



Informe 1:

Abyss, de base 1 a 64

Arquitectura y Organización de Computadores

Profesor: Mauricio Solar

Estudiante: Diego Rosales León – Rol: 201810531-7

09 de Mayo de 2021

1. Resumen

En este informe se confeccionó una consola de videojuegos con el objetivo de que, al entregarle una entrada numérica lo transformara en otro tipo de información, a través de un código en específico. Por ejemplo: de base 1 a base 64, de base 10 a Bcd, de base 2 a Gray.

Para lo anterior, se analizaron diferentes casos que cumplieran con variados tipos de requisitos (número (n), base de n (b), base destino (t)). Una vez hecho esto, se distinguió entre numéricos y de código para determinar la transformación correspondiente, en el caso primero se mantuvo y en el caso segundo se convirtió a binario.

En específico se trabajaron los siguientes ejemplos, a saber, (1) el resultado entregado de n , b , t , cuando los mismos estaban fuera del rango, (2) el output cuando había igualdad entre b y t , y (3) el análisis del lugar del código en b y t .

2. Introducción

El presente trabajo trata respecto a una tarea entregada por la empresa *OculusTM*, en donde su objetivo es crear un programa que traduzca distintos tipos de códigos en bases distintas, desde código en unario (base 1), hasta código en base 64, para que la consola de juego *Abyss*, funcione correctamente. A su vez, *OculusTM* se preocupa mucho por la seguridad de la empresa, por lo que además solicita que se implementen los siguientes tipos de código:

Código	Expresión
BCD ¹	bcd
Gray	gry
Exceso de 3	ed3
Johnson	jsn
Paridad	par
PentaBit	pbt
Hamming	ham

Tabla 1. Códigos y su expresión. En la parte de anexo se incluye una referencia para cada uno de éstos.

A grandes rasgos, se busca crear un procesador que “entienda” diferentes bases numéricas y distintos tipos de códigos. Asimismo, que incluya variadas fórmulas que se encarguen de traspasar los valores a decimal y viceversa:

$$\text{valor \% base. valor // base.} \quad (1)$$

Ver anexo, ejemplo de función donde se usa esta ecuación.

Cabe mencionar, que mayoritariamente se trabajó con listas y strings:

```
list[indice].pop() .insert() .append()  
    .replace() string[: -1]
```

¹ BCD, Binary coded decimal o Decimal codificado en Binario.

3. Desarrollo

Este apartado se divide en varias partes, desde comprobar casos necesarios y obligatorios, hasta respetar cada código y base por su cuenta para finalmente convertir de unos a otros y mostrar por pantalla la codificación respectiva. Los datos que se ingresaron fueron de tres a la vez, donde n es el valor por convertir, b es la base en la que se encuentra n y t , la base a la que se quiere transformar.

N	$n \in \text{base}_{10}: 1 \leq n \leq 1000$
b^2	$b: 1 \leq b \leq 64 \vee b \in \text{código}^3$
t^4	$t: 1 \leq t \leq 64 \vee t \in \text{código}$

Tabla 2. Datos ingresados y sus respectivos requerimientos.

Es necesario entender que la base unaria sólo utiliza el símbolo de 1, mientras que las bases entre 2 y 64, pueden utilizar los siguientes símbolos:

0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+?

En caso de que no se cumpla con los requerimientos de la Tabla 2, se debe mostrar por pantalla:

Entrada Invalida

Por el contrario, si los valores ingresados son válidos, se comienza con el programa y suceden los siguientes casos:

- 1) Que b y t sean bases o códigos iguales, donde el valor a devolver es n .
- 2) Que b y t sean valores numéricos, donde el valor a devolver es el valor de n en la base t .
- 3) Que b sea un código: se debe revisar que n cumpla la condición de b , luego el valor de b se transforma a binario (Base 2) y finalmente se transforma al valor de t correspondiente.
- 4) Que t sea equivalente a un código: si es equivalente a bcd o $ed3$, el valor de n debe estar en decimal para ser codificado, y para el resto de los códigos, se necesita un n en binario.

² Códigos posibles: bcd , gry , $ed3$, jsn , par , pbt o ham .

³ Si b es código, se debe revisar que el valor de n cumpla la condición de b , y luego se asume que b es binario.

⁴ Si t es un código, para bcd y $ed3$, n debe estar en decimal para ser codificado, y para el resto de los códigos, se debe ingresar un n en binario.

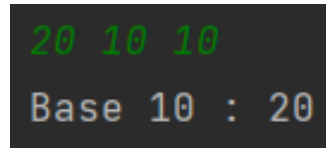


Imagen 1. Caso de prueba cuando b y t son bases iguales.

