TABLE OF CONTENTS

Unit - I

Chapter - 1 Introduction to Data Structures (1 - 1) to (1 - 56)

1.1 Abstract Data Types ..... 1 - 1

1.2 Linear List ..... 1 - 1

1.2.1 Singly Linked List Implementation ..... 1 - 2

1.2.2 Doubly Linked List ..... 1 - 11

1.2.3 Circular Linked List ..... 1 - 13

1.3 Stack ..... 1 - 17

1.3.1 Array Representation of Stack ..... 1 - 18

1.3.2 Operations on Stack ..... 1 - 18

1.3.3 Linked List Representation of Stack ..... 1 - 23

1.3.4 Applications of Stack ..... 1 - 28

1.4 Queues ..... 1 - 35

1.4.1 Array Representation of Queue ..... 1 - 36

1.4.2 Linked List Representation of Queue ..... 1 - 44

Fill in the Blanks with Answers for Mid Term Exam ..... 1 - 47

Multiple Choice Questions with Answers for Mid Term Exam ..... 1 - 48

Unit - II

Chapter - 2 Dictionaries and Hash Table Representation (2 - 1) to (2 - 19)

2.1 Linear List Representation ..... 2 - 1

2.2 Skip List Representation ..... 2 - 2

2.2.1 Operations on Skip List ..... 2 - 3

2.3 Hash Table Representation ..... 2 - 4

2.3.1 Hash Functions ..... 2 - 5

2.3.2 Collision Resolution Technique ..... 2 - 7

Fill in the Blanks with Answers for Mid Term Exam ..... 2 - 1[illegible]

Multiple Choice Questions with Answers for Mid Term Exam ..... 2 - 1[illegible]

Unit - III

Chapter - 3 Search Trees (3 - 1) to (3 - 34)

3.1 Binary Search Trees ..... 3 - 1

3.2 AVL Trees ..... 3 - 11

3.3 Red Black Trees ..... 3 - 20

Fill in the Blanks with Answers for Mid Term Exam ..... 3 - 32

Multiple Choice Questions with Answers for Mid Term Exam ..... 3 - 32

Unit - IV

Chapter - 4 Graphs and Sorting (4 - 1) to (4 - 42)

4.1 Introduction to Graphs ..... 4 - 1

4.1.1 Implementation Methods ..... 4 - 3

4.1.2 Traversal Methods ..... 4 - 7

4.1.3 Applications of Graph ..... 4 - 15

4.2 Introduction to Sorting ..... 4 - 15

4.2.1 Heap Sort ..... 4 - 16

4.2.2 External Sorting ..... 4 - 34

Fill in the Blanks with Answers for Mid Term exam ..... 4 - 38

Multiple Choice Questions with Answers for Mid Term Exam ..... 4 - 38

Unit - V

Chapter - 5 Pattern Matching and Tries (5 - 1) to (5 - 19)

5.1 Pattern Matching Algorithms ..... 5 - 1

Scanned by CamScanner

5.1.1 Brute Force Algorithm . . . . . 5 - 1

5.1.2 Boyer Moore Algorithm . . . . . 5 - 3

5.1.3 Knuth Morris Pratt Algorithm . . . . . 5 - 5

Standard Tries. . . . . 5 - 13

Compressed Tries . . . . . 5 - 16

Suffix Tries. . . . . 5 - 17

Fill in the Blanks with Answers for Mid Term Exam . . . . . 5 - 19

Multiple Choice Questions with Answers for Mid Term Exam . . . . . 5 - 19

ved Model Question Paper (M - 1) to (M - 2)

Scanned by CamScanner

UNIT - I

1 Introduction to Data Structures

1.1 : Abstract Data Types

Q.1 What is abstract data type ? What are all not concerned in an ADT ?

☞[JNTU : Part A, Marks 2]

Ans. :

1) The abstract data type is a triple of D i.e. set of axioms, F-set of functions and A - Axioms in which only what is to be done is mentioned but how is to be done is not mentioned.
2) Thus ADT is not concerned with implementation details.

Q.2 State the advantages of abstract data type.

☞[JNTU : Part A, Marks 2]

Ans. : Following are the advantages of abstract data type -

1. Abstract data type is useful for representing all possible operations of the data structure.
2. The implementation details can be hidden in ADT.
3. The implementation details can be changed later on during maintenance and bug fixing when the ADT is available.
4. ADT helps to define the program definition in general form.

Q.3 Write ADT for arrays.

☞[JNTU : Part B, Marks 5]

Ans. :

AbstractDataType Array

{

Instances : An array A of some size, index i and total number of elements in the array n.

Operations :

1. Store () - This operation stores the desired elements at each successive location.
2. display () - This operation displays the elements of the array.

}

1.2 : Linear List

Q.4 What is a linked list ?

☞[JNTU : Part A, Marks 2]

OR Define node in singly linked list.

☞[JNTU : Part A, Dec.-17, Marks 2]

Ans. : A linked list is a set of nodes where each node has two fields 'data' and 'link'.

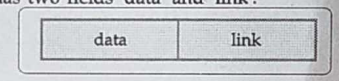


Fig. Q.4.1 Structure of node

The data field is used to store actual piece of information and link field is used to store address of next node.

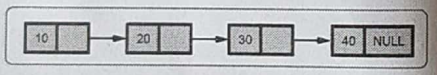


Fig. Q.4.2

Q.5 What are the pitfalls encountered in singly linked list ?

☞[JNTU : Part A,Marks 2]

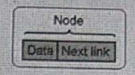
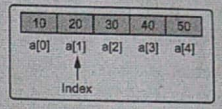
Ans. : Following are the pitfalls encountered in singly linked list

1. The singly linked list has only forward pointer and no backward link is provided. Hence the traversing of the list is possible only in one direction. Backward traversing is not possible.
2. Insertion and deletion operations are less efficient because for inserting the element at desired position the list needs to be traversed. Similarly, traversing of the list is required for locating the element which needs to be deleted.

Q.6 Differentiate between arrays and lists.

☞[JNTU : Part B, Marks 5]

Ans. :

Sr. No.	Linked List	Array
1.	The linked list is a collection of nodes and each node is having one data field and next link field. For example 	The array is a collection of similar types of data elements. In arrays the data is always stored at some index of the array. For example 
2.	Any element can be accessed by sequential access only.	Any element can be accessed randomly i.e. with the help of index of the array.
3.	Physically the data can be deleted.	Only logical deletion of the data is possible.
4.	Insertions and deletion of data is easy.	Insertions and deletion of data is difficult.
5.	Memory allocation is dynamic. Hence developer can allocate as well as deallocate the memory. And so no wastage of memory is there.	The memory allocation is static. Hence once the fixed amount of size is declared then that much memory is allocated. Therefore there is a chance of either memory wastage or memory shortage.

Q.7 What is the advantage of linked list over arrays ?

☞[JNTU : Part A, Marks 3]

Ans. :

- The linked list makes use of the dynamic memory allocation. Hence the user can allocate or de allocate the memory as per his requirements.
- On the other hand, the array makes use of the static memory location. Hence there are chances of wastage of the memory or shortage of memory for allocation.

1.2.1 : Singly Linked List Implementation

Q.8 Give the ADT for linked list.

Ans. : In ADT the implementation details are hidden. Hence the ADT will be

Abstract DataType List

{

Instances : List is a collection of elements which are arranged in a linear manner.

Operations : Various operations that can be carried out on list are -

1. Insertion : This operation is for insertion of element in the list.
2. Deletion : This operation removed the element from the list.
3. Searching : Based on the value of the key element the desired element can be searched.
4. Modification : The value of the specific element can be changed without changing its location.
5. Display : The list can be displayed in forward or in backward manner.

}

Scanned by CamScanner

Q.9 How will you represent the linked list in C ? ☞ [Part A, Marks 3]

Ans. :

```
typedef struct node
    {
        int data; /* data field */
        struct node * next; /* link field */
    } SLL;
```

while declaring C structure for a linked list -

- Declare a structure in which two members are there i.e. data member and next pointer member.
- The data member can be character or integer or real kind of data depending upon the type of the information that the linked list is having.
- The 'next' member is essentially of a pointer type. And the pointer should be of structure type. Because the 'next' field holds the address of next node. And each node is basically a structure consisting of 'data' and 'next'.

Q.10 Write down the steps to modify a node in linked lists. ☞ [JNTU : Part A, Marks 3]

Ans. :

1. Enter the position of the node which is to be modified.
2. Enter the new value for the node to be modified.
3. Search the corresponding node in the linked list.
4. Replace the original value of that node by a new value.
5. Display the message as "The node is modified".

Q.11 Write a Pseudocode for creation of linked list. ☞ [JNTU : Part B, Marks 5]

Ans. :

```
node*  create()
{
node *temp, *New, *head;
    int   val, flag;
    char ans='y';
    node *get_node();
    temp = NULL ;
    flag = TRUE; /* flag to indicate whether a new node is created for the first time or not */
    do
    {
        printf("\nEnter the Element :");
        scanf("%d", &val);
        /* allocate new node */
        New =get_node();
        if ( New == NULL )
            printf("\nMemory is not allocated");
        New -> data  = val ;
        if (flag==TRUE) /* Executed only for the first time */
        {
```



```
            head = New;
            temp=head; /*head is a first node in the  SLL*/
            flag = FALSE;
        }
        else
        {
            /* temp keeps track of the most recently created node */
            temp ->next = New;
            temp = New;
        }
        printf("\n Do you Want to enter more elements?(y/n)");
        ans=getche();
        }while(ans=='y');
        printf("\nThe Singly Linked List is created\n");
        getch();
        clrscr();
        return head;
    }
    node *get_node()
    {
        node *temp;
        temp=( node *) malloc( sizeof(node) );
        temp->next=NULL;
        return temp;
    }
```

Q.12 Write a C code to insert a node in the linked list as :
(a) a header node (b) intermediate node (c) Last node

Ans. :

(a) header node

```
node *insert_head(node *head)
{
node *New,*temp;
    New=get_node();
    printf("\n Enter The element which you want to insert");
    scanf("%d",&New->data);
    if(head==NULL)
        head=New;
    else
    {
        temp=head;
        New->next=temp;
        head=New;
    }
    return head;
}
void insert_last(node *head)
```


Scanned by CamScanner

```
{
    node *New,*temp;
    New=get_node();
    printf("\n Enter The element which you want to insert");
    scanf("%d",&New->data);
    if(head==NULL)
      head=New;
    else
    {
      temp=head;
      while(temp->next!=NULL)
        temp=temp->next;
      temp->next=New;
      New->next=NULL;
    }
}
```

Q.13 Write a program to delete an element from single linked list [JNTU : Part B, Dec 17, Marks 5]

Ans. :

```
void dele(node **head)
{
    node *temp,*prev ;
    int  key;
    temp = *head;
    if ( temp == NULL )
    {
      printf("\nThe list is empty\n");
      getch();
      clrscr();
      return
    }
    clrscr();
    printf("\nEnter the Element you want to delete: ");
    scanf("%d", &key);
    temp = search(*head,key);
    if ( temp != NULL )
    {
      prev = get_prev(*head,key);
      if ( prev != NULL )
      {
        prev -> next = temp->next;
        free (temp);
      }
      else
      {
        *head = temp ->next;
        free(temp);
```

Firstly the node to be deleted is searched in the linked list

Once the node to be deleted is found then get the previous node of that node in variable prev

If we want to delete head node, then set next adjacent element as head and then deallocate old head node.


```
      }
      printf("\nThe Element is deleted\n");
      getch();
      clrscr();
    }
}
```

Q.14 Write a C function to display a linked list. [JTNU : Part B, Marks 5]

Ans. :

```
void  display(node *head)
{
    node *temp ;
    temp = head;
    if ( temp == NULL )
    {
      printf("\nThe list is empty\n");
      getch();
      clrscr();
      return;
    }
    while ( temp != NULL )
    {
      printf(" %d -> ", temp->data );
      temp = temp -> next;
    }
    printf("NULL");
    getch();
    clrscr();
}
```

Q.15 Write a C function to search the desired element from the linked list. [JNTU : Part B, Marks 5]

Ans. :

```
node *search(node *head, int key)
{
    node *temp;
    int found;

    temp = head;
    if ( temp == NULL )
    {
      printf("The Linked List is empty\n");
      getch();
      clrscr();
      return NULL;
    }
    found = FALSE;
    while ( temp != NULL && found==FALSE)
```


Scanned by CamScanner

```
    {
        if ( temp->data != key)
            temp = temp -> next;
        else
            found = TRUE;
    }
    if ( found==TRUE )
    {
        printf("\nThe Element is present in the list\n");
        getch();
        return temp;
    }
    else
    {
        printf("The Element is not present in the list\n");
        getch();
        return NULL;
    }
}
```

Compare each node value with key value. If not matching then move to next node.

If the node containing desired data is obtained in the linked list then **found** variable is set to TRUE

Q.16 Should arrays of linked lists be used for the following type of applications ? Justify your answer.
a) Many search operations in sorted list.b) Many search operations in unsorted list.
☞[JNTU : Part A, Marks 2]

Ans. :

a) If the list is sorted then using linked list the desired element can be searched, simply by moving forward using 'next' pointer.

b) If the list is not sorted then using arrays the desired element can be searched. The arrays facilitates the access to random element.

Q.17 Given a singly linked list, write a function swap to swap every two nodes e.g. 1->2->3->4->5->6 should become 2->1->4->3->6->5.
☞[JNTU : Part B, Dec.-14, Marks 10]

Ans. :

Assumption

1. The linked is created using create routine.
2. The display routine is written to display the nodes of the linked list.
3. The swap routine for swapping two nodes is as follows -

```
void swap(node **head)
{
  // If linked list is empty or there is only one node in list
  if (*head == NULL || (*head)->next == NULL)
     return;
  node *prev = *head;//maintaining the previous node
  node *current = (*head)->next;//and curret node.
   //Thus two nodes are focused at a time

  *head = current;  // Change head before proceeding
```



```
  // Traverse the list
  while (true)
  {
     node *temp = current->next;
     current->next = prev;    // Change next of current as previous node
     if (temp == NULL || temp->next == NULL)
     {
        prev->next = temp;
        break;
     }

     // Change next of previous to next of temp
     prev->next = temp->next;

     // Update previous and current nodes
     prev = temp;
     current = prev->next;
  }
}
```

Q.18 Write a C program to implement insertion to the immediate left of kth node in singly linked list.
☞[JNTU : Part B, Dec. 16, Marks 5]

Ans. :

```
void insert(int k)
{
    node *temp,*prev_node,*New;
    int count;
    temp = head;
    prev_node = head;
    count = 1;
    if (temp == NULL)
    {
        printf("The Linked List is empty\n");

    }
    while (temp != NULL)
    {
        if(count==k)
            break;
        else
        {
        //keeping track of previous node
            prev_node = temp;
            temp = temp->next;
            count++;
        }
```


Scanned by CamScanner

```
        }
        if (count==k)
        {
            New = new node;
            printf("\n Enter the new node: ");
            scanf("%d",&New->data);
            prev_node->next = New;
            New->next = temp;
            printf("\n The element is inserted in the linked list!!!");
        }
        else
        {
            printf("The Element is not present in the list\n");

        }
    }
```

Q.19 Given an ordered linked list whose node represented by 'key' as information and next as 'link' field. Write C program to implement deleting number of nodes(consecutive) whose 'key' values are greater than or equal to 'Kmin' and less than 'Kmax'. ☞ [JNTU : Part B, Dec. 16, Marks 10]

Ans. :

i) The Pseudo code for deleting elements less than or equal to Kmax is as follows -

```
void delet()
{
    node *temp,*NextNode;
    int Kmax;
    temp = head;
    printf( "\n Enter the node value as Kmin");
    scanf("%d",&Kmax);
    if (temp == NULL)
    {
        prinf("The Linked List is empty\n");

    }
    while (temp != NULL)
    {
        if (Kmax == temp->data)
            break;
        else
        {
            NextNode = temp->next;
            temp->next = NULL;
            free(temp);
            temp = NextNode;
        }
    }
```



```
    NextNode = temp->next;
    temp->next = NULL;
    free(temp);
    temp = NextNode;
    head = temp;
}
```

ii) The Pseudo code for deleting elements greater than or equal to Kmin is as follows -

```
void delet()
{
    node *temp,*NextNode;
    int Kmin;
    temp = head;
    printf( "\n Enter the node value as Kmin");
    scanf("%d",&Kmin);
    if (temp == NULL)
    {
        printf("The Linked List is empty\n");

    }
    while(Kmin != temp->data)
    {
        if(Kmin==temp->data)
            break;
        else
        {
            temp = temp->next;
        }
    }
    while (temp != NULL)
    {
        NextNode = temp->next;
        temp->next = NULL;
        free(temp);
        temp = NextNode;
    }
}
```

Q.20 Write an algorithm to count the number of nodes between given two nodes in a linked list. ☞ [JNTU : Part B]

Ans. :

```
void count_between_nodes(node *temp1, node *temp2)
{
 int count=0;
 //temp1 represents the starting node
 //temp2 represents the end node
 //we have to count number of nodes between temp1 and temp2
 while(temp1->next!=temp2)
```


Scanned by CamScanner

```
    count++;
    printf("\n Total number of nodes between % and %d is %d ", temp1->data, temp2->data, count); }
```

Q.21 Write an algorithm to find the location of an element in the given linked list. Is the binary search will be suitable for this search ? Explain the reason. [JNTU : Part B]

Ans. :

```
void search_element(node *head,int key)
{
 node *Temp;
 int count=1;
 //head is a starting node of the linked list
 //key is the element that is to be searched in the linked list.
 Temp=head;
 while(Temp->next!=NULL)
 {
  if(temp->data==key)
  {
    printf("\n The element is present at location %d",count);
    break;
  }
  count++;
  Temp=Temp->next;
 }
}
```

Q.22 List the applications of linked list. [JNTU : Part A, Marks 3]

Ans. : Various applications of linked list are -

1. Linked list can be used to implement linear data structures such as stacks, and queues
2. Linked list is useful for implementing the non-linear data structures such as trees or graphs.
3. Polynomial representation, and operations such as addition, multiplication and evaluation can be performed using linked list.

1.2.2 : Doubly Linked List

Q.23 Define doubly linked list. [JNTU : Part A, May 09,15, Dec. 12, 16, Marks 2]

Ans. : Doubly linked list is a kind of linked list in which each node has two link fields. One link field stores the address of previous node and the other link field stores the address of next node.

Using doubly linked list traversing the linked list in forward and backward direction becomes efficient.

The typical structure of each node in doubly linked list is like this.

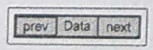
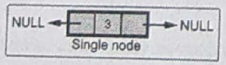


Fig. Q.23.1



Q.24 Discuss about doubly linked list. *(Refer Q. 23)* [JNTU : May 2018, Marks 3]

Q.25 What is linked list? Write advantages of doubly linked list over singly linked list. [JNTU : Part A, Dec 16, Marks 2]

Ans. : Linked list is a linear data structure in which each node contains two fields - data and next pointer. The next pointer field points to the next node in the linked list. The next pointer of the last node is NULL. For example -

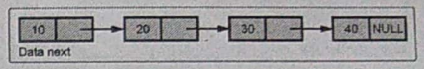


Fig. Q.25.1 Singly linked list

The main advantage of doubly linked list over singly linked list is that one can always keep track of previous node as well as next node from the current node. The reason behind is that - the doubly linked list contains two pointers - next link pointer and previous link pointer.

Q.26 State the advantages of circular lists over doubly linked list. [JNTU : Part A, Marks 2]

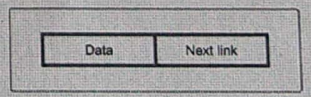
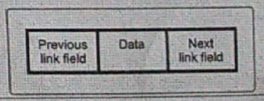
Ans. :
- In circular list the next pointer of last node points to head node, whereas in doubly linked list each node has two pointers : One previous pointer and another is next pointer.
- The main advantage of circular list over doubly linked list is that with the help of single pointer field we can access head node quickly.
- Hence some amount of memory get saved because in circular list only one pointer field is reserved.

Q.27 What are the advantages of doubly linked list over singly linked list ?

Ans. :
- The doubly linked list has two pointer fields. One field is previous link field and another is next link field.
- Because of these two pointer fields we can access any node efficiently whereas in singly linked list only one pointer field is there which stores forward pointer, which makes accessing of any node difficult one.

Q.28 What is the difference between singly linked list and doubly linked list ? [JNTU : Part B, May 09, Marks 4]

Ans. :

Sr. No.	Singly Linked List	Doubly Linked List
1.	Singly linked list is a collection of nodes and each node is having one data field and next link field. For example : 	Doubly linked list is a collection of nodes and each node is having one data field, one previous link field and one next link field. For example : 
2.	The elements can be accessed using next link.	The elements can be accessed using both previous link as well as next link.

Scanned by CamScanner

3.	No extra field is required; hence node takes less memory in SLL.	One field is required to store previous link hence node takes more memory in DLL.
4.	Less efficient access to elements.	More efficient access to elements.

1.2.3 : Circular Linked List

☞ [JNTU : Part A, Marks 3]

Q.29 What is Circular Linked List ?

Ans. : The circular linked list is a kind of linked list in which the last node is connected to the first node or head node of the linked list. For example - Refer Fig. Q.29.1

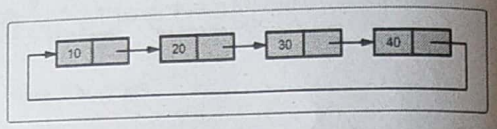


Fig. Q.29.1

☞ [JNTU : Part A, Marks 2]

Q.30 What is header node in linked list ?

Ans. : The header node is the very first node of the linked list. Sometimes a dummy value such as -999 is stored in the data field of header node. Refer Fig. Q.30.1

This node is useful for getting the starting address of the linked list. As there is only a forward link present in the linked list, for accessing the entire linked list it is necessary to obtain the starting address of the linked list.

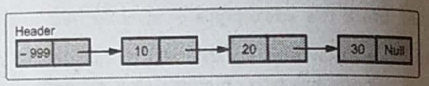


Fig. Q.30.1 Singly linked list with header node

Q.31 Explain the operations of circular linked list. ☞ [JNTU : Part B, Nov.-15, May-15, Marks 5]

Ans. :

(1) Creation of Circular Linked List

```
struct node *Create()
{
    char ans;
    int flag=1;
    struct node *head,*New,*temp;
    struct node *get_node();
    clrscr();
    do
    {
        New = get_node();
        printf("\n\n\n\tEnter The Element\n");
        scanf("%d",&New->data);
        if(flag==1)/*flag for setting the starting node*/
        {
            head = New;
            New->next=head;
            flag=0;/*reset flag*/
        }
        else        /* find last node in list */
```



```
        {
            temp=head;
            while (temp->next != head)/*finding the last node*/
                temp=temp->next; /*temp is a last node*/

            temp->next=New;/*attaching the  node*/

            New->next=head; /*each time making the list circular*/
        }
        printf("\n Do you want to enter more nodes?");

        ans=getch();
    }while(ans=='y'||ans=='Y');
    return head;
}
/* function used while allocating memory for a node*/
struct node *get_node()
{
    struct node *New;
    New =(node *) malloc(sizeof(struct node));
    New->next = NULL;
    return(New);/*created node is returned to calling function*/
}
```

2. Display of circular linked list

```
void Display(struct node *head)
{
    struct node *temp;
    temp = head;
    if(temp == NULL)
     printf("\n Sorry ,The List Is Empty\n");
    else
    {
     do
     {
        printf("%d\t",temp->data);
        temp = temp->next;
     }while(temp != head);
    }
    getch();
}
```

3. Insertion of a node into the circular linked list

```
struct node *insert_head(struct node *head)
{
    struct node *get_node();
    struct node *New,*temp;
    New=get_node();
```


Scanned by CamScanner

```
        printf("\n Enter The element which you want to insert ");
        scanf("%d",&New->data);
        if(head==NULL)
            head=New;
        else
        {

            temp=head;
            while(temp->next!=head)
                temp=temp->next;
            temp->next=New;
            New->next=head;
            head=New;
            printf("\n The node is inserted!");
        }
        return head;
}
/*Insertion of node at last position*/
void insert_last(struct node *head)
{
    struct node *New,*temp;
    New=get_node();
    printf("\n Enter The element which you want to insert ");
    scanf("%d",&New->data);
    if(head==NULL)
        head=New;
    else
    {
        temp=head;
        while(temp->next!=head)
            temp=temp->next;
        temp->next=New;
        New->next=head;
        printf("\n The node is inserted!");
    }
}
void insert_after(struct node *head)
{
    int key;
    struct node *New,*temp;
    New= get_node();
    printf("\n Enter The element which you want to insert ");
    scanf("%d",&New->data);
    if(head==NULL)
    {
        head=New;
```



```
    }
    else
    {
        printf("\n Enter The element after which you want to insert the node ");
        scanf("%d",&key);
        temp=head;
        do
        {
            if(temp->data==key)
            {
                New->next=temp->next;
                temp->next=New;
                printf("\n The node is inserted");
                return;
            }
            else
                temp=temp->next;
        }while(temp!=head);
    }
}
```

4. Deletion of node from a circular linked list

```
struct node *Delete(struct node *head)
{
 int key;
 struct node *temp,*temp1;
 printf("\n Enter the element which is to be deleted");
 scanf("%d",&key); temp=head;
 if(temp->data==key)/*If header node is to be deleted*/
 {
     temp1=temp->next;
     if(temp1==temp)
     {
         temp=NULL;
         head=temp;
         printf("\n The node is deleted");
     }
     else
     {
         while(temp->next!=head)
         temp=temp->next;/*searching for the last node*/
         temp->next=temp1;
         head=temp1;/*new head*/
         printf("\n The node is deleted");
     }
 }
 else
```


Scanned by CamScanner

```
    {
        while(temp->next!=head) /* if intermediate node is to be deleted*/
        {
        if((temp->next)->data==key)
        {
          temp1=temp->next;
          temp->next=temp1->next;
          temp1->next=NULL;
          free(temp1);
          printf("\n The node is deleted");
        }
        else
          temp=temp->next;
      }
    }
    return head;
  }
```

Q.32 Define and explain about circularly linked list and its operations with example. *(Refer Q. 31)*

[JNTU : May 18, Marks 5]

1.3 : Stack

Q.33 Define and explain the concept of stack. **[JNTU : Part A, Marks 2]**

Ans. :

Definition :

A stack is an **ordered list** in which all insertions and deletions are made at **one end**, called the **top**. If we have to make stack of elements 10, 20, 30, 40, 50, 60 then 10 will be the bottommost element and 60 will be the topmost element in the stack. A stack is shown in Fig. Q.33.1.

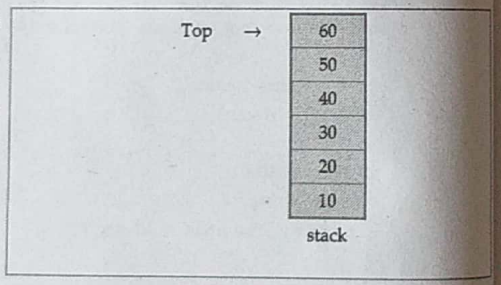


Fig. Q.33.1 Stack

Example

The typical example can be a stack of coins. The coins can be arranged one on another, when we add a new coin it is always placed on the previous coin and while removing the coin the recently placed coin can be removed. The example resembles the concept of stack exactly.

Q.34 What is a top pointer of a stack ? **[JNTU : Part A, Marks 2]**

Ans. : The top denotes the only one end of the stack from which the element can be inserted or deleted. When we push the element onto the stack, the top is incremented. When we pop the element from the stack the top is decremented.

1.3.1 : Array Representation of Stack

Q.35 Write about stack using array with an example. **[JNTU : Part B, Dec.-16, Marks 5]**

Ans. :

Declaration 1 :

```
#define size 100
int stack[size], top = -1;
```

In the above declaration stack is nothing but an array of integers. And most recent index of that array will act as a top.

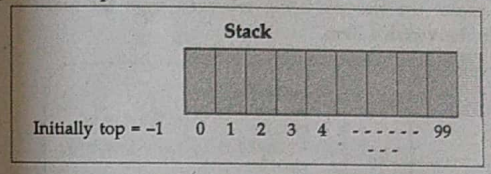


Fig. Q.35.1 Stack using one dimensional array

The stack is of the size 100. As we insert the numbers, the top will get incremented. The elements will be placed from 0th position in the stack. At the most we can store 100 elements in the stack, so at the most last element can be at (size-1) position, i.e., at index 99.

Declaration 2 :

```
#define size 10
struct stack {
        int s[size];
          int top;
        }st;
```

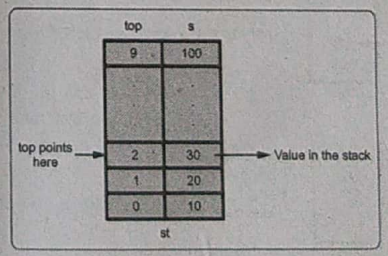


Fig. Q.35.2 Stack using structure

1.3.2 : Operations on Stack

Q.36 Write the stack ADT. **[JNTU : Part A,May-15,Dec.-18, Marks 2]**

Ans. : Abstract DataType stack

{

Instances : Stack is a collection of elements in which insertion and deletion of elements is done by one end called top.

Preconditions :

1. **Stfull () :** This condition indicates whether the stack is full or not. If the stack is full then we cannot insert the elements in the stack.
2. **Stempty () :** This condition indicates whether the stack is empty or not. If the stack is empty then we cannot pop or remove any element from the stack.

Operations :

1. **Push :** By this operation one can push elements onto the stack. Before performing push we must check stfull () condition.
2. **Pop :** By this operation one can remove the elements from stack. Before popping the elements from stack we should check stempty () condition.

}

Q.37 Write functions and explain for push and pop operations of a stack ? **[JNTU : Part B, Jan 14, Marks 7]**

Ans. :

(1) The Push Function

```
void push(int item)
{
 st.top++; / * top pointer is set to next location */
 st.s[st.top] =item; /* placing the element at that
location */
}
```

- Note that the push function takes the parameter **item** which is actually the element which we want to insert into the stack.
- Before calling the push function we have checked whether the stack is full or not, if the stack is not full then only the insertion of the element can be achieved by means of push operation.

Scanned by CamScanner

(2) Pop Function

```
int pop()
{
 int item;
 item = st.s[st.top];
 st.top – –;
 return(item);
}
```

- Note that the item on the top of the stack will always gets popped off.
- The pop function returns the element which is at the top of the stack.
- The value at the top is stored in some variable as item and it then decrements the value of the top, which now points to the element which is just under the element being retrieved from the stack. Finally it returns the value of the element stored in the variable **item**.

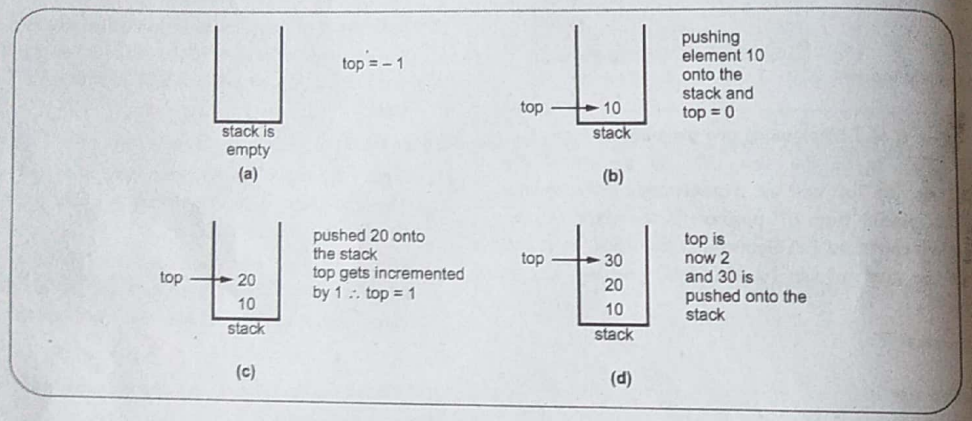


Fig. Q.37.1 Performing push operation

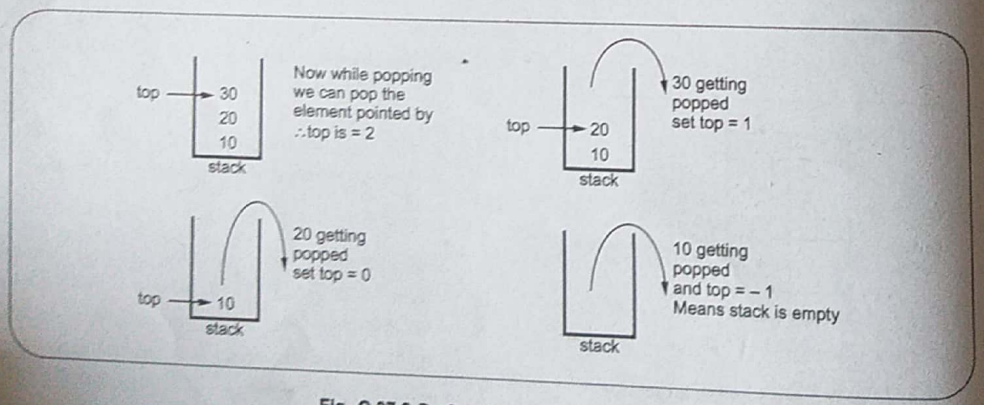


Fig. Q.37.2 Performing pop operation

Q.38 Write a program to push the element into a stack.
(Refer Q.39)

☞ [JNTU : Part B, Dec 17, Marks 5]

Q.39 Write a C program for array implementation of stack.

☞ [JNTU : Part B, Marks 5]

Ans. :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
/* stack structure*/
struct stack {
        int s[size];
        int top;
        }st;
/*
 The stfull Function
 Output:returns 1 or 0 for stack full or not
*/
int stfull()
{
    if(st.top>=size-1)
        return 1;
    else
        return 0;
}
/*
 The push Function
 Input:item which is to be pushed
 Output:none-simply pushes the item onto the stack
*/
void push(int item)
{
    st.top++;
    st.s[st.top] =item;
}
/*
 The stempty Function
 Output:returns 1 or 0 for stack empty or not
*/
int stempty()
{
    if(st.top==-1)
        return 1;
    else
        return 0;
}
```


Scanned by CamScanner

```
/*
 The pop Function
 Output:returns the item which is popped from the stack
*/
int pop()
{
    int item;
    item=st.s[st.top];
    st.top--;
    return(item);
}
/*
 The display Function
 Output:none-displays the contents of the stack
*/
void display()
{
    int i;
    if(stempty())
        printf("\n Stack Is Empty!");
    else
    {
        for(i=st.top;i>=0;i--)
            printf("\n%d",st.s[i]);
    }
}
/*
 The main Function
 Calls:push,pop,stempty,stfull,display
*/
void main(void)
{
    int item,choice;
    char ans;
    st.top=-1;
    clrscr();
    printf("\n\t\t Implementation Of Stack");
    do
    {
        printf("\n Main Menu");
        printf("\n1.Push\n2.Pop\n3.Display\n4.exit");
        printf("\n Enter Your Choice");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("\n Enter The item to be pushed");
```



```
            scanf("%d",&item);
            if(stfull())
                printf("\n Stack is Full!");
            else
                push(item);
        break;
        case 2:if(stempty())
                printf("\n Empty stack!Underflow !!");
            else
            {
                item=pop();
                printf("\n The popped element is %d",item);
            }
        break;
        case 3:display();
        break;
        case 4:exit(0);
    }
    printf("\n Do You want To Continue?");
    ans=getche();
  }while(ans =='Y' ||ans =='y');
  getch();
}
/******************* End Of Program ************************/
```

Output

```
Implementation Of Stack
    Main Menu
    1. Push
    2. Pop
    3. Display
    4. exit
    Enter Your Choice1
    Enter The item to be pushed 10
    Do You want To Continue?y
    Main Menu
    1. Push
    2. Pop
    3. Display
    4. exit
    Enter Your Choice 1
    Enter The item to be pushed 20
    Do You want To Continue?y
```


Scanned by CamScanner

```
Main Menu
1. Push
2. Pop
3. Display
4. exit
Enter Your Choice3
20
10
Do You want To Continue?y
Main Menu
1. Push
2. Pop
3. Display
4. exit
Enter Your Choice2
The popped element is 20
Do You want To Continue?n
```

1.3.3 : Linked List Representation of Stack

Q.40 Give the representation of linked stack ☞ [JNTU : Part B, Marks 5]

Ans. : • The advantage of implementing stack using linked list is that we need not have to worry about the size of the stack.

- Since we are using linked list as many elements we want to insert those many nodes can be created. And the nodes are dynamically getting created so there won't be any stack full condition.
- The typical 'C' structure for linked stack can be

```
struct stack
{
    int data;
    struct stack *next;
}node;
```

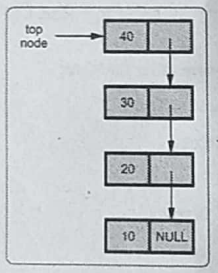


Fig. Q.40.1 Representing the linked stack

- Each node consists of data and the next field. Such a node will be inserted in the stack. Following figure represents stack using linked list.

Q.41 Explain push and pop operations for implementing linked stack ☞ [JNTU : Part B, Marks 5]

Ans. :

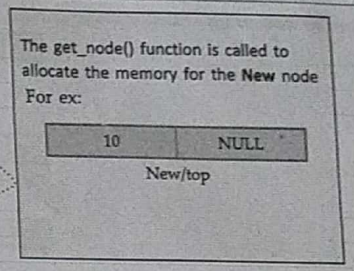
1. Push operation

The pseudo code for push operation as given below

```
void Push(int Item, node **top)
{
```

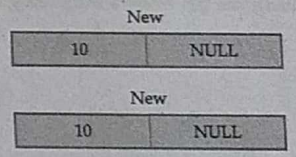


```
    node *New;
    node * get_node(int);

    New = get_node(Item);
    New -> next = *top;
    *top = New;
}

node * get_node( int item )
{
    node * temp;
    temp =(node *) malloc(sizeof(node) );
    if ( temp == NULL )
     printf("\nMemory Can not be allocated \n");
    temp -> data = item;
    temp -> next = NULL;
    return( temp);
}
```



In the main function, when the item to be inserted is entered, a call to push function is given. In push function another function get_node is invoked to allocate the memory for a node as follows :

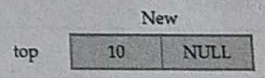
In the main function, when the item to be inserted is entered, a call to push function is given. In push function another function get_node is invoked to allocate the memory for a node as follows :



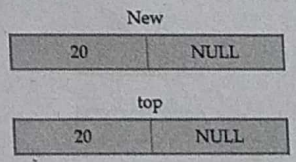
If item = 10, then by get node function node will be allocated. Then

*top = New

This is our new **top** node. Hence stack will look like this



If again **Push** function is called for pushing the value 20 then



Scanned by CamScanner

This can be done using following statements -

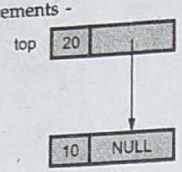


Fig. Q.41.1

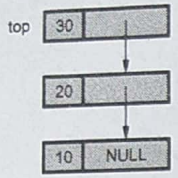


Fig. Q.41.1

```
New->next=*top
      top=New;
```

And we will get

When 30 is pushed then

Thus by pushing the items repeatedly we can create a stack using linked list.

2. Pop operation

```
int Pop(node **top)
{
    int item;
    node *temp;
    item = (*top) ->data;
    temp = *top;

    *top = (*top) -> next;
    free(temp);
    return(item);
}
```

Taking the address of **top** in **temp** node and assigning new **top** to next node

If stack is as follows -

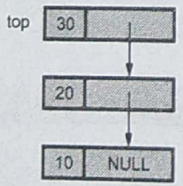


Fig. Q.41.2

Then for popping the element we write following code -

```
item = *(top)->data        then item=30
temp=*top
*top=*top->next            now top=20
free(temp);                deallocating memory of node 30.That means node 30 is deleted.
```

Q.42 Write a C program to implement linked stack [Part B, Marks 5]

Ans. :

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <stdlib.h>
#include <alloc.h>
/*Declaration for data structure of linked stack*/
typedef struct stack
{
    int data;
    struct stack *next;
}node;
void main ( )
{
    /* Local declarations */
    node *top;
    int data,item,choice;
    char ans,ch;
    void Push(int , node **);
    void Display(node **);
    int Pop(node **);
    int Sempty(node *);
    clrscr();
    top = NULL;
    printf("\n\t\t Stack Using Linked List");
    do
    {
        printf("\n\n The main menu");
        printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");
        printf("\n Enter Your Choice");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("\n Enter the data");
                   scanf("%d", &data);
                   Push(data,&top);
                   break;
            case 2:if(Sempty(top))   /*before popping the data
                       whether stack is empty or not is checked*/
```


Scanned by CamScanner

```
                printf("\n stack underflow!");
            else
            {
                item = Pop(&top);
                printf("\n The popped node is%d",item);
            }
                break;
        case 3:Display(&top);
        /*displays stack from top to bottom*/
            break;
        case 4:printf("\n Do You want To Quit?(y/n)");
            ch=getche();
            if(ch=='y')
                exit(0);
            else
                break;
    }
    printf("\n Do you want to continue?");
    ans =getche();
    getch();
    clrscr();
  }while(ans =='Y' ||ans =='y');
  getch();
}

void Push(int Item, node **top)
{
    node *New;
    node * get_node(int);

    New = get_node(Item);
    New -> next = *top;
    *top = New;
}

node * get_node( int item )
{
    node * temp;
    temp =(node *) malloc(sizeof(node) );
    if ( temp == NULL )
     printf("\nMemory Can not be allocated \n");
    temp -> data = item;
    temp -> next = NULL;
    return( temp);
}
```



```
int Sempty(node *temp)
{
    if(temp ==NULL)
        return 1;
    else
        return 0;
}

int Pop(node **top)
{
    int item;
    node *temp;
    item = (*top) ->data;
    temp = *top;
    *top = (*top) -> next;
    free(temp);
    return(item);
}
void Display(node **head )
{
    node *temp ;
    temp = *head;
    if(Sempty(temp))
        printf("\n The stack is empty!");
    else
    {
        while ( temp != NULL )
        {
            printf("%d\n",temp-> data);
            temp = temp -> next;
        }
    }
    getch();
}
```

Q.43 Discuss about linked implementation of stack ADT. *(Refer Q.42)* **[JNTU : May 18, Marks 5]**

1.3.4 : Applications of Stack

Q.44 What is stack ? List the applications of stack. **[JNTU : Part A, Dec.-16, Marks 3]**

Ans. : Stack is a linear data structure in which insertion of element in the stack and deletion of element from the stack is done using only one end called top.

