MERGE TWO SORTED ARRAYS

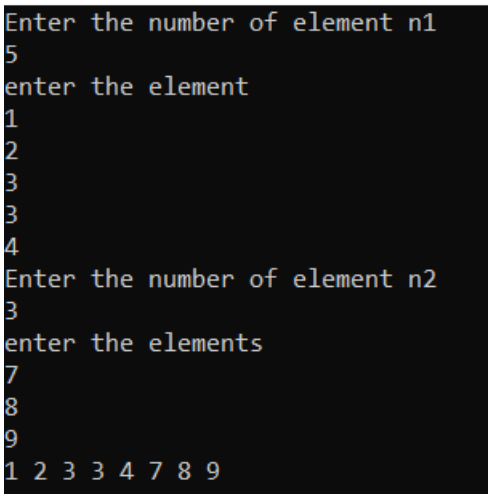
PROGRAM:

```
#include<stdio.h>
void read(int *,int);
void read1(int *,int);
int main()
{
int a[20],b[20],c[20],k=0,n1,n2;
printf("Enter the number of element n1\n");
scanf("%d",&n1);
read(a,n1);
printf("Enter the number of element n2\n");
scanf("%d",&n2);
read1(b,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 n1)
{
        int i;
printf("enter the element\n");
for(i=0; i<n1;i++)
{
scanf("%d",&a[i]);
}
}
void read1(int *b,int n2)
```

```
{
int j;
printf("enter the elements\n");
for(j=0 ;j<n2;j++)
{
scanf("%d",&b[j]);
}
}
```

OUT PUT:



CIRCULAR QUEUE

PROGRAM:

```
#include<stdio.h>
void insert(int *);
void display(int *);
void delet(int *);
void search(int *);
int front=-1,rear=-1,sz=4;
void main()
{
int q[20],opt;
do {
printf("\nMenu");
printf("\n1.Insert\n2.Delete\n3.Search\n4.Display\n5.Exit\n");
printf("Select your option\n");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert(q);
break;
case 2:
delet(q);
break;
case 3:
search(q);
break;
case 4:
display(q);
break;
default:
printf("Exited");
}
```

```
}while(opt!=5);
}
void insert(int *q)
{
if(front==(rear+1)%sz)
{
printf("Queue is full\n");
return;
}
if(front==-1)
front=0;
rear=(rear+1)%sz;
printf("Enter the element to insert\n");
scanf("%d",&q[rear]);
}
void delet(int *q)
{
if(front==-1)
{
printf("Queue is empty\n");
return;
}
printf("Deleted Element %d",q[front]);
if(front==rear)
front=rear=-1;
else
front=(front+1)%sz;
printf("\n");
return;
}
void display(int *q)
{
int f;
if(front==-1)
```

```
{
printf("\nQ is empty");
return;
}
f=front;
printf("\nElements in the queue:");
while(1)
{
printf("%d\t",q[f]);
if(f==rear)
break;
f=(f+1)%sz;
}
printf("\n");
}
void search(int *q)
{
int f,n,c=0;
printf("Enter the element to search\n");
scanf("%d",&n);
if(front==-1)
{
printf("Q is empty");
return;
}
f=front;
while(1)
{
if(n==q[f])
{
printf("%d",q[f]);
printf("\nElement found");
break;
}
```

```
if(f==rear)
{
printf("\nElement not found");
break;
}
f=(f+1)%sz;
}
printf("\n");
}
```

OUT PUT :

```
Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
1
Enter the element to insert
1

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
1
Enter the element to insert
2

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
1
Enter the element to insert
3

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
1
Enter the element to insert
4
```

```
Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
4

Elements in the queue:1 2       3       4

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
2
Deleted Element 1

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
4

Elements in the queue:2 3       4

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
3
Enter the element to search
1

Element not found
```

```
Element not found

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
3
Enter the element to search
2
2
Element found

Menu
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Select your option
```

SINGLY LINKED STACK

PROGRAM:

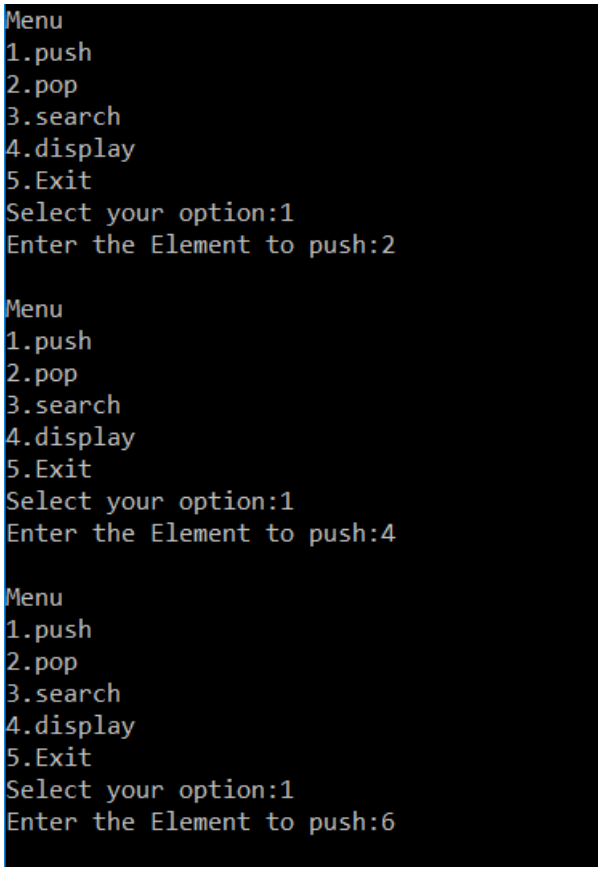
```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void search();
void display();
struct node
{
int data;
struct node *next;
};
struct node *top=NULL;
void main()
{
int opt;
do
{
 printf("\nMenu");
 printf("\n1.push\n2.pop\n3.search\n4.display\n5.Exit");
 printf("\nSelect your option:");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

```
 case 4:
 display();
 break;
 default:
 printf("Exited");
 break;
 }
}while(opt!=5);
}
void push()
{
int x;
struct node *ne;
printf("Enter the Element to push:");
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()
{
int item;
struct node *ptr;
if(top==NULL)
printf("\nStack is empty");
else
{
item=top->data;
```

```
ptr=top;
printf("\nPoped Element=%d",item);
top=top->next;
free(ptr);
}
}
void search()
{
int c=0,x,f=0;
struct node *ptr;
if(top==NULL)
printf("\nStack is empty");
else
{
printf("\nEnter the element to search:");
scanf("%d",&x);
ptr=top;
while(ptr!=NULL)
{
if(ptr->data==x)
{
f=1;
printf("\nElement found at node %d",c);
break;
}
ptr=ptr->next;
c++;
}
if(f==0)
printf("\nElement not found");
}
}
void display()
{
```

```
struct node *ptr;
if(top==NULL)
printf("Stack empty");
else
{
ptr=top;
printf("\nElements in stack:");
while(ptr!=NULL)
{
printf("%d\t",ptr->data);
ptr=ptr->next;
}}}
```

OUT PUT:



```
Menu
1.push
2.pop
3.search
4.display
5.Exit
Select your option:4

Elements in stack:6     4       2
Menu
1.push
2.pop
3.search
4.display
5.Exit
Select your option:2

Poped Element=6
Menu
1.push
2.pop
3.search
4.display
5.Exit
Select your option:4

Elements in stack:4     2
Menu
1.push
2.pop
3.search
4.display
5.Exit
Select your option:3

Enter the element to search:1

Element not found
```

```
Menu
1.push
2.pop
3.search
4.display
5.Exit
Select your option:3

Enter the element to search:2

Element found at node 1
```

DOUBLY LINKED LIST

PROGRAM:

```
#include<stdlib.h>
#include<stdio.h>
void insert_first();
void insert_last();
void insert_pos();
void delete_first();
void delete_last();
void delete_pos();
void search();
void display();
struct node
{
struct node *left;
int data;
struct node *right;
};
struct node *head=NULL;
void main()
{
int opt;
do
{
 printf("\nMenu");
 printf("\n1.Insert At First\n2.Insert At Last\n3.Search\n4.display\n5.Delete First\n6.Delete Last\n7.Insert
at position\n8.Delete At Position\n9.Exit");
 printf("\nSelect your option:");
 scanf("%d",&opt);
 switch(opt)
 {
 case 1:
 insert_first();
 break;
```

