Unit 3:Hashing
Chapter 4

4.0. Objective

This chapter would make you understand the following concepts:

- **Different Hashing Techniques**
- **Address calculation Techniques**
- **Collision resolution techniques**

4.1 Hashing

Hashing is the change of a line of character into a normally more limited fixed-length worth or key that addresses the first string.
Hashing is utilized to list and recover things in a data set since it is quicker to discover the thing utilizing the briefest hashed key than to discover it utilizing the first worth. It is likewise utilized in numerous encryption calculations.
A hash code is produced by utilizing a key, which is an exceptional worth.
Hashing is a strategy where given key field esteem is changed over into the location of capacity area of the record by applying a similar procedure on it.

The benefit of hashing is that permits the execution season of fundamental activity to stay consistent in any event, for the bigger side.

4.2 Why we need Hashing?

Assume we have 50 workers, and we need to give 4 digit key to every representative (with respect to security), and we need subsequent to entering a key, direct client guide to a specific position where information is put away.

In the event that we give the area number as per 4 digits, we should hold 0000 to 9999 locations since anyone can utilize anybody as a key. There is a great deal of wastage.

To tackle this issue, we use hashing which will deliver a more modest estimation of the record of the hash table relating to the key of the client.

4.3 Universal Hashing

Leave H alone a limited assortment of hash works that map a given universe U of keys into the reach {0, 1. m-1}. Such an assortment is supposed to be all inclusive if for each pair of particular keys k,l∈U, the quantity of hash capacities h∈ H for which h(k)= h(l) is all things considered |H|/m. As such, with a hash work haphazardly browsed H, the possibility of an impact between particular keys k and l is close to the opportunity 1/m of a crash if h(k) and h(l)were arbitrarily and autonomously looked over the set {0,1,. m-1}.

4.4. Rehashing

On the off chance that any stage the hash table turns out to be almost full, the running time for the activities of will begin taking an excess of time, embed activity may fall flat in such circumstance, the most ideal arrangement is as per the following:

1. Make another hash table twofold in size.
2. Output the first hash table, register new hash worth and addition into the new hash table.
3. Free the memory involved by the first hash table.

Model: Consider embeddings the keys 10, 22, 31,4,15,28,17,88 and 59 into a hash table of length m = 11 utilizing open tending to with the essential hash work h' (k) = k mod m .Illustrate the aftereffect of embeddings these keys utilizing straight examining, utilizing quadratic testing with c1=1 and c2=3, and utilizing twofold hashing with h2(k) = 1 + (k mod (m-1)).

Arrangement: Using Linear Probing the last condition of hash table would be:

0	22
1	88
2	/
3	/
4	4
5	15
6	28
7	17
8	59
9	31
10	10

Using Quadratic Probing with c1=1, c2=3, the final state of hash table would be h (k, i) = (h' (k) +c1*i+ c2 *i2) mod m where m=11 and h' (k) = k mod m.

0	22
1	88
2	/
3	17
4	4
5	/
6	28
7	59
8	15
9	31
10	10

Using Double Hashing, the final state of the hash table would be:

0	22
1	/
2	59
3	17
4	4
5	15
6	28
7	88
8	/
9	31
10	10

4.5 Hash Tables

It is an assortment of things which are put away so as to make it simple to discover them later.

Each position in the hash table is called opening, can hold a thing and is named by a number worth beginning at 0.

The planning between a thing and a space where the thing should be in a hash table is known as a Hash Function. A hash Function acknowledges a key and returns its hash coding, or hash esteem.

Expect we have a bunch of whole numbers 54, 26, 93, 17, 77, 31. Our first hash work needed to be as "leftover portion strategy" just takes the thing and separation it by table size, returning remaining portion as its hash esteem for example

h item = item % (size of table)

Let us say the size of table = 11, then

54 % 11 = 10 26 % 11 = 4 93 % 11 = 5

17 % 11 = 6 77 % 11 = 0 31 % 11 = 9

ITEM	HASH VALUE
54	10
26	4
93	5
17	6
77	0
31	9

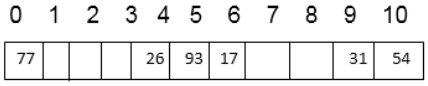


Fig: Hash Table

Now when we need to search any element, we just need to divide it by the table size, and we get the hash value. So we get the O (1) search time.

