

# Algorithmique des structures arborescentes

L2 Info et Math-info, 2018–19

Marc Zeitoun

21 mars 2019

# Organisation



- ▶ Aujourd'hui = avant-dernier cours en amphi.
- ▶ DS le jeudi 28/03/2019.
- ▶ Les groupes seront annoncés sous Moodle.

# Plan

Retour sur les parcours d'arbres

Arbres planaires

Énumération et bijections

# Parcours en profondeur préfixe

Écriture naturelle de façon récursive.

**DFS**(t) :

**si** t est vide **alors**

Retourner

**sinon**

Traiter(t.racine)

DFS(t.fils\_gauche)

DFS(t.fils\_droit)

**fin si**

# Parcours en profondeur préfixe

Écriture naturelle de façon récursive.

**DFS**(t) :

**si** t est vide **alors**

Retourner

**sinon**

Traiter(t.racine)

DFS(t.fils\_gauche)

DFS(t.fils\_droit)

**fin si**

Comment adapter l'algorithme pour

- ▶ générer la **liste** des nœuds dans l'ordre préfixe,
- ▶ en complexité  $O(n)$ , où  $n$  = nombre de nœuds ?

# Parcours en profondeur préfixe

Écriture naturelle de façon récursive.

**DFS**(t) :

si t est vide **alors**

Retourner

**sinon**

Traiter(t.racine)

DFS(t.fils\_gauche)

DFS(t.fils\_droit)

**fin si**

Comment adapter l'algorithme pour

- ▶ générer la **liste** des nœuds dans l'ordre préfixe,
- ▶ en complexité  $O(n)$ , où  $n$  = nombre de nœuds ?

**À éviter** : Concaténations de listes  $\rightsquigarrow O(n^2)$ .

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\leadsto$  concaténation de listes en  $O(1)$ .

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\leadsto$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.



# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**



1

Push(1)

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**



Push(1) Push(2)

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**



Push(1) Push(2) Push(3)

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\leadsto$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**



Push(1) Push(2) Push(3) Pop

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**

1

Push(1) Push(2) Push(3) Pop Pop

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\leadsto$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**



Push(1) Push(2) Push(3) Pop Pop Push(4)

# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**

1

Push(1) Push(2) Push(3) Pop Pop Push(4) Pop



# Parcours en profondeur préfixe

- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\rightsquigarrow$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**

Push(1) Push(2) Push(3) Pop Pop Push(4) Pop Pop

# Parcours en profondeur préfixe

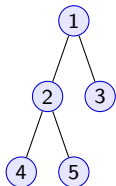
- ▶ 1<sup>re</sup> solution : utiliser des listes doublement chaînées.  
     $\leadsto$  concaténation de listes en  $O(1)$ .
- ▶ 2<sup>e</sup> solution : maintenir les sous-arbres non traités dans **une pile**.

**Rappel sur les piles. 2 opérations : Push(x) et Pop**

**Pour le parcours, ce sont des sous-arbres qu'on mémorise**

# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)



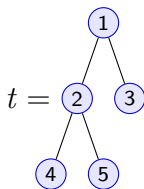
Accumulateur

Actions

```
t = Pop();  
Accumuler(t.racine);  
Push(t.right);  
Push(t.left);
```

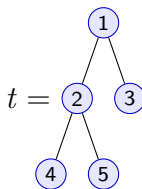
# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
		<pre>t = Pop(); Accumuler(t.racine); Push(t.right); Push(t.left);</pre>



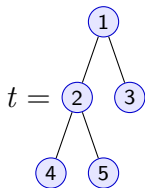
# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
	<b>1</b>	<b>Accumuler(t.racine);</b> Push(t.right); Push(t.left);



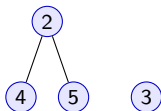
# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p style="text-align: center;">③</p>	<p style="text-align: center;">1</p>	<p style="text-align: center;">Push(t.right); Push(t.left);</p>



# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)

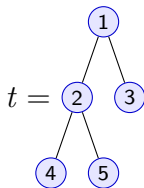


Accumulateur

1

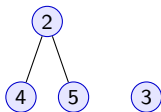
Actions

Push(t.left);



# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)



Accumulateur

1

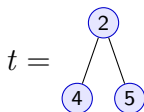
Actions

```
t = Pop();  
Accumuler(t.racine);  
Push(t.right);  
Push(t.left);
```



