Estructuras de Datos y Algoritmos 1

Práctico 5

Tema: TADs Diccionario, Conjunto, Tabla y Variantes.

OBJETIVOS

- **Familiarizarse con los conceptos relacionados con la especificación de TADs**
- **Discutir los conceptos y herramientas relacionados con la implementación de TADs en C y C++**
- **Familiarizarse con los conceptos de función virtual, función virtual pura, clase abstracta. clase concreta e implementación de TADs en C++**
- **Comprender el papel que cumplen los constructores copia, operador de asignación, operadores de inserción y de extracción de flujo**
- **Comprender como puede elegirse dinámicamente (en tiempo de ejecución) la implementación de un TAD**

0) Resolver los ejercicios pendientes de los módulos 10 y 11 del teórico: TADs Conjuntos, Diccionarios y Tablas.

TAD Diccionario

1) Considere las especificaciones de Diccionario e Iterador del ANEXO:
   Desarrolle las implementaciones de Diccionario:
   - TAD Lista
   - Array con tope
   - ABB

   - Note que la implementación basada en listas lineales se basa en otro TAD cuya implementación, a su vez puede elegirse posteriormente.

   - Discuta bajo que circunstancias son aplicables cada una de ellas

   - Desarrolle la clase diccionario Mutante que según el cardinal del diccionario cambia de implementación entre implementaciones no acotadas de acuerdo a la discusión anterior.

TAD Conjunto (Set)

2) Agregue al TAD Diccionario los operadores + (union), * (intersección) y – (diferencia) para obtener un TAD Conjunto. Note que si estos operadores retornan un Dicionario<T>& y no afectan a this puede escribirse (A+B)*C sin perder generalidad puesto que también es posible escribir A= (A+B)*C.

TAD Tabla

3) Especifique el TAD Tabla

4) Proporcione implementaciones para el TAD Tabla basadas en lista de pares, array de pares con tope, lista encadenada de pares y ABB.

TAD Arbol Binario de Búsqueda (ABB)

5) Considere la especificación del TAD ABB y su correspondiente implementación vista en el teórico (ver anexo al archivo:***11H_tadArboles.ppt***):

a) En vez de considerar en la parte de "valores" del módulo de implementación del TAD ABB un puntero a una estructura Nodo_Arb, podemos considerar un objeto de tipo AB (TAD Arbol Binario, visto en el teórico).

b) De esta manera los métodos Vacio, Raiz (o RaizABB), SubArbIzq, SubArbDer, EsVacio, BorrarAB (o BorrarABB), y los tres recorridos podrían ser definidos en ABBImp a partir de los correspondientes a AB.

c) Los otros métodos de ABBImp se definirían a partir de los de AB. Si se desea que ABBImp acceda a los datos protegidos de AB debería ponerse en AB la cláusula: `friend class ABBImp.`

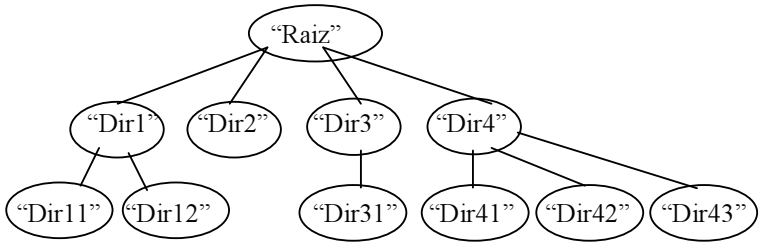
Se propone como ejercicio realizar la implementación alternativa de ABBImp según la observación previa.

Notar que la herencia no es un mecanismo apropiado entre AB y ABBImp, ya que `CreoI` y `CreoD` son métodos públicos de AB y no deberían ser métodos de ABBImp (éstos permiten crear árboles no necesariamente ordenados).

d) Implemente las operaciones del TAD ABB considerando que el mismo debe estar balaceado (AVL), según se vio en el teórico.