Now taking one more element 44 when we apply the hash function on 44, we get (44 % 11 = 0), But 0 hash value already has an element 77. This Problem is called as Collision.

Collision: According to the Hash Function, two or more item would need in the same slot. This is said to be called as Collision.

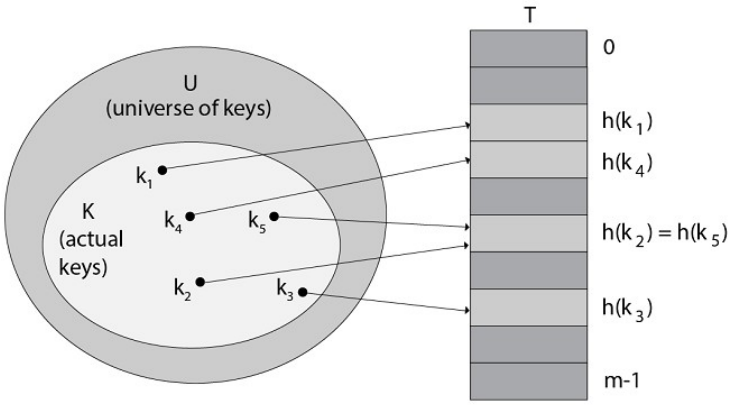


Figure: using a hash function h to map keys to hash-table slots. Because keys K2 and k5 map to the same slot, they collide.

4.6. Why use HashTable?

1. If U (Universe of keys) is large, storing a table T of size [U] may be impossible.
2. Set k of keys may be small relative to U so space allocated for T will waste.

So Hash Table requires less storage. Indirect addressing element with key k is stored in slot k with hashing it is stored in h (k) where h is a hash fn and hash (k) is the value of key k. Hash fn required array range.

4.7. Application of Hash Tables:

Some use of Hash Tables are:

1. Data set System: Specifically, those that are required effective arbitrary access. Generally, information base frameworks attempt to create between two sorts of access techniques: successive and arbitrary. Hash Table is an essential piece of proficient arbitrary access since they give an approach to find information in a consistent measure of time.
2. Image Tables: The tables used by compilers to keep up information about images from a program. Compilers access data about images much of the time. In this manner, it is fundamental that image tables be actualized productively.
3. Information Dictionaries: Data Structure that supports adding, erasing, and looking for information. Albeit the activity of hash tables and an information word reference are comparable, other Data Structures might be utilized to actualize information word references.
4. Cooperative Arrays: Associative Arrays comprise of information orchestrated so nth components of one exhibit compare to the nth component of another. Cooperative Arrays are useful for ordering a consistent gathering of information by a few key fields.

4.8 .Methods of Hashing

There are two main methods used to implement hashing:

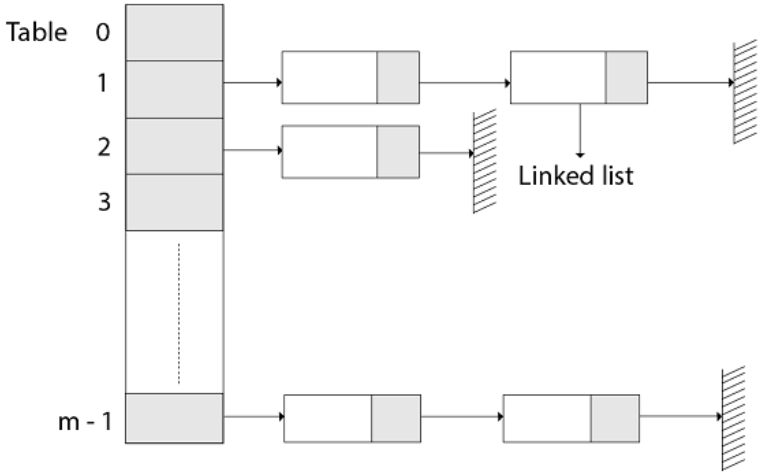
2.4.1.Hashing with Chaining

2.4.2.Hashing with open addressing

4.8.1. Hashing with Chaining

In Hashing with Chaining, the component in S is put away in Hash table T [0...m-1] of size m, where m is to some degree bigger than n, the size of S. The hash table is said to have m spaces. Related with the hashing plan is a hash work h which is planning from U to {0...m-1}.Each key k ∈S is put away in area T [h (k)], and we say that k is hashed into opening h (k). In the event that more than one key in S hashed into a similar opening, we have a crash.

