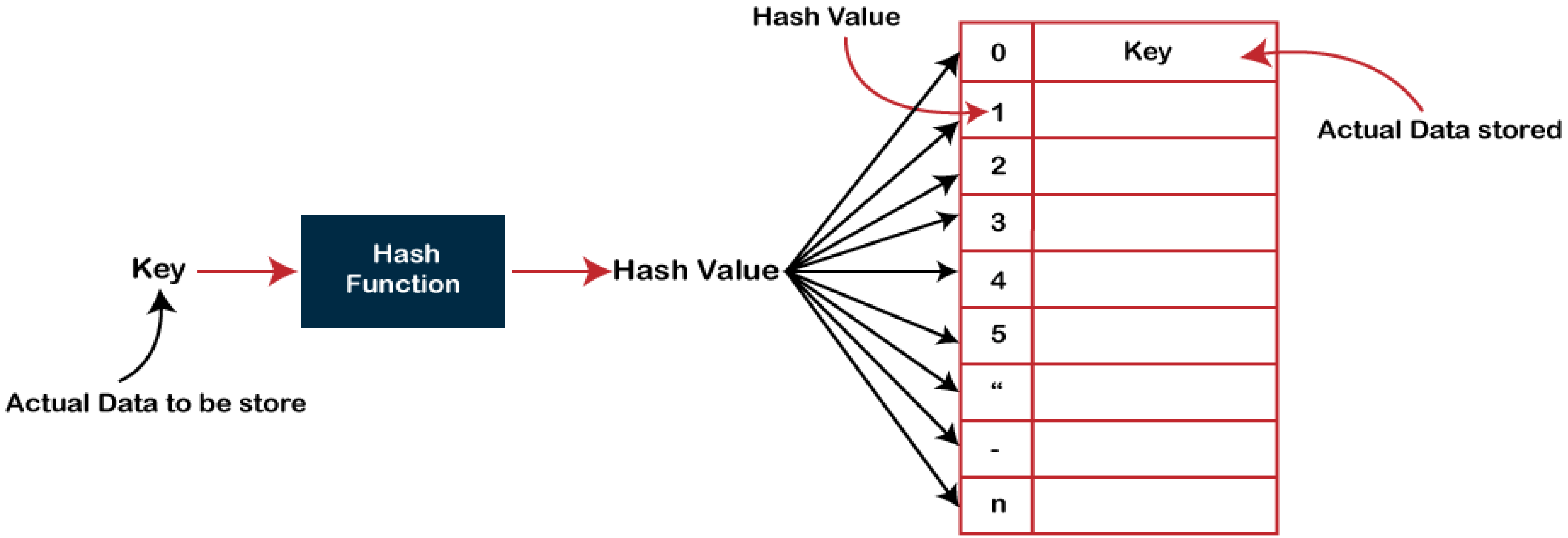
Hashing Concepts & Methods

Hashing Concepts & Methods

- **Hashing:** Hashing is an important data structure designed to ***solve the problem of efficiently finding and storing data in an array.***
- **For example**, if you have a list of 20000 numbers, and you have given a number to search in that list- you will scan each number in the list until you find a match.
- **Hash Function:** It is a technique of mapping a ***large chunk of data into small tables*** using a ***hashing function***.
- Hash function is also known as the ***message digest*** function.
- **Hash Table:** ***Hash table*** is one of the most important data structures that uses a ***special function*** known as a ***hash function*** that maps a given value with a key to access the elements faster.
- A Hash table is a data structure that stores some information, and the information has basically two main components, i.e., ***key*** and ***value***. The hash table can be implemented with the help of an ***associative array***.