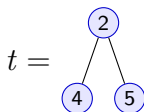
# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p style="text-align: center;">③</p>	<b>1</b>	<pre>t = Pop(); Accumuler(t.racine); Push(t.right); Push(t.left);</pre>




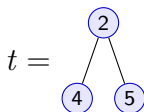
# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p style="text-align: center;">3</p>	<b>2, 1</b>	<b>Accumuler(t.racine);</b> Push(t.right); Push(t.left);



# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
 <p>Diagram showing a stack of nodes: 5 (top) and 3 (bottom).</p>	<p>2, 1</p>	<p>Push(t.right); Push(t.left);</p>



# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)

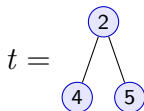
Accumulateur

Actions

2, 1



Push(*t.left*);



# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p data-bbox="212 492 404 533">④ ⑤ ③</p>	<p data-bbox="692 238 775 279">2, 1</p>	<pre data-bbox="843 295 1227 502">t = Pop(); Accumuler(t.racine); Push(t.right); Push(t.left);</pre>

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p data-bbox="257 495 377 536">⑤ ③</p>	<p data-bbox="696 241 772 282">2, 1</p>	<pre data-bbox="847 293 1225 505">t = Pop(); Accumuler(t.racine); Push(t.right); Push(t.left);</pre>
	<p data-bbox="614 655 772 697"><math>t =</math> ④</p>	

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p data-bbox="257 495 377 536">⑤   ③</p>	<p data-bbox="647 240 775 281">4, 2, 1</p>	<p data-bbox="847 350 1223 505">Accumuler(t.racine); Push(t.right); Push(t.left);</p>
	<p data-bbox="614 654 775 696"><math>t =</math> ④</p>	

# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)

Accumulateur

Actions

4, 2, 1

5 3

Push(t.right);  
Push(t.left);

$t =$  4



# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p data-bbox="257 495 377 536">⑤   ③</p>	<p data-bbox="646 241 773 282">4, 2, 1</p>	<p data-bbox="847 458 1071 505">Push(t.left);</p>
	<p data-bbox="611 655 773 697"><math>t =</math> ④</p>	

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<p data-bbox="257 495 377 536">⑤   ③</p>	<p data-bbox="646 239 773 280">4, 2, 1</p>	<pre data-bbox="845 294 1222 505">t = Pop(); Accumuler(t.racine); Push(t.right); Push(t.left);</pre>

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
<span style="border: 1px solid blue; border-radius: 50%; padding: 2px 8px;">3</span>	<span style="color: red;">4, 2, 1</span>	<pre style="margin: 0;"><span style="color: red;">t = Pop();</span> Accumuler(t.racine); Push(t.right); Push(t.left);</pre>

$t =$  5

# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)

Accumulateur

Actions

5, 4, 2, 1

3

```
Accumuler(t.racine);  
Push(t.right);  
Push(t.left);
```

$t =$  5

# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)

Accumulateur

Actions

5, 4, 2, 1

3

Push(t.right);  
Push(t.left);

$t =$  5

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
$\textcircled{3}$	$5, 4, 2, 1$	$\text{Push}(t.\text{left});$
	$t = \textcircled{5}$	

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
③	5, 4, 2, 1	<pre>t = Pop(); Accumuler(t.racine); Push(t.right); Push(t.left);</pre>

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
	5, 4, 2, 1	<code>t = Pop();</code> <code>Accumuler(t.racine);</code> <code>Push(t.right);</code> <code>Push(t.left);</code>

$t =$  ③



# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
	3, 5, 4, 2, 1	Accumuler(t.racine); Push(t.right); Push(t.left);

$t =$  ③

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
	3, 5, 4, 2, 1	Push(t.right); Push(t.left);

$t =$  ③

# Parcours en profondeur préfixe

Pile d'arbres  
(haut de pile à gauche)

Accumulateur

Actions

3, 5, 4, 2, 1

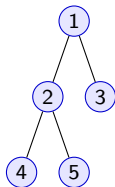
Push(t.left);

$t =$  ③

# Parcours en profondeur préfixe

Pile d'arbres (haut de pile à gauche)	Accumulateur	Actions
	3, 5, 4, 2, 1	

Parcours en profondeur préfixe : 1, 2, 4, 5, 3



# Parcours en profondeur : programmation

- ▶ On peut utiliser une liste pour simuler la pile.
- ▶ Les opérations **Push** et **Pop** s'implémentent en  $O(1)$ .

