A bijection of plane increasing trees with bounded relaxed binary trees

ALEA Days 03/2018

Michael Wallner

Erwin Schrödinger-Fellow (Austrian Science Fund (FWF): J4162)

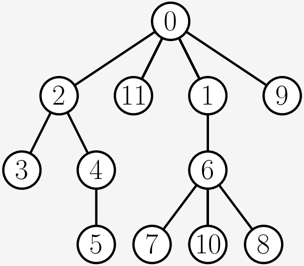
Laboratoire Bordelais de Recherche en Informatique, Université de Bordeaux, France

March 12th, 2018

Based on the paper:
A bijection of plane increasing trees with
relaxed binary trees of right height at most one.
ArXiv:1706.07163

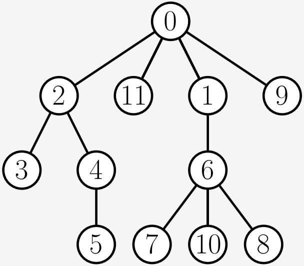
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



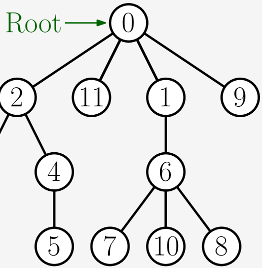
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \ldots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



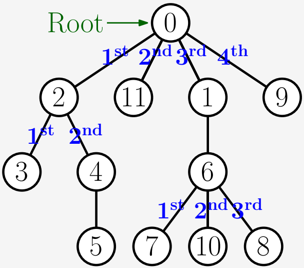
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \ldots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



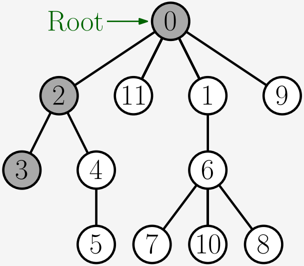
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \ldots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



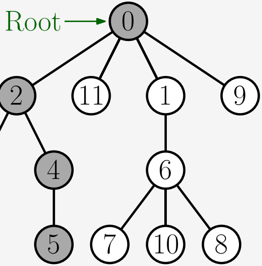
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \ldots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



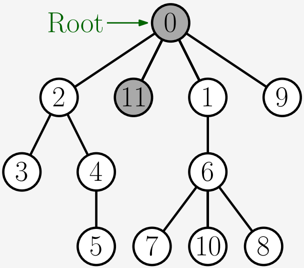
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



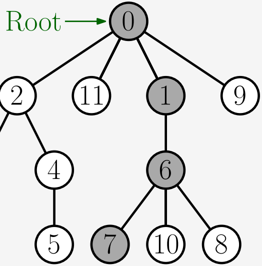
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



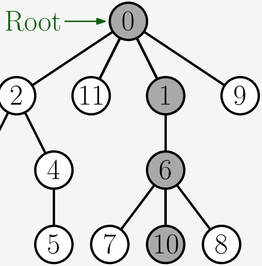
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



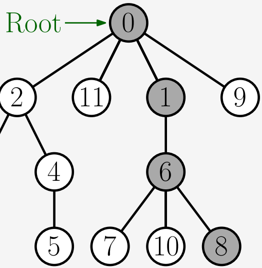
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



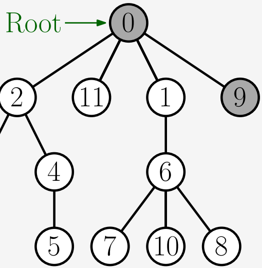
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



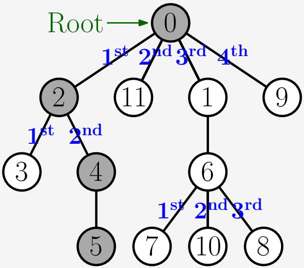
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \ldots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



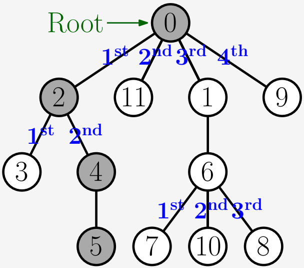
Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



Rooted plane increasing trees

- *Labeled*: Nodes get labels $0, \dots, n$
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node with label 0
- *Plane*: Children are equipped with a left-to-right order
- *Increasing*: Labels along any path from the root are increasing



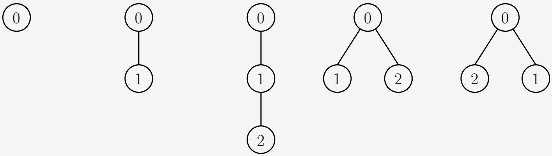


Figure: All rooted plane increasing trees of size $0, 1$, and 2.

A growth process

- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation

A growth process

- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation

A growth process

- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation

A growth process

- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation

A growth process

- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation

A growth process

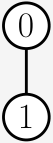
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

⇒ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

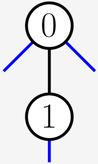
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

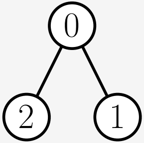
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

⇒ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

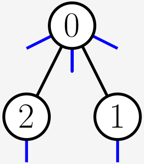
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

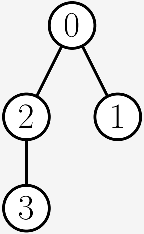
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation

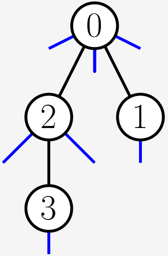


A growth process

- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are
$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$
rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

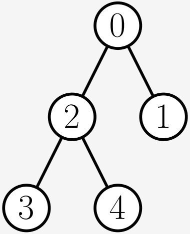
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

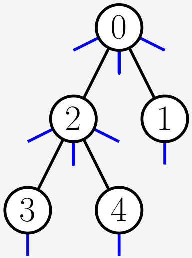
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

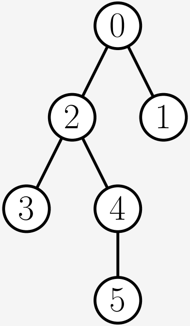
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

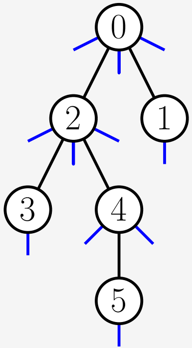
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

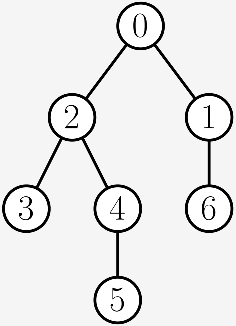
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1) \cdot (2n-3) \cdots 3 \cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

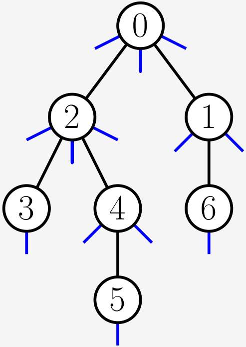
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

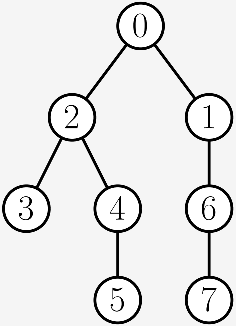
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

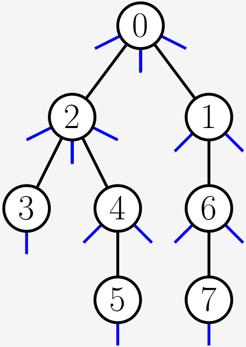
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

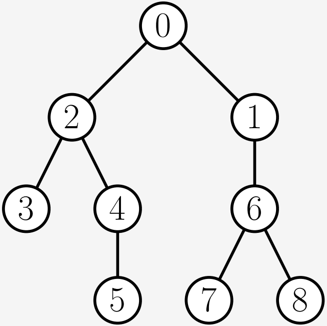
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

⇒ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

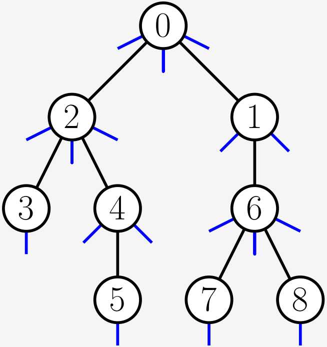
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

