Department of Mathematics

UCL

THE QUEST FOR STRUCTURE

Continued Fractions in Enumerative combinatorics[1]

[1] Research project under the supervision of Professor Alan Sokal (UCL) and Dr Bishal Deb (Sorbonne Université) through funds from the UCL Department of Mathematics

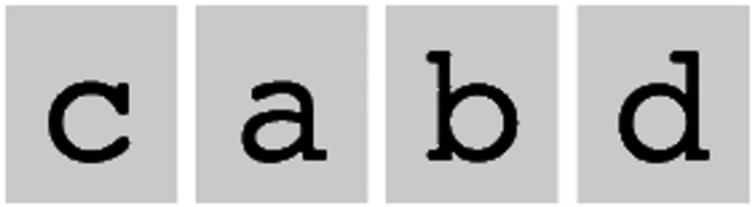
What is *Enumerative Combinatorics*?

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.



What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.

- $$\mathfrak{F} = (S_i)_{i \in I}$$

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.

- $\mathfrak{F} = (S_i)_{i \in I}$

combinatorial family

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.

- $\mathfrak{F} = (S_i)_{i \in I}$

combinatorial family

finite sets

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.

- $\mathfrak{F} = (S_i)_{i \in I}$

combinatorial family — finite sets — indexing set

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.

- $\mathfrak{F} = (S_i)_{i \in I}$

  combinatorial family — finite sets — indexing set

- Goal: find nice closed formula that describes $|S_i|$.

What is *Enumerative Combinatorics*?

- Enumerative Combinatorics deals with counting number of ways some patterns arise.
- $\mathfrak{F} = (S_i)_{i \in I}$

  combinatorial family — finite sets — indexing set

- Goal: find nice closed formula that describes $|S_i|$.
- Not always possible: Generating Functions.

Generating Functions

Generating Functions

- [**Definition**]: "A Generating Function is a representation of a sequence of numbers as coefficients of a power series."

Generating Functions

- [**Definition**]: "A Generating Function is a representation of a sequence of numbers as coefficients of a power series."

$$\sum_{n=0}^{\infty} |S_n| t^n$$

ordinary generating function

$$\sum_{n=0}^{\infty} |S_n| \frac{t^n}{n!}$$

exponential generating function

Generating Functions

- [**Definition**]: "A Generating Function is a representation of a sequence of numbers as coefficients of a power series."

$$\sum_{n=0}^{\infty} |S_n| t^n$$

ordinary generating function

$$\sum_{n=0}^{\infty} |S_n| \frac{t^n}{n!}$$

exponential generating function

- The above are viewed as *formal power series*.

Generating Functions

- [**Definition**]: "A Generating Function is a representation of a sequence of numbers as coefficients of a power series."

$$\sum_{n=0}^{\infty} |S_n| t^n$$

ordinary generating function

$$\sum_{n=0}^{\infty} |S_n| \frac{t^n}{n!}$$

exponential generating function

- The above are viewed as *formal power series*.
- Some motivation: as formal power series they behave nicely.

Generating Functions

- [**Definition**]: "A Generating Function is a representation of a sequence of numbers as coefficients of a power series."

$$\sum_{n=0}^{\infty} |S_n| t^n$$

ordinary generating function

$$\sum_{n=0}^{\infty} |S_n| \frac{t^n}{n!}$$

exponential generating function

- The above are viewed as *formal power series*.
- Some motivation: as formal power series they behave nicely.
- Curiosity: set of all formal power series forms an integral domain. Particularly is a PID and any ideal as the form (t^n).

Continued Fractions

Continued Fractions

One way to represent a generating function: continued fractions.

Continued Fractions

One way to represent a generating function: continued fractions.

$$\sum_{n=0}^{\infty} |S_n| t^n = \cfrac{1}{1 - \cfrac{\alpha_1 t}{1 - \cfrac{\alpha_2 t}{1 - \cfrac{\alpha_3 t}{1 - \cfrac{\alpha_4 t}{1 - ....}}}}}$$

Stieltjes-type continued fractions
S-fractions

$$\sum_{n=0}^{\infty} |S_n| t^n = \cfrac{1}{1 - \gamma_0 t - \cfrac{\beta_1 t^2}{1 - \gamma_1 t - \cfrac{\beta_2 t^2}{1 - \gamma_2 t - \cfrac{\beta_3 t^2}{1 - \gamma_3 t - \cfrac{\beta_4 t^2}{1 - ....}}}}}$$

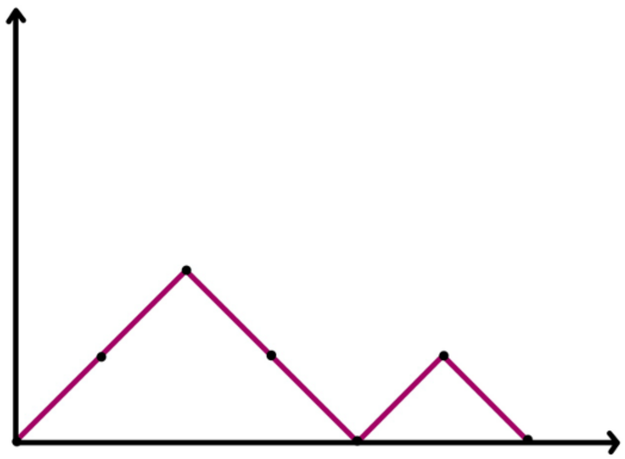
Jacobi-type continued fractions
J-fractions

$$\sum_{n=0}^{\infty} |S_n| t^n = \cfrac{1}{1 - \gamma_1 t - \cfrac{\alpha_1 t}{1 - \gamma_2 t - \cfrac{\alpha_2 t}{1 - \gamma_3 t - \cfrac{\alpha_3 t}{1 - \gamma_4 t - \cfrac{\alpha_4 t}{1 - ....}}}}}$$

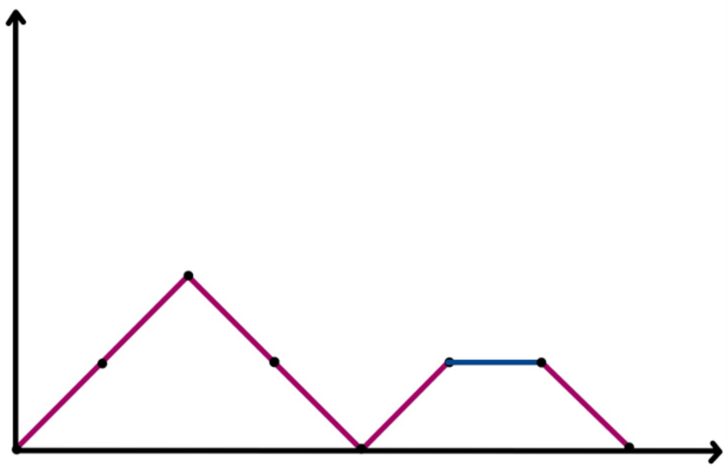
Thron-type continued fractions
T-fractions

Flajolet (1980) showed the connection between S-, J- and T-fractions with Dyck, Motzkin, Schröder paths.

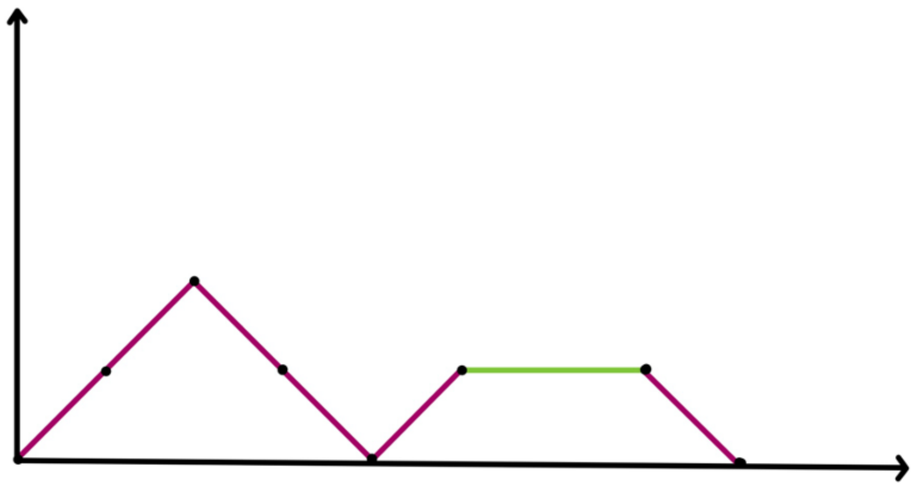
Flajolet (1980) showed the connection between S-, J- and T-fractions with Dyck, Motzkin, Schröder paths.



Dyck Path

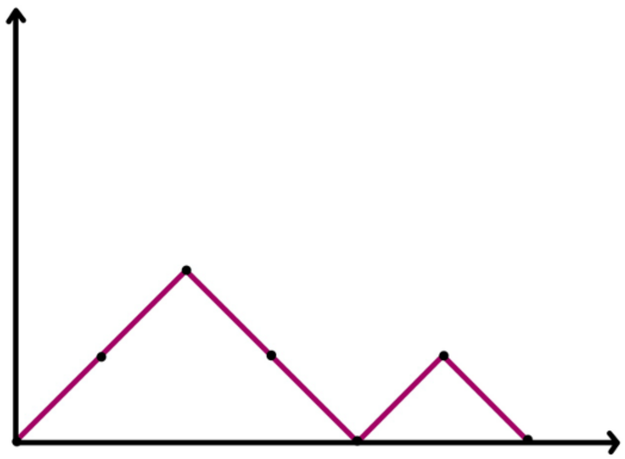


Motzkin Path

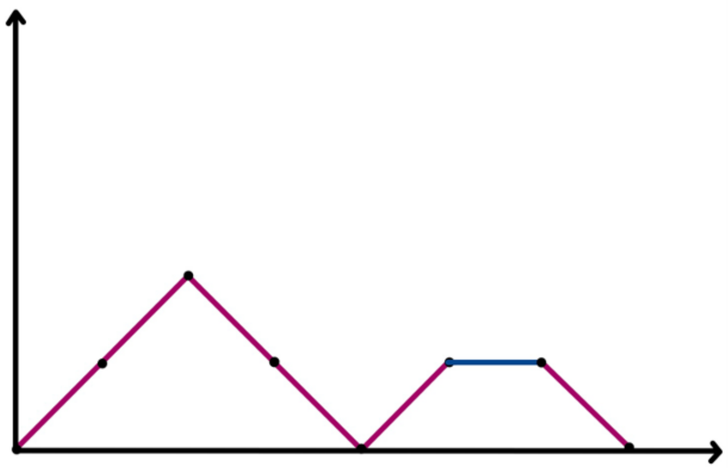


Schröder Path

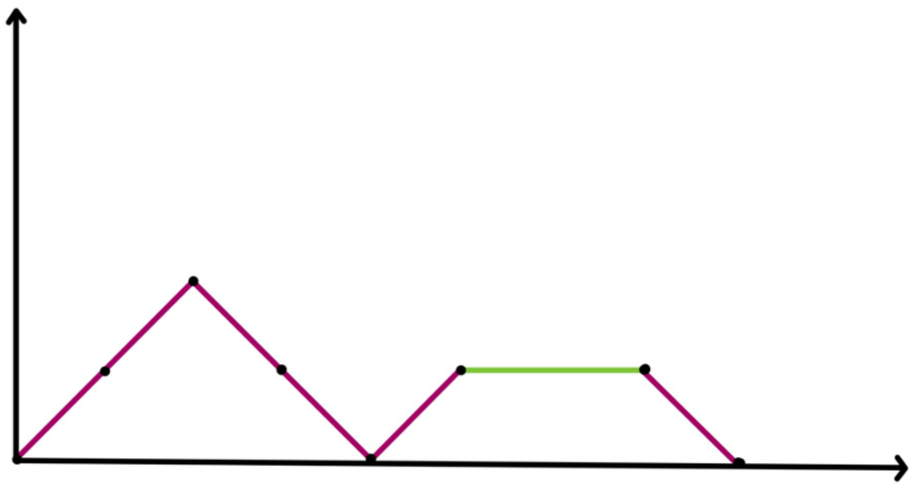
Flajolet (1980) showed the connection between S-, J- and T-fractions with Dyck, Motzkin, Schröder paths.



Dyck Path



Motzkin Path



Schröder Path

How? Let's consider the easiest case: (weighted) Dyck paths.

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$$S_0 = 1$$

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$$S_0 = 1$$

•

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$$S_0 = 1$$

$$S_1 = \alpha_1$$

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$S_0 = 1$

$S_1 = \alpha_1$

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$S_0 = 1$

$S_1 = \alpha_1$

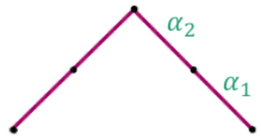
$S_2 = \alpha_1\ \alpha_2 + {\alpha_1}^2$

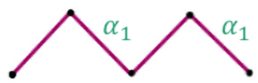
$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$S_0 = 1$

$S_1 = \alpha_1$

$S_2 = \alpha_1\ \alpha_2 + {\alpha_1}^2$



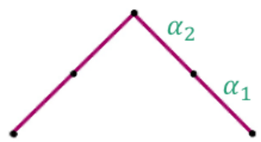


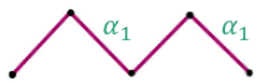
$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

$S_0 = 1$

$S_1 = \alpha_1$

$S_2 = \alpha_1\ \alpha_2 + {\alpha_1}^2$





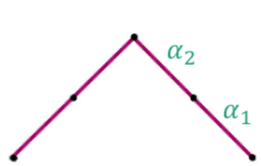
$S_3 = \alpha_1\alpha_2\alpha_3 + 2{\alpha_1}^2\alpha_2 + {\alpha_1}^3$

$\boldsymbol{\alpha} = (\alpha_i)_{i\geq 1}$, $S_n(\boldsymbol{\alpha})$ *Stieltjes-Rogers polynomials*

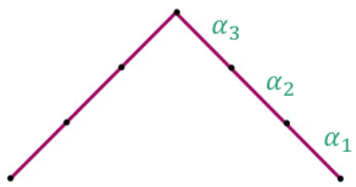
$S_0 = 1$

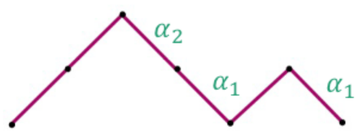
$S_1 = \alpha_1$

$S_2 = \alpha_1\ \alpha_2 + {\alpha_1}^2$



$S_3 = \alpha_1\alpha_2\alpha_3 + 2{\alpha_1}^2\alpha_2 + {\alpha_1}^3$

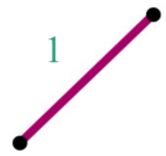




- So the weights are :

- So the weights are :

- So the weights are :



- So the weights are :

- So the weights are :

- So the weights are :

- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.

- So the weights are :

- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.

- So the weights are :

- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

- So the weights are :

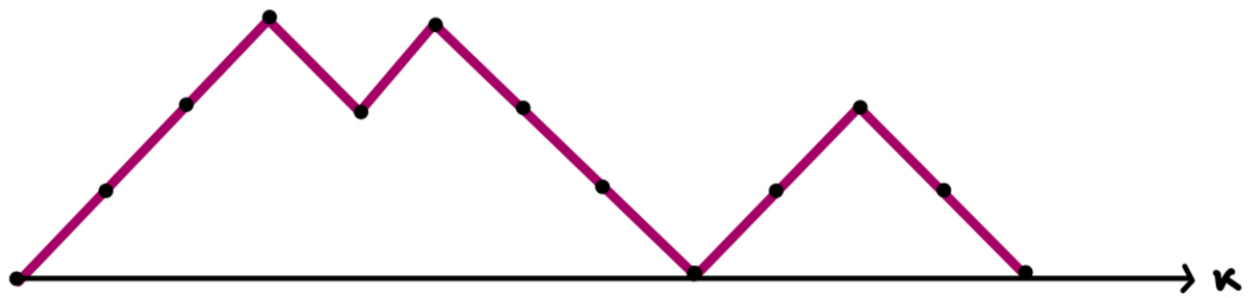
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_k(t) = 1 + \alpha_{k+1} t f_k(t) f_{k+1}(t)$$

- So the weights are :

- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_k(t) = 1 + \alpha_{k+1} t f_k(t) f_{k+1}(t)$$



- So the weights are :

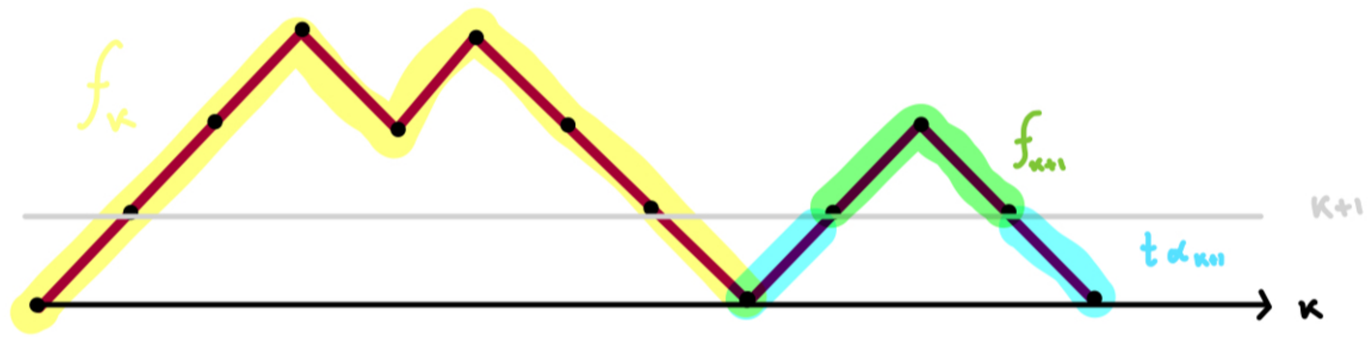
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_k(t) = 1 + \alpha_{k+1} t f_k(t) f_{k+1}(t)$$

- So the weights are :

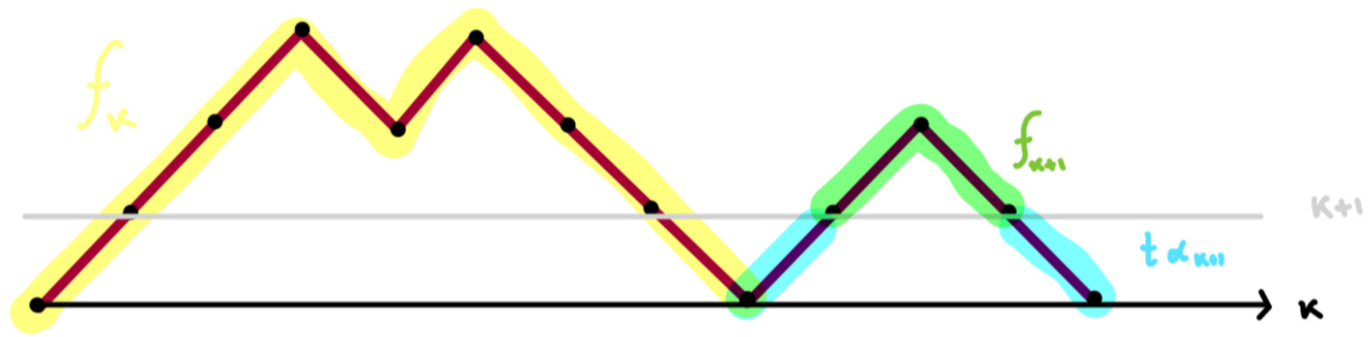
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_k(t) = 1 + \alpha_{k+1} t f_k(t) f_{k+1}(t)$$



- So the weights are :

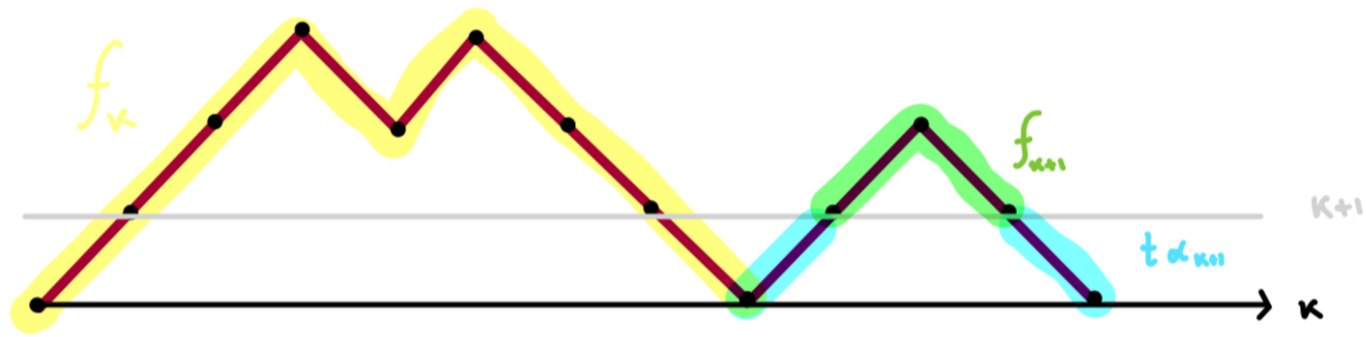
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:



- So the weights are :

