Unit-7
Searching and Hashing

What is Searching?

- We know that today's computer stores a lot of information.
- To retrieve this information effectively we need very efficient searching algorithms.
- Searching is an operation or a technique that helps to find the location of a given elements or value in the list or array.

Or

- Searching refers to the operation of finding the location of given data item in a collection of items. It is the retrieval process from the sorted data

- Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not.

In data structures,

- There are several searching techniques like **Linear Search, Binary Search, Tree Search** etc.

- In these techniques, time taken to search any particular element depends on the total number of elements.

Types of Searching Algorithm:

1. Sequential Search Algorithm

2. Binary Search Algorithm

3. Tree Search Algorithm

Drawback:

The main drawback of these techniques is:

- As the number of elements increases, time taken to perform the search also increases.

- This becomes problematic when total number of elements become too large.

Evaluation of Search Strategies

- **Completeness** - does it guarantee to find a solution when there is one?
- **Time complexity** - how long does it take to find a solution?
- **Space complexity** - how much memory does it require?
- **Optimality** - does it return the best solution when there are many?

1. Sequential search algorithm

It is also called linear search algorithm. It can be applied to array as well as linked list. This method is used for both sorted and unsorted data items.

Algorithm:

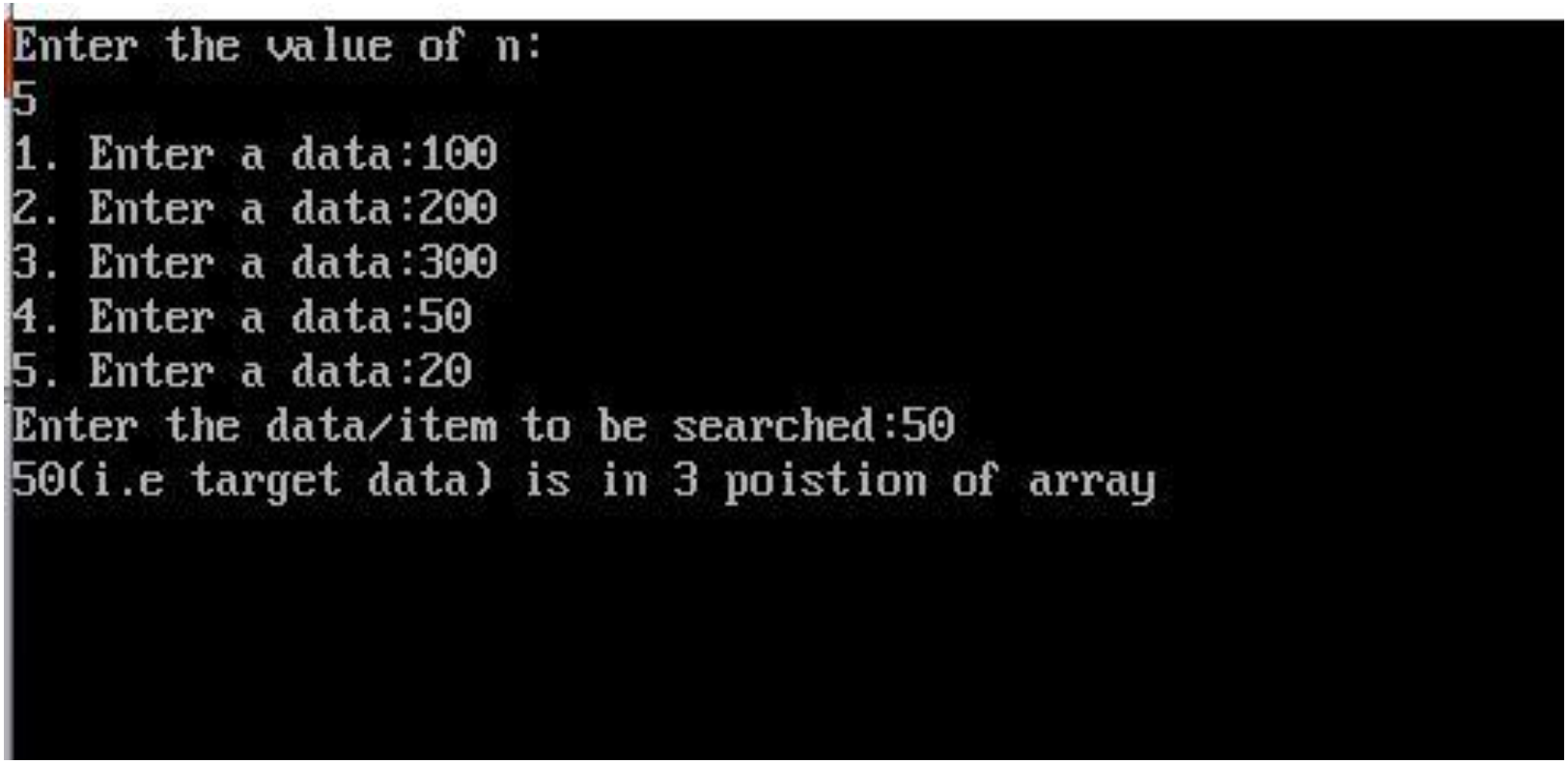
1. Read the target value to find in an array.
2. Set key=0
3. While (key<n) n is number of elements of array 'a'.

```
{
        if (a[key]==target)
                Print the index of array i.e. key
                Stop
        else
                key++
}
```

4. If data is not found, display the message.

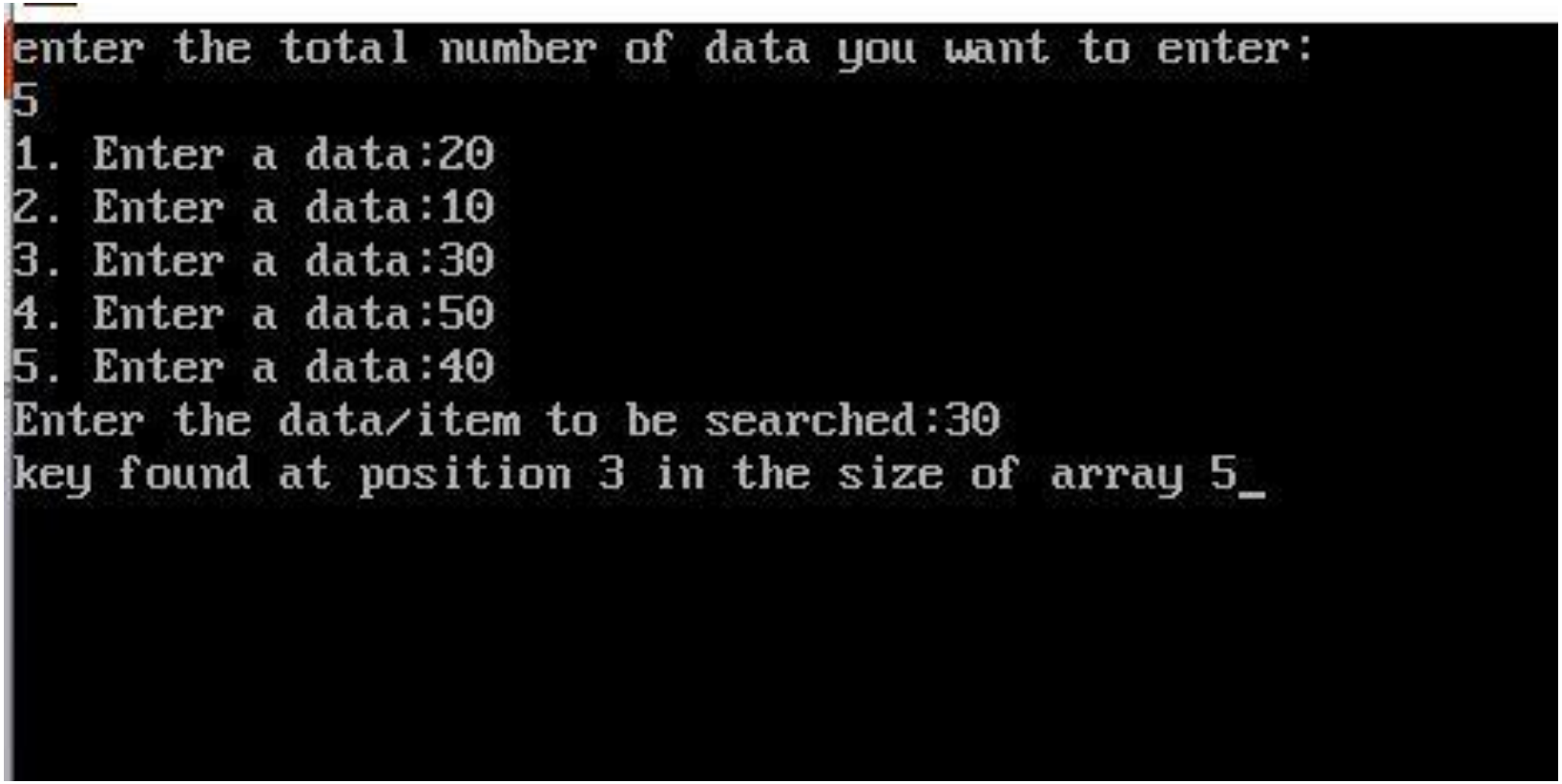
//Method1: sequential search program using while loop with function. (seq1.cpp)

```
#include<stdio.h>
#include<conio.h>
int seq_search(int target,int a[], int n);
void main()
{
   int i, n, target, pos, a[20];
   clrscr();
   printf("Enter the value of n:\n");
   scanf("%d", &n);
   for (i = 0; i < n; i++)
   {
            printf("%d. Enter a data:",i+1);
            scanf("%d", &a[i]);
   }
   printf("Enter the data/item to be searched:");
   scanf("%d", &target);

   pos = seq_search(target, a, n);

   if (pos<0)
            printf("Search unscccessful");
   else
            printf("%d(i.e target data) is in %d
poistion of array" ,target, pos);
            getch();
}

int seq_search(int target,int a[], int n)
{
   int key=0;
   while(key<n)
   {
            if(a[key] == target)
                     return key;
            else
                     key++;

   }

}
```


