6 Hashing

HASHING TECHNIQUES

Introduction:

- Hashing is a searching technique.
- Hashing is an efficient searching technique in which the number of keys that must be inspected before obtaining the desired one is minimised.
- Worst case, time complexity in case of hashing is expected as O(1).
- Basically, the motive behind hashing is to retrieve each key stored in the table in a single access.
- In hashing, the location of the record within the table depends only on the key resulting in retrieving each record in a single access. It does not depend on the locations of other keys, as in a tree.

Hash table and hash function:

1) Hash table:

- Hash table is a table used to maintain keys in it.
- The most efficient way to organize such a table is as array.
- Each key is stored in the hash table at a specific offset from the base address of the table.
- If the record keys are integers, the keys themselves can serve as indices to the array, which is used as a table.
- An example of a hash table using an array:

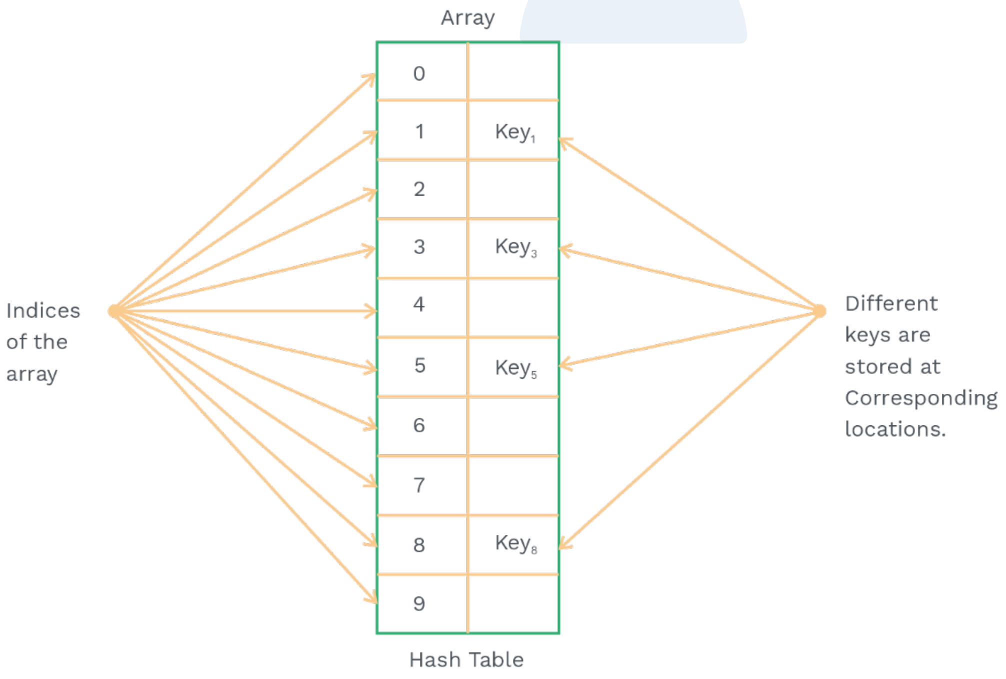


Table 6.1

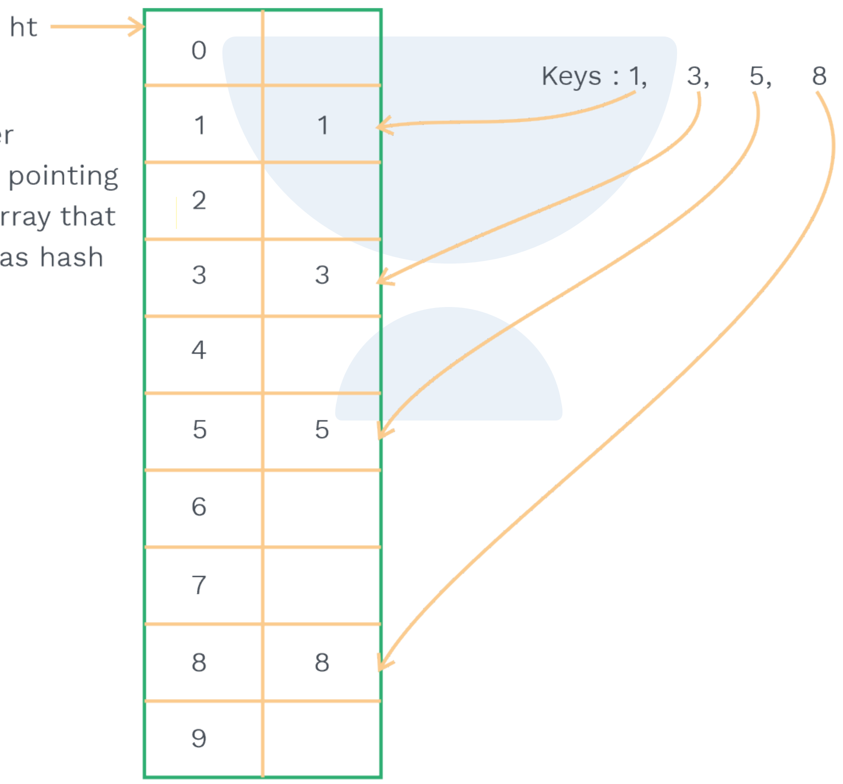
2) Hash function:

- A function which transforms a key into a table index is called as a hash function.
- Consider the situation where keys are supposed to be stored in a hash table directly by their values, i.e., key ‘k‘ will be stored in the kth position only.

For example:

⇒ keys are 1, 3, 5, and 8, so we need a table of size at least 8. let’s take it of size 10.

⇒ Each key will go to its corresponding location as shown below:



ht: A pointer variable pointing to the array that is used as hash table

Table 6.2

The above hash table is called the direct address table.

- But suppose, if one key value is large enough,

For example:

keys are 1, 3, 5, and 100058.

So, the minimum table size required is 1,00,058. Thus, for storing only 4 keys

(1, 3, 5, and 100058), a table of size 1,00,058 is needed, by doing so, a lot of space is wasted.

