Estructuras de Datos

Grupos B y G

Convocatoria ordinaria 2023-2024

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

Ejercicio 1 [2.5 puntos]

Deseamos añadir una nueva operación `rotar` a la implementación del TAD `Lista` basada en nodos doblemente enlazados. El prototipo de dicha operación es el siguiente:

```
void rotar(int k);
```

Si *k* es positivo, debe mover los *k* últimos elementos al comienzo de la lista. Si *k* es negativo, debe mover los |*k*| primeros elementos al final de la lista. En ambos casos, el orden relativo entre los elementos movidos no se verá alterado. En caso de que la lista tenga |*k*| elementos o menos, la operación no tendrá ningún efecto.

Por ejemplo, si la lista contiene los elementos `[1,2,3,4,5]`, `rotar(2)` convertirá dicha lista en `[4,5,1,2,3]` (habrá movido los dos últimos elementos `4,5` al principio de la lista), mientras que `rotar(-2)` la convertirá en `[3,4,5,1,2]` (habrá movido los dos primeros elementos `1,2` al final de la lista).

Ten en cuenta que

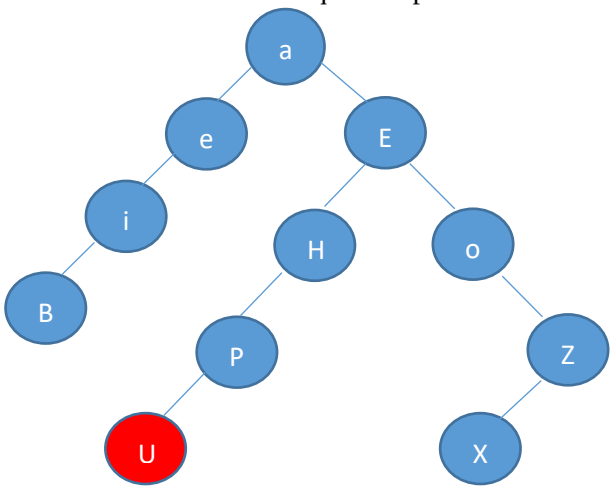
- La operación no podrá invocar, ni directa, ni indirectamente, ninguna operación de manejo de memoria dinámica (*new*, *delete*). Tampoco podrá asignar valores a los contenidos de los nodos, ni realizar copias de dichos valores en otras variables.
- La operación deberá ser lo más eficiente posible.

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

Ejercicio 2 [3 puntos]

Dado un árbol de caracteres, la *vocalización* de un nodo es el número de vocales que hay en la rama que une a dicho nodo con la raíz (nodo y raíz incluidos). Debes desarrollar un algoritmo eficiente que, dado un árbol de caracteres, con una letra en cada nodo, devuelva la letra del nodo con mayor *vocalización*. En caso de empate, deberá devolver la letra del nodo que se encuentra a mayor profundidad, y, si el empate persiste, la del que esté más a la izquierda. En caso de que el árbol esté vacío, deberá devolver el carácter `'?'`.

Por ejemplo, en el siguiente árbol se señala el nodo que cumple la condición establecida por el problema.



La *vocalización* del nodo señalado es 3 (en la rama que lo une con la raíz hay 3 vocales: `a`, `E` y `U`). Los nodos que contienen las letras `i`, `B`, `o` y `Z` tienen también vocalización 3, pero están a una profundidad menor. El nodo que contiene la letra `X` tiene también una vocalización de 3, está a la misma profundidad que el señalado, pero más a la derecha. Por tanto, cuando procese este árbol el algoritmo deberá devolver la letra `U`.

El punto de entrada a este algoritmo será la función:

```
char max_vocalizado(const Arbin<char>& a)
```

Aparte de implementar el algoritmo, debes determinar justificadamente su complejidad.

Ejercicio 3 [4.5 puntos]

La multinacional española `Min®` ha lanzado nuevo sistema de vídeo-por-internet, llamado `RedPelis`. Los usuarios pagan una cuota fija al mes, más un pequeño extra por cada minuto de visionado. Nos han encargado desarrollar un TAD para gestionar este sistema, que debe incluir las siguientes operaciones:

- Un constructor `RedPelis(cuotaMensual, extraMinuto)`, en el que podremos indicar la cuota mensual fija que deben abonar los usuarios (`cuotaMensual`), más el precio extra por minuto de visionado (`extraMinuto`).
- Una operación `registra_usuario(id_usuario, tarjeta)`, que registrará en el sistema un usuario con identificador `id_usuario`, y con una tarjeta de crédito asociada (`tarjeta`). En caso de que ya exista en el sistema un usuario con el mismo identificador se levantará una excepción `EUsuarioDuplicado`.
- Una operación `registra_peli(id_pelicula, num_registro)`, que registrará en el sistema una película con identificador `id_pelicula` y con número de registro `num_registro`. En caso de que la película con el identificador indicado ya esté dada de alta en el sistema, levantará una excepción `EPeliculaDuplicada`.
- Una operación `mira(id_usuario,id_pelicula,minutos)`, que actualizará la información del sistema cuando un usuario con identificador `id_usuario` ha visualizado una película con identificador `id_pelicula` una determinada cantidad de minutos (`minutos`). En caso de que el usuario no exista, la operación levantará una excepción `EUsuarioNoExiste`. Si la película no existe, la operación levantará una excepción `EPeliculaNoExiste`.
- Una operación `factura(id_usuario)`, que devuelve la factura mensual para un usuario con identificador id_`usuario`. La factura devuelta contendrá los siguientes datos: identificador del usuario, número de tarjeta, y cantidad total facturada. En caso de que el usuario no exista, levantará una excepción `EUsuarioNoExiste`.
- Una operación `hits(k)`, que devuelve una lista con los identificadores de las *k* películas más vistas, ordenadas descendentemente (de mayor a menor) por minutos de visionado recibidos. En caso de que haya que listar varias películas con el mismo tiempo de visionado, se listarán ordenadas ascendentemente por su número de registro (de todas las que tengan el mismo tiempo de visionado aparecerá primero el identificador de la que tenga el número de registro más pequeño, en segundo lugar la que tenga el número de registro siguiente más pequeño, y así sucesivamente). Si hay menos de *k* películas, se devolverá una lista con los identificadores de todas las películas registradas, ordenadas según los criterios indicados.

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.

> **Pista**: Ordenar una serie de películas de manera descendente por el tiempo de visionado *t* es equivalente a ordenar dichas películas de forma ascendente (de menor a mayor) utilizando *-t* como criterio de ordenación.