SREE NARAYANA GURUKULAM COLLEGE OF ENGINEERING

KADAYIRUPPU, KOLENCHERY 682311

LABORATORY RECORD

YEAR: 2021 **TO** 2022

NAME: MUHAMMED RASHID A P

SEMESTER: 1 **ROLL NO:**23

BRANCH: COMPUTER APPLICATIONS

Certified that this is a Bonafede 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-2023 **on----------------**

External Examiner Internal Examiner

INDEX PAGE

PROGRAM

```
#include<stdio.h>
void read(int *,int);
void main()
{
        int a[20],b[20],c[20],i=0,j=0,k=0,n1,n2;
        printf("Enter the no of elements in a :");
        scanf("%d",&n1);
        printf("Enter the elements in a (in sorted order):");
        read(a,n1);
        printf("Enter the no of elements in b (in sorted order):");
        scanf("%d",&n2);
        read(b,n2);
        i=0;
        while(i<n1 && j<n2)
        {
                if(a[i]<b[j])
                        {
                        c[k]=a[i];
                        k++;
                        i++;
                        }
                else if(a[i]>b[j])
                        {
                        c[k]=b[j];
                        k++;
                        j++;
                        }
                else
                        {
```

```
                    c[k]=a[i];
                    }
        }
        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\t",c[i]);
                }
        printf("\n");
}
void read(int *p,int y)
{
int j;
        for(j=0;j<y;j++)
        {
        scanf("%d",&p[j]);
        }
}
```

OUTPUT

```
Enter the no of elements in a :3

Enter the elements in a (in sorted order):

1

2

3

Enter the no of elements in b :3

Enter the elements in b (in sorted order):

4

5

6

1	2	3	4	5	6


Process returned 10 (0xA) execution time : 36.565 s

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 an Element \n 2.Delete an Element\n 3.Search an Element\n
        4.Dispaly \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;
        default:printf("exit");
        }
        }
        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 an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 1
 enter the element
 2

 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 1
 enter the element
 5

 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 1
 enter the element
 36
 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 4
2
5
36

 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 3
enter the element to be search36
element found
 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 2
deleted element 2
```

```
 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 4
 5
36

 MENU
 1.Insert an Element
 2.Delete an Element
 3.Search an Element
 4.Dispaly
 5 Exit
 Enter the option : 5
```

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("Choose operation :\n 1.PUSH \n2. POP \n3. Display\n4.Search\n");
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\n");
                break;
                }
                ptr=ptr->next;
                }
                if(ptr==NULL)
        {
        printf("Element Not Found\n");
        }
        }
}
```

OUTPUT

```
Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
1
Enter the element to PUSH
5
Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
1
Enter the element to PUSH
10
Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
1
Enter the element to PUSH
20
Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
```

```
2
20 is popped
Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
3
10      5       Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
4
Enter the element to search
20
Element Not Found
Choose operation :
 1.PUSH
2. POP
3. Display
4.Search
4
Enter the element to search
10
Element Found
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>

#include<stdlib.h>


void frst_insert();

void lst_insert();

void display();

void search();

void frst_delete();

void lst_delete();

void insert_pos();

void delete_pos();


struct node

{

int data;

struct node *next;

struct node *left;

struct node *right;

};

struct node *head;


void main()

{

	int opt;

	do

	{

	printf("Choose operation :\n1.Insert an element at FIRST\n2.Insert an element at 
LAST\n3.Display\n4.SEARCH\n5.Delete First Element… \n6.Delete Last 
Element. ..\n7.Insert an item at position\n8.Delete an item at Position\n");

	scanf("%d",&opt);
```

```
                    switch(opt)
                         {
                         case 1:
                           frst_insert();
                           break;
                        case 2:
                           lst_insert();
                           break;
                          case 3:
                           display();
                           break;
                         case 4:
                           search();
                           break;
                         case 5:
                           frst_delete();
                           break;
                         case 6:
                           lst_delete();
                           break;
                         case 7:
                           insert_pos();
                           break;
                         case 8:
                           delete_pos();
                           break;
                         }
           }
           while(opt!=0);
     }
```

