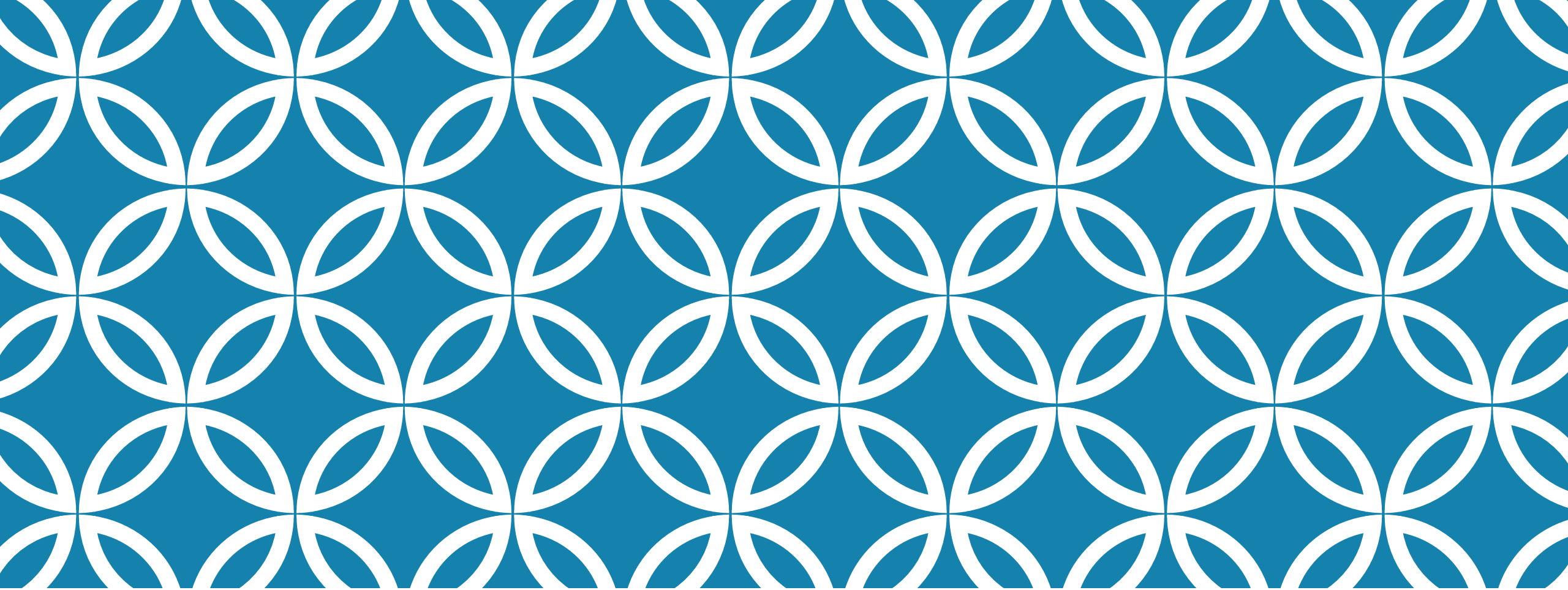


DATA STRUCTURE AND ALGORITHM

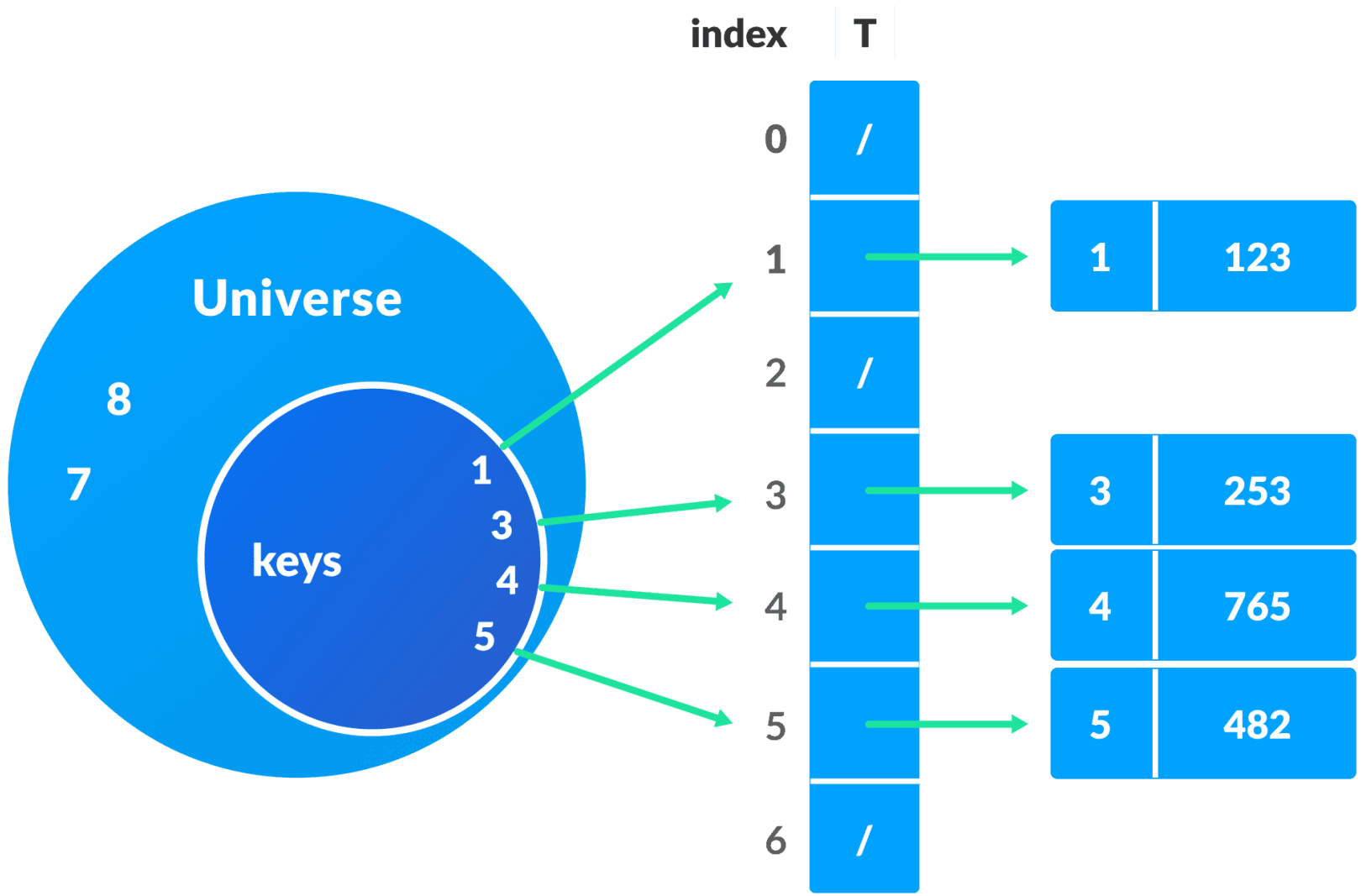
UNIT - 5

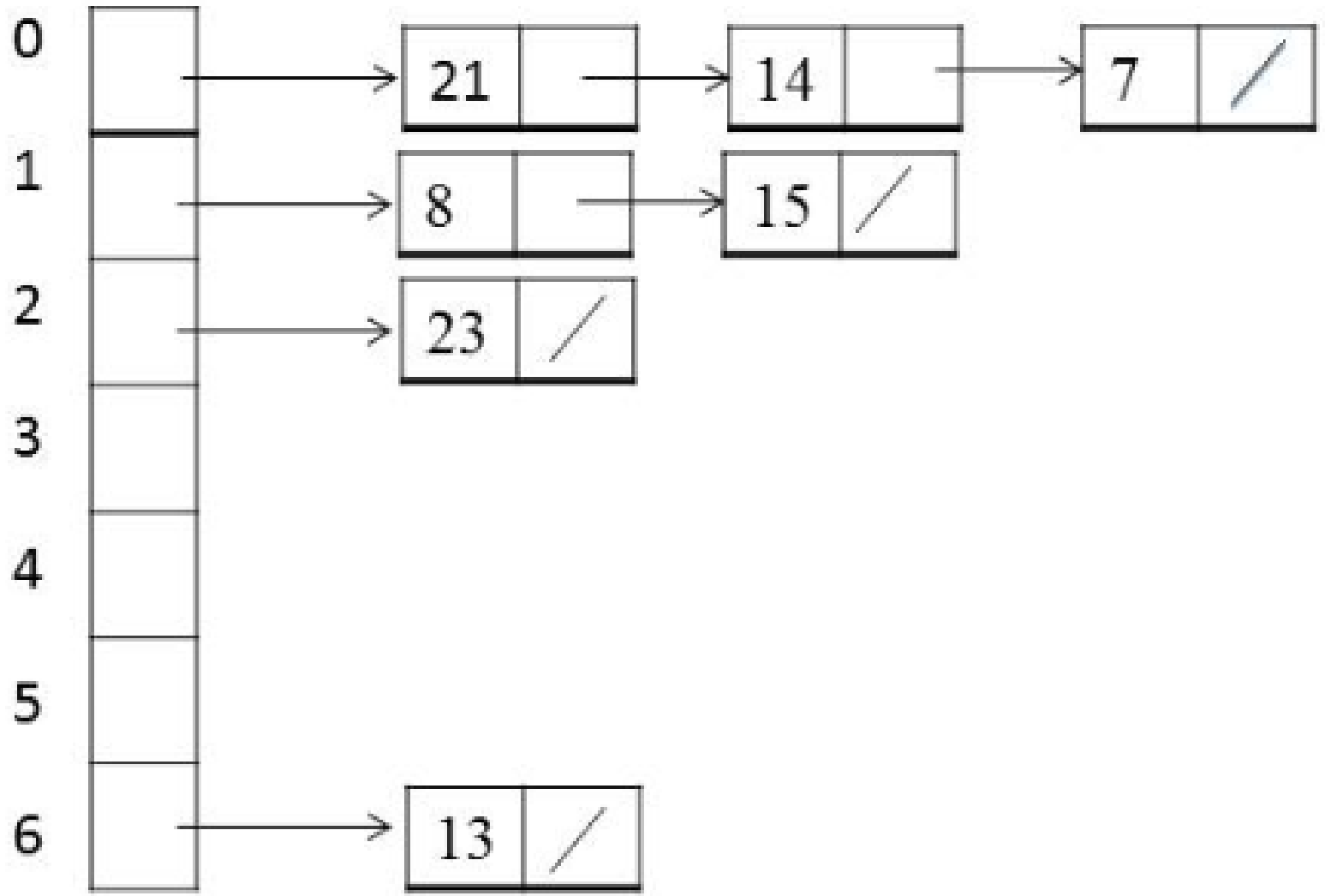
Hashing

Prepared by: Prof. Kinjal Patel.

INTRODUCTION

- Linear search has a running time proportional to O(n)
- while binary search takes time proportional to O(log n)
- But what if we want time proportional to O(1) means a constant time irrespective to the size of list?
- New technique to store element into the list using its **Key** value, and this can be implemented using **Hash Function and Hash Table.**
- **Hash Table** is a new data structure in which the location of a data item is determined directly as a function of data item itself rather than by a sequence of comparison
