**Program No: 1**

**Aim: Write a program to search a number linerally**

#include<stdio.h>

#include<conio.h>

void main()

{

int a[100],n,num,i,c=0;

clrscr();

printf("\n\nLINEAR SEARCH");

printf("\nEnter The Array Limit: ");

scanf("%d",&n);

printf("\nEnter The Elements In Array: ");

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("\nEnter The Searching Element: ");

scanf("%d",&num);

for(i=0;i<n;i++)

{

if(a[i]==num)

{

printf("\n%d Is Found At %d Position.",num,i+1);

c++;

}

}

if(c==0)

{

printf("\nNumber Is Not Found. Enter Correctly..");

}

else

{

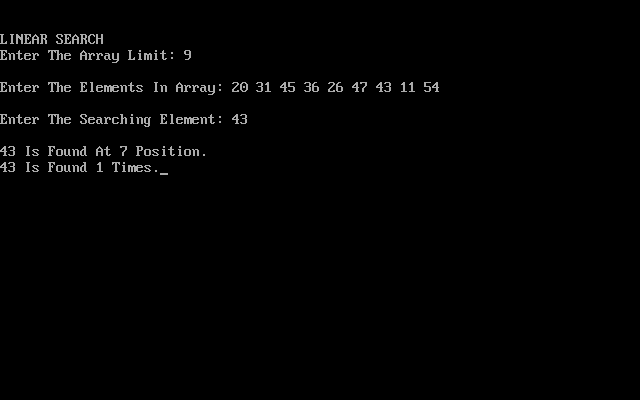
printf("\n%d Is Found %d Times.",num,c);

}

getch();

}

**Output**



**Program No: 2**

**Aim: Write A Program to search a number using binary search**

#include<stdio.h>

#include<conio.h>

void main()

{

int i, fi, la, mid, n, search, arr[100];

clrscr();

printf("\n\nBINARY SEARCH");

printf("\n\nENTER THE NUMBER OF ELEMENTS: ");

scanf("%d",&n);

printf("\nEnter %d ELEMENTS IN ASCENDING ORDER: ",n);

for (i=0;i<n;i++)

scanf("%d",&arr[i]);

printf("\nENTER THE NO TO FIND: ");

scanf("%d",&search);

fi=0;

la=n-1;

mid=(fi+la)/2;

while (fi <= la)

{

if (arr[mid] < search)

fi = mid + 1;

else if (arr[mid] == search)

{

printf("%d FOUND AT %d LOCATION.\n", search, mid+1);

break;

}

else

la = mid - 1;

mid = (fi + la)/2;

}

if (fi > la)

{

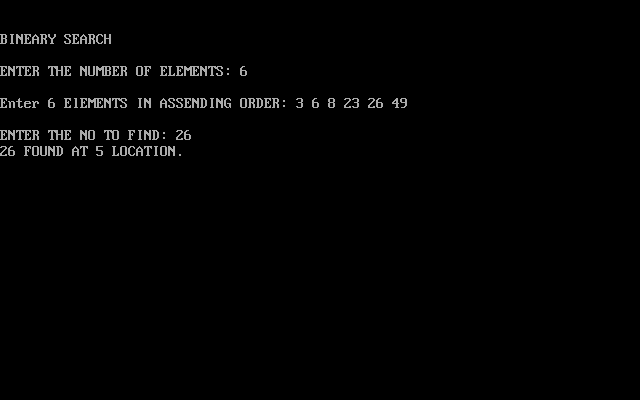
printf("Not found! %d is not present in the list.\n", search);

}

getch();

}

**OUTPUT-**



**Program No: 3**

**Aim: Write a program to sort an array using bubble sort**

#include<stdio.h>

#include<conio.h>

void main()

{

int a[100],n,i,j,temp;

clrscr();

printf("\n\nBUBBLE SORT...");

printf("\n\nENTER THE ARRAY LIMIT: ");

scanf("%d",&n);

printf("\nENTER %d ELEMENTS IN ARRAY: ",n);

for (i=0; i<n; i++)

scanf("%d",&a[i]);

for (i=0; i<(n-1); i++)

{

for (j=0; j<(n-i-1); j++)

{

if (a[j] > a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

printf("\nSORTED ARRAY IN ASCENDING ORDER: ");

