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

Extending de Bruijn sequences to larger alphabets

Tesis de Licenciatura en Ciencias de la Computación

Lucas Cortés
lucascortes@me.com

Directora: Verónica Becher

Buenos Aires, 28 de noviembre de 2018

EXTENSIÓN DE SECUENCIAS DE BRUIJN EN ALFABETOS MÁS GRANDES

Un secuencia circular de Bruijn de orden n en k colores es una secuencia en la que cada palabra de longitud n ocurre exactamente una vez. En esta tesis demostramos que para cada secuencia circular de Bruijn v de orden n en k colores hay otra secuencia circular de Bruijn w de orden n pero en $k+1$ colores tal que v es una subsecuencia de w y entre cualesquiera dos ocurrencias sucesivas del nuevo símbolo en w hay a lo sumo $n+2k-2$ símbolos consecutivos de v. Damos un algoritmo que recibe una tal secuencia v y produce la secuencia w. Damos además un algoritmo mucho más rápido que recibe una tal secuencia v y produce una secuencia w pero sin la garantía de que el nuevo símbolo esté balanceado.

Palabras claves: secuencias de Bruijn, ciclos Eulerianos.

EXTENDING DE BRUIJN SEQUENCES TO LARGER ALPHABETS

A circular de Bruijn sequence of order n in k colors is a sequence in which every possible word of length n occurs exactly once. In this thesis we show that for any given circular de Bruijn sequence v of order n in k colors there is another circular de Bruijn sequence w of order n but in $k+1$ colors such that v is a subsequence of w and such that in between two successive occurrences of the new colored symbol in w there are at most $n+2k-2$ consecutive symbols of v. We provide an algorithm that given such an input sequence v produces the output sequence w. And we give a much faster algorithm that also receives as input such a sequence v and outputs a sequence w without the guarantee of the fair distribution of the new colored symbol.

Keywords: de Bruijn sequences, Eulerian cycles.

CONTENTS

1. INTRODUCTION AND STATEMENT OF RESULTS

We start with the classical definitions. A *word* is a finite sequence of symbols in a given alphabet. A *rotation* is the operation that moves the final symbol of a word to the first position while shifting all other symbols to the next position, or it is the composition of this operation with itself an arbitrary number of times. A *circular word* is the equivalence class of a word under rotations. We write $[abc]$ to denote the circular word formed by the rotations of abc. In this work we use the terms word and sequence interchangeably.

We say that a *subsequence* of a sequence $a_1a_2\ldots a_n$ is a sequence $b_1b_2\ldots b_k$ defined by $b_i = a_{n_i}$ for $1 \leq i \leq k$, where $n_1 \leq n_2 \leq \ldots \leq n_k$ is an increasing sequence of indices. The same applies to circular words, assuming any starting position. For example, $[1, 2, 3]$, $[2, 4, 6]$ and [5,6,1,2] are subsequences of $[1, 2, 3, 4, 5, 6]$.

A *circular de Bruijn sequence* of order n, also known as a circular de Bruijn word of order n, on a size-k alphabet A is a circular word of size k^n in which every possible size-n word on A occurs exactly once as a contiguous subsequence [3, 6]. See [2] for a fine presentation and history. To denote the set of circular de Bruijn words of order n in an alphabet of k symbols we write $B(k, n)$. For example, $[0, 0, 1, 1]$ is in $B(2, 2)$.

In this note we show that for any given circular de Bruijn sequence v of order n in an alphabet of k symbols there is another circular de Bruijn sequence w of order n but in an alphabet of $k+1$ symbols such that v is a subsequence of w and such that in between two successive occurrences of the new symbol in w there are at most $n + 2k - 2$ consecutive symbols of v. We provide an algorithm that given such an input sequence v produces the output sequence w. And we give a much faster algorithm that also receives as input such a sequence v and outputs a sequence w without the guarantee of the fair distribution of the new symbol. Thus, Theorems 1 and 2 stated below are the main results of this note:

Theorem 1. *Given a circular de Bruijn sequence v in $B(k, n)$ there is a circular de Bruijn sequence w in $B(k+1, n)$ such that v is a subsequence of w and for any $2k+n-1$ consecutive symbols in w there is at least one occurrence of the new symbol s. Moreover, there is an algorithm that given as input such a sequence v generates the sequence w after performing $O(k^{3n-2})$ mathematical operations and it uses $O((k + 1)^n)$ space.*

For example, given the $B(2, 3)$ sequence

$$v = [1, 1, 0, 0, 0, 1, 0, 1]$$

the following $B(3, 3)$ sequence

$$w = [1, 2, 2, 2, 1, 2, 1, 1, 1, 0, 0, 2, 2, 0, 2, 0, 0, 0, 1, 2, 0, 1, 0, 2, 1, 0, 1]$$

satisfies the conditions of the theorem for $k = 2$ and $n = 3$ in the alphabet $A = \{0, 1\}$ where the new symbol s is the symbol 2. Observe that the symbol 2 occurs $(k+1)^{n-1} = 9$ times in w and given any $n + 2k - 1 = 6$ consecutive symbols in w there is at least one occurrence of the symbol 2.

It is not hard to see that given a sequence v in $B(k,n)$ there is a sequence w in $B(k+1,n)$ such that v is a subsequence of w. But we aim to guarantee that the new symbol s is fairly distributed along the extended de Bruijn sequence w. The first difficulty is to mathematically define this condition. The second difficulty is to prove the existence of such an extended sequence w and to provide an elegant and fast algorithm to construct it.

In addition to classical elements from graph theory such as de Bruijn graphs, Eulerian cycles and graph transformations, we use the Edmonds-Karp algorithm [4, 8]. Note that the output sequence obtained by our algorithm has size $(k+1)^n$. Thus, Theorem 1 states that the algorithm is practically cubic on the output size and this time complexity is dominated by the Edmonds-Karp $O(V^2E)$ time complexity when operating on a graph with V vertices and E edges.

In case we ask for no guarantee on the distribution of the new symbol in the extended sequence, we obtain a faster algorithm.

Theorem 2. *There is an algorithm that given a circular de Bruijn sequence v in $B(k,n)$ generates a circular de Bruijn sequence sequence w in $B(k+1,n)$ such that v is a subsequence of w, after performing at most $O(n^2(k+1)^n)$ mathematical operations and it uses $O((k+1)^n)$ space.*

Note that the sequence w generated by this second algoritm has size $(k+1)^n$. Thus, the time complexity of this second algorithm is just above the size of the input. Precisely, for each symbol of the generated sequence w this second algorithm performs a number of operations that is the square of the logarithm of the size of the output sequence.

The proof of Theorem 2 is elementary and this second algorithm formalizes the intuition one may have about extending a de Bruijn sequence to a larger alphabet. We shall see that the algorithm is greedy, making just some computations on each step.

It is worth to mention that recently Gabriel Thibeault in [7], proved that the lexicographically greatest de Bruijn sequence v in $B(k,n)$ is the suffix of the lexicographically greatest sequence w in $B(k+1,n)$. Thus, for the lexicographically greatest de Bruijn sequence in $B(k,n)$ there is a very simple solution to the problem stated in Theorem 2, which is to construct the lexicographically greatest de Bruijn sequence in $(k+1,n)$, and this can be done with a greedy algorithm.

The extension problem considered in [7] and also in this note was formulated first by Ariel Zylber in 2017, and it is dual to the extension problem studied by Becher and Heiber in [1], where they considered the problem of extending a sequence v in $B(k,n)$ to a sequence w in $B(k,n+1)$ such that v is a suffix of w.

The document is organized as follows. Chapter 2 presents the classic material related to de Bruijn graphs and we fix the notation. Since Theorem 2 is simpler to prove than Theorem 1 we start by proving Theorem 2 in Chapter 3. In Chapter 4 we elaborate the definition of fair distribution of the new symbol in the extended sequence. We devote the last chapter, Chapter 5, to the proof of Theorem 1.

2. DE BRUIJN GRAPHS AND TREES OF PETALS

Fix a finite alphabet A. Without loss of generality, when we consider an alphabet A of k symbols we assume $A = \{0, 1, \dots k-1\}$. As usual, we write A^n to denote the set of words of size n whose symbols belong to A.

A de Bruijn graph $G(k,n)$ is a directed graph (V,E) where V is the set of words of size n on a size-k alphabet A and whose set of edges E is the set of pairs (u,v) for $u = a_1a_2\dots a_n$ and $v = a_2\dots a_nb$ with $b \in A$. Thus, the graph has k^n vertices and k^{n+1} edges, it is strongly connected and every vertex has the same in-degree and out-degree.

Each circular de Bruijn word in $B(k,n)$ can be constructed by taking a Hamiltonian cycle on the $G(k,n)$ graph given that each vertex of the graph has each possible word of size k in an alphabet of k symbols. Moreover, since the line graph of $G(k,n)$ is $G(k,n+1)$ then each circular de Bruijn word in $B(k,n+1)$ can be constructed as an Eulerian cycle in $G(k,n)$. For example, in the $G(2,2)$ graph if one traverses the edge labelled 1 from 00, one arrives at 01 thereby indicating the presence of the contiguous subsequence 001 in the de Bruijn sequence.

Notice that $G(k,n)$ is a subgraph of $G(k+1,n)$. To see that, observe that the vertices of the first graph are all the possible size-n words in a size-k alphabet and the vertices of the second graph are those of the first one plus all the possible size-n words in an alphabet of size $k+1$ with at least one occurrence of the new symbol. Also, the edges of the second graph are the same as the ones from the first graph plus the ones representing words with at least one occurrence of the new symbol. This means that we can add vertices and edges to $G(k,n)$ and obtain $G(k+1,n)$. This motivates the following definition of the augmenting graph $D(k+1,n)$.

If w is a word on alphabet A and a is a symbol of A we write $|w|_a$ to denote the number of occurrences of a in w. Similarly, if u is a word we write $|w|_u$ to denote the number of occurrences of u in w.

Definition 3 (augmenting graph)**.** Let $\widehat{A} = A \cup \{s\}$ where A has k symbols and s is a symbol not in A. We define the augmenting graph $D(k+1,n) = (V,E)$ where

$$V = \widehat{A}^n$$
$$E = \{(v,w) : \text{ if } v = a_1\dots a_n \text{ then } w = a_2\dots a_nb \text{ where } b \in \widehat{A} \text{ and } (|v|_s > 0 \text{ or } |w|_s > 0)\}$$

To prove Theorems 1 and 2 we have to transform a given de Bruijn word in $B(k,n)$ into a de Bruijn word in $B(k+1,n)$ in such a way that the first one is a subsequence of the second one. Thus, given an Eulerian cycle c in $G(k,n-1)$ we need to construct an Eulerian cycle in $G(k+1,n-1)$ where we preserve the relative order of the edges in c. We can observe that in the augmenting graph $D(k+1,n-1)$ each of the vertices of $G(k,n-1)$ has exactly one incoming and outgoing edge. Also note that the outgoing edge is always labelled with the new symbol. So, the only way to define the expected Eulerian cycle in $G(k+1,n-1)$ is by interleaving disjoint cycles of the augmenting graph $D(k+1,n-1)$ on each of the vertices of $G(k,n-1)$. We will concentrate in some particular disjoint cycles

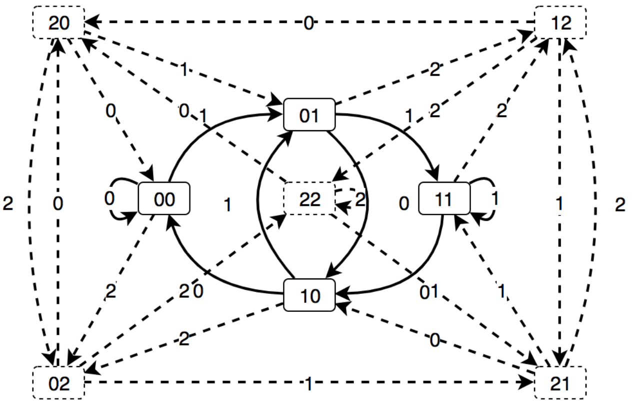

Fig. 2.1: The edges of the graph $D(3,2)$ are shown in dashed lines.

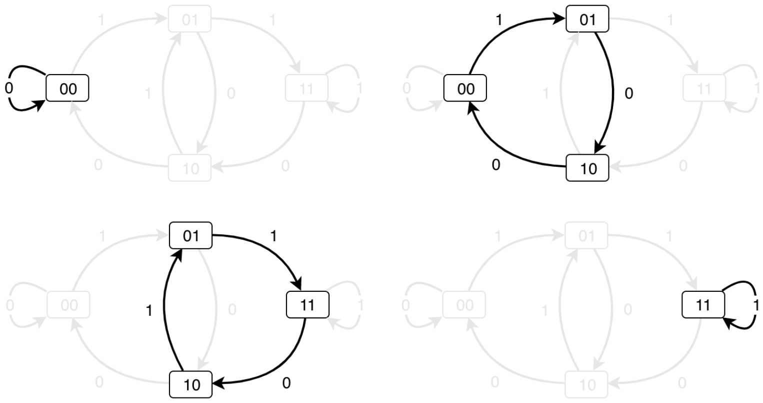

Fig. 2.2: Given a size-2 alphabet, there are 4 circular words of size 3: [000], [100], [110] and [111], each one associated with a cycle in $G(2,2)$.

in the augmenting graph $D(k+1,n-1)$ that we call *petals*. In order to do that, we use the following proposition.

Proposition 4. *Fix an integer k greater or equal to 2. The set of edges in $G(k,n)$ can be partitioned into a set of cycles identified by the circular words of size $n+1$.*

Proof. First note that we can unequivocally identify an edge of $G(k,n)$ by concatenating the outgoing vertex label with the label of that edge. Thus, each edge of $G(k,n)$ is identified with a word of size $n+1$. Also this word identifies a circular word of size $n+1$, which is the class of all the rotations of this word. Now notice that each circular word of size $n+1$ corresponds to exactly one cycle in $G(k,n)$. Thus the partition of the set of words of size $n+1$ in the equivalence class given by the rotations of these words determines a partition of the set of edges in $G(k,n)$ into cycles. □

In the following proposition we write $\sqcup$ to denote the disjoint union of two sets. It states that the augmenting graph $D(k+1,n)$ contains the set of cycles associated to circular

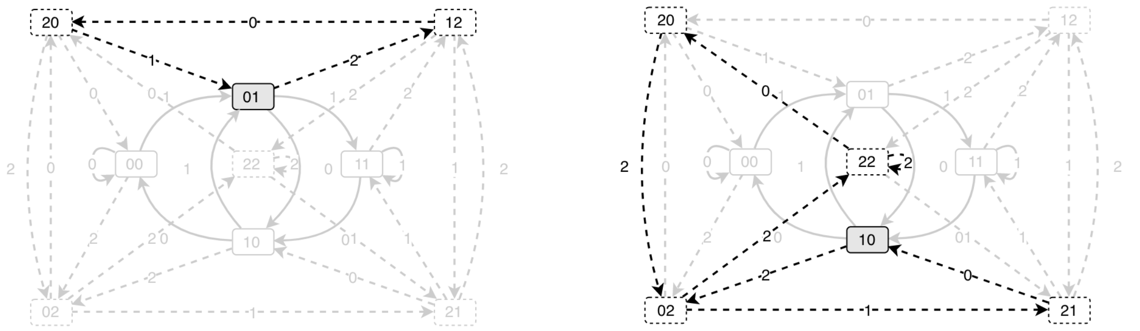

Fig. 2.3: Two examples of petals of a $G(2, 2)$ graph. The left figure has a petal for the vertex 01 that only contains one cycle, the one associated to the circular word [012]. The right figure has a petal for the vertex 10 that contains three cycles associated to the circular words [102], [022] and [222].

words of size $n + 1$ with at least one occurrence of the new symbol. It is immediate to verify that the proposition holds.

Proposition 5. *Let C be the set of cycles in $G(k, n)$ associated to the circular words of size $n+1$ in an alphabet of k symbols. Let $\widehat{C}$ be the set of cycles in $G(k+1, n)$ associated to the circular words of size $n + 1$ in an alphabet of $k + 1$ symbols. Then $\widehat{C} = C \sqcup P$, where P is the set of cycles associated to the circular words of size $n+1$ with at least one occurrence of the new symbol.*

We are now ready to define a *petal*.

Definition 6 (petal)**.** A *petal* of $G(k, n)$ is a cycle of cycles in $D(k + 1, n)$ associated to circular words of size $n + 1$ that traverses only one vertex of $G(k, n)$.

We aim to define the wanted Eulerian cycle in $G(k + 1, n)$ as the given cycle c in $G(k, n)$ interleaved with the petals of the augmenting graph $D(k + 1, n)$. The difficulty lies in determining how to define petals using *every* edge of $D(k + 1, n)$ and also how to interleave these petals in c to make sure that the occurrences of the new symbol are fairly distributed to satisfy the requirement of the theorem.

Definition 7 (Petals tree)**.** Let A be an alphabet with cardinality k with $k \geq 3$, r a circular de Bruijn word in $B(k - 1, n)$ and $s \in A$ such that $s \notin r$. We define the *Petals tree* $t(k, n, s)$ as a rooted tree subgraph of the directed graph $(V \cup \{r\}, E)$ where

$$\begin{aligned} V &= \{[w] : w \in A^n \text{ and } |w|_s \geq 1\} \\ E &= \{([v], [w]) : v, w \in A^n, \exists u \in A^{n-1}, |v|_u > 0, |w|_u > 0, |w|_s = |v|_s + 1\} \cup \\ &\qquad \{(r, [v]) : |v|_s = 1\}. \end{aligned}$$

Notice that the vertices with distance 1 to the root have exactly one occurrence of the symbol s, and each vertex of $t(k, n, s)$ with distance d to the root has exactly d occurrences of the new symbol.

Note that when two vertices are connected in $t(k, n, s)$ they have a common contiguous subsequence of size $n - 1$. We shall define a cycle that goes through several connected

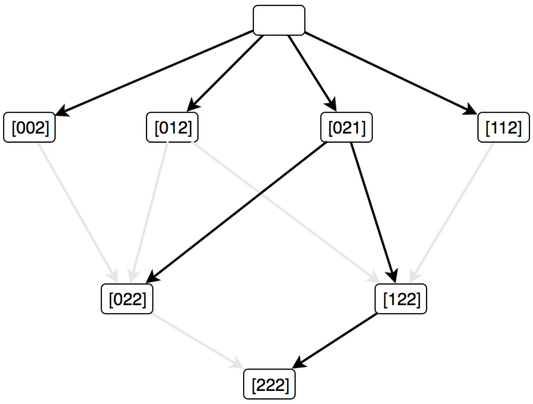

Fig. 2.4: A possible t(3,3,2) tree. The root r determines four petals, one for each branch. The first petal has the circular word [002], the second has [012], the third has [021], [022], [122] and [222] and the fourth has [112].

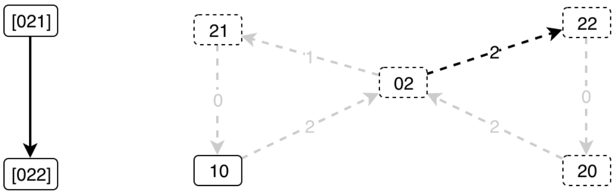

Fig. 2.5: On the left we have the circular words [021] and [022] from the Petals tree and their associated cycles on the right. Note that the first circular word has one occurrence of the symbol 2 and the second one has two. Also note that their associated cycles have the common vertex 02. Suppose we traverse the first cycle starting from the vertex 21. We would go through the edges 0 and 2 until we get to the common vertex 02. At that point, we start traversing the second cycle starting with the symbol 2 which guarantees a circular word with two occurrences of the symbol 2. We traverse 2, 0 and 2. After that, we finish the first cycle with the label 1.

cycles. In order to compose two cycles u and v we traverse the first circular word u until we find a common vertex w such that the next edge in u is not labelled with the new symbol s. Note that w has the same number of occurrences of s as u. Consequently, an edge labelled with s that starts from w corresponds to a circular word with more occurrences of the symbol s.

3. PROOF OF THEOREM 2

3.1 Extending a de Bruijn sequence to a larger alphabet

We give an algorithm that formalizes the intuition one may have about extending a de Bruijn sequence to a larger alphabet. Consider the graph for de Bruijn sequences of order n and alphabet in $k+1$ symbols. Notice that a de Bruijn sequence v in $B(k,n)$ is associated to an Eulerian cycle in $G(k,n-1)$ and a cycle in $G(k+1,n-1)$. The idea is to traverse this cycle and, greedily, at each vertex use the outgoing edge labelled with the new symbol s to extend the cycle. We already introduced a tool to traverse the de Bruijn graph $G(k+1,n-1)$ using a Petals tree starting with a $B(k,n)$ de Bruijn sequence q that represents an Eulerian cycle in the de Bruijn graph $G(k,n-1)$. There can be three different possibilities on each step. Consider the current vertex w and the edge s. One possibility is that they have not been traversed. In this case, start traversing the new cycle. Another possibility is that they are the current circular word. In this case we keep traversing the same circular word. A last possibility is that they have already been traversed. In this case we ignore this circular word.

In the following example we perform the first six steps of the algorithm just described. Assume as input a $B(2,3)$ de Bruijn sequence [00101110]. We begin the traversal in vertex 10 and immediately try to add a circular word with one occurrence of the symbol 2. The circular word of the vertex 10 and label 2 is the [210]. We traverse the edge 2 to the vertex 02. Then again we try to find a new circular word by traversing another edge labelled 2. The vertex 02 with the edge labelled 2 determines the circular word [202] and we traverse the edge 2 to the vertex 22. Again, we search a new circular word. The vertex 22 with the edge labelled 2 determines the circular word [222]. We traverse the edge 2 and get to the same vertex. Now, the circular word [222] is already used, so we have to go to the next edge of the current circular word. We traverse the edge labelled 0 to the vertex 20. Again, the circular word [220] is already in use, so we continue. When we get to the vertex 21 we again can start a new circular word, the [221].

3.2 An Algorithm to prove Theorem 2

Algorithm 1 takes a $B(k,n)$ de Bruijn word v and returns a $B(k+1,n)$ de Bruijn word w such that v is a subsequence of w where the new symbol s occurs $(k+1)^{n-1}$ times in w.

The main idea of the algorithm is to traverse an array with the original sequence adding petals and cycles whenever possible. We first determine the alphabet size and the order of the de Bruijn sequence. To find the alphabet size we just have to count how many different symbols the sequence has. We can get the order of the sequence by solving $k^n = \#edges$. Then we make a copy of the original sequence that we will modify to get the extended sequence. There are several variables to keep track of things. The variable *pos* keeps track of the current position in the array and represents the edges that we already traversed. The variable *vertex* indicates in which vertex are placed at each step. To make sure that

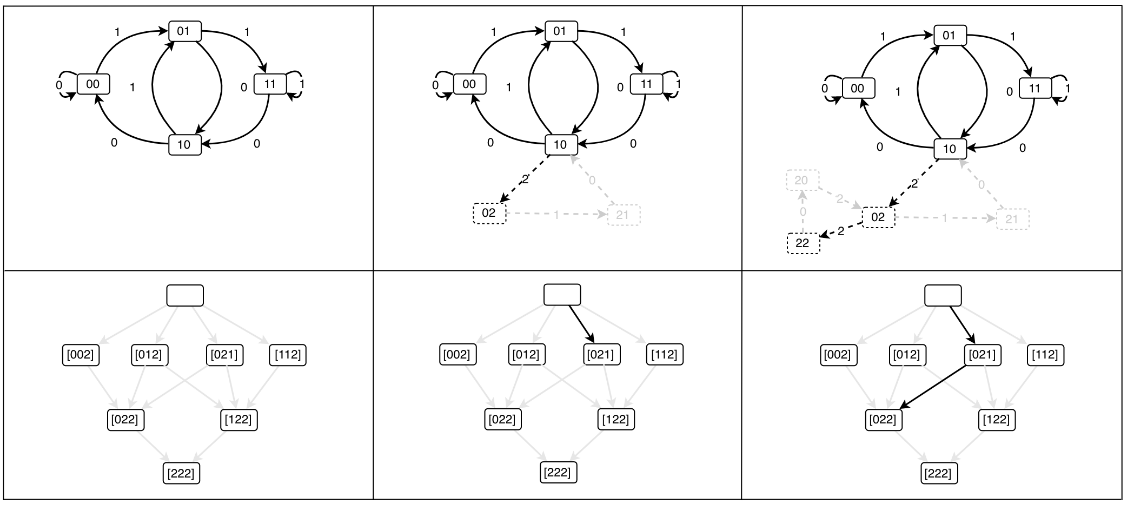

Fig. 3.1: First steps of the algorithm with a $B(2,3)$ de Bruijn sequence [00101110] as input. We show the circular words added on the petals tree.

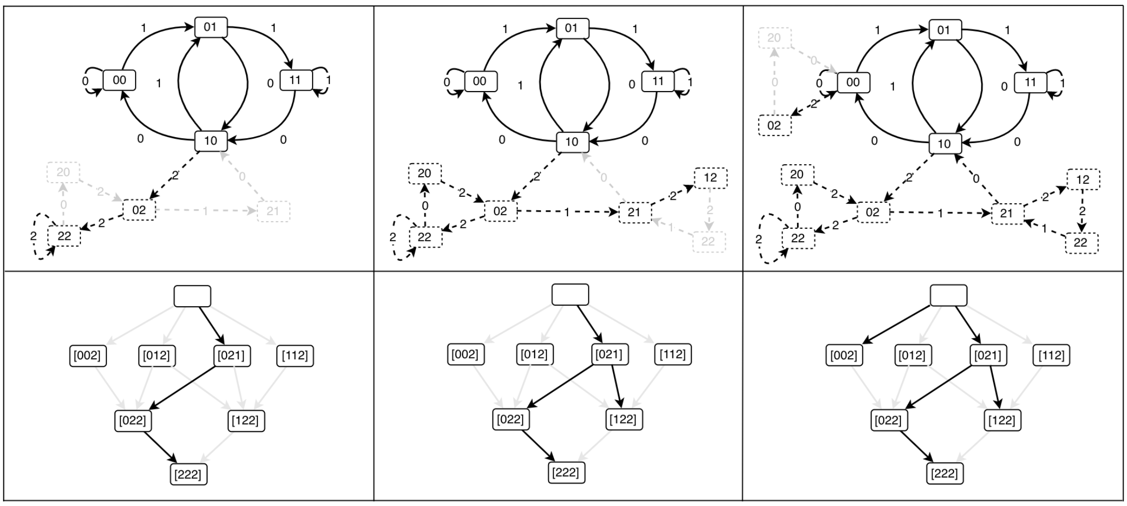

Fig. 3.2: Next steps of the algorithm with a $B(2,3)$ de Bruijn sequence [00101110] as input.

we do not traverse any cycle more than once we have to keep track of every edge of a traversed cycle. We can reduce space by just keeping track of the vertices such that their outgoing edge labelled with the new symbol belongs to a traversed cycle. We keep track of them in the array *visitedVertices*. In this way we can unequivocally decide whether or not we should add a cycle at each vertex.

Algorithm 1

```
function extendDeBruijn(originalSequence: [Int])
    let alphabetSize = getSize(originalSequence)
    let newAlphabetSize = alphabetSize + 1
    let order = getOrder(originalSequence)
    let newSymbol = alphabetSize
    var sequence = originalSequence
    var pos = 0
    var vertex = originalSequence.last(order - 1)
    var visitedVertices = [false, ...]
    while pos $\leq$ sequence.count do
        vertex = vertex.last(vertex.count - 1) + [edgeValue]
        pos += 1
        if !visitedVertices[vertex] then
            let edge = vertex + [newSymbol]
            var cycle = []
            for for i in 0..<edge.count do
                let newEdge = edge.last(edge.count - i) + edge.first(i)
                if if i > 0 && edge == newEdge then
                    break
                end if
                let newVertex = newEdge.first(order-1)
                cycle += newEdge.last(1)
                if newEdge.last == newSymbol then
                    visitedVertices[newVertex] = true
                end if
            end for
            sequence = sequence.first(pos) + cycle + sequence.last(sequence.count - pos)
        end if
    end while
    return sequence
end function
```

The main loop of Algorithm 1 iterates through every edge of the original sequence adding cycles. On each vertex v in position *pos* of the array we have two possibilities. If we already added the circular word determined by the concatenation of v and the new symbol s we ignore that circular word, increment *pos* and go to the next vertex in the sequence. If we did not already added that circular word, we have to add it. To do that, we traverse each edge of the cycle, add them to the sequence on the current position, and for those labelled with s we mark their outgoing vertex as *visited*.

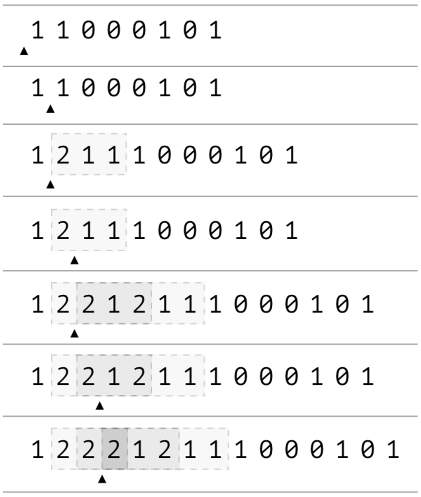

Fig. 3.3: The first steps of the algorithm for the $B(2,3)$ sequence 11000101.

Extra care is taken in writing the edges of the cycles. Note that not always the cycle associated to a word of size n has n edges. There are as many edges as equivalence classes of the word. The algorithm starts adding each edge of the cycle until it reaches the original vertex. Once the cycle is formed we place it in the current position and keep moving forward. This process continues until we reach the last position of the array.

Lemma 8. *The previous algorithm has time complexity $O(n^2(k{+}1)^n)$ and space complexity $O((k+1)^n)$ where k is the size of the alphabet and n is the order of the input de Bruijn sequence.*

Proof. To calculate the space complexity observe that there are two big arrays. The $visitedVertices$ array has size $(k{+}1)^{n-1}$ since it has a slot for each vertex of the $(k{+}1)$-sized alphabet. But the actual output, the $B(k+1,n)$ sequence, will grow up to size $(k+1)^n$. To calculate the time complexity of the main cycle observe that we iterate $(k+1)^n$ times, which is the number of edges for the increased alphabet and also the final size of the sequence array. Then, for each vertex there can be a cycle to add. Adding a cycle has time complexity $O(n^2)$. This is because we iterate through the edges of the cycle (up to n edges) and for each of those edges we check for the equality of words of size n. Then the main cycle has time complexity $O(n^2(k+1)^n)$. □

4. FAIR DISTRIBUTION OF THE NEW SYMBOL

We have shown a mechanism that given an Eulerian cycle c in the $G(k,n)$ graph creates an Eulerian cycle c' in the graph $G(k+1,n)$ with the property that c' preserves the order of the edges in c. This is achieved by placing petals of the augmenting graph $D(k+1,n)$ on each vertex of $G(k,n)$. Remember that each vertex in $G(k,n)$ has k incoming and k outgoing edges. That means that we have $k+1$ options to place a petal for each vertex in the Eulerian cycle. Note that only petals have edges labelled with the new symbol s and no edge in $G(k,n)$ is labelled with s. So in order to have a fair distribution of the symbol s we need to interleave each petal in the an appropriate part of the cycle. This motivates the following definition.

Definition 9 (section of a cycle)**.** Given an Eulerian cycle $c = e_1 \to e_2 \to \cdots \to e_n$ in $G(k,n)$, the *section* j of c is a list of vertices of c composed by the head of each edge e_i of c such that $\lfloor i/k \rfloor = j$.

Note that a $G(k,n)$ de Bruijn graph g has k^n vertices and k^{n+1} edges, so a cycle in g has k^n sections with k vertices each section. Given that there are the same number of sections and vertices, we would like to choose one vertex from each section to place the petal in a way that every vertex is used exactly once. Notice that each section has k vertices and each vertex in g belongs to k sections, not necessarily different.

Definition 10 (Petals Distribution graph)**.** Given an Eulerian cycle c in a $G(k,n)$ de Bruijn graph g, the *Petals Distribution* graph $PD(k,n)$ is a k-regular bipartite graph in which the vertices of g and the sections of c are the two vertex classes and the edges of $PD(k,n)$ are the set of (v,j) such that the vertex v belongs to the section j.

Given a graph G, a *matching* M in G is a set of edges such that no two edges share a common vertex. A vertex is *matched* if it is an endpoint of one of the edges in the matching. A *perfect matching* is a matching which matches all vertices of the graph.

Lemma 11. *For every Petals Distribution graph there is a perfect matching.*

Proof. Let G be a finite bipartite graph with bipartite sets X and Y. For a set W of vertices in X, let $NG(W)$ denote the neighborhood of W in G, that is, the set of all vertices in Y adjacent to some element of W. Hall's marriage theorem [5] states that there is a matching that entirely covers X if and only if for every subset W of X, $|W| \leq |NG(W)|$. Let X be the set of vertices of the original graph and Y the set of vertices for the sections. Observe that for any W such that $|W| = r$, the sum of the degrees of the r vertices is rk. Given that the degree for any vertex in Y is k, we have that $|NG(W)| \geq r$. Then there is a matching that entirely covers X. Furthermore, as $|X| = |Y|$, the matching is perfect. □

In order to compute the perfect match in a Petals Distribution graph we can use any method for computing the maximum flow in a network. We introduce two vertices s and t for the source and sink and add an edge from s to each vertex of X and an edge from each

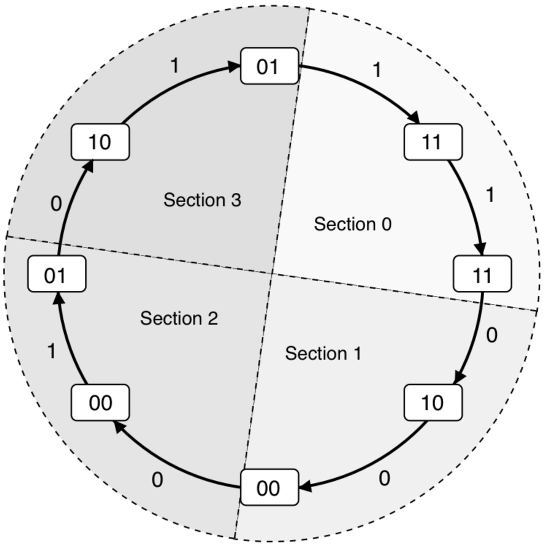

Fig. 4.1: Given the de Bruijn sequence [11000101] there are four sections: the section 0 has the vertex 11 twice, the section 1 has the vertices 10 and 00, the section 2 has the vertices 00 and 01, and the section 3 has the vertices 10 and 01.

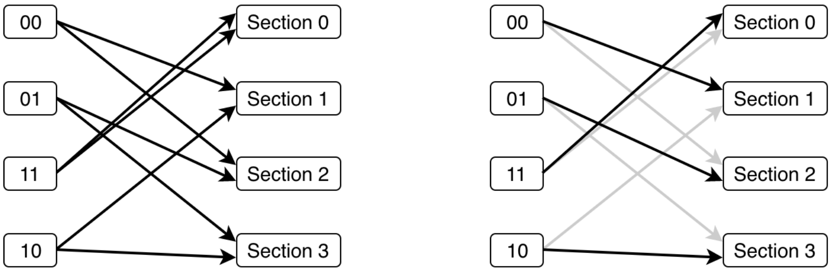

Fig. 4.2: The Petals Distribution graph for the de Bruijn sequence [11000101]. The left figure shows the possible sections for each vertex. The right figure shows a possible assignment of those vertices and sections.

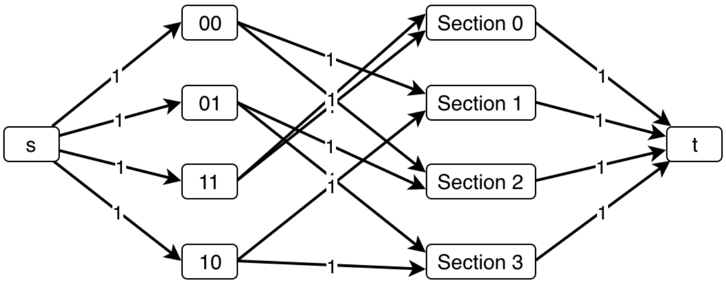

Fig. 4.3: Flow network for PD(2,2) where each edge has capacity 1.

vertex of Y to t. We assign capacity 1 to each of the edges of the flow network. We can see that the maximum flow of the network is $|X|$, so this flow has the edges of a perfect match.

Lemma 12. *Given a de Bruijn word v in $B(k, n)$, for any $2k+n-1$ consecutive symbols in the new sequence w there is at least one occurrence of s.*

Proof. First note that each section v has k vertices. Two petals can be at most $2k-1$ edges away, which corresponds to placing one petal on the first vertex of one section and another in the last vertex of the next section. Also remember that for any given vertex $v = a_1a_2 \dots a_{n-1}$ of $G(k, n-1)$ in $D(k+1, n-1)$ the outgoing edge is labelled with the new symbol s and that determines a cycle associated with the circular word $a_1a_2 \dots a_{n-1}s$. In consequence, the tail vertex of the last edge in that cycle is $sa_1a_2 \dots a_{n-2}$. This means that there is an edge labelled s exactly n edges before the end of the petal. In consequence, between the last occurrence of s in a petal and the first occurrence of s in the next petal there can be at most $2k+n-2$ edges.

Also note that the vertices of the petals have at least one symbol s, therefore we are guaranteed that given any $n-1$ consecutive edges of a petal there is at least one occurrence of s. □

5. PROOF OF THEOREM 1

Algorithm 2 takes a $B(k,n)$ de Bruijn word v and returns a $B(k+1,n)$ de Bruijn word w such that v is a subsequence of w, the new symbol s occurs $(k+1)^{n-1}$ times in w and given any $2k+n-1$ consecutive symbols in w there is at least one occurrence of s. This algorithm is similar to the Algorithm 1, but balances the occurrences of the new symbol s. For this purpose, we have to find a maximum flow for the Petals Distribution graph. We use the Edmonds-Karp algorithm as described before to determine a vertex from each section to start a petal. Then we store that in the $vertexForSection$ array.

In addition to the steps of Algorithm 1, we now keep track of the position in the original sequence, that means, how many edges of the original sequence we have already traversed. That is used to determine which is the actual section and therefore what petal should be placed next.

In the main loop of Algorithm 2 we iterate through every edge of the original sequence adding cycles. On each vertex we check if we can add a cycle. But in this case, if the current vertex belongs to the original graph then adding a cycle implies starting a petal. For that reason, in those cases we have to check if the current vertex can start a petal for the current section, otherwise we do not add the cycle. If the vertex does not belong to the original graph then to add a cycle we just have to check that such cycle has not been already used, because we are not starting a petal. The rest of the algorithm works the same way as the Algorithm 1.

Lemma 13. *For an input $B(k,n)$ circular de Bruijn word the Algorithm 2 produces a $B(k+1,n)$ circular de Bruijn word performing at most $O(k^{3n-2})$ operations and using $O((k+1)^n)$ space.*

Proof. Note that the space complexity of Algorithm 2 is the same as the one for Algorithm 1 given that the only addition in space is the $vertexForSection$ array that has size k^{n-1}, which is smaller than $visitedVertices$. Regarding time complexity note that the search of the maximum flow is the most expensive operation of the algorithm. To see this remember that Edmonds-Karp algorithm has running time $O(V^2E)$, see [4, 8]. In our case, the vertices of the flow graph are the vertices of the original de Bruijn graph and the section vertices, so $V = 2k^{n-1}$. Also note that there are $k+2$ edges in the flow graph associated to each vertex of the original de Bruijn graph. So $E = (k+2) * k^{n-1}$ and then the Edmonds-Karp time complexity is

$$O((2k^{n-1})^2 * (k+2) * k^{n-1}) = O(k^{3n-2}).$$

This is higher than the main cycle time complexity

$$O(n^2(k+1)^n).$$

This completes the proof. □

Algorithm 2

```
function extendDeBruijn(originalSequence: [Int])
    let alphabetSize = getSize(originalSequence)
    let newAlphabetSize = alphabetSize + 1
    let order = getOrder(originalSequence)
    let newSymbol = alphabetSize
    let vertexForSection = EdmondsKarp(originalSequence)
    var sequence = originalSequence
    var originalSequencePos = 0
    var pos = 0
    var vertex = originalSequence.last(order - 1)
    var visitedVertices = [false, ...]
    while pos $\leq$ sequence.count do
        let edgeValue = sequence[pos]
        originalSequencePos += (vertex + [edgeValue]).contains(newSymbol) ? 0 : 1
        vertex = vertex.last(vertex.count - 1) + [edgeValue]
        pos += 1
        let section = floor(originalSequencePos/alphabetSize)
        let shouldStartPetal = !vertex.contains(newSymbol) && vertex == vertexForSection[section] && !visitedVertices[vertex]
        let shouldAddcycle = vertex.contains(newSymbol) && !visitedVertices[vertex]
        if shouldStartPetal || shouldAddcycle then
            let edge = vertex + [newSymbol]
            var cycle = []
            for for i in 0..<edge.count do
                let newEdge = edge.last(edge.count - i) + edge.first(i)
                if i > 0 && edge == newEdge then
                    break
                end if
                let newVertex = newEdge.first(order-1)
                cycle += newEdge.last(1)
                if newEdge.last == newSymbol then
                    visitedVertices[newVertex] = true
                end if
            end for
            sequence = sequence.first(pos) + cycle + sequence.last(sequence.count - pos)
        end if
    end while
    return sequence
end function
```

BIBLIOGRAPHY

[1] Verónica Becher and Pablo Ariel Heiber. On extending de Bruijn sequences. *Information Processing Letters*, 111(18):930–932, 2011.

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

[3] Nicolaas G. de Bruijn. A combinatorial problem. *Nederl. Akad. Wetensch., Proc.*, 49:758–764 = Indagationes Math. 8, 461–467 (1946), 1946.

[4] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[5] Philip Hall. On representatives of subsets. *Lond. Math. Society*, 10, 1935.

[6] Camille Flye Sainte-Marie. Question 48. *L'interm. des math.*, 1:107–110, 1894.

[7] Gabriel Thibeault. Greatest de bruijn sequences in many colors. Tesis de Licenciatura en Ciencias de la Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. Dirección: Verónica Becher, ongoing, 2018.

[8] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms.* MIT Press, 2009.