Q1 Create a Hash table and resolve collisions using linear probing with and without replacement : 9, 45, 13, 59, 12, 75, 88, 11, 105, 46
Hash table size = 10 and Hash function = key mod 10.

Steps for with replacement
1. Initialize an empty hash table with a size of 10.
2. Calculate the hash value for each value in the given list using the hash function
3. If the calculated index is empty, insert the value at that index
4. If the calculated index is occupied, perform linear probing by checking the next index, ~~replace the value~~ until the empty index is found.
5. If linear probing encounters an occupied index, replace the value at that index with the new value.
6. Repeat steps 3-5 for all the values in the given list.

Given values : 9, 45, 13, 59, 12, 75, 88, 11, 105, 46

Applying Hash f^n on values and insert in table.

0	59
1	11
2	12
3	13
4	75
5	45
6	46
7	105
8	88
9	9

Steps for without replacement

[1] Start by initializing an empty hash table with a size of 10.

[2] Calculate the hash value for each value in the given list using the hash function

[3] If the calculated index is empty, insert the value at the index

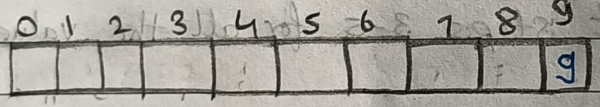
[4] If the calculated index is empty, insert the value at that index.

[5] Repeat step 4 until an empty index is found or until you reach the end of the table.

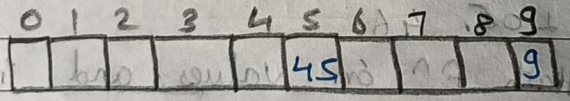
[6] If you reach the end of the table without finding an empty index, it means there is no available space for that value.

[7] Repeat steps 3-6 for all the values in the given list.

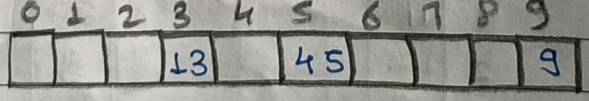
Insert 9 : 9 %. 10 = 9, insert at 9 index



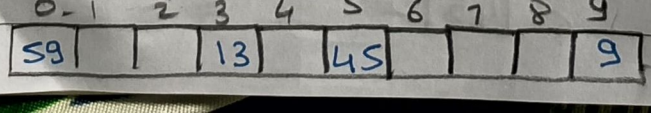
Insert 45: 45 %. 10 = 5, insert at 5 index



Insert 13: 13 %. 10 = 3, insert at 3 index



Insert 59: 59 %. 10 = 9, [collisions]
Probe linearly, next index at 0



Steps for without replacement

[1] Start by initializing an empty hash table with a size of 10.

[2] Calculate the hash value for each value in the given list using the hash function

[3] If the calculated index is empty, insert the value at the index

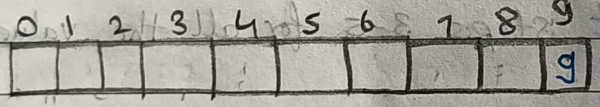
[4] If the calculated index is empty, insert the value at that index.

[5] Repeat step 4 until an empty index is found or until you reach the end of the table.

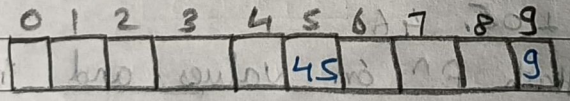
[6] If you reach the end of the table without finding an empty index, it means there is no available space for that value.

[7] Repeat steps 3-6 for all the values in the given list.

