A Universal Cycle for Strings with Fixed-Content

J. Sawada[⋆] A. Williams[⋆⋆]

Abstract. We develop the first universal cycle construction for strings with fixed-content (also known as multiset permutations) using shorthand representation. The construction runs a necklace concatenation algorithm on cool-lex order for fixed-content strings, and is implemented to generate the universal cycle in amortized $O(1)$-time per symbol. This generalizes two previous results: a universal cycle for shorthand permutations by Ruskey, Holroyd, and Williams [*Algorithmica* 64 (2012)] and a universal cycle for shorthand fixed-weight binary strings by Ruskey, Sawada, and Williams [*SIAM J. on Disc. Math.* 26 (2012)]. A consequence of our construction is the first shift Gray code for fixed-content necklaces.

Keywords: de Bruijn cycle, universal cycle, fixed-content, multiset permutation, necklace, Gray code, cool-lex

1 Introduction

A *universal cycle* for a set $\mathbf{S}$ of length n strings, is a circular string of length $|\mathbf{S}|$ where every string in $\mathbf{S}$ appears exactly once as a substring. When $\mathbf{S}$ is the set of *all* k-ary strings of length n, universal cycles for $\mathbf{S}$ are known as *de Bruijn sequences*, and there is a rich history of various constructions (see [6, 7] for recent surveys and the online project `debruijnsequence.org`).

Universal cycles for many interesting sets are known to exist [2, 11, 3, 12, 20]; for instance weak orders [10, 21] and weight-range strings [19, 17, 18]. However for other common sets such as the set of permutations or the set of fixed-weight (the number of 1s is fixed) binary strings, it is easy to see that universal cycles do not exist. Fortunately, for each of these two cases, a *shorthand* notation can be adopted as both permutations and fixed-weight strings can be represented by their length $n-1$ prefixes, since the final symbol is redundant. For these *shorthand* representations, efficient universal cycle constructions are known (for permutations [15, 9], for fixed-weight strings [14]).

Example 1 Consider the set $\mathbf{S}_1 = \{12, 13, 21, 23, 31, 32\}$ of shorthand permutations of order $n = 3$. Observe that 231321 is a universal cycle for $\mathbf{S}_1$.

[⋆] Computing and Information Science, University of Guelph, Canada. Research supported by NSERC. `E-mail: jsawada@uoguelph.ca`

[⋆⋆] Computer Science, Williams College, Williamstown, USA. `.E-mail: aaron.williams@williams.edu`

Example 2 Consider the set $\mathbf{S}_2 = \{0001, 0010, 0100, 1000, 0011, 0110, 0101, 1001, 1010, 1100\}$ of shorthand fixed-weight strings for $n = 5$ weight 2. Observe that 1010011000 is a universal cycle for $\mathbf{S}_2$.

Strings with fixed-content, or mutiset permutations, generalize both permutations and fixed-weight binary strings. For permutations or order n, the content is $\{1, 2, \ldots, n\}$ and for strings of length n with fixed weight d, the content is the multiset containing d 1s and $n - d$ 0s. In this paper we consider the more general question of constructing universal cycles for fixed-content strings, using their shorthand representation. A more general example is given later in the paper.

Main Result. The first known construction of universal cycles for strings with fixed-content. The construction is based on a known concatenation construction applied to the cool-lex order of necklaces to generate the universal cycles in $O(1)$-amortized time per symbol.

Along the way, we develop an algorithm to list necklaces with fixed-content in a shift Gray code order in $O(n)$-amortized time per necklace.

The rest of the paper is presented as follows. Section 2 discusses preliminary concepts, including fixed-content necklaces and cool-lex order. Section 3 provides a new recursive algorithm for generating fixed-content necklaces in cool-lex order. Section 4 presents our universal cycle construction for strings with fixed-content.

Our universal cycle construction for fixed weight strings is implemented in C and is available in the Appendix.[1]

2 Preliminaries

In this section, we introduce the basic concepts and notation used in the construction of our universal cycle.

Let S be a multiset over the alphabet $\{1, 2, \ldots, k\}$, denoting the fixed-content of our strings with $n = |S|$, and let $\boldsymbol{S}(S)$ denote the set of all strings with fixed-content S. Let $\alpha = a_1 a_2 \cdots a_n$ be a string. Let α^t denote the string composed of t copies of α. The *period* of α is the smallest value j such that $\alpha = (a_1 \cdots a_j)^t$ for some integer t; we say $a_1 \cdots a_j$ is the *aperiodic prefix* of α. If α has period n (it is the same as its aperiodic prefix), we say it is *aperiodic*; otherwise we say it is *periodic*.

2.1 Necklaces with Fixed-Content

A *necklace* is defined to be the lexicographically smallest string in an equivalence class of strings under rotation. Let $\boldsymbol{N}(S)$ denotes the set of all necklaces with fixed-content

[1] Its output can also be viewed at `debruijnsequence.org`.

S. The number of fixed-content necklaces can be deduced using Pólya theory as discussed in [8]. In the following formula, it is assumed that the content S is composed of $n_i \geq 1$ occurrences of each symbol i, $|S| = n$, and $k \geq 1$:

$$\boldsymbol{N}(S) = \frac{1}{n} \sum_{j | gcd(n_1, n_2, \ldots, n_k)} \phi(j) \frac{(n/j)!}{(n_1/j)! \cdots (n_k/j)!} \tag{1}$$

where Euler's totient function $\phi(j)$ denotes the number of positive integers less than or equal to j that are relatively prime to j.

