Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

De código Gray binario a código Gray ternario

Iván Charabora

Directora: **Dra. Verónica Becher**

Noviembre 2024

Resumen

Los códigos Gray son secuencias circulares de palabras de una longitud fija dada donde cada palabra difiere de la anterior sólo en una posición. Se las usa en sistemas digitales por su conveniencia para reducir errores pues minimiza las variaciones entre valores consecutivos. Aunque estos códigos se usan mucho en sistemas binarios, hoy en día muchos sistemas necesitan trabajar con más valores lo que hace necesario extender los códigos Gray a sistema ternarios, o sistemas con una cantidad mayor de símbolos.

En esta tesis damos un algoritmo que transforma un código Gray en un alfabeto dado a un código Gray en otro alfabeto de más símbolos, de manera tal que el código original sea una subsecuencia del código obtenido. Comenzamos con la transformación de un código Gray binario a uno ternario. Generalizamos esto para pasar de un código de b a otro de $b+1$ símbolos, para cualquier $b \geq 2$. Aplicando esta transformación δ veces, conseguimos una transformación de b símbolos a $b+\delta$ símbolos. El código Gray que obtenemos contiene al código Gray como subsecuencia. Para lograrlo, introducimos conceptos como "pétalos" y "superpétalo" que permiten insertar las palabras adicionales necesarias y dar código extendido.

Presentamos dos variantes de nuestro algoritmo. Uno es muy simple. El otro es de flujo máximo, que asegura que las palabras insertadas estén más repartidas a lo largo del código original. Damos una cota superior de la complejidad temporal del algoritmo.

Con este trabajo mostramos que los códigos Gray pueden extenderse haciendo inserciones de palabras, hasta obtener un nuevo código Gray en más símbolos, que contiene como subsecuencia al código Gray original.

Índice

1. Introducción y resultados **3**

2. Definiciones y propiedades **6**
2.1. Código Gray binario reflexivo 6
2.2. Código Gray ternario 6
2.3. Shift de un código Gray 8
2.4. Pétalos 9
2.5. Superpétalo 10

3. Algoritmo simple **11**
3.1. Inserción de P_0 11
3.2. Inserción de P_{n-1} en P_0 13
3.3. Inserción de P_i en P_{i+1} 13
3.4. Complejidad del Algoritmo simple 14

4. Algoritmo de flujo máximo **16**

5. Ejemplos **19**
5.1. Ejemplo gráfico 19
5.2. Ejemplo con el algoritmo simple 20
5.3. Ejemplo con el algoritmo de flujo máximo 22

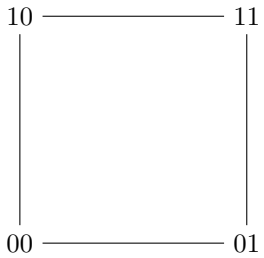
6. Código Gray en b símbolos a código Gray en $b+1$ símbolos **27**
6.1. Construcción código Gray en b símbolos 27
6.2. Definiciones y notaciones 28
6.3. Nuevos pétalos 29
6.4. Algoritmo simple para alfabetos arbitrarios 31
6.4.1. De alfabeto con cardinalidad par a impar 31
6.4.2. De alfabeto con cardinalidad impar a par 32

7. Bibliografía **34**

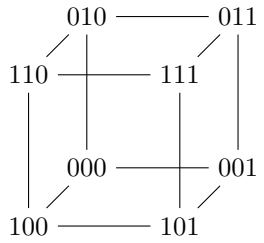
1. Introducción y resultados

Un código Gray de longitud n en b símbolos es un listado exhaustivo de las b^n palabras distintas de longitud n, en el cual las palabras sucesivas difieren exactamente en una posición. En un código Gray de longitud 4 en 2-símbolos, es decir binario, las palabras "0001" y "0011" pueden ser sucesivas. Dicho de otra forma, un código Gray de longitud n en b símbolos es un ordenamiento de todas las palabras de longitud n que se pueden formar con los b símbolos de forma tal que las palabras sucesivas tiene una distancia de Hamming de uno. Además, este ordenamiento es circular, lo que significa que para cualquier convención de inicio del listado, la última palabra tiene como siguiente a la primera. Una presentación sobre códigos Gray y varios algoritmos distintos para generarlos se puede leer del libro de Knuth [8].

Los códigos Gray binarios, se corresponden con ciclos Hamiltonianos en un grafo hipercúbico de dimensión n, donde cada vértice representa una palabra binaria. Dos vértices están conectados si las palabras correspondientes difieren en exactamente un bit. De esta manera, cada código Gray binario de longitud n es un ciclo Hamiltoniano que recorre todos los vértices (palabras binarias) del grafo hipercúbico, cambiando solo un bit en cada paso. Tomemos el caso de $n = 2$, donde el grafo hipercúbico correspondiente tiene 4 vértices, que representan las palabras binarias $00, 01, 10$ y 11. Los vértices están conectados si sus cadenas difieren en un solo bit, lo que da lugar al siguiente grafo:



Para extender esta construcción a $n = 3$, tomamos el grafo hipercúbico de dimensión 3. Este grafo tiene 8 vértices, que corresponden a las palabras binarias de 3 bits: $000, 001, 010, 011, 100, 101, 110$ y 111. Visualmente, podemos imaginar este grafo como dos copias del grafo de dimensión 2, una para las palabras con prefijo 0 y otra para las palabras con prefijo 1, conectadas entre sí de la siguiente manera:



Los códigos Gray son una herramienta fundamental en el mundo digital donde los sistemas deben minimizar errores al cambiar de estado rápidamente, asegurando precisión en la transmisión de información. Por ejemplo, se usan para prevenir estados incorrectos de interruptores electromecánicos o para facilitar la corrección de errores en comunicaciones digitales. El uso del código de Gray ayuda a simplificar las operaciones lógicas y a reducir los errores en la práctica. Sin embargo, a medida que la tecnología avanza, surge la necesidad de sistemas de codificación más flexibles y potentes que puedan manejar más de dos estados. Aquí es donde los códigos Gray ternarios pueden jugar un papel importante. La capacidad de transformar un código Gray binario en uno ternario permite extender su aplicabilidad a sistemas que requieren tres o más estados, como los sistemas de comunicación moderna y los dispositivos electrónicos que operan en múltiples niveles de voltaje.

¿Dado un código Gray para el alfabeto binario $\{0, 1\}$, de cualquier longitud n, siempre hay otro para el alfabeto $\{0, 1, 2\}$ de la misma longitud n que mantiene el orden de las palabras del código Gray binario original?

En esta tesis contestamos que sí. Damos un algoritmo para transformar un código Gray binario en un código Gray ternario de manera tal que el código binario original sea una subsecuencia del código obtenido. Nuestro algoritmo se generaliza para transformar un código Gray en un alfabeto dado a cualquier otro alfabeto más grande.

Teorema 1. *Dado un código Gray en b símbolos de longitud n, existe un código Gray en $(b+1)$-símbolos también de longitud n que tiene como subsecuencia al código Gray original.*

Aplicar el Teorema repetidas veces, digamos δ veces, da una transformación de un código Gray de longitud n en b símbolos en un código Gray en $b + \delta$ símbolos tal que el código original es una subsecuencia del código final.

Nuestro algoritmo no contempla que el código extendido preserve propiedades del código original. Por ejemplo, un código Gray es balanceado [4] cuando los cambios en los las posiciones están lo más equidistribuidas posible. Nuestro algoritmo no garantiza que el resultado de preserve la propiedad de balance.

El problema de extender códigos Gray de un alfabeto a otro más grande fue considerado anteriormente para secuencias de Bruijn [3, 1]. Para otros resultados sobre combinatoria de palabras se puede consultar [2].

Para demostrar el Teorema 1, primero presentamos un algoritmo que transforma un código Gray de un alfabeto binario a uno ternario. Luego, generalizamos este algoritmo para convertir un código Gray de un alfabeto de b símbolos a uno de $b + 1$ símbolos, para $b \geq 2$. Además, proponemos un algoritmo más eficiente específicamente para el caso de alfabeto binario a ternario, y argumentamos que este también puede generalizarse para transformar un alfabeto de b símbolos a uno de $b + 1$.

Mostraremos que es posible insertar en cualquier código Gray binario de longitud n todas las palabras ternarias de longitud n que contienen un "2", de manera tal de obtener un código Gray ternario de longitud n. Consideremos el siguiente ejemplo. Dado un código Gray binario de $n = 2$:

0	0
0	1
1	1
1	0

Nuestro objetivo es transformar este código Gray binario en un código Gray ternario insertando las palabras restantes (02, 12, 20, 21, 22).

0	0
0	2
0	1
2	1
2	2
1	2
1	1
1	0
2	0

Cuadro 1: Código Gray ternario manteniendo el orden del binario

Proporciones y cantidades. Presentamos a continuación las proporciones y cantidades relacionadas con las palabras binarias y ternarias y la cantidad de códigos Gray en cada caso. En

las siguientes tablas mostramos la cantidad de palabras, los posibles ordenes con esas palabras empezando por la cadena de n 0s y la cantidad de códigos Gray.

n	Palabras binarias	Permutaciones de 2^n palabras binarias	Códigos Gray binarios
1	2	$1! = 1$	1
2	4	$3! = 6$	2
3	8	$7! = 5.040$	12
4	16	$15! \approx 1.31$ x 10^{12}	2688
5	32	$31! \approx 8.2$ x 10^{33}	1.813.091.520

Cuadro 2: Cantidades para el alfabeto binario

n	Palabras ternarias	Permutaciones de 3^n palabras ternarias	Códigos Gray ternarios
1	3	$2! = 2$	2
2	9	$8! = 40.320$	96
3	27	$26! \approx 4.03$ x 10^{26}	$\approx$ 2.690.217.600

Cuadro 3: Cantidades para el alfabeto ternario

Determinamos esta cantidad de códigos Gray binarios y ternarios mediante un conteo exhaustivo. Las cantidades obtenidas en el caso binario coinciden con las reportadas en la bibliografía. Las cantidades en el caso ternario no aparecen en la bibliografía [7]. Ninguno de los dos casos tiene una fórmula cerrada. La cantidad de códigos Gray binarios de n bits es igual a la cantidad de ciclos Hamiltonianos en un grafo hipercúbico de dimensión n.

El espacio de trabajo para el alfabeto ternario es considerablemente mayor que para el alfabeto binario. Por ejemplo, para longitud $n = 2$ hay 48 veces más códigos Gray ternarios que binarios y para $n = 3$ hay 224.184.800 veces más. Sin embargo, no todos los códigos Gray ternarios contienen una subsecuencia que es un código Gray binario. Por ejemplo, en $n = 2$ donde el 83 % de los códigos Gray ternarios son códigos Gray binarios con inserciones de palabras ternarias (es decir, si se eliminan las palabras ternarias, queda un código Gray binario) pero solo un 16 % lo es en $n = 3$.

