Tree Traversals and Permutations

Todd Feil, Kevin Hutson, R. Matthew Kretchmar

Abstract

We build upon the previous work on the subset of permutations known as stack words and stack-sortable words. We use preorder, inorder and postorder traversals of binary trees to establish multiple bijections between binary trees and these words. We show these operators satisfy a sort of multiplicative cancellation. We further expand the study of these operators by demonstrating how properties on trees are related to properties on words.

1 Introduction

Knuth [9] considered permutations created by using a stack. Consider passing the sequence $1, 2, 3, \ldots, n$ through a single stack by means of a series of pushes and pops, where a push will push the next number in the input string and a pop removes the top of the stack to the output string, with the proviso that the stack empties after all integers have been pushed. For example, if $n = 4$, then the sequence, push, push, pop, push, pop, push, pop, pop, would result in the permutation (2341). (Our convention is to represent the permutation π with the sequence $\pi(1)$, $\pi(2)$,..., $\pi(n)$. Thus, if $\pi = (2341)$ then $\pi(1) = 2$, $\pi(2) = 3$, and so on.) Figure 1 shows midway through the creation of the permutation (2341).

It is easy to see that a permutation obtained in this way is a 312-avoiding permutation (For example, see Grimaldi [7].); that is, a permutation $\pi(1)$, $\pi(2)$,..., $\pi(n)$ for which there is no $i < j < k$ where $\pi(j) < \pi(k) < \pi(i)$. A permutation π obtained in this manner is called a *stack word.* For example, the permutation $\pi = (25134)$ is not 312-avoiding since $\pi(2) = 5$, $\pi(4) = 3$ and $\pi(5) = 4$.

Likewise, a permutation that can be sorted using a stack in the manner described is called a *stack-sortable word.* These are precisely the 231-avoiding permutations; that is, a permutation π for which there is no $i < j < k$ where $\pi(k) < \pi(i) < \pi(j)$.

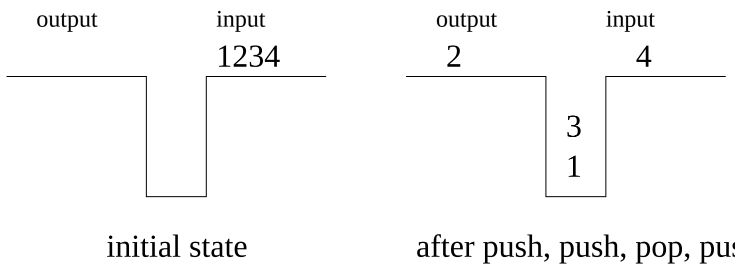


Figure 1: Creating a stack word

Recently, there has been interest in stack words, restricted permutations, stack-sortable words, and their extensions. (See [1], [2], [3], [4], [8], [9], [10], [14], [16], [17], and [18].) Several have related these restricted permutations to various structures, for example, stacks. While bijections between stack words (or stack-sortable words) and binary trees have been given previously (see [4], [6], [15], and especially [9], [12], and [13]), we give additional bijections in which the structure of the tree reflects the stack operations in a straightforward manner.

These stack and stack-sortable words can be produced by applying various traversals of binary trees. This idea has been used by [9], [12], [13], and [4]. In effect, we are modeling stack operations with these traversals. We apply the three common binary tree traversal algorithms of preorder, inorder and postorder to first label the tree, then read the labels. The order the labels are read gives us a permutation. That is, for a given binary tree T, we pick a labeling traversal from preorder, inorder and postorder traversals. We then pick a reading traversal from among these three options. Thus, we produce a permutation by:

1. Labeling the nodes of T with $1, 2, \ldots, n$ using the chosen labeling traversal.
2. Reading the labels from the nodes of T produced in step (1) using the reading traversal.

Note for a given binary tree T, each choice of labeling and reading traversal produces a potentially different permutation of $1, 2, \ldots, n$. Obviously, if the labeling and reading traversals are the same, the resultant permutation is the identity permutation. If a permutation is obtained from a binary tree by labeling with a preorder traversal and reading with an inorder traversal, we'll call it a *PreIn permutation.* Permutations obtained by this process using different labeling and reading traversals will be similarly named.

In Section 2 we show a bijection between stack words and binary trees which are labeled using an inorder traversal and read using a postorder traversal. That is, we can convert stack words to trees and vice versa. Section 3 shows how an inorder labeling and a preorder reading produces stack-sortable words. Furthermore, we can produce stack words and stack sortable words with two other label/read traversals.

Section 5 reveals an interesting "canceling" feature of these traversal operators when they are composed. The preorder labeled and postorder read permutations possess some properties not present in the other stack words; these are discussed in Section 6.

We can use a combination of operators to convert from a binary tree to a stack word and then back to another (different) binary tree. If we repeat this process, we create a series of trees (and their intervening stack words). This process creates cycles that partition the set of stack words. We develop an equivalence relation in Section 7.

2 Stack Words, Stack-sortable Words and Binary Trees

Notice that the sequence of pushes and pops necessary to produce a stack word (or sort a stack-sortable word) can be conveniently given by a sequence of n pairs of parentheses, where a left parenthesis corresponds to a push and a right parenthesis corresponds to a pop. (This was observed by West [16].) For example, the sequence of pushes and pops producing (2341) can be expressed as (()()()). If you label the parentheses pairs starting at 1, then reading off the label of the right parentheses will give the corresponding stack word. In our example $(_1(_2)_2(_3)_3(_4)_4)_1$ yields 2341. This establishes a bijection between the set of stack words of length n and the set of n pairs of parentheses.

If π is a stack word, that is, a 312-avoiding permutation of $\{1, 2, \ldots, n\}$, then clearly π^{-1} is a stack-sortable word; that is, a 231-avoiding permutation. Furthermore, the sequence of parentheses (sequence of pushes and pops) that sorts π^{-1} is precisely the sequence of parentheses that creates π. Thus, the stack word we obtained above, 2341 corresponds to the stack-sortable word 4123, which is sorted by the sequence of stack operations given by (()()()).

