**Title:** Binary Search using Divide and Conquer

#include <iostream>

using namespace std;

int main()

{

int array[10],i,search;

cout<<"enter array elements"<<endl;

for(i=0;i<10;i++)

{

cin>>array[i];

}

cout<<"enter element you want to search"<<endl;

int l=0;

int up=9;

int mid=(l+up)/2;

cin>>search;

while(l<=up)

{

if(search>array[mid])

{

l=mid+1;

}

else if(search==array[mid])

{

cout<<"element found"<<endl;

break;

}

else

{

up=mid-1;

}

mid=(l+up)/2;

}

if(l>up)

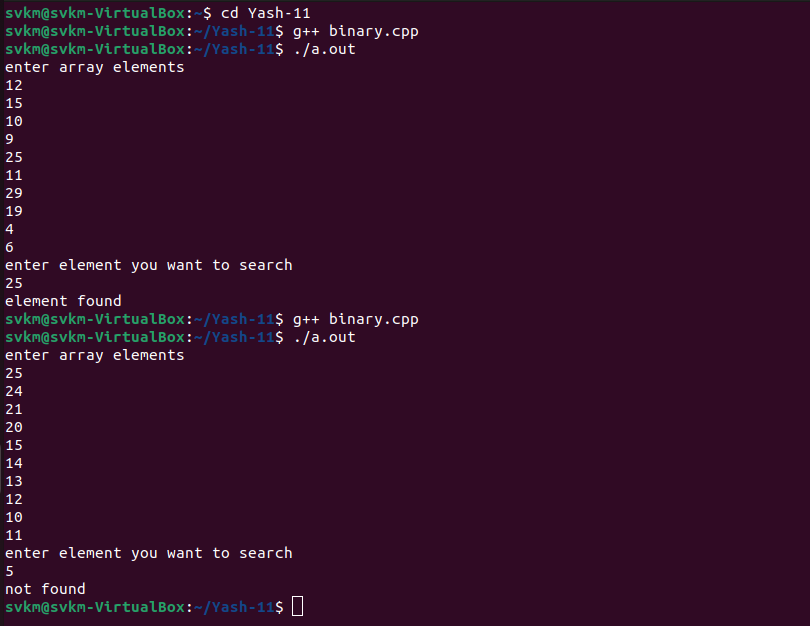
{

cout<<"not found"<<endl;

}

return 0;

}



**Title:** Merge Sort using Divide and Conquer

#include<iostream>

using namespace std;

#define max 100

void merge\_sort(int arr[],int low,int up);

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

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

int main()

{

int i,n,arr[max];

cout<<"enter the size of array:"<<endl;

cin>>n;

cout<<"enter array elements "<<endl;

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

{

cin>>arr[i];

}

merge\_sort(arr,0,n-1);

cout<<"sorted list is "<<endl;

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

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

return 0;

}

void merge\_sort(int arr[],int low,int up)

{

int mid;

int temp[max];

if(low<up)

{

mid=(low+up)/2;

merge\_sort(arr,low,mid); //left sublist

merge\_sort(arr,mid+1,up); //right sublist

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

copy\_s(arr,temp,low,up);

}

}

void merge\_s(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\_s(int arr[],int temp[],int low,int up)

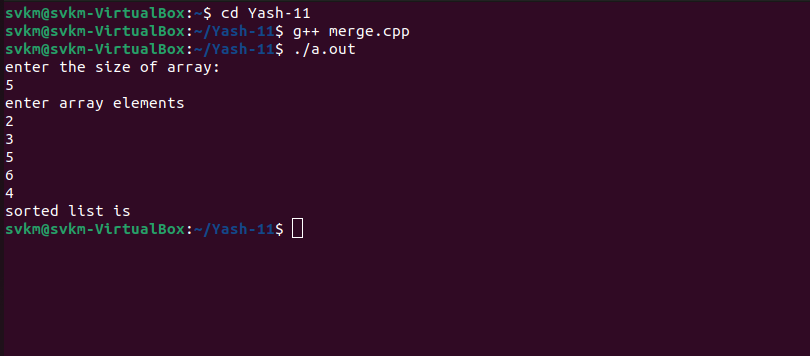
{

int i;

