Homework5-FCOE7205

Dijkstra's algorithm using unsorted array for priority Q.

```
#include <iostream>

#include <vector>

#include <list>

#include <queue>

using namespace std;

#define INT_MAX 100000 //initializing the distance to infinity in beginning


typedef pair<int, int> component ;

class Node // Defining graph with vertices and its respective weight

  {

  int Vertex;

  list<pair<int, int>>* com; // to store vertices and weight

public:

  //Initializing a default constructor

  Node(int Vertex)

  {

    this->Vertex = Vertex;

    com = new list<component>[Vertex];

  }

  void addNode(int a, int b, int c);

  void shortdis(int dis);

};

// function to add edge oppositely

void Node::addNode(int a, int b, int c)

{

  com[a].push_back(make_pair(b, c));
```

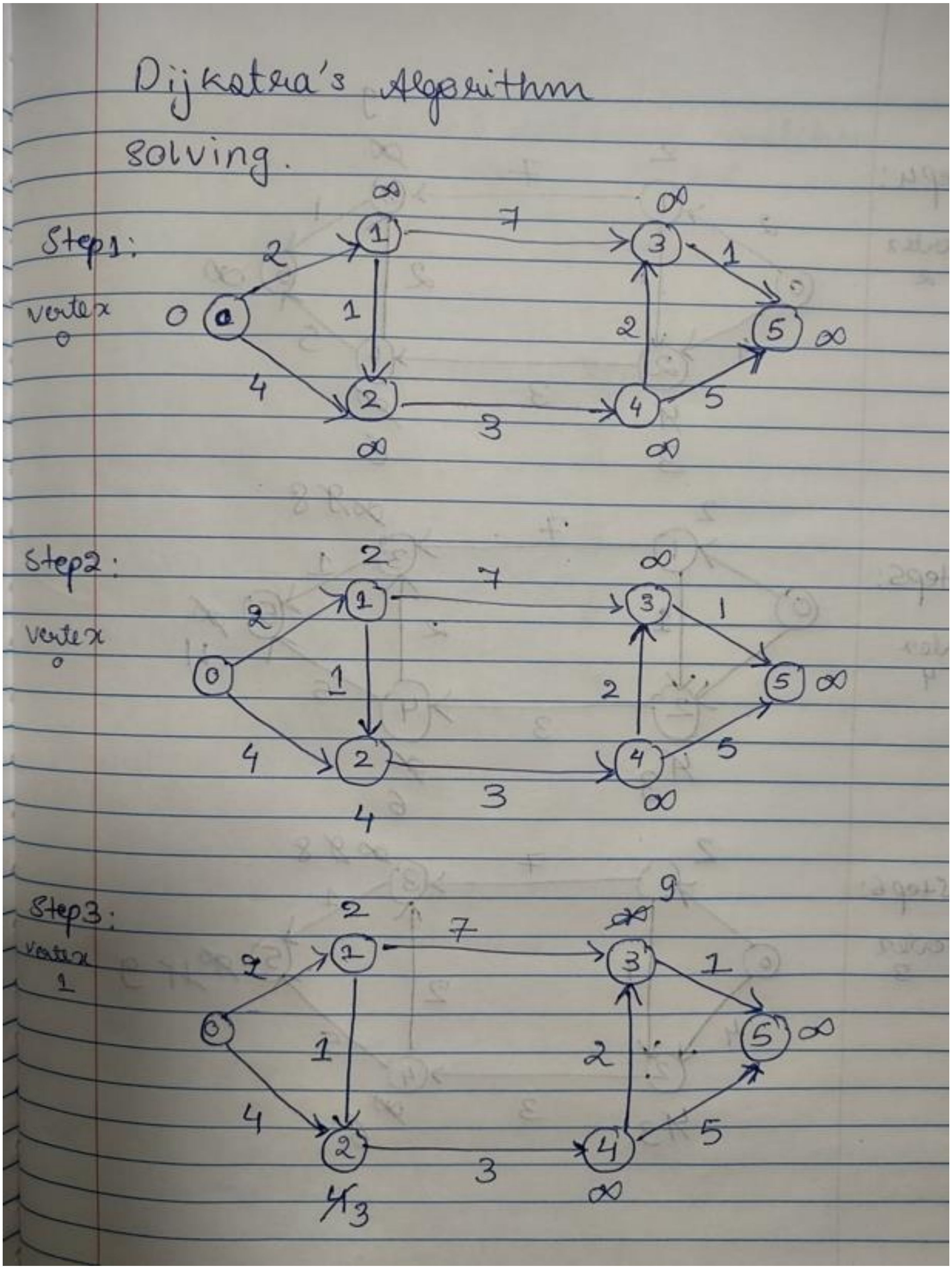
```
    com[b].push_back(make_pair(a, c));
}

void Node::shortdis(int source) // source vertex
{
    // Creating a priority queue
    priority_queue<component, vector<component>, greater<component>> qp;
    vector<int> cost(Vertex, INT_MAX);
    qp.push(make_pair(0, source)); // push operation in queue
    cost[source] = 0;
    while (!qp.empty())
    {   //Extracting min queue
        int a = qp.top().second;
        qp.pop();
        // 'i' gives adjacent vertices of each node in vertex
        list<pair<int, int>>::iterator i;
        for (i = com[a].begin(); i != com[a].end(); i++)
        //not going to the adjacent list
        {
            int b = (*i).first;
            int weight = (*i).second;

            // Check the shortest path
            if (cost[b] > cost[a] + weight)
            {
                // Updated distance
                cost[b] = cost[a] + weight;
                qp.push(make_pair(cost[b], b));
            }
```

```
        }
    }
    cout<<"Vertex \tDistance from Source \n";
    for (int i = 0; i < Vertex; i++)
        cout<<i<<"\t\t"<<cost[i]<<"\n";
}
int main()
{
    int Vertex ;
    cout<<" Input enter the number of the vertices ";
    cin>>Vertex;
    Node n(Vertex);
    n.addNode(0, 1, 2); // add root node with neighbor vertex and weight
    n.addNode(0, 2, 4);
    n.addNode(1, 2, 1);
    n.addNode(1, 3, 7);
    n.addNode(2, 4, 3);
    n.addNode(3, 5, 1);
    n.addNode(4, 3, 2);
    n.addNode(4, 5, 5);
    n.shortdis(0); // call the function to find shortest path of graph using Dijkstra algorithm
    return 0;
}
```