There exists a $O(1)$-amortized time algorithm to list $\boldsymbol{N}(S)$ [16].

2.2 Cool-lex order

Cool-lex order for fixed-content strings was introduced in [22]. The order is a *Gray code*, meaning that successive strings differ by a simple operation. More specifically, it is a *prefix-shift Gray code*, meaning that successive strings differ by a single prefix-shift. A *prefix-shift* removes a single symbol and reinserts it as the first symbol; in a linked list representation, this corresponds to a removing a node and reinserting it as the head. The order is also *cyclic* meaning that a prefix-shift also transforms the last string in the order into the first. As an example, the set $\boldsymbol{S}(\{1,1,2,2,3,3\})$ is listed in cool-lex order on the left side of Figure 1.

One of the most notable features of cool-lex order is that it has a simple *successor rule*. In other words, the prefix-shift that creates the next fixed-content string in the order is relatively easy to specify. To describe the rule, let *the non-decreasing prefix* of a string be its longest prefix with no increases. In other words, if $a_1a_2 \cdots a_n$ is the string, then its non-decreasing prefix is $a_1a_2 \ldots a_p$ with $a_i \leq a_{i+1}$ for $1 \leq i < p$ and either $p = n$ or $a_p > a_{p+1}$. Now we can describe the cool-lex successor rule.

Cool-lex Successor Rule

If the non-decreasing prefix of $a_1a_2 \cdots a_n$ has length $p < n$, then the next string in cool-lex order is obtained by a prefix-shift of length $p+1$ if $p = n-1$ or $a_p > a_{p+2}$, and otherwise by a prefix-shift of length $p+2$.

This rule specifies every transition in the cyclic order except one: If the string itself is non-decreasing, then the next string is obtained by a prefix-shift of length n.

Another benefit of cool-lex order is that its relative order provides shift Gray codes for other sets of strings. This phenomenon was discussed for fixed-weight sets in [13], and for fixed-content sets in [23]. In particular, this occurs for necklaces, as illustrated in Figure 1 for $S = \{1,1,2,2,3,3\}$. By adapting the techniques from [13, 23], we obtain the successor rule given below for necklaces with fixed-content. Our shift notation is discussed after the rule is presented.

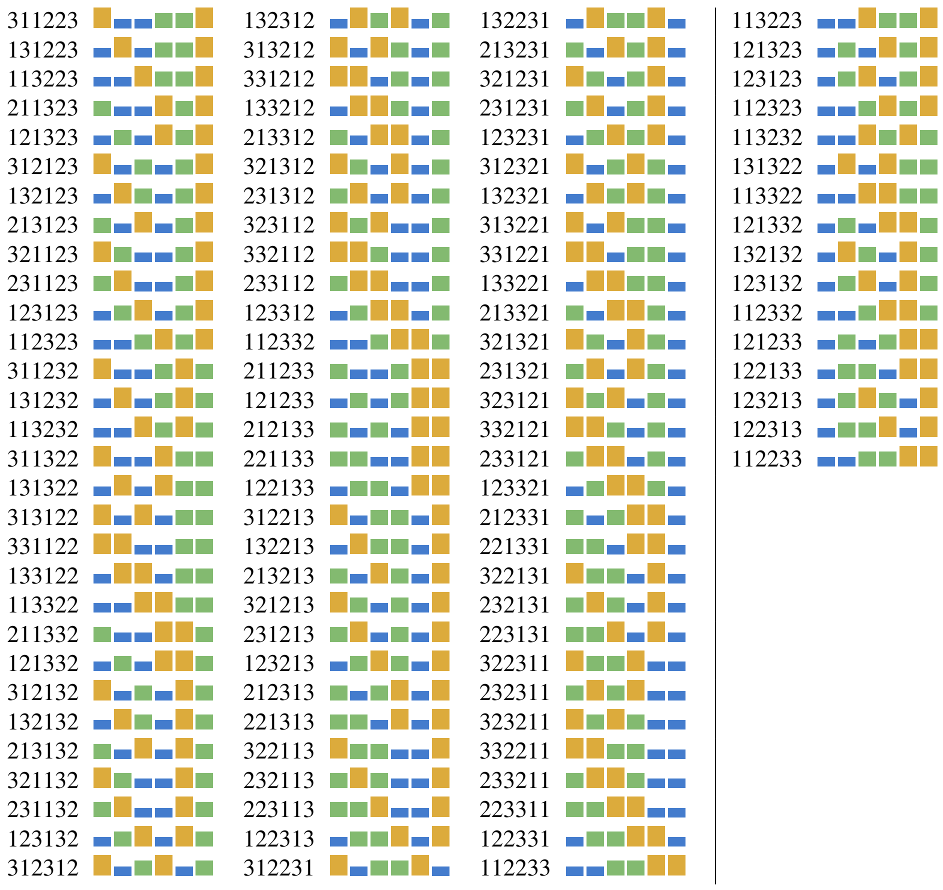


Fig. 1. The columns to the left of the vertical line illustrate cool-lex order for strings with content $S = \{1, 1, 2, 2, 3, 3\}$. Observe that each string is obtained from the previous by a prefix-shift, and the order is cyclic in this regard. The column to the right of the vertical line illustrates the necklaces with content S as they appear in cool-lex order. Observe that each necklace is obtained from the previous one by a shift, and the order is cyclic in this regard.