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

arr[i]=temp[i];

}



**Title:** Quick Sort using Divide and Conquer

#include<iostream>

using namespace std;

void quick(int a[], int l, int up);

int partition(int a[], int l, int up);

int main()

{

int n;

cout<<"enter the size of an array"<<endl;

cin>>n;

int arr[n];

cout<<"enter array elements"<<endl;

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

{

cin>>arr[i];

}

int low = 0;

int up = n-1;

quick(arr, low, up);

cout << "sorted elements are" << endl;

for (int i = 0; i <= up; i++) {

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

}

cout << endl;

return 0;

}

void quick(int a[], int l, int up) {

if (l >= up) {

return;

}

int pvtloc = partition(a, l, up);

quick(a, l, pvtloc - 1); //left sublist

quick(a, pvtloc + 1, up); //right sublist

}

int partition(int a[], int l, int up) {

if (l >= up) {

return l;

}

int temp, pvt;

int i = l + 1;

int j = up;

pvt = a[l];

while (i <= j) {

while (a[i] < pvt) {

i++;

}

while (a[j] > pvt) {

j--;

}

if (i < j) {

temp = a[i];

a[i] = a[j];

a[j] = temp;

i++;

j--;

} else {

i++;

}

}

// Swap pivot with element at position j

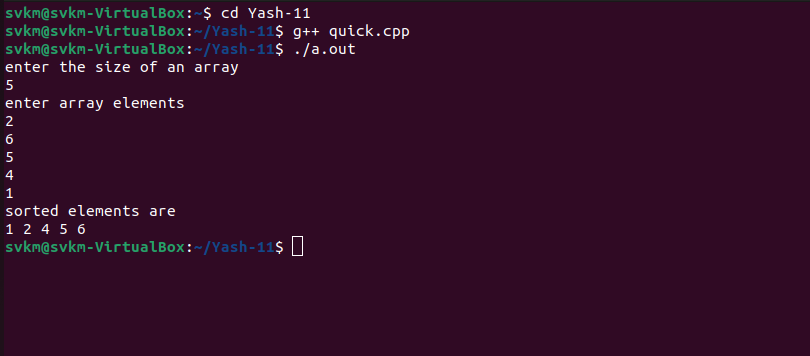
temp = a[l];

a[l] = a[j];

a[j] = temp;

return j;

}



**Title:** Strassen’s Matrix Multiplication using Divide and Conquer

#include <iostream>

using namespace std;

int main()

{

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

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

cout<<"enter the 4 elements of first matrix: "<<endl;

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

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

cin>>a[i][j];

cout<<"enter the 4 elements of Second matrix: "<<endl;

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

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

cin>>b[i][j];

cout<<"The First matrix"<<endl;

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

{

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

{

cout<<a[i][j];

}

cout<<endl;

}

cout<<"The second matrix"<<endl;

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

{

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

{

cout<<b[i][j];

}

cout<<endl;

}

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<<endl<<"the strassen matrix after multiplication is "<<endl;

