Musaliar College of Engineering & Technology

MUSALIAR COLLEGE P.O., PATHANAMTHITTA - 689 653

....

LABORATORY RECORD

Certified that this is the Bonafide Record of the work done by

Sri/Smt…………………………….…………………….………

of.……….……Semester Class of ……MCA…...(Roll NO……)

of.…………………………………………..…………Branch

in the…………………………….……………………Laboratory

during the academic year 20 - 20

Name of Examination:

Reg. No.	External Examiner	Staff in- charge

DEPARTMENT OF COMPUTER APPLICATIONS

VISION

“To produce competent and dynamic professionals in the field of Computer Applications to thrive and cater the changing needs of the society through research and education”.

MISSION

To impart high quality technical education and knowledge in Computer Applications.

To introduce moral, ethical and social values to Computer Application students.

To establish industry institute interaction to enhance the skills of Computer Application students.

To promote research aimed towards betterment of society.

INDEX

PROGRAM-1

Title: SUM OF TWO NUMBERS

Objectives: write a program to add two numbers using standard input and standard output.

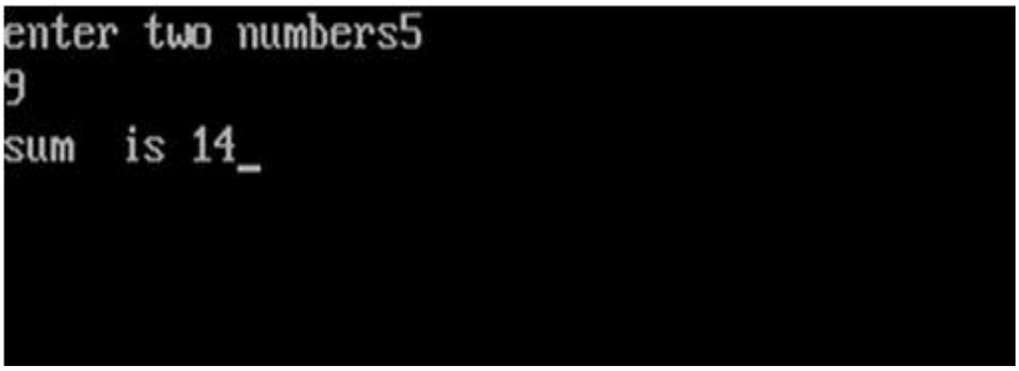
Algorithm:

```
Step 1: Start
Step 2: Declare a, b, sum
Step 3: Read a, b
Step 4: Perform sum = a + b
Step 5: Print sum
Step 6: Stop
```

Input:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
clrscr();
printf("enter two numbers");
scanf("%d%d",&a,&b);
c=a+b;
printf("sum  is %d",c);
getch();
}
```

Output:



```
enter two numbers5
9
sum  is 14_
```

Result: The program is executed successfully and output is verified.

PROGRAM-2

Title: SUM OF TWO NUMBERS USING FUNCTION

Objective: Write a program to modify by adding to find the sum of two numbers.

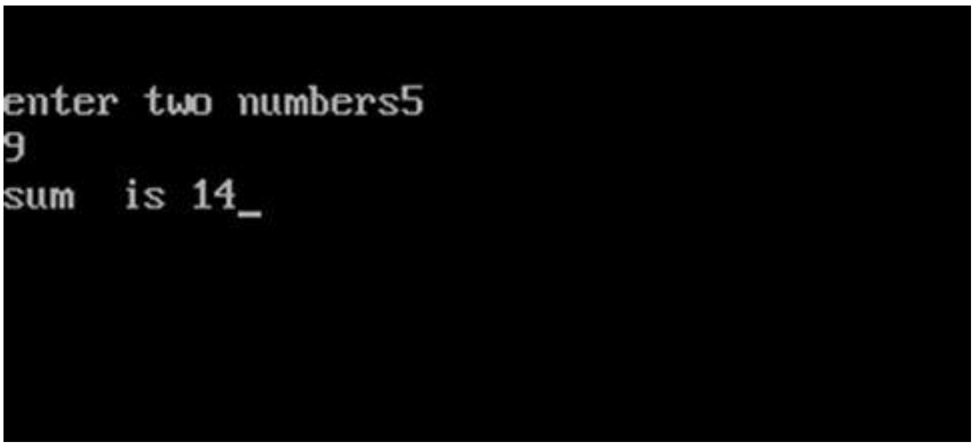
Algorithm:

```
Step 1: Start
Step 2: Declare function add
Step 3: Declare variables a, b, sum
Step 4: Read a, b
Step 5: sum=add(a,b)
Step 6: Print sum
Step 7: int add(int x,int y)
            return(x + y)
Step 8: Stop
```

Input:

```
#include<stdio.h>
#include<conio.h>
int add(int,int);
void main()
{
int a,b,sum;
clrscr();
printf("Enter the numbers");
scanf("%d%d",&a,&b);
sum=add(a,b);
printf("sum=%d",sum);
getch();
}
int add(int x,int y)
{
return(x+y);
}
```

Output:



```
enter two numbers5
9
sum  is 14_
```

Result: The program is executed successfully and output is verified.

PROGRAM-3

Title: MERGING OF TWO ARRAY

Objective: Write a program to merge two sorted array and store in the third array.

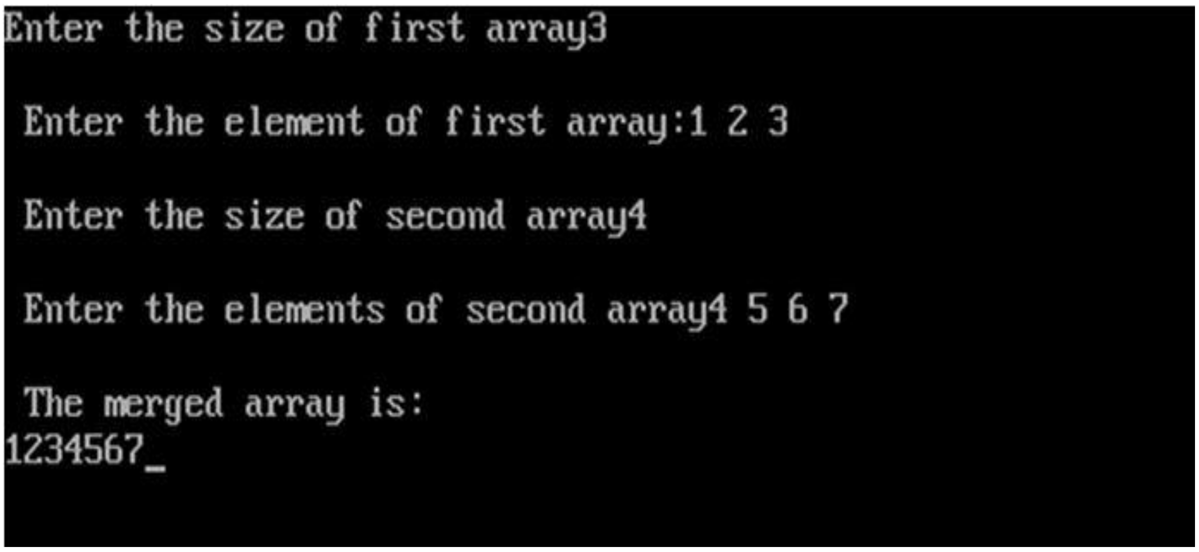
Algorithm:

```
Step 1: Start
Step 2: Declare arr1,arr2, size1, size2, i, k, merge
Step 3: Read size1, arr1
Step 4: Copy the elements of first array to merge when initializing it.
Step 5: Read size2, arr2
Step 6: Copy the elements of second array to merge when initializing it.
Step 7: Sort merged array.
Step 8: Display merge
Step 9: Stop
```

Input:

```
#include<stdio.h>
 #include<conio.h>
 int main()
 {
 int arr1[50],arr2[50],size1,size2,k,i,merge[100];
 clrscr();
 printf("Enter the size of first array");
 scanf("%d",&size1);
 printf("\n Enter the element of first array:");
 for(i=0;i<size1;i++)
{
scanf("%d",&arr1[i]);


merge[i]=arr1[i];
}
k=i;
printf("\n Enter the size of second array");
scanf("%d",&size2);
printf("\n Enter the elements of second array");
for(i=0;i<size2;i++)
{
scanf("%d",&arr2[i]);
merge[k]=arr2[i];
k++;
}
printf("\n The merged array is:\n");
for(i=0;i<k;i++)
{
printf("%d",merge[i]);
}
getch();
return 0;
 }
```

Output:



```
Enter the size of first array3

 Enter the element of first array:1 2 3

 Enter the size of second array4

 Enter the elements of second array4 5 6 7

 The merged array is:
1234567_
```

Result: The program is executed successfully and output is verified.

PROGRAM-4

Title: SINGLY LINKED LIST STACK IMPLEMENTATION

Objective: Write a program for singly linked list stack implementation.

Algorithm:

```
Step 1: Start

Step 2: Define structure of node

Step 3: Declare topelement function

Step 4: Declare push function

Step 5: Declare pop function

Step 6: Declare empty (), display (), destroy (), stack-count () and create ()

Step 7: Initialise count = 0

Step 8: Variable used no, ch and e

Step 9: Display a menu driven 1 for push, 2 for pop, 3 for Top,  4 for empty, 5 for
        Exit, 6 for display, 7 for stack count and 8 for destroy stack.