Cool-lex Successor Rule for Fixed-Content Necklaces

Let $\alpha = a_1a_2 \cdots a_n = \lambda\gamma \in \boldsymbol{N}(S)$, where $|\alpha| = |S| = n$, λ is α's non-decreasing prefix, and $m = |\lambda|$. The necklace following α in cool-lex order is denoted $\mathsf{next}(\alpha)$ and is obtained from α by the shift in the following cumulative cases

$$\mathsf{next}(\alpha) = \begin{cases} \mathsf{lshift}_\alpha(m) & \text{if } m = n & \text{(2a)} \\ \mathsf{lshift}_\alpha(m{+}1) & \text{if } m = n-1 \text{ or } a_m > a_{m+2} \text{ or } \beta \notin \boldsymbol{N}(S) & \text{(2b)} \\ \mathsf{lshift}_\alpha(m{+}2) & \text{otherwise} & \text{(2c)} \end{cases}$$

where β is obtained from α by swapping the two symbols after γ (if they exist).

Now we define the lshift operation used in (2). If $\alpha = a_1a_2 \cdots a_n$ is a necklace, then $\mathsf{lshift}_\alpha(i)$ bubbles a_i as far to the left as possible while always maintaining that the result is still a necklace. In particular, $\mathsf{lshift}_\alpha(i) = \alpha$ if the first swap results in a non-necklace. For example, consider the necklace $\alpha = a_1a_2a_3a_4a_5a_6 = 122313$. We can determine the result of $\mathsf{lshift}_\alpha(4)$ by bubbling the symbol $a_4 = 3$ to the left, starting from α, as follows:

$$\begin{aligned} a_1a_2a_3a_4a_5a_6 &= 122313 \text{ is a necklace;} \\ a_1a_2a_4a_3a_5a_6 &= 123213 \text{ is a necklace;} \\ a_1a_4a_2a_3a_5a_6 &= 132213 \text{ is not a necklace.} \end{aligned}$$

The result is the last necklace in this list. Hence, $\mathsf{lshift}_\alpha(4) = a_1a_2a_4a_3a_5a_6 = 123213$. To motivate the next section, it is important to note that this calculation involved testing if multiple strings were necklaces. This means that without further optimization, the necklace successor rule runs in $O(n^2)$ time.

2.3 Necklace-Prefix Algorithm

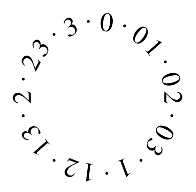
Perhaps the most well-known de Bruijn sequence is the so-called *granddaddy de Bruijn sequence*; it is the lexicographically smallest k-ary de Bruijn sequence of order n. It can be generated very elegantly using an approach that is often referred to as the *FKM construction* or *FKM algorithm*, due to the authors who discovered it [4, 5]. As discussed in [14], the authors prefer to describe the construction using a slightly different approach called the *necklace-prefix algorithm*. The latter approach constructs the granddaddy de Bruijn sequence in a nearly identical manner, but it is often more well-suited for creating other sequences.

The necklace-prefix algorithm takes an order of strings, filters out the non-necklaces, reduces the remaining necklaces to their aperiodic prefix, and concatenates these prefixes. Amazingly, the granddaddy de Bruijn sequence is created by applying the necklace-prefix algorithm to the k-ary strings of length n in lexicographic order. This is illustrated in Figure 2 for $n = 2$ and $k = 4$. The approach is generalized to other sets in [20].

lexicographic order ↓

necklaces	aperiodic prefixes
00	0
01	01
02	02
03	03
11	1
12	12
13	13
22	2
23	23
33	3
(a)	(b)

granddaddy de Bruijn sequence



(c)

Fig. 2. The necklace-prefix algorithm applied to the 4-ary strings of length 2 constructs the granddaddy de Bruijn sequence for $n = 2$ and $k = 4$. The algorithm starts with the lexicographic order of 4-ary strings of length 2 (which are not shown), then reduces the order to the necklaces in column (a), and their aperiodic prefixes in column (b), and concatenates these prefixes to get the granddaddy de Bruijn sequence 0010203112132233 in (c).

Unfortunately, the magic runs out when we try this approach for fixed-content strings, even for their shorthand representatives. To illustrate the issue, note that the lexicographic order of necklaces with content $S = \{1, 1, 2, 2, 3, 3\}$ places the following two necklaces consecutively,

$$\ldots 113322, 121233, \ldots,$$

and so, the necklace-prefix algorithm will genereate $\cdots 1133\mathbf{22121}233 \cdots$. The bolded substring of length $n-1 = 5$ is not shorthand for a string with the content S because it contains too many copies of 2. When considering the transition between the two necklaces, the cause of the issue becomes clear: A copy of 2 moves several positions to the left between the two necklaces. This issue leads us to instead use a reversed version of cool-lex order, since this will ensure that individual symbols move at most one position to the left between successive necklaces.

