UNIT-II: HASHING

Introduction - Static Hashing - Hash Table- Hash Functions - Secure Hash Function - Overflow Handling - Theoretical Evaluation of Overflow Techniques, Dynamic Hashing - Motivation for Dynamic Hashing - Dynamic Hashing Using Directories - Directory less Dynamic Hashing.

Introduction:

When a dictionary with n entries is represented as a binary search tree, the dictionary operations search, insert, and delete take **O(n)** time. These dictionary operations may be performed in **O(log n)** time using a balanced binary search tree. In this chapter, we examine a technique called hashing, that enables us to perform the dictionary operations search, insert and delete in **O(1)** time expected.

Hashing:

Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. Some examples of how hashing is used in our lives include:

- In universities or colleges, each student is assigned a unique roll number that can be used to retrieve information about them.
- In libraries, each book is assigned a unique number that can be used to determine information about the book, such as its exact position in the library or the users it has been issued to etc.

In both these examples the students and books were hashed to a unique number.

Assume that you have an object and you want to assign a key to it to make searching easy. To store the key/value pair, you can use a simple array like a data structure where keys (integers) can be used directly as an index to store values.

In hashing, large keys are converted into small keys by using **hash functions**. The values are then stored in a data structure called **hash table**. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key (converted key). By using that key you can access the element in **O(1)** time. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

Hashing is implemented in two steps:

1. **An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.**

2. **The element is stored in the hash table where it can be quickly retrieved using hashed key.**

There are many possibilities for representing the dictionary and one of the best methods for representing is hashing. Hashing is a type of a solution which can be used in almost all situations. Hashing is a technique which uses less key comparisons. This method generally used the hash functions to map the keys into a table, which is called a hash table.

Hashing is of two types.

1) Static Hashing and
2) Dynamic Hashing

Static Hashing

1) Hash Table

Hash table is a type of data structure which is used for storing and accessing data very quickly. Insertion of data in a table is based on a key value. Hence every entry in the hash table is defined with some key. By using this key data can be searched in the hash table by few key comparisons and then searching time is dependent upon the size of the hash table.

2) Hash Function

Hash function is a function which is applied on a key by which it produces an integer, which can be used as an address of hash table. Hence one can use the same hash function for accessing the data from the hash table. In this the integer returned by the hash function is called hash key.

Types of Hash Functions

There are various types of hash function which are used to place the data in a hash table,

1) **Division Method**
2) **Mid square Method**
3) **Digit folding Method**
4) **Digit Analysis Method/Binary/Radix Method**

1)Division Method

In this method the hash function is dependent upon the remainder of a division.

For example if the record **52, 68, 99, 84** is to be placed in a hash table and let us take the table size is 10.

Then:

h(key)= key % table_size
h(52) = 52 % 10 = 2
h(68) = 68 % 10 = 8
h(99) = 99 % 10 = 9
h(84) = 84 % 10 = 4

Division Method

0	
1	
2	52
3	
4	84
5	
6	
7	
8	68
9	99

Hash Table

2. Mid Square Method

In this method firstly key is squared and then mid part of the result is taken as the index. For example: consider that if we want to place a record of **3101** and the size of table is 1000.

So, **3101*3101=9616201**

i.e. h (3101) = 162 (middle 3 digit).

k	k*k	Square	index h(k)
15	15 * 15	225	2
22	22 * 22	484	8
16	16 * 16	256	5
29	29 * 29	841	4
31	31 * 31	961	6
5	5 * 5	25	0
3	3 * 3	9	9

Mid Square Method

0	5
1	
2	15
3	
4	29
5	16
6	31
7	
8	22
9	3

Hash Table

If the square value contains even number of digits the index is 0.

If it contains odd value index is middle value.

If the square value is a single digit, the value will be placed in that index only. Value 3 is stored at location 9.

