Algorithms and Data Structures Using Java

Soumya

Hashing

• Hashing is the process of transforming any given key or a string of characters into another value.

• This is usually represented by a shorter, fixed-length value or key that represents and makes it easier to find or employ the original string.

Unlike other searching techniques,

– Hashing is extremely efficient.

– The time taken by it to perform the search does not depend upon the total number of elements.

– It completes the search with constant time complexity O(1).

Hashing

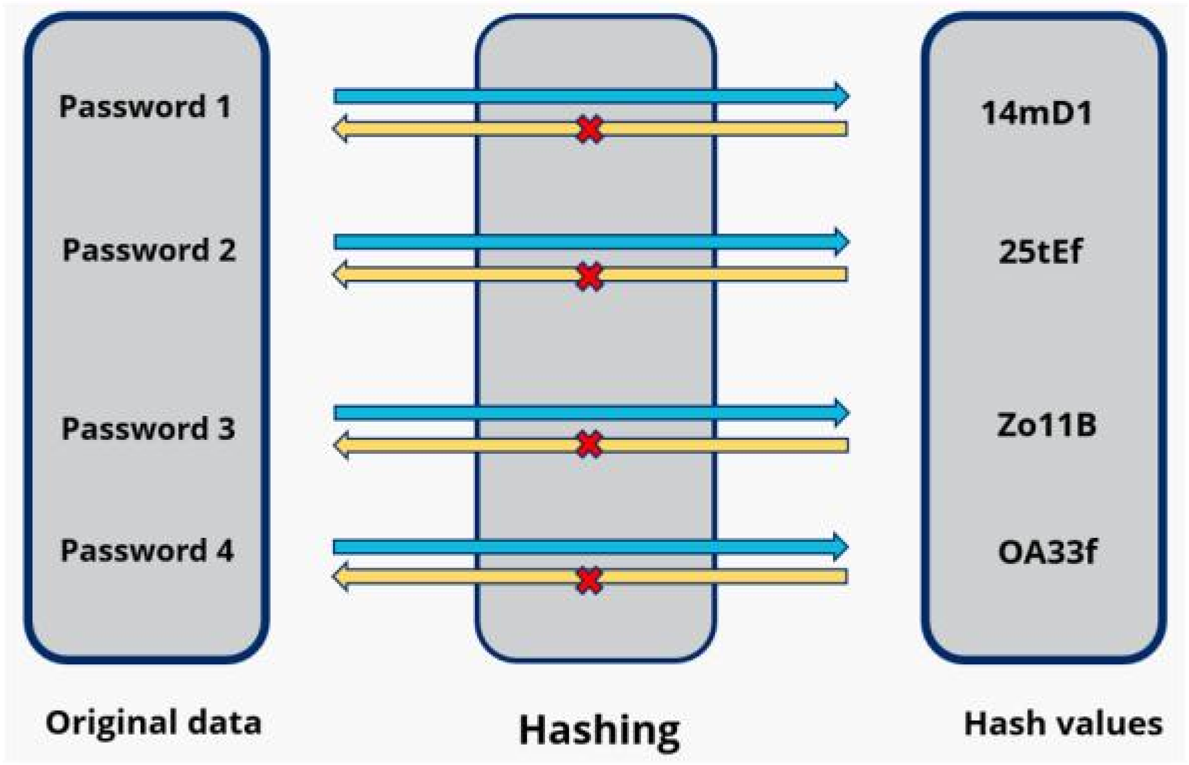
- A hash table stores key and value pairs in a list that is accessible through its index.
- Because key and value pairs are unlimited, the hash function will map the keys to the table size.
- A hash value then becomes the index for a specific element.

Hashing

- A hash function generates new values according to a mathematical hashing algorithm, known as a hash value or simply a hash.
- To prevent the conversion of hash back into the original key, a good hash always uses a one-way hashing algorithm.
- Hashing is relevant to -- but not limited to -- data indexing and retrieval, digital signatures,cybersecurity and cryptography.

Hashing

- Hashing uses functions or algorithms to map object data to a representative integer value.
- A hash can then be used to narrow down searches when locating these items on that object data map.
- For example, in hash tables, developers store data – perhaps a customer record -- in the form of key and value pairs.
- The key helps identify the data and operates as an input to the hashing function, while the hash code or the integer is then mapped to a fixed size.


Password 1
Password 2
Password 3
Password 4
14mD1
25tEf
Zo11B
OA33f
Original data
Hashing
Hash values

Advantages of Hashing

Here, are pros/benefits of using hash tables:

1. Hash tables have high performance when looking up data,inserting, and deleting existing values.
2. The time complexity for hash tables is constant regardless of the number of items in the table.
3. They perform very well even when working with large datasets.

Disadvantages of Hashing

Here, are cons of using hash tables:

1. You cannot use a null value as a key.
2. Collisions cannot be avoided when generating keys using. hash functions. Collisions occur when a key that is already in use is generated.
3. If the hashing function has many collisions, this can lead to performance decrease.