3 Recursive Algorithm

In [22], a recursive description is given to list all strings with fixed-content S in cool-lex order. In that description, the focus is on strings in reverse lexicographic order, whereas, we will focus on lexicographic order. In this section, we restate this recurrence using the original terminology and then apply it to generate necklaces with fixed-content S.

The *tail* of S, denoted $tail(S)$, is the unique non-decreasing string composed of *all* the elements of S. A *scut* of S is any non-decreasing string α composed of *some* of the elements of S such that α is not a suffix of $tail(S)$, but every proper suffix of α is a suffix of $tail(S)$. Let $\alpha_i(S)$ denote the i-th scut of S when they are listed in decreasing order of the first symbol, then by decreasing length. Let R_i denote the multi-set S with the content of $\alpha_i(S)$ removed.

Example 3 Consider $S = \{1,1,2,2,3,3\}$. Then $tail(S) = 112233$ and the scuts of S in decreasing order of the first symbol, then decreasing length, are:

$$23,\ 2,\ 1233,\ 133,\ 13,\ 1.$$

Note $\alpha_4(S) = 133$ and $R_4 = \{1,1,2,2,3,3\} \setminus \{1,3,3\} = \{1,2,2\}$.

If S has j scuts, then the following recurrence $\boldsymbol{\mathcal{C}}(S,\gamma)$ (simplified from Definition 2.4 in [22]) produces a listing for all strings of the form $\beta\gamma$ where β has content S as they appear in cool-lex order:

$$\boldsymbol{\mathcal{C}}(S,\gamma) = \boldsymbol{\mathcal{C}}(R_1,\alpha_1\gamma), \boldsymbol{\mathcal{C}}(R_2,\alpha_2\gamma), \ldots, \boldsymbol{\mathcal{C}}(R_j,\alpha_j\gamma), tail(S)\gamma.$$

Note that $\boldsymbol{\mathcal{C}}(S,\epsilon)$ will produce a listing of all strings with fixed-content S. Recall Figure 1 illustrating the cool-lex order for $\boldsymbol{S}(\{1,1,2,2,3,3\})$. This is the same listing generated by $\boldsymbol{\mathcal{C}}(\{1,1,2,2,3,3\},\epsilon)$. In particular observe that the strings are ordered by suffixes corresponding to the scuts: 23, 2, 1233, 133, 13, 1.

We now focus on how to modify this recurrence to list the necklaces with fixed-content S as they appear in cool-lex order.

Lemma 1. *If $a_1a_2\cdots a_n$ is a necklace that contains a smallest index t such that $a_t > a_{t+1}$, then $a_1\cdots a_{t-1}a_{t+1}a_ta_{t+2}\cdots a_n$ is a necklace.*

Proof. Let $\beta = b_1b_2\cdots b_n = a_1\cdots a_{t-1}a_{t+1}a_ta_{t+2}\cdots a_n$. Let β_j denote the rotation of β starting at b_j and let α_j denote the rotation of $\alpha = a_1a_2\cdots a_n$ starting at a_j. If $\beta \le \beta_j$ for each $2 \le j \le n$, then β is a necklace. Since α is a necklace, each $a_i \ge a_1$ and thus each $b_i \ge b_1$. Since $b_1\cdots b_{t-1}$ is non-decreasing it is straightforward to observe that $\beta_j > \beta$ for $2 \le j \le t+1$. Now consider the prefix of length t for β_j where $t+2 \le j \le n$. This prefix is the same as the length t prefix of α_j. If this prefix if less than or equal to $b_1\cdots b_t$, then it must be strictly less than $a_1\cdots a_t$ since $a_t > b_t$. But this contradicts the fact that α is a necklace. Thus this prefix must be strictly greater than $b_1\cdots b_t$. Thus $\beta_j \ge \beta$ for each $2 \le j \le n$ and hence β is a necklace.

Lemma 2. *$\boldsymbol{\mathcal{C}}(S,\gamma)$ contains a necklace if and only if $tail(S)\gamma$ is a necklace.*

Proof. ($\Leftarrow$) $tail(S)\gamma$ is in $\boldsymbol{\mathcal{C}}(S,\gamma)$ by definition. Thus if $tail(S)\gamma$ is a necklace then $\boldsymbol{\mathcal{C}}(S,\gamma)$ contains a necklace. ($\Rightarrow$) If $\boldsymbol{\mathcal{C}}(S,\gamma)$ contains necklace then it must be of the form $\lambda\gamma$ where λ has content S. If $\lambda = tail(S)$, then we are done. Otherwise, repeated application of Lemma 1 implies that $tail(S)\gamma$ is a necklace.

