COOLER THAN COOL: COOL-LEX ORDER FOR GENERATING NEW COMBINATORIAL OBJECTS

PAUL LAPEY

1. Introduction

1.1. **Combinatorial Generation: Let's Look at all the Possibilities.** Combinatorial generation is defined as the exhaustive listing of combinatorial objects of various types. Frank Ruskey duly notes in his book *Combinatorial Generation* that the phrase "Let's look at all the possibilities" sums up the outlook of his book and the field as a whole [Rus03]. Examining all possibilities fitting certain criteria is frequently necessary in fields ranging from mathematics to chemistry to operations research. Combinatorial generation as an area of study seeks to find an underlying combinatorial structure to these possibilities and utilize it to obtain an algorithm to efficiently enumerate an appropriate representation of them [Rus03].

A quintessential result of the combinatorial generation in practice is Frank Gray's reflected binary code, or Gray code. Gray codes give a "reflected" ordering of binary strings such that each successive string in the ordering differs from the previous string by exactly one bit. This is notably different from a lexicographic ordering of binary strings, in which a n-digit binary string can differ by up to n digits from its predecessor and will differ by approximately two (more precisely $\sum_{i=0}^{n} 2^i$, which is 1.9375 for 4 bit values and 1.996 for 8 bit values) bits on average[1]. The binary reflected Gray code, therefore, provides an ordering that requires as many bit switches as the more intuitive lexicographic order. Binary reflected Gray codes are widely used in electromechanical switches to reduce error and prevent spurious output associated with asynchronous bit switches. Crucially, Frank Gray's reflected binary code achieved a tangible benefit in error reduction through the use of an alternative method of enumerating binary strings. The technique of reflecting all or certain parts of a string to generate new strings has become one of the most widely used techniques in combinatorial generation.

1.2. **Cool-Lex Order.** More recently, cool-lex order has introduced the idea of rotating sublists to enumerate languages. Different versions of cool-lex order have been shown to enumerate several sets of combinatorial objects, including binary strings, fixed weight binary strings, Dyck words, and multiset permutations. Cool-lex orders often lead to algorithms that are faster and simpler than standard lexicographic order. For example, the "multicool" package in R uses a loopless cool-lex algorithm to efficiently enumerate multiset permutations. The package started using cool-lex order for multiset permutations in versoin 1.1 and as of version 1.12 has been downloaded nearly a million times [CWKB21].

[1]Consecutive pairs of binary digits in lexicographic order will differ in the bit at position i with probability $\frac{1}{2^i}$. Therefore, the average number of differing bits between two binary strings of length n is $\sum_{i=0}^{n} 2^i$, which converges to 2 as n grows large.

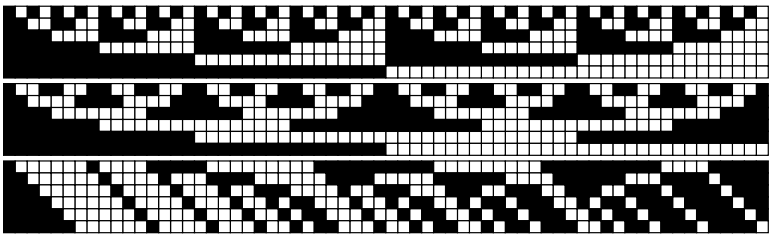

Figure 1. Lexicographic (top), binary reflected Gray code (middle), and cool-lex (bottom) enumerations of 6-bit binary strings. Individual strings are read vertically from top to bottom.

1.3. **Goals of this Thesis.** Cool-lex has been shown to provide a minimal-change cyclic ordering for the sets of fixed-weight binary strings, multiset permutations, binary and k-ary Dyck words, and other languages [Wil09b]. A common thread in the cool-lex algorithms for combinatorial generation is their focus on the *first increase* of string, or the longest prefix of a string such that each successive symbol in the prefix is less than or equal to the previous symbol in the string.

This thesis will examine the use of cool-lex orders to enumerate other languages. Among these are Lukasiewicz, Motzkin, and Schröder paths, which are lattice paths that share similarities with Dyck paths. Shift Gray codes for enumerating these languages have been developed and are given in 1.3.

Dyck, Motzkin, Schröder, and Lukasiewicz paths all share bijections with various combinatorial objects. For example, Dyck paths of length $2n$ share a bijection with binary trees with n nodes. Ruskey and Williams found that the cool-lex successor rule for enumerating Dyck words corresponded directly to a loopless successor rule for enumerating binary trees with a constant number of pointer changes [RW08]. This thesis will examine the efficiency of using cool-lex order to enumerate other sets of combinatorial objects in bijective correspondence with these languages.

TODO: Describe new results of thesis?

2. Background

2.1. **Dyck Words.** The language of binary Dyck words is the set of sequences of binary digits that satisfy the following conditions: The sequence has an equal number of ones and zeroes and there is no prefix of the sequence in which the number of zeroes exceeds the number of ones. The Dyck language can equivalently be thought of as the set of balanced parentheses, with ones representing open parentheses and zeroes representing closing parentheses. In addition to balanced parentheses, Dyck words of length $2n$ are also in bijective correspondence with extended binary trees with n internal nodes. Given an extended binary tree B with n internal nodes, a Dyck word can be obtained by traversing B in preorder and recording each internal node as a 1 and each leaf with a 0, ignoring the final leaf of the tree.

2.2. **Generalizations of Dyck words: Motzkin, Schröder, and Łukasiewicz paths.** Motzkin, Schröder, and Łukasiewicz paths provide generalizations of Dyck words.

