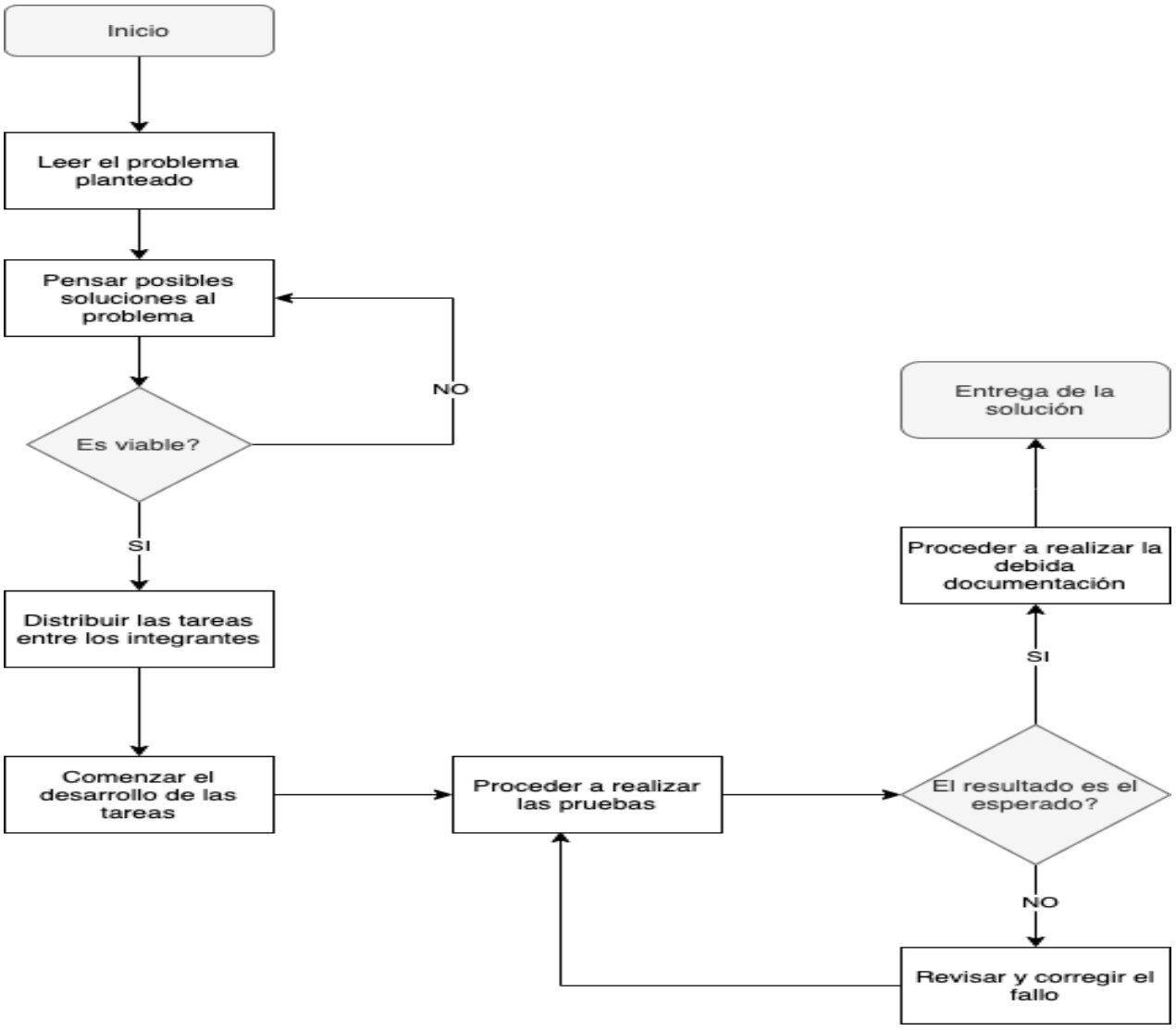
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INFORMACIÓN TECNOLOGÍA Y COMUNICACIONES