Enter the value of n:
5
1. Enter a data:100
2. Enter a data:200
3. Enter a data:300
4. Enter a data:50
5. Enter a data:20
Enter the data/item to be searched:50
50(i.e target data) is in 3 poistion of array

//Method2: sequential search program using for loop with function. (seq.cpp)

```
#include <stdio.h>
#include <conio.h>
int seq_search(int key, int a[], int n)
{
   int i;
   for (i = 0; i < n; i++)
   {
            if(a[i] == key)
            return i+1;
   }
   return 0;
}

void main()
{
   int i, n, key, pos, a[20];
   clrscr();
   printf("enter the total number of data you want to enter:\n");
   scanf("%d", & n);
   for (i = 0; i < n; i++)
   {
            printf("%d. Enter a data:",i+1);
            scanf("%d", &a[i]);
   }
   printf("Enter the data/item to be searched:");
   scanf("%d", &key);

   pos = seq_search(key, a, n);
   if (pos == 0)
            printf("Search unscccessful");
   else
            printf("key found at position %d in the size of array %d", pos,n);
            getch();
}
```


enter the total number of data you want to enter:
5
1. Enter a data:20
2. Enter a data:10
3. Enter a data:30
4. Enter a data:50
5. Enter a data:40
Enter the data/item to be searched:30
key found at position 3 in the size of array 5_

```
//METHOD 3:  using only for loop no use of function. (Seq2.cpp)
void main()
{
   int i, n, target, a[20], found=0;
   printf("Enter how many data you want to enter:\n");
   scanf("%d", &n);
   printf("enter %d data:\n",n);
   for (i = 0; i < n; i++)
   {
                scanf("%d", &a[i]);
   }
   printf("Enter the data/element to be searched:");
   scanf("%d", &target);
   for(i=0; i<n; i++)
   {
                if(a[i] == target)
                {
                                printf("element found in index %d",i);
                                found=1;
                                break;
                }
   }
   if(found==0)
                printf("data not found in an array.");
   getch();
}
```

```
Enter how many data you want to enter:
5
enter 5 data:
11
22
33
44
55
Enter the data/element to be searched:33
element found in index 2
```

```
Enter how many data you want to enter:
5
enter 5 data:
11
22
33
44
55
Enter the data/element to be searched:100
data not found in an array.
```

```
//METHOD 4 : using only for loop no use of function. (Seq3.cpp)
void main()
{
   int i, n, target, a[20];
  printf("Enter how many data you want to enter:\n");
   scanf("%d", &n);
   printf("enter %d data:\n",n);
   for (i = 0; i < n; i++)
   {
            scanf("%d", &a[i]);
   }
   printf("Enter the data/element to be searched:");
   scanf("%d", &target);
   for(i=0; i<n; i++)
   {
            if(a[i] == target)
            {
                     printf("element found in index %d",i);
                     break;
            }
   }
   if(i==n)
            printf("data not found in an array.");
   getch();
}
```

```
Enter how many data you want to enter:
5
enter 5 data:
11
22
33
44
55
Enter the data/element to be searched:44
element found in index 3_
```

```
Enter how many data you want to enter:
5
enter 5 data:
11
22
33
44
55
Enter the data/element to be searched:100
data not found in an array.
```

```
//METHOD 5: using only for loop no use of function. (Seq4.cpp)
void main()
{
   int a[10],i,size,item,pos,found=0;
   printf("Enter the size of an array:\n");
   scanf("%d",&size);
   printf("Enter the elements of the array:\n");
   for(i=0;i<size;i++)
   {
            scanf("%d",&a[i]);
   }
   printf("Enter the element to be searched:");
   scanf("%d",&item);
   for(i=0;i<size;i++)
   {
            if(item==a[i])
            {
               pos=i;
               flound=1;
               break;
            }
   }
   if(found==1)
            printf("The position of %d in array is: %d",item, pos+1);
   else
            printf("The element is not found");
   getch();
}
```

```
Enter the size of an array:
5
Enter the elements of the array:
11
22
33
44
55
Enter the element to be searched:33
The position of 33 in array is: 3
```

```
Enter the size of an array:
5
Enter the elements of the array:
11
22
33
44
55
Enter the element to be searched:100
The element is not found_
```

```
//METHOD 6: using only while loop no use of function. (Seq5.cpp)
void main()
{
   int i, n, target, pos, key=0, a[20];
   printf("Enter the value of n:\n");
   scanf("%d", &n);
   for (i = 0; i < n; i++)
   {
            printf("%d. Enter a data:",i+1);
            scanf("%d", &a[i]);
   }
   printf("Enter the data/item to be searched:");
   scanf("%d", &target);
   while(key<n)
   {
            if(a[key] == target)
            {
                     pos=key;
                     printf("%d is in %d position of array.",target, pos+1);
                     break;
            }
            else
                     key++;
   }
   if(key==n)
            printf("%d is not in array",target);
   getch();
}
```

```
Enter the value of n:
5
1. Enter a data:11
2. Enter a data:22
3. Enter a data:33
4. Enter a data:44
5. Enter a data:55
Enter the data/item to be searched:22
22 is in 2 position of array.
```

```
Enter the value of n:
5
1. Enter a data:11
2. Enter a data:22
3. Enter a data:33
4. Enter a data:44
5. Enter a data:55
Enter the data/item to be searched:66
66 is not in array_
```

Time complexity of linear search:

- If we have data in array as follows:

0	1	2	3	4	5	6	7
14	44	55	33	21	88	9	76

- If we want to find data 14 then in one comparison we find data. So **Best Case: O(1)**
- If we want to find data 76 then in N comparison needed. So **Worse Case: O(n)**
- **For Average Case:** $\frac{\sum \text{all cases}}{\text{No. of cases}} = \frac{1+2+3+\ldots\ldots+n}{n}$

$$= \frac{n(n+1)}{2 \cdot n} = \frac{(n+1)}{2}$$

So Average Case: O($\frac{n+1}{2}$)

2. Binary search algorithm:

- The sequential search situation will be in worse case i.e. if the element is at the end of the list. For eliminating this problem one efficient searching technique called binary search is used in this case.

- Binary search algorithm is types of searching algorithm which is very fast and efficient. To search using this method, the list of elements must be in **sorted order.**

- In binary search, to search an element the comparison is made with element present in middle. If equal then the search is said to be successful otherwise the array is divided into two parts, one starting from 0 to (n/2-1) and the second portion from (n/2+1) to (n-1).

- As a result, the first portion contains elements less than mid element and second portion contains element greater than mid element. If element to be search is smaller than mid element then searching will be continue in first portion, otherwise second portion, so, this process is repeated till element is found or division cannot be done further.

Algorithm:

1.Set top=no. of data-1 and bottom=0

2. While(bottom<=top)

{

Mid = (top+bottom)/2

if (key == a[mid]

return mid

if (key < a[mid])

Top = mid-1

else

bottom = mid+1

}

3. If data is not found then display data not found message.

```
//binary search using function. (Binary1.cpp)
//method1:
#include<stdio.h>
#include<conio.h>
int binary_search(int n, int a[], int key);
void main()
{
 int i, pos, n, key, a[100];
 clrscr();
 printf("Enter number of elements:\n");
 scanf("%d", &n);
 printf("Enter %d integers:\n", n);
 for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
 printf("Enter value to find:");
 scanf("%d", &key);
 pos=binary_search(n, a, key);
 if(pos>=0)
 printf("%d is location of data in array.",pos);
 else
 printf("%d in not present in list",key);
 getch();
 }
 int binary_search(int n, int a[], int key)
 {
        int bottom=0, mid;
        int top=n-1;
        while (bottom <= top)
        {
                mid=(top+bottom)/2;
                if(key==a[mid])
                        return mid;
                if (key< a[mid])
                        top=mid-1;
                else
                        bottom=mid+1;
        }
        return -1;
    }
```

```
Enter number of elements:
10
Enter 10 integers:
11
22
33
44
55
66
77
88
99
100
Enter value to find:77
6 is location of data in array._
```

Output 1

```
Enter number of elements:
5
Enter 5 integers:
11
22
33
44
55
Enter value to find:100
100 in not present in list_
```

Output 2

//binary search program without using function. (Binary.cpp) [method2:]

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int i, bottom=0, n, top=n-1, mid, flag, key, a[100];
  clrscr();
  printf("Enter number of elements:\n");
  scanf("%d", &n);
  printf("Enter %d integers in ascending order:\n", n);
  for (i = 0; i < n; i++)
              scanf("%d", &a[i]);
  printf("Enter the element to be searched:");
  scanf("%d", &key);
  bottom=0;
  top=n-1;
  while (bottom <=top)
  {
    mid=(top+bottom)/2;
    if(key==a[mid])
    {
                flag=1;
                break;
    }
    else
    {
     if (key< a[mid])
                top=mid-1;
     else
                bottom=mid+1;
    }
  }
  if(flag==1)
                printf("data found at location: %d",mid+1);
  else
                printf("%d is not present in the list.", key);
    getch();
    return 0;
}
```

```
Enter number of elements:
5
Enter 5 integers in ascending order:
11
22
33
44
55
Enter the element to be searched:44
data found at location: 4
```

Output 1

