07: Searching

* Search = A search (al) is a method of locating a specific item of info in a larger collection of data.
* 2 types →

1) Linear search / Sequential search =

* Here, we search an element / value in a gn array by traversing the array frm starting, till the desired element is found.
* It compares the elemnt to be searched with all the elemnts prsnt in array & whn the elemnt is matched successfully it returns the index of elemnt in the array.
* used for unsorted & unordered small list of elemnts.
* It has time complexity of O(n) [time is linearly dependent on nos of elemnts].
* It is very simple to implemnt.

* (Al) =

Returns location LOC of ITEM in the LIST.

Linear Search (LIST, ITEM).

1) Set LOC = 0
2) Repeat stp 2 to 3 for each elemnt in LIST
3) if LIST [LOC] == ITEM
4) return the elemnts location LOC.
5) LOC = LOC + 1
6) Return.

* Prgm =

```
printf ("Enter the no. to search: ");
scanf ("%d", &num);
for (i=0; i<n; i++) {
    if (a[i] == num) {
        printf ("element %d is prsnt at location %d", num, i);
        break;
    }
}
if (i==n)          (index == size of array)
    printf ("not prsnt");
return 0;
}
```

2) Binary search =

* It is very fast & efficient technique for finding a particular value in a linear array, by "ruling out" half of the data at each step.
* This method requires that the list of elemnts be in sorted order.
* It finds the middle elemnt of the list & makes the comparison to determine whether the middle elemnt is =, > or < than the desired value.
  - If they are =, search has been completed successfully.
  - If it is > than the item being searched for, search process is repeated on the 1st half of the array.

otherwise, the process is repeated in 2nd half.

* b.search is an eg of a ÷ & conquer (d).
* most common appli- of b.search is to find a specific value in a sorted list.
* time complexity is O(log n).

* (Al) =

BinarySearch (LIST, ITEM, LOWER, UPPER)

1) if UPPER < LOWER then
   return not found.
2) MID = (LOWER + UPPER) / 2.
3) if LIST[MID] = ITEM.
   return MID.
4) if item < LIST[MID]
   return binarysearch (LIST, ITEM, lower, MID-1)
5) else.
   return " (LIST, ITEM, MID+1, UPPER)
6) return.

* prgm = lower=0, upper, flag=0, ~~

```
upper = elemnts - 1
for (mid = (lower + upper) / 2 ; lower <= upper;
        mid = (lower + upper) / 2)
{
    if (a[mid] == num)
        flag = 1
    if (a[mid] > num)
        upper = mid - 1
    else
        lower = mid + 1
}
if (flag)
    printf("%d found at %d", num, mid);
else
    printf("not found");
```

⇒ Hashing = (H).

* It is a technique /process of mapping keys, values into the hash table by using a hash().
* (H) is designed to solve the prbm of needing to efficiently find / store an item in a collection.
* It is done for faster access to elemnts.
* The efficiency of mapping depends on the efficiency of hash () used.
* Hash table uses a hash () to compute an index, also → hash code, into an array of buckets / slots. frm which desired value can be found.
* The key is hashed & the resulting hash indicates where the corresponding value is stored.
* (H) is a technique which uses less key comparisions & searches the elemnt in
  - O(n) time in the worst case.
  - O(1) time " avg case.
* There are 4 key components in (H) -
  a) hash table = It is a storage location in memory that records the hashed values created by (H) (al).
  b) Hashing function = It is any () that can be used to map a data set of

an arbitary size to a data set of a fixed size, which falls into the hash table.

* values returned by a hash() → hash value, hash codes, hash sums.

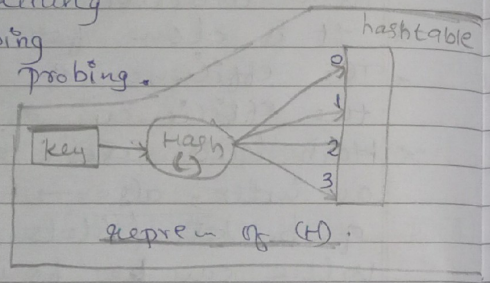
c) collisions = It is when 2 distinct data items produce the same hash value & thus want to be stored in the same table location.

~~Here are are linked lopes~~.

d) collision resolution techniques =

* chaining (L.S)
* Double chaining
* Linear probing
* Quadratic probing.