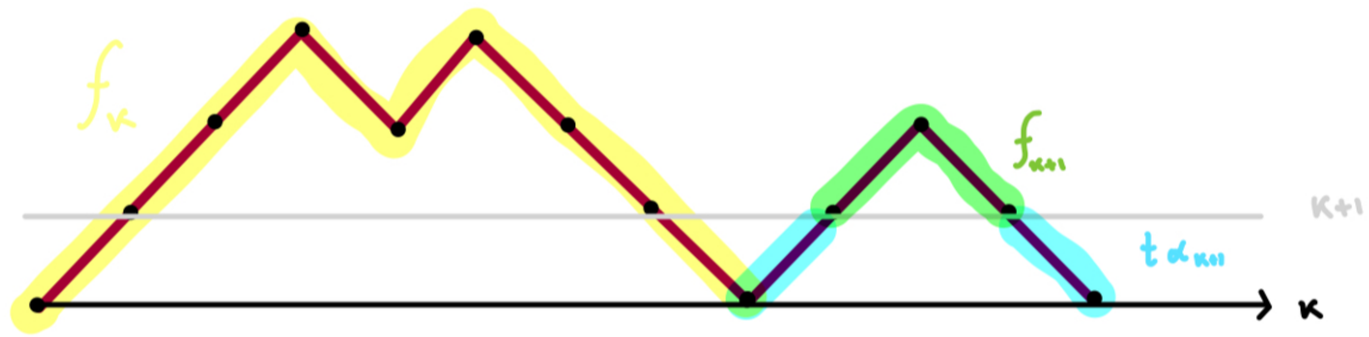
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_k(t) = \frac{1}{1 - \alpha_{k+1} t f_{k+1}(t)}$$



- So the weights are :

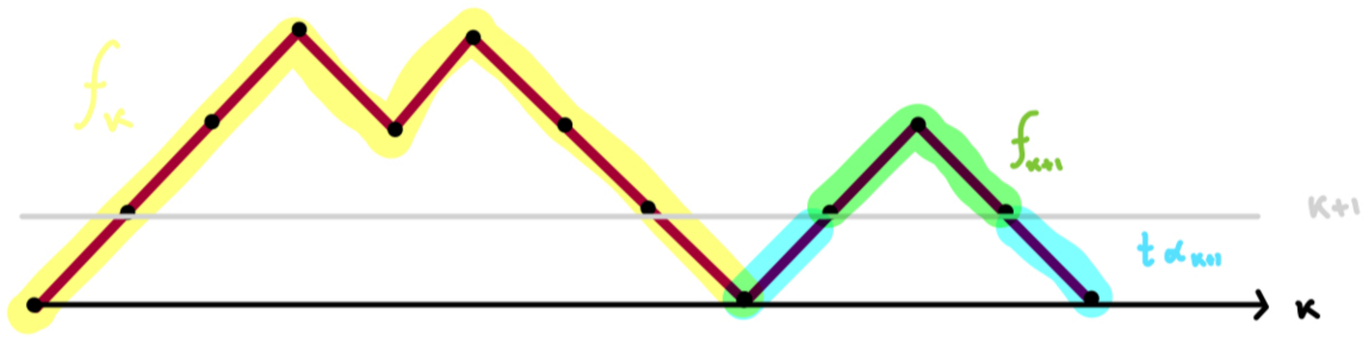
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:



- So the weights are :

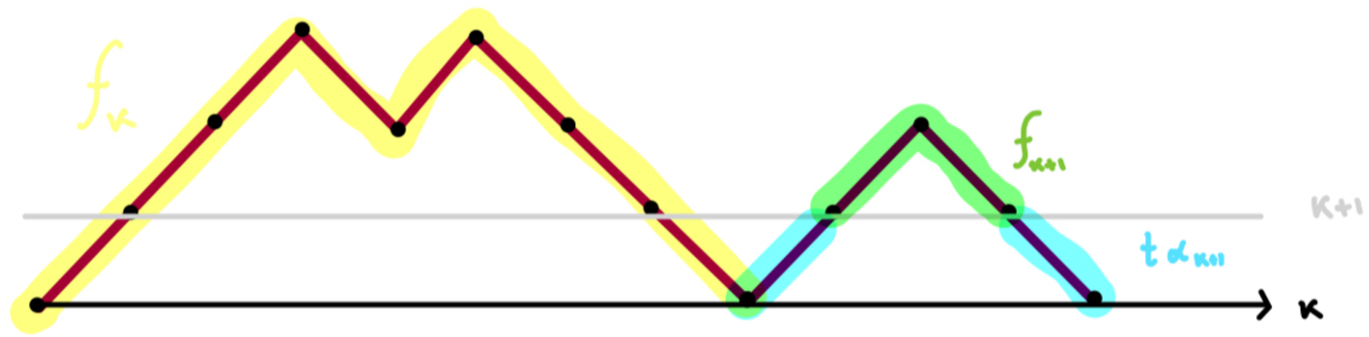
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_k(t) = \cfrac{1}{1 - \cfrac{\alpha_{k+1}t}{1 - \cfrac{\alpha_{k+2}t}{1 - \cfrac{\alpha_{k+3}t}{1 - \cfrac{\alpha_{k+4}t}{1 - ....}}}}}$$



- So the weights are :

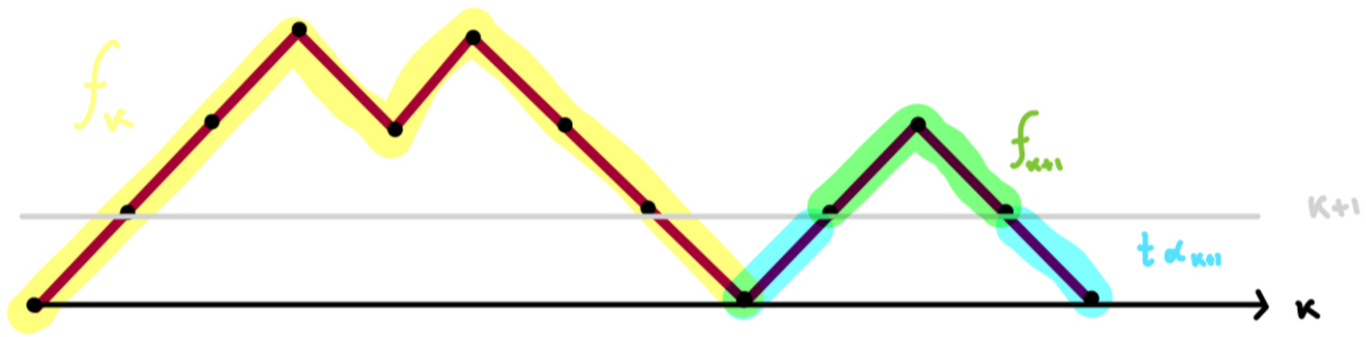
- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:



- So the weights are :

- Let $f_0(t) = \sum_{n=0}^{\infty} S_n(\boldsymbol{\alpha}) t^n$ be the o.g.f. for weighted Dyck Paths.
- In general, let $f_k(t)$ be the o.g.f for Dyck Paths at level k.
- Then:

$$f_0(t) = \cfrac{1}{1 - \cfrac{\alpha_1 t}{1 - \cfrac{\alpha_2 t}{1 - \cfrac{\alpha_3 t}{1 - \cfrac{\alpha_4 t}{1 - \ldots.}}}}}$$



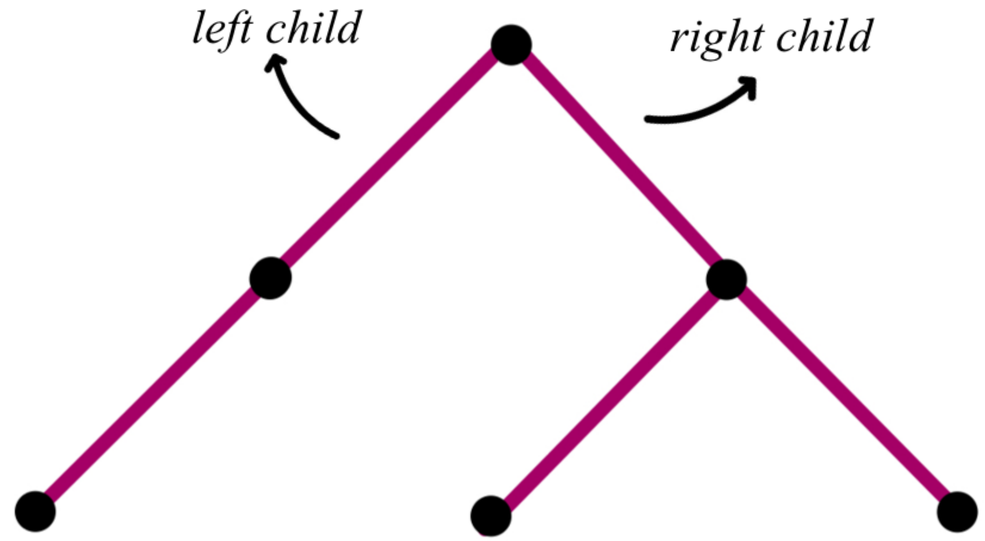
An example: (unlabelled) binary trees

An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

An example: (unlabelled) binary trees

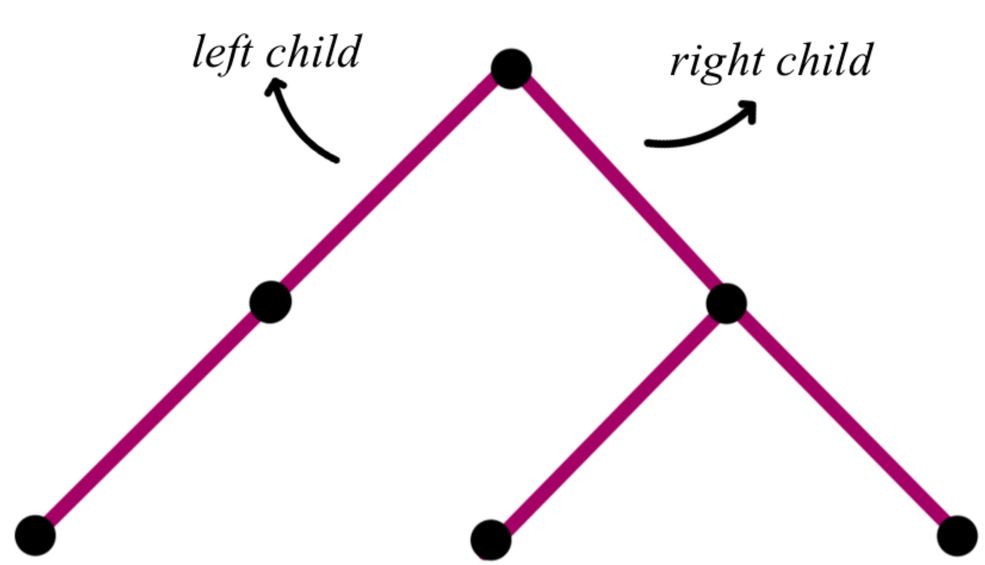
[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

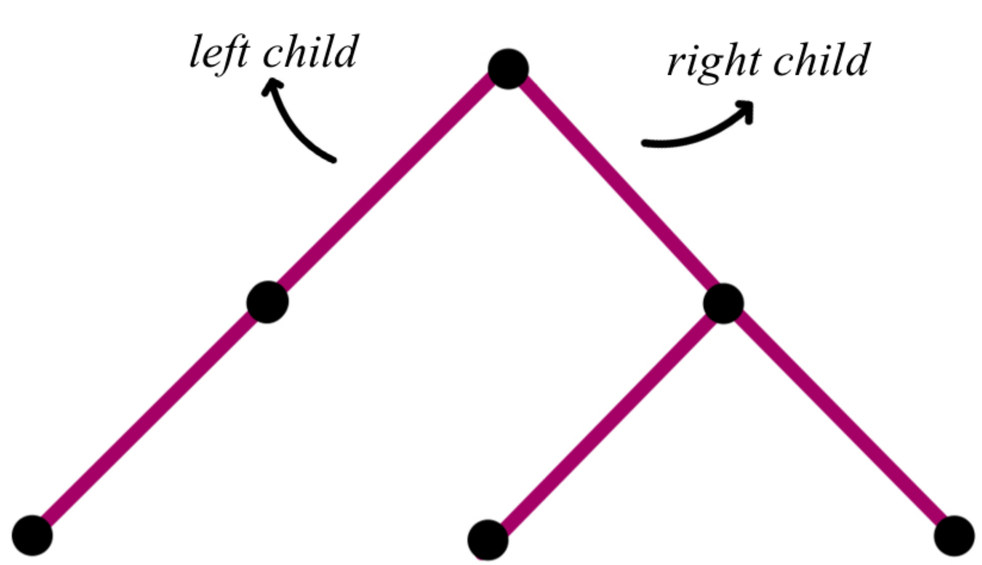


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

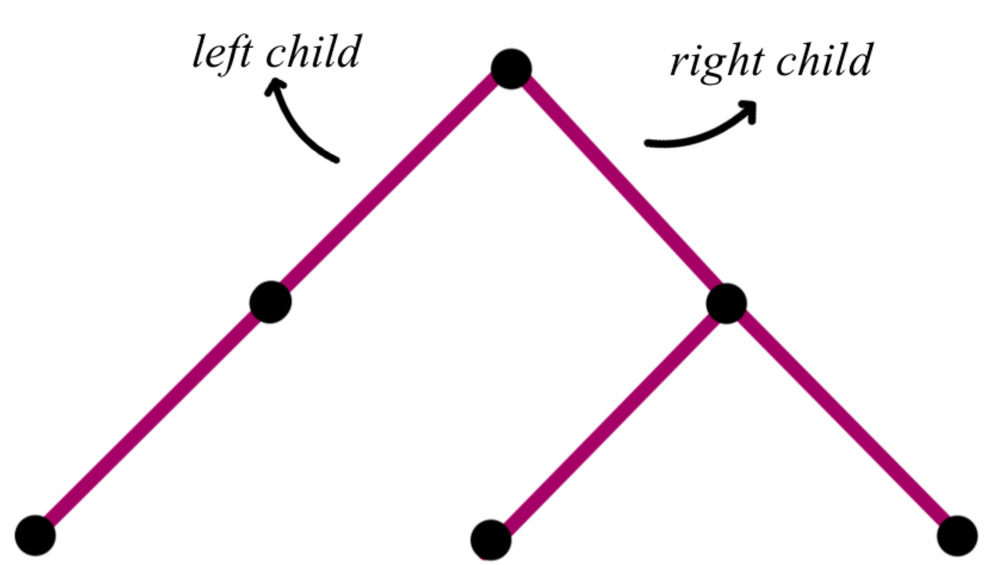
$$f(t) = 1 + tf(t)^2$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

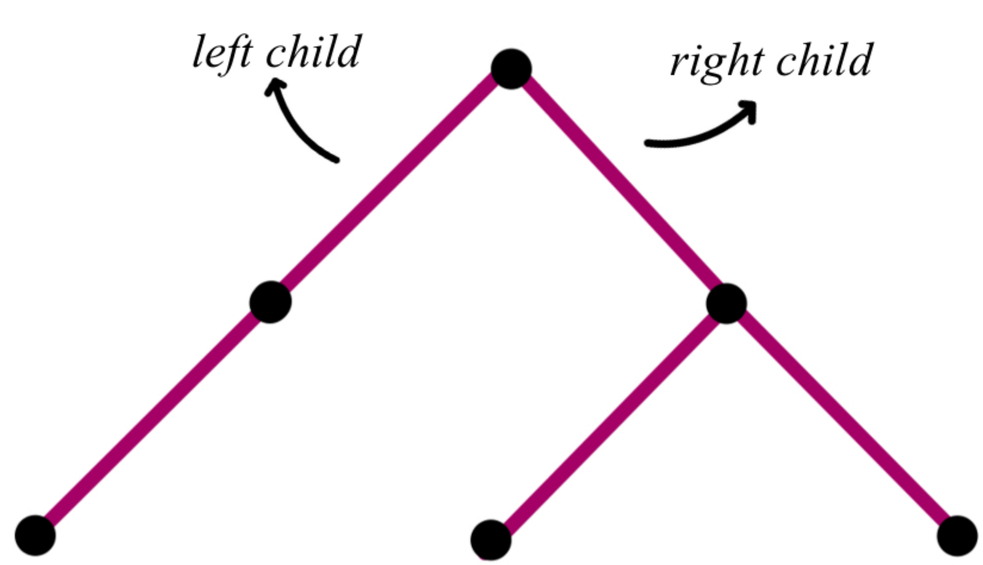


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

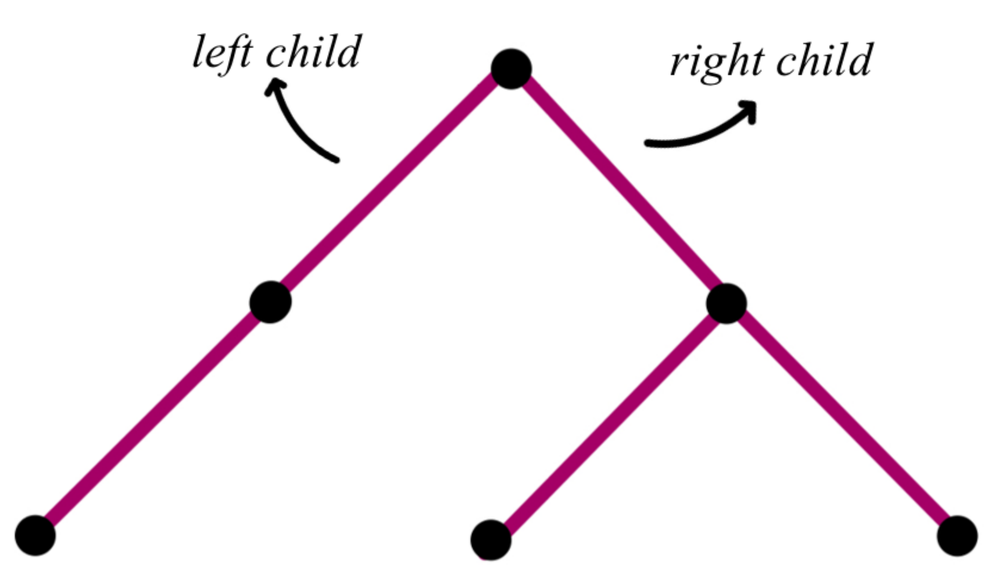
$$f(t) = \frac{1 \pm \sqrt{1-4t}}{2t}$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

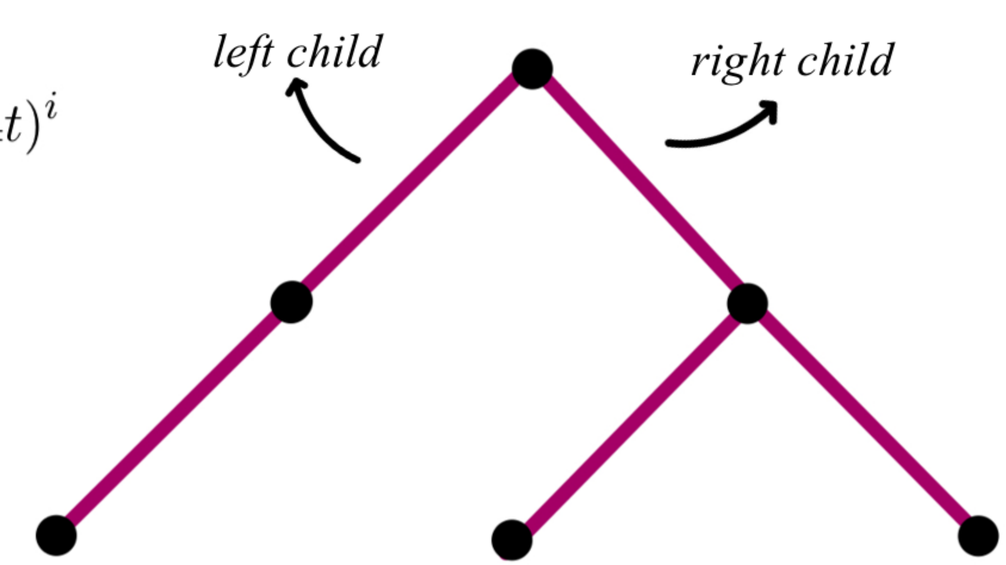


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

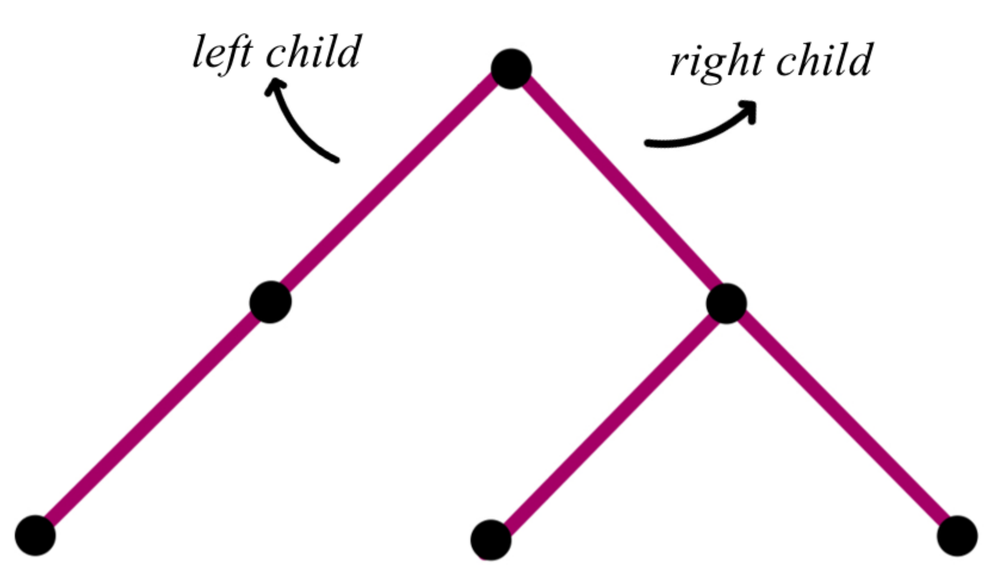
$$(1-4t)^{\frac{1}{2}} = \sum_{i=0}^{n} \binom{\frac{1}{2}}{i} (-4t)^i = ... = (1-2t) \sum_{i=2}^{n} \frac{(2i-3)!!}{2^i i!} (-4t)^i$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

