

Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software (grupo A)

Examen Final, 3 de Junio de 2014.

1. (3 puntos) Alonso Rodríguez tiene que hacer la compra de la semana. Ha hecho una lista de n productos que quiere comprar. En su barrio hay m supermercados en cada uno de los cuales se dispone de todos esos productos. Pero como es un comprador compulsivo no quiere comprar más de tres productos en cada uno de los supermercados ya que así pasa más tiempo comprando (se puede suponer que $n \leq 3m$). Dispone de una lista de precios de los productos en cada uno de los supermercados. Se pide diseñar un algoritmo de vuelta atrás que permita a Alonso decidir cómo hacer la compra de forma que compre todo lo que necesita y que el coste total sea mínimo.
2. (3 puntos) Dado un árbol binario, en cuya raíz se encuentra situado un tesoro y cuyos nodos internos pueden contener un dragón o no contener nada, se pide diseñar un algoritmo que nos indique la hoja del árbol cuyo camino hasta la raíz tenga el menor número de dragones. En caso de que existan varios caminos con el mismo número de dragones, el algoritmo devolverá el que se encuentre más a la izquierda de todos ellos.

Para ello implementar una función que reciba un árbol binario cuyos nodos almacenan cadenas de caracteres:

- La raíz tiene la cadena *Tesoro*
- Los nodos internos tienen la cadena *Dragón* para indicar que en el nodo hay un dragón o la cadena *Vía libre* para indicar que no hay dragón.
- En cada hoja se almacena un identificador que no puede estar repetido.

y devuelva el identificador de la hoja del camino seleccionado. El árbol tiene como mínimo un nodo raíz y un nodo hoja diferente de la raíz. La operación no se implementa como parte de ningún TAD.

El coste de la operación implementada debe ser $\mathcal{O}(n)$.

Por ejemplo, dado el siguiente árbol el algoritmo devolverá la cadena de caracteres *Puerta falsa*.



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

3. (4 puntos) Eres un prestigioso diseñador de videojuegos, y estás trabajando ahora en uno llamado *EspacioMatic*, en el que habrá naves espaciales con distintos módulos (motores, cabinas, escudos, láseres, etcétera). Necesitarás implementar, como poco, las siguientes operaciones:

- *EspacioMatic*: inicializa el sistema de juego.
- *nuevaNave*: añade una nueva nave espacial (vacía) al sistema, con el identificador numérico proporcionado. Si se intenta usar un identificador ya existente produce error. No devuelve nada.
- *equipaNave*: dado el identificador de una nave, un nombre de módulo (una cadena como “motor”, “cabina”, “láser”, etcétera) y un nivel de funcionalidad (un entero ≥ 1), añade el módulo correspondiente a esa nave con el nivel indicado. Si esa nave ya tenía ese módulo, se suma el nuevo nivel al anterior (esto permite reparar módulos de naves). No devuelve nada.
- *estropeaNave*: dado el identificador de una nave, y un nombre de módulo, resta 1 al nivel de ese módulo en esa nave (asumiendo que tuviese un nivel > 0). Devuelve `true` si el módulo existía y tenía un nivel positivo antes de hacer la resta, ó `false` si ha sido imposible restar nivel ya que el módulo no existía o estaba ya a 0.
- *navesDefectuosas*: devuelve una lista de identificadores de naves que tienen uno o más módulos completamente estropeados (con un nivel de 0).
- *modulosNave*: dado el identificador de una nave, devuelve una lista con los nombres de los módulos que tiene equipados (tengan el nivel que tengan), ordenada alfabéticamente.

Desarrolla en C++ una implementación de la clase *EspacioMatic* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones. En cada operación, indica también, de forma razonada, su complejidad.

The logo for Cartagena99 features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the 'Cartagena' part. The text is set against a light blue background with a subtle gradient and a soft shadow effect.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

JUNIO 2014

① Supermercado $(n \leq 3m)$

- n productos
- m supermercados
- no más de 3 productos/super
- Precios
- Coste total mínimo

Planteamiento

La solución es un vector de n posiciones (una por producto) que guarde en qué super lo compramos.

- Requisitos explícitos
 $(x_1, \dots, x_n) \quad 0 \leq x_i < m$

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

- Requisitos implícitos
 - El mismo súper no está repetido más de 3 veces.
 - Coste total mínimo.