- In hashing Hash function was used for address-calculation sorting.
- **Collision** occurs when more than one key valued is mapped with same location.
- Collision can be minimized by collision-resolution techniques.


index
T
Universe
8
7
keys
1
3
4
5
0
/
1
1
123
2
/
3
3
253
4
4
765
5
5
482
6
/


0
1
2
3
4
5
6
21
14
7
8
15
23
13

HASHING FUNCTION

- Hash function h is simply a mathematical formula that manipulates the key in some form to compute the index for this key in the hash table.
- This process of mapping key to appropriate slots in hash table is known as **hashing**

Characteristics of a Good Hash Function

- A good hash function avoids collisions.
- A good hash function tends to spread keys evenly in the array.
- A good hash function is easy to compute

DIFFERENT HASHING FUNCTIONS

1. Division-Method
2. Mid square Methods
3. Folding Method
4. Digit Analysis
5. Length Dependent Method
6. Algebraic Coding
7. Multiplicative Hashing

DIVISION-METHOD

- In this method we use modular arithmetic system to divide the key value by some integer divisor m (may be table size).
- It gives us the location value, where the element can be placed.
- We can write,

$$L = (K \bmod m) + 1$$

where L = location in hash table/file

K = key value

m = table size/number of slots in file

Eg. k = 23, m = 10 then

L = (23 mod 10) + 1= 3 + 1=4, The key whose value is 23 is placed in 4th location.