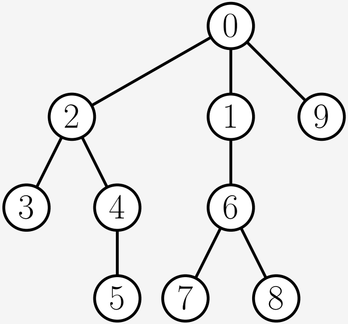
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1) \cdot (2n-3) \cdots 3 \cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

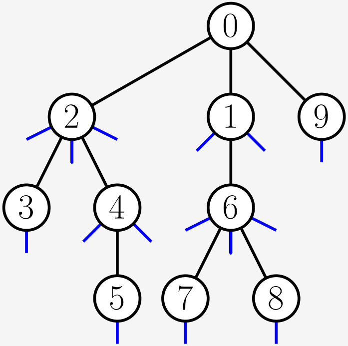
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

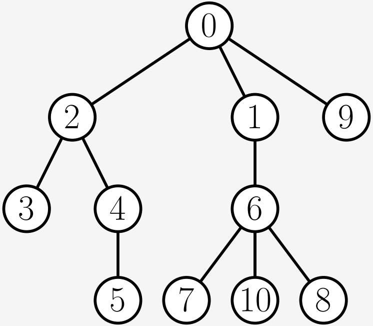
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

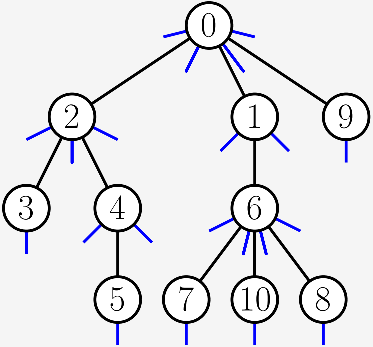
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

$\Rightarrow$ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



A growth process

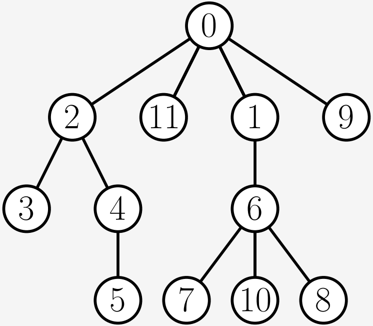
- Start with a root and label 0
- After $i-1$ steps there are $2i-1$ possible steps to insert node i

⇒ There are

$$(2n-1)!! = (2n-1)\cdot(2n-3)\cdots 3\cdot 1$$

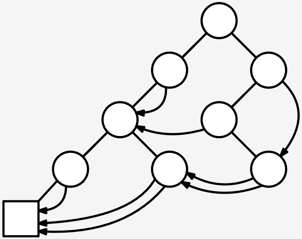
rooted plane increasing trees

- Gives a linear time algorithm for uniform random generation



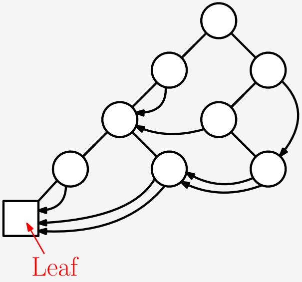
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



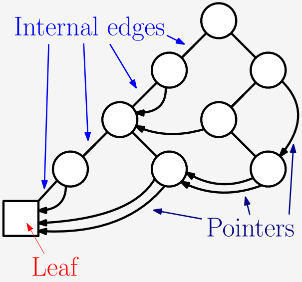
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



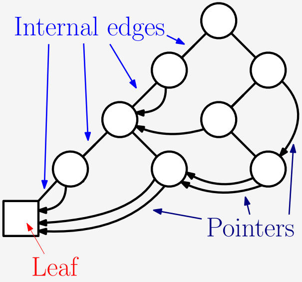
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



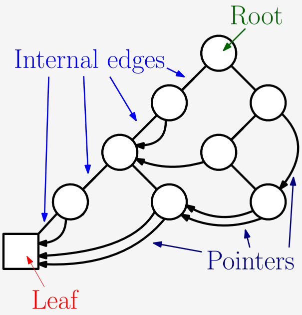
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



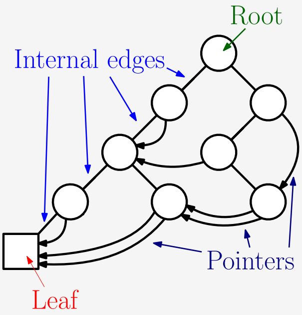
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



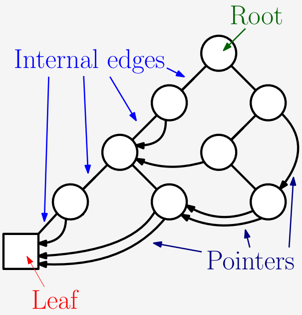
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



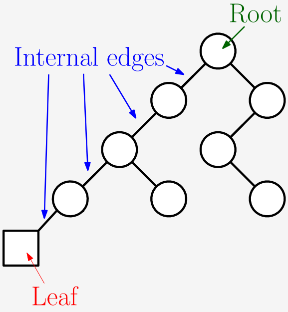
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



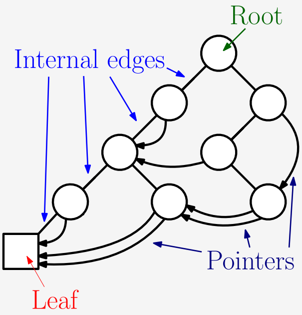
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



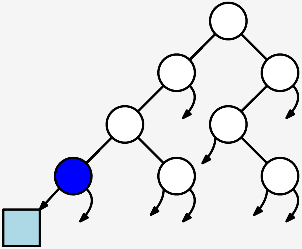
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



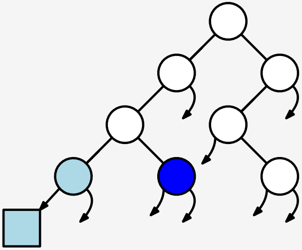
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



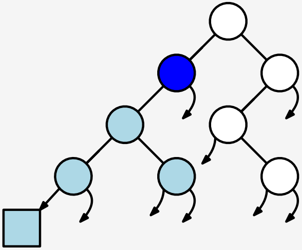
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



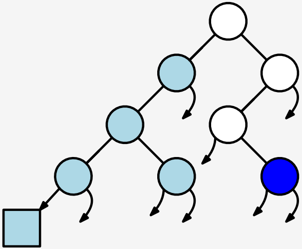
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



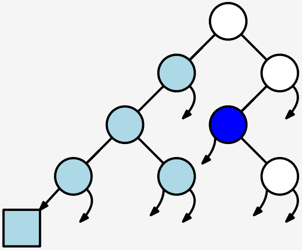
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



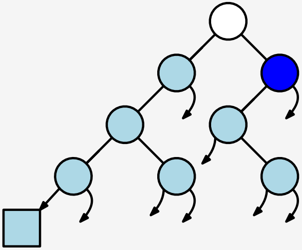
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



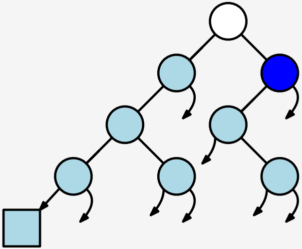
Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



Relaxed binary trees

- Directed acyclic graph (DAG)
- *Nodes*: n (internal) nodes and 1 leaf
- *Edges*: n internal edges and n pointers
- *Size*: n (nodes minus one)
- *Rooted*: Unique distinguished node
- *Plane*: Children are equipped with a left-to-right order
- *Structure*: Deleting the pointers gives a plane (binary) tree
- *Pointers*: Point to a node previously visited in **postorder**



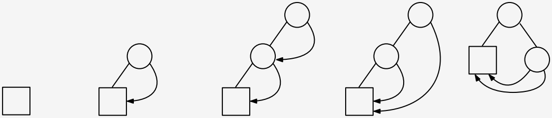


Figure: All relaxed binary trees of size $0, 1$, and 2.