Now consider binary trees. We describe a bijection between binary trees with n nodes and sequences of n pairs of parentheses. There are many such bijections, of course. For example, see [4] and [6]. The one we give has the advantage of naturally extending to stack words and stack-sortable words. We do this recursively. The empty tree corresponds to zero pairs of parentheses; that is, the empty sequence. A tree of one node corresponds to the sequence (). A tree with more than one node corresponds to the sequence $L(R)$, where L is the sequence of parentheses corresponding to the tree rooted at the left child of the root node and R is the sequence of parentheses corresponding to the tree rooted at the right child of the root node. Some examples are given in Figure 2.

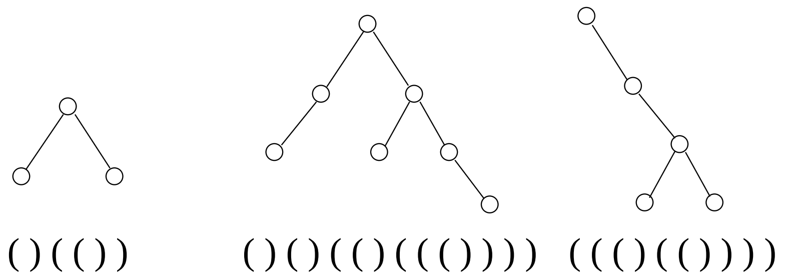


Figure 2: Binary trees and nested parentheses

The stack word associated with a binary tree T with n vertices can also be realized by labeling the vertices of T with $1, 2, \ldots, n$ using an inorder traversal and then reading the labels using a postorder traversal. Call such a word an *inorder labeled, postorder read word for the tree T* and denote the permutation by $[In : Post]_T$. (This idea is similar to that given in [4].) We formalize this in the following proposition.

Proposition 1 *There is a bijection between the inorder labeled, postorder read words and stack words.*

Proof: Since stack words are precisely the 312-avoiding permutations, we show that a word is inorder labeled, postorder read if and only if it is 312-avoiding. In a binary tree, we'll say a is a *right descendant* of b if the node labeled a is in the subtree rooted at the right child of the node labeled b. *Left descendant* is similarly defined.

Suppose π is a permutation with a 312 subsequence. Hence there exist i, j, k where $i < j < k$ with $\pi(j) < \pi(k) < \pi(i)$. Now if $\pi(j) < \pi(i)$ on an inordered labeled tree, then one of the three situations hold: (1) $\pi(i)$ is a right descendant of $\pi(j)$, (2) $\pi(j)$ is a left descendant of $\pi(i)$, or (3) there is some node x where $\pi(i)$ is a right descendant of x and $\pi(j)$ is a left descendant of x. These three situations are pictured in Figure 3.

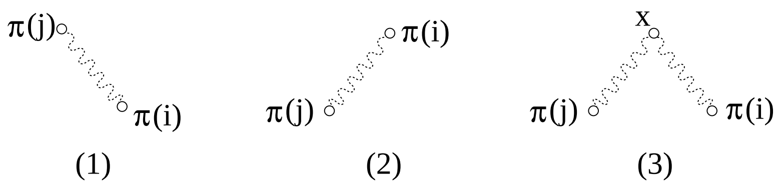


Figure 3: Possible trees for 312 subsequence

But if $i < j$ in the postorder output, only situation (1) can hold. Now, restricting our attention to the situation (1), if $\pi(j) < \pi(k) < \pi(i)$ either (a) $\pi(k)$ is a right descendant of $\pi(j)$ and $\pi(i)$ is a right descendant of $\pi(k)$, or (b) $\pi(k)$ is a left descendant of $\pi(i)$. These are pictured in Figure 4.

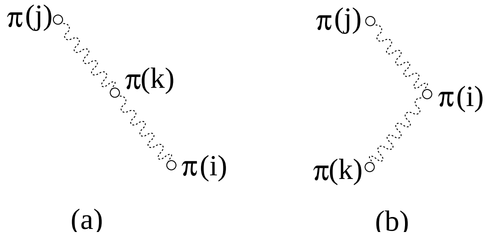


Figure 4: Possible trees if $\pi(j) < \pi(k) < \pi(i)$

But in postorder output, (a) yields the order $\pi(i), \pi(k), \pi(j)$ and (b) yields $\pi(k), \pi(i), \pi(j)$. In either case, this is not the order desired ($\pi(i)$, $\pi(j)$, $\pi(k)$). Thus, there is no binary tree T where $[In : Post]_T = \pi$.

Now suppose π is a 312-avoiding permutation. We construct the binary tree that is inorder labeled, postorder read that yields π. Since we have a postorder read, label the root node $\pi(n)$ (the last number in the permutation). Due to the inorder labeling, the right subtree of $\pi(n)$ will be all nodes labeled $\pi(k)$ with $\pi(k) > \pi(n)$ and the left subtree of $\pi(n)$ will be all nodes labeled $\pi(j)$ where $\pi(j) < \pi(n)$. Let $L = \{\pi(j) : \pi(j) < \pi(n)\}$ and $R = \{\pi(k) : \pi(k) > \pi(n)\}$. We claim that if $\pi(j) \in L$ and $\pi(k) \in R$, then $j < k$. That is $\pi(j)$ comes before $\pi(k)$ in the permutation π. For otherwise we have $k < j < n$ and $\pi(j) < \pi(n) < \pi(j)$, which is a 312 subsequence.

We now apply this construction recursively to L and R. Note a postorder reading of the tree will output L, then R, and then $\pi(n)$. □

Given the bijection between binary trees and nested parentheses noted before this proposition, one can see there is also a bijection between nested parentheses and stack words.

If π is a stack word, call the binary tree T such that $[In : Post]_T = \pi$ the *InPost tree of* π. We give an example in Figure 5 of the conversion from a stack word to an inorder labeled, postorder read tree, as given in the proof of Proposition 1.

