Entramar dos listas doblemente enlazadas circulares

Estructuras de Datos
Facultad de Informática - UCM

	Nombre y apellidos de los/as componentes del grupo	ID juez
1		
2		
3		
4		

Entramar dos listas `xs` y `zs` de igual longitud consiste en fusionarlas de modo que los elementos de `zs` se intercalan con los de `xs`. Por ejemplo, al entramar las listas `[10, 20, 30]` y `[4, 7, 9]` se obtiene como resultado `[10, 4, 20, 7, 30, 9]`.

En este ejercicio partimos de la clase `ListLinkedDouble`, que implementa el TAD lista mediante listas doblemente enlazadas circulares con nodo fantasma. Queremos añadir un nuevo método, llamado `zip()`:

```
class ListLinkedDouble {
private:
  struct Node {
    int value;
    Node *next, *prev;
  };
  Node *head;   // Nodo fantasma
  int num_elems;
  ...
};
```

1. Implementa los siguientes métodos privados en la clase `ListLinkedDouble`:

   ```
   void detach(Node *n);
   void attach(Node *n, Node *position);
   ```

   El método `detach` desengancha de la lista enlazada el nodo `n` pasado como parámetro, sin liberarlo del *heap*. El método `attach` engancha el nodo `n` en la lista enlazada, justo después del nodo apuntado por `position`.

2. Implementa el método `zip()` con la siguiente especificación:

   ```
   {  this = [x_1, x_2, ..., x_n], other = [y_1, y_2, ..., y_n]  }
   void ListLinkedDouble::zip(ListLinkedDouble &other);
   {  this = [x_1, y_1, x_2, y_2, ..., x_n, y_n], other = []  }
   ```

   El método `zip()` entrama las listas `this` y `other`, dejando el resultado en `this`, y dejando la lista `other` vacía. Por ejemplo, si `xs = [1, 2, 3]` y `zs = [10, 20, 30]`, tras hacer `xs.zip(zs)` tenemos que `xs = [1, 10, 2, 20, 3, 30]` y `zs = []`.

   Para implementar este método haz uso de los métodos `attach()` y `detach()` del apartado anterior.

   Importante: Para la implementación del método no pueden crearse, directa o indirectamente, nuevos nodos mediante `new` ni borrar nodos mediante `delete`; han de reutilizarse los nodos de las listas de entrada. Tampoco se permite copiar valores de un nodo a otro.

3. Indica y justifica el coste de los métodos `attach()`, `detach()` y `zip()`.

Solución

```
void ListLinkedDouble::detach(Node *node) {
  node->prev->next = node->next;
  node->next->prev = node->prev;
}

void ListLinkedDouble::attach(Node *node, Node *position) {
  node->next = position->next;
  node->prev = position;
  position->next->prev = node;
  position->next = node;
}


void ListLinkedDouble::zip(ListLinkedDouble &other) {
  // Punteros a sendas listas
  Node *cur_this = head->next;
  Node *cur_other = other.head->next;

  // Mientras nos queden elementos de this
  while (cur_this != head) {
    // Guardamos temporalmente los nodos siguientes a cur y other,
    // que seran los cur y other de la siguiente iteracion
    Node *sig_this = cur_this->next;
    Node *sig_other = cur_other->next;
    // Quitamos el de la lista other y lo anyadimos despues
    // del puntero cur_this (dado que cur_this->next == sig_this)
    detach(cur_other);
    attach(cur_other, cur_this);
    // Avanzamos ambos punteros
    cur_this = sig_this;
    cur_other = sig_other;
  }

  // Actualizamos num_elems
  num_elems += other.num_elems;
  other.num_elems = 0;
}
```

Los métodos `attach()` y `detach()` tienen coste constante ($O(1)$). El método `zip` tiene un bucle que realiza tantas iteraciones como elementos tiene la lista `this`. Dentro de ese bucle solo se hacen operaciones de coste constante. Por tanto, el coste de `zip` es $O(N)$, donde N es la longitud de la lista `this` (y también de la lista `other`, pues se supone que ambas tienen la misma longitud).