S.E (Computer Engineering)

DATA STRUCTURE AND ALGORITHM (2019 Pattern)

INSEM MARCH-2024

Time : 1 Hour ***[Max. Marks : 30]***

Instructions to the candidates:

1) ***Solve Que. 1 or 2, Que.3 or 4 .***
2) ***Neat diagrams must be drawn wherever necessary.***
3) ***Assume suitable data if necessary.***
4) ***Figures to the right indicate full marks.*** **Date : 21/03/2024**

DSA INSEM MODEL ANSWER MARCH-2024

Q.1 a) For the given set of values 76, 40, 48, 5, 55. Construct a hash table with size 7 and resolve collision using quadratic probing? 6-M

→ **STEP-1: No collision**

0	
1	
2	
3	
4	
5	
6	76

TableSize = 7
Insert:

76	(76 % 7 = 6)
40	(40 % 7 = 5)
48	(48 % 7 = 6)
5	(5 % 7 = 5)
55	(55 % 7 = 6)

76 % 7 = 6

STEP-2 : No Collision

0	
1	
2	
3	
4	
5	40
6	76

TableSize = 7
Insert:

76	(76 % 7 = 6)
40	(40 % 7 = 5)
48	(48 % 7 = 6)
5	(5 % 7 = 5)
55	(55 % 7 = 6)

40 % 7 = 5

STEP-3: Collision occurs at 48

0	48
1	
2	
3	
4	
5	40
6	76

TableSize = 7
Insert:
76 (76 % 7 = 6)
40 (40 % 7 = 5)
48 (48 % 7 = 6)
5 (5 % 7 = 5)
55 (55 % 7 = 6)

$6 + 1^2 \% 7 = 0$

STEP-4 : Collision occurs

0	48
1	
2	5
3	
4	
5	40
6	76

TableSize = 7
Insert:
76 (76 % 7 = 6)
40 (40 % 7 = 5)
48 (48 % 7 = 6)
5 (5 % 7 = 5)
55 (55 % 7 = 6)

$5 + 1^2 \% 7 = 6$ collision hence $5 + 2^2 \% 7 = 2$

STEP-5:

0	48
1	
2	5
3	55
4	
5	40
6	76

TableSize = 7
Insert:
76 (76 % 7 = 6)
40 (40 % 7 = 5)
48 (48 % 7 = 6)
5 (5 % 7 = 5)
55 (55 % 7 = 6)

$6 + 1^2 \% 7 = 0$ collision hence $6 + 2^2 \% 7 = 3$

All elements are mapped in hash table using Quadratic Probing techniques.

b) Explain about Skip list in hashing. Give applications of Skip List. 4-M

→ A skip list is a probabilistic data structure. The skip list is used to store a sorted list of elements or data with a linked list. It allows the process of the elements or data to view efficiently. In one single step, it skips several elements of the entire list, which is why it is known as a skip list.

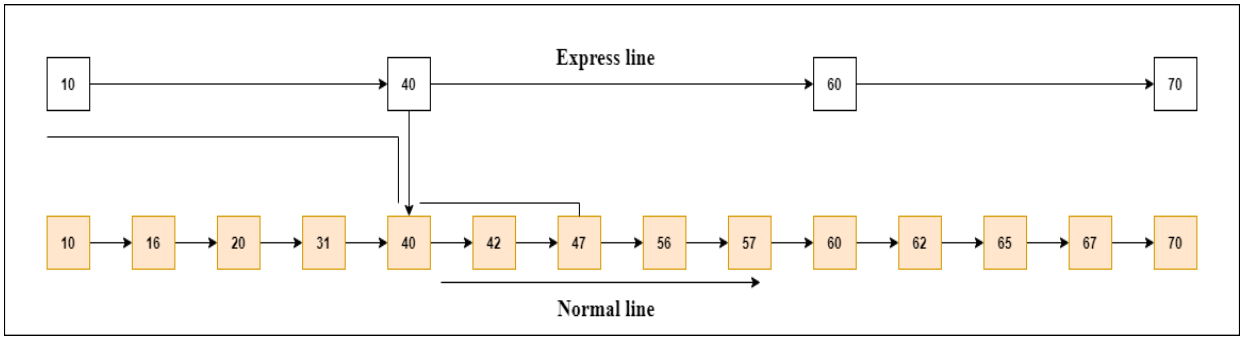
It is built in two layers: The lowest layer and Top layer.

The lowest layer of the skip list is a common sorted linked list, and the top layers of the skip list are like an "express line" where the elements are skipped. **1 - Mark**

Working of the Skip list : **3 - Mark**

The lower layer is a common line that links all nodes, and the top layer is an express line that links only the main nodes, as you can see in the diagram.

Suppose you want to find 47 in this example. You will start the search from the first node of the express line and continue running on the express line until you find a node that is equal a 47 or more than 47.



You can see in the example that 47 does not exist in the express line, so you search for a node of less than 47, which is 40. Now, you go to the normal line with the help of 40, and search the 47, as shown in the diagram.

