EDA (E.T.S. de Ingeniería Informática)
Curso 2015-2016
Práctica 3. Uso de una Tabla de Dispersión como implementación eficiente de un Map de Imágenes
Parte II: Map de imágenes y su descripción textual

Índice

Objetivos formativos y trabajo previo

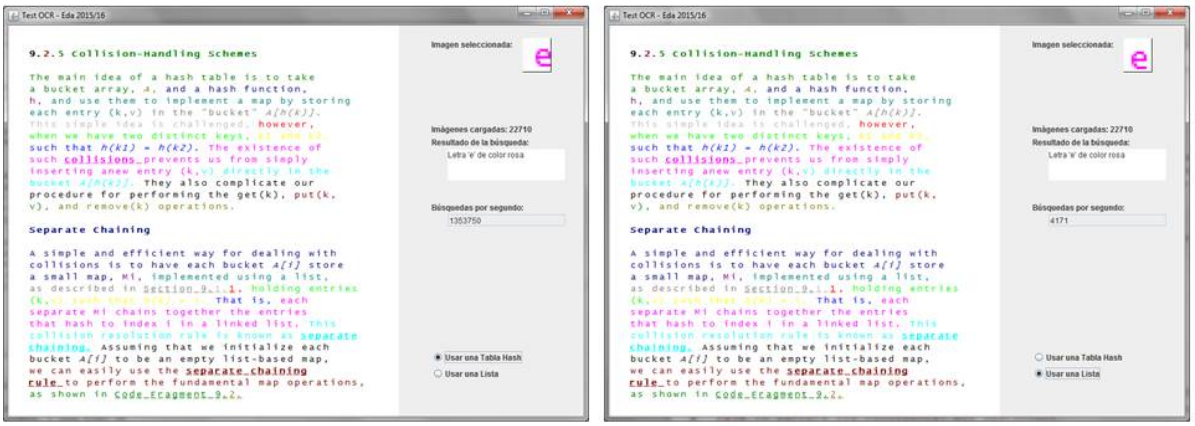
Una vez realizada esta práctica el alumno debe ser capaz de:

- Diseñar la clase `Imagen` que permite, utilizando la clase `Pixel`, una representación básica de una imagen en color.
- Definir e implementar en Java funciones de dispersión sobre los datos que se almacenarán en las tablas mediante la sobrescritura del método `hashCode()` de `Object`. Como ejemplo se definirán funciones de dispersión alternativas sobre la clase `Imagen`.
- Diseñar e implementar código Java para evaluar el comportamiento de las diferentes funciones de dispersión (efectividad en la dispersión y eficiencia) y elegir la más adecuada para el problema que se plantea.
- Diseñar e implementar código Java para comprobar experimentalmente que los costes de las operaciones de un `Map<C, V>` son constantes en tiempo amortizado cuando se implementa con `TablaHash<C, V>` con una adecuada política de redispersión.
- Utilizar el código diseñado en una aplicación de identificación de imágenes en un texto digitalizado, cuyo código se proporciona completo.

Además se reforzarán los objetivos transversales a todas las prácticas de la asignatura y que están relacionados con la calidad de los programas desarrollados: utilización de paquetes para facilitar la organización, reutilización y mantenimiento del software, utilización de los mecanismos de herencia y genericidad que proporciona el lenguaje, elaboración de juegos de prueba para validar código y generación de documentación asociada al código desarrollado. Para aprovechar al máximo la sesión de laboratorio, antes se debe realizar una lectura comprensiva de este boletín y del código de las clases que se proporcionan a través de PoliformaT.

1. Identificación de letras en un texto digitalizado

Actualmente es de indudable interés la posibilidad de digitalizar textos; es decir, de identificar en una imagen (de forma automática) los símbolos o caracteres pertenecientes a un determinado alfabeto; es lo que se suele denominar reconocimiento óptico de carácteres (`OCR`, de las iniciales en inglés *Optical Character Recognition*). Un caso sencillo de OCR se tiene cuando la imagen del documento corresponde a texto tipografiado y sin errores. Supongamos que el usuario se puede mover con el ratón sobre el texto seleccionando una imagen; una vez seleccionada dicha imagen se trata de compararla con las que se tienen como patrones para identificarla. Para realizar esta búsqueda de forma eficiente se utilizará un `Map<Imagen, String>` implementado como una Tabla de Dispersión; cada entrada del `Map` tiene como clave una imagen y como valor su descripción textual. En la figura siguiente se muestra dos capturas de ejecución de la aplicación `TestOCR` en las que se observa la identificación de la *Letra e de color rosa*. Se puede ver que utilizando la implementación de Tabla Hash se pueden realizar 1.353.750 búsquedas por segundo mientras que utilizando una lista sólo 4171.



