solucion.pdf

GIInfo

Estructuras de Datos no Lineales

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

La capital de Zuelandia está alcanzando niveles de toxicidad muy elevados, por ello se ha decretado el cierre a la ciudad como paso de tránsito hacia otras ciudades. (Para ir de una ciudad a otra no se podrá pasar por C.Zuelandia, pero si se podrá ir si es residente de la misma empleándola como ciudad destino u origen).

Implemente un subprograma que dada la capital y un grafo ponderado con los km de las carreteras existentes entre las ciudades del país, nos devuelva los caminos resultantes de nuestra nueva política "Sin pasar por la capital, por favor".

Nota impotante: Se ha de definir todos los tipos de dato, prototipo de las operaciones empleadas en los TADs y también los prototipos de los grafos vistos en clase que se empleen.

MAIN.CPP

```
#include <iostream>
#include "alg_grafoPMC.h"
#include "alg_grafo_E-S.h"
#include "examen.hpp"
using namespace std;
typedef unsigned int tCoste;
int main()
{
   GrafoP<tCoste> Zuelandia("examen.txt");
   GrafoP<tCoste>::vertice capital = 0;
   matriz<typename GrafoP<tCoste>::vertice> resul;
   resul = nuevosCaminos(Zuelandia,capital);
   cout << resul << endl;
   return 0;
}
```

EXAMEN.HPP

```
#include "alg_grafoPMC.h"

template <typename tCoste>

matriz<typename GrafoP<tCoste>::vertice> nuevosCaminos(const GrafoP<tCoste>& G, typename GrafoP<tCoste>::vertice capital)

{ matriz<typename GrafoP<tCoste>::vertice> caminos; //Aqui guardaremos los caminos  result.

typename GrafoP<tCoste>:: vertice v;

 //Hacemos una copia del grafo

 GrafoP<tCoste> g(G);

   //Ponemos a infinito  desde el origen a cualquier destino y desde cualquier destino al origen

   for(v = 0; v<g.numVert(); v++)

       g[capital][v] = g[v][capital] =  GrafoP<tCoste>::INFINITO;

   //Ahora que tenemos 'aislada' a la capital, aplicamos Floyd para saber los caminos de coste minimo entre cada ciudad

   matriz<tCoste> f = Floyd(g,caminos);

   //ESTO DE ABAJO SE HACE PARA CONTROLAR LOS VIAJES DE LOS REISDENTES

   //Ahora al grafo Original le aplicamos Dijkstra para averiguar el coste de los caminos mas cortos desde la capital a cualquier destino.

   //Nos interesa quedarnos con la P1, donde encontraremos los caminos y asi luego modificar nuestra matriz resultante.

   vector<typename GrafoP<tCoste>::vertice> P1;

   vector<tCoste> v1 = Dijkstra(G,capital,P1);

   //Del mismo modo aplicamos Dijkstra inversa para averiguar lo mismo que antes pero esta
//vez desde cualquier destino a la capital.

   vector<typename GrafoP<tCoste>::vertice> P2;

   vector<tCoste> v2 = DijkstraInv(G,capital,P2);

   //Ahora solo copiamos estos dos vectores en la matriz 'caminos'

   for(v = 0; v < G.numVert(); v++)

   {  caminos[capital][v] = P1[v];

        caminos[v][capital] = P2[v]; }

return caminos;

 }
```