Estructuras de datos

Grados de la Facultad de Informática (UCM)

26 de Mayo de 2022

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. Al terminar el examen, dirígete al puesto del profesor y rellena con tus datos la hoja de firmas que él tendrá. Muéstrale tu documento de identificación.

Primero resuelve el problema. Entonces, escribe el código.

— John Johnson

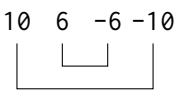
Comentar el código es como limpiar el cuarto de baño;
nadie quiere hacerlo, pero el resultado es siempre
una experiencia más agradable para uno mismo y sus invitados.

— Ryan Campbell

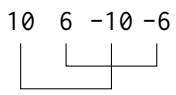
Ejercicio 1. Listas equilibradas de números enteros (2 puntos)

Supongamos una lista de números enteros, todos distintos entre sí y todos distintos de 0, tal que para cada número *x* de la lista, su opuesto *–x* también está en la lista. En este caso, decimos que *x* y *–x* están *emparejados*. Una lista de este tipo está *equilibrada* si, para cada pareja (*x*, *–x*) de la lista, el número positivo de la pareja siempre aparece antes que el negativo y además las parejas están correctamente anidadas unas dentro de otras. Es decir, para cada número negativo de la lista, su pareja es el número anterior más cercano que aún no estaba emparejado con otro. Por ejemplo, la lista `10 6 -6 -10` está equilibrada, pero la lista `10 6 -10 -6` no lo está:

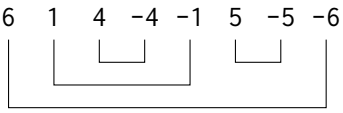
Equilibrada



No equilibrada



Otro ejemplo de lista equilibrada es `6 1 4 -4 -1 5 -5 -6`:



Tenía varias secuencias equilibradas guardadas en un fichero, pero resulta que el fichero se ha corrompido de una manera un tanto extraña, porque solo se han visto afectados los números negativos. Algunos de los que estaban al final de la lista han desaparecido sin más. Otros han sido reemplazados por un `0`, y por si fuera poco, han aparecido números negativos que no estaban en la lista original. Por ejemplo, al intentar recuperar la lista anterior me he encontrado lo siguiente:

Como puedes ver, el número `-6`, que estaba al final de la lista, ha desaparecido. Los números `-4` y `-1` han sido reemplazados por ceros. Además ha aparecido el número `-7`, que no estaba en la lista original. Pese al fastidio, creo que he tenido suerte, porque puedo reconstruir la lista original a partir de la lista corrupta.

Requisitos de implementación: En este ejercicio se pide implementar una función con la siguiente cabecera:

```
void reconstruir(list<int> &lista);
```

La función recibe una lista corrupta, y debe transformarla para obtener la lista equilibrada original. Para ello se deben aplicar los cambios directamente sobre la `lista` pasada como parámetro, pudiéndose eliminar, añadir o modificar elementos de ella. Se permite el uso de otros TADs auxiliares, pero **no** se permite el uso de una lista paralela en la que ir construyendo el resultado, para luego volcarlo a la lista pasada como parámetro.

No olvides indicar el coste de la función.

Entrada

La entrada consta de varios casos de prueba. Cada uno de ellos ocupa dos líneas. La primera línea contiene el número *N* de elementos de la lista corrupta. La segunda lista contiene los *N* elementos de la lista corrupta, separados por espacios.

Salida

Para cada caso de prueba ha de imprimirse una línea con los elementos de la lista equilibrada original, separados por espacios. Si la lista original es vacía, debe imprimirse una línea en blanco.

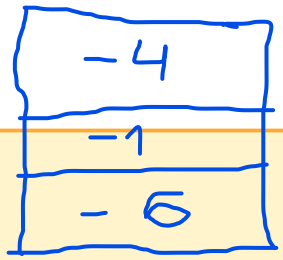
Entrada de ejemplo

```
8
6 1 4 0 0 5 -5 -7
5
-2 4 5 0 3
1
-5
1
1
```

