Universidad De La República | Administración Nacional de la Educación Pública
Consejo de Educación Técnico Profesional | Facultad de Ingeniería

Estructuras de Datos y Algoritmos

Examen 12/12/2011

_____	_____	_____
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

Considere la siguiente declaración, en C, del tipo de los nodos de listas dinámicas de enteros:

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

Implemente en C una función iterativa **F** que dadas dos listas ordenadas (en forma ascendente) de enteros, C1 y C2, y sin elementos repetidos que representan a dos conjuntos, retorne una nueva lista ordenada y sin elementos repetidos que represente al conjunto **(C**1-**C**2)∪(**C**2-**C**1).

A tener en cuenta:

- La nueva lista no deberá compartir registros de memoria con las listas parámetros.
- Se requiere que esta función recorra a lo sumo una vez cada lista parámetro.
- No se permite usar TADs auxiliares.
- Si usa funciones o procedimientos auxiliares deberá implementarlos, pero recuerde que cada una de las listas pueden recorrerse a lo sumo una vez (se usen o no funciones o procedimientos auxiliares).

La firma de la función es la siguiente: **`nodoLista * F (nodoLista * C1, nodoLista * C2);`**

SOLUCIÓN

```
// Devuelve una lista ordenada de mayor a menor en caso de que C1 y/o C2 no sean vacías.
nodoLista * F (nodoLista * C1, nodoLista * C2){
  nodoLista *devolver;
  devolver = NULL;
  int info;
  while (C1 != NULL || C2 != NULL){
    if (C1 != NULL){
      if (C2 != NULL){
        if (C1->info == C2->info){
          C1 = C1->sig;
          C2 = C2->sig;
        }else if (C1->info < C2->info){
          info = C1->info;
          C1 = C1->sig;
        }else if (C1->info > C2->info){
          info = C2->info;
          C2 = C2->sig;
        }
      }else{
        info = C1->info;
        C1 = C1->sig;
      }
    }else{
      info = C2->info;
      C2 = C2->sig;
    }
    nodoLista *aux = new nodoLista;
    aux->info = info;
    aux->sig = devolver;
    devolver = aux;
  }
  return devolver;
}
```

Ejercicio 2

Parte a) Implemente un procedimiento recursivo ***borrarMayores*** que dados un árbol binario de búsqueda de enteros A de tipo ABB (sin elementos repetidos) y un entero x, elimine de A todos los elementos mayores a x. El árbol resultado debe ser un árbol binario de búsqueda.

NOTAS: Se sugiere usar un procedimiento auxiliar, que deberá ser implementado, que elimine todos los nodos de un árbol, dejando a éste vacío. No se permite definir procedimientos ni funciones auxiliares, además del procedimiento sugerido. Tampoco se pueden definir estructuras de datos auxiliares. No se permite recorrer el árbol más de una vez. Deberá liberarse la memoria de todas las celdas cuyos elementos sean eliminados.

SOLUCIÓN

```
struct nodoArbol{
  int valor;
  nodoArbol * izq;
  nodoArbol * der;
};

typedef struct nodoArbol * ArbolBB;

void borrarArbol(ArbolBB &arbol){
  if (arbol != NULL){
    borrarArbol(arbol->izq);
    borrarArbol(arbol->der);
    delete arbol;
    arbol = NULL;
  }
}


void borrarMayores(ArbolBB &A, int x){
  if (A != NULL){
    if (A->valor <= x){
      borrarMayores(A->der, x);
    }else{
      borrarArbol(A->der);
      ArbolBB tmp = A->izq;
      borrarMayores(tmp, x);
      delete A;
      A = tmp;
    }
  }
}
```

Parte b) ¿Cuál es el orden (O) de tiempo de ejecución en el peor caso del procedimiento ***borrarMayores***?. Explique breve e informalmente.

SOLUCIÓN

BorrarMayores es O(N) en peor caso. Este se da cuando X es menor a todos los nodos del árbol.

Ejercicio 3

Se quiere especificar e implementar un TAD *ColaDoble* que permita insertar y eliminar elementos (de a uno por vez) al comienzo y al final de una estructura lineal. Se pide:

a) Especificar, incluyendo pre y postcondiciones, el TAD *ColaDoble* de elementos de un tipo genérico adecuado para modelar una estructura lineal no acotada, con operaciones que permitan:
- Crear la estructura vacía,
- Agregar un elemento al comienzo de la estructura lineal,
- Agregar un elemento al final de la estructura lineal,
- Testear por la estructura vacía,
- Eliminar y retornar el elemento al comienzo de la estructura lineal (si la estructura es no vacía),
- Eliminar y retornar el elemento al final de la estructura lineal (si la estructura es no vacía),

b) Implementar el TAD anterior, sin usar TADs auxiliares, de tal manera que todas las operaciones (con excepción de la que crea la estructura vacía) sean O(1) peor caso. Desarrolle el código de las siguientes 3 operaciones: la operación que crea la estructura vacía; la que elimina y retorna un elemento al final; y, la que agrega un elemento al comienzo. Omita el código del resto de las operaciones del TAD.

SOLUCIÓN

```
struct nodoLista{
  int valor;
  nodoLista * sig;
  nodoLista * ant;
};

typedef struct nodoLista * NodoDE;
```

```
struct cabezal{
  NodoDE primero;
  NodoDE ultimo;
};

typedef struct cabezal * ListaDE;


// precondicion: no tiene
// postcondicion: Crear y retorna la estructura vacía
ListaDE crearVacia(){
  ListaDE de = new cabezal;
  de->primero = NULL;
  de->ultimo = NULL;
  return de;
}

// precondicion: recibe una estructura lineal creada.
// postcondicion: Agregar un elemento al comienzo de la estructura lineal
void AgregarComienzo(ListaDE de, int valor){
  NodoDE nodo = new nodoLista;
  nodo->valor = valor;
  nodo->ant = NULL;
  nodo->sig = de->primero;
  de->primero = nodo;
  if (de->ultimo == NULL){
    de->ultimo = de->primero;
  }
}

// precondicion: recibe una estructura lineal creada.
// postcondicion: Agregar un elemento al final de la estructura lineal
void agregarFinal(ListaDE de, int valor);

// precondicion: recibe una estructura lineal creada.
// postcondicion: devuelve true si la estructura esta vacia, false en cualquier otro caso.
bool esVacia(ListaDE de);

// precondicion: la estructura de es no vacia.
// postcondicion: Eliminar y retornar el elemento al comienzo de la estructura lineal
int elementoComienzo(ListaDE de);
```

```
// precondicion: la estructura de es no vacia.
// postcondicion: Eliminar y retornar el elemento al final de la estructura lineal
int elementoFinal(ListaDE de){
  NodoDE nodo = de->ultimo;
  int devolver = nodo->valor;
  if (nodo->ant == NULL){
    de->ultimo = NULL;
    de->primero = NULL;
  }else{
    de->ultimo = nodo->ant;
    de->ultimo->sig = NULL;
  }
  delete nodo;
  nodo = NULL;
  return devolver;
}
```