More Probabilistic Proofs of Hook Length Formulas Involving Trees

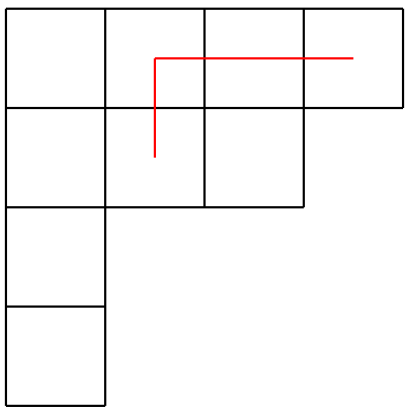
David Grabiner (joint with Mark Tiefenbruck)

March 31, 2011

Motivation: Young tableaux

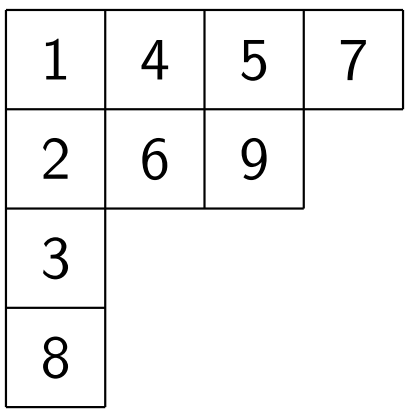
The term "hook length" is motivated by a problem in the enumeration of Young tableaux.

In the Young diagram of a partition, the *hook* of a cell contains the cell, all cells below it, and all cells to the right; the name comes from its shape. The *hook length* h_{ij} of cell (i, j) is the number of cells in the hook, here 4.



Motivation: Young tableaux

A *standard Young tableau* is a filling of the Young diagram with the numbers $1, \ldots, n$ so that all rows and columns are increasing.



Motivation: Young tableaux

The classical hook length formula gives the number of standard Young tableaux of shape λ as

$$n! \Big/ \prod_{(i,j)\in\lambda} h_{ij}.$$

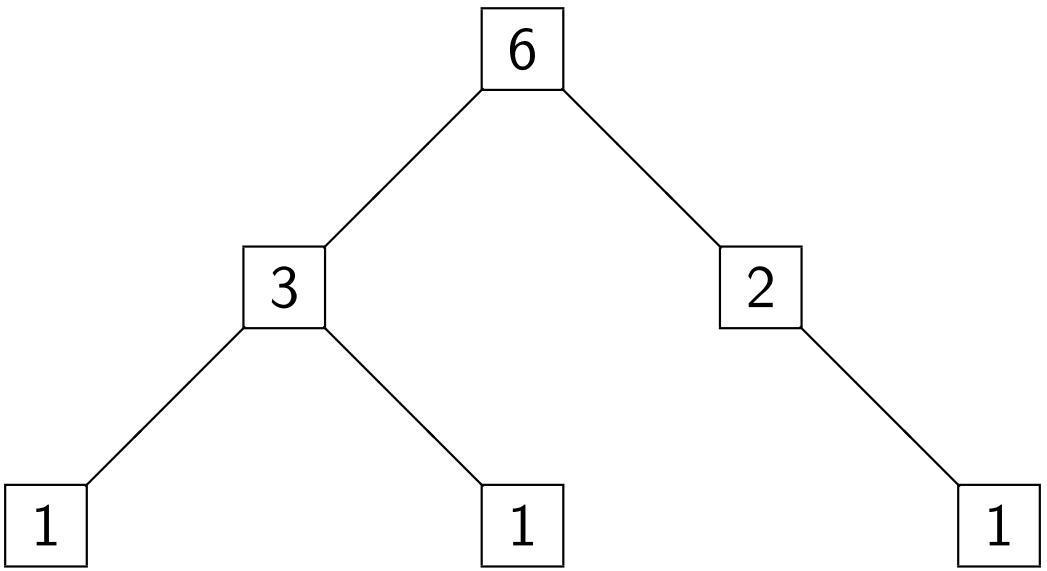
Since the lengths of the hooks appear in the denominator, you might guess that they can be used in a probabilistic proof of the formula; this has been done [Greene/Nijhenhuis/Wilf].

Hooks in trees

Definition

The *hook* of a vertex v in a rooted tree T is the set of all descendants, including the vertex v itself. The *hook length* h_v is the number of vertices in the hook.

Here is a tree with its hook lengths.



A hook length formula for trees

Definition

An *increasing labeling* of a rooted tree is a labeling of the vertices with the numbers $1, \ldots, n$ in which the label of every vertex is less than the labels of all its children.

Theorem (Knuth)

For any rooted tree T with n vertices, the number of increasing labelings is

$$n! \Big/ \prod_{v \in T} h_v.$$

This theorem is well known, and it is easy to prove probabilistically [Sagan/Yeh].

