BYJU'S EXAM PREP

Vision 2023

A Course for GATE & PSUs

Computer Science Engineering

Algorithm

CHAPTER 2

Hashing

CHAPTER 2 ALGORITHM

HASHING

Hash Table is a data structure that stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access to data becomes very fast if we know the index of the desired data.

In data structures,

- Hashing is a well-known technique to search for any particular element among several elements.
- It minimizes the number of comparisons while performing the search.

Advantage-

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 Mechanism-

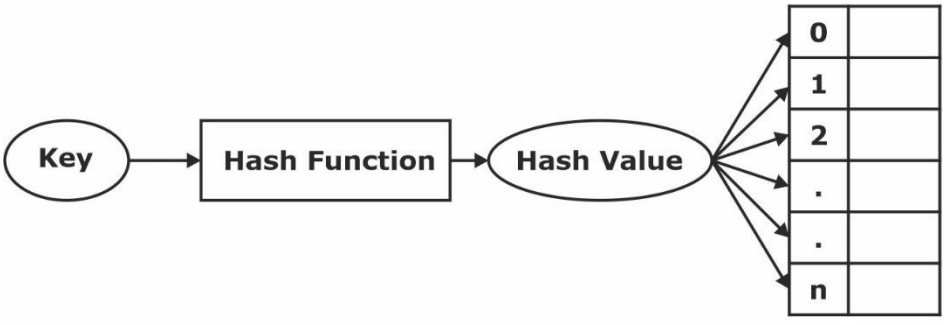
In hashing,

- An array data structure called a **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 stored in the hash table.
- Hash key value is generated using a hash function.



Hashing Mechanism

HASH FUNCTION

The 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 a hash value.
- Hash value of the data item is then used as an index for storing it into the hash table.

Types of Hash Functions-

There are various types of hash functions available such as-

1. Mid Square Hash Function
2. Division Hash Function
3. Folding Hash Function etc

It depends on the user which hash function they want to use.

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.

Clustering

Primary clustering:

The tend is for long sequences of preoccupied positions still become longer, primarily at one place.

Secondary clustering:

The tend is for long sequence of preoccupied position still become longer primarily at different places.

Collision in Hashing-

When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a **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.

Collision Resolution Techniques-

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

Collision resolution techniques are classified as-

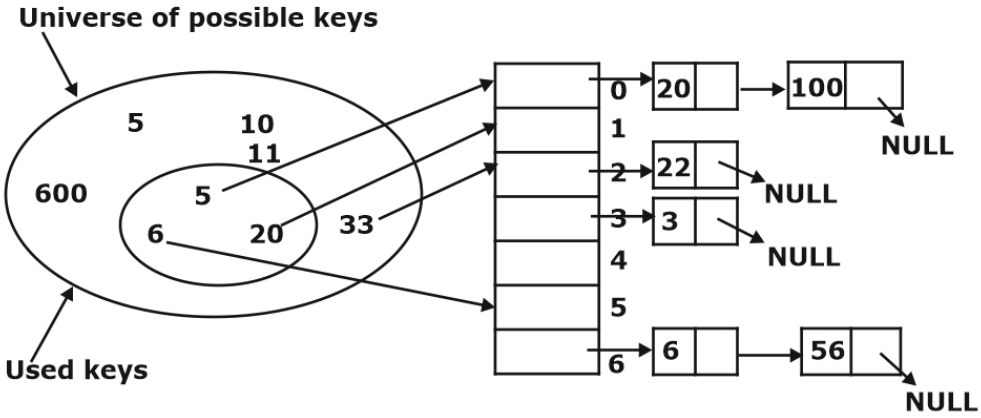
1. Separate Chaining
2. Open Addressing

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**.

Collision resolution by chaining combines linkes representation with a hash table. When two or more records have the same location, these records are constituted into a singly-linked list called a chain.