- To overcome the above situation, the hash function is used.
- Basically, the hash function is used to compress the keys which are to be stored in the hash table.
- If ‘h’ is a hash function and the key is ‘k’, then h(k) is called the ‘hash value of key’.
- Let’s understand the hash function by taking h(k) = k%10, where ‘%’ is the modulo operator.

Example:

Keys are 105, 110, 202, 403, 509

Since the hash function used is “key modulo 10”, so table size should be 10.

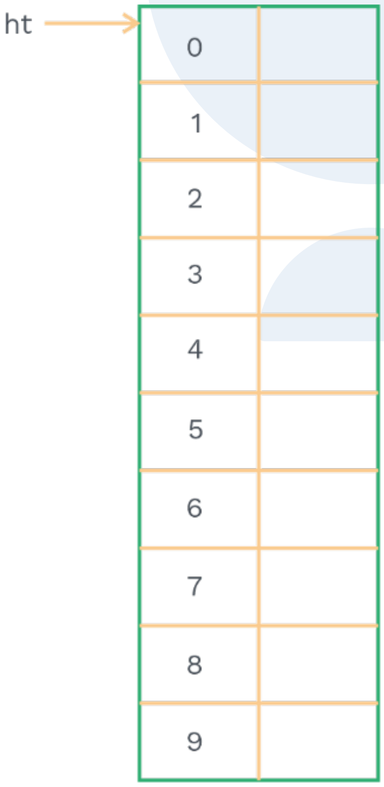


Table 6.3

Table size is 10 because after applying hash function h(k) = k%10, the last digit of the key will decide the location/position where the key is to be stored. Thus, 0 to 9 digits are sufficient to store all the keys.

So, h(105) = 105 % 10 = 5

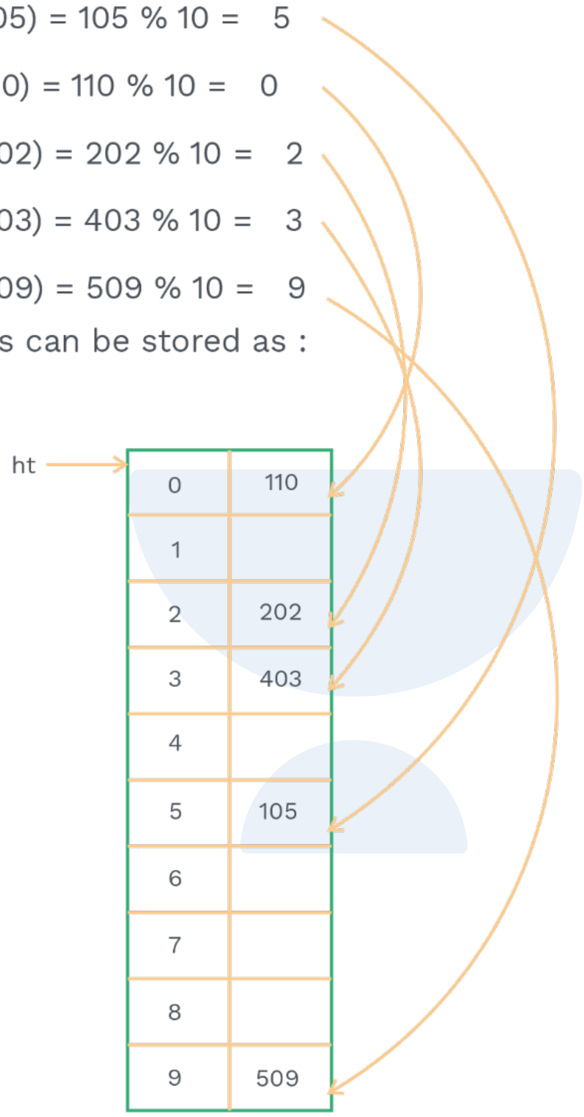
h(110) = 110 % 10 = 0

h(202) = 202 % 10 = 2

h(403) = 403 % 10 = 3

h(509) = 509 % 10 = 9

Now, keys can be stored as :



0	110
1	
2	202
3	403
4	
5	105
6	
7	
8	
9	509

Table 6.4

So, this table size is minimized by compressing the given keys.

Types of hash functions:

Different types of hash functions are:

1) Division Modulo Method

2) Digit Extraction Method

3) Mid Square Method

4) Folding Method

Rack Your Brain

A hash function is given as h(k) = K%69, where K indicates a key and h(k) indicates hash function of the key. What should be the minimum size of hash table and indices of a hash table?

Detailed discussion on each of the hash function types:

1) Division modulo method:

- Division Modulo Method is used to compress the key.
- It uses function as, h(k) = k mod (hash_table_size).
- Obtained mod value of key with hash_table_size gives the position for the particular key to be stored at.

- For example:

Hash table size is S = 1000 and hash index varies from 0 to 999.

Key = 5231119265

Now, the key is to be stored at?

Mod value for the key:

h(k) = k mod (hash_table_size)

= (5231119265) mod 1000

= 265; It is the position where the key will be stored.

Previous Years' Question

Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function x mod 10, which of the following statements are true?

i) 9679, 1989, 4199 hash to the same value

ii) 1471, 6171 hash to the same value

iii) All elements hash to the same value

iv) Each element hashes to a different value

a) i only **b)** ii only

c) i and ii only **d)** iii or iv

Sol: c) **(GATE CSE: 2004)**

like,

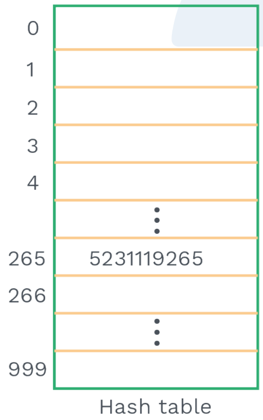


Table 6.5

- It is the easiest method to create a hash function.

