Facultad Politécnica - Departamento de Informática – Carrera: Ingeniería Informática
Algoritmos y Estructura de Datos III - Primer Examen Parcial – 2024/1er Periodo. Sección TQ/TR

Prof. Cristian Cappo/ Prof. Diego Ihara Viernes, *26/04/2024* Puntos: 58 + 2 (orden, claridad, pulcritud) = **60**

EXAMEN SIN MATERIAL. APAGUE SU CELULAR. La hoja de examen se devuelve. La interpretación de los temas forma parte de la evaluación. TENGA A MANO HOJAS DE EXAMEN, CALCULADORA, LAPIZ, BORRADOR Y BOLÍGRAFO. Duración 150 minutos.

Nombre y Apellido: _____ Nro. CIC:_____ Sección:___

Tema 1 (15p)

Por cada expresión indique si es *Verdadero* o *Falso,* fundamentando la respuesta.

a) El siguiente código imprime *Hola* alrededor de $2n^2$ veces

```
for ( i = n*n; i > 1; i = i/2 )
    for ( j=0; j < i; j++ )
        System.out.println("Hola");
```

Verdadero.

Asumimos n potencia de 2

Si m = n^2

m + m/2 + m/4 + m/8 .. = 2m

esto es 2m => 2 n^2 por suma geométrica

b) Considere una matriz bidimensional de *n* x *n* elementos enteros en donde cada entero en cada fila se encuentran en orden ascendente y que el primer elemento de cada fila es mayor al último de la fila previa. Dado un entero ***x***, ¿Es posible verificar en tiempo *O(log n)* si ***x*** se encuentra en la matriz?

Verdadero. Si es posible.

Convertimos en un arreglo lineal de n^2 de longitud y aplicamos búsqueda binaria. Esto da log(n^2) = O(log n). Obviamente no se debe pasar todo a un arreglo (esto no respetaría la cota) sino debe ajustar la búsqueda mapeando los elementos a la matriz.

c) $2^{2n} \in O\left(2^n\right)$

Falso.

2^2n =2^n^n >c2^n para cualquier n > log c. Entonces 2^2n no es menor o igual a c2^n para cualquier c > 0 y una n lo suficientemente grande. Es aplicación de la definición de O.

d) En el siguiente código java la función *t()* es llamada cerca de $n \log n$ veces (asuma que *n* es potencia de 2).

```
public static void func( int n ){
    for ( int i=1; i < n; i=i*2 ) func(i);
    t();
}
```

Falso.
Es exactamente O(n), sumatoria de potencias de 2 desde 2^0 hasta 2^ (log(n) - 1) + 1

e) $\sum_{i=1}^{n-1} (n - i)i \in \theta\left(n^3\right)$

Verdadero.
Desarrollamos la sumatoria dejando en términos de sumatorias conocidas.

$\sum_{i=1}^{n-1} ni - \sum_{i=1}^{n-1} i^2$

Criterio de corrección por cada ítem: Indica correctamente si fue Verdadero o Falso (1p). Fundamenta correctamente (2p)

Tema 2 (15p)

Diseñe una estructura de datos en Java denominada CentroCola. La ED CentroCola es una cola que soporta las operaciones de agregar o remover un elemento al frente o al final de la cola, además de remover (retornando) el elemento central. Si existe un número par de elementos en la cola entonces el elemento central es el más cercano al frente. La especificación y el funcionamiento se muestra abajo:

```
Especificación de la Estructura de dato
public class CentroCola<..> // Definición
-------------------------------------------
CentroCola()              // Crea la ED
void agregarFrente(E x)   // Agrega x al frente
void agregarFinal(E x)    // Agrega x al final

E removerFrente()         // Borra el frente y
retorna
E removerFinal()          // Borra el final y retorna
E removerCentral()        // Borra el elemento
central
                             y lo retorna
int tamanho()             // retorna la cant. de
                          // elementos en la cola
```

```
// Ejemplo de uso
CentroCola<String> c = new CentroCola<String>(); //
[]
c.agregarFinal("A");      // [A]
c.agregarFinal("B");      // [A B]
c.agregarFinal("C");      // [A B C]
c.agregarFinal("D");      // [A B C D]
c.agregarFinal("E");      // [A B C D E]
c.removerCentral();       // [A B D E]   -> C
c.agregarFrente("F");     // [F A B D]
c.removerCentral();       // [F A D E]   -> B
c.removerCentral();       // [F D E]     -> A
c.removerFrente();        // [D E]       -> F
c.removerCentral();       // [E]         -> D
```

