SNS COLLEGE OF TECHNOLOGY

Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 23ITT201 DATA STRUCTURES

II YEAR/ III SEMESTER

UNIT – V SEARCHING,SORTING AND HASHING

Topic: Hashing

Hashing

- What is Hashing , Hash Function and Hash Table
- Collision
- Collision Resolution Techniques
- Separate Chaining
- Open Addressing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing
- Rehashing
- Extendible Hashing

Hashing

In data structures,

Hashing is a well-known technique to search any particular element among several elements.

It minimizes the number of comparisons while performing the search.

Advantage-

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).

Hashing

Two key terms in the concept of Hashing : Hash Table & Hash Function

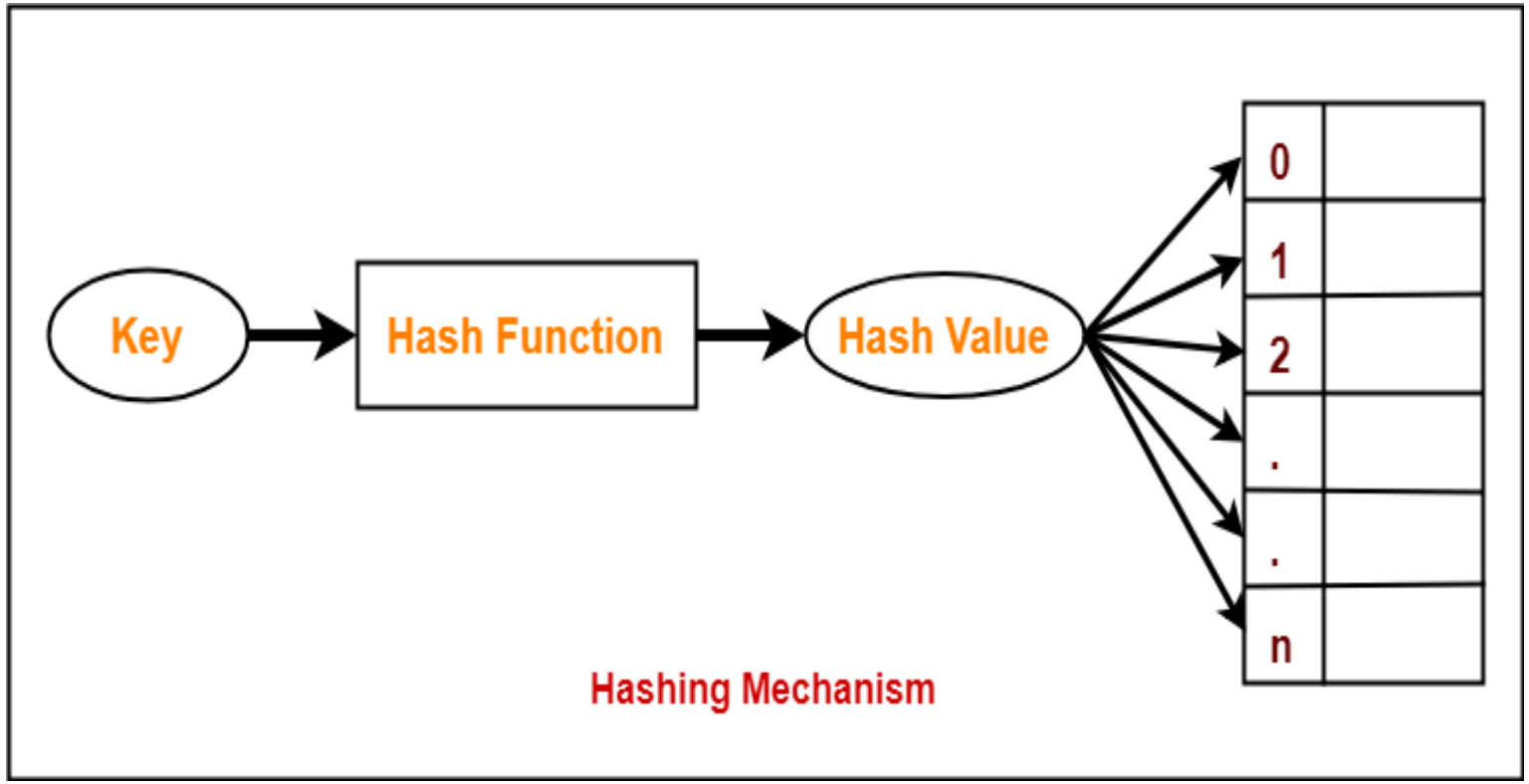
Hashing Mechanism-

In hashing,
An array data structure called as **Hash table** is used to store the data items.
Based on the hash key value, data items are inserted into the hash table.

