HASHING

Day 1

General Guideline

HASHING

Introduction

- Hashing is a technique that is ***used*** to ***uniquely identify a specific object*** from a ***group of similar objects***.
- Some examples of how hashing is used in our lives include:
  - In universities, each student is assigned a unique roll number that can be used to retrieve information about them.
  - In libraries, each book is assigned a unique number that can be used to determine information about the book, such as its exact position in the library or the users it has been issued to etc.
- In both these examples the students and books were hashed to a ***unique number***.

Hashing

- Assume that you have an ***object*** and you want to ***assign a key*** to it to make ***searching easy***. To store the key/value pair, you can use a ***simple array*** like a data structure where keys (integers) can be used ***directly as an index*** to store values. However, in cases where the ***keys are large*** and cannot be used directly as an index, you should use ***hashing***.

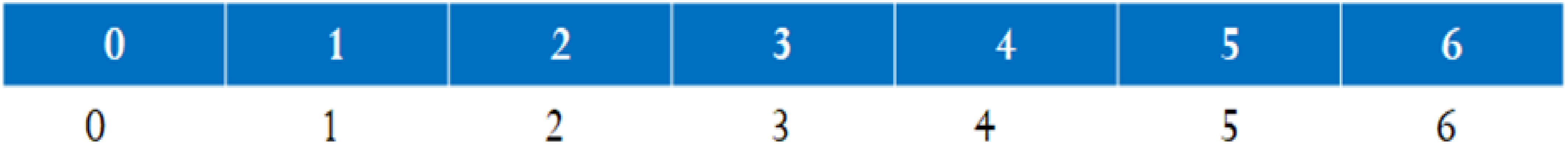
Hashing

- Search complexity dependent on input size. If we don’t want our search dependent on input size, then we use hashing.

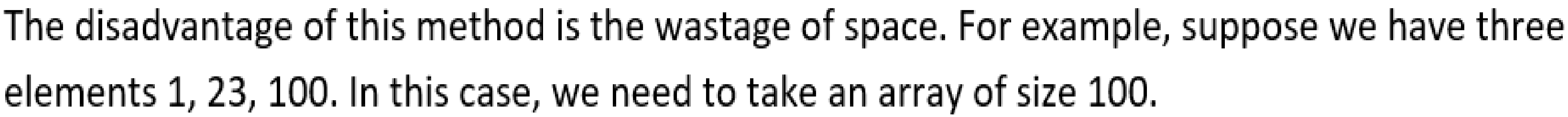
- H: K-→ L
- Apply hash functions on key k (search element) to find the location L.
- It provides constant time searching technique.

Hashing

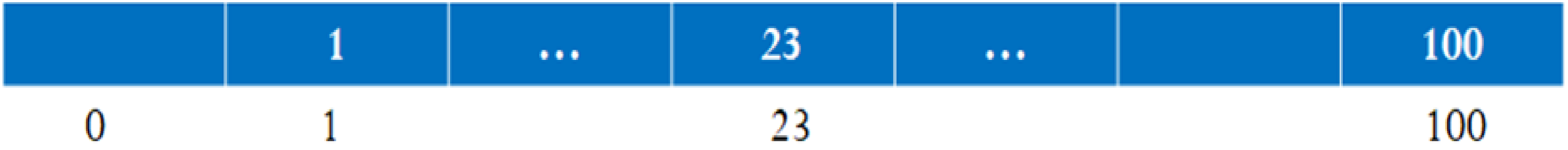
- if there are four elements 3, 5, 7, 2. Then in order to search any of these, we can take array of size 7 and can search in O(1) constant time



0	1	2	3	4	5	6
0	1	2	3	4	5	6



The disadvantage of this method is the wastage of space. For example, suppose we have three elements 1, 23, 100. In this case, we need to take an array of size 100.