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

{

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

{

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

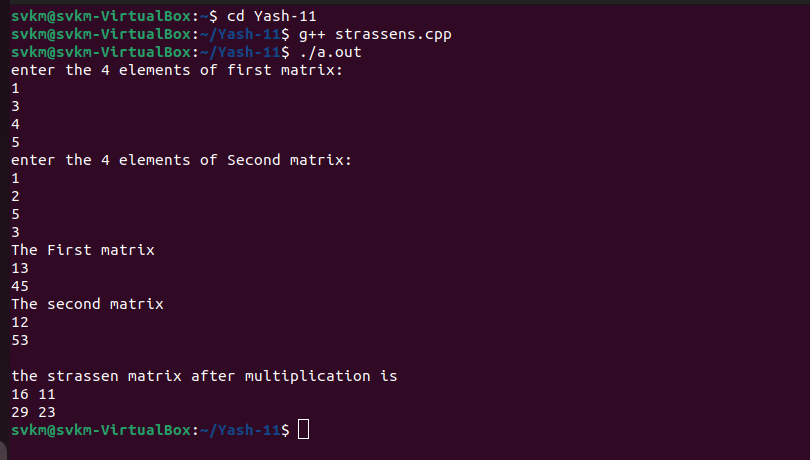
}

cout<<endl;

}

return 0;

}



**Title:** Fractional Knapsack Problem Using Greedy Method

#include<iostream>

using namespace std;

void knapsack(int n,float weight[],float profit[],float capacity);

int main()

{

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

int num,i,j;

float ratio[20],temp;

cout<<"enter the number of objects"<<endl;

cin>>num;

cout<<"enter the weights and profits of each object"<<endl;

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

{

cin>>weight[i]>>profit[i];

}

cout<<"enter the capacity of knapsack"<<endl;

cin>>capacity;

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

{

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

}

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

{

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

{

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

{

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

}

knapsack(num,weight,profit,capacity);

return 0;

}

void knapsack(int n,float weight[],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(weight[i]>u)

break;

else

{

x[i]=1.0;

tp=tp+profit[i];

u=u-weight[i];

}

}

if(i<n)

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

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

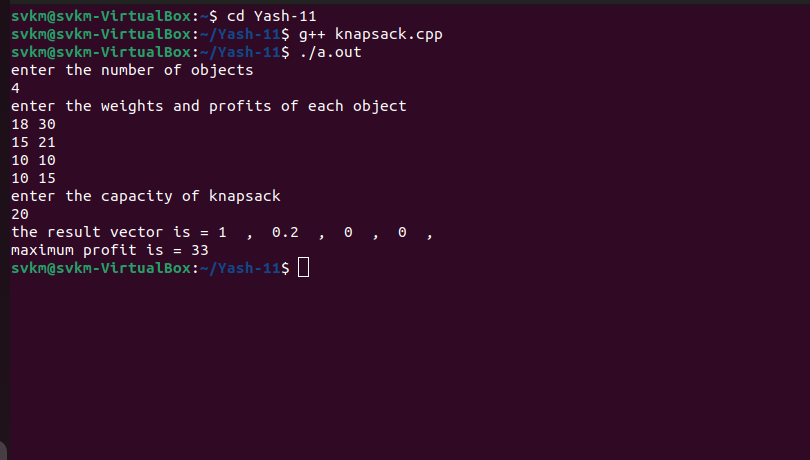
cout<<"the result vector is = ";

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

cout<<x[i]<<" , ";

cout<<endl<<"maximum profit is = "<<tp<<endl;

}



**Title:** Single Source Shortest Path Problem Dijkstra’s Algorithm

#include <iostream>

using namespace std;

#define max 100

#define infinity 9999

#define nil -1

#define temp 0

#define permanent 1

int adj[max][max];

int predecer[max];

int pathlength[max];

int status[max];

int create\_graph();

void djekstra(int src,int n);

int min\_path(int n);

int findpath(int s,int v);

int main()

{

int src,v;

int n=create\_graph();

cout<<"enter source vertex of graph"<<endl;

cin>>src;

djekstra(src,n);

while(1)

{

cout<<"enter destination vertex : -1 for exit"<<endl;

cin>>v;

if(v==-1)

break;

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

cout<<"this vertex does not exist"<<endl;

else if(v==src)

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

else if(pathlength[v]==infinity)

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

else

findpath(src,v);

}

return 0;

}

int create\_graph()