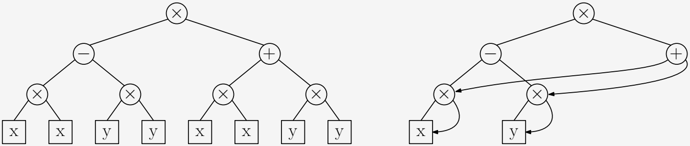
Why “relaxed”?

Compacted trees

- Trees are widely used data structures
- Contain often a lot of redundant information

⇒ **Save every distinct subtree only once and mark repeated occurrences**

- Applications: XML, compilers, computer algebra



Why "relaxed"?

Compacted trees

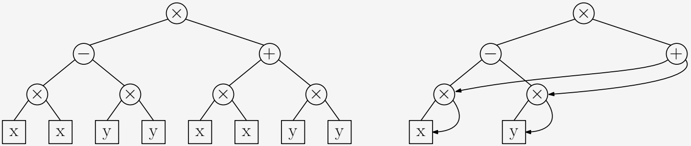
- Trees are widely used data structures
- Contain often a lot of redundant information

$\Rightarrow$ **Save every distinct subtree only once and mark repeated occurrences**

- Applications: XML, compilers, computer algebra

Important

- Subtrees are unique
- Bijection!



Why "relaxed"?

Compacted trees

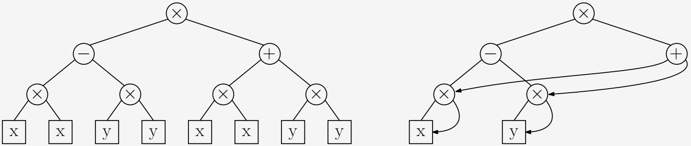
- Trees are widely used data structures
- Contain often a lot of redundant information

$\Rightarrow$ **Save every distinct subtree only once and mark repeated occurrences**

- Applications: XML, compilers, computer algebra

Important

- Subtrees are unique
- Bijection!



Relaxed (compacted) trees

- Drop uniqueness
- No bijection anymore

Bounded right height

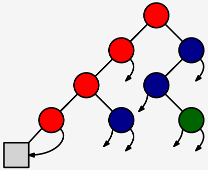
Right height

The maximal number of **right children on any path from the root to a leaf**.

Bounded right height

Right height

The maximal number of **right children on any path from the root to a leaf**.

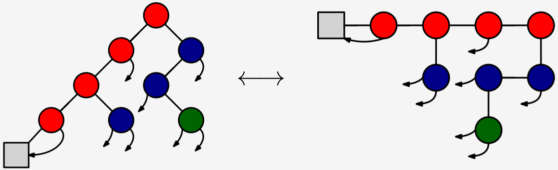


A binary tree with right height 2. Nodes of level 0 are colored in red, nodes of level 1 in blue, and the node of level 2 in green.

Bounded right height

Right height

The maximal number of **right children on any path from the root to a leaf**.



A binary tree with right height 2. Nodes of level 0 are colored in red, nodes of level 1 in blue, and the node of level 2 in green.

Relaxed trees of right height $\leq k$

Figure: Right height ≤ 0.

Relaxed trees of right height $\leq k$

Figure: Right height ≤ 0.

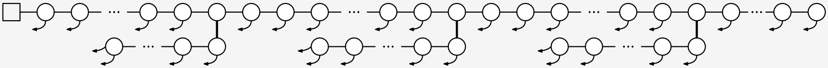


Figure: Right height ≤ 1.

Relaxed trees of right height $\leq k$

Figure: Right height ≤ 0.

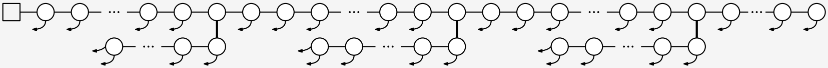


Figure: Right height ≤ 1.

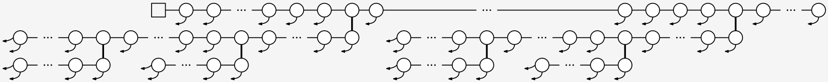


Figure: Right height ≤ 2.

Relaxed trees of right height $\leq k$

Figure: Right height ≤ 0.

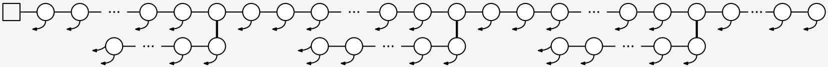


Figure: Right height ≤ 1.

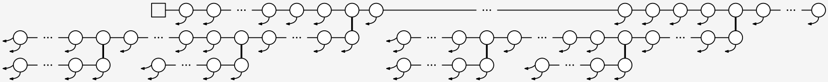


Figure: Right height ≤ 2.

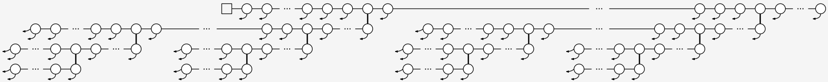


Figure: Right height ≤ 3.

Relaxed trees of right height $\leq k$

Asymptotic number of relaxed and compacted binary trees with right height $\leq$ k of size $n \to \infty$

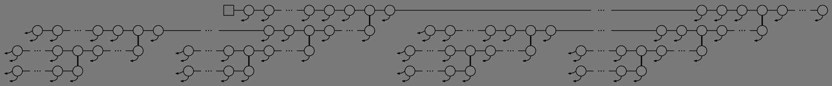


Figure: Right height ≤ 3.

Relaxed trees of right height $\leq k$

Asymptotic number of relaxed and compacted binary trees with right height $\leq$ k of size $n \to \infty$

$$\#\{\text{Relaxed}\} \sim \gamma_k n! \left(4\cos\left(\frac{\pi}{k+3}\right)^2\right)^n n^{-k/2}$$

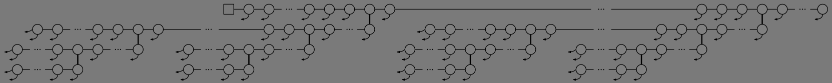


Figure: Right height ≤ 3.

Relaxed trees of right height $\leq k$

Asymptotic number of relaxed and compacted binary trees with right height $\leq$ k of size $n \to \infty$

$$\#\{\text{Relaxed}\} \sim \gamma_k n! \left(4\cos\left(\frac{\pi}{k+3}\right)^2\right)^n n^{-k/2}$$

$$\#\{\text{Compacted}\} \sim \kappa_k n! \left(4\cos\left(\frac{\pi}{k+3}\right)^2\right)^n n^{-\frac{k}{2}-\frac{1}{k+3}-\left(\frac{1}{4}-\frac{1}{k+3}\right)\cos\left(\frac{\pi}{k+3}\right)^{-2}}$$

Companion paper together with A. Genitrini, B. Gittenberger, and M. Kauers: *Asymptotic Enumeration of Compacted Binary Trees*, ArXiv:1703.10031.

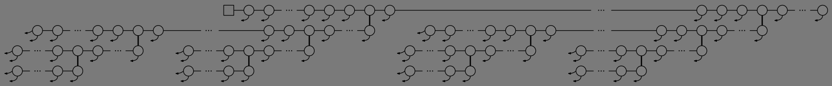
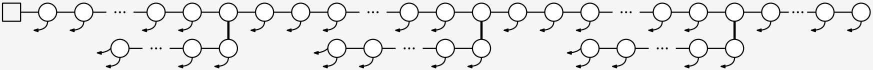


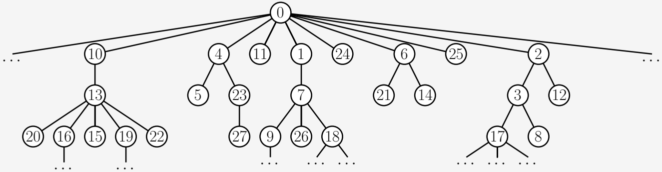
Figure: Right height $\leq$ 3.

The goal of this talk

Main result

There exists a bijection between relaxed binary trees of right height at most one and rooted plane increasing trees constructable as a linear time algorithm.