Time Complexity-

For Searching-

- In the worst case, all the keys might map to the same bucket of the hash table.
- In such a case, all the keys will be present in a single linked list.
- Sequential search will have to be performed on the linked list to perform the search.
- So, time taken for searching in the worst case is O(n).

For Deletion-

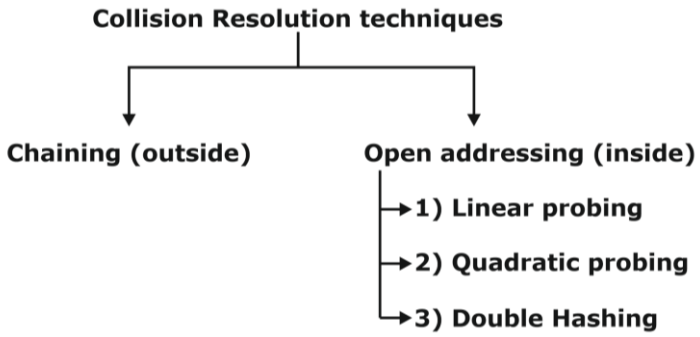
- In the worst case, the key might have to be searched first and then deleted.
- In the worst case, time taken for searching is O(n).
- So, time taken for deletion in the worst case is O(n).

Load Factor (α)-

Load factor (α) is defined as-

$$\text{Load Factor } (\alpha) = \frac{\text{Number of elements present in the hash table}}{\text{Total size of the hash table}}$$

If Load factor (α) = constant, then the time complexity of Insert, Search, Delete = Θ(1)



Example-

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

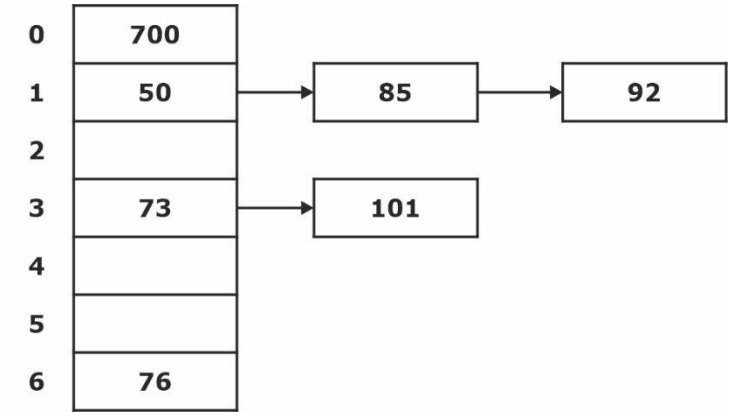
Use a separate chaining technique for collision resolution.

Solution-

The given sequence of keys will be inserted in the hash table as-

- 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.
- 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 inserted in bucket-1 of the hash table.

Similarly, we insert all the keys. The final hash table looks like-



Open addressing vs closed addressing:

Disadvantages of open addressing:

1. Collided records required more probes.
2. Deletion is not possible.
3. Overflow problem

Advantages of chaining:

1. Collided records required less probes.
2. Deletion possible
3. No overflow problem

To avoid this problem we will use double hashing :-

T.C.

insertion	searching	deletion
B.C.→ O(1)	B.C. → O(1)	B.C. → O(1)
W.C. → O(m)	W.C → O(m)	W.C → O(m)

Double hashing:-

m = 10(0, …a)

$H.F_1$ (key) = key mod m;

$H.F_2$ (key) = 1+(key mod m-2);

$$D11(key, 1) = (H_1F_1(key) + i * H.F_2 (key)) \bmod m$$

i = 0, 1, …. 9(m – 1)

