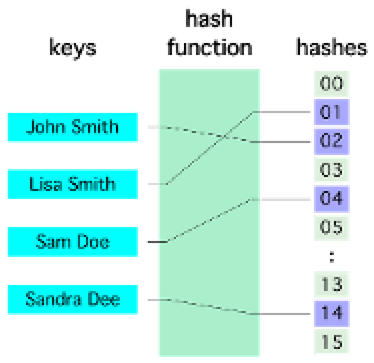
Hashing

Hashing

- Hashing is the process of mapping large amount of data item to smaller table with the help of hashing function.
- Hashing is also known as **Hashing Algorithm** or **Message Digest Function**.
- It is a technique to convert a range of key values into a range of indexes of an array.
- Hashing is used with a database to enable items to be retrieved more quickly.
- It is used in the encryption and decryption of digital signatures.

Hash Function

A function which employs some algorithm to computes the key K for all the data elements in the set U, such that the key K which is of a fixed size. The same key K can be used to map data to a hash table and all the operations like insertion, deletion and searching should be possible. The values returned by a hash function are also referred to as hash values, hash codes, hash sums, or hashes.



Hash Table

It is a Data structure where the data elements are stored (inserted), searched, deleted based on the keys generated for each element, which is obtained from a hashing function. In a hashing system the keys are stored in an array which is called the Hash Table.

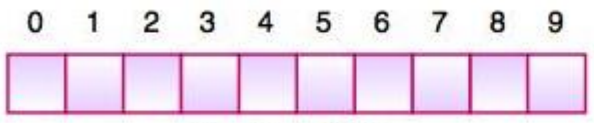


Fig. Hash Table

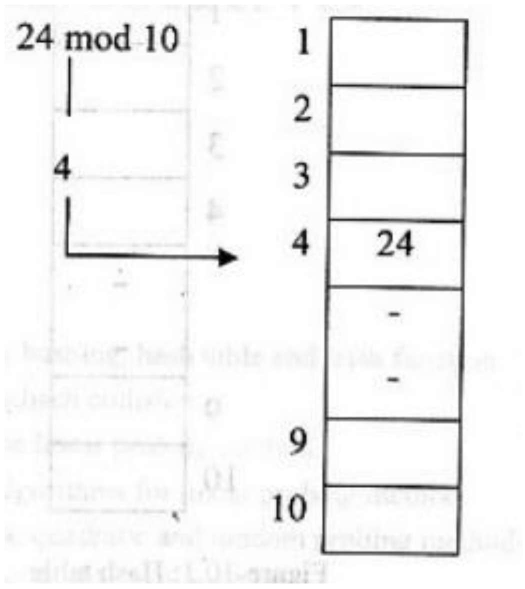
There are several methods to find out the direct address using hash function such as division method, mid-square method, folding method etc.

Division Method

In this method, we use modular arithmetic system to divide the key value by some integer divisor m (may be the table size). It gives us the location value, where the element can be placed. We can write:

h (key_value) = key_value mod m

Example: Let m = 10, key_values are 13, 4, 12, 9, 25 etc. Then direct addresses are 3, 4, 2, 9, 5 etc.



Mid-square Method

In **mid-square method**, we first square the item, and then extract some portion of the resulting digits.

For example, if the item were 44, we would first compute 442=1,936. By extracting the middle two digits, 93, and performing the remainder step, we get 5 (93 % 11).

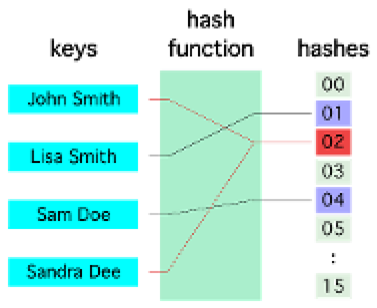
Folding Method

The **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) **(Fold Shifting)**.
- After the addition, 43 + 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 % 11 is 1, so the phone number 436 – 555 - 4601 hashes to slot 1.

- Some folding methods go one step further and reverse every other piece before the addition **(Fold Boundary)**. For the above example, we get 34 + 56 + 55 + 64 + 10=219 which gives 219 % 11=10.