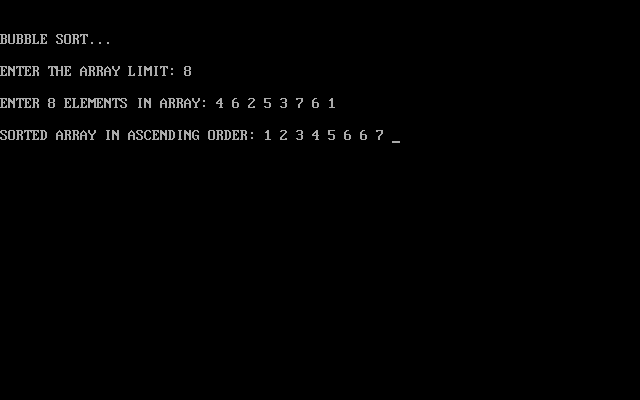
for (i=0; i<n; i++)

printf("%d ",a[i]);

getch();

}

**OUTPUT-**



**Program No: 4**

**Aim: Write A Program to sort an array using insertion sort**

#include <stdio.h>

#include<conio.h>

void main()

{

int n,a[1000],i,j,temp;

clrscr();

printf("\n\nINSERTION SORT...");

printf("\nENTER THE SIZE OF ARRAY: ");

scanf("%d",&n);

printf("\nEnter %d THE ELEMENTS IN ARRAY: ", n);

for (i=0; i<n; i++)

{

scanf("%d", &a[i]);

}

for (i=1; i<=(n- 1); i++)

{

j=i;

while ( j>0 && a[j]<a[j-1])

{

temp=a[j];

a[j]=a[j-1];

a[j-1]=temp;

j--;

}

}

printf("\nSORTED LIST IN ASCENDING ORDER: ");

for (i=0; i<=(n-1); i++)

{

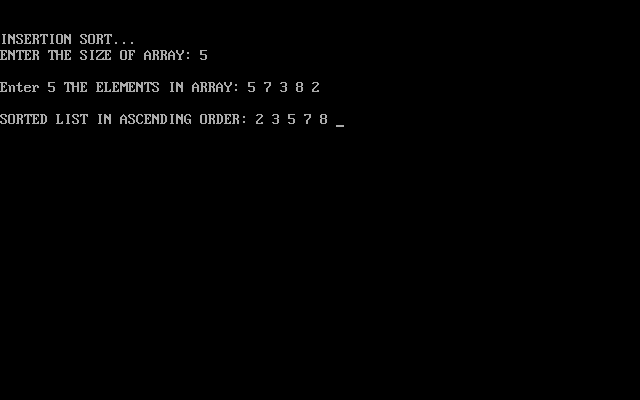
printf("%d ", a[i]);

}

getch();

}

**OUTPUT-**



**Program No: 5**

**Aim: Write A Program Of Merge Sort**

#include<stdio.h>

#include<conio.h>

void merge(int [ ], int , int , int );

void part(int [ ], int , int );

int main()

{

int arr[30], i, size;

clrscr();

printf("\n\nMERGE SORT METHOD...");

printf("\n\nENTER THE SIZE OF AN ARRAY: ");

scanf("%d", &size);

for(i=0; i<size; i++)

{

printf("Enter %d element : ",i+1);

scanf("%d", &arr[i]);

}

part(arr, 0, size-1);

printf("\nSORTED ELEMENTS\n\n");

for(i=0; i<size; i++)

{

printf("%d ", arr[i]);

}

getch();

return 0;

}

void part(int arr[ ], int min, int max)

{

int mid;

if(min<max)

{

mid=(min+max)/2;

part(arr, min, mid);

part(arr, mid+1, max);

merge(arr, min, mid, max);

}

}

void merge(int arr[ ], int min, int mid, int max)

{

int tmp[30];

int i, j, k, m;

j=min;

m=mid+1;

for(i=min; j<=mid && m<=max ; i++)

{

if(arr[j]<=arr[m])

{

tmp[i]=arr[j];

j++;

}

else

{

tmp[i]=arr[m];

m++;

}

}

if(j>mid)

{

for(k=m; k<=max; k++)

{

tmp[i]=arr[k];

i++;

}

}

else

{

for(k=j; k<=mid; k++)

