TRABAJO DE TIC

EFICACIA EN FUENTES EXTENDIDAS

Abel Mayorga González
Alberto Rodríguez Gómez

EFICACIA EN FUENTES EXTENDIDAS

El trabajo consiste en crear un programa que, dada una fuente binaria y una de las probabilidades, p, calcule:

• Las fuentes extendidas F^k hasta un orden *k* que sea computable con las herramientas de hardware y software utilizados.

• Para cada fuente extendida F^k calcular un código óptimo binario C_k y su eficacia.

• Explicar los resultados obtenidos para las probabilidades p= $\frac{1}{n}$ con $2 \leq n \leq 10$.

Reparto del trabajo

El trabajo nos lo hemos repartido de la siguiente manera: Abel se ha encargado de la parte de programación y diseño del programa, y Alberto se ha encargado de la parte teórica, explicación de algoritmos y resultados obtenidos.

Explicación del algoritmo

Para este programa hemos creado una serie de clases: *Racional*, *Arbol*, *Fuente*, *ParejaPcodigo*, *TablaCodigo*.

La clase *Racional* nos permite un completo tratamiento de los números racionales, básico para el manejo de fracciones. Cada número racional es un objeto de esa clase.

La clase *Arbol* es la clase que modela los árboles necesarios para obtener el código mediante Huffman.

La clase *ParejaPcodigo* contiene una probabilidad con su correspondiente palabra código, y el conjunto de todas ellas es un objeto de la clase *TablaCodigo*.

Funcionamiento de la clase *Fuente*:

La fuente se construye a partir de un número racional. Esta fuente se va extendiendo mediante el método *extenderFuente* hasta que el propio ordenador no lo permita más. Si la probabilidad es $p = \frac{1}{q}$, se puede calcular hasta un orden k tal que

$$q^k < 2^{63} - 1$$

Para extender la fuente, se multiplica cada una de las probabilidades por la probabilidad base, *p*, que viene dada por el usuario, y por la probabilidad *1-p*. Se obtiene la lista de probabilidades de la fuente extendida.

Cálculo del código óptimo:

Para calcular los códigos óptimos de las fuentes extendidas utilizamos el método de Huffman, y aprovechamos el API (Interfaz de Programación de Aplicaciones) de Java que implementa métodos para ordenar listas de menor a mayor automáticamente.

Como nuestro trabajo no es el del método de Huffman, no hemos implementado la versión más eficiente. Primero, creamos una lista de árboles que son todo hojas (las probabilidades), y se ordenan de menor a mayor. Después sumamos los valores de los dos primeros árboles, que se sacan de la lista, y creamos un nuevo árbol cuya raíz sea dicha suma y los hijos sean las anteriores probabilidades. Este nuevo árbol se mete en la lista, que se reordena, y, recursivamente, volvemos a hacer lo mismo que en el paso anterior hasta tener una lista con un solo árbol, cuya raíz será de probabilidad 1.

Para obtener el código, calculamos el número cuyos bits constituyen el código para cada probabilidad. Inicialmente es 0. Recursivamente, calculamos el código del subárbol izquierdo y del derecho, y los juntamos. La forma de calcular el código es la siguiente: Cuando bajamos al subárbol izquierdo, hacemos un desplazamiento lógico a la derecha, de forma que el número (viéndolo en bits) sea el mismo, pero con un 0 más al final. Cuando bajamos al subárbol derecho, hacemos un desplazamiento lógico a la derecha y sumamos 1, de forma que el número (viéndolo en bits) sea el mismo, pero con un 1 más al final. Para terminar, cuando se llega a una hoja, pasamos ese número a una cadena de bits con el método toBinaryString, y nos quedamos con tantos bits finales como marque la profundidad de la hoja.

Cálculo de la entropía:

Para calcular la entropía, guardamos el orden de la fuente y la entropía de la fuente de orden 1, y aplicamos la ecuación:

$$H(F) = \sum_{i=1}^{m} p_i \log_2 \left(\frac{1}{p_i}\right)$$

Dado que queremos calcular la entropía de una fuente extendida, lo hacemos aplicando:

$$H(F^k) = kH(F)$$

Al no poder operar con logaritmos en base 2, operamos con logaritmos neperianos, dividiendo el logaritmo neperiano del número que nos piden entre el logaritmo neperiano de 2, quedando la fórmula:

$$H(F) = \sum_{i=1}^{m} p_i \left(-\frac{\ln(p_i)}{\ln(2)} \right)$$

Cálculo de la longitud del código:

Para calcular la longitud del código, se suman todos los productos de cada probabilidad por la longitud de su palabra código.

$$l(C) = \sum_{i=1}^{m} p_i l(c_i)$$

Siendo c_i la palabra código correspondiente a p_i.