```
 case 2:
 insert_last();
 break;
 case 3:
 search();
 break;
 case 4:
 display();
 break;
 case 5:
 delete_first();
 break;
 case 6:
 delete_last();
 break;
 case 7:
 insert_pos();
 break;
 case 8:
 delete_pos();
 break;
 default:
 printf("Exited");
 }
}while(opt!=9);
}
void insert_first()
{
int x;
struct node *ne;
ne=(struct node *)malloc(sizeof(struct node));
if(ne==NULL)
printf("Insufficient Memory");
else
```

```
{
printf("\nEnter the data to insert\n");
scanf("%d",&x);
ne->data=x;
ne->left=NULL;
ne->right=NULL;
if(head==NULL)
 head=ne;
else
{
ne->right=head;
head->left=ne;
head=ne;
}}
}
void insert_last()
{
int x;
struct node *ne,*ptr;
ne=(struct node *)malloc(sizeof(struct node));
if(ne==NULL)
printf("Insufficient Memory");
else
{
printf("\nEnter the data to insert\n");
scanf("%d",&x);
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 insert_pos()
{
int x,k;
struct node *ne,*ptr,*ptr1;
ne=(struct node *)malloc(sizeof(struct node));
if(ne==NULL)
printf("Insufficient Memory");
else
{
printf("\nEnter the data to insert\n");
scanf("%d",&x);
printf("\nEnter the key value\n");
scanf("%d",&k);
ne->data=x;
ne->left=NULL;
ne->right=NULL;
if(head==NULL)
 head=ne;
else
{
ptr=head;
while(ptr->right!=NULL && ptr->data!=k)
ptr=ptr->right;
if(ptr->right==NULL)
{
ptr->right=ne;
```

```
ne->left=ptr;
}
else
{
ptr1=ptr->right;
ne->right=ptr1;
ptr1->left=ne;
ptr->right=ne;
ne->left=ptr;
}}}}
void delete_first()
{
struct node *ptr;
if(head==NULL)
printf("List is Empty");
else
{
ptr=head;
if(ptr->right==NULL)
{
head=NULL;
free(ptr);
}
else
{
if(head!=NULL)
{
head->left=NULL;
head=head->right;
free(ptr);
}}}}

void delete_last()
{
```

```
struct node *ptr,*prev;
if(head==NULL)
printf("List is Empty");
else
{
if(head->right==NULL)
{
free(head);
head=NULL;
}
else
{
ptr=head;
while(ptr->right!=NULL)
{
ptr=ptr->right;
}
prev=ptr->left;
prev->right=NULL;
free(ptr);
}}}

void delete_pos()
{
struct node *ptr,*next,*prev;
int x;
if(head==NULL)
printf("\nList is empty");
else
{
printf("\nEnter the data:\n");
scanf("%d",&x);
if(head->data==x)
{
```

```
ptr=head;
head=head->right;
if(head!=NULL)
{
head->left=NULL;
}
free(ptr);return;
}
ptr=head;
while(ptr->data!=x && ptr->right!=NULL)
ptr=ptr->right;
if(ptr->data==x)
{
next=ptr->right;
prev=ptr->left;
prev->right=ptr->right;
if(next!=NULL)
next->left=prev;
free(ptr);
return;
}
printf("\nElement not found");
}}
void display()
{
struct node *ptr;
if(head==NULL)
printf("List is empty");
else
{
ptr=head;
printf("List:");
while(ptr!=NULL)
{
```

```
printf("%d\t",ptr->data);
ptr=ptr->right;
}}}
void search()
{
struct node *ptr;
int x,c=0;
if(head==NULL)
printf("List is empty");
else{
printf("Enter the element to search\n");
scanf("%d",&x);
ptr=head;
while(ptr!=NULL)
{
if(ptr->data==x)
{
c=1;
printf("\nElement found:");
break;
}
ptr=ptr->right;
}
if(c==0)
printf("\nElement not found");
}}
```

OUTPUT:

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

Enter the data to insert
1

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

Enter the data to insert
2
```