Eg:- Key(n)= 25, 98, 57, 75, 97, 18, 78

Total collisions → 7

0	
1	**97**
2	**78**
3	
4	**18**
5	**25**
6	
7	**57**
8	**98**
9	**75**

Hash Table

25 $H.F_1(25) = 25 \bmod = 5$

$HF_2 (25) = 1 + (25 \bmod 8) = 2$

D.H. (key, i) = D.H (25, 0) = 5 mod 10 = 5

98 $H.F_1$ (98) = 8

$H.F_2$ (98) = 1 + (98 mod 8) = 3

DH (98, 0) = 8

$H.F_1$(57) = 7

$H.F_2$ (57) = 1 + 1 = 2

D.H = (75, 0) = 5 + 0 = 5 → collision

(75, 1) = 5 + 4 = 9

$H.F_1$ = 5

$H.F_2$ = 4

(97, 0) = 7 → collision (97, 1) = 9 → Collision

(18, 0) = 8 → collision

(8, 0) $H.F_1$ = 8

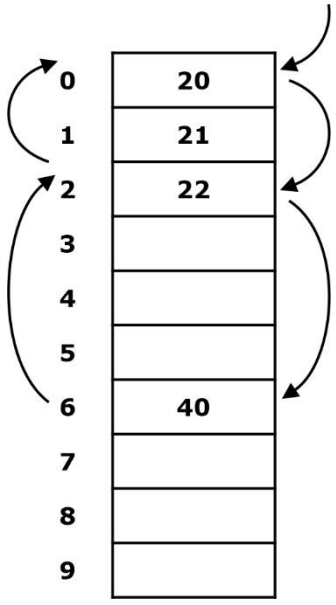
$H.F_2$ = 1 + 2 = 3 8 + 3 = 1 mod 10 = 1

1 = 2 8 + 6 = 14 mod 10 → 4

78 → 8 → collision

(78, 1) = 8 + 7 → 15 → collision

1 + 78 mods (78, 2) = 8 + 14 = 22 → 2



Eg:- Keys (n) = 20, 21, 22, 40, 50

QP(20, 0) = 20 + 0 + 0 = 0

QP(21, 0) = 21 + 0 + 0 = 1

QP(22, 0) = 22 + 0 + 0 = 2

QP(40, 0) = 0 → Collision

40 + 1 + 1 = 42 → 2 → collision

40 + 2 + 4 = 46 → 6 →

QP(50, 0) = 0 → Collision
(50, 1) = 50 + 1 + 1 = 52 → 2 → Collision
(50, 2) = 50 + 6 → 6 → collision
(50, 3) = (50 + 3 + 9) = 2 → collision
(50, 4) = (50 + 4 + 16) → 0 → Collision
= 50 + 2 + 25 → 0

Secondary clustering:-

If the two keys are mapped onto the same starting location in the hash table then they both follow the same path unnecessarily in the quadratic manner. because of this search time complexity will increase.

Time Complexity

searching	insertion	deletion
B.C.→ O(1)	B.C. → O(1)	B.C. → O(1)
W.C. → O(m)	W.C → O(m)	W.C → O(m)

Conclusion :- if n ≤ m: then by using perfect hashing we can achieve worst call search time complexity as O(1). (99%)

NOTE:

1) The Expected no. of probe's in an unsuccessful search of open addressing technique is

$\frac{1}{1-\alpha}$ where α is load factor $\alpha = \frac{n}{m}$

2) the Expected no. of probe's in an successful search of open addressing technique is

$\frac{1}{\alpha}\log\frac{1}{1-\alpha}$ where , α , Load factor $\boxed{\alpha = \frac{n}{m}}$

= 8 + 1 = 9 → Collision
= 8 + 2 = 10 → 0 → collision
= 8 + 3 = 11 → 1 →
LP(65, 0) = 5 + 0 = 5 collision
(65, 1) = 5 + 1 = 6 → collision
(65, 2) = 5 + 2 = 7 → 0
(20, 0) = 0 + 0 → 0 → collision

0 + 1 → 1 → collision

0 + 2 → 2 →

Load factor :- (α)

No. of keys getting stored in one slot is called as load factor.

In M slots — we are storing n keys.