In addition to representing balanced parentheses, Dyck paths can be thought of as paths on a cartesian plane. Dyck paths are paths from $(0, 0)$ to $(2n, 0)$ that use $2n$ steps of either $(1, 1)$ (northeast) or $(1, -1)$ (southeast) and never cross below the x axis. In the binary string representation of Dyck words, ones correspond to $(1, 1)$ steps and zerores correspond to $(1, -1)$ steps.

Motzkin paths allow for (1,0) horizontal steps in addition to (1,1) and (1,-1) steps. Schröder paths are identical to Motzkin paths except they allow for $(2, 0)$ horizontal steps instead of $(1, 0)$. Łukasiewicz paths allow (1,-1) steps, (1,0) steps and any (1,k) step where k is a positive integer. All three languages retain the requirement that the path start at the origin, end on the x axis, and never step below the x axis.

These paths can be encoded in a number of different ways. In a *-1-based encoding*, each $(1, i)$ step is encoded as i, and every prefix must have a nonnegative sum. In a *0-based encoding*, each $(1, i)$ step is encoded as $i + 1$, and the sum of every prefix must be as large as its length. We primarily use the 0-based encoding. See Fig. 2 for examples of these paths using the 0-based encoding.

We refer to Motzkin, Schröder, and Lukasiewicz paths ending at $(n, 0)$ as paths of *order n*. This contrasts slightly with the classification of Dyck words of order n, which terminate at $(2n, 0)$

In the context of fixed-content generation, Motzkin and Schröder paths are identical: Both will have northeast steps encoded as twos, horizontal steps encoded as ones, and southeast steps encoded as zeroes. However, their graphical representations Notably, Łukasiewicz are a generalization of Motzkin and Schröder paths, as any Motzkin or Schröder path is also a Lukasiewicz path.

The number of Dyck words with n zeroes and n ones are counted by the nth Catalan number. Similarly, the number of Motzkin and Schröder paths of order n are counted by the nth Motzkin and big Schröder number respectively. The number of Lukasiewicz paths of order n are counted by the n Motzkin, Schröder, and Lukasiewicz paths bear a number of interesting bijective correspondences with other combinatorial objects. Richard Stanely's *Catalan Objects* outlines hundreds of interesting examples.

Lukasiewicz paths of order n bear a particularly nice correspondence to rooted ordered trees with $n + 1$ nodes. See Fig. 3 for an illustration of this.