```
Enter number of elements:
5
Enter 5 integers in ascending order:
11
22
33
44
55
Enter the element to be searched:66
66 is not present in the list.
```

Output 2

Time complexity of binary search:

Worst Case: O(log n)

Best Case: O(1)

Average Case: O(log n)

3. Tree search algorithm

The tree search algorithm is accomplished by two ways:

1. General search tree algorithm:

- For general search tree algorithm, every node is traversed until the desired data item is found. The traversed may be in-order, preorder or post-order.

2. Binary search tree algorithm:

- In binary search tree algorithm, first comparison is made with root node and accordingly either left sub-tree or right sub-tree is traversed for successful search.

Algorithm for binary search tree:

1. Read the target value.
2. Compare the target value with the root node.
3. If equal then, display the data found message.

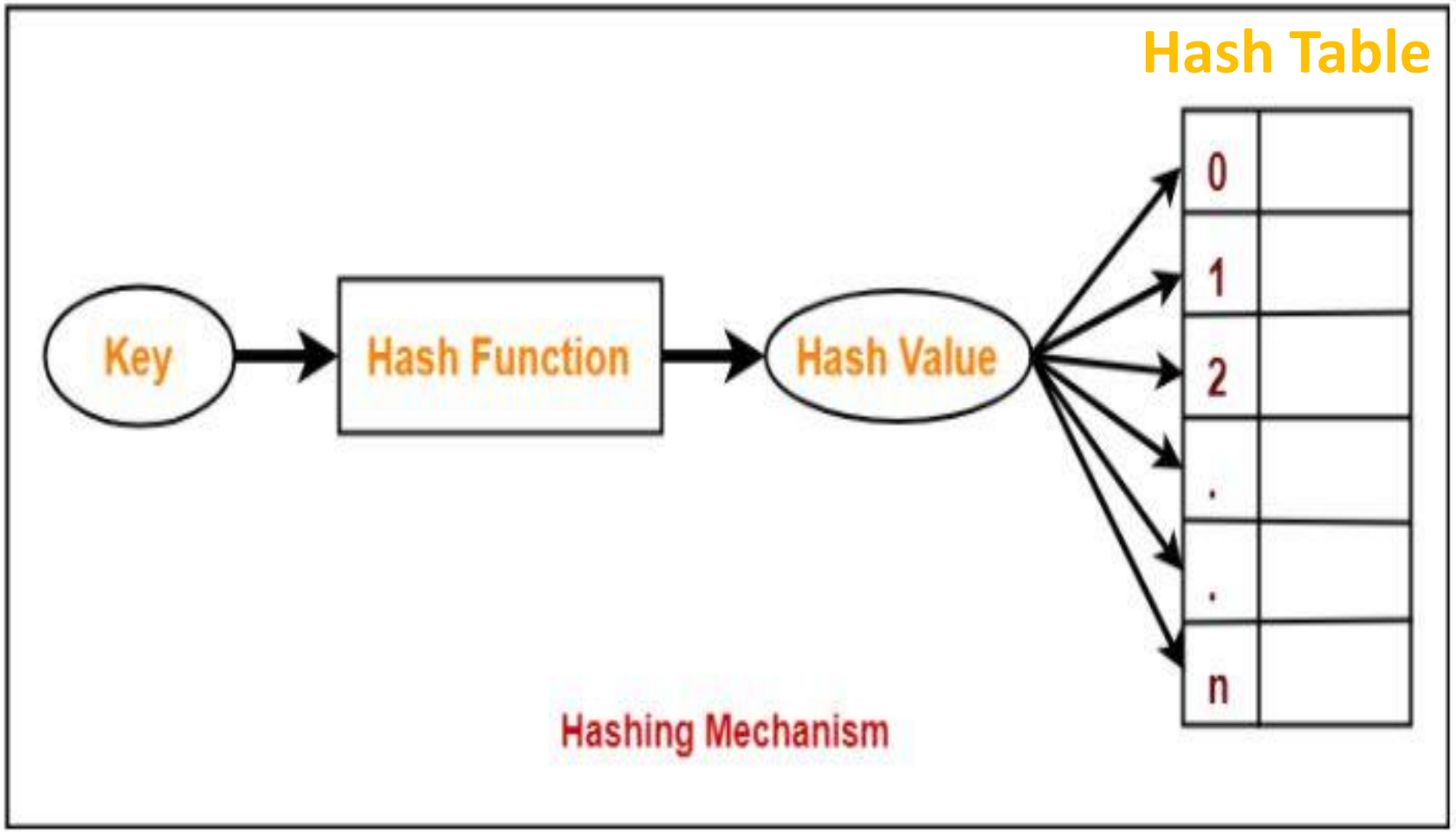
   else

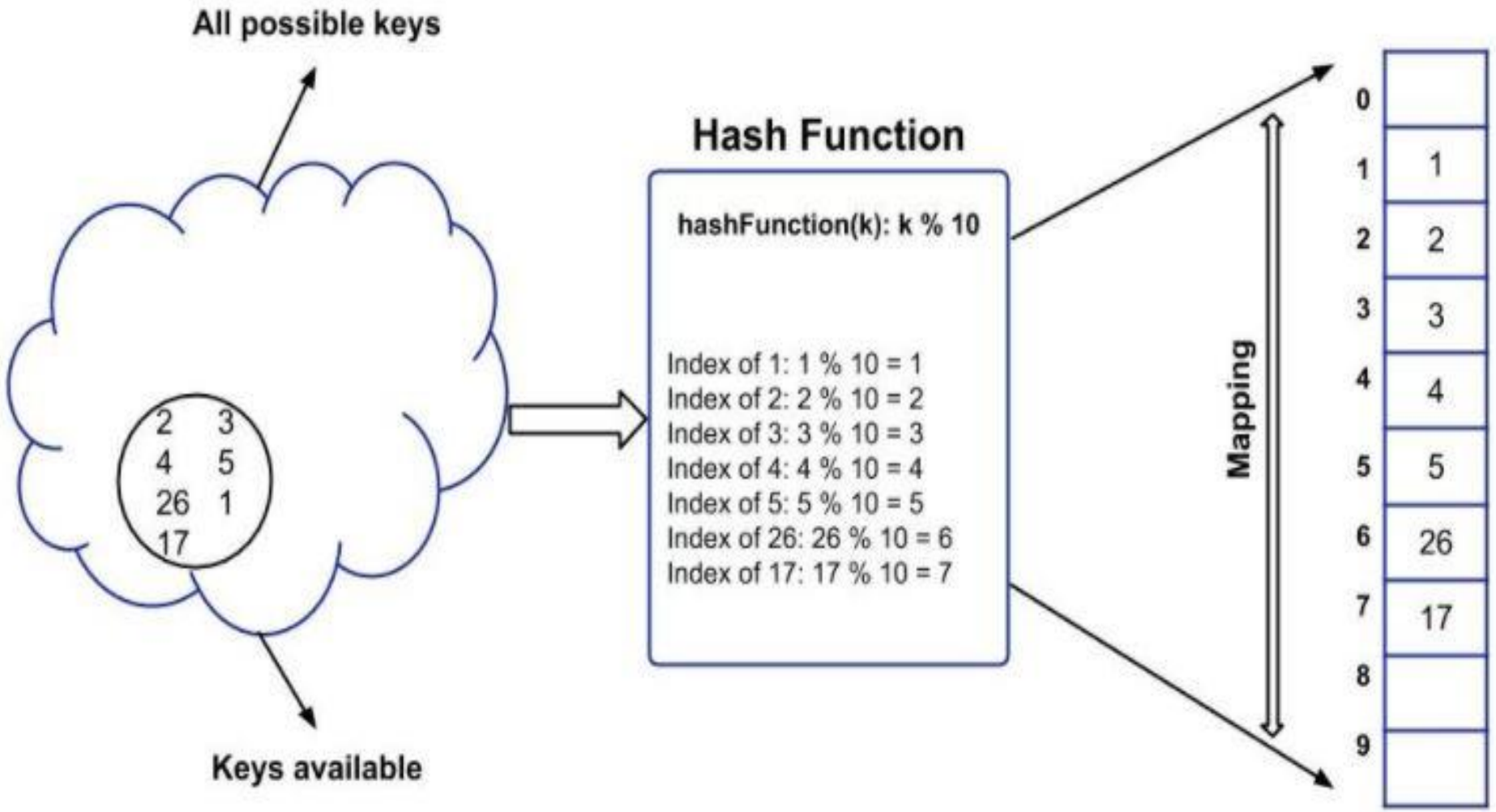
   If the target is less than root node, traversed to the left sub-tree else traversed right sub-tree.

5. Repeat step 2 and 3 until data is found or traverse is over.
6. stop

Hashing:

- **Hashing** is a technique or process of mapping keys or values into the **hash table** by using a **hash function.**
- It is done for faster access to elements.
- In data structures, Hashing is a well-known technique to search any particular element among several elements.
- It minimizes the number of comparison while performing the search.


Hash Table
Key
Hash Function
Hash Value
0
1
2
.
.
n
Hashing Mechanism



Hashing Mechanism

Hash table:

- The table used for storage a key is called **hash table.**

Or

- An array data structure called as **Hash table** is used to store the data items.

Hash Key Value:

- Based on the hash key value, data items are inserted into the **hash table.**
- Hash key value is a special value that serves as an index for a data item.
- It indicates where the data item should be stored in the hash table.
- Hash key value is generated using a **hash function.**

Hash function:

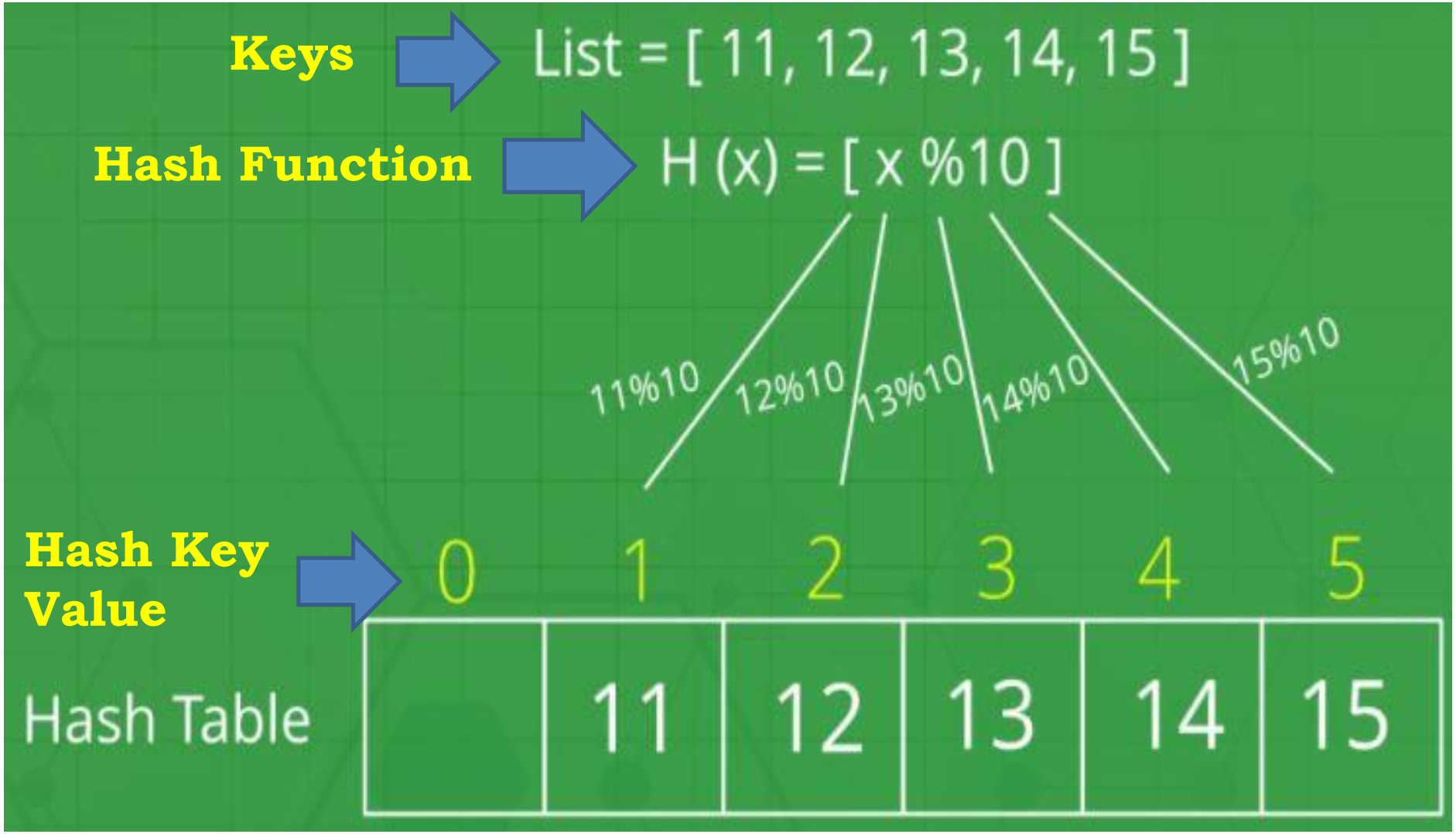
- A hash function 'H' is simply a mathematical formula that manipulates the key in some form to compute the index of keys in hash table.

Or

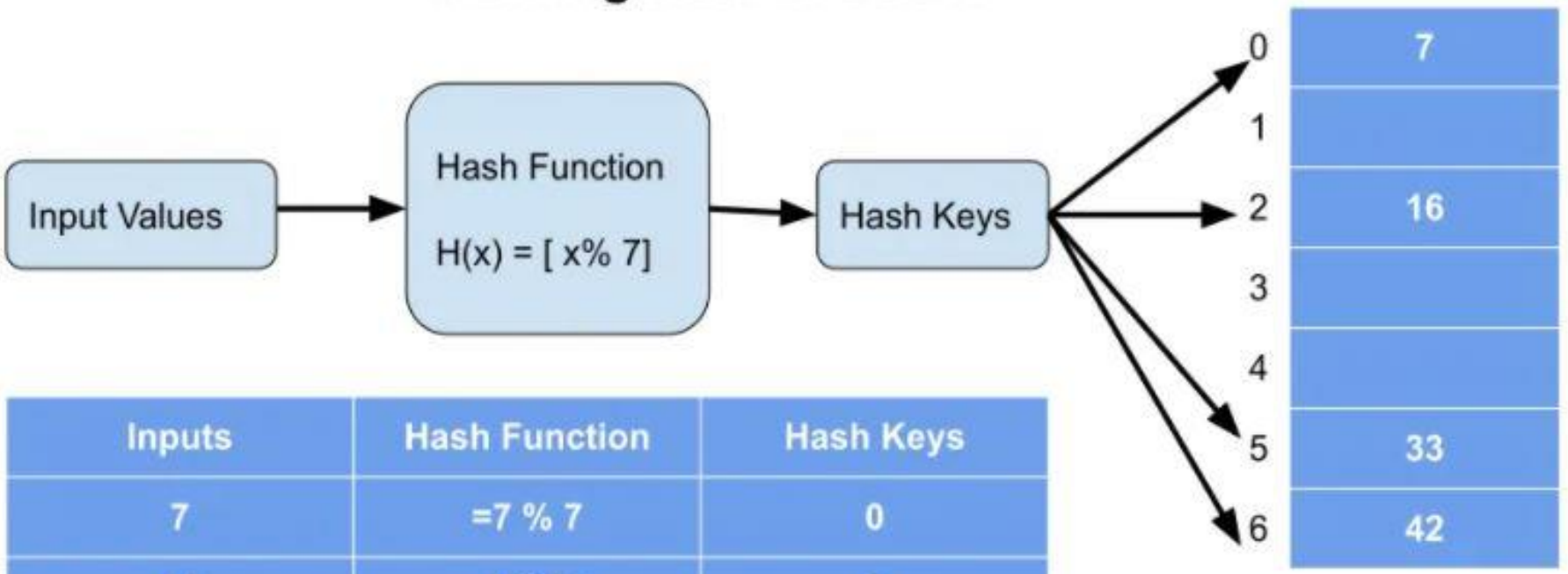
- **Hash function is a mathematical function which is used to compute the hash value for a key.**

- So, the function that transforms a key into table index is called **hash function.**

- Let a hash function H(x) maps the value at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.


Keys
List = [11, 12, 13, 14, 15]
Hash Function
H (x) = [x %10]
11%10
12%10
13%10
14%10
15%10
