MAYO 4, 2016

CODIFICACIÓN HDB3 Y B8ZS

MANUAL TÉCNICO

UNIVERSIDAD CATÓLICA DE COLOMBIA
REDES COMPUTACIONALES

Contenido

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.

```php
$secuenciaEntrada = $_GET["trama"];
$tipo = $_GET["tipo"];

//convertir trama a vector
$vectorBits = str_split($secuenciaEntrada);

switch($tipo)
{
        case "ami":
                $c = false; //cambios de señal
                $t = 0; //contador de unos
                echo RenderVector($vectorBits, ami($vectorBits, $c, $t));
                break;
        case "hdb3":
                echo RenderVector($vectorBits, hdb3($vectorBits));
                break;
        case "b8zs":
                echo RenderVector($vectorBits, b8zs($vectorBits, true));
                break;
}
```

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.

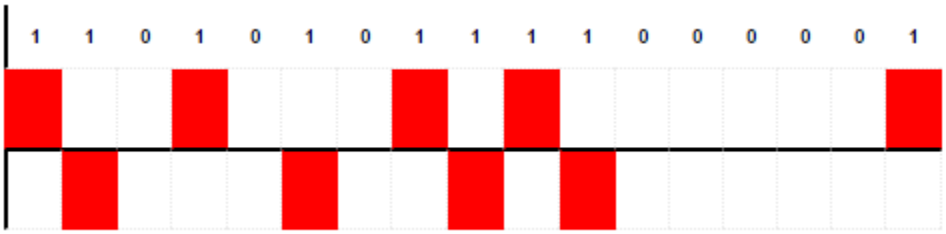
- $t : *int*
  Lleva el conteo de unos (1) tanto positivos como negativos. Se usa en posterior codificación *hdb3*.

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.

```
case "b8zs":
        echo RenderVector($vectorBits, b8zs($vectorBits, true));
        break;
```

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.

```
$bitAux = null;
$ret = [];

//recorrer todo el vector
foreach($vector as $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.

```
            if($bit == 1)
            {
                    //colocar 1 o -1 dependiendo del cambio de señal
                    $bitAux = $signalChange ? -1 : 1;

                    //cambiar señal
                    $signalChange = !$signalChange;

                    //sumar el # de 1 transmitidos independiente si es positivo o negativo
                    //luego lo usaremos para determinar si es par o impar
                    $totalUnosTransmitidos++;
            }
```

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

```
            else
            {
                $bitAux = 0;
            }
```

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

```
            //adicionar item al vector
            array_push($ret, $bitAux);
    }

    return $ret;
```

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.

```
$ret = []; //retorno

//vector de busqueda
$secuenciaBuscar = [0,0,0,0,0,0,0,0];

//Vector de reemplazo cuando el bit anterior es positivo
$secuenciaPositiva = [0,0,0,1,-1,0,-1,1];

//Vector de reemplazo cuando el bit anterior es negativo
$secuenciaNegativa = [0,0,0,-1,1,0,1,-1];
```

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.

```
//dividir el vector segun la secuencia de 8 ceros junto a sus sobrantes
$vectorCoincidencias = DividirVector($secuenciaBuscar, $vector);

//determina los cambios de señal hechos por ami
$cambioSenal = !$logicaPositiva;
$totalUnosTransmitidos = 0;

foreach($vectorCoincidencias as $vectorOperar)
```

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.

```
//determinar cual es el metodo que aplica para resolver la secuencia
$metodoResolucion = DetectarMetodoResolucion($vectorOperar, "b8zs");

switch($metodoResolucion)
{
        case "ami":
                $ret[] = ami($vectorOperar, $cambioSenal, $totalUnosTransmitidos);
                break;
        case "b8zs":
                //hacer los reemplazos de la secuencia dependiendo del signo anterior
                $ret[] = $cambioSenal == true ? $secuenciaPositiva : $secuenciaNegativa;
                break;
}
```

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.

```
//como el resultado es un vector, usamos el operador ... para concatenar todos los vectores internos
return array_merge(...$ret);
```

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.

```
$ret = []; //retorno

//vector de busqueda
$secuenciaBuscar = [0,0,0,0];

//vectores de reemplazo (reglas)
$secuenciaPositiva_Par = [-1,0,0,-1];
$secuenciaNegativa_Par = [1,0,0,1];
$secuenciaPositiva_Impar = [0,0,0,1];
$secuenciaNegativa_Impar = [0,0,0,-1];
```

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.

```
//dividir el vector segun la secuencia de 4 ceros junto a sus sobrantes
$vectorCoincidencias = DividirVector($secuenciaBuscar, $vector);

//determina los cambios de señal hechos por ami
$cambioSenal = false;
$totalUnosTransmitidos = 0;

foreach($vectorCoincidencias as $vectorOperar)
```

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.

```
//determinar cual es el metodo que aplica para resolver la secuencia
$metodoResolucion = DetectarMetodoResolucion($vectorOperar, "hdb3");

switch($metodoResolucion)
{
        case "ami":
                $ret[] = ami($vectorOperar, $cambioSenal, $totalUnosTransmitidos);
                $cambioSenal = !$cambioSenal;
                break;
        case "hdb3":
                if($totalUnosTransmitidos % 2 == 0) //Si es par
                {
                        $ret[] = $cambioSenal ? $secuenciaNegativa_Par : $secuenciaPositiva_Par;
                        $totalUnosTransmitidos = $totalUnosTransmitidos + 2;
                }
                else //es impar
                {
                        $ret[] = $cambioSenal ? $secuenciaNegativa_Impar : $secuenciaPositiva_Impar;
                        $totalUnosTransmitidos++;
                }

                break;
}
```

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.

```
//como el resultado es un vector, usamos el operador ... para concatenar todos los vectores internos
return array_merge(...$ret);
```

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).

```
((count($vectorRev) == 4 || count($vectorRev) == 8)
```

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

```
&& (count(array_unique($vectorRev)) == 1)
```

- Que ese valor del indice inicial sea cero.

```
&& $vectorRev[0] == 0
```

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)

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] para convertir tanto el patrón de búsqueda como el vector a operar en cadenas.

```
//convertir ambos vectores a strings para poder manipularlos mediante expresiones regulares
$patron2 = implode("", $patron);
$vectorBuscar2  = implode("", $vectorBuscar);
```

Realizamos un reemplazo del patrón a buscar concatenando comas al inicio y al final usando la función preg_reglace[2]. De este modo tendremos un resultado similar a 11,0000,1,0000,0000,101

```
//añadimos comas antes y despues de la cadena a buscar
$vectorBuscar2  = preg_replace("/" . $patron2 . "/", "," . $patron2 . ",", $vectorBuscar2);
```

[1] http://php.net/manual/en/function.implode.php
[2] http://php.net/manual/en/function.preg-replace.php

Hacemos de nuevo la operación inversa de convertir de cadena a vector usando la función preg_split[3]. De este modo tendremos un resultado similar a [11],[0000],[1],[0000],[0000],[101]

```
//usando las comas, puedo armar un vector haciendo un split, ahi tengo tanto las coincidencias como los sobrantes
$vectorBuscar2  = preg_split("/,/", $vectorBuscar2, 0, PREG_SPLIT_NO_EMPTY);
```

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.

```
for($x = 0; $x < count($vectorBuscar2); $x++)
{
        $vectorBuscar2[$x] = str_split($vectorBuscar2[$x]);
}

//devuelve el vector con los vectores internos
return $vectorBuscar2;
```

[3] http://php.net/manual/en/function.preg-split.php