PRG (ETS de Ingeniería Informática) - Curso 2017-2018

Práctica 5. Implementación y uso de estructuras de datos lineales

(3 sesiones)

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Contexto y trabajo previo

En el contexto académico, esta práctica corresponde al *"Tema 5. Estructuras de datos lineales"*. Los objetivos detallados de la práctica son los siguientes:

- Trabajar con elementos habituales en el uso de la memoria enlazada: referencias, nodos, etc.
- Implementar una clase conjunto de `String` usando una lista o secuencia enlazada ordenada, poniendo especial énfasis en las operaciones de inserción, búsqueda y eliminación de elementos, así como la intersección y unión de conjuntos.
- Usar la clase conjunto anterior para la resolución de problemas de análisis y comparación de las palabras que aparecen en una serie de textos.

Antes de la sesión de laboratorio, has de leer el boletín de prácticas tratando de resolver, en la medida de lo posible, los problemas propuestos.

2. Planteamiento del problema

Se desea desarrollar una clase `SetString` cuyos objetos representen conjuntos de cadenas de caracteres o `String`.

Esta clase se puede usar en la resolución de ciertos problemas de comparación de textos. Por ejemplo, dados dos ficheros de texto, obtener el conjunto de palabras común a ambos textos, o el conjunto unión de las palabras de ambos textos.

Los métodos de la clase corresponderán a las operaciones básicas de conjuntos, y sus perfiles y especificación vendrán dados por:

```
/** Crea un conjunto vacio. */
public SetString()

/** Inserta s en el conjunto.
 *  Si s ya pertenece al conjunto, el conjunto no cambia.
 *  @param s String. Elemento que se inserta en el conjunto.
 */
public void add(String s)

/** Comprueba si s pertenece al conjunto.
 *  @param s String.
 *  @return true sii s pertenece al conjunto.
 */
 public boolean contains(String s)

/** Elimina s del conjunto.
 *  Si s no pertenece al conjunto, el conjunto no cambia.
 *  @param s String.
 */
public void remove(String s)

/** Devuelve la talla o cardinal del conjunto.
 *  @return la talla del conjunto.
 */
public int size()

/** Devuelve el conjunto interseccion del conjunto y del otro conjunto.
 *  @param other SetString.
 *  @return el conjunto interseccion.
 */
public SetString intersection(SetString other)

/** Devuelve el conjunto union del conjunto y del otro conjunto.
 *  @param other SetString.
 *  @return el conjunto union.
 */
public SetString union(SetString other)
```

Una representación posible consiste en que cada objeto de la clase almacene en una secuencia enlazada los elementos que forman parte del conjunto. Esta es la estructura de datos que se va a desarrollar en esta práctica. En la asignatura de *Estructuras de Datos y Algoritmos* de segundo curso se presentarán otras representaciones más especializadas y eficientes.

En la práctica se van a completar las siguientes tareas:

1. Desarrollo de la clase `SetString`. En concreto, las secciones 3 y 4 de esta práctica se dedican a la implementación enlazada de la clase.

2. Desarrollo de un programa de aplicación a la comparación de textos. La sección 5 se dedica a desarrollar una aplicación de la clase como la mencionada al inicio de esta sección.

Las sucesivas actividades a completar se van proponiendo a lo largo de estas secciones.

3. Implementación de la clase `NodeString`

Para poder desarrollar secuencias enlazadas de `String` se va a implementar como primera actividad una clase cuyos objetos sean los nodos de la secuencia. Cada nodo deberá tener por lo tanto un dato de tipo `String`.

Actividad 1: Atributos y constructores de la clase `NodeString`

Crear en el proyecto `prg` un paquete `pract5` específico para la práctica.

Dentro de este paquete se creará una clase `NodeString` cuyos atributos y métodos constructores serán análogos a los de la clase `NodeInt` vista en teoría, con la salvedad de que los datos serán de tipo `String`.

4. Implementación enlazada de la clase `SetString`

La información de los objetos de la clase se estructurarán en los siguientes componentes:

- Una secuencia enlazada que contenga en sus nodos los elementos del conjunto, sin elementos repetidos, y que se mantendrá ordenada ascendentemente por el orden de `String`.
- Un entero que mantenga el número de elementos en el conjunto, es decir, el cardinal del conjunto.

La ordenación de la secuencia se debe a razones de implementación de los métodos. Como se irá discutiendo en las siguientes actividades, además de favorecer la búsqueda de elementos en el conjunto, fundamentalmente permitirá aplicar estrategias eficientes al problema de la unión y la intersección.