Todas las operaciones deben hacerse en tiempo constante O(1). La cantidad de memoria utilizada debe estar en O(n), donde n es la cantidad de elementos en la cola. Puede utilizar cualquiera de las estructuras de datos vistas en clase (listas, pilas, colas, BST, AVL o tablas de dispersión) y asumir que ya están implementadas en su forma estándar. **No utilizar ninguna colección del API de Java (esto invalidará la solución). Justificar los tiempos de su implementación.**

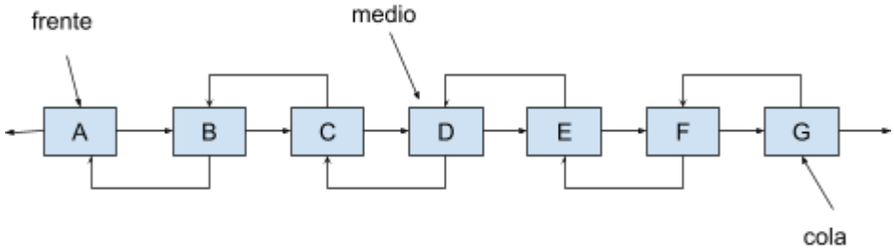
Criterio de corrección:

	Implementación correcta	Funciona en el tiempo solicitado	Justificación de tiempo
Estructura de datos	3p	-	-
agregar# (2)	1p	0.5p	0.5p
remover# (3)	1p	0.5p	0.5p
tamanho	1p	0.5p	0.5p

Es una aplicación directa de la implementación de una cola pero se incluye un puntero adicional. Al frente, al final y un puntero en el medio.

Entonces la estructura básica sería:

```
public class CentroCola<T> {
	private Nodo frente, cola, medio;
	private int n; // mantiene la cantidad de elementos

	private class Nodo {
		private T dato;
		private Nodo sgte;
		private Nodo prev;
	}
	…
}
```



AgregarFrente: Agregar un nuevo nodo en el frente como un Cola normal. Avanza el puntero medio al nodo previo si n es impar. Incrementar n.
AgregarFinal: Agregar un nuevo nodo al final como un Cola normal. Avanza el puntero medio al nodo siguiente si n es par. Incrementar n.
RemoverCentral: Actualizar medio al nodo sgte si n es par o al nodo prev si n es impar. Decrementar n.

Otra opción es mantener dos colas, una con los primeros n/2 elementos y la otra con los siguientes n/2 elementos. El medio será siempre el último elemento de la primera cola.

Aqui una implementación completa;

