Holonomic equations and efficient random generation of binary trees

Pierre Lescanne

École normale supérieure de Lyon

CLA 2023

January 2023

Holonomic equation

An holonomic recurrence is

$$P_s(n)F_{n+s} + P_{s-1}(n)F_{n+s-1} + \ldots + P_0(n)F_n = 0$$

where the $P_s(n)$ are polynomials in n.

The paradigm is

$$(n+1)C_n - 2(2n-1)C_{n-1} = 0$$

an equation for Catalan numbers known from *Olinde Rodrigues* in 1838.

Catalan, Motzkin, Schröder

Numbers

Catalan

$$(n+1)C_n = 2(2n-1)C_{n-1}$$

counts binary trees.

Motzkin

$$(n+2)M_n = (2n+1)M_{n-1} + 3(n-1)M_{n-2}$$

counts unary binary trees.

Schröder

$$3(2n-1)S_n = (n+1)S_{n+1} + (n-2)S_{n-1}$$

counts binary trees in which every nonnull right link is colored either white or black.

Catalan, Motzkin, Schröder

Numbers

Catalan

$$(n+1)C_n = 2(2n-1)C_{n-1}$$

counts binary trees.

Motzkin

$$(n+2)M_n = (2n+1)M_{n-1} + 3(n-1)M_{n-2}$$

counts unary binary trees.

Schröder

$$3(2n-1)S_n = (n+1)S_{n+1} + (n-2)S_{n-1}$$

counts binary trees in which every nonnull right link is colored either white or black.

Constructive proofs

Rémy

Catalan, Motzkin, Schröder

Numbers

Catalan

$$(n+1)C_n = 2(2n-1)C_{n-1}$$

counts binary trees.

Motzkin

$$(n+2)M_n = (2n+1)M_{n-1} + 3(n-1)M_{n-2}$$

counts unary binary trees.

Schröder

$$3(2n-1)S_n = (n+1)S_{n+1} + (n-2)S_{n-1}$$

counts binary trees in which every nonnull right link is colored either white or black.

Constructive proofs

Rémy

Dulucq and Penaud

Catalan, Motzkin, Schröder

Numbers

Catalan

$$(n+1)C_n = 2(2n-1)C_{n-1}$$

counts binary trees.

Motzkin

$$(n+2)M_n = (2n+1)M_{n-1} + 3(n-1)M_{n-2}$$

counts unary binary trees.

Schröder

$$3(2n-1)S_n = (n+1)S_{n+1} + (n-2)S_{n-1}$$

counts binary trees in which every nonnull right link is colored either white or black.

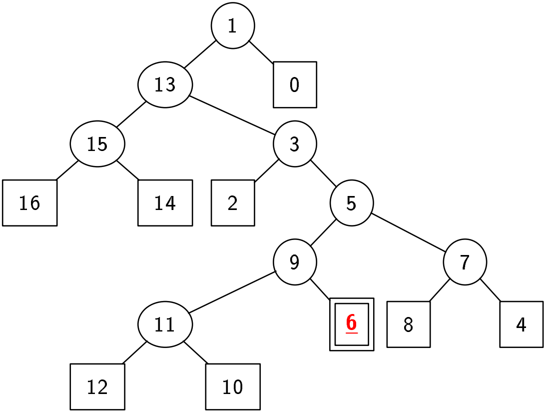
Constructive proofs

Rémy

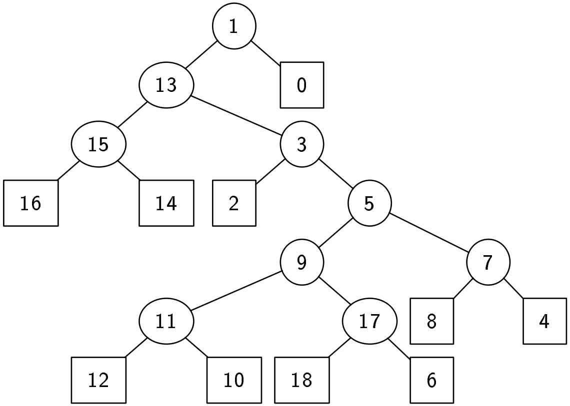
Dulucq and Penaud

Foata and Zeilberger

Rémy's construction



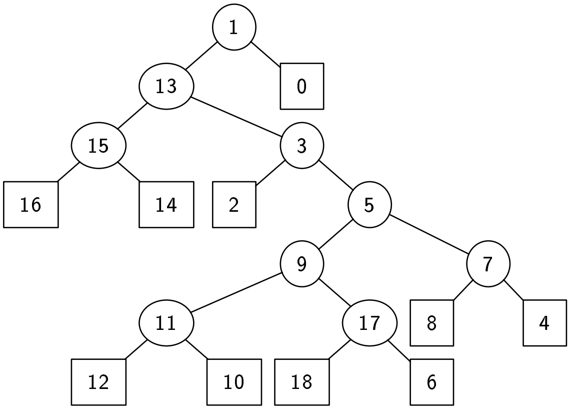
Implementation in an array of size $2n + 1$



- Nodes are labeled by odd numbers
- Leaves are labeled by even numbers

indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
values	1	13	0	2	5	9	7	8	4	11	17	12	10	15	3	16	14	18	6

Implementation in an array of size $2n+1$



- Nodes are labeled by odd numbers
- Leaves are labeled by even numbers
- root is index 0
- left child of node labeled by $2n+1$ is at index $2n+1$
- right child of node labeled by $2n+1$ is at index $2n+2$.

indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
values	1	13	0	2	5	9	7	8	4	11	17	12	10	15	3	16	14	18	6

Two differences

Motzkin

$$(n+2)M_n = (2n+1)M_{n-1} + 3(n-1)M_{n-2}$$

In the construction, an oracle chooses between two possibilities.

Two differences

Motzkin

$$(n+2)M_n = (2n+1)M_{n-1} + 3(n-1)M_{n-2}$$

In the construction, an oracle chooses between two possibilities.

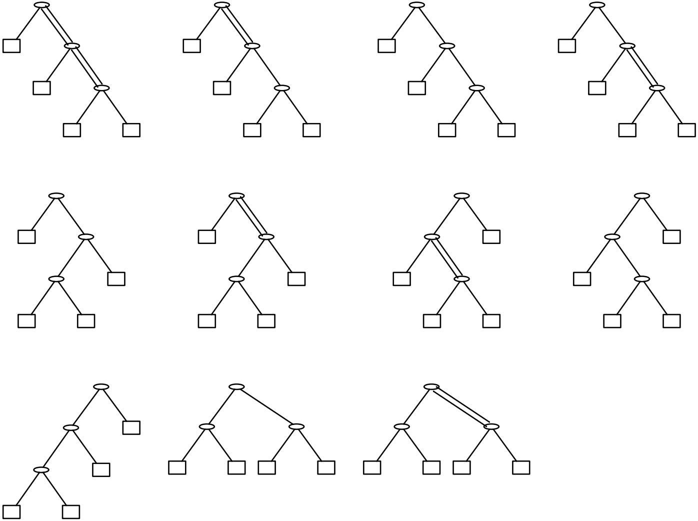
Schröder

$$(n+1)S_{n+1} = 3(2n-1)S_n - (n-2)S_{n-1}$$

In the construction,

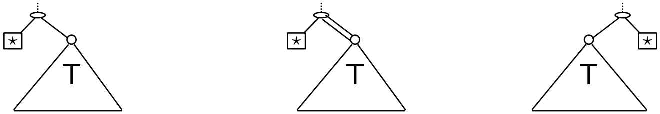
- one builds a Schröder tree of size $n+1$ from a tree of size n, or
- one fails.

Foata & Zeilberger construction for Schröder trees

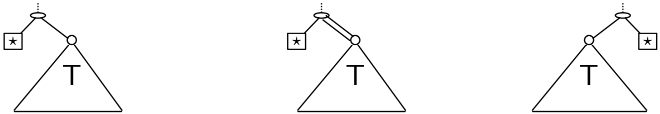


The 11 Schröder trees with 4 leaves.

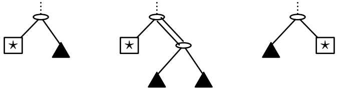
Insertion of a leaf in a Schröder tree



Insertion of a leaf in a Schröder tree

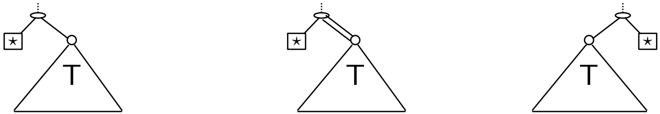


Three impossible insertions of leaves

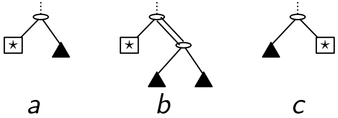


a *b* *c*

Insertion of a leaf in a Schröder tree



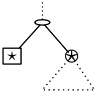
Three impossible insertions of leaves



Two unreachables

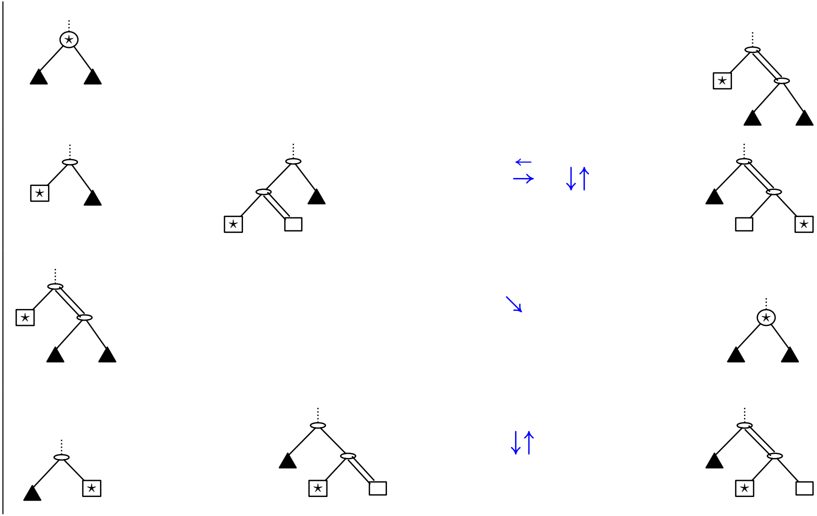
Foata-Zeilberger Isomophism (first case)

$\mathbf{L_1}$ 



Foata-Zeilberger Isomophism (second case)

$\mathbf{L_2}$



Foata-Zeilberger Isomophism (third case)

$\mathbf{R_1}$

The data structure

Like for Rémy's algorithm, one uses an array of size $2n+1$.

To represent colors of the links, one adds a boolean component.

- A Node is labeled by a pair of an odd number and a boolean.
- A Leaf is labeled by a pair of an even number and a boolean.

The boolean says that the right link that starts from this node is white.

Therefore when one considers a triple $(m,(k,b))$:

- m is an index (for a right link),
- (k,b) is located at m in the array (k corresponds to a node).

If $b \equiv \mathit{True}$ then m is even and k is odd.

The data structure

Like for Rémy's algorithm, one uses an array of size $2n + 1$.

To represent colors of the links, one adds a boolean component.

- A Node is labeled by a pair of an odd number and a boolean.
- A Leaf is labeled by a pair of an even number and a boolean.

The boolean says that the right link that starts from this node is white.

Therefore when one considers a triple $(m, (k, b))$:

- m is an index (for a right link),
- (k, b) is located at m in the array (k corresponds to a node).

If $b \equiv True$ then m is even and k is odd.

In other words, triples with $True$ are of the form $(2p, (2q + 1, True))$.

The data structure

Like for Rémy's algorithm, one uses an array of size $2n+1$.

To represent colors of the links, one adds a boolean component.

- A Node is labeled by a pair of an odd number and a boolean.
- A Leaf is labeled by a pair of an even number and a boolean.

The boolean says that the right link that starts from this node is white.

Therefore when one considers a triple $(m,(k,b))$:

- m is an index (for a right link),
- (k,b) is located at m in the array (k corresponds to a node).

If $b \equiv \mathit{True}$ then m is even and k is odd.

In other words, triples with True are of the form $(2p,(2q+1,\mathit{True}))$.

This must be checked when designing the algorithm.

The algorithm (1)

6 cases L_1, L_2 (with 4 subcases), R_1.

Draw a number x between 0 and $6n - 4$.

- L_1 if $x \textbf{ mod } 3 \equiv 0$
- L_2 if $x \textbf{ mod } 3 \equiv 1$
- R_1 if $x \textbf{ mod } 3 \equiv 2$.
- Let us call k the number $x \div 3$.

Assume the k^{th} is (h, b).

The algorithm (1)

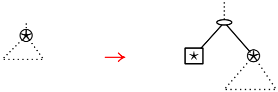
6 cases L_1, L_2 (with 4 subcases), R_1.

Draw a number x between 0 and $6n-4$.

- L_1 if x **mod** $3 \equiv 0$
- L_2 if x **mod** $3 \equiv 1$
- R_1 if x **mod** $3 \equiv 2$.
- Let us call k the number $x \div 3$.

Assume the k^{th} is (h, b).

- L_1 : $\left\{ \begin{array}{rcl} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & v[k] \end{array} \right.$



The algorithm (1)

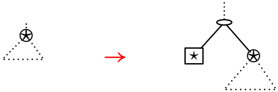
6 cases L_1, L_2 (with 4 subcases), R_1.

Draw a number x between 0 and $6n-4$.

- L_1 if x **mod** $3 \equiv 0$
- L_2 if x **mod** $3 \equiv 1$
- R_1 if x **mod** $3 \equiv 2$.
- Let us call k the number $x \div 3$.

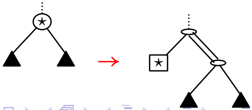
Assume the k^{th} is (h, b).

- L_1 : $\left\{\begin{array}{rcl} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & v[k] \end{array}\right.$



- L_2 **and** h **is odd** :

$$\left\{\begin{array}{rcl} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & (fst(v[k]), True) \end{array}\right.$$



The algorithm (1)

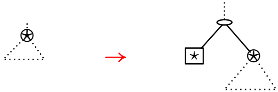
6 cases L_1, L_2 (with 4 subcases), R_1.

Draw a number x between 0 and $6n-4$.

- L_1 if x **mod** $3 \equiv 0$
- L_2 if x **mod** $3 \equiv 1$
- R_1 if x **mod** $3 \equiv 2$.
- Let us call k the number $x \div 3$.

Assume the k^{th} is (h, b).

• L_1 : $\left\{ \begin{array}{rcl} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & v[k] \end{array} \right.$



• L_2 **and** h **is odd** :

$$\left\{ \begin{array}{rcl} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & (fst(v[k]), True) \end{array} \right.$$

Check the constraint !

The algorithm (1)

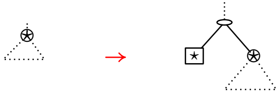
6 cases L_1, L_2 (with 4 subcases), R_1.

Draw a number x between 0 and $6n - 4$.

- L_1 if x **mod** $3 \equiv 0$
- L_2 if x **mod** $3 \equiv 1$
- R_1 if x **mod** $3 \equiv 2$.
- Let us call k the number $x \div 3$.

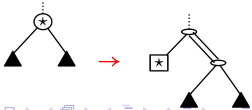
Assume the k^{th} is (h, b).

- L_1 : $\begin{cases} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & v[k] \end{cases}$



- L_2 **and** h **is odd** :

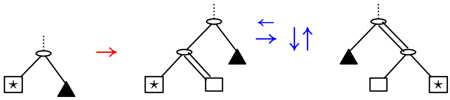
$$\begin{cases} v[k] & \leftarrow & (2n-1, False) \\ v[2n-1] & \leftarrow & (2n, False) \\ v[2n] & \leftarrow & (fst(v[k]), True) \end{cases}$$



The algorithm (2)

- L_2 **and** h **is even and** k **is odd and the second component of** $v[k+1]$ **is** *False*

$$\begin{cases} v[k] & \leftarrow & v[k+1] \\ v[k+1] & \leftarrow & (2n-1, True) \\ v[2n-1] & \leftarrow & v[k] \\ v[2n] & \leftarrow & (2n, False) \end{cases}$$



The algorithm (2)

• L_2 **and** h **is even and** k **is odd and the second component of** $v[k+1]$ **is** *False*

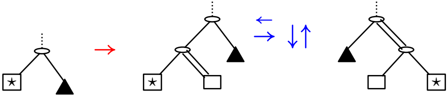
$$\begin{cases} v[k] & \leftarrow & v[k+1] \\ v[k+1] & \leftarrow & (2n-1, \mathit{True}) \\ v[2n-1] & \leftarrow & v[k] \\ v[2n] & \leftarrow & (2n, \mathit{False}) \end{cases}$$

Check the constraint !

The algorithm (2)

• L_2 and h is even and k is odd and
the second component of $v[k+1]$ is *False*

$$\begin{cases} v[k] & \leftarrow & v[k+1] \\ v[k+1] & \leftarrow & (2n-1, True) \\ v[2n-1] & \leftarrow & v[k] \\ v[2n] & \leftarrow & (2n, False) \end{cases}$$

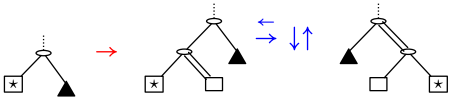


• L_2 and h is even and k is odd and
the second component of $v[k+1]$ is *True*

The algorithm (2)

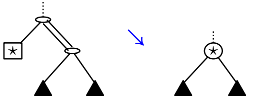
- L_2 and h is even and k is odd and the second component of $v[k+1]$ is *False*

$$\begin{cases} v[k] & \leftarrow & v[k+1] \\ v[k+1] & \leftarrow & (2n-1, \mathit{True}) \\ v[2n-1] & \leftarrow & v[k] \\ v[2n] & \leftarrow & (2n, \mathit{False}) \end{cases}$$



- L_2 and h is even and k is odd and the second component of $v[k+1]$ is *True*

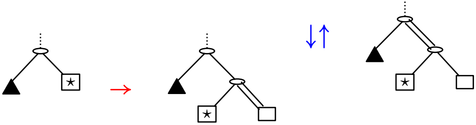
Failure **Retry**



The algorithm (3)

- L_2 **and** h **is even and** k **is even**

$$\left\{\begin{array}{rcl} v[k] & \rightarrow & (2n-1, \mathit{True}) \\ v[2n-1] & \rightarrow & v[k] \\ v[2n] & \rightarrow & (2n, \mathit{False}) \end{array}\right.$$



The algorithm (3)

- L_2 **and** h **is even and** k **is even**

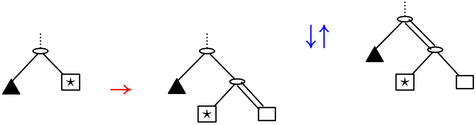
$$\begin{cases} v[k] & \rightarrow & (2n-1, True) \\ v[2n-1] & \rightarrow & v[k] \\ v[2n] & \rightarrow & (2n, False) \end{cases}$$

Check the constraint !

The algorithm (3)

- L_2 **and** h **is even and** k **is even**

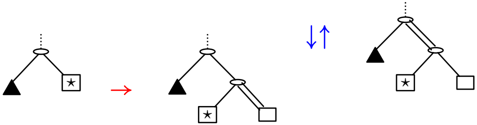
$$\begin{cases} v[k] & \rightarrow & (2n-1, \mathit{True}) \\ v[2n-1] & \rightarrow & v[k] \\ v[2n] & \rightarrow & (2n, \mathit{False}) \end{cases}$$



The algorithm (3)

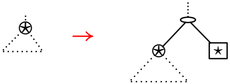
- L_2 **and** h **is even and** k **is even**

$$\begin{cases} v[k] & \rightarrow & (2n-1, True) \\ v[2n-1] & \rightarrow & v[k] \\ v[2n] & \rightarrow & (2n, False) \end{cases}$$



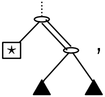
- R_1

$$\begin{cases} v[k] & \rightarrow & (2n-1, False) \\ v[2n-1] & \rightarrow & v[k] \\ v[2n] & \rightarrow & (2n, False) \end{cases}$$



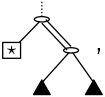
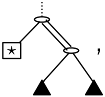
Complexity

The algorithm is quasi-linear.

- Except for  , all the cases require a computation in $O(1)$ to build a tree of size n from a tree of size $n-1$.

Complexity

The algorithm is quasi-linear.

- Except for  , all the cases require a computation in $O(1)$ to build a tree of size n from a tree of size $n-1$.
- In case  , one fails and retries with probability less that $\frac{1}{3}$ and the total average complexity of building a tree of size n is in $O(n)$.

Benchmarks

size	time	ratio
1 000	$0.012s$	0.024
5 000	$0.031s$	0.0288
10 000	$0.064s$	0.025
50 000	$0.200s$	0.0269
100 000	$0.290s$	0.02707
500 000	$1.295s$	0.027762
1 000 000	$3.065s$	0.027883
5 000 000	$15.183s$	0.0276378
10 000 000	$30.738s$	0.0275827

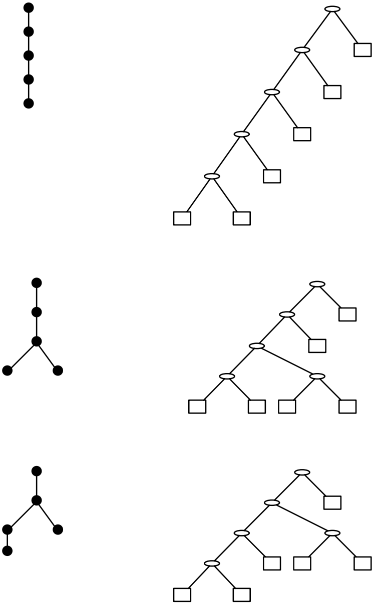
Thank you !

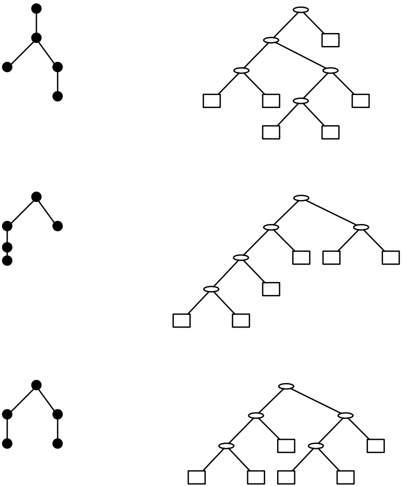
Thank you !

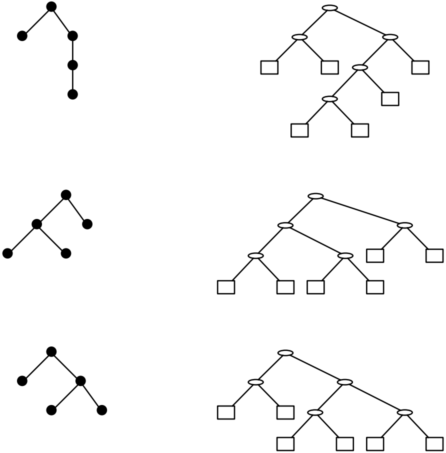
Any Question ?

The 9 Motzkin trees and the slanted binary trees

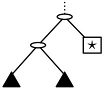
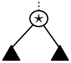
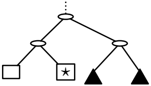
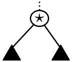
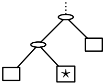
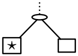
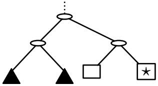
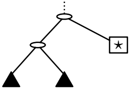
Instead of Motzkin trees, we consider slanted binary trees.

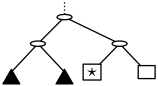
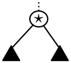
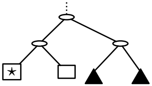
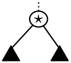
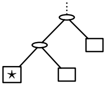
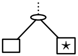






The 7 patterns of leaf-marked slanting trees

	leaf-marked slanting trees	marked slanting trees	Choice and node-marked slanting trees
1.			
2.			LR , 
3.			
4.			RR , 

5.			**RL** , 
6.			**LL** , 
7.			

The key choice

In the algorithm there are two cases

1. Building a tree of size n from a tree of size $n-1$,
2. Building a tree of size n from a tree of size $n-2$,

The key choice

In the algorithm there are two cases

1. Building a tree of size n from a tree of size $n-1$,
2. Building a tree of size n from a tree of size $n-2$,

Assume we draw a number between 0 and 1, and

- if $c \leq \frac{(2n+1)Mn-1}{(n+2)M_n}$, we choose case1,
- if $c > \frac{(2n+1)Mn-1}{(n+2)M_n}$, we choose case2.

For an implementation with no recursion

For an imlementation with a while loop, I proceed as follows :

1. I create the stack of recursive calls,

For an implementation with no recursion

For an imlementation with a while loop, I proceed as follows :

1. I create the stack of recursive calls,
2. I pop the stack, building the Motzkin trees from the small ones to the large ones.

I can build a random Motzkin tree of size 10 millions in 45s.