Generating Trees

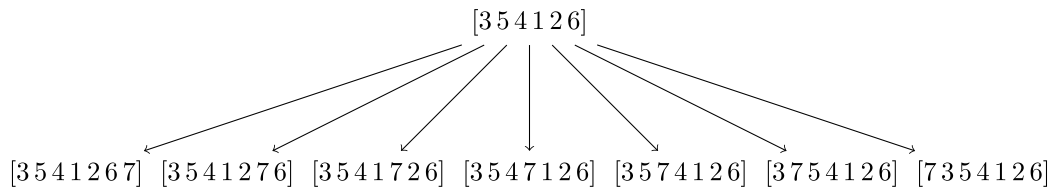
Contents

1 Generating Trees

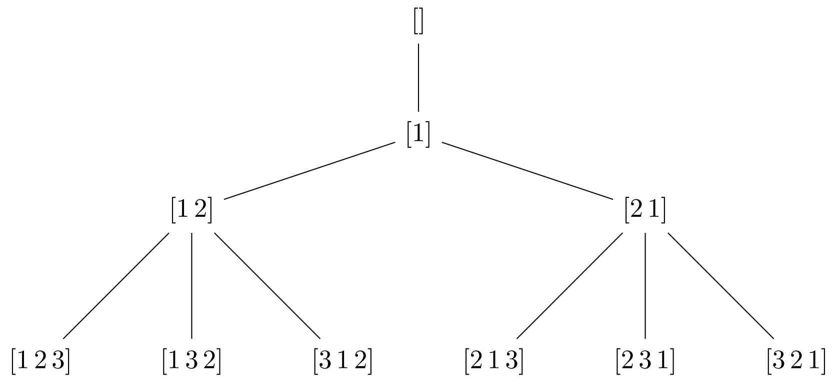
Generating trees offer a more global approach for exhaustive generation. They were described in this form as early as 1978 by Chung, Graham, Houggatt and Kleeman for some restricted families of permutations. They are useful for enumeration and for exhaustive generation.

1.1 Permutations

We start this discussion with an example, permutations. We can generate a subset of permutations of length $n+1$ from a permutation of length n. Consider the one line notation for the permutation σ– we can insert n into any of the positions.



Any permutation of length n will yield $n+1$ new permutations. Thus, all $n!(n+1) = (n+1)!$ are generated. We can make a tree starting from $[1]$ to generate permutations. The start looks like:



The key feature is that we can predict the number of children of any node: If it is a permutation of size n, then it has $n+1$ children. Let k be the number of children of a given vertex. A permutation of size n has $n+1$ children, hence label $n+1$. We know the label for each of the children, and we can list them in the following notation, where the first component indicates the root.

We write this

$$[(1); \{(k) \to \underbrace{(k+1)(k+1)\cdots(k+1)}_{k\,\text{times}}\}] \equiv [[(0); \{(k) \to (k+1)^k\}].$$

1.2 Generating Trees

Let us now generalize this idea.

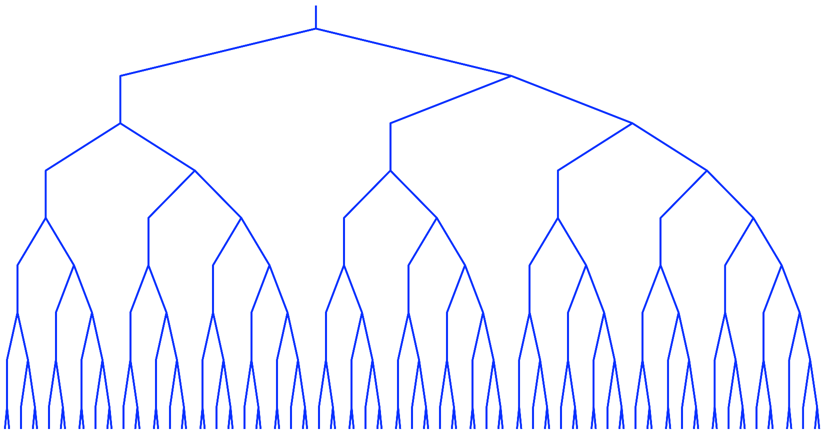
Definition. A generating tree is a rooted labeled tree whose labels follow a specied rule of the form:

$$[(m); \{(k) \to (a_1(k)), (a_2(k)), ..., (a_k(k))\}]$$

where the root of the tree has label m and for each node with label k, the node has k children whose labels are respectively: $a_1(k), a_2(k), \ldots, a_k(k)$.