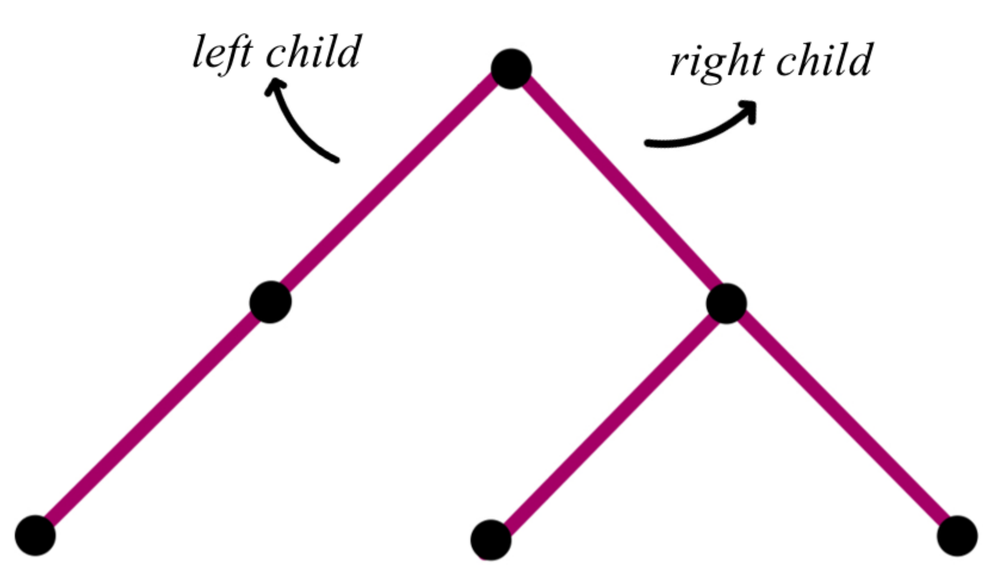


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

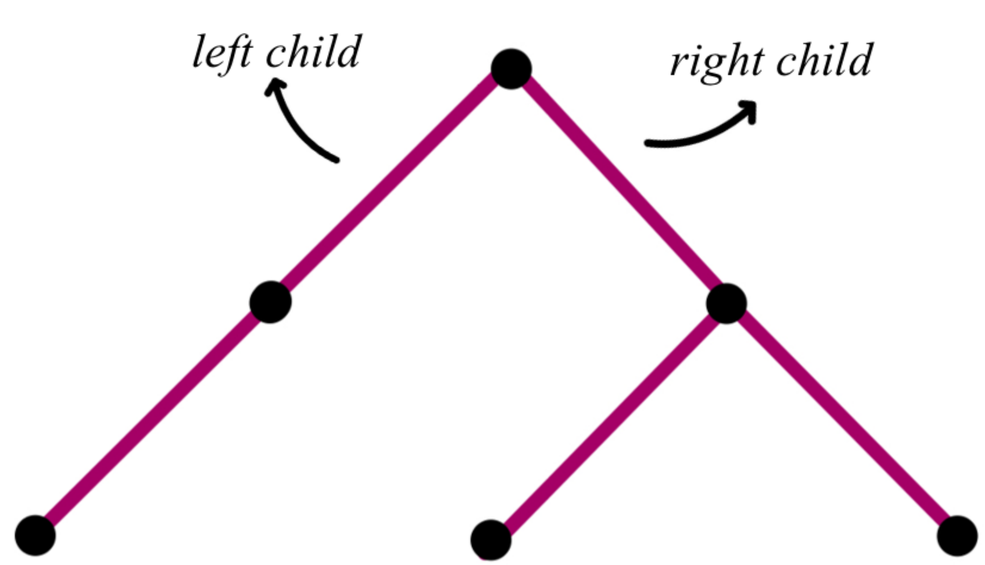
$$\frac{1 + 1 - 2t + ...}{2t}$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

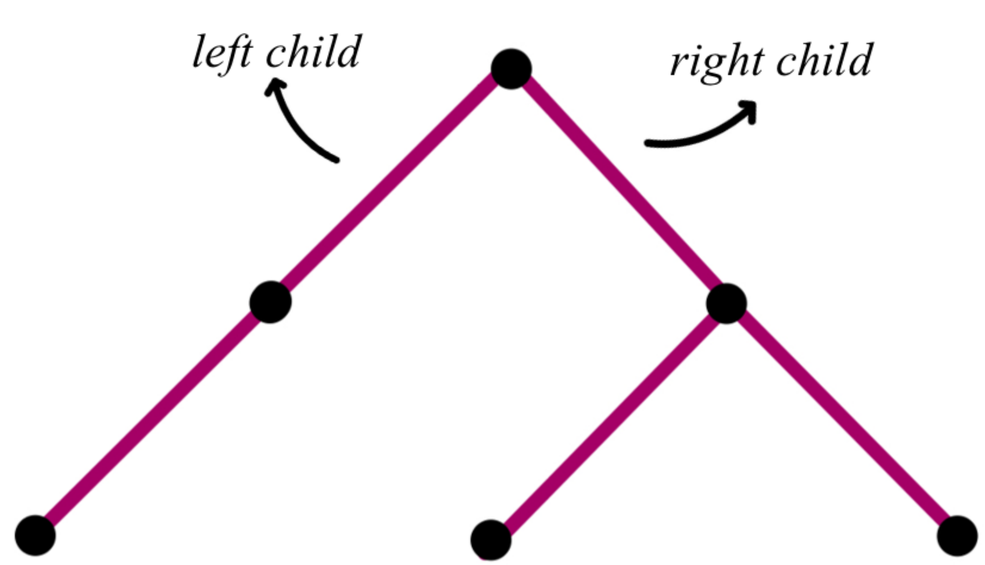


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

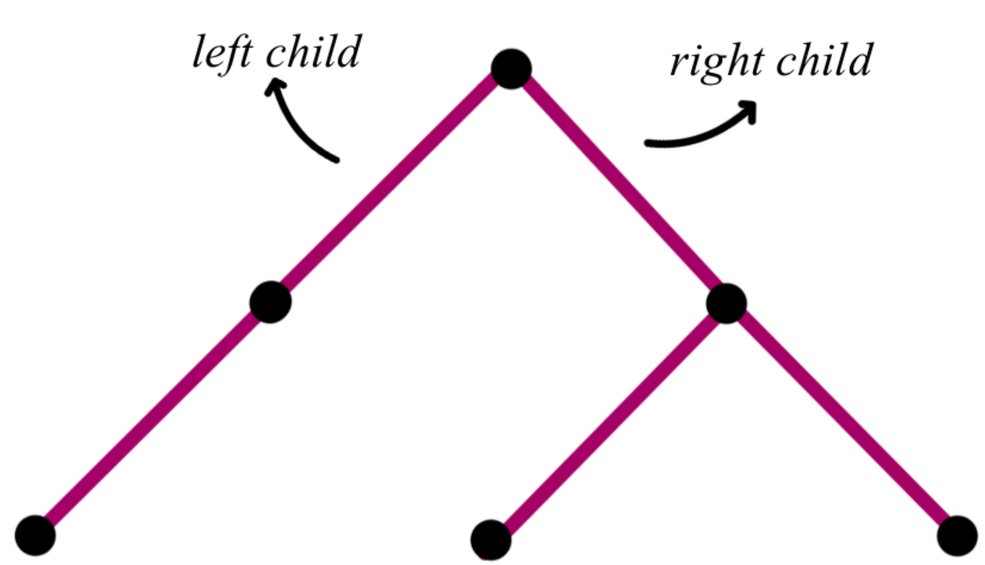
$$\frac{1 - 1 + 2t - ....}{2t}$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

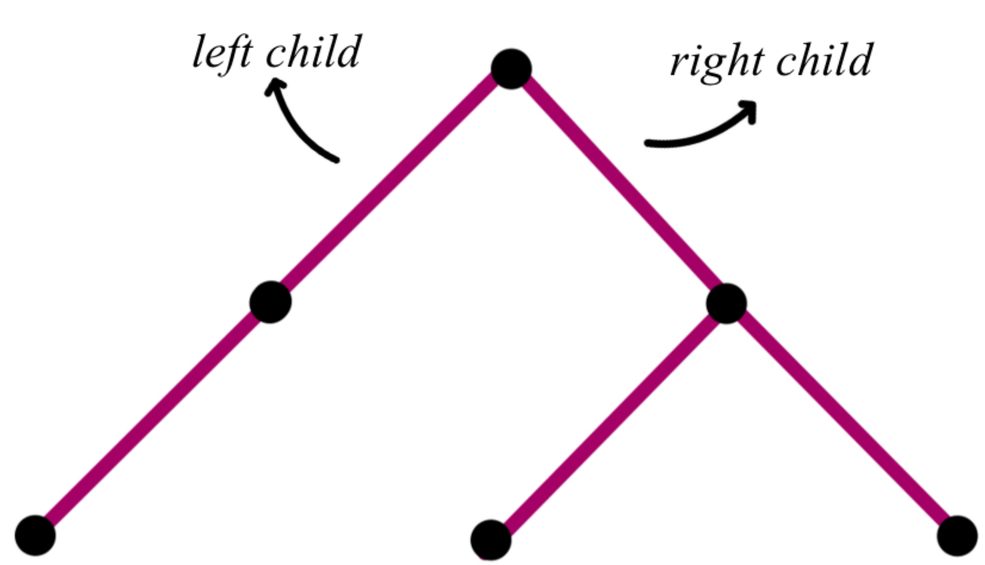


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

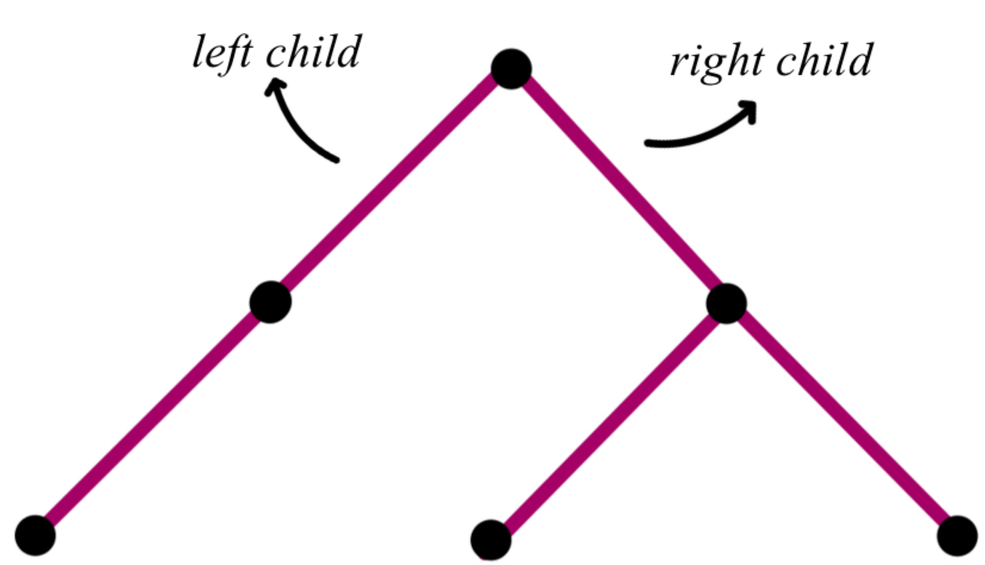
$$f(t) = \frac{1-\sqrt{1-4t}}{2t}$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$



An example: (unlabelled) binary trees

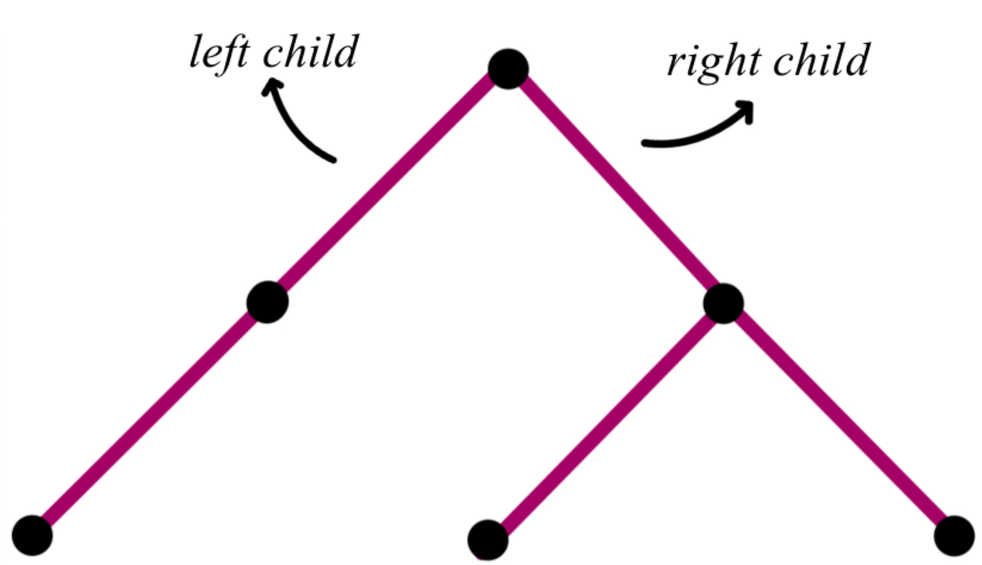
[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

$$f(t) = 2t - \sum_{n=2}^{\infty} \left(\frac{(2n-3)!!(-1)^{n-1}(-t)^n}{2^n n!} \right) \frac{1}{2t} =$$

$$= 1 + \sum_{n=2}^{\infty} \frac{2^{n-1}(2n-3)!!t^{n-1}}{n!} =$$

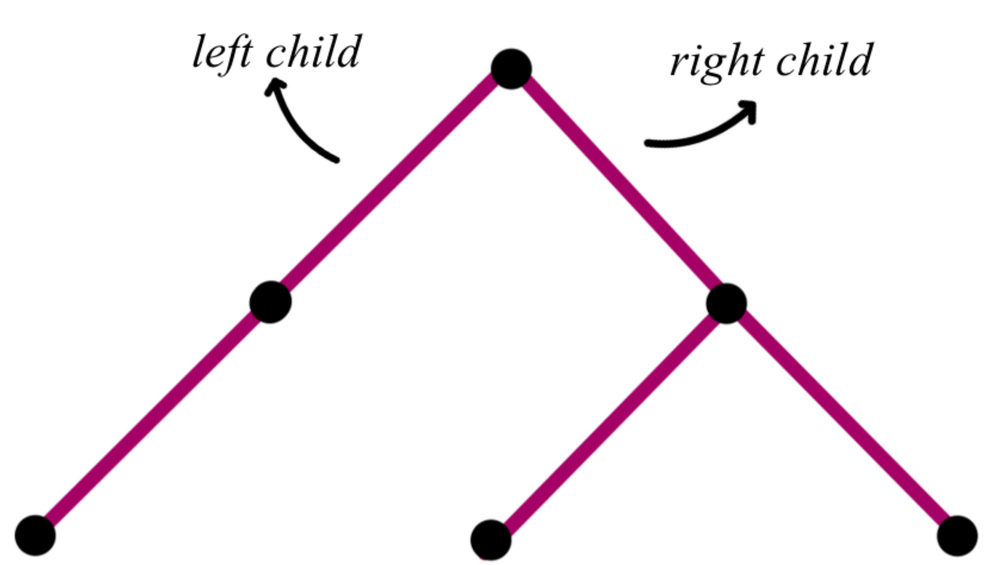
$$= 1 + \sum_{n=1}^{\infty} \frac{2^n(2n-1)!!t^n}{(n+1)!}$$



An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

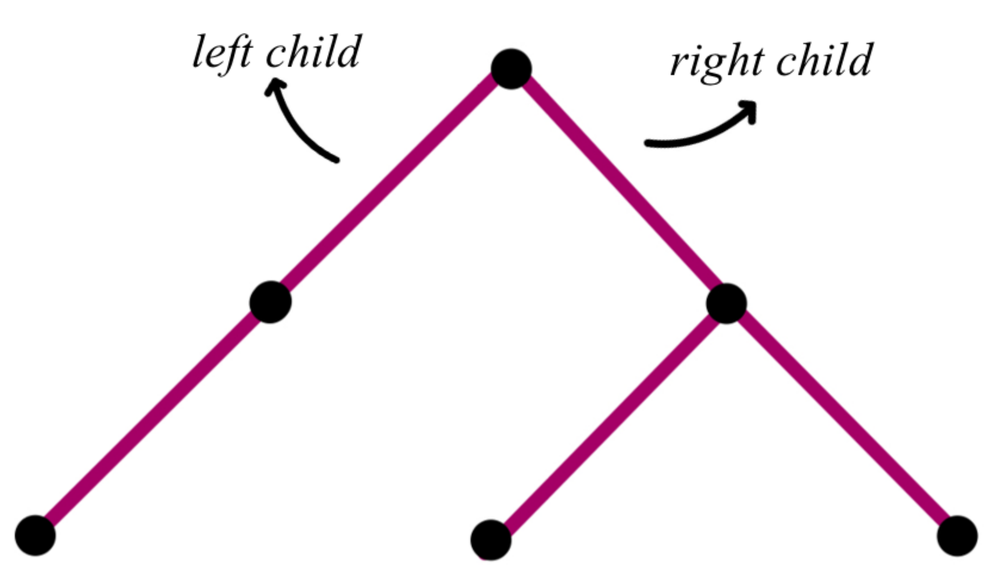


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

$$T_n = \frac{2^n(2n-1)!!}{(n+1)!} = ... = \frac{1}{n+1}\binom{2n}{n}$$



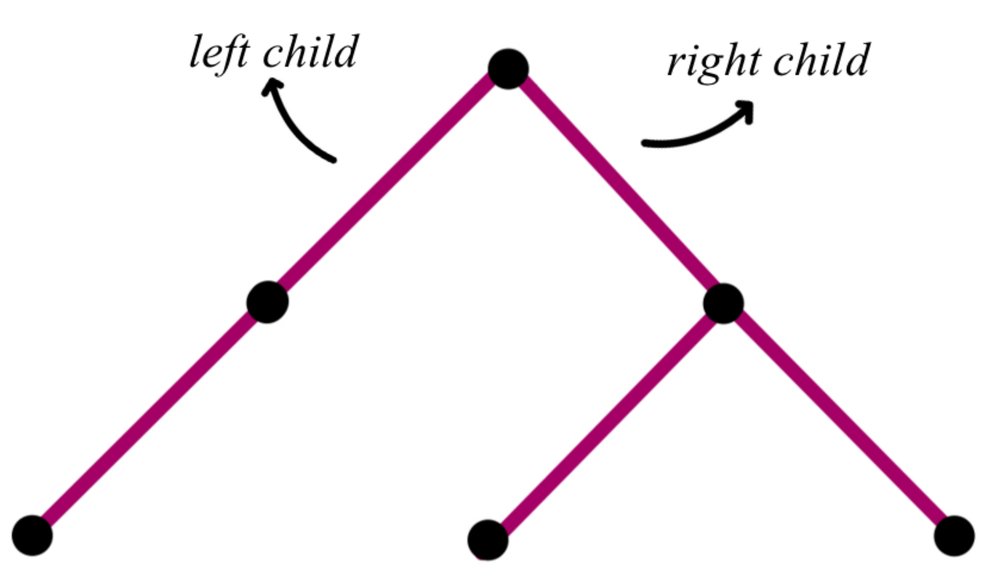
An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

$$T_n = \frac{2^n(2n-1)!!}{(n+1)!} = ... = \frac{1}{n+1}\binom{2n}{n}$$

Catalan Numbers!

