SREE NARAYANA GURUKULAM COLLEGE OF ENGINEERING

KADAYIRUPPU, KOLENCHERY 682311

LABORATORY RECORD

YEAR: 2021 TO 2022

NAME: SITHARA MOL K S

SEMESTER: 1 **ROLL NO:** 35

BRANCH: COMPUTER APPLICATIONS

Certified that this is a Bonafide Record of Practical work done in partial fulfillment of the requirements for the award of the Degree in Master of Computer Applications of Sree Narayana Gurukulam College of Engineering.

Kadayiruppu

Date:

Head of the Department Course Instructor

Submitted for University Practical Examination

Reg. No: SNG21MCA-2035 **on**----------

External Examiner Internal Examiner

INDEX PAGE

NO	PROGRAM	DATE
1	Merge two Sorted Array	26/11/2021
2	Circular Queue	29/11/2021
3	Stack using Linked List	30/11/2021
4	Doubly Linked List	03/12/2021
5	Binary Search Tree	21/12/2021
6	Set operations using BitString	04/01/2022
7	Disjoint Set	07/01/2022
8	Minimum Spanning Tree using Kruskal's algorithm	14/01/2022
9	Red Black Tree	18/01/2022
10	DFS Topological Sort	21/01/2022
11	Strongly Connected Components	25/01/2022
12	Minimum Spanning Tree using Prim's Algorithm	01/02/2022
13	Single Source Shortest Path	03/02/2022
14	Breadth First Search	04/02/2022

PROGRAM

```
#include<stdio.h>
void read(int*,int*,int,int);
void main()
{
int a[20],b[20],c[20],k=0,n1,n2;
printf("Enter the number of element n1:\n");
scanf("%d",&n1);
printf("Enter the number of element n2:\n");
scanf("%d",&n2);
read(a,b,n1,n2);
int i=0;
int j=0;
k=0;
while(i<n1 && j<n2)
{
if(a[i]<b[j])
{
c[k]=a[i];
i++;
}
else if(a[i]>b[i])
{
c[k]=b[j];
j++;
}
else
{
c[k]=a[i];
i++;
```

```
j++;
}
k++;
}
while(i<n1)
{
c[k]=a[i];
i++;
k++;
}
while(j<n2)
{
c[k]=b[j];
j++;
k++;
}
for(i=0;i<k;i++)
{
printf("%d ",c[i]);
}
}
void read(int *a,int *b,int n1,int n2)
{
        int i,j;;
printf("enter the element in 1st array\n");
for(i=0;i<n1;i++)
{
scanf("%d",&a[i]);
}
printf("enter the elements in 2nd array\n");
```

```
for(j=0;j<n2;j++)
{
scanf("%d",&b[j]);
}

}
```

OUTPUT

```
Enter the number of element n1:
3
Enter the number of element n2:
4
enter the element in 1st array
2
3
6
enter the elements in 2nd array
8
9
12
15
2 3 6 8 9 12 15
--------------------------------
Process exited after 49.22 seconds with return value 7
Press any key to continue . . .
```

..

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
int s=4;
int front=-1,rear=-1;
void insert(int *);
void del(int *);
void search(int *);
void display(int *);
void main()
{
int q[20], option;
do
{
printf("\n MENU\n 1.insert \n 2.delete \n 3.search \n 4.display \n 5.exit \n enter the option:");
scanf("%d",&option);
switch(option){
case 1:insert(q);
      break;
case 2:del(q);
      break;
case 3:search(q);
      break;
case 4:display(q);
     break;
}
}
while(option!=5);
}
void insert(int *q)
{
```

```
if(front==(rear+1)%s)
{
printf("queue is full \n");
return;
}
if(front==-1)
front=0;
rear=(rear+1)%s;
printf("enter the element \n");
scanf("%d",&q[rear]);
}
void del(int *q)
{
if(front==-1)
{
printf("queue empty\n");
return;
}
printf("deleted element %d \n",q[front]);
if(front==rear)
{
front=rear=-1;
}
else
{
front=(front+1)%s;
}
return;
}
void search(int *q)
```

```
{
int se,f;
printf("enter the element to be search");
scanf("%d",&se);
if(front==-1)
{printf("q is empty\n");
return;
}
f=front;
while(1)
{
if(se==q[f])
{
printf("element found");
break;
}
if(f==rear)
{
printf("element not found");
break;
}
f=(f+1)%s;
}
return;
}
void display(int *q)
{
int f;
if(front==-1)
{
```

```
printf("q  empty\n");
return;
}
f=front;
while(1)
{
printf("%d \n",q[f]);
if(f==rear)
break;
f=(f+1)%s;
}
return;
}
```

OUTPUT

```
 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:1
enter the element
34

 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:1
enter the element
56

 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:1
enter the element
67

 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:4
34
56
67

 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:3
enter the element to be search56
element found
 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:2
deleted element 34

 MENU
 1.insert
 2.delete
 3.search
 4.display
 5.exit
 enter the option:5


--------------------------------
Process exited after 27.26 seconds with return value 0
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();
void search();
struct node
{
int data;
struct node *next;
};
struct node *top=NULL;
void main()
{
int opt;
do
{
printf("\nMENU :\n1.push \n2. pop \n3. Display\n4.Search\n5.exit\nEnter the option:");
scanf("%d",&opt);
        switch(opt)
                {
        case 1:
                push();
                break;
        case 2:
                pop();
                break;
        case 3:
                display();
```

