The Prüfer correspondence

Contents

1 The Prüfer correspondence

1.1 Labelled objects briefly

Consider a combinatorial class $\mathcal{C}$ which has a specification, recursive or iterative, involving $\mathcal{E}$, $\mathcal{Z}$, and combinatorial constructions ($+$, $\times$, and SEQ(), but also the ones we haven't studied). Then any object of $\mathcal{C}$ can be viewed as built out of copies of $\mathcal{Z}$.

Definition. With $\mathcal{C}$ as above, a labelling of $c \in \mathcal{C}$ is a bijection from the copies of $\mathcal{Z}$ building up c to $\{1, \ldots, |c|\}$

For example a labelled rooted tree is a rooted tree t where each vertex (these are the copies of $\mathcal{Z}$) is assigned a number in $\{1, \ldots, |t|\}$ and no two vertices are assigned the same number.

Two labelled combinatorial objects are the same if the unlabelled objects which build them up are the same *and* the bijections are the same.

For more detail on labelled combinatorial objects read the supplemental notes.

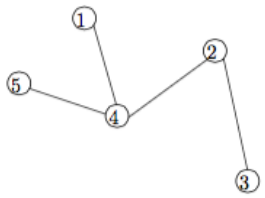
1.2 The Prüfer correspondence for labelled non-rooted trees

For the Prüfer correspondence we are interested in labelled non-rooted trees. Let $\mathcal{U}$ be the set of non-rooted trees, that is the set of connected graphs with no cycles. Let $\mathcal{L}$ be the set of labelled non-rooted trees, that is each element of $\mathcal{L}$ is a pair

$$(t, f)$$

with $t \in \mathcal{U}$ and f a labelling of t.

For example, the following tree t has five atoms (the nodes), and f bijectively maps the five atoms to $\{1, 2, 3, 4, 5\}$.



How should we represent a tree like this for our algorithm? Since the vertices are labelled we can think of the edges as sets of two positive integers. It will be most useful to represent the trees by their sets of edges. For example, the above tree t would be represented by the pair (E, n), where $n = 5$ and $E = \{\{5, 4\}, \{1, 4\}, \{4, 2\}, \{2, 3\}\}$ is the edge set of t.

The Prüfer correspondence is an algorithm which takes a tree $t \in \mathcal{L}$ and returns Prüfer(t), where Prüfer(t) is a list of length $|t| - 2$ of integers in $\{1, 2, \ldots, |t|\}$. To show this is bijective we will describe both the algorithm `Prufer` and the inverse algorithm `invPrufer`.

Algorithm: Prufer

```
input: E, n.   E is the edge set of a labelled non-rooted tree of size n
d = (0,...,0) (length n)
for {x,y} in E
  d(x) = d(x)+1
  d(y) = d(y)+1
for i from 1 to n-2
  x=n
  while d(x) != 1
    x=x-1
  y=n
  while {x,y} not in E
    y=y-1
  L(i)=y
  d(x)=d(x)-1
  d(y)=d(y)-1
  E = E - {x,y}
output: L
```

What is going on in this algorithm? The first loop makes d into the vector of the degrees of each vertex. Inside the second for loop, the first while loop searches for the highest-labeled leaf x, and the second while loop finds the edge $\{x, y\}$ incident to this leaf. Then the other end of this edge, y, is appended to the list, and this edge $\{x, y\}$ is removed.

Applying this to the example graph above, the edge set is

$$E = \{\{1, 4\}, \{2, 3\}, \{2, 4\}, \{4, 5\}\}$$

First we calculate

$$d = (1, 2, 1, 3, 1)$$

Now in the second for loop, we have $i = 1$. $x = 5$ is the largest leaf, $y = 4$ gives the edge, so $L(1) = 4$ and

$$E = \{\{1, 4\}, \{2, 3\}, \{2, 4\}\} \quad d = (1, 2, 1, 2, 0)$$

Now we have $i = 2$. $x = 3$ is the largest leaf, and the other end of the edge is $y = 2$, so $L(2) = 2$ and

$$E = \{\{1, 4\}, \{2, 4\}\} \quad d = (1, 1, 0, 2, 0)$$

Now we have $i = 3$. $x = 2$ is the largest leaf, and the other end is $y = 4$, so $L(3) = 4$ and

$$E = \{\{1, 4\}\} \quad d = (1, 0, 0, 1, 0)$$

and the algorithm terminates returning

$$L = (4, 2, 4)$$

Observe the following fact

Proposition. *Let $t \in \mathcal{L}$ and let E be the edge set of t. Then each vertex v appears $\deg(v) - 1$ times in the sequence output by* PRÜFER$(E, |t|)$.