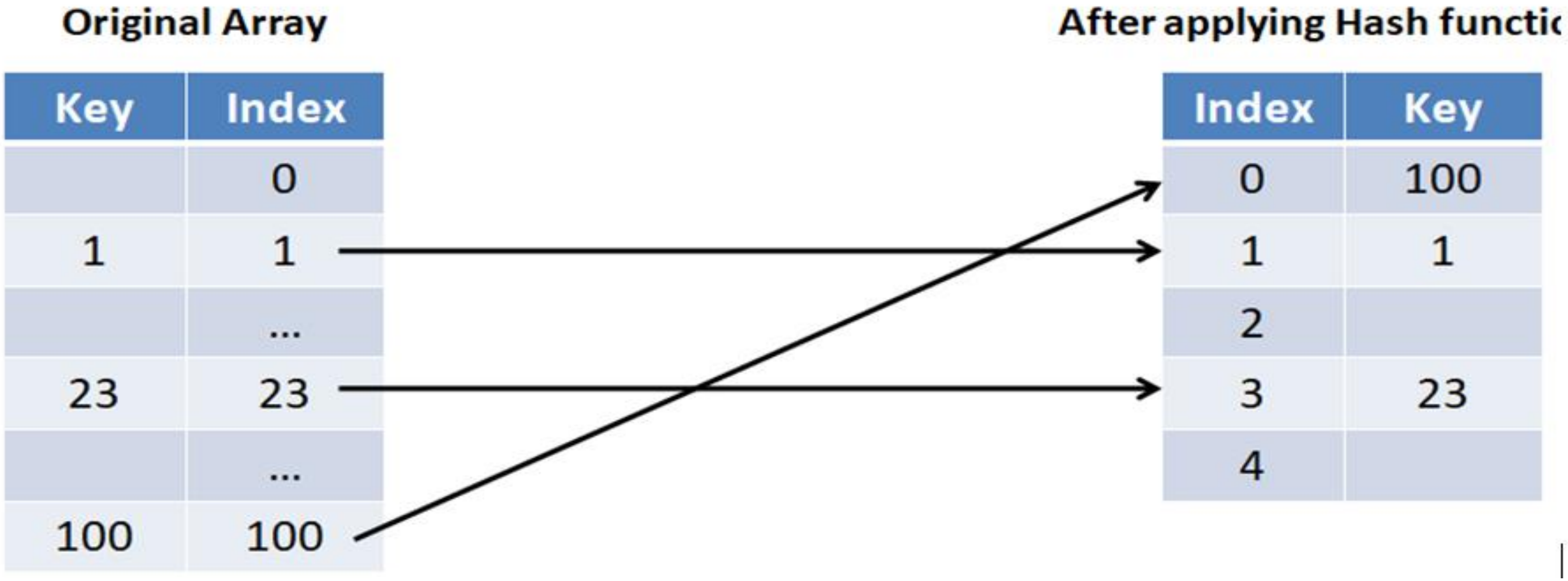
	1	...	23	...		100
0	1		23			100

Hashing

- In ***hashing***, ***large keys*** are ***converted*** into ***small keys*** by using ***hash functions***.
- The values are then ***stored in a data structure*** called ***hash table***.
- The ***idea*** of hashing is to ***distribute entries*** (key/value pairs) ***uniformly across an array***.
- Each element is assigned a key (converted key).
- By using that key you can access the element in **O(1)** time. Using the key, the ***algorithm*** (hash function) computes an index that suggests where an entry can be ***found*** or ***inserted***.

Hashing

The above elements are mapped using hash function H(x) = H(x) mod 5



Original Array

Key	Index
	0
1	1
	...
23	23
	...
100	100

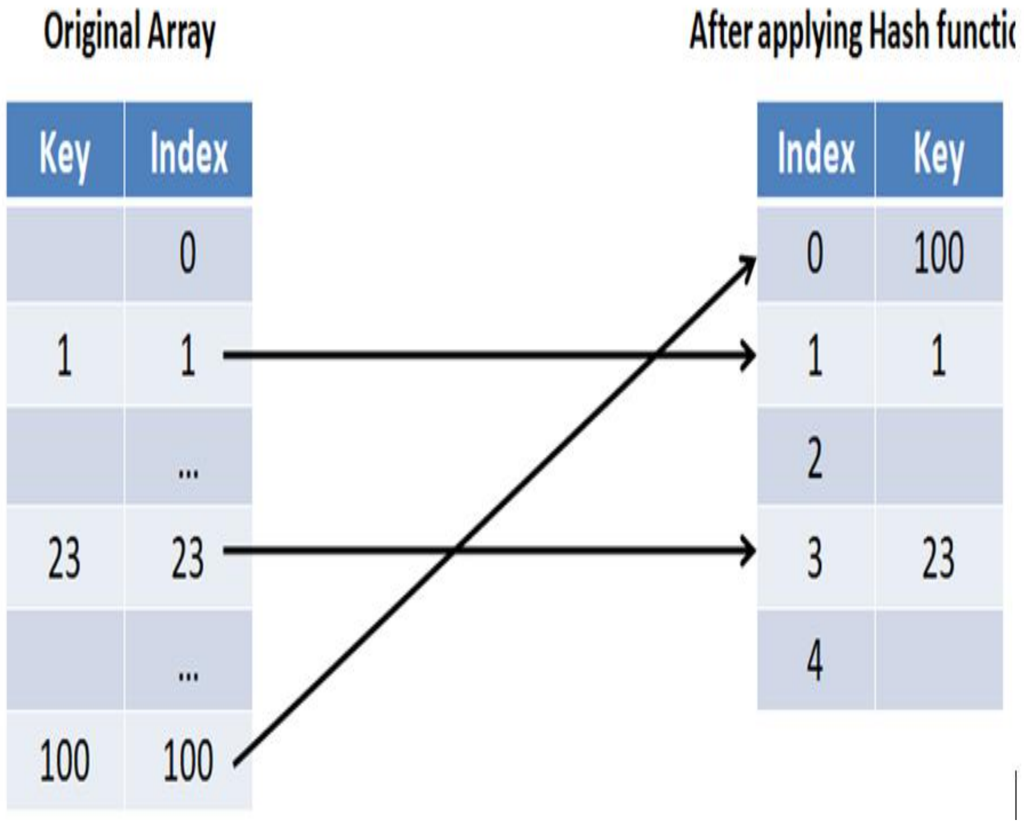
After applying Hash functic

Index	Key
0	100
1	1
2	
3	23
4	

Suppose we want to search a topic hashing in a Data Structure Book. Then, instead of using linear search or binary search technique, we can directly use the help of the index page and can see its exact page number and search this in O(1) time.

