

EDNL - Examen Junio 2024 RESUELTO...



FranVi_10range



Estructuras de Datos no Lineales



2º Grado en Ingeniería Informática



Escuela Superior de Ingeniería
Universidad de Cádiz

MÁSTER

Inteligencia Artificial & Data Management

MADRID

Conquista el mundo de la IA
en 10 meses



Ahora
25%
DE DESCUENTO

Aprenderás:

- Datos a IA generativa
- Big Data, ML, LLMs
- MLOps + cloud
- Visión estratégica

EOI Escuela de
organización
industrial



Info y descuentos

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

EXAMEN EDNL JUNIO 2024 RESUELTO (GRAFOS)

ENUNCIADO:

El archipiélago de Grecoland (Zuelandia) está formado únicamente por **tres** islas, Fobos, Deimos y **Europa**, que tienen N_1 , N_2 y N_3 ciudades, respectivamente, de las cuales C_1 , C_2 y C_3 ciudades son costeras (obviamente $C_1 \leq N_1$, $C_2 \leq N_2$ y $C_3 \leq N_3$). 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.

WUOLAH

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.

```
1  /*
2  AVISO: PUEDE QUE NO SEA 100% CORRECTO NI EL MÉTODO MÁS EFICIENTE
3  Es el inverso al E7 P8 de EDNL
4
5  En este caso, como nos dicen que todos los puentes son más baratos que todas las carreteras
6  debemos buscar el puente más caro (que aún así será más barato que cualquier carretera), sumar su coste
7  a todas las distancias de las carreteras que conectan pares de ciudades, incluidas las costeras
8  TAMBIÉN SE PUEDE HACER CON UN APOMAX, PERO HABRÍA QUE ESCRIBIR EL TAD ENTERO EN EL EXAMEN
9
10 Posteriormente, con la cantidad sumada a los costes del supergrafo, hacemos Kruskall
11
12 Ahora restamos esa cantidad del puente para normalizar de nuevo los costes y así poder viajar adecuadamente
13 Colocamos los dos puentes más baratos que conecten dos islas distintas en el grafo
14
15 Finalmente hacemos Dijkstra con la ciudad origen y destino, con todo construido
16 */
17
18 #include "../Material para las prácticas de grafos/alg_grafoPMC.h"
19 #include "../Material para las prácticas de grafos/alg_grafo_E-S.h"
20 #include "../Material para las prácticas de grafos/grafaPMC.h"
21 #include <cmath>
22 #include <vector>
23
24 using namespace std;
25
26 typedef struct{
27     double coordX, coordY;
28 }ciudad;
29
30 // ES IGUAL DE VÁLIDO REPRESENTAR LAS COSTERAS CON UN VECTOR DE BOOLS
31 double viajeGrecolandTodo(vector<ciudad> cDeimos, vector<ciudad> cFobos, vector<ciudad> cEuropa,
32     vector<ciudad> costerasDeimos, vector<ciudad> costerasFobos, vector<ciudad> costerasEuropa,
33     ciudad orig, ciudad dest)
34 {
35     typedef typename GrafoP<tCoste>::vertice vert;
36     GrafoP<double> super(cDeimos.size() + cFobos.size() + cEuropa.size());
37
38     // Buscamos el puente más LARGO entre las islas costeras
39     size_t minD, minF, minE;
40     // Tomamos como referencia la distancia entre la primera ciudad costera de Deimos y Fobos
41     double distPuenteMasLargo = sqrt(pow(costerasDeimos[0].coordX - costerasFobos[0].coordX, 2) + pow(costerasDeimos[0].coordY - costerasFobos[0].coordY, 2));
42
43     // Buscamos si está entre Fobos y Deimos
44     for (size_t i = 0; i < costerasDeimos.size(); i++)
45     {
46         for (size_t j = 0; j < costerasFobos.size(); j++)
47         {
48             // Distancia entre la costera de Deimos y la de Fobos de turno
49             double distFB = sqrt(pow(costerasDeimos[i].coordX - costerasFobos[j].coordX, 2) + pow(costerasDeimos[i].coordY - costerasFobos[j].coordY, 2));
50
51             if (distFB >= distPuenteMasLargo)
52             {
53                 distPuenteMasLargo = distFB; // Coste
54                 minD = i; // Índice ciudad costera
55                 minF = j; // Índice ciudad costera
56
57                 super[minD][minF + cDeimos.size()] = super[minF + cDeimos.size()][minD] = distPuenteMasLargo;
58             }
59         }
60     }
61
62     // Buscamos si está entre Europa y Deimos
63     for (size_t i = 0; i < costerasDeimos.size(); i++)
64     {
