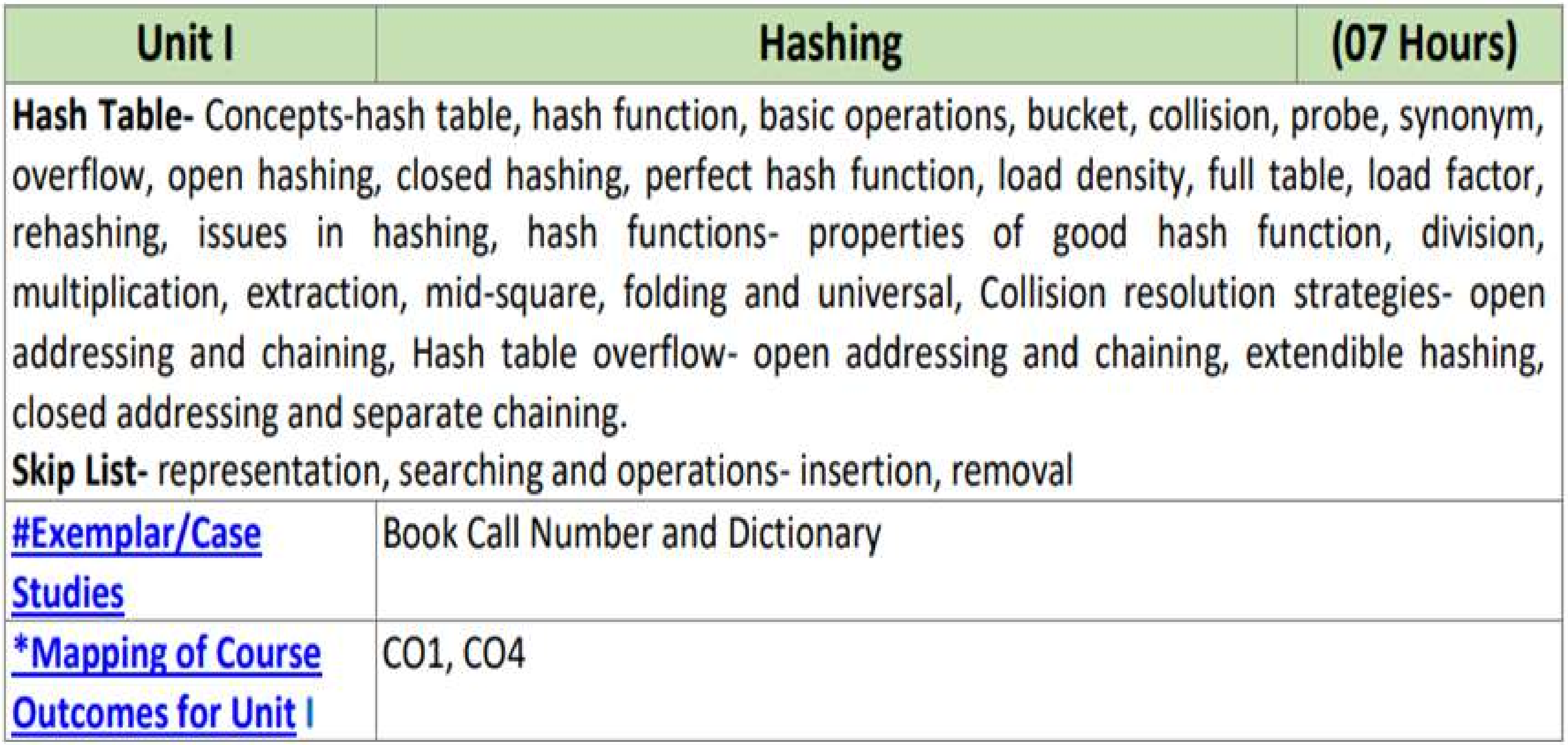
ONE DAY ONLINE SYLLABUS IMPLEMENTATION FACULTY DEVELOPMENT PROGRAM
ON
Data Structures and Algorithms

UNIT 1
HASHING

Mrs. Shalaka P. Deore
M.E.S. College of Engineering, Pune.

Unit 1 Hashing: Syllabus



Unit I	**Hashing**	**(07 Hours)**
Hash Table- Concepts-hash table, hash function, basic operations, bucket, collision, probe, synonym, overflow, open hashing, closed hashing, perfect hash function, load density, full table, load factor, rehashing, issues in hashing, hash functions- properties of good hash function, division, multiplication, extraction, mid-square, folding and universal, Collision resolution strategies- open addressing and chaining, Hash table overflow- open addressing and chaining, extendible hashing, closed addressing and separate chaining. **Skip List-** representation, searching and operations- insertion, removal		
#Exemplar/Case Studies	Book Call Number and Dictionary	
*Mapping of Course Outcomes for Unit I	CO1, CO4	

Unit 1 Hashing: objective and outcome

OBJECTIVE:

- To understand advanced data structures like Hash tables, Skip list to solve complex problems in various domains.
- To study various hash functions.
- To understand various collision resolution techniques.

OUTCOME:

- **Identify and articulate** the complexity goals and benefits of a good hashing scheme for real-world applications.
- **Analyze** the algorithmic solutions for resource requirements and optimization

Need of hashing

Suppose we want to design a system for storing students records and we want to perform following operations efficiently:

- Insert, Search and Delete operations on the basis of student Id.

```
class Studentinfo
{
   long Id;          // Unique Student Id
   String name;      // Student name
   String class;   // Student class
}
```

Possible data structure options and their respective time complexity

- A array implementation would take **O(log n)**time if binary search is used.
- A linked list implementation would take **O(n)** time.

Is there an alternative to get O(1) access time ?

Hashing

Hashing is the process of indexing and retrieving element in a data structure to provide faster way of finding the element using the hash key.

(With hashing we get O(1) search time on average (under reasonable assumptions) and O(n) in worst case.)