2) Digit extraction method:

- Digit Extraction Method is also used to compress the key.
- In this method, selected digits are extracted from the key to be stored and used as the position where key has to be stored.
- For example:

Suppose hash table addresses are from 0 to 999, means of size 1000 if keys are of 7 digits and are as follows: 5231119,

4290386,

1296043,

5930532,

6080239,

According to the digit extraction method, we could extract digits from positions 2, 3 and 7 from the keys.

like,

5 ② ③ 1 1 1 ⑨ ⟶ 239; 5231119 will be stored at 239

4 ② ⑨ 0 3 8 ⑥ ⟶ 296; 4290386 will be stored at 296 and so on.

1 ② ⑨ 6 0 4 ③ ⟶ 293

5 ⑨ ③ 0 5 3 ② ⟶ 932

6 ⓪ ⑧ 0 2 3 ⑨ ⟶ 089

The hash table looks like this,

0	
1	
2	
	⋮
089	6080239
	⋮
239	5231119
	⋮
293	1296043
296	4290386
	⋮
932	5930532
	⋮
999	

Table 6.6

Note:

Digits can be extracted from any desired position in the keys.

3) Mid square method:

- It is used to compress the key, which is supposed to be stored in the hash table.
- In this method, we perform a square operation on the key and take the middle digits as addresses for keys.

Grey Matter Alert!

The Digit extraction method is also called Truncation Method.

- For example:

Suppose the key is 32165 and the size of the hash table is 1000.

So, h(k) = $(32165)^2$ plus take middle digits.

= 1 0 3 4 5 8 7 2 2 5

position for the key (we consider 3 digits by referring the size of the hash table).

Hash table looks like,

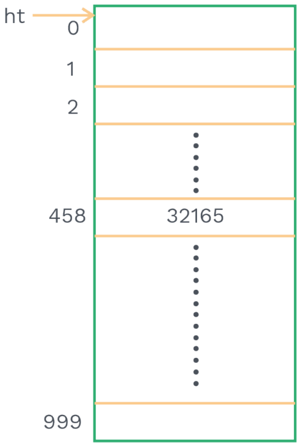


Table 6.7

4) **Folding method:**

- In this method, we select a number of digits from the key and perform addition on digits and make it concise.
- The value resulting from doing the above changes, acts as the address/position for the key to be stored in the hash table.

- For example:

Suppose the key to be stored is 321653533 and the size of the hash table is 1000.

Since table indices vary from 0 to 999, we will take the number of digits equal to 3 to fold from the key.

So, $h(k) = \underbrace{321}\underbrace{653}\underbrace{533}$

$$\Rightarrow \begin{array}{r} 321 \\ 653 \\ 533 \\ \hline \underbrace{150}\underbrace{7} \end{array}$$

$$\Rightarrow \begin{array}{r} 150 \\ 007 \\ \hline \boxed{157} \end{array}$$

This location will act as the address to store the key.

Hash table looks like,

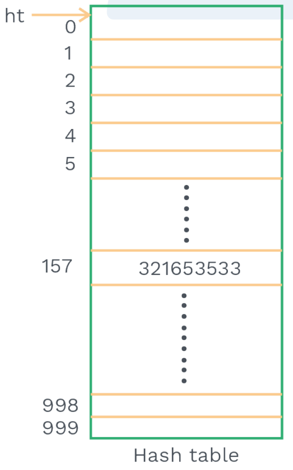


Table 6.8

Note:

The folding method and mid square method are better among all the methods. Since every digit is participating, it is difficult to find other counter keys which generate the same address.

Retrieval of keys from hash table:

Retrieval of keys means we need to find the location of the key exactly where it is stored in the hash table.

So, depending on the type of hash functions, we search the keys. Let’s check one by one:

1) **Division modulo method:**

The hash table size is 1000 (0 to 999).

The key is 5231119265 and we need to find its location. So, applying the division modulo method,

h (5231119265) = (5231119265) % (1000)

= 265; 265 is the address, for the key.

Thus, it just takes constant time to search a key.

means O(1) in Every Case.

Previous Years’ Question

Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020?

a) h(i) = i^2 mod 10

b) h(i) = i^3 mod 10

c) h(i) = $(11*i^2)$ mod 10

d) h(i) = (12*i) mod 10

Sol: b)

(GATE CSE: 2015 (Set–2))

2) **Digit extraction method:**

Hash table size is 1000 (0 to 999). Key is 5231119, and we need to find its location.

So, applying the digit extraction method,

Extract the digits from positions exactly which are used to store.

h(523119) = 5 ② ③ 1 1 1 ⑨ ⟶ 239; 239 is the address for the key.

3) **Mid square method:**

Hash table size is 1000 (0 to 999)

The key is 32165, and we need to find its location.

So, applying mid square method,

h(32165) = $(32165)^2$ plus taking middle digits

= 1 0 3 $\boxed{4\ 5\ 8}$ 7 2 2 5

458 is the address for the key.

4) **Folding method:**

Hash table size is 1000 (0 to 999)

The key is 321653533, and we need to find its location.

So, applying the folding method in the same way,

$$h(321653533) = \underbrace{321}\,\underbrace{653}\,\underbrace{533}$$

$$\Rightarrow \quad \begin{array}{r} 321\\ 653\\ 533\\ \hline \underbrace{150}\underbrace{7} \end{array}$$

$$\Rightarrow \quad \begin{array}{r} 150\\ 007\\ \hline \boxed{157} \end{array}$$; is the address for the key.

Note:

For retrieval of keys by using any of the above methods, constant time is needed i.e. O(1) Every Case.

Load factor and collisions:

Load factor:

- The number of keys per slot of the hash table is called the load factor.
- Load factor is denoted by ‘μ’.
- Suppose the number of slots in the hash table equals to N and the number of keys to be stored in the hash table is x.
- ∴ Number of keys per slot = $\frac{x}{N}$

This means the average number of keys per slot = $\frac{x}{N}$

$$\therefore \qquad \boxed{\mu = \frac{x}{N}}$$

Previous Years’ Question

Consider the following two statements:

i) A hash function (these are often used for computing digital signatures) is an injective function.

ii) encryption technique such as DES performs a permutation on the elements of its input alphabet.

