

Plan du cours

Période 2

- Expression fonctionnelle (décembre, janvier - 7 cours et 7 td)
 - 5. Définitions récursives
 - 6. Arbres
- Expression actionnelle (janvier-février, 7 cours et 7 td)
 - 7. Composition récursive des actions
 - 8. Arbres et forêts : représentation chaînée

Agenda à partir de décembre

- Cours tous les lundis 13h30
+ les mardis 9h45 (en alternance avec le TD)
- TD tous les lundis 15h15
+ les mardis 9h45 (en alternance avec le cours)
- Devoir surveillé : fin janvier (1h sans documents)
- Examen : fin février (3h - sans documents)

S-1

P.-C. Scholl, C. Vigouroux – décembre 2025

Styles d'expression

Langage des actions

- Variable : association nom-valeur
 - Affectation
 - Compositions séquentielle, conditionnelle, itérative, **récursive**
 - Tableaux, chaînage, adresses, indirection
- Action : décrit des changements d'états
 - Spécification, réalisation, utilisation
- Description de valeurs : langage des expressions

Langage des expressions

revoir chapitre 1

- Types, valeurs, expressions
 - Compositions fonctionnelle, conditionnelle, **récursive**
 - Nommer une expression : construction "**soit ... dans ...**"
- Fonction : décrit des valeurs
 - Spécification, réalisation, utilisation
- Types construits : produit, textes, séquences, **arbres**

S-3

P.-C. Scholl, C. Vigouroux – décembre 2025

5. Définitions récursives

5.0. Introduction

Expression fonctionnelle

Introduction à la récursivité - Principes d'analyse récurrente

Fonctions : spécification, réalisation, vérification

Faire abstraction des représentations

5.0. Introduction

- 5.1. Définitions récursives de types et de fonctions
- 5.2. Modèles d'analyse récurrente
- 5.3. Découpage d'une séquence selon une propriété
- 5.4. Compléments



Connaître les définitions

S'approprier le langage des expressions

Savoir définir une fonction par des équations de récurrence

S-2

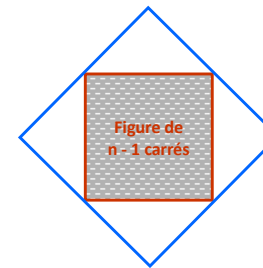
P.-C. Scholl, C. Vigouroux – décembre 2025

a) Notion de définition récursive

Définition

Texte définissant **une entité** et faisant référence à **une entité de même nature**

Une **figure** de n carrés est formée
d'un **carré** et d'une **figure** de $n-1$ carrés



n carrés ($n > 1$)

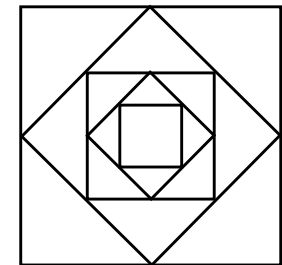
vision récursive



1 carré

objet de base

Une figure est formée
de n carrés emboîtés



n carrés emboîtés

vision itérative ?

S-4

P.-C. Scholl, C. Vigouroux – décembre 2025

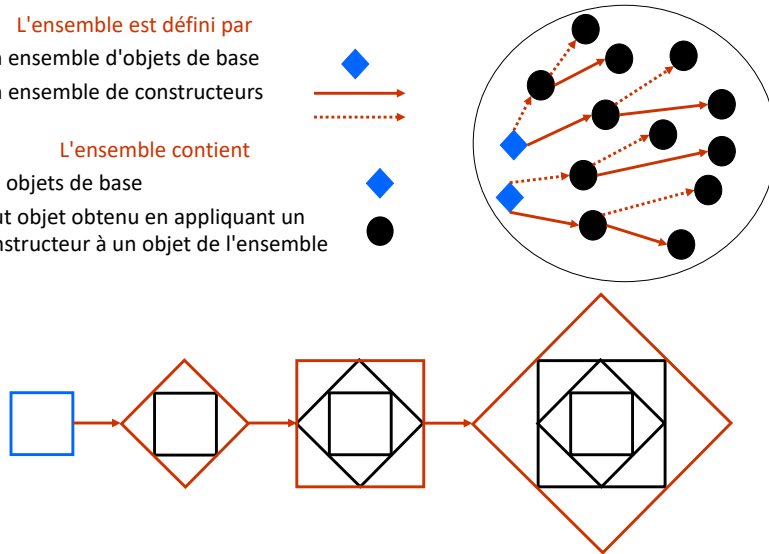
b) Définition récursive d'un ensemble

L'ensemble est défini par

- Un ensemble d'objets de base
- Un ensemble de constructeurs

L'ensemble contient

- Les objets de base
- Tout objet obtenu en appliquant un constructeur à un objet de l'ensemble



S-5

P.-C. Scholl, C. Vigouroux – décembre 2025

Division entière

Quotient : fonction $(A : \text{entier} \geq 0, B : \text{entier} > 0) \rightarrow \text{entier} \geq 0$

Idée : procéder par soustractions successives

Propriétés

1. $A < B$: quotient $(A, B) = 0$
2. $A \geq B$: quotient $(A, B) = 1 + \text{quotient}(A-B, B)$

Réalisation récursive

Quotient (A, B) :

retour :

si $A < B$ alors 0

sinon $1 + \text{Quotient}(A-B, B)$

Réalisation itérative

Quotient (A, B) :

$Q : \text{entier} \geq 0 ; R : \text{entier} \geq 0$

$Q \leftarrow 0 ; R \leftarrow A$

tant que $R \geq B$

$\{A = B \times Q + R\}$

$Q \leftarrow 1 + Q ; R \leftarrow R - B$

$\{A = B \times Q + R \text{ et } 0 \leq R < B\}$

retour : Q

S-7

P.-C. Scholl, C. Vigouroux – décembre 2025

c) Procédures : réalisations récursives

Récursivité directe

La réalisation de P contient au moins un appel récursif de P

P(X) :

• • • • •

P(X')

• • • • •

Récursivité croisée

P appelle Q et Q appelle P

P(X) :

• • • • •

Q(X')

• • • • •

Q(Y) :

• • • • •

P(Y')

• • • • •

Au moins une conditionnelle distinguant un cas non récursif

S-6

P.-C. Scholl, C. Vigouroux – décembre 2025

d) Observation de l'évaluation d'une fonction

Trace par réécriture

Les équations sont interprétées comme des règles de réécriture

Quotient(17, 5) = 1 + Quotient(17-5, 5) {équation 2}

= 1 + Quotient(12, 5)

= 1 + (1 + Quotient(7, 5)) {équation 2}

= 1 + (1 + (1 + Quotient(2, 5))) {équation 2}

= 1 + (1 + (1 + 0)) {équation 1}

= 1 + (1+1)

= 1 + 2

= 3

(1) $A < B$: Quotient $(A, B) = 0$

(2) $A \geq B$: Quotient $(A, B) = 1 + \text{Quotient}(A-B, B)$

S-8

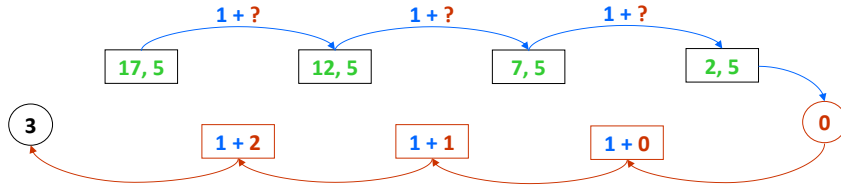
P.-C. Scholl, C. Vigouroux – décembre 2025

d) Observation de l'évaluation d'une fonction

Liste des appels récursifs engendrés

On observe les valeurs successives des paramètres des appels récursifs

Quotient (17, 5)



Deux parcours de la liste des appels
 "Aller" : mise en attente d'opérations
 "Retour" : évaluation des opérations mises en attente (ordre inverse)

5-9

P.-C. Scholl, C. Vigouroux – décembre 2025

Quotient et reste

Division entière

Reste : fonction (A : entier ≥ 0, B : entier > 0) → entier ≥ 0

QR : fonction (A : entier ≥ 0, B : entier > 0) → entier ≥ 0, entier ≥ 0 {quotient, reste}

Analyse récurrente

(1) A < B : Reste(A, B) = A (2) A ≥ B : Reste(A, B) = Reste(A-B, B)

Réalisations récursives

Reste (A, B) :

retour : si A < B alors A sinon Reste(A-B, B)

QR(A, B) :

retour : si A < B alors <0, A>

sinon soit <q, r> = QR(A-B, B)

dans <1+q, r>

5-10

P.-C. Scholl, C. Vigouroux – décembre 2025

5.1 Définitions récursives : types, fonctions

Définitions récursives

5.0. Introduction

5.1. Définitions récursives de types et de fonctions

5.2. Modèles d'analyse récurrente

5.3. Découpage d'une séquence selon une propriété

5.4. Compléments



Connaître les définitions récursives des entiers et des séquences
 Comprendre la différence entre
 équations de récurrence et réalisation récursive

5-11

P.-C. Scholl, C. Vigouroux – décembre 2025

a) Définition récursive des séquences

3 versions

Version 1 : ajout à droite

• Base

- séquence vide : []

• Construction récursive

- S • e

SD [s e]
 S = début(SD) ; e = dernier(SD)

Version 2 : ajout à gauche

• Base

- séquence vide : []

• Construction récursive

- e o S

SD [e s]
 e = premier(SD) ; S = fin(SD)

Version 3 : ajout aux deux extrémités

• Base

- séquence vide : [] ; singleton : [e]

• Construction récursive

- e1 o S • e2

SD [e1 s e2]
 e1 = premier(SD) ; S = fin(début(SD)) ; e2 = dernier(SD)

5-12

P.-C. Scholl, C. Vigouroux – décembre 2025

a) Définition réursive des séquences

Terminologie (exemple de la version 1)

Définitions

- **[]** et **•** sont les **constructeurs** (de séquences)
 - construction de toutes les séquences et uniquement elles
- $((([] • 1) • 2) • 3) • 4$ correspond à $[1, 2, 3, 4]$

- **début** et **dernier** sont les **sélecteurs**
 - sélection des composants qui ont servi à construire un élément

$$SD = S • e \Leftrightarrow \begin{cases} S = \text{début}(SD) \\ e = \text{dernier}(SD) \end{cases}$$

- **EstVide?** est un **testeur**
 - test de la règle qui a servi à construire un élément

$SD = [] \Leftrightarrow \text{EstVide?}(SD)$ a la valeur vrai

5-13

P.-C. Scholl, C. Vigouroux – décembre 2025

Suppression des espaces

Complément Polycopié E5.14

SansEspaces : fonction (TD : texte) → texte

espace : constante ' ' de type caractère

- Propriétés
 1. SansEspaces([]) = []
 2. SansEspaces(c • T) =
 - si c = espace alors SansEspaces(T) sinon c • SansEspaces(T)

Réalisation réursive

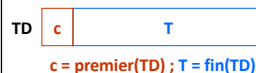
SansEspaces(TD) :

retour :

si EstVide?(TD) alors []

sinon soit c = premier(TD), T = fin(TD)

dans • • • • •



Exercice

Faire la trace de l'évaluation (réécriture) pour un exemple simple

5-15

P.-C. Scholl, C. Vigouroux – décembre 2025

Nombre de 'A'

Polycopié E5.2

NbA : fonction (TD : texte) → entier ≥ 0

- Propriétés
 1. NbA([]) = 0
 2. NbA(T • c) = NbA(T) + (si c = 'A' alors 1 sinon 0)



Réalisation réursive abstraite

NbA(TD) :

retour :

si EstVide?(TD) alors 0

sinon NbA(début(TD))

+ si dernier(TD) = 'A' alors 1

sinon 0

Réalisation itérative concrète

NbA(Tab, Nb) :

R : entier ≥ 0

R ← 0

pour i allant de 1 a Nb

si Tab_i = 'A' alors **R** ← R+1

retour : R

5-14

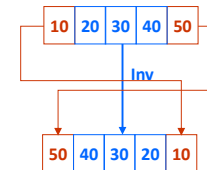
P.-C. Scholl, C. Vigouroux – décembre 2025

Inverse d'une séquence

Polycopié E5.3

Élément : type ; **Séq** : type séquence d'Élément

Inv : fonction (SD : Séq) → Séq



Propriétés

1. Inv([]) = []
2. Inv([e]) = [e]
3. Inv(e1 • S • e2) = e2 • Inv(S) • e1

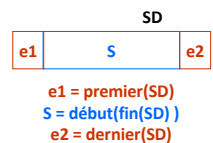
Réalisation réursive

Inv(SD) :

retour :

si EstVide?(SD) ou alors EstVide?(fin(SD)) alors SD

sinon dernier(SD) • Inv(début(fin(SD))) • premier(SD)



Exercices

Faire la trace de l'évaluation (réécriture) pour un exemple simple

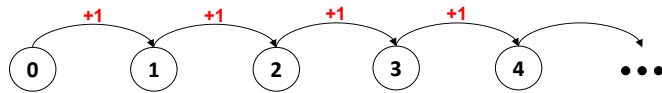
Étudier les solutions itératives, selon la représentation des séquences

5-16

P.-C. Scholl, C. Vigouroux – décembre 2025

b) Définition récursive des entiers naturels

Version 1



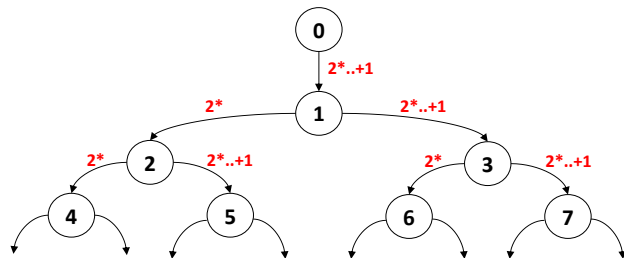
- Base : **0**
- Construction récursive
 - si x est un entier naturel alors $x+1$ est un entier naturel
 - **+1** est le constructeur
 - $((0+1)+1)+1$ correspond à l'entier usuel 3
 - **-1** est le destructeur/sélecteur

5-17

P.-C. Scholl, C. Vigouroux – décembre 2025

b) Définition récursive des entiers naturels

Version 2



- Base : **0**
- Construction récursive
 - si x est un entier naturel, $2*x$ et $2*x+1$ sont des entiers naturels
 - $2*x$ et $2*x+1$ sont les deux constructeurs
 - **quotient 2** est le destructeur/sélecteur
 - testeur : $x \text{ reste } 2 = 0$ (ou on s'autorisera **pair(x)**)

Voir le lien avec la représentation en base 2

5-19

P.-C. Scholl, C. Vigouroux – décembre 2025

Produit de deux naturels

Version 1

Som : fonction $(X, Y : \text{entier} \geq 0) \rightarrow \text{entier} \geq 0$

Prod : fonction $(X, Y : \text{entier} \geq 0) \rightarrow \text{entier} \geq 0$

- Propriétés du produit en termes de la somme

1. $\text{Prod}(A, 0) = 0$
2. $\text{Prod}(A, B+1) = \text{Som}(\text{Prod}(A, B), A)$

Réalisation récursive

Prod (X, Y) :

retour :

si $Y = 0$ alors 0

sinon Som $(\text{Prod}(X, Y-1), X)$

$Y = B+1 \Leftrightarrow B = Y-1$

5-18

P.-C. Scholl, C. Vigouroux – décembre 2025

Produit de deux naturels

Version 2

Complément Polycopié E5.24

Prod : fonction $(X, Y : \text{entier} \geq 0) \rightarrow \text{entier} \geq 0$

- Propriétés

1. $\text{Prod}(A, 0) = 0$
2. $\text{Prod}(A, 2*B) = 2*\text{Prod}(A, B)$ $\{B > 0\}$
3. $\text{Prod}(A, 2*B+1) = \text{Som}(2*\text{Prod}(A, B), A)$

Réalisation récursive

Prod (X, Y) :

retour :

si $Y = 0$ alors 0

sinon

soit $B = Y \text{ quotient } 2$

dans si $Y \text{ reste } 2 = 0$ alors $2*\text{Prod}(X, B)$

sinon Som $(2*\text{Prod}(X, B), X)$

$Y = 2*B \Rightarrow Y \text{ reste } 2 = 0$

$Y = 2*B \Rightarrow B = Y \text{ quotient } 2$

$Y = 2*B+1 \Rightarrow Y \text{ reste } 2 = 1$

$Y = 2*B+1 \Rightarrow B = Y \text{ quotient } 2$

5-20

P.-C. Scholl, C. Vigouroux – décembre 2025

d) Question de terminaison

Quelles que soient les valeurs (valides) des paramètres de l'appel initial, la suite des paramètres des appels récursifs doit converger vers un cas de base.