Nicolás Javier Salazar Echeverry A00348466
Duvan Alexis García Tovar A00346605
Mateo Gallego Ramírez A00347937
Brayan S. Garcés Portillo A00345862

Proyecto Comunicaciones Digitales

Resumen de Actividades:



Descripción del proyecto:

Se ha pedido a nuestro equipo de trabajo desarrollar un prototipo que permita la transmisión de datos relacionados con el ingreso de los asistentes al evento "BirdFair 2021", evento desarrollado por una empresa internacional de avistamiento de pájaros a tomar lugar en Cali, cerca al Cristo Rey.
Se espera que el prototipo esté en la capacidad de recibir datos por medio de una boleta digital. Los datos que se esperan recibir son:
-El número único de boleta
-Nombres y Apellidos del asistente
-Documento de identidad
-Edad y género
-Fotografía reciente(de menos de 6 meses de 180x180 píxeles)
Aunque se requiere que los datos se transmitan utilizando ondas sonoras desde un smartphone(Tx) donde corra la app hasta un receptor para así permitir el registro, para este prototipo se espera solo comprobar el funcionamiento de la transmisión inalámbrica, por lo tanto se espera probarlo solo usando un micrófono y parlantes para verificar la viabilidad del mismo.

Los requerimientos funcionales que se esperan son:
-Que esté en la capacidad de manejar archivos en el PC
-Que sea capaz de codificar y decodificar archivos de texto plano
-Que sea capaz de codificar y decodificar archivos en formato de imágenes
-Que sea capaz de codificar y decodificar archivos binarios

Desarrollo del proyecto:
El desarrollo del proyecto fue dividido en 4 partes, cada una de las partes representaba uno de los requerimientos funcionales que se pedían. Por tanto aquí se explicará cuál fue el acercamiento que se le dio a cada parte y como se desarrollo la solución, para al final unir cada una de estas en una sola solución

Requerimiento 1: Manejo de archivos en el PC

Tal como su nombre lo indica, se espera que el programa esté en la capacidad de leer archivos que se encuentren en el PC, para que así posteriormente sea capaz de procesarlos y realizar la tarea para la que se requiera esta información.
Por lo tanto para cumplir con este requerimiento, se planteó la solución de la siguiente manera:

- Para la lectura de archivos de texto plano en formato ASCII de 8 bits se empleó el método fileReader que acepta como parámetro la ruta del archivo que se desea leer y

posteriormente a la lectura se guardó el contenido del archivo en una variable de tipo String para su posterior tratamiento.

- En la obtención de la información binaria de un archivo de texto plano, fue necesario extraer primero cada uno de los bytes del archivo para su posterior conversión en cadenas de 8 bits.

Requerimiento 2: Codificación y decodificación de archivos de Texto

1. **Codificación:** Las clases MainLZW y CodificacionLZW son las encargadas de llevar a cabo todos los procesos necesarios para la codificación de fuente.

   Dentro la clase MainLZW se encuentra el método codificar dentro del cual se toma el mensaje a ser codificado a partir de un archivo de texto plano, en formato ASCII, para posteriormente pasar por parámetro a la clase CodificacionLZW.

```
public void codificar() {
    String cadena = lecEsc.leerTexto(ARCHIVO_ASCII);
    String cadena1 =lecEsc.leerBytes(ARCHIVO_ASCII);
    codificador.setMensaje(cadena);//le pasa el mens
```

