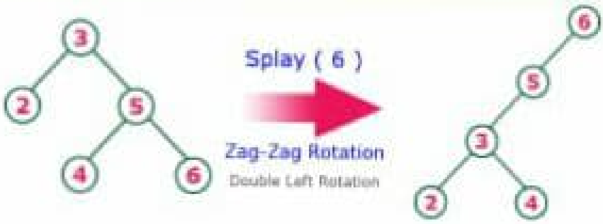
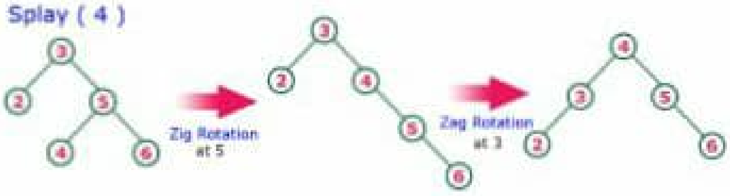


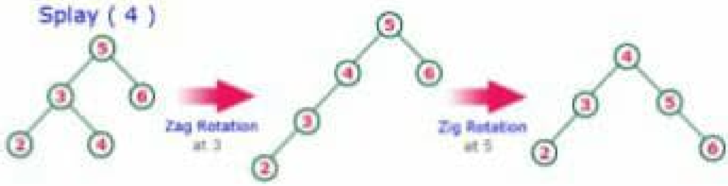
Zig-Zag Rotation

The **Zig-Zag Rotation** in splay tree is a sequence of zig rotation followed by zag rotation. In zig-zag rotation, every node moves one position to the right followed by one position to the left from its current position. Consider the following example...



Zag-Zig Rotation

The **Zag-Zig Rotation** in splay tree is a sequence of zag rotation followed by zig rotation. In zag-zig rotation, every node moves one position to the left followed by one position to the right from its current position. Consider the following example...



Every Splay tree must be a binary search tree but it is need not to be balanced tree.

In splay tree, to splay any element we use the following rotation operations...

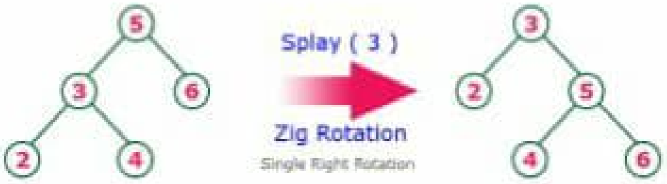
Rotations in Splay Tree

- 1. Zig Rotation
- 2. Zag Rotation
- 3. Zig - Zig Rotation
- 4. Zag - Zag Rotation
- 5. Zig - Zag Rotation
- 6. Zag - Zig Rotation

Example

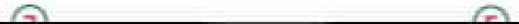
Zig Rotation

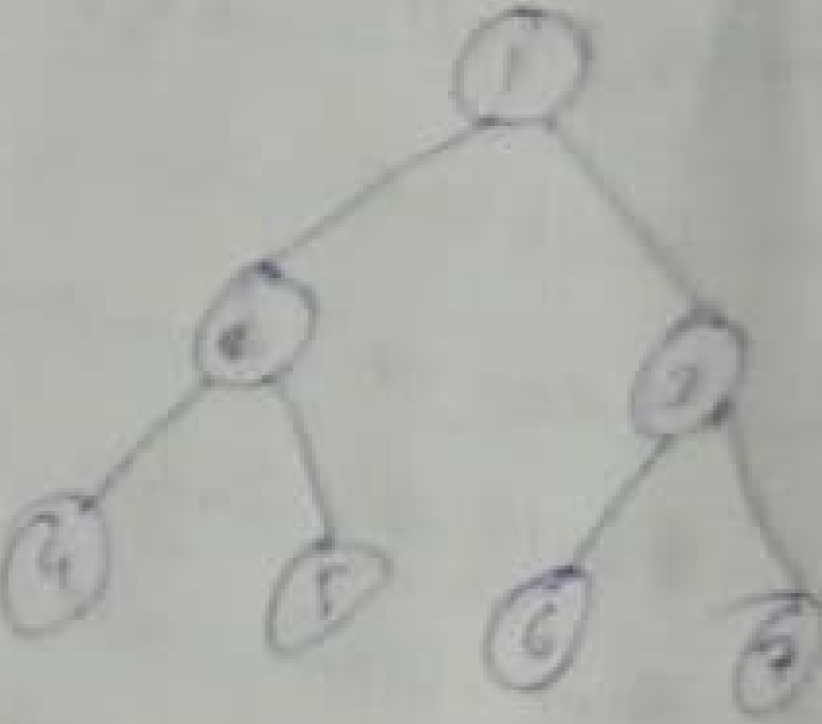
The **Zig Rotation** in splay tree is similar to the single right rotation in AVL Tree rotations. In zig rotation, every node moves one position to the right from its current position. Consider the following example...



Zag Rotation

The **Zag Rotation** in splay tree is similar to the single left rotation in AVL Tree rotations. In zag rotation, every node moves one position to the left from its current position. Consider the following example...





* Traverse through left Subtree first & traverse through the right Subtree(s)

5.3 : Compressed Tries

Q.17 What is compressed trie ? Explain it with suitable example. **[JNTU : Part B, Marks 5]**

Ans. : A compressed trie is a kind of standard trie in which internal node has atleast a degree of two. The redundant nodes are obtained by compressing the chains from standard trie.

For example

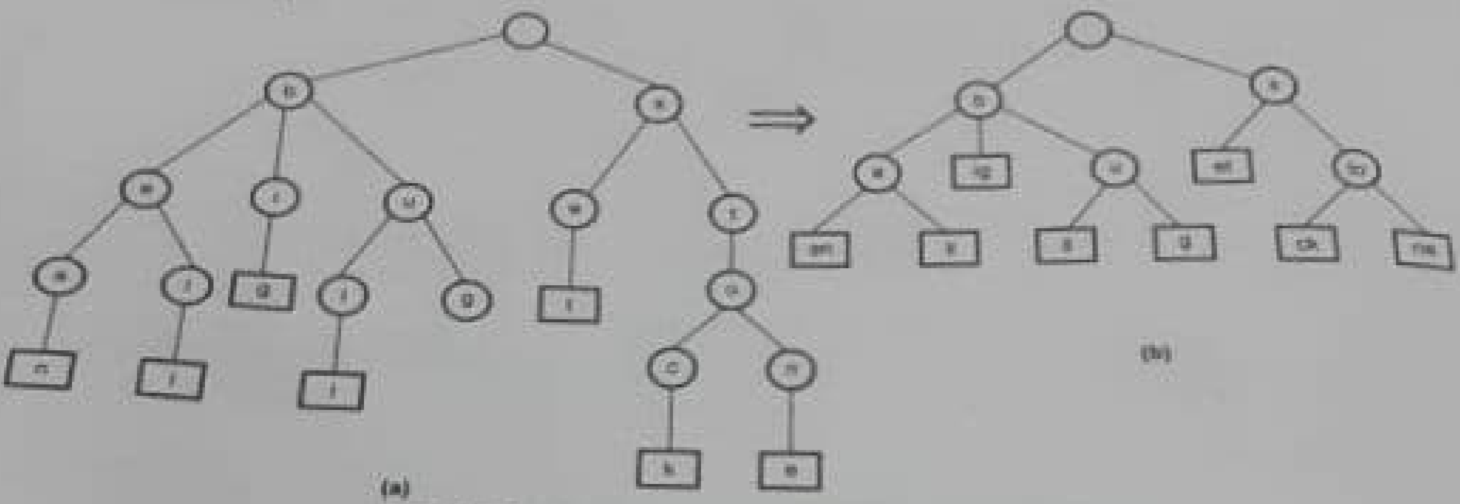
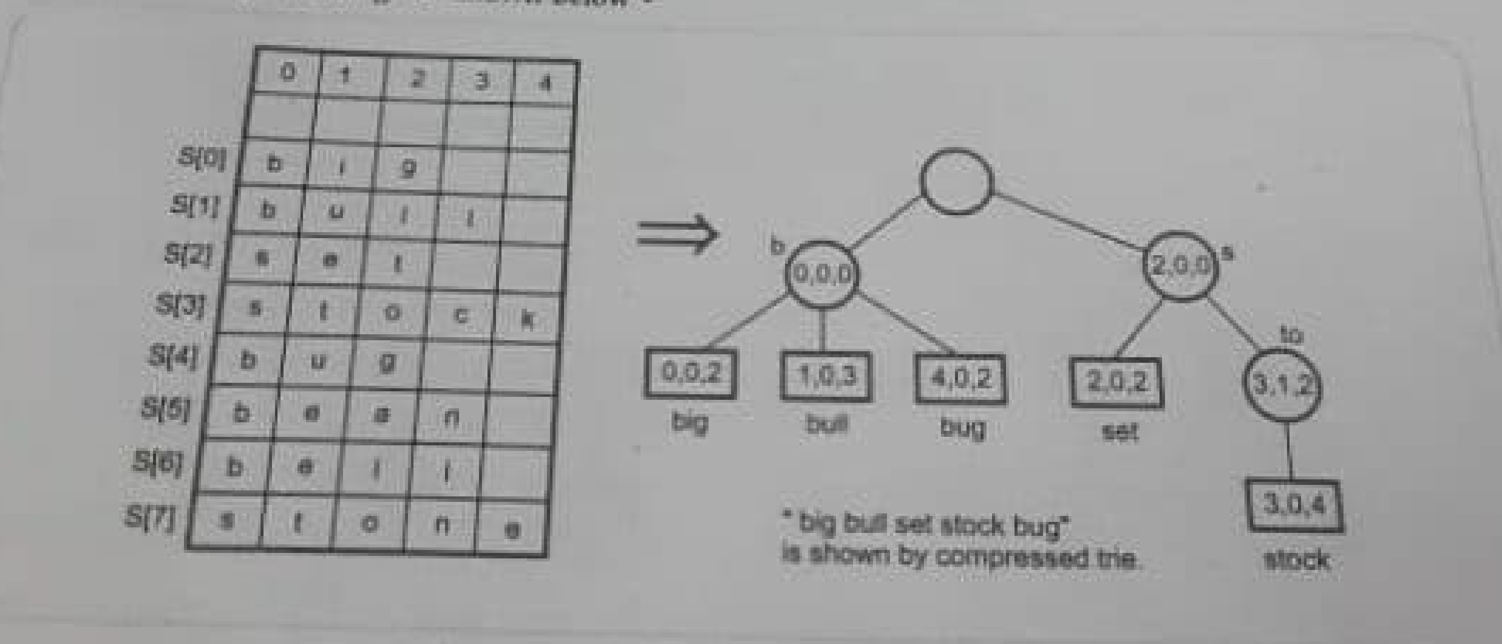


Fig. Q.17.1 Compressed tree obtained from standard trie

The compact representation of compressed trie can be done by using array of strings. Consider the array storing substrings as shown below -



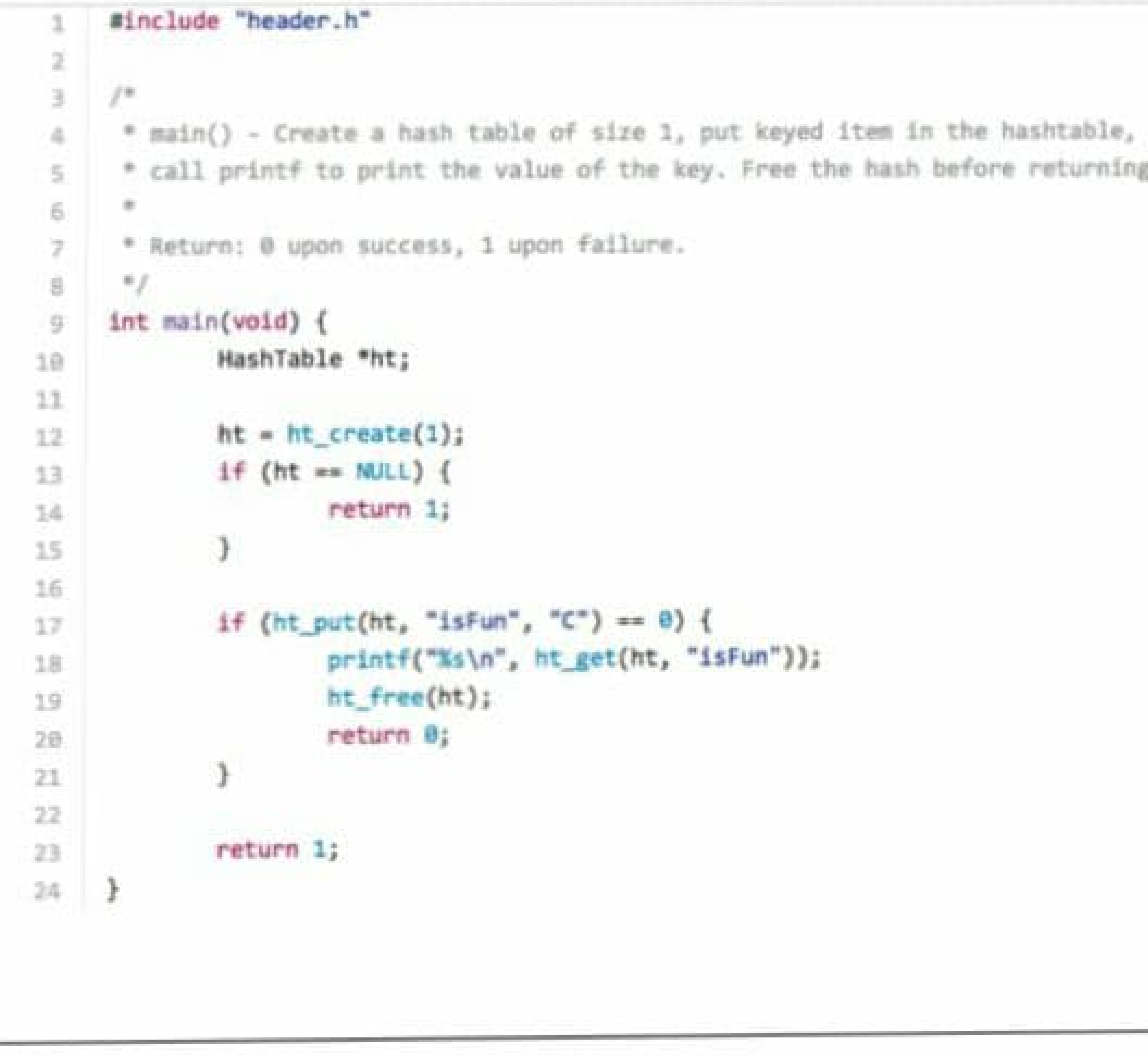
	0	1	2	3	4
S[0]	b	i	g		
S[1]	b	u	l	l	
S[2]	s	e	t		
S[3]	s	t	o	c	k
S[4]	b	u	g		
S[5]	b	e	a	n	
S[6]	b	e	l	l	
S[7]	s	t	o	n	e

" big bull set stock bug"
is shown by compressed trie.

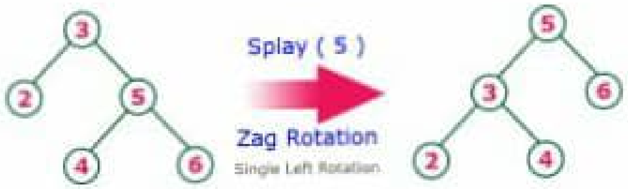
Fig. Q.17.2 Compact representation of trie

The compressed trie takes O(n) space to store the text where n is number of strings in set S.

