

HASH TABLES

INSTRUCTOR: KASHFIA SAILUNAZ

SLIDES ADAPTED FROM THE TEXTBOOK (CHAPTER 11) & ENSF 593/594 LECTURE BY MOHAMMAD MOSHIRPOUR



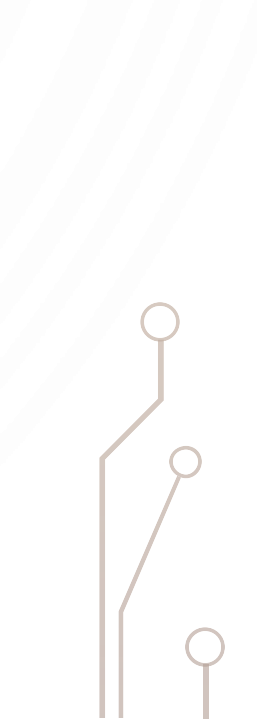


OUTLINE

- Hashing
 - Hash Tables
 - Hash Functions
 - Hashing Algorithm Idea
 - Hash Operations
 - Collision Resolution
- 
- 
- 

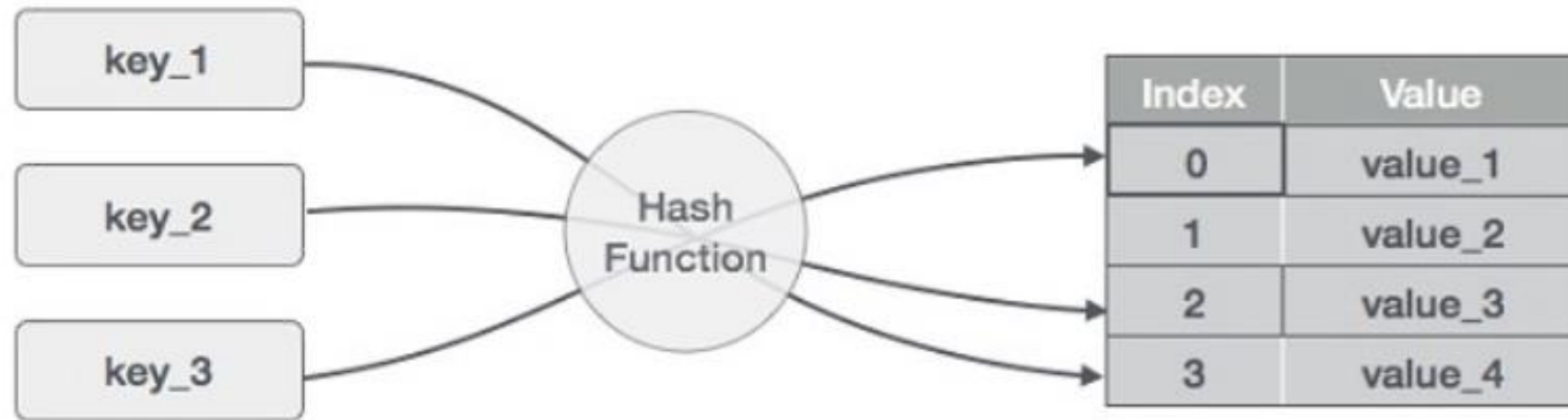


LEARNING OUTCOME

- At the end of this lecture, we will be able to-
 - Understand the basics of hashing, hash tables, hash functions, and
 - Discuss various hashing techniques.
- 
- 
- 

HASHING

- A technique for mapping keys or elements
- It produces a fixed sized output from a variable sized input
- It maps elements into a hash table using some hash functions
- It converts key values into indexes
- It provides a unique identifier to an element or a set of elements to access all information of that element



HASHING

- Example

(Key, Value)

(1,20)

(2,70)

(42,80)

(4,25)

(12,44)

(14,32)

(17,11)

(13,78)

(37,98)

Sr.No.	Key	Hash	Array Index
1	1	$1 \% 20 = 1$	1
2	2	$2 \% 20 = 2$	2
3	42	$42 \% 20 = 2$	2
4	4	$4 \% 20 = 4$	4
5	12	$12 \% 20 = 12$	12
6	14	$14 \% 20 = 14$	14
7	17	$17 \% 20 = 17$	17
8	13	$13 \% 20 = 13$	13
9	37	$37 \% 20 = 17$	17

HASHING

- Steps of hashing-

- The key to an element or set of elements is passed through a hash function to be converted to an integer index
- The element or set of elements are then stored in the hash table using that index

