Notes on the Prüfer Code

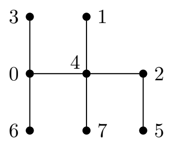
Scribe: Lesly Canales   Lecturer/Editor: Chris Eppolito

24 April 2020

Our goal for today is to illustrate a method of enumerating all trees T with $V(T) = \{0\} \cup [n]$. We let 0 be the *root* of our trees (i.e. 0 is distinguished). A *leaf* of a rooted tree is a non-root vertex of degree one.

Our method for enumerating labeled trees naturally involves efficiently encoding such trees.

Example 1. We explore several methods of encoding trees, illustrating on the tree below.



1. Graph encoding.

   We can naïvely encode our tree as a graph $T = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \{03, 06, 04, 14, 24, 25, 47\})$.

2. Matrix Encoding.

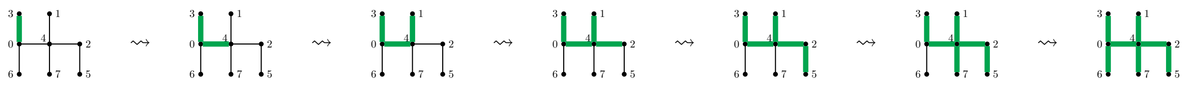
   As trees are connected, every vertex is the end of some edge (unless $n = 0$); thus we can recover the vertex set $V(T) = \{0\} \cup [n]$ from the edge set, and we encode $E(T)$ in a $2 \times n$ matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 2 & 2 & 4 \\ 3 & 6 & 4 & 4 & 4 & 5 & 7 \end{bmatrix}$$

   Note that there are many encodings of T by such a matrix; we want a canonical encoding.

3. Depth First Search Matrix.

   Starting at the root 0, we use a depth-first-search to traverse the tree and recording edges in a column as we encounter them (we record the parent in the top row and the child in the bottom row). We first follow the minimum-possible-index at a branching if there is a choice.



   This yields a canonical encoding of T (we have eliminated all possible choices). Note that the bottom row of the depth-first-search matrix is a permutation of the non-root nodes of T.

$$\begin{bmatrix} 0 & 0 & 4 & 4 & 2 & 4 & 0 \\ 3 & 4 & 1 & 2 & 5 & 7 & 6 \end{bmatrix}$$

A best possible encoding of a tree would forget a row of our matrix, only storing the minimum necessary data to reconstruct the tree. None of the encodings thus far are structured enough for this kind of reduction.

We will now describe the Prüfer[1] code. Instead of traveling from the root outwards (as we did with the depth-first-search), we'll decompose the tree by removing one leaf at a time. In order to make sense of this step, we need the following simple lemma (which we have seen before) and corollary.

[1]This name is German; it is pronounced roughly the same as you would say "proofer" English.

Lemma. *Every tree with at least two vertices has a leaf.*

Corollary. *Every rooted tree can be obtained from K_1 by adding leaves.*

Given a sequence $S = (s_1, s_2, \dots, s_k)$ and an x, for ease of notation we let $(S, x) \coloneqq (s_1, s_2, \cdots, s_k, x)$ denote the sequence obtained by appending x to the end of S. Now we give the algorithm defining the Prüfer code of a labeled tree.

Algorithm (Prüfer Code Construction)**.** Let T be a tree with $V(T) = \{0\} \cup [n]$ and root 0.

1. Let $k = 0$ and define $T_0 \coloneqq T$ and $C_0 \coloneqq ()$.
2. While $k \leq n - 1$:
   - (a) Let v_k denote the minimum leaf of T_k and let u_k denote its unique neighbor in T_k.
   - (b) Define $T_{k+1} \coloneqq T_k \setminus v_k$ and $C_{k+1} \coloneqq (C_k, u_k)$.
   - (c) Increment k and continue.
3. Output C_{n-1}, the *Prüfer code* of T.

Example 2. We compute the Prüfer code of the labeled tree from Example 1. We visualize the construction as a sequence of matrices, adding the minimum-index leaf-edge at each step; we enter the leaf at the top row of the matrix and its parent on the bottom row. The first $7 - 1$ entries of the bottom row is the Prüfer code.

$$\begin{bmatrix} 1 & 3 & 5 & 2 & 6 & 7 & 4 \\ 4 & 0 & 2 & 4 & 0 & 4 & 0 \end{bmatrix}$$

Remark. Let T be a rooted tree with $V(T) = \{0\} \cup [n]$.

1. The Prüfer code of T is a sequence in $V(T)$ with $n - 1$ entries; this encoding does not require us to keep the last parent (which would always be 0).
2. For each $n \in \mathbb{N}$ the Prüfer code defines a function $P_n \colon \{\text{Trees on } \{0, 1, \dots, n\}\} \to \{0, 1, \dots, n\}^{n-1}$.

The Prüfer code recovers the tree structure; the inverse function P_n^{-1} is given via the algorithm below.

Algorithm (Inverse Prüfer Code Construction)**.** Let $C \in \{0, 1, \dots, n\}^{n-1}$.

1. Define $k = 1$, let $A_1 \coloneqq \{i \in [n] : i \text{ is not an entry of } C\}$, and define $L_1 = ()$.
2. While $A_k \neq \emptyset$:
   - (a) Define $m_k \coloneqq \min A_k$ and $L_{k+1} \coloneqq (L_k, m_k)$.
   - (b) Let a denote the k^{th} entry of C and define $A_{k+1} \coloneqq A_k \setminus \{m_k\}$ if a appears in C after the k^{th} entry, and $A_{k+1} \coloneqq A_k \setminus \{m_k\} \cup \{a\}$ otherwise.
   - (c) Increment k and continue.
3. Output $T = (\{0, 1, \dots, n\}, \{l_k c_k : k \in [n]\})$ where $c_n = 0$.

Example 3. Tree assigned to the code $C = (4, 0, 2, 4, 0, 4)$ is computed precisely the tree from Example 1. We visualize the procedure as filling out the top row of a matrix with bottom row $(C, 0)$, adding the smallest available label to the top row at each step.

The following proposition (which we will not prove here) yields the solution to our initial question.

Proposition. *For all $n \in \mathbb{Z}^+$, the function P_n is a bijection.*

Corollary. *There are $(n + 1)^{n-1}$ labeled trees on $n + 1$ vertices for all $n \in \mathbb{N}$.*