Applications

1. Stack is used in conversion of infix expression to postfix.
2. Stack is used in evaluating the postfix expression.

Scanned by CamScanner

3. Stack is used for reversing the given string.
4. Stack is used in checking well formedness of parenthesis.

Q.45 List the characteristics of stacks.

Ans. :

1. Insertion and deletion can be made by one end only. Hence the element inserted last will be first one to come out. Hence sometimes it is called LIFO.
2. Stacks are useful for evaluating an expression.
3. Stacks can store the functions calls.

Q.46 State different ways of representing expression.

Ans. : Different ways of representing expression are

1. Infix expression
2. Prefix expression
3. Postfix expression

Q.47 Explain different types of expressions used.

Ans. :

1. Infix Expression :

In this type of expressions the arrangement of operands and operator is as follows.

Infix expression = operand1 operator operand2

For example 1. (a+b)
2. (a+b) * (c – d)
3. (a+b/e) * (d+f)

Parenthesis can be used in these expressions infix expression are the most natural way of representing the expressions.

2. Postfix Expression :

In this type of expressions the arrangement of operands and operator is as follows

Postfix expression = operand1 operand2 operator

For example1. ab+
2. ab + cd – *
3. ab + e / df + *

In postfix expression there is no parenthesis used. All the corresponding operands come first and then operator can be placed.

3. Prefix Expression :

In prefix expression the arrangement of operands and operators is as follows.

Prefix expression = operator operand1 operand2

For example

1. + ab
2. * + ab – cd
3. * / + abe + df

In prefix expression, there is no parenthesis used. All the corresponding operators come first and then operands are arranged.

Q.48 What are the advantages of infix expression ?

Ans. :

1) This is an easiest method of representing expression in which it is always clear that which operation is to be done first.
2) It is mathematical way of representing expression.

Q.49 What are the advantages of postfix or prefix expression ?

Ans. :

1) There is no ambiguity in rules of precedence in this type of expressions
2) There is no ambiguity in deciding the precedence of expression.
3) There is no need to use parenthesis in the expression.

Q.50 What do you understand by polish notation ? Explain.

Ans. : This is also called as prefix notation. In this type of notation the operator followed by two operands. For example if (a+b)*c is a given expression then its polish notation will be *+ abc.

Q.51 Write the steps for converting expression from infix to postfix.

☞ [JNTU : Part A, May-15, Marks 3, Part B, Dec.-17,18, Marks 5]

Ans. :

1. Read the infix expression for left to right one character at a time.
2. Initially push $ onto the stack. For $ in stack set priority as - 1.
3. If the input symbol read is an operand then place it in postfix expression.
4. If the input symbol read is operator then
   a) Check if priority of operator in stack is greater than the priority of incoming (or input read) operator then pop that operator from stack and place it in the postfix expression. Repeat step 4(a) till we get the operator in the stack which has greater priority than the incoming operator.
   b) Otherwise push the operator being read, onto the stack.
   c) If we read input operator as ')' then pop all the operators until we get "(" and append popped operators to postfix expression. Finally just pop "("
5. Final pop the remaining contents, from the stack until stack becomes empty append them to postfix expression.
6. Print the postfix expression as a result.

Q.52 Write an algorithm to convert infix expression into postfix. *(Refer Q. 51)* ☞ [JNTU : Dec 17, Marks 5]

Q.53 What are various applications of Stacks? Explain infix to postfix conversion.

☞ [JNTU : May 18, Marks 5]

Ans. : Applications of Stack - Refer Q.44.
Infix to Postfix Conversion - Refer Q.51.

Q.54 Give postfix notation for the following infix expression.
A*(B+C/(D+E))-F*(G/H+I) ☞ [JNTU : Part B]

Ans. :

Input	Action	Stack	Output
A*(B+C/(D+E))-F*(G/H+I)	Read A Print A		A
(B+C/(D+E))-F(G/H+I)	Read *,Push	*	A
(B+C/(D+E))-F*(G/H+I)	Push (	*(	A
B+C/(D+E))-F*(G/H+I)	Read B,Print B	*(	AB
+C/(D+E))-F*(G/H+I)	Push +	*(+	AB
C/(D+E))-F*(G/H+I)	Read C, Print B	*(+	ABC
/(D+E))-F*(G/H+I)	Read /, Push /	*(+/	ABC
(D+E))-F*(G/H+I)	Push (	*(+/(	ABC
D+E))-F*(G/H+I)	Read D, Print D	*(+/(	ABCD
+E))-F*(G/H+I)	Push +	*(+/(+	ABCD
E))-F*(G/H+I)	Read E, Print E	*(+/(+	ABCDE
))-F*(G/H+I)	Read), pop +, print, pop (.	*(+/	ABCDE+

Scanned by CamScanner

)-F*(G/H+I)	Read), pop /, print,Pop + print, pop (.	*	ABCDE+/+
-F*(G/H+I)	Pop *, print, and push -	-	ABCD+/+*
F*(G/H+I)	Read F, print F	-	ABCD+/+*F
*(G/H+I)	Push *	-*	ABCD+/+*F
(G/H+I)	Push (	-*(	ABCD+/+*F
G/H+I)	Read G, Print G	-*(	ABCD+/+*FG
/H+I)	Push /	-*(/	ABCD+/+*FG
H+I)	Read H, Print H	-*(/	ABCD+/+*FGH
+I)	Pop /, Push +	-*(+	ABCD+/+*FGH/
I)	Read I, Print I	-*(+	ABCD+/+*FGH/I
)	Pop, +, print. Pop (.	-*	ABCD+/+*FGH/I+
end of input	Pop *, print. Pop -, print it.	Empty	ABCD+/+*FGH/I+*-

Q.55 Convert the following expression into postfix form. Show all the steps and stack content : 4$2*3 – 3 + 8/4(1 + 1) ☞[JNTU : Part B]

Ans. :

Input symbol	Action	Stack	Postfix expression
4	Print 4	empty	4
$	Push $	$	4
2	Print 2	$	42
*	POP $, Print it. Push *	*	42 $
3	Print 3	*	42 $ 3
–	POP *, Print it Push –	-	42 $ 3*
3	Print it	-	42 $ 3*3
+	POP –, Print it. Then push +	+	42 $ 3*3 –
8	Print 8	+	42 $ 3*3 – 8
/	Push /	+ /	42 $ 3*3 – 8
4	Print 4	+ /	42 $ 3*3 – 84
/	POP/, Print it Then Push/	+ /	42 $ 3*3 – 84 /
(	Push (	+ / (	42 $ 3*3 – 84 /
1	Print 1	+ / (	42 $ 3*3 – 84 / 1
+	Push +	+ / (+	42 $ 3*3 – 84 / 1

1	Print 1	+ / (+	42 $ 3*3 – 84 / 11
)	POP +, Print it POP (	+ /	42 $ 3*3 – 84 / 11 +
end of input	POP/, Print POP +, Print	empty	42 $ 3*3 – 84 / 11 + / +

The postfix expression is **42$3*3 – 84/11+/+**

Q.56 What are the steps in evaluating the postfix expression ? ☞[JNTU : Part A, Dec.-16, Marks 3]

Ans. :

1. Read the postfix expression from left to right.
2. If the input symbol read is an operand then push it on to the stack.
3. If the operator is read POP two operands and perform arithmetic operations if operator is

+	then result = operand 1 + operand 2
–	then result = operand 1 – operand 2
*	then result = operand 1 operand 2
/	then result = operand 1 / operand 2

4. Push the result onto the stack.
5. Repeat steps 1-4 till the postfix expression is not over.

Q.57 Show in detail the contents of stack to evaluate the given postfix expression [1 2 3 + * 3 2 1 - + *]. ☞[JNTU : Part A, Dec.-16, Marks 2]

Ans. : Postfix expression : 123 + * 321 - + *

Input read	Action	Stack
1	Push 1	1
2	Push 2	1 2
3	Push 3	1 2 3
+	POP 3, POP 2 perform 3 + 2 = 5 Push 5	1 5
	POP 5, POP 1 Perform 1 * 5 = 5 Push 5	5
3	Push 3	5 3
2	Push 2	5 3 2
1	Push 1	5 3 2 1
–	POP 1, POP 2 Perform 2 – 1 = 1 Push	5 3 1
+	POP 1, POP 3 Perform 3 + 1 = 4 Push 4	5 4
	POP 4, POP 5 Perform 5 * 4 = 20 Push 20	20

The result of evaluation is 20.

Scanned by CamScanner

Q.58 Write the postfix notation for expression : (A+B)*C-(D-E) ^ F.

Ans. : Following steps can be followed for computing the postfix expression -

Step 1 : (AB+)*C-(D-E)^F

Step 2 : (AB+C*)-(D-E)^F

Step 3 : (AB+C*)-(DE-)^F

Step 4 : (AB+C*)-(DE-F^)

Step 5 : AB+C*DE-F^-

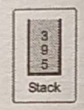
Q.59 Evaluate the following postfix expression 5 9 3 + 4 2 ** 4 2 7 + * using stack show the content the stack at each step. ☞ [JNTU : Part B]

Ans. :

Step 1 :

5 9 3 + 4 2 * * 7 + *

Read 5, 9 and 3 one by one and push onto the stack.



Step 2 :

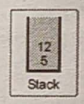
5 9 3 + 4 2 * * 7 + *

Read +, POP 3 and POP 9

op2 = 3, op1 = 9. Perform

op1 + op2 i.e. 9 + 3 = 12.

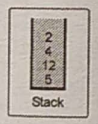
Push 12 onto the stack.stack.



Step 3 :

5 9 3 + 4 2 * * 7 + *

Push 4 and 2 onto the stack.



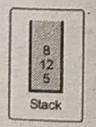
Step 4 :

5 9 3 + 4 2 * * 7 + *

Read *, pop 2 and 4.

op2 = 2 and op1 = 4.

Perform 4 * 2 = 8. Push 8 onto the stack.



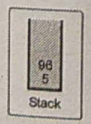
Step 5 :

5 9 3 + 4 2 * * 7 + *

Read *, pop 8 and then

12. Perform 12 * 8 = 96.

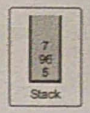
Push 96 onto the stack.



Step 6 :

Read 7 and push it

onto the stack.



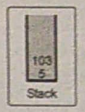
Step 7 :

5 9 3 + 4 2 * * 7 + *

Read +, pop 7 and

then 96. Perform

96 + 7 = 103. Push 103 onto the stack.

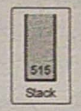


Step 8 :

5 9 3 + 4 2 * * 7 + *

Read *, pop 103 and then 5. perform

5 * 103 = 515. Push onto the stack.



Step 9 :

POP the contents of stack and

print the result of evaluation as **515.**

Q.60 Evaluate the following postfix expression using stack
623 + – 382 / + * 2 ∧
Note : ∧ stands for power and all operands are single digit. ☞ [JNTU : Part B]

Ans. :

Input read	Action	Stack	Operand 1	Operand 2	Value
6	Push 6	6			
2	Push 2	6 2			
3	Push 3	6 2 3			
+	POP two operands and perform + operation. Push the result		2	3	2 + 3 = 5
–	POP two operand and perform - operation. Push the result	6 5	6	5	6 – 5 = 1
3	Push 3	1 3			
8	Push 8	1 3 8			
2	Push 2	1 3 8 2			
/	POP two operands perform /. Push result	1 3	8	2	8/2 = 4
+	POP two operands and perform + Push the result	1 3 4	3	4	3 + 4 = 7

Scanned by CamScanner

*	POP two operands and perform *. Push the result	1 7	1	7	1 * 7 = 7
2	Push 2	7 2			
^	POP two operands and perform ^. Push the result		7	2	7 ^ 2 = 49
	POP the stack and print result	49			

The result of evaluation is 49.

Q.61 How to evaluate postfix expression? Explain. *(Refer Q.60)* **[JNTU : Dec 18, Marks 5]**

1.4 : Queues

Q.62 Write the Queue ADT. **[JNTU : Part A, Dec.-16, Marks 2]**

Ans. :

AbstractDataType queue

{

Instance : The queue is collection of elements in which the element can be inserted by one end called rear and elements get deleted from the end called front.

Operations :

1. que_full ( ) - Checks whether queue is full or not.
2. que_empty ( ) - Checks whether queue is empty or not.
3. Q_insert ( ) - Inserts the element in queue from rear end.
4. Q_delete ( ) - Deletes the element from the queue by front end.

}

Q.63 Suppose a queue is represented by circular array of size N. F and R are used to denote front and rear positions. If F points a location before front element of queue and R points to last element of queue, how many elements are there in the queue ? **[JNTU : Part A, Nov.-15, Marks 3]**

Ans. : If F points to the location before front element of a queue.

Rear R points to the last element of queue.

Then the elements in the queue are equal to N.

Q.64 What do the terms LIFO and FIFO means ? Explain. **[JNTU : Part A, Marks 2]**

Ans. :

(1) The LIFO stands for Last In First Out. This property belongs to stack. That means the element inserted lastly will be the element to be popped off first from the stack.

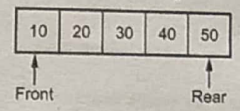
(2) The FIFO stands for First In First Out. This property belongs to queue. That means the element inserted first in the queue will be the first to get deleted.

Q.65 What are the front and rear pointers of queue ? **[JNTU : Part A, Marks 2]**

Ans. :

(1) The front end of the queue from which the element gets deleted is called front and the end from which the element is inserted in the queue is called rear.

(2) For example :



1.4.1 : Array Representation of Queue

Q.66 What is enqueue and dequeue ? **[JNTU : Part A, Marks 2]**

Ans. :

(1) The enqueue is a queue operation for inserting the element from rear end.

(2) The dequeue is an operation performed on the queue for deleting the element from the queue.

Q.67 How do you test for an empty queue ? **[JNTU : Part A, Marks 2]**

Ans. : Following is a C code which is used to test an empty queue -

```
if((Q.front==-1)||(Q.front>Q.rear))
    printf("\n The Queue is Empty!!");
else
    printf("\n The Queue is not Empty!!");
```

Q.68 Explain about the operations of queue with an example. **[JNTU: Part B, Nov 15, Marks 5]**

Ans. : Various operations on the queue are -

1. Queue overflow.
2. Insertion of the element into the queue.
3. Queue underflow.
4. Deletion of the element from the queue.
5. Display of the queue.

'C' representation of queue.

```
struct queue
{
    int que [size];
    int front;
    int rear;
} Q;
```

1. Insertion of element into the queue

The insertion of any element in the queue will always take place from the rear end.

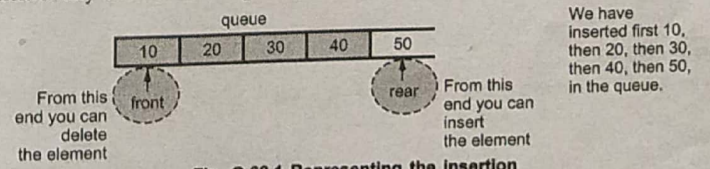


Fig. Q.68.1 Representing the insertion

Scanned by CamScanner

Before performing insert operation you must check whether the queue is full or not. If the rear pointer is going beyond the maximum size of the queue then the queue overflow occurs.

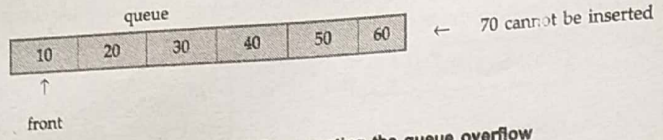


Fig. Q.68.2 Representing the queue overflow

2. Deletion of element from the queue

The deletion of any element in the queue takes place by the front end always.

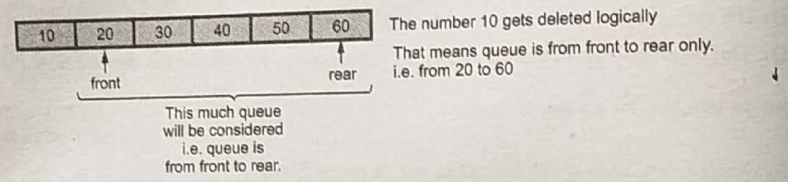


Fig. Q.68.3 Representing the deletion

Before performing any delete operation one must check whether the queue is empty or not. If the queue is empty, you cannot perform the deletion. The result of illegal attempt to delete an element from the empty queue is called the Queue Underflow condition.

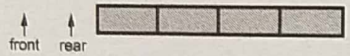


Fig. Q.68.4 Representing the queue underflow

Q.69 Write a C Program to implement Queue using Arrays. **[JNTU : Part B, Marks 5]**

Ans. :

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define size 5
struct queue
{
int que[size];
int front,rear;
}Q;
/*
The Qfull Function
Output:1 or 0 for q full or not
*/
Qfull()
{
```

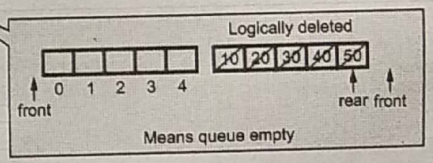
Queue data structure declared with array que [ ] , front and rear

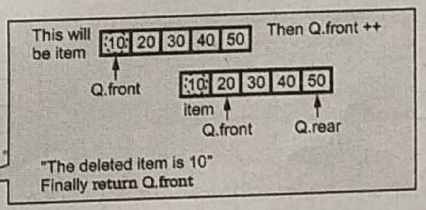

```
if(Q.rear >=size-1)
 return 1;
else
 return 0;
}
/*
The insert Function
Input:item -which is to be inserted in the Q
Output:rear value
*/
int insert(int item)
{
 if(Q.front == -1)
  Q.front++;
 Q.que[++Q.rear] = item;
 return Q.rear;
}
int Qempty()
{
 if((Q.front == - 1) || (Q.front > Q.rear))
  return 1;
 else
  return 0;
}
/*
The delet Function
Output:front value
*/
int delet()
{
 int item;
 item = Q.que[Q.front];
 Q.front++;
 printf("\n The deleted item is %d",item);
 return Q.front;
}
/*
The display Function
*/
void display()
{
 int i;
 for(i=Q.front;i<=Q.rear;i++)
  printf("  %d",Q.que[i]);
}
void main(void)
```

If Queue exceeds the maximum size of the array then it returns 1 - means queue full is true otherwise 0 means queue full is false

This condition will occur initially when queue is empty when single element will be present then both front and rear points to the same

Always increment the rear pointer and place the element in the queue



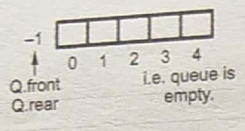


Printing the queue from front to rear

Scanned by CamScanner

```
{
 int choice,item;
 char ans;
 clrscr();
 Q.front = -1;
 Q.rear  = -1;
 do
 {
  printf("\n Main Menu");
  printf("\n1.Insert\n2.Delete\n3.Display");
  printf("\n Enter Your Choice");
  scanf("%d",&choice);
  switch(choice)
    {
case 1:if(Qfull())      //checking for Queue overflow
                printf("\n Can not insert the element");
        else
        {
                printf("\n Enter The number to be inserted");
               scanf("%d",&item);
                insert(item);
        }
       break;
    case 2:if(Qempty())
            printf("\n Queue Underflow!!");
           else
            delete();
           break;
    case 3:if(Qempty())
            printf("\nQueue Is Empty!");
           else
            display();
           break;
    default:printf("\n Wrong choice!");
           break;
    }
 printf("\n Do You Want to continue?");
 ans =getche();
 }while(ans =='Y'||ans =='y');
}
/********************** End Of Program **********************/
```

Initially



Output

```
Main Menu
    1. Insert
    2. Delete
    3. Display
     Enter Your Choice 1
     Enter The number to be inserted 10
```



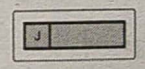
```
Do You Want to continue?y
Main Menu
1. Insert
2. Delete
3. Display
Enter Your Choice 1
Enter The number to be inserted 20
Do You Want to continue?y
Main Menu
1. Insert
2. Delete
3. Display
Enter Your Choice 3
  10   20
Do You Want to continue?y
Main Menu
1. Insert
2. Delete
3. Display
Enter Your Choice 2
The deleted item is 10
Do You Want to continue?n
```

Q.70 A letter means enqueue and an asterisk means dequeue in the following sequence. Give the sequence of values returned by the get operation when this sequence of operations is performed on an initially empty FIFO queue.

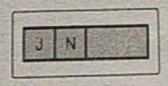
J N T*U*H Y DE R***A B*A**B*** ☞ [JNTU : Part B, Marks 5]

Ans. : Initially queue is empty

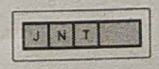
Step 1 : Insert J



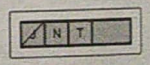
Step 2 : Insert N



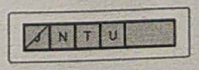
Step 3 : Insert T



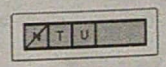
Step 4 : Delete from queue J is returned



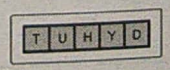
Step 5 : Insert U



Step 6 : Delete from queue N will be returned

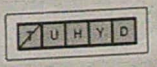


Step 7 : Insert H, Y, D



Step 8 : Delete from queue.

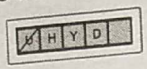
T will be returned.



Scanned by CamScanner

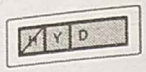
Step 9 : Delete from queue.

U will returned.

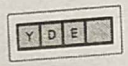


Step 10 : Delete from queue.

H will be returned.



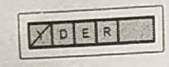
Step 11 : Insert E in queue



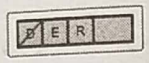
Step 12 : Insert R in queue

Step 13 : Delete from queue

return Y



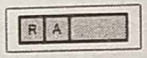
Step 14 : Delete from queue return D



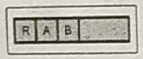
Step 15 : Delete from queue

return E

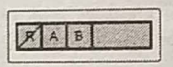
Step 16 : Insert A



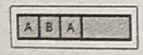
Step 17 : Insert B



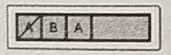
Step 18 : Delete from queue return R



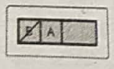
Step 19 : Insert A in queue



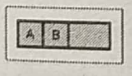
Step 20 : Delete from queue return A.



Step 21 : Delete from queue return B



Step 22 : Insert B in queue



Q.71 Show how to implement a queue using two stacks ? Analyse the running time of the queue operations.

[JNTU : Part B, Marks 5]

Ans. : There are two stacks namely stack1, stack2. Following steps can be performed for **insertq** and **deleteq** operations to implement queue using two stacks.

insertq (item)

{

Step 1 : push everything from stack1 to stack2 while stack1 is not empty.

Step 2 : push **item** to stack1.

Step 3 : Push all elements from stack2 to stack1.

}

deleteq ()

{

Step 1 : pop an item from stack1 and return it.

}

Now we will analyze time for queue operations

insertq operation

- Step 1 will require **O(n)** time.
- Step 2 will 'require O(1).
- Step 3 will require O(n).

Thus total O(n) + O(1) + O(n) = O(n) time will be required by insert operation of queue.

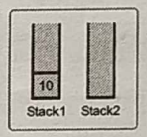
deleteq operation : Step 1 will require O(1) time.

Consider elements 10, 20, 30, 40 to be inserted in queue. The steps are as follows -

Step 1 : Insertq(10)

As there is nothing in stack1 initially hence nothing will be pushed to stack1.

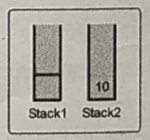
push 10 to stack1



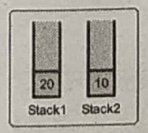
Nothing will be copied to stack1. From stack2 as stack2 is empty. If deleteq operation is to be performed then top element of stack1 will be returned.

Step 2 : Insertq(20)

push everything from stack1 to stack2



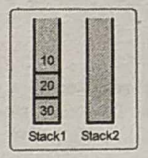
push 20 to stack1



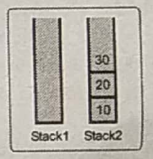
pop everything from stack2 and push it to stack1

Step 3 : Insertq (30)

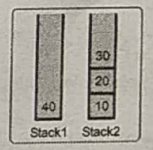
push everything from stack1 to stack2



push 30 to stack1.



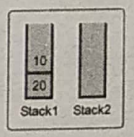
pop from stack2 and push it onto stack1



If deleteq operation is to be performed then pop from stack1.

Step 4 : Insertq (40)

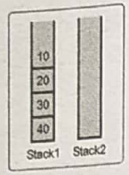
push everything from stack1 to stack2



push 40 onto stack1

Scanned by CamScanner

push everything from stack2 to stack1



If deleteq operation is to be performed the pop from stack1.

Q.72 Show how to implement a stack using two Queues ? Analyse the run time of the stack operations ? [JNTU : Part B, Marks 5]

Ans. : We will push the elements 10, 20, 30, 40, 50 onto the stack. During the push operations one can invoke pop at any time. Note that these push and pop operations can be performed only with the help of two queues Q1 and Q2.

Following steps will be followed during push or pop operations.

push (item)

{

Step 1 : Insert item in Q2

Step 2 : One by one delete everything from Q1 and insert them to Q2.

Step 3 : Swap names of Q1 and Q2.

That is change name of Q2 as Q1 and Q1 as Q2.

}

pop ( )

{

Delete an item from Q1

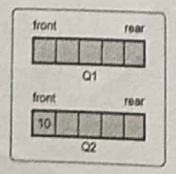
and return it.

}

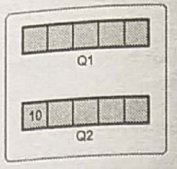
Now let us use above steps.

Step 1 : push 10

Insert 10 to Q2



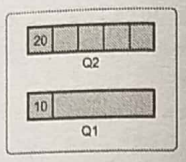
As there is nothing in Q1 no element will be copied to Q2. Now make Q2 as Q1 and Q1 as Q2.



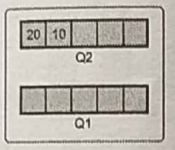
If pop operation needs to be performed then simply delete element from Q1 from front end.

Step 2 : push 20

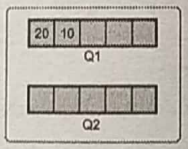
Insert 20 to Q2.



Copy all elements from Q1 to Q2.



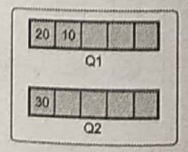
Now make Q2 as Q1 and Q1 as Q2.



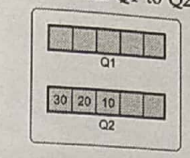
If pop operation needs to be performed then simply delete element from Q1.

Step 3 : push 30.

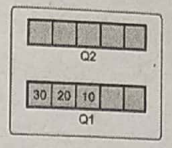
Insert 30 to Q2.



Copy all the elements from Q1 to Q2.



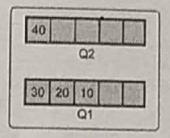
Make Q2 as Q1 and Q1 as Q2.



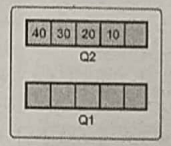
If pop operation needs to be performed then, simply delete element from Q1.

Step 4 : push (40)

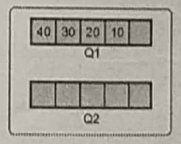
Insert 40 to Q2



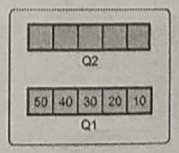
Copy all the elements from Q1 to Q2



Now exchange the names of queues.



If pop operation needs to be performed then simply delete element from Q1. In the similar manner if push (50) operation can be implemented and finally -



1.4.2 : Linked List Representation of Queue

Q.73 Discuss about linked list implementation of queue ADT. [JNTU : Part B, Dec 18, Marks 5]

Ans. :

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
/*Declaration of Linked Queue data structure*/
typedef struct node
{
  int data;
  struct node *next;
 }Q;
  Q *front,*rear;
void main(void)
{
  char ans;
  int choice;
  void insert();
  Q *delet();
  void display(Q *);
  front=NULL;
  rear= NULL;
  do
  {
    printf("\n\tProgram For Queue Using Linked List\n");
    printf("\n  Main Menu");
    printf("\n1.Insert\n2.Delete \n3.Display");
    printf("\n Enter Your Choice");
    scanf("%d",&choice);
    switch(choice)
    {
     case 1 : insert();
          break;
     case 2 :front =delet();
          break;
     case 3 :display(front);
          break;
    default: printf("\nYou have entered Wrong
Choice\n");
          break;
  }
  printf("\nDo you want to continue?\n");
  flushall();
  ans = getch();
```

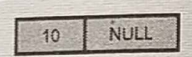
Queue is initialized to NULL in this way

Keeping track of front pointer

Scanned by CamScanner

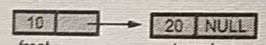
```
}while(ans == 'y' || ans == 'Y');
getch();
clrscr();
}
Q *get_node(Q *temp)
{
temp =(Q*) malloc(sizeof(Q));
temp->next = NULL;
return temp;
}
void insert()
{
 char ch;
 Q *temp;
 clrscr();
 temp =get_node(temp); /*allocates memory for temp node*/
 printf("\n\n\n\tInsert the element in the Queue\n");
 scanf("%d",&temp->data);

 if(front == NULL)/*creating first node*/
 {
  front= temp;
  rear=temp;
 }
 else /*attaching other nodes*/
  {
     rear->next=temp;
     rear=rear->next;
     }
}
int Qempty(Q *front)
{
 if(front==NULL)/*front is NULL means memory is not
          allocated to queue*/
 return 1;
 else
 return 0;
}
Q *delet()
{
 Q *temp;
 temp=front;
 if(Qempty(front))
 {
   printf("\n\n\t\tSorry!The Queue Is Empty\n");
   printf("\n Can not delete the element");
```

Suppose if 10 is inse rted in queue



front / rear /temp

The node to be inserted is as temp node. We are attaching previous rear to temp and marking that temp as **rear**



front temp/rear

Qempty (front) function returns 1 (true). Hence queue is empty.


```
 }
 else
  {
     printf("\n\tThe deleted Element Is %d ",temp->data);
     front=front->next;/*deleting the front node*/
     temp->next=NULL;
     free(temp);
  }
 return front;
 }
void display(Q *front)
{
 if(Qempty(front))
  printf("\n The Queue Is Empty\n");
 else
 {
  printf("\n\t The Display Of Queue Is \n ");
/*queue is displayed from front to rear*/
  for(;front !=rear->next;front=front->next)
     printf("\t%d ",front->data);
 }
 getch();
}
```

After delete operation, front is changed. Hence updated front pointer has to be returned from delete function.

Take the current front pointer and from **front** to **front! = rear->next** scan the queue each time by moving one node ahead.

Output

```
Program For Queue Using Linked List
   Main Menu
 1.Insert
 2.Delete
 3.Display
 Enter Your Choice1

   Insert the element in the Queue
 10
 Do you want to continue?
Program For Queue Using Linked List
   Main Menu
 1.Insert
 2.Delete
 3.Display
 Enter Your Choice1

   Insert the element in the Queue
 20
 Do you want to continue?
   Program For Queue Using Linked List
   Main Menu
 1.Insert
```


Scanned by CamScanner

2.Delete
3.Display
Enter Your Choice1
Insert the element in the Queue
30
Do you want to continue?

Program For Queue Using Linked List
Main Menu
1.Insert
2.Delete
3.Display
Enter Your Choice3

The Display Of Queue Is
10 20 30
Do you want to continue?

Q.74 What is the advantage of linked implementation of queue over the array implementation ? ☞ [JNTU : Part A, Marks 2]

Ans. :

(1) In the linked implementation, when the memory is allocated when a new node is inserted in the queue.

(2) Similarly when the element is to be removed from the queue, the nodes allocated get freed and is available for other allocation.

Q.75 What is the use of queue in operating system ?

OR Write any four applications of queues. ☞ [JNTU : Part A, Marks 2]

Ans. :

1. In operating system for scheduling jobs, priority queues are used.
2. In Local Area Network (LAN) multiple computers share few printers. Then these printers accumulate jobs in a queue. Thus printers can process multiple print commands with the help of queues.
3. For categorizing data, queues are used.
4. In simulation and modeling queues are used.
5. In computer networks, while broadcasting message, the message packets are accumulated in queue. And then these packets are forwarded.

Fill In the Blanks for Mid Term Exam

Q.1 Primitive data structures are generally ___ data types in programming language. ☞ [JNTU : Aug.-16]

Q.2 A linear collection of the nodes containing the data and pointer to next node is called _____.

Q.3 What is true about linked list ? (Static/dynamic)

Q.4 In comparison with sequential representation the computing time required for manipulating the list is _____.

Q.5 Linked list uses null pointer to signal _____.

Q.6 A list with no nodes is called _____.

Q.7 The address field of linked list contains address of _____.

Q.8 A doubly linked list facilitates list traversal in _____.

Q.9

```
void insert(struct node **q, int num)
    {
    struct node *temp;
    temp=malloc(sizeof(struct node));
    temp->data=num;
    temp->link=*q;
    *q=temp;
    }
```

This function is for inserting an element in the linked list at _____.

Q.10 In worst case the number of comparisons needed to search a singly linked list of length n for a given element is _____.

Q.11 To delete a node at the beginning of the list, the location of the list is modified as address of _____.

Q.12 ____ is a linked list in which there is no NULL link field

Q.13 The following node structure is used in _____.