```
                break;
        case 4:
                search();
                break;
                }
}
while(opt!=5);
}
void push()
{
        struct node *ne;
        int x;
        printf("Enter the element to push:\n");
        scanf("%d",&x);
        ne=(struct node *)malloc(sizeof(struct node));
        if (ne == NULL)
                {
                printf("Stack Overflow!") ;
                return;
                }
        ne->data=x;
        ne->next= top;
        top = ne;
}
void pop()
{
struct node *ptr;
if (top == NULL)
        {
        printf("Stack Empty!") ;
```

```
        return;
        }
printf("%d  is popped\n",top->data);
ptr=top;
top=top->next;
free(ptr);
}
void display()
{
struct node *ptr;
if (top == NULL)
        {
        printf("Stack Empty:") ;
        return;
        }
else
        {
        ptr=top;
        while(ptr!=NULL)
                {
                printf("%d \t",ptr->data);
                ptr=ptr->next;
                }
        }
}
void search()
{
struct node *ptr;
int x;
printf("Enter the element to search:\n");
```

```
scanf("%d",&x);
if (top == NULL)
        {
        printf("Stack Empty") ;
        return;
        }
else
        {
        ptr=top;
        while(ptr!=NULL)
            {
             if(ptr->data==x)
              {
                  printf("Element Found:");
                  break;
              }
                  ptr=ptr->next;
            }
                  if(ptr==NULL)
{
printf("Element Not Found:");
}
        }

}
```

OUTPUT

```
MENU :
1.push
2. pop
3. Display
4.Search
5.exit
Enter the option:1
Enter the element to push:
67

MENU :
1.push
2. pop
3. Display
4.Search
5.exit
Enter the option:1
Enter the element to push:
89

MENU :
1.push
2. pop
3. Display
4.Search
5.exit
Enter the option:1
Enter the element to push:
23

MENU :
1.push
2. pop
3. Display
4.Search
5.exit
Enter the option:3
23      89      67
MENU :
1.push
2. pop
3. Display
4.Search
5.exit
Enter the option:4
Enter the element to search:
89
Element Found:
```

```
MENU
1.insert
2.delete
3.search
4.display
5.exit
enter the option:2
deleted element 34

MENU
1.insert
2.delete
3.search
4.display
5.exit
enter the option:5

--------------------------------
Process exited after 27.26 seconds with return value 0
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdlib.h>
#include<stdio.h>
struct node
  {
   int data,q;
   struct node*left;
   struct node*right;
  };
 struct node*head=NULL;
void insertf(int);
void insertl(int);
void display();
void deletef();
void deletel();
void insertpos();
void deletepos();
void search();
  void main()
 {
   int opt,q;
 do
  {
    printf("\n Menu:\n");

printf("1.Insertfirst\n2.Insertlast\n3.Display\n4.Deletefirst\n5.Deletelast\n6.Insertpos\n7.Delet
epos\n8.Search\n9.Exit\nEnter the choice:");
    scanf("%d",&opt);
    switch(opt)
     {
      case 1:insertf(q);
```

```
            break;
       case 2:insertl(q);
            break;
      case 3:display();
            break;
       case 4:deletef();
            break;
       case 5:deletel();
            break;
       case 6:insertpos();
            break;
       case 7:deletepos();
            break;
        case 8:search();
            break;

     }
  }
     while(opt!=9);
}
void insertf(int q)
 {
  struct node *n;
  n=(struct node*)malloc(sizeof(struct node));
  if(n==NULL)
  {
   printf("Insufficient memory");
  }
  printf("Enter the element to insert at FIRST\n");
  scanf("%d",&q);
```

```
  n->data=q;
  n->left=NULL;
  n->right=NULL;
  if(head==NULL)
   {
     head=n;
   }
  else
  {
    n->right=head;
    head->left=n;
    head=n;
  }
}
void insertl(int q)
 {
  struct node *n,*ptr;
  n=(struct node*)malloc(sizeof(struct node));
   if(n==NULL)
  {
   printf("Insufficient memory");
  }
  printf("Enter the element to insert at LAST\n");
  scanf("%d",&q);
  n->data=q;
  n->right=NULL;
  n->left=NULL;
  if(head==NULL)
   {
    head=n;
```

```
   }
  else
  {
   ptr=head;
   while(ptr->right!=NULL)
    {
     ptr=ptr->right;
    }
    ptr->right=n;
    n->left=ptr;
  }
}
void insertpos(int q)
{
int key;
 struct node *n,*ptr,*ptr1;
  n=(struct node*)malloc(sizeof(struct node));
   if(n==NULL)
   {
    printf("Insufficient memory");
   }
   printf("Enter the element to insert at POSITION\n");
   scanf("%d",&q);
   n->data=q;
   n->left=NULL;
   n->right=NULL;

 if(head==NULL)
 {
   head=n;
```

```
 }
 printf("enter a key  value");
 scanf("%d",&key);
 ptr=head;
 while(ptr->right!=NULL&&ptr->data!=key)
 {
  ptr=ptr->right;
 }
 if(ptr->right==NULL)
 {
          ptr->right=n;
          n->left=ptr;
          printf("Element inserted at LAST POSITION \n");
 }
 else
  {
    ptr1=ptr->right;
    n->right=ptr1;
    n->left=ptr;
    ptr->right=n;
    ptr1->left=n;
  }
  printf("Element inserted\n");
  display();
}
void search()
{
          struct node *ptr;
          int x;
          printf("Enter the element to SEARCH\n");
```

