**NAME: DIVYANI DIPAK PATIL**

**ROLL NO: 35**

**PRN: 2254491246036**

**1)Binary search:**

**Using divide and conquer method write a program for binary search.**

**Program:**

#include <iostream>

using namespace std;

int binarySearch(int array[], int x, int LOW, int UP) {

if (UP >= LOW) {

int mid = (LOW +UP)/ 2;

// If found at mid, then return it

if (array[mid] == x)

return mid;

// Search the left half

if (array[mid] > x)

return binarySearch(array, x, LOW, mid - 1);

// Search the right half

return binarySearch(array, x, mid + 1, UP);

}

return -1; // Element not found

}

int main() {

int array[50];

int i,n;

cout<<"Enter size of array:\n";

cin>>n;

cout<<"Enter the elements in array:\n";

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

cin>>array[i];

int x;

cout << "Enter the element you want to search: ";

cin >> x;

int result = binarySearch(array, x, 0, n - 1);

if (result == -1)

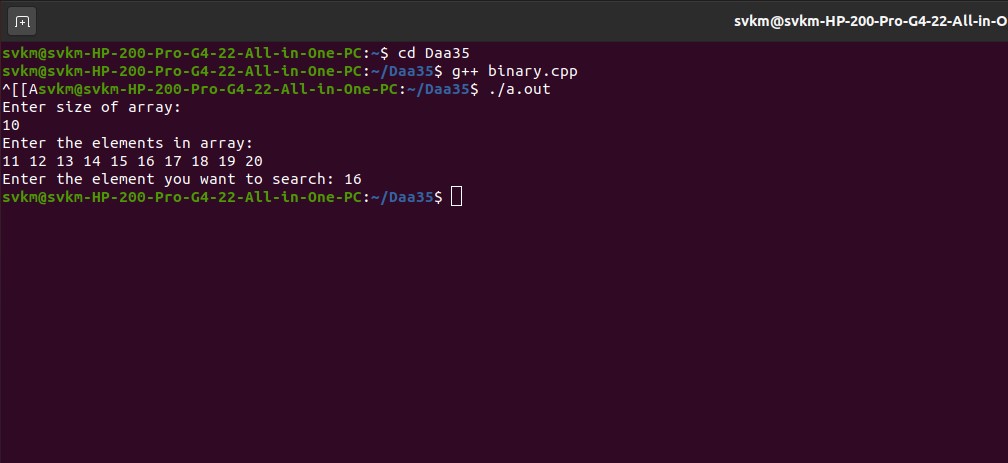
cout << "Element not found";

else

cout << "Element is found at index " << result;

return 0;

}

Output:

1. **Merge sort:**

**Using divide and conquer method and recursive approch write a proram to sort the list using top down merge sort.**

**Program:**

#include<iostream>

using namespace std;

void mergeR(int arr[],int low,int up);

void merge(int arr[],int temp[],int low1,int up1,int low2,int up2);

void copy(int arr[],int temp[],int low,int up);

int main()

{

int i;

int n;

cout<<"\nEnter the size of an array :";

cin>>n;

int arr[n];

cout<<"\nEnter the elements in an array :";

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

{

cin>>arr[i];

}

mergeR(arr,0,n-1);

cout<<"\nThe merge sort result is :";

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

{

cout<<arr[i]<<"\t";

}

return 0;

}

void mergeR(int arr[],int low,int up)

{

int mid;

int temp[10];

if(low<up)

{

mid=(low+up)/2;

mergeR(arr,low,mid);

mergeR(arr,mid+1,up);

merge(arr,temp,low,mid,mid+1,up);

copy(arr,temp,low,up);

}

}

void merge(int arr[],int temp[],int low1,int up1,int low2,int up2)

{

int i=low1;

int j=low2;

int k=low1;

while((i<=up1)&&(j<=up2))

{

if(arr[i]<arr[j])

{

temp[k++]=arr[i++];

}

else

{

temp[k++]=arr[j++];

}

}

while(i<=up1)

{

temp[k++]=arr[i++];

}

while(j<=up2)

{

temp[k++]=arr[j++];

}

}

void copy(int arr[],int temp[],int low,int up)

{

int i;

for(i=low;i<=up;i++)