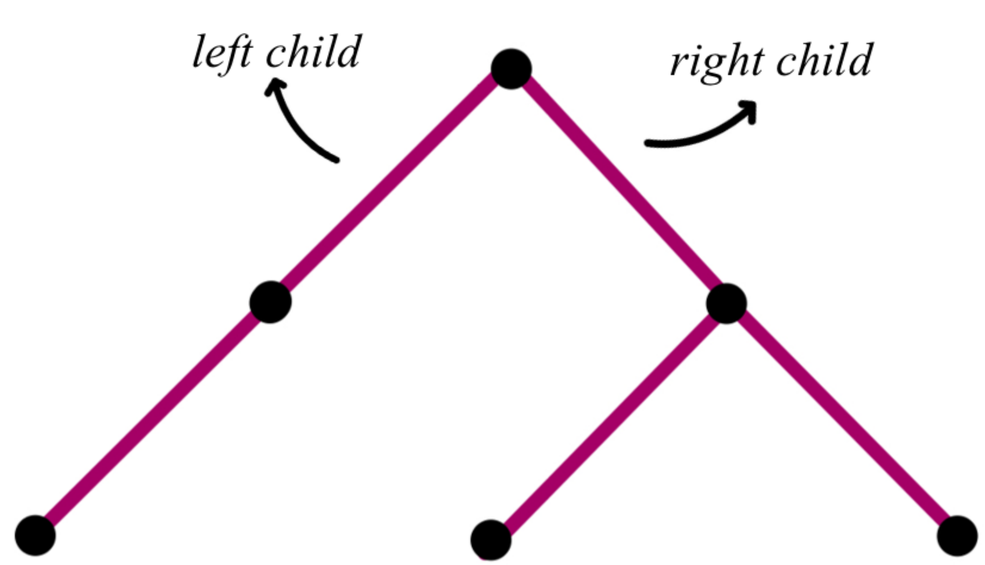


An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

$$T_n = \frac{2^n(2n-1)!!}{(n+1)!} = ... = \frac{1}{n+1}\binom{2n}{n}$$



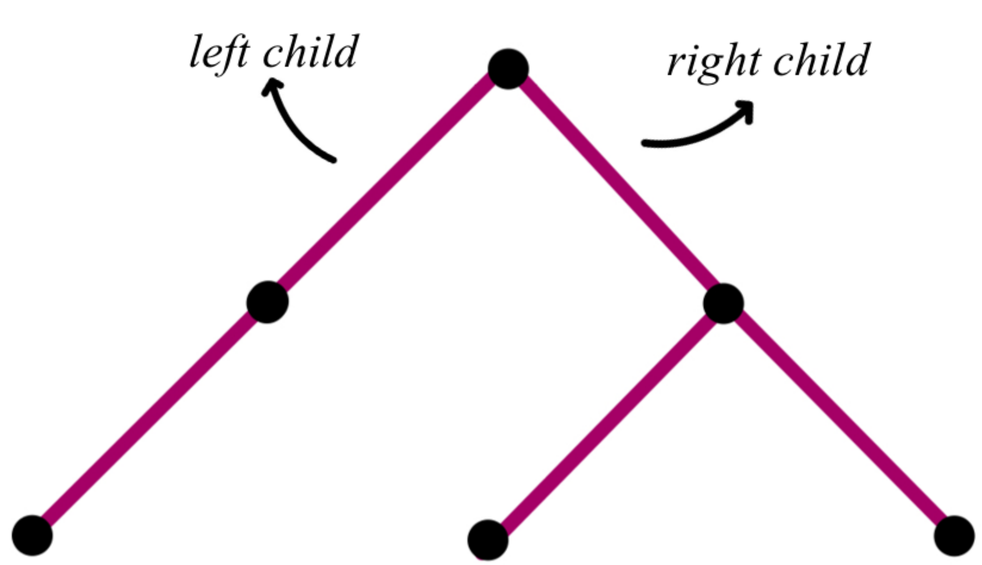
An example: (unlabelled) binary trees

[**Definition**]: "A tree is a connected graph with no circuit. We call its vertices nodes. In particular, a tree is said to be binary if it is either empty or is made of a root, a left and/or right subtree both of which are binary."

$$\mathcal{F} = (T_n)_{n\in\mathbb{N}}$$

$$T_n = \frac{2^n(2n-1)!!}{(n+1)!} = ... = \frac{1}{n+1}\binom{2n}{n}$$

$$\sum_{n=0}^{\infty} T_n t^n = \cfrac{1}{1-\cfrac{t}{1-\cfrac{t}{1-\cfrac{t}{1-\cfrac{t}{1-....}}}}}$$



How does the bijection work?

How does the bijection work?

- [**Definition**]: “A binary tree is said to be complete if every node has either two children or no child at all.”

How does the bijection work?

- [**Definition**]: “A binary tree is said to be complete if every node has either two children or no child at all.”

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves

How does the bijection work?

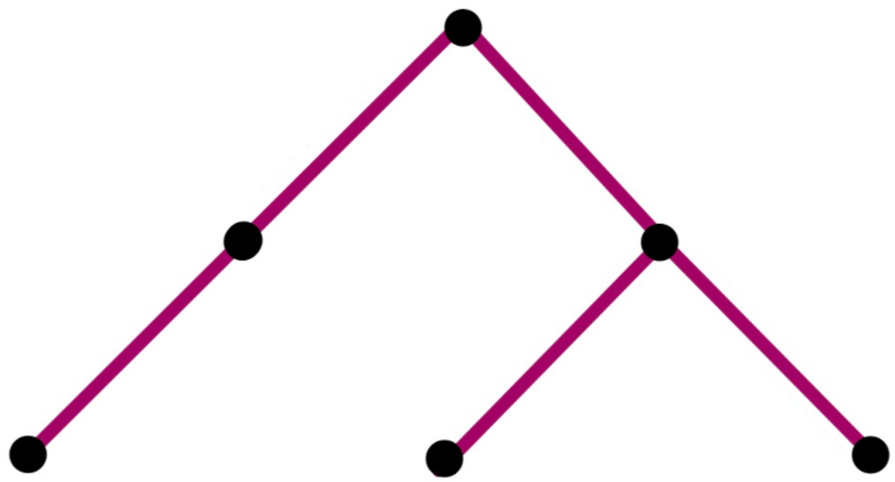
- [**Definition**]: "A binary tree is said to be complete if every node has either two children or no child at all."

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves
- *complete* binary tree => binary tree : remove the n+1 leaves

How does the bijection work?

- [**Definition**]: "A binary tree is said to be complete if every node has either two children or no child at all."

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves
- *complete* binary tree => binary tree : remove the n+1 leaves



How does the bijection work?

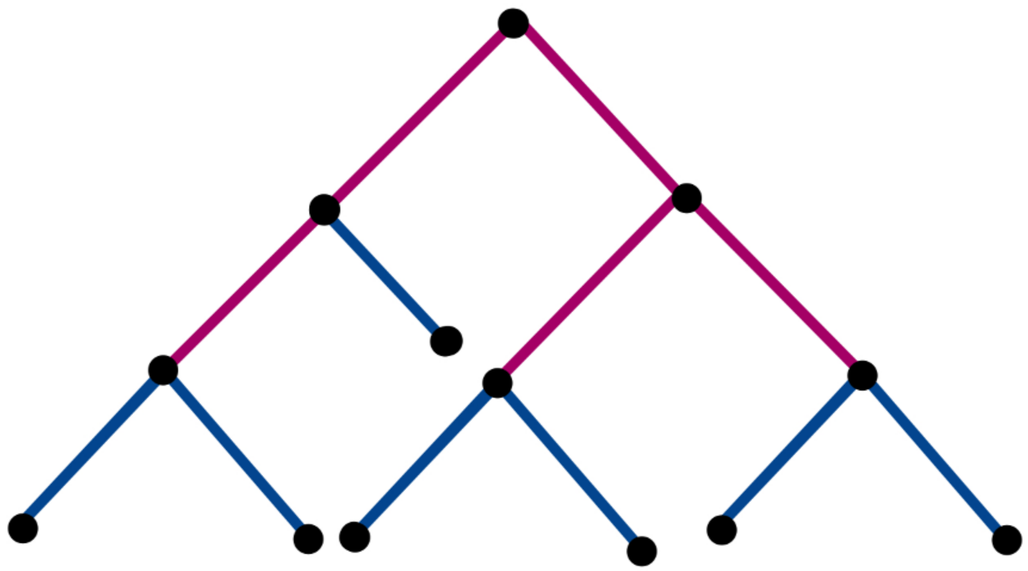
- [**Definition**]: "A binary tree is said to be complete if every node has either two children or no child at all."

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves
- *complete* binary tree => binary tree : remove the n+1 leaves

How does the bijection work?

- [**Definition**]: "A binary tree is said to be complete if every node has either two children or no child at all."

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves
- *complete* binary tree => binary tree : remove the n+1 leaves



How does the bijection work?

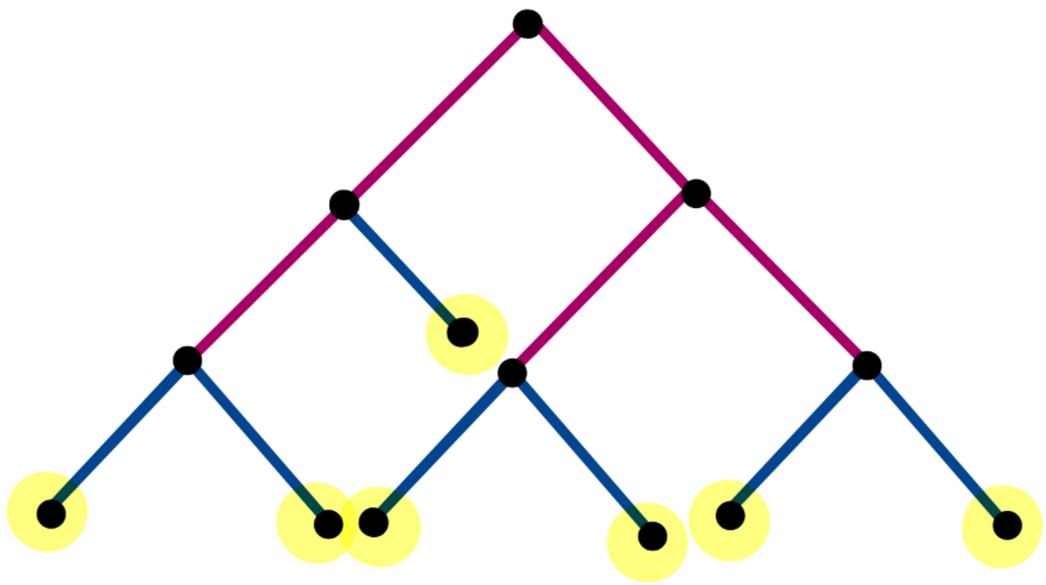
- [**Definition**]: "A binary tree is said to be complete if every node has either two children or no child at all."

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves
- *complete* binary tree => binary tree : remove the n+1 leaves

How does the bijection work?

- [**Definition**]: "A binary tree is said to be complete if every node has either two children or no child at all."

- binary tree (n nodes) => *complete* binary tree (2n+1 nodes) : add the missing leaves
- *complete* binary tree => binary tree : remove the n+1 leaves



How does the bijection work?

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child
  fall => add leaf

How does the bijection work?

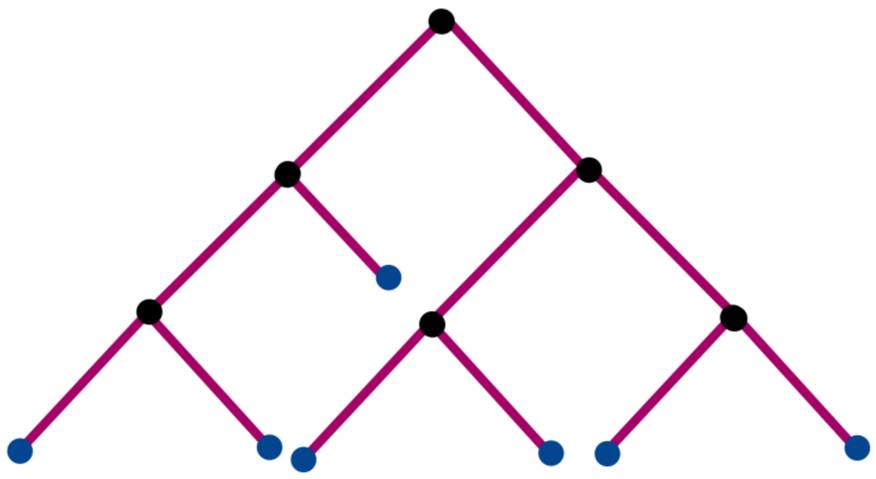
- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child
  fall => add leaf
  NLR

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child
  fall => add leaf
  NLR



How does the bijection work?

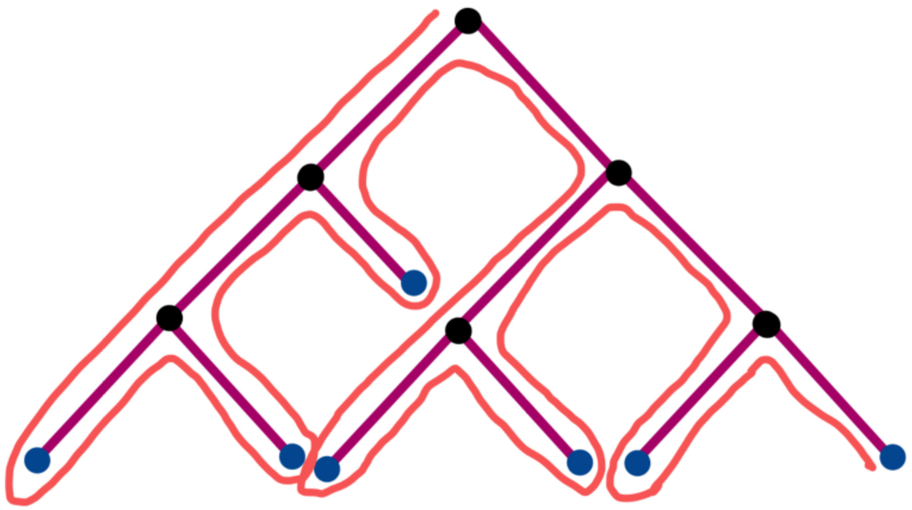
- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child
  fall => add leaf
  NLR

How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

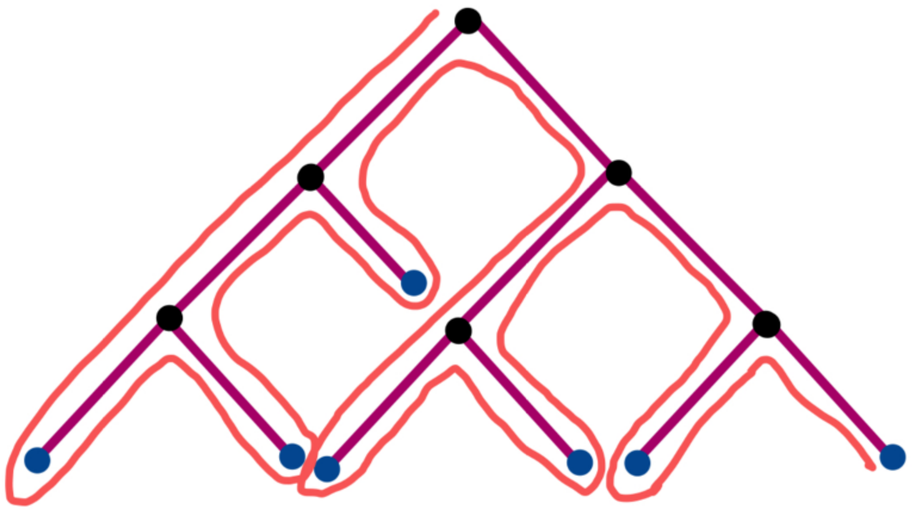
- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child
  fall => add leaf
  NLR

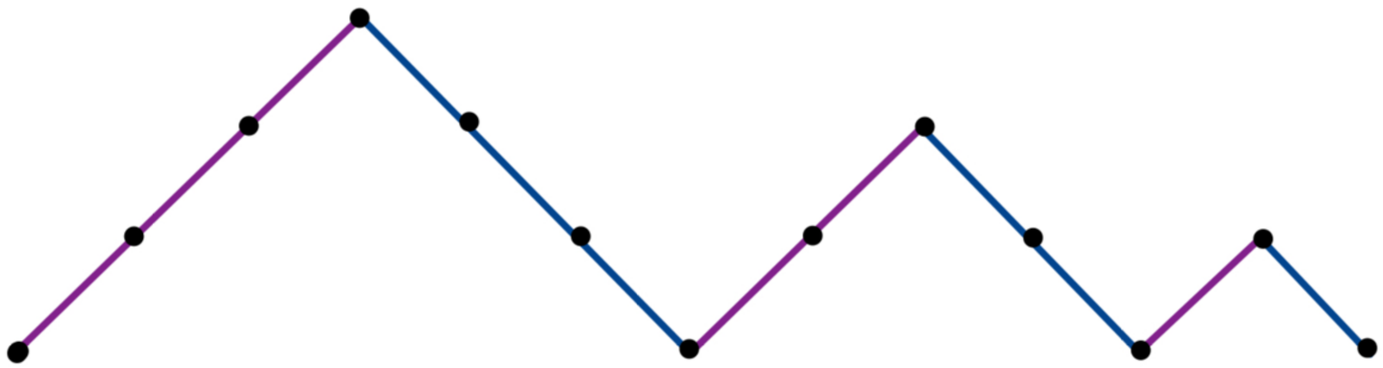


How does the bijection work?

- *complete* binary tree (2n+1) => Dyck Path (2n) : traverse tree in pre-order NLR.
  internal vertex => rise
  leaf => fall

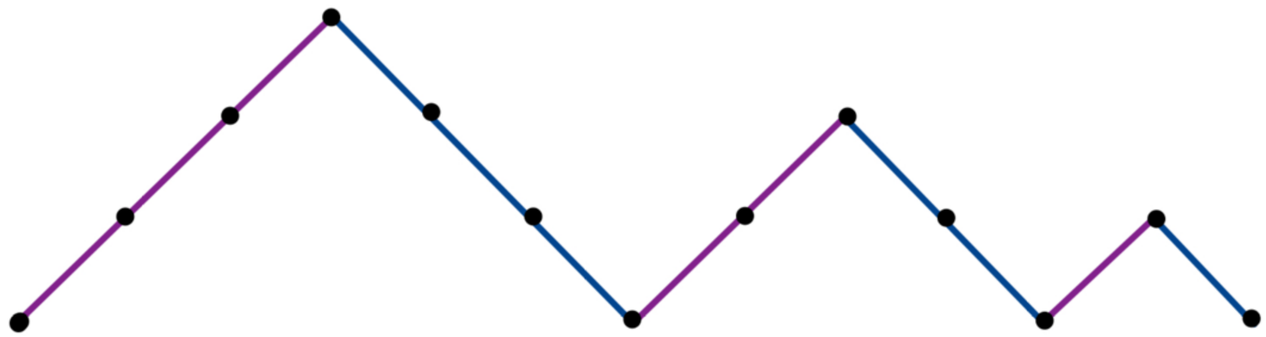
- Dyck Path (2n) => *complete* binary tree (2n+1) : rise => add left and right child
  fall => add leaf
  NLR



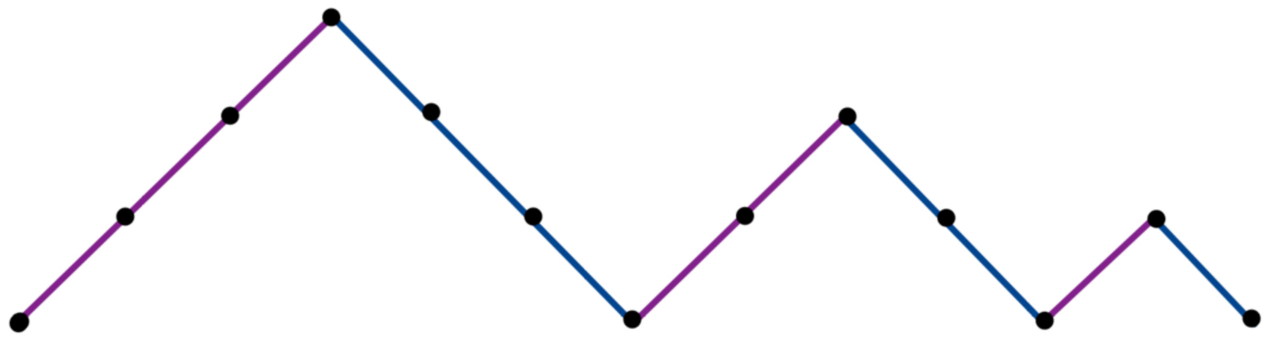


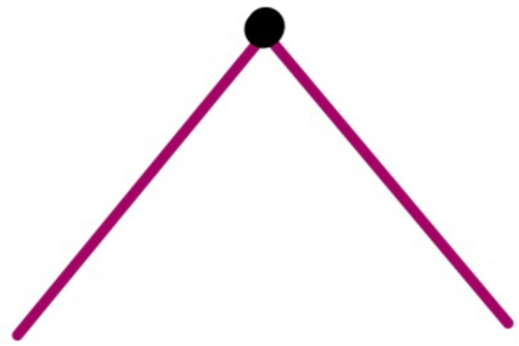
How does the bijection work?

How does the bijection work?

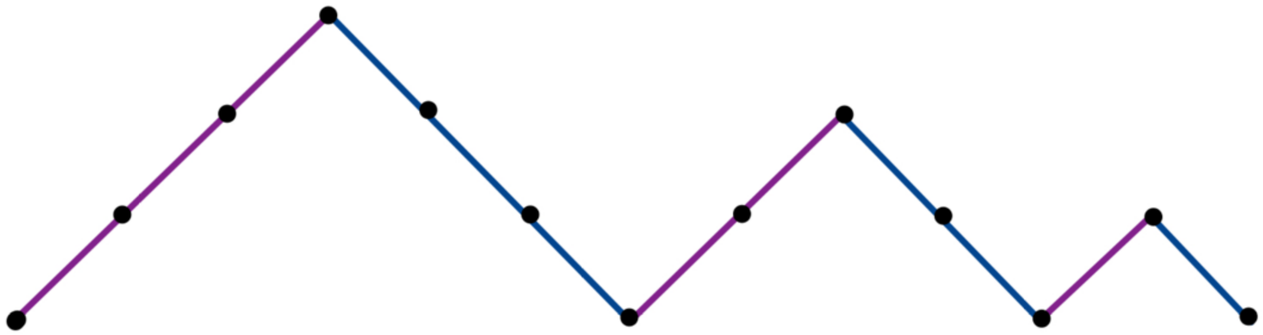


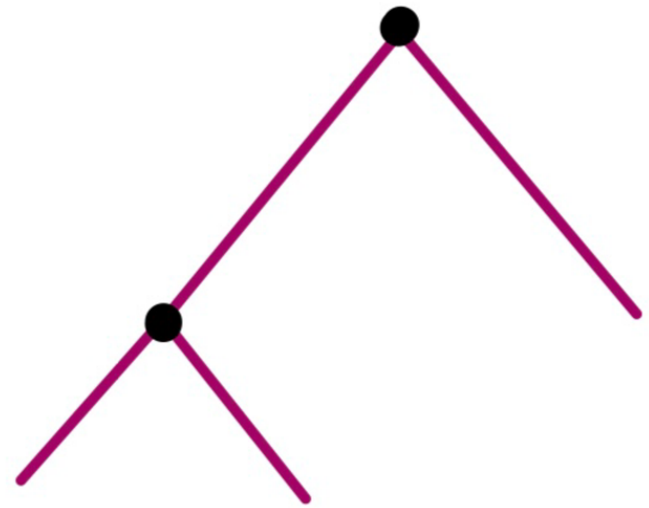
How does the bijection work?