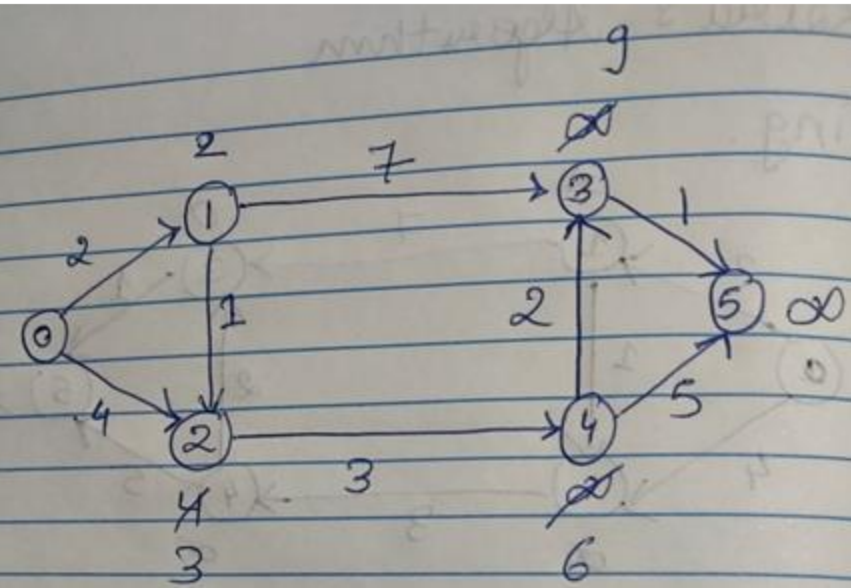
OUTPUTS :

```
 Input enter the number of the vertices 6
Vertex   Distance from Source
0                0
1                2
2                3
3                8
4                6
5                9
```