{

int n,max\_e,i,origin,destination,wt,j;

cout<<"enter vertices of graph"<<endl;

cin>>n;

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

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

{

cout<<"enter the origin and destination of graph"<<endl;

cin>>origin>>destination;

cout<<"enter the weight"<<endl;

cin>>wt;

adj[origin][destination]=wt;

}

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

{

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

{

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

}

cout<<endl;

}

return n;

}

void djekstra(int src,int n)

{

int i,current;

// 1 make all vertices temporary and initiliase pathlenght with infinity and predecer as nil

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

{

status[i]=temp;

pathlength[i]=infinity;

predecer[i]=nil;

}

// 2 make source vertex pathlenght is 0

pathlength[src]=0;

while(1)

{

//3 from all temporary vertices find min pathlengh of vertices make it current and permanent

current=min\_path(n);

if(current==nil)

return;

status[current]=permanent;

//from all adjacy temporary vertices from current

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

{

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

{

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

{

predecer[i]=current;

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

}

}

}

}

}

int min\_path(int n)

{

int i;

int min=infinity;

int k=nil;

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

{

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

{

min=pathlength[i];

k=i;

}

}

return k;

}

int findpath(int s,int v)

{

int i,u;

int path[max];

int shortdist=0;

int count=0;

while(v!=s)

{

count++;

path[count]=v;

u=predecer[v];

shortdist+=adj[u][v];

v=u;

}

count++;

path[count]=s;

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

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

{

cout<<path[i];

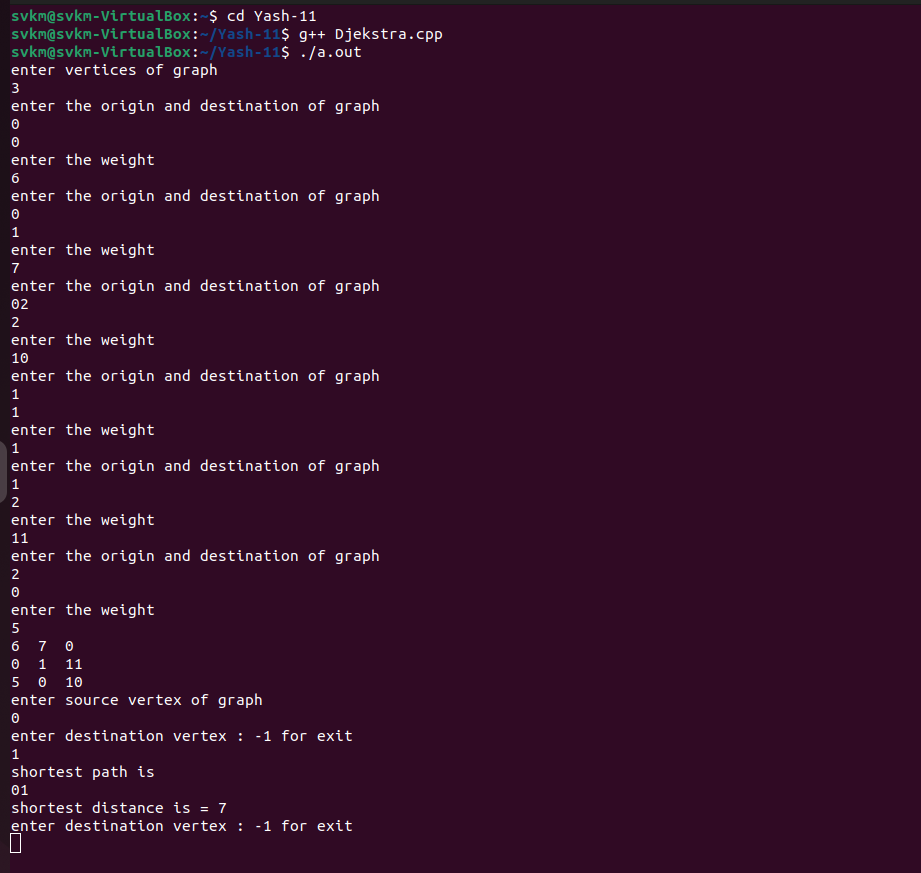
}

cout<<endl;

cout<<"shortest distance is = "<<shortdist<<endl;

return 0;

