The Prüfer correspondence

Contents

1 The Prüfer correspondence **1**
1.1 Labelled objects briefly 1
1.2 The Prüfer correspondence for labelled non-rooted trees 1
1.3 Ranking and unranking for labelled non-rooted trees 3

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 $\textsc{Seq}()$, 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 examples 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.

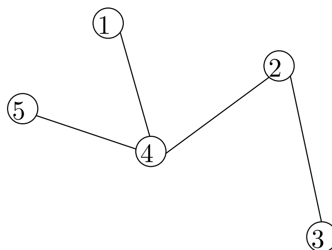
1.2 The Prüfer correspondence for labelled non-rooted trees

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



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.

The Prüfer correspondence is an algorithm which takes a tree $t \in \mathcal{L}$ and returns a list of length $|t| - 2$ of integers in $\{1, 2, \ldots, |t|\}$. This correspondence is bijective and so we can also define the inverse Prüfer algorithm.

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 a leaf, and the second while loop finds the edge incident to this leaf. Then the other end of this edge is put in the list, and this edge 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 E the edge set of t. Then x appears deg$(x) - 1$ times in* PRUFER$(E, |t|)$.

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.

A vertex v only appears in PRUFER$(E, |t|)$ if it has an edge incident to it whose other end was a leaf at that stage of the algorithm. If v is also a leaf then the graph at that stage had only one edge, but this is impossible because the algorithm terminates before this last edge is considered. Thus v is not a leaf when it is added to the list.

Furthermore, when an edge is removed, the degree of the two incident vertices goes down, so each vertex x can appear at most deg$(x) - 1$ times.

Next notice that

$$\sum_{v \in t}(\deg(v) - 1) = \sum_{v \in t}\deg(v) - v(t) = 2e(t) - v(t)$$

where $e(t)$ is the number of edges of t and $v(t)$ is the number of vertices of t. But t is a tree so $e(t) + 1 = v(t)$. Thus

$$\sum_{v \in t}(\deg(v) - 1) = e(t) - 1$$

which is the length of the list. So each vertex x must appear exactly deg$(x) - 1$ 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
```

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 update to take care of this 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 $\{1,\dots,n\}$. There are n^{n-2} such lists so we have

Proposition.

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

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