Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software

Examen Segundo Parcial, Junio 2015

1. [3 puntos] Extiende el TAD `List` visto en clase, con una nueva operación que modifique la lista intercalando sus nodos de la siguiente forma: supongamos que los nodos de la lista de izquierda a derecha son $n_1, n_2, n_3, n_4, \ldots, n_{k-3}, n_{k-2}, n_{k-1}, n_k$, al finalizar la ejecución del método los nodos estarán colocados como $n_k, n_1, n_{k-1}, n_2, n_{k-2}, n_3, n_{k-3}, n_4 \ldots$.

Indica la complejidad de tu implementación. Esta debe ser lo más eficiente posible. Para ello, se debe evitar liberar y reservar memoria, y hacer copias de los campos. Si haces uso de métodos auxiliares, impleméntalos también.
A continuación se muestra a modo de recordatorio las partes relevantes del TAD `List`.

```
class List {
   private:
     class Nodo {
         public:
           Nodo() : _sig(NULL), _ant(NULL) {}
           Nodo(const T &elem) : _elem(elem), _sig(NULL), _ant(NULL) {}
           Nodo(Nodo *ant, const T &elem, Nodo *sig) :
             _elem(elem), _sig(sig), _ant(ant) {}

           T _elem;
           Nodo *_sig;
           Nodo *_ant;
     };
     Nodo * _prim;   Nodo* _ult;  ...
   public:
      void doblarLista();  ...
}
```

2. [3 puntos] Podemos utilizar los árboles binarios para representar los caminos en la falda de una montaña. La raíz del árbol representa la cima de la que salen una o dos rutas. Las distintas rutas según se va ensanchando la falda de la montaña se dividen en dos formando caminos que nunca se volverán a conectar. Un escalador está en la cima de la montaña (raíz del árbol) y se da cuenta de que en distintas intersecciones (marcadas en el árbol con 'X') hay amigos que necesitan su ayuda para subir. Tiene que bajar a cada una de las 'X' y ayudarles a subir de uno en uno. Implementa una función con la cabecera

en esa asignatura (o si no ha faltado nunca). Para gestionar esta información diseñan un tipo abstracto de datos *Faltas* con tres operaciones generadoras:

- *anadirAlumno*, que añade un alumno en todas las asignaturas con 0 faltas.
- *anadirFalta*, que incrementa en 1 el número de faltas de un alumno en una asignatura.
- *anadirAsignatura*, que construye una lista con los mismos alumnos de las demás asignaturas, cada uno de ellos con 0 faltas.

A la hora de implementar este TAD han decidido que la lista de faltas de una asignatura viene representada por un diccionario con clave `IdAlumno` y valor asociado el número de faltas del alumno en esa asignatura; y que todas las listas de faltas se hallan almacenadas en un diccionario con clave `IdAsignatura` (identificador de la asignatura) y valor asociado la lista de faltas de esa asignatura. El invariante de la representación incluye el hecho de que las listas de todas las asignaturas contienen exactamente los mismos alumnos:

```
class Faltas
{public :
    void anadirAlumno(const IdAlumno& a);
        //incorpora al alumno en todas las asignaturas que haya con 0 faltas
    void anadirFalta(const IdAlumno& a,const IdAsignatura& s);
        //incrementa en 1 las faltas del alumno en la asignatura.
    void anadirAsignatura(const IdAsignatura& s);
        // incorpora todos los alumnos que ya esten presentes
        // en las otras asignaturas con 0 faltas

          ...otros metodos...
 private:
     Diccionario<IdAsignatura,Diccionario<IdAlumno,int> > listas_faltas;}
```

En la reunión de fin de curso, ponen en común su información y desean añadir a este TAD las siguientes operaciones:

- *noFaltas*, que por orden alfabético (el dado sobre `IdAlumno`) devuelve una lista con todos los alumnos que no han faltado a ninguna clase en ninguna de las asignaturas.
- *totalFaltas*, que dado un alumno, devuelve el número de faltas que acumula entre todas las asignaturas.
- *maxFaltas*, que devuelve la asignatura donde mayor número de faltas hay entre todos los alumnos; si hay varias con el máximo número de faltas devuelve una cualquiera.

Se pide:

1.[0,5 ptos] Elegir justificadamente la implementación de cada uno de los dos diccionarios e indicar, en base a esta decisión, qué le exiges a los tipos `IdAsignatura` e `IdAlumno`, y cual es el coste que tendrían las operaciones generadoras arriba mencionadas.

en este apartado y el anterior implementar la generadora *anadirFalta*.