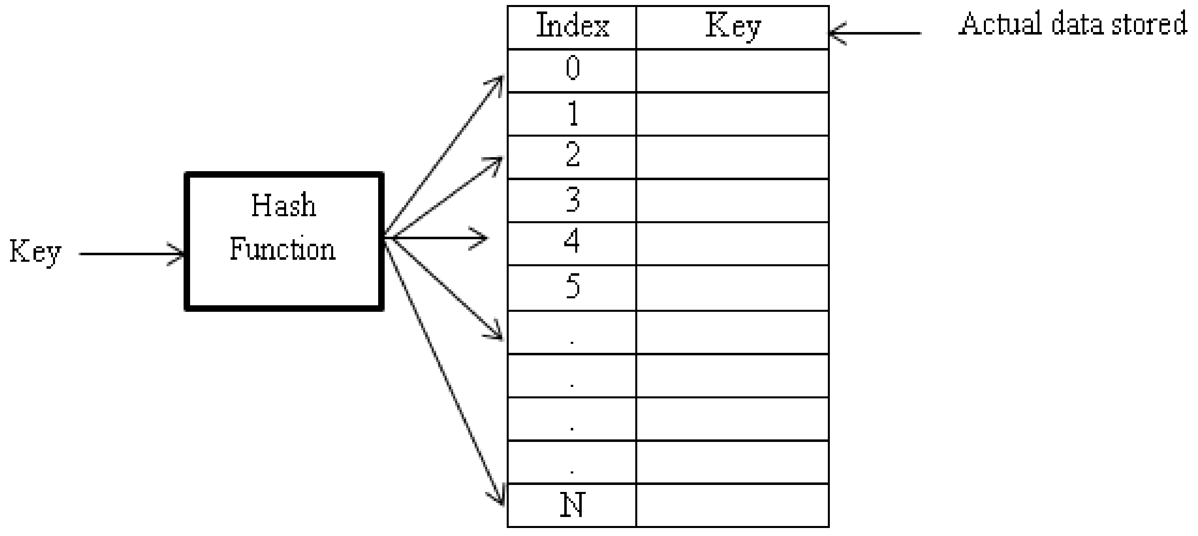
Duration : 1 Hour

Reference :

1. "Fundamentals of Data Structures in C++" by Horowitz, Sahani, Dinesh Mehata
2. https://www.geeksforgeeks.org/hashing-data-structure/#basicHashing

Basic concepts : hash table

- Hash table is a data structure used for storing and retrieving data quickly.
- All data is inserted into hash table based on hash key value. It is used to map the data with the index in the hash table.



Hash Table

Other basic concepts of hashing

- **Buckets**: A bucket in a hash file is unit of storage (typically a disk block) that can hold one or more records.The hash table consists of b buckets and each bucket consists of s slots. (Usually s = 1)

- **Collision**: Collision is situation in which hash function returns the same address for more than one record.

- **Probe:** Alternative list of location produces after collision occurs.

- **Synonym**: The set of keys that hash to the same location are called as synonyms.

Other basic concepts of hashing

- **Overflow:** When hash table becomes full and new record needs to be inserted then it is called overflow. An overflow occurs when we hash a new identifier into a full bucket

- **Perfect hash function**: It is a function that maps distinct key elements into hash table with no collision

- **Load factor or Load density of hash table:**

  lo = n/m

  n = no. of elements stored in the table

  m = size of the table

Hash Functions

- A **good hash function** should:
  - *Minimize* collisions.
  - Be *easy* and *quick* to compute.
  - Distribute key values *evenly* in the hash table.
  - Use *all the information* provided in the key.

Duration : 1 Hour
Reference :
1. “Fundamentals of Data Structures in C++” by Horowitz, Sahani, Dinesh Mehat
2. https://www.tutorialspoint.com/Hash-Functions-and-Hash-Tables

Division Method

Idea:

- Computes hash value from key using the % operator.
- Map a key k into one of the m slots by taking the remainder of k divided by m

$$h(k) = k \bmod m$$

Example: k=1276, n=10,

$$h(1276) = 1276 \bmod 10 = 6$$

- **Advantage**: fast, requires only one operation
- **Disadvantage**:
  - Certain values of m are not good choice, e.g.,
    - power of 2 : Table size that is a power of 2 like 32 and 1024 should be avoided, for it leads to more collisions.

Multiplication Method

Idea:

- Multiply key k by a constant A, where $0 < A < 1$
- Extract the fractional part of kA and multiply the fractional part by m
- Take the floor of the result

$$h(k) = \lfloor m\,(k\,A \bmod 1) \rfloor$$

fractional part of $kA = kA - \lfloor kA \rfloor$

Example: k=123, m=100, A=0.618033

h(123) = 100 (123 * 0.618033 mod 1)

= 100 (76.018059 mod 1)

= 100 (0.018059) = 1

- **Advantage:** Value of m is not critical and it can work with any

Digit Extraction method

Idea

- Selected digits are extracted form the key and used as address

Address = selected digits from key

Example: If six digit employee number is **379245** then select first digit as the index so **379** is the key address.

- **Disadvantages:**
  - May not evenly distribute key values in the hash table

Folding

Idea

- It involves splitting keys into two or more parts and then combining the parts to form the hash addresses.

- To map the key **25936715** to a range between 0 and 9999, we can:
  - · split the number into two as **2593** and **6715** and
  - · add these two to obtain **9308** as the hash value.

- Very useful if we have keys that are very large.

- **Advantage:**
  - · Fast and simple especially with bit patterns
  - · It is ability to transform non-integer keys into integer values.

Mid-square method

Idea

- The key is squared and the middle part of the result taken as the hash value.

- To map the key **3121** into a hash table of size **1000**, we square it $3121^2 = 9740641$ and extract **406** as the hash value.

- **Advantage :** Works well if the keys do not contain a lot of leading or trailing zeros.