Actividad 2: Atributos y constructores de la clase `SetString`

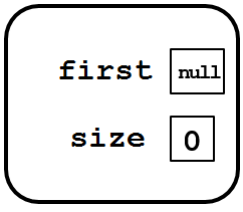
Agregar al paquete de la práctica el fichero `SetString.java`, disponible en `PoliformaT` de `PRG`, en la carpeta `Recursos/Laboratorio/Práctica 5`.

La clase contiene ya implementado un método `toString` que permite hacer pruebas de los métodos a medida que se vayan desarrollando.

Por lo que se ha comentado más arriba, en la clase se deben declarar los atributos privados:

- `first` de tipo `NodeString`, que dará acceso al primer elemento de la secuencia enlazada en la que se disponen ordenadamente los elementos del conjunto;
- `size` de tipo `int`, que contendrá en cada momento el número de elementos en el conjunto.

El constructor creará el conjunto vacío, como se muestra en la figura:



Actividad 3: Implementación de los métodos `add` y `size`

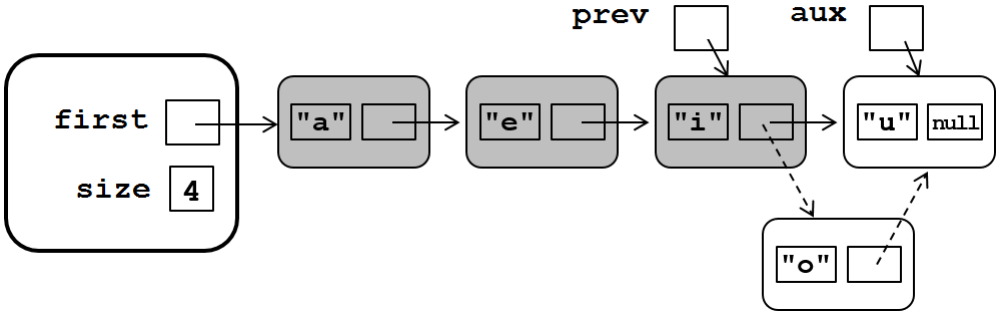
La primera operación a desarrollar será el método `add`, que permitirá ir construyendo un conjunto mediante sucesivas inserciones a partir del conjunto vacío, manteniendo la ordenación de los datos. Cada una de estas operaciones deberá insertar el nuevo elemento en el lugar adecuado de la secuencia y aumentar la talla del conjunto, siempre que dicho elemento no aparezca previamente en el conjunto.

En primer lugar, se deberá buscar el primer nodo en la secuencia con un dato mayor o igual que `s` (según la comparación de `String`), y delante del cual habrá que insertar `s`.

Hay que tener presente que si `s` ya aparece en la secuencia no hay que insertarlo, por lo que se recomienda controlar la búsqueda mediante una variable entera `compare` que se actualice pasada a pasada del bucle y que tome un valor ≥ 0 cuando se encuentre un nodo con un dato mayor o igual que `s`, tal como se muestra en el siguiente esquema de cuerpo del método:

```
NodeString aux = this.first; // nodo a revisar, inicialmente el primero
NodeString prev = null; // anterior al nodo aux, inicialmente null
int compare = -1; // valor inicial de la comparación
while (aux != null && compare < 0) {
    // comparar aux.data con s,
    // y si aux.data es menor entonces avanzar en la secuencia
}
// Si no se ha encontrado un elemento igual a s, aumentar this.size
// e insertar s en su posición:
//   - detrás de prev si es diferente de null (el último elemento menor que s),
//   - al inicio en caso contrario (ningún elemento es menor que s).
```

Como en el esquema típico de búsqueda en una secuencia enlazada, el valor final de `prev` determina la posición detrás de la que habría que insertar `s`, como en el ejemplo de la siguiente figura, en donde se han sombreado los nodos por los que ya ha avanzado la búsqueda:



Recordar que `s` no se debe insertar si ya aparece en la secuencia, lo que se puede comprobar mediante el valor de la última comparación realizada.

La implementación del método `size` se resuelve devolviendo el valor del atributo del mismo nombre.

Para probar estos métodos, se recomienda crear por ejemplo el conjunto de vocales mediante los siguiente pasos:

1. Construir el conjunto vacío en el banco de trabajo.
2. Añadir sucesivamente los elementos `"i"` (en el conjunto vacío), `"a"` (delante del primero), `"u"` (detrás del último), `"e"`, `"o"` (en posiciones intermedias).
3. Probar a añadir un elemento que ya está en el conjunto, `"u"` por ejemplo.

