A Correspondence between Stack Permutations and Binary Trees via Hille Encoding

Kelan Liu, Lijuan Du*
College of Applied Science and Technology, Beijing Union University, Beijing, China
**Corresponding Author.*

Abstract: This paper systematically investigates the bijective or isomorphic construction problem between stack permutations and binary tree, with a focus on the equivalence and differences between Hille codes and stack codes. Through constructive proofs, we correct algorithmic errors in the original literature and rigorously demonstrate the equivalence between stack codes and Hille codes in the bijective sense. Furthermore, we analyze the intersection size of Hille codes and stack codes, revealing that when the number of binary tree nodes $n > 8$, the overlapping portion constitutes less than half of the total cases. Additionally, we provide precise upper and lower bounds for the effective length of Hille codes, proving that their maximum value is $3n - 4$ and showing that the binary tree structure achieving this bound is unique. These results offer new tools and perspectives for the study of stacks and binary trees, while also laying a foundation for subsequent algorithm design and combinatorial optimization.

Keywords: Encoding; Binary Trees; Stack Permutations; Correspondence

1. Introduction

The isomorphic relationship between binary trees and stack permutations is a classical problem in combinatorics and computer science, centered on constructing a bijection between the two structures. Knott defined a rank for each binary tree [1] and provided an efficient algorithm for computing the rank sum and its inverse. Inspired by this, Hille proposed Hille encoding [2], aiming to establish a correspondence between binary trees and stack permutations using binary sequences. However, the original encoding algorithm in Hille's work contained flaws, resulting in incorrect computation of Hille codes for certain binary trees. This observation prompted us to re-examine the specifics of bijective construction.

2. Constructing the Bijection

A stack permutation is also called a stack shuffle. Given a non-empty stack A and empty stacks B and S, only the following operations are permitted at each step: (i) pop A and push S. (ii) pop S and push B. It's easy to see that eventually all elements of A must enter B. Such B is called a stack permutation of A.

All stack permutations of are exactly all possible pop sequences of A. It's well known that an n-element stack has $\frac{1}{n+1}\binom{2n}{n}$ possible pop scenarios. On the other hand, n-nodes can form $\frac{1}{n+1}\binom{2n}{n}$ binary trees, both being Catalan numbers. This shows that the two sets are isomorphic as collections, and the natural and important question is: how to implement this isomorphism? That is, to explicitly write out this bijection. The work in this direction first came from Hille [2].

Since our discussion doesn't involve specific elements, without loss of generality, we can fix the push order of stack permutations as $123\cdots n$. These numbers are also the labels of binary tree nodes. In keeping with the tradition of computer science, the binary trees discussed in this article distinguish between nodes that go to the left and nodes that go to the right, except for the root node.

On the other hand, what affects the pop sequence are only the push and pop operations, while constructing binary trees appears to allow one more type of operation at first glance. We encode stack pushes and pops as 1 and 0 respectively, and call the corresponding binary sequence a stack code. Correspondingly, the construction of binary trees is characterized by Hille encoding [2].

2.1 Hille Encoding

The following three rules describe the

conversion process from Hille encoding to binary trees. It can be easily verified that given any binary tree, its Hille encoding can be obtained through these three rules. 1 represents adding a left subtree to the current tree node. For example, 111 represents the tree (·(·(·))). 01 repre-sents adding a right subtree to the current tree node. For example, 10101 represents the tree (·_(·_(·))). As mentioned earlier, the binary trees we are discussing must distinguish between left and right nodes. Therefore, when using this method of expression, it is inevitable that symbols _ for empty nodes will be introduced for placeholders. For the 0s before 01, these 0s are used to indicate backtracking the current tree node to its k-th ancestor. For example, 11001 represents the tree (·(·)(·)). Note that each tree node has a unique parent node, therefore the backtracking operation is well-defined, where a backtrack means changing the current node to its parent.

Accordingly, we can define parsing rules for Hille encoding based on this, which are used to convert such a valid binary sequence into a series of operations for constructing a binary tree. We use a BNF-like grammar to define this parser.

<lchild> : = "1"
<rchild> := "01"
<up> := "0"
<parser> := (<lchild>|<rchild>|<up>)*

2.2 $n = 3$

To make it easier for readers to understand the issues discussed in this article and to prepare for constructing counterexamples later, we present a simple enough example to illustrate that Hille encoding contains information about both constructing a binary tree and the stack replacement sequence.

Figure 1 are all 5 possible cases of three-node binary trees to demonstrate how their binary sequences correspond to stack permutations. The binary sequences are padded with trailing zeros based on the stack being emptied.