```
void frst_insert()
{
        struct node *ne;
        int x;
        printf("Enter the element to INSERT\n");
        scanf("%d",&x);
        ne=(struct node *)malloc(sizeof(struct node));
        if (ne == NULL)
                {
                printf("Insufficient Memory") ;
                return;
                }
        ne->data=x;
        ne->left=NULL;
        ne->right=NULL;
        if(head == NULL)
                {
                head=ne;
                }
        else
                {
                ne->right=head;
                head->left=ne;
                head=ne;
                }
}

void lst_insert()
{
        struct node *ne;
        struct node *ptr;
        int x;
        printf("Enter the element to INSERT\n");
        scanf("%d",&x);
        ne=(struct node *)malloc(sizeof(struct node));
        if (ne == NULL)
                {
                printf("Insufficient Memory") ;
                return;
                }
        ne->data=x;
        ne->left=NULL;
        ne->right=NULL;
        if(head == NULL)
                {
                head=ne;
                }
        else
                {
                ptr=head;
                while(ptr->right!=NULL)
                        {
                        ptr=ptr->right;
                        }
                ptr->right=ne;
                ne->left=ptr;
                }
}

void display()
{
struct node *ptr;
```

```
if (head == NULL)
          {
          printf("LInked List is Empty") ;
          return;
          }
ptr=head;
while(ptr!=NULL)
                    {
                    printf("%d \t",ptr->data);
                    ptr=ptr->right;
                    }
printf("\n");
}
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");
                                        printf("\n");
                                        break;
                              }
                              ptr=ptr->right;
                              }
                              if(ptr==NULL)
                                        {
                                        printf("Element Not Found");
                                        printf("\n");
                                        }
          }
}

void frst_delete()
{
          struct node *ptr;
          if (head == NULL)
                    {
                    printf("LInked List is Empty") ;
                    return;
                    }
          ptr = head;
          head = head->right;
          if (head == NULL)
                    {
                    head->left=NULL;
                    }
          free(ptr);
```

```
}

void lst_delete()
{
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;
free(ptr);
}

void insert_pos()
{
          struct node *ne;
          struct node *ptr,*ptr1;
```

```
    int x,key;
    printf("Enter the element to INSERT\n");
    scanf("%d",&x);
    printf("Enter the position where you want to insert item: ");
scanf("%d", &key);
    ne=(struct node *)malloc(sizeof(struct node));
    if (ne == NULL)
            {
            printf("Insufficient Memory") ;
            return;
            }
    ne->data=x;
    ne->left=NULL;
    ne->right=NULL;
    if(head == NULL)
    {
            head=ne;
            return;
    }
    ptr = head;
    while(ptr->data!=key&&ptr->right!=NULL)
    {
            ptr=ptr->right;
    }
    if(ptr->right==NULL)
    {
            ptr->right=ne;
            ne->left=ptr;
    }
    else
```

```
        {
                ne->left=ptr;
                ne->right=ptr->right;
                ptr1=ptr->right;
                ptr->right=ne;
                ne->left=ne;
        }
}

void delete_pos()
{
        struct node *ptr;
        struct node *prev;
        struct node *next;
        int x;
        if (head == NULL)
        {
                printf("LInked List is Empty") ;
        }
        printf("Enter the Item you want to DELETE : ");
        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);
        }
}
}
```

OUTPUT

```
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
```

```
7.Insert an item at position
8.Delete an item at Position
1
Enter the element to INSERT
24
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
2
Enter the element to INSERT
45
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
1
Enter the element to INSERT
21
Choose operation :
```

```
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
2
Enter the element to INSERT
68
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
3
21      24      45      68
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
```