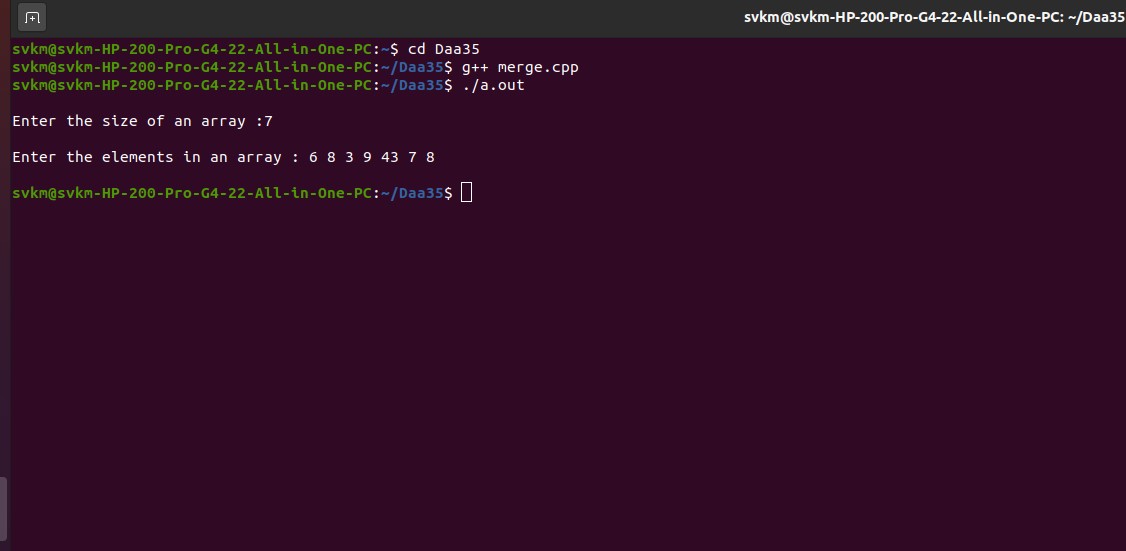
{

arr[i]=temp[i];

}

}

OUTPUT:-



**3)QUICK SORT:**

**Problem statement : Sort a given set of n integer elements using quick sort method and compute its time complexity . Run the program for varied values of n and record the time taken to sort. Demonstrate how divide and conquer method works along with complexity analysis: worst case ,average case and best case.**

**Program:**

#include<iostream>

using namespace std;

void quick(int arr[] , int low , int up);

int partition(int arr[],int low,int up);

int main()

{

int i;

int n;

cout<<"\nEnter the size of an array :";

cin>>n;

int arr[n];

cout<<"\nEnter the elements in an array :";

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

{

cin>>arr[i];

}

quick(arr,0,n-1);

cout<<"\nThe quick sort result is :";

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

{

cout<<arr[i]<<" ";

}

return 0;

}

void quick(int arr[],int low,int up)

{

int pvtloc;

if(low>=up)

return;

pvtloc=partition(arr,low,up);

quick(arr,low,pvtloc-1);

quick(arr,pvtloc+1,up);

}

int partition(int arr[],int low,int up){

int temp,i,j,pivot;

i = low+1;

j = up;

pivot = arr[low];

while(i<=j)

{

while((arr[i]<pivot)&&(i<up))

i++;

while(arr[j]>pivot)

j--;

if(i<j)

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

i++;

j--;

}

else

{

i++;

}

}

arr[low] = arr[j];

arr[j] = pivot;

return j;

}

OUTPUT:



**4)Strassens Matrix:**

**Problem statement : Write a program to implement Strassen’s Matrix Multiplication.**

**Program:**

#include<iostream>

using namespace std;

int main()

{ int i,j;

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

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

cout<<"Enter elements of first matrix:\n";

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

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

{

cin>>a[i][j];

}

}

cout<<"\nEnter elements of second matrix:\n";

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

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

{

cin>>b[i][j];

}

}

cout<<"\nprint the first matrix:\n";

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

{

cout<<"\n";

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

{

cout<<a[i][j];

}

}

cout<<"\nprint the second matrix:\n";

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

{

cout<<"\n";

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

{

cout<<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;

cout<<"\nafter multiplication using strassen's algorithm\n";

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

{

cout<<"\n";

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

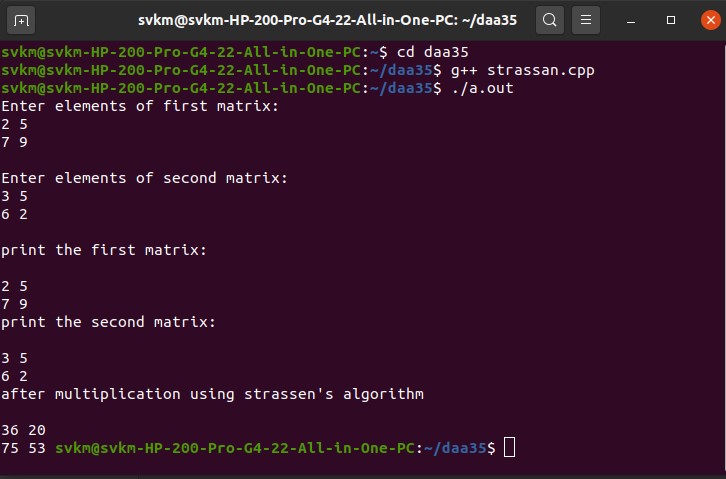
cout<<c[i][j];

}

return 0;

}

Output:



**5)knapsack:**

**Fractional knapsack problem using greedy method**

**Using greedy method write a program to implement fractional knapsack problem.**

**Program:**

#include<iostream>

using namespace std;

void knapsac(int n,float wt[],float profit[], float capacity)

{ float x[20],tp=0;

int i,j,u;

u=capacity;

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

x[i]=0,0;