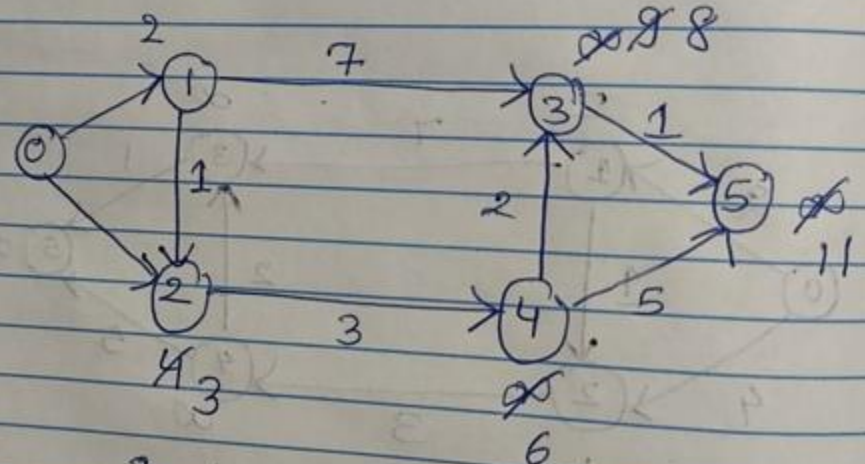
Solving Dijkstra's algorithm :



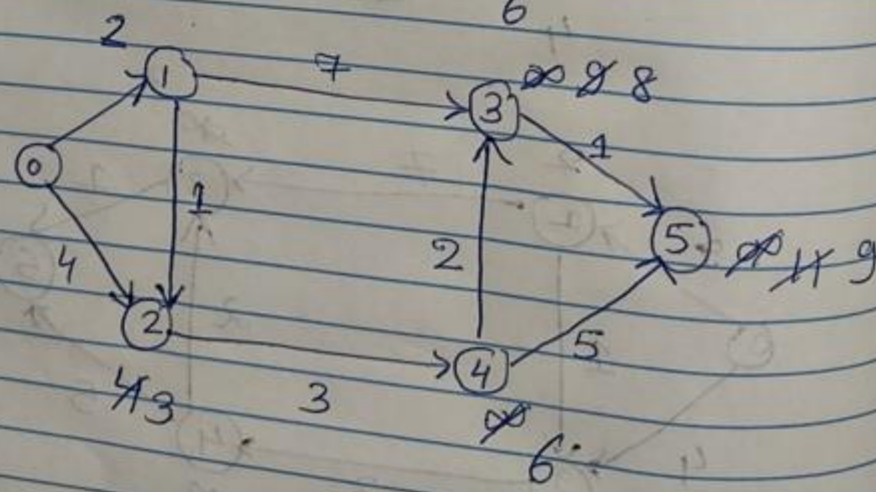
Step4:
Vertex 2



Step5:
Vertex 4



Step6:
Vertex 3



Hence the final distances matches as per the program output.