A hook length formula for trees

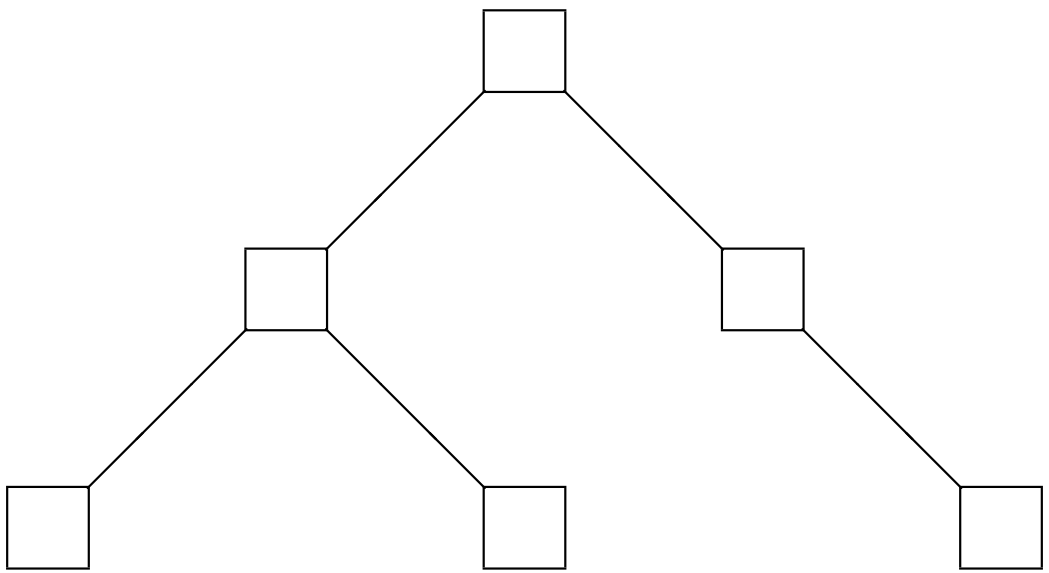
We present a probabilistic proof of this hook length formula which motivates our proofs of the main results.

Choose a random labeling of T with labels in a set S as follows:

1. Assign a random label to the root.
2. If the root has children $v_1, \ldots, v_k$, partition the remaining labels randomly into subsets $S_{v_1}, \ldots, S_{v_k}$ of size $h_{v_1}, \ldots, h_{v_k}$.
3. Apply the algorithm recursively to the hook of each child v_i with the subset S_{v_i}.

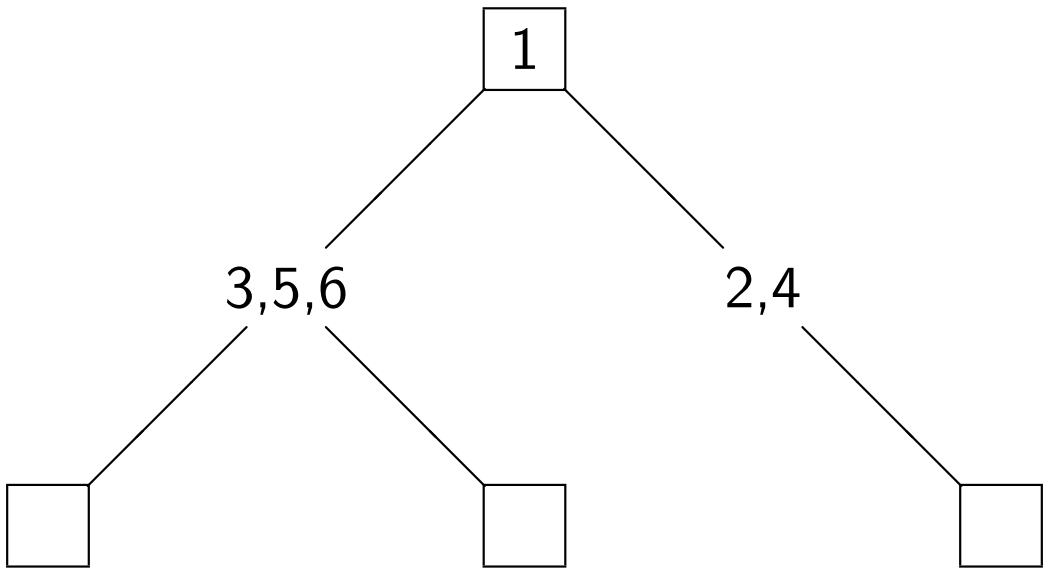
A hook length formula for trees

Here is an example; we happen to hit the $1/(6 \cdot 3 \cdot 2)$ chance that we get an increasing labeling of this tree.