Somme de deux entiers

Som : fonction $(X, Y : \text{entier} \geq 0) \rightarrow \text{entier} \geq 0$

Erreurs ?

Version 1

Som(X, Y) :
retour : si $X = 0$ alors Y sinon Som(X, Y-1) + 1

Som(4,1) = Som(4, 0) + 1
= Som(4, -1) + 1 + 1
= ...

erreur de type ou
processus infini

Version 2

Som(X, Y) :
retour : si $Y = 0$ alors X sinon Som(Som(X, Y-1), 1)

Som(1,1) = Som(Som(1,0), 1)
= Som(1,1)
= ...

processus infini

Version 3

Som(X, Y) :
retour : si $Y = 0$ alors X sinon Som(X, Y-1) + 1

CORRECTE !

5-21

P.-C. Scholl, C. Vigouroux – décembre 2025

5.2 Modèles d'analyse récurrente

Définitions récursives

5.0. Introduction

5.1. Définitions récursives de types et de fonctions

5.2. Modèles d'analyse récurrente

5.3. Découpage d'une séquence selon une propriété

5.4. Compléments



S'appropriier les étapes de construction d'une solution récursive
Connaître les modèles d'analyse récurrente

5-23

P.-C. Scholl, C. Vigouroux – décembre 2025

e) Analyse quantitative

Nombre d'appels récursifs engendrés par un appel initial ?

Prod(X, Y)

1. Prod(A, 0) = 0
2. Prod(A, 2*B) = 2*Prod(A, B)
3. Prod(A, 2*B+1) = Som(2*Prod(A, B), A)

Prod(3, 39) engendre 6 appels



NbAp(X, Y)

{nombre d'appels engendrés par l'appel Prod(X,Y)}

1. NbAp(A, 0) = 0
2. NbAp(A, 2*B) = 1 + NbAp(A, B)
3. NbAp(A, 2*B+1) = 1 + NbAp(A, B)

Nombre d'appels récursifs pour évaluer Prod(X, Y) : $\approx \log_2 Y$ (nb de puissances de 2 dans Y)

5-22

P.-C. Scholl, C. Vigouroux – décembre 2025

a) Plan général d'analyse récurrente

Objectif : Décrire F(D) en termes de F(D') ; D' est "plus petit" que D

Étapes

- Recherche de propriétés
- Choix d'une définition récurrente
 - ensemble cohérent d'équations de récurrence
 - exclusion mutuelle, convergence vers les cas de base
- Réalisation récursive
 - traduction directe de la définition récurrente
- Examen de propriétés
 - correction, analyse quantitative, affinements

À voir ou à revoir : raisonnement par récurrence

5-24

P.-C. Scholl, C. Vigouroux – décembre 2025

Somme de deux naturels

Recherche de propriétés

Som : fonction $(X, Y : \text{entier} \geq 0) \rightarrow \text{entier} \geq 0$

Recherche de propriétés

- Cas simples
 - (1) Som (A, 0) = A
 - (2) Som (0, B) = B
 - (3) Som (0, 0) = 0
- La récurrence porte sur l'un des paramètres
 - (4) Som (A, B+1) = Som (A, B) + 1
 - (6) Som (A+1, B) = Som (A, B) + 1
 - (5) Som (A, B+1) = Som (A+1, B)
 - (7) Som (A+1, B) = Som (A, B+1)
- La récurrence porte sur les deux paramètres
 - (8) Som (A+1, B+1) = (Som (A, B) + 1) + 1

Définitions récurrentes : ensembles cohérents d'équations

- v1 (4) + (1)
- v2 (5) + (1)
- v3 (6) + (2)
- v4 (7) + (2)
- v5 (8) + (1) avec A > 0 + (2) avec B > 0 + (3)

Contre-exemples

- (4) + (2)
- (8) + (1)

5-25

P.-C. Scholl, C. Vigouroux – décembre 2025

b) Paramètres de type entier naturel

Modèles d'analyse

La récurrence porte sur un seul paramètre

Découpage	Forme des équations	Forme des réalisations de f(X)	Variante : entier > 0
version 1	$f(0) = \dots$ $f(n+1) = \dots f(n)$	si X = 0 alors $\dots$ sinon $\dots f(X-1)$	$f(1) = \dots$ $f(n+1) = \dots f(n) \quad \{n > 0\}$
version 2	$f(0) = \dots$ $f(2*n) = \dots f(n) \{n > 0\}$ $f(2*n+1) = \dots f(n)$	si X = 0 alors $\dots$ sinon si X reste 2 = 0 alors $\dots f(X \text{ quotient } 2)$ sinon $\dots f(X \text{ quotient } 2)$	$f(1) = \dots$ $f(2*n) = \dots f(n) \quad \{n > 0\}$ $f(2*n+1) = \dots f(n) \{n > 0\}$

La récurrence porte sur plusieurs paramètres. Exemple : $f(X, Y)$

Combinaison	V1, V1	V1, V2
	$f(0, 0) = \dots$ $f(0, m+1) = \dots$ $f(n+1, 0) = \dots$ $f(n+1, m+1) = \dots$	$f(0, 0) = \dots$ $f(0, 2*m) = \dots \{m > 0\}$ $f(0, 2*m+1) = \dots$ $f(n+1, 0) = \dots$ $f(n+1, 2*m) = \dots \{m > 0\}$ $f(n+1, 2*m+1) = \dots$

5-26

P.-C. Scholl, C. Vigouroux – décembre 2025

Forme binaire d'un entier

Analyse

Bit : type entier sur [0,1]

EvB : fonction $(E : \text{entier} \geq 0) \rightarrow \text{séquence de Bit}$

Définition récurrente

- (1) $\text{EvB}(0) = \text{[0]}$
- (2) $\text{EvB}(2*A) = \text{EvB}(A) \bullet 0 \quad \{A > 0\}$
- (3) $\text{EvB}(2*A+1) = \text{EvB}(A) \bullet 1$

Erreur ?

$$13 = 2 * 6 + 1$$



Réalisation

EvB(E) :
retour :
si E = 0 alors [0]
sinon
 $\text{EvB}(E \text{ quotient } 2) \bullet E \text{ reste } 2$

Test

- EvB(13) = ?
01101

Comment éviter le 0 de gauche ?

5-27

P.-C. Scholl, C. Vigouroux – décembre 2025

Forme binaire d'un entier

Démonstration par récurrence

EvB(E) :

retour :

si E = 0 alors [0]

sinon

$\text{EvB}(E \text{ quotient } 2) \bullet E \text{ reste } 2$

Montrer que pour tout E, EvB(E) a un zéro en tête

On examine le texte de EvB(E)

- cas de base
 - on montre que la propriété est vraie pour EvB(0)
- cas récursif
 - hypothèse de récurrence
 - on suppose que EvB(E quotient 2) vérifie la propriété
 - Preuve d'hérédité
 - on montre qu'alors EvB(E) la vérifie aussi

5-28

P.-C. Scholl, C. Vigouroux – décembre 2025

Forme binaire d'un entier

Eviter le zéro de gauche : deux solutions

Solution 1

Ajouter un cas de base

(1') $\text{EvB}(1) = [1]$

et changer (3) en (3')

Solution 2

Spécifier une nouvelle fonction

$\text{EvB1} : \text{fonction } (E : \text{entier} > 0) \rightarrow \text{séquence de Bit}$

Réaliser EvB en termes de EvB1

(0) $\text{EvB}(E) = \text{si } E = 0 \text{ alors } [0] \text{ sinon } \text{EvB1}(E)$

Reprendre l'analyse pour EvB1

(1') $\text{EvB1}(1) = [1]$

(2) $\text{EvB1}(2 * A) = \text{EvB1}(A) \bullet 0 \quad \{A > 0\}$

(3') $\text{EvB1}(2 * A + 1) = \text{EvB1}(A) \bullet 1 \quad \{A > 0\}$

5-29

P.-C. Scholl, C. Vigouroux – décembre 2025

Egalité de deux textes

Polycopié E5.4

$\text{EgTexte} : \text{fonction } (T1, T2 : \text{texte}) \rightarrow \text{booléen}$

(1) $\text{EgTexte}([], []) = \text{vrai}$

(2) $\text{EgTexte}([], eoS) = \text{faux}$

(3) $\text{EgTexte}(eoS, []) = \text{faux}$

(4) $\text{EgTexte}(e1oS1, e2oS2) = \text{EgTexte}([e1], [e2]) \text{ et } \text{EgTexte}(S1, S2)$

Erreur ?

($e1 = e2$) et $\text{EgTexte}(S1, S2)$

Observation de l'évaluation

$\text{EgTexte}(\text{"abc"}, \text{"ab"})$: nombre de comparaisons de caractères ?

Hypothèses :

- les opérandes du **et** sont évalués si nécessaire de **gauche à droite**
- les opérandes du **et** sont évalués si nécessaire de **droite à gauche**

5-31

P.-C. Scholl, C. Vigouroux – décembre 2025

c) Paramètres de type séquence

Modèles d'analyse

Polycopié § 5.A.c

La récurrence porte sur un seul paramètre de type séquence

Découpage isolant	Forme des équations	Forme des réalisations de $f(X)$	Variante Séquence non vide
premier élément	$f([]) = \bullet \bullet \bullet$ $f(eoS) = \bullet \bullet \bullet f(S)$	si $\text{EstVide?}(X)$ alors $\bullet \bullet \bullet$ sinon soit $e = \text{premier}(X)$, $S = \text{fin}(X)$ dans $\bullet \bullet \bullet$	$f([e]) = \bullet \bullet \bullet$ $f(eoS) = \bullet \bullet \bullet$ $\{S \text{ non vide}\}$
dernier élément	$f([]) = \bullet \bullet \bullet$ $f(S \bullet e) = \bullet \bullet \bullet f(S)$	si $\text{EstVide?}(X)$ alors $\bullet \bullet \bullet$ sinon soit $e = \text{dernier}(X)$, $S = \text{début}(X)$ dans $\bullet \bullet \bullet$	$f([e]) = \bullet \bullet \bullet$ $f(S \bullet e) = \bullet \bullet \bullet$ $\{S \text{ non vide}\}$
premier et dernier	$f([]) = \bullet \bullet \bullet$ $f([e]) = \bullet \bullet \bullet$ $f(e1oS \bullet e2) = \bullet \bullet \bullet f(S)$	si $\text{EstVide?}(X)$ alors $\bullet \bullet \bullet$ sinon si $\text{EstVide?}(\text{fin}(X))$ alors $\bullet \bullet \bullet$ sinon soit $e1 = \text{premier}(X)$, $e2 = \text{dernier}(X)$, $S = \text{début}(\text{fin}(X))$ dans $\bullet \bullet \bullet$	$f([e]) = \bullet \bullet \bullet$ $f([e1, e2]) = \bullet \bullet \bullet$ $f(e1oS \bullet e2) = \bullet \bullet \bullet$ $\{S \text{ non vide}\}$
deux premiers	$f([], f([e])$ $f(e1oS2oS)$		
deux derniers	$f([], f([e])$ $f(S \bullet e1 \bullet e2)$		

5-30

P.-C. Scholl, C. Vigouroux – décembre 2025

Somme d'une séquence d'entiers

A propos d'évaluation

$f : \text{fonction } (SD : \text{séquence non vide d'entier} > 0) \rightarrow \text{entier} > 0$

(1) $f([e]) = e$

(2) $f(e \bullet S) = e + f(S) \quad \{S \text{ non vide}\}$

(3) $f(S \bullet e) = e + f(S) \quad \{S \text{ non vide}\}$

version 1 : (1) + (2)

version 2 : (1) + (3)

réécriture de $f([10, 20, 30, 40])$

version 1 : découpage à gauche

$[10, 20, 30, 40] = 10 \bullet [20, 30, 40]$

$10 + (20 + (30 + (40)))$: éléments traités de droite à gauche

version 2 : découpage à droite

$[10, 20, 30, 40] = [10, 20, 30] \bullet 40$

$40 + (30 + (20 + (10)))$: éléments traités de gauche à droite

5-32

P.-C. Scholl, C. Vigouroux – décembre 2025

d) Analyse récurrente

Ce n'est pas forcément la fonction de départ qui est décrite récursivement

- Spécification
 - précision, cas limites !!!
- Recherche de la récurrence
 - exemples simples, **dessins !!!**
- Définition récurrente
 - application de **modèles d'analyse récurrente**
 - choix du ou des paramètres sur lesquels porte la récurrence
 - choix d'un mode de découpage
 - combinaison éventuelle des cas

5-33

P.-C. Scholl, C. Vigouroux – décembre 2025

5.3 Découpage selon une propriété

Définitions récursives

5.0. Introduction

5.1. Définitions récursives de types et de fonctions

5.2. Modèles d'analyse récurrente

5.3. Découpage d'une séquence selon une propriété

5.4. Compléments

5-35

P.-C. Scholl, C. Vigouroux – décembre 2025

N^{ème} élément d'une séquence

Polycopié E5.6

Spécification

Élément : **type** ; Séq : **type séquence d'Élément**

Nième : **fonction** (N : entier > 0, SD : Séq non vide) → booléen, Élément

{Posons < OK, E > = Nième(N, SD) :

$N > \text{longueur}(SD) \Rightarrow OK = \text{faux et } E \text{ non spécifié}$

$N \leq \text{longueur}(SD) \Rightarrow OK = \text{vrai et } E \text{ est le } N^{\text{ème}} \text{ élément de } SD\}$

Définition récurrente

(1) Nième(1, [e])	= <vrai, e>	
(2) Nième(n+1, [e])	= <faux, ?>	{n>0}
(3) Nième(1, eoS)	= <vrai, e>	{S non vide}
(4) Nième(n+1, eoS)	= Nième(n, S)	{n>0, S non vide}

Variante : regroupement de cas

(1') Nième(1, eoS)	= <vrai, e>	
(2') Nième(n+1, [e])	= <faux, ?>	{n>0}
(3') Nième(n+1, eoS)	= Nième(n, S)	{n>0, S non vide}

5-34

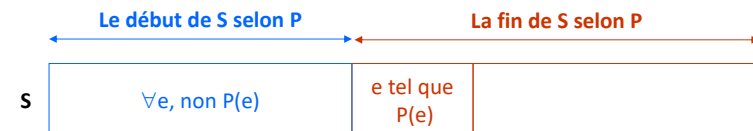
P.-C. Scholl, C. Vigouroux – décembre 2025

a) Principes

Définitions

Découpage d'une séquence S en deux séquences

l'élément de séparation est le premier élément vérifiant une propriété P donnée



Cas limites

	Le début	La fin
$S = []$	$[]$	$[]$
$S \neq []$ et $\forall e \in S, \text{non } P(e)$	S	$[]$
$S \neq []$ et $P(\text{premier}(S))$	$[]$	S