Hashing : Applications

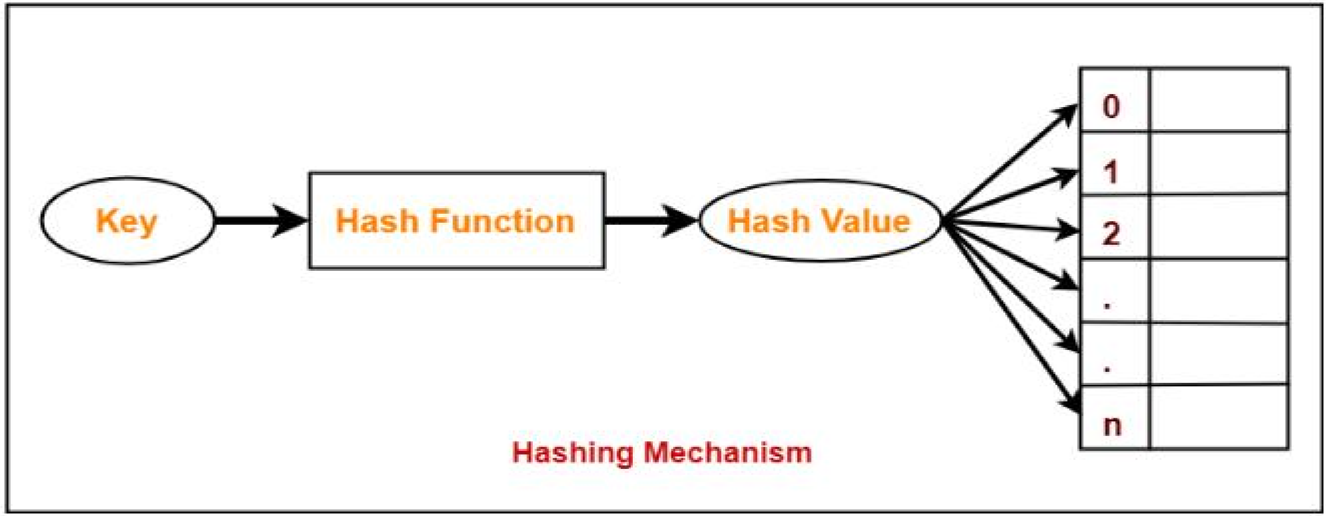
- Databases: Hashing can be used to index data in a database,which makes it faster to search for and retrieve data.
- Cryptography: Hashing can be used to create digital signatures and other cryptographic primitives.
- Caching: Hashing can be used to implement caches, which are data structures that store frequently accessed data in memory for faster access.
- Bloom filters: Hashing can be used to implement Bloom filters, which are a space-efficient probabilistic data structure used to test whether an element is a member of a set.

Hashing Mechanism

- An array data structure called as Hash table is used to store the data items.
- Based on the hash key value, data items are inserted into the hash table.

Hash Key value

- Hash key value is a special value that serves as an index for a data item.
- It indicates where the data item should be be stored in the hash table.
- Hash key value is generated using a hash function.



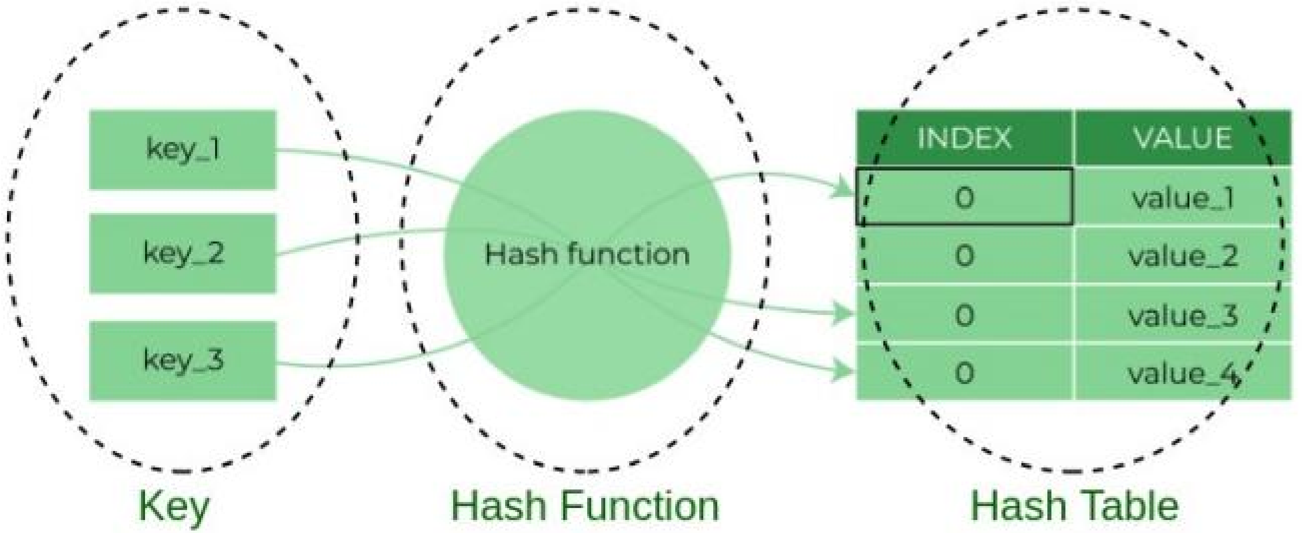
Hash Table

- A hash table is a data structure that maps keys to values. It is implemented using an array, where each element of the array is a linked list of key-value pairs.
- The keys are hashed into a fixed-size bucket index using a hash function.
- To insert a new key-value pair into the hash table, the hash function is used to generate a bucket index.
- The new key-value pair is then added to the linked list at the corresponding bucket index.

Hash Table

To search for a value in the hash table, the hash function is used to generate a bucket index.

- The linked list at the corresponding bucket index is then searched for the key. If the key is found, the corresponding value is returned. Otherwise, null is returned.
- Hash tables are a very efficient way to store and retrieve data, especially when the keys are evenly distributed.
- However, if the keys are not evenly distributed, the hash table can become inefficient.


key_1
key_2
key_3
Hash function
INDEX
VALUE
0
value_1
0
value_2
0
value_3
0
value_4
Key
Hash Function
Hash Table

Hash Table

Here are some examples of how hash tables are used:

– In a database, a hash table can be used to index the data, which makes it faster to search for and retrieve data.

– In a web browser, a hash table can be used to cache frequently accessed web pages.

– In a compiler, a hash table can be used to store the symbols in the program.

– In a programming language, a hash table can be used to implement a dictionary or associative array.