C Program To Implement Dictionary Using Hashing Algorithms

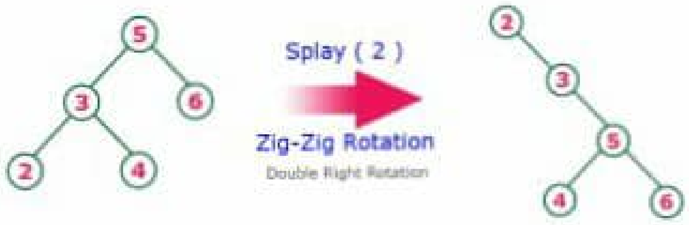


```c
#include "header.h"

/*
 * main() - Create a hash table of size 1, put keyed item in the hashtable,
 * call printf to print the value of the key. Free the hash before returning
 *
 * Return: 0 upon success, 1 upon failure.
 */
int main(void) {
        HashTable *ht;

        ht = ht_create(1);
        if (ht == NULL) {
                return 1;
        }

        if (ht_put(ht, "isFun", "C") == 0) {
                printf("%s\n", ht_get(ht, "isFun"));
                ht_free(ht);
                return 0;
        }

        return 1;
}
```



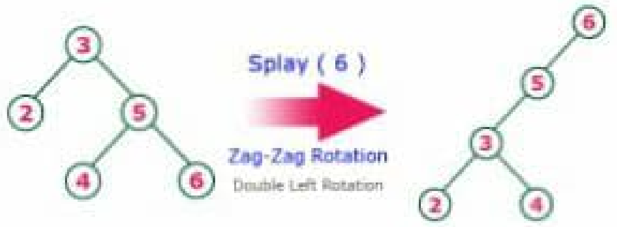
Zig-Zig Rotation

The **Zig-Zig Rotation** in splay tree is a double zig rotation. In zig-zig rotation, every node moves two positions to the right from its current position. Consider the following example...



Zag-Zag Rotation

The **Zag-Zag Rotation** in splay tree is a double zag rotation. In zag-zag rotation, every node moves two positions to the left from its current position. Consider the following example...



Zig-Zag Rotation

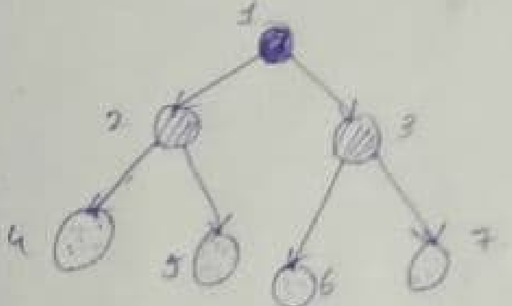
The **Zig-Zag Rotation** in splay tree is a sequence of zig rotation followed by zag rotation. In zig-zag rotation, every node moves one position to the right followed by one position to the left from its current position.

matching
ted in text
to Search
Search f
for the
Right
then P in
ch occur
not
shift
shift P
in P.

*

BFS:- Stands for Breadth First Search.

BFS is an algorithm that is Used to graph data on Searching Tree or traversing Structure. this algorithm Selects a Single node in a graph and then visit all the nodes adjacent to the Selected node. BFS accesses these nodes one by one.



Traverse through one level of children nodes, then traverse through the level of grand children node and so on.

DFS:- Stand for Depth first Search is an algorithm for traversing or Searching tree or graph data structure. the algorithm Starts at the root node and Explores as far as possible along each branch before backtracking

Postorder F G $\boxed{C}$

Inorder F C G

'C' is the parent node, F is the left child and G is the right child. So finally the tree will be as shown in Fig. 7

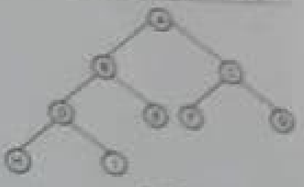


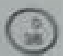
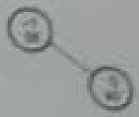
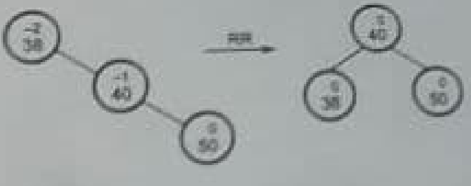
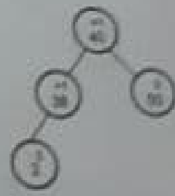
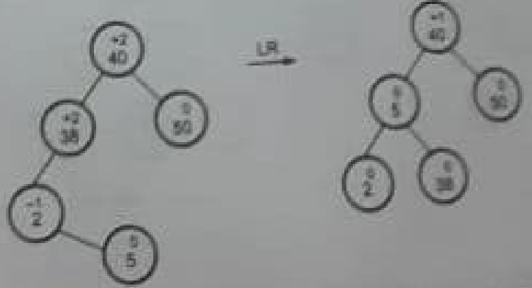
Fig. 7

OR

Q.7 *Construct the AVL tree of the following data*

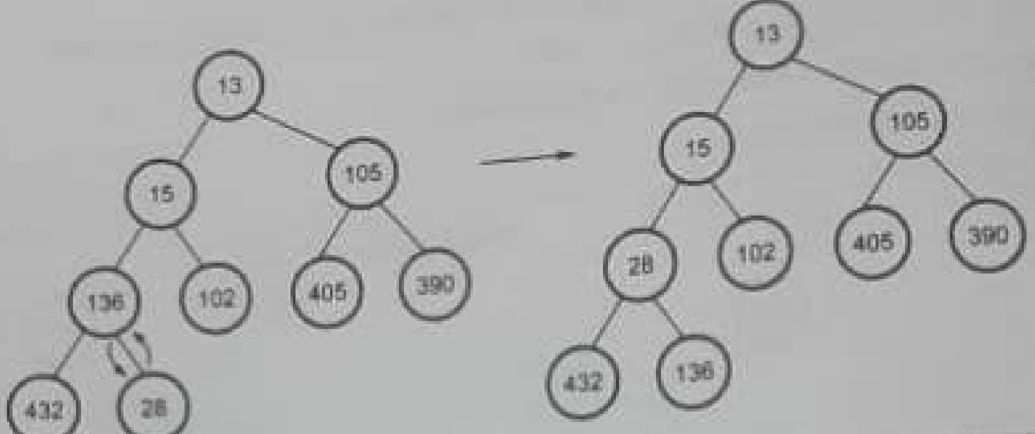
38 , 40, 50, 2, 5, 76, 25, 14, 7. (10)

Ans. :

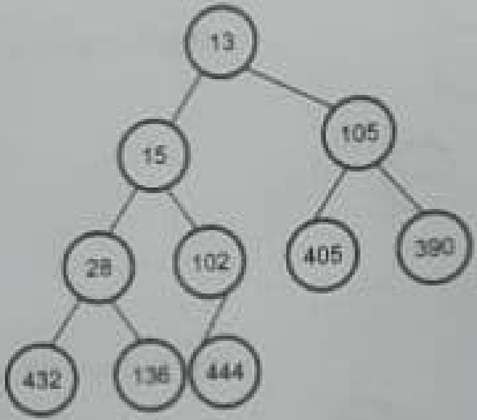
Step 1 : Insert 38 	Step 2 : Insert 40 
Step 3 : Insert 50 	Step 4 : Insert 2 
Step 5 : Insert 5 	

haracter of the
with the current character of
is called Bad Character

Step 9 : Insert 28

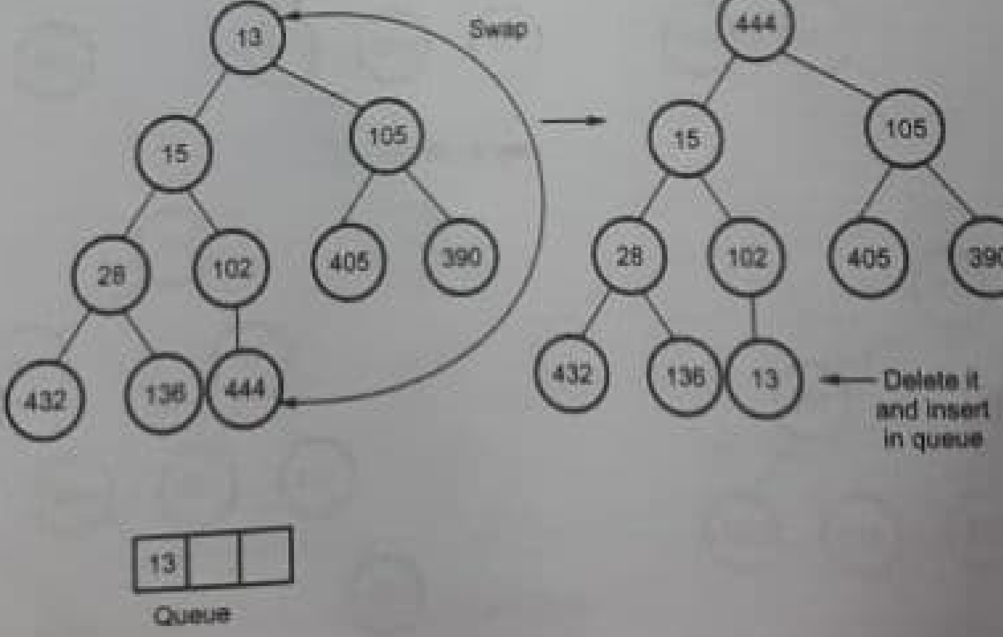


Step 10 : Insert 444

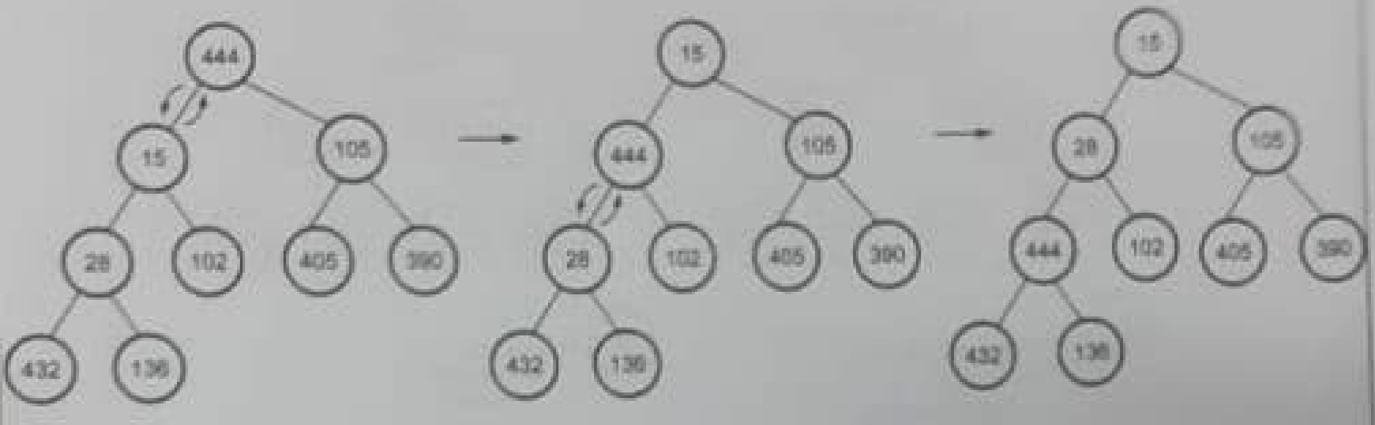


Deletion of root :

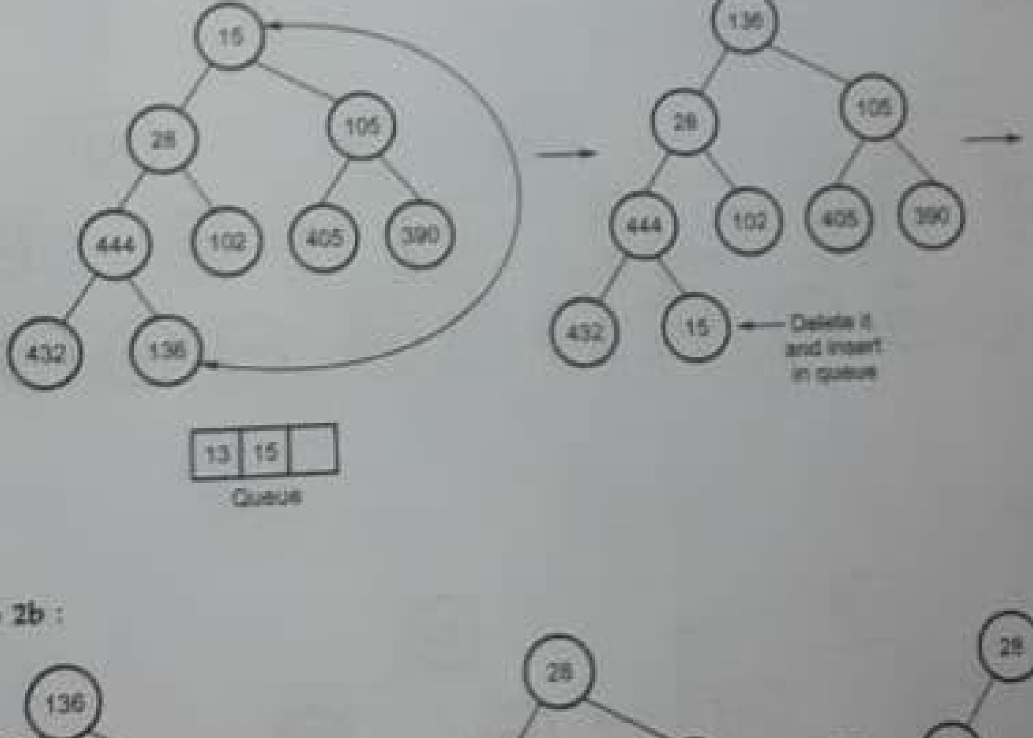
Step 1a :



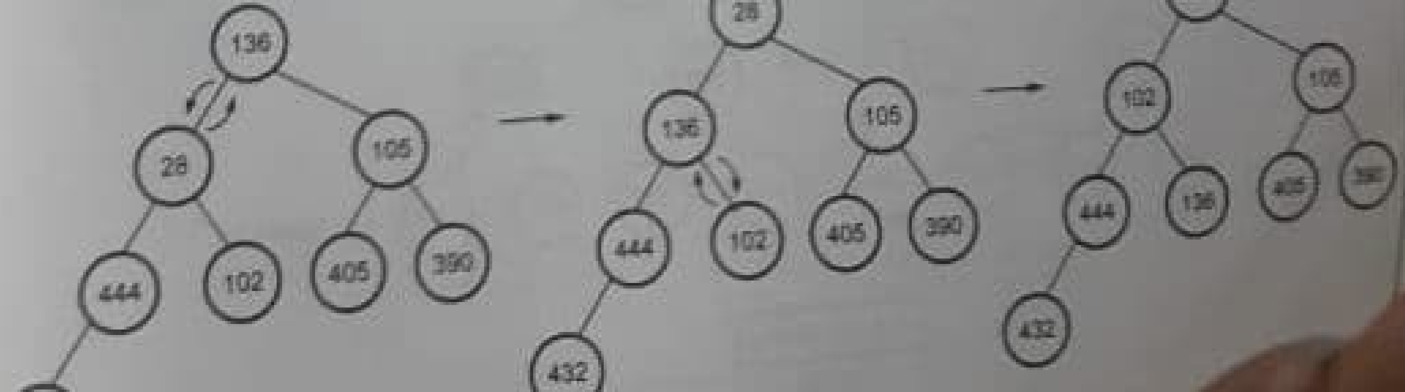
Step 1b :



Step 2a :



Step 2b :



h) Write about splay tree. [3]

Ans. : Splay tree is a self-adjusted binary search tree in which every operation on element rearranges the tree so that the element is placed at the root position of the tree.

Splaying an element, is the process of bringing it to the root position by performing suitable rotation operations.

i) What is graph ? Define degree of vertex. [3]

Ans. : Graph is a collection of vertices and nodes

Degree of the vertex of a graph is number of edges that are incident to the vertex. For example - In above graph - Degree(A) = 3, Degree (E) = 2, Degree (C) = 2

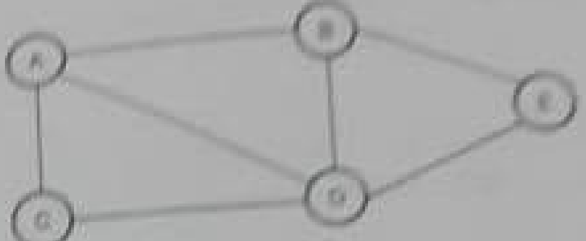


Fig. 2

j) Write a short notes standard tries. (Refer Q.16 of Chapter 5) [3]

PART - B (50 Marks)

Q.2 What is priority queue ? Explain the implementation of priority queue ? Write an algorithm for operations on priority queue. [10]

Ans. : The priority queue is a data structure having a collection of elements which are associated with specif ordering. There are two types of priority queues -

1. Ascending priority queue
2. Descending priority queue

1. **Ascending Priority Queue** - It is a collection of items in which the items can be inserted arbitarily but only smallest element can be removed.
2. **Descending Priority Queue** - It is a collection of items in which insertion of items can be in an order but only largest element can be removed.