3.Digit Folding Method

In this method, we partition the identifier k into several parts. All parts, except for the last one have the same length. We then add the parts together to obtain the hash address for k. There are two ways of carrying out this addition. In the first method, we shift all parts except for the last one, so that the least significant bit of each part lines up with the corresponding bit of the last part. We then add the parts together to obtain h(k). This method is known as **shift folding.**

The second method, known as **folding at the boundaries**, the key is folded at the partition boundaries, and digits falling into the same position are added together to obtain h(k). This is equivalent to reversing every other partition before adding.

Example 1: Suppose that k=12320324111220, and we partition it into parts that are 3 decimal digits long. The partitions are P1=123, P2=203, P3=241, P4=112 and P5=20

Shift Folding

$$h(k) = \sum_{i=1}^{5} P_i$$

= P1+P2+P3+P4+P5
= 123 + 203 + 241 + 112 + 20
= 699

Folding at the boundaries

When folding at boundaries is used, we first reverse P2 and P4 to obtain 302 and 211 respectively. Next the five partitions are added to obtain

$$h(k) = \sum_{i=1}^{5} P_i$$

= P1+P2+P3+P4+P5
= 123 + 302 + 241 + 211 + 20
= 897

Example 2) For example: consider a record of 12465512 then it will be divided into parts.
i.e. 124, 655, 12. After dividing the parts combine these parts by adding it.

Shift folding
H(key) =124+655+12 = 791
791 is the index to store the value 12465512

Folding at the boundaries
H(key) =124+556+12 = 692
692 is the index to store the value 12465512

Characteristics of Good Hashing Function

1) The hash function should generate different hash values for the similar string.

2) The hash function is easy to understand and simple to compute.

3) The hash function should produce the keys which will get distributed, uniformly over an array.

4) A number of collisions should be less while placing the data in the hash table.

5) The hash function is a perfect hash function when it uses all the input data.

Collision

It is a situation in which the hash function returns the same hash key for more than one record, it is called as collision. Sometimes when we are going to resolve the collision it may lead to a overflow condition and this overflow and collision condition makes the poor hash function.

Collision resolution technique

If there is a problem of collision occurs then it can be handled by apply some technique. These techniques are called as collision resolution techniques. There are generally four techniques which are described below.

1) Chaining

2) Linear Probing (Open addressing)

3) Quadratic Probing (Open addressing) and

4) Double Hashing (Open addressing).

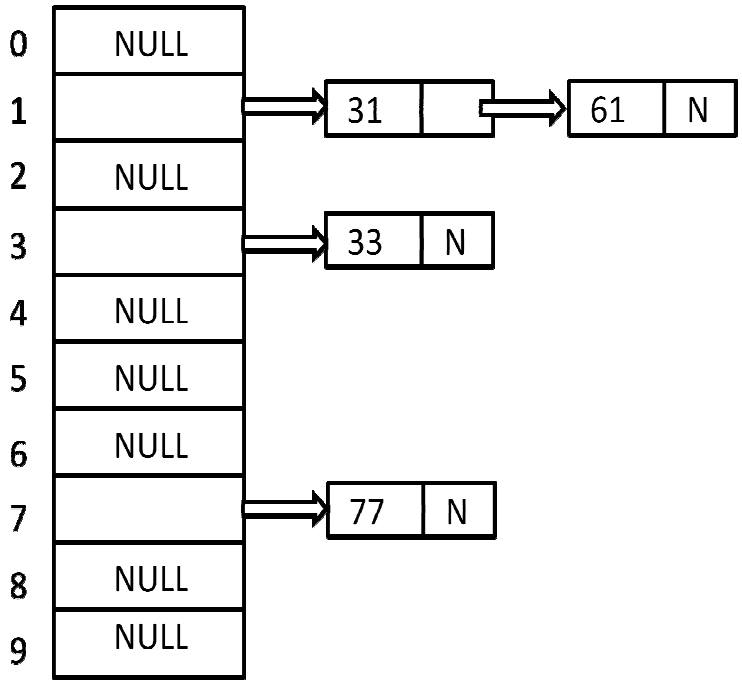
1) Chaining

It is a method in which additional field with data i.e. chain is introduced. A chain is maintained at the home bucket. In this when a collision occurs then a linked list is maintained for colliding data.