The reverse of this process is non-trivial. If we only consider the case of $n = 3$, as shown in the figure above, we can observe that these stack permutations precisely correspond to the in-order traversal sequences of binary trees when nodes are numbered according to their insertion order. That is, nodes are added step-by-step following the Hille encoding. For the binary tree 1101000100, numbering the nodes in insertion order and performing an in-order traversal yields 2314.

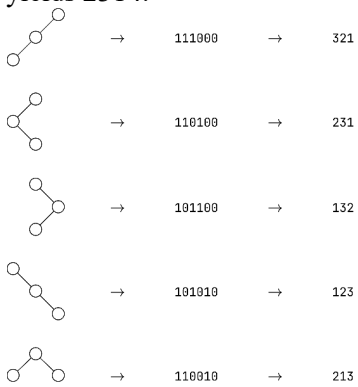

Figure 1. All Three Node Trees

Next, we will explain the non-trivial aspects of this phenomenon and the specific conditions under which this coincidence occurs, as detailed in Proposition 3.1 and its construction process. Subsequently, we will also make precise assertions regarding the intersection size of the two types of encodings generated by all binary trees with a fixed number of nodes, as well as the range of lengths for such encodings. This involves a detailed examination of how different binary trees, when subjected to our revised encoding methods, produce sequences that may overlap in certain aspects. By analyzing the intersection size, we can gain insights into the commonalities and differences between these encodings, which is crucial for understanding the structural properties of binary trees and their representations. Furthermore, determining the length range for these encodings provides practical guidelines for their application in various computational scenarios, ensuring that the encodings are both efficient and effective. This dual analysis not only enhances our theoretical understanding but also offers valuable information for practical implement-ations in fields such as data structures, algorithms, and computational theory.

3. Correction to the Original Text

First, an observation regarding the original literature: The algorithm proposed in Hille's paper [2] actually contains an error. We have rewritten it in Lean4 language, namely,

```
def encode: Tree -> String
| . node l r = > "1" ++ encode l ++ encode r
| _ = > "0"
```

Now, denote this encode function as m. A simple example reveals that converting a binary tree B into its Hille encoding $h(B)$ is not

equivalent to a straightforward in-order traversal $m(B)$ as shown in Figure 2.

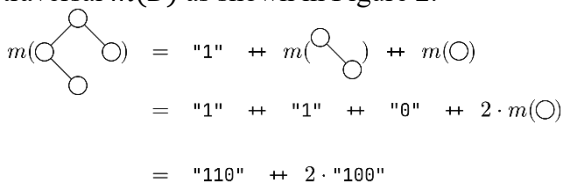

Figure 2. Counterexample Calculation

It can be verified that starting from the encoding 110100100, the original binary tree structure cannot be directly recovered. The correct Hille encoding should be 1101000100. However, if we interpret the sequence 110100100 as stack push/pop operations (i.e., as a stack encoding), it correctly yields the stack permutation 2314. This implies that for binary trees with node count $n > 3$, there exist cases where a tree's stack encoding differs from its Hille encoding.

Recalling the bijective relationship between binary trees and stack permutations, this suggests there must exist a constructive procedure allowing conversion between the two. Below, we will explicitly present this result (see Proposition 3.1). Subsequently, through Proposition 3.2, we explain why in-order traversal frequently yields correct Hille encodings for small values of n.

Proposition 3.1. Stack Encoding and Hille Encoding are equivalent.

proof. First, it is straightforward to verify that in-order traversal always provides a mapping from a binary tree $b \in B$ to a stack permutation $s \in S$. Conversely, for every stack permutation $s \in S$, a unique binary sequence i.e. the stack encoding $c \in C$ —can be derived. Ignoring trailing 0s, when the stack encoding c matches the Hille encoding $h \in H$ of the binary tree b, applying in-order traversal directly to B will yield the correct Hille encoding. We externally illustrate the equivalence of the two encodings and the specific methods of mutual conversion by proving the commutativity of the following diagram as shown in Figure 3.

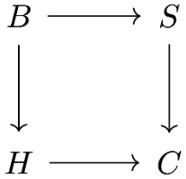

Figure 3. Commutative Diagram

Let $B \to H$ be denoted as f. According to Section 2.1, f is a bijection. Based on the properties of binary trees, $g: B \to S$ represents in-order traversal, where the push order of stack permutations is fixed as $123\cdots n$, thereby determining the level-order traversal of B. Thus, g is also a bijection. Now consider $h: S \to C$, which is clearly a bijection as well. Finally, we can recover the information of $c \in C$ from $h \in H$ by treating the h sequence as stack encoding and ignoring pops from an empty stack, which implies that $H \to C$ is a surjection. Then, using $C \to S \to B \to H$, we obtain $H \cong C$.

