Documentación de la primera etapa del proyecto final de Comunicaciones digitales

Parte 1: Codificación de la fuente

Para la imagen

1) **Codificación**

   Se nos pidió realizar un código que transformara la información de color RGB a información binaria usando el algoritmo RLE.

   Para realizar esto, se investigó en primer medida cómo trabajar con imágenes en el lenguaje de programación java. Encontramos que la clase Color de java nos permite tomar la información de una imagen y traducirla a un tipo de variable que contiene cada uno de los 3 valores RGB de cada pixel en números enteros que van de 0 a 255. Luego, utilizamos la clase BufferedImage la cual permite mantener la representación de una imagen en memoria. Para trabajar con esta variable tipo Color, se utilizaron los métodos getRed() , getGreen() y getBlue()

Para empezar a hacer la compresión utilizamos la siguiente imagen como prueba:

- Método Compresión RLE.

Al principio se creó un método para realizarle una comprensión a la información de los pixeles en RGB que se encontraban en una matriz de String. Dicha matriz en cada posición contiene una cadena concatenada por tres números separadas por el carácter “-” . Al principio lo que se quería realizar es tener una sola cadena de String que estuviera comprimida la información de todos los números tanto los “Blue”, “Red” y “Green”. Lo que no tuvimos en cuenta fue que el tipo de dato String tiene un determinado tamaño de almacenamiento. Al tratar de concatenarle los números a la cadena en cada procedimiento de compresión nos arrojaba errores , en el cual el máximo tamaño de tipo de dato String ya se había excedido, por lo cual tuvimos que recurrir a otra solución.

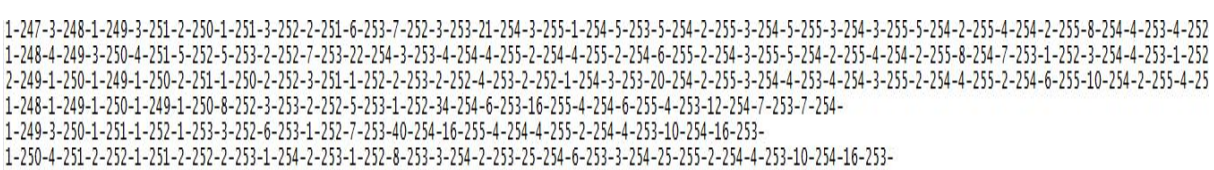
La segunda solución fue almacenar los datos de los pixeles de colores “Green” , “Red” y “Blue” en tres matrices diferentes, cada matriz iba a almacenar su número de píxel correspondiente a su color. De esta manera la comprensión a la información utilizando el algoritmo RLE era más fácil de manejar.

```
public ArrayList<String> CompresionRLE(int matriz[][]){
```

El algoritmo funciona de esta manera:
Primero se recibe la matriz del color correspondiente como parámetro, después , se realiza un ciclo anidado para almacenar todos los números de la matriz en una Queue. Después de tener almacenado todos los datos se realiza otra ciclo para realizar la compresión RLE. Esta información se almacena en cadenas de String cada que llega a un número total de números.Este número total de números corresponde a la cantidad de filas que tiene la matriz, cada que llega a este número tope de números , la cadena de String se almacena en una Lista de tipo String y se continúa con el ciclo anterior hasta que la cola quede vacía.

```java
public ArrayList<String> CompresionRLE(int matriz[][]){
    Queue<Integer> numeros = new LinkedList<Integer>();
    for (int i = 0; i < alto; i++) {
        for (int j = 0; j < ancho; j++) {
            numeros.add(matriz[i][j]);
        }
    }
    int contadorVecesAparece = 1;
    int numeroSiguiente = 0;
    int numeroActual=0;
    String cantApareceNum ="";
    int contadorFila = 0;
    boolean seAgrego=false;
    ArrayList<String> filas = new ArrayList<String>();
    while (!numeros.isEmpty()) {
        contadorFila++;
        if(contadorFila==120) {
            if(seAgrego) {
                numeroActual=numeros.remove();
                cantApareceNum+=contadorVecesAparece+"-"+numeroActual+"-";
                seAgrego=false;
            }
            else {
                cantApareceNum+=contadorVecesAparece+"-"+numeroActual+"-";
                numeros.remove();
            }
            filas.add(cantApareceNum);
            cantApareceNum="";
            contadorFila=0;
            contadorVecesAparece=1;
        }
        else {
            numeroActual=numeros.remove();

            if(!numeros.isEmpty()) {
                numeroSiguiente=numeros.peek();
                if(numeroActual==numeroSiguiente) {
                    contadorVecesAparece++;
                }
                else {
                    cantApareceNum+=contadorVecesAparece + "-" + numeroActual +"-";
                    contadorVecesAparece=1;
                    if(contadorFila==119) {
                        seAgrego=true;
                    }
                }
            }
        }
    }
    return filas;
}
```

Al finalizar nuestro método de compresión RLE , las cadena de String con la información comprimida almacenada en la Lista (ArrayList) se ve así:



Al principio de cada cadena se indica cuántos números iguales se encuentran unos seguidos de otro y se coloca este total separado de un carácter “-” y seguido de este el número que se repite, así sucesivamente hasta el final de cada cadena.

