UNIT - II

HASH FUNCTION

A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.

To achieve a good hashing mechanism, It is important to have a good hash function with the following basic requirements:

- Easy to compute: It should be easy to compute and must not become an algorithm in itself.
- Uniform distribution: It should provide a uniform distribution across the hash table and should not result in clustering.
- Less collisions: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

Division Method:

One common method of determining a hash key of the division method of hashing

The formula that will be used is:

H(key) = key % no.of slots in the table

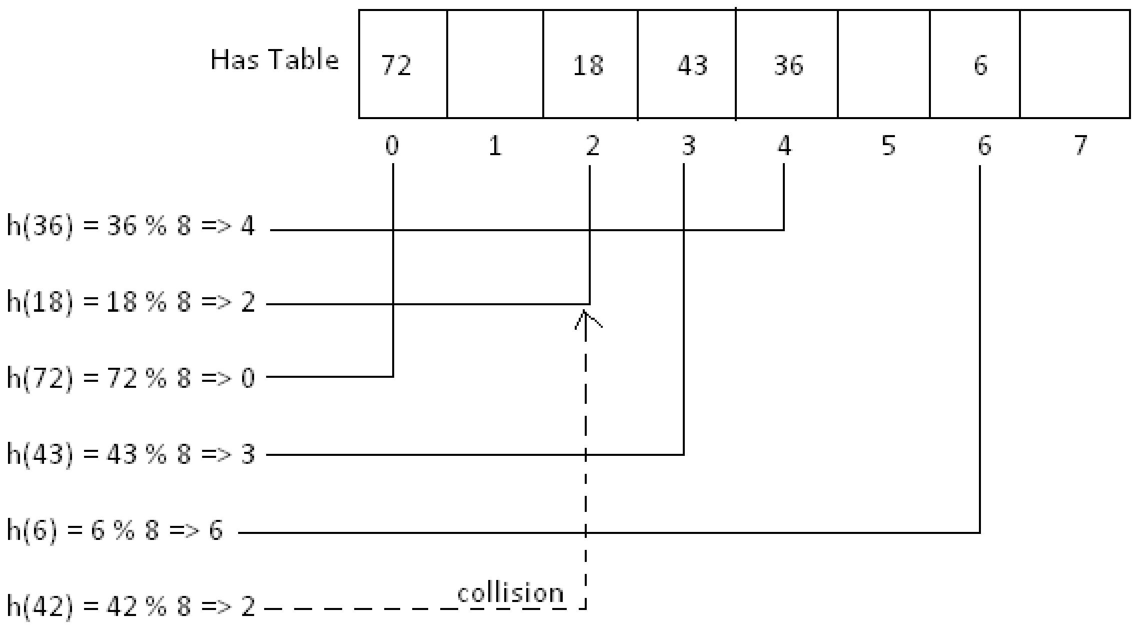
i.e. $\boxed{\text{h(key) = key mod array size}}$

For example:

Consider a table with 8 slots. i.e. array size 8.

Hash key = key % table size

The key values are 36, 18, 72, 43, 6, 42



The division method is generally a reasonable strategy, unless the key happens to have some undesirable properties.

Note: if the table size is 10 and all of the keys end in zero.
In the above example 42 mod 8 => 2, it's already filled position in the hash table. This is known as *collision.* i.e. two or more record keys map to the same array index.
In This case, the choice of hash function and table size needs to be carefully considered. The best table sizes are *prime numbers*.

Multiplication method:

The simplest situation when the keys are floating –point numbers known to be in affixed range.

For example:
If the keys are numbers that are greater than 0 and less than 1, we can just multiply by m (table size) and round off to the nearest integer to get an address between 0 an m-1.
Algorithm:

1. Choose constant A in the range 0 < A < 1.
2. Multiply key k by A.
3. Extract the fractional part of *k*A*
4. Multiply the fractional part by number of slots*, m*.
5. Take the floor of the result.