Assume a table with 8 slots:

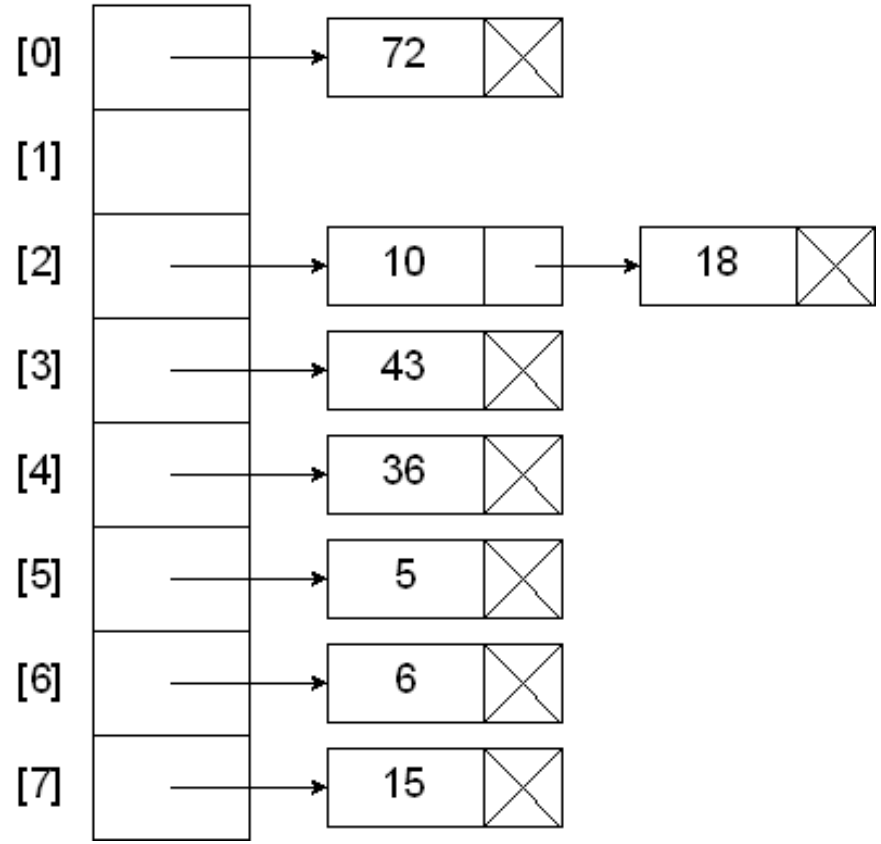
Hash key = key % table size

4 = 36 % 8
2 = 18 % 8
0 = 72 % 8
3 = 43 % 8
6 = 6 % 8

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Hash key = key % table size

4 = 36 % 8
2 = 18 % 8
0 = 72 % 8
3 = 43 % 8
6 = 6 % 8
2 = 10 % 8
5 = 5 % 8
7 = 15 % 8



MIDSQUARE METHODS

- A key is multiplied by itself and the address is obtained by selecting an appropriate number of bits or digits from the middle of the square.
- The number of bits and digits depends on the table size and can fit into one computer word of memory.
- Eg. K=123456

  square of key = 15241383936

  If three digits address is required , positions 5 to 7 could be chosen, giving address L = 138

  square of key = 1524**138**3936

FOLDING METHOD

- The key is actually partitioned into number of parts, each part having the same length as that of the required address with the possible exception of the last part.
- The parts are then added together, ignoring the final carry, to form an address
- The key are in binary form , the EX-OR operation may be substituted for addition
- Eg. Key= 356942781 and require 3-digits address then
- **Fold-shifting:** Here actual values of each parts of key are added
  - 356 + 942 + 781 = 2079
  - Here ignoring the final carry 2 , getting address L = 079
- **Fold-boundary:** Here the reversed values of outer parts of key are added
  - 653 + 942 + 187 = 1782
  - Here ignoring the final carry 1 , getting address L = 782

DIGIT ANALYSIS

- This hashing function is a distribution-dependent
- Digit analysis forms address by selecting and shifting digits or bits of the original key
- Eg. Key = 7546123

  L = 2164 (select digits in positions 3 to 6 and reverse their order)
- In some cases , statistical analysis as revealed the fact that the positions 2,4,5,9 have the most uniform distribution of digits, so that places are selected.
- Eg. Key = 1234567890

  L = 9542 (select digits from that places and reverse their order)

LENGTH DEPENDENT METHOD

- In this type of hashing function we use the length of the key along with some portion of the key to produce the address, directly
- In indirect method, the length of the key along with some portion of the key to produce intermediate key and then apply division method to produce address location.