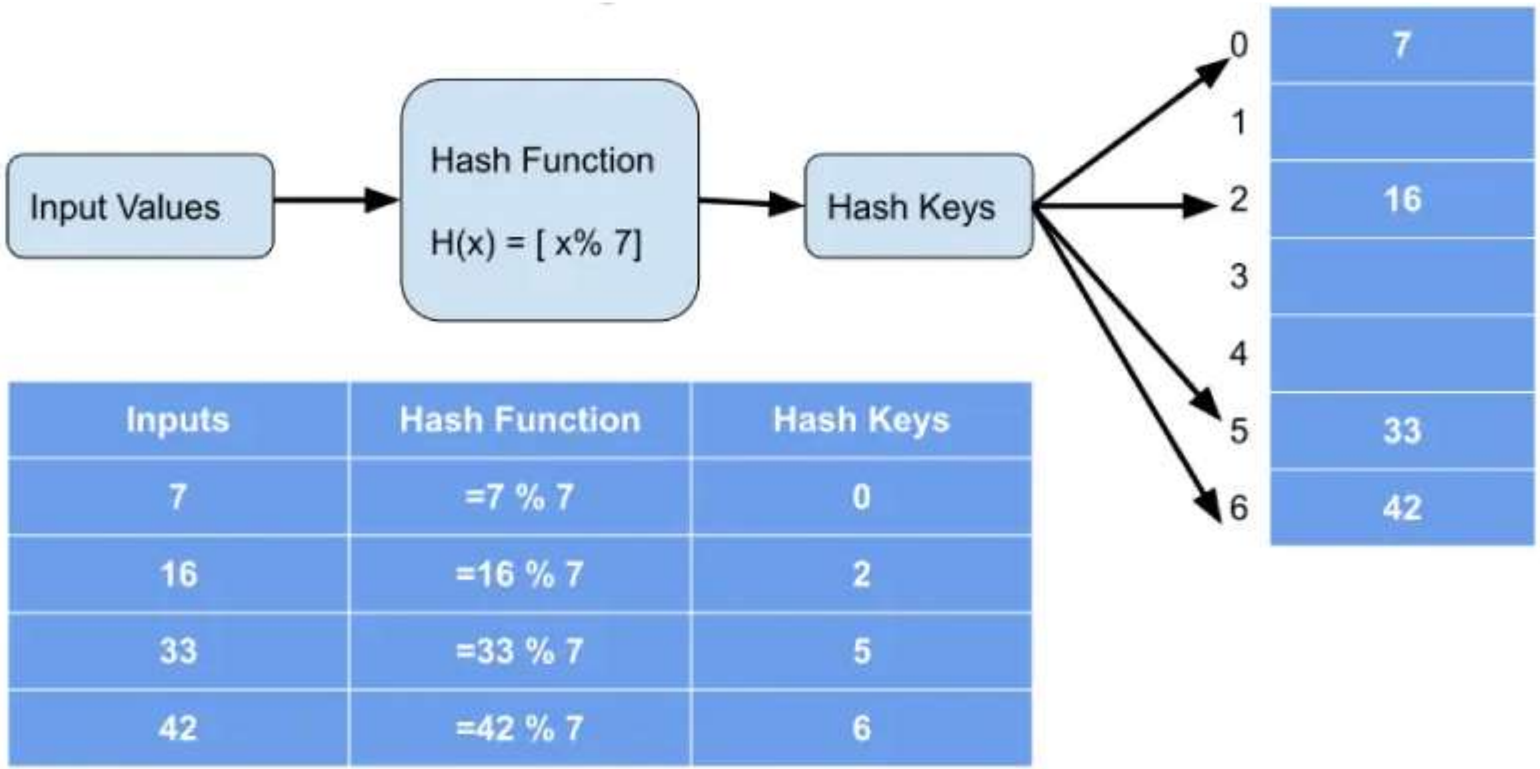
Hashing Concepts & Methods

- In Hashing technique, the ***hash table*** and ***hash function*** are used. Using the hash function, we can ***calculate the address*** at which the value can be stored in hash table.
- **Key:** Key is the raw data that has to be hashed in a hash table.
- *Hash Key = Key Value % Number of Slots in the Table*
- The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as:



Hashing Concepts & Methods

- **Example 1:**
- The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as:



0	7
1	
2	16
3	
4	
5	33
6	42

Inputs	Hash Function	Hash Keys
7	=7 % 7	0
16	=16 % 7	2
33	=33 % 7	5
42	=42 % 7	6

Hashing Concepts & Methods

- **Example 2:**

Key	Value
Italy	Rome
France	Paris
England	London
Australia	Canberra
Switzerland	Berne

Position (hash = key length)	Key	Value
1		
2		
3		
4		
5	Italy	Rome
6	France	Paris
7	England	London
8		
9	Australia	Canberra
10		
11	Switzerland	Berne

Hashing Concepts & Methods

- **Example 3: Find the array index of given data. Table size is 20.**

(Key, Value)
(1,20) (2,70) (42,80) (4,25) (12,44) (14,32) (17,11) (13,78) (37,98)

Sr. No.	Key	Hash	Array Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14
7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

