Mayo 4, 2016

Universidad católica de colombia

Redes computacionales

Codificación hdb3 y b8zs

manual técnico

Contenido

[Funcionamiento General 2](#_Toc450134065)

[Descripción de los métodos principales 4](#_Toc450134066)

[ami 4](#_Toc450134067)

[Parámetros de entrada 4](#_Toc450134068)

[Detalle del algoritmo 4](#_Toc450134069)

[b8zs 6](#_Toc450134070)

[Parámetros de entrada 6](#_Toc450134071)

[Detalle del algoritmo 6](#_Toc450134072)

[hdb3 8](#_Toc450134073)

[Parámetros de entrada 8](#_Toc450134074)

[Detalle del algoritmo 8](#_Toc450134075)

[Descripción de los métodos auxiliares 10](#_Toc450134076)

[DetectarMetodoResolucion 10](#_Toc450134077)

[Parámetros de entrada 10](#_Toc450134078)

[Detalle del algoritmo 10](#_Toc450134079)

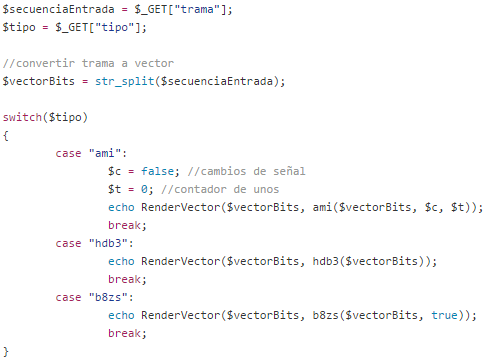
[DividirVector 11](#_Toc450134080)

[Parámetros de entrada 11](#_Toc450134081)

[Detalle del algoritmo 11](#_Toc450134082)

# Funcionamiento General

La página *codificacion.php* recibe vía get dos parámetros: *trama* y *tipo*, los cuales permiten identificar el tipo de procesamiento que se va a aplicar en la secuencia.



Para el caso de codificación *ami*, se pasan dos variables por referencia que corresponden a:

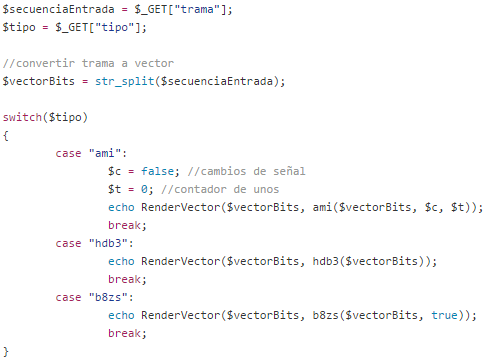
* $c : *boolean*

Indica si empieza con la señal invertida. Esto sirve para los casos donde se deba procesar con lógica negativa.

* $t : *int*

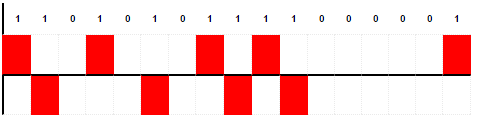
Lleva el conteo de unos (1) tanto positivos como negativos. Se usa en posterior codificación *hdb3*.

En *b8zs*, el segundo parámetro corresponde al tipo de lógica con la cual se inicia el algoritmo (positiva o negativa). El algoritmo por defecto funciona con lógica positiva.



El método *RenderVector* sirve para convertir una secuencia binaria a una tabla HTML. Recorre la secuencia bit por bit y arma una tabla con 3 filas. La primera fila corresponde al bit que se procesó, la segunda incluye un marcador para aquellos bits en uno (1) y la tercera marcador para los bits en menos uno (-1). Lo que queda entre las filas 2 y 3 es gestionado como un cero (0).

En la siguiente imagen se ve la salida para la secuencia ***11010101111000001***:



# Descripción de los métodos principales

Esta sección explica la estructura y como funcionan los algoritmos implementados para codificar una secuencia binaria.

## ami

### Parámetros de entrada

* $vector: *array*

Vector de valores binarios a codificar.

* &$signalChange: *boolean*

Indica si hubo cambio de señal en el último bit en uno (1) transmitido.

* &$totalUnosTransmitidos: *int*

Lleva el conteo de los unos (1) tanto positivos como negativos que se han transmitido durante la codificación.

### Detalle del algoritmo

Una vez suministrados los parámetros de entrada, se inicializan las variables de retorno del método y se procede a iterar el vector bit por bit.