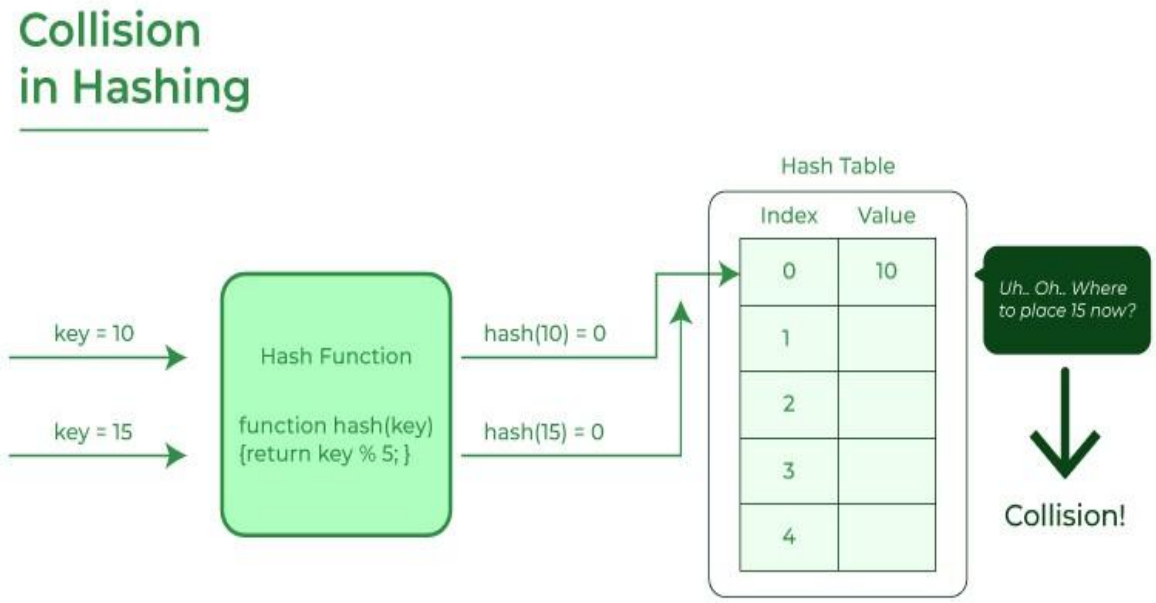
c) What is hash function. What are charactristics of hash function. 5-M

→ **Definition : Hashing** is the process of converting a given key into another smaller value for O(1) retrieval time.

- **A hash table** is an array that stores pointers to data mapping to a given hashed key.

- A **hash function** is any function that can be used to map data of arbitrary size to fixed-size values, though there are some hash functions that support variable . **1 – Mark**

Diagram :



1 - Mark

Characteristics of Hash Function : **3 - Mark**

1) The hash value is fully determined by the data being hashed.

2) The hash function uses all the input data.

3) The hash function "uniformly" distributes the data across the entire set of possible hash values.

4) The hash function generates very different hash values for similar strings.

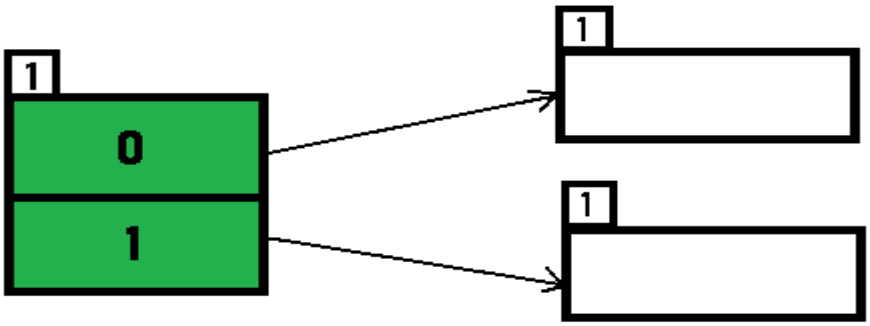
OR

Q.2 a) Prepare Hash table by inserting following elements into hash table using Extendible hashing: 16, 4, 6, 22, 24, 10, 31, 7, 9, 20, 26. Bucket size - 3 **6-M**

→ First, calculate the binary forms of each of the given numbers.

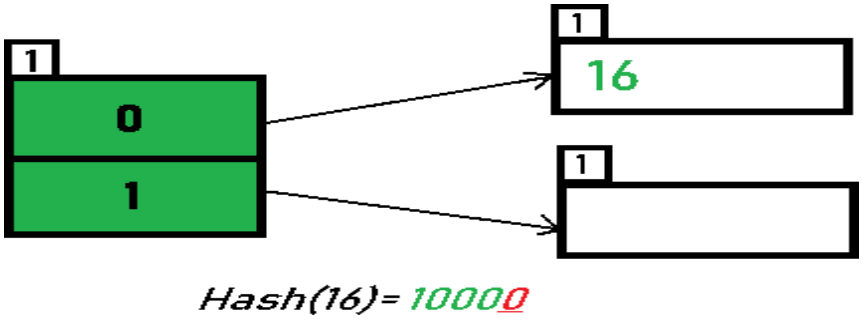
16- 10000
4- 00100
6- 00110
22- 10110
24- 11000
10- 01010
31- 11111
7- 00111
9- 01001
20- 10100
26- 11010

Initially, the global-depth and local-depth is always 1. Thus, the hashing frame looks like this:

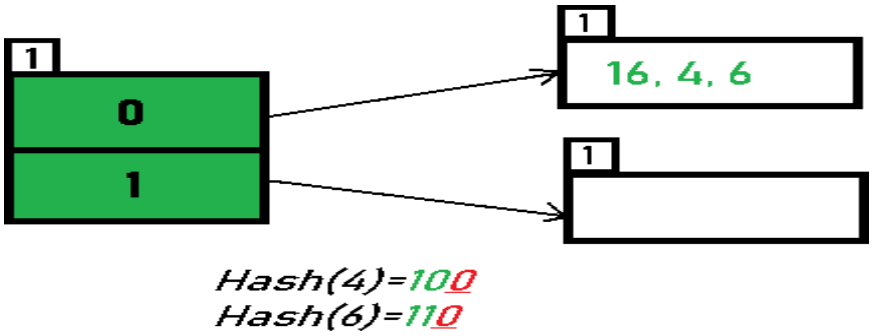


Inserting 16:

The binary format of 16 is 10000 and global-depth is 1. The hash function returns 1 LSB of 1000**0** which is 0. Hence, 16 is mapped to the directory with id=0.