5-36

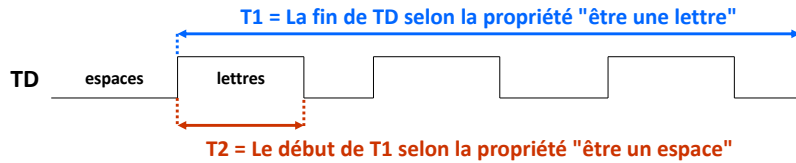
P.-C. Scholl, C. Vigouroux – décembre 2025

Premier mot d'une phrase

Texte formé de lettres et d'espaces

Polycopié E5.8

Idée : supprimer les espaces du début, puis extraire les lettres du début



Premier niveau d'analyse

mot : **type** séquence de lettre

PM : fonction (TD : texte) → mot {Premier Mot}

SED : fonction (TD : texte) → texte {Sans Espace au Début}

LLD : fonction (TD : texte) → mot {Les Lettres du Début}

PM(TD) : retour : LLD(SED(TD))

Définitions récurrentes

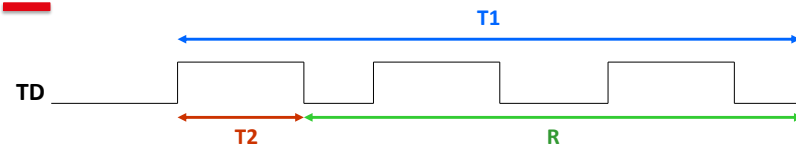
- (1) SED([]) = [] (2) SED(coT) = si c ≠ espace alors coT sinon SED(T)
 (3) LLD([]) = [] (4) LLD(coT) = si c = espace alors [] sinon co LLD(T)

5-37

P.-C. Scholl, C. Vigouroux – décembre 2025

Les mots d'une phrase

Réalisation récursive de LesMots



LesMots(TD) :

retour :

soit T1 = SED(TD)

dans soit T2 = LLD(T1), R = ALD(T1)

dans si EstVide(T2) alors []

sinon

T2 o LesMots(R)

Exercice

Étudier une solution itérative (texte sous forme contiguë)

5-39

P.-C. Scholl, C. Vigouroux – décembre 2025

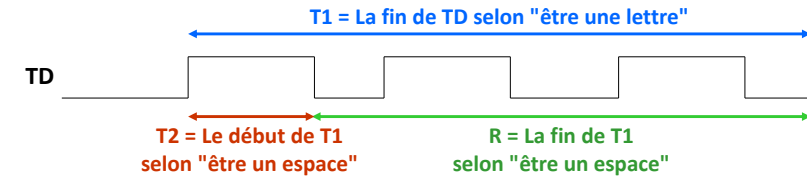
Les mots d'une phrase

Texte formé de lettres et d'espaces

Polycopié E5.9

Idée de départ

Les mots de TD = premier mot de TD suivi par les mots du reste de TD



LesMots : fonction (TD : texte) → séquence de (mot non vide)

SED : fonction (TD : texte) → texte {Sans Espace au Début}

LLD : fonction (TD : texte) → mot {Les Lettres du Début}

ALD : fonction (TD : texte) → texte {Après les Lettres du Début}

- (5) ALD([]) = [] (6) ALD(coT) = si c = espace alors coT sinon ALD(T)

5-38

P.-C. Scholl, C. Vigouroux – décembre 2025

b) Schémas

Polycopié § 5.A.d

Spécifications

Élément : **type** ; Séq : **type** séquence d'Élément

P : fonction (E : Élément) → booléen

LeDébutSelonP : fonction (SD : Séq, •••) → Séq

LaFinSelonP : fonction (SD : Séq, •••) → Séq

Définitions récurrentes

- (1) LeDébutSelonP([], •••) = []
 (2) LeDébutSelonP(eoS, •••) = si P(e) alors [] sinon eo LeDébutSelonP(S, •••)
 (3) LaFinSelonP([], •••) = []
 (4) LaFinSelonP(eoS, •••) = si P(e) alors eoS sinon LaFinSelonP(S, •••)

5-40

P.-C. Scholl, C. Vigouroux – décembre 2025

Devoir à la maison

A faire en binôme

A rendre en TD

- E5.30, "Sous-séquences"
- E5.39, "Somme d'une suite de nombres"

5.4 Compléments

Définitions récursives

5.0. Introduction

5.1. Définitions récursives de types et de fonctions

5.2. Modèles d'analyse récurrente

5.3. Découpage d'une séquence selon une propriété

5.4. Compléments

5-41

P.-C. Scholl, C. Vigouroux – décembre 2025

a) Couples d'éléments consécutifs

Une séquence est-elle croissante ?

Polycopié E5.5

Élément : **type** ; Séq : **type séquence d'Élément**
EstCrois : **fonction** (SD : Séq **non vide**) → **booléen**

- (1) EstCrois([e]) = **vrai**
- (2) EstCrois([e1,e2]&S) = $e1 \leq e2$ **et** EstCrois(e2oS)

Schéma pour le traitement de couples d'éléments consécutifs

- (1) f([]) = •••
- (2) f([e]) = •••
- (3) f([e1,e2]&S) = ••• f(e2oS) •••

Observer les rôles différents des deux cas de base

5-42

P.-C. Scholl, C. Vigouroux – décembre 2025

b) À propos de séquences triées

Insertion, suppression

Plus : **fonction** (SD : Séq, E : Élément) → Séq {insertion}
Moins : **fonction** (SD : Séq, E : Élément) → Séq {suppression}

Insertion

- (1) Plus([], e) = [e]
- (2) Plus(e1oS, e2) =
selon e1, e2
 $e2 < e1$: e2oS e1oS
 $e2 \geq e1$: e1o Plus(S, e2)
{Conservation de l'ordre d'arrivée en cas d'égalité}

Suppression

- (1) Moins([], e) = []
- (2) Moins(e1oS, e2) =
selon e1, e2
 $e2 < e1$: e1oS
 $e1 = e2$: S
 $e2 > e1$: e1o Moins(S, e2)

Erreur ?

5-43

P.-C. Scholl, C. Vigouroux – décembre 2025

5-44

P.-C. Scholl, C. Vigouroux – décembre 2025

b) À propos de séquences triées

Classement par insertion

Principe

Isoler un élément et l'insérer en bonne place après avoir classé les autres

PO : fonction (SD : Séq) → Séq {Permutation Ordonnée}

(1) PO([]) = []
(2) PO(eoS) = Plus(PO(S), e)
(2') PO(S•e) = Plus(PO(S), e)

Observer l'ordre dans lequel les éléments sont traités selon que l'on utilise (2) ou (2')

5-45

P.-C. Scholl, C. Vigouroux – décembre 2025

c) Calculs redondants

PO(SD) :
retour : si EstVide?(SD) alors []
sinon PO(SansUn(SD, Max(SD))) • Max(SD)

PO(SD) :
retour : si EstVide?(SD) alors []
sinon soit M = Max(SD) dans PO(SansUn(SD, M)) • M

fonction
regroupant SansUn
et Max

MaxEtSans : fonction (SD: Séq non vide) → Élément, Séq
{MaxEtSans(SD) = <Max(SD), SansUn(SD, Max(SD))>}

PO(SD) :
retour : si EstVide?(SD) alors []
sinon soit <M,R> = MaxEtSans(SD) dans PO(R) • M

(1) MaxEtSans([e]) = <e, []>
(2) MaxEtSans(eoS) = soit <M,R> = MaxEtSans(S) {S non vide}
dans si e > M alors <e, MoR> sinon <M, eoR>

5-47

P.-C. Scholl, C. Vigouroux – décembre 2025

b) À propos de séquences triées

Classement par sélection du maximum

Complément Polycopié E5.36

Principe

Isoler un élément de valeur maximum, le placer en queue, et classer le reste

Fonctions intermédiaires

Max : fonction (SD : Séq non vide) → Élément {valeur maximum}
SansUn : fonction (SD : Séq non vide, E : Élément) → Séq {Sans un exemplaire}

PO(SD) :
retour : si EstVide?(SD) alors []
sinon PO(SansUn(SD, Max(SD))) • Max(SD)

(1) Max([e]) = e
(2) Max(e1oeoS) = Max(si e1 < e2 alors e2oS sinon e1oS)
(3) SansUn([e1], e2) = si e1 = e2 alors [] sinon [e1]
(4) SansUn(e1oS, e2) = si e1 = e2 alors S sinon e1oSansUn(S, e2) {S non vide}

5-46

P.-C. Scholl, C. Vigouroux – décembre 2025

Sous-séquences d'une séquence

Enumérer les sous-séquences

Complément Polycopié E5.30

[10, 20, 30]

Selon le premier
[10], [10, 20], [10, 20, 30], [10, 30]
[20], [20, 30]
[30]

Par longueurs
[10], [20], [30]
[10, 20], [10, 30], [20, 30]
[10, 20, 30]

Idée de départ

Répartir l'ensemble E des sous-séquences selon qu'elles commencent (E1) ou non (E2) par le premier élément de la séquence donnée : E = E1 ∪ E2

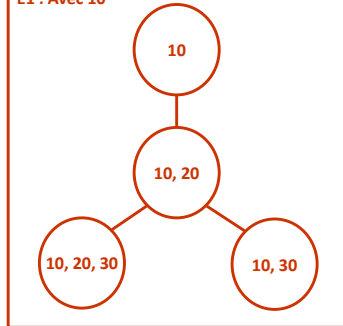
5-48

P.-C. Scholl, C. Vigouroux – décembre 2025

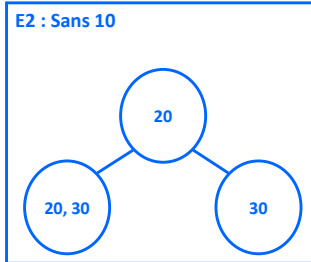
Sous-séquences d'une séquence

Recherche de la récurrence

E1 : Avec 10



E2 : Sans 10



Observations (Eureka!)

E2 = les sous-séquences de [20, 30]

E1 = { [10] } $\cup$ (10 $\oplus$ E2)

5-49

P.-C. Scholl, C. Vigouroux – décembre 2025

Le compte est bon

Une forme simplifiée

Polycopié E5.35

Exprimer E comme une somme d'éléments de SD

E : entier > 0 ; SD : séquence d'entiers > 0

- Existence d'une solution
- Nombre de solutions
- Construction d'une solution
- Construction de toutes les solutions

E = 20, SD = [18, 3, 1, 4, 20]

E = 25, SD = [21, 3, 2, 4, 2]

E = 23, SD = [21, 3, 2, 4, 2]

Idées

1. Construire l'ensemble des sous-séquences
 - fonctions intermédiaires : LesSousSéq, Somme
2. Analyse récurrente directe
 - choisir un élément de SD
 - répartir l'ensemble des solutions selon qu'elles comportent cet élément ou non

5-51

P.-C. Scholl, C. Vigouroux – décembre 2025

Sous-séquences d'une séquence

Equations de récurrence

Élément : type ; Séq : type séquence d'Éléments

LesSousSéq : fonction (SD : Séq) $\rightarrow$ séquence de (Séq non vide)

PlusG : fonction (E : Élément, SSD : séquence de (Séq non vide))

$\rightarrow$ séquence de (Séq non vide)

{Séquence de séquences obtenue à partir de SSD en ajoutant e à gauche de toutes les séquences de SSD. PlusG(E, []) = []}

(1) LesSousSéq([]) = []

(2) LesSousSéq(eoS) = soit E2 = LesSousSéq(S) dans [[e]] & PlusG(e, E2) & E2

(3) PlusG(e, []) = []

(4) PlusG(e, soS) = (eoS) $\circ$ PlusG(e, S)

Nombre de sous-séquences d'une séquence de longueur n

Montrer par récurrence que c'est $2^n - 1$

5-50

P.-C. Scholl, C. Vigouroux – décembre 2025

Le compte est bon

Spécifications

SéqE : type séquence d'entier > 0

ExisteSol : fonction (SD : SéqE, E : entier > 0) $\rightarrow$ booléen

NbSol : fonction (SD : SéqE, E : entier > 0) $\rightarrow$ entier $\geq$ 0

UneSol : fonction (SD : SéqE, E : entier > 0) $\rightarrow$ SéqE

{UneSol(SD, E) = [] $\Leftrightarrow$ aucune solution}

LesSol : fonction (SD : SéqE, E : entier > 0) $\rightarrow$ séquence de (SéqE non vide)

{LesSol(SD, E) = [] $\Leftrightarrow$ aucune solution}

Forme générale des équations

(1) F([], E) =

{aucune solution}

(2) F(S $\bullet$ e, E) = selon S, e

e > E : {pas de solution avec e}

e = E : {au moins une solution : [e]}

e < E : {une solution avec e ? Une solution sans e?}

5-52

P.-C. Scholl, C. Vigouroux – décembre 2025

Le compte est bon

Analyse par cas

	F([], E)	F(S•e, E)		
		e > E	e = E	e < E
ExisteSol	Faux	ExisteSol(S,E)	Vrai	ExisteSol(S,E-e) ou ExisteSol(S,E)
NbSol	0	NbSol(S,E)	1 + NbSol(S,E)	NbSol(S,E-e) + NbSol(S,E)
UneSol	[]	UneSol(S,E)	[e]	soit Z = UneSol(S,E-e) dans si Z ≠ [] alors Z•e sinon UneSol(S,E)
LesSol	[]	LesSol(S,E)	[[e]] & LesSol(S,E)	PlusD(LesSol(S,E-e),e) & LesSol(S,E)

PlusD : fonction (E:Élément, SSD:séquence de (Séq non vide)) → séquence de (Séq non vide)
{Séquence de séquences obtenue à partir de SSD en ajoutant e à droite de toutes les séquences de SSD. PlusD(E, []) = []}

5-53

P.-C. Scholl, C. Vigouroux – décembre 2025

Le compte est bon

Construction d'une solution, de toutes les solutions

(1) UneSol([], E) = []
 (2) UneSol(S•e, E) = selon S, e
 e > E : UneSol(S, E)
 e = E : [e]
 e < E : soit Sol = UneSol(S, E-e)
 dans si EstVide?(Sol) alors UneSol(S, E)
 sinon Sol•e

(1) LesSol([], E) = []
 (2) LesSol(S•e, E) = selon S, e
 e > E : LesSol(S, E)
 e = E : [[e]] & LesSol(S, E)
 e < E : PlusD(LesSol(S,E-e), e) & LesSol(S, E)

5-55

P.-C. Scholl, C. Vigouroux – décembre 2025

Le compte est bon

Existence d'une solution, nombre de solutions

(1) ExisteSol([], E) = faux
 (2) ExisteSol(S•e, E) = selon S, e
 e > E : ExisteSol(S, E)
 e = E : vrai
 e < E : ExisteSol(S, E-e) ou ExisteSol(S, E)

(1) NbSol([], E) = 0
 (2) NbSol(S•e, E) = selon S, e
 e > E : NbSol(S, E)
 e = E : 1 + NbSol(S, E)
 e < E : NbSol(S, E-e) + NbSol(S, E)

5-54

P.-C. Scholl, C. Vigouroux – décembre 2025

Conclusion du chapitre 5

Éléments pour guider le travail hors séance

- Connaître les définitions
- Comprendre les notions sous-jacentes, savoir les utiliser
 - définition récursive des entiers, des séquences
 - constructeur, sélecteur, testeur d'un type défini récursivement
 - définition récurrente d'une fonction - liste d'appels récursifs
- Connaître les modèles d'analyse récurrente
 - fonctions à paramètres de type entier, de type séquence
 - découpage d'une séquence selon une propriété
- Savoir
 - Analyser un problème et exprimer une solution de manière fonctionnelle
 - Appliquer le plan général d'analyse récurrente
 - propriétés → définition récurrente → réalisation récursive
 - Appliquer les modèles d'analyse récurrente
 - Observer l'évaluation d'une fonction
 - réécriture, liste des appels récursifs
 - Et toujours
 - Exprimer un principe de solution, vérifier une solution

