SYLLABUS OF SEMESTER-I, MCA (Artificial Intelligence and Machine Learning)

Course Code: 24CS60TH1177 Course: Data Structures

Unit - V :

Hashing: Introduction to Hashing, Different Hashing techniques, Collision handling mechanisms.

HASHING

The search time of each algorithm discussed so far depends on the number n of elements in the collection S of data. This section discusses a searching technique, called *hashing* or *hash addressing*, which is essentially independent of the number n.

The terminology which we use in our presentation of hashing will be oriented toward file management. First of all, we assume that there is a file F of n records with a set K of keys which uniquely determine the records in F. Secondly, we assume that F is maintained in memory by a table T of m memory locations and that L is the set of memory addresses of the locations in T. For notational convenience, we assume that the keys in K and the addresses in L are (decimal) integers. (Analogous methods will work with binary integers or with keys which are character strings, such as names, since there are standard ways of representing strings by integers.)

The general idea of using the key to determine the address of a record is an excellent idea, but it must be modified so that a great deal of space is not wasted. This modification takes the form of a function H from the set K of keys into the set L of memory addresses. Such a function,

$$H: K \rightarrow L$$

is called a *hash function* or *hashing function*. Unfortunately, such a function H may not yield distinct values: it is possible that two different keys k_1 and k_2 will yield the same hash address. This situation is called *collision*, and some method must be used to resolve it. Accordingly, the topic of hashing is divided into two parts: (1) hash functions and (2) collision resolutions.

Hash Functions

The two principal criteria used in selecting a hash function $H: K \to L$ are as follows. First of all, the function H should be very easy and quick to compute. Second the function H should, as far as possible, uniformly distribute the hash addresses throughout the set L so that there are a minimum number of collisions. Naturally, there is no guarantee that the second condition can be completely fulfilled without actually knowing beforehand the keys and addresses.

(a) *Division method.* Choose a number m larger than the number n of keys in K. (The number m is usually chosen to be a prime number or a number without small divisors, since this frequently minimizes the number of collisions.) The hash function H is defined by

$$H(k) = k \pmod{m} \qquad \text{or} \qquad H(k) = k \pmod{m} + 1$$

Here $k \pmod{m}$ denotes the remainder when k is divided by m. The second formula is used when we want the hash addresses to range from 1 to m rather than from 0 to $m - 1$.

(b) *Midsquare method.* The key k is squared. Then the hash function H is defined by

$$H(k) = l$$

where l is obtained by deleting digits from both ends of k^2. We emphasize that the same positions of k^2 must be used for all of the keys.

(c) *Folding method.* The key k is partitioned into a number of parts, $k_1, \ldots, k_r$, 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. That is,

$$H(k) = k_1 + k_2 + \cdots + k_r$$

where the leading-digit carries, if any, are ignored. Sometimes, for extra "milling," the even-numbered parts, $k_2, k_4, \ldots$, are each reversed before the addition.

Consider the company each of whose 68 employees is assigned a unique 4-digit employee number. Suppose L consists of 100 two-digit addresses: 00, 01, 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

$$H(3205) = 4, \quad H(7148) = 67, \quad H(2345) = 17$$

That is, dividing 3205 by 97 gives a remainder of 4, dividing 7148 by 97 gives a remainder of 67, and dividing 2345 by 97 gives a remainder of 17. In the case that the memory addresses begin with 01 rather than 00, we choose that the function $H(k) = k(\text{mod } m) + 1$ to obtain:

$$H(3205) = 4 + 1 = 5, \quad H(7148) = 67 + 1 = 68, \quad H(2345) = 17 + 1 = 18$$

(b) *Midsquare method.* The following calculations are performed:

k:	3205	7148	2345
k^2:	10 272 025	51 093 904	5 499 025
$H(k)$:	72	93	99

Observe that the fourth and fifth digits, counting from the right, are chosen for the hash address.

(c) *Folding method.* Chopping the key k into two parts and adding yields the following hash addresses:

$$H(3205) = 32 + 05 = 37, \quad H(7148) = 71 + 48 = 19, \quad H(2345) = 23 + 45 = 68$$

Observe that the leading digit 1 in $H(7148)$ is ignored. Alternatively, one may want to reverse the second part before adding, thus producing the following hash addresses:

$$H(3205) = 32 + 50 = 82, \quad H(7148) = 71 + 84 = 55, \quad H(2345) = 23 + 54 = 77$$

Collision Resolution Techniques

When two values hash to the same array location, this is called a collision. Collisions are normally treated as “first come, first served”—the first value that hashes to the location gets it.

