Faster Generation of Shorthand Universal Cycles for Permutations

Alexander Holroyd[1], Frank Ruskey[2⋆], and Aaron Williams[2]

[1] Dept. of Computer Science, University of Victoria, CANADA
[2] Microsoft Research, Redmond, WA, USA

Abstract. A universal cycle for the k-permutations of $\langle n\rangle = \{1, 2, ..., n\}$ is a circular string of length $(n)_k$ that contains each k-permutation exactly once as a substring. Jackson (Discrete Mathematics, 149 (1996) 123–129) proved their existence for all $k \leq n-1$. Knuth (*The Art of Computer Programming, Volume 4*, Fascicle 2, Addison-Wesley, 2005) pointed out the importance of the $k = n-1$ case, where each $(n-1)$-permutation is "shorthand" for exactly one permutation of $\langle n\rangle$. Ruskey-Williams (ACM Transactions on Algorithms, in press) answered Knuth's request for an explicit construction of a shorthand universal cycle for permutations, and gave an algorithm that creates successive symbols in worst-case $O(1)$-time. This paper provides two new algorithmic constructions that create successive blocks of n symbols in $O(1)$ amortized time within an array of length n. The constructions are based on: (a) an approach known to bell-ringers for over 300 years, and (b) the recent shift Gray code by Williams (SODA, (2009) 987-996). For (a), we show that the majority of changes between successive permutations are full rotations; asymptotically, the proportion of them is $(n-2)/n$.

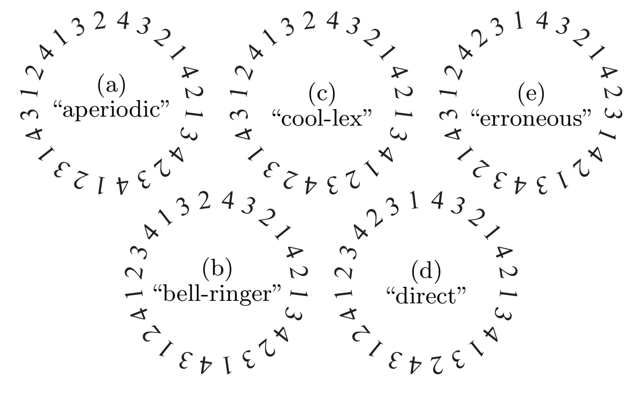


Fig. 1. (a)-(d) are Ucycles for $\Pi_3(4)$, or equivalently shorthand Ucycles for $\Pi(4)$. The symbol 4 is periodic in (b)-(e) since it is in every nth position. (e) is not a Ucycle for $\Pi_3(4)$ due to two types of errors: an erroneous substring 242, and an extra copy of 142.

⋆ Research supported in part by NSERC

1 Introduction

A *universal cycle* (or *Ucycle*) [1] is a circular string containing every object of a particular type exactly once as substring. For example, consider the circular string in Figure 1(a). Starting from 12 o'clock, and proceeding clockwise, its substrings of length three are

$$432, 321, 214, 142, \ldots, 413, 132, 324, 243. \tag{1}$$

In total there are 24 substrings, and these substrings include every 3-permutation of $\langle 4 \rangle$ exactly once. For this reason, Figure 1(a) is a *Ucycle for the* 3*-permutations of* $\langle 4 \rangle$. Let $\Pi_k(n)$ denote the set of all k-permutations of $\langle n \rangle$. Notice that $|\Pi_k(n)| = (n)_k := n(n-1)\cdots(n-k+1)$ (the *falling factorial*). In the special case $k = n$, we use $\Pi(n)$ to represent the permutations of $\langle n \rangle$. Jackson proved that Ucycles of $\Pi_k(n)$ exist whenever $k < n$ [4]. On the other hand, the reader may easily verify that Ucycles of $\Pi(n)$ do not exist when $n \geq 3$.

This section describes four additional interpretations for Ucycles of $\Pi_{n-1}(n)$, then discusses applications, relevant history, and outlines our new results.