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
}
```

Q.14 Each node in the doubly linked list has _____.

Q.15 The statement
head -> next -> next -> next = NULL
terminates the linked list after its _____ node.

Q.16 The memory address of first element is called _____.

Q.17 The technique of allocation and de-allocation of the memory as per the requirement is called _____.

Q.18 The linked list which can be processed in either direction is _____.

Q.19 The data structure required to evaluate a postfix expression is called _____. ☞ [JNTU : Aug.-16]

Q.20 Stack can be implemented using _____ and _____.

Q.21 The data structure used to evaluate the postfix expression is _____.

Q.22 The data structure used to reverse the given string _____.

Q.23 The data structure used to handle recursion _____.

Q.24 The postfix form of A*B+C is _____.

Q.25 What is the maximum number of parenthesis that appear on the stack at any one time when algorithm analyses ((())(())(())) ?

Q.26 Following sequence of operations is performed on a stack push(1), push(2)pop, pop, pop, push(2), pop. The sequence of poped out values are _____.

Q.27 The postfix form of the expression is _____.

Q.28 Polish notation is often called as _____.

Q.29 Reverse polish notation is often called as _____.

Q.30 The only one end that is used in the stack is called _____.

Q.31 The condition for stack being empty is checked by _____.

Q.32 If there is only one element in the stack then the value of the top is _____.

Q.33 The removing of the element from the stack is called _____.

Q.34 The insertion of the element onto the stack is called _____.

Q.35 What data structure would you most likely see in a non recursive implementation of a recursive algorithm ?

Q.36 The functions like factorial can be well demonstrated by _____ data structure.

Q.37 A linear list in which the elements can be inserted or deleted at either end but not middle is called _____.

Q.38 To simulate people waiting in a line, ____ data structure is preferable.

Q.39 In queue an element can be inserted and deleted by using ___and ____ respectively.

Q.40 In order to input the list of values and output them in order ____is used and in order to input the list of values and output them in opposite order _____ is used.

Multiple Choice Questions for Mid Term Exam

Q.1 Abstract data type means :

a separation of physical representation of data from its logical level

b a collection of data elements on which various operations for its storage and retrieval can be done

c a logical picture of data type of various data elements and just the specification of the operations

d none of these

Scanned by CamScanner

UNIT - II

2 Dictionaries and Hash Table Representation

Part I : Dictionaries

2.1 : Linear List Representation

Q.1 What is dictionary ? Explain the method of representing dictionary.

Ans. : **Definition** : Dictionary is a collection of pairs of key and value where every value is associated with the corresponding key.

The dictionary can be represented as a linear list. The linear list is a collection of pair and value. There are two method of representing linear list.

1. **Sorted Array -** An array data structure is used to implement the dictionary.
2. **Sorted Chain A linked list data structure is used to implement the dictionary.**

The dictionary can be represented as given below -

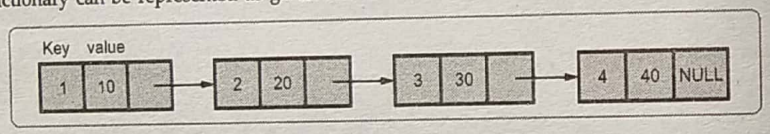


Fig. Q.1.1

Q.2 Give ADT for Dictionary.

Ans. : The ADT for dictionary can be given as below -

AbstractDataType Dictionary

{

Instance :

Dictionary is a collection of a pair key and value.

Operations :

Size() - It returns the size of the dictionary if it returns 0 then the dictionary is supposed to be empty.

Insert() - The pair of key and value is inserted in the dictionary.

Delete() - The pair of key and value is deleted from the dictionary.

Search() - The desired value is searched in the dictionary by using key.

}

Q.3 Give the applications of Dictionary.

Ans. :

1. Student registration application

A record of student can be inserted, deleted or updated from the dictionary. When a new student registers for the course his/her record is inserted in the dictionary. When the students get passed out/leaves the course his/her record is deleted from the dictionary. An instructor can update the record for changing the assignment test score.

Every record in this dictionary is considered as a key value pair. For instance -

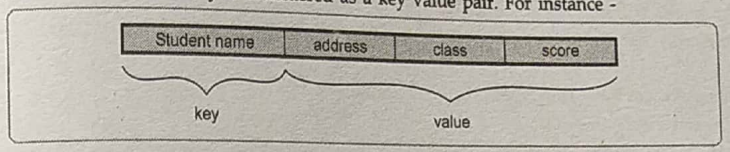


Fig. Q.3.1 Student record structure present in the dictionary

2. Telephone directory

This is an application of **dictionary with duplicates**. The dictionary with duplicates is similar to dictionary but it allows two or more pairs with same key. In the telephone directory the persons' surname acts as a **key** and his phone number, address and other information acts as **value**. These records are stored in alphabetical manner.

3. Word dictionary

This is an application of **dictionary with duplicates.** The spelling/meaning can be easily searched by 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.

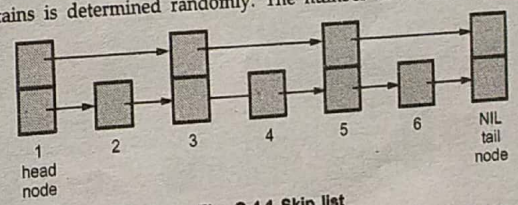


Fig. Q.4.1 Skip list

Scanned by CamScanner

- The skip list is an efficient implementation of dictionary using **sorted chain**. This is because in skip list each node consists of forward references of more than one node at a time. Following Figure shows 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 which is 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 -

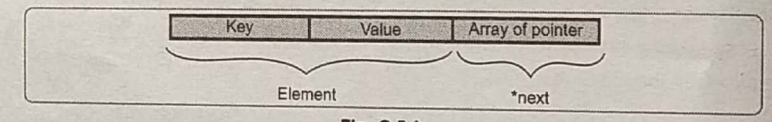


Fig. Q.5.1

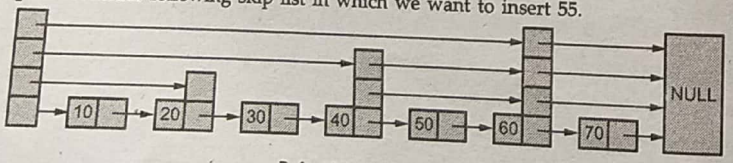
2.2.1 : Operations on Skip List

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

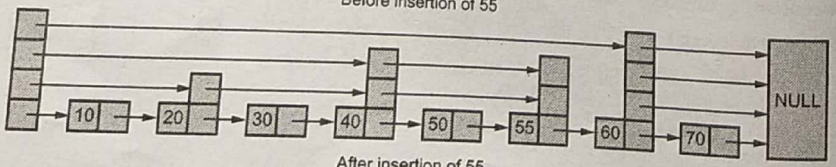
Ans. :

- 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

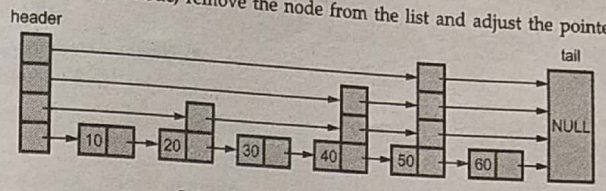


After insertion of 55

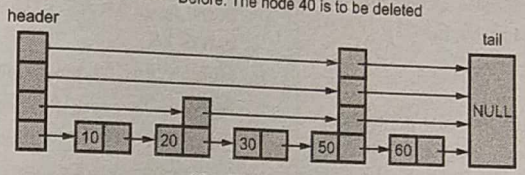
Q.7 Explain deletion of a node from the skip list with some suitable example.

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 3]

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 -

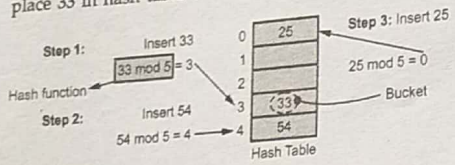
- Hash Table is a data structure used for storing and retrieving data very quickly. Insertion of data in the hash table is based on the key value. Hence every entry in the hash table is associated with some key. For example for storing an employee record in the hash table the employee ID will work as 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.

Scanned by CamScanner

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.

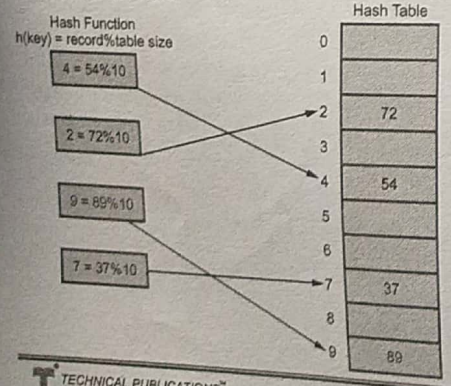
2.3.1 : Hash Functions

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 in detail ☞ [JNTU : Part B, Marks 5]

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 where A is 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

$$h(k) = \lfloor m \{ kA \} \rfloor$$

Fractional part

Donald Knuth suggested to use A = 0.61803398987

Example :

Let key k = 107, assume m = 50.

A = 0.61803398987

$$h(k) = \lfloor m * 107 * 0.61803398987 \rfloor$$

66.12

0.12 Fractional part

h(k) = 50 * 0.12

= 6

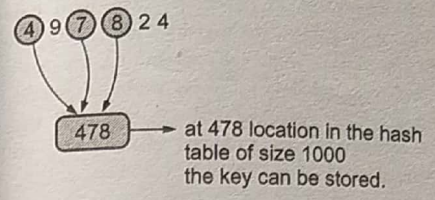
h(k) = 6

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 address location in hash table.

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



(4) 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**

$$\therefore \quad (3111)^2$$

9 6 7 8 3 2 1

For the hash table of size of 1000

H(3111) = 783

(5) 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

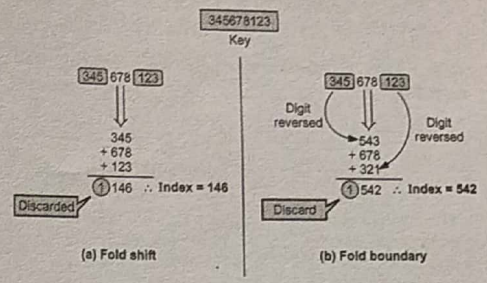


Fig. Q.14.1 Folding techniques

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.

Scanned by CamScanner

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. Chaning without replacement

In collision handling method chaining is a conc which introduces an additional field with data chain. A separate chain table is maintained colliding data. When collision occurs we store second colliding data by linear probing method. address of this colliding data can be stored with first colliding element in the chain table, witho replacement.

For example consider elements, 131, 3, 4, 21, 61, 6, 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 maintained the number who demands for location First number 131 comes we will place at index Next comes 21 but collision occurs so by line probing we will place 21 at index 2, and chain maintained by writing 2 in chain table at index similarly next comes 61 by linear probing we c place 61 at index 5 and chain will be maintained index 2. Thus any element which gives hash key a will be stored by linear probing at empty location b a chain is maintained so that traversing the ha table will be efficient.

2. Chaining with replacement

As previous method has a drawback of loosing t meaning of the hash function, to overcome t drawback the method known as chaining w replacement is introduced. Let us discuss the examp to understandsz the method. Suppose we have 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 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 wherever 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 ats 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 -

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

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.

Scanned by CamScanner

We will fill the hash table step by step

37%10 = 7
90%10 = 0
55%10 = 5
22%10 = 2
11%10 = 1

0	90
1	11
2	22
3	
4	
5	55
6	
7	37
8	
9	

Now if we want to place 17 a collision will occur as 17%10 =7 and bucket 7 has already an element 37. Hence we will apply quadratic probing to insert this record in the hash table.

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

we will choose value i = 0, 1, 2, ..., whichever is applicable.

Consider i = 0 then

$(17+0^2)\ \%10 = 7$

$(17+1^2)\ \%10 = 8$, when i = 1

The bucket 8 is empty hence we will place the element at index 8.

Then comes 49 which will be placed at index 9.

49%10 = 9

0	90
1	11
2	22
3	
4	
5	55
6	
7	37
8	17
9	49

Fig. Q.20.3

Now to place 87 we will use quadratic probing.

(87 + 0)%10 = 7

(87 + 1)%10 = 8 ... but already occupied

$(87 + 2^2)\%10 = 1$ ... already occupied

$(87 + 3^2)\%10 = 6$ ... this slot is free

∴ We place 87 at 6th index.

It is observed that if we want to place all the necessary elements in the hash table the size of divisor (m) should be twice as large as total number of elements.

0	90
1	11
2	22
3	
4	
5	55
6	87
7	37
8	17
9	49

Fig. Q.20.4

3. Double hashing

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

There are two important rules to be followed for the second function :

- It must never evaluate to zero.
- Must make sure that all cells can be probed.

The formula to be used for double hashing is

$H_1(key)$ = key mod tablesize

$H_2(key) = M - (\text{ key mod } M\ )$

where M is a prime number smaller than the size of the table.

Consider the following elements to be placed in the hash table of size 10

37, 90, 45, 22, 17, 49, 55

Initially insert the elements using the formula for $H_1(key)$.

Insert 37, 90, 45, 22.

$H_1(37) = 37\%10 = 7$
$H_1(90) = 90\%10 = 0$
$H_1(45) = 45\%10 = 5$
$H_1(22) = 22\%10 = 2$
$H_1(49) = 49\%10 = 9$

0	90
1	
2	22
3	
4	
5	45
6	
7	37
8	
9	49

Now if 17 is to be inserted then

$H_1(17) = 17\%10 = 7$

$H_2(key) = M - (key\%M)$

Here M is a prime number smaller than the size of the table. Prime number smaller than table size 10 is 7.

Hence M = 7

$H_2(17) = 7 - (17\%7) = 7 - 3 = 4$

That means we have to insert the element 17 at 4 places from 37. In short we have to take 4 jumps. Therefore the 17 will be placed at index 1.

Now to insert number 55.

$H_1(55) = 55\%10 = 5$ ... collision

$H_2(55) = 7 - (55\%7) = 7 - 6 = 1$

0	90
1	17
2	22
3	
4	
5	45
6	
7	37
8	
9	49

Fig. Q.20.5

That means we have to take one jump from index 5 to place 55. Finally the hash table will be -

0	90
1	17
2	22
3	
4	
5	45
6	55
7	37
8	
9	49

Q.21 Explain in detail about linear probing and quadratic probing.

[JNTU : Part B, May-18, Marks 5]

Ans. : Refer Q.20.

Q.22 What is overflow in hashing ?

[JNTU : Part A, Marks 2]

Ans. : The situation in which there is no room for a new pair in the hash table is called overflow.

Q.23 What is the major problem in linear probing ? **[JNTU : Part A,Marks 2]**

Ans. : One major problem in linear probing is primary clustering. Primary clustering is the process in which a block of data is formed in the hash table when collision is resolved.

Q.24 What is the advantage of chained hash table over open addressing scheme ?

[JNTU : Part A, Marks 2]

Ans. : The deletion of particular record from the hash function becomes easier.

Q.25 State some major drawback of chaining.

[JNTU : Part A, Marks 2]

Ans. :

(1) Requirement of additional data structure is the major drawback of chaining.

(2) On inserting any new element in the hash table, the adjustment in chaining need to be done.

Scanned by CamScanner

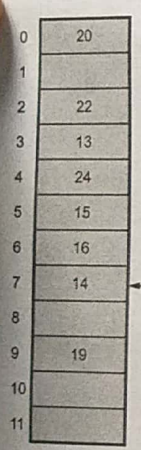
Q.26 What do you understand by collision in hashing ? Represent the following keys in memory using linear probing with or without replacement. Use modulo (10) as your hashing function : (24, 13, 16, 15, 19, 20, 22, 14, 17, 26, 84, 96)

☞ [JNTU : Part B]

Ans. : Collision in hashing - Refer Q.19.

i) Linear probing with replacement

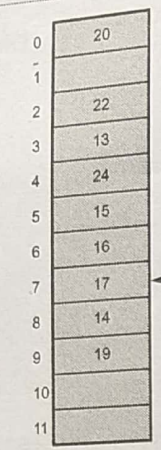
We will consider the hash function as modulo (10) for the hash table size 12, that is from 0 to 11. In linear probing with replacement, we first find the probable position of the key element using hash function. If the location which we obtain from hash function is empty then place the corresponding key element at that location. If the location is not empty and the key element which is present at that location belongs to that location only then, move down in search of empty slot. Place the record at the empty slot.



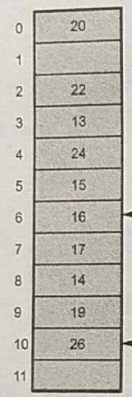
Place 14 here because 14 % 10 = 4. The index 4 contain the element 24. This is a proper record at its place. Hence to place 14 we have to move down in search of an empty slot.

If the location contains a record which does not belong to that location then replace that record by the current key element. Place the replaced record at some empty slot, which can be obtained by moving linearly down.

We get the empty location at index 7. Hence 14 is placed at index 7.

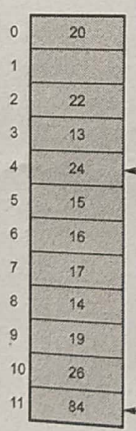


17 % 10 = 7. This position was occupied by 14. But 14 is not the proper record at this place. Hence replace 14 by 17. Then place 14 at next empty slot.



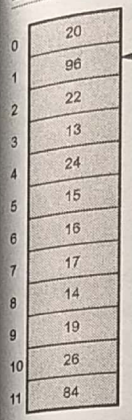
26 % 10 = 6. But 16 is a proper record at this place.

Hence at next empty location 26 is placed.

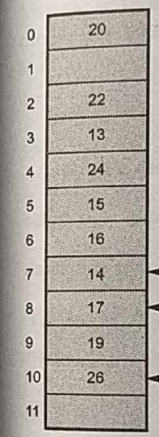


84 % 10 = 4. But 24 is occupying its own location

Then by moving linearly down we can place 84 at the empty location found.



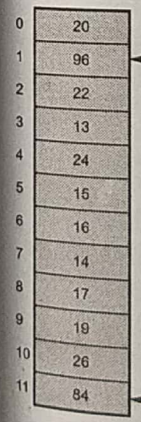
As 96 % 10 = 6. But index 6 holds a record 16 which is correct for that location. By moving down linearly we get no empty slot. Hence we roll back and get the empty slot at index 1. Hence 96 will be replaced at index 1.



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.

84 % 10 = 4. But index 4 contains key element 24. Hence by linear probing, at empty slot 11, the element 84 is placed.

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, Nove.-09, Marks 8]

Ans. : For insertion operation we will use mod 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 double hashing choose M, Where M is a primary number whose value is less than table size.

Set M = 7

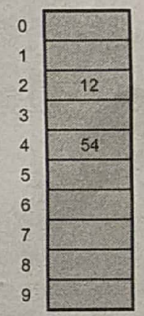


Fig. Q.27.1

$h_2(key) = M - (key \bmod M)$

$= 7 - (62\%7)$

$= 7 - 6$

$h_2(key) = 1$

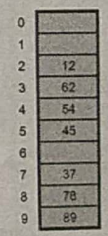


Fig. Q.27.2

Scanned by CamScanner

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.

$\therefore$ $H_2(key) = H_2(28) = 7 - (28\%7)$

$= 7 - 0$

$= 7$

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.

$\therefore$ $H_2(49) = 7 - (49\%7)$

$= 7$

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 after 7 jumps we will not get an empty slot. The only one empty slot. The only one empty slot remaining in hash table. Hence we will place 49 at index 0.

The hash table will be as shown in Fig. Q.28.4.

Q.28 What are the problems associated with quadratic probing.

[JNTU : Part B, Nove.08, Marks 8]

Ans. : One major problem associated with quadratic probing is that typically not all hash table slots will be on the probe sequence. Using $H(K, i) = i^2$ 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 addressing hash table with second hash functions h2 (x) = 7 - (xmod7)

[JNTU : Part B, Nov.-09, Marks 16]

Ans. : a) Open addressing hash table using quadratic probing

- Insert 3417.

3417%10 = 7

- Insert 3132.

3132%10 = 2

0	
1	
2	3132
3	7122
4	5344
5	
6	6796
7	3417
8	
9	5199

Fig. Q.29.1

- Insert 7122.

7122%10 = 2 Collision occurs.

Hence by quadratic probing,

$(7122+0^2)\ \%\ 10 = 2$

$(7122+1^2)\ \%\ 10 = 3$ empty slot, hence place it

- Insert 5199

5199 % 10 = 9

- Insert 5344

5344 % 10 = 4

- Insert 6796.

6796%10 = 6 (Refer Fig. Q.29.1)

- Insert 1893.

1893%10 = 3 collision occurs

$\therefore$ Apply quadratic probing

$(1893+1^2)\ \%\ 10 = 4$ collision occurs

$(1893+2^2)\ \%\ 10 = 7$ collision occurs

$(1893+3^2)\ \%\ 10 = 2$ collision occurs

$(1893+4^2)\ \%\ 10 = 9$ collision occurs

$(1893+5^2)\ \%\ 10 = 8$ empty slot.

0	
1	
2	3132
3	7122
4	5344
5	
6	6796
7	3417
8	1893
9	5199

Fig. Q.29.2

$\therefore$ Insert 1893 at index 8. The hash table will then be -

b) Open addressing hash table with second hash functions h2(x)=7-(xmod7)

- Insert 3417.

3417%10 = 7

- Insert 3132.

3132%10 = 2

- Insert 7122.

0	
1	
2	3132
3	1893
4	5344
5	
6	7122
7	3417
8	6796
9	5199

Fig. Q.29.3

7122%10 = 2 Collision occurs.

$\therefore$ $h_2(7122) = 7 - (7122\%7)$

$= 7 - 3 = 4$

Take 4 jumps from 3132 and place 7122 at index 6.

- Insert 5199.

5199%10 = 9

- Insert 5344.

5344%10 = 4

- Insert 6796.

6796%10 = 6 Collision occurs.

$\therefore$ $h_2(6796) = 7 - (6796\%7) = 7 - 6 = 1$

By taking 1 jump from 7122 we again get collision. Hence store record at next empty location. That means at index 8.

- Insert 1893.

1893 % 10 = 3

Q.30 What is rehashing ?

[JNTU : Part A, Marks 3]

Ans. : Rehashing is a technique in which the table is resized. That means the size of the table is doubled by creating a new table. The total size of the table is usually a prime number. Following are the situations in which the rehashing is required -

1. When table is completely full.
2. With quadratic probing the table is filled half.
3. When insertions fail due to overflow.

Scanned by CamScanner

Q.31 Explain rehashing technique with suitable example. ☞[JNTU : Part B, Marks 5]

Ans. : Consider we have to insert the elements 37, 90, 55, 22, 17, 49 and 87. The table size is 10 and will use hash function,

H(key) = key mod tablesize

37%10 = 7
90%10 = 0
55%10 = 5
22%10 = 2
17%10 = 7 Collision solved by linear probing
49%10 = 9

0	90
1	
2	22
3	
4	
5	55
6	
7	37
8	17
9	49

Now this table is almost full and if we try to insert more elements collisions will occur and eventually further insertions will fail. Hence we will rehash by doubling the table size. The old table size is 10 then we should double this size for new table, that becomes 20. But 20 is not a prime number, we will prefer to make the table size as 23. And new hash function will be

H(key) = key mod 23

37%23 = 14
90%23 = 21
55%23 = 9
22%23 = 22
17%23 = 17
49%23 = 3
87%23 = 18

0	
1	
2	
3	49
4	
5	
6	
7	
8	
9	55
10	
11	
12	
13	
14	37
15	
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 2]

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

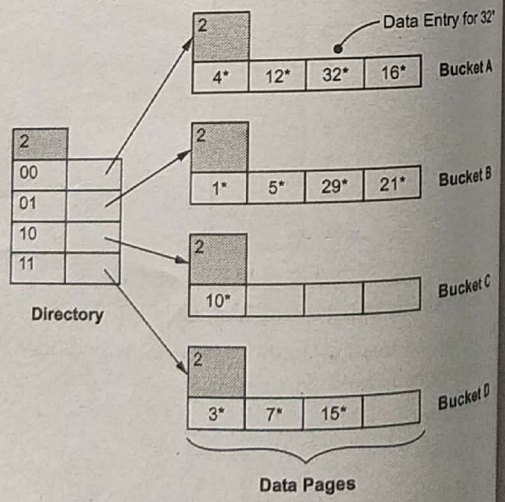


Fig. Q.33.1 Extendible hash file

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 insetion of 20* is as shown in following Fig. Q.33.2.

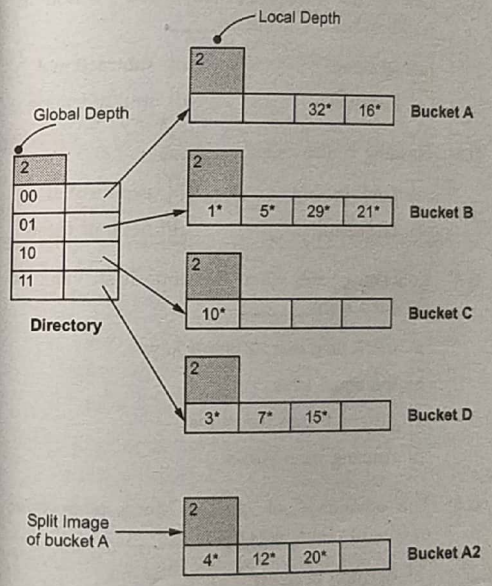


Fig. Q.33.2 During insertion process

- 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 insetion 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.

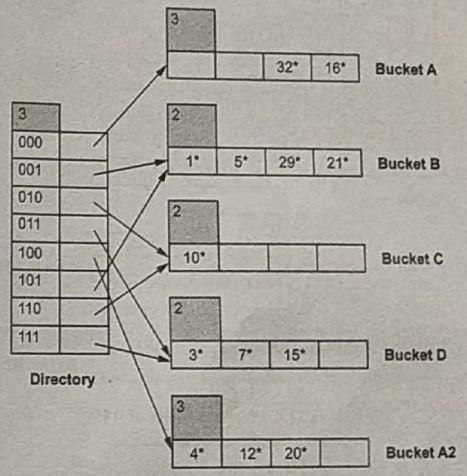


Fig. Q.33.3 After insertion of 20*

- 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

Scanned by CamScanner

After insertion of 11* we get the scenario as follows

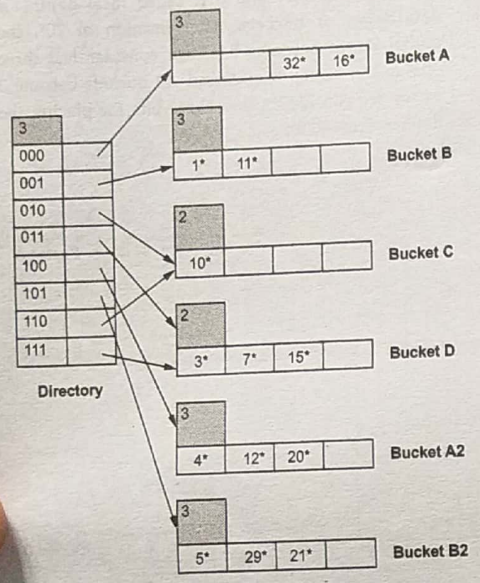


Fig. Q.33.4 After insertion of 11*

Fill in the Blanks for Mid Term Exam

Q.1 Assuming that the hash function for a table works well and the size of the hash table is reasonably large compared to the number of items in the table, the expected (average) time needed to find an item in a hash table containing n items is_____.

Q.2 A hash function f defined as f(key) = key mod 7,with linear probling,insert the keys 37, 38, 72, 48, 98, 11, 56 into a table. The 11 will be stored in the location_____.

Q.3 _____ occurs when the hashing function produces the same address for two different keys.

Q.4 One major problem of linear probing is_____.

Q.5 Rehashing is a technique in which_____.

Q.6 Double hashing is a technique in which_____.

Q.7 Space gets doubled in _____ collision resolution technique.

Q.8 ____ is hashing technique makes use of bit prefix

Q.9 Primary clustering is associated with_____.

Q.10 The sorted array and sorted chain methods of linear list representation are used in _____

Multiple Choice Questions for Mid Term Exam

Q.1 In hashing a record is located using ___.

a key b function
c index d none of these

Q.2 One of the most commonly used method in building hash function is_____.

a addition b subtraction
c division d multiplication

Q.3 Hashing is also called as _____.

a randomizing b exponential
c arithmetic d none of these

Q.4 Collision occurs when the same hash value is obtained from_____.

a more then one different keys
b the equal keys
c different hash functions
d resizing hash tables

Q.5 The advantage of chained hash table over open addressing scheme is_____.

a deletion is easier
b space used is less
c worst case complexity of search operation is less
d none of these

Q.6 In following collision resolution technique the key causing collision is placed at first vacant position_____.

a chaining
b quadratic probing
c double hashing
d linear probing

Q.7 Which one is the negative side effect of linear probing ?

a Too many values hash to the same key and hence create long chain
b Too many linked list in the table which increases the time complexity
c Too many values, all hash to the same area of the hash table and cause clustering
d None of these

Q.8 If a hash table has n cells and there are 10*n objects. Use of linear probing_____.

a efficient
b inefficient
c depends on the keys
d depends on the hash function being used

Q.9 Adding the objects to the hash table for the list of elements which are already sorted give following result

a Placing of the elements in the table becomes time efficient
b There is no need to resize the table
c Placing of the elements in the table becomes time efficient
d No effect

Q.10 A good hashing function must have_____.

a minimize collisions
b easy and quick to compute
c distribute the keys evenly over the hash table
d all of the above.

Q.11 In which of the hashing function the arithmetic or logical function is applied on different field value to calculate hash address ?

a Division method b Folding
c Open hashing d Chaining

Q.12 Following hashing technique is preferred when huge amount of data is available ___.

a Double hashing
b Rehashing
c Quadratic probing
d Extensible hashing

Q.13 In browser programs for caching the web pages ____is used.

a linked list b arrays or buffers
c hashing d none of these

Q.14 What do you understand by collision resolution by open addressing ?

a When collision happens we create a new memory location outside the existing table and use a chain to link to the new memory location
b When collision occurs we enlarge the hash table.
c When collision occurs we look for an unoccupied memory location in the existing hash table.
d We use an extra hash table to collect all the collided data.

Q.15 Which is the following statements denotes the meaning of the term collision ?

a A hash function produces the same address for two different keys. H(key1) = h(key2)
b Two different hash functions produces the same address for a given key. H1(key) = H2(key)
c Two different hash functions produce the same result for the two different keys. H1(key1) = H2(key2)

Scanned by CamScanner

UNIT - III

3 Search Trees

3.1 : Binary Search Trees

Q.1 Define the term - Tree

[JNTU : Part A, Marks 2]

Ans. : Definition of a tree :

A tree is a finite set of one or more nodes such that -

i) **Root node :** It is a specially designated node.

ii) There are remaining n nodes which can be partitioned into disjoint sets $T_1, T_2, T_3, \ldots T_n$ where $T_1, T_2, T_3, \ldots T_n$ are called subtrees of the root.

The concept of tree can be represented by following figure -

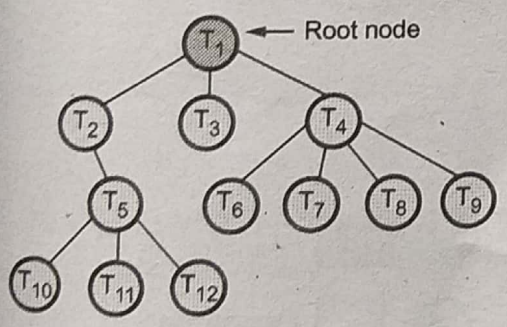


Fig. Q.1.1 Tree

Q.2 Explain the terms - i) Leaves ii) Degree of node iii) Degree of tree iv) Level of tree v) Height of Tree

[JNTU : Part B, Marks 5]

Ans. :

i) Leaves

These are the terminal nodes of the tree.

For example

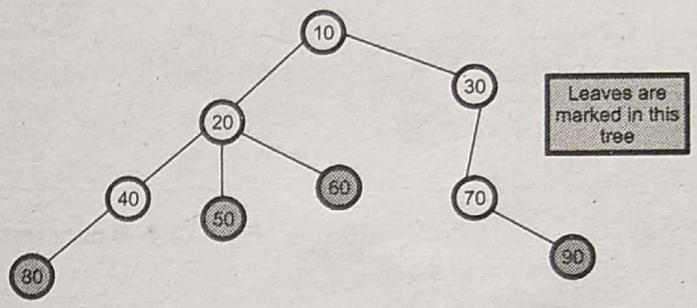


Fig. Q.2.1 A tree with leaf nodes

ii) Degree of the node

The total number of sub-trees attached to that node is called the degree of a node.

For example

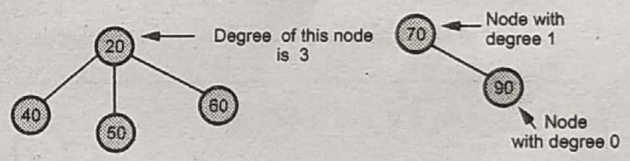


Fig. Q.2.2 Binary tree

iii) Degree of tree

The maximum degree in the tree is degree of tree.

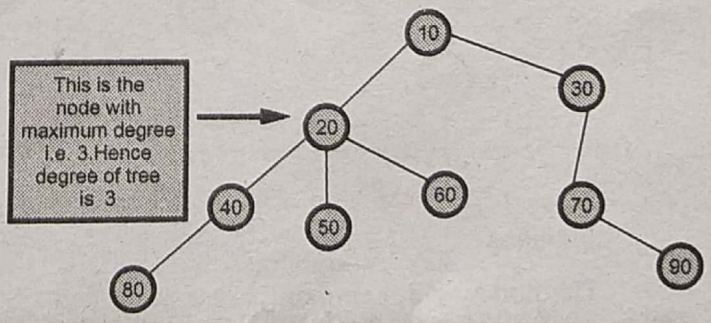


Fig. Q.2.3 Binary tree

iv) Level of the tree

The root node is always considered at level zero.

Scanned by CamScanner

The adjacent nodes to root are supposed to be at level 1 and so on.

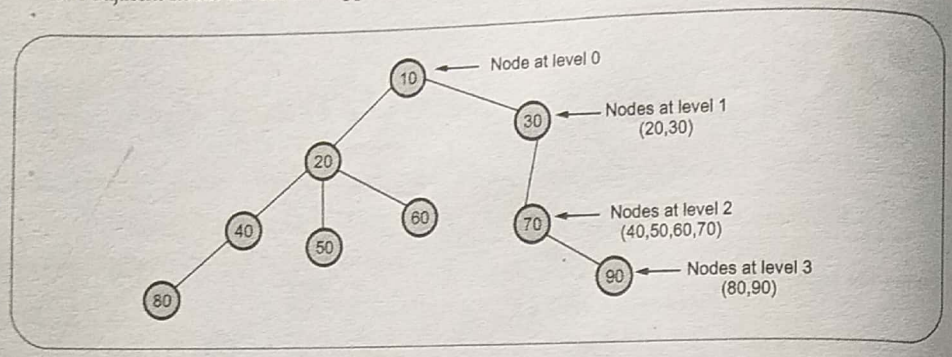


Fig. Q.2.4 Binary tree in which levels are shown

v) Height of the tree ☞[JNTU : Part A, Dec.-17, Marks 2]

The maximum level is the height of the tree. In Fig. Q.2.4 the height of tree is 3. Sometimes height of the tree is also called **depth** of tree.

Q.3 Explain the terms - i) Internal and external nodes ii) Sibling

Ans. :

i) Internal and external nodes

Leaf node means a node having no child node. As leaf nodes are not having further links, we call leaf nodes **external nodes** and non leaf nodes are called **internal nodes**.

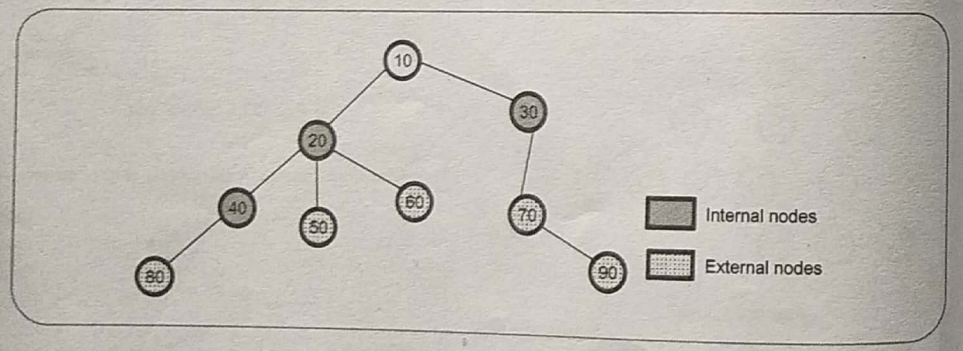
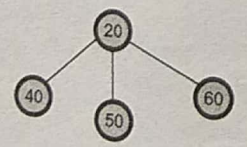


Fig. Q.3.1 Representing internal and external nodes

ii) Sibling

The nodes with common parent are called siblings or brothers.

For example



The nodes 40, 50 and 60 are siblings of each other.

Fig. Q.3.2 Representation of sibling

Q.4 Define tree. Explain all terms associated with trees.

Ans. : Refer Q.1. ☞[JNTU : Dec.-18, Marks 5]

Q.5 Define height of binary tree.

Ans. : Refer Q.2. ☞[JNTU : Dec.-17, Marks 3]

Q.6 Explain binary tree with an example.

Ans. : ☞[JNTU : Part B, Dec.-16, Marks 5]

Definition of a binary tree : *A binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the left sub-tree and right sub-tree.*

The binary tree can be as shown below.

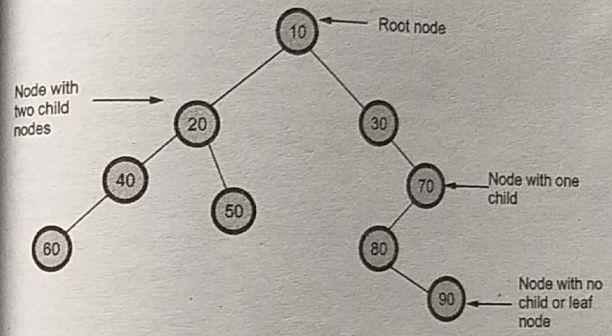


Fig. Q.6.1 Binary tree

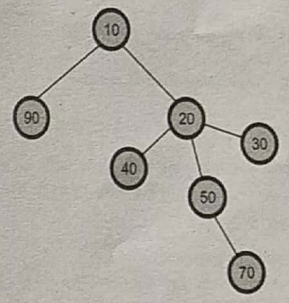


Fig. Q.6.2 Not a binary tree

Q.7 Write an algorithm to declare nodes of a tree structure. ☞[JNTU : Part B, Marks 5]

Ans. :

Algorithm :

1. Declare a C structure for the node of a binary tree

```
typedef struct node
{
    int data;
    struct node *left;
    struct node *right;
}bin;
```

2. Create a node by allocating memory by using the dynamic memory allocation.