En el anexo se agregarán ejemplos de los casos restantes

Para los cuatro casos mencionados, se crearon funciones personalizadas. En el caso de las bases numéricas, basta con hacer funciones que pasen un valor de cualquier base a decimal, y viceversa, mientras que, para el caso de los códigos, se hizo una función respectiva para cada uno, donde difieren en el input⁵, el cual también se trabaja con las funciones de decimal a base y viceversa. Cuanto t es una base, el output⁶, será el valor transformado en esa base, y si es código, es un binario o decimal, respectivamente.

Input	Código y base	Output
Valor de n en su base b	Base 1 \rightarrow Base 64	Valor transformado a la base correspondiente
Decimal	bcd	Binario
Binario	gry	Binario
Decimal	ed3	Binario
Binario	jsn	Binario
Binario	par	Binario
Binario	pbt	Binario
Binario	ham	Binario

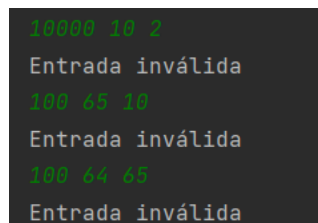
Tabla 3. Inputs y outputs de cada código y base respectivamente.

Finalmente, si se hace ingreso de un “-“, el programa finalizará sin aceptar más datos. Cualquier valor previo a esto, se muestra por pantalla.

4. Resultados

En los resultados se usaron casos de prueba para:

- 1) Los valores de entrada n , b y t
- 2) Casos donde b y t sean iguales.
- 3) Casos cuando el código está en b o t , o en ambos a la vez.



Como se aprecia en la Imagen 2, los valores de n , b y t fueron ingresados, pero eran entradas inválidas, debido a que no cumplían con los requerimientos de la Tabla 2.

Imagen 2. Casos cuando n es inválido, b es inválido y cuando t es inválido respectivamente

⁵ Input: entrada del programa.

⁶ Output: salida de un cálculo o proceso.

Para el caso de la *Imagen 3*, los resultados dados fueron ingresados de manera correcta, visualizando la existencia de coincidencias entre b y t y sus valores correspondientes por separados, ya que ellos forman el esquema para la resolución.

Imagen 3. Resultados de otras pruebas de código

```
1a 64 64
Base 64 : 1a
10 bcd 20
Base 20 : 2
7 2 gry
Codigo Gray: 101
1111 2 ham
Codigo Hamming: 11110
```

```
11110 2 phr
Codigo Pentabit: 1
101 2 jsn
Codigo Johnson: 1110
-
Fin del programa
```

Otros casos de código pueden ser apreciados en la *Imagen 4*, donde, cabe destacar cuando t es código y su respectiva resolución. Además, se puede apreciar el fin del programa.

Imagen 4. Otros resultados con pruebas distintas.

5. Análisis

Para la *Imagen 2*, los casos de prueba fueron inválidos, debido a que no cumplían con los requerimientos propuestos en la *Tabla 2*. Es por ello, que se muestra una *Entrega Inválida*. Una complicación identificada, fue el hecho de trabajar con enteros y strings, debido a que no pueden relacionarse directamente entre sí, exceptuando el caso donde una de ellas se transforme en la otra.

- En el primer caso, 10000 está en decimal, esto implica que no cumple el requisito de $10000 \notin 1 \leq n \leq 1000$
- Para el segundo caso, el error está en el valor de b , debido a que es mayor a lo permitido. $65 \notin 1 \leq b \leq 64$
- Para el tercer caso ocurre lo mismo que en el caso anterior, solo que con el valor de t . $65 \notin 1 \leq t \leq 64$

En el caso de la *Imagen 3*, se debe poner atención principalmente en los valores de b y t , debido a que son la base de la transformación de n . La complejidad de esto fue el hecho de trabajar con código en b , debido a que se asumió que el valor de n estaba en la base b , por lo que se realizó el cambio automáticamente sin una segunda revisión. Esto puede llevar a tener varios errores, sin embargo, se pasó por alto debido a la certeza de que son casos que, de cumplirse, están correctos.

- Las bases son idénticas, por lo que se imprime directamente el valor de 1^a
- Como $b = bcd$, b se reemplaza por 2 y el valor de 10 en base 2 se convierte a un valor en base 20, por lo que da 2.
- En este caso, había dificultad en crear los valores de Gray para identificar el índice, pero se obtiene con total naturalidad, se revisa si el 7 es binario, si no es así se pasa a binario, para luego pasar a Gray y buscar el índice del valor que se requiere, en binario.

- Finalmente, también fue difícil lograr la corrección de error para el código Hamming, pero se recibe el valor en binario, de 11111 y retorna 11110, que es el valor con su respectivo error corregido.

Por último, para la *Imagen 4*, la mayor complejidad fue trabajar con código Johnson, debido a la dificultad de crear la tabla con la cantidad de Bits⁷ necesarios.