Example (Fibonacci Numbers)**.** If you think about the bunny interpretation of Fibonacci numbers (An adult bunny (label 2) gives rise to a baby bunny (label 1) and continues on. A baby bunny grows up to become an adult:

$$[(1); (1) \to (2), (2) \to (1)(2)]$$

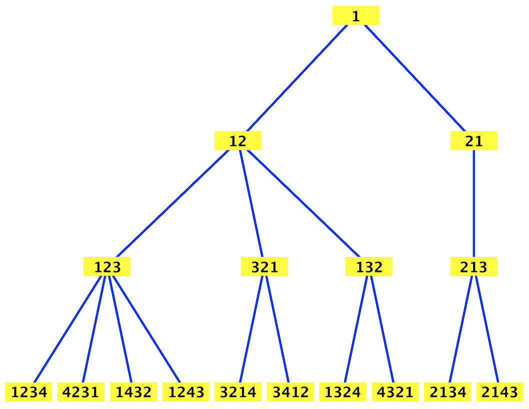


Each vertex represents a bunny, and the bunnies at depth n represent the bunny population at time n. The number of vertices on level i is F_i, the i-th Fibonacci number.

Example (Involutions)**.** An involution is a permutation which is its own inverse.

We can generate involutions of size $n+1$ by multiplying an involution, σ of size n, by any transposition $(i, n+1)$ where i is a fixed point of σ or by leaving $n+1$ as a fixed point. Thus, the number of children is one more than the number of fixed points. The involution created by adding a fixed point has one additional fixed point, and the other children will have one fewer:

$$[(2); (k) \to (k+1)(k-1)^{k-1}]$$



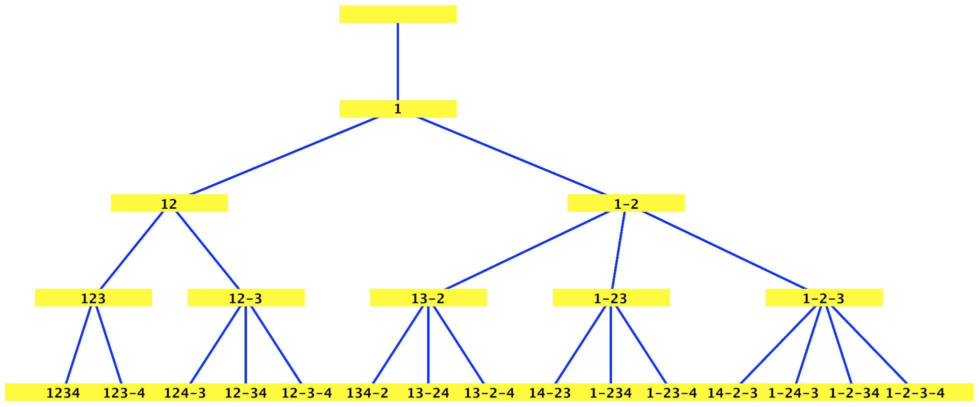
Example (Set Partitions)**.** A set partition of $X = \{1, \dots, n\}$ is a set of subsets of X which are disjoint and whose union is X. For example $\{\{1,4\},\{2\},\{3\}\}$ is a set partition of $\{1,2,3,4\}$. To save brackets we can write this more compactly as $14-2-3$.

Now for the generating tree. The label of a set partition is the number of parts. We create a set partition of size $n+1$ by taking a set partition $\pi = \pi_1|\pi_2|\dots|\pi_k$ of size n and either make a new part consisting of $n+1$ alone, or we add $n+1$ to any of the blocks. Thus, the generation rule is:

$$\pi \to (\pi_1 \cup \{n+1\}|\pi_2|\dots|\pi_k)(\pi_1|\pi_2 \cup \{n+1\}|\dots|\pi_k)\dots(\pi_1|\pi_2|\dots|\pi_{k-1}|\pi_k \cup \{n+1\})(\pi|\{n+1\})\}$$

The generating tree by number of children is

$$[(1);(k) \to (k)^{k-1}(k+1)]$$



1.3 Generating trees to ogf

A key advantage of a generating tree approach, is that we can access the generating functions. We can also predict some properties about the generating function. Let F_n be the number of elements at depth n, and let $F(z) = \sum_n F_n z^n$.

Theorem. *If finitely many labels appear in the tree, then $F(z)$ is rational.*

There are also conditions for algebraicity.