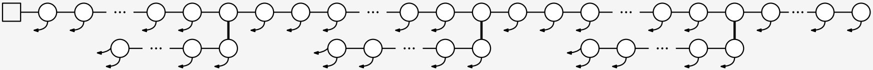


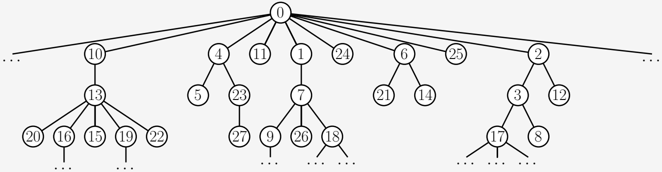


The goal of this talk

Main result

There exists a bijection between relaxed binary trees of right height at most one and rooted plane increasing trees constructable as a linear time algorithm.



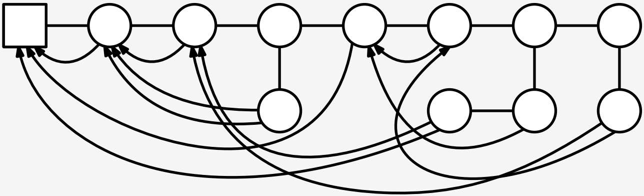


Corollary

Relaxed binary trees of right height at most one can be sampled uniformly at random in linear time.

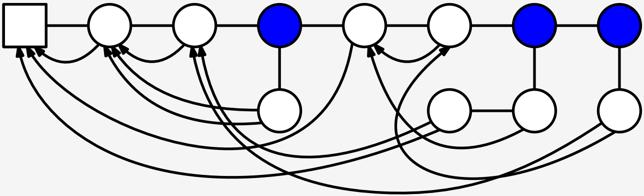
Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.



Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

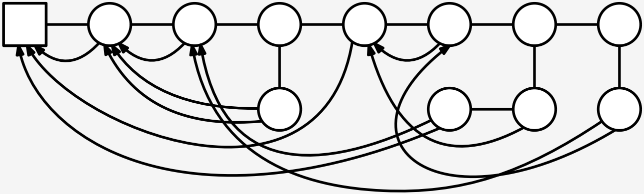
A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.



Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

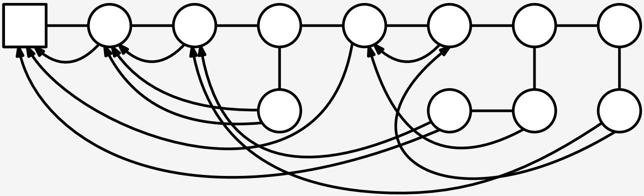


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

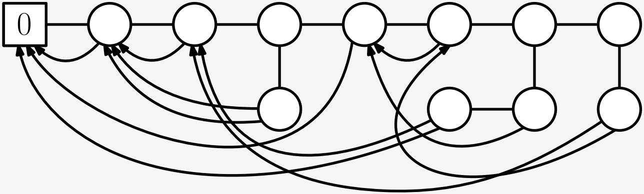


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

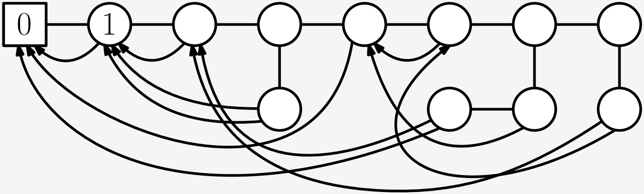


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

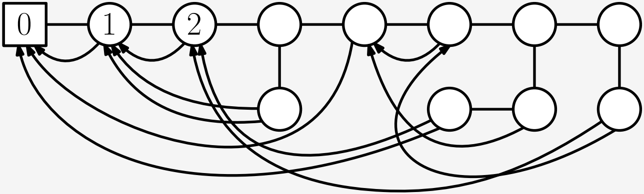


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

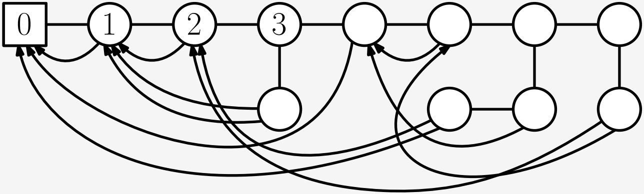


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

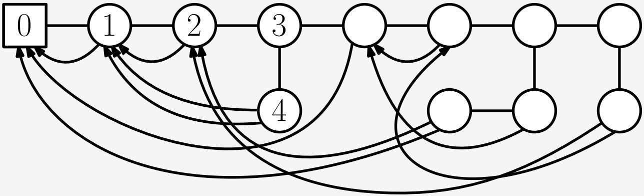


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \dots, v_n$

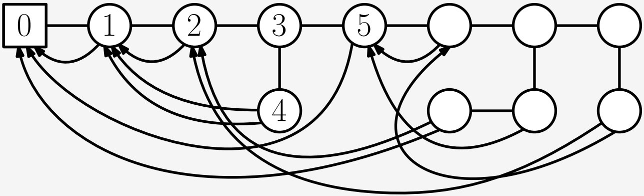


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

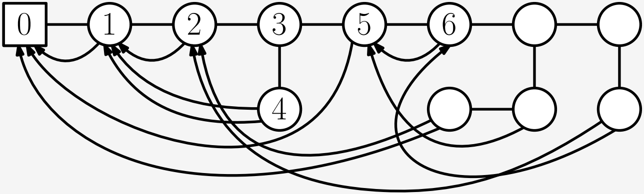


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

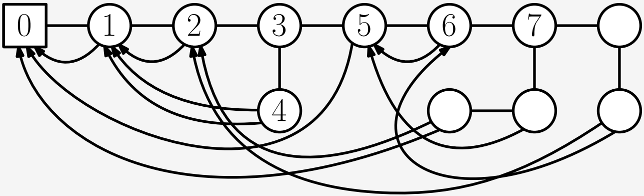


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \dots, v_n$

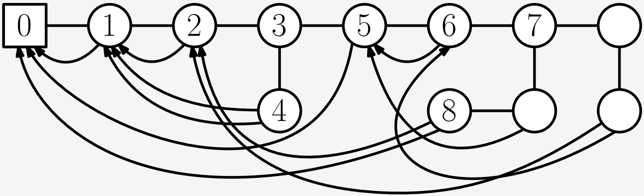


Relaxed binary tree $\mathcal{R}$ → Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

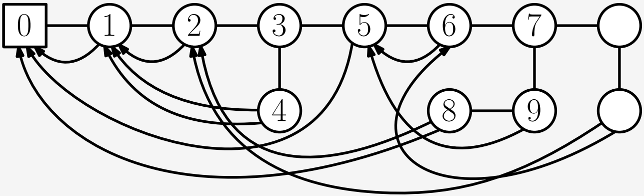


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \dots, v_n$

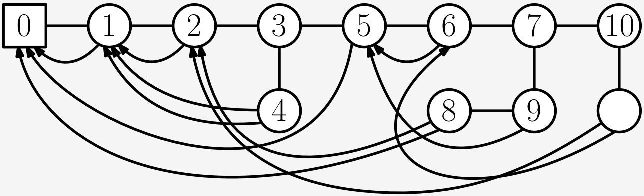


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \dots, v_n$

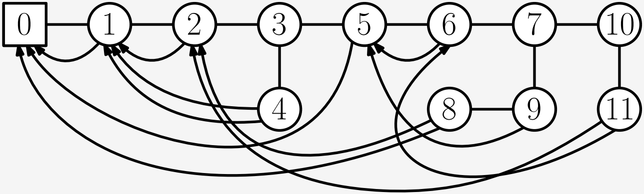


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$



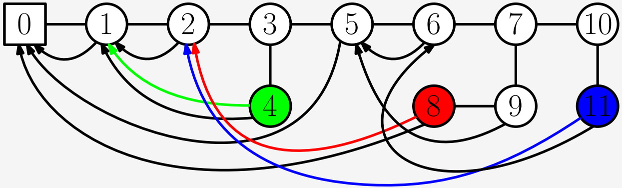
Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

2: For each cherry v_i move left pointer to v_{i-1} $\triangleright$ v_{i-1} is a branch node



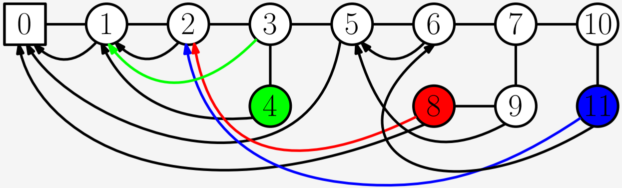
Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

