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

El algoritmo de Huang

VERSION JURADO - Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Néstor Darío Ocles García

Buenos Aires, 29 de marzo de 2024

EL ALGORITMO DE HUANG

Resumen: Fijemos un alfabeto. Una secuencia de De Bruijn de orden n es una secuencia de símbolos del alfabeto que contiene todas las palabras de longitud n exactamente una vez. Estas secuencias fueron descubiertas y redescubiertas más de una vez a partir de finales de los años 1800. En este trabajo analizamos el algoritmo de Yuejiang Huang del año 1990 que produce secuencias de Bruijn en un alfabeto de dos símbolos. Es un algoritmo óptimo en tiempo y memoria, que arroja secuencias de Bruijn muy balanceadas, esto significa que en cada segmento de la secuencia los dos símbolos del alfabeto aparecen casi la misma cantidad de veces. Mostramos aquí de qué manera el algorimo de Huang construye una secuencia de Bruijn de orden n definiendo un camino Euleriano en el grafo de Bruijn de orden *n-1*, uniendo una clase particular de ciclos simples.

Palabras claves: Secuencias de Bruijn, grafos de Bruijn, ciclos eulerianos, Complemented Clycling Register.

Dedicatoria.

Índice general

1. INTRODUCCIÓN

Las secuencias de Bruijn deben su nombre al matemático Nicolaas Govert de Bruijn [3], aunque fueron descubiertas por varios otros también. Una interesante exposición de su historia puede encontrarse en [1].

Las secuencias de Bruijn resultan de suma utilidad tanto en algoritmos y estructuras de datos. Su construcción no es trivial y existen diversas formas de construirlas.

Las secuencias de Bruijn están en correspondencia uno a uno con los ciclos hamiltonianos en los llamados grafos de Bruijn. Dado que el grafo de Bruijn de palabras de longitud $k+1$ es el grafo de línea del grafo de Bruijn de palabras de longitud k, los ciclos hamiltonianos en el grafo de Bruijn de palabras de longitud k resultan ser ciclos eulerianos en el grafo de Bruijn de palabras de longitud k.

Construir una secuencia de Bruijn equivale a dar un ciclo euleriano en dichos grafos. El algoritmo de Huang [5] da un método muy valorado para construir secuencias de Bruijn. Por un lado, es muy eficiente: con palabras de longitud k utiliza $4k$ únidades de tiempo y requiere k bits de memoria para producir el siguiente símbolo de la secuencia.

Por otro lado, la secuencia de Bruijn producida por el algoritmo de Huang es muy balanceada. Una palabra es balanceada cuando para cada una de sus subpalabras la cantidad de 0s y la cantidad de 1s es muy parecida. Se sabe que para cada $k \geq 1$ la secuencia de Bruijn lexicogáficamente máxima es altamente no balanceada, y este desbalance fue cuantificado exactamente en [2]. Empíricamente se sabe que la secuencia de Bruijn generada por el algoritmo de Huang es muy balanceada [4], pero hasta ahora no hay una cuantificación exacta de cuán balanceada es. Ésta es la motivación principal para comprender con mayor profundidad el algoritmo de Huang.

Primero, comenzamos con la definición de las palabras de Bruijn y su representación en grafos de Bruijn. Luego introduciremos el concepto de ciclo CCR (Complemented Cycling Register) en el grafo de Bruijn. Finalmente, presentamos el algoritno de Huang en términos de un recorrido en un nuevo grafo, el grafo de los ciclos CCRs. De esta manera nuestra presentación del algoritmo de Huang da inmediatamente su correctitud.

2. SECUENCIAS DE BRUIJN Y GRAFOS DE BRUIJN

2.1. Notación

Sea el alfabeto $\{0, 1\}$ escribimos $\overline{1} = 0$, $\overline{0} = 1$ y si a es símbolo del alfabeto entonces $\overline{\overline{a}} = a$.

Una palabra es una secuencia finita de símbolos del alfabeto. Si Σ es el alfabeto, Σ^k es el conjunto de todas als palabras de longitud k. La longitud de una palabra S la denotamos con $|S|$. Numeramos sus posiciones a partir de la posición 1. Dada una palabra S consideraremos subpalabras de $S[i, .., j]$ formadas por los símbolos de S desde la posción i hasta la posición j. El símbolo en la posición i-ésima de una palabra es $S[i]$, que también podremos escribir S_i.

2.2. Secuencias de Bruijn y grafos de Bruijn

Definimos la secuencia de Bruijn de orden k como una palabra donde cada palabra de longitud k aparece exactamente una vez. Veamos un ejemplo a continuación, el total de palabras de longitud 3 es 8 y éstas son:

000, 001, 010, 011, 100, 101, 110, 111

Una secuencia de Bruijn de orden 3 es *0001011100*. Cada subpalabra aparece una única vez. Si en vez de considerar palabras consideramos palabras circulares o collares, no son necesarios los últimos dos símbolos ya que éstos son idénticos a los dos primeros, por ende quedaría así *00010111*.

000 10111	**00** 01011 **1**
0 **001** 0111	000 **101** 11
00 **010** 111	**0** 00101 **11**
0001 **011** 1	00010 **111**

Figura 2.1: Distintas palabras de longitud 3 en la palabra 00010111 vista como collar.

También podemos representar las secuencias de Bruijn de orden k como ciclos eulerianos en los llamados grafos de Bruijn. Un ciclo euleriano en un grafo es un ciclo que usa todas las aristas del grafo exactamene una vez, comienza en un nodo y regresa al mismo habiendo usado todas las aristas y sin haber pasado dos veces por la misma arista.

Un grafo de Bruijn es un grafo de orden *k-1* dirigido cuyos nodos son las palabras de logitud *k-1* y las aristas unen pares de nodos y se asocian con palabras de longitud k, de manera tal que los primeros *k-1* símbolos corresponden al nodo de partida y los últimos *k-1* símbolos corresponden al nodo de llegada. La Figura 2.2 muestra el grafos de Bruijn de orden *k-1*.

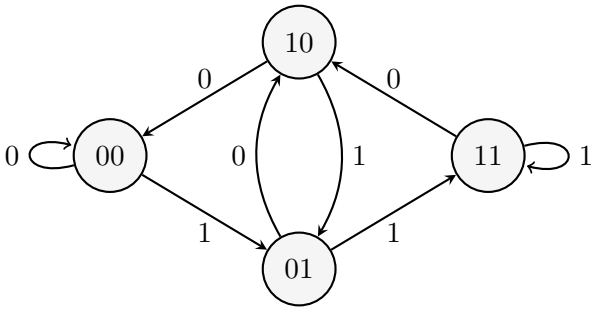


Figura 2.2: Grafo de Bruijn de orden 2.

Dada esta representacióon podemos pensar que cada arista es una palabra de k símbolos conectando 2 nodos de k-1 símbolos, tal como se puede observar en la Figura 2.3.

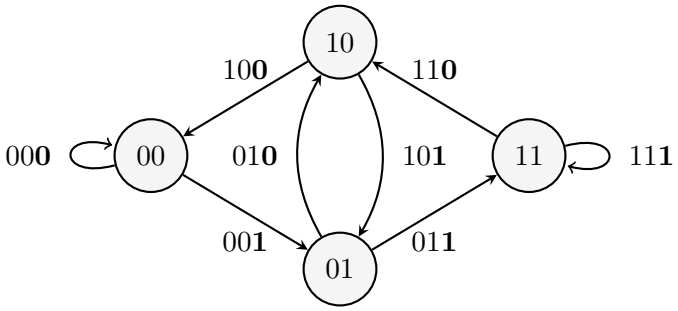


Figura 2.3: Grafo de Bruijn de orden 2 con aristas de nombre completo.

De la Figura 2.3 se desprende que toda palabra de k símbolos es la representación de una arista que conecta dos nodos en el grafo de Bruijn de orden k-1. En el ejemplo cada palabra de 3 símbolos es una arista conectando dos nodos de longitud k-1 de la siguiente manera:

$$v_1 \rightarrow v_2$$

$$arista = \overbrace{a_1 a_2}^{v_1} a_3 = a_1 \overbrace{a_2 a_3}^{v_2}$$

$$v_1 = a_1 a_2$$

$$v_2 = a_2 a_3$$

En el grafo de Bruijn de orden 2 la palabra *011* es la arista que conecta el nodo *01* con *11* tal como se aprecia en la Figura 2.3.

3. SOBRE CICLOS CCR (COMPLEMENTED CYCLING REGISTER)

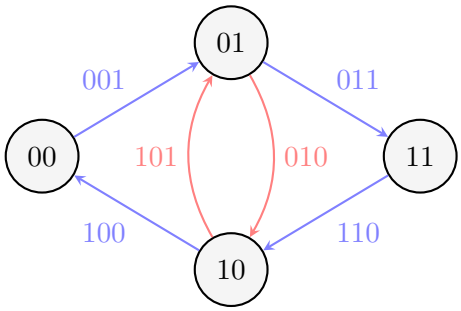


Figura 3.1: Grafo de Bruijn de orden 2 y sus dos CCRs coloreados en azul y rojo.

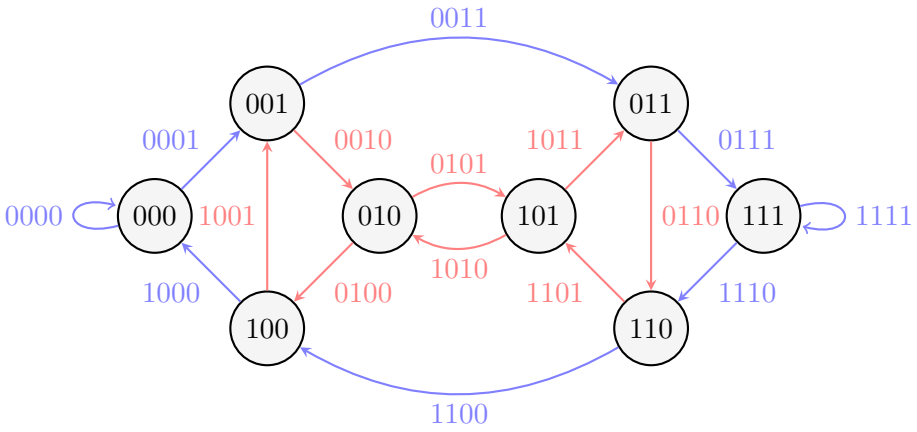


Figura 3.2: Grafo de Bruijn de orden 3 y sus dos CCRs coloreados en azul y rojo.

3.1. Extensión de una palabra

Sea una palabra $a_1a_2a_3...a_k$ de longitud k, la función de extensión $E : \Sigma^k \to \Sigma^{k+1}$ se define :

$$E(a_1a_2...a_k) = a_1a_2...a_k\overline{a_1}.$$

Por ejemplo, $E(010) = 010\overline{0} = 0101$.

3.2. Siguiente de una palabra

Utilizando la función extensión para cada k, definimos la función $siguiente : \Sigma^k \to \Sigma^k$,

$$siguiente(w) = E(w)[2, ..., k+1]$$

Si $w = a_1a_2..a_k$ entonces

$$siguiente(w) = a_1a_2...a_k\overline{a_1}[2, ..., k+1] = a_2...a_k\overline{a_1}.$$

Para referirnos a múltiples aplicaciones sucesivas escribimos

$$siguiente^j(w) = \underbrace{siguiente(siguiente(siguiente(siguiente...}_{\text{j aplicaciones sucesivas}}(w)))$$

$Siguiente_i$	Descomposición	Resultado
$siguiente^1(00)$	$E(00)[2,..,3]$	01
$siguiente^2(00)$	$siguiente(siguiente(00)) = siguiente(01)$	11
$siguiente^3(00)$	$siguiente(siguiente(siguiente(00))) = siguiente(11)$	10
$siguiente^4(00)$	$siguiente(siguiente(siguiente(siguiente(00)))) = siguiente(10)$	00
$siguiente^5(00)$	$siguiente(siguiente^4(00)) = siguiente(00) = siguiente^1(00)$	01

Tabla 3.1: Ejemplo de llamadas consecutivas de la palabra 00.

3.3. Ciclos CCR

Recordemos que asumimos el alfabeto binario. Un ciclo CCR de orden k es un conjunto no vacío de palabras de longitud k tal que si $w \in C$ entonces

$$C = \{siguiente^j(w) : j \geq 1\}$$

Por ejemplo, $\{000, 001, 011, 111, 110, 100\}$ es un ciclo CCR de orden 3. Y $\{0101, 1011, 0110, 1101, 1010, 0100, 1001, 0010\}$ es un ciclo CCR de orden 4.

Cada palabra del ciclo CCR corresponde con la etiqueta de una arista en el grafo de Bruijn de orden *k-1* y estos son ejes sucesivos y forman un ciclo.

Sea w una palabra arbitraria de longitud k

$$c_1 = siguiente(w) = E(w)[2..k+1] = a_1a_2...a_k$$

$$c_2 = siguiente(c_1) = E(c_1)[2..k+1] = a_2a_3...a_k\overline{a_1}$$

$$c_3 = siguiente(c_2) = E(c_2)[2..k+1] = a_3...a_k\overline{a_1a_2}$$

$$c_4 = siguiente(c_3) = E(c_3)[2..k+1] = ...a_k\overline{a_1a_2a_3}$$

$$...$$

$$c_k = siguiente(c_{k-1}) = \overline{a_1a_2a_3}...\overline{a_k}$$

$$c_{k+1} = siguiente(c_k) = \overline{a_2a_3}...\overline{a_k}a_1$$

$$c_{k+2} = siguiente(c_{k+1}) = \overline{a_3}...\overline{a_k}a_1a_2$$

$$...$$

$$c_{k+k} = siguiente(c_{k+k-1}) = \overline{a_k}a_1a_2...a_{k-1}$$

$$c_{k+k+1} = c_1 = siguiente(c_{k+k}) = a_1a_2...a_{k-1}a_k$$

Cada ciclo CCR contiene como máximo $2k$ palabras donde cada una de estas palabras constituye las aristas en el grafo de Bruijn de orden *k-1*. Asimismo, el orden en

cómo recorrer ese ciclo está dado por la función siguiente. El ciclo CCR es el conjunto $\{c_1, c_2, .., c_{2k-1}, c_{2k}\}$.

Asumiendo que el alfabeto es $\{0, 1\}$, para longitud 3 hay exactamente dos CCRs

palabra	E(palabra)	Siguiente palabra en el CCR
101	101**0**	010
010	010**1**	101

Tabla 3.2: CCR de la palabra 101

palabra	E(palabra)	Siguiente palabra en el CCR
000	000**1**	001
001	001**1**	011
011	011**1**	111
111	111**0**	110
110	110**0**	100
100	100**0**	000 (se vuelve a empezar)

Tabla 3.3: CCR de la palabra 000.

En general, para toda longitud k, los ciclos CCR determinan una partición del conjunto de palabras de longitud k.

4. ALGORITMO DE HUANG

4.1. *r-diferencia* de un CCR

Entendemos la *r-diferencia* de una palabra de longitud k como la cantidad de pares de símbolos consecutivos de la palabra extendida que son distintos.

Definimos la función *r-diferencia:*$\{0,1\}^* \to \mathbb{N}$ de la siguiente manera:

$$\textit{r-diferencia}(a_1a_2a_3...a_k) = \sum_{i=1}^{k} E(a_1a_2a_3...a_k)[i] \neq E(a_1a_2a_3...a_k)[i+1]$$

Ejemplos:

$$\textit{r-diferencia}(000) = \sum_{i=1}^{n} E(000)[i] \neq E(000)[i{+}1] = \sum_{i=1}^{n} 0001[i] \neq 0001[i{+}1] = 1$$

$$\textit{r-diferencia}(000) = 1$$

La palabra 000 tiene una r-diferencia de 1.

$$\textit{r-diferencia}(010) = \sum_{i=1}^{n} E(010)[i] \neq E(010)[i{+}1] = \sum_{i=1}^{n} 0101[i] \neq 0101[i{+}1] = 3$$

$$\textit{r-diferencia}(010) = 3$$

La palabra 010 tiene una *r-diferencia* de 3.

La palabra *010* tiene una *r-diferencia* igual a 3 y por notación podemos decir que la palabra *010* es una palabra *3-diferencia*. Notar que reemplazamos **r** con el valor concreto asociado a esa palabra. Esto será útil en algunas demostraciones como en el Lema 3.

4.2. Rotaciones de palabras de orden k

Si consideramos cada palabra de longitud k como un collar podemos rotar la palabra a izquierda o a derecha formando nuevas palabras que representan aristas del grafo de Bruijn de orden *k-1*. Definimos las rotaciones como:

$$\text{rotacion-izq}(a_1a_2...a_{k-1}a_k) = a_2...a_{k-1}a_ka_1$$

$$\text{rotacion-der}(a_1a_2...a_{k-1}a_k) = a_ka_1a_2...a_{k-1}$$

4.3. Construyendo salidas de un CCR

Sea w una palabra de longitud k que representa una arista en el grafo de Bruijn de orden *k-1* conectando los nodos $v_1 \to v_2$

$$w = \overbrace{a_1 a_2 ... a_{k-1}}^{v_1} a_k = a_1 \overbrace{a_2 ... a_{k-1} a_k}^{v_2}$$
$$v_1 = a_1 a_2 ... a_{k-1}$$
$$v_2 = a_2 ... a_{k-1} a_k$$

Como vimos anteriormente, w pertenece a un ciclo CCR y la siguiente arista está determinada por la función siguiente. Sea

$$w_s = siguiente(w)$$
$$w_s = \overbrace{a_2 ... a_{k-1} a_k}^{v_{s1}} \overline{a_1} = a_2 \overbrace{...a_{k-1} a_k \overline{a_1}}^{v_{s2}}$$

Observamos que al rotar w a izquierda la palabra resultante representa una arista del grafo de Bruijn de orden *k-1* que comparte el mismo nodo de salida que *siguiente(w)* pero potencialmente distinto nodo de llegada

$$\text{rotacion-izq}(w) = \overbrace{a_2 a_3 ... a_{k-1} a_k}^{v_{s1}} a_1 = a_2 \overbrace{a_3 ... a_{k-1} a_k a_1}^{v_{r2}}$$
$$v_{s2} \neq v_{r2}$$

La rotación a izquierda y siguiente de w crea dos palabras distintas que representan aristas en el grafo de Bruijn de orden *k-1*. Estas dos palabras comparten el nodo de salida pero poseen distintos nodos de llegada. La rotación a izquierda puede generarnos palabras dentro del mismo CCR que w o palabras pertenecientes a otros CCR.

Para ejemplificar tomaremos los valores de las Tablas 3.2 y 3.3, consideramos dos palabras del CCR formado desde *000*.

$$CCR(000) = \{000, 001, 011, 111, 110, 100\}$$
$$w_1 = 000$$
$$\text{rotacion-izq}(w_1) = 000$$
$$\text{rotacion-izq}(w_1) \in CCR(000)$$

En el caso de tomar *110*

$$CCR(110) = \{000, 001, 011, 111, 110, 100\}$$
$$w_2 = 110$$
$$\text{rotacion-izq}(w_2) = 101$$
$$\text{rotacion-izq}(w_2) \notin CCR(110)$$

Como se puede ver en las tablas 3.2 y 3.3 comprobamos que rotacion-izq(w_2) pertenece a otro CCR que w_2. Cuando esto sucede llamaremos a w_2 como arista de salida del CCR. Cabe aclarar que en un CCR puede haber múltiples aristas de salidas.

4.4. Construyendo entradas de CCRs

Como vimos anteriormente, al rotara izquierda las palabras generamos posibles aristas de salida del CCR actual a un posible CCR'. Si tomamos esa misma palabra del CCR' y la rotamos a derecha volvemos a generar una palabra del CCR original.

Si consideramos las rotaciones a izquierda como posibles salidas, de forma análoga podemos ver las rotaciones a derecha como posibles entradas al CCR actual.

4.5. Entrada distinguida de cada CCR

Los ciclos CCR forman una partición de nuestro grafo de Bruijn, al tener una forma únivoca de entrar y salir de cada ciclo nos garantizan una forma discreta y única de recorrer cada uno de los ciclos. Para tal fin definimos una arista en particular del CCR y le llamaremos entrada distinguida.

Las palabras dentro de un CCR tienen un orden lexicográfico que coincide con el orden de su representación decimal. Cuando hablamos de la palabra más grande en un CCR hablamos de la palabra lexicográfica más grande o la palabra de valor decimal más grande.

Teniendo en cuenta las rotaciones podemos calcular todas las posibles entradas a un CCR rotando a derecha cada palabra del CCR. De todas estas posibles entradas se va a elegir una que debe cumplir uno de los siguientes criterios:

- Si existe en el CCR actual al menos una entrada con *r-diferencia* menor al CCR actual, elegimos la entrada que llegue a la palabra lexicograficamente más grande del CCR actual (o su valor decimal más grande).
- Si no existe una entrada con *r-diferencia* menor, elegimos de las *r-diferencia* igual al CCR actual la palabra que llega a la palabra más grande desde un CCR distinto al actual.

4.6. El algoritmo de Huang en alfabeto {0,1}

El algoritmo de Huang trabaja sobre tres ideas principales: Define unos ciclos llamados CCR que forman una partición del grafo, demuestra que existe una forma de conectar dichos ciclos y, por último, define una forma únivoca de recorrer cada ciclo.

Construir una secuencia de Bruijn de orden k se resuelve formando un camino hamiltoneano en el grafo de Bruijn de orden *k-1*. Como vimos anteriormente en laSección 2.2 cada nodo es una palabra de *k-1* símbolos del cual salen dos aristas representando el símbolo faltante para formar una palabra de k símbolos que le dará el nombre a dicha arista. Al formar el ciclo hamiltoneano sobre este grafo iremos guardando el símbolo que cada arista representa y al finalizar el ciclo tendremos la secuencia de Bruijn de orden k que será una palabra de 2^k símbolos.

La secuencia de Bruijn de orden k se crea recorriendo aristas representando las palabras de k símbolos en el grafo de Bruijn de orden *k-1*. A dicha arista la llamaremos arista y será una de las entradas del algoritmo.

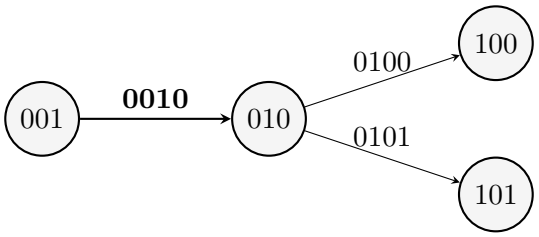
Un ciclo CCR es entonces una lista de aristas. Para construir la secuencia de Bruijn de orden k tenemos el siguiente algoritmo que recibe de argumento el k y una palabra de k símbolos que representa una arista del grafo de Bruijn de orden *k-1*. La construcción

del ciclo hamiltoniano empezará desde la arista que recibe como argumento y calculará 2^k pasos dentro del grafo.

Algoritmo 1: selcor

Output: una palabra de Bruijn
Input: *arista*
Input: k
Result: *palabra*
secuencia $\leftarrow$ [];
paso_actual $\leftarrow$ *arista*;
while *Repetir* 2^k *veces* **do**
  Agregar a secuencia el último símbolo de *paso_actual*;
  paso_actual $\leftarrow$ *paso_bruijn*(*paso_actual*)
palabra $\leftarrow$todos los símbolos de la secuencia concatenados en una palabra

Calculamos cada paso de Bruijn a partir de una arista. La arista es la última que recorrimos y ahora debemos definir cuál es la siguiente arista a recorrer dentro del grafo. Para lograr esto calculamos cuál es el siguiente paso en el CCR de la arista y la posible salida (rotación a izquierda). Si la posible salida es una arista perteneciente al mismo CCR la ignoramos y el siguiente paso es la arista siguiente en el CCR. En cambio, si la salida pertenece a otro CCR, nos fijamos si ésta es la arista distinguida de dicho CCR. Si lo es, el próximo paso es esta última arista.



Si todo lo anterior no se cumple, verificamos si el paso siguiente en el CCR es la arista distinguida de este CCR. Si lo es significa que es el punto de entrada/salida desde otro CCR hacia éste. En caso de serlo, el siguiente paso es la salida calculada al principio.

El último paso es determinar cuál es la arista distinguida de un CCR. El CCR es una lista de aristas, calculadas con la funcion siguiente desde una arista inicial. Para determinar la arista distinguida de un CCR verificaremos todas las posibles entradas (rotaciones a izquierda) desde CCR con *r-diferencia* menor al actual. Para hacer esto, rotaremos a derecha (proceso inverso al anterior) todas las aristas del CCR actual y veremos cuales pertenecen a CCRs con *r-diferencia* menor. De todas esas aristas seleccionamos la arista con mayor valor lexicográfico. Si no existen entradas desde CCRs con *r-diferencia* menor, nos fijamos si existen entradas desde un CCR con *r-diferencia* igual al CCR actual. Repetimos el procedimiento, de esta lista seleccionamos la arista con mayor valor lexicográfico.

Algoritmo 2: paso_bruijn

Output: La próxima arista a recorrer
Input: *arista*
Result: *proxima*
sig ← *siguiente*(*arista*);
posible_salida ← rotacion-izq(*arista*);
ccr_actual ← *buildCCR*(*arista*);
if *posible_salida* ∈ *ccr_actual* **then**
    proxima ← *sig*;
else
    ccr_salida ← *buildCCR*(*posible_salida*);
    if *posible_salida* == *arista_distinguida*(*ccr_salida*) **then**
        proxima ← *posible_salida*;
    else if *sig* == *arista_distinguida*(*ccr_actual*) **then**
        proxima ← *posible_salida*;
    else
        proxima ← *sig*;

Algoritmo 3: arista_distinguida

Output: Si existe, devuelve la arista distinguida del CCR
Input: *ccr*
Result: *distinguida*
$r\_diferencia\_ccr$ ← calculamos la *r-diferencia* de cualquier arista del ccr;
rotadas_derecha ← rotamos a derecha todas las palabras del ccr;
$menores\_rdif$ ← $\{e | \forall e \in rotadas\_derechas : r\text{-}diferencia(e) < r\_diferencia\_ccr\}$;
$iguales\_rdif$ ← $\{e | \forall e \in rotadas\_derechas : r\text{-}diferencia(e) = r\_diferencia\_ccr\}$;
if $menores\_rdif.lenght > 0$ **then**
    distinguida ← $mas\_grande\_lexicografica(menores\_rdif.map$(rotacion-izq)$)$;
else if $iguales\_rdif.lenght > 0$ **then**
    distinguida ← $mas\_grande\_lexicografica(iguales\_rdif.map$(rotacion-izq)$)$;

5. EL ALGORITMO DE HUANG ES CORRECTO

Teorema 1. *Dado un entero positivo k, el algoritmo de Huang construye un ciclo euleriano en el grafo de Bruijn de orden k-1.*

Lema 1. *Sean β_0 y β_1 dos palabras del mismo ciclo CCR, entonces*

$$r\text{-}diferencia(\beta_0) \equiv r\text{-}diferencia(\beta_1).$$

Demostración. Sea β_0 una palabra arbitraria de un ciclo CCR, entonces:

$$\beta_0 = a_1 a_2 a_3 .. a_k$$

$$E(\beta_0) = a_1 a_2 a_3 .. a_k \overline{a_1}$$

Y sea β_1 la siguiente palabra a β_0 en el ciclo CCR

$$\beta_1 = siguiente(\beta_0) = E(\beta_0)[2..k+1] = a_2 a_3 .. a_k \overline{a_1}$$

$$E(\beta_1) = a_2 a_3 .. a_k \overline{a_1 a_2}$$

Al remover a_1 de β_1 podríamos estar reduciendo en 1 la *r-diferencia* al comparar a_1 con a_2. Al calcular la *r-diferencia* de β_1 usamos la palabra $a_2 a_3 .. a_k \overline{a_1 a_2}$ entonces notar que estamos sumando la posible diferencia de $\overline{a_1 a_2}$ por lo cual, si $a_1 = a_2$ entonces $\overline{a_1} = \overline{a_2}$, y de forma análoga, si eran distintos continúan siendo distintos manteniendo así la misma *r-diferencia* entre β_0 y β_1. Esto vale para cualquier par de palabras consecutivas dentro del mismo CCR, luego todas las palabras del mismo CCR tienen la misma *r-diferencia.* □

Lema 2. *Si un ciclo CCR A puede ser accedido desde un ciclo CCR B, B también puede acceder A.*

Demostración. Sea la palabra $a_1 a_2 .. a_k$ una palabra de A un ciclo CCR y supongamos que $a_2 .. a_k a_1$ pertenece a otro ciclo CCR llamado B.

Rotando a izquierda la palabra $a_1 a_2 .. a_k$ queda $a_2 .. a_k a_1$ llegando así del ciclo CCR A, al ciclo CCR B. De forma analoga, rotamos a derecha $a_2 .. a_k a_1$ formando la palabra $a_1 a_2 .. a_k$ pudiendo ir desde el ciclo CCR B al ciclo CCR A. □

Lema 3. *Por cada $r \geq 3$ hay al menos un ciclo con r-diferencia de r-2 que se pueden acceder mutuamente.*

Demostración. De todos los ciclos con una *r-diferencia* de r hay al menos uno tiene la pinta

$$\beta_0 = i\ \bar{i}\ i\ a_4 a_5 .. a_k$$

$$r = r\text{-}diferencia(\beta_0) \geq 3$$

Donde i puede valer 0 ó 1. Y notemos que la siguiente también es una palabra en el mismo

ciclo CCR que β_0

$$\beta_1 = siguiente(\beta_0) = E(\beta_0)[2..k+1] = \bar{i}\ i\ a_4a_5..a_k\bar{i}$$

Al rotar a izquierda β_1 queda

$$\alpha_0 = i\ a_4a_5..a_k\bar{i}\ \bar{i}$$

$$r\text{-}diferencia(\alpha_0) = r - 2$$

Al cancular la *r-diferencia* de α_0 debemos extender la palabra quedando $i\ a_4a_5..a_k\bar{i}\ \overline{ii}$ y podemos ver que esta palabra pertenece a otro CCR porque disminuímos en dos la *r-diferencia* de β_0 ya que $i\ \bar{i}\ i$ sumaban 2 diferencias que ya no están.

Y por Lema 1 y 2 queda demostrado que para cada $r \geq 3$, existe un ciclo CCR con una *r-diferencia* de r y otro de *r-2* que pueden accederse mutuamente.

□

Si un ciclo B puede ser accedido desde un ciclo A (ó si B=A), decimos que A alcanza B. Si A alcanza B y el ciclo B alcanza C, entonces A alcanza ciclo C.

Lema 4. *Cualquier ciclo con una r-diferencia de r con $r \geq 3$ puede alcanzar otro ciclo con una r-diferencia de r que puede ser accedido desde un ciclo con una r-diferencia de r-2 sin pasar atraves de ningun ciclo con una r-diferencia de r+2*

Demostración. Si el ciclo tiene una palabra "$i\bar{i}i$..." es obvio por el lema 3. Si el ciclo T_1 no tiene la palabra "$i\bar{i}i$...", la palabra de valor más grande debe ser de la forma

$$(1\overbrace{111}^{t_{11}}10\overbrace{...}^{cualquier\ palabra}01\overbrace{111}^{t_{12}}1)$$

Donde t_{11} y t_{12} son tiras de 1 y tiene longitud mayor a 1. Asumímos que

$$(1\overbrace{111}^{t_{11}}10\overbrace{...}^{cualquier\ palabra}01\overbrace{111}^{t_{12}}1)$$

es la palabra de mayor valor en el ciclo CCR T_1. La palabra

$$(1\overbrace{111}^{t_{11}}10\overbrace{...}^{cualquier\ palabra}01\overbrace{111}^{t_{12}-1}1)$$

no puede pertenecer a T_1 pues su longitud no es la correcta. Luego la palabra

$$(1\overbrace{.,111..}^{t_{11}+1}10\overbrace{...}^{cualquier\ palabra}01\overbrace{.,111..}^{t_{12}-1}1)$$

no pertenece a T_1 pues su valor sería mayor a la palabra más grande de T_1, necesariamente esta palabra debe existir en otro ciclo CCR llamado T_2, que mantiene su *r-diferencia* y puede ser accedido desde T_2 ya que la operación que se realizo fue una rotación a derecha.

Luego T_2 mantiene su *r-diferencia* y si T_2 tiene la palabra "$i\bar{i}i$..." el lema queda probado.

Si T_2 no tiene la palabra "$i\bar{i}i$...", asumimos que tiene una palabra de mayor valor con la forma

$$(1\overbrace{.,111..}^{t_{21}}10\overbrace{...}^{cualquier\ palabra}01\overbrace{.,111..}^{t_{22}}1)$$

Notemos que $t_{21} > t_{11}$. Con el argumento anterior existe una palabra

$$(1\overbrace{.,111..}^{t_{21}+1}10\overbrace{...}^{cualquier\ palabra}01\overbrace{.,111..}^{t_{22}-1}1)$$

que pertenece a un ciclo T_3 que puede ser accedido desde T_2.

Repitiendo el procedimiento, tendremos los ciclos $T_1, T_2, ..., T_j$ con la misma *r-diferencia* que pueden ser accedidos entre ellos. Dado que $t_{11} < t_{21} < ... < t_{j1}$, debe existir un ciclo T_j tal que tenga la palabra "$i\bar{i}i$...". Notar que al incrementar j llegaremos a la palabra

$$(1\overbrace{.,111..}^{t_{j1}+1}10\overbrace{...}^{cualquier\ palabra}01)$$

que al extenderla formamos la palabra "... $i\bar{i}i$" y que al calcular el CCR completo llegamos a "$i\bar{i}i$..." □

Lema 5. *Si un ciclo con una r-diferencia de r tiene una palabra* $\beta_1 = (i^{\Theta+1}i\underbrace{a_1a_2...a_j}_{arbitraria})$ *con* $\Theta \geq 3$*, debe existir un ciclo con r-diferencia de r+2 que puede ser accedido desde dicho ciclo.*

Demostración. Sea

$$\beta_2 = siguiente(\beta_1) = E(\overbrace{iii...}^{\Theta} ia_1...a_j)[2..k+1] = (\overbrace{ii...}^{\Theta-1} ia_1...a_j\bar{i})$$

que pertenece al mismo *r-diferencia* ciclo. Como

$$\Gamma = \text{rotacion-izq}(\beta_2) = (\overbrace{i...}^{\Theta-2} ia_1...a_j\bar{i}i)$$

$$r\text{-}diferencia(\Gamma) = r\text{-}diferencia(\overbrace{i...}^{\Theta-2} ia_1...a_j\bar{i}i) = r\text{-}diferencia(\overbrace{ii...}^{\Theta-1} ia_1...a_j\bar{i}) + 2$$

el ciclo con una *r-diferencia* de *r+2* que contiene a Γ puede ser accedido desde el ciclo original. □

5.1. Demostración del Teorema 1

Por el Lema 5, el ciclo con una *r-diferencia* de 1 para una palabra de Bruijn de orden $n \geq 3$ puede acceder a un ciclo con una *r-diferencia* de 3.

Cuando $n = 1$ ó $n = 2$, existe un único CCR. En este caso producimos la palabra de Bruijn sin acceder a otros ciclos.

El algoritmo trabaja de la siguiente forma: Si un ciclo A con una *r-diferencia* de r tiene varias entradas que provienen de ciclos con una *r-diferencia* de *r-2*, la entrada seleccionada (distinguida) es la de valor más grande. Si el ciclo A no puede ser accedido desde un ciclo con una *r-diferencia* de *r-2*, por el Lema 4 puede ser accedido desde otro ciclo con una *r-diferencia* de r. Cuando el algoritmo vuelve al ciclo con una *r-diferencia* de *r-2* u otro ciclo con una *r-diferencia* de r, va a salir del ciclo A por el predecesor de la entrada distinguida y entra al ciclo anterior en el sucesor de la salida.

Si para cada tupla $(b_1, b_2, ..., b_n)$ la tupla $(b_2, ..., b_n, b_1)$ es una tupla de otro ciclo, los Lemas 3, 4, 5 garantizan que ninguna tupla sea ignorada por el algoritmo y el criterio de selección de entradas garantiza que ninguna tupla aparezca más de una vez.

6. ANEXOS

6.1. Publicación original del algoritmo de Yuejiang Huang

JOURNAL OF ALGORITHMS 11, 44–51 (1990)

A New Algorithm for the Generation of Binary de Bruijn Sequences

YUEJIANG HUANG*

University of Pittsburgh, Pennsylvania 15224

Received June 6, 1988; revised December 15, 1988

1. INTRODUCTION

A binary de Bruijn sequence is a binary sequence of length 2^n for which all n-tuple sequences are distinct. The general methods of approach to produce this kind of sequences are either based on the theory of finite fields [5] or a direct combinatorial approach [2–4, 6, 9]. A comprehensive survey of past work on this subject can be found in the paper by Fredricksen [1]. One common combinatorial approach is to consider a shift register producing many short cycles, e.g., the pure cycling register. These cycles are then joined together to form a full cycle.

The same practice is followed in this paper. We propose a new algorithm which joins all the pure cycles of the complementing circulating register (CCR) $f(X_1, X_2, \ldots, X_n) = \bar{X}_1$ to form a full cycle of 2^n. This algorithm uses about $4n$ bits of storage and $4n$ units of time to produce the next cycle bit from the last n bits. Since the algorithm uses adjacent bit differences instead of weights as the index of joining, it is expected that the sequences produced by the algorithm have a relatively good characteristic of local 0–1 balance compared with the primary "prefer one" algorithm [1, 3].

2. DEFINITION AND ALGORITHM

DEFINITION 1. For a binary n-tuple $(b_1, b_2, \ldots, b_n)$, an extended representation is given by an $(n+1)$-tuple $(b_1, b_2, \ldots, b_n) = (b_1, b_2, \ldots, b_n, b_{n+1})$, where $b_{n+1} = \bar{b}_1$ (the dash on the top of the symbol means logical "not"). If $b_j \neq b_{j+1}$ $(1 \leq j \leq n)$, we say b_j is different from b_{j+1}.

*Visiting scholar from Sichuan Communications Institute, P.O. Box 854, Chengdu, China.

0196-6774/90 $3.00

The number of differences in $E(b_1, b_2, \ldots, b_n)$ is expressed as $D(b_1, b_2, \ldots, b_n)$. The decimal value of the n-tuple $(b_1, b_2, \ldots, b_n)$ is expressed as $B(b_1, b_2, \ldots, b_n)$. The CCR cycle shifts of $(b_1, b_2, \ldots, b_n)$ are $(b_2, \ldots, b_n, \bar{b}_1)$, $(b_3, \ldots, b_n, \bar{b}_1, \bar{b}_2)$, and so on.

ALGORITHM.

(a) From $\beta_i = (b_i, b_{i+1}, \ldots, b_{i+n-1})$ we produce $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n})$.

(b) Forms $\beta_{i1}^* = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, b_i)$ and $\beta_{i2}^* = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, \bar{b}_i)$.

(c) If $\beta_{i1}^* = (1,1,\ldots,1)$ or $\beta_{i2}^* = (1,1,\ldots,1)$, $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, \bar{b}_i)$. Otherwise,

(d) If $D\beta_{i1}^* < D\beta_i$, go to (f); if $D\beta_{i1}^* = D\beta_i$, go to (g). Otherwise

(e) Consider all the cycle shifts of β_{i1}^*. If there is an n-tuple $M_j = (m_j, m_{j+1}, \ldots, m_{j+n-1})$ which makes $D(m_{j+n-1}, m_j, m_{j+1}, \ldots, m_{j+n-2}) < D\beta_{i1}^*$ and $BM_j > B\beta_{i1}^*$ hold, $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, \bar{b}_i)$. Otherwise, $\beta_{i+1} = b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, b_i)$. Stop.

(f) Consider all the cycle shifts of β_{i2}^*. If there is an n-tuple $M_j = (m_j, m_{j+1}, \ldots, m_{j+n-1})$ which makes $D(m_{j+n-1}, m_j, m_{j+1}, \ldots, m_{j+n-2}) < D\beta_{i2}^*$ and $BM_j > B\beta_{i2}^*$ hold, $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, \bar{b}_i)$. Otherwise $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, b_i)$. Stop.

(g) Consider all the cycle shifts of β_{i1}^*. If there is an n-tuple $M_j = (m_j, m_{j+1}, \ldots, m_{j+n-1})$ which makes $D(m_{j+n-1}, m_j, m_{j+1}, \ldots, m_{j+n-2}) < D\beta_{i1}^*$ or, $D(m_{j+n-1}, m_j, m_{j+1}, \ldots, m_{j+n-2}) = D\beta_{i1}^*$ and $BM_j > B\beta_{i1}^*$ hold, go to (h). Otherwise $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, b_i)$. Stop.

(h) Consider all the cycle shifts of β_{i2}^*. If there is an n-tuple $M_j = (m_j, m_{j+1}, \ldots, m_{j+n-1})$ which makes $D(m_{j+n-1}, m_j, \ldots, m_{j+n-2}) < D\beta_{i2}^*$ or, $D(m_{j+n-1}, m_j, \ldots, m_{j+n-2}) = D\beta_{i2}^*$ and $BM_j > B\beta_{i2}^*$ hold, $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, \bar{b}_i)$. Otherwise $\beta_{i+1} = (b_{i+1}, b_{i+2}, \ldots, b_{i+n-1}, b_i)$.

As an example of the algorithm, we consider the case of $n = 4$. Figure 1 shows the CCR pure cycles of $n = 4$.

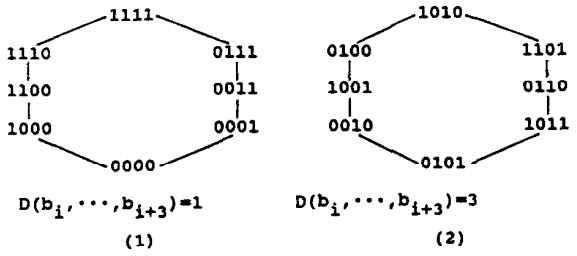


FIG. 1. The pure CCR cycles of $n = 4$.

TABLE 1
The List of DM_j' for β_{21}^*

M_j	M_j'	DM_j'
(1,0,1,0)	(0,1,0,1)	3
(0,1,0,0)	(0,0,1,0)	3
(1,0,0,1)	(1,1,0,0)	1
(0,0,1,0)	(0,0,0,1)	1
(0,1,0,1)	(1,0,1,0)	3
(1,0,1,1)	(1,1,0,1)	3
(0,1,1,0)	(0,0,1,1)	1
(1,1,0,1)	(1,1,1,0)	1

Suppose the algorithm begins with initial 4-tuple (1, 1, 1, 1). The algorithm forms $\beta_{11}^* = (1,1,1,1)$ and $\beta_{12}^* = (1,1,1,0)$. Since $\beta_{11}^* = (1,1,1,1)$, the algorithm executes step (c) and $\beta_2 = (1,1,1,0)$. For $\beta_2 = (1,1,1,0)$, the algorithm forms $\beta_{21}^* = (1,1,0,1)$ and $\beta_{22}^* = (1,1,0,0)$. Since $D\beta_{21}^* = D(1,1,0,1) = 3 > D\beta_2 = D(1,1,1,0) = 1$, the algorithm goes to (e). At step (e), the algorithm examines all the cycle shifts of β_{21}^* $(1,0,1,0)$, $(0,1,0,0),\ldots,(1,1,0,1)$. For each $M_j = (m_j, m_{j+1},\ldots, m_{j+n-1})$ of the cycle shifts, we use M_j' to represent $(m_{j+n-1}, m_j, m_{j+1},\ldots, m_{j+n-2})$. Table 1 lists the value of each DM_j'.

We can see from Table 1 that only (1, 0, 0, 1), (0, 0, 1, 0), (0, 1, 1, 0), and (1, 1, 0, 1) meet $DM_j' < D\beta_{21}^*$, but none of them meet $BM_j > B\beta_{21}^*$, so $\beta_3 = \beta_{21}^* = (1,1,0,1)$. Now the algorithm exits cycle (1) at (1, 1, 1, 0) and enters cycle (2) at (1, 1, 0, 1). With repetition of the algorithm, it will go through cycle (2) and exit cycle (2) at (0, 1, 1, 0) and reenter cycle (1) at (1, 1, 0, 0). Thus we have a binary de Bruijn sequence of length 16,

1111010010110000.

3. Proof of the Algorithm

As we know, the state transitions of any shift register can be expressed in terms of a directed graph G_n. The pure cycles of CCR can be viewed as a factor F of G_n. To join the pure CCR cycles into a full cycle is to find a Hamiltonian path through the G_n according to the F. The algorithm for production of de Bruijn sequences can be thought as a Hamiltonian path constructor. With this in mind, we begin to prove our algorithm.

Lemma 1. *Assume β_0 and α_0 are two arbitrary n-tuples in a* CCR *cycle. Then $D\beta_0 = D\alpha_0$.*

Proof. Assume $\beta_0 = (b_1, b_2, \ldots, b_n)$. Then $E\beta_0 = (b_1, b_2, \ldots, b_n, \bar{b}_1)$. For the successor $\beta_1 = (b_2, \ldots, b_n, \bar{b}_1)$, $E\beta_1 = (b_2, \ldots, b_n, \bar{b}_1, \bar{b}_2)$. If b_1 is different from b_2, $\bar{b}_1$ is also different from $\bar{b}_2$. Thus $D\beta_1 = D\beta_0$. Since this also holds for the successor of β_1 and so on, the lemma is proved.

If $D\beta_0 = r$ for a n-tuple β_0 in a CCR cycle, the CCR cycle is called an r-difference cycle ($r - d$ cycle for short).

DEFINITION 2. If a CCR cycle A has an n-tuple $\beta_0 = (b_1, b_2, \ldots, b_n)$ and $\beta_1 = (b_2, \ldots, b_n, b_1)$ is in another CCR cycle B, we say cycle B can be entered from cycle A. β_0 is called the exit and β_i the entry.

LEMMA 2. *If a* CCR *cycle B can be entered from a* CCR *cycle A, the cycle A can also be entered from cycle B.*

Proof. From the assumption of Lemma 2, let $(b_1, b_2, \ldots, b_n)$ in cycle A and $(b_2, \ldots, b_n, b_1)$ in cycle B. Since $(\bar{b}_1, b_2, \ldots, b_n)$ is the predecessor of $(b_2, \ldots, b_n, b_1)$ and in cycle B, $(b_2, \ldots, b_n, \bar{b}_1)$ is the successor of $(b_1, b_2, \ldots, b_n)$ and in cycle A, the lemma is proved by Definition 2.

According to Golomb [8], it is clear that if two CCR cycles can be entered from each other, the two cycles can be joined together.

LEMMA 3. *For each $r \geq 3$, there are at least one $r - d$ cycle and one $(r - 2) - d$ cycle which can be entered from each other.*

Proof. Of all $r - d$ cycles, there is at least one $r - d$ cycle which has an n-tuple $(ii'i \cdots)$, where $i = 1$ or 0, $i' = \bar{i}$. "$\cdots$" is used to represent the irrelevant part of the n-tuple. Since $(i'i \cdots i')$ is also an n-tuple in the cycle and $D(i \cdots i'i') = r - 2$. By Definition 2 and Lemma 2, the lemma is proved.

DEFINITION 3. If cycle B can be entered from cycle A or $B = A$, we say cycle A can reach cycle B. If cycle A can reach cycle B and cycle B can reach cycle C, we say cycle A can reach cycle C.

It is obvious that the reachability is reflexive, symmetrical and transitive.

Reachability between any two CCR cycles is a necessary condition that the pure CCR cycles can be joined into a full cycle.

LEMMA 4. *Any $r - d$ cycle ($r \geq 3$) can reach an $r - d$ cycle which can be entered from an $(r - 2) - d$ cycle without going through any $(r + 2) - d$ cycle.*

Proof. It is obvious when an $r - d$ cycle has an n-tuple $(ii'i \cdots)$. If an $r - d$ cycle T_1 does not have an n-tuple $(ii'i \cdots)$, the n-tuple of highest

value in the cycle must be of the form

$$\left(1\overbrace{\cdots}^{t_{11}}10\cdots01\overbrace{\cdots}^{t_{12}}1\right),$$

where t_{11} and t_{12} are greater than 1. Here we assume that

$$\left(1\overbrace{\cdots}^{t_{11}}10\cdots01\overbrace{\cdots}^{t_{12}}1\right)$$

is the n-tuple of highest value in cycle T_1. Then,

$$\left(1\overbrace{\cdots}^{t_{11}}10\cdots01\overbrace{\cdots}^{t_{12}-1}10\right)$$

cannot be in cycle T_1; otherwise, n-tuple

$$\left(1\overbrace{\cdots}^{t_{11}+1}10\cdots01\overbrace{\cdots}^{t_{12}-1}1\right)$$

is in cycle T_1 and its value is higher than the one of

$$\left(1\overbrace{\cdots}^{t_{11}}10\cdots01\overbrace{\cdots}^{t_{12}}1\right).$$

Hence cycle T_1 can be entered from cycle T_2 which has

$$\left(1\overbrace{\cdots}^{t_{11}+1}10\cdots01\overbrace{\cdots}^{t_{12}-1}1\right).$$

Since T_2 is also an $r-d$ cycle, if T_2 has an n-tuple $(ii'i\cdots)$, our result is proved.

If T_2 does not have n-tuple $(ii'i\cdots)$, we assume that the n-tuple of highest value in cycle T_2 is

$$\left(1\overbrace{\cdots}^{t_{21}}10\cdots01\overbrace{\cdots}^{t_{22}}1\right).$$

It is obvious that $t_{21} > t_{11}$. Similar to that showed above, the cycle can be

entered from cycle T_3 which has n-tuple

$$\left(1\overbrace{\cdots}^{t_{21}+1}10\cdots01\overbrace{\cdots}^{t_{22}-1}1\right).$$

Repeating this procedure, we will have $r-d$ cycles $T_1, T_2, \cdots, T_j \cdots$ which can reach one another. Since we also have $t_{11} < t_{21} < \cdots < t_{j1}$, there must exist an $r-d$ cycle T_j which has an n-tuple $(ii'i\cdots)$ as j increases.

LEMMA 5. *If an $r-d$ CCR cycle has an n-tuple*

$$\left(\overbrace{i\cdots i}^{\theta}\cdots\right) \qquad (\theta \geq 3),$$

there must exist an $(r+2)-d$ cycle which can be entered from the $r-d$ cycle.

Proof.

$$\left(\overbrace{i\cdots i}^{\theta-1}\cdots i'\right)$$

must be in the same $r-d$ cycle. Since

$$D\left(\overbrace{i\cdots i}^{\theta-2}\cdots i'i\right) = D\left(\overbrace{i\cdots i}^{\theta-1}\cdots i'\right) + 2,$$

the $(r+2)-d$ cycle containing

$$\left(\overbrace{i\cdots i}^{\theta-2}\cdots i'i\right)$$

can be entered from the $r-d$ cycle.

Because of Lemma 5, the $1-d$ cycle of $n \geq 3$ can enter a $3-d$ cycle. When $n = 1$ or $n = 2$, CCR has only one cycle. In this case, we can produce a binary de Bruijn sequence without joining. The algorithm works as follows:

If an $r-d$ cycle A has many entries by which the cycle can be entered from many $(r-2)-d$ cycles, the selected entry is the entry with the highest value. If the $r-d$ cycle cannot be entered from any $(r-2)-d$ cycle, by Lemma 4 it can be entered from other $r-d$ cycles. The selected entry is also the entry with the highest value of all the entries. When the algorithm goes back to the $(r-2)-d$ or another $r-d$ cycle, it will exit

cycle A at the predecessor of selected entry and enter the previous cycle through only the successor of the exit.

If for any n-tuple $(b_1, b_2, \ldots, b_n)$, $(b_2, \ldots, b_n, b_1)$ is an n-tuple of another cycle, it is clear that Lemmas 3 to 5 guarantee that no n-tuples will be ignored by the algorithm and the selection of entries guarantees that no n-tuples will appear twice or more. What we have to explain further is the case where $(b_1, b_2, \ldots, b_n)$ and $(b_2, \ldots, b_n, b_1)$ are in the same cycle. This is because if $(b_1, b_2, \ldots, b_n)$ and $(b_2, \ldots, b_n, b_1)$ are in the same cycle and $(b_2, \ldots, b_n, b_1)$ is of highest value in the cycle, the algorithm may enter a loop of several n-tuples or ignore some n-tuples. We will prove that this case never happens.

Let us first assume that there is a pair of $(b_1, b_2 \cdots b_n)$ and $(b_2, \ldots, b_n, b_1)$ in an $r - d$ cycle which can be entered from an $(r - 2) - d$ cycle. Since $D(b_2, \ldots, b_n, b_1)$ is also r, the algorithm will not let $\beta_{i+1} = (b_2, \ldots, b_n, b_1)$ because the cycle can be entered from an $(r - 2) - d$ cycle.

If the $r - d$ cycle cannot be entered from an $(r - 2) - d$ cycle and $(b_2, \ldots, b_n, b_1)$ is of highest value in the cycle, as shown in the proof of Lemma 4, $(b_1, b_2, \ldots, b_n)$ can only be in another cycle. It is true except $(1, 1, \ldots, 1)$, but the algorithm treats this n-tuple specially. If $(b_2, \ldots, b_n, b_1)$ is not of highest value, the algorithm never considers it as an entry. Thus we have

THEOREM 1. *The algorithm of Section 2 produces binary de Bruijn sequences.*

4. TIME-STORAGE REQUIREMENT AND DISCUSSION

The storage required for the above algorithm is n bits for the current n-tuple β_i, n bits for the cycling β_{i1}^* or β_{i2}^* (β_{i2}^* can utilize the storage of β_{i1}^*), n bits for $B\beta_{i1}^*$ or $B\beta_{i2}^*$ and less than n bits for $D\beta_{i1}^*$ or $D\beta_{i2}^*$. Since a pure CCR cycle has at most $2n$ n-tuples and two cycling will be needed when an $r - d$ cycle goes back to another $r - d$ cycle, the algorithm will require at most $4n$ units of time to produce the next bit.

The Xie's algorithm [9] can produce all of de Bruijn Sequences. To this extent, it is a very appreciable algorithm. Unfortunately, the algorithm needs 2^{n-1} bits of storage when producing binary de Bruijn sequences of span n. It is infeasible to use the algorithm in the applications, such as cryptography, which require large n (see $n = 100$).

Our algorithm needs only $o(n)$ bits of storage and $o(n)$ units of time to produce the next bit of a de Bruijn sequence from the last n bits, so it may be used in the applications of large n requirement. After some modifica-

tions, the algorithm can also produce many different Binary de Bruijn sequences of same span n.

The "prefer one" algorithm proposed by Fredricksen [1, 3] joins the pure cycles of circulating register (CR) in order according to the weights of the n-tuples and most of pure CR cycles are not 0–1 balanced, so some part of the sequence may contain many heavily weighed n-tuples and it leads to a bad local 0–1 balance.

The algorithm proposed here joins the pure cycles of CCR together in order according to the bit differences of the n-tuples and each pure CCR cycle is well 0–1 balanced. It seems clear that the sequences produced by our algorithm have a relatively good characteristic of local 0–1 balance in comparison with the ones produced by the "prefer one" algorithm.

Another good characteristic of the algorithm is that it can begin with any n-tuple.

REFERENCES

1. H. FREDRICKSEN, A survey of full length nonlinear shift register cycle algorithms, *SIAM Rev.* **24**, No. 2 (1982), 195–220.
2. H. FREDRICKSEN, A class of nonlinear de Bruijn cycles, *J. Combin. Theory Ser. A* **19** (1975), 192–199.
3. H. FREDRICKSEN, Generation of the Ford sequence of length 2^n, n large, *J. Combin. Theory Ser. A* **12** (1972), 153–154.
4. H. FREDRICKSEN AND J. MAIORANA, Necklaces of beads in K colors and k-ary de Bruijn sequences, *Discrete Math.* **23** (1978), 207–210.
5. D. E. KNUTH, "The Art of Computer Programming," Vol. 2, Addison–Wesley, Reading, MA, 1969.
6. A. RALSTON, A new memoryless algorithm for de Bruijn sequences, *J. Algorithms* **2** (1981), 50–62.
7. T. ETZION AND A. LEMPEL, Algorithms for the generation of full length shift register sequences, *IEEE Trans. Inform. Theory* **IT-30**, No. 3 (1984).
8. S. W. GOLOMB, "Shift Register Sequences," Holden–Day, San Francisco, 1967.
9. S. XIE, Note on de Bruijn sequences, *Discrete Appl. Math.* **16** (1987), 157–177.

6.2. Algoritmo de esta tesis

```
type Binario = 0 | 1;
type Arista = Array<Binario>;

function neg(n: Binario) {
    return n === 0 ? 1 : 0;
}

function extender(arista: Arista): Arista {
    return [...arista, neg(arista[0])];
}

function siguiente(arista: Arista): Arista {
    return extender(arista).slice(1);
}

function ccr(arista: Arista): Arista[] {
    const ccr_list: Arista[] = [];
    let actual = arista;

    while (!ccr_list.map(e => e.toString()).includes(actual.toString())) {
        ccr_list.push(actual);
        actual = siguiente(actual);
    }

    return ccr_list;
}

function rotacion_izq(arista: Arista): Arista {
    return [... arista.slice(1), arista[0]];
}

function rotacion_der(arista: Arista): Arista {
    return [arista[arista.length-1], ...arista.slice(0, -1)];
}

function binario_a_decimal(arista: Arista): number {
    return parseInt(arista.join(''), 2);
}

function r_diferencia(arista: Arista): number {
    const extendida = extender(arista);
    let diferencia = 0;
    for (let i = 0; i < arista.length; i++) {
        if (extendida[i] !== extendida[i+1]) {
            diferencia += 1;
        }
    }
    return diferencia;
}

function arista_mas_grande_en_decimal(aristas: Arista[]): Arista {
    return aristas.reduce((a, b) => binario_a_decimal(a) > binario_a_decimal(b) ? a : b);
}

function arista_distinguida(ccr: Arista[]) {
    const r_diferencia_ccr_actual = r_diferencia(ccr[0]);

    // todas las posibles formas de entrar a este CCR
    const aristas_rotadas_derecha = ccr.map(rotacion_der);

    const aristas_menor_rdif = aristas_rotadas_derecha.filter(e => r_diferencia(e) < r_diferencia_ccr_actual);
    if (aristas_menor_rdif.length) {
        return arista_mas_grande_en_decimal(aristas_menor_rdif.map(rotacion_izq));
    }

    const aristas_mismo_rdif = aristas_rotadas_derecha.filter(e => r_diferencia(e) === r_diferencia_ccr_actual)
        ;
    if (aristas_mismo_rdif.length) {
        return arista_mas_grande_en_decimal(aristas_mismo_rdif.map(rotacion_izq));
    }
}

function paso_de_bruijn(actual: Arista): Arista {
    const sig = siguiente(actual);
    const salida = rotacion_izq(actual);
    const ccr_actual = ccr(actual);

    // chequear si la salida es parte del CCR actual
    if (ccr_actual.map(e => e.toString()).includes(salida.toString())) {
        return sig;
    }

    // evaluo si salida es la arista distinguida de del CCR correspondiente a salida
    const ccr_salida = ccr(salida);
    const ccr_salida_arista_distinguida = arista_distinguida(ccr_salida);

    // chequear si ccr_salida_arista_distinguida no esta vacia y es igual a la salida
    if (ccr_salida_arista_distinguida && ccr_salida_arista_distinguida.toString() === salida.toString()) {
```

```
88          return salida;
89      }
90
91      const ccr_actual_arista_distinguida = arista_distinguida(ccr_actual);
92      if (ccr_actual_arista_distinguida && ccr_actual_arista_distinguida.toString() === sig.toString()) {
93          return salida
94      }
95
96      return sig;
97  }
98
99  function selcor(arista: Arista, k: number) {
100     const historial: Binario[] = [];
101     let paso_actual = arista;
102     const n = 2**k;
103
104     for (let i = 0; i < n + k - 1; i++) {
105         historial.push(paso_actual[k-1]);
106         paso_actual = paso_de_bruijn(paso_actual);
107     }
108     return historial.join('');
109 }
110
111 console.log(selcor([0, 0, 0, 0, 1, 1, 1, 1], 8))
```

6.3. Salidas para k=1,2,3,4,5,6,7,8

Exponemos las salidas del algoritmo para los k=1,2,3,4,5,6,7,8 empezando el algoritmo desde la arista de todos símbolos 0.

Secuencia de Bruijn de orden 1 empezando desde 0

01

Secuencia de Bruijn de orden 2 empezando desde 00

01100

Secuencia de Bruijn de orden 3 empezando desde 000

0111010001

Secuencia de Bruijn de orden 4 empezando desde 0000

0111101001011000011

Secuencia de Bruijn de orden 5 empezando desde 00000

01111101010001011100100110110000111

Secuencia de Bruijn de orden 6 empezando desde 000000

0111111010100101011010000101111001100100011011100010011101100000001111

Secuencia de Bruijn de orden 7 empezando desde 0000000

0111111101010100010101110100000101111100110001100111001010011010110010000011011110001000111011100001001111011010010010101011000000011111

Secuencia de Bruijn de orden 8 empezando desde 00000000

0111111110101010010101011010100001010111101001000101101110100000101111110011100011000111001101001100101100110000110011100101000110101110010000011011111000100011101110000100111110110100010010111011001001001101101100000000011111

Bibliografía

[1] Jean Berstel and Dominique Perrin. The origins of combinatorics on words. *European Journal of Combinatorics*, 28(3):996–1022, 2007.

[2] Joshua Cooper and Christine Heitsch. The discrepancy of the lex-least de Bruijn sequence. *Discrete Mathematics*, 310(6-7):1152–1159, 2010.

[3] N. G. de Bruijn. A combinatorial problem. *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, 49:758–764, 1946. (=Indagationes Math. 8 (1946) 461-467).

[4] Daniel Gabric and Joe Sawada. Investigating the discrepancy property of de Bruijn sequences. *Discrete Mathematics*, 345(4):112780, 2022.

[5] Yue Jiang Huang. A new algorithm for the generation of binary de Bruijn sequences. *Journal of Algorithms*, 11(1):44–51, 1990.