MODULE 6

SEQUENTIAL SEARCH

Sequential_search(key)

Input: An unsorted array a[], n is the no.of elements, key indicates the element to be searched

Output: Target element found status

DS: Array

1. Start
2. i=0
3. flag=0
4. While i<n and flag=0
    1. If a[i]=key
        1. Flag=1
        2. Index=i
    2.end if
    3. i=i+1
5. end while
6. if flag=1
    1. print “the key is found at location index”
7. else
    1.print “key is not found”

7.end if

8. stop

Analysis

In this algorithm the key is searched from first to last position in linear manner. In the case of a successful search, it search elements up to the position in the array where it finds the key. Suppose it finds the key at first position, the algorithm behaves like best case, If the key is at the last position, then algorithm behaves like worst case. Thus the worst case time complexity is equal to the no. of comparison at worst case ie., equal to O(n). The time complexity in best case is O(1).

The average case time complexity =(no. of comparisons required when the key is in the first position + no. of comparisons required when the key is in second position+…+ no. of comparison when key is in nth position)/n

$$\frac{1+2+\ldots..n}{n} = \frac{n(n+1)}{2n} = O(n)$$

Binary Search

Binary Search(key)

Input: An unsorted array a[], n is the no.of elements, key indicates the element to be searched

Output: Target element found status

DS: Array

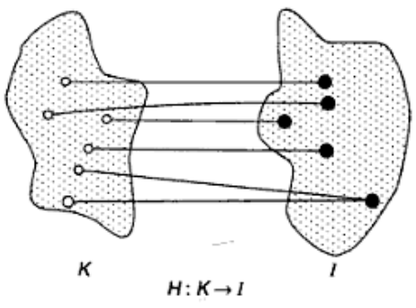
1. Start
2. Start=0,end=n-1
3. Middle=(start+end)/2
4. While key!=a[middle] and start<end
    1. If key>a[middle]
        1. Start=middle+1
    2. else
        1. end= middle-1
    3. end if
    4. middle=(start+end)/2
5. end while
6. if key=a[middle]
    1. print “the key is found at the position”
7. else
    1. print “the key is not found”
8. end if
9. stop

HASHING

We have seen about different search techniques (linear search, binary search) where search time is basically dependent on the no of elements and no. of comparisons performed.

Hashing is a technique which gives constant search time. In hashing the key value is stored in the hash table depending on the hash function. The hash function maps the key into corresponding index of the array(hash table).

The main idea behind any hashing technique is to find one-to-one correspondence between a key value and an index in the hash table where the key value can be placed. Mathematically, this can be expressed as shown in figure below where K denotes a set of key values, I denotes a range of indices, and H denotes the mapping function from K to I.



All key values are mapped into some indices and more than one key value may be mapped into an index value. The function that governs this mapping is called the hash function. There are two principal criteria in deciding hash function H:K->I as follows.

1) The function H should be very easy and quick to compute
2) It should be easy to implement

As an example let us consider a hash table of size 10 whose indices are 0,1,2,...9.Suppose a set of key values are 10,19,35,43,62,59,31,49,77,33.Let us assume the hash function as stated below

1) Add the two digits in the key
2) Take the digit at the unit place of the result as index , ignore the digits at tenth place if any

Using this hash function, the mapping from key values to indices and to hash tables are shown below.

K	I
10	1
19	0
35	8
43	7
62	8
59	4
31	4
49	3
77	4
33	6

H: K → I

0	19
1	10
2	
3	49
4	59, 31, 77
5	
6	33
7	43
8	35, 62
9	

Hash table

HASH FUNCTIONS

There are various methods to define hash function

Division method

One of the fast hashing functions, and perhaps the most widely accepted, is the division method, which is defined as follows:

Choose a number h larger than the number n of keys in K. The hash function H is then defined by

H(k)=k(MOD h) if indices start from 0

Or

H(k)=k(MOD h)+1 if indices start from 1

Where k€K, a key value. The operator MOD defines the modulo arithmetic operator operation, which is equal to dividing k by h. For example if k=31 and **h=13 then,**