2: For each cherry v_i move left pointer to v_{i-1} $\triangleright$ v_{i-1} is a branch node



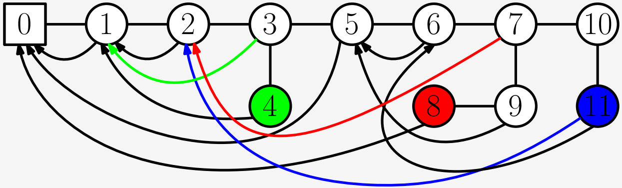
Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

2: For each cherry v_i move left pointer to v_{i-1} $\triangleright$ v_{i-1} is a branch node



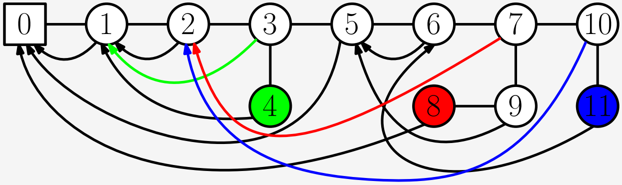
Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$

2: For each cherry v_i move left pointer to v_{i-1} $\qquad \triangleright$ v_{i-1} is a branch node

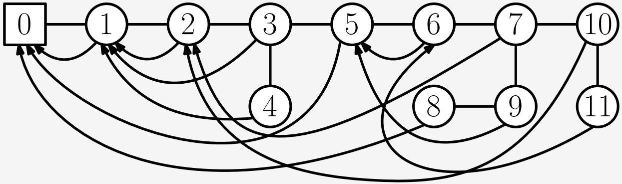


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$
2: For each cherry v_i move left pointer to v_{i-1} $\triangleright$ v_{i-1} is a branch node
3: For each node set $p_i :=$ target of pointer of v_i

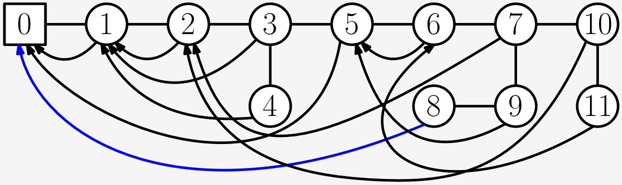


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \ldots, v_n$
2: For each cherry v_i move left pointer to v_{i-1} $\qquad \triangleright$ v_{i-1} is a branch node
3: For each node set $p_i :=$ target of pointer of v_i
4: **if** level$(v_i) = 1$ and $p_i = v_0$ **then**
5: $\quad p(v_i) :=$ Branch node of branch of v_i
6: **end if**

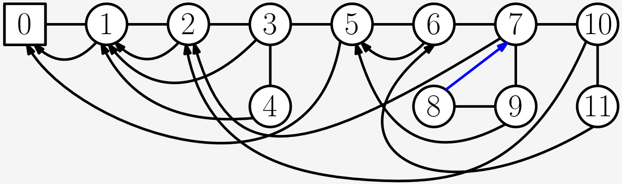


Relaxed binary tree $\mathcal{R}$ → Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

Setup

1: Label nodes of $\mathcal{R}$ in-order $v_0, v_1, \dots, v_n$
2: For each cherry v_i move left pointer to v_{i-1} ▷ v_{i-1} is a branch node
3: For each node set $p_i :=$ target of pointer of v_i
4: **if** level$(v_i) = 1$ and $p_i = v_0$ **then**
5: $\quad p(v_i) :=$ Branch node of branch of v_i
6: **end if**

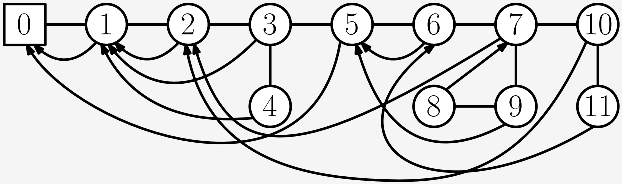


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

A *branch node* is a node on level 0 without pointers to which a branch of nodes on level 1 is attached.

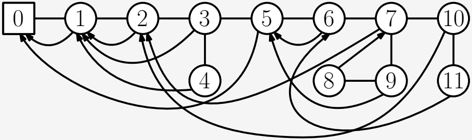
Setup

```
Label nodes of R in-order v_0, v_1, ..., v_n
For each cherry v_i move left pointer to v_{i-1}          ▷ v_{i-1} is a branch node
For each node set p_i := target of pointer of v_i
if  level(v_i) = 1 and p_i = v_0 then
    p(v_i) := Branch node of branch of v_i
end if
```



Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

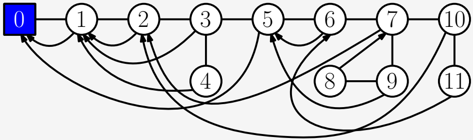
Transformation



Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

1: Leaf $v_0 \to$ Root of $\mathcal{T}$

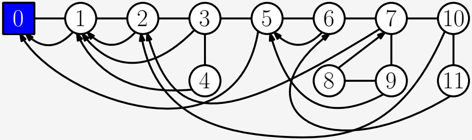


Relaxed binary tree $\mathcal{R}$ → Plane increasing tree $\mathcal{T}$

Transformation

Leaf $v_0 \to$ Root of $\mathcal{T}$
for i from 1 to n **do**

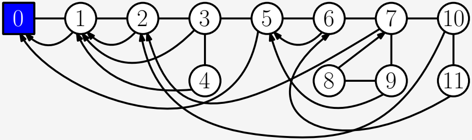
end for



Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

Transformation

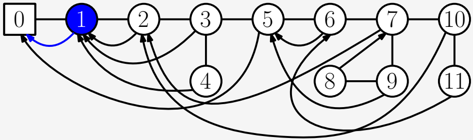
Leaf $v_0 \rightarrow$ Root of $\mathcal{T}$
for i from 1 to n **do**
  if $\text{level}(v_i) = 0$ **then** ▷ Parent-pointer
    Attach v_i as first child to p_i
  else ▷ Sibling-pointer
    Attach v_i as sibling right of p_i
  end if
end for

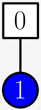


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

Transformation

```
Leaf v_0 → Root of T
for i from 1 to n do
    if level(v_i) = 0 then                    ▷ Parent-pointer
        Attach v_i as first child to p_i
    else                                      ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for
```

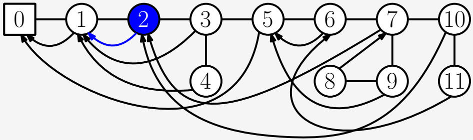




Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

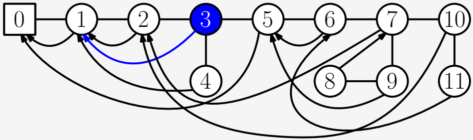
Leaf $v_0 \to$ Root of $\mathcal{T}$
for i from 1 to n **do**
    if level$(v_i) = 0$ **then** ▷ Parent-pointer
        Attach v_i as first child to p_i
    else ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for

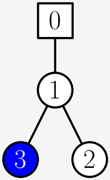


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

Leaf $v_0 \to$ Root of $\mathcal{T}$
for i from 1 to n **do**
  if level$(v_i) = 0$ **then** ▷ Parent-pointer
    Attach v_i as first child to p_i
  else ▷ Sibling-pointer
    Attach v_i as sibling right of p_i
  end if
end for

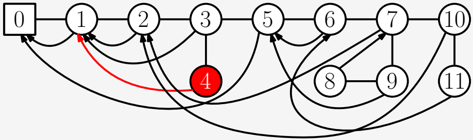


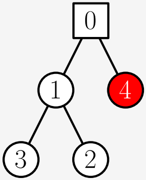


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

Transformation

```
Leaf v_0 → Root of T
for i from 1 to n do
    if level(v_i) = 0 then                          ▷ Parent-pointer
        Attach v_i as first child to p_i
    else                                            ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for
```

