

EDNL - Examen Febrero 2025 RESUE...



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

EDNL – EXAMEN FEBRERO 2025 RESUELTO

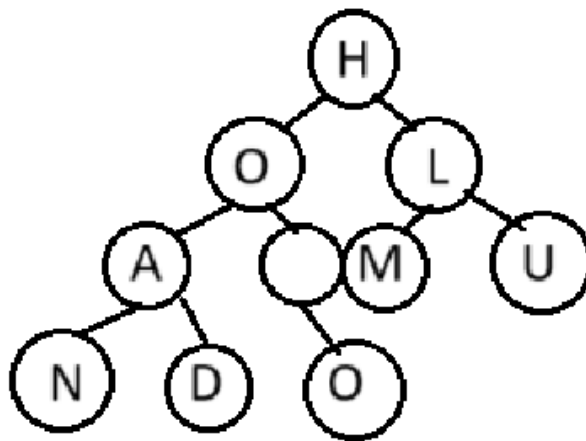
AVISO: Puede que las soluciones aquí propuestas no sean 100% correctas ni las más eficientes. Se trata de proporcionar una solución a modo de guía a lo que se pedían en los ejercicios del examen. No me hago responsable de los posibles suspensos por copiar justo lo siguiente si volviera a caer.

NOTA 1: Como siempre, es absolutamente necesario escribir en el examen todos los TADs conocidos (parte privada) así como los métodos públicos de ellas y los algoritmos de grafos vistos en clases (cabecera).

NOTA 2: En este examen está expresamente prohibido utilizar contenedores de la STL en ambos ejercicios. 🤖

1. Dada una cadena de texto, se pretende realizar un algoritmo criptográfico que cifre la cadena utilizando árboles binarios. El método de cifrado será disponer los caracteres en anchura y posteriormente leerlos en preorden para obtener su cifrado.

Es decir, si tenemos la cadena “HOLA MUNDO”



El resultado debería ser: HOAND OLMU

Recordamos que no se pueden usar vectores, pilas, colas, etc... de la STL.

WUOLAH

```

4 #include "abin.h"
5 #include "abin_E-S.h"
6 #include <iostream>
7
8 using namespace std;
9
10 template <typename T = char>
11 void volcarTextoAlArbol2(Abin<T>& A, const T* texto)
12 {
13     A.insertarRaiz(texto[0]); // Primer elemento
14
15     typename Abin<T>::nodo actual = A.raiz();
16
17     // Inicializamos un arreglo dinámico para simular la cola
18     typename Abin<T>::nodo* cola = new typename Abin<T>::nodo[100]; // Suponemos un tamaño máximo de 100 elementos
19     size_t inicio = 0, fin = 0;
20
21     cola[fin++] = actual;
22
23     size_t i = 1; // Segundo elemento
24     while (texto[i] != '\0')
25     {
26         actual = cola[inicio]; // Obtener el primer elemento de la "cola"
27         inicio++;
28
29         if (texto[i] != '\0')
30         {
31             A.insertarHijoIzqdo(actual, texto[i++]);
32             cola[fin++] = A.hijoIzqdo(actual); // Añadir al final de la "cola"
33         }
34         if (texto[i] != '\0')
35         {
36             A.insertarHijoDrcho(actual, texto[i++]);
37             cola[fin++] = A.hijoDrcho(actual); // Añadir al final de la "cola"
38         }
39     }
40
41     preorden(A.raiz(), A);
42
43     delete[] cola;
44 }
45
46 template <typename T = char>
47 void preordenAbin(typename Abin<T>::nodo n, const Abin<T>& A)
48 {
49     if (n != Abin<T>::NODO_NULO)
50     {
51         cout << A.elemento(n);
52         preordenAbin(A.hijoIzqdo(n), A);
53         preordenAbin(A.hijoDrcho(n), A);
54     }
55 }

```

2. El archipiélago de Grecoland (Zuelandia) está formado únicamente por dos islas, Fobos y Deimos, que tienen N_1 y N_2 ciudades, respectivamente, de las cuales C_1 y C_2 ciudades son costeras (obviamente $C_1 \leq N_1$ y $C_2 \leq N_2$). Se desea construir un puente que una ambas islas. Nuestro problema es elegir el puente a construir entre todos los posibles, sabiendo que el coste de construcción del puente se considera irrelevante. Por tanto, escogeremos aquel **puente que minimice el coste global de viajar entre todas las ciudades de las dos islas**, teniendo en cuenta las siguientes premisas:

1. Se asume que el coste viajar entre las dos ciudades que una el puente **coincide con la distancia euclídea de ambas**.
2. Para poder plantearse las mejoras en el transporte que implica la construcción de un puente frente a cualquier otro, se asume que se realizarán exactamente el mismo número de viajes entre cualesquiera ciudades del archipiélago. Por ejemplo, se considerará que el número de viajes entre la ciudad P de Fobos y la Q de Deimos será el mismo que entre las ciudades R y S de la misma isla. Dicho de otra forma, todos los posibles trayectos a realizar dentro del archipiélago son igual de importantes.