5-56

P.-C. Scholl, C. Vigouroux – décembre 2025

Savoir donner une définition ,un exemple

- fonction versus action - expression fonctionnelle
- définition récursive d'un type
- définition récurrente d'une fonction
- constructeur, sélecteur, testeur d'un type
- procédure récursive – récursivité croisée

- modèle d'analyse récurrente
- découpage d'une séquence selon une propriété
- regroupement d'équations
- liste d'appels récursifs – nombre d'appels récursifs engendrés
- redondance de calculs

6. Arbres : une introduction

6.1. Structuration en arbre

Découverte d'une structure fondamentale

Renforcement de l'analyse récurrente

6.1. Structuration en arbre

6.2. Arbres binaires



Connaître la terminologie, les définitions
S'appropriier les notations

a) Arbres n-aires

Définition constructive récursive

Pour structurer un ensemble non vide en arbre

- Choisir un élément
- Répartir les autres éléments (s'il en reste) en sous-ensembles **disjoints**, et les **structurer en arbre** à leur tour

Définition

Un **arbre n-aire** est formé
d'une **racine**
et de **n sous-arbres n-aires** ($n \geq 0$)

6-1

P.-C. Scholl, C. Vigouroux – janvier 2026

6-2

P.-C. Scholl, C. Vigouroux – janvier 2026

a) Arbres n-aires

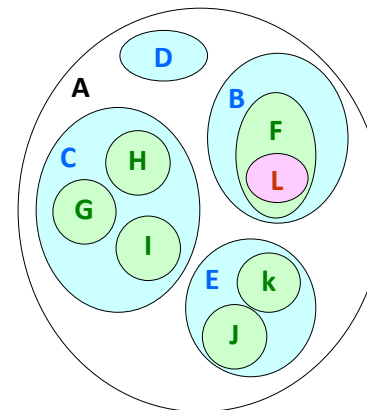
Exemples

- Sommaire d'un livre
 - ensemble de titres (livre, section, sous-section, etc.)
- Expressions algébriques
 - ensemble de noms (opérateurs, constantes, identificateurs)
- Arborescence de fichiers
 - ensemble de noms (dossiers, fichiers)
- Arbre de décision
 - ensemble de questions
- Classification
 - ensemble de noms de classes

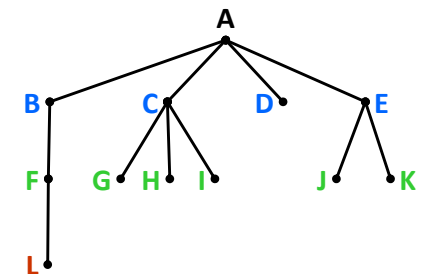
b) Diverses représentations

Ensembles emboîtés, graphe

Ensembles emboîtés



Graphe



6-3

P.-C. Scholl, C. Vigouroux – janvier 2026

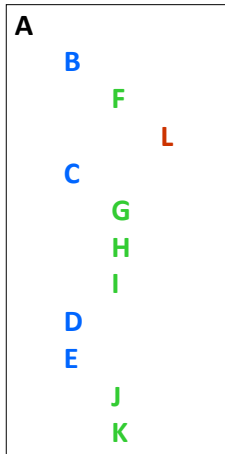
6-4

P.-C. Scholl, C. Vigouroux – janvier 2026

b) Diverses représentations

Indentation, formes linéaires parenthésées

Indentation



Préfixé

racine puis sous-arbres

(A (B (F (L))) (C (G) (H) (I)) (D) (E (J) (K)))

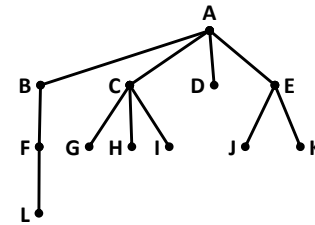
Postfixé

sous-arbres puis racine

((((L) F) B) ((G) (H) (I) C) (D) ((J) (K) E) A)

c) Terminologie

Relation père-fils



Définitions

- A est (à) la **racine** de l'arbre
- C est **père** de H ; H est **fils** de C
- G, H, I sont **frères**
- F est **petit-fils** de A ; A est **grand-père** de F
- G et K sont **cousins** (G et H ne le sont pas)
- F et L forment la **descendance** de B
- L, H et K sont trois **descendants** de A
- F, B et A forment l'**ascendance** de L

6-5

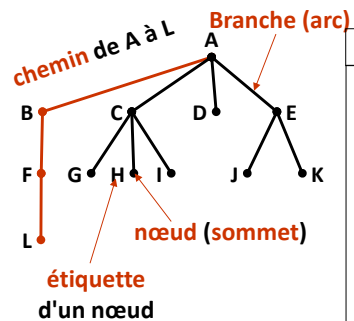
P.-C. Scholl, C. Vigouroux – janvier 2026

6-6

P.-C. Scholl, C. Vigouroux – janvier 2026

c) Terminologie

Nœuds et arcs



Définitions

- Feuille** (nœud **externe** ou **terminal**) :
nœud qui n'a pas de fils
- Longueur** d'un chemin :
nombre de **nœuds** sur le chemin
- Niveau** d'un nœud :
Longueur du chemin de la racine au nœud
 - niveau de la racine = 1
 - niveau d'un nœud = 1 + niveau du père
- Profondeur** d'un arbre : niveau maximum

6.2. Arbres binaires

Arbres

6.1. Structuration en arbre

6.2. Arbres binaires



Comprendre la différence entre arbre binaire et arbre n-aire
Maîtriser les modèles d'analyse récurrente

6-7

P.-C. Scholl, C. Vigouroux – janvier 2026

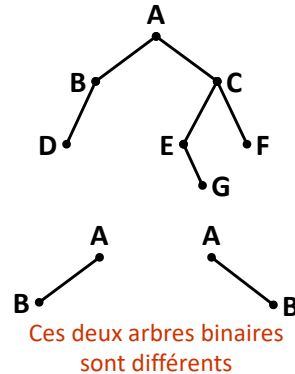
6-8

P.-C. Scholl, C. Vigouroux – janvier 2026

a) Définition récursive des arbres binaires

Idée intuitive

- Tout nœud a deux sous-arbres
 - un sous-arbre **gauche** et un sous-arbre **droit**
- Un sous-arbre peut être vide
 - un nœud a au plus deux fils



Définition

Un arbre binaire est
vide
ou formé
d'une racine,
d'un sous-arbre gauche
et d'un sous-arbre droit

6-9

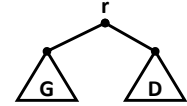
P.-C. Scholl, C. Vigouroux – janvier 2026

b) Le constructeur de types Arbre Binaire

Point de vue langage

Définitions

- $/\backslash$ dénote l'arbre vide
- $/G, r, D\backslash$ dénote un arbre binaire
 - de racine r
 - de sous-arbre gauche G
 - de sous-arbre droit D .
- Constructeurs : $/\backslash, / \bullet \bullet \bullet, \bullet \bullet \bullet \backslash$
- Sélecteurs : **Racine**, **Gauche**, **Droit**
- Testeur : **EstVide?**



$A = /G, r, D\backslash \Leftrightarrow \text{non EstVide?}(A)$ et

$G = \text{Gauche}(A)$
 $r = \text{Racine}(A)$
 $D = \text{Droit}(A)$

6-10

P.-C. Scholl, C. Vigouroux – janvier 2026

Nombre d'éléments positifs

Analyse

NbP : fonction $(A : \text{arbre binaire d'entier}) \rightarrow \text{entier} \geq 0$

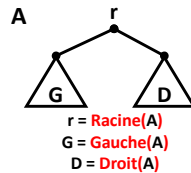
- Analyse récursive
 1. $\text{NbP}(/ \backslash) = 0$
 2. $\text{NbP}(/G, r, D\backslash) = (\text{si } r \geq 0 \text{ alors } 1 \text{ sinon } 0) + \text{NbP}(G) + \text{NbP}(D)$

Réalisation récursive

NbP (A) :

retour:

si EstVide?(A) **alors** 0
sinon (**si** Racine(A) ≥ 0 **alors** 1 **sinon** 0)
+ NbP(Gauche(A)) + NbP(Droit(A))



6-11

P.-C. Scholl, C. Vigouroux – janvier 2026

Nombre d'éléments positifs

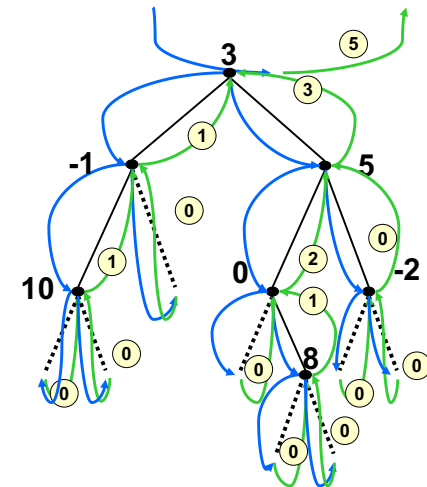
Processus d'évaluation postfixée

$\text{NbP}(/ \backslash) = 0$

$\text{NbP}(/G, r, D\backslash) =$

(**si** $r \geq 0$ **alors** 1 **sinon** 0)
+ NbP(G) + NbP(D)

Mise en attente des additions
lors de la "descente" vers les
feuilles

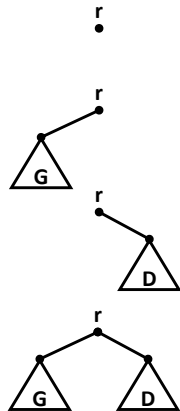


6-12

P.-C. Scholl, C. Vigouroux – janvier 2026

c) Arbres binaires non vides

Définition récursive

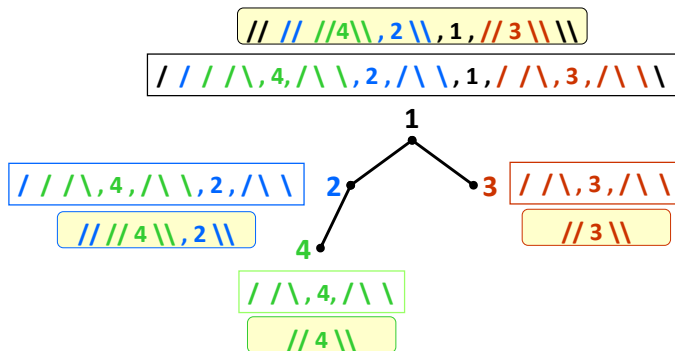


- Une racine
nœud **singleton**
- Une racine et un sous-arbre gauche **non vide**
nœud **unaire gauche**
- Une racine et un sous-arbre droit **non vide**
nœud **unaire droit**
- Une racine et deux sous-arbres **non vides**
nœud **binaire**

6-13

P.-C. Scholl, C. Vigouroux – janvier 2026

Expression à valeur arbre



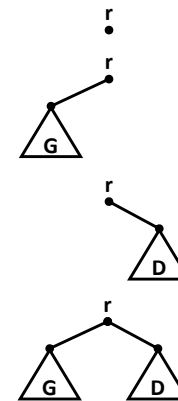
Formes abrégées

6-15

P.-C. Scholl, C. Vigouroux – janvier 2026

c) Arbres binaires non vides

Notations abrégées



- $///, r, \\$ $//r\\$
- $/G, r, \\$ $//G, r\\$
 $\{G \text{ non vide}\}$
- $///, r, D\\$ $//r, D\\$
 $\{D \text{ non vide}\}$
- $/G, r, D\\$ $//G, r, D\\$
 $\{G \text{ et } D \text{ non vides}\}$

6-14

P.-C. Scholl, C. Vigouroux – janvier 2026

d) Analyse récurrente

Modèles d'analyse

Modèle 1 en incluant l'arbre vide	Modèle 2 en excluant l'arbre vide
$F(/ \backslash) = \dots$	$F(//r \backslash) = \dots$
$F(/G, r, D \backslash) = \dots$	$F(//G, r \backslash) = \dots$
	$F(//r, D \backslash) = \dots$
	$F(//G, r, D \backslash) = \dots$

6-16

P.-C. Scholl, C. Vigouroux – janvier 2026

Les appels récursifs engendrés ont une structure d'arbre correspondant à l'arbre traité

Nombre d'appels récursifs engendrés ?

Cas du "parcours" d'un arbre de N noeuds

Modèle 1 : 2N

Modèle 2 : N-1

Nb1 : fonction (A : arbre binaire d'entier) $\rightarrow$ entier ≥ 0

(1) $Nb1(/ \setminus) = 0$

$$(2) \text{Nb1}(\text{G}, r, D) = 1 + \text{Nb1}(\text{G}) + \text{Nb1}(D)$$

Nombre d'appels
récur­sifs engendrés ?

Pour tout arbre non vide : 2 appels récurifs

N nœuds $\Rightarrow$ N arbres non vides : donc **2N** appels récurifs engendrés

Au total (y compris appel initial) : $2N+1$

Nb2: fonction (A : arbre binaire non vide d'entier) → entier >0

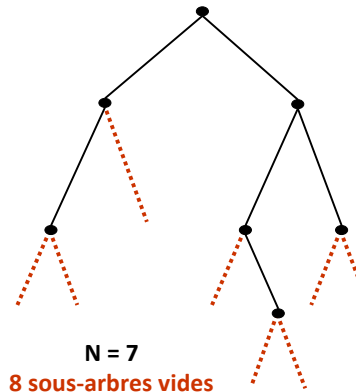
$$(1) \text{Nb}_2(r) = 1$$
$$(2) \text{Nb}_2(\text{G}, r) = 1 + \text{Nb}_2(\text{G})$$
$$(3) \text{Nb}_2(\text{r}, \text{D}) = 1 + \text{Nb}_2(\text{D})$$
$$(4) \text{Nb}_2(\text{G}, r, D) = 1 + \text{Nb}_2(G) + \text{Nb}_2(D)$$

Nombre d'appels
récur­sifs engendrés ?