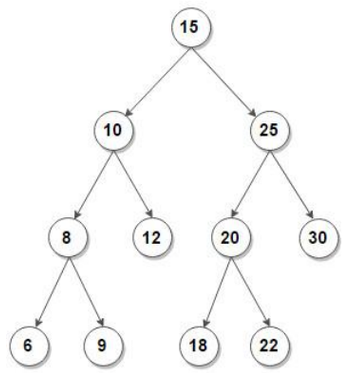
```
public class CentroCola<T> {
   private Nodo frente, cola, medio;
   private int n;

   private class Nodo {
      private T dato;
        private Nodo sgte;
        private Nodo prev;

   }

   public CentroCola() {
      this.n = 0;
        frente = null;
        cola   = null;
        medio  = null;
   }

   public void agregarFrente( T d ) {
        Nodo nvo = new Nodo();
        nvo.dato = d;
        if ( frente == null ) {
         this.frente = nvo;
         this.cola   = nvo;
         this.medio  = nvo;
           this.n = 1;
       }
       else {
         Nodo tmp    = frente;
         frente      = nvo;
         frente.sgte = tmp;
         tmp.prev    = frente;
         frente.prev = null;
         if ( n % 2 != 0 )
            medio = medio.prev;
         this.n += 1;
       }
   }

   public void agregarFinal( T d ) {
       Nodo nvo = new Nodo();
       nvo.dato = d;
       if ( cola == null ) {
         this.frente = nvo;
         this.cola   = nvo;
         this.medio  = nvo;
         this.n = 1;
       }
       else {
         Nodo tmp    = cola;
         cola        = nvo;
         cola.prev   = tmp;
         tmp.sgte    = cola;
         cola.sgte = null;
         if ( n % 2 == 0 )
            medio = medio.sgte;
             this.n += 1;
        }
   }

   public T removerCentral() {
```

```
        T dato = medio.dato;
        Nodo tmp_prev = medio.prev;
        Nodo tmp_sgte = medio.sgte;

        if ( n % 2 == 0 )
           medio = medio.sgte;
        else
           medio = medio.prev;

        tmp_sgte.prev = tmp_prev;
        tmp_prev.sgte = tmp_sgte;

        this.n -= 1;

        return dato;
   }

   public static void main (String [] args ) {
        CentroCola<String> cc = new CentroCola<String>();
        cc.agregarFinal("A");
        cc.agregarFinal("B");
        cc.agregarFinal("C");
        cc.agregarFinal("D");
        cc.agregarFinal("E");
        cc.agregarFinal("F");
        cc.agregarFinal("G");
        cc.agregarFrente("H");
        cc.agregarFrente("I");
        cc.agregarFinal("J");
        cc.agregarFinal("K");

        System.out.println("Frente: "+ cc.frente.dato);
        System.out.println("Medio: " + cc.medio.dato);
        System.out.println("Cola : " + cc.cola.dato);
        System.out.println("cnt elementos " + cc.n);

        CentroCola.Nodo tmp = cc.frente;
        while ( tmp != null ) {
         System.out.println(tmp.dato + ",");
         tmp = tmp.sgte;
        }

        System.out.println("Central removido " + cc.removerCentral());
        System.out.println("Medio: " + cc.medio.dato);

        tmp = cc.frente;
        while ( tmp != null ) {
          System.out.println(tmp.dato + ",");
          tmp = tmp.sgte;
        }
   }
}
```

Tema 3 (10p)

Dado un BST y dos nodos *x* e *y* posiblemente presentes en el BST, escriba un algoritmo (Java o Pseudocódigo) que encuentre el Ancestro Común más Bajo (ACB) de los nodos *x* e *y*. La solución deber retornar null si *x* o *y* no están en el árbol. El ACB de dos nodos *x* e *y* en un BST es el nodo más bajo (o bien más profundo) que tiene a *x* e *y* como descendientes, donde cada nodo puede ser descendiente de sí mismo (por ejemplo, si *x* es alcanzable desde *w*, entonces *w* es el ACB). En otras palabras, el ACB de *x* e *y* es el ancestro compartido de *x* e *y* que se ubica más lejos de la raíz. En el BST de la figura ejemplo el ACB(9,10) = 10, el ACB(22,30)=25 y el ACB(22, 5)=null.

El algoritmo propuesto debe ser menor a O(n^2) en términos de tiempo.



Indique la T(n) de su algoritmo en el peor caso y su coste asintótico temporal en término de O grande. Fundamente.

Criterio de corrección

Algoritmo	Implementación correcta sin respetar tiempo (1p)	Implementación correcta respetando el tiempo (4p)
T(n)	Cálculo correcto sin justificación (1p)	Cálculo correcto con justificación (3p)
Costo asintótico	Cálculo correcto sin justificación (1p)	Cálculo correcto con justificación (3p)

Una idea sencilla es realizar dos búsquedas para el primer elemento y para el segundo, volcar el camino de la búsqueda en dos vectores (un para cada valor buscado). Si ambas búsquedas fueron exitosas entonces comenzar a comparar los arreglos desde la posición 0 (que debería ser la raíz), continuar hasta que los valores de las posiciones sean diferentes, en ese caso el ACB sería el valor anterior igual. La carga de cada vector con el camino de búsqueda es 2n (a lo sumo). La comparación en n a lo sumo, así el costo total de este algoritmo está en O(n)

Tema 4 (10p)

Dado dos arreglos enteros, `a[]` y `b[]`, la diferencia simétrica entre `a[]` y `b[]` es el conjunto de elementos que aparece en exactamente solo uno de los arreglos. Diseñe y escriba un algoritmo en Java que reciba dos arreglos ordenados (`a` y `b`), cada uno de *n* elementos distintos, y retorne el tamaño de la diferencia simétrica entre ambos.