- Para el caso de PentaBit, se recibe un binario y se debe revisar si el largo de ese número es múltiplo de 5. Si esto se cumple, se debe retornar *I*, en caso contrario, *0*
- Para este caso, se ingresa un binario y se deben crear los valores de la tabla con la cantidad de bits correspondientes, con un largo equivalente a la potencia de 2 suficiente (n bits permiten codificar 2^n bits), se busca el binario en la tabla y obtenemos la posición que se imprime en binario
- Finalmente, se ingresa un “-” y se aprecia que el programa termina y no se puede continuar, con esto se comprueba que el “-” termina el programa.

6. Conclusión

Después de todo, se puede apreciar que se realizó la implementación de distintas bases numéricas y de distintos tipos de códigos, para poder codificar distintos valores, de una base determinada a una nueva.

En cuanto a los resultados obtenidos, cuando existe un código en la posición de b , puede que lleve a errores, debido a que no se hizo una revisión general de que el valor de n esté realmente en la base b , y se asumió que automáticamente pasaría a ser binario, pero si no se considera este caso, debería funcionar con una total funcionalidad.

Finalmente, al referirse a las bases y los códigos, ambos tienen distintas funcionalidades. Por ejemplo, las bases se usan bastante en el área de la computación, específicamente para distintos tipos de tareas, desde seguridad hasta codificar cosas importantes. En el caso de los códigos, se usan para facilitar la conversión entre uno y otro, además de aumentar la seguridad de datos. En el caso de la tarea, los códigos fomentan la seguridad de la consola, y las bases trabajan para que la consola funcione con normalidad.

⁷ Bits, Binary Digit o Dígito Binario

7. Bibliografía/Anexo

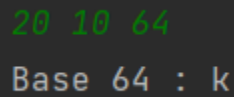
- Desconocido. (16 de mayo 2020). *Sistema de numeración unario*. <https://amp.es.autograndad.com/2428523/1/sistema-de-numeracion-unario.html>
- Base64. Wikipedia, La enciclopedia libre. 7 febrero 2021. 9 mayo 2021. <https://es.wikipedia.org/wiki/Base64>
- Notación Posicional. Wikipedia, La enciclopedia libre. 5 mayo 2021. 9 mayo 2021. https://es.wikipedia.org/wiki/Notaci%C3%B3n_posicional
- EcuRed, (s. f). *Código BCD*. https://www.ecured.cu/C%C3%B3digo_BCD
- Electrónica Unicrom. (s.f). *Código Gray*. <https://unicrom.com/codigo-gray/>
- Usuario Gissisipi. (30 de agosto del 2014). *Código de exceso 3*. *Blog Electrónica Radical*. <https://electronicaradical.blogspot.com/2014/08/codigo-de-exceso-3.html>
- LinkFang. (12 de marzo del 2020). *Código Johnson*. https://es.linkfang.org/wiki/C%C3%B3digo_Johnson
- Paridad (Telecomunicaciones). Wikipedia, La enciclopedia libre. 7 octubre 2020. 9 mayo 2021. [https://es.wikipedia.org/wiki/Paridad_\(telecomunicaciones\)](https://es.wikipedia.org/wiki/Paridad_(telecomunicaciones))
- Código Hamming y otros. Wikipedia, La enciclopedia libre. 11 marzo 2021. 9 mayo 2021. https://es.wikipedia.org/wiki/C%C3%B3digo_Hamming.

```
def dec_to_cualquier_base(n, b):  
    n = int(n)  
    new_number = []  
    base_nueva = ""  
    if b == 1:  
        base_nueva += "1" * n #Caso que sea unario!!  
    else:  
        while(n != 0): #Si es 0, ya no hay numero al que buscar resto  
            new_number.append(str(n % b))  
            n = n // b  
        new_number.reverse() #Lo ordenamos de atrás hacia adelante  
        for pos in range(len(new_number)): #Buscamos el valor en la posicion correspondiente.  
            base_nueva += bases_posibles[int(new_number[pos])]  
    return base_nueva  
#Si retorno base_nueva, me trae el valor directamente basado en la lista de bases posibles. ej(1e23d)
```

Imagen 5. Uso de la ecuación (1)

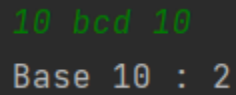
- Acrónimo:** BCD es Binary coded decimal o Decimal codificado en Binario.
- Acrónimo:** Bits, Binary Digit o Dígito Binario

8. Bibliografía/Anexo



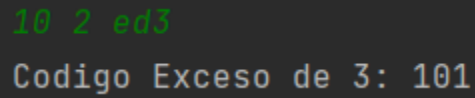
```
20 10 64
Base 64 : k
```

Imagen 6. caso cuando las dos bases (b y t) son numéricas.



```
10 bcd 10
Base 10 : 2
```

Imagen 7. Caso donde el valor de b es un código, se transforma a binario y n se pasa a la base t



```
10 2 ed3
Codigo Exceso de 3: 101
```

Imagen 8. Caso donde se tiene un código en t, el 10 en base 2 se transforma a decimal, y se transforma a ed3