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

{

if (wt[i]>u)

break;

else

{

x[i]=1.0;

tp=tp+profit[i];

u=u-wt[i];

}

}

if(i<n)

x[i]=u/wt[i];

tp=tp+(x[i]\*profit[i]);

cout<<"\n the resultvector is:";

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

cout<<x[i];

cout<<"\n Maximum profit is:";

cout<<tp;

}

int main()

{

float wt[20],profit[20],capacity;

int n,i,j;

float ratio[20],temp;

cout<<"Enter no.of objects:";

cin>>n;

cout<<"\n Enter wt and profit of each object";

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

{

cin>>wt[i];

cin>>profit[i];

}

cout<<"Enter the capacity of knapsac:";

cin>>capacity;

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

{

ratio[i]=profit[i]/wt[i];

}

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

{

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

{

if (ratio[i]<ratio[j])

{

temp=ratio[j];

ratio[j]=ratio[i];

ratio[i]=temp;

temp=wt[j];

wt[j]=wt[i];

wt[i]=temp;

temp=profit[j];

profit[j]=profit[i];

profit[i]=temp;

}

}

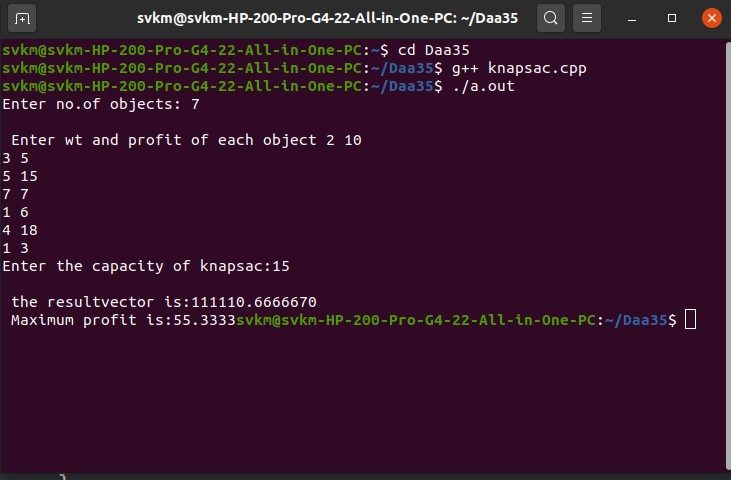
}

knapsac(n,wt,profit,capacity);

return 0;

}

OUTPUT:



**6)Dijkstra:**

**Single source shortest path problem with for non negative weights**

**From a given vertex in a weighted connected graph with non negative weights, write a program to find the shortest path to other vertices using dijkstra’s algorithm.**

**Program:**

#include<iostream>

using namespace std;

# define max 100

# define temp 0

#define perm 1

# define infinity 9999

#define nil -1

void find\_path(int s,int v);

void djkstra(int s);

int min\_temp();

void creat\_graph();

int n;

int adj[max][max];

int predecessor[max];

int pathlen[max];

int status[max];

int main()

{

int s,v;

creat\_graph();

cout<<"Enter source vertex:"<<endl;

cin>>s;

djkstra (s);

while(1)

{

cout<<"enter destination vertex(-1 to quit):"<<endl;

cin>> v;

if(v==-1)

break;

if(v<0||v>=n)

cout<<"vertex does not exists:;";

else if(v==s)

cout<<"source and destination vertices are same";

else if(pathlen[v]==infinity)

cout<<"there is no path from source to destination";

else

find\_path(s,v);

}

return 0;

}

void djkstra(int s)

{ int i, current;

/\* make all vertices temp\*/

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

{

predecessor[i]=nil;

pathlen[i]=infinity;

status[i]=temp;

}

/\*make pathlen of source vertex 0 \*/

pathlen[s]=0;

while(1)

{

/\*search of temp vertex having mini pathlen and make it current vertex\*/

current=min\_temp();

if (current==nil)

return;

status[current]=perm;

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

{

if(adj[current][i]!=0 && status[i]==temp)

if(pathlen[current]+adj[current][i]<pathlen[i])

{

predecessor[i]=current;

pathlen[i]=pathlen[current];

}

}

}

}

int min\_temp()

{

int i;

int min=infinity;

int k=nil;

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

{

if (status[i]==temp&& pathlen[i]<min)

{

min=pathlen[i];

k=i;

}

}

return k;

}

void find\_path(int s,int v)

{

int i,u;

int path[max];

int shortdis=0;

int count=0;

while(v!=s)

{

count++;

path[count]=v;

u=predecessor[v];

shortdis+=adj[u][v];

v=u;

}

count++;

path[count]=s;

cout<<"shortest path is:"<<endl;

for(i=count;i>=1;i--)

cout<<path[i]<<endl;

cout<<"shortest distance is:"<<shortdis<<endl;

}

void creat\_graph()

{

int i,max\_e,orign,dest,wt;

cout<<"entr no of vertices:"<<endl;

cin>>n;

max\_e=n\*(n-1);

for(i=1;i<=max\_e;i++)