# Parcours en profondeur : programmation

- ▶ On peut utiliser une liste pour simuler la pile.
- ▶ Les opérations `Push` et `Pop` s'implémentent en  $O(1)$ .
- ▶ Si la pile contient `t::fin` où `t=(Bin(x,left,right))` on retourne `DFS (left::right::fin) (x::acc)`

# Parcours en profondeur : programmation

- ▶ On peut utiliser une liste pour simuler la pile.
- ▶ Les opérations `Push` et `Pop` s'implémentent en  $O(1)$ .
- ▶ Si la pile contient `t::fin` où `t=(Bin(x,left,right))` on retourne `DFS (left::right::fin) (x::acc)`
- ▶ La pile passe de `t::fin` à `left::right::fin`. On a :
  1. Dépilé `t`,
  2. puis empilé `right`,
  3. puis empilé `left`.

## Autres parcours en profondeur : exercice

Adapter l'algorithme pour les parcours en profondeur **infixe**.



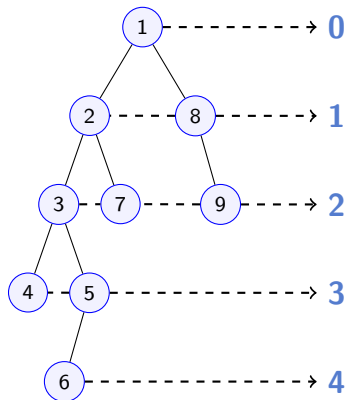
# Parcours en largeur (BFS, Breadth-First Search)

Ce type de parcours visite les nœuds par **profondeurs croissantes**.

- ▶ On visite d'abord les nœuds à profondeur 0 (= la racine),
- ▶ Puis les nœuds à profondeur 1 (= les fils de la racine),
- ▶ Puis les nœuds à profondeur 2,
- ▶ etc.

# Parcours en largeur : exemple

Visite des nœuds par **profondeurs croissantes**.



Parcours en largeur : 1 2 8 3 7 9 4 5 6

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**



Enqueue(1)

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**



Enqueue(1) Enqueue(2)

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**



Enqueue(1) Enqueue(2) Enqueue(3)

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**



Enqueue(1) Enqueue(2) Enqueue(3) Dequeue

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**

3

Enqueue(1) Enqueue(2) Enqueue(3) Dequeue Dequeue



# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**



Enqueue(1) Enqueue(2) Enqueue(3) Dequeue Dequeue Enqueue(4)

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**

4

Enqueue(1) Enqueue(2) Enqueue(3) Dequeue Dequeue Enqueue(4)  
Dequeue

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**

Enqueue(1) Enqueue(2) Enqueue(3) Dequeue Dequeue Enqueue(4)  
Dequeue Dequeue

# Parcours en largeur

- ▶ Même algorithme en remplaçant la **pile** par une **file**.

**Rappel sur les files. 2 opérations : Enqueue(x) et Dequeue**

**Pour le parcours, ce sont des sous-arbres qu'on mémorise**

# Parcours en largeur

# Parcours en largeur : programmation

- ▶ But : implémenter **Enqueue**, **Dequeue** en temps  $O(1)$ .
  - ↪ soit utiliser une liste doublement chaînée,
  - ↪ soit simuler la file par **2 piles**.

# Parcours en largeur : programmation

- ▶ But : implémenter **Enqueue**, **Dequeue** en temps  $O(1)$ .
  - ↪ soit utiliser une liste doublement chaînée,
  - ↪ soit simuler la file par **2 piles**.
- ▶ Si la file contient `t::fin` où `t=(Bin(x,left,right))`  
on voudrait retourner **BFS** (`fin @ [left;right]`) (`x::acc`)

# Parcours en largeur : programmation

- ▶ But : implémenter **Enqueue**, **Dequeue** en temps  $O(1)$ .
  - ↪ soit utiliser une liste doublement chaînée,
  - ↪ soit simuler la file par **2 piles**.
- ▶ Si la file contient `t::fin` où `t=(Bin(x, left, right))` on voudrait retourner **BFS** (`fin @ [left;right]`) (`x::acc`)
- ▶ Non efficace à cause de la concaténation `@`.
- ▶ Implémentation efficace ? Simulation d'une file par **2 piles**.