{

tmp[i]=arr[k];

i++;

}

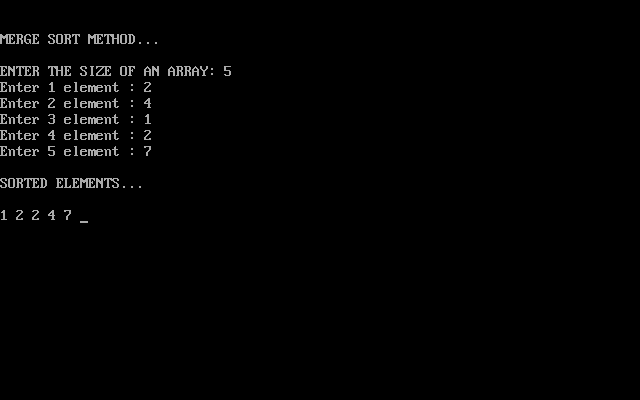
}

for(k=min; k<=max; k++)

arr[k]=tmp[k];

}

**OUTPUT-**



**Program No: 6**

**Aim: Write A Program Of Quick Sort**

#include<stdio.h>

#include<conio.h>

void quicksort(int array[], int firstIndex, int lastIndex)

{

int pivotIndex, temp, index1, index2;

if(firstIndex < lastIndex)

{

pivotIndex = firstIndex;

index1 = firstIndex;

index2 = lastIndex;

while(index1 < index2)

{

while(array[index1] <= array[pivotIndex] && index1 < lastIndex)

{

index1++;

}

while(array[index2]>array[pivotIndex])

{

index2--;

}

if(index1<index2)

{

temp = array[index1];

array[index1] = array[index2];

array[index2] = temp;

}

}

temp = array[pivotIndex];

array[pivotIndex] = array[index2];

array[index2] = temp;

quicksort(array, firstIndex, index2-1);

quicksort(array, index2+1, lastIndex);

}

}

void main()

{

int array[100],n,i;

clrscr();

printf("\n\t\t\t\tWELCOME");

printf("\n\nQUICK SORT METHOD...");

printf("\nEnter The Array Limit: ");

scanf("%d",&n);

printf("\nEnter Elements In The List : \n");

for(i = 0; i < n; i++)

{

printf("Enter %d element : ",i+1);

scanf("%d",&array[i]);

}

quicksort(array,0,n-1);

printf("Sorted Elements After Quick Sort: ");

for(i=0;i<n;i++)

{

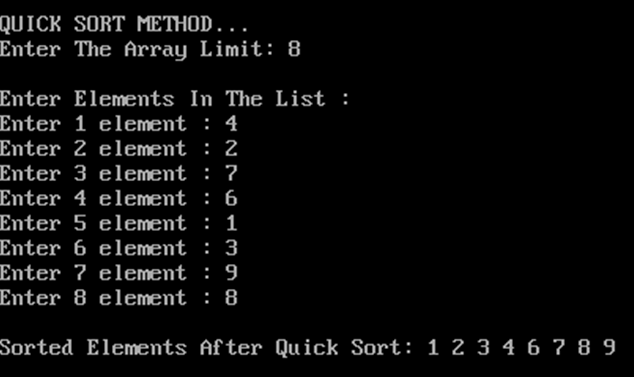
printf("%d ",array[i]);

}

getch();

}

**OUTPUT-**



**PROGRAM 6:**

**AIM- Longest Common Subsequence**

#include<stdio.h>

#include<string.h>

inti,j,m,n,c[20][20];

char x[20],y[20],b[20][20];

void print(inti,int j)

{

if(i==0 || j==0)

return;

if(b[i][j]=='c')

{

print(i-1,j-1);

printf("%c",x[i-1]);

}

else if(b[i][j]=='u')

print(i-1,j);

else

print(i,j-1);

}

voidlcs()

{

m=strlen(x);

n=strlen(y);

for(i=0;i<=m;i++)

c[i][0]=0;

for(i=0;i<=n;i++)

c[0][i]=0;

for(i=1;i<=m;i++)

for(j=1;j<=n;j++)

{

if(x[i-1]==y[j-1])

{

c[i][j]=c[i-1][j-1]+1;

b[i][j]='c';

}

else if(c[i-1][j]>=c[i][j-1])

