DSA UNIT – 6 SEARCHING

DEPARTMENT OF ICT ENGINEERING
MARWADI UNIVERSITY
RAJKOT

HASH Tables

- **Hash table** is a data structure in which keys are mapped to array positions by a hash function.
- we map the keys to array locations or array indices. A value stored in a hash table can be searched in O(1) time by using a hash function which generates an address from the key

HASH Function

- A **hash function** is a mathematical formula which, when applied to a key, produces an integer which can be used as an index for the key in the hash table.
- The main aim of a hash function is that elements should be **relatively, randomly, and uniformly distributed**.

HASH Function

Properties of a Good Hash Function:

- Low cost
- Determinism
- Uniformity

HASH Function

1. **Division Method**
2. **Multiplication Method**
3. **Mid-Square Method**
4. **Folding Method**

HASH Function

Division Method

example 15.1

Calculate the hash values of keys 1234 and 5462.

Setting M = 97, hash values can be calculated as:

h(1234) = 1234 % 97 = 70

h(5642) = 5642 % 97 = 16

HASH Function

Multiplication Method

example 15.2

Given a hash table of size 1000, map the key 12345 to an appropriate location in the hash table.

We will use A = 0.618033, m = 1000, and k = 12345

h(12345) => [1000 (12345 x 0.618033 mod 1)]

=> [1000 (7629.617385 mod 1)]

=> [1000 (0.617385)] => 617.385 => **617**

HASH Function

Mid-Square Method

example 15.3

Calculate the hash value for keys 1234 and 5642 using the mid-square method. The hash table has 100 memory locations.

Note that the hash table has 100 memory locations whose indices vary from 0 to 99. This means that only two digits are needed to map the key to a location in the hash table,

so r = 2.

When k = 1234, k2 = 1522756, h (1234) = 27

When k = 5642, k2 = 31832164, h (5642) = 21

3rd and 4th From Right are selected

HASH Function

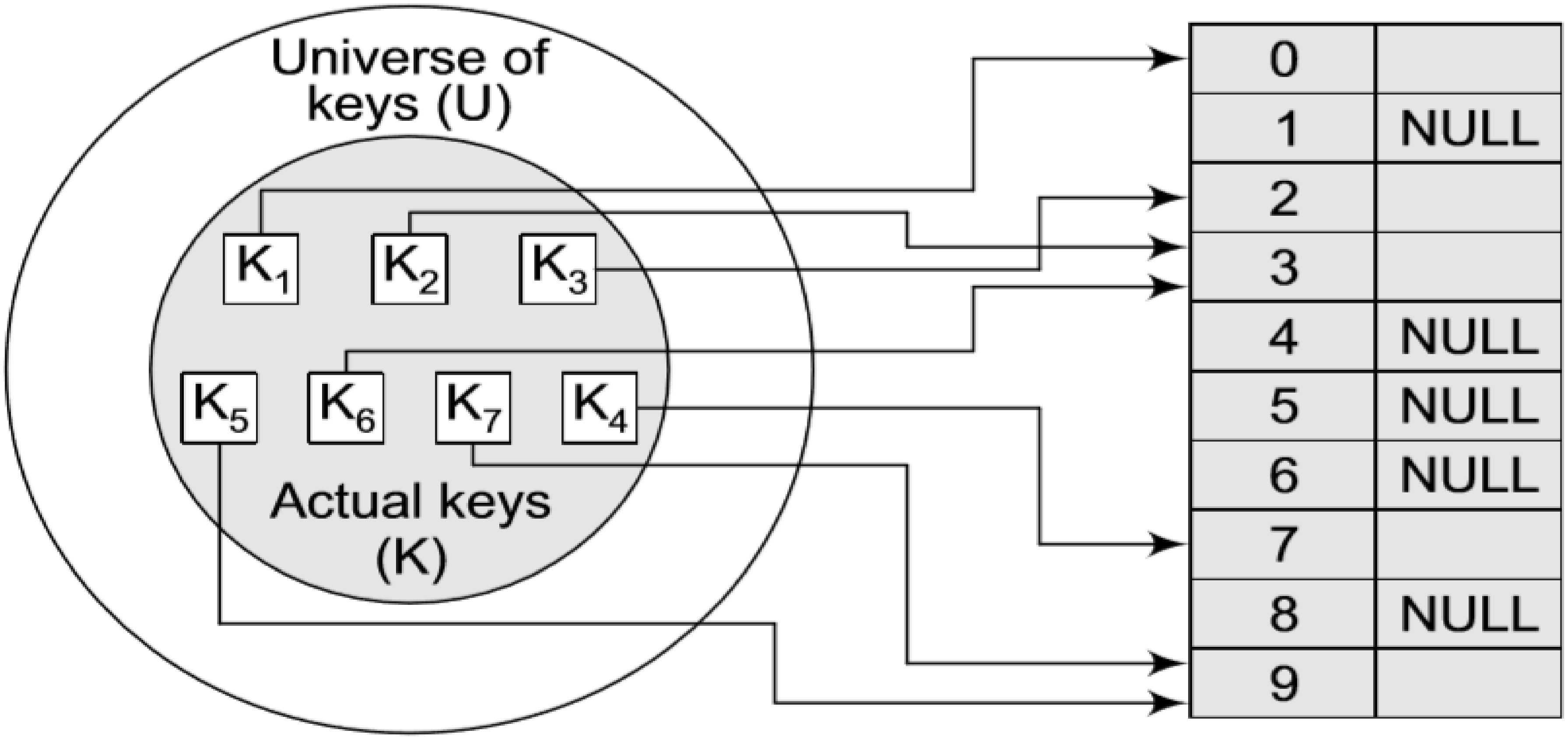
Folding Method

example 15.4

Given a hash table of 100 locations, calculate the hash value using folding method for keys 5678, 321, and 34567.

Since there are 100 memory locations to address, we will break the key into parts where each part (except the last) will contain two digits. The hash values can be obtained as shown below:

key	5678	321	34567
Parts	56 and 78	32 and 1	34, 56 and 7
Sum	134	33	97
Hash value	34 (ignore the last carry)	33	97



- Hash table in which each key from the set K is mapped to locations generated by using a hash function. Note that keys k2 and k6 point to the same memory location. ***This is known as collision***.

Collisions Resolution

- The hash table contains two types of values:
  1. Sentinel values (e.g., –1) and
  2. Data values.
- The presence of a sentinel value indicates that the location contains no data value at present but can be used to hold a value.

Collisions Resolution

- Collisions occur when the hash function maps two different keys to the same location.
- Obviously, two records cannot be stored in the same location.
- Therefore, a method used to solve the problem of collision, also called collision resolution technique, is applied.
- The two most popular methods of resolving collisions are: 1. Open addressing 2. Chaining

Open Addressing

- The process of examining memory locations in the hash table is called probing.
- Open addressing technique can be implemented using

1. Linear probing,
2. Quadratic probing,
3. Double hashing, and
4. Rehashing.

Linear Probing

- The simplest approach to resolve a collision is linear probing.
- In this technique, if a value is already stored at a location generated by $h(k)$, then the following hash function is used to resolve the collision:
- $h(k, i) = [h'(k) + i] \bmod m$
- Where m is the size of the hash table, $h'(k) = (k \bmod m)$, and **i** is the probe number that varies from **0 to m–1**.

Example 15.5 Consider a hash table of size 10. Using linear probing, insert the keys 72, 27
36, 24, 63, 81, 92, and 101 into the table.

Let $h'(k) = k \bmod m$, $m = 10$

Initially, the hash table can be given as:

0	1	2	3	4	5	6	7	8	9
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1

Step 1 Key = 72

$$h(72, 0) = (72 \bmod 10 + 0) \bmod 10$$
$$= (2) \bmod 10$$
$$= 2$$

0	1	2	3	4	5	6	7	8	9
0	81	72	63	24	–1	36	27	–1	–1

Step 7

```
        Key = 92
h(92, 0) = (92 mod 10 + 0) mod 10
         = (2) mod 10
         = 2
```

Now `T[2]` is occupied, so we cannot store the key 92 in `T[2]`. Therefore, try again for the next location. Thus probe, `i = 1`, this time.

```
        Key = 92
h(92, 1) = (92 mod 10 + 1) mod 10
         = (2 + 1) mod 10
         = 3
```

Now `T[3]` is occupied, so we cannot store the key 92 in `T[3]`. Therefore, try again for the next location. Thus probe, `i = 2`, this time.

```
        Key = 92
h(92, 2) = (92 mod 10 + 2) mod 10
         = (2 + 2) mod 10
         = 4
```

Now `T[4]` is occupied, so we cannot store the key 92 in `T[4]`. Therefore, try again for the next location. Thus probe, `i = 3`, this time.

```
        Key = 92
h(92, 3) = (92 mod 10 + 3) mod 10
         = (2 + 3) mod 10
         = 5
```

Since `T[5]` is vacant, insert key 92 at this location.

Quadratic probing

- In this technique, if a value is already stored at a location generated by h(k), then the following hash function is used to resolve the collision:
- $h(k, i) = [h'(k) + c1\ i + c2\ i^2]\ \text{mod}\ m$
- where m is the size of the hash table,
- $h'(k) = (k\ \text{mod}\ m)$,
- i is the probe number that varies from 0 to m–1, and c1 and c2 are constants such that c1 and c2 #(not equal) 0.

Example 15.6 Consider a hash table of size 10. Using quadratic probing, insert the keys 72, 27, 36, 24, 63, 81, and 101 into the table. Take $c_1 = 1$ and $c_2 = 3$.