The encoding algorithm used in Hille's original work is precisely $h \circ g: B \to C$, while the intended correct implementation should be f, hence the two differ by an isomorphism in their results.

Proposition 3.2. Let the set of all Hille encodings for n-node binary trees be denoted as H_n, and the set of all stack encodings for n-node binary trees as C_n. For the size $a_n := |H_n \cap C_n|$, we have the following characterization (1):

$$a_n = \sum_{k=0}^{n-1} \sum_{j=0}^{n-k} \frac{1}{j+1} \binom{n-k-j}{j} \binom{k}{j} \binom{k+j+2}{j} \quad (1)$$

Let $N = n + 1$. The proof of this equality can be obtained by analyzing a Dyck path of semi-length that does not contain the subsequence UUDD. This also corresponds to the number of lattice paths from $(0,0)$ to (N, N) that do not cross the diagonal and allow step sizes $(1, k), (k, 1), k \geqslant 1$. More simply, it is the number of skew Motzkin paths of length [3]. This also means that we can use a_n inversely to provide a new representation for these special combinatorial sequences.

proof. We only provide an outline of the proof, transforming it into a problem that has already been verified in combinatorics [3]. First, it needs to be proven that a_n satisfies the recursive relation $a_{n+1} = \sum_{k=2}^{n} a_k a_{n-k}$, which can be achieved through induction. Then, based on a few initial values $(a_i)_{0 \leqslant i \leqslant 2}$ and the recursive relation for Proposition 3.2, the expression is inductioned again. The specific calculation process is not suitable to be unfolded here.

Corollary 3.3. Let the Catalan number be c_n, which is also the size of H_n or C_n. Naturally, we may ask about the relation-ship between the proportion $|H_n \cap C_n|$ among all n-node binary trees and n. Based on Proposition 3.2 and the asymptotic estimate of Catalan numbers $c_n \sim \frac{4^n}{\sqrt{\pi} n^{3/2}}$ [4], we know that

$\lim_{n\to\infty}\frac{a_n}{c_n}=0$. In fact, when the number of nodes in binary trees exceeds 8, the overlapping portion between H_n and C_n will be less than half of the total.

4. Bounds on Effective Length

An interesting question to explore is, given an n-node binary tree, to determine the range of effective lengths $\ell(H_n)$ for its Hille encoding H_n. Here "effective" naturally refers to removing trailing consecutive zeros 0. We will provide a precise answer to this question.

Proposition 4.1. (Bounds on Effective Length) For any n-node binary tree, the effective length of its Hille encoding satisfies the inequality (2):

$$n\leqslant\ell_n\leqslant max(0,2n-1,3n-4) \quad (2)$$

The verification of the lower bound n is straightforward, constructing an n-node binary tree requires at least 1s. For our discussion here, we only need to verify that when $n\geqslant 3$, the maximum effective length of Hille encoding is $3n-4$.

proof. Note that to maximize the effective length, the corresponding binary tree should have as many right nodes and backtracking operations as possible. Satisfying both requirements implies: (i) there must be at least one left node besides the root. Otherwise, the binary tree would only take forms like $1010101\cdots$, with length $2n-1$; (ii) the tree must have at least one node to the right of the root. Otherwise the length contribution from backtracking wouldn't reach maximum. As illustrated below Figure 4:

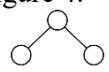

Figure 4. 11001

From this starting point, we add all remaining $n-3$ nodes to the right of the tree's only left node. That is Figure 5:

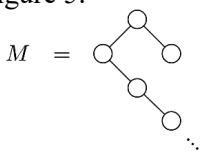

Figure 5. 110101…00001

Now we just need to calculate $\ell(M)$ to obtain the maximum value of ℓ_n. Let's directly write out $h(M)$ as shown in Figure 6:

$$\begin{aligned} h(M) &= \texttt{"110101...00001"} \\ &= \texttt{"11"} \mathbin{+\!\!+} (n-3)\cdot\texttt{"01"} \\ &\quad \mathbin{+\!\!+} (n-2)\cdot\texttt{"0"} \mathbin{+\!\!+} \texttt{"01"} \end{aligned}$$

Figure 6. $h(M)$ Calculation

We can immediately see that $\ell(M)=2+2(n-3)+(n-2)+2=3n-4$.

