Instituto de Computación - Facultad de Ingeniería, Universidad de la República

Segundo parcial de Programación 2

8 de Julio de 2016

Generalidades:

a. La prueba es individual y sin material. b. La duración es 3hs. c. Sólo se contestan dudas acerca de la letra.	d. Escriba las hojas de un sólo lado y con letra clara. e. Numere cada hoja, indicando en la primera el total. f. Coloque su nro. de cédula y nombre en cada hoja.

Problema 1 (16 puntos)

Considere la siguiente representación del tipo *AB*, de árboles binarios de enteros:

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

Implemente un procedimiento llamado `internosPorNiveles` que dado un árbol de tipo *AB*, imprima sus elementos, de izquierda a derecha y por niveles de arriba hacia abajo. Los elementos se deben imprimir en un sólo renglón separados por un espacio y **sin incluir** los elementos de las hojas del árbol.

Se debe implementar el procedimiento de forma iterativa, accediendo directamente a la representación del árbol, de forma tal que tenga O(*n*) de tiempo de ejecución en el peor caso, siendo *n* la cantidad de nodos del árbol.

Use el TAD *Cola* de elementos de tipo *AB* para resolver el problema. Debe especificar pero no implementar el TAD *Cola* utilizado (se asume que el TAD *Cola* está implementado eficientemente, de modo que todas las operaciones básicas son O(1) de tiempo de ejecución en el peor caso). No use otras estructuras de datos o TADs auxiliares.

Solución Problema 1

```
// Crea y devuelve una Cola de ABs vacía
Cola crearC();

// Inserta a al final de la c
void encolarC(AB a, Cola &c);

// Devuelve true si c esta vacía y false en caso contrario
bool esVaciaC(Cola c);

// Devuelve el primer elemento de c. Pre: !esVaciaC(c)
AB primeroC(Tabla t);

// Borra el primer elemento de c. Pre: !esVaciaC(c)
void desencolarC(Cola &c);

// Libera la memoria de la cola
void destruirC(Cola &c);
```

```
void ImpNiveles (AB a){
Cola colaAux;
AB arbActual;
    crearC(colaAux);
    if(a != NULL)
        encolarC(a,colaAux);

    while (! esVaciaC(colaAux)) {
        arbActual = primeroC(colaAux);
        desencolarC(colaAux);
        if ((arbActual->izq != NULL) || (arbActual->der != NULL)){
            printf("%i ", arbActual->raiz);
            if (arbActual->izq != NULL)
                encolarC(arbActual->izq, colaAux);
            if (arbActual->der != NULL)
                encolarC(arbActual->der, colaAux);
        };
    };
destruirC(colaAux);
};
```

Problema 2 (22 puntos)

Se quiere especificar e implementar un TAD *Pedidos* que permita registrar una colección de pedidos y atender los pedidos respetando el orden de llegada. Un pedido tiene un identificador que es un número en el rango [0:K] y una descripción que es un *string*. Se pide:

a) Especifique completamente, en C* el TAD *Pedidos* con operaciones que permitan:
   - Crear una estructura de pedidos vacía, que pueda contener hasta *K+1* pedidos.
   - Insertar un pedido con identificador *i* ($0 \leq i \leq K$) y descripción *d*, en una colección de pedidos *p*, siempre que no exista otro pedido con el mismo identificador *i* en *p;* en caso contrario la operación no tendrá efecto.
   - Eliminar el pedido más antiguo (el primero ingresado) de una colección de pedidos no vacía y devolver la descripción de dicho pedido.
   - Consultar si una colección de pedidos es vacía.
   - Consultar si hay un pedido con identificador *i* ($0 \leq i \leq K$) en una colección de pedidos.

b) Defina una representación del TAD anterior de tal manera que las operaciones de inserción y eliminación de pedidos tengan O(1) de tiempo de ejecución en el peor caso.

c) Implemente las operaciones para crear e insertar de la parte a) accediendo directamente a la representación propuesta. Omita el código del resto de las operaciones del TAD (que puede asumir están implementadas).

Solución Problema 2

a) Especificación completa del TAD *Pedidos:*

```
// Crea y devuelve una Colección de pedidos vacía
Cola crearP();

// Devuelve true si p esta vacía y false en caso contrario
bool esVaciaP(Pedidos p);

// Inserta el pedido con identificador i y descripción desc
// en la colección de pedidos p
void insertarP(int i, string desc, Pedidos &p);

// Elimina el pedido más antiguo y devuelve su descripción.
// Pre: !esVaciaP(p)
string borrarP(Pedidos &p);

// Devuelve true si existe un pedido con identificador i en p
bool existePedidoP(int i, Pedidos p);
```

b) Representación del TAD *Pedidos:*

```
struct nodoP {
    int idP;
    string descP;
    nodoP * sig;
};
typedef nodoP * colaP;

struct cabezalP {
    bool * idesP;
    colaP inicioPed;
    colaP finPed;
};
typedef cabezalP * Pedidos;
```

c) Implemente las operaciones para crear e insertar de la parte a).

```
// Crea y devuelve una Colección de pedidos vacía
Cola crearP(){
Pedidos p;
    p = new (cabezalP);
    p->idesP = new bool[K+1];
    for (int i=0; i <= K; i++)
        p-> idesP[i] = false;
    p->inicioPed = NULL;
    p->finPed = NULL;
    return p;
};

// Inserta el pedido con identificador i y descripción desc
// en la colección de pedidos p
void insertarP(int i, string desc, Pedidos &p){
colaP nuevoP;
    if (!p->idesP[i]){
        p->idesP[i] = true;
        nuevoP = new nodoP;
        nuevoP-> idP = i;
        nuevoP-> descP = desc;
        nuevoP-> sig = NULL;

        if (p->inicioPed == NULL)
            p->inicioPed = nuevoP;
        else
            p->finPed->sig = nuevoP;
        p->finPed = nuevoP;
    };
};
```