Vista general de las filas comprimidas:

- Pasar la información comprimida a binario.

Para esta etapa se utilizó el método binarizar Matriz RLE la cual se encargaba de binarizar cada cadena comprimida de la lista de String de la etapa anterior.

.

```
    public ArrayList<String> binarizarMatrizRLE(ArrayList<String> matrizRLE) {
        String cadenaBinarizada = "";
        String cadena = "";
        char guion = '-';
        String numerosActuales = "";
        ArrayList<String> filaBinarizada = new ArrayList<String>();
        for (int i = 0; i < matrizRLE.size(); i++) {
            cadena = matrizRLE.get(i);
            for (int j = 0; j < cadena.length(); j++) {
                char actual = cadena.charAt(j);
                if (cadena.length() != (j + 1)) {
                    char siguiente = cadena.charAt(j + 1);
                    if (actual != guion) {
                        numerosActuales += actual + "";
                        if (siguiente != guion) {
//                          numerosActuales+=actual+"";
                        } else {
                            cadenaBinarizada += decimalToBinary(numerosActuales);
                            numerosActuales = "";
                        }
                    }
                    else {
                        cadenaBinarizada += guionBinarizado();
                    }
                } else {
                    cadenaBinarizada += guionBinarizado();
                }
            }
            filaBinarizada.add(cadenaBinarizada);
            cadenaBinarizada = "";
        }
        return filaBinarizada;
    }
```

Este método utiliza dos métodos auxiliares para su funcionamiento; decimalToBinary y guionBinarizado. El método principal se encarga de convertir a binario cada número o carácter de la cadena de String que se encuentra en la lista. Si la variable a binarizar es un número utiliza el código ***decimalToBinary***:

```
public String decimalToBinary(String numeros) {
    String cadenaNumeroBin = "";
    String num = numeros;
    int numero = Integer.parseInt(num);
    String binario = "";
    if (numero > 0) {
        while (numero > 0) {
            if (numero % 2 == 0) {
                binario = "0" + binario;
            } else {
                binario = "1" + binario;
            }
            numero = (int) numero / 2;
        }
    } else if (numero == 0) {
        binario = "0";
    } else {
        binario = "No se pudo convertir el numero. Ingrese solo números positivos";
    }

    cadenaNumeroBin = concatenarCerosFaltantes(binario);
    return cadenaNumeroBin;
}
```

Este método se encarga de convertir un número que se encuentra en decimal en base dos o binario.
Para evitar confusiones en etapas posteriores , los números que estaban en binario deben tener 8 números en binario. Por los cual, en este método se utilizó otro método auxiliar llamado ***concatenarCeroFaltantes*** que lleva como parámetro el número convertido en binario:

```
public static String concatenarCerosFaltantes(String binario) {

    int base8 = 8;
    int cerosFaltantes = base8 - binario.length();
    String ceros = "";
    for (int i = 0; i < cerosFaltantes; i++) {
        ceros += 0;
    }
    return ceros + binario;
}
```

Este método agrega o concatena ceros (0) al principio de la cadena de String en el cual se encuentra el número en binario almacenado con el fin de que al final hayan 8 números en binario(0 y 1) para evitar confusiones y complicaciones futuras.

Cuando el carácter a binarizar es ***"-"*** se utiliza el método ***guiónBinarizado:***