# Plan

Retour sur les parcours d'arbres

Arbres planaires

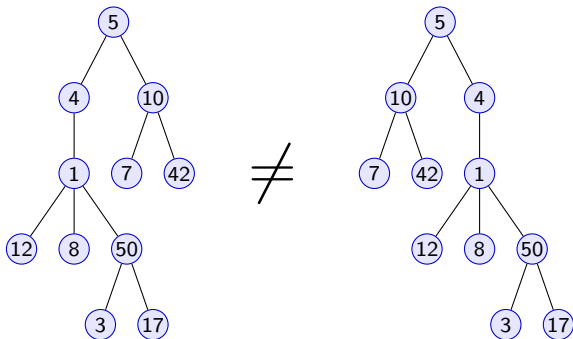
Énumération et bijections



# Arbres planaires

**Définition informelle** : arbres dans lesquels

- ▶ l'arité n'est pas contrainte.
- ▶ l'ordre des fils à une importance.



# Arbres planaires

**Définition informelle** : arbres dans lesquels

- ▶ l'arité n'est pas contrainte.
- ▶ l'ordre des fils à une importance.

**Naturels** (ex. arbres de décision en IA, arbres de récursion).

# Arbres planaires

**Définition informelle** : arbres dans lesquels

- ▶ l'arité n'est pas contrainte.
- ▶ l'ordre des fils à une importance.

**Naturels** (ex. arbres de décision en IA, arbres de récursion).

**Définition récursive des arbres planaires**

- ▶ Constitué d'un nœud, et

# Arbres planaires

**Définition informelle** : arbres dans lesquels

- ▶ l'arité n'est pas contrainte.
- ▶ l'ordre des fils à une importance.

**Naturels** (ex. arbres de décision en IA, arbres de récursion).

**Définition récursive des arbres planaires**

- ▶ Constitué d'un nœud, et
- ▶ d'une liste d'arbres planaires.

# Arbres planaires

**Définition informelle** : arbres dans lesquels

- ▶ l'arité n'est pas contrainte.
- ▶ l'ordre des fils à une importance.

**Naturels** (ex. arbres de décision en IA, arbres de récursion).

**Définition récursive des arbres planaires**

- ▶ Constitué d'un nœud, et
- ▶ d'une liste d'arbres planaires.

**Note.** Cette définition exclut l'arbre vide.

# Plan

Retour sur les parcours d'arbres

Arbres planaires

Énumération et bijections



# Énumération (de squelettes) d'arbres

Combien y a-t-il de squelettes

- ▶ d'arbres binaires à  $n$  nœuds ?
- ▶ d'arbres binaires complets à  $n$  nœuds ?
- ▶ d'arbres planaires à  $n$  nœuds ?

# Énumération (de squelettes) d'arbres

Combien y a-t-il de squelettes

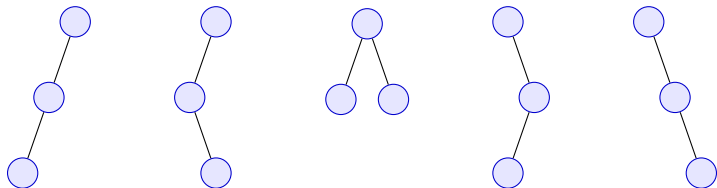
- ▶ d'arbres binaires à  $n$  nœuds ?
- ▶ d'arbres binaires complets à  $n$  nœuds ?
- ▶ d'arbres planaires à  $n$  nœuds ?

## Intérêt de ces questions

- ▶ représentation différentes de mêmes objets,
- ▶ génération aléatoire,
- ▶ analyse d'algorithmes **en moyenne**.

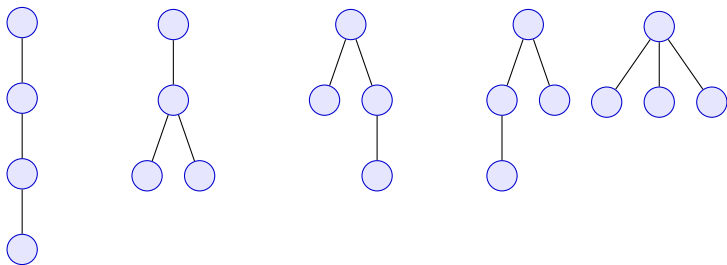
# Énumération d'arbres binaires

Combien y a -t-il de squelettes d'arbres **binaires à 3 nœuds** ?



