Proyecto LE12:
Compresión de imágenes en escala de grises con modo run

Tutores:
Gadiel Seroussi y Álvaro Martin

Alexis Baladón Ferreira de Araújo,
Juan Ignacio Fernández Desantis

Aplicaciones de la Teoría de la Información al Procesamiento de Imágenes 2021
Núcleo de Teoría de la Información
Facultad de Ingeniería. Universidad de la República
Montevideo, Uruguay

1 Introducción

Dentro de los algoritmos de compresión de imágenes sin perdida ”LOCO-I”, parte del estándar JPEG-LS, destaca por su baja complejidad y alto rendimiento en términos de velocidad y capacidad de compresión [1].

El algoritmo se basa en recorrer linealmente los pixeles de la imagen, realizando predicciones sobre el valor del próximo utilizando predictores basados en modelos estadísticos.

Una de las claves de su eficiencia es el uso de propiedades de ”imágenes naturales” [2], como la suavidad, que plantea que el nivel en pixeles contiguos varía levemente. Como resultado, LOCO-I es capaz de clasificar contextos de pixeles en grupos reducidos, y plantea una modalidad denominada ”run”, que se activa al atravesar un área suave (con variaciones muy leves en el brillo de los pixeles), cubriendo las deficiencias del predictor original en este tipo de zonas.

En este informe se describe la implementación de una variante del algoritmo LOCO-I en el lenguaje de programación C. Primero, se describe el problema y se hace un breve marco teórico, mostrándose similitudes y diferencias con el algoritmo original. En la sección 4 se muestra la estructura general del programa, así como el procedimiento de compilación y modo de uso.

Finalmente, se analizan los resultados en la base de imágenes de prueba y se comparan las tasas de compresión de los dos algoritmos, obteniendo resultados similares a JPEG-LS, aunque ni en el mejor de los casos obteniendo una menor tasa de compresión. Se comprueba además el impacto positivo aunque circunstancial del modo de run.

2 Problema

El programa debe ser capaz de comprimir y descomprimir imágenes en escala de grises en formato PGM en modo P5. En este se cuenta con un encabezado detallando el modo, tamaño de las imágenes y el valor máximo que puede tomar un pixel (nos restringiremos a valores no negativos menores a 256). Los valores de pixel se representan como una cadena de números en binario inmediatamente después del encabezado.
Para la evaluación se contará con un set de imágenes provisto por los docentes del

curso. Además, se contará con tablas de datos sobre la tasas de compresión logradas por el algoritmo original de JPEG-LS en las mismas imágenes. A partir de estas se determinará el valor óptimo de los parámetros del programa, que se definirán como parámetros por defecto.
Además se buscará contestar un conjunto de preguntas que orientarán el desarrollo del programa, y ayudarán a reflexionar sobre la importancia de características como el modo de run.

3 Algoritmo

El algoritmo a implementar utiliza el mismo predictor que JPEG-LS [1] pero un modelo de contextos diferente. Se etiquetan las clases de contextos con el extracto definido a partir de la ecuación

$$f(C) = 8 * Q(X) + T$$

Donde:

$$Q(X) = \left\lfloor \frac{X}{2^{10-s}} \right\rfloor \tag{1}$$

$$X = |c - \hat{x}| + |b - \hat{x}| + |a - \hat{x}| \tag{2}$$

$$T = (c > \hat{x}) \cdot (b > \hat{x}) \cdot (a > \hat{x}) \tag{3}$$

$$\hat{x} = MED(a, b, a + b - c) \tag{4}$$

$\_ \cdot \_$ es el operador binario de concatenación de bits.

$MED(X)$ es la mediana del vector X.

Notar que T es un número de tres bits (por ser la concatenación de tres bits) y el sumando $8 * Q(X)$ equivale a un corrimiento de tres bits a la izquierda, por lo que el extracto $f(C)$ es la concatenación en binario de $Q(X)$ y T, o equivalentemente

$$f(C) = Q(X) \cdot T \tag{5}$$

Respuesta P1

El nivel de actividad X puede representarse usando 10 bits. En efecto, cada sumando (no negativo) es un número entre 0 y 255 por ser la diferencia entre valores de pixel, y el máximo valor que puede alcanzar la suma es 765.