Dadas:

- **Matriz de adyacencia de Fobos (carreteras).**
- **Matriz de adyacencia de Deimos (carreteras).**
- **Coordenadas cartesianas de las ciudades de Fobos.**
- **Coordenadas cartesianas de las ciudades de Deimos.**
- **Relación de las ciudades costeras de Fobos.**
- **Relación de las ciudades costeras de Deimos.**

Implementa un subprograma que calcule las dos ciudades que unirá el puente.

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 oohh
esto con 1 coin me
lo quito yo...

WUOLAH

```
1 // Es el EJ 12 de P7 pero sin usar la STL y con coste de viajar por el puente
2 // Nos dan:
3 // Matriz de adyacencia de las ciudades de Fobos y Deimos (existencia de carreteras)
4 // Coordenadas cartesianas de las ciudades de Fobos y Deimos
5 // Relación de las ciudades costeras de Fobos y Deimos
6 // Implementa un subprograma que calcule las dos ciudades que unirá el puente
7
8 #include "../Material para las prácticas de grafos/alg_grafoPMC.h"
9 #include "../Material para las prácticas de grafos/alg_grafo_E-5.h"
10 #include "../Material para las prácticas de grafos/grafoPMC.h"
11 #include <algorithm>
12 #include <iostream>
13 #include <cmath>
14
15 using namespace std;
16
17 typedef struct{
18     double coordX, coordY;
19 }ciudad;
20
21 template <typename tCoste>
22 typedef struct{
23     typename GrafoP<tCoste>::vertice ciudad1, ciudad2;
24 }solucion_puente;
25
26 template <typename tCoste>
27 solucion_puente grecolandNoSTL(const GrafoP<tCoste>& Fobos, const GrafoP<tCoste>& Deimos,
28     const ciudad* ciudadesFobos, const ciudad* ciudadesDeimos,
29     const ciudad* costerasFobos, const ciudad* costerasDeimos)
30 {
31     // Primero calculamos los costes mínimos entre todas las ciudades usando Floyd
32     typedef typename GrafoP<tCoste>::vertice vert;
33     GrafoP<tCoste> super(Fobos.numVert() + Deimos.numVert()); // Supergrafo para simular el viaje con el puente
34
35     for(size_t i = 0; i < Fobos.numVert(); i++)
36     {
37         for(size_t j = 0; j < Fobos.numVert(); j++)
38         {
39             if(Fobos[i][j])
40                 super[i][j] = sqrt(pow(ciudadesFobos[i].coordX - ciudadesFobos[j].coordX, 2) + pow(ciudadesFobos[i].coordY - ciudadesFobos[j].coordY, 2));
41         }
42     }
43
44     for(size_t i = 0; i < Deimos.numVert(); i++)
45     {
46         for(size_t j = 0; j < Deimos.numVert(); j++)
47         {
48             if(Deimos[i][j])
49                 super[i + Fobos.numVert()][j + Fobos.numVert()] = sqrt(pow(ciudadesDeimos[i].coordX - ciudadesDeimos[j].coordX, 2) + pow(ciudadesDeimos[i].coordY - ciudadesDeimos[j].coordY, 2));
50         }
51     }
52
53     size_t puenteMinimo = GrafoP<tCoste>::INFINITO;
54     size_t puenteAct = sqrt(pow(costerasFobos[0].coordX - costerasDeimos[0].coordX, 2) + pow(costerasFobos[0].coordY - costerasDeimos[0].coordY, 2));
55
56     matriz<vert> mVert(super.numVert());
57
58     solucion_puente<tCoste> solu;
59
60     // Comparamos los puentes entre las ciudades costeras de Fobos y Deimos
61     for(size_t i = 0; i < costerasFobos.size(); i++)
62     {
63         for(size_t j = 0; j < costerasDeimos.size(); j++)
64         {
65             puenteAct = sqrt(pow(costerasFobos[i].coordX - costerasDeimos[j].coordX, 2) + pow(costerasFobos[i].coordY - costerasDeimos[j].coordY, 2));
66             if(puenteAct < puenteMinimo)
67             {
68                 puenteMinimo = puenteAct;
69                 super[i][j] = puenteMinimo; // Contamos el puente para hacer Floyd (SU COSTE NO ES 0)
70
71                 matriz<tCoste> mCostes = Floyd(super, mVert); // Simulamos el viaje
72
73                 // Si minimiza el coste global de viajar con ese puente, guardamos las ciudades costeras
74                 if(mCostes[i][j] < puenteMinimo)
75                 {
76                     solu.ciudad1 = i; // Ciudad de Fobos
77                     solu.ciudad2 = j; // Ciudad de Deimos
78                 }
79
80                 super[i][j] = GrafoP<tCoste>::INFINITO; // Por tanto, luego debemos eliminarlo para no poner todos los puentes por error
81             }
82         }
83     }
84
85     return solu;
86 }
```