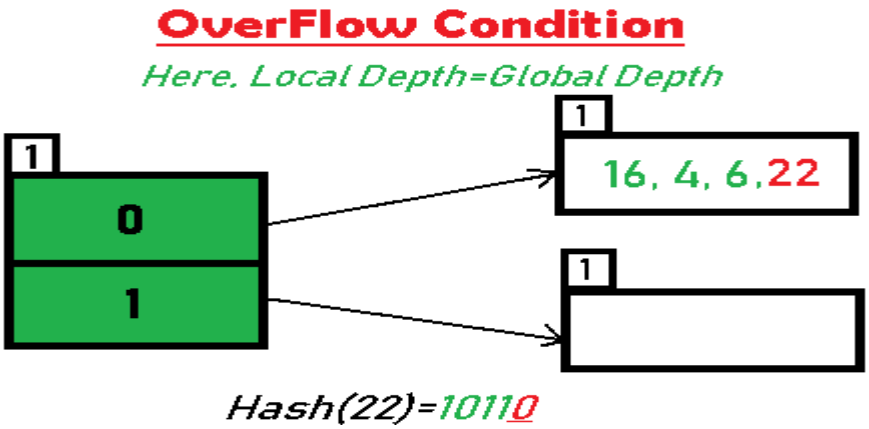


Inserting 4 and 6:
Both 4(10**0**) and 6(11**0**)have 0 in their LSB. Hence, they are hashed as follows:



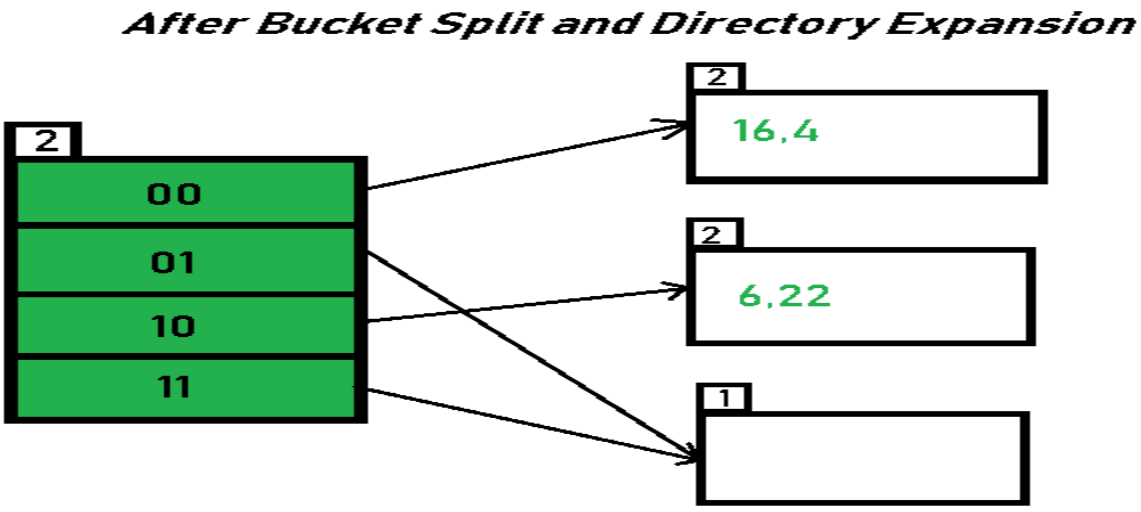
Hash(4)=100
Hash(6)=110

Inserting 22: The binary form of 22 is 1011**0**. Its LSB is 0. The bucket pointed by directory 0 is already full. Hence, Over Flow occurs.



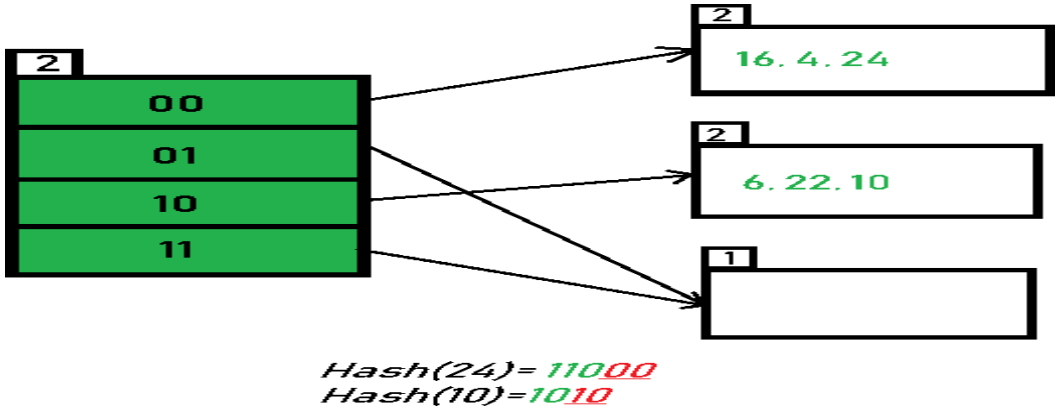
Hash(22)=10110

As directed by **Step 7-Case 1**, Since Local Depth = Global Depth, the bucket splits and directory expansion takes place. Also, rehashing of numbers present in the overflowing bucket takes place after the split. And, since the global depth is incremented by 1, now,the global depth is 2. Hence, 16,4,6,22 are now rehashed w.r.t 2 LSBs.[16(100**00**),4(1**00**),6(1**10**),22(101**10**)]