- Hashing makes it faster to search for an element using the hash key

- The hash function and hash tables are needed to extract the original elements

- Applications of hashing

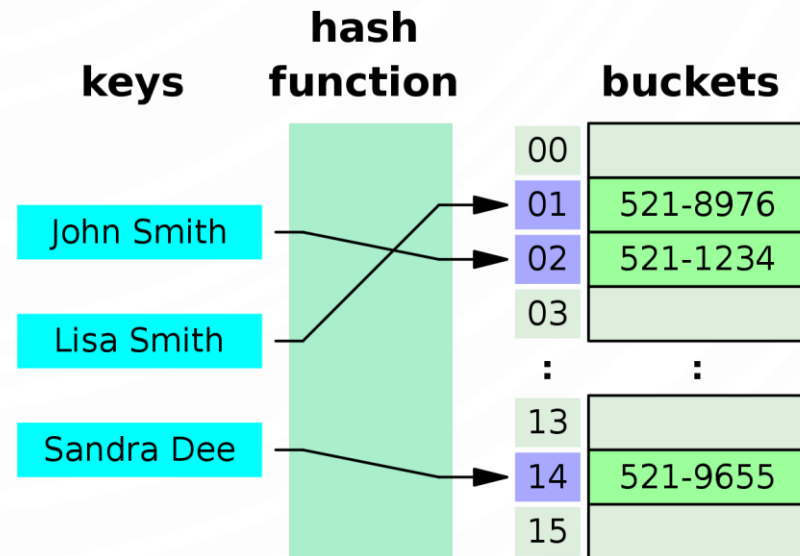
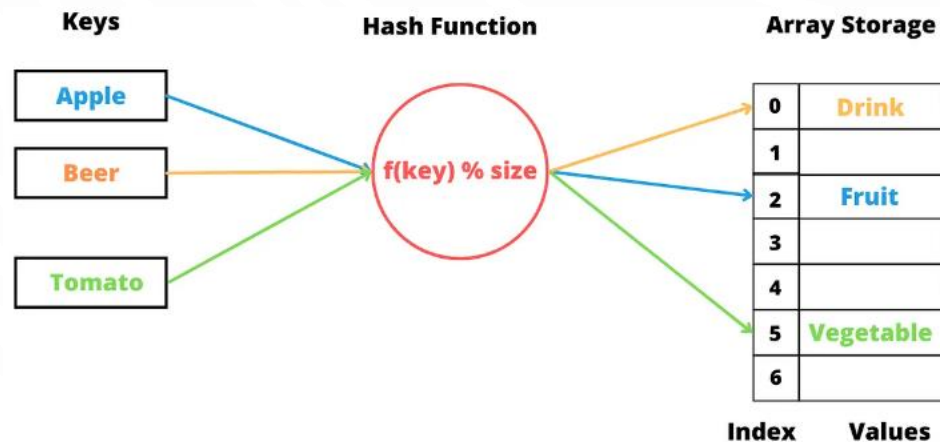
- Data dictionary
- Password verification
- Data encryption
- Transaction verification, etc.

HASH TABLE

- A data structure to store information
- It generalizes the idea of an array
- Works with two components –
 - Key – unique integer used for indexing
 - Value – data element(s) associated with key
- Data is stored in an associative way
- Each data is mapped to an array position using the hash function on the unique key
- Allows direct access to data in $O(1)$ time

HASH TABLE

- Operations
 - Insert
 - Delete
 - Search
- Hash table nodes have no predecessors or successors
- Not suitable for ordered lists or searches that need predecessors or successors



HASH TABLE

- Arrays vs. Hash Tables

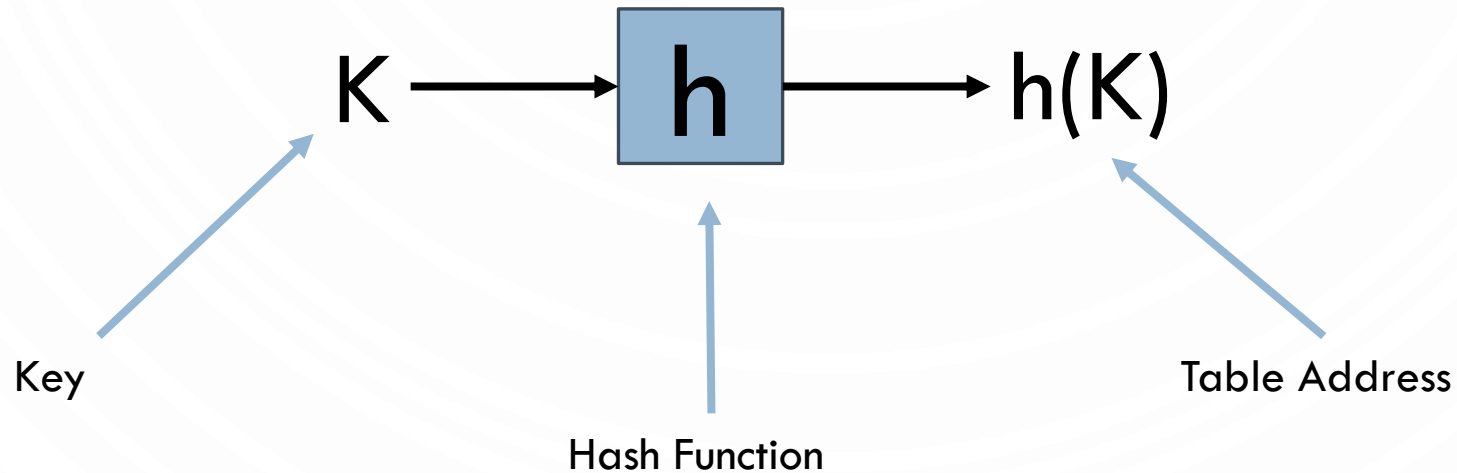
- Any elements in an array can be directly accessed using the array index, whereas the hash function and key are needed to get the data location
- Arrays are fixed sized, whereas hash tables can have flexible sizes
- Elements are inserted in an order in Arrays, whereas there are no ordering in inserting elements in hash tables
- Searching in array needs to loop through the array elements, whereas searching a hash table can directly point to the search element
- Insertion and deletion in an array needs shifting the elements, whereas hash tables can insert or delete elements without effecting other elements

HASH TABLE

- Assume an array A , a search key K , search element S
- If $A[K] = S$, then we can find the element directly. But generally, that is not the case with arrays.
- With hash tables, we compute $A[h(K)]$ to find S as $A[h(K)] = S$, where h is a hash function
- In hash tables, accessing any element doesn't depend on n , so searching takes constant time $O(1)$
- In arrays, linear search takes $O(n)$ and binary search takes $O(\lg n)$