Inspeccionando el objeto resultante cada vez que se añade un elemento, o con los métodos `toString` y `size`, se comprobará que el conjunto se crea correctamente, y con la talla que corresponde.

Actividad 4: Implementación de los métodos `contains` y `remove`

La implementación de estos métodos debe tener en cuenta que la secuencia de elementos está ordenada ascendentemente.

El método `contains` se puede escribir entonces como la búsqueda del primer nodo con un dato mayor o igual que `s`. Teminada la búsqueda, hay que devolver `true` sii la búsqueda ha terminado encontrando un dato exactamente igual a `s`.

El método `remove` puede seguir un esquema análogo al de añadir, salvo que se eliminará el nodo encontrado solamente si contiene a `s` como dato, y disminuyendo la talla en dicho caso.

Para probar ambos métodos se deberán hacer como mínimo las siguientes pruebas:

1. Construir el conjunto vacío en el banco de trabajo y comprobar si `"a"` pertenece al conjunto.

2. Añadir sucesivamente al conjunto cada uno de los elementos `"a"`, `"e"`, `"i"`, `"o"`, `"u"`, y comprobar si cada elemento que se añade pertenece al conjunto, antes y después de añadirlo.

3. En el conjunto de vocales obtenido por las inserciones anteriores, probar a eliminar por este orden los elementos `"z"` (elemento que no pertenece al conjunto), `"a"` (el primero), `"u"` (el último), `"i"` (posición intermedia), `"e"`, `"o"` (elementos restantes), y de nuevo `"a"` (eliminación en el conjunto vacío). Inspeccionando el objeto resultante de cada una de estas operaciones, o con los métodos `toString` y `size` se comprobará que el conjunto se actualiza correctamente, y con la talla que corresponde.

Actividad 5: Implementación del método `intersection`

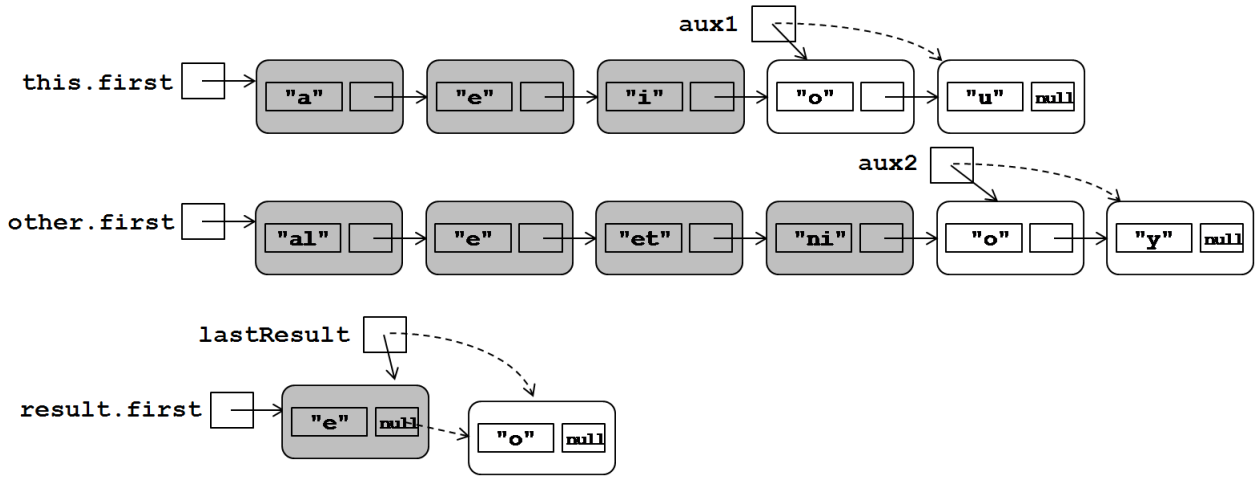
La ordenación de las secuencias enlazadas con la que se implementan los objetos de la clase facilita que la operación de intersección se pueda resolver con un coste `O(this.size + other.size)`, es decir, recorriendo a los sumo una vez los nodos de las secuencias enlazadas `this.first` y `other.first`.

Para ello se puede seguir una estrategia muy parecida a la de *mezcla natural* de dos arrays ordenados. Es decir, para encontrar los elementos en común se pueden ir revisando los datos en el mismo orden en el que aparecen en las secuencias. Cuando se encuentre un elemento común, se incorporará al final del conjunto resultante, manteniendo así el orden preceptivo; además, hay que tener cuidado en que los elementos en común sólo se añadan una vez al conjunto intersección.