In such case, all keys that hash into a similar space are put in a connected rundown related with that opening, this connected rundown is known as the chain at opening. The heap factor of a hash table is characterized to be a=n/m it addresses the normal number of keys per opening. We normally work in the reach m=θ(n), so a is typically a consistent by and large a<1.



4.8.2. Collision Resolution by Chaining:

In anchoring, we place all the components that hash to a similar opening into a similar connected rundown, As fig shows that Slot j contains a pointer to the top of the rundown of all put away components that hash to j ; if there are no such components, space j contains NIL.

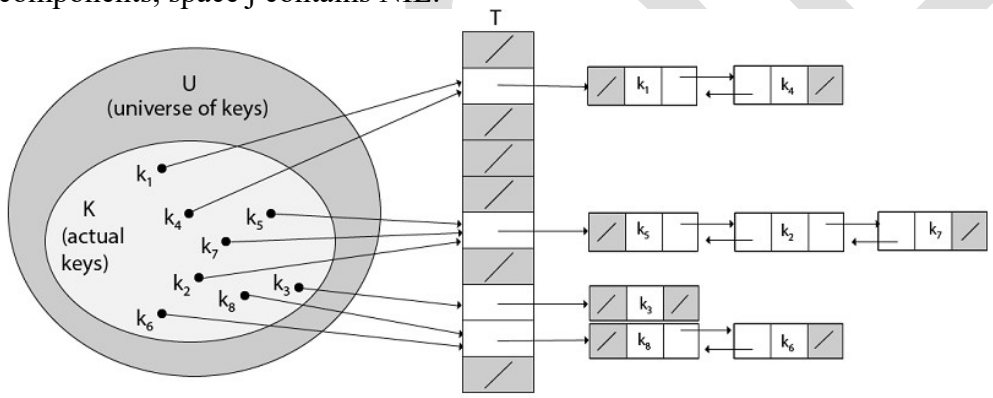


Fig: Collision resolution by chaining.

Each hash-table slot T [j] contains a linked list of all the keys whose hash value is j. For example, h (k1) = h (k4) and h (k5) = h (k7) =h (K2). The linked list can be either singly or doubly linked; we show it as doubly linked because deletion is faster that way.

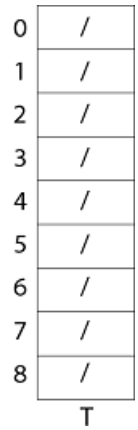
4.8.3. Analysis of Hashing with Chaining:

Given a hash table T with m spaces that stores n components, we characterize the heap factors α for T as n/m that is the normal number of components put away in a chain. The most pessimistic scenario running time for looking is in this manner θ(n) in addition to an opportunity to figure the hash work no better compared to on the off chance that we utilized one connected rundown for all the components. Obviously, hash tables are not utilized for their most pessimistic scenario execution.

The normal presentation of hashing relies upon how well the hash work h circulates the arrangement of keys to be put away among the m openings, all things considered.

Model: let us think about the inclusion of components 5, 28, 19,15,20,33,12,17,10 into an anchored hash table. Allow us to assume the hash table has 9 openings and the hash work be h (k) =k mod 9.