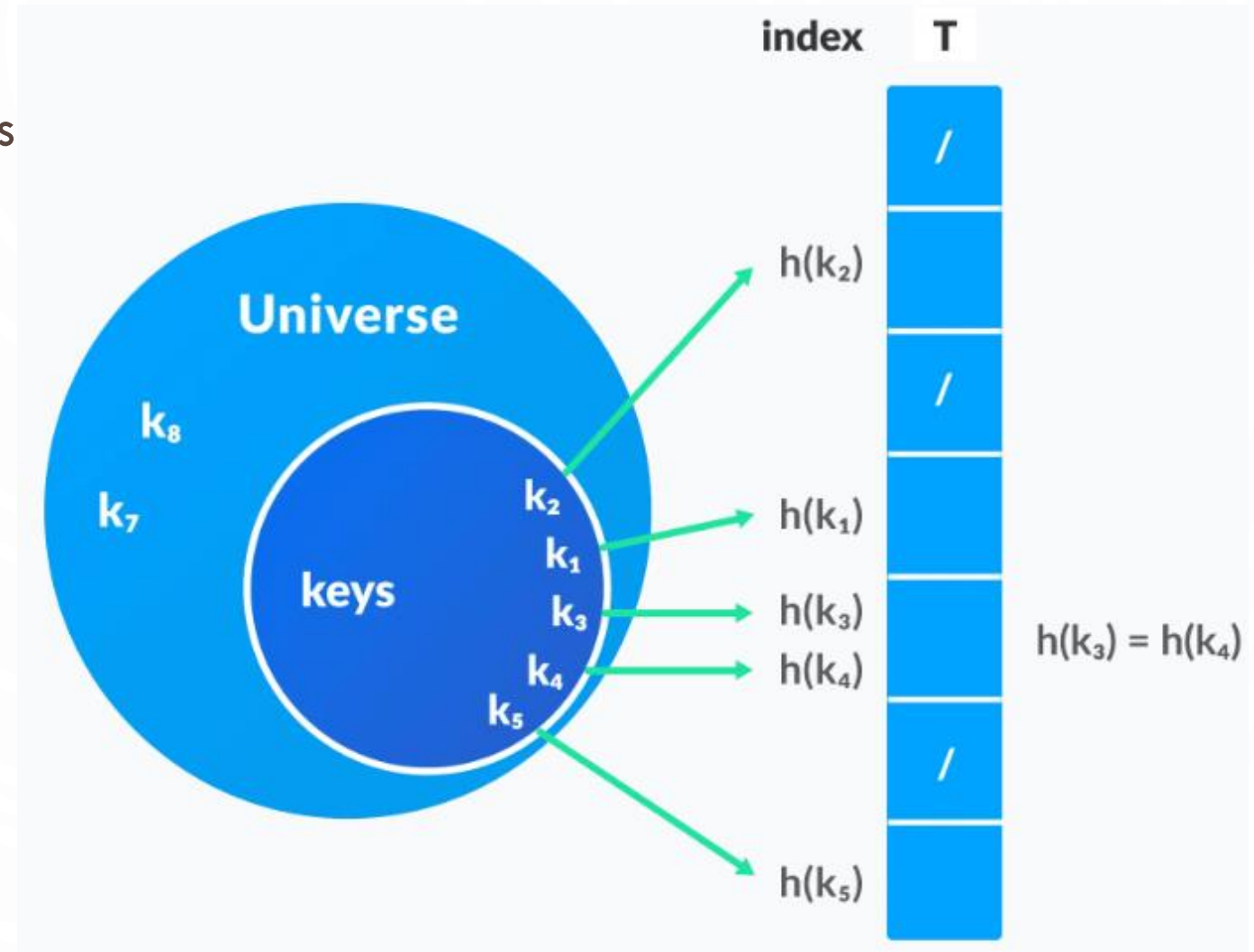
HASH FUNCTIONS

- A function that converts any inputs to an integer (i.e., index)
- Transforms a key into hash table address



HASH FUNCTIONS

- Table address is used to access elements in hash table
- Key K_1 hashes to $h(K_1)$



HASH FUNCTIONS

- Some cells of the tables may be unused
- Some cells of the table may have collisions
 - i.e., if more than one key have same hash index

