UNED

ETS de Ingeniería Informática

Estrategias de Programación y Estructuras de Datos

Grado en Ingeniería Informática
Grado en Tecnologías de la Información

Práctica curso 2014-2015

Enunciado

Índice

1. Presentación del problema

Cuando tecleamos nuestra consulta en un buscador Web, el buscador puede proporcionarnos sugerencias de búsqueda mientras escribimos basándose en las consultas más frecuentes que han hecho otros usuarios. En la imagen 1 vemos un ejemplo: El usuario está tecleando "*practicas de pro*" y el buscador sugiere consultas como "*practicas de programación en c*", que es una de las consultas más frecuentes de los usuarios que tienen ese comienzo.

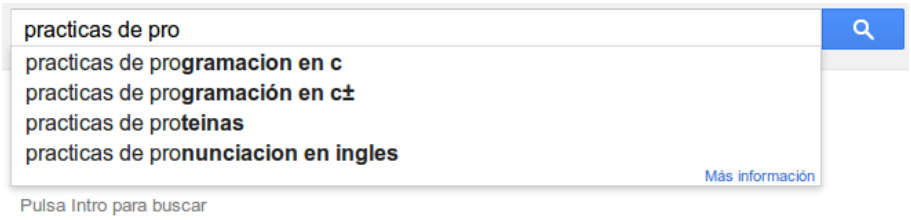

Imagen 1: Ejemplo de sugerencias de búsqueda proporcionadas por Google

El objetivo de la práctica es simular un sistema de sugerencias de ese estilo. A partir de un histórico de búsquedas (una lista de consultas – *queries* – realizadas con anterioridad) y una búsqueda (*query*) incompleta, nuestro programa debe proporcionar una lista, ordenada por frecuencia descendente, de las consultas compatibles con esa consulta incompleta.

El aspecto clave del programa es la elección de la estructura de datos con la que vamos a representar el histórico de consultas:

- Por un lado, el tamaño del histórico de búsquedas en un buscador Web es muy grande: de hecho, puede ser varios órdenes de magnitud más grande que la propia Web. Por ello hay que diseñar una estructura de datos que almacene toda la información sobre las consultas usando el espacio en memoria de forma eficiente.
- Por otro lado, las sugerencias de búsqueda hay que realizarlas dinámicamente (según el usuario va tecleando) así que la estructura de datos debe permitir una consulta muy rápida.

2. Diseño.

En esta práctica trabajaremos con dos diseños alternativos para la estructura de datos: uno inmediato pero poco eficiente, y otro más eficiente que tiene en cuenta los requisitos mencionados. Siguiendo las instrucciones de este enunciado de prácticas, los estudiantes deberán (i) razonar sobre el coste espacial y temporal de cada una de las soluciones propuestas, de forma comparada; (ii) implementar ambas soluciones respecto a un tipo de datos abstracto común a ambas; y (iii) realizar un estudio empírico de su eficiencia respecto al juego de pruebas proporcionado por el equipo docente.

2.1 Tipo de datos de referencia.

En primer lugar debemos especificar la interfaz (el tipo de datos de referencia) para las posibles estructuras de datos con las que representaremos el histórico de consultas al buscador. Llamemos **`QueryDepot`** (*depósito de consultas*) al tipo de datos. A efectos de ese depósito de consultas, nos interesa conocer el texto de la consulta (una cadena de caracteres) junto con su frecuencia (el número de veces que ese texto aparece en el registro de búsquedas).

Por simplicidad, para esta práctica se considerará que dos consultas son la misma si las cadenas de caracteres de ambas son idénticas. Por ejemplo, las consultas "*camion rojo*" y "*camión rojo*" se considerarán diferentes.

Para el manejo de nuestro depósito de consultas necesitamos considerar las siguientes operaciones:

- **Operaciones de creación**:
  - Crear un nuevo depósito de consultas vacío.
  - Crear un nuevo depósito de consultas a partir del registro de consultas, que supondremos es simplemente un fichero con una consulta (una cadena de caracteres) por línea, posiblemente con repeticiones. A los efectos de la práctica ignoraremos el resto de información útil que un buscador puede asociar a cada consulta (usuario, IP, momento en el que se realiza la consulta, localización geográfica, clicks del usuario en los resultados de búsqueda, etc.).
- **Operaciones de modificación**:
  - Cuando se produce una nueva búsqueda, debe ser incluida en el depósito de consultas. Si ya existe, se incrementará en uno su frecuencia. Si no existe, se incluirá en el depósito con frecuencia 1.
  - Por completitud (aunque estrictamente no se necesita para el escenario propuesto, sí puede ser útil en otras aplicaciones), también incluiremos la operación contraria: dado el texto de una consulta se decrementará su frecuencia en 1 dentro del depósito. Si la nueva frecuencia fuese 0, se deberá eliminar toda la información que se refiera únicamente a esa consulta.
- **Operaciones de consulta**:
  - Una operación que devuelva el número de consultas diferentes (sin contar repeticiones) que están almacenadas en el depósito.
  - Dado el texto de una consulta, se desea obtener la frecuencia de aparición de dicha consulta dentro del depósito.
  - Dado un comienzo de consulta (una cadena de caracteres), queremos obtener una lista, ordenada por frecuencia decreciente, de todas las consultas que comienzan por esa cadena de caracteres. Esta es la operación que permitirá realizar sugerencias de búsqueda al usuario dinámicamente. Si dos consultas tienen la misma frecuencia, se devolverán por orden lexicográfico.

Una posible interfaz que responde a los requisitos para las operaciones de modificación y consulta es la siguiente:

```
public interface QueryDepot {

  /* Devuelve el número de consultas diferentes (sin contar repeticiones) */
  /* que hay almacenadas en el depósito */
  /* @returns el número de consultas diferentes almacenadas */
  public int numQueries ();

  /* Consulta la frecuencia de una consulta en el depósito */
  /* @returns la frecuencia de la consulta. Si no está, devolverá 0 */
  /* @param el texto de la consulta */
  public int getFreqQuery (String q);

  /* Dado un prefijo de consulta, devuelve una lista, ordenada por */
  /* frecuencias de mayor a menor, de todas las consultas almacenadas */
  /* en el depósito que comiencen por dicho prefijo */
  /* @returns la lista de consultas ordenada por frecuencias y orden */
```

```
  /*  lexicográfico en caso de coincidencia de frecuencia */
  /* @param el prefijo */
  public ListIF<Query> listOfQueries (String prefix);

  /* Incrementa en uno la frecuencia de una consulta en el depósito */
  /* Si la consulta no existía en la estructura, la deberá añadir */
  /* @param el texto de la consulta */
  public void incFreqQuery (String q);

  /* Decrementa en uno la frecuencia de una consulta en el depósito*/
  /* Si la frecuencia decrementada resultase ser 0, deberá eliminar */
  /* la información referente a la consulta del depósito */
  /* @precondición la consulta debe estar ya en el depósito */
  /* @param el texto de la consulta */
  public void decFreqQuery (String q);
}
```

Las operaciones de creación se definirán mediante constructores para cada una de las implementaciones posibles de la interfaz (dado que no es posible definir un constructor en una interfaz en Java).

Esta interfaz utiliza la clase **`Query`**, que podemos definir así:

```
public class Query {

  /* Construye una nueva query con el texto pasado como parámetro */
  public Query (String text);

  /* Modifica la frecuencia de la query */
  public void setFreq();

  /* Devuelve el texto de una query */
  public String getText();

  /* Devuelve la frecuencia de una query */
  public int getFreq();
}
```

2.2 Primera aproximación: lista de consultas ordenadas alfabéticamente.

La forma más sencilla de implementar la interfaz **`QueryDepot`** es mediante una lista de objetos **`Query`**, ordenados por orden alfabético. Para encontrar todas las consultas que comienzan por una cadena de caracteres dada es necesario recorrer la lista hasta encontrar todas las consultas que comienzan con esa cadena (que estarán consecutivas en la lista), y después realizar una ordenación por frecuencia decreciente.

Para ver un ejemplo de este diseño, supongamos que el listado de consultas realizadas al buscador es el siguiente:

```
caso
casa
casos
caso
casa
```

`cosa`

Una posible representación gráfica del contenido de la estructura **`QueryDepot`** implementada mediante listas de objetos **`Query`** sería la que se muestra en la Imagen 2.

`casa`	2
`caso`	2
`casos`	1
`cosa`	1

Imagen 2: Ejemplo de implementación de la estructura mediante lista de objetos query

ya que las consultas "`caso`" y "`casa`" aparecen dos veces en el listado, mientras que "`casos`" y "`cosa`" sólo aparecen una única vez.

Preguntas teóricas (trabajo del estudiante).

Las siguientes preguntas son previas al trabajo de implementación y deberán ser respondidas por el estudiante e incluidas en la memoria de práctica para su evaluación.