ALGEBRAIC CODING

- Here a n bit key value is represented as a polynomial.
- The divisor polynomial is then constructed based on the address range required.
- The modular division of key-polynomial by divisor polynomial, to get the address-polynomial.
- Let f(x) = polynomial of n bit key = $a_1 + a_2x + \ldots\ldots + a_nx^{n-1}$
- d(x) = divisor polynomial = $x^1 + d_1 + d_2x + \ldots + d_1x^{1-1}$
- then the required address polynomial will be f(x) mod d(x)

MULTIPLICATIVE HASHING

- This method is based on obtaining an address of a key, based on the multiplication value.
- If X is the non-negative key, and a constant c, (0 < c < 1), compute

  $H(X) = [m(CX \bmod 1)] + 1$

  $CX \bmod 1$ which is a fractional part of CX and [ ] denotes the greatest integer less than or equal to its content.
- This Multiplicative Hashing function give good result if the constant c is properly chosen.

WHAT IS COLLISION?

- hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique.

- There are mainly two methods to handle collision:

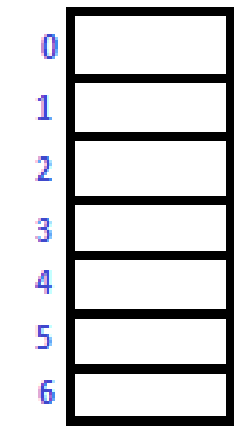
1) Closed Addressing (Open Hashing)
   - Separate Chaining

2) Open Addressing (Closed Hashing)
   - Linear probing
   - Quadratic probing
   - Double hashing

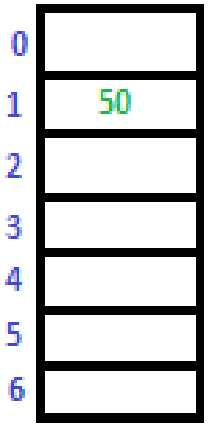
SEPARATE CHAINING

The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

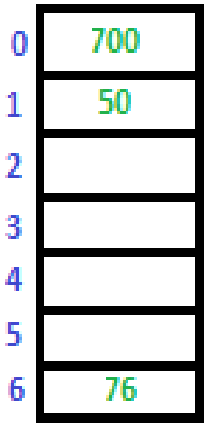
Let us consider a simple hash function as "**key mod 7**" and sequence of keys as 50, 700, 76, 85, 92, 73, 101



