EXAMEN EDNL JUNIO 2024 RESUELTO (GRAFOS)

ENUNCIADO:

El archipiélago de Grecoland (Zuelandia) está formado únicamente por **tres** islas, Fobos, Deimos y **Europa**, que tienen N1, N2 y N3 ciudades, respectivamente, de las cuales C1, C2 y C3 ciudades son costeras (obviamente $C1 \leq N1$, $C2 \leq N2$ y $C3 \leq N3$). Se dispone de las coordenadas cartesianas (x, y) de todas y cada una de las ciudades del archipiélago.

El huracán Isadore acaba de devastar el archipiélago, con lo que todas las carreteras y puentes construidos en su día han desaparecido. En esta terrible situación se pide ayuda a la ONU, que acepta reconstruir el archipiélago (es decir volver a comunicar todas las ciudades del archipiélago) siempre que se haga al mínimo coste. De cara a poder comparar costes de posibles reconstrucciones se asume lo siguiente:

1. El coste de construir cualquier carretera o cualquier puente es proporcional a su longitud (distancia euclídea entre las poblaciones de inicio y fin de la carretera o del puente).
2. Cualquier puente que se construya siempre será **más barato** que cualquier carretera que se construya.

De cara a poder calcular los costes de VIAJAR entre cualquier ciudad del archipiélago se considerará lo siguiente:

1. El coste directo de viajar, es decir de utilización de una carretera o de un puente, coincidirá con su longitud (distancia euclídea entre las poblaciones origen y destino de la carretera o del puente).

En estas condiciones, implementa un subprograma que calcule el coste mínimo de viajar entre dos ciudades de Grecoland, origen y destino, después de haberse reconstruido el archipiélago, dados los siguientes datos:

1. Lista de ciudades de Fobos representadas mediante sus coordenadas cartesianas.
2. Lista de ciudades de Deimos representadas mediante sus coordenadas cartesianas.
3. Lista de ciudades de Europa representadas mediante sus coordenadas cartesianas.
4. Lista de ciudades costeras de Fobos.
5. Lista de ciudades costeras de Deimos.
6. Lista de ciudades costeras de Europa.
7. Ciudad origen y destino del viaje.