{

cout<<"enter edge"<<i<<"(enter -1-1 to exit)";

cin>>orign>>dest;

if((orign==-1)&&(dest==-1))

break;

cout<<"enter the weight of this edge:"<<endl;

cin>>wt;

if(orign>n||dest>=n||orign<0||dest<0)

{

cout<<"invalid edge!";

i--;

}

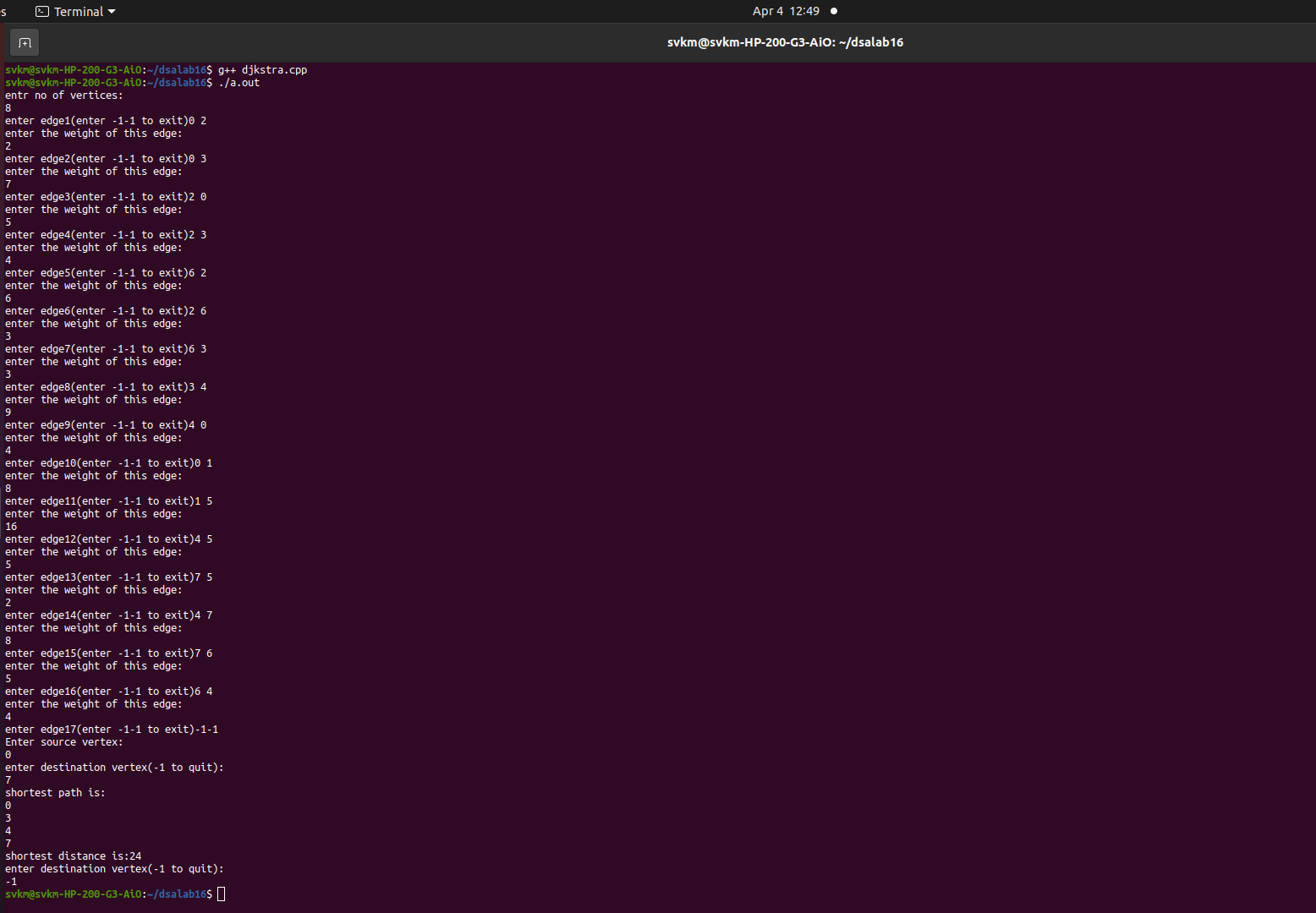
else

adj[orign][dest]=wt;

}

}

OUTPUT:



**7)Bellman ford:**

**Single source shortest path problem with for negative weights**

**Write a program to implimante Bellman ford algorithm for solving single source shortest path problem considering negative weights for graph.**

**Program:**

#include<iostream>

using namespace std;

#define max 100

#define infinity 9999

#define nil -1

#define true 1

#define false 0

int n;

int adj[max][max];

int pre[max];

int pl[max];

int ispresent\_in\_queue[max];

int front ,rear;

int queue [max];

void initialize\_q();

void insert\_q(int u);

int delete\_q();

int isempty\_q();

void create\_graph();

void find\_path(int s,int v);

int bellmanford(int s);

int main()

{

int flag,s,v;

create\_graph();

cout<<"Enter the source vertex:";

cin>>s;

flag=bellmanford(s);

if(flag==-1)