1.1 Interpretations and Applications

For the first interpretation, notice that $|\Pi_{n-1}(n)| = n_{n-1} = n! = |\Pi(n)|$. Each $(n-1)$-permutation of $\langle n \rangle$ extends to a unique permutation of $\langle n \rangle$ by appending its *missing symbol* from $\langle n \rangle$. For example, the first string in (1) is 432, and it is missing the symbol 1. Thus, the substring 432 is *shorthand* for 4321. Similarly, the substrings in (1) are shorthand for the following permutations

$$4321, 3214, 2143, 1423, 4213, \ldots, 2413, 4132, 1324, 3241, 2431. \tag{2}$$

For this reason, Ucycles for $\Pi_{n-1}(n)$ can be interpreted as Ucycles for permutations called *shorthand Ucycles for* $\Pi(n)$. The substrings comprising $\Pi_{n-1}(n)$ are the Ucycle's *substrings*, and the extended strings comprising $\Pi(n)$ are the Ucycles's *permutations*. This leads to the remaining interpretations.

The second interpretation is the *binary representation*. Given a length $n-1$ substring in a shorthand Ucycle for $\Pi(n)$, the *next symbol* is the symbol that follows this substring in the shorthand Ucycle, and the *next substring* is the length $n-1$ substring that ends with this next symbol. That is, if $s_1 s_2 \cdots s_{n-1}$ is a substring in a shorthand Ucycle for $\Pi(n)$, then the next symbol is some $x \in \langle n \rangle$, and the next substring is $s_2 s_3 \cdots s_{n-1} x$. Since $s_2 s_3 \cdots s_{n-1} x$ is in $\Pi_{n-1}(n)$, then there are only two choices for x. Either x is s_1, or x is the missing symbol from $s_1 s_2 \cdots s_{n-1}$. This dichotomy gives rise to the binary representation. The ith bit in the binary representation is 1 if the substring starting at position i has its next symbol equal to its first symbol; otherwise the next symbol equals its missing symbol and the ith bit in the binary representation is 0. The binary representation can be visualized by placing two copies of the shorthand Ucycle for $\langle n \rangle$ above itself, with the second copy left-shifted $(n-1)$ positions. This comparison vertically aligns the first symbol of each length n substring with its next symbol. Accordingly, 1s are recorded precisely when the vertically aligned symbols are equal. This is illustrated below for Figure 1(a), where $\underline{4}$ denotes the first symbol in the shorthand Ucycle (above) and in its rotation (below).

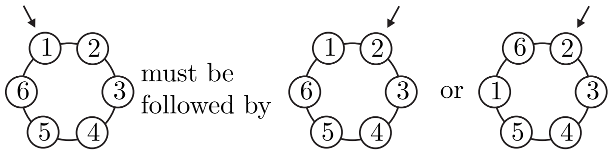
$$\begin{array}{l} \underline{4}32142134234123143124132 \\ 142134234123143124132\underline{4}32 \\ \hline 001100011000101100100011 \end{array}$$

To check the binary string, notice that its first bit is 0. This is because the first substring is 432, and its first symbol 4 does not equal its next symbol 1. On the other hand, the third bit in the binary string is 1. This is because the third substring is 214, and its first symbol 2 equals its next symbol 2. (As above, the shorthand Ucycle for $\Pi(n)$ is assumed to "start" with $n\ (n{-}1)\ \cdots\ 2$.)

The third interpretation is a Gray code for permutations. If $p_1p_2\cdots p_n$ is a permutation in a shorthand Ucycle for $\Pi(n)$, then the *next permutation* begins with $p_2p_3\cdots p_{n-1}$. Therefore, the next permutation is either $p_2p_3\cdots p_{n-1}p_np_1$ or $p_2p_3\cdots p_{n-1}p_1p_n$. These cases are obtained by rotating the first symbol of $p_1p_2\cdots p_n$ into one of the last two positions. More precisely, if the ith bit in the binary representation is 0 or 1, then the ith permutation is transformed into the $(i+1)$st permutation by the rotation $\sigma_n = (1\ 2\ \cdots\ n)$ or $\sigma_{n-1} = (1\ 2\ \cdots\ n{-}1)$, respectively. Thus, permutations in a shorthand Ucycle for $\Pi(n)$ are in a *circular Gray code* using σ_n and σ_{n-1}. For example, the first bit in the binary string representation for (2) is 0, so σ_4 transforms the first permutation in (2), 4321, into the second permutation in (2), 3214. Conversely, every circular Gray code of $\Pi(n)$ using σ_n and σ_{n-1} provides a shorthand Ucycle for $\Pi(n)$ by appending the first symbols of each permutation.