Conversely, starting from tree M, we can verify that any operation maintaining the same number of nodes won't increase its Hille encoding's effective length. Furthermore, we can assert that any such operation will strictly decrease the effective length. To illustrate this assertion, we only need to analyze the construction M of maximum encoding length. The first three nodes come from Figure 2, which represents the maximum encoding length construction for a 3-node binary tree. The remaining $n-3$ nodes have only two adjustment methods: either changing to a leftward node or placing it to the right of the root node. The former reduces the encoding length by 1, and the latter reduces the encoding length by 2 or 1, corresponding to moving left or right, respectively. In other words, the binary tree structure B that achieves $\ell(B)=3n-4$ is unique, namely M.

It is evident that from this proof process, we can discover that by assigning appropriate codes to binary trees, the verification of related properties can be transformed into pure equality and inequality issues. This implies that we can apply these techniques to a wider range of similar problems. It demonstrates that the methods proposed in this paper have significant versatility and potential for application. Please note that the evaluation here does not merely imply the shortest coding in the sense of information theory, but also considers requirements such as maintaining structural correspondence between two structures after data updates. For the binary tree coding problem in pure information theory, Cover & Thomas [5] have provided answers. Munro & Raman [6] proposed a near-optimal coding for binary trees, which uses only $2n+o(n)$ bits, close to the lower bound $2n-O(logn)$ given by information theory. As potential future research, exploring how to use Hille encoding or stack permutation-binary tree isomorphism to provide alternative interpretations of the Blass-Lawvere theorem [7,8] i.e., "seven trees in one" presents a promising direction [9-11]. This conclusion indicates that binary trees are isomorphic to seven times themselves.

5. Summary and Outlook

This paper investigates the construction problem of stack permutation-binary tree

isomorphism. By correcting the Hille encoding algorithm, quantitatively analyzing encoding differences, and determining effective length bounds, we have strengthened the theoretical foundation in this field. Specifically, we proved the equivalence between stack encoding and Hille encoding Proposition 3.1, and showed that encoding differences begin to emerge when $n > 3$. A phenomenon further explained through asymptotic analysis of intersection size Corollary 3.3. Additionally, determining the range of effective Hille encoding lengths Proposition 4.1 provides concrete information for evaluating related encodings. Finally, we point out that the methods and techniques presented in this paper may be applied to understand the Blass-Lawvere theorem, which provides new insights for research in related fields.

References

[1] Knott, Gary D. "A Numbering System for Binary Trees." Communications of the ACM, vol. 20, no. 2, ACM New York, NY, USA, 1977, pp. 113–15.

[2] Hille, Reinhold Friedrich. "Stack Permutations and an Order Relation for Binary Trees." Working Paper 82-8, University of Wollongong, 1982.

[3] Flajolet, Philippe, and Robert Sedgewick. Analytic Combinatorics. cambridge University press, 2009.

[4] Knuth, Donald E. The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Pearson Education India, 2011.

[5] Amir Dembo, Thomas M Cover, and Joy A Thoma. "Information Theoretic Inequalities." IEEE Transactions on Information Theory, vol. 37, no. 6, IEEE, 1991, pp. 1501–18.

[6] Munro, J. Ian, and Venkatesh Raman. "Succinct Representation of Balanced Parentheses and Static Trees." SIAM Journal on Computing, vol. 31, no. 3, SIAM, 2001, pp. 762–76.

[7] Blass, Andreas. "Seven Trees in One." Journal of Pure and Applied Algebra, vol. 103, no. 1, Elsevier, 1995, pp. 1–21.

[8] Fiore, Marcelo, and Tom Leinster. "Objects of Categories as Complex Numbers." Advances in Mathematics, vol. 190, no. 2, Elsevier, 2005, pp. 264–77.

[9] Kitaev, Sergey, and Philip B. Zhang. "Non-Overlapping Descents and Ascents in Stack-Sor Permutations." Discrete Applied Mathematics, vol. 344, Elsevier, 2024, pp. 112–19.

[10] Filip Sieczkowski, Sergei Stepanenko, Jonathan Sterling, and Lars Birkedal. "The Essence of Generalized Algebraic Data Types." Proceedings of the ACM on Programming Languages, vol. 8, no. POPL, ACM New York, NY, USA, 2024, pp. 695–723.

[11] Opler, Michal. "An Optimal Algorithm for Sorting Pattern-Avoiding Sequences." 2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2024, pp. 689–99.