```
New=(bin*)malloc(sizeof(bin));
```

3. Initialize the Left and Right child pointers to NULL.

```
New->left=NULL;
New->right=NULL;
```

Thus a single node gets created. This is the first node of the tree, hence it is called as the root node.

```
root=New;
```


Scanned by CamScanner

4. Attach the next subsequent nodes to either as a left child or a right child to the corresponding parent nodes depending on the user's choice.

Q.8 What are the applications of binary tree ?
[JNTU : Part B, Dec.-14, Marks 2]

Ans. : Binary tree is used in various applications such as -

1. Binary search tree 2. Game tree

3. Expression tree 4. Threaded binary tree.

Q.9 What are the tasks that are carried out in tree traversal ? **[JNTU : Part A, Marks 3]**

Ans. :

1) Visiting a node

2) Traversing the left subtree

3) Traversing the right subtree

Q.10 Explain tree traversals with an example
[JNTU : Part B, Nov.-15, Marks 5]

Ans. :

1. Inorder traversal :

C-B-A-D-E is the inorder traversal i.e. first we go towards the leftmost node i.e C so print that node C. Then go back to the node B and print B. Then root node A then move towards the right sub-tree print D and finally E. Thus we are following the tracing sequence of LDR. This type of traversal is called inorder traversal. The basic principal is to traverse left sub-tree then root and then the right sub-tree.

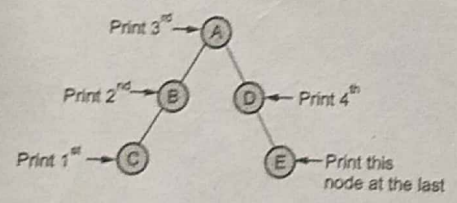


Fig. Q.10.1 Inorder traversal

2. Preorder traversal :

A-B-C-D-E is the preorder traversal of the Fig. Q.19.2. We are following DLR path. i.e data at the root node will be printed first then we move on the left sub-tree and go on printing the data till we reach to the leftmost node.Print the data at that node and move to the right subtree. Follow the same principle at each sub-tree and go on printing the data accordingly.

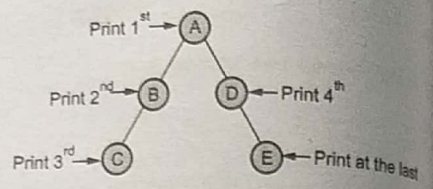


Fig. Q.10.2 Preorder traversal

3. Postorder traversal :

From Fig. Q.10.3 the postorder sequence is C-D-B-E-A. In the postorder traversal we are following the LRD principle i.e. move to the leftmost node check if right sub-tree is there or not if not then print the leftmost node, if right subtree is there move towards the rightmost node. The key idea here is that at each sub-tree we are following the LRD principle and print the data accordingly.

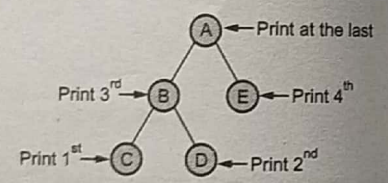


Fig. Q.10.3 Postorder traversal

Q.11 Write about binary search tree
[JNTU : Part A, Dec 16, Marks 3]

Ans. : Definition :

Binary search tree is a binary tree in which the left subtree contains all the nodes with values less than the root node and right subtree contains all the nodes with values greater than root node.

For example

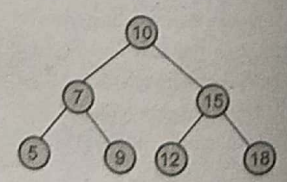


Fig. Q.11.1 Binary search tree

Q.12 Write the properties of binary search trees.
[JNTU : Part A, May-15, Marks 2]

Ans. : In binary search tree

1. Every left node has a value less than its parent node.

2. Every right node has a value greater than its parent node.

Q.13 Derive binary search tree ADT.
[JNTU : Part B,Marks 7]

Ans. : AbstractDataType BST

{

Instances : Binary search tree is a binary tree in which value of left child is less than its parent node and value of right child is greater than its parent node.

Operations :

1. **Create** - Using this operation a binary search tree can be created.

2. **Display** - This operation is used for displaying all the nodes of BST.

3. **Insert** - For insertion of any node in the BST, this operation is useful.

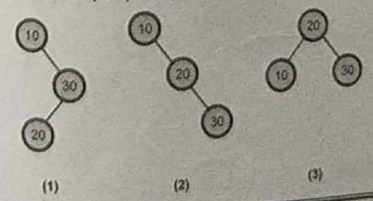
4. **Delete** - Using this operation, any node from BST can be deleted.

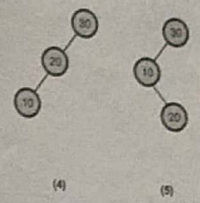
5. **Search** - This function is used for searching any node from BST.

}

Q.14 How many Trees are possible with 3 nodes?
[JNTU : Part A, Marks 2]

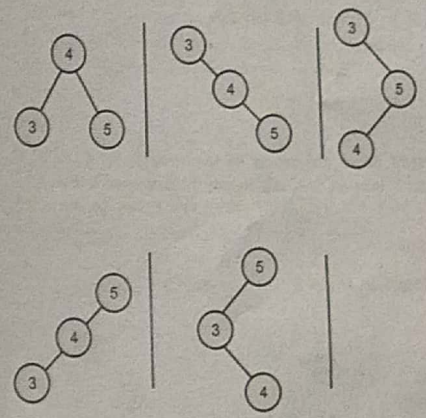
Ans. : For 3 nodes there are 2^3 - 3 trees possible. That means 5 such trees can be drawn. For example - If there are 3 nodes with the value 10, 20, 30 then





Q.15 Draw all possible binary tree whose inorder traversal is 3, 4, 5. **[JNTU : Part B, Marks 5]**

Ans. :



Q.16 Write an insertion algorithm and C++ code for insertion of node in binary search tree
[JNTU : Part B, Marks 5]

Ans. :

Algorithm :

1. Read the value for the node which is to be created, and store it in a node called **New**.

2. Initially if (root!=NULL) then root=**New**.

3. Again read the next value of node created in **New**.

4. If (New->value < root->value) then attach **New** node as a left child of root otherwise attach **New** node as a right child of root.

5. Repeat step 3 and 4 for constructing required binary search tree completely.

Scanned by CamScanner

```
void bintree::insert(node  *root,node *New)
{
   if(New->data<root->data)
    {
      if(root->left==NULL)
       root->left=New;
      else
      insert(root->left,New);
    }
   if(New->data>root->data)
    {
    if(root->right==NULL)
          root->right=New;
    else
          insert(root->right,New);
    }
}
```

Attaching left node to the current node.

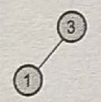
If the root node has already some left child, then move on to left subtree

Q.17 Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.
[JNTU : Part B, Jan.-14, Marks 7]

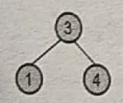
Ans. :

Step 1 : Insert 3

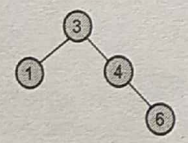
Step 2 : Insert 1



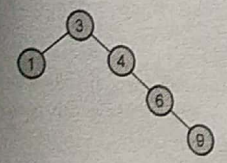
Step 3 : Insert 4



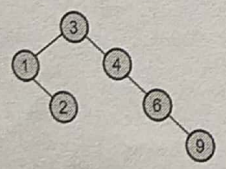
Step 4 : Insert 6



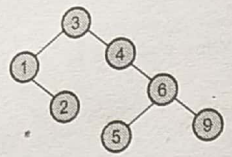
Step 5 : Insert 9



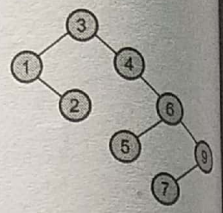
Step 6 : Insert 2



Step 7 : Insert 5



Step 8 : Insert 7

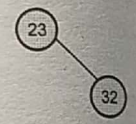


Q.18 Create binary search tree for the following elements (23, 32, 24, 36, 15, 12, 39, 2, 19)
[JNTU : Dec.-17, Marks 5]

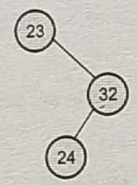
Ans. :

Step 1 :

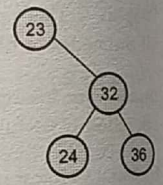
Step 2 :



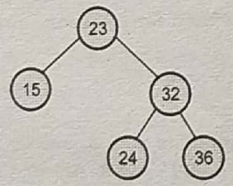
Step 3 :



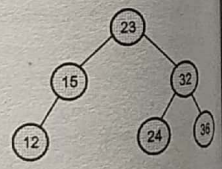
Step 4 :



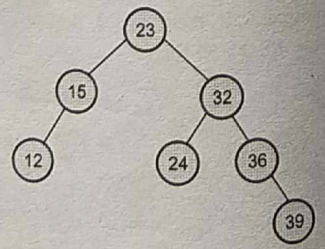
Step 5 :



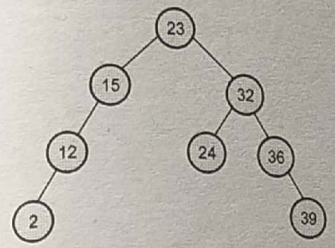
Step 6 :



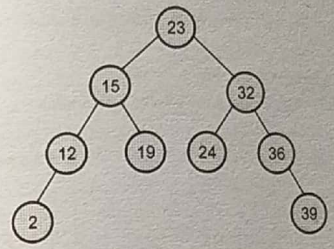
Step 7 :



Step 8 :



Step 9 :



Q.19 Explain three cases for deletion of a node from binary search tree **[JNTU : Part B, Marks 5]**

Ans. :

Deletion of an element from the binary tree

For deletion of any node from binary search tree there are three cases which are possible.

i. Deletion of leaf node.

ii. Deletion of a node having one child.

iii. Deletion of a node having two children.

Let us discuss the above cases one by one.

i. Deletion of leaf node

This is the simplest deletion, in which we set the left or right pointer of parent node as NULL.

From the tree in Fig. Q.19.1, if, we want to delete the node having value 6 then we will set left pointer of its parent node as NULL. That is left pointer of node having value 8 is set to NULL. Refer Fig. Q.19.2.

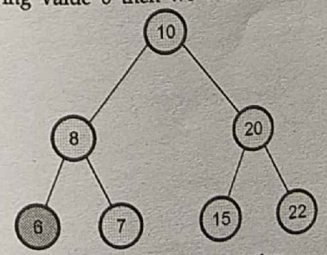


Fig. Q.19.1 Before deletion

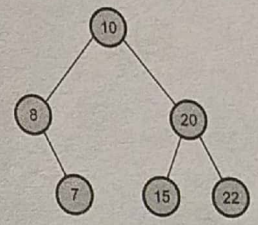


Fig. Q.19.2 After deletion

ii. Deletion of a node having one child

To explain this kind of deletion, consider a tree as given in Fig. Q.19.3.

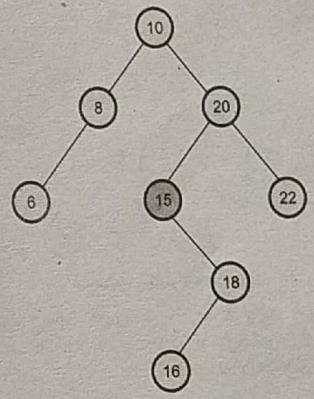


Fig. Q.19.3

If we want to delete the node 15, then we will simply copy node 18 at place of 15 and then set the node free. The inorder successor is always copied at the position of a node being deleted.

iii. The node having two children

Again, let us take some example for discussing this kind of deletion.

Let us consider that we want to delete node having value 7 (Fig. Q.19.4). We will then find out the inorder successor of node 7. The inorder succesor will be simply copied at location of node 7. Thats it !

Scanned by CamScanner

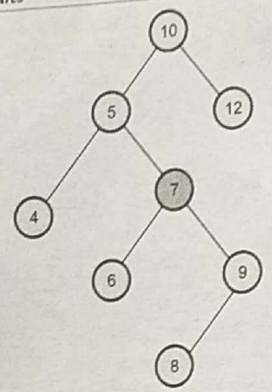


Fig. Q.19.4 Before deletion

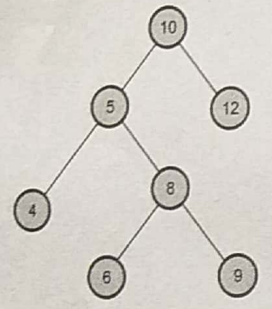


Fig. Q.19.5 After deletion

That means copy 8 at the position where value of node is 7. Set left pointer of 9 as NULL. This completes the deletion procedure.

Q.20 Write a C function for searching a node from a binary search tree. ☞ [JNTU : Part B, Marks 5]

Ans. :

```
void del(node *root,int key)
{
node *temp_succ;
 if(root==NULL)
 printf("Tree is not created");
 else
 {
 temp=root;
 search(&temp,key,&parent);
//temp node is to be deleted
/*deleting a node with two children*/
if(temp->left!=NULL && temp->right!=NULL)
{
 parent=temp;
temp_succ=temp->right;
```

temp is node which is to be deleted. Parent is a node which keeps track of parent node. temp and temp_succ is for keeping track of successor of temp node.

```
while(temp_succ->left!=NULL)
 {
  parent=temp_succ;
  temp_succ=temp_succ->left;
      <----------Moving to leftmost sub-tree
 }
 temp->data=temp_succ->data;
        //copying the immediate successor
 temp->right=NULL;
printf(" Now Deleted it!");
 return;
}
```

Copying the successor node's data to temp node. Thus contents of temp node gets deleted.

```
/*deleting a node having only one child*/
/*The node to be deleted has left child*/
if(temp->left!=NULL && temp->right==NULL)
{
 if(parent->left==temp)
      <------------ Finding the parent of temp
  parent->left=temp->left;
 else
  parent->right=temp->left;
 temp=NULL;
 free (temp);
 printf (" Now Deleted it!");
 return;
}

/*The node to be deleted has right child*/
if(temp->left==NULL && temp->right!=NULL)
{
 if(parent->left==temp)
  parent->left=temp->right;
 else
  parent->right=temp->right;
 temp=NULL;
 free (temp);
 printf(" Now Deleted it!");
 return;
}
/*deleting a node which is having no child*/
```



```
if(temp->left==NULL && temp->right==NULL)
 {
    if(parent->left==temp)
     parent->left=NULL;
    else
     parent->right=NULL;
 printf(" Now Deleted it!");
 return;
 }
}
}
```

Q.21 Write a C function for searching a node from a binary search tree. ☞ [JNTU : Part B, Marks 5]

Ans. :

- In searching, the node which we want to search is called a **key node.**
- The key node will be compared with each node starting from root node if value of key node is greater than current node then we search for it on right sub-branch otherwise on left sub-branch. If we reach to leaf node and still we do not get the value of key node then we declare **"node is not present in the tree".**

In the above tree, if we want to search for value 9. Then we will compare 9 with root node 10. As 9 is less than 10 we will search on left sub-branch. Now compare 9 with 5, but 9 is greater than 5. So we will move on right sub-branch. Now compare 9 with 8 but as 9 is greater than 8 we will move on right sub-branch. As the node we will get holds the value 9. Thus the desired node can be searched.

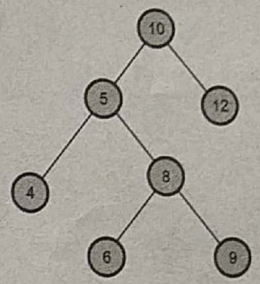


Fig. Q.21.1 After deletion

C Function for Searching a node

```
void search(node **temp,int key,node **parent)
{
 if(*temp==NULL)
  printf("Tree is Not Created");
 else
 {
  while(*temp!=NULL)
  {
     if((*temp)->data==key)
     {
      printf("Element is Present");
       break;
     }
  *parent=*temp;//stores the parent value<------
  if((*temp)->data>key) <----------------------
     *temp=(*temp)->left; <----------------------
  else
     *temp=(*temp)->right;
  }
 }
 return;
}
```

- Marking the parent node
- If current node is greater than key
- Search for the left subtree

Q.22 Draw the binary search tree that results from inserting the integers 57, 85, 35, 9, 47, 20, 26, 99, 93, 10 starting with 57 and ending with 10. ☞ [JNTU H : Part A, Dec.-14, Marks 3]

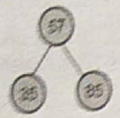
Scanned by CamScanner

Ans. :

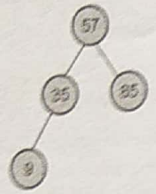
Step 1 : Insert 57

Step 2 : Insert 85

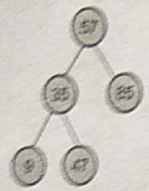
Step 3 : Insert 35



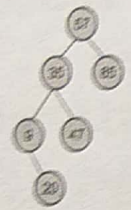
Step 4 : Insert 9



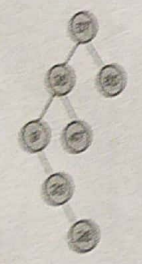
Step 5 : Insert 47



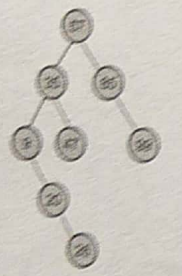
Step 6 : Insert 20



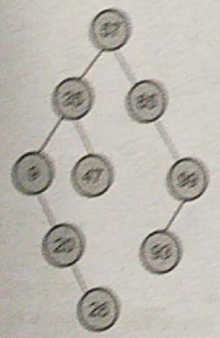
Step 7 : Insert 26



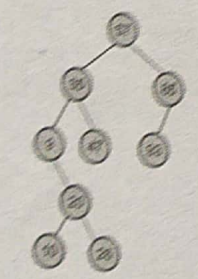
Step 8 : Insert 99



Step 9 : Insert 93



Step 10 : Insert 10



Required binary tree

Q.23 Write a pseudo C routine to find the depth of the binary tree. [JNTU : Part B]

Ans. :

```
int tree::Height(btree *temp)
{
 int d1,d2;
 if(temp==NULL)
   return 0;
 if(temp->left==NULL && temp->right==NULL)
   return 0;
 d1=Height(temp->left);//computing height of leftsubtree
 d2=Height(temp->right);//computing height of rightsubtree
  if(d1>d2)
   return d1+1;
  else
   return d2+1;
//maximum depth=height of the tree
}
/*
-----------------------------------------------
   The display function
-----------------------------------------------
*/

void tree::display()
{
  cout<<"The height of the tree is "<<Height(root);
}
/*
```

Q.24 Construct a binary tree having following traversal sequences :

Preorder traversal : A B C D E F G H I

Inorder traversal : B C A E D G H F I

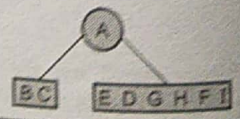
[JNTU(H) : Part B, Dec.-16, Marks 5]

Ans. : The first position element in preorder sequence represents root node. Hence A will be the root node. This A will then be located in inorder sequence. The left sublist to A forms left sub-branch and right sublist to A forms right sub-branch.

Step 1 :

Preorder : [A] B C D E F G H I

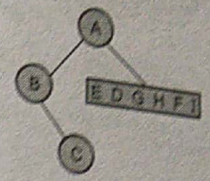
Inorder : B C [A] E D G H F I



Step 2 :

Preorder: [B] C

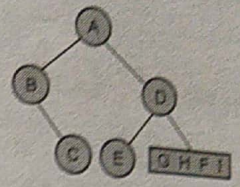
Inorder : [B] C



Step 3 :

Preorder : [D] E F G H I

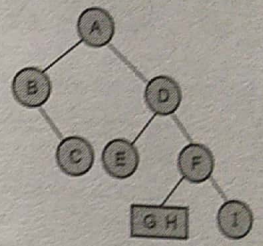
Inorder : E [D] G H F I



Step 4 :

Preorder: [F] G H I

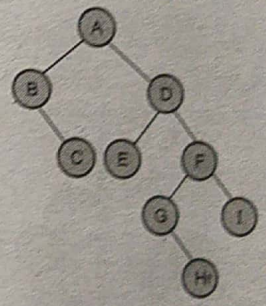
Inorder : G H [F] I



Step 5 :

Preorder: [G] H

Inorder : [G] H



Binary tree

3.2 : AVL Trees

Q.25 Define AVL Tree. Give example. [JNTU : Part A, Dec 16, Marks 2, Dec.-18, Marks 5]

Ans. : An empty tree is height balanced if T is a non empty binary tree with T_L and T_R as its left and right subtrees. The T is height balanced if and only if

i) T_L and T_R are height balanced.

ii) $h_L - h_R <= 1$ where h_L and h_R are heights of T_L and T_R.

The idea of balancing a tree is obtained by calculating the balance factor of a tree.

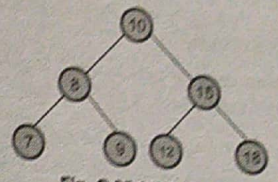


Fig. Q.25.1 AVL tree

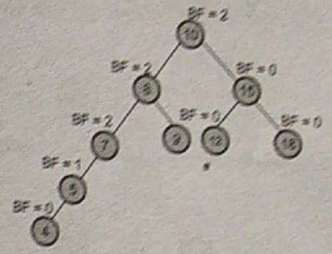


Fig. Q.25.2 Not an AVL tree

Q.26 Define balance factor. [JNTU : Part A, Marks 2]

Ans. : The balance factor of a node in binary tree is defined to be hR - hL where hL and hR are heights of left and right subtrees of T. For any node in AVL tree the balance factor should be 1, 0 or - 1.

Q.27 When AVL tree property is violated and how to solve it ? [JNTU : Part A, Marks 2]

Ans. : On inserting a new node in AVL tree if the balance factor is other than 1, 0 or - 1 then the AVL tree property is violated.

Check the affected node adjust subtree by applying rotations.

Q.28 What are four different rotations used in AVL tree ? [JNTU : Part A, Marks 2]

Ans. :

1. Insertion of left child in left subtree(LL Rotation).
2. Insertion of right child in left subtree(LR Rotation).
3. Insertion of right child in left subtree(RL Rotations).
4. Insertion of right child in right subtree(RR Rotations).

Scanned by CamScanner

Q.29 Discuss about AVL tree rotations [JNTU : Part B, Dec.-16, Marks 5]

Ans. :

1. LL rotation

When node '1' gets inserted as a left child of node 'C' then AVL property gets destroyed i.e. node A has balance factor + 2.

The LL rotation has to be applied to rebalance the nodes.

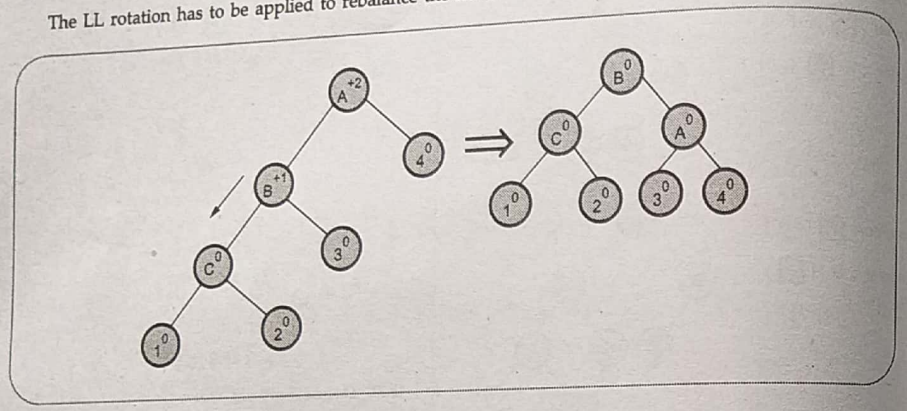


Fig. Q.29.1 LL rotation

2. RR rotation

When node '4' gets attached as right child of node 'C' then node 'A' gets unbalanced. The rotation which needs to be applied is RR rotation as shown in Fig. Q.29.2.

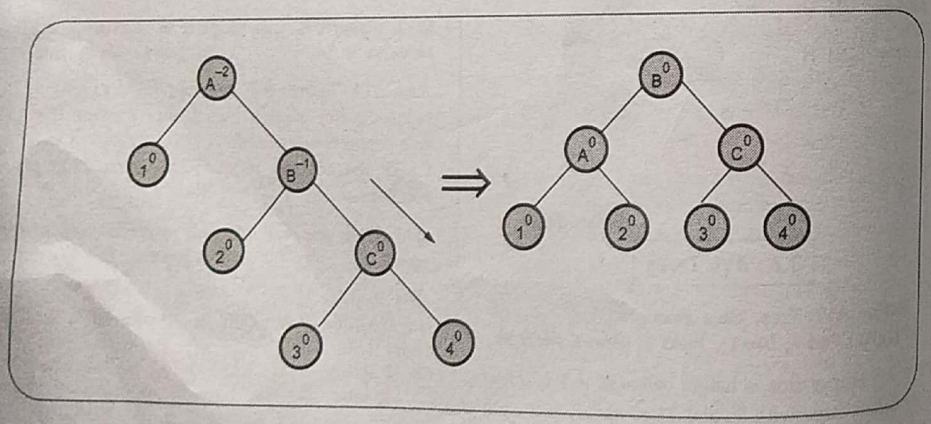


Fig. Q.29.2 RR rotation

3. LR rotation

When node '3' is attached as a right child of node 'C' then unbalancing occurs because of LR. Hence LR rotation needs to be applied.

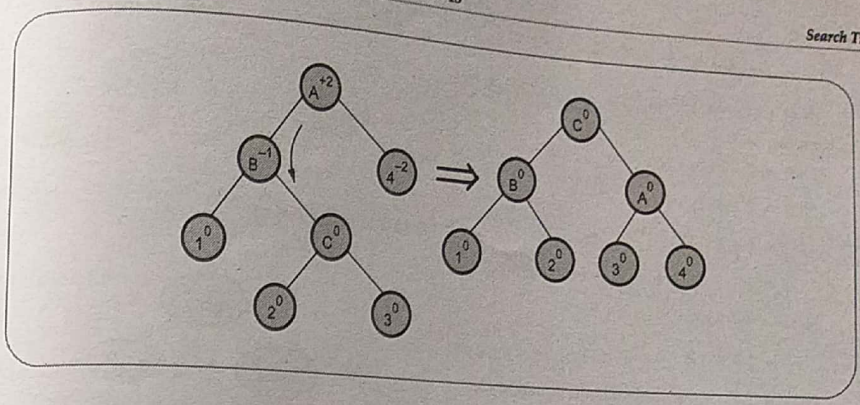


Fig. Q.29.3 LR rotation

4. RL rotation

When node '2' is attached as a left child of node 'C' then node 'A' gets unbalanced as its balance factor becomes – 2. Then RL rotation needs to be applied to rebalance the AVL tree.

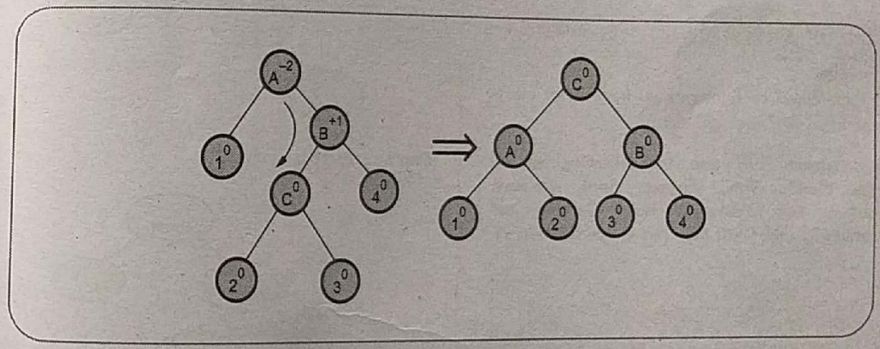


Fig. Q.29.4 RL rotation

Q.30 What is the minimum number of nodes in an AVL tree of height 5.

Ans. : If h is the height of an AVL tree then minimum number of nodes can be obtained for such AVL tree using the following formula

$$N(h) = 1 + N(h-1) + N(h-2) \quad \text{... (1)}$$

If there is only 1 node then height h is 0. Similarly for 2 nodes the maximum height can be 1. We can formulate and say.

$$N(0) = 1, \quad N(1) = 2$$

Hence we can obtain minimum number of nodes for height 5 using equation (1) as

$$N(5) = 1 + N(4) + N(3) \quad \text{... (2)}$$

Scanned by CamScanner

Let us compute, N(4)

$N(4) = 1 + N(3) + N(2)$ ... (3)

Let us compute, N(3) ... (4)

$N(3) = 1 + N(2) + N(1)$

$N(2) = 1 + N(1) + N(0)$

$= 1 + 2 + 1$

$\boxed{N(2) = 4}$ → put this value in equation (4)

$N(3) = 1 + N(2) + N(1)$

$= 1 + 4 + 2$

$\boxed{N(3) = 7}$ → put this value in equation (3)

$N(4) = 1 + N(3) + N(2)$

$= 1 + 7 + 4$

$\boxed{N(4) = 12}$ → put this value in equation (2)

$N(5) = 1 + N(4) + N(3)$

$= 1 + 12 + 7$

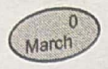
$\boxed{N(5) = 20}$

Thus there could be minimum 20 node those can be arrange in an AVL tree of height 5.

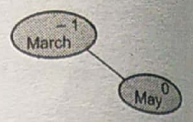
Q.31 Construct AVL tree for following set of months March, May, August, April, January, December, July, February, June, October and September. ☞ [JNTU (A) : Part B, Nov-12, Marks 7]

Ans. :

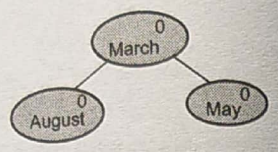
Step 1 : Insert March



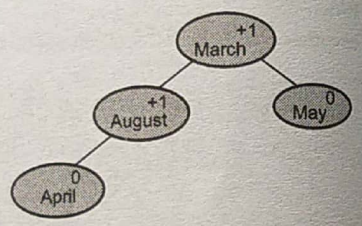
Step 2 : Insert May



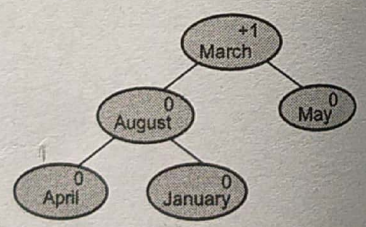
Step 3 : Insert August



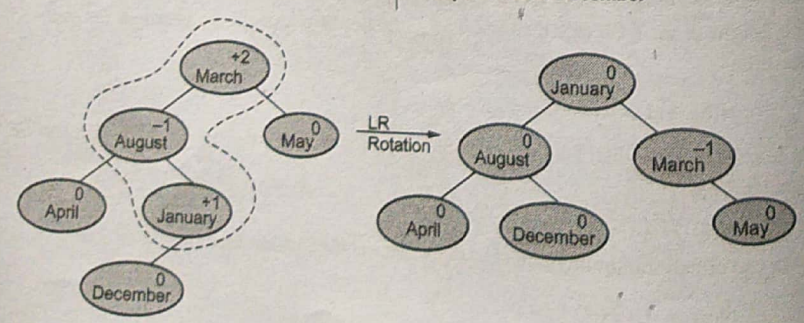
Step 4 : Insert April



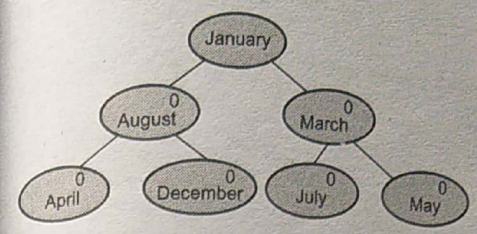
Step 5 : Insert January



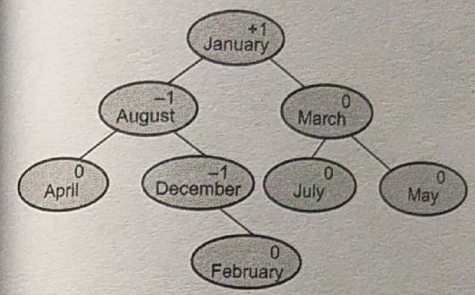
Step 6 : Insert December



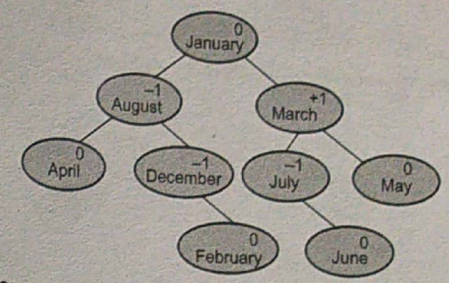
Step 7 : Insert July



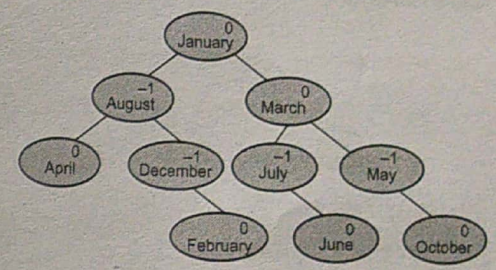
Step 8 : Insert February



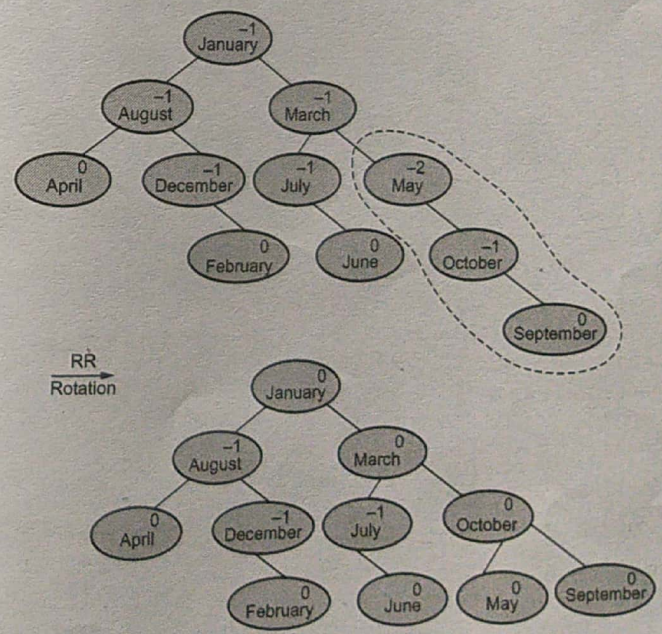
Step 9 : Insert June



Step 10 : Insert October



Step 11 : Insert September



This is Final AVL Tree

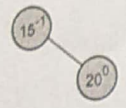
Scanned by CamScanner

Q.32 Build an AVL tree with the following values : {15, 20, 24, 10, 13, 7, 30, 36, 25, 42, 29} [JNTU [A] : Part B, Dec.-16, Marks 5]

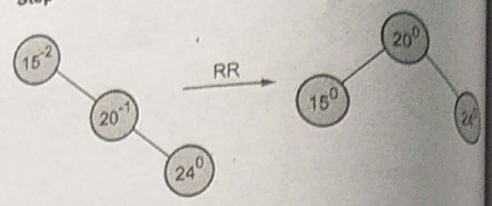
Ans. :

Step 1 : Insert 15

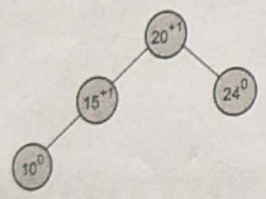
Step 2 : Insert 20



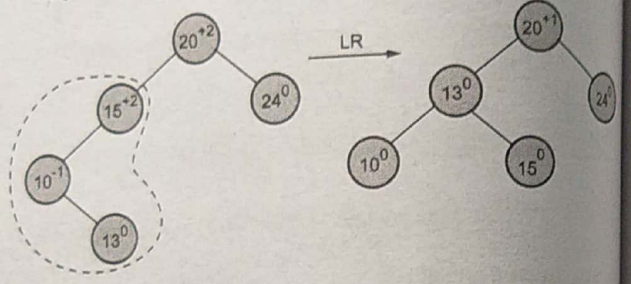
Step 3 : Insert 24



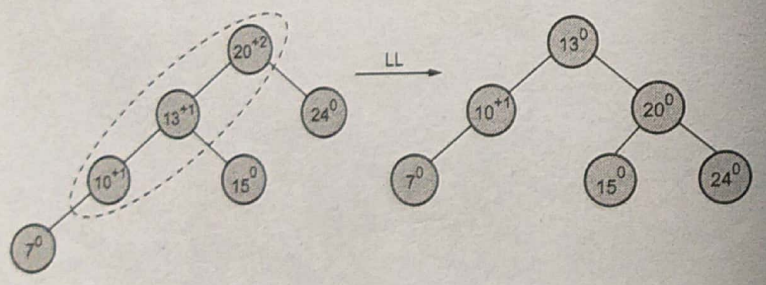
Step 4 : Insert 10



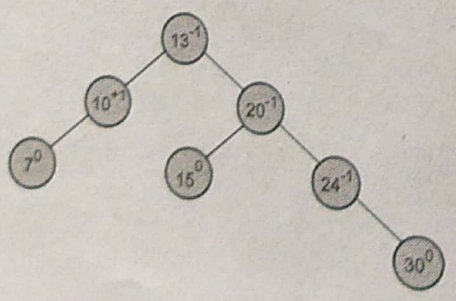
Step 5 : Insert 13



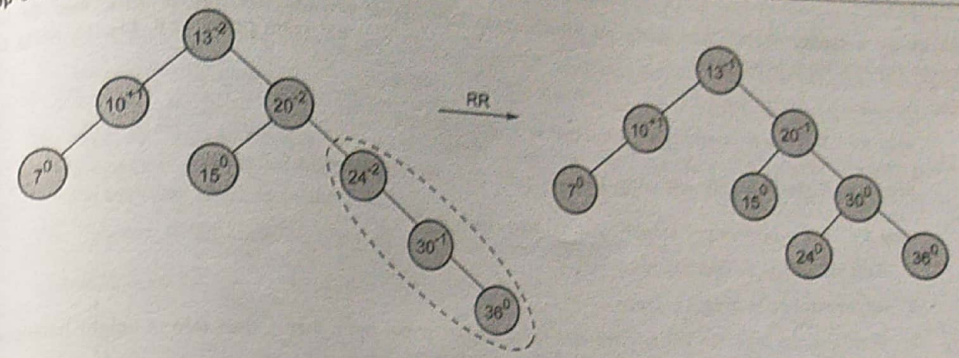
Step 6 : Insert 7



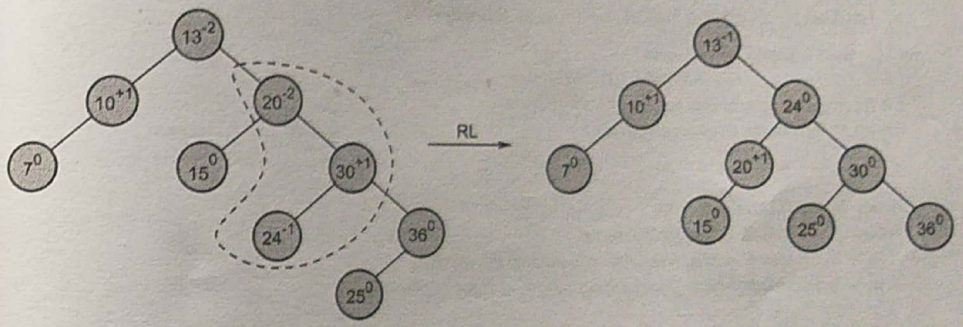
Step 7 : Insert 30



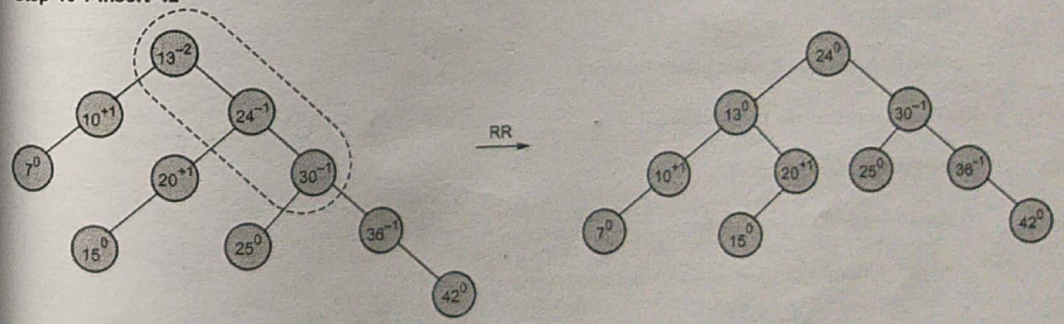
Step 8 : Insert 36



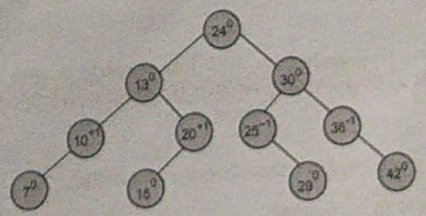
Step 9 : Insert 25



Step 10 : Insert 42



Step 11 : Insert 29



Scanned by CamScanner

Q.33 Given a binary search tree write an algorithm to tree check whether the tree is AVL tree or not. Discuss the run time complexity [JNTU [H] : Part B, Dec.-14, Marks 10]

Ans. :

Step 1 : If the tree is empty then the tree is height balanced.

Step 2 : If there is only one node, i.e. root node then the tree is height balanced.

Step 3 : For a non empty height balanced tree check if

1. Left subtree is height balanced
2. Right subtree is height balance
3. If difference between left subtree and right subtree is not more than 1 then tree is height balanced otherwise it is not height balanced.

The function to check if it is AVL or not is as follows -