# Énumération d'arbres planaires

Combien y a-t-il de squelettes d'arbres **planaires à 4 nœuds** ?



# Comptage

Il y a autant

- ▶ de squelettes d'arbres binaires à  $n$  nœuds.
- ▶ de squelettes d'arbres binaires complets à  $2n + 1$  nœuds.
- ▶ de squelettes d'arbres planaires à  $n + 1$  nœuds.
- ▶ de mots de Dyck de longueur  $2n$ .
- ▶ de chemins de Dyck de longueur  $2n$ .

# Comptage

Il y a autant

- ▶ de squelettes d'arbres binaires à  $n$  nœuds.
- ▶ de squelettes d'arbres binaires complets à  $2n + 1$  nœuds.
- ▶ de squelettes d'arbres planaires à  $n + 1$  nœuds.
- ▶ de mots de Dyck de longueur  $2n$ .
- ▶ de chemins de Dyck de longueur  $2n$ .

Leur nombre est

$$C_n = \frac{1}{n+1} \binom{2n}{n}.$$

Nombres de Catalan [Lien 1,2] : 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...

# Comptage

Il y a autant

- ▶ de squelettes d'arbres binaires à  $n$  nœuds.
- ▶ de squelettes d'arbres binaires complets à  $2n + 1$  nœuds.
- ▶ de squelettes d'arbres planaires à  $n + 1$  nœuds.
- ▶ de mots de Dyck de longueur  $2n$ .
- ▶ de chemins de Dyck de longueur  $2n$ .

Leur nombre est

$$C_n = \frac{1}{n+1} \binom{2n}{n}.$$

Nombres de Catalan [Lien 1,2] : 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...

Il y a des **bijections simples** entre ces ensembles.

# Mots de Dyck

Un **mot de Dyck** est une suite de symboles  $\uparrow, \downarrow$  tq :

1. Il y a autant de  $\uparrow$  que de  $\downarrow$ .
2. Quand on lit la suite de gauche à droite, il y a toujours au moins autant de  $\uparrow$  que de  $\downarrow$ .

## Exemple

▶  $\uparrow \uparrow \downarrow \downarrow$

▶  $\uparrow \downarrow \uparrow \downarrow$



# Mots de Dyck

Un **mot de Dyck** est une suite de symboles  $\uparrow, \downarrow$  tq :

1. Il y a autant de  $\uparrow$  que de  $\downarrow$ .
2. Quand on lit la suite de gauche à droite, il y a toujours au moins autant de  $\uparrow$  que de  $\downarrow$ .

## Exemple

▶  $\uparrow \uparrow \downarrow \downarrow$   $(( ))$

▶  $\uparrow \downarrow \uparrow \downarrow$   $()()$

Un mot de Dyck est aussi appelé mot bien parenthésé.

# Mots de Dyck

Un **mot de Dyck** est une suite de symboles  $\uparrow, \downarrow$  tq :

1. Il y a autant de  $\uparrow$  que de  $\downarrow$ .
2. Quand on lit la suite de gauche à droite, il y a toujours au moins autant de  $\uparrow$  que de  $\downarrow$ .

## Exemple

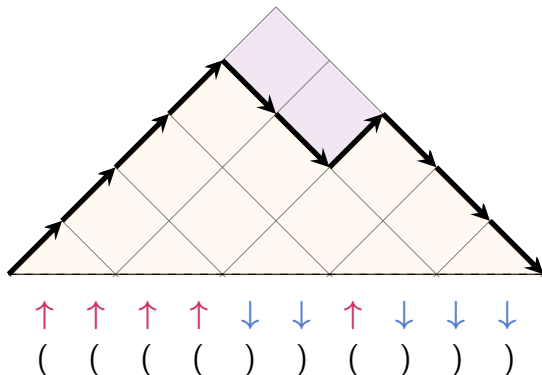
- ▶  $\uparrow \uparrow \downarrow \downarrow$   $(( ))$
- ▶  $\uparrow \downarrow \uparrow \downarrow$   $()()$

Un mot de Dyck est aussi appelé mot bien parenthésé.