The fourth interpretation is an equivalence to Hamiltonian cycles in the directed Cayley graph on $\Pi(n)$ with generators σ_n and σ_{n-1} (see [7]).

The binary representation provides a natural application for shorthand Ucycles of $\Pi(n)$: $n!$ permutations are encoded in $n!$ bits. The Gray code interpretation also provides applications. The rotations σ_n and σ_{n-1} can also be described respectively as *prefix shifts* of length n and $n-1$. Prefix shifts of length n and $n-1$ can be performed as basic operations within linked lists or circular arrays. In particular, a prefix shift of length n simply increments the starting position within a circular array, whereas a prefix shift of length $n-1$ increments the starting position and then performs an adjacent-transposition of the last two symbols. This is illustrated below with permutation 123456 (below left) being followed by 234561 or 234516 (below right).



1.2 History

Ucycles for combinatorial objects were introduced by Chung, Diaconis, and Graham [1] as natural generalizations of *de Bruijn cycles* [2], which are circular strings of length 2^n that contain every binary string of length n exactly once as substring. They pointed out that Ucycles of $\Pi(n)$ do not exist, and suggested instead using order-isomorphism to represent the permutations. A string is *order-isomorphic* to a permutation of $\langle n \rangle$ if the string contains n distinct integers whose relative orders are the same as in the permutation.

Knuth suggested the use of what we call *shorthand-isomorphism*, and asked for an explicit construction [5]. An explicit construction was discovered by Ruskey-Williams [7], who also provided an algorithm that creates each successive symbol in the Ucycle (or bit in the binary representation) in worst-case O(1)-time. The construction has the property that the symbol n appears in every nth position in the Ucycle. For this reason, it is said to have a *periodic symbol*. See Figure 1(d) for the construction when $n = 4$, and notice that 4 appears in every 4th position. (Figure 1(a) illustrates that shorthand Ucycles of $\Pi(n)$ do not necessarily contains a periodic symbol, and it is also obvious that a shorthand Ucycles of $\Pi(n)$ can contain at most one periodic symbol when $n \geq 3$.)

When a shorthand Ucycle for $\Pi(n)$ has n as its periodic symbol, then its remaining symbols are divided into blocks of length $n-1$. Furthermore, each block must be a distinct permutation of $\langle n-1 \rangle$. Given this situation, the permutations of $\langle n-1 \rangle$ are called the *sub-permutations*. Figure 2 summarizes the substrings, permutations, binary representation, and sub-permutations of Figure 1(d).

substrings: $432, 321, 214, 142, \ldots$
permutations: $4321, 3214, 2143, 1423, \ldots$
binary representation: $001 \cdots$

periodic symbol: 4
sub-permutations: $321, 213, 132, 312, 123, 231$

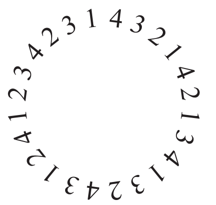


Fig. 2. The direct construction for $n = 4$.

Given this terminology, there is a succinct description of the construction in [7]: The permutations for n are the sub-permutations for $n+1$. For example, the Ucycle for $n = 3$ is 321312, and so the permutations are $321, 213, 132, 312, 123, 231$. Notice that the permutations are identical to the sub-permutations found in Figure 2. For this reason, the construction in [7] can be described as the *direct construction*. The binary representation of the direct construction also has a nice description. If $x_1 x_2 \cdots x_{n!}$ is the binary string representation for n, then

$$001^{n-2}\ \overline{x}_1 001^{n-2}\ \overline{x}_2 \cdots 001^{n-2}\ \overline{x}_{n!} \tag{3}$$

is the binary string representation for $n+1$, where $\overline{x_i} = 1 - x_i$, and 1^{n-2} represents $n-2$ copies of 1. For example, the binary representation for $n = 2$ is

00, and so the binary representation for $n = 3$ and $n = 4$ are $00\,\bar{0}\,00\,\bar{0} = 001001$ and

$$001\,\bar{0}\,001\,\bar{0}\,001\,\bar{1}\,001\,\bar{0}\,001\,\bar{0}\,001\,\bar{1} = 001100110010001100110010.$$

1.3 Our New Approach

This paper expands upon [7] by providing two additional constructions. Our first construction is based on a technique from change-ringing (English-style church bell ringing). The general method (involving "half-hunts", in change-ringing terminology) is more than 300 years old (see [3, 8]), but we believe that our application of it to the Ucycle problem is novel. See Figure 1(b).