Hash Table Methods

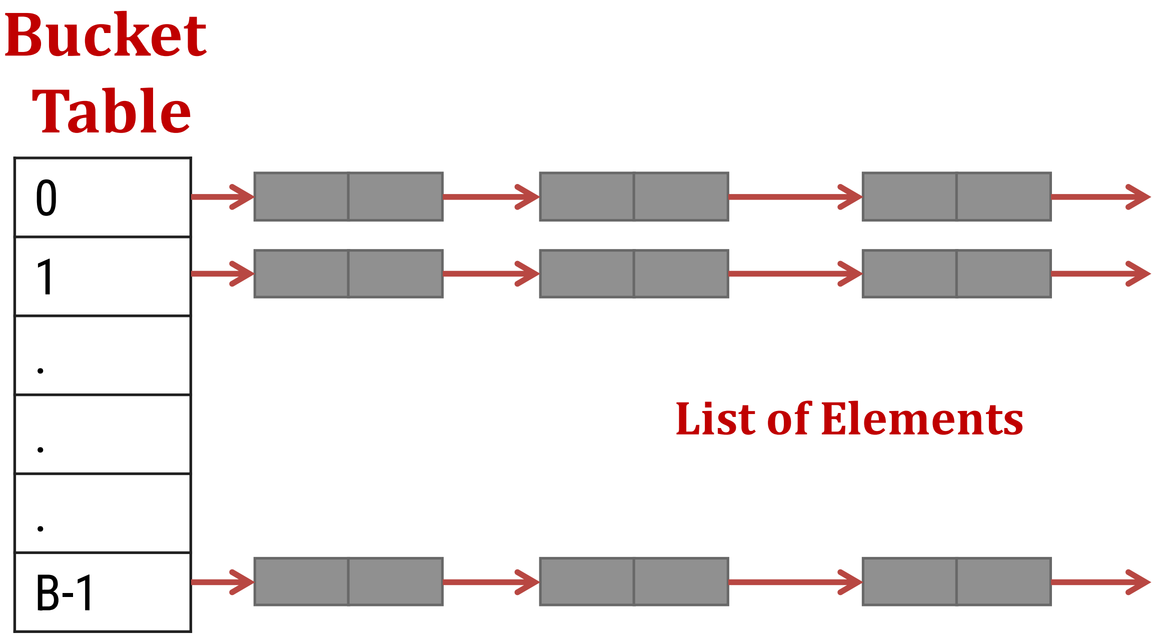
Hash Table Methods

- There are two different forms of hashing.

1. **Open hashing or external hashing**
   - Open or external hashing, allows records to be stored in unlimited space (could be a hard disk).
   - It places no limitation on the size of the tables.
2. **Close hashing or internal hashing**
   - Closed or internal hashing, uses a fixed space for storage and thus limits the size of hash table.

Open Hashing Data Structure

- The basic idea is that the **records [elements]** are **partitioned** into **B classes**, numbered 0,1,2 … B-1.
- A Hashing function **f(x)** maps a record with **key x** to an integer value between **0 and B-1.**
- Each **bucket** in the **bucket table** is the **head** of the **linked list** of records mapped to that bucket.



The open hashing data organization

Close Hashing Data Structure

- A closed hash table **keeps the elements in the bucket** itself.
- Only **one element can be put** in the bucket.
- If we **try to place an element** in the bucket and find **it already holds** an element, then we say that a **collision** has **occurred**.
- In **case of collision**, the element should be **rehashed** to alternate empty location within the bucket table.
- In closed hashing, collision handling is a very important issue.

0	A
1	
2	C
3	
4	
5	B

Introduction of hash Functions

Introduction of Hash Functions

- **Hash Function:** It is a technique of mapping a ***large chunk of data into small tables*** using a ***hashing function***.
- Hash function is also known as the ***message digest*** function.

- **Characteristics of a Good Hash Function**
  - A good hash function avoids collisions.
  - A good hash function tends to spread keys evenly in the array.
  - A good hash function is easy to compute.

Introduction of Hash Functions

Different hashing functions

1. Division-Method
2. Midsquare Method
3. Folding Method
4. Digit Analysis Method

Division Method

- In this method we use **modular arithmetic system** to **divide** the **key value** by **some integer** divisor **m** (may be table size).
- It gives us the location value, where the element can be placed.
- We can write, **L = (K mod m)**,
  - **L** = location in table/file
  - **K** = key value
  - **m** = table size/number of slots in file
- **Example:** Suppose, **k = 23, m = 10** then
  - L = (23 mod 10) = 3 = 3
  - The key whose **value is 23** is placed in **3rd location**.