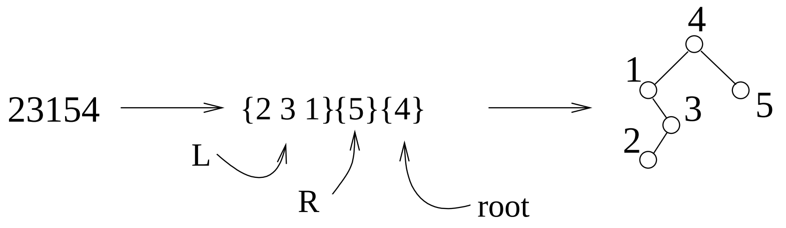


Figure 5: 312-avoiding sequence to InPost tree

Note that if $[In : Post]_T = \pi$ and we label the nodes of the same binary tree T inorder with the labels π^{-1}, then a postorder reading of the nodes will yield the nodes in sorted order $1, 2, \ldots, n$. We observed this phenomenon before when we noted that the same stack operations sort π^{-1} as create π.

One can realize π^{-1} by labeling the nodes of the InPost tree of π with $1, 2, \ldots, n$ using a postorder labeling and reading the labels in an inorder traversal. (Similar to our previous notation, we'll denote such a permutation by $[Post : In]_T$.) We illustrate with the InPost tree of $\pi = (23154)$ in Figure 6.

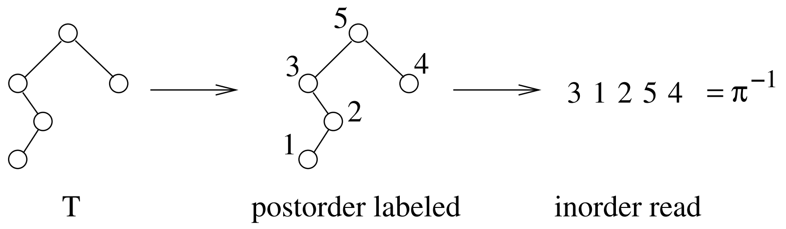


Figure 6: Postorder label, inorder read

This is true since if we label the InPost tree of π using a postorder labeling, then the label $\pi(i)$ in the original labeling is replaced by i in the new labeling for each i. Thus each i in the original inorder labeling will be replaced by $\pi^{-1}(i)$. Now reading the new labels using an inorder traversal will produce labels in the order $\pi^{-1}(1)$,$\pi^{-1}(2)$,$\pi^{-1}(3)$,$\ldots$, $\pi^{-1}(n)$.

3 Stack-sortable Words and Preorder Traversals

In light of the above proposition, it is natural to ask what is the permutation that results from an inorder labeled, preorder read tree. Adopting notation similar to above, we denote a permutation obtained in this way by $[In : Pre]_T$. The following proposition answers this, which has been noted in section 2.3.1, exercise 6 of [9] and further exploited in [13] and [12].

Proposition 2 *There is a bijection between the inorder labeled, preorder read words and stack-sortable words.*

Proof: The proof is similar to the proof of Proposition 1, noting that stack-sortable words are precisely those 231-avoiding permutations. When showing that a 231-avoiding permutation, ρ, can be realized as an inorder labeled, preorder read tree, we label the root with $\rho(1)$, the leftmost number of the permutation, and note that all $\rho(j)$ with $\rho(j) < \rho(1)$ come before all $\rho(k)$ with $\rho(k) > \rho(1)$. As before, we construct the tree recursively. $\square$

In Figure 7 we illustrate by constructing the inorder labeled, preorder read tree for the stack-sortable word (31254). The reader should note that the tree constructed in this manner does not reflect the stack operations necessary to sort the stack-sortable word in a straightforward way. Indeed, the tree structures look very different; compare the trees in Figures 6 and 7. Furthermore, if ρ is a stack-sortable word and T is the binary tree so that $\rho = [In : Pre]_T$, then the permutation $[Pre : In]_T$ will be the stack word ρ^{-1}. (See Corollary 1 in Section 5.)

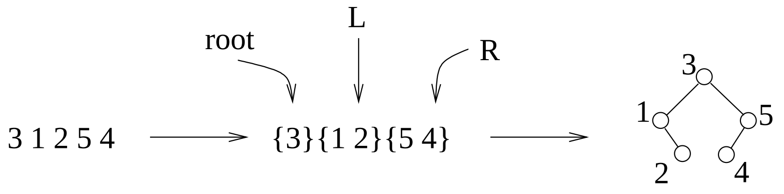


Figure 7: 231-avoiding permutation to InPre tree

The conversion of a stack word to a preorder labeled, inorder read tree will be used in following sections and so we will illustrate how this tree is constructed directly from the stack word. ([13] and [12] give a similar construction for InPre trees from stack-sortable words.) Note that since T will be preorder labeled, the root node must be labeled with 1. Those values in the stack word to the left of 1 will comprise the left subtree while those to the right of 1 will comprise the right subtree. We apply this technique recursively to each subtree, labeling the root with the smallest number in the subtree. We illustrate this conversion on the stack word (23154) in Figure 8.

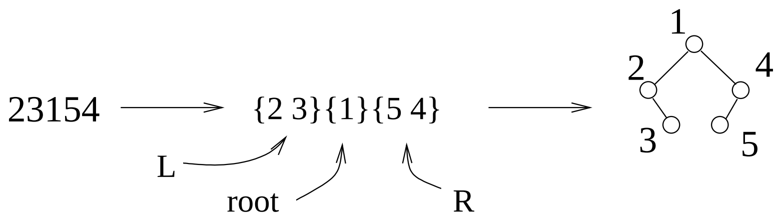


Figure 8: Stack word to PreIn tree

If π is a stack word, we call the binary tree T such that $[Pre : In]_T = \pi$ the *PreIn tree of* π.

4 Counting Permutations that are both Stack Words and Stack-sortable Words

Rotem [12] counted the number of permutations on $\{1, 2, \ldots, n\}$ that are both stack words and stack-sortable words, but his proof was difficult and required advanced graph theory techniques and terminology such as interval and permutation graphs. The characterization of stack words on $\{1, 2, \ldots, n\}$ as a permutation $[Pre : In]_T$ for some binary tree T with n nodes, as noted in the last section, allows us to perform this count easily. Since a stack word that is also stack-sortable is a stack word that is 231-avoiding, we need only characterize those trees T where $[Pre : In]_T$ have a 231 subsequence. We then will be able to count the remaining trees with n nodes.