Pour tout nœud fils : 1 appel récursif

N nœuds $\Rightarrow$ N-1 nœuds fils : donc **N-1** appels récurifs engendrés

Au total (y compris appel initial) : N



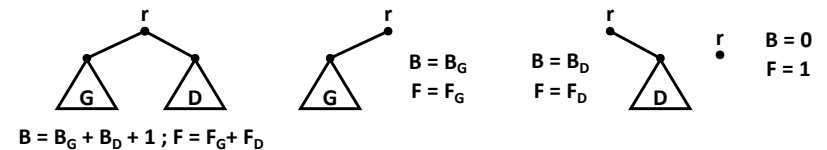
Un arbre non vide de N nœuds
a $N+1$ sous-arbres vides

- Nombre de sous-arbres : $2N$
- Nombre de sous-arbres non vides : $N - 1$
tout nœud est fils, sauf la racine
- Nombre de sous-arbres vides
 $(2N) - (N - 1) = N + 1$

- **Base** : montrer que la propriété est vraie pour l'arbre singleton
- **Hypothèse de récurrence** : la propriété est vraie pour les sous-arbres
- **Hérédité** : En déduire qu'elle est vraie pour l'arbre

Example

B nœuds binaires, U nœuds unaires, F feuilles

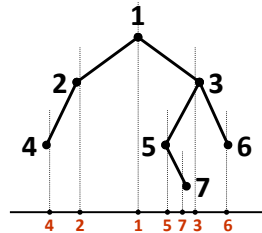
$$B = F^{-1} (\forall N > 0)$$


H.R. : $B_G = F_G - 1$; $B_D = F_D - 1$
donc : $B = F - 1$

f) Parcours d'arbres

Définitions

- **Préordre** : racine, sous-arbre gauche, sous-arbre droit
- **Ordre symétrique** : sous-arbre gauche, racine, sous-arbre droit
- **Ordre terminal** : sous-arbre gauche, sous-arbre droit, racine



Préordre : 1, 2, 4, 3, 5, 7, 6
 Ordre symétrique : 4, 2, 1, 5, 7, 3, 6
 Ordre terminal : 4, 2, 7, 5, 6, 3, 1

6-21

P.-C. Scholl, C. Vigouroux – janvier 2026

g) Recherche en arbre

Spécifications

- Existence d'un élément vérifiant P
- Extraction d'un élément vérifiant P
- Extraction du premier vérifiant P selon un ordre de parcours

Élément : **type**

ArbreBin : **type arbre binaire d'Élément**

P : **fonction** (E : Élément) → **booléen**

ExisteP : **fonction** (A : ArbreBin, ●●●) → **booléen**

UnP : **fonction** (A : ArbreBin, ●●●) → **booléen**, Élément

PremierPsym : **fonction** (A : ArbreBin, ●●●) → **booléen**, Élément

6-23

P.-C. Scholl, C. Vigouroux – janvier 2026

f) Parcours d'arbres

Linéarisation d'un arbre binaire

SéqSym : **fonction** (A : **arbre binaire d'Élément**) → **séquence d'Élément**
 { Séquence des nœuds de l'arbre en ordre symétrique }

(1) SéqSym (/ \) = []

(2) SéqSym (/G, r, D\) = SéqSym(G) & [r] & SéqSym(D)

6-22

P.-C. Scholl, C. Vigouroux – janvier 2026

g) Recherche en arbre

Equations

(1) ExisteP(/ \, ●●●) = **faux**

(2) ExisteP(/G, r, D\, ●●●) = P(r) ou ExisteP(G, ●●●) ou ExisteP(D, ●●●)

(1) UnP(/ \, ●●●) = <faux, ?>

(2) UnP(/G, r, D\, ●●●) =

si P(r) alors <vrai, r>

sinon soit <trouvé, E> = UnP(G, ●●●)

dans si trouvé alors <vrai, E> sinon UnP(D, ●●●)

(1) PremierPsym(/ \, ●●●) = <faux, ?>

(2) PremierPsym(/G, r, D\, ●●●) =

soit <trouvé, E> = PremierPsym(G, ●●●)

dans si trouvé alors <vrai, E>

sinon si P(r) alors <vrai, r> sinon PremierPsym(D, ●●●)

6-24

P.-C. Scholl, C. Vigouroux – janvier 2026

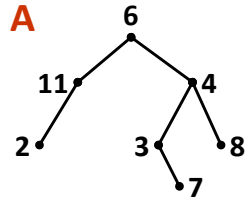
Nombre de descendants

Spécification

NbDesc : fonction ($A : \text{ArbreBin}, e : \text{Élément}$) $\rightarrow$ entier

{ Nombre de descendants de l'élément de valeur z dans A ou -1 si e n'est pas dans A .

Pré-condition : éléments de A distincts deux à deux }



NbDesc ($A, 4$) = 3

NbDesc ($A, 7$) = 0

NbDesc ($A, 5$) = -1

NbN : fonction ($A : \text{ArbreBin}$) $\rightarrow$ entier ≥ 0

{ Nombre de nœuds de l'arbre A }

6-25

P.-C. Scholl, C. Vigouroux – janvier 2026

h) Chemin dans un arbre

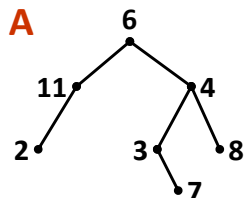
Définitions

- Chemin d'accès à un nœud : chemin de la racine au nœud
- Valeur d'un chemin : séquence des étiquettes associées aux nœuds ou aux arcs des chemins (dans l'ordre père $\rightarrow$ fils)

ChA : fonction ($A : \text{ArbreBin}, e : \text{Élément}$) $\rightarrow$ Séquence d'Élément

{ Chemin d'accès au nœud d'étiquette e ou la séquence vide si $e \notin A$.

Pré-condition : éléments de A distincts deux à deux }



ChA ($A, 7$) = [6, 4, 3, 7]

ChA($A, 5$) = []

6-27

P.-C. Scholl, C. Vigouroux – janvier 2026

Nombre de descendants

Équations

Solution 1

(1) NbDesc($/ \backslash, e$) = -1

(2) NbDesc($/G, r, D \backslash, e$) =

si $r = e$ alors NbN(G) + NbN(D)

sinon soit $N = \text{NbDesc}(G, e)$

dans si $N = -1$ alors NbDesc(D, e) sinon N

Solution 2

(1) NbDesc($/ \backslash, e$) = -1

(2) NbDesc($/G, r, D \backslash, e$) =

si $r = e$ alors NbN(G) + NbN(D)

sinon NbDesc(G, e) + NbDesc(D, e) + 1

Est-elle correcte ? Démonstration par récurrence

6-26

P.-C. Scholl, C. Vigouroux – janvier 2026

Chemin d'accès à un nœud

Équations

(1) ChA($/ \backslash, e$) = []

(2) ChA($/G, r, D \backslash, e$) =

si $r = e$ alors [r]

sinon soit $C = \text{ChA}(G, e)$

dans si $C \neq []$ alors C

sinon ChA(D, e)

si $C \neq []$ alors $r \circ C$

sinon $r \circ \text{ChA}(D, e)$

Erreurs ?

Erreurs ?

si $C \neq []$ alors $r \circ C$

sinon soit $C = \text{ChA}(D, e)$

dans si $C \neq []$ alors $r \circ C$

sinon []

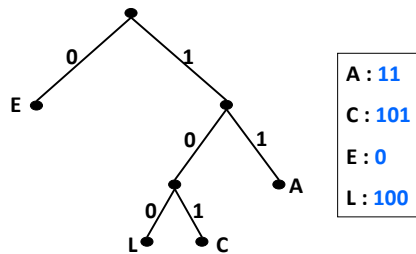
6-28

P.-C. Scholl, C. Vigouroux – janvier 2026

Codage d'un texte

Arbre binaire étiqueté

- arcs : 0 pour gauche, 1 pour droit
- feuilles : ensemble des lettres du texte ; nœuds internes : pas d'étiquette
- Code d'une lettre : valeur du chemin d'accès à la lettre



LEAALACLE
10001111100111011000
0100100011101111000
ELLEACALE

6-29

P.-C. Scholl, C. Vigouroux – janvier 2026

Vocabulaire

Savoir donner une définition ,un exemple

- arbre binaire, arbre n-aire, sous-arbre, racine, feuille
- sommet, nœud, étiquette, arc
- chemin, longueur d'un chemin, niveau d'un nœud, profondeur
- nœud singleton, nœud unaire, nœud binaire
- relations entre nœuds : père, fils, frère, cousin, ...
- descendance, ascendance
- chemin d'accès à un nœud, valeur d'un chemin
- parcours d'arbres binaires
 - Préordre, ordre symétrique, ordre terminal
- recherche en arbre

6-31

P.-C. Scholl, C. Vigouroux – janvier 2026

Conclusion du chapitre 6

Éléments pour guider le travail hors séance

- **Connaître les définitions**
 - Comprendre les notions sous-jacentes, savoir les utiliser
 - arbre n-aire, terminologie sur les arbres
 - définitions récursives des arbres binaires (2)
 - constructeurs, sélecteurs, testeurs pour les arbres binaires
- **Connaître les schémas de base**
 - modèles d'analyse récurrente
 - parcours d'arbres
 - recherche en arbre
- **Savoir**
 - Analyser un problème et exprimer une solution de manière fonctionnelle
 - Appliquer le plan général d'analyse récurrente
 - Appliquer les modèles d'analyse récurrente
 - Exprimer un principe de solution, vérifier une solution
 - Observer l'évaluation d'une fonction

6-30

P.-C. Scholl, C. Vigouroux – janvier 2026

7. Composition récursive

Récursivité : du fonctionnel à l'actionnel

Raisonnement sur les actions : compléments, renforcement

7.1. Réalisation récursive d'une action

7.2. Traitement récursif des séquences

7.3. Du récursif à l'itératif

Revoir : chapitres 3 et 4

7-1

P.-C. Scholl, C. Vigouroux – février 2026

Forme binaire d'un entier

Action d'affichage : réalisation récursive

Analyse fonctionnelle

EntVersBin : fonction (E : entier > 0) $\rightarrow$ texte {formé de '0' et de '1'}

(1) EntVersBin (1) = "1"

(2) EntVersBin ($2 \cdot n$) = EntVersBin(n) • '0' { $n > 0$ }

(3) EntVersBin ($2 \cdot n + 1$) = EntVersBin(n) • '1' { $n > 0$ }

Réalisation récursive de l'action

AfBin1(E) :

si $E = 1$ alors AfCar('1')

sinon AfBin1(E quotient 2)

AfCar (si pair(E) alors '0' sinon '1')

Dans le cas de AfBin1(13), l'affichage est "1101".

Quelle est l'instruction AfCar qui provoque l'affichage

- du dernier caractère du texte ?
- du premier caractère du texte ?

7-3

P.-C. Scholl, C. Vigouroux – février 2026

Forme binaire d'un entier

Énoncé

Polycopié E7.1

Afficher la forme binaire d'un entier > 0

- Contexte : machine d'affichage (polycopié §A.a)
 - Écran alphanumérique : état caractérisé par ce qui est affiché et la position du curseur sur la ligne courante
 - Primitives utiles : ViderEcran, AfCar(C), ALaLigne

Exemple

Donnée : 13

Résultat affiché : le texte "1101"

Spécification

AfBin1 : action (donnée E : entier > 0)

{ Affiche la forme binaire de E .

e.i. : soit p la position initiale du curseur sur l'écran ;

e.f. : forme binaire de E affichée à partir de la position p ; le curseur se trouve à la position suivant directement le dernier caractère affiché. }

7-2

P.-C. Scholl, C. Vigouroux – février 2026

Forme binaire d'un entier

Observation de l'exécution

Réécriture

AfBin1(13) $\rightarrow$

AfBin1(6) ; AfCar('1')

$\rightarrow$

AfBin1(3) ; AfCar('0') ; AfCar('1')

Cas de base $\rightarrow$

AfBin1(1) ; AfCar('1') ; AfCar('0') ; AfCar('1')

$\rightarrow$

AfCar('1') ; AfCar('1') ; AfCar('0') ; AfCar('1')

AfBin1(13)

AfBin1(6)

AfBin1(3)

AfBin1(1)

AfCar('1')

AfCar('1')

AfCar('0')

AfCar('1')

Appels successifs : mise en attente d'actions

Cas de base

Exécution des actions mises en attente dans l'ordre inverse de la mise en attente

7-4

P.-C. Scholl, C. Vigouroux – février 2026

Forme binaire d'un entier

Prise en compte de 0

AfBin2 : action (donnée E : entier ≥ 0)

AfBin2(E) : {version 1}

si E = 0 alors AfCar('0')

sinon si E=1 alors AfCar('1')

sinon AfBin2(E quotient 2)

AfCar (si pair(E) alors '0' sinon '1')

Effet de AfBin2(13) ?

01101

Comment éviter
le 0 de gauche ?

Une autre solution ?

AfBin2(E) : {version 2}

si E = 0 alors AfCar('0')

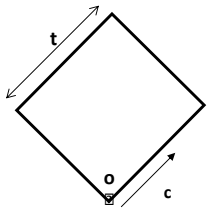
sinon AfBin1(E)

7-5

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

Spécification d'une figure

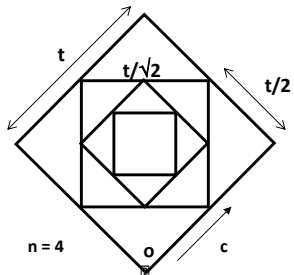


Spécification d'un carré

t : taille du côté, c : cap du carré

o : origine du carré

L'observateur voit le carré à sa gauche
quand, placé au point O, il regarde dans le
sens de C.



Spécification de la figure

t, c, o : carré extérieur, n : nombre de carrés

Les carrés se joignent au milieu des côtés.

L'observateur voit la figure à sa gauche ...

7-7

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

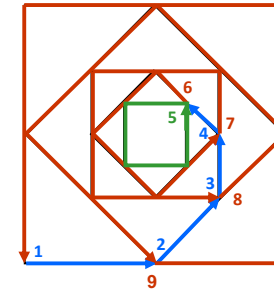
Énoncé

Polycopié E7.2

Dessiner une figure formée de carrés emboîtés, sans
lever la plume et sans tracer deux fois le même trait.

- Contexte : machine tracés (cf polycopié)
- Exemple

Tracé à la main ?



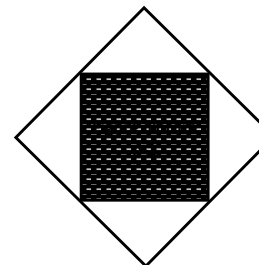
7-6

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

Analyse récurrente

Analyse de la figure



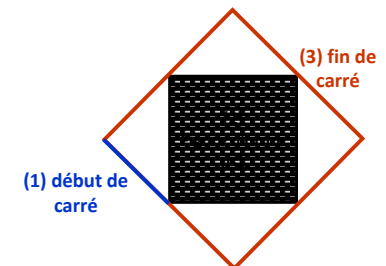
n > 1 : cas récursif



n = 1 : cas de base

Analyse du tracé avec contrainte

- Diviser le tracé du carré extérieur :
deux parties séparées par l'un des milieux de côté
- Tracer successivement
début du carré extérieur ; figure intérieure ; fin du
carré extérieur



7-8

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

Spécification de l'action de tracé

Tcaremb : action (donnée n : entier > 0 , t : réel > 0)

{ Trace une figure de n carrés en respectant la contrainte

e.i. : plume basse à un point p et à un cap c

e.f. : la figure d'ordre n , de côté t , de sommet p , de cap c est tracée et la plume est dans l'état initial. }

Tdébutcar : action (donnée t : réel > 0)

Tfincar : action (donnée t : réel > 0)

{Tdébutcar(t) ; Tfincar(t) $\equiv$ Tcarré(t)}

7-9

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

Étapes de réalisation



1. Mettre en place la structure : cas de base, cas récurifs, appels récurifs
2. Fixer les paramètres des appels récurifs
3. Donner les pré-conditions et les post-conditions des appels récurifs
4. Compléter la réalisation: avant appel, après appel, cas de base

Tcaremb(n, t) :

{plume basse, au sommet O et au cap c de la figure}

si $n = 1$ alors

Tdébutcar(t) ; Tfincar(t)

sinon

Tdébutcar(t) ; TrG(45)

{plume basse, milieu coté, cap figure intérieure}

Tcaremb($n-1, t / \sqrt{2}$)

{figure intérieure tracée ;

plume basse, milieu coté, cap figure intérieure}

TrD(45) ; Tfincar(t)

{figure tracée, plume basse, au sommet O et au cap c }

État attendu pour le tracé de la figure intérieure par un appel récurif

État final garanti par l'action de tracé

7-10

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

Structure des appels récurifs

Alpha(n, t) : Tdébutcar(t) ; Tfincar(t)

Beta(n, t) : Tdébutcar(t) ; TrG(45)

Gamma(n, t) : TrD(45) ; Tfincar(t)

T(n, t) : $n-1, t / \sqrt{2}$

Liste des appels récurifs

(n, t), T(n, t), T(T(n, t)), ..., $T^{n-2}(n, t)$, $T^{n-1}(n, t)$

A(n, t) {figure d'ordre n , de côté t }

Beta(n, t)

A(T(n, t))

Beta(T(n, t))

A(T(T(n, t)))

...

