Estructuras de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software

Examen Segundo Parcial, Septiembre 2015

1. [2 ptos] Extiende la implementación de las listas añadiendo una operación `invierte()` que invierta la lista de forma que el primer elemento pase a ser el último, el segundo el penúltimo y así sucesivamente. No debes utilizar memoria auxiliar (está permitido usar variables locales de tipo puntero siempre y cuando no se hagan `new`'s, pero por ejemplo no pueden usarse variables de tipo `T`).

No se admitirán soluciones recursivas.

A continuación se muestra a modo de recordatorio las partes relevantes del TAD `List`.

```
template <typename T>
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 invierte();  ...
}
```

2. [2,5 ptos] Implementa la siguiente función:

```
template <typename T>
bool coinciden(const Arbin<T> &a, const List<T> &pre);
```

que diga si la lista `pre` coincide con el recorrido en preorden del árbol `a`. No puedes hacer uso de estructuras de datos auxiliares (arrays, pilas, colas, listas, árboles etc.) ni pedir/liberar memoria adicional (hacer `new` o `delete`).

3. [1,5 ptos] Implementa la siguiente función:

```
template <typename T>
void invierteBase(Stack<T> &p, int n);
```

que reciba la pila `p` y un valor entero `n` ($0 \leq n \leq p.size()$) y modifique `p`, de forma que los n valores de la cima queden en el orden que están y los $p.size() - n$ valores del fondo de la pila queden en orden inverso al de entrada.

En la implementación de la función deben utilizarse únicamente TADs vistos en clase, se valorará que se seleccionen los TADs más apropiados para la resolución del problema. No se pueden utilizar los arrays de C++. Se valorará la eficiencia de la solución propuesta.

4. [4 ptos] El periódico local quiere poner orden en su base de datos de noticias. Buscan poder relacionar rápidamente las noticias que se quieren publicar con noticias ya publicadas anteriormente para dar una información más completa a los lectores. Para ello quieren implementar un TAD que les permita manejar las noticas publicadas, bien obteniendo las últimas noticias que se han publicado sobre un tema o bien, refinando un poco más, obtener las últimas noticias sobre un tema que además contengan una cierta palabra. Como sólo son interesantes algunas palabras sobre cada tema, los periodistas pueden añadir al sistema las palabras de cada tema que consideren importantes. Las palabras interesantes pueden añadirse en cualquier momento y borrarse cuando ya no sean interesantes.

Las operaciones mínimas que se requieren son:

- *nuevaNoticia*: Añade una noticia sobre un cierto tema. Si el tema no existe en el sistema, se añade. Se garantiza que las noticias se añaden en el orden en que se producen, es decir las llamadas a la operación *nuevaNoticia* se realizarán por orden de publicación de las noticias.
- *ultimasNoticias*: Obtiene las n últimas noticias dadas de alta en el sistema sobre un cierto tema. La primera noticia de la lista debe ser la última que se ha producido. Si hay menos de n noticias sobre este tema en el sistema, devuelve todas ellas.
- *noticiasTermino*: Obtiene las últimas n noticias de un tema en que aparece una cierta palabra. La primera noticia de la lista debe ser la última que se ha producido. Si hay menos de n noticias, devuelve todas ellas.
- *nuevoTerminoBusqueda*: Crea un nuevo término de búsqueda para un cierto tema. Si el término ya existe la operación no tiene efecto.
- *eliminaTerminoBusqueda*: Elimina un término de búsqueda, pero no las noticias en las que aparece el término. Si el término no estaba dado de alta la operación no tiene efecto.
- *listarTerminos*: Obtiene una lista ordenada alfabéticamente con todos los términos definidos para un cierto tema.

Se supone que existe una clase *Noticia* que nos da acceso al contenido de una cierta noticia. Esta clase cuenta con un operación `bool esta (const string & p) const;` que dada una palabra nos dice si pertenece a la noticia.

Desarrolla en C++ una implementación de la clase *NoticiasPublicadas* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones. Indica qué le exiges a los tipos que almacenan noticias, términos y temas. Determina de forma razonada cuál es la complejidad de cada operación.