{

cout<<"error:Negative cycle in graph\n";

exit (1);

}

while(1)

{

cout<<"Enter destination verex(-1 to exit):";

cin>>v;

if(v==-1)

break;

if(v<0||v>=n)

cout<<"the vrtex does not exist\n";

else if(v==s)

cout<<"source and destination verex are same\n";

else if (pl[v]==infinity)

cout<<"thre is no pathfrom s to v\n";

else

find\_path(s,v);

}

return 0;

}

void find\_path(int s,int v)

{

int i,u;

int path[max];

int shortdis=0;

int count=0;

while(v!=s)

{

count++;

path[count]=v;

u=pre[v];

shortdis+=adj[u][v];

v=u;

}

count++;

path[count]=s;

cout<<"Shortest pah is: ";

for(i=count;i>=1;i--)

cout<<path[i];

cout<<"shortest distance is:"<<shortdis;

}

int bellmanford(int s)

{

int k=0,i,current;

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

{

pre[i]=nil;

pl[i]=infinity;

ispresent\_in\_queue[i]=false;

}

initialize\_q();

pl[s]=0;

insert\_q(s);

ispresent\_in\_queue[i]=true;

while(!isempty\_q())

{

current=delete\_q();

ispresent\_in\_queue[current]=false;

if(s==current)

k++;

if(k>n)

return -1;

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

{

if (adj[current][i]!=0)

if(pl[i]>pl[current]+adj[current][i])

{

pl[i]=pl[current]+adj[current][i];

pre[i]=current;

if(!ispresent\_in\_queue[i])

{

insert\_q(i);

ispresent\_in\_queue[i]=true;

}

}

}

}

return 1;

}

void initialize\_q()

{ int i;

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

queue[i]=0;

rear=-1;front=-1;

}

int isempty\_q()

{

if(front==-1||front>rear)

return 1;

else

return 0;

}

void insert\_q(int added\_ele)

{

if(rear==max-1)

{

cout<<"queue underflow";

exit(1);

}

else

{

if(front==-1)

front =0;

rear+=1;

queue[rear]=added\_ele;

}

}

int delete\_q()

{

int d;

if(front==-1||front>rear)

{

cout<<"queue underflow";

exit(1);

}

else

{

d=queue[front];

front=front+1;

}

return d;

}

void create\_graph()

{

int i,max\_e,orign,dest,wt;

cout<<"enter no of vertices:"<<endl;

cin>>n;

max\_e=n\*(n-1);

for(i=1;i<=max\_e;i++)

{

cout<<"enter edge"<<i;

cin>>orign>>dest;

if((orign==-1)&&(dest==-1))

break;

cout<<"enter the weight of this edge:"<<endl;

cin>>wt;

if(orign>n||dest>=n||orign<0||dest<0)

{

cout<<"invalid edge!";

i--;

