Departamento de Lenguajes y Sistemas Informáticos

Material permitido: NINGUNO. **Duración:** 2 horas

Entregue este folio con sus datos consignados

Alumno:	**Identificación:**
C. Asociado en que realizó la Práctica Obligatoria:	

P1 (1'5 puntos) **Práctica**. Describa como implementaría un procedimiento de gestión de la entrada al aparcamiento si en lugar de existir una única cola de entrada en la que se mezclan coches familiares y no familiares, existiesen dos colas (cada una con su barrera de entrada), una para cada tipo de coches (nos referimos a la subsección **C. Control de Flujos. 1. Gestión del flujo de entrada** de la introducción del enunciado de la práctica). Detállense las modificaciones necesarias en las estructuras de datos que intervengan, así como el nuevo algoritmo de control de flujo (puede hacerse en pseudocódigo o explicarse, pero debe ser completo y no presentar ambigüedades).

---

1. (1 punto) Un árbol binario de enteros tiene la propiedad de ser un montículo de máximos si el valor de cada nodo padre es estrictamente mayor que el de cualquiera de sus nodos hijos. Implemente un algoritmo recursivo para el TAD árbol binario de enteros (`BTreeIF<int>`) que compruebe si un árbol binario de enteros cumple la propiedad de ser un montículo de máximos o no.

2. Un array disperso (Sparse Array) es una estructura que se diferencia de los arrays normales en el hecho de que si un índice no ha sido inicializado, no se reserva memoria para el mismo y, por tanto, no puede ser consultado. Suelen representarse mediante pares ordenados en los que la primera componente es el índice del array disperso y la segunda componente es el valor del array disperso para ese índice.

   Por ejemplo, un array disperso A que sólo tiene inicializados los índices 1 y 100000 a v y w respectivamente (es decir, $A[1] = v$ y $A[100000] = w$ utilizando la notación convencional de los arrays), se representaría como $\{(1, v), (100000, w)\}$.

   SparseArrayIF

```
//Representa un array disperso de elementos
public interface SparseArrayIF<T>{

/* Comprueba si el array disperso contiene la componente
*  i-ésima [0'5 puntos]*/
public boolean contains(int i);

/* Devuelve el primer índice ocupado del array disperso
*  [0'25 puntos] */
public int firstIndex();

/* Devuelve el último índice ocupado del array disperso
*  [0'25 puntos] */
public int lastIndex();

/* Devuelve el siguiente índice ocupado del array estrictamente
*  mayor que i. Si no existe ninguno, debe devolver i
*  [0'5 puntos] */
public int nextIndex(int i);

/* Elimina la componente i-ésima [0'5 puntos] */
public void remove (int i);
```

```
    /* Modifica la componente i-ésima para que contenga el valor v.
    *  Si no existe, la deberá añadir [1 punto] */
    public void modify(int i, T v);

    /* Devuelve el valor contenido en la componente i-ésima, la cual
    *  deberá existir [1 punto] */
    *  @prec { this.contains(i); }
    public <T> value(int i);

    /* Comprueba si dos arrays dispersos tienen o no intersección
    * (componentes con igual índice) [1 punto] */
    public boolean intersects(SparseArray<T> A);

    /* Devuelve la unión de dos arrays dispersos. Si alguna
    *  componente existe en ambos, devolverá la del objeto
    *  llamante (this) [1 punto] */
    public SparseArrayIF<T> union(SparseArrayIF<T> A);
}
```

a) (1 punto) Detalle el constructor de una clase que implemente esta interfaz. Describa detalladamente cómo realizaría la representación interna de este tipo. Justifique su elección. (Pista: Puede ser conveniente crearse una clase que represente elementos de un multiconjunto)

b) (6 puntos) Basándose en la respuesta anterior, implemente todos los métodos de la interfaz `SparseArrayIF`. Se valorará la eficiencia. (Nota: las puntuaciones asignadas a cada método se indican en la especificación de dicho método en la interfaz del tipo)

c) (0'5 puntos) Calcule el coste asintótico temporal en el caso peor del método `SparseArrayIF< T> union( SparseArrayIF<T> A)` para la implementación realizada.

