Estructuras de Datos
Grupos B y D
Convocatoria extraordinaria 2022-2023

Recuerda poner tu nombre y tu usuario del juez en un comentario al principio de todos los archivos que entregues

Ejercicio 1 [2,75 puntos]

Se desea añadir una nueva operación `extrae_e_inserta` a la implementación del TAD Lista basada en nodos doblemente enlazados, que extrae una secuencia de elementos consecutivos de una lista, e inserta en su lugar otra secuencia de elementos dada. La operación tendrá el siguiente prototipo:

```
Lista extrae_e_inserta(unsigned int pos_inicio, unsigned int n_elems_a_extraer, Lista & elems_a_insertar);
```

El parámetro `pos_inicio` marca la posición de inicio del tramo a extraer (las posiciones en la lista comienzan a contarse desde 0, es decir, la posición 0 se corresponde con el primer elemento de la lista, la posición 1 con el segundo, y así sucesivamente…). El parámetro `n_elems_a_extraer` representa el número de elementos a extraer, y el parámetro `elems_a_insertar` contiene los elementos que van a sustituir al tramo extraído. De esta forma:

- Si no se puede extraer ningún elemento (bien porque la posición `pos_inicio` no existe, bien porque se indique `0` como valor de `n_elems_a_extraer`), o no se puede insertar ningún elemento (debido a que `elems_a_insertar` está vacía), la operación no tiene ningún efecto, y el valor devuelto por la función es la lista vacía.
- En otro caso, la operación extrae `n_elems_a_extraer` elementos desde la posición `pos_inicio` (incluida), dejando en su lugar los elementos de la lista `elems_a_insertar` (si se pide extraer más elementos de los que hay a partir del punto de extracción, extraerá todos los que haya a partir de dicho punto). Así mismo, `elems_a_insertar` pasa a ser la lista vacía, y el valor devuelto por la función es la lista con los elementos extraídos.

Ejemplo 1: dadas las listas *a* y *b*

a

10	11	12	13

b

41	42	43

tras la invocación `a.extrae_e_inserta(2,3,b)` la lista *a* pasa a ser:

a

10	11	41	42	43

la lista *b* pasa a ser la lista vacía, y la operación devuelve la siguiente lista con los elementos extraídos:

12	13

Ejemplo 2: dadas las listas *a* y *b*

a

10	11	12	13

b

41	42	43

tras la invocación `a.extrae_e_inserta(0,2,b)` la lista *a* pasa a ser:

a

41	42	43	12	13

la lista *b* pasa a ser la lista vacía, y la operación devuelve la siguiente lista con los elementos extraídos:

10	11

Ten en cuenta que

- La operación no podrá invocar, ni directa, ni indirectamente, ninguna operación de manejo de memoria dinámica (*new*, *delete*). Tampoco podrá asignar valores a los contenidos de los nodos, ni realizar copias de dichos valores en otras variables.
- La operación deberá ser lo más eficiente posible.

Además de implementar esta operación, deberás determinar justificadamente su complejidad.

Ejercicio 2 [2,75 puntos]

Un árbol binario de enteros es *peculiar* cuando:

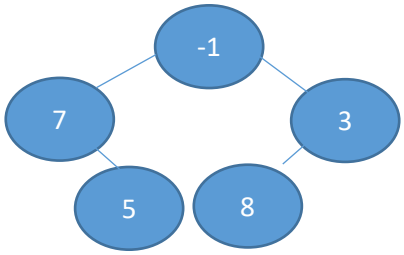
1. Es vacío, o
2. Tiene un único nodo, o
3. Tiene más de un nodo y además se cumplen las siguientes condiciones:
   3.1. El valor de la **raíz** del árbol es:

3.1.1. Menor que el mínimo de los valores de su hijo izquierdo, cuando el hijo derecho es vacío.
3.1.2. Mayor que el máximo de los valores de su hijo derecho, cuando el hijo izquierdo es vacío.
3.1.3. Igual a la diferencia del número de valores pares en el hijo derecho menos el número de valores impares en el hijo izquierdo, cuando ninguno de los dos hijos (el izquierdo, y el derecho) son vacíos.

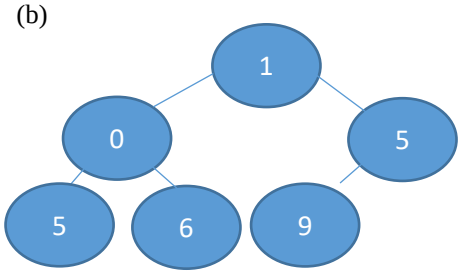
3.2. Además, tanto el hijo izquierdo como el derecho son peculiares.