1.1 ¿Cómo depende el tamaño de almacenamiento del número de consultas diferentes en el registro de consultas? ¿Cómo depende del número de repeticiones?
1.2 ¿Cómo depende el tiempo de localizar todas las posibles sugerencias de búsqueda del número de consultas diferentes? ¿Y del número de repeticiones? ¿Y de la longitud máxima de las consultas medida como el número de caracteres de su texto? ¿Y del tamaño del conjunto de caracteres permitido? Razona en términos del coste asintótico temporal en el caso peor que se podría conseguir para el método **`listOfQueries`** con este diseño.
1.3 Consideremos una estructura de datos alternativa en la que disponemos de una lista de consultas similar a la considerada por cada carácter inicial (es decir, tendríamos una lista con todas las consultas que empiezan por el carácter "a", otra con las que empiezan por "b", etc.) ¿Cuánto se podría reducir el tiempo de búsqueda en caso peor? ¿Cambiaría el coste asintótico temporal?
1.4 ¿Se ajusta este diseño a los requisitos del problema? Explica por qué.

2.3 Segunda aproximación: árbol de caracteres.

Consideremos ahora un diseño alternativo para la estructura de datos, en el que utilizaremos un árbol con las siguientes características:

a) **Raíz del árbol**: es un nodo que no contiene ningún tipo de información, pero es necesario ya que de él colgarán los nodos con los caracteres iniciales del texto de cada consulta almacenada.
b) **Nodos intermedios**: Cada nodo intermedio contendrá un carácter. Los que cuelgan del nodo raíz serán todos aquellos caracteres que son el carácter inicial de, al menos, una de las consultas hechas al buscador. Los hijos de cada nodo serán aquellos caracteres que continúen expresando alguna de las consultas hechas al buscador (ver ejemplo en la Imagen 3).
c) **Nodos hoja**: estos nodos contendrán la información sobre la frecuencia de aparición de la consulta asociada a ellos. La consulta asociada a un nodo hoja es el resultado de

concatenar (por orden) los caracteres de los nodos intermedios que forman el camino desde la raíz del árbol hasta él.

Las consultas del ejemplo anterior, almacenadas en este nuevo diseño para la interfaz queryDepot, tendrían el aspecto mostrado en la Imagen 3.

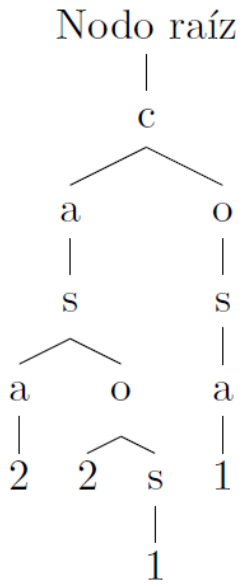

Imagen 3: Ejemplo de implementación de la estructura mediante un árbol

Preguntas teóricas (trabajo del estudiante).

2.1 ¿Cómo depende el tamaño de almacenamiento del número de consultas diferentes en el registro de consultas? ¿Y del tamaño máximo de las consultas? ¿Y del tamaño del conjunto de caracteres permitido? Consideremos un conjunto de 50 caracteres y un tamaño máximo de consulta de 10 caracteres. Compara el tamaño máximo del árbol en la segunda aproximación con el tamaño máximo de la lista en la primera aproximación. ¿La diferencia se agrandará o se reducirá para conjuntos de caracteres mayores y consultas más largas?

2.2 ¿Cómo depende el tiempo de localizar **una** posible sugerencia de búsqueda del número de consultas diferentes? ¿Y del número de repeticiones? ¿Y de la longitud máxima de las consultas medida como el número de caracteres de su texto? ¿Y del tamaño del conjunto de caracteres permitido?

2.3 ¿Cómo depende el tiempo de localizar **todas** las posibles sugerencias de búsqueda del número de consultas diferentes? ¿Y del número de repeticiones? ¿Y de la longitud máxima de las consultas medida como el número de caracteres de su texto? ¿Y del tamaño del conjunto de caracteres permitido? Razona en términos del coste asintótico temporal en el caso peor que se podría conseguir para el método **`listOfQueries`** con este diseño.

2.4 A la vista de las respuestas a las preguntas anteriores y de los requisitos de nuestro problema, ¿consideras éste un diseño más adecuado para **`QueryDepot`**?