2. Definiciones y propiedades

2.1. Código Gray binario reflexivo

Sea el alfabeto $A = \{0, 1\}$, buscamos construir un código Gray donde cada palabra difiere en un bit. Para lograrlo, se usa una técnica denominada reflexión. Reflejar un código Gray implica invertir el orden de las palabras, de modo que la última palabra se convierte en la primera, y así sucesivamente. Una vez reflejado, se añaden ceros a las palabras originales y unos a las palabras reflejadas.

Por ejemplo, si empezamos con un código Gray de longitud $n = 1$, como podría ser 0-1, y queremos generar uno de longitud $n = 2$, procedemos de la siguiente manera:

0
1

Reflejamos el código:

1
0

Luego, agregamos ceros al principio de las palabras originales y unos al principio de las palabras reflejadas para formar el código Gray de longitud $n = 2$.

0	0
0	1
1	1
1	0

Cuadro 4: Código Gray binario de $n = 2$ (00-01-11-10)

Este proceso se puede repetir para generar códigos Gray de cualquier longitud, simplemente reflejando el código existente y agregando ceros y unos según corresponda. La presentación original del código Gray reflexivo aparece en [5].

2.2. Código Gray ternario

La construcción de un código Gray ternario que presentamos a continuación sigue el enfoque propuesto en [6].

Ahora, consideremos el alfabeto ternario $A = \{0, 1, 2\}$ y construyamos un código Gray donde la transición entre palabras sucesivas cambie solo un símbolo (trit).

Primero, construimos un código Gray de longitud $n = 1$. Por ejemplo:

0
1
2

Luego, para extenderlo formamos otro código Gray comenzando con la última palabra del código anterior:

0
1
2
2
0
1

Agregamos una secuencia adicional que comience con el último símbolo de la segunda secuencia y termine con el primero de la primera secuencia.

0
1
2
2
0
1
1
2
0

Cuadro 5: Patrón circular que repite 3 veces cada símbolo

Para transformar este patrón en un código Gray de longitud $n = 2$, copiamos el código de $n = 1$ y lo extendemos, repitiendo cada símbolo 3 veces en la siguiente posición. Luego, para extenderlo a uno de $n = 3$, hacemos un procedimiento similar. Primero, triplicamos la longitud del símbolo menos significativo repitiéndolo. Para la segunda columna, usamos el patrón de la Tabla 5 y extendemos las triplas restantes, siguiendo el mismo procedimiento que en $n = 2$. Para la última columna, volvemos a agarrar la primer tripla y la estiramos pero esta vez 9 símbolos por símbolo.

0	0
0	1
0	2
1	2
1	0
1	1
2	1
2	2
2	0

0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	1
0	2	0	2	0	0	0	0	2
1	2	1	2	0	1	0	1	2
1	0	1	0	0	1	0	1	0
1	1	1	1	0	1	0	1	1
2	1	2	1	0	2	0	2	1
2	2	2	2	0	2	0	2	2
2	0	2	0	0	2	0	2	0
	0	2	0	1	2	0	1	2
	1	2	1	1	2	0	1	0
	2	2	2	1	2	0	1	1
	2	0	2	2	2	0	2	1
	0	0	0	2	2	0	2	2
	1	0	1	2	2	0	2	0
	1	1	1	1	2	1	0	1
	2	1	2	1	2	1	0	2
	0	1	0	1	2	1	0	0
	1	0	1	1	2	1	0	1
	1	1	1	1	2	1	1	1
	2	1	2	1	2	1	1	2
	0	1	0	1	2	1	1	0
	0	1	0	2	2	1	2	0
	1	1	1	2	2	1	2	1
	2	1	2	2	2	1	2	2
	2	2	2	2	2	2	2	2
	0	2	0	2	2	2	2	0
	1	2	1	2	2	2	2	1
	1	0	1	0	2	2	0	1
	2	0	2	0	2	2	0	2
	0	0	0	0	2	2	0	0

Código Gray ternario de $n = 2$ Construcción de código Gray ternario de $n = 3$

En resumen, para pasar de un código Gray ternario de longitud n a uno de $n+1$ tenemos que:

- Para las columnas de 0 a $n-1$: repetir 3 veces.
- Para la columna n: agregar las últimas dos triplas (201 y 120) con estiramientos (repetir 3^{n-1} veces cada símbolo).
- Para la columna $n+1$: agregar la primer tripla (012) con estiramientos (repetir 3^n cada símbolo).

Distancia de Hamming. La distancia de Hamming es una medida utilizada para cuantificar la diferencia entre dos tiras de símbolos de igual longitud, definida como el número de posiciones en las que los símbolos correspondientes son diferentes. En el contexto de los códigos Gray, esta distancia es utilizada para garantizar que solo un dígito cambie entre valores consecutivos. Una propiedad que usaremos a lo largo del documento es $HammingDistance(u, v) = HammingDistance(v, u)$ para todo u, v.

Código Gray parcial. Definimos código Gray parcial de longitud n a una lista de palabras distintas que cumple la propiedad de tener $HammingDistance = 1$ en toda palabra consecutiva y también la circularidad. A diferencia de un código Gray, el código Gray parcial no tiene todas las palabras posibles de longitud n.

2.3. Shift de un código Gray

La operación de shift de una palabra $a_1a_2 \dots a_n$ es la palabra $a_2a_3 \dots a_na_1$, es decir, se obtiene desplazando todos los símbolos una posición a la izquierda y colocando el primer símbolo al final. Este shift se puede aplicar reiterada veces.

Para demostrar la existencia del código Gray ternario buscado usaremos que la función shift aplicada a cada palabra de un código Gray produce otro código Gray. Esto se debe a que el símbolo distinto entre cada dos palabras sólo se mueve de posición y, por otro lado, todas las palabras de la longitud n seguirían estando solo que, posiblemente, en otra posición.

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Cuadro 6: Código reflexivo de longitud $n = 3$ marcando en cada palabra el único símbolo distinto al de la palabra anterior.

Los posibles resultados distintos a base de shifts son:

0	0	0
0	1	0
1	1	0
1	0	0
1	0	1
1	1	1
0	1	1
0	0	1

0	0	0
1	0	0
1	0	1
0	0	1
0	1	1
1	1	1
1	1	0
0	1	0

Cuadro 7: Códigos Gray de longitud $n = 3$ con un y dos, respectivamente, shifts a izquierda del código reflexivo.

2.4. Pétalos

En vías de dar un método para transformar un código Gray binario en otro ternario definimos listas de palabras ternarias a las que llamamos *pétalos*. Dentro de cada pétalo damos un código Gray parcial. Cada pétalo agrupa las palabras según la posición de su "2" más significativo. Esto lo que genera, en un código Gray de longitud n, son n pétalos donde cada palabra del pétalo tiene una parte binaria, un "2" y una parte ternaria.

Por ejemplo:

2	0
2	1
2	2

0	2
1	2

Cuadro 8: Listas de palabras para $n = 2$. La parte verde es binaria, la parte azul es "2", y la parte celeste es ternaria.

2	0	0
2	0	1
2	0	2
2	1	0
2	1	1
2	1	2
2	2	0
2	2	1
2	2	2

0	2	0
0	2	1
0	2	2
1	2	0
1	2	1
1	2	2

0	0	2
0	1	2
1	1	2
1	0	2

Cuadro 9: Listas de palabras para $n = 3$.

Cada lista es un código Gray parcial. A cada lista la llamamos pétalo.

Observaciones:

- Los pétalos no tiene ninguan palabra en común. Es decir, no puede haber una palabra en dos pétalos distintos.
- Todas las palabras ternarias que no están en el código Gray binario están en algún pétalo.
- Si llamamos P_i al pétalo con las palabras con su "2" más significativo en la posición i (por ejemplo P_0 sería el que tiene toda parte binaria con un "2" al final), P_i tiene $2^{n-1-i} * 3^i$ palabras.

Códigos Gray parciales en pétalos. Mostraremos que, dado un código Gray binario arbitrario de longitud n, podemos insertarle cada uno de los n pétalos de manera de obtener un código Gray ternario.

Notemos que los códigos Gray parciales son circulares. Por lo tanto, podemos usar como primer palabra del pétalo a cualquiera que pertenece. Al insertar un pétalo en un código Gray parcial, obtenemos otro código Gray parcial. Para cada $i = 0, 1, 2, \ldots, n-1$ definimos el pétalo P_i dando un código Gray binario de longitud $n - 1 - i$. A cada una de las palabras de este código Gray la concatenamos con un código Gray ternario de longitud i, insertando el "2" entre medio. Para mantener la propiedad de distancia de Hamming uno, al cambiar de una palabra binaria a otra, es necesario reflejar el código Gray ternario. Por lo tanto, en la construcción hicimos tantas reflexiones como cantidad de elementos del código Gray binario.

Veamos que la lista circular que construimos es un código Gray parcial.

En cada sublista dada por una reflexión la parte binaria queda fija, el "2" queda fijo y la parte ternaria tiene distancia de Hamming 1, por ser código Gray. Entre dos de estas sublistas también tenemos distancia de Hamming 1, porque la parte ternaria se mantiene pero la parte binaria tiene distancia de Hamming 1.

Dado que el código Gray binario original es circular, este nuevo código también lo es y la distancia de Hamming entre la última cadena de la última lista y la primer cadena de la primer lista es exactamente el caso anterior (análisis entre dos sublistas).

El pétalo P_i para $i = 1$ en $n = 4$ se define así. Primero, generamos un código Gray binario de longitud $n' = n - i - 1 = 2$. Para esto, usamos el código Gray binario reflexivo explicado en la sección Código Gray binario reflexivo.

0	0
0	1
1	1
1	0

Luego, hacemos lo mismo con el código Gray ternario de longitud $n'' = i = 1$ con la estrategia explicada en Código Gray ternario

0
1
2

Por último, agregamos el "2" entre medio.

0	0	2	0
0	0	2	1
0	0	2	2
0	1	2	2
0	1	2	1
0	1	2	0
1	1	2	0
1	1	2	1
1	1	2	2
1	0	2	2
1	0	2	1
1	0	2	0

Cuadro 10: Pétalo P_1 para $n = 4$

2.5. Superpétalo

Un superpétalo es una lista de n pétalos.

3. Algoritmo simple

Veamos que es siempre posible insertar cada uno de los pétalos en el código Gray binario dado.