We have to find something to do with the second and subsequent values that hash to this same location.

There are two broad ways of collision resolution:

1. Separate Chaining: An array of **linked list implementation**.

2. Open Addressing: **Array-based implementation**

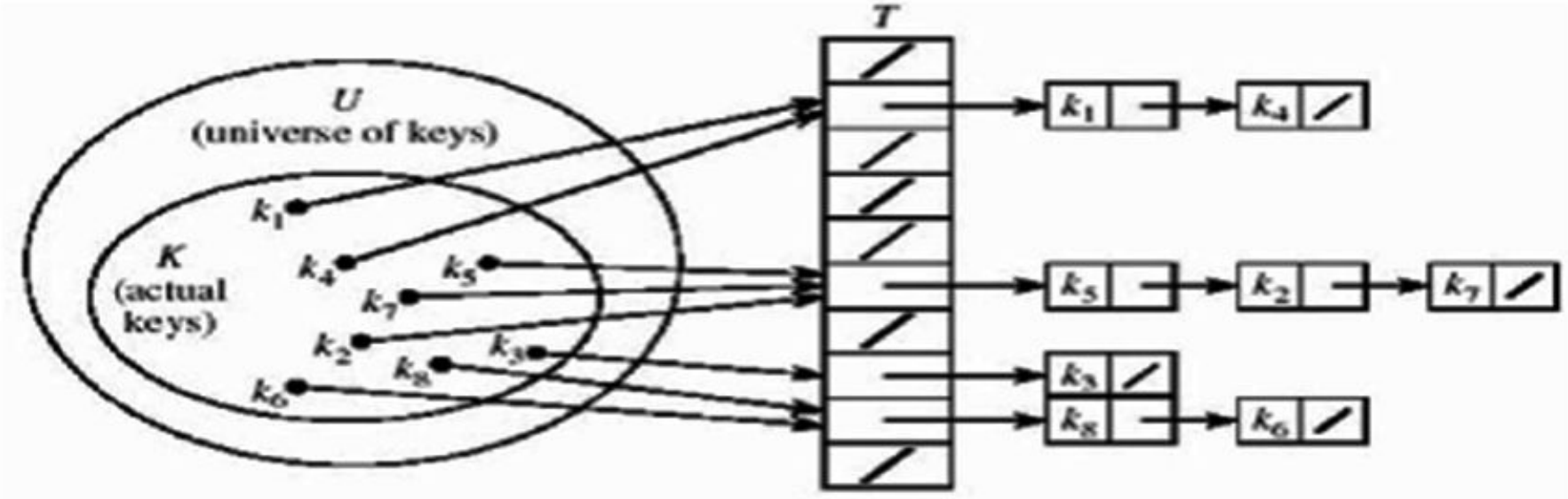
(i) **Linear probing** (linear search)

(ii) **Quadratic probing** (nonlinear search)

(iii) **Double hashing** (uses two hash functions)

Separate Chaining

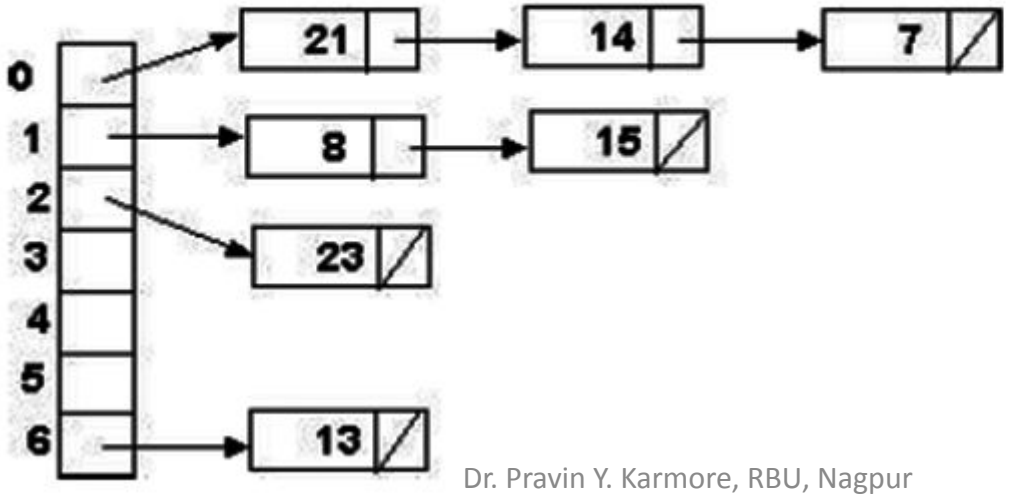
- The hash table is implemented as an array of linked lists.
- Inserting an item, r, which hashes at index i is simply insertion into the linked list at position i.
- Synonyms are chained in the same linked list