For Example: Let us consider a hash table of size 10 and we apply a hash function of H(key)=key % size of table. Let us take the keys to be inserted are **31, 33, 77, 61**.

In the diagram we can see at same bucket 1 there are two records which are maintained by linked list or we can say by chaining method.

H(31) = 31%10 = 1
H(33) = 33%10 = 3
H(77) = 77%10 = 7
H(61) = 61%10 = 1



2) Linear probing (Open addressing)

It is very easy and simple method to resolve or to handle the collision. In this, collision can be solved by placing the second record linearly down, whenever the empty place is found. In this method there is a problem of clustering which means at some place block of a data is formed in a hash table.

Example: Let us consider a hash table of size 10 and hash function is defined as H(key)=key % table_size. Consider that following keys are to be inserted that are **56, 64, 36, 71**.

56 % 10 = 6
64 % 10 = 4

0	NULL
1	NULL
2	NULL
3	NULL
4	64
5	NULL
6	56
7	NULL
8	NULL
9	NULL

36 % 10 = 6

The index 6 is already filled with 56
It is not empty
Collision occurred
To resolve this check the next location
i.e. 6+1 = 7
index 7 is NULL so insert 36 at index 7.

0	NULL
1	NULL
2	NULL
3	NULL
4	64
5	NULL
6	56
7	36
8	NULL
9	NULL

71 % 10 = 1

As the index 1 is null
We can insert 71 at index 1

0	NULL
1	71
2	NULL
3	NULL
4	64
5	NULL
6	56
7	36
8	NULL
9	NULL

In this diagram we can see that 56 and 36 need to be placed at same bucket but by linear probing technique the records linearly placed downward if place is empty i.e. it can be seen 36 is placed at index 7.

3) Quadratic Probing (Open addressing)

This is a method in which solving of clustering problem is done. In this method the hash function is defined by the

H(key)=(H(key)+x*x)%table_size

Let us consider we have to insert following elements that are:-
67, 90, 55, 17, 49.

67%10 = 7

90%10 = 0

55%10 = 5

17%10 = 7

49%10 = 9

0	90
1	
2	
3	
4	
5	55
6	
7	67
8	
9	

In this we can see if we insert 67, 90, and 55 it can be inserted easily but in the case of 17 hash function is used in such a manner that :-

To insert 17

The initial index generated is 17%10 = 7

But the index 7 is already filled with 67. Collision occurred.
To resolve this we try

(7+0*0)%10 = 7

(when x=0 it provide the index value 7 only) by making the increment in value of x. let x =1 so ,

(7+1*1)%10 = 8.

in this case bucket 8 is empty hence we will place 17 at index 8.

67%10 = 7

90%10 = 0

55%10 = 5

17%10 = 7

49%10 = 9

0	90
1	
2	
3	
4	
5	55
6	
7	67
8	17
9	49

4) Double hashing (Open addressing)

It is a technique in which two hash functions are used when there is an occurrence of collision. In this method 1 hash function is simple as same as division method. But for the second hash function there are two important rules which are

1. It must never evaluate to zero.
2. Must sure about the buckets, that they are probed.

The hash functions for this technique are:

H1(key) = key % table_size
H2(key) = P-(key mod P)

Where, **p** is a prime number which should be taken smaller than the size of a hash table.

Example: Let us consider we have to insert **67, 90,55,17,49**.

67%10 = 7
90%10 = 0
55%10 = 5
17%10 = 7
= 7 - (17%7)
= 7 - 3
= 4

49%10 = 9

0	90
1	17
2	
3	
4	
5	55
6	
7	67
8	
9	49

In this we can see 67, 90 and 55 can be inserted in a hash table by using first hash function but in case of 17 the bucket is full and in this case we have to use the second hash function which is

H2(key) = P - (key mod P)

where **P is a prime number which should be taken smaller than the hash table so value of P will be 7.**

i.e. **H2(17) = 7 - (17%7)**
= 7 - 3
= 4

that means we have to take **4 jumps for placing 17**. Therefore 17 will be placed at index 1.