H(31)=31 MOD 13=5 (OR)

H(31)=31(MOD 13)+1=6

h is generally chosen to be a prime number and equal to the sizeof hash table

MID SQUARE METHOD

Another hash function which has been widely used in many applications is the mid square method. The hash function H is defined by H(k)=x, where x is obtained by selecting an appropriate number of bits or digits from the middle of the square of the key value k. example-

k	**:**	**1234**	**2345**	**3456**
k^2	**:**	**1522756**	**5499025**	**11943936**
H(k)	**:**	**525**	**492**	**933**

For a three digit index requirement, after finding the square of key values,the digits at 2nd, 4th and 6th position are chosen as their hash values.

FOLDING METHOD

Another fair method for a hash function is folding method. In this method, the key k is partitioned into a number of parts k_1, k2..kn where each part has equal no.of digits as the required address(index) width. Then these parts are added together in the hash function.

H(k)=k1+k2+…+kn. Where the last carry, if any is ignored. There are mainly two variations of this method.

1)fold shifting method

2) fold boundary method

Fold Shifting Method

In this method, after the partition even parts like k2, k4 are reversed before addition.

Fold boundary method

In this method, after the partition the boundary parts are reversed before addition

Example

-Assume size of each part is 2 then, the hash function is computed as follows

Key values k :	**1522756**	**5499025**	**11943936**
Chopping :	**01 52 27 56**	**05 49 90 25**	**11 94 39 36**
Pure folding :	**01+52+27+56=136**	**05+49+90+25=169**	**11+94+39+36=180**
Fold Shifting:	**10+52+72+56=190**	**50+49+09+25=133**	**11+94+93+36=234**
Fold Boundary :	**10+52+27+65=154**	**50+49+90+52=241**	**11+94+39+63=207**

DIGIT ANALYSIS METHOD

This method is particularly useful in the case of static files where the key values of all the records are known in advance. The basic idea of this hash function is to form hash address by extracting and/or shifting the extracted digits of the key. For any given set of keys, the position in the keys and the same rearrangement pattern must be used consistently. The decision for extraction and rearrangement is finalized after analysis of hash functions under different criteria.

Example: given a key value 6732541, it can be transformed to the hash address 427 by extracting the digits from even position. And then reversing this combination.ie 724 is the hash address.

Collision resolution and overflow handling techniques

There are several methods to resolve collision. Two important methods are listed below:

1) **Closed hashing(linear probing)**
2) **Open hashing (chaining)**

CLOSED HASHING

Suppose there is a hash table of size h and the key value is mapped to location i, with a hash function. The closed hashing can then be stated as follows.

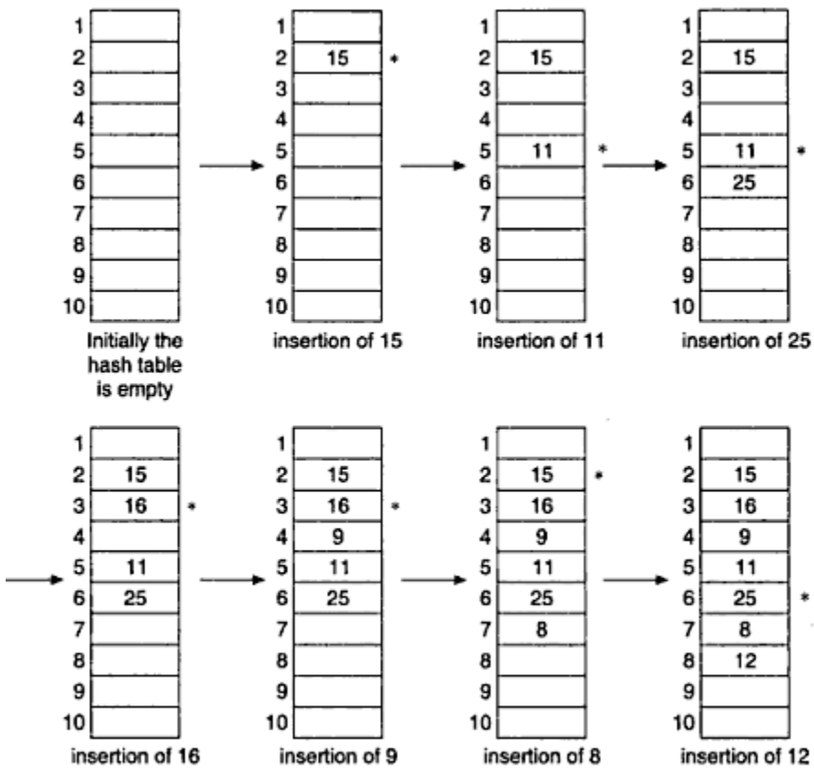
Start with the hash address where the collision has occurred,let it be i. Then carry out a sequential search in the order:- i, i+1,i+2..h,1,2...,i-1

