EDNL - Examen Septiembre 2024 RE...

FranVi_10range

Estructuras de Datos no Lineales

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

FICHA, ALINEA, COMPITE
Y GANA
DESCÁRGATE BIWENGER

EDNL – EXAMEN SEPTIEMBRE 2024 RESUELTO

AVISO: puede que las soluciones aquí propuestas no sean 100% correctas ni las más eficientes. Se trata de proporcionar una solución como guía a lo que se pedía en el examen. No me hago responsable de los posibles suspensos por copiar justo lo siguiente en caso de que cayera de nuevo.

NOTA: como siempre, es absolutamente necesario escribir en el examen todos los TADs conocidos usados (parte privada), así como los prototipos de las operaciones y otros algoritmos conocidos usados.

1. Dado un árbol general donde sus nodos pueden estar vivos o muertos, implementa una función en C++ que, dado un nodo n de éste con una riqueza acumulada, simule un reparto entre todos sus herederos.

   Son herederos de un nodo n sus hijos vivos y sus hijos muertos con descendientes vivos. Para hacer el reparto se divide en cantidades exactamente iguales entre todos sus herederos y, en cuyo caso no sea exacto, el resto se lo quedará el erario público.

   Un nodo seguirá heredando hasta el final, si sigue cumpliendo la condición, o hasta que no quede más riqueza que repartir.

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```cpp
/*
Dado un árbol general donde sus nodos pueden estar vivos o muertos, implementa una función en C++ que,
dado un nodo n de éste con una riqueza acumulada, simule un reparto entre todos sus herederos.

Son herederos de un nodo n sus hijos vivos y sus hijos muertos con descendientes vivos.
Para hacer el reparto se divide en cantidades exactamente iguales entre todos sus herederos y,
en cuyo caso no sea exacto, el resto se lo quedará el erario público.

Un nodo seguirá heredando hasta el final, si sigue cumpliendo la condición, o hasta que no quede más riqueza que repartir.
*/

#include <iostream>
#include "agen.h"
#include "agen_E-S.h"

// Función auxiliar para verificar si un nodo tiene descendientes vivos
template <typename T = double>
bool tieneDescendientesVivos(typename Agen<T>::nodo n, const Agen<T>& A)
{
    typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
    while (hijo != Agen<T>::NODO_NULO)
    {
        if (A.elemento(hijo) > 0 || tieneDescendientesVivos(hijo, A)) // Si el valor es mayor a 0, el nodo está vivo
            return true;
        hijo = A.hermDrcho(hijo);
    }
    return false;
}

// Función para contar los herederos de un nodo n (NO SE REPARTEN ENTRE TODOS LOS HIJOS)
template <typename T = double>
int contarHerederos(typename Agen<T>::nodo n, const Agen<T>& A)
{
    int numHerederos = 0;

    typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
    while (hijo != Agen<T>::NODO_NULO)
    {
        if (A.elemento(hijo) > 0 || tieneDescendientesVivos(hijo, A))
            numHerederos++;
        hijo = A.hermDrcho(hijo);
    }

    return numHerederos;
}

template <typename T = double>
void repartirRiqueza(typename Agen<T>::nodo n, Agen<T>& A)
{
    if (n != Agen<T>::NODO_NULO && A.elemento(n) > 0)
    {
        // Contar herederos del nodo n
        int numHerederos = contarHerederos(n, A);

        if (numHerederos > 0)
        {
            // Calcular la cantidad que corresponde a cada heredero
            double cantidadPorHeredero = A.elemento(n) / numHerederos; // Si no es exacto, el resto se ignora

            // Repartir la riqueza entre los herederos
            typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
            while (hijo != Agen<T>::NODO_NULO)
            {
                if (A.elemento(hijo) > 0)
                {
                    // El hijo está vivo, le damos su parte
                    A.elemento(hijo) = cantidadPorHeredero;
                    repartirRiqueza(hijo, A);
                    // A.elemento(n) = 0; // El nodo muere al repartir
                }
                else if (tieneDescendientesVivos(hijo, A))
                {
                    // El hijo está muerto pero tiene descendientes vivos, le damos su parte para que se reparta
                    A.elemento(hijo) = cantidadPorHeredero;
                    repartirRiqueza(hijo, A);
                    // A.elemento(n) = 0; // El nodo muere al repartir
                }
                hijo = A.hermDrcho(hijo);
            }
        }
    }
}

// Sólo para comprobar, el que te pide NO debe procesar todo el árbol, sino un nodo n cualquiera
template <typename T = double>
void repartirRiqueza(Agen<T>& A)
{
    return repartirRiqueza(A.raiz(), A);
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Revoluciona tu forma de estudiar con Gemini, tu asistente de IA de Google.