In priority queue, the elements are arranged in any order and out of which only the smallest largest element allowed to delete each time.

1. Insertion operation

While implementing the priority queue we will apply a simple logic. That is while inserting element we will insert the element in the array at the proper position. For example, if the elem are placed in the queue as

9	12			
que[0]	que[1]	que[2]	que[3]	que[4]
front	rear			

And now if an element 8 is to be inserted in the queue then it will be at 0th location as -

8	9	12		
que[0]	que[1]	que[2]	que[3]	que[4]
front		rear		

DECEMBER - 2019

Data Structures (R18) (153AK)

Solved Paper

B. Tech., II - I [Common to CSE, IT]

Time : 3 Hours] [Maximum Marks : 75

Note : This question paper contains two parts A and B. Part A is compulsory which carries 25 marks. Answer all questions in Part A. Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b as sub questions.

PART - A

(25 Marks)

Q.1 a) *Explain how does linked stack differ from a linear stack.* [2]

Ans. : In linked stack the linked list is used in linear stack the array is used.

b) *Define searching.* [2]

Ans. : Searching is a process of locating desired element from a list of elements.

c) *How many binary trees are possible with four nodes ?* [2]

Ans. : There are 14 different binary trees. These are as follows -

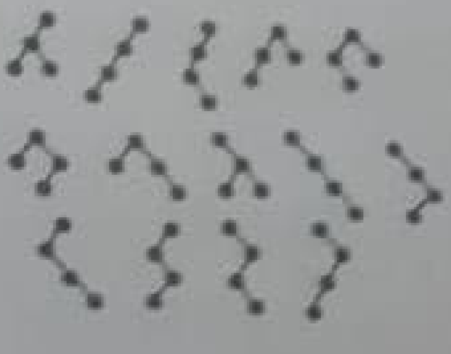


Fig. 1

d) *Define tree traversal.* (Refer Q.10 of Chapter 3) [2]

e) *What is pattern ?* [2]

Ans. : The pattern matching algorithm uses the character string called pattern which is searched from the given text.

f) *Write the pseudo code for reversing the list using stacks.* [3]

Ans. : Step 1 : Traverse each node and push it onto the stack.

Step 2 : Repeat step 1 until the linked list is empty

Step 3 : Pop the node and display the value of the node.

Step 4 : Repeat step 3 until stack is empty.

g) *Discuss about linear probing.* (Refer Q.19 of Chapter 2) [3]

If the next element comes as 11 then the queue will be -

8	9	11	12	
que[0] front	que[1]	que[2]	que[3] rear	que[4]

The C function for this operation is as given below -

```
int insert(int que[SIZE],int rear,int front)
{
    int item,j;
    printf("\nEnter the element: ");
    scanf("%d",&item);
    if(front ==-1)
         front++;
    j=rear;
    while(j>=0 && item<que[j])
    {
     que[j+1]=que[j];
     j--;
    }
     que[j+1]=item;
     rear=rear+1;
     return rear;
}
```

2. Deletion operation

In the deletion operation we are simply removing the element at the front.

For example, if queue is created like this -

8	9	11	12	
que[0] front	que[1]	que[2]	que[3] rear	que[4]

Then the element at que[0] will be deleted first.

~~8~~	9	11	12	
~~que[0]~~	que[1] front	que[2]	que[3] rear	que[4]

and then new front will be que[1].

The deletion operation in C is as given below -

```
int delet(int que[SIZE],int front)
{
    int item;
    item=que[front];
    printf("\n The item deleted is %d",item);
    front++;
    return front;
}
```

OR

Q.3 a) *Discuss about the stack with examples.* **(Refer Q.33 of Chapter 1)** [5+5]

b) *Write an algorithm to implement queue using stack.* **(Refer Q.71 of Chapter 1)**

Q.4 *What is collision ? Explain different collision resolution techniques with examples.* [10]

(Refer Q.20 of Chapter 2)

OR [10]

Q.5 *Describe the operations of skip list with an example.* **(Refer Q.6 and Q.7 of Chapter 2)** [10]

Q.6 *Write an algorithm for creation of binary tree using in - order traversal and post order traversals.*

Ans. : **Postorder** : H I D E B F G C A

Inorder : H D I B E A F C G

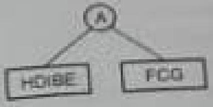


Fig. 3

Step 1 :

The last node in postorder sequence is the root node. In above example "A" is the root node. Now observe inorder sequence locate the "A". Left sequence of "A" indicates the left subtree and right sequence of "A" indicates the right sub-tree

i.e. as shown in Fig. 3.

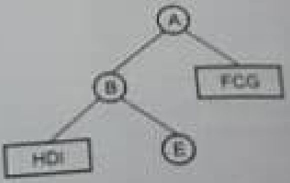


Fig. 4

Step 2 :

Now, with these alphabets H, D, I, B, E observe the postorder and sequences.

Postorder H I D E B

Inorder H D I B E

Here B is parent node, therefore pictorically tree will be,

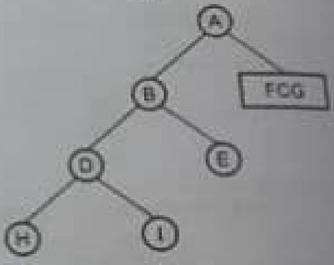


Fig. 5

Step 3 :

With the alphabets H, D and I observe both the sequences.

Postorder H I D

Inorder H D I

D is the parent node, H is leftmost node and I is the right child of D node. So tree will be as shown in Fig. 5.

Step 4 :

Now we will slove for right sub-tree of root "A". With the alphabets F, C, G observe both the sequences.

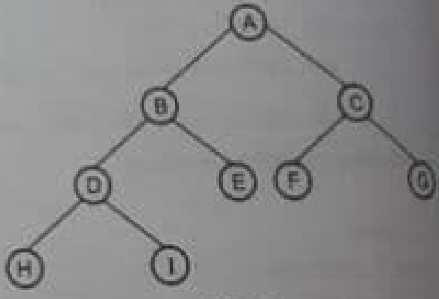
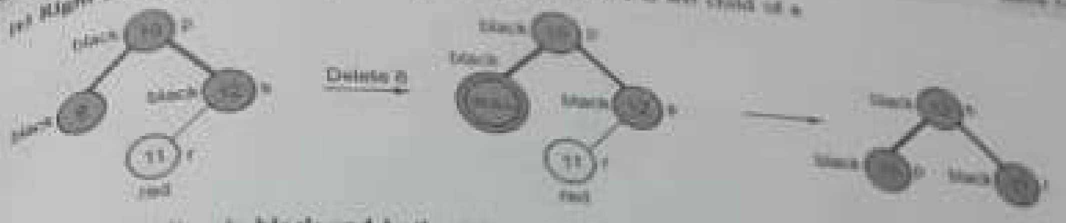


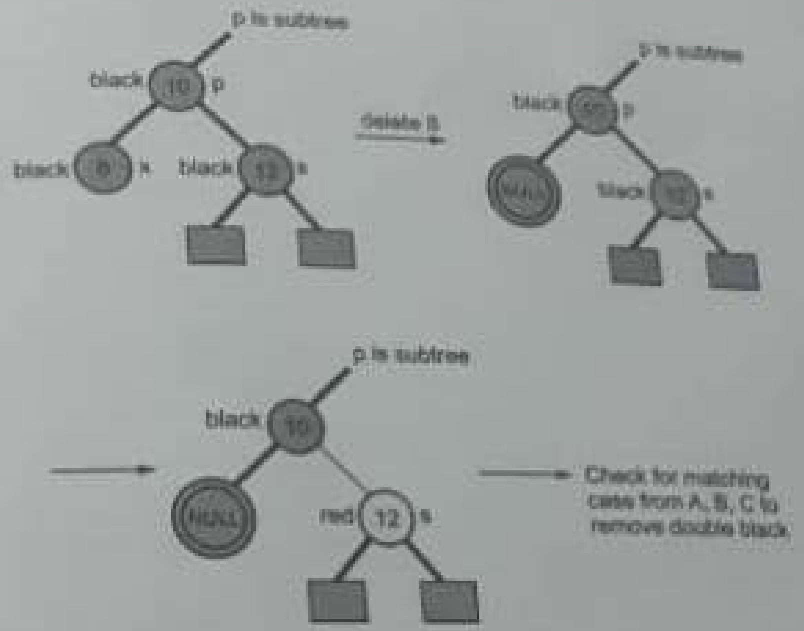
Fig. 6

iv) Right left case - The s is right child of its parent and r is left child of s.



Case B : If sibling is black and both of its children are black. In this case after removing the desired node, the recoloring is performed. And then to remove double black parent, the case matching from A, B and C is reapplied.

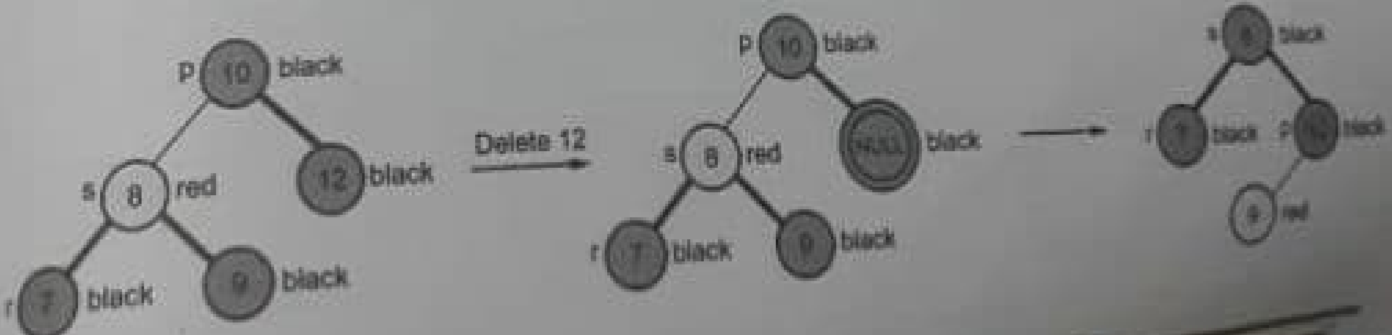
For example :



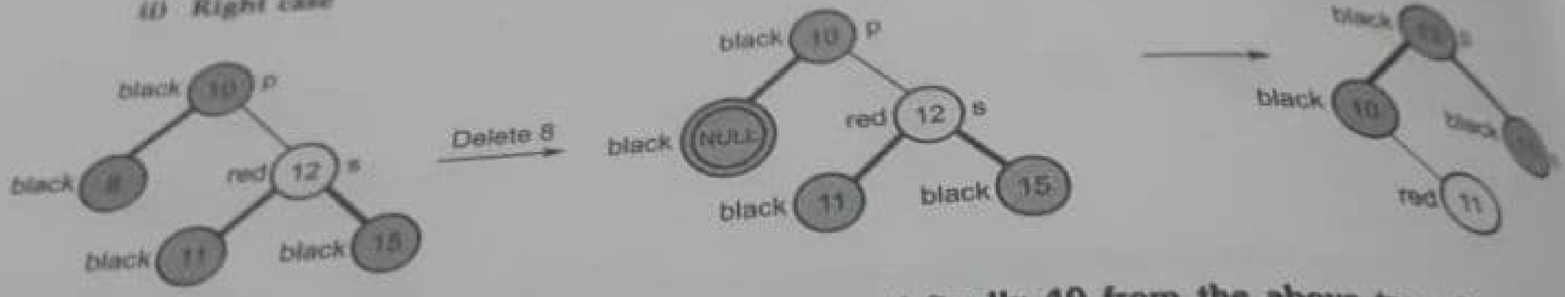
Case C : If sibling is red.

This case has two subcases

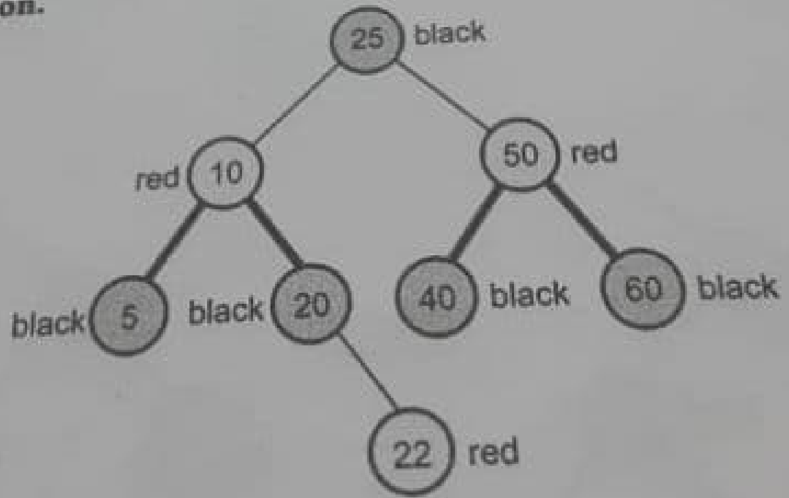
i) **Left case**



ii) Right case



Q.42 Following is a Red-Black tree. Delete 60, then 50 and finally 40 from the above tree. Show clearly balancing done after each deletion.

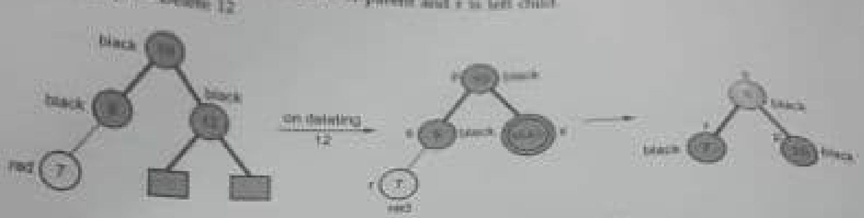


The above shown cases can be handled as follows -

Case A : If Sibling s is black and at least one child of it is red.

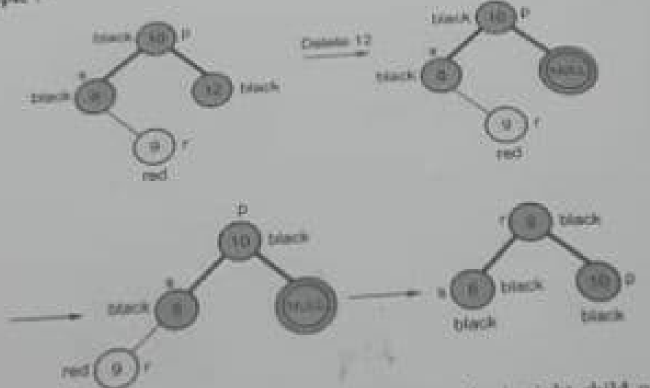
i) **Left left case** - The s is left child of parent and r is left child.

For example - Delete 12



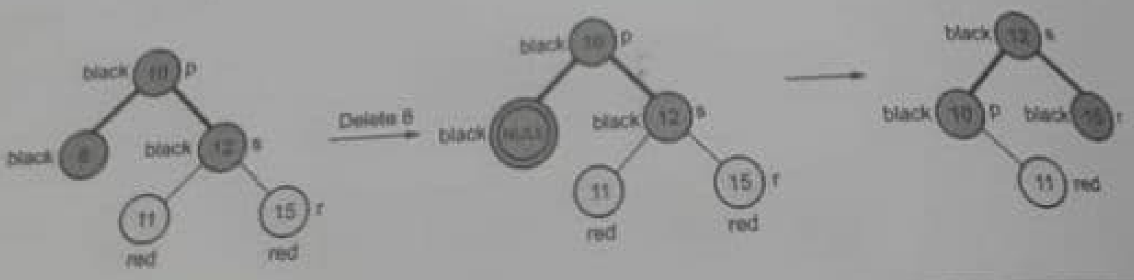
ii) **Left right case** - The s is a left child of its parent and r is a right child.