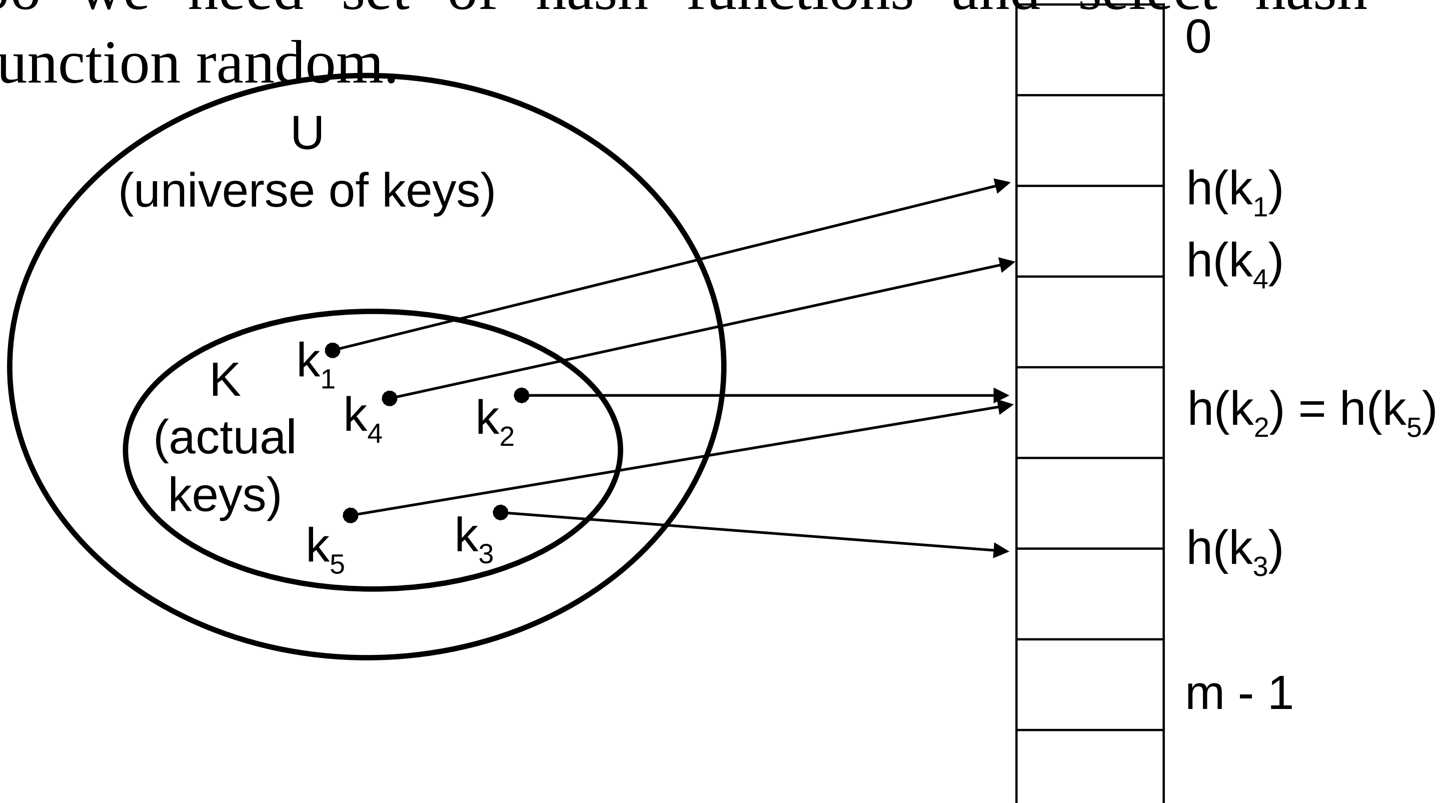
- **Disadvantage:**
  - · Selection of middle part
  - · Non-integer keys have to be pre-processed to

obtain corresponding integer

Universal hashing

Idea

- For any hash function if the table size m is much smaller than universe size U, then for any hash function h, there is some large subset of U that has the same hash value.
- So we need set of hash functions and select hash function random.



Collision handling techniques

- Separate chaining

- Open addressing
  - -Linear Probing
  - -Quadratic Probing
  - -Double Probing

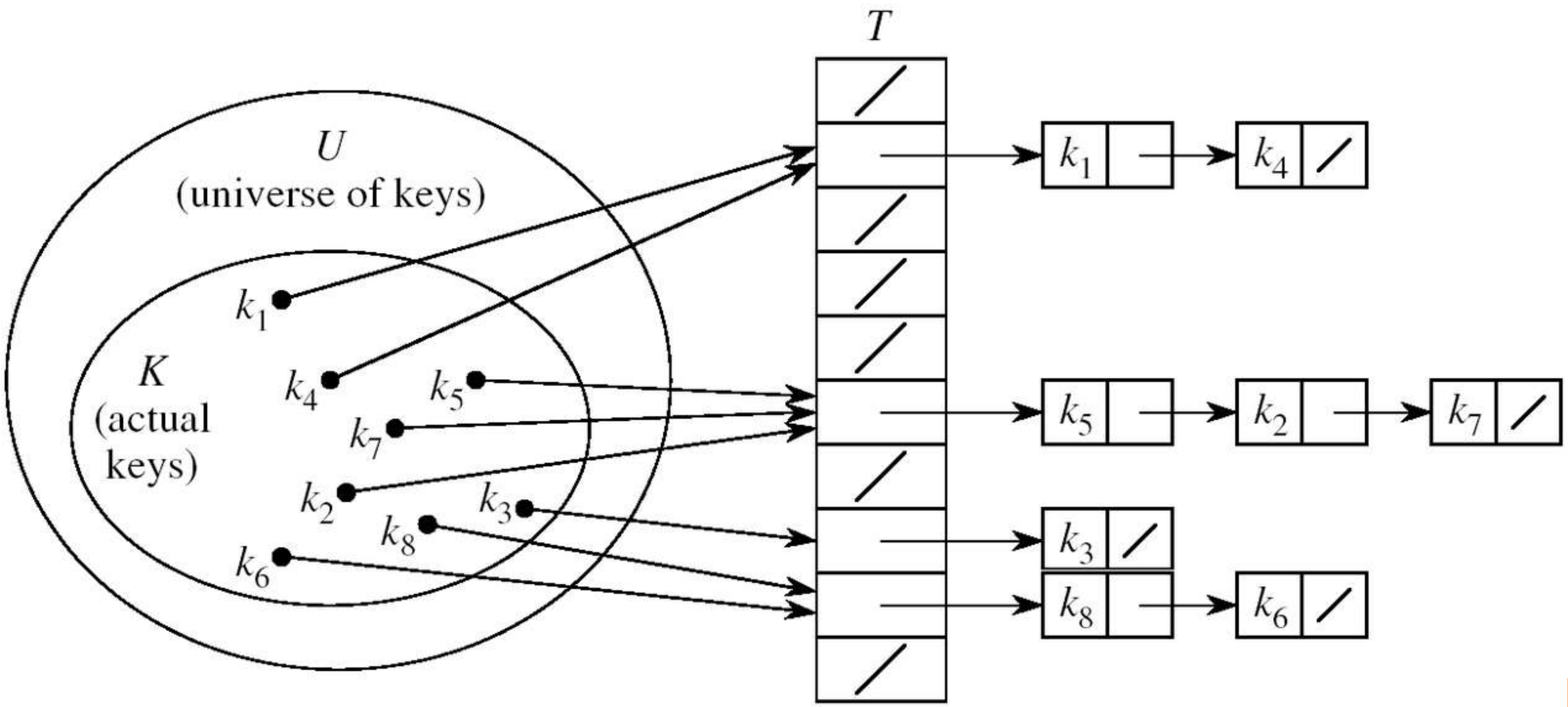
Duration : 3 Hour

Reference :