}

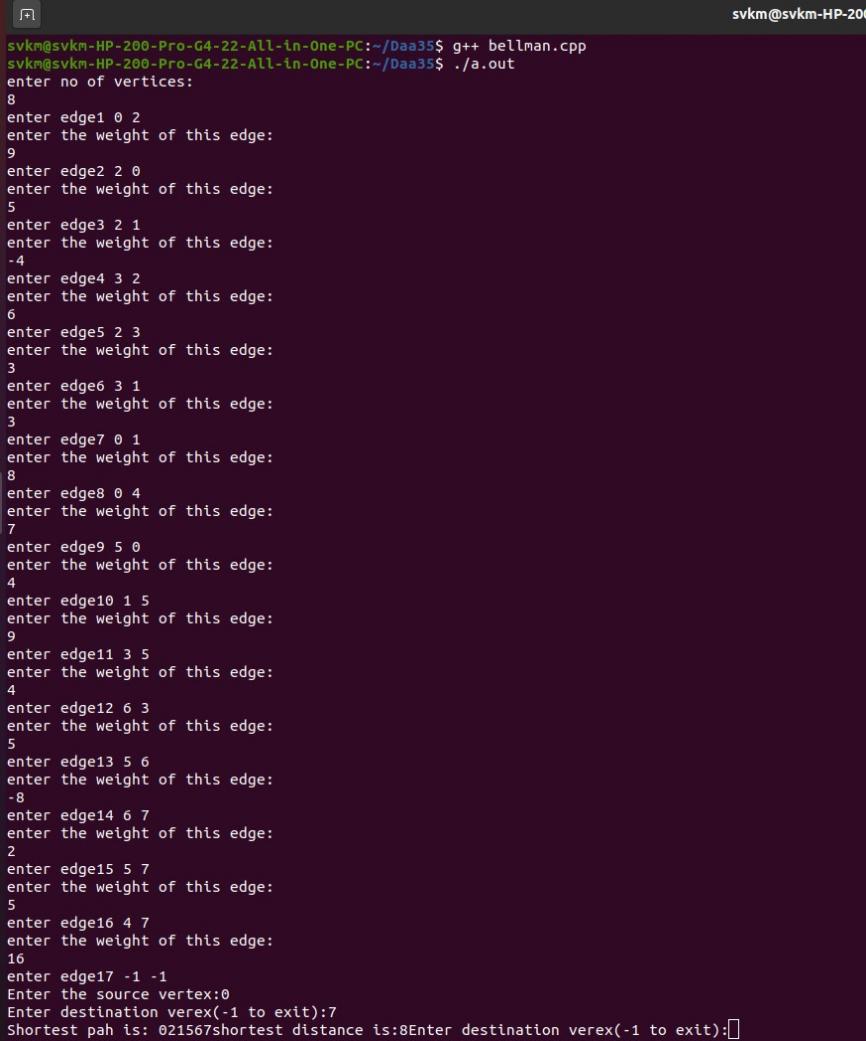
else

adj[orign][dest]=wt;

}

}

OUTPUT:



**8)BFS PROGRAM:**

**Breadth First Search**

**Write a program to create a graph and traverse using Breadth First Search.**

**Program:**

#include<iostream>

using namespace std;

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX];

int state[MAX];

int queue[MAX], front = -1, rear = -1;

void create\_graph();

void BF\_traversal();

void bfs(int v);

void insert\_queue(int vertex);

int delete\_queue();

int isempty\_queue();

int main() {

create\_graph();

BF\_traversal();

return 0;

}

void create\_graph() {

int i, max\_edges, origin, destin;

cout << "Enter number of vertices: ";

cin >> n;

max\_edges = n \* (n - 1);

for (i = 0; i < max\_edges; i++) {

cout << "Enter edge (-1 -1 to quit): ";

cin >> origin >> destin;

if ((origin == -1) && (destin == -1))

break;

adj[origin][destin] = 1; // Marking the adjacency

}

}

void BF\_traversal() {

int v;

cout << "Enter starting vertex for breadth-first search: ";

cin >> v;

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

state[i] = initial;

bfs(v);

}

void bfs(int v) {

int i;

insert\_queue(v);

state[v] = waiting;

while (!isempty\_queue()) {

v = delete\_queue();

cout << v << " ";

state[v] = visited;

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

if (adj[v][i] == 1 && state[i] == initial) {

insert\_queue(i);

state[i] = waiting;

}

}

}

cout << endl;

}

void insert\_queue(int vertex) {

if (rear == MAX - 1)

cout << "queue is overflow:";

else {

if (front == -1)

front = 0;

rear = rear + 1;

queue[rear] = vertex;

}

}

int isempty\_queue() {

if (front == -1 || front > rear)

return 1;

else

return 0;

}

int delete\_queue() {

int del\_item;

if (front == -1 || front > rear) {

cout << "queue is underflow:";

exit(1);

}

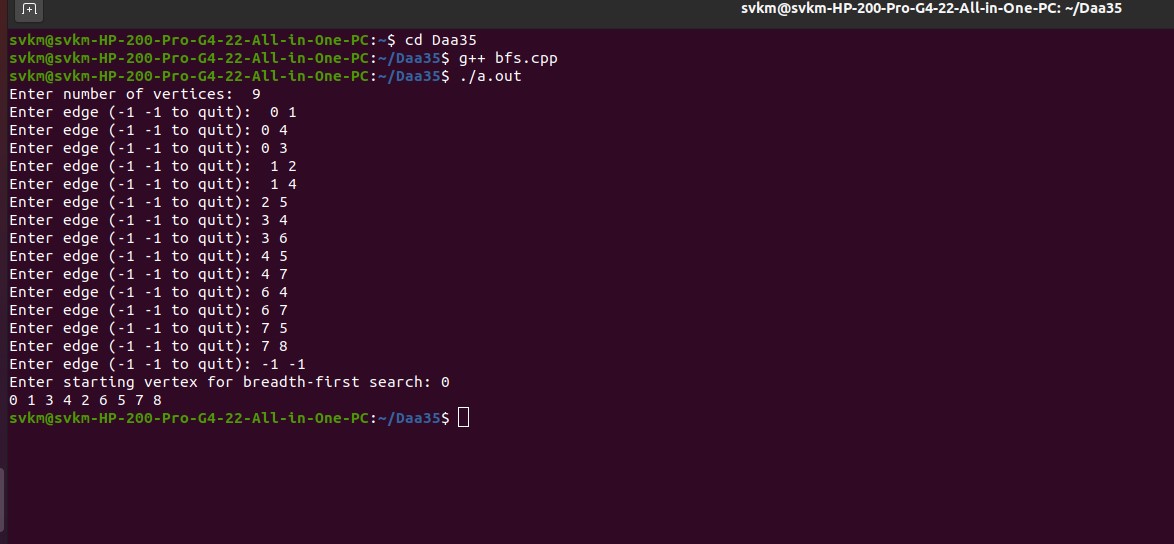
del\_item = queue[front];

front = front + 1;

return del\_item;

}

OUTPUT:



**9)DFS PROGRAM:**

**Depth First Search**

**Write a program to create a graph and traverse using Depth First Search.**

**Program:**

Output:#include<iostream>

#include<stdlib.h> // for exit() function

using namespace std;