- Retrieval of an item, r, with hash address, i, is simply retrieval from the linked list at position i.
- Deletion of an item, r, with hash address, i, is simply deleting r from the linked list at position i

Example: Load the keys 23, 13, 21, 14, 7, 8, and 15 , in this order, in a hash table of size 7 using separate chaining with the hash function: h(key) = key % 7

h(23) = 23 % 7 = 2
h(13) = 13 % 7 = 6
h(21) = 21 % 7 = 0
h(14) = 14 % 7 = 0 collision
h(7) = 7 % 7 = 0 collision
h(8) = 8 % 7 = 1
h(15) = 15 % 7 = 1 collision



Open Addressing

- All items are stored in the hash table itself
- In addition to the cell data (if any), each cell keeps one of the three states: EMPTY. OCCUPIED, DELETED
- While inserting, if a collision occurs, alternative cells are tried until an empty cell is found.
- Deletion: (lazy deletion): When a key is deleted the slot is marked as DELETED rather than EMPTY otherwise subsequent searches that hash at the deleted cell will fail.
- Probe sequence: A probe sequence is the sequence of array indexes that is followed in searching for an empty cell during an insertion, or in searching for a key during find or delete operations.
- The most common probe sequences are of the form:

$$h_i(key) = [h(key) + c(i)] \% n,$$

for i = 0, 1, …, n-1. where h is a hash function and

n is the size of the hash table

The function c(i) is required to have the following two properties:

- Property 1: $c(0) = 0$
- Property 2: The set of values $\{c(0) \% n, c(1) \% n, c(2) \% n, \ldots, c(n-1) \% n\}$ must be a permutation of $\{0, 1, 2, \ldots, n-1\}$, that is, it must contain every integer between 0 and n - 1 inclusive
- The function c(i) is used to resolve collisions.
- To insert item r, we examine array location $h_0(r) = h(r)$. If there is a collision, array locations $h_1(r), h_2(r), \ldots, h_{n-1}(r)$ are examined until an empty slot is found
- Similarly, to find item r, we examine the same sequence of locations in the same order.
- Note: For a given hash function h(key), the only difference in the open addressing collision resolution techniques (linear probing, quadratic probing and double hashing) is in the definition of the function c(i).
- Common definitions of c(i) are:

Collision Resolution Technique	C(i)
Linear Probing	I
Quadratic Probing	$\pm i^2$
Double Hashing	$i*h_p(key)$

where $h_p(key)$ is another hash function.

Table size= smallest prime ≥ number of items in table/ desired load factor

Open Addressing: Linear Probing

- c(i) is a linear function in i of the form c(i) = a*i.
- Usually c(i) is chosen as:

$$c(i) = i \qquad \text{for } i = 0, 1, \ldots, tableSize - 1$$

- The probe sequences are then given by:

$$h_i(key) = [h(key) + i] \% tableSize \quad \text{for } i = 0, 1, \ldots, tableSize - 1$$

For c(i) = a*i to satisfy Property 2, a and n must be relatively prime.

e.g. Perform the operations given below, in the given order, on an initially empty hash table of size 13 using linear probing with c(i) = i and the hash function: h(key) = key % 13

insert(18), insert(26), insert(35), insert(9), find(15), find(48), delete(35), delete(40), find(9), insert(64), insert(47), find(35)