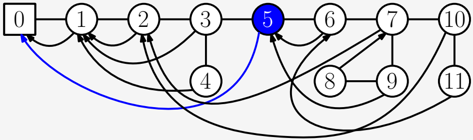


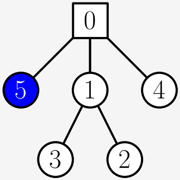


Relaxed binary tree $\mathcal{R}$ → Plane increasing tree $\mathcal{T}$

Transformation

```
1: Leaf v_0 → Root of T
2: for i from 1 to n do
3:     if level(v_i) = 0 then              ▷ Parent-pointer
4:         Attach v_i as first child to p_i
5:     else                                ▷ Sibling-pointer
6:         Attach v_i as sibling right of p_i
7:     end if
8: end for
```

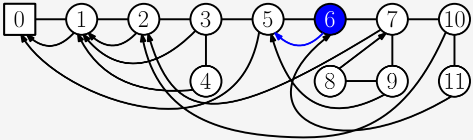


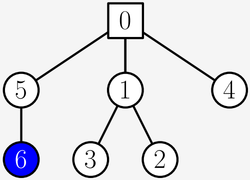


Relaxed binary tree $\mathcal{R}$ → Plane increasing tree $\mathcal{T}$

Transformation

1: Leaf $v_0 \to$ Root of $\mathcal{T}$
2: **for** i from 1 to n **do**
3:     **if** level(v_i) = 0 **then** ▷ Parent-pointer
4:         Attach v_i as first child to p_i
5:     **else** ▷ Sibling-pointer
6:         Attach v_i as sibling right of p_i
7:     **end if**
8: **end for**

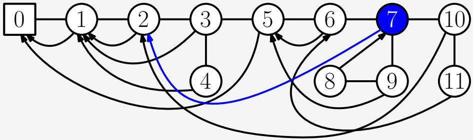


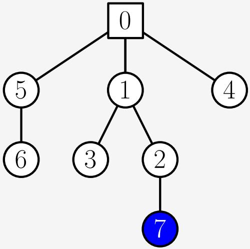


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

```
Leaf v_0 → Root of T
for i from 1 to n do
    if level(v_i) = 0 then                      ▷ Parent-pointer
        Attach v_i as first child to p_i
    else                                        ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for
```

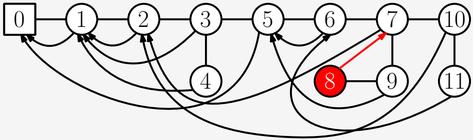


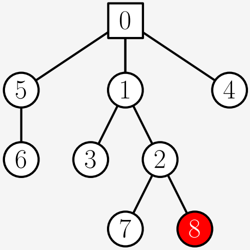


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

1: Leaf $v_0 \to$ Root of $\mathcal{T}$
2: **for** i from 1 to n **do**
3: **if** level$(v_i) = 0$ **then** ▷ Parent-pointer
4: Attach v_i as first child to p_i
5: **else** ▷ Sibling-pointer
6: Attach v_i as sibling right of p_i
7: **end if**
8: **end for**

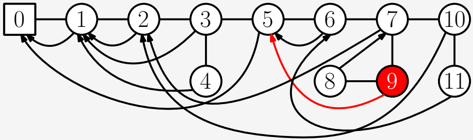


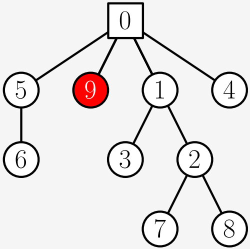


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

Leaf $v_0 \to$ Root of $\mathcal{T}$
for i from 1 to n **do**
    if $\text{level}(v_i) = 0$ **then** ▷ Parent-pointer
        Attach v_i as first child to p_i
    else ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for

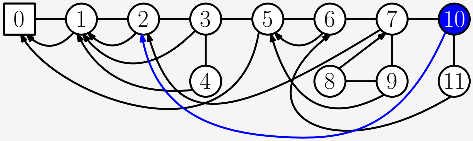


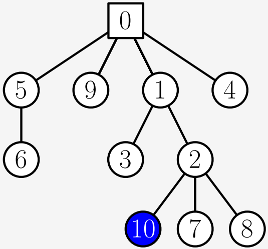


Relaxed binary tree $\mathcal{R} \rightarrow$ Plane increasing tree $\mathcal{T}$

Transformation

Leaf $v_0 \rightarrow$ Root of $\mathcal{T}$
for i from 1 to n **do**
    if level$(v_i) = 0$ **then** ▷ Parent-pointer
        Attach v_i as first child to p_i
    else ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for

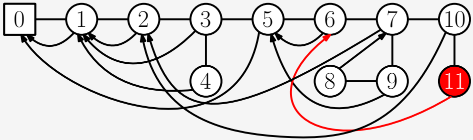


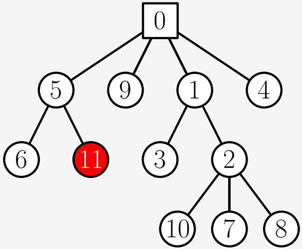


Relaxed binary tree $\mathcal{R} \to$ Plane increasing tree $\mathcal{T}$

Transformation

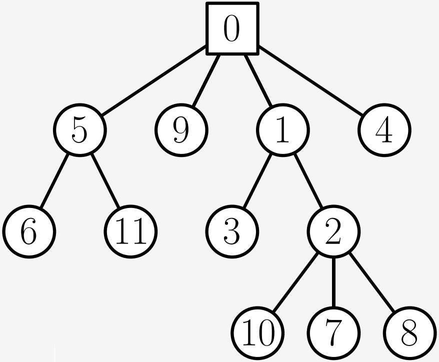
Leaf $v_0 \to$ Root of $\mathcal{T}$
for i from 1 to n **do**
    if $\text{level}(v_i) = 0$ **then** ▷ Parent-pointer
        Attach v_i as first child to p_i
    else ▷ Sibling-pointer
        Attach v_i as sibling right of p_i
    end if
end for





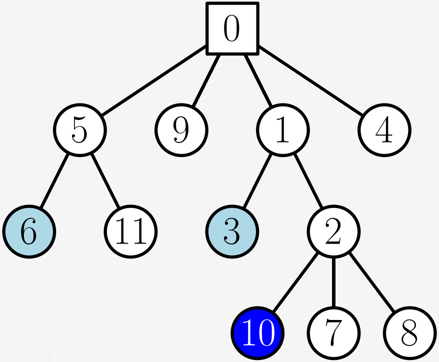
Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

- A *young leaf* is a leaf without left sibling.
- A *maximal young leaf* is a young leaf with maximal label.
- $\mathcal{T}_k$ is the tree restricted to the labels $0, \ldots, k$.

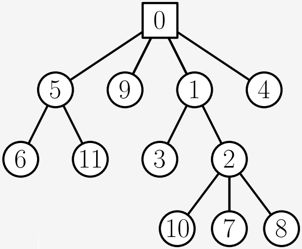


Reversed: Plane increasing tree $\mathcal{T} \rightarrow$ Relaxed binary tree $\mathcal{R}$

- A *young leaf* is a leaf without left sibling.
- A *maximal young leaf* is a young leaf with maximal label.
- $\mathcal{T}_k$ is the tree restricted to the labels $0, \dots, k$.