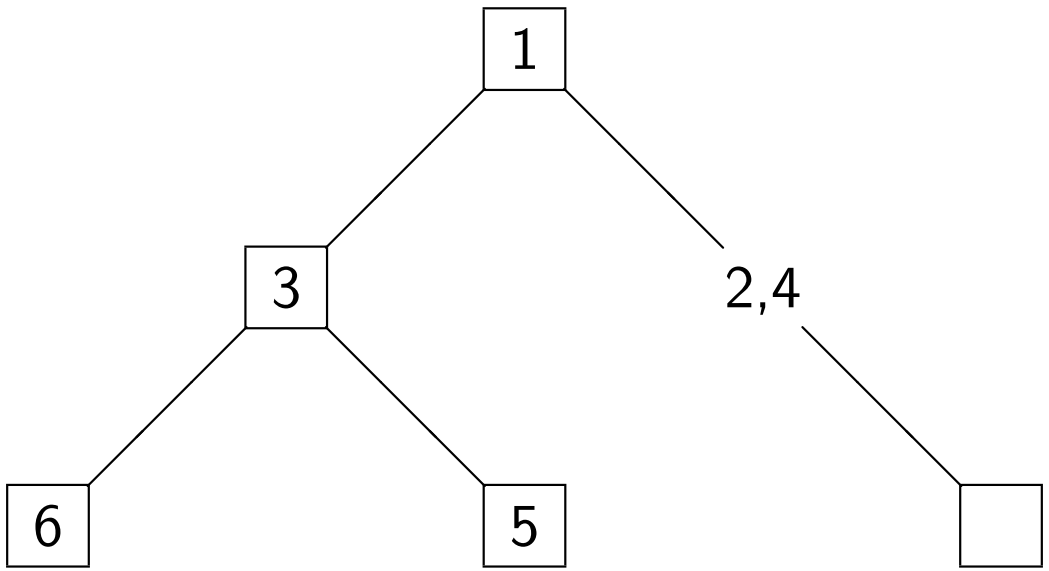
A hook length formula for trees

Here is an example; we happen to hit the $1/(6 \cdot 3 \cdot 2)$ chance that we get an increasing labeling of this tree.



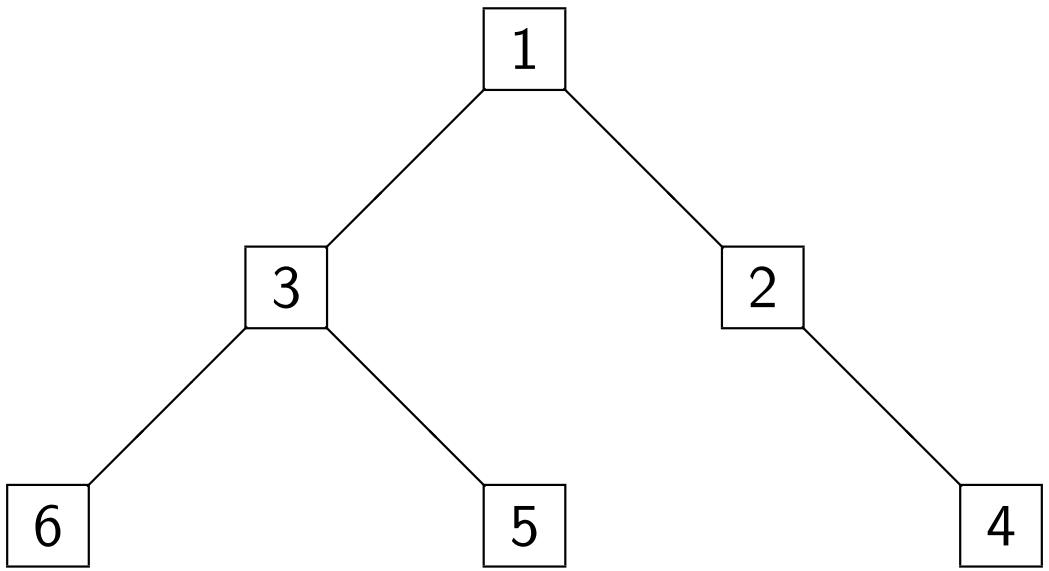
A hook length formula for trees

Here is an example; we happen to hit the $1/(6 \cdot 3 \cdot 2)$ chance that we get an increasing labeling of this tree.



A hook length formula for trees

Here is an example; we happen to hit the $1/(6 \cdot 3 \cdot 2)$ chance that we get an increasing labeling of this tree.



A hook length formula for trees

This algorithm gives a uniform distribution on all labelings, since the root of each hook was assigned a random label from all labels available for the hook. And for each vertex v, the probability that it was assigned the lowest entry in its hook is $1/h_v$, so the probability that we get an increasing labeling is $1/\prod_{v \in T} h_v$, and the total number of increasing labelings is $n!$ times this probability.

Hook length formulas with exponents

Han proved two hook length formulas for sums over all binary trees; the hook lengths appear in the denominator in the form $1/h_v 2^{h_v-1}$. Yang generalized the first formula to a weighted sum over all ordered trees, with a factor $1/h_v m^{h_v-1}$. Unlike other hook length formulas, the hook lengths appear as exponents. Both formulas suggest that there could be a probabilistic proof.

Sagan (presented at this seminar a year ago) found a probabilistic proof of Han's first formula, Yang's generalization, and a further generalization, and conjectured that the second formula has a probabilistic proof as well.

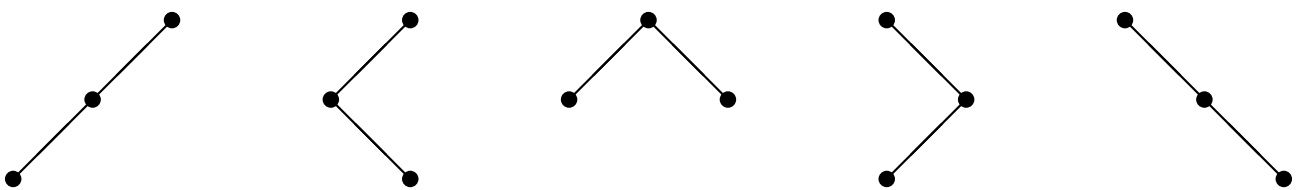
Outline of our work

We use these probabilities directly in an algorithm which gives a probabilistic proof; that is, for each vertex v, we make one h_v-way choice, followed by $h_v - 1$ 2-way or m-way choices to build a random labeled tree. Sagan's algorithm builds a tree by assigning the labels in order; we build the tree recursively by a process analogous to depth-first search.