```
        scanf("%d",&x);
        if (head == NULL)
                {
                printf("LInked List is Empty") ;
                return;
                }
        else
    {
        ptr=head;
        while(ptr!=NULL)
     {
         if(ptr->data==x)
      {
        printf("Element Found%d",ptr->data);
        printf("\n");
        break;
       }
        ptr=ptr->right;
      }
        if(ptr==NULL)
        {
         printf("Element Not Found");
         printf("\n");
       }
    }
}
void deletepos()
{
        struct node *ptr;
        struct node *prev;
```

```
        struct node *next;
        int x;
        if (head == NULL)
        {
                printf("Linked List is Empty") ;

        }
        printf("Enter the position where you want to DELETE item: ");
        scanf("%d", &x);
if(head->data==x)
        {
                ptr = head;
                head = head->right;
                if (head != NULL)
                {
                        head->left=NULL;
                }
                free(ptr);
        }
else
        {
        ptr = head;
        while(ptr->data != x && ptr->right!=NULL)
        {
                ptr=ptr->right;
        }
        if(ptr!=NULL)
        {
                prev=ptr->left;
                next=ptr->right;
```

```
                prev->right=ptr->right;
        }

        if(prev->right!=NULL)
        {
                next->left=ptr->left;
                free(ptr);
        }
        }
}
void display()
 {
  struct node *ptr;
 if(head==NULL)
  {
   printf("list empty");
  }
 else
   ptr=head;
   while(ptr!=NULL)
    {
     printf("%d\n",ptr->data);
     ptr=ptr->right;
    }
 }
 void deletef()
 {
  struct node *ptr;
  if(head==NULL)
   {
```

```
     printf("lIST IS EMPTY");
    }
    else
    {
     ptr=head;
     head=head->right;
    }
     if(head!=NULL)
     {
      head->left=NULL;
     }
     printf("deleted data is:%d\n",ptr->data);
     free(ptr);
   }
void deletel()
{
struct node *ptr;
struct node *prev;
if (head == NULL)
{
 printf("Linked List is Empty") ;
 return;
}
if (head->right == NULL)
{
 free(head);
 head = NULL;
}
ptr = head;
while(ptr->right!=NULL)
```

```
{
 ptr=ptr->right;
}
prev=ptr->left;
prev->right=NULL;
printf("deleted data is %d",ptr->data);
free(ptr);
}
```

OUTPUT

```
Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:1

Enter the data to insert
56

Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:1

Enter the data to insert
78

Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:2

Enter the data to insert
23

Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:2

Enter the data to insert
11

Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:4
List:78 56      23      11
Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:7

Enter the data to insert
15

Enter the key value
1

Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
```

```
9.Exit
Select your option:4
List:78 56      23      11      15
Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:8

Enter the data:
23

Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:4
List:78 56      11      15
Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:3
Enter the element to search
11

Element found:
Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
```

```
5.DeleteFirst
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:9
Exited
--------------------------------
Process exited after 65.93 seconds with return value 9
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
struct node *left;
int data;
struct node *right;
};
void insert();
void search();
void inorder(struct node *);
void preorder(struct node *);
void postorder(struct node *);
void delet(int);
struct node *root=NULL;
int main()
{
int opt,x;
do
{
 printf("\nMenu");
 printf("\n1.Insertion\n2.Inorder\n3.Preorder\n4.Postorder\n5.Search\n6.Deletion\n7.Exit");
 printf("\nSelect your option:");
 scanf("%d",&opt);
 switch(opt)
 {
 case 1:
            insert();
            break;
```

```
 case 2:
            inorder(root);
            break;
 case 3:
            preorder(root);
            break;
 case 4:
            postorder(root);
            break;
 case 5:
            search();
            break;
 case 6:
           printf("\nEnter the element to delete:\n");
           scanf("%d",&x);
            delet(x);
            break;
 default:
            printf("Exited\n");
 }
}while(opt!=7);
}
void insert()
{
int x;
struct node *ne,*ptr,*ptr1;
ne=(struct node *)malloc(sizeof(struct node));
if(ne==NULL)
{
printf("Insufficient Memory");
```

```
return;
}
printf("Enter the data to insert:");
scanf("%d",&x);
ne->left=NULL;
ne->right=NULL;
ne->data=x;
if(root==NULL)
{
root=ne;
return;
}
ptr=root;
while(ptr!=NULL)
{
if(x==ptr->data)
{
printf("Item already exist\n");
return;
}
if(x>ptr->data)
{
ptr1=ptr;
ptr=ptr->right;
}
else
{
ptr1=ptr;
ptr=ptr->left;
}
```

```
}
if(ptr==NULL)
{
if(x>ptr1->data)
ptr1->right=ne;
else
ptr1->left=ne;
}
}
void inorder(struct node * ptr)
{
if(ptr!=NULL)
{
inorder(ptr->left);
printf("%d ",ptr->data);

inorder(ptr->right);
}
}
void preorder(struct node * ptr)
{
        if(ptr!=NULL)
        {
                printf("%d ",ptr->data);
                preorder(ptr->left);
                preorder(ptr->right);
        }
}
void postorder(struct node * ptr)
{
```

```
        if(ptr!=NULL)
        {
                postorder(ptr->left);
                postorder(ptr->right);
                printf("%d ",ptr->data);
        }
}
void search()
{
struct node *ptr;
int x;
ptr=root;
printf("Enter the data to search:");
scanf("%d",&x);
while(ptr!=NULL)
{
if(ptr->data==x)
{
        printf("Data present\n");
        return;
}
if(x>ptr->data)
ptr=ptr->right;
else
ptr=ptr->left;
}
if(ptr==NULL)
printf("Data not present\n");
}
void delet(int x)
```