1. “Fundamentals of Data Structures in C++” by Horowitz, Sahani, Dinesh Mehata
2. https://courses.cs.washington.edu/courses/cse326/00wi/handouts/lecture16/sld025.htm

Separate chaining

- Maintain array of linked list
- Separate list is maintained for all elements mapped to the same value



Separate chaining: pros and cons

PROS:

- Collision resolution is simple
- No problem of load factor can hold more number of elements
- Table size need not be prime number

CONS:

- Implementation of separate data structure (linked list) required for chains
- The main cost of chaining is the extra space required for linked list

Open addressing : Linear probing

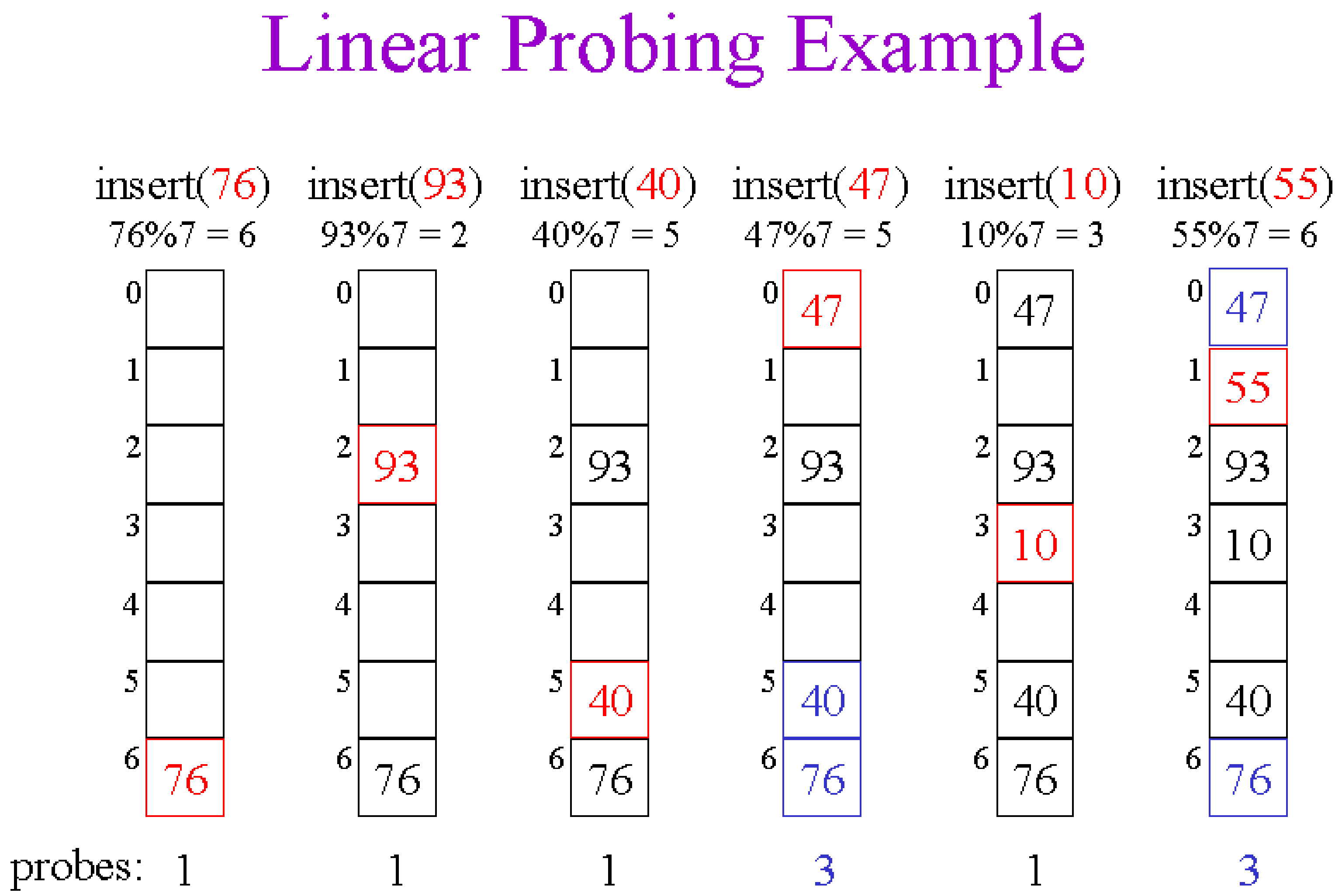
- Table remains a simple array of size m

- On insert(x),

  First compute h(x)= x mod m,
  if the collision occur,
  find another location by sequentially
  searching for the next available slot

- Go to h(x)+1, h(x)+2 etc..

Insert following keys into hash table using linear probing where table size m=7 and h(x)= x mod m, keys={ 76,93,40,47,10,55}



Types of linear probing

Linear probing with chaining (without replacement)

- Excessive collision when occurs it becomes very difficult to maintain indexes of same hash key
- Extra field is added to maintain chain

Example: Let M = 10 , H(X)= X MOD M,
KEYS={0,1,4,71,64,89,11,33}

	Key	Chain
0	0	-1
1	1	2
2	71	3
3	11	-1
4	4	5
5	64	-1
6	33	-1
7		-1
8		-1
9	89	-1

Types of linear probing

Linear probing with chaining (with replacement)

- Problem of misplaced starting location of the chain is handled.
- Extra field is added to maintain chain