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

Enter the data to insert
3

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

Enter the data to insert
4

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

Element found:
```

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

Enter the data to insert
3

Enter the key value
8

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

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

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

Enter the data:
8

Element not found
Menu
1.Insert At First
2.Insert At Last
3.Search
4.display
5.Delete First
6.Delete Last
7.Insert at position
8.Delete At Position
9.Exit
Select your option:4
List:1  3       4
```

BINARY SEARCH TREE

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-Binary Search Tree");
 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;
}
```

OUT PUT:


Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:12
Menu-Binary Search Tree
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-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:34
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:1
Enter the data to insert:45
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:5
Enter the data to search:34
Data present
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:5
Enter the data to search:1
Data not present
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:2
12 23 34 45
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:3
12 23 34 45
Select your option:3
12 23 34 45
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:4
45 34 23 12
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:6
Enter the element to delete:
34
Element deleted
Menu-Binary Search Tree
1.Insertion
2.Inorder
3.Preorder
4.Postorder
5.Search
6.Deletion
7.Exit
Select your option:

SET DATASTRUCTURE OPERATIONS USING BITSTRING

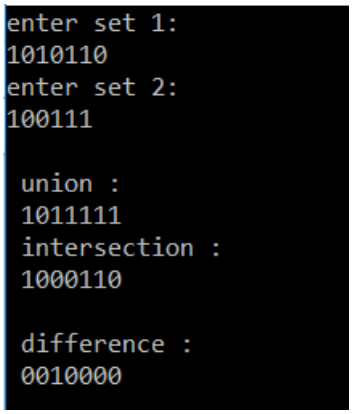
PROGRAM:

```
#include<stdio.h>
#include<stdlib.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 set 1:\n");
scanf("%s",s1);
printf("enter set 2:\n");
scanf("%s",s2);
setunion(s1,s2,s3);
printf("\n union :\n %s",s3);
setintersection(s1,s2,s3);
printf("\n intersection :\n %s",s3);
setdifference(s1,s2,s3);
printf("\n difference :\n %s",s3);
printf("\n");
}
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';
printf("\n");
}
```

OUT PUT:



DISJOINT SETS

PROGRAM:

```
#include<stdlib.h>
#include<stdio.h>
struct node {
int data;
struct node *next;
};
void makeset();
void unionset();
int find(int);
void display();
int n=0;
struct node *first[20];
void main()
{
int opt,x,i;
do {
printf("Menu");
printf("\n1.makeset\n2.union\n3.find\n4.display\n5.exit");
printf("\nselect your option");
scanf("%d",&opt);
switch(opt)
{
case 1:
makeset();
break;
case 2:
unionset();
break;
case 3:
printf("Enter the value for x:");
scanf("%d",&x);
i=find(x);
```

```
if(i==-1)
printf("Element not found");
else
printf("Element=%d",first[i]->data);
break;
case 4:
display();
break;
}
}while(opt!=5);
}
void makeset()
{
int x,pos;
printf("\nEnter the element:");
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("Element already exist");
}
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,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;
}}
void display()
{
int i;
struct node *p;
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");
}}
```

OUT PUT:

```
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option1

Enter the element:8
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option1

Enter the element:7
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option1

Enter the element:3
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option1

Enter the element:5
Menu
```

```
1.makeset
2.union
3.find
4.display
5.exit
select your option2

Enter the first element:8

Enter the second element:7
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option2