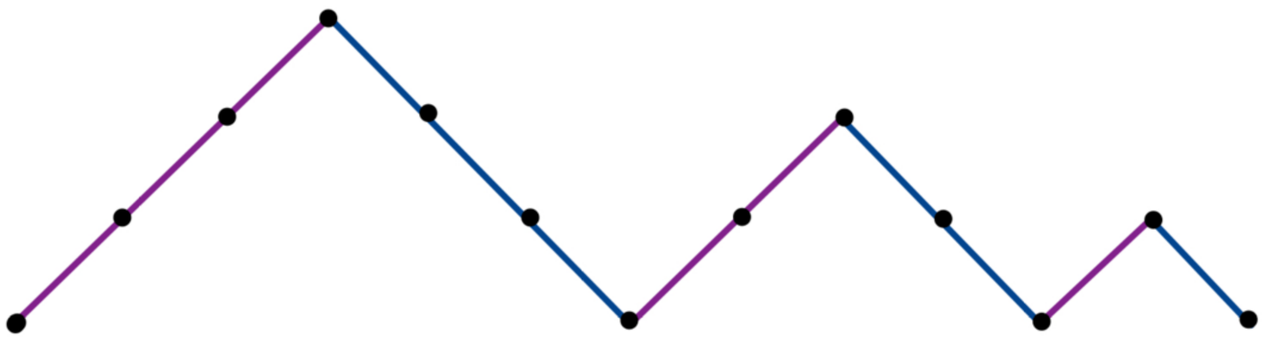


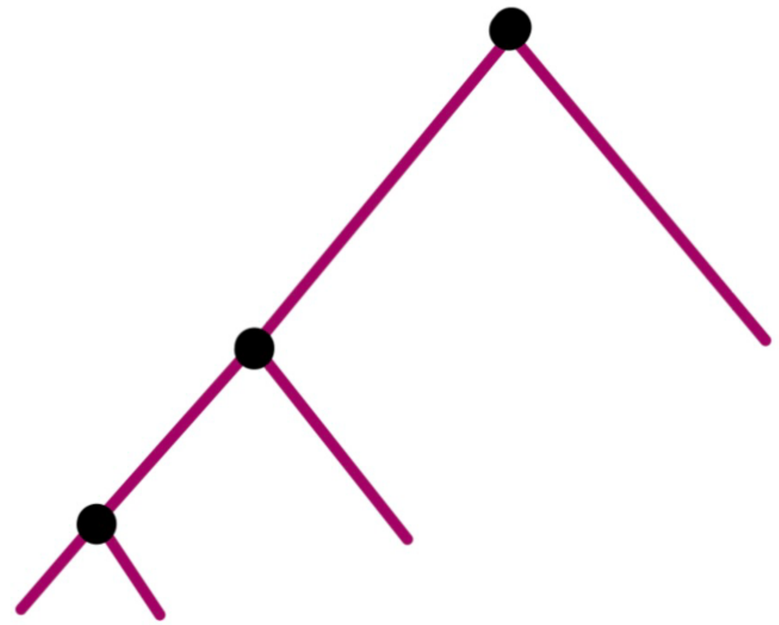
How does the bijection work?



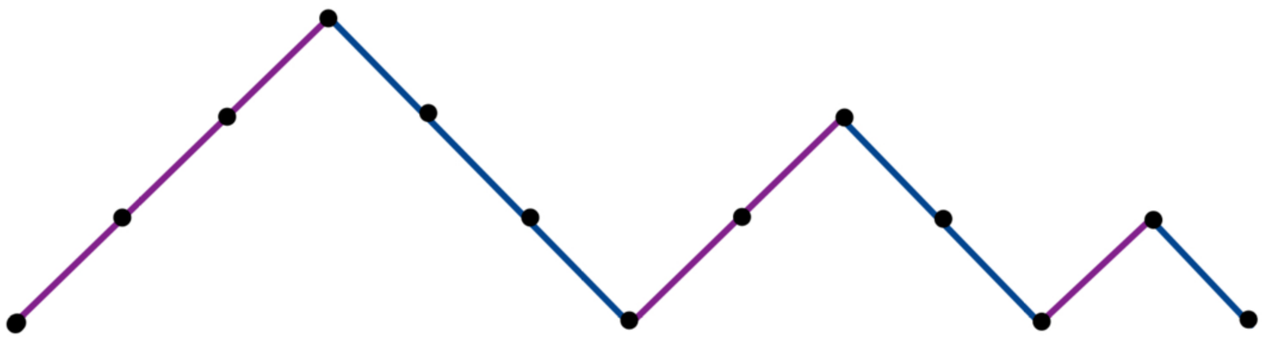


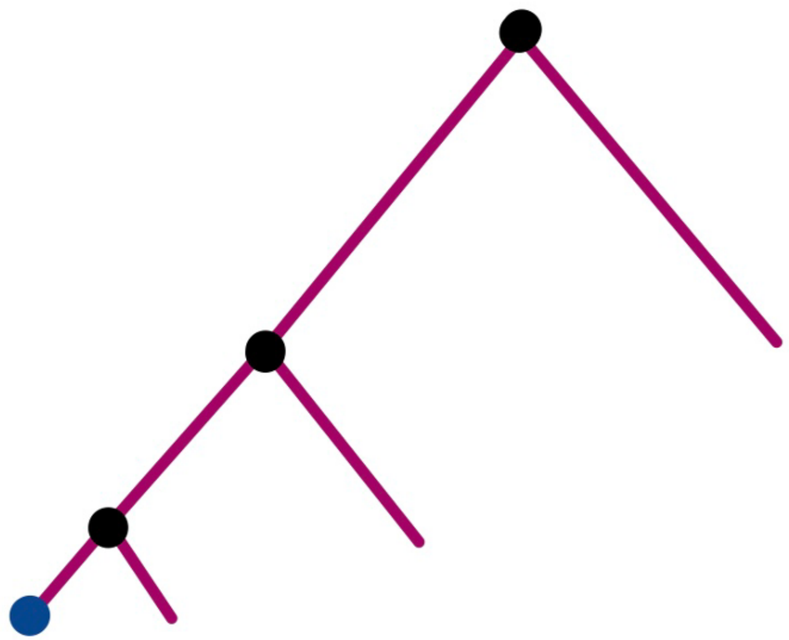
How does the bijection work?



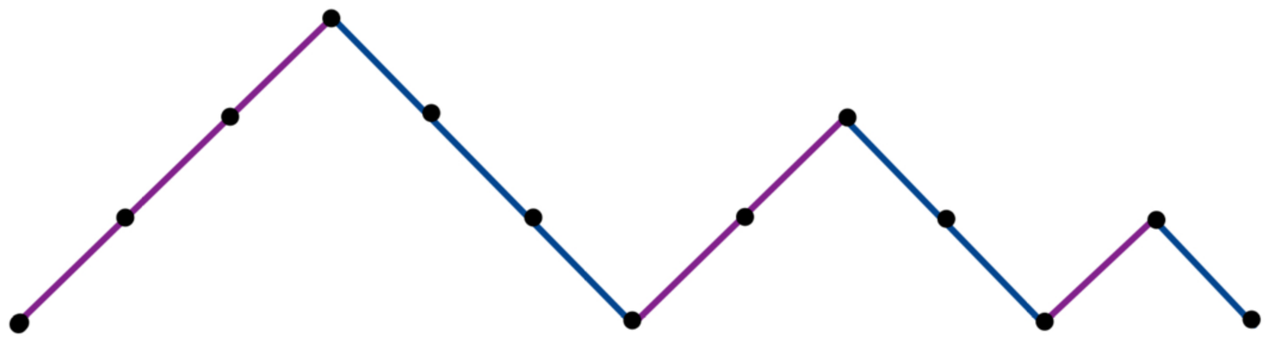


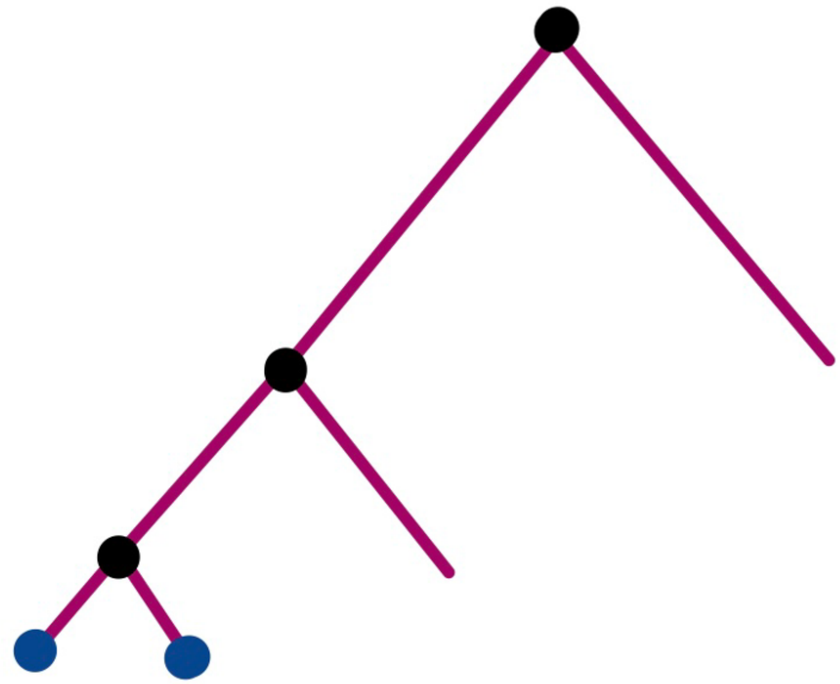
How does the bijection work?



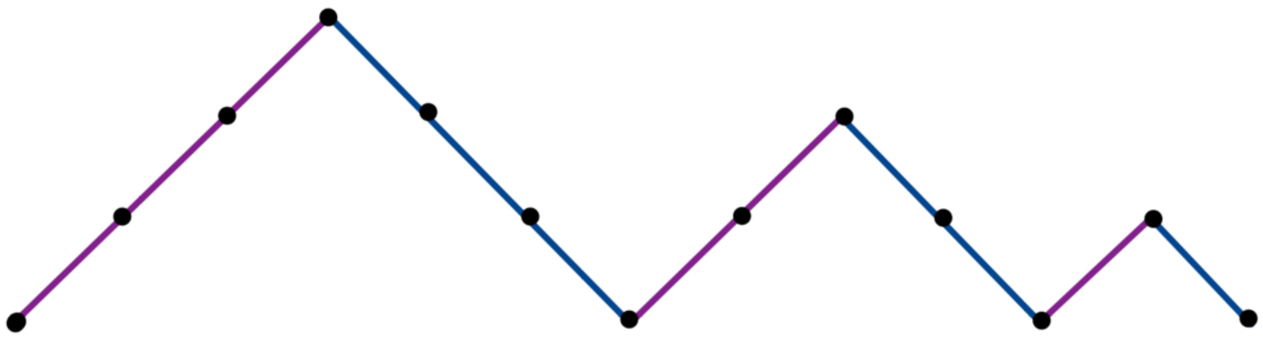


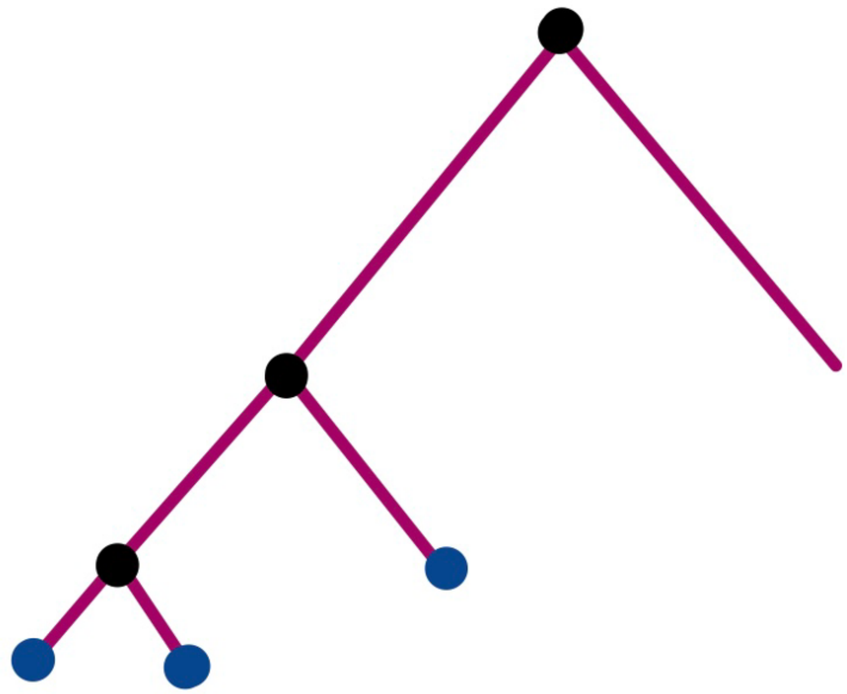
How does the bijection work?



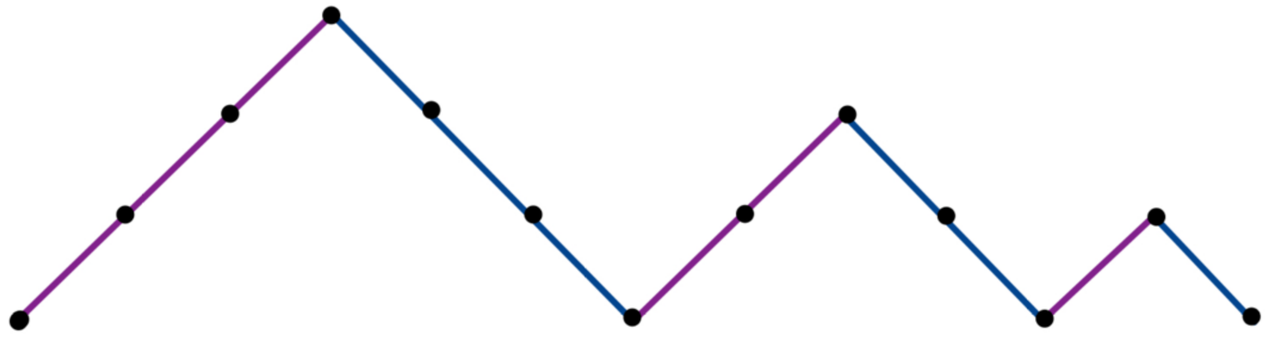


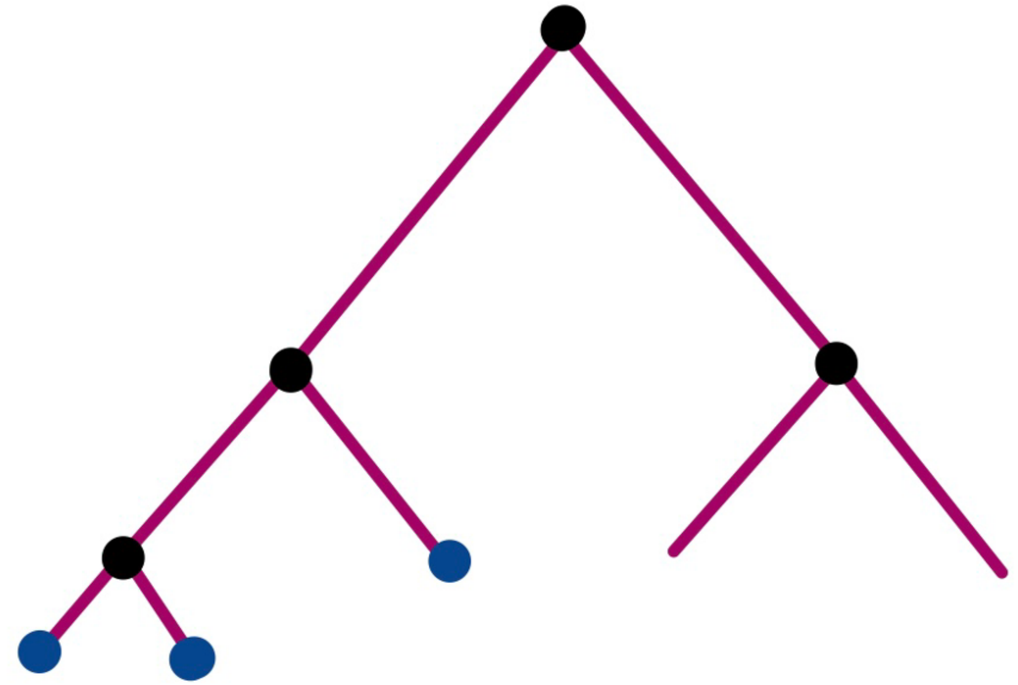
How does the bijection work?



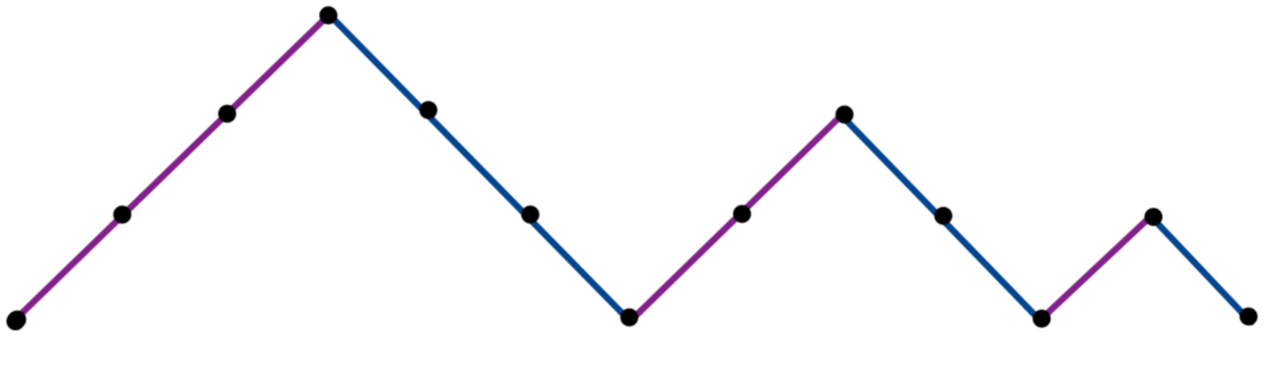


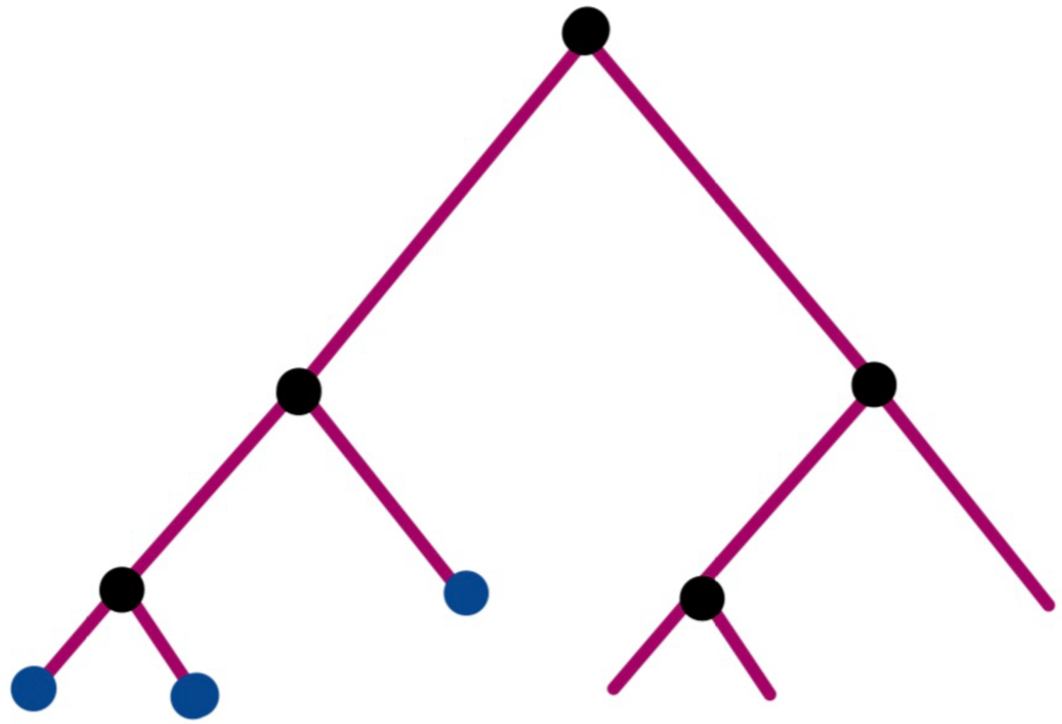
How does the bijection work?



