Estructuras de Datos y Algoritmos 1

Solución - Práctico 5

Tema: TADs Diccionario, Conjunto y Tabla. TADs Arboles.

TAD Tabla - Especificación
Archivo: Tabla.h

```
#ifndef TABLA_H
#define TABLA_H

template <class Dominio, class Rango>
class Tabla {
public:
      virtual ~ListaImp(){}

      /* Constructoras */
      virtual void CreoTabla() = 0;
      // retorna la tabla vacía

      virtual void Insertar(const Dominio &d,const Rango &r) = 0;
      /* Sea T la tabla a la cual se le aplica el método.
      Insertar define T(d) = r, independientemente de que T(d) estuviera ya
      definido */

      /* Predicados */
      virtual bool esVacia()=0;
      // retorna true si y sólo si la tabla está vacía.

      virtual bool estaDefinida(const Dominio &d)=0;
      // retorna true si y sólo si la tabla está definida en d.

      /* Selectoras */
      virtual Rango Recuperar (const Dominio &d)=0;
      /* Sea T la tabla a la cual se le aplica el método.
      Recuperar retorna el elemento T(d). Pre-condición: T.isdef(d) es true,
      es decir, T está definida para d */

      virtual bool Eliminar (const Dominio &d)=0;
      /* Eliminar borra la asociación ligada a d, siempre que ésta exista.*/
};

#endif
```

TAD Tabla – Implementación (Listas enlazadas)
Archivo: TablaImp.h

```
#ifndef TABLAIMP_H
#define TABLAIMP_H

#include "Tabla.h"

template <class Dominio, class Rango>
class TablaImp: public Tabla<Dominio, Rango> {
protected:
      struct nodo;
      typedef struct nodo *LISTA;

      struct nodo{
            Dominio dom;
            Rango ran;
            LISTA prox;
      };

      LISTA ppio;
      void BorrarLista();
public:
      TablaImp();
      ~TablaImp();
      void CreoTabla();
      void Insertar(const Dominio &d,const Rango &r);
      bool estaDefinida(const Dominio &d);
      Rango Recuperar (const Dominio &d);
      bool Eliminar (const Dominio &d);

};

#include "TablaImp.cpp" //Solo para evitar error

#endif
```

TAD Tabla – Implementación (Listas enlazadas)
Archivo: TablaImp.cpp

```
#ifndef TABLAIMP_CPP
#define TABLAIMP_CPP

#include "TablaImp.h"

template <class Dominio, class Rango>
TablaImp<Dominio,Rango>::TablaImp(){
      ppio = NULL;
}

template <class Dominio, class Rango>
TablaImp<Dominio,Rango>::~TablaImp(){
      BorrarLista();
}
```

```
template <class Dominio, class Rango>
void TablaImp<Dominio,Rango>::CreoTabla(){
	Borrarlista();
	ppio = NULL;
}

template <class Dominio, class Rango>
void TablaImp<Dominio,Rango>:: Insertar(const Dominio &d,const Rango &r){
	LISTA aux = ppio, nuevo;
	bool esta = false;
	if(!ppio ){
		ppio = new nodo;
		ppio->dom = d;
		ppio->ran = r;
		ppio->prox = NULL;
	}
	else{
		while (aux && !esta){
			if (aux->dom == d){
				aux->ran = r;
				esta = true;
			}
			else
				aux = aux->prox;
		}
		if (!esta){
			nuevo = new nodo;
			nuevo ->dom = d;
			nuevo ->ran = r;
			nuevo ->prox = ppio;
			ppio = nuevo;
		}
	}
}

template <class Dominio, class Rango>
bool TablaImp<Dominio,Rango>::estaDefinida(const Dominio &d){
	LISTA aux = ppio;
	bool esta = false;
	while (aux && !esta){
		if (aux->dom == d)
			esta = true;
		else
			aux = aux->prox;
	}
	return esta;
}

template <class Dominio, class Rango>
Rango TablaImp<Dominio,Rango>::Recuperar (const Dominio &d){
	LISTA aux = ppio;
	bool esta = false;
	while (aux && !esta){
		if (aux->dom == d)
			esta = true;
		else
			aux = aux->prox;
```

```
	}
	return aux->ran;

}


template <class Dominio, class Rango>
bool TablaImp<Dominio,Rango>:: Eliminar (const Dominio &d){
	bool esta = false;
	LISTA aux = ppio, ant;
	if(ppio && ppio->dom == d){
		borrar = ppio;
		ppio = ppio->prox;
		delete borrar;
	}
	else{
		while (aux && !esta){
			if (aux->dom == d)
				esta = true;
			else{
				ant = aux;
				aux = aux->prox;
			}
		}
	}
	if (esta){
		ant->prox = aux->prox;
		delete aux;
	}

	return esta;

}

template <class Dominio, class Rango>
void TablaImp<Dominio,Rango>::BorrarLista(){
	LISTA aux = ppio, borrar;
	while (aux){
		borrar = aux;
		aux = aux->prox;
		delete borrar;
	}
	}
```