Salida de ejemplo

```
6 1 4 -4 -1 5 -5 -6
4 5 -5 3 -3 -4

1 -1
```



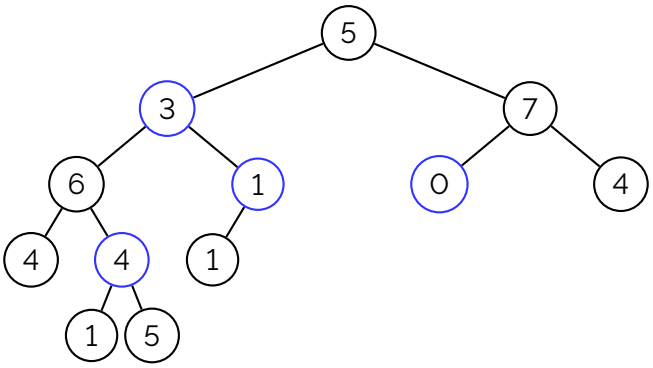
if(it==0){
it=top();
++it;
}

Correcto → if(it=top){
pop;
++it;

Incorrecto → else{
it=erase(it);
}

Ejercicio 2. Número de nodos intermedios de un árbol binario (2 puntos)

Dado un árbol binario no vacío de números enteros mayores o iguales que cero, decimos que un nodo es *intermedio* si su valor coincide con la diferencia (en valor absoluto) entre las sumas de los descendientes izquierdos y los derechos módulo el valor del nodo padre. Si un nodo no tiene ningún hijo izquierdo, asumimos que la suma de sus descendientes izquierdos es 0, y análogamente para los derechos. Además, la raíz de un árbol no será nunca un nodo intermedio, porque no tiene padre.



En el árbol anterior solo los nodos con borde azul son intermedios. Por ejemplo, el nodo 3 es intermedio puesto que se cumple $3 = |20 - 2| \ mod\ 5$, donde 20 es el resultado de la suma de todos sus nodos izquierdos, 2 la suma de todos sus nodos derechos y 5 es el valor de su nodo padre. Análogamente, el nodo 0 es intermedio ya que cumple $0 = |0 - 0| \ mod\ 7$.

Por otro lado, el nodo 7 (la raíz del hijo derecho del árbol) no es intermedio ya que $7 \neq |0 - 4| \ mod\ 5$. Observamos también que si el padre de un nodo tiene valor 0, ese nodo no puede ser intermedio (ya que $x \ mod\ 0$ no está definido para ningún x).

Dado un árbol binario queremos averiguar el número de nodos intermedios que tiene. En el ejemplo anterior el número de nodos intermedios es 4.

En este ejercicio se pide que se escriba una función recursiva que, dado un árbol binario, devuelva el número de nodos intermedios que tiene.

Entrada

La entrada comienza con un número que indica el número de casos de prueba que vienen a continuación. Cada caso de prueba consiste en una línea con la descripción de un árbol binario de enteros (todos ellos mayores o iguales que cero) mediante la siguiente notación inorden: el árbol vacío se representa mediante `.` y el árbol no vacío mediante `(iz x dr)`, siendo `x` la raíz, e `iz`, `dr` las representaciones de ambos hijos.

Salida

Para cada árbol se escribirá una línea con el número de nodos intermedios que tiene.

Entrada de ejemplo

```
5
(. 0 .)
(((. 6 .) 3 (. 3 .)) 5 .)
((((. 4 .) 6 ((. 1 .) 4 (. 5 .))) 3 ((. 1 .) 1 .)) 5 ((. 0 .) 7 (. 4 .)))
((. 3 .) 2 (. 1.))
.
```

Salida de ejemplo

```
0
1
4
0
0
```

Ejercicio 3. Trenes en Ferrovistán (3 puntos)

En la república de Ferrovistán quieren hacer honor a su nombre construyendo una amplia red de trenes. Han decidido comenzar construyendo las vías de los trenes, formando líneas ferroviarias, para luego construir estaciones sobre la red de vías. Una misma estación puede dar servicio a varias líneas, para que los viajeros puedan hacer transbordo de una línea a otra.

Se pide implementar un TAD `Ferrovistan` con las operaciones que se describen a continuación. Todas las excepciones descritas son de la clase `domain_error`.

- `void nueva_linea(const string &nombre)`

  Añade una nueva línea al sistema ferroviario. El parámetro indica el nombre que tendrá la línea. Si ya existe una línea con ese nombre, se lanzará una excepción con el mensaje `Linea existente`.