Definition of Hashing

- The process of transforming an element into a secret element using a hash code is known as hashing.
- The mathematical function used for transforming an element into a mapped one or to secret code is known as a Hash Function.



Original Array

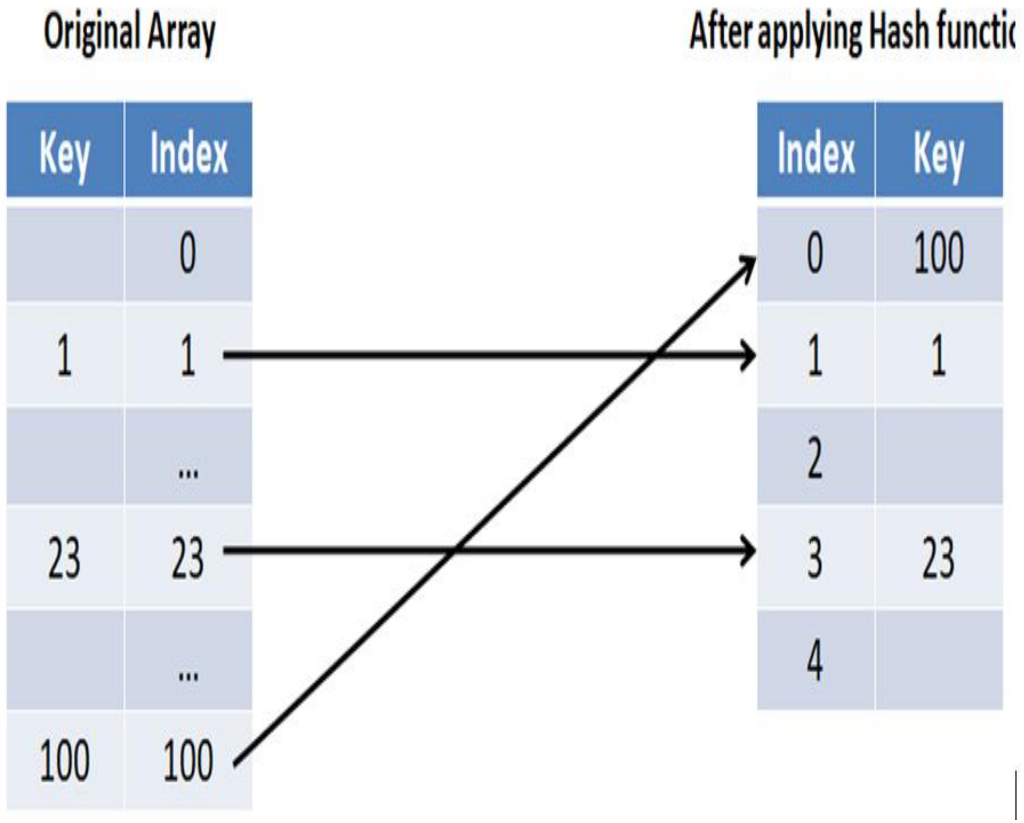
Key	Index
	0
1	1
	...
23	23
	...
100	100

After applying Hash functi

Index	Key
0	100
1	1
2	
3	23
4	

Definition of Hashing

- Hash Table: A structure that maps keys to Values.



Original Array

Key	Index
	0
1	1
	...
23	23
	...
100	100

After applying Hash functic

Index	Key
0	100
1	1
2	
3	23
4	

Hashing Functions

Three Types

1. Division Method
2. Mid Square Method
3. Folding Method

Hashing Functions

Modulo Method (Division Modulo)

This method computes hash code using the division method. The simple formula for calculating hash code is given by

Hash Code(key) = key mod HashTableSize

The size of Hash Table is decided based on the input data set.

Input elements as 35, 44, 22, 19, 11, 20, 43, 6, 88, 27

Hash code(35) = 35 mod 10 = 5,

Hash code(44) = 44 mod 10 = 4

0	1	2	3	4	5	6	7	8	9
20	11	22	43	44	35	6	27	88	19

Hashing Functions

Modulo Method (Division Modulo)

Input elements as 35, 44, 22, 19, 11, 20, 43, 6, 88, 27

Hash Location: 1 digit

Divide the key by 10.

Hash code(35) = 35 mod 10 = 5,

Hash code(44) = 44 mod 10 = 4

0	1	2	3	4	5	6	7	8	9
20	11	22	43	44	35	6	27	88	19

Hashing Functions

Consider a Hash Table of Size 100,

The Hash code for 123,223,323,423

123 mod 100 =23

223 mod 100 =23

323 mod 100 =23

423 mod 100 =23

Several key values might have same location. This actually **collision.**

We cannot store these keys at the same location in the Hash Table

WHAT CAN BE DONE IN SUCH CASE??

Hashing Functions

WHAT CAN BE DONE IN SUCH CASE??

We find the number that has least chance of collision.

So, we chose prime no. nearest to upper bound of an array.