• Parámetros

- (int) - N (nº de productos) (Global)
- (int) - M (nº de súper) (Global)
 - (float) precios $[N][M]$ (Global)
 $\begin{matrix} \text{file} & \text{columna} \end{matrix}$
- int sol []
- int k etapa
- int solOpt []
- float costeAct, costeMin.
- int marcas [M]

marcas se inicializa a 0 y cuando compramos en el súper i hacemos $marcas[i]++$

```
void compra (int sol [ ], int k, int solOpt [ ],
            float & costeAct, float & costeMin, int marcas [ ]) {
  for (int super = 0 ; super < M; super++) {
    sol [k] = super;
    costeAct = costeAct + precio [k] [super];
    marcas [super]++;
    if (marcas [super] <= 3) // es válida
      if (k == N-1)
```

Cartagena99

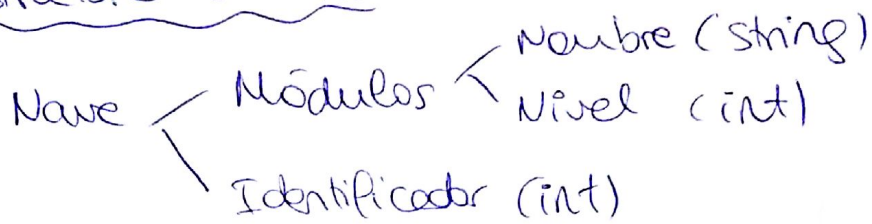
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP: 689 45 44 70

de...
 del if } marcas [super] --;

ESPACIOMÁTIC

(ES. 3- Ex. Junio 2014)

Estructura de datosHashMap con clave

Identificador de la nave y valor un TreeMap cuyas claves sean nombres de módulos y cuyos valores sean los niveles.

```
class Espaciomatic {
```

```
private:
```

```
HashMap <int, TreeMap <string, int >>
```

```
-naves;
```

```
HashMap <int, List <string>>
```

```
-detectores; treeMap <string, int>
```

```
public:
```

```
Espaciomatic ();
```

```
void naveName (const int &id);
```

```
void equipaNave (const int &id, const string &mod,
                const int &nivel);
```

```
void estropeaNave (const int &id, const string &mod);
```

```
... (int) ...
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

Cartagena99

```

void EspacioMatic :: EspacioMatic ( ) { };
void EspacioMatic :: newNave (const int & id) {
    if (_naves.contains(id)) {
        throw naveExistente();
    }
    _naves.insert(id, new TreeMap < string, int > ());
}

void EspacioMatic :: equipaNave (const int & id, -nave
                                const string & mod, const int & nivel) {

    if (_naves.contains(id)) { // si está la nave
        TreeMap < string, int > modulos = _naves.at(id)
        * → if (!modulos.contains(mod)) {
            // Si no está lo insertamos
            modulos.insert(mod, 0) // Inicializamos a 0
        }
        _modulos.insert(mod, modulos.at(mod) + nivel); // insert. modulos
        _naves.insert(id, modulos); // Actualizamos nave
    }

    * if (modulos.at(mod) == 0) // Este defectuoso y nivel 0,
        if (_defectuosos.at(id).contains(mod)) { presuponemos que este
            TreeMap < string, int >
            _modDef = _defectuosos.at(id);
            _modDef.erase(mod); // lo quitamos
        }
}

```

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

quitamos de defectuosos

```
bool EspacioModic :: EstropeaName (const int &id,
const string & mod);
```

```
if (baves. contains (id) ) cte // No existe la nave
    throw NoExisteName ();
```

```
if (b-aves. at (id). contains (mod) ) cte
    return false; // La nave no tiene
                    // el mod
```

```
Treemap <string, int> modulos = _naves. at (id); cte
```

```
if (modulos. at (mod) == 0) // Modulo defect
    return false;
```

```
int nivel = modulos. at (mod); los
modulos. insert (mod, nivel - 1); log // Decrementamos
                                        // el nivel
```

```
_naves. insert (id, modulos); // Actualizamos cte las
                                        // naves
```

```
if ( nivel - 1 == 0 ) // Si el modulo en que
                    // defectoso
```

```
if ( _defectuosos. contains (id) ) cte
```

```
    modulos = _defectuosos. at (id); cte
```

```
    modulos. insert (mod, 0); los
```

```
    _defectuosos. insert (id, modulos); cte
```

```
else
```

```
Treemap <string, int> mod2; // Creamos en uno nuevo
                            // (modulo)
```

```
mod2. insert (mod, 0); // Insertamos cte
```

```
_defectuosos. insert (id, mod2); // Actualizamos
```

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