Note that $[Pre : In]_T$ has a 231 subsequence if and only if there is a left elbow subtree (as shown in Figure 9) in T. The proof of this follows along lines similar to the proof of Proposition 1 and will not be included here. Thus to count the number stack words of length n that are 231-avoiding, we count binary trees with n nodes with no left elbows.

This can be done by first considering the binary tree of n nodes where every node (except the bottom one) has exactly one right child, as shown in Figure 9. We label these nodes $1, 2, \ldots, n$ from top to bottom. Now any tree without left elbows can be realized by choosing a subset of nodes from $\{2, 3, \ldots, n\}$. If node i is chosen, it becomes the left child of node $i-1$. For example, let $n = 7$ and suppose $\{3, 4, 7\}$ is the chosen set. The resulting tree (with labels) is illustrated in Figure 9.

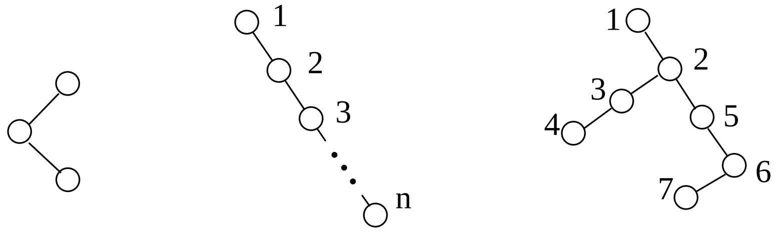


Figure 9: Left elbow, Tree with all right children, Tree with nodes $\{3, 4, 7\}$ chosen to be left children

Clearly, then, there is a one-to-one correspondence between subsets of $\{2, 3, \ldots, n\}$ and the binary trees with n nodes and no left elbows. Specifically, there are $\binom{n-1}{k}$ binary trees with no left elbows with exactly k left children. Thus, we the following.

Proposition 3 *There are*

$$\sum_{k=0}^{n-1} \binom{n-1}{k} = 2^{n-1}$$

permutations of $\{1, 2, \ldots, n\}$ *that are both 312-avoiding and 231-avoiding.*

5 Labeling and Reading Trees

In previous sections we have seen the correspondence between 312-avoiding permutations (stack words) and InPost and PreIn permutations. Likewise we've seen correspondence between 231-avoiding permutations and PostIn and InPre permutations. For permutations obtained from the same tree, there is a nice cancellation when multiplying permutations. For example, for a binary tree T, $[Pre : In]_T[In : Post]_T = [Pre : Post]_T$. (We multiply permutations right-to-left.) Indeed it is true that $[X : Y]_T[Y : Z]_T = [X : Z]_T$ for any choice of In, Pre, $Post$ as values for X, Y and Z.

Proposition 4 *For a given binary tree T, $[X : Y]_T[Y : Z]_T = [X : Z]_T$ for any choice of In, Pre, $Post$ as values for X, Y and Z.*

Proof: We will show that $[Pre : In]_T[In : Post]_T = [Pre : Post]_T$ for any binary tree T. Other choices for X, Y and Z follow in a similar manner. We denote T_{in}, T_{pre} and T_{post} to be the tree T with nodes labeled $1, 2, \ldots, n$ in an inorder, preorder and postorder traversal, respectively.

Note the following: (1) If $\pi = [In : Post]_T$, then $\pi(i)$ is the label of the node of T_{in} corresponding to the node of T_{post} with label i. (2) If $\sigma = [Pre : In]_T$, then $\sigma(j)$ is the label of the node of T_{pre} corresponding to the node of T_{in} with label j. (3) If $\rho = [Pre : Post]_T$, then $\rho(k)$ is the label of the node of T_{pre} corresponding to the node of T_{post} with label k.

So, if $\sigma = [Pre : In]_T$, $\pi = [In : Post]_T$ and $\rho = [Pre : Post]_T$, then $\sigma(\pi(i))$ is the label of the node of T_{pre} corresponding to the node on T_{in} with label $\pi(i)$. But $\pi(i)$ is the label of the node of T_{in} corresponding to the node of T_{post} with label i. Hence $\sigma(\pi(i))$ is the label of T_{pre} corresponding to the node of T_{post} labeled i. But this is precisely $\rho(i)$. $\Box$

Corollary 1 *If $\pi = [X : Y]_T$ for some binary tree T, then $\pi^{-1} = [Y : X]_T$, for any choice of In, Pre, $Post$ as values for X and Y.*

6 Preorder labeled, Postorder read binary trees

For a given binary tree T there are nine different permutations $[X : Y]_T$ that can be formed from various choices of X and Y from among In, Pre, and $Post$. The trivial cases of $X = Y$ (i.e., $[Pre : Pre]$) obviously yield the identity permutation. We have explored the cases of $[Pre : In]_T$, $[In : Pre]_T$, $[Post : In]_T$, $[In : Post]_T$ and their relation to stack words and stack-sortable words. The two remaining permutations, $[Pre : Post]_T$ and $[Post : Pre]_T$, possess some properties not seen in the other operators.

Proposition 5 *For a binary tree T, the permutation $[Pre : Post]_T$ produces a stack word (that is, a 312-avoiding permutation).*

The proof of this is similar to our previous proof that $[In : Post]_T$ is a stack word and will not be given here. Note, however, that this proposition does not propose a bijection between stack words and $[Pre : Post]_T$ permutations, since clearly any permutation (on n items) obtained in this manner must have $\pi(n) = 1$. Thus the stack words which can be generated by $[Pre : Post]_T$ are a subset of all stack words which we will refer to as *PrePost words* or *PrePost permutations.* Figure 10 shows an example of two different preorder labeled, postorder read trees that yield the permutation (3241).