Hash Key Value-

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.

Hashing



Hashing Mechanism

Hashing

Hash Function-

Hash function is a function that maps any big number or string to a small integer value.

Hash function takes the data item as an input and returns a small integer value as an output.

The small integer value is called as a hash value.

Hash value of the data item is then used as an index for storing it into the hash table.

Hashing

Types of Hash Functions-

There are various types of hash functions available such as-

Mid Square Hash Function

Division Hash Function

Folding Hash Function etc

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

Properties of Hash Function-

It is efficiently computable.

It minimizes the number of collisions.

It distributes the keys uniformly over the table.

Hashing_Collision

Collision in Hashing-

In hashing,

- Hash function is used to compute the hash value for a key.
- Hash value is then used as an index to store the key in the hash table.
- Hash function may return the same hash value for two or more keys.

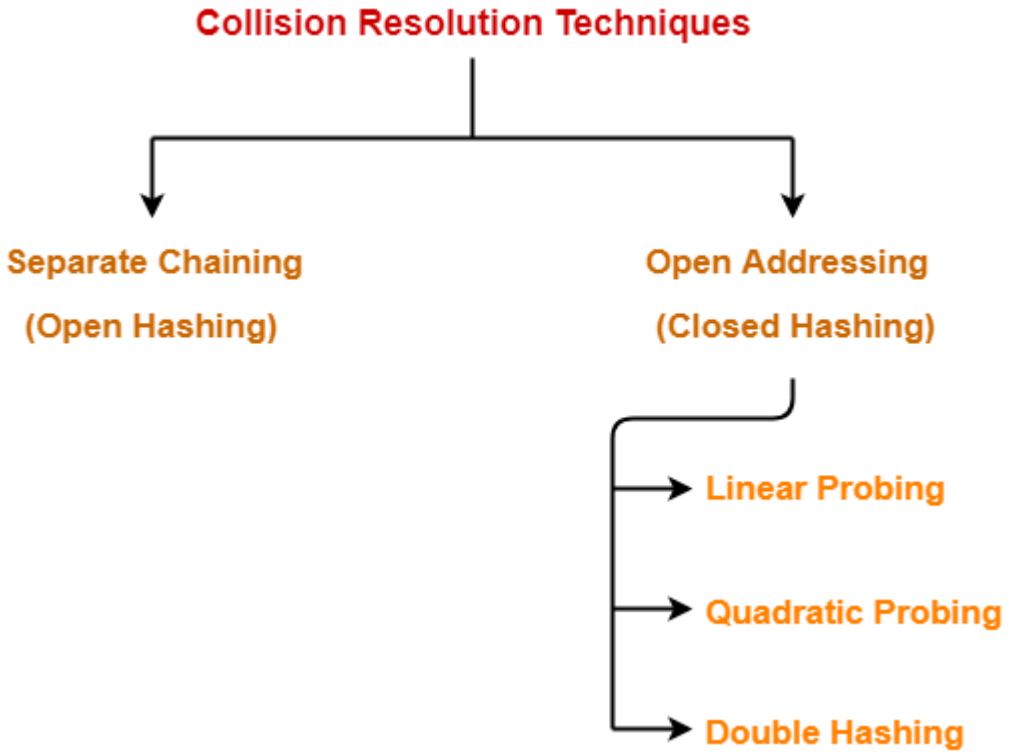
When the hash value of a key maps to an already occupied bucket of the hash table,

it is called as a **Collision**.