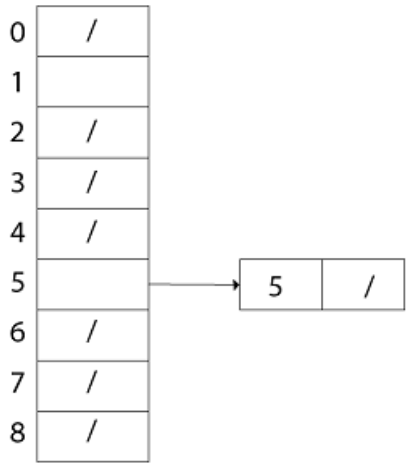
Arrangement: The underlying condition of tied hash table



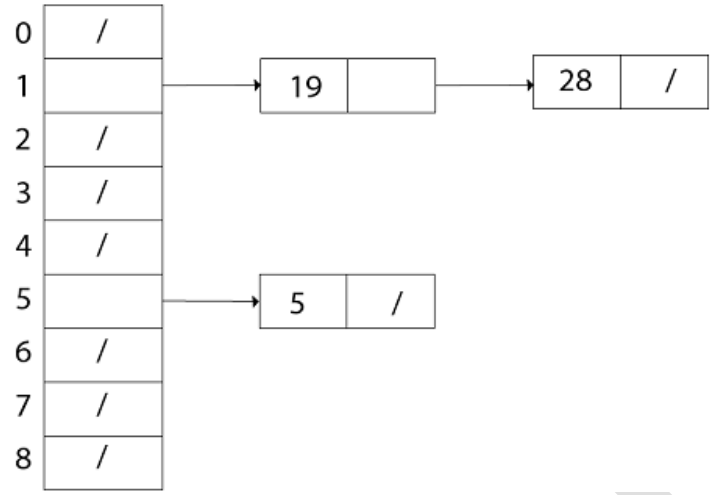
Insert 5:

h (5) = 5 mod 9 =5

Create a linked list for T [5] and store value 5 in it.



Similarly, insert 28. h (28) = 28 mod 9 =1. Create a Linked List for T [1] and store value 28 in it. Now insert 19 h (19) = 19 mod 9 = 1. Insert value 19 in the slot T [1] at the beginning of the linked-list.



Now insert h 15, h (15) = 15 mod 9 = 6. Create a link list for T [6] and store value 15 in it.

Similarly, insert 20, h (20) = 20 mod 9 = 2 in T [2].

Insert 33, h (33) = 33 mod 9 = 6

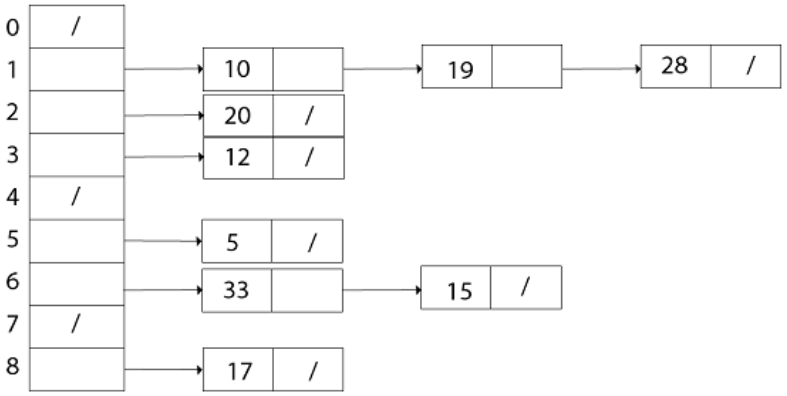
In the beginning of the linked list T [6]. Then,

Insert 12, h (12) = 12 mod 9 = 3 in T [3].

Insert 17, h (17) = 17 mod 9 = 8 in T [8].

Insert 10, h (10) = 10 mod 9 = 1 in T [1].

Thus the chained- hash- table after inserting key 10 is



4.9.Hashing with Open Addressing

In Open Addressing, all components are put away in hash table itself. That is, each table passage comprises of a segment of the powerful set or NIL. While looking for a thing, we reliably analyze table openings until possibly we locate the ideal article or we have verified that the component isn't in the table. Accordingly, in open tending to, the heap factor α can never surpass 1.

The upside of open tending to is that it dodges Pointer. In this, we figure the grouping of spaces to be inspected. The additional memory liberated by not sharing pointers furnishes the hash table with a bigger number of openings for a similar measure of memory, conceivably yielding less crash and quicker recovery.

The way toward looking at the area in the hash table is called Probing.