En la iteración, si el bit actual es uno (1), se procede a realizar el cambio de señal, aumentar el total de unos transmitidos y llenar la variable de retorno con el uno y el signo que corresponda.



Cuando el bit es cero (0), no hay que realizar ningún proceso adicional, solo incluir en la variable de retorno ese cero (0).



Una vez procesado el bit, adicionarlo al vector de retorno y devolver el resultado del método.



## b8zs

### Parámetros de entrada

* $vector: array

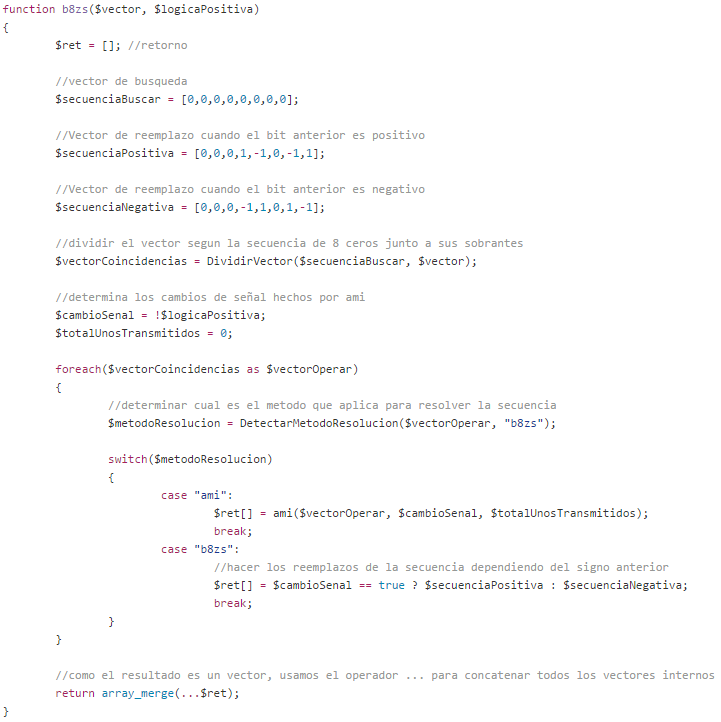
Vector de valores binarios a codificar.

* $logicaPositiva: boolean

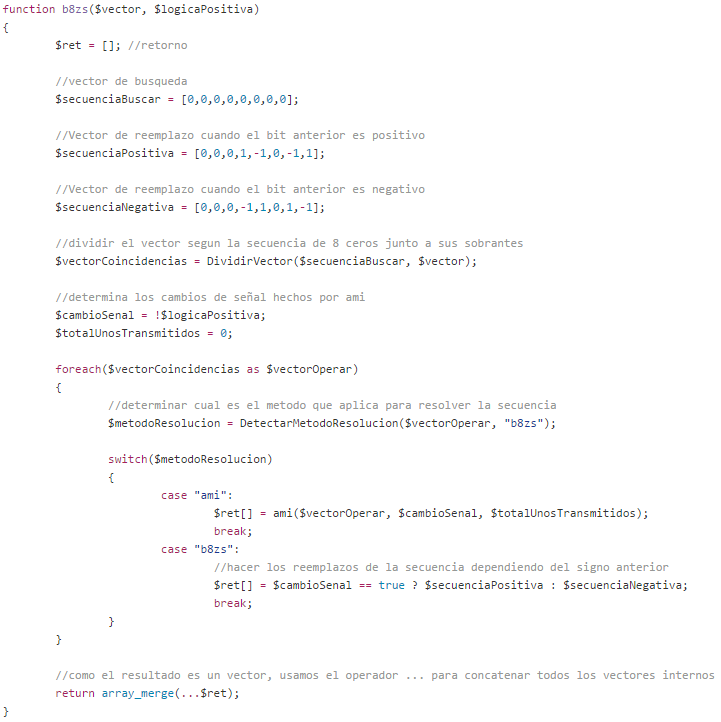
Indica si el algoritmo inicia con lógica positiva o negativa.