Situation: In which the hash function returns the same hash key for more than one record

Hashing

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



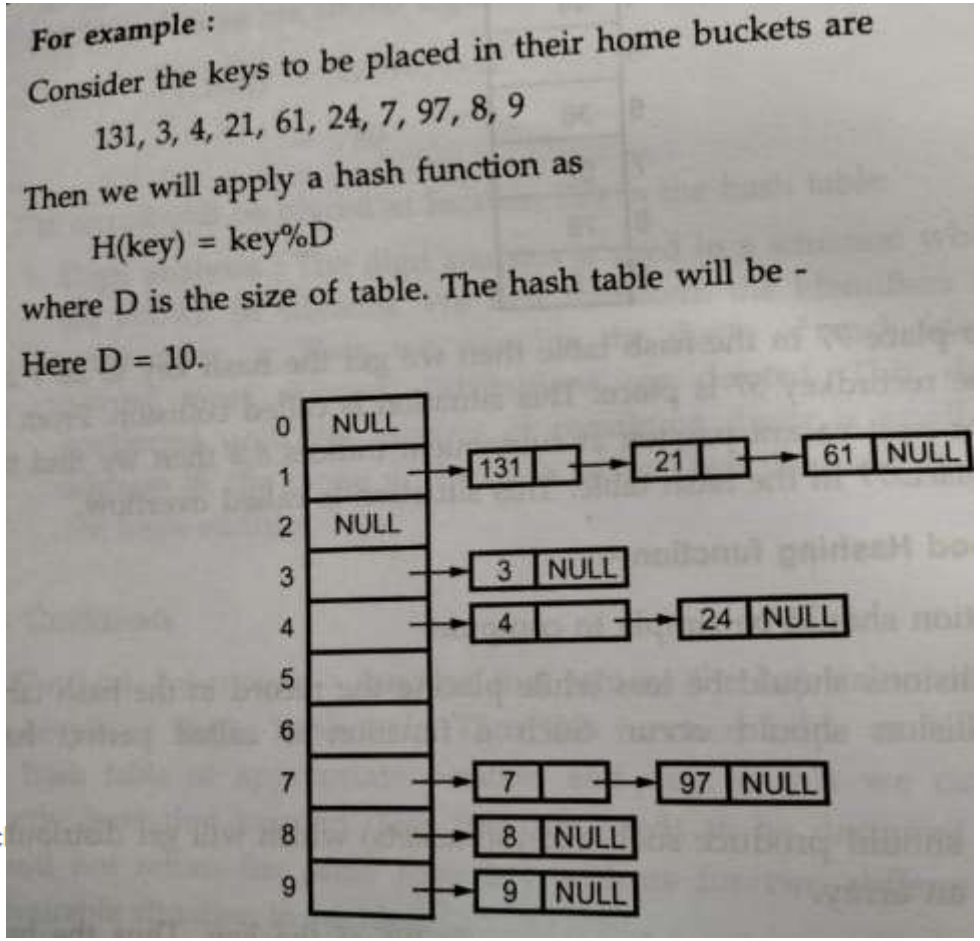
Rehashing
Extendible Hashing

Hashing

Separate Chaining-

To handle the collision,

- This technique creates a linked list to the slot for which collision occurs.
- The new key is then inserted in the linked list.
- These linked lists to the slots appear like chains.
- That is why, this technique is called as **separate chaining**.



Time Complexity-

For Searching-

- In worst case, all the keys might map to the same bucket of the hash table.
- In such a case, all the keys will be present in a single linked list.
- Sequential search will have to be performed on the linked list to perform the searc
- So, time taken for searching in worst case is O(n).

Hashing

For Deletion-

- In worst case, the key might have to be searched first and then deleted.
- In worst case, time taken for searching is O(n).
- So, time taken for deletion in worst case is O(n).

Load Factor (α)-

Load factor (α) is defined as-

$$\text{Load Factor } (\alpha) = \frac{\text{Number of elements present in the hash table}}{\text{Total size of the hash table}}$$

If Load factor (α) = constant, then time complexity of Insert, Search, Delete = Θ(1)

Hashing

Open Addressing-

In open addressing,

- Unlike separate chaining, all the keys are stored inside the hash table.
- No key is stored outside the hash table.

Techniques used for open addressing are-

- Linear Probing
- Quadratic Probing
- Double Hashing

Hashing

Operations in Open Addressing-

Let us discuss how operations are performed in open addressing-

Insert Operation-

- Hash function is used to compute the hash value for a key to be inserted.
- Hash value is then used as an index to store the key in the hash table.