Based on Lemma 1, the recurrence $\boldsymbol{\mathcal{C}}(S,\gamma)$ can be updated to list only the necklaces as follows (where $\langle\,\rangle$ denotes an empty list).

$$\boldsymbol{\mathcal{N}}(S,\gamma) = \begin{cases} \langle\,\rangle & \text{if } tail(S)\gamma \text{ is not a necklace;} \\ \boldsymbol{\mathcal{N}}(R_1,\alpha_1\gamma), \ldots, \boldsymbol{\mathcal{N}}(R_j,\alpha_j\gamma), tail(S)\gamma & \text{otherwise,} \end{cases}$$

Algorithm 1 Recursive algorithm to list all necklaces with content S as they appear in cool-lex order. The string $a_1a_2 \cdots a_n$ is intialized to $tail(S)$, and the initial call is COOL(n).

```
1: procedure COOL( $t$ )
2:     $i \leftarrow t$
3:     while $a_i \neq a_1$ do
4:         while $a_i = a_{i-1}$ do  $i \leftarrow i-1$
5:         for $j$ from i to $t$ do
6:             SWAP($j-1, j$)
7:             if $a_1a_2 \cdots a_n$ is a necklace then COOL($j-1$)
8:         for $j$ from $t$ down to $i$ do  SWAP($j-1, j$)
9:         $i \leftarrow i-1$
10:    VISIT( )
```

The function COOL(t) in Algorithm 1 implements the above recurrence. Given content S, by initializing the global string $a_1a_2 \cdots a_n$ to $tail(S)$, the initial call COOL(n) generates all necklaces with fixed-content S. The parameter t passed in the function COOL(t) indicates how the string $a_1a_2 \cdots a_n$ is partitioned into the two pieces based on $\boldsymbol{\mathcal{N}}(S', \gamma)$: $a_1a_2 \cdots a_t = tail(S')$ and $a_{t+1} \cdots a_n = \gamma$. Each call COOL($t$) corresponding to $\boldsymbol{\mathcal{N}}(S', \gamma)$ iterates through the scuts of S' in the proper order. This is done by scanning $tail(S') = a_1 \cdots a_t$ from right to left until we reach an index i where $a_i \neq a_{i-1}$ (Line 4). To produce all scuts starting with a_{i-1}, and their corresponding recursive calls if a necklace can be produced, we iteratively shift this symbol through positions $i, i+1, \ldots, t$ obtaining a new scut for each swap (Lines 5-7). Once all scuts starting with a_{i-1} have been processed we restore $a_1 \cdots a_t$ to $tail(S')$ (Line 8). We repeat this approach by continuing to traverse $tail(S)$ from right to left until we reach a symbol that is the same as a_1 (Line 3). The function VISIT() outputs the string $a_1a_2 \cdots a_n$, and the function SWAP(i, j) swaps the symbols at index i and j in $a_1a_2 \cdots a_n$.

When analyzing this algorithm, if every string tested in Line 7 was a necklace, then the work done by each necklace test could be assigned to the following recursive call. Since each recursive call generates at least one necklace, and since the necklace testing can be done in $O(n)$-time [1], the overall algorithm would run in $O(n)$-amortized time algorithm. However, within each recursive call, there could be a number of negative necklace tests. For instance, consider the string $\alpha = 112233112233$ and the call to COOL(6). This results in necklace tests for the following 6 strings, none of which are necklaces since the rotation starting with the suffix 112233 is smaller than string in question:

$$112323112233,\ 112332112233,\ 121233112233,$$

$$122133112233,\ 122313112233,\ 122331112233.$$

Fortunately there exists a simple optimization: once a string tested on Line 7 is **not** a necklace, then by applying Lemma 1 (and further observing the definition of a necklace) none of the following strings tested will be either. This optimization can be applied to COOL(t) by replacing Line 7 with the following fragment:

```
if $a_1a_2\cdots a_n$ is a necklace then COOL($j-1$)
else
    for $s$ from $j$ down to $i$ do  SWAP($s-1, s$)
    VISIT( )
    return
```

By applying this optimization there is at most one negative necklace test per recursive call.

Theorem 1. *Let S denote a multi-set from the elements $1, 2, \ldots, k$ If $a_1a_2\cdots a_n$ is initialized to $tail(S)$, then a call to the optimized* COOL(n) *lists all necklaces with fixed-content S in cool-lex order in $O(n)$-amortized time per string.*

4 Constructing a Shorthand Universal Cycle for Fixed-Content

In this section, we provide the first explicit construction of a shorthand universal cycle for fixed-content strings. If the content of the strings is the multiset of symbols S, then the shorthand universal cycle is obtained by the applying the necklace-prefix algorithm to cool-lex order for S. More precisely, we use reverse cool-lex order, meaning that the first string is the unique non-decreasing string, and successive strings differ by our notion of a right-shift, which means that a single symbol is removed and reinserted further to the right, while the intermediate symbols move one position to the left. Also recall that the relative order of the symbols has been inverted in our presentation, with respect to the original presentation of fixed-content cool-lex [22], so that we can use the traditional notion of a necklace being a string in its lexicographically least rotation.

Let $\mathcal{U}(S)$ denote the string resulting from the necklace-prefix algorithm applied to $\boldsymbol{S}(S)$ when listed in reverse cool-lex order. That is, $\mathcal{U}(S)$ is the concatenation of the aperiodic prefixes of necklaces with content S in reverse cool-lex order. An example of $\mathcal{U}(S)$ is provided in Figure 3 for $S = \{1, 1, 2, 2, 3, 3\}$. Let $\boldsymbol{S^{-1}}(S)$ denote the shorthand representations of the strings in $\boldsymbol{S}(S)$.

We start by proving a preliminary result. Let $\mathcal{U}^+(S)$ be the result of concatenating the necklaces with content S in reverse cool-lex order. In other words, $\mathcal{U}^+(S)$ is the same as $\mathcal{U}(S)$, however, the periodic necklaces are not reduced to their aperiodic prefix.