Example: Let M = 10 , H(X)= X MOD M,

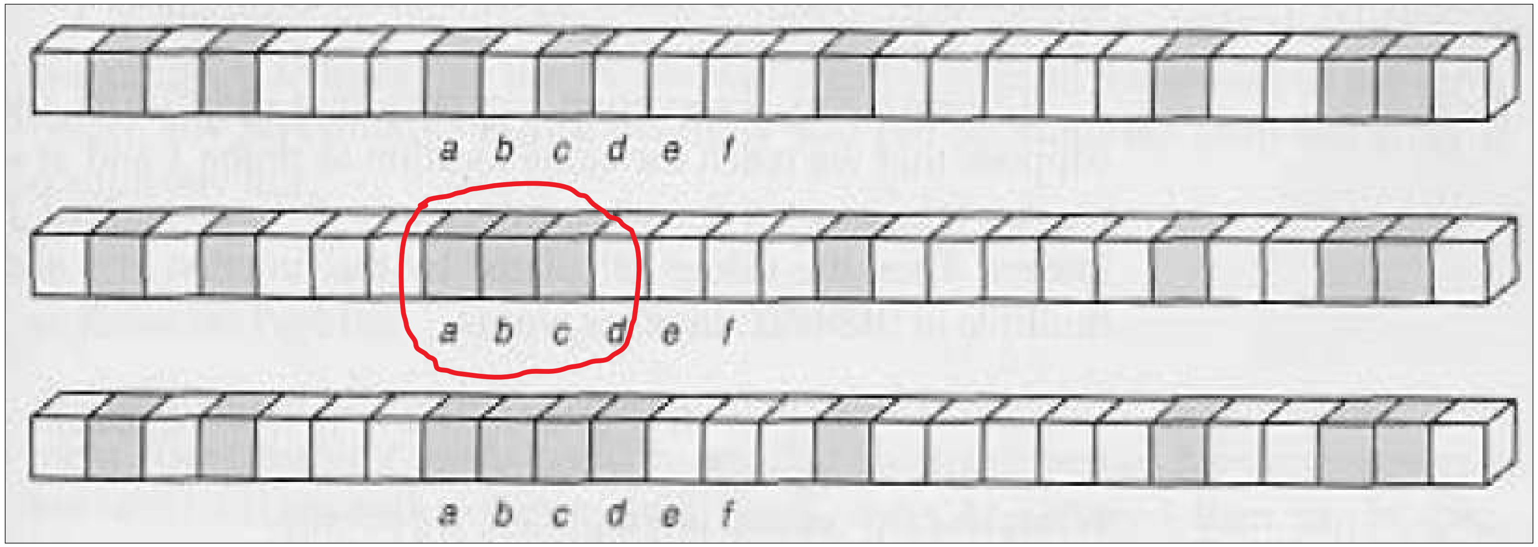
KEYS={0,1,4,71,64,89,11,33}

if add 22 : 1->11->71

	Key	Chain		
0	0	-1	0	-1
1	1	2	1	3
2	71	3	22	-1
3	11	-1	11	7
4	4	5	4	5
5	64	-1	64	-1
6	33	-1	33	-1
7		-1	71	-1
8		-1		
9	89	-1		

Primary clustering problem

- Long chunks of occupied slots are created
- Increases search time



Quadratic hashing

- It is one of the ways to reduce "Primary clustering problem"

- Resolve collisions by examining certain cells away from the original probe point

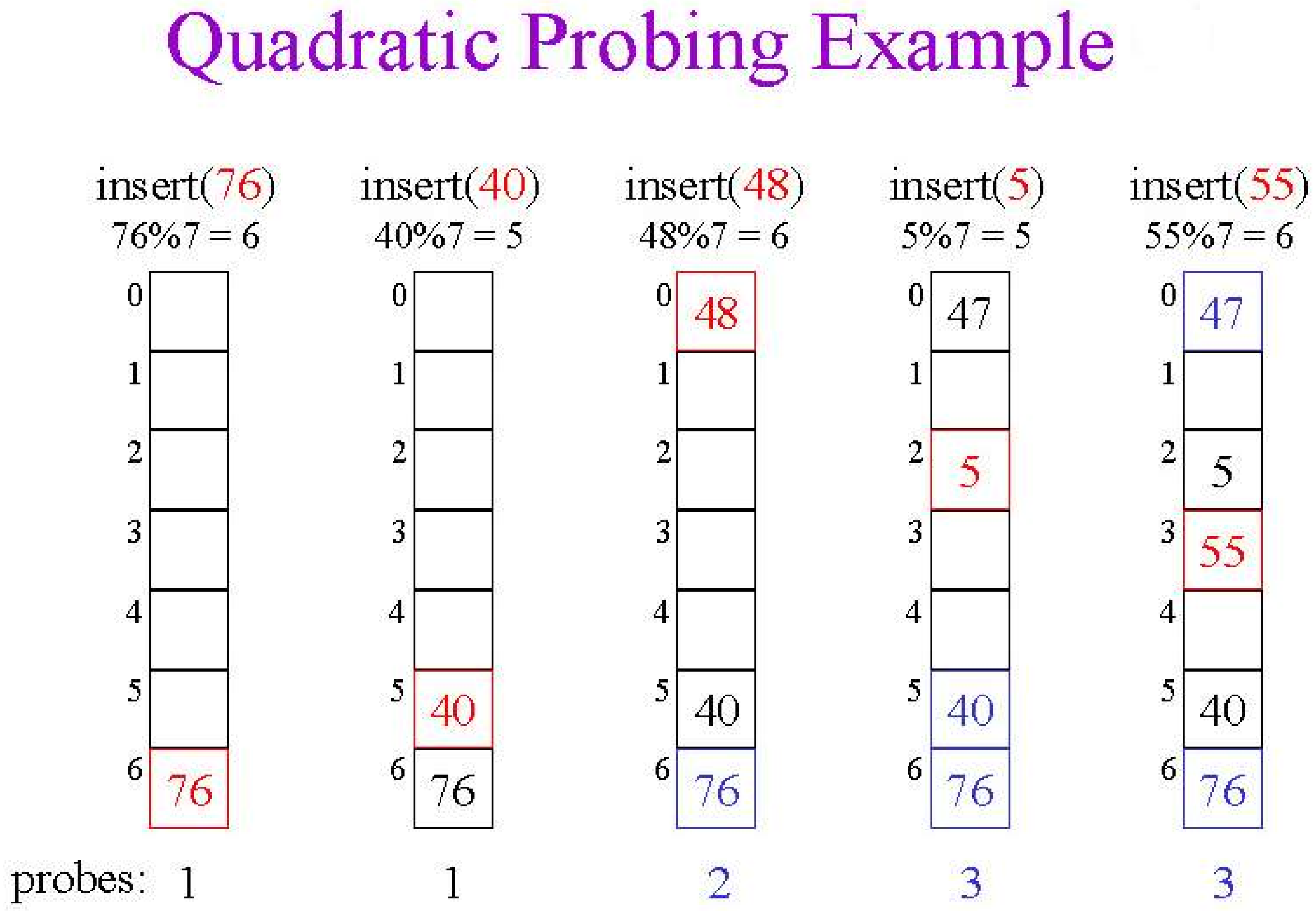
- Collision policy:
  - Start from original hash location i
  - If collision occur search for $i+1^2$, $i+2^2$, $i+3^2$…..

- Hash function :

$$h_i(x) = (h(x) + i^2) \bmod m$$

where i = 0,1,2,3…….