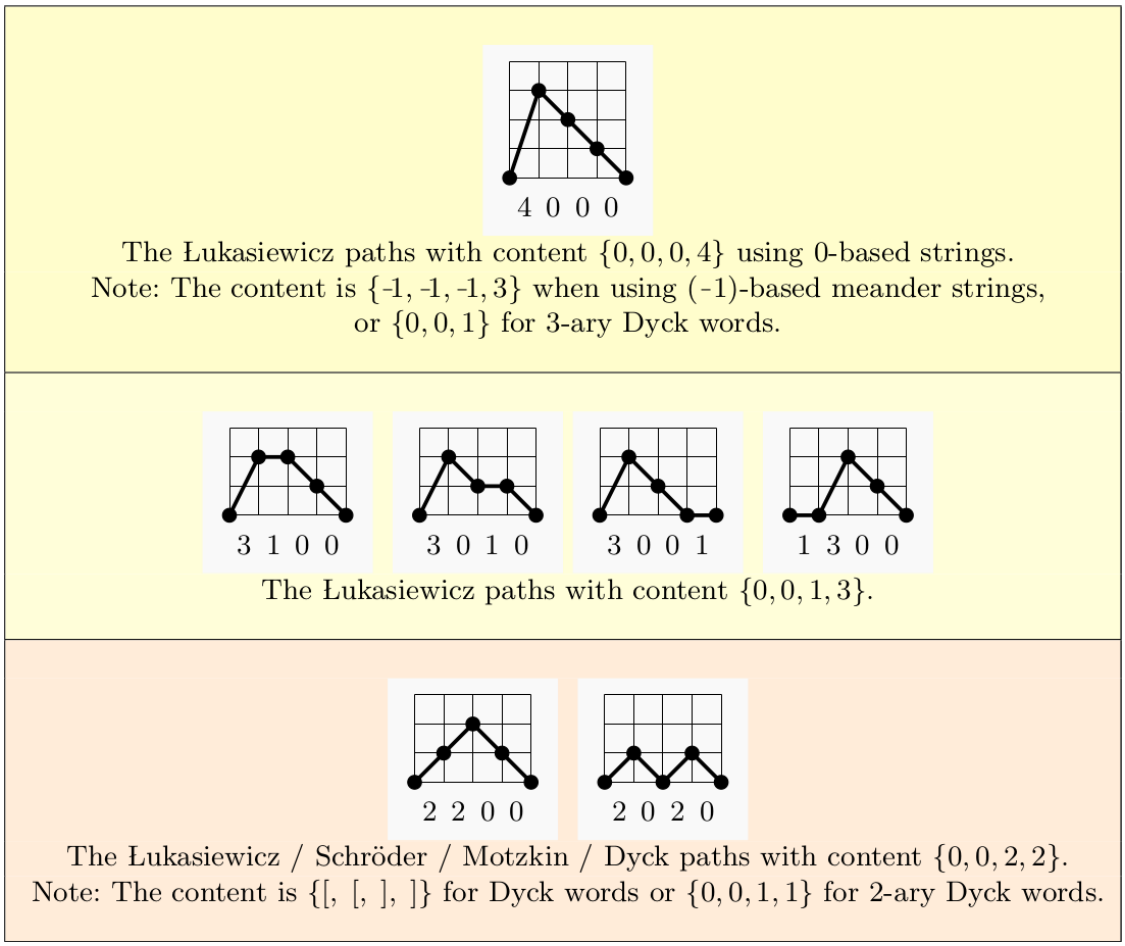

FIGURE 2

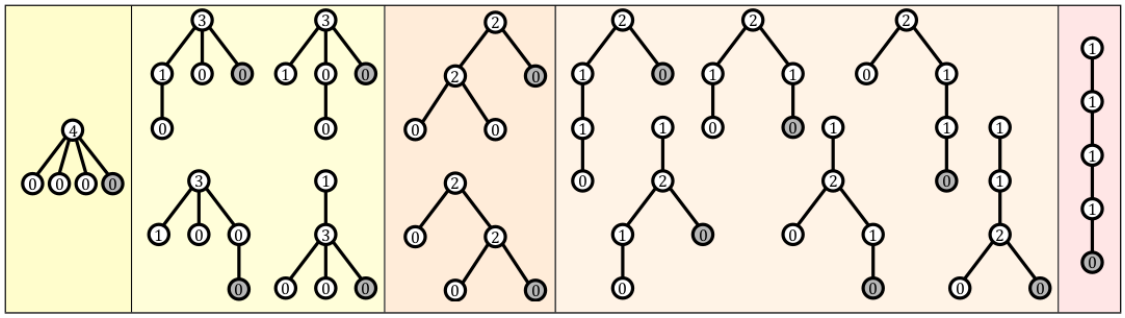

FIGURE 3. The $\mathcal{C}_4$=14 Lukasiewicz paths of order $n = 4$ are in bijective correspondence with the 14 rooted ordered trees with $n + 1 = 5$ nodes. Given a tree, the corresponding word is obtained by recording the number of children of each node in preorder traversal; the zero from the rightmost leaf is omitted. For example, the two trees in the middle section correspond to 2200 (top) and 2020 (bottom) respectively.

2.3. **Cool-Lex Order on Different Combinatorial Objects.**

2.3.1. *Combinations: Fixed-Weight Binary Strings.* Generating all binary strings with s zeroes and t ones is often referred to as combinations, since each string can be used to represent a choice of t elements from a set of size of $s + t$. The cool-lex successor rule for generating all fixed-weight binary strings was given by Aaron Williams in his Ph. D thesis and is as follows[Wil09b]:
Let S be a binary string of length n.

Let y be the position of the leftmost zero in S and x be the position of the leftmost 1 in S such that $x \geq y$. Additionally, note that $S_1...S_{x-1}$ is the non-increasing prefix of S.

Let $\mathsf{left}(S, x)$ be a function that rotates the first i bits of a string S left circularly by one.

More formally, $\mathsf{left}(S, x) = S_2, S_3, ..., S_i, S_1, S_{i-1}, S_{i+1}, S_{i+2}, ..., S_{2n}$

$$\overleftarrow{\mathrm{cool}}(S) = \begin{cases} \mathsf{left}(S, x) & \text{if } S_{x+1} = 1 \\ \mathsf{left}(S, x+1) & otherwise \end{cases}$$

Note that $S_1...S_{x-1}$ must be exactly $1^{y-1}0^{x-y}$, where exponentiation denotes repeated symbols. Because of this, the two left-shift operations can be replaced with can be replaced with either one or two symbol transpositions.

Let $\mathsf{transpose}(S, i, j)$ with $1 \leq i \leq j \leq n$ be a function that swaps S_i an S_j. More formally, $\mathsf{transpose}(S, i, j) = S_1, S_2, \ldots, S_{i-1}, S_j S_{i+1} \ldots S_{j-1} S_i S_{j+1} \ldots S_n$ The left-shift rule can be re-stated as follows:

$$\overleftarrow{\mathrm{cool}}(S) = \begin{cases} \mathsf{transpose}(S, y, x) & \text{if } S_{x+1} = 1 \\ \mathsf{transpose}(\mathsf{transpose}(S, y, x), 1, x+1) & otherwise \end{cases}$$

2.3.2. *Cool Lex Order on Dyck Paths and Binary Trees.* Ruskey and Williams found the following successor rule for enumerating binary Dyck words, dubbed "CoolCat" due to its cool-lex order and (cat)alan numbers [RW08]: We will use $\mathbf{B}_n$ to denote binary Dyck words with n ones and n zeroes. Note that the length of any string in $\mathbf{B}_n$ is thus $2n$.

Let $S \in \mathbf{B}_n$

Let the ith prefix shift of S, denoted by $\mathsf{preshift}(S, i)$, be a function that rotates the second through ith symbols of S one to the right circularly. More formally,

$\mathsf{preshift}(S, i) = S_1, S_i, S_2, ..., S_{i-1}, S_{i+1}, S_{i+2}, ..., S_{2n}$

Let k be the index of the 1 in the leftmost 01 substring in S if it exists. Note that if S has no 01 substring, then $S = 1^n 0^n$. The successor rule for S is as follows:

$$\overrightarrow{\mathsf{coolCat}}(S) = \begin{cases} \mathsf{preshift}(S, 2n) & \text{if } S \text{ has no 01 substring} & \text{(1a)} \\ \mathsf{preshift}(S, k+1) & \text{if } \mathsf{preshift}(S, k+1) \in \mathbf{B}_n & \text{(1b)} \\ \mathsf{preshift}(S, k) & \text{otherwise} & \text{(1c)} \end{cases}$$

Ruskey and Williams's algorithm can also enumerate a broader set of strings: The algorithm enumerates any set $\mathbf{B}_{s,t}$ where for any $S \in \mathbf{B}_{s,t}$ satisfies the constraint that each prefix of S has as many ones as zeroes. This is slightly broader than the language of Dyck words, as it does not have the requirement that a string have an equal number of ones and zeroes. We will focus on $\mathbf{B}_n$ languages due to their correspondce with Dyck words.