Proof. Observe two things: First, at each step of the algorithm we remove an edge incident to a leaf, so the graph remains connected throughout. Second, the algorithm terminates before we remove the final edge.

At each stage of the algorithm, a vertex labeled y is made a member of the list $\texttt{L} = \texttt{Prufer}(E, |t|)$ only if there is an edge in E joining y to another vertex x, where x is a leaf of the current tree. If y is also a leaf of the current tree, then the graph at that stage had only one edge. This is impossible because the algorithm terminates before this last edge is considered. Thus y is not a leaf in the current tree, whenever y is added to the list. Furthermore, when an edge is removed, the degree of its two incident vertices goes down, so each vertex y can appear at most $\deg(y) - 1$ times in $\texttt{Prufer}(E, |t|)$.

Next we notice that

$$\sum_{y\in[n]}(\deg(v)-1)=\left(\sum_{y\in[n]}\deg(y)\right)-n=2e(t)-n$$

where $e(t)$ is the number of edges of t. But t is a tree so $e(t)+1=n$. Therefore

$$\sum_{y\in[n]}(\deg(y)-1)=e(t)-1,$$

which is the length of the list. Since each vertex y appears at most $\deg(y)-1$ times in the list, every vertex y must appear in the list $\texttt{Prufer}(E,n)$ exactly $\deg(y)-1$ times. □

This observation lets us construct the inverse algorithm

Algorithm: InvPrufer

```
input: L, n.   L is a list of length n-2 with elements in {1,...,n}
L(n-1)=1
d = (1,...,1) (length n)
for i from 1 to n-2
  d(L(i))=d(L(i))+1
for i from 1 to n-1
  x=n
  while d(x) != 1
    x=x-1
  y=L(i)
  d(x)=d(x)-1
  d(y)=d(y)-1
  E = E union {x,y}
output: E
```

Example. Without using a computer, find INVPRUFFER(L), where $L=(4,2,4)$.

Solution:

1. The list L has length 3, so $n=5$ and the vertex set is $V=[5]$.
2. Find the degree sequence d of the tree by adding 1 to the frequency in L of each vertex.
$$d=(d_1,d_2,d_3,d_4,d_5)=(1+0,\ 1+1,\ 1+0,\ 1+2,\ 1+0).$$
3. Make a table with columns labeled d, x and y, and initialized with d in row 1 and L in column 3, as shown at right.
4. Execute steps 4(a) and 4(b) exactly $n-2$ times.
   (a) Find the highest labeled leaf, x, and write it in the table. Note x is the index of the rightmost 1 in d.
   (b) Decrement the degrees of vertices x and y to get the next row of d. Note: Each row d is the degree sequence of the tree after deleting all the vertices x appearing earlier in the table.
5. Finally, write the indices x, y, of the two 1's in the last row d. Necessarily, either x or y will equal 1.
6. Output: The edges of the reconstructed tree INVPRUFFER(L) are the $n-1$ pairs $\{x,y\}$ listed in the table.

Input: $L=(4,2,4)$.

$V=\{1,2,3,4,5\}$

$d=$					E	
d_1	d_2	d_3	d_4	d_5	x	y
1	2	1	3	1		4
						2
						4

d					x	y
1	2	1	3	1	5	4
1	2	1	2	0		2
						4

⋮

d					x	y
1	2	1	3	1	5	4
1	2	1	2	0	3	2
1	1	0	2	0	2	4
1	0	0	1	0	1	4

Output:
$E=\{\{5,4\},\{3,2\},\{2,4\},\{1,4\}\}$.

Why is this the inverse algorithm? By the proposition, both algorithms calculate the same degree sequence d. Then both algorithms in the second for loop begin by finding the largest leaf remaining. PRUFER next finds the corresponding edge and puts its other end in the list, while INVPRUFER takes the corresponding element out of the list and puts that edge in the edge set. Then both algorithms update the degrees of the vertices incident the edge.

The only question remaining is the final edge. Note that in PRUFER the leftover edge must include vertex 1 (otherwise it would have been taken at an earlier step), and so its other end must be strictly larger. Thus if the algorithm were run one step longer the next entry of L would be 1. INVPRUFER, puts this extra 1 back in L and thus gives back the final edge at the final step.

