Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software (grupo A)

Examen Segundo Cuatrimestre, 3 de Junio de 2014.

1. (3 puntos) Extiende el TAD `Lista` visto en clase con una nueva operación cuya cabecera en C++ es:

```
void inserta(const T &elem, int pos);
```

que inserta el elemento `elem` en la posición `pos`, de forma que cuando `pos` es 0 se añade *al principio* de la lista (operación `Cons`), mientras que cuando es igual al número de elementos de la lista, se insertará al final (como `ponDr`). Indica la complejidad de la operación.

Si llamas a otros métodos, *debes implementarlos también*.

A continuación aparece, a modo de recordatorio, las partes relevantes del TAD `Lista`:

```
template <class T>
class Lista {
   ...
private:
   class Nodo {
      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, *_ult;

   unsigned int _numElems;
};
```

2. (3 puntos) Dado un árbol binario, en cuya raíz se encuentra situado un tesoro y cuyos nodos internos pueden contener un dragón o no contener nada, se pide diseñar un algoritmo que nos indique la hoja del árbol cuyo camino hasta la raíz tenga el menor número de dragones. En caso de que existan varios caminos con el mismo número de dragones, el algoritmo devolverá el que se encuentre más a la izquierda de todos ellos.

Para ello implementar una función que reciba un árbol binario cuyos nodos almacenan cadenas de caracteres:

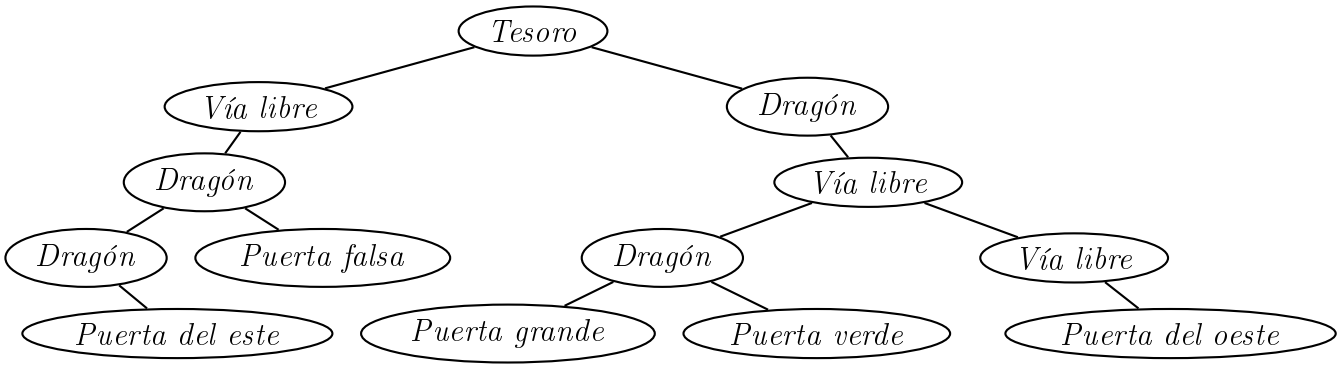
- La raíz tiene la cadena *Tesoro*

- Los nodos internos tienen la cadena *Dragón* para indicar que en el nodo hay un dragón o la cadena *Vía libre* para indicar que no hay dragón.
- En cada hoja se almacena un identificador que no puede estar repetido.

y devuelva el identificador de la hoja del camino seleccionado. El árbol tiene como mínimo un nodo raíz y un nodo hoja diferente de la raíz. La operación no se implementa como parte de ningún TAD.

El coste de la operación implementada debe ser $\mathcal{O}(n)$.

Por ejemplo, dado el siguiente árbol el algoritmo devolverá la cadena de caracteres *Puerta falsa*.



3. (4 puntos) Eres un prestigioso diseñador de videojuegos, y estás trabajando ahora en uno llamado *EspacioMatic*, en el que habrá naves espaciales con distintos módulos (motores, cabinas, escudos, láseres, etcétera). Necesitarás implementar, como poco, las siguientes operaciones:

- *EspacioMatic*: inicializa el sistema de juego.
- *nuevaNave*: añade una nueva nave espacial (vacía) al sistema, con el identificador numérico proporcionado. Si se intenta usar un identificador ya existente produce error. No devuelve nada.
- *equipaNave*: dado el identificador de una nave, un nombre de módulo (una cadena como "motor", "cabina", "láser", etcétera) y un nivel de funcionalidad (un entero ≥ 1), añade el módulo correspondiente a esa nave con el nivel indicado. Si esa nave ya tenía ese módulo, se suma el nuevo nivel al anterior (esto permite reparar módulos de naves). No devuelve nada.
- *estropeaNave*: dado el identificador de una nave, y un nombre de módulo, resta 1 al nivel de ese módulo en esa nave (asumiendo que tuviese un nivel > 0). Devuelve `true` si el módulo existía y tenía un nivel positivo antes de hacer la resta, ó `false` si ha sido imposible restar nivel ya que el módulo no existía o estaba ya a 0.
- *navesDefectuosas*: devuelve una lista de identificadores de naves que tienen uno o más módulos completamente estropeados (con un nivel de 0).
- *modulosNave*: dado el identificador de una nave, devuelve una lista con los nombres de los módulos que tiene equipados (tengan el nivel que tengan), ordenada alfabéticamente.

Desarrolla en C++ una implementación de la clase *EspacioMatic* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones. En cada operación, indica también, de forma razonada, su complejidad.