}



**Title:** Single Source Shortest Path Problem Bellman Ford Algorithm

#include <iostream>

using namespace std;

#define max 100

#define infinity 9999

#define nil -1

#define true 1

#define false 0

int n; //number of vertices in graph

int adj[max][max];

int predecessor[max];

int pathlength[max];

int ispresent\_in\_queue[max];

int queue[max];

int front,rear;

int create\_graph();

int bellmonford(int s);

void initilize\_queue();

void insert\_queue(int added\_item);

int is\_empty\_queue();

int delete\_queue();

int findpath(int s,int v);

int main()

{

int s,flag,v;

create\_graph();

cout<<"enter the source vertex"<<endl;

cin>>s;

flag=bellmonford(s);

if(flag==-1)

{

cout<<"ERRor : negative cycle in graph"<<endl;

exit(1);

}

while(1)

{

cout<<"enter destination vertex : -1 for exit"<<endl;

cin>>v;

if(v==-1)

break;

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

cout<<"this vertex does not exist"<<endl;

else if(v==s)

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

else if(pathlength[v]==infinity)

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

else

findpath(s,v);

}

return 0;

}

int create\_graph()

{

int max\_e,i,origin,destination,wt,j;

cout<<"enter vertices of graph"<<endl;

cin>>n;

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

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

{

cout<<"enter the origin and destination of graph"<<endl;

cin>>origin>>destination;

cout<<"enter the weight"<<endl;

cin>>wt;

adj[origin][destination]=wt;

}

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

{

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

{

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

}

cout<<endl;

}

return 0;

}

int bellmonford(int s)

{

int k=0,i,current;

// 1 initialise pathlength by infinity and predecerr is nil and not any vertex is present in queue

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

{

predecessor[i]=nil;

pathlength[i]=infinity;

ispresent\_in\_queue[i]=false;

}

initilize\_queue();

// 2 make path length of source vertex equal to 0 and insert it into queue

pathlength[s]=0;

insert\_queue(s);

ispresent\_in\_queue[s]=true;

while(!is\_empty\_queue())

{

// 3 delete the vertex from queue and make it current

current=delete\_queue();

ispresent\_in\_queue[current]=false;

if(s==current)

k++;

if(k>=n)

return -1; //negative cycle can be reachable form source vertex

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

{

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

{

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

{

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

predecessor[i]=current;

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

{

insert\_queue(i);

ispresent\_in\_queue[i]=true;

}

}

}

}

}

return 1;

}

void initilize\_queue()

{

int i;

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

{

queue[i]=0;

}

rear=-1;

front=-1;

}

void insert\_queue(int added\_item)

{

if(rear==max-1)

{

cout<<"queue is overflow"<<endl;

exit(1);

}

else

{

if(front==-1)

front=0;

rear+=1;

queue[rear]=added\_item;

}

}

int is\_empty\_queue()

{

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

return 1;

else

return 0;

}

int delete\_queue()

{

int d;

if(is\_empty\_queue())

{

cout<<"queue is underflow"<<endl;

exit(1);

}

else

{

d=queue[front];

front=front+1;

}

return d;

}

int findpath(int s,int v)

{

int i,u;

int path[max];

int shortdist=0;

int count=0;

while(v!=s)

{

count++;

path[count]=v;

u=predecessor[v];

shortdist+=adj[u][v];

v=u;

}

count++;

path[count]=s;

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

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

{

cout<<path[i];