Hash Key Value
0
1
2
3
4
5
Hash Table
11
12
13
14
15

Hashing Data structure



Inputs	Hash Function	Hash Keys
7	=7 % 7	0
16	=16 % 7	2
33	=33 % 7	5
42	=42 % 7	6

Types of Hash Functions-

There are various types of hash functions available such as-

1. Mid Square Hash Function
2. Division Hash Function
3. Folding Hash Function etc

It depends on the user which hash function he wants to use.

Properties of Hash Function-

The properties of a good hash function are-

- It is efficiently computable.
- It minimizes the number of collisions.
- It distributes the keys uniformly over the table.

Advantages of Hashing:

Unlike other searching techniques,

- Hashing is extremely efficient.

- The time taken by it to perform the search does not depend upon the total number of elements.

- It completes the search with constant time complexity O(1).

Collision in Hashing/ Hash Collision:

- If hash function produces same index or slots for different data than **hash collision** occurs.

- So, a **Collision** is a phenomenon that occurs when more than one key maps to the same slots in the hash table.

Or

- When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a Collision.

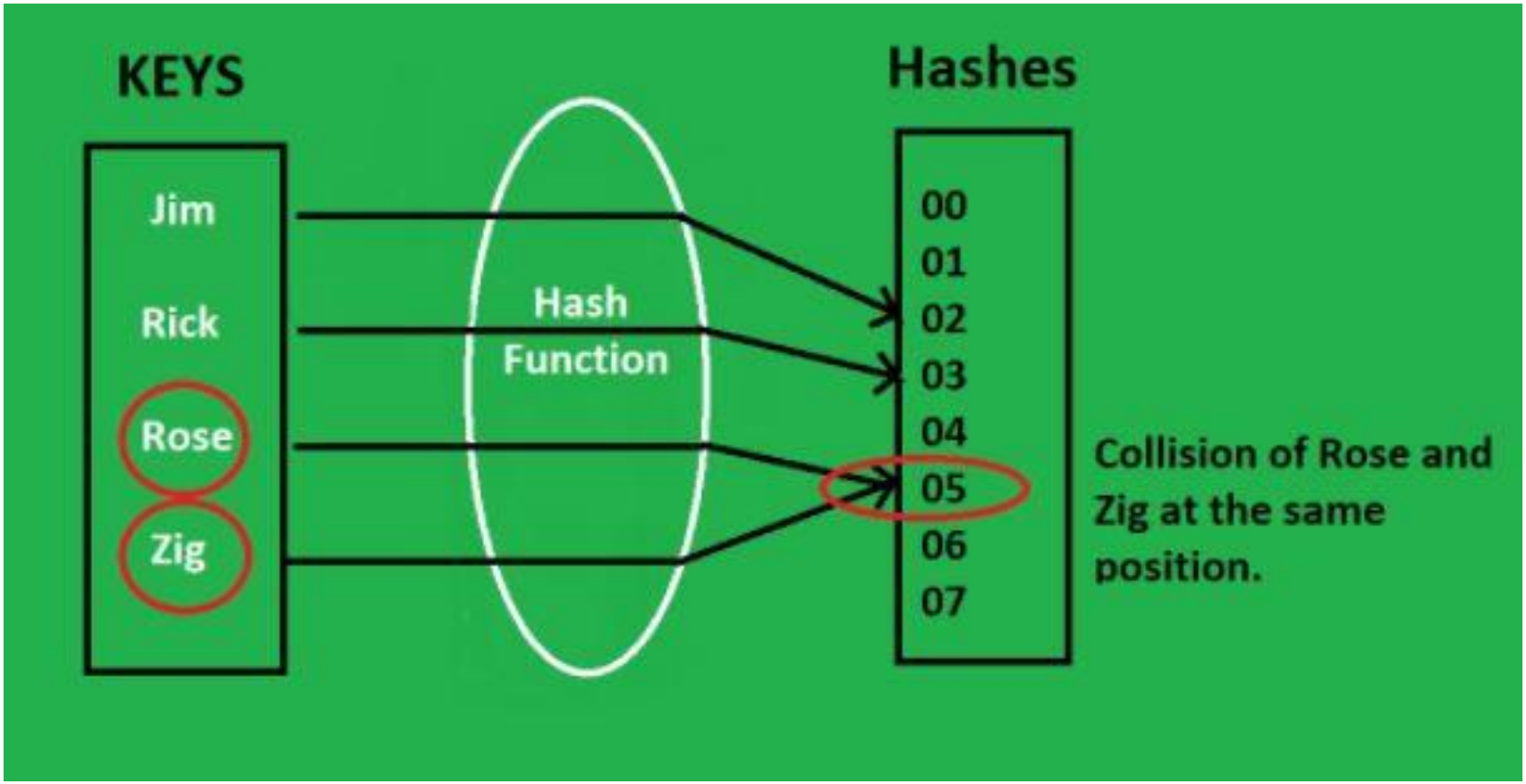


Fig: Collision in Hashing

Note: https://www.gatevidyalay.com/collision-resolution-techniques-separate-chaining/

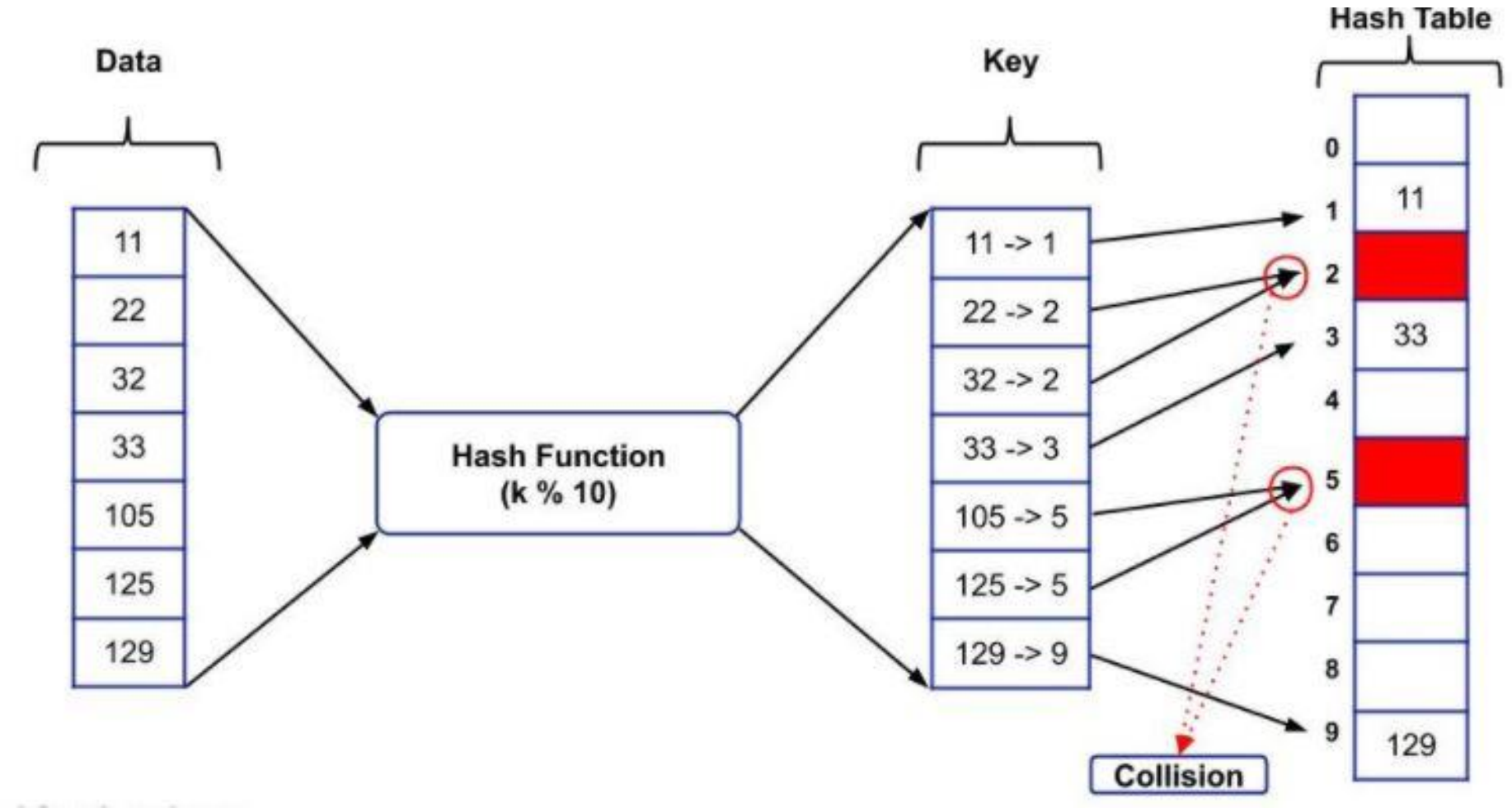
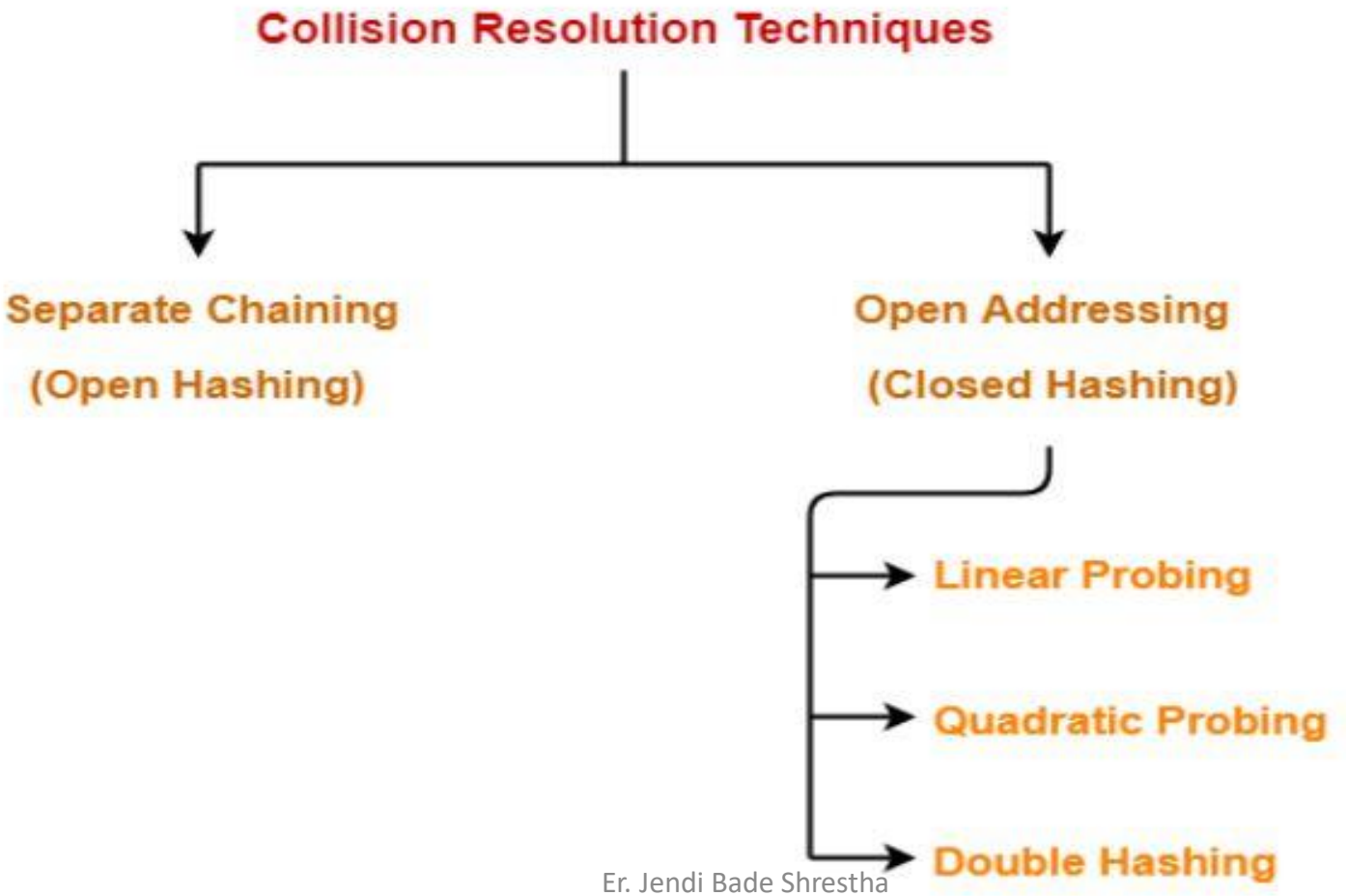


Fig: Collision in Hashing

Collision Resolution Techniques:

- Collision Resolution Techniques are the techniques used for resolving or handling the collision.
- Collision resolution techniques are classified as:



Separate Chaining:

- Chaining is a collision resolution technique where we make use of the **linked list** or list data structure to store the keys that are having the same value.

Question 1:

If the data/key is 3, 23, 24, 26, 36, 78, and 99, and the hash Function(k): k % 10 then store the key using separate chaining method.

Solution:

The hash function(k)=k%10

And the key is as follows:

3, 23, 24, 26, 36, 78, and 99

Now using hash function we get,

3%10=3

23%10=3

24%10=4

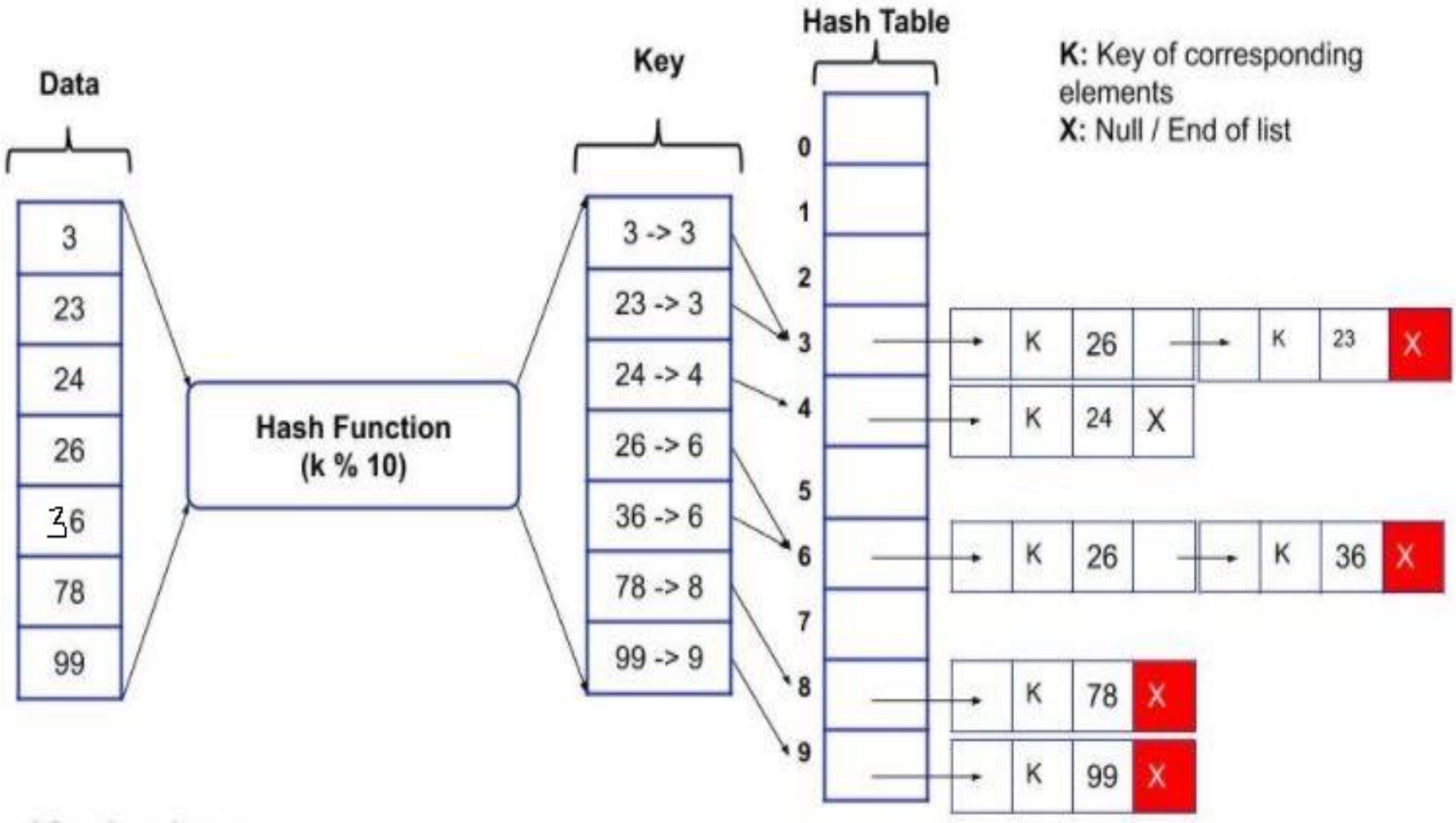
26%10=6

36%10=6

78%10=8

99%10=9

So, the storage of the data/key in the hash table can be represented as:



Question2:

Let us consider a simple hash function as "key mod 7" and sequence of keys as 50, 700, 76, 85, 92, 73, 101. now store the key using separate chaining method.

Solution:

- Given hash function is key mod 7.

- Sequence of keys is as follows:

  50, 700, 76, 85, 92, 73, 101.

Now using hash function we get,

50%7=1

700%7=0

76%7=6

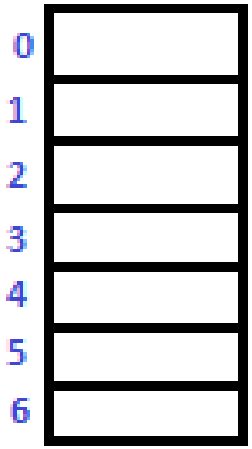
85%7=1

92%7=1

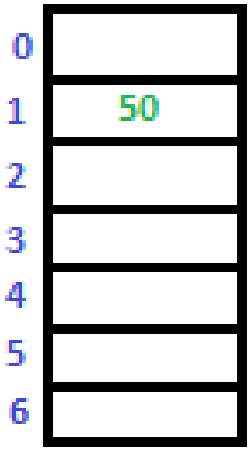