   posteriormente se hace el llamado al método compresión (de la clase CodificacionLZW). encargado de llevar a cabo el algoritmo LZW para codificar el mensaje. inicializando el diccionario de datos y posteriormente construir la cadena de datos ya codificada a partir del mensaje inicial.dentro de este metodo tambien se calcula la cantidad de bits por símbolo que se requieren para representar la cadena ya codificada de manera binaria.

```
145    public void compresion() {
146        inicializarDiccionario();// este metodo se encarga de la inicializacion del diccionario.
147        String w = "";
148        String k = "";
149        String wk = "";
150        String cadenaSalida = "";
151        for(int i = 0; i< mensaje.toCharArray().length;i++) {
152            k =""+ mensaje.charAt(i);
153            wk = w+k;
154            if(buscarEnLista(wk) != null) {
155                w = wk;
156            }else {
157                cadenaSalida = cadenaSalida +" "+ buscarEnLista(w).getIndice();
158                diccionario.add(new NodoLZW(wk, contadorDiccionario));// se van adicionando los nuevos simbolos a
159                contadorDiccionario++;
160                w = k;
161
162            }
163        }
164        cadenaSalida = cadenaSalida + " " +buscarEnLista(w).getIndice();
165        numeroBits = (int) Math.ceil((Math.log(diccionario.get(diccionario.size()-1).getIndice()) / Math.log(2)));
166        cadenaCodificada = cadenaSalida;// se agrega el mensaje a transmitir completamente codificado a su variabl
167    }
```

   A continuación se emplea el método getCadenaBinariaCompleta dentro de la cual se toman los los valores de la cantidad de caracteres del diccionario, el número de bits por símbolo del mensaje codificado, el diccionario de codificación y el mensaje codificado, los traduce a binario para después los concatenarlos y retornarlos.

```
269    public void getCadenaBinariaCompleta() {
270        String salida= "";
271
272        salida = agregarCeros(valorBinario(caracteresDiccionario)+"", 8) + agregarCeros(valorBinario(numeroBits)+"", 8) + diccionarioBinario() + salidaBinario();
273
274        cadenaBinaria = salida;
275    }
```

cadenaBinaria = Caracteres diccionario + Numero de bits por símbolo + Diccionario + Mensaje Codificado

por último en el método codificar se crean dos archivos de texto plano el primero, codificacion.txt que contiene la cadena binaria generada anteriormente, el segundo archivo resumenCodificacion.txt contiene un archivo con un resumen de todo el proceso de codificación, donde se puede encontrar el mensaje original, su traducción a binario siguiendo el formato de ascii de 8 bits y cuantos bits tiene dicha cadena. posteriormente se muestra el mensaje ya codificado, cuantos bits por símbolo requiere, la cadena final traducida a binario, cuántos bits tiene esta cadena y la relación de compresión con respecto al formato ascii de 8 bits.

2. **Decodificación:** Las clases MainLZW y DecodificacionLZW son las encargadas de llevar a cabo todos los procesos necesarios para la decodificación de fuente.

dentro de la clase MainLZW se encuentra el método decodificar. dentro del cual primero se procede a leer de un archivo de texto plano, en este caso codificacion.txt, la información que se desea codificar y luego proceder a pasar esta información por parámetro a la clase DecodificacionLZW.

```
99    public void decodificar() {
100       String cadena = lecEsc.leerTexto(ARCHIVO_CODIFICACION);
101       decodificador.setCadenaBinaria(cadena);
```

posteriormente se hace el llamado al método inicializar encargado de tomar toda la cadena de bits empleada para la decodificación y es traducida para así poder inicializar el proceso de descompresión, primero tomando el el número de caracteres del diccionario.(que servirá para saber cuando termina el diccionario dentro de la cadena a decodificar)

```
134    public void inicializar () {
135        String bits = cadenaBinaria;
136        int contadorPosicion = 0; // contador que permite el desplasam
137        String cadenacruda = ""; // String que representara una cadena
138
139        while(contadorPosicion <= 7) {
140            cadenacruda = cadenacruda + bits.charAt(contadorPosicion);
141            contadorPosicion++;
142        }
143        caracteresDiccionario = getNumero(cadenacruda);// se agrega el
144
```

después el número de bits por símbolo, que será necesario para extraer y traducir a decimal cada símbolo del mensaje codificado y es guardado en una variable local..

```
146         cadenacruda = "";// se vacia la cadena para su posterior uso
147
148         while(contadorPosicion <= 15) {
149             cadenacruda = cadenacruda + bits.charAt(contadorPosicion);
150             contadorPosicion++;
151         }
152
153         numeroBits = getNumero(cadenacruda);// se agrega el numero de
154         cadenacruda = "";
```