Evaluating whether $\mathsf{preshift}(S, k+1) \in B$ can be determined by looking S_{k+1} and the sum of the first k symbols of S:

The above algorithm can be Let $S' = \mathsf{preshift}(S, k+1)$

Note that we know $S \in \mathbf{B}_n$.

Since preshift only rotates symbols, S' will automaticallly satisfy the requirement that strings in $\mathbf{B}_n$ must have an equal number of zeroes and ones since S satisfied that requirement. Thus, $S' \in \mathbf{B}_n$ will be determined by whether or not all prefixes of S' have at least as many ones as zeroes.

If S_{k+1} is a 1, then every prefix i of S' will have at least as many ones as the corresponding ith prefix of S. Thus, S' must be $\in \mathbf{B}_n$, as rotating a 1 to earlier in the string will never invalidate the requirement that every prefix of the string has at least as many ones as zeroes.

Note that the kth prefix of S must be of the form $1^a 0^b 1$, as otherwise there would be an earlier 01 prefix. Fruthermore, $a \geq b$ as otherwise the bth prefix of S would have more zeroes than ones and S would not be a valid Dyck word.

If S_{k+1} is a 0, then $S' \notin \mathbf{B}_n$ if and only if rotating a 0 to index 2 creates a prefix of S with more zeroes than ones. This will only happen if the k-1th prefix is exactly $1^{\frac{k-1}{2}} 0^{\frac{k-1}{2}}$.

Therefore, $\mathsf{preshift}(S, k+1) \in \mathbf{B}_n \iff S_{k+1} = 1$ or S starts with more than $\lfloor \frac{k-1}{2} \rfloor$ ones

$$\overrightarrow{\mathsf{coolCat}}(S) = \begin{cases} \mathsf{preshift}(S, 2n) & \text{if } S \text{ has no 01 substring} & \text{(2a)} \\ \mathsf{preshift}(S, k+1) & S_{k+1} = 1 \text{ or } S \text{ starts with more than } \lfloor \frac{k-1}{2} \rfloor \text{ ones} & \text{(2b)} \\ \mathsf{preshift}(S, k) & \text{otherwise} & \text{(2c)} \end{cases}$$

Since k is the index of the first 01 substring in S, $\sum_{i=1}^{k} S_i$ is actually just the number of consecutive ones to start S, which simplifies the evaluation of this conditional even further.

Ruskey and Williams provided a pseudocode implementation of CoolCat that utilized this fact to enumerate any $\mathbf{B}_{s,t}$ using at most 2 conditionals per successor [RW08].

Due to its simplicity and efficienty, Don Knuth included the cool-lex algorithm for Dyck words in his 4th volume of *The Art of Computer Programming* and also provided an implementation of it for his theoretical MMIX processor architecture [Knu15].

2.3.3. *Multiset Permutations.* Cool-lex order has also been shown to enumerate multiset permutations via prefix shifts. The rule given by Williams is as follows [Wil09a]:

Let S be a multiset of length n.
Let i be the maximum value such that $S_{j-1} \geq s_j$ for all $2 \leq j \leq i$. In other words, i is the length of the non-increasing prefix of S.
Let $\sigma_j(S)$ be a function that shifts the ith value of S into the first position, or equivalently rotates the first i elements of S right circularly. More formally,
$\sigma_j(S) = S_j, S_1, S_1, \ldots, S_{j-1}, S_j + 1, \ldots, S_n$
Then

$$\text{nextPerm}(S) = \begin{cases} \sigma_{i+1}(S) & \text{if } i \leq n-2 \text{ and } s_{i+2} > s_i \\ \sigma_{i+2}(S) & \text{if } i \leq n-2 \text{ and } s_{i+2} \leq s_i \\ \sigma_n(S) & \textit{otherwise} \end{cases}$$

See Fig. **??** for an example comparison of cool-lex and lexicographic order for two multisets.

This successor rule has the nice property of ensuring that length of the successor's non-increasing prefix is easy to find.

In particular, if S_{i+2} is shifted, then the length of the non-increasing prefix is either 1 if $S_{i+2} \leq S_1$ or $i+1$ otherwise.

Similarly, if S_{i+1} is shifted, then the length of the non-increasing prefix is either 1 if $S_{i+1} \leq S_1$ or $i+1$ otherwise.

This allows for a loopless implementation of the successor rule, as scanning the string to find the length of the non-increasing prefix is not required. Due to the simplicity and efficiency of this rule, it is used in the "multicool" package in R, which is used for generating multiset permutations, Bell numbers, and other combinatorial objects [CWKB21]. Further information on the package is available here: https://www.rdocumentation.org/packages/multicool/versions/0.1-12

TODO: Common threads among cool-lex order. Non-increasing prefix.

3. New Results

This thesis provides successor rules and implementations for enumerating the following languages: Ordered trees with a fixed number of nodes, Lukasiewicz words with fixed content, and Motzkin/Schroder words with fixed content. The algorithm for ordered trees is loopless and The algorithm for enumerating Lukasiewicz paths also provides a generalization of the cool-lex successor rule for multiset permutations, given in section 2.