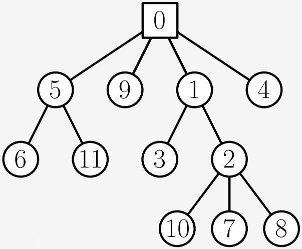
Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$



Reversed: Plane increasing tree $\mathcal{T} \rightarrow$ Relaxed binary tree $\mathcal{R}$

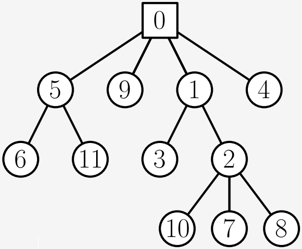
$\mathcal{B} := \emptyset$
for k from 0 to n **do**

end for



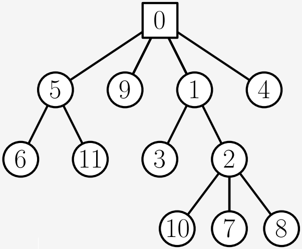
Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

```
B := ∅
for k from 0 to n do
    if v_k is a maximal young leaf in T_k then
        Attach v_k as new root with a pointer to the parent of v_k in T_k
        Attach B to previous root and move its pointer to last node of B on the left
        B := ∅
end for
```



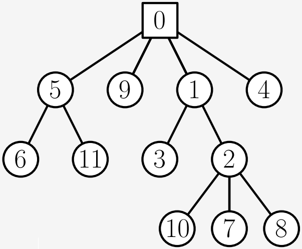
Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

$\mathcal{B} := \emptyset$
for k from 0 to n **do**
  if v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
    Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
    Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
    $\mathcal{B} := \emptyset$
  else
    Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
  end if
end for



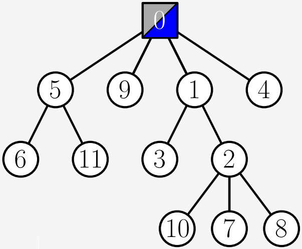
Reversed: Plane increasing tree $\mathcal{T} \rightarrow$ Relaxed binary tree $\mathcal{R}$

1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5



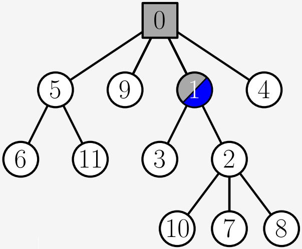
Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3: **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4: Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5: Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6: $\mathcal{B} := \emptyset$
7: **else**
8: Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9: **end if**
10: **end for**
11: Perform 5



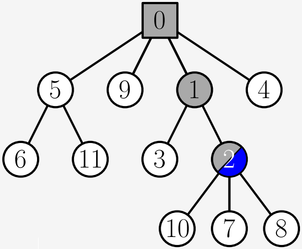
Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

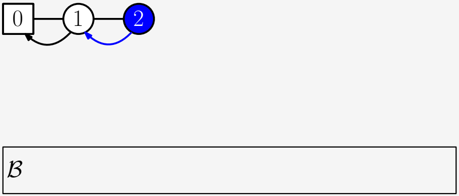
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3: **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4: Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5: Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6: $\mathcal{B} := \emptyset$
7: **else**
8: Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9: **end if**
10: **end for**
11: Perform 5



Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

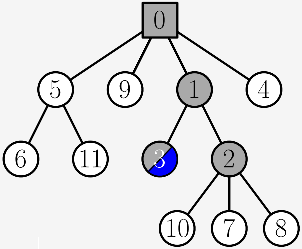
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

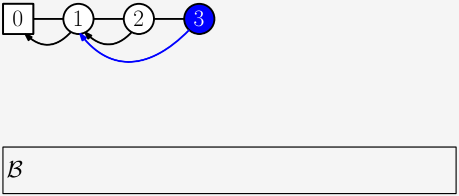




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

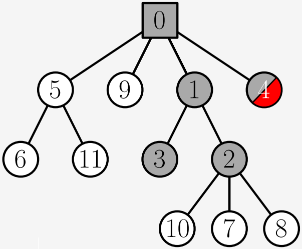
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

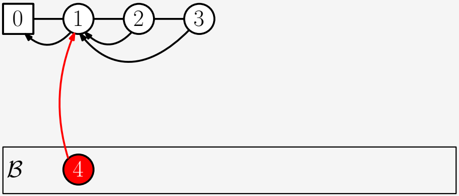




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

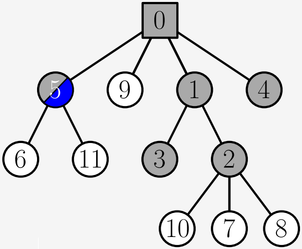
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

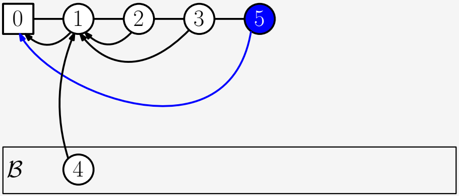




Reversed: Plane increasing tree $\mathcal{T} \rightarrow$ Relaxed binary tree $\mathcal{R}$

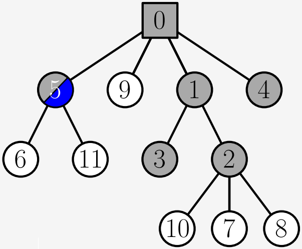
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

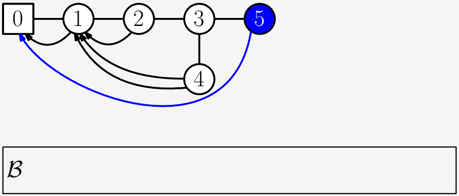




Reversed: Plane increasing tree $\mathcal{T} \rightarrow$ Relaxed binary tree $\mathcal{R}$

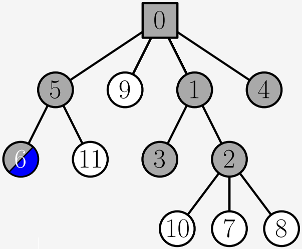
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

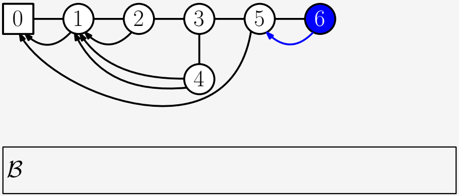




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

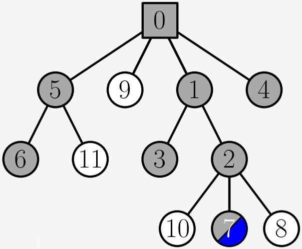
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

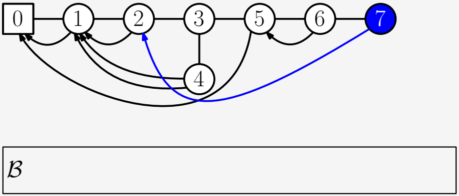




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

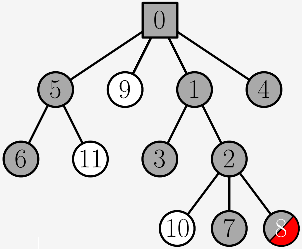
```
1: 𝓑 := ∅
2: for k from 0 to n do
3:     if v_k is a maximal young leaf in 𝓣_k then
4:         Attach v_k as new root with a pointer to the parent of v_k in 𝓣_k
5:         Attach 𝓑 to previous root and move its pointer to last node of 𝓑 on the left
6:         𝓑 := ∅
7:     else
8:         Attach v_k as root to 𝓑 with a pointer to the left sibling of v_k in 𝓣_k
9:     end if
10: end for
11: Perform 5
```

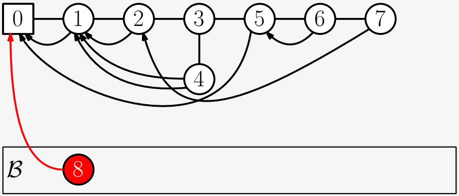




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

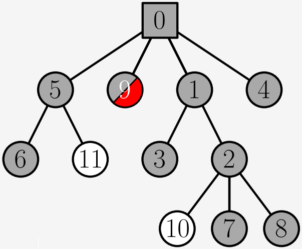
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

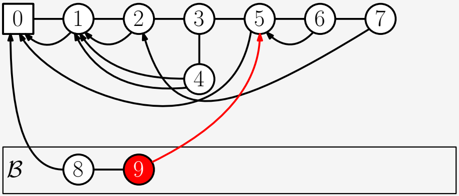




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

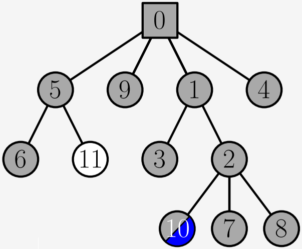
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3: **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4: Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5: Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6: $\mathcal{B} := \emptyset$
7: **else**
8: Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9: **end if**
10: **end for**
11: Perform 5

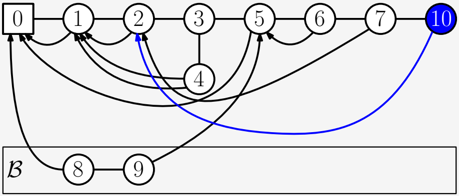




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

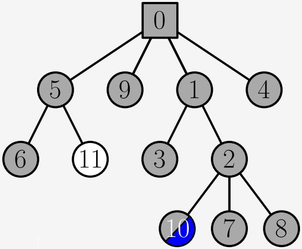
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

