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

Primer Parcial de Programación 2

Setiembre de 2024

*En la primera hoja poner el **número de parcial**, nombre completo, la cédula y la cantidad de hojas que entrega. En las siguientes hojas indicar el nombre, la cédula y el número de hoja. Escribir de manera clara y legible (con lapiz o lapicera). Empezar cada problema en hoja nueva y no escribir las hojas de ambos lados.*

Problema 1 (15 puntos: 13+2)

a) Se desea trabajar con listas ordenadas de identificadores de productos de un negocio. Considere la siguiente definición de listas de enteros (identificadores de productos):

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

Implemente una función iterativa `Lista comunes(Lista l1, Lista l2)` que dadas dos listas de enteros ordenadas de menor a mayor y sin elementos repetidos, construya y retorne una nueva lista ordenada de menor a mayor que contenga solamente a los elementos (identificadores de productos) comunes; es decir, que están presentes en ambas listas. Si no hay elementos comunes, el resultado debe ser la lista vacía, NULL.
La lista resultado no deberá tener elementos repetidos ni compartir memoria con las listas parámetro, que no pueden modificarse. La función debe tener O(n+m) peor caso, siendo n y m los largos de las listas parámetro. No implemente funciones o procedimientos auxiliares, ni use arreglos/vectores.

Por ejemplo, si las listas son [1,**3**,**4**,8,**9**] y [2,**3**,**4**,6,**9**,10,17], el resultado debería ser: [**3**,**4**,**9**].

b) Justifique informal y brevemente el orden de tiempo de ejecución requerido para el peor caso de **`comunes`**.

Problema 2 (22 puntos: 10+1+10+1)

Para trabajar con salarios de empleados en una empresa, considere la siguiente definición del tipo **`ABB`** de árboles binarios de búsqueda, que contienen códigos de empleados (de tipo **`int`**) y sus salarios (de tipo **`float`**):

```
struct nodoABB {
      int codigo;       // código del empleado
      float salario;    // salario del empleado
      nodoABB * izq, * der;
};
typedef nodoABB * ABB;
```

La información de los empleados en la empresa está modelada con un árbol de tipo **`ABB`**, organizado (ordenado) por códigos de empleados, que NO pueden repetirse. Los salarios de distintos empleados de la empresa si podrían repetirse. Tanto los códigos de los empleados como los salarios son valores no negativos. Los empleados con salario cero (0) se asume que están con licencia sin goce de sueldo.

a) Implemente un procedimiento recursivo `void licenciar(ABB & t, int c)` que, dado un árbol ***t*** de tipo **`ABB`** y un código (entero) ***c***, asigna cero (0) al salario de los empleados de ***t*** que tienen código menor estricto que ***c***; es decir, los pone en licencia sin goce de sueldo. Los empleados en ***t*** con código mayor o igual que ***c*** no modifican su salario. Si el árbol es vacío (NULL), el procedimiento no tendrá efecto. El orden de tiempo de ejecución de **`licenciar`** debe ser O(n) peor caso, siendo n la cantidad de nodos de ***t***, aunque el procedimiento deberá evitar recorrer nodos de ***t*** que no sean necesarios. No implemente operaciones auxiliares.

b) Explique muy brevemente cuál sería el peor caso de **`licenciar`**.

c) Implemente una función recursiva `ABB conAumento(ABB t, float piso, float aumento)` que, dado un árbol ***t*** de tipo **`ABB`** y dos elementos de tipo float, ***piso*** y ***aumento***, retorna un nuevo árbol (que no comparte memoria con ***t***) que tiene los mismos empleados que están en t pero con salarios actualizados: en el nuevo árbol, el salario de un empleado tendrá el mismo sueldo que en ***t,*** si éste es mayor que ***piso***; si el salario de un empleado en ***t*** es menor o igual que ***piso***, en el árbol resultado su salario se incrementará, sumando el valor ***aumento***. Si ***t*** es vacío (NULL), el resultado de la función debe ser NULL. El orden de tiempo de ejecución de **`conAumento`** debe ser O(n) peor caso, siendo n la cantidad de nodos de ***t***.

d) ¿El caso promedio de **`conAumento`** es mejor que su peor caso? **Explique** muy brevemente.

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

Primer Parcial de Programación 2

Setiembre de 2024

SOLUCIONES

Problema 1

a)

```
Lista comunes(Lista l1, Lista l2){
    Lista res = NULL;
    Lista fin_res, nodo;
    while (l1!=NULL && l2!=NULL){
        if (l1->dato == l2->dato){
            nodo = new nodoLista;
            nodo->dato = l1->dato;
            nodo->sig = NULL;
            if (res == NULL) res = nodo;          // la primera inserción de comunes en res
            else fin_res->sig = nodo;             // se inserta al final de res si no es vacía
            fin_res = nodo;                       // se actualiza siempre el final de res
            l1 = l1->sig;
            l2 = l2->sig;
        } else if  (l1->dato < l2->dato) l1 = l1->sig;   // l1->dato no está en los comunes
        else  l2 = l2->sig;                              // l2->dato no está en los comunes
    }
    return res;
}
```

b) La función **comunes** es O(*n*+*m*) en el peor caso, siendo *n* y *m* el lago de cada lista parámetro, ya que recorre ambas listas una única vez, haciendo operaciones de O(1).

Problema 2

a)

```
void licenciar(ABB & t, int c){
    if (t != NULL){
        if (t->codigo < c){
            t->salario = 0;          // pone al empleado en licencia (con salario cero)
            licenciar(t->der, c);    // puede haber códigos menores que c en t->der
        }
        licenciar(t->izq, c);        // siempre recorre t->izq por códigos menores que c
    }
}
```

b) El peor caso de ***licenciar*** se da cuando todos los códigos son menores que ***c***. En dicho caso debe recorrerse todo el árbol.

c)

```
ABB conAumento(ABB t, float piso, float aumento){
    if (t == NULL) return NULL;
    else {   ABB res = new nodoABB;
        res->codigo = t->codigo;
        if (t->salario > piso) res->salario = t->salario;   // con el mismo salario
        else res->salario = t->salario + aumento;           // aumenta el salario
        res->izq = conAumento(t->izq, piso, aumento);
        res->der = conAumento(t->der, piso, aumento);
        reurn res;
    }
}
```

Nota: Si alguien consideró que a los empleados con sueldo 0 (en licencia sin goce de sueldo) no les correspondía el aumento, se considerará bien también, aunque esto no se requería en la parte *c*).

d) El caso promedio de ***conAumento*** es igual al peor caso, ya que recorre siempre todo el árbol.