In this manner, the hash work becomes
h : U x {0,1,....m-1} → {0,1,. ..,m-1}.
With open addressing, we require that for every key k, the probe sequence

{h, (k, 0), h (k, 1). ..h (k, m-1)}
Be a Permutation of (0, 1 .....m-1)
The HASH-INSERT procedure takes as input a hash table T and a key k

HASH-INSERT (T, k)
1. i ← 0
2. repeat j ← h (k, i)
3. if T [j] = NIL
4. then T [j] ← k
5. return j
6. else ← i= i +1
7. untili=m
8. error "hash table overflow"

The procedure HASH-SEARCH takes as input a hash table T and a key k, returning j if it finds that slot j contains key k or NIL if key k is not present in table T.

HASH-SEARCH.T (k)
1. i ← 0
2. repeat j ← h (k, i)
3. if T [j] =j
4. then return j
5. i ← i+1
6. until T [j] = NIL or i=m
7. return NIL

0	1	2	3	4	5	6	7	8	9	10
/	65	21	/	26	37	/	/	/	59	76

4.10 Open Addressing Techniques

Three techniques are commonly used to compute the probe sequence required for open addressing:

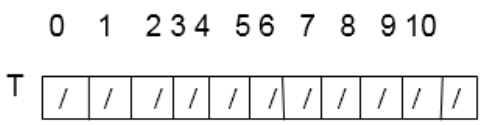
4.10.1. Linear Probing.

4.10.2.Quadratic Probing.

4.10.3.Double Hashing.

4.10.1. Linear Probing:

It is a Scheme in Computer Programming for settling impact in hash tables.
Assume another record R with key k is to be added to the memory table T however that the memory areas with the hash address H (k). H is as of now filled.
Our regular key to determine the impact is to intersection R to the most readily accessible area following T (h). We accept that the table T with m area is round, so T [i] comes after T [m].
The above impact goal is designated "Direct Probing".
Direct examining is easy to execute, yet it experiences an issue known as essential grouping. Long runs of involved spaces develop, expanding the normal pursuit time. Bunches emerge on the grounds that a vacant space continued by I full openings gets filled next with likelihood (I + 1)/m. Long runs of involved spaces will in general get longer, and the normal hunt time increments.
Given a normal hash work h': U {0, 1...m-1}, the technique for direct examining utilizes the hash work.
h (k, i) = (h' (k) + i) mod m
Where 'm' is the size of hash table and h' (k) = k mod m. for i=0, 1. m-1.
Given key k, the first slot is T [h' (k)]. We next slot T [h' (k) +1] and so on up to the slot T [m-1]. Then we wrap around to slots T [0], T [1] until finally slot T [h' (k)-1].
Since the initial probe position dispose of the entire probe sequence, only m distinct probe sequences are used with linear probing.
Example: Consider inserting the keys 24, 36, 58,65,62,86 into a hash table of size m=11 using linear probing, consider the primary hash function is h' (k) = k mod m.
Solution: Initial state of hash table



Insert 24. We know h (k, i) = [h' (k) + i] mod m
Now h (24, 0) = [24 mod 11 + 0] mod 11
= (2+0) mod 11 = 2 mod 11 = 2
Since T [2] is free, insert key 24 at this place.

Insert 36. Now h (36, 0) = [36 mod 11 + 0] mod 11
= [3+0] mod 11 = 3
Since T [3] is free, insert key 36 at this place.

Insert 58. Now h (58, 0) = [58 mod 11 +0] mod 11
= [3+0] mod 11 =3
Since T [3] is not free, so the next sequence is
h (58, 1) = [58 mod 11 +1] mod 11
= [3+1] mod 11= 4 mod 11=4
T [4] is free; Insert key 58 at this place.

Insert 65. Now h (65, 0) = [65 mod 11 +0] mod 11

= (10 +0) mod 11= 10

T [10] is free. Insert key 65 at this place.

Insert 62. Now h (62, 0) = [62 mod 11 +0] mod 11

= [7 + 0] mod 11 = 7

T [7] is free. Insert key 62 at this place.

Insert 86. Now h (86, 0) = [86 mod 11 + 0] mod 11

= [9 + 0] mod 11 = 9

T [9] is free. Insert key 86 at this place.