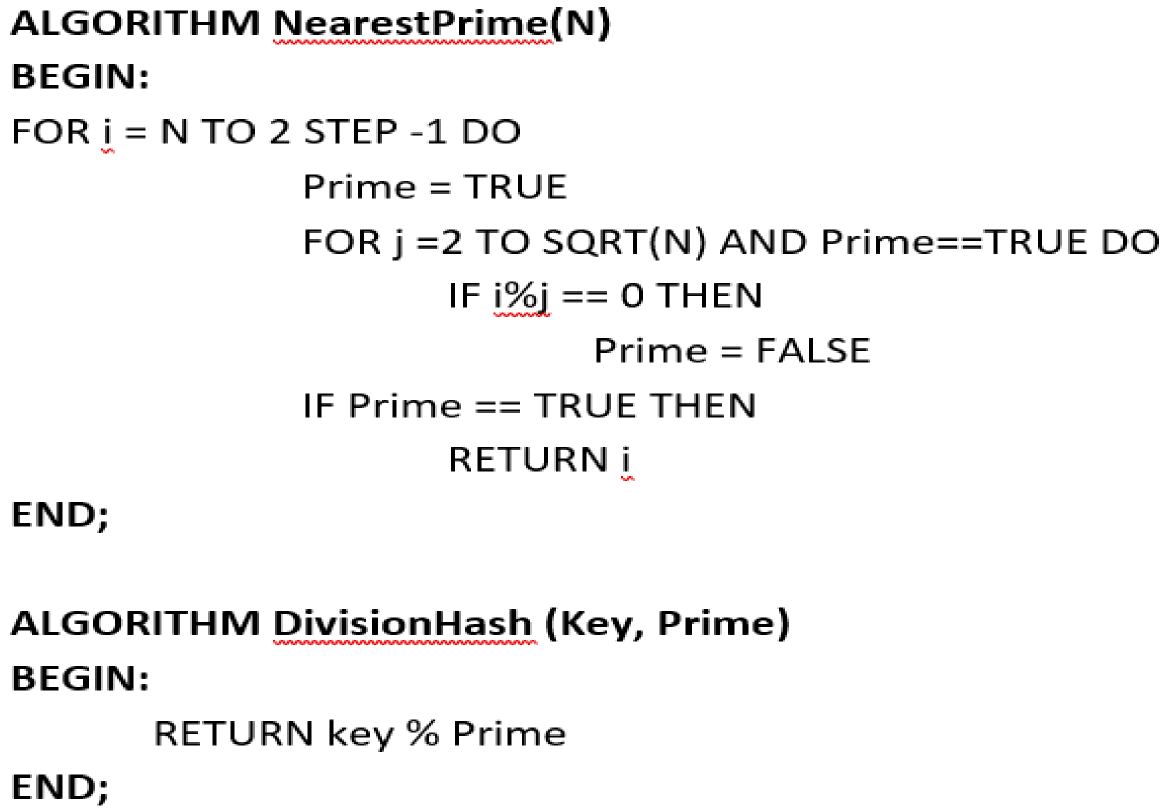
For above eg. We chose prime number nearest to 100.

So , we select 97.

K%97

Hashing Functions

We can divide the keys with some number with the least factors, a Prime number



```
ALGORITHM NearestPrime(N)
BEGIN:
FOR i = N TO 2 STEP -1 DO
        Prime = TRUE
        FOR j =2 TO SQRT(N) AND Prime==TRUE DO
            IF i%j == 0 THEN
                Prime = FALSE
        IF Prime == TRUE THEN
            RETURN i
END;

ALGORITHM DivisionHash (Key, Prime)
BEGIN:
    RETURN key % Prime
END;
```

Disadvantage: This method may suffer from the collision. Two elements when converted to hash function, if result in having one hash code then collision is said to have occurred.

Hashing Functions

Mid Square Method

unique key is extracted from the middle of the square of the key

If the number of digits of the highest possible index in the chosen hash table (k), then this hashing process suggests picking k digits from the square of the keys to act as the hash code (if the hash table size is in the powers of 10) else modulus is taken of these mid k digits with the table size.