El cuerpo del método puede seguir un esquema como el siguiente:

```
SetString result = new SetString();
NodeString aux1 = this.first; // nodo a revisar del conjunto this
NodeString aux2 = other.first; // nodo a revisar de otro
NodeString lastResult = null; // último nodo de result, inicialmente null
while (aux1 != null && aux2 != null) {
    // comparar los datos de aux1 y aux2;
    // si son iguales:
    //    - insertar el dato detrás de lastResult (al inicio de result si es null),
    //    - avanzar en las dos secuencias,
    //    - actualizar la talla de result,
    // si no, avanzar en la secuencia en la que aparece un dato menor.
}
```

El ejemplo de la siguiente figura muestra un estado intermedio del cálculo de la intersección de dos conjuntos:



Cuando se agota una de las dos secuencias, ya no puede haber elementos en común, y se debe terminar devolviendo el conjunto resultante construido.

Una vez implementado el método, se deberán hacer como mínimo las siguientes pruebas:

1. Crear dos conjuntos vacíos `c1` y `c2` en el banco de objetos y comprobar su intersección.
2. Añadir a `c1` los elementos `"a"`, `"c"` y `"e"`, y comprobar la intersección con `c2`.
3. Añadir a `c2` los elementos `"o"`, `"p"` y `"q"`, y comprobar la intersección con `c1`.
4. Añadir a `c1` los elementos `"d"`, `"f"`, y `"o"`, y a `c2` los elementos `"d"`, y `"f"`, y comprobar su intersección.
5. Comprobar la intersección de `c1` consigo mismo.

Actividad 6: Implementación del método `union`

La implementación del método de unión de los conjuntos también va a seguir una estrategia de mezcla natural, por lo que la estructura será muy parecida a la del método anterior.

Las diferencias a tener en cuenta residen en el elemento que habría que añadir a `result` como resultado de comparar los datos de `aux1` y `aux2`, y el correspondiente avance en las secuencias.

Además, cuando una de las secuencias se haya agotado, quedará por añadir a `result` los nodos restantes de la otra.

Una vez implementado el método, se deberán hacer pruebas análogas a las realizadas para el método de intersección.

5. Una aplicación de `SetString` a la comparación de textos

En esta sección se aplicará el tipo de datos `SetString` a la realización de una pequeña aplicación Java que permita obtener la unión o la intersección de las palabras que se encuentran en dos ficheros de texto.

En particular, dados dos ficheros de texto cualesquiera `f1` y `f2` se va a implementar una clase Java, `TextComparing`, que permitirá obtener la unión o intersección de las palabras en ambos ficheros, mediante su ejecución[1] como:

```
$ java TextComparing opción f1 f2
```

donde `opción` es bien el modificador `-i`, bien `-u` según lo que se desea obtener sea, respectivamente, la intersección o la unión de los ficheros `f1` y `f2`. El resultado de la ejecución del comando anterior (las palabras que son la unión o intersección de las palabras en los ficheros) se muestra como resultado, en la salida estándar.

Actividad 7: Revisión del material que se proporciona

Para facilitar el trabajo anterior, se proporciona la clase `TextComparing.java`, parcialmente realizada, y cuyo método `main` se encarga de ejecutar las órdenes del comando que representa la clase `TextComparing`.

A este respecto, conviene recordar que cuando se ejecuta el método `main` de una clase (tanto si se usa un entorno como el BlueJ como si dicha ejecución se hace en modo comando), se recibe como argumento un array de `String` con las cuales es posible operar, que se corresponden a las cadenas de caracteres que aparecen a continuación del nombre de la clase que se ejecuta.

Por ejemplo, cuando se ejecuta en una terminal del sistema un comando, similar al ya mencionado:

```
$ java TextComparing -i f1 f2
```

[1] Para ejecutar el `main` de la clase como un comando del sistema, es necesario abrir un terminal, situarse en el directorio donde se encuentra el fichero `bytecode` ejecutable (`TextComparing.class`) y ejecutar el comando descrito.

el método `main` de la clase `TextComparing` recibe como argumento un array[2] de tres elementos `String`, cuyos valores en las posiciones 0, 1 y 2 son, respectivamente, las `String`: `"-i"`, `"f1"` y `"f2"`.

Esta ejecución se puede probar en BlueJ haciendo la llamada `main({"-i", "f1", "f2"})` en el desplegable de métodos públicos de `TextComparing`.

En cualquier caso, es necesario leer el código que aparece en el fichero `TextComparing.java` detenidamente, para poder efectuar en él las modificaciones que se propongan en dicha clase.