Theorem 2. *The circular string $\mathcal{U}^+(S)$ contains every string in $\boldsymbol{S^{-1}}(S)$ at least once as a substring.*

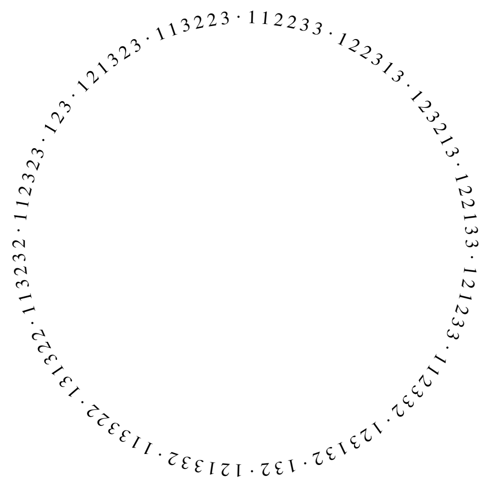
Proof. Consider a string in $\boldsymbol{S}(S)$ whose last symbol is $x \in \{1, 2, \ldots, k\}$. At least one rotation of this string is a necklace. In other words, the string can be written as $\mathbf{pq}x$, such that $\mathbf{q}x\mathbf{p} \in \boldsymbol{N}(S)$. We need to find the string's shorthand reprsentation, $\mathbf{pq}$, as a substring in the universal cycle $\mathcal{U}^+(S)$. In all of our cases, we will use the fact that $\mathbf{q}x\mathbf{p}$ is a necklace.

cool-lex order ↑

necklaces $\boldsymbol{N}(S)$	aperiodic prefixes
112233	112233
122313	122313
123213	123213
122133	122133
121233	121233
112332	112332
123132	123132
132132	132
121332	121332
113322	113322
131322	131322
113232	113232
112323	112323
123123	123
121323	121323
113223	113223

(a) (b)

universal cycle $\mathcal{U}(S)$



(c)

Fig. 3. A universal cycle for $\boldsymbol{S^{-1}}(S)$, where $S = \{1, 1, 2, 2, 3, 3\}$. The cycle uses the shorthand representation, and is constructed using the necklace-prefix algorithm on reverse cool-lex order. The fixed-content necklaces over S are given in reverse cool-lex order in column (a), they are reduced to their aperiodic prefix in column (b), and their concatenation gives the universal cycle in column (c).

We first consider two special cases.

– If $\mathbf{p}$ is empty, then desired substring $\mathbf{pq} = \mathbf{q}$ is contained in the necklace $\mathbf{q}x\mathbf{p} = \mathbf{q}x$, and hence $\mathcal{U}^+(S)$.
– If $\mathbf{q}$ is empty, then desired substring $\mathbf{pq} = \mathbf{p}$ is contained in the necklace $\mathbf{q}x\mathbf{p} = x\mathbf{p}$, and hence $\mathcal{U}^+(S)$.

In the remaining cases, we will need to search across two necklaces to find the desired substring $\mathbf{pq}$. In other words, we need to find a necklace $\alpha \in \boldsymbol{N}(S)$, such that $\mathbf{pq}$ is a substring of $\mathsf{next}(\alpha) \cdot \alpha$, and thereby a substring of $\mathcal{U}^+(S)$. Specifically, we will find $\alpha \in \boldsymbol{N}(S)$ with prefix $\mathbf{q}$, such that $\mathsf{next}(\alpha)$ has suffix $\mathbf{p}$. The following point handles the easiest remaining case.

– If $\mathbf{q}$ is not non-decreasing (i.e. its non-decreasing prefix is a strict prefix), then the necklace $\alpha = \mathbf{q}x\mathbf{p}$ again suffixes. To see why this is true, observe that α has prefix $\mathbf{q}$, and we claim that $\mathsf{next}(\alpha)$ has suffix $\mathbf{p}$. This is because $\mathsf{next}(\alpha)$ is obtained from α by shifting either a symbol in $\mathbf{q}$, or x, to the left by 2. Hence, the suffix $\mathbf{p}$ carries over from α to $\mathsf{next}(\alpha)$.

In the remaining cases, $\mathbf{p}$ and $\mathbf{q}$ are both non-empty, and $\mathbf{q}$ is non-decreasing by itself. We proceed with two representative cases, based on the non-decreasing prefixes of $\mathbf{q}x\mathbf{p}$ and $\mathbf{qp}$. In both cases, we create α by starting from the necklace $\mathbf{q}x\mathbf{p}$, and shifting x far enough to the right, so that it becomes the symbol that will be shifted to the left in $\mathsf{next}(\alpha)$.