73%7=3

101%7=3

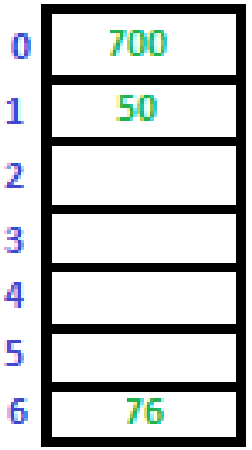
So, the storage of the data/key in the hash table can be represented as:



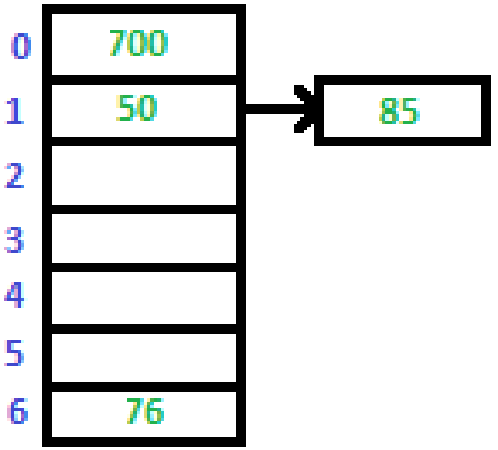
Initial Empty Table



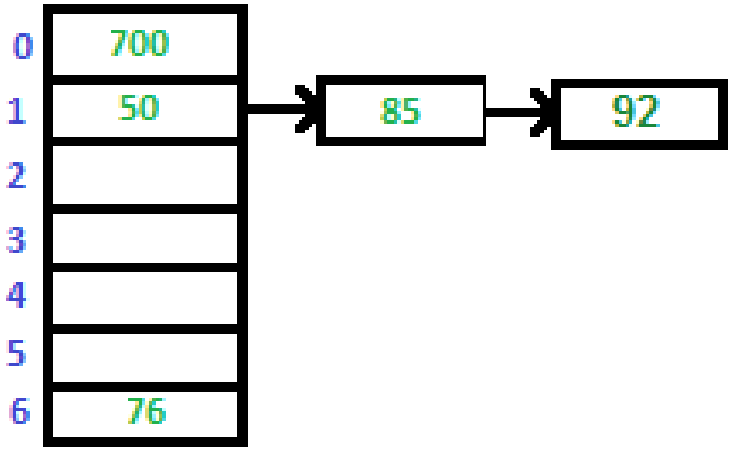
Insert 50



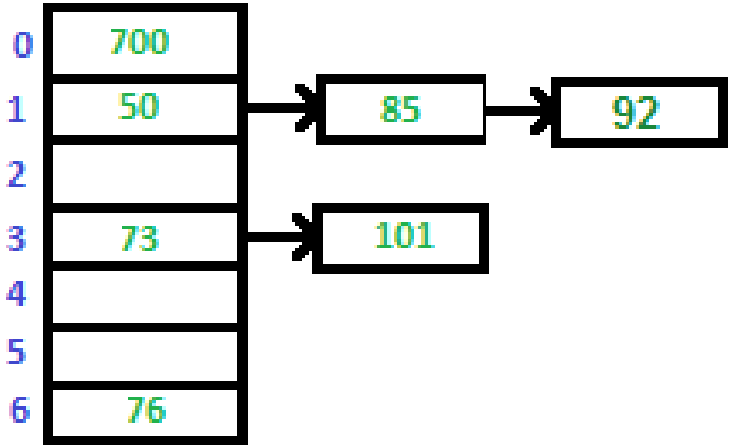
Insert 700 and 76



Insert 85: Collision
Occurs, add to chain



Inser 92 Collision
Occurs, add to chain



Insert 73 and 101

Advantages of Separate Chaining:

- Simple to implement.
- Hash table never fills up, we can always add more elements to the chain.
- It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Disadvantages of Separate Chaining:

- Wastage of Space (some index of hash table are never used)
- If the chain becomes long, then search time can become O(n) in the **worst case.**
- Uses extra space for links.

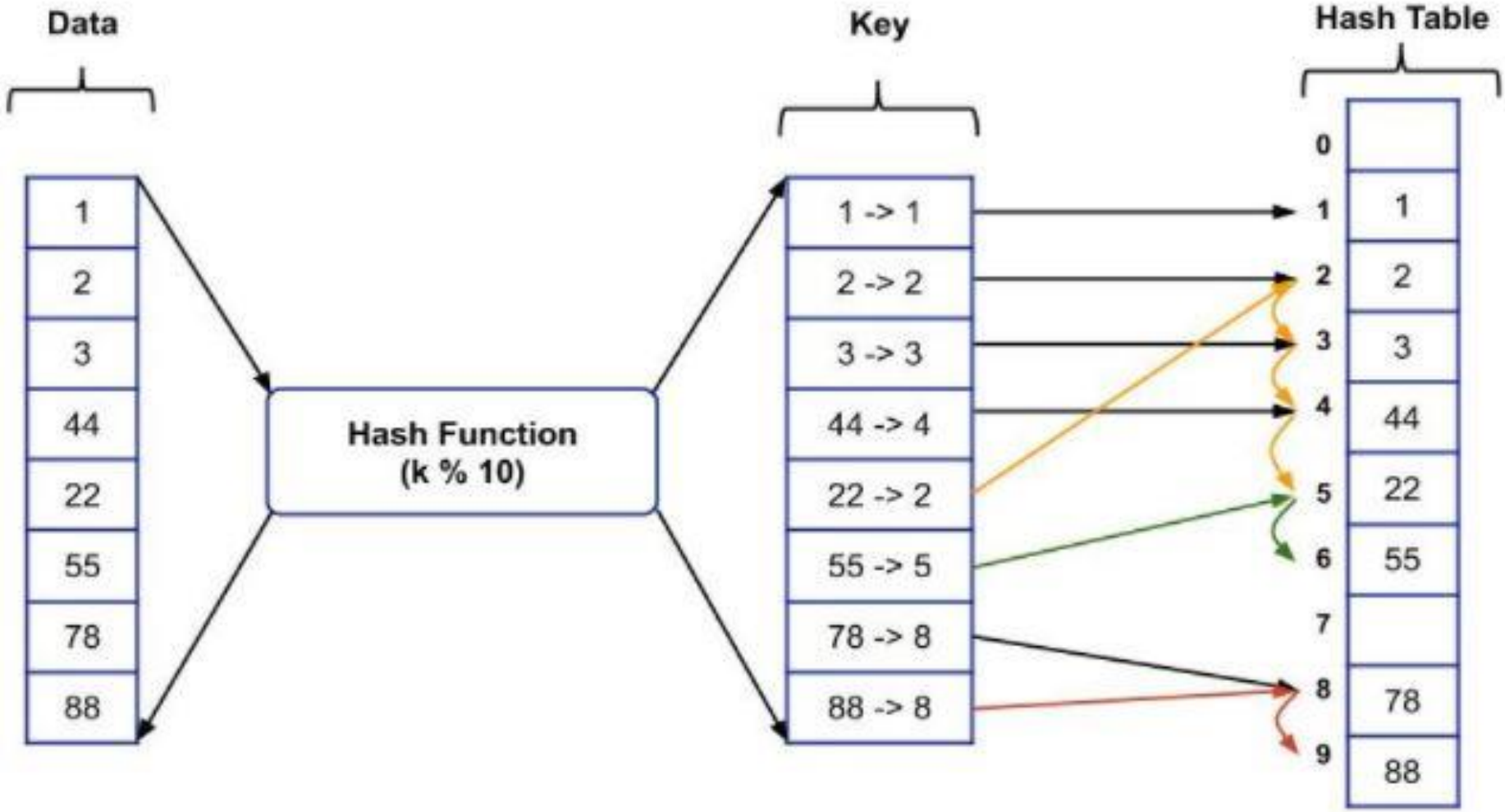
Note: https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/

Open Addressing

- Like separate chaining, open addressing is a method for handling collisions.

- In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys.

- Open Addressing is done in the following ways:
  - Linear Probing
  - Quadratic Probing
  - Double Hashing


Data
Key
Hash Table
1
2
3
44
22
55
78
88
Hash Function
(k % 10)
1 -> 1
2 -> 2
3 -> 3
44 -> 4
22 -> 2
55 -> 5
78 -> 8
88 -> 8
0
1
2
3
4
5
6
7
8
9
1
2
3
44
22
55
78
88

1. Linear Probing

- In linear probing,

a. When collision occurs, we linearly probe for the next bucket.

b. We keep probing until an empty bucket is found.

- The purpose of examining the slot in hash table is called probing or rehashing.
- Linear probing uses the following hash function

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

for i=0,1,2,3,4,.....,m-1
where, m is size of hash table
h'(k) = k mod m
i is prob no.

Q No.1 Insert keys i.e 76, 26, 37, 5, 9, 21, 65, 85 into hash table of size m = 11 using Linear probing with the primary hash function $h'(k) = k \bmod m$.

soln :-

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	N	N	N	N	N	N	N

Fig: Initial state of Hash Table

Now, using Linear Probing we have,

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

$$h(k; i) = [\{(k \bmod m) + i\} \bmod m]$$

→ Now for key 76

$$h(76, 0) = [\{(76 \bmod 11) + 0\} \bmod 11]$$

$$= [10 \bmod 11]$$

$$= 10$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	N	N	N	N	N	N	76

Fig: Inserting key i.e 76

→ Now Again for key 26

$$h(26,0) = [\{(26 \bmod 11) + 0\} \bmod 11]$$

$$= [4 \bmod 11]$$

$$= 4$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	N	N	N	N	N	76

fig: Inserting key i.e 26

→ Now Again For key 37

$$h(37,0) = [\{(37 \bmod 11) + 0\} \bmod 11]$$

$$= [4 \bmod 11]$$

