# Examen de Programación 2

Febrero de 2023

## Problema 1 (40 puntos: 15+15 + 5 + 5)

Considere la siguiente definición para árboles binarios de enteros (*AB*):

```
struct nodoAB{
    unsigned int dato;
    nodoAB *izq, *der;
}
typedef struct nodoAB* AB;
```

**Parte a)** Implemente un procedimiento recursivo (sin usar iteración) ***borrarTodo*** que dado un árbol binario ***t*** de tipo *AB*, elimine de ***t*** todos sus nodos y devuelva la cantidad de nodos borrados. Si ***t*** es NULL el procedimiento no tendrá efecto y devuelve 0.

```
int borrarTodo(AB & t)
```

**Parte b)** Implemente un procedimiento recursivo (sin usar iteración) ***borrar*** que dados un árbol binario ***t*** de tipo *AB* y un entero ***x***, elimine de ***t*** a ***x*** y a sus subárboles, si ***x*** está en ***t*** y devuelva la cantidad de nodos borrados; en caso contrario (en particular si ***t*** es NULL) el procedimiento no tendrá efecto y devuelve 0. Asumimos que ***t*** no tiene elementos repetidos, aunque no puede asumirse que los elementos en ***t*** están ordenados como en un árbol binario de búsqueda.

```
// PRE: t no tiene elementos repetidos
int borrar(AB & t, int x)
```

**Parte c)** ¿Cuál es el orden de tiempo de ejecución para el peor caso de ***borrar***? Explique muy brevemente.

**Parte d)** ¿Cambia en algo la implementación dada de ***borrar*** si ahora ***t*** admitiera elementos repetidos y se debieran eliminar todas las ocurrencias de ***x*** (y sus subárboles) en ***t***? Explique muy brevemente.

## Problema 2 (60 puntos: (35 + 5) + 20)

Un empresa quiere desarrollar una aplicación para gestionar los comentarios que recibe en su página web, sobre artículos que publica. Cada comentario tiene un texto y un número (entero no negativo) que identifica al artículo involucrado. Los comentarios se muestran desde el más reciente, en primer lugar, hasta el más antiguo, en último lugar. Siempre es posible ingresar nuevos comentarios, aunque solamente estarán disponibles los últimos *k* ingresados, siendo *k* un valor entero parametrizado en la aplicación.

Considere la siguiente especificación del TAD ***Comentarios***:

```
struct representacionComentarios;
typedef representacionComentarios* Comentarios;

// Pre: k>0
// Pos: crea una instancia Comentarios vacía que puede guardar, a lo sumo,
//los k comentarios más recientes
     Comentarios crear(unsigned int k);

// Pos: ingresa un nuevo comentario manteniendo como máximo los k comentarios
//más recientes, siendo k el parámetro definido en crear
     void ingresar(Comentarios &c, char* texto, unsigned int idArticulo);

// Pos: retorna true si y sólo si es vacío
     bool esVacio(Comentarios c);

// Pre: !esVacio(c)
//Pos: retorna el número de artículo al que refiere el comentario más reciente
     unsigned int idMasReciente(Comentarios c);

// Pre: !esVacio(c)
```

# Examen de Programación 2

## Febrero de 2023

```
// Pos: retorna el texto del artículo al que refiere el comentario más reciente
     char* textoMasReciente(Comentarios c);

// Pos: retorna la cantidad total de comentarios existentes
     int cantidad(Comentarios c)

// Pos: elimina todos los comentarios recibidos sobre el artículo dado
     void eliminar(Comentarios &c, unsigned int idArticulo);

// Post retorna un copia exacta que no comparte memoria
     Comentarios copia(Comentarios c);

// Pos: elimina la instancia del TAD, liberando su memoria
     void destruir(Comentarios &c);
```

Se pide:

**Parte a)**

1. Implementar el TAD ***Comentarios***, de manera que todas las operaciones salvo *eliminar, copia* y *destruir* tengan tiempo de ejecución O(1) en el peor caso. Desarrolle únicamente la representación ***representacionComentarios*** y las operaciones: ***crear***, ***ingresar***, ***idMasReciente*** y ***eliminar***. Asuma implementadas las demás operaciones.
2. ¿Cuál es el Orden de tiempo de ejecución en peor caso de la operación ***eliminar***? Justifique brevemente.