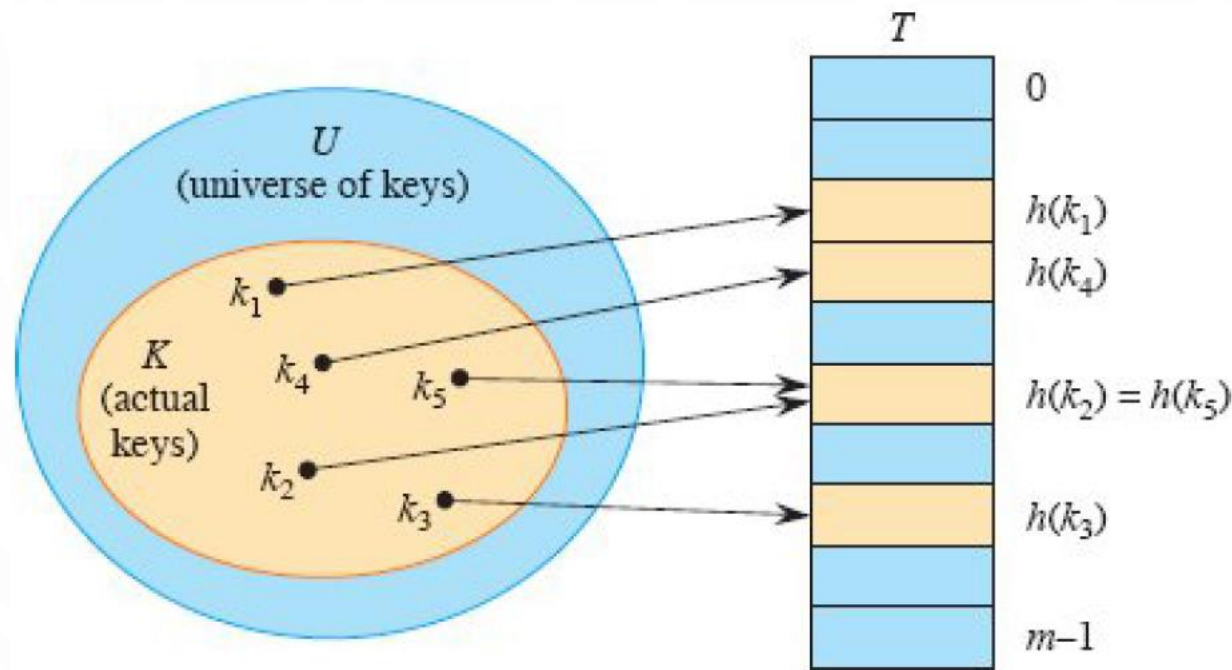


Figure 11.2 Using a hash function h to map keys to hash-table slots. Because keys k_2 and k_5 map to the same slot, they collide.

HASH FUNCTIONS

- **Perfect Hash Function**
 - Always transforms keys into unique hash indices
 - Never generates collision
 - Difficult to achieve in real-time systems
- Hash function choices depend on
 - Nature of the keys
 - Density requirements of the hash table
 - Collision avoidance or not
- Generally, a hash function is chosen that can randomly distribute the data in a simple uniform distribution

HASHING ALGORITHM IDEA

- Most hashing algorithms have 3 major steps-
 - Step 1: Represent keys in numerical forms
 - For example, if the key is already not a number, and a string, then take the ASCII code of each character
 - Step 2: Apply arithmetic operations on the key to generate another numerical value
 - For example, take the summation of the ascii codes
 - Step 3: Divide by the size of the hash table and take the reminder
 - For example, if the size of the hash table is n , then the results will be from 0 to $n-1$

HASHING ALGORITHM IDEA

- Step 2 can have lots of variations –
 - Simple addition
 - Fold and Add (shift folding)
 - Reverse fold and Add (boundary folding)
 - Square (square the value)
 - Mid-square (square the value and take only the middle part)
 - Random bit or digit extraction
 - Radix transformation (use any base other than 10), etc.

HASH OPERATIONS

- DIRECT-ADDRESS-SEARCH(T, k)

return $T[k]$

- DIRECT-ADDRESS-INSERT(T, x)

$T[x.key] = x$

- DIRECT-ADDRESS-DELETE(T, x)