Insert following keys into hash table using quadratic probing where table size m=7 and h(x)= x mod m, keys={ 76,40,48,5,55}



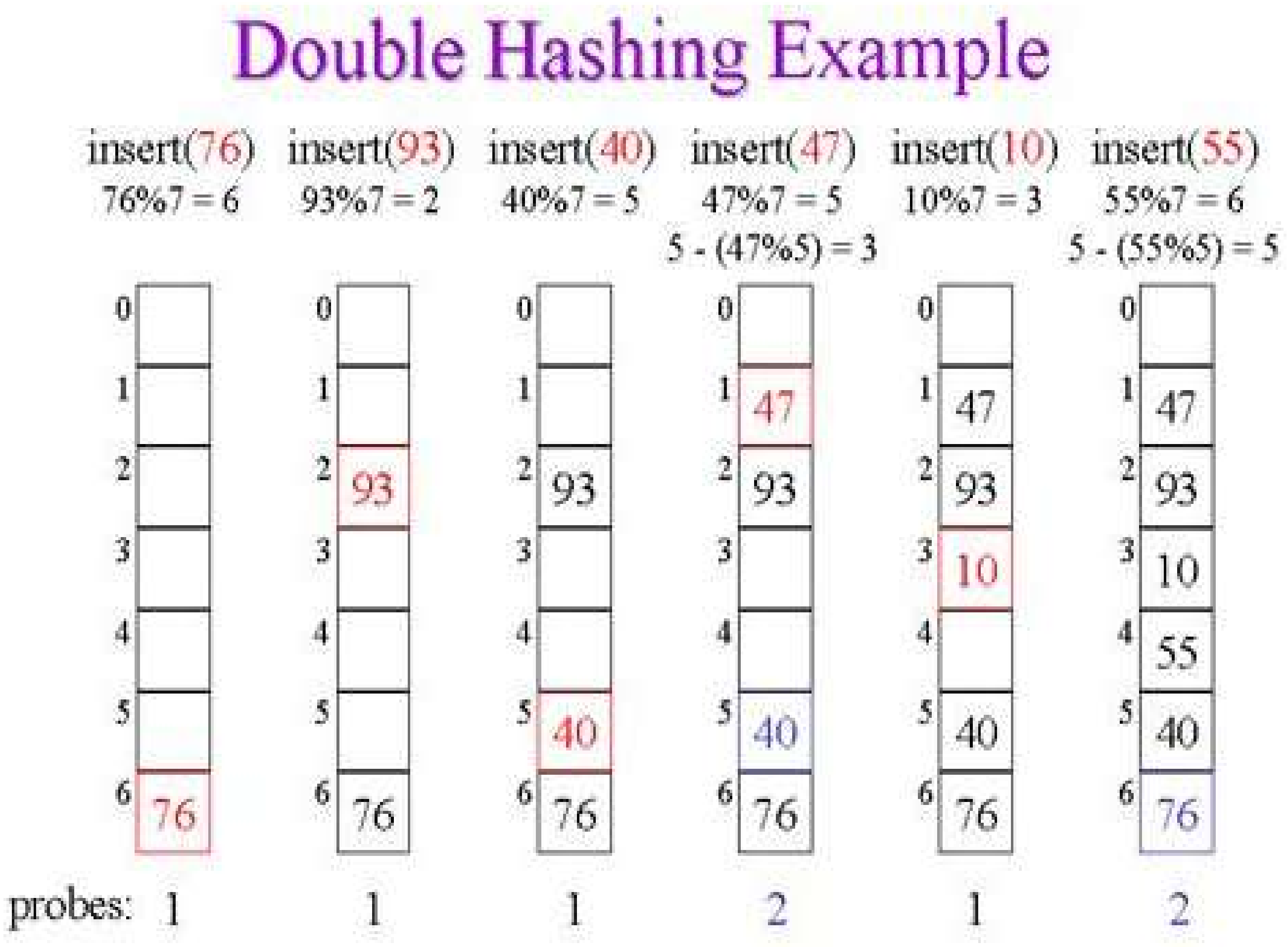
Double hashing

- It reduces clustering in a better way
- Use primary hash function h_1 (k) to determine the first slot
- Use a second hash function h_2 (k) to determine the increment for the probe sequence

$$h(k,i) = (h_1(k) + i\ h_2(k))\ \text{mod}\ m, \quad i=0,1,...$$

- Initial probe: $h_1(k)$
- Second probe is offset by $h_2(k)$ mod m, so on ...
- Advantage: avoids clustering

Insert following keys into hash table using quadratic probing where table size m=7 and h1(x)= x mod m, H2(x)= 5- (x mod 5),keys={76,40,48,5,55}



Open addressing: pros and cons

PROS:

- All data items are stored in the hash table itself no need of separate data structure
- More efficient storage-wise

CONS:

- Dependent on choosing a proper table size
- The keys of the objects to be hashed must be distinct

Hash table overflow

- An overflow occurs when the home bucket for a new pair (key, element) is full.
- We may tackle overflows by searching the hash table in some systematic manner for a bucket that is not full.

Linear probing (linear open addressing).

Quadratic probing.

Random probing.

- Eliminate overflows by allowing each bucket to keep a list of all pairs for which it is home bucket.

Array linear list.

Chain.

- Open addressing is performed to ensure that all elements are stored directly into the hash table

Extendible hashing

- Dynamic hashing method
- Bulky data : Disk accesses increased
- Extendible hashing reduces disk accesses while retrieving the data
- It handle large amount of data
- It required to convert data into binary format
- **Directories:** The directories store addresses of the buckets in pointers. An id is assigned to each directory which may change each time when Directory Expansion takes place.
- **Buckets:** The buckets are used to hash the actual data.

Duration : 1 Hour
Reference:

- https://www.geeksforgeeks.org/extendible-hashing-dynamic-approach-to-dbms/

Extendible hashing

- **Global depth:** It is associated with the directories. They denote the number of bits which are used by the hash function to categorize the keys.

  Global depth = Number of bits in directory id.
- **Local depth:** Local depth is associated with the buckets. Local depth in accordance with the global depth is used to decide the action that to be performed in case an overflow occurs. Local depth is always less than or equal to the Global depth.
- **Bucket splitting:** When the number of elements in a bucket exceeds a particular size, then the bucket is split into two parts.
- **Directory expansion:** Directory expansion takes place when a bucket overflows.