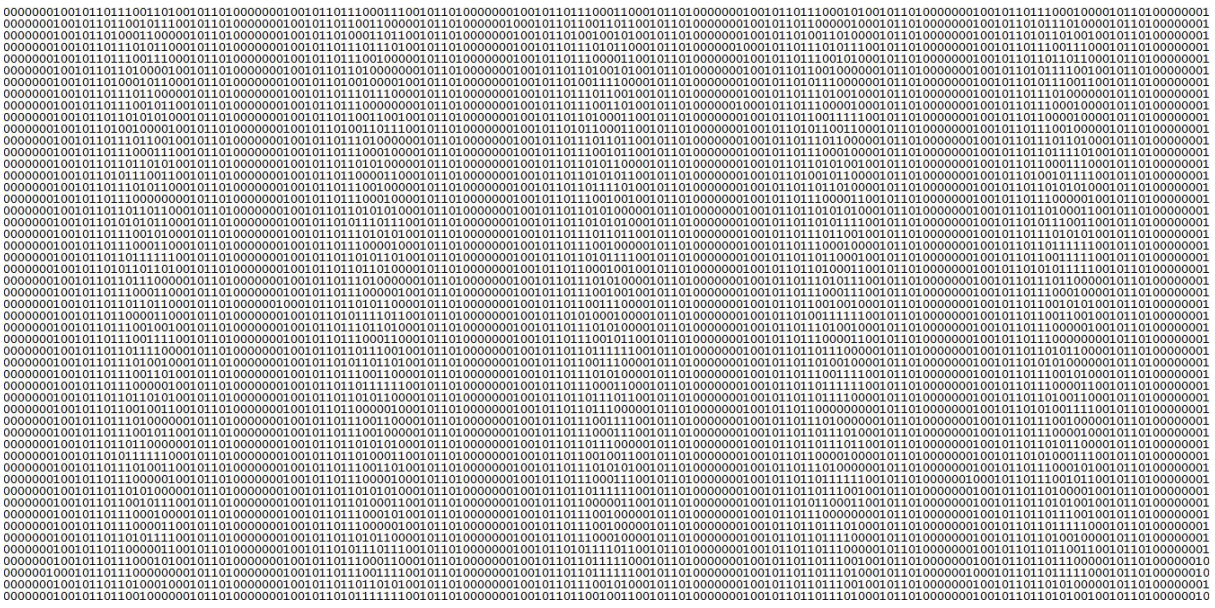
```
public String guionBinarizado() {
    String guion = "-";
    String guionBinarizado;
    String binario = "";
    int x = 0;
    x = guion.charAt(0);
    guionBinarizado = Integer.toBinaryString(x);

    binario = concatenarCerosFaltantes(guionBinarizado);

    return binario;
}
```

El método convierte el carácter ***"-"*** a binario utilizando el método de la clase Integer toBinaryString. Al igual que el método anterior, se utiliza el método auxiliar ***concatenarCeroFaltantes*** por si el length de el string binarizado es menor a 8.

El resultado de esta etapa se vé de la siguiente manera:

```
00000001001011011100110100101101000000010010110111000111001011010
00000001001011011001011100101101000000010010110110011000001011010
00000001001011010001100000101101000000010010110100011011001011010
00000001001011011101011000101101000000010010110111011101001011010
```

Vista general de todas las filas:



2) **Decodificación de la imagen**
   Se pide desarrollar un código que transforme la información binaria a información de color RGB para cada canal usando el algoritmo RLE

- Pasar la información que se encuentra a binario a decimal.

Para esta etapa de pasar los números a formato decimal se utilizó el método decodificarRLEBinarizado. El funcionamiento es muy sencillo, al recibir la lista de cadenas binarizadas, coje cada cadena y hace un recorrido de esta, almacena los caracteres hasta que sean 8 números en binario concatenados a una variable auxiliar, cuando están los 8 números se hace una verificación si la cadena con numero binarios corresponde a un número decimal o el carácter **"-"**. Para saber si es un carácter se compara con la cadena "**00101101"** y si no se utiliza el método auxiliar ***binarioToDecimal***.

```
public ArrayList<String> decodificarRLEBinarizado(ArrayList<String> filas) {
    int base2 = 8;
    int contador = 0;
    String cadenaDecodificada = "";
    String cadenaADecodificar = "";
    String valoresActuales = "";
    String valorAnterior = "";
    int contadorBina = 0;
    ArrayList<String> filasDecoficadas = new ArrayList<String>();
    for (int i = 0; i < filas.size(); i++) {
        cadenaADecodificar = filas.get(i);
        for (int j = 0; j < cadenaADecodificar.length(); j++) {
            contador++;
            if (contador < base2) {
                valoresActuales += cadenaADecodificar.charAt(j);
            }
            else if (contador == 8) {
                valoresActuales += cadenaADecodificar.charAt(j);
                if (esGuion(valoresActuales)) {
                    if (valorAnterior.equals("00101101")) {
                        contadorBina++;
                        if (contadorBina == 1) {
                            cadenaDecodificada += binarioADecimal(valoresActuales);
                            valorAnterior = valoresActuales;
                            valoresActuales = "";
                            contador = 0;
                        }
                        if (contadorBina == 2) {

                            cadenaDecodificada += "-";
                            valorAnterior = "00101101";
                            valoresActuales = "";
                            contador = 0;
                            contadorBina = 0;
                                contadorBina = 0;
                            }
                        }
                        else {
                            cadenaDecodificada += "-";
                            valorAnterior = "00101101";
                            valoresActuales = "";
                            contador = 0;
                        }
                    }
                    else {
                        cadenaDecodificada += binarioADecimal(valoresActuales);
                        valorAnterior = valoresActuales;
                        valoresActuales = "";
                        contador = 0;
                    }
                } else {
                    valoresActuales += cadenaADecodificar.charAt(j);
                }
            }
            filasDecoficadas.add(cadenaDecodificada);
            cadenaDecodificada = "";
        }
        return filasDecoficadas;
    }
```