The search will continue until any one of the following occurs

- The key value is found
- An unoccupied location is found
- The searches reaches the location where search had started

The first case corresponds to successful search , and the other two case corresponds to unsuccessful search.Here the hash table is considered circular, so that when the last location is reached, the search proceeds to the first location of the table. This is why the technique is termed closed hashing. Since the technique searches in a straight line, it is alternatively termed as linear probing.

Example- Assume there is a hash table of size 10 and hash function uses the division method of remainder modulo 7, namely H(k)=k(MOD 7)+1.The construction of hash table for the key values 15,11,25,16,9,8,12,8 is illustrated below.



Drawback of closed hashing and its remedies

The major drawback of closed hashing is that as half of the hash table is filled, there is a tendency towards clustering.That is key values are clustered in large groups.and as a result sequential search becomes slower and slower. This kind of clustering is known as primary clustering.

The following are some solutions to avoid this situation

1)Random probing

2)Double hashing

3)Quadratic probing

Random Probing

Instead of using linear probing that generates sequential locations in order, a random location is generated using random probing.

An example of pseudo random number generator that generates such a random sequence is given below:

I=(i+m)MOD h+1

Where m and h are prime numbers. For example if m=5, and h=11 and initially=2 then random probing generates the sequence

8,3,9,4,10,5,11,6,1,7,2

Here all numbers are generated between 1 and 11 in a random order. Primary clustering problem is solved. Where as there is an issue of clustering when two keys are hashed into the same location and then they make use of the same sequence locations generated by the random probing, which is called as secondary clustering

Double Hashing

An alternative approach to solve the problem of secondary clustering is to make use of second hash function in addition to the first one. Suppose H1(k) is initially used hash function and H2(k) is the second one. These two functions are defined as

$$H1(k)=(k \text{ MOD } h)+1$$

$$H2(k)=(k \text{ MOD } (h-4))+1$$

Let h=11, and k=50 for an instance, then

H1(50)=7 and H2(50)=2.

Now let k=28, then H1(28)=7 and H2(28)=5

Thus for the two key values hashing to the same location, rehashing generates two different locations alleviating the problem of secondary clustering.

Quadratic Probing

It is a collision resolution method that eliminates the primary clustering problem of linear probing. For linear probing, if there is a collision at location i, then the next locations i+1, i+2..etc are probed.But in quadratic probing next locations to be probed are $i+1^2$,$i+2^2$,$i+3^2$ ..etc . This method substantially reduces primary clustering, but it doesn't probe all the locations in the table.

