Assignment No - 01 (Group A)

Problem statement: Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone Number. Make Use of two Collision Techniques and Compare them using number of Comparisons required to find a set of telephone numbers

Pre-requisite

- Knowledge of concept of Hashing
- Knowledge of Collision Resolution Techniques
- Knowledge of Python Programing

Objective

1. To analyze advanced data structure like hash table.
2. To understand collision handling techniques.
3. To understand practical implementation of hash table

Input

- Client record: Name and Telephone number

Output

- Hash table with all clients information
- Print number of comparisons required to find a telephone number

Software and Hardware requirements:-

1. **Operating system:** Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10,
2. **RAM-** 2GB RAM (4GB preferable)
3. You have to install **Python3** or higher version

Theory-

What is Hashing?

Hashing is the technique of mapping a large chunk of data into small tables using a hash function.

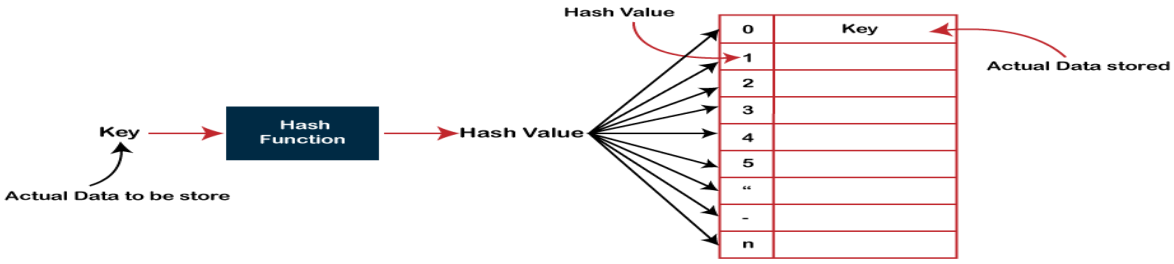
Hash Table

A hash table is a data structure that stores records in an array, called a hash table.

It is data structure where data elements are stored based on their hash key values generated by hash function.

On these stored data elements operation like insertion, deletion and search can be performed with constant time complexity. This Hash key value is generated for any data elements by using hash function.

In short we use Hash function to generate key values and using these key value data elements is stored at particular position.



Operations of Hash Table

1. **Insertion** – this Operation is used to add an element to the hash table
2. **Searching –** this Operation is used to search for elements in the hash table using the key
3. **Deleting –** this Operation is used to delete elements from the hash table

Hash Function

It is function which can be used to map large data values into small key values which are used to store data in hash table.

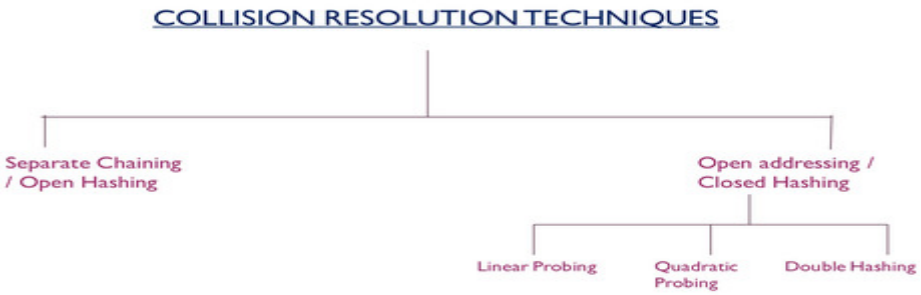
Each element in hash table is assigned with a key.

By using that key you can access the element in O(1) time. Using the key the algorithm (hash function) computes an index.

Collision resolution strategies:-

Writing a perfect functions which avoids collision is extremely difficult. Instead, we can choose a hash function which minimizes collision & whenever collision happens we can use different collision resolution strategies.

Following are collision resolution techniques we can use



1. Linear Probing

The idea behind Linear Probing simple we a fixed sized hash table & whenever collision occurs we traverse linearly / Sequential in cyclic manner to find next empty slot.

It mean when we want to insert key & if a key is mapped (hashed) to an index which is already occupied then check for next available position to place them. i.e.

H(key) = h, h(k)+1,h(k)+2,h(k)+3,…….,h(k)+I

Example-

let us consider a simples= hash function as key % table Size which means,

Hash(key)=key % S where s→ size of the table.

Let us consider s=7 & index value in table are form 0 to 6.

Now if we want to insert following value in table (50,700,76,85,92,73,101)

Index / hash key is calculated for every key as follows

50 % 7 = 1

700 %7 = 0

76 % 7 = 6

85 % 7 =1

92% 7 =1

73 %7=3

101 % 7 =3

1) When we try to insert key ‘50’ it is inserted at index position 1.

2) Similarly key ‘700’ is inserted at index position 0.

3) Similarly key ‘76’ is inserted at index position 6 as shown in the hash table step 3.

4) When we try to insert ‘85’ since position calculated for ‘85’ is 1 collision occurs as ‘50’ is already present at position 1 so we store ‘85’ at next empty position i.e. at index position 2.

5) When we try to insert ‘92’ since position calculated for ‘92’ is 1 again collision occurs as ‘50’ is already present at position 1 so we store ‘92’ at next empty position i.e. at index position 3

6) When we try to insert '73' since position calculated for '92' is 3 collision occurs as '92' is already present at position 3 so we store '73' at next empty position i.e. at index position 4.

7) Finally while inserting '101' since index position '3' is already occupied to avoid collision we need to insert to insert at next empty position i.e.5.

Step 3

0	700
1	50
2	
3	
4	
5	
6	76

Step 4

0	700
1	50
2	85
3	
4	
5	
6	76

Step 5

0	700
1	50
2	85
3	92
4	
5	
6	76

Step 6

0	700
1	50
2	85
3	92
4	73
5	
6	76

Step 7

0	700
1	50
2	85
3	92
4	73
5	101
6	76

Double Hashing

Double Hashing is technique in which a second hash function is applied to key when collision occurs.

By applying the second hash function we will get the number of position from the point of collision to insert.

There are two important rules to be followed for the second function.

1. It must never evaluate to ZERO.

2. Must make sure that all the cell can be probed.

H_1(key) = key % Table Size

H_2(key) =(M-(key % M))

Where M is the a prime number smaller than the size of the table.

$$Hash(x)=(Hash_1(x)+ i * Hash_2(x))\% S$$

Example Consider the following elements to be placed in the hash table of size 10. {37,90,45,22,49,17,55} .

Initially , insert the elements using the formula H_1(key) = key % Table Size

H_1(37) = 37 % 10=7

H_1(90) = 90 % 10=0

H_1(45) = 45 % 10=5

H_1(22) = 22 % 10=2

H_1(49) = 49 % 10=9

0	90
1	
2	22
3	
4	
5	45
6	
7	37
8	
9	49

Now, H_1(17) = 17 % 10=7 collision so H_2(17) =(7-(17 % 7))=7-3=4 that means we have to place '17' at 4th place from '37'. In short we have to take 4 jumps therefore '17' is placed at

0	90
1	17
2	22
3	
4	
5	45
6	55
7	37
8	
9	49

1st index position in the hash table.

Now, $H_1(55) = 55 \% 10 = 5$ collision so $H_2(55) = (7-(55 \% 7)) = 7-6 = 1$ that means we have to place '55' at 1st place from '45'. In short we have to take 1 jumps therefore '55' is placed at 6th index position in the hash table.

Insert Operation

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code.

Search Operation -

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code.

Conclusion:

By this way, we can perform the operations on hash table and use collision handling techniques such as linear Probing and Double hashing.