Step 10: Call create ()

Step 11: Using while and choice from the user

Step 12: Using Switch case, case 1 for read data and call push () case 2 for call pop (),
         case 3 for if(top == NULL) then display no elements in the stack else call
         topelement () and display top element, case 4 for call case 5 for exit (), case 6
         for call display () case 7 for Call stack_count case 8 for call destroy () and
         when the user enter a invalid entry display wrong choice. Please enter correct
         choice

Step 13: Define create () and assign top=NULL

Step 14: Define stack_count () and display no: of elements in the stack.

Step 15: Define push (), if (top == NULL) then allocate a memory space for top
         element and top->ptr = NULL, top->top = data else, allocate a memory space
         for new element temp -> ptr = top temp -> info = data. And increment
         element count

Step 16: Define display () if (top 1 = NULL) then display stack is empty. Otherwise
         display elements.
```

Step 17: Define pop (), if (top1 == NULL) then display an error message, otherwise pop the element and free up the stack.

Step 18: Define top element () and return top element of the stack.

Step 19: Define empty(), if (top ==NULL) then display stack is empty. Otherwise display the no: of stack elements.

Step 20: Define destroy ()

Step 21: Stop.

Input:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node*ptr;

}

*top,*top1,*temp;

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stackcount();

void create();

int count=0;

void main()
```

```
{
int no,ch,e;
clrscr();
printf("\n 1.push");
printf("\n 2.pop");
printf("\n 3.top");
printf("\n 4.Empty");
printf("\n 5.exit");
printf("\n 6.display");
printf("\n 7.stack count");
printf("\n 8.destroy");
create();
while(1)
{
printf("\n Enter choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("Enter data");
scanf("%d",&no);
push(no);
break;
case 2:
pop();
```

```
break;
case 3:
if(top==NULL)
printf("\n No elements in stack");
else
{
e=topelement();
printf("\n Top element:%d",e);
}
break;
case 4:
empty();
break;
case 5:
exit(0);
case 6:
display();
break;
case 7:
stackcount();
break;
case 8:
destroy();
break;
default:
```

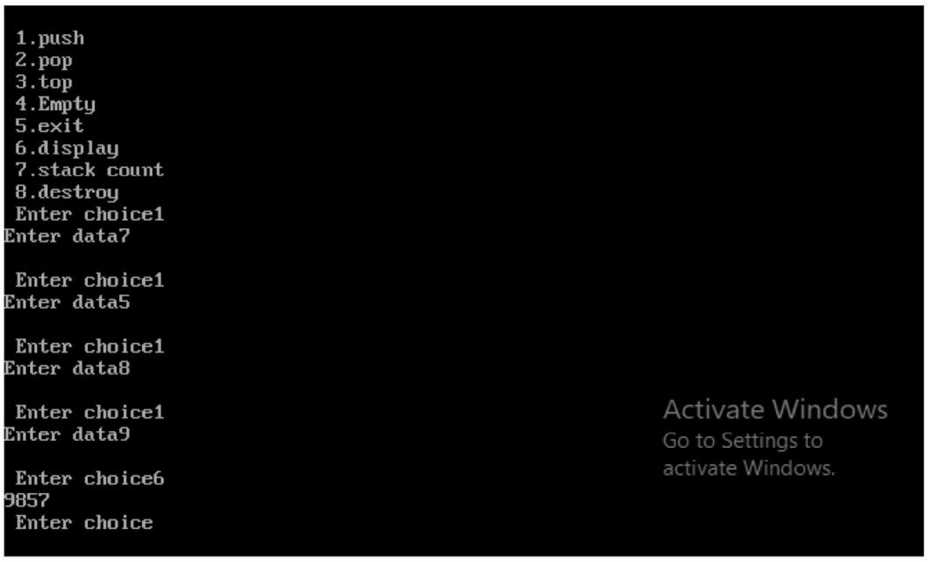
```
printf("\n Wrong choice,please select a valid choice");
break;
}
}
}
void create()
{
top=NULL;
}
void stackcount()
{
printf("\n number of elements in stack:%d",count);
}
void push(int data)
{
if(top==NULL)
{
top=(struct node*)malloc(1*sizeof(struct node));
top->ptr=NULL;
top->info=data;
}
else
{
temp=(struct node*)malloc(1*sizeof(struct node));
temp->ptr=top;
```

```
temp->info=data;
top=temp;
}
count++;
}
void display()
{
top1=top;
if(top1==NULL)
{
printf("\n stack is empty");
return;
}
while(top1!=NULL)
{
printf("%d",top1->info);
top1=top1->ptr;
}
}
void pop()
{
top1=top;
if(top1==NULL)
{
printf("\n error:trying to pop from empty stack");
```

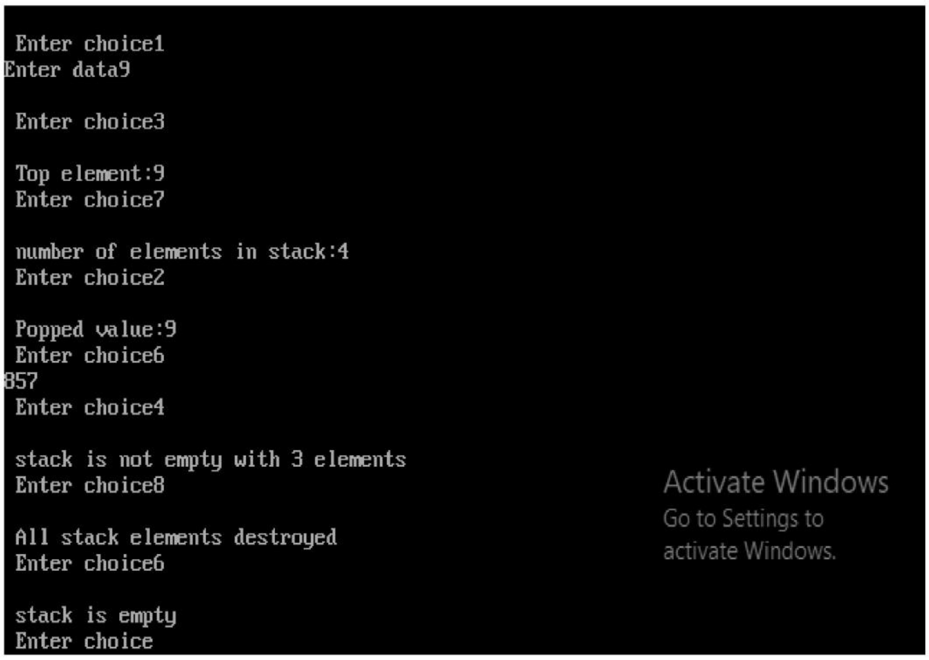
```
return;
}
else
{
top1=top1->ptr;
printf("\n Popped value:%d",top->info);
free(top);
top=top1;
count--;
}
}
int topelement()
{
return(top->info);
}
void empty()
{
if(top==NULL)
printf("\n stack is empty");
else
printf("\n stack is not empty with %d elements",count);
}
void destroy()
{
top1=top;
```

```
while(top1!=NULL)

{

top1=top->ptr;

free(top);

top=top1;

top1=top1->ptr;

}

free(top1);

top=NULL;

printf("\n All stack elements destroyed");

count=0;

}
```

Output:



```
 1.push
 2.pop
 3.top
 4.Empty
 5.exit
 6.display
 7.stack count
 8.destroy
 Enter choice1
Enter data7

 Enter choice1
Enter data5

 Enter choice1
Enter data8

 Enter choice1
Enter data9

 Enter choice6
9857
 Enter choice
```



```
 Enter choice1
Enter data9

 Enter choice3

 Top element:9
 Enter choice7

 number of elements in stack:4
 Enter choice2

 Popped value:9
 Enter choice6
857
 Enter choice4

 stack is not empty with 3 elements
 Enter choice8

 All stack elements destroyed
 Enter choice6

 stack is empty
 Enter choice
```

Result: The program is executed successfully and output is verified.

PROGRAM-5

Title: BINARY SEARCH TREE

Objective: Write a program for binary search tree.