Thus,

	0	1	2	3	4	5	6	7	8	9	10
T	/	/	24	36	58	/	/	62	/	86	65

4.10.2.Quadratic Probing:

Assume a record R with key k has the hash address H (k) = h then as opposed to looking through the area with addresses h, h+1, and h+ 2...We directly search the areas with addresses

h, h+1, h+4, h+9...h+i2

Quadratic Probing uses a hash capacity of the structure

h (k,i) = (h' (k) + c1i + c2i2) mod m

Where (as in direct testing) h' is a helper hash work c1 and c2 ≠0 are assistant constants and i=0, 1...m-1. The underlying position is T [h' (k)]; later position tested is counterbalanced by the sum that depend in a quadratic way on the test number I.

Model: Consider embeddings the keys 74, 28, 36,58,21,64 into a hash table of size m =11 utilizing quadratic examining with c1=1 and c2=3. Further consider that the essential hash work is h' (k) = k mod m.

Arrangement: For Quadratic Probing, we have

h (k, i) = [k mod m +c1i +c2 i2) mod m

This is the initial state of hash table

Here c1= 1 and c2=3

h (k, i) = [k mod m + i + 3i2] mod m

Insert 74.

h (74,0)= (74 mod 11+0+3x0) mod 11
= (8 +0+0) mod 11 = 8
T [8] is free; insert the key 74 at this place.

Insert 28.

h (28, 0) = (28 mod 11 + 0 + 3 x 0) mod 11
= (6 +0 + 0) mod 11 = 6.
T [6] is free; insert key 28 at this place.

Insert 36.

h (36, 0) = (36 mod 11 + 0 + 3 x 0) mod 11
= (3 + 0+0) mod 11=3
T [3] is free; insert key 36 at this place.

Insert 58.

h (58, 0) = (58 mod 11 + 0 + 3 x 0) mod 11
= (3 + 0 + 0) mod 11 = 3
T [3] is not free, so next probe sequence is computed as
h (59, 1) = (58 mod 11 + 1 + 3 x12) mod 11
= (3 + 1 + 3) mod 11
=7 mod 11= 7
T [7] is free; insert key 58 at this place.

Insert 21.

h (21, 0) = (21 mod 11 + 0 + 3 x 0)
= (10 + 0) mod 11 = 10
T [10] is free; insert key 21 at this place.

Insert 64.

h (64, 0) = (64 mod 11 + 0 + 3 x 0)
= (9 + 0+ 0) mod 11 = 9.
T [9] is free; insert key 64 at this place.
Thus, after inserting all keys, the hash table is

0	1	2	3	4	5	6	7	8	9	10
/	/	/	36	/	/	28	58	74	64	21

4.10.3.Double Hashing:

Double Hashing is one of the best techniques available for open addressing because the permutations produced have many of the characteristics of randomly chosen permutations.

Double hashing uses a hash function of the form

h (k, i) = (h1(k) + i h2 (k)) mod m
Where h1 and h2 are auxiliary hash functions and m is the size of the hash table.

h1 (k) = k mod m or h2 (k) = k mod m'. Here m' is slightly less than m (say m-1 or m-2).

Example: Consider inserting the keys 76, 26, 37,59,21,65 into a hash table of size m = 11 using double hashing. Consider that the auxiliary hash functions are h1 (k)=k mod 11 and h2(k) = k mod 9.

Solution: Initial state of Hash table is

	0	1	2	3	4	5	6	7	8	9	10
T	/	/	/	/	/	/	/	/	/	/	/

1. Insert 76.
h1(76) = 76 mod 11 = 10
h2(76) = 76 mod 9 = 4
h (76, 0) = (10 + 0 x 4) mod 11
= 10 mod 11 = 10
T [10] is free, so insert key 76 at this place.

2. Insert 26.
h1(26) = 26 mod 11 = 4
h2(26) = 26 mod 9 = 8
h (26, 0) = (4 + 0 x 8) mod 11
= 4 mod 11 = 4
T [4] is free, so insert key 26 at this place.

