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

Primer Parcial de Programación 2

Mayo de 2025

Problema 1 (18 puntos: 2+14+2)

Considere la siguiente definición del tipo ***Lista*** para **listas de enteros**, en memoria dinámica:

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

Se quiere implementar la función iterativa ***combinar*** que dados dos **arreglos *a1*** y ***a2*** de enteros de tamaños ***n*** **y** ***m*** respectivamente *(con* ***n****>0 y* ***m****>0)*, ordenados de menor a mayor, retorne una lista de tipo *Lista* que contenga los elementos presentes en ambos arreglos, ordenados menor a mayor. Los arreglos parámetros pueden contener elementos repetidos, pero la lista resultado no debe tener repeticiones (cada elemento aparece a lo sumo una vez). La función ***combinar*** debe tener O(***n***+***m***) en el peor caso, evitando recorridas innecesarias.

Ejemplo: ***a1*** = [2,4,4,4,6,7,9], ***a2*** = [1,2,4,8,13,22], ***n*** = 7, ***m*** = 6 → lista resultado = < 1,2,4,6,7,8,9,13,22 >

Concretamente, para resolver este problema:

a) Implemente el procedimiento ***insertar*** que agrega un elemento dado ***x*** al comienzo de una lista ***l***, excepto que ***x*** sea ya sea el primer elemento de ***l***, en cuyo caso el procedimiento no tendrá efecto:

```
void insertar (Lista & l, int x)
```

b) Implemente ahora iterativamente (sin usar recursión) la función ***combinar***, usando la operación ***insertar*** de la parte **a)** para generar la lista resultado:

```
PRE: a1 y a2 están ordenados de menor a mayor, n > 0 y m > 0
Lista combinar (int* a1, int* a2, int n, int m)
```

c) Justifique brevemente el orden de tiempo de ejecución de la implementación de ***combinar***.

Problema 2 (19 puntos: 17+2)

Considere la siguiente definición del tipo ***AB*** de los ***Árboles Binarios*** **de enteros**, en memoria dinámica:

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

a) Implemente una función recursiva ***copiarSubArbol*** que dado un **árbol binario de enteros** ***t*** de tipo *AB* sin elementos repetidos y dado un entero ***x***, retorne una copia del subárbol de ***t*** que tenga a ***x*** como raíz, sin compartir memoria con ***t***. Si ***x*** no está en ***t*** el resultado debe ser el árbol vacío (NULL). Si ***x*** es la raíz de ***t***, debe retornar una copia completa de ***t***. La función ***copiarSubArbol*** debe tener O(***n***) peor caso, siendo ***n*** la cantidad de nodos del árbol. Se sugiere implementar una función auxiliar, recursiva.

```
Pre: t no tiene elementos repetidos
AB copiarSubArbol (AB t, int x)
```

b) Justifique brevemente el orden de tiempo de ejecución para el peor caso de ***copiarSubArbol***.

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

Primer Parcial de Programación 2

Mayo de 2025

SOLUCIONES

Problema 1

1-a)

Pos: Inserta x al comienzo de l, si x no es su primer elemento

```
void insertar (Lista & l, int x){
     if (l==NULL) || l->dato!x){
                Lista nuevo  = new nodoLista;
                nuevo->dato = x;
                nuevo->sig = l;
                l = nuevo;
     }
}
```

1-b)

```
Lista combinar (int* a1, int* a2, int n, int m){

/* Se recorren los arreglos al revés y se agrega al comienzo de la lista resultado
el mayor cada vez (con insertar). De esta manera se logra generar la lista resul-
tado, sin repetidos, de menor mayor con los elementos de a1 y a2 */
     int pos_a1 = n-1;
     int pos_a2 = m-1;
     Lista res = NULL; // se inicializa la lista resultado a generar
     while (pos_a1>=0 && pos_a2>=0){ // se recorren los arreglos al revés
                if (a1[pos_a1]>a2[pos_a2]){
                      insertar(res, a1[pos_a1]);
                      pos_a1--;
                }
                else{ insertar(res, a2[pos_a2]);
                      pos_a2--;
                }
     }
     for (int i=pos_a1; i>=0; i--)  // se insertan los elementos que queden de a1
                insertar(res, a1[i]);
     for (int i= pos_a2; i>=0; i--) // se insertan los elementos que queden de a2
                insertar(res, a2[i]);
     return res;
}
```

1-c)

*La función **combinar es O(n+m)** en el peor caso, siendo n y m el lago de cada arreglo parámetro, ya que recorre ambos arreglos una única vez, haciendo operaciones de O(1). Notar que en particular insertar es O(1) peor caso.*

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

Primer Parcial de Programación 2

Mayo de 2025

Problema 2

2-a)

```
AB copiarSubArbol (AB t, int x){
      if (t != NULL){
            if (x == t->dato)
                  return copiarArbol(t); //función auxilar que copia todo un árbol
            else{ AB t_izq = copiarSubArbol(t->izq, x);
                  if (t_izq != NULL) return t_izq; //x está en t->izq (t_izq)
                  else return copiarSubArbol(t->der, x);
                                                //x puede o no estar en t->der
            }
      }
      else return NULL;
}

// Retorna una copia del árbol parámetro, sin compartir memoria
AB copiarArbol (AB t){
      if (t == NULL)
            return NULL;
      else{ AB res = new nodoAB;
            res->dato = t->dato;
            res->izq = copiarArbol(t->izq);
            res->der = copiarArbol(t->der);
            return res;
      }
}
```

2-b)

copiarSubArbol es O(n) peor caso, siendo n la cantidad de elementos/nodos del árbol. La función recorre siempre todo el árbol haciendo para cada nodo acciones de O(1) peor caso. Notar que cada nodo del árbol se visita a lo sumo una vez, ya sea en la función principal o en la auxiliar.