UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

INFORME 1, ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORES.

IMPLEMENTACIÓN DE ALGORITMOS PARA TRANSFORMAR DATOS ENTRE BASES NUMÉRICAS Y CÓDIGOS

Gabriela Acuña, rol 201973504-7.
Profesor Mauricio Solar.

9 de Mayo, 2021

Índice

1 Resumen

Para la implementación de esta tarea, se decidió crear una función para cada transformación a realizar, todas nombradas de la forma *bToT*. Además se creo una función *validación* para revisar si los datos entregados se ajustan a lo solicitado en la tarea. Es importante notar que cuando se informa que el número de entrada *n* estaba en un código *b*, se consideró que *n* estaba codificado en b, por lo tanto primero se decodifica y luego se transforma a la base objetivo *t*. En cuanto a la base teorica para el desarrollo de este programa, se utilizaron las clases impartidas, el libro guía del ramo y diferentes paginas de Internet. Se obtienen los resultados esperados en base a los ejemplos entregados y vistos en clases.

2 Introducción

El objetivo de esta tarea, fue realizar un programa que transformara un dato desde cualquier base o código, a cualquier base o código, reconociendo ciertas condiciones de entrada y utilizando una variedad de algoritmos vistos en clases.

En concreto, para la mayoría de los casos, se utilizo la base decimal como intermediaria para las transformaciones. Las formulas utilizadas fueron los algoritmos de transformación para cada código, explicadas cada una en detalle mas adelante. Y la fórmula de tranformación entre bases numéricas y base decimal, para utilizarla como intermedio.

Para realizar el desarrollo de esta tarea, fue importante entender que son los Sistemas Numéricos y cada uno de los códigos a utilizar. Los Sistemas Numéricos son conjuntos de reglas y símbolos utilizados para representar números, y datos que luego pueden ser puestos en un contexto para así verlos como información. Las bases numéricas indican cuantos símbolos se utilizan para la representación de datos y según la cantidad de bits disponibles se puede extraer el rango de datos posibles de representar.

Respecto a los códigos, el Código *BCD* representa cada dígito de un numero en base decimal, en base binaria. El código Gray, también llamado código binario reflejado, fue diseñado para prevenir señales ilegales en aparatos electrodomésticos y actualmente es utilizado para corrección de errores en los sistemas de comunicación. El código exceso de 3 es un *BCD* no ponderado y se obtiene sumándole 3 a cada combinación *BCD*. El código Johnson es cíclico no ponderado y sus palabras difieren en un bit con sus vecinos. Los códigos detectores de error implementados fueron el de paridad, que revisa si la cantidad de 1 en un numero binario es par o impar. Y el código Penta Bit de Bell, que contiene palabras de 5 bits, con solo dos bits con valor 1. Por último, el código Hamming es detector y corrector de errores en los que cambia un bit, y detector de errores en 2 bits,

funciona a través de bits de paridad insertados a lo largo del dato, para que cada bit del número sea revisada más de una vez.

3 Desarrollo

Para resolver el problema propuesto en esta tarea, se organizo el programa a través de funciones que contenían el algoritmo de transformación desde y para cada base y código. Respecto a cada algoritmo implementado, en algunos casos se utilizaron los expuestos en clases , en otros estos se utilizaron algoritmos encontrados a través de investigación propia y en algunos códigos se creo un algoritmo en base a patrones encontrados en las tablas de transformación de los mismos.

A continuación se explica cada uno de los algoritmos utilizados.

3.1 De base decimal a bases numéricas y viceversa

En este caso, para pasar desde cualquier base numérica a base decimal, se implementó la *fórmula 1* ya que es bastante general, y se puede realizar de forma bastante literal con los operadores que ofrece python. Para los casos en que los dígitos del número inicial no estaban en el rango 0-9, se creo una lista con todos los dígitos posibles, en orden, y se utilizo el indice del dígito en la lista para realizar los cálculos.

$$d_1 d_2 d_3 ... d_m = d_1 \times b^{m-1} + d_2 \times b^{m-2} + d_3 \times b^{m-3} + ... d_m \times b^0 \quad (1)$$

En el caso opuesto, para transformar de decimal a cualquier base, se dividió el numero a transformar por la base objetivo, sumando el resto al resultado y utilizando la parte entera para repetir el proceso. Este algoritmo fue presentado en clases, y se tomo la decisión de aplicarlo literalmente debido a su eficiencia y lo directo que se pude realizar en python.

3.2 Binary Coded Decimal (BCD) a base decimal y viceversa

Se tomo la decisión de implementar el algoritmo para transformar de *BCD* a base decimal ya que una vez que se realiza esta transformación, rápidamente se podrá traducir el resultado a cualquier base utilizando el algoritmo presentado en la *sección 3.1*.