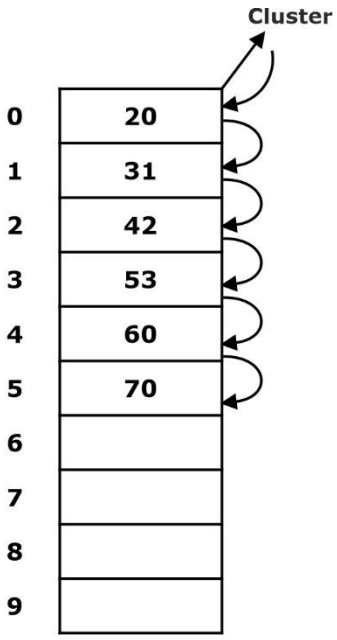
In 1 slot — ?

$$\frac{1}{m} * n = \frac{n}{m}$$

$$\alpha = \frac{n}{m}$$

Eg:-

Keys(n) = 20, 31, 42, 53, 60, 70



L.P. = (60 + 0) → 0

(60 + 1) = 1

(60 + 2) = 2

(60 + 3) = 3

(60 + 4) = 4

(70, 0) = 0

(70, 1) = 1

(70, 2) = 2

(70, 3) = 3

(70, 4) = 4

(70, 5) = 5

Primary clustering:

1) If the two keys are mapped onto the same starting location in the hash table then they both follow the same path unnecessarily in the linear manner because of this search time complexity will increase.

c) To avoid this problem quadratic propping is used.

insertion	**searching**	**deletion**
B.C.→ O(1)	B.C. → O(1)	B.C. → O(1)
W.C. → O(m)	W.C → O(m)	W.C → O(m)

Eg:- Keys (n) = 25, 98, 57, 75, 97, 18

0	**18**
1	**75**
2	
3	
4	
5	**25**
6	
7	**57**
8	**98**
9	**75**

Hash Table

(n)

1) QP (25, 0) = 5 + 0 + 0 = 5
2) QP(98, 0) = 8 + 0 + 0 = 8
3) QP(57, 0) = 7 + 0 + 0 = 7
4) QP(75, 0) = 5 + 0 + 0 → 5 → Collision
   (75, 1) = 5 + 1 + 1 = 7 → collision
   (75, 2) = 5 + 6 = 11% 10 → 1
5) QP(97, 0) = 97 + 0 + 0 = 97 → 7 → collision
   97 + 1 + 1 = 9 →
6) QP(18, 0) = 18 + 0 + 0 → 8 → collision
   18 + 1 + 1 + 20%10 → 0
   4 → collision → total

Note: Delection will be problem to others keys but we can manage by storing the special symbol use $, or #.

If more no. of deletions occur perform rehashing

Quadratic probing :-

m = 10 (0 …. 9)

H.F (key) = key mod m;

Q.P(key, i) = (M.F(key) + C_1*i + C_2 *i^2) nodn

C_1 = 1, C_2 = 1, i = 0, 1……9(m – 1)

Eg:- key(n) = 25, 38, 43, 68, 79, 46, 58, 65, 20

0	**79**
1	**58**
2	**20**
3	**43**
4	
5	**25**
6	**46**
7	**65**
8	**38**
9	**68**

Hash Table (n)

1) LP(25, 0) = (H.F.(25 + 0) mod 10) = 5 + 0
2) LP(38, 0) = (8 + 1) = 8
3) LP (43, 0) = (3 + 0) = 3
   LP(68, 1) = (8 + 1) = 9
5) LP(79, 0) = (9 + 0) = 9 → collision
   LP(79, 1) = (9 + 1) = 0 → 0
6) LP(46, 0) = (6 + 0) = 6
7) LP(58, 0) = 8 + 0 = 8 → collision

Chaining:

Chaining is implemented with the help of linked list.

2) keys will be stored outside the hash table.

H.F. (key) = key mod n

n = 10 (0…..9)

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

****