PONTIFICIA UNIVERSIDAD JAVERIANA CALI

PROYECTO FINAL

SEGUNDA ENTREGA

FUNDAMENTOS Y ESTRUCTURAS DE PROGRAMACIÓN
LABORATORIO DE PROGRAMACIÓN

PREPARADO POR:

MARIA PAULA CARRERO

DIEGO FELIPE GALARZA

ENRIQUE JOSÉ PARÍS

PROFESORES:

CARLOS ALBERTO RAMIREZ

JUAN CAMILO RADA

NOMBIEMBRE 28

2016-2

Introducción

Para el desarrollo de nuestro proyecto final tuvimos que realizar un sistema de compresión de imágenes con pérdida (lossy), implementando varios sistemas de compactación de información como lo son la Transformada del Coseno y el Algoritmo de Huffman y elaborando un lector de imágenes en formato PGM/ASCII, para poder interpretar la información.

El proceso de la Transformada Discreta del Coseno o DCT, por sus siglas en inglés, es usado principalmente, al igual que la Transformada de Fourier en métodos de compresión (con pérdida de información) de archivos de imágenes y audio, y transforma linealmente una secuencia finita de datos como una suma de funciones coseno, oscilando en varias frecuencias. En nuestro proyecto utilizamos la DCT-II debido a que es la transformada que se aplica principalmente en la compresión de imágenes de formato JPEG, ya que organiza la información en una matriz de NxN y es aplicada la formula a cada fila y columna.

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1.$$

$$\begin{aligned} X_{k_1,k_2} &= \sum_{n_1=0}^{N_1-1}\left(\sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]\right)\cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \\ &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right]\cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]. \end{aligned}$$

El algoritmo de Huffman, fue desarrollado por David Huffman en 1952, y permite la creación de árboles binarios siguiendo una secuencia lógica de pasos:

1. Crear un árbol por cada dato ingresado (lista de enteros, alfabeto, etc.), con un nodo sin ningún hijo asociado, y que contiene el símbolo al que está asociado y su frecuencia de aparición.
2. Se toman los dos árboles de menor frecuencia y se unen para crear un nuevo árbol, siendo la raíz un nodo que contiene la suma de frecuencias de sus hijos, el árbol derecho será entonces el árbol de mayor frecuencia entre ambos y el izquierdo el árbol de menor frecuencia entre ambos.
3. Se repite el paso 2 para generar un único árbol con la suma total de frecuencias de todos los datos.

El formato de imágenes PGM (Portable Grey Map) es la forma más primitiva de representar imágenes en escala de grises, a través de un arreglo de enteros y otros datos o especificaciones asociadas a la imagen. Cada imagen PGM está conformada por una serie de datos que en su orden de aparición son, un “número mágico” que identifica el tipo de archivo, se consiste de dos caracteres normalmente “P2” para imágenes en formato ASCII y “P5” para imágenes en formato binario; después le sigue una etiqueta o comentario, usualmente con el nombre del archivo o su creador; más adelante tenemos dos números que son respectivamente filas y columnas máximas en la imagen, luego un valor que simboliza el valor máximo que pueden tener cada dato de la imagen, se representa 0 para negro total y dicho valor para blanco total; por último está la imagen representada como una matriz de enteros.

```
P2
# feep.ascii.pgm
24 7
15
0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0
0 3  3  3  3  0  0  7  7 7  7  0  0 11 11 11 11 0  0 15 15 15 15  0
0 3  0  0  0  0  0  7  0 0  0  0  0 11  0  0  0 0  0 15  0  0 15  0
0 3  3  3  0  0  0  7  7 7  0  0  0 11 11 11  0 0  0 15 15 15 15  0
0 3  0  0  0  0  0  7  0 0  0  0  0 11  0  0  0 0  0 15  0  0  0  0
0 3  0  0  0  0  0  7  7 7  7  0  0 11 11 11 11 0  0 15  0  0  0  0
0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0  0 0  0  0  0  0  0  0
```

Para facilitar todo el proceso de compresión de imágenes generamos un proceso de truncado para determinar qué calidad de compresión quería el usuario; si el usuario pide máxima compresión, significa mínima calidad, y al pasar la imagen por los procesos del programa se ve afectada en cantidad, ya que la DCT redondea coeficientes a un valor especifico y el algoritmo de Huffman los toma como uno mismo para agruparlos en un árbol, lo que da como imagen resultado, una imagen de más baja calidad (visualmente mas pixelada).

Por otro lado si el usuario pide mínima compresión, significa mayor calidad, por lo que la calidad de la imagen no se ve afectada y no se reduce de manera tan drástica; la imagen se reduce muy poco en tamaño y durante el proceso de compresión en la DCT los coeficientes resultantes de cada pixel no cambian entrando iguales al proceso del algoritmo de Huffman.