```
8.Delete an item at Position
4
Enter the element to search
45
Element Found
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
8
Enter the Item you want to DELETE : 45
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
3
21      24      68
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
```

```
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
7
Enter the element to INSERT
25
Enter the position where you want to insert item: 24
Choose operation :
1.Insert an element at FIRST
2.Insert an element at LAST
3.Display
4.SEARCH
5.Delete First Element....
6.Delete Last Element....
7.Insert an item at position
8.Delete an item at Position
3
21      24      25      68
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
int data;
struct node *left;
struct node *right;
};
struct node *ROOT=NULL;

void insert();
void search();
void inorder(struct node *);
void preorder(struct node *);
void postorder(struct node *);
void delete(int);

void main()
{
        int opt,x;
        do
        {
        printf("Choose operation
:\n1.INSERT\n2.Inorder_traversal\n3.PreOrder_Traversal\n4.PostOrder_Traversal\n5.search
\n6.Delete\n");
        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("Enter Data to be deleted\n");
                        scanf("%d",&x);
                        delete(x);
                    break;
                 }
        }
        while(opt!=7);
}

void insert()
{
struct node *ne,*ptr,*ptr1;
int x;
ne=(struct node *)malloc(sizeof(struct node));
        if (ne == NULL)
                {
                printf("Insufficient Memory") ;
                return;
                }

        printf("Enter Data to insert\n");
```

```
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("Data 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 search()
{
int x;
struct node *ptr=ROOT;
printf("\nEnter the data to search: ");
scanf("%d",&x);
while(ptr!=NULL)
        {
                if(ptr->data==x)
                        {
                        printf("Data is present...\n") ;
                        break;
                        }
                if(x > ptr->data)
                        {
                        ptr= ptr->right;
                        }
                else
                {
                    ptr= ptr->left;
                    }
          }
if(ptr==NULL)
          printf("\n Data not present. ..\n");
}

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 delete(int x)
{
struct node *ne,*ptr,*parent,*p;
int dat;
        if (ROOT == NULL)
                {
                printf("\n Tree is Empty \n") ;
                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("\n Data not present. ..\n");
                        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;
        delete(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("Element deleted");
free(ptr);
return;
}
```

OUTPUT

```
Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
1
Enter Data to insert
25
Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
1
Enter Data to insert
57
Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
1
Enter Data to insert
```

```
69
Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
2
25 57 69 Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
3
25 57 69 Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
4
69 57 25 Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
```

```
5.search
6.Delete
5

Enter the data to search: 57
Data is present....
Choose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
6
Enter Data to be deleted
69
Element deletedChoose operation :
1.INSERT
2.Inorder_traversal
3.PreOrder_Traversal
4.PostOrder_Traversal
5.search
6.Delete
2
25 57
```

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:
101010001101
Enter set2:
100100110010

union
101110111111
intersection
100000000000
setdifference
001010001101
Process returned 27 (0x1B) execution time : 47.981 s
Press any key to continue.
```

RESULT

Program executed successfully and result is verified

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;

 struct node *next;
};
 struct node *first[20];
 void makeset(int);
 void unions(int,int);
 int find(int);
 int n=0;
 struct edge
 {
  int start;
  int weight;
  int end;
 };
 struct edge adj[30], A[30];
void main()
{

 int i,nv,sum,k,en,s,w,e,u,v,c,count;
 printf("Enter no: of vertices: ");
 scanf("%d",&nv);
 for(i=1;i<=nv;i++)
 makeset(i);
 printf("Enter the no. of edges:");
```

```
scanf("%d",&e);
printf("Enter the edjes\n");
printf("start end weight\n");
c=-1;
for(i=1;i<=e;i++)
{
 scanf("%d %d %d",&s,&en,&w);
 for(k=c;k>=0;k--)
 {
  if(adj[k].weight>w)
  adj[k+1]=adj[k];
  else
  break;
 }

 adj[k+1].start=s;
 adj[k+1].end=en;
 adj[k+1].weight=w;
 c++;
}
count=0;
for(i=0;i<c;i++)
{
 u=adj[i].start;
 v=adj[i].end;
 if(find(u)!=find(v))
 {
  A[count].start=u;
  A[count].end=v;
  A[count].weight=adj[i].weight;
```