```
int CheckAVL(struct node *root)
{
  int lh; /* for height of left subtree */
  int rh; /* for height of right subtree */


  if(root == NULL)//empty tree
   return 1; //hence it is height balanced
     lh = height(root->left); //obtaining height of left subtree
     rh = height(root->right);//obtaining height of right subtree

  if( abs(lh-rh) <= 1 && CheckAVL(root->left) && CheckAVL (root->right))
    return 1;

  return 0;//otherwise it is not height balanced
}
```

The height function can be as follows -

```
int height(struct node* node)
{
   if(node == NULL)//empty tree
     return 0;
  return 1 + max(height(node->left), height(node->right));
}
```

Analysis

The time complexity is $O(n^2)$

Q.34 Write an algorithm for deletion of node from AVL tree. Explain it with suitable example [JNTU : Part B, Marks 5]

Ans. : Algorithm for deletion

The deletion algorithm is more complex than insertion algorithm.

1. Search the node which is to be deleted.

2. a) If the node to be deleted is a leaf node then simply make it NULL to remove.

   b) If the node to be deleted is not a leaf node i.e. node may have one or two children, then the node must be swapped with its inorder successor. Once the node is swapped, we can remove this node.

3. Now we have to traverse back up the path towards root, checking the balance factor of every node along the path. If we encounter unbalancing in some subtree then balance that subtree using appropriate single or double rotations.

The deletion algorithm takes **O(log n)** time to delete any node.

Consider an AVL tree.

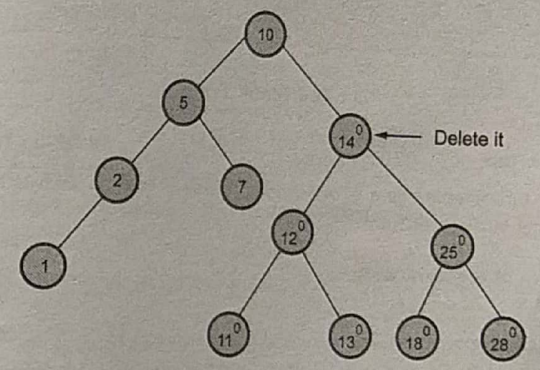


Fig. Q.34.1

The tree becomes

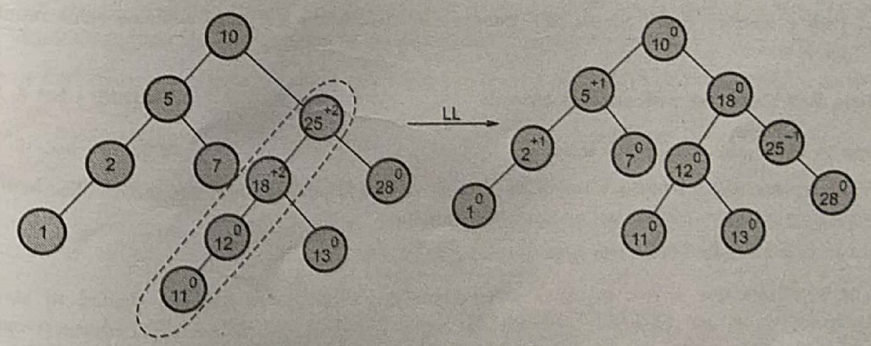


Fig. Q.34.2

Scanned by CamScanner

Q.35 Compare AVL tree with binary search tree [JNTU : Part A, Marks 4]

Ans. :

Sr. No.	AVL tree	Binary search tree
1.	An AVL tree is an height balanced search tree.	Binary search tree is not a balanced tree.
2.	There is a requirement of computation of balanced factor for each node in an AVL tree.	There is no concept of balance factor.
3.	Searching of any desired node is faster due to balancing of height.	Searching is not efficient when there are large number of nodes in the tree.
4.	Complex to implement.	Simple to implement.

3.3 : Red Black Trees

Q.36 Write about red-black tree. [JNTU : Part A, Dec.-16, 17, Marks 2]

Ans. : Red-Black tree is a binary search tree in which every node is colored with either red or black. It is a type of self balancing binary search tree.

Q.37 Explain the properties of red-black trees. [JNTU : Part A, Marks 2]

Ans. :

1. **Root property :** The root is black.
2. **External property :** Every leaf (Leaf is a NULL child of a node) is black in red-black tree.
3. **Internal property :** The children of a red node are black. Hence possible parent of red node is a black node.
4. **Depth property :** All the leaves have the same black depth.
5. **Path property :** Every simple path from root to descendant leaf node contains same number of black nodes.

Q.38 Explain Red black tree with suitable example. [JNTU : Part A, Marks 4]

OR Discuss in detail about red-black trees. [JNTU : Dec.-18, Marks 5]

Ans. : While representing a Red-Black tree color of every node and pointer colors are shown. The leaf nodes are simply NULL nodes and are always black in color.

As an example a Red-Black tree is as shown in Fig. Q.38.1.

The Red-Black tree shown in above given figure has black nodes that are shaded in black and unshaded nodes are red nodes. Similarly the leaves are black and all are NULL pointers only. The pointers to black node are black pointers which are shown by thick lines remaining are red pointers. But in practice, we explicitly mention the color of nodes. The above given Red-Black tree follows all the properties of Red-Black tree -

1. It is a binary search tree.
2. The root node is black.

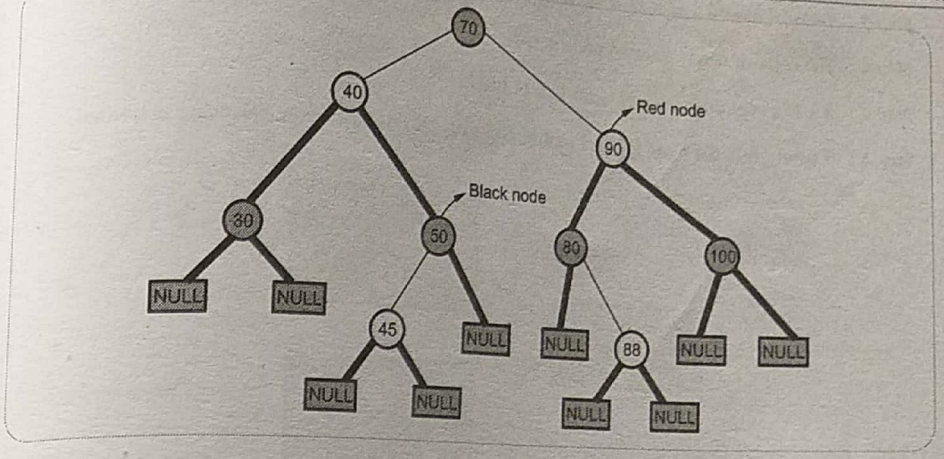


Fig. Q.38.1 Red-Black tree

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.39 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 -

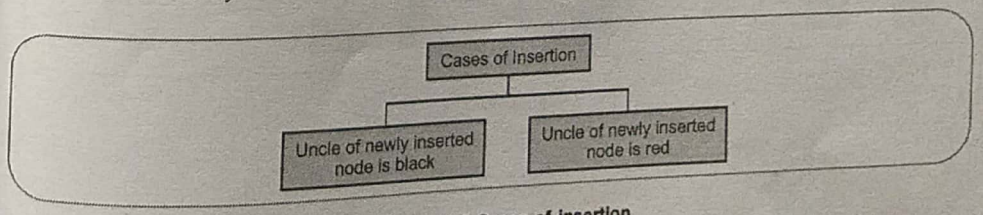


Fig. Q.41.1 Cases of insertion

Let us understand insertion operation in Red black tree.

Scanned by CamScanner

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.

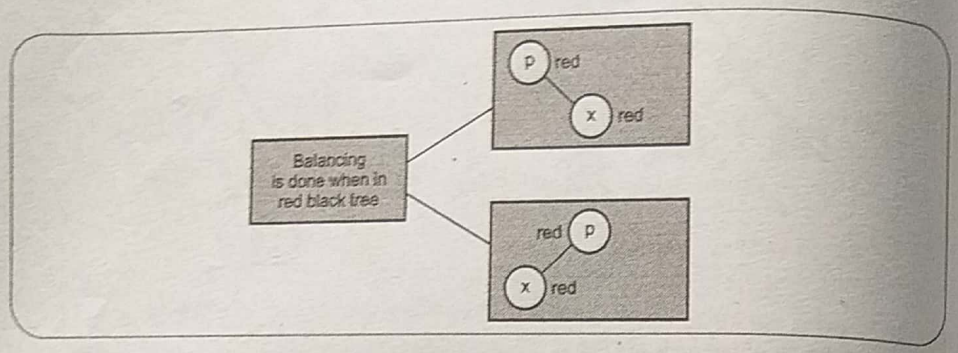


Fig. Q.39.2

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.

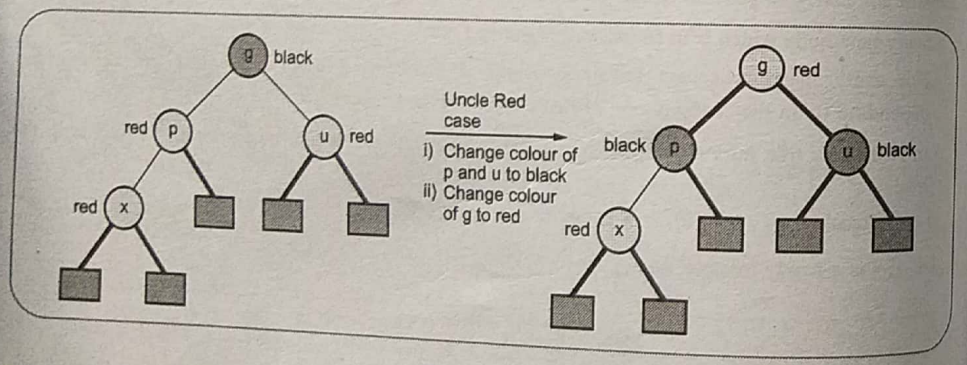


Fig. Q.39.3

i) Left left case :

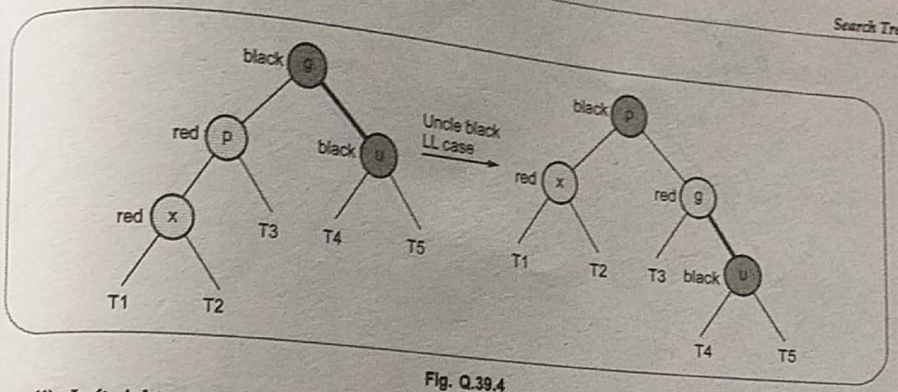


Fig. Q.39.4

ii) Left right case :

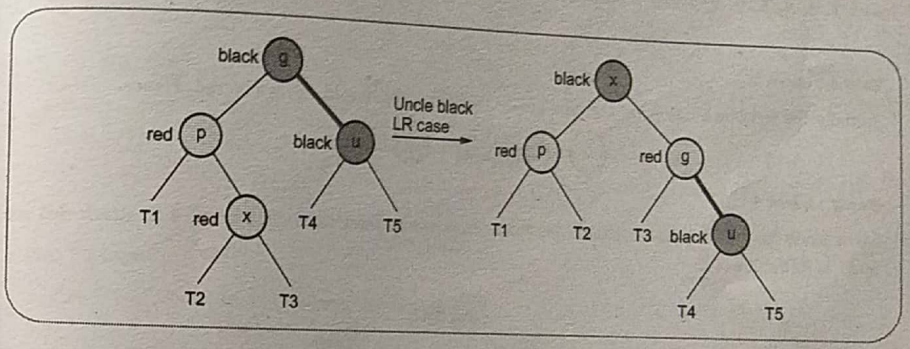


Fig. Q.39.5

iii) Right right case :

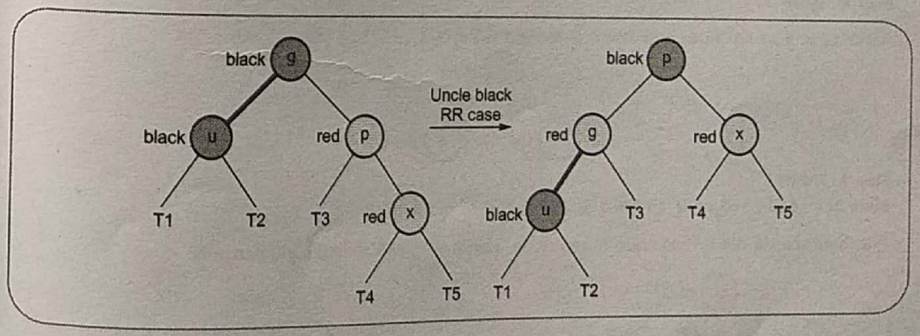


Fig. Q.39.6

Scanned by CamScanner

iv) Right left case :

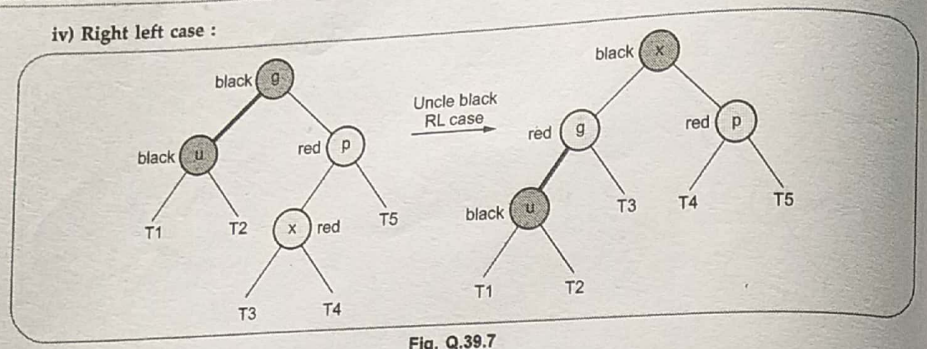


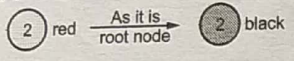
Fig. Q.39.7

Q.40 Insert 2, 1, 4, 5, 9, 3, 6 and 7 for red black tree. [JNTU : Part B, Marks 5]

Ans. :

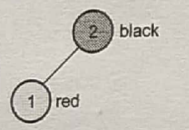
Step 1 : Insert 2

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



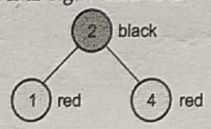
Step 2 : Insert 1

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



Step 3 : Insert 4

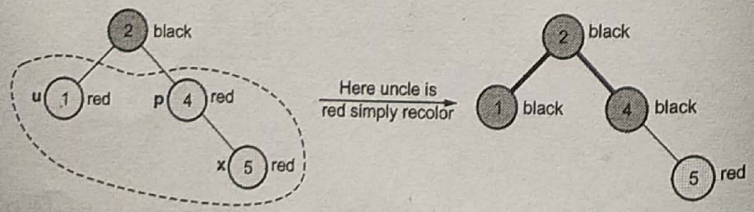
Make node 4 as red node and attach it as right child of 2.



Step 4 : Insert 5

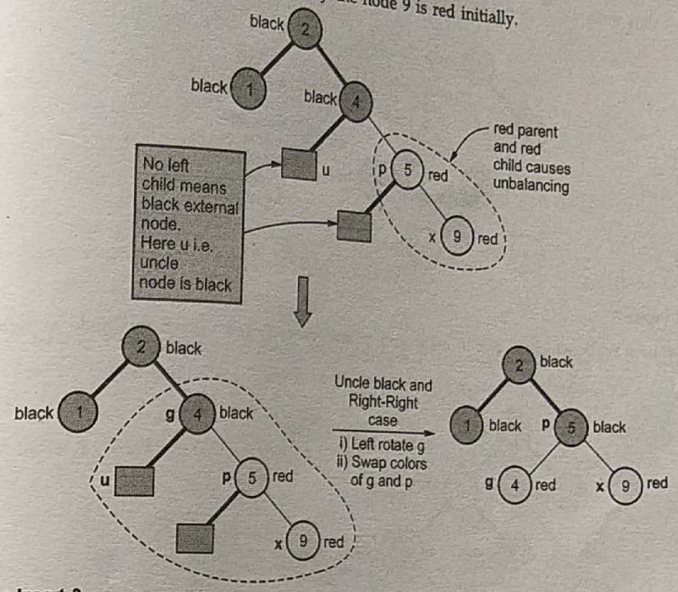
Here node 5 is a red node attached as right child of red node 4.

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



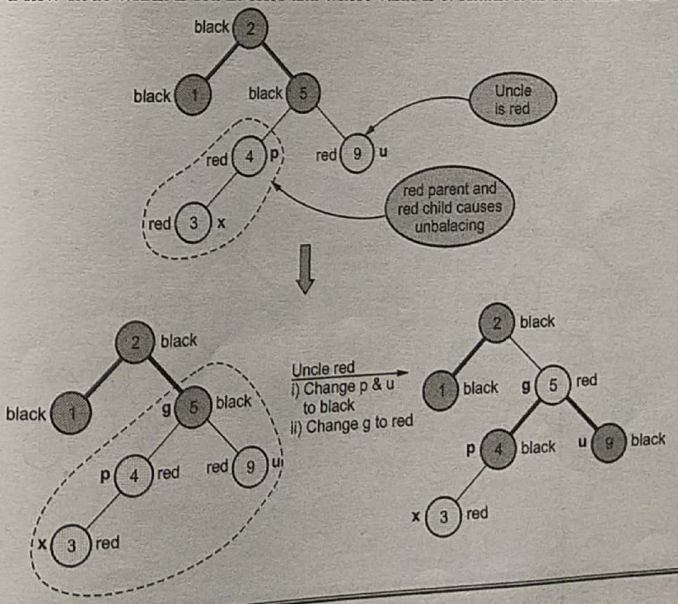
Step 5 : Insert node 9

Insert node 9 as a right child of 5. Naturally the node 9 is red initially.



Step 6 : Insert 3

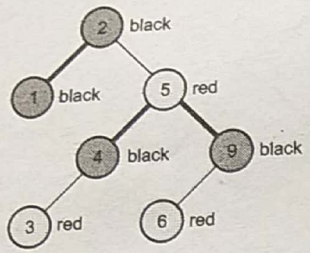
Create a new node which is red in color and whose value is 3. Attach it as left child of 4.



Scanned by CamScanner

Step 7 : Insert 6

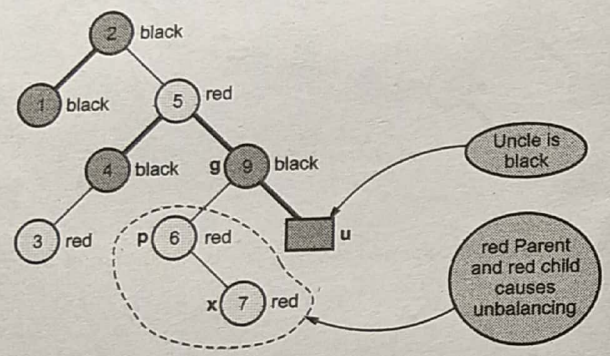
Create a new node which is red in color and whose value is 6. Attach this node as a left child of 9.

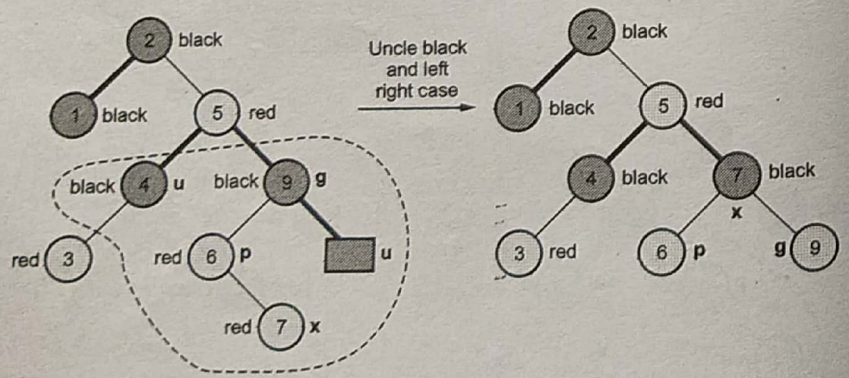


As here parent node i.e. 9 is black and child node i.e. 6 is red, there is no need for adjustment.

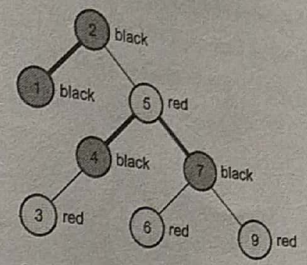
Step 8 : Insert 7

Create a new node which is red in color and whose value is red. Attach this node as right child of node 6.





Thus finally we get, following red black tree.

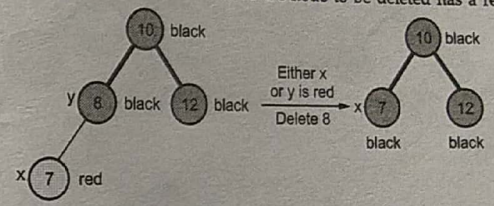


Q.41 Explain deletion operation in red black tree.

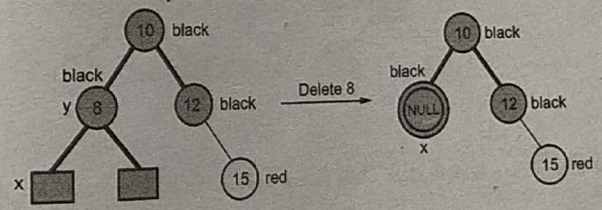
Ans. : [JNTU : Part B, Marks 10]

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.

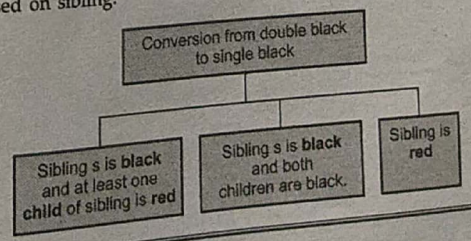


Step 3 : Case 2 : If Both x and y are black.



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.



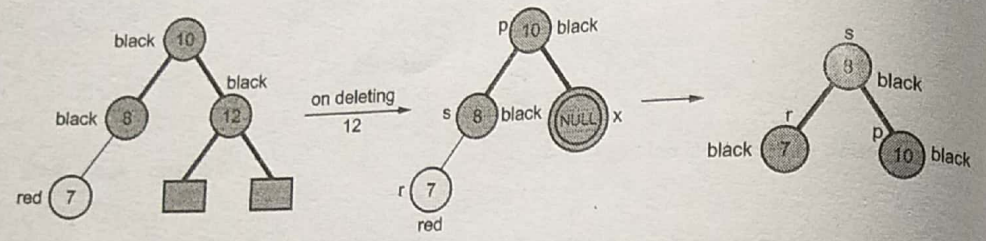
Scanned by CamScanner

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.

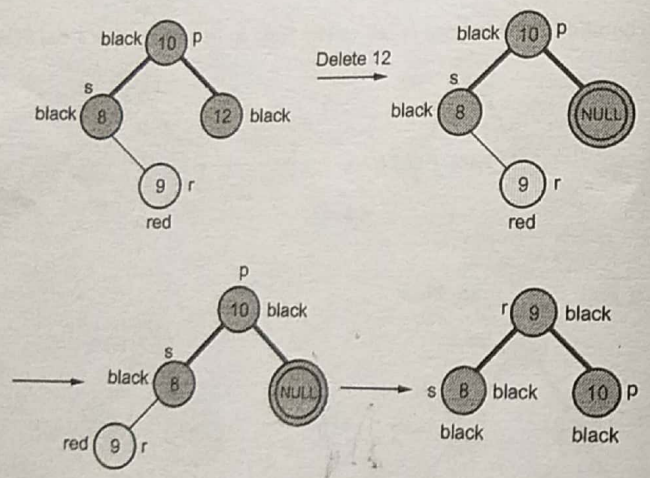
i) Left left case - The s is left child of parent and r is left child.

For example - Delete 12



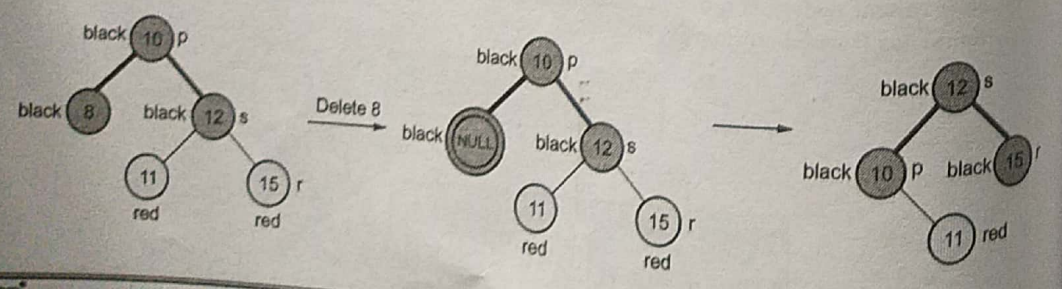
ii) Left right case - The s is a left child of its parent and r is a right child.

For example :

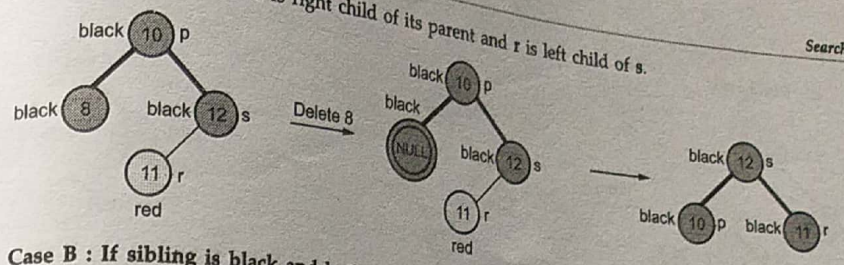


iii) Right right case - The s is right child of its parent node and r is right child of s. Both the children of s are red.

For example :

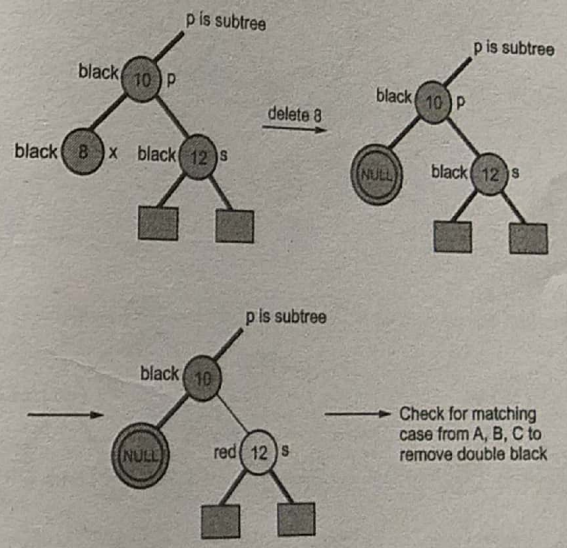


iv) Right left case - The s is right child of its parent and r is left child of s.



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.

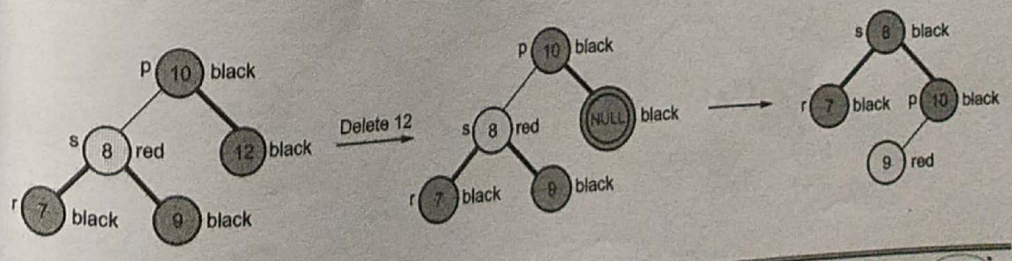
For example :



Case C : If sibling is red.

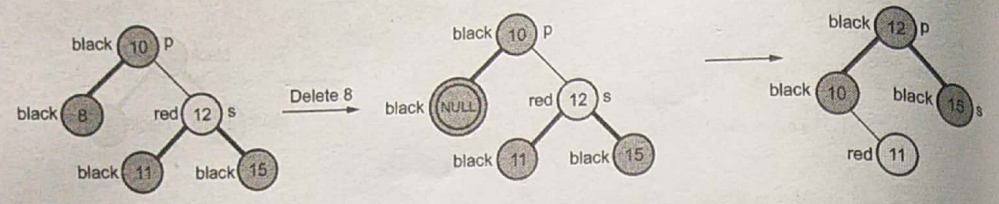
This case has two subcases

i) Left case

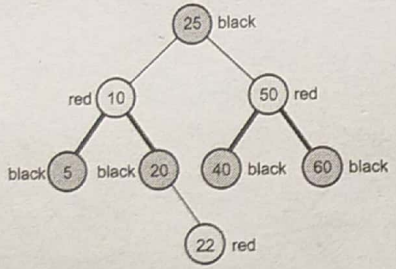


Scanned by CamScanner

ii) Right case

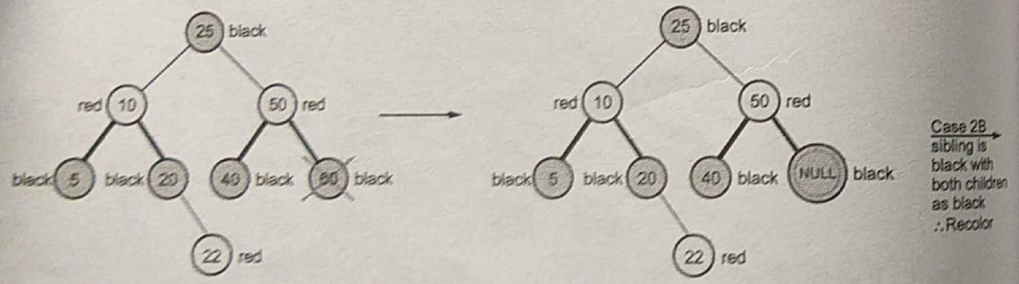


Q.42 Following is a Red-Black tree. Delete 60, then 50 and finally 40 from the above tree. Show clearly the balancing done after each deletion.



Ans. :

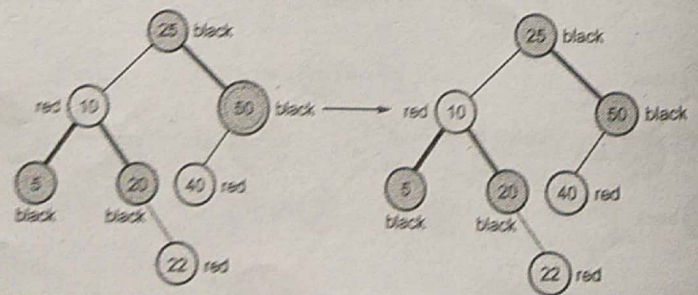
Step 1 : Delete 60

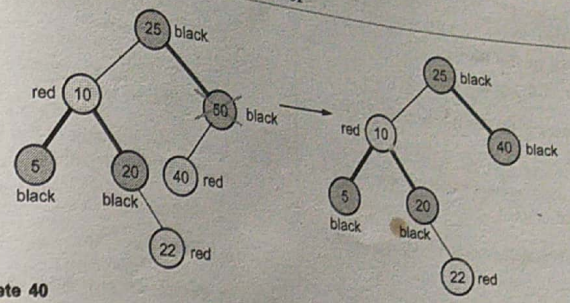


Case 2B sibling is black with both children as black ∴ Recolor

Step 2 : Delete 50

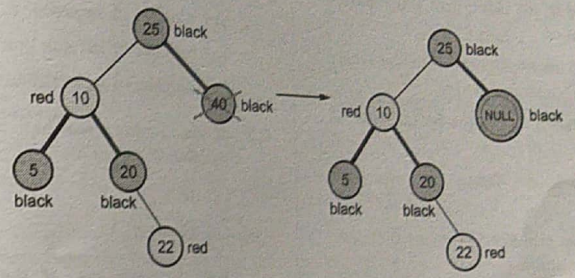
This is case 1 in which the node to be deleted has one red child.



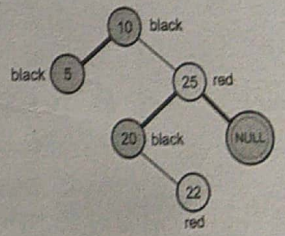


Step 3 : Delete 40

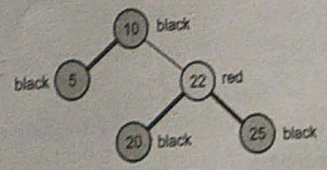
Here case C i.e. sibling of double black node is red- is applicable.



The left case is applicable and we rotate in such a way that 10 becomes the root node.



Now apply case A (ii) as sibling is black with red child. Hence apply LR rotation to get.



Thus we get balanced Red. Black tree after deletions.

Scanned by CamScanner

UNIT - IV

4 Graphs and Sorting

PART - I : GRAPHS

4.1 : Introduction to Graphs

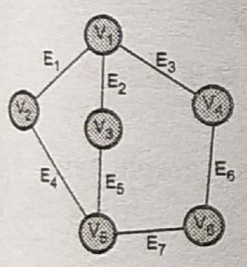
Q.1 Define Graph

☞ [JNTU : Part A, May 15, Marks 2]

Ans. : A graph is a collection of two sets V and E where V is a finite non-empty set of vertices and E is a finite non-empty set of edges.

- Vertices are nothing but the nodes in the graph.
- Two adjacent vertices are joined by **edges**.
- Any graph is denoted as G = {V, E}.

For example :



$G = \{\{V_1, V_2, V_3, V_4, V_5, V_6\}, \{E_1, E_2, E_3, E_4, E_5, E_6, E_7\}\}$

Fig. Q.1.1 Graph G

Q.2 Distinguish between tree and Graph with example

☞ [JNTU : Part A, Nov.-15, Marks 15]

Ans. :

Sr. No.	Graph	Tree
1.	Graph is a non-linear data structure.	Tree is a non-linear data structure.
2.	It is a collection of vertices/nodes and edges.	It is a collection of nodes and edges.
3.	Each node can have any number of edges.	General trees consist of the nodes having any number of child nodes. But in case of binary trees every node can have at the most two child nodes.
4.	There is no unique node called root in graph.	There is a unique node called **root** in trees.
5.	A cycle can be formed.	There will not be any cycle.
6.	Applications : For finding shortest path in networking graph is used.	Applications : For game trees, decision trees, the tree is used.

Q.3 What is directed and undirected graphs ?

☞ [JNTU : Part A, Dec.-17, Marks 3]

OR Differentiate between directed and undirected graph

☞ [JNTU : Dec.-17, Marks 3]

OR What is directed graph ?

☞ [JNTU : May-18, Marks 2]

OR What is undirected graph ?

☞ [JNTU : Dec.-18, Marks 2]

Ans. : In the **directed graph** the directions are shown on the edges. As shown in the Fig. Q.3.1, the edges between the vertices are ordered. In this type of graph, the edge E_1 is in between the vertices V_1 and V_2. The V_1 is called head and the V_2 is called the tail. Similarly for V_1 head the tail is V_3 and so on. We can say E_1 is the set of (V_1, V_2) and not of (V_2, V_1). Similarly, in an undirected graph, the edges are not ordered. See Fig. Q.3.2 for clear understanding of undirected graph. In this type of graph the edge E_1 is set of (V_1, V_2) or (V_2, V_1).

Scanned by CamScanner

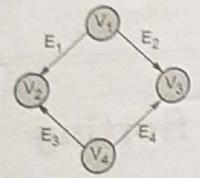


Fig. Q.3.1 Directed graph

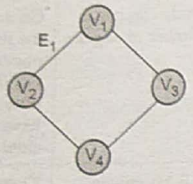


Fig. Q.3.2 Undirected graph

Q.4 Explain the terms - 1) Connected Graph 2) Weighted Graph 3) Path and Cycle

☞ [JNTU : Part A, Marks 3]

Ans. : i) Connected graph :

An undirected graph is said to be connected if for every pair of distinct vertices V_i and V_j in V(G) there is an edge V_i to V_j in G.

Note that there is path from any two vertices.

For example :

- $V_1 - V_2$ • $V_2 - V_1$ • $V_3 - V_1$ • $V_4 - V_3 - V_1$
- $V_1 - V_3$ • $V_2 - V_1 - V_3$ • $V_3 - V_1 - V_2$ • $V_4 - V_2$
- $V_1 - V_4$ • $V_2 - V_4$ • $V_3 - V_4$ • $V_4 - V_3$

ii) Weighted graph

A weighted graph is a graph which consists of weights along its edges.

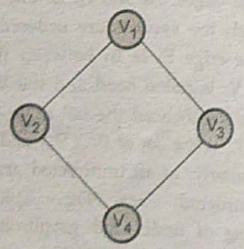


Fig. Q.4.1 Connected graph

For example :

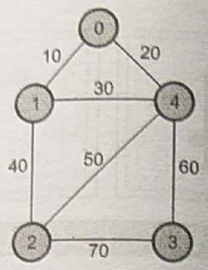


Fig. Q.4.2 Weighted graph

iii) Path :

A path is denoted using sequence of vertices and there exists an edge from one vertex to the next vertex.

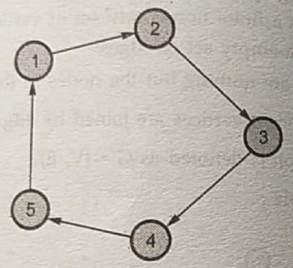


Fig. Q.4.3 Path of G is 1-2-3-4-5

iv) Cycle :

A closed walk through the graph with repeated vertices, mostly having the same starting and ending vertex is called a cycle.

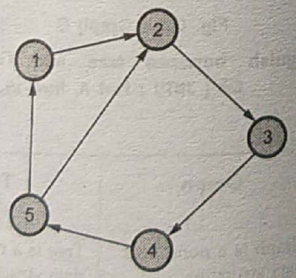


Fig. Q.4.3 Cycle 2-3-4-5-2 or 1-2-3-4-5-1

Q.5 Explain in detail about graph ADT.

☞ [JNTU : May-18, Marks 5]

Ans. : The abstract data type for graph is as given below.

AbstractDataType Graph

{

Instances

Graph is a collection of vertices and edges.

Operations

1) tot_vertices ( ) : This function returns total number of vertices.
2) tot_edges ( ) : This function returns total number of edges.
3) Is_edge (V_1, V_2) : This function returns true if there exists an edge between V_1 and V_2.
4) In_degree ( ) : This function returns total number of edges that are incident to the vertex.
5) Out_degree ( ) : This function returns out degree of the vertex.
6) Display ( ) : This function displays the graph in DFS or in BFS manner.

}

Q.6 What is graph? Explain types with examples.

(Refer Q.1 and Q.3) ☞ [JNTU : May-18, Marks 5]

4.1.1 : Implementation Methods

Q.7 What are the storage representation of the graph ? ☞ [JNTU : Part A, Marks 2]

Ans. : The graph can be represented by using arrays or linked list.

1. The adjacency matrix is a storage representation of the graph by which the graph is stored in two dimensional arrays.
2. The adjacency list is a storage representation of a graph by which the graph is stored in linked list.

Q.8 What is graph ? Explain various representations of graphs.

☞ [JNTU : Part B, Dec.-16,17, Marks 5]

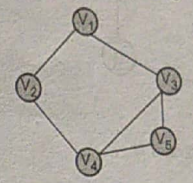
Ans. : Graph - Refer Q. 1

Representation - Normally a graph can be represented by two representations and those are

1) Adjacency matrix : In this representation, matrix or 2 dimensional array is used to represent the graph. Consider a graph G of n vertices and the matrix M. If there is an edge present between vertices V_i and V_j then M[i][j] = 1 else M[i][j] = 0. Note that for an undirected graph if M[i][j] = 1 then for M[j][i] is also 1. Here are some graphs shown by adjacency matrix.

Adjacency matrix for weighted graph

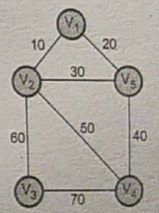
- In the weighted graph, weights or distances are given along every edge. Hence in an adjacency



$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

Fig. Q.8.1 An adjacency matrix representation

matrix representation any edge which is present between vertices V_i and V_j is denoted by its weight. Hence M[i][j] = Weight of edge.



Weighted graph

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 10 & 0 & 0 & 20 \\ 2 & 10 & 0 & 60 & 50 & 30 \\ 3 & 0 & 60 & 0 & 70 & 0 \\ 4 & 0 & 50 & 70 & 0 & 40 \\ 5 & 20 & 30 & 0 & 40 & 0 \end{array}$$

Adjacency matrix

Fig. Q.8.2 Adjacency matrix representation for weighted graph

- If there is no edge between V_i and V_j then, M[i][j] = 0.

2) Adjacency list : In this representation, a **linked list** is used to represent a graph.

There are two methods of representing graph using adjacency list.

Method 1 :

As we know the graph is a set of vertices and edges, we will maintain the two structures, for vertices and edges respectively.

Scanned by CamScanner

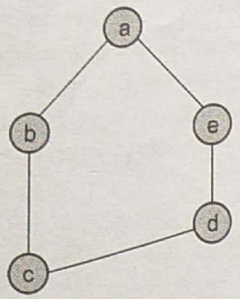


Fig. Q.8.3 Graph G

For example :

Now in Fig. Q.8.3, graph has the nodes as a, b, c, d, e. So we will maintain the linked list of these head nodes as well as the adjacent nodes.