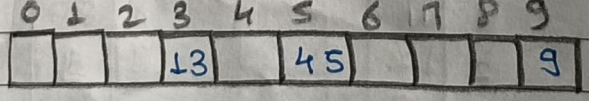
Insert 9 : 9 %. 10 = 9, insert at 9 index



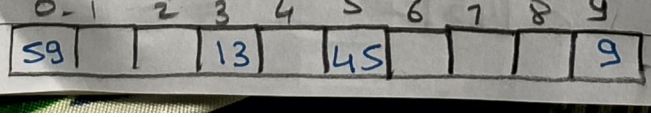
Insert 45: 45 %. 10 = 5, insert at 5 index



Insert 13: 13 %. 10 = 3, insert at 3 index



Insert 59: 59 %. 10 = 9, [collisions]
Probe linearly, next index at 0



M T W T F S S

Page No.: YOUVA

Date:

Insert 12: 12 % 10 = 2 insert at index 2.

0	1	2	3	4	5	6	7	8	9
59		12	13		45				9

Insert 75: 75 % 10 = 5, [collisions]
probe linearly, next index 6

0	1	2	3	4	5	6	7	8	9
59		12	13		45	75			9

Insert 88: 88 % 10 = 8 insert: index 8

0	1	2	3	4	5	6	7	8	9
59		12	13		45	75		88	9

Insert 11: 11 % 10 = 1 insert at index 1

0	1	2	3	4	5	6	7	8	9
59	11	12	13		45	75		88	9

Insert 105: 105 % 10 = 5 [collisions]
probe linearly, next index 7

0	1	2	3	4	5	6	7	8	9
59	11	12	13		45	75	105	88	9

Insert 46: 46 % 10 = 6 [collisions]
probe linearly; next index 4

0	1	2	3	4	5	6	7	8	9
59	11	12	13	46	45	75	105	88	9

Q 2. What is the use of hash Table

ANS:

1. **Database System:** Specifically, those that are required efficient random access. Usually, database systems try to develop between two types of access methods: sequential and random. Hash Table is an integral part of efficient random access because they provide a way to locate data in a constant amount of time.

2. Symbol Tables: The tables utilized by compilers to maintain data about symbols from a program. Compilers access information about symbols frequently. Therefore, it is essential that symbol tables be implemented very efficiently.

3. Data Dictionaries: Data Structure that supports adding, deleting, and searching for data. Although the operation of hash tables and a data dictionary are similar, other Data Structures may be used to implement data dictionaries.

4. Associative Arrays: Associative Arrays consist of data arranged so that nth elements of one array correspond to the nth element of another. Associative Arrays are helpful for indexing a logical grouping of data by several key field

Q. 3 Write a note on characteristics of good hash function.

ANS:

A good hash function is essential for the efficient operation of hash tables and other hashing-based data structures. Here are some characteristics of a good hash function:

1. The hash function should be simple to compute.
2. Number of collisions should be less while placing the record in the hash table. Ideally no collision should occur. Such a function is called perfect hash function.
3. Hash function should produce such keys which will get distributed uniformly over an array.
4. The hash function should depend on every bit of the key. Thus the hash function that simply extracts the portion of a key is not suitable

Q 4. Explain rehashing concept with example.

ANS: The process of re-calculating the hash of already stored data (Key-Value pairs) and to migrate these to another bigger size hash space when threshold is crossed/reached, is called Rehashing
If any stage the hash table becomes nearly full, the running time for the operations of will start taking too much time, insert operation may fail in such situation, the best possible solution is as follows:

1. Create a new hash table double in size.
2. Scan the original hash table, compute new hash value and insert into the new hash table.
3. Free the memory occupied by the original hash table

Load Factor (λ)

- n = number of entries
- N = number of buckets

Load factor $\lambda = \frac{n}{N}$

$\lambda < 1$ i.e $N > n$

when $\lambda > 1$, increase the no of buckets.

Reshashing : Increase the buckets (N') and modify the hash function [change $x \bmod N$ to $x \bmod N'$]

Apply $h'(x)$ [modified hash f^n] to existing elements

N' = Closest prime to $2N$

Example : key (6, 7, 8), $h(x) = x \bmod 3$

$n = 3$, $N = 3$ so $\lambda = 1$

0	1	2
6	7	8

$\therefore$ $N' = 7$ and $h'(x) = x \bmod 7$

$n = 3$, $N = 7$ so, $\lambda = \frac{3}{7} < 1$

0	1	2	3	4	5	6
7	8					6

Q 5. Define the terms:collision,probe,synonym,overflow, perfect hash function, Load density

ANS:

Probe : Each calculation of an address and test for success is known as a probe.