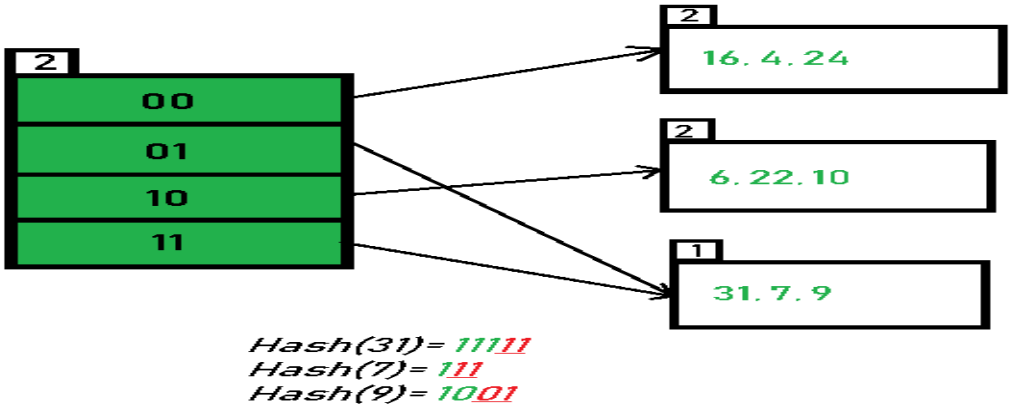


*Notice that the bucket which was underflow has remained untouched. But, since the number of directories has doubled, we now have 2 directories 01 and 11 pointing to the same bucket. This is because the local-depth of the bucket has remained 1. And, any bucket having a local depth less than the global depth is pointed-to by more than one directories.

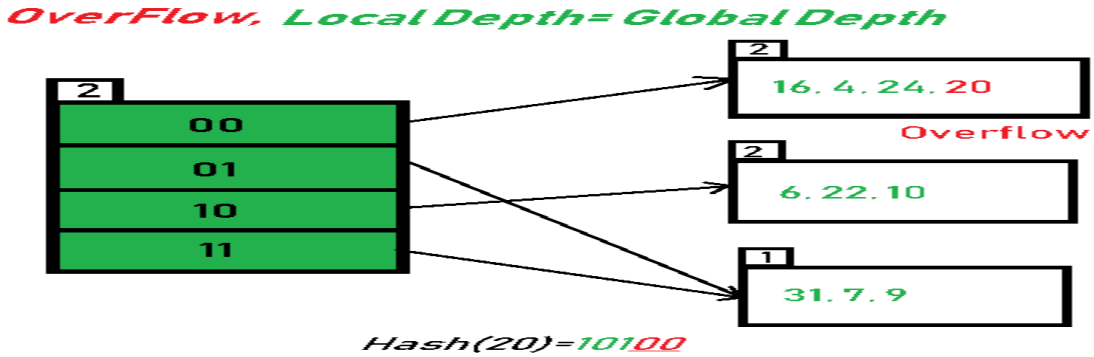
Inserting 24 and 10: 24(11**00**) and 10 (10**10**) can be hashed based on directories with id 00 and 10. Here, we encounter no overflow condition.



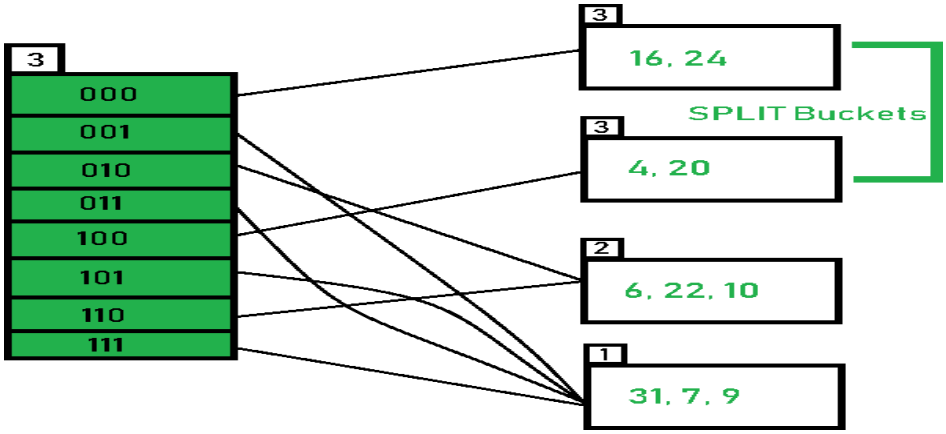
Inserting 31,7,9: All of these elements[31(111**11**), 7(1**11**), 9(10**01**)] have either 01 or 11 in their LSBs. Hence, they are mapped on the bucket pointed out by 01 and 11. We do not encounter any overflow condition here.



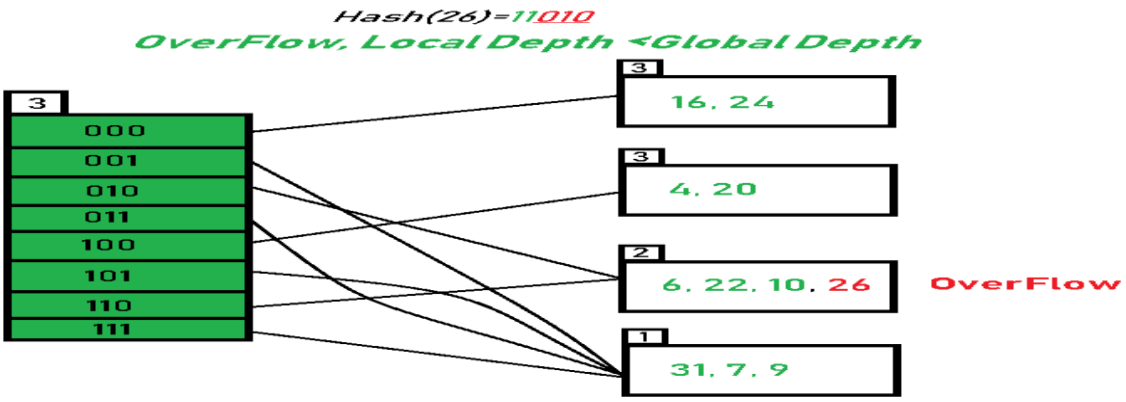
Inserting 20: Insertion of data element 20 (101**00**) will again cause the overflow problem.