For example :



iii) **Right right case** - The s is right child of its parent node and r is right child of s. Both the children of s are red.

For example :



iv) Right left case :

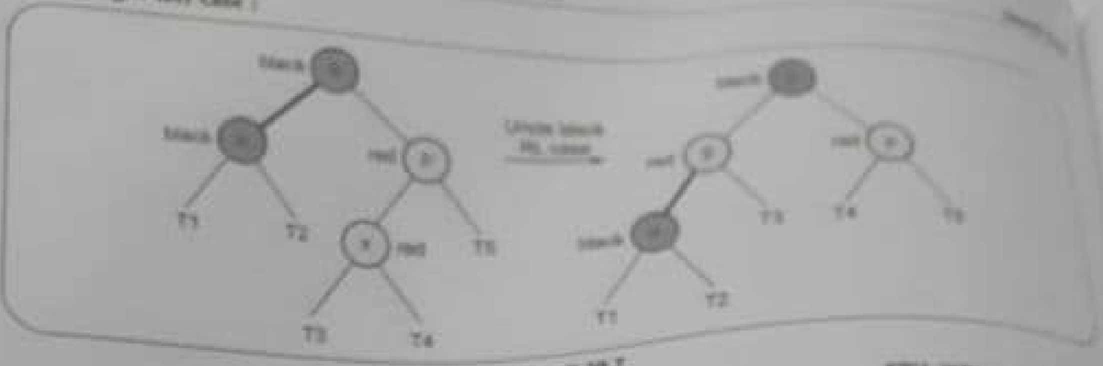


Fig. Q.39.7

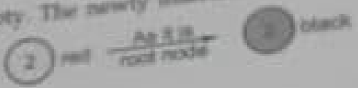
SE'1 2013 : Part A, Marks

Q.40 Insert 2, 1, 4, 5, 9, 3, 6 and 7 for red black tree.

Ans. :

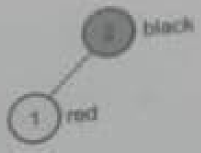
Step 1 : Insert 2

Initially the red black tree is empty. The newly inserted node should always be red. Hence



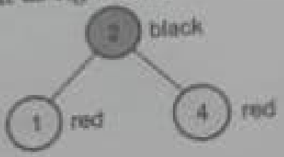
Step 2 : Insert 1

Make node for value 1 and it should be red as it is newly inserted node. Since 1 < 2, attach the node as left child of 2.



Step 3 : Insert 4

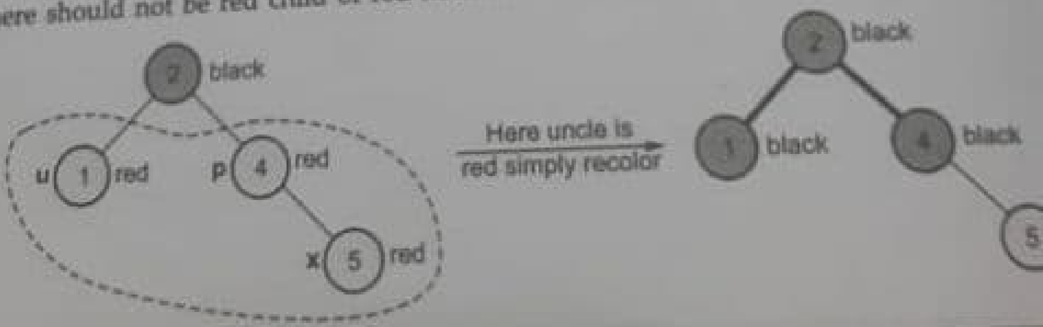
Make node 4 as red node and attach it as right child of 2.



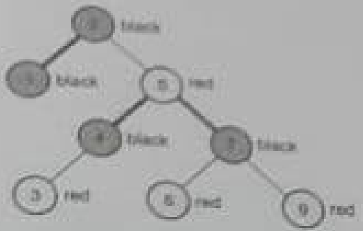
Step 4 : Insert 5

Here node 5 is a red node attached as right child of red node 4.

But there should not be red child of red node. Hence we need to make adjustments.



Thus finally we get, following red black tree.

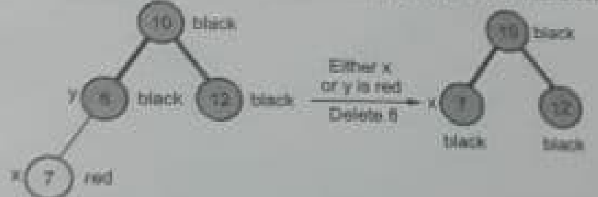


Q.[illegible] Explain deletion operation in red black tree. ☞ [JNTU : Part B, Marks 10]

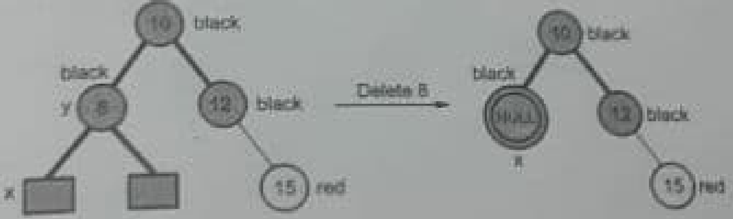
Ans. :

Step 1 : In Red Black tree we will handle the - deletion of leaf node or the node having one child. After deleting the node its place is replaced by its inorder successor (just similar to deletion operation in BST).

Step 2 : Case 1 : The node to be deleted is red or the node to be deleted has a red child.

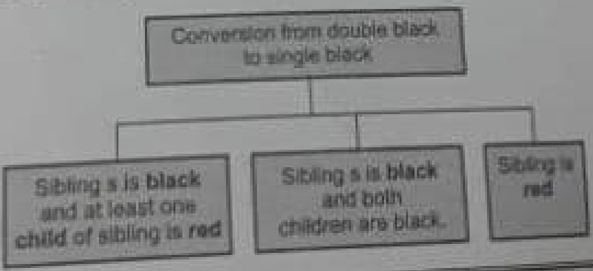


Step 3 : Case 2 : If Both x and y are black.



i) Delete node y and make the node x as double black.

ii) Now current node x is a double black node and we have to convert it to single black. Now this conversion is based on sibling.


Conversion from double black to single black
Sibling s is **black** and at least one **child** of sibling is **red**
Sibling s is **black** and both children are black.
Sibling is **red**

Boyre Moore Algorithm

* the algorithm :

i) Left left case :

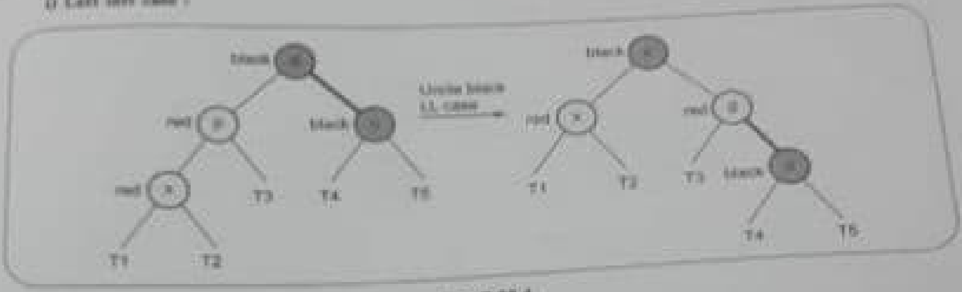


Fig. Q.39.4

ii) Left right case :

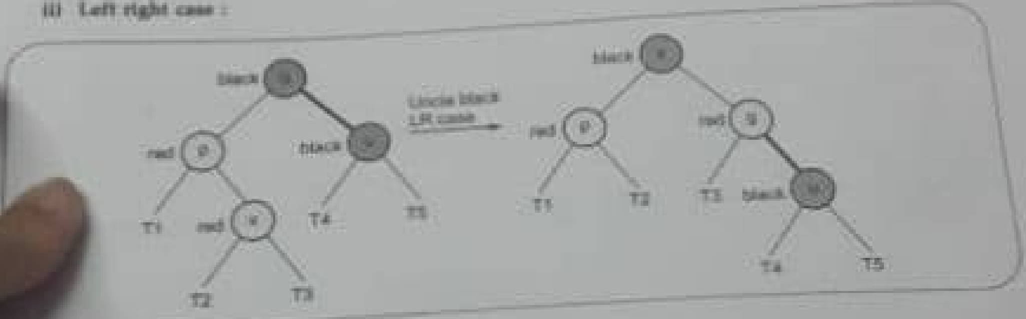


Fig. Q.39.5

iii) Right right case :

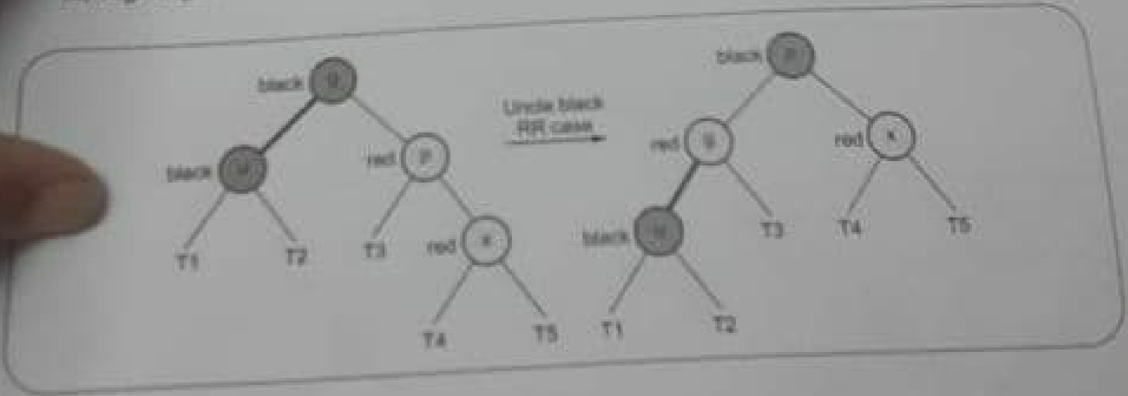


Fig. Q.39.6

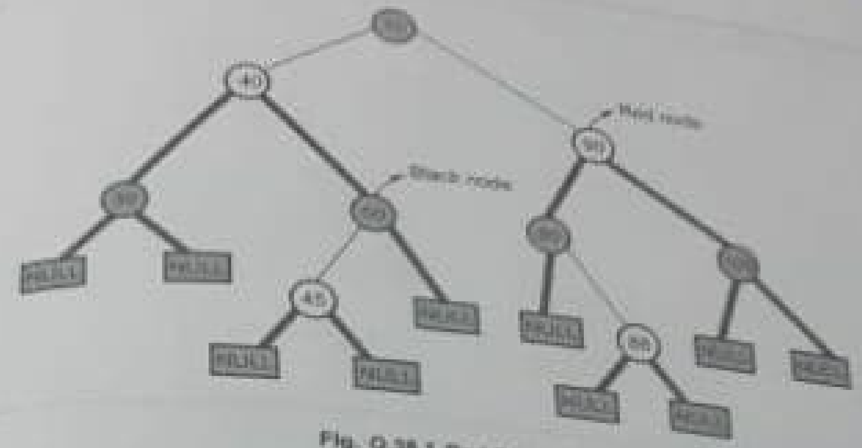


Fig. Q.38.1 Red-Black tree

3. The children of red node are black.
4. No root - to external node path has two consecutive red nodes (e.g. 70-90-80-88-NULL)
5. All the root to external node paths contain same number of black nodes (including root and external node).

For e.g. : Consider path 70-40-30-NULL and 70-90-80-88-NULL in both these paths 3 black nodes are there. Similarly other paths can be checked.

Q.41 Explain the insertion process in red black tree. **[JNTU : Part B, Marks 10]**

Ans. :
- Every new node which is to be inserted is marked red.
- Not every insertion causes imbalancing but if imbalancing occurs then that can be removed depending upon the configuration of tree before new insertion made.
- In Red black tree during insertion of a node two operations need to be performed for balancing the tree.

i) Recoloring ii) Rotation

Let, if x is a newly inserted node then there exists two cases -

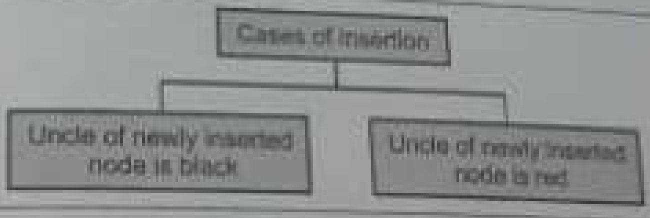

Cases of insertion
Uncle of newly inserted node is black
Uncle of newly inserted node is red

Fig. Q.41.1 Cases of insertion

Let us understand insertion operation in Red black tree.

Step 1 : Perform insertion of node x same like insertion in binary search tree.

Step 2 : The color of newly inserted node is red.

Step 3 : If x is a root node, change color of x to **black.**

Step 4 : If newly inserted node x is red and its parent is also red then only balancing is needed.

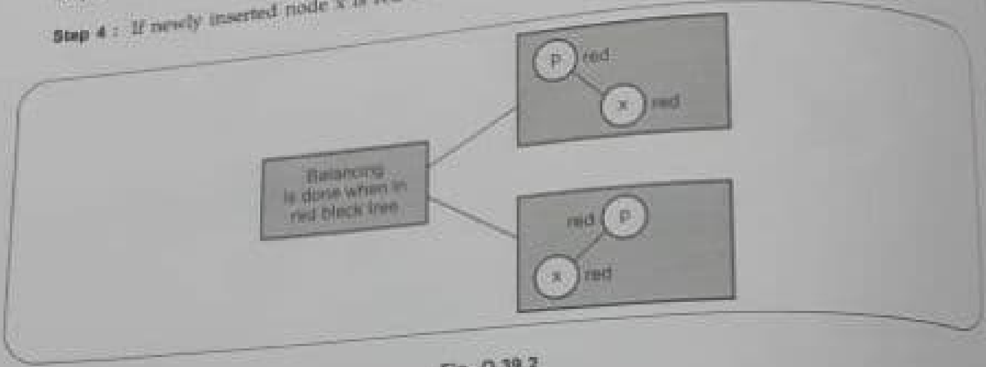


Fig. Q.39.2

Step 5 : As discussed earlier, there are two cases.

I) If x's uncle is **RED** then

i) Change color of parent and uncle as **black.**

ii) Change color of grand parent as red

II) If x's uncle is **black** then there are four configurations just similar to AVL tree. These configurations are LL, LR, RR and RL case. Let us understand them in detail.

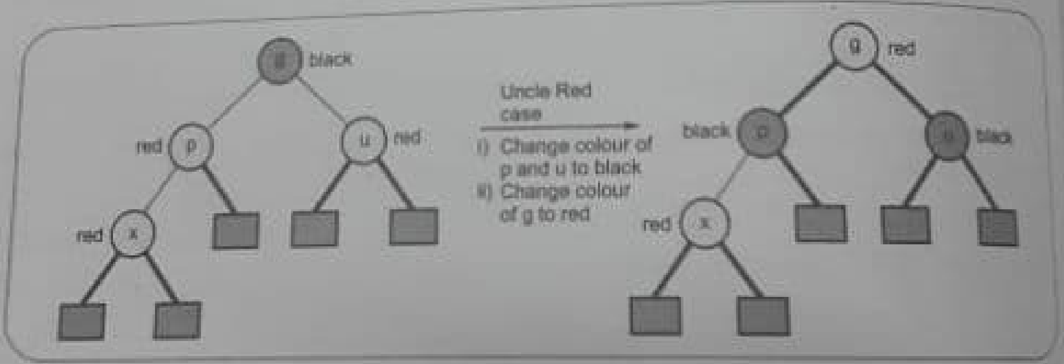

g
black
red
p
u
red
red
x
Uncle Red case
i) Change colour of p and u to black
ii) Change colour of g to red
g
red
black
p
u
black
red
x

Fig. Q.39.3

... is sufficiently large ...

