UNIT-II

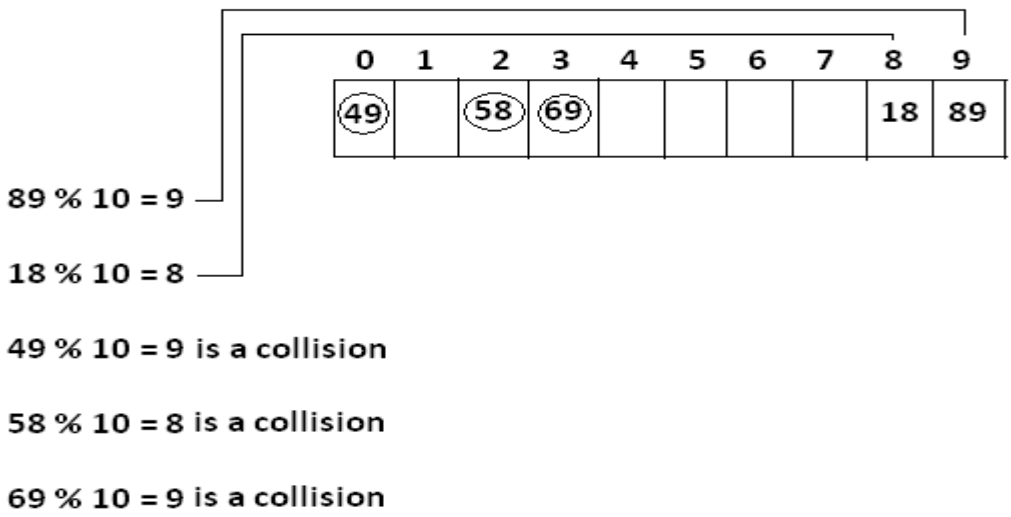
HASHING WITH QUADRATIC PROBING:

In this probe, rather than always moving one spot, move ' i^2 ' spots from the point of collision, where 'i' is the no. of attempts to resolve the collision.

In linear probe, if the primary hash index is x, subsequent probe go to x+1, x+2, x+3 and so on. In Quadratic probing, probes go to x+1, x+4, x+9, and so on, the distance from the initial probe is the square of the step number: $x+1^2$, $x+2^2$, $x+3^2$, $x+4^2$ and so on.

i.e., at first it picks the adjacent cell, if that is occupied, it tries 4 cells away, if that is occupied it tries 9 cells away, and so on. It eliminates the primary clustering problem with linear probe.

Consider the above exercise problem, keys 89, 18, 49, 58, 69 with table size 10.



Here each key that hashes to same location will require a longer probe. This phenomenon is called **secondary clustering**. It is not a serious problem, but quadratic probing is not often used because there is a slightly better solution.

Hashing with Pseudo – random probing:

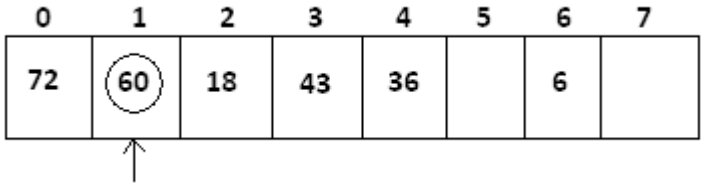
This method uses pseudo – random number to resolve the collision i.e. this probe function would select the next position on the probe sequence at random from among the unvisited slots that is the probe sequence should be a random permutation of hash table positions.

Unfortunately, we can't actually select the next position in the probe sequence at random, because we would not be able to duplicate this same probe sequence when searching for the key.

In this probing, the i^{th} slot in the probe sequence is (h (key) + r_1) mod M) where r_1 is the i^{th} value in the random permutation of the numbers from 1 to M-1.

All insertions and searches use the same sequence of random numbers.

Consider the same example of division method:



36 % 8 = 4

18 % 8 = 2

72 % 8 = 0 now insert 60

43 % 8 = 3 60 % 8 = 4; **is a collision**

6 % 8 = 6

The Pseudo – random permutation to use is: **0 6 5 2 3 4 1 7**

For collision resolution

Current slot = (4 + **0**) % 8 = 4; searching slot 4 and it is **occupied**

Again, Current slot = (4 + **6**) % 8 = 2; **occupied**

Current slot = (2 + **5**) % 8 = 1; it is **empty;** key 60 is occupies slot 1

Pseudo random numbers are a relatively simple solution, but they have one significant limitation all keys follow only one collision resolution path through the list. Because this collision resolution can create significant secondary clustering