Estructuras de datos

Grados de la Facultad de Informática (UCM)

26 de Mayo de 2023

Normas de realización del examen

1. El examen dura **3 horas**.
2. Debes desarrollar e implementar soluciones para cada uno de los ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc`.
3. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen. El nombre de usuario y contraseña que has estado utilizando durante la evaluación continua **no** son válidos.
4. Escribe tu **nombre y apellidos** en el espacio reservado para ello en los ficheros proporcionados.
5. Del enlace **Material para descargar** dentro del juez puedes descargar un archivo comprimido que contiene material que puedes utilizar para la realización del examen (transparencias de clase, implementación de las estructuras de datos, una plantilla de código fuente y ficheros de texto con los casos de prueba de cada ejercicio del enunciado).
6. Dispones de un fichero plantilla para cada ejercicio. Debes utilizarlo atendiendo a las instrucciones que se dan en cada fichero y escribiendo tu solución en los espacios reservados para ello, siempre entre las etiquetas `//@ <answer>` y `//@ </answer>`.
7. Tus soluciones serán evaluadas por el profesor independientemente del veredicto del juez automático. Para ello, el profesor tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.
8. Si necesitas consultar la documentación de C++, está disponible en `http://exacrc/cppreference`.
9. A lo largo del examen se te pedirá que te identifiques y rellenes tus datos en una hoja de firma.

Primero resuelve el problema. Entonces, escribe el código.

— John Johnson

Comentar el código es como limpiar el cuarto de baño;
nadie quiere hacerlo, pero el resultado es siempre
una experiencia más agradable para uno mismo y sus invitados.

— Ryan Campbell

Ejercicio 1. Rotar una lista hacia la derecha (2 puntos)

Supongamos que, dada una lista de entrada, queremos desplazar todos los elementos que ocupan posiciones impares (comenzando a contar desde la posición 1), de modo que cada uno de ellos se mueva dos posiciones hacia la derecha. De este modo, el primer elemento de la lista pasará a ser el tercero, el tercer elemento de la lista pasará a ser el quinto, el quinto pasará a ser el séptimo, y así sucesivamente hasta llegar al último elemento que ocupe una posición impar, que acabará situado al principio de la lista.

Por ejemplo, supongamos la lista `["A", "B", "C", "D", "E", "F", "G"]`, donde se muestran en color los elementos que se encuentran en posiciones impares. Tras aplicar la transformación mencionada anteriormente, la lista pasará a ser `["G", "B", "A", "D", "C", "F", "E"]`. De un modo similar, al transformar la lista `[9, 4, 2, 6, 3, 8]` se obtiene `[3, 4, 9, 6, 2, 8]`.

Partiendo de una implementación de las **colas dobles** con listas doblemente enlazadas circulares con nodo fantasma, añade un método genérico `rotar_derecha` que realice esta transformación sobre la lista `this`, e indica su coste con respecto a la longitud de dicha lista.

Importante: en la implementación de `rotar_derecha`, la transformación de la lista enlazada debe hacerse solamente modificando los punteros de la misma. No está permitido crear nuevos nodos mediante `new`, ni liberar nodos mediante `delete`. Tampoco se permite copiar los valores de un nodo a otro.

Entrada

La entrada consta de varios casos de prueba. Cada uno de ellos ocupa dos líneas. La primera línea de cada caso indica la longitud N de la lista de entrada ($0 \leq N \leq 10^5$) y la segunda línea contiene los N elementos de la lista. Cada elemento de la lista es una cadena de texto sin espacios.

Salida

Para cada caso de prueba debe escribirse una línea con la lista resultante tras llamar al método `rotar_derecha`. La lista irá delimitada entre corchetes y sus elementos estarán separados por comas.