... ages of rehashing ?
[JNTU : Part A, Marks 2]

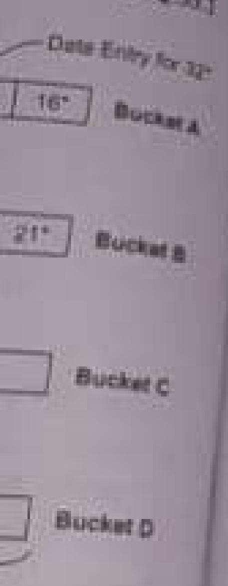
... vantages :

... es the programmer ...
... table size if required ...
... bled with simple hash ...
... urrence of collisions ...

... hing ? Explain with ...
... NTU : Part B, Marks 5]

... s a dynamic hashing ...
... ket is overflow, then ...
... led and data entries ...

... the directory of ...
... llowing Fig. Q.33.1



To locate a data entry, we apply a hash function to search the data we us last two digits of binary representation of number. For instance binary representation of 32* = 10000000. The last two bits are 00. Hence we store 32* accordingly.

Insertion operation :

- Suppose we want to insert 20* (binary 10100). But with 00, the bucket A is full. So we must **split the bucket** by allocating new bucket and redistributing the contents across the old bucket and its split image.
- For splitting, we consider last three bits of h(r).
- The redistribution while insertion of 20* is as shown in following Fig. Q.33.2.

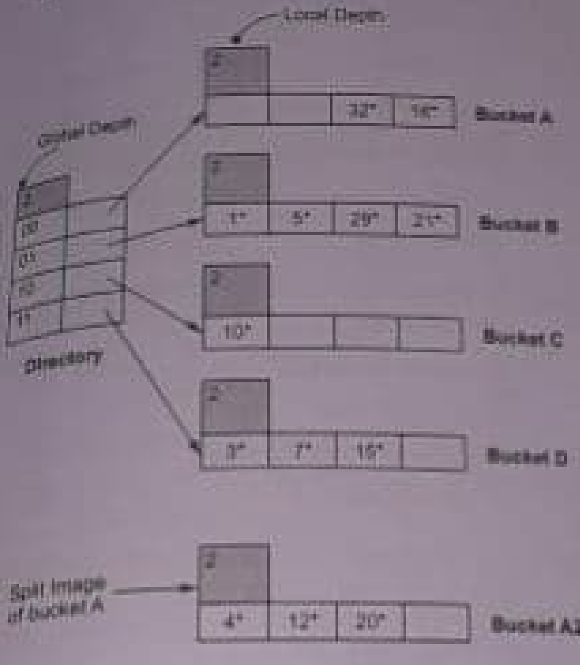


Fig. Q.33.2 During insertion process

- The split image of bucket A i.e. A2 and old bucket A are based on last two bits i.e. 00. Here we need two data pages, to adjacent additional data record. Therefore here it is necessary to **double the directory** using three bits instead of two bits.
- Hence there will be binary versions for buckets A and A2 as 000 and 100.

- In extendible hashing, last bits d is called **global depth** for directory and d is called **local depth** for data pages or buckets. After insertion of 20*, the **global depth** becomes 3 as we consider last three bits and local depth of A and A2 buckets become 3 as we are considering last three bits for placing the data records. Refer Fig. Q.33.3.

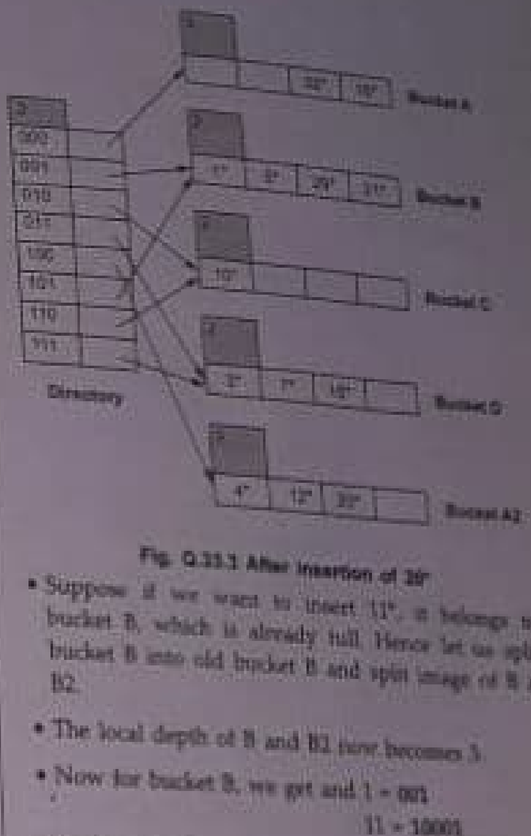


Fig. Q.33.3 After insertion of 20*

- Suppose if we want to insert 11*, it belongs to bucket B, which is already full. Hence let us split bucket B into old bucket B and split image of B as B2.
- The local depth of B and B2 now becomes 3.
- Now for bucket B, we get and 1 = 001
  11 = 10001
- For bucket B2, we get
  5 = 101
  29 = 11101
  and 21 = 10101

Ans. : One major problem associated with quadratic probing is that typically not all hash table slots will be on the probe sequence. Using H(K, i) = i² gives particularly inconsistent results. For many hash table sizes this probe function will cycle through a relatively small number of slots. If all slots on that cycle happen to be full, this means that the record cannot be inserted at all even if the remaining slots happen to be empty.

Q.29 Show the resulting input (3417, 3132, 7122, 5199, 5344, 6796 and 1893) and hash function h (n) = x (mod 10).

a) Open addressing hash table using quadratic probing.

b) Open [illegible] hash [illegible] with second hash funct[illegible]

[illegible] Nov.-09, Marks 16]

[illegible] tab[illegible]g quadratic

1	[illegible]
2	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	6796
[illegible]	3417
8	
9	5199

Fig. Q.29.1

16	
17	17
18	87
19	
20	
21	90
22	22

Now the hash table is sufficiently large to accommodate new insertions.

Q.32 What are the advantages of rehashing ? [JNTU : Part A, Marks 3]

Ans. : Following are the advantages -

1. This technique provides the programmer a flexibility to enlarge the table size if required.
2. Only the space gets doubled with simple hash function which avoids occurrence of collisions.

Q.33 What is extendible hashing ? Explain with suitable example. [JNTU : Part B, Marks 5]

Ans. : The extendible hashing is a dynamic hashing technique in which, if the bucket is overflow, then the number of buckets are doubled and data entries in buckets are re-distributed.

Example of extendible hashing :

In extendible hashing technique the **directory** of pointers to bucket is used. Refer following Fig. Q.33.1

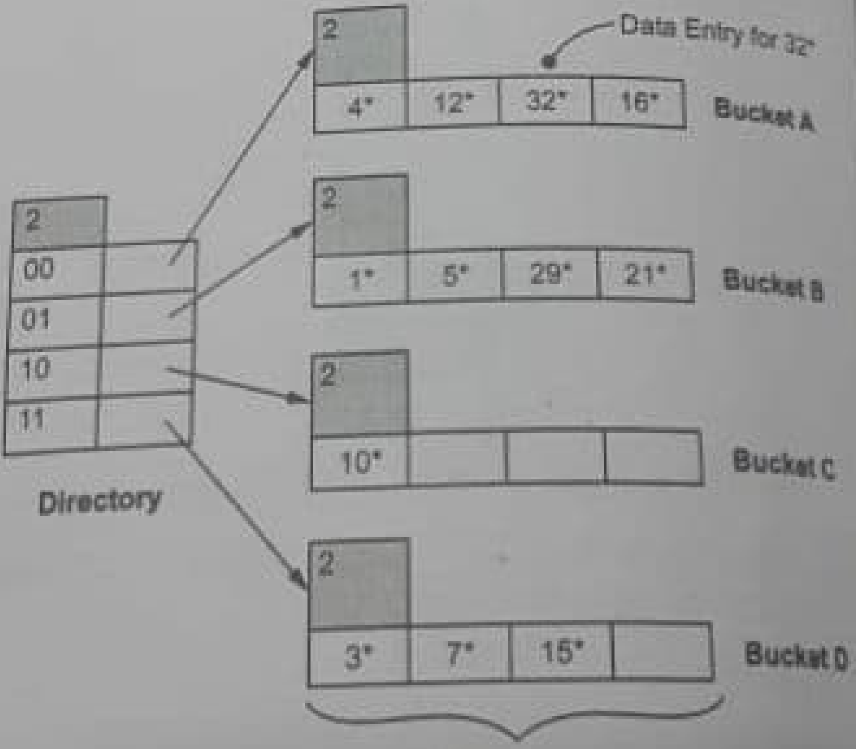


Fig. Q.33.1 Extendible hash file

0	-1	-1
1	131	2
2	21	3
3	31	-1
4	4	-1
5	5	-1
6		
7		
8		
9		

Now next element is 2. As hash function will indicate hash key as 2 but already at index 2. We have stored element 21. But we also know that 21 is not of that position at which currently it is placed.

Hence we will replace 21 by 2 and accordingly chain table will be updated. See the table :

Index	Data	Chain
0	-1	-1
1	131	6
2	2	-1
3	31	-1
4	4	-1
5	5	-1
6	21	3
7	-1	-1
8	-1	-1
9	-1	-1

The value -1 in the hash table and chain table indicate the empty location.

The advantage of this method is that the meaning of hash function is preserved.

2. Open Addressing or Closed Hashing

Various techniques used in open addressing are

1. Linear probing
2. Quadratic probing
3. Double hashing

1. Linear probing

When collision occurs i.e. when two records demand for the same location in the hash table, then the collision can be solved by placing second record linearly down whenever the empty location is found.

For example

Index	data
0	
1	131
2	21
3	31
4	4
5	5
6	61
7	7
8	8
9	

Fig. Q.20.2 Linear probing

In the hash table given in Fig. Q.20.2 the hash function used is number % 10. If the first number which is to be placed is 131 then 131 % 10 = 1 i.e. remainder is 1 so hash key = 1. That means we are supposed to place the record at index 1. Next number is 21 which gives hash key = 1 as 21 % 10 = 1. But already 131 is placed at index 1. That means collision is occurred. We will now apply linear probing. In this method, we will search the place for number 21 from location of 131. In this case we can place 21 at index 2. Then 31 at index 3. Similarly 61 can be stored at 6 because number 4 and 5 are stored before 61.

2. Quadratic probing

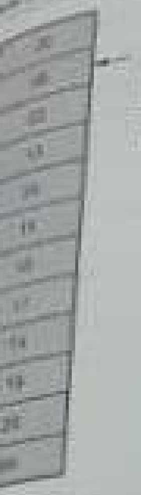
Quadratic probing operates by taking the original hash value and adding successive values of an arbitrary quadratic polynomial to the starting value. This method uses following formula -

$$\boxed{H_i(key) = (Hash(key)+i^2)\%m}$$

where m can be a table size or any prime number.

For example : If we have to insert following elements in the hash table with table size 10 :

37, 90, 55, 22, 11, 17, 49, 87



As 96 % 10 = 6. But index 6 holds a record 16 which is correct for that location. By moving down linearly we get no empty slot. Hence we roll back and get the empty slot at index 1. Hence 96 will be replaced at index 1.

20
22
13
24
15
16
14
17
9
5

Collision occurs at index 4. Hence probing 14 at the next empty slot

At index 7, the 14 is already placed. Hence at next empty slot 17 is placed

26 % 10 = 6. The collision occurs. Hence 26 is placed at next empty slot

96 % 10 = 6. But at location 6, the element 16 is placed. Hence we go linearly down in search of an empty slot. But since table gets full, we may not get an empty slot. Therefore roll back to search an empty slot. At index 1, we can then place 96

64 % 10 = 4. But index 4 contains key element 24. Hence by linear probing, at empty slot 11,

Q.27 Perform the insertion operation using double hashing for the following list.
12, 54,62,45,37,78,89,28,61,49

[JNTU : Part B, Nov.-09, Marks 8]

Ans. : For insertion operation we will use mo function. hence hash function can be defined as

hashfunction = key mod table size

The table size is 10. The table is -

12%10 = 2

54%10 = 4

62%10 = 2 collision occurs.

∴ We will apply double hashing. For applying do hashing choose M. Where M is a primary nu whose value is less than table size.

Set M = 7

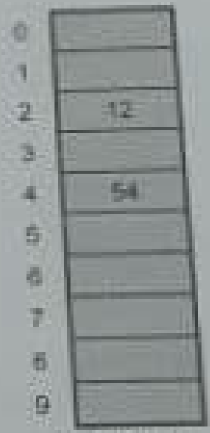


Fig. Q.27.1

$$h_2(\text{key}) = M - (\text{key mod } M)$$
$$= 7 - (62\%7)$$
$$= 7 - 6$$
$$h_2(\text{key}) = 1$$

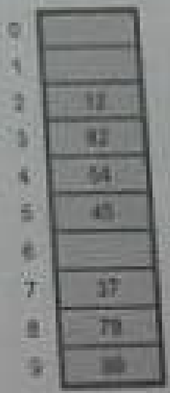


Fig. Q.27.2

That means to insert 62 we have to take 1 jump from 12. Hence 62 will be inserted at index 3.

45%10 = 5

37%10 = 7

78%10 = 8

89%10 = 9

28%10 = 8 collision occurs.

∴ H_2(key) = H_2(28) = 7 – (28%7)

= 7 – 0

= 7

0	
1	61
2	12
3	62
4	54
5	45
6	28
7	37
8	78
9	89

Fig. Q.28.3

If we take 7 jumps from 78, we will reach at 45. Again collision occurs. Hence go linearly down to prob element at empty slot. We will insert it at index 6.

61%10 = 1

49%10 = 9. Collision occurs.

∴ H_2(49) = 7 – (49%7)

= 7

0	49
1	61
2	12
3	62
4	54
5	45
6	28
7	37
8	78
9	89

Fig. Q.28.4

But when 7 jumps we will not get only one empty slot. The remaining in hash table. Hence index 0.

The hash table will be as shown

Q.28 What are the problem quadratic probing

☞ [2012 - Fa

Ans. : One major problem wi probing is that typically not be on the probe sequence. particularly inconvenient sizes, this probe function relatively small number of cycle happen to be full, th cannot be inserted at all ev happen to be empty.

Q.29 Show the resulting 5199, 5344, 6796 and 1 h (x) = x (mod 10).

a) Open addressing ha probing.

b) Open addressing ha functions h2 (x) = 7 - (

☞ [JNT

Ans. : a) Open addressi probing

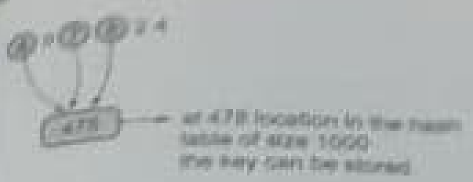
- Insert 3417.

3417%10 = 7

- Insert 3132.

3132%10 = 2

For example : Suppose first, third and fourth digit from left is selected for hash key.


4 5 7 8 2 4
478
at 478 location in the hash table of size 1000 the key can be stored

iv) **Mid Square** : This method works in following steps

1) Square the key

2) Extract middle part of the result. This will indicate the location of the key element in the hash table.

Note that if the key element is a string then it has to be preprocessed to produce a number.

Let key = 3111

$$(3111)^2$$
$$\downarrow$$
$$9\ 6\ \boxed{7\ 8\ 3}\ 2\ 1$$

For the hash table of size of 1000

$H(3111) = 783$

v) **Folding**

There are two folding techniques

i) Fold shift ii) Fold boundary

i) **Fold shift** : In this method the key is divided into separate parts whose size matches with the size of required address. Then left and right parts are shifted and added with the middle part.

ii) **Fold boundary** : In this method the key is divided into separate parts. The leftmost and rightmost parts are folded on fixed boundary and added with the middle part.

