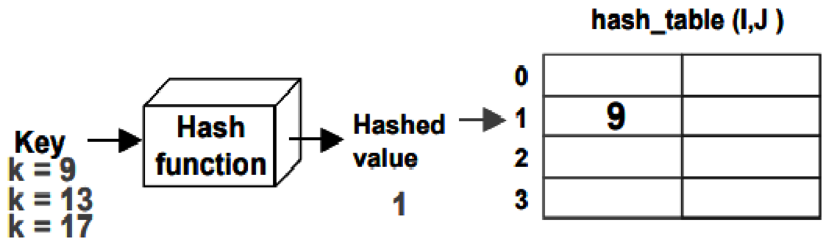
Hash Tables Data Structure

- ❖ Hash tables are used to implement map and set data structures in most common programming languages and is an **unordered** collection of **key-value** pairs, where each key is **unique**.
- ❖ Hashing is an important Data Structure which is designed to use a special function called the Hash function.
- ❖ Hash function is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.



Hashing Performance There is three factors the influence the performance of hashing:

Hash function

- ➢ should distribute the keys and entries evenly throughout the entire table
- ➢ should minimize collisions

Collision resolution strategy

- ➢ Open Addressing(Liner probing): store the key/entry in a different position
- ➢ Separate Chaining: chain several keys/entries in the same position

Table size

- ➢ Too large a table, will cause a wastage of memory
- ➢ Too small a table will cause increased collisions and eventually force rehashing (creating a new hash table of larger size and copying the contents of the current hash table into it)
- ➢ The size should be appropriate to the hash function used and should typically be a prime number. Why? Discussed this.

❖ Selecting Hash Functions

The hash function converts the key into the table position. It can be carried out using three techniques:

1-Modular Arithmetic: Compute the index by dividing the key with some value and use the remainder as the index. This forms the basis of the next two techniques.

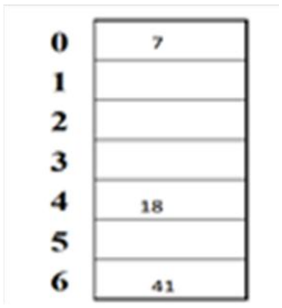
Example 1: index: = key MOD table size

Insert: 7, 18, 41,… size table is 7

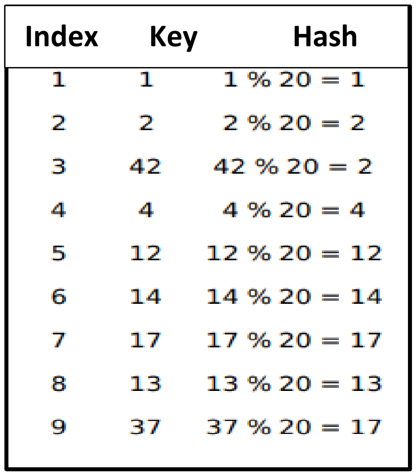
7%7=0 $\xRightarrow{\textit{Bucket no } 0}$

18%7=4

41%7=6



Example 2: Insert 1, 2, 42, 4, 12, 14, 17, 13, 37…, table size is 20



Index	Key	Hash
1	1	1 % 20 = 1
2	2	2 % 20 = 2
3	42	42 % 20 = 2
4	4	4 % 20 = 4
5	12	12 % 20 = 12
6	14	14 % 20 = 14
7	17	17 % 20 = 17
8	13	13 % 20 = 13
9	37	37 % 20 = 17

2-mid-square method. We first square the item, and then extract some portion (middle) of the resulting digits. For example, if the item were 44, we would first compute 44^2=1,936. By extracting the middle two digits, 93.

Item	mid-square
17	8
77	4
31	6
4	6

3- folding method for constructing hash functions begins by dividing the item into equal-size pieces (the last piece may not be of equal size). These pieces are then added together to give the resulting hash value. For example, if our item was the phone number 436-555-4601, we would take the digits and divide them into groups of 2 (43,65,55,46,01). After the addition, 43+65+55+46+0143+65+55+46+01, we get 210. If we assume our hash table has 11 slots, then we need to perform the extra step of dividing by 11 and keeping the remainder. In this case 210 % 11210 % 11 is 1, so the phone number 436-555-4601 hashes to slot 1.