continuando con la inicialización se lee y traduce toda la información del diccionario de decodificación y es guardado en una variable local.

```
157          int contadordic = caracteresDiccionario;// lleva la cuenta regresi
158          while(contadordic > 0) {//dentro de este ciclo se traduciran todos
159              int contador1 = contadorPosicion+8;// contador para sustraer e
160              while(contadorPosicion < contador1) {// dentro de este ciclo s
161                  cadenacruda = cadenacruda + bits.charAt(contadorPosicion);
162                  contadorPosicion++;
163              }
164              byte[] letra = new byte[1];
165              letra[0]= (byte) getNumero(cadenacruda);
166              String cadenaLetra = new String(letra);// se guarda el simbolo
167              cadenacruda = "";
168              cadenacruda = "";
169              contador1 = contadorPosicion+numeroBits;// se inicializa de nu
170
171              while(contadorPosicion < contador1) {//dentro de este ciclo se
172                  cadenacruda = cadenacruda + bits.charAt(contadorPosicion);
173                  contadorPosicion++;
174              }
175              int cadenaIndice = getNumero(cadenacruda);// se guarda el indi
176              cadenacruda = "";
177
178              diccionario.add(new NodoLZW(cadenaLetra, cadenaIndice));// se
179
180              contadordic--;
181          }
```

por último se extrae y traduce toda la información del mensaje a codificar y es guardado en una variable local.

```
183         contadorPosicion++;
184         while(contadorPosicion < bits.length()) {
185             int contador2 = contadorPosicion+numeroBits;
186             while(contadorPosicion < contador2) {
187                 cadenacruda = cadenacruda + bits.charAt(contadorPosicion-1);
188                 contadorPosicion++;
189             }
190             cadenaCodificada =cadenaCodificada +" "+getNumero(cadenacruda);
191             cadenacruda= "";
192         }
193
194     }
```

el siguiente paso es hacer el llamado al método descomprecion. encargado de elaborar la descompresión del mensaje usando toda la información antes traducida de la cadena

original.

```
214    public void descomprecion() {
215        String salida= "";
216        String caracter ="";
217        int codigoViejo;
218        int codigoNuevo;
219        String cadena= "";
220        String[] codigo = cadenaCodificada.split(" ");
221        codigoViejo = (int)Integer.parseInt(codigo[1]);
222        caracter = buscarEnDiccionario(codigoViejo);
223        salida = salida + caracter;
224        for( int i = 2; i <codigo.length;i++ ) {
225            codigoNuevo = (int)Integer.parseInt(codigo[i]);
226            if(buscarEnDiccionario(codigoNuevo) == null) {
227                cadena = buscarEnDiccionario(codigoViejo);
228                cadena = cadena+caracter;
229            }else {
230                cadena = buscarEnDiccionario(codigoNuevo);
231            }
232
233            salida =salida+ cadena;
234            caracter = cadena.charAt(0)+"";
235            diccionario.add(new NodoLZW(buscarEnDiccionario(codigoViejo)+caracter, caracteresDiccionario+i-2));
236            codigoViejo = codigoNuevo;
237        }
238
239        mensaje = salida;
240    }
```

por último se crea un archivo de texto llamado resumenDeCodificacion.txt que contiene el mensaje ya codificado.

Nota: los archivos anteriormente mencionados en codificacion y decodificacion se encuentran dentro de la carpeta resultados primera entrega que se encuentra dentro de la carpeta data del proyecto de java.
si desea modificar el mensaje a codificar modifique ÚNICA Y EXCLUSIVAMENTE el archivo de texto plano llamado textoModificable.txt de la carpeta resultados primera entrega.