Para realizar la transformación de *BCD* a base decimal, se decidió implementar la definición de bcd literalmente. Se rellena con ceros a la izquierda para que el largo de 'n' sea una cantidad múltiplo de 4 de bits, y luego se recorre de a 4 bits, transformándolos de binario a decimal y agregándolos al resultado (en forma de string, para que la adición solo agregue y no los sume).

Y para realizar la transformación de decimal a bcd se transformo cada dígito a binario, luego se dejo en 4 bits[1] en caso de ser necesario, y por último se sumo al resultado.

3.3 Código de Gray a binario y viceversa

Para este algoritmo, se utilizo el operador básico que viene incluido en python, $\wedge$ (xor). Este operador representa al or exclusivo lógico, que en este caso fue muy útil. En base a las herramientas otorgadas por el lenguaje, se decidió implementar el algoritmo de transformación de código Gray a Binario que consiste en traspasar el primer dígito del valor original al valor en binario, y luego sumarle al valor en binario el resultado de la operación xor entre el segundo dígito del valor original y el último dígito agregado al valor binario. Así sucesivamente hasta que se acaba el numero codificado.

Para la transformación de un número binario y código gray, copia el primer dígito del numero a transformar. luego, se realiza la operación *xor* entre el ultimo dígito agregado al resultado y el siguiente dígito del numero a transformar. Este algoritmo considera que el numero binario esta codificado en Gray y el resultado es el indice en la tabla de este.

3.4 Código de Exceso de 3 a decimal y viceversa

En este caso, el algoritmo utilizado fue bastante apegado a la definición del código, para realizar la transformación de decimal a Exceso de 3, a cada dígito del numero en decimal se le suma 3 y luego se transforma a binario. Por último, agrega al resultado en cuatro bits y como string.

Para transformar de Exeso de 3 a decimal, se realiza el proceso inverso del anterior, se recorre el numero codificado, de a 4 bits, se transforma ese trozo a decimal, se le resta 3 y se agrega al resultado en forma de string.

3.5 Johnson a binario y viceversa

Antes de realizar la codificación de un numero a Johnson, se pueden notar las siguientes relaciones. Al realizar el calculo para utilizar la menor cantidad de bits posibles para representar un numero, se puede notar que solo se utilizará de una mitad de la tabla de transformación de Johnson ya que si el numero a transformar es menor o igual a la cantidad de bits de la tabla, este se puede codificar con menos bits. Un ejemplo para que quede clara esta idea, en el *Anexo 1*.

[1] Para cumplir con los bits necesarios, se implemento una función bastante simple llamada `toXbits` . Esta función recibe un string y un entero, y se le agregaran '0' a la izquierda al string hasta que su longitud sea igual al entero, en el caso de tener mayor o igual longitud no se realizan cambios.

Otra relación que se puede hacer entre el numero a transformar y su version codificada, es que la cantidad de ceros que posee es igual a al numero a transformar menos la cantidad de bits, y la cantidad de unos que posee es igual a dos veces la cantidad de bits menos el numero a transformar. Ejemplo en el *Anexo 1*

En base a las relaciones presentadas, para transformar de un numero binario a Johnson, primero se transforma a decimal, despues se calcula la minima cantidad de bits necesarios para representarlo y por último se realiza el calculo de la cantidad de unos y ceros pertenecientes al numero codificado.

Para transformar de Johnson a binario, considerando que el numero entregado esta representado en la menor cantidad de bits posibles, se puede utilizar la misma formula que se utilizo para cantidad de ceros para rescatar el valor original.

3.6 Revisión de Paridad

Para revisar la paridad de un numero binario, se lo trata como string, para así usar el método incluido en python, *str.count('1')* y con ese resultado, basta con utilizar la operación *resto (%)*, para saber la paridad del número.

3.7 Revision de PentaBit

En cuanto a la revisión de PentaBit, se utilizaron los operadores básicos de python y se trato el numero binario como un string para poder utilizar la funcion *len()* y así revisar si la cantidad de bits entregados son múltiplos de tres. Luego, ya que es regla del código, se revisa que solo existan dos bits con valor 1 cada cinco bits.

3.8 Correccion con código Hamming

Para la implementacion del código Hamming, se realizaron dos funciones, una pera corregir un dato codificado en Hamming, y otra para extraer el numero binario codificado en Hamming.

Respecto a la corrección, se utilizó la técnica de hacer la operación *xor* entre todas las posiciones en las cuales el dígito vale 1. Si el resultado de esta operación es 0, hay dos opciones, no hay errores, o hay más de dos errores y estos no son detectables. Si el resultado de la operación no es 0, entonces es la posición en la cual se encuentra un bit con error. Este se corrige, y se vuelve a buscar un error, si este es 0 se retorna el error corregido. Si se encuentra de nuevo un error entonces este ya e no es corregible. Por otro lado, si el resultado de la operación no es 0, pero la paridad del dato es 0, se tiene la certeza de que hay 2 errores por lo tanto no se puede corregir.

