Dividir una lista doblemente enlazada circular

Estructuras de Datos
Facultad de Informática - UCM

Supongamos una implementación del TAD Lista mediante listas doblemente enlazadas circulares con nodo fantasma:

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

Escribe un método `split` con la siguiente especificación:

```
{  l1 = [x_0, ..., x_{n-1}], 0 ≤ i ≤ n  }
l2 = l1.split(i);
{  l1 = [x_0, ..., x_{i-1}], l2 = [x_i, ..., x_{n-1}]  }
```

El método `split` recibe un número i comprendido entre `0` y `l1.num_elems`, y extrae los nodos de la lista `this` que se encuentren a partir de la posición `i`-ésima, devolviendo otra lista con los nodos extraídos. La función debe tener coste $O(i)$, donde i es el índice de la lista a partir del cual se produce la división.

Importante: Deben reutilizarse los nodos de la lista de entrada. No es posible crear nuevos nodos mediante `new`, salvo el nodo fantasma de la lista devuelta como resultado. Este nodo fantasma se crea en el constructor de `ListLinkedDouble`.

Solución

```
ListLinkedDouble split(int n) {
  assert (n >= 0);
  int i = 0;
  Node *current = head->next;
  while (i < n && current != head) {
    i++;
    current = current->next;
  }
  // Tambien valdria: current = nth_node(n); en lugar del bucle

  Node *old_last = head->prev;
  Node *new_last = current->prev;
  ListLinkedDouble result;
  if (current != head) {
    result.head->next = current;
    result.head->prev = old_last;
    current->prev = result.head;
    old_last->next = result.head;

    result.num_elems = num_elems - n;
    num_elems = n;
  }

  new_last->next = head;
  head->prev = new_last;

  return result;
}
```