Por ejemplo si `a = [3, 6, 10, 14, 17, 18]` y `b = [2, 3, 7, 12, 14, 15]`, donde `n = 6`. Entonces el conjunto de la diferencia simétrica es `[2,6,7,10,12, 15,17,18]` y el algoritmo debe retornar `8`.

La solución debe tener un rendimiento temporal en *O(n)* y un rendimiento espacial en *O(1)*. El análisis de rendimiento en el peor caso. Los arreglos `a` y `b` no deben ser modificados.

Fundamente el tiempo y espacio de su solución

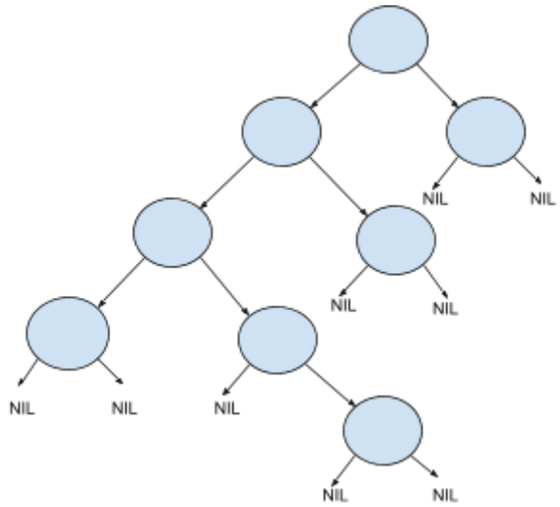
Criterio de corrección:

Algoritmo	Implementación correcta en tiempo y espacio requerido (8p)
Fundamentación	Correcta de tiempo y espacio (2p)

Se aplica básicamente la misma técnica de la operación de Merge (de MergeSort) atendiendo que ambos vectores están ordenados. Como el recorrido es a lo sumo de 2n (todos los elementos son diferentes) entonces el costo es O(n) no siendo necesario ningún espacio adicional excepto algunas las variables temporales para los índices y el contador.

Tema 5 (8p)

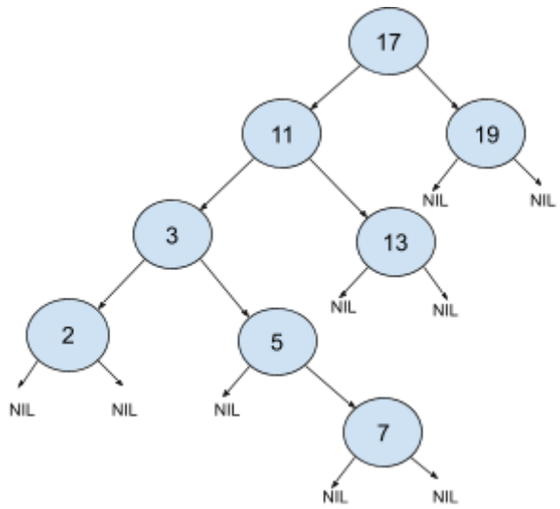
a) Considerar este árbol binario, asignar las claves `2,3,5,7,11,13,17 y 19` a los nodos en blanco de manera que satisfaga la condición de BST.



b) Explique si es posible que este árbol pueda corresponder a un árbol rojinegro. Si es posible, convertirlo en un rojinegro asignando los colores a los nodos. Si NO es posible, explique porque y plantee una solución para convertir el árbol BST en un árbol rojinegro..

Criterios de corrección: Parte a) asignación correcta de claves - 2p Parte b) Explicación correcta mostrando el árbol rojinegro final. 6p.

Posible ordenación



Por simple inspección puede verse que no puede asignarse los colores a los nodos para formar un rojinegro, fijarse que el nodo 19 tendrá un longitud de número de nodos negros hasta la hoja muy inferior que el nodo 7.

Una posible solución es hacer un rotación a la derecha sobre la raíz y asignar los colores de forma que los caminos negros sean todos del mismo número tomando cualquier nodo (por ejemplo desde la raiz el camino negro es de 4 hasta cualquier hoja)