Hash Collision

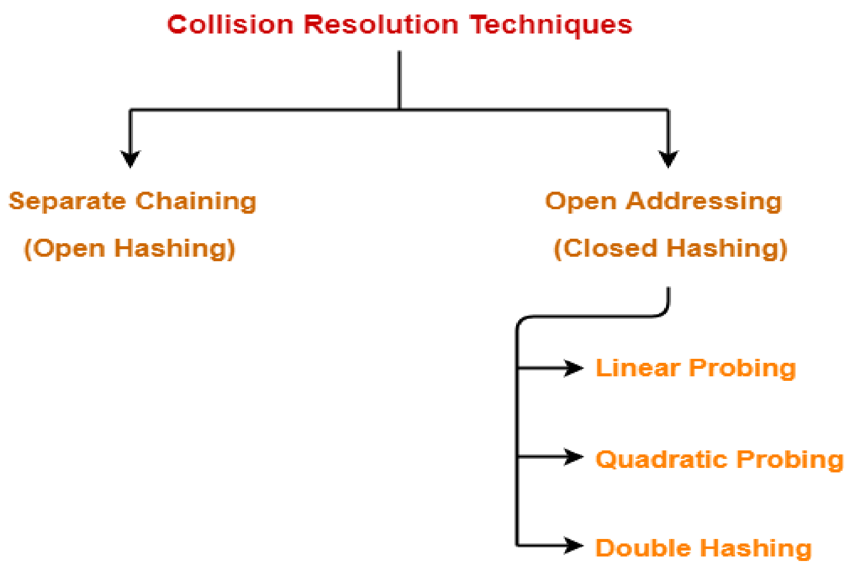


A situation when the resultant hashes for two or more data elements in the data set *U*, maps to the same location in the has table, is called a hash collision.

Collision Resolution

We now return to the problem of collisions. When two items hash to the same slot, we must have a systematic method for placing the second item in the hash table. This process is called collision resolution. As we stated earlier, if the hash function is perfect, collisions will never occur. However, since this is often not possible, collision resolution becomes a very important part of hashing.

Collision resolution techniques are classified as-



1. Separate Chaining

To handle the collision,

- This technique creates a linked list to the slot for which collision occurs.
- The new key is then inserted in the linked list.
- These linked lists to the slots appear like chains. That is why, this technique is called as **separate chaining**.

Problem

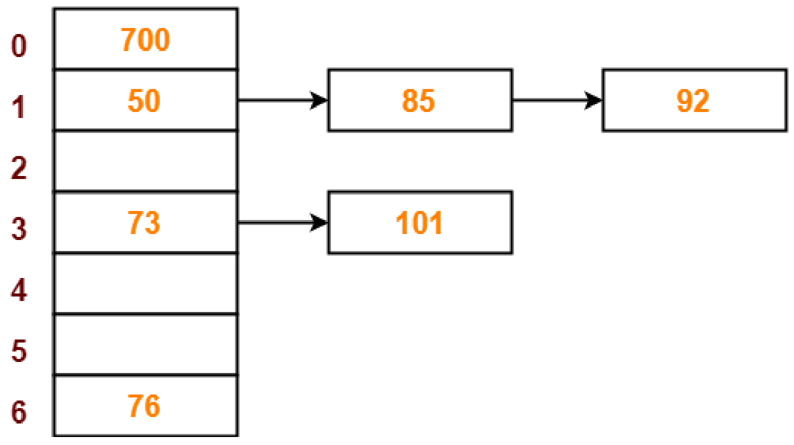
Using the hash function '**key mod 7**', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101

Use separate chaining technique for collision resolution.

Solution

50 mod 7 = 1	700 mod 7 = 0	76 mod 7 = 6
85 mod 7 = 1	92 mod 7 = 1	73 mod 7 = 3
101 mod 7 = 3		



2. Open Addressing

In open addressing,

- Unlike separate chaining, all the keys are stored inside the hash table.
- No key is stored outside the hash table.

i. Linear Probing

In linear probing,

- When collision occurs, we linearly probe for the next bucket.
- We keep probing until an empty bucket is found.

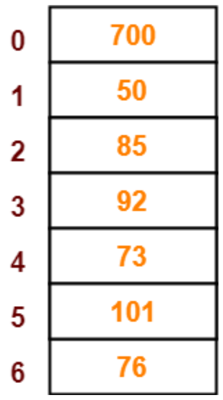
Problem

Using the hash function '**key mod 7'**, insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101

Use linear probing technique for collision resolution.