We also prove the second formula in two ways, using minor variations of both our algorithm and Sagan's.

m-ary trees

We define a binary (or m-ary) tree to be a tree in which each vertex has 2 (or m) ordered slots for children, which may be occupied or vacant. For example, these are the five binary trees on three vertices:



Han's first theorem

Theorem (Han)

For all positive integers n,

$$\sum_{T} \prod_{v \in T} \frac{1}{h_v 2^{h_v - 1}} = \frac{1}{n!},$$

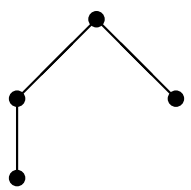
where the sum is over all binary trees on n vertices.

Han proved this by induction, and Sagan gave a probabilistic proof.

Ordered trees

An *ordered tree* is a rooted tree in which an order is assigned to the children of each vertex.

For a fixed m, Yang assigns a weight $w_m(T) = \prod_{v \in T} \binom{m}{c_v}$ to the tree T, where vertex v has c_v children. If m is an integer, this is the number of ways to make T into an m-ary tree, preserving the order of children at each vertex. (Sagan noticed this for $m = 2$, but it works for all m.) For example, the following tree has weight $m\binom{m}{2}$:



Yang's formula

Theorem (Yang)

For all positive integers n and m,

$$\sum_{T} w_m(T) \prod_{v \in T} \frac{1}{h_v m^{h_v - 1}} = \frac{1}{n!},$$

where the sum is over all ordered trees on n vertices.

Sagan gave a probabilistic proof of the formula in this form.

Equivalent form of Yang's formula

Our algorithm works more naturally if we eliminate the weights by converting the ordered trees to m-ary trees; note that Han's theorem is the case $m = 2$.

Corollary

For all positive integers n and m,

$$\sum_T \prod_{v \in T} \frac{1}{h_v m^{h_v - 1}} = \frac{1}{n!},$$

where the sum is over all m-ary trees on n vertices.

Generating a random labeled tree

Here is our main algorithm, which gives our probabilistic proofs.

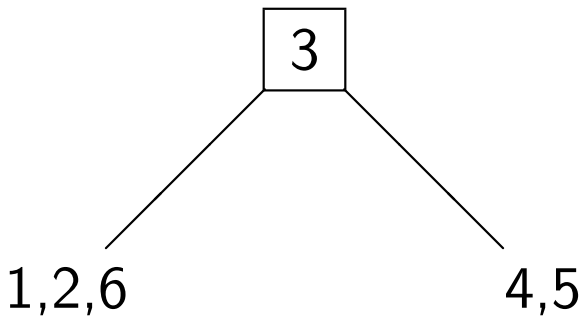
Choose a random labeled m-ary tree with labels in a set S as follows:

1. Assign a random label to the root.
2. Partition the remaining labels randomly into m (possibly empty) subsets $S_1, \ldots, S_m$.
3. If the set S_i is not empty, put a child v_i in slot i, and apply the algorithm recursively to construct subtrees rooted at v_i with vertices labeled by S_i.

Example of the algorithm (for binary trees)

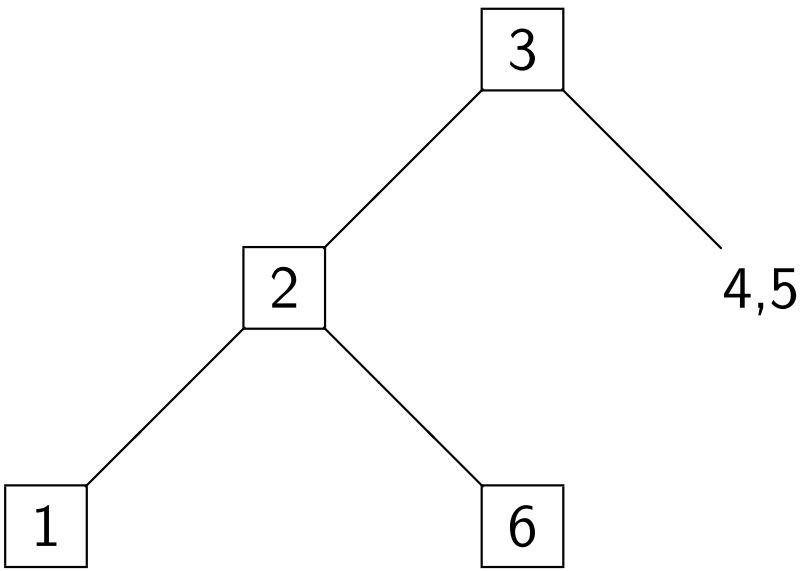
Choices:

Example of the algorithm (for binary trees)