Al igual que en JPEG-LS se codifican los errores de predicción con códigos de Golomb PO2 cuyo parámetro k se calcula a partir de las estadísticas A y N de cada clase de contexto.

Respuesta P2

El parámetro s define el grado de cuantización del nivel de actividad. Se puede ver a partir de 1 que $Q(X)$ se forma con los s bits más significativos de X; por lo tanto $f(C)$ tendrá $s + 3$ bits, generando como máximo 2^{s+3} posibles valores para las clases de contexto.

3.1 Modo de RUN

El modo RUN permite aumentar la tasa de compresión en imágenes con secuencias de pixeles con mismo valor. Si se cumple la condición $a = b = c = d$, el compresor entra al modo y codifica el número de pixeles consecutivos con el mismo valor con un código GPO2 de parámetro $k = 3$. La habilitación de este modo es un parámetro del compresor y se debe indicar explícitamente.

Respuesta P3

De las ecuaciones 1, 2 y 3 se desprende inmediatamente que si se cumple la condición de entrada al modo RUN $X = 0$ y $T = 0$, ya que $\hat{x} = a = b = c = d$; por lo tanto $Q(X) = 0$ y $f(C) = 0$. Sin embargo, que el extracto sea nulo no es condición suficiente para entrar al modo ya que en el caso $\hat{x} = a = b = c \neq d$ se tendría $f(C) = 0$, pero la condición de entrada no se cumple.

4 Implementación

4.1 Estructura

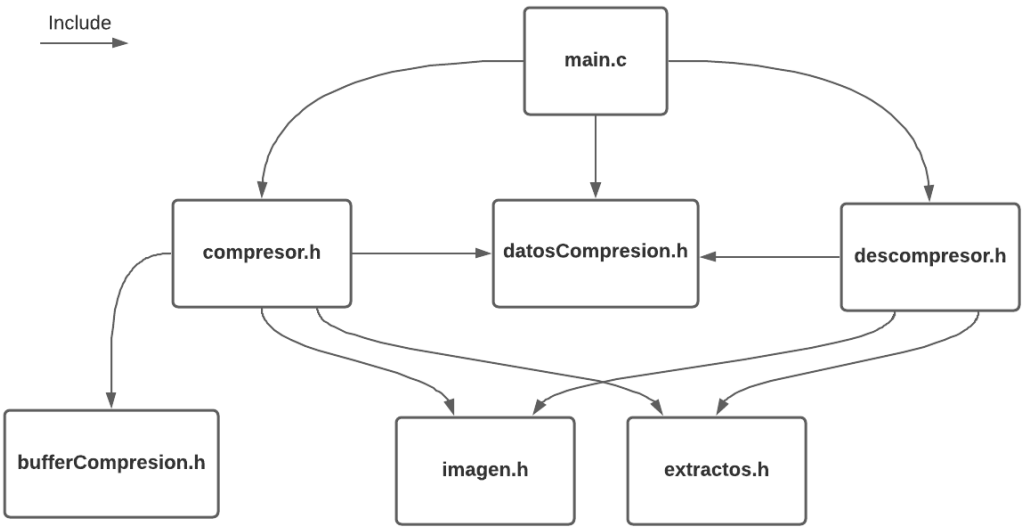

Fig. 1. Grafo de dependencia de módulos

main.c: Aquí se leen las flags del usuario, donde se decidirá que modalidad ejecutar y bajo qué parámetros. Posteriormente se imprimirán los datos de la compresión/descompresión

compresor.h: Aquí se encuentra el esqueleto del algoritmo de compresión LOCO, donde se solucionan problemas únicos del compresor y se utilizan otros módulos para resolver tareas también realizadas por el descompresor.

descompresor.h: Aquí se encuentra el esqueleto del algoritmo de descompresión LOCO, donde se solucionan problemas únicos del descompresor y se utilizan otros módulos para resolver tareas también realizadas por el compresor.