### Detalle del algoritmo

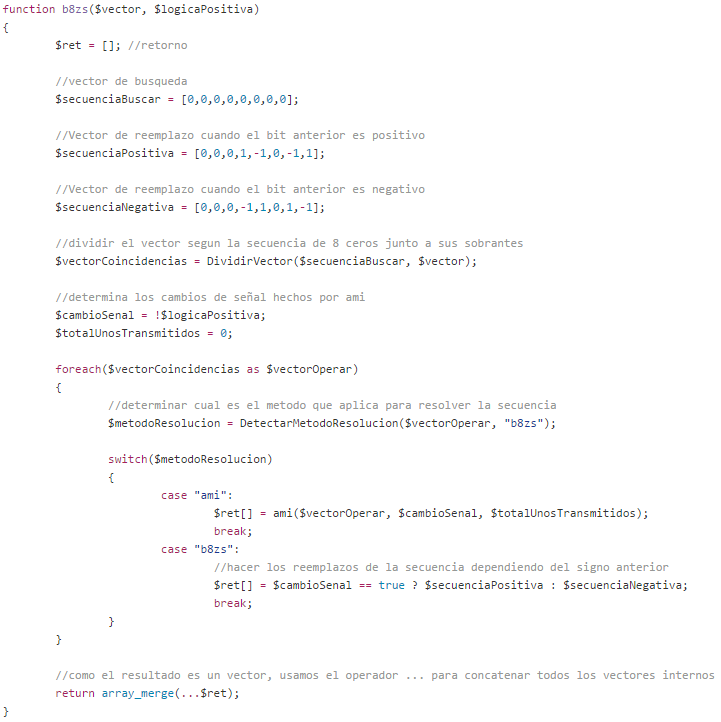
Una vez suministrados los parámetros de entrada, se inicializan las variables de retorno y las secuencias de reemplazo, es decir, la secuencia de ocho (8) ceros continuos y las secuencias que dependiendo del signo del bit del ultimo cambio de señal serán reemplazadas.



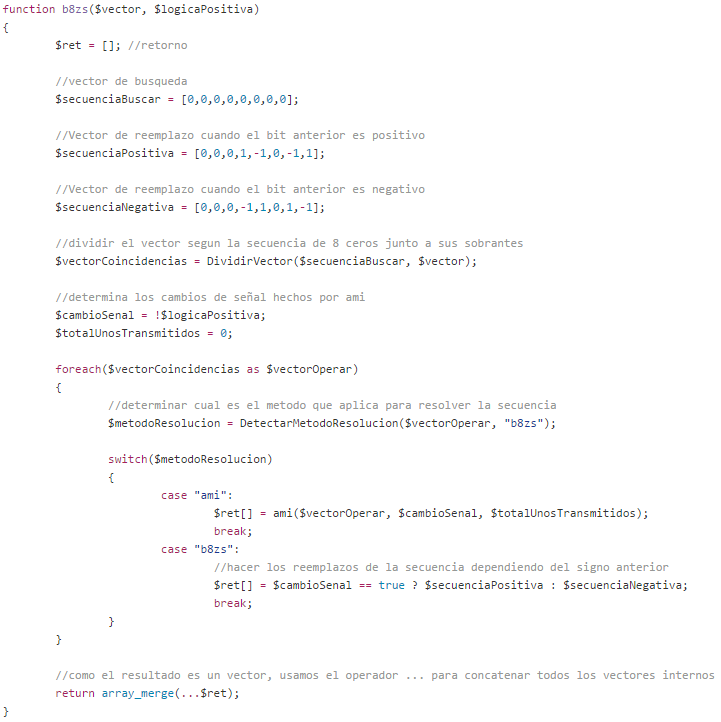
Se procede a dividir el vector en partes, de ese modo se puede operar por *b8zs* o *ami* el segmento que se encuentra en la iteración.



Realizando la iteración del vector segmentado, se procede a determinar si se debe codificar en *ami* o *b8zs*. Si es por *ami*, se invoca el método previamente definido y se realiza su respectivo proceso, pero si es *b8zs*, se pregunta por el cambio de señal y al vector de resultado se le coloca una de las secuencias de reemplazo previamente definidas.



Al haber recorrido todo el vector segmentado, se une en uno solo el vector de resultado, de ese modo queda listo para retornar el método.



## hdb3

### Parámetros de entrada

* $vector: *array*

Vector de valores binarios a codificar.