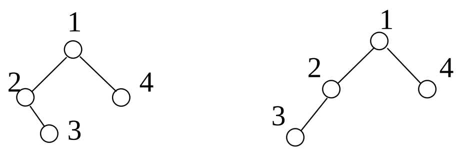


Figure 10: Binary trees with PrePost permutation (3241)

The key observation concerns nodes with one child. Notice that switching the left and right position of a parent's only child has no effect on the outcome of a $[Pre : Post]_T$ permutation. Indeed, this is the only way two different binary trees can produce the same PrePost word. In fact, this characterizes full binary trees (that is, all nodes have either 0 or 2 children) if and only if the permutation $[Pre : Post]_T$ can be obtained in this manner uniquely from T:

Proposition 6 *T is a full binary tree if and only if there is no other binary tree T' such that $[Pre : Post]_{T'} = [Pre : Post]_T$.*

Proof: Suppose T is a full binary tree. Without loss of generality, assume T has $n > 1$ nodes and $\pi = [Pre : Post]_T$. Then the root node has two children. Let $\pi(n)$ be the label of the root, $\pi(i)$ be the label of a node in the left subtree of T, and $\pi(j)$ be a label of a node in the right subtree of T. Then $\pi(n) = 1$, $i < j$ (since π is postorder read), and $\pi(i) < \pi(j)$ (since π is preorder labeled). If T' were another preorder labeled, postorder read binary tree with $\pi = [Pre : Post]_{T'}$, then $\pi(i)$ would necessarily be a label of a node in the left subtree of T' and $\pi(j)$ would be a label of a node in the right subtree of T'. Thus the left subtrees of T and T' have exactly the same number of nodes with the same set of labels. Likewise for the right subtrees of T and T'. Applying this reasoning inductively to those left and right subtrees, we see that T and T' are isomorphic as binary trees (when left and right children are differentiated).

Conversely, if T is not full, then there is at least one node with exactly one child. Pick one such node. Moving the child of that node from right to left or vice versa will yield the same PrePost permutation. $\square$

We now show we can distinguish PrePost permutations from other stack words. PrePost words satisfy the following condition, which we call *Condition P*.

A permutation π of $1, 2, \ldots, n$ satisfies *Condition P* if

1. π is 312-avoiding,

2. $\pi(n) = 1$, and
3. if $\pi(i) = \pi(j) + 1$, where $j < i$, then $\pi(i+1)$ is the largest integer less than $\pi(i)$ not included in $\{\pi(1), \pi(2), \ldots, \pi(i)\}$.

Now suppose that $\pi = (\pi(1)\,\pi(2)\,\cdots\,\pi(n))$ and $\pi(i) = 2$. Let $L = (\pi(1)\,\cdots\,\pi(i))$ and $R = (\pi(i+1)\,\cdots\,\pi(n-1))$. (Note that R might be empty.) Then we can write $\pi = LR1$, thinking of π as a sequence of integers. For example, if $\pi = (4\,5\,3\,6\,2\,8\,9\,7\,1)$, then $L = (4\,5\,3\,6\,2)$ and $R = (8\,9\,7)$.

Before showing that Condition P characterizes a PrePost permutation, we offer some useful lemmas:

Lemma 1 *If π satisfies Condition P, and we write $\pi = LR1$, then all numbers in L are less than all numbers in R.*

Proof: If not, then π is not 312-avoiding. $\square$

Lemma 2 *If π satisfies Condition P, then $\pi(n-1) = i+1$, where $\pi(i) = 2$. That is, $\pi(n-1)$ is the smallest number in R.*

Proof: L contains the integers $2, 3, \ldots, i$ and R contains the integers $(i+1), \ldots, n$. Let $\pi(k) = i+1$. Then $\pi(j) = i$ for $j < i < k$. Condition P requires that $\pi(k+1) = 1$. $\square$

Lemma 3 *Suppose π satisfies Condition P, and we write $\pi = LR1$. The previous two lemmas recursively hold for blocks L and R. That is L (and R) can be partitioned as ABm, where m is the smallest number in L (or R), the last element of A is $m+1$, all elements of A are less than all elements of B (which might be empty), and if $B \neq \emptyset$ then the smallest element of B is the last one. This can be applied recursively to A and B.*

The proof of this lemma follows the proofs above.

We are now ready to prove that Condition P characterizes a PrePost permutation:

Proposition 7 *A permutation satisfies Condition P if and only if it is a PrePost permutation.*

Proof: Suppose π satisfies condition P. We construct a binary tree T where $[Pre : Post]_T = \pi$. Since $\pi = LR1$, label the root node of the binary tree with 1. The left subtree will be the $[Pre : Post]$ tree for L and the right subtree will be the $[Pre : Post]$ tree for R. Repeat recursively on the partitioning of L and R as noted in Lemma 3. We illustrate on the permutation (4 3 5 2 8 7 6 1) in Figure 11. Here, $L = (4\,3\,5\,2)$ and $R = (8\,7\,6)$.

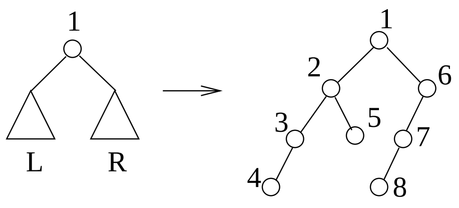


Figure 11: PrePost tree construction

Now suppose that $\pi = [Pre : Post]_T$ for some binary tree T. Clearly $\pi(n) = 1$ and we have shown that π is 312-avoiding in Proposition 5. Suppose $\pi(i) = \pi(j) + 1$ where $j < i$. Since $j < i$, then in a postorder traversal of a preorder labeled tree T either (1) $\pi(j)$ is a descendant of $\pi(i)$, or (2) $\pi(j)$ and $\pi(i)$ have a common ancestor $\pi(k)$ with $\pi(j)$ being a left descendant of $\pi(k)$ and $\pi(i)$ being a right descendant of $\pi(k)$. But $\pi(i) > \pi(j)$ which eliminates possibility (1). So, let's assume (2).

Now since $\pi(i) = \pi(j) + 1$, this means node $\pi(i)$ is labeled immediately after node $\pi(j)$ in a preorder labeling of T. Thus $\pi(i)$ must be the right *child* of node $\pi(k)$ (and so $\pi(k) < \pi(i)$). Furthermore, all the preorder labels in the left subtree of node $\pi(k)$ are the set $\{\pi(k) + 1, \ldots, \pi(i) - 1\}$. That is, $\pi(j) = \pi(i) - 1$. And so node $\pi(k)$ is the next node visited after node $\pi(i)$ in the postorder traversal of T and is indeed the largest integer less than $\pi(i)$ not yet used. $\square$

It is straight forward to construct a recurrence relation which counts the number of these special PrePost words:

Proposition 8 *For trees with n nodes, the number of n-length PrePost words is given by:*

$$P_n = \sum_{i=1}^{n-1} P_i P_{(n-1)-i}$$

Notice that this recurrence is nearly identical to that of the Catalan Sequence, differing only in the lower limit of the summation; i sums from 0 to $n - 1$ for Catalans.

