Estrategias de Programación y Estructuras de Datos

Grado en Ingeniería Informática

Grado en Tecnologías de la Información

Práctica curso 2019-2020

Enunciado

1. Presentación del problema

El conocido concurso "*Cifras y Letras*" propone a sus participantes dos tipos de pruebas: una prueba de cifras en la que deben aproximarse lo más posible a un número objetivo utilizando operaciones básicas sobre una serie de números y una prueba de letras, en la que deben encontrar la palabra (válida) más larga que se pueda formar con un grupo de letras determinado.

En esta práctica vamos a crear un programa que resuelve esta prueba de forma automática. Concretamente el programa recibirá una serie de letras y nos devolverá todas las palabras válidas que se puedan construir con ellas. Para comprobar si una palabra es válida o no, es imprescindible el uso de un diccionario.

Para ello, se proporcionará un fichero con el diccionario en forma de lista de 79517 palabras pertenecientes al Diccionario de la RAE para que los estudiantes puedan probar su programa. Para evitar problemas debidos a la codificación de caracteres, se han borrado todas las palabras que contienen la ñ y se han eliminado los acentos gráficos.

La práctica se dividirá en dos fases diferentes:

1. **Creación del diccionario**: se cargará un fichero de texto que contiene las palabras que vamos a considerar válidas (una palabra por línea) y se insertará cada una de ellas en una estructura de diccionario sobre la cual se consultarán las posibles palabras.
2. **Búsqueda de palabras**: una vez se disponga del diccionario, el programa cargará un segundo fichero de búsquedas a realizar, cada una de las cuales consistirá en una secuencia de letras a analizar y un tamaño de palabra (que podrá ser un número o bien ALL para que el programa considere todos los posibles tamaños de palabra presentes en el diccionario). El programa devolverá todas las posibles palabras válidas que se puedan construir a partir de esas letras (y del tamaño adecuado), ordenadas alfabéticamente y agrupadas por tamaño de mayor a menor.

La única restricción que se plantea es que ***no se permite el uso de iteradores***, por lo que cualquier acceso a las estructuras de datos deberá hacerse prescindiendo de ellos.

2. Enunciado de la práctica

La práctica consiste en elaborar un programa en Java, utilizando los Tipos de Datos programados por el Equipo Docente, que resuelva el problema propuesto. Los estudiantes dispondrán de una serie de clases ya programadas (total o parcialmente) y deberán completar el programa de forma que pueda realizar correctamente la tarea que se ha indicado.

En este apartado vamos a ver cuál será la estructura de datos sobre la que se va a crear el diccionario, cómo habrán de insertarse en ella las diferentes palabras y cómo se podrá realizar el proceso de búsqueda, de manera que se obtenga el resultado esperado.

2.1 Ejemplos de funcionamiento

En primer lugar, vamos a ver unos ejemplos de funcionamiento en los que se utiliza el diccionario completo de 79517 palabras:

Ejemplo 1: se pide obtener todas las palabras de cualquier tamaño que puedan ser construidas a partir de las letras de la secuencia "`aosc`". El resultado sería:

```
Secuencia: "aosc"
-Palabras de 4 letras: asco, caos, caso, cosa, saco, soca
-Palabras de 3 letras: cao, cas, coa, oca, osa, sao
-Palabras de 2 letras: as, ca, oc, os, so
-Palabras de 1 letra: a, c, o, s
```

Ejemplo 2: se pide obtener todas las palabras *de exactamente seis letras* que puedan ser construidas a partir de las letras de la secuencia "`riomwalfo`". El resultado sería:

```
Secuencia: "riomwalfo"
-Palabras de 6 letras: almori, amorfo, amorio, ariolo, filmar, firmal, foliar,
formal, formol, marfil, mariol, moflir
```

Como se puede ver, en el primer ejemplo las palabras se presentan agrupadas por tamaño (de mayor a menor) y para un mismo tamaño en orden alfabético. Esto último también puede comprobarse en el segundo ejemplo.