The second construction uses the recently discovered shift Gray code of Williams [9], which generalizes *cool-lex order*. Aside from the intrinsic interest of having additional answers to Knuth's query, both of these constructions have two distinct advantages over the previous construction found in [7].

First, the constructions have fewer 1s in the binary representations. The bell-ringer algorithm is particularly advantageous in this regard since the vast majority of bits are 0 (Theorem 3); asymptotically, the proportion of them is $(n-2)/n$.

Second, the resulting algorithms are faster in the following scenario: Suppose an application wants a shorthand Ucycle for the permutations of $\{1, 2, \ldots, n\}$. Since $n!$ is prohibitively large, the shorthand universal is given to the application n symbols at a time within an array of length n. Our new algorithm updates this array in average-case $O(1)$-time. In other words, it provides successive blocks of n symbols in *constant amortized time*. This is an improvement over the previous algorithm [7], which would have required $\Omega(n)$-time to obtain the next n symbols.

Sections 2 and 3 describe these new constructions. In both cases the strategy is to generate a list of $\Pi(n)$ that will become the sub-permutations for a shorthand Ucycle for $\Pi(n+1)$. When proving that the result is in fact a shorthand Ucycle for $\Pi(n+1)$, we use the following definition and notation.

Definition 1. *Let $P = p_1, p_2, \ldots, p_{n!}$ be a list of permutations. By $\mathsf{R}(P)$ we denote the cyclic string $(n{+}1)p_1(n{+}1)p_2 \cdots (n{+}1)p_{n!}$. The list P is* recyclable *if $\mathsf{R}(P)$ is a shorthand universal cycle for the permutations of $\langle n{+}1\rangle$.*

Figure 1(e) illustrates that not every order of $\Pi(n)$ is recyclable. Theorem 1 proves that its two types of "errors" are the only obstacles to being recyclable.

Theorem 1. *A circular list of permutations $P = p_1, p_2, \ldots, p_{n!}$ is recyclable if and only if the following two conditions are satisfied.*

- *If α and β are successive permutations, then $\alpha_i^{-1} - \beta_i^{-1} \leq 1$ for all $i \in \langle n\rangle$.*
- *If α and β are two successive permutations and α' and β' are also two successive permutations, then whenever there is a $j \in \langle n\rangle$ such that*

$$\alpha_{j+1} \cdots \alpha_n \beta_1 \cdots \beta_{j-1} = \alpha'_{j+1} \cdots \alpha'_n \beta'_1 \cdots \beta'_{j-1},$$

then $\alpha = \alpha'$ (and hence $\beta = \beta'$).

Proof. Let $X = \mathsf{R}(P) = (n{+}1)p_1(n{+}1)p_2 \cdots (n{+}1)p_{n!}$. The first condition guarantees that any n successive symbols from X are distinct; that is, they are all n-permutations of $\langle n{+}1 \rangle$. The condition says that no symbol in the successor moves left more than one position (it could move right).

The second condition will guarantee that all of the length n substrings of X are distinct. Since the length of X is obviously $(n+1)!$, this will finish the proof. Let ω be a substring of X of length n. Clearly, if ω does not contain $n{+}1$ then it is distinct, since in that case we must have $\omega = p_i$ for some i. If ω contains $n{+}1$, then $\omega = \alpha_{j+1} \cdots \alpha_n (n{+}1) \beta_1 \cdots \beta_{j-1}$ for some value of $j \in \langle n \rangle$ and successive permutations α and β. If ω is not distinct, then there would be some other two permutations α' and β' such that $\omega' = \alpha'_{j+1} \cdots \alpha'_n (n{+}1) \beta'_1 \cdots \beta'_{j-1}$. □