AVISO: La solución que se propone más adelante PUEDE QUE NO SEA 100% CORRECTA, pero es una buena aproximación de lo que se pedía en el examen, al ser muy similar a un ejercicio de prácticas de la asignatura. Es por ello por lo que recomiendo que la uses a modo orientativo. No me hago responsable de los posibles ceros de DLH por copiar justo esto si cayera de nuevo.

```cpp
/*
AVISO: PUEDE QUE NO SEA 100% CORRECTO NI EL MÉTODO MÁS EFICIENTE
Es el inverso al EJ7 P8 de EDNL

En este caso, como nos dicen que todos los puentes son más baratos que todas las carreteras
debemos buscar el puente más caro (que aún así será más barato que cualquier carretera), sumar su coste
a todas las distancias de las carreteras que conectan pares de ciudades, incluídas las costeras
TAMBIÉN SE PUEDE HACER CON UN APOMAX, PERO HABRÍA QUE ESCRIBIR EL TAD ENTERO EN EL EXAMEN

Posteriormente, con la cantidad sumada a los costes del supergrafo, hacemos Kruskall

Ahora restamos esa cantidad del puente para normalizar de nuevo los costes y así poder viajar adecuadamente
Colocamos los dos puentes más baratos que conecten dos islas distintas en el grafo

Finalmente hacemos Dijkstra con la ciudad origen y destino, con todo construido
*/

#include "../Material para las prácticas de grafos/alg_grafoPMC.h"
#include "../Material para las prácticas de grafos/alg_grafo_E-S.h"
#include "../Material para las prácticas de grafos/grafoPMC.h"
#include <cmath>
#include <vector>

using namespace std;

typedef struct{
    double coordX, coordY;
}ciudad;

// ES IGUAL DE VÁLIDO REPRESENTAR LAS COSTERAS CON UN VECTOR DE BOOLS
double viajeGrecolandTodo(vector<ciudad> cDeimos, vector<ciudad> cFobos, vector<ciudad> cEuropa
                          vector<ciudad> costerasDeimos, vector<ciudad> costerasFobos, vector<ciudad> costerasEuropa
                          ciudad orig, ciudad dest)
{
    typedef typename GrafoP<tCoste>::vertice vert;
    GrafoP<double> super(cDeimos.size() + cFobos.size() + cEuropa.size());

    // Buscamos el puente más LARGO entre las islas costeras
    size_t minD, minF, minE;
    // Tomamos como referencia la distancia entre la primera ciudad costera de Deimos y Fobos
    double distPuenteMasLargo = sqrt(pow(costerasDeimos[0].coordX - costerasFobos[0].coordX, 2) + pow(costerasDeimos[0].coordY - costerasFobos[0].coordY, 2));

    // Buscamos si está entre Fobos y Deimos
    for (size_t i = 0; i < costerasDeimos.size(); i++)
    {
        for (size_t j = 0; j < costerasFobos.size(); j++)
        {
            // Distancia entre la costera de Deimos y la de Fobos de turno
            double distFB = sqrt(pow(costerasDeimos[i].coordX - costerasFobos[j].coordX, 2) + pow(costerasDeimos[i].coordY - costerasFobos[j].coordY, 2));

            if (distFD >= distPuenteMasLargo)
            {
                distPuenteMasLargo = distFD; // Coste
                minD = i;                    // Índice ciudad costera
                minF = j;                    // Índice ciudad costera

                super[minD][minF + cDeimos.size()] = super[minF + cDeimos.size()][minD] = distPuenteMasLargo;
            }
        }
    }

    // Buscamos si está entre Europa y Deimos
    for (size_t i = 0; i < costerasDeimos.size(); i++)
    {
        for (size_t j = 0; j < costerasEuropa.size(); j++)
        {
            // Distancia entre la costera de Deimos y la de Europa de turno
            double distED = sqrt(pow(costerasDeimos[i].coordX - costerasEuropa[j].coordX, 2) + pow(costerasDeimos[i].coordY - costerasEuropa[j].coordY, 2));

            if (distED >= distPuenteMasLargo)
            {
                distPuenteMasLargo = distED; // Coste
                minD = i;                    // Índice ciudad costera
                minE = j;                    // Índice ciudad costera

                super[minD][minE + cDeimos.size() + cFobos.size()] = super[minE + cDeimos.size() + cFobos.size()][minD] = distPuenteMasLargo;
            }
        }
    }

    // Buscamos si está entre Europa y Fobos
    for (size_t i = 0; i < costerasEuropa.size(); i++)
    {
        for (size_t j = 0; j < costerasFobos.size(); j++)
        {
            // Distancia entre la costera de Europa y la de Fobos de turno
            double distEF = sqrt(pow(costerasEuropa[i].coordX - costerasFobos[j].coordX, 2) + pow(costerasEuropa[i].coordY - costerasFobos[j].coordY, 2));

            if (distEF >= distPuenteMasLargo)
            {
