C02: Mezcla de dos listas ordenadas

Estructuras de Datos
Facultad de Informática - UCM

Este ejercicio consta de una entrega obligatoria y una opcional:

- **Entrega obligatoria**. Corresponde a los apartados 1, 2, 3 y 4 de este enunciado. Debe entregarse en el problema *DOMjudge* que tiene identificador `C02-1` antes del **12 de febrero a las 23:55**.
- **Entrega opcional**. Corresponde a los apartados 5, 6, 7 y 8 de este enunciado. En caso de querer hacerla, debe entregarse en el problema *DOMjudge* que tiene identificador `C02-2`. La fecha tope es la misma que la de la entrega obligatoria.

Cada entrega consiste en un único fichero `.cpp` que se subirá a *DOMjudge*. Podéis subir tantos intentos como queráis. Se tendrá en cuenta el último intento con el veredicto `CORRECT` que se haya realizado antes de la hora de entrega.

No olvidéis poner el nombre de los componentes del grupo en cada fichero `.cpp` que entregéis. Solo es necesario que uno de los componentes del grupo realice la entrega.

Las preguntas relativas a costes deben responderse como comentarios en el fichero `.cpp`.

Evaluación: La entrega obligatoria se puntuará de 0 a 10. Para poder obtener una calificación superior a 0 es necesario obtener un veredicto `CORRECT`. La entrega opcional puede otorgar 2 puntos adicionales a la calificación obtenida en la entrega obligatoria. En este último caso, si al sumar esta bonificación se obtiene un resultado superior a 10, la calificación será 10.

Partimos de la siguiente clase que define el TAD de las listas de enteros, implementada mediante listas enlazadas simples:

```
class ListLinkedSingle {
public:
  ListLinkedSingle();
  ~ListLinkedSingle();
  ListLinkedSingle(const ListLinkedSingle &other);

  void push_front(const int &elem);
  void push_back(const int &elem);
  void pop_front();
  void pop_back();

  int size() const;
  bool empty() const;
  const int & front() const;
  int & front();
  const int & back() const;
  int & back();
  const int & at(int index) const;
  int & at(int index);
  void display() const;

private:
```

```
  // ...
};
```

El presente ejercicio tiene como objetivo *mezclar* los elementos de dos listas ordenadas en una sola, de modo que la lista resultado también esté ordenada. Por ejemplo, a partir de las listas $l_1 = [2, 7, 10, 14]$ y $l_2 = [1, 9, 12, 14, 20]$ queremos obtener la lista $[1, 2, 7, 9, 10, 12, 14, 14, 20]$.

Realizaremos dos versiones de este ejercicio, la segunda de ellas opcional.

- La primera de ellas accede directamente a los nodos de ambas listas y manipula los punteros `next` para que los nodos acaben formando una única lista ordenada de manera ascendente. Esta versión se implementa como un método de la clase `ListLinkedSingle`.
- La segunda versión utiliza únicamente métodos de la interfaz pública de `ListLinkedSingle`, sin acceder a atributos privados y sin manipular directamente nodos. Esta versión se implementa como una función aparte, ajena a los métodos de la clase `ListLinkedSingle`.

Procedemos del siguiente modo:

1. En el Campus Virtual encontrarás un fichero `.cpp` de plantilla. Modifica este fichero implementando el siguiente método en la clase `ListLinkedSingle`:

   ```
   void merge(ListLinkedSingle &l2);
   ```

   Este método modifica la lista `this` (que ya se asume ordenada) insertando todos los elementos de la lista `l2` recibida como parámetro, de modo que `this` siga estando ordenada, y `l2` quede vacía. Por ejemplo, si `l1` representa la lista $[1, 5, 7, 10]$, `l2` representa la lista $[3, 7, 9]$ y hacemos lo siguiente:

   ```
   l1.merge(l2);
   ```

   Al final de la llamada al método `merge`, `l1` representará la lista $[1, 3, 5, 7, 7, 9, 10]$ y `l2` representará la lista vacía.

   Importante: Al ser un método de la clase `ListLinkedSingle`, puedes (y debes) acceder a los atributos privados de esta clase. Debes modificar los atributos `next` de los nodos de ambas listas para que al final queden conectados en una única lista ascendente. **No está permitido crear nuevos nodos** mediante `new`, ya sea directamente, ya sea mediante llamadas a otros métodos de `ListLinkedSingle`.

2. Indica el coste en tiempo, en el caso peor, del método `merge`.

3. Escribe un programa que lea de la entrada varios casos de prueba, cada uno consistente en un par de listas, e imprima por pantalla el resultado de mezclar ambas listas. El formato de la entrada y salida se describen al final de este enunciado.

4. Entrega el resultado de todos los apartados realizados hasta ahora en el problema de *DOMjudge* con identificador `C02-1`.

5. Dentro del mismo fichero plantilla, implementa una función `merge_lists`

   ```
   ListLinkedSingle merge_lists(const ListLinkedSingle &l1, const ListLinkedSingle &l2)
   ```

   que, dadas dos listas ordenadas `l1` y `l2`, devuelva el resultado de mezclarlas, de modo que el resultado sea una lista ordenada que contenga los elementos de `l1` y `l2`. Al ser `merge_lists` una función externa a la clase `ListLinkedSingle`, solo podrá acceder a los métodos públicos de `l1` y `l2`. No se pueden modificar las listas `l1` ni `l2`, pero sí se pueden realizar copias de las mismas.

   Importante: Independientemente del veredicto del juez, no se admitirán soluciones de coste cuadrático con respecto a la longitud de alguna de las listas.

6. Indica el coste en tiempo, en el caso peor, de la función `merge_lists`.
7. Modifica el programa del apartado 3 para que mezcle las listas recibidas por la entrada mediante el método `merge` del apartado 1, en lugar de utilizar la función `merge_lists`.
8. Entrega el resultado en el problema de *DOMjudge* con identificador `C02-2`.

Entrada

La entrada comienza con un `int` que indica el número de casos de prueba que vienen a continuación. Cada caso de prueba consiste en dos lineas, describiendo sendas listas de entrada. Cada lista se representa mediante una serie de números que pueden estar comprendidos entre `1` y `5000`, finalizando con un `0` que no se considera parte de la lista. Todas las listas de la entrada están ordenadas de manera ascendente.

Salida

Para cada caso se escribirá una línea con la lista que resulta de mezclar las dos listas de entrada. El formato de cada lista en la salida es el mismo que el que produce el método `display()`, explicado en la teoría.

Entrada de ejemplo

```
6
1 3 5 7 0
2 4 6 8 0
2 4 6 8 0
1 3 5 7 0
1 2 3 0
6 7 8 0
1 10 20 0
2 6 8 12 20 22 0
0
1 2 3 0
1 1 2 2 0
0
```

Salida de ejemplo

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 7, 8]
[1, 2, 6, 8, 10, 12, 20, 20, 22]
[1, 2, 3]
[1, 1, 2, 2]
```

Créditos

Adaptación del problema *Mezclar listas enlazadas ordenadas* de Alberto Verdejo.