2.2 Estructura del diccionario

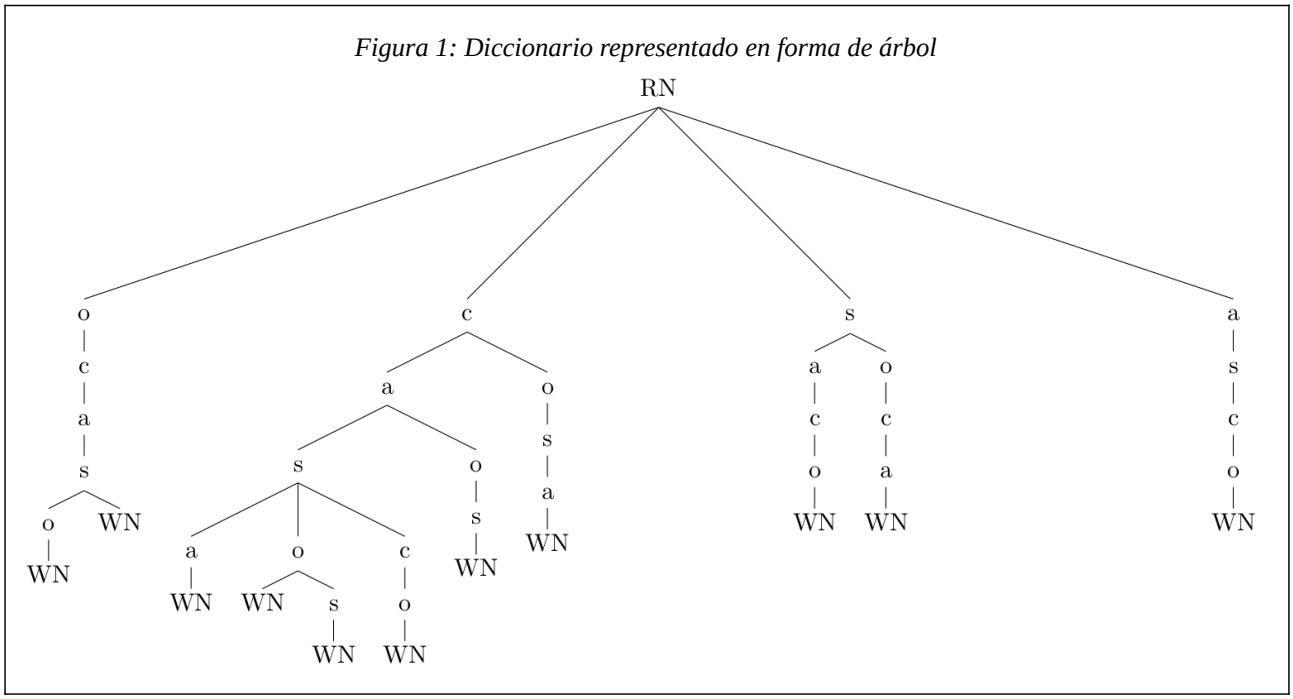
En este apartado vamos a explicar cuál será la estructura que va a tener nuestro diccionario, lo que va a condicionar cómo se han de introducir en él las palabras leídas.

La idea va a ser utilizar un árbol de caracteres en el que cada nodo intermedio (sin incluir la raíz, que es un caso especial) va a contener un carácter y los nodos hoja van a darnos información acerca de las palabras que contiene el diccionario, de forma que la concatenación de los caracteres que vayamos encontrando en el camino desde la raíz hasta un nodo hoja será una palabra del diccionario.

A continuación vamos a ver un ejemplo gráfico de un pequeño diccionario (diferente al utilizado en los ejemplos anteriores) construido a partir de la siguiente lista de palabras:

```
["casa","caso","casco","caos","saco",
"casos","cosa","ocas","asco","soca","ocaso"]
```

Figura 1: Diccionario representado en forma de árbol



Aquí podemos ver el diccionario resultante representado gráficamente en forma de árbol. Como se indicó previamente, los nodos intermedios (salvo la raíz, representada como RN) son los únicos que contienen caracteres. Y la concatenación de los caracteres que forman el camino desde la raíz a cualquiera de los nodos hoja (representados como WN) nos indica las palabras presentes en el diccionario.