2 The Bell-Ringer Construction and 7-order

In order to describe the bell-ringing-inspired Ucycle, we introduce an ordering of the permutations of $\langle n \rangle$ that we call "seven order" and denote 7-order. It is a recursive method in which every permutation π of $\langle n{-}1 \rangle$ is expanded into n permutations of $\langle n \rangle$ by inserting n in every possible way into π. These insertions are done in a peculiar order that is reminiscent the way the number seven is normally written, and is what inspires us to call it 7-order. That is, the first permutation is $n\pi$, the second is πn, and the remaining permutations are obtained by moving the n one position to the left until it is in the second position. We use the notation 7_n to denoted the 7-order of $\langle n \rangle$. Thus $7_2 = 21, 12$ and $7_3 = 321, 213, 231, 312, 123, 132$ and the first 4 permutations of 7_4 are $4321, 3214, 3241, 3421$.

Theorem 2. *The list 7_n is recyclable.*

Proof. We use Theorem 1. The first condition is clearly met by 7_n. To verify the second condition, our strategy is to show that every symbol in α is determined by $\beta_1 \cdots \beta_{j-1}$ and $\alpha_{j+1} \cdots \alpha_n$. First note that either n is in $\alpha_{j+1} \cdots \alpha_n$ or it is in $\beta_1 \cdots \beta_{j-1}$. If n is in $\alpha_{j+1} \cdots \alpha_n$, then $\alpha = \beta_1 \cdots \beta_{j-1} x \alpha_{j+1} \cdots \alpha_n$, where $x = \langle n \rangle \setminus \{\beta_1, \ldots, \beta_{j-1}, \alpha_{j+1}, \ldots, \alpha_n\}$. If $n = \beta_k$ is in $\beta_1 \cdots \beta_{j-1}$, but $n \neq \beta_1$, then $\alpha = \beta_1 \cdots \beta_{k-1} \beta_{k+1} n \beta_{k+2} \cdots \beta_{j-1} x \alpha_{j+1} \cdots \alpha_n$. If $\beta_1 = n$ then the result follows by induction. □

We define the *bell-ringer order* to be the order of permutations obtained by recycling 7_n.

2.1 The binary interpretation of bell-ringer order

In this subsection we infer the recursive 0/1 structure of $\mathsf{R}(7_n)$. Since the ns are n apart, every nth and $(n+1)$st bit is 0. (This is because $ns_1s_2 \cdots s_{n-1} \in \Pi(n)$ and $s_1s_2 \cdots s_{n-1}n \in \Pi(n)$ when $s_1s_2 \cdots s_{n-1}$ is a sub-permutation.) We thus may think of the 0/1 string as having the form

$$\Upsilon(n) \;=\; 00\ B(n)_1\ 00\ B(n)_2\ 00\ \cdots\ 00\ B(n)_{(n-1)!},$$

where $|B(n)_j| = n-2$. The initial 0 represents the σ_n that takes $n(n-1)\cdots 21$ to $(n-1)\cdots 21n$. We will now describe how to get $\Upsilon(n+1)$ from $\Upsilon(n)$.

We use several strings which are defined below. We omit n from the first two notations since it will be clear from context. Let

$$A = 1^{n-2}, \quad Z_k = 0^{n-k-2}10^{k-1}, \text{ and } \quad X_{n-1} = A00Z_100Z_200\cdots 00Z_{n-3}.$$

Note that $|A| = |Z_k| = n-2$ and $|X_n| = (n-1)(n-2)$. Given $\Upsilon(n)$, the 0/1 string for $n+1$ is

$$\Upsilon(n+1) \;=\; 00\ X_n\ 00\ 1B(n)_1\ 00\ X_n\ 00\ 1B(n)_2\ 00\ \cdots\ 00\ X_n\ 00\ 1B(n)_{(n-1)!}.$$

Here are the bitstrings for $\Upsilon(4)$ and $\Upsilon(5)$, where the initial case is $\Upsilon(3) = 00\,1\,00\,1 = 00\ B(3)_1\ 00\ B(3)_2$. First, $\Upsilon(4) = 00\ 11\ 00\ 01\ 00\ 1\underline{1}\ 00\ 11\ 00\ 01\ 00\ 1\underline{1}$, which can also be written as $\Upsilon(4) = 00\ A\ 00\ Z_1\ 00\ 1B(3)_1\ 00\ A\ 00\ Z_1\ 00\ 1B(3)_2 = 00\ X_2\ 1B(3)_1\ 00\ X_2 00\ 1B(3)_2$. And $\Upsilon(5)$ is