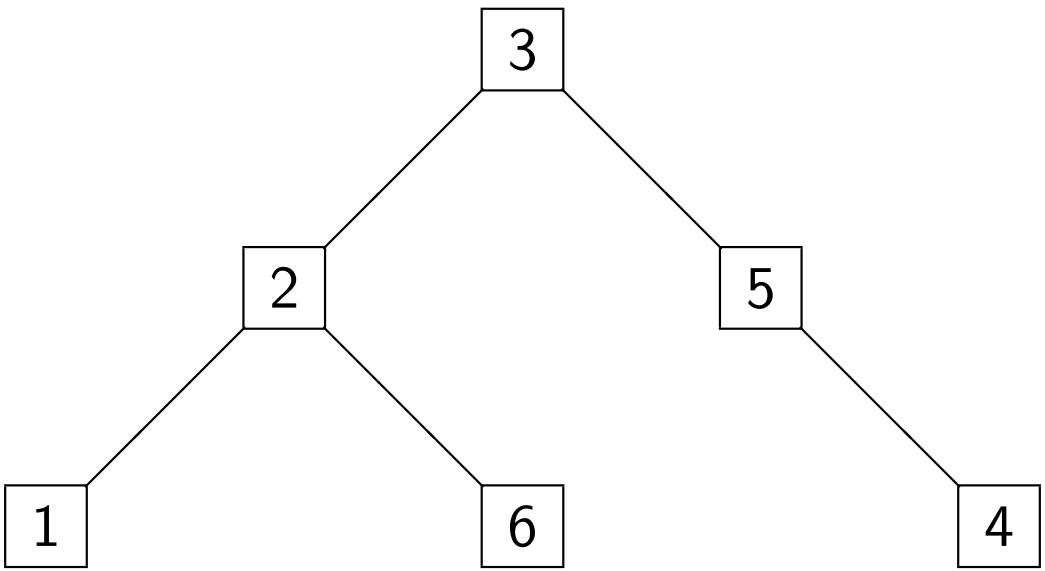
Choices: $6, 2^5$

Example of the algorithm (for binary trees)



Choices: $6, 2^5, 3, 2^2$

Example of the algorithm (for binary trees)



Choices: $6, 2^5, 3, 2^2, 2, 2^1$

Proof of Han's and Yang's formula

With starting set of size n, we can get any labeling of any m-ary tree on n vertices. For each vertex v, we had h_v choices for the label when that vertex was the root, and each of the remaining $h_v - 1$ labels was randomly assigned to one of the m subtrees. Thus the probability of getting a tree T and a particular labeling is

$$\prod_{v \in T} \frac{1}{h_v m^{h_v - 1}}.$$

We sum this over all $n!$ labelings of all m-ary trees to get

$$n! \sum_{T} \prod_{v \in T} \frac{1}{h_v m^{h_v - 1}} = 1,$$

which is Yang's formula multiplied by $n!$.

A further generalization [Sagan]

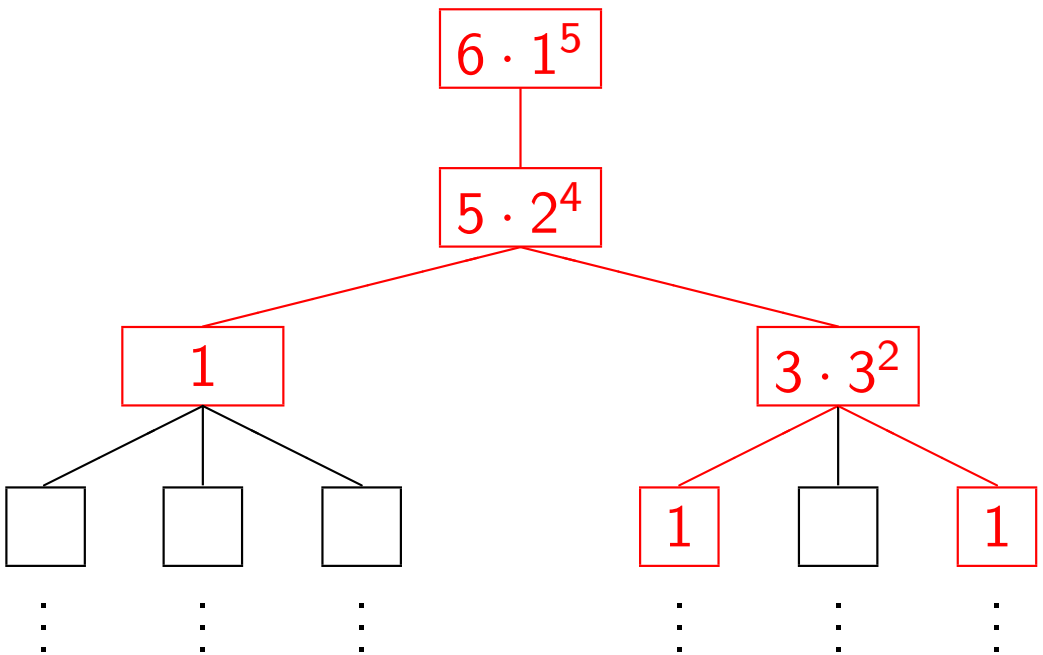
We can allow the value of m to vary with the position of v in the tree. That is, we can take any infinite tree $\overline{T}$ in which a vertex at position v has $m_v > 0$ slots for children, and take a sum over all subtrees:

$$\sum_{T \subset \overline{T}} \prod_{v \in T} \frac{1}{h_v m_v^{h_v - 1}} = \frac{1}{n!}.$$

Our algorithm and proof are exactly the same; Han's and Yang's formulas are the cases in which $\overline{T}$ is an infinite binary or m-ary tree.

Illustration of Sagan's generalization

Here is an example of a subtree of an infinite tree (here, m_v is 1 more than the depth of v, but the algorithms still work even if m_v varies within a depth); each box contains the factor $h_v m_v^{h_v-1}$.



Sagan's algorithm

Here is Sagan's algorithm, for creating a random increasingly labeled tree. (Sagan used different probabilities for his proof of Yang's formula, but his proof of the generalization includes the m-ary version of Yang's formula as a special case.)