datosCompresion.h: Aquí se define la estructura que contiene los datos resultantes de la compresión (tasa de compresión, cantidad de pixeles de la imagen y bytes comprimidos) que serán impresos.

bufferCompresion.h: Aquí se define el buffer que almacena los caracteres

comprimidos. Para no almacenar mucha memoria se imprimen los caracteres en el archivo comprimido después de sobrepasar una cantidad fija.

imagen.h: Aquí se define la estructura que almacena los pixeles de la imagen. Con el fin de no almacenar memoria innecesaria, se almacenan solo dos filas de pixeles a la vez. El número 2 se debe a que es necesario recordar los pixeles de la actual fila siendo comprimida y la anterior para conocer el contexto de un pixel. Se define además la estructura DatosCabezal que contiene datos del cabezal PGM de la imagen original.

extractos.h: Aquí se definen las estructuras correspondientes a los datos de los extractos y aquella que los contiene. Además, se realizan operaciones que involucran los datos de los mismos.

Debido al dominio no negativo e inyectividad de la función f se decide implementar la colección de extractos a modo de arreglo, donde el valor de $f(C)$ determina el índice del mismo. Esta decisión se justifica debido a que un arreglo es una de las estructuras de datos de más rápido acceso, y de bajo consumo de memoria.

4.2 Encabezado de archivo comprimido

Se decidió mantener el encabezado original del estándar PGM, agregando el valor del parámetro s seguido de un fin de línea, seguido del bit de habilitación del modo RUN (1 modo RUN, 0 modo normal) seguido de otro fin de línea. Esto simplificará la escritura y lectura del cabezal por en el compresor y descompresor, sin afectar significativamente el tamaño del archivo comprimido.

Debido al modo en que codifica el programa, el archivo de salida debería ser idéntico al de entrada.

Ejemplo, dados los parámetros:

- $s = 5$
- $run = true$
- $imagen =$ "*imagenOriginal.pgm*"

La relación entre los cabezales de las imágenes y el archivo comprimido se muestra en la figura 2.

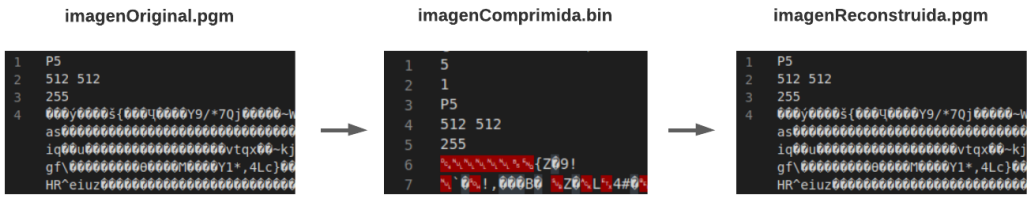

Fig. 2. Cabezal de archivo comprimido y descomprimido

4.3 Compilación

El programa fue creado para funcionar en el sistema operativo Linux, utilizando el compilador *gcc*. Para compilar, se debe situar en la carpeta donde se encuentra el makefile (*./TareaGrupo3LOCO/Programa* en el repositorio) y escribir desde la consola el comando:

```
$ make
```

4.4 Modo de uso

Para correr el archivo *main.c* desde la consola, debe utilizarse:

Al descomprimir:

```
$ ./main -d inputFile outputFile
```

Al comprimir:

```
$ ./main -c [modalidad] [-s valorS] [-m] inputFile outputFile
```

Donde:

- -c habilita la compresión. Si recibe 1 comprime en modo de run, Si recibe un 0 o nada lo hace en modo normal.
- -d habilita la descompresión.
- -s *number* indica el valor del parámetro *s*. Si no hay parámetro se tomará un valor por defecto.
- -m silencia el output de la consola (excepto para errores)
- inputFile indica el archivo de entrada
- inputFile indica el archivo de salida

Nota: Valores encerrados entre '[]' son opcionales.