Algoritmo 1 (Algoritmo simple)**.**
Sea B un código Gray binario de longitud n.
Construir los pétalos $P_0, P_1, \dots P_{n-1}$.
Insertar P_0 dentro de B y guardar el resultado en res.
Insertar P_{n-1} dentro de P_0 en res.
Para $i = n-2, \dots 1$, insertar P_i en P_{i+1}.
El output es un código Gray ternario de longitud n que tiene como subsecuencia al código Gray binario B.

Primero, definamos cuándo podemos poner un código Gray parcial dentro de otro. Para la inserción, necesitamos un par de palabras consecutivas en cada código que puedan ser consecutivas con las del otro código en el código Gray parcial resultante de juntarlos. Dicho de otra forma, sea u_1, u_2 palabras consecutivas en un código Gray parcial y v_1, v_2 en otro, tenemos dos opciones:

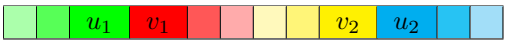
- $HammingDistance(u_1, v_1) = 1$ y $HammingDistance(u_2, v_2) = 1$
- $HammingDistance(u_1, v_2) = 1$ y $HammingDistance(u_2, v_1) = 1$

Para entender mejor esto, veamos un ejemplo. Supongamos que tenemos el código Gray parcial G_1 con las palabras u_1 y u_2

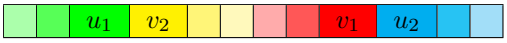
Y el código Gray parcial G_2 con las palabras v_1 y v_2

Tenemos los dos casos:

- $HammingDistance(u_1, v_1) = 1$ y $HammingDistance(u_2, v_2) = 1$



- $HammingDistance(u_1, v_2) = 1$ y $HammingDistance(u_2, v_1) = 1$



En ambos casos pudimos ingresar un código Gray parcial dentro de otro y obtuvimos un nuevo código Gray parcial.

3.1. Inserción de P_0

El primer pétalo que insertamos es P_0 entre la primera palabra (0^n) y la siguiente o anterior. Recordemos que P_0 es el que tiene un "2" en la posición menos significativa y ningún otro. Por ejemplo para $n = 2, 3, 4$:

0	2
1	2

0	0	2
0	1	2
1	1	2
1	0	2

0	0	0	2
0	0	1	2
0	1	1	2
0	1	0	2
1	1	0	2
1	1	1	2
1	0	1	2
1	0	0	2

No siempre se puede insertar un pétalo en la primera palabra del código Gray binario sin realizar cambios en el pétalo. Por ejemplo, en el caso de $n = 4$, es posible insertar el pétalo si 0010 o 1000 son consecutivos 0000, pero no si 0001 o 0100 lo son. Con esto en mente, surge la idea de hacer este pétalo dinámicamente dependiendo que le sigue a 0^n para que se pueda insertar siempre.

Además sabemos que la primera palabra del código Gray binario, de longitud $n-1$, construido para el pétalo es 0^{n-1} y la segunda todo "0" con un "1" en alguna posición. Por lo tanto, si shifteamos estas dos palabras, tenemos que el código empieza con 0^{n-1} (no modifica al ser shifteado) y sigue con todo "0" con un "1" en la posición que se quiera dependiendo cuantos shifts se hacen. Entonces, la idea es conectar la palabra 0^n del código Gray binario original con $0^{n-1}2$ perteneciente a P_0 y la siguiente del código con la siguiente del pétalo.

Un detalle a tener en cuenta, cuando queremos conectar la segunda palabra de P_0 con la del código, es que P_0 se construye mediante un código Gray binario de longitud $n-1$. Entonces, con los shifts, puede tener $n-1$ posibles segundas palabras pero hay n posibles segundas palabras en el código Gray binario. El problema pasa cuando la segunda palabra del código Gray binario es $0^{n-1}1$. Esto es porque la palabra del pétalo que se debería conectar con ésta es $0^{n-1}2$ pero ya se usa para conectar el pétalo con 0^n.

Entonces, vamos a dividir el análisis en 2 casos.

El primer caso es que en B la palabra siguiente a 0^n no es $0^{n-1}1$, sino que es $0^{n-\ell}10^{\ell}$ con $0 < \ell$. Necesariamente una de las $n-1$ variantes de P_0 tendrá $0^{n-1}2$ seguido de $0^{n-\ell}10^{\ell-1}2$.

Por ejemplo, para $n = 4$, podemos generar los siguientes P_0:

0	0	0	2
0	0	1	2
0	1	1	2
0	1	0	2
1	1	0	2
1	1	1	2
1	0	1	2
1	0	0	2

0	0	0	2
0	1	0	2
1	1	0	2
1	0	0	2
1	0	1	2
1	1	1	2
0	1	1	2
0	0	1	2

0	0	0	2
1	0	0	2
1	0	1	2
0	0	1	2
0	1	1	2
1	1	1	2
1	1	0	2
0	1	0	2

Estas variantes de P_0 aseguran que podemos conectar P_0 con cualquiera de las palabras siguientes a 0^n, menos $0^{n-1}1$. Se puede ver en el siguiente cuadro:

0	0	0	0
0	0	0	2
1	0	0	2
1	0	1	2
1	1	1	2
1	1	0	2
0	1	0	2
0	1	1	2
0	0	1	2
0	0	1	0

0	0	0	0
0	0	0	2
0	0	1	2
0	1	1	2
1	1	1	2
1	0	1	2
1	0	0	2
1	1	0	2
0	1	0	2
0	1	0	0

0	0	0	0
0	0	0	2
0	1	0	2
1	1	0	2
1	1	1	2
0	1	1	2
0	0	1	2
1	0	1	2
1	0	0	2
1	0	0	2

En el segundo caso, si en B la siguiente palabra a 0^n es $0^{n-1}1$, simplemente se inserta P_0 entre 0^n y la palabra anterior $0^{n-\ell}10^\ell$, con el mismo argumento que en el primer caso.

3.2. Inserción de P_{n-1} en P_0

Insertaremos el último pétalo, P_{n-1}, sin modificarlo en P_0. Para usar la técnica de 3 damos un par de palabras consecutivas en P_{n-1} y un par en P_0. Estos dos pares son:

- $20^{n-2}2$ y $20^{n-3}12$ en P_{n-1}.
- $10^{n-2}2$ y $10^{n-3}12$ en P_0.

Para ver que $20^{n-2}2$ y $20^{n-3}12$ están consecutivos en P_{n-1}, vemos la construcción del pétalo. Este pétalo es un código Gray ternario, construido como 2.2, de longitud $n-1$ con un "2" en la posición más significativa. Por lo tanto, alcanza con verificar que, en la construcción del código Gray ternario, siempre $0^{n-2}2$ y $0^{n-3}12$ estén consecutivos. Esto pasa siempre debido a que las primeras tres palabras del código Gray son 0^x00, 0^x01 y 0^x02 y luego viene el 0^x12.

Ahora, para hacer el mismo análisis para el P_0 hay que tener en cuenta que ese pétalo es dinámico. Entonces podemos demostrar que $10^{n-2}2$ y $10^{n-3}12$ está en todos los shifts de P_0 o, equivalentemente, que todos los shifts de $10^{n-2}2$ y $10^{n-3}12$ esté en algún P_0 fijo. En realidad, el "2" es el que le agregamos al pétalo. Entonces, buscamos ver que todos los shifts de 10^x y $10^{x-1}1$ estén en el código Gray reflexivo binario. Este análisis, lo vamos a dividir en más casos:

- 10^x y $10^{x-1}1$ siempre están consecutivos en el código Gray binario reflexivo, son las últimas dos palabras.
- 010^{x-1} y 110^{x-1} (un shift a derecha) también siempre están consecutivos en el reflexivo. 10^{x-1} es la última palabra de un código reflexivo de un bit menor. Al extenderlo tenemos que reflejar el código, a la primera mitad agregarle un "0" y a la otra mitad un "1". Por lo tanto, quedan juntas 010^{x-1} y 110^{x-1}.
- 0^y010^z y 0^y110^z ($y+1$ con $1 \leq y \leq x-1$ shifts a derecha, cualquier otro caso). Sabemos que ambas palabras comienzan con un 0. Por la construcción del reflexivo, que agrega "0" a la primera mitad después de reflejar, sabemos que esas dos palabras están en la primera mitad. Eso significa, que esas dos palabras estaban en el código de un bit menos. Por lo tanto, hay que fijarse en $0^{y-1}010^z$ y $0^{y-1}110^z$ que puede caer en los casos de arriba o en este y seguir iterando. Esto tiene fin porque en algún momento el primer bit de la primera palabra es "0" y de la segunda un "1" (cae en algun caso anterior).

3.3. Inserción de P_i en P_{i+1}

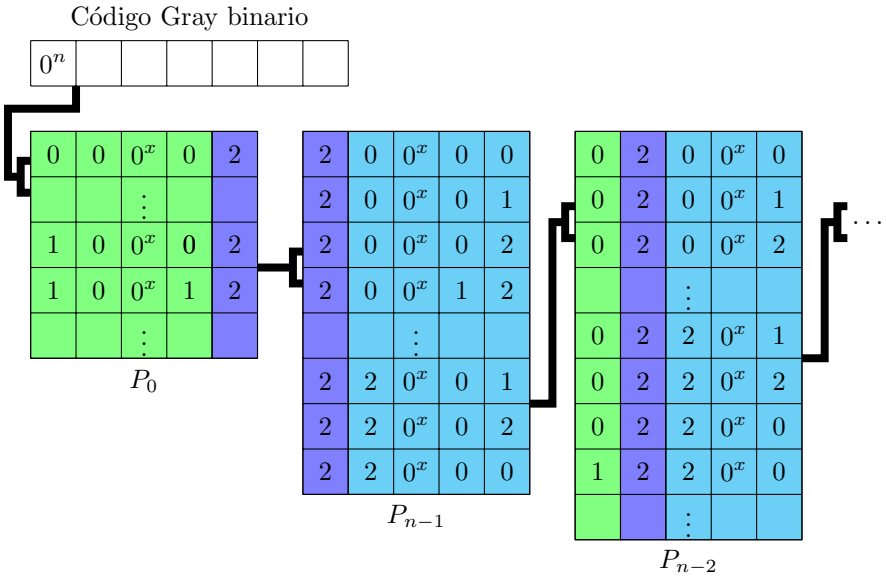
Ya tenemos P_0 y P_{n-1} insertados en el código Gray original. Ahora, queremos mostrar que podemos insertar P_i en P_{i+1} para $i = n-2, n-3, \ldots 1$. Necesitamos dos palabras consecutivas en P_i y dos en P_{i+1}. Estos pares son:

- $0^x220^{i-1}1$ y $0^x220^{i-1}2$ en P_{i+1} con $x = n - i - 2$.
- $0^x020^{i-1}1$ y $0^x020^{i-1}2$ en P_i.