```
{
struct node *ptr,*parent,*p;
int dat;
if(root==NULL)
{
printf("Tree is empty");
return;
}
parent=NULL;
ptr=root;
while(ptr!=NULL)
        {
                if(ptr->data==x)
                break;
                parent=ptr;
                if(x>ptr->data)
                ptr=ptr->right;
                else
                ptr=ptr->left;
}
if(ptr==NULL)
{
printf("Item not present");
return;
}
if(ptr->right==NULL && ptr->left==NULL)
{
if(parent==NULL)
root=NULL;
else
```

```
        if(parent->right==ptr)
            parent->right=NULL;
else
parent->left=NULL;
printf("Element deleted");
free(ptr);
return;
}
if(ptr->right!=NULL && ptr->left!=NULL)
{
p=ptr->right;
while(p->left!=NULL)
{
p=p->left;
}
dat=p->data;
delet(p->data);
ptr->data=dat;
return;
}
if(parent==NULL)
{
if(ptr->right==NULL)
root=ptr->left;
else
root=ptr->right;
}
else
{
if(parent->right==ptr)
```

```
{
if(ptr->right==NULL)
parent->right=ptr->left;
else
parent->right=ptr->right;
}
else
{
if(ptr->left==NULL)
parent->left=ptr->right;
else
parent->left=ptr->left;
}
}
printf("\nElement deleted");
free(ptr);
return;
}
```

OUTPUT

```
Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:56

Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:23

Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:11

Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:26

Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:2
11 23 26 56
Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:3
56 23 11 26
Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:4
11 26 23 56
Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:5
Enter the data to search:23
Data present

Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:6

Enter the element to delete:
26
Element deleted
```

```
Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:2
11 23 56
Menu
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:7
Exited

--------------------------------
Process exited after 55.11 seconds with return value 7
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
void setunion(char *,char *,char *);
void setintersection(char *,char *,char *);
void setdifference(char *,char *,char *);

void main()
{
        char s1[20],s2[20],s3[20];
        printf("Enter set1:\n");
        scanf("%s",s1);
        printf("Enter set2:\n");
        scanf("%s",s2);
        //check whether the two strings are equal or not
        setunion(s1,s2,s3);
        printf("\nunion\n%s",s3);
        setintersection(s1,s2,s3);
        printf("\nintersection\n%s",s3);
        setdifference(s1,s2,s3);
        printf("\nsetdifference\n%s",s3);
}

void setunion(char *s1,char *s2,char *s3)
{
 int i,l=strlen(s1);
for(i=0;i<l;i++)
{
 if(s1[i]=='0'&& s2[i]=='0')
```

```
s3[i]='0';
else
s3[i]='1';
}
s3[i]='\0';
}

void setintersection(char *s1,char *s2,char *s3)
{
int i,l=strlen(s1);
for(i=0;i<l;i++)
{
if(s1[i]=='1'&& s2[i]=='1')
s3[i]='1';
else
s3[i]='0';
}
s3[i]='\0';
}

void setdifference(char *s1,char *s2,char *s3)
{ int i,l=strlen(s1);
for(i=0;i<l;i++)
{ if(s1[i]=='1'&&  s2[i]=='0')
s3[i]='1';
else
s3[i]='0';
}
s3[i]='\0';
}
```

OUTPUT

```
Enter set1:
1011011101
Enter set2:
1000111000

union
1011111101
intersection
1000011000
setdifference
0011000101
--------------------------------
Process exited after 13.81 seconds with return value 25
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
 struct node
{
struct node *next;
int data;
};
void makeset();
void display();
int find(int);
void unions();
struct node *first[30];
int n;
void main()
{
        int i,ch,x,y;
        struct node *ne;
        printf("Enter number :");
        scanf("%d",&n);
for(i=0;i<n;i++)
        {
        ne=(struct node *)malloc(sizeof(struct node));
        ne->next=NULL;
        ne->data=i+1;
        first[i]=ne;
        }
display();
```

```
        do
        {
        printf("Choose operation :\n1.Display\n2.Union\n3.Find\n4.Makeset\n5.Exit \n");
        scanf("%d",&ch);
                switch(ch)
                        {
                    case 1:display();
                            break;
                    case 2:unions();
                           break;
                     case 3:printf("Enter the element to Find\n");
                            scanf("%d",&x);
                            i=find(x);\
                            if(i==-1)
                            printf("No such element\n");
                            else
                            printf("Element PRESENT in set-> %d\n",first[i]->data);
                            break;
                   case 4:makeset();
                           break;
                     }
        }
        while(ch!=5);
}
void makeset()
        {
        int pos,x;
        pos=find(x);
        printf("Enter the number :");
        scanf("%d",&x);
```

```
        pos=find(x);
         if (pos==-1)
                {
                 first[n]=(struct node *)malloc(sizeof(struct node));
                 first[n]->data=x;
                 first[n]->next=NULL;
                 n++;
                }
        else
                printf("\n The no: exits in another set");
        }
void display()
        {
        struct node *p;
        int i;
        printf("The Sets are :");
        for(i=0;i<n;i++)
        {
                p=first[i];
                if (p==NULL)
                 continue;
                printf("{");
                while(p!=NULL)
                {
                printf("%d ",p->data);
                p=p->next;
                }
            printf("}\n");
          }
        }
```