#define MAX 100

#define initial 1

#define visited 2

int n;

int adj[MAX][MAX];

int state[MAX];

int stack[MAX]; // Fix: stack array declaration

int top=-1;

void push(int v);

int pop();

void create\_graph();

void DF\_traversal();

void dfs(int v);

int isempty\_stack();

int main() {

create\_graph();

DF\_traversal();

return 0;

}

void DF\_traversal() {

int v;

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

state[v] = initial;

cout << "Enter starting vertex for depth-first search: ";

cin >> v;

dfs(v);

}

void dfs(int v) {

int i;

push(v);

while (!isempty\_stack()) {

v = pop();

if(state[v]==initial) {

cout << v << " ";

state[v] = visited;

}

for (i = n-1; i >=0 ; i--) {

if (adj[v][i] == 1 && state[i] == initial)

push(i);

}

}

}

void create\_graph() {

int i, max\_edges, origin, destin ;

cout << "Enter the number of nodes: ";

cin >> n;

max\_edges = n \* (n - 1);

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

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

adj[i][j] = 0;

}

}

for (i = 1; i <= max\_edges; i++) {

cout << "Enter edge (-1 -1 to quit): ";

cin >> origin >> destin;

if (origin == -1 && destin == -1)

break;

if (origin == -1 || destin >= n || origin < 0 || destin < 0) {

cout << "Invalid edge.\n";

i--;

}

else

adj[origin][destin] = 1;

}

}

void push(int v) {

if (top >= (MAX-1)) // Fix: use >= instead of ==

{

cout << "stack overflow\n";

return;

}

top = top + 1;

stack[top] = v;

}

int pop() {

int v;

if(top == -1)

{

cout << "stack is underflow\n";

// Fix: return an error code instead of exit(1)

return -1;

}

else

{

v = stack[top];

top = top - 1;

return v;

}

}

int isempty\_stack() {

if (top == -1)

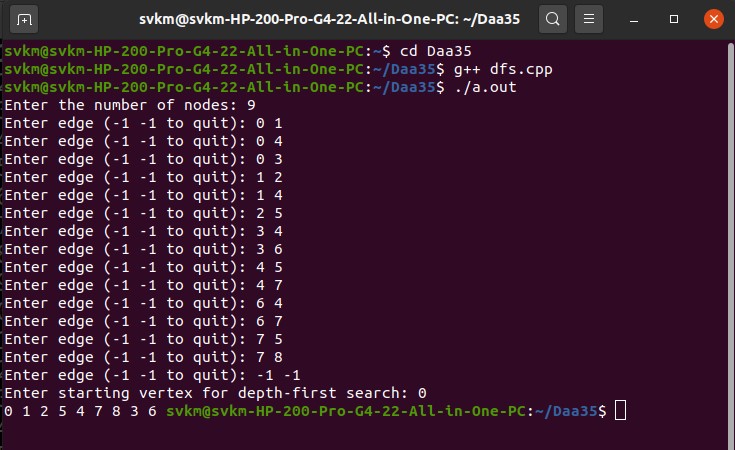
return 1;

else

return 0;

}

OUTPUT:



**Warshall**

#include<iostream>

using namespace std;

#define infi 99999

#define max 100

int n;

int adj[max][max];

int D[max][max];

int pre[max][max];

void createg();

void warshall();

void find\_path(int s,int d);

void display(int mat[max][max],int n);

int main(){

int s,d;

createg();

warshall();

while(1){

cout<<"Enter the source(-1) to exit: ";

cin>>s;

if(s==-1)

break;

cout<<"\nEnter destinaton vertex: ";

cin>>d;

if(s<0 || s>n-1||d<0 ||d>n-1){

cout<<"Enter a valid vertex\n";

continue;}

cout<<"Shortest path is :";

find\_path(s,d);

cout<<"Length of path is "<<D[s][d];

return 0;

}

}

void createg(){

int i,max\_edges,o,d,wt;

cout<<"Enter the number of vertices: ";

cin>>n;

max\_edges=n\*(n-1);

for(i=1;i<=max\_edges;i++){

cout<<"Enter the edge (-1 -1 to quit) "<<i<<":";

cin>>o>>d;

if((o==-1)&&(d==-1))

break;

cout<<"Enter the weight for this edge :";

cin>>wt;

if(o>=n||d>=n|o<0||d<0){

cout<<"Invalid edge\n";

i--;}

else{

adj[o][d]=wt;

}

}

}

void warshall(){

int i,j,k;

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

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

if(adj[i][j]==0){

D[i][j]=infi;

pre[i][j]=-1;

}

else{

D[i][j]=adj[i][j];

pre[i][j]=i;

}}

for(k=0;k<n;k++){

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

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

if(D[i][k]+D[k][j] <D[i][j]){

D[i][j]=D[i][k]+D[k][j];

pre[i][j]=pre[k][j];

}

}

cout<<"Shortest path matrix is :\n";

display(D,n);

cout<<"Predecessor matrix is :\n";

display(pre,n);