Synonym : The set of keys that has to the same location are called synonyms. For example –In above given hash table computation 25 and 55 are synonyms.

Overflow: When hash table becomes full an new record needs to be inserted then it is called overflow.

Perfect hash function : The perfect hash function is a function that maps distinct-key elements into the hash table with no collisions.

load density:

Load density in hashing refers to the ratio of the number of elements stored in a hash table to the total number of slots available in the table. It is a measure of how full or crowded the hash table is. Load density is typically calculated using the formula:

```
Load Density = Number of Elements / Total Number of Slots
```

Q 6. what is hash function?Enlist characteristics of good hash function. Explain modulo Division and folding method

ANS:

A hash function is a mathematical algorithm that takes an input (or "key") and produces a fixed-size string of characters, known as a hash value or hash code. The primary purpose of a hash function is to map data of arbitrary size to data of a fixed size, typically for use in hash tables, cryptography, data retrieval, and checksum generation.

Characteristics of a good hash function:

1. The hash function should be simple to compute.

2. Number of collisions should be less while placing the record in the hash table. Ideally no collision should occur. Such a function is called perfect hash function.

3. Hash function should produce such keys which will get distributed uniformly over an array.

4. The hash function should depend on every bit of the key. Thus the hash function that simply extracts the portion of a key is not suitable.

Modulo Division Method:

The modulo division method is one of the simplest techniques for generating hash codes. In this method, the key is divided by the size of the hash table, and the remainder (the modulus) is used as the hash value.

Example: Suppose we have a hash table of size 10, and we want to hash a key with the value of 35. Hash Value = Key % Table_Size Hash Value = 35 % 10 = 5

Folding Method:

The folding method involves breaking the key into smaller parts and adding them together to produce the hash value. This is particularly useful when dealing with keys of variable length.

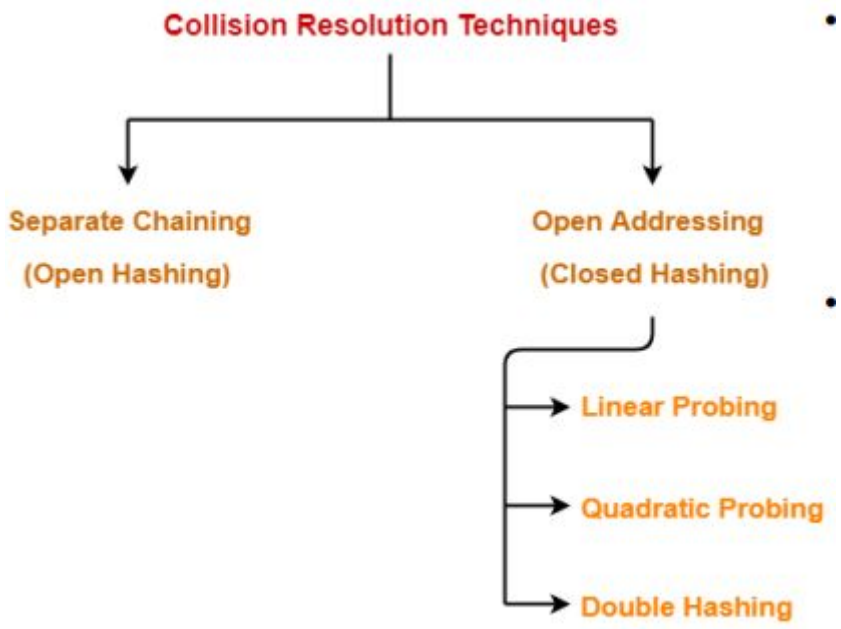
Example: Suppose we have a key with the value of 123456789, and we want to hash it into a hash table with a size of 10.