```
void unions()
{
int a,b,i,j;
struct node *p;
printf("\nEnter the first element:");
scanf("%d",&a);
printf("\nEnter the second element:");
scanf("%d",&b);
i=find(a);
j=find(b);
if (i==-1 || j ==-1)
{
printf("element not found");
return;
}
if (i==j)
printf("Both are in the same set");
else
{
p=first[i];
while(p->next!=NULL)
p=p->next;
p->next=first[j];
first[j]=NULL;
}
}
int find(int x)
{
struct node *p;
int i,j,flag;
```

```
flag=0;
  for(i=0;i<n;i++)
          {
          p=first[i];
              while(p!=NULL)
                   {
                 if (p->data==x)
                   {
                            flag=1;   break;
                    }
                   p=p->next;
                   }
              if (flag==1)
                   break;
          }
if (flag==1)
      return i;
   else
       return -1;
```

OUTPUT

```
Enter number :4
The Sets are :{1 }
{2 }
{3 }
{4 }
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
4
Enter the number :7
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
1
The Sets are :{1 }
{2 }
{3 }
{4 }
{7 }
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
2

Enter the first element:3

Enter the second element:7
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
1
The Sets are :{1 }
{2 }
{3 7 }
{4 }
Choose operation :
1.Display
2.Union
```

```
3.Find
4.Makeset
5.Exit
3
Enter the element to Find
7
Element PRESENT in set-> 3
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
5


--------------------------------
Process exited after 32.24 seconds with return value 5
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdlib.h>
#include<stdio.h>
struct node
{
int data;
struct node *next;
};
struct edge {
int start;
int weight;
int end;
};
void makeset(int x);
void unionset(int a,int b);
int find(int);
int n=0;
struct node *first[20];
struct edge adj[20],a[20];
void main()
{int v,e,c=-1,s,count=0,i,start,end,weight,k,v1,u,w;
printf("Enter the no of vertices:");
scanf("%d",&v);
for(i=1;i<=v;i++)
{
makeset(i);
}
printf("\nEnter the no of edges:");
scanf("%d",&e);
printf("\nEnter the edges:");
```

```
printf("\nStart\tend\tweight:\n");
for(i=0;i<e;i++)
{scanf("%d%d%d",&start,&end,&weight);
for(k=c;k>=0;k--)
if(adj[k].weight>weight)
adj[k+1]=adj[k];
else
break;
adj[k+1].start=start;
adj[k+1].end=end;
adj[k+1].weight=weight;
c++;
}
count=0;
for(i=0;i<c;i++)
{
u=adj[i].start;
v1=adj[i].end;
w=adj[i].weight;
if(find(u)!=find(v1))
{
a[count].start=u;
a[count].end=v1;
a[count].weight=w;
count++;
unionset(u,v1);
}
}
printf("Spanning tree edges:\n");
s=0;
```

```
for(i=0;i<count;i++)
{
printf("%d->%d\tw-%d\n",a[i].start,a[i].end,a[i].weight);
s=s+a[i].weight;
}
printf("\nTotal cost=%d",s);
}
void makeset(int x)
{
int pos;
pos=find(x);

if (pos==-1)
{
first[n]=(struct node *)malloc(sizeof(struct node *));
first[n]->data=x;
first[n]->next=NULL;
n++;
}
else
printf("Element already exist.\n");
}
int find(int x)
{
int i,flag=0;
struct node *p;
for(i=0;i<n;i++)
{
p=first[i];
while(p!=NULL)
```

```
{if(p->data==x)
{
flag=1;
break;
}
p=p->next;
}
if (flag==1)
break;
}
if(flag==1)
return i;
else
return -1;
}
void unionset(int a,int b){
int i,j;
struct node *p;
i=find(a);
j=find(b);
if (i==-1 || j ==-1)
{printf("Element not founded.\n");
return;
}
if (i==j)
printf("Both are in the same set.\n");
else
{
p=first[i];
while(p->next!=NULL)
```

```
p=p->next;
p->next=first[j];
first[j]=NULL;
}
}
```

OUTPUT

```
Enter the no of vertices:5

Enter the no of edges:7

Enter the edges:
Start   end     weight:
1 2 1
1 4 10
1 3 7
1 5 5
2 3 3
3 4 4
4 5 2
Spanning tree edges:
1->2    w-1
4->5    w-2
2->3    w-3
3->4    w-4

Total cost=10
--------------------------------
Process exited after 69.67 seconds with return value 14
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#define red  1
#define black 0
struct node
{   int data,color;
    struct node *right,*left;
} ;
void doop(struct node *,struct node *,struct node *);
void RRRotation(struct node *);
void LLRotation(struct node *);
struct node *ROOT=NULL;
struct node* findParent(struct node *n)  ;
//function to reserve memory for a node
struct node * getNode()
{
    struct node *ne;
    ne=(struct node *) malloc(sizeof(struct node));
    if (ne==NULL)
       printf("No Memory");
    return ne;
}
//function for inorder traversal
void inorder(struct node *ptr)
{  if (ptr!=NULL)
   {   inorder(ptr->left);
       printf("%d(%c)  ",ptr->data,ptr->color==0?'b':'r');
       inorder(ptr->right);
    }
```