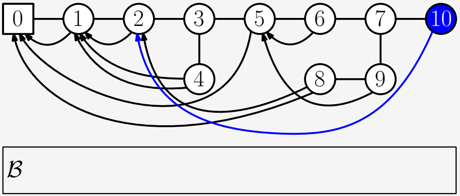




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

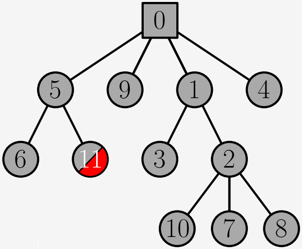
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

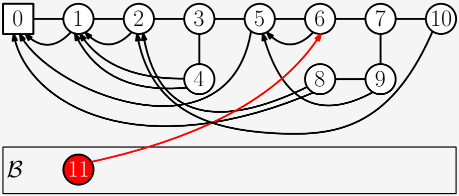




Reversed: Plane increasing tree $\mathcal{T} \rightarrow$ Relaxed binary tree $\mathcal{R}$

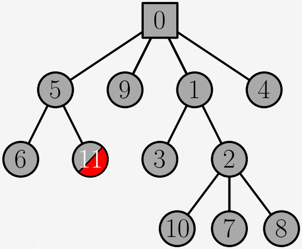
1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5

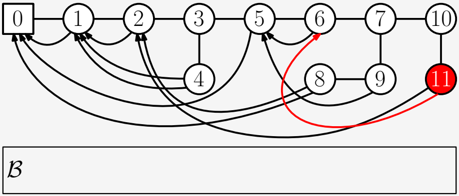




Reversed: Plane increasing tree $\mathcal{T} \to$ Relaxed binary tree $\mathcal{R}$

1: $\mathcal{B} := \emptyset$
2: **for** k from 0 to n **do**
3:     **if** v_k is a maximal young leaf in $\mathcal{T}_k$ **then**
4:         Attach v_k as new root with a pointer to the parent of v_k in $\mathcal{T}_k$
5:         Attach $\mathcal{B}$ to previous root and move its pointer to last node of $\mathcal{B}$ on the left
6:         $\mathcal{B} := \emptyset$
7:     **else**
8:         Attach v_k as root to $\mathcal{B}$ with a pointer to the left sibling of v_k in $\mathcal{T}_k$
9:     **end if**
10: **end for**
11: Perform 5





Number of maximal young leaves

Correspondence

- Maximal young leaves in the growth process in rooted plane increasing trees
- Nodes in level 0 in relaxed binary trees of right height ≤ 1

Number of maximal young leaves

Correspondence

- Maximal young leaves in the growth process in rooted plane increasing trees
- Nodes in level 0 in relaxed binary trees of right height ≤ 1

Let X_n be its random variable when drawn uniformly at random among all such trees of size n.

Number of maximal young leaves

Correspondence

- Maximal young leaves in the growth process in rooted plane increasing trees
- Nodes in level 0 in relaxed binary trees of right height ≤ 1

Let X_n be its random variable when drawn uniformly at random among all such trees of size n.

Theorem

The standardized random variable

$$\frac{X_n - \mu_1 n}{\sigma_1 \sqrt{n}},$$

with

$$\mu_1 = \frac{1}{2} + \frac{\log(n)}{4n} + \mathcal{O}\left(\frac{1}{n}\right) \qquad \text{and} \qquad \sigma_1^2 = \frac{1}{4} - \frac{\pi^2}{32n} + \mathcal{O}\left(\frac{1}{n^2}\right),$$

converges in law to a standard normal distribution $\mathcal{N}(0,1)$.

Number of dominating young leaves

We call a young leaf with label k *dominating* if it is still a young leaf in $\mathcal{T}_{k+1}$, i.e., not immediately replaced by a new one.

Number of dominating young leaves

We call a young leaf with label k *dominating* if it is still a young leaf in $\mathcal{T}_{k+1}$, i.e., not immediately replaced by a new one.

Correspondence

- Dominating young leaves in rooted plane increasing trees
- Branches in relaxed binary trees of right height ≤ 1

Number of dominating young leaves

We call a young leaf with label k *dominating* if it is still a young leaf in $\mathcal{T}_{k+1}$, i.e., not immediately replaced by a new one.

Correspondence

- Dominating young leaves in rooted plane increasing trees
- Branches in relaxed binary trees of right height ≤ 1

Let Y_n be its random variable when drawn uniformly at random among all such trees of size n.

Number of dominating young leaves

We call a young leaf with label k *dominating* if it is still a young leaf in $\mathcal{T}_{k+1}$, i.e., not immediately replaced by a new one.

Correspondence

- Dominating young leaves in rooted plane increasing trees
- Branches in relaxed binary trees of right height ≤ 1

Let Y_n be its random variable when drawn uniformly at random among all such trees of size n.

Theorem

The standardized random variable

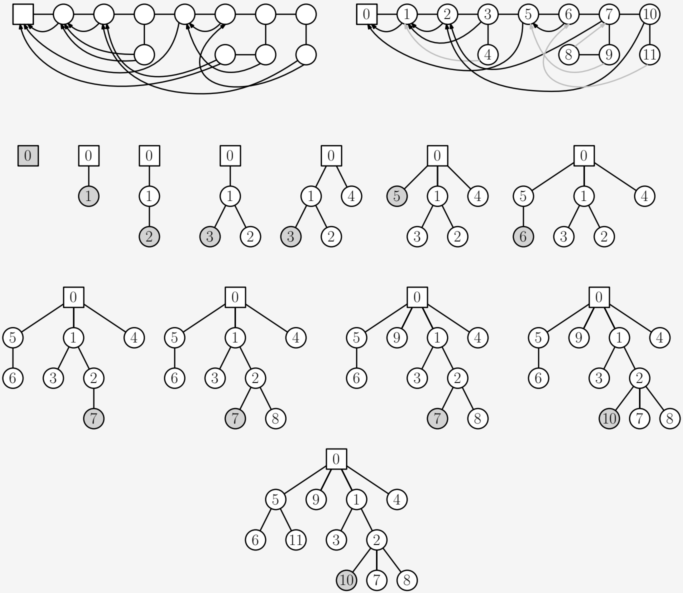
$$\frac{Y_n - \mu_2 n}{\sigma_2 \sqrt{n}},$$

with

$$\mu_2 = \frac{1}{4} - \frac{1}{8n} + \mathcal{O}\left(\frac{1}{n^2}\right) \qquad \textit{and} \qquad \sigma_2^2 = \frac{1}{16} + \frac{1}{32n} + \mathcal{O}\left(\frac{1}{n^2}\right),$$

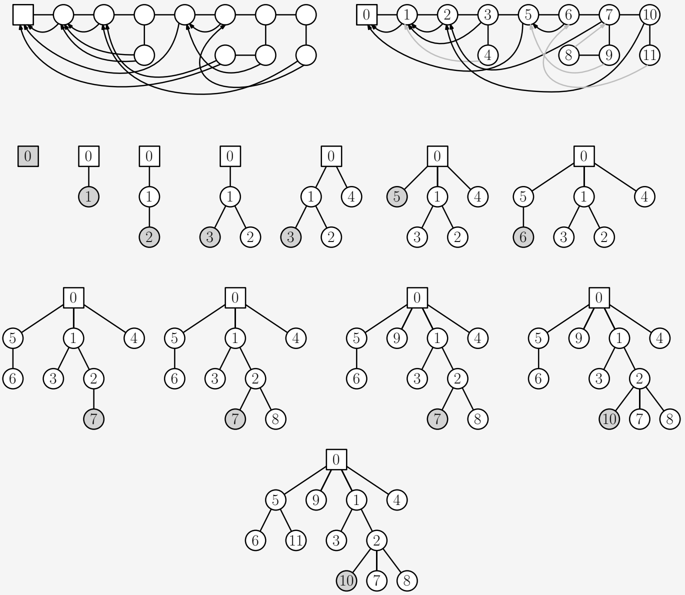
converges in law to a standard normal distribution $\mathcal{N}(0,1)$.

Un bon croquis vaut mieux qu'un long discours



Un bon croquis vaut mieux qu'un long discours

MERCI



BEAUCOUP