For example

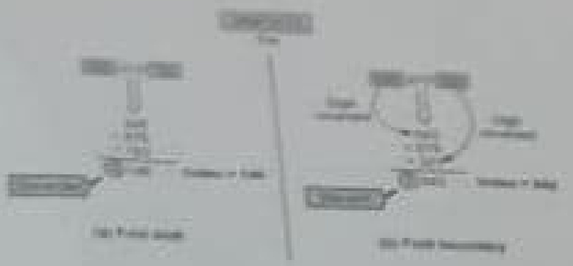


Fig. Q.14.1 Folding techniques

Q.15 Which hash function maintains the record in order of hash field values ?

[JNTU : Part A, Marks 2]

Ans. : The folding method maintains the record in order of hash field values.

Q.16 What are the applications of hashing ?

[JNTU : Part A, Marks 2]

Ans. : Applications of Hashing are -

1. In compilers to keep track of declared variables.
2. For online spelling checking the hashing functions are used.
3. Hashing helps in Game playing programs to store the moves made.
4. For browser program while caching the web pages, hashing is used.

Q.17 What is hash function ? Name two desirable properties of a hash function ?

[JNTU : Part B, Dec.-14, Marks 5]

Ans. : Hash Function - Refer Q.9.

Properties of 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.

... should depend on every bit of ... the hash function that simply ... of a key is not suitable.

...fference between hashing and ...

	Skip List
	Skip lists are used to implement dictionary operations using randomized processes.
	It does not require hash function.
	The sorted data improves the performance of skip list.
	The worst case space requirement for skip list ...
	The skip lists are not that much efficient.
	... space ... is larger for ... list than hashing.

... Techniques

... Dec.-17, Marks 3]

... hash function ... than one record

... vent collision

... Marks 10]

1. Chaining without replacement

In collision handling method chaining is a concept which introduces an additional field with data i.e. chain. A separate chain table is maintained for colliding data. When collision occurs we store the second colliding data by linear probing method. The address of this colliding data can be stored with the first colliding element in the chain table, without replacement.

For example consider elements 131, 3, 4, 21, 61, 6, 71, 8, 9

Index	Data	Chain
0	-1	-1
1	131	2
2	21	5
3	3	-1
4	4	-1
5	61	7
6	6	-1
7	71	-1
8	8	-1
9	9	-1

Fig. Q.20.1 Chaining without replacement

From the example, you can see that the chain is maintained the number who demands for location 1. First number 131 comes we will place at index 1. Next comes 21 but collision occurs so by linear probing we will place 21 at index 2, and chain is maintained by writing 2 in chain table at index 1. similarly next comes 61 by linear probing we can place 61 at index 5 and chain will be maintained at index 2. Thus any element which gives hash key as 1 will be stored by linear probing at empty location but a chain is maintained so that traversing the hash table will be efficient.

2. Chaining with replacement

As previous method has a drawback of loosing the meaning of the hash function, to overcome this drawback the method known as chaining with replacement is introduced. Let us discuss the example to understandsz the method. Suppose we have to store following elements :

131, 21, 31, 4, 5

0	-1	-1
1	131	2
2	21	3
3	31	-1
4	4	-1
5	5	-1
6		
7		
8		
9		

Now next element is 2. As hash function will indicate hash key as 2 but already at index 2. We have stored element 21. But we also know that 21 is not of that position at which currently it is placed.

Hence we will replace 21 by 2 and accordingly chain table will be updated. See the table :

Index	Data	Chain
0	-1	-1
1	131	6
2	2	-1
3	31	-1
4	4	-1
5	5	-1
6	21	3
7	-1	-1
8	-1	-1
9	-1	-1

The value -1 in the hash table and chain table indicate the empty location.

The advantage of this method is that the meaning of hash function is preserved.

2. Open Addressing or Closed Hashing

Various techniques used in open addressing are

1. Linear probing
2. Quadratic probing
3. Double hashing

1. Linear pro...

When collis... for the ... collision ... linearly ...

For examp...

In ... h...

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.18 What is the difference between hashing and skip list ?

Ans. :

Hashing	Skip List
This method is used to carry out dictionary operations using randomized processes.	Skip lists are used to implement dictionary operations using randomized processes.
It is based on hash function.	It does not require hash function.
If the sorted data is given then hashing is not an effective method to implement dictionary.	The sorted data improves the performance of skip list.
The space requirement in hashing is for hash table and a forward pointer is required per node.	The forward pointers are required for every level of skip list.
Hashing is an efficient method than skip lists.	The skip lists are not that much efficient.
Skip lists are more versatile than hash table.	Worst case space requirement is larger for skip list than hashing.

2.3.2 : Collision Resolution Technique

Q.19 What is collision in hashing ?
[JNTU : Part A, Marks 2, Dec.-17, Marks 3]

Ans. : The situation in which the hash function returns the same hash key for more than one record is called collision.

Q.20 What is collision ? Explain different collision resolution techniques with examples.
[JNTU : Part B, Dec.-16, Marks 10]

Ans. : **Collision** - Refer Q.19

Collision Resolution Technique -

1. Open Hashing or Chaining

1. Chaining without replacement

In collision handling method chaining is a concept which introduces an additional field with data i.e. chain. A separate chain table is maintained for colliding data. When collision occurs we store the second colliding data by linear probing method. The address of this colliding data can be stored with the first colliding element in the chain table, without replacement.

For example consider elements 131, 3, 4, 21, 61, 6, 71, 8, 9

Index	Data	Chain
0	-1	[illegible]
1	131	[illegible]
2	21	[illegible]
3	3	[illegible]
4	4	[illegible]
5	61	[illegible]
6	6	[illegible]
7	71	[illegible]
8	8	[illegible]
9	9	[illegible]

Fig. Q.20.1 Chaining without replacement

From the example, you can see that the chain is maintained the number who demands for location 1. First number 131 comes we will place at index 1. Next comes 21 but collision occurs so by linear probing we will place 21 at index 2, and chain is maintained by writing 2 in chain table at index 1. similarly next comes 61 by linear probing we can place 61 at index 5 and chain will be maintained at index 2. Thus any element which gives hash key as 1 will be stored by linear probing at empty location but a chain is maintained so that traversing the hash table will be efficient.

2. Chaining with replacement

As previous method has a drawback of loosing the meaning of the hash function, to overcome this drawback the method known as chaining with replacement is introduced. Let us discuss the example to understandsz the method. Suppose we have to store following elements :

131, 21, 31, 4, 5

going sequentially through a to z.

4. Symbol table used in compiler

Symbol table is a kind of buffer used in compiler for storing the identifiers and constants encountered in the source program(such as C, C++ and so on). Compiler looks up the symbol table as soon as it encounters an identifier or a constant. If it is already defined in the symbol table then the compiler retrieves the values of corresponding identifier. If the identifier is not there in the symbol table then that corresponding entry of identifier is inserted in the symbol table.

2.2 : Skip List Representation

Q.4 What is skip list ?

Ans. :

- Skip lists are made up of a series of nodes connected one after the other. Each node contains a key and value pair as well as one or more references, or pointers, to nodes further along in the list.
- The number of references each node contains is determined randomly. The number of references a node contains is called its node level.
- There are two special nodes in the skip list one is **head node** which is the starting node of the list and **tail node** is the last node of the list.

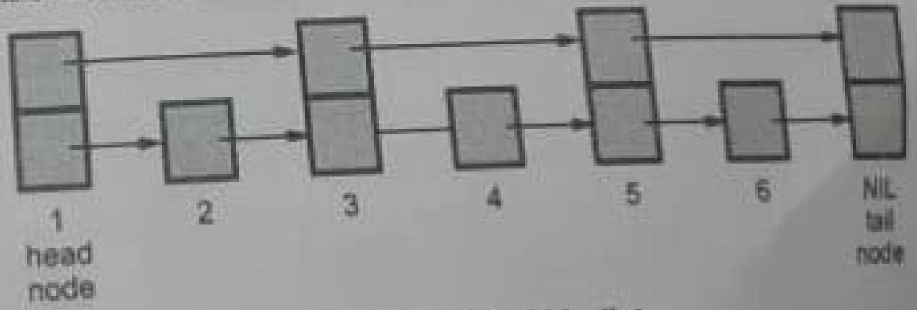
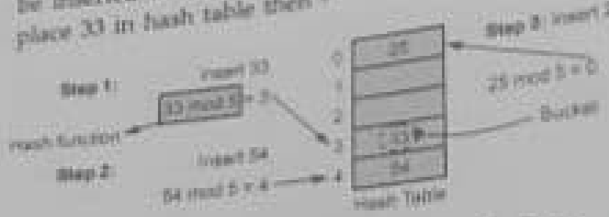


Fig. Q.4.1 Skip list

For example : Consider hash function as **key mod 5**. The hash table of size 5. The key is a value that is to be inserted in Hash table. For instance if we want to place 33 in hash table then -



Q.10 List out the various techniques of hashing.

Ans. : Various techniques of hashing are -

1. Division method
2. Mid square method
3. Multiplicative hash function
4. Digit folding
5. Digit analysis

Q.11 Explain the concept of hash table with an example [JNTU : Dec.-18, Marks 5]

Ans. : Refer Q.9.

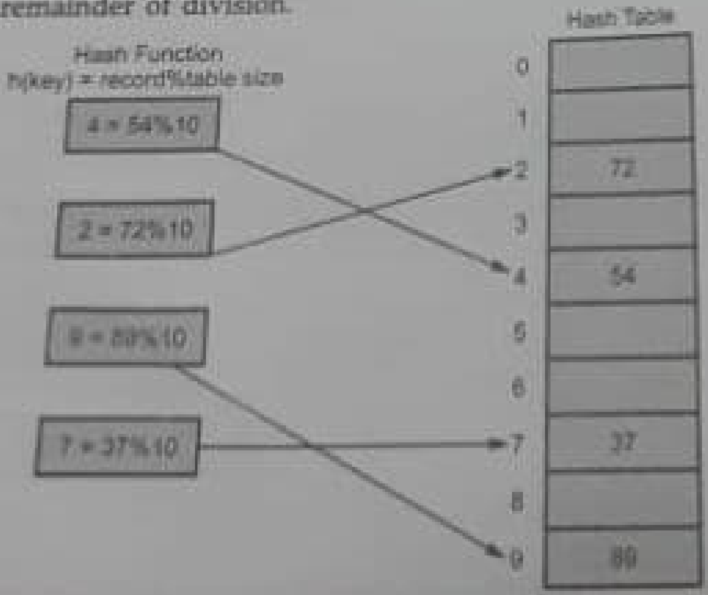
2.3.1 : Hash Functions

Q.12 What is hash function ? [JNTU : May-18, Marks 2]

Ans. : Refer Q.9.

Q.13 What is Division Hash Function ? [JNTU : Part A ,Dec.-16, Marks 3]

Ans. : The hash function depends upon the remainder of division.



Typically the divisor is table length. For example : If the record 54, 72, 89, 37 is to be placed in the hash table and if the table size is 10 then

Q.14 Explain various hashing methods [JNTU : Part

Ans. : **(1) Division Method** - Refer Q.13.

(2) Multiplicative Hash Function

The multiplicative hash function works in following steps

1) Multiply the key 'k' by a constant A in the range 0 < A < 1. Then extract the fractional part of kA.

2) Multiply this fractional part by m and take the floor.

The above steps csan be formulated as

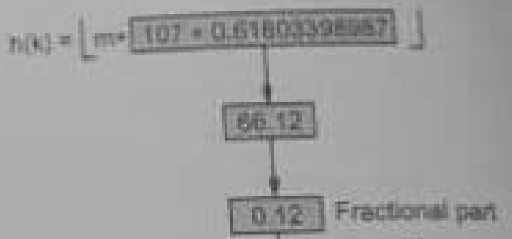
$$h(k) = \lfloor m \underbrace{\{kA\}}_{\text{Fractional part}} \rfloor$$

Donald Knuth suggested to use A = 0.61803398987

Example :

Let key k = 107, assume m = 50.

A = 0.61803398987



h(k) = 50 * 0.12

= 6

h(k) = 6

That means 107 will be placed at index 6 in hash table.

Advantage : The choice of m is not critical

(3) Extraction : In this method some digits are extracted from the key to form the location in hash table.

Data Structures

For example : Su

from left is selecte

(4) Mid Sq

steps

1) Squar

2) Extra

indic

hash

Note th

be prep

Let ke

Page [illegible] 42.

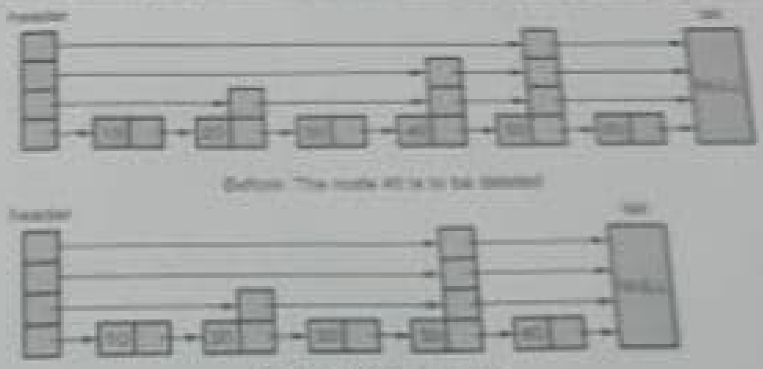
[illegible]

Q.7 Explain deletion of a node from the skip list with some suitable example.

Ans. : The deletion of a node works in two steps -

1) Search the node to be deleted from the skip list

2) On obtaining the desired node, remove the node from the list and adjust the pointers.



Before: The node 40 is to be deleted

After: The node 40 is deleted

Fig. Q.22.1 Deletion operation

Part II : Hash Table

2.3 : Hash Table Representation

Q.8 What is hashing ? [JNTU : Part A, Dec.-16, Marks 2]

Ans. :

(1) Hashing is a technique of storing the elements directly at the specific location in the hash table. The hashing makes use of hash function to place the record at its position.

(2) Using the same hash function the data can be retrieved directly from the hash table.

Q.9 Explain the concept of hash table and hash function [JNTU : Part A, Marks 2]

Ans. : Hash Table -

- Hash Table is a data structure used for storing and retrieving data very quickly. Insertion of data in the hash table is based on the key value. Hence every entry in the hash table is associated with some key. For example for storing an employee record in the hash table the employee ID will work as a key.
- Using the hash key the required piece of data can be searched in the hash table by few or more key comparisons. The searching time is then dependant upon the size of the hash table.

Hash Function -

- Hash function is a function used to place data in hash table.
- Similarly hash function is used to retrieve data from hash table.
- Thus the use of hash function is to implement hash table.

- The skip list is an efficient implementation of dictionary using **sorted chain**. This is bec list each node consists of forward references of more than one node at a time. Following Figu the skip list.

Q.5 Explain the node structure of skip list.

Ans. : Each node in the skip list consists of pair of key and value given by element and a next pointer basically an array of pointers.

```
template <class K, class E>
struct skipNode
{
 typedef pair<const K,E> pair_type;
 pair_type element;
 skipNode<K,E> **next;
 skipNode(const pair_type &New_pair,int MAX):element(New_pair)
{
 next=new skipNode<K,E>*[MAX];
}
};
```

The individual node will look like this -

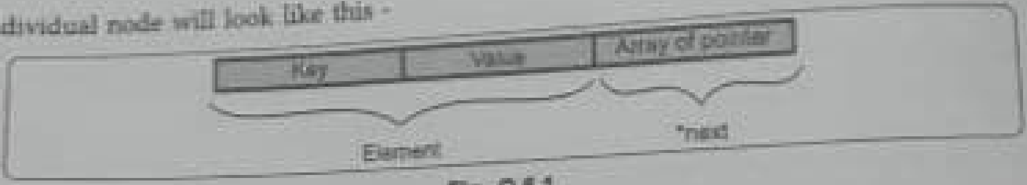


Fig. Q.5.1

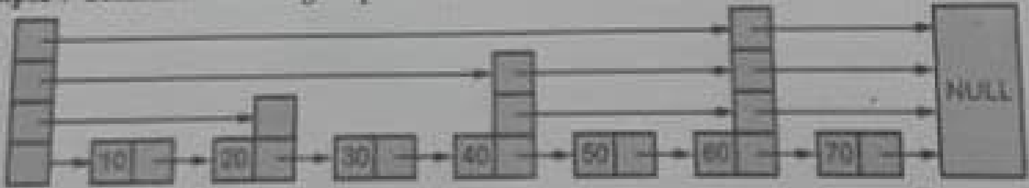
2.2.1 : Operations on Skip List

Q.6 Explain insertion of a node in skip list with some suitable example.