Proof: The constructive proof of this uses induction. Clearly, $P_0 = P_1 = 1$. Assume that the equation for P_n holds for values of $n \leq m$. We show how to construct trees with $m+1$ nodes that generate different stack words.

After accounting for the root node, there are m more nodes for the remaining tree. It is possible that all m of these are in the left subtree of the root while 0 are in the right subtree. Or it is possible that $m-1$ are in the left subtree while 1 node is in the right subtree. And so on. However, any tree with 0 nodes in the left subtree and m nodes in the right subtree will produce the same stack word as m nodes in the left subtree and 0 nodes in the right subtree.

There is only one way to generate a 0-length subtree (namely the empty tree), thus $P_0 = 1$. Thus we arrive at the desired summation. $\square$

The first few numbers in this sequence are: $1, 1, 1, 2, 4, 9, 21, 51, 127, 303$.

7 An Equivalence Relation on Binary Trees and an Equivalence on Stack Words

In previous sections we give two different binary trees associated with a stack word π; namely, T_1, the binary tree for which $\pi = [Pre : In]_{T_1}$, and the binary tree T_2 for which $\pi = [In : Post]_{T_2}$. In this section will describe a transformation from T_1 to T_2. Of course the binary tree T_2 is also a PreIn tree for a stack word for which we can find an InPost tree via this transformation. This will give rise to an equivalence relation on the set of binary trees with n nodes.

7.1 Transformations Between Binary Trees

Given a stack word π, we can define a transformation between T_1 and T_2 as follows. Given a preorder labeling of T_1, $t(T_1)$ produces a tree T_2 with an inorder labeling under the following conditions:

1. If $i \in T_1$ is in the left subtree of $j \in T_1$, then $i \in T_2$ is in the right subtree of $j \in T_2$.
2. If $i \in T_1$ is the right child of $j \in T_1$, then $j \in T_2$ will be the left child of $i \in T_2$.

Proposition 9 *Given a stack word permutation π and binary trees T_1 and T_2 such that $\pi = [Pre : In]_{T_1} = [In : Post]_{T_2}$, then $t(T_1) = T_2$.*

Proof: Let $\pi = [Pre : In]_{T_1} = [In : Post]_{T_2}$. In T_1, if $k < j$ and $\pi(k) > \pi(j)$ then k appears in j's left subtree. However, in T_2, if $k < j$ then k appears in j's right subtree, which satisfies condition (1) of the transformation t.

Now, if $\pi(j)$ has a right child it must be labeled $\pi(k) > \pi(j)$ and is in position $k > j$. Also, every $\pi(l)$ where $j < l < k$ has $\pi(l) > \pi(k)$ since each $\pi(l)$ would be in $\pi(k)$'s left subtree in T_1. In tree T_2, since $\pi(j) < \pi(k)$ and we've labeled T_2 inorder, $\pi(j)$ is either in $\pi(k)$'s left subtree or $\pi(k)$ is in $\pi(j)$'s right subtree. Since T_2 is traversed postorder, $\pi(k)$ would be traversed before $\pi(j)$ if $\pi(k)$ is a descendant of $\pi(j)$. Thus $\pi(j)$ is in the left subtree of $\pi(k)$ in T_2. Further, since $j < l < k$ and $\pi(l) > \pi(k)$, $\pi(l)$ is in the right subtree of $\pi(k)$ in T_2. Thus $\pi(j)$ is the left child of $\pi(k)$ in T_2. Thus $t(T_1) = T_2$. $\square$

There is a corresponding characterization of the inverse transformation for t from T_2 to T_1 where $[In : Post]_{T_2} = [Pre : In]_{T_1}$, but we leave the details to the reader. Likewise, there are corresponding transformations between trees and related stack-sortable words. Again, we do not include the details here.

Given a binary tree T, t gives a sequence of trees: $T = T_1, t(T_1) = T_2, t(T_2) = T_3, \ldots$. Eventually, $t(T_m) = T_1$ for some m. It follows that t partitions the binary trees with n vertices and so gives rise to an equivalence relation on the binary trees with n vertices. We show two such equivalence classes in Figure 12. Note that the transformation t moves counter-clockwise around these diagrams.

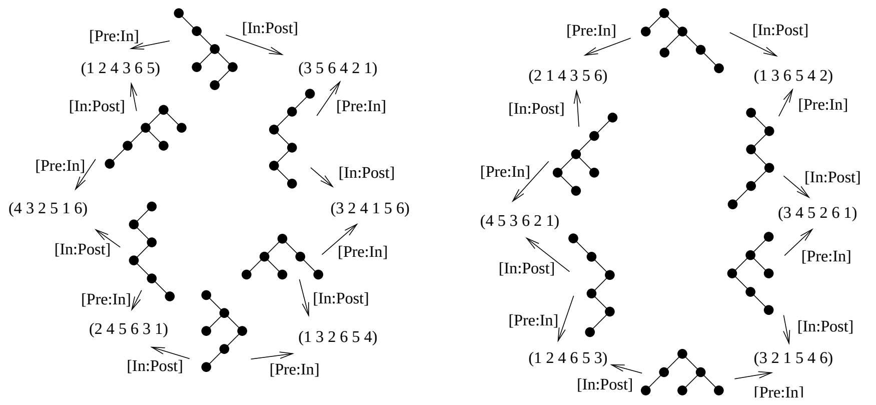


Figure 12: Two equivalence classes for $n = 6$

7.2 Notes on Distinguishing Equivalence Classes

It is not clear how to completely characterize these equivalence classes of trees. But, certain properties are preserved within classes which offer some insight. It is evident from the description of the transformation at the beginning of this section that for a binary tree T with l nodes that are left children and r nodes that are right children, then $t(T)$ has r nodes that are right children and l nodes that are right children. (The reader should examine Figure 12 for examples of this.) Thus we have:

Proposition 10 *If T is a binary tree with a different number of nodes that are left children than nodes that are right children, then the number of trees in the equivalence class of T is even.*

The transformation t causes a sort of rotation and sliding motion as tree T changes to tree $t(T)$. The tree T is rotated counter-clockwise, causing some edges to ascend instead of descend, making the resultant figure no longer a tree. These offending portions are then slid down to the left. This is illustrated in Figure 13 below. Donaghey [5] noticed a similar transformation between Catalan trees.