}

void find\_path(int s,int d){

int i;

int path[max];

int count ;

if(D[s][d]==infi){

cout<<"No path";

return;

}

count=-1;

do{

path[++count]=d;

d=pre[s][d];

}

while(d!=s);

path[++count]=s;

for(i=count ;i>=0;i--)

cout<<path[i]<<" ";

cout<<"\n";

}

void display(int mat[max][max],int n){

int i,j;

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

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

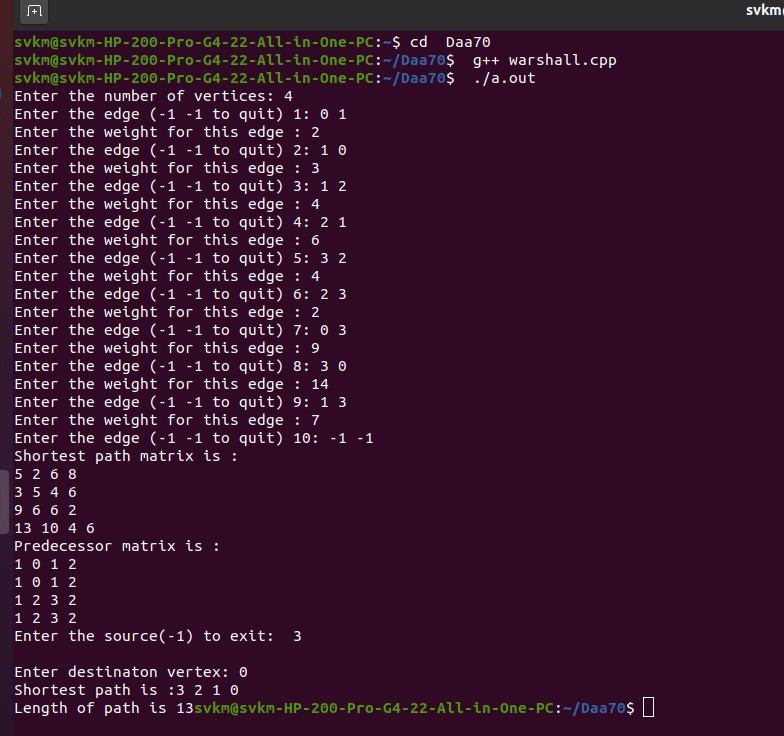
cout<<mat[i][j]<<" ";

cout<<"\n";

}

}

OUTPUT:-



**Implementation of Prim’s Algorithm**

**Input :**

#include<iostream>

using namespace std;

#define MAX 10

#define TEMP 0

#define PERM 1

#define infinity 9999

#define NIL -1

struct edge

{

int u;

int v;

};

int n;

int adj[MAX][MAX];

int predecessor[MAX];

int status[MAX];

int length[MAX];

void create\_graph();

void make\_tree(int r,struct edge tree[MAX]);

int min\_temp();

int main()

{

int wt\_tree=0;

int i,root;

struct edge tree[MAX];

create\_graph();

cout<<"\nEnter the root vertex : ";

cin>>root;

make\_tree(root,tree);

cout<<"\nEdges to be included in spanning tree are :\n";

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

{

cout<<tree[i].u<<" ";

cout<<tree[i].v;

cout<<"\n";

wt\_tree += adj[tree[i].u][tree[i].v];

}

cout<<"The weight of spanning tree is :";

cout<<wt\_tree;

return 0;

}

void create\_graph()

{

int i,max\_edges,origin,destin,wt;

cout<<"\nEnter the number of vertices :";

cin>>n;

max\_edges=n\*(n-1)/2;

for(i=1;i<=max\_edges;i++)

{

cout<<"Enter edge(-1 -1 to quit) "<<i<<" : " ;

cin>>origin>>destin;

if((origin==-1)&&(destin==-1))

break;

cout<<"Enter the weight for this edge :";

cin>>wt;

if(origin>=n||destin>=n||origin<0||destin<0)

{

cout<<"\nInvalid Edge";

i--;

}

else

{

adj[origin][destin]=wt;

adj[destin][origin]=wt;

}

}

}

void make\_tree(int r, struct edge tree[MAX])

{

int current,i;

int count=0;

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

{

predecessor[i]=NIL;

length[i]=infinity;

status[i]=TEMP;

}

length[r]=0;

while(1)

{

current=min\_temp();

if(current==NIL)

{

if(count==n-1)

return;

else

{

cout<<"\nGraph is not connected,No spanning tree is possible";

exit(1);

}

}

status[current]=PERM;

if (current!=r)

{

count++;

tree[count].u=predecessor[current];

tree[count].v=current;

}

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

if (adj[current][i]>0 && status[i]==TEMP)

if (adj[current][i]<length[i])

{

predecessor[i]=current;

length[i]=adj[current][i];

}

}

}

int min\_temp()

{

int i;

int min=infinity;

int k=-1;

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

{

if(status[i]==TEMP && length[i]<min)

{

min=length[i];

k=i;

}

}

return k;

}

**Output :**

