Examen de Programación 2

Diciembre de 2023

Problema 1 (40 puntos: 20+20)

Se quiere representar una estructura de directorios (carpetas) de un sistema operativo, donde cada directorio tiene un nombre único que lo identifica y posee una lista de archivos. Cada directorio puede contener un número finito de subdirectorios y una lista de archivos, donde cada archivo tiene un nombre y un contenido (ambos de tipo *char* *).

Considere el tipo ***Directorios,*** definido como árboles generales de *Archivos* e implementado como árboles binarios con la semántica primer hijo (pH) – siguiente hermano (sH):

```
struct nodoArchivo {
    char * nombreArchivo;
    char * contenidoArchivo;
    nodoArchivo * sig;
}

typedef nodoArchivo * Archivos
```

```
struct nodoDirectorio {
    char * nombreDirectorio;
    Archivos listaArchivos;
    nodoDirectorio * pH, * sH;
}

typedef nodoDirectorio * Directorios
```

a) Defina una función recursiva ***ubicar*** que, dados un directorio *D* sin elementos repetidos y el nombre *nom_dir* de un directorio, retorna el puntero al nodo donde se encuentra *nom_dir* en *D* ó NULL si no se encuentra. Utilice la función `strEq` (que se asume implementada) para comparar strings (elementos de tipo *char* *); `strEq` retorna `true` si y sólo si dos strings son iguales.

Directorios ubicar (Directorios D, char * nom_dir)

b) Usando la función ***ubicar*** implemente el procedimiento ***borrarArchivos*** que, dados un directorio *D* sin elementos repetidos y el nombre *nom_dir* de un directorio, elimina todos los archivos de *nom_dir*, liberando completamente la memoria de éstos. El procedimiento no tendrá efecto si *nom_dir* no está en *D* ó si *nom_dir* no posee archivos (su lista de archivos es vacía).

void borrarArchivos (Directorios & D, char * nom_dir)

Problema 2 (30 puntos: 10+20)

Considere el TAD Cola no acotada de enteros (***ColaInt***) que contempla la política FIFO, con operaciones exclusivamente para: *(1) crear una cola vacía*; *(2) insertar*; *(3) eliminar y retornar el elemento correspondiente*; *(4)* chequear si una cola está vacía; *(5)* retornar una copia de una cola sin compartir memoria; y *(6)* destruir una cola, liberando su memoria.

a) **Especifique el TAD *ColaInt***, incluyendo pre y postcondiciones. NO se pide implementar *ColaInt*.

b) **Implemente** la función: ***bool iguales (ColaInt c1, ColaInt c2)***, que dadas dos colas de tipo *ColaInt* retorne true si y solo si *c1* tiene los mismos elementos que *c2*, en orden FIFO. La función no puede acceder a la representación del TAD ni modificar sus parámetros. Use solamente el TAD *ColaInt* para resolver este problema; no use otros TADs ni estructuras de datos auxiliares diferentes a *ColaInt*.

Examen de Programación 2

Diciembre de 2023

Problema 3 (30 puntos: 18+12)

Considere un TAD ***Multiset*** de cadenas de caracteres (de tipo ***char ****) que tiene (entre otras) las siguientes operaciones:

- ***void insertar (Multiset & m, char * cad)***, que agrega una ocurrencia de la cadena 'cad' al multiset 'm'.
- ***int ocurrencias (Mutiset m, char * cad)***, que retorna la cantidad de ocurrencias de 'cad' en 'm'.

Considere una implementación del TAD ***Multiset*** usando *hashing* abierto, con la siguiente representación:

```
struct nodoHash {
        char * cadena;
        int ocurrencias; // ocurrencias de la cadena
        nodoHash * sig;
}
```

```
struct representacionMultiset {
        nodoHash ** tabla; // tabla de hash
        unsigned int cota; // tamaño de la tabla de hash
        unsigned int cantidad; // cantidad total de elementos
}

typedef representacionMultiset * Multiset
```

Implemente las operaciones ***insertar*** y ***ocurrencias***, considerando la representación previa de un multiset con *hashing* abierto. Asuma la existencia de una función de hash: ***unsigned int h (char * cad)***, que ***crear*** recibe como parámetro la cantidad estimada de elementos diferentes del multiset, y que las cadenas de caracteres (de tipo *char **) se pueden manipular directamente con los operadores básicos: =, == y < (como si fueran enteros).

Examen de Programación 2

Diciembre de 2023

SOLUCIONES

PROBLEMA 1

```
Directorios ubicar (Directorios D, char * nom_dir){
        if (D == NULL)
                return NULL;
        else if (strEq(D->nombreDirectorio, nom_dir))
                return D;
        else {  Directorios estaPH = ubicar(D->pH, nom_dir);
                if (estaPH != NULL)
                        return estaPH;
                else return ubicar(D->sH, nom_dir);
        }
}

void borrarArchivos (Directorios & D, char * nom_dir){
        Directorios estaDIR = ubicar(D, nom_dir);
        if (estaDIR != NULL)
                borrarListaArchivos(estaDIR->listaArchivos);
}

void borrarListaArchivos (Archivos & A){
        if (A != NULL) {
                borrarListaArchivos(A->sig);
                delete [] A->nombreArchivo;
                delete [] A->contenidoArchivo;
                delete A;
                A = NULL;
        }
}
```

PROBLEMA 2

a)

```
// PRE: -
ColaInt crear()
// POS: retorna una nueva cola vacía

// PRE: -
void encolar (ColaInt &c, int x)
// POS: inserta un entero x en una cola c
```

Examen de Programación 2

Diciembre de 2023

```
// PRE: cola no vacía
int eliminar (ColaInt &c)
// POS: elimina el primer entero ingresado en la cola c y lo retorna

// PRE: -
bool esVacia (ColaInt c)
// POS: retorna true si y solo si la cola c está vacía

// PRE: -
// POS: retorna una copia de la cola c sin compartir memoria
ColaInt copia (ColaInt c)

// PRE: -
void destruir (ColaInt &c);
// POS: destruye la cola c, liberando su memoria
```

b)

```
bool iguales (ColaInt c1, ColaInt c2){
   ColaInt clon_c1 = copia(c1);
   ColaInt clon_c2 = copia(c2);
   bool res = true;
   while (!esVacia(clon_c1) && !esVacia(clon_c2) && res){
       res = (eliminar(clon_c1) == eliminar(clon_c2));
   }
   res = res && esVacia(clon_c1) && esVacia(clon_c2);
   destruir(clon_c1);
   destruir(clon_c2);
   return res;
}
```

Examen de Programación 2

Diciembre de 2023

PROBLEMA 3

```
void insertar (Multiset & m, char * cad){
        int posicion = h(cad)%(t->cota);
        nodoHash* lista = t->tabla[posicion];
        while (lista!=NULL && lista->cadena!=cad)
                lista = lista->sig;
        if (lista==NULL){
                nodoHash* nuevo = new nodoHash;
                nuevo->cadena = cad;
                nuevo->ocurrencias = 1;
                nuevo->sig = t->tabla[posicion];
                t->tabla[posicion] = nuevo;
        }
        else lista->ocurrencias++;
        t->cantidad++;
}

int ocurrencias (Multiset m, char * cad){
        int posicion = h(cad)%(t->cota);
        nodoHash* lista = t->tabla[posicion];
        while (lista!=NULL && lista->cadena!=cad)
                lista = lista->sig;
        if (lista==NULL) return 0;
        else return lista->ocurrencias;
}
```