Ejemplos:

```
$ ./main -c 1 -s 10 "imagen.pgm" "imagen.bin"
$ ./main -d "imagen.bin" "imagen.pgm"
```

5 Resultados

5.1 Obtención de datos

Para la recolección de datos y pruebas se utilizaron dos scripts ubicados dentro de la carpeta (*./Programa/ObtencionDeDatos/*). Ambos utilizan todas las imágenes ubicadas en la carpeta *imagenes*, estos son:

Tasas de compresión:

Para esto fue creado un nuevo main, encargado de recolectar tasas de compresión variando s y modo de run, además de obtener el promedio de las mismas para cada valor de s. Asimismo, se crea un nuevo makefile para no utilizar la flag *'-g'* del compilador, y no compilar funcionalidades del descompresor.

Para obtener los datos, debe correrse el script *datos.sh*, lo cual generará, dentro de la carpeta *resultados* del mismo directorio un archivo de formato *csv* con sus columnas separadas por espacios.

Para ejecutar el script, debe escribirse en la consola:

```
$ bash datos.sh
```

Test de compresión sin pérdida:

Para esto solo fue necesario crearse un nuevo main y un makefile, para utilizar distintas flags.

Para realizar este test, debe correrse el script *test.sh*, lo cual generará, dentro de la carpeta *resultados* del mismo , un archivo con texto plano, mostrando para cada imagen si el programa es capaz de comprimirlos y descomprimirlos sin que hayan diferencias entre ambos. Esto se logra con la ayuda del comando *cmp* de Linux, el

cual determina si dos archivos, en este caso el original y descomprimido, son idénticos.

Para ejecutar el script, debe escribirse en la consola:

```
$ bash test.sh
```

5.2 Imágenes de prueba

En las tablas 1 y 2 se muestran las tasas de compresión obtenidas para distintos valores del parámetro *s*, sin y con habilitación del modo RUN. Las tasas fueron calculadas a partir del tamaño de las imágenes sin contar cabezales.

Table 1
Tasas de compresión por imagen con modo RUN activado

	4	5	6	7	8	9	10
imagen							
baloons.pgm	0.063	0.063	0.062	0.062	0.062	0.061	0.061
barbara.pgm	0.688	0.672	0.663	0.660	0.661	0.663	0.664
bike.pgm	0.601	0.590	0.583	0.581	0.579	0.579	0.579
faxballsL.pgm	0.152	0.150	0.149	0.147	0.145	0.145	0.144
kodim07.pgm	0.539	0.532	0.526	0.522	0.520	0.520	0.520
kodim20.pgm	0.461	0.454	0.450	0.450	0.448	0.449	0.449
kodim24.pgm	0.633	0.623	0.615	0.612	0.611	0.613	0.614
tools.pgm	0.713	0.706	0.704	0.703	0.703	0.704	0.705
womanc.pgm	0.600	0.589	0.579	0.579	0.577	0.576	0.576

Table 2
Tasas de compresión por imagen sin modo RUN desactivado

	4	5	6	7	8	9	10
imagen							
baloons.pgm	0.205	0.196	0.192	0.191	0.191	0.191	0.191
barbara.pgm	0.688	0.672	0.663	0.659	0.660	0.662	0.664
bike.pgm	0.599	0.589	0.582	0.579	0.577	0.577	0.577
faxballsL.pgm	0.198	0.196	0.195	0.193	0.191	0.191	0.191
kodim07.pgm	0.535	0.528	0.522	0.517	0.517	0.516	0.516
kodim20.pgm	0.472	0.466	0.461	0.460	0.457	0.458	0.458
kodim24.pgm	0.651	0.639	0.628	0.623	0.619	0.619	0.620
tools.pgm	0.713	0.706	0.705	0.703	0.704	0.705	0.706
womanc.pgm	0.599	0.588	0.578	0.578	0.576	0.575	0.575

5.3 Determinación del parámetro s

Se utiliza la media simple de las tasas de compresión en la base de imágenes con modo RUN habilitado como medida de desempeño global del algoritmo. En la figura 3 se grafican las tasas en función del parámetro *s*.