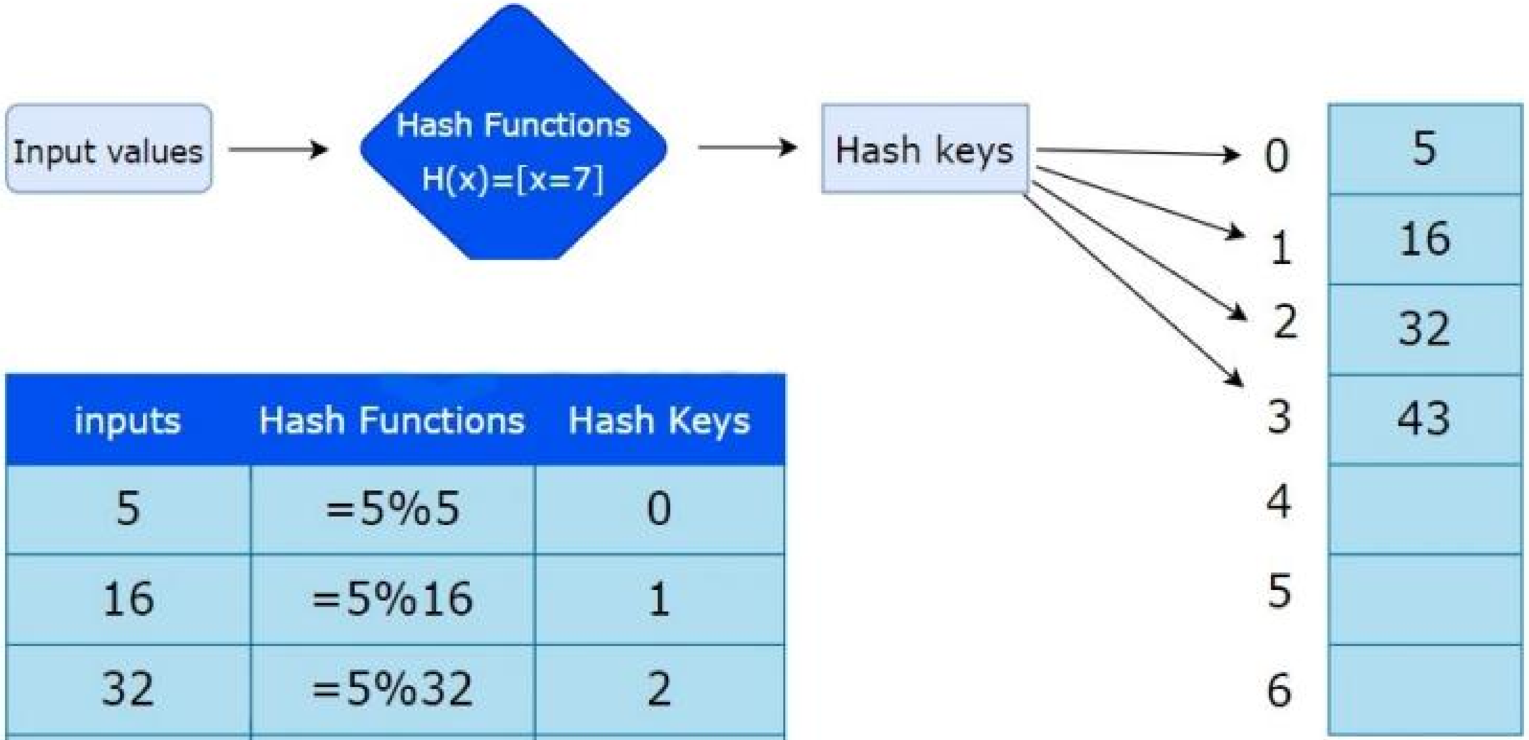
Hash Function

- Hash function is a function that maps any big number or string to a small integer value.
- Hash function takes the data item as an input and returns a small integer value as an output.
- The small integer value is called as a hash value.
- Hash value of the data item is then used as an index for storing it into the hash table.

Bucket

- Bucket is an index position in hash table that can store more than one record
- When the same index is mapped with two keys, then both the records are stored

  in the same bucket



inputs	Hash Functions	Hash Keys
5	=5%5	0
16	=5%16	1
32	=5%32	2
43	=5%43	3

Hash Function

The primary types of hash functions are:

– Division Method.

– Mid Square Method.

– Folding Method.

– Multiplication Method.

Properties of Hash Function

- The properties of a good hash function are-

– It is efficiently computable.

– It minimizes the number of collisions.

– It distributes the keys uniformly over the table.

Types of Hash Functions

- **Division Method**

– The easiest and quickest way to create a hash value is through division.

The k-value is divided by M in this hash function, and the result is used.

- Formula:

h(K) = k mod M

- (where k = key value and M = the size of the hash table)

Types of Hash Functions

- Advantages:

– This method is effective for all values of M.

– The division strategy only requires one operation, thus it is quite quick.

- Disadvantages:

– Since the hash table maps consecutive keys to successive hash values, this could result in poor performance.

– There are times when exercising extra caution while selecting M's value is necessary.

Types of Hash Functions

Example:

$k = 1987$

$M = 13$

$h(1987) = 1987 \bmod 13$

$h(1987) = 4$

Types of Hash Functions

Mid Square Method

• The following steps are required to calculate this hash method:

– k*k, or square the value of k

– Using the middle r digits, calculate the hash value.

• Formula:

h(K) = h(k x k) (where k = key value)

Types of Hash Functions

- Advantages:

– This technique works well because most or all of the digits in the key value affect the result. All of the necessary digits participate in a process that results in the middle digits of the squared result.

– The result is not dominated by the top or bottom digits of the initial key value.

- Disadvantages:

– The size of the key is one of the limitations of this system; if the key is large, its square will contain twice as many digits.

– Probability of collisions occurring repeatedly.

Types of Hash Functions

Example:

k = 60

Therefore,

k = k x k

k = 60 x 60

k = 3600

Thus,

h(60) = 60

Types of Hash Functions

Folding Method

- ➢ The process involves two steps:
- ➢ With the exception of the last component, which may have fewer digits than the other parts, the key-value k should be divided into a predetermined number of pieces, such as k1, k2, k3,..., kn, each having the exact same amount of digits.
- ➢ Add each element individually. The hash value is calculated without taking into account the final carry, if any.