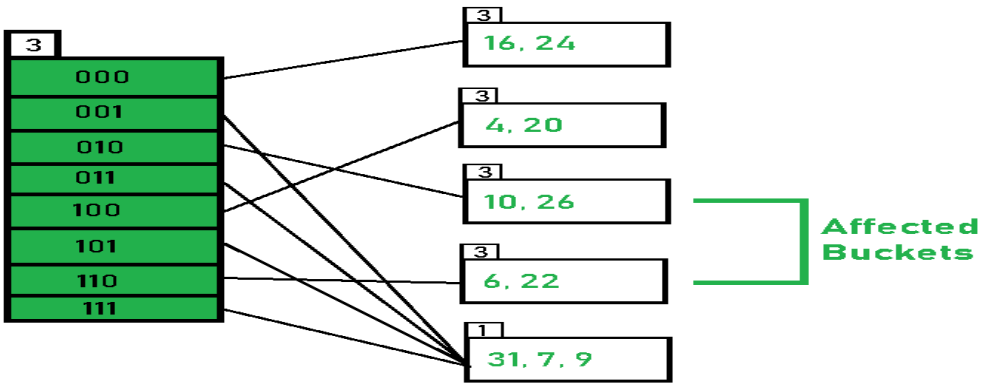
20 is inserted in bucket pointed out by 00. As directed by **Step 7-Case 1**, since the **local depth of the bucket = global-depth**, directory expansion (doubling) takes place along with bucket splitting. Elements present in overflowing bucket are rehashed with the new global depth. Now, the new Hash table looks like this:



Inserting 26: Global depth is 3. Hence, 3 LSBs of 26(11**010**) are considered. Therefore 26 best fits in the bucket pointed out by directory 010.



The bucket overflows, and, as directed by **Step 7-Case 2,** since the **local depth of bucket < Global depth (2<3)**, directories are not doubled but, only the bucket is split and elements are rehashed.
Finally, the output of hashing the given list of numbers is obtained.



- **Hashing of 11 Numbers is Thus Completed.**

b) Explain applications of Hash Table. 4-M

→ **1 – Mark for each application explanation**

1. File System :

The hashing is used for the linking of the file name to the path of the file. When you interact with a file system as a user, you see the file name, maybe the path to the file. But to actually 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.

2. Pattern Matching :

The hashing is also used to search for patterns in the strings. One of the famous algorithms that use hashing for the searching of a pattern in a string is Rabin-Karp Algorithm . The pattern matching is also used to detect plagiarism.

3. Programming Language :

In most of the programming languages, there are built-in data types or data structures in the standard library that are based on hash tables. For example, dict or dictionary in Python, or HashMap in Java.

4. Compilers :

For identifying the keywords in the programming languages, the compiler uses the hash table to store these keywords and other identifiers to compile the program.

C) What is Hashing. Explain Different methods of Hash function calculations. 5-M

→

Hashing is a fundamental data structure that efficiently stores and retrieves data in a way that allows for quick access. It involves mapping data to a specific index in a hash table using a hash function, enabling fast retrieval of information based on its key. **1 - Mark**

Types of Hash Function :

1. Division Method.
2. Mid Square Method.
3. Folding Method.
4. Multiplication Method.

1. Division Method: **2 – Mark for each type explanation(any 2)**

This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

h(K) = k mod M

Here,

k is the key value, and

M is the size of the hash table.

It is best suited that M is a prime number as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division.

Example:

k = 12345

M = 95

h(12345) = 12345 mod 95

= 90

2. Mid Square Method:

The mid-square method is a very good hashing method. It involves two steps to compute the hash value-Square the value of the key k i.e. k2

Extract the middle r digits as the hash value.

Formula:

h(K) = h(k x k)

Here,

k is the key value.

The value of r can be decided based on the size of the table.

Example:

Suppose the hash table has 100 memory locations. So r = 2 because two digits are required to map the key to the memory location.

k = 60

k x k = 60 x 60

= 3600

h(60) = 60

The hash value obtained is 60

3. Digit Folding Method:

This method involves two steps:

Divide the key-value k into a number of parts i.e. k1, k2, k3,….,kn, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.

Add the individual parts. The hash value is obtained by ignoring the last carry if any.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

Here,

s is obtained by adding the parts of the key k

Example:

k = 12345

k1 = 12, k2 = 34, k3 = 5

s = k1 + k2 + k3

= 12 + 34 + 5

= 51

h(K) = 51

Q.3 a) Explain how to convert general tree to binary tree with example. 4-M

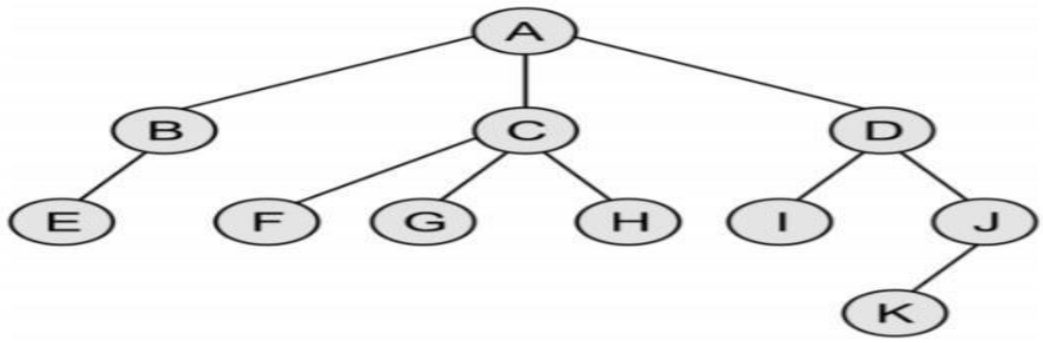
→

Following are the rules to convert a Generic(N-array Tree) to a Binary Tree: 1 - Mark

- The root of the Binary Tree is the Root of the Generic Tree.
- The left child of a node in the Generic Tree is the Left child of that node in the Binary Tree.
- The right sibling of any node in the Generic Tree is the Right child of that node in the Binary Tree.