In case of collision,

- Probing is performed until an empty bucket is found.
- Once an empty bucket is found, the key is inserted.
- Probing is performed in accordance with the technique used for open addressing.

Hashing

Search Operation-

To search any particular key,

- Its hash value is obtained using the hash function used.
- Using the hash value, that bucket of the hash table is checked.
- If the required key is found, the key is searched.
- Otherwise, the subsequent buckets are checked until the required key or an empty bucket is found.
- The empty bucket indicates that the key is not present in the hash table.

Delete Operation-

- The key is first searched and then deleted.
- After deleting the key, that particular bucket is marked as “deleted”.

Hashing

Open Addressing Techniques-

Techniques used for open addressing are-

1. Linear Probing-

In linear probing,

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

We keep probing until an empty bucket is found.

Advantage-

It is easy to compute.

Disadvantage-

The main problem with linear probing is Primary clustering.

Many consecutive elements form groups.

Then, it takes time to search an element or to find an empty bucket.

Time Complexity-

Worst time to search an element in linear probing is O (table size).

Hashing

2. Quadratic Probing-

In quadratic probing,

- When collision occurs, we probe for i^2'th bucket in i^{th} iteration.
- We keep probing until an empty bucket is found.

3. Double Hashing-

In double hashing,

- We use another hash function hash2(x) and look for i * hash2(x) bucket in i^{th} iteration.
- It requires more computation time as two hash functions need to be computed.

Hashing

Quadratic Probing

$$H_i(key)=(Hash(key)+i^2)$$

Double Hashing

It is a technique in which a second hash function is applied to the key when a collision occurs. By applying the second hash function we will get the number of position from the point of collision to insert

$$H_1(key)=Key \bmod Table\ size$$

$$H_2(key)=M-(Key \bmod M)$$

Hashing

Collision Handling

DOUBLE HASHING

hash1(k) + j hash2(k)

hash1(k) = k mod 13
hash2(k) = 7 - k mod 7

keys (18, 41, 22, 44)

k	hash1	hash2
18	5	3
41	2	1
22	9	6
44	5	5

j = 0 → until index is empty & I can fill that index.

Hashing-Rehashing

- If the table gets too full, then
  - The running time for the operations will start taking too long
  - Insertions might fail for open addressing hashing with quadratic resolution.
- How to measure the degree of fullness?
  - Define the load factor, λ, of a hash table to be the ratio of the number of elements in the hash table to the table size.
  - If can be $\lambda > 1$?

◆ Rehashing: If the hash table gets too full, then
- build another table that is about twice as big (with an associated new hash function)
- Scan down the entire original hash table, computing the new hash value for each (nondeleted) element and inserting it in the new table.

0	6
1	15
2	
3	24
4	
5	
6	13

Hashing

- For example,
  - Suppose the elements 13, 15, 24, and 6 are inserted into an open addressing hash table of size 7.
  - The hash function is hash(x) = x mod 7
  - Suppose linear probing is used to resolve collisions. f(i) = i.

- If 23 is inserted into the table, the resulting table will be over 70 percent full.
- Because the table is so full, the performance becomes bad. For example, if we want to insert 20, we need to search for almost the whole table.