Entrada de ejemplo

```
5
A B C D E
4
Pim Pam Pum Fuera
2
Feliz Verano
```

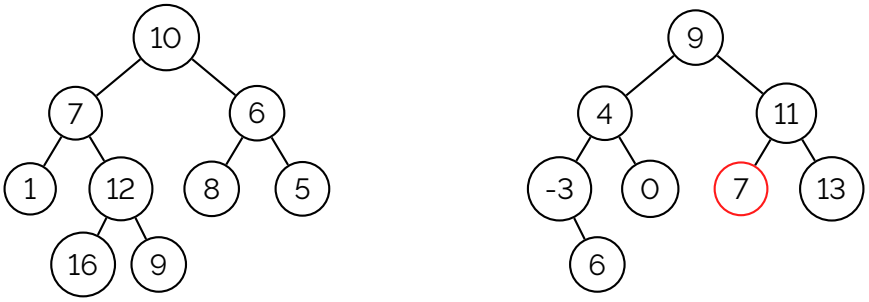
Salida de ejemplo

```
[E, B, A, D, C]
[Pum, Pam, Pim, Fuera]
[Feliz, Verano]
```

Ejercicio 2. Árbol binario de búsqueda por paridad (2 puntos)

Dado un árbol binario de números enteros, decimos que es *binario de búsqueda por paridad* (abreviado por ABBP) si cumple:

- es el árbol vacío, o bien es un árbol unitario, o bien
- si su raíz es par, todos los elementos pares de su hijo izquierdo son estrictamente mayores que la raíz y todos los elementos pares de su hijo derecho son estrictamente menores que la raíz, además sus hijos son también ABBP, o bien
- si su raíz es impar, todos los elementos impares de su hijo izquierdo son estrictamente menores que la raíz y todos los elementos impares de su hijo derecho son estrictamente mayores que la raíz, además sus hijos son también ABBP.



Por ejemplo, el árbol de la izquierda sí cumple que es ABBP; mientras que el árbol de la derecha no lo cumple ya que su hijo izquierdo tiene un valor impar (7) que es menor que el valor impar de su raíz (nótese que en cambio sus dos hijos sí son ABBP).

Importante: Se debe implementar una función recursiva *externa* a la clase `Arbin` que explore el árbol de manera eficiente averiguando si es un árbol binario de búsqueda por paridad o no. La función no podrá tener parámetros de entrada/salida.

Entrada

La entrada comienza indicando el número de casos de prueba que vendrán a continuación. Cada caso consiste en la descripción de un árbol binario de enteros con valores comprendidos entre -10^8 y 10^8. El árbol vacío se representa con un '`.`' y un árbol no vacío con raíz `R`, hijo izquierdo `i` e hijo derecho `dr` se representa como `(iz R dr)`.

Salida

Para cada árbol, se escribirá una línea con un `SI` si es un árbol binario de búsqueda por paridad y un `NO` si no lo es.

Entrada de ejemplo

```
6
(((. 1 .) 7 ((. 16 .) 12 (. 9 .))) 10 ((. 8 .) 6 (. 5 .)))
(((. -3 (. 6 .)) 4 (. 0 .)) 9 ((. 7 .) 11 (. 13 .)))
((. 6 .) 2 .)
((. 3 .) 2 (. 3 .))
(((. 1 .) 5 .) 8 ((. 6 .) 2 .))
(((. 6 .) 2 .) 8 (. 1 .))
```

Salida de ejemplo

```
SI
NO
SI
SI
SI
NO
```

Ejercicio 3. Detective de moda (3 puntos)

Rocío tiene una tienda un tanto peculiar enfrente de su casa. La tienda solo vende camisetas, y cada camiseta tiene un color, aunque se puede dar el caso de que varias camisetas a la venta tengan el mismo color.

La tienda tiene todas las camisetas dispuestas en una barra horizontal, y tiene una política de compras un tanto restrictiva: solamente se permite comprar las camisetas que se encuentren en los extremos de la barra, para que no se desordenen mucho.