```
}
//function to find the parent node of a node
struct node* findParent(struct node *n)
{   struct node *ptr=ROOT,*parent=NULL;
     int x=n->data;
   while(ptr!=n)
       {   parent=ptr;
            if (x>ptr->data)
                ptr=ptr->right;
             else
               ptr=ptr->left;
           }
    return parent;
}
//function to insert a value in the Binary search tree
void insert()
{  int x;
   struct node *ne,*parent,*ptr,*pparent,*uncle;
//Perform standard BST insertion and make the colour of newly inserted nodes as RED.
   printf("Enter the element to insert:");
   scanf("%d",&x);
   ne=getNode();
   if (ne==NULL)
      return;
   ne->data=x;
   ne->left=ne->right=NULL;
   ne->color=red;
//If x is the root, change the colour of x as BLACK and return
   if (ROOT==NULL)
     { ROOT=ne;
```

```
        ne->color=black;
        return;
  }
 ptr=ROOT;
 while(ptr!=NULL)
 {   if (ptr->data==x)
        {   printf("Data already present");
            break;
        }
        parent=ptr;
        if (x>ptr->data)
           ptr=ptr->right;
        else
          ptr=ptr->left;
 }
 if (ptr!=NULL)
     return;
if(x>parent->data)
   parent->right=ne;
else
  parent->left=ne;
while(ne!=ROOT)
{      //find uncle
        parent=findParent(ne);
        if (parent->color==black)
            break;
        if (parent->color==red)
        {
          pparent=findParent(parent);
        if (pparent->right==parent)
```

```
            uncle=pparent->left;
          else
            uncle=pparent->right;
//If x's uncle is BLACK, or NULL then call doop()

      if (uncle==NULL)
          {       doop(ne,parent,pparent);
                  break;
          }
      if (uncle->color==black )
           {      doop(ne,parent,pparent);
                  break;
          }
  /* If x's uncle is RED (Grandparent must have been black from property 4)
(1)Change the colour of parent and uncle as BLACK.
(ii) Colour of a grandparent as RED.
(iii) Change x = x's grandparent, repeat steps 2 and 3 for new x. */

      if (uncle->color==red)
          {       parent->color=uncle->color=black;
                  if (pparent!=ROOT)
                  {  if (pparent->color==red)
                          pparent->color=black;
                    else
                          pparent->color=red;
                    if(pparent->color==red)
                           ne=pparent;
                  }
                  else
                                break;
```

```
            }
}
 }
  }
void doop(struct node *ne,struct node *parent,struct node *pparent)
{       /*(i) Left Left Case (p is left child of g and x is left child of p)
        (ii) Left Right Case (p is left child of g and x is the right child of p)
        (iii) Right Right Case (Mirror of case i)
        (iv) Right Left Case (Mirror of case ii)*/

        if(ne==parent->left && parent==pparent->left)
        {    struct node *left=pparent->left;
             LLRotation(pparent);
             parent->color=parent->color==1?0:1;
             pparent->color=pparent->color==1?0:1;
                  if (pparent==ROOT)
                    ROOT=left;
        }
          else if (parent==pparent->left && ne==parent->right)
           {
              struct node *left=parent->right;
              RRRotation(parent);
            LLRotation(pparent);
           ne->color=ne->color==1?0:1   ;
           pparent->color=pparent->color==1?0:1;
                  if (pparent==ROOT)
                    ROOT=left;

           }
          else if ( ne==parent->right && parent==pparent->right)
```

```
        {     struct node *right=pparent->right;
              RRRotation(pparent);
              parent->color=parent->color==0?1:0;
              pparent->color=pparent->color==0?1:0;
              if (pparent==ROOT)
                ROOT=right;
              }
          else if (parent==pparent->right && ne==parent->left)
              {  struct node *left=parent->left;
                 LLRotation(parent);
                RRRotation(pparent);
                pparent->color=pparent->color==1?0:1;
                ne->color=ne->color==1?0:1;
              if (pparent==ROOT)
                ROOT=left;
                    }
}
void LLRotation(struct node *y) // function for Right Rotation
{    struct node *p=findParent(y);
     struct node *x=y->left;
     struct node *T2= x->right;
     if (x!=NULL)
        x->right=y;
        y->left=T2;
        if (p!=NULL)
        if (p->right==y)
          p->right=x;
        else
         p->left=x;
```

```
 }
void RRRotation(struct node *x)  // function for left rotation
{ struct node *p=findParent(x);
struct node *y=x->right;
struct node *T2=y->left;
if (y!=NULL)
y->left=x;
x->right=T2;
if (p!=NULL)
if (p->right==x)
  p->right=y;
else
  p->left=y;
}
void main()
{
int ch;
do{
    printf("\nMENU\n1.Insert\n2.display\n3.Exit\nEnter Your choice:");
    scanf("%d",&ch);
    switch(ch)
    {  case 1:insert();
                break;
       case 2:inorder(ROOT);
                    break;
    }
  }while(ch!=3);
}
```

OUTPUT

```
MENU
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:56

MENU
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:32

MENU
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:67

MENU
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:23

MENU
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:11

MENU
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:10

MENU
```

```
1.Insert
2.display
3.Exit
Enter Your choice:
1
Enter the element to insert:11
Data already present
MENU
1.Insert
2.display
3.Exit
Enter Your choice:
2
10(r)  11(b)  23(r)  32(b)  56(b)  67(b)
MENU
1.Insert
2.display
3.Exit
Enter Your choice:
3

--------------------------------
Process exited after 111.5 seconds with return
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
int vertex;
struct node *next;
}*adj[20];
int v,e;
int visited[20],top[20];
int t=0;
void dfs();
void dfsvisit();
void main()
{
        int s,i,en;
        struct node *ne;
        printf("Enter no of vertices");
        scanf("%d",&v);
        for(i=0;i<=v;i++)
                adj[i]=NULL;
        printf("Enter no of edjes");
        scanf("%d",&e);
        printf("Enter edges");
        printf("\nstart End\n");
        for(i=0;i<e;i++)
        {
                scanf("%d%d",&s,&en);
                ne=(struct node*)malloc(sizeof(struct node));
```

