Generating Trees

Contents

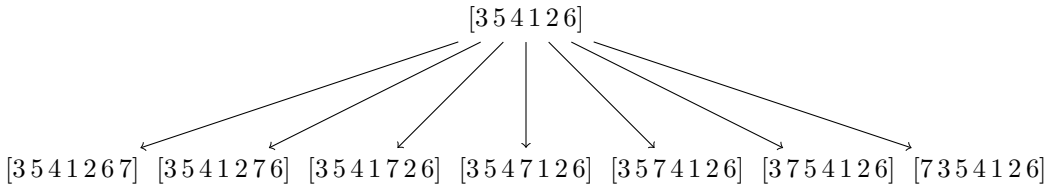
1 Generating Trees **1**
1.1 Permutations 1
1.2 Generating Trees 2
1.3 Generating trees to ogf 3

1 Generating Trees

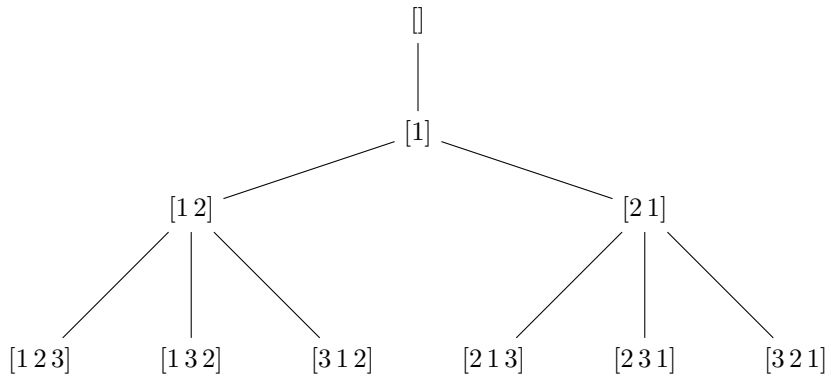
Generating trees offer a more global approach for exhaustive generation. They were described in this form as early as 1978 by Chung, Graham, Houggatt and Kleeman for some restricted families of permutations. They are useful for enumeration and for exhaustive generation.

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 $\sigma-$ 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 without knowing the actual permutation represented by the node: If it is a permutation of size n, then it has $n+1$ children. Let us label the node representing a permutation of length n by the symbol (n). By examining the label of a node, we can determine exactly how many children that node has, and what the labels on its children will be according to the following rule.

> If a node has label (k), then that node will have exactly $k+1$ children, each of which will have the label $(k+1)$.

If we specify that the root note gets label (0), the shape of the tree and the labels on its nodes are completely specified by the rule. The rule for generating the labeled the tree for permutations is recorded with the following notation.

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

The initial (0) is the label of the root, and the set contains a set of rules (there is only one rule in this case) which specify the number of children and the labels on its children.

1.2 Generating Trees

Let us now generalize this idea. The following definition can be intimidating, but the examples that follow it should clarify its meaning.

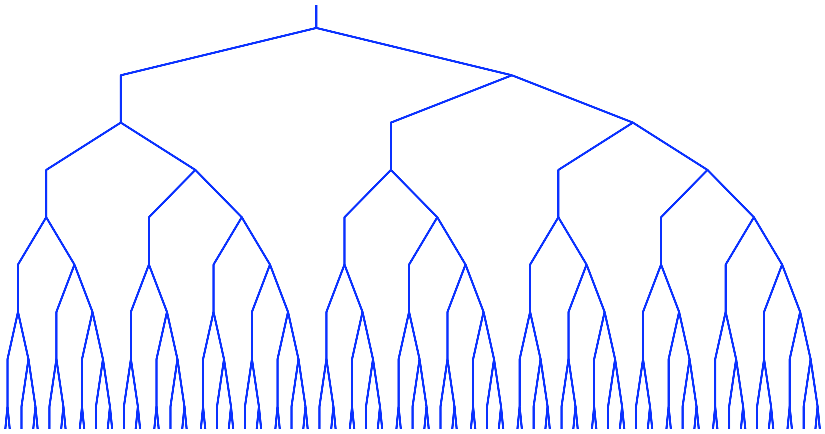
Definition. A generating tree is a rooted labeled tree whose labels follow a specified rule of the form:

$$[(m);\{(k)\to(a_1(k))(a_2(k))\cdots(a_\ell(k))\}].$$

This specifies that the root of the tree has label (m), and for each node with label (k), that node has exactly ℓ children whose labels are, respectively: $(a_1(k))$, $(a_2(k))$, …, $(a_\ell(k))$. Here $a_i(k)$ is a specified function of k for $i=1,2,\ldots,\ell$. If there is a special value of k, say $k=c$ where the labeling rule is different from the general rule labeling rule $(k)\to(a_1(k))(a_2(k))\cdots(a_\ell(k))$, then we can specify the special case rule adding by adding within the curly braces "$\{\ldots\}$" another entry of the form $(c)\to(b_1(c))(b_2(c))\cdots(b_{\ell'}(c))$. We may add several special case rules, if necessary.
Moreover if several of the functions $a_i(k)$ are the same, we can group them together using "exponential notation". For example, the specification $(k)\to(k-1),(k+1),(k+1)$ can be written as $(k)\to(k-1),(k+1)^2$.

Example (Fibonacci Numbers)**.** Let us recall the bunny rabbit interpretation of Fibonacci numbers. Here, each baby bunny gets label (1) and becomes an adult bunny with label (2) in the next generation. Each adult bunny survives to the next generation and gives rise to a new baby bunny. If we start with one baby bunny, the specification is as follows.