{

c[i][j]=c[i-1][j];

b[i][j]='u';

}

else

{

c[i][j]=c[i][j-1];

b[i][j]='l';

}

}

}

int main()

{

printf("Enter 1st sequence:");

scanf("%s",x);

printf("Enter 2nd sequence:");

scanf("%s",y);

printf("\nThe Longest Common Subsequence is ");

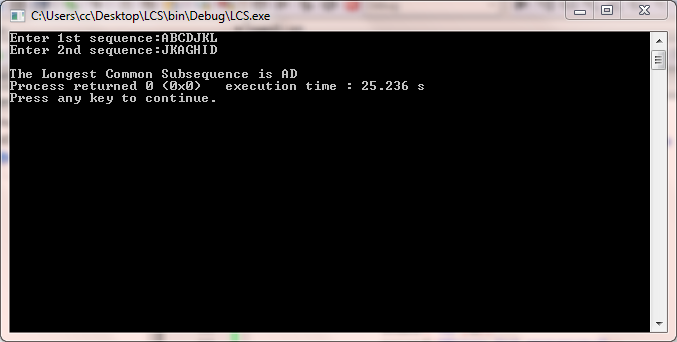
lcs();

print(m,n);

return 0;

}

**OUTPUT-**

****

**PROGRAM-8**

**AIM-Write a C Program to Implement Strassen’s Algorithm**

#include<stdio.h>

int main(){

int a[2][2], b[2][2], c[2][2], i, j;

int m1, m2, m3, m4 , m5, m6, m7;

printf("Enter the 4 elements of first matrix: ");

for(i = 0;i < 2; i++)

for(j = 0;j < 2; j++)

scanf("%d", &a[i][j]);

printf("Enter the 4 elements of second matrix: ");

for(i = 0; i < 2; i++)

for(j = 0;j < 2; j++)

scanf("%d", &b[i][j]);

printf("\nThe first matrix is\n");

for(i = 0; i < 2; i++){

printf("\n");

for(j = 0; j < 2; j++)

printf("%d\t", a[i][j]);

}

printf("\nThe second matrix is\n");

for(i = 0;i < 2; i++){

printf("\n");

for(j = 0;j < 2; j++)

printf("%d\t", b[i][j]);

}

m1= (a[0][0] + a[1][1]) \* (b[0][0] + b[1][1]);

m2= (a[1][0] + a[1][1]) \* b[0][0];

m3= a[0][0] \* (b[0][1] - b[1][1]);

m4= a[1][1] \* (b[1][0] - b[0][0]);

m5= (a[0][0] + a[0][1]) \* b[1][1];

m6= (a[1][0] - a[0][0]) \* (b[0][0]+b[0][1]);

m7= (a[0][1] - a[1][1]) \* (b[1][0]+b[1][1]);

c[0][0] = m1 + m4- m5 + m7;

c[0][1] = m3 + m5;

c[1][0] = m2 + m4;

c[1][1] = m1 - m2 + m3 + m6;

printf("\nAfter multiplication using Strassen's algorithm \n");

for(i = 0; i < 2 ; i++){

printf("\n");

for(j = 0;j < 2; j++)