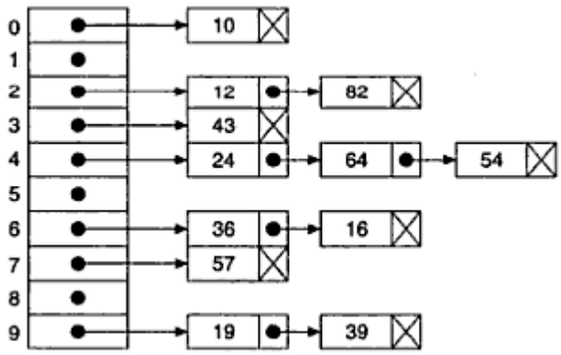
Open Hashing

Closed hashing method for collision resolution deals with arrays as hash tables and thus random positions can be quickly refer red. Two main disadvantages of closed hashing are

1) It is very difficult to handle the problem of overflow in a satisfactory manner
2) The key values are haphazardly intermixed and, on the average majority of the key values are from their hash locations increasing the number of probes which degrades the overall performance

To resolve these problems another hashing method called open hashing or separate chaining is used.

The chaining method uses hash table as an array of pointers. Each pointer points a linked list. That is here the hash table is an array of list of headers. Illustrated below is an example with hash table of size10.



For searching a key in hash table requires the following steps

1)Key is applied to hash function

2) Hash function returns the starting address of a particular linked list(where key may be present)

3)Then key is searched in that linked list

Performance Comparison Expected

Algorithm Name	**Best Case**	**Average Case**	**Worst Case**
Quick Sort	**O(n log n)**	**O(n log n)**	**$O(n^2)$**
Merge Sort	**O(n log n)**	**O(n log n)**	**O(n log n)**
Heap Sort	**O(n log n)**	**O(n log n)**	**O(n log n)**
Bubble Sort	**O(n)**	**$O(n^2)$**	**$O(n^2)$**
Selection Sort	**$O(n^2)$**	**$O(n^2)$**	**$O(n^2)$**
Insertion Sort	**O(n)**	**$O(n^2)$**	**$O(n^2)$**
Binary Search	**O(1)**	**O(log n)**	**O(log n)**
Linear Search	**O(1)**	**O(n)**	**O(n)**

```
        {   mid=((beg+end)/2) ;
            if(item==a[mid])
            {   printf("Search is successfull\n") ;
                loc=mid ;
                printf("Position of the item %d\n",loc+1) ;
                flag=flag+1 ;
            }
            else if(item<a[mid])
            end=mid-1 ;
            else
            beg=mid+1 ;
        }
        if(flag==0)
        {
            printf("Search is not successfull\n") ;
        }
        getch( ) ;
    }
```

The output of the above program is :

```
How many elements
    6
Enter the element of the array
    11
    22
    33
    44
    55
    66
Enter the element to be searching
    55
Search is successful
Position of the item 5
```

Note : *In above example, it is considered that the numbers are entered in the ascending order. If we want to enter in the zig-zag order then first we sort the numbers and then apply the binary search technique.*

11.4 HASHING

The searching techniques that were discussed in the previous sections are based on comparision of keys. We need search techniques in which there are no unnecessary comparisions of keys. We need search techniques in which there are no unnecessary comparisions. Therefore, the objective here is to minimize the number of comparisions in order to find the desired record efficiently.

There is an approach, in which we compute the location of the desired record in order to retrieve it in a single access. This avoid the unnecessary comparision. In this method, the location of the desired record present in the search table depends only on the given key but not on other keys. For example, if we have a table of n students records each of which is defined by the roll number key. This roll number key takes value from 1 to n inclusive. If the roll number key takes values from 1 to n inclusive. If the roll number is used as an index into the students table, we can directly find the information of student in question. Therefore, array can be used to organize records in such a search table.

11.5 HASH TABLE

A hash table is a data structure where we store a key value after applying the hash function, it is arranged in the form of an array that is addressed via a hash function. The hash table is divided into a number of buckets and each bucket is in turn capable of storing a number of records. Thus we can say that a bucket has number of slots and each slot is capable of holding one record.