3.1. **Loopless Ordered Tree Generation.** Ruskey and Williams previously gave a cool-lex successor rule for looplessly generating all Dyck words of a given length via prefix shifts [RW08]. The successor rule was also shown to enumerate all binary trees with a fixed number of nodes, in accordance with the bijection between Dyck words of order n and binary trees with n nodes.

This thesis provides an additional adaptation of the successor rule to looplessly generating ordered trees with a fixed number of nodes via node shifts. Notably, this algorithm generates ordered trees stored as pointer structures, rather than degree sequences as in other ordered tree generation algorithms. This facilitates the practical use of the trees generated by this algorithm, as a translation step between degree sequences and ordered trees is not necessary.

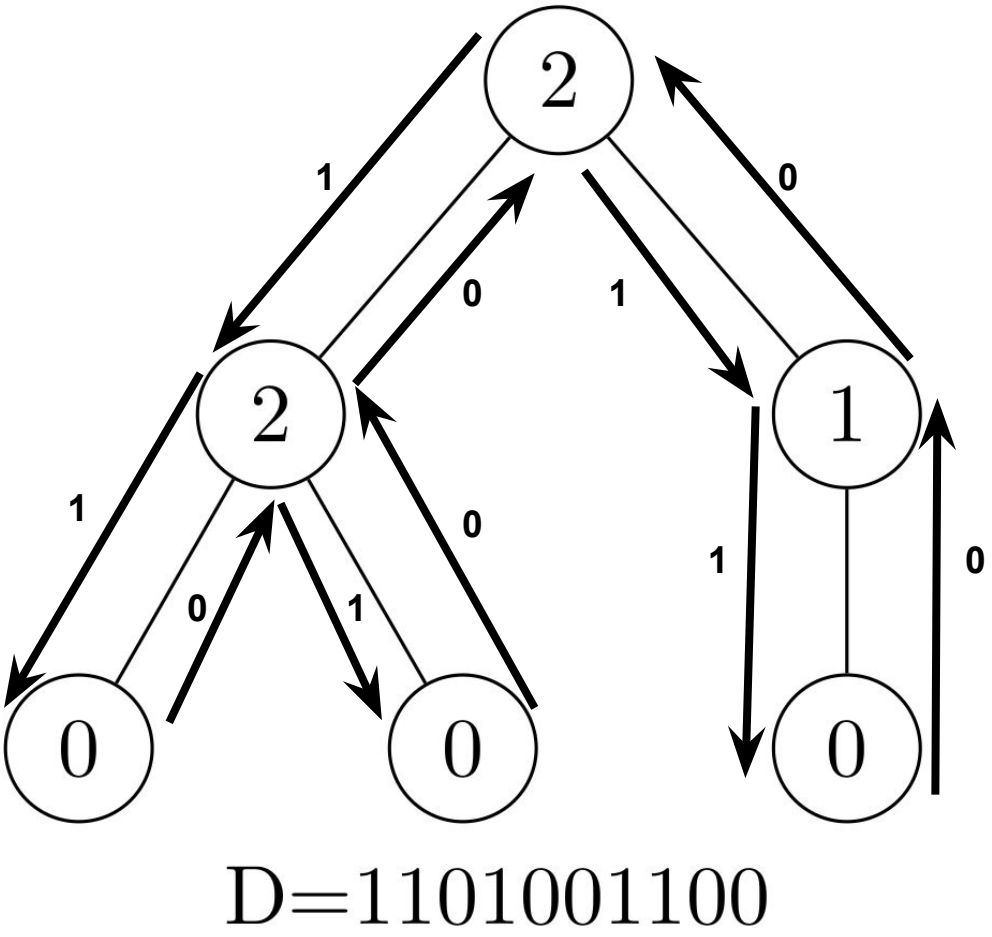

FIGURE 4. An ordered tree with $5 + 1 = 6$ nodes corresponding to the order 5 Dyck word 1101001100.

3.1.1. *Ordered Trees $\iff$ Dyck Words.* This algorithm will use the bijection between ordered trees and Dyck words specified in [Sta15]. The bijection described by Stanely is as follows:[2]

Given an ordered tree T, with $n + 1$ nodes: Traverse T in preorder. Whenever going "down" an edge, or away from the root, record a 1. Whenever going "up" an edge, or towards the root, record a 0. The resulting binary sequence is a Dyck word D corresponding to the ordered tree T.

This process can be inverted as follows:

Let $D = d_1...d_{2n}$ be a dyck word of order n with $n > 0$. Construct an ordered tree T via the following steps.

Create a root node of T. Keep track of a current node *curr*; set $curr = root$.

- For each d_i such that $1 \leq i \leq 2n$
  - if $d_i = 1$: append a rightmost child *ch* to *curr*'s children; set $curr = ch$
  - if $d_i = 0$, set *curr* equal to *curr*'s parent.

Figure 4 demonstrates both directions of this process. Note that each t_i with $1 \leq i \leq n$ in a preorder traversal of T corresponds to the $i^{\underline{th}}$ 1 in D.

Let $\mathsf{OTree}(D)$ and $\mathsf{Dyck}(T)$ be functions that convert a Dyck word to an ordered tree and an ordered tree to a Dyck word respectively via the above process.