Para ver que $0^x220^{i-1}1$ y $0^x220^{i-1}2$ son consecutivos en P_{i+1}, examinamos nuevamente la construcción de este pétalo. En todas las palabras tenemos una parte binaria (en este caso 0^x en ambas palabras), un "2" en la posición $i+1$ y una parte ternaria ($20^{i-1}1$ y $20^{i-1}2$ respectivamente). Al tener la misma parte binaria, por construcción de la sección 2.4, solo tenemos que ver si $20^{i-1}1$ y $20^{i-1}2$ aparecen consecutivos en la construcción del código Gray ternario. De hecho, siempre pasa eso debido a que el código siempre termina con $20^{i-1}0$ y le sigue $20^{i-1}2$ y $20^{i-1}1$, como queríamos probar.

Luego, para ver que $0^x020^{i-1}1$ y $0^x020^{i-1}2$ son consecutivas en P_i, hacemos lo mismo pero demostrando que $0^{i-1}1$ y $0^{i-1}2$ son consecutivas en el código Gray ternario construido para el pétalo. Esto pasa siempre ya que empieza con $0^{i-1}0$ y le sigue $0^{i-1}1$ y $0^{i-1}2$.

Un paso a paso se podría ver así:



3.4. Complejidad del Algoritmo simple

Como es usual usamos la notación asintótica O grande : dadas dos funciones $f, g : \mathbb{N} \to \mathbb{R}$ decimos $f(n)$ es $O(g(n)$ si existe una constante C tal que para todo n suficientemente grande, $|f(n)| \leq |C\ g(n)|$.

Teorema 2. *El algoritmo simple transforma un código Gray binario de longitud n en otro ternario de longitud n en $O(n \times 3^n)$ operaciones.*

Sea n la longitud del código Gray, $b = 2^n$, $t = 3^n$ y $p = 3^{n-1}$ (pétalo de mayor longitud), el análisis de la complejidad se detalla a continuación:

1. **Generación de pétalos**: En este paso generamos los n pétalos donde cada uno tiene una parte binaria y una parte ternaria. La complejidad de este paso es:

   $\sum_{i=0}^{n-1}$ código Gray ternario de longitud i + código Gray binario de longitud $n-1-i$ + juntarlos.

   Generar el código Gray binario y ternario de longitud n tiene costo $O(b)$ y $O(t)$ respectivamente. Esto se debe a que es complejidad lineal en cuanto a la cantidad de palabras que se

quiere generar. La combinación de los códigos Gray consiste en emparejar cada código Gray binario con cada código Gray ternario, que termina siendo un producto de las complejidades.

Entonces, se traduce en:

$$\sum_{i=0}^{n-1} O(3^i) + O(2^{n-1-i}) + O(3^i \times 2^{n-1-i}) = O(n \times t).$$

La sumatoria se simplifica tomando el término dominante (en este caso, t cuando $i = n-1$), resultando en un término de orden $O(n \times t)$.

2. **Generación del superpétalo**: En cada iteración, un pétalo se inserta en res. El costo de la inserción de un pétalo P_k dentro de otro pétalo P_j es $O(|P_k|+|P_j|) = O(\text{máx}(|P_k|, |P_j|)) = O(p)$, que corresponde al recorrido de P_j para encontrar la posición y, luego, insertar las palabras de P_k porque sabemos que palabras utilizar para la inserción. Durante el for, cada iteración tiene un costo de $O(\text{máx}(|P_i|, |res|))$. Sabiendo que res puede tener todas las palabras de los pétalos se deduce que $|res| = t - b$. Entonces queda $O(\text{máx}(|P_i|, t-b)) = O(t)$. Sumando todas las iteraciones, el costo total es $O(n \times t)$.

3. **Inserción en la primera palabra**: Este paso tiene un costo de $O(t)$, que es la longitud de res, el cual debe ser insertado en el código Gray binario original.

Todos estos costos son teniendo en cuenta que copiamos las palabras en las inserciones. La complejidad final del código es $O(n \times t) = O(n \times 3^n)$ que es el costo de generar los pétalos y el superpétalo.

4. Algoritmo de flujo máximo

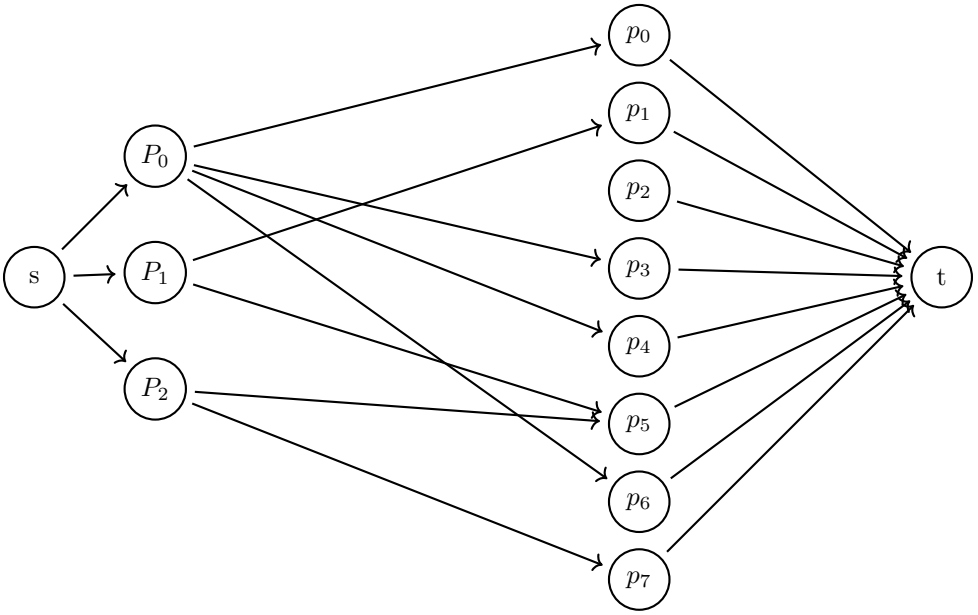
En la solución dada por el Algoritmo Simple todas las palabras que no estaban en el código Gray binario quedan juntas. ¿Qué pasa si queremos que estén más separadas o distribuidas? Para esto, podemos plantear el problema como un problema de flujo máximo. La idea es que cada pétalo vaya a una posición distinta del código Gray binario. Entonces, contamos con los siguientes nodos:

- S: fuente.
- P_i: los n pétalos.
- p_i: las posibles 2^n posiciones para ingresar un pétalo. Por ejemplo, p_0 es la posición entre la palabra 0 y 1.
- T: sumidero.

Las aristas son (todas con capacidad 1):

- $S \to P_i \ \forall i$: la fuente se conecta a todos los pétalos.
- $P_i \to p_j$ si P_i puede ser insertado en la posición p_j respetando las propiedades de código Gray.
- $p_i \to T \ \forall i$: todas las posiciones se conectan al sumidero.

Un ejemplo de un grafo en $n = 3$ se podría ver así:



Si ejecutamos el algoritmo de flujo máximo sobre este grafo obtenemos la mayor cantidad de pétalos que pueden ser insertados en distintas posiciones. La pregunta que nos surgió es si siempre se van a poder insertar todos los pétalos en posiciones distintas. Esto lo pudimos rechazar observacionalmente ya que, en $n = 4$, encontramos 8 contraejemplos donde no se pudo insertar P_3 en ninguna posición. Para estos casos podemos insertar estos pétalos dentro de otro pétalo como lo hicimos para formar el superpétalo anteriormente.

Una vez ejecutado el algoritmo de flujo, si el valor de flujo máximo F es n significa que todos los pétalos se pudieron insertar en posiciones diferentes del código Gray binario. Si $F < n$ se pudieron insertar F pétalos en distintas posiciones, los $n - F$ que no se pudieron, se insertan dentro de otro pétalo. Notar que si $F = 1$ es equivalente al caso del algoritmo simple visto antes.

Algoritmo 2 (Algoritmo de flujo máximo simple)**.**
Sea B un código Gray binario de longitud n.
Construir los pétalos $P_0, P_1, \dots P_{n-1}$. Obtener en que posiciones se puede insertar cada pétalo.
Ejecutar algoritmo de flujo máximo.
Insertar pétalos según el flujo máximo.
Insertar pétalos que quedaron sin insertar dentro de otros pétalos.
El output es un código Gray ternario de longitud n que tiene como subsecuencia al código Gray binario B.

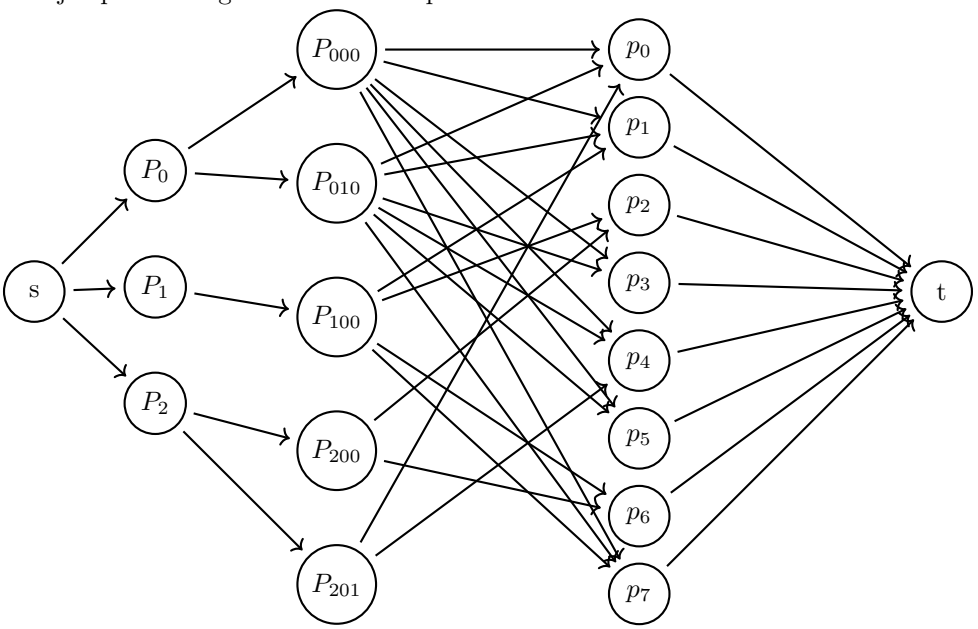
Si queremos intentar separar más los pétalos, y no insertar uno dentro de otro, podemos hacer un grafo más complejo donde tenemos en cuenta los shifts de los pétalos. Antes de generar el grafo, hacemos todos los shifts posibles de cada pétalo (se puede shiftear la parte binaria, la ternaria y ambas). Con eso en mente, le agregamos los siguientes nodos en el nuevo grafo:

- P_{ijk}: representa el pétalo P_i con j shifts en la parte binaria y k shifts en la parte ternaria

Las aristas son (todas con capacidad 1):

- $S \to P_i \ \forall i$: la fuente se conecta a todos los pétalos.
- $P_i \to P_{ijk}$ si tienen el mismo i
- $P_{ijk} \to p_j$ si P_{ijk} puede ser insertado en la posición p_j respetando las propiedades de código Gray.
- $p_i \to T \ \forall i$: todas las posiciones se conectan al sumidero.