1.3 Ranking and unranking for labelled non-rooted trees

The Prüfer correspondence gives us a bijection between $\mathcal{L}_n$ and the set of lists of length $n-2$ using the letters $\{1,\ldots,n\}$. There are n^{n-2} such lists so we have

Proposition. *The number of n-node labeled trees using the letters $\{1,2,\ldots,n\}$ is*

$$|\mathcal{L}_n| = n^{n-2}.$$

We can also order trees according to the lexicographic order of their lists. We can view these lists as the sequences of digits of a number in base n representation. All such numbers are valid, so this gives a simple rank and unrank.

Algorithm: RankTree

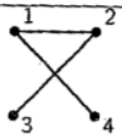
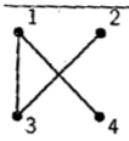
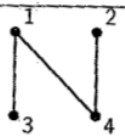
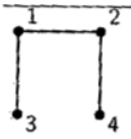
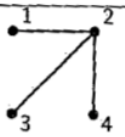
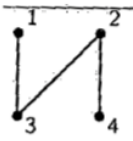
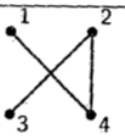
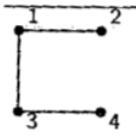
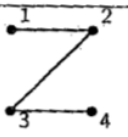
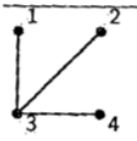
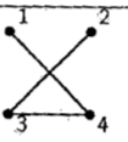
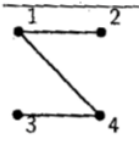
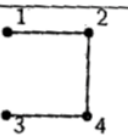
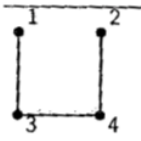
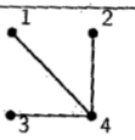
```
input: E, n.   E is the edge set of a labelled non-rooted tree of size n
L=Prufer(E,n)
r=0
p=1
for i from n-2 to 1
  r = r + (L(i)-1)p
  p = np
output: r
```

Algorithm: UnankTree

```
input: r, n.
for i from n-2 to 1
  L(i) = (r mod n) + 1
  r = floor((r-L(i)+1)/n)
output: InvPrufer(L,n)
```

Here's a table of the ranks of all labelled trees on 4 vertices from Kreher and Stinson, *Combinatorial Algorithms*, section 3.3.

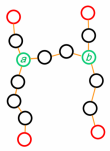
TABLE 3.5
The labeled trees on four vertices and their ranks

T	Prüfer(T)	rank(T)	T	Prüfer(T)	rank(T)
	$(1,1)$	0		$(1,2)$	1
	$(1,3)$	2		$(1,4)$	3
	$(2,1)$	4		$(2,2)$	5
	$(2,3)$	6		$(2,4)$	7
	$(3,1)$	8		$(3,2)$	9
	$(3,3)$	10		$(3,4)$	11
	$(4,1)$	12		$(4,2)$	13
	$(4,3)$	14		$(4,4)$	15

1.4 Counting trees with restricted degrees

Since the list $L = \text{PRUFFER}(t)$ encodes the degree sequence of the tree t, we can often use basic combinatorics to determine the number of labeled trees with specified degree sequences.

Example. An *H-tree* is a vertex-labeled tree with degree sequence $3, 3, 2, 2, \ldots, 2, 1, 1, 1, 1$. A typical H-tree (without labels) is shown at right. How many H-treess t have n vertices?



Solution: Let us use the notation $[n] = \{1, 2, \ldots, n\}$. We must count the sequences $L \in [n]^{n-2}$ having exactly $n - 4$ distinct entries where two of those entries, say $a, b \in [n]$, each appear twice in L. Here is one way to do this.

There are $\binom{n}{4}$ choices for the set of leaf vertices. Among the set N of *non-leaf* vertices here are $\binom{n-4}{2}$ ways to select the pair $\{a, b\}$. We temporarily distinguish the two occurrances of a in L by using subcripts a_1, a_2. We do the same with b. There are $(n-2)!$ permutations of $N \cup \{a_1, a_2, b_1, b_2\} \setminus \{a, b\}$. Each such permutation becomes the Prüfer sequence of an H-tree after suppressing the four subscripts.

Because of the subscripts, the Prüfer sequence of each H-tree is constructed exactly $2!\,2! = 4$ times in this way. Therefore the number of H-trees with n vertices is

$$\binom{n}{4} \cdot \binom{n-4}{2} \cdot (n-2)! \cdot \frac{1}{2!\,2!}.$$

This expression can be rewritten several ways such as

$$\binom{n-2}{4}\binom{n}{2}\binom{n-2}{2}\binom{n-4}{2}(n-6)!$$