Beta($T^{n-2}(n, t)$)

A($T^{n-1}(n, t)$)

Alpha($T^{n-1}(n, t)$)

Gamma($T^{n-2}(n, t)$)

...

Gamma(T(n, t))

Gamma(n, t)

Appels récurifs successifs : exécution des Beta, mise en attente des Gamma

Cas de base : exécution d'Alpha

Exécution des Gamma mises en attente, dans l'ordre inverse de la mise en attente

7-11

P.-C. Scholl, C. Vigouroux – février 2026

Carrés emboîtés

Vérification de la terminaison, analyse quantitative

• Prouver la terminaison

- Par récurrence sur n .
- Ou : montrer que le nombre d'appels récurifs est fini.

• Analyse quantitative

Tous les calculs s'appuient sur le nombre d'appels récurifs engendrés.

• Dénombrer les appels récurifs engendrés

- Définir une fonction :

NbAp(n) : nombre d'appels récurifs engendrés par Tcaremb(n, t).

- Équations de récurrence

On raisonne sur le texte de la réalisation

(1) NbAp(1) = 0

(2) NbAp($n+1$) = 1 + NbAp(n) { $n > 0$ }

- Conclusion

NbAp(n) = $n - 1$

7-12

P.-C. Scholl, C. Vigouroux – février 2026

Affichage d'un texte

Spécification, analyse récurrente

Polycopié E7.3

AfChaîne1 : **action** (donnée T : **texte**)

{ Affiche le texte T.

e.i. : soit p la position initiale du curseur sur l'écran ;

e.f. : le texte T est affiché à partir de la position p ;

le curseur se trouve sur la position suivant celle du dernier caractère affiché. Si le texte est vide : action vide. }

AfChaîne1(T) : {version 1}

si non EstVide?(T) alors

AfCar(**premier(T)**)

AfChaîne1(**fin(T)**)

AfChaîne1(T) : {version 2}

si non EstVide?(T) alors

AfChaîne1(**début(T)**)

AfCar(**dernier(T)**)



T = premier(T) o fin(T)



T = début(T) • dernier(T)

7-13

P.-C. Scholl, C. Vigouroux – février 2026

Affichage d'un texte

AfChaîne 2 : des réalisations

Idée 1 : modifier **AfChaîne1** en ajoutant des appels de **AlaLigne**.

AfChaîne2(T) : {version 1}

si non EstVide?(T) alors

AfCar(**premier(T)**)

AfChaîne2(**fin(T)**)

AlaLigne

sinon AlaLigne

AfChaîne2(T) : {version 2}

si non EstVide?(T) alors

AfChaîne2(**début(T)**)

AfCar(**dernier(T)**)

AlaLigne

sinon AlaLigne

Ces réalisations sont incorrectes

Nombre d'appels de **AlaLigne** ? Effet sur un exemple simple ?

Idée 2 : utiliser **AfChaîne1**

AfChaîne2(T) : {version 3}

AfChaîne1(**T**) ; AlaLigne

7-15

P.-C. Scholl, C. Vigouroux – février 2026

Affichage d'un texte

Une autre spécification

AfChaîne2 : **action** (donnée T : **texte**)

{ Affiche le texte T.

e.i. : soit p la position initiale du curseur sur l'écran ;

e.f. : le texte T est affiché à partir de la position p ;

*le curseur se trouve **au début de la ligne suivant** celle du dernier caractère affiché. Si le texte est vide, le curseur se trouve **au début de la ligne suivant** celle de l'état initial. }*

7-14

P.-C. Scholl, C. Vigouroux – février 2026

7.2. Traitement récursif des séquences

7. Composition récursive

7.1. Réalisation récursive d'une action

7.2. Traitement récursif des séquences

7.3. Du récursif à l'itératif

7-16

P.-C. Scholl, C. Vigouroux – février 2026

a) Représentation chaînée

Réalisation récursive

Le traitement d'une séquence est décrit en termes du même traitement portant sur la séquence de fin (la séquence donnée sans son premier élément)

Séquence en paramètre d'une action

L'adresse d'un élément E d'une séquence S donne accès à la sous-séquence de S

- commençant en E
- et se terminant par le dernier élément de S

7-17

P.-C. Scholl, C. Vigouroux – février 2026

Sélection des éléments positifs

Construction d'une séquence

Polycopié E7.5

Construire une séquence formée des entiers strictement positifs d'une séquence d'entiers donnée. L'ordre initial doit être maintenu.

Analyse récurrente (fonctionnelle)

SélPos : fonction (SD : séquence d'entier) → séquence d'entier > 0

(1) SélPos([]) = []

(2) SélPos(e o S) = si e > 0 alors e o SélPos(S) sinon SélPos(S)

7-19

P.-C. Scholl, C. Vigouroux – février 2026

Somme des éléments

Parcours d'une liste chaînée : réalisation récursive

Polycopié E7.4

adCel : type pointeur de Cel
Cel : type <E : entier, Suc : adCel>
Somme : fonction (Z : adCel) → entier
T : adCel

••••• {construction de la liste de tête T}

Écrire(Somme(T))

(1) Somme([]) = 0 (2) Somme(e o S) = e + Somme(S)

Somme (Z) :
retour :
si Z = Nil alors 0
sinon Z↑.E + Somme(Z↑.Suc)

Trace pour [1,2,3,4] ?

1 + (2 + (3 + (4 + 0)))

7-18

P.-C. Scholl, C. Vigouroux – février 2026

Sélection des éléments positifs

Représentation chaînée, réalisation 1

AdCel : type pointeur de Cel ; Cel : type <E : entier ; Suc : AdCel>

CréerSélPosListe : action (donnée T1: AdCel ; résultat T2: AdCel)

{ e.i. : soit s0 la séquence représentée par la liste de tête T1

e.f. : la liste de tête T2 représente SélPos(s0)}

CréerSélPosListe(T1, T2) : {la liste de tête T1 représente s0}
si T1 = Nil alors T2 ← Nil {s0 est vide}
sinon si T1↑.E ≤ 0 alors
 CréerSélPosListe(T1↑.Suc, T2)
sinon {premier(s0) > 0}
 Allouer(T2) ; T2↑.E ← T1↑.E ; T2↑.Suc ← Nil {création du singleton}
 CréerSélPosListe(T1↑.Suc, T2↑.Suc)

Ordre d'ajout des éléments : "de gauche à droite" (ordre de la donnée)

7-20

P.-C. Scholl, C. Vigouroux – février 2026

Sélection des éléments positifs

Représentation chaînée : réalisation 2

```
CréerSélPosListe(T1, T2) :      {la liste de tête T1 représente s0}
  Z : AdCel
  si T1 = Nil alors T2 ← Nil      {s0 est vide}
  sinon CréerSélPosListe(T1↑.Suc, T2)
    {la liste de tête T2 représente SélPos(fin(s0))}
    si T1↑.E > 0 alors      {premier(s0) > 0}
      Allouer(Z)      {ajout en tête}
      Z↑.E ← T1↑.E; Z↑.Suc ← T2; T2 ← Z
```

Ordre d'ajout des éléments : "de droite à gauche" (inverse de la donnée)

7-21

P.-C. Scholl, C. Vigouroux – février 2026

Suppression des éléments négatifs

Représentation chaînée : réalisation 2

```
SupprimerNégListe(T) :      {la liste de tête T représente s0}
  Z : adCel
  si T ≠ Nil alors
    si T↑.E ≤ 0 alors      {premier(s0) ≤ 0}
      Z ← T ; T ← T↑.Suc ; libérer(Z)      {suppression en tête}
      SupprimerNégListe(T)
  sinon SupprimerNégListe(T↑.Suc)
```

Erreur ?

Ordre de suppression des éléments : "de gauche à droite"

7-23

P.-C. Scholl, C. Vigouroux – février 2026

Suppression des éléments négatifs

Représentation chaînée : réalisation 1

Polycopié E7.6

AdCel : type pointeur de Cel
 Cel : type <E : entier ; Suc : AdCel>
 SupprimerNégListe : action (donnée-résultat T : AdCel)
 { e.i. : soit s0 la séquence représentée par la liste de tête T
 e.f. : la liste de tête T représente SélPos(s0) }

```
SupprimerNégListe(T) :      {la liste de tête T représente s0}
  Z : AdCel
  si T ≠ Nil alors
    SupprimerNégListe(T↑.Suc)
    {la liste de tête T↑.Suc représente SélPos(fin(s0))}
    si T↑.E ≤ 0 alors      {premier(s0) ≤ 0}
      Z ← T ; T ← T↑.Suc ; libérer(Z)      {suppression en tête}
```

Ordre de suppression des éléments : "de droite à gauche"

7-22

P.-C. Scholl, C. Vigouroux – février 2026

Suppression des éléments négatifs

Représentation chaînée : réalisation itérative

```
SupprimerNégListe(T) :
  F : adCel      {élément fictif temporaire}
  AC, AP : adCel      {adresse courante et prédécesseur}
  Allouer(F) ; F↑.Suc ← T      {connexion du fictif en tête}
  AP ← F ; AC ← T
  tant que AC ≠ Nil
    {Les éléments négatifs précédant AC ont été supprimés ; AC = AP↑.Suc}
    si AC↑.E ≤ 0 alors
      AP↑.Suc ← AC↑.Suc ; Libérer(AC)      {suppression de AC}
      AC ← AP↑.Suc
  sinon
    AP ← AC ; AC ← AP↑.Suc
  T ← F↑.Suc ; Libérer(F)      {libération du fictif de tête}
```

7-24

P.-C. Scholl, C. Vigouroux – février 2026

b) Représentation contiguë

Réalisation récursive

Le traitement d'une séquence est décrit en termes du même traitement portant sur une sous-séquence connexe, par exemple, la sous-séquence de début

Séquence en paramètre

- Le tableau contenant la séquence
 - variante : le tableau est déclaré comme une variable globale
- Une borne et la longueur de la sous-séquence
 - variante : la longueur seule si toutes les sous-séquences ont une borne commune

7-25

P.-C. Scholl, C. Vigouroux – février 2026

Sélection des éléments positifs

Construction d'une séquence

Polycopié E7.8

Construire une séquence formée des entiers strictement positifs d'une séquence d'entiers donnée. L'ordre initial doit être maintenu.

Analyse récurrente

SélPos : fonction (SD : séquence d'entier) → séquence d'entier > 0

(1) SélPos([]) = []

(3) SélPos(S•e) = si e > 0 alors SélPos(S)•e sinon SélPos(S)

7-27

P.-C. Scholl, C. Vigouroux – février 2026

Somme des éléments

Parcours : réalisation récursive

Polycopié E7.7

Lmax : constante de type entier > 0

TabE : type tableau sur [1 ... Lmax] d'entier

Somme : fonction (L : entier sur [0 ... Lmax], T : TabE) → entier

LD : entier sur [0 ... Lmax], TD : TabE

••••• {construction de la séquence <LD, TD>}

Écrire(Somme(LD, TD))

Analyse récurrente

(1) Somme([]) = 0

(2) Somme(S•e) = Somme(S) + e

Réalisation récursive

Somme (L, T) :

retour:

si L = 0 alors 0

sinon Somme(L-1, T) + T_L

Trace pour [1,2,3,4] ?

((0 + 1) + 2) + 3 + 4

7-26

P.-C. Scholl, C. Vigouroux – février 2026

Sélection des éléments positifs

Représentation contiguë

Lmax : constante de type entier > 0 ; TabE : type tableau sur [1... Lmax] d'entier

CréerSélPosTab : action (donnée T1 : TabE, L1 : entier sur [0 ... Lmax])

résultat T2 : TabE, L2 : entier sur [0 ... Lmax])

{ e.i. : soit s0 la séquence représentée par <T1, L1>

e.f. : <T2, L2> représente SélPos(s0) }

CréerSélPosTab(T1, L1, T2, L2) :

{<T1, L1> représente s0}

si L1 = 0 alors L2 ← 0

{s0 est vide}

sinon CréerSélPosTab(T1, L1-1, T2, L2)

{<T2, L2> représente SélPos(début(s0))}

si T1_{L1} > 0 alors

{dernier(s0) > 0}

L2 ← L2 + 1 ; T2_{L2} ← T1_{L1}

{ajout en queue}

Ordre d'ajout des éléments : "de gauche à droite"

7-28

P.-C. Scholl, C. Vigouroux – février 2026

Suppression des éléments négatifs

Modification d'une séquence, représentation contiguë

Polycopié E7.9

Supprimer les entiers négatifs ou nuls d'une séquence (ordre initial maintenu).

Lmax : constante de type entier > 0 ; TabE : type tableau sur [1...Lmax] d'entier

SupprimerNégTab : action

(donnée-résultat T : TabE, L : entier sur [0 ... Lmax])

{ e.i. : soit s0 la séquence représentée par <T, L>

e.f. : <T, L> représente SélPos(s0) }

```
SupprimerNégTab(T, L) :           {<T, L> représente s0}
  L1 : entier sur [0...Lmax]
  si L ≠ 0 alors
    si TL ≤ 0 alors                {dernier(s0) ≤ 0}
      L ← L - 1                    {suppression en queue}
      SupprimerNégTab(T, L)
    sinon
      L1 ← L-1 ; SupprimerNégTab(T, L1)
      TL1+1 ← TL ; L ← L1 + 1      {ajout en queue}
```

7-29

P.-C. Scholl, C. Vigouroux – février 2026

a) Schéma général

Schéma récursif considéré

A (P) :

si EstBase(P) alors

Alpha(P)

sinon Beta(P)

A(T(P))

Gamma(P)

Carrés emboîtés

Tcaremb (n,t) :

si n = 1 alors

Tdébutcar(t) ; Tfincar(t)

sinon Tdébutcar(t) ; TrG(45)

Tcaremb(n-1, t/√2)

TrD(45) ; Tfincar(t)

Param : type

{type des paramètres}

EstBase : fonction (P : Param) → booléen

{ vrai ⇔ la valeur de P correspond au cas de base }

Alpha, Beta, Gamma : action (donnée : Param)

T : fonction (P : Param) → Param

7-31

P.-C. Scholl, C. Vigouroux – février 2026

7.3. Du récursif à l'itératif

7. Composition récursive

7.1. Réalisation récursive d'une action

7.2. Traitement récursif des séquences

7.3. Du récursif à l'itératif

7-30

P.-C. Scholl, C. Vigouroux – février 2026

b) Schéma itératif associé

Hypothèse : T⁻¹ est définie

T⁻¹ : inverse de T {T⁻¹(T(P)) = P }

Schéma récursif

A (P) :

si EstBase(P) alors

Alpha(P)

sinon

Beta(P)

A(T(P))

Gamma(P)

Schéma itératif

A (P) :

Pcour : Param

Pcour ← P

tant que non EstBase(Pcour)

Beta(Pcour)

Pcour ← T(Pcour)

Alpha(Pcour)

tant que non (Pcour=P)

Pcour ← T⁻¹(Pcour)

Gamma(Pcour)

7-32

P.-C. Scholl, C. Vigouroux – février 2026

Tcaremb(n, t) :

{plume basse, au sommet O et au cap c de la figure}

nc : entier > 0 ; tc : réel > 0

$nc \leftarrow n$; $tc \leftarrow t$

tant que non ($nc = 1$)

Tdébutcar(tc) ; TrG(45)

$nc \leftarrow nc - 1$; $tc \leftarrow tc/\sqrt{2}$

Tdébutcar(tc) ; Tfincar(tc)

tant que non ($nc = n$)

$nc \leftarrow nc + 1$; $tc \leftarrow tc*\sqrt{2}$

TrD(45) ; Tfincar(tc)

{figure tracée, plume basse, position O , cap c }

$$T(n, t) = n-1, t/\sqrt{2}$$

$$T^{-1}(n, t) = n+1, t\sqrt{2}$$

7-33

P.-C. Scholl, C. Vigouroux – février 2026

c) Action récursive terminale

- Cas particuliers : Beta ou Gamma vides
- Lorsque Gamma est vide : action récursive terminale

Schéma récursif

A (P) :

si EstBase(P) **alors**

Alpha(P)

sinon

Beta(P)

A(T(P))

Schéma itératif

A (P) :

Pcour : Param

Pcour $\leftarrow$ P

tant que non EstBase(Pcour)

Beta(Pcour) ; Pcour $\leftarrow$ T(Pcour)

Alpha(Pcour)

7-35

P.-C. Scholl, C. Vigouroux – février 2026

b) Schéma itératif associé

Utilisation d'une pile

T^{-1} non définie. On utilise une *séquence* S pour mémoriser les valeurs de P .