**Parte b)**

Implemente la función iterativa ***masPopular*** que dada una instancia de Comentarios no vacía ***c*** retorna el identificador del artículo con mayor cantidad de comentarios en ***c,*** sin acceder a la representación del TAD ni modificar ***c***. En caso que haya más de un artículo con la cantidad máxima de comentarios recibidos, debe retornar el que tenga el identificador menor entre éstos.

```
// Pre: !esVacio(c)
// Pos: retorna el identificador del artículo más comentado.
// Si hay varios, retorna el id menor entre éstos
       unsigned int masPopular(Comentarios c)
```

**Ejemplos**

| Comentarios ***c***, k=5 | Operación | Resultado |
|---|---|---|
| (Muy buen producto, 123), (Malo, 13), (Muy lindo, 123), (Regular, 13), (Lo recomiendo, 125) | `cantidad(c)` | 5 |
| (Muy buen producto, 123), (Malo, 13), (Muy lindo, 123), (Regular, 13), (Lo recomiendo, 125) | `idMasReciente(c)` | 123 |
| (Muy buen producto, 123), (Malo, 13), (Muy lindo, 123), (Regular, 13), (Lo recomiendo, 125) | `textoMasReciente(c)` | Muy buen producto |
| (Muy buen producto, 123), (Malo, 13), (Muy lindo, 123), (Regular, 13), (Lo recomiendo, 125) | `eliminar(c, 123)` | (Malo, 13), (Regular, 13), (Lo recomiendo, 125) |
| (Malo, 13), (Regular, 13), (Lo recomiendo, 125) | `cantidad(c)` | 3 |
| (Malo, 13), (Regular, 13), (Lo recomiendo, 125) | `idMasReciente(c)` | 13 |
| (Malo, 13), (Regular, 13), (Lo recomiendo, 125) | `textoMasReciente(c)` | Malo |

# Examen de Programación 2

Febrero de 2023

## Problema 1 – Solución

### Parte a)

```
int borrar(AB & t, int x){
int borrados=0;
        if (t != NULL){
                if (t->dato == x)
                        borrados =borrarTodo(t);
                else{
                        borrados = borrar(t->izq, x) +
                        borrar(t->der, x);
                }
        }
    return borrados;
}

//Alternativamente, sin recorrer nodos innecesariamente
int borrar(AB & t, int x){
int borrados=0;
      if (t != NULL){
        if (t->dato == x)
                borrados =borrarTodo(t);
        else{
                 borrados=borrar(t->izq, x);
                if  (borrados <1)
                        borrados=borrar(t->der, x);
        }
    }
   return borrados;
}
```

```
int borrarTodo(AB & t){
int borrados=0;
        if (t != NULL){
            borrados = 1+ borrarTodo(t->izq, x)+
                borrarTodo(t->der, x);
                delete t;
                t = NULL;
        }
    return borrados;
}
```

### *Parte b*)e

*El procedimiento borrar, junto con el auxiliar borrarTodo, da un solución al problema orden de tiempo de ejecución O(n) en el pero caso, siendo n la cantidad de elementos o nodos, ya que recorre todo el árbol. Notar que sobre cada nodo del árbol se hace acciones de O(1) (de comparación o liberación de memoria).*

### *Parte c*)

*No cambia nada en el código de la primera alternativa procedimiento borrar, ya que se recorre todo el árbol y si elemento está repetido, se eliminarán todas sus ocurrencias y subárboles asociados. La segunda alternativa debe transfomarse en la primera.*