2.5 Compara el coste de encontrar todas las sugerencias posibles con el coste de ordenarlas por frecuencia (consideremos que el coste de ordenación en el caso peor sea del orden O(n·log(n))). ¿Sería aconsejable comenzar a realizar sugerencias antes incluso de que el usuario comience a teclear, con este diseño? ¿Por qué?

3. Implementación.

Se deberá realizar un programa en Java llamado **`eped2015.jar`** que contenga dos clases que implementen las dos aproximaciones descritas para el interfaz **`QueryDepot`**:

- La primera aproximación (implementación mediante una lista) se realizará con una clase llamada **`QueryDepotList`**.
- La segunda aproximación (implementación mediante un árbol) se realizará con una clase llamada **`QueryDepotTree`**.

Para la implementación de las estructuras de datos **se deberán utilizar las interfaces proporcionadas por el Equipo Docente** de la asignatura.

El programa deberá contener, además, todas las clases necesarias para:

- Identificar los *parámetros de entrada* pasados al programa.
- Leer un *fichero de consultas* con el que construir el depósito.
- Leer un *fichero de operaciones* que contendrá una lista de operaciones de consulta que se deberán ejecutar sobre el contenido del depósito.
- Realizar el cálculo empírico del coste de la ejecución de las operaciones de consulta.

3.1 Parámetros de entrada.

El programa recibirá tres parámetros de entrada que determinarán su comportamiento. El orden y significado de los parámetros será el siguiente:

1. Selección de la estructura de datos que se desea utilizar, que tendrá únicamente dos valores válidos:
    - L para utilizar la implementación mediante una lista.
    - T para utilizar la implementación mediante un árbol.
2. Fichero de consultas, que contendrá el nombre del fichero de consultas con el que se construirá el depósito.
3. Fichero de operaciones, que contendrá el nombre del fichero de operaciones que se desean realizar sobre el contenido del depósito.

3.2 Estructura del fichero de consultas.

El fichero de consultas contendrá una consulta por línea, con posibles repeticiones.

En nuestro ejemplo anterior, el contenido del fichero de consultas sería el siguiente:

```
caso
casa
casos
caso
casa
cosa
```

La acción esperada por el programa será construir el depósito de consultas (utilizando la estructura de datos determinada por el primer parámetro de entrada) realizando los siguientes pasos:

1. Incrementar en 1 la frecuencia de la consulta `caso` en el depósito.
2. Incrementar en 1 la frecuencia de la consulta `casa` en el depósito.
3. Incrementar en 1 la frecuencia de la consulta `casos` en el depósito.
4. Incrementar en 1 la frecuencia de la consulta `caso` en el depósito.
5. Incrementar en 1 la frecuencia de la consulta `casa` en el depósito.
6. Incrementar en 1 la frecuencia de la consulta `cosa` en el depósito.

3.3 Estructura del fichero de operaciones.

El fichero de operaciones contendrá una operación por línea. Existirán dos tipos de operaciones de consulta, que se identificarán por el primer carácter de la línea:

- Operación de consulta de frecuencia. Los dos primeros caracteres de la línea serán “F” seguido de un espacio y el resto de la línea contendrá el texto de la consulta de la cual se desea conocer su frecuencia dentro del depósito.
- Operación de consulta de sugerencias. Los dos primeros caracteres de la línea serán “S” seguido de un espacio y el resto de la línea contendrá un prefijo de consulta del cual se desea obtener una lista ordenada de aquellas consultas contenidas en el depósito que comienzan por dicho prefijo.

Un ejemplo de fichero de operaciones puede ser el siguiente:

```
F caso
F casos
F coco rallado
S ca
```

Y la acción esperada por el programa será la siguiente:

1. Imprimir la frecuencia de la consulta `caso` en la estructura, que es `2`.
2. Imprimir la frecuencia de la consulta `casos` en la estructura, que es `1`.
3. Imprimir la frecuencia de la consulta `coco rallado` en la estructura, que es `0` (ya que no ha sido añadida por no aparecer en el fichero de consultas).
4. Imprimir la lista de consultas almacenadas (por orden decreciente de frecuencia) que comienzan por el prefijo ca, que, en nuestro ejemplo, sería:

casa	2
caso	2
casos	1

   ya que si dos consultas tienen la misma frecuencia, se devolverán según su orden lexicográfico, de forma que haya una única posible salida ante una misma entrada.