Schéma récursif

A (P) :

si EstBase(P) **alors**

Alpha(P)

sinon

Beta(P)

A(T(P))

Gamma(P)

Schéma itératif

A (P) :

S : séquence de Param

Pcour : Param

$S \leftarrow []$; Pcour $\leftarrow$ P

tant que non EstBase(Pcour)

Beta(Pcour)

$S \leftarrow S \bullet Pcour$; Pcour $\leftarrow$ T(Pcour)

Alpha(Pcour)

tant que non EstVide?(S)

Pcour $\leftarrow$ dernier(S) ; $S \leftarrow$ début(S)

Gamma(Pcour)

Ajouts et suppressions dans S se font toujours à la même extrémité : S est une *pile*.

7-34

P.-C. Scholl, C. Vigouroux – février 2026

Conclusion du chapitre 7

Éléments pour guider le travail hors séance

- Comprendre le processus d'exécution d'un algorithme récursif
 - Structure des appels récursifs
 - Trace par réécriture
 - Comparaison "récursif – itératif"
- Savoir
 - Vérifier une solution à partir d'une spécification
 - induction sur le texte de la solution
 - Analyser le nombre d'appels récursifs
 - équations de récurrence d'une fonction "nombre d'appels engendrés"
 - Construire une solution à partir d'une analyse récurrente en particulier dans le cas de traitement de séquences
 - étude fonctionnelle : modèles de découpage
 - spécification de l'action de traitement (représentation contiguë ou chaînée)
 - réalisation récursive à partir de l'étude fonctionnelle

7-36

P.-C. Scholl, C. Vigouroux – février 2026

8. Représentations des arbres

8.1. Arbres binaires : représentation chaînée

Techniques liées aux arbres
Construction raisonnée des programmes
Analyse par abstractions successives
Structures de données

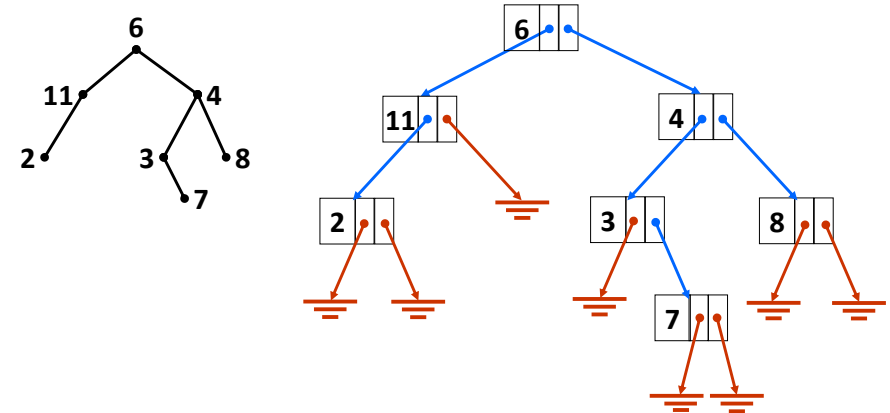
8.1. Arbres binaires : représentation chaînée

8.2. Arbres n-aires et forêts

8.3. Arbres binaires complets

a) Principe de la représentation

Deux liens sont associés à chaque élément :
l'un vers le fils gauche, l'autre vers le fils droit



a) Principe de la représentation

Lexique général

Élément : **type** {étiquettes portées par les nœuds}
adNœud : **type pointeur de** Nœud
{l'adresse d'un nœud est aussi l'adresse de l'arbre dont il est racine}
Nœud : **type** <R : Élément ; G, D : adNœud >
{étiquette de la racine et adresses (éventuellement Nil) des sous-arbres gauche et droit}
{Nil représente l'arbre vide.}

Pour un arbre de N nœuds, il y a 2N liens, dont N+1 valant Nil

a) Principe de la représentation

De l'abstrait au concret

Abstrait	Concret
A : arbre binaire	A : adNœud
EstVide?(A)	A = Nil
Racine(A)	A↑.R
Gauche(A)	A↑.G
Droit(A)	A↑.D
A ← / \	A ← Nil
A ← /G, R, D\	{Allouer(A) a été fait} A↑ ← <R, G, D>

Égalité de deux arbres

Eg_AB : **fonction** (A1, A2 : adNœud) → **booléen**

{ vrai ⇔ les deux arbres donnés sont égaux (même structure, mêmes étiquettes). }

Eg_AB(A1, A2) :

retour :

selon A1, A2

A1 = Nil et A2 = Nil : **vrai**

A1 ≠ Nil et A2 = Nil : **faux**

A1 = Nil et A2 ≠ Nil : **faux**

A1 ≠ Nil et A2 ≠ Nil : A1↑.R = A2↑.R

et Eg_AB(A1↑.G, A2↑.G)

et Eg_AB(A1↑.D, A2↑.D)

8-5

P.-C. Scholl, C. Vigouroux – février 2026

Libérer un arbre binaire

Libérer_AB : **action** (donnée-résultat A : adNœud)

{ Libère toutes les cellules. A l'état final, A=Nil }

Libérer_AB(A) :

si A ≠ Nil **alors**

Libérer_AB(A↑.G)

Libérer_AB(A↑.D)

Libérer(A)

A ← Nil

*Peut-on modifier
l'ordre des
instructions ?*

8-7

P.-C. Scholl, C. Vigouroux – février 2026

Recopier un arbre binaire

Copier_AB : **action** (donnée A1: adNœud ; **résultat** A2: adNœud)

{ Construit une copie de l'arbre représenté par A1 et donne son adresse dans A2. }

Analyse fonctionnelle

Copie : **fonction** (A : **arbre binaire** d'Élément) → **arbre binaire** d'Élément

(1) Copie(/ \) = / \

(2) Copie(/G, r, D\) = /Copie(G), r, Copie(D)\

Copier_AB(A1, A2) :

si A1 = Nil **alors** A2 ← Nil

sinon

Allouer(A2)

A2↑.R ← A1↑.R

Copier_AB(A1↑.G, A2↑.G)

Copier_AB(A1↑.D, A2↑.D)

*Peut-on modifier
l'ordre des
instructions ?*

8-6

P.-C. Scholl, C. Vigouroux – février 2026

b) Traitement séquentiel

Schémas récurrents de parcours et de recherche

Parcours en préordre

ParcoursPréordre (A) :

si A ≠ Nil **alors**

Traiter(A↑.R)

ParcoursPréordre(A↑.G)

ParcoursPréordre(A↑.D)

Parcours en préordre, arbre non vide

ParcoursPréordre (A) : {A ≠ Nil}

Traiter(A↑.R)

si A↑.G ≠ Nil **alors** ParcoursPréordre(A↑.G)

si A↑.D ≠ Nil **alors** ParcoursPréordre(A↑.D)

Pour l'ordre symétrique et l'ordre terminal

Changer la place de Traiter(A↑.R)

Schémas de recherche

Existence d'un élément vérifiant une propriété

Adresse du premier vérifiant une propriété

Voir le polycopié

8-8

P.-C. Scholl, C. Vigouroux – février 2026

b) Traitement séquentiel

Application directe des schémas récurifs de parcours

Contexte d'appel

AD : ... {adresse de l'arbre à traiter, supposé initialisé}

V : ... {variables décrivant le résultat}

ParcoursArbre : **action** (donnée A : adNœud; donnée-résultat X : ...)

{ Traite, une fois et une seule, tous les éléments de l'arbre de racine A en utilisant et modifiant X. Trois versions selon l'ordre choisi. }

{Algorithme}

Init(V) ; ParcoursArbre (AD, V) ; Terminer(V)

8-9

P.-C. Scholl, C. Vigouroux – février 2026

Somme des éléments d'un arbre

Application d'un schéma de parcours

adNœud : **type** **pointeur de** Nœud

Nœud : **type** <R : entier ; G, D : adNœud>

AD : adNœud

{arbre donné}

S : entier

{résultat}

AjouterSommeArbre : **action** (donnée A : adNœud; donnée-résultat E : entier)

{ Ajoute à E la somme des éléments de l'arbre de racine A (parcours en préordre de l'arbre de racine A) }

{Algorithme de sommation}

S ← 0 ; AjouterSommeArbre(AD, S) ; Écrire(S)

Init

AjouterSommeArbre(A, E) :

si A ≠ Nil alors

E ← E + A↑.R

AjouterSommeArbre(A↑.G, E)

AjouterSommeArbre(A↑.D, E)

Traiter

Term

Peut-on modifier
l'ordre des
instructions ?

8-10

P.-C. Scholl, C. Vigouroux – février 2026

Construire la liste en ordre symétrique

Version 1 : analyse récurrente

adCel : **type** **pointeur de** Cel ; Cel : **type** <E : Élément, Suc : adCel>

CréerLSym : **action** (donnée A : adNœud ; résultat T : adCel)

{ Construit la liste chaînée des éléments de l'arbre d'adresse A, en ordre symétrique, et donne son adresse dans T. }

Analyse fonctionnelle

(1) SéqSym(/ \) = []

(2) SéqSym(/G, r, D \) = SéqSym(G) & [r] & SéqSym(D)

8-11

P.-C. Scholl, C. Vigouroux – février 2026

Construire la liste en ordre symétrique

Réalisation

Der : **fonction** (L : adCel) → adCel

{ Adresse du dernier élément de la liste d'adresse L. L ≠ Nil. }

CréerLSym(A, T) :

Q : adCel

si A = Nil alors T ← Nil

sinon {créer, en T, la liste en ordre symétrique du sous-arbre gauche de A}

CréerLSym(A↑.G, T)

{ajouter la racine en queue de T}

Erreur ?

Allouer(Q) ; Q↑.E ← A↑.R ; ~~Der(T)↑.Suc ← Q~~

si T = Nil alors T ← Q sinon Der(T)↑.Suc ← Q

{ajouter la liste en ordre symétrique du sous-arbre droit de A en queue de T}

CréerLSym(A↑.D, Q↑.Suc)

8-12

P.-C. Scholl, C. Vigouroux – février 2026

Construire la liste en ordre symétrique

Version 2 : application d'un schéma de parcours d'arbre

ConcatSym : **action** (donnée A: adNœud, donnée-résultat Q: adCel)

{ Concatène à la liste de queue Q, la liste en ordre symétrique de l'arbre de racine A. A l'état final, Q est l'adresse de queue de la liste construite.

Pré-condition : $Q \neq \text{Nil}$ }

CréerLSym(A, T) : {crée en T, la liste en ordre symétrique de l'arbre A}

F, QR : adCel {élément fictif et adresse de queue de la liste résultat}

Allouer(F) ; QR ← F {création d'un fictif de tête}

ConcatSym(A, QR)

QR↑.Suc ← Nil

T ← F↑.Suc ; **Libérer**(F) {suppression du fictif}

ConcatSym(A, Q) : {parcours en ordre symétrique : traiter = ajout en queue}

N : adCel

si A ≠ Nil **alors**

ConcatSym(A↑.G, Q)

Allouer(N) ; N↑.E ← A↑.R ; Q↑.Suc ← N ; Q ← N {ajout en queue}

ConcatSym(A↑.D, Q)

8-13

P.-C. Scholl, C. Vigouroux – février 2026

8.2. Arbres n-aires et forêts

Représentations des arbres

8.1. Arbres binaires : représentation chaînée

8.2. Arbres n-aires et forêts

8.3. Arbres binaires complets

8-14

P.-C. Scholl, C. Vigouroux – février 2026

a) Définitions

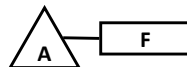
• Arbre n-aire = racine + forêt des sous-arbres

- $\langle r, F \rangle$ dénote l'arbre formé de
 - la racine r
 - la forêt de sous-arbres F (éventuellement vide)

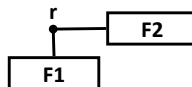


• Forêt = séquence d'arbres

- $[]$ dénote la forêt vide
- $A \circ F$ dénote une forêt non vide formée
 - d'un premier arbre A
 - et d'une forêt F



- $\langle r, F1 \circ F2 \rangle$ dénote une forêt non vide formée
 - d'un premier arbre $\langle r, F1 \rangle$
 - et d'une forêt F2



8-15

P.-C. Scholl, C. Vigouroux – février 2026

b) Principes d'analyse

- Traiter un arbre A
 - Traiter la racine et traiter la forêt des sous-arbres.
 - Ou Traiter la forêt ne comportant que l'arbre A
- Traiter une forêt
 - Appliquer les modèles d'analyse du traitement de séquences.

On décrit deux fonctions

- L'une portant sur un arbre.
- L'autre portant sur une forêt.

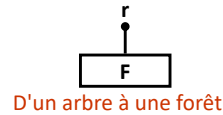
8-16

P.-C. Scholl, C. Vigouroux – février 2026

Nombre de nœuds d'un arbre

NbN_A : fonction (AD : arbre d'Élément) → entier ≥ 0 {nombre de nœuds}

NbN_F : fonction (FD : Forêt d'Élément) → entier ≥ 0 {nombre de nœuds}



$$(1) \text{NbN_A}(\langle r, F \rangle) = 1 + \text{NbN_F}(F)$$

ou

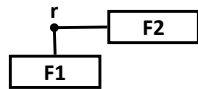
$$(1') \text{NbN_A}(A) = \text{NbN_F}([A])$$



NbN_F : version 1

$$(2) \text{NbN_F}([]) = 0$$

$$(3) \text{NbN_F}(A \circ F) = \text{NbN_A}(A) + \text{NbN_F}(F)$$



NbN_F : version 2

$$(2) \text{NbN_F}([]) = 0$$

$$(4) \text{NbN_F}(\langle r, F1 \rangle \circ F2) = 1 + \text{NbN_F}(F1) + \text{NbN_F}(F2)$$

8-17

P.-C. Scholl, C. Vigouroux – février 2026

Degré maximum d'un arbre n-aire

Version 1

Le **degré** d'un nœud est le nombre de ses fils

dm_A : fonction (AD : arbre d'Élément) → entier ≥ 0

{valeur maximum de degré des nœuds de l'arbre AD}

dm_F : fonction (FD : Forêt non vide d'Élément) → entier ≥ 0

{valeur maximum de degré des nœuds de la forêt FD}

nba_F : fonction (FD : Forêt d'Élément) → entier ≥ 0

{nombre d'arbres de la forêt FD (sa longueur)}

$$(1) \text{dm_A}(\langle r, [] \rangle) = 0$$

$$(2) \text{dm_A}(\langle r, F \rangle) = \max(\text{nba_F}(F), \text{dm_F}(F)) \quad \{F \text{ non vide}\}$$

$$(3) \text{dm_F}([A]) = \text{dm_A}(A)$$

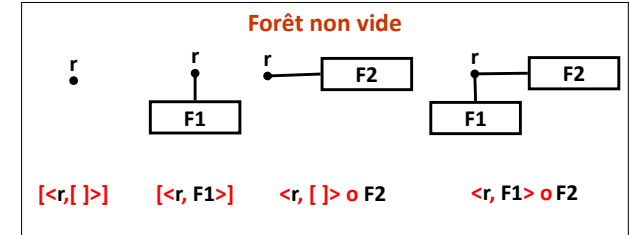
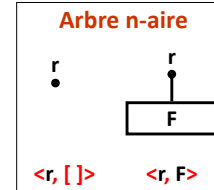
$$(4) \text{dm_F}(A \circ F) = \max(\text{dm_A}(A), \text{dm_F}(F)) \quad \{F \text{ non vide}\}$$