Problema 3 (22 puntos)

Considere la siguiente especificación del TAD *Tabla* (funciones parciales) entre enteros (dominio) y *strings* (codominio):

```
// Crea y devuelve una Tabla vacía
Tabla crearT();

// Asocia a d el valor s en t; si ya tenía un valor asociado, lo redefine
void insertarT(int d, string s, Tabla &t);

// Devuelve true si t esta vacía y false en caso contrario
bool esVaciaT(Tabla t);

/* Devuelve true si el elemento d tiene asociado un elemento en t y false en caso
contrario */
bool estaDefinidaEnT(int d, Tabla t);

/* Devuelve el elemento del codominio asociado a d en t
Pre: estaDefinidaEnT(d,t) */
string recuperarT(int d, Tabla t);

/* Modifica t de modo que d no tenga un elemento asociado.
Si !estaDefinidaEnT(d,t), la operación no tiene efecto */
void borrarT(int d, Tabla &t);
```

Se pide:

a) Proponga una implementación del TAD *Tabla*, sin usar TADs auxiliares y asumiendo que:

   i) es igualmente probable que se desee asociar un valor a cualquier elemento del dominio y que

   ii) la cantidad de asociaciones a almacenar en la tabla puede estimarse en un valor *N*.

   Las operaciones selectoras y la operación de inserción deben tener O(1) de tiempo de ejecución en el caso promedio. Defina el tipo en C* para la representación elegida del TAD y accediendo directamente a la representación, desarrolle el código de las operaciones `crearT` y **`recuperarT`**. Omita el código del resto de las operaciones del TAD (que puede asumir están implementadas).

b) Usando las operaciones del TAD *Tabla*, definidas anteriormente, implemente la operación ***`buscar`*** que, dado un string *s* y una lista ordenada en forma ascendente de enteros *li*, devuelve el mayor de los enteros al cual le corresponde *s*:

```
/* Dados un string s, una lista ordenada de forma ascendente de enteros li y
una Tabla t, devuelve el mayor de los enteros que tiene imagen s en t.
Pre: existe al menos un elemento d en li tal que estaDefinidaEnT(d,t) y el
resultado de recuperarT(d,t)es el string s */
int buscar(string s, ListaEnt li, Tabla t);
```

Utilice la función `strEq` (que se asume implementada) para comparar strings; `strEq` retorna true si y sólo si dos strings son iguales. También debe utilizar el TAD *ListaEnt* con las siguientes operaciones básicas (asuma que están implementadas):

```
ListaEnt vaciaLEnt();                    // Crea y devuelve una lista vacía
ListaEnt insertarInicioLEnt(ListaEnt l, int x); // Inserta x al principio de l
int obtenerPrimeroLEnt(ListaEnt l);      // Devuelve el primer elem de l no vacía
ListaEnt restoLEnt(ListaEnt l);          // Devuelve el resto de l no vacía
bool EsVaciaLEnt(ListaEnt l);            // Devuelve true si l es vacía
```

Solución Problema 3

a) La implementación elegida para el TAD *Tabla* es una tabla de hash abierta, donde la función de hash es *h(i) = i mod N.*

```
struct nodoT {
    int dom;
    string codom;
    nodoT * sig;
};
typedef nodoT * ListaT;

struct cabezalT {
    ListaT * hashT;
};
typedef cabezalT * Tabla;
```

Implemente las operaciones para `crearT` y `recuperarT`.

```
// Crea y devuelve una Tabla vacía
Tabla crearT(){
Tabla t;
    t = new (cabezalT);
    t->hashT = new ListaT[N];
    for (int i=0; i < N; i++)
        t->hashT[i] = NULL;
    return t;
};

// Devuelve el elemento del codominio asociado a d en t
// Pre: estaDefinidaEnT(d,t)
string recuperarT(int d, Tabla t){
ListaT lst = t->hashT[d%N];

    while (lst->dom != d)    //sabemos que está!!
        lst = lst->sig;
    return lst->codom;
};
```

c) Operación ***buscar*** que, dado un string *s* y una lista ordenada en forma ascendente de enteros *li*, devuelve el mayor de los enteros al cual le corresponde *s*:

```
/* Dados un string s, una lista ordenada de forma ascendente de enteros li y
una Tabla t, devuelve el mayor de los enteros que tiene imagen s en t.
Pre: existe al menos un elemento d en li tal que estaDefinidaEnT(d,t) y el
resultado de recuperarT(d,t)es el string s */

int buscar(string s, ListaEnt li, Tabla t){
int candidato, resultado;
    while (!vaciaLEnt(li)){
        candidato = obtenerPrimeroLEnt(li);
        li = restoLEnt(li);
        if (estaDefinidaEnT(candidato,t) &&
            strEq(recuperarT(candidato,t),s))  //circuito corto
            resultado =  candidato;
    };
    return resultado;
};
```