Actividad 8: Finalización de la clase `TextComparing`

Para completar la clase, se recomienda completar en primer lugar el siguiente método `private`:

```
/**
 *  Devuelve el SetString de las palabras leidas de s
 *  segun los separadores dados, por defecto, en DELIMITERS.
 *  @param s Scanner.
 *  @return el conjunto de palabras leidas de s.
 */
private static SetString setReading(Scanner s)
```

cuyo objetivo es obtener el conjunto de las palabras que se lean de `s`, considerando como separadores de las palabras los signos de puntuación habituales (y no únicamente los blancos Java).

Para ello, antes que nada se cambiará su configuración o modo de trabajo con el método `useDelimiter` propio de `Scanner`, de la siguiente forma:

```
s.useDelimiter(DELIMITERS);
```

en donde la constante `DELIMITERS` de la clase define[3] como separadores entre tokens cualquier secuencia de caracteres que sean dígitos, signos de puntuación (incluyendo los signos en castellano `'¿'` y `'¡'`), además de los blancos Java.

Los tokens leídos se añadirán al `SetString` que se debe devolver como resultado del método.

Hecho lo anterior, puede implementarse el método restante, cuyo perfil es:

```
/**
 *  Escribe en la salida estandar el resultado de comparar
 *  los conjuntos de palabras de los ficheros de texto cuyos
 *  nombres estan en nF1 y nF2. Si la opcion es "-i", escribe la
 *  interseccion de ambos conjuntos, si es "-u" escribe la union.
 *  @param nF1 String, nombre del primer fichero.
 *  @param nF2 String, nombre del segundo fichero.
 *  @param option String.
 */
public static void compare(String nF1, String nF2, String option) {
```

Como se indica en los comentarios de documentación, dados los nombres de dos ficheros de texto, este método debe mostrar en la salida estándar la lista de palabras comunes a ambos textos, o la lista de palabras que aparecen en uno u otro.

Para implementar este método, se abrirán un par de `Scanner` a partir de `nF1` y `nF2`, y se obtendrán los conjuntos de palabras de ambos ficheros usando el método `setReading` anterior. Los conjuntos obtenidos se procesarán usando las operaciones de `SetString`, `union` o `intersection` según la opción que se pase como parámetro, escribiendo el conjunto resultante en la salida estándar.

Este procesamiento de los datos sólo se puede llevar a cabo si ambos ficheros se pueden abrir con éxito. Si fallara el acceso a alguno de los ficheros, se capturará la excepción `FileNotFoundException` producida y se escribirá la frase `Mal acceso al fichero:`, seguida del mensaje que se extraiga de la excepción.

Finalmente, se deberá asegurar que los ficheros que se hubieran abierto queden cerrados.

[2] El array `args` en el código que se proporciona.

[3] Mediante unos patrones que se describen en la clase `Pattern` de Java.

Actividad 9: Pruebas de funcionamiento de `TextComparing`

Entre el material que se proporciona, se encuentran los ficheros `javaReserved.txt` y `cReserved.txt` que contienen respectivamente la lista de palabras reservadas de los lenguajes Java y C. Estos ficheros se copiarán dentro del proyecto `prg`, y se podrán usar para la siguientes pruebas:

1. Obtener la intersección de las palabras reservadas de Java y C.
2. Obtener la unión de las palabras reservadas de Java y C.

En la siguiente figura se muestran las listas de palabras que se deben obtener de las pruebas 1 y 2. A la izquierda aparece la lista de palabras reservadas comunes a C y Java. A la derecha, el total de palabras reservadas de ambos lenguajes.

```
break
case
char
const
continue
default
do
double
else
enum
float
for
goto
if
int
long
return
short
static
switch
void
volatile
while
```

```
abstract      long
assert        native
auto          new
boolean       null
break         package
byte          private
case          protected
catch         public
char          register
class         return
const         short
continue      signed
default       sizeof
do            static
double        strictfp
else          struct
enum          super
extends       switch
extern        synchronized
false         this
final         throw
finally       throws
float         transient
for           true
goto          try
if            typedef
implements    union
import        unsigned
instanceof    void
int           volatile
interface     while
```

Se podrán hacer otras pruebas de uso como, por ejemplo, listar las palabras reservadas de Java que aparecen en el fichero `./pract5/TextComparing.java`, etc.

6. Evaluación

Esta práctica forma parte del segundo bloque de prácticas de la asignatura que será evaluada en el segundo parcial de la misma. El valor de este bloque es de un 60 % respecto al total de las prácticas. El valor porcentual de les prácticas en la asignatura es de un 20 % de la nota final.