$00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11}$ $(= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_1 = 00X_3001B(4)_1)$
$00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{01}$ $(= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_2 = 00X_3001B(4)_2)$
$00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11}$ $(= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_3 = 00X_3001B(4)_3)$
$00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11}$ $(= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_4 = 00X_3001B(4)_4)$
$00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{01}$ $(= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_5 = 00X_3001B(4)_5)$
$00\ 111\ 00\ 001\ 00\ 010\ 00\ 1\underline{11}$ $(= 00\ A\ 00\ Z_1\ 00\ Z_2\ 00\ 1B(4)_6 = 00X_3001B(4)_6)$.

Theorem 3. *The number of 1s in $\Upsilon(n)$ is $2((n-1)!-1)$.*

Proof. If c_n is the number of 1s then our recursive construction implies that $c_{n+1} = c_n + 2(n-2)(n-2)!$ with $c_3 = 2$. The solution of this recurrence relation is $2((n-1)!-1)$. □

Asymptotically, this means that the relative frequency of σ_{n-1} transitions is about $2/n$ and the relative frequency of σ_n transitions is asymptotically $(n-2)/n$. This answers an open question listed at the end of [7]; it asks whether there is a Ucycle whose binary representation uses more 1s than 0s. The bell-ringer listing clearly does so.

2.2 Iterative Rules

Now let us describe a rule for transforming one permutation of $\langle n-1\rangle$ in 7-order into the next. This is useful for efficiently generating 7-order and the bell-ringer shorthand Ucycle, as well as proving that it is indeed a Ucycle.

Let $\boldsymbol{s} = s_1 s_2 \cdots s_{n-1} \in \Pi(n-1)$. Let h be the index such that $s_h = n-1$. If $h = 2$, then let i be the maximum value such that

$$s_1\ s_2\ \cdots\ s_i = s_1\ (n{-}1)\ (n{-}2)\ \cdots\ (n{-}i{+}1)$$

and let j be chosen to maximize the value of s_j such that $i+1 \leq j \leq n-1$. (Notice that j is undefined when $i = n-1$, and otherwise $j > i+1$.) The next permutation in 7-order denoted $7(\boldsymbol{s})$ and is obtained from the following formula

$$\begin{cases} s_1 s_2 \cdots s_{h-2} s_h s_{h-1} s_{h+1} s_{h+2} \cdots s_{n-1} & \text{if } h > 2 \quad \text{(4a)} \\ s_2 s_3 \cdots s_i s_1 s_{i+2} s_{i+2} \cdots s_{j-2} s_j s_{j-1} s_{j+1} s_{j+2} \cdots s_{n-1} & \text{if } h{=}2 \,\&\, s_1{<}n{-}i \quad \text{(4b)} \\ s_2 s_3 \cdots s_{n-1} s_1 & \text{otherwise.} \quad \text{(4c)} \end{cases}$$

To see why (4) generates 7-order, one can simply compare each of the cases to the recursive definition of seven order:

– (4c) is performed when the largest symbol is in the first position of $\boldsymbol{s}$, and the result is the first symbol of $\boldsymbol{s}$ is moved into the last position;
– (4a) is performed when the largest symbol is in neither of the first two positions of $\boldsymbol{s}$, and the result is the largest symbol moves one position left;
– (4b) is performed when the largest symbol is in the second position, and the result is that symbols $s_2 s_3 \cdots s_i$ move one position to the left by recursion, and then the jth symbol moves one position to the left.

Algorithmically, successive iterations of (4) can be generated by a constant amortized time (CAT) algorithm when $\boldsymbol{s}$ is stored in array. That is, the 7-order for $\Pi(n-1)$ can be generated in $\mathrm{O}((n-1)!)$-time. To do this, one needs to simply keep track of the position of the largest symbol in a variable h. More precisely, given the value of h, (4c) is performed $1 \cdot (n-2)!$ times, and involves $O(n-1)$ work each time. Similarly, (4a) is performed $(n-2) \cdot (n-2)!$ times, and involves $O(1)$ work. Finally, (4b) is performed $1 \cdot (n-2)!$ times, and involves $O(n-1)$ work each time. Therefore, the overall implementation $\mathrm{O}((n-1)!)$-time since

$$n \cdot (n-2)! + n \cdot (n-2)! + (n-2) \cdot (n-2)! = 3n - 2 \cdot (n-2)! \leq 4 \cdot (n-1)!.$$

This proves the following theorem.

