

## 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  $n$ th position)/ $n$

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

## 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

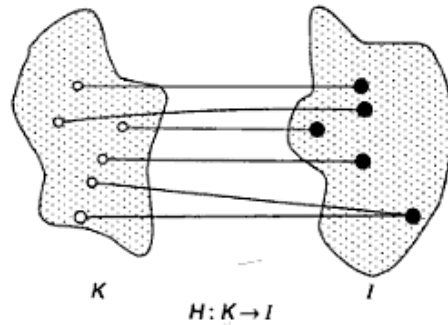
**DS:** Array

1. Start
2.  $Start=0, end=n-1$
3.  $Middle=(start+end)/2$
4. While  $key \neq a[middle]$  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  $H: K \rightarrow I$  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

$H: K \rightarrow I$

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

$$H(k) = k(\text{MOD } h) \text{ if indices start from 0}$$

Or

$$H(k) = k(\text{MOD } h) + 1 \text{ if indices start from 1}$$

Where  $k \in K$ , 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,

$$H(31)=31 \text{ MOD } 13=5 \text{ (OR)}$$

$$H(31)=31( \text{ MOD } 13)+1=6$$

$h$  is generally chosen to be a prime number and equal to the size of 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-

<b>k</b>	:	<b>1234</b>	<b>2345</b>	<b>3456</b>
<b><math>k^2</math></b>	:	<b>1522756</b>	<b>5499025</b>	<b>11943936</b>
<b>H(k)</b>	:	<b>525</b>	<b>492</b>	<b>933</b>

For a three digit index requirement, after finding the square of key values, the digits at 2<sup>nd</sup>, 4<sup>th</sup> and 6<sup>th</sup> 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  $k_1, k_2..k_n$  where each part has equal no. of digits as the required address(index) width. Then these parts are added together in the hash function.

$H(k)=k_1+k_2+...+k_n$ . 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  $k_2, k_4$  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

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

## **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.

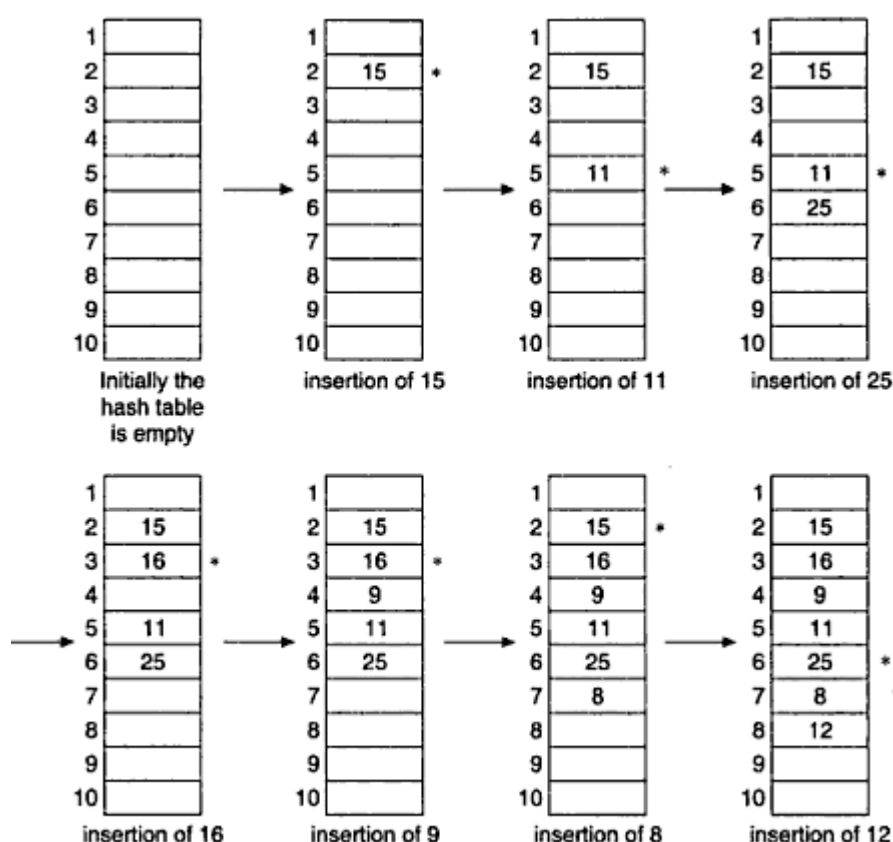
Start with the hash address where the collision has occurred, let it be  $i$ . Then carry out a sequential search in the order:-  $i, i+1, i+2, \dots, h-1, 0, 1, 2, \dots, i-1$

The search will continue until any one of the following occurs

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

The first case corresponds to successful search, and the other two cases correspond 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  $H(k) = k \text{ (MOD } 7) + 1$ . 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:

$$I = (i + m) \text{MOD } h + 1$$

Where  $m$  and  $h$  are prime numbers. For example if  $m=5$ , and  $h=11$  and initially  $i=2$  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. Whereas 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

$$H1(50)=7 \text{ and } H2(50)=2.$$

Now let  $k=28$ , then

$$H1(28)=7 \text{ and } H2(28)=5$$

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.

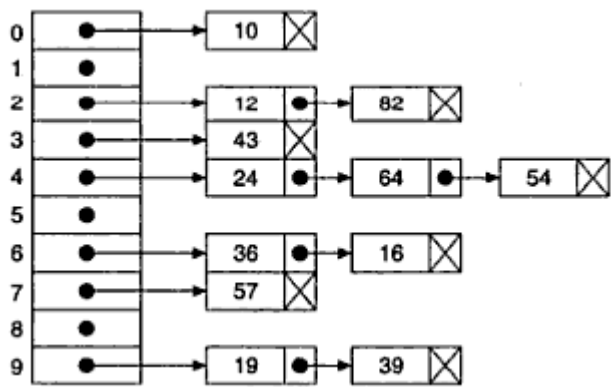
## Open Hashing

Closed hashing method for collision resolution deals with arrays as hash tables and thus random positions can be quickly referred. 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 size 10.



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	$O(1)$	$O(n)$	$O(n)$



MEA Engineering College, Perinthalmanna