Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

15 de Febrero de 2019

Problema 1 (50 puntos)

a) Defina una estructura de tipo **LISTA** que represente el tipo de datos listas de enteros en memoria dinámica, simplemente encadenadas.

```
typeded nodoLista* LISTA;
struct nodoLista { int dato; LISTA sig; }
```

b) Defina una estructura de tipo **ABB** que represente el tipo de datos árboles binarios de búsqueda de enteros en memoria dinámica.

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

c) Implemente una función int maximo(ABB a) que dado un árbol binario de búsqueda *a* de tipo **ABB** no vacío, retorne el valor máximo que contiene. La función debe evitar recorrer nodos de *a* que no sean estrictamente necesarios.

```
int maximo(ABB a){
        while (a->der != NULL)
            a = a->der;
        return a->dato;
}
```

d) Implemente una función **Lista listaOrdenada(ABB a, int k)** que dado un árbol binario de búsqueda *a* de tipo **ABB**, retorne una lista de enteros ordenada de menor a mayor que contenga todos los elementos de *a* menores a *k*. Si no hay elementos de *a* menores a *k* la función debe retornar la lista vacía. Puede utilizar operaciones auxiliares, que deberá implementar. La función debe tener tiempo de ejecución O(*n*) en el peor caso, siendo *n* la cantidad de nodos del árbol *a*, y debe evitar recorrer nodos de *a* que no sean estrictamente necesarios.

```
LISTA listaOrdenada(ABB a, int k){
        LISTA result = NULL;
        listaOrdenadaAcumulativa(a, k, result);
        return result;
}

/* Deja en la lista result (al comienzo) todos los elementos menores a k del ABB a,
ordenados de menor a mayor. */
void listaOrdenadaAcumulativa(ABB a, int k, LISTA & result){
        if (a != NULL){
```

```
                if (a->dato < k){
                        listaOrdenadaAcumulativa(a->der, k, result);
                        insertarComienzo(result, a->dato);
                }
                listaOrdenadaAcumulativa(a->izq, k, result);
        }
}

// Inserta el elemento x al comienzo de la lista l.
void insertarComienzo(LISTA & l, int x){
        LISTA nodo = new nodoLista;
        nodo->dato = a->dato;
        nodo->sig = l;
        l = nodo;
}
```

e) Dos árboles binarios de búsqueda son similares si contienen los mismos elementos. Usando las operaciones desarrolladas anteriormente (partes c) y d)), implemente una función que retorne true si y sólo si dos árboles binarios de búsqueda son similares: **bool similares(ABB a, ABB b)**.

```
bool similares(ABB a, ABB b){
        LISTA la = listaOrdenada(a, maximo(a)+1);
        LISTA lb = listaOrdenada(b, maximo(b)+1);
        while (la!=NULL && lb!=NULL && la->dato==lb->dato){
                borrarPrimero(la);
                borrarPrimero(lb);
        }
        bool result = (la==NULL && lb==NULL);
        destruirLista(la);
        destruirLista(lb);
        return result;
}

// elimina el primer elemento de la lista, si no es vacía.
void borrarPrimero(Lista & l){
        if (l!=NULL){
                LISTA aBorrar = l;
                l = l->sig;
                delete aBorrar;
        }
}

// vacía la lista liberando la memoria asociada a ésta.
void destruirLista(Lista & l){
        while (l!=NULL)
                borrarPrimero(l);

}
```

Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

15 de Febrero de 2019

f) Explique brevemente cuál es el orden de tiempo de ejecución en peor caso de la función similares desarrollada en e)

La función es O(n+m) en el peor caso, siendo n y m la cantidad de elementos (nodos) de cada árbol parámetro. La función listaOrdenada recorre una sola vez cada nodo del árbol parámetro, insumiendo O(1) para incorporar cada elemento a la lista ordenada resultante. La funciones auxiliares y el recorrido de las listas resultantes de aplicar listaOrdenada tienen a lo sumo el orden de la cantidad de elementos de la estructura. Luego, el orden de tiempo de ejecución en el peor caso de la función similares se justifica a partir de la regla de la suma (el máximo en una secuencia), como sigue:

```
bool similares (ABB a, ABB b){
        LISTA la = listaOrdenada(a, maximo(a)+1);                       //O(n)
        LISTA lb = listaOrdenada(b, maximo(b)+1);                       //O(m)
        while (la!=NULL && lb!=NULL && la->dato==lb->dato){
                borrarPrimero(la);                                      //O(1)
                borrarPrimero(lb);                                      //O(1)
        }                                                         //O(min(n,m))
        bool result = (la==NULL && lb==NULL);                           //O(1)
        destruirLista(la);                                              //O(n)
        destruirLista(lb);                                              //O(m)
        return result;                                                  //O(1)
}
```

Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

15 de Febrero de 2019

Problema 2 (50 puntos)

Un sistema de administración de información maneja datos de dos categorías. La información en cada caso consiste de un elemento de un tipo *string* (*char* *), un identificador entero en el rango [0..K] (siendo K una constante) y el tipo de categoría (0 ó 1). Cuando llega información al sistema de administración, se almacena esperando su turno. No son admisibles incorporaciones con identificadores repetidos (en espera de ser atendidas). El procesamiento de la información en el sistema de administración respeta la siguiente política: primero se procesa la información de categoría 1 y, si no hay información de categoría 1, recién luego se atiende información de categoría 0. Dentro de cada categoría (0 ó 1) se sigue la política FIFO.

Se pide:

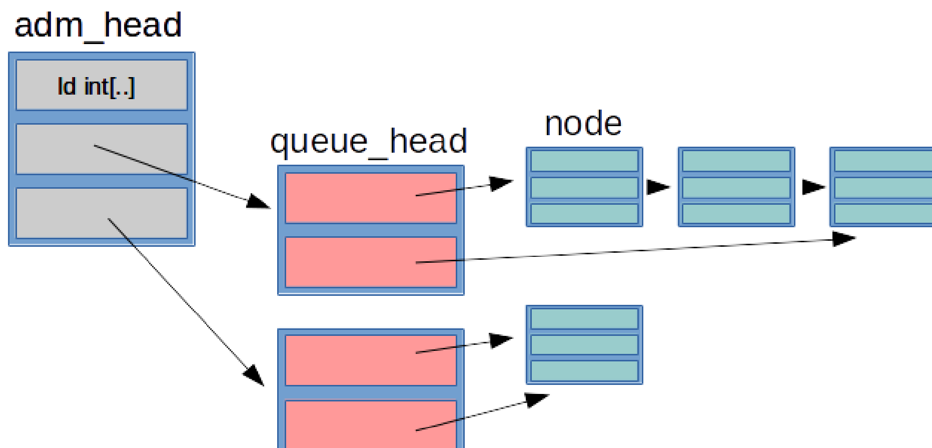
a) Especifique un TAD **ADM** que permita resolver adecuadamente el sistema de administración de información previamente descrito. Considere las siguientes operaciones:

- ***CrearADM***, que retorna un sistema de administración de información vacío.
- ***Insertar(e,cat,id,adm)***, que agrega un elemento *e* (un *string*) con categoría *cat*, si es que no existe el *id* en un sistema *adm* dado, de lo contrario no tiene efecto. Considere definido el tipo enumerado *enum Categoria {0,1}*.
- ***EsVacio,*** que retorna true si y solo si un sistema dado no tiene elementos.
- ***Recuperar***, que retorna el elemento, de tipo *string*, que corresponda (según la política definida para el sistema de administración de información) de un sistema no vacío.
- ***Eliminar***, que borra el elemento que corresponda (según la política definida para el sistema de administración de información) si el sistema no es vacío, de lo contrario no tiene efecto.
- ***Destruir***, que destruye un sistema de administración de información, liberando toda la memoria asociada a éste.
- ***Clon***, que retorne una copia de un sistema de administración de información, que no comparte memoria.

```
typedef adm  * ADM;

/* ****** Constructoras ******** */
/*
 * Post: Devuelve un sistema administrativo vacio
 */
ADM CrearADM ();

/*
 * Pre:  adm es un sistema administrativo
 * Post: adm tiene incluido al elemento formado por los componentes e, cat y id (si el id no existe en
adm)
 */
void Insertar (char* e, Categoria cat, int id, ADM & adm);
```

```
/* ****** Selectoras ******** */
/*
 * Pre: adm es un sistema administrativo
 * Pre: adm es no vacio
 */
char * Recuperar (ADM adm);

/* ****** Operaciones adicionales ******** */

/*
 * Pre: adm es un sistema administrativo
 */
bool EsVacio (ADM adm);

/*
 * Clon, que retorne una copia de un sistema de administración de información, que no comparte memoria.
 */
ADM Clon (ADM adm);


/* ****** Destructoras ******** */
/*
 * Pre:  adm es un sistema administrativo
 * Post: Se elimina al elemento que corresponde segun la politica definida de adm. En caso de ser un
adm vacio, la operacion no tiene efecto
 */
void Eliminar (ADM & adm);

/*
 * Pre:  adm es un sistema administrativo
 * Post: Se libera la memoria utilizada por adm.
 */
void Destruir (ADM & adm);
```

b) Desarrolle una implementación del TAD **ADM**, sin usar TADs auxiliares, en la cual las operaciones *Insertar*, *Eliminar* y *Recuperar* sean O(1) peor caso. Para esto, defina primero y explique (puede ser gráficamente) la representación elegida para el TAD y luego implemente las operaciones para la representación propuesta. Asuma que las operaciones *Destruir* y *Clon* están implementadas. Puede utilizar operaciones sobre strings (char *), tales como *strcpy* y *strcmp*.



Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

15 de Febrero de 2019

```
#include <strings.h>
#define K ...

struct node
{
    char * elem;
    int  id;
    node * sig;
};

struct queue_head
{
    node * first;
    node * last;
};

struct adm_head
{
    bool id[K + 1];
    queue_head *  queue_cat0;
    queue_head *  queue_cat1;
};

typedef adm_head  * ADM;

ADM CrearADM ()
{
    ADM adm = new (adm_head);
    //inicializo los id como libres
    for (int i = 0; i <= K; i++)
    {
        adm->id[i] = false;
    }
    adm->queue_cat0 = new (queue_head);
    adm->queue_cat0->first = NULL;
    adm->queue_cat0->last  = NULL;

    adm->queue_cat1 = new (queue_head);
    adm->queue_cat1->first = NULL;
    adm->queue_cat1->last  = NULL;
    return adm;
}

void Insertar (char* e, Categoria cat, int id, ADM & adm)
{
    if (adm->id[id] = false)
    {
        adm->id[id]  = true;                       //seteo el id como ocupado
        node * qnode = new (node);
        qnode->elem  = new char[strlenth(e) + 1];
        strcpy (qnode->elem, e);
        qnode->id  = id;
        qnode->sig = NULL;
        if (cat == 0)                        //filtro por categoria
        {
            if (adm->queue_cat0->first = NULL)
            {
                adm->queue_cat0->first = qnode;
```

```
                adm->queue_cat0->last  = qnode;
            }
            else
            {
                adm->queue_cat0->last->sig  = qnode;
                adm->queue_cat0->last       = qnode;
            }
        }
        else
        {
            if (adm->queue_cat1->first = NULL)
            {
                adm->queue_cat1->first = qnode;
                adm->queue_cat1->last  = qnode;
            }
            else
            {
                adm->queue_cat1->last->sig  = qnode;
                adm->queue_cat1->last       = qnode;
            }
        }
    }
}

char* Recuperar (ADM adm)
{
    if (adm->queue_cat1->first != NULL)
        return adm->queue_cat1->first->elem;
    else
        return adm->queue_cat0->first->elem;
}

bool EsVacio (ADM adm)
{
    return (adm->queue_cat1->first == NULL && adm->queue_cat0->first == NULL)
}

void Eliminar (ADM & adm)
{
    node * aux =  NULL;
    if (adm->queue_cat1->first != NULL)
    {
        aux = adm->queue_cat1->first;
        adm->queue_cat1->first = adm->queue_cat1->first->sig;
    }
    else
    {
        aux = adm->queue_cat0->first;
        adm->queue_cat0->first = adm->queue_cat0->first->sig;
    }
    adm->id[aux->id] = false;
    delete aux->elem;
    delete aux;
}
```

c) Defina una función ***indistinguibles*** que, dados *adm1* y *adm2* del tipo abstracto ADM, retorne *true* si y sólo si los elementos, de tipo *string*, que se obtienen de *adm1* y *adm2* son iguales, y en el mismo orden. La función no debe modificar *adm1* ni *adm2*. Tampoco es posible acceder a la representación del TAD ADM. Puede utilizar la operación *strcmp* para comparar strings.

Examen de Programación 2

15 de Febrero de 2019

bool indistinguibles (ADM* adm1, ADM* adm2)

```
bool indistinguibles (ADM * adm1, ADM * adm2)
{
    ADM adm1_clon = Clon (adm1);
    ADM adm2_clon = Clon (adm2);
    bool iguales  = true;
    while (!EsVacio (adm1_clon) && !EsVacio (adm2_clon) && iguales)
    {
        char* elem1 = Recuperar (adm1_clon);
        char* elem2 = Recuperar (adm2_clon);
        if (strcmp (elem1, elem2) != 0)             //Si no son iguales
        {
            iguales = false;
        }
        else
        {
            Eliminar (adm1_clon);
            Eliminar (adm2_clon);
        }
    }//fin while
    if (!EsVacio (adm1_clon) || !EsVacio (adm2_clon))
        iguales = false;
    Destruir (adm1_clon);
    Destruir (adm2_clon);
    return iguales;
}
```