Cálculo de la eficacia:

La eficacia se calcula mediante la ecuación:

$$\eta(C) = \frac{H(F)}{\log_2(q)\, l(C)}$$

Pero como en nuestro programa q=2, y $\log_2 2 = 1$, únicamente tenemos que dividir la entropía entre la longitud del código.

Problemas encontrados:

Hemos tenido ha sido al operar con números grandes, ya que, al haberlos declarado enteros, se nos salían de rango fácilmente. Esto lo arreglamos declarándolos de tipo *long,* que admiten números más grandes. Por último añadimos la restricción de que se permite calcular una fuente de probabilidad $p = \frac{1}{q}$ de orden k si:

$$k \log(q) < 2^{63} - 1$$

Otro problema ha aparecido al calcular fuentes de orden grande, mayor que 14. Como cada fuente de orden k tiene una lista de 2^k números, parece ser que al calcula el código recursivamente, se desborda la pila de recursión. Por tanto, hemos limitado el orden máximo a 14.

También hemos tenido problemas con la interfaz gráfica, ya que cosas que funcionaban en Linux no funcionaban en Windows y viceversa, problema que aparece a veces dependiendo de la configuración del ordenador, del sistema operativo y de la máquina virtual usada, por lo que nos ha sido imposible depurarlo para todas las plataformas.

Por último, parece ser que no se libera toda la memoria correctamente al borrar el panel calculado. No sabemos por qué ocurre ya que de eso se ocupa el recolector de basura de Java. Por tanto, si se observa que se ralentiza el ordenador al calcular varias fuentes seguidas, se recomienda cerrar el programa y volverlo a abrir.

Otras consideraciones:

Parte de la GUI (Interfaz Gráfica de Usuario) y la clase *Racional* las hemos reutilizado de otras prácticas hechas en otra asignatura, adaptándolas a los requisitos del programa, así como el método de Huffman utilizado, que lo habíamos escrito en el lenguaje funcional Scheme para otra asignatura, el cual adaptamos al lenguaje Java y a las necesidades de la práctica. Por lo tanto, la mayor parte del trabajo ha sido el cálculo correcto de las fuentes extendidas y la depuración de errores. Una parte muy importante también ha sido el desarrollo de la interfaz gráfica, que aunque parte ha sido reutilizada de otras prácticas, la mayoría tuvo que ser diseñado de nuevo, como el dibujo del gráfico.

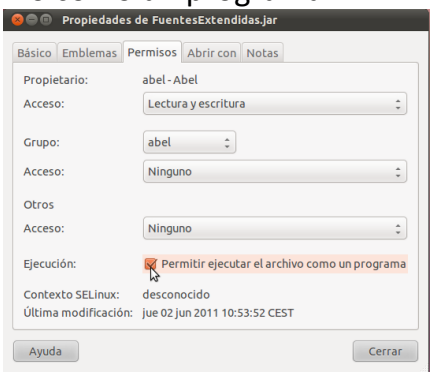
Hemos elegido el lenguaje de programación Java porque es multiplataforma, es decir, un mismo archivo ejecutable funciona correctamente en todos los sistemas operativos. Además proporciona una interfaz de programación de aplicaciones inmensa, mucho más grande que otros lenguajes, descargando de trabajo al programador.

EXPLICACIÓN DEL PROGRAMA

El requisito para el correcto funcionamiento del programa es tener instalado el *Java Runtime Environment 1.6*. Puedes descargar la última versión a fecha de 2 de Junio de 2011 desde: http://www.oracle.com/technetwork/java/javase/downloads/jre-6u25-download-346243.html

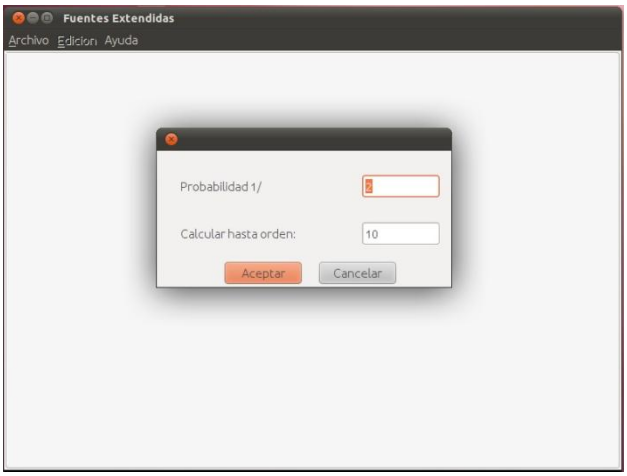
Se incluye un fichero comprimido en zip, llamado FuentesExtendidas.zip. Contiene 2 ficheros, FuentesExtendidas.jar, el cual es el ejecutable de java, y PracticaTIC.zip, el cual es el directorio del proyecto de eclipse que usamos para desarrollar la práctica. Si se desea ver el código fuente, basta con descomprimir PracticaTIC.zip y dentro de la carpeta src se encuentra todo el código.