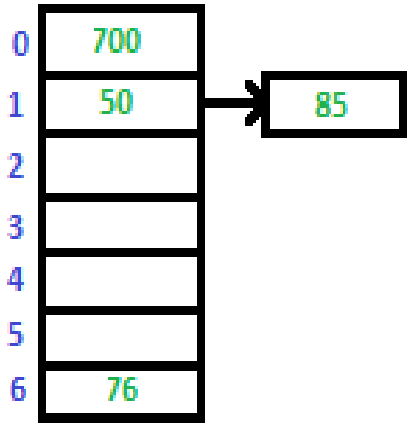
Initial Empty Table



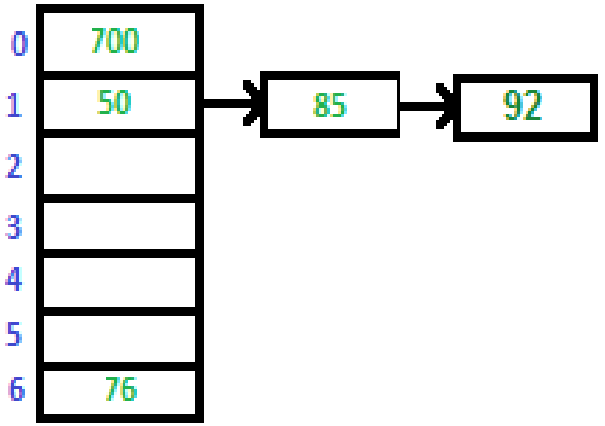
Insert 50



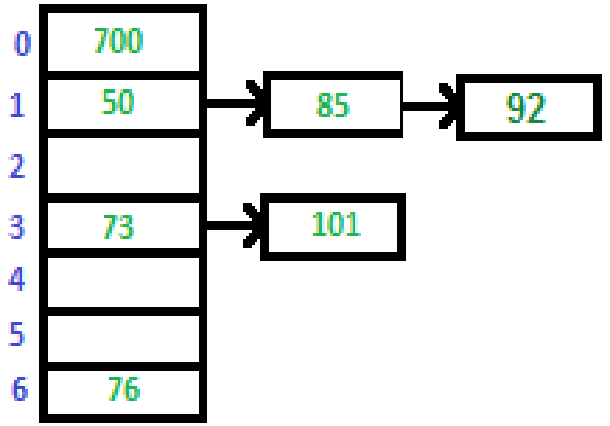
Insert 700 and 76



Insert 85: Collision Occurs, add to chain



Inser 92 Collision Occurs, add to chain



Insert 73 and 101

Advantages:

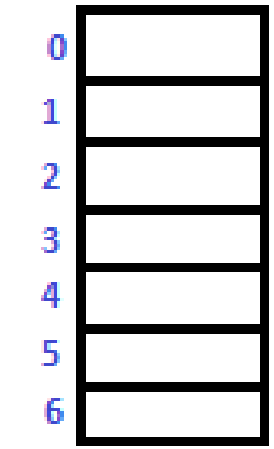
1) Simple to implement.
2) Hash table never fills up, we can always add more elements to the chain.
3) Less sensitive to the hash function or load factors.
4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Disadvantages:

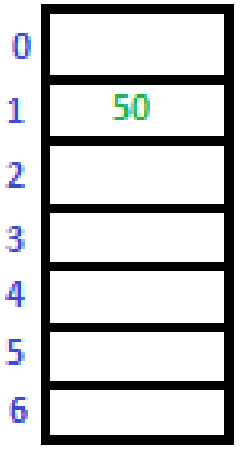
1) Cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table.
2) Wastage of Space (Some Parts of hash table are never used)
3) If the chain becomes long, then search time can become O(n) in the worst case.
4) Uses extra space for links.

LINEAR PROBING

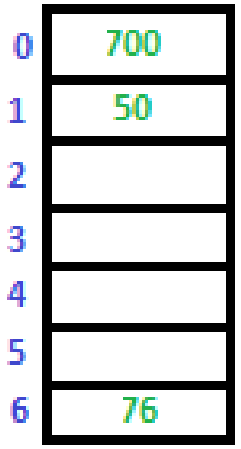
- Insert : When the hash function causes a collision by mapping a new key to a cell of the hash table that is already occupied by another key, linear probing searches the table for the closest following free location and inserts the new key there
- Insert Ki at first free location from (u+ i) % m where I = 0 to (m-1)
- Search : Searching the table sequentially starting at the position given by the hash function, until finding a cell with a matching key or an empty cell.
- It is easy to compute.
- **Disadvantage-**
  - The main problem with linear probing is clustering.
  - Many consecutive elements form groups.
  - Then, it takes time to search an element or to find an empty bucket.



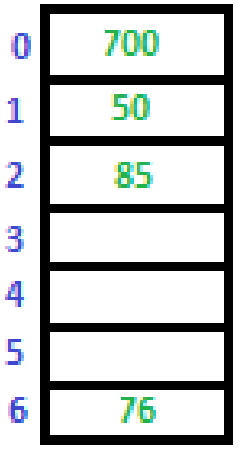
Initial Empty Table



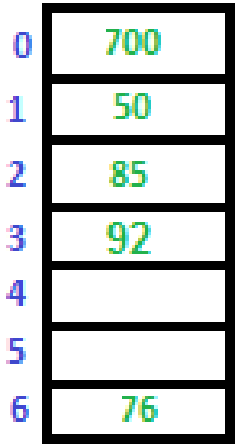
Insert 50



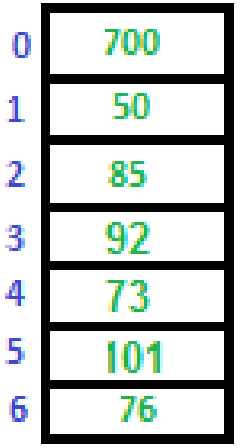
Insert 700 and 76



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

The Keys = 3,2,9,6,11,13,7,12 inserted into an initially empty hash table of length 10 using open addressing with hash function h(k)= 2k+3 mod 10 and linear probing. What is the content of hash table?