Para vestir de exclusiva, Rocío quiere comprar una camiseta que sea única, es decir que no haya otra de su mismo color en la tienda en el momento de adquirirla. Está dispuesta a comprar camisetas que no sean únicas con tal de llevarse una que sí lo sea. Pero como tampoco le sobra el dinero, no quiere que sean demasiadas.

Queremos ayudar a Rocío con un TAD `TiendaCamisetas` con las siguientes operaciones:

- `TiendaCamisetas()` (constructor): crea una tienda vacía, sin camisetas.
- `inserta_izquierda(color)`: registra que una nueva camiseta de color `color` ha sido añadida a la barra de camisetas por su extremo izquierdo.
- `inserta_derecha(color)`: registra que una nueva camiseta de color `color` ha sido añadida a la barra de camisetas por su extremo derecho.
- `compra_izquierda()`: registra el hecho de que la camiseta del extremo izquierdo ha sido comprada, por lo que ya no estará disponible en la tienda. Si en ese momento la tienda no tiene camisetas a la venta se lanzará una excepción `domain_error` con el mensaje `Tienda sin camisetas`.
- `compra_derecha()`: registra el hecho de que la camiseta del extremo derecho ha sido comprada. Si en ese momento la tienda no tiene camisetas a la venta se lanzará una excepción `domain_error` con el mensaje `Tienda sin camisetas`.
- `pregunta()`: devuelve un `string` con la respuesta a la pregunta que se hace Rocío sobre cuántas camisetas consecutivas como mínimo tiene que comprar, empezando por uno de los extremos, para llevarse a casa una camiseta que sea única en la tienda. La respuesta comenzará por ese número (para convertir un entero en un `string` puedes utilizar la función `to_string`), al que seguirá, separado por un espacio, el nombre del extremo en cuestión: `IZQUIERDA` o `DERECHA`. Si pudiesen ser ambos, el nombre del extremo se sustituirá por la palabra `CUALQUIERA`. Si en el momento de hacer una pregunta no hay ninguna camiseta de un color único en la tienda (o simplemente no hay camisetas), la respuesta devuelta será `NADA INTERESANTE`. La tienda no se modifica como resultado de la pregunta.

Importante: La implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir la representación más adecuada para el TAD, implementar las operaciones y justificar la complejidad resultante.

Los métodos del TAD no deben mostrar nada por pantalla. El manejo de la entrada y salida de datos se realizará en funciones externas al TAD.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso está formado por una serie de líneas, en las que se muestran las operaciones a llevar a cabo, una por cada línea: el nombre de la operación seguido de sus argumentos. Los nombres de los colores serán cadenas de caracteres sin espacios. La palabra `FIN` en una línea indica el final de cada caso de prueba.

Salida

Por cada caso de prueba el programa deberá responder a todas las preguntas que se haga Rocío, en orden cronológico, escribiendo la respuesta devuelta por la operación `pregunta`.

Si una operación produce un error, entonces se escribirá una línea con `ERROR:`, seguido del mensaje de error que devuelve la operación.

Al final de cada caso de prueba se escribirán tres guiones –.

Entrada de ejemplo

```
inserta_derecha amarillo
inserta_derecha verde
inserta_derecha violeta
inserta_izquierda azul
inserta_derecha azul
pregunta
inserta_izquierda rojo
pregunta
inserta_izquierda violeta
inserta_derecha amarillo
inserta_derecha rojo
pregunta
inserta_izquierda verde
pregunta
FIN
inserta_derecha blanco
inserta_derecha negro
inserta_derecha gris
inserta_derecha negro
inserta_derecha blanco
pregunta
compra_izquierda
pregunta
inserta_derecha blanco
pregunta
FIN
compra_derecha
FIN
```

Salida de ejemplo

```
2 CUALQUIERA
1 IZQUIERDA
5 CUALQUIERA
NADA INTERESANTE
---
3 CUALQUIERA
1 DERECHA
2 IZQUIERDA
---
ERROR: Tienda sin camisetas
---
```