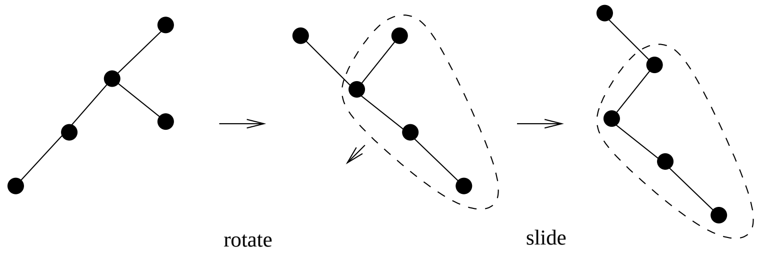


Figure 13: Transformation rotation

Consider this rotational view of the transformation t applied to the so-called left elbow, right elbow, and fork subtrees of T (as shown in Figure 14). Under t, left elbows become right elbows, right elbows become forks and forks become left elbows. Again, the reader should examine the equivalence classes in Figure 12 for examples.

Thus, each binary tree T has an 'elbow-triple' (l, r, f) giving the number of left elbows (l), right elbows (r), and forks (f) of the tree. It is clear, then, that the tree $t(T)$ has an elbow-triple of (f, l, r). Thus, we have:

Proposition 11 *If T is a binary tree with elbow-triple (l, r, f) where l, r,*

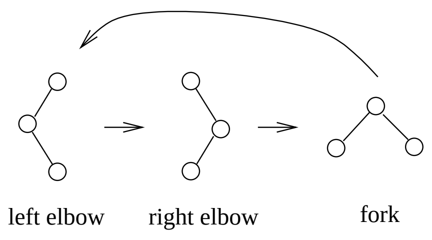


Figure 14: left elbow, right elbow, fork

and f are not all equal, the number of trees in the equivalence class of T is a multiple of 3.

If the elbow-triple for a tree is (n, n, n) and there are the same number of nodes that are left children as those that are right children, then the size of the equivalence class of the tree may not be a multiple of 3 nor a multiple of 2, as Figure 15 show.

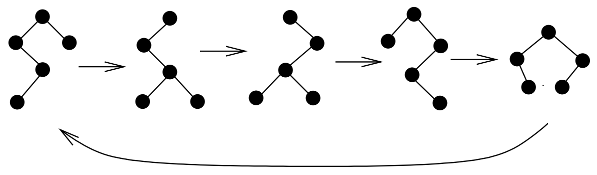


Figure 15: An equivalence class with five trees.

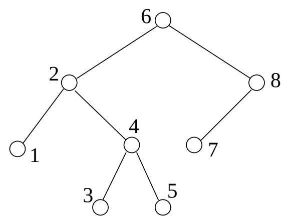
Thus, we can say something about when two binary trees (with the same number of nodes) are not in the same equivalence class. Let (L, R) be the number of left children and right children in tree T. Call this the *left-right pair* of T. Then $t(T)$ has left-right pair (R, L). Thus, two trees are not equivalent if there left-right pairs are not either identical or reversed. Similarly, two trees are not equivalent if their elbow-triples are not rotations of each other.

This is not a characterization of equivalence classes. It is easy to find two trees with the same elbow-triple (or rotation) and same left-right pair (or reverse) that are not equivalent. (We leave this as an exercise.) The problem of characterization seems to be most difficult in situations when the elbows and forks overlap, as in Figure 15.

7.3 Transformation Between Stack Words

We've just seen a transformation between two trees giving the same permutation. Likewise, we'd like to find a transformation between two permutations that are produced by the same tree, without referring to the tree. Two permutations yielded by the inorder labeled binary tree T are $\pi_1 = [In : Pre]_T$ (a stack-sortable word or 231-avoiding permutation) and $\pi_2 = [In : Post]_T$ (a stack word or 312-avoiding permutation). Two stack words created by the tree T are $[In : Post]_T$ and $[Pre : In]_T$, the latter being the inverse of $[In : Pre]_T$, by Corollary 1.

We wish to show a transformation α between permutations $[In : Post]_T$ and $[Pre : In]_T$ without resorting to using binary tree T. We do this by focusing on the relationship between the postorder traversal and the preorder traversal of the inorder labeled tree T. That is, we focus on $[In : Post]_T = \pi_1$ and $[In : Pre]_T = \pi_2$. In Figure 16 is an example of an inorder labeled tree and these permutations.



$$[\mathrm{In:Pre}]_T = (6\ 2\ 1\ 4\ 3\ 5\ 8\ 7) = \pi_1$$

$$[\mathrm{In:Post}]_T = (3\ 5\ 4\ 1\ 2\ 7\ 8\ 6) = \pi_2$$

Figure 16: Inorder labeled tree and permutations

A preorder traversal of T recursively reads the nodes of T as *middle, left, right* and a postorder traversal reading recursively reads the nodes of T as *left, right, middle.* Reversing the postorder reading process (i.e. reading π_2 right to left) yields a recursive reading of the nodes of T as *middle, right, left.* Note that the only difference in a preorder traversal of T and a "backwards" postorder traversal of T is that the "backwards" postorder reads right before left. However, if T is labeled inorder, nodes to the right of any subtree root of T have a larger label than those at the subtree root or to the left of that root. Hence, decreasing sequences of numbers in the "backwards" π_2 correspond to left branches of the tree and increasing sequences correspond to right branches. So to create π_1 from π_2 we perform the following steps. First, read π_2 right to left and find a *greedy* decreasing sequence beginning at $\pi_2(n)$and ending at $1 = \pi_2(i)$. By greedy we mean take the first available decrease as the permutation is read right to left, skipping any values not decreasing. In our example, the first right-to-left greedy decreasing sequence of (3 5 4 1 2 7 8 6) is (6 2 1).