1. Split the key into chunks of a fixed size (e.g., 2 digits): 12, 34, 56, 78, 9.
2. Add these chunks together: 12 + 34 + 56 + 78 + 9 = 189.
3. Take the modulus of the sum with the size of the hash table: 189 % 10 = 9

Q 7. What is collision?what are the different collision resolution Technique.

ANS: Collisions happen when different inputs produce the same output hash value. Collisions are a common occurrence in hash functions, especially when dealing with a large number of keys and a limited range of hash values.

Collision resolution techniques are used to handle collisions and ensure that each key is stored in the hash table without overwriting or losing any data. Here are some common collision resolution techniques:



Q 8. Given the Inputs :{4371, 1323, 6173, 4199, 4344, 9679, 1989} Hash table size =10 and Hash function = key mod 10. show the results for the following: i)Open addressing hash table using Linear Probing ii)Open addressing hash table using Quadratic Probing iii)open addressing hash table with second hash function h2(X)=7-(x mod 7)

ANS:

Q8 Given the inputs : {4371, 1323, 6173, 4199, 4344, 9679, 1989} Hash table size = 10 and Hash Function = key mod 10. Show the results for the following

i) Open addressing hash table using Linear Probing

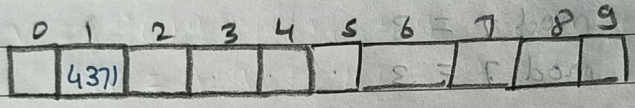
ii) Open addressing HT using Quadratic Probing

iii) Open add... HT with Second hash fn:

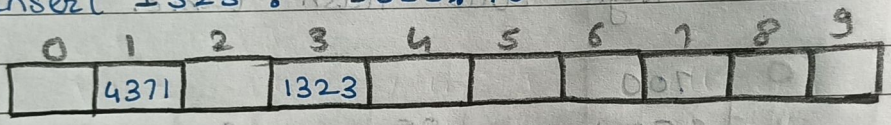
$h2(x) = 7 - (x \bmod 7)$

→ i) Open Addressing HT using Linear Probing.

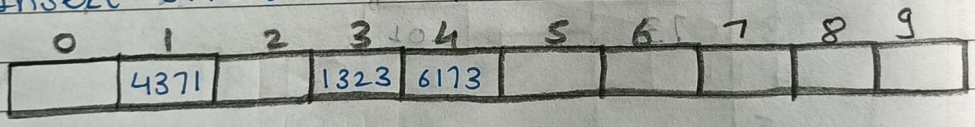
Insert 4371 : 4371 % 10 = 1



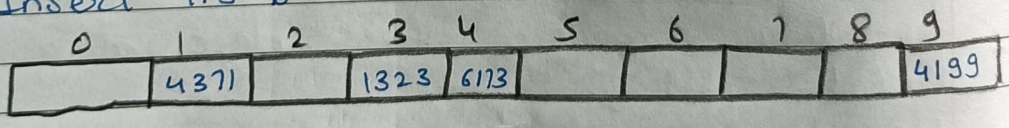
Insert 1323 : 1323 % 10 = 3



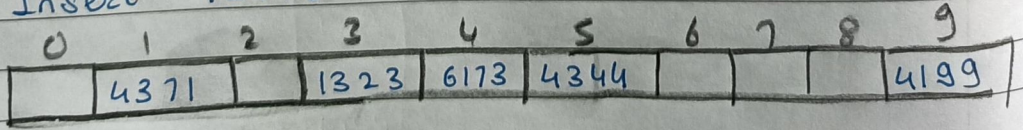
Insert 6173 : 6173 % 10 = 3 [collision]



Insert 4199 : 4199 % 10 = 9



Insert 4344 : 4344 % 10 = 4



Insert 9679, 1989;
both mod fn value is 9 [collision]

