Estructuras de Datos y Algoritmos

SEGUNDO PARCIAL – 03/12/2009

SOLUCIÓN

_____	_____	_____
Nombre y Apellido	C.I.	Turno

Por favor siga las siguientes indicaciones:

- Escriba con lápiz.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Escriba las hojas de un solo lado.
- Comience cada ejercicio en una hoja nueva.
- El total máximo de puntos del parcial es **60**.
- El parcial contiene un total de: 7 páginas.
- La prueba es individual y sin material.
- Solo se contestan dudas acerca de la letra de los ejercicios.

	Problema 1:	Problema 2:	Problema 3:	Total:
Solo para uso docente				

Problema 1 [25 puntos: 12 parte a) y 13 parte b)]

a) Especificar en C/C++ un TAD que permita trabajar con relaciones binarias, no acotadas, de elementos de dos tipos genéricos S y T. considerar operaciones constructoras, selectoras/destructoras y predicados, además de las siguientes operaciones adicionales:

- $Imagenes\ (x, R) = \{y\ \epsilon\ T\ |(x, y)\epsilon\ R\}$.
- $PreImagenes\ (y, R) = \{x\ \epsilon\ S\ |(x, y)\ \epsilon\ R\}$.
- $EsFuncional\ (R).\ Retorna\ TRUE\ sii\ R\ es\ una\ relacion\ funcional.$

Si usa TADs auxiliares, deberá especificarlos.

```
//Especificación del TAD RelBinST
typedef RelBinST;

RelBinST* CrearRel();
// Crea la Relacion Binaria de elementos SxT vacia.

RelBinST* AgregarRel(ParST *p, RelBinST* R);
// Agrega a la Relacion Binaria R de elementos SxT el par p.
// Precondicion: p no pertenece a R.

bool EsVacia(RelBinST* R);
// Retorna TRUE si y solo si la Relacion Binaria R es vacia.

ParST* ObtenerPar(RelBinST* R);
// Obtiene un par SxT perteneciente a la Relacion Binaria R.
// Precondicion: R no es vacia.

RelBinST* SacarRel(RelBinST* R);
// Elimina un par SxT perteneciente a la Relacion Binaria R.
// Precondicion: R no es vacia.

ListaT* Imagenes(S* x, RelBinST* R);
// Devuelve la lista con las imagenes de x en la Relacion R.

ListaS* PreImagenes(T* y, RelBinST* R);
// Devuelve la lista con las preimagenes de y en la Relacion R.

bool EsFuncional(RelBinST* R);
// Devuelve TRUE si la Relacion Binaria R es una relacion
// funcional.
```

NOTA: podrían usarse conjuntos en vez de listas para trabajar con las imágenes y pre-imágenes de los elementos.

```
// TAD's Auxiliares

// Especificacion del TAD ParST
typedef ParST;

ParST* CrearPar(S* x, T* y);
// Crea el par de elementos (x,y).
```

```
S* ObtenerS(ParST* p);
// Obtiene la componente en S de un par SxT.

T* ObtenerT(ParST* p);
// Obtiene la componente en T de un par SxT.


// Especificacion del TAD ListaGen

typedef ListaGen;
// Podrian considerarse un conjunto o diccionario en vez de una lista
// para las imágenes/preimagenes.

ListaGen* Vacia();
// Retorna la lista vacia.

ListaGen* Cons(Gen* n, ListaGen* s);
// Retorna la lista con el elemento n.

bool esVacia(ListaGen* s);
// Retorna TRUE si s es vacia.

Gen* Primero(ListaGen* s);
// Precondicion: s no es vacia.
// Retorna el primer elemento de s.

ListaGen* Resto(ListaGen* s);
// Precondicion: s no es vacia.
// Retorna la lista s sin el primer elemento.
```

b) Escribir en C/C++ una representación que sirva de implementación para el TAD anterior. **NO** se pide desarrollar el código de las operaciones.

Para implementar el TAD puede utilizarse una lista no ordenada

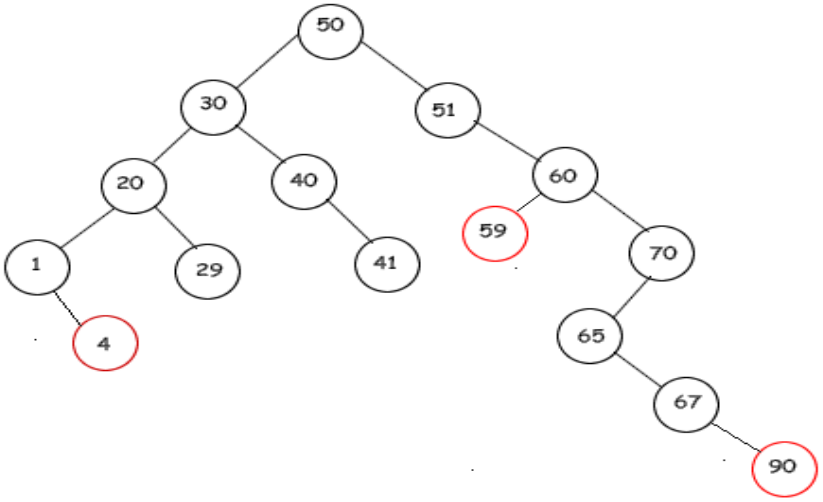
```
struct RelBinST {
      ParST* info;
      RelBinST* sig;
};
```

Problema 2 [5 puntos: 1 parte a), 2 parte b) y 2 parte c)]

Considere la siguiente instancia de un árbol binario de búsqueda de Naturales, que se muestra en la figura.