Types of Hash Functions

Advantages:

– Creates a simple hash value by precisely splitting the key value into equal-sized segments.

– Without regard to distribution in a hash table.

• Disadvantages:

– When there are too many collisions, efficiency can occasionally suffer.

Types of Hash Functions

Formula:

$k = k1, k2, k3, k4, ..., kn$

$s = k1 + k2 + k3 + k4 + ... + kn$

$h(K) = s$

(Where, s = addition of the parts of key k)

Types of Hash Functions

Example:

k = 12345

k1 = 12, k2 = 34, k3 = 5

s = k1 + k2 + k3

= 12 + 34 + 5

= 51

h(K) = 51

Types of Hash Functions

Multiplication Method

– Determine a constant value. A, where (0, A, 1)

– Add A to the key value and multiply.

– Consider kA's fractional portion.

– Multiply the outcome of the preceding step by M, the hash table's size.

• Formula:

h(K) = floor (M (kA mod 1))

(Where, M = size of the hash table, k = key value and A = constant value)

Types of Hash Functions

Advantages:

– Any number between 0 and 1 can be applied to it, however, some values seem to yield better outcomes than others.

Disadvantages:

– The multiplication method is often appropriate when the table size is a power of two since multiplication hashing makes it possible to quickly compute the index by key.

Types of Hash Functions

- Example:

k = 5678

A = 0.6829

M = 200

Now, calculating the new value of h(5678):

h(5678) = floor[200(5678 x 0.6829 mod 1)]

h(5678) = floor[200(3881.5702 mod 1)]

h(5678) = floor[200(0.5702)]

h(5678) = floor[114.04]

h(5678) = 114

So, with the updated values, h(5678) is 114.

Collision in Hashing

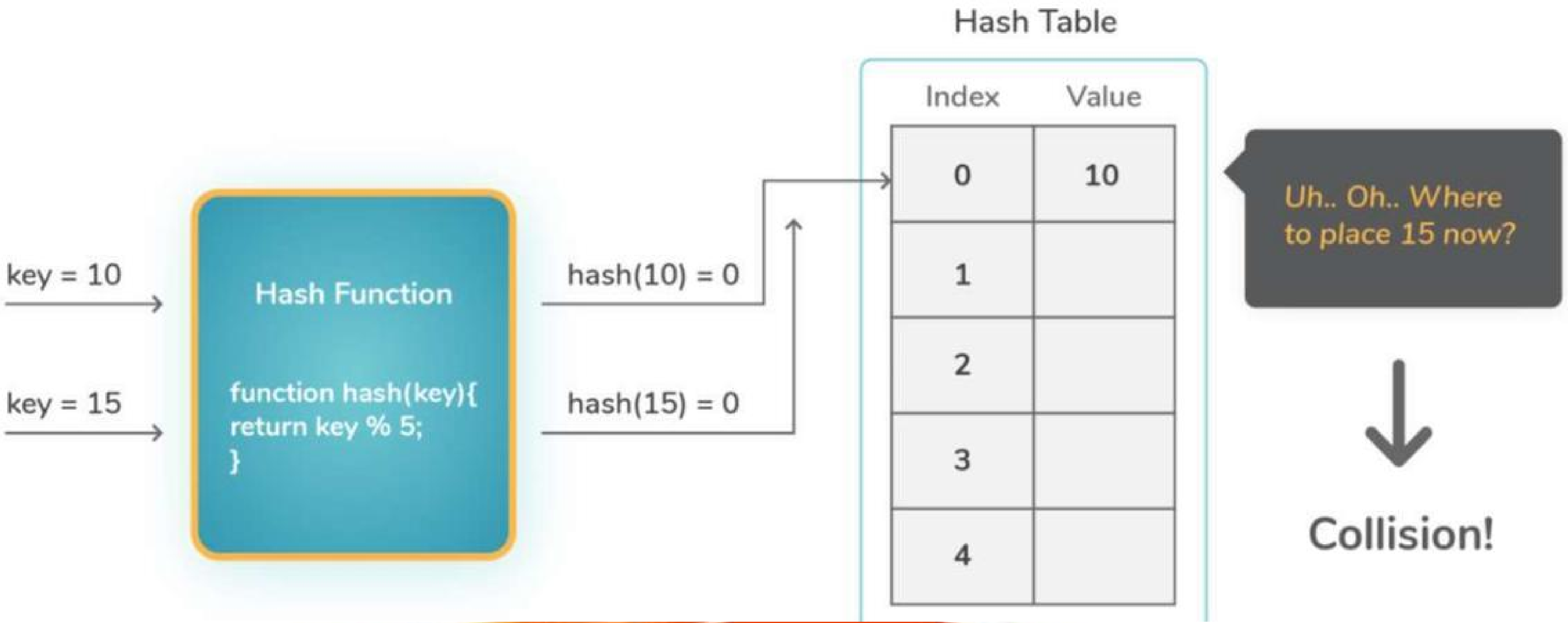
- Hash function is used to compute the hash value for a key.
- Hash value is then used as an index to store the key in the hash table.
- Hash function may return the same hash value for two or more keys.
- When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a Collision.

Collision in Hashing

– When a hash algorithm generates the same hash value for two separate input values, this is known as a hash collision.

- However, it's crucial to note that collisions are not an issue; rather, they constitute a key component of hashing algorithms.
- Because various hashing methods used in data structures convert each input into a fixed-length code regardless of its length, collisions happen.
- The hashing algorithms will eventually yield repeating hashes since there are an infinite number of inputs and a finite number of outputs.