Midsquare Method

- In this case, we **square the value of a key** and take the **number of digits required** to form an address, from the **middle position** of squared value.
- Suppose a **key** value is **16**
  - Its **square is 256**
  - Now if we want **address of two digits**
  - We select the address as **56** (i.e. two digits starting from middle of 256)
- **Example:** Suppose a **key** value is **60.** Suppose the hash table has **100** memory locations. So **r=2** because two digits are required to map the key to the memory location.
  - k = 60
  - k x k = 60 x 60 = 3600
  - h(60) = 60
  - The hash value obtained is 60.

Folding Method

- **This method involves two steps:**
- **Step-1:** Divide the key-value k into a number of parts i.e. k1, k2, k3,….,kn, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.
- **Step-2:** Add the individual parts. The hash value is obtained by ignoring the last carry if any.
- **Formula:**

  k = k1, k2, k3, k4, ….., kn

  s = k1+ k2 + k3 + k4 +….+ kn

  h(K)= s

  Here, **s** is obtained by adding the parts of the key **k.**

Folding Method

- **Example:** Key = 12345678. Perform folding method of hash function. (i.e., Digit = 2)
- Here **actual values** of **each parts** of key are **added**
  - Suppose, the **key** is : **12345678**, and the required address is of two digits,
  - Break the key into: **12, 34, 56, 78**
  - Add these, we get 12 + 34 + 56 + 78 : **180**, ignore first “1” we get **80 as location**

Digital Analysis Method

- Here we make a **statistical analysis** of **digits** of the **key**, and **select** those **digits** (of fixed position) which **occur** quite **frequently**
- Then reverse or **shifts the digits** to get the **address**
- For example,
  - The key is : **9861234**
  - If the statistical analysis has revealed the fact that the **third** and **fifth** position digits occur quite frequently,
  - We **choose** the **digits** in **these positions** from the key
  - So we get, **62**. **Reversing** it we get **26 as the address**

Collision in Hashing

Collision in Hashing

- Collision resolution is the main problem in hashing.
- A **collision** occurs when **more than one value to be hashed** by a particular hash function, **hash to the same slot in the table** or data structure (hash table) being generated by the hash function.
- A hash collision or hash clash is when two pieces of data in a hash table share the same hash value.
- Hashing collisions can have negative impacts on the performance, security, and integrity of the data and the system.