0	6
1	15
2	23
3	24
4	20
5	
6	13

Hashing

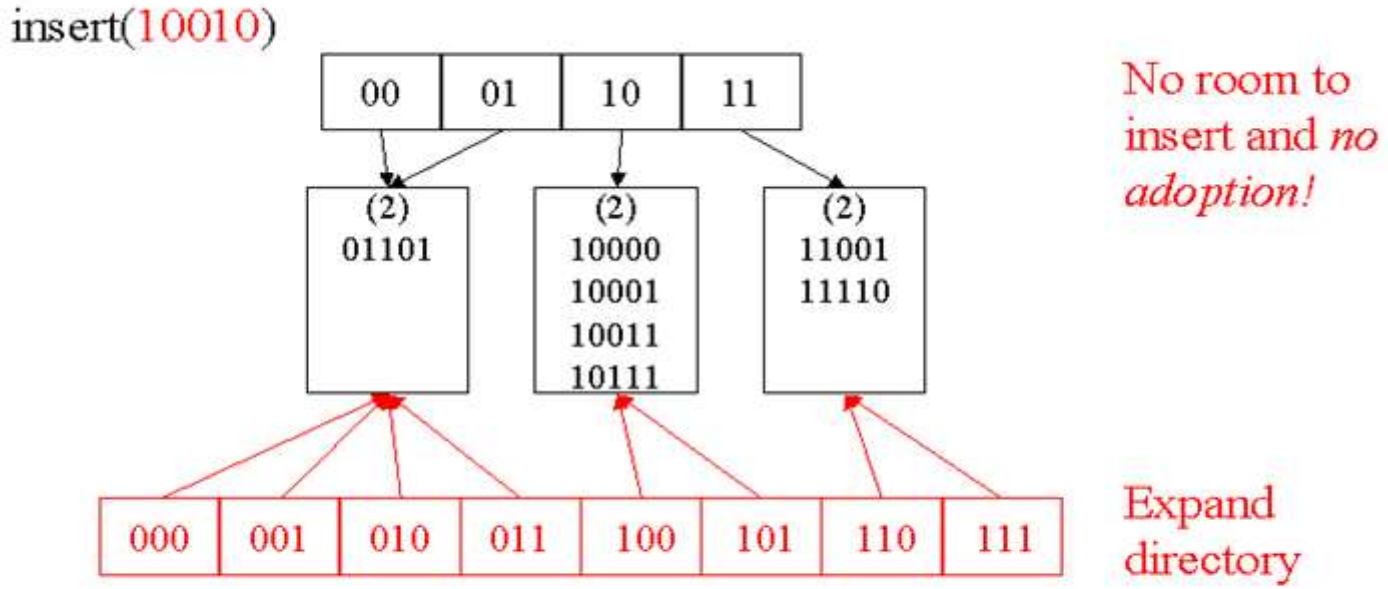
An Example: Rehashing

- Rehashing:
  - Create a new table
  - The size of this table is 17, because this is the first prime that is twice as large as the old table size.
  - The new hash function is then hash(x) = x mod 17.
  - The old table is scanned, and elements 6, 15, 23, 24, and 13 are inserted into the new table.

0	6
1	15
2	23
3	24
4	
5	
6	13

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Extendible Hashing



Now, it's just a normal split.

Hashing

Extendible Hashing

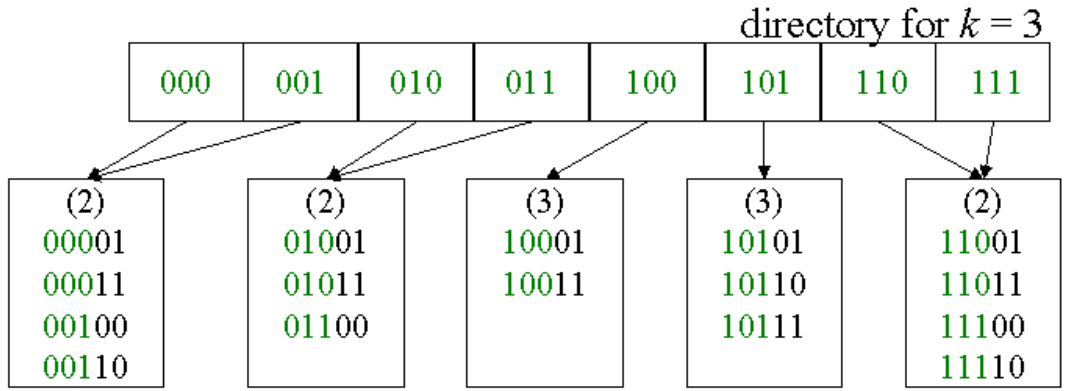
- Hashing technique for huge data sets
  - optimizes to reduce disk accesses
  - each hash bucket fits on one disk block
  - better than B-Trees if order is not important

- Table contains
  - buckets, each fitting in one disk block, with the data
  - a directory that fits in one disk block used to hash to the correct bucket