Enter the first element:7

Enter the second element:3
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option4
{8 7 3 }
{5 }
Menu
1.makeset
2.union
3.find
4.display
5.exit
select your option3
Enter the value for x:8
Element=8Menu
```

MINIMUM SPANNING TREE USING KRUSKAL'S ALGORITHM

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define NIL -1
struct edge
{
	int u;
	int v;
	int weight;
	struct edge *link;
}*front = NULL;
void make_tree(struct edge tree[]);
void insert_pque(int i,int j,int wt);
struct edge *del_pque();
int isEmpty_pque( );
void create_graph();
int n;
int main()
{
	int i;
	struct edge tree[MAX];
	int wt_tree = 0;
	create_graph();
	make_tree(tree);
	printf("\nEdges to be included in minimum spanning tree are :\n");
	for(i=1; i<=n-1; i++)
	{
```

```
            printf("\n%d->",tree[i].u);
            printf("%d\n",tree[i].v);
            wt_tree += tree[i].weight;
        }
        printf("\nWeight of this minimum spanning tree is : %d\n", wt_tree);
return 0;
}
void make_tree(struct edge tree[])
{
        struct edge *tmp;
        int v1,v2,root_v1,root_v2;
        int father[MAX];
        int i,count = 0;
      for(i=0; i<n; i++)
            father[i] = NIL;
        while( !isEmpty_pque( ) && count < n-1 )
        {
            tmp = del_pque();
            v1 = tmp->u;
            v2 = tmp->v;
        while( v1 !=NIL )
            {
                root_v1 = v1;
                v1 = father[v1];
}
            while( v2 != NIL )
            {
                root_v2 = v2;
                v2 = father[v2];
            }
```

```
            if( root_v1 != root_v2 )/*Insert the edge (v1, v2)*/
            {
               count++;
                  tree[count].u = tmp->u;
                  tree[count].v = tmp->v;
                  tree[count].weight = tmp->weight;
                  father[root_v2]=root_v1;
            }       }
if(count < n-1)
      {
            printf("\nGraph is not connected, no spanning tree possible\n");
            exit(1);
      }
}/*End of make_tree()*/
/*Inserting edges in the linked priority queue */
void insert_pque(int i,int j,int wt)
{
      struct edge *tmp,*q;
      tmp = (struct edge *)malloc(sizeof(struct edge));
      tmp->u = i;
      tmp->v = j;
      tmp->weight = wt;
      /*Queue is empty or edge to be added has weight less than first edge*/
      if( front == NULL || tmp->weight < front->weight )
      {
            tmp->link = front;
            front = tmp;
      }
      else
      {
```

```
            q = front;
            while( q->link != NULL && q->link->weight <= tmp->weight )
                  q = q->link;
            tmp->link = q->link;
            q->link = tmp;
            if(q->link == NULL) /*Edge to be added at the end*/
                  tmp->link = NULL;
      }
}/*End of insert_pque()*/
struct edge *del_pque()
{
      struct edge *tmp;
      tmp = front;
      front = front->link;
      return tmp;
}
int isEmpty_pque( )
{
      if ( front == NULL )
            return 1;
      else
            return 0;
}
void create_graph()
{
      int i,wt,max_edges,origin,destin;
      printf("\nEnter number of vertices : ");
      scanf("%d",&n);
      max_edges = n*(n-1)/2;
      for(i=1; i<=max_edges; i++)
```

```
    {
        printf("\nEnter edge %d(-1 -1 to quit): ",i);
        scanf("%d %d",&origin,&destin);
        if( (origin == -1) && (destin == -1) )
            break;
        printf("\nEnter weight for this edge : ");
        scanf("%d",&wt);
      if( origin >= n || destin >= n || origin<0 || destin<0)
        {
            printf("\nInvalid edge!\n");
            i--;
        }
        else
            insert_pque(origin,destin,wt);
    }
}
```

OUT PUT:

```
Enter number of vertices : 4

Enter edge 1(-1 -1 to quit): 0
0

Enter weight for this edge : 1

Enter edge 2(-1 -1 to quit): 0
1

Enter weight for this edge : 4

Enter edge 3(-1 -1 to quit): 1
3

Enter weight for this edge : 5

Enter edge 4(-1 -1 to quit): 3
2

Enter weight for this edge : 8

Enter edge 5(-1 -1 to quit): 0
2

Enter weight for this edge : 1

Enter edge 6(-1 -1 to quit): 0
3

Enter weight for this edge : 6

Edges to be included in minimum spanning tree are :

0->2

0->1

1->3

Weight of this minimum spanning tree is : 10
```

RED BLACK TREE OPERATIONS

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;
}
int main()
{
int ch;
do{
    printf("\n1.Insert \n 2.display  3.Exit\nEnter Your choice");
    scanf("%d",&ch);
    switch(ch)
    {  case 1:insert();
  break;
       case 2:inorder(ROOT);
break;
    }
  }while(ch!=3);
```

OUT PUT:

```
1.Insert
 2.display  3.Exit
Enter Your choice1
Enter the element to insert23

1.Insert
 2.display  3.Exit
Enter Your choice1
Enter the element to insert45

1.Insert
 2.display  3.Exit
Enter Your choice1
Enter the element to insert3

1.Insert
 2.display  3.Exit
Enter Your choice1
Enter the element to insert39

1.Insert
 2.display  3.Exit
Enter Your choice1
Enter the element to insert23
Data already present
1.Insert
 2.display  3.Exit
Enter Your choice1
Enter the element to insert8

1.Insert
 2.display  3.Exit
Enter Your choice2
3(b)  8(r)  23(b)  39(r)  45(b)
1.Insert
 2.display  3.Exit
Enter Your choice
```

DFS AND TOPOLOGICAL SORTING ON GRAPHS

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 Number of vertices\n");
        scanf("%d",&v);
        for(i=0;i<=v;i++)
                adj[i]=NULL;
        printf("\nEnter number of edjes\n");
        scanf("%d",&e);
        printf("\nEnter edges\n");
        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++;
}
```

OUT PUT:

```
Enter Number of vertices
5

Enter number of edjes
8

Enter edges

start------End
1 2
1 3
1 5
2 3
2 4
2 5
4 3
4 5

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

STRONGLY CONNECTED COMPONENTS

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 number of vertices\n");
        scanf("%d",&v);

        for(i=0;i<=v;i++)
```

```
		adj[i]=adj1[i]=NULL;
	printf("\nEnter nunber of edges:\n");
	scanf("%d",&e);
	printf("\nEnter the edges\n");
	printf("\nStart--------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;
        }}
```

OUT PUT :

```
Enter number of vertices
7

Enter nunber of edges:
10

Enter the edges

Start--------End
1 2
1 4
3 1
2 4
4 3
2 3
4 5
5 6
6 4
7 6

DFS
1 4 5 6 3 2 7
 components
{7 }
{1 3 2 4 6 5 }
```

MINIMUM SPANNING TREE USING PRIM'S ALGORITHM

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];
void main()
{
int i,s,en,we,e,u,w,sum=0;
   struct node *ptr;
   printf("Enter Number of vertices: \n");
   scanf("%d",&v);
   for(i=1;i<=v;i++)
   {
p[i]=0;
   key[i]=inf;
   q[i]=1;
   adj[i]=NULL;
```

```
    }
    printf("\nNumber of edges:\n ");
    scanf("%d",&e);
    printf("\nEnter the adges\n");
    printf("\nstart--------end---------weight\n");
    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("\NThe total cost is %d \N",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;
 }
```

OUT PUT:

```
Enter Number of vertices:
7

Number of edges:
 12

Enter the adges

start--------end---------weight
2 1 1
1 3 2
1 4 6
3 2 10
2 6 2
4 3 8
3 5 11
5 4 5
6 3 4
5 6 9
6 7 3
7 3 7
Spanning tree edges
(2-1) w:1
(3-1) w:2
(4-1) w:6
(5-4) w:5
(6-2) w:2
(7-6) w:3
NThe total cost is 19 N
--------------------------------
```

SINGLE SOURCE SHORTEST PATH

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

#define inf 999

void printpath(int,int);

int v,adj[20][20],dist[20],visit[20],pred[20];

void main()

{       int e,st,en,w,i,j,src,ver,k;

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");
   }}
```

```
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;
```

OUT PUT:

```
Enter the no: of vertices4
Enter the no: of edges5
Enter the edges
start end weight
1 2 1
2 4 5
4 1 3
1 3 6
3 2 3
Enter the starting vertex2
path cost to 1= 8   2 ->4  ->1
path cost to 2= 0   2 ->2
path cost to 3= 14   2 ->4  ->1  ->3
path cost to 4= 5   2 ->4
```

BREADTH FIRST SEARCH

PROGRAM:

```
#include<stdlib.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\n");
scanf("%d",&v);
adj= (struct node **)malloc((v+1)*sizeof(struct node *));
for(i=0;i<=v;i++)
adj[i]=NULL;
printf("\nenter number of edges\n");
scanf("%d",&e);
printf("\nenter edges\n");
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;
}
printf("\n BFS \n");
bfs();
getch();
}
```

OUT PUT:

```
Enter No: of vertices
5

enter number of edges
8

enter edges

start-------------end
1 2
1 3
1 4
1 5
2 4
3 4
5 4
2 5

 BFS
15432
```