Mathematically

$h(k) = \lfloor m \cdot (k \cdot A \bmod 1) \rfloor$

where $k \cdot A \bmod 1 = k \cdot A - \lfloor k \cdot A \rfloor$ = fractional part of $k \cdot A$

- Disadvantage: Slower than division method.
- Advantage: Value of m is not critical.

Example:

$m = 8$ (implies $m = 2^3$, $p = 3$)

$w = 5$

$k = 21$

Must have $0 < s < 2^5$; choose $s = 13 \rightarrow A = 13/32$.

- $h(k) = \lfloor m \cdot (k \cdot A \bmod 1) \rfloor$

  $h(21) = \lfloor 8 \cdot (21 \cdot 13/32 \bmod 1) \rfloor = 4$

  $k \cdot A = 21 \cdot 13/32 = 273/32 = 8\ 17/32 \rightarrow k \cdot A \bmod 1 = 17/32$

  $\rightarrow m \cdot (k \cdot A \bmod 1) = 8 \cdot 17/32 = 17/4 = \rightarrow 4\ 1/4 \rightarrow \lfloor m \cdot (k \cdot A \bmod 1) \rfloor = 4$

  So that $h(21) = 4$.

Example:

$m = 8$ (implies $m = 2^3$, $p = 3$)
$w = 5$
$k = 21$
$s = 13$

$k \cdot s = 21 \cdot 13$

$= 273$

$= 8 \cdot 2^5 + 17$

$= r_1 . r_0$

$r_1 = 8 \cdot 2^5$
$r_0 = 17 = 10001_2$

Written in w = 5 bits, $r_0 = 10001_2$ The p = 3 most significant bits of r_0 is 100_2 or 4_{10}, so h(21) = 4.

Exercise Example:

m = 4 (implies $m = 2^2$, p = 2)
w = 3
k = 12
s = 5 $0 < s < 2^w = 2^3 = 8$

$k \cdot s = 12 \cdot 5$

$= ?$

$= ? \cdot 2^3 + ?$

$= r_1 . r_0$

$r_1 = ? \cdot 2^3$
$r_0 = ? = ?_2$

- Written in w = 3 bits, $r_0 = ?_2$
- The p = 2 most significant bits of r_0 is $?_2$ or $?_{10}$, so h(12) = ?.

Universal method:

Hashing is a fun idea that has lots of unexpected uses. Here, we look at a novel type of hash function that makes it easy to create a family of universal hash functions. The method is based on a random binary matrix and is very simple to implement.
The idea is very simple. Suppose you have an input data item that you have input data with m – bits and you want a hash function that produces n – bits then first generate a random binary matrix (M) of order nxm.

The hash function is $\boxed{}$ Where x to be a binary vector
For example, Suppose you have a key 11, the binary form is 1011 and it is a four bit input value(m) and want to generate output a three bit hash value(n).
Then generate a random matrix gives say:

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

and if the data value was 1011 the hash value would be computed as:

(0 1 0 0) (1) (0)

$$h(x) = Mx = \begin{array}{ll} (1\ 0\ 1\ 1) & (0) \\ (1\ 1\ 0\ 1) & (1) \\ & (1) \end{array} = \begin{array}{l} (1) \\ (0) \\ \end{array}$$

There are a number of other ways to look at the way the arithmetic is done that suggest different ways of implementing the algorithm.

The first is to notice that what you are doing is anding each row with the data column vector. That is taking the second row as an example: (1 0 1 1)And (1 0 1 1) = (1 0 1 1)
and then you add up the bits in the result: 1+0+1+1=1

now the index is 010, convert that into decimal is 2.

There is no.of other ways to look at the way the arithmetic is done that suggest different ways of implementing the algorithm.

Hashing gives an alternative approach that is often the fastest and most convenient way of solving these problems like AI – search programs, cryptography, networks, complexity theory.