Algorithm:

```
Step 1: Start
Step 2: Define structure of a node.
Step 3: Define structure of a new node. In this we allocate a memory space for new
        node. temp->left, temp->right=NULL
Step 4: Define a function for inorder traversal if(root!=NULL) then traverse the left
        subtree of the root and print root->key. After traversing the left subtree
        traverse the right subtree of the root.
Step 5: Define a structure for insertion operation if(node == NULL) then return the
        new node. if(key<node->key),then insert the new node to the left of that node
        else, if (key>node->key), insert the new node to the right of that node.
Step 6: Insert 50 to the root node.
Step 7: Insert 30, 20, 40, 70 ,60 and 80 into the right positions in the tree.
Step 8: Inorder(root)
Step 9: Stop
```

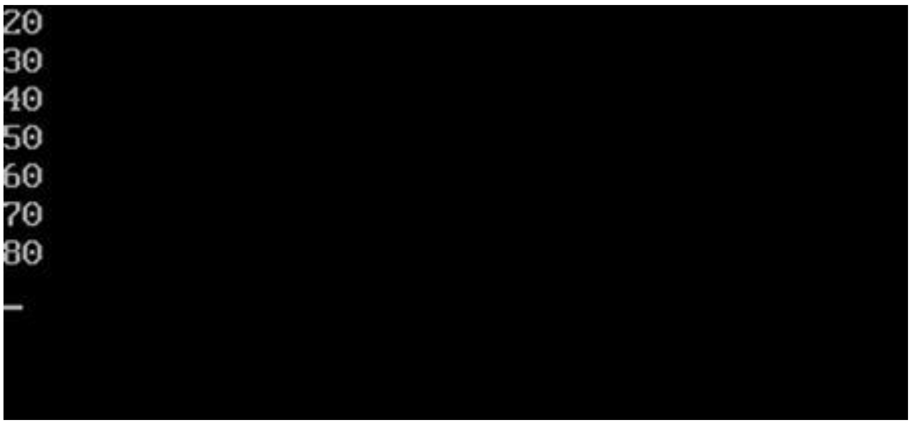
Input:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node*left,*right;

};

struct node*newnode(int item)
```

```
{
struct node*temp=(struct node*)malloc(sizeof(struct node));
temp->key=item;
temp->left=temp->right=NULL;
return temp;
}
void inorder(struct node*root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\n",root->key);
inorder(root->right);
}
}
struct node*insert(struct node*node,int key)
{
if(node==NULL)
return newnode(key);
if(key<node->key)
node->left=insert(node->left,key);
else if(key>node->key)
node->right=insert(node->right,key);
return node;
}
```

```
int main()

{

struct node*root=NULL;

clrscr();

root=insert(root,50);

insert(root,30);

insert(root,20);

insert(root,40);

insert(root,70);

insert(root,60);

insert(root,80);

inorder(root);

getch();

return 0;

}
```

Output:



```
20
30
40
50
60
70
80

_
```

Result: The program is executed successfully and output is verified.

PROGRAM-6

Title: SET DATA STRUCTURE AND OPERATIONS

Objective: Write a program to implement a set data structure and set operations using bit string.

Algorithm:

```
Step1: Start
Step 2: Declaration of union and intersection functions.
Step 3: Variables used a[10], b[10], m, n, i ,j, ch
Step 4: Read the number of elements in first set.
Step 5: Enter the elements of first set.
Step 6: Display the elements of first set.
Step 7: Read the number of element in second set.
Step 8: Enter the elements of second set.
Step 9: Display the elements of second set.
Step 10: Display a menu driven list, 1 for union, 2 for intersection and 3 for exit.
Step 11: Read the user choice.
Step 12: Using switch case, in case1-calling the union function, in case 2-calling the
         intersection function and case3- for exit()
Step 13: Define the union function it declare some variable such as c[20], i, j and set
         k=o and flag=0
Step 14: Using for loop we write the code for union operation. It checks the both set
         and take all the elements from both sets.
Step 15: Display the element of resultant set.
Step 16: Define the intersection function, it declare some variable such as c[20], i, j
         and set k=0 and flag=0 using for loops we can write the code for intersection
         operation it checks the both set and take common elements from both sets.
Step 17: If (k==0) then display the resultant set is null set.
```

Step 18: Else display the resultant set after intersection operation.

Step 19: Stop

Input:

```
#include<stdio.h>
#include<stdlib.h>
void Union(int set1[10],int set2[10],int m,int n);
void intersection(int set1[10],int set2[10],int m,int n);
void main()
{
int a[10],b[10],m,n,j,i;
int ch;
clrscr();
printf("\n Enter the number of elements in first set:");
scanf("%d",&m);
printf("\n Enter the elements:");
for(i=0;i<m;i++)
{
scanf("%d",&a[i]);
}
printf("\n elements of first set");
for(i=0;i<m;i++)
{
printf("\t %d",a[i]);
}
printf("\n Enter the number of elements in second set:");
```

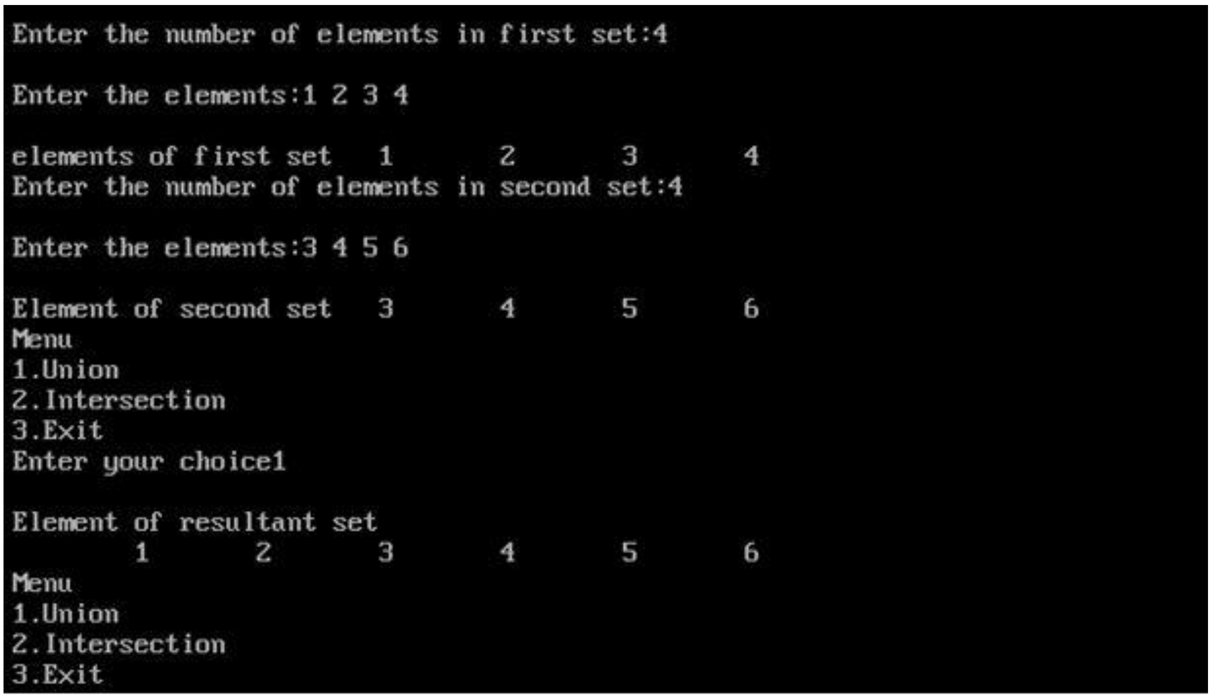
```
scanf("%d",&n);
printf("\n Enter the elements:");
for(i=0;i<n;i++)
{
scanf("\t %d",&b[i]);
}
printf("\n Element of second set");
for(i=0;i<n;i++)
{
printf("\t %d",b[i]);
}
for(;;)
{
printf("\n Menu \n 1.Union \n 2.Intersection \n 3.Exit");
printf("\n Enter your choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
Union(a,b,m,n);
break;
case 2:
intersection(a,b,m,n);
break;
case 3:
```

```
exit(0);
}
}
}
void Union(int a[10],int b[10],int m,int n)
{
int c[20],i,j,k=0,flag=0;
for(i=0;i<m;i++)
{
c[k]=a[i];
k++;
}
for(i=0;i<n;i++)
{
flag=0;
for(j=0;j<m;j++)
{
if(b[i]==c[j])
{
flag=1;
break;
}
}
if(flag==0)
{
```

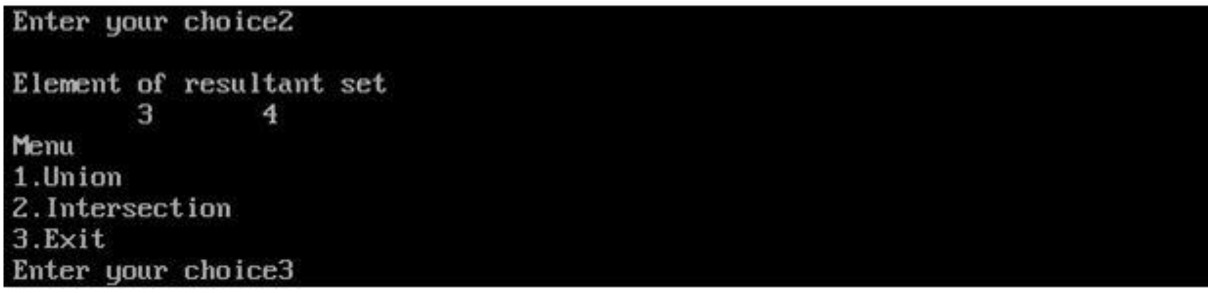
```
c[k]=b[i];
k++;
}
}
printf("\n Element of resultant set \n");
for(i=0;i<k;i++)
{
printf("\t %d",c[i]);
}
}
void intersection(int a[10],int b[10],int m,int n)
{
int c[20],i,j,k=0,flag=0;
for(i=0;i<m;i++)
{
flag=0;
for(j=0;j<n;j++)
{
if(a[i]==b[j])
{
flag=1;
break;
}
}
if(flag==1)
```