Para una mejor comprensión de la estructura de datos a utilizar, sugerimos a los estudiantes que inviertan un tiempo recorriendo el diccionario mostrado en la *Figura 1* para comprobar que contiene exactamente las once palabras indicadas en la lista que se ha utilizado para crearlo.

2.3 Consultas al diccionario

En este apartado vamos a suponer que ya tenemos nuestro diccionario construido y vamos a explicar cómo se realizan las consultas al mismo.

Por ejemplo, si buscamos la secuencia "`cascao`" sobre el diccionario representado en la *Figura 1* , deberíamos obtener la siguiente lista de palabras en orden alfabético:

```
["asco","caos","casa","casco","caso","cosa","ocas","saco","soca"]
```

Nótese que "`casos`" y "`ocaso`", presentes en el diccionario, no pueden formarse con las letras de la secuencia, ya que ésta sólo contiene una '`s`' (y "`casos`" tiene dos) y una '`o`' (y "`ocaso`" tiene dos).

En cada momento de la búsqueda habremos realizado un recorrido previo que nos ha llevado a un nodo del árbol (que no es una hoja). Tendremos, por tanto, una secuencia de caracteres aún no utilizados y el comienzo de una palabra formado por los caracteres presentes en el camino seguido desde la raíz hasta el nodo actual.

La búsqueda continúa analizando todos los hijos del nodo actual:

- Si un hijo es una hoja (representada como WN en la *Figura 1*), significará que la palabra que ya tenemos formada pertenece al diccionario, con lo cual, habrá que añadirla.
- Si un hijo no es una hoja, será un nodo intermedio que contendrá un carácter. Únicamente si dicho carácter pertenece a la secuencia dada (en cualquier posición, no necesariamente el primero), deberemos explorar recursivamente este hijo, añadiendo a la palabra ya formada el carácter del nodo actual y eliminando dicho carácter de la secuencia.

Nuevamente volvemos a recomendar que los estudiantes inviertan un tiempo en realizar este recorrido sobre el árbol de la *Figura 1* para la secuencia "`cascao`" y comprueben así que se obtienen exactamente las palabras indicadas.

2.4 Cuestiones teóricas sobre el enunciado

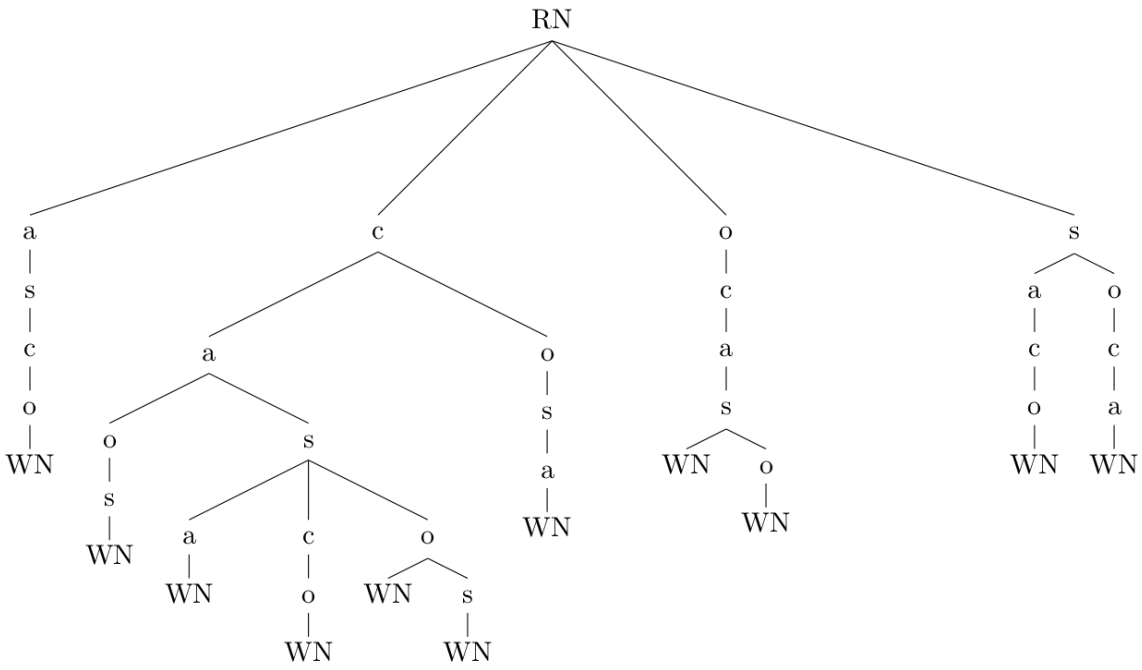
A continuación se enuncian una serie de preguntas teóricas sobre lo que se ha visto hasta ahora de la práctica. La respuesta a estas preguntas implica la aplicación de los conocimientos teóricos de la asignatura al problema propuesto en la práctica. Responder a estas preguntas no es obligatorio para aprobar la práctica, pero la adecuación de las respuestas influirá en la calificación final de la práctica.