[2]Stanley's text refers to ordered trees as *plane trees* and Dyck words as *ballot sequences*

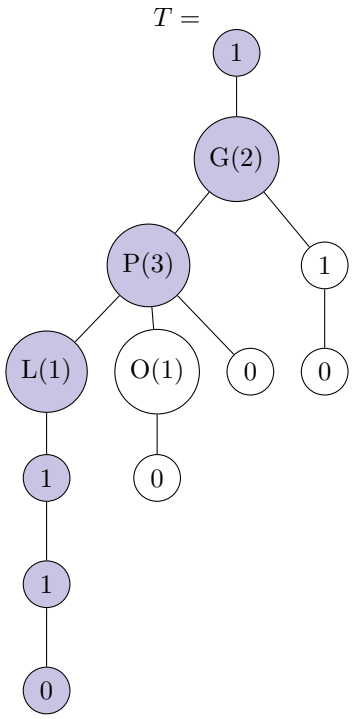

FIGURE 5. An ordered tree with 12 nodes corresponding to the Dyck word 111111000011001011000. Each node is labelled with its number of children; the left down path of T is highlighted in purple. Nodes are labelled by their degree.

3.1.2. *Successor Rule.* Define the left-down path of an ordered tree T, denoted $\mathsf{LD}(T)$ to be the first $m+1$ nodes in a preorder traversal of T such that, for each t_i with $0 \leq i \leq m$, $i = 0$ or t_i is the leftmost child of t_{i-1}.

Given an ordered tree T, let O be the first node in a preorder traversal of T that is not in the left-down path of T. If there is no such node, in the case where the tree is a single path, let O be the tree's (only) leaf node. Let P be O's parent. Let G be P's parent, and let L be P's leftmost child (or, equivalently, O's left sibling). The labels P, G, and L are mnemonics for O's (p)arent, (g)randparent, and (l)eft sibling. Fig. 5 gives an example illustrating O,P,G,L, and the left-down path in an tree.

The successor rule for enumerating ordered trees with n nodes can be stated as folows:

$$\mathsf{nextree}(T) = \begin{cases} \mathsf{shiftree}(T, O, root) & \mathsf{LD}(T) = T & \text{(3a)} \\ \mathsf{shiftree}(\mathsf{shiftree}(T, L, G), O, root) & \text{if } P \neq root\text{; O has no children} & \text{(3b)} \\ \mathsf{shiftree}(T, L, O) & \text{otherwise} & \text{(3c)} \end{cases}$$

3.1.3. *Proof of Correctness.* – still somewhat of an outline.

HELPFUL FUNCTIONS

$\mathsf{Depth}(t_i)$ = length of path between root and t_i. $\mathsf{Depth}(root) = 0$

$\mathsf{i^{\underline{th}}one}(D, i)$ = the index of the $i^{\underline{th}}$ one in D.

Let $\mathsf{DyckIndex}(N)$ be a function that converts a ordered tree node n to the index of its corresponding 1 in $\mathsf{Dyck}(T)$. Similarly,

HELPFUL LEMMAS

(1) t_i corresponds to the ith one in D for $1 \leq i \leq n$ This is due to the nature of the bijection between Dyck words and ordered trees. Each one in D creates a new node; zeroes in D do not create nodes. Generating an ordered tree from a Dyck word generates the nodes of the tree in preorder. Thus, t_i corresponds to the ith one in D for $1 \leq i \leq n$.

(2) $\mathsf{Depth}(t_i) = \mathsf{Depth}(t_{i-1}) + 1 - (\mathsf{i}^{\underline{\mathsf{th}}}\mathsf{one}(D, i) - \mathsf{i}^{\underline{\mathsf{th}}}\mathsf{one}(D, i-1) - 1)$
Note that $(\mathsf{i}^{\underline{\mathsf{th}}}\mathsf{one}(D, i) - \mathsf{i}^{\underline{\mathsf{th}}}\mathsf{one}(D, i-1) - 1)$ is equal to the number of zeroes between the $i^{\underline{th}}$ and $(i-1)^{\underline{st}}$ ones in D.
Informally, depth of t_i is the depth of t_{i-1} plus 1 minus the number of zeroes between t_{i-1} and t_i
This also follows naturally from the bijection between Dyck words and ordered trees. Each zero corresponds to a step up in the tree before adding the next child.
If there are zero zeroes between the $i^{\underline{th}}$ and $(i-1)^{\underline{st}}$ ones in D, t_i is a child of t_{i-1}; $\mathsf{Depth}(t_i) = \mathsf{Depth}(t_{i-1}) + 1$
If there is one zero between the $i^{\underline{th}}$ and $(i-1)^{\underline{st}}$ ones in D, t_i is a child of t_{i-1}'s parent; $\mathsf{Depth}(t_i) = \mathsf{Depth}(t_{i-1}) + 1$.
Each subsequent zero between t_{i-1} and t_i decreases $\mathsf{Depth}(t_i)$ by one. Thus, the depth of t_i is the depth of t_{i-1} plus 1 minus the number of zeroes between t_{i-1} and t_i.

(3) $\mathsf{Depth}(O) = m - j + 1$
t_m is the last node in the $\mathsf{LD}(T)$, as the left-down path has m+1 nodes. It has depth m, as it is exactly m left-down steps from the root. Note that $O = t_{m+1}$. The number of zeroes between t_m and t_{m+1} is the number of zeroes between the $m^{\underline{th}}$ and $(m+1)^{\underline{st}}$ ones in D_i.

