Examen de Programación 2

Julio de 2024

Problema 1 (25 puntos: 20+5)

Considere las siguientes definiciones de listas de enteros (*Lista*) y de pares de enteros (*ListaPar)*:

```
typedef nodoLista * Lista
struct nodoLista{ int dato; Lista sig; }

typedef nodoListaPar * ListaPar
struct nodoListaPar{ int dato; int cantidad; ListaPar sig; }
```

a) Implemente una **función iterativa** ***ordenar*** que dada una lista *l* de tipo *Lista* que puede contener valores exclusivamente en el rango [0 : *m*] (entre 0 y *m* inclusive, con $m > 0$), retorne una nueva lista de tipo *ListaPar* ordenada de mayor a menor según los datos de *l*, que contenga para cada elemento de *l* la cantidad de sus repeticiones. Si *l* es vacía (NULL), el resultado debe ser NULL. Se pueden usar estructuras de datos auxiliares, manejando adecuadamente la memoria (pedido y liberación, si corresponde). **La función *ordenar* debe ser O(max(*n*,*m*)) en el peor caso**, siendo *n* el largo de *l*.

```
PRE: Cada elemento x de la lista l cumple: 0 ≤  x ≤ m, con m > 0
ListaPar ordenar (Lista l, int m)
```

Por ejemplo, si *l* es [2, 4, 83, 4, 4, 99, 2, 83, 5] y *m* es 100, el resultado debe ser [(99,1), (83,2), (5,1), (4,3), (2,2)].

b) Justifique muy brevemente el cumplimento del orden exigido en la parte a) para su implementación de ***ordenar***.

Problema 2 (30 puntos)

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 sub-directorios 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; // subdirectorios
   nodoDirectorio * sH;
}
typedef nodoDirectorio * Directorios
```

Implemente el **procedimiento recursivo** ***eliminar*** que, dada una estructura de directorios *D* (de tipo *Directorios*) que no puede tener nombres de directorios repetidos y el nombre *nom_dir* de un directorio, elimina el directorio *nom_dir* de D y todos sus archivos, liberando la memoria correspondiente. El **procedimiento no tendrá efecto si** *nom_dir* no está en *D* ó si *nom_dir* posee algún subdirectorio. Se sugiere implementar un procedimientos auxiliar recursivo para eliminar la lista de archivos de un directorio dado.

```
void eliminar (Directorios & D, char * nom_dir)
```

Problema 3 (45 puntos: 6+24+15)

a) Especifique, con pre y postcondiciones, un TAD ***cola de prioridad*** no acotado de elementos de tipo *char* * con prioridades que toman valores enteros y con las siguientes operaciones:

1) *crear*, que genera una cola de prioridad vacía.
2) *encolar,* que inserte un string en una cola de prioridad asociándole cierta prioridad dada.
3) *obtener*, que retorna el elemento de tipo *char* * con mayor prioridad (mayor valor entero) de una cola de prioridad no vacía. En el caso en que exista más de un elemento en la cola con la misma prioridad máxima, deberá retornarse el primero de ellos en ingresar a la cola.
4) *desencolar*, que elimina el elemento con mayor prioridad de una cola de prioridad no vacía. En el caso en que exista más de un elemento en la cola con la misma prioridad máxima, deberá eliminarse el primero de ellos en ingresar a la cola.
5) *cantidad,* que retorna la cantidad de elementos de una cola de prioridad.
6) *destruir,* que destrute la cola de prioridad, liberando toda su memoria

b) Desarrolle una **implementación** del TAD anterior en la que las operaciones 1, 2, 3 y 5 tengan O(1) de tiempo de ejecución en el peor caso. Escriba el código de las operaciones 2 y 3, y asuma que las restantes están implementadas. Pueden utilizarse funciones de cadenas de caracteres como *strlen*, *strcpy* u otras.

c) Implementar la operación ***imprimir*** que, dada una lista de elementos de tipo char * (cadenas), imprima las cadenas ordenadas por su largo, de mayor a menor. El orden entre cadenas de igual largo no es relevante. Implemente *imprimir* usando una *cola de prioridad* según la especificación de la parte **a)**. Asuma que elementos de tipo char * (cadenas) se pueden imprimir directamente con *printf* o *cout*. Pueden utilizarse otras funciones de cadenas de caracteres como *strlen*, *strcpy* u otras.

void imprimir (LCadenas l) donde:

```
typedef nodoLCadenas * LCadenas
struct nodoLCadenas { char * cadena; LCadenas sig; }
```

SOLUCIONES

1-a)