1. (**0,5 puntos**) Visto el algoritmo de búsqueda descrito en el apartado anterior, ¿se describe un recorrido del árbol en anchura o en profundidad? Razone su respuesta.
2. (**1 punto**) Realizando el recorrido propuesto como respuesta en la pregunta anterior sobre el

árbol de la *Figura 1*, ¿se obtiene directamente la lista de palabras en orden alfabético? En caso negativo, ¿cuál sería el coste de ordenar la lista de palabras a posteriori? Razone su respuesta.

3. (**1 punto**) A continuación se muestra otra versión del diccionario en forma de árbol que vimos en la *Figura 1*:

Figura 2: Nueva versión del diccionario en forma de árbol



Realice el mismo recorrido que hizo sobre el árbol de la *Figura 1* sobre el árbol mostrado en la Figura 2. ¿Se obtienen ahora las palabras en orden alfabético? Comparando los dos árboles, ¿cómo habrían de organizarse los hijos de un nodo para que con este recorrido se obtengan las palabras directamente en orden alfabético? Razone su respuesta.

3. Diseño de la práctica

A continuación vamos a ver el diseño de clases de la práctica necesario para su implementación. Para aquellas clases ya programadas, incluiremos una descripción de su funcionamiento y para aquellas que deban ser completadas por los estudiantes indicaremos qué funciones han de incluir y cuál será su comportamiento esperado.

3.1 Nodos del árbol

Como hemos visto, el diccionario va a consistir en un árbol con tres tipos de nodos diferentes:

- Raíz: es un nodo que no es hoja y no contiene ningún tipo de información adicional.
- Nodo intermedio: es un nodo que no es hoja y contiene un carácter.
- Nodos hoja: estos nodos no contienen ningún tipo de información adicional y marcan las palabras que están contenidas en el diccionario.

Como las colecciones de elementos (y, en particular, los árboles) deben contener elementos del mismo tipo, crearemos una clase abstracta `Node` que englobará todos los tipos de nodo del árbol donde almacenaremos nuestro diccionario:

```
public abstract class Node {
	public enum NodeType {
		ROOTNODE, LETTERNODE, WORDNODE
	}

	/* Prescribe un getter que devuelve el tipo de nodo */
	public abstract NodeType getNodeType();
}
```

Así pues, la clase abstracta `Node` (proporcionada por el Equipo Docente) nos define un enumerado con los tres tipos de nodos existentes: raíz (`ROOTNODE`), nodo intermedio que contiene una letra (`LETTERNODE`) y nodo raíz que nos marca el final de una palabra del diccionario (`WORDNODE`).

De esta clase deberán heredar tres clases no abstractas que definan los tres tipos de nodos. Todas ellas deberán incluir el método `getNodeType`, que está prescrito por la clase abstracta padre:

- Clase `RootNode`: representará el nodo raíz.
- Clase `LetterNode`: representará un nodo intermedio, por lo que deberá almacenar un carácter y ofrecer un getter para poder obtenerlo.
- Clase `WordNode`: representará un nodo hoja.

El primer paso en la programación de la práctica será la creación de estas tres clases como hijas de la clase `Node`.

3.2 Secuencias de palabras

Nuestro programa deberá ser capaz de devolver las secuencias de palabras ordenadas alfabéticamente y agrupadas por tamaño (en caso de que se pidan todas las palabras). Para ello vamos a utilizar dos clases: `WordListN` y `WordList`.

