

# Estructuras de Datos no Lineales

## Práctica 1

### Problemas de árboles binarios I

#### TRABAJO PREVIO

1. **Antes de asistir a la primera sesión de prácticas es obligatorio tener implementado y probado** el TAD *árbol binario* con las tres representaciones estudiadas, vectorial, vector de posiciones relativas y enlazada.
2. **También es obligatorio antes de asistir a la sesión de prácticas:**
  - 2.1. Imprimir copia de este enunciado.
  - 2.2. Lectura profunda del mismo.
  - 2.3. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
  - 2.4. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

#### PASOS A SEGUIR

1. Escribir módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
2. Para cada uno de los problemas escribir un programa de prueba, independiente de la representación del TAD elegida, donde se realicen las llamadas a los subprogramas del paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba.

#### ENTRADA Y SALIDA DE ÁRBOLES BINARIOS

Se proporciona la cabecera `abin_E-S.h` que incluye cuatro funciones genéricas para la lectura y escritura de árboles binarios a través de flujos de entrada y salida:

```
template <typename T> void rellenarAbin (Abin<T>& A, const T& fin)
```

Pre: *A* está vacío.

Post: Rellena el árbol *A* con la estructura y elementos leídos en preorden de la entrada estándar, usando *fin* como elemento especial para introducir nodos nulos.

```
template <typename T> void rellenarAbin (istream& is, Abin<T>& A)
```

Pre: *A* está vacío.

Post: Extrae los nodos de *A* del flujo de entrada *is*, que contendrá el elemento especial que denota un nodo nulo seguido de los elementos en preorden, incluyendo los correspondientes a nodos nulos.

*template <typename T> void imprimirAbin (const Abin<T>& A)*

Post: Muestra los nodos de *A* en la salida estándar.

*template <typename T>*

*void imprimirAbin (ostream& os, const Abin<T>& A, const T& fin)*

Post: Inserta en el flujo de salida *os* los nodos de *A* en preorden, precedidos del elemento especial usado para denotar un nodo nulo.

### Ejemplo:

```
#include <iostream>
#include <fstream>
#include "abin.h"
#include "abin_E-S.h"

using namespace std;

typedef char tElto;
const tElto fin = '#';    // Fin de lectura.

int main ()
{
    Abin<tElto> A, B;

    cout << "*** Lectura del árbol binario A ***\n";
    rellenarAbin(A, fin);    // Desde std::cin

    ofstream fs("abin.dat");    // Abrir fichero de salida.
    imprimirAbin(fs, A, fin);    // En fichero.
    fs.close();
    cout << "\n*** Árbol A guardado en fichero abin.dat ***\n";

    cout << "\n*** Lectura de árbol binario B de abin.dat ***\n";
    ifstream fe("abin.dat");    // Abrir fichero de entrada.
    rellenarAbin(fe, B);    // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol binario B ***\n";
    imprimirAbin(B);    // En std::cout
}
```

### Salida del programa:

```
*** Lectura del árbol binario A ***
Raíz (Fin = #): a
Hijo izqdo. de a (Fin = #): b
Hijo izqdo. de b (Fin = #): c
Hijo izqdo. de c (Fin = #): d
Hijo izqdo. de d (Fin = #): #
Hijo drcho. de d (Fin = #): #
Hijo drcho. de c (Fin = #): #
Hijo drcho. de b (Fin = #): e
Hijo izqdo. de e (Fin = #): #
Hijo drcho. de e (Fin = #): f
Hijo izqdo. de f (Fin = #): #
```

```
Hijo drcho. de f (Fin = #): #
Hijo drcho. de a (Fin = #): g
Hijo izqdo. de g (Fin = #): h
Hijo izqdo. de h (Fin = #): #
Hijo drcho. de h (Fin = #): #
Hijo drcho. de g (Fin = #): i
Hijo izqdo. de i (Fin = #): j
Hijo izqdo. de j (Fin = #): #
Hijo drcho. de j (Fin = #): #
Hijo drcho. de i (Fin = #): k
Hijo izqdo. de k (Fin = #): l
Hijo izqdo. de l (Fin = #): #
Hijo drcho. de l (Fin = #): #
Hijo drcho. de k (Fin = #): #
```

\*\*\* Árbol A guardado en fichero abin.dat \*\*\*

\*\*\* Lectura de árbol binario B de abin.dat \*\*\*

\*\*\* Mostrar árbol binario B \*\*\*




```
Raíz del árbol: a
Hijo izqdo de a: b
Hijo izqdo de b: c
Hijo izqdo de c: d
Hijo derecho de b: e
Hijo derecho de e: f
Hijo derecho de a: g
Hijo izqdo de g: h
Hijo derecho de g: i
Hijo izqdo de i: j
Hijo derecho de i: k
Hijo izqdo de k: l
```

Fichero abin.dat:

```
#
a b c d # # # e # f # # g h # # i j # # k l # # #
```

Las funciones de `abin_E-S.h` serán muy útiles para introducir y mostrar árboles binarios en los programas de prueba de los ejercicios, especialmente si se usan ficheros para ello, pues se podrá repetir la ejecución de las pruebas sin tener que teclear una y otra vez los mismos datos.

## PROBLEMAS

1. Implementa un subprograma que calcule el número de nodos de un árbol binario. 
2. Implementa un subprograma que calcule la altura de un árbol binario. 
3. Implementa un subprograma que, dados un árbol binario y un nodo del mismo, determine la profundidad de este nodo en dicho árbol. 
4. Añade dos nuevas operaciones al TAD *árbol binario*, una que calcule la profundidad de un nodo y otra que calcule la altura de un nodo en un árbol dado. Implementa esta operación para la representación vectorial (índices del padre, hijo izquierdo e hijo derecho).

5. Repite el ejercicio anterior para la representación enlazada de árboles binarios (punteros al padre, hijo izquierdo e hijo derecho).

6. Implementa un subprograma que determine el nivel de desequilibrio de un árbol binario, definido como el máximo desequilibrio de todos sus nodos. El desequilibrio de un nodo se define como la diferencia entre las alturas de los subárboles del mismo.



7. Implementa un subprograma que determine si un árbol binario es o no *pseudocompleto*.

En este problema entenderemos que un árbol es *pseudocompleto*, si en el penúltimo nivel del mismo cada uno de los nodos tiene dos hijos o ninguno.