```
typedef struct head
{
   char data;
   struct head *down;
   struct head *next;
};
typedef struct node
{
 int ver;
struct node *link;
};
```

Explanation : This is purely the adjacency list graph. The down pointer helps us to go to each node in the graph where as the next node is for going to adjacent node of each of the head node.

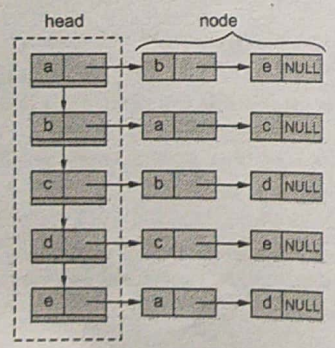


Fig. Q.8.4 Adjacency list (Method 1)

Method 2 : In this method of representing the adjacency list, we take mixed data structure. That means instead of taking the head list as a linked list we will take an array of head nodes. So only one 'C' structure will be there representing the adjacent nodes. See the Fig. Q.8.5 representing the adjacency list for the above graph.

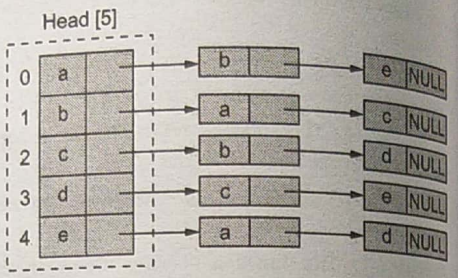


Fig. Q.8.5 Adjacency list (Method 2)

Q.9 Specify the situation in which the adjacency list representation of graph is preferred than adjacency matrix. ☞ [JNTU : Part A, Marks 2]

Ans. : The situation in which the number of nodes in the graph is not fixed and we need to create the graph dynamically, then linked representation is preferred than graph representation.

Q.10 Explain the graph creation algorithm with C code for adjacency matrix implementation. ☞ [JNTU : Part B, Marks 5]

Ans. : Creation of graph using adjacency matrix is quite simple task. The adjacency matrix is nothing but a two dimensional array. The algorithm for creation of graph using adjacency matix will be as follows :

1) Declare an array of M[size] [size] which will store the graph.
2) Enter how many nodes you want in a graph.
3) Enter the edges of the graph by two vertices each, say , indicates some edge.
4) If the graph is directed set M[i][j] = 1. If graph is undirected set M[i][j] = 1 and M[j][i] = 1 as well.
5) When all the edges for the desired graph is entered print the graph M[i][j].


```
void create()
{
    int ch,flag;   //flag for directed or undirected graph
    flag = TRUE;
    n=0;
    printf("\n\t\t This is a Progrm To Create a Graph");
    printf("\n\t\t The Display Is In Breadth First Manner");
    printf("\n Press d/D for directed and u/U for undirected graph\n");
    ch = toupper(getche());
    if ( ch == 'D')
      flag = FALSE;

    do
    {
        printf("\nEnter the Edge of a graph ");
        printf("(And for termination type99)\n");
        scanf("%d%d,&v1,&v2);
        if ( v1 == 99)
          break;
        if ( v1 >= size || v2 >= size)
            printf("Invalid Vertex Value\n" );
        else
          g[v1][v2] = TRUE;
          if (flag)
            g[v2][v1] = TRUE;
    n++;//for keeping track of total number of nodes
    }while(1);
}
```

Q.11 Explain the graph creation algorithm with C code for adjacency list implementation.

Ans. :

1. Declare node structure for creating adjacency list.
2. Initialize an array of nodes. This array will act as head nodes. Say '*head [10]'. The index of head[ ] will be the starting vertex.

Scanned by CamScanner

3. The create function will create the adjacency list for given graph G as follows -

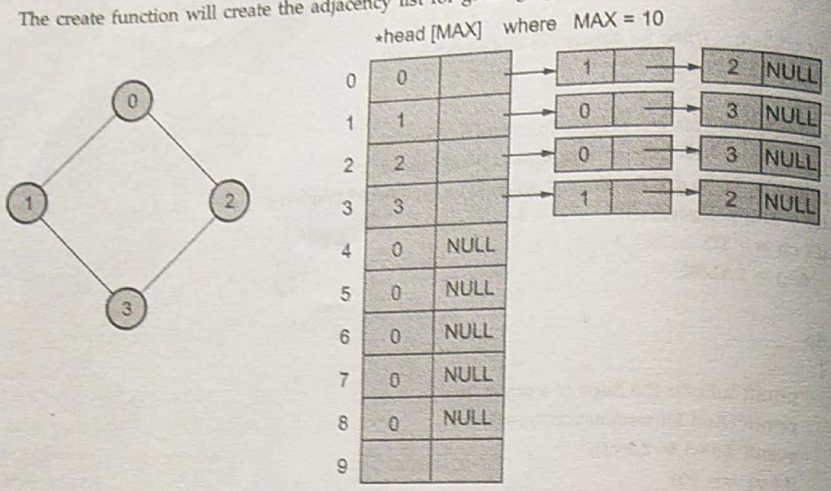


Fig. Q.11.1 Graph and its adjacency list representation

The pseudo code for create function is as follows :

```
void create()
{
    int V1, V2;
    char ans='y';
    node *New,*first;
    printf("\n\nEnter the vertices no. beginning with 0");
    do
    {
      printf("\nEnter the Edge of a graph \n");
       scanf("%d%d",&V1,&V2);
       if ( V1 >= MAX || V2 >= MAX)
          printf("Invalid Vertex Value\n");
       else
        {
          // creating link from V1 to V2
          New = new node;
          if ( New == NULL )
           printf("Insufficient Memory\n");
          New-> vertex = V2;
          New-> next = NULL;
          first = head[V1];
          if ( first == NULL )
           head[V1] = New;
          else
           {
             while ( first-> next != NULL )
```



```
                first = first-> next;
              first-> next = New;
           }
          // creating link from V2 to V1
           New = new node;
           if ( New == NULL )
            printf("Insufficient Memory\n");
           New-> vertex = V1;
           New-> next = NULL;
           first = head[V2];
           if ( first == NULL )
            head[V2] = New;
           else
            {
              while ( first-> next != NULL )
                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.

Scanned by CamScanner

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

C Code

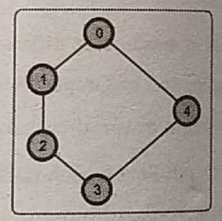
```
void bfs(int v1)
{
    int v2;
    visit[v1] = TRUE;
    front = rear = -1;
    Q[++rear] = v1;
    while ( front != rear )
    {
        v1 = Q[++front];
        printf("\n"%d",v1);
        for ( v2 = 0; v2 < n; v2++)
        {
            if ( g[v1][v2] == TRUE && visit[v2] == FALSE )
        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

For example



Q.15 Implement DFS algorithm. ☞ [JNTU : Part B, Dec.-16, Marks 5]

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);
}
```


Q.16 For the graph given below, find DFS stepwise. ☞ [JNTU : Part B, Marks 5]

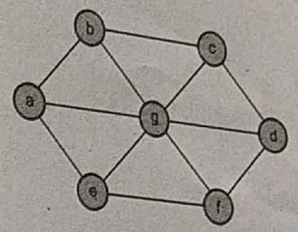
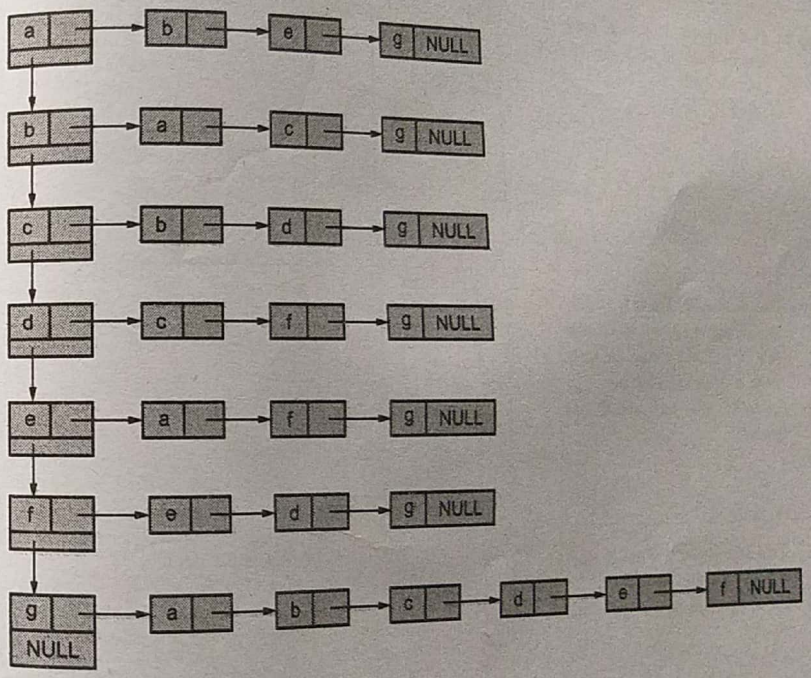


Fig. Q.16.1

Ans. : We will create adjacency list for given graph.



Depth first search (DFS)

Scanned by CamScanner

Action	Stack	Visited array	Output
Step 1 : Start with vertex a, mark it visited by writing 1 to index a of visited array. The print a as an output.		a 1, b, c, d, e, f, g	a
Step 2 : Find adjacent of b, and mark it visited. Push it onto the stack. Later b will be popped and printed.	b	a 1, b 1, c, d, e, f, g	a, b
Step 3 : Find adjacent b. The a is already visited, hence ignore. Next node will be c. Insert c onto the stack mark it as visited. Push it. Later c will be popped and printed.	c	a 1, b 1, c 1, d, e, f	a, b, c
Step 4 : Find adjacent of c. Push d, mark it visited. Later d will be popped and printed.	d	a 1, b 1, c 1, d 1, e, f, g	a, b, c, d

Step 5 : Find adjacent of d. Push f, mark it as visited. Later f will be popped and printed.	f	a 1, b 1, c 1, d 1, e, f 1, g	a, b, c, d, f
Step 6 : Find adjacent of f. Push e, mark it as visited. Later e will be popped and printed.	e	a 1, b 1, c 1, d 1, e 1, f 1, g	a, b, c, d, f, e
Step 7 : Find adjacent of e. Push g, mark it as visited. Later g will be popped and printed.	g	a 1, b 1, c 1, d 1, e 1, f 1, g 1	a, b, c, d, f, e, g
As all nodes are visited and stack is empty we will stop traversing.			

The DFS sequence is **a, b, c, d, f, e, g**.

Q.17 Explain Depth First Search Method *(Refer Q.15 and Q.16)* [JNTU : May-18, Marks 5]

Q.18 For the graph given below, find BFS. [JNTU : Part B, Marks 5]

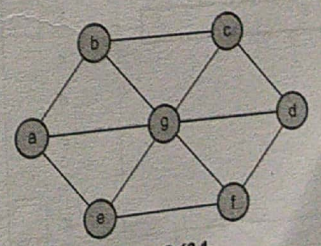
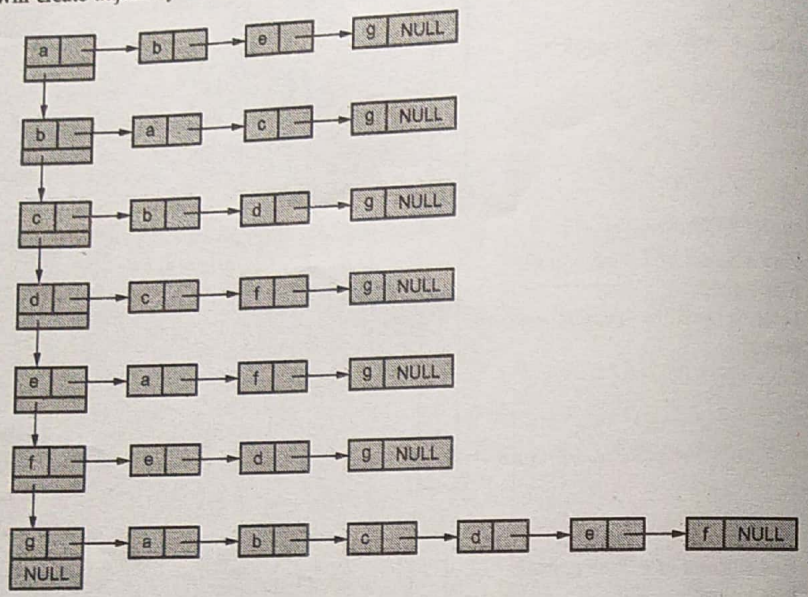


Fig. Q.18.1

Scanned by CamScanner

Ans. : We will create adjacency list for given graph.



Breadth first search (BFS)

Action	Queue	Visited array	Output
Step 1 : Insert a in queue. Mark visited (a) = 1	a	a 1, b, c, d, e, f, g	
Step 2 : Delete a and print	~~a~~	a 1, b, c, d, e, f, g	a

Step 3 : Find adjacent of a and insert them in queue. Mark corresponding adjacent nodes as visited.	~~a~~ b e g	a 1, b 1, c, d, e 1, f, g 1	a
Step 4 : Delete b and print. Find adjacent of b and if those nodes are not visited then insert them in the queue. Mark corresponding nodes as visited.	~~a~~ ~~b~~ e g c	a 1, b 1, c 1, d, e 1, f, g 1	a, b
Step 5 : Delete e and print. Find adjacent of e and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.	~~a~~ ~~b~~ ~~e~~ g c f	a 1, b 1, c 1, d, e 1, f 1, g 1	a, b, e
Step 6 : Delete g and print. Find adjacent of g and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.	~~a~~ ~~b~~ ~~e~~ ~~g~~ c f d	a 1, b 1, c 1, d 1, e 1, f 1, g 1	a, b, e, g
Step 7 : Delete c and print. Find adjacent of c and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.	~~a~~ ~~b~~ ~~e~~ ~~g~~ ~~c~~ f d	a 1, b 1, c 1, d 1, e 1, f 1, g 1	a, b, e, g, c

Scanned by CamScanner

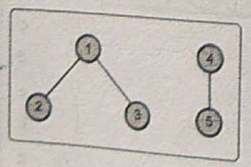
Step 8 : Delete f and print. Find adjacent of f and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.	a b e g c f d	a 1 b 1 c 1 d 1 e 1 f 1 g 1	a, b, e, g, c, f
Step 9 : Delete d and print. Find adjacent of d and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.	a b e g c f d	a 1 b 1 c 1 d 1 e 1 f 1 g 1	a, b, e, g, c, f, d
Step 10 : As queue is empty and all entries in the visited array are visited them stop. At display we will get BFS sequence.			

Hence BFS sequence is a, b, e, g, c, f, d.

Q.19 Explain how BFS can be used to identify the connected components in a graph with an example

[JNTU : Part B, Marks 5]

Ans. :
- The BFS algorithm is used to find the connected components of graph.
- In BFS we begin with particular vertex V and find entire connected components containing V.
- In BFS the vertices at all levels are visited one by one.
- For example - Consider the graph



By BFS we get (i) 1, 2, 3 and (ii) 4, 5 as two connected components.

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

Q.20 What is the difference between DFS and BFS traversals ?

[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 n 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 3]

Ans. : The graph theory is used widely in the computer science very widely. The applications of graph theory are -

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

PART II SORTING

4.2 : Introduction to Sorting

Q.23 Define sorting and list sorting methods.

[JNTU : Part A, May-15, Dec.-16 Marks 2]

Ans. : Sorting is a technique of arranging the elements in ascending or descending order. Various 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 searching ?

[JNTU : Part A, Dec.-16, Marks 3]

Ans. : Sorting is a technique of arranging the data in ascending or descending order.
Searching is a technique of finding the desired value from the list of elements.

Q.25 What is the need for sorting ?

[JNTU : Part A, Marks 2]

Ans. : The sorting is useful for -

1. Searching the desired data efficiently.
2. Responding to the queries.

Q.26 What is the meaning of sort key ?

[JNTU : Part A, Marks 2]

Ans. : Sort key is a field in the record based on which the sorting is conducted.

Scanned by CamScanner

Q.27 Name the slowest and fastest sorting technique. ☞[JNTU : Part A, Marks 2]

Ans. : Insertion sort is a slowest sorting technique and quick sort is the fastest sorting technique.

Q.28 Differentiate between internal and external sorting. ☞[JNTU : Part A, Marks 2]

Ans. : **Internal sorting :** This is a type of sorting technique in which data resides on main memory of computer.

External sorting : This is a sorting technique in which there is huge amount of data and it resides on secondary storage devices while sorting.

Q.29 Enlist four internal sorting techniques.

Ans. : Following are some internal sorting techniques -

1. Insertion sort 2. Selection sort 3. Radix sort 4. Heap Sort

Q.30 Explain the meaning of the term passes in context with sorting. ☞[JNTU : Part A, Marks 2]

Ans. : While sorting the elements in some specific order, there is lot of arrangement of elements. The phases in which the elements are moving to acquire their proper position is called passes.

Q.31 What is ascending and descending order ? ☞[JNTU : Part A, Marks 2]

Ans. : Ascending order is the sorting order in which the elements are arranged from low value to high value. In other words it is called increasing order. For example : 10, 20, 30, 40.

Descending order is the sorting order in which the elements are arranged from high value to low value. In other words it is called decreasing order. For example : 40, 30, 20, 10.

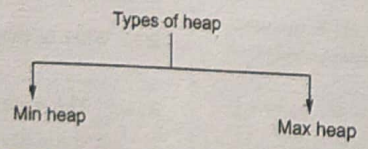
4.2.1 : Heap Sort

Q.32 What do you mean by heap ? ☞[JNTU : Part A, Marks 2]

Ans. : Heap is a complete binary tree or almost complete binary tree in which every parent node be either greater or lesser than the parent node.

Q.33 Explain the two types of heap. ☞[JNTU : Part A, Marks 3]

Ans. : Heap can be **min heap** or **max heap.**



A **max heap** is a tree in which value of each node is greater than or equal to the value of its children nodes.

For example :

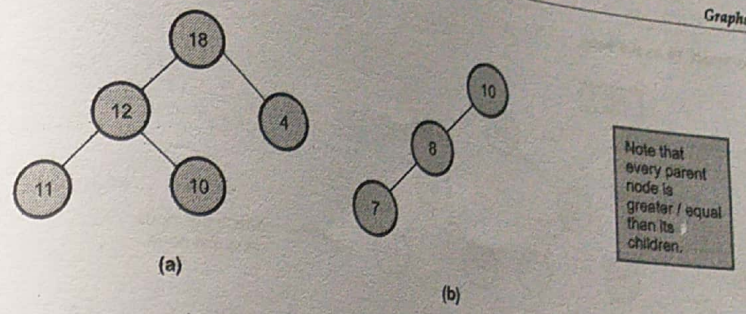


Fig. Q.33.1 Max heap

A **min heap** is a tree in which value of each node is less than or equal to value of its children nodes.

For example :

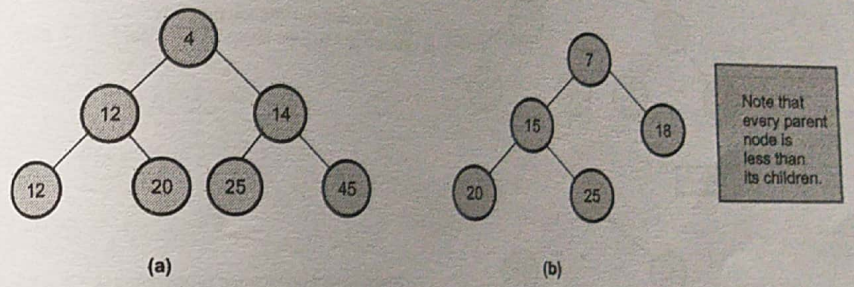


Fig. Q.33.2 Min heap

Q.34 Define a Max heap. *(Refer Q.33)* ☞[JNTU : May-18,Dec.-18, Marks 2]

Q.35 Explain the insertion of element in Heap

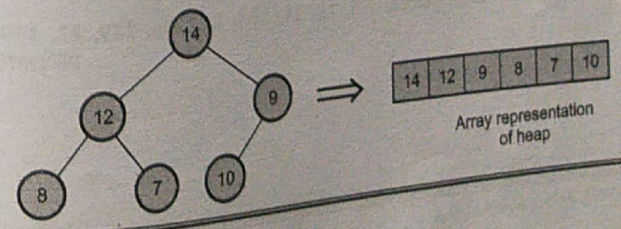
Ans. : The algorithm for inserting an element in the heap can be as given below -

Algorithm for insertion of element :

1. Insert element at the last position in heap.
2. Compare with its parent, swap the two nodes if parental dominance condition gets violated.

   [Parental dominance = Parent is greater than its children]
3. Continue comparing the new element with nodes up, each time by satisfying parental dominance condition.

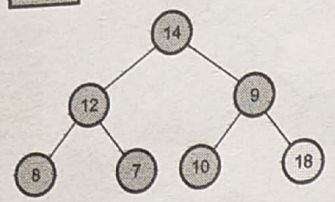
For example :



Scanned by CamScanner

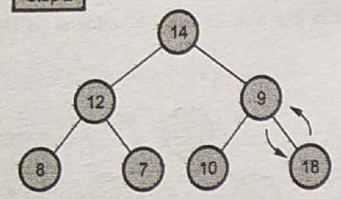
If we have to insert 18 to the heap.

Step 1



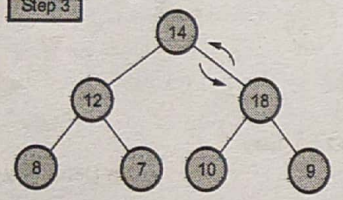
Attach this node as a last node.

Step 2



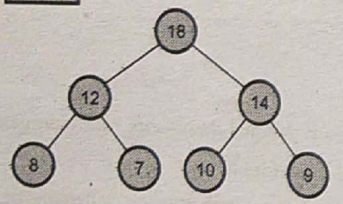
Since parent dominance property is violeted (9 <18) we swap two values.

Step 3



Since parent dominance property is violeted (14 <18), we swap two values.

Step 4



Thus we get a heap by inserting element.

In top down approach the heap is constructed from top to bottom scanning of a tree, each time checking parent dominance property.

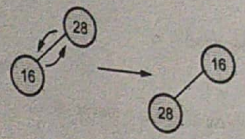
Q.36 Create max heap for the following elements (28, 16, 14, 103, 52, 105, 139, 27, 190)

[JNTU : Dec.-17, Marks 5]

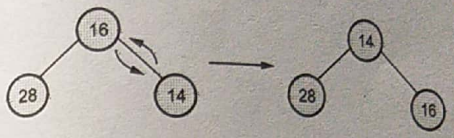
Ans. :

Step 1 : Inset 28

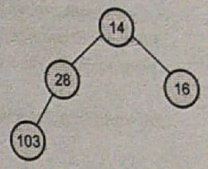
Step 2 : Insert 16



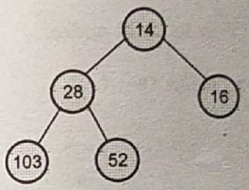
Step 3 : Insert 14



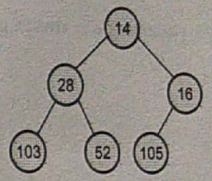
Step 4 : Insert 103



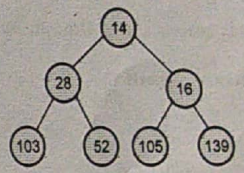
Step 5 : Insert 52



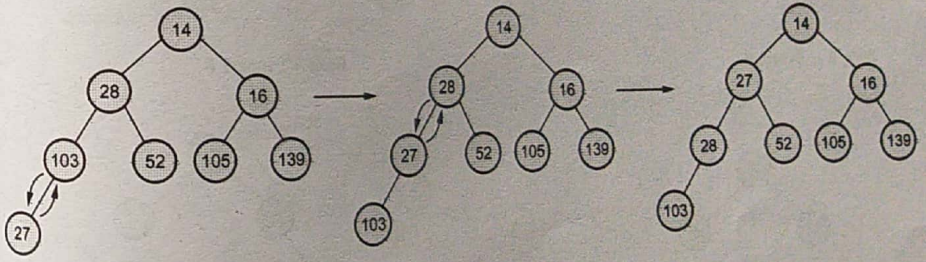
Step 6 : Insert 105



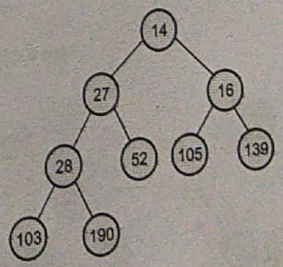
Step 7 : Insert 139



Step 8 : Insert 27



Step 9 : Insert 190



Scanned by CamScanner

Q.37 Write a function to insert n elements in a heap one by one. [JNTU : Part B, Marks 5]

Ans. :

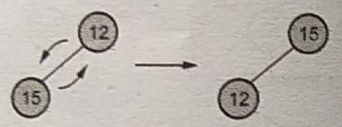
```
void Heap insert(int item)
  {
  int temp;
  //temp node starts at leaf and moves up
  temp=++size;
  while(temp!=1 && heap[temp/2]<item)
  {
 // the element cannot be placed in array H
  H[temp]=H[temp/2];//moving element down
   temp=temp/2;//finding the parent
  }
  H[temp]=item;//placing the element at its position
 }
```

Q.38 Define a Max heap. Construct a Max heap for the following {12, 15, 9, 8, 10, 18, 7, 20, 25}. [JNTU : Part B, Dec.-16, Marks 5]

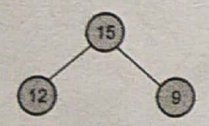
Ans. : Max heap is a heap structure in which the parent node is always greater than its child nodes.

Step 1 : Insert 12

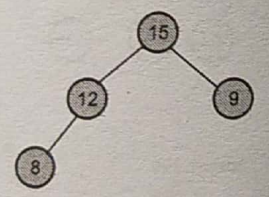
Step 2 : Insert 15



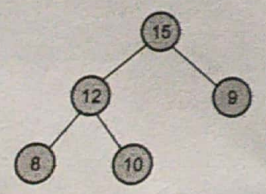
Step 3 : Insert 9



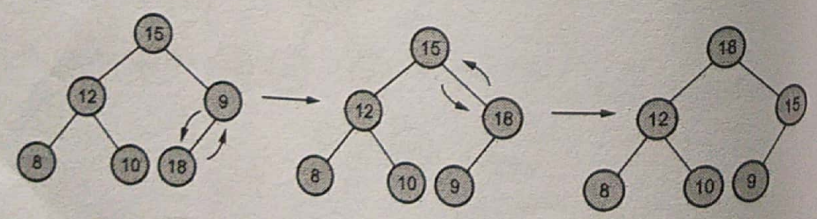
Step 4 : Insert 8



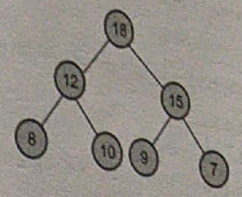
Step 5 : Insert 10



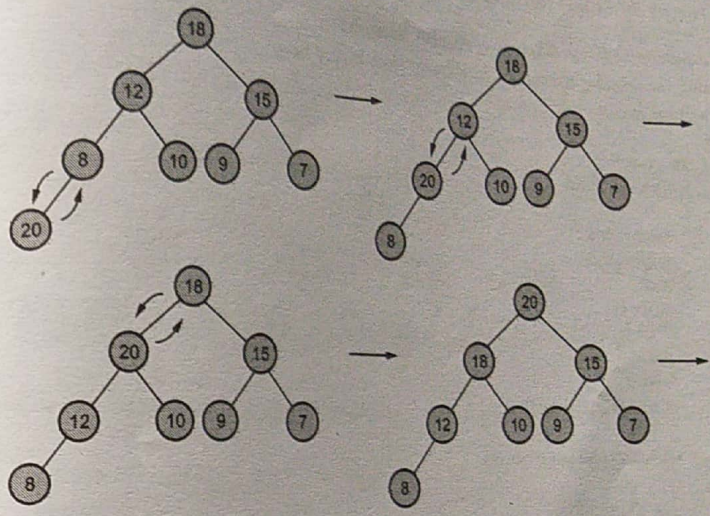
Step 6 : Insert 18



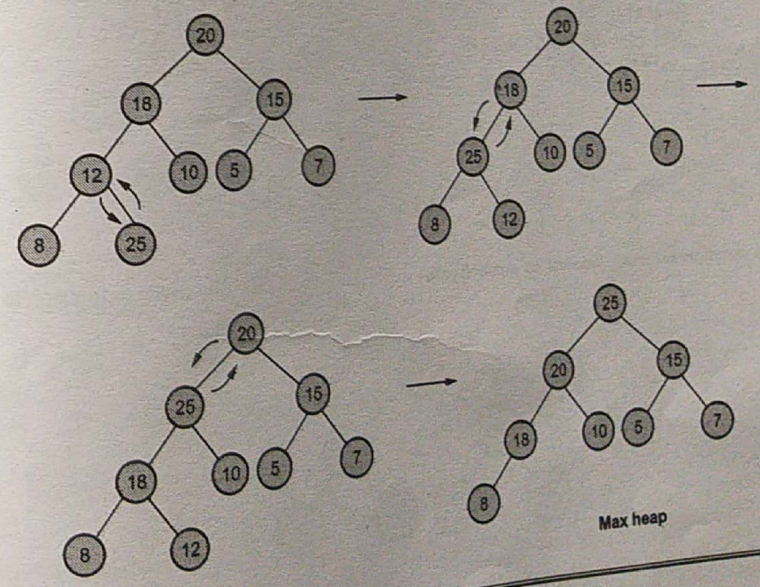
Step 7 : Insert 7



Step 8 : Insert 20



Step 9 : Insert 25



Max heap

Scanned by CamScanner

Q.39 Give an algorithm to delete an element from heap [JNTU : Part A, Marks 3]

Ans. : Algorithm for deletion of element from heap :

The root of a heap can be deleted and the heap fixed up can be as follows :

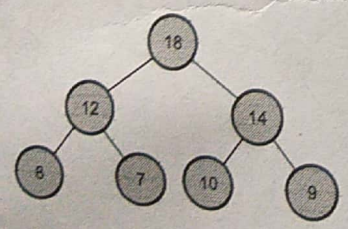
1. Exchange the root with the last leaf.
2. Decrease the heap's size by 1.
3. Heapify the smaller tree using bottom up construction algorithm.