Key	104
Key^2	10816
H(K)	10816

Key	4012
Key^2	16096144
H(K)	16096144

Hashing Functions

Mid Square Method

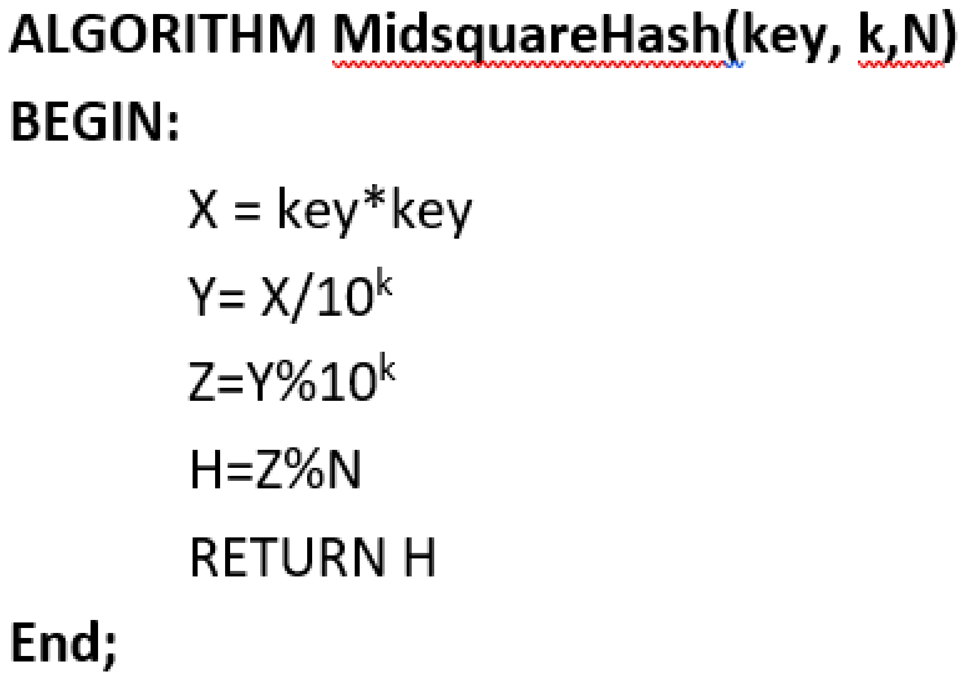
Key	7394
Key^2	54671236
H(K)	54671236

Key	2309
Key^2	5331481
H(K)	5331481

In case of odd numbers(h(K)): more digits discard from the right side.

Hashing Functions

hash table having size N. Each hash table location has an address of k digits.



```
ALGORITHM MidsquareHash(key, k,N)
BEGIN:
        X = key*key
        Y= X/$10^k$
        Z=Y%$10^k$
        H=Z%N
        RETURN H
End;
```

Hashing Functions

Folding Method

- Divide the key into equal size of pieces (of the same length as that of the length of the largest address in the table size) and then these are added together.
- Modulus is taken of sum with the table size, which results in the hash code.

eg.

H(k) = sum mod N

Hashing Functions

Folding Method

eg.

2.

Table size (N) be 1000, i.e. addresses will range from 0 – 999. The largest address is of 3 digit. If the key is 12345678, breaking it down into groups of 3 digits each.

$$12 + 345 + 678 = 1035$$

Hash code = 1035 % 1000 = 35

Hashing Functions

Folding Method (Variations)

- Divide the key into equal size of pieces.
- Even number parts (k2,k4,k6…..). Each even parts Reversed before addition.
- Modulus is taken of sum with the table size, which results in the hash code.

H(k) = sum mod N

eg.

1. Table size (N) be 100, i.e. addresses will range from 0 – 99. The largest address is of 2 digits. If the key is 1503291547, breaking it down into groups of 2 digits each.

15 + 03 + 29 + 15 + 47

15 + 30 + 29 + 51 + 47 =172

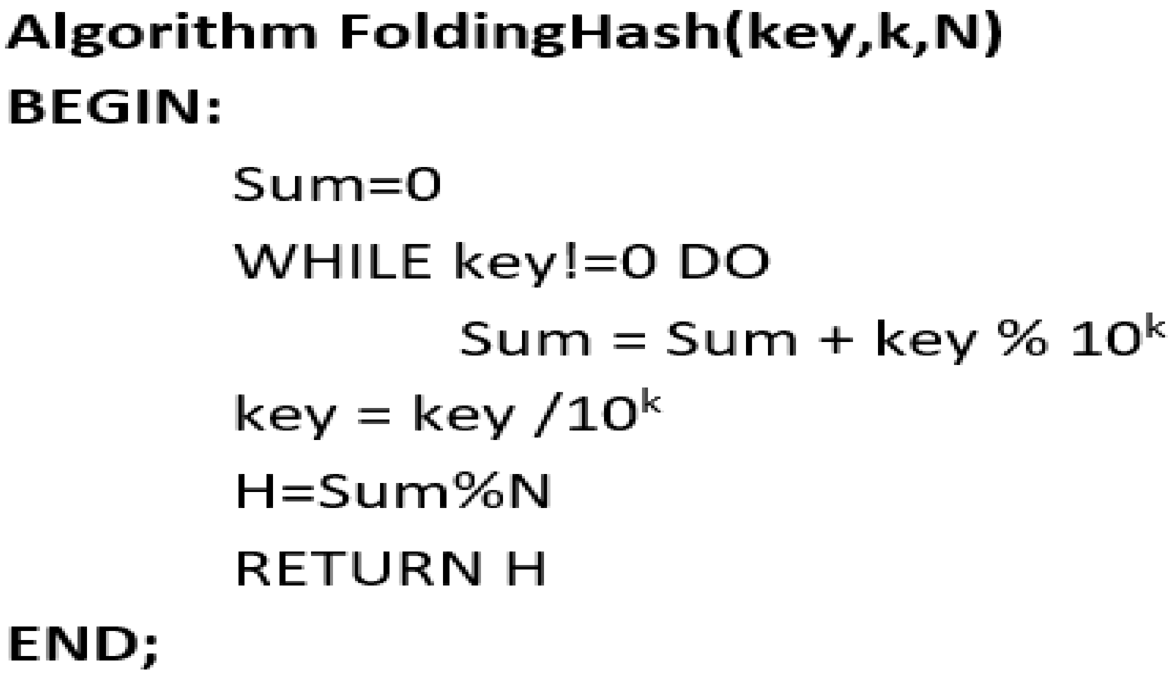
Hash code = 172 % 100 = 72

Hashing Functions

If the Table size (N) is 13, i.e. address will range from 0 – 12. Largest address is of 2 digits, the key will be divided into groups of 2 digits each. If the key is 12345678,