Por ejemplo, el árbol (a) en la siguiente figura es *peculiar*, mientras que el árbol (b) no lo es:

(a)



(b)



Desarrolla un algoritmo eficiente que, tomando como entrada un árbol binario de enteros, devuelta *true* si el árbol es *peculiar*, y *false* en caso contrario. Además de implementar el algoritmo, debes determinar justificadamente su complejidad.

Ejercicio 3 [4,5 puntos]

El altamente exclusivo restaurante *ApetitoXO* dispone de una única mesa y nos han encargado la implementación de un sistema de reservas que dé soporte al siguiente comportamiento:

- Un cliente que quiera reservar la mesa debe realizar una solicitud de reserva, para lo cual debe indicar su DNI (que lo identifica unívocamente) y la fecha para la que solicita la reserva, y quedará registrada su solicitud de reserva en el sistema de reservas y el cliente colocado al final de una fila de espera.
- Un mismo cliente no puede tener más de una solicitud de reserva.
- Se puede consultar si la mesa está disponible para una cierta fecha.
- Las solicitudes de reserva de la mesa se atienden por estricto orden de solicitud. Para que pueda efectuarse la reserva en el momento en que se atiende una solicitud, la mesa debe estar disponible para la fecha indicada en dicha solicitud; si eso ocurre, el cliente conseguirá reservar la mesa y quedará registrada su reserva en el sistema, y su solicitud de reserva desaparecerá del sistema; en caso contrario, la reserva no podrá realizarse y el cliente, después de haber esperado, desaparecerá del sistema de reservas sin lograr su objetivo (su solicitud de reserva desaparecerá del sistema).
- La dirección del restaurante debe poder obtener un listado de las reservas conseguidas, ordenado crecientemente por las fechas de las mismas.
- Un cliente *c1* que ha efectuado una solicitud de reserva y que, por tanto, esté en la fila de espera, puede ceder su puesto en la fila a otro *c2* que vaya a efectuar una solicitud, lo que hará que: (a) la solicitud de reserva de *c1* desaparezca del sistema, y (b) la solicitud de reserva de *c2* se incorpore al sistema, pero ocupando la posición en la fila de espera que ocupaba *c1* en lugar de colocarse al final de la lista de espera.

Para llevar a cabo esta parte de la implementación de este sistema de reservas hemos decidido desarrollar un TAD `Apetitoxo` con las siguientes operaciones:

- `Apetitoxo()`: Operación constructora que crea un sistema de reservas vacío.
- `annadeSolicitud(dni, fecha)`: Incorpora la solicitud de reserva del cliente con DNI `dni` para la fecha `fecha`. Si hay un cliente con ese DNI esperando en la fila de reservas, la operación lanzará una excepción `EErrorCliente`.
- `enEspera(dni)`→`fecha`: Devuelve la fecha solicitada por el cliente con DNI `dni` que está en la fila de espera. Si no hay un cliente con ese DNI esperando en la fila de reservas, la operación lanzará una excepción `EErrorCliente`.
- `primeroEnEspera()`→`dni`: Devuelve el `dni` de la primera solicitud de reserva. Si no hay nadie esperando, la operación lanzará una excepción `ENoHaySolicitudes`.
- `atiendeSolicitud()`→`bool`: Realiza la reserva correspondiente a la primera solicitud en espera. Si no hay solicitudes la operación no tendrá efecto. Si hay al menos una solicitud, para poder realizar la que está en primer lugar la mesa debe estar disponible para la fecha pedida; si eso ocurre, el cliente conseguirá reservar la mesa para la fecha pedida y dejará de estar a la espera; en caso contrario, la reserva no tendrá éxito y el cliente desaparecerá del sistema de reservas. Devuelve `false` si no hay solicitudes que atender o si hay pero la primera no tiene éxito; y `true` si la primera solicitud puede realizarse.
- `disponible(fecha)`→`bool`: Devuelve `true` si en la fecha recibida la mesa está disponible y `false` en caso contrario.
- `reservas()`→`lista<fecha+dni>`: Devuelve un listado de las reservas realizadas, ordenado crecientemente por fecha.
- `cedePuesto(dni_cede, dni_nuevo, fecha_nuevo)`: Hace que el cliente con DNI `dni_cede` que tiene una solicitud de reserva deje su puesto en la fila de reservas a un nuevo cliente, con DNI `dni_nuevo`, que va a efectuar una solicitud de reserva para la fecha `fecha_nuevo`. La solicitud de reserva del cliente con DNI `dni_cede` desaparecerá del sistema. La solicitud de reserva del cliente con DNI `dni_nuevo` quedará registrada en el sistema. Si el cliente con DNI `dni_cede` no ha solicitado una reserva o el cliente con DNI `dni_nuevo` ya tiene una solicitud de reserva en el sistema, la operación lanzará una excepción `EErrorCliente`.

Ten en cuenta que la implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD. Además, debes implementar las operaciones y justificar la complejidad de cada una de ellas.