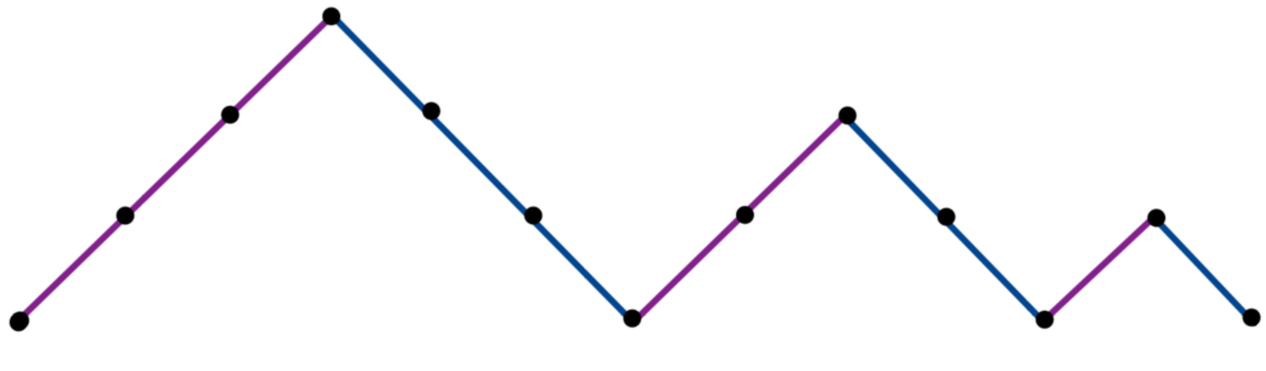


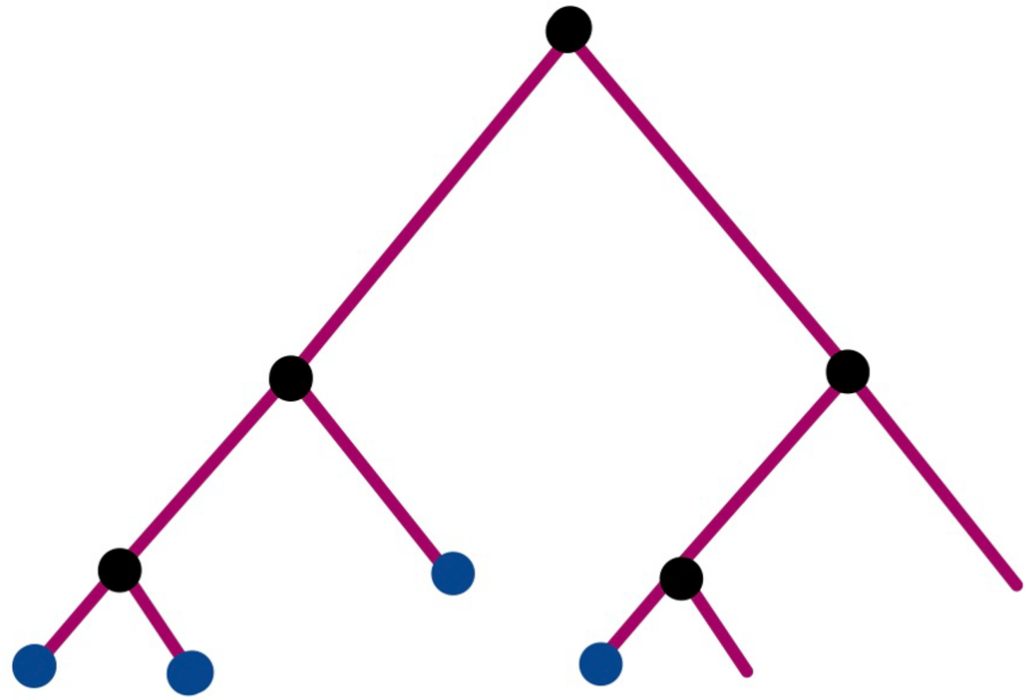
How does the bijection work?



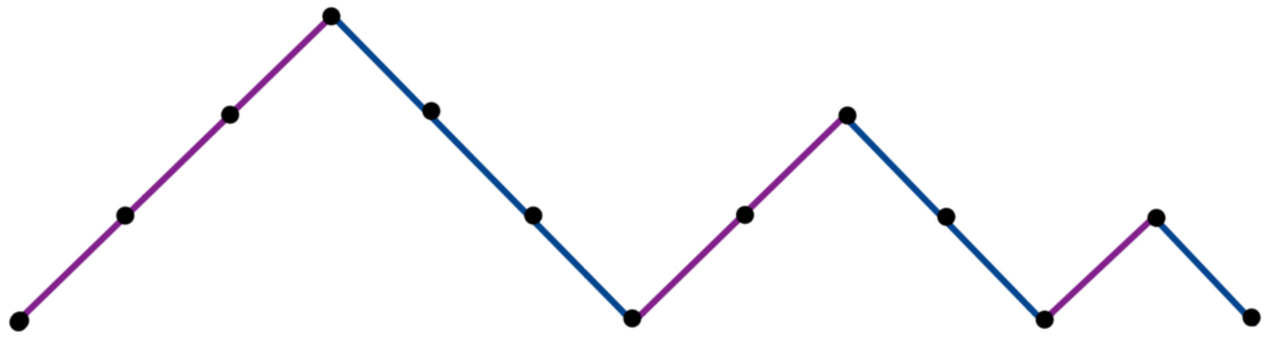


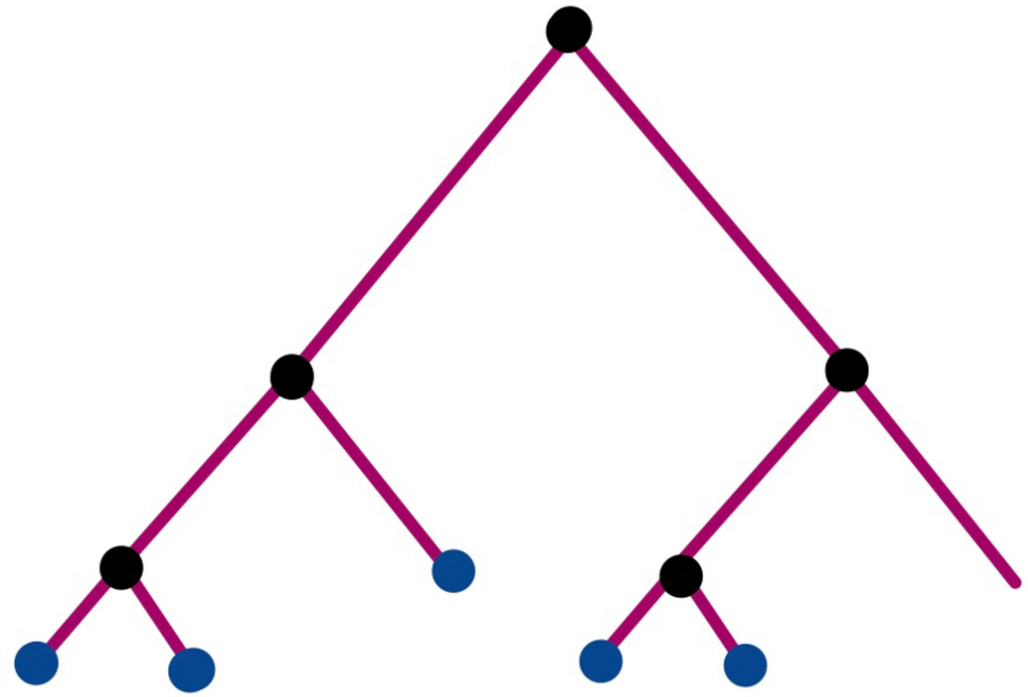
How does the bijection work?



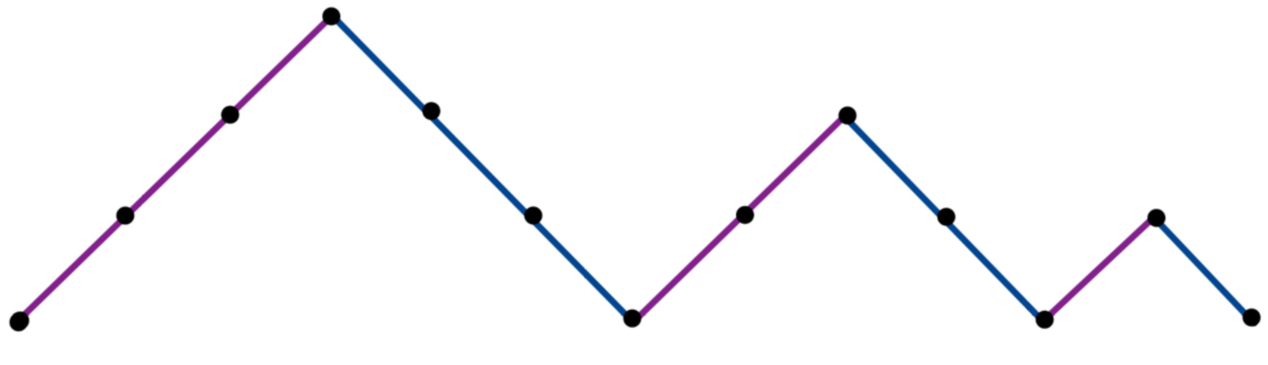


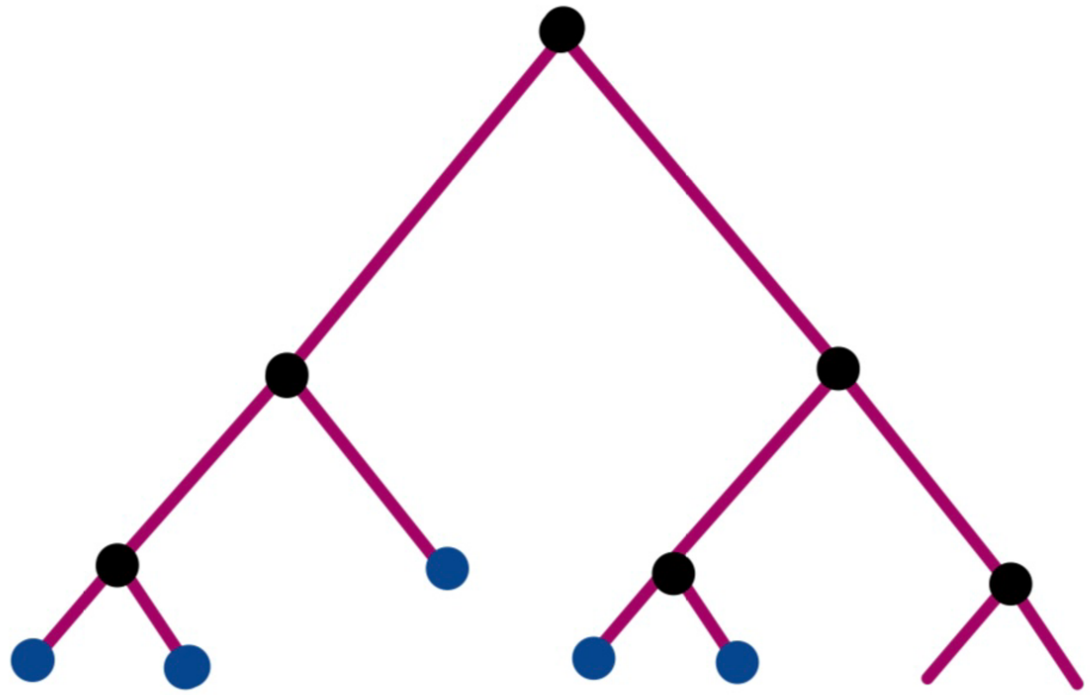
How does the bijection work?



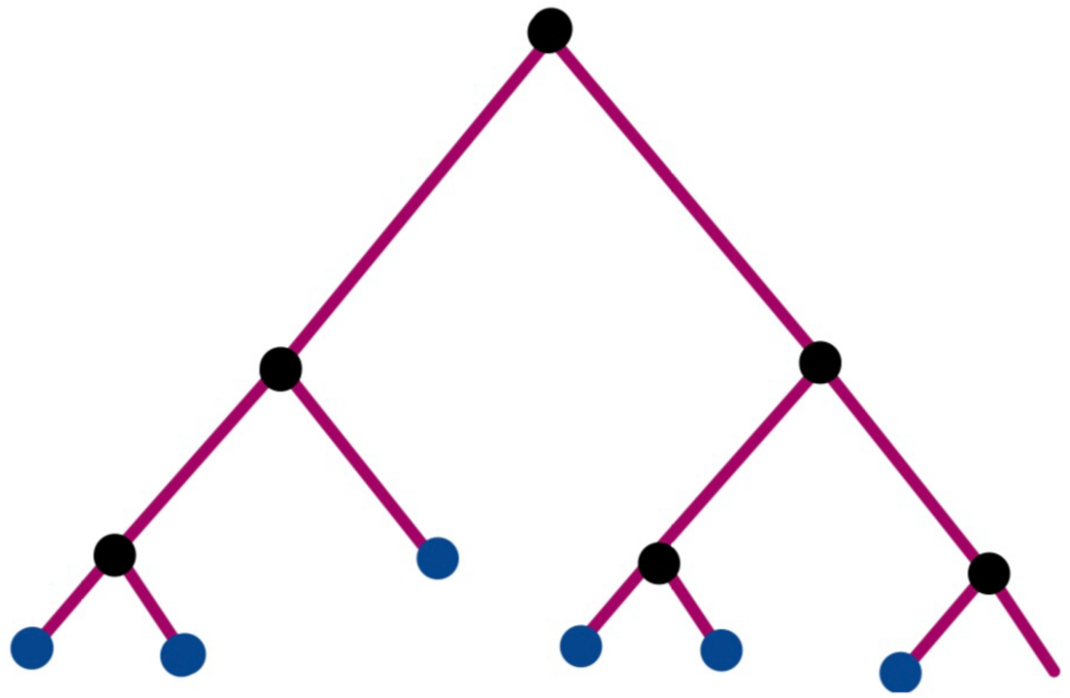


How does the bijection work?

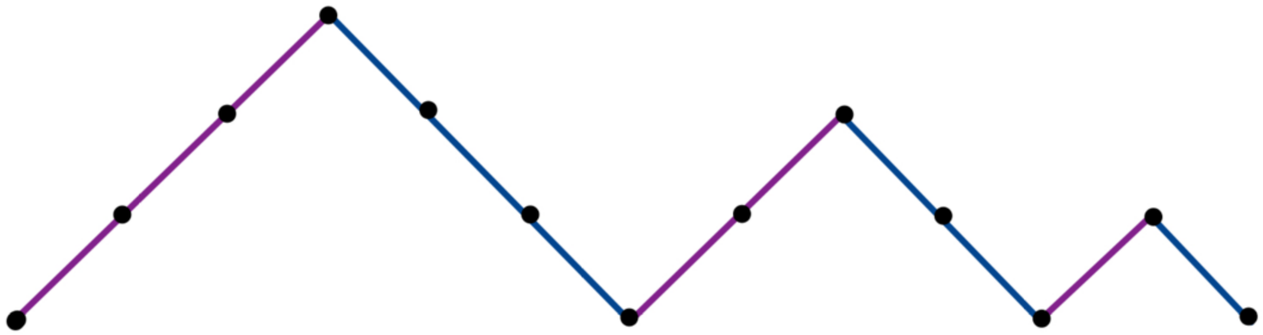


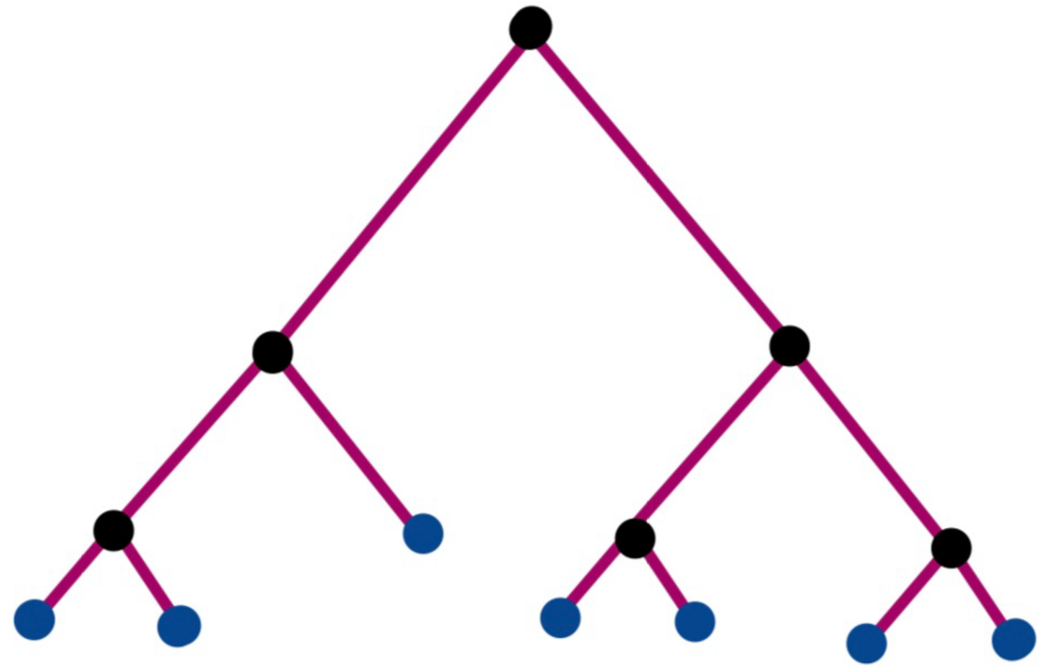


How does the bijection work?



How does the bijection work?





Another example: increasing binary trees

Another example: increasing binary trees

- [**Definition**]: "A single-labelled binary tree (of order n) is a binary tree such that each node is labelled with a number from $[n]$. In particular, a labelled binary tree is increasing if each child has label bigger than its parent."

Another example: increasing binary trees

- [**Definition**]: "A single-labelled binary tree (of order n) is a binary tree such that each node is labelled with a number from $[n]$. In particular, a labelled binary tree is increasing if each child has label bigger than its parent."

- Increasing binary trees on n vertices $\Leftrightarrow$ permutations of $[n]$.

Another example: increasing binary trees

- [**Definition**]: "A single-labelled binary tree (of order n) is a binary tree such that each node is labelled with a number from $[n]$. In particular, a labelled binary tree is increasing if each child has label bigger than its parent."

- Increasing binary trees on n vertices $\Leftrightarrow$ permutations of $[n]$.
- # increasing binary trees on n vertices $= n!$

Another example: increasing binary trees

- [**Definition**]: "A single-labelled binary tree (of order n) is a binary tree such that each node is labelled with a number from $[n]$. In particular, a labelled binary tree is increasing if each child has label bigger than its parent."

- Increasing binary trees on n vertices $\Leftrightarrow$ permutations of $[n]$.
- # increasing binary trees on n vertices = $n!$
- Continued fraction expansion (Euler 1760):

Another example: increasing binary trees

- [**Definition**]: "A single-labelled binary tree (of order n) is a binary tree such that each node is labelled with a number from $[n]$. In particular, a labelled binary tree is increasing if each child has label bigger than its parent."

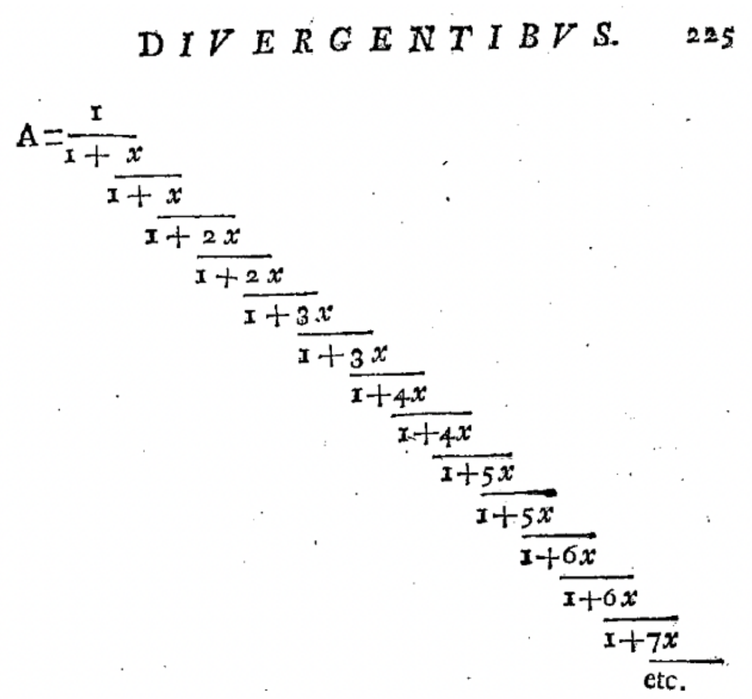
- Increasing binary trees on n vertices $\Leftrightarrow$ permutations of $[n]$.
- # increasing binary trees on n vertices $= n!$
- Continued fraction expansion (Euler 1760):

$$\sum_{n=0}^{\infty} n!\, t^n = \cfrac{1}{1 - \cfrac{1t}{1 - \cfrac{1t}{1 - \cfrac{2t}{1 - \cfrac{2t}{1 - \ldots.}}}}}$$

Another example: increasing binary trees

- [**Definition**]: "A single-labelled binary tree (of order n) is a binary tree such that each node is labelled with a number from $[n]$. In particular, a labelled binary tree is increasing if each child has label bigger than its parent."

- Increasing binary trees on n vertices $\Longleftrightarrow$ permutations of $[n]$.
- # increasing binary trees on n vertices $= n!$
- Continued fraction expansion (Euler 1760):



DIVERGENTIBVS. 225

$$A = \cfrac{1}{1+\cfrac{x}{1+\cfrac{x}{1+\cfrac{2x}{1+\cfrac{2x}{1+\cfrac{3x}{1+\cfrac{3x}{1+\cfrac{4x}{1+\cfrac{4x}{1+\cfrac{5x}{1+\cfrac{5x}{1+\cfrac{6x}{1+\cfrac{6x}{1+\cfrac{7x}{\text{etc.}}}}}}}}}}}}}}$$

§. 22. Quemadmodum autem huiusmodi fractionum continuarum valor sit inuestigandus, alibi ostendi: Scilicet cum singulorum denominatorum partes integrae sint vnitates, soli numeratores in computum veniunt; sit ergo $x = 1$, atque inuestigatio summae A sequenti modo instituetur:

How does the bijection work?

How does the bijection work?

- [**Definition**]: “A permutation is said to be written in standard representation if

How does the bijection work?

- [**Definition**]: "A permutation is said to be written in standard representation if
  - each cycle is written with its largest element first;

How does the bijection work?

- [**Definition**]: "A permutation is said to be written in standard representation if
  - each cycle is written with its largest element first;
  - the cycles are written in increasing order of their largest element."