- The result of two keys hashing into the same address is called collision

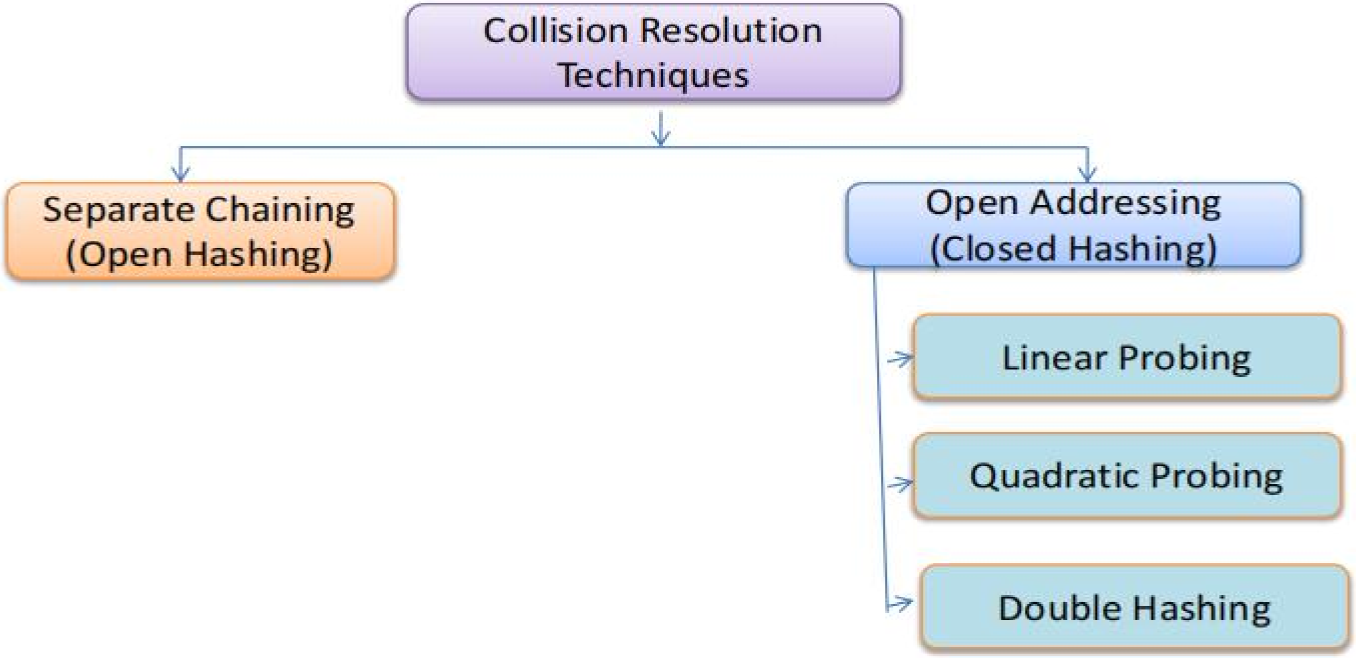


```
function hash(key){
return key % 5;
}
```

Index	Value
0	10
1	
2	
3	
4	

Collision Resolution Techniques

• Collision Resolution Techniques are the techniques used for resolving or handling the collision.


Collision Resolution Techniques
Separate Chaining (Open Hashing)
Open Addressing (Closed Hashing)
Linear Probing
Quadratic Probing
Double Hashing

Collision Resolution Techniques

The following are the collision techniques:

- Open Hashing: It is also known as closed addressing.
- Closed Hashing: It is also known as open addressing.

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.

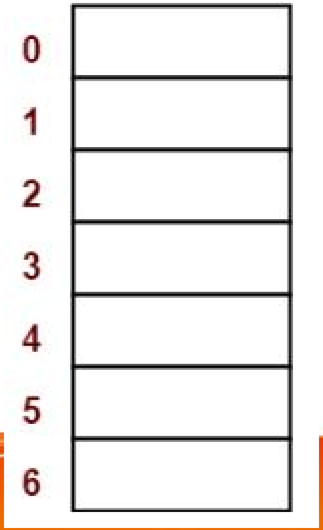
– That is why, this technique is called as separate chaining.

Example-Separate Chaining

- 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

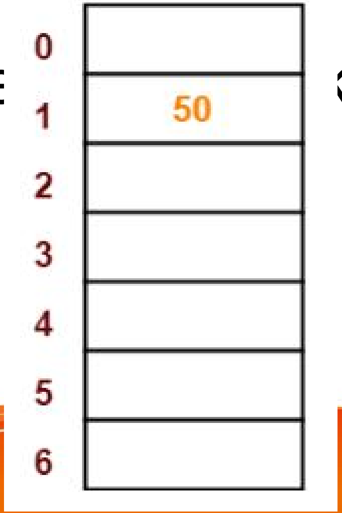
Step-1

- Draw an empty hash table.
- For the given hash function, the possible range of hash values is [0, 6].
- So, draw an empty hash table consisting of 7 buckets as-



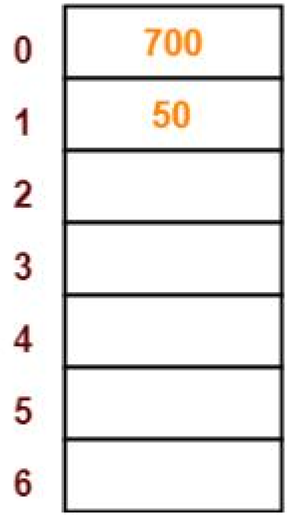
Step-2

- Insert the given keys in the hash table one by one.
- The first key to be inserted in the hash table = 50.
- Bucket of the hash table to which key 50 maps = 50 mod 7 = 1.
- So, key 50 will be inse ket-1 of the hash table as-



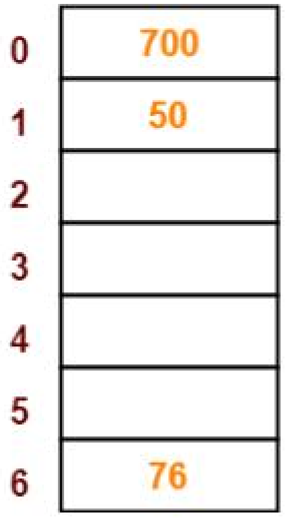
Step-3

- The next key to be inserted in the hash table = 700.
- Bucket of the hash table to which key 700 maps = 700 mod 7 = 0.
- So, key 700 will be inserted in bucket-0 of the hash table as-