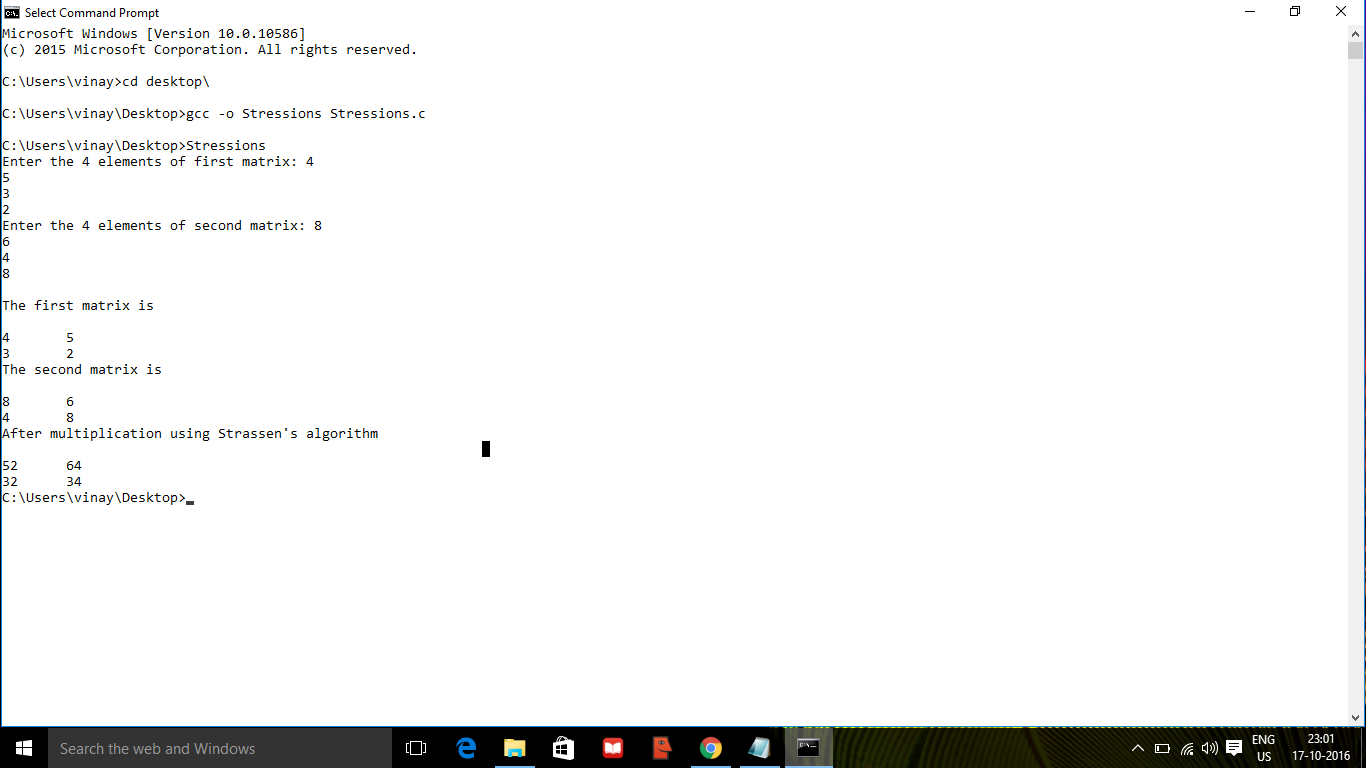
printf("%d\t", c[i][j]);

}

return 0;

}

**OUTPUT-**



**PROGRAM-9**

**AIM-**  **Write a C Program to implement Dijkstra's Algorithm**

#include<stdio.h>

#define inf 9999

#define size 10/\*Defining maximum size of the matrix\*/

main()

{

int a[size][size],i,j,n,v1,v2,lcost;

int dij(int[][j],int,int,int);

printf("Enter the number of vertex : ");

scanf("%d",&n);

/\*Input 0 if there is no direct edge between vertex pair\*/

printf("Enter a weighted matrix(with weights) as input :\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

printf("Enter the value of a[%d][%d] : ",i,j);

scanf("%d",&a[i][j]);

}

}

printf("The entered matrix is:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

printf("%d\t",a[i][j]);

printf("\n");

}

printf("Enter starting vertex v");

scanf("%d",&v1);

printf("Enter ending vertex v");

scanf("%d",&v2);

/\*Check for validity of input vertices\*/

if(v1<0||v1>n-1||v2<0||v2>n-1)

{

printf("!!!!!ERROR!!!!!n");

printf("!!!!!Invalid vertex given!!!!!");

return 0;

}

printf("Shortest path between v%d & v%d : ",v1,v2);

lcost=dij(a,n,v1,v2);

printf("Shortest cost between v%d & v%d : ",v1,v2);

printf("%d",lcost);/\*Print the shortest cost\*/

}

/\*The input graph,no. of vertices n,source vertex v1 and destination vertex v2 are passed as parameters\*/

int dij(int a[size][size],int n,int v1,int v2)

{

int length[size],set[size],path[size],i,j,s,z,tmp,temp[size],c=0,f=0;

s=v1;

z=v2;

int srch\_min(int[],int[],int);

for(i=0;i<n;i++)

set[i]=0;

for(i=0;i<n;i++)

