```cpp

/*****************************************************************************
Author: Jose Eduardo Morales
shortestPath.cpp
Date: march 22, 2023
*****************************************************************************/

#include <iostream>
#include <iterator>
#include <limits.h>
#include <algorithm>
#include "shortestPath.h"
#include "graph.h"

template<typename T>
void printVector(const T& t) {
    std::copy(t.cbegin(), t.cend(), std::ostream_iterator<typename T::value_type>(std
    ::cout, " -> "));
}

shortestPath::shortestPath(){

}

shortestPath::shortestPath(Graph g):graph(g){
    // get graph
    int size = graph.V();
    validPath = false;
    distance = std::vector<int> (size, INT_MAX) ; //init vector with max dist
    checkDistance = std::vector<bool> (size,true);//init vector with true
}

int shortestPath::minDistanceIndex(std::vector<int> dist, std::vector<bool>
checkDistance, int size){
    int min = INT_MAX; //start min value at infinite
    int index;
    //find the minimun distance
    for (int i = 0; i < size; i++){
        if (dist.at(i) <= min && checkDistance.at(i)){
            min = dist.at(i);
            index = i;
        }
    }
    return index;
}

void shortestPath::calc(int x){
    //init variables
    validPath = false;
    int size = graph.V();
    distance.clear();
    checkDistance.clear();
    previousVertex.clear();
    route.clear();
    distance = std::vector<int> (size, INT_MAX) ;
    checkDistance = std::vector<bool> (size,true);
    previousVertex = std::vector<int> (size, -1);
    source = x;

    //set source distance to zero
    distance.at(x) = 0;
    for(int j = 0; j < size - 1; j++){ //checks once per vertex
        //choose vertex K = minimun distance
        int k = minDistanceIndex(distance, checkDistance, size);
        checkDistance.at(k) = false; //processed
        for (int i = 0; i < size; i++){ //check all nodes connecter to k
            //checks if node is a valid minimun connetion
            if(distance.at(k) != INT_MAX &&  checkDistance.at(i) &&
            distance.at(k) + graph.getEdge(k,i) < distance.at(i) &&
            graph.getEdge(k,i) != 0){
                //update minimun distance
                distance.at(i) = distance.at(k) + graph.getEdge(k,i);
                //save previous vertex to later recreate shortest path
```

```
                previousVertex.at(i) = k;
            }
        }

    }
    return;
}

bool shortestPath::connected(int y){
    return (distance.at(y) < INT_MAX);
}


int shortestPath::dist(int y){
    if (connected(y))
        return distance.at(y);
    else
        return -1;
}

std::vector<int> shortestPath::path (int y){
    validPath = false;
    route.clear();
    if (connected(y)){ //checks if vertex y is connected to source x
        int i = y;
        while (i != source){ //loop back from vertex y all the way to source
            route.push_back(i); // save route
            i = previousVertex.at(i);
        }
        route.push_back(source); //save source vertex
        std::reverse(route.begin(), route.end());   //reverse the order
        validPath = true;
        return route;
    }
    else{
        return std::vector<int> (0);
    }
}

void shortestPath::printPath() {
    if (validPath){
        for (int i : route){
            std::cout << "n" << i << " -> ";
        }
        std::cout << "\b\b\b   " ; // delete last ->
    }
    else{
        std::cout << "no valid path ";// << std::endl;
    }
}

void shortestPath::printAllPaths() {
    connectedVert = 0;
    totalDist = 0;
    std::cout << "DISTANCES FROM n" << source << std::endl;
    std::cout << "Vertex \tDist \tPath" << std::endl;
    for (int i = 0; i < graph.V(); i++){
        std::cout << "n" << i << "\t";
        if (connected(i)){
            totalDist += dist(i);
            std::cout << dist(i) << "\t";
            std::vector<int> ipath = path(i);
            printPath();
            std::cout << std::endl;
            connectedVert++;
        }
        else{
            std::cout << "not connected" << std::endl;
        }

    }
}
float shortestPath::avgDist(){
```

```
145         return static_cast<float>(totalDist) / connectedVert;
146     }
```