```
{c[k]=a[i];
k++;
}
}
if(k==0)
{
printf("\n Resultant set is null set\n");
}
else
{
printf("\n Element of resultant set\n");
for(i=0;i<k;i++)
{
printf("\t %d",c[i]);
}
}
}
```

Output:



```
Enter the number of elements in first set:4

Enter the elements:1 2 3 4

elements of first set   1       2       3       4
Enter the number of elements in second set:4

Enter the elements:3 4 5 6

Element of second set   3       4       5       6
Menu
1.Union
2.Intersection
3.Exit
Enter your choice1

Element of resultant set
        1       2       3       4       5       6
Menu
1.Union
2.Intersection
3.Exit
```



```
Enter your choice2

Element of resultant set
        3       4
Menu
1.Union
2.Intersection
3.Exit
Enter your choice3
```

Result: The program is executed successfully and output is verified.

PROGRAM-7

Title: RED BLACK TREE

Objective: Write a program for red black tree and its operation.

Algorithm:

```
Step 1: Start
Step 2: Define a structure called node which contain an integer elements called d and
        and a pointer to a structure of type node called struct *p,*r and *l.
Step 3: Insert new node(Root).
Step 4:  Do the coloring of parent if not black or it's not a root
Step 5: If uncle is red, change color of parent and uncle to black color of grand parent
        as red change grand parent, Repeat
Step 6: If uncle is black, left left case, left right case, right right case, right left case,
        change parent, repeat.
Step 7: Stop
```

Input:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

int d;

int c;

struct node*p;

struct node*r;

struct node*l;

};

struct node*root=NULL;
struct node*bst(struct node*trav,struct node*temp)
{
if(trav==NULL)
return temp;
if(temp->d<trav->d)
{
trav->l=bst(trav->l,temp);
trav->l->p=trav;
}
else if(temp->d>trav->d)
{
trav->r=bst(trav->r,temp);
trav->r=trav;
}
return trav;
}
void rightrotate(struct node*temp)
{
struct node*left=temp->l;
temp->l=left->r;
if(temp->l)
temp->l->p=temp;
left->p=temp->p;
if(!temp->p)
root=left;
else if(temp==temp->p->l)
```

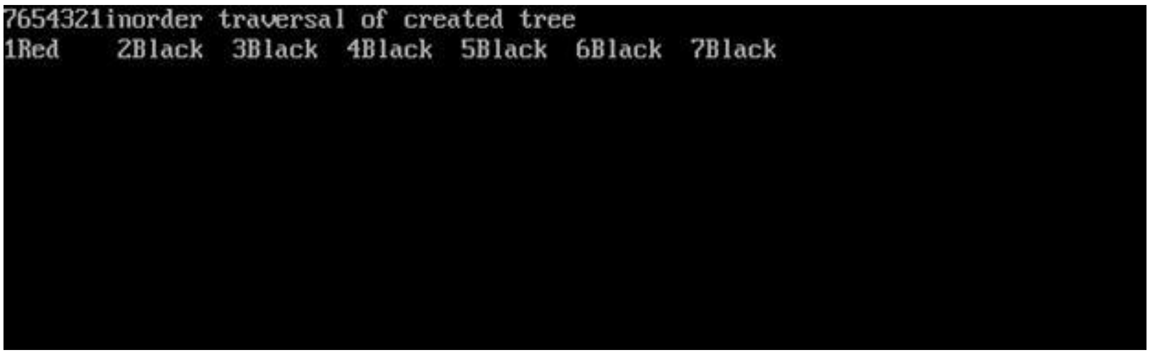
```
temp->p->l=left;
else
temp->p=left;
left->r=temp;
temp->p=left;
}
void leftrotate(struct node*temp)
{
struct node*right=temp->r;
temp->r=right->l;
if(temp->r)
temp->r->p=temp;
right->p=temp;
right->p=temp->p;
if(!temp->p)
root=right;
else if(temp==temp->p->l)
temp->p->l=right;
else
temp->p->r=right;
right->l=temp;
temp->p=right;
}
void fixup(struct node*root,struct node*pt)
{
struct node*parent_pt=NULL;
struct node*grand_parent_pt=NULL;
int t;
while((pt!=root)&&(pt->c!=0)&&(pt->p->c==1))
{
parent_pt=pt->p;
grand_parent_pt=pt->p->p;
if(parent_pt==grand_parent_pt->l)
{
struct node*uncle_pt=grand_parent_pt->r;
if(uncle_pt!=NULL&&uncle_pt->c==1)
{
grand_parent_pt->c=1;
parent_pt->c=0;
uncle_pt->c=0;
pt=grand_parent_pt;
}
else
{
if(pt==parent_pt->r)
{
leftrotate(parent_pt);
pt=parent_pt;
parent_pt=pt->p;
}
rightrotate(grand_parent_pt);
t=parent_pt->c;
```

```
parent_pt->c=grand_parent_pt->c;
grand_parent_pt->c=t;
pt=parent_pt;
}
}
else
{
struct node*uncle_pt=grand_parent_pt->l;
if((uncle_pt!=NULL)&&(uncle_pt->c==1))
{
grand_parent_pt->c=1;
parent_pt->c=0;
uncle_pt->c=0;
pt=grand_parent_pt;
}
else
{
if(pt==parent_pt->l)
{
rightrotate(parent_pt);
pt=parent_pt;
parent_pt=pt->p;
}
leftrotate(grand_parent_pt);
t=parent_pt->c;
parent_pt->c=grand_parent_pt->c;
```

```
grand_parent_pt->c=t;
pt=parent_pt;
}
}
}
root->c=0;
}
void inorder(struct node*trav)
{
if(trav==NULL)
return;
inorder(trav->l);
printf("%d",trav->d);
if(trav->c==0)
printf("Black \t");
else
printf("Red \t");
inorder(trav->r);
}
int main()
{
int n=7;
int a[7]={7,6,5,4,3,2,1};
int i;
clrscr();
for(i=0;i<n;i++)
```

```
{struct node*temp=(struct node*)malloc(sizeof(struct node));
temp->r=NULL;
temp->l=NULL;
temp->p=NULL;
temp->d=a[i];
printf("%d",temp->d);
temp->c=1;
root=bst(root,temp);
fixup(root,temp);
}
printf("inorder traversal of created tree \n");
inorder(root);
getch();
return 0;
}
```

Output:



```
7654321inorder traversal of created tree
1Red    2Black  3Black  4Black  5Black  6Black  7Black
```

Result: The program is executed successfully and output is verified.

PROGRAM-8

Title: FIBONACCI HEAP

Objective: Write a program for Fibonacci heap and its operation.

Algorithm:

```
Step 1: Start
Step 2: Create new node for the element.
Step 3: Check if the leap is empty.
Step 4: If the leap is empty, set the new node as a root and mark it.
Step 5: Else, insert the node into the root list and update final min. The min element is
        always given min pointer.
Step 6: Concatenate the root of both leap .
Step 7: Update min by selecting a min key and from the new root list.
Step 8: Delete the min node.
Step 9: Set the min pointer to the next root list.
Step 10: Create an array of size equal to the next degree of the tree in the leap before
        deletion.
Step 11: Do the following until there are no multiple root with the same degree.
Step 12: Map the degree of current root (min) to the degree in the array.
Step 13: Map the degree of next root to the degree in array.
Step 14: If there are more than two mapping for the same degree, then apply union
         min at the root.
Step 15: Stop
```

Input:

```
#include <math.h>

#include <stdio.h>

#include<conio.h>

#include <stdlib.h>


struct NODE {

   int key;

   int degree;

   struct NODE *left_sibling;

   struct NODE *right_sibling;

   struct NODE *parent;

   struct NODE *child;

   int mark;

   int visited;

};


struct FIB_HEAP {

   int n;

   struct NODE *min;

   int phi;

   int degree;

};


struct FIB_HEAP *make_fib_heap();

void insertion(struct FIB_HEAP *H, int val);
```