Which one of the following options is valid for the above two statements?

a) Both are false

b) Statement (i) is true, and the other is false

c) Statement (ii) is true, and the other is false

d) Both are true

Sol: c) **(GATE CSE: 2007)**

SOLVED EXAMPLES

Q1 **Given a hash table with 39 slots that store 250 keys. Calculate the load factor ‘μ’. (up to two decimal places)**

Sol: **6.41**

Given,

Number of slots = 39
Number of keys = 250

$$\therefore \mu = \frac{\text{Number of keys}}{\text{Number of slots}} = \frac{250}{39} = 6.41 \text{ Ans.}$$

Collision:

- After applying the hash function on keys to map them to some address of the hash table, if more than one key are mapped to the same address of the hash table, then this phenomenon results in a collision.
- In simple words, if one slot of hash table is getting more than one key to be stored at it, it is called a collision.
- Let’s understand it with an example:

Keys are given as: 49, 26, 32, 59, 60, 29, 41

Hash table size is given as: 10 (0 to 9)

Hash function is given as: $\boxed{h(K) = K \bmod 10}$

Previous Years’ Question

Consider a hash function that distributes keys uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5.

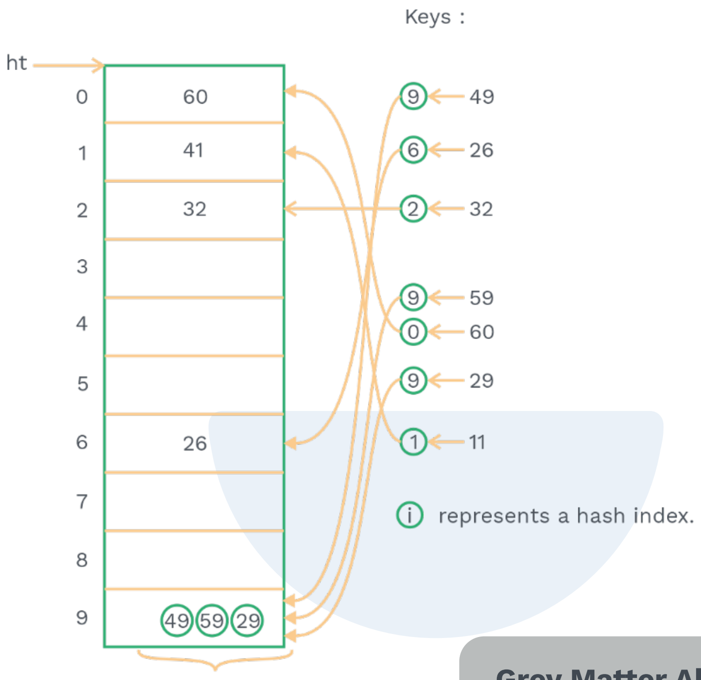
a) 5 **b)** 6
c) 7 **d)** 10

Sol: d) **(GATE CSE: 2007)**

Calculating hash indices for each key,

1) h(49) = 49 mod 10 = 9; this key will go to hash index 9 and so on.
2) h(26) = 26 mod 10 = 6
3) h(32) = 32 mod 10 = 2
4) h(59) = 59 mod 10 = 9
5) h(60) = 60 mod 10 = 0
6) h(29) = 29 mod 10 = 9
7) h(41) = 41 mod 10 = 1

Mapping each key to its location in the hash table,



Three keys are mapped
to same hash index,
it is called a collision.

Grey Matter Alert!

Hash collision is also termed a hash clash.

Table 6.9

Q2 **There is a hash table of size 10. Keys are given as 539, 510, 201, 202, 503, 405, 406, 911, 922, 906. Find if there is any collision present in the system and if yes, find the number of collisions encountered. The division modulo method is used as the hash function.**

Sol: **Yes, 3 collisions.**

Calculating hash indices for each key:

Given the hash function used is the division modulo method.

$$\text{So,} \quad \underset{\substack{\uparrow \\ \text{hash of key}}}{h(k)} = \underset{\substack{\uparrow \\ \text{key}}}{k} \bmod \underset{\substack{\uparrow \\ \text{table size}}}{10}$$

h(539) = 539 mod 10 = 9

h(510) = 510 mod 10 = 0

h(201) = 201 mod 10 = 1

h(202) = 202 mod 10 = 2

h(503) = 503 mod 10 = 3

h(405) = 405 mod 10 = 5

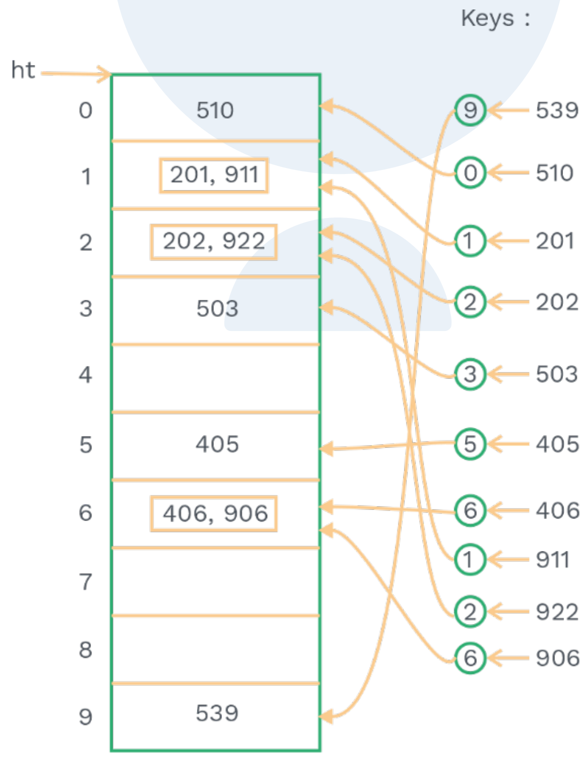
h(406) = 406 mod 10 = 6

h(911) = 911 mod 10 = 1

h(922) = 922 mod 10 = 2

h(906) = 906 mod 10 = 6

Mapping each key to its location in hash table,



i) Yes, At index no. 1, 2, 6, collisions are present.

ii) Calculating no. of collisions:

Index 1 → One collision

Index 2 → One collision

Index 6 → One collision

∴ Total no. of collisions = 3 Ans.