```
   unions(u,v);
   count++;
  }
 }
 printf("\nSpanning Tree edjes\n");
 sum=0;
 for(i=0;i<count;i++)
 {
  printf("(%d->%d) w:%d\n" ,A[i].start,A[i].end,A[i].weight);
  sum=sum+A[i].weight;
 }
 printf("\nTotal Cost=%d" ,sum);
}
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("Number already exist\n");
 }
}
void unions(int x,int y)
```

```
{
struct node *p;
int i,j;
i=find(x);
j=find(y);
  if (i==-1 || j ==-1)
      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 :5
The Sets are :{1 }
{2 }
{3 }
{4 }
{5 }
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
```

```
4
Enter the number :8
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
1
The Sets are :{1 }
{2 }
{3 }
{4 }
{5 }
{8 }
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
2

Enter the first element:2
Enter the second element:8
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
```

```
5.Exit
1
The Sets are :{1 }
{2 8 }
{3 }
{4 }
{5 }
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
3
Enter the element to Find
2
Element PRESENT in set-> 2
Choose operation :
1.Display
2.Union
3.Find
4.Makeset
5.Exit
5

Process returned 5 (0x5) execution time : 112.015 s
Press any key to continue.
```

RESULT

Program executed successfully and result is verified.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;

 struct node *next;
};
 struct node *first[20];
 void makeset(int);
 void unions(int,int);
 int find(int);
 int n=0;
 struct edge
 {
  int start;
  int weight;
  int end;
 };
 struct edge adj[30], A[30];
void main()
{

 int i,nv,sum,k,en,s,w,e,u,v,c,count;
 printf("Enter no: of vertices: ");
 scanf("%d",&nv);
 for(i=1;i<=nv;i++)
 makeset(i);
 printf("Enter the no. of edges:");
```

```
scanf("%d",&e);
printf("Enter the edjes\n");
printf("start end weight\n");
c=-1;
for(i=1;i<=e;i++)
{
 scanf("%d %d %d",&s,&en,&w);
 for(k=c;k>=0;k--)
 {
  if(adj[k].weight>w)
  adj[k+1]=adj[k];
  else
  break;
 }

 adj[k+1].start=s;
 adj[k+1].end=en;
 adj[k+1].weight=w;
 c++;
}
count=0;
for(i=0;i<c;i++)
{
 u=adj[i].start;
 v=adj[i].end;
 if(find(u)!=find(v))
 {
  A[count].start=u;
  A[count].end=v;
  A[count].weight=adj[i].weight;
```

```
   unions(u,v);
   count++;
  }
 }
 printf("\nSpanning Tree edjes\n");
 sum=0;
 for(i=0;i<count;i++)
 {
  printf("(%d->%d) w:%d\n" ,A[i].start,A[i].end,A[i].weight);
  sum=sum+A[i].weight;
 }
 printf("\nTotal Cost=%d" ,sum);
}
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("Number already exist\n");
 }
}
void unions(int x,int y)
```

```
{
struct node *p;
int i,j;
i=find(x);
j=find(y);
  if (i==-1 || j ==-1)
      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 no: of vertices: 7
Enter the no. of edges:9
Enter the edjes
start end weight
1 2 28
2 3 16
3 4 12
4 5 22
5 6 25
6 1 10
5 7 24
7 2 14
7 4 18
```

```
Spanning Tree edjes
(6->1) w:10
(3->4) w:12
(7->2) w:14
(2->3) w:16
(4->5) w:22
(5->6) w:25

Total Cost=99
Process returned 14 (0xE) execution time : 102.497 s
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)  ;

struct node * getNode()
{
   struct node *ne;
   ne=(struct node *) malloc(sizeof(struct node));
   if (ne==NULL)
     printf("No Memory");
   return ne;
}

void inorder(struct node *ptr)
{  if (ptr!=NULL)
   {  inorder(ptr->left);
      printf("%d(%c) ",ptr->data,ptr->color==0?'b':'r');
      inorder(ptr->right);
   }
```

```
}

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;
}

void insert()
{ int x;
   struct node *ne,*parent,*ptr,*pparent,*uncle;

   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 (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)
  {
          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 (uncle==NULL)
     {          doop(ne,parent,pparent);
                break;
     }
if (uncle->color==black )
       {        doop(ne,parent,pparent);
                break;
     }

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)
{
            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)
    {     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)
   { 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("\n 1.Insert\n 2.display\n 3.Exit\n Enter Your choice :");
    scanf("%d",&ch);
    switch(ch)
    {  case 1:insert();
                break;
       case 2:inorder(ROOT);
                    break;
    }
  }while(ch!=3);
}
```