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



Each node represents a bunny, and the bunnies at depth n represent the bunny population at the nth generation. The number of nodes on level i is F_i, the i-th Fibonacci number. It is a coincidence that we specified labels in such a way that every node with label (k) has exactly k children in the tree, $k=1,2$. In this example, we specified the labeling rule by listing two special cases, $\{(1)\to\ldots,(2)\to\ldots\}$,

and did not use a general rule of the form $(k) \to \ldots$. Here is an alternative specification of the same generating tree which uses just one general rule instead of two special case rules.

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

If you substitute $k = 1$ and $k = 2$ in to the rule $(k) \to (1)^{k-1}(2)$, then you will get exactly the two special cases in $\{(1) \to (2), (2) \to (1)(2)\}$.

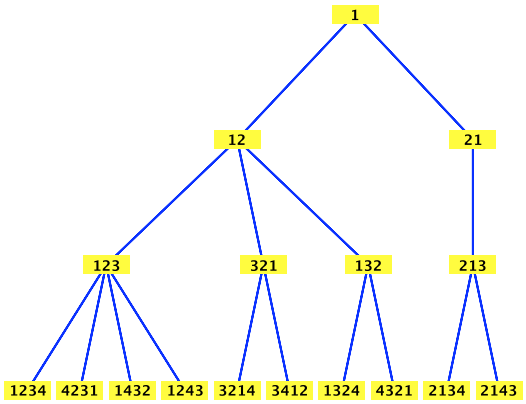
Example (Involutions)**.** An involution is a permutation which is its own inverse. That is, a permutation σ of $\{1, 2, \ldots, n\}$ is an involution if, in cyclic notation, it takes the form

$$\sigma = (a_1 a_2)(a_3 a_4) \ldots (a_{2k-1} a_{2k}) a_{2k+1}, a_{2k+2}, \ldots a_n$$

where $\{a_1, a_2, \ldots, a_n\} = \{1, 2, \ldots, n\}$. Each a pair (a_{2i-1}, a_{2i}) is a *transposition* and each of the elements a_{2k+1}, a_{2k+2}, $\ldots a_n$ is a *fixed point* of σ.

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:

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



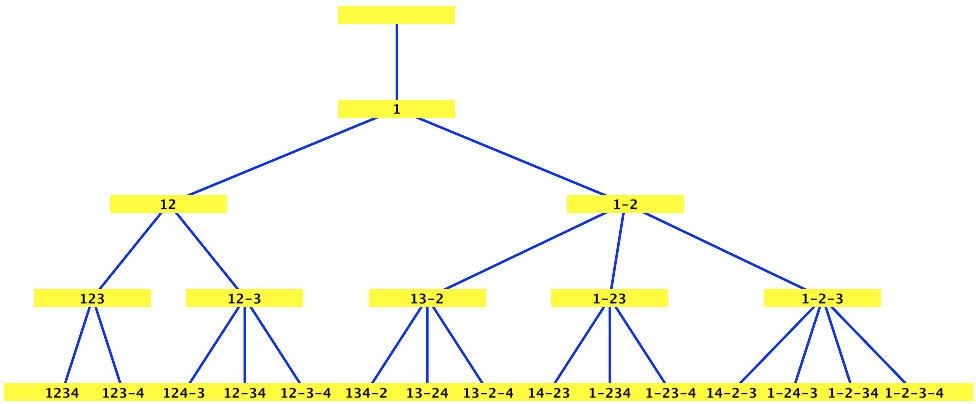
Example (Set Partitions)**.** A set partition of $X = \{1, \ldots, n\}$ is a set of subsets of X which are disjoint and whose union is X. For example $\{\{1, 4\}, \{2\}, \{3\}\}$ is a set partition of $\{1, 2, 3, 4\}$. To save brackets we can write this more compactly as $14|2|3$.

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 $\pi = \pi_1|\pi_2|\ldots|\pi_k$ 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:

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

The generating tree by number of children is

$$[(0); (k) \to (k)^k(k + 1).]$$



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.*

An example of this are the specifications that we saw earlier $[(1); \{(1) \to (2), (2) \to (1)\}]$ which resulted in the Fibonacci numbers and its rational generating function

$$F(z) = \frac{1}{1 - z - z^2}.$$

There are also conditions for *algebraicity* of $F(z)$, where $F(z)$ satisfies a polynomial identity. We saw in Lecture 5 an example of this. The Catalan numbers have an algebraic OGF $C(z)$, namely

$$zC(z) - C(z) + 1 = 0, \text{ which solves to } C(z) = \frac{1 - \sqrt{1 - 4z}}{2z}.$$

Here is a typical result, which is found in [C. Banderier et al, arXiv:math/0411250v1, 2004]. Its proof is technical, and you are not required to know this result for this course.

Theorem. *If a generating tree specification can be obtained from*

$$[(m); \{(k) \to (r)(r+1)(r+2)\dots(k-2)(k-1)\}]$$

by deleting a finite number of terms of the form $(k-b)$ (where $b \geq 1$) and adding a multiset of terms of the form $(k+a)$, where a is an integer, then the ordinary generating function $F(z)$ for the number of objects at level n in the generating tree satisfies a polynomial equation involving z and $F(z)$, which can be computed from the specification (details are omitted here).

For example, with $m = r = 2$ and adding the terms $(k)(k+1)(k+1)$ we obtain the specification

$$[(2); \{(k) \to (2)(3)(4)\dots(k-1)(k)(k+1)^2\}]$$

The resulting generating tree has C_n nodes at depth n, where C_n is the nth Catalan number.