COLLISION RESOLUTION TECHNIQUES

- Ideally, no two keys should be compressed into the same integer. Unfortunately, such an ideal technique doesn't exist.
  So, here we will discuss some techniques which will resolve when collisions are resulted, and leads us close to ideal.

- Basically, there are two techniques of dealing with hash collisions.
  1) Chaining
  2) Open addressing

1) Chaining:

- Chaining is a collision resolution technique.
- This technique resolves collision by building a linked list of all keys hashed to the same hash index.
- Hash table is in the form of the array only, but it is the array of pointers in this case.
- Keys are stored in nodes of a linked list.
- At a specific hash index, one address is stored that is of the first node of the chain in the form of linked list.
- Applying chaining for some keys like:

Example: Keys are given as 49, 26, 32, 59, 69, 79, 60, 50, 30, 41, 61, 29

Table size is given as 10 (0 to 9).

Hash function, h(k) = k mod 10.

The collision resolution technique is chaining.

Calculating hash indices for each key,
h(49) = 49 mod 10 = 9

h(26) = 26 mod 10 = 6

h(32) = 32 mod 10 = 2

h(59) = 59 mod 10 = 9

h(69) = 69 mod 10 = 9

h(79) = 79 mod 10 = 9

h(60) = 60 mod 10 = 0

h(50) = 50 mod 10 = 0

h(30) = 30 mod 10 = 0

h(41) = 41 mod 10 = 1

h(61) = 61 mod 10 = 1

h(29) = 29 mod 10 = 9

Mapping each key to its location in the hash table,

49→⑨,26→⑥,32→②,59→⑨,69→⑨,79→⑨,60→⓪,50→⓪,30→⓪,41 →①,61 →①,29→⑨

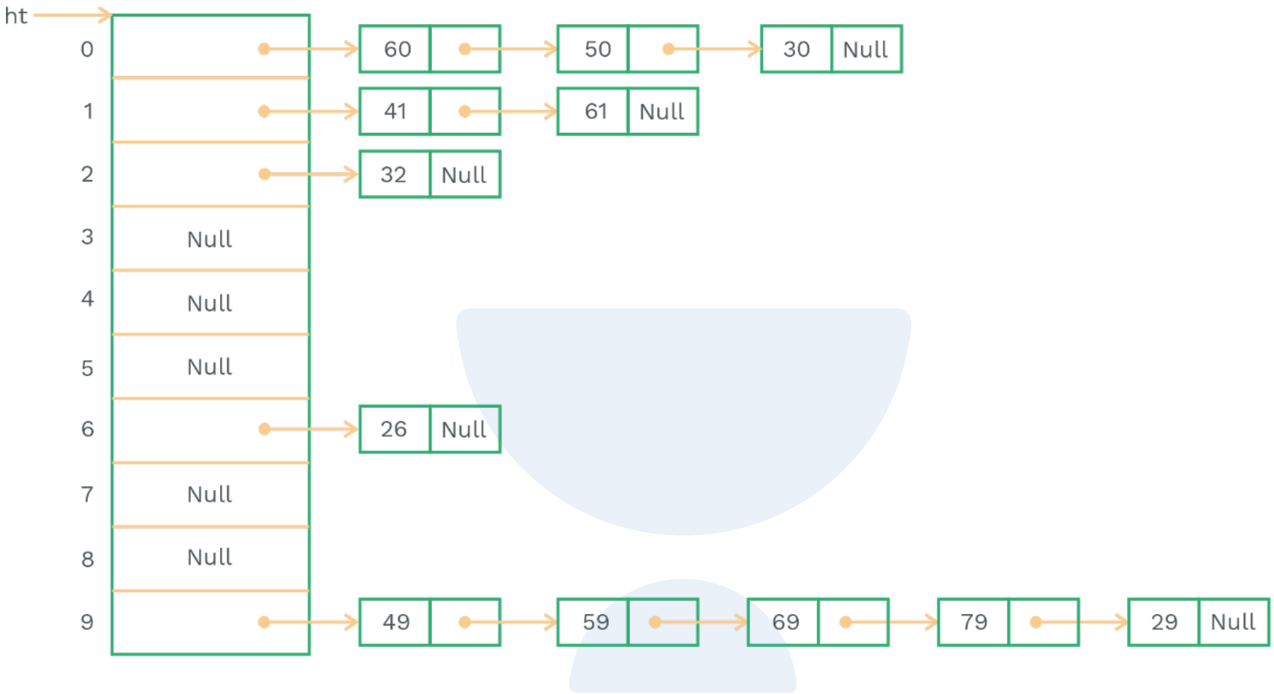


Table 6.10

- Array positions which don't have any node to point to are set to Null pointer.

Advantages of chaining:

- By applying chaining, a hash index can handle many keys in it. We can deal with the infinite number of collisions.
- Any insertion of a key in hash table will take constant time, i.e. O(1) every case.
- Deletion in chaining is easy because we store keys in an unsorted manner, so deletion of one key doesn't create trouble for other keys.