Example:

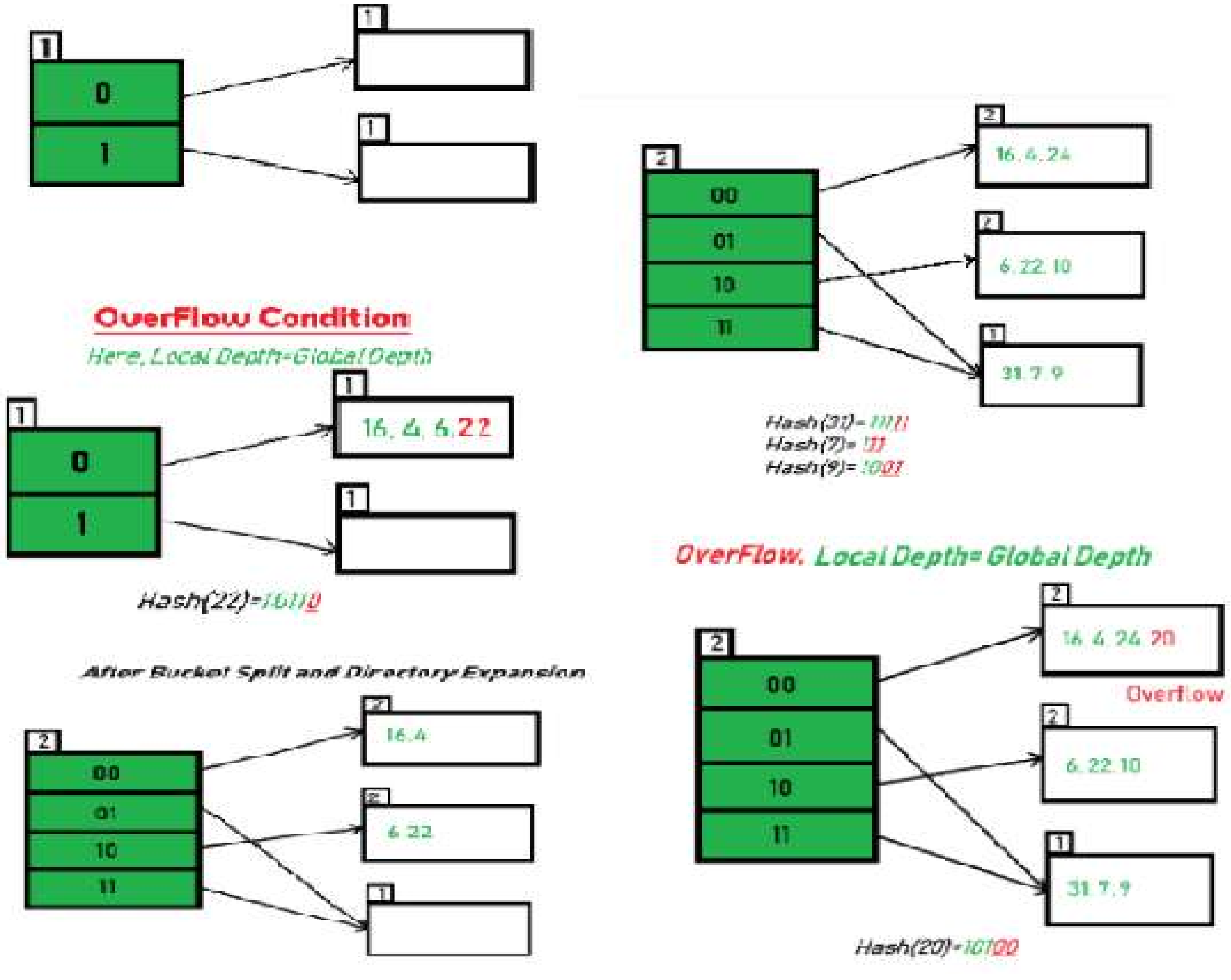
Elements: **16,4,6,22,24,10,31,7,9,20,26.**

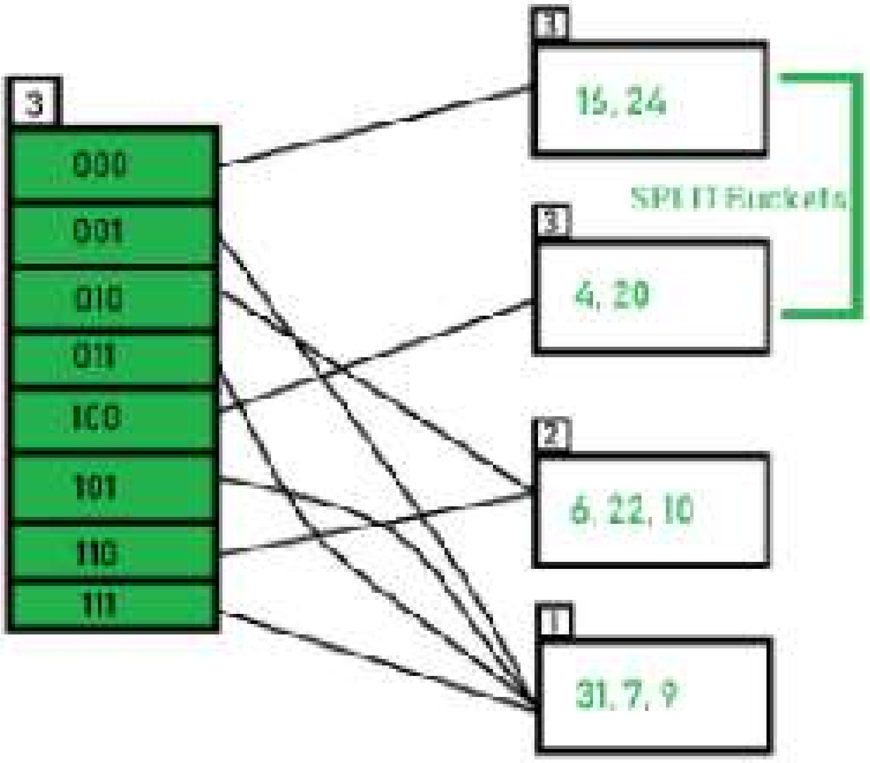
Bucket Size: 3 (Assume)

- **Hash Function:** Suppose the global depth is X. Then the Hash Function returns X LSBs.

  {16- 10000, 4- 00100, 6- 00110, 22- 10110, 24- 11000,
  10- 01010,31- 11111,7- 00111,9- 01001,20- 10100 }

{16- 10000, 4- 00100, 6- 00110, 22- 10110, 24- 11000, 10- 01010,31- 11111,7- 00111,9- 01001,20- 10100 }





3
000
001
010
011
100
101
110
111
3
16, 24
SPLIT Buckets
3
4, 20
2
6, 22, 10
1
31, 7, 9

Some Applications of Hash Table

- **Database systems**: Specifically, those that require efficient random access. Generally, database systems try to optimize between two types of access methods: sequential and random. Hash tables are an important part of efficient random access because they provide a way to locate data in a constant amount of time.