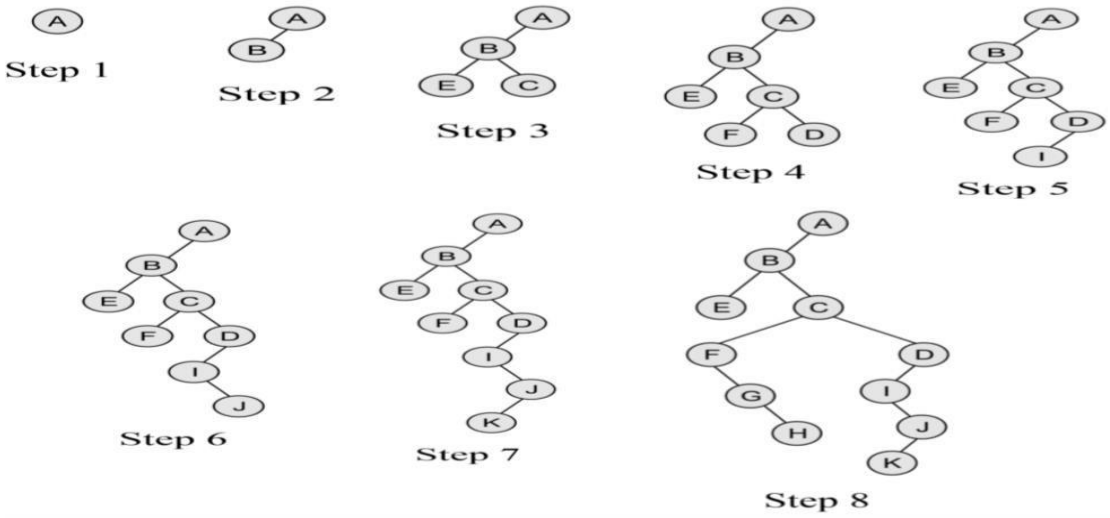
Examples: 3 - Mark

Convert the following Generic Tree to Binary Tree:



4

Solution :



b) Write a Non-recursive pseudocode for inorder traversal of binary tree. 5-M

→

Pseudo Code: **5 - Mark**

1. Initialize current as root
2. While current is not NULL

If the current does not have left child

a) Print current's data

b) Go to the right, i.e., current = current->right

Else

a) Find rightmost node in current left subtree OR

node whose right child == current.

If we found right child == current

a) Update the right child as NULL of that node whose right child is current

b) Print current's data

c) Go to the right, i.e. current = current->right

Else

a) Make current as the right child of that rightmost

node we found; and

b) Go to this left child, i.e., current = current->left

-

b) Describe Binary Search tree deletion with example. 6-M

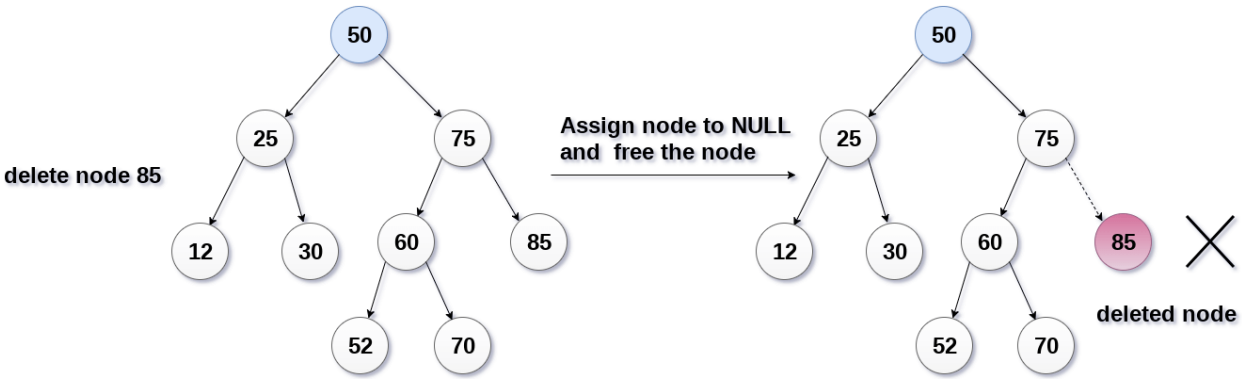
→ Deletion

Delete function is used to delete the specified node from a binary search tree. However, we must delete a node from a binary search tree in such a way, that the property of binary search tree doesn't violate. There are three situations of deleting a node from binary search tree.

1. The node to be deleted is a leaf node **2 - Mark**

It is the simplest case, in this case, replace the leaf node with the NULL and simple free the allocated space.