How does the bijection work?

- [**Definition**]: "A permutation is said to be written in standard representation if
  - each cycle is written with its largest element first;
  - the cycles are written in increasing order of their largest element."

- (14)(2)(375)(6) becomes (2)(41)(6)(753)

How does the bijection work?

- [**Definition**]: "A permutation is said to be written in standard representation if
  - each cycle is written with its largest element first;
  - the cycles are written in increasing order of their largest element."

- (14)(2)(375)(6) becomes (2)(41)(6)(753)

- [**Definition**]: "Let $\omega \in \sigma_n$. The we define with $\widehat{\omega}$ the permutation obtained from ω by writing it in standard form and erasing the parenthesis."

How does the bijection work?

- [**Definition**]: "A permutation is said to be written in standard representation if
  - each cycle is written with its largest element first;
  - the cycles are written in increasing order of their largest element."

- (14)(2)(375)(6) becomes (2)(41)(6)(753)

- [**Definition**]: "Let $\omega \in \sigma_n$. The we define with $\widehat{\omega}$ the permutation obtained from ω by writing it in standard form and erasing the parenthesis."

- Remark! The above is actually a bijection. Given $\widehat{\omega}$ we get back ω proceeding every *left-to-right* maximum.

How does the bijection work?

How does the bijection work?

- permutation => tree: done recursively:

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

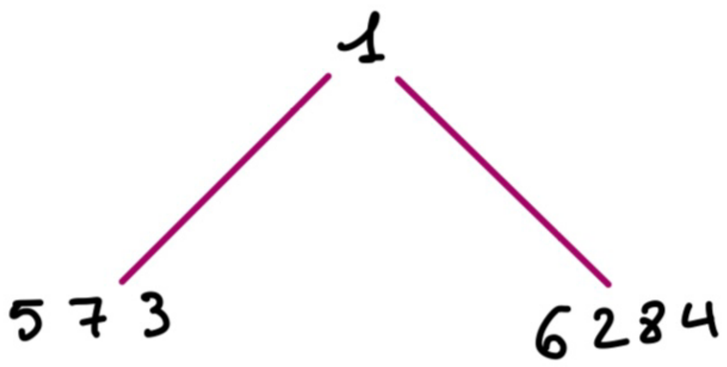
5 7 3 1 6 2 8 4

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

5 7 3 1 6 2 8 4



How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

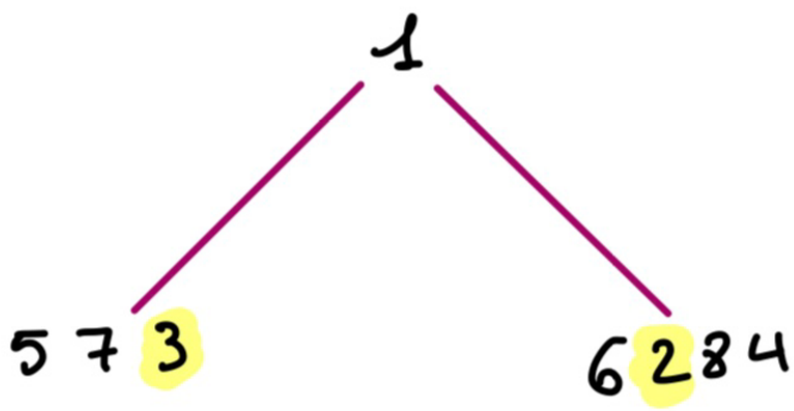
5 7 3 1 6 2 8 4

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

5 7 3 1 6 2 8 4



How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

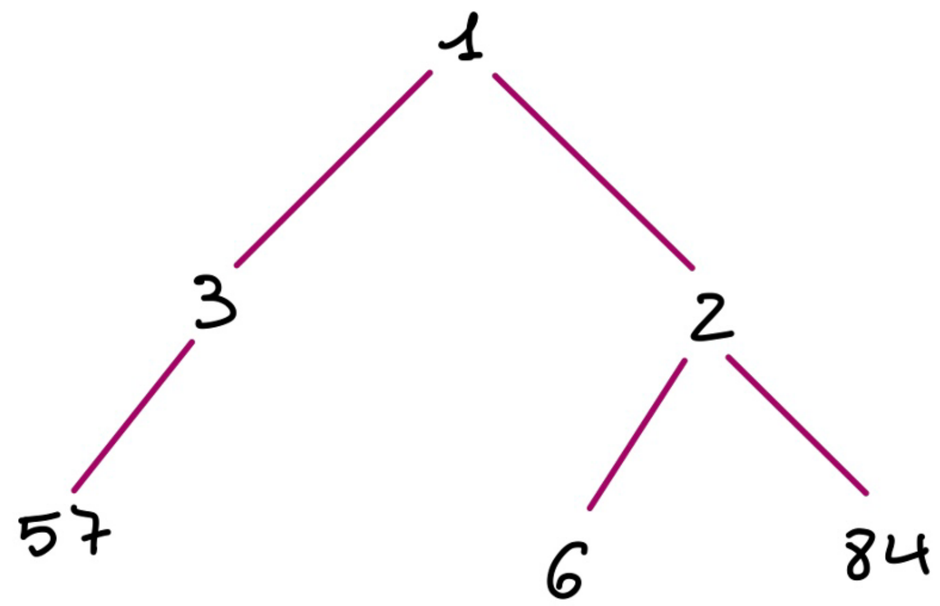
5 7 3 1 6 2 8 4

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

5 7 3 1 6 2 8 4



How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

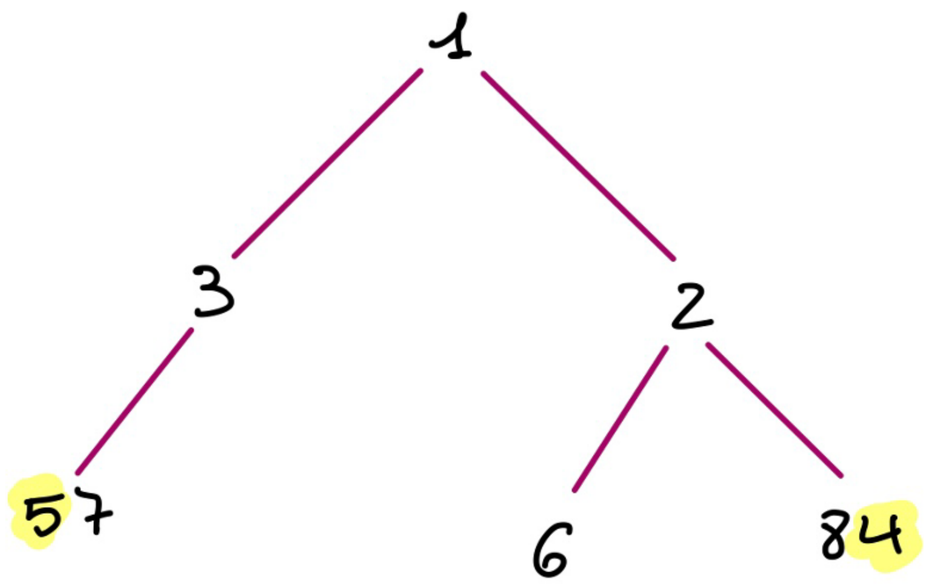
5 7 3 1 6 2 8 4

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

5 7 3 1 6 2 8 4



How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

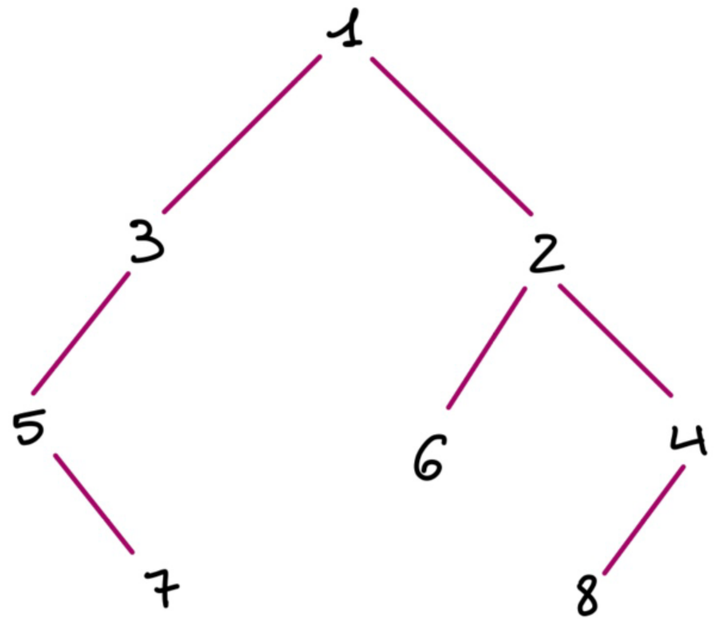
5 7 3 1 6 2 8 4

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

5 7 3 1 6 2 8 4



How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

5 7 3 1 6 2 8 4

How does the bijection work?

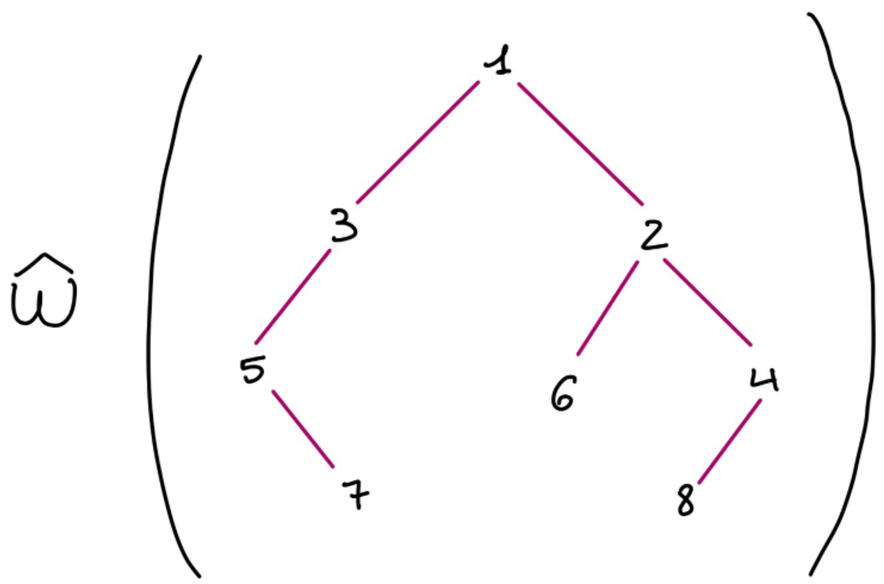
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

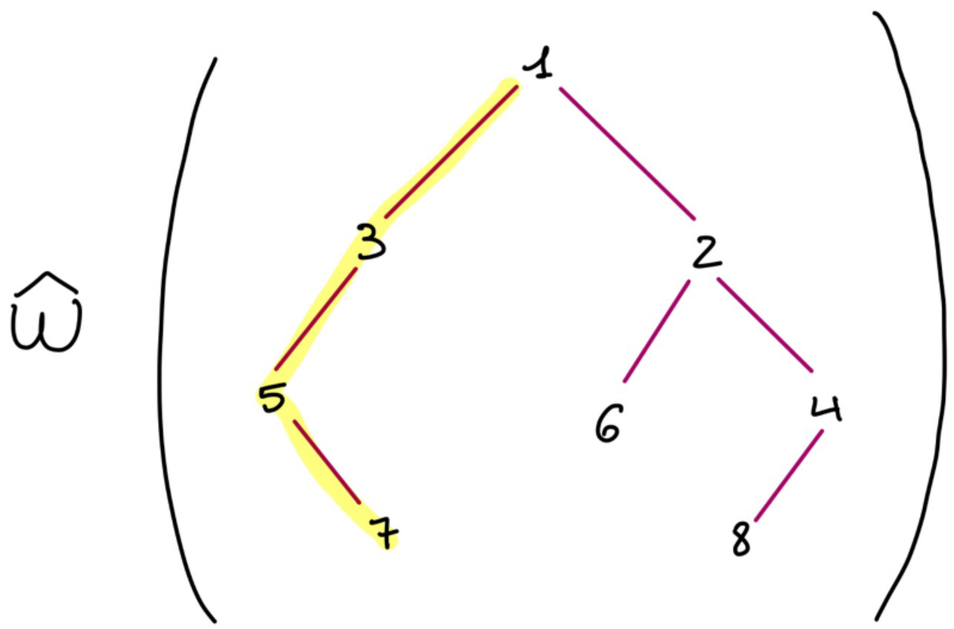
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

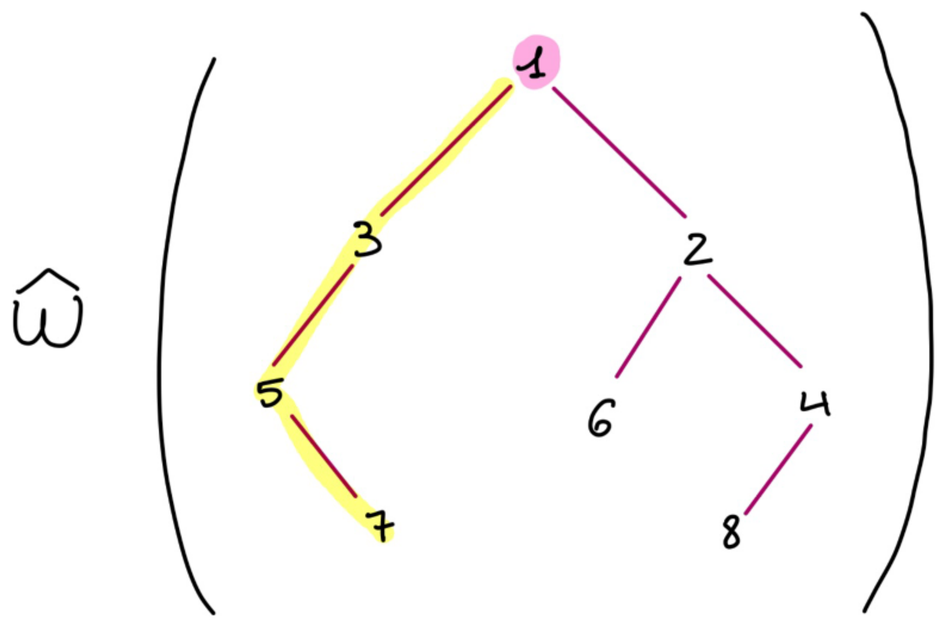
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

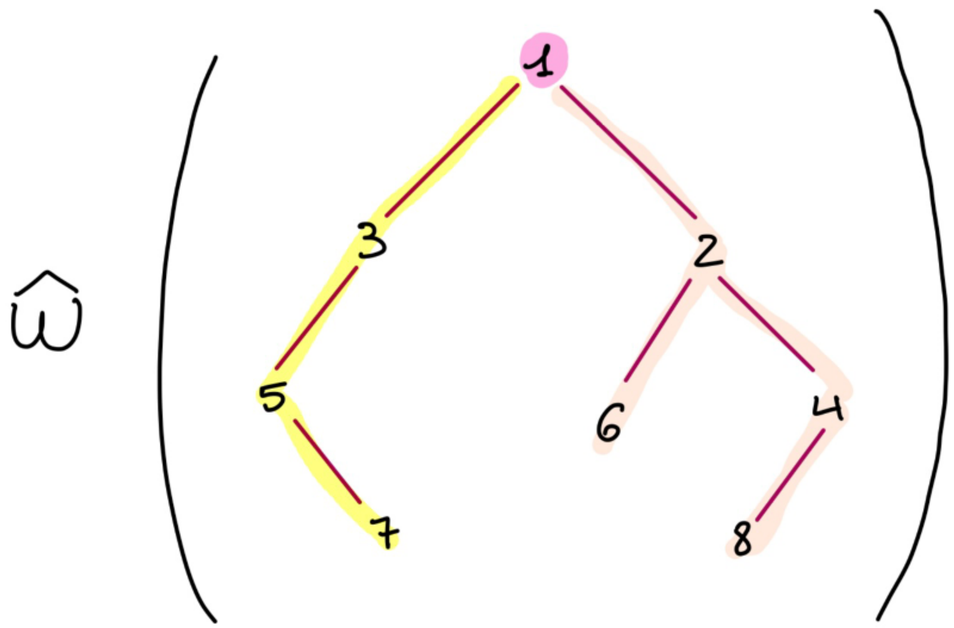
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

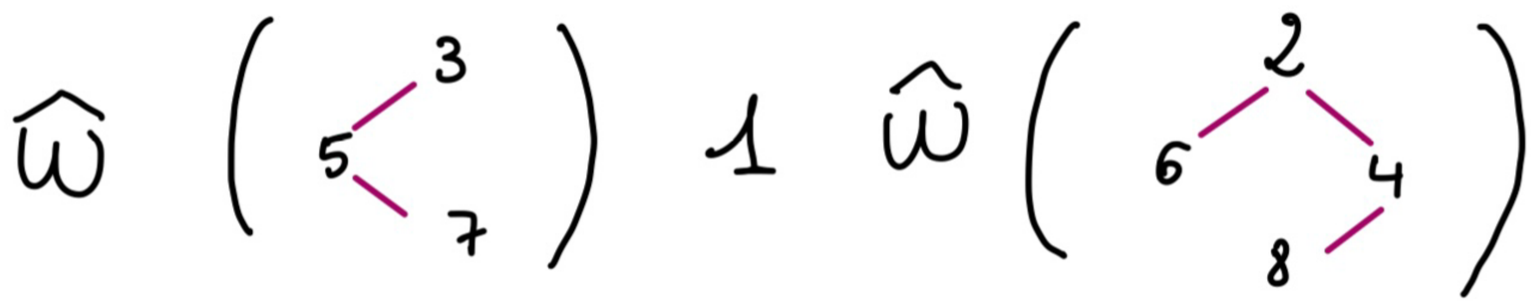
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

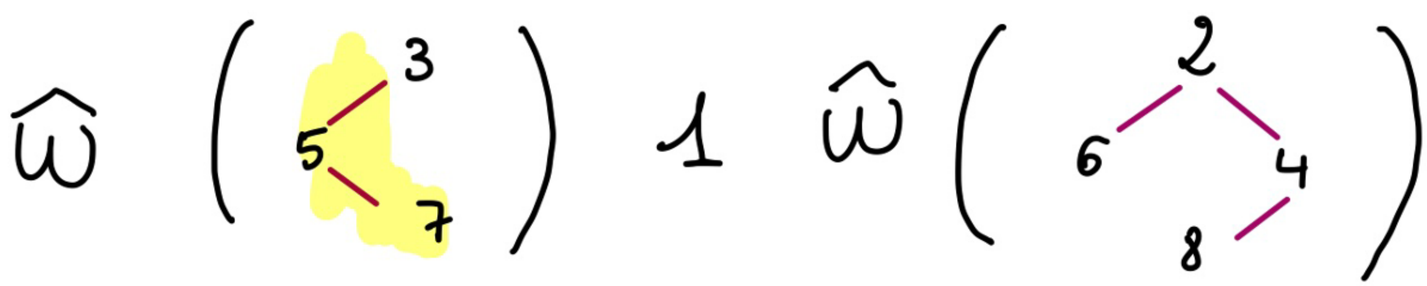
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

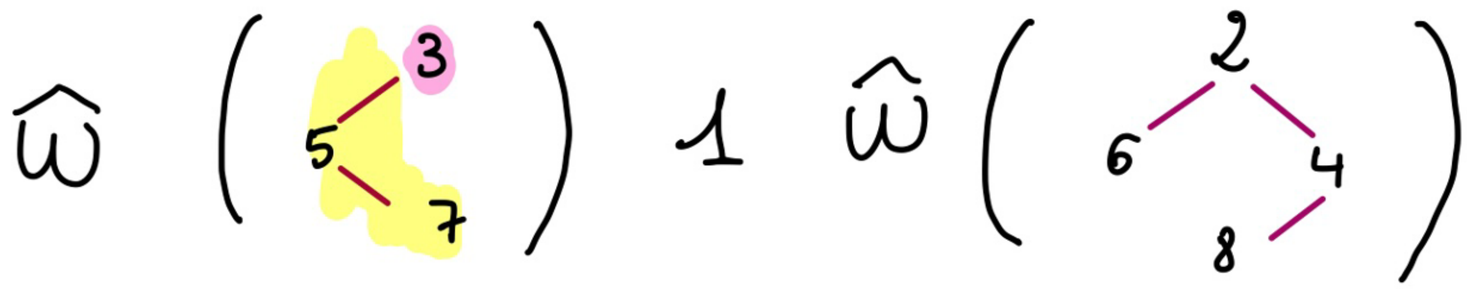
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

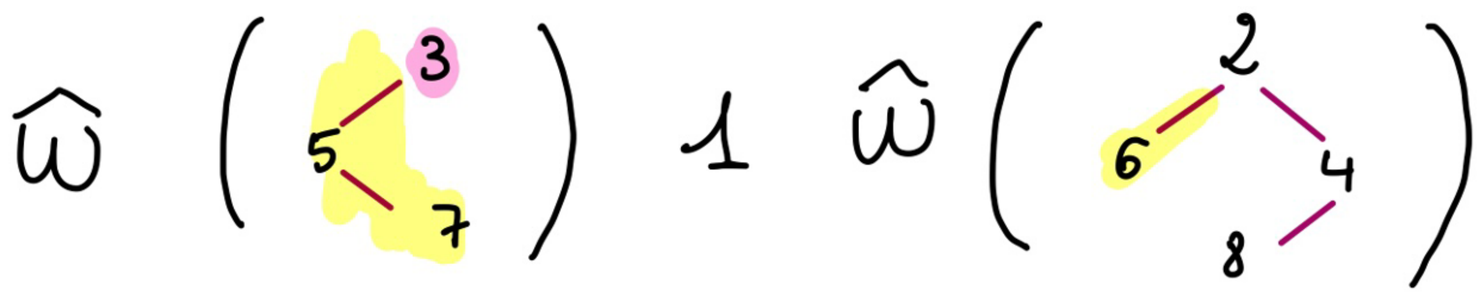
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