Si se ejecuta desde Windows, basta con hacer doble clic sobre el fichero jar. Si se ejecuta desde Linux, antes hay que marcarlo como ejecutable. Para ello, hay que hacer clic con el botón derecho sobre el archivo y entrar en propiedades. Dentro, ir a la pestaña “Permisos” y marcar la casilla que pone “Permitir ejecutar el archivo como un programa”.



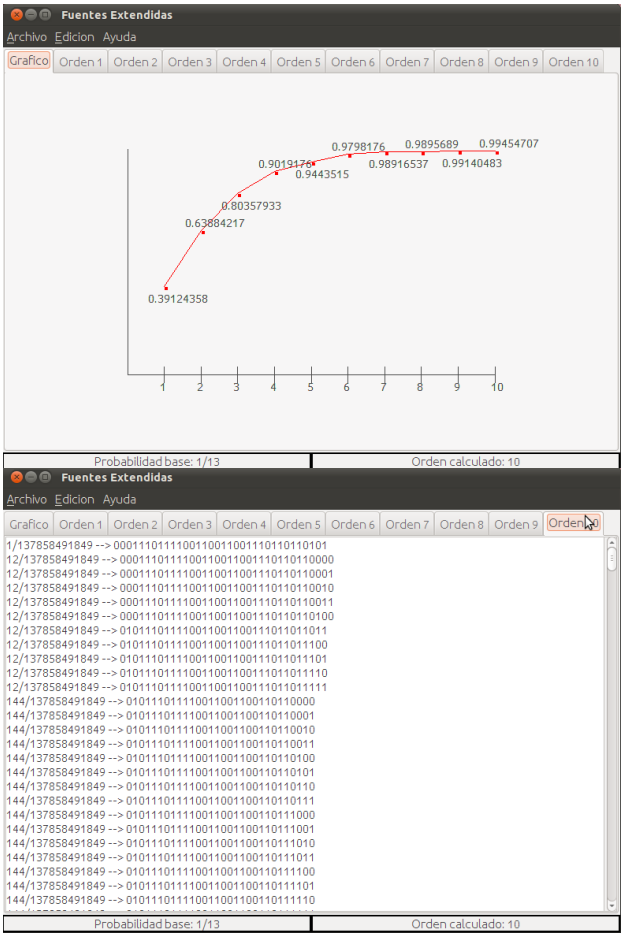
Después de hacer esto, para ejecutarlo hacemos clic con el botón derecho y pinchamos en “ejecutar con” seguido de la máquina virtual de java que tengamos instalada.

Al ejecutar el programa se abre una ventana en blanco con tres pestañas, para crear una nueva fuente hay que pinchar en la que pone “Archivo”, y luego en “Nuevo”. Se abrirá otra ventana donde habrá que escribir dos números, el primero de ellos es el denominador de la probabilidad *p*, el segundo, el orden hasta el que queremos calcular fuentes extendidas.



Al hacer clic en “Aceptar”, nos aparecerán una serie de pestañas. La primera de todas muestra una gráfica cuyo eje x representa el orden de la fuente extendida, y el eje y representa la eficacia, de forma que se pueda ver la mejora de la eficacia. Las demás pestañas muestran las probabilidades de cada una de las fuentes extendidas y el código óptimo para dicha fuente.

Para poder evaluar otra probabilidad basta con volver a hacer “clic” en “Archivo” → “Nuevo”.



Resultados obtenidos:

Para la probabilidad p=$\frac{1}{2}$, la fuente es equiprobable, por tanto la eficacia es 1 para todas las fuentes, y la gráfica es una línea recta paralela al eje x.

Para la probabilidad p=$\frac{1}{3}$, las eficacias son bastante altas, pero se aprecia un ligero aumento desde la fuente original hacia otras más altas.

A partir de la probabilidad p=$\frac{1}{4}$, las eficacias de las fuentes van creciendo de la misma manera, bruscamente desde la fuente original hasta la fuente de orden 3, y a partir de ahí suavemente, pero a medida que vamos disminuyendo la probabilidad, van teniendo cada vez menos eficacia, pasando de eficacia 1 en la fuente de probabilidad $\frac{1}{2}$ hasta eficacia 0,46 en la fuente de probabilidad $\frac{1}{10}$. Esto se debe a la disminución de la entropía de la fuente al disminuir la probabilidad de que aparezca uno de los símbolos.