Name : Pratik Rajesh Jade

Roll no : A 70

Aim - From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in C++.

#include <iostream>

#include <limits>

using namespace **std**;

#define **MAXV** 1000

class **EdgeNode**{

    public:

        int key;

        int weight;

**EdgeNode** \*next;

**EdgeNode**(int, int);

};

**EdgeNode**::**EdgeNode**(int key, int weight){

    this->key = key;

    this->weight = weight;

    this->next = **NULL**;

}

class **Graph**{

    bool directed;

    public:

**EdgeNode** \*edges[**MAXV** + 1];

**Graph**(bool);

**~Graph**();

        void **insert\_edge**(int, int, int, bool);

        void **print**();

};

**Graph**::**Graph**(bool directed){

    this->directed = directed;

    for(int i = 1; i < (**MAXV** + 1); i ++){

        this->edges[i] = **NULL**;

    }

}

**Graph**::**~Graph**(){

*//to do*

}

void **Graph**::**insert\_edge**(int x, int y, int weight, bool directed){

    if(x > 0 && x < (**MAXV** + 1) && y > 0 && y < (**MAXV** + 1)){

**EdgeNode** \*edge = new **EdgeNode**(y, weight);

        edge->next = this->edges[x];

        this->edges[x] = edge;

        if(!directed){

**insert\_edge**(y, x, weight, true);

        }

    }

}

void **Graph**::**print**(){

    for(int v = 1; v < (**MAXV** + 1); v ++){

        if(this->edges[v] != **NULL**){

            cout **<<** "Vertex " **<<** v **<<** " has neighbors: " **<<** **endl**;

**EdgeNode** \*curr = this->edges[v];

            while(curr != **NULL**){

                cout **<<** curr->key **<<** **endl**;

                curr = curr->next;

            }

        }

    }

}

void **init\_vars**(bool discovered[], int distance[], int parent[]){

    for(int i = 1; i < (**MAXV** + 1); i ++){

        discovered[i] = false;

        distance[i] = **std**::**numeric\_limits**<int>::**max**();

        parent[i] = -1;

    }

}

void **dijkstra\_shortest\_path**(**Graph** \*g, int parent[], int distance[], int start){

    bool discovered[**MAXV** + 1];

**EdgeNode** \*curr;

    int v\_curr;

    int v\_neighbor;

    int weight;

    int smallest\_dist;

**init\_vars**(discovered, distance, parent);

    distance[start] = 0;

    v\_curr = start;

    while(discovered[v\_curr] == false){

        discovered[v\_curr] = true;

        curr = g->edges[v\_curr];

        while(curr != **NULL**){

            v\_neighbor = curr->key;

            weight = curr->weight;

            if((distance[v\_curr] + weight) < distance[v\_neighbor]){

                distance[v\_neighbor] = distance[v\_curr] + weight;

                parent[v\_neighbor] = v\_curr;

            }

            curr = curr->next;

        }

*//set the next current vertex to the vertex with the smallest distance*

        smallest\_dist = **std**::**numeric\_limits**<int>::**max**();

        for(int i = 1; i < (**MAXV** + 1); i ++){

            if(!discovered[i] && (distance[i] < smallest\_dist)){

                v\_curr = i;

                smallest\_dist = distance[i];

            }

        }

    }

}

void **print\_shortest\_path**(int v, int parent[]){

    if(v > 0 && v < (**MAXV** + 1) && parent[v] != -1){

        cout **<<** parent[v] **<<** " ";

**print\_shortest\_path**(parent[v], parent);

    }

}

void **print\_distances**(int start, int distance[]){

    for(int i = 1; i < (**MAXV** + 1); i ++){

        if(distance[i] != **std**::**numeric\_limits**<int>::**max**()){

            cout **<<** "Shortest distance from " **<<** start **<<** " to " **<<** i **<<** " is: " **<<** distance[i] **<<** **endl**;

        }

    }

}

int **main**(){

**Graph** \*g = new **Graph**(false);

    int parent[**MAXV** + 1];

    int distance[**MAXV** + 1];

    int start = 1;

    g->**insert\_edge**(1, 2, 4, false);

    g->**insert\_edge**(1, 3, 1, false);

    g->**insert\_edge**(3, 2, 1, false);

    g->**insert\_edge**(3, 4, 5, false);

    g->**insert\_edge**(2, 4, 3, false);

    g->**insert\_edge**(2, 5, 1, false);

    g->**insert\_edge**(4, 5, 2, false);

**dijkstra\_shortest\_path**(g, parent, distance, start);

*//print shortest path from vertex 1 to 5*

**print\_shortest\_path**(5, parent);

**print\_distances**(start, distance);

    delete g;

    return 0;

}

Output:

