Chapitre 4

Structures arborescentes

Définition: un arbre est un ensemble de nœuds organisés à partir d'un nœud distingué appelé racine.

Applications : - organisation des fichiers : unix

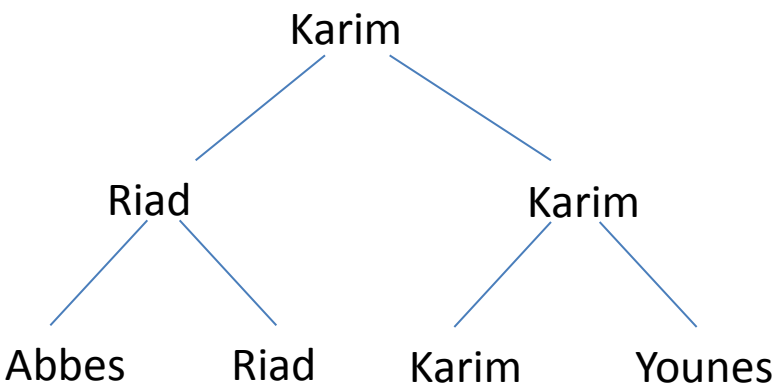
- Représentation des pgmes en compilation

Propriété intrinsèque : récursivité naturelle.

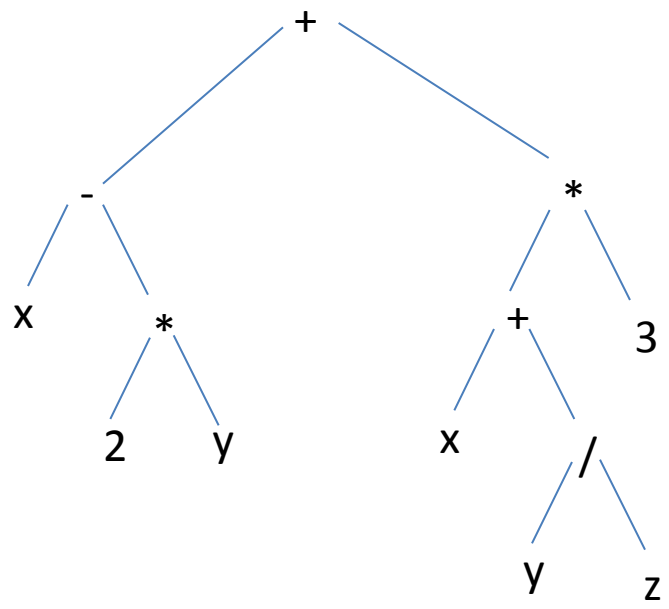
Arbres binaires

Exemples :

1) Tournoi de Tennis



2) Expressions arithmétiques

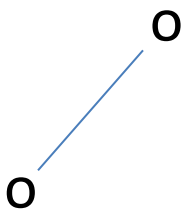


Définition : un arbre binaire est soit vide (ϕ) soit B=<o, B_1, B_2>; où B_1, B_2 sont des arbres binaires disjoints, 'o' est un nœud appelé racine.

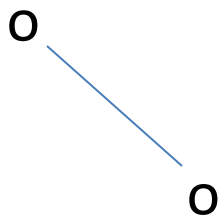
En d'autres termes : B=ϕ + <o, B, B>

Non symétrie gauche-droite :

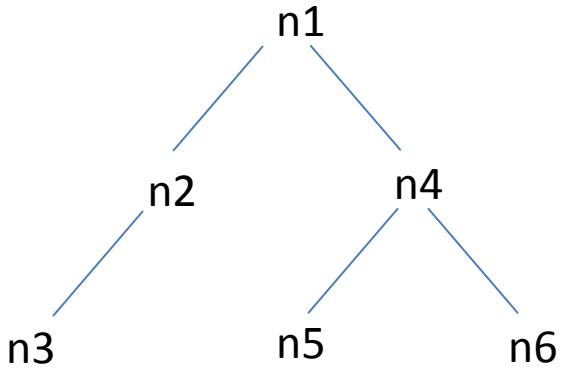
<o, <o, ϕ, ϕ>, ϕ> $\neq$ <o, ϕ, <o, ϕ,ϕ>>



$\neq$



Pour repérer les nœuds, il faut leur associer des noms différents comme dans :



Arbre étiqueté

TAD arbre binaire

Sorte Arbre

Utilise Nœud, Elément

Opérations

arbre-vide	:		$\rightarrow$ Arbre
< _ , _ , _ >	:	Nœud x Arbre x Arbre	$\rightarrow$ Arbre
racine	:	Arbre	$\rightarrow$ Nœud
g	:	Arbre	$\rightarrow$ Arbre
d	:	Arbre	$\rightarrow$ Arbre
contenu	:	Nœud	$\rightarrow$ Elément

Pré-conditions : où B_1, B_2 : Arbre; o: Nœud

racine (B_1) est-défini-ssi $B_1 \neq$ arbre-vide

g(B_1) est-défini-ssi $B_1 \neq$ arbre-vide

d(B_1) est-défini-ssi $B_1 \neq$ arbre-vide

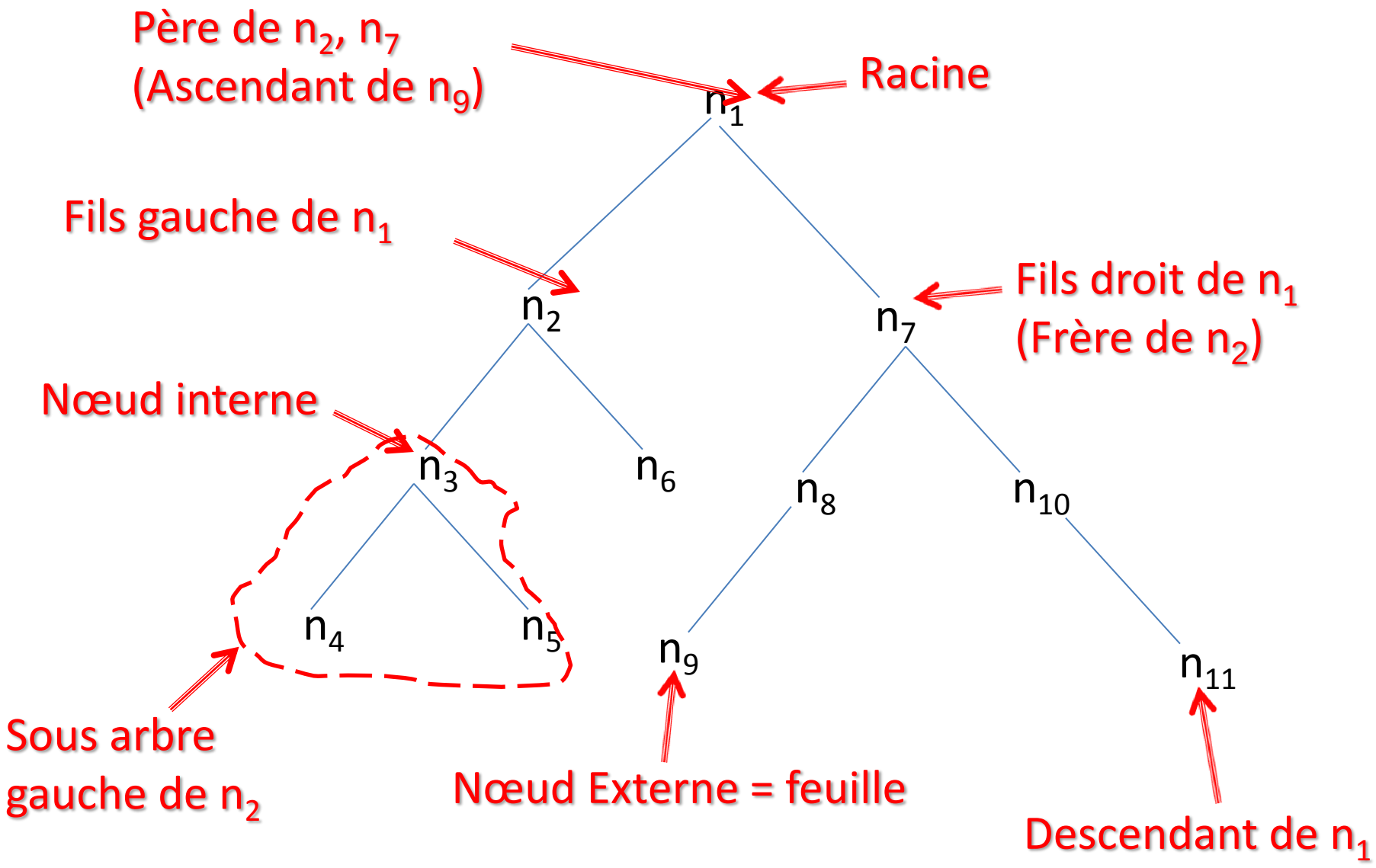
Axiomes :

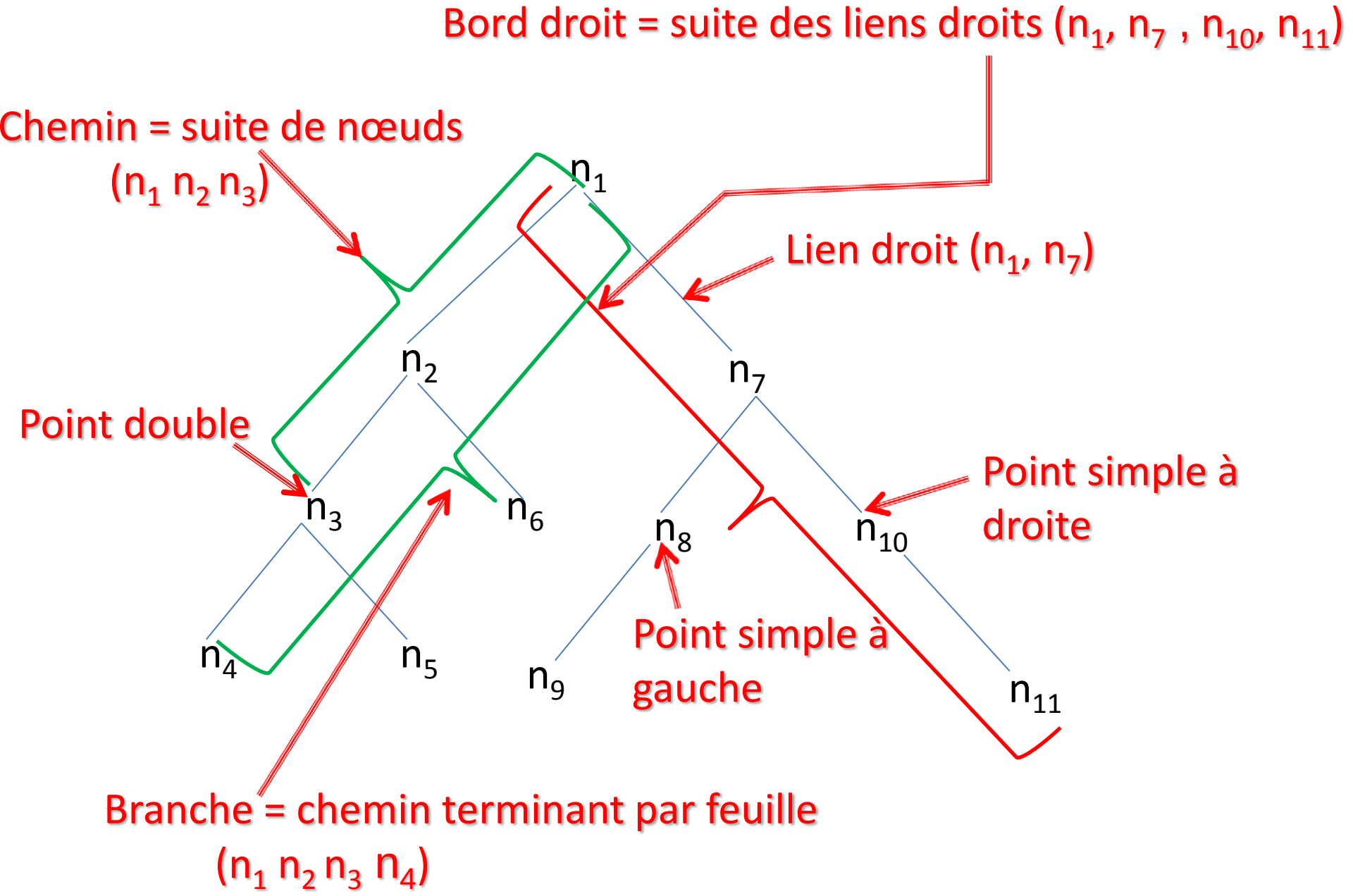
racine (<o, B_1, B_2>)=o

g(<o, B_1, B_2>)=B_1

d(<o, B_1, B_2>)=B_2

Vocabulaire




Bord droit = suite des liens droits (n_1, n_7 , n_{10}, n_{11})
Chemin = suite de nœuds
(n_1 n_2 n_3)
Lien droit (n_1, n_7)
Point double
Point simple à droite
Point simple à gauche
Branche = chemin terminant par feuille
(n_1 n_2 n_3 n_4)
n_1
n_2
n_3
n_4
n_5
n_6
n_7
n_8
n_9
n_{10}
n_{11}

Mesures sur les arbres

Taille=nombre des nœuds

Taille (arbre-vide)=0

Taille(<o, B_1, B_2 >)=1+taille(B_1)+taille(B_2)

Hauteur=Profondeur=niveau d'un nœud x d'un arbre B

h(x)=0 si x = racine (B)

h(x)=1+h(y) si y est le père de x

C'est le nombre de liens de la racine vers x.

Hauteur = Profondeur d'un arbre B

h(B)=max {h(x); x nœud de B}

Longueur de cheminement externe d'un arbre B :

LCE(B)= $\sum$ h(f) où f est une feuille de B.

Longueur de cheminement interne d'un arbre B :

LCI(B)= $\sum$ h(x) où x est un nœud interne de B.

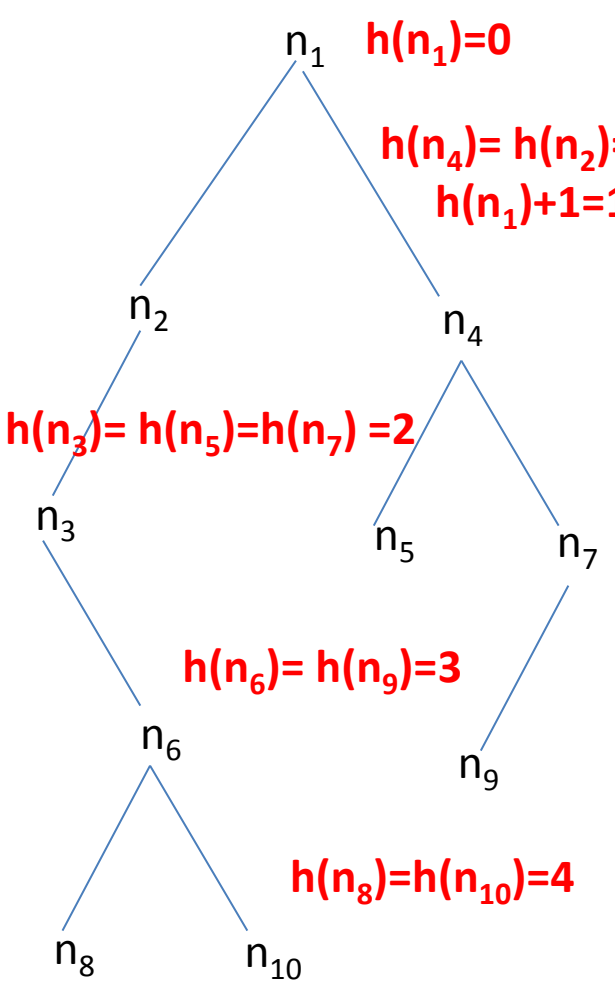
Longueur de cheminement d'un arbre B :

LC(B) = $\sum$ h(x) pour tous les nœuds x de B.

=LCE(B) + LCI(B)

Profondeur moyenne externe d'un arbre B ayant *f* feuilles :

PE(B)=LCE(B) / f



Donc : h(Arbre B)=4

LCE(B) = $h(n_5)+h(n_9)+h(n_{10})+ h(n_8)=13$

LCI(B) = $h(n_6)+h(n_3)+h(n_7)+ h(n_4)+h(n_2)=9$

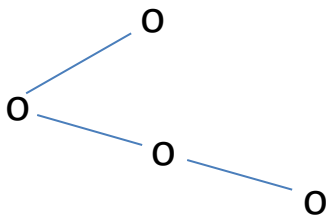
LC(B)=13+9=22

PE(B)=13/4 = 3.25

Arbres binaires particuliers

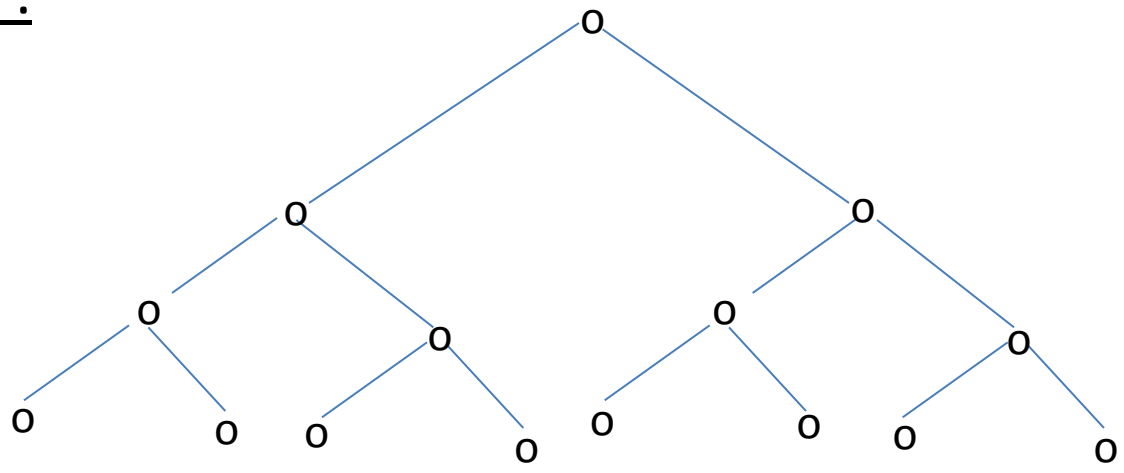
- Un arbre binaire **dégénéré** ou **filiforme** n'est formé que de points simples.

  Exemple:



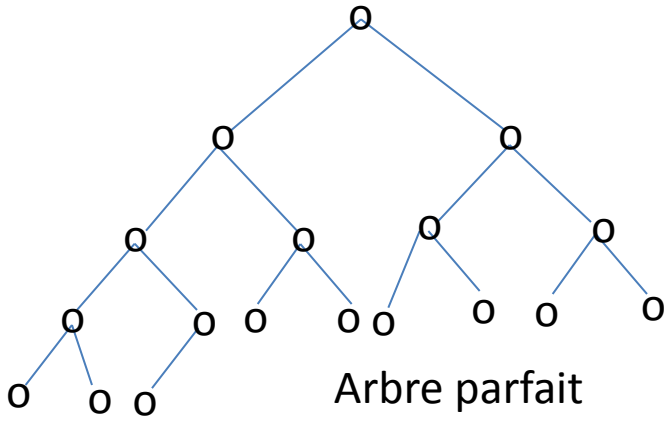
- Un arbre binaire est **complet** si chaque niveau i contient 2^i nœuds (càd complètement rempli) et la taille de l'arbre = $2^{h+1}-1$ où h=hauteur de l'arbre.

  Exemple :

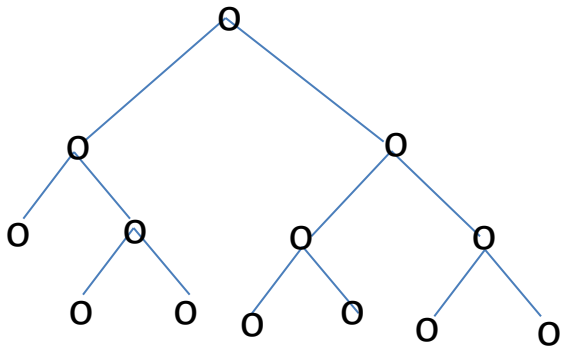


- Un arbre binaire est dit **parfait** si tous les niveaux sont complètements remplis sauf, éventuellement, le dernier niveau dont les nœuds sont groupés le plus à gauche possible.

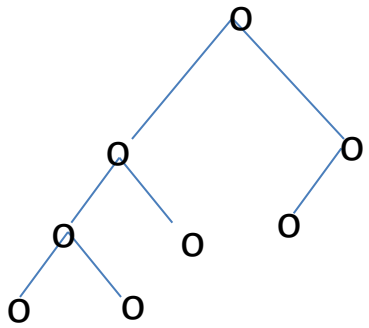
  Exemples:



Arbre parfait



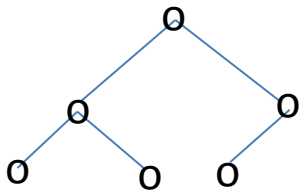
Arbre non parfait



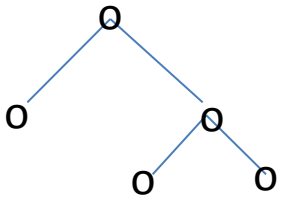
Arbre non parfait

- Un arbre binaire **localement complet** est un arbre non vide qui n'a pas de points simples.

  Exemples:



Non localement complet



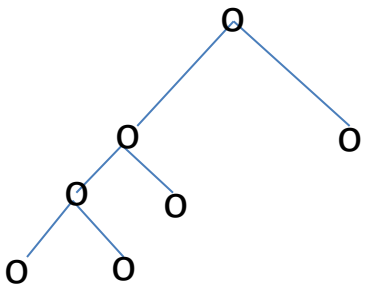
localement complet

Récursivement , on définit un arbre localement complet par :

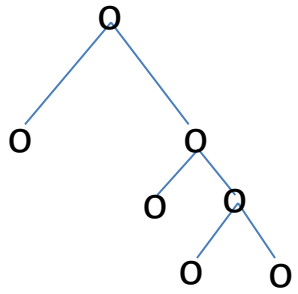
BC = <o, ϕ, ϕ> + <o, BC, BC>

- Un **peigne gauche (droit)** est un arbre binaire localement complet dans lequel tout fils droit (gauche) est une feuille.

  PG=<o, ϕ, ϕ> + <o, PG, <o, ϕ, ϕ>>



Peigne gauche



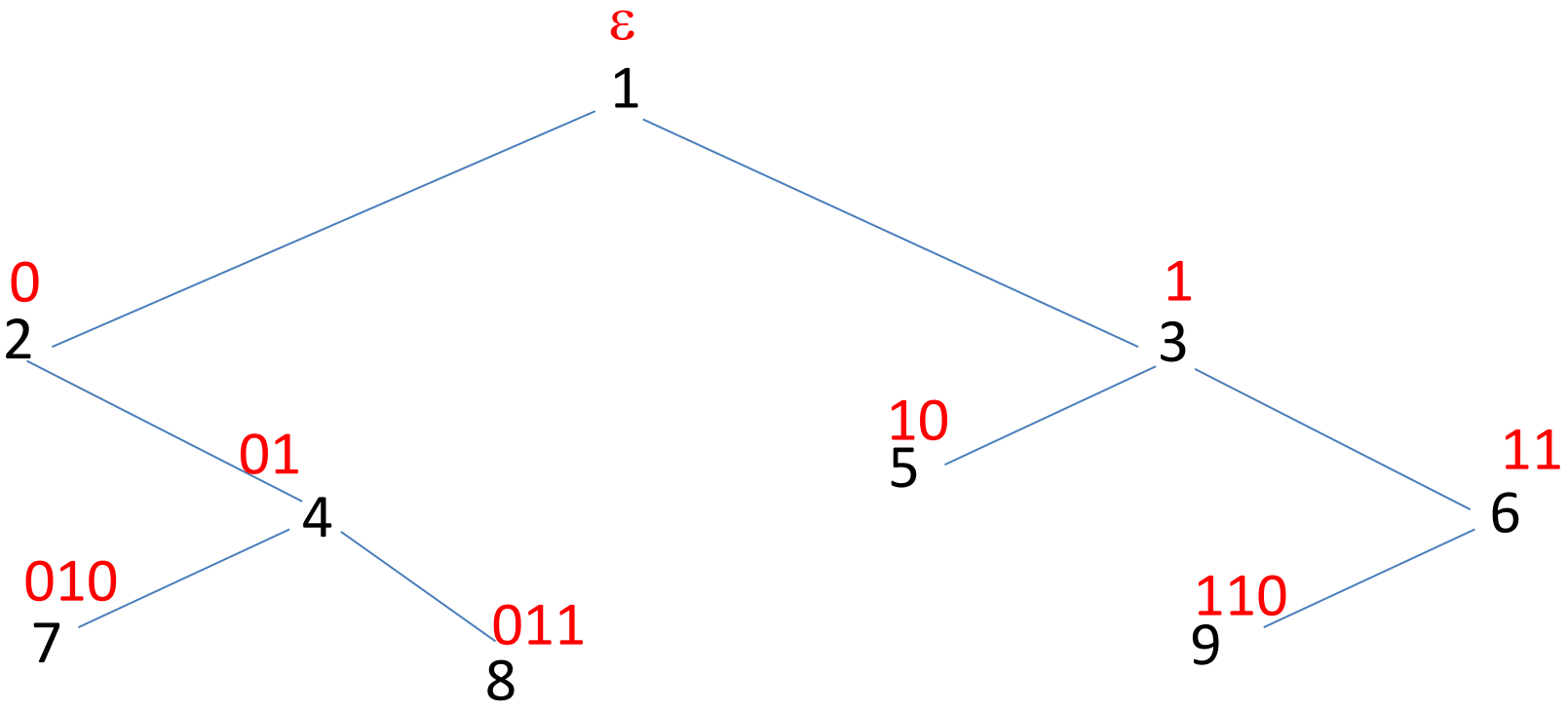
Peigne droit

Occurrence :

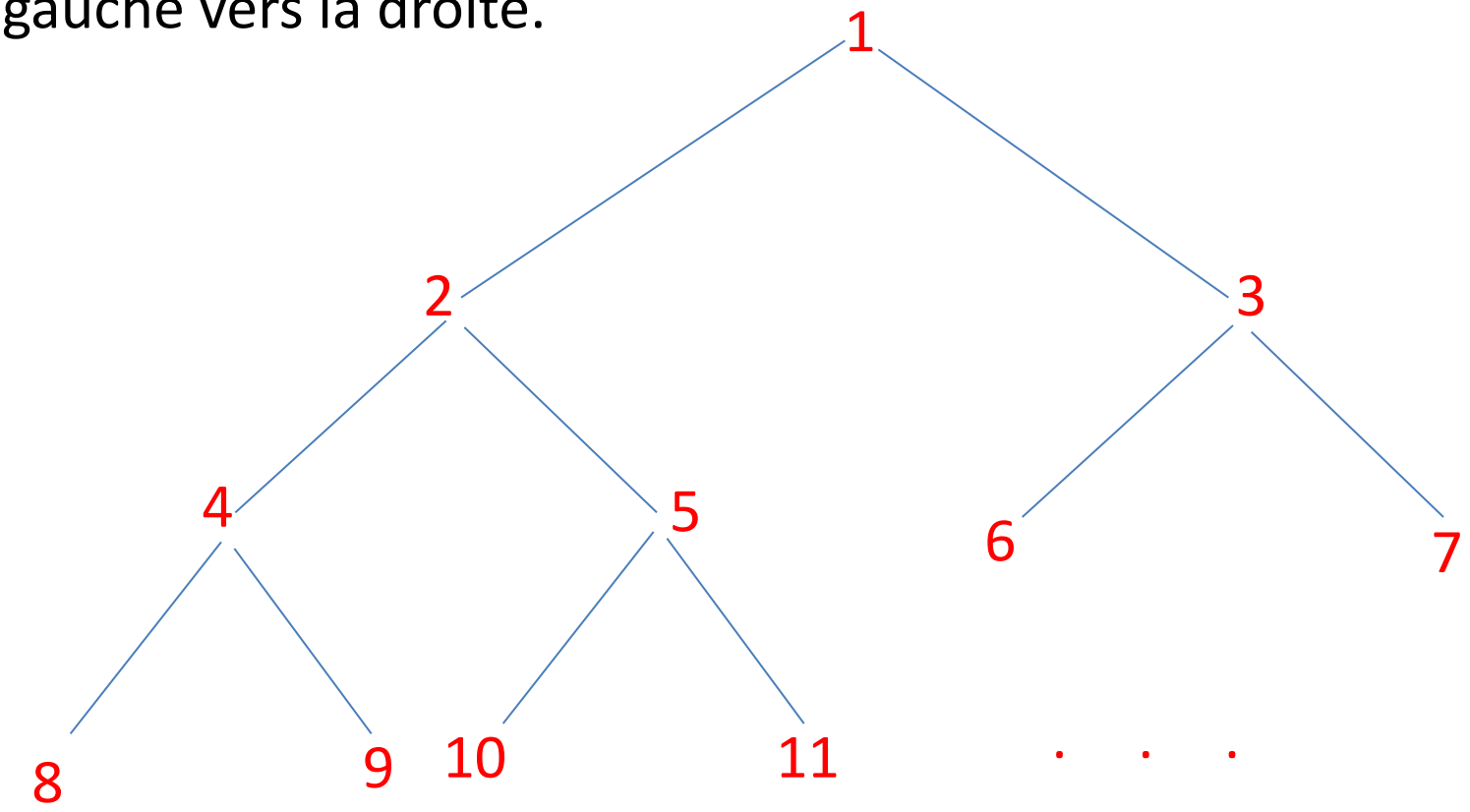
Le chemin de la racine vers un nœud est décrit par un mot sur $\{0,1\}^*$, appelé occurrence tel que :

- L'occurrence de la racine = mot vide = ε
- Si un nœud a pour occurrence μ, alors son fils gauche a pour occurrence $\mu\mathbf{0}$, et son fils droit a pour occurrence $\mu\mathbf{1}$.

Exemple : Soit l'arbre B suivant :



Numérotation en ordre hiérarchique : Niveau par niveau de la gauche vers la droite.



Pour un nœud d'ordre i et d'occurrence μ on a :

$i = 2^{[\log_2 i]} + m$ où m est l'entier représenté par μ ($m=(\mu)_{10}$) et [k] = Partie entière de k.

Exercices du TD N° 04

Exercice 01 : On a vu, en cours, qu'on peut représenter un nœud dans un arbre binaire par son occurrence qui est un mot sur $\{0, 1\}^*$.

1) Trouver l'arbre correspondant à l'ensemble d'occurrences suivant :

B= **{** ε, 0, 1, 00,01, 010, 011, 10, 11, 110, 1100, 1101, 111 **}**

2) Exprimer dans cette représentation :

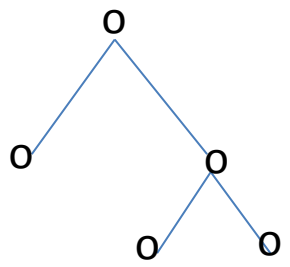
a) Le bord gauche et le bord droit

b) La hauteur d'un nœud, la hauteur de l'arbre

c) la longueur de cheminement.

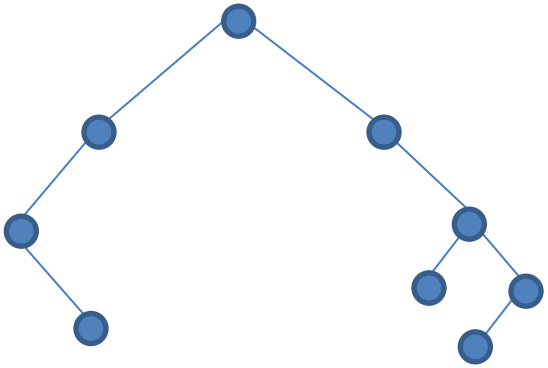
Propriétés fondamentales :

a) Un arbre binaire B localement complet avec n nœuds internes possède (n+1) feuilles et on a : LCE (B) =LCI(B) + 2n



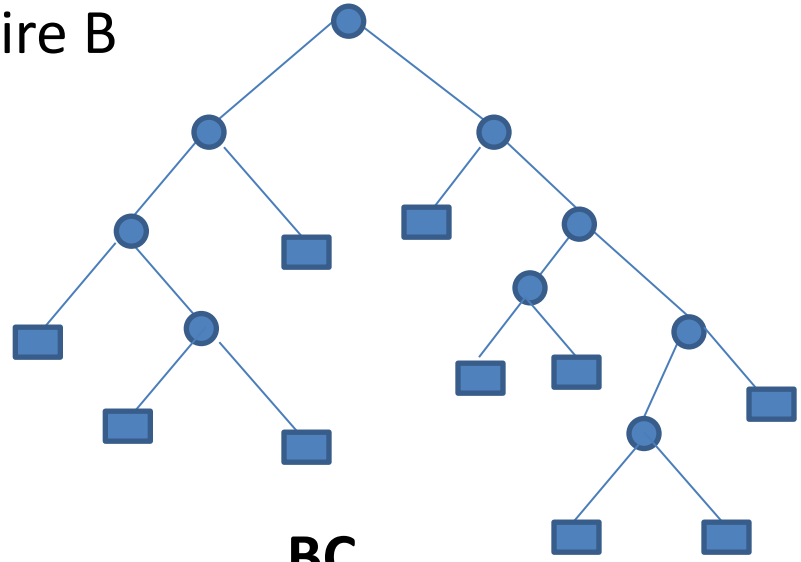
2 nœuds internes, 3 feuilles,
LCE=5, LCI=1

b) **Complétion locale** d'un arbre binaire B



B

⇒



BC

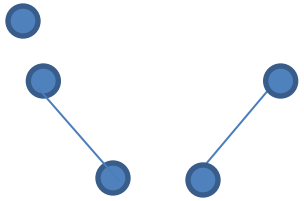
chaque nœud de B possède deux fils dans BC

c) Bijection entre B_n l'ensemble des arbres binaires ayant n nœuds, et l'ensemble BC_n des arbres binaires localement complets ayant 2n+1 nœuds.

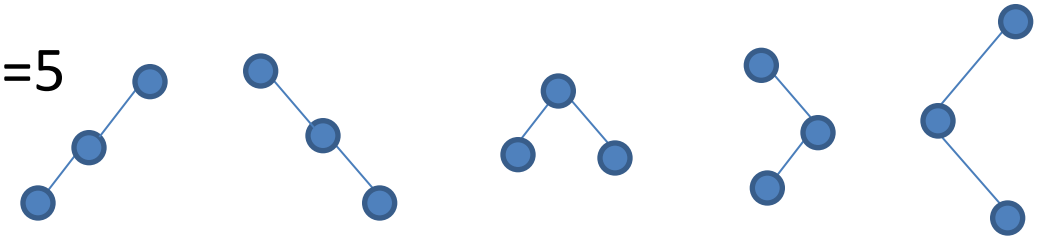
d) Le nombre d'arbres binaires de taille n est $$b_n = \frac{1}{n+1} C_{2n}^n$$

Pour n=1 ; $b_1 = 1$

Pour n=2 ; $b_2 = 2$

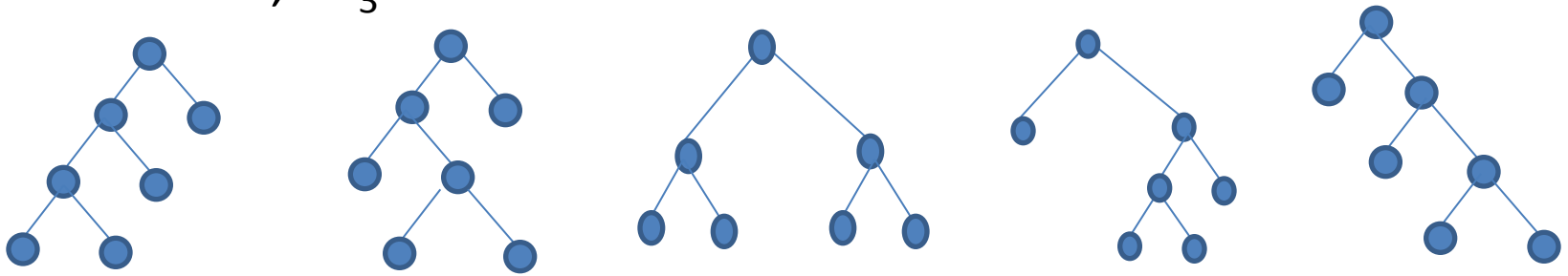


Pour n=3 ; $b_3 = 5$



e) Le nombre d'arbres binaires localement complets ayant (2n+1) nœuds est $$bc_n = \frac{1}{n+1} C_{2n}^n$$

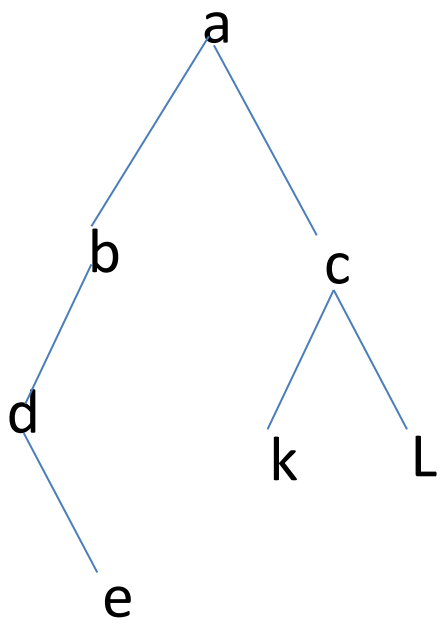
Pour n=3 ; $bc_3 = 5$



Représentation des arbres binaires

a) Utilisation des pointeurs : à chaque nœud correspondent deux pointeurs : vers le sous arbre gauche et vers le sous arbre droit.

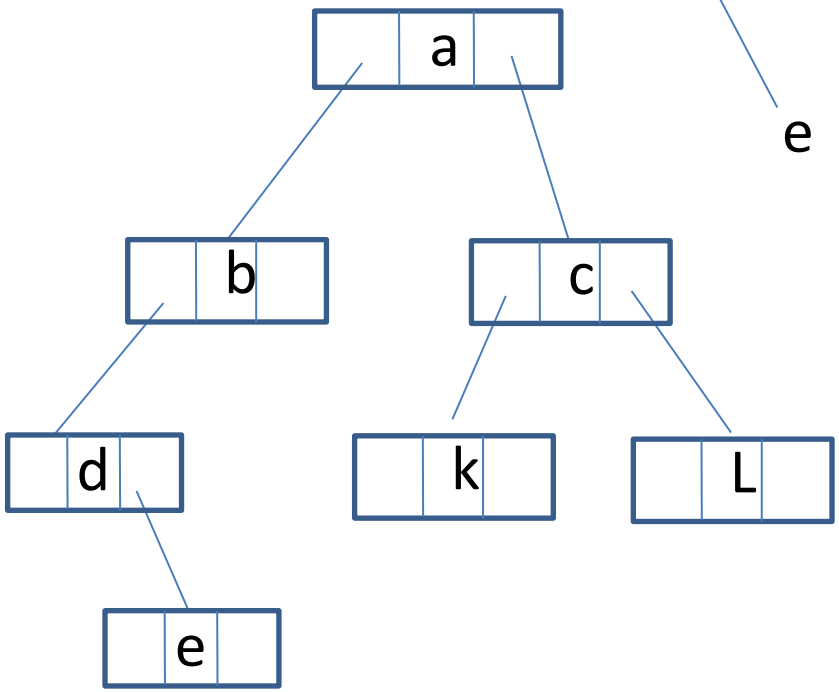
L'arbre est déterminé par l'adresse de sa racine.




```
Type    Arbre= ^Nœud ;
        Nœud= record
                  val  : Elément;
                  g, d : Arbre;
        end;
```

Opérations :

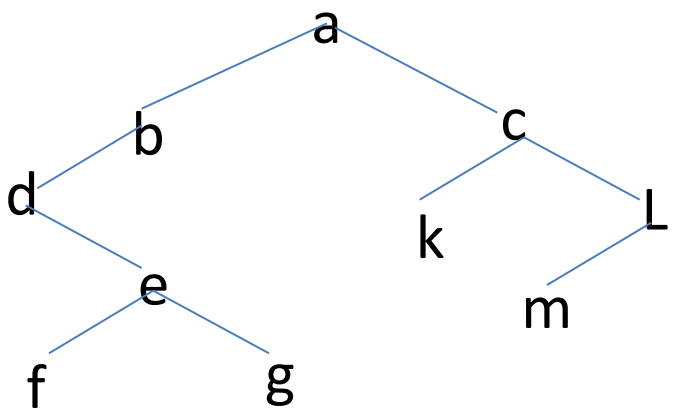
arbre-vide(A)	**A:=Nil**
Contenu(Racine(A))	A^.val
g(A)	A^.g
d(a)	A^.d



Exercice 02 : Programmer l'opération de construction :

< _ , _ , _ > : Nœud x Arbre x Arbre $\rightarrow$ Arbre pour la représentation chaînée.

b) Utilisation des tableaux :



Rac

3

tab

	V	G	D
1			
2	d	0	10
3	a	5	6
4	g	0	0
5	b	2	0
6	c	13	11
7			
8	f	0	0
9	m	0	0
10	e	8	4
11	L	9	0
12			
13	K	0	0

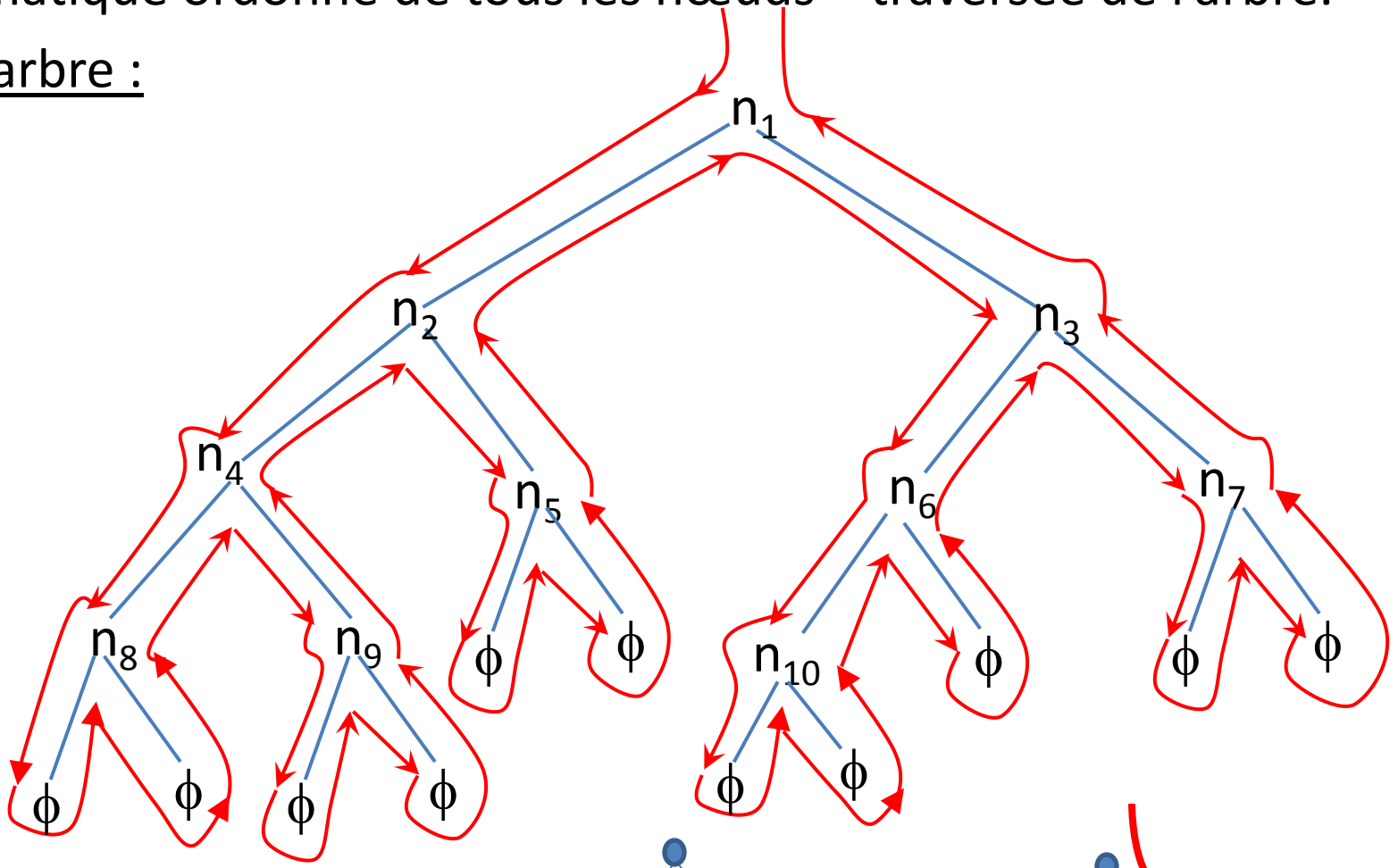
Traduction des opérations:

Arbre vide(A)	A.rac=0
Racine(A)	A.Rac
Contenu(Racine(A))	A.tab[A.rac]
g(A)	A.tab[A.rac].g
d(A)	A.tab[A.rac].d

Remarque: Les cases libres peuvent être chaînées entre elles en pile ou en file.

Parcours en profondeur à main gauche d'un arbre binaire : Examen systématique ordonné de tous les nœuds = traversée de l'arbre.

Pour un arbre :



Pour un nœud :

Algorithme récursif de parcours d'un arbre binaire

```
Procedure Parcours (A : Arbre);
Begin        If A = Arbre-vide  then Term
                                 else   begin
                                              T1 ;
                                               Parcours (g(A));
                                              T2 ;
                                               Parcours (d(A));
                                              T3 ;
                                        end;
End;
```

Cas particuliers : Ordres classiques d'exploration :

1) Ordre préfixe ou pré-ordre : Pas de T_2 + T_3 ; seul T_1 en ordre préfixe (à la descente) + Term.

   Sur l'exemple : n_1, n_2, n_4, n_8, n_9, n_5, n_3, n_6, n_{10} , n_7

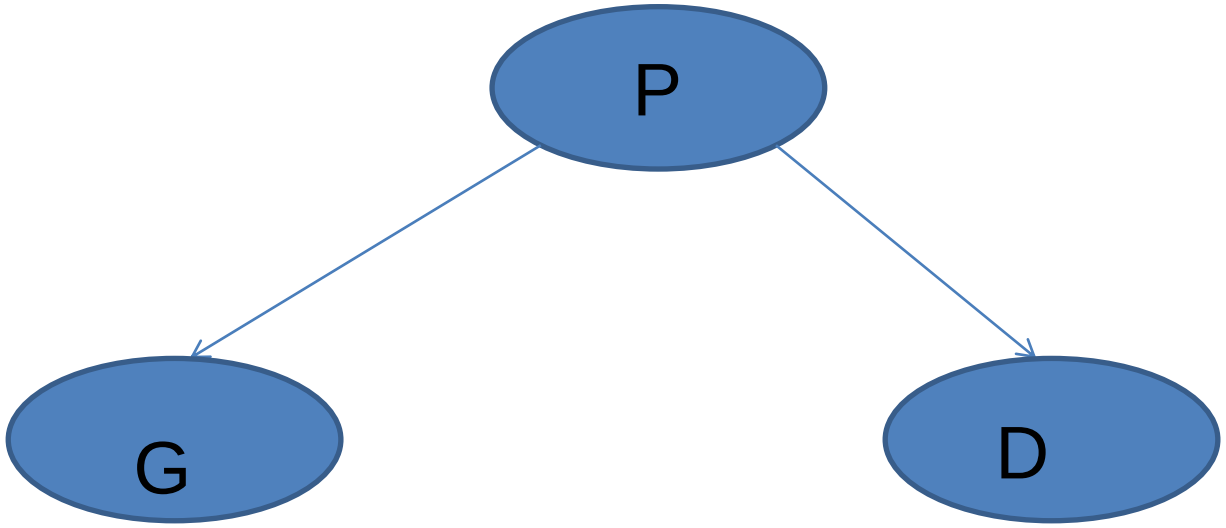
2) Ordre infixe ou symétrique : Pas de T_1 + T_3 ; seul T_2 en ordre infixe (à la montée de gauche) + Term.

   Sur l'exemple : n_8, n_4, n_9, n_2, n_5, n_1, n_{10}, n_6, n_3 , n_7

3) Ordre suffixe ou post-ordre : Pas de T_1 + T_2 ; seulT_3 en ordre suffixe (à la montée de droite) + Term.

   Sur l'exemple : n_8, n_9, n_4, n_5, n_2, n_{10}, n_6, n_7, n_3 , n_1

Remarque : l'ordre hiérarchique $\Leftrightarrow$ parcours par niveaux.



Essayer cette simplification des parcours

Préfixe : P-G-D

Infixe : G-P-D

Suffixe : G-D-P

Exercices du TD N° 04

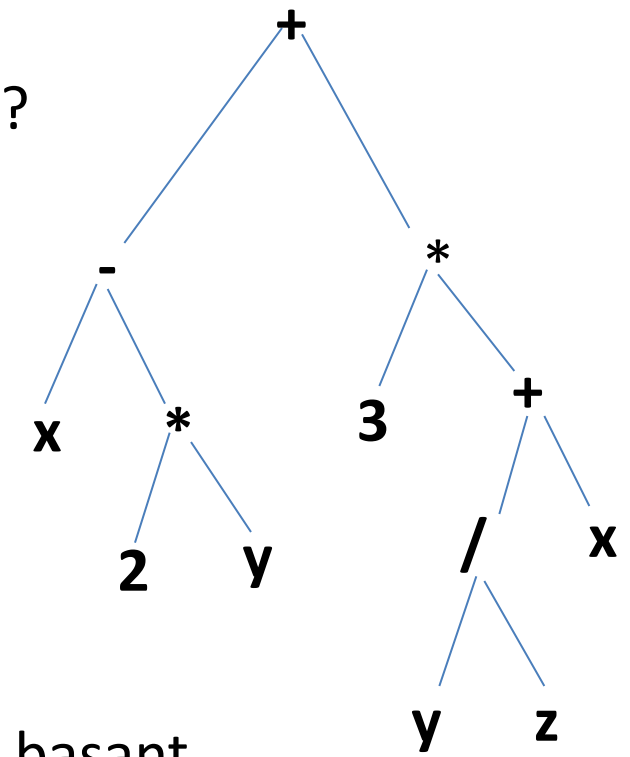
Exercice 03 : Donner tous les arbres binaires localement complets dont le parcours symétrique donne l'expression 2+3*4+5

Exercices du TD N° 04

Exercice 04 : Une expression arithmétique peut être représentée par un arbre binaire. S'il ne s'agit que d'opérateurs binaires (+, -, *, /) l'arbre est localement complet.

Soit l'expression arithmétique représentée par l'arbre A :

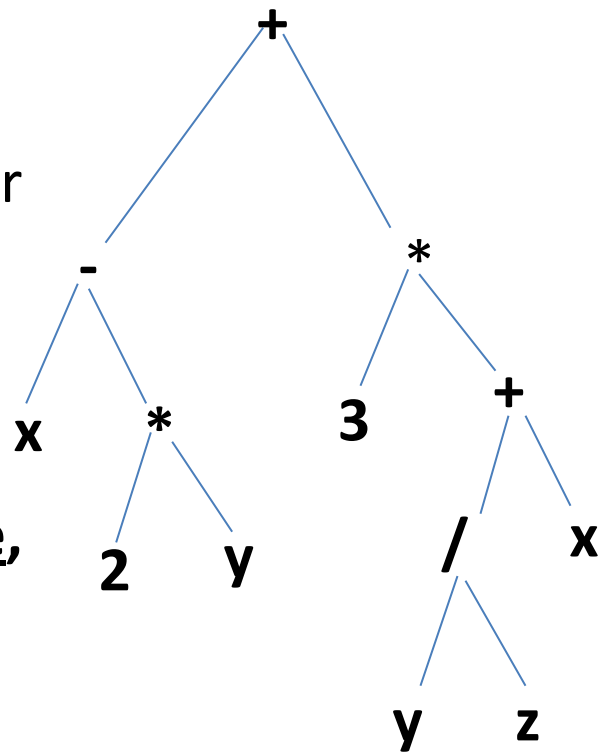
1) A est-il dégénéré ? Complet ? Parfait ?
2) A contient – il un sous arbre peigne gauche ?
   Peigne droit ?
3) Donner les expressions représentées par A lorsque A est parcouru en ordre préfixe, infixe et post-fixe. Conclure
4) Donner une procédure qui permet de bien parenthéser (ajouter des parenthèses à) une expression représentée par un arbre, en se basant sur la procédure du parcours donnée en cours.



SUITE du TD N° 04

Exercice 05 :

Soit l'expression arithmétique représentée par l'arbre binaire localement complet A :



Donner les expressions représentées par A

Dans le parcours en profondeur à main **droite**, préfixe,

infixe et post-fixe. Conclure

Arbres Planaires généraux ou arbres généraux

Le nombre de fils d'un nœud peut être > 2.

TAD Arbre :

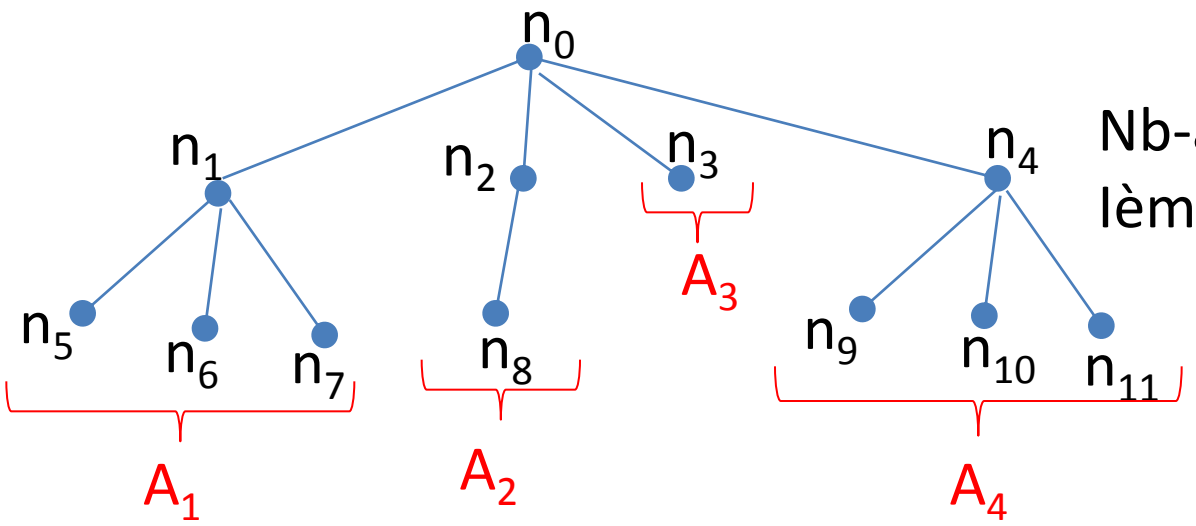
A=<o, A_1, A_2, . . . , A_p > où o : racine et A_1, A_2, . . . , A_p : arbres disjoints.

<A_1, A_2, . . . , A_p > est appelée forêt.

D'où : A = <o, F> avec F = ϕ + A + <A, A> + <A, A, A> + . . .

A: ensemble des arbres. F= ensemble des forêts. ϕ=forêt vide.

Exemple : Soit l'arbre A suivant :



Racine (A)=n_0

List-arbres (A)= <A_1, A_2, A_3, A_4>
= F

Nb-arbres (F) = 4

Ième(F, 2)= A_2

Signature des TAD Arbre, Forêt :

Sorte ArbreGen, Forêt

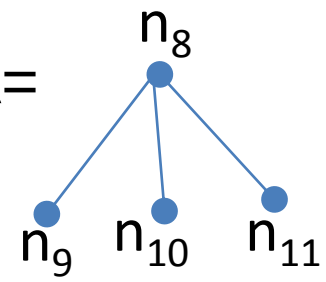
Utilise Nœud, Entier

Opérations

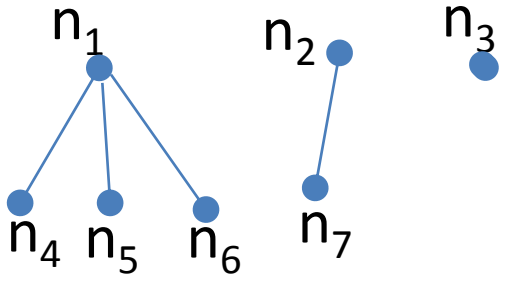
cons	: Nœud x Forêt	→ ArbreGen
racine	: ArbreGen	→ Nœud
list-arbres	: ArbreGen	→ Forêt
Forêt-vide	:	→ Forêt
ième	: Forêt x Entier	→ ArbreGen
nb-arbres	: Forêt	→ Entier
insérer	: Forêt x Entier x ArbreGen	→ Forêt

- Possibilité d'extension par d'autres opérations : supprimer un arbre dans une forêt, contenu d'un nœud . . . etc.
- Pas de notion gauche droite
- Fils ordonnés de la gauche vers la droite
- Un arbre n'est jamais vide.

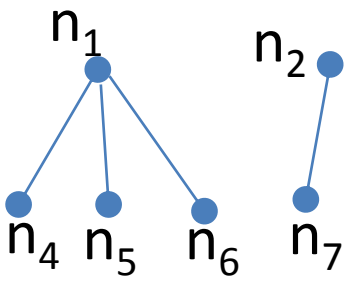
A=

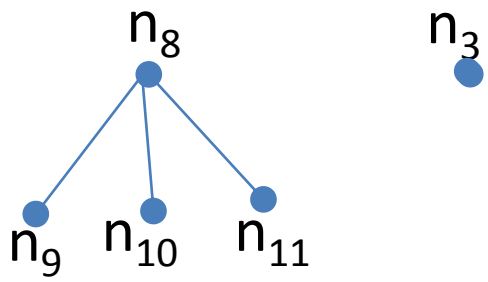


F=



F'=
inserer(F, i=3, A)





ième (F', k=1 / 2)= ième(F, k)

ième (F', k=3) = A

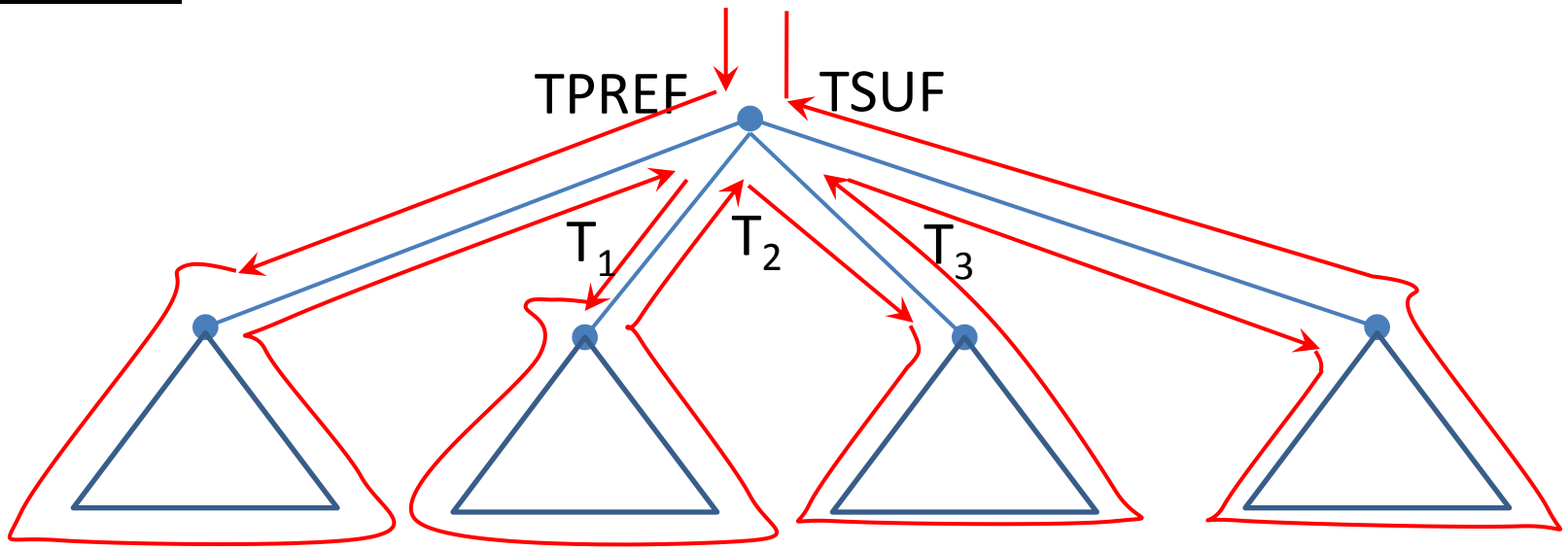
ième (F', k=4)= ième(F, k-1)

Parcours des arbres

Extension du parcours des arbres binaires. Ainsi, un nœud ayant n fils est parcouru (n+1) fois.

- au premier passage, appliquer le traitement TPREF
- au (i+1) ème passage, le traitement T(i)
- au dernier passage, le traitement TSUF
- au nœuds sans fils, le traitement TERM

Exemple :



Soit l'opération feuille : Arbre → Booléen telle que :

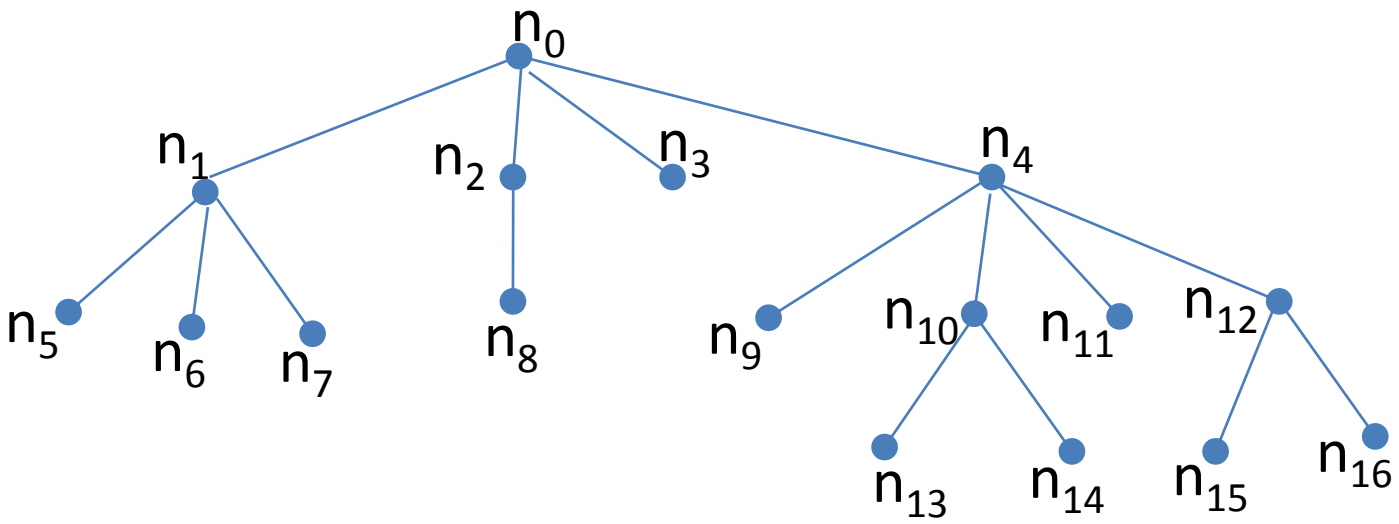
feuille (A) ← (list-arbres (A) =forêt-vide)

Procedure parcrs (A : ArbreGen);

```
Var i, nb : Integer;
Begin
    nb:=nb-arbres (list-arbres(A));
    if feuille (A) then  TERM
                else   Begin  TPREF; {traitement avant de voir ses fils}
                              for i:= 1 to nb-1 do
                                begin  parcrs (ième (list-arbres (A),i));
                                       T(i);
                                 end;
                              Parcrs (ième (list-arbres (A),nb));
                              TSUF;  {traitement après avoir vu ses fils}
                       end;
End;
```

Suite au TD N° 04

Exercice 06: Soit l'arbre général suivant :



1) Donner une procédure qui permet de parcourir l'arbre comme affiché dans l'expression :

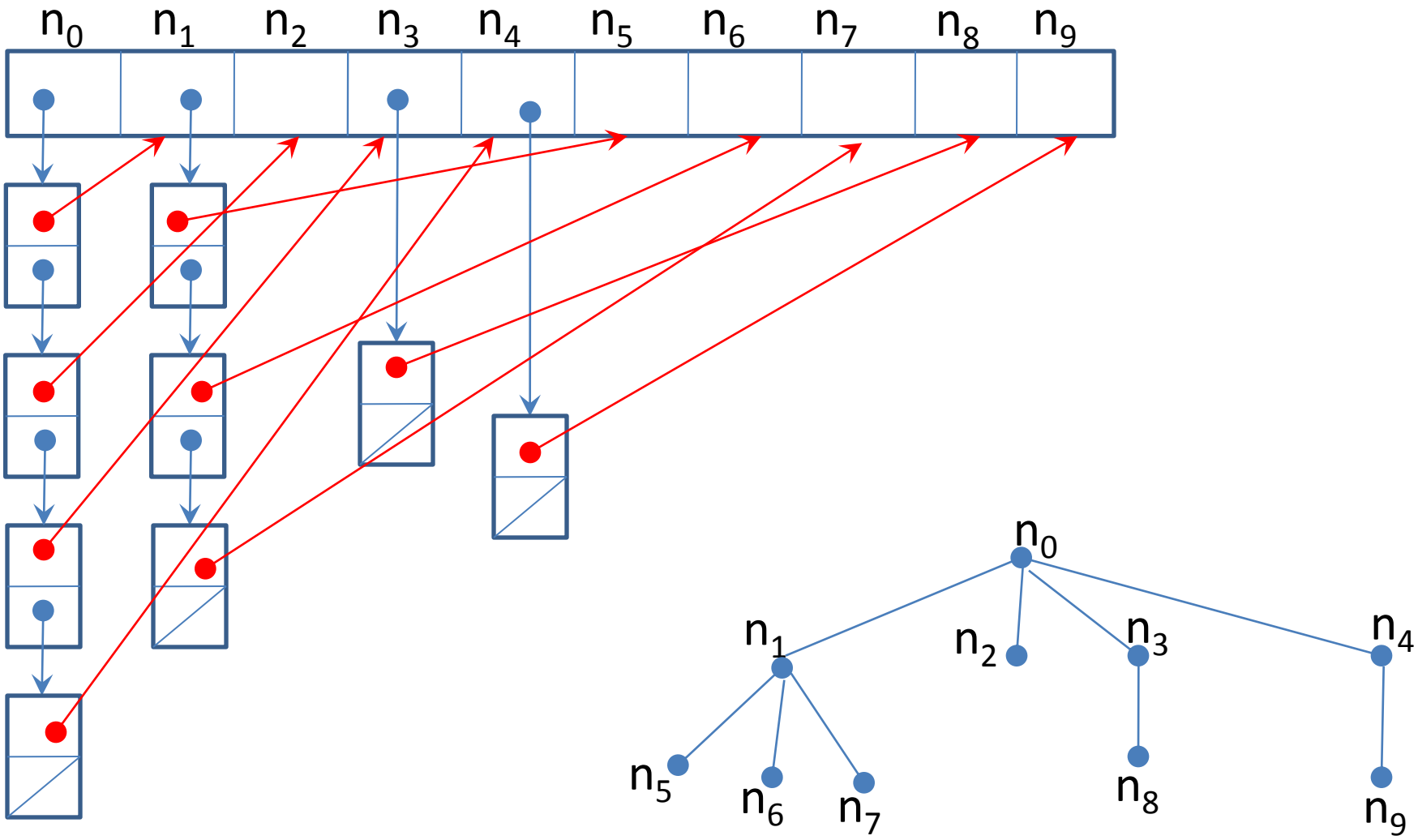
$(n_0(n_1(n_5)(n_6)(n_7))(n_2\ (n_8))(n_3)(n_4(n_9)(n_{10}(n_{13})(n_{14}))(n_{11})(n_{12}(n_{15})(n_{16}))))$

2) Même question pour :

$n_0(n_1(n_5, n_6, n_7)\ ,\ n_2\ (n_8),\ n_3\ ,n_4(n_9,\ n_{10}(n_{13},\ n_{14}),\ n_{11},\ n_{12}(n_{15},\ n_{16})))$

Représentation des arbres généraux

a) ***Représentation par liste de fils :*** cette représentation se prête mal à une gestion dynamique à cause du tableau de nœuds.

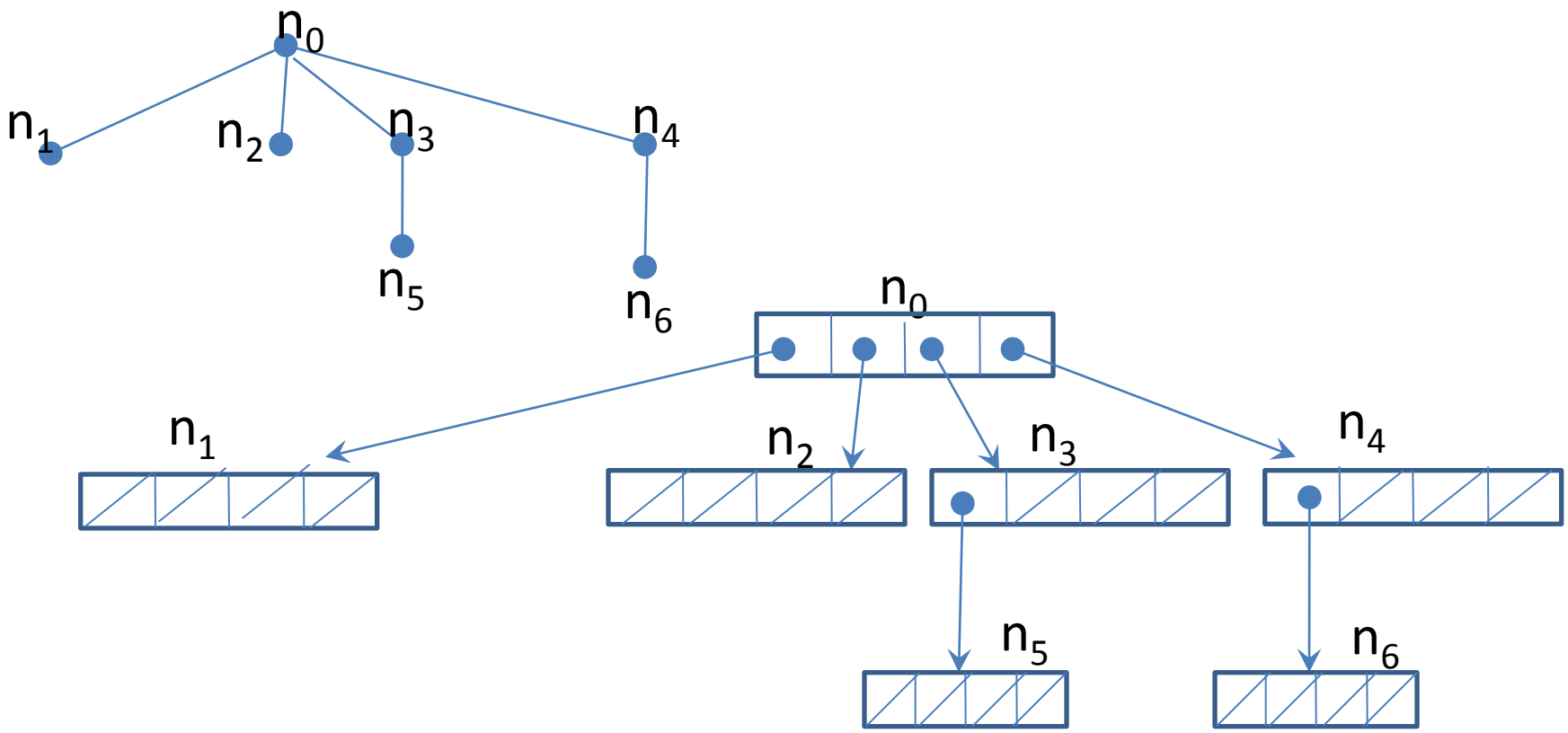


Suite au TD N° 04

Exercice 07 : Donner les déclarations pour représenter un arbre général par liste de fils.

b) *Représentation par un pointeur vers chaque sous arbre :*

représentation trop gourmande en espace mémoire.



c) *Représentation sous forme d'arbres binaires :*

la taille d'un arbre général est le nombre total de ses nœuds.

taille (forêt-vide)=0

taille (insérer (F, i, A)) = taille (F) + taille (A)

taille (cons (o, F)) = 1+ taille (F)

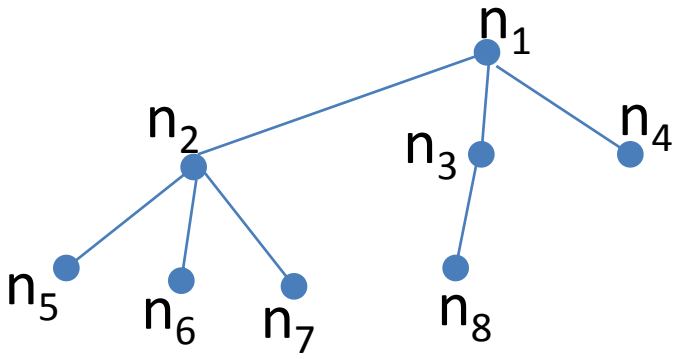
Lemme : Il existe une bijection entre les arbres généraux de taille n+1 et les forêts de taille n.

Preuve :

On associe à l'arbre A= <o, A_1, A_2, . . . , A_p> la forêt F=<A_1, A_2, . . . , A_p>

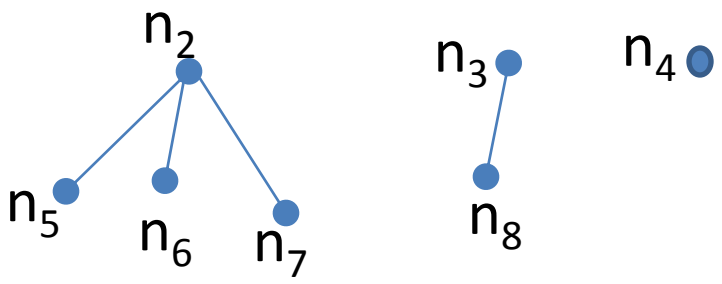
Exemple :

A



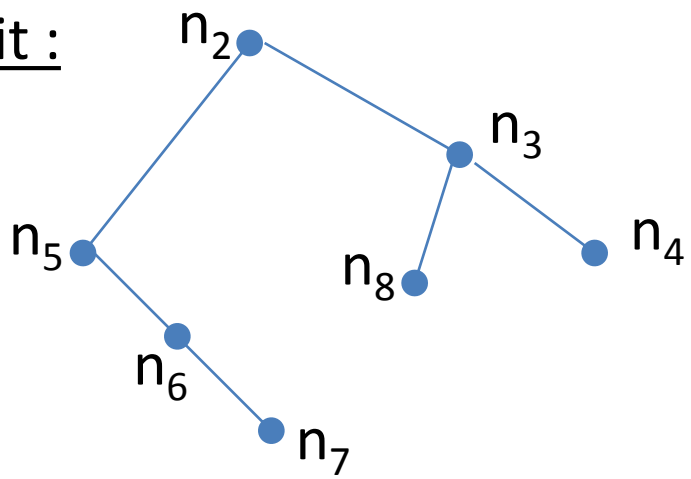
⇒

F



Bijection fils aîné – frère droit :

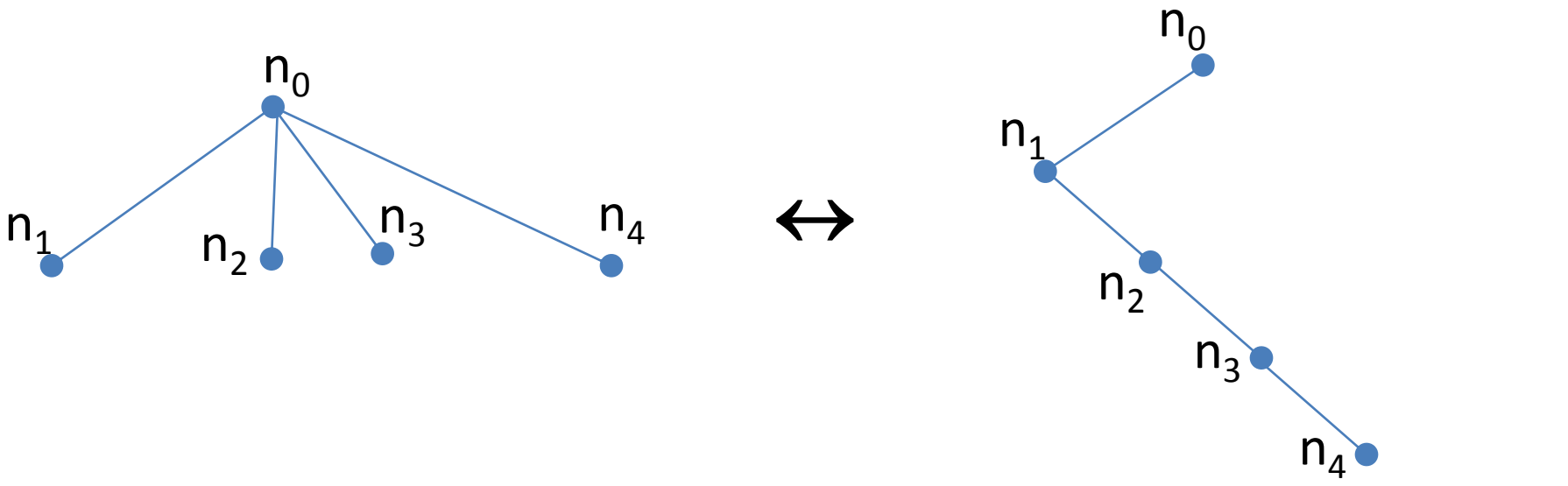
F ⇒



Proposition 1 : Il existe une bijection entre les forêts de taille n et les arbres binaires de taille n.

Preuve :

- L'arbre binaire se construit par le procédé suivant : étant donné un nœud dans une forêt, on crée un lien gauche vers son premier fils (fils aîné) et un lien droit vers son frère situé à droite.
- La forêt se construit à partir de l'arbre binaire par la conservation des liens gauches et la suppression des liens droits tout en les mettant au même niveau que le fils aîné.



Suite au TD N° 04

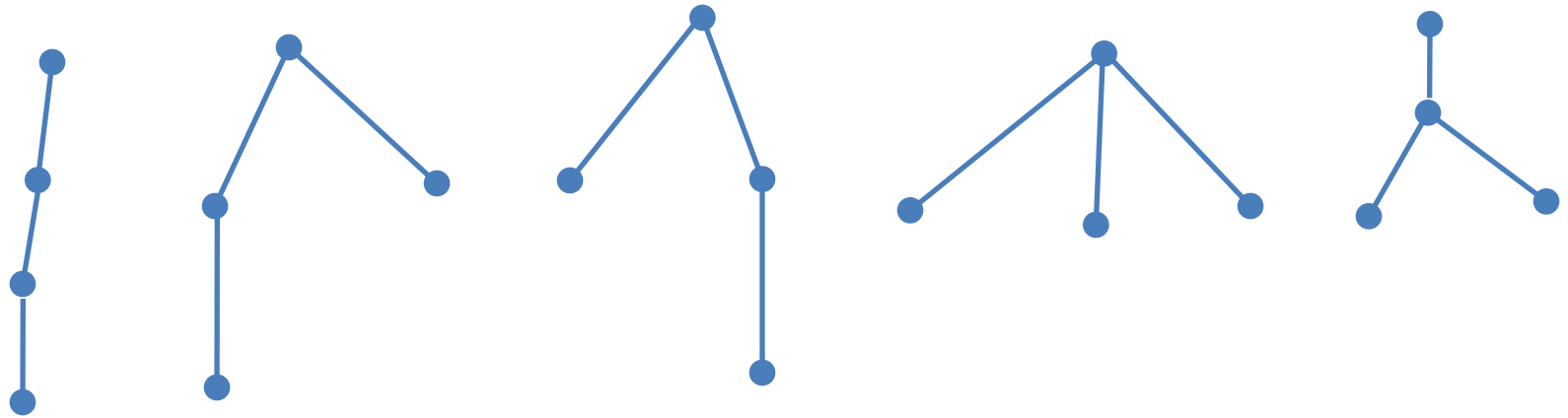
Exercice 08 : Programmer la bijection fils aîné – frère droit pour transformer un arbre général en un autre binaire.

Proposition 2 : Le nombre a_{n+1} d'arbres planaires de taille n+1 est :

$$a_{n+1} = \frac{1}{n+1} C_{2n}^{n}$$

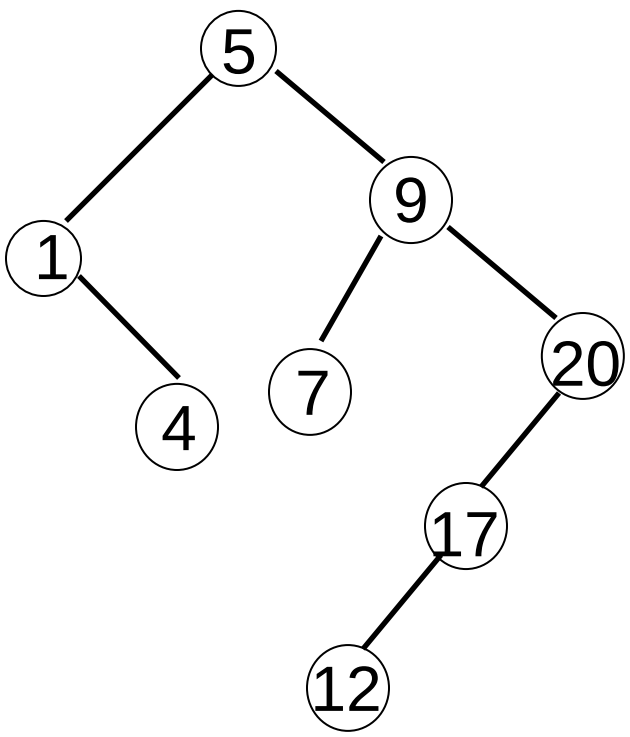
Exemple : Pour n=3; $a_4 = \frac{1}{4} C_6^3 = 5$

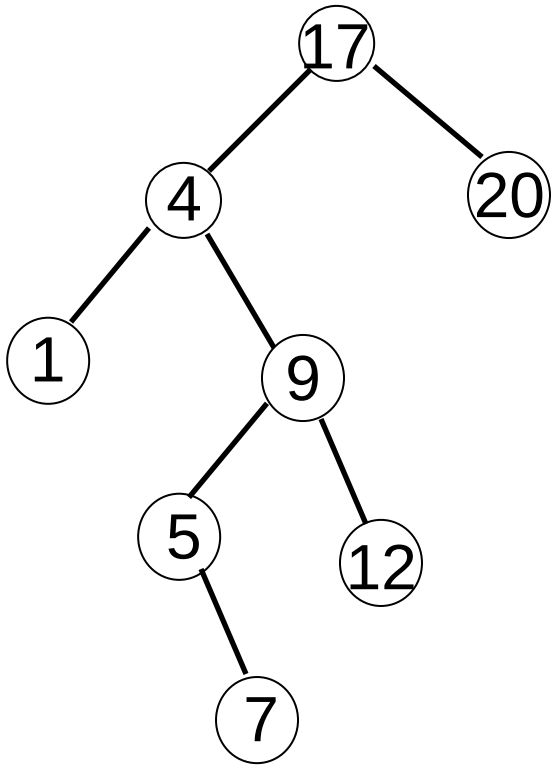
Avec 4 nœuds, on peut construire 5 arbres différents :



Arbres binaires de recherche

Exemples d'arbres binaires de recherche





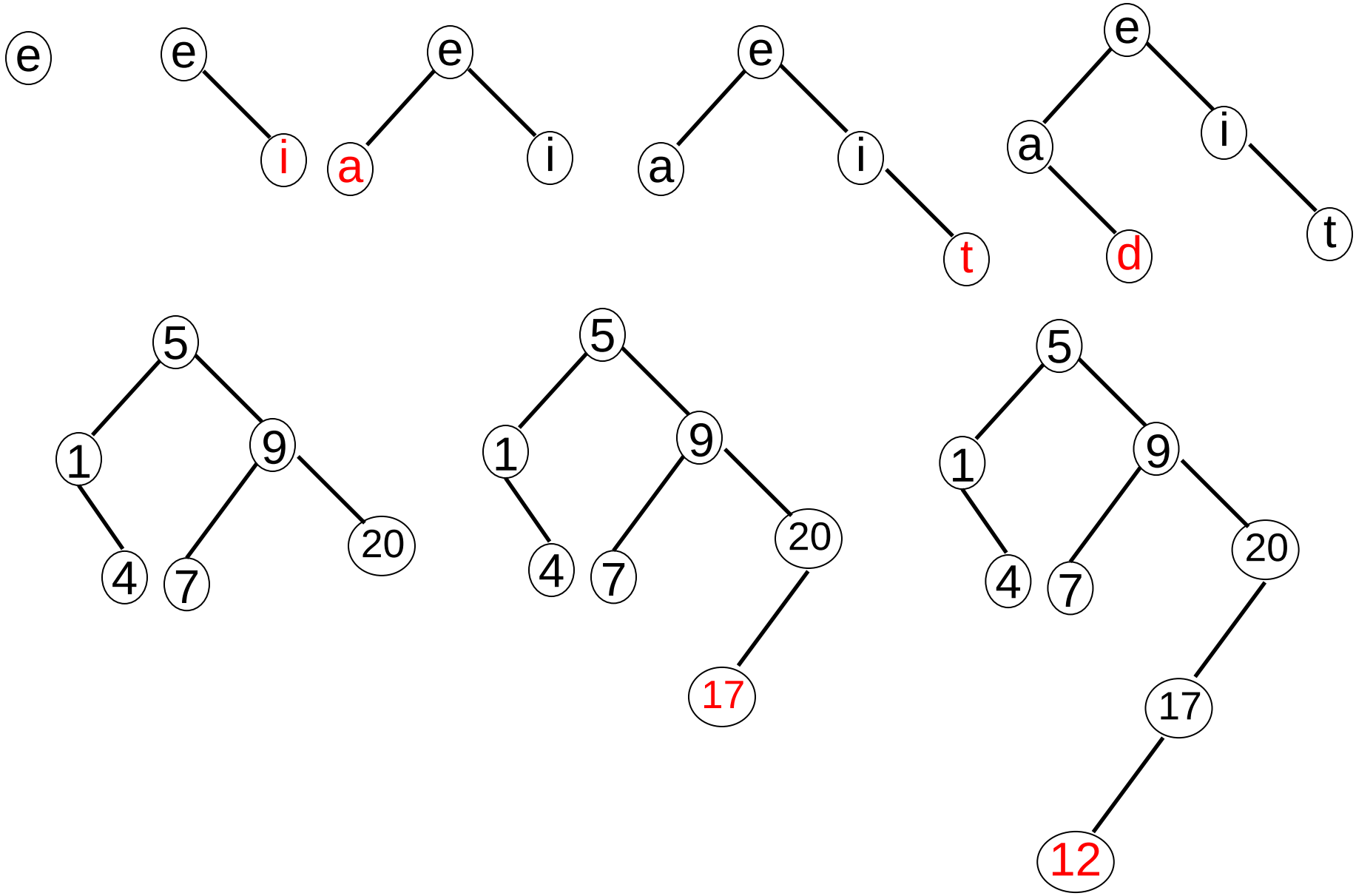
Définition : Un arbre binaire de recherche est un arbre binaire étiqueté tel que pour tout nœud **v** de l'arbre :

-Les éléments de tous les nœuds du sous arbre gauche de **v** sont inférieurs ou égaux à l'élément **v** .

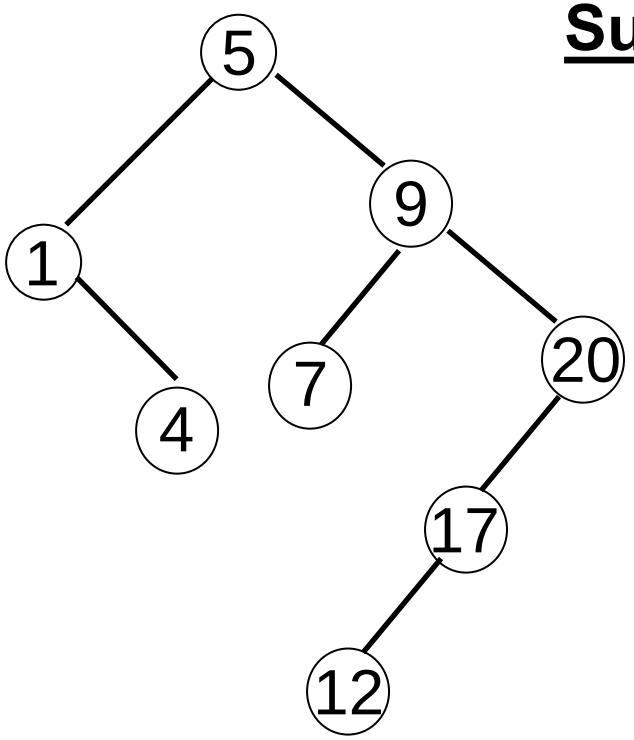
- Les éléments de tous les nœuds du sous arbre droit de **v** sont supérieurs à l'élément **v.**

Il résulte, immédiatement, de cette définition que le parcours symétrique d'un arbre binaire de recherche produit la suite des éléments triée en ordre croissant.

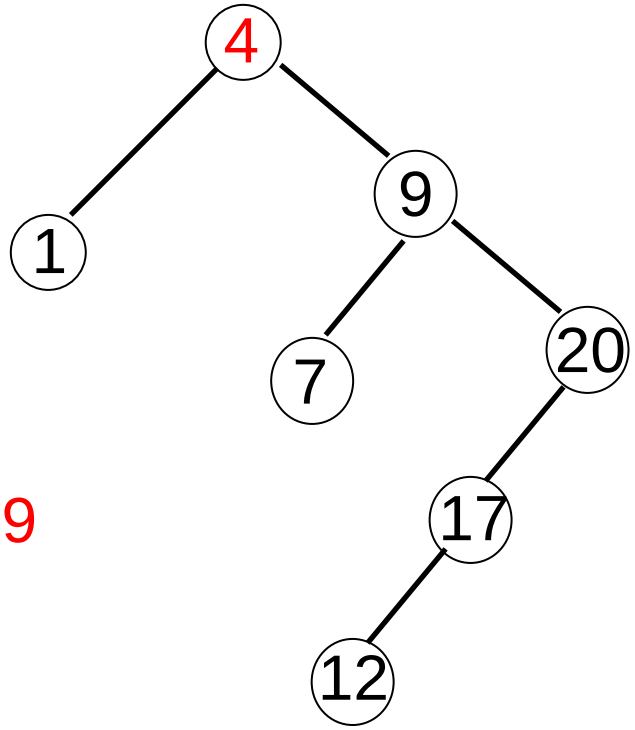
Construction par adjonctions successives aux feuilles



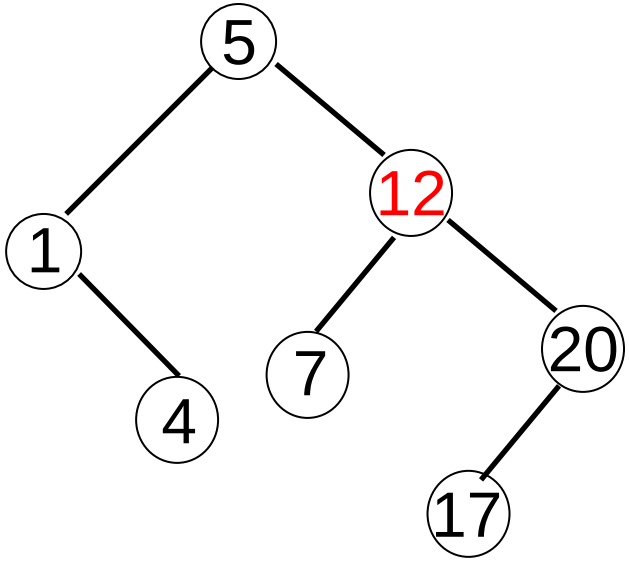
Suppression d'un élément

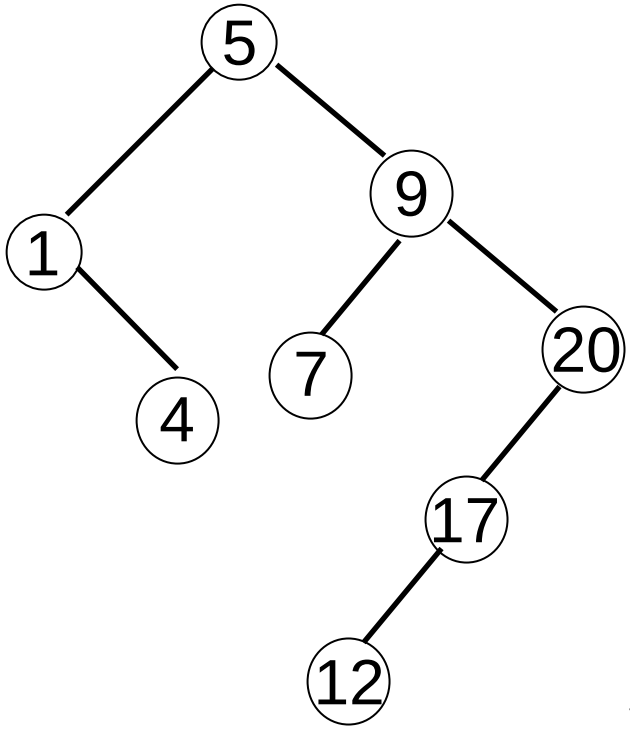


Suppression de 5

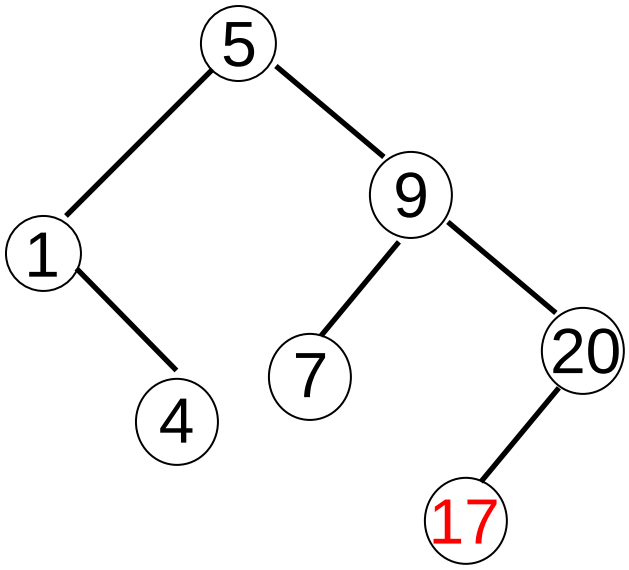


Suppression de 9

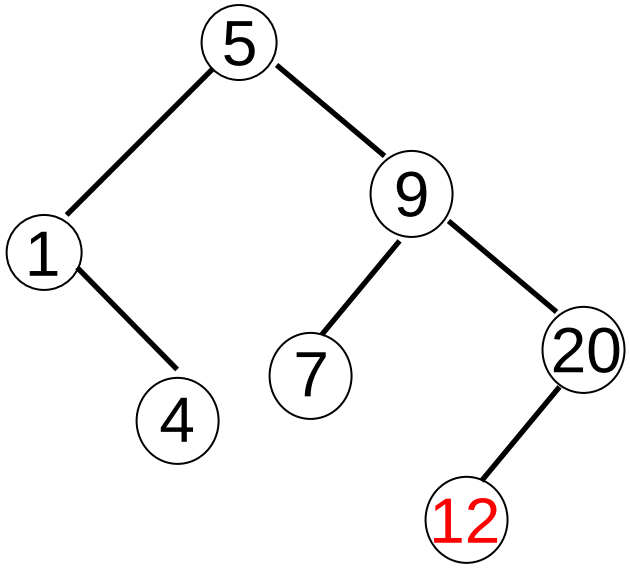




Suppression de 12

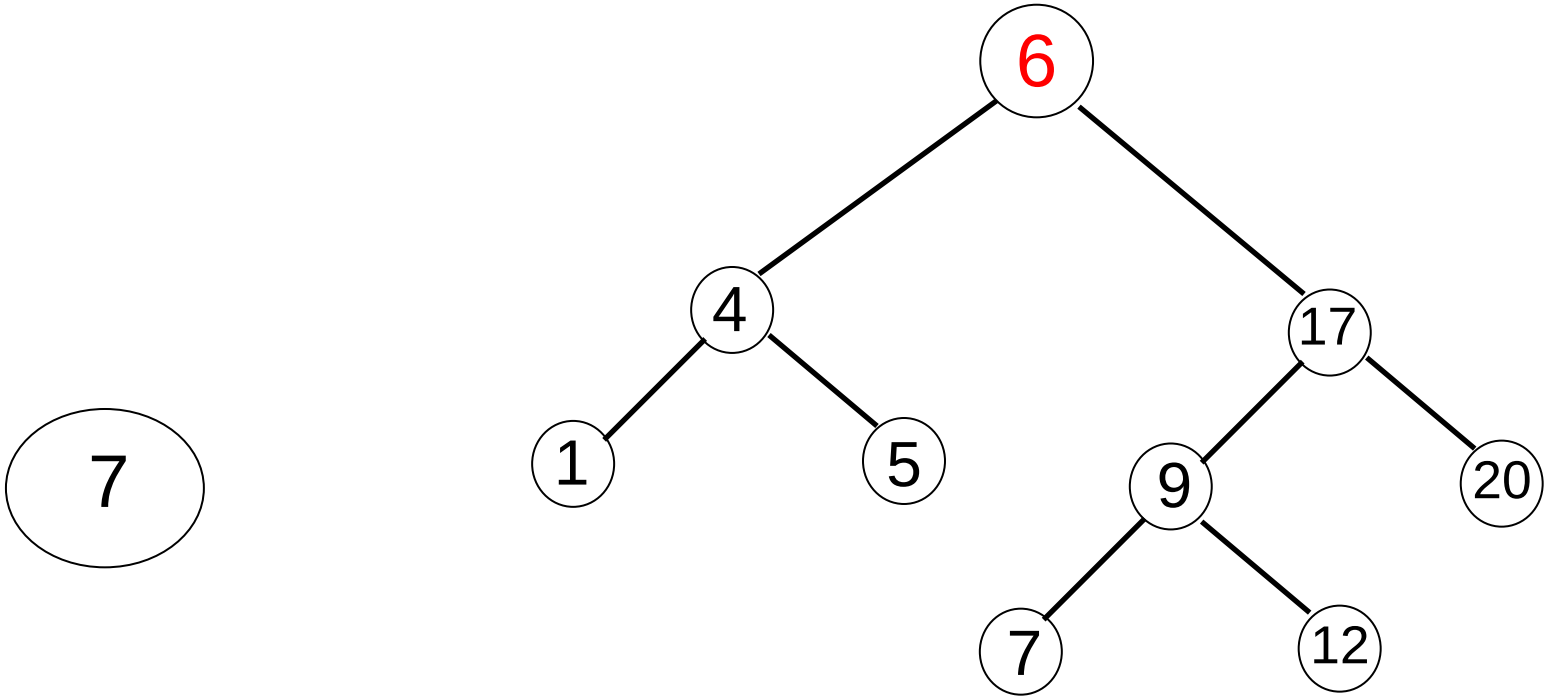


Suppression de 17



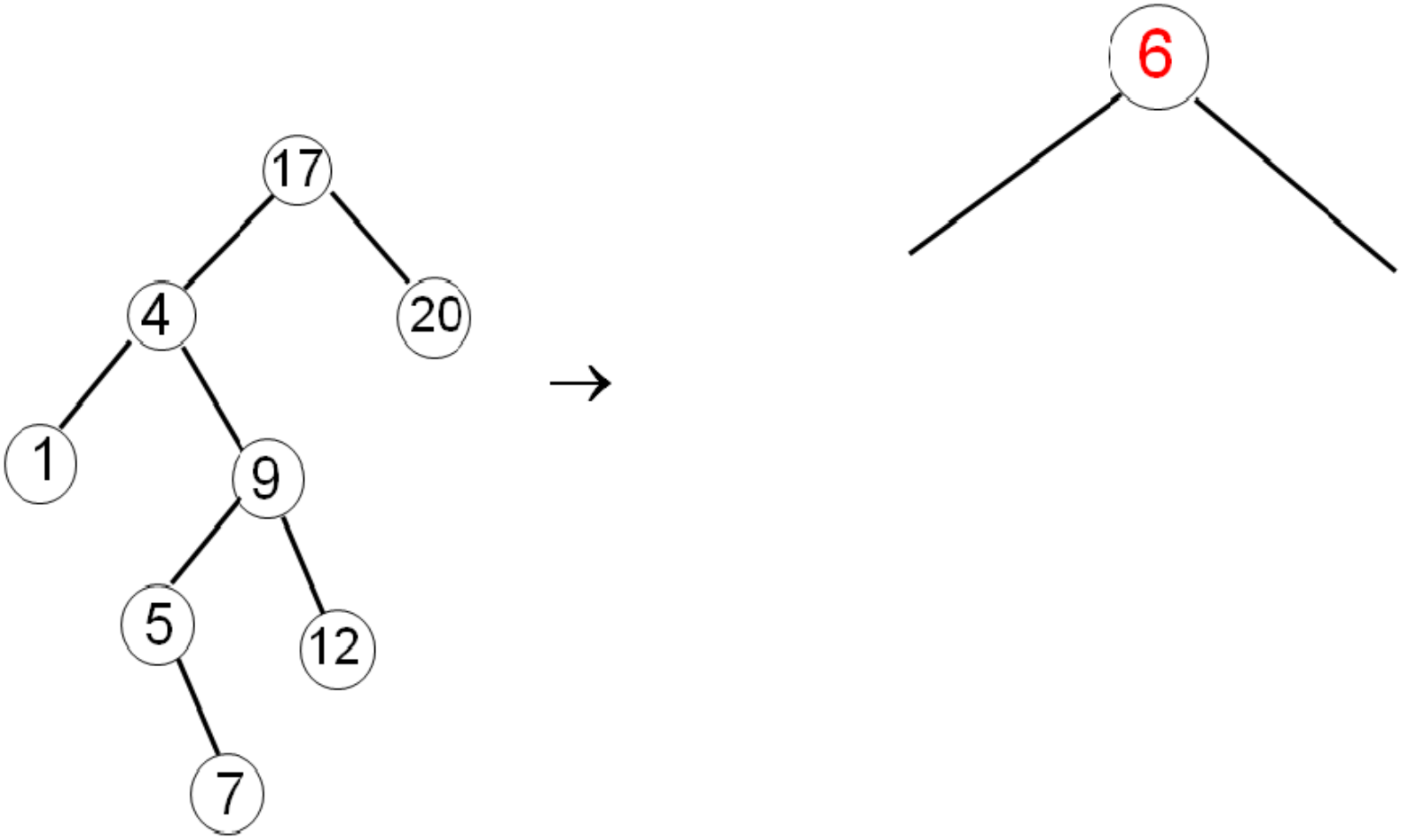
Adjonction à la racine et coupure

Coupure selon l'élément 6



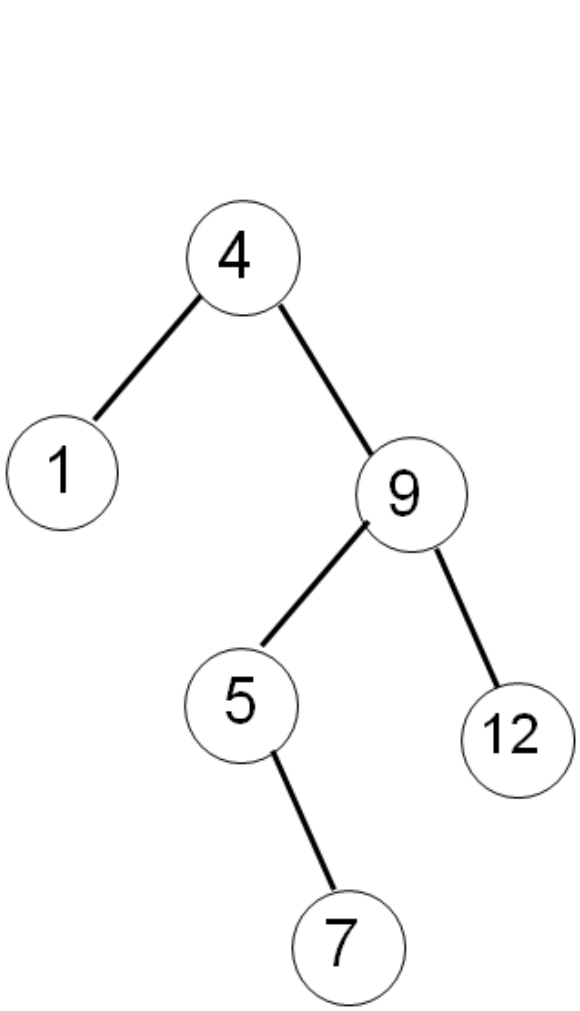
Adjonction à la racine et coupure

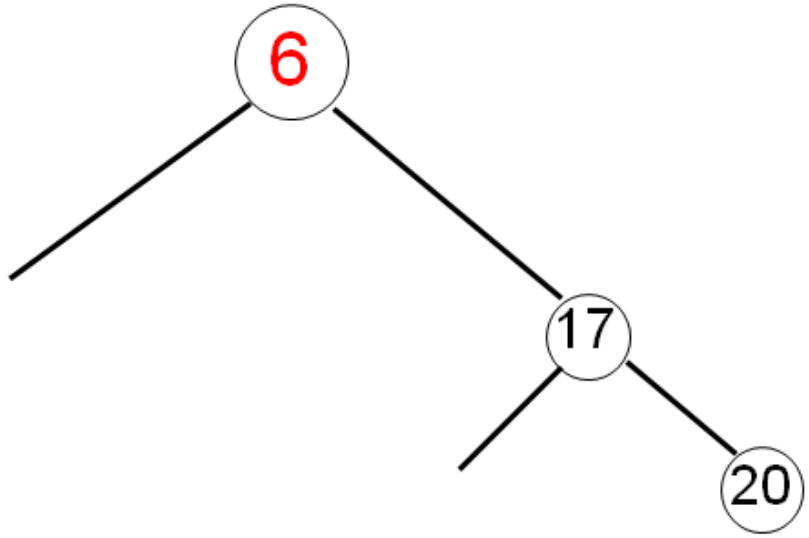
Coupure selon l'élément 6



Adjonction à la racine et coupure

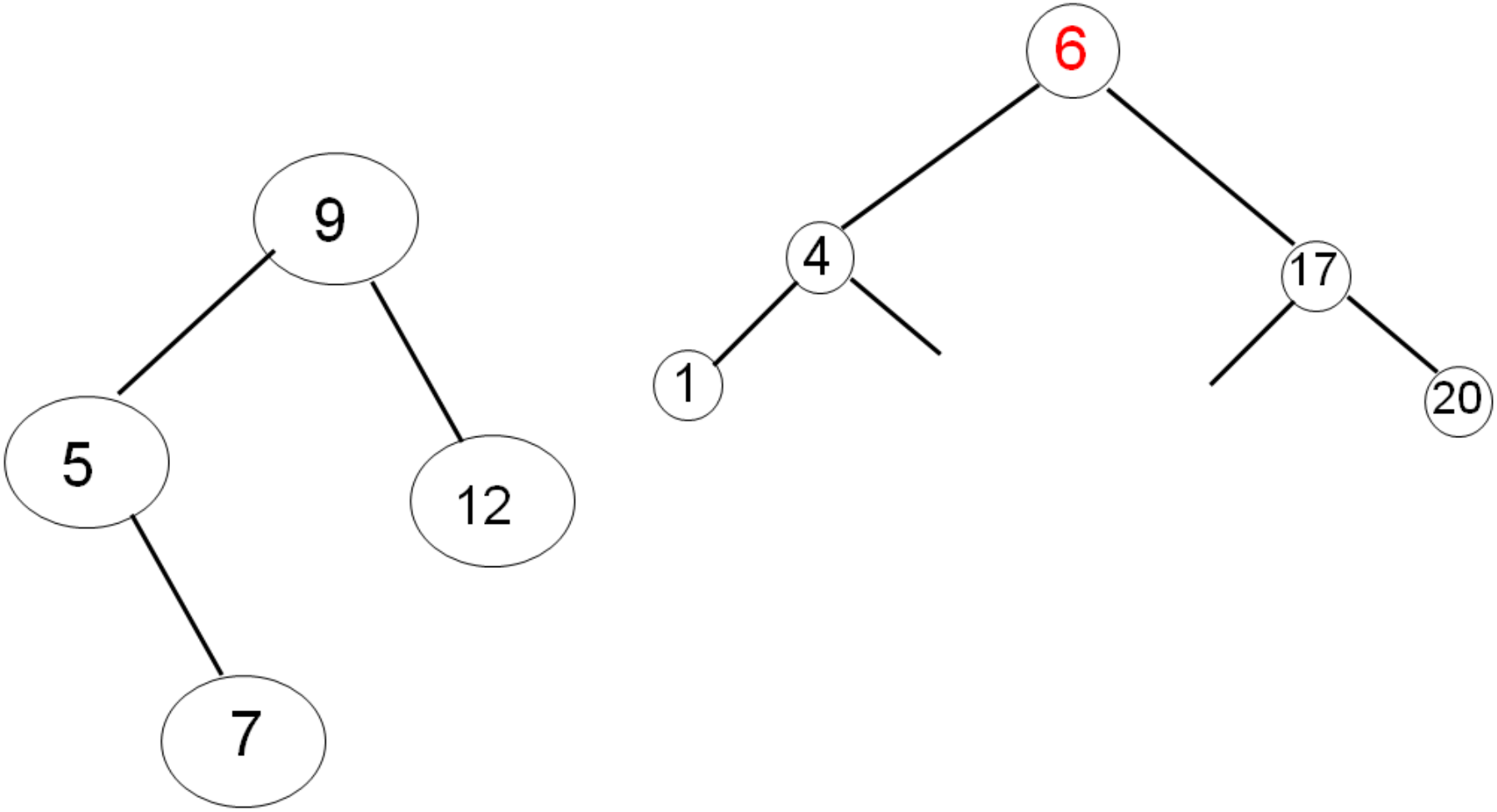
Coupure selon l'élément 6





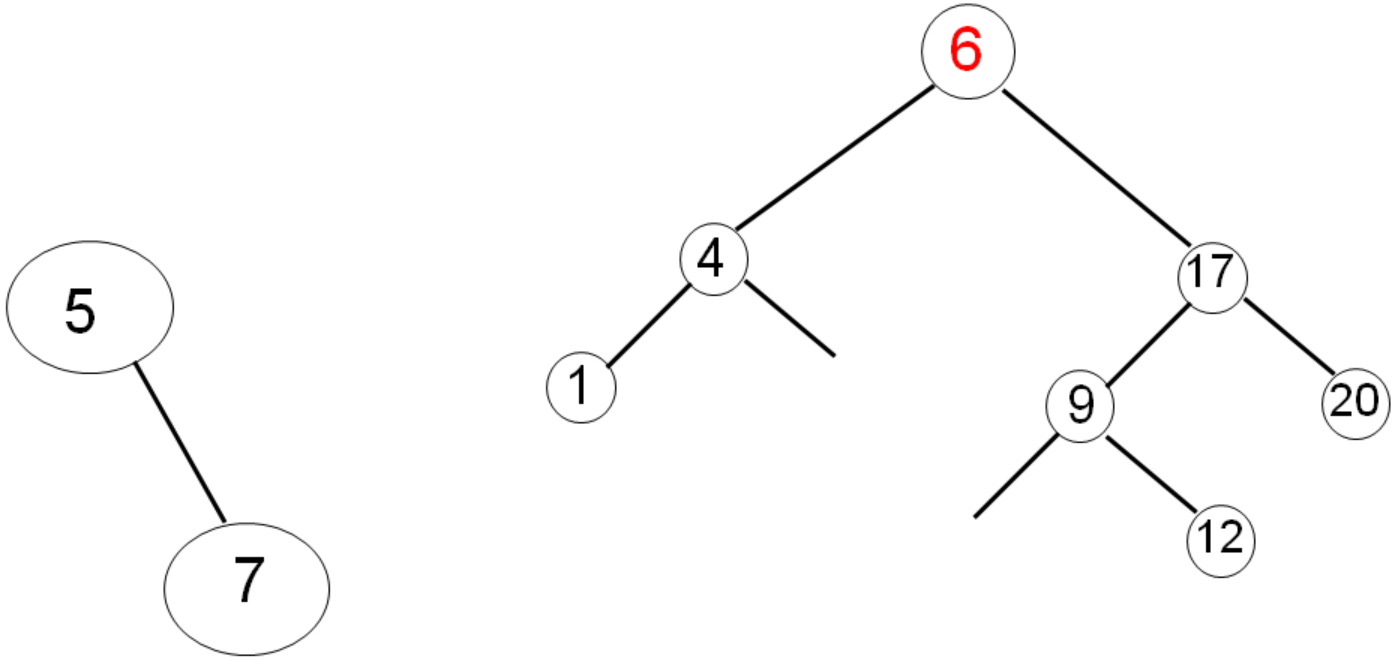
Adjonction à la racine et coupure

Coupure selon l'élément 6



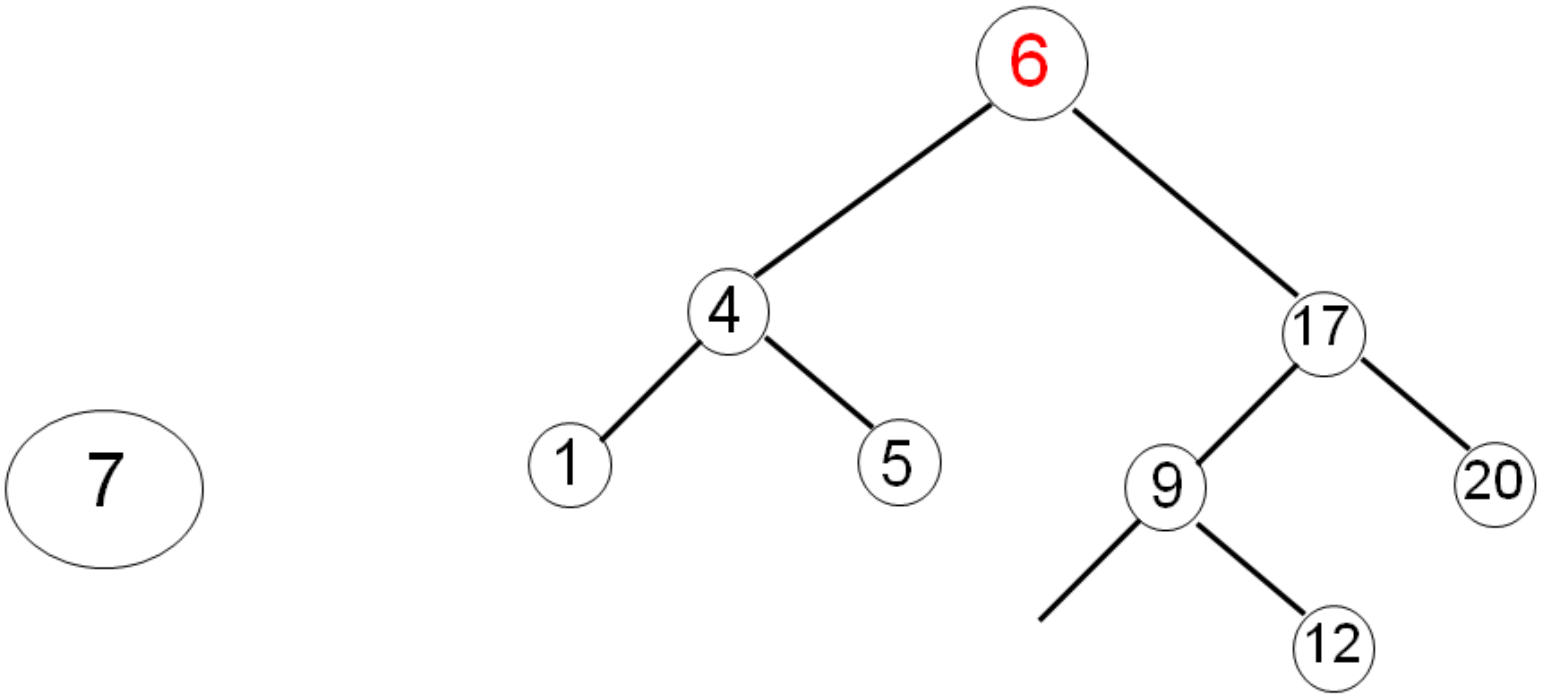
Adjonction à la racine et coupure

Coupure selon l'élément 6



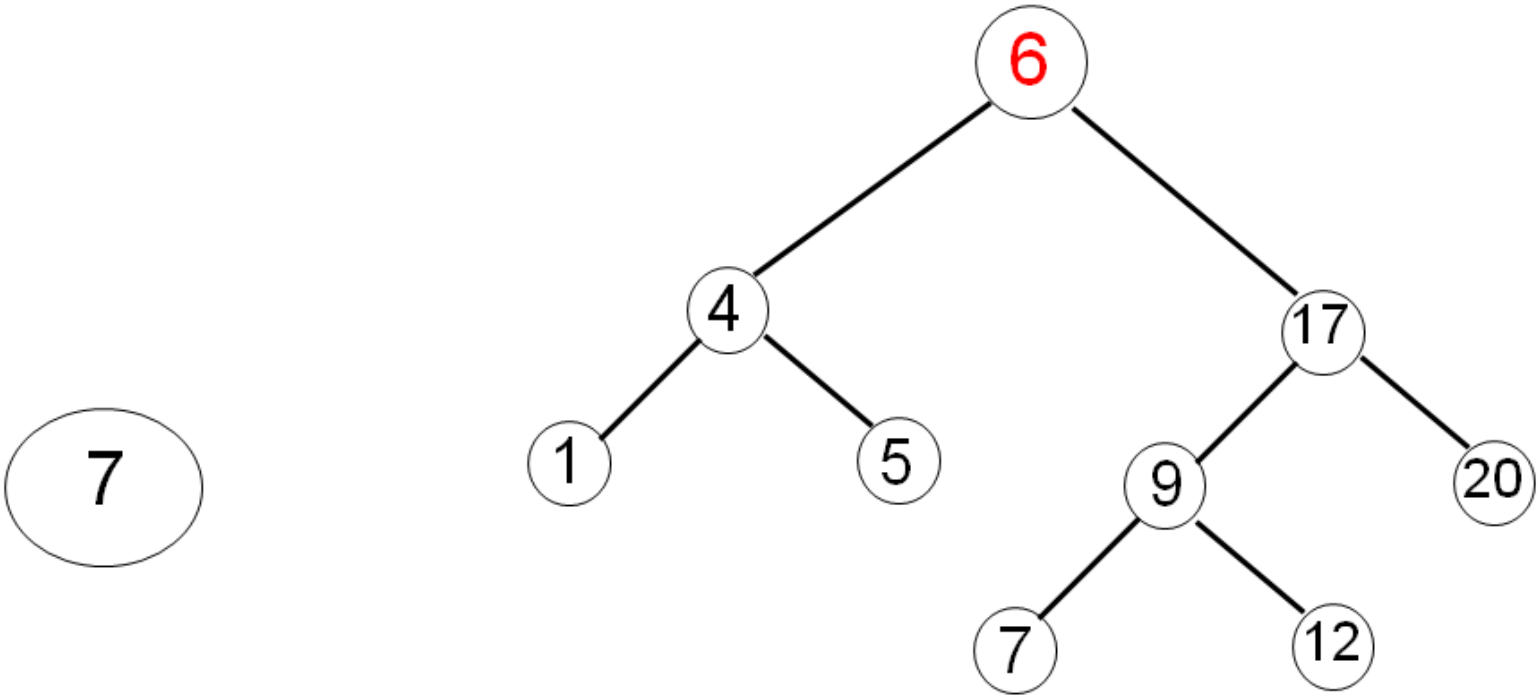
Adjonction à la racine et coupure

Coupure selon l'élément 6



Adjonction à la racine et coupure

Coupure selon l'élément 6



Suite au TD N° 04

Exercice 08 : Programmer l'adjonction aux feuilles pour les arbres binaires de recherche (ABR)

Exercice 09 : Programmer l'adjonction à la racine d'un ABR

Exercice 10 : Programmer la suppression dans un ABR

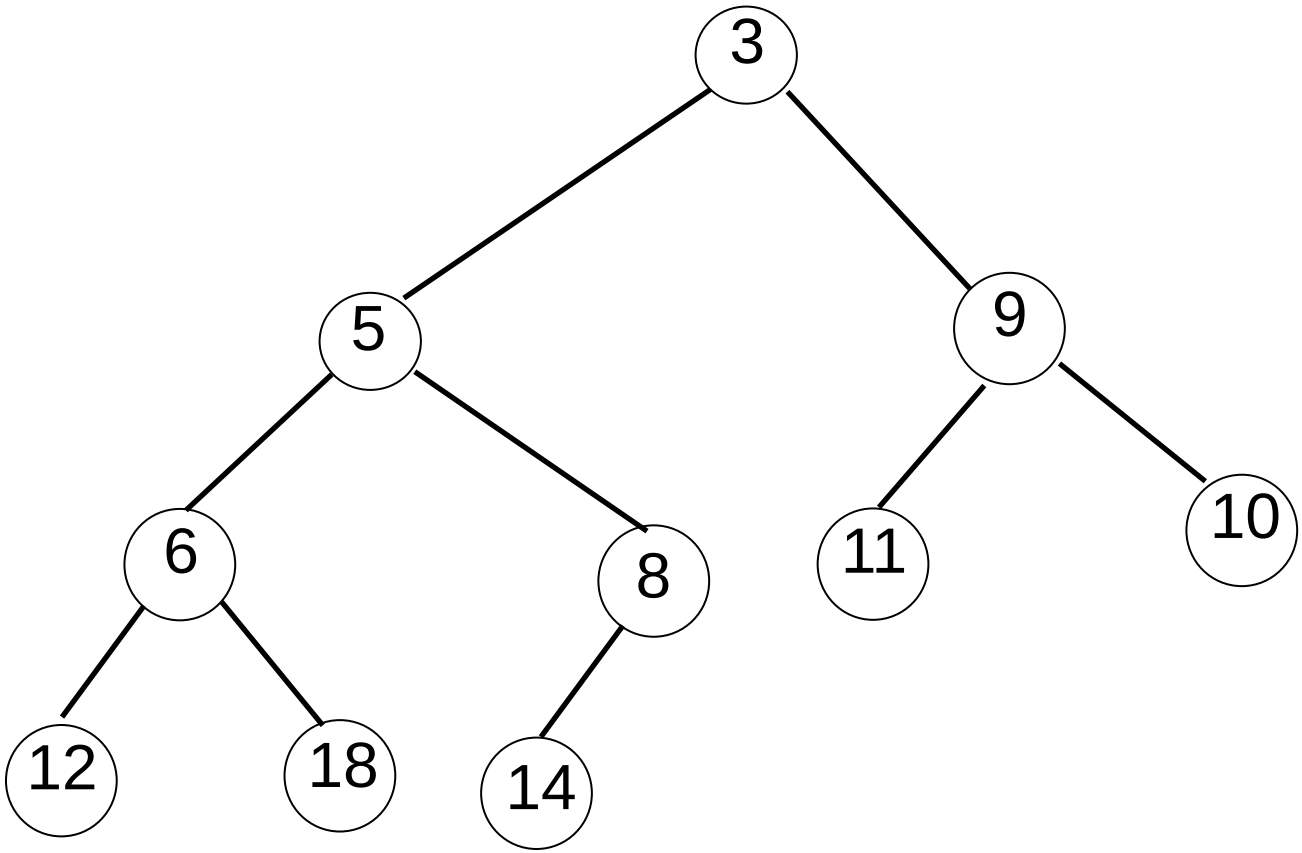
Structure arborescente : le Tas (Heap)

Structure du tas : Définition

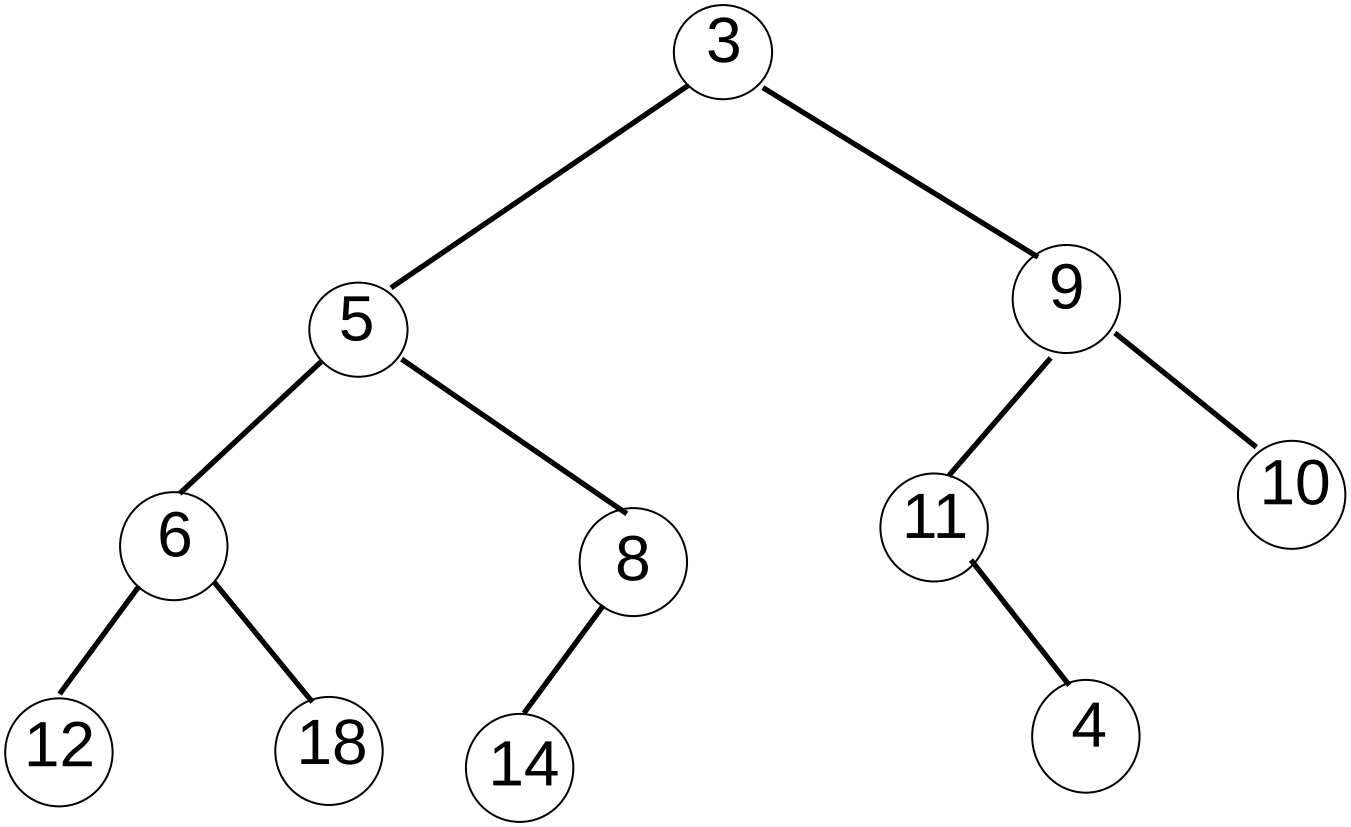
On appelle ***tas*** *un tableau représentant un* ***arbre binaire parfait partiellement ordonné****.*

Arbre parfait

c'est un arbre binaire dont tous les niveaux sont remplis sauf éventuellement le dernier niveau dont les nœuds doivent être regroupés à partir de la gauche de l'arbre.



un arbre binaire parfait

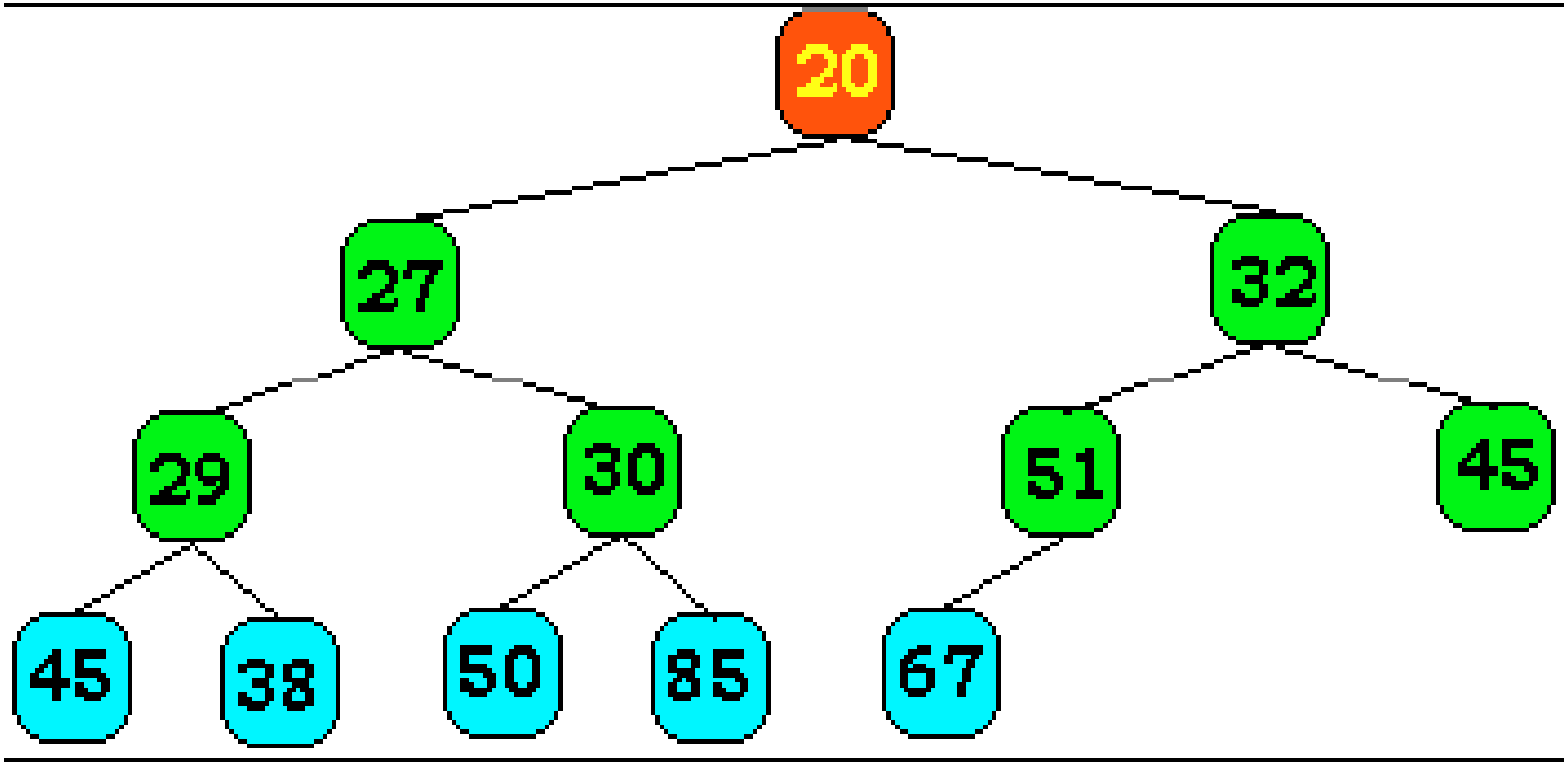


Exemple d'arbre binaire non parfait

Arbre binaire partiellement ordonné

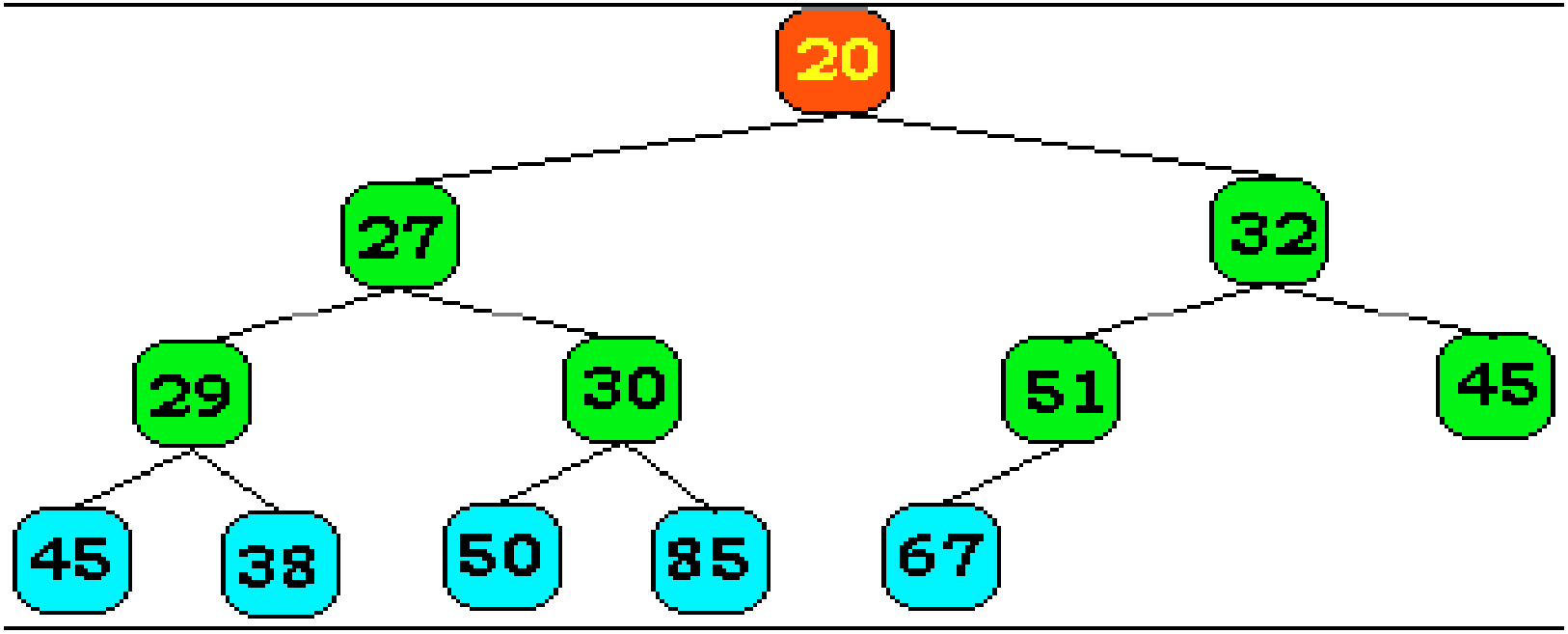
- *C'est un arbre étiqueté dont les nœuds appartiennent à un ensemble muni d'une relation d'ordre total (Entiers, réels etc …) tel que tout nœud donné a une valeur inférieure ou égale à celles de ses fils.*
- *Si on représente une liste ou un ensemble d'éléments par un tel arbre, on a, toujours, un élément minimum à la racine.*

- *Exemple d'un arbre partiellement ordonné sur l'ensemble {20, 27,29, 30, 32, 38, 45, 45, 50, 51, 67 ,85 } d'entiers naturels .*



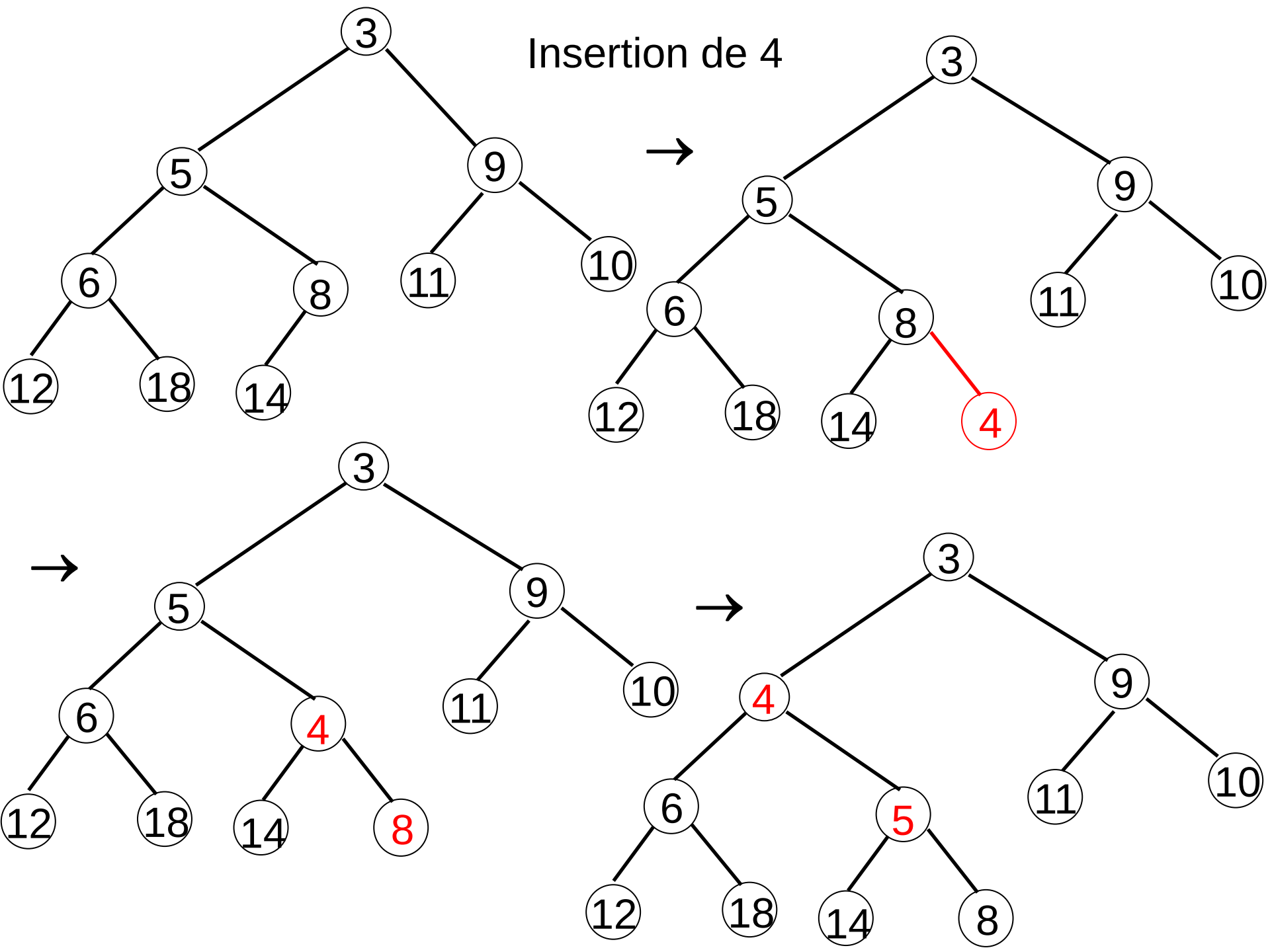
Un arbre binaire partiellement ordonné