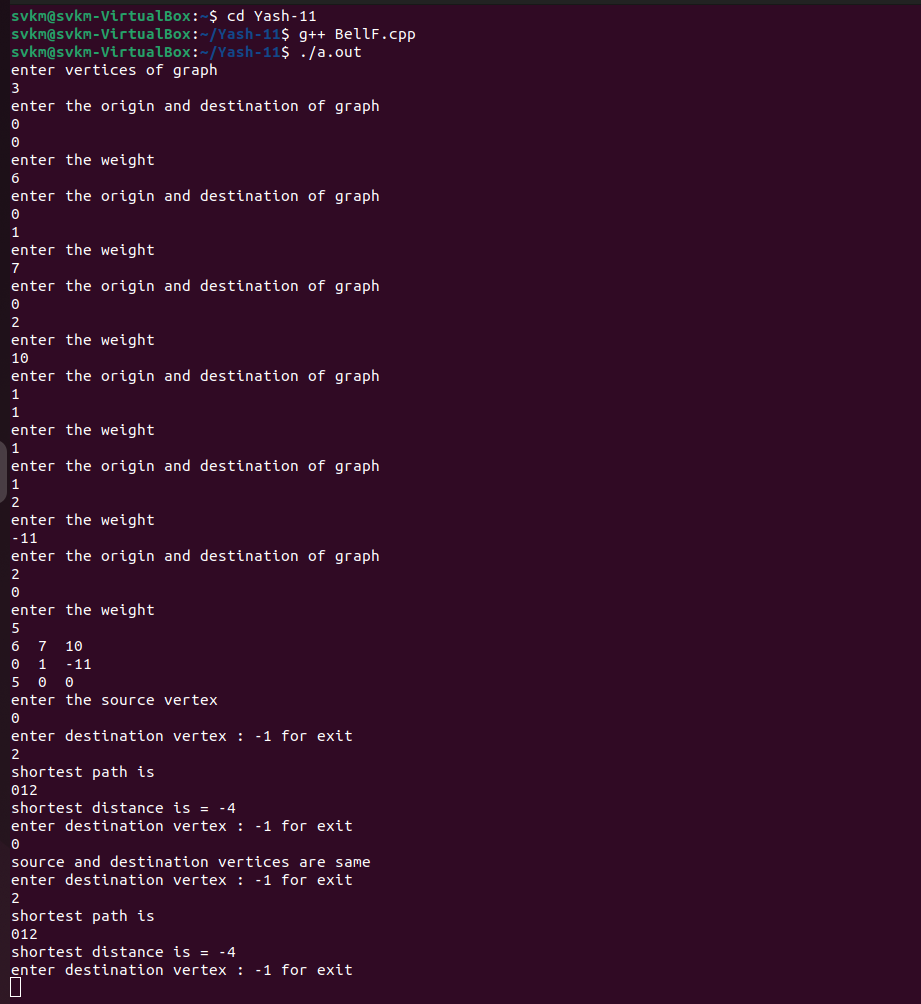
}

cout<<endl;

cout<<"shortest distance is = "<<shortdist<<endl;

return 0;

}



**Title:** Breadth First Search Using Queue

#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];

void create\_graph();

void BF\_Traversal ();

void BFS(int v);

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

void insert\_queue(int vertex);

int delete\_queue ();

int isEmpty\_queue();

int main()

{

create\_graph();

BF\_Traversal();

}

void create\_graph()

{

int max\_e,i,origin,destination,j;

cout<<"enter vertices of graph"<<endl;

cin>>n;

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

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

{

cout<<"enter the origin and destination of graph"<<endl;

cin>>origin>>destination;

adj[origin][destination]=1;

}

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

{

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

{

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

}

cout<<endl;

}

}

void BF\_Traversal()

{

int v;

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

state[v]=initial;

cout<<"enter starting vertex for breadth search"<<endl;

cin>>v;

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"<<endl;

else

{

if(front==-1)

front=0;

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"<<endl;

exit(1);

}

del\_item=queue[front];

front+=1;

return del\_item;

}