→ Types of (H) () =

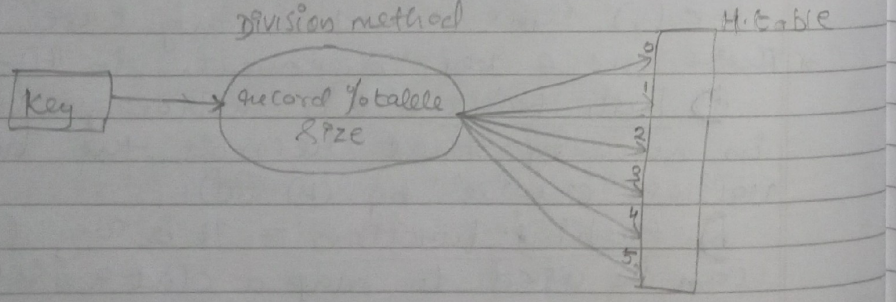


(a) Division method =

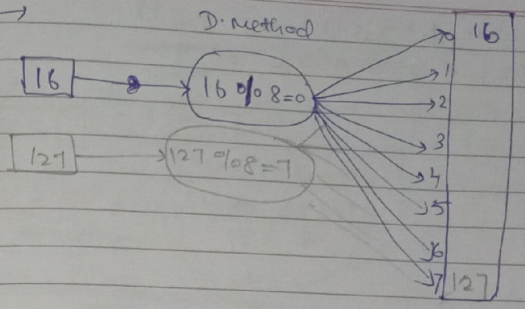
Here it is dependent upon the remainder (%) of a division.

h(key) = record % table size

record → data.



eg →

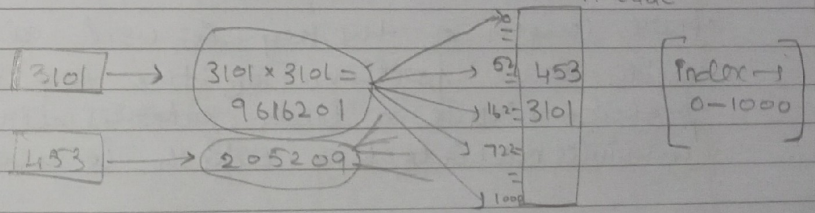


8 → total size
16 → key.
16 % 8 = 0
0th index we store 16

(b) Mid Square method =

* Here, firstly key is squared & then mid part of result is taken as the indx.
* eg → ① key → 3101.
  key² → 3101 × 3101 = 9616201
  ∴ h(3101) = 162 , (middle 3 digit)

② key → 453
key² → 453 × 453 = 205209.
∴ h(453) = 52



(c) ~~Dog~~ Folding method =

* Here, key is divided into seperate parts & by using some simple operations these parts are combined to produce a hash key.
* eg → key → 12465512.

* Folding at boundaries = alternative to reverse then

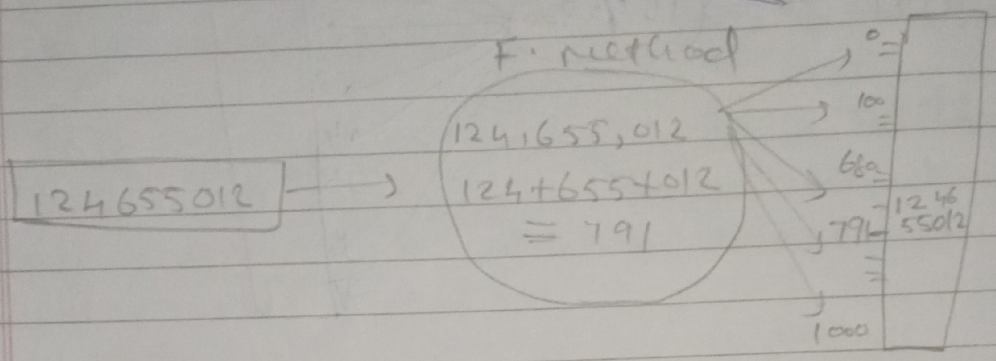
eg → 30250124 → 30|25|01|24

→ 30|52|01|42

→ 30+52+01+42 = 125

* 124,655,012.

* 124+655+012 = 791



d) Digit Analysis =

* This method forms addresses by selecting & shifting digits / bits of original key.

eg → key → 9861234. ಈಲ್ಲಿ keys ಅಲ್ಲಿ 2ನೆ ಮತ್ತು 3rd & 5th position keys same position ಅಲ್ಲಿ repeat ಆಗ್ತಿದೆ.

So we select. 62 & reverse it → 26

26 → address

e) Length dependent method =

* Here, length of the key is used along with some portion of the key to produce either a table address directly, an intermediate key which is used.