## Problema 2 - Solución

### Parte a)

1. La representación propuesta contempla una lista simplemente encadenada de comentarios de artículos, con puntero al inicio y al final. Los comentarios se insertan al final (tope) y, cuando se superan los *k* comentarios recibidos se elimina al inicio (el más viejo), para no superar en ningún momento el registro de los *k* comentarios más recientes. Luego, el comentario más reciente está siempre al final (tope de esta pila especial).

```
struct nodo{
   char* texto;
   unsigned int id;
   nodo* sig;
}

struct representacionComentarios{
   nodo* primero; // sobre inicio está el primer comentario recibido (considerando la limitante de la cota)
   nodo* ultimo;// sobre final se ingresan comentarios nuevos y obtiene el más reciente
   unsigned int cota;  // valor que restringe la cantidad máxima de comentarios a considerar (k, según la letra)
   unsigned int cant; // cantidad total de comentarios recibidos (menor o igual a cota)
```

```
}
typedef representacionComentarios* Comentarios;

// Pre: k>0
// Pos: crea una instancia Comentarios vacía que puede guardar, a lo sumo, los k comentarios más recientes
Comentarios crear(unsigned int k){
    Comentarios result = new representacionComentarios;
    result->primero = NULL;
    result->ultimo = NULL;
    result->cota = k;
    result->cantidad = 0;
    return result;
}

/* Pos: ingresa un nuevo comentario manteniendo como máximo los k comentarios más recientes, siendo k el
parámetro definido en crear */
void ingresar(Comentarios &c, char* texto, unsigned int idArticulo){
    nodo* nuevoNodo = new nodo;
    nuevoNodo->texto = texto;
    nuevoNodo->id = idArticulo;
    nuevoNodo->sig = NULL;
    if (c->primero == NULL){
        c->primero = nuevoNodo;
        c->ultimo = nuevoNodo;
    } else{
        c->primero->sig = nuevoNodo;
        c->primero = nuevoNodo;
    }
    if (c->cantidad == c->cota){
        nodo* aEliminar = c->ultimo;
        c->ultimo = c->ultimo->sig;
        delete aEliminar;
    } else
        c->cantidad++;
}

// Pre: !esVacio(c)
//Pos: retorna el número de artículo al que refiere el comentario más reciente
unsigned int idMasReciente(Comentarios c){
    return c->primero->id;
}

// Pos: elimina todos los comentarios eventualmente recibidos sobre el artículo dado
void eliminar(Comentarios &c, unsigned int idArticulo){
    while (c->ultimo != NULL && c->ultimo->id == idArticulo){
        nodo* aEliminar = c->ultimo;
        c->ultimo -> c->ultimo->sig;
        delete aEliminar;
        c->cantidad--;
    }
    if (c->ultimo != NULL){
        nodo* iter = c->ultimo;
        while (iter->sig != NULL){
            if (iter->sig->id == idArticulo){
                nodo* aEliminar = iter;
                iter->sig = iter->sig->sig;
                delete aEliminar;
                c->cantidad--;
            }
```

```
            iter = iter->sig;
        }
    }
}
```

**2.** *La operación eliminar tiene O(n) de tiempo de ejecución en el peor caso, ya que debe recorrer toda la lista comparando el id del artículo en cada nodo. Para cada elemento/nodo la acción insume O(1).*

***Parte b)***

```
unsigned int masPopular(Comentarios c){
    Comentarios copia = copia(c);
    unsigned int idMasComentado; // id del artículo más popular
    int masRepetido = MIN_INT; // repeticiones del artículo más popular
    int cantidadAntes;
    int cantidadDespues;
    while (!esVacio(copia)){
        cantidadAntes = cantidad(copia);
        unsigned int idMasReciente = idMasReciente(copia);
        eliminar(copia, idMasReciente);
        cantidadDespues = cantidad(copia);
        if (cantidadAntes-cantidadDespues > masRepetido ||
                (cantidadAntes-cantidadDespues == masRepetido && idMasReciente < idMasComentado)){
                    idMasComentado = idMasReciente;
                    masRepetido = cantidadAntes-cantidadDespues;
        }
    }
    destruir(copia);
    return idMasComentado;
}

// Pre: !esVacio(c)
    // Pos: retorna el identificador del artículo más comentado. Si hay varios, retorna el id menor entre éstos
    unsigned int masPopular(Comentarios c){
        Comentarios copia = copia (c);
        int cantMayor = 0;
        unsigned int idMayor;
        unsigned int idActual;
        while (!esVacio(copia)){
            idActual = idMasReciente (copia);
            cantAnterior = cantidad (copia)
            eliminar (copia, idActual);
            if (cantAnterior - cantidad (copia) > cantMayor || (cantAnterior - cantidad (copia) == cantMayor) && (idActual
    < idMayor)){
                cantMayor = cantAnterior - cantidad (copia);
                idMayor = idActual;
            }
        }
        destruir (copia);
        return idMayor;
    }
```