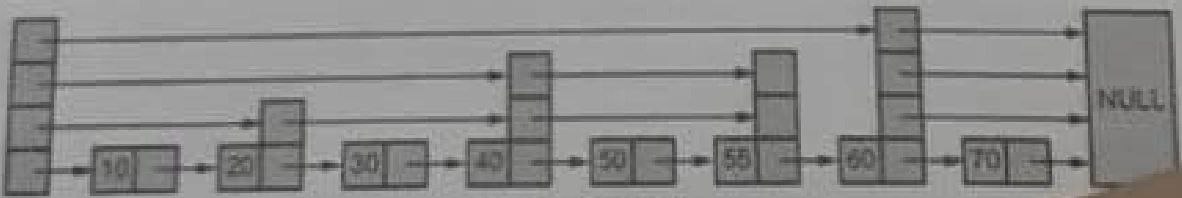
Ans. :

- While inserting a new node in the skip list, it is necessary to find its appropriate location in the skip lists. Note that after inserting a new node in the skip list, the sorted order need to be maintained.
- The level of the new node is determined randomly.

For example : Consider following skip list in which we want to insert 55.



Before insertion of 55



After insertion of 55

6. Repeat the step 5, till the queue is not empty.

7. Stop.

C Code

```
void bfs(int v1)
{
    int v2;
    visit[v1] = TRUE;
    front = rear = -1;
    Q[++rear] = v1;
    while ( front != rear )
    {
        v1 = Q[++front];
        printf("\n%d",v1);
        for ( v2 = 0; v2 < n; v2++)
        {
            if ( g[v1][v2] == TRUE && visit[v2] == FALSE )
            {
                Q[++rear] = v2;
                visit[v2] = TRUE;
            }
        }
    }
}
```

Q.14 What is Depth First Search technique? ☞ [JNTU : Part A, Marks 3]

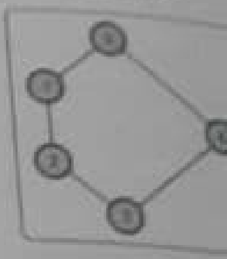
Ans. :
- In depth first search traversal we start from one vertex and traverse the path as deeply as we can go. When there is no vertex further, we traverse back and search for unvisited vertex.
- An array is maintained for storing the visited vertex.
- The DFS will be (if we start from vertex 0) 0 - 1 - 2 - 3 - 4
- The DFS will be (if we start from vertex 3) 3 - 4 - 0 - 1 - 2

Q.15 Implement DFS algorithm.

Ans. :

```
void Dfs(int v1)
{
    int v2;

    printf("\n%d",v1);
    v[v1] = TRUE;
    for ( v2 = 0; v2 < n; v2++)
        if ( g[v1][v2] == TRUE && v[v2] == FALSE )
        Dfs(v2);
}
```

For example



☞ [JNTU : Part B, Dec.

```
                first = first->next;
            first->next = New;
          }
        }
      printf("\nWant to add more edges?(y/n)");
      ans=getche();
    }while(ans=='y');
  }
```

4.1.2 : Traversal Methods

Q.12 Define a graph. List different graph traversal techniques ☞[JNTU : Part A, Dec.-16, Marks 2]

Ans. : **Definition of Graph** - Refer Q.1

Different traversal technique are -

1. Depth First Search (DFS)
2. Breadth First Search (BFS)

Q.13 Write an algorithm/pseudo code to implement BFS. ☞[JNTU : Part B,May-15, Dec.-16, 17, Marks 5]

Ans. :
- In BFS we start from some vertex and find all the adjacent vertices of it. This process will be repeated for all the vertices so that the vertices lying on same breadth get printed.
- For avoiding repetition of vertices, we maintain array of **visited** nodes.
- A queue data structure is used to store adjacent vertices.

Algorithm :

1. Create a graph. Depending on the type of graph i.e. directed or undirected set the value of the flag as either 0 or 1 respectively.
2. Read the vertex from which you want to traverse the graph say V_i.
3. Initialize the visited array to 1 at the index of V_i.
4. Insert the visited vertex V_i in the queue.
5. Visit the vertex which is at the front of the queue. Delete it from the queue and place its adjacent nodes in the queue.

h a g c f

a, b, e, g, c, f, d

...n example

: Part B, Marks 5]

...ing V.

... find all the connected components of a graph. ... BFS algorithm loops through its vertices, ... a new breadth first search whenever the ... reaches a vertex that has not already been ... included in previously found connected component.

... What is the difference between DFS and BFS ... example ?

☞ [JNTU : Part A, Marks 3]

Ans. :

Sr. No.	Breadth First Search	Depth First Search
1	BFS is simple to implement.	DFS is complex to implement as it may suffer from infinite loop problem.
2	BFS will perform poor is for large numbers of vertices in graph.	DFS will perform better in case of large complex graph.
3	BFS requires more memory.	DFS requires less memory.
4	BFS will find the shortest path if the weight on the links are uniform.	DFs can not obtain shortest path.
5	BFS is not useful in sorting.	DFS is used in sorting.
6	This algorithm works in single stage. The visited vertices are removed from the queue and then displayed at once.	This algorithm works in two stages - in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.

Q.21 What is the time complexity of DFS traversal as a vertex simple graph that is represented with adjacent matrix structure ?

☞ [JNTU : Part A, Nov.-15, Marks 3]

Ans. : The time complexity of DFS traversal when implemented using adjacency matrix structure is $O(n^2)$ because the DFS routine is recursively called for each row and column values.

4.1.3 : Applications of Graph

Q.22 What are applications of graph?

☞ [JNTU : May-18, Dec-18, Marks ...

Ans. : The graph theory is used widely ... computer science very widely. The application ... graph theory are :

1. In computer networking such as Local ... Network (LAN), Wide Area Networking ... internetworking the graph is used.
2. In telephone cabling graph theory is ... used.
3. In job scheduling algorithms the graph ...

PART II SORTING

4.2 : Introduction to Sorting

Q.23 Define sorting and list sorting me...

☞ [JNTU : Part A, May-15, Dec...

Ans. : Sorting is a technique of ... elements in ascending or descending ... sorting methods are

1. Insertion sort
2. Quick Sort
3. Selection Sort
4. Merge Sort
5. Heap Sort.

Q.24 What is sorting ? What is s...

☞ [JNTU : Part ...

Ans. : Sorting is a technique o... in ascending or descending orde... Searching is a technique of find... from the list of elements.

Q.25 What is the need for so...

☞ [...

Ans. : The sorting is useful ...

1. Searching the desired da...
2. Responding to the quer...

Q.26 What is the meaning ...

Ans. : Sort key is a fie...

Q.19 How will you search a pattern in suffix trie ? ☞ [JNTU : Part B, Marks 5]

Ans. :

1. Start from root of suffix trie and perform following for every character.

   i) For current character in pattern if there is an edge in suffix trie from current node then follow the edge.

   ii) If there is no edge from current node for the current character then print the message "Pattern does not exist"

2. If all characters are processed and we reach to $ in suffix trie then print "Pattern exists".

For example - We can find the pattern "aaba" as follows.

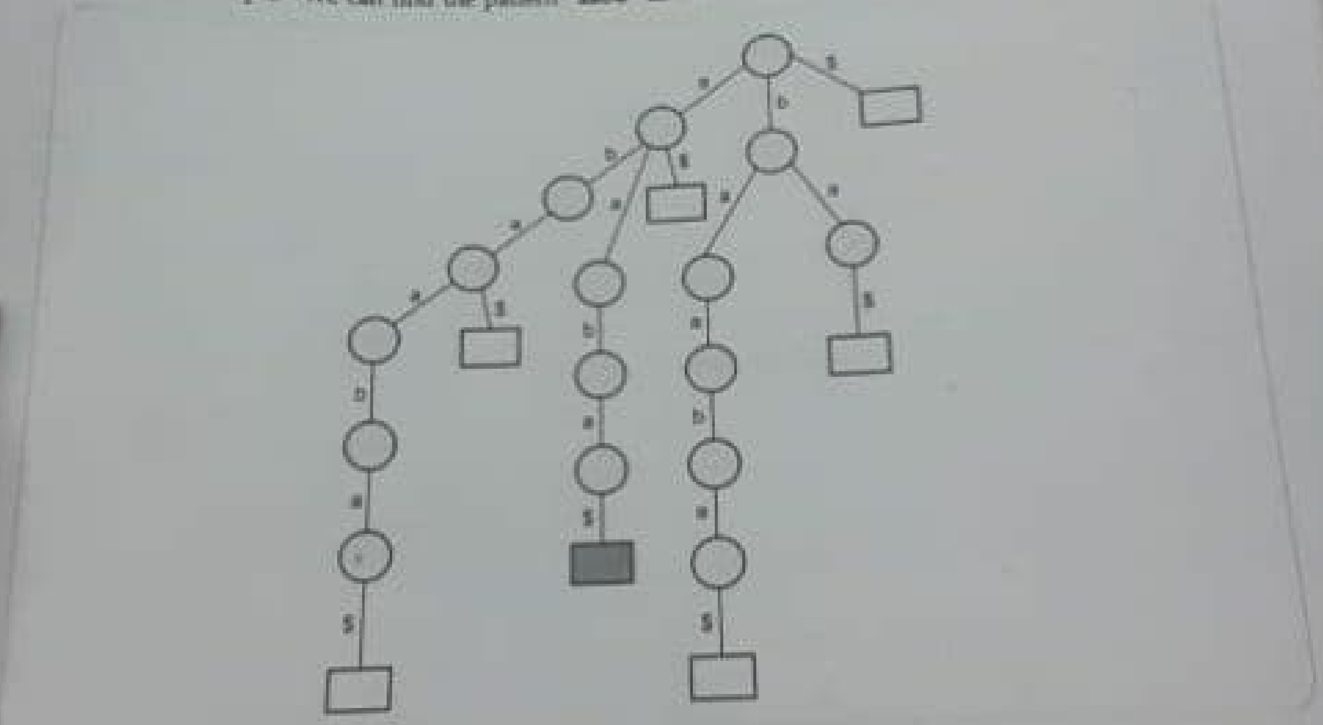


Fig. Q.19.1

Q.20 What are the applications of suffix trie ? ☞ [JNTU : Part A, Marks

Ans. :

1) For finding desired substring.
2) For finding longest common substring.
3) For finding shortest pattern in the given string.
4) To count number of occurrences of particular word.
5) To check if particular suffix exists or not.

– prefix_table[pattern_index_of_unmatched character – 1]

= 10 – prefix_table[5]

= 10 – 2 = 8

That means shift pattern starting at index 8.

Step 5 :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
								✓	✓	×												
								a	b	c	d	a	b	c	y							

The Text[10] is not matching with pattern[2].

Hence new position = 10 – prefix_table[1]

= 10 – 0 = 10

That means shift pattern at starting index 10.

Step 6 :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
										×												
										a	b	c	d	a	b	c	y					

The Text[10] is not matching with pattern[0].

Hence shift pattern by one position.

Step 7 :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
											✓	✓	✓	✓	✓	✓	✓	×				
											a	b	c	d	a	b	c	y				

The Text[18] is not matching with pattern[7].

Hence new position = 18 – prefix_table[6]

= 18 – 3

= 15

That means shift pattern at starting position 15.

5.4 : Suffix Tries

Q.18 What is suffix trie ? Give an example for construction of suffix trie.

[JNTU : Part B, Marks 5]

Ans. : Suffix Trie : A suffix trie is a compressed trie with suffixes.

Steps for construction of Suffix Trie :

Step 1 : Generate all the suffixes for the given word.

Step 2 : Consider all suffixes and build a compressed trie.

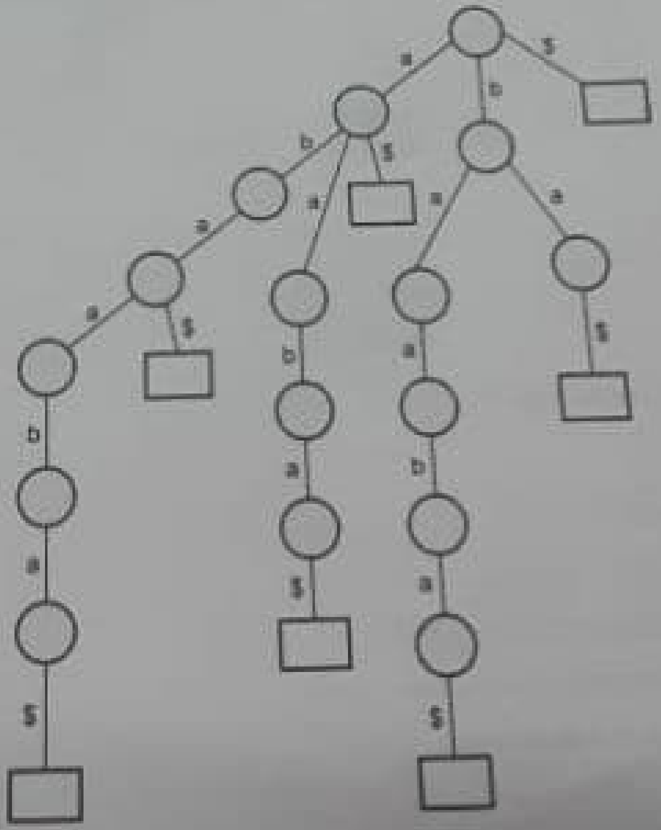
For example : Consider the string

S = abaaba$

Step 1 : We will write all the suffixes of given string

a b a a b a $
b a a b a $
a a b a $
a b a $
b a $
a $
$

Step 2 : The suffix trie is constructed as follows :



Step 1 : We will construct prefix_table or failure function table for given pattern as follows.

0	1	2	3	4	5	6	7
a	b	c	d	a	b	c	y
0	0	0	0	1	2	3	0

prefix-table

Step 2 : Now start matching search for pattern against the text with the help of prefix_table

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
✓	✓	✓	×																			
a	b	c	d	a	b	c	y															

The pattern[3] is not matching with Text[3]. Hence we find position using formula :

text_index_of_unmatched_character

− prefix_table[pattern_index − 1]

= 3 − prefix_table[3 − 1]

= 3 − 0

= 3

That means shift pattern starting at index 3.

Step 3 :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
			×																			
			a	b	c	d	a	b	c	y												

As pattern[0] is not matchine with Text[3]. Hence simply shift pattern by one position.

Step 4 :

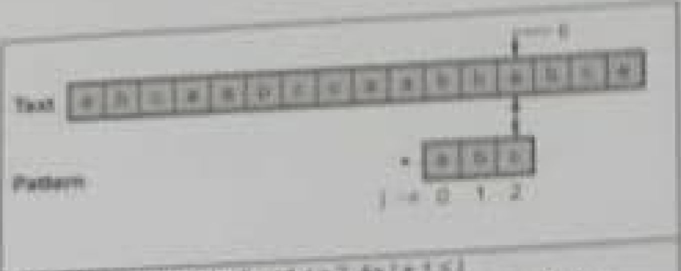
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
				✓	✓	✓	✓	✓	✓	×												
				a	b	c	d	a	b	c	y											

The Text[10] is not matching with pattern[6].

Apply formula

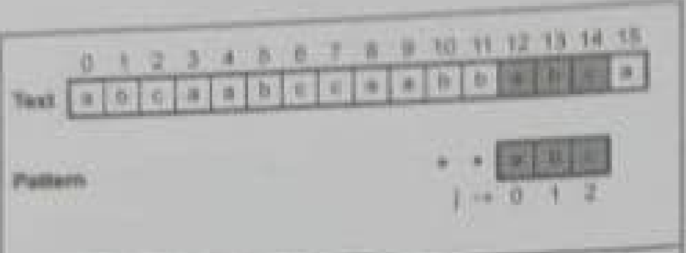
text_index_of_unmatched_character

Step 6 :



We will find l = last (a) = 0 and j = 2. As l + 1 ≤ j i.e. 0 + 1 ≤ 2. We will shift the pattern by j − l i.e. 2 − 0 = 2

Step 7 :



The match is found at index 12 in text array.
Thus we have search the pattern at index 0, 4 and 12 in text array.