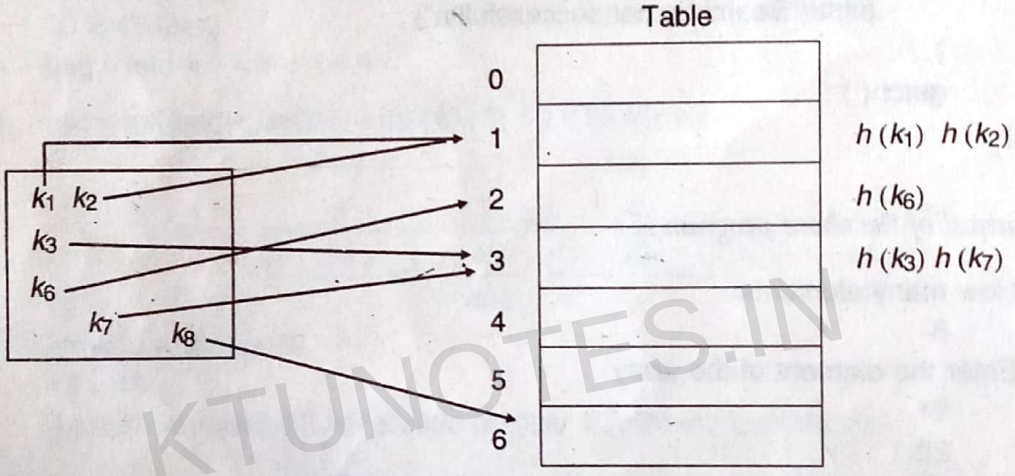


Fig. 11.2

The time required to locate any element in the hash table is 0 (1). It is constant and it is not depend on the number of data elements stored in the table. Now question is how we map the number of keys to a particular location in the hash table *i.e.*, $h(k)$. It is computed using the hash function.

11.6 HASH FUNCTION

The basic idea in hashing is the transformation of a key into the corresponding location in the hash table. This is done by a hash function. A hash function can be defined as a function that take key as input and transforms it into a hash index. It is usually denoted by H

$$H : K \rightarrow M$$

where
- H is a Hash function
- K is a set of keys
- M is a set of memory addresses.

Sometimes, such a function H may not yield distinct values, it is possible that two different keys K_1 and K_2 will yield the same hash address. This situation is called Hash collision.

There are different types of Hash functions are available. To chose the hash function $H : K \rightarrow M$ there are two things to consider. Firstly the function H should be very easy and quick to compute. Secondly the function H should, distribute the keys to the number of locations of hash table with less no of collisions. Different hash functions are

(*i*) Division reminder method,
(*ii*) Mid square method.
(*iii*) Folding method.

11.6.1 Division Reminder Method

In division reminder method, key k is divided by a number m larger than the number n of keys in k and the reminder of this division is taken as index into the hash table, *i.e.*,

$$h(k) = k \bmod m$$

The number m is usually chosen to be a prime number or a number without small divisors, since this frequently minimizes the number of collisions.

The above hash function will map the keys in the range 0 to $m-1$ and is acceptable in C/C++. But if we want the hash addresses to range from 1 to m rather than from 0 to $m-1$ we use the formula

$$h(k) = k \bmod m + 1$$

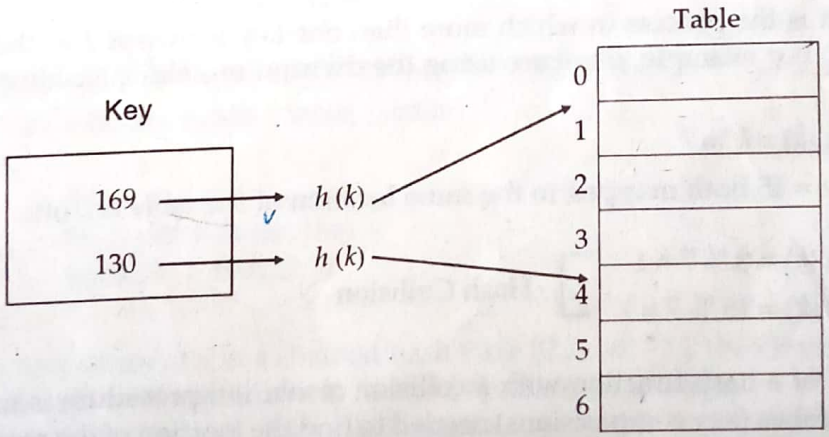
EXAMPLE 11.3: *Consider a hash table with 7 slots i.e., $m = 7$, then hash function $h(k) = k \bmod m$ will map the key 169 to slot 1 since*