El objetivo de esta parte de la práctica es completar la clase `Imagen`, tipo de las claves del `Map`. Los métodos imprescindibles para que objetos de tipo `Imagen` puedan instanciar las claves de `TablaHash<C, V>`, implementación eficiente de `Map<C, V>`, son `equals(Object o)` y `hashCode()`. En lo que sigue se plantea una serie de actividades que conducirán a la elección de una función de dispersión efectiva para el problema planteado.

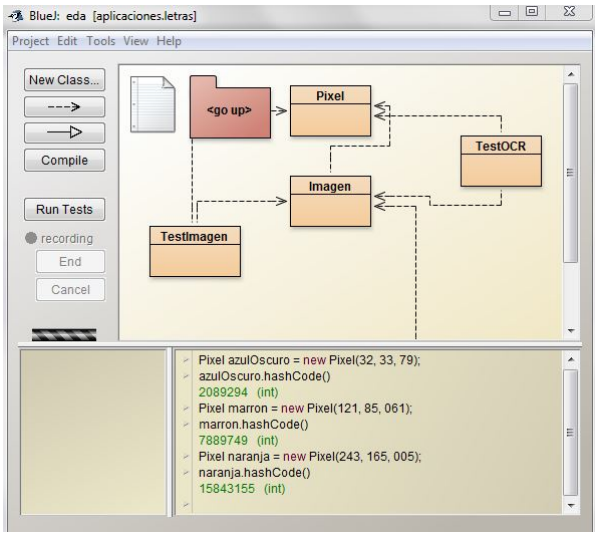
2. Las clases de la aplicación `Pixel` e `Imagen`

Los datos con los que se va a trabajar, que serán las claves del `Map<C, V>`, son imágenes; cada imagen se representará básicamente como una matriz de píxeles, y cada pixel se representa mediante su código RGB (*Red Green Blue*) con 256 valores de intensidad para cada componente. En la aplicación final (`TestOCR`) las imágenes representan letras de diferentes colores y estilos (subrayado, negrita, etc.).

Actividad 1: las clases `Pixel` e `Imagen`

El alumno deberá realizar las siguientes acciones:

- Crear un nuevo paquete, denominado `aplicaciones.letras`, que contendrá la aplicación destinada a la identificación de letras. En él añadir la clase `Pixel`, la clase incompleta `Imagen` y `TestImagen`.
- Abrir la clase Pixel y observar la definición del método `hashCode()`. Crear, en el CodePad de BlueJ, algunas instancias de esta clase, consultando sus valores de hashing.



- Implementar los métodos privados para la obtención del valor de Hash, `hashCode()`, de la clase `Imagen`. Puesto que el cómputo de la función de dispersión puede ser costoso, al igual que se ha hecho en la clase `Pixel`, se guardará en un atributo `valorHash`, valor que se inicializará siempre que se cree una `Imagen`. La implementación debe permitir utilizar diferentes funciones de dispersión (funciones que generan un valor numérico a partir de una imagen). Se deben considerar al menos cuatro funciones de dispersión haciendo intervenir todos los valores de píxeles o sólo unos pocos (los centrales de cada imagen) y sumando los valores o añadiendo ciertos pesos, a saber:

(a) suma de los `hashCode()` de todos los píxeles.

(b) suma ponderada (por la constante `Pixel.BASE`) de los `hashCode()` de todos los píxeles.

(c) sumar los `hashCode()` de los 3x3 píxeles centrales.

(d) suma ponderada (por la constante `Pixel.BASE`) de los 3x3 píxeles centrales.

- Implementar el método `equals(Object o)` de la clase `Imagen`. Nótese que el uso del atributo `valorHash` permite ahorrar algunos cálculos, ya que sólo se compararan componente a componente cuando el valor del atributo `valorHash` de ambas sea igual.

- Comprobar la corrección del código, utilizando la clase `TestImagen` que se proporciona en PoliformaT. Nótese que aunque cualquier recorrido de la matriz de bits en el que se pondere adecuadamente es una solución correcta, el test que se proporciona sólo comprueba la corrección de una implementación en la que el recorrido por filas/columnas se realiza en sentido ascendente.

3. Estudio de la efectividad de la función de dispersión

Dos son los factores que intervienen en el coste de las operaciones `insertar(E e)`, `recuperar()` y `eliminar()` de una Tabla de Dispersión: el coste del cálculo del `indiceHash()` asociado al elemento a tratar, `e`, y el coste de la búsqueda dinámica en la *ListaConPI* `elArray[indiceHash(e)]`. Según ya se ha estudiado en las clases de teoría, las propiedades de la Función de Dispersión son las siguientes:

- Efectividad: debe producir una dispersión efectiva de las claves en las distintas listas.

- Eficiencia: debe de poder ser calculada de forma eficiente, ya que las operaciones básicas sobre la tabla requieren su cálculo.