(4) Preorder listing of T $\implies$ D
Let $T = t_0, t_1, ...t_n$ be a preorder traversal of T. Note that t_0 is the root of T
Construct D as follows:
- skip t_0
- Let $D = \epsilon$
- For each t_i, $1 \leq i \leq n$
  - Append a 1 to D
  - Append $1 - \mathsf{Depth}(t_{i-1}) + \mathsf{Depth}(t_i)$ zeroes to D.

(5) Every non-leaf node below P in $\mathsf{LD}(T)$ has exactly 1 child.
If a node below P in $\mathsf{LD}(T)$ had a second child, that child would not be in $\mathsf{LD}(T)$ and would be be traversed before O in preorder, which would be a contradiction.

Ruskey and Williams proved that, given a Dyck word of order n, 2 iteratively generates all Dyck words of order n. This proof will use the bijection between Dyck words of order n and ordered trees with $n+1$ nodes to show that that 3 generates all ordered trees with a given number of nodes.

To prove that $\mathsf{nextree}(T)$ generates all ordered trees with $|T|$ nodes, it is sufficient to show that, given a Dyck word D and its corresponding ordered tree T,
$\overrightarrow{\mathsf{coolCat}}(D) = \mathsf{Dyck}(\mathsf{nextree}(T))$

In other words, we aim to show that $\overrightarrow{\mathsf{coolCat}}(D) = \mathsf{Dyck}(\mathsf{nextree}(\mathsf{OTree}(D)))$
First, consider the node O in the specification of the $\mathsf{nextree}(T)$ algorithm.
Let $D = \mathsf{Dyck}(T)$ and let k be the index of the 1 in the leftmost 01 substring of D. Let $t_0...t_m = \mathsf{LD}(T)$; $O = t_{m+1}$. We will show that O corresponds to D_k.
(This can be done better)
Note that each 1 in D corresponds to a step down; each 0 to a step up. Consequently, $\mathsf{LD}(T)$ corresponds to the "all-one" prefix of D. In other words, $\mathsf{LD}(T) = t_0, t_1, ...t_m$ such that $i = 0$ or $D_{\mathsf{DyckIndex}(t_i)} = 1$. Note that t_{m+1} is therefore the first node in a preorder traversal of T such that $D_{\mathsf{i^{th}one}(D,m+1)} = 1$ and $D_{\mathsf{i^{th}one}(D,m+1)-1} = 0$ and the first node in a preorder traversal of T such that $t_m \notin \mathsf{LD}(T)$. Therfore, $\mathsf{i}^{\underline{\mathsf{th}}}\mathsf{one}(D, m+1) = k$, i.e., $t_{m+1} = O$ corresponds to the 1 in the leftmost 01 substring of D.
$\overrightarrow{\mathsf{coolCat}}(D)$ and $\mathsf{nextree}(T)$ are each broken down into 3 cases in equations 2 and 3 respectively.

$$\overrightarrow{\mathsf{coolCat}}(D) = \begin{cases} \mathsf{preshift}(D, 2n) & \text{if } D \text{ has no 01 substring} \quad (4a) \\ \mathsf{preshift}(D, k+1) & D_{k+1} = 1 \quad (4b) \\ \mathsf{preshift}(D, k+1) & D_{k+1} = 0 \text{ and } D \text{ starts with more than } \lfloor \frac{k-1}{2} \rfloor \text{ ones} \quad (4c) \\ \mathsf{preshift}(D, k) & \text{otherwise} \quad (4d) \end{cases}$$

For convenience, equation 4 gives an expanded version of $\overrightarrow{\mathsf{coolCat}}(D)$
We will show the following equivalences:

(1) 3a corresponds to 2a
(2) 3b corresponds to 4c
(3) 3c corresponds to 4b and 4d

To show these, first we show a few auxillary equivalences:

Let m be the number of consecutive ones to start D, let j be the number of consecutive zeroes starting at d_{m+1}. Note that $j = (k - m - 1)$; $d_k = 1$

- D has no 01 substring $\iff LD(T) = T$
  If D has no 01 substring, $D = 1^n0^n$, and T is $n + 1$ nodes where t_0 is the root and each t_i for $1 \leq i \leq n$ is a child of t_{i-1} In this case, T is essentially a path of $n + 1$ nodes, and the left-down path of T is the entire tree.
- $D_{k+1} = 0 \iff O$ has no children
  This follows logically from the bijection between Dyck words and ordered trees. D_k corresponds to O. If $D_{k+1} = 0$, an "upward" step is taken after O and consequently the next node after O cannot be a child of O. Since the ones in D give the nodes of T in preorder, O must have no children.
  Informally, once you go "up" from O, the bijection between Dyck words and ordered trees gives no way to go "back down" to give O an additional child.
- $P = root \iff D$ starts with exactly $\frac{k-1}{2}$ ones.
  First, note that $P = root$ simply means that O is a child of the root.
  Next, note that the number of nodes in $\mathsf{LD}(T)$ is equal to the number of consecutive ones at the start of D. Let m be the number of consecutive ones at the start of D. Note that $k - m - 1$ is the number of consecutive zeroes following d_m, since d_k is the index of the 1 in the first 01 substring of D. Since each of the m ones constitutes a step down from the root in T and each of the $k - m - 1$ zeroes starting at d_m corresponds a step up towards the root in T, $m = k - m - 1 \iff$ O is a child of the root.
  This equivalence can can be rewriten as follows:
  $m = k - m - 1$
  $2m = k - 1$
  $m = \frac{k-1}{2}$
  Therefore, $P = root \iff D$ starts with $\frac{k-1}{2}$ ones.
- L corresponds to D_{m-j+1}

(1) 3a corresponds to 4a
  - It was previously shown that D has no 01 substring $\iff LD(T) = T$.
    Thus, $\mathsf{nextree}(T)$ executes case 3a if and only if $\overrightarrow{\mathsf{coolCat}}(D)$ executes case 4a
    TODO: show the shifts are equivalent.