1. Autant d'ouvrantes que de fermantes.
2. Jamais plus de fermantes que d'ouvrantes quand on lit de gauche à droite.

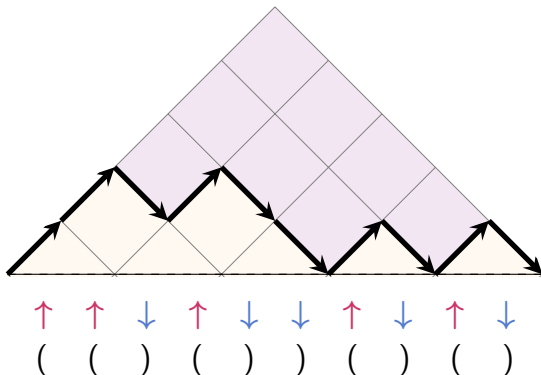
# Chemins de Dyck

Simplement une représentation graphique des mots de Dyck.



# Chemins de Dyck

Simplement une représentation graphique des mots de Dyck.

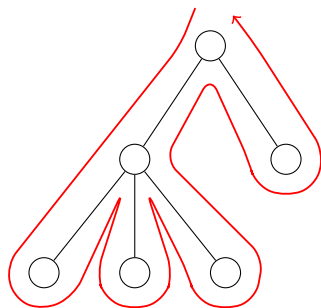


# Chemins de Dyck

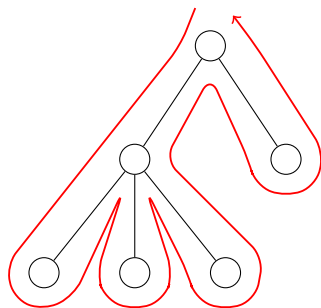
## Ces chemins

- ▶ Partent de l'origine  $(0,0)$ ,
- ▶ 2 types de pas : haut ou bas.
- ▶ Restent au dessus de l'altitude 0.
- ▶ Terminent à l'altitude 0.

# Bijection arbres planaires $\leftrightarrow$ chemins de Dyck

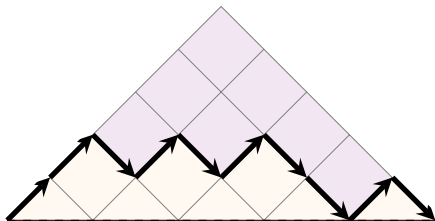
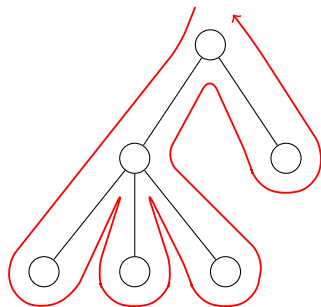


## Bijection arbres planaires $\leftrightarrow$ chemins de Dyck



- ▶ Parcours en profondeur de l'arbre.
- ▶ On produit ( lorsqu'on descend et ) lorsqu'on monte.

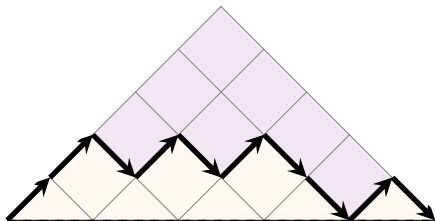
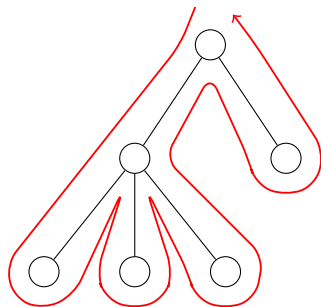
# Bijection arbres planaires $\leftrightarrow$ chemins de Dyck



- ▶ Parcours en profondeur de l'arbre.
- ▶ On produit ( lorsqu'on descend et ) lorsqu'on monte.



# Bijection arbres planaires $\leftrightarrow$ chemins de Dyck



C'est une **bijection**

- ▶ Chaque arbre donne un chemin différent,
- ▶ Chaque chemin est obtenu depuis un arbre.

# Arbres planaires représentés par arbres binaires

Représentation simple d'un arbre planaire : associer à chaque nœud

- ▶ son fils gauche,
- ▶ son frère droit.

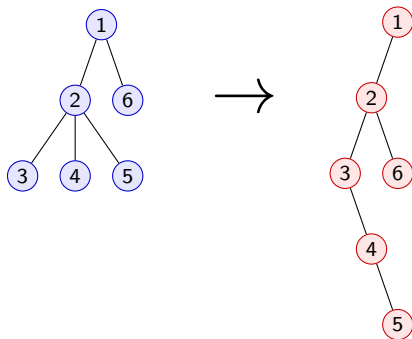
Par exemple, en C, on pourrait définir

```
struct planar_tree
{
    int key;
    struct planar_tree *left_child, *next_brother;
};
```

Ce codage donne l'idée d'une bijection.

# Arbres planaires représentés par arbres binaires

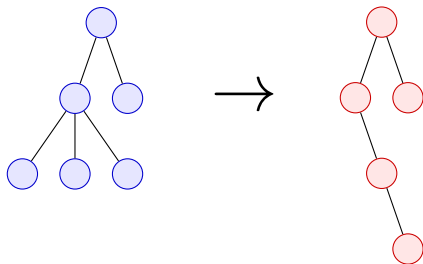
On donne comme fils d'un nœud dans l'arbre binaire son fils gauche et son frère droit dans l'arbre planaire.



La racine de l'arbre binaire obtenu n'a jamais de fils droit.

# Arbres planaires représentés par arbres binaires

Pour obtenir une bijection, entre arbres planaires à  $n + 1$  nœuds et arbres binaires à  $n$  nœuds, on supprime la racine de l'arbre obtenu.



## Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $C_n$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .

## Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .

## Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .
- ▶ On doit retrancher les chemins descendant sous l'altitude 0.  
⇒ Combien de chemins descendent sous l'altitude 0?

## Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .
- ▶ On doit retrancher les chemins descendant sous l'altitude 0.  
⇒ Combien de chemins descendent sous l'altitude 0?  
Autant que de chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .



# Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .
- ▶ On doit retrancher les chemins descendant sous l'altitude 0.  
⇒ Combien de chemins descendent sous l'altitude 0?  
Autant que de chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .



**Bijection** entre

- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .
- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$  en passant sous l'altitude 0.

## Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .
- ▶ On doit retrancher les chemins descendant sous l'altitude 0.  
⇒ Combien de chemins descendent sous l'altitude 0 ?  
Autant que de chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .



**Bijection** entre

- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .
- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$  en passant sous l'altitude 0.
- ▶ Il y a  $\binom{2n-1}{n}$  tels chemins (choix des  $n+1$  descentes).

## Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .
- ▶ On doit retrancher les chemins descendant sous l'altitude 0.  
⇒ Combien de chemins descendent sous l'altitude 0?  
Autant que de chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .



**Bijection** entre

- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .
- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$  en passant sous l'altitude 0.
- ▶ Il y a  $\binom{2n}{n+1}$  tels chemins (choix des  $n+1$  descentes).

# Nombre de chemins de Dyck de taille $2n$

- ▶ Il y a  $\binom{2n}{n}$  chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$ .
- ▶ On doit retrancher les chemins descendant sous l'altitude 0.  
⇒ Combien de chemins descendent sous l'altitude 0?  
Autant que de chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .



**Bijection** entre

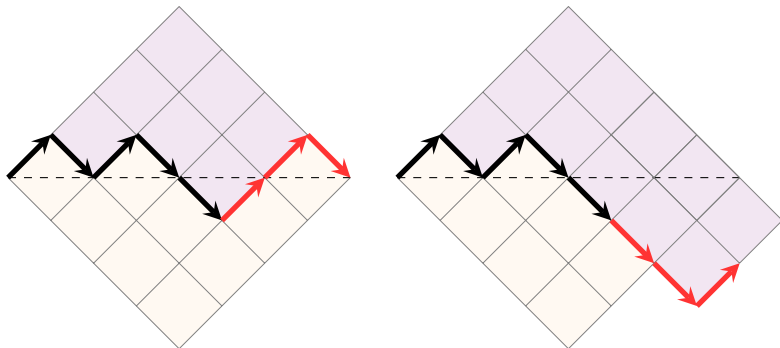
- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,-2)$ .
- ▶ chemins de taille  $2n$  allant de  $(0,0)$  à  $(0,0)$  en passant sous l'altitude 0.
- ▶ Il y a  $\binom{2n}{n+1}$  tels chemins (choix des  $n+1$  descentes).
- ▶ Le nombre de chemins de Dyck de taille  $2n$  est donc

$$\binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1} \binom{2n}{n}.$$

# Chemins descendant sous l'altitude 0

Bijection : inversion des pas à partir du premier point d'altitude  $-1$ .

## Exemple 1



# Chemins descendant sous l'altitude 0

Bijection : inversion des pas à partir du premier point d'altitude  $-1$ .

## Exemple 2