En cuanto al algoritmo para extraer el numero codificado en Hamming, es bastante simple, solo se extraen los bits de paridad, sin revisar si el dato está correcto o no.

4 Resultados

Estos resultados(*tabla 1*) son correctos tomando en cuenta todas las consideraciones expuestas en este informe y en la tarea.

Table 1: Resultados

Input	**Output**
1100 par 10	Base 10 : 12
000110 2 par	Codigo Paridad : 1
1100 pbt 10	Entrada invalida
00110 2 pbt	Codigo Pentabit : 1
00011 jsn 10	Base 10 : 8
9 10 jsn	Codigo Johnson : 10000
1000 ed3 10	Base 10 : 5
4 10 ed3	Codigo Exceso de 3 : 0111

Input	**Output**
011000110101 bcd 10	Base 10 : 635
23 8 bcd	Codigo BCD : 00011001
110 gry 3	Base 3 : 5
43 16 gry	Codigo Gray : 1111101
00110001110 ham 2	Base 2 : 100110
110001110 2 ham	Codigo Hamming : 110000110
5f 16 8	Base 8 : 137
2D 64 10	Base 10 : 167

5 Análisis

Los resultados encontrados en el desarrollo de esta tarea coinciden con lo explicado a lo largo de este informe, y cumplen con las especificaciones solicitadas respecto a los valores inválidos.

La complicación importante que encontré para el desarrollo de esta tarea, fue conservar los ceros al comienzo de un numero, que eran relevantes para código Hamming y PentaBit. Además de filtrar las condiciones y posibilidades para los input.

6 Conclusión

Se cumplieron la mayoría de los requerimientos de esta tarea, el único que no se cumplió fue la aclaración realizada en alguna de las versiones de la tarea, en la que se especificaba que si un código se encuentra en *b*, solo se debe revisar que *n* cumpla con las condiciones del código y luego transformar de binario a la base o código entregado en t. En este caso, se tiene que si había un código en *b*, *n* esta codificado en este y se debe traducir.

7 Anexo

7.1 Anexo 1: Ejemplos de codigos de Johnson

Table 2: Código de Johnson de 3 bits:

n	**jsn**
0	000
1	001
2	011
3	111
4	110
5	100

Table 3: Código de Johnson de 2 bits:

n	**jsn**
0	00
1	01
2	11
3	10

En las *tablas 2 y 3* podemos notar que el los únicos números representables en su menor potencia de bits posible son 4 y 5 para 3 bits. Ya que todos los números menores o iguales a la cantidad de bits, son representables con menos bits.

Para la segunda realización, si se va a transformar 5, la cantidad de unos a incluir sería, $(2 \times 3) - 5 = 6 - 5 = 1$ y de ceros, $5 - 3 = 2$. Y al encontrarse en la segunda mitad de la tabla por que se representa con la menor cantidad de bits posibles, los 1 van a la izquierda y los 0 a la derecha.

7.2 Anexo 2: Funciones de las bases numéricas y códigos

En cuanto a la función de las bases numéricas, fuera de lo ya visto en clases, la base octal aveces se utiliza en vez de la hexadecimal ya que no necesita símbolos diferentes a los numéricos. Esta también se utiliza para resumir a la base binaria, ya que puede representar un rango mayor de datos en la misma cantidad de bits. Por otro lado, la base 16, sistema hexadecimal, es utilizado para resumir números binarios ,ya que existe una relación lineal entre la base 2 y la base2^4. Este sistema se utiliza para definir colores en páginas web, representar direcciones de *Media Accss Control (MAC)*, mostrar la ubicación en memoria de un error, entre otros.

Una base muy interesante es la base 60, el Sistema Sexagesimal, originado en mesopotamia, donde era utilizado para medir tiempos y ángulos (en grádos). Por último la base 64, es la mayor potencia que puede ser representada utilizando exclusivamente los caracteres ASCII. Lo cual puede ser muy útil para la transmisión de caracteres especiales, y traspasar cualquier tipo de dato a un archivo de texto plano. El problema de esta base es que en vez

de comprimir los datos de texto, los expande en un 33,3%. Por otro lado, reduce considerablemente la cantidad de bits necesarios para representar otros valores numéricos[2].

Respecto a los códigos, dentro de la misma tarea se utilizo la comprobación de paridad. El código *BCD* es muy útil para representar números decimales en displays, al estar cada dígito codificado por separado, facilita la conversión a decimal pero utiliza mas bits de los que se usan con el sistema binario[3]. El código Gray, es utilizado en codificadores rotatorios y ópticos, mapas de Karnaugh y detección de errores[4].

[2] base64guru

[3] Practical Electronics/Binary-coded Decimal

[4] Tutorials Point