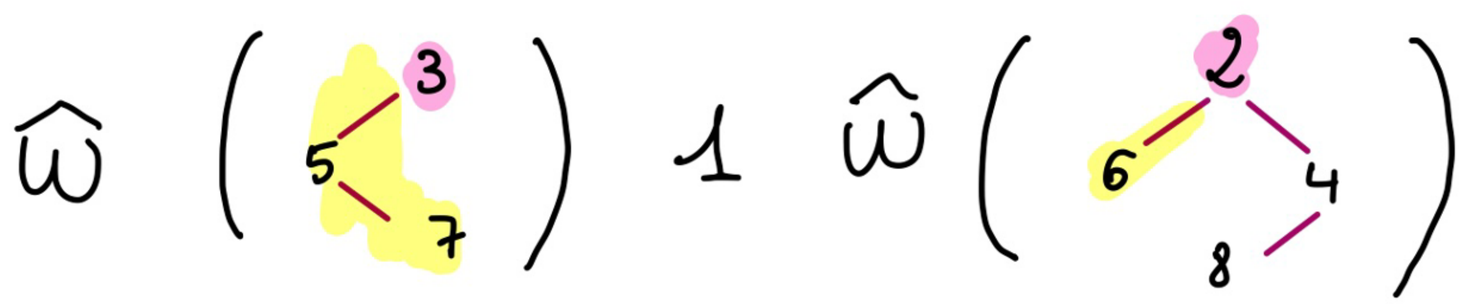
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

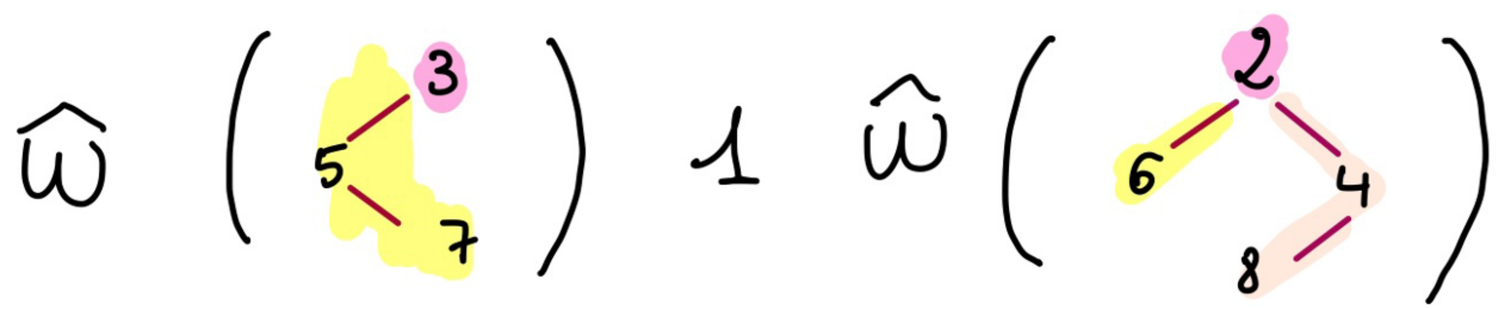
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

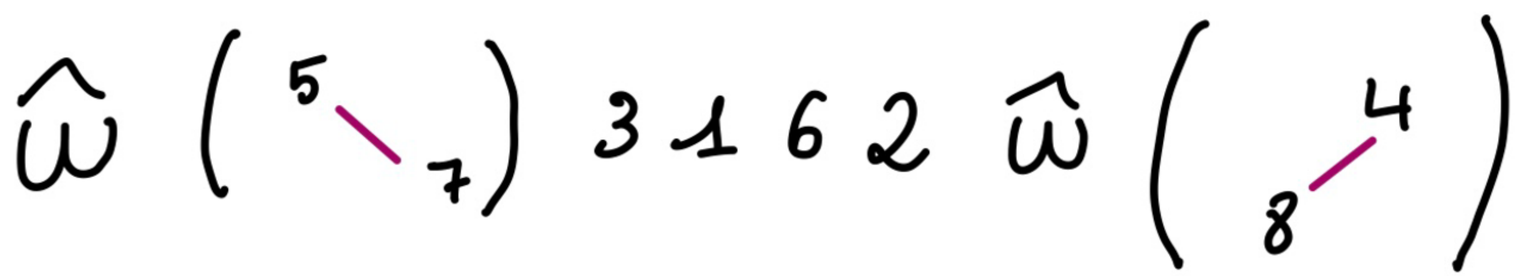
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

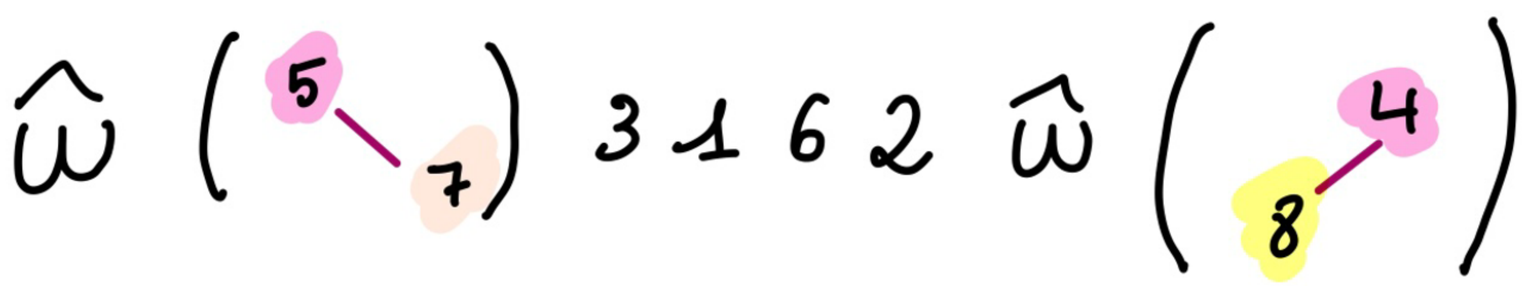
- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).



How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

How does the bijection work?

- permutation => tree: done recursively:
  - $\hat{\omega} = \emptyset$ => $T(\hat{\omega})=\emptyset$
  - $\hat{\omega} \neq \emptyset$. i least element of $\hat{\omega}$. Write $\hat{\omega}=uiv$ =>
    i root of $T(\hat{\omega})$, T(u) and T(v) are the left and right subtree

- Tree => permutation: read the labels of the tree in symmetric order (LNR).

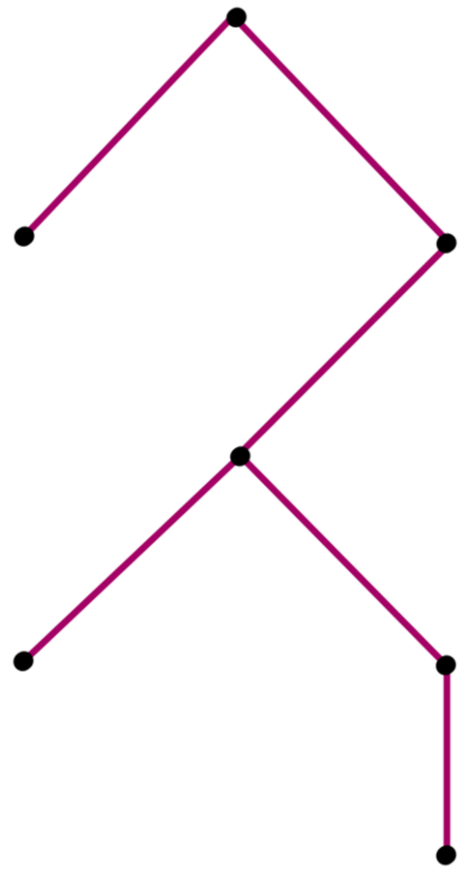
Our Conjecture

Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.

Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.



Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.

Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.
- Example of *experimental mathematics:*

Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.
- Example of *experimental mathematics:*
  - OEIS (https://oeis.org)

Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.
- Example of *experimental mathematics:*
  - OEIS (https://oeis.org)
  - Mathematica

Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.
- Example of *experimental mathematics:*
  - OEIS (https://oeis.org)
  - Mathematica
- We started with the T-fraction and asked what sequence it represents.

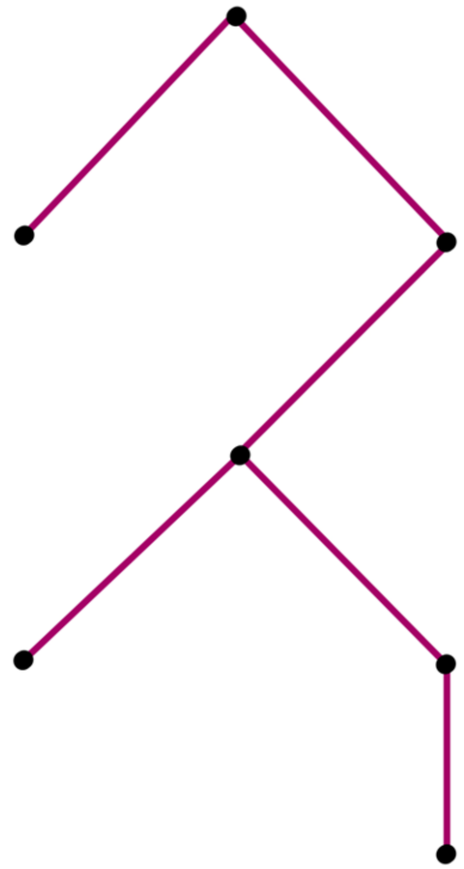
Our Conjecture

- We look at a particular type of single-labelled increasing ternary trees: each parent can have a left child, a right child and a middle child s.t. a middle child does not have any siblings.
- Example of *experimental mathematics:*
  - OEIS (https://oeis.org)
  - Mathematica

```
In[•]:= MyT2 = Select[Tuples[{0, 1}, 6], (! (#[[3]] == #[[4]] == #[[5]] == #[[6]] == 0)) &];

In[•]:= Do[Print["nn=", nn, " ", MyT2[[nn]], " ", (Spec2 @@ MyT2[[nn]])], {nn, 1, 60}]


In[•]:= Do[Print["nn=", nn, " ", MyT2[[nn]], " ",
      SequenceInOEIS2[(Spec2 @@ MyT2[[nn]])]], {nn, 1, 60}]

nn=50 {1, 1, 0, 1, 0, 1} {A230008} (**)
```

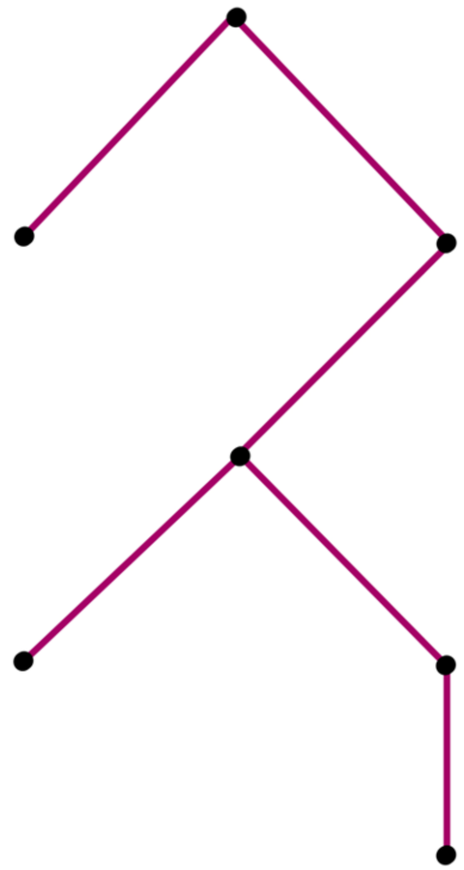
- We started with the T-fraction and asked what sequence it represents.

Our Conjecture



Our Conjecture

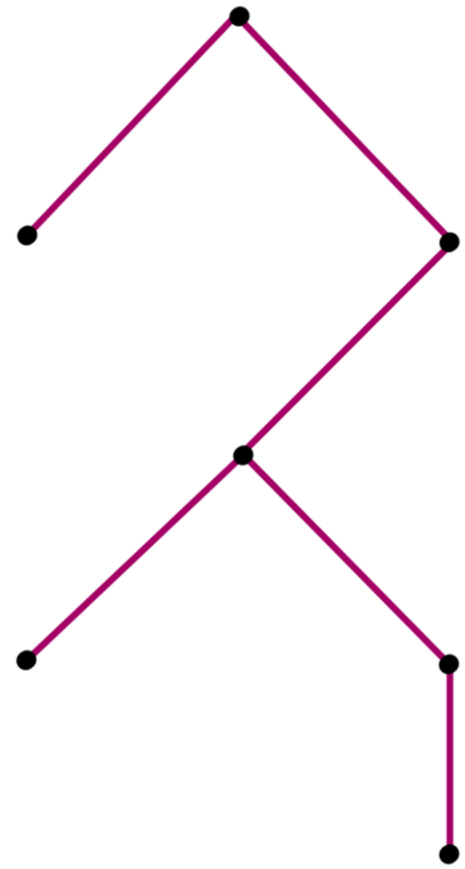
- We have shown that such type of trees are enumerated by OEIS A230008: 1, 1, 3, 11, 51, 295, 2055, 16715, ….



Our Conjecture

- We have shown that such type of trees are enumerated by OEIS A230008: 1, 1, 3, 11, 51, 295, 2055, 16715, ….

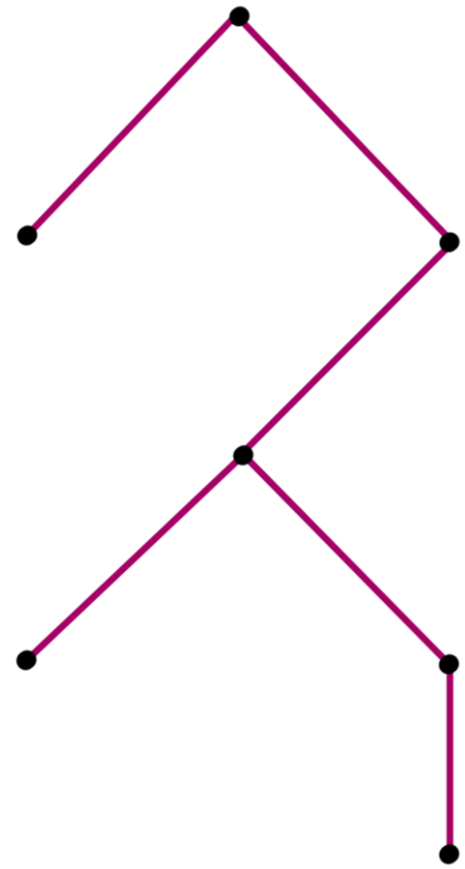
- We are now working on the bijection between this type of trees and Schröder Path in order to prove T-fraction:



Our Conjecture

- We have shown that such type of trees are enumerated by OEIS A230008: 1, 1, 3, 11, 51, 295, 2055, 16715, ….

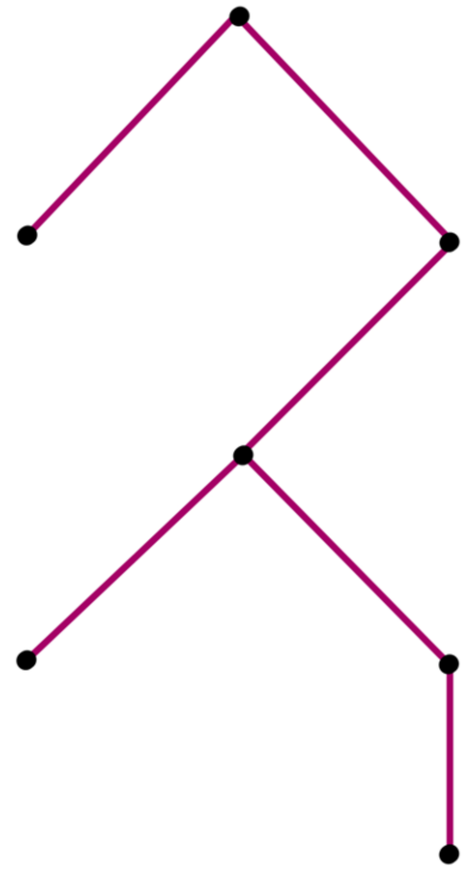
- We are now working on the bijection between this type of trees and Schröder Path in order to prove T-fraction:
  - $\boldsymbol{\alpha} = 1, 1, 2, 2, 3, 3, 4, 4, \ldots$.



Our Conjecture

- We have shown that such type of trees are enumerated by OEIS A230008: 1, 1, 3, 11, 51, 295, 2055, 16715, ….

- We are now working on the bijection between this type of trees and Schröder Path in order to prove T-fraction:
  - $\boldsymbol{\alpha} = 1, 1, 2, 2, 3, 3, 4, 4, \ldots$.
  - $\boldsymbol{\delta} = 0, 1, 0, 2, 0, 3, 0, 4, \ldots$.

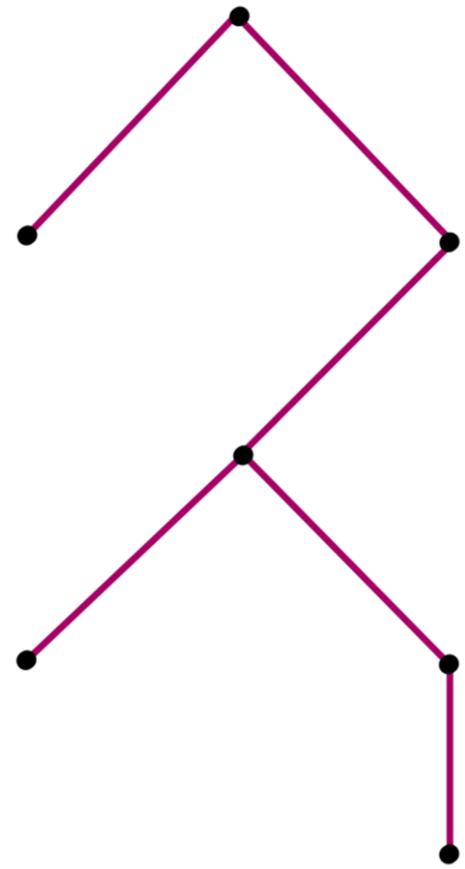


Our Conjecture

- We have shown that such type of trees are enumerated by OEIS A230008: 1, 1, 3, 11, 51, 295, 2055, 16715, ….

- We are now working on the bijection between this type of trees and Schröder Path in order to prove T-fraction:
  - $\boldsymbol{\alpha} = 1, 1, 2, 2, 3, 3, 4, 4, \ldots.$
  - $\boldsymbol{\delta} = 0, 1, 0, 2, 0, 3, 0, 4, \ldots.$

- Can we generalize this to multilabelled ternary trees?



Bibliography

1. Deb, B. (2023). Bijection between increasing binary trees and rook placements on double staircases. *The Electronic Journal of Combinatorics*, *30*(1). https://doi.org/10.37236/10926

2. Deb, B., & Sokal, A. D. (2022). Classical continued fractions for some multivariate polynomials generalizing the genocchi and median genocchi numbers.

3. Chen, X., & Sokal, A. D. (2023). Total positivity of some polynomial matrices that enumerate labeled trees and forests. ii. rooted labeled trees and partial functional digraphs. Elvey Price, A., & Sokal, A. D. (2020). Phylogenetic trees, augmented perfect matchings, and a thron-type continued fraction (t-fraction) for the ward polynomials. *The Electronic Journal of Combinatorics*, *27*(4). https://doi.org/10.37236/9571

4. Flajolet, P. (1980). *Combinatorial aspects of continued fractions*. https://doi.org/10.1016/0012365X(80)90050-3

5. Flajolet, P., & Sedgewick, R. (2009). *Analytic combinatorics*. Cambridge University Press. https: //doi.org/10.1017/CBO9780511801655

6. Kuba, M., & Panholzer, A. (2014). Combinatorial families of multilabelled increasing trees and hook-length formulas.

7. Pétréolle, M., & Sokal, A. D. (2021). Lattice paths and branched continued fractions ii. multivariate lah polynomials and lah symmetric functions. *European Journal of Combinatorics*, *92,* 103235. https://doi.org/10.1016/j.ejc.2020.103235

8. Pétréolle, M., Sokal, A. D., & Zhu, B.-X. (2020). Lattice paths and branched continued fractions: An infinite sequence of generalizations of the stieltjes–rogers and thron–rogers polynomials, with coefficientwise hankel-total positivity.

9. Sokal, A. D. (2023). A simple algorithm for expanding a power series as a continued fraction. *Expositiones Mathematicae*, *41*(2), 245–287. https://doi.org/10.1016/j.exmath.2022.12.001

10. Sokal, A. D., & Zeng, J. (2022). Some multivariate master polynomials for permutations, set partitions, and perfect matchings, and their continued fractions. *Advances in Applied Mathematics*, *138*, 102341. https://doi.org/10.1016/j.aam.2022.102341

11. Stanley, R. P. (2011). *Enumerative combinatorics* (2nd ed., Vol. 1). Cambridge University Press. https://doi.org/10.1017/CBO9781139058520

12. Stanley, R. P., & Fomin, S. (1999). *Enumerative combinatorics* (Vol. 2). Cambridge University Press. https://doi.org/10.1017/CBO9780511609589

13. Wilf, H. S. (2006). *Generatingfunctionology*. A. K. Peters, Ltd.

14. https://www.viennot.org/abjc1-bijections.html

veronica.bitonti.21@ucl.ac.uk

UCL

Thank You!