

Grafos-Septiembre2022-RESUELTO



Apuntesgii



Estructuras de Datos no Lineales



2º Grado en Ingeniería Informática



Escuela Superior de Ingeniería Universidad de Cádiz



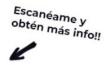
Invita a otros estudiantes, crea contenido y gana los premios que te alegrarán el verano













ApuntesGII

Examen Grafos 01/09/22

Se dispone de un laberinto de NxNxN casillas del que se conocen las casillas de entrada y salida del mismo.

Si te encuentras en una casilla sólo puedes moverte (en el mejor caso) en las siguientes seis direcciones (arriba, abajo, derecha, izquierda, adentro, afuera).

Por otra parte, entre algunas de las casillas hay una piedra que impide moverse hacia ella. Implementa un subprograma que dados:

- N (dimensión del laberinto),
- la lista de casillas que poseen una piedra,
- la casilla de entrada, y
- la casilla de salida,

calcule el coste del camino más corto para ir de la entrada a la salida y su longitud.

*Nota: definir tipos de datos, prototipos de las operaciones de los TADs y de los algoritmos de grafos.





Solución:

```
#include "alg_grafo_E-5.h"
#include "alg_grafoPMC.h"
#include "grafoPMC.h"
#include "matriz.h"
using namespace std;
vertice CasillaToNodo(casilla c,size_t N)
casilla NodoToCasilla(vertice v,size_t N)
       caux.x = \mathbf{v} \times \mathbf{N}; // ejemplos: casilla 3 (x = 0), casilla 10 (x = 1)... caux.y = (\mathbf{v} / \mathbf{N}) \times \mathbf{N}; // ejemplos: casilla 5 (y = 1), casilla 25(x = 2)... caux.z = \mathbf{v} / (\mathbf{N}^*\mathbf{N});
bool abyacentes (casilla c1, casilla c2)
              return (c1.x == c2.x && c1.y == c2.y && abs(c1.z - c2.z) == 1);
```

2 -ApuntesGII



```
for(vertice j = 0; j < G.numVert(); j++)</pre>
//Bloqueamos aquellos caminos en los que la casilla sea una piedra for(vertice i = 0; i < G.numVert(); i++) // podriamos usar int o unsigned tambien, pero uso vertice para hacer referencia a que me muevo de un nodo a otro :)
            G[i][CasillaToNodo(piedras[j],N)] = GrafoP<tCoste>::INFINITO; //NO podemos ir desde cualquier casilla hacia esa casilla
G[CasillaToNodo(piedras[j],N)][i] = GrafoP<tCoste>::INFINITO; //no necesario ya que incomunicamos la casilla que tiene la piedra, udado para tener un Grafo mas realista
vector<vertice> P(G.numVert());
vector<vertice> P(G.numVert());
vector<vertice> D = Dijkstra(G,CasillaToNodo(entrada,N),P); //En D tenemos los costes de ir desde entrada a cualquier nodo
return D[CasillaToNodo(salida,N)]; //devolvemos el coste de ir desde entrada hacia casilla
piedras[0].x = 0; piedras[0].y= 0; piedras[0].z= 0; // casilla 0
piedras[1].x = 2; piedras[1].y= 0; piedras[1].z= 1; // casilla 11
piedras[2].x = 2; piedras[2].y= 1; piedras[2].z= 1; // casilla 14
// FIN DEFINICION DE PIEDRAS
entrada.x = 2; entrada.y = 0; entrada.z = 0; salida.x = 2; salida.y = 0; salida.z = 2;
 tCoste resultado = laberinto(N,piedras,entrada,salida);
```

3 -ApuntesGII

