Práctica 6. Contenedores asociativos de STL

Sesiones de prácticas: 1

Objetivos

Manejar contenedores asociativos de STL. Programa de prueba para comprobar su correcto funcionamiento y ampliación del diseño de texto predictivo.

Descripción de la EEDD

La relación entre Diccionario y Palabra ha ido cambiando y en la última implementación el `AVL<T>` permitió mejorar el tiempo del proceso de búsqueda. En esta ocasión dicho `AVL` se va a sustituir por un mapa de STL. Este mapa tendrá como clave el término de tipo String y como dato objetos de tipo Palabra. El resto de la funcionalidad del problema se va cambiar un poco añadiendo a Usuario.

Diseño: Personalizar el texto predictivo para cada usuario.

El diseño anterior manejaba un solo diccionario alimentado por el único usuario del sistema. Este diseño se modifica ahora ligeramente para añadir diferentes usuarios dentro del sistema de Texto predictivo. En esta nueva versión el Diccionario original de palabras es `dicBase`, al cual no se le van a añadir nuevas palabras, aunque sí nuevos sucesores siempre que también el sucesor esté dentro del diccionario. De este modo, el diccionario no se ve contaminado con términos ajenos, aunque éstos sean válidos.

Las palabras que no estén en el diccionario general y sus sucesores correspondientes se añadirán a un nuevo diccionario particular de cada usuario. Este diccionario por tanto aparecerá vacío inicialmente, y se irá nutriendo de las palabras y expresiones que tenga cada usuario particular. Por lo demás la funcionalidad es similar.

Clase Usuario:

El usuario introduce frases mediante la función `escribeFrase()` y de este modo entrena el sistema de la siguiente forma:

- descompone la frase paulatinamente en parejas de términos (`término1`, `término2`) y se los pasa en primer lugar a `TextoPredictivo:entrena()`

- Si `término1` y `término2` están ambos en el diccionario base, entonces `TextoPredictivo:entrena(): true` y se añade el `término2` como sucesor
- Pero si alguno de ellos no forma parte de dicho diccionario, entonces el proceso se realiza pero con el diccionario particular del usuario `miDicc::inserta()`[1].
- El funcionamiento de `miDicc` es por tanto parecido a cómo funcionaba el diccionario general en las prácticas anteriores, ya que añade palabras nuevas y sucesores en cuanto alguno de ellos no pertenezcan al diccionario general.

Por ejemplo el entrenamiento de la frase siguiente (eliminando signos de puntuación):

Hola colega, como andas? Yo regu porque yo estoy pillao con las prácticas de estructuras.

Hola colega -> se añade "colega" como sucesor de "Hola" en dicBase

colega como -> se añade "como" como sucesor de "colega" en dicBase

como andas -> se añade "como" a miDicc y "andas" como sucesor de "como" en miDicc

andas yo -> se añade "andas" a miDicc y "yo" como sucesor de "andas" en miDicc

yo regu -> se añade "yo" a miDicc y "regu" como sucesor de "yo" en miDicc

regu porque -> se añade "regu" a miDicc y "porque" como sucesor de "regu" en miDicc

porque yo -> se añade "yo" como sucesor de "porque" en dicBase

yo estoy -> se añade "estoy" como sucesor de "yo" en miDicc porque ya existía "yo"

...