Push onto a stack the skipped subsequences of contiguous skipped number in the permutation (including the unused leftmost subsequence to the left of $\pi_2(i) = 1$) as they are encountered. Now repeat this process on the skipped subsequences visiting these subsequences in a last-in, first-out manner. This exactly mimics how a preorder traversal reads the inorder labeled tree. The skipped subsequences represent nodes corresponding to right branches in the tree because they have larger labels than the node previously read.

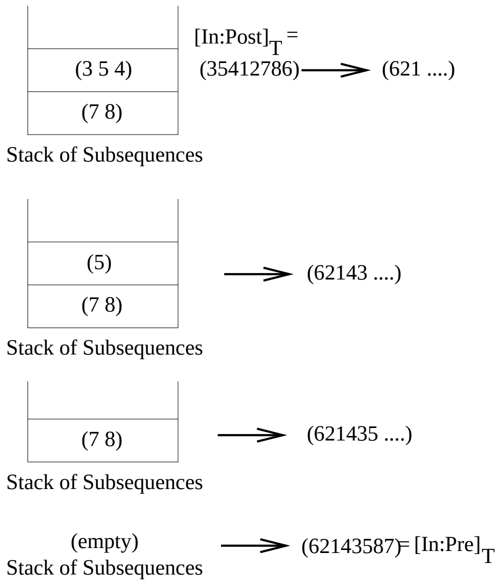


Figure 17: Creating an InPre permutation from an InPost permutation

A preorder traversal would revisit these skipped right branches starting with the most recently skipped. Now to create $[Pre : In]_T$, just take the inverse of the permutation created by the above process. Namely create a permutation that sends each $\pi_2(i)$ to the order number in which $\pi_2(i)$ was read in the above process. Thus, in our example, since $\pi_2 = [In : Pre]_T = (6\,2\,1\,4\,3\,5\,8\,7)$, we have $[Pre : In]_T = (3\,2\,5\,4\,6\,1\,7\,8)$. Hence, we have shown the following transformation α takes $[In : Post]_T$ to $[Pre : In]_T$.

Proposition 12 *Given an inorder labeled binary tree T, the following transformation α takes $\pi = [In : Post]_T$ to $[Pre : In]_T$:*

```
1. Push the current sequence $\pi(n), \ldots, \pi(1)$ on stack $P$.
2. Set $order$ to 1.
3. While $P$ is not empty do
4. Pop the top subsequence off $P$ and assign to $S$.
5. Find a greedy decreasing sequence $\bar{S}$ in $S$ and for each $\pi(i)$
   in $\bar{S}$, set $\alpha(\pi(i))$ to $order$ and increment $order$.
6. For each contiguous subsequence $S_j$ of $S$ that is skipped over
   in forming $\bar{S}$, push $S_j$ onto $P$.
7. end while
```

8 Concluding Comments and Open Questions

We have described a natural equivalence among binary trees with n nodes and have given some nice properties preserved in a given class. But the natural open problem is to characterize equivalence classes of trees, as described in Section 7. Likewise, characterize equivalence classes of stack words (or stack-sortable words) as described in Section 7.

We have concentrated on stack words and stack-sortable words and their relation to traversals of binary trees. There might be restrictions on the structure of these trees that give rise to interesting restrictions on stack words and stack-sortable words and vice versa. This could be a fruitful area for future work.

References

[1] M. D. Atkinson. “Restricted Permutations,” *Discrete Mathematics*, 195 (1999), 27–38.

[2] Miklos Bona. “Permutations Avoiding Certain Patterns: The Case of Length 4 and Some Generalizations,” *Discrete Mathematics*, 175 (1997), 55–67.

[3] Mikos Bona. “Exact Enumeration of 1342-Avoiding Permutations: A Close Link with Labeled Trees and Planar Maps,” *Journal of Combinatorial Theory*, Series A, 80 (1997), 257–272.

[4] Mireille Bousquet-Mèlou. “Sorted and/or Sortable Permutations,” *Discrete Mathematics*, 223 (2000), 23–30.

[5] Robert Donaghey. “Automorphisms on Catalan Trees and Bracketings,” *Journal of Comb. Theory*, Series B, 29 (1980), 75-90.

[6] Shimon Even. *Graph Algorithms*. Computer Science Press, 1979.

[7] Ralph Grimaldi. *Discrete and Combinatorial Mathematics*, 3rd edition. Addison-Wesley, 1994.

[8] O. Guibert. “Stack Words, Standard Young Tableaux, Permutations with Forbidden Subsequences and Planar Maps,” *Discrete Mathematics*, 210 (2000), 71–85.

[9] Donald E. Knuth. *The Art of Computer Programming: Volume 1*. Addison-Wesley, 1997.

[10] D. Marinov and R. Radoicic. “Counting 1324-Avoiding Permutations,” *The Electronic Journal of Combinatorics* 9(2) (2003) #R13.

[11] N. J. A. Sloane. *The Online Encyclopedia of Integer Sequences*. http://www.research.att.com/ njas/sequences/. Sequence A000992.

[12] D. Rotem. “Stack Sortable Permutations,” *Discrete Mathematics*, 33 (1981), 185–196.

[13] D. Rotem and Y. Varol. “Generating Binary Trees from Ballot Sequences,” *JACM* 25 (1978), 396–404.

[14] Zvedina E. Stankova. “Forbidden Subsequences,” *Discrete Mathematics* 132 (1994), 291–316.

[15] Richard Stanley. *Enumerative Combinatorics*, Vol 2. Cambridge University Press, 1999.

[16] Julian West. “Sorting Twice Through a Stack,” *Theoretical Computer Science*, 17 (1993), 303–313.

[17] Julian West. “Generating the Catalan and Schröder numbers,” *Discrete Mathematics*, 146 (1995), 247–262.

[18] Julian West. “Generating Trees and Forbidden Sequences,” *Proceedings of the 6th Conference on Formal Power Series and Algebraic Combinatorics*, DIMACS France (1996), 363–374.