Requerimiento 3: Codificación y decodificación de archivos en formato de Imágenes

1. **Codificación:** Para el procesamiento de las imágenes se utilizó el algoritmo RLE, para ello era necesario primero obtener la información de la imagen basándose en [1] se hizo la primera parte de obtener la imagen, luego de esto se proceso con el algoritmo de RLE y se guardó en una cadena de caracteres que luego se transformó a un vector de binarios
2. **Decodificación:** La decodificación de imagen se realiza en la clase ImageProcessor, que es la clase encargada de controlar todo lo relacionado acerca de la imagen. .Se utiliza el algoritmo RLE para decodificar. La decodificación RLE consiste en examinar el mensaje formado por pares (carácter, número de repeticiones) y escribir el texto equivalente escribiendo el carácter el número correspondiente de veces. Vemos que ingresa una cadena de String st con la información la cual se va a decodificar, dicha cadena se convierte a una variable tipo char, con el fin de analizar valor por valor, y de mientras se crea otra variable sb que es el lugar donde va a parar la información decodificada.

```java
public static String decode(final String st) {
    final StringBuilder sb = new StringBuilder();

    final char[] chars = st.toCharArray();

    int i = 0;
    while (i < chars.length) {
        int repeat = 0;
        while ((i < chars.length) && Character.isDigit(chars[i])) {
            repeat = repeat * 10 + chars[i++] - '0';
        }
        final StringBuilder s = new StringBuilder();
        while ((i < chars.length) && !Character.isDigit(chars[i])) {
            s.append(chars[i++]);
        }

        if (repeat > 0) {
            for (int j = 0; j < repeat; j++) {
                sb.append(s.toString());
            }
        } else {
            sb.append(s.toString());
        }
    }

    return sb.toString();
}
```

Requerimiento 3:Codificación y decodificación de archivos binarios

1. **Codificación:** El proceso de codificación de canal es llevado a cabo dentro de la clase CodificacionCanal.

   el primer paso a seguir es leer la matriz de paridad de un archivo de texto plano y proceder a asignar sus valores a una matriz dentro del programa. esto con el método inicializarMatriz.

```java
public void inicializarMatriz() {
    String[] datosMatriz = lecEsc.leerTexto(MATRIZ_DE_PARIDAD).split("\n");// se toman los

    String[] temporal = datosMatriz[0].split(" ");
    matrizParidad = new int[Integer.parseInt(temporal[0])][Integer.parseInt(temporal[1])];/

    for(int i = 1;i<datosMatriz.length;i++) {                    //-----------------------
        temporal = datosMatriz[i].split(" ");                    //dentro de esta seccion
        for(int j = 0;j<temporal.length;j++) {                   //acabo la asignacion de
            matrizParidad[i-1][j]= Integer.parseInt(temporal[j]); //datos de la matriz en s
        }                                                        //posciciones------------
    }                                                            //-----------------------

}
```

   posteriormente se emplea el método estaSistematizada que retorna tru si la matriz está sistematizada o false si no lo esta, a manera de verificación.

```java
public boolean estaSistematizada() {
    boolean condicion = true;
    int veriFila= 0; // contador que se mueve a travez de la posicion fila de cada pivote de la sub matriz de id
    int veriColum = matrizParidad[0].length-matrizParidad.length;// contador que se mueve a travez de la posicio
    for(int i = 0;condicion && i<matrizParidad.length;i++) {// se inicia desde la fila 0 hasta la fila n
        for (int j = matrizParidad[0].length-matrizParidad.length;condicion && j<matrizParidad[0].length;j++) {/
            if(i == veriFila && j == veriColum && matrizParidad[i][j] == 1) {// verifica que los pivotes sean un
                veriFila++; // incremente los contadores de fila y de columna--+
                veriColum++;// para avanzar al siguiente pivote----------------+
            }else if(i != veriFila && j != veriColum &&matrizParidad[i][j] != 0){// verifica si las demas posici
                condicion = false;// de no serlo la matriz no es sistematica
            }else if(i == veriFila && j == veriColum && matrizParidad[i][j] != 1) {
                condicion = false;//en cualquier otro caso la matriz no es sistematica
            }
        }
    }

    return condicion;
}
```

el siguiente paso se hace uso del método sistematizar, que en caso de que la matriz de paridad no esté sistematizada se encarga de elaborar operaciones de filas para volverla sistemática mediante el algoritmo de gauss jordan con algunas modificaciones para este caso en específico.