3. Insert 37.
h1(37) = 37 mod 11 = 4
h2(37) = 37 mod 9 = 1
h (37, 0) = (4 + 0 x 1) mod 11 = 4 mod 11 = 4

T [4] is not free, the next probe sequence is

h (37, 1) = (4 + 1 x 1) mod 11 = 5 mod 11 = 5

T [5] is free, so insert key 37 at this place.

4. Insert 59.

h1(59) = 59 mod 11 = 4

h2(59) = 59 mod 9 = 5

h (59, 0) = (4 + 0 x 5) mod 11 = 4 mod 11 = 4

Since, T [4] is not free, the next probe sequence is

h (59, 1) = (4 + 1 x 5) mod 11 = 9 mod 11 = 9

T [9] is free, so insert key 59 at this place.

5. Insert 21.

h1(21) = 21 mod 11 = 10

h2(21) = 21 mod 9 = 3

h (21, 0) = (10 + 0 x 3) mod 11 = 10 mod 11 = 10

T [10] is not free, the next probe sequence is

h (21, 1) = (10 + 1 x 3) mod 11 = 13 mod 11 = 2

T [2] is free, so insert key 21 at this place.

6. Insert 65.

h1(65) = 65 mod 11 = 10

h2(65) = 65 mod 9 = 2

h (65, 0) = (10 + 0 x 2) mod 11 = 10 mod 11 = 10

T [10] is not free, the next probe sequence is

h (65, 1) = (10 + 1 x 2) mod 11 = 12 mod 11 = 1

T [1] is free, so insert key 65 at this place.

Thus, after insertion of all keys the final hash table is

4.11 Hash Function

Hash Function is utilized to list the first worth or key and afterward utilized later each time the information related with the worth or key is to be recovered. Along these lines, hashing is consistently a single direction activity. There is no compelling reason to "figure out" the hash work by examining the hashed values.

4.12Characteristics of Good Hash Function:

1. The hash esteem is completely dictated by the information being hashed.
2. The hash Function utilizes all the information.
3. The hash work "consistently" conveys the information across the whole arrangement of conceivable hash esteems.
4. The hash work creates convoluted hash esteems for comparative strings.

4.13 Some Popular Hash Function is:

4.13.1. Division Method: Pick a number m more modest than the quantity of n of keys in k (The number m is typically picked to be an indivisible number or a number without little divisors, since this often a base number of crashes).

The hash work is:

h (k) = k mod m	h (k) = k mod m + 1

For Example: if the hash table has size m = 12 and the key is k = 100, at that point h (k) = 4. Since it requires just a solitary division activity, hashing by division is very quick.

4.13.2. Duplication Method:

The increase technique for making hash capacities works in two stages. In the first place, we increase the vital k by a steady An in the reach $0 < A < 1$ and concentrate the fragmentary piece of kA. At that point, we increment this incentive by m and take the floor of the outcome.

The hash work is:

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

Where "k A mod 1" means the fractional part of k A, that is, k A -⌊k A⌋.

4.13.3. Mid Square Method:

The key k is squared. Then function H is defined by

H (k) = L

Where L is obtained by deleting digits from both ends of k2. We emphasize that the same position of k2 must be used for all of the keys.

4.13.4. Folding Method:

The key k is partitioned into a number of parts k1, k2.... kn 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.

H (k) = k1+ k2+ ....+kn

Example: Company has 68 employees, and each is assigned a unique four- digit employee number. Suppose L consist of 2- digit addresses: 00, 01, and 02… 99. We apply the above hash functions to each of the following employee numbers:

3205, 7148, 2345

(a) Division Method: Choose a Prime number m close to 99, such as m =97, Then

H (3205) = 4, H (7148) = 67, H (2345) = 17.

That is dividing 3205 by 17 gives a remainder of 4, dividing 7148 by 97 gives a remainder of 67, dividing 2345 by 97 gives a remainder of 17.

(b) Mid-Square Method:

k = 3205 7148 2345
k2= 10272025 51093904 5499025
h (k) = 72 93 99
Observe that fourth & fifth digits, counting from right are chosen for hash address.

(c) Folding Method:
Divide the key k into 2 parts and adding yields the following hash address:

H (3205) = 32 + 50 = 82 H (7148) = 71 + 84 = 55
H (2345) = 23 + 45 = 68