- **Data dictionaries**: Data structures that support adding, deleting, and searching for data. Although the operations of a hash table and a data dictionary are similar, other data structures may be used to implement data dictionaries. Using a hash table is particularly efficient.

- **Symbol tables**: The tables used by compilers to maintain information about symbols from a program. Compilers access information about symbols frequently. Therefore, it is important that symbol tables be implemented very efficiently.

Some Applications of Hash Table

- **Network processing algorithms**: Hash tables are fundamental components of several network processing algorithms and applications, including route lookup, packet classification, and network monitoring.

- **File System :** The hashing is used for the linking of the file name to the path of the file. To store the correspondence between the file name and path, and the physical location of that file on the disk, the system uses a map, and that map is usually implemented as a hash table.

- **Password Verification:** Cryptographic hash functions are very commonly used in password verification

- **Pattern Matching:** The hashing is also used to search for patterns in the strings. Rabin-karp algorithm is use hashing for the searching of a pattern in a string The pattern matching is also used to detect plagiarism

Problems for which hash tables are not suitable

- Problems for which data ordering is required.
  - Hash table is an unordered data structure, certain operations like iterating through the keys in order efficiently

- Problems having multidimensional data.

- Problems in which the data does not have unique keys.
  - Open-addressed hash tables cannot be used if the data does not have unique keys. An alternative is use separate-chained hash tables.

Skip list:

Randomized Data Structure

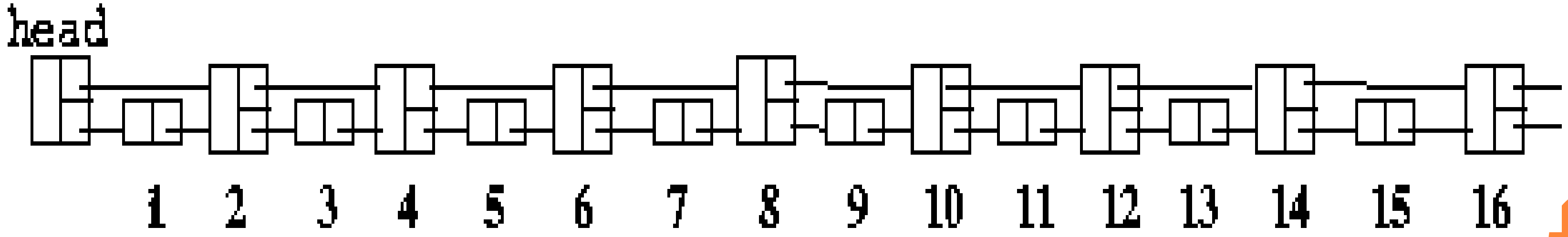
Duration : 2 Hour

Reference :

1. "Advanced Data Structures", by Peter Brass
2. https://courses.cs.washington.edu/courses/cse326/00wi/handouts/lecture16/sld025.htm

Introduction of Skip list

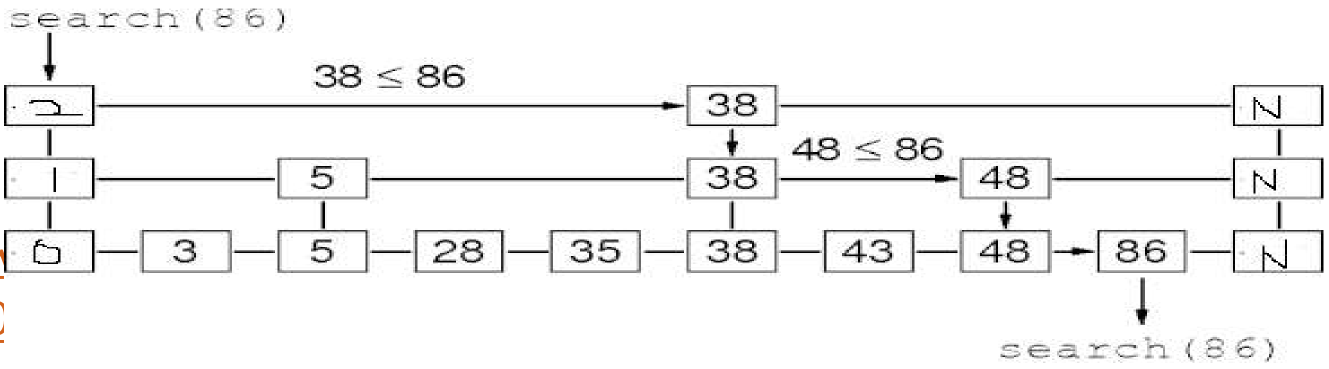
- A skip list is a probabilistic data structure and an extended version of the linked list.
- The skip list is used to store a sorted list of elements or data with a linked list and very useful for concurrently accessing element.
- In one single step, it skips elements of the entire list, hence referred as skip list.
- It allows the user to search, remove, and insert the element very quickly.



There are the following operations:

- Insertion operation: It is used to add a new node
- Deletion operation: It is used to delete a node in a specific situation.
- Search Operation: The search operation is used to search a particular node in a skip list.
- Average case time complexity of all above operations is O(logn)

Head



[https://w
v=UGaC

Applications of the Skip list

- It is used in distributed applications. In distributed systems, the nodes of **skip list** represents the computer systems and pointers represent network connection

Case Study: Book call number and dictionary

- Report writing points:
  - Dictionary concepts with phone book call number

    example
  - Explain Dictionary as ADT
  - Types of Dictionaries :
    - Ordered Dictionaries
    - Unordered Dictionaries
  - Comparison with other data structures

References

- Horowitz, Sahani, Dinesh Mehata, “Fundamentals of Data Structures in C++”, Galgotia Publisher, ISBN: 8175152788, 9788175152786.
- Peter Brass, “Advanced Data Structures”, Cambridge University Press, ISBN: 978-1-107- 43982-5
- https://www.geeksforgeeks.org/hashing-data-structure/#basicHashing
- https://www.tutorialspoint.com/Hash-Functions-and-Hash-Tables
- https://courses.cs.washington.edu/courses/cse326/00wi/handouts/lecture16/sld025.htm
- https://www.geeksforgeeks.org/extendible-hashing-dynamic-approach-to-dbms/
- https://iq.opengenus.org/skip-list/
- https://www.youtube.com/watch?v=UGaOXaXAM5M&ab_channel=ShusenWang

Thank you

shalaka.deore@mescoepune.org