Solution



50 mod 7 = 1 (free)

700 mod 7 = 0 (free)

76 mod 7 = 6 (free)

85 mod 7 = 1 (occupied); 1 + 1 =2 (free)

92 mod 7 = 1 (occupied); 1 + 1 = 2 (occupied); 1+ 2 = 3 (free)

73 mod 7 = 3 (occupied); 3+ 1 = 4 (free)

101 mod 7 = 3 (occupied); 3 + 1 = 4 (occupied) ; 3 + 2 = 5 (free)

ii. **Quadratic Probing**

A variation of the linear probing idea is called **quadratic probing**. Instead of using a constant "skip" value, we use a rehash function that increments the hash value by 1, 3, 5, 7, 9, and so on. This means that if the first hash value is *h*, the successive values are h+1, h+4, h+9, h+16, and so on.

In general,

$\mathbf{Y_{i+1} = (Y_i + i^2) \bmod m}$; where i = 1, 2, 3 etc. and so on

And $\mathbf{Y_i = key\_value \bmod m}$

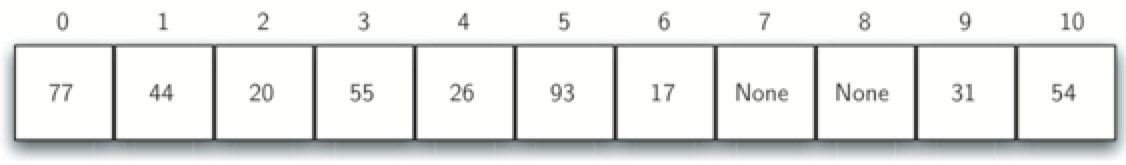
Problem

Using the hash function '**key mod 11'**, insert the following sequence of keys in the hash table-

54, 26, 93, 17, 77, 31, 44, 55 and 20.

Use quadratic probing technique for collision resolution.

Solution



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

54 mod 11 = 10 (free)

26 mod 11 = 4 (free)

93 mod 11 = 5 (free)

17 mod 11 = 6 (free)

77 mod 11 = 0 (free)

31 mod 11 = 9 (free)

44 mod 11 = 0 (occupied)

$(0 + 1^2) \bmod 11 = 1$ (free)

55 mod 11 = 0 (occupied)

$\mathbf{(0 + 1^2) \bmod 11 = 1}$ **(occupied)**

$\mathbf{(0 + 2^2) \bmod 11 = 4}$ **(occupied)**

$\mathbf{(0 + 3^2) \bmod 11 = 9}$ **(occupied)**

$\mathbf{(0 + 4^2) \bmod 11 = 5}$ **(occupied)**

$(0 + 5^2) \bmod 11 = 3$ (free)

20 mod 11 = 9 (occupied)

$\mathbf{(9 + 1^2) \bmod 11 = 10}$ **(occupied)**

$(9 + 2^2) \bmod 11 = 2$ (free)

ii. Double Hashing

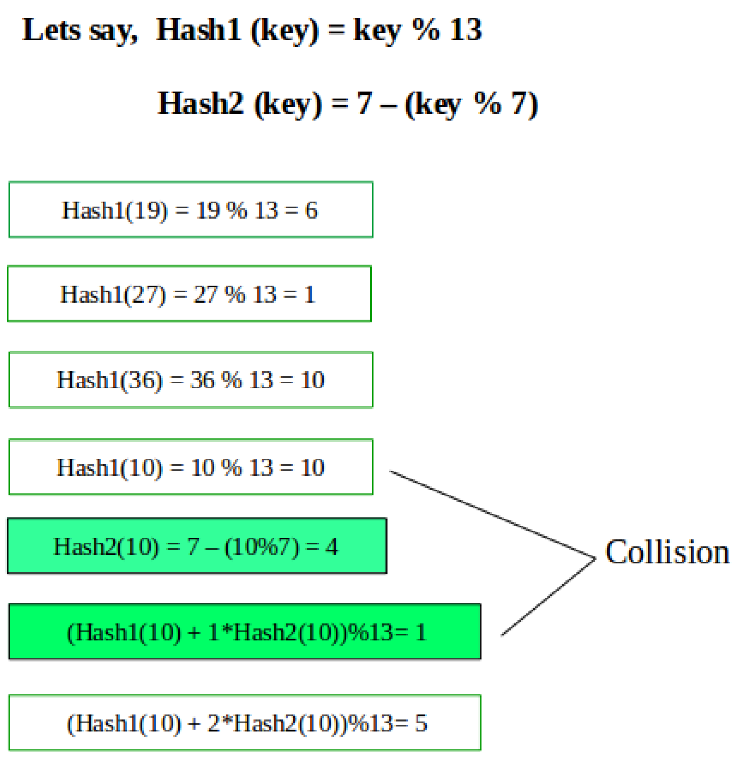
Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Double hashing can be done using :