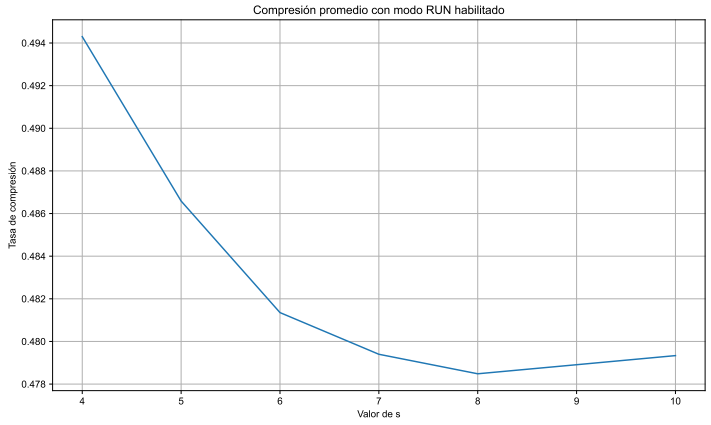

Fig. 3. ”Tasas de compresión promedio en la base de imágenes”

Observando esta gráfica es fácil ver que se alcanza un mínimo para el valor de $s = 8$. En vista de que son pocos los casos en que otros parámetros obtienen menores tasas de compresión, y cuando lo hacen la diferencia es ínfima; agregando que tampoco se observa mayor beneficio en usar otro parámetro al tener en cuenta los resultados sin modo de run, se concluye que el valor de s recomendado para este programa es 8. Consecuentemente, se decide utilizar este parámetro por defecto en el programa y utilizarlo para los experimentos y análisis posteriores.

5.4 Impacto del modo RUN

Se observan mejor tasas de compresión con modo RUN habilitado para 5 de las 9 imágenes de prueba, aunque la diferencia no es importante salvo en casos particulares. Incluso, hay casos donde la habilitación del modo perjudica la capacidad de compresión.

Fig. 4. ballons.pgm

Al examinar en particular a las imágenes *baloons.pgm* 4 y ”faxballsL.pgm” 5, aquellas con mayor suavidad, se presenta una diferencia absoluta de 0.129 y 0.046 respectivamente entre tasas de compresión. En cambio, la compresión de imágenes sin modo de run activado logran superar en el mejor de los casos por 0.002.

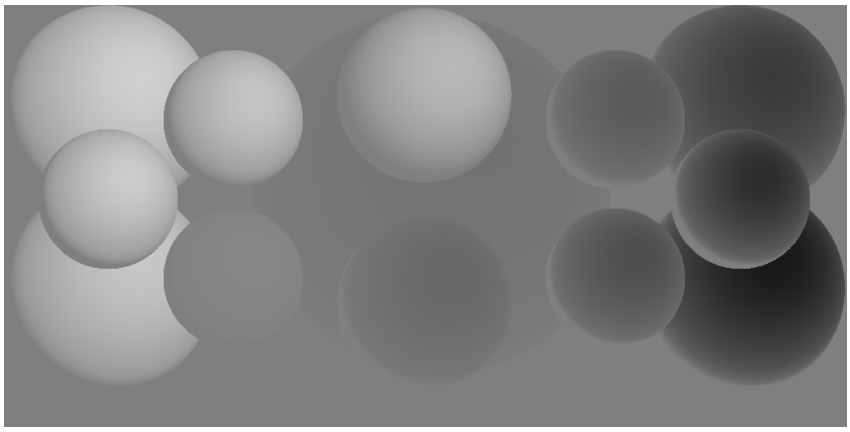

Fig. 5. ballons.pgm

Se concluye que el modo de run tiene un impacto casi indistinguible a menos de estar frente a imágenes suaves.

5.5 Comparación con JPEG-LS

Para comparar los resultados obtenidos se utilizan datos facilitados por los docentes del curso, tabulados en 6.