– If the non-decreasing prefix in $\mathbf{q}x\mathbf{p}$ is $\mathbf{q}$. In this case, we start with $\alpha = \mathbf{q}x\mathbf{p}$, and repeatedly update the necklace until we have the desired properties. Let m be the length of the non-decreasing prefix, with regard to the successor rule in (2), and note that x is in the $(m+1)$st position in α. if $m = n-1$, or $a_m > a_{m+2}$, or β is not a necklace, then we stop and use the current value of α. Otherwise, bubble x one position to the right to create a new value of α. Notice that α is a necklace due, to the fact that β is a necklace. Furthermore, x is again one symbol to the right of the non-decreasing prefix of α due to the fact that we had $a_m \leq a_{m+2}$ with respect to the previous value of α. Therefore, we can repeat the above steps until we find a suitable α. Observe that $\mathsf{next}(\alpha)$ will shift x to the left by (2b).
– If the non-decreasing prefix in $\mathbf{q}x\mathbf{p}$ is $\mathbf{q}x$, and the non-decreasing prefix in $\mathbf{qp}$ is $\mathbf{q}$. In this case, the last symbol of $\mathbf{q}$ is larger than the first symbol in $\mathbf{p}$. Let α be the result of bubbling x one position to the right in $\mathbf{q}x\mathbf{p}$. Thus, the non-decreasing prefix of α is precisely $\mathbf{q}$. Let $m = |\mathbf{q}|$, with regard to the successor rule in (2), and note that x is in the $(m+1)$st position in α. Furthermore, $a_m \leq a_{m+2}$ based on the assumptions of this case, and $\beta = \mathbf{q}x\mathbf{p}$ is the necklace we started from. Therefore, α suffices since $\mathsf{next}(\alpha)$ will shift x to the left by (2c).

Now we prove our main result.

Theorem 3. *The string $\mathcal{U}(S)$ is universal cycle for fixed-content strings using their shorthand representation. In other words, every string in $\boldsymbol{S^{-1}}(S)$ appears in $\mathcal{U}(S)$ exactly once as a substring.*

Proof. Observe that $\mathcal{U}(S)$ has the correct length. That is, $|\mathcal{U}(S)| = |\boldsymbol{S^{-1}}(S)|$. This is due to the fact that every necklace contributes equally to both quantities. More precisely, a necklace whose period is p will contribute p symbols to $\mathcal{U}(S)$, and its p unique rotations to $\boldsymbol{S^{-1}}(S)$. Due to this equality, we only need to prove that each string in $\boldsymbol{S^{-1}}(S)$ appears in $\mathcal{U}(S)$ at least once as a substring. Because of Theorem 2, we can accomplish this by showing that $\mathcal{U}(S)$ has the same set of substrings of length $n-1$ as $\mathcal{U}^+(S)$. In other words, we can prove the result by showing that no substrings are lost when we reduce each necklace to its aperiodic prefix in the concatenation.

Consider an arbitrary periodic necklace whose aperiodic prefix is γ. Since the necklace is periodice, we can write it as γ^r for some $r > 1$. First we prove that $\mathsf{next}(\gamma^r)$ is aperiodic. To see why this is true, observe that $\mathsf{next}(\gamma^r)$ will contain a prefix that is lexicographically smaller than γ. Thus, $\mathsf{next}(\gamma^r)$ is not periodic. Similarly, $\mathsf{prev}(\gamma^r)$ is not periodic.

Now we compare the local area around γ^r in $\mathcal{U}^+(S)$, and the local area around its reduction tp γ in $\mathcal{U}(S)$.

$$\underbrace{\cdots \mathsf{next}(\gamma^r) \cdot \gamma^r \cdot \mathsf{prev}(\gamma^r) \cdots}_{\mathcal{U}^+(S)} \qquad \underbrace{\cdots \mathsf{next}(\gamma^r) \cdot \gamma \cdot \mathsf{prev}(\gamma^r) \cdots}_{\mathcal{U}(S)}$$

We claim that the two different concatenations have the same set of substrings of length $n-1$. To help establish this fact, let the length of the aperiodic prefix of the necklace γ^r be $t = |\gamma| = n/r$. Now we make two observations. First, $\mathsf{next}(\gamma^r)$ and γ^r share a suffix of length at least $n-t-2$. This is due to the cool-lex successor rule in (2) and the fact that the length of the non-decreasing prefix in γ^r is at most t. Second, γ^r and $\mathsf{prev}(\gamma^r)$ must share a prefix of length at least 1, since they are both necklaces. Therefore, the concatenation in $\mathcal{U}^+(S)$ contains at least $(n-t-2)+t+1 = n-1$ consecutive symbols from γ^r, which means that it contains γ^r in shorthand. Furthermore, the substrings of length $n-1$ in $\mathcal{U}^+(S)$ that are before and after this shorthand copy of γ^r are identical to those found in $\mathcal{U}(S)$.

4.1 Efficiency

To construct the reverse of the universal cycle $\mathcal{U}(S)$, we can directly apply Algorithm 1 to list $\boldsymbol{N}(S)$ in cool-lex order with a simple modification. Instead of outputting the current necklace $\alpha = a_1a_2\cdots a_n$, the function VISIT( )

- ▷ determines the length p of the aperiodic prefix of α and then
- ▷ outputs $a_pa_{p-1}\cdots a_1$.

Since the aperiodic prefix of α can be determined in $O(n)$ (see [1]), the modified algorithm still runs in $O(n)$-amortized time per symbol. Since the total length of $\mathcal{U}(S)$ is proportional to $n|\boldsymbol{N}(S)|$ (see Section 5 in [16] which implies $|\mathcal{U}(S)| \geq n|\boldsymbol{N}(S)|/2$) we obtain the following theorem.