{

if(a[s][i]==0)/\*There is no direct edge between vertices s and i\*/

{

length[i]=inf;

path[i]=0;/\*Empty path\*/

}

else

{

length[i]=a[s][i];

path[i]=s;/\*s is immediate predecessor of i\*/

}

}

set[s]=1;/\*s is included in the set\*/

length[s]=0;/\*s is implicitly enumerated with length as 0\*/

while(set[z]!=1)/\*Iteration will be considered until final vertex z belongs to s\*/

{

j=srch\_min(length,set,n);/\*Select a vertex j with minimum label such that it is not included in the set[]\*/

set[j]=1;/\*Vertex j is included in the set[]\*/

for(i=0;i<n;i++)

{

if(set[i]!=1)

{

if(a[i][j]!=0)

{

if(length[j]+a[i][j]<length[i])/\*When exsisting label is not minimum only then replacement is done\*/

{

length[i]=length[j]+a[i][j];

path[i]=j;

}

}

}

}

}

j=0;

i=z;

while(i!=s)

{

tmp=path[i];

temp[j]=tmp;

i=tmp;

j++;

c++;

}

for(j=c-1;j>=0;j--)

{

printf("%d->",temp[j]);/\*Print the shortest path\*/

if(temp[j]==z)

f=1;

}

if(f!=1)

printf("%d",z);

printf("\n");

return length[z];

}

/\*This function will return a vertex with minimum label such that it is not included in set[]\*/

int srch\_min(int length[],int set[],int n)

{

int min,i,min\_index;

min=99999,min\_index;

for(i=1;i<n;i++)

{

if(set[i]!=1)

{

if(length[i]<min)

{

min=length[i];

min\_index=i;