Hashing

Extendible Hash Table

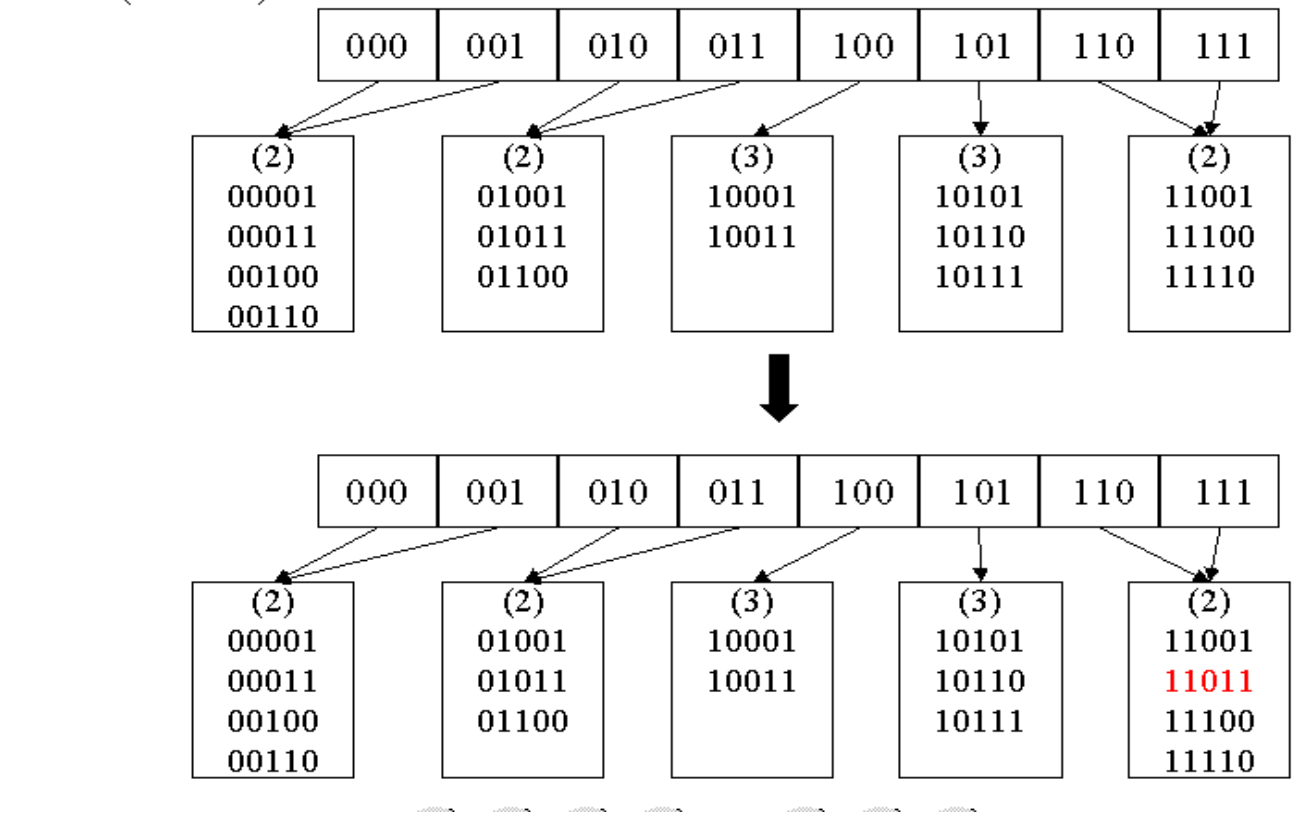
- Directory contains entries labelled by k bits plus a pointer to the bucket with all keys starting with its bits
- Each block contains keys+data matching on the first $j \leq k$ bits