$T[x.key] = \text{NIL}$

HASH OPERATIONS

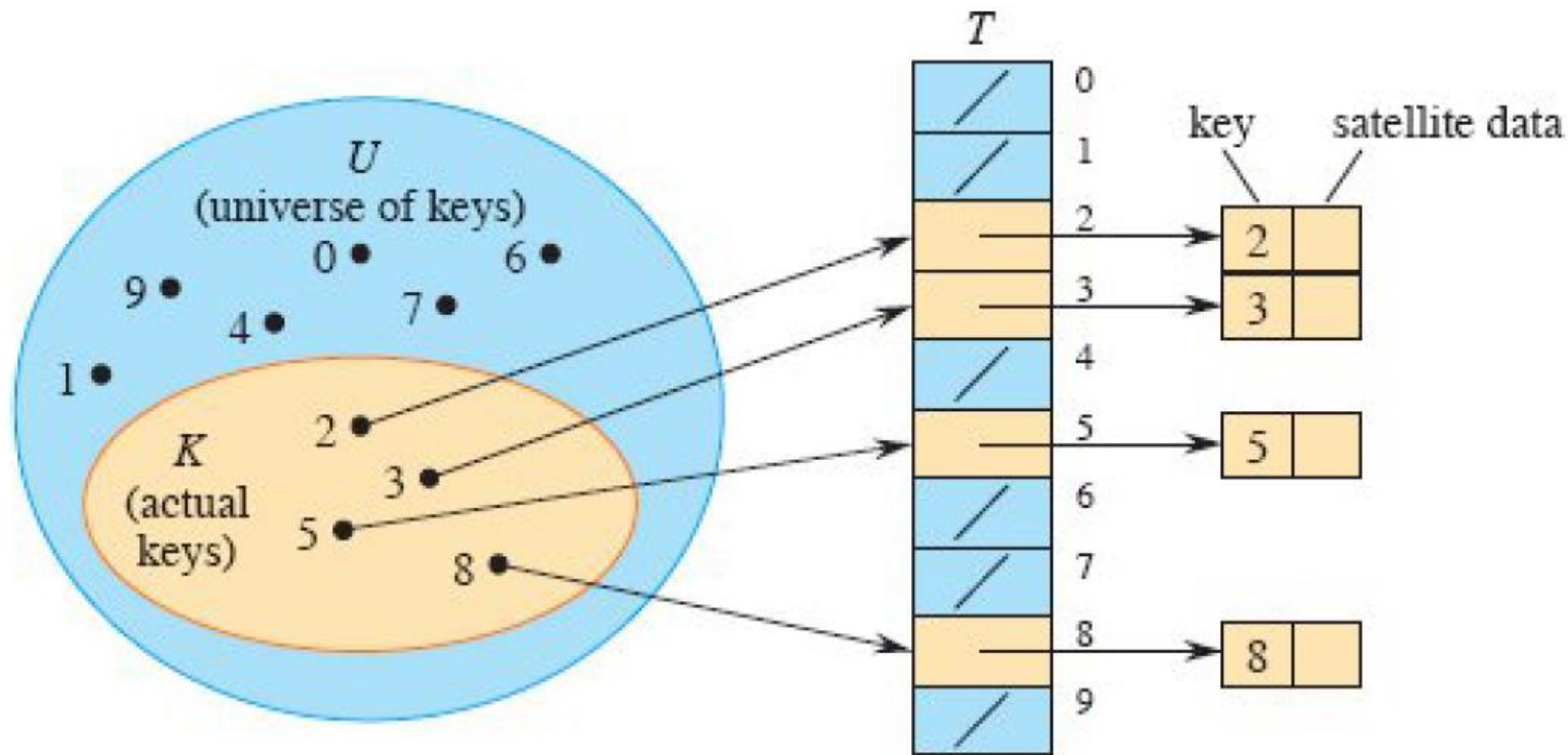
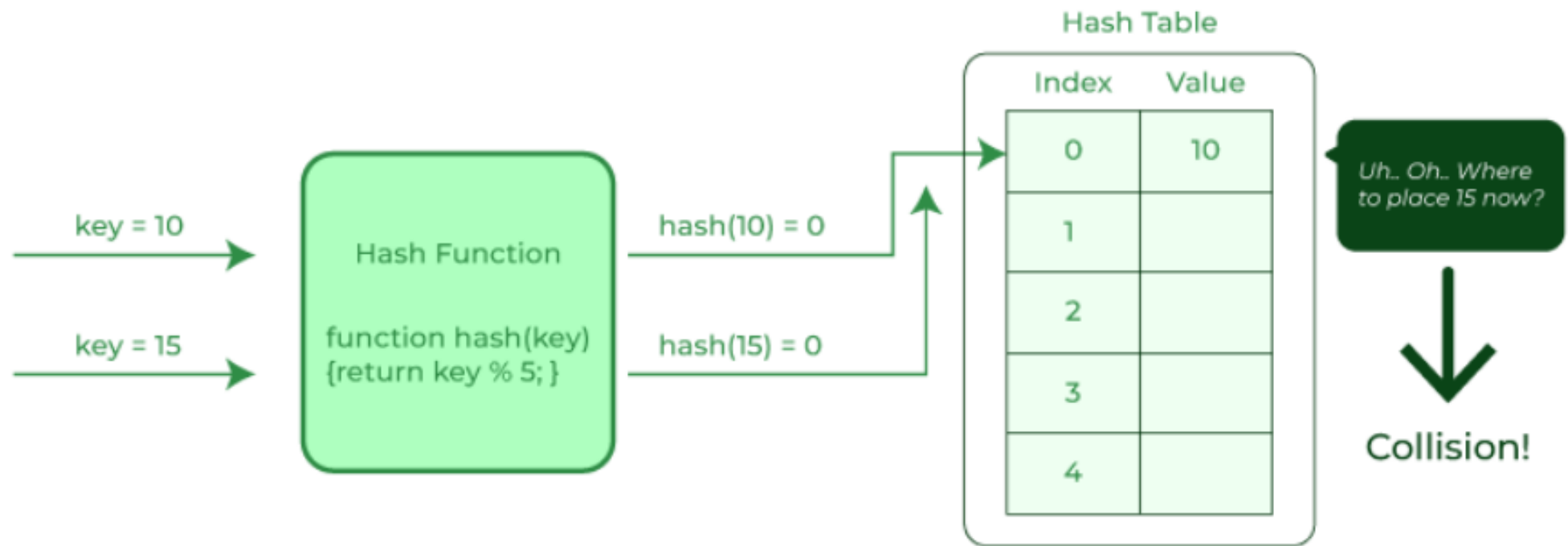


Figure 11.1 How to implement a dynamic set by a direct-address table T . Each key in the universe $U = \{0, 1, \dots, 9\}$ corresponds to an index into the table. The set $K = \{2, 3, 5, 8\}$ of actual keys determines the slots in the table that contain pointers to elements. The other slots, in blue, contain NIL.