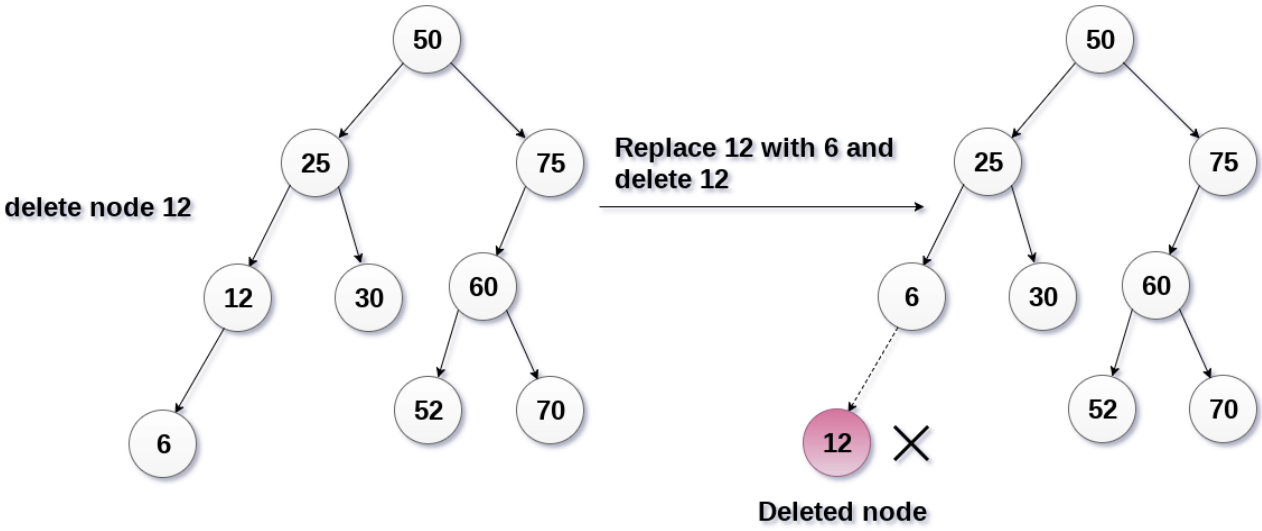
In the following image, we are deleting the node 85, since the node is a leaf node, therefore the node will be replaced with NULL and allocated space will be freed.



2. The node to be deleted has only one child. **2 - Mark**

In this case, replace the node with its child and delete the child node, which now contains the value which is to be deleted. Simply replace it with the NULL and free the allocated space.

In the following image, the node 12 is to be deleted. It has only one child. The node will be replaced with its child node and the replaced node 12 (which is now leaf node) will simply be deleted.



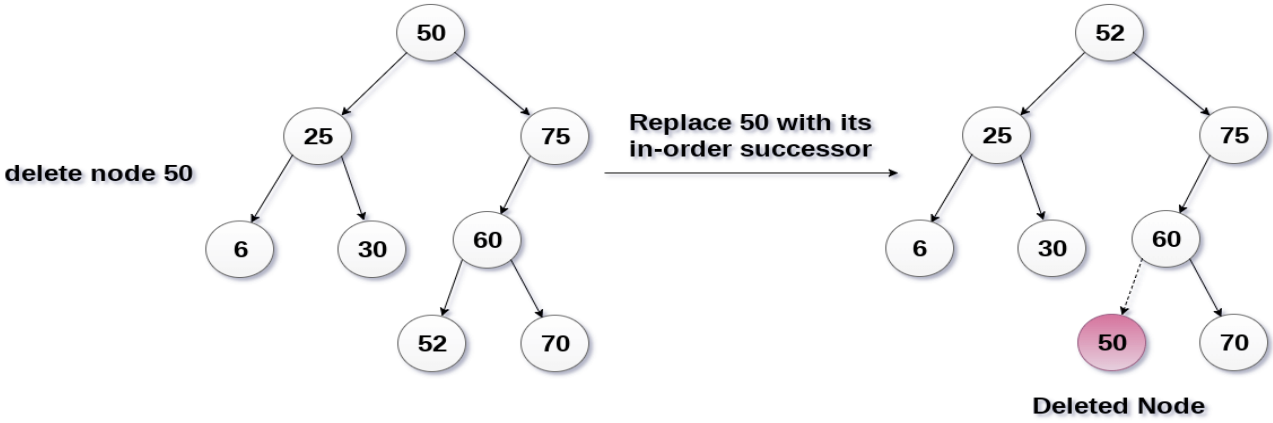
3. The node to be deleted has two children. **2 - Mark**

It is a bit complexed case compare to other two cases. However, the node which is to be deleted, is replaced with its in-order successor or predecessor recursively until the node value (to be deleted) is placed on the leaf of the tree. After the procedure, replace the node with NULL and free the allocated space.

In the following image, the node 50 is to be deleted which is the root node of the tree. The in-order traversal of the tree given below.

6, 25, 30, 50, 52, 60, 70, 75.

replace 50 with its in-order successor 52. Now, 50 will be moved to the leaf of the tree, which will simply be deleted.



OR

Q.4 a) Write a pseudo code for BFS tree traversal with example. 4-M

- →

```
BFS (G, s)              //Where G is the graph and s is the source node
    let Q be queue.
    Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

    mark s as visited.
    while ( Q is not empty)
        //Removing that vertex from queue,whose neighbour will be visited now
        v = Q.dequeue( )

      //processing all the neighbours of v
      for all neighbours w of v in Graph G
          if w is not visited
                Q.enqueue( w )          //Stores w in Q to further visit its neighbour
                mark w as visited.
```

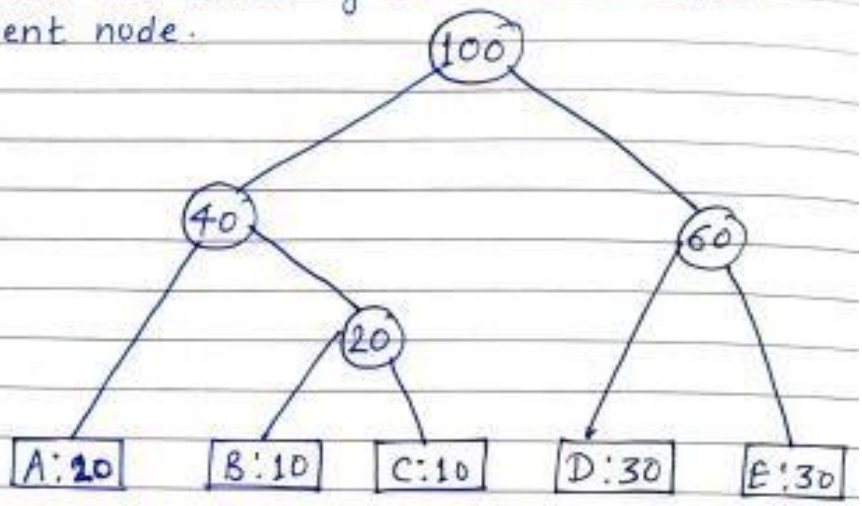
b) Construct Huffman's tree. Determine code for following characters whose frequencies as are given : 5-M