Q.40 Write a pseudo code for deletion of element fromheap

Ans. : Function for deleting the minimum element from binary heap

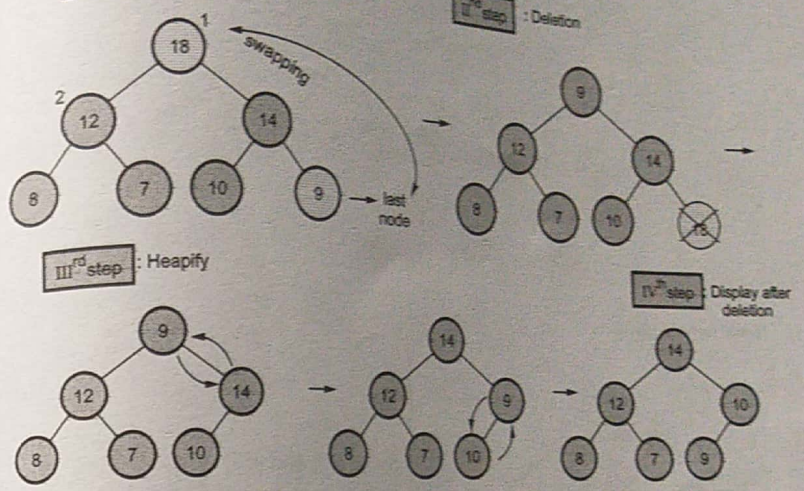
```
void del_node()
{
    int i,k,temp,j;
    for(i=n-1;i>0;i--)
    {
        temp=arr[i];
        arr[i]=arr[0];
        k=0;
        if(i==1)
            j=-1;
        else
            j=1;
        if(i>2&&arr[2]>arr[1])
            j=2;
        while(j>=0&&temp<arr[j])
        {
            arr[k]=arr[j];
            k=j;
            j=2*k+1;
            if(j+1<=i-1&&arr[j]<arr[j+1])
                j++;
            if(j>i-1)
                j=-1;
        }
        arr[k]=temp;
    }
}
```

Q.41 Perform deletion on heap present in given figure [JNTU : Part B, Marks 5]



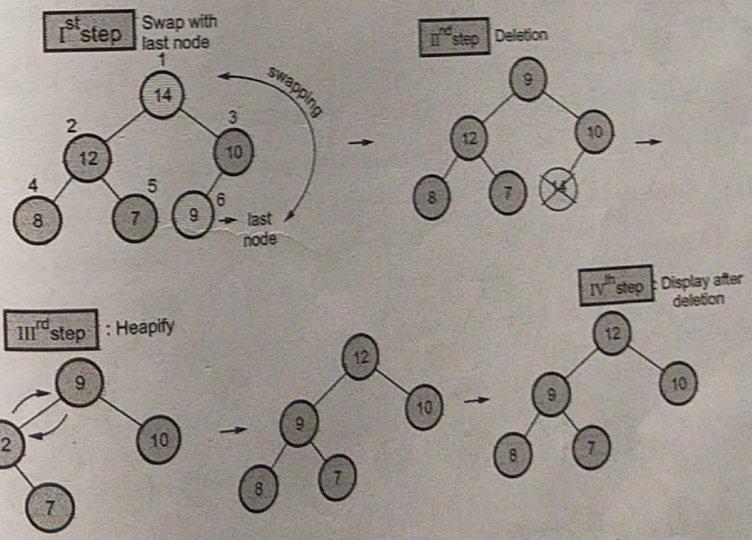
Ans. :

Deletion of key 18



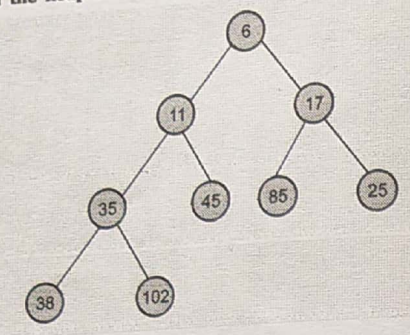
Continuing in this fashion we can perform deletion operation on given heap.

Deletion of key 14



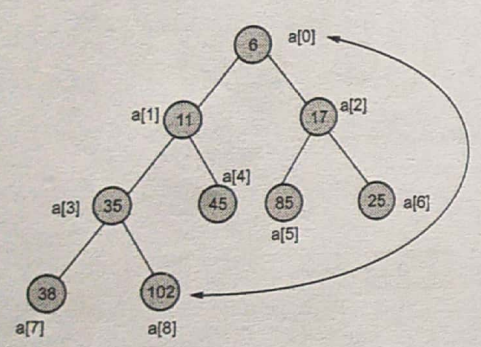
Scanned by CamScanner

Q.42 Delete the element 6 from the heap structure given below. ☞[JNTU : Part B, Marks 5]



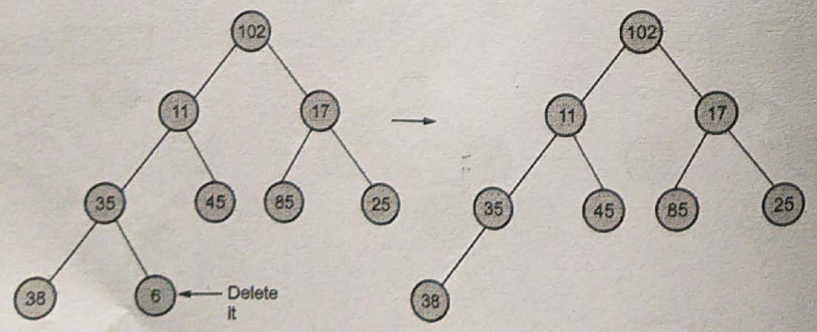
Ans. :

Step 1 : Swap root and last node.

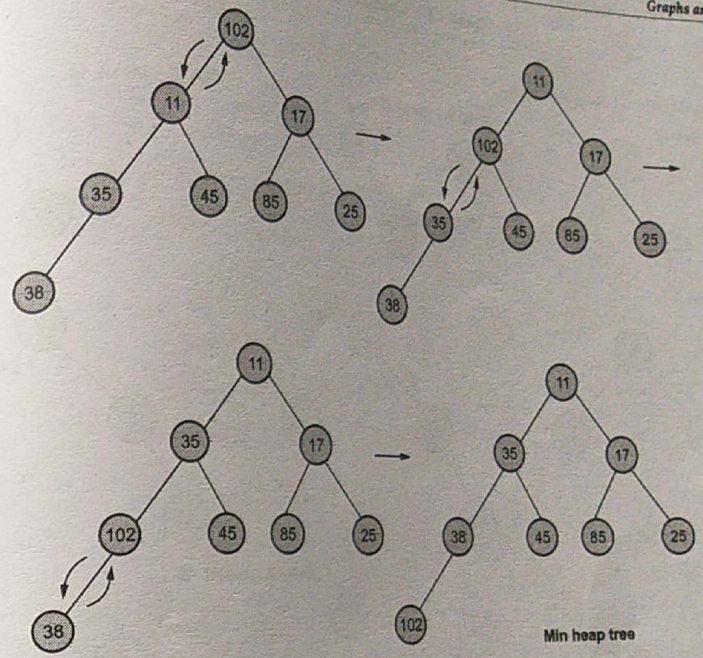


Swap a[0] and a[8]
i.e.
Swap first and last and last element in heap.

Step 2 : Delete the last node.



Step 3 : Heapify to maintain min heap



Min heap tree

Q.43 What are the applications of heap ? ☞[JNTU : Part A, Marks 2]

Ans. :

1) Heap is used for implementing the priority queue.
2) Heap is also useful for sorting the elements using a sorting technique called heap sort.

Q.44 What are the two stages in which heap sort is conducted ? ☞[JNTU : Part A, Marks 2]

Ans. : Following are the two stages in which the heap sort is conducted -

(1) Heap construction : The heap structure is build for given list of elements

(2) Deletion of maximum key : In this phase the root key is deleted for n-1 times.

Q.45 Create a heap and sort following list elements. {2, 8, 10, 6, 24, 40, 6, 11, 9, 18, 14} ☞[JNTU [H] : Part B, Nov.-15, Marks : 5]

Ans. : • For sorting the elements in ascending order, we have to create min heap. The minimum heap is a heap strcuture in which the parent node or root node has smaller value than its child nodes.

• The heap sort is carried out in two phases -

(i) construction of heap, ii) Deletion of node.

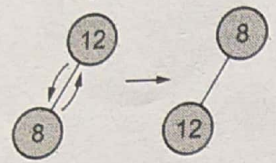
Scanned by CamScanner

Phase - I : Construction of Heap

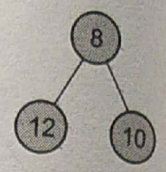
We will construct min heap structure.

Step 1 : Insert 12

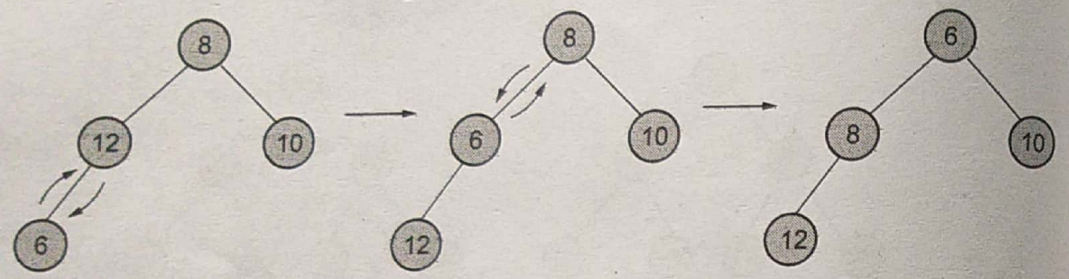
Step 2 : Insert 8



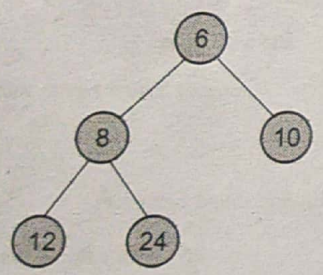
Step 3 : Insert 10



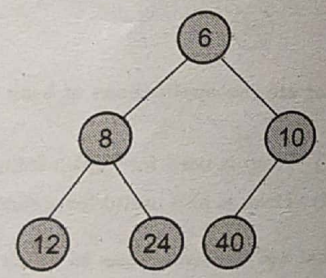
Step 4 : Insert 6



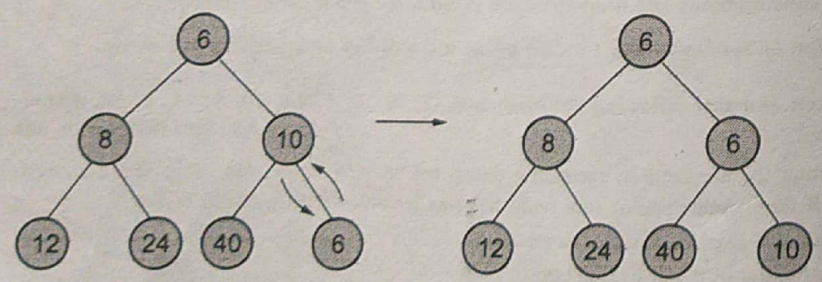
Step 5 : Insert 24



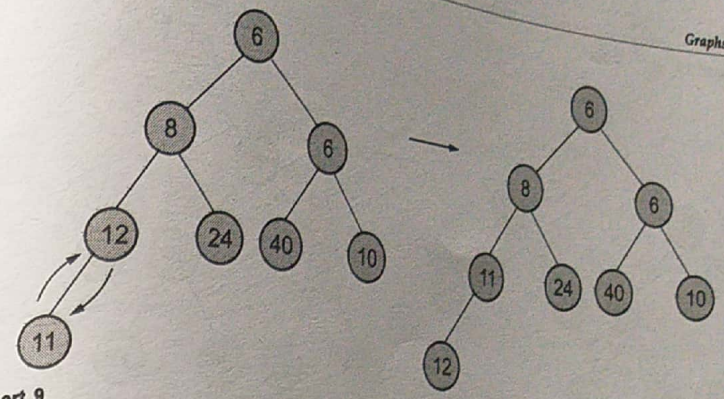
Step 6 : Insert 40



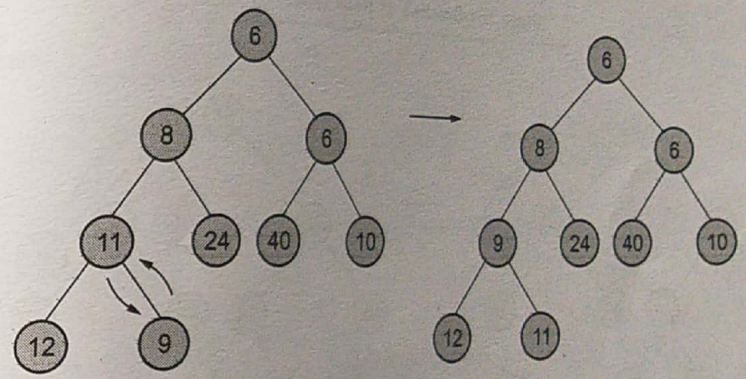
Step 7 : Insert 6



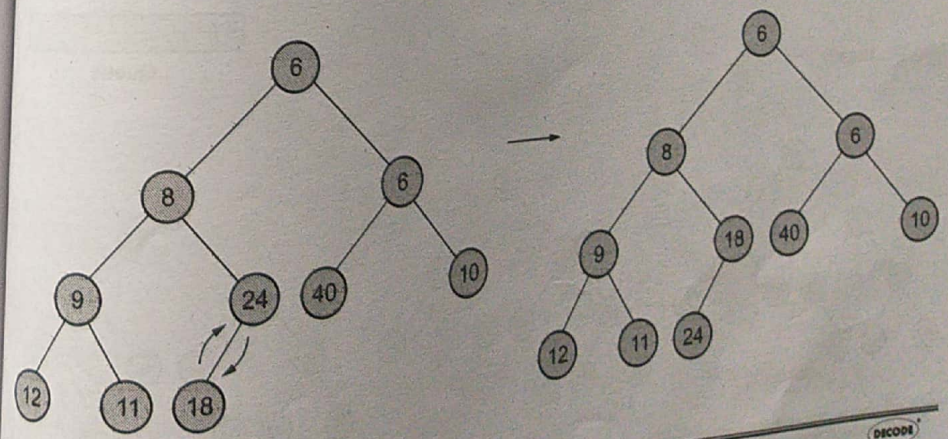
Step 8 : Insert 11



Step 9 : Insert 9

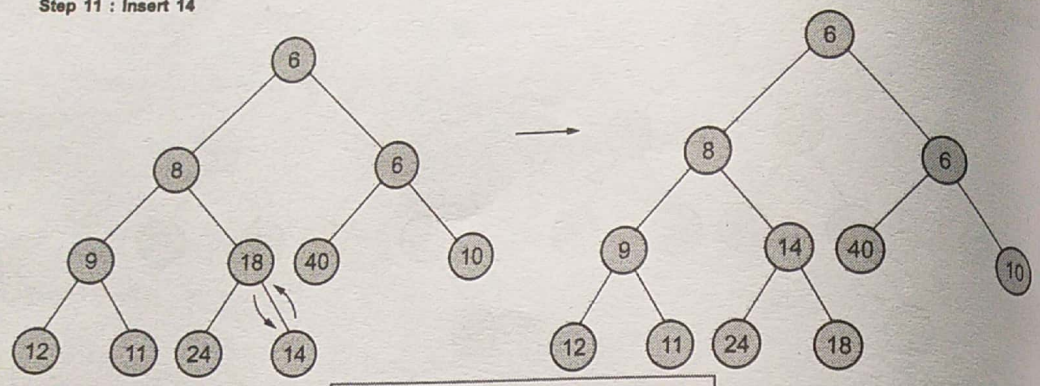


Step 10 : Insert 18



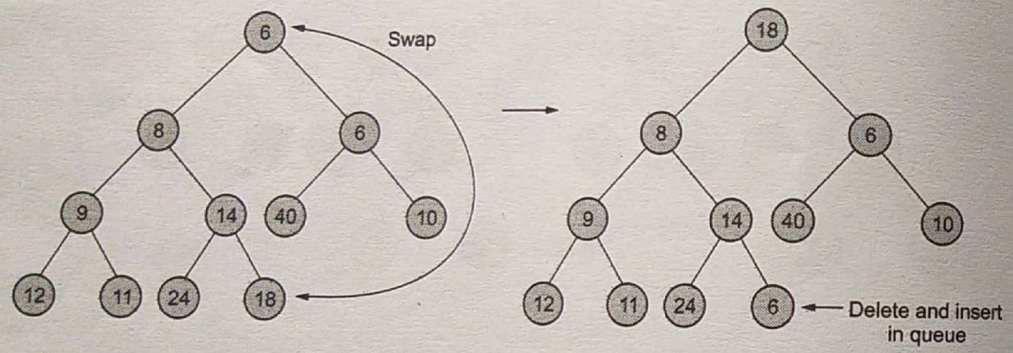
Scanned by CamScanner

Step 11 : Insert 14



Stage II : Deletion of Root Node

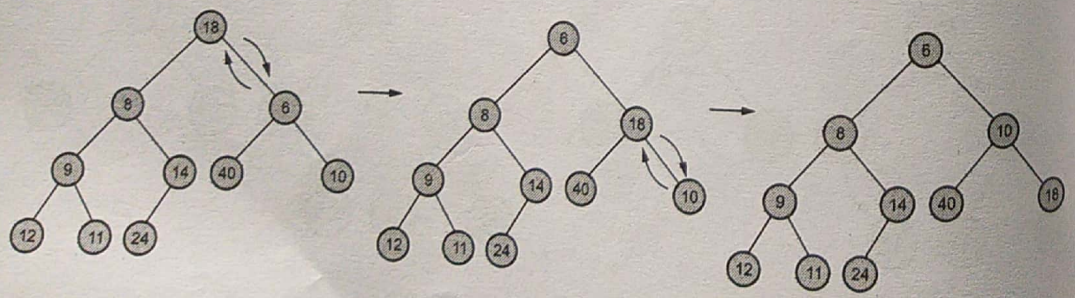
Step 1a :



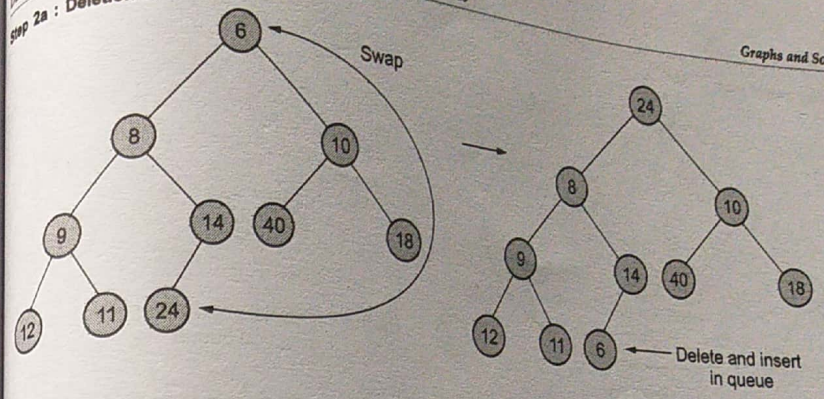
6	

Queue

Step 1b : Heapify



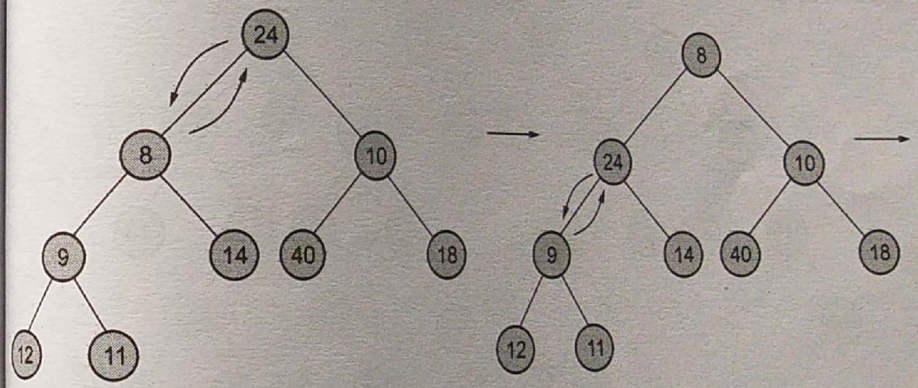
Step 2a : Deletion of root

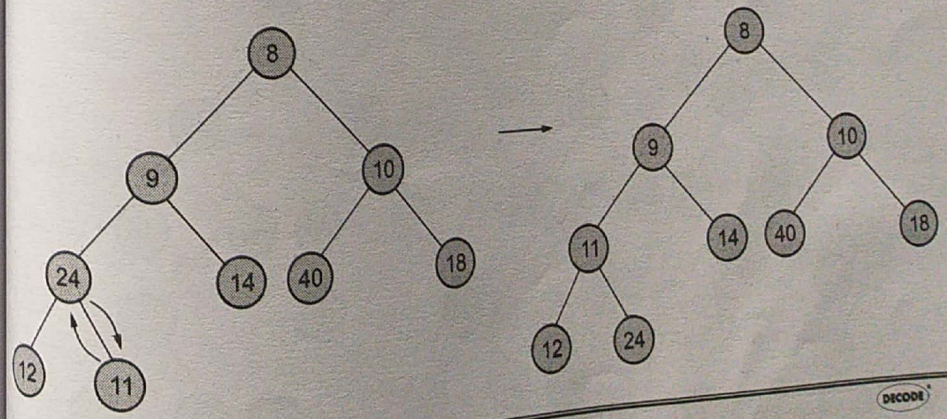


6	6

Queue

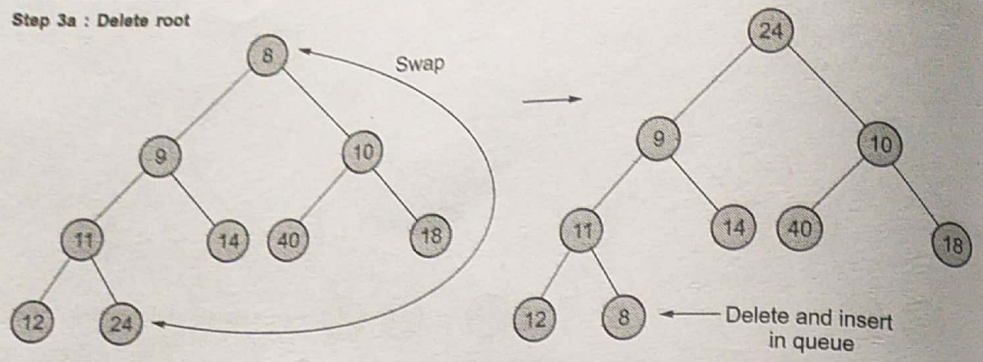
Step 2b : Heapify





Scanned by CamScanner

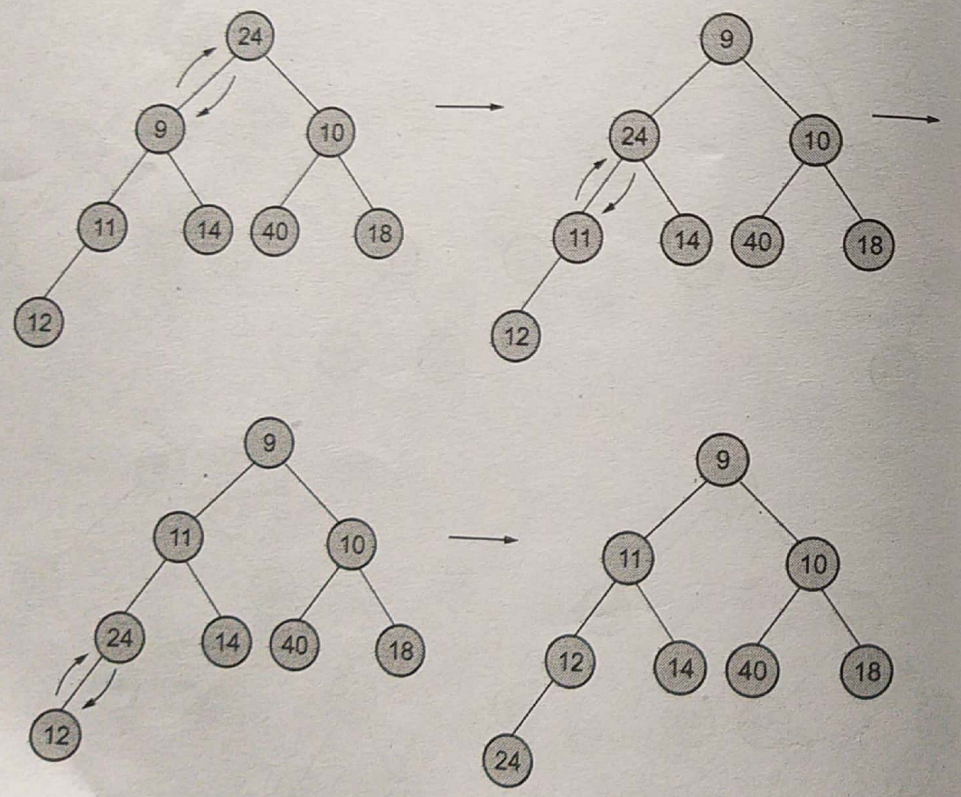
Step 3a : Delete root



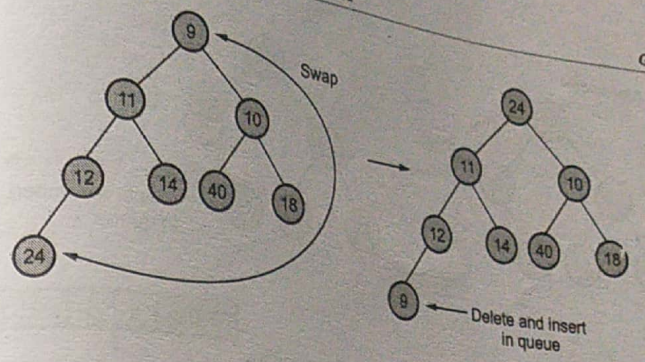
6	6	8	

Queue

Step 3b : Heapify



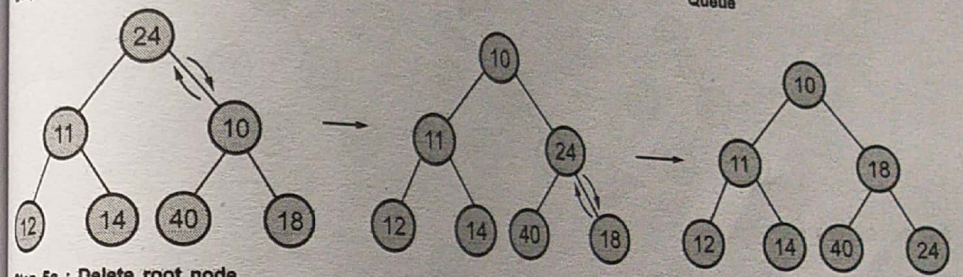
Step 4a : Delete a root node



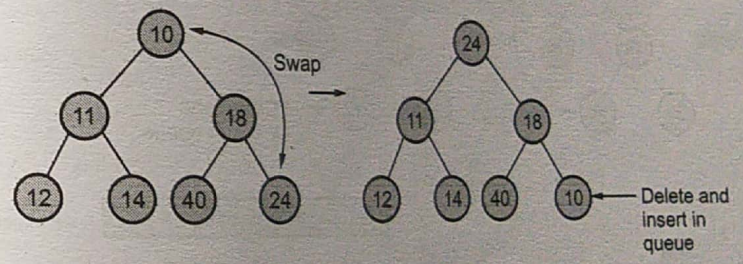
6	6	8	9

Queue

Step 4b : Heapify



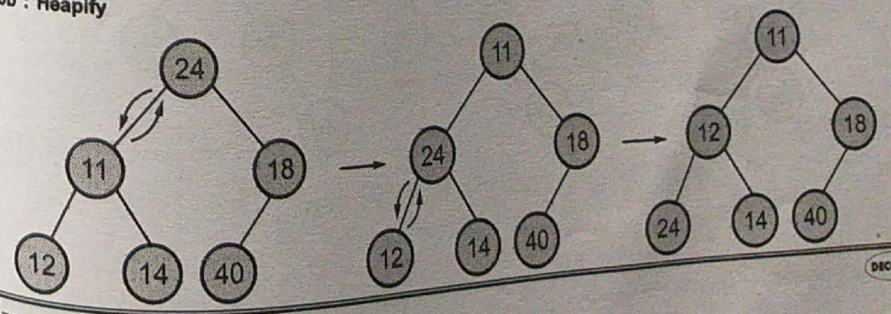
Step 5a : Delete root node



6	6	8	9	10

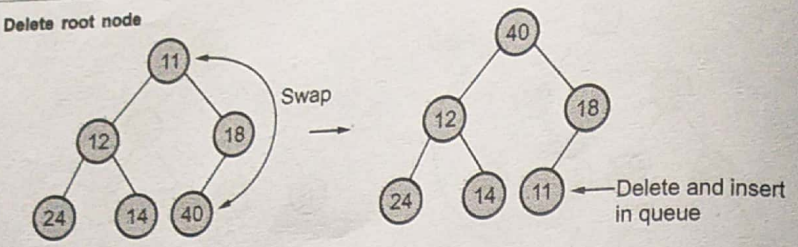
Queue

Step 5b : Heapify



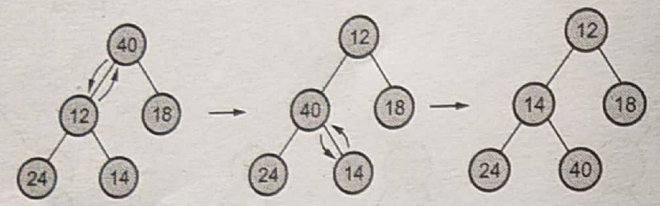
Scanned by CamScanner

Step 6a : Delete root node

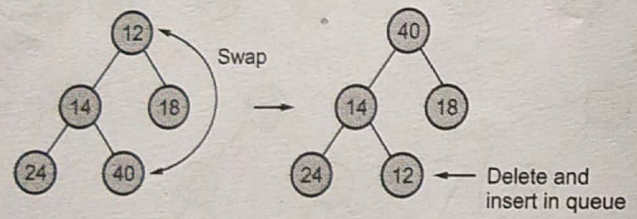


6	6	8	9	10	11

Step 6b : Heapify

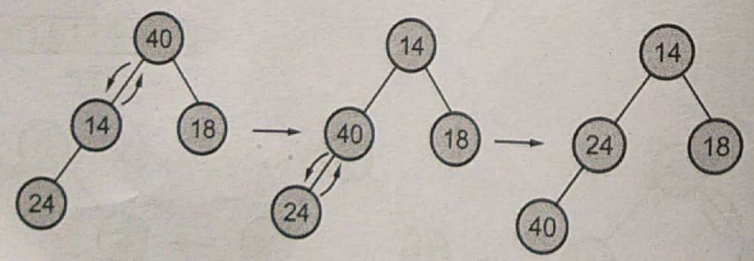


Step 7a : Delete a root

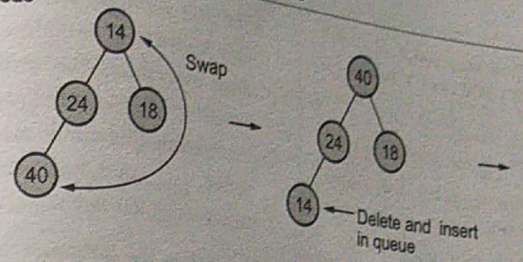


6	6	8	9	10	11	12

Step 7b : Heapify



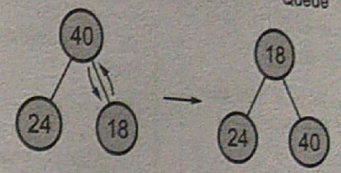
Step 8a : Delete root node



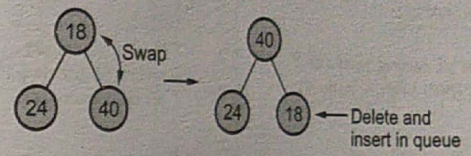
6	6	8	9	10	11	12	14

Queue

Step 8b : Heapify



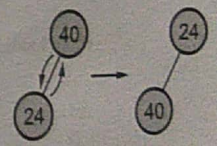
Step 9a : Delete root node



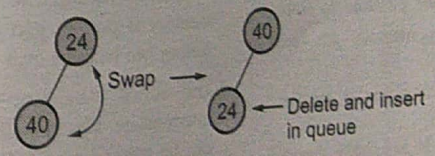
6	6	8	9	10	11	12	14	18

Queue

Step 9b : Heapify



Step 10 : Delete root node



6	6	8	9	10	11	12	14	18	24

Queue

Step 11 : Delete a node and insert in queue

6	6	8	9	10	11	12	14	18	24	40

This is sorted list

Scanned by CamScanner

Q.46 What are the features of heap sort ? [JNTU : Part B, Marks 5]

Ans. :

1. The time complexity of heap sort is Θ (n log n).
2. This is an in-place sorting algorithm. That means it does not require extra storage space while sorting the elements.
3. For random input it works slower than quick sort.
4. Heap sort is not a stable sorting method.
5. The space complexity of heap sort is O (1). As it does not require any extra storage to sort.

Q.47 Explain about heap sorting technique with an example. *(Refer Q.45)* [JNTU : May-18, Marks 5]

4.2.2 : External Sorting

Q.48 What are the two types of sorting methods ? [JNTU : Part A, Marks 2]

Ans. :

- **Internal Sorting :**
  - The internal sorting is a sorting in which the data resides in the main memory of the computer.
  - Various methods that make use of internal sorting are -
    1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Radix Sort and so on.
- **External Sorting :**
  - For many applications it is not possible to store the entire data on the main memory for two reasons,
    i) amount of main memory available is smaller than amount of data.
    ii) Secondly the main memory is a volatile device and thus will lost the data when the power is shut down.
  - To overcome these problems the data is stored on the secondary storage devices. The technique which is used to sort the data which resides on the secondary storage devices are called external sorting.
  - The data stored on secondary memory is part by part loaded into main memory, sorting can be done over there. The sorted data can be then stored in the intermediate files. Finally these intermediate files can be merged repeatedly to get sorted data. Thus huge amount of data can be sorted using this technique.

Q.49 What is the difference between internal and external sorting? [JNTU : Part A, Marks 2]

Ans. :

Sr. No.	Internal Sorting	External Sorting
1.	The internal sorting can be performed on small amount of data.	For sorting large amount of data external sorting is applied.
2.	In case of internal sorting all the data lies on same memory.	In case of external sorting, all the data can not be accommodated on the single memory . Some amount of memory needs to be kept on auxiliary memory such as hard disk, compact disk and so on.

Q.50 Explain model for external sorting. [JNTU : Part B, Marks 5]

Ans. :

- In external sorting, the **data stored on secondary memory** is part by part loaded into main memory, sorting can be done over there.
- The sorted data can be then stored in the intermediate files. Finally these intermediate files can be merged repeatedly to get sorted data. Thus **huge amount** of data can be sorted using this technique.
- The external merge sort is a technique in which the data is loaded in intermediate files. Each intermediate file is sorted independently and then combined or merged to get the sorted data.

For example :

- Consider that there are 10,000 records that has to be sorted. Clearly we need to apply external sorting method. Suppose main memory has a capacity to store 500 records in blocks, with each block size of 100 records.

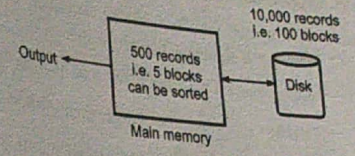


Fig. Q.50.1

- The sorted 5 blocks (i.e. 500 records) are stored in intermediate file. This process will be repeated 20 times to get all the records sorted in chunks.
- In the second step, we start merging a pair of intermediate files in the main memory to get output file.
- This can be represented as follows -

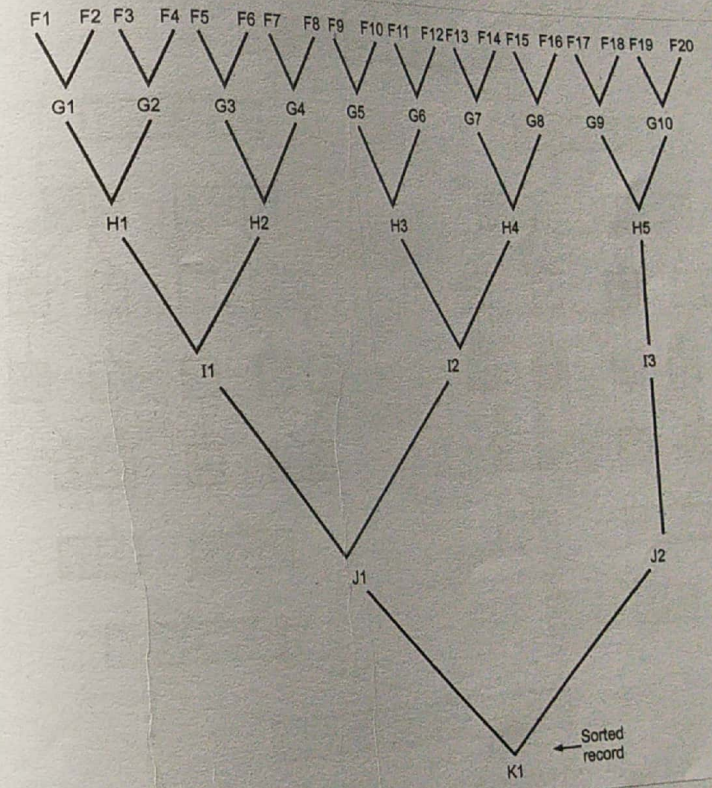


Fig. Q.50.2

Scanned by CamScanner

Q.51 Explain the concept of merge sort in detail. [JNTU : Dec.-18, Marks 5]

Ans. : The merge sort is a sorting algorithm that uses the divide and conquer strategy. In this method division is dynamically carried out.

Merge sort on an input array with n elements consists of three steps :

Divide : Partition array into two sub lists s1 and s2 with n/2 elements each.

Conquer : Then sort sub list s1 and sub list s2.

Combine : Merge s1 and s2 into a unique sorted group.

Example : Sort the following elements using merge sort.
10, 5, 7, 6, 1, 4, 8, 3, 2, 9.