### Detalle del algoritmo

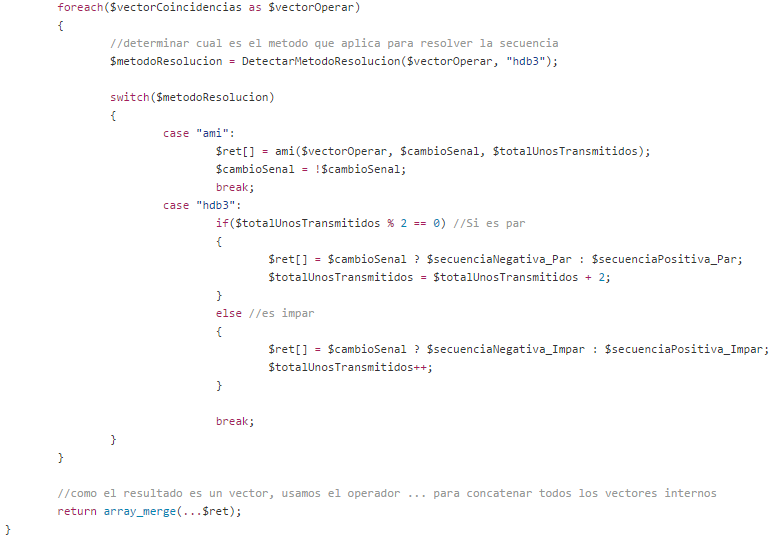
Una vez suministrados los parámetros de entrada, se inicializan las variables de retorno y las secuencias de reemplazo, es decir, la secuencia de cuatro (4) ceros continuos y las secuencias que dependiendo de la paridad y signo serán reemplazadas.



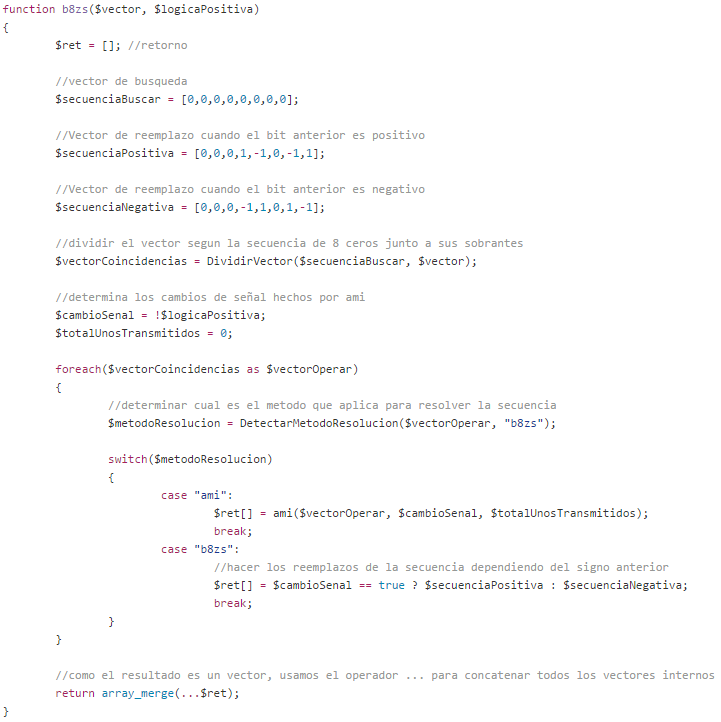
Se procede a dividir el vector en partes, de ese modo se puede operar por *hdb3* o *ami* el segmento que se encuentra en la iteración.



Realizando la iteración del vector segmentado, se procede a determinar si se debe codificar en *ami* o *hdb3*. Si es por *ami*, se invoca el método previamente definido y se realiza su respectivo proceso, pero si es *hdb3*, se pregunta por la paridad de los unos transmitidos y por el cambio de señal, de ese modo determinamos si la secuencia de reemplazo es par o impar, positiva o negativa.



Al haber recorrido todo el vector segmentado, se une en uno solo el vector de resultado, de ese modo queda listo para retornar el método.



# Descripción de los métodos auxiliares

Los siguiente métodos generalizan algunas operaciones en común que se deben realizar cuando se codifica vía *b8zs* o *hdb3*

## DetectarMetodoResolucion

Analiza un vector y determina según su longitud y contenido si se debe codificar por un método diferente a *ami*

### Parámetros de entrada

* $vectorRev: array

Vector a analizar para determinar su codificación

* $metodoVerificar: string

Nombre del método con el cual se debe codificar el vector en caso que no sea por *ami*

### Detalle del algoritmo

Para determinar que el vector se debe codificar bajo un método distinto a *ami*, se deben cumplir las siguientes reglas:

* El conteo de los items que contiene el vector, debe ser cuatro (4) u ocho (8).



* Cada uno de los valores en sus indices debe ser igual.



* Que ese valor del indice inicial sea cero.