```
                ne->vertex=en;
                ne->next=adj[s];
                adj[s]=ne;
        }
        dfs();
        printf("\nTopological sort order\n");
        for(i=t-1;i>=0;i--)
        printf("%d ",top[i]);
        getch();
}
void dfs()
{
        int i;
        for(i=0;i<=v;i++)
                visited[i]=0;
        printf("\nDFS\n");
        for(i=1;i<=v;i++)
                if(visited[i]==0)
                        dfsvisit(i);
}

void dfsvisit(int u)
{
        int w;
        struct node *ptr;
        visited[u]=1;
        printf("%d ",u);
        ptr=adj[u];
        while(ptr!=NULL)
        {
```

```
            w=ptr->vertex;
            if(visited[w]==0)
                dfsvisit(w);
            ptr=ptr->next;
        }
        top[t]=u;
        t++;
}
```

OUTPUT

```
Enter no of vertices5
Enter no of edjes5
Enter edges
start End
0 1
0 2
0 3
1 2
2 4

DFS
1 2 4 3 5
Topological sort order
5 3 1 2 4
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
        int vertex;
        struct node *next;
};
int v,e;
struct node *adj[20], *adj1[20];
int t=0,visited[20],ft[20];
void dfs();
void dfsvisit(int);
void dfs1();
void dfsvisit1(int);
void adjlistrep(struct node **adj,int s,int en)
{
        struct node *ne=(struct node*)malloc(sizeof(struct node));
        ne->vertex=en;
        ne->next=adj[s];
        adj[s]=ne;
}

void main()
{
        int s,i,en;
        struct node *ptr;
        printf("Enter no. of vertices");
        scanf("%d",&v);
```

```
	for(i=0;i<=v;i++)
		adj[i]=adj1[i]=NULL;
	printf("Enter no. of edges:");
	scanf("%d",&e);
	printf("Enter the edges\n");
	printf("Start End\n");
	for(i=0;i<e;i++)
	{
		scanf("%d%d",&s,&en);
		adjlistrep(adj,s,en);
		adjlistrep(adj1,en,s);
	}
	dfs();
	dfs1();
	getch();
}

void dfs()
{
	int i;
	for(i=0;i<=v;i++)
		visited[i]=0;
	printf("\nDFS\n");
	for(i=1;i<=v;i++)
	{
		if(visited[i]==0)
		{
			dfsvisit(i);
		}
	}
```

```
}

void dfsvisit(int u)
{
        int w;
        struct node *ptr;
        visited[u]=1;
        printf("%d ",u);
        ptr=adj[u];
        while(ptr!=NULL)
        {
                w=ptr->vertex;
                if(visited[w]==0)
                        dfsvisit(w);
                ptr=ptr->next;
        }
        t++;
        ft[u]=t;
}

void dfs1()
{
        int i,max=0,ver;
        printf("\n components\n");
        for(i=0;i<=v;i++)
                visited[i]=0;
        while(1)
        {
                max=0;
                for(i=1;i<=v;i++)
```

```
                {
                        if(visited[i]==0 && ft[i]>max)
                        {
                                ver=i;
                                max=ft[i];
                        }
                }
                if(max==0)
                        break;
                printf("{");
                dfsvisit1(ver);
                printf("}\n");
        }
}
void dfsvisit1(int u)
{
        int w;
        struct node *ptr;
        visited[u]=1;
        printf("%d ",u);
        ptr=adj1[u];
        while(ptr!=NULL)
        {
                w=ptr->vertex;
                if(visited[w]==0)
                        dfsvisit1(w);
                ptr=ptr->next;
        }
}
```

OUTPUT

```
Enter no. of vertices5
Enter no. of edges:5
Enter the edges
Start End
0 1
0 2
0 3
1 2
2 4

DFS
1 2 4 3 5
 components
{5 }
{3 0 }
{1 }
{2 }
{4 }
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define inf 999
void addtoadjlist(int s,int en,int w);
int emptyQ();
int extractminQ();
struct node
{
        int vertex;
   int weight;
   struct node *next;
}*adj[20];
int v;
int p[20],key[20],q[20];

int main()
{
        int i,s,en,we,e,u,w,sum=0;
   struct node *ptr;
   printf("Enter No: of vertices:");
   scanf("%d",&v);
   for(i=1;i<=v;i++)
   {
                p[i]=0;
        key[i]=inf;
        q[i]=1;
        adj[i]=NULL;
   }
```

```
printf("No: of edges: ");
scanf("%d",&e);
printf("Enter the adges\n");
printf("start end weight");
for(i=1;i<=e;i++)
{
        scanf("%d%d%d",&s,&en,&we);
    addtoadjlist(s,en,we);
    addtoadjlist(en,s,we);
    }
    key[1]=0;
    while(!emptyQ())
    {
        u=extractminQ();
    ptr=adj[u];
    while(ptr!=NULL)
    {
            w=ptr->vertex;
            if (q[w]==1 && ptr->weight < key[w])
            {
                key[w]=ptr->weight;
            p[w]=u;
            }
        ptr=ptr->next;
    }
    }
    sum=0;
    printf("Spanning tree edges\n");
    for(i=2;i<=v;i++)
    {
```