COLLISION RESOLUTION

- When multiple keys generate the same hash index, a collision occurs



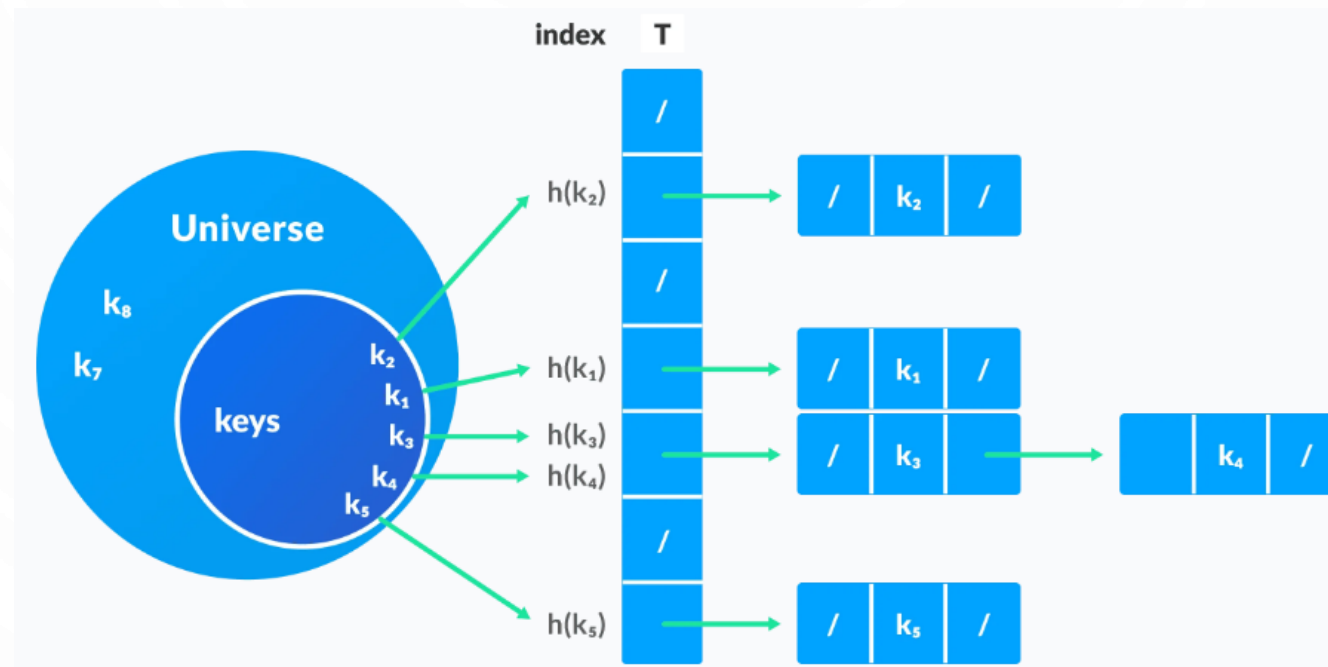
COLLISION RESOLUTION

- Many ways to solve a collision-
 - Find a hash function to avoid collision altogether
 - A perfect hash function
 - Difficult to find
 - Reduce collisions to an acceptable range
 - Find a hash function that can uniformly distribute elements
 - Increase the table size
 - Add more than one element at the same address
 - Separate chaining
 - Bucket addressing
 - Insert the element some other place than the hash index
 - Open addressing
 - Coalesced chaining

COLLISION RESOLUTION

- External/Separate Chaining

- Each position in the table is associated with a linked list
- If linked lists are short and ordered, search can be faster
- Takes up larger space for longer linked lists

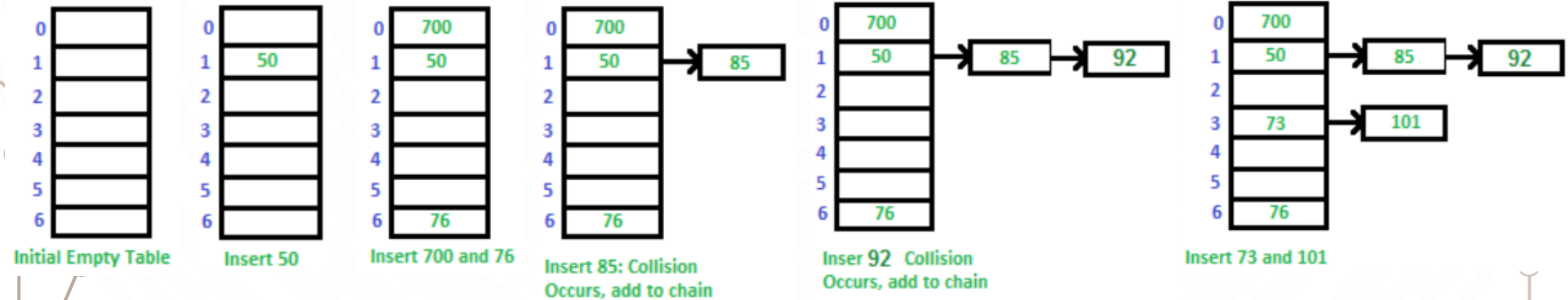


Source: <https://www.programiz.com/dsa/hash-table>

COLLISION RESOLUTION

- External/Separate Chaining

Keys: 50, 700, 76, 85, 92, 73, 101 Hash Function = $\text{Key} \% 7$



Source: <https://www.geeksforgeeks.org/separate-chaining-collision-handling-technique-in-hashing/>