$$h(169) = 169 \bmod 7 = 1$$

similarly

$$h(130) = 130 \bmod 7 = 4$$

is mapped to slot 4.



11.6.2 Mid Square Method

In the mid square method the key is first squared. Therefore the hash function is defined by

$$h(k) = p$$

where p is obtained by deleting digits from both sides of k^2. To properly implement this the same position of k^2 must be used for all the keys.

EXAMPLE 11.4. *Consider a hash table with 50 slots i.e., $m = 50$ and key values $k = 1632, 1739, 3123$.*

SOLUTION.

k	:	1632	1739	3123
k^2	:	2663424	3024121	9753129
$h(k)$	:	34	41	31

The hash values are obtained by taking the fourth and fifth digits counting from right.

11.6.3 Folding Method

In folding method the key, k is partitioned into a number of parts $k_1\ k_2$ ..... k_r where each part, except possibly the last, has the same number of digits as the required address. Then the parts are added together, ignoring the last carry *i.e.*,

$$H(k) = k_1 + k_2 + ... + k_r$$

where the leading-digits carries, if any are ignored.

EXAMPLE 11.5. *Consider a hash table with 100 slots i.e., $m = 100$ and key values $k = 7325, 76321, 1623, 7613$.*

SOLUTION.

k	*Parts*	*Sum of parts*	$h(k)$
7325	73,25	98	98
76321	76,32,1	109	09
1623	16,23	39	39
7613	76,13	89	89

11.7 RESOLVING COLLISION

Hash collision is the process in which more than one key is mapped to the same memory location in the table. For example, if we are using the division reminder hashing with following hash function

$$h(k) = k \% 7$$

then key = 8 and key = 15 both mapped to the same location of the table *i.e.*, one

$$h(k) = 8 \% 7 = 1$$
$$h(k) = 15 \% 7 = 1$$

Hash Collision

The efficiency of a hash function with a collision resolution procedure is measured by the average number of probes (key comparisions) needed to find the location of the record with a given key k. The efficiency depends mainly on the load factor λ. Specifically, we are interested in the following quantities :

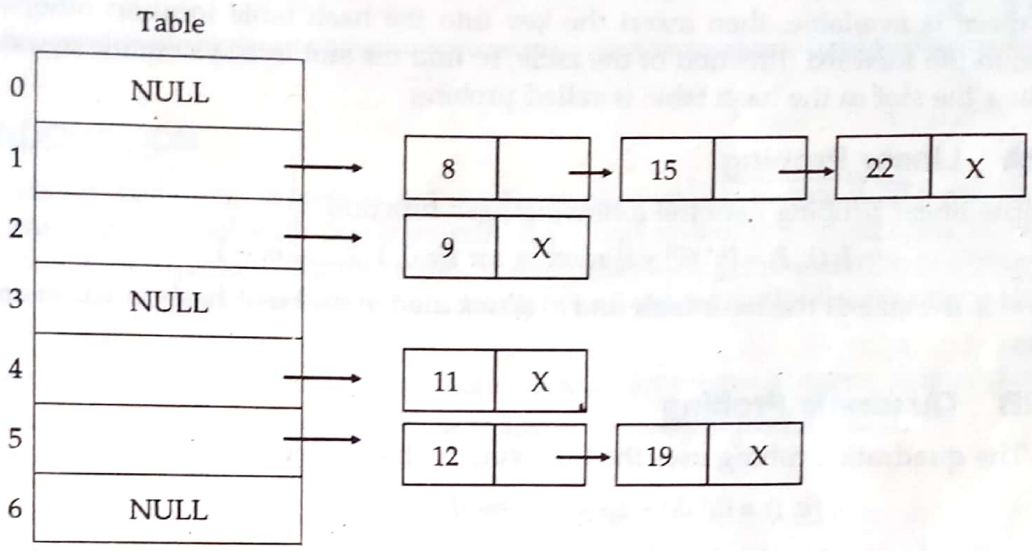
$S(\lambda)$ = average number of probes for a successful search