por último procede a hacer la codificación de canal haciendo uso de la matriz de paridad en el método codificar. Después crea un archivo de texto plano, resumen_codificacionCanal.txt, con un resumen de la parte de codificación de canal que contiene La matriz de paridad ya sistematizada, el mensaje sin codificar y su longitud en bit, también el mensaje ya codificado y su longitud en bits.

Nota: los archivos anteriormente mencionados en codificación de canal se encuentran dentro de la carpeta resultados segunda entrega que se encuentra dentro de la carpeta data del proyecto de java.
si desea cambiar los valores de la matriz de paridad modifique ÚNICA Y EXCLUSIVAMENTE el archivo llamado MatrizDeParidad.txt que está dentro de la carpeta resultados segunda entrega, por favor procure respetar el formato ahí presente.

2. Decodificación:

La clase *DecodificadorCanal* se encarga de hacer la decodificación de canal, tiene como atributo una matriz de paridad que le es asignada con por el codificador de canal por el método *setMatrizParidad.*

```java
public class DecodificadorCanal {

    private int[][] matrizParidad;

    public DecodificadorCanal() {

    }

    public void setMatrizParidad(int[][] matrizParidad) {
        this.matrizParidad = matrizParidad;
    }
```

Figura #. DecodificadorCanal

Dentro de esta clase se encuentran los siguientes métodos:

1. *calcularSindrome,* este método calcula el síndrome de una una palabra que le llega por parámetro, la cual es representada por un arreglo de enteros, retorna el síndrome de dicha palabra

```java
public int[] calcularSindrome(int[] r) {
    int rows =  matrizParidad.length;
    int columns = matrizParidad[0].length;
    int[] sindrome = new int[rows];

    for (int i = 0; i < rows; i++) {
        int sum = 0;
        for (int j = 0; j < columns; j++) {
            sum &= matrizParidad[i][j]*r[j];
        }
        sindrome[i] = sum;
    }
    return sindrome;
}
```

Figura #. calcularSindrome

2. *liderDeCoclase,* este método se encarga de calcular el líder de coclase dado el síndrome, el cual es representado con un arreglo de enteros, retorna un entero *i* que representa el valor de la columna *i-ésima* donde se presentó el error

```java
public int liderDeCoclase(int[] sindrome) {
    int rows =  matrizParidad.length;
    for (int i = 0; i < rows; i++) {
        if (comparteVectors(matrizParidad[i], sindrome) ) {
            return i;
        }
    }
    return 0;
}
```

Figura #. liderDeCoclase

3. comparteVectors, compara dos vectores que le llegan por parámetro si son diferentes retorna *false,* retorna *true* de lo contrario

```
public boolean comparteVectors(int[] vecM, int[] vec ) {
    for (int i = 0; i < vec.length; i++) {
        if( (vecM[i] ^ vec[i]) !=  0) {
            return false;
        }
    }
    return true;
}
```

Figura #. comparteVectors

4. *deteccionError,* este método detecta si existe un error a partir del sindorme con el método *binaryAdd,* Figura # este método toma el síndrome como parámetro y suma cada una de sus posiciones si retorna un valor diferente de cero el síndrome es diferente del vector nulo y por ende se presenta un error en en mensaje.

```
public boolean deteccionError(int[] sindrome) {
    if(binaryAdd(sindrome) == 0)
        return false;
    else
        return true;
}
```

Figura #. deteccionError

```
private int binaryAdd(int[] r) {
    int s = 0;
    for (int i = 0; i < r.length-1; i++) {
        s &= r[i];
    }
    return s;
}
```

Figura #. binaryAdd

Referencias

[1] Efford, N. (2000). Digital image processing: a practical introduction using Java. 976. https://doi.org/10.1049/ep.1978.0474