Un ejemplo de un grafo en $n = 3$ se podría ver así:



Con este nuevo modelo, analizamos el resultado con todos los códigos Gray binarios de $n = 2, 3, 4$ y con los que generamos de $n = 5$ y en ningún caso fue necesario insertar un pétalo dentro de otro.

Algoritmo 3 (Algoritmo de flujo máximo con shifts)**.**
Sea B un código Gray binario de longitud n.
Construir los pétalos $P_0, P_1, \dots P_{n-1}$. Generar todos los posibles shifts de cada pétalo. Obtener en que posiciones se puede insertar cada shift de cada pétalo. Ejecutar algoritmo de flujo máximo. Insertar pétalos según el flujo máximo. Insertar pétalos que quedaron sin insertar dentro de otros pétalos.

El output es un código Gray ternario de longitud n que tiene como subsecuencia al código Gray binario B.

Teorema 3. *Dado un código Gray binario B de longitud n, el algoritmo de flujo máximo lo transforma en un código Gray ternario de longitud n en $O(n^2 * 2^{3n}) = O(n^2 * 8^n)$ operaciones.*

La complejidad temporal de ambos algoritmos, Algoritmos 2 y Algoritmo 3, es la máxima de las complejidades de armar el grafo, ejecutar el algoritmo de flujo máximo y, en el peor caso, agregar los pétalos que quedaron afuera.

En el breve análisis que sigue vamos a llamar $b = 2^n$, $t = 3^n$, $p = 3^{n-1}$ al pétalo de mayor longitud, $S_i = (n - i - 1) * i$ a la cantidad de posibles shifts que tiene el pétalo P_i y S al mayor de esos S_i.

Armar el grafo, en el primer modelo, implica construir los pétalos ($O(n \times t)$, explicado arriba) y ver en que posición se puede insertar ($O(t \times b)$ que es por cada palabra de los pétalos revisar si puede ubicarse en cada posición). En el segundo modelo es similar solo que tenemos que agregar los shifts de los pétalos, donde se agregan $n \times S$ (las distintas variantes de cada pétalo) y los shifts en cada pétalo pueden costar hasta $O(n \times p)$.

Ejecutar el algoritmo de flujo máximo puede ser $O(V \times E^2)$, con V siendo la cantidad de vértices y E las aristas si se usa Edmonds-Karp. En el primer modelo V=$2 + n + b$ y $E < n + (n * b) + b$ y en el segundo $V = 2 + n + (n * S) + b$ y $E < n + (n * S) + (n * S * b) + b$.

En caso de tener que agregar pétalos que quedaron afuera, se agrega $O(p)$ por cada uno.

Por lo tanto tenemos estos costos:

Etapa	**Modelo 1**	**Modelo 2**
Generar los pétalos	$O(n \times t)$	$O(n \times t + n \times S \times n \times p)$
Posicionar los pétalos	$O(t \times b)$	$O(n \times S \times p \times b)$
Ejecutar algoritmo de flujo máximo	$O(V \times E^2)$	$O(V \times E^2)$
Número de vértices (V)	$O(n + b)$	$O(n + (n \times S) + b)$
Número de aristas (E)	$O(n + (n \times b) + b)$	$O(n + (n \times S) + (n \times S \times b) + b)$
Agregar pétalos que quedaron fuera	$O(p \times n)$	$O(p \times n)$

Cuadro 11: Comparación de complejidades entre Modelo 1 y Modelo 2

Cabe destacar que en el trabajo [3] también han usado un algoritmo de flujo máximo para transformar secuencias de Bruijn en un alfabeto a secuencias de Bruijn en un alfabeto mayor.

5. Ejemplos

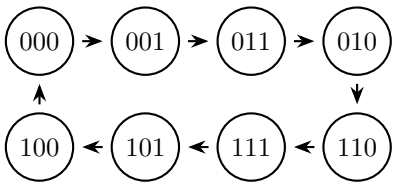
En esta sección, se presentarán ejemplos completos que resumen y aplican los conceptos desarrollados a lo largo de esta tesis. El código Gray binario a transformar es:

0	0	0	0
0	1	0	0
1	1	0	0
1	0	0	0
1	0	1	0
1	1	1	0
1	1	1	1
1	0	1	1
1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	1
0	0	1	1
0	1	1	1
0	1	1	0
0	0	1	0

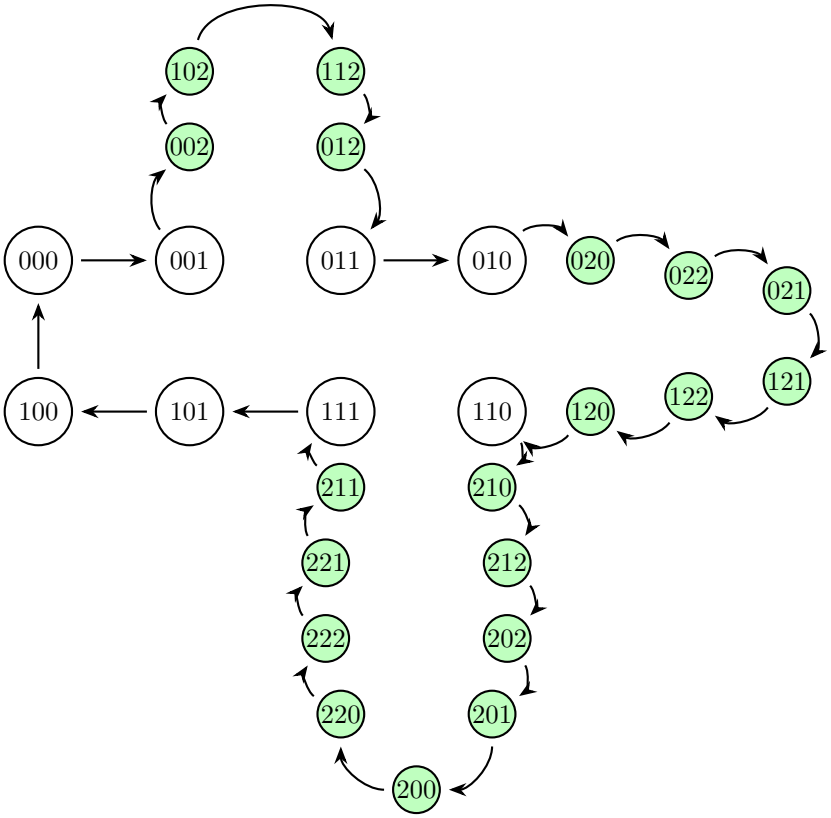
Cuadro 12: Código Gray Binario de ejemplo

5.1. Ejemplo gráfico

Antes de avanzar con los ejemplos de los modelos presentados, ilustraremos gráficamente la idea que deseamos desarrollar. Por ejemplo, consideremos el código Gray binario reflexivo de longitud $n = 3$:

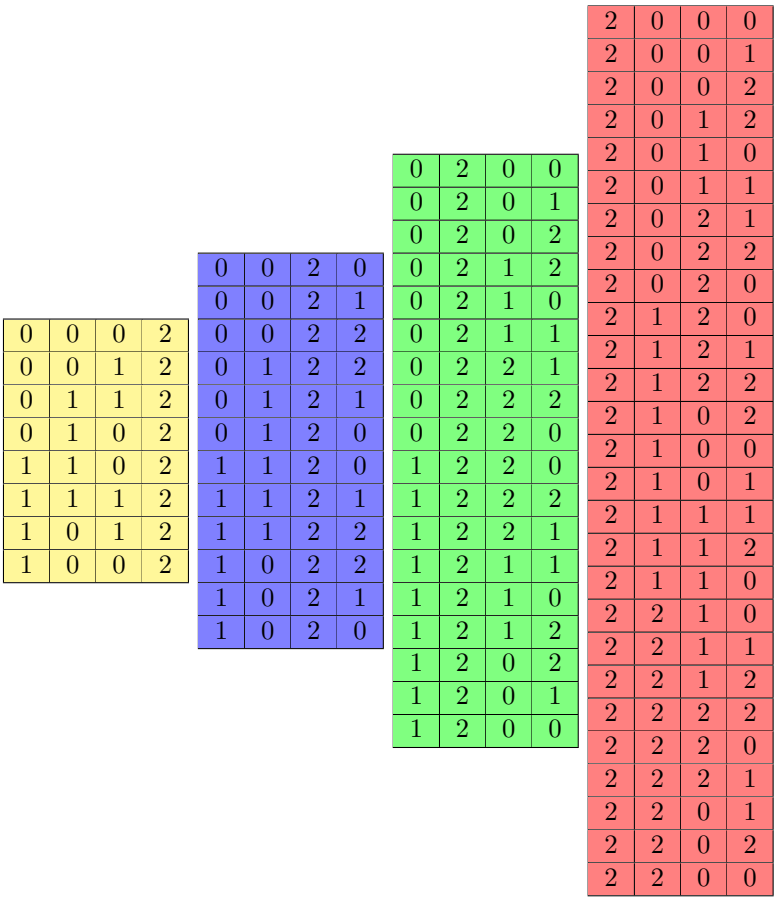


Mostraremos como formar un código Gray ternario mediante la inserción de los pétalos en el código Gray binario.