Método ***binarioToDecimal:***

```
public int binarioADecimal(String numeroBinario) {
    int longitud = numeroBinario.length();// Numero de digitos que tiene nuestro binario
    int resultado = 0;// Aqui almacenaremos nuestra respuesta final
    int potencia = longitud - 1;
    for (int i = 0; i < longitud; i++) {// recorremos la cadena de numeros
        if (numeroBinario.charAt(i) == '1') {
            resultado += Math.pow(2, potencia);
        }
        potencia--;// drecremantamos la potencia
    }
    return resultado;
}
```

El método convierte la cadena que está concatenada con unos y ceros y lo convierte a decimal.

Inconvenientes en esta etapa :

Al finalizar esta etapa tuvimos algunas complicaciones en el resultado. Algunas cadenas demostraron anomalías en relación a los caracteres “-” seguido de los numeros asi:

```
'-1-20-1-29-1-23-1---1-49-1-29-1-34-1-32-1-28-1-48-1-38-1-27-1·
```

Algunas cadenas tienen el carácter **“-”** dos veces consecutivas. Había un factor que no habíamos tenido en cuenta, el número 45 en binario es exactamente igual que el carácter “-” en ASCII, por lo cual siempre iba a almacenar el carácter “-” más no el número como tal. Por lo anterior , tuvimos que modificar el algoritmo y así obtener los resultados esperados.

- Descomprimir la información

Para esta etapa final, consistió solamente en descomprimir la información a su estado original, obteniendo la totalidad de los números que corresponden a los píxeles de la imagen. Para ello se utilizó el algoritmo ***descompirmirRLE***:

```java
public Queue<Integer> descomprimirRLE(ArrayList<String> filasRLE) {
    Queue<Integer> numeros = new LinkedList<Integer>();

    String cadenaRLE = "";
    int contador = 0;
    for (int i = 0; i < filasRLE.size(); i++) {
        cadenaRLE = filasRLE.get(i);
        String arreglo[] = cadenaRLE.split("-");
        for (int k = 0; k < arreglo.length; k++) {
            int actualNumero = Integer.parseInt(arreglo[k]);
            if (k % 2 == 0) {
                contador += actualNumero;
                int numeroSiguiente = Integer.parseInt(arreglo[k + 1]);
                for (int l = 0; l < actualNumero; l++) {
                    numeros.add(numeroSiguiente);
                }
            }
        }
        contador = 0;
    }
    return numeros;
}
```

El método tiene un funcionamiento sencillo, cada número en la cadena está separada por el carácter “-”, entonces primero se utiliza el método split de la clase String para poder tener solo los números y almacenarlos en un arreglo. las posiciones en las que quedaron almacenados los números corresponden de esta manera, en las posiciones pares se encuentran los números que indican la cantidad de veces que aparece consecutivamente el número que se encuentra en la posición siguiente, de esta manera se utiliza un ciclo en el cual recorre la el número n que indicaba la cantidad de veces que aparece el número en la posición siguiente y agrega ese número siguiente en una Queue, esto se realiza hasta finalizar con todas las cadenas almacenadas en la Lista.

Al finalizar se obtiene una Queue con todos los números que corresponden a los pixeles de determinado color. Después de esta paso, se procede a construir nuevamente las matrices , cada una con su determinada Queue de números y llenar la matriz hasta que la Queue quede vacía.

Finalmente, se construye nuevamente y se imprime , obteniendo este resultado:

Imagen a codificar	Imagen decodificada
	

Como se puede observar, la imagen decodificada tiene algunos píxeles fuera de su lugar y es complemente diferente a la imagen inicial, por lo cual tuvimos que hacer un retroceso en las últimas etapas para averiguar cuál era el error.
Al revisar las matrices, nos dimos cuenta que habian posiciones en la matriz que estaban vacías, o sea estaban llenas de ceros, lo que nos indicaba que en las etapas anteriores habían números que se habían perdido. El error se encontraba en la parte inicial, en el método comprimirRLE , el cual estaba perdiendo información a la hora de comprimir. Ese error se soluciono y se pudo obtener la imagen original como resultado.

Aspectos importantes para el funcionamiento del programa

- Cuando ejecute el programa, encontrará 3 archivos en una carpeta llamada: “salida”. Podrá ver la imágen después de ser binarizada y decodificada, también un archivo .txt llamado: “decodificarRLEBinarizado” en donde se puede evidenciar cómo el programa utiliza los datos de la imágen por medio de variables de tipo String; y tercero encontrará el archivo: “EncoderBinario.txt” en el cual puede observar la binarización de la imagen.