Clase WordListN

Un objeto de esta clase va a contener una secuencia de palabras ordenadas alfabéticamente y todas ellas de la misma longitud.

```
public class WordListN {
	/* Atributos de la clase con la estructura adecuada */
	…
	/* Atributos de la clase con la estructura adecuada */

	/* Constructor */
	public WordListN(int size) {…}

	/* Añade una palabra a la estructura */
	public void add(String word) {…}

	/* Devuelve el tamaño de las palabras de la estructura */
	public int getWordSize() {…}

	/* Construye un String con el contenido de la estructura */
	public String toString() {
		StringBuilder salida = new StringBuilder();
		int numPalabras = /* Longitud de la secuencia de palabras */
		salida.append("-Palabras de ");
		salida.append(this.getWordSize());
		salida.append(" letra");
```

```
            if ( this.getWordSize() > 1 ) { salida.append('s'); }
            salida.append(": ");
            for (int pos = 1 ; pos <= numPalabras ; pos++) {
                  /* Estas líneas dependen de la estructura escogida */
                  String word = /* Obtener la siguiente palabra */
                  /* Avanzar a la siguiente sin destruir la estructura */
                  …
                  /* Estas líneas dependen de la estructura escogida */
                  salida.append(word);
                  if ( pos < numPalabras ) {
                        salida.append(", ");
                  }
            }
            salida.append('\n');
            return salida.toString();
      }
}
```

La tarea de programación para esta clase será:

1. Decidir qué estructura secuencial es la más adecuada para almacenar la secuencia de palabras de la misma longitud en orden alfabético y rellenar las líneas que definan los atributos adecuados para representar esa estructura.
2. En base a la estructura escogida, crear el código del constructor de la clase.
3. En base a la estructura escogida, crear el código del método `add(String word)`, que añade una nueva palabra a la estructura. Recordemos que siempre se deberá mantener las palabras dentro de la secuencia en orden alfabético.
4. En base a la estructura escogida, crear el código del método `getWordSize()`, que devuelve el tamaño de las palabras almacenadas en la estructura (el cual, recordemos, ha de ser el mismo para todas ellas).
5. En base a la estructura escogida, rellenar las líneas necesarias del método `toString()` para obtener la siguiente palabra y avanzar en el recorrido sin destruir la estructura. Este método crea una cadena de caracteres con el contenido de la estructura.

Clase WordList

Los objetos de la clase `WordList` van a almacenar las palabras de cualquier longitud devueltas por una búsqueda sobre el diccionario, agrupadas por tamaño decreciente en objetos de la clase `WordListN`.

```
public class WordList {
      private ListIF<WordListN> wordList;

      /* Constructor */
      public WordList() {
            this.wordList = new List<WordListN>();
      }

      /* Método que añade una nueva palabra a la estructura */
      public void add(String word) {…}

      /* Construye un String con el contenido de la estructura */
      public String toString() {
            StringBuilder salida = new StringBuilder();
            for ( int pos = 1 ; pos <= this.wordList.size() ; pos++ ) {
                  salida.append(this.wordList.get(pos).toString());
```

```
            }
            return salida.toString();
      }
}
```

La tarea de programación para esta clase consiste en programar el método `add(String word)` que añade una nueva palabra a la estructura. Este método deberá delegar en el método `add(String word)` de la clase `WordListN` tras localizar en la lista `wordList` el objeto `WordListN` adecuado donde realizar la inserción.

Cuestiones teóricas

A continuación se enuncian una serie de preguntas teóricas sobre lo que se ha visto en este punto de la práctica. La respuesta a estas preguntas implica la aplicación de los conocimientos teóricos de la asignatura al problema propuesto en la práctica. Responder a estas preguntas no es obligatorio para aprobar la práctica, pero la adecuación de las respuestas influirá en la calificación final de la práctica.