Implementación de soluciones:

Lector de imágenes PGM

Principalmente para la lectura de imágenes plain PGM las cuales fueron seleccionadas por el profesor para el proyecto, se tiene que tener en cuenta que este archivo es muy similar a un archivo plano de texto, sabiendo esto es posible usar las librerías estándares de C para abrir y cerrar archivos. Para la lectura del archivo principal, lo que se hizo fue guardar todo en una estructura llamada struct info, en esta estructura se almacenan el número de filas, columnas, el máximo valor en la escala de grises que se presente en la imagen y finalmente se guarda la matriz la cual representa a la imagen. Para saltar los comentarios y espacios se creó una función auxiliar la cual hacia más fácil la lectura de cada línea, usando también la función fgets() que trabaja con strings de texto. Para leer la matriz se creó un ciclo en el cual por medio de la función fscanf() con la cual se guardaban todos los valores que representaban a un entero (ignorando así al valor del espacio en blanco), al tener todo guardado en la estructura se tiene la información principal que será usada para continuar con el código.

Transformada Discreta de Coseno

Tanto para la primera aplicación de la DCT, como la de su inversa, se utilizan tres funciones (tres funciones que tienen una versión DCT, y una inversa DCT).

- compressDCT / decompressIDCT: es la primera función que se llama desde el main. Se le ingresa una matriz, y retorna otra matriz en donde la DCT respectiva fue aplicada en bloques de 8x8 pixeles extraídos de la matriz original, y reingresadas en la posición de donde fueron extraídas. Consiste de dos Fors que empiezan en la posición 0,0 de la matriz (n1=0, n2=0), y avanzan en términos de 8. Estos ciclos envían matrices pequeñas de 8x8 que empiezan en la posición (n1,n2) a la siguiente función.
  Tiene una condición especial en la que, si la matriz que se le mandó tiene dimensiones menores a 8 en el ancho o alto, reajusta para partir la matriz en submatrices de no 8x8, sino de un tamaño que se acomode.
- matrixDCT / matrixIDCT: aplica la transformada de coseno respectiva a cada elemento de una matriz nxn. Lo hace iterando sobre la matriz A que la función 1 le envió, y aplicando la siguiente función, que es la fórmula como tal de la DCT/IDCT sobre cada posición A[i][j]. Toda la función retorna la matriz A en el que cada entero inicial fue reemplazado por su coeficiente DCT.
- DCT / IDCT: aplica la fórmula de la transformada de coseno a un número de una matriz nxn, para calcular su llamado “coeficiente de DCT”. O, en el caso de la IDCT, toma el coeficiente DCT de un número y retorna el número original. La fórmula consiste de una sumatoria de valores que reciben como parámetros la posición del coeficiente/número a calcular en la matriz de la

que origina, y todos los otros elementos de la matriz nxn de la cual fue extraída. Esta sumatoria se implementa en algoritmo de C mediante el uso de dos fors (son dos ∑) que va sumando los cálculos a una variable X, utilizando el coseno de la librería estándar math.h. Retorna el coeficiente/número original calculado de la fórmula.

El input – la primera matriz que le entra a compressDCT, es la matriz leída del archivo PGM, y retorna una matriz de doubles a la cual se le aplica el proceso de truncado en el siguiente paso.

En el caso de la IDCT (al pasar del texto codificado a la imagen reconstruida), recibe la matriz leída del texto codificado - que pasó por el proceso de truncado inverso (explicado a continuación) – y retorna la matriz original PGM reconstruida que se da a la escritura del nuevo PGM reconstruido.

Proceso de Truncado

El truncado propuesto, la función quantify(), consiste en dividir todos los números de la matriz resultante de la compressDCT por un número fijo: que es 10 en la compresión mínima, 35 en compresión media y 60 en compresión alta. Lo que esto logra es reducir los números de la matriz, logrando cada vez más cadenas de cero: un número que ocupa menos bytes en texto que cualquier doublé resultante del coeficiente. Existe también la función inversa, antiquantify(), que toma una matriz que fue truncada con quantify, y reconstruye los valores originales multiplicando todos los números por el valor fijo por el que se dividieron antes.

Mientras más alto sea el número usado para dividir en quantify(), más pérdida de datos hay, pero más homogéneos son los números y por ende, menos espacio ocupan.

Algoritmo de Huffman

Antes de empezar a explicar cómo se implementó en si el algoritmo de Huffman se debe tener clara la estructura escogida para la realización del código, para esta se usó una Cola, que fue una estructura vista en clase, la cual tiene un puntero a un nodo que a su vez apunta a un árbol, la implementación de la Cola se hizo más que todo para no tener que trabajar con apuntadores dobles cuando se quisieran cambiar los valores de un nodo en específico sin tenerlo que retornar y guardar más memoria de la necesitada. Cuando se lee la matriz en el paso anterior, lo primero

que se hace es ir contado cada vez que vaya a agregar un elemento a la cola, se hace un chequeo de cuantas veces ha aparecido, o se le crea un nodo nuevo para agregarlo al final de la cola. Al tener la cola con las frecuencias de cada uno de los valores, se llama a la función HuffmanCodes, esta función recibe como parámetro la cola con la que se ha trabajado, y al ser una lista de árboles, lo que se realiza es en esta función se va iterando hasta que la longitud de la cola sea mayor que 1, cada vez se llama a la función extractMin, esta función busca en la cola el mejor valor que se tenga en ese momento y lo saca de la lista, de nuevo llama a la función, entonces se tendrá que existen dos árboles con menor frecuencia, se crea un árbol nuevo al cual se le asignan como nodo derecho al nodo con mayor frecuencia entre los dos y al izquierdo al menor entre los dos, en adición se les asigna a cada uno el padre que será este árbol nuevo, de manera iterativa esto sucede hasta que se deje de cumplir la condición.