Disadvantages of chaining:

- Deletion and searching in chaining takes O(n) time complexity.
- Consider the previous instance,

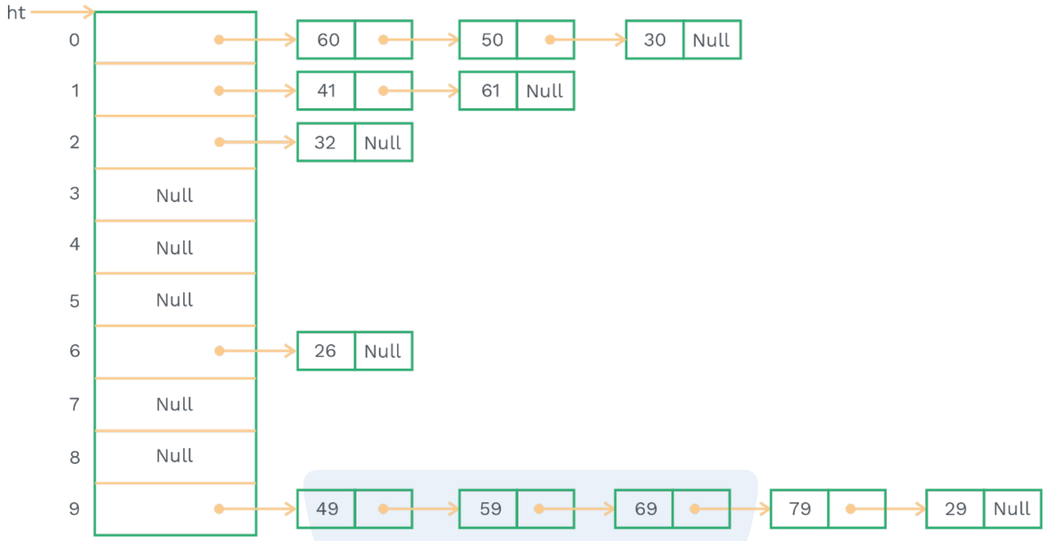


Table 6.11

we can analyse clearly that even though many slots are empty but we can't store keys in them, and forcefully, we need to take space outside the array. Chaining has a drawback in terms of space consumption.

Load factor:

Load factor for chaining; $\mu \geq 0$

It means keys per slot is 0 or any positive integer.
One slot can point to 0–key, 1–key or more than one key.

SOLVED EXAMPLES

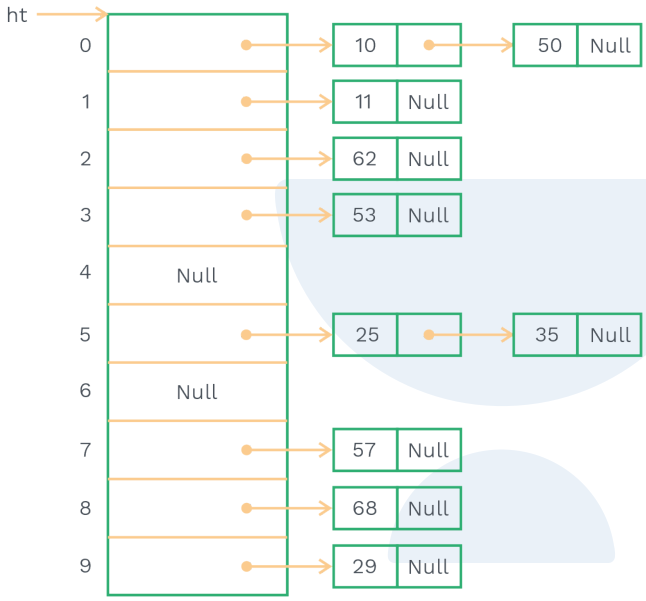
Q3 **Consider a hash table with 10 slots. The hash function used is h(k) = k mod 10. The collisions are resolved by using chaining. Keys 25, 68, 29, 35, 10, 53, 62, 57, 50, 11 are inserted in the given order. The maximum, minimum and average chain lengths for the hash table, respectively are: ___________**

Sol: **2, 0, 1.**

Calculating hash indices for each key.
h(25) = 25 mod 10 = 5
h(68) = 68 mod 10 = 8
h(29) = 29 mod 10 = 9
h(35) = 35 mod 10 = 5

h(10) = 10 mod 10 = 0
h(53) = 53 mod 10 = 3
h(62) = 62 mod 10 = 2
h(57) = 57 mod 10 = 7
h(50) = 50 mod 10 = 0
h(11) = 11 mod 10 = 1

Applying Chaining,



Index no.	Chain length
0	2
1	1
2	1
3	1
4	0
5	2
6	0
7	1
8	1
9	1

∴ Maximum chain length = 2

Minimum chain length = 0

Average chain length = $\frac{2+1+1+1+0+2+0+1+1+1}{10} = \frac{10}{10} = 1$

So, 2, 0, 1 Ans.

Q4 **Consider a hash table with 10 slots which uses chaining for collision resolution. Assuming the table is initially empty and considering simple uniform hashing. What would be the probability that after 3 keys are inserted, a chain of size at least 2 results?**

Sol: **0.28**

- We have given Table size = 10
- Collision resolution technique – chaining
- Table is empty initially
- Simple uniform hashing
- 3 keys are inserted
- Simple uniform hashing – It evenly distributes elements into the slots of a hash table.

Probability of a chain of size atleast 2
= [Probability of a chain of size exactly 2]
+ [Probability of a chain of size exactly 3]

$$= 10 \times \left[{}^3C_2 \times \frac{1}{10} \times \frac{1}{10} \times \frac{9}{10} \right] + 10 \times \left[{}^3C_3 \times \frac{1}{10} \times \frac{1}{10} \times \frac{1}{10} \right]$$

$$= \left[\underset{\uparrow}{{}^3C_2} \times \underset{\uparrow}{\frac{1}{10}} \times \frac{1}{10} \times \underset{\uparrow}{\frac{9}{10}} \right] \underset{\uparrow}{\times 10} + \left[\underset{\uparrow}{{}^3C_3} \times \frac{1}{10} \times \frac{1}{10} \times \frac{1}{10} \right] \times 10$$

Selecting 2 elements from 3 inserted elements

for every 10 slots

Selecting all 3 inserted elements

Probability of going an element to a slot

Probability of going 3rd element to any of other 9 slots.

$$= 3 \times \frac{1}{10} \times \frac{1}{10} \times \frac{9}{10} \times 10 + 1 \times \frac{1}{10} \times \frac{1}{10} \times \frac{1}{10} \times 10$$

$$= \frac{3 \times 9}{10 \times 10} + \frac{1}{10 \times 10}$$

$$= \frac{27}{100} + \frac{1}{100}$$

$$= \frac{28}{100}$$

= 0.28 Ans.