TAD Arbol General

6) Defina una estructura de datos en C/C++ capaz de representar árboles generales de strings.
7) Explique como se modela en la estructura por usted propuesta el siguiente árbol general:



8) Defina un procedimiento *print_nivel* en C/C++ que dados un árbol general de strings y un entero positivo *i*, imprima todos los elementos del árbol que se encuentran en el nivel *i*. Recordar que la raíz en un árbol general no vacío se encuentra en el nivel 1. Si el árbol es vacío o *i* es mayor que la cantidad de niveles del árbol, el procedimiento no debe imprimir nada.

NOTA: No se pueden utilizar TADs

9) Considere la siguiente definición de las operaciones del TAD ArbolesGenerales no vacíos y sin elementos repetidos, de un tipo genérico T:

- **ArbolHoja**: Dado un elemento genera un árbol que sólo contiene dicho elemento (como una hoja).
- **Insertar**: Dados un Arbol y dos elementos v y w, inserta a v como el **primer** hijo de w en el árbol (hijo más a la izquierda), siempre que w pertenezca al árbol y v no pertenezca al árbol. En caso contrario, la operación no tiene efecto.
- **EsArbolHoja**: Dado un árbol, retorna true si, y sólo si, el árbol es un árbol hoja (con un solo elemento).
- **Pertenece**: Dados un árbol y un elemento, retorna true si y sólo si el elemento pertenece al árbol.
- **Borrar**: Dados un árbol y un elemento, elimina al elemento del árbol siempre que éste pertenezca al árbol, no sea la raíz del mismo y no tenga ningún hijo. En caso contrario, la operación no tiene efecto.

Se pide:

a) Especifique en C++ el TAD ArbolesGenerales.
b) Implemente el TAD ArbolesGenerales. Considere la representación de árboles generales basada en árboles binarios, como una estructura de datos ("primer hijo" –"siguiente hermano").
No utilice clases ni TADs auxiliares en este ejercicio.
Si necesita usar funciones o procedimientos auxiliares en la implementación del TAD, desarrolle los códigos correspondientes.

ANEXOS

```
// ********************************************************************* //
#ifndef ITERABLE_H
#define ITERABLE_H

template < class T >
class Iterable {
public:
      Iterable(){}
      virtual ~Iterable(){}

      // PREDICADOS //

      // Retorna un elemento del objeto iterable
      // Pre:
      // 1) el objeto sobre el que se iterna no ha sido modificado
      // 2) EsFin() == false
      virtual const T& Elemento() = 0;

      // Avanza un elemento en el objeto iterable
      // Pre:
      // 1) el objeto sobre el que se iterna no ha sido modificado
      virtual void Resto() = 0;

      // Crea un iterador dejando la posicion el ppio del mismo
      virtual void Principio() = 0;

      // Retorna true si es el final del iterador
      // Pre:
      // 1) el objeto sobre el que se iterna no ha sido modificado
      virtual bool EsFin() = 0;
};

#endif

// ********************************************************************* //
#ifndef DICCIONARIOIT_H
#define DICCIONARIOIT_H

#include "Iterable.h"
#include <iostream>
using namespace std;

template < class T >
class DiccionarioIT: public Iterable<T> {
            friend ostream& operator<< <>(ostream& out, DiccionarioIT<T> &d);

public:
```

```
        // CONSTRUCTORAS //

        DiccionarioIT(){}
        DiccionarioIT(const DiccionarioIT<T> &d);

        // Deja al diccionario vacio, sin elementos
        virtual void Vacio() = 0;

        // Agrega e si el mismo no estaba en el diccionario, en caso
        // contrario no hace nada.-
        virtual void Agregar(const T&e) = 0;

        // DESTRUCTORAS //

        virtual ~DiccionarioIT(){}

        // Borra "e" si el mismo estaba en el diccionario, en caso
        // contrario no hace nada.-
        virtual void Eliminar(const T &e) = 0;

        // PREDICADOS //

        // Retorna true si el diccionario es vacío.-
        virtual bool EsVacio() = 0;

        // Retorna true si "e" se encuentra en el diccionario, en caso
        // contrario retorna false.-
        virtual bool EsMiembro(const T &e) = 0;
};

#endif
// ********************************************************************** //
```