1. (0,5 puntos) ¿Qué estructura secuencial ha escogido para almacenar las secuencias de palabras de la misma longitud dentro de los objetos de la clase `WordListN`? ¿Por qué ha descartado otras posibles estructuras? Justifique su respuesta.
2. (1 punto) En base a la estructura escogida, calcule el coste asintótico temporal en el caso peor del método `add(String word)` dentro de las clases `WordListN` y `WordList`.

3.3 Diccionario

Una vez tengamos ya programadas las clases de los tipos de nodo que tenemos en el árbol y las clases de las secuencias de palabras (`WordListN` y `WordList`), podemos pasar a la clase `Dictionary`, que es la que se encargará de la creación y gestión de nuestro diccionario.

```
public class Dictionary {
      private GTree<Node> dict; /* Estructura: un árbol general de nodos */

      /* Constructor de la clase */
      public Dictionary() {…}

      /* Método de inserción de una nueva palabra en el diccionario */
      public void insert(String word) {
            /* Insertamos la palabra a partir del nodo raíz del árbol */
            insertInTree(word,this.dict);
      }

      /* Método privado llamado por el anterior */
      private void insertInTree(String word, GTreeIF<Node> node) {…}

      /* Método de búsqueda de todas las palabras de cualquier tamaño *
       * a partir de una secuencia                                    */
      public WordList search(String sequence) {
            WordList salida = new WordList();
            searchInTree(sequence,"",this.dict,salida);
            return salida;
      }

      /* Método privado llamado por el anterior */
      private void searchInTree(String sequence, String word,
                                GTreeIF<Node> node, WordList salida) {…}
```

```
        /* Método de búsqueda de todas las palabras de tamaño size  *
         * a partir de una secuencia                                */
        public WordListN search(String sequence, int size) {
              WordListN salida = new WordListN(size);
              searchInTreeN(sequence,"",this.dict,salida,size);
              return salida;
        }
        /* Método privado llamado por el anterior */
        private void searchInTreeN(String sequence, String word,
                                   GTreeIF<Node> node, WordListN salida,
                                   int size) {…}
}
```

Como se puede ver, la clase Dictionary posee (además del constructor) tres métodos públicos:

- `insert(String word)`: recibe una palabra en forma de cadena de caracteres y la inserta en el diccionario en forma de árbol. El código del método se da ya implementado y consiste en una llamada al método privado `insertInTree` que es el que realiza la inserción de forma recursiva del resto de la palabra a insertar a partir de un nodo del árbol.
- `search(String sequence)`: recibe una secuencia de letras en forma de cadena de caracteres. Devuelve un objeto de tipo WordList con todas las palabras presentes en el diccionario que pueden formarse con las letras de la secuencia recibida en la entrada. El código del método ya se da implementado y consiste en una llamada a un método privado `searchInTree` que recibe los caracteres aún no utilizados de la secuencia, la palabra que se ha formado hasta el nodo actual, el nodo actual y el objeto `WordList` donde se va construyendo la salida.
- `search(String sequence, int size)`: recibe una secuencia de letras en forma de cadena de caracteres y un entero `size` que representa un tamaño. Devuelve un objeto de tipo `WordListN` con todas las palabras presentes en el diccionario con un tamaño de exactamente `size` letras que pueden formarse con las letras de la secuencia recibida en la entrada. El código del método ya se da implementado y consiste en una llamada a un método privado `searchInTreeN` que recibe los caracteres aún no utilizados de la secuencia, la palabra que se ha formado hasta el nodo actual, el nodo actual, el objeto `WordListN` donde se va construyendo la salida y el tamaño de las palabras buscadas. **Para realizar este método no se podrá llamar a los métodos `search(sequence)` ni a `searchInTree`.**

La tarea de programación para esta clase, con la que finalizaremos la programación de la práctica, consiste en programar el constructor de la clase y los métodos privados `insertInTree`, `searchInTree` y `searchInTreeN` descritos anteriormente.

3.4 Programa principal

La clase `Main` contiene el programa principal y se da completamente terminada por parte del Equipo Docente. A continuación se explica el funcionamiento del método `main`:

1. Nuestro programa requiere exactamente dos argumentos para funcionar: el fichero conteniendo la lista de palabras a cargar en el diccionario y el fichero con las búsquedas a realizar. En caso de que no se especifiquen correctamente, se presenta un mensaje indicando cómo se ha de usar el programa.
2. Si se han proporcionado adecuadamente los dos parámetros, se interpreta que el primero representa el fichero con las palabras, así que se crea un nuevo objeto de tipo `Dictionary`, se abre el fichero y se van leyendo todas las palabras e insertándolas en el diccionario con llamadas al método `insert` de `Dictionary`.