12+34+56+78 = 180

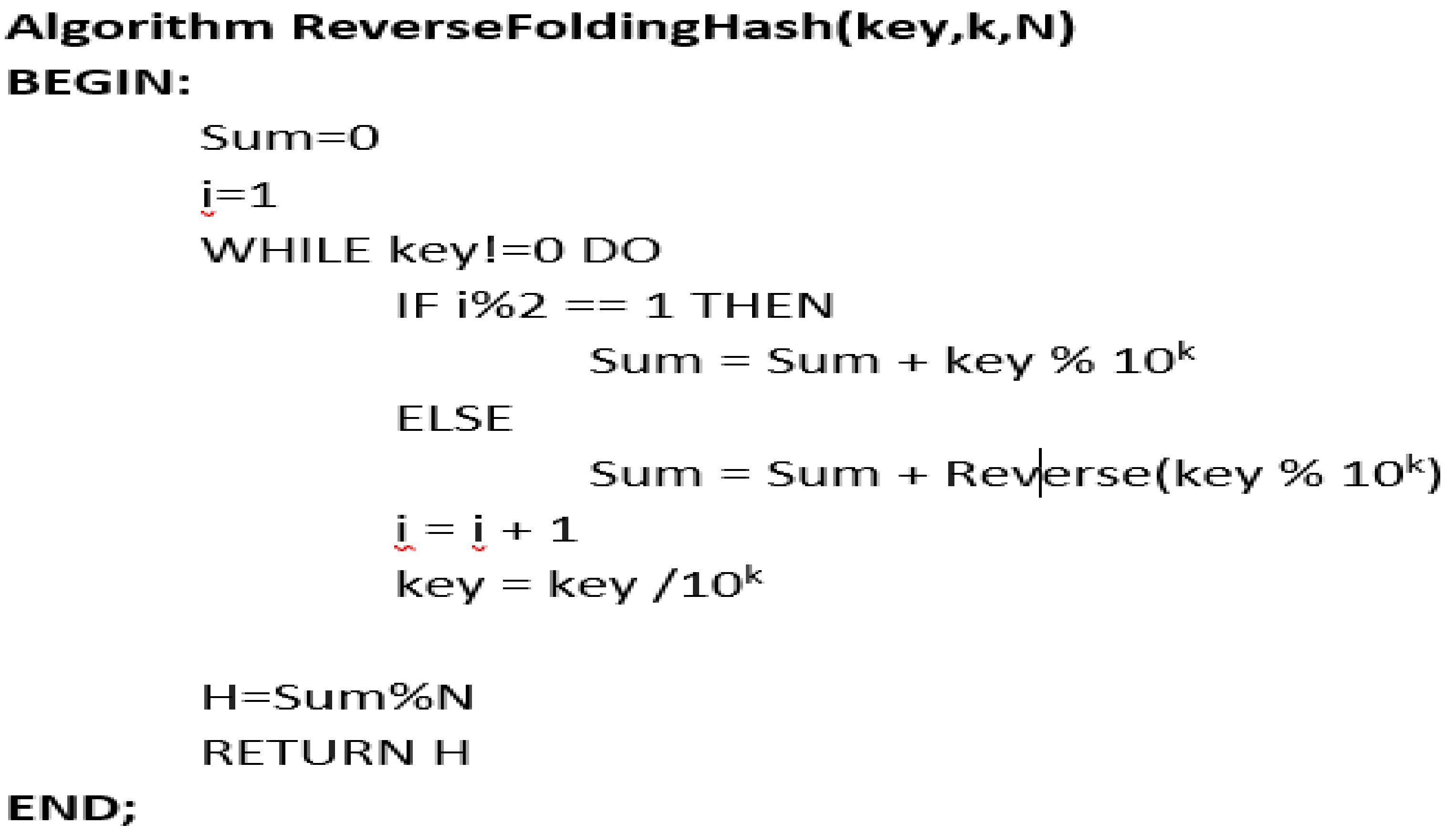
Hash Code = 180 % 13 =11



```
Algorithm FoldingHash(key,k,N)
BEGIN:
        Sum=0
        WHILE key!=0 DO
                Sum = Sum + key % $10^k$
        key = key /$10^k$
        H=Sum%N
        RETURN H
END;
```

Hashing Functions

A variation of the folding method is Reverse Folding, in which either the odd group or the even group is reversed before addition.



```
Algorithm ReverseFoldingHash(key,k,N)
BEGIN:
        Sum=0
        i=1
        WHILE key!=0 DO
                IF i%2 == 1 THEN
                        Sum = Sum + key % $10^k$
                ELSE
                        Sum = Sum + Reverse(key % $10^k$)
                i = i + 1
                key = key /$10^k$

        H=Sum%N
        RETURN H
END;
```

Collision Resolution in Hashing

The situation when the location found for two keys are same, the situation can be termed as collision.

$$H:Key_1 \rightarrow L$$

$$H:Key_2 \rightarrow L$$

As two data values cannot be kept in the exact location, collision is not desirable situation.