3.4 Medición de tiempos.

Se desea realizar un estudio empírico del coste que emplean para su ejecución los métodos **`getFreqQuery`** y **`listOfQueries`** de las clases **`QueryDepotList`** y **`QueryDepotTree`**. Para ello es necesario medir el tiempo real que tardan en ejecutarse, por lo que se utilizarán los temporizadores que ofrece Java, que devuelven (en un dato de tipo **`long`**) el número de milisegundos transcurridos desde el 1 de Enero de 1970.

La forma de medir el tiempo es almacenar en una variable (de tipo **`long`**) el tiempo antes de ejecutar el método, llamarlo y medir nuevamente el tiempo al finalizar la ejecución:

```
long tInicial = System.currentTimeMillis(); // almacena el tiempo inicial
metodo(); // llamada al método del que quiere medir el tiempo de ejecución
long tFinal = System.currentTimeMillis(); // almacena el tiempo final
long duracion = tFinal – tInicial; // duración
```

Como es muy probable que la ejecución de los métodos tarde menos de un milisegundo, en lugar de medir el tiempo de una única ejecución del método, se aconseja medir el tiempo de varias ejecuciones, para luego dividir el tiempo total entre el número de ejecuciones:

```
long tInicial = System.currentTimeMillis(); // almacena el tiempo inicial
for ( cont = 1 ; cont < rep ; cont++ ) {
  metodo(); // llamada al método del que quiere medir el tiempo de ejecución
}
long tFinal = System.currentTimeMillis(); // almacena el tiempo final
double duracion = ((double)tFinal – (double)tInicial) / (double)rep; // duración
```

De esta forma podemos obtener una precisión inferior a una milésima de segundo.

Estudio empírico del coste temporal (trabajo del estudiante)

3.1 Realice un estudio empírico del coste temporal de los métodos **`getFreqQuery`** y **`listOfQueries`** para la primera aproximación (implementación mediante una lista) en el que se mida cómo varía el coste según el número de consultas diferentes almacenadas en el depósito. Compárelo con el estudio teórico realizado en la pregunta 1.2.

3.2 Realice un estudio empírico del coste temporal de los métodos **`getFreqQuery`** y **`listOfQueries`** para la segunda aproximación (implementación mediante un árbol), en el que se mida cómo varía el coste según el número de consultas diferentes almacenadas en el depósito. Compárelo con el estudio teórico realizado en la pregunta 2.3.

3.5 Salida del programa.

La salida del programa se realizará por la salida estándar de Java y consistirá en:

- La primera línea indicará cuántas consultas diferentes se han almacenado en el depósito de consultas.
- Para cada operación presente en el fichero de operaciones se indicará el resultado y, en la línea siguiente, se indicará su coste.

En nuestro ejemplo (y teniendo en cuenta que el coste indicado no es más que un ejemplo) la salida esperada sería la siguiente:

```
Consultas almacenadas: 4.
La frecuencia de “caso” es 2.
-Tiempo: 6
La frecuencia de “casos” es 1.
-Tiempo: 8
La frecuencia de “coco rallado” es 0.
-Tiempo: 4
La lista de sugerencias para “ca” es:
  “casa” con frecuencia 2.
  “caso” con frecuencia 2.
  “casos” con frecuencia 1.
-Tiempo: 11
```

4. Ejecución y juegos de prueba.

El Equipo Docente proporcionará, a través del curso virtual, unos ficheros de juegos de prueba consistentes en un fichero de consultas, un fichero de operaciones y una salida esperada (teniendo en cuenta que la medición de tiempos variará de una implementación a otra) para que los estudiantes puedan comprobar el correcto funcionamiento del programa.

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

```
java –jar eped2015.jar <estructura> <consultas> <operaciones>
```

siendo:

- **<estructura>** el parámetro que selecciona la estructura de datos que se desea utilizar.
- **<consultas>** el nombre del fichero que contiene las consultas que han de almacenarse en el depósito.
- **<operaciones>** el nombre del fichero que contiene las operaciones que se han de realizar sobre las consultas almacenadas en el depósito.

Para la realización del estudio empírico del coste, los estudiantes podrán crear diferentes ficheros de consultas con un número diferente de consultas únicas, de manera que se pueda medir cómo varía el tiempo que emplean las operaciones según el número de consultas únicas almacenadas en el depósito.

5. 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 estudiantes deberán asistir a una sesión obligatoria de prácticas con su tutor en el Centro Asociado. 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 e incluir el estudio empírico del coste detallado en este documento.
- Implementación en Java de la práctica, de la cual se deberá aportar tanto el código fuente como el programa compilado.