Theorem 4. *The universal cycle $\mathcal{U}(S)$ for fixed-content strings using their shorthand representation can be generated in $O(1)$-amortized time per symbol using $O(n)$ space.*

References

1. K. S. Booth. Lexicographically least circular substrings. *Information Processing Letters*, 10(4/5):240–242, 1980.
2. F. Chung, P. Diaconis, and R. Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110(1):43–59, 1992.
3. P. Diaconis and R. L. Graham. Products of universal cycles, in *A Lifetime of Puzzles. E. Demaine, M. Demaine and T. Rodgers, eds., A K Peters/CRC Press*, pages 35–55, 2008.
4. H. Fredricksen and I. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Math.*, 61(2):181 – 188, 1986.
5. H. Fredricksen and J. Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Math.*, 23:207–210, 1978.
6. D. Gabric, J. Sawada, A. Williams, and D. Wong. A framework for constructing de Bruijn sequences via simple successor rules. *Discrete Mathematics*, 341(11):2977 – 2987, 2018.
7. D. Gabric, J. Sawada, A. Williams, and D. Wong. A successor rule framework for constructing k -ary de Bruijn sequences and universal cycles. *IEEE Transactions on Information Theory*, 66(1):679–687, 2020.
8. E. N. Gilbert and J. Riordan. Symmetry types of periodic sequences. *Illinois J. Math.*, 5(4):657–665, 12 1961.
9. A. E. Holroyd, F. Ruskey, and A. Williams. Shorthand universal cycles for permutations. *Algorithmica*, 64(2):215–245, 2012.
10. V. Horan and G. Hurlbert. Universal cycles for weak orders. *SIAM J. Discrete Math.*, 27(3):1360–1371, 2013.
11. B. Jackson, B. Stevens, and G. Hurlbert. Research problems on Gray codes and universal cycles. *Discrete Mathematics*, 309(17):5341 – 5348, 2009.
12. A. Leitner and A. Godbole. Universal cycles of classes of restricted words. *Discrete Mathematics*, 310(23):3303 – 3309, 2010.
13. F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex Gray codes. *Journal of Combinatorial Theory, Series A*, 119(1):155–169, 2012.
14. F. Ruskey, J. Sawada, and A. Williams. De Bruijn sequences for fixed-weight binary strings. *SIAM Discrete Math*, 2012, to appear.
15. F. Ruskey and A. Williams. An explicit universal cycle for the (n-1)-permutations of an n-set. *ACM Transactions on Algorithms (TALG)*, 6(3):45, 2010.
16. J. Sawada. A fast algorithm to generate necklaces with fixed content. *Theoretical Computer Science*, 301(1):477–489, 2003.
17. J. Sawada, B. Stevens, and A. Williams. De Bruijn sequences for the binary strings with maximum density. pages 182–190, 2011.
18. J. Sawada, A. Williams, and D. Wong. Universal cycles for weight-range binary strings. pages 388–401, 2013.
19. J. Sawada, A. Williams, and D. Wong. The lexicographically smallest universal cycle for binary strings with minimum specified weight. *Journal of Discrete Algorithms*, 28:31–40, 2014. StringMasters 2012, 2013 Special Issue (Volume 1).
20. J. Sawada, A. Williams, and D. Wong. Generalizing the classic greedy and necklace constructions of de Bruijn sequences and universal cycles. *Electron. J. Comb.*, 23(1):P1.24, 2016.
21. J. Sawada and D. Wong. Efficient universal cycle constructions for weak orders. *Discrete Mathematics*, 343(10):112022, 2020.
22. A. Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 987–996, USA, 2009. SIAM.
23. A. Williams. *Shift Gray codes*. PhD thesis, University of Victoria, 2009.

Appendix - Universal cycles for fixed-content strings using shorthand representation in $O(1)$ time per symbol

```
#include <stdio.h>
int N, K, a[100];
//-----------------------------------------------------------------
// If a[1..n] is a necklace return its period p; otherwise return 0
//-----------------------------------------------------------------
int Necklace() {
    int i,n,p;

    for (i=1; i<=N; i++)  a[N+i] = a[i];
    a[2*N+1] = -1;
    n = 2;
    p = 1;
    while ( a[n-p] <= a[n]) {
        if (a[n-p] < a[n]) p = n;
        n++;
    }
    if (n<=2*N) return 0;
    return p;
}
//-------------------------------
void Visit() {
    int i;
    for (i=Necklace(); i>=1; i--) printf("%d ", a[i]);
}
//-------------------------------
void Swap(int i, int j) {
    int temp;
    temp = a[i];  a[i] = a[j];   a[j] = temp;
}
//----------------------------------
void Gen(int t) {
    int i,j,s;

    i = t;
    while (a[i] != a[1]) {
        while (a[i] == a[i-1]) i--;
        for (j=i; j<=t; j++) {
            Swap(j-1,j);
            if (Necklace()) Gen(j-1);
            else {
                for (s=j; s>=i; s--) Swap(s-1,s);
                Visit();
                return;
            }
        }
        for (j=t; j>=i; j--) Swap(j-1,j);
        i--;
    }
    Visit();
}
//----------------------------------
int main() {
    int i,j,tmp;

    printf("Enter K: "); scanf("%d", &K);
    N = 0;
    for (i=1; i<=K; i++) {
        printf("N_%d: ", i);  scanf("%d", &tmp);
        for (j=1; j<=tmp; j++) a[N+j] = i;
        N += tmp;
    }
    Gen(N);
}
```