8-19

P.-C. Scholl, C. Vigouroux – février 2026

b) Principes d'analyse

Analyse en termes de forêts



Forêts : Modèle 1 en incluant la forêt vide	Forêts : Modèle 2 en excluant la forêt vide
$F([]) = \dots$ $F(\langle r, F1 \rangle \circ F2) = \dots$	$F(\langle r, [] \rangle) = \dots$ $F(\langle r, F1 \rangle) = \dots$ {F1 non vide} $F(\langle r, [] \rangle \circ F2) = \dots$ {F2 non vide} $F(\langle r, F1 \rangle \circ F2) = \dots$ {F1 et F2 non vides}

8-18

P.-C. Scholl, C. Vigouroux – février 2026

Degré maximum d'un arbre n-aire

Autres versions

$$\text{dm_A}(A) = \text{dm_F}([A])$$

$$(1) \text{dm_F}(\langle r, [] \rangle) = 0$$

$$(2) \text{dm_F}(\langle r, F \rangle) = \max(\text{nba_F}(F), \text{dm_F}(F)) \quad \{F \text{ non vide}\}$$

$$(3) \text{dm_F}(\langle r, [] \rangle \circ F) = \text{dm_F}(F) \quad \{F \text{ non vide}\}$$

$$(4) \text{dm_F}(\langle r, F1 \rangle \circ F2) = \max(\text{nba_F}(F1), \text{dm_F}(F1), \text{dm_F}(F2)) \quad \{F1 \text{ et } F2 \text{ non vides}\}$$

On étend la définition de **dm_F**

dm_F : fonction (F : Forêt d'Élément) → entier ≥ 0 ∪ {-1}

{valeur maximum de degré des nœuds de la forêt F ou -1 si F est vide}

$$\text{dm_A}(A) = \text{dm_F}([A])$$

$$(1) \text{dm_F}([]) = -1$$

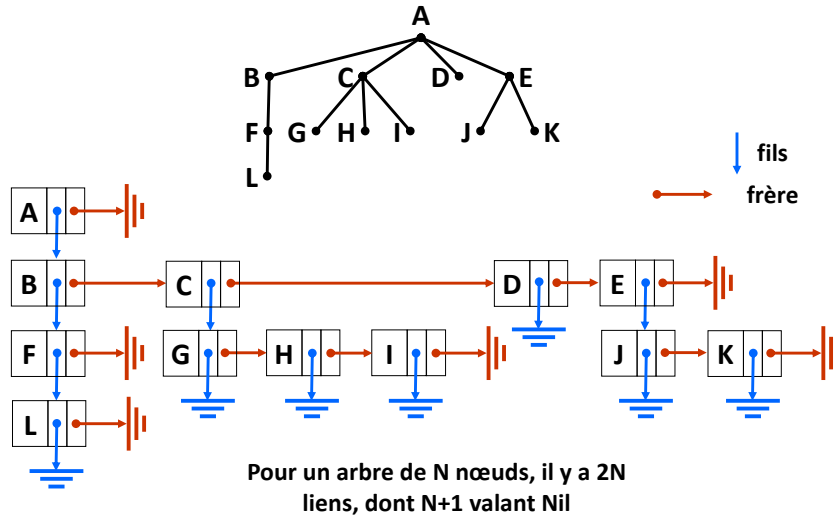
$$(2) \text{dm_F}(\langle r, F1 \rangle \circ F2) = \max(\text{nba_F}(F1), \text{dm_F}(F1), \text{dm_F}(F2))$$

8-20

P.-C. Scholl, C. Vigouroux – février 2026

c) Principe de la représentation chaînée

Liste de frères, listes de fils aînés



8-21

P.-C. Scholl, C. Vigouroux – février 2026

c) Principe de la représentation chaînée

Lexique

Élément : **type** {étiquettes portées par les nœuds}

adNœud : **type pointeur de Nœud**

{une adresse permet d'accéder à un nœud, à un arbre ou à une forêt.}

Nœud : **type** <R : Élément ; Fils, Frère : adNœud >

{étiquette de la racine et adresses (éventuellement Nil) du fils aîné (forêt des fils) et du frère cadet (forêt des frères.)}

{Un arbre n-aire est accessible par l'adresse de sa racine. Une forêt est accessible par l'adresse de son premier arbre. Nil représente l'arbre vide ou la forêt vide.}

8-22

P.-C. Scholl, C. Vigouroux – février 2026

Nombre de nœuds d'un arbre n-aire

Version 1 : itération sur la liste des sous-arbres

NbN_A : **fonction** (A : adNœud) → entier ≥ 0

{ Nombre de nœuds de l'arbre d'adresse A. Précondition : A ≠ Nil. }

NbN_A(A) : {A ≠ Nil}

N : entier > {résultat à calculer}

AC : adNœud {adresse du fils courant}

{traiter la racine}

N ← 1

{traiter la forêt des sous-arbres : itération de parcours}

AC ← A↑.Fils

tant que AC ≠ Nil

N ← N + NbN_A(AC) ; AC ← AC↑.Frère

retour : N

8-23

P.-C. Scholl, C. Vigouroux – février 2026

Nombre de nœuds d'un arbre n-aire

Version 2 : en se ramenant au nombre de nœuds d'une forêt

NbN_F : **fonction** (F : adNœud) → entier ≥ 0

{ Nombre de nœuds de la forêt d'adresse F. }

NbN_A(A) :

retour : 1 + NbN_F(A↑.Fils)

NbN_F(F) :

retour :

si F = Nil **alors** 0

sinon 1 + NbN_F(F↑.Fils) {premier arbre de F}

+ NbN_F(F↑.Frère) {autres arbres de F}

8-24

P.-C. Scholl, C. Vigouroux – février 2026

Nombre de nœuds de niveau n

Version 1 : itération sur la liste des sous-arbres

NbNiv_A : fonction (A : adNœud, n : entier > 0) → entier ≥ 0

{ Nombre de nœuds de niveau n dans l'arbre d'adresse A. Pré-condition : A ≠ Nil. }

```
NbNiv_A(A, n) : {A ≠ Nil}
  Nb : entier ≥ 0 ; AC : adNœud
  si n = 1 alors Nb ← 1
  sinon {nombre de nœuds de niveau n - 1 dans les sous-arbres}
    Nb ← 0 ; AC ← A↑.Fils
    tant que AC ≠ Nil
      Nb ← Nb + NbNiv_A(AC, n - 1) ; AC ← AC↑.Frère
  retour : Nb
```

8-25

P.-C. Scholl, C. Vigouroux – février 2026

Nombre de nœuds de niveau n

Version 2 : en se ramenant au nombre de nœuds de niveau n dans une forêt

NbNiv_F : fonction (F : adNœud, n : entier > 0) → entier ≥ 0

{ Nombre de nœuds de niveau n de la forêt d'adresse F. }

```
NbNiv_A(A, n) :
  retour : si n = 1 alors 1 sinon NbNiv_F(A↑.Fils, n - 1)
NbNiv_F(F, n) :
  retour:
    si F = Nil alors 0
    sinon (si n = 1 alors 1 sinon NbNiv_F(F↑.Fils, n - 1) ) {premier arbre de F}
      + NbNiv_F(F↑.Frère, n - 1) {autres arbres de F}
        n
```

Erreur ?

8-26

P.-C. Scholl, C. Vigouroux – février 2026

d) Traitement séquentiel

Parcours préfixé

Parcours préfixé d'un arbre n-aire non vide

```
ParcoursPréfixéArbre(A) : {A ≠ Nil}
  AC : adNœud
  Traiter(A↑.R)
  AC ← A↑.Fils {parcours de la liste des sous-arbres}
  tant que AC ≠ Nil
    ParcoursPréfixéArbre(AC) ; AC ← AC↑.Frère
```

Parcours préfixé d'une forêt

```
ParcoursPréfixéForêt(F) :
  si F ≠ Nil alors
    Traiter(F↑.R) ; ParcoursPréfixéForêt(F↑.Fils) {premier arbre de F}
  ParcoursPréfixéForêt(F↑.Frère) {autres arbres de F}
```

Pour l'ordre postfixé
Changer la place de Traiter

8-27

P.-C. Scholl, C. Vigouroux – février 2026

d) Traitement séquentiel

Recherche dans un arbre n-aire

Existence d'un élément vérifiant P dans un arbre non vide

```
ExisteParbre(A) : {A ≠ Nil}
  AC : adNœud ; Ok : booléen
  si P(A↑.R) alors Ok ← vrai
  sinon AC ← A↑.Fils {recherche dans la liste des sous-arbres}
    tant que AC ≠ Nil et puis non ExisteParbre(AC)
      AC ← AC↑.Frère
  Ok ← AC ≠ Nil
  retour : Ok
```

Adresse du premier élément vérifiant P (ordre préfixé)

```
PremierPréfixéArbreSelonP(A) : {A ≠ Nil, si non trouvé le résultat est Nil}
  AC, Y : adNœud
  si P(A↑.R) alors Y ← A
  sinon Y ← Nil ; AC ← A↑.Fils {Y=Nil ⇔ on n'a pas trouvé}
    tant que AC ≠ Nil et Y = Nil {recherche dans la liste des sous-arbres}
      Y ← PremierPréfixéArbreSelonP(AC) ; AC ← AC↑.Frère
  retour : Y
```

8-28

P.-C. Scholl, C. Vigouroux – février 2026

d) Traitement séquentiel

Adresse du premier élément vérifiant P dans une forêt

Adresse du premier élément vérifiant P (ordre préfixé)

PremierPréfixéForêtSelonP(F) :

retour :

si F = Nil ou alors P(F↑.R) alors F

sinon

soit B = PremierPréfixéForêtSelonP(F↑.Fils)

dans si B ≠ Nil alors B sinon PremierPréfixéForêtSelonP(F↑.Frère)

Adresse du premier élément vérifiant P (ordre infixé)

PremierInfixéForêtSelonP(F) :

retour :

si F = Nil alors Nil

sinon soit B = PremierInfixéForêtSelonP(F↑.Fils)

dans si B ≠ Nil alors B

sinon si P(F↑.R) alors F sinon PremierInfixéForêtSelonP(F↑.Frère)

8-29

P.-C. Scholl, C. Vigouroux – février 2026

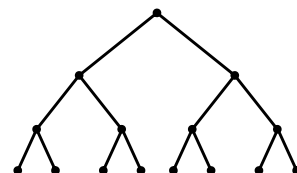
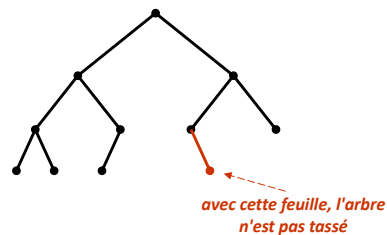
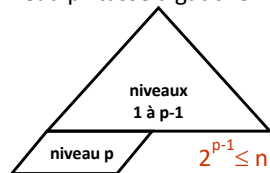
a) Arbres binaires complets

définitions

On considère des arbres de profondeur p

Arbre binaire "tassé"

- toutes les feuilles aux niveaux p ou p-1
- niveau p "tassé à gauche"



Arbre binaire "complet"

- tassé
- niveau p plein

8-31

P.-C. Scholl, C. Vigouroux – février 2026

8.3. Arbres binaires complets

Représentations des arbres

8.1. Arbres binaires : représentation chaînée

8.2. Arbres n-aires et forêts

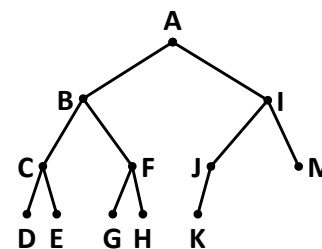
8.3. Arbres binaires complets

8-30

P.-C. Scholl, C. Vigouroux – février 2026

b) Arbres binaires complets

Représentation contiguë



1	2	3	4	5	6	7	8	9	10	11	12
A	B	I	C	F	J	M	D	E	G	H	K

- Séquence des nœuds : ordre par niveaux
- T : tableau sur [1...n] d'Élément
 - Racine en position 1
 - X en position i
 - Gauche(X) en position 2i
 - Droit(X) en position 2i+1

8-32

P.-C. Scholl, C. Vigouroux – février 2026

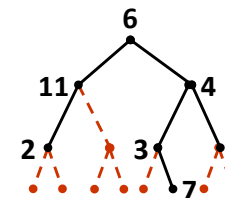
b) Arbres binaires complets

De l'abstrait au concret

Abstrait	Concret
A : arbre binaire	A : indice
EstVide?(A)	$A > n$
Racine(A)	T_A
Gauche(A)	$2 * A$
Droit(A)	$2 * A + 1$
ExisteGauche(A)	$2 * A \leq n$
ExisteDroit(A)	$2 * A + 1 \leq n$
EstFeuille(A)	$2 * A > n$

c) Arbres binaires : représentation contiguë

Principe



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	11	4	2		3	8						7		

• T : tableau sur $[1 \dots 2^p - 1]$ d'Élément

• Racine en position 1

• X en position i

- Gauche(X) en position $2i$

- Droit(X) en position $2i+1$

• Une marque pour caractériser l'absence de fils

• EstVide?(A) : $T_A = \text{marque}$

8-33

P.-C. Scholl, C. Vigouroux – février 2026

8-34

P.-C. Scholl, C. Vigouroux – février 2026

d) Tas, files d'attente à priorité

- Arbre binaire partiellement ordonné
 - l'étiquette d'un nœud est supérieure ou égale à celle de ses fils
 - la racine a une valeur maximale

Tas

Arbre binaire tassé partiellement ordonné

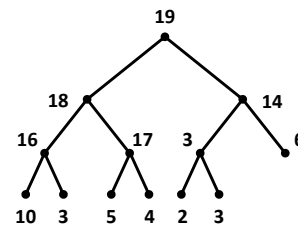
- Mises à jour "efficaces"
- L'arbre est représenté dans un tableau

File d'attente à priorité

- Ensemble de doublets $\langle \text{Élément}, \text{Priorité} \rangle$
- Primitives : Insérer(E,P), Extraire(E,P)

Extraire donne un élément de priorité maximum et sa priorité

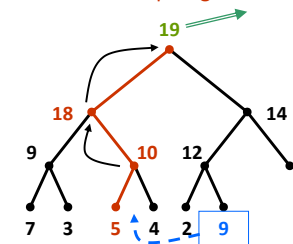
- Représentation par un tas



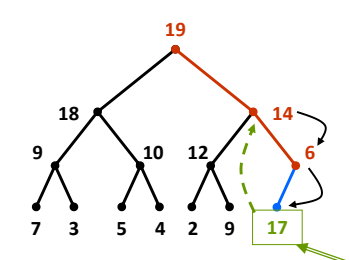
d) Tas, file d'attente à priorité

Opérations

Extraire : suppression de la racine
déplacement de la "dernière" feuille
"descente" à partir de la racine le long du
chemin des plus grands fils



Insérer
Placement en "dernière" feuille
"remontée" le long du chemin
conduisant à la racine



Chacune des deux opérations est une insertion dans une séquence triée
coût d'ordre $p = \lfloor \log_2 n \rfloor + 1$

8-35

P.-C. Scholl, C. Vigouroux – février 2026

8-36

P.-C. Scholl, C. Vigouroux – février 2026

C'est fini

Merci
et
Bonne fin d'année !