```
Lista <int> Espaciomatic :: new defectuosas() {
```

```
Lista <int> l;
```

```
HashMap <int, TreeMap <string, int>>:: const_iterator it =  
_defectuosas.cbegin();
```

```
while (it != _defectuosas.cend()) {  
    l.push_back (it->key());  
    it++;  
}  
return l;
```

vuelta (n = defect)

// Iterador al principio

// lo vemos al final de la lista

Complejidad: $O(n)$

```
Lista <string> Espaciomatic :: modulosNaves (const int &id) {
```

```
Lista <string> l;
```

```
TreeMap <string, int>:: const_iterator it;
```

```
if (!naves.contains (id))  
    throw NoExisteNaves;
```

```
it = _naves.at (id).cbegin();
```

```
while (it != _naves.at (id).cend()) {
```

```
    l.push_back (it->key);
```

```
    it++;
```

$O(n)$

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

25/5/15

MULTIUMATIC

Planteamiento de las estruct. de datos:

Para los tramos:

```

template < class T, class C >

```

Tramos (pointing to T) *Cameras* (pointing to C)

```

struct _infoTramo {

```

```

    C CamIni;

```

```

    C CamFin;

```

```

    int seg;

```

Matriculas (pointing to seg)

```

    Lista string - cochesMultidos;

```

```

    HashMap < T, _infoTramo > _tramos;

```

```

struct _infoCoche {

```

```

    T tramo;

```

```

    int nMultas;

```

```

    time_t instanteEntrada;

```

```

    bool enTramo;

```

```

HashMap < matricula, _infoCoche > - coches;

```

```

HashMap < C, T > - cameras // Asocia cameras a tramos

```

```

HashMap < T, infoTramo > _tramos;

```

```

class Multiumatic {

```

```

private:

```

```

    struct infoTramo;

```

```

    struct infoCoche;

```

```

public:

```

```

    insertTramo

```

```

    fotoEntrada

```

```

    fotoSalida

```

```

    multasPorMatricula

```

```

    multasPorTramos

```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP: 689 45 44 70

~~void insertaTramo~~

```
void Multitallatic <T, C> insertaTramo (const T & t,  
    const C & cini,  
    const C & cfin,  
    const int & minSec) {
```

```
    if (-tramos.contains(t)) {  
        throw LoContiene();  
    }  
    else {  
        if (-cameras.contains(cini) || -cameras.contains(cfin)) {  
            throw Error ErrorCamera();  
        }  
        infoTramo mitramo;  
        mitramo.CameraIni = cini;  
        mitramo.CameraFin = cfin;  
        mitramo.Duracion = minSec;  
        -tramos.insert(t, mitramo);  
        -cameras.insert(cini, t);  
        -cameras.insert(cfin, t);  
    }  
}
```

```
template < class T, class C >  
void Multitallatic < T, C > :: fotoEntrada (const C & cam,  
    const string & matricula,  
    time_t time) {  
    if (! -cameras.contains(cam))  
        throw Excepcion();  
    infoCoche aux;  
    aux.tramo = -cameras.at(cam);  
}
```

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

```
template <class T, class C>
```

```
void MultaMatic :: Fotosalida (const C& cam,  
                             const string & matricula,  
                             time_t time) {
```

```
if (!_cameras.contains(cam)) {  
    throw NoEstaC();  
}
```

```
// if (!_coches.contains(matricula)) {
```

```
//     throw NoEstaCoches();
```

// No le podemos poner multa ya que el coche no está

```
// {  
if (_coches.contains(matricula)) {
```

```
    InfoCoches aux = _coches.at(matricula);
```

```
    InfoTramo auxT = _tramos.at(_cameras.at(cam));
```

```
if (aux.entramo == true && _cameras.at(cam) == aux.tramo)
```

```
if (time - aux.tempoentrad < auxT.tempo/min)
```

```
    aux.nMultas++;
```

```
    auxT.multas.pushback(matricula);
```

```
    aux.entramo = false;
```

```
    _coches.insert(matricula, aux);
```

```
    _tramos.insert(_tramos.at(_cameras.at(cam)), auxT);
```

o/m

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70

```

template <class T, class C >
int Multalotie <T, C> :: MultasPorMatricula (const string & matricula) const
{
    if (! _coches.contains(matricula))
        throw CocheNoExiste();

    return _coches.at(matricula).multas;
}

```

```

template <class T, class C >
Lista <string> Multalotie <T, C> :: MultasPorTramo
(const T & t) const
{
    if (! _tramos.contains(t))
        throw TramoNoExiste();

    return _tramos.at(t).multas;
}

```

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP: 689 45 44 70