The worst case time complexity is O(n).

5.1.3 : Knuth Morris Pratt Algorithm

Q.8 What is the principle idea behind the Knuth Morris Pratt algorithm ?

Ans. : The basic idea behind this algorithm is to build a prefix array. Some times this array is also called π array. This prefix array is built using the prefix and suffix information of pattern. The overlapping prefix and suffix is used in K-M-P algorithm.

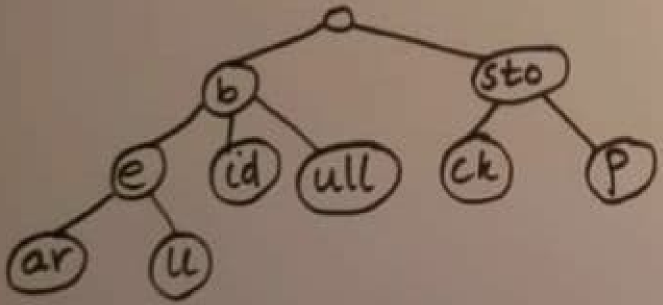
Q.9 Write the Knuth Morris Pratt pattern matching algorithm and apply the same to search the pattern 'abcdabcy' in the text 'abcxabcdabxabcdabcdabcy'.

Ans. : Knuth Morris Pratt Algorithm

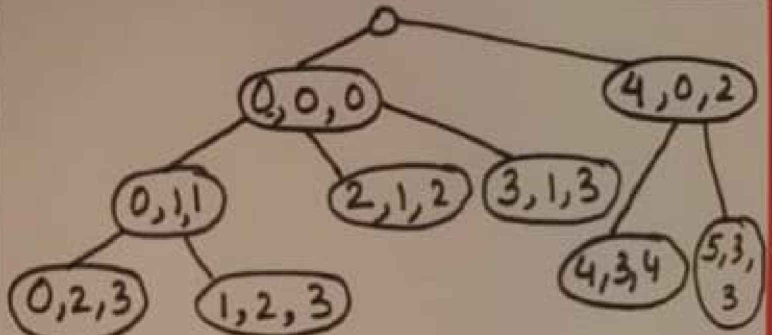
- In the pattern matching algorithms, we often compare the pattern characters that do not match in the text, and on occurrence of mismatch we simply throw away the information and restart the comparison, for another set of characters from the text.
- Thus again and again with next incremental position of text, the characters from pattern are matched. This ultimately reduces the efficiency of pattern matching algorithm. Hence the Knuth-Morris-Pratt algorithm came up which avoids the repeated comparison of characters.
- This algorithm is named after the scientists Knuth, Morris and Pratt.
- The basic idea behind this algorithm is to build a prefix_table. Sometimes this array is also called π array or failure function table.
- This prefix_table is built using the prefix and suffix information of pattern.
- The overlapping prefix and suffix is used in K-M-P algorithm.

Compressed Tries

	0	1	2	3	4
S[0] =	b	e	a	r	
S[1] =	b	e	l	l	
S[2] =	b	i	d		
S[3] =	b	u	l	l	
S[4] =	s	t	o	c	k
S[5] =	s	t	o	p	



Node (i, j, k)

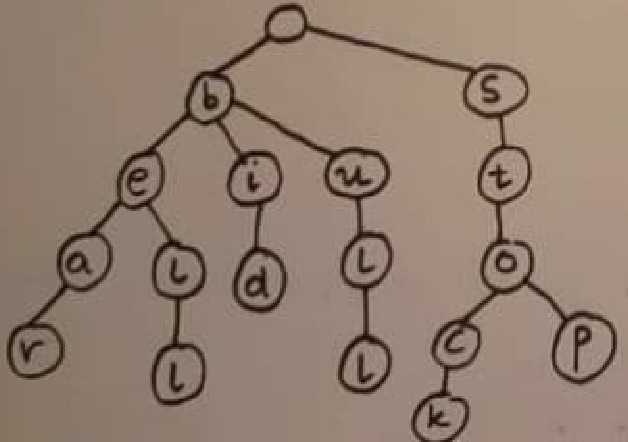


i - Index of S
j - Start
k - end

Tries

- →Tree
- →stores a set of strings
- →every node (except root) will store a letter in the alphabet

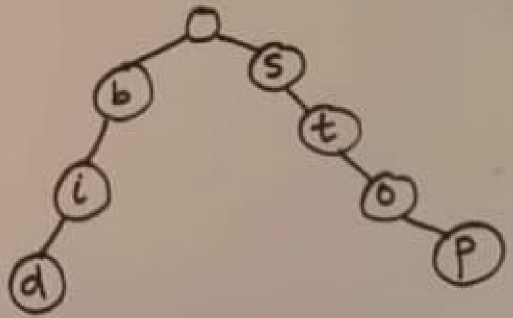
Ex: S = { bear, bell, bid, bull, stock, stop }



NOTE: For standard trie, No word in S should be the prefix of the other.

Worst Case

S = { bid, stop }



n - no. of letters in all the strings of S

O(n)

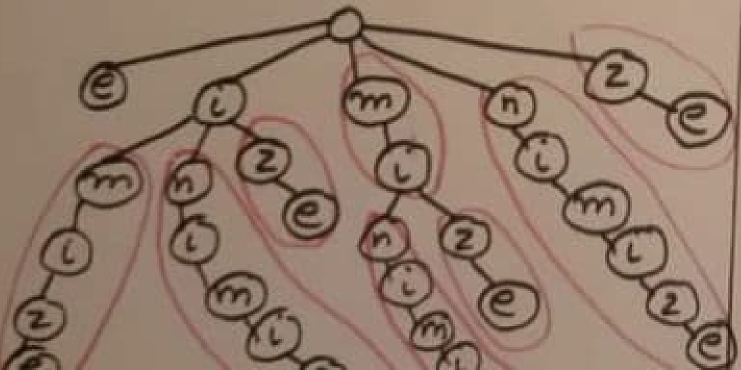
Suffix Tries

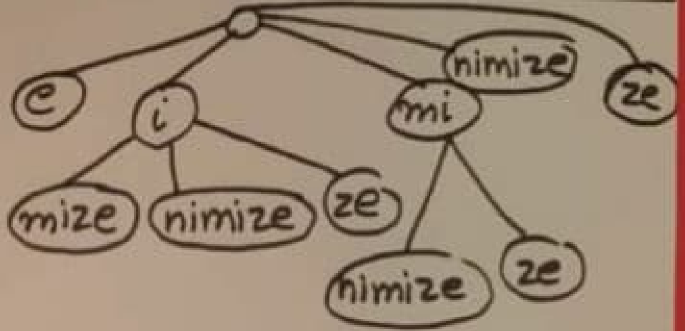
Ex: minimize

Suffixes:

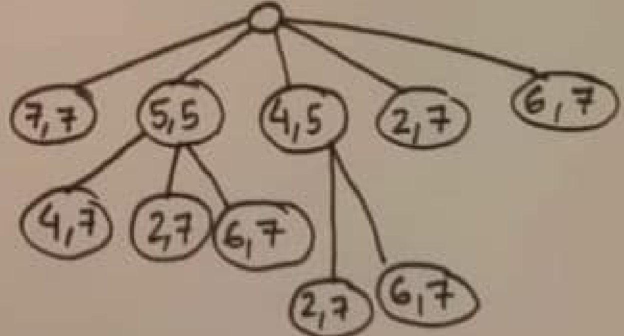
e
ze
ize
mize
imize
nimize
inimize
minimize

S

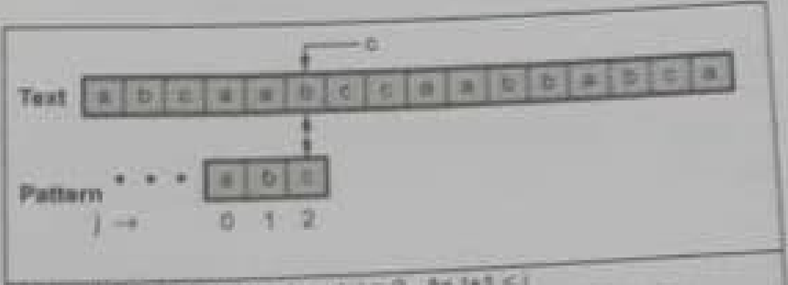




0 1 2 3 4 5 6 7
minimize

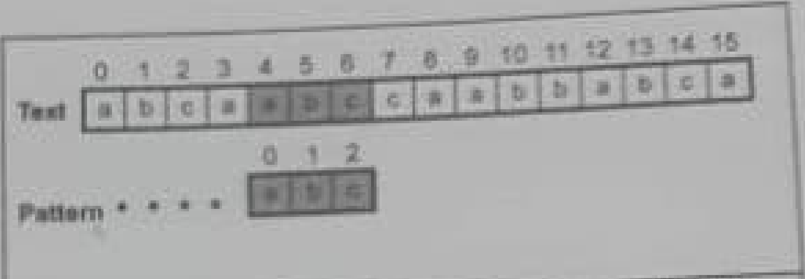


Step 2 :



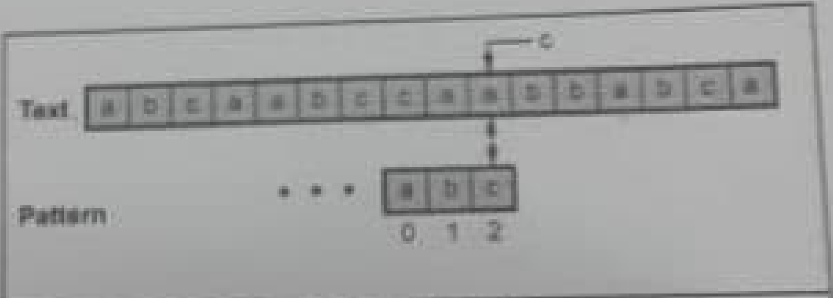
We will find l = last (c) = 1 and j = 2. As l+1 ≤ j
i.e. 1 + 1 = 2. We shift the pattern by j–l i.e. 2–1 = 1 Position

Step 3 :



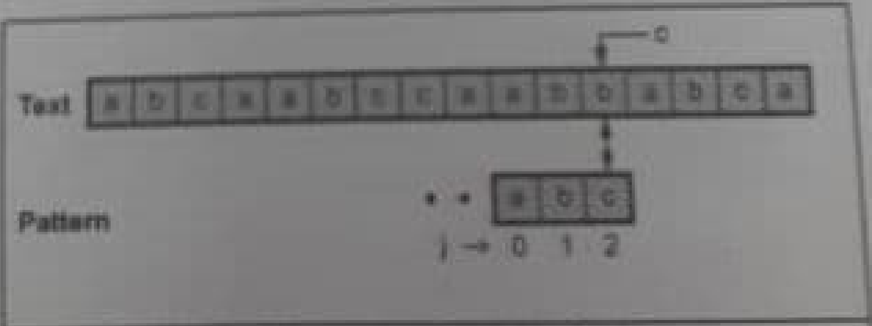
A match is found at index 4 in text. Hence we will shift the pattern by 3 positions.

Step 4 :



We will find l = last (a) = 0, j = 2. As 1 + l ≤ j
i.e. 1 + 0 ≤ 2 we shift the pattern by j – l i.e. 2 – 0 = 2.

Step 5 :



We will find l = last (b) = 1 and j = 2. As l + 1 ≤ j
i.e. 1+1 ≤ 2 we will shift the pattern by j – l i.e. 2 – 1 = 1.

5.1.2 : Boyer Moore Algorithm

Explain Boyer Moore Pattern Matching Algorithm with suitable example.

The Boyer-Moore algorithm was invented by **Boyer** and **Moore**. Hence is the name. The Boyer-Moore scans characters of the search pattern from right to left. If a match is not found then a shift is made by some set of characters. This algorithm is also called "looking glass heuristic".

Example - Refer Q.7.

Find the Boyer-Moore string matching algorithm for the given pattern against the text. abcaabccaabbabca Pattern abc.

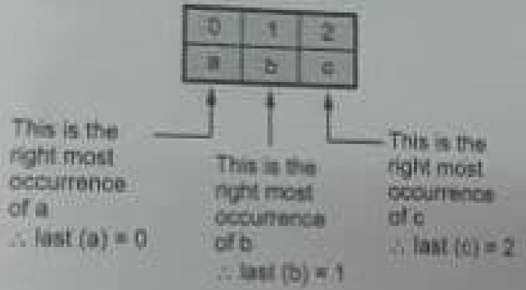
Consider the string "abcaabccaabbabca" as text and "abc" as the pattern to match against the text.

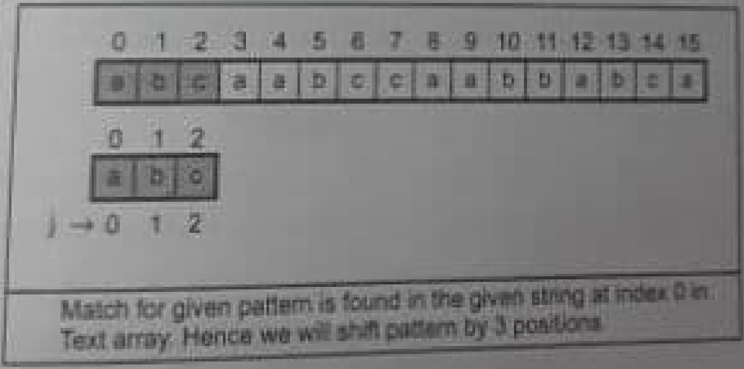
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	a	a	b	c	c	a	a	b	b	a	b	c	a

0	1	2
a	b	c

We will first build the last table using following steps.

1. Arrange the characters of the pattern in an array starting from index 0.
2. Find the rightmost position of every new character.


0
1
2
a
b
c
This is the right most occurrence of a
∴ last (a) = 0
This is the right most occurrence of b
∴ last (b) = 1
This is the right most occurrence of c
∴ last (c) = 2



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	a	a	b	c	c	a	a	b	b	a	b	c	a

0	1	2
a	b	c

j → 0 1 2

Match for given pattern is found in the given string at index 0 in Text array. Hence we will shift pattern by 3 positions

```
        while(j<=high)
        {
            temp[k]=A[j];
            j++;
            k++;
        }
        //copy the elements from temp array to A
        for(k=low;k<=high;k++)
            A[k]=temp[k];
}
```

Reached at the end of left sub lis
elements of right sub list are rem
Then copy the remaining elem
right sub list to **temp**

Fill in the Blanks for Mid Term

Graph is a collection of ____________.

Boyre Moore Algorithm

Q.52 Write a C function for merge sort. ☞ [JNTU : Part B, Dec.-17, Marks

Ans. :

```
    void MergeSort(int low,int high)
    {
        int mid;
        if(low < high)
        {
          mid = (low+high)/2;//split the list at mid
          MergeSort(low,mid);//first sublist
          MergeSort(mid+1,high);//second sublist
          Combine(low,mid,high);//merging of two sublists
        }


 }
 /* This function is for merging the two sublists
 */
 void Combine(int low,int mid,int high)
 {
 int i,j,k;
      int temp[10];
      k=low;
      i=low;
      j=mid+1;
      while(i <= mid && j <= high)
      {
        if(A[i]<=A[j])
        {
          temp[k]=A[i];
          i++;
          k++;
        }
        else
         {
         temp[k]=A[j];
         j++;
         k++;
        }
 }
 while(i<=mid)
 {
    temp[k]=A[i];
    i++;
    k++;
}
```

We compare the elements from left sub list and right sub list. If the elements in the **left sub list is lesser than the** elements in the **right sub list** then copy that smaller element of **left sub list to temp** array

We compare the elements from left sub list and right sub list. If the elements in the **right sub list is lesser than the** elements in the **left sub list** then copy that smaller element of **right sub list to temp** array

Reached at the end of right sub list and elements of left sub list are remaining. Then copy the remaining elements of left sub list to **temp**