❖ **Collision :**

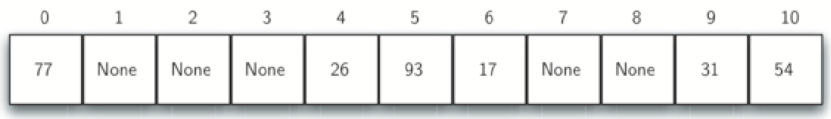
When two items hash to the same slot (position), we must have a systematic method for placing the second item in the hash table. This process is called **collision resolution.**

Two methods to resolve collisions:

1. **Open addressing (linear probing)**: is one method for resolving collisions looks into the hash table and tries to find another open slot to hold the item that caused the collision. A simple way to do this is to start at the original hash value position and then move in a sequential manner through the slots until we encounter the first slot that is empty. Note that we may need to go back to the first slot (circularly) to cover the entire hash table. By systematically visiting each slot one at a time, we are performing an open addressing technique called **linear probing.**

Table 1: shows an extended set of integer items under the simple remainder(Modular Arithmetic) method hash function (54,26,93,17,77,31,44,55,20).

Table 1:

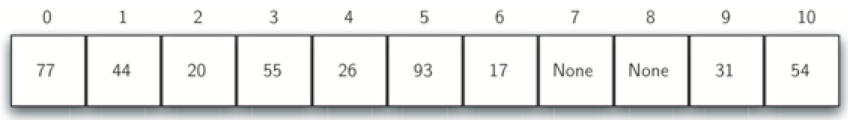


0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

When we attempt to place 44 into slot 0, a collision occurs. Under linear probing, we look sequentially, slot by slot, until we find an open position. In this case, we find slot 1.

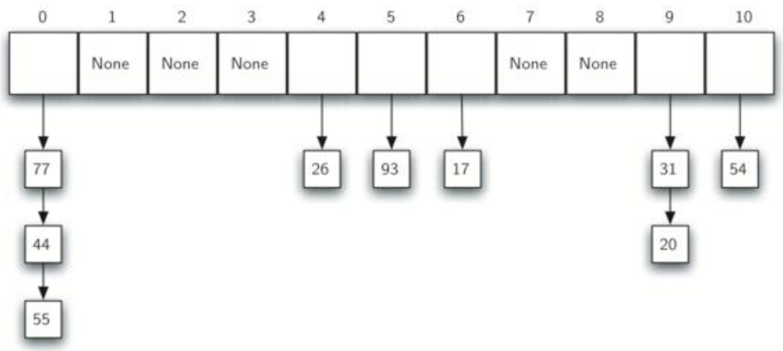
Again, 55 should go in slot 0 but must be placed in slot 2 since it is the next open position. The final value of 20 hashes to slot 9. Since slot 9 is full, we begin to do linear probing. We visit slots 10, 0, 1, and 2, and finally find an empty slot at position 3.

Table 2:

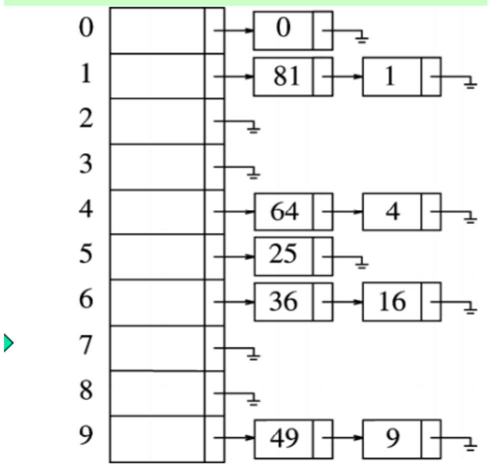


0	1	2	3	4	5	6	7	8	9	10
77	44	20	55	26	93	17	None	None	31	54

2. Separate Chaining: allows many items to exist at the same location in the hash table. When collisions happen, the item is still placed in the proper slot of the hash table. As more and more items hash to the same location, the difficulty of searching for the item in the collection increases. Table 1 shows the items as they are added to a hash table that uses chaining to resolve collisions.



sequence: { 0 1 4 9 16 25 36 49 64 81 }



mid-square method