- No es inyectiva: se pueden producir colisiones.

Para que se puedan implementar las operaciones en tiempo constante el cálculo de `indiceHash()` debe realizarse, a su vez, en tiempo constante y, para ello, las longitudes de las listas deben estar acotadas por una constante, por ejemplo un valor menor que 2. Para estudiar la efectividad de la función de dispersión utilizada se calculará la desviación típica de las cubetas y se analizará el histograma de ocupación de las mismas que describe la forma en la que la función consigue repartir los elementos entre las cubetas, tal y como se ha hecho en la primera parte de la práctica.

Actividad 2: la efectividad de las funciones de dispersión

Para comprobar la efectividad de las cuatro funciones de dispersión definidas se deberá:

- Añadir en el paquete `letras` la clase incompleta `EvaluaFuncionDispersion`. Completarla para obtener los histogramas de ocupación de las cubetas de las Tablas Hash creadas con todas las imágenes que se encuentran en el fichero binario `Letras.img`. El método privado `cargarImagenes(int fdis)` lee las imágenes y las guarda en una Tabla Hash utilizando la función de dispersión que se indica (`fdis`). El fichero `Letras.img` contiene del orden de 23000 imágenes 11×13 que representan todas las letras, los dígitos, los símbolos de puntuación, etc., en diversos colores y estilos. Debido al tamaño del fichero, casi 10 Mb no se copiará a la carpeta del alumno sino que se abrirá directamente en Poliformat (utilizando las clases Java `URL` y `URLConnection` de `java.net`).

- Utilizando la herramienta `gnuplot`, dibujar los histogramas obtenidos para cada función de dispersión:

  ```
  gnuplot> plot "histogXXXX.txt" using 1:2 with boxes
  ```

  donde `histoXXXX.txt` hace referencia al fichero que resulta cuando se utiliza la función de dispersión `XXXX`.

  A la vista de los resultados obtenidos se elegirá la función de dispersión más adecuada para la aplicación que se propone, que será la que se utilizará en la siguiente actividad.

4. Uso de la aplicación `TestOCR`

Actividad 3: uso de la aplicación `testOCR`

Para finalizar la práctica se comprobará el correcto funcionamiento de la aplicación `TestOCR`. Para ello se deberá:

- Añadir la clase `TestOCR` al paquete `aplicaciones.letras`;
- Copiar la imagen `Pagina.bmp` a la carpeta `letras`
- Ejecutar el `main(String[] args)` de la clase `TestOCR`.

Nótese que la primera acción que tiene lugar es la carga de las imágenes desde el fichero binario `Letras.img`. Como ya se ha comentado, estas imágenes se representan en un `Map<Imagen, String>`, la clave es la propia imagen y su valor es su descripción textual, p.e. `"Letra 'e' de color rosa"`; como implementación de esta interfaz se puede utilizar una `TablaHash<Imagen, String>` o una Lista enlazada de pares `Imagen-String`. A continuación se abre la imagen que contiene el texto de ejemplo, `Pagina.bmp`, y se puede interactuar con la aplicación moviendo el cursor y seleccionando para su identificación algunos de los caracteres de la imagen. Obsérvese la diferencia en la rapidez para identificar las letras según se utilice una u otra implementación del Map.

5. Actividad voluntaria: estudio de la eficiencia de las Tablas de Dispersión

A continuación se debe evaluar empíricamente el coste temporal de la búsqueda en una tabla de dispersión. Para ello se utilizarán las mismas imágenes que se han utilizado en la actividad anterior y se tomarán medidas de tiempos que permitan establecer el coste empírico promedio de la búsqueda de una imagen en una colección. La medida del tiempo de ejecución de un segmento de código se puede realizar utilizando, como en la práctica anterior, el método `static long nanoTime()` de la clase `java.lang.System`, que retorna el valor actual del temporizador más preciso del sistema en nanosegundos.

Actividad 4: análisis del coste de la búsqueda

Esta actividad consiste en:

- Descargar de Poliformat la clase incompleta `EvaluaCosteBusqueda` en el paquete `letras` y completarla para analizar la eficiencia de la búsqueda en una `TablaHash<C, V>` utilizando como función de dispersión la que se haya decidido en la actividad anterior. Para ello se deben crear diferentes *TablasHash* con diferente número de imágenes y estimar en cada caso el número de búsquedas por unidad de tiempo que se pueden realizar o el tiempo promedio de realizar una búsqueda. Se debe asegurar que el número de búsquedas con éxito y sin éxito sea equiprobable.
- Dibujar la gráfica de costes utilizando `gnuplot`.

Al finalizar la práctica, y completadas todas las actividades, la estructura de proyectos, paquetes y ficheros realizada deberá ser la siguiente: