Universidad De La República | Administración Nacional de la Educación Pública

Consejo de Educación Técnico Profesional | Facultad de Ingeniería

Estructuras de Datos y Algoritmos

Examen 13/02/2012

_____	_____	_____
Nombre y Apellido	C.I.	Turno

Por favor siga las siguientes indicaciones:

- Escriba con lápiz.
- Escriba su nombre y c.i. en todas las hojas que entregue.
- Numere las hojas he 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 examen es de **100.**
- Se necesitan 60 puntos para aprobar.
- El parcial contiene un total de: **3 carillas.**
- La prueba es individual y sin material.
- Solo se contestan dudas acerca de letra de los ejercicios.

Ejercicio 1(35 Puntos)

Considere la siguiente declaración, en C/C++, del tipo *ListaEnt* de listas encadenadas de enteros:

```
struct nodoLista {
      int info;
      nodoLista *sig; };
typedef nodoLista* ListaEnt;
```

a) Implemente un procedimiento iterativo ***InsOrd*** que dada una lista de enteros L, ordenada de menor a mayor, y dado un número entero K, inserte a K en L de tal manera que la lista siga estando ordenada. Las listas pueden contener elementos repetidos.

b) Implemente un procedimiento recursivo ***Ord*** que dada una lista de enteros, la ordene de menor a mayor. Use el procedimiento *InsOrd* para la definición de *Ord*.

c) ¿Cuál es el orden de tiempo de ejecución en el peor caso de las operaciones *InsOrd* y *Ord*?. Justifique.

SOLUCION

a)
```
ListaEnt insOrd(int k, ListaEnt l){
   nodoLista *aux = new nodoLista;

  aux->info = k;

  aux->sig = NULL;

  nodoLista *lista = l;

  if (lista == NULL){

     l = aux;

  }else{

     bool salir = false;

     if (k <= lista->info){

        aux->sig = l;

        l = aux;

     }else{

        nodoLista *anterior = lista;

        while(lista != NULL && !salir){

           if (k <= lista->info){

              salir = true;

              aux->sig = lista;

              anterior->sig = aux;

           }else{

              anterior = lista;

              lista = lista->sig;
```

```
            }
         }
         if (!salir){
            anterior->sig = aux;
         }
      }
   }
   return l;
}
```

b)
```
ListaEnt Ord(ListaEnt l){
   if (l == NULL){
     return l;
   }else if (l->sig == NULL){
     return l;
   }else{
     return insOrd(l->info, Ord(l->sig));
   }

 }
```

c) La función InsOrd es O(n), por que en el peor caso tendría que recorrer toda la lista.
La función Ord es O(n^2), por que recorre la lista y a su vez hace un llamado a a insOrd en cada paso recursivo.

Ejercicio 2(30 Puntos)

Considere las siguientes declaraciones, en C/C++, del tipo de los árboles binarios de búsqueda de enteros, de tipo ABB:

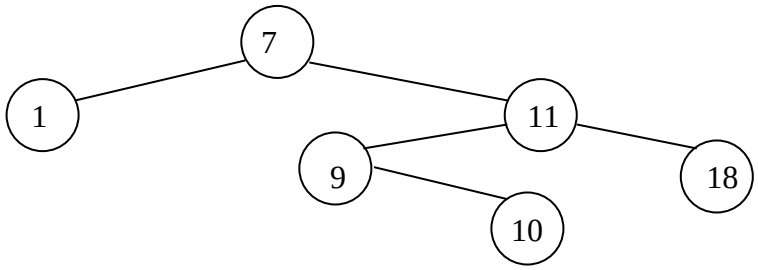
```
struct nodoABB { int dato;
                 nodoABB* izq;
                 nodoABB* der; };
typedef nodoABB *ABB;
```

Se pide implementar las siguientes funciones recursivamente, accediendo directamente a la representación y sin usar operaciones auxiliares propias **:**

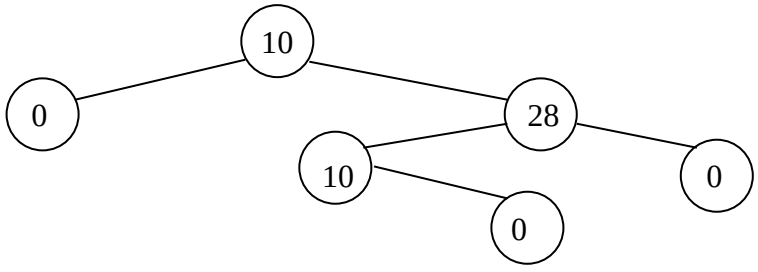
a) Max, que dado un árbol A de tipo ABB, devuelve el valor del mayor elemento de A.

b) Min, que dado un árbol A de tipo ABB, devuelve el valor del menor elemento de A.

c) *Anotar*, que dado un árbol A de tipo ABB, devuelve otro árbol estructuralmente igual a A que contiene en cada nodo la suma del mayor elemento de la izquierda con el menor elemento de la derecha (Max(izq) + Min(der)).



El resultado de Anotar para el árbol anterior seria:



SOLUCION

```
int Max(ABB t){
   if (t == NULL){
      return 0;
   }else if (t->der == NULL){
      return t->dato;
   }else{
      return Max(t->der);
   }
}

int Min(ABB t){
   if (t == NULL){
      return 0;
   }else if (t->izq == NULL){
      return t->dato;
   }else{
      return Min(t->izq);
   }
}
```

```
ABB Anotar(ABB t){
   if (t == NULL){
      return NULL;
   }else{
      ABB nuevo = new nodoABB;
      nuevo->dato = (Max(t->izq) + Min(t->der));
      nuevo->izq = Anotar(t->izq);
      nuevo->der = Anotar(t->der);
      return nuevo;
   }
}
```

Ejercicio 3(35 Puntos)

a) Especifique en C/C++, con pre y postcondiciones, un TAD Set (Conjunto) no acotado de enteros que contenga operaciones constructoras, predicados y las operaciones de conjuntos unión e intersección.

b) Implemente el TAD Set anterior usando Listas encadenadas como representación. Desarrolle el código de todas las operaciones especificadas en la parte a), las operaciones de unión e intersección deben devolver un nuevo conjunto que no comparte memoria con los originales.

c) ¿Qué cambia en la especificación de la parte a) si se considera el modelo acotado para un Set? Explique muy brevemente.

d) ¿Es igualmente válida la implementación de la parte b) para la especificación de la parte c), si se pretende que los predicados se implementen sin recorrer la estructura elegida como representación? Explique muy brevemente.

SOLUCION

a)

```
Set crearSet();

//Pre: N.A.

//Post: devuelve un set vacio


Set add(int x, Set s);

//Pre: x no pertenece a s

//Post: agrega x al Set s


bool pertenece(int x, Set s);

//Pre: N.A.

//Post: retorna true si x pertenece a s


bool isEmpty(Set s);

//Pre: N.A.

//Post: retorna true si s es vacio


Set unionSet(Set s1, Set s2);

//Pre: N.A.

//Post: retorna un nuevo set que tiene los elementos que estan en s1 y s2


Set intersectionSet(Set s1, Set s2);

//Pre: N.A.

//Post: retorna un nuevo set que tiene los elementos que estan a la vez en s1 y s2
```

```
void borrarElem(Set s, int x);
//Pre: s no es vacio
//Post: elimina a x de s, si x no pertenece no hace nada
```

b)

```
struct nodo{
        int elem;
        nodo* sig;
}

typedef nodo* Set;


Set crearSet(){
    return NULL;
}

Set add(int x, Set s){
    nodo* n = new nodo;
    n->elem = x;
    n->sig = s;
    s = n;
    return s;
}
```

```
bool pertenece(int x, Set s){
	while ((s != NULL) && (s->elem != x))
		s = s->sig;
	if (s == NULL)
		return false;
	else
		return true;
}

bool isEmpty(Set s){
	return (s == NULL);
}

Set unionSet(Set s1, Set s2){
	Set res = crearSet();
	while (s1 != NULL){
		res = add(s1->elem, res);
		s1 = s1->sig;
	}
	while (s2 != NULL){
		if (!pertenece(s2->elem, res))
			res = add(s2->elem, res);
		s2 = s2->sig;	}
	return res;
}
```

```
Set intersectionSet(Set s1, Set s2){
        Set res = crearSet();
        if (s2 != NULL){
                while (s1 != NULL){
                        if (pertenece(s1->elem, s2)){}
                                res = add(s1->elem, res);
                        s1 = s1->sig;
                }
        }
        return res;
```

c) Se necesita una nueva operación que nos diga si el Set esta lleno.

d) No, necesitaríamos agregar un atributo de tipo int que nos diga la cantidad de elementos del Set.