0	1	2	3	4	5	6	7	8	9
9679	4371	1989	1323	6173	4344				4199

2) Open Addressing HT using Quadratic Probing

Insert 4371 : 4371 %. 10 = 1 at index 1
Insert 1323 : 1323 %. 10 = 3 at index 3
Insert 6173 : 6173 %. 10 = 3 [collision]
Quadratic Probing, ie at index 7
Insert 4199 : 4199 %. 10 = 9 at index 9
Insert 4344 : 4344 %. 10 = 4 at index 4
Insert 9679 : 9679 %. 10 = 9 [collision]
Quadratic Probing, ie = at index 8
Insert 1989 : 1989 %. 10 = 9 [collision]
Quadratic Probing ie at index 6

Therfore:

0	1	2	3	4	5	6	7	8	9
	4371		1323	4344		1989	6173	9679	4199

3) Open Add

Insert 4371 :
Hash value = 4371 %. 10 = 1 index at 1

Insert 1323 :
Hash value = 1323 %. 10 = 3 index at 3

Page No.: YOUVA
Date:

Insert 6173:
Hash value : 6173 % 10 = 3 [collision]
Secondary HV : 7 − 3 = 4
Probe linearly, next index 4, at index 4.

Insert 4199:
Hash value : 4199 % 10 = 9 at index 9

Insert 4344:
HV = 4344 % 10 = 4 : [collision]
~~probe linearly, because [7-4] = 3 [collision]~~

Insert 9679:
HV = 9679 % 10 = 9 (collision)
Sec^n HV = 7 - 9679 % 7 = 7 3 = 4
Probe linearly, next index,

Insert 1989.
HV = 1989 % 10 = 9 (collision)
Secondary HV = 7 − (1989 % 7) = 4
probe linearly, next index.

0	
1	4371
2	
3	1323
4	6173
5	4344
6	9679
7	1989
8	
9	4199

Q 9. Explain Extendible hashing with suitable example.

ANS: Extendible hashing is a dynamic hashing technique used for organizing and managing large amounts of data efficiently. It's particularly useful in database systems where the size of the data can vary over time.

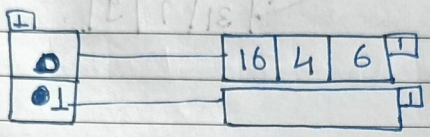
In extendible hashing, data is organized into a directory and a collection of buckets. Each bucket can hold a fixed number of data items or records. Initially, the directory contains a small number of pointers to buckets, and each bucket is associated with a certain number of bits from the hash value of the keys.

Example: 16, 4, 6, 22, 24, 10, 31, 7, 9, 20,

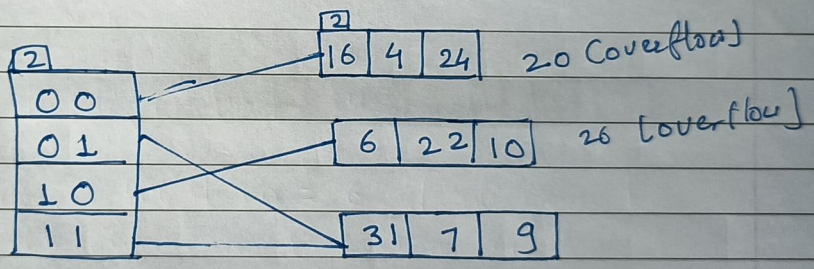
Assum bucket size 3. In binary

16 = 10000 4 = 00100 6 = 00110
22 = 10110 24 = 11000 10 = 01010
31 = 11111 7 = 00111 9 = 01001
20 = 10100 26 = 11010

