L02: Intersección de listas enlazadas simples

Estructuras de Datos
Facultad de Informática - UCM

Para realizar este ejercicio, descarga la plantilla que se proporciona en este enunciado (icono del clip al lado del título) o a través del Campus Virtual.

La entrega de este ejercicio 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 por parte de alguno de los miembros de la pareja.

No olvidéis poner el nombre de los componentes de la pareja en el fichero `.cpp` que entreguéis. Solo es necesario que uno de los componentes de la pareja realice la entrega.

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

Evaluación: La práctica se puntuará de 0 a 10, desglosada del siguiente modo:

- Si la pareja ha obtenido el veredicto `CORRECT`, se obtendrán hasta 9 puntos por la entrega, más 1 punto por la evaluación que haga de la entrega de otra pareja.
- Si la pareja no ha obtenido el veredicto `CORRECT`, pero ha realizado al menos una entrega en el juez, podrá obtener hasta 2 puntos por la evaluación que haga de las entregas de otras dos parejas.
- Si la pareja no ha realizado ninguna entrega, la calificación de la práctica será de 0.

Partimos de la clase `ListLinkedSingle`, que implementa el TAD de las listas de números enteros mediante listas enlazadas simples. Queremos añadir un nuevo método, llamado `intersect()`:

```
class ListLinkedSingle {
private:
  struct Node {
    int value;
    Node *next;
  };
  Node *head;

public:
  ...
  void intersect(const ListLinkedSingle &other);
};
```

Este método calcula la intersección entre la lista `this` y la lista `other`, guardando el resultado en `this`. En otras palabras, elimina de `this` aquellos elementos que no se encuentren en `other`. Para realizar este ejercicio puedes suponer que tanto la lista `this` como `other` tienen sus elementos en orden creciente y no contienen duplicados.

Por ejemplo, dadas las listas `xs = [1, 3, 4, 5, 8, 9]` y `zs = [2, 4, 8, 10]`, tras la ejecución de `xs.intersect(zs)` la lista `xs` acaba con el valor `[4, 8]`. La lista `zs` no se modifica.

Se pide:

1. Implementar el método `intersect()`.
2. Indicar su coste con respecto al tamaño de las listas de entrada.

Importante: Para la implementación del método no pueden crearse, directa o indirectamente, nuevos nodos mediante `new`. Tampoco se permite copiar valores de un nodo a otro.

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 cuatro líneas. La primera línea contiene un número *N* indicando cuántos elementos tiene la lista `this`. La segunda línea contiene esos *N* elementos, separados por espacios. La tercera línea contiene un número *M* indicando cuántos elementos tiene la lista `other`. La cuarta línea contiene esos *M* elementos, separados por espacios. Puedes suponer que los elementos de ambas listas están en orden creciente, y que ninguna lista tiene elementos duplicados.

Salida

Para cada caso de prueba se imprimirá el contenido de la lista `this` tras llamar al método `intersect()`. Puedes utilizar el método `display()` de esta clase.

Entrada de ejemplo

```
2
6
1 3 4 5 8 9
4
2 4 8 10
3
1 4 6
2
2 9
```

Salida de ejemplo

```
[4, 8]
[]
```