                distPuenteMasLargo = distEF; // Coste
                minE = i;                    // Índice ciudad costera
                minF = j;                    // Índice ciudad costera

                super[minF][minE + cDeimos.size() + cFobos.size()] = super[minE + cDeimos.size() + c.Fobos.size()][minF] = distPuenteMasLargo;
            }
        }
    }
```

```cpp
// Rellenamos el supergrafo con la distancia entre las ciudades de cada isla (diagonal matriz 3x3) sumando el puente más largo
// DEIMOS (1º Cuadrante)
for (size_t i = 0; i < cDeimos.size(); i++)
{
    for (size_t j = 0; j < cDeimos.size(); j++)
        super[i][j] = sqrt(pow(cDeimos[i].coordX - cDeimos[j].coordX, 2) + pow(cDeimos[i].coordY - cDeimos[j].coordY, 2)) + distPuenteMasLargo;
}

// FOBOS (5º Cuadrante)
for (size_t i = cDeimos.size(); i < cDeimos.size() + cFobos.size(); i++)
{
    for (size_t j = cDeimos.size(); j < cDeimos.size() + cFobos.size(); j++)
        super[i][j] = sqrt(pow(cFobos[i].coordX - cFobos[j].coordX, 2) + pow(cFobos[i].coordY - cFobos[j].coordY, 2)) + distPuenteMasLargo;
}

// EUROPA (9º Cuadrante)
for (size_t i = cDeimos.size() + cFobos.size(); i < cDeimos.size() + cFobos.size() + cEuropa.size(); i++)
{
    for (size_t j = cDeimos.size() + cFobos.size(); j < cDeimos.size() + cFobos.size() + cEuropa.size(); j++)
        super[i][j] = sqrt(pow(cEuropa[i].coordX - cEuropa[j].coordX, 2) + pow(cEuropa[i].coordY - cEuropa[j].coordY, 2)) + distPuenteMasLargo;
}


// Reconstruímos el archipiélago entero
GrafoP<double> Grecoland = Kruskall(super);

// Ahora hay que normalizar, restamos el peso del puente más largo para hacer Dijktra
// DEIMOS (1º Cuadrante)
for (size_t i = 0; i < cDeimos.size(); i++)
{
    for (size_t j = 0; j < cDeimos.size(); j++)
        Grecoland[i][j] -= distPuenteMasLargo;
}

// FOBOS (5º Cuadrante)
for (size_t i = cDeimos.size(); i < cDeimos.size() + cFobos.size(); i++)
{
    for (size_t j = cDeimos.size(); j < cDeimos.size() + cFobos.size(); j++)
        Grecoland[i][j] -= distPuenteMasLargo;
}

// EUROPA (9º Cuadrante)
for (size_t i = cDeimos.size() + cFobos.size(); i < cDeimos.size() + cFobos.size() + cEuropa.size(); i++)
{
    for (size_t j = cDeimos.size() + cFobos.size(); j < cDeimos.size() + cFobos.size() + cEuropa.size(); j++)
        Grecoland[i][j] -= distPuenteMasLargo;
}
```

```
150     // DLH quiere que tanto el origen como destino sean abstraidos como ciudades, por tanto,
151     // Buscamos los índices de orig y dest para pasárselo a Dijkstra
152     int indOrig, indDest;
153     bool encontOrig = false;
154     bool encontDest = false;
155 
156     // Primero buscamos en Deimos
157     for (size_t i = 0; i < cDeimos.size() && (!encontOrig || !encontDest); i++)
158     {
159         if (cDeimos[i].coordX == orig.coordX && cDeimos[i].coordY == orig.coordY)
160         {
161             indOrig = i;
162             encontOrig = true;
163         }
164         if (cDeimos[i].coordX == dest.coordX && cDeimos[i].coordY == dest.coordY)
165         {
166             indDest = i;
167             encontDest = true;
168         }
169     }
170 
171     // Luego buscamos en Fobos
172     for (size_t i = 0; i < cFobos.size() && (!encontOrig || !encontDest); i++)
173     {
174         if (cFobos[i].coordX == orig.coordX && cFobos[i].coordY == orig.coordY)
175         {
176             indOrig = i + cDeimos.size();
177             encontOrig = true;
178         }
179         if (cFobos[i].coordX == dest.coordX && cFobos[i].coordY == dest.coordY)
180         {
181             indDest = i + cDeimos.size();
182             encontDest = true;
183         }
184     }
185 
186     // Y por último en Europa
187     for (size_t i = 0; i < cEuropa.size() && (!encontOrig || !encontDest); i++)
188     {
189         if (cEuropa[i].coordX == orig.coordX && cEuropa[i].coordY == orig.coordY)
190         {
191             indOrig = i + cDeimos.size() + cEuropa.size();
192             encontOrig = true;
193         }
194         if (cEuropa[i].coordX == dest.coordX && cEuropa[i].coordY == dest.coordY)
195         {
196             indDest = i + cDeimos.size() + cEuropa.size();
197             encontDest = true;
198         }
199     }
200 
201     // Por fin aplicamos Dijkstra y devolvemos el coste del viaje (desde orig hasta dest)
202     vector<vert> uVert(Grecoland.numVert());
203     vector<double> coste = Dijkstra(Grecoland, indOrig, uVert);
204 
205     return coste[indDest];
206 
207 }
```