Theorem 4. *7-order for $\Pi(n-1)$ can be generated in $O((n-1)!)$-time when the permutations are stored in an array. Using the same algorithm, the bell-ringer shorthand Ucycle for $\Pi(n)$ can be generated in $O((n-1)!)$-time when successive blocks of length n are stored in an array (and the first element of the array is fixed at value n).*

The iterative rule in (4) also allows us to state a simple iterative rule for directly generating the permutations from the bell-ringer Ucycle.

Theorem 5. *Let $\boldsymbol{s} = s_1 s_2 \cdots s_n \in \Pi(n)$ be a permutation in the bell-ringer shorthand Ucycle for $\Pi(n)$, where m is the maximum value of s_1 and s_n, and k is the maximum value such that $n\ (n{-}1)\ \cdots\ k$ appears in the permutation as a circular substring. If $k-1 \leq m \leq n-1$, then the next permutation is $s_2 s_3 \cdots s_{n-1} s_1 s_n$. Otherwise, (if $m = n$ or $k-1 < m$) the next permutation is $s_2 s_3 \cdots s_n s_1$.*

Proof. Omitted. □

3 Cool-lex Construction

This section discusses the *cool-lex order* for $\Pi(n)$ [9]. Given a permutation, a *prefix left-shift* moves the symbol at a given position into the first position of the resulting permutation. This operation is denoted by $\lhd$ as follows

$$\lhd(s_1 s_2 \cdots s_n,\ j) = s_j s_1 s_2 \cdots s_{j-1} s_{j+1} s_{j+2} \cdots s_n.$$

A *prefix right-shift* is the inverse operation and involves moving the symbol in the first position into a given position. This operation is denoted by $\rhd$ as follows

$$\rhd(s_1 s_2 \cdots s_n,\ j) = s_2 s_3 \cdots s_{j-1} s_1 s_{j+1} s_{j+2} \cdots s_n.$$

Cool-lex order is generated by a single operation that is repeated over and over again. The operation was originally stated in terms of prefix left-shifts, but for the purposes of this document it will be useful to restate the definition in terms of prefix right-shifts. Both the *cool left-shift* and *cool right-shift* involve the notion of a *non-increasing prefix* which is defined below.

Definition 2 ($\urcorner\!\!\llcorner$). *If* $\boldsymbol{s} = s_1 s_2 \ldots s_n$ *is a string, then the* non-increasing prefix *of* $\boldsymbol{s}$ *is*

$$\urcorner\!\!\llcorner(\boldsymbol{s}) = s_1 s_2 \cdots s_j$$

where j *is the maximum value such that* $s_{j-1} \geq s_j$ *for all* $2 \leq j \leq \urcorner\!\!\llcorner(\boldsymbol{s})$*.*

For example, $\urcorner\!\!\llcorner(55432413) = 55432$ and $\urcorner\!\!\llcorner(33415312) = 33$.

Given a list of strings, the *reflected* list begins with the last string in the original list and ends with the first string in the original list. In particular, the reflected cool-lex order for $\Pi(n)$ is generated by repeated applications of the cool right-shift which is defined below.

Definition 3. *Let* $\boldsymbol{s} = s_1 \cdots s_n$ *and* $\boldsymbol{s}' = s_2 s_3 \cdots s_n$ *and* $k' = |\urcorner\!\!\llcorner(\boldsymbol{s}')|$*. Then,*

$$\overrightarrow{\mathsf{cool}}(\boldsymbol{s}) = \begin{cases} \rhd(\boldsymbol{s},\ k'+1) & \textit{if } k' \leq n-2 \textit{ and } s_1 > s_{k'+1} & \text{(5a)} \\ \rhd(\boldsymbol{s},\ k'+2) & \textit{if } k' \leq n-2 \textit{ and } s_1 < s_{k'+1} & \text{(5b)} \\ \rhd(\boldsymbol{s},\ n) & \textit{otherwise (if } k' \geq n-1\textit{).} & \text{(5c)} \end{cases}$$

For example, $\overrightarrow{\mathsf{cool}}$ circularly generates the following list of $\Pi(3)$. These lists for $\Pi(n)$ are denoted by $\overrightarrow{\mathcal{C}}(n)$, as in $\overrightarrow{\mathcal{C}}(3) = 321\ 213\ 123\ 231\ 312\ 132$.