ListIF (Lista)

```
/* Representa una lista de elementos */
public interface ListIF<T>{
   /* Devuelve la cabeza de una lista*/
    *
    public T getFirst ();
    /* Devuelve: la lista excluyendo la
       cabeza. No modifica la estructura
       */
    public ListIF<T> getTail ();
   /* Inserta una elemento (modifica la
      estructura)
    * Devuelve: la lista modificada
    * @param elem El elemento que hay que
       añadir*/
    public ListIF<T> insert (T elem);
    /* Devuelve: cierto si la lista esta
       vacia */
    public boolean isEmpty ();
    /* Devuelve: cierto si la lista esta
       llena*/
    public boolean isFull();
    /* Devuelve: el numero de elementos
       de la lista*/
    public int getLength ();
    /* Devuelve: cierto si la lista
       contiene el elemento.
     * @param elem El elemento buscado */
    public boolean contains (T elem);
    /* Ordena la lista (modifica la lista)
     * @Devuelve: la lista ordenada
     * @param comparator El comparador de
        elementos*/
    public ListIF<T> sort
       (ComparatorIF<T> comparator);
    /*Devuelve: un iterador para la
       lista*/
    public IteratorIF<T> getIterator ();
}
```

StackIF (Pila)

```
/* Representa una pila de elementos */
public interface StackIF <T>{
    /* Devuelve:  la cima de la pila */
     public T getTop ();
    /* Incluye un elemento en la cima de
       la pila (modifica la estructura)
     * Devuelve: la pila incluyendo el
        elemento
     * @param elem Elemento que se quiere
        añadir */
     public StackIF<T> push (T elem);
    /* Elimina la cima de la pila
       (modifica la estructura)
     * Devuelve: la pila excluyendo la
        cabeza */
     public StackIF<T> pop ();
    /* Devuelve: cierto si la pila esta
       vacia */
     public boolean isEmpty ();
    /* Devuelve: cierto si la pila esta
       llena */
     public boolean isFull();
    /* Devuelve: el numero de elementos
       de la pila */
     public int getLength ();
    /* Devuelve: cierto si la pila
       contiene el elemento
     * @param elem Elemento buscado */
     public boolean contains (T elem);
    /*Devuelve: un iterador para la pila*/
     public IteratorIF<T> getIterator ();
}
```

QueueIF (Cola)

```
/* Representa una cola de elementos */
public interface QueueIF <T>{
    /* Devuelve: la cabeza de la cola */
     public T getFirst ();
    /* Incluye un elemento al final de la
       cola (modifica la estructura)
     * Devuelve: la cola incluyendo el
        elemento
     * @param elem Elemento que se quiere
        añadir */
     public QueueIF<T> add (T elem);
    /* Elimina el principio de la cola
       (modifica la estructura)
     * Devuelve: la cola excluyendo la
        cabeza    */
     public QueueIF<T> remove ();
    /* Devuelve: cierto si la cola esta
       vacia */
     public boolean isEmpty ();
    /* Devuelve: cierto si la cola esta
       llena */
     public boolean isFull();
    /* Devuelve: el numero de elementos
       de la cola */
     public int getLength ();
    /* Devuelve: cierto si la cola
       contiene el elemento
     * @param elem elemento buscado */
     public boolean contains (T elem);
    /*Devuelve: un iterador para la cola*/
     public IteratorIF<T> getIterator ();
}
```