(2) 3b corresponds to 4c
  - It was previously shown that $P = root \iff D$ starts with exactly $\frac{k-1}{2}$ ones.
    It was also previously shown that $D_{k+1} = 0 \iff O$ has no children. Thus, $\mathsf{nextree}(T)$ executes case 3b if and only if $\overrightarrow{\mathsf{coolCat}}(D)$ executes case 4c
  - We now show that the execution of 3b is equivalent to the execution of 4c.
    Given $\mathsf{Dyck}(T) = D = 1^m0^j10d_{k+2}d_{k+3}...d_{2n}$, we aim to show that
    $\mathsf{Dyck}(\mathsf{nextree}(T)) = \overrightarrow{\mathsf{coolCat}}(\mathsf{Dyck}(T))$

    Note that $\mathsf{nextree}(T) = \mathsf{shiftree}(\mathsf{shiftree}(T, L, G), O, root)$.
    Let $T' = \mathsf{shiftree}(T, L, G)$; $T'' = \mathsf{shiftree}(T', O, root)$
    Note that $\mathsf{nextree}(T) = T''$
    Since $P \neq root$, we know that G, the parent of P, exists. Thus, we can assume that $G, P, L \in \mathsf{LD}(T)$. Furthermore, recall that L (and all other non-leaf nodes $\in \mathsf{LD}(T)$ must have exactly one child. Therefore, every node below L in $\mathsf{LD}(T)$ has its depth reduced by one; no other nodes have their depth affected by this shift.

    $T =$

node	t_0	t_1	$\dots$	$G = t_{m-j-1}$	$P = t_{m-j}$	$L = t_{m-j+1}$	$\dots$	t_m	$O = t_{m+1}$	$\dots$
depth	0	1	$\dots$	$(m-j-1)$	$(m-j)$	$(m-j+1)$	$\dots$	m	$(m-j+1)$	$\dots$
Dyck		1^m							0^j1	$0\dots$

$T' =$

node	t_0	t_1	$\dots$	$G = t_{m-j-1}$	$L = t_{m-j+1}$	$\dots$	t_m	$P = t_{m-j}$	$O = t_{m+1}$	$\dots$
depth	0	1	$\dots$	$(m-j-1)$	$(m-j)$	$\dots$	$m-1$	$(m-j)$	$(m-j+1)$	$\dots$
Dyck		1^{m-1}						0^j1	1	$0\dots$

This operation makes P G's second child, consequently removing P from the left-down path of T' and making P the first node in a preorder traversal of T' that is not in the left-down path of T'. Therefore, $|\mathsf{LD}(T')| = m-1$; $O' = P$.
Note in the case where $j = 1$, $L = t_m$; i.e. L is the leaf of the left-down path of T.
Recovering a Dyck word from T', we obtain
D'=$1^{m-1}0^j110d_{k+1}d_{k+3},\dots,d_{2n}$
Next, we use $\mathsf{shiftree}(T', O, root)$ to obtain $T'' = \mathsf{nextree}(T)$
$\mathsf{shiftree}(T', O, root)$ shifts O to become the first child of the root. Note that we know that O has no children. Consequently, no nodes other than O have their depth affected by this shift. Thus,

$T'' =$

node	t_0	$O = t_{m+1}$	t_1	t_2	$\dots$	$G = t_{m-j-1}$	$L = t_{m-j+1}$	$\dots$	t_m	$P = t_{m-j}$	$\dots$
depth	0	1	1	2	$\dots$	$(m-j-1)$	$(m-j)$	$\dots$	$m-1$	$(m-j)$	$\dots$
Dyck		1	01^{m-1}							0^j1	$\dots$

Therefore, since $T'' = \mathsf{nextree}(T)$, $\mathsf{Dyck}(\mathsf{nextree}(T)) = 101^{m-1}0^j1\dots$
Since $\mathsf{Dyck}(T) = D = 1^m0^j10\dots$ 4b gives that
$\overrightarrow{\mathsf{coolCat}}(\mathsf{Dyck}(T)) = 101^{m-1}0^j1\dots$
Therefore, we have shown that $\mathsf{Dyck}(\mathsf{nextree}(T)) = \overrightarrow{\mathsf{coolCat}}(\mathsf{Dyck}(T)) = 101^{m-1}0^j1\dots$

(3) 3c corresponds to 4b and 4d TODO

3.1.4. *Loopless Implementation.* todo

References

[CWKB21] James Curran, Aaron Williams, Jerome Kelleher, and Dave Barber. Package 'multicool', Jun 2021.

[Knu15] Donald Ervin Knuth. The art of computer programming: Combinatorial algorithms, vol. 4, 2015.

[Rus03] Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.

[RW08] Frank Ruskey and Aaron Williams. Generating balanced parentheses and binary trees by prefix shifts. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 107–115, 2008.

[Sta15] Richard P Stanley. *Catalan numbers*. Cambridge University Press, 2015.

[Wil09a] Aaron Williams. Loopless generation of multiset permutations by prefix shifts. In *SODA 2009, Symposium on Discrete Algorithms*, 2009.

[Wil09b] Aaron Michael Williams. *Shift gray codes*. PhD thesis, 2009.