$$= 4$$

since slot 4 is occupied, the next prob sequence is computed as:

$$h(37,1) = [\{(37 \bmod 11) + 1\} \bmod 11]$$

$$= [(4+1) \bmod 11]$$

$$= [5 \bmod 11]$$

$$= 5$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	37	N	N	N	N	76

Fig: Inserting key i.e 37

→ Now Again for key 5

$$h(5,0) = [\{(5 \bmod 11) + 0\} \bmod 11]$$

$$= [5 \bmod 11]$$

$$= 5$$

since slot 5 is occupied, the next prob sequence is computed as;

$$h(5,1) = [\{(5 \bmod 11) + 1\} \bmod 11]$$

$$= [6 \bmod 11]$$

$$= 6$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	37	5	N	N	N	76

Fig: Inserting Key i.e 5

→ Again for key 9

$$h(9,0) = [\{(9 \bmod 11) + 0\} \bmod 11]$$

$$= [9 \bmod 11]$$

$$= 9$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	37	5	N	N	9	76

Fig: Inserting key i.e 9

→ Again for key 21

$$h(21,0) = [\{(21 \bmod 11) + 0\} \bmod 11]$$

$$= [10 \bmod 11]$$

$$= 10$$

since slot 10 is occupied, the next prob sequence is computed as;

$$h(21,1) = [\{(21 \bmod 11) + 1\} \bmod 11]$$

$$= [11 \bmod 11]$$

$$= 0$$

0	1	2	3	4	5	6	7	8	9	10
21	N	N	N	26	37	5	N	N	9	76

Fig: Inserting key i.e 21

→ Again for key 65

$$h(65,0) = [\{(65 \bmod 11) + 0\} \bmod 11]$$

$$= [10 \bmod 11]$$

$$= 10$$

since slot 10 is occupied, the next prob sequence is computed as;

$$h(65,1) = [\{(65 \bmod 11) + 1\} \bmod 11]$$

$$= [11 \bmod 11]$$

$$= 0$$

since slot 0 is occupied, the next prob sequence is computed as;

$$h(65,2) = [\{(65 \bmod 11) + 2\} \bmod 11]$$

$$= [12 \bmod 11]$$

$$= 1$$

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

Fig: Inserting key i.e 65

→ Again for key 85

$$h(85, 0) = [\{(85 \bmod 11) + 0\} \bmod 11]$$

$$= [8 \bmod 11]$$

$$= 8$$

0	1	2	3	4	5	6	7	8	9	10
21	65	N	N	26	37	5	N	85	9	76

Fig : Final Hash Table

Q2. Let us consider a hash function h'(k)=k mod m where m=7 and sequence of keys as 50, 700, 76, 85, 92, 73, 101. Now store the keys using linear probing method.

Solution:

Using Linear Probing we have,

$$h(K,i) = [(h'(K) + i) \bmod m]$$

$$h(K,i) = [\{(K \bmod m) + i\} \bmod m]$$

→ Now For Key 50

$$h(50,0) = [\{(50 \bmod 7) + 0\} \bmod 7]$$

$$= 1 \bmod 7$$

$$= 1$$

→ Again For Key 700

$$h(700,0) = [\{(700 \bmod 7) + 0\} \bmod 7]$$

$$= 0 \bmod 7$$

$$= 0$$

→ Again For Key 76

$$h(76, 0) = [\{(76 \bmod 7) + 0\} \bmod 7]$$

$$= 6 \bmod 7$$

$$= 6$$

→ Again For Key 85

$$h(85, 0) = [\{(85 \bmod 7) + 0\} \bmod 7]$$

$$= 1 \bmod 7$$

$$= 1$$

Since slot 1 is already occupied, the next prob sequence is computed as:

$$h(85, 1) = [\{(85 \bmod 7) + 1\} \bmod 7]$$

$$= 2 \bmod 7$$

$$= 2$$

→ Again for key 92

$$h(92, 0) = [\{(92 \bmod 7) + 0\} \bmod 7]$$

$$= 1 \bmod 7$$

$$= 1$$

since slot 1 is occupied, the next prob sequence is computed as:

$$h(92, 1) = [\{(92 \bmod 7) + 1\} \bmod 7]$$

$$= 2 \bmod 7$$

$$= 2$$

since slot 2 is occupied, the next prob sequence is computed as:

$$h(92, 2) = [\{(92, \bmod 7) + 2\} \bmod 7]$$

$$= 3 \bmod 7$$

$$= 3$$

→ Again For Key 73

$$h(73,0) = [\{(73 \bmod 7) + 0\} \bmod 7]$$

$$= 3 \bmod 7$$

$$= 3$$

since slot 3 is occupied, the next prob sequence is computed as;

$$h(73,1) = [\{(73 \bmod 7) + 1\} \bmod 7]$$

$$= 4 \bmod 7$$

$$= 4$$

→ Again for key 101

$$h(101, 0) = [\{(101 \bmod 7) + 0\} \bmod 7]$$

$$= 3 \bmod 7$$

$$= 3$$

Since slot 3 is occupied, the next prob sequence is computed as:

$$h(101, 1) = [\{(101 \bmod 7) + 1\} \bmod 7]$$

$$= 4 \bmod 7$$

$$= 4$$

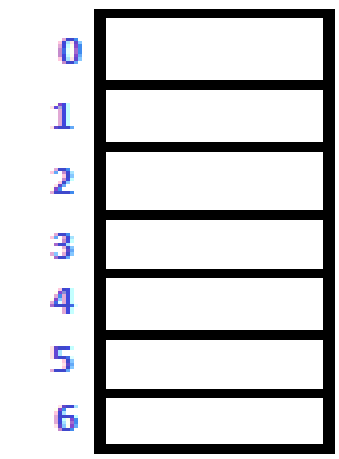
Since slot 4 is occupied, the next prob sequence is computed as:

$$h(101, 2) = [\{(101 \bmod 7) + 2\} \bmod 7]$$

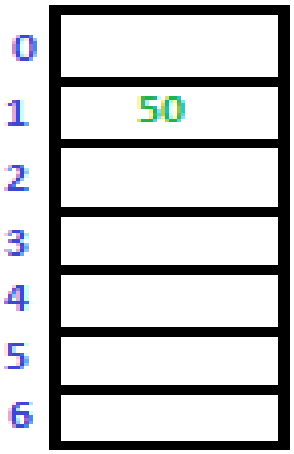
$$= 5 \bmod 7$$

$$= 5$$

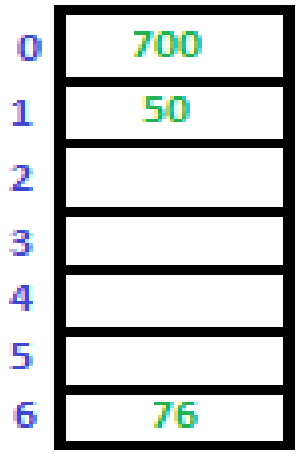
So, the storage of the data/key in the hash table can be represented as:



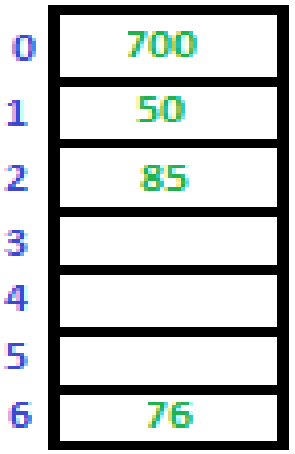
Initial Empty Table



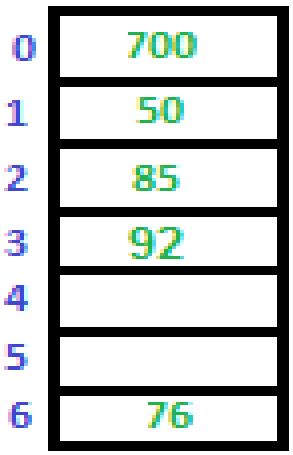
Insert 50



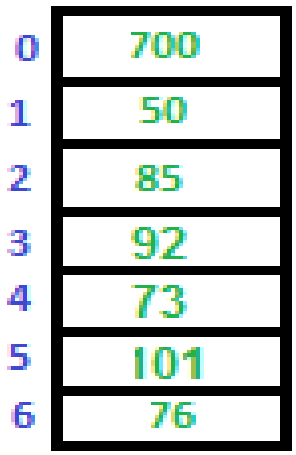
Insert 700 and 76



Insert 85: Collision Occurs, insert 85 at next free slot.



Insert 92, collision occurs as 50 is there at index 1. Insert at next free slot



Insert 73 and 101

Advantage

- It is easy to compute.

Disadvantage

- It takes time to search an element or to find an empty bucket.

- **Primary Clustering:** One of the problems with linear probing is Primary clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search for an element.

2. Quadratic Probing:

In quadratic probing,

- When collision occurs, we probe for i^2th bucket in i^{th} iteration.
- We keep probing until an empty bucket is found.
- To avoid the problem of primary clustering quadratic probing is used with following hash function.

$h(k, i) = [\, h'(k) + c_1 i + c_2 i^2 \,] \bmod m$
for i = 0,1,2,3,4……,m-1
where
m is size of hash table
h'(k)=k mod m is basic hash function
i is prob no.
c1 and c2 is constant

Q1. Insert keys i.e. 76, 26, 37, 5, 9, 21, 65, 85 into hash table of size m=11 using quadratic probing with the primary hash function h'(k)=k mod m and c1=1 and c2=3 respectively.

Solution:

Soln :-

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	N	N	N	N	N	N	N

Fig : Initial state of Hash Table

Now using Quadratic Probing we have,

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

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

→ Now for key 76

$$h(76,0) = [\, (76 \bmod 11) + 1\times 0 + 3\times 0^2 \,] \bmod 11$$

$$= 10 \bmod 11$$

$$= 10$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	N	N	N	N	N	N	76