1. Start with an empty tree, with probability 1 assigned to adding a root.
2. Choose a vertex slot according to the current probability distribution, and create a vertex v there with the lowest remaining label.
3. Replace the assigned probability $p(v)$ for v with a probability of $p(v)/m_v$ for each of the m_v slots for children of v.
4. Return to step 2 until all labels are used.

Sagan's algorithm

The probability $p(v)$ for an individual vertex is

$$\prod_{w>v} \frac{1}{m_w},$$

the reciprocal of the product of the number of potential children of its ancestors. In other words, it is the probability that a random path starting from the root will pass through v.

In particular, if m_w is a constant m (as in Yang's formula), then the probability assigned to a potential vertex at depth d_v is $1/m^{d_v}$. In our previous example, $m_w = d_w + 1$, giving probability $1/d_v$!

Proof for Sagan's algorithm

The probability we add v is $\prod_{w>v} 1/m_w$, and when we add vertex v, we add 1 to the hook lengths of all its ancestors, so we add a factor of $1/m_w$ to the product $\prod_{w \in T} 1/m_w^{h_w - 1}$ for every $w > v$. Thus the probability of choosing a particular increasingly labeled T with Sagan's algorithm is

$$\prod_{v \in T} p(v) = \prod_{v \in T} \prod_{w > v} \frac{1}{m_w} = \prod_{w \in T} \prod_{v < w} \frac{1}{m_w} = \prod_{w \in T} \frac{1}{m_w^{h_w - 1}}.$$

and summing over all $n!/\prod_{v \in T} h_v$ increasing labelings of all T gives Han's, Yang's, or Sagan's formula.

Modifying our algorithm to match Sagan's distribution

Our algorithm gives an arbitrary labeled tree, not an increasingly labeled tree. To get an increasingly labeled tree, we could always assign the lowest outstanding label in each subtree to the root, rather than choosing a random label. We lose the factor of $1/h_v$, so the probability of choosing a particular increasingly labeled T is

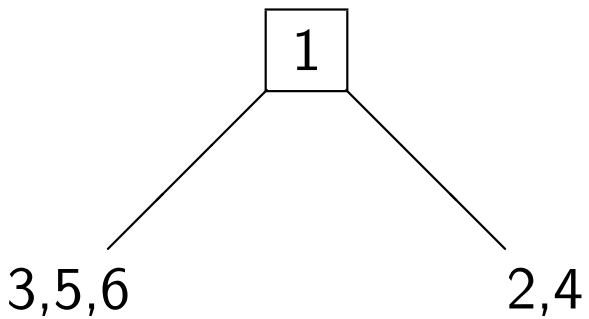
$$\prod_{v \in T} \frac{1}{m_v^{h_v - 1}},$$

which we just showed is the distribution from Sagan's algorithm.

Example of our modified algorithm

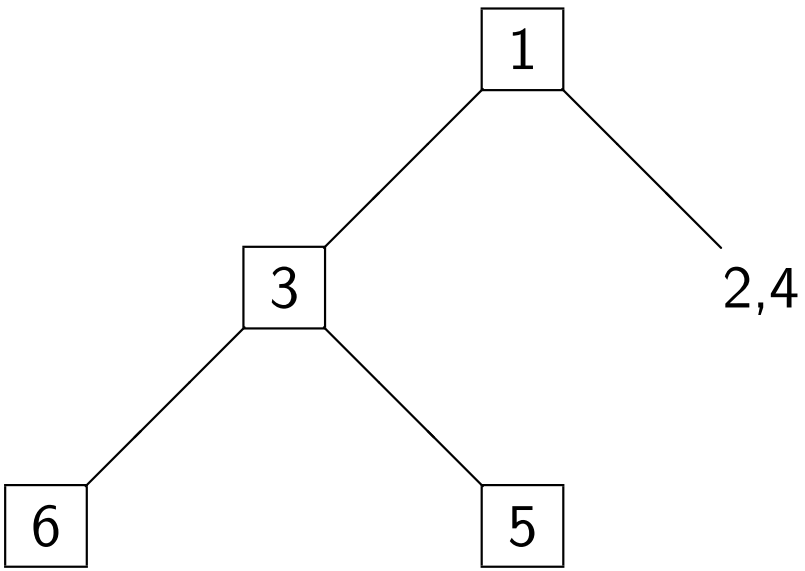
Choices:

Example of our modified algorithm



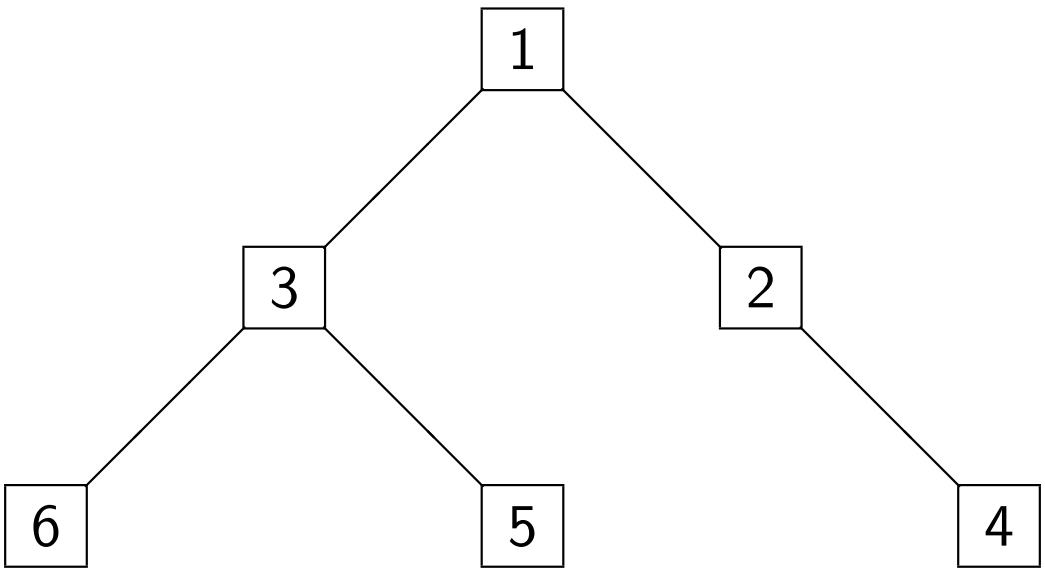
Choices: 2^5

Example of our modified algorithm



Choices: $2^5, 2^2$

Example of our modified algorithm



Choices: $2^5, 2^2, 2^1$

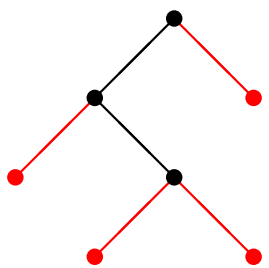
Complete trees

Definition

A binary tree T is *complete* if every non-leaf has two children.

Definition

The *completion* of the binary tree T is the complete tree $\hat{T}$ obtained by adding children to all $n+1$ empty slots, as shown by the red edges below.



Note that completing the tree changes the hook length of v from h_v to $2h_v + 1$.

Han's second formula

Theorem (Han)

For every positive integer n, we have

$$\sum_T \prod_{v \in T} \frac{1}{(2h_v+1)2^{2h_v-1}} = \frac{1}{(2n+1)!}$$

where the sum is over binary trees on n vertices.

This is almost the same form as Han's first formula modified to use the completed tree, except that the factor is 2^{2h_v-1} rather than 2^{2h_v}. Given the formulation in terms of completed trees, Sagan conjectured that there should be a probabilistic proof. We will provide two, using a version of our algorithm and an application of Sagan's algorithm.

Algorithm for complete binary trees

Construct a random labeled complete binary tree with labels in a set S of odd size as follows:

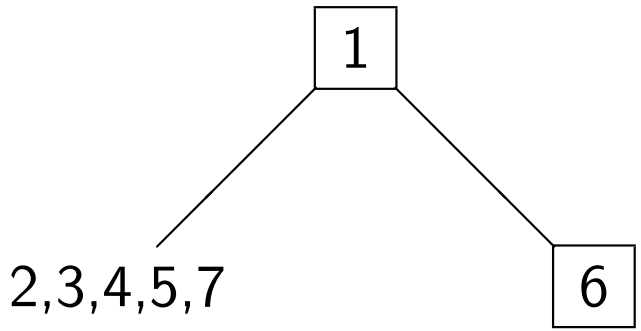
1. Assign a random label to the root.
2. If there is more than one label in S, partition the remaining labels randomly into two subsets S_1 and S_2 of odd size.
3. If we have constructed S_1 and S_2, they are not empty; assign two children v_1 and v_2 to the root, and apply the algorithm recursively to construct subtrees rooted at v_i with vertices labeled by S_i.

Recall that Han's second formula had a factor of 2^{2h_v-1} for every vertex v in T. In the completion $\hat{T}$, the hook length becomes $2h_v + 1$; step 2 gives 2^{2h_v-1} rather than 2^{2h_v} choices because the two subsets must be odd.

Example of the complete-trees algorithm

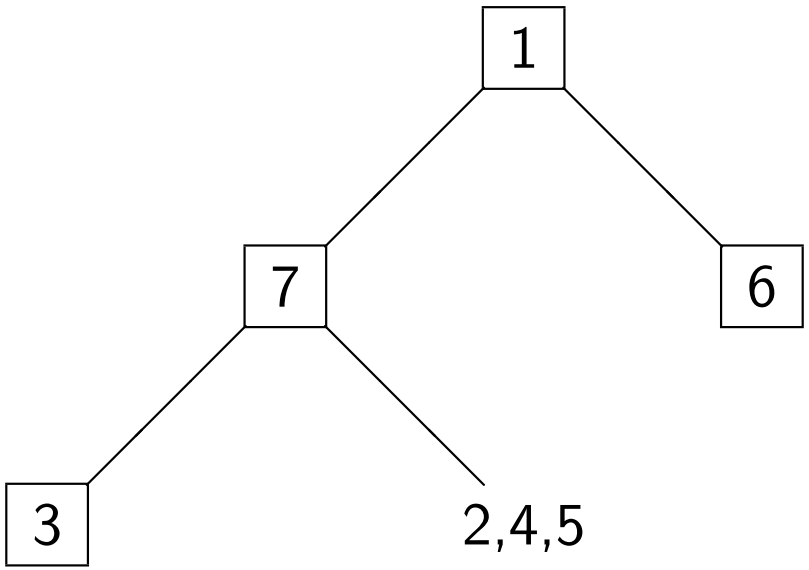
Choices:

Example of the complete-trees algorithm



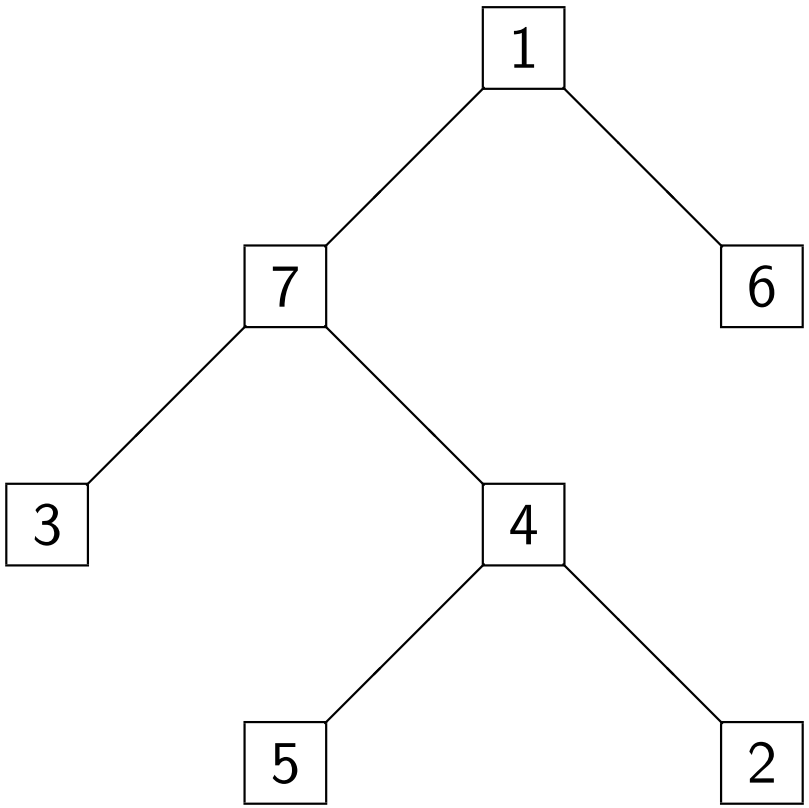
Choices: $7, 2^5$

Example of the complete-trees algorithm



Choices: $7, 2^5, 5, 2^3$

Example of the complete-trees algorithm



Choices: $7, 2^5, 5, 2^3, 3, 2^1$

Proof for complete binary trees

If $\hat{T}$ is the completion of T, then we had $2h_v + 1$ choices for the label at vertex v, where h_v is the hook length in T, and we had 2^{2h_v-1} ways to assign the remaining labels to two subsets of odd size. Therefore, the probability of getting a particular $\hat{T}$ and a particular labeling is

$$\prod_{v \in T} \frac{1}{(2h_v + 1)2^{2h_v-1}}.$$

We sum this over all $(2n+1)!$ labelings of all complete trees to get

$$(2n+1)! \sum_{T} \prod_{v \in T} \frac{1}{(2h_v + 1)2^{2h_v-1}} = 1,$$

which is Han's formula multiplied by $(2n+1)!$

Not as nice for m-ary trees

The analogous formula for complete m-ary trees is not as natural. The probability that a random partition of a set of size $2k$ into two parts has both parts odd is always $1/2$, but the probability that a random partition of a set of size mk into m parts has all parts congruent to 1 mod m depends on the size of the set, so we get an extra factor corresponding to that probability.

Our extension of Sagan's algorithm for binary trees

We will use Sagan's algorithm to construct an increasingly labeled binary tree $\hat{T}$ on $2n+1$ vertices, and then see whether it is complete.

1. Start with an empty tree, with probability 1 assigned to adding a root.
2. Choose a vertex slot according to the current probability distribution, and create a vertex $\hat{v}$ there with the lowest remaining label.
3. Replace the assigned probability $1/2^{d_{\hat{v}}}$ for $\hat{v}$ at depth $d_{\hat{v}}$ with a probability of $1/2^{d_{\hat{v}}+1}$ for each of the two slots for children of $\hat{v}$.
4. Return to step 2 until all labels are used.

Is the tree complete?

A binary tree is complete if and only if every hook length is odd; that is, if every non-leaf vertex has left and right subtrees both of odd size. In Sagan's algorithm, given any vertex which is not a leaf, the probability that its left subtree has an odd number of vertices is $1/2$, since the last leaf added could be added on either side. To get a complete tree with n internal vertices, we must pass n such tests, which has probability $1/2^n$.

The completion of another tree

Our tree $\hat{T}$ is the completion of a tree T, and if $v \in T$ has hook length h_v, then $\hat{v}$ has hook length $2h_v + 1$ in $\hat{T}$. Thus the probability of getting a particular labeling of the completion of T is

$$\prod_{v \in T} \frac{1}{2^{2h_v}},$$

and summing over all $(2n+1)!/\prod_{v \in T}(2h_v+1)$ increasing labelings of all $\hat{T}$ gives

$$(2n+1)! \sum_{T} \prod_{v \in T} \frac{1}{(2h_v+1)2^{2h_v}} = \frac{1}{2^n},$$

as desired.

Main references

Han, G.-N., New hook length formulas for binary trees, *Combinatorica* **30**(2) (2010), 253–256, `arXiv:0804.3638`.

Sagan, B. E., Probabilistic proofs of hook length formulas involving trees, *Seminaire Lotharingien de Combinatoire* **61A**(2009), Article B61A.

Yang, L. L. M., Generalizations of Han's hook length formula, `arXiv:0805.1019`.