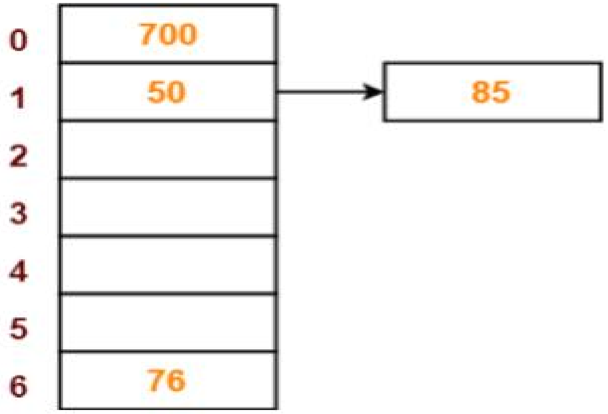
Step- 4

- The next key to be inserted in the hash table = 76.
- Bucket of the hash table to which key 76 maps = 76 mod 7 = 6.
- So, key 76 will be inserted in bucket-6 of the hash table as-



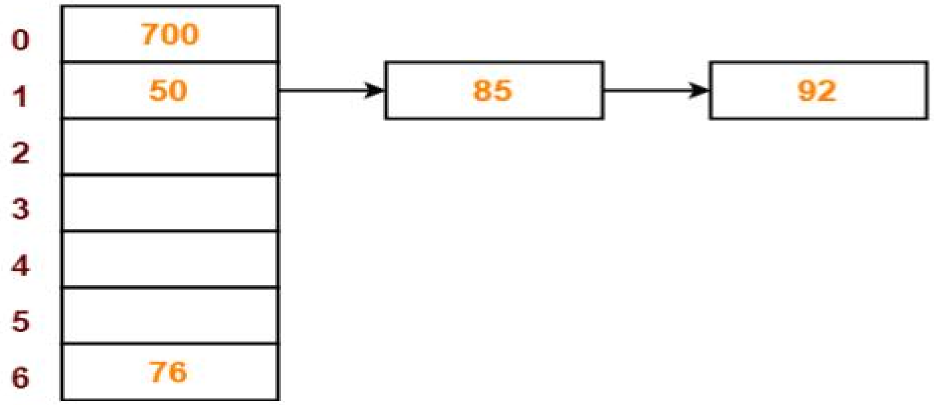
Step - 5

- The next key to be inserted in the hash table = 85.
- Bucket of the hash table to which key 85 maps = 85 mod 7 = 1.
- Since bucket-1 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-1.
- So, key 85 will be inserted in bucket-1 of the hash table as-



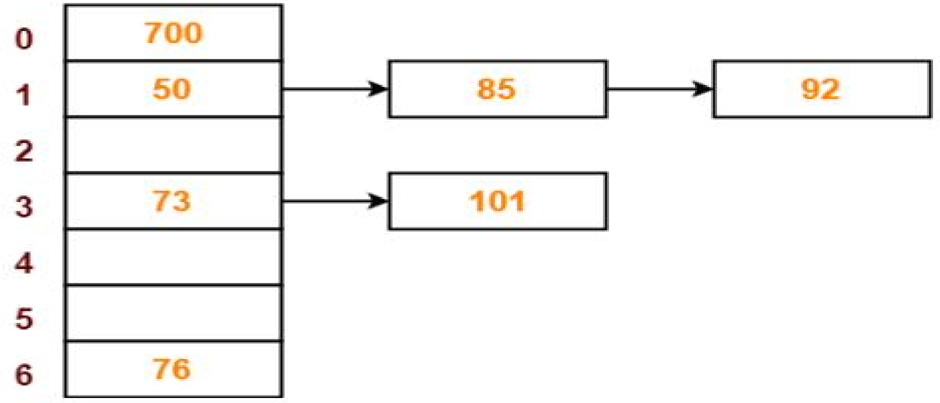
Step - 6

- The next key to be inserted in the hash table = 92.
- Bucket of the hash table to which key 92 maps = 92 mod 7 = 1.
- Since bucket-1 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-1.
- So, key 92 will be inserted in bucket-1 of the hash table as-



Step - 7

- The next key to be inserted in the hash table = 101.
- Bucket of the hash table to which key 101 maps = 101 mod 7 = 3.
- Since bucket-3 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-3.
- So, key 101 will be inserted in bucket-3 of the hash table as-



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.

- Techniques used for open addressing are-

– Linear Probing

– Quadratic Probing

– Double Hashing

Linear Probing

- Linear probing includes inspecting the hash table sequentially from the very beginning. If the site requested is already occupied, a different one is searched. The distance between probes in linear probing is typically fixed (often set to a value of 1).

- Formula:

index = key % hash Table Size

- Sequence

index = (hash(n) % T)

(hash(n) + 1) % T

(hash(n) + 2) % T

(hash(n) + 3) % T ... and so on.

Linear probing

In linear probing,

– When collision occurs, we linearly probe for the next bucket.

– We keep probing until an empty bucket is found.

• Advantage-

– It is easy to compute.

• Disadvantage-

– The main problem with linear probing is clustering.

– Many consecutive elements form groups.

– Then, it takes time to search an element or to find an empty bucket.

Linear Probing:Example

Let's understand the linear probing through an example.

Consider the above example for the linear probing:

A = 3, 2, 9, 6, 11, 13, 7, 12 where m = 10, and h(k) = 2k+3

The key values 3, 2, 9, 6 are stored at the indexes 9, 7, 1, 5 respectively.

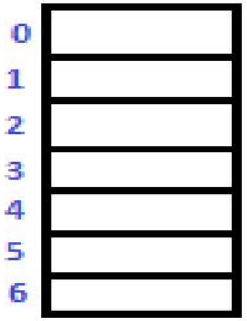
The calculated index value of 11 is 5 which is already occupied by another key value, i.e., 6.

When linear probing is applied, the nearest empty cell to the index 5 is 6; therefore, the value 11 will be added at the index 6.