$U(\lambda)$ = average number of probes for an unsuccessful search

Load factor (λ) is the ratio of the number n of keys in k to the number m of hash address in L. This ratio, $\lambda = n/m$ is called load factor.

11.7.1 Collision Resolution by Separate Chaining

In this method all the elements where keys hash to the same hash-table slot are put in a one linked list. Therefore, the slot *i* in the hash table contains a pointer to the head of the linked list of all the elements that hash to value *i*. If there is no such elements that hash to value *i*, the slot *i* contains NULL value.



The structure of linked list looks like typed of struct node

```
{
    int key ;
    struct node * next ;
} NODE ;
```

To initiation the a chained hash table.

Table [0 ..... $m-1$], NULL pointer is assigned to each slot :

```
Void initialize (node * table[ ], int m)
{
    int i ;
    for (i = 0, i <= m ; i++)
    table [ i ] = NULL ;
}
```

To insert a new element n in a chained hash table [0 ..... $m-1$], the element is inserted in the beginning of the linked whose pointers to its head is stored at slot h [x].

```
void insert (node * table[ ], int num)
{
    node * p ;
    p = (node *) malloc (size of (node)) ;
    p → key = num ;
    p → next = table [h[x]]
    table[h[x]] = p ;
}
```

11.7.2 Collision Resolution by Open Addressing

In open addressing the keys to be hashed is to put in the separate location of the hash table. Each location contains some key or the some other character to indicate that the particular location is free.

In this method to insert key into the table we simply hash the key using the hash function. If the space is available, then insert the key into the hash table location otherwise, search the location in the forward direction of the table, to find the slot in a systematic manner. The process of finding the slot in the hash table is called probing

11.7.2A Linear Probing

The linear probing uses the following hash function

$$h(k, i) = [h'(k) + i] \bmod m \text{ for } i = 0, 1, 2, \ldots m - 1$$

where m is the size of the hash table and $h'(k) = k \bmod m$ the basic hash function and i is the probe number.

11.7.2B Quadratic Probing

The quadratic probing uses the following hash function.

$$h(k, i) = [h'(k) + c_1 i + c_2 i^2] \bmod m \quad \text{for} \quad i = 0, 1, 2, \ldots m - 1$$

where m is the size of hash table $h'(k) = k \bmod m$ the basic hash function, c_1 and $c_2 \neq 0$ are auxiliary and is the probe number.

11.7.2C Double Hashing

In double hashing second hashing function H' is used for resolving a collision suppose a record R with key k has the hash addresses $H(k) = h$ and $H'(k) = h' \neq m$.

Then we linearly search the location with addresses

$$h, h + h', h + 2h', h + 3h' \ldots$$

If m is prime number, then the above sequence will access all the locati ns in the table T.

11.8 REHASHING

If at any stage the hash table become full or overflow then it will be very difficult to find the free slot and it will increase the execution time. In such a situation, create a new hash table of size double then the original hash table. Then, scan the previous hash table and for each key, compute the new hash value and insert into the new hash table. Thereafter free the memory oceryned by the original hash table.

Self Review Questions

1. What do you mean by searching ?
2. Differentiate between the linear and binary search.
3. What is the prerequisite for the binary search.
4. What do you mean by the hashing ?
5. What are the different method of hashing ?
6. How we resolve the collision ?
7. What is linear probing how it differ from quadratic probing ?