Se pide:

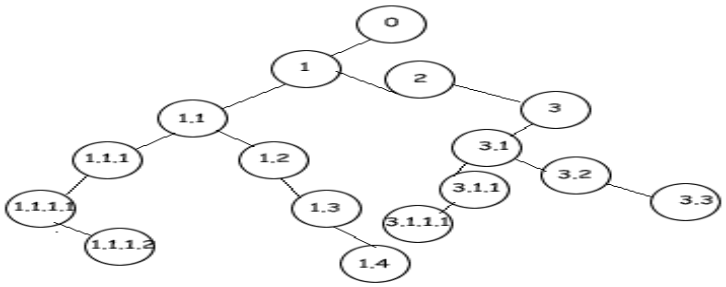
a) Dibujar la instancia del árbol que queda como resultado de agregar al árbol binario de búsqueda de la figura los elementos: 4, 59, 90.



b) Describir el recorrido *InOrden* de los elementos del árbol que obtuvo en la parte a).

1-4-20-29-30-40-41-50-51-59-60-65-67-70-90

c) Dado el siguiente árbol finitario como se muestra en la figura de abajo, dibujar su respectiva representación como árbol binario.



Problema 3 [30 puntos: 6 parte a), 10 parte b) y 14 parte c)]

a) Especificar y dar una representación en C/C++ del TAD árbol binario de búsqueda de palabras (elementos de tipo `string(char*)`).
Considerar las operaciones constructoras, selectoras y predicados, para describir el TAD.

Nota: El orden va a estar dado por la función `strcmp`.

```
// Especificación

ABBString* CrearArbolVacio();
// Post: Retorna el árbol vacío.

ABBString* agregarPalabra(ABBString *t, char* palabra);
// Pre: la palabra 'palabra' no pertenece a t.
// Post: Ingresa la palabra 'palabra' en el árbol binario de búsqueda
t.

bool esVacioABBString(ABBString* t);
// Post: Devuelve TRUE si t no tienen ningún elemento.

char* obtenerRaiz(ABBString* t);
// Pre: el árbol t no es vacío.
// Post: retorna la palabra de la raíz de t.

ABBString* obtenerSubArbolIzquierdo(ABBString* t);
// Pre: NO es vacio t.
// Post: retorna el sub arbol izquierdo de t.

ABBString* obtenerSubArbolDerecho(ABBString *t);
// Pre: NO es vacio t.
// Post: retorna el sub árbol derecho de t.



// Representación

typedef struct ABBString {
      char* palabra;
      ABBStringNodo* izq,*der;

};
```

b) Implementar accediendo directamente a la representación del TAD especificado en la parte a) la operación auxiliar:

```
int getCantidadDeAparicionesDeLaLetra(ABBString *t,char *letra);
// Pre: el árbol t no es vacío
// Post: retorna la cantidad de veces que aparece el carácter
// "letra" en las palabras del árbol t.
```

Se pueden utilizar TAD's y funciones auxiliares, pero debe implementar y describir todo lo utilizado.

```
// Función auxiliar contar letras en palabra
// pre: palabra no es vacía
int cantLetraXEnPalabra(char* palabra, char letra) {
	int largo=strlen(palabra);
	int cont=0;

	for(int i=0;i<largo;i++) {
		if(letra == palabra[i])
		cont++;
	}
	return cont;
}

int getCantidadDeAparicionesDeLaLetra(ABBString *t, char letra) {
	if (t == NULL) {
		return 0;
	} else {

		return cantLetraXEnPalabra(t->palabra, letra)
			+ getCantidadDeAparicionesDeLaLetra(t->izq)
			+ getCantidadDeAparicionesDeLaLetra(t->der);
	}

}
```

c) Implementar **sin** acceder a la representación del TAD especificado en la parte a) la operación auxiliar:

```
void imprimirPorNiveles(ABBString *t);
```

Que imprime los nodos del árbol, recorriendo de izquierda a derecha los distintos niveles del árbol. (Comienza tratando el nivel 1, que sólo contiene el nodo raíz, y seguidamente el nivel 2, el 3,…)

Puede utilizar TADS auxiliares, pero en caso de hacerlo debe dar una especificación mínima de las operaciones selectoras, constructoras y predicados con sus respectivas pre y post condiciones. Además debe dar una breve justificación de la solución implementada.

```
void imprimirPorNiveles(ABBString *t) {
	Cola cola;
	ABBString* aux;

	if (!EsVacioABBString(t)) {
		cola=crearCola();
		encolar(cola, t);

		while (!EsColaVacia(cola)) {
			aux=desencolar(cola);
			printf("%s\n", obtenerRaiz(aux));
			if (!EsVacioABBString (obtenerSubArbolIzquierdo(aux)))
				cola=encolar(cola,obtenerSubArbolIzquierdo(aux));

			if (!EsVacioABBString (obtenerSubArbolDerecho(aux)))
				cola=encolar(cola,obtenerSubArbolDerecho(aux));
		}
	}
}
```

Se uso el TAD Auxiliar Cola de Elementos de tipo ABBString*.
Especificación mínima del TAD Cola.

```
typedef Cola;

Cola* CrearCola();
//Post: crea la cola vacía

Cola* encolar(Cola c, ABBString* t);
//Post: Inserta al final de la cola c el elemento t

bool EsColaVacia(Cola* c);
//Post: Retorna verdadero si la cola c no tienen ningún elemento

ABBString* desencolar(Cola* c);
// Pre: la cola c no es vacía
// Post: Quita el primer elemento de la cola(el más antiguo),
// y lo devuelve
```