5.2. Ejemplo con el algoritmo simple

El primer paso es formar los pétalos de $n = 4$ juntando códigos Gray binarios y ternarios de menor longitud:

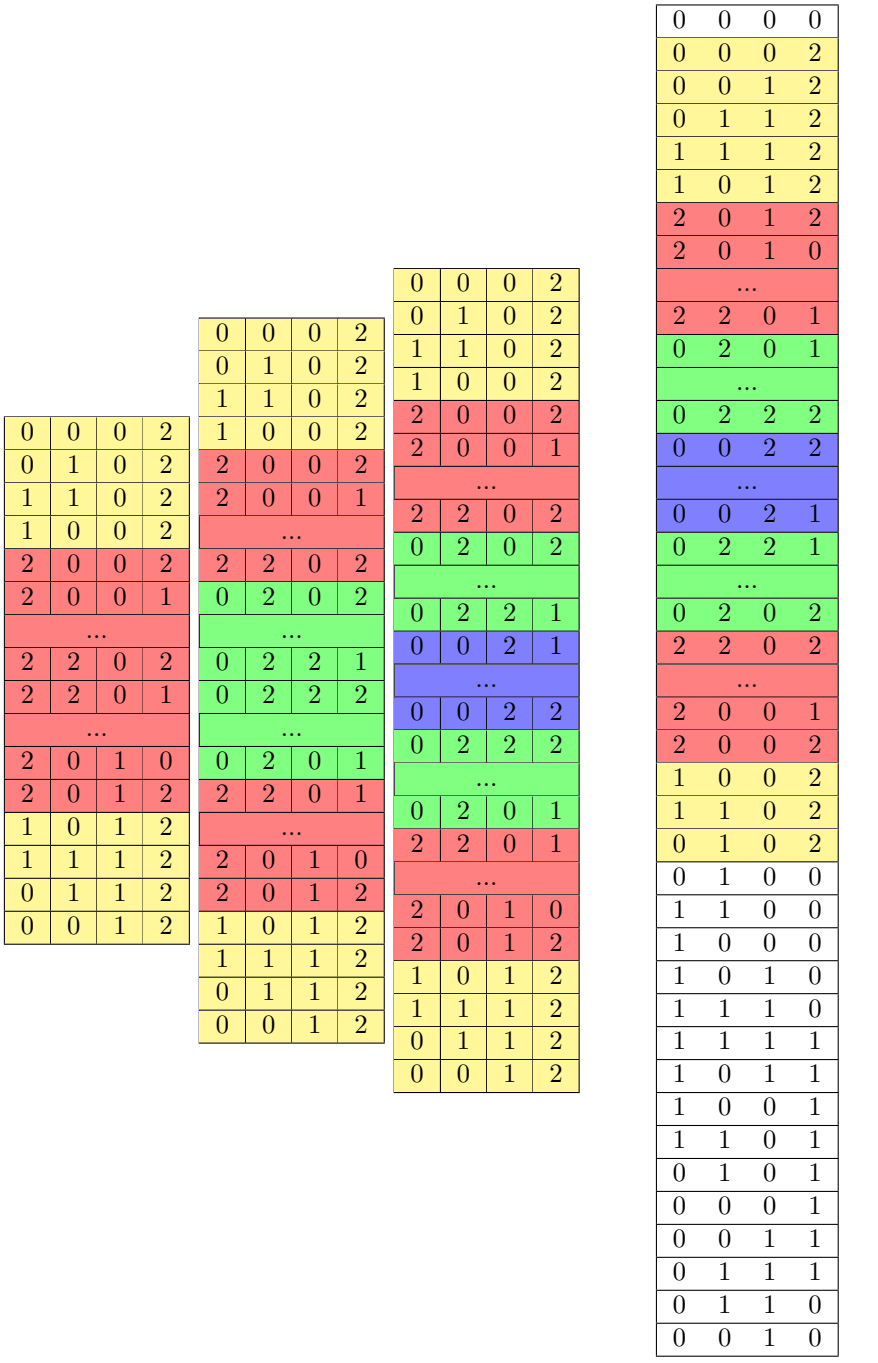


Cuadro 13: $P0$, P_1, P_2 y P_3

Luego, shifteamos el P_0 para poder insertarlo entre 0000 y 0100:

0	0	0	2
0	1	0	2
1	1	0	2
1	0	0	2
1	0	1	2
1	1	1	2
0	1	1	2
0	0	1	2

A continuación, vamos a insertar cada pétalo restante dentro del siguiente para formar el superpétalo y luego insertar el superpétalo en el código Gray binario.



Formación del superpétalo Código Gray ternario

5.3. Ejemplo con el algoritmo de flujo máximo

El primer paso es el mismo que en el ejemplo anterior, formar los pétalos:

0	0	0	2
0	0	1	2
0	1	1	2
0	1	0	2
1	1	0	2
1	1	1	2
1	0	1	2
1	0	0	2

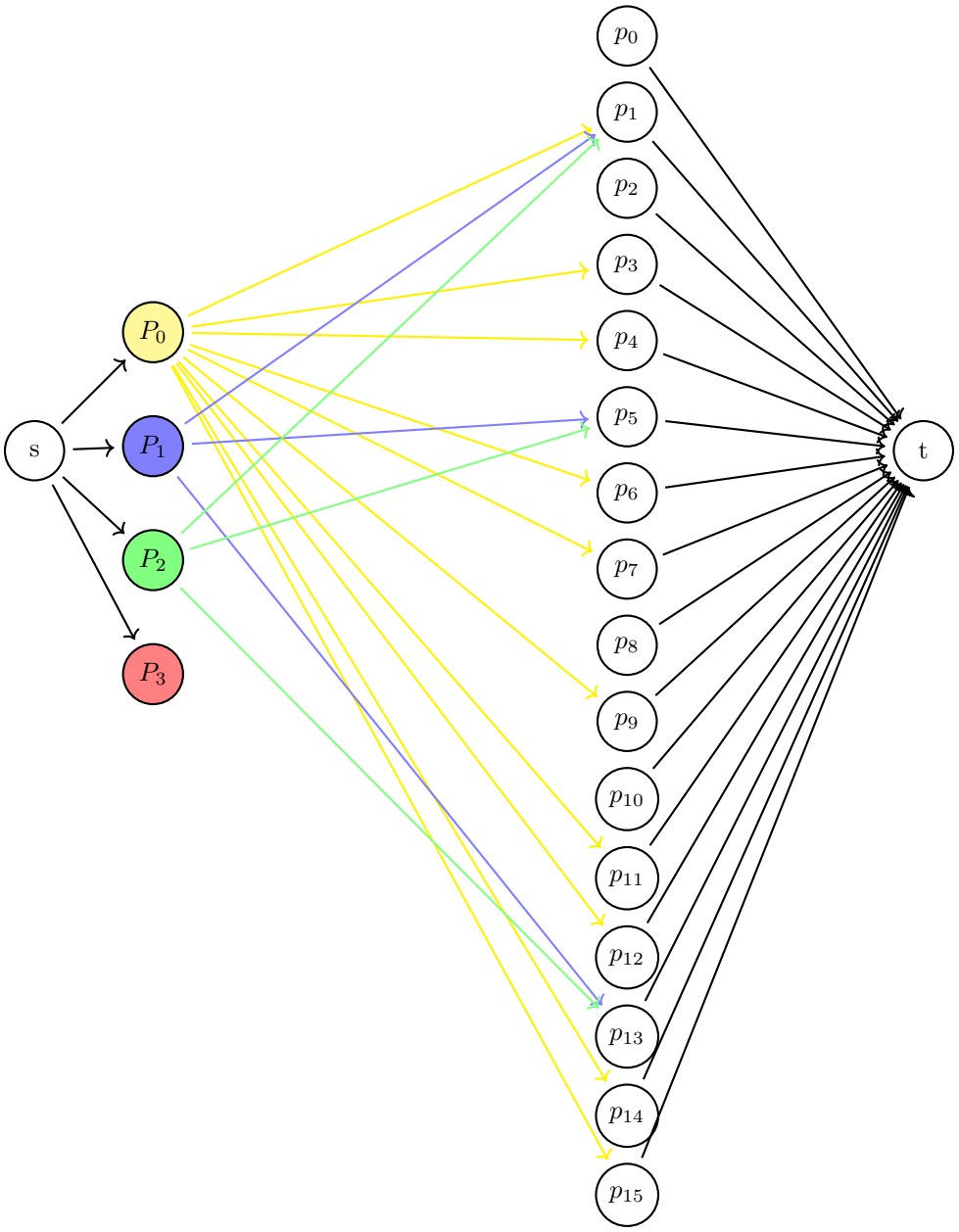
0	0	2	0
0	0	2	1
0	0	2	2
0	1	2	2
0	1	2	1
0	1	2	0
1	1	2	0
1	1	2	1
1	1	2	2
1	0	2	2
1	0	2	1
1	0	2	0

0	2	0	0
0	2	0	1
0	2	0	2
0	2	1	2
0	2	1	0
0	2	1	1
0	2	2	1
0	2	2	2
0	2	2	0
1	2	2	0
1	2	2	2
1	2	2	1
1	2	1	1
1	2	1	0
1	2	1	2
1	2	0	2
1	2	0	1
1	2	0	0

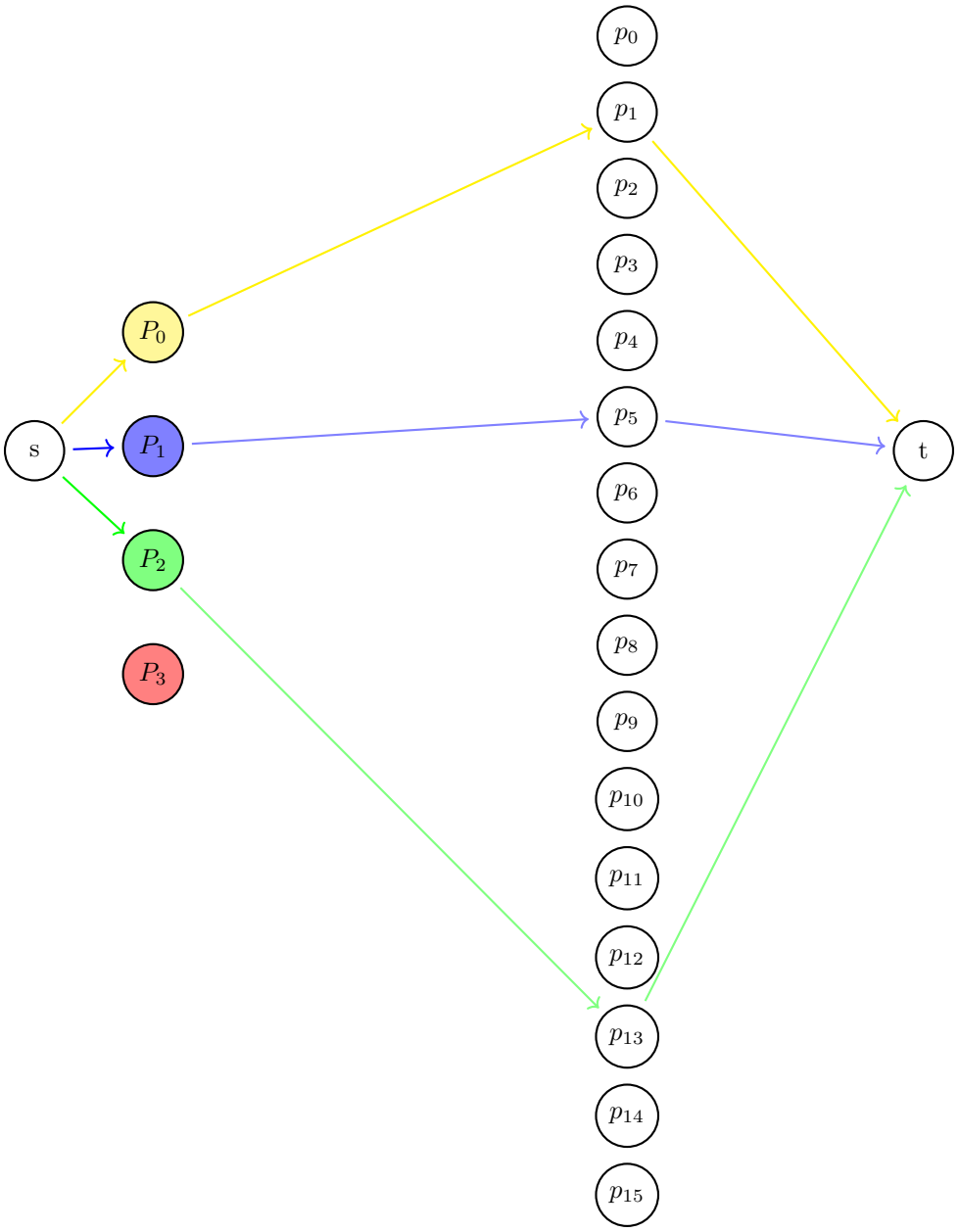
2	0	0	0
2	0	0	1
2	0	0	2
2	0	1	2
2	0	1	0
2	0	1	1
2	0	2	1
2	0	2	2
2	0	2	0
2	1	2	0
2	1	2	1
2	1	2	2
2	1	0	2
2	1	0	0
2	1	0	1
2	1	1	1
2	1	1	2
2	1	1	0
2	2	1	0
2	2	1	1
2	2	1	2
2	2	2	2
2	2	2	0
2	2	2	1
2	2	0	1
2	2	0	2
2	2	0	0

Cuadro 14: $P0$, P_1, P_2 y P_3

Luego, generamos el grafo de flujo máximo:



Si ejecutamos el algoritmo de flujo máximo podemos obtener un resultado como este:



Esto significa que podemos insertar P_0, P_1 y P_2 en posiciones distintas y P_3 lo insertamos dentro de P_0. Esto gráficamente se ve así:

0	0	0	0
0	1	0	0
0	1	0	2
0	1	1	2
...			
0	1	1	2
1	1	1	2
1	1	0	0
1	0	0	0
1	0	1	0
1	1	1	0
1	1	2	0
0	1	2	0
...			
1	1	2	2
1	1	2	1
1	1	1	1
1	0	1	1
1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	1
0	0	1	1
0	1	1	1
0	2	1	1
0	2	2	1
...			
0	2	1	2
0	2	1	0
0	1	1	0
0	0	1	0

Pétalos insertados con el algoritmo de flujo

0	0	0	0
0	1	0	0
0	1	0	2
0	1	1	2
...			
1	0	0	2
2	0	0	2
...			
2	0	1	2
1	0	1	2
...			
0	1	1	2
1	1	1	2
1	1	0	0
1	0	0	0
1	0	1	0
1	1	1	0
1	1	2	0
0	1	2	0
...			
1	1	2	2
1	1	2	1
1	1	1	1
1	0	1	1
1	0	0	1
1	1	0	1
0	1	0	1
0	0	0	1
0	0	1	1
0	1	1	1
0	2	1	1
0	2	2	1
...			
0	2	1	2
0	2	1	0
0	1	1	0
0	0	1	0

Código Gray ternario final
luego de insertar los pétalos faltantes

6. Código Gray en b símbolos a código Gray en b+1 símbolos

El algoritmo simple que transforma un código Gray binario a uno ternario que tiene al binario como subsecuencia se puede generalizar a alfabetos arbitrarios. El algoritmo usa la técnica del superpétalo, que permite insertar de manera ordenada las palabras adicionales necesarias para la transición entre alfabetos.

El algoritmo de flujo máximo también se puede generalizar para alfabetos arbitrarios.

6.1. Construcción código Gray en b símbolos

La construcción de un código Gray usando el alfabeto $A = \{0, 1, ..., b-1\}$ es similar a la que vimos en la sección Código Gray ternario. Un código de $n = 1$ es cualquier permutación de los símbolos de A. Luego para n mayores tenemos que hacer un patrón circular como en "Patrón circular que repite 3 veces cada símbolo" solo que se repite b veces cada símbolo. Por ejemplo, para $b = 4$, podría ser:

0
1
2
3
3
0
1
2
2
0
3
1
1
2
3
0

Cuadro 15: Patrón circular que repite b veces cada símbolo

Lo importante de este patrón es que si dividimos el patrón en bloques de b símbolos tenemos una permutación distinta de A. Además cada símbolo aparece una vez al inicio de un bloque y una vez al final de un bloque. Este patrón siempre existe, lo único que tiene que cumplir es que el símbolo final de un bloque y el inicio del siguiente sean el mismo para poder repetirlo y que cumpla la propiedad de circularidad. Nuestra forma de generar este patrón fue hacer que el símbolo final de cada bloque vaya de $b-1$ a 0, que el símbolo inicial sea el final del bloque anterior y que los del medio vayan en orden creciente con los símbolos faltantes.

Luego, una vez tenido el patrón, la idea es parecida a lo que vimos en $b = 3$:

- Para las columnas de 0 a $n-1$: repetir b veces.
- Para la columna n: agregar los $b-1$ bloques faltantes con estiraciones.
- Para la columna $n+1$: agregar el primer bloque con estiraciones.

Por ejemplo, supongamos que tenemos un código Gray de base 4 y $n = 2$. Si lo queremos pasar a uno de $n = 3$, la primer columna la repetimos 4 veces, la segunda agregamos los 3 bloques que faltan y en la nueva tercera columna agregamos el primer bloque:

0	0
0	1
0	2
0	3
1	3
1	0
1	1
1	2
2	2
2	0
2	3
2	1
3	1
3	2
3	3
3	0

0	0	0
0	0	1
0	0	2
0	0	3
0	1	3
0	1	0
0	1	1
0	1	2
0	2	2
0	2	0
0	2	3
0	2	1
0	3	1
0	3	2
0	3	3
0	3	0
1	3	0
1	3	1
1	3	2
1	3	3
1	0	3
1	0	0
1	0	1
1	0	2
1	1	2
1	1	0
1	1	3
1	1	1
1	2	1
1	2	2
1	2	3
1	2	0
2	2	0
	⋮	
2	1	0
3	1	0
	⋮	
3	0	0

Código Gray de base 4 y $n = 2$ Código Gray de base 4 y $n = 3$

Llamaremos $Z(b, n)$ al código generado.

6.2. Definiciones y notaciones

Antes de continuar damos unas breves definición que nos va a facilitar a la hora de explicar la existencia del superpétalo.

- **Código espejado**: código que se le aplica la reflexión. Por ejemplo, si tenemos

0
1
2

y lo espejamos 3 veces queda:

0
1
2
2
1
0
0
1
2

Notar que si reflejamos una cantidad par de veces queda circular.

- f (from): $b-1$. Último símbolo de la base origen.
- p (pivot): símbolo que hace la distinción entre la primer palabra (0*) y la segunda en un código Gray. Por ejemplo, si la segunda palabra es 000500, el pivot es 5. En el caso de binario que veníamos analizando no hacía falta este término porque siempre era 1.
- l (last): en la sección siguiente se explicará por qué este valor pero este símbolo representa el que va antes que b en el patrón explicado en Construcción código Gray en b símbolos para un código base $b+1$ o el último símbolo en el patrón en un código Gray de base b.

$$l = \begin{cases} f & \text{si } f \neq p \\ f-1 & \text{si } f = p \end{cases}$$

6.3. Nuevos pétalos

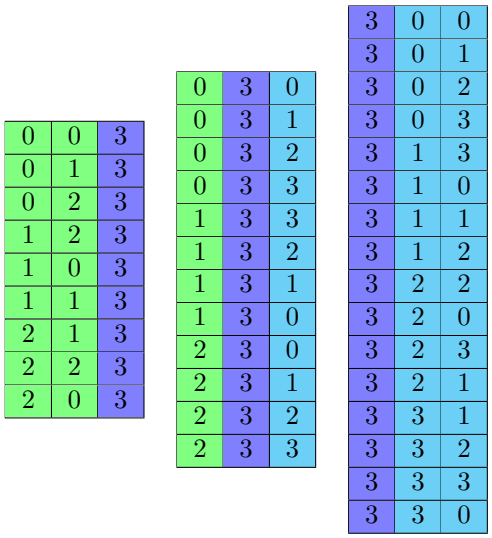
Debemos definir los pétalos $P_0, \dots P_{n-1}$. El P_i es la lista de palabras de n símbolos formadas por $Z(b, n-i-1)$ a la izquierda, $Z(b+1, i)$ a la derecha y el símbolo b en el medio.

En caso de que b sea par, la parte de la izquierda del pétalo es el código $Z(b, n-i-1)$ con multiplicidad $(b+1)^i$ en cada palabra y la parte derecha es el código Gray $Z(b+1, i)$ espejado b^{n-i-1} veces.

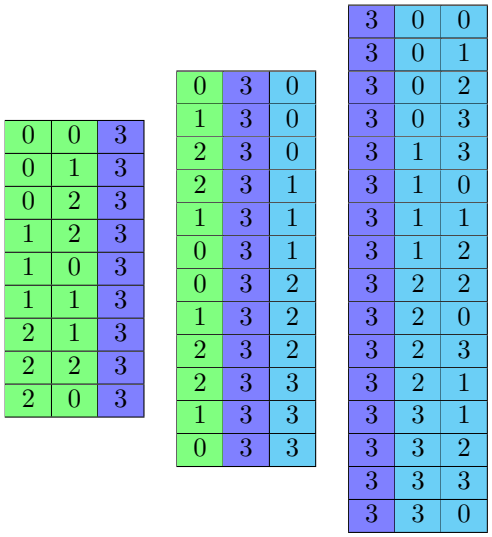
En caso de que b sea impar, la parte izquierda es el código $Z(b, n-i-1)$ espejado $(b+1)^i$ veces y la parte derecha es $Z(b+1, i)$ con multiplicidad b^{n-i-1} en cada palabra.

Notar que b^{n-i-1} con b par y $(b+1)^i$ con b impar son números pares. Esto lo hacemos ya que tenemos que tener un número par de reflexiones si queremos que nuestros pétalos sean circulares.