Fig : Inserting key 76

→ Again For Key 26

$$h(26,0) = \left[(26 \bmod 11) + 1 \times 0 + 3 \times 0^2\right] \bmod 11$$

$$= 4 \bmod 11$$

$$= 4$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	N	N	N	N	N	76

Fig: Inserting Key 26

→ Again For Key 37

$$h(37,0) = \left[(37 \bmod 11) + 1 \times 0 + 3 \times 0^2\right] \bmod 11$$

$$= 4 \bmod 11$$

$$= 4$$

Since slot 4 is occupied, the next prob sequence is computed as:

$$h(37, 1) = [(37 \bmod 11) + 1 \times 1 + 3 \times 1^2] \bmod 11$$

$$= [4 + 1 + 3] \bmod 11$$

$$= 8 \bmod 11$$

$$= 8$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	N	N	N	37	N	76

Fig : Inserting Key 37

→ Again for Key 5

$$h(5, 0) = [(5 \bmod 11) + 1 \times 0 + 3 \times 0^2] \bmod 11$$

$$= 5 \bmod 11$$

$$= 5$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	5	N	N	37		76

Fig : Inserting Key 5

→ Again for key 9

$$h(9,0) = \left[(9 \bmod 11) + 1 \times 0 + 3 \times 0^2 \right] \bmod 11$$

$$= 9 \bmod 11$$

$$= 9$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	N	26	5	N	N	37	9	76

Fig : Inserting key 9

→ Again for key 21

$$h(21,0) = \left[(21 \bmod 11) + 1 \times 0 + 3 \times 0^2 \right] \bmod 11$$

$$= 10 \bmod 11$$

$$= 10$$

since slot 10 is occupied, the next prob sequence is computed as ;

$$h(21,1) = [(21 \bmod 11) + 1 \times 1 + 3 \times 1^2] \bmod 11$$

$$= [10 + 1 + 3] \bmod 11$$

$$= 14 \bmod 11$$

$$= 3$$

0	1	2	3	4	5	6	7	8	9	10
N	N	N	21	26	5	N	N	37	9	76

Fig: Inserting Key 21

→ Again for key 65

$$h(65,0) = [(65 \bmod 11) + 1 \times 0 + 3 \times 0^2] \bmod 11$$

$$= 10 \bmod 11$$

$$= 10$$

Since slot 10 is occupied, the next prob sequence is computed as:

$$h(65,1) = [(65 \bmod 11) + 1 \times 1 + 3 \times 1^2] \bmod 11$$

$$= [10 + 1 + 3] \bmod 11$$

$$= 14 \bmod 11$$

$$= 3$$

since slot 3 is occupied, the next prob sequence is computed as:

$$h(65,2) = [(65 \bmod 11) + 1 \times 2 + 3 \times 2^2] \bmod 11$$

$$= [10 + 2 + 12] \bmod 11$$

$$= 24 \bmod 11$$

$$= 2$$

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

Fig: Inserting Key 65

→ Again For key 85

$$h(85,0) = [(85 \bmod 11) + 1\times0 + 3\times0^2] \bmod 11$$

$$= 8 \bmod 11$$

$$= 8$$

Since slot 8 is occupied, the next prob sequence is computed as:

$$h(85,1) = [(85 \bmod 11) + 1\times1 + 3\times1^2] \bmod 11$$

$$= [8 + 1 + 3] \bmod 11$$

$$= 12 \bmod 11$$

$$= 1$$

0	1	2	3	4	5	6	7	8	9	10
N	85	65	21	26	5	N	N	37	9	76

Fig: Final Hash Table After inserting 85.

Do this question by yourself.

Q2. Let us consider a hash function h'(k)=k mod m where m=7 and sequence of keys as 50, 700, 76, 85, 92, 73, 101. Now store the keys using quadratic probing method where, c1=1 and c2=3 respectively.

Solution: