Ranking and unranking bordered and unbordered words

Daniel Gabric
Department of Math/Stats
University of Winnipeg
Winnipeg, MB R3B 2E9
Canada
d.gabric@uwinnipeg.ca

Abstract

A *border* of a word w is a word that is both a non-empty proper prefix and suffix of w. If w has a border, then it is said to be *bordered*; otherwise, it is said to be *unbordered*. The main results of this paper are the first algorithms to rank and unrank length-n bordered and unbordered words over a k-letter alphabet. We show that, under the unit-cost RAM model, ranking bordered and unbordered words can be done in $O(kn^3)$ time using $O(n)$ space, and unranking them can be done in $O(n^4 k \log k)$ time using $O(n)$ space.

Keywords— Ranking, unranking, bordered words, unbordered words, bifix-free words

1 Introduction

A word u is said to be a *border* of a word w if u is a non-empty proper prefix and suffix of w. If w has a border, then it is said to be *bordered*; otherwise, it is said to be *unbordered*. For example, the word `alfalfa` has `a` and `alfa` as borders, so it is bordered. The word `unbordered` has no borders, so it is unbordered. Bordered and unbordered words are important combinatorial objects in computer science. Borders are fundamental to the most efficient string searching algorithms [1, 2]. Borders are also useful for figuring out statistics on words [3, 4]. Additionally, unbordered words appear as optimal sync words in frame synchronization [5, 6].

In computer science, three problems often arise when a particular combinatorial object is discovered. We want to know how to efficiently count, exhaustively generate, and randomly generate distinct instances of the object. Nielsen [7] gave a recurrence to count the number $u_k(n)$ of length-n unbordered words over a k-letter alphabet. Since every word is either bordered or unbordered, the number of length-n bordered words over a k-letter alphabet is $k^n - u_k(n)$. Using his recurrence he showed that there are $\Theta(k^n)$ length-n bordered and unbordered words. Nielsen also showed how to exhaustively generate all bordered and unbordered words using a recursive procedure. This procedure requires $O(n)$ time per length-n word generated. The problem of efficiently randomly generating bordered and unbordered words is open.

One way of randomly generating an instance of a combinatorial object is with the use of ranking and unranking algorithms. Let $o_1, o_2, \ldots, o_m$ be an ordered list of distinct combinatorial objects. The *rank* of o_i is i, the position of o_i within the list. A ranking algorithm for the list computes i given o_i. An unranking algorithm for the list computes o_i given i. From here it is easy to see that one can randomly generate one of the objects in the list by randomly generating a rank between 1 and m and then unranking the object at that rank. A naïve way to rank and unrank is to generate the list of objects and then determine the position of a particular object. However, this is not ideal, especially if the number of distinct objects is exponential. Thus, we want efficient ranking and unranking algorithms. By efficient we mean that the ranking and unranking algorithms run in $o(f(n))$ time using $o(f(n))$ space where $f(n)$ is the number of distinct objects of "order" n.

Efficient ranking and unranking algorithms have been discovered for many different combinatorial objects, such as permutations [8, 9], trees [10, 11, 12, 13], and necklace variations [14, 15, 16, 17]. See [18] for a more general survey on the topic of ranking and unranking algorithms. In this paper, we consider bordered and unbordered words. We present the first efficient ranking and unranking algorithms for bordered and unbordered words in lexicographic order.

The rest of the paper is structured as follows. In Section 2 we introduce some definitions and results that are necessary to prove our main results. In Section 3 we present a recurrence that will serve as the basis for our ranking algorithms. In Section 4 we present our ranking algorithms for bordered and unbordered words. In Section 5 we show how to unrank bordered and unbordered words. In Section 6 we give some concluding remarks and an open problem. For all the algorithms presented in this paper, the unit-cost RAM model is assumed. Under this computational model, integer variables use constant space and integer arithmetic takes constant time. A complete C implementation of our ranking and unranking algorithms can be found here: https://github.com/DanielGabric/RankUnrank.

2 Preliminaries

Let Σ_k denote the alphabet $\{1, 2, \ldots, k\}$ where $1 < 2 < \cdots < k$. Let Σ_k^n denote the set of all length-n words over Σ_k. Let $x = x_1x_2\cdots x_m \in \Sigma_k^m$ and $y = y_1y_2\cdots y_n \in \Sigma_k^n$. Then $x < y$ in *lexicographic order* either if x is a prefix of u or if $x_i < y_i$ for the smallest i such that $x_i \neq y_i$.

Let $w = w_1w_2\cdots w_n$ be a length-n word. The *unbordered prefix indicator* $a[1..n]$ of w is a length-n integer array such that $a[i] = 1$ if $w_1w_2\cdots w_i$ is unbordered and $a[i] = 0$ otherwise. The *border indicator* $b[1..n]$ of w is a length-n integer array such that $b[i] = 1$ if w has a border of length i and $b[i] = 0$ otherwise. For example, consider the binary word $w = 011101110$. The unbordered prefix indicator of w is $a[1..9] = [1, 1, 1, 1, 0, 0, 0, 0, 0]$. The border indicator of w is $b[1..9] = [1, 0, 0, 0, 1, 0, 0, 0, 0]$.

Lemma 1. *Let $n \geq 1$ be an integer. Let $w = w_1w_2\cdots w_n$ be a word of length n. Then the unbordered prefix indicator of w can be computed in $O(n)$ time.*

Proof. Recall the failure function of a word, also known as the longest prefix suffix array, from the Knuth-Morris-Pratt string searching algorithm [1]. The failure function of w is a length-n integer array $A[1..n]$ such that $A[i] = j$ if and only if the longest proper prefix of $w_1w_2\cdots w_i$ that matches a suffix of $w_1w_2\cdots w_i$ is of length j. Thus $A[i] = 0$ if and only if $w_1w_2\cdots w_i$ is unbordered. Since the failure function encodes the longest borders of all prefixes of a word, we call it the *Prefix Border Array* (PBA). Let PBA$[1..n]$ be the PBA array of the word w. One can easily obtain the

unbordered prefix indicator of w from PBA$[1..n]$ since PBA$[i] = 0$ implies $w_1 w_2 \cdots w_i$ is unbordered. Knuth, Morris, and Pratt [1] showed that PBA$[1..n]$ can be computed in $O(n)$ time. Therefore, the unbordered prefix indicator of w can also be computed in $O(n)$ time. □

Lemma 2. *Let $n \geq 1$ be an integer. Let w be a word of length n. Then the border indicator of w can be computed in $O(n)$ time.*

Proof. A length-n word $w = w_1 w_2 \cdots w_n$ is said to have a *period* p if $w_i = w_{i+p}$ for all $1 \leq i \leq n-p$. The *auto-correlation* [3, 4] of w is a binary word $a_1 a_2 \cdots a_n$ such that $a_i = 1$ if and only if i is a period w. It is easy to see that a length-n word has a period p if and only if it has a border of length $n - p$. Thus one can easily obtain the border indicator of w from the auto-correlation of w. Corollary 2 in [19] shows how to compute the auto-correlation of a length-n word in $O(n)$ time using the failure function from the Knuth-Morris-Pratt string searching algorithm. Therefore, we can also compute the border indicator of w in $O(n)$ time. □

The following two lemmas will serve as the basis for the recurrence in Section 3.

Lemma 3. *Let w be a length-n bordered word. Let u be a border of w. Then u is the shortest border of w if and only if u is unbordered.*

Proof. We prove the contrapositive of both directions. If u is bordered, then the border of u is a shorter border of w. If u is not the shortest border, then there exists a shorter border u' of w. But this border is now both a non-empty proper prefix and suffix of u, so u is bordered. □

Lemma 4. *Let w be a length-n bordered word. Let u be the shortest border of w. Then $|u| \leq n/2$.*

Proof. Suppose to the contrary, that $|u| > n/2$. Then u must overlap itself within w. So u is bordered. But this border of u must also be a border of w. This contradicts the assumption that u is the shortest border of w. Thus $|u| \leq n/2$. □

3 Recurrence

In this section we give a recurrence for the number of bordered words with a given prefix. This is the basis for our ranking algorithm. Let $n \geq 1$ be an integer. Let u be a word of length less than or equal to n. Let $B_k(u, n)$ denote the number of length-n bordered words over Σ_k that have u as a prefix.

Theorem 5. *Let $n \geq p \geq 1$ and $k \geq 2$ be integers. Let u be a length-p word over Σ_k. Let $b[1..p]$ be the border indicator of u. Let $a[1..p]$ be the unbordered prefix indicator of u. Then*

$$B_k(u,n) = \begin{cases} \sum_{i=1}^{n-p} a[i]k^{n-p-i} + \sum_{i=n-p+1}^{\lfloor n/2 \rfloor} a[i]b[i-(n-p)], & \text{if } n \leq 2p; \\ \sum_{i=1}^{p} a[i]k^{n-p-i} + \sum_{i=p+1}^{\lfloor n/2 \rfloor} (k^{i-p} - B_k(u,i))k^{n-2i}, & \text{otherwise.} \end{cases}$$

Proof. Let w be a length-n bordered word. Suppose that u is a prefix of w. Let v be the shortest border of w. Our strategy is to split up the set of all length-n bordered words with u as a prefix into sets, $S_1, \ldots, S_n$, such that $\sum_{i=1}^{n} |S_i| = B_k(u, n)$. We choose S_i to be the set of all length-n bordered words that have u as a prefix and have a length-i shortest border. These sets are clearly disjoint, and their union is just the set of all length-n bordered words with u as a prefix. By Lemma 3 and Lemma 4 we have that v is unbordered and $|v| \leq n/2$. Therefore, we have $|S_i| = 0$ for $i > n/2$. Since v is unbordered, we have $a[|v|] = 1$ when $|v| \leq |u| = p$. There are two cases to consider. The first case is when $|u| = p$ is greater than or equal to half the length of w, or $2p \geq n$. The other case is when $2p < n$.

Suppose $2p \geq n$. Since v is unbordered and $|v| \leq n/2 \leq p$, we have that v must occur as an unbordered prefix of u. We further split this case into two subcases, one where $|v| \leq n - p$ and one where $n - p + 1 \leq |v| \leq n/2$. If $|v| \leq n - p$, then $w = vxyv$ where $u = vx$ and y is a word of length $n - p - |v|$. Since v and x are determined by u, we have that there are $k^{n-p-|v|}$ choices for w. If $n - p + 1 \leq |v| \leq n/2$, then the instance of v that is a suffix of w must also have a non-empty overlap with u. So $w = vxyz$ where x is a possibly empty word and y, z are non-empty words such that $u = vxy$ and $v = yz$. But this means that $u = yzxy$. Thus u must have a border of length $|y| = |v| - |z| = |v| - (n - p)$. So w is bordered if and only if $b[|v| - (n - p)] = 1$. When summing over all possible prefixes v for the case when $2p \geq n$, we use the unbordered indicator $a[1..p]$ to only include those v that are unbordered. So we get that

$$B_k(u, n) = \sum_{i=1}^{n-p} a[i]k^{n-p-i} + \sum_{i=n-p+1}^{\lfloor n/2 \rfloor} a[i]b[i - (n - p)].$$

Suppose $2p < n$. Again, we further split this case into two subcases, one where $|v| \leq p$, and one where $p + 1 \leq |v| \leq n/2$. If $|v| \leq p$, then v must occur as an unbordered prefix of u. As in the previous case, this leads to there being $k^{n-p-|v|}$ choices for w. So suppose $p + 1 \leq |v| \leq n/2$. In this case, we have that u must occur as a prefix of v. So this means that v is unbordered and has u as a prefix. The number of such words v is clearly $k^{|v|-p} - B_k(u, |v|)$ (i.e., all length-$|v|$ bordered words with u as a prefix subtracted from all length-$|v|$ words with u as a prefix). We can write $w = vxv$ where x is a word of length $n - 2|v|$. There are $k^{|v|-p} - B_k(u, |v|)$ choices for v and $k^{n-2|v|}$ choices for x. Therefore, there are $(k^{|v|-p} - B_k(u, |v|))k^{n-2|v|}$ choices for w in this case. Summing over all prefixes v for the case $2p < n$, and using the unbordered indicator $a[1..p]$ to only include those v that are unbordered and are of length less than or equal to $|u| = p$, we get

$$B_k(u, n) = \sum_{i=1}^{p} a[i]k^{n-p-i} + \sum_{i=p+1}^{\lfloor n/2 \rfloor} (k^{i-p} - B_k(u, i))k^{n-2i}.$$

□

Theorem 6. *Let $n \geq 1$. Let u be a word of length less than or equal to n. The recurrence $B_k(u, n)$ can be computed in $O(n^2)$ time using $O(n)$ space.*

Proof. Before starting the computation of $B_k(u, n)$, we first need to compute the border indicator of u and the unbordered prefix indicator of u. From Lemma 1 and Lemma 2 we see that both the border indicator and unbordered prefix indicator of u can be computed in $O(n) \in O(n^2)$ time. For each $i \geq |u|$, we have that calculating $B_k(u, i)$ involves computing a sum of $O(i)$ terms. Thus, using standard dynamic programming techniques, we can compute $B_k(u, n)$ in $O(n^2)$ time using $O(n)$ space. □

4 Ranking

In this section we show how to efficiently calculate the ranks of bordered and unbordered words. Let $\text{rankB}_k(w)$ (resp. $\text{rankU}_k(w)$) denote the rank of the word w in the lexicographic listing of bordered (resp. unbordered) words of length $|w|$ over the alphabet Σ_k.

Theorem 7. *Let $n \geq 1$ and $k \geq 2$ be integers. Let $w = w_1 w_2 \cdots w_n$ be a length-n bordered word over Σ_k. Then*

$$\text{rankB}_k(w) = 1 + \sum_{i=1}^{n} \sum_{c=1}^{w_i - 1} B_k(w_1 w_2 \cdots w_{i-1} c, n).$$

Proof. The rank of w is just the position of w in the lexicographic listing of all length-n bordered words. This is equal to 1 plus the number of length-n bordered words that are smaller than w in lexicographic order. By definition, for any word $u = u_1 u_2 \cdots u_n$ that is smaller than w, there exists an i such that $u_1 u_2 \cdots u_{i-1} = w_1 w_2 \cdots w_{i-1}$ and $u_i < w_i$. Thus, any length-n bordered word that begins with $w_1 w_2 \cdots w_{i-1} c$ for some $c < w_i$ is smaller than w. Summing over all possible i and $c < w_i$ we have that the number of length-n bordered words that are smaller than w is

$$\sum_{i=1}^{n} \sum_{c=1}^{w_i - 1} B_k(w_1 w_2 \cdots w_{i-1} c, n).$$

□

Theorem 8. *Let $n \geq 1$ and $k \geq 2$ be integers. Let $w = w_1 w_2 \cdots w_n$ be a length-n unbordered word over Σ_k. Then*

$$\text{rankU}_k(w) = 2 + \sum_{i=1}^{n} (w_i - 1) k^{n-i} - \text{rankB}_k(w).$$

Proof. This proof follows the same structure as the proof of Theorem 7. The rank of w is equal to 1 plus the number of length-n unbordered words that are smaller than w. By definition, a length-n word $u = u_1 u_2 \cdots u_n$ is smaller than w if a there exists an i such that $u_1 u_2 \cdots u_{i-1} = w_1 w_2 \cdots w_{i-1}$ and $u_i < w_i$. Therefore, any length-n unbordered word that begins with $w_1 w_2 \cdots w_{i-1} c$ for $c < w_i$ is smaller than w. The number of length-n unbordered words that begin with $w_1 w_2 \cdots w_{i-1} c$ is equal to the number length-n bordered words that begin with $w_1 w_2 \cdots w_{i-1} c$ subtracted from all length-n words that begin with $w_1 w_2 \cdots w_{i-1} c$. This is just equal to $k^{n-i} - B_k(w_1 w_2 \cdots w_{i-1} c, n)$. Summing over all possible i and $c < w_i$, we have that the number of length-n unbordered words that are smaller than w is

$$\begin{aligned}\sum_{i=1}^{n} \sum_{c=1}^{w_i - 1} (k^{n-i} - B_k(w_1 w_2 \cdots w_{i-1} c, n)) &= \sum_{i=1}^{n} \sum_{c=1}^{w_i - 1} k^{n-i} - \sum_{i=1}^{n} \sum_{c=1}^{w_i - 1} B_k(w_1 w_2 \cdots w_{i-1} c, n) \\ &= 1 + \sum_{i=1}^{n} (w_i - 1) k^{n-i} - \text{rankB}_k(w).\end{aligned}$$

□

Algorithm 1 Computing $\text{rankB}_k(w)$ and $\text{rankU}_k(w)$ given w and k.

```
1: function RANKB(w = w_1 w_2 ··· w_n, k)
2:     total ← 0
3:     for i ← 1 to n do
4:         for c ← 1 to w_i − 1 do
5:             total ← total + B_k(w_1 w_2 ··· w_{i−1} c, n)
6:     return 1 + total

7: function RANKU(w = w_1 w_2 ··· w_n, k)
8:     total ← 0
9:     for i ← 1 to n do
10:        total ← total + (w_i − 1) k^{n−i}
11:    return 2 + total − RANKB(w)
```

Suppose w is of length n. While computing $\text{rankB}_k(w)$ in Algorithm 1, we have that for every index i of w, we loop through $w_i - 1 \in O(k)$ different symbols. For each of these symbols we compute $B_k(u, n)$ for some word u of length i, which takes at most $O(n^2)$ time using $O(n)$ space by Theorem 6. Thus, we can compute $\text{rankB}_k(w)$ in $O(kn^3)$. Since we are only required to store the length-n word w, an integer variable which stores the rank, and a single length-n integer array that we reuse to calculate $B_k(u, n)$, we have that computing $\text{rankB}_k(w)$ uses only $O(n)$ space. Since $\text{rankU}_k(w)$ uses $\text{rankB}_k(w)$ and does only $O(n)$ extra work, it also runs in $O(kn^3)$ time using $O(n)$ space.

Theorem 9. *Let $n \geq 1$ and $k \geq 2$ be integers. Let w be a length-n word over Σ_k. Then* $\text{rankB}_k(w)$ *can be computed in $O(kn^3)$ time using $O(n)$ space.*

Theorem 10. *Let $n \geq 1$ and $k \geq 2$ be integers. Let w be a length-n word over Σ_k. Then* $\text{rankU}_k(w)$ *can be computed in $O(kn^3)$ time using $O(n)$ space.*

5 Unranking

In this section we show how to unrank bordered and unbordered words. We describe the algorithm to compute the length-n bordered word with rank r. The same algorithm can be adapted to unrank unbordered words as well.

Suppose $w = w_1 w_2 \cdots w_n$ is a length-n bordered word with rank r that we are trying to determine. First, we initialize w to be 1^n, the lexicographically smallest length-n word. Then, starting with $i = 1$, we determine w_i by applying binary search on $1, 2, \ldots, k$ to find the largest x such that the rank of $w_1 w_2 \cdots w_{i-1} x 1^{n-j}$ is less than or equal to r. Then we set $w_i = x$, and move on to w_{i+1}.

The reason we are choosing the largest x is due to the fact that rankB_k can be applied to both bordered and unbordered words. It is easy to show that the largest word with a given rank is the desired bordered word at that rank. Observe that when unranking unbordered words, choosing the largest word at a particular rank can output a bordered word if the rank is larger than the number of unbordered words. In this case, the output is $kk \cdots k$.

Algorithm 2 Unranking bordered words given a rank r, a length n, alphabet size k.

```
1: function UNRANKB(r, n, k)
2:     w_1 w_2 ··· w_n ← 11 ··· 1
3:     for i ← 1 to n do
4:         left ← 1
5:         right ← k
6:         while left < right do
7:             save ← w_i
8:             mid ← ⌈(left + right)/2⌉
9:             w_i ← mid
10:            if RANKB(w_1 w_2 ··· w_n, k) ≤ r then left ← mid
11:            else
12:                w_i ← save
13:                right ← mid
14:    return w_1 w_2 ··· w_n
```

For every index i of w we perform a binary search on the alphabet of size k. For each iteration in the binary search we calculate $\text{rankB}_k(w)$. There are n indices of w, binary search on the alphabet of size k takes $O(\log k)$ time, and calculating $\text{rankB}_k(w)$ takes $O(kn^3)$ time. Thus, running $\text{UnRankB}(r, n, k)$ takes $O(n^4 k \log k)$ time. Running $\text{UnRankB}(r, n, k)$ also takes $O(n)$ space since it only requires us to use a length-n word, and a constant number of integer variables. Additionally, when calculating $\text{rankB}_k(w)$ within the unranking procedure we can reuse a length-n integer array to store $B_k(u, n)$. To adapt the algorithm to unrank unbordered words, just replace $\text{RANKB}(w, k)$ with $\text{RANKU}(w, k)$.

