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.** Suponga que se permite que una de las pistas (y como máximo una) de nuestro aeropuerto pueda estar inoperativa. Para ello, se incorpora a la torre de control el método `public void closeRunway (int index)`, que marca a la pista con índice `index` (los índices comienzan en 1) como cerrada, de modo que ya no se puede realizar ninguna inserción en ella. Además, este método reubica a todos los vuelos de la pista que se ha marcado como cerrada en las demás pistas, cumpliéndose los requisitos que se tenían antes para la asignación de pista cuando se pedía pista para una operación. Implemente el método `public void closeRunway (int index)` e indique si hay que realizar modificaciones adicionales en la torre de control

---

1. (1 punto) Diseñe un método `ListIF<int> digits (int input)` que, dado un número, obtenga la lista de dígitos que lo representa

2. (1'5 puntos) Diseñe un método `ListIF<int> repeated(ListIF<int> input)` que genere una lista (de salida) con el número de veces que aparece cada elemento en la lista de entrada, donde el elemento i-ésimo de la lista de salida se corresponde con el número de veces que aparece el elemento i-ésimo (distinto) en la de entrada (por ejemplo, para la entrada $[0, 3, 0, 2, 8, 8, 4, 5]$ el resultado sería $[2, 1, 1, 2, 1, 1]$ (0 aparece 2 veces, 3 aparece 1, 2 está una vez, hay 2 ochos, un cuatro y un cinco). Se valorará la eficiencia

3. Se pretende implementar, a partir de los TAD estudiados en la asignatura, un nuevo TAD Dictionary, cuyo propósito consiste en albergar una colección de palabras con sus respectivas acepciones. Las palabras se encuentran almacenadas en el diccionario en orden lexicográfico (el de un diccionario al uso). Una palabra puede tener asociada una colección de posibles acepciones.

   Un ejemplo de entrada de diccionario sería la siguiente:

   trueque

   a) Acción y efecto de trocar o trocarse.

   b) Intercambio directo de bienes y servicios, sin mediar la intervención de dinero.

   La definición del interfaz del tipo es la siguiente:

   DictionaryIF

```
/** Representa un diccionario de terminos (palabras)
* cada una de las cuales incluye una colección de
* acepciones  */
public interface DictionaryIF{

 // Añade una definición a una palabra del diccionario
 public void addDefinition (String term, String definition);

 // Devuelve la primera definición del término
 public String getDefinition (String term);

 // Devuelve la index-ésima definición del término
 public String getDefinition (String term, int index);
```

```
    // Devuelve la lista completa de definiciones del término
    public ListIF<String> getDefinitions (String term);
}
```

Nota: se permite utilizar los métodos de la clase `String`, en particular, se recomiendan **`int`** `length()`, que devuelve la longitud de una cadena (String) y **`char`** `charAt(`**`int`** `index)`, que, dada una posición de la cadena indexada desde 0, devuelve el carácter que ocupa dicha posición.

a) (0'5 puntos) Defina la estructura o representación interna del tipo (usando los TAD estudiados en la asignatura). Justifique su elección

b) (5 puntos) Implemente todos los metodos de la interfaz `DictionaryIF`

c) (0'5 puntos) Obtenga el coste asintótico temporal en el caso peor del metodo `getDefinitions`

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 ();
}
```