Vertex	Distance
0	0
1	2
2	3
3	8
4	6
5	9

Bellman ford algorithm

```
#include <iostream>

#include <climits>

#define Int_max 10000;

using namespace std;

 struct node

{

   int source;

   int destination;

   int cost;
```

```
};
struct graph{
    int v,e;
    struct node* n;
};
void BellmanFord(struct graph* g,int V,int E, int source)
{
   // int V= g->V;
   // int E= g->E;
    int distance[V];
    //step 1
    for (int i = 0; i < V; i++)
                        distance[i] = INT_MAX;
            distance[source] = 0;
            //step 2
            for (int i = 1; i <= V - 1; i++)
            {
                        for (int j = 0; j < E; j++)
                        {
                                    int a = g->n[j].source;
                                    int b = g->n[j].destination;
                                    int cost = g->n[j].cost;
                                    if (distance[a] != INT_MAX && distance[a] + cost < distance[b])
                                                distance[b] = distance[a] + cost;
                        }
            }
            //step3
            for (int i = 0; i < E; i++)
```

```
        {
                int c = g->n[i].source;
                int d = g->n[i].destination;
                int cost = g->n[i].cost;
                if (distance[c] != INT_MAX && distance[c] + cost < distance[d])
                {
                        cout<<"Graph contains negative weight cycle";
                        return; // If negative cycle is detected, simply return
                }
        }
        //print Graph
        cout<<"Vertex Distance from Source \n";
        for (int i = 0; i < V; ++i)
                printf("%d \t\t %d\n", i, distance[i]);
}
int main()
{
   int v,e;
   cout<<"Enter the  Number of vertices ";
   cin>>v;
   cout<<"Enter the number of edges ";
   cin>>e;
   struct graph* g = new graph;
   g->v = v;
        g->e = e;
        g->n = new node[e];
         g->n[0].source=0;
        g->n[0].destination = 1;
        g->n[0].cost = 6;
```

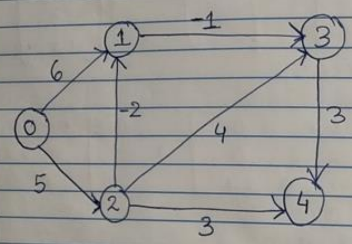
```
        g->n[1].source=0;
        g->n[1].destination = 2;
        g->n[1].cost = 5;
    g->n[2].source=1;
        g->n[2].destination = 3;
        g->n[2].cost = -1;
        g->n[3].source=2;
        g->n[3].destination = 1;
        g->n[3].cost = -2;
        g->n[4].source=2;
        g->n[4].destination = 3;
        g->n[4].cost = 4;
        g->n[5].source=2;
        g->n[5].destination = 4;
        g->n[5].cost = 3;
        g->n[6].source=3;
        g->n[6].destination = 4;
        g->n[6].cost = 3;
        BellmanFord(g,v,e,0);
        return 0;
}
```

OUTPUT :

```
Enter the  Number of vertices 5
Enter the number of edges 7
Vertex Distance from Source
0                 0
1                 3
2                 5
3                 2
4                 5
```

Solving of BellmanFord Algorithm :

Bellman-Ford Algorithm



→ Vertices

No of edges traversed ↑	1	2	3	4	5
1	0	6	5	∞	∞
2	0	3	5	5	8
3	0	3	5	2	8
4	0	3	5	2	5

$\text{Min} = [6, 0+6, 5+(-2), \infty+\infty, \infty+\infty]$
$= 3$

$\text{min} = [5, 0+\infty, 6+\infty, \infty+\infty, \infty+\infty]$
$= 5$

$\text{Min} = [\infty, 0+\infty, 6+(-1), 5+4, \infty+\infty]$
$= 5$

$\text{min} = [\infty, 0+\infty, 6+\infty, 5+3, \infty+3]$
$= 8$

$\text{Min} = [3, 0+6, 5+(-2), 5+\infty, 8+\infty]$
$= 3$

$\text{min} = [5, 0+\infty, 3+\infty, 5+\infty, 8+\infty]$
$= 5$

$\text{Min} = [5, 0+\infty, 3+(-1), 5+4, 8+\infty]$
$= 2$

$\text{Min} = [8, 0+\infty, 3+\infty, 5+3, 5+3]$
$= 8$

$\text{min} = [3, 0+6, 5+(-2), 2+\infty, 8+\infty]$
$= 3$

$\text{min} = [5, 0+\infty, 3+\infty, 2+\infty, 8+\infty]$
$= 5$

$$\min = [2, 0 + \infty, 3 + (-1), 5 + 4, 8 + \infty]$$
$$= 2$$

$$\min = [8, 0 + \infty, 3 + \infty, 8, 5]$$
$$= 5$$

Hence the output matches with program output too for shortest distance from every vertex to source.

Vertex	Distance
0	0
1	3
2	5
3	2
4	5