Theorem 11. *Let $n \geq 1$ and $k \geq 2$ be integers. A bordered word $w \in \Sigma_k^n$ with rank r can be computed in $O(n^4 k \log k)$ time using $O(n)$ space.*

Theorem 12. *Let $n \geq 1$ and $k \geq 2$ be integers. An unbordered word $w \in \Sigma_k^n$ with rank r can be computed in $O(n^4 k \log k)$ time using $O(n)$ space.*

6 Conclusions and Open Problems

In this paper we presented the first efficient ranking and unranking algorithms for bordered and unbordered words. We gave a $O(kn^3)$ ranking algorithm that uses only $O(n)$ space for bordered and unbordered words. We also showed how to use this ranking algorithm to unrank bordered and unbordered words in $O(n^4 k \log k)$ time using $O(n)$ space. We conclude by posing an open problem.

- Can ranking and unranking bordered and unbordered words be done more efficiently? Can they be done in quadratic time?
  - It seems believable that the ranking and unranking algorithms in this paper can be improved by a factor of n. We briefly lay out the reasoning. As is, the recurrence $B_k(u, n)$ takes $O(n^2)$ time to determine. However, if we compute $B_k(u, n) - kB_k(u, n-1)$, all but a constant number of terms cancel out in the case that $n > 2|u|$. In the case

that $n \leq 2|u|$, we are still left with $O(n)$ terms in the summation. If this summation could be computed in constant time, then computing $B_k(u, n)$ would take $O(n)$ time, effectively reducing the time complexity of ranking and unranking by a factor of n.

References

[1] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[2] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[3] L. J. Guibas and A. M. Odlyzko. Periods in strings. *J. Combin. Theory Ser. A*, 30(1):19–42, 1981.

[4] L. J. Guibas and A. M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *J. Combin. Theory Ser. A*, 30(2):183–208, 1981.

[5] J. Massey. Optimum frame synchronization. *IEEE Trans. Commun.*, 20(2):115–119, 1972.

[6] R. Scholtz. Frame synchronization techniques. *IEEE Trans. Commun.*, 28(8):1204–1213, 1980.

[7] P. T. Nielsen. A note on bifix-free sequences. *IEEE Trans. Inform. Theory*, IT-19:704–706, 1973.

[8] W. Myrvold and F. Ruskey. Ranking and unranking permutations in linear time. *Inform. Process. Lett.*, 79(6):281–284, 2001.

[9] M. Mareš and M. Straka. Linear-time ranking of permutations. In L. Arge, M. Hoffmann, and E. Welzl, editors, *Algorithms – ESA 2007*, volume 4698 of *Lecture Notes in Computer Science*, pages 187–193, Berlin, Heidelberg, 2007. Springer.

[10] J. M. Pallo. Enumerating, ranking and unranking binary trees. *Computer J.*, 29(2):171–175, 01 1986.

[11] L. Li. Ranking and unranking of AVL-trees. *SIAM J. Comput.*, 15(4):1025–1035, 1986.

[12] U. Gupta, D. T. Lee, and C. K. Wong. Ranking and unranking of B-trees. *J. Algorithms*, 4(1):51–60, 1983.

[13] U. Gupta, D. T. Lee, and C. K. Wong. Ranking and unranking of 2-3 trees. *SIAM J. Comput.*, 11(3):582–590, 1982.

[14] P. Hartman and J. Sawada. Ranking and unranking fixed-density necklaces and Lyndon words. *Theoret. Comput. Sci.*, 791:36–47, 2019.

[15] J. Sawada and A. Williams. Practical algorithms to rank necklaces, Lyndon words, and de Bruijn sequences. *J. Disc. Alg.*, 43:95–110, 2017.

[16] D. Adamson, V. V. Gusev, I. Potapov, and A. Deligkas. Ranking Bracelets in Polynomial Time. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[17] D. Adamson. Ranking binary unlabelled necklaces in polynomial time. In Yo-Sub Han and György Vaszil, editors, *Descriptional Complexity of Formal Systems*, volume 13439 of *Lecture Notes in Computer Science*, pages 15–29, Cham, 2022. Springer International Publishing.

[18] F. Ruskey. *Combinatorial Generation*. Working version (1j), 2003. https://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf.

[19] D. A. Lind. Perturbations of shifts of finite type. *SIAM J. Disc. Math.*, 2(3):350–365, 1989.