OUTPUT

```
 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 8

 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 18

 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 5

 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 15

 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 17

 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 25

 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 40

 1.Insert
 2.display
 3.Exit
 Enter Your choice :2
5(b)  8(b)  15(b)  17(r)  18(r)  25(b)  40(r)
 1.Insert
 2.display
 3.Exit
 Enter Your choice :1
Enter the element to insert : 80

 1.Insert
 2.display
 3.Exit
 Enter Your choice :2
5(b)  8(r)  15(b)  17(b)  18(b)  25(r)  40(b)  80(r)
```

```
 1.Insert
 2.display
 3.Exit
 Enter Your choice :3

Process returned 3 (0x3) execution time : 191.859 s
Press any key to continue.
```

RESULT

Program executed successfully and result is verified

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{  int vertex;
   struct node  *next;
};
int v,e;
struct node* adj[20];
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 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;
    }


    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;
}
```

OUTPUT

```
Enter No: of vertices5
enter No: of Edjes7
Enter the edges
start End
1 5
5 4
4 3
2 3
1 2
1 3
1 4

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

RESULT

Program executed successfully and result is verified

PROGRAM

```
#include<stdlib.h>
struct node
{  int vertex;
   struct node  *next;
};
int v,e;
struct node *adj[20],*adj1[20];
int visited[20],ft[20];
int t=0;
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 Edjes ;");
    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 strongly connected 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 vertices :7
enter No: of Edjes :8
Enter the edges
start End
7          5
5          6
6          7
4          5
1          4
1           2
2          3
3          1

dfs
1 2 3 4 5 6 7
strongly connected components
{ 1 3 2 }
{ 4 }
{ 5 7 6 }
```

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 5 5
5 4 2
4 3 4
2 3 3
1 2 1
1 3 7
1 4 10
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 vertices9
Enter the no: of edges14
Enter the edges
start end weight
0 1 4
```

```
0 7 8

1 7 11

1 2 8

7 6 1

7 8 7

2 8 2

8 6 6

2 5 4

2 3 7

6 5 2

3 5 14

3 4 9

5 4 10

Enter the starting vertex0

path cost to 1= 4  0 ->1

path cost to 2= 12  0 ->1  ->2

path cost to 3= 19  0 ->1  ->2  ->3

path cost to 4= 21  0 ->7  ->6  ->5  ->4

path cost to 5= 11  0 ->7  ->6  ->5

path cost to 6= 9  0 ->7  ->6

path cost to 7= 8  0 ->7

path cost to 8= 14  0 ->1  ->2  ->8
```

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:4
enter No of Edges:6
Enter the edges:
start End
0 2
2 0
0 1
1 2
2 3
3 3
bfs
1 2 3 0
```

RESULT

Program executed successfully and result is verified.