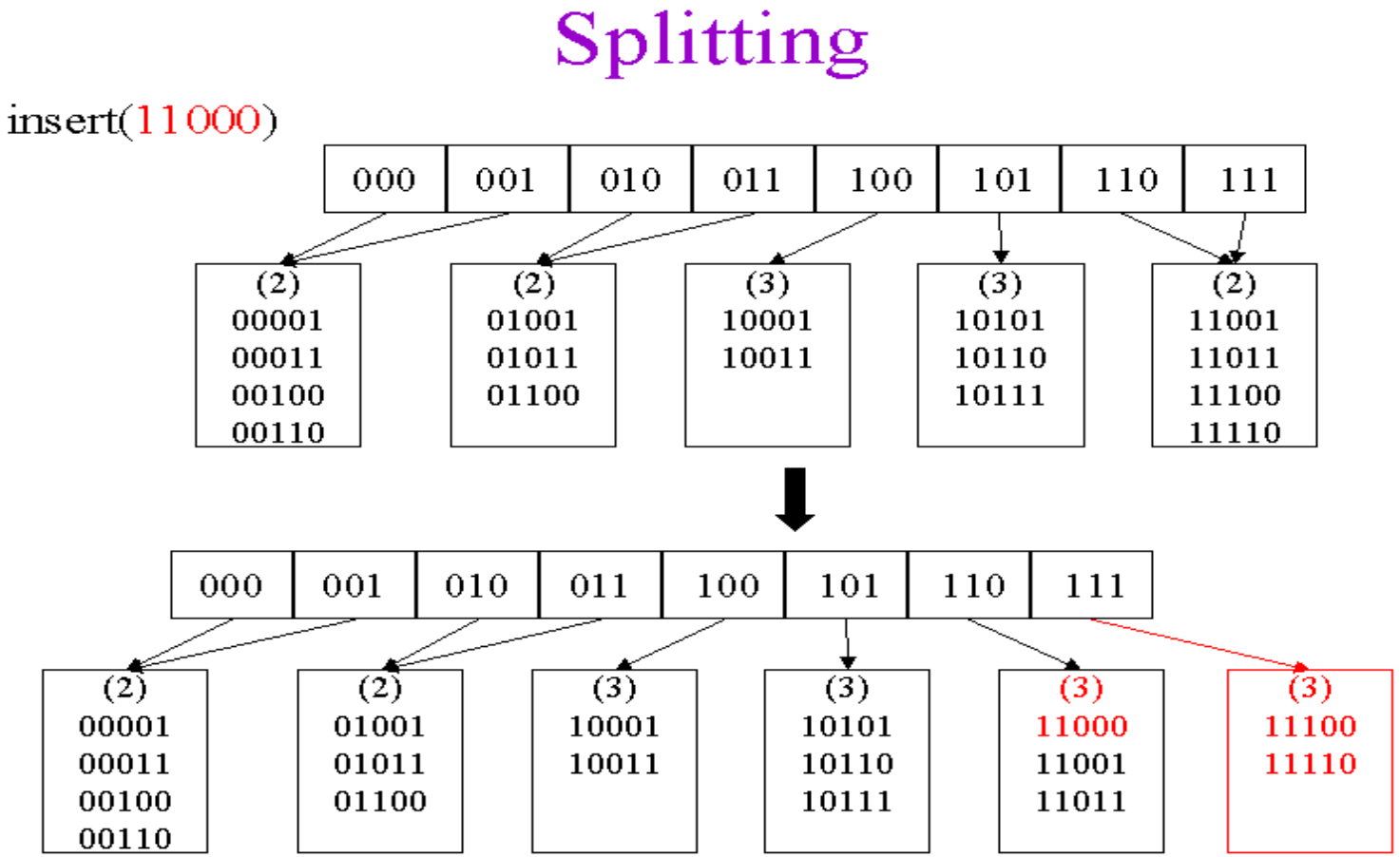
Hashing

Inserting (easy case)

insert(11011)



Hashing


Splitting
insert(11000)
000
001
010
011
100
101
110
111
(2)
00001
00011
00100
00110
(2)
01001
01011
01100
(3)
10001
10011
(3)
10101
10110
10111
(2)
11001
11011
11100
11110
000
001
010
011
100
101
110
111
(2)
00001
00011
00100
00110
(2)
01001
01011
01100
(3)
10001
10011
(3)
10101
10110
10111
(3)
11000
11001
11011
(3)
11100
11110

References

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 8TH Edition, 2008.
2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007
3. Ashok Kamthane, " Data Structures Using C ", Pearson Education, 2nd Edition, 2012.
4. Sahni Horowitz, “Fundamentals of Data Structures in C”Universities Press; Second edition 2008

Thank You