```
PRE: Cada elemento x de la lista l cumple: 0 ≤  x ≤ m, con m > 0
Lista ordenar(Lista l, int m){
      ListaPar ret = NULL;
      int * elementos = new int[m+1];
      for (int i=0; i<=m; i++)
            elementos[i] = 0;
      while (l!=NULL){
            elementos[l->dato]++;
            l = l->sig;
      }
      for (int i=0; i<=m; i++){
            if(elementos[i]>0)
                  insComienzo(i, elementos[i], ret); //inserción al inicio de ret
      }
      delete [] elementos;
      return ret;
}

// inserta un par (e,cant) al comienzo de una lista de pares l
void insComienzo(int e, int cant, ListaPar & l){
      ListaPar nuevo = new nodoListaPar;
      nuevo -> dato = e;
      nuevo -> cantidad = cant;
      nuevo -> sig = l;
      l = nuevo;
}
```

1-b)

Hay tres iteraciones en secuencia de: O(m), O(n) y O(m). El resto de las acciones son O(1), incluida insComienzo. Luego, el orden del peor caso es O(max(n,m)), por la regla de la suma de órdenes.

2)

```
void eliminar (Directorios & D, char * nom_dir){
       if (D != NULL){
              if (!strcmp(D->nombreDirectorio, nom_dir)){
                     eliminar(D->pH, nom_dir);
                     eliminar(D->sH, nom_dir);
              }
              else if (D->pH == NULL){ // Es nom_dir y no tiene subdirectorios
                     Directorios aBorrar = D;
                     D = D->sH;
                     borrarListaArchivos(aBorrar->listaArchivos);
                     delete [] D->nombreDirectorio;
                     delete aBorrar;
              }
       }
}
```

```
// Elimina los archivos de A, liberando la memoria asociada y dejando A en NULL
void borrarListaArchivos (Archivos & A){
	if (A != NULL) {
		borrarListaArchivos(A->sig);
		delete [] A->nombreArchivo;
		delete [] A->contenidoArchivo;
		delete A;
		A = NULL;
	}
}
```

3-a)

```
// PRE: -
// POS: retorna una nueva cola de prioridad vacía
ColaPrioridad crearColaPrioridad ();

// PRE: -
/* POS: inserta un string s con prioridad p a la cola de prioridad. Los elementos
con igual prioridad se agregan en orden FIFO */
void encolar(ColaPrioridad &cp, char* s, int p);

// PRE: cantidad(cp)!=0
/* POS: retorna el elemento con mayor prioridad de la cola de prioridad. Ante
elementos de igual prioridad retorna el más antiguo */
char* obtener(ColaPrioridad cp);

// PRE: cantidad(cp)!=0
/* POS: elimina el elemento con mayor prioridad de la cola de prioridad. Ante
elementos de igual prioridad elimina el más antiguo */
void desencolar(ColaPrioridad &cp);

// PRE: -
// POS: retorna la cantidad de elementos presentes en la cola de prioridad
int cantidad(ColaPrioridad cp);

// PRE: -
/* POS: elimina todos los elementos presentes en la cola de prioridad y libera la
memoria */
void destruir(ColaPrioridad &cp);
```

3-b)

```
struct nodoCola { char* dato; int prioridad; nodoCola* sig; }
typedef nodoCola* nodoCola;
struct repColaPrioridad { nodoCola* ppio; nodoCola* mayor; int largo; }
typedef repColaPrioridad* ColaPrioridad;

/* Para cumplir los requisitos de Orden se mantienen: un puntero al inicio de una
lista simplemente enlazada donde se hace la inserción; un puntero al nodo con el
dato de mayor prioridad que se actualiza al insertar (en tiempo constante) y al
eliminar (buscando la mayor prioridad más antigua en la lista); y un entero con la
cantidad de elementos presentes en la lista */
```

```
void encolar(ColaPrioridad &cp, char* s, int p){
  nodoCola nuevo = new nodoCola;
  int largo = strlen(s)+1;
  nuevo->dato = new char[largo];
  strcpy_s(nuevo->dato, largo, s);
  nuevo->prioridad = p;
  if (cantidad(cp)==0 || p > cp->mayor->prioridad){
     cp->mayor = nuevo;
  }
  nuevo->sig = cp->ppio;
  cp->ppio = nuevo;
  cp->largo++;
}

char* obtener(ColaPrioridad cp){
  assert(cantidad(cp)!=0);
  return cp->mayor->dato;
}
```

3-c)

```
void imprimir (LCadenas l){
  char * cad;
  ColaPrioridad cp = crearColaPrioridad();
  while (l!=NULL){
     encolar(cp, lista->cadena, strlen(lista->cadena));
     l = l->sig;
  }
  while (cantidad(cp)!=0){
     cout << obtener(cp);
     desencolar(cp);
  }
  destruir(cp);
}
```