COLLISION RESOLUTION

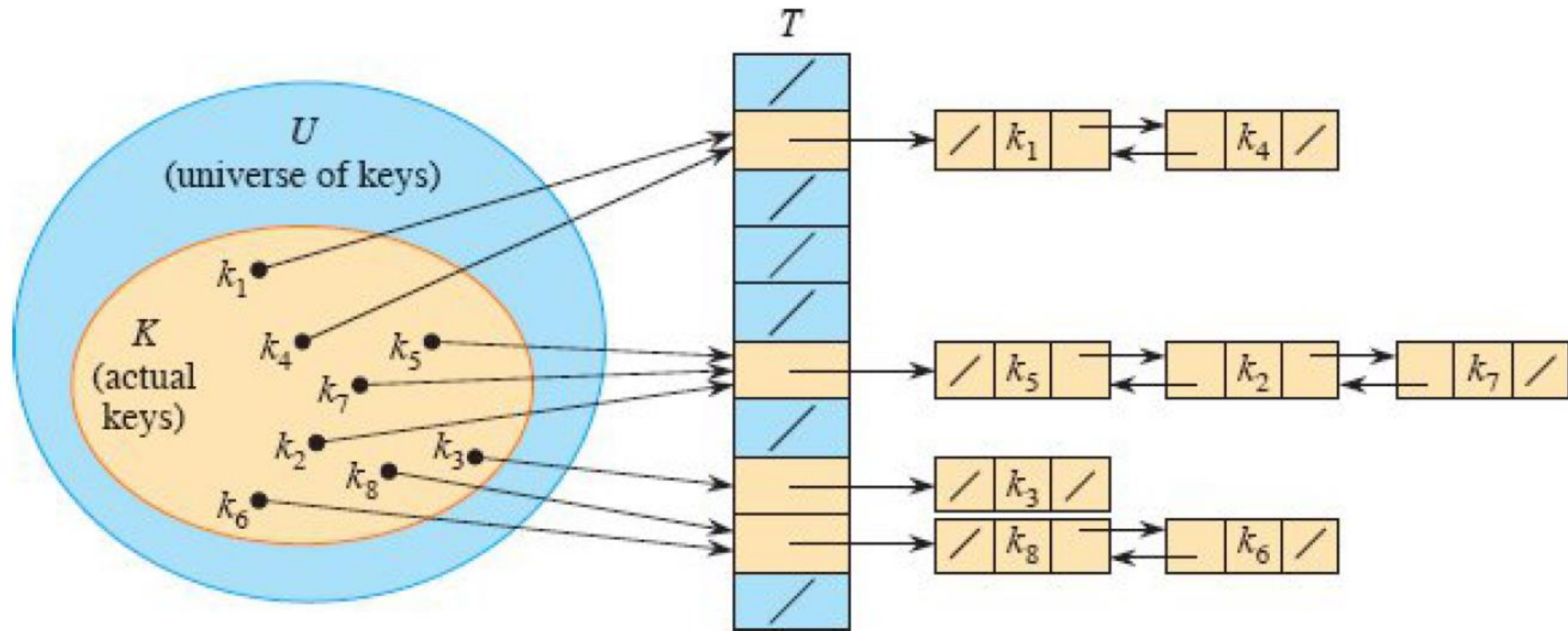


Figure 11.3 Collision resolution by chaining. Each nonempty hash-table slot $T[j]$ points to a linked list of all the keys whose hash value is j . For example, $h(k_1) = h(k_4)$ and $h(k_5) = h(k_2) = h(k_7)$. The list can be either singly or doubly linked. We show it as doubly linked because deletion may be faster that way when the deletion procedure knows which list element (not just which key) is to be deleted.

COLLISION RESOLUTION

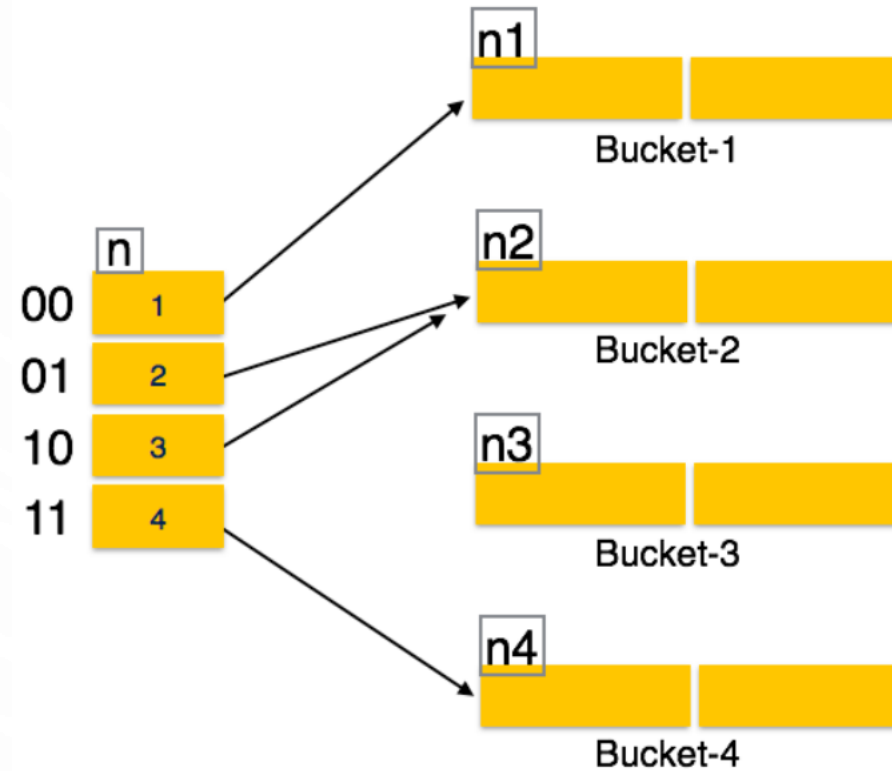
- Chaining

- CHAINED-HASH-INSERT(T, x)
LIST-PREPEND($T[h(x.key)], x$)
- CHAINED-HASH-SEARCH(T, k)
return LIST-SEARCH($T[h(k)], k$)
- CHAINED-HASH-DELETE(T, x)
LIST-DELETE($T[h(x.key)], x$)

COLLISION RESOLUTION

- Bucket Addressing

- Each table element is considered as a block of memory or 2D array
- Not efficient in terms of space
- Buckets can overflow



COLLISION RESOLUTION

- Open Addressing

- Collisions are resolved by finding an 'open' position in the has table
- If for k , position $h(k)$ is occupied, then the hash table is probed until
 - an open position is found
 - the same position is tried repeatedly
 - table is full
- Probing position is found in different ways-
 - Linear probing
 - Quadratic probing
 - Double hashing

COLLISION RESOLUTION

- Open Addressing
 - Linear probing – sequentially searches after $h(k)$ for an open position

Insert (76)		Insert (93)		Insert (40)		Insert (47)		Insert (10)		Insert (55)	
$76\%7 = 6$		$93\%7 = 2$		$40\%7 = 5$		$47\%7 = 5$		$10\%7 = 3$		$55\%7 = 6$	
0		0		0		0	47	0	47	0	47
1		1		1		1		1		1	55
2		2	93	2	93	2	93	2	93	2	93
3		3		3		3		3	10	3	10
4		4		4		4		4		4	
5		5		5	40	5	40	5	40	5	40
6	76	6	76	6	76	6	76	6	76	6	76

COLLISION RESOLUTION

- Open Addressing

- Quadratic probing – searches after $h(k)$ with quadratic polynomial for an open position
- i.e., after $h(k)$ it will look for $h(k)+1^2$, $h(k)-1^2$, $h(k)+2^2$, $h(k)-2^2$, etc.