Collision Resolution Techniques

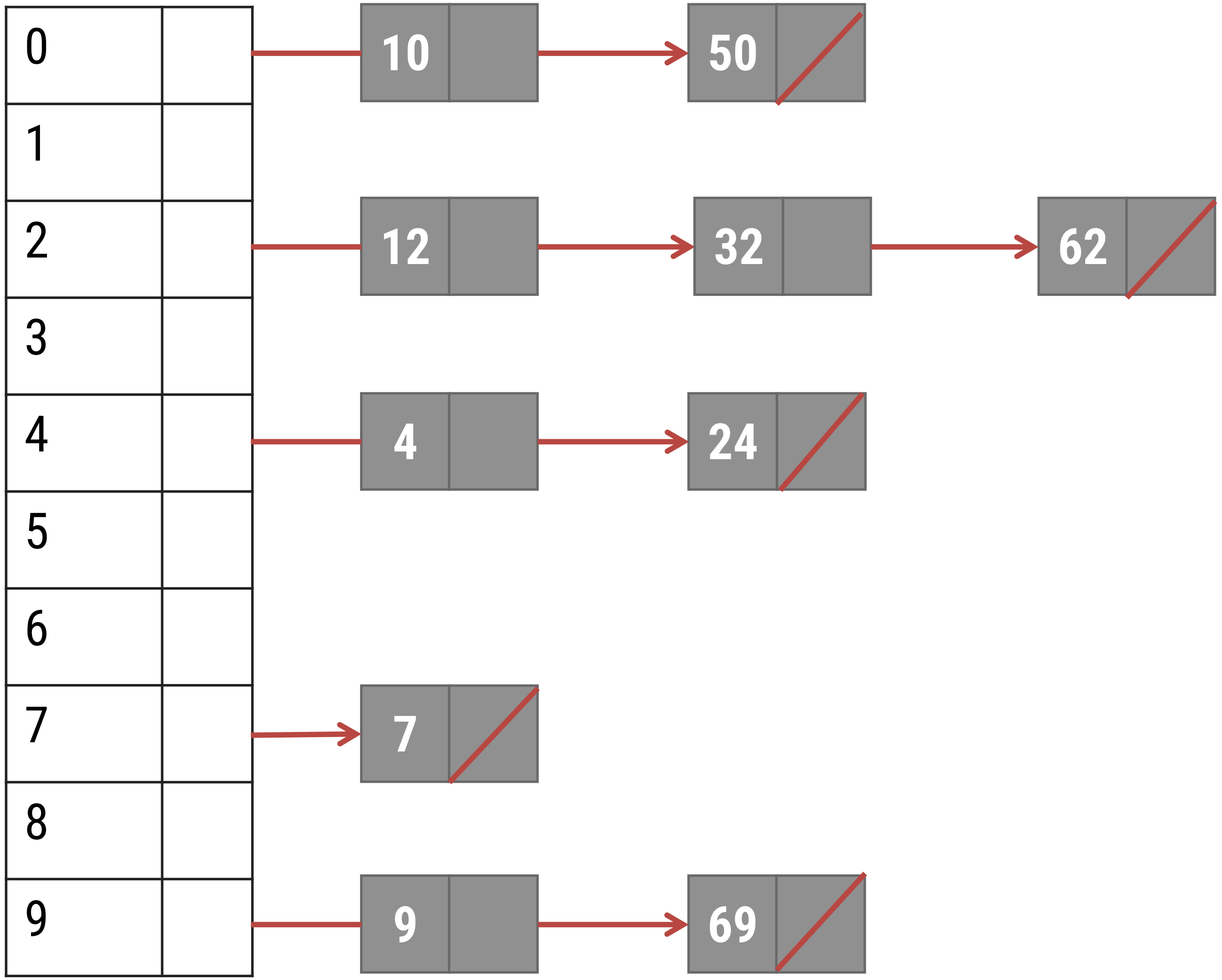
Collision Resolution Techniques

- Collision resolution is the main problem in hashing.
- If the element to be inserted is mapped to the same location, where an element is already inserted then we have a **collision** and it must be resolved.
- There are several strategies for collision resolution. The most commonly used are :
  - **Separate chaining** - used with open hashing
  - **Open addressing** - used with closed hashing

Separate Chaining Method

- In this strategy, a **separate list** of all elements mapped to the same value is maintained.
- Separate chaining is based on **collision avoidance**.
- If memory space is tight, separate chaining should be avoided.
- Additional memory space for links is wasted in storing address of linked elements.
- **Hashing function** should **ensure even distribution** of elements among buckets; otherwise the **timing behaviour** of most operations on hash table **will deteriorate**.