(h_1(key_value) + i * h_2(key_value)) mod m; Here h_1() and h_2() are hash functions and m is size of hash table. (We repeat by increasing i when collision occurs)

First hash function is typically **h_1(key_value) = key_value mod m**

A popular second hash function is: **h_2(key_value) = PRIME – (key_value mod PRIME)**; where PRIME is a prime smaller than the size of the hash table, m.


Lets say, Hash1 (key) = key % 13
Hash2 (key) = 7 – (key % 7)
Hash1(19) = 19 % 13 = 6
Hash1(27) = 27 % 13 = 1
Hash1(36) = 36 % 13 = 10
Hash1(10) = 10 % 13 = 10
Hash2(10) = 7 – (10%7) = 4
(Hash1(10) + 1*Hash2(10))%13= 1
(Hash1(10) + 2*Hash2(10))%13= 5
Collision

Problem

Consider a hash table of size 10 with hash function **h_1(k) = (2k + 3) mod m**. Draw the table that results after inserting, in the given order, the following values: **3, 2, 9, 6, 11, 13, 7, 12.** When collisions are handled by double hashing using a second hash function **h_2 (k) = (3k + 1) mod m**

Solution

$h_1(3) = (2*3 + 3) \bmod 10 = 9$ (free)

$h_1(2) = (2*2 + 3) \bmod 10 = 7$ (free)

$h_1(9) = (2*9 + 3) \bmod 10 = 1$ (free)

$h_1(6)$ = (2*6 + 3) mod 10 = 5 (free)

$h_1(11)$ = (2*11 + 3) mod 10 = 5 (occupied)

$h_2(11)$ = (3*11 + 1) mod 10 = 4

($h_1(11)$ + i * $h_2(11)$) mod m = (5 + 0 * 4) mod 10 = 5 (occupied)

($h_1(11)$ + i * $h_2(11)$) mod m = (5 + 1 * 4) mod 10 = 9 (occupied)

($h_1(11)$ + i * $h_2(11)$) mod m = (5 + 2 * 4) mod 10 = 3 (free)

$h_1(13)$ = (2*13 + 3) mod 10 = 9 (occupied)

$h_2(13)$ = (3*13 + 1) mod 10 = 0

($h_1(13)$ + i * $h_2(13)$) mod m = (9 + 0 * 0) mod 10 = 9 (occupied) [We are not able to add 13 to the hash table]

$h_1(7)$ = (2*7 + 3) mod 10 = 7 (occupied)

$h_2(7)$ = (3*7 + 1) mod 10 = 2

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 0 * 2) mod 10 = 7 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 1 * 2) mod 10 = 9 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 2 * 2) mod 10 = 1 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 3 * 2) mod 10 = 3 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 4 * 2) mod 10 = 5 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 5 * 2) mod 10 = 7 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 6 * 2) mod 10 = 9 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 7 * 2) mod 10 = 1 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 8 * 2) mod 10 = 3 (occupied)

($h_1(7)$ + i * $h_2(7)$) mod m = (7 + 9 * 2) mod 10 = 5 (occupied) [We are not able to add 7 to the hash table]

$h_1(12)$ = (2*12 + 3) mod 10 = 7 (occupied)

$h_2(12)$ = (3*12 + 1) mod 10 = 7

($h_1(12)$ + i * $h_2(12)$) mod m = (7 + 0 * 7) mod 10 = 7 (occupied)

($h_1(12)$ + i * $h_2(12)$) mod m = (7 + 1 * 7) mod 10 = 4 (free)

0	1	2	3	4	5	6	7	8	9
	9		11	12	6		2		3