0	
1	
2	
3	
4	
5	
6	

Initial Empty Table

0	
1	50
2	
3	
4	
5	
6	

Insert 50

0	700
1	50
2	
3	
4	
5	
6	76

Insert 700
and 76

0	700
1	50
2	85
3	
4	
5	
6	76

Insert 85:
Collision occurs.
Insert at $1 + 1^2$ position

0	700
1	50
2	85
3	
4	
5	92
6	76

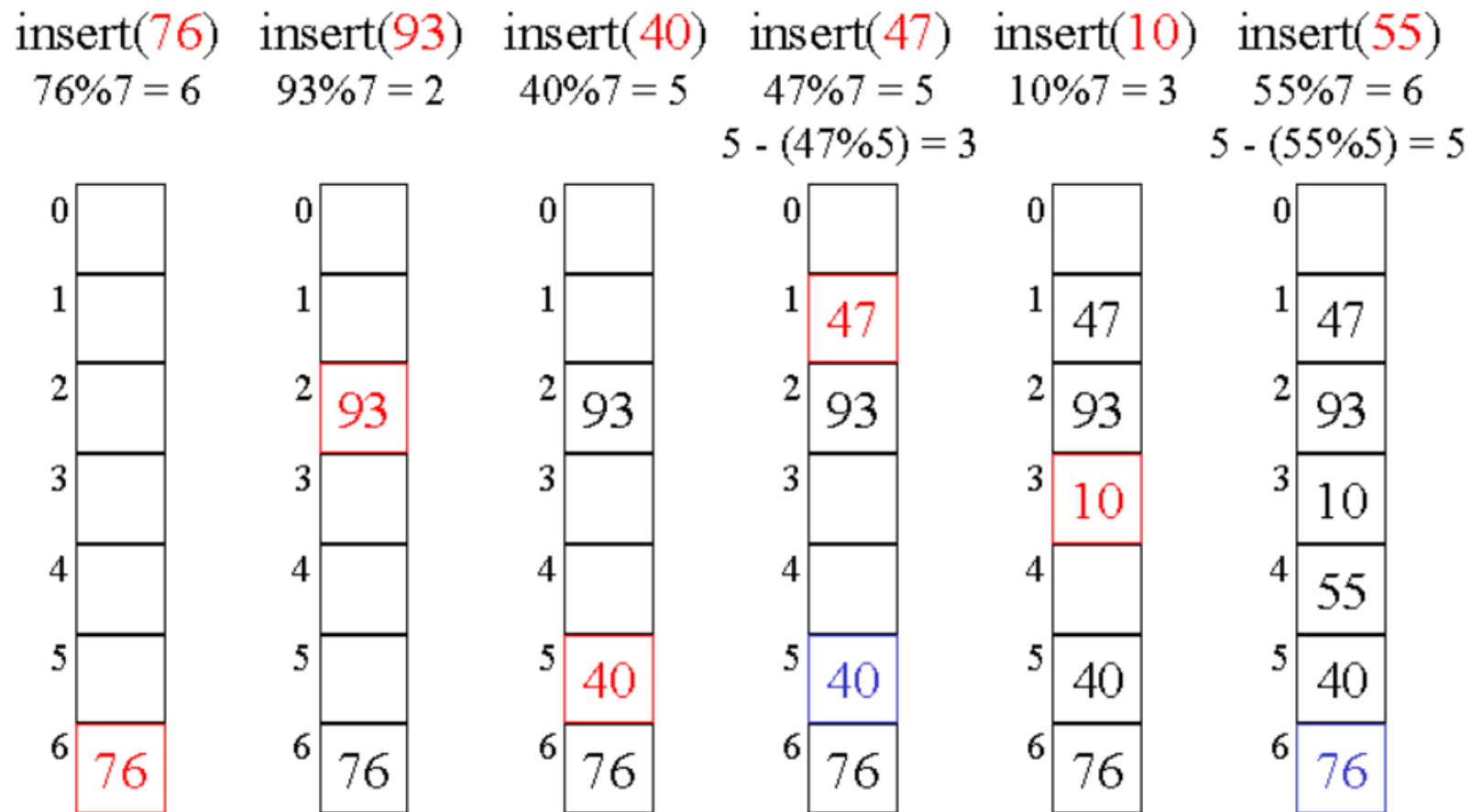
Insert 92:
Collision occurs at 1.
Collision occurs at $1 + 1^2$ position
Insert at $1 + 2^2$ position.

0	700
1	50
2	85
3	73
4	101
5	92
6	76

Insert 73 and 101

COLLISION RESOLUTION

- Open Addressing
 - Double hashing – using a 2nd hash function



SUMMARY

- Hashing needs a hash function to convert a key to hash index to store data in hash table
- Hashing is applied for data security, efficient data storing and searching
- Average case time complexity for searching in hash tables is $O(1)$
- Various open and closed hashing techniques can be applied for hashing data using linear 1D or 2D arrays, linked lists, and memory blocks



THANK YOU