Separate Chaining Method



A Separate Chaining

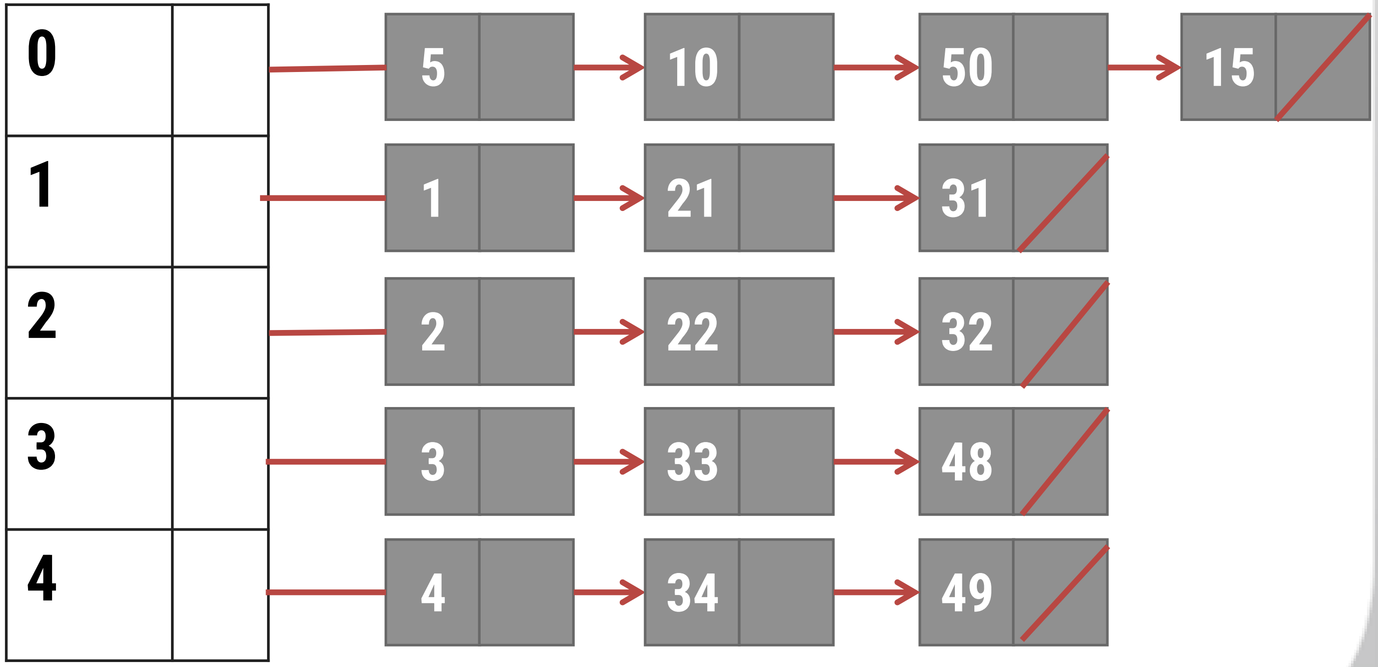
Hash Table

Example - Separate chaining

Example : The integers given below are to be **inserted** in a **hash table** with **5 locations** using chaining to resolve collisions. Construct hash table and use simplest hash function.
1, 2, 3, 4, 5, 10, 21, 22, 33, 34, 15, 32, 31, 48, 49, 50

An **element** can be **mapped** to a location in the hash table using the mapping **function key % 10**

Hash Table Location	Mapped elements
0	5, 10, 15, 50
1	1 , 21, 31
2	2, 22, 32
3	3 , 33, 48
4	4 , 34, 49



Hash Table

Open Addressing Method

- Separate chaining requires additional memory space for pointers.
- Open addressing hashing is an alternate method of handling collision.
- In **open addressing**, if a **collision** occurs, **alternate cells are tried** until an empty cell is found.
  a. Linear probing
  b. Quadratic probing
  c. Double hashing.

Linear Probing

- In **linear probing**, whenever there is a **collision**, **cells are searched sequentially** (with wraparound) **for an empty cell**.
- Fig. shows the result of inserting keys **{5,18,55,78,35,15}** using the hash function (f(key)= **key%10**) and linear probing strategy.

	Empty Table	After 5	After 18	After 55	After 78	After 35	After 15
0							15
1							
2							
3							
4							
5		5	5	5	5	5	5
6				55	55	55	55
7						35	35
8			18	18	18	18	18
9					78	78	78

Linear Probing

- Linear probing **is easy to implement** but it suffers from "**primary clustering**"
- When many **keys** are **mapped** to the **same location** (clustering), linear probing **will not distribute** these keys **evenly** in the hash table.
- These **keys** will be **stored** in **neighbourhood** of the location where they are mapped.
- This will **lead to clustering** of keys around the point of collision

Quadratic probing

- One way of **reducing** "**primary clustering**" is to use quadratic probing to resolve collision.
- Suppose the "**key**" is mapped to the location **j** and the cell **j** is already **occupied**.
- In quadratic probing, the **location j, (j+1), (j+4), (j+9),** ... are examined to find the first empty cell where the key is to be inserted.
- This table **reduces primary clustering**.
- It **does not ensure** that all cells in the table will be examined to **find an empty cell**.
- Thus, it may be **possible** that **key** will **not be inserted** even **if there is an empty cell** in the table.

Double Hashing

- This method requires **two hashing functions** f1 (key) and f2 (key).
- Problem of **clustering** can **easily** be **handled** through double hashing.
- Function **f1 (key)** is known as **primary hash function**.
- In case the address obtained by f1 (key) is already occupied by a key, the function f2 (key) is evaluated.
- The second function **f2 (key) is used** to **compute** the **increment** to be added to the address obtained by the first hash function f1 (key) in case of collision.
- The search for an empty location is made successively at the addresses
  - f1(key) + f2(key),
  - f1(key) + 2 * f2(key),
  - f1 (key) + 3 * f2(key),...

Thank You