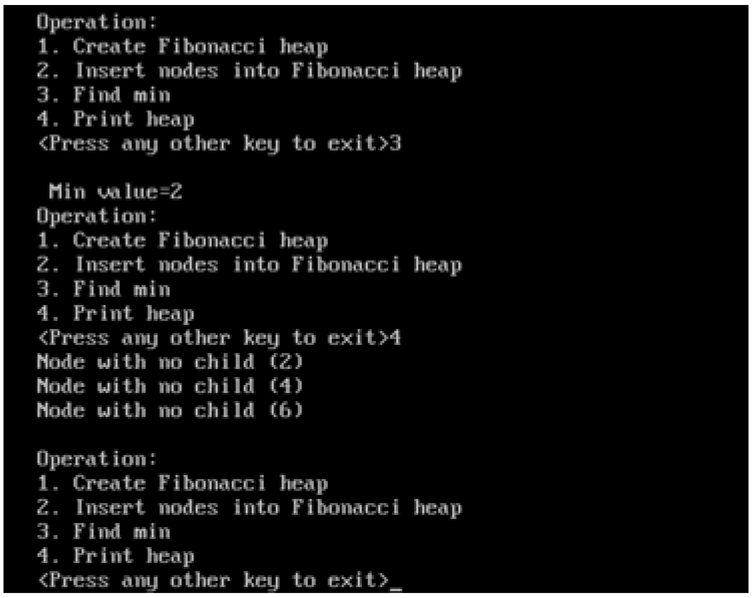
```
struct NODE *find_min_node(struct FIB_HEAP *H);
void print_heap(struct NODE *n);


struct FIB_HEAP *make_fib_heap()
{
   struct FIB_HEAP *H = (struct FIB_HEAP *)malloc(sizeof(struct FIB_HEAP));
   H->n = 0;
   H->min = NULL;
   H->phi = 0;
   H->degree = 0;
   return H;
}


void print_heap(struct NODE *n)
{
   struct NODE *x;
   for (x = n;; x = x->right_sibling) {
            if (x->child == NULL) {
               printf("Node with no child (%d)\n", x->key);
            } else {
               printf("Node (%d) with child (%d)\n", x->key, x->child->key);
               print_heap(x->child);
            }
            if (x->right_sibling == n) {
               break;
            }
   }
```

```
}


void insertion(struct FIB_HEAP *H, int val)
{
   struct NODE *new = (struct NODE *)malloc(sizeof(struct NODE));
   new->key = val;
   new->degree = 0;
   new->mark = 0;
   new->parent = NULL;
   new->child = NULL;
   new->visited = 0;
   new->left_sibling = new;
   new->right_sibling = new;

   if (H->min == NULL) {
           H->min = new;
   } else {
           H->min->left_sibling->right_sibling = new;
           new->right_sibling = H->min;
           new->left_sibling = H->min->left_sibling;
           H->min->left_sibling = new;

           if (new->key < H->min->key) {
              H->min = new;
           }
   }
   (H->n)++;
```

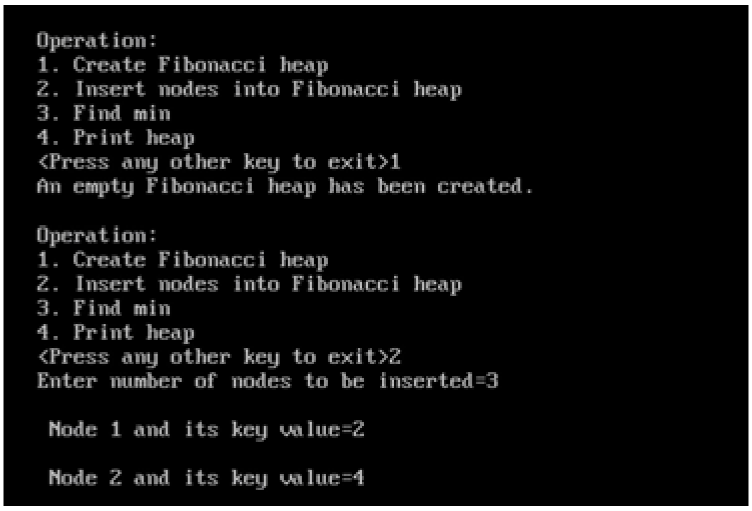
```
}


struct NODE *find_min_node(struct FIB_HEAP *H)

{

   if (H == NULL) {

            printf("\nFibonacci heap not yet created.\n");

            return NULL;

   } else {

            return H->min;

   }

}


int main()

{

   struct FIB_HEAP *heap = NULL;

   int operation_no, new_key, dec_key, ele, i, no_of_nodes;

   struct NODE *min_node;

   clrscr();


   while (1) {

            printf("\nOperation:\n1. Create Fibonacci heap\n2. Insert nodes into Fibonacci heap\n3.
Find min\n4. Print heap\n<Press any other key to exit>");

            scanf("%d", &operation_no);


            switch (operation_no) {

               case 1:

                        heap = make_fib_heap();

                        printf("An empty Fibonacci heap has been created.\n");
```

```
        break;
    case 2:
        if (heap == NULL) {
            heap = make_fib_heap();
        }
        printf("Enter number of nodes to be inserted=");
        scanf("%d", &no_of_nodes);
        for (i = 1; i <= no_of_nodes; i++) {
            printf("\n Node %d and its key value=", i);
            scanf("%d", &ele);
            insertion(heap, ele);
        }
        break;
    case 3:
        min_node = find_min_node(heap);
        if (min_node == NULL) {
            printf("No minimum value");
        } else {
            printf("\n Min value=%d", min_node->key);
        }
        break;
    case 4:
        print_heap(heap->min);
        break;
    default:
        printf("Invalid choice");
        exit(0);
        }
   }
return 0;
}
```

Output:



```
Operation:
1. Create Fibonacci heap
2. Insert nodes into Fibonacci heap
3. Find min
4. Print heap
<Press any other key to exit>3

 Min value=2
Operation:
1. Create Fibonacci heap
2. Insert nodes into Fibonacci heap
3. Find min
4. Print heap
<Press any other key to exit>4
Node with no child (2)
Node with no child (4)
Node with no child (6)

Operation:
1. Create Fibonacci heap
2. Insert nodes into Fibonacci heap
3. Find min
4. Print heap
<Press any other key to exit>_
```



```
Operation:
1. Create Fibonacci heap
2. Insert nodes into Fibonacci heap
3. Find min
4. Print heap
<Press any other key to exit>1
An empty Fibonacci heap has been created.

Operation:
1. Create Fibonacci heap
2. Insert nodes into Fibonacci heap
3. Find min
4. Print heap
<Press any other key to exit>2
Enter number of nodes to be inserted=3

 Node 1 and its key value=2

 Node 2 and its key value=4
```

Result: The program is executed successfully and output is verified.

PROGRAM-9

Title: BREADTH FIRST TRAVERSAL

Objective: Write a program for breadth first traversal of a graph.

Algorithm:

```
Step 1: Start
Step 2: Start with initial node.
Step 3: Mark the initial node as visited and enqueue it.
Step 4: While the queue is not empty, dequeue a node from the queue if the queued
        node is the goal node stop the search. Otherwise, enqueue any successors that
        have not yet been visited.
Step 5: If the queue is empty, every node on the graph has been visited or there is no
        path from the initial node to the goal node.
Step 6: If the node was found return the path that was followed.
Step 7: Stop.
```

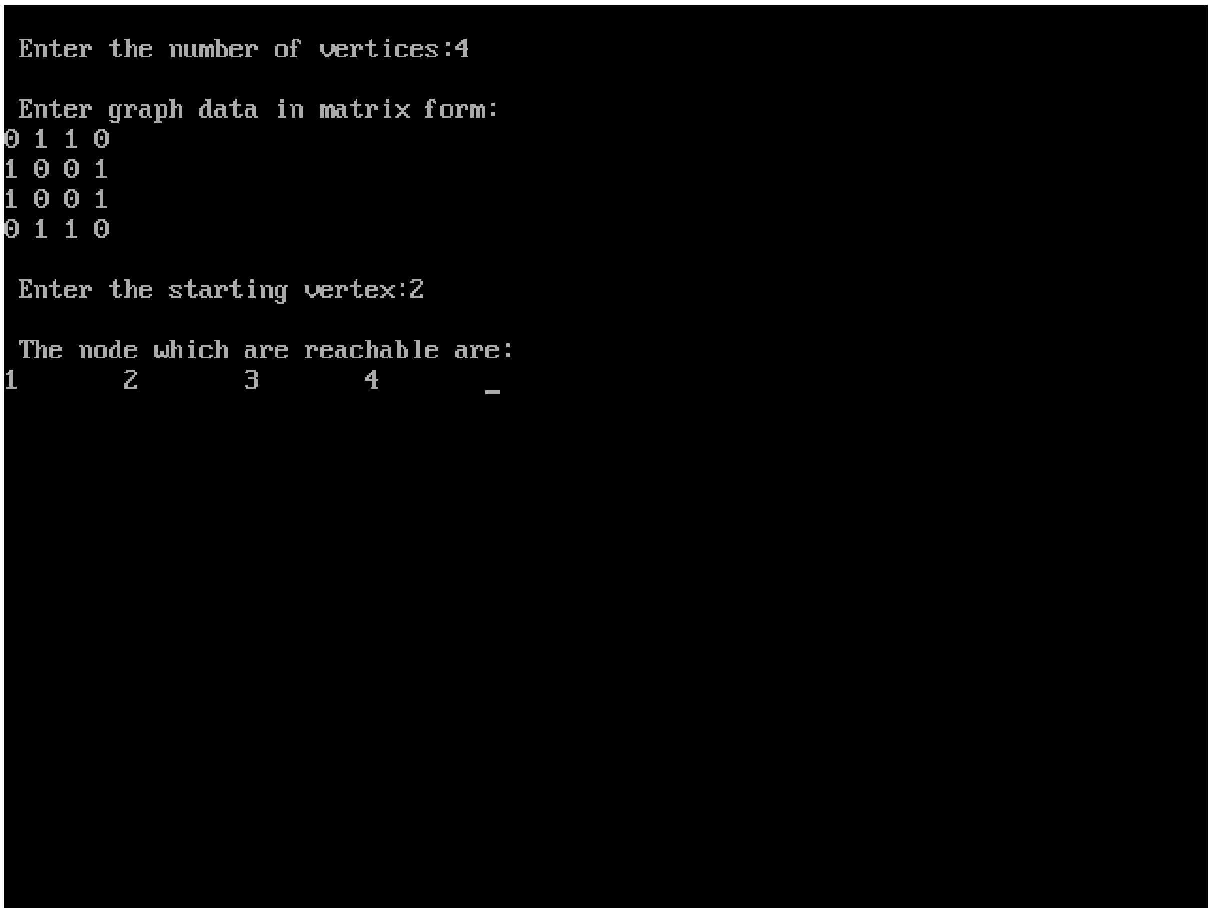
Input:

```
#include<stdio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
for(i=0;i<=n;i++)
if(a[v][i]&&!visited[i])
q[++r]=i;
if(f<=r)
{
visited[q[f]]=1;
bfs(q[f++]);
}
```

```
}
void main()
{
int v;
clrscr();
printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<n;i++)
{
q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
{
```

```
if(visited[i]){
printf("%d\t",i);
}
else
{
printf("\n BFS is not possible.Not all nodes are reachable");
break;
}
}
getch();
}
```

