----------code----------

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
struct edge{

    string v1="";
    string v2="";
    float weight=0;
};
struct calculator{
    string vertex="";
    string predesser="";
    float  value=INT_MAX;
};
 void input(edge e[],int m)
 {
    cout<<"Enter Edges Information:"<<endl;
    for(int i=0;i<m;i++)
    {
        cout<<"Enter starting vertex of edge:"<<endl;
        cin>>e[i].v1;
        cout<<"Enter ending vertex of edge:"<<endl;
        cin>>e[i].v2;
        cout<<"Enter corresponding Weight:"<<endl;
        cin>>e[i].weight;
    }
 }

// find path
// n=no of vertices.
// a is made in bellamn.this is string of vertex
 void findpath(calculator c[],int n,string a)
{

    for(int j=1;j<n;j++)
    {
        string advance_a="";
        //cout<<c[j].vertex<<"-->";
        advance_a=advance_a+c[j].vertex;
        int z=j;
        // here in place of c[0].vertex a[0] is not valid because a[0] is char and
        // c[0].predesser/vertes are string
        while(c[z].predesser!=c[0].vertex)
        {
            z=a.find(c[z].predesser);
            advance_a=advance_a+">--"+a[z];
        }
        advance_a=advance_a+">--"+a[0];
        // print string advance in reverse manner
        for(int i=advance_a.length()-1;i>=0;i--)
        {
            cout<<advance_a[i];
        }
```

```
        cout<<""<<endl;
    }
}

//n=number of vertex
//m=number of edges. i.e in vectore v
void belmanford(edge e[],calculator c[],set<string>s,int n,int m)
{
    input(e,m);
    bool t=false; //is to determine nagative edges cycle
    string a=""; // to find vertex in calculator array
    //a =string of vertices.(from set)

    for(int i=0;i<m;i++)
    {
        //to get  array of (only one  time)vertex
        s.insert(e[i].v1);
        s.insert(e[i].v2);
    }
    int j=0;
    for(auto i :s)
    {
        //put vertex to calculator array
        c[j].vertex=i;
        a=a+i;
        j++;
    }

    c[0].value=0;//initial vertex has 0 weight/value
    // c[0].predesser="null";
    for(int i=0;i<n-1;i++)//loop runs n-1 times by default it is rule
    {

        for(int k=0;k<m;k++)
        {
            int z1=a.find(e[k].v1);
            /*get index of v1 vertex from string to check
            value in calculator array so that  comparision can be done */
            int z=a.find(e[k].v2);
            if(e[k].weight+c[z1].value< c[z].value)
            {
                c[z].value=e[k].weight+c[z1].value;
                c[z].predesser=e[k].v1;
            }
        }
    }

    //to check negative edges cycle
    for(int k=0;k<m;k++)
    {
        int z1=a.find(e[k].v1);
        int z=a.find(e[k].v2);
        if(e[k].weight+c[z1].value< c[z].value)
        {
            // c[z].value=v[k].weight+c[z1].value;
```

```
            // c[z].predesser=v[k].v1;
            t=true;
        }
    }

    c[0].value=0;
    c[0].predesser="null";
    if(t==true)
    {
        cout<<"Graph contains nagative edges cycle"<<endl;

    }
    else
    {
        cout<<"vertex"<<"    "<<"predesser"<<"    "<<"value"<<endl;
        for(int i=0;i<n;i++)
        {
            cout<<c[i].vertex<<"    "<<c[i].predesser<<"    "<<c[i].value<<endl;
        }

        // print path
        cout<<"path across all vertices are:"<<endl;
        findpath(c,n,a);
}
    }



int main ()
{
    int n;
    cout<<"Enter total number of vertex:"<<endl;
    cin>>n;
    int m;
    cout<<"Enter total number of edges:"<<endl;
    cin>>m;
    edge  e[m];
    set <string>s;
    calculator c[n];
    belmanford(e,c,s,n,m);
    return 0;
}
```

----------test cases----------

```
1-->
Enter total number of vertex:
6
Enter total number of edges:
7
Enter Edges Information:
Enter starting vertex of edge:
a
Enter ending vertex of edge:
b
```

Enter corresponding Weight:
5
Enter starting vertex of edge:
b
Enter ending vertex of edge:
d
Enter corresponding Weight:
2
Enter starting vertex of edge:
b
Enter ending vertex of edge:
c
Enter corresponding Weight:
1
Enter starting vertex of edge:
c
Enter ending vertex of edge:
e
Enter corresponding Weight:
1
Enter starting vertex of edge:
d
Enter ending vertex of edge:
f
Enter corresponding Weight:
2
Enter starting vertex of edge:
e
Enter ending vertex of edge:
d
Enter corresponding Weight:
-1
Enter starting vertex of edge:
f
Enter ending vertex of edge:
e
Enter corresponding Weight:
-3

Graph contains nagative edges cycle

2-->
Enter total number of vertex:
5
Enter total number of edges:
9
Enter Edges Information:
Enter starting vertex of edge:
a
Enter ending vertex of edge:
b
Enter corresponding Weight:
4
Enter starting vertex of edge:
a

Enter ending vertex of edge:
c
Enter corresponding Weight:
2
Enter starting vertex of edge:
b
Enter ending vertex of edge:
c
Enter corresponding Weight:
3
Enter starting vertex of edge:
b
Enter ending vertex of edge:
d
Enter corresponding Weight:
2
Enter starting vertex of edge:
b
Enter ending vertex of edge:
e
Enter corresponding Weight:
3
Enter starting vertex of edge:
c
Enter ending vertex of edge:
e
Enter corresponding Weight:
5
Enter starting vertex of edge:
c
Enter ending vertex of edge:
d
Enter corresponding Weight:
4
Enter starting vertex of edge:
c
Enter ending vertex of edge:
b
Enter corresponding Weight:
1
Enter starting vertex of edge:
e
Enter ending vertex of edge:
d
Enter corresponding Weight:
-5

vertex predesser value
a null 0
b c 3
c a 2
d e 1
e b 6

path across all vertices are:
a-->c-->b
a-->c
a-->c-->b-->e-->d
a-->c-->b-->e