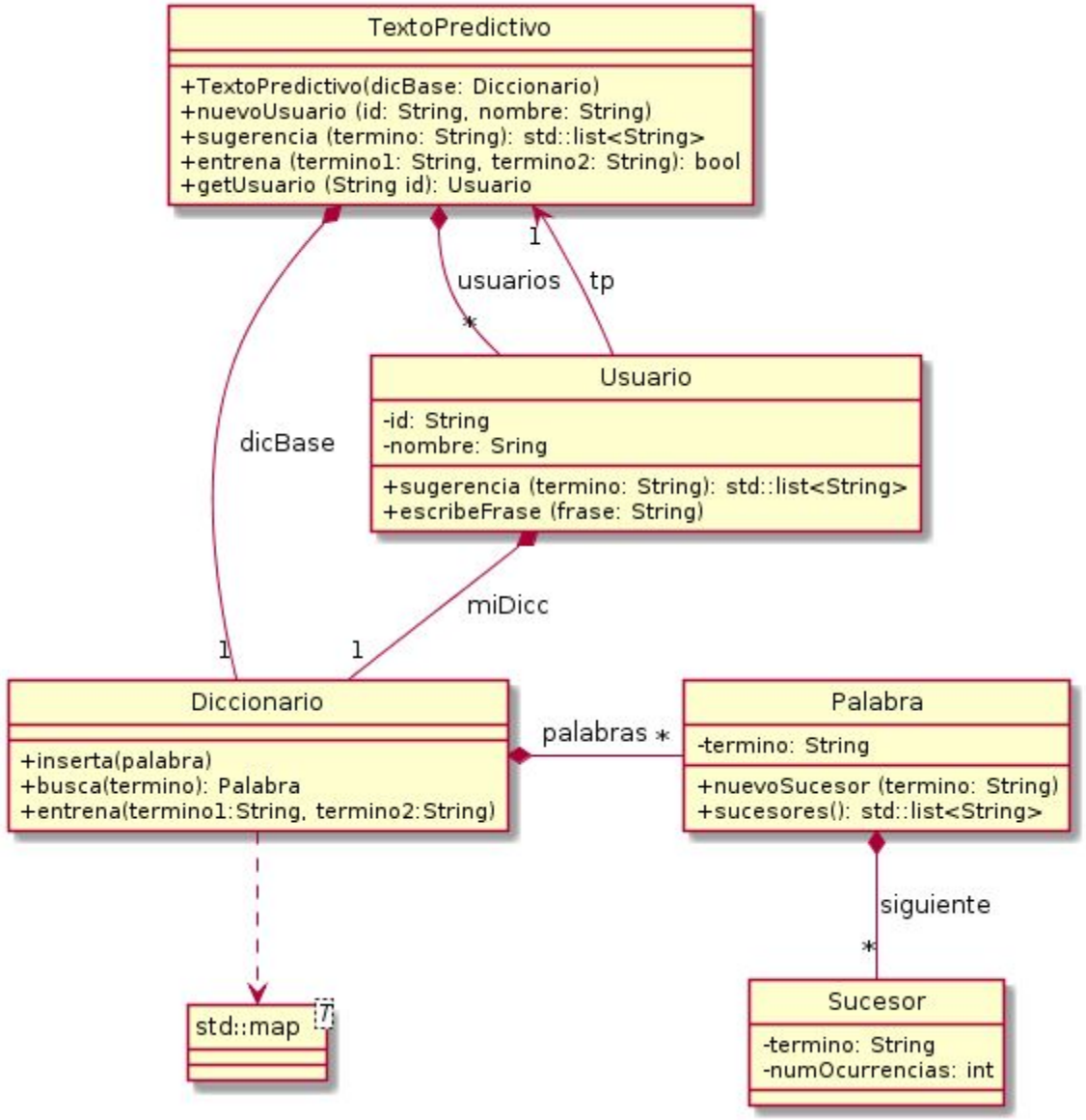
Al usuario también se le mostrarán sugerencias a las palabras con la función `Usuario::sugerencia()` según trabaje la interfaz de la aplicación. Pero en esta ocasión, la palabra que introduzca puede estar en ninguno, uno o dos diccionarios.

- Si no existe el término en ambos diccionarios se llama `miDic::inserta()` según se ha explicado anteriormente y no se muestran sugerencias
- Si el término sólo existe en uno de los dos diccionarios entonces se muestran los sucesores de ese diccionario de forma habitual (hasta 10)
- Si termino estuviera en ambos diccionarios, lo cual puede pasar, al usuario se le muestran en primer lugar los sucesores más prioritarios de su diccionario (hasta 5) y en segundo lugar los del diccionario general (hasta 5).

Programa de prueba:

[1] La funcionalidad de entrenar un diccionario a partir de una frase se pasa ahora a la clase Usuario, mientras que el método `Diccionario::entrena` se especializa para trabajar únicamente con dos términos.

Crear un programa de prueba de texto predictivo con el diccionario a partir del diccionario base y crear dos usuarios. Entrenar a ambos usuarios con porciones diferentes del corpus y con frases conocidas para ver las diferentes sugerencias en cada caso.



Un posible programa de prueba sería:

```
int main (){

    Diccionario dic (“diccionario.txt”);

    TextoPredictivo tp (dic);

    tp.nuevoUsuario ("usr1", "Jose Pérez");

    tp.nuevoUsuario ("usr2", "Lola Gutierrez");

    Usuario *us1 = tp.getUsuario ("usr1");
```

```
   us1 -> escribeFrase ("Hola colega como andas Yo regu porque yo estoy pillao con las
prácticas de estructuras");

    list<String> lus1 = us1 -> sugerencia ("yo");



    Usuario *us2 = tp.getUsuario ("usr2");

    us2 -> escribeFrase (" el whatsapp el colega el tuit el pillao  ")

    list<String> lus2 = us2 -> sugerencia ("el");

}
```

Estilo y requerimientos del código:

1. El código debe ser claro, tener un estilo definido y estar perfectamente indentado, para ello se pueden seguir algunos de los estilos preestablecidos para el lenguaje C++ (http://geosoft.no/development/cppstyle.html).

2. Deben comprobarse todas los posibles errores y situaciones de riesgo que puedan ocurrir (desbordamientos de memoria, parámetros con valores no válidos, etc.) y lanzar las excepciones correspondientes, siempre que tenga sentido. Leer el tutorial de excepciones disponible en el repositorio de la asignatura en docencia virtual.

3. Se valorará positivamente la calidad general del código: claridad, estilo, ausencia de redundancias, etc.