TAD Arbol General - Especificación
Archivo: AGeneral.h

```
#ifndef AGENERAL_H
#define AGENERAL_H

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

      /* Constructoras */
      virtual void ArbolHoja(const T &d) = 0;
      // Dado un elemento genera un árbol que sólo contiene dicho
       elemento (como una hoja).

      virtual void Insertar(const T &v, const T &w) = 0;
      /* 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.
      */

      /* Predicados */
      virtual bool esArbolHoja()=0;
      // un árbol, retorna true si, y sólo si, el árbol es un árbol hoja
      (con un solo elemento).

      virtual bool Pertenece(const T &v)=0;
      // Dados un árbol y un elemento, retorna true si y sólo si el elemento
      pertenece al árbol

      /* Selectoras */

      virtual void Borrar (const Dominio &d)=0;
      /* 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 */
};

#endif
```

TAD Arbol General - Implementación
Archivo: AGeneralImp.h

```
#ifndef AGENERALIMP_H
#define AGENERALIMP_H

#include "AGeneral.h"

template <class T>
class AGeneralImp: public AGeneral<T> {
protected:
      struct nodo;
      typedef struct nodo * AGRAL;

      struct nodo{
            T dato;
            AGRAL pHijo;
            AGRAL sHermano;
      };

      AGRAL raiz;
      void BorrarArbol(AGRAL &r);
      void InsertarPriv(AGRAL &r, const T &v, const T &w);
      bool PerteneecePriv(AGRAL r, const T &v);
      void BorrarPriv(AGRAL &r,const T &v);

public:
      AGeneral();
      ~AGeneral();

      void ArbolHoja(const T &d);

      void Insertar(const T &v, const T &w);
      bool esArbolHoja();
      bool Pertenece(const T &v);
      void Borrar (const Dominio &d);
};

#include "AGeneralImp.cpp" //Solo para prevenir errores

#endif
```

TAD Arbol General - Implementación
Archivo: AGeneralImp.cpp

```
#ifndef AGENERALIMP_CPP
#define AGENERALIMP_CPP

#include "AGeneralImp.h"

template <class T>
void AGeneralImp <T>::BorrarArbol(AGRAL &r){

      if(r != NULL){
            if (r->sHermano)
                        BorrarArbol(r->sHermano);
            if (r->pHijo)
                    BorrarArbol(r->pHijo);
            cout<<"BORRO: "<<r->dato<<" - ";
            delete r;
      }
}

template <class T>
void AGeneralImp<T>::InsertarPriv(AGRAL &r,const T &v, const T &w){
      AGRAL nuevo = new nodoAG;
      AGRAL aux = r;
      bool inserte = false;
      //assert(PertenecePriv(r,v));
      //assert(!PertenecePriv(r,w));
      while (aux != NULL && !inserte){
            if (aux->dato == w){
                  nuevo->dato = v;
                  nuevo->pHijo = NULL;
                  nuevo->sHermano = aux->pHijo;
                  aux->pHijo = nuevo;
                  inserte = true;
            }
            else{
                  InsertarPriv(aux->pHijo, v, w);
                  aux = aux->sHermano;
            }
      }
}

template <class T>
bool AGeneralImp<T>::PertenecePriv(AGRAL r, const T &v){
      bool ret = false;
      if(r->dato == v)
            return true;
      else{
            if (r->sHermano)
                        ret = ret || PertenecePriv (r->sHermano, v);
            if (r->pHijo)
                    ret = ret || PertenecePriv (r->pHijo, v);
      }
      return ret;
}
```

```
template <class T>
void AGeneralImp<T>::BorrarPriv(AGRAL &r,const T &v){
	AGRAL borrar, ant, a;

	if(r){
		if(r->dato == v){
			if(r->pHijo == NULL){
				borrar = r;
				r = r->sHermano;
				delete borrar;
			}
		}else{
			ant = a = r;
			while(a && a->dato != v){
				ant = a;
				a = a->sHermano;
			}
			if (a && a->pHijo == NULL){
				ant->sHermano = a->sHermano;
				delete a;
			}
			else{
				a = r;
				while(a){
					BorrarPriv(a->pHijo, v);
					a = a->sHermano;
				}
			}
		}
	}
}

//CONSTRUCTORA
template <class T>
AGeneralImp<T>::AGeneralImp(){
	raiz = NULL;
}

//DESTRUCTORA
template <class T>
AGeneralImp<T>::~AGeneralImp(){
	BorrarArbol(raiz);
	raiz = NULL;
}

//PUBLICOS
template <class T>
void AGeneralImp<T>::ArbolHoja(const T &e){
	raiz = new nodoAG;
	raiz->dato = e;
	raiz->pHijo = raiz->sHermano = NULL;
}

template <class T>
void AGeneralImp<T>::Insertar(const T &v, const T &w){
	InsertarPriv(raiz, v, w);
}
```

```
template <class T>
bool AGeneralImp<T>::EsArbolHoja(){
      return (raiz->pHijo == NULL);
}

template <class T>
bool AGeneralImp<T>::Pertenece(const T &v){
      return PertenecePriv(raiz, v);
}

template <class T>
void AGeneralImp<T>::Borrar(const T &v){
      if(raiz->dato != v){
            if(PertenecePriv(raiz, v)){
                  BorrarPriv(raiz->pHijo, v);
            }
            else
                  cout<<"No se encuentra al dato en el arbol\n";
      }
      else
            cout<<"No se puede borrar la raíz del arbol\n";
}

#endif
```