Si todas las reglas se cumplen, se retorna el valor suministrado en el parámetro *$metodoVerificar*, de lo contrario una cadena con el texto “*ami*”

## DividirVector

Asi como se realiza un split sobre una cadena, se creó este método para realizar un split sobre un vector, pero sin eliminar los items que no pertenecen a la secuencia. Esto es usado para determinar por cada sub-vector el método de resolución que le corresponde (ver [DetectarMetodoResolucion](#_DetectarMetodoResolucion))

**Ejemplo**

Entrada: [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]

Codificación: hdb3 dividido por secuencias de cuatro (4) ceros (0) seguidos

Split: [[1, 1], [0, 0, 0, 0], [1], [0, 0, 0, 0], [0, 0, 0, 0], [1, 0, 1]]

Resaltados en amarillo están las secuencias que deben ser codificadas mediante *hdb3*, el resto va por *ami*

### Parámetros de entrada

* $patron: array

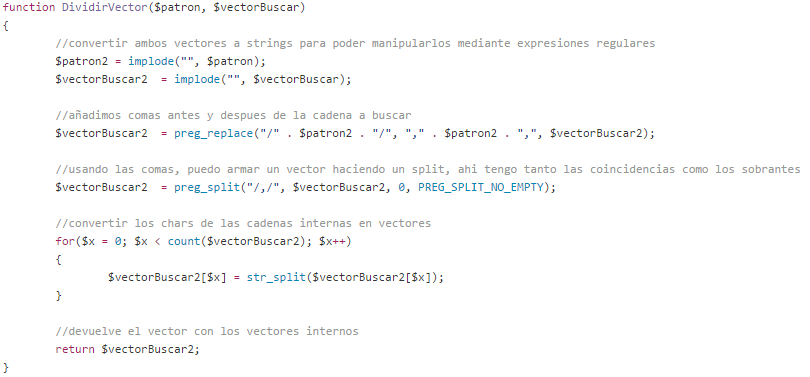
Es el item por el cual se va a realizar el split

* $vectorBuscar: array

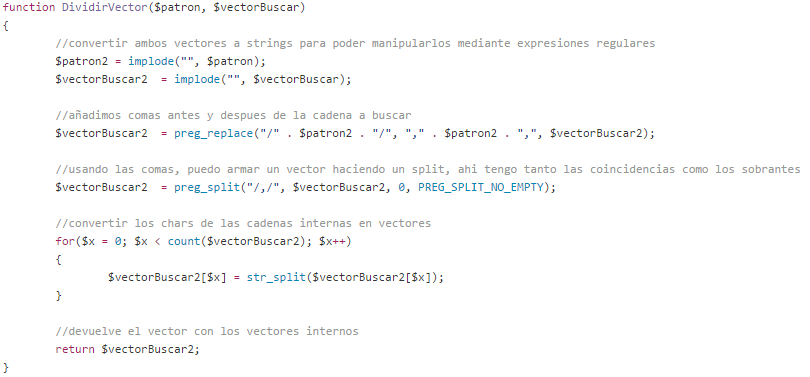
Vector al cual se le va a realizar el split

### Detalle del algoritmo

Como el split no puede realizarse directamente a un vector, usamos la función *implode*[[1]](#footnote-1) para convertir tanto el patrón de búsqueda como el vector a operar en cadenas.



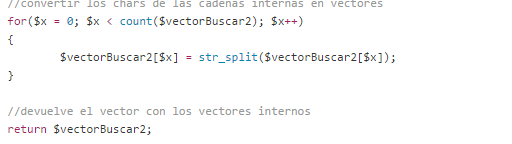
Realizamos un reemplazo del patrón a buscar concatenando comas al inicio y al final usando la función preg\_reglace[[2]](#footnote-2). De este modo tendremos un resultado similar a 11,0000,1,0000,0000,101



Hacemos de nuevo la operación inversa de convertir de cadena a vector usando la función preg\_split[[3]](#footnote-3). De este modo tendremos un resultado similar a [11],[0000],[1],[0000],[0000],[101]



Al tener el vector segmentado, se procede a convertir cada string de cada vector en un vector de caracteres, asi tendremos nuestros binarios listos para operar.



1. http://php.net/manual/en/function.implode.php [↑](#footnote-ref-1)
2. http://php.net/manual/en/function.preg-replace.php [↑](#footnote-ref-2)
3. http://php.net/manual/en/function.preg-split.php [↑](#footnote-ref-3)