2. Dado un tablero de ajedrez compuesto por 8 x 8 escaques o casillas y una casilla de entrada, implementa una función que retorne el mínimo número de movimientos de caballo de ajedrez hacia el escaque o casilla de salida.
SOLUCIÓN 1

```cpp
// Dado un tablero de ajedrez compuesto por 8 x 8 escaques o casillas y una casilla de entrada,
// implementa una función que retorne el mínimo número de movimientos de caballo de ajedrez hacia el escaque o casilla de salida.

#include "../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 <vector>
#include <tuple>

using namespace std;

typedef typename GrafoP<tCoste>::vertice nodo;

typedef struct{
    size_t fila, columna;
}Casilla;

Casilla nodo_to_casilla(nodo n)
{
    Casilla c;
    c.fila = n / 8;
    c.columna = n % 8;

    return c;
}

nodo casilla_to_nodo(Casilla c)
{
    return c.fila * 8 + c.columna;
}

// Movimientos posibles de un caballo de ajedrez
const vector<pair<int, int>> movimientos = {
    {-2, 1}, {-1, 2}, {1, 2}, {2, 1}, {2, -1}, {1, -2}, {-1, -2}, {-2, -1}
};


// i == fila, j == columna
template <typename tCoste>
void movs_caballo(nodo i, GrafoP<tCoste>& tablero)
{
    Casilla c = nodo_to_casilla(i);
    for (auto [df, dc] : movimientos) // Movimientos del caballo de ajedrez (vector de parejas)
    {
        int nf = c.fila + df;
        int nc = c.columna + dc;

        if (nf >= 0 && nf < 8 && nc >= 0 && nc < 8)
        {
            nodo j = casilla_to_nodo({nf, nc}, M);
            tablero[i][j] = tablero[j][i] = 1;
        }
    }
}

template <typename tCoste>
size_t sol_ajedrez(Casilla origen, Casilla destino)
{
    GrafoP<tCoste> tablero(8 * 8);

    for (size_t i = 0; i < tablero.numVert(); i++)
        movs_caballo(i, tablero); // Puede que de INFINITO al no poder salir del tablero

    vector<nodo> uVert(tablero.numVert());
    vector<tCoste> uCostes = Dijkstra(tablero, casilla_to_nodo(origen), uVert);

    if (uCostes[casilla_to_nodo(destino)] == GrafoP<tCoste>::INFINITO)
        return GrafoP<tCoste>::INFINITO;
    else
        return uCostes[casilla_to_nodo(destino)];
}
```

Lleva tu estudio al siguiente nivel con Gemini, tu asistente de IA de Google.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

SOLUCIÓN 2

```cpp
// Dado un tablero de ajedrez compuesto por 8 x 8 escaques o casillas y una casilla de entrada,
// implementa una función que retorne el mínimo número de movimientos de caballo de ajedrez hacia el escaque o casilla de salida.

#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 <vector>

using namespace std;

typedef typename GrafoP<tCoste>::vertice nodo;

typedef struct{
    size_t fila, columna;
}Casilla;

bool esMovCaballo(Casilla c1, Casilla c2)
{
    // Movimiento en 'L' sobre el tablero de ajedrez en cualquiera de sus sentidos
    if (abs(c1.fila - c2.fila) == 1 && abs(c1.columna - c2.columna) == 2)
        return true;
    else if ((abs(c1.fila - c2.fila) == 2 && abs(c1.columna - c2.columna) == 1))
        return true;
    else
        return false;
}

Casilla nodo_to_casilla(size_t n)
{
    Casilla c;
    c.fila = n / 8;
    c.columna = n % 8;

    return c;
}

nodo casilla_to_nodo(Casilla c)
{
    return c.fila * 8 + c.columna;
}

template <typename tCoste>
size_t sol_ajedrez(Casilla origen, Casilla destino)
{
    GrafoP<tCoste> tablero(8 * 8);

    // Recorremos el grafo comprobando si i y j son casillas adyacentes
    // Si lo son colocamos un 1, si no, INFINITO
    for (size_t i = 0; i < tablero.numVert(); i++)
    {
        for (size_t j = 0; j < tablero.numVert(); j++)
        {
            if (i == j)
                tablero[i][j] = 0; // No cuesta nada
            else if (esMovCaballo(nodo_to_casilla(i), nodo_to_casilla(j)))
                tablero[i][j] = tablero[j][i] = 1;
            else
                tablero[i][j] = tablero[j][i] = GrafoP<tCoste>::INFINITO;
        }
    }

    // Ahora ya podemos aplicar Dijkstra
    vector<vert> uVert(tablero.numVert());
    vector<tCoste> uCostes = Dijkstra(tablero, casilla_to_nodo(origen), uVert);

    if (uCostes[casilla_to_nodo(destino)] == GrafoP<tCoste>::INFINITO)
        return GrafoP<tCoste>::INFINITO;
    else
        return uCostes[casilla_to_nodo(destino)];
}
```

descarga randstad app y empieza hoy.
descargar app en app store
descargar app en google play
randstad

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.