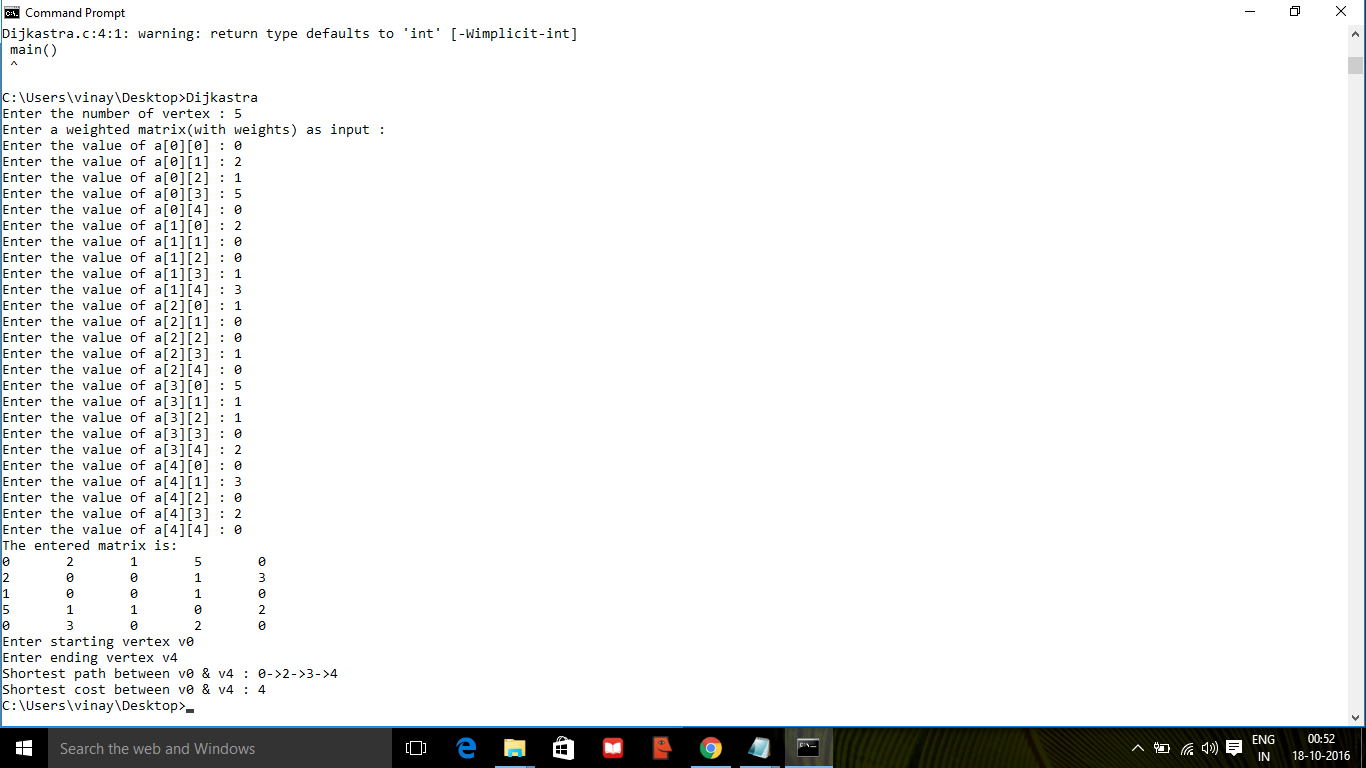
}

}

}

return min\_index;

}



**PROGRAM-10**

**AIM- Write A C program for Bellman-Ford's single source shortest path algorithm.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

// a structure to represent a weighted edge in graph

struct Edge

{

int src, dest, weight;

};

// a structure to represent a connected, directed and

// weighted graph

struct Graph

{

// V-> Number of vertices, E-> Number of edges

int V, E;

// graph is represented as an array of edges.

struct Edge\* edge;

};

// Creates a graph with V vertices and E edges

struct Graph\* createGraph(int V, int E)

{

struct Graph\* graph =

(struct Graph\*) malloc( sizeof(struct Graph) );

graph->V = V;

graph->E = E;

graph->edge =

(struct Edge\*) malloc( graph->E \* sizeof( struct Edge ) );

return graph;

}

// A utility function used to print the solution

void printArr(int dist[], int n)

{

printf("Vertex Distance from Source\n");

for (int i = 0; i < n; ++i)

printf("%d \t\t %d\n", i, dist[i]);

}

// The main function that finds shortest distances from src to

// all other vertices using Bellman-Ford algorithm. The function

// also detects negative weight cycle

void BellmanFord(struct Graph\* graph, int src)

{

int V = graph->V;

int E = graph->E;

int dist[V];

// Step 1: Initialize distances from src to all other vertices

// as INFINITE

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX;

dist[src] = 0;

// Step 2: Relax all edges |V| - 1 times. A simple shortest

// path from src to any other vertex can have at-most |V| - 1

// edges

for (int i = 1; i <= V-1; i++)

{

for (int j = 0; j < E; j++)

{

int u = graph->edge[j].src;

int v = graph->edge[j].dest;

int weight = graph->edge[j].weight;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

dist[v] = dist[u] + weight;

}

}

// Step 3: check for negative-weight cycles. The above step

// guarantees shortest distances if graph doesn't contain

// negative weight cycle. If we get a shorter path, then there

// is a cycle.

for (int i = 0; i < E; i++)

{

int u = graph->edge[i].src;

int v = graph->edge[i].dest;

int weight = graph->edge[i].weight;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

printf("Graph contains negative weight cycle");

}

printArr(dist, V);

return;

}

// Driver program to test above functions

int main()

{

/\* Let us create the graph given in above example \*/

int V = 5; // Number of vertices in graph

int E = 8; // Number of edges in graph

struct Graph\* graph = createGraph(V, E);

// add edge 0-1 (or A-B in above figure)

graph->edge[0].src = 0;

graph->edge[0].dest = 1;

graph->edge[0].weight = -1;

// add edge 0-2 (or A-C in above figure)

graph->edge[1].src = 0;

graph->edge[1].dest = 2;

graph->edge[1].weight = 4;

// add edge 1-2 (or B-C in above figure)

graph->edge[2].src = 1;

graph->edge[2].dest = 2;

graph->edge[2].weight = 3;

// add edge 1-3 (or B-D in above figure)

graph->edge[3].src = 1;

graph->edge[3].dest = 3;

graph->edge[3].weight = 2;

// add edge 1-4 (or A-E in above figure)

graph->edge[4].src = 1;

graph->edge[4].dest = 4;

graph->edge[4].weight = 2;

// add edge 3-2 (or D-C in above figure)

graph->edge[5].src = 3;

graph->edge[5].dest = 2;

graph->edge[5].weight = 5;