We now prove that $\overrightarrow{\mathcal{C}}(n)$ becomes a universal cycle for $\Pi_n(n+1)$ after prefixing $n+1$ as a periodic symbol. For example, when $n = 3$

$$\overrightarrow{\mathcal{C}}(3) = \ \ 321\ \ 213\ \ 123\ \ 231\ \ 312\ \ 132$$
$$\mathsf{R}(\overrightarrow{\mathcal{C}}(3)) = 432142134123423143124132$$

and $\mathsf{R}(\overrightarrow{\mathcal{C}}(3))$ is a universal cycle for the 3-permutations of $\langle 4 \rangle$. In general, if $\mathcal{L}$ is a list of $\Pi(n)$ then $\mathsf{R}(\mathcal{L})$ denotes the result of prefixing $n+1$ to every permutation in $\mathcal{L}$ and then concatenating the resulting strings together. Using this notation, the main result can be stated as follows.

Theorem 6. *The string* $\mathsf{R}(\overrightarrow{\mathcal{C}}(n))$ *is a universal cycle for* $\Pi_n(n+1)$.

Proof. This result follows from a judicious application of Lemma 1, below, that is stated informally as follows: For every value of j satisfying $0 \le j \le n-1$, there are consecutive strings in $\overrightarrow{\mathcal{C}}(n)$ that contain the last j symbols of $\boldsymbol{p}$ as a prefix of one string, and the first $(n-1)-j$ symbols as a suffix of the previous string. The rest of the proof is omitted due to space limitations. □

Lemma 1. *If* $\boldsymbol{p} \in \Pi_{n-1}(n)$ *and* j *satisfies* $0 \le j \le n-1$, *then there exists* $\boldsymbol{s} = s_1 s_2 \cdots s_n \in \Pi(n)$ *followed by* $\boldsymbol{t} = t_1 t_2 \cdots t_n \in \Pi(n)$ *in* $\overrightarrow{\mathcal{C}}(\mathcal{L})$ *such that*

$$s_{j+2} s_{j+3} \cdots s_n t_1 t_2 \cdots t_j = \boldsymbol{p}. \tag{6}$$

In other words, there exist consecutive strings in $\overrightarrow{\mathcal{C}}(n)$ *whose concatenation has* $\boldsymbol{p}$ *as a substring. Moreover, the substring uses* j *symbols from the second string.*

Proof. The proof of this technical lemma is omitted here. □

References

1. F. Chung, P. Diaconis, and R. Graham, *Universal cycles for combinatorial structures*, Discrete Mathematics, 110 (1992) 43–59.
2. N.G. de Bruijn, *A Combinatorial Problem*, Koninkl. Nederl. Acad. Wetensch. Proc. Ser A, 49 (1946) 758–764.
3. R. Duckworth and Fabian Stedman, *Tintinnalogia*, 1668.
4. B. Jackson, *Universal cycles of k-subsets and k-permutations*, Discrete Mathematics, 149 (1996) 123–129.
5. D.E. Knuth, *The Art of Computer Programming, Volume 4, Generating All Tuples and Permutations*, Fascicle 2, Addison-Wesley, 2005.
6. F. Ruskey, and A. Williams, *The coolest way to generate combinations*, Discrete Mathematics, 309 (2009) 5305–5320. .
7. F. Ruskey and A. Williams, *An explicit universal cycle for the* $(n-1)$*-permutations of an n-set*, ACM Transactions on Algorithms, in press.
8. A. T. White, *Fabian Stedman: The First Group Theorist?*, The American Mathematical Monthly, 103 (1996) 771–778.
9. A. Williams, *Loopless Generation of Multiset Permutations Using a Constant Number of Variables by Prefix Shifts*, Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009, pp. 987–996.