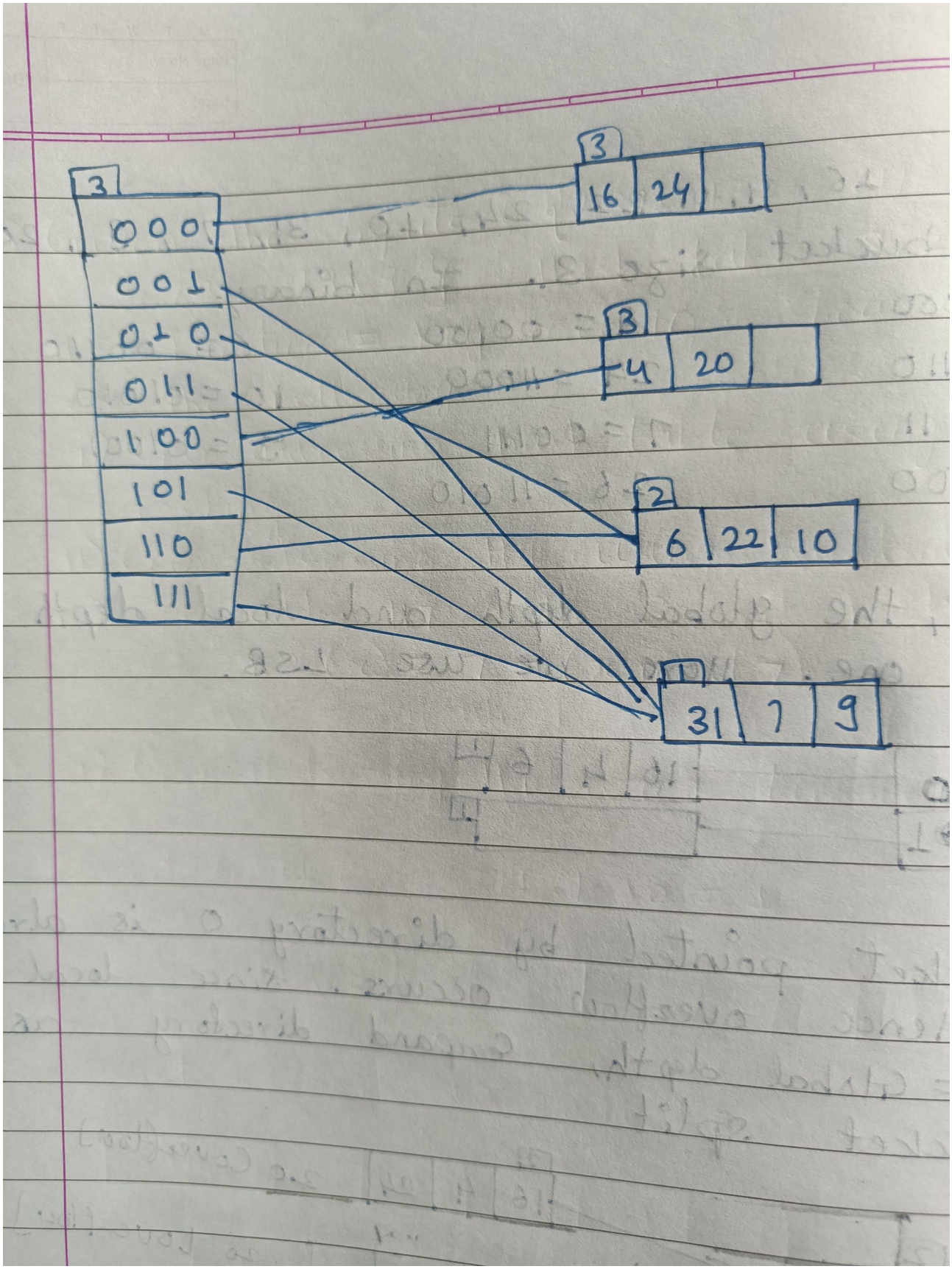
Initially, the global depth and local depth is always one. Here we use LSB.



The bucket pointed by directory 0 is already full. Hence overflow occurs. Since local depth = Global depth, Expand directory as well as bucket split



Since, local depth = global depth
Directory expansion takes place along with bucket spliting.
Elements present in overflowing bucket are rehashed with the new global depth.


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

Q 10. Write a short note on separate chaining

ANS: Separate chaining is a collision resolution technique used in hash tables to handle collisions by maintaining a linked list or other data structure at each slot in the hash table. When a collision occurs, instead of overwriting the existing element, the new element is simply added to the linked list at the corresponding slot. This allows multiple elements with different keys but the same hash value to coexist within the same slot in the hash table.