Avoiding collisions completely is difficult, even with a good hash function. Some method should be used to resolve this.

Collision Resolution in Hashing

There are two known methods of collision resolution in Hashing.

- Open Addressing
- Separate Chaining

Collision Resolution in Hashing

Open Addressing

Every key considers the entire table as the storage space. Thus, if it does not find the appropriate place for storage through the hash function, it tries to find the next free available slot.

- There are 3 different Open addressing mechanisms named as
    - **Linear Probing**
    - **Quadratic Probing**
    - **Rehashing/Double hashing**

Collision Resolution in Hashing

Linear Probing

If the key cannot be stored/searched at the given hash location, try to find the next available free slot by traversing sequentially

If the table size is TS,

$$H(K,j) = (H(K) + j) \text{ modulus } TS$$

$$j=0, 1, 2, \ldots$$

Sequence of investigation:

- H(K)
- (H(K) + 1) modulus TS
- (H(K) + 2) modulus TS,
- …

Modulus is applied because the Hash table is considered to be circular in nature.

Collision Resolution in Hashing

Disadvantages of Linear Probing

One main disadvantages of linear probing is that records tends to cluster that is appear next to one another, when the load factor is greater than 50%. In this case the average search time for a record is increasing.

Collision Resolution in Hashing

Quadratic Probing

Idea: when there is a collision, check the next available position in the table using the quadratic formula:

H(K,j) = (H(K) + a*j + b*j^2) modulus TS

j =0, 1, 2, ...

if a=1/2, b=1/2

Sequence of investigation:

- H(K)
- (H(K) + ½ + ½) modulus TS
- (H(K)+ ½*2+ ½ *2^2)modulus TS
- (H(K)+ ½*3+ ½ *3^2) modulus TS and so on

i.e. H(K), H(K)+1, H(K)+3, H(K)+6, ...

if a=0, b=1

Sequence of investigation:

- **H(K)**
- **(H(K) + 1) modulus TS**
- **(H(K)+ 1*2^2)modulus TS**
- **(H(K)+ 1 *3^2) modulus TS and so on**

i.e. H(K), H(K)+1, H(K)+4, H(K)+9, ...

Collision Resolution in Hashing

Quadratic Probing

In Quadratic probing insertion may be impossible is the table is more than half full;

Linear probing guarantee that all empty positions will be examined if necessary, quadratic probing doesnot.

Collision Resolution in Hashing

Rehashing/Double Hashing

The first hash function is used to find the hash table location and the second hash function to find the increment sequence

$$H(K,j) = (H(K) + j\ H'(K)\ )\ \text{modulus TS}, \quad j=0,1,\ldots$$

Sequence of investigation:

- H(K)
- (H(K) + H'(K)) modulus TS
- (H(K) + 2*H'(K)) modulus TS and so on
- (H(K) + 3*H'(K)) modulus TS and so on

Collision Resolution in Hashing

Example

- H(K) = k modulus 13
- H’(K) = 1+ (k modulus 11)
- H(K,i) = (H(K) + i H’(K)) mod 13

- **Insert key 14 in the Given Table**

H(14,0) = 14 modulus 13 = 1

H’(K) = 1+ (k modulus 11)