Previous Years' Question

An advantage of a chained hash table (external hashing) over the open addressing scheme is

a) Worst-case complexity of search operations are less

b) Space used is less

c) Deletion is easier

d) None of the above

Sol: c) **(GATE CSE: 1996)**

2) Open addressing:

- Open addressing is also a collision resolution technique like chaining.
- Only a hash table is used to store keys, unlike chaining.
- In open addressing, at any point, the size of the table must be greater than or equal to the total number of keys.

- There are three types of open addressing, considering the way the collisions are resolved:

  i) Linear Probing

  ii) Quadratic Probing

  iii) Double Hashing

Let's understand each of the above three types of open addressing one by one:

i) Linear probing:

- In this method, if some collision arises, then to resolve it, we probe next slot to store the key linear.
- Linear_Probing(k, i) = (h(k) + i) mod S

$$\forall\, i \in \{0, 1, 2, 3, \ldots\ldots, S-1\}$$

Where k is a key
h(k) is the hash of key 'k'.
i is iteration number or collision number.
S is the size of the hash table.

- When any collision is encountered for some key, the above definition is applied to resolve the collision according to Linear Probing.
- Let’s understand linear probing with an example:

Keys are given as: 49, 26, 36, 32, 59, 60, 70, 80.
Hash table size given as: 10 (0 to 9)
Hash function used is: $\boxed{h(k) = k \bmod 10}$
Linear probing (k, i): (h(k) + i) mod S

Insert keys one by one into the hash table.
Initially, table is empty:

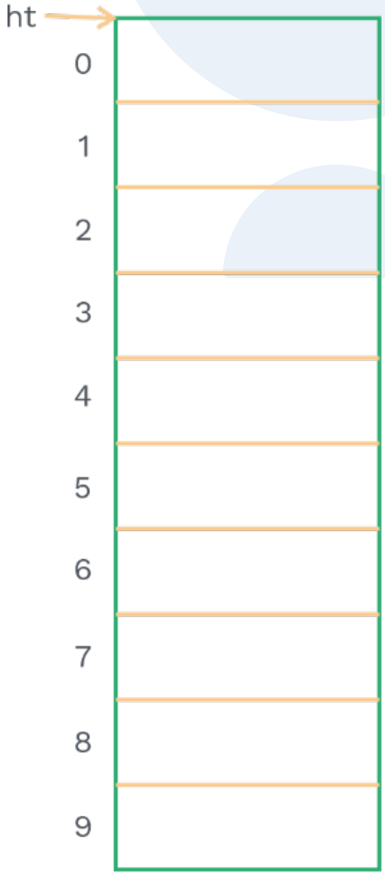


Table 6.12

1) h(49) = 49 mod 10 = 9 ✓

2) h(26) = 26 mod 10 = 6 ✓

3) h(36) = 36 mod 10 = 6 ⟶ 1–Collision

As we can see, key–36 is hashed to position '6', but we can't store key–36 to position '6' because one key is already present to position '6'.

ht	
0	59
1	60
2	32
3	70
4	80
5	
6	26
7	36
8	
9	49

Table 6.13

So, applying linear probing,
h(k) = 36 mod 10 = 6
Linear probing (k, i) = (h(k) + i) mod S

$$= (6 + \underset{\substack{\uparrow \\ i\,=\,1 \text{ in first iteration and so on.}}}{1}) \bmod S$$

$= 7 \bmod 10 = 7$ ✓

Now, key–36 will go to position '7'.

4) h(32) = 32 mod 10 = 2 ✓

5) h(59) = 59 mod 10 = 9 ⟶ 1–Collision

Linear probing (k, 1) = (9 + 1) mod 10
= 10 mod 10 = 0 ✓

6) h(60) = 60 mod 10 = 0 ⟶ 1–Collision

Linear probing (k, 1) = (0 + 1) mod 10
= 1 mod 10 = 1 ✓

7) h(70) = 70 mod 10 = 0 → 1 – Collision
Linear Probing (k, 1) = (0 + 1) mod 10

= 1 mod 10 = 1 →1 – Collision again
Linear probing (k, 2) = (0 + 2) mod 10
= 2 mod 10 = 2 →1 – Collision again
Linear probing (k, 3) = (0 + 3) mod 10

= 3 mod 10 = 3 ✓

8) h(80) = 80 mod 10 = 0 →1 – Collision
Linear probing (k, 1) = (0 + 1) mod 10
= 1 mod 10 = 1 →1 – Collision again
Linear probing (k, 2) = (0 + 2) mod 10
= 2 mod 10 = 2 →1 – Collision again
Linear probing (k, 3) = (0 + 3) mod 10
= 3 mod 10 = 3 →1 – Collision again
Linear probing (k, 4) = (0 + 4) mod 10

= 4 mod 10 = 4 ✓

Now, all keys are stored by following linear probing.

Advantages of linear probing:
The main advantage of linear probing is, the each position/the hash index of the hash table can be used to store keys.

Disadvantages of linear probing:

i) While probing, each slot is being checked to get to store some key at it, so it will take more time by probing slowly.

ii) In linear probing, we can probe only in the manner of one slot after another.

iii) Deletion is difficult because once we delete some key, it creates trouble for other keys while any search is performed.

Time complexity:

i)	Searching	:	O(1) [Best case]
			O(S) [Worst case], [Average Case]
ii)	Insertion	:	O(1) [Best case]
			O(S) [Worst case], [Average Case]
iii)	Deletion	:	O(1) [Best case]
			O(S) [Worst case], [Average Case]

Primary clustering:

- In case two keys have the same start of the hash index, then while performing linear probing to resolve the collision, both will follow the same path in a linear manner, which is unnecessary. It is called primary clustering.

- Because of primary clustering, the average searching time increases.
- Linear probing is suffering from primary clustering.

Grey Matter Alert!

Linear probing is also called closed hashing.

Q5 **Consider a hash table with 8 buckets which use linear probing to resolve collisions. The key values are integers, and hash function used is key mod 8. If the keys 55, 133, 50, 123, 62 are inserted in the hash table in order then what is the index of the key 62?**

Sol: 0^{th} position of the hash table.