- `void nueva_estacion(const string &linea, const string &nombre, int posicion)`

  Añade una nueva estación para que dé servicio a una `linea` dada. La estación tendrá el `nombre` indicado como segundo parámetro. El tercer parámetro (`posicion`) es la distancia que hay entre la cabecera de la línea y la estación. Si la línea no existe, se lanzará una excepción con mensaje `Linea no existente`. Si la línea pasada como parámetro ya tiene una estación con el `nombre` dado, se lanzará una excepción con mensaje `Estacion duplicada en linea`. No obstante, sí se permite tener una misma estación dando servicio a varias líneas. Si la línea ya tenía una estación en la `posicion` dada, se lanzará una excepción con el mensaje `Posicion ocupada`.

- `void eliminar_estacion(const string &estacion)`

  Elimina de la red ferroviaria la `estacion` pasada como parámetro. Esta estación dejará de dar servicio a todas las líneas en las que se encuentra. Si la estación no existe en la red, se debe lanzar una excepción con mensaje `Estacion no existente`.

- `vector<string> lineas_de(const string &estacion) const`

  Devuelve una lista de las líneas a las que da servicio la `estacion` dada. La lista devuelta debe estar ordenada de manera ascendente. Si la `estacion` no existe, se debe lanzar una excepción con el mensaje `Estacion no existente`.

- `string proxima_estacion(const string &linea, const string &estacion) const`

  Devuelve el nombre de la próxima estación que se encontraría un tren que saliese de la `estacion` dada al circular por la `linea` pasada como parámetro, suponiendo que el tren va en dirección opuesta a la cabecera de la línea. Si la `linea` no existe, se lanzará una excepción con el mensaje `Linea no existente`. Si la línea no contiene una estación con el nombre dado, se lanzará una excepción con el mensaje `Estacion no existente`. Si la `estacion` pasada como parámetro es la última de la línea, se lanzará una excepción con el mensaje `Fin de trayecto`.

Recuerda que:

1. La implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD, e indicar el coste de las operaciones.
2. 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 varios 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. La palabra `FIN` en una línea indica el final de cada caso.

Los nombres de las líneas y las estaciones son cadenas de caracteres sin espacios en blanco.

Salida

Las operaciones `nueva_linea`, `nueva_estacion` y `eliminar_estacion` no producen salida, salvo en caso de error. Con respecto a las restantes:

- Tras llamar a la operación `lineas_de XXX` debe imprimirse una línea con el texto `Lineas de XXX:` seguida de los nombres de las líneas devueltas, separados por espacios.
- Tras llamar a la operación `proxima_estacion` debe imprimirse una línea con el nombre de la estación devuelta por el método.

Si una operación produce un error, entonces se escribirá una línea con `ERROR:`, seguido del error que devuelve la operación, y no se escribirá nada más para esa operación.

Cada caso termina con una línea con tres guiones (–).

Entrada de ejemplo

```
nueva_linea A
nueva_linea B
nueva_estacion A Sevistan 10
nueva_estacion A Cadistan 50
nueva_estacion B Malaguistan 30
nueva_linea C
nueva_estacion C Sevistan 10
nueva_estacion C Malaguistan 20
nueva_estacion C Cordobistan 40
lineas_de Sevistan
proxima_estacion A Sevistan
proxima_estacion B Sevistan
proxima_estacion C Sevistan
eliminar_estacion Malaguistan
proxima_estacion C Sevistan
FIN
nueva_linea A
nueva_linea A
nueva_linea B
nueva_estacion A Almeristan 10
nueva_estacion A Almeristan 20
nueva_estacion A Granadistan 10
nueva_estacion A Almeristan 10
eliminar_estacion Inexistan
lineas_de Inexistan
proxima_estacion A Almeristan
proxima_estacion B Almeristan
proxima_estacion C Almeristan
proxima_estacion A Inexistan
FIN
```

Salida de ejemplo

```
Lineas de Sevistan: A C
Cadistan
ERROR: Estacion no existente
Malaguistan
Cordobistan
---
ERROR: Linea existente
ERROR: Estacion duplicada en linea
ERROR: Posicion ocupada
ERROR: Estacion duplicada en linea
ERROR: Estacion no existente
ERROR: Estacion no existente
ERROR: Fin de trayecto
ERROR: Estacion no existente
ERROR: Linea no existente
ERROR: Estacion no existente
---
```