// add edge 3-1 (or D-B in above figure)

graph->edge[6].src = 3;

graph->edge[6].dest = 1;

graph->edge[6].weight = 1;

// add edge 4-3 (or E-D in above figure)

graph->edge[7].src = 4;

graph->edge[7].dest = 3;

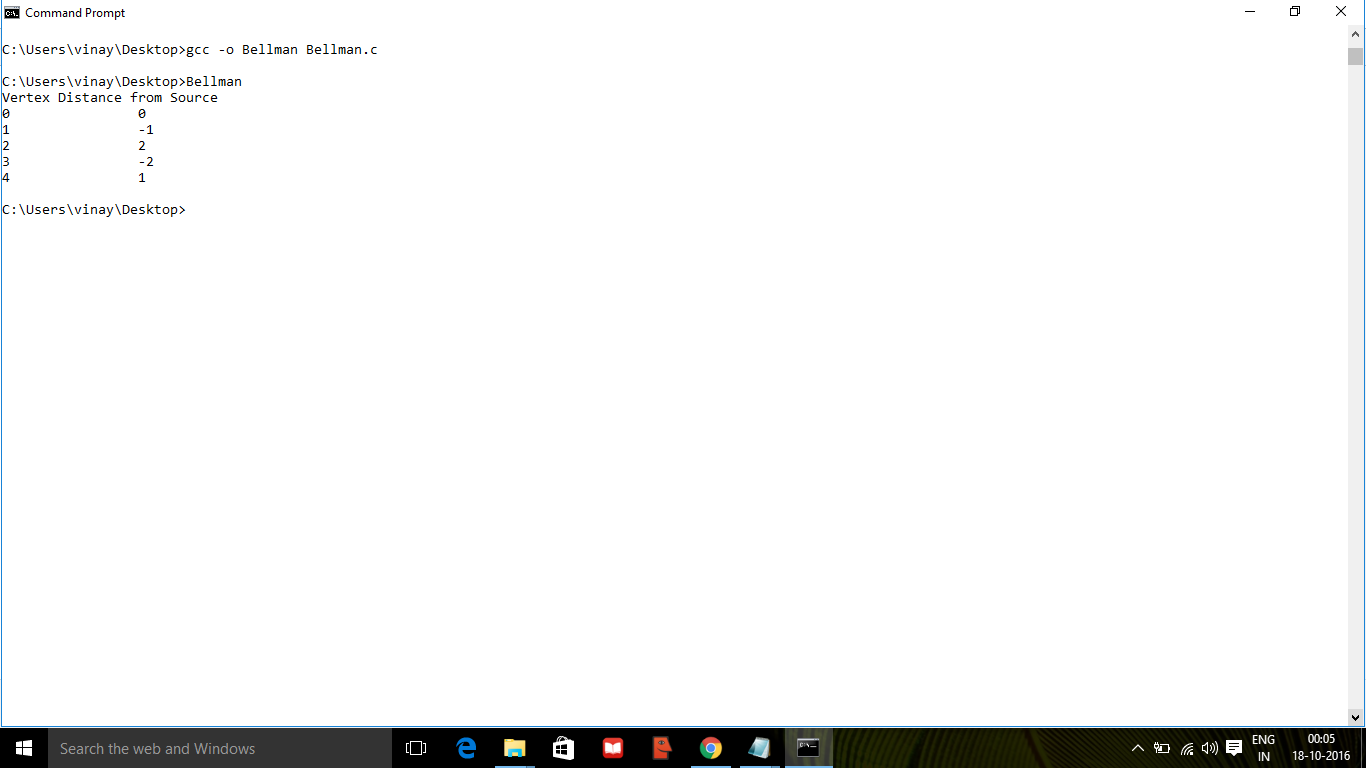
graph->edge[7].weight = -3;

BellmanFord(graph, 0);

return 0;

}

**OUTPUT-**



**PROGRAM-11**

**AIM-** **Write a C Program implement Kruskal's algorithm.**

#include<stdio.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()

{

printf("\n\n\tImplementation of Kruskal's algorithm\n\n");

printf("\nEnter the no. of vertices\n");

scanf("%d",&n);

printf("\nEnter the 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("\nThe edges of Minimum Cost Spanning Tree are\n\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("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n\tMinimum cost = %d\n",mincost);

}

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-**