Given, Number of buckets/slots = 8
Means hash table size is 8. (0 to 7)
Hash function = key% 8, (% = mod)
Collision resolution technique = Linear probing.

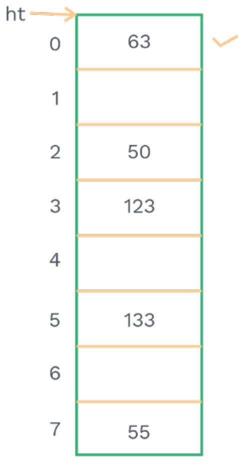
Mapping keys to their location one by one:

1) h(55) = 55% 8 = 7 **2)** h(133) = 133% 8 = 5
3) h(50) = 50% 8 = 2 **4)** h(123) = 123% 8 = 3
5) h(63) = 63% 8 = 7 – 1 collision

Applying linear probing,

$$h(63) = 63\%\ 8 = 7$$

Linear probing (k, i) = (h(k) + i) mod 8
⇒ Linear probing (k, 1) = (7 + 1) mod 8 = 8 mod 8 = 0
Resulting hash table is:



So, key–63 would be inserted at location ‘0’ of the hash table.

2) Quadratic probing:

- In this method, if some collision arises, then to resolve the collision, we probe the next slot to store the key in a quadratic manner.
- Quadratic_probing(k, i) = (h(k) + i^2) mod S

  $\forall$ i $\in$ {0, 1, 2, 3, ..., S–1}

Where k is a key.
h(k) is the hash of key ‘k’.
i is iteration number or collision number.
S is the size of the hash table.

- When any collision is encountered for some key, the above definition is applied to resolve the collision according to quadratic probing.
- In quadratic probing, we probe one slot after another in quadratic manner.

Time complexity:

i) Searching : O(1) [Best case]
O(S) [Worst case], [Average Case]

ii) Insertion : O(1) [Best case]
O(S) [Worst case], [Average Case]

iii) Deletion : O(1) [Best case]
O(S) [Worst case], [Average Case]

- Quadratic probing faces the problem of secondary clustering.

Secondary clustering:

- In case two keys have the same start of the hash index of the hash table while performing quadratic probing to resolve the collision, both will follow the same path in quadratic manner, which is unnecessary. It is called secondary clustering.
- Because of secondary clustering average searching time for a key increases.

Previous Years’ Question

Consider a hash table of size seven, with starting index zero, and a hash function (3x + 4) mod 7. Assuming the hash table is initially empty, which of the following are the contents of the table when the sequence 1, 3, 8, 10 is inserted into the table using closed hashing? Note that ‘_’ denotes an empty location in the table.

a) 8, _, _, _, _, _ , 10
b) 1, 8, 10, _, _, _, 3
c) 1, _, _, _, _, _, 3
d) 1, 10, 8, _, _, _, 3

Sol: b) **(GATE CSE: 2007)**

Previous Years’ Question

Consider a hash table of size 11 that uses open addressing with linear probing. Let h(k) = k mod 11 be the hash function used. A sequence of records with keys

43 36 92 87 11 4 71 13 14 is inserted into an initially empty hash table, the bins of which are indexed from zero to ten. What is the index of the bin into which the last record is inserted?

a) 2 **b)** 4
c) 6 **d)** 7

Sol: d) **(GATE CSE: 2008)**

3) Double hashing:

- In this method, if some collision arises then to resolve the collision, a second hash function to key is applied.
- Double_hashing(k, i) = (hash_1(k) + i × hash_2(k)) mod S
  $\forall$ i $\in$ {0, 1, 2, 3, ..., S–1}

  Where k is a key.
  hash 1 (k) is first hash function which is applied to the key.
  hash 2 (k) is second hash function which is applied to the key.
  i is the iteration number or collision number.
  S is the size of the hash table.
- First hash function is: hash1 (k) = k mod S.
- Second hash function can be taken in many ways by keeping in mind the goodness of the hash function.
- Good second hash function must never evaluate to zero and must make sure all cells should be covered while probing.
- When any collision is encountered for some key, the above definition is applied to resolve the collisions according to the double hashing definition.
- In double hashing, we can probe every slot of the hash table for some key randomly, but every slot should be covered.
- Double_hashing doesn't have the problem of primary clustering or secondary clustering.

Comparison between open addressing and chaining:

i) Open addressing scheme is complex in consideration of computational work.	**i)** Chaining is simple.
ii) All the keys are stored only in the table.	**ii)** We need extra space outside the hash table to store the keys.
iii) In open addressing, a key can be sent to a slot which is not mapped for the key.	**iii)** When keys are not mapped to a slot, then the slot space is wasted.
iv) Open addressing schemes have problems like primary clustering and secondary clustering.	**iv)** There is nothing like clustering in chaining.

v) Hash table may become full.	**v)** In chaining, hash table never fill up, we can only add keys to chain.
vi) It provides better cache performance.	**vi)** Cache performing in chaining is not good.

- Hashing is a searching technique.
- Basically, the motive behind hashing is to retrieve each key stored in hash table in single access.
- Worst case searching time in case of hashing is expected O(1) [Every Case].
- Hash table is used to maintain keys in it.

Chapter Summary

- A function which transforms a key into a table index is called as hash function.
- **Types of hash functions:**
  1) Division modulo method
  2) Digit extraction method
  3) Mid square method
  4) Folding method
- **Load Factor:** Number of keys per slot of the hash table is called load factor. $(\mu); \mu = \frac{x}{N}$, x: Number of keys, N: Number of slots.
- **Collision:** In simple words, if one slot of hash table is getting more than one key to be stored in it, is called as collision.
- **Collision Resolution Techniques:**
  Techniques which can resolve collision, basically are of two kinds.
  i) Chaining:
  This technique resolves collision by building a linked list of all keys hashed to the same hash index.
  ii) Open addressing:
  Open addressing scheme works in three ways.
  a) Linear Probing: Probes the slots lineary to store the keys.
  b) Quadratic Probing: Probes the slots quadratically to store the keys.
  c) Double Hashing: Covers all the slots randomly to store a key and so on.