3. Se interpreta que el segundo parámetro es el fichero con las búsquedas a realizar, así que se abre y se leen las secuencias y los tamaños buscados. Si el tamaño es `ALL` (con independencia de la capitalización de las letras) se interpreta que se trata de una búsqueda de todas las palabras y se llama al método `search` de `Dictionary` que recibe un único parámetro (secuencia de caracteres). Si el tamaño no es `ALL`, se interpreta que se trata de un entero y se llama al método `search` de `Dictionary` que recibe dos parámetros (secuencia y tamaño).
4. Si en cualquier momento hubiera un error de entrada/salida, se mostrará un mensaje indicándolo y se detendrá el programa.

4. Implementación.

Se deberá realizar un programa en Java llamado **`eped2020.jar`** que contenga todas las clases anteriormente descritas completamente programadas. Todas ellas se implementarán en un único paquete llamado:

`es.uned.lsi.eped.pract2019_2020`

Para la implementación de las estructuras de datos de soporte **se deberá utilizar las interfaces y las implementaciones proporcionadas por el Equipo Docente** de la asignatura. Recuérdese que **no se permite el uso de iteradores, por lo que si se detecta su uso la práctica estaría automáticamente suspensa**.

A través del entorno virtual el Equipo Docente proporcionará el código de todas las clases (total o parcialmente) implementado según la descripción que se ha hecho en este documento.

Esto significa que la lectura de los parámetros de entrada, lectura de los ficheros y salida del programa ya está programada por el Equipo Docente y los estudiantes no tienen que realizar ni modificar nada sobre estos temas (salvo completar el método `toString` de `WordListN`, tal y como se ha indicado previamente).

5. Ejecución y juegos de prueba.

Para la ejecución del programa se deberá abrir una consola y ejecutar:

```
java –jar eped2020.jar <diccionario> <busquedas>
```

siendo:

- **`<diccionario>`** nombre del fichero con las palabras a insertar en el diccionario.
- **`<busquedas>`** nombre del fichero con las búsquedas a realizar.

El Equipo Docente proporcionará, a través del curso virtual, unos juegos de prueba para que los estudiantes puedan comprobar el correcto funcionamiento del programa. Si se supera el juego de pruebas privado de los tutores (que será diferente del proporcionado a los estudiantes), la práctica se considerará aprobada (con una calificación de 6 puntos) a falta de la evaluación de las preguntas teóricas.

6. Documentación y plazos de entrega.

La práctica supone un 20% de la calificación de la asignatura, y es necesario aprobarla para superar la asignatura. Además será necesario obtener, al menos, un 4 sobre 10 en el examen presencial para

que la calificación de la práctica sea tenida en cuenta de cara a la calificación final de la asignatura.

Los Centros Asociados organizarán sesiones de prácticas en las que los tutores monitorizarán y orientarán a los estudiantes en su realización de la práctica. La asistencia a estas sesiones no es obligatoria, pero se recomienda encarecidamente a los estudiantes asistir a ellas. La interacción con el tutor en esas sesiones presenciales puede influir en la calificación de la práctica. Estas sesiones son organizadas por los Centros Asociados teniendo en cuenta sus recursos y el número de estudiantes matriculados, por lo que en cada Centro las fechas serán diferentes. Los estudiantes deberán, por tanto, dirigirse a su tutor para conocer las fechas de celebración de estas sesiones.

De igual modo, el plazo y forma de entrega son establecidos por los tutores de forma independiente en cada Centro Asociado, por lo que deberán ser consultados también con ellos.

La documentación que debe entregar cada estudiante consiste en:

- Memoria de práctica, en la que se deberán responder a las preguntas teóricas.
- Implementación en Java de la práctica, de la cual se deberá aportar **tanto el código fuente como el programa compilado**.