PGM files (grayscale)				
image	symbols in	bytes out	bits/sample	comp. ratio
baloons.pgm	328725	15375	0.374	0.046772
barbara.pgm	262144	159340	4.863	0.607834
bike.pgm	5242880	2749295	4.195	0.524386
faxballsL.pgm	524288	53232	0.812	0.101532
kodim07.pgm	393216	177279	3.607	0.450844
kodim20.pgm	393216	154263	3.138	0.392311
kodim24.pgm	393216	226306	4.604	0.575526
tools.pgm	1828800	1235563	5.405	0.675614
womanc.pgm	5242880	2767857	4.223	0.527927

Fig. 6. Tasas de compresión de JPEG-LS

Si se comparan las tasas de compresión obtenidas con las logradas por JPEG-LS, puede verse que estas últimas siempre son menores. Estudiando la diferencia

absoluta entre ambos resultados, se obtiene 0.015 para imagen de menor diferencia; y 0.069 para la mayor.

Fue esperado desde un principio que este algoritmo no supere al original, principalmente por la simpleza de esta versión y la omisión de características como la codificación del error $M'(e) = M(-1 - e)$, la reducción de errores de predicción módulo 256 al rango $-128 \leq e \leq 127$, y la corrección del sesgo. Aun así, fue sorprendente que las tasas de compresión para el s elegido no sean nunca mayores a 0.7.

Si se observan las imágenes con menor diferencia de resultados, puede verse que las mismas son: ballons (0.015), tools (0.028), kodim24 (0.037), y faxballsL (0.044).

Si se intenta diferenciar este tipo de imágenes ante el resto, puede llegarse a la conclusión de que estas son las figuras con mayor cantidad de zonas suaves (baloons y faxballsL) y las de menor (tools y kodim24). Es posible que esto se deba a que JPEG-LS logra aprovechar mejor la compresión en zonas donde no se activa el modo de run, aunque obteniendo resultados opuestos en imágenes con menor redundancia.

6 Conclusiones

A lo largo del informe se buscó analizar características y el rendimiento del programa realizado al igual que las diferencias entre este y el algoritmo JPEG-LS.

Inicialmente fueron analizados datos particulares del algoritmo, los cuales fueron de ayuda para la creación y optimización del programa. Por ejemplo, la cantidad de memoria que ocupan determinadas variables3, ayudaron a definir que estructuras utilizar y dentro del programa.

Una vez estudiado el formato PGM utilizado en el programa, el cual se caracteriza por tener un cabezal con datos de la imagen, seguido por pixeles representados en forma de bytes, se decide que el programa codificará al cabezal idénticamente en el compresor junto con otros datos. Esto quitará complejidad a la lógica del programa y facilitaría el testeo de compresión sin pérdida, debido a que tanto pixeles como cabezal serán idénticos en la imagen original y descomprimida.

Mas adelante, mediante scripts de bash y algunas rutinas en C, se obtienen diferentes datos sobre la funcionalidad del programa. Se verifica que la compresión efectivamente se logra sin pérdida con ayuda del comando *cmp* de Linux. Además, se concluye estudiando todas las imagenes de prueba que el parámetro s que minimiza la tasa de compresión del programa es 8, y que parámetros mayores a 6 obtienen resultados similares.

Al comparar resultados del programa con los obtenidos con JPEG-LS, se obtienen datos similares aunque nunca menores en cuando a tasa de compresión. La menor diferencia se da para imágenes con muchas zonas de suavidad e imágenes con mucha redundancia, lo cual se atribuye a la eficiencia de la codificación sin modo de run del algoritmo original en contraste con este, el cual resulta no ser tan buen predictor al tener poca información.

Finalmente, se concluye la efectividad del modo de run únicamente al presentarse imágenes suaves, y lo poco que se pierde por su utilización en otros casos.

Como conclusión adicional, se destaca la eficiencia tanto en tasa de compresión (0.06 en el mejor de los casos), como en la velocidad del algoritmo, el cual era capaz de comprimir y descomprimir imágenes casi instantáneamente.

References

[1] M. J. Weinberger, G. Seroussi and G. Sapiro, ”The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS,” in IEEE Transactions on Image Processing, vol. 9, no. 8, pp. 1309-1324, Aug. 2000, doi: 10.1109/83.855427.

[2] B. Carpentieri, M. J. Weinberger and G. Seroussi, ”Lossless compression of continuous-tone images,” in Proceedings of the IEEE, vol. 88, no. 11, pp. 1797-1809, Nov. 2000, doi: 10.1109/5.892715.