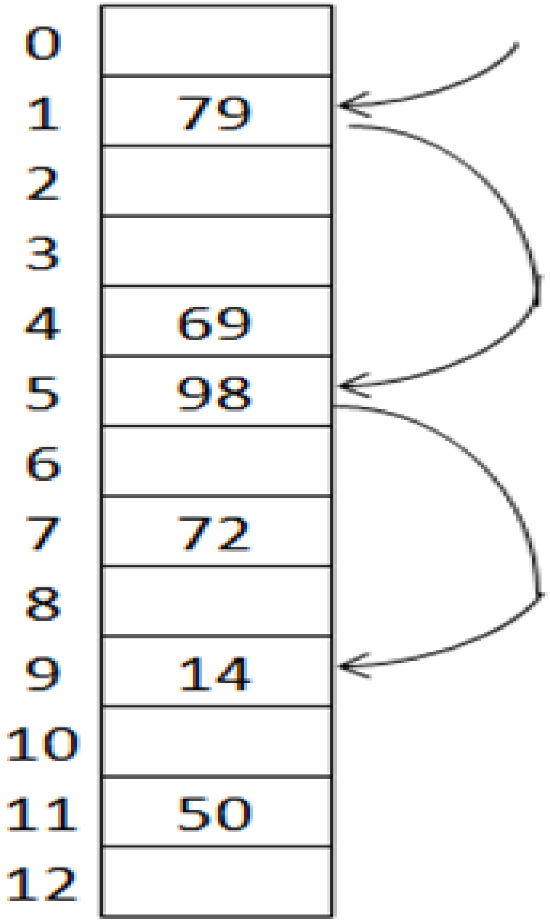
H’(14) = 1+ (14 modulus 11)= 1+3=4

H(14,1) = (H(14) + H’(14)) modulus 13

= (1 + 4) modulus 13 = 5

H(14,2) = (H(14) + 2 H’(14)) modulus 13

= (1 + 8) modulus 13 = 9

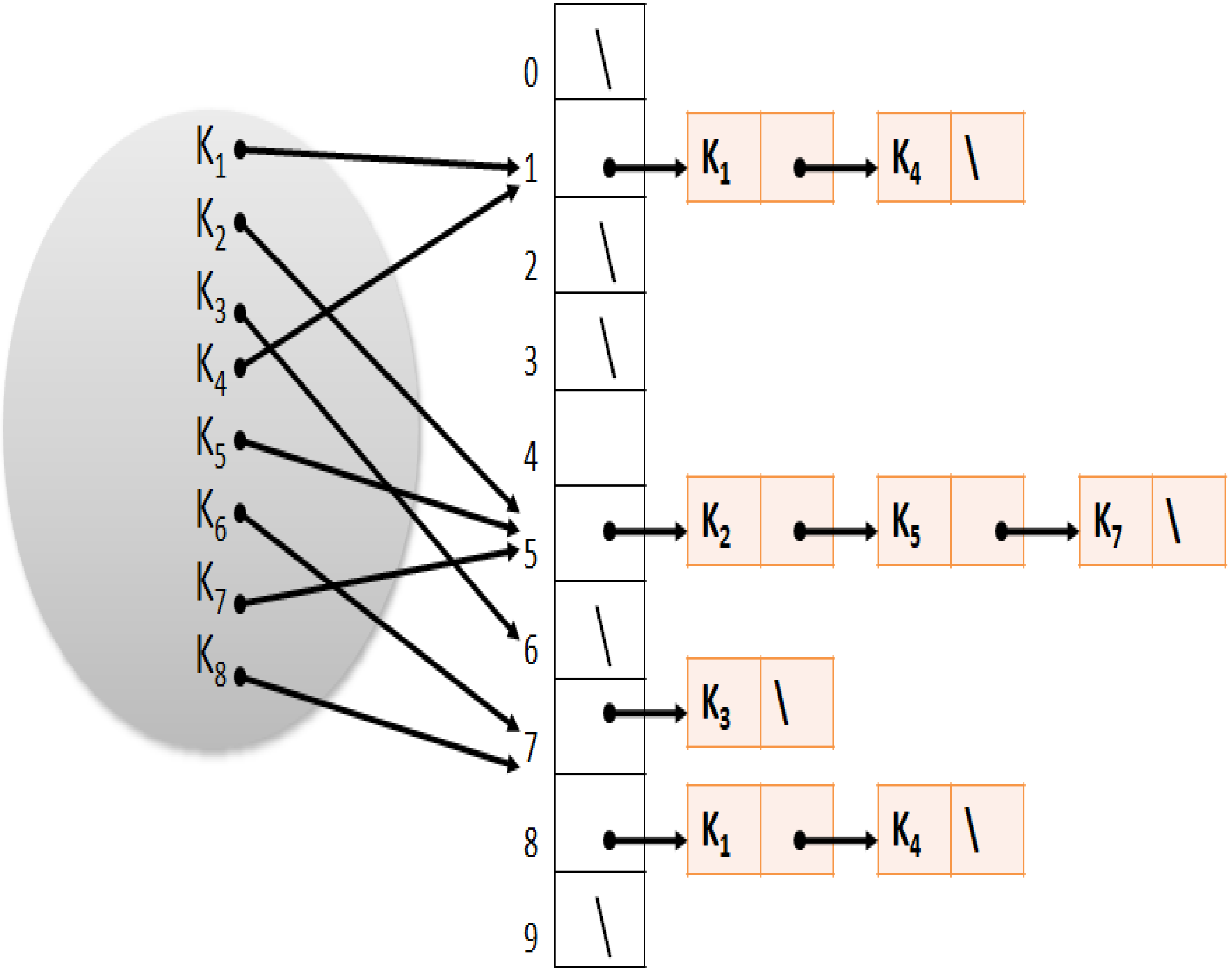


0	
1	79
2	
3	
4	69
5	98
6	
7	72
8	
9	14
10	
11	50
12	

Collision Resolution in Hashing

Chaining

- The chaining method takes the array of linked lists in contrast to the linear array for Hash Table.
- The keys with the same hash address go to the same linked list.



Load Factor of a Hash Table

- **S(λ): Average number of probes for successful search**

- Load factor is defined as

  S(λ) = N/TS

  S(λ): Average number of probes for successful search

  - N refers to the Total number of keys stored in the hash table
  - TS refers to the Hash Table Size

Load Factor of a Hash Table

- **U(λ): Average number of probes for unsuccessful search**

- Load factor is defined as

$$U(\lambda) = N/TS$$

U(λ): Average number of probes for unsuccessful search

  - N refers to the Total number of keys stored in the hash table
  - TS refers to the Hash Table Size

Thank You