A[0]	13
A[1]	9
A[2]	12
A[3]	
A[4]	
A[5]	6
A[6]	11
A[7]	2
A[8]	7
A[9]	3

Key	Location (u)	Probs
3	2*3 +3 % 10 =9	1
2	2*2 +3 % 10 =7	1
9	2*9 +3 % 10 =1	1
6	2*6 +3 % 10 =5	1
11	2*11 +3 % 10 =5	2
13	2*13 +3 % 10 =9	2
7	2*7 +3 % 10 =7	2
12	2*12 +3 % 10 =7	6
Hash table : 13,9,12,__,__,6,11,2,7,3		

QUADRATIC PROBING

- One way of reducing "primary clustering" is to use quadratic probing to resolve collision.
- Suppose the "key" is mapped to the location j and the cell j is already occupied. In quadratic probing, the location j, (j+1), (j+4), (j+9), ... are examined to find the first empty cell where the key is to be inserted.
- This table reduces primary clustering.
- It does not ensure that all cells in the table will be examined to find an empty cell. Thus, it may be possible that key will not be inserted even if there is an empty cell in the table.

- Insert Ki at first free location from $(u + i^2) \% m$ where $I = 0$ to $(m-1)$

The Keys = 3,2,9,6,11,13,7,12 inserted into an initially empty hash table of length 10 using open addressing with hash function h(k)= 2k+3 mod 10 and quadratic probing

A[0]	13
A[1]	9
A[2]	
A[3]	12
A[4]	
A[5]	6
A[6]	11
A[7]	2
A[8]	7
A[9]	3

Key	Location (u)	Probs
3	2*3 +3 % 10 =9	1
2	2*2 +3 % 10 =7	1
9	2*9 +3 % 10 =1	1
6	2*6 +3 % 10 =5	1
11	2*11 +3 % 10 =5	2
13	2*13 +3 % 10 =9	2
7	2*7 +3 % 10 =7	2
12	2*12 +3 % 10 =7	5
Hash table : 13,9,___,12,___,6,11,2,7,3		

DOUBLE HASHING

- This method requires two hashing functions f1 (key) and f2 (key).
- Problem of clustering can easily be handled through double hashing.
- Function f1 (key) is known as primary hash function.
- In case the address obtained by f1 (key) is already occupied by a key, the function f2 (key) is evaluated.
- The second function f2 (key) is used to compute the increment to be added to the address obtained by the first hash function f1 (key) in case of collision.
- The search for an empty location is made successively at the addresses f1 (key) + f2(key), f1 (key) + 2f2 (key), f1 (key) + 3f2(key),...

- Insert Ki at first free location from (u + v*i) % m where I = 0 to (m-1)

The Keys = 3,2,9,6,11,13,7,12 inserted into an initially empty hash table of length 10 using open addressing with hash function h1(k)= 2k+3 and h2(k) = 3k+1

A[0]	
A[1]	9
A[2]	
A[3]	11
A[4]	12
A[5]	6
A[6]	
A[7]	2
A[8]	
A[9]	3

Key	Location (u)	v	Probs
3	2*3 +3 % 10 =9		1
2	2*2 +3 % 10 =7		1
9	2*9 +3 % 10 =1		1
6	2*6 +3 % 10 =5		1
11	2*11 +3 % 10 =5	3*11+1 %10 =4	3
13	2*13 +3 % 10 =9	3*13 + 1 % 10 =0	Key not inserted
7	2*7 +3 % 10 =7	3*7 + 1 %10 =2	Key not inserted
12	2*12 +3 % 10 =7	3 *12 +1 %10 =7	2
Hash table : __,9,__,11,12,6,__,2,__3			