Solution

Let $h'(k) = k \bmod m$, $m = 10$

Initially, the hash table can be given as:

0	1	2	3	4	5	6	7	8	9
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1

We have,

$$h(k, i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$$

Step 1 Key = 72

$$\begin{aligned} h(72, 0) &= [72 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10 \\ &= [72 \bmod 10] \bmod 10 \\ &= 2 \bmod 10 \\ &= 2 \end{aligned}$$

Since T[2] is vacant, insert the key 72 in T[2]. The hash table now becomes:

0	1	2	3	4	5	6	7	8	9
–1	–1	72	–1	–1	–1	–1	–1	–1	–1

Double Hashing

- To start with, double hashing **uses one hash value** and then repeatedly steps forward an interval until an empty location is reached. The interval is decided using a second, independent hash function hence the name double hashing.
- In double hashing, we use **two hash functions** rather than a single function.

Example 15.7 Consider a hash table of size = 10. Using double hashing, insert the keys 7
7, 36, 24, 63, 81, 92, and 101 into the table. Take $h_1 = (k \bmod 10)$ and $h_2 = (k \bmod 8)$.

Solution

Let $m = 10$

Initially, the hash table can be given as:

0	1	2	3	4	5	6	7	8	9
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1

We have,

$$h(k, i) = [h_1(k) + ih_2(k)] \bmod m$$

Step 1 Key = 72

$$\begin{aligned} h(72, 0) &= [72 \bmod 10 + (0 \times 72 \bmod 8)] \bmod 10 \\ &= [2 + (0 \times 0)] \bmod 10 \\ &= 2 \bmod 10 \\ &= 2 \end{aligned}$$

Step 7 Key = 92

$$h(92, 0) = [92 \bmod 10 + (0 \times 92 \bmod 8)] \bmod 10$$
$$= [2 + (0 \times 4)] \bmod 10$$
$$= 2 \bmod 10$$
$$= 2$$

Now T[2] is occupied, so we cannot store the key 92 in T[2]. Therefore, try again for the nex location. Thus probe, $i = 1$, this time.

Key = 92

$$h(92, 1) = [92 \bmod 10 + (1 \times 92 \bmod 8)] \bmod 10$$

$$= [2 + (1 \times 4)] \bmod 10$$

$$= (2 + 4) \bmod 10$$

$$= 6 \bmod 10$$

$$= 6$$

Now T[6] is occupied, so we cannot store the key 92 in T[6]. Therefore, try again for the next location. Thus probe, i = 2, this time.

$$\text{Key} = 92$$

$$h(92, 2) = [92 \bmod 10 + (2 \times 92 \bmod 8)] \bmod 10$$

$$= [2 + (2 \times 4)] \bmod 10$$

$$= [2 + 8] \bmod 10$$

$$= 10 \bmod 10$$

$$= 0$$

Since T[0] is vacant, insert the key 92 in T[0]. The hash table now becomes:

Rehashing

The hash function used is h(x) = x % 5. Rehash the entries into to a new hash table. Now, rehash the key values from the old hash table into the new one using hash function h(x) = x % 10.

0	1	2	3	4
	26	31	43	17

Note that the new hash table is of 10 locations, double the size of the original table.

0	1	2	3	4	5	6	7	8	9

Now, rehash the key values from the old hash table into the new one using hash function—`h(x) = x % 10`.

0	1	2	3	4	5	6	7	8	9
	31		43			26	17		

Chaining

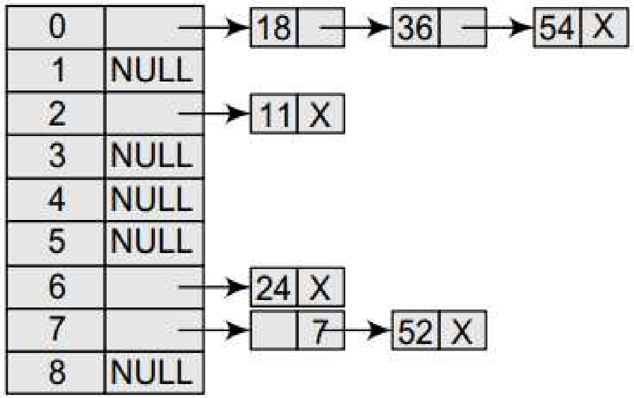
Example 15.8

Insert the keys 7, 24, 18, 52, 36, 54, 11, and 23 in a chained hash table of 9 memory locations. Use h(k) = k mod m. In this case, m=9.

Step 7: Key = 11

h(k)= 11 mod 9 = 2

Create a linked list for location 2 and store the key value 11 in it as its only node.



Step 8: Key = 23

h(k) = 23 mod 9 = 5

Create a linked list for location 5 and store the key value 23 in it as its only node.