Character	A	B	C	D	E
Frequencies	20	10	10	30	30

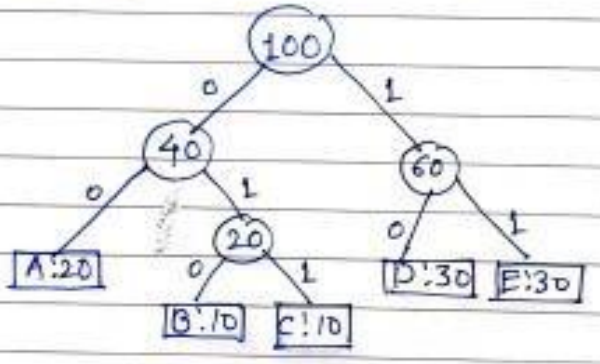
→

Q.4 b)

Ans → Step 1 : Combine two entries of the table to form parent node.



Step 2 : For encoding, the left branch is encoded as '0' and right branch as '1'



Step 3 : Huffmans code table

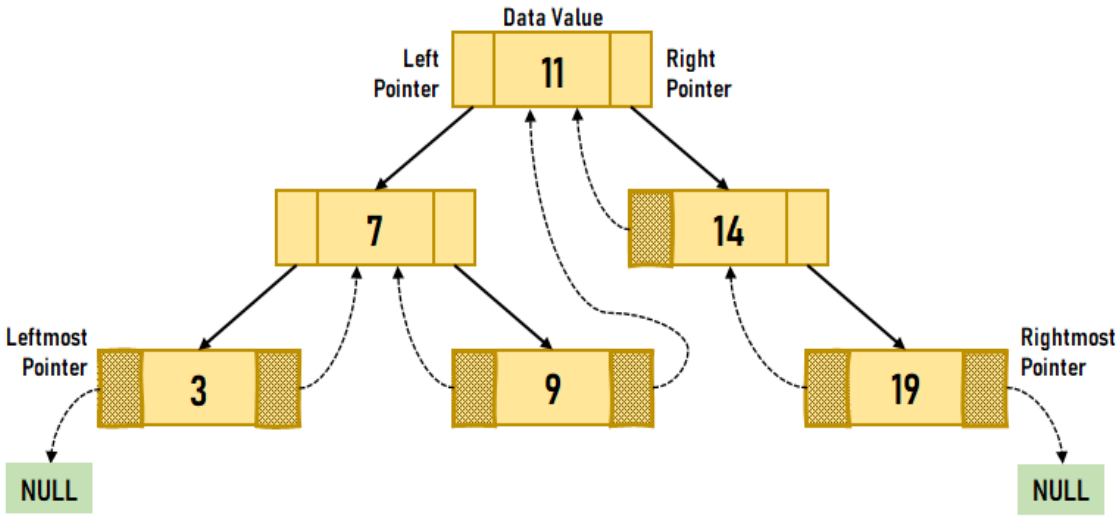
Data	Weight	Code
A	20	00
B	10	010
C	10	011
D	30	10
E	30	11

c) What is necessity of threaded binary tree? Explain advantages and disadvantages of TBT.

6-M

→ The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

- A threaded binary tree is a type of binary tree data structure where the empty left and right child pointers in a binary tree are replaced with threads that link nodes directly to their in-order predecessor or successor, thereby providing a way to traverse the tree without using recursion or a stack.
- Threaded binary trees can be useful when space is a concern, as they can eliminate the need for a stack during traversal. However, they can be more complex to implement than standard binary trees. **2 - Mark**



Advantages of Threaded Binary Tree : **2 - Mark**

1. It eliminates the use of stack as it perform linear traversal, so save memory.
2. Enables to find parent node without explicit use of parent pointer
3. Threaded tree give forward and backward traversal of nodes by in-order fashion
4. Nodes contain pointers to in-order predecessor and successor
5. In threaded binary tree there is no NULL pointer present. Hence memory wastage in occupying NULL links is avoided.
6. The threads are pointing to successor and predecessor nodes. This makes us to obtain predecessor and successor node of any node quickly.
7. There is no need of stack while traversing the tree, because using thread links we can reach to previously visited nodes.

Disadvantages of Threaded Binary Tree : **2 - Mark**

1. Every node in threaded binary tree need extra information(extra memory) to indicate whether its left or right node indicated its child nodes or its inorder predecessor or successor. So, the node consumes extra memory to implement.
2. Insertion and deletion are way more complex and time consuming than the normal one since both threads and ordinary links need to be maintained.
3. Implementing threads for every possible node is complicated.
4. **Increased complexity:** Implementing a threaded binary tree requires more complex algorithms and data structures than a regular binary tree. This can make the code harder to read and debug.
5. **Extra memory usage:** In some cases, the additional pointers used to thread the tree can use up more memory than a regular binary tree. This is especially true if the tree is not fully balanced, as threading a skewed tree can result in a large number of additional pointers.
6. **Limited flexibility:** Threaded binary trees are specialized data structures that are optimized for specific types of traversal. While they can be more efficient than regular binary trees for these types of operations, they may not be as useful in other scenarios. For example, they cannot be easily modified (e.g. inserting or deleting nodes) without breaking the threading.
7. **Difficulty in parallelizing:** It can be challenging to parallelize operations on a threaded binary tree, as the threading can introduce data dependencies that make it difficult to process nodes independently. This can limit the performance gains that can be achieved through parallelism

**********THE END**********