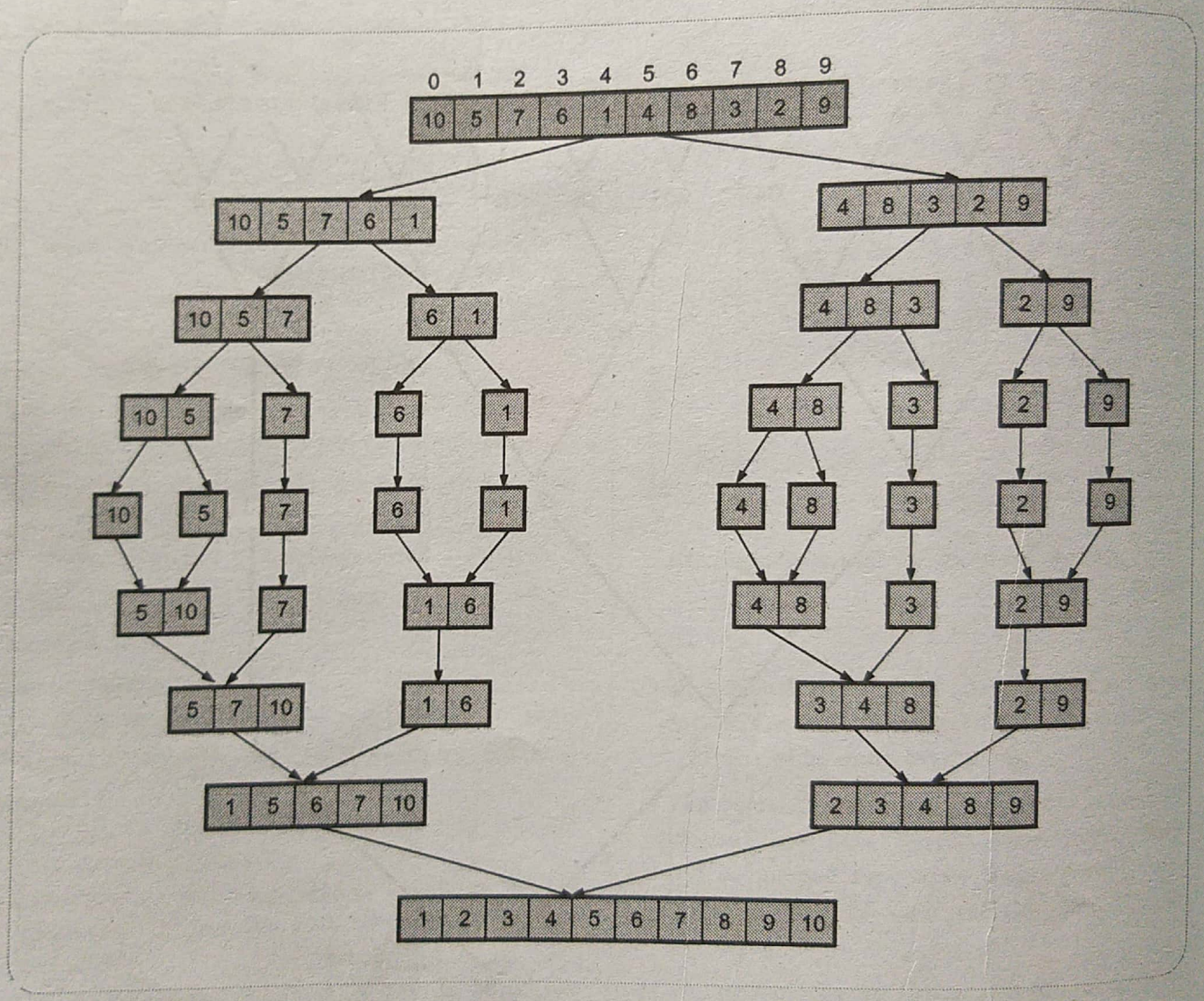


Fig. Q.51.1

Scanned by CamScanner

Q.52 Write a C function for merge sort.

☞ [JNTU : Part B, Dec.-17, Marks 5]

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**

Scanned by CamScanner

```
        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 list and elements of right sub list are remaining. Then copy the remaining elements of right sub list to **temp**

Fill in the Blanks for Mid Term Exam

Q.1 Graph is a collection of _____.

Q.2 The terminal vertices of a path of degree _____ .

Q.3 _____ data structure is used in the network data model.

Q.4 In a graph a vertex with degree 0 is called _____ .

Q.5 The number of loops in a simple graph are _____ .

Q.6 _____ data structures is used by breadth first search as an auxiliary structure to hold nodes for future processing.

Q.7 _____data structure is used in implementing a graph using adjacency matrix.

Q.8 ____data structure is used in implementing a graph using adjacency list.

Q.9 If a graph G has no edge then the corresponding adjacency matrix is called _____

Q.10 Consider a graph with 4 nodes and 5 edges the sum of degree of all the nodes is _____.

Q.11 Arranging the records in ascending or descending order is called ___.

Q.12 ___ heap is a kind of heap in which the parent node is always greater than its child nodes.

Q.13 While deleting a node in heap ___ node is always deleted.

Q.14 After deletion of a node from heap ____ operation is required.

Q.15 In the priority queues, items are entered in a____order. ☞ Aug.-16

Multiple Choice Questions for Mid Term Exam

Q.1 Which of the following data structure is non linear data structure ?

a. Stack
b. Queue
c. Linked list
d. Graphs

Q.2 In graph G if e = [v1,v2] then that means _____ .

a. v1 and v2 are adjacent to each other
b. v1 and v2 are not connected
c. v2 is source vertex and v1 is sink vertex
d. v1 and v2 are one and the same

Scanned by CamScanner

UNIT - V

5 Pattern Matching and Tries

5.1 : Pattern Matching Algorithms

5.1.1 : Brute Force Algorithm

Q.1 What is the applicability of string matching algorithm ?

Ans. : String matching algorithm is used in text processing.

Q.2 Name few string matching algorithms.

Ans. : 1. Naïve Method 2. Rabin Karp Method 3. Finite automata for string matching

Q.3 Name the algorithmic technique on which the Naïve method is used.

Ans. : The Naïve method is based on Brute Force method.

Q.4 Explain Naive string matching algorithm with example.

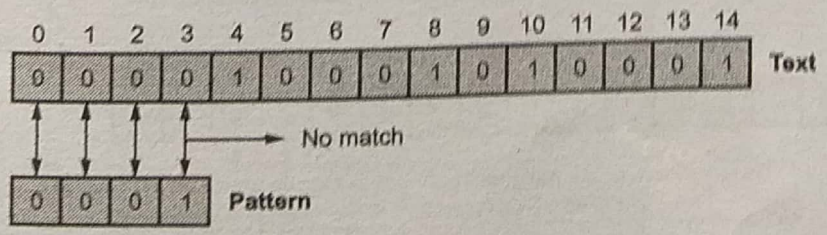
Ans. :

- This is the simplest string matching algorithm.
- This algorithm performs a checking at all positions in the text, whether an occurrence of the pattern starts there or not. Then after each attempt, it shifts the pattern by exactly one position to the right.
- If the match is found then it returns otherwise the matching process continued by shifting one character to the right.

Example - Refer Q.5.

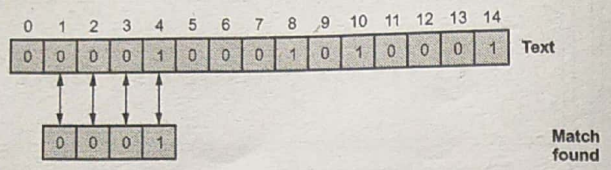
Q.5 Show the comparisons the native string matcher makes for the pattern P = 0001 in the text T = 0000 1000 1010001.

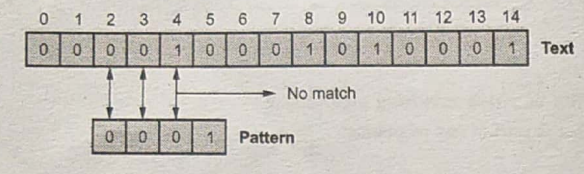
Ans. : Let,



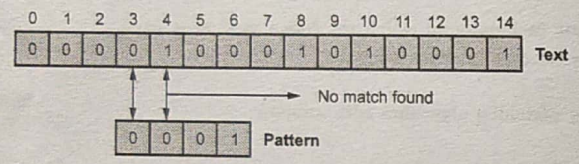
Scanned by CamScanner

Shift to right by one position.

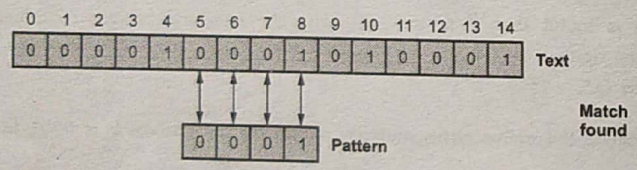




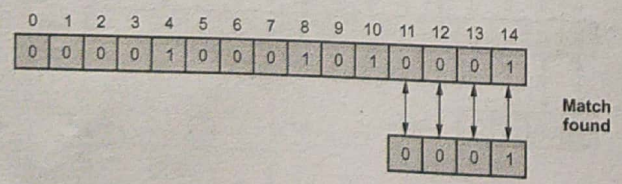
Shift pattern right by one position.



Shift pattern right by one position, each time and check pattern against text. The matched pattern will be



Similarly



Thus pattern matching can be obtained.

5.1.2 : Boyer Moore Algorithm

Q.6 Explain Boyer Moore Pattern Matching Algorithm with suitable example.

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

Example - Refer Q.7.

Q.7 Find the Boyer-Moore string matching algorithm for the given pattern against the text.
Text : abcaabccaabbabca Pattern abc.

Ans. : 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
Text	a	b	c	a	a	b	c	c	a	a	b	b	a	b	c	a

	0	1	2
Pattern	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.

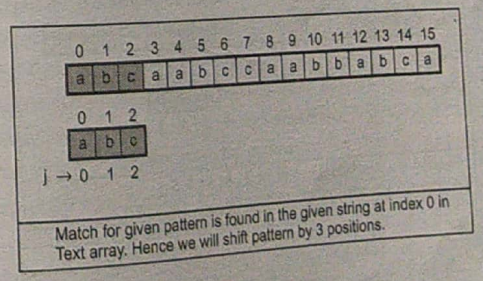
0	1	2
a	b	c

This is the right most occurrence of a ∴ last (a) = 0

This is the right most occurrence of b ∴ last (b) = 1

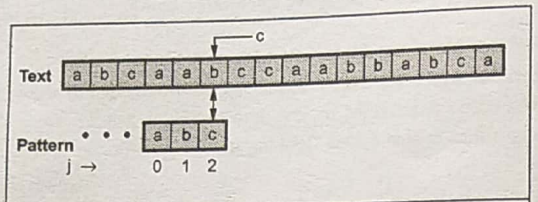
This is the right most occurrence of c ∴ last (c) = 2

Step 1 :



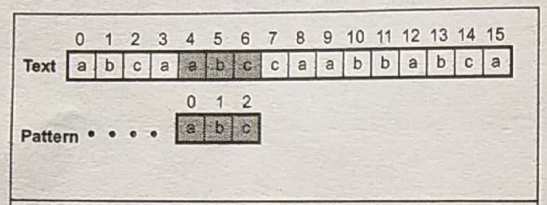
Scanned by CamScanner

Step 2 :



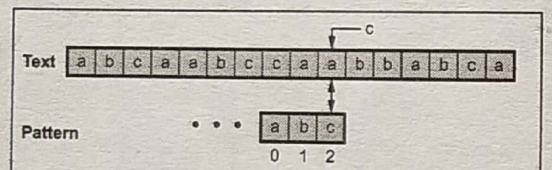
We will find l = last (c) = 1 and j = 2 As $l+1 \le j$
i.e. 1 + 1 = 2. We shift the pattern by j–t i.e. 2–1 = 1 Position.

Step 3 :



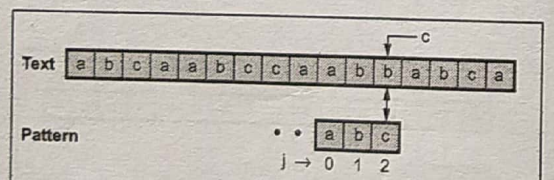
A match is found at index 4 in text. Hence we will shift the pattern by 3 positions.

Step 4 :



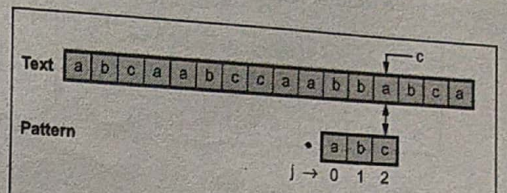
We will find l = last (a) = 0, j = 2 As $1 + l \le j$
i.e. $1 + 0 \le 2$ we shift the pattern by $j - l$ i.e. $2 - 0 = 2$.

Step 5 :



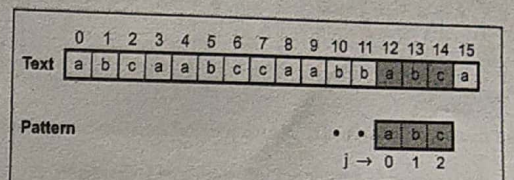
We will find l = last (b) = 1 and j = 2 As $l + 1 \le j$
i.e. $1+1 \le 2$ we will shift the pattern by $j - l$ i.e. $2 - 1 = 1$.

Step 6 :



We will find l = last (a) = 0 and j = 2 As $l + 1 \le j$
i.e. $0 + 1 \le 2$. 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 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.

Scanned by CamScanner

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

– 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.

Scanned by CamScanner

Step 8 :

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

Thus we found a match for pattern at starting position Text[15].

Q.10 Explain with the help of example, the process of building the prefix table.

Ans. :

Consider

Initially we will put 0 in 0th location of prefix array.	0 1 2 3 4 5 a b a d a b **0**
Consider string **ab** No match of prefix and suffix. Hence we will put 0 in prefix array at 1st location.	**Prefix :** ε, a **Suffix :** ε, b 0 1 2 3 4 5 a b a d a b 0 **0**
Consider string **aba** The length of matching prefix-suffix is 1. Hence make entry 1 in prefix table.	**Prefix :** ε, a, ab **Suffix :** ε, a, ba 0 1 2 3 4 5 a b a d a b 0 0 **1**
Consider string **abad** The length of matching prefix-suffix is 0. Hence make entry 0 in prefix table.	**Prefix :** ε, a, ab, aba **Suffix :** ε, d, ad, bad 0 1 2 3 4 5 a b a d a b 0 0 1 **0**
Consider string **abada** The length of matching prefix-suffix is 1. Hence make entry 1 in prefix table.	**Prefix :** ε, a, ab, aba, abad **Suffix :** ε, a, da, ada, bada 0 1 2 3 4 5 a b a d a b 0 0 1 0 **1**

Consider string **abadab** The length of matching prefix-suffix is 2. Hence make entry 2 in prefix table.	**Prefix :** ε, a, ab, aba, abad, abada **Suffix :** ε, b, ab, dab, adab, badab 0 1 2 3 4 5 a b a d a b 0 0 1 0 1 **2**

Q.11 Give the fail indexes used by the KMP algorithm for the following patterns
(a) AAAB (b) AABAACAABABA (c) ABRACADABRA (d) ASTRACASTRA

Ans. :

(a) AAAB

Intially put 0 in 0th location of failure array.	0 1 2 3 A A A B 0
Consider string AA. Length of matching prefix-suffix pair is 1. Hence put 1 at index 1 in failure array.	Prefix ε, A Suffix ε, A 0 1 2 3 A A A B 0 1
Consider string AAA. Length of matching prefix-suffix pair is 2. Hence put 2 at index 2 in failure array.	Prefix ε, A, AA Suffix ε, A, AA 0 1 2 3 A A A B 0 1 2
Consider string AAAB. No match. Hence put 0 at index 3.	Prefix ε, A, AA, AAA Suffix ε, B, AB, AAB 0 1 2 3 A A A B 0 1 2 0

Scanned by CamScanner

(b) AABAACAABABA

0	1	2	3	4	5	6	7	8	9	10	11
A	A	B	A	A	C	A	A	B	A	B	A
0	1	0	1	2	0	1	2	3	4	0	0

(c) ABRACADABRA

0	1	2	3	4	5	6	7	8	9	10
A	B	R	A	C	A	D	A	B	R	A
0	0	0	1	0	1	0	1	2	3	4

(d) ASTRACASTRA

0	1	2	3	4	5	6	7	8	9	10
A	S	T	R	A	C	A	S	T	R	A
0	0	0	0	1	0	1	2	3	4	5

Q.12 Discuss KMP string matching algorithm and also find the prefix function for the following pattern : ababbabb.

Ans. : KMP algorithm : Refer Q.9.

Initially, put 0 in 0th location of failure array.	0 1 2 3 4 5 6 7 a b a b b a b b 0 Failure array
Consider string ab. There is no matching prefix. Suffix pair. Hence put 0 in the array at index 1.	Prefix : ε, a Suffix : ε, b 0 1 2 3 4 5 6 7 a b a b b a b b 0 0
Consider string aba. There the matching prefix-suffix pair is a. The length of this string is 1. Hence put 1 at index 2.	Prefix : ε, a, ab Suffix : ε, a, ba 0 1 2 3 4 5 6 7 a b a b b a b b 0 0 1
Consider string abab. The matching prefix-suffix pair is length whose length is 2. Hence put 2 at index 3.	Prefix : ε, a, ab, aba Suffix : ε, b, ab, bab 0 1 2 3 4 5 6 7 a b a b b a b b 0 0 1 2

Consider the string ababb. There is no matching prefix-suffix pair. Hence put 0 at index 4.	Prefix : ε, a, ab, aba, abab Suffix : ε, b, bb, abb, babb 0 1 2 3 4 5 6 7 a b a b b a b b 0 0 1 2 0
Consider the string ababba. There is only one matching prefix-suffix pair, i.e. a whose length is 1. Hence put 1 at index 5.	Prefix : ε, a, ab, aba, abab, ababb Suffix : ε, a, ba, bba, abba, babba 0 1 2 3 4 5 6 7 a b a b b a b b 0 0 1 2 0 1
Consider the string ababbab. There is only one matching prefix-suffix pair i.e. ab whose length is 2. Hence put 2 at index 6.	Prefix : ε, a, ab, aba, abab, ababb, ababba Suffix : ε, b, ab, bab, bbab, abbab, babbab 0 1 2 3 4 5 6 7 a b a b b a b b 0 0 1 2 0 1 2
Consider the string ababbabb. There is no matching prefix-suffix pair. Hence put 0 at index 7.	Prefix : ε, a, ab, aba, abab, ababb, ababba, ababbab Suffix : ε, b, bb, abb, babb, bbabb, abbabb, babbabb 0 1 2 3 4 5 6 7 a b a b b a b b 0 0 1 2 0 1 2 0

Q.13 What is string matching algorithm ? Write Knuth-Morris-Pratt algorithm and also calculate the prefix function for the pattern P = ababaaca.

Ans. : String Matching Algorithm - Refer Q.4.

KMP Algorithm - Refer Q.9.

Initially, put 0 in 0th location of failure array.	0 1 2 3 4 5 6 7 a b a b a a c a 0 failure array.
Consider string ab. There is no matching prefix - suffix pair is a. Hence put 0 at index 1.	Prefix : ε, a Suffix : ε, b 0 1 2 3 4 5 6 7 a b a b a a c a 0 0

Scanned by CamScanner

Consider string aba. There is matching prefix suffix pair i.e. 'a'. The length of 'a' is 1. Hence we write 1 at index 2.	Prefix : ε, a, ab Suffix : ε, a, ba

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

Consider string abab. The matching prefix-suffix pair is 'ab' whose length is 2.	Prefix : ε, a, ab, aba Suffix : ε, b, ab, bab

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

Consider string 'ababa'. The matching prefix-suffix pair is 'aba' whose length is 3. Hence we will write 3 at index 4.	Prefix : ε, a, ab, aba, abab Suffix : ε, a, ba, aba, baba

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

Consider string 'ababaa'. There is one matching prefix-suffix pair i.e. a. Hence put 1 at index 5.	Prefix : ε, a, ab, aba, abab, ababa Suffix : ε, a, aa, baa, abaa, babaa

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

Consider string 'ababaac'. There is no matching prefix-suffix pair. Hence put 0 at index 6.	Prefix : ε, a, ab, aba, abab, ababa, ababaa Suffix : ε, c, ac, aac, baac, abaac, babaac.

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

Consider string 'ababaaca'. The matching pair is 'a' of length 1. Hence put 1 at index 7.	Prefix : ε, a, ab, aba, abab, ababa, ababaa, ababaac Suffix : ε, a, ca, aca, aaca, baaca, abaaca, babaaca.

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

5.2 : Standard Tries

Q.14 Write a short note on - Trie tree.

Ans. : Definition : A trie (derived from retrieval) is a multiway tree data structure used for storing strings over an alphabet. It is used to store a large amount of strings. The pattern matching can be done efficiently using tries. Following Fig. Q.14.1 represents the trie.

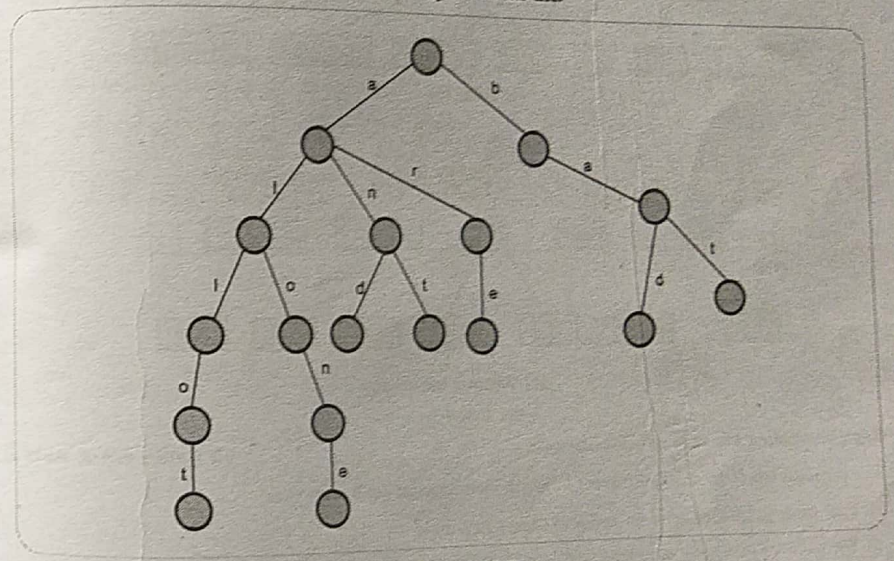


Fig. Q.14.1 Trie

The trie shows words like allot, alone, ant, and, are, bat, bad. The idea is that all strings sharing common prefix should come from a common node. The tries are used in spell checking programs.

- Preprocessing pattern improves the performance of pattern matching algorithm. But if a text is very large then it is better to preprocess text instead of pattern for efficient search.
- A trie is a data structure that supports pattern matching queries in time proportional to the pattern size.

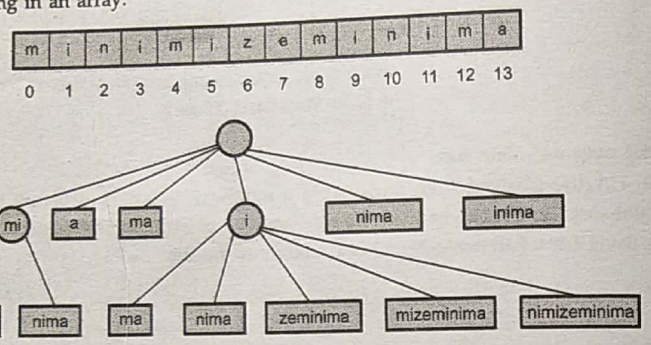
Scanned by CamScanner

Applications of tries :

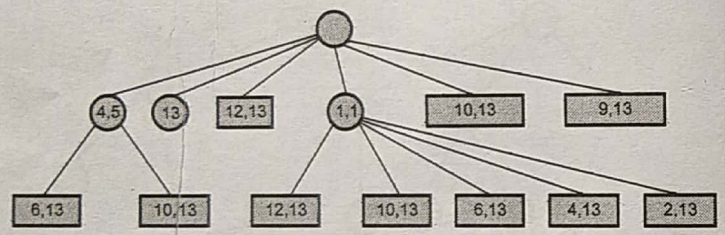
1. Tries has an ability to insert, delete or search for the entries. Hence they are used in building dictionaries such as entries for telephone numbers, English words.
2. Tries are also used in spell-checking softwares.

Q.15 Draw the compact representation of the suffix trie for the string "minimizeminima"

Ans. : Let us store the string in an array.



(a) Suffix trie



(b) Compact trie

Fig. Q.15.1

Q.16 Explain standard trie with suitable example. ☞ **[JNTU : Part B, Marks 5]**

Ans. : The standard trie is an ordered tree for building the strings of set S such that

1. Each node is labelled with an alphabet except root node.
2. The children of a node are alphabetically ordered.
3. The path from root to external node should produce a string from set S.

For example -

Consider a set of strings

S = {bean, bell, big, bull, bug, set, stock, stone}

The searching, insertion and deletion operations are done in time O(dm) where

d is size of the alphabet.

m is size of string parameter of the operation.

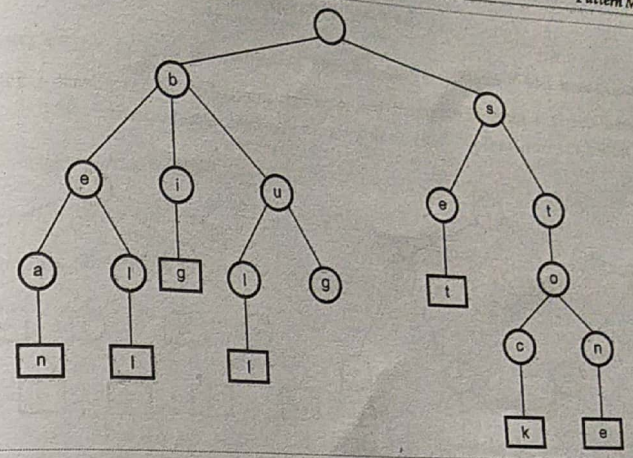


Fig. Q.16.1 Standard trie

The space complexity of standard trie is O(n) where, n is total size of string S. The word matching is done with standard trie as follows.

Consider a text

Word matching = {0, 4, 9, 13, 19}

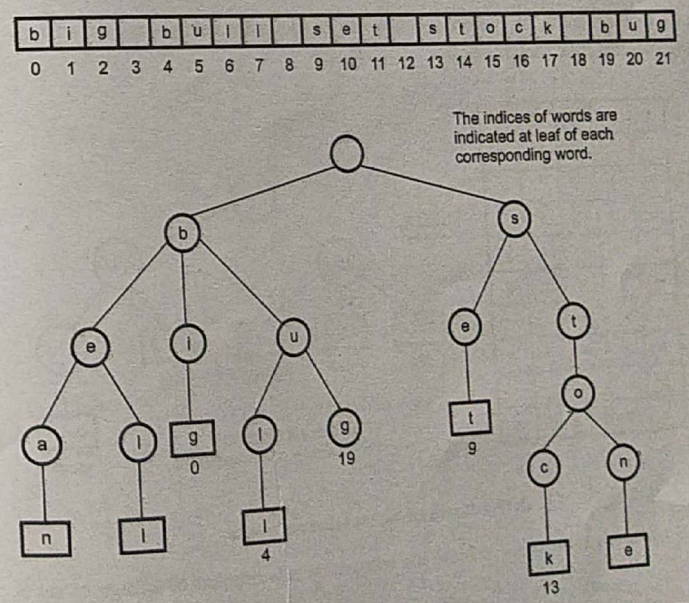


Fig. Q.16.2

Scanned by CamScanner

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

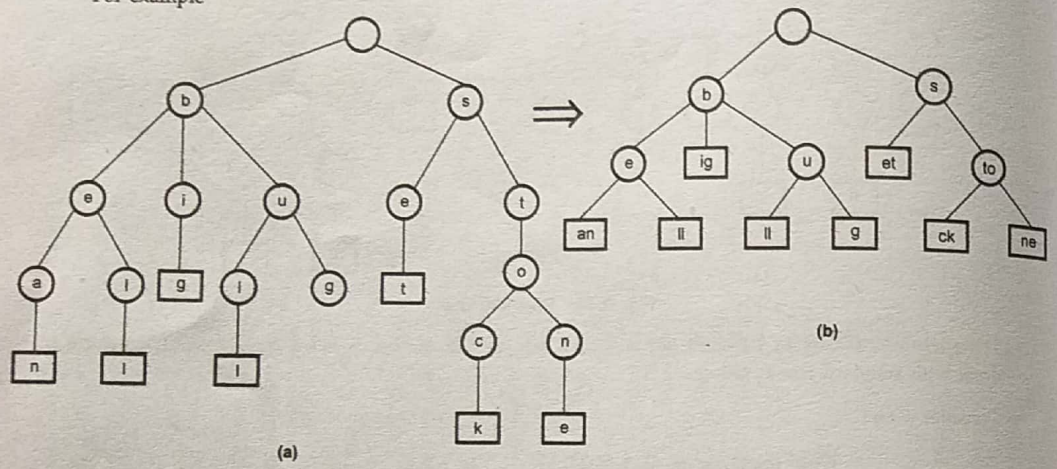
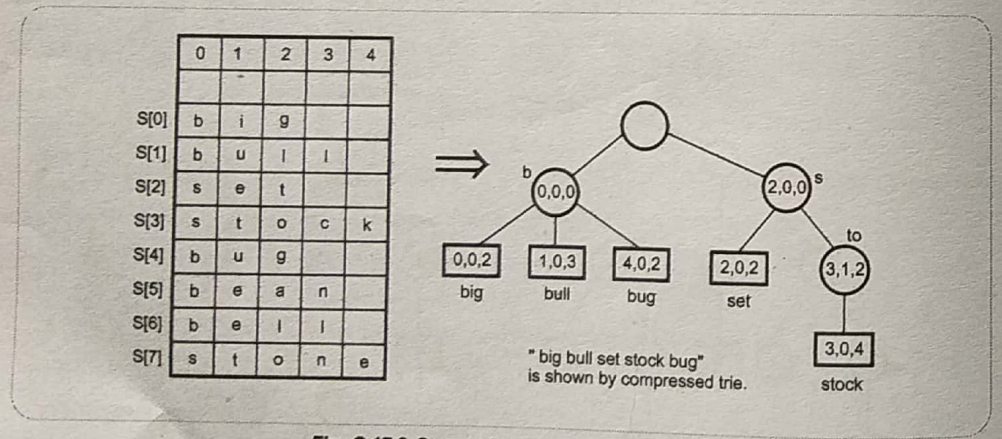


Fig. Q.17.1 Compressed tree obtained from standard trie

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

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.

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.

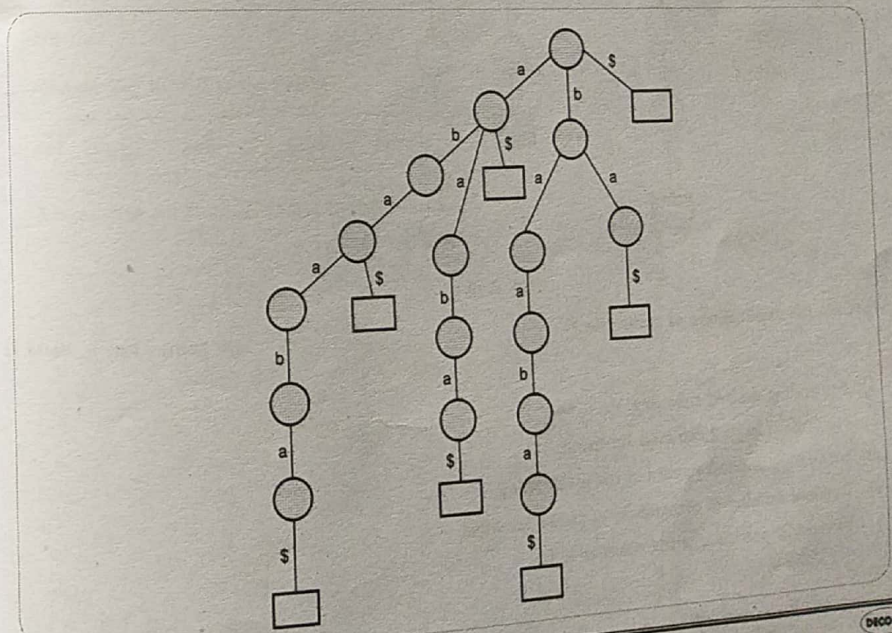
For example : Consider the string

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 :



Scanned by CamScanner

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.

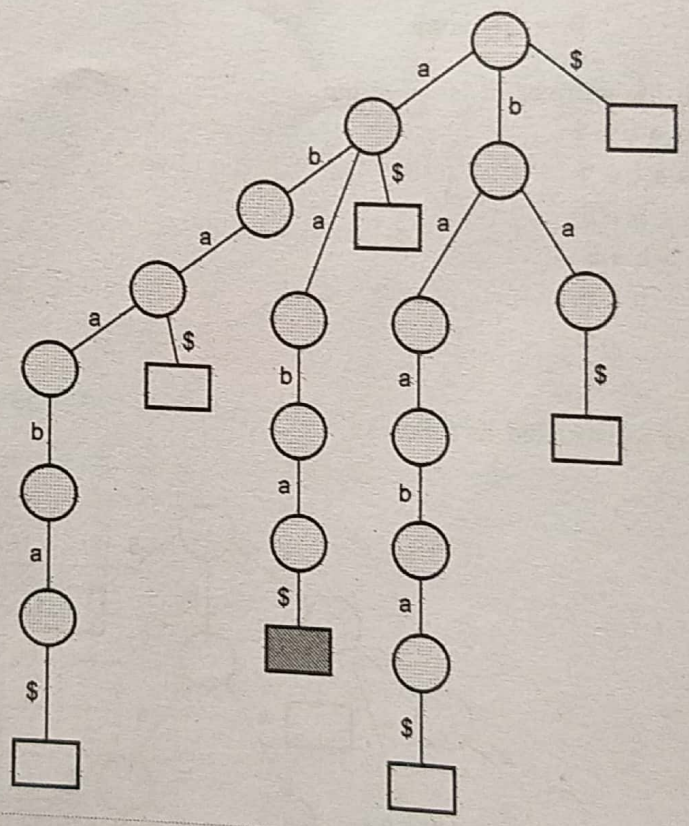


Fig. Q.19.1

Q.20 What are the applications of suffix trie ?

Ans. :

☞ [JNTU : Part A, Marks 2]

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.

Scanned by CamScanner

SOLVED MODEL QUESTION PAPER

Data Structures [As per R-18 Pattern]

IInd Year B. Tech., Sem - I [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,c as subquestions.

PART - A [25 Marks]

Q.1 **a)** *What is abstract data type ? What are all not concerned in an ADT ?* **(Refer Q.1 of chapter 1)** [2]

b) *What is the advantage of linked list over arrays ?* **(Refer Q.7 of chapter 1)** [3]

c) *Define and explain the concept of stack.* **(Refer Q.33 of chapter 1)** [2]

d) *What is dictionary ? Explain the method of representing dictionary.* **(Refer Q.1 of chapter 2)** [3]

e) *What is skip list.* **(Refer Q.4 of chapter 2)** [2]

f) *Define height of binary tree.* **(Refer Q.5 of chapter 3)** [3]

g) *How many trees are possible with 3 nodes.* **(Refer Q.14 of chapter 3)** [2]

h) *Distinguish between tree and graph.* **(Refer Q.2 of chapter 4)** [3]

i) *What are two types of sorting methods.* **(Refer Q.48 of chapter 4)** [2]

j) *What is the use of string matching algorithm ? Also mention the names of some string matching algorithms.* **(Refer Q.1 and Q.2 of chapter 5)** [3]

PART - B [50 Marks]

Q.2 **a)** *Write a pseudocode for creation of linked list.* **(Refer Q.11 of chapter 1)**

b) *Write functions and explain for push and pop operations of a stack ?* **(Refer Q.37 of chapter 1)** [5+5]

OR

Q.3 **a)** *Give postfix notation for the following infix expression A * (B + C/(D + E)) - F* (G/H + I).* **(Refer Q.54 of chapter 1)**

b) *Explain about the operations of queue with an example.* **(Refer Q.68 of chapter 1)** [5 + 5]

Scanned by CamScanner

Q.4 a) *Explain insertion of a node in skip list with some suitable example.* **(Refer Q.6 of chapter 2)**

b) *Explain various hashing methods in detail.* **(Refer Q.14 of chapter 2)** [5 + 5]

OR

Q.5 *What do you understand by collision in hashing ? Represent the following keys in memory using linear probing with or without replacement. Use modulo (10) as your hashing function : (24, 13, 16, 15, 19, 20, 22, 14, 17, 26, 84, 96).* **(Refer Q.26 of chapter 2)** [10]

Q.6 a) *Explain the terms - i) Leaves ii) Degree of node iii) Degree of tree iv) Level of tree v) Height of tree.* **(Refer Q.2 of chapter 3)**

b) *Explain tree traversals with an example.* **(Refer Q.10 of chapter 3)** [5 + 5]

OR

Q.7 a) *Create binary search tree for the following elements (23, 32, 24, 36, 15, 12, 39, 2, 19).* **(Refer Q.18 of chapter 3)**

b) *Write a C function for searching a node from a binary search tree.* **(Refer Q.20 of chapter 3)** [5 + 5]

Q.8 a) *Explain in detail about graph ADT.* **(Refer Q.5 of chapter 4)**

b) *What is graph ? Explain various representations of graphs.* **(Refer Q.8 of chapter 4)** [5 + 5]

OR

Q.9 *Explain depth first search method with suitable example.* **(Refer Q.15 and Q.16 of chapter 4)** [10]

Q.10 a) *What are the features of heap sort ?* **(Refer Q.46 of chapter 4)**

b) *Explain the concept of merge sort in detail.* **(Refer Q.51 of chapter 4)** [5 + 5]

OR

Q.11 a) *Explain Naïve string matching algorithm with example.* **(Refer Q.4 and Q.5 of chapter 5)**

b) *Find the Boyer-Moore string matching algorithm for the given pattern against the text. Text : abcaabccaabbabca Pattern: abc.* **(Refer Q.7 of chapter 5)** [5 + 5]

END...

Scanned by CamScanner