Imaginemos que reflejamos siempre la parte derecha como hicimos cuando pasamos de binario a ternario. Si hacemos eso se puede producir un error cuando la base de la parte izquierda es impar. Por ejemplo con $b+1=4$ y $n=3$, los pétalos podrían quedar así:



Se puede ver que P_1 no es circular debido a que se hicieron tres reflexiones de la parte derecha del pétalo. Este problema se soluciona cambiando que parte del pétalo se refleja. En el ejemplo anterior, reflejamos la parte derecha que tiene base $b = 4$ y quedaría así:

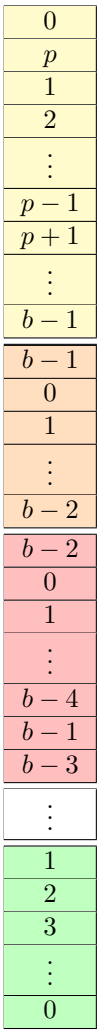


Cuadro 16: Pétalos de $b = 4$ y $n = 3$

Otra diferencia es que ahora el pivot no es constante. Por lo tanto, tenemos que hacer que, en el pétalo que se conecta al código Gray, la segunda palabra tenga p. Por ejemplo, en P_0 y con pivot p, este pétalo va a empezar con:

0	...	0	...	b
0	...	p	...	b
		$\vdots$		

Para hacer esto alcanza con hacer que el patrón en Construcción código Gray en b símbolos empiece con $0p$ en todos los pétalos. Por lo tanto, los patrones que generamos van a ser de la siguiente pinta (notar que los que van quedando como último del bloque siguen el orden de $b - 1$ a 0):



Cuadro 17: Patrón circular que repite b veces cada símbolo con inicio $0p$

A este código lo llamamos con la misma Z pero con un parámetro más, el pivot, $Z(b, n, p)$.

6.4. Algoritmo simple para alfabetos arbitrarios

Para demostrar la existencia de solución debemos demostrar que se puede formar un superpétalo y se puede insertar en el código Gray dado. Lo segundo se ve fácilmente porque armamos P_0 para que pueda ser insertado entre las primeras dos palabras. El único caso que falla esto es cuando la segunda palabra es 0^xp (el mismo problema que teníamos de binario a ternario) pero lo solucionamos insertando P_{n-1}. Para ver que podemos meter todos los pétalos dentro del siguiente computamos todas las formas de insertar un pétalo dentro del otro para distintos b, n y p y buscamos patrones que se repiten.

6.4.1. De alfabeto con cardinalidad par a impar

En esta sección mostramos los resultados de cuando pasamos de b par a $b+1$ impar. Recordar que se refleja la parte derecha de los pétalos.

- P_0 dentro de P_1

- $(0^x 0lb,\ 0^x plb)$ en P_0: el único caso que nos interesa analizar en P_0 es si la segunda palabra del código es $0^x p$, porque en cualquier otro caso insertamos P_0 en el código y no en P_1. P_0 es $Z(b, n-1, p)$ agregandole b al final de cada palabra. Sabemos que l es el último símbolo de ese código Gray y p el segundo, por lo tanto al reflejar por primera vez pasamos de $0l$ a pl
- $(0^x 0bb,\ 0^x pbb)$ en P_1: similar al anterior, sabemos que b es el último símbolo de $Z(b{+}1, 1, p)$ y p el segundo. Entonces, al llegar a b en la parte derecha, reflejamos y cambiamos la parte izquierda donde el siguiente es pasando de 0 a p.

- P_i dentro de P_{i+1}
  - $(0^x 0b0^y 1,\ 0^x 0b0^y 2)$ en P_i: viendo el lado derecho de b, $Z(b{+}1, i, p)$ sigue el orden de 0^n, $0^{n-1}p$, $0^{n-1}1$, $0^{n-1}2$, ... que cumple con este par dado. Solo cambia si $p = 2$ pero siguen siendo consecutivos, solo que el 2 previo al 1. El lado izquierdo es todo 0 en ambos.
  - $(0^x bb0^y 1,\ 0^x bb0^y 2)$ en P_{i+1}: viendo el lado derecho de b, $Z(b+1, i+1, p)$ siempre termina con $b0^x 0$, ese mismo bloque empieza con $b0^x 1$, $b0^x 2$. El lado izquierdo es todo 0 en ambos.

- P_{n-1} dentro de P_0
  - Definición previa: s (shifts): símbolo que representa 0 cuando no hay shifts y l si los hubo en P_0

$$s = \begin{cases} 0 & \text{si } p \text{ está en la anteúltima posición (no hay shifts)} \\ l & \text{si } p \text{ está en otra posición} \end{cases}$$

  - $(b0^x 0b,\ b0^x pb)$ en P_{n-1}: $0^x 0b$ es el último que empieza con $0^x 0$, entonces al reflejarse pasa a $0^x p$.
  - $(s0^x 0b,\ s0^x pb)$ en P_0: dividámoslo en distintos casos dependiendo cual es la segunda palabra:
    - $0^x p$: no es necesario analizarlo porque en este caso P_{n-1} se inserta en el código y no en P_0.
    - $0^x p0$: no se hacen shifts, entonces las dos primeras palabras de P_0 son $(00^x 0b, 00^x pb)$ que es el par con $s = 0$.
    - $0^i p0^{j+1}$: se hacen j shifts a izquierda. Verificar que esté el par de palabras $(l0^x 0b, l0^x pb)$ en P_0 shifteado es como verificar que esté $(l0^x 0b, l0^x pb)$ shifteado en P_0. Todos los shifts de este par tienen forma $(0^x 0l0^y b, 0^x pl0^y b)$ (recordar que b no se shiftea). Esto siempre pasa porque al llegar a $0^x l0^y$ se pasa de 0 a p el anterior símbolo a l.

6.4.2. De alfabeto con cardinalidad impar a par

En esta sección mostramos los resultados de cuando pasamos de b impar a $b+1$ par. Recordar que se refleja la parte izquierda de los pétalos.

- P_i dentro de P_{i+1}
  - Definición previa: o_s (one-two sufix): símbolo que representa con que sufijo de reflexión se juntan el 1 y 2 en el lado izquierdo del pétalo en bases impares.

$$o_s = \begin{cases} f-1 & \text{si } p \leq 2 \\ f-2 & \text{si } p > 2 \end{cases}$$

- $(0^x 1 o_s b 0^y,\ 0^x 2 o_s b 0^y)$ en P_i: viendo el lado izquierdo de b, el 1 y 2 en la segunda columna aparecen como consecutivo en la segunda o tercer reflexión dependiendo si p es un número mayor a ellos. Entonces queda $f-1$ o $f-2$ ya que, con el patrón que hicimos para un código base b, los últimos de cada bloque van de f a 0 ($\{f, f-1, ..., 0\}$).

  Hay un caso único especial con $b=3$ y $p=2$ que al no contar con tantos símbolos, los últimos de cada bloque son $\{2,1,0\}$ y no cumple que el segundo símbolo sea $f-1$. Solo en este caso hacemos que en vez de o_s se use un 2.
- $(0^x 1 b b 0^y,\ 0^x 2 b b 0^y)$ en P_{i+1}: Viendo el lado derecho, la palabra $b0^y$ es la última de todo $Z(b, y+1, p)$. Por lo tanto sabemos que estamos en la última reflexión del lado izquierdo. $Z(b+1, x, p)$ reflejado termina con 0^n, $0^{n-1}p$, $0^{n-1}1$, $0^{n-1}2$.

- P_{n-2} dentro de P_{n-1}
  - Definición previa: o_p (one-two prefix): símbolo que representa con que prefijo de reflexión se juntan el 1 y 2 en bases impares.

$$o_p = \begin{cases} 0 & \text{si } p \leq 2 \\ l & \text{si } p > 2 \end{cases}$$

  - $(o_p b 0^x 1,\ o_p b 0^x 2)$ en P_{n-2}: la parte izquierda es Z(b,1,p), por lo tanto va de 0 a l y de l a 0 haciendo reflexiones. Si p es 1 o 2 significa que el patrón junta al 1 y 2 en la segunda y tercera posición (0,1,2, ... o 0,2,1 ...). En este caso, en la segunda reflexión de la parte izquierda juntaríamos al 1 y 2 en la parte derecha, por eso empieza con $o_p = 0$. Si p es mayor a 2, 1 y 2 se juntan en la tercera reflexión, $o_p = l$.
  - $(bb0^x 1,\ bb0^x 2)$ en P_{n-1}: $Z(b, n-1, p)$ termina con $b0^{n-3}0$, ese mismo bloque empieza con $b0^{n-3}1$, $b0^{n-3}2$.

- P_{n-1} dentro de P_0: mismo análisis que en la sección anterior

Damos a continuación el algoritmo general.

Algoritmo 4 (Algoritmo simple Código Gray en b símbolos a Código Gray en $(b+1)$ símbolos)**.**
Sea B un código Gray de base b de longitud n.
Construir los pétalos $P_0, P_1, \ldots P_{n-1}$ con el pivot adecuado.
Si la segunda palabra de B no es $0^x p$ insertar P_0.
Insertar P_{n-1} en P_0.
Para $i = n-2, \ldots 1$, insertar P_i en P_{i+1}.
Si la segunda palabra de B es $0^x p$ insertar P_{n-1}. Para $i = n-2, \ldots 0$, insertar P_i en P_{i+1}.
Insertar P_0 en B. Insertar P_{n-1} en P_0. Para $i = n-2, \ldots 1$, insertar P_i en P_{i+1}.
El output es un código Gray de base $b+1$ de longitud n que tiene como subsecuencia a B.

7. Bibliografía

Referencias

[1] Verónica Becher. Insertion in constructed normal numbers. *Uniform Distribution Theory*, 17(1):55–76, 2022.

[2] Verónica Becher and Olivier Carton. Normal numbers and computer science. In V. Berthé and editors M. Rigo, editors, *Sequences, Groups, and Number Theory*, Trends in Mathematics Series, pages 233–269. Birkhäuser/Springer, 2018.

[3] Verónica Becher and Lucas Cortés. Extending de bruijn sequences to larger alphabets. *Information Processing Letters*, 168(106085), 2021.

[4] Girish S. Bhat and Carla Savage. Balanced Gray codes. *The Electronic Journal of Combinatorics*, 2(1), 1996.

[5] Frank Gray. Pulse code communication. U.S Patent No. 2632058, March 15 1953.

[6] Hackaday. Gray codes. `https://hackaday.io/project/164907-ternary-computing-menagerie/log/162168-gray-codes`, 2024. Ternary Computing Menagerie.

[7] OEIS Foundation Inc. Sequence a006069. `https://oeis.org/A006069`, 2024. Number of directed Hamiltonian cycles (or Gray codes) on n-cube with a marked starting node.

[8] Donald Knuth. Generating all n-tuples. In *The Art of Computer Programming. Volumen 4A: Enumeration and Backtracking. Prefascicle*. Addison Wesley, 2004.