Output:



```
 Enter the number of vertices:4

 Enter graph data in matrix form:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0

 Enter the starting vertex:2

 The node which are reachable are:
1       2       3       4       _
```

Result: The program is executed successfully and output is verified.

PROGRAM-10

Title: DEPTH FIRST TRAVERSAL

Objective: Write a program for depth first traversal graph.

Algorithm:

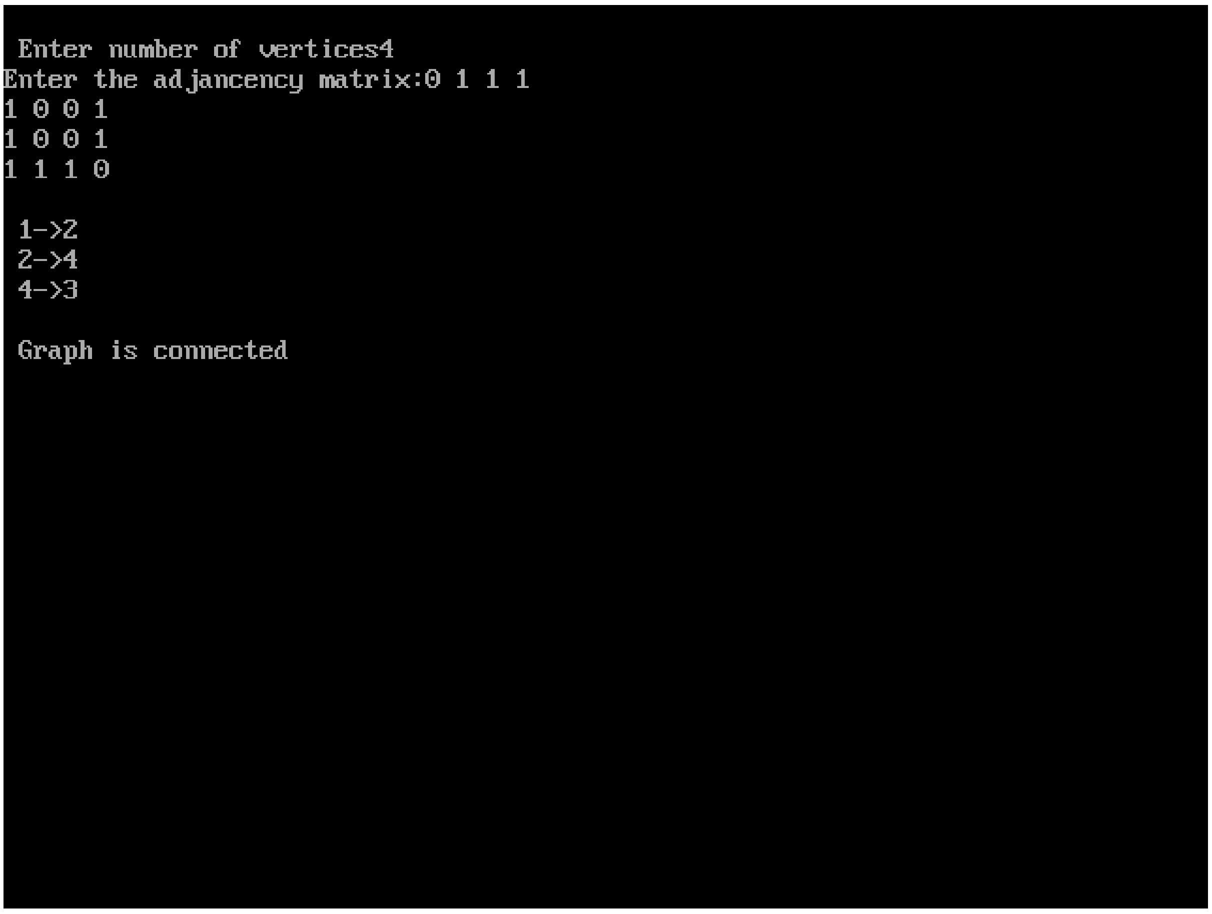
```
Step 1: Start
Step 2: Create a stack with the total number of vertices in the graph.
Step 3: Choose any vertex as the starting point of traversal and put the vertex into the
        stack.
Step 4: PUSH a non-visited vertex to the top of the stack.
Step 5: Repeat steps 3 and 4 until no vertices are left to visit from the vertex on the
        stack's top.
Step 6: If no vertex is left go back and pop a vertex from the stack.
Step 7: Repeat steps 2,3 and 4 until stack is empty.
Step 8: Stop
```

Input:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
int i;
reach[v]=1;
for(i=1;i<=n;i++)
if(a[v][i]&&!reach[i])
{
printf("\n %d->%d",v,i);
dfs(i);
}
}
void main()
{
int i,j,count=0;
clrscr();
printf("\n Enter number of vertices");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
reach[i]=0;
for(j=1;j<=n;j++)
a[i][j]=0;
}
printf("Enter the adjancency matrix:");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
dfs(1);
printf("\n");
for(i=1;i<=n;i++)
{
if(reach[i])
count++;
}
```

```
if(count==n)

printf("\n Graph is connected");

else

printf("\n Graph is not connected");

getch();

}
```

Output:



```
 Enter number of vertices4
Enter the adjancency matrix:0 1 1 1
1 0 0 1
1 0 0 1
1 1 1 0

 1->2
 2->4
 4->3

 Graph is connected
```

Result: The program is executed successfully and output is verified.

PROGRAM-11

Title: TOPOLOGICAL SORTING

Objective: Write a program for topological sorting.

Algorithm:

```
Step 1: Start
Step 2: Start from a vertex, we first print it and then recursively call DFS for its
        adjacent vertices.
Step 3: Recursively call topological sorting for all its adjacent vertices, then push it to
        a stack and print the content.
Step 4: Initialize visited array of size N to keep the record of visited nodes.
Step 5: Run a loop from 0 till N.
Step 6: Print all the elements in the stack.
Step 7: Stop
```

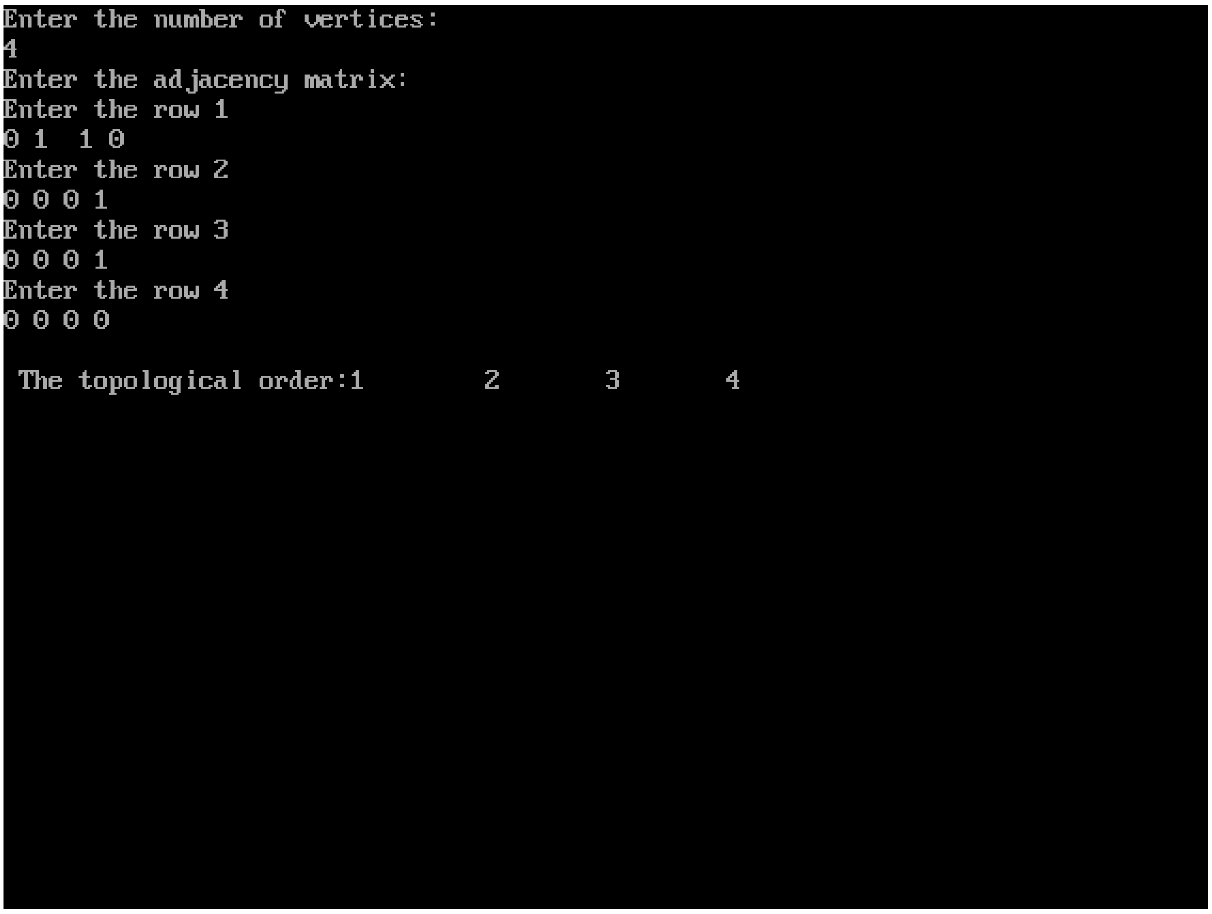
Input:

```
#include<stdio.h>
int main()
{
int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;
clrscr();
printf("Enter the number of vertices:\n");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(i=0;i<n;i++)
{
printf("Enter the row %d\n",i+1);
for(j=0;j<n;j++)
```

```
scanf("%d",&a[i][j]);
}
for(i=0;i<n;i++)
{
indeg[i]=0;
flag[i]=0;
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
indeg[i]=indeg[i]+a[j][i];
printf("\n The topological order:");
while(count<n)
{
for(k=0;k<n;k++)
{
if((indeg[k]==0)&&(flag[k]==0))
{
printf("%d\t",(k+1));
flag[k]=1;
}
for(i=0;i<n;i++)
{
if(a[i][k]==1)
indeg[k]--;
}
}
```

```
        count++;

        }

        getch();

        return 0;

        }
```

Output:



```
Enter the number of vertices:
4
Enter the adjacency matrix:
Enter the row 1
0 1  1 0
Enter the row 2
0 0 0 1
Enter the row 3
0 0 0 1
Enter the row 4
0 0 0 0

 The topological order:1        2       3       4
```

Result: The program is executed successfully and output is verified.

PROGRAM-12

Title: PRIM’S ALGORITHM

Objective: Write a program for prim’s algorithm implementation.

Algorithm:

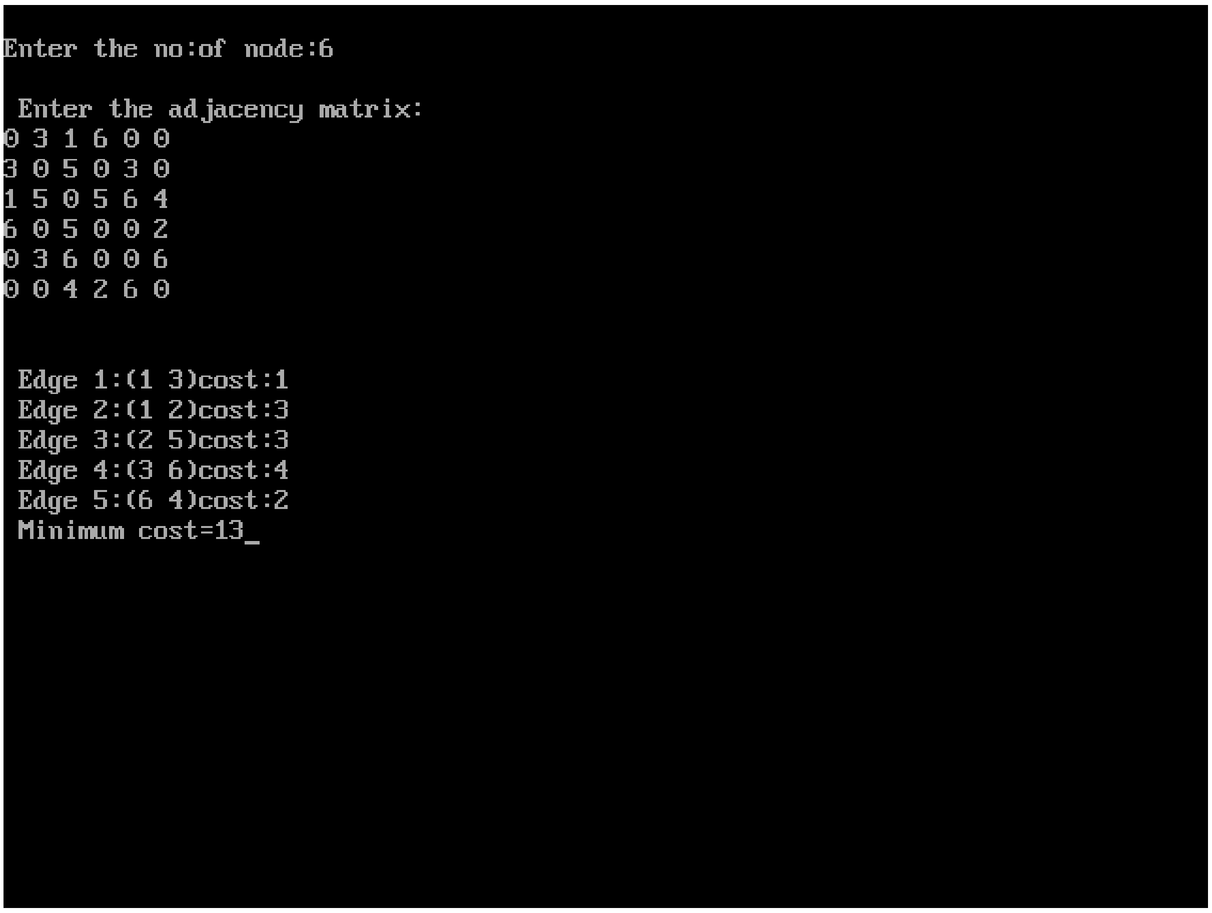
```
Step 1: Start
Step 2: variables used cost[10][10],a, b, u, v, n, i, j, min and initialize ne=1,
        visited[10]={0} and mincost=0
Step 3: Read the number of nodes from the user.
Step 4: Read the adjacency matrix and if cost[i][j]=0 then set cost[i][j]=999.
Step 5: Check the condition,i.e., number of edges is less than n . If (cost [i][j]<min)
        and if (visited[i]!=0) then min=cost[i][j]. Assign i=u and j=v if (visited[u]==0
        and visited[v]==0) then display the edge with cost 0 and calculate the
        minimum cost.
Step 6: Display the minimum cost.
Step 7: Stop.
```

Input:

```
#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0;

cost[10][10];

void main()

{

clrscr();

printf("\nEnter the no:of node:");

scanf("%d",&n);

printf("\n Enter the adjacency matrix:");
```

```
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
visited[1]=1;
printf("\n");
while(ne<n)
{
for(i=1,min=999;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]<min)
if(visited[i]!=0)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
{
 printf("\n Edge %d:(%d %d)cost:%d",ne++,a,b,min);
 mincost+=min;
 visited[b]=1;
 }
cost[a][b]=cost[b][a]=999;
}
printf("\n Minimum cost=%d",mincost);
getch();
}
```

Output:



```
Enter the no:of node:6

 Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0


 Edge 1:(1 3)cost:1
 Edge 2:(1 2)cost:3
 Edge 3:(2 5)cost:3
 Edge 4:(3 6)cost:4
 Edge 5:(6 4)cost:2
 Minimum cost=13_
```

Result: The program is executed successfully and output is verified.

PROGRAM-13

Title: KRUSKAL’S ALGORITHM

Objective: Write a program for kruskal’s algorithm implementation.

Algorithm:

```
Step 1: Start
Step 2: Variables used i,j,k,a,b,u,v,n,min,cost[9][9] parent [9] and initialize ne=1 and
        mincost=0
Step 3: Declare find() and uni() functions.
Step 4: Print “implementation of kriskal’s algorithm”.
Step 5: Read the number of vertices from the user.
Step 6: Read the cost adjacency matrix if(cost[i][j]==0) then set cost[i][j]=999.
Step 7: Display “ The edge of minimum cost spanning tree”. Check the condition that
        is number of edges(ne) less than n. If(cost[i][j]<min) then assign
        min=cost[i][j] and set u=I and v=j.
Step 8: Call find(u) and find(v) fn.
Step 9: Display the edges with cost and calculate the minimum cost and display it.
Step 10: Define find() fn.
Step 11: Define uni() function.
Step 12: Stop.
```

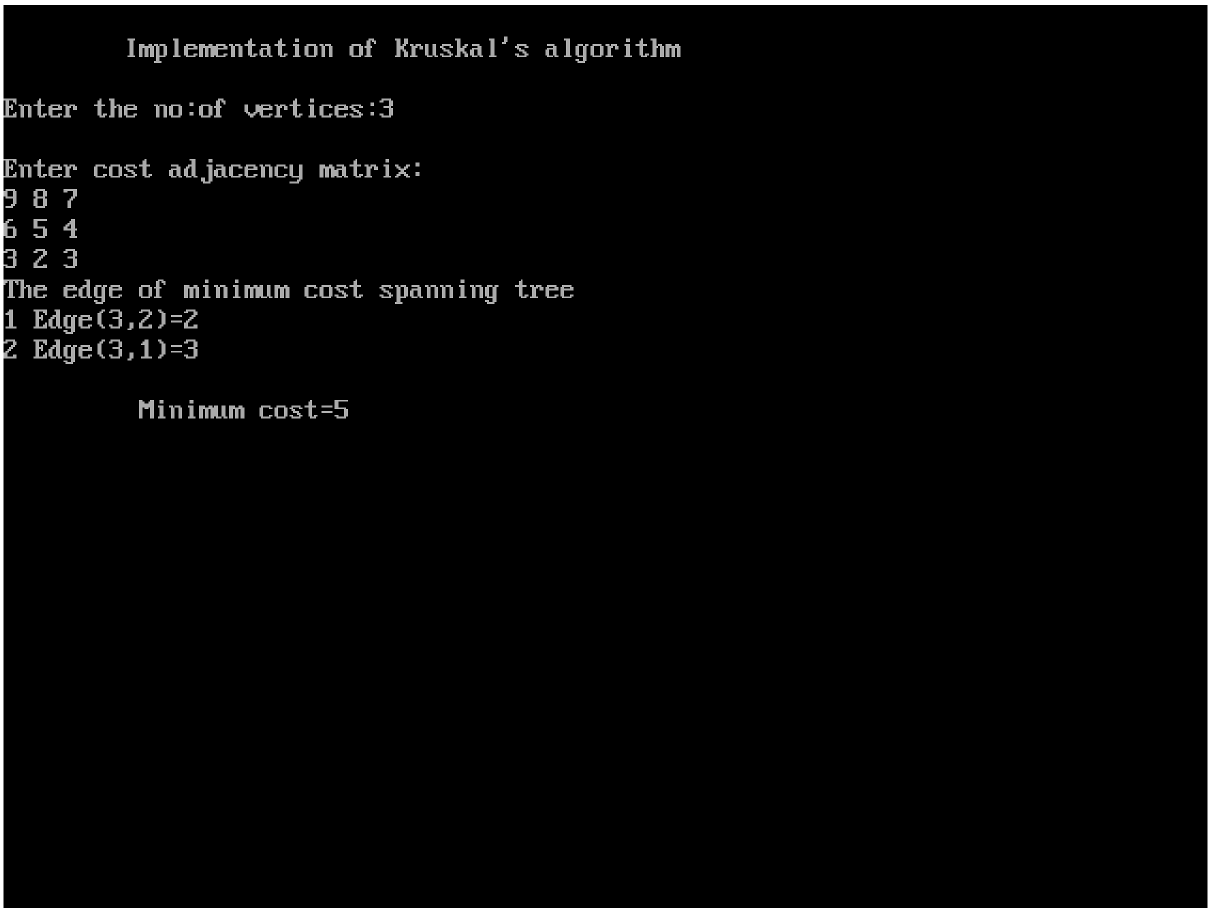
Input:

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);
```

```
void main()
{
clrscr();
printf("\n \tImplementation of Kruskal's algorithm\n");
printf("\nEnter the no:of vertices:");
scanf("%d",&n);
printf("\nEnter cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("The edge of minimum cost spanning tree\n");
while(ne<n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(cost[i][j]<min)
 {
min=cost[i][j];
a=u=i;
b=v=j;
}
}
}
u=find(u);
v=find(v);
if(uni(u,v))
{
printf("%d Edge(%d,%d)=%d\n",ne++,a,b,min);
mincost+=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n \t Minimum cost=%d\n",mincost);
getch();
}

int find(int i)
{
while(parent[i])
i=parent[i];
return i;
}
int uni(int i,int j)
{
```

```
if(i!=j)

{


parent[j]=i;

return 1;

}

return 0;

}
```

Output:



```
        Implementation of Kruskal's algorithm

Enter the no:of vertices:3

Enter cost adjacency matrix:
9 8 7
6 5 4
3 2 3
The edge of minimum cost spanning tree
1 Edge(3,2)=2
2 Edge(3,1)=3

         Minimum cost=5
```

Result: The program is executed successfully and output is verified.

PROGRAM-14

Title: SINGLE SOURCE SHORTEST PATH ALGORITHM

Objective: Write a program for single source shortest path algorithm implementation.

Algorithm:

```
Step 1 : Create a set shortPath to store vertices that come in the way of the shortest
         path tree.
Step 2 : Initialize all distance values as INFINITE and assign distance values as 0 for
         source vertex so that it is picked first.
Step 3 : Loop until all vertices of the graph are in the shor tpath.
    Step 3.1: Take a new vertex that is not visited and is nearest.
    Step 3.2: Add this vertex to short Path.
    Step 3.3: For all adjacent vertices of this vertex update distances. Now check every
              adjacent vertex of V, if sum of distance of u and weight of edge is else the
              update it.
Step 4: Stop
```

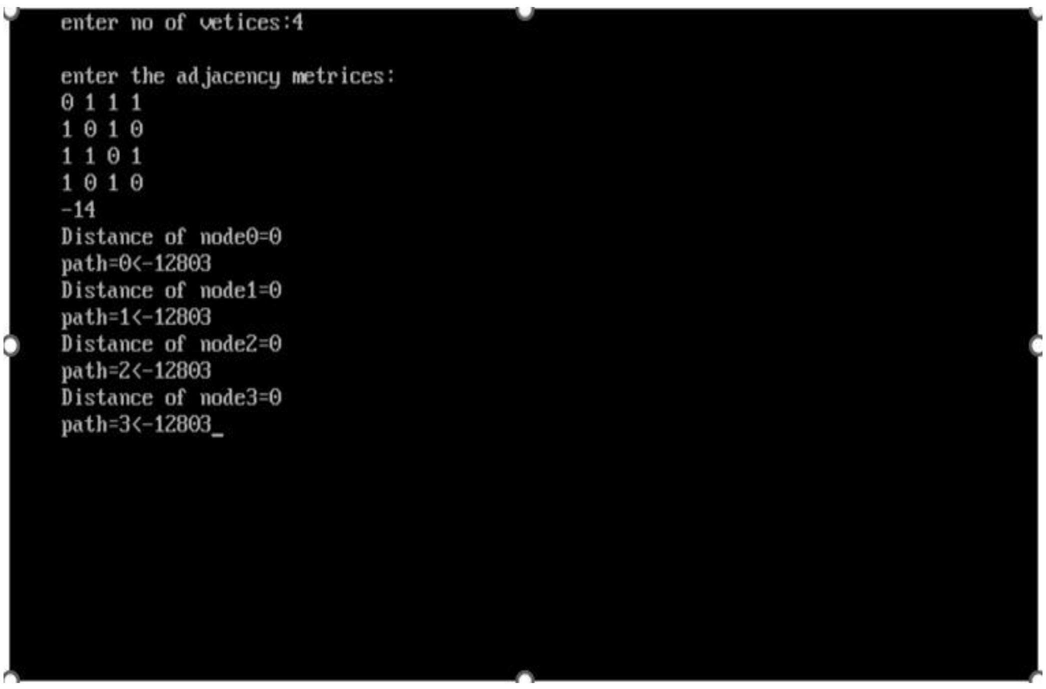
Input:

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

clrscr();

printf("enter no of vetices:");

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

```
printf("\nenter the adjacency metrices:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("%d",&u);
dijkstra(G,n,u);
return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
```

```
count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i]);
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\npath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}
while(j!=startnode);
}
getch();
}
```

Output:



```
enter no of vetices:4

enter the adjacency metrices:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
-14
Distance of node0=0
path=0<-12803
Distance of node1=0
path=1<-12803
Distance of node2=0
path=2<-12803
Distance of node3=0
path=3<-12803_
```

Result: The program is executed successfully and output is verified.