65         for (size_t j = 0; j < costerasEuropa.size(); j++)
66         {
67             // Distancia entre la costera de Deimos y la de Europa de turno
68             double distED = sqrt(pow(costerasDeimos[i].coordX - costerasEuropa[j].coordX, 2) + pow(costerasDeimos[i].coordY - costerasEuropa[j].coordY, 2));
69
70             if (distED >= distPuenteMasLargo)
71             {
72                 distPuenteMasLargo = distED; // Coste
73                 minD = i; // Índice ciudad costera
74                 minE = j; // Índice ciudad costera
75
76                 super[minD][minE + cDeimos.size() + cFobos.size()] = super[minE + cDeimos.size() + cFobos.size()][minD] = distPuenteMasLargo;
77             }
78         }
79     }
80
81     // Buscamos si está entre Europa y Fobos
82     for (size_t i = 0; i < costerasEuropa.size(); i++)
83     {
84         for (size_t j = 0; j < costerasFobos.size(); j++)
85         {
86             // Distancia entre la costera de Europa y la de Fobos de turno
87             double distEF = sqrt(pow(costerasEuropa[i].coordX - costerasFobos[j].coordX, 2) + pow(costerasEuropa[i].coordY - costerasFobos[j].coordY, 2));
88
89             if (distEF >= distPuenteMasLargo)
90             {
91                 distPuenteMasLargo = distEF; // Coste
92                 minE = i; // Índice ciudad costera
93                 minF = j; // Índice ciudad costera
94
95                 super[minF][minE + cDeimos.size() + cFobos.size()] = super[minE + cDeimos.size() + cFobos.size()][minF] = distPuenteMasLargo;
96             }
97         }
98     }
99 }
```



```

101 // Rellenamos el supergrafo con la distancia entre las ciudades de cada isla (diagonal matriz 3x3) sumando el puente más largo
102 // DEIMOS (1º Cuadrante)
103 for (size_t i = 0; i < cDeimos.size(); i++)
104 {
105     for (size_t j = 0; j < cDeimos.size(); j++)
106         super[i][j] = sqrt(pow(cDeimos[i].coordX - cDeimos[j].coordX, 2) + pow(cDeimos[i].coordY - cDeimos[j].coordY, 2)) + distPuenteMasLargo;
107 }
108
109 // FOBOS (5º Cuadrante)
110 for (size_t i = cDeimos.size(); i < cDeimos.size() + cFobos.size(); i++)
111 {
112     for (size_t j = cDeimos.size(); j < cDeimos.size() + cFobos.size(); j++)
113         super[i][j] = sqrt(pow(cFobos[i].coordX - cFobos[j].coordX, 2) + pow(cFobos[i].coordY - cFobos[j].coordY, 2)) + distPuenteMasLargo;
114 }
115
116 // EUROPA (9º Cuadrante)
117 for (size_t i = cDeimos.size() + cFobos.size(); i < cDeimos.size() + cFobos.size() + cEuropa.size(); i++)
118 {
119     for (size_t j = cDeimos.size() + cFobos.size(); j < cDeimos.size() + cFobos.size() + cEuropa.size(); j++)
120         super[i][j] = sqrt(pow(cEuropa[i].coordX - cEuropa[j].coordX, 2) + pow(cEuropa[i].coordY - cEuropa[j].coordY, 2)) + distPuenteMasLargo;
121 }
122
123
124 // Reconstruimos el archipiélago entero
125 GrafoP<double> Grecoland = Kruskall(super);
126
127 // Ahora hay que normalizar, restamos el peso del puente más largo para hacer Dijkstra
128 // DEIMOS (1º Cuadrante)
129 for (size_t i = 0; i < cDeimos.size(); i++)
130 {
131     for (size_t j = 0; j < cDeimos.size(); j++)
132         Grecoland[i][j] -= distPuenteMasLargo;
133 }
134
135 // FOBOS (5º Cuadrante)
136 for (size_t i = cDeimos.size(); i < cDeimos.size() + cFobos.size(); i++)
137 {
138     for (size_t j = cDeimos.size(); j < cDeimos.size() + cFobos.size(); j++)
139         Grecoland[i][j] -= distPuenteMasLargo;
140 }
141
142 // EUROPA (9º Cuadrante)
143 for (size_t i = cDeimos.size() + cFobos.size(); i < cDeimos.size() + cFobos.size() + cEuropa.size(); i++)
144 {
145     for (size_t j = cDeimos.size() + cFobos.size(); j < cDeimos.size() + cFobos.size() + cEuropa.size(); j++)
146         Grecoland[i][j] -= distPuenteMasLargo;
147 }

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

```
150 // DLH quiere que tanto el origen como destino sean abstraídos 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 }
```

pierdo espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