TreeIF (Árbol general)

```
/* Representa un arbol general de
   elementos */
public interface TreeIF <T>{
    public int PREORDER  = 0;
    public int INORDER   = 1;
    public int POSTORDER = 2;
    public int BREADTH   = 3;
    /* Devuelve:  elemento raiz del arbol
       */
     public T getRoot ();
    /* Devuelve: lista de  hijos de un
       arbol.*/
     public ListIF <TreeIF <T>>
        getChildren ();
```

```
    /* Establece el elemento raiz.
     * @param elem Elemento que se quiere
        poner como raiz*/
     public void setRoot (T element);
    /* Inserta un subarbol como ultimo
       hijo
     * @param child el hijo a insertar*/
     public void addChild (TreeIF<T>
        child);
    /* Elimina el subarbol hijo en la
       posicion index-esima
     * @param index indice del subarbol
        comenzando en 0*/
     public void removeChild (int index);
    /* Devuelve: cierto si el arbol es un
       nodo hoja*/
     public boolean isLeaf ();
    /* Devuelve: cierto si el arbol es
       vacio*/
     public boolean isEmpty ();
    /* Devuelve: cierto si la lista
       contiene el elemento
     * @param elem Elemento buscado*/
     public boolean contains (T element);
    /* Devuelve: un iterador para la lista
     * @param traversalType el tipo de
        recorrido, que
     * sera PREORDER, POSTORDER o BREADTH
        */
     public IteratorIF<T> getIterator
        (int traversalType);
}
```

BTreeIF (Árbol Binario)

```
/* Representa un arbol binario de
   elementos */
public interface BTreeIF <T>{
  public int PREORDER  = 0;
  public int INORDER   = 1;
  public int POSTORDER = 2;
  public int LRBREADTH = 3;
  public int RLBREADTH = 4;
 /* Devuelve: el elemento raiz del arbol
    */
  public T getRoot ();
 /* Devuelve: el subarbol izquierdo o
    null si no existe */
  public BTreeIF <T> getLeftChild ();
 /* Devuelve: el subarbol derecho o null
    si no existe */
  public BTreeIF <T> getRightChild ();
 /* Establece el elemento raiz
  * @param elem Elemento para poner en la
     raiz */
  public void setRoot (T elem);
 /* Establece el subarbol izquierdo
  * @param tree el arbol para poner como
     hijo izquierdo */
  public void setLeftChild (BTreeIF <T>
     tree);
 /* Establece el subarbol derecho
  * @param tree el arbol para poner como
     hijo derecho */
  public void setRightChild (BTreeIF <T>
     tree);
 /* Borra el subarbol izquierdo */
  public void removeLeftChild ();
 /* Borra el subarbol derecho */
  public void removeRightChild ();
 /* Devuelve: cierto si el arbol es un
    nodo hoja*/
  public boolean isLeaf ();
 /* Devuelve: cierto si el arbol es vacio
    */
  public boolean isEmpty ();
 /* Devuelve: cierto si el arbol contiene
    el elemento
  * @param elem Elemento buscado */
  public boolean contains (T elem);
 /* Devuelve un iterador para la lista.
  * @param traversalType el tipo de
     recorrido que sera
    PREORDER, POSTORDER, INORDER,
       LRBREADTH o RLBREADTH */
 public IteratorIF<T> getIterator (int
    traversalType);
}
```

ComparatorIF

```
/* Representa un comparador entre
   elementos */
public interface ComparatorIF<T>{
  public static int LESS  = -1;
  public static int EQUAL = 0;
  public static int GREATER = 1;
 /* Devuelve: el orden de los elementos
  * Compara dos elementos para indicar si
     el primero es
  * menor, igual o mayor que el segundo
     elemento
  * @param e1 el primer elemento
  * @param e2 el segundo elemento   */
  public int compare (T e1, T e2);
 /* Devuelve: cierto si un elemento es
    menor que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento */
  public boolean isLess (T e1, T e2);
 /* Devuelve: cierto si un elemento es
    igual que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento */
  public boolean isEqual (T e1, T e2);
 /* Devuelve: cierto si un elemento es
    mayor que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento*/
  public boolean isGreater (T e1, T e2);
}
```

IteratorIF

```
/* Representa un iterador sobre una
   abstraccion de datos */
public interface IteratorIF<T>{
```

```
    /* Devuelve: el siguiente elemento de
       la iteracion */
     public T getNext ();
    /* Devuelve: cierto si existen mas
       elementos en el iterador */
     public boolean hasNext ();
    /* Restablece el iterador para volver
       a recorrer la estructura */
     public void reset ();
}
```