Al tener el árbol de Huffman lo que se hace es llamar a la función HuffmanCodes, esta crea una lista que va a tener un char* al número binario que se le será asignado a cada nodo, y un int que representa el valor del árbol, usando funciones auxiliares se busca cada uno de los valores que aparecieron principalmente antes de crear el árbol y al encontrarse se realiza una búsqueda de su padre para ir verificando cual es el valor que se le va a asignar, este valor es guardado en el char* mencionado anteriormente, así hasta que se encuentren todos los valores principales.

Cuando se tiene la lista nueva, se llama a la función lastMatrix que toma ese valor de binarios y lo invierte ya que el código fue adquirido de manera invertida, se llaman a diversas funciones

En el caso de que el usuario pida una compresión normal, la imagen pasa por el proceso de compresión de manera usual, y la DCT genera los coeficientes con 8 dígitos (estándar), se hace una aproximación a 2 dígitos para agruparlos y finalmente realizar el proceso de Huffman.

A parte de las operaciones principales que requería el proyecto, usamos algunas funciones adicionales que nos facilitaban la lectura de archivos, manejo de variables y manejo de estructuras de datos, logrando así crear un programa que no solo comprimiera una imagen dada en formato PGM sino que además diera la posibilidad de descomprimirla.

RESULTADOS:

Se logró comprimir y descomprimir la imagen, al realizar todo el proceso de compresión el cual incluye el paso de la matriz por la DCT, el truncado y finalmente por el algoritmo de Huffman, logramos crear un archivo .txt el cual presenta aproximadamente la mitad del peso del archivo pgm al cual se le aplico el proceso,

es decir que la compresión de la imagen con los nuevos valores fue exitosa, puesto que se necesitaba generar un archivo de menor tamaño y se logró.

Todo esto depende de la calidad con la cual la persona seleccionó comprimir el archivo, si quiso comprimirlo con máxima compresión, el archivo pesara menos que si se llegara a seleccionar alguna otra calidad.

Al tener el archivo comprimido, se realiza el proceso de descompresión que fue explicado anteriormente, con este proceso la imagen pasa por un cambio de los valores que tiene a los valores que tenía antes, después por la DCT Inversa y finalmente se escribe en un archivo PGM nuevo. El resultado presenta una imagen con una línea diagonal que se da por la DCT, una línea diagonal que se presenta por el proceso de la DCT en matrices 8x8.

Si se efectuó máxima compresión, habrá aproximadamente un 15% de perdida en el peso de la imagen, y al descomprimirse, esta se escribirá con mayor escala de grises que las que tenía antes, es decir que la calidad fue perdida exitosamente.

Si se efectuó compresión normal, habrá aproximadamente un 12% de perdida en el peso de la imagen, y al descomprimirse, esta se escribirá una imagen que tendrá gris, pero se verá mejor que si se hubiera comprimido en máxima compresión.

Finalmente, si se escogió mínima compresión, habrá aproximadamente un 10% de perdida en el peso de la imagen, y al descomprimirse, esta se escribirá una imagen que será un poco más gris que la normal, pero es más clara que todas las demás imágenes.

Presentamos dificultades para varios procesos del proyecto, en el sistema operativo de Windows, el lector de la imagen presentaba errores, no lograba leer de manera correcta la matriz así que no lograba realizar ninguno de los demás procesos.

En el sistema operativo de Linux logramos que la transformada discreta funcionara, el proceso de compresión y descompresión de la imagen es muy parecido al de un compresor real, puesto que dependiendo de la calidad de la compresión se pierden los valores de algunos pixeles, la calidad es menor a la original.

En el sistema operativo de IOS todo el proceso funciona, la unión de la DCT y del algoritmo de Huffman escriben en un archivo nuevo, la imagen parecida a la original pero más gris.

Otras dificultades las cuales eran interpretar la información recibida de manera correcta, revisar que las frecuencias retornadas por el código de Huffman sean correctas para imágenes de gran tamaño, entre otros. En conclusión, estas dificultades fueron corregidas y logramos finalizar nuestro proyecto de manera exitosa pudiendo comprimir y descomprimir varios tamaños de imágenes por igual.

Les agradecemos a nuestros profesores encargados del Proyecto por sus enseñanzas que nos permitieron entender, investigar, leer y desarrollar con mayor facilidad todo lo relacionado con este proyecto.