The required probe sequences are given by

$$h_i(key) = (h(key) + i) \% 13 \qquad i = 0, 1, 2, \ldots, 12$$

OPERATION	PROBE SEQUENCE	COMMENT
insert(18)	$h_0(18) = (18 \% 13) \% 13 = 5$	SUCCESS
insert(26)	$h_0(26) = (26 \% 13) \% 13 = 0$	SUCCESS
insert(35)	$h_0(35) = (35 \% 13) \% 13 = 9$	SUCCESS
insert(9)	$h_0(9) = (9 \% 13) \% 13 = 9$	COLLISION
	$h_1(9) = (9+1) \% 13 = 10$	SUCCESS
find(15)	$h_0(15) = (15 \% 13) \% 13 = 2$	FAIL because location 2 has **Empty** status
find(48)	$h_0(48) = (48 \% 13) \% 13 = 9$	COLLISION
	$h_1(48) = (9 + 1) \% 13 = 10$	COLLISION
	$h_2(48) = (9 + 2) \% 13 = 11$	FAIL because location 11 has **Empty** status
withdraw(35)	$h_0(35) = (35 \% 13) \% 13 = 9$	SUCCESS because location 9 contains 35 and the status is **Occupied** The status is changed to **Deleted**; but the key 35 is not removed.
find(9)	$h_0(9) = (9 \% 13) \% 13 = 9$	The search continues, location 9 does not contain 9; but its status is **Deleted**
	$h_1(9) = (9+1) \% 13 = 10$	SUCCESS
insert(64)	$h_0(64) = (64 \% 13) \% 13 = 12$	SUCCESS
insert(47)	$h_0(47) = (47 \% 13) \% 13 = 8$	SUCCESS
find(35)	$h_0(35) = (35 \% 13) \% 13 = 9$	FAIL because location 9 contains 35 but its status is **Deleted**

0	26
1	
2	
3	
4	
5	18
6	
7	
8	47
9	35
10	9
11	
12	64

Quadratic probing:

$f(i) = i^2$

$H(x) = x \bmod Tablesize$

$H_i(x) = (H(x) + i^2) \bmod Tablesize$

Ex. Insert 3, 5, 13, 24, 45, 33, 54 in table of 10 using quadratic probing.

0	
1	
2	
3	3
4	13
5	5
6	45
7	33
8	24
9	

$H_0(3) = (3 + 0) \bmod 10 = 3$

$H_0(5) = (5 + 0) \bmod 10 = 5$

$H_0(13) = (3 + 0) \bmod 10 = 3$

$H_1(13) = (3 + (1)^2) \bmod 10 = 4$

$H_0(24) = (4 + 0) \bmod 10 = 4$

$H_1(24) = (4 + (1)^2) \bmod 10 = 5$

$H_2(24) = (4 + (2)^2) \bmod 10 = 8$

$H_0(45) = (5 + 0) \bmod 10 = 5$

$H_1(45) = (5 + (1)^2) \bmod 10 = 6$

$H_0(33) = (3 + 0) \bmod 10 = 3$

$H_1(33) = (3 + (1)^2) \bmod 10 = 4$

$H_2(33) = (3 + (2)^2) \bmod 10 = 7$

Double hashing:

$f(i) = i * hash_2(x)$

$H(x) = x \bmod Tablesize$

$H_2(x) = R - x \bmod R$

where $R = Primenumber < Tablesize$

Rule:

1. **The hash function must $hash_2(x)$ never evaluate to zero.**
2. **All cells must be probed.**

Ex. Insert 89, 18, 49, 58, 69 to table of 10 using double hashing.

	89	18	49	58	69
0					69
1					
2					
3				58	58
4					
5					
6			49	49	49
7					
8		18	18	18	18
9	89	89	89	89	89

H(89) = 89 mod 10 = 9

H(18) = 18 mod 10 = 8

H(49) = 49 mod 10 = 9

$H_1(49) = (9 + H_2(49)) \bmod 10$

$H_2(49) = (7 - 49 \bmod 7) = 7$

$H_1(49) = (9 + 1 * 7) \bmod 10 = 6$

H(58) = 58 mod 10 = 8

$H_1(58) = (8 + H_2(58)) \bmod 10$

$H_2(58) = (7 - 58 \bmod 7) = 5$

$H_1(58) = (8 + 1 * 5) \bmod 10 = 3$

H(69) = 69 mod 10 = 9

$H_1(69) = (9 + H_2(69)) \bmod 10$

$H_2(69) = (7 - 69 \bmod 7) = 1$

$H_1(69) = (9 + 1 * 1) \bmod 10 = 0$