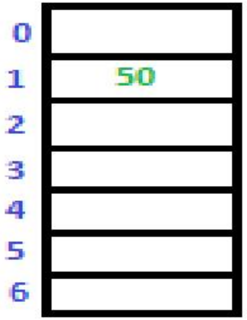
The next key value is 13. The index value associated with this key value is 9 when hash function is applied. The cell is already filled at index 9. When linear probing is applied, the nearest empty cell to the index 9 is 0; therefore, the value 13 will be added at the index 0

CLOSED HASHING

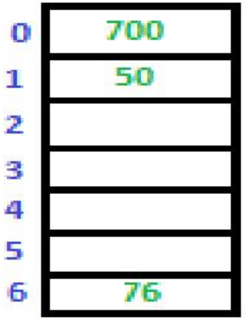
Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



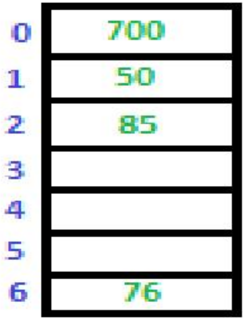
Initial Empty Table



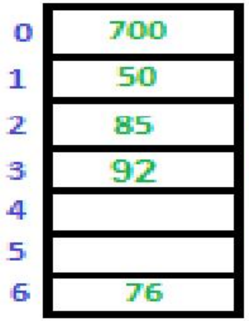
Insert 50



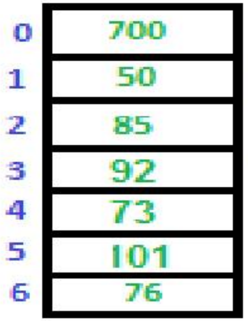
Insert 700 and 76



Insert 85: Collision Occurs, insert 85 at next free slot.



Insert 92, collision occurs as 50 is there at index 1. Insert at next free slot



Insert 73 and 101

Quadratic Probing

The distance between subsequent probes or entry slots is the only difference between linear and quadratic probing. You must begin traversing until you find an available hashed index slot for an entry record if the slot is already taken. By adding each succeeding value of any arbitrary polynomial in the original hashed index, the distance between slots is determined.

- Formula:

index = index % hash Table Size

- Sequence:

index = (hash(n) % T)

(hash(n) + 1 x 1) % T

(hash(n) + 2 x 2) % T

(hash(n) + 3 x 3) % T ... and so on

Quadratic Probing

- In case of linear probing, searching is performed linearly. In contrast,quadratic probing is an open addressing technique that uses quadratic polynomial for searching until a empty slot is found.
- It can also be defined as that it allows the insertion ki at first free location from (u+i2)%m where i=0 to m-1.

$h' = (x) = x \bmod m$

$h(x, i) = (h'(x) + i^2) \bmod m$

We can put some other quadratic equations also using some constants

The value of i = 0, 1, . . ., m – 1. So we start from i = 0, and increase this until we get one free space. So initially when i = 0, then the h(x, i) is same as h´(x).

Quadratic Probing -- Example

- Example:
  - Table Size is 11 (0..10)
  - Hash Function: **h(x) = x mod 11**
  - Insert keys: 20, 30, 2, 13, 25, 24, 10, 9
    - 20 mod 11 = 9
    - 30 mod 11 = 8
    - 2 mod 11 = 2
    - 13 mod 11 = 2 $\rightarrow$ $2+1^2=3$
    - 25 mod 11 = 3 $\rightarrow$ $3+1^2=4$
    - 24 mod 11 = 2 $\rightarrow$ $2+1^2$, $2+2^2=6$
    - 10 mod 11 = 10
    - 9 mod 11 = 9 $\rightarrow$ $9+1^2$, $9+2^2$ mod 11, $9+3^2$ mod 11 =7

0	
1	
2	2
3	13
4	25
5	
6	**24**
7	**9**
8	30
9	20
10	10

Double Probing/Double Hashing

- The time between probes is determined by yet another hash function. Double hashing is an optimized technique for decreasing clustering. The increments for the probing sequence are computed using an extra hash function.

- Formula

(first hash(key) + i * second Hash(key)) % size of the table

- Sequence

index = hash(x) % S

(hash(x) + 1*hash2(x)) % S

(hash(x) + 2*hash2(x)) % S

(hash(x) + 3*hash2(x)) % S ... and so on

Double Probing/Double Hashing

It is a collision resolution technique which uses two hash functions to handle collision.

The interval (i.e., the distance it probes) is decided by the second hash function which is independent.

The table size should be chosen in such a way that it is prime so that all the cells can be inserted. The second hash function should be chosen in such a way that the function does not evaluate to zero.i.e., 1 can be added to the hash function(non-zero). To overcome secondary clustering, double hashing is used. The collision function is,

hi(key)=(Hash(key)+ F(i)) % Tablesize

F(i) = i * hash2 (X)

Where **hash2(X) = R – (X % R)** R is a prime number. It should be smaller than the tablesize

Double Probing/Double Hashing

Example 1 :89, 18, 49, 58, 69,60

h0(49)=9

h1(49) = (9+(1*7)) % 10 = (9+7) % 10 = 16 % 10 = 6

58

7 – (58 % 7) = 7 – 2 = 5

i=1

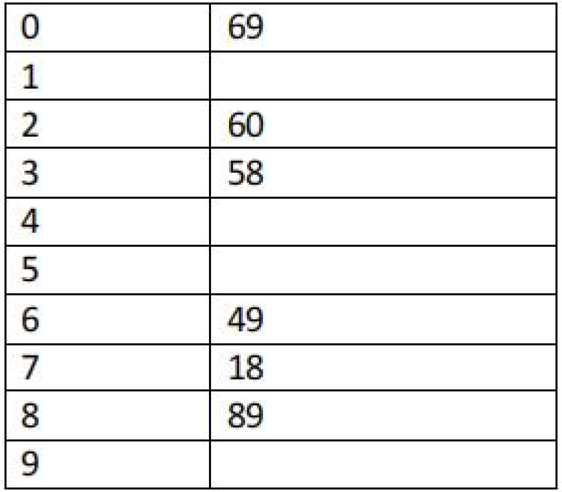
h1(58) = (8+(1*5)) % 10 = 13 % 10 = 3

69

7 – (69 % 7) = 7 – 6 = 1

h1(69) = (9+1) % 10 = 10 % 10 = 0

60



0	69
1	
2	60
3	58
4	
5	
6	49
7	18
8	89
9	

Double Probing/Double Hashing

7 – (60 % 7) = 7 – 4 = 3

h1(60) = (0+1*3) % 10 = 3

h2(60) = (0+2*3) % 10 = 6

h3(60) = (0+9) % 10 = 9

h4(60)=(0+12)%10=2

Double Probing/Double Hashing

APPLICATIONS:

- ➢ It is used in caches.
- ➢ It is used in finding duplicate records.
- ➢ Used in finding similar records.
- ➢ Finding similar substrings.

ADVANTAGES:

- ➢ No index storage is required.
- ➢ It provides rapid updates.

DISADVANTAGES:

- ➢ Problem arises when the keys are too small.
- ➢ There is no sequential retrieval of keys.
- ➢ Insertion is more random.

Re-Hashing

- It is a technique in which the table is resized i.e., the size of the table is doubled by creating a new table.
- Rehashing is a technique that is used to improve the efficiency of the closed hashing techniques. This can be done by reducing the running time. If the table gets too full, the running time for the operations will start taking too long and *inserts* might fail for closed hashing with quadratic resolution. This can happen if there are too many deletions intermixed with insertions. A solution, then, is to build another table that is about twice as big (with associated new hash function) and scan down the entire original hash table, computing the new hash value for each (non-deleted) element and inserting it in the new tableRe-Hashing is required when, The table is completely filled in the case of double-Hashing.
- The table is half-filled in the case of quadratic and linear probing
- The insertion fails due to overflow.
- **IMPLEMENTATION:**
- Rehashing can be implemented in
- 1.Linear probing
- 2.Double hashing
- 3.Quadratic probing

Separate Chaining	Open Addressing
Keys are stored inside the hash table as well as outside the hash table.	All the keys are stored only inside the hash table.
The number of keys to be stored in the hash table can even exceed the size of the hash table.	The number of keys to be stored in the hash table can never exceed the size of the hash table.
Deletion is easier.	Deletion is difficult.
Extra space is required for the pointers to store the keys outside the hash table.	No extra space is required.
Cache performance is poor. This is because of linked lists which store the keys outside the hash table.	Cache performance is better. This is because here no linked lists are used.
Some buckets of the hash table are never used which leads to wastage of space.	Buckets may be used even if no key maps to those particular buckets.

Hashing Example:Phone Book

Design a data structure to store your contacts: names of people along with their phone numbers. The following operations should be fast:

- ➢ Add and delete contacts,
- ➢ Call person by name,
- ➢ Determine who is calling given their phone number.

Hashing Example:Phone Book

We need two Maps:

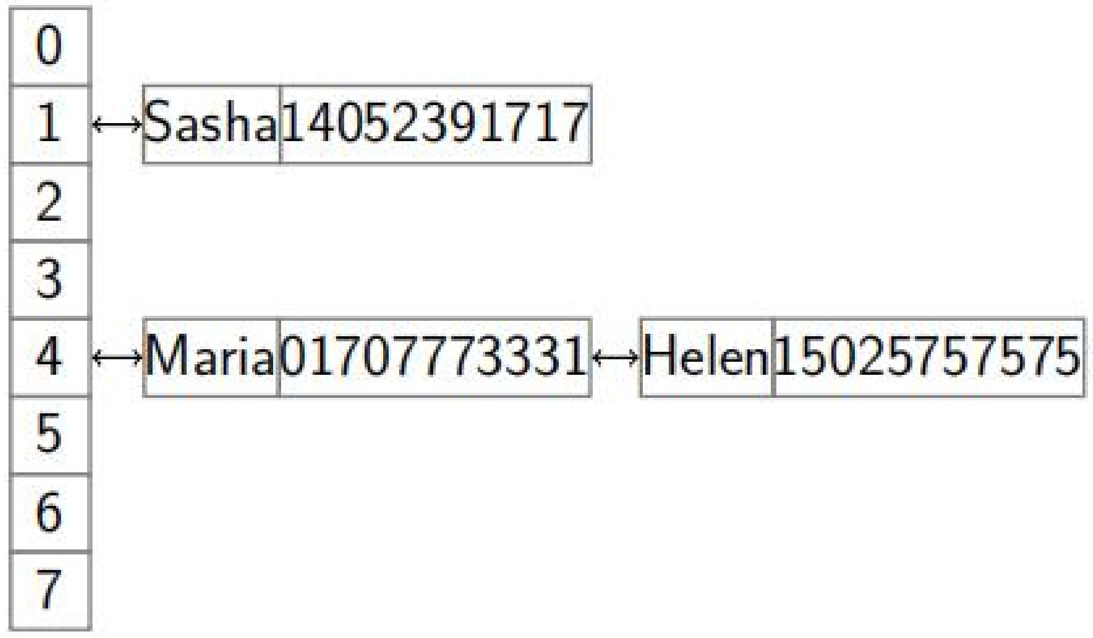
- (phone number → name) and
- (name → phone number)
- Implement these Maps as hash tables
- First, we will focus on the Map from phone numbers to names

Hashing Example:Phone Book

Chaining for Phone Book

- Select hash function *h* of cardinality *m*
- Create array Chains of size *m*
- Each element of Chains is a list of pairs (name, phoneNumber), called *chain*
- Pair (name, phoneNumber) goes into chain at position
- *h*(Convert To Int(phoneNumber)) in the array Chain

Hashing Example: Phone Book


0
1
2
3
4
5
6
7
Sasha
14052391717
Maria
01707773331
Helen
15025757575