```
                printf("(%d-%d) w:%d \n",i,p[i],key[i]);
        sum=sum+key[i];
        }
        printf("The total cost is %d",sum);
        getch();
}
int emptyQ()
{
        int i,flag=1;
        for(i=1;i<=v;i++)
        {
                if (q[i]==1)
        {
                    flag=0;
                        break;
        }
        }
        return flag;
}
int extractminQ()
{
        int i,min=inf,ver;
        for(i=1;i<=v;i++)
        {
                if (key[i]<min && q[i]==1)
        {
                        ver=i;
                        min=key[i];
        }
    }
```

```
        q[ver]=0;
   return ver;
}
void addtoadjlist(int s,int en,int w)
{
      struct node *ne=(struct node *)malloc(sizeof(struct node));
      ne->vertex=en;
      ne->weight=w;
      ne->next=adj[s];
      adj[s]=ne;
 }
```

OUTPUT

```
Enter No: of vertices:5
No: of edges: 7
Enter the adges
start end weight
1 2 1
1 3 7
1 4 10
1 5 5
2 3 3
3 4 4
4 5 2
Spanning tree edges
(2-1) w:1
(3-2) w:3
(4-3) w:4
(5-4) w:2
The total cost is 10
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define inf 999
void printpath(int,int);
int extractmin();
int v,adj[20][20],dist[20],visit[20],pred[20];
void main()
{
int e,st,en,w,i,j,src,ver,k;
//clrscr();
printf("Enter the no: of vertices");
scanf("%d",&v);
printf("Enter the no: of edges");
scanf("%d",&e);
        for(i=0;i<=v;i++)
        {
                for(j=0;j<=v;j++)
                        adj[i][j]=inf;
                        dist[i]=inf;
                        visit[i]=0;
        }
        printf("Enter the edges\n");
        printf("start end weight\n");
        for(i=1;i<=e;i++)
        {
                scanf("%d%d%d",&st,&en,&w);
                adj[st][en]=w;
        }
```

```
printf("Enter the starting vertex");
scanf("%d",&src);
dist[src]=0;
pred[src]=src;
for(k=1;k<=v;k++)
{
        ver=extractmin();
        visit[ver]=1;
   if (dist[ver]==inf) continue;
   for(i=1;i<=v;i++)
   {
        if (adj[ver][i]!=inf&& visit[i]==0 )
                        if (dist[i]>dist[ver]+adj[ver][i])
                        {
                          dist[i]=dist[ver]+adj[ver][i] ;
                    pred[i]=ver;
                        }
        }
}
for(i=1;i<=v;i++)
{
        if (dist[i]==inf)
                continue;
   printf("path cost to %d= %d   ",i,dist[i]);
   if( dist[i]!=inf)
   {
          printpath(i,src);
          printf("->%d",i);
          printf("\n");
   }
```

```
        }
   getch();
}
void printpath(int i,int src)
{
        if (pred[i]==src)
   {
                printf("%d ",src);return;
   }
   printpath(pred[i],src);
   printf("->%d  ",pred[i]);
}
int extractmin()
{
        int min=inf,i,ver;
   for(i=1;i<=v;i++)
   {
                if (visit[i]==0 && dist[i]<min)
                {
                        min=dist[i];
                        ver=i;
                }
   }
   return ver;
}
```

OUTPUT

```
Enter the no: of vertices4
Enter the no: of edges5
Enter the edges
start end weight
0 1 5
0 2 8
1 2 9
1 3 2
3 2 6
Enter the starting vertex0
path cost to 1= 5   0 ->1
path cost to 2= 8   0 ->2
path cost to 3= 7   0 ->1  ->3
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdlib.h>
#include<stdio.h>
struct node{
int vertex;
struct node *next;
};
int v,e;
struct node **adj;
int que[30],visited[30];
int f=-1,r=-1;
void enq(int x)
{ if (f==-1 && r==-1)
f=0;
r=(r+1)%v;
que[r]=x;
}
int dequ()
{
 int data;
data=que[f];
if (f==r)
f=r=-1;
else
f=(f+1)%v;
return data;
}
void bfs(){
 struct node *ptr;
int ver,i,w;
```

```
for(i=0;i<=v;i++)
visited[i]=0;
enq(1);
visited[1]=1;
printf("%d ",1);
while(!(f==-1)){
 ver=dequ();
ptr=adj[ver];
while(ptr!=NULL){
w=ptr->vertex;
if (visited[w]==0){
 enq(w);
printf("%d ",w);
visited[w]=1;
}
ptr=ptr->next;
}
}
}
void main()
{
int s,i,en;
struct node *ne;
printf("Enter No of vertices:");
scanf("%d",&v);
adj= (struct node **)malloc((v+1)*sizeof(struct node *));
for(i=0;i<=v;i++)
adj[i]=NULL;
printf("enter No of Edges:");
scanf("%d",&e);
```

```
printf("Enter the edges:\n");
printf("start End\n");
for(i=0;i<e;i++){
scanf("%d%d",&s,&en);
ne=(struct node*)malloc(sizeof(struct node));
ne->vertex=en;
ne->next=adj[s];
adj[s]= ne;
}
printf("\nbfs\n");
bfs();
getch();
}
```

OUTPUT

```
Enter No of vertices:5
enter No of Edges:5
Enter the edges:
start End
0 1
0 2
0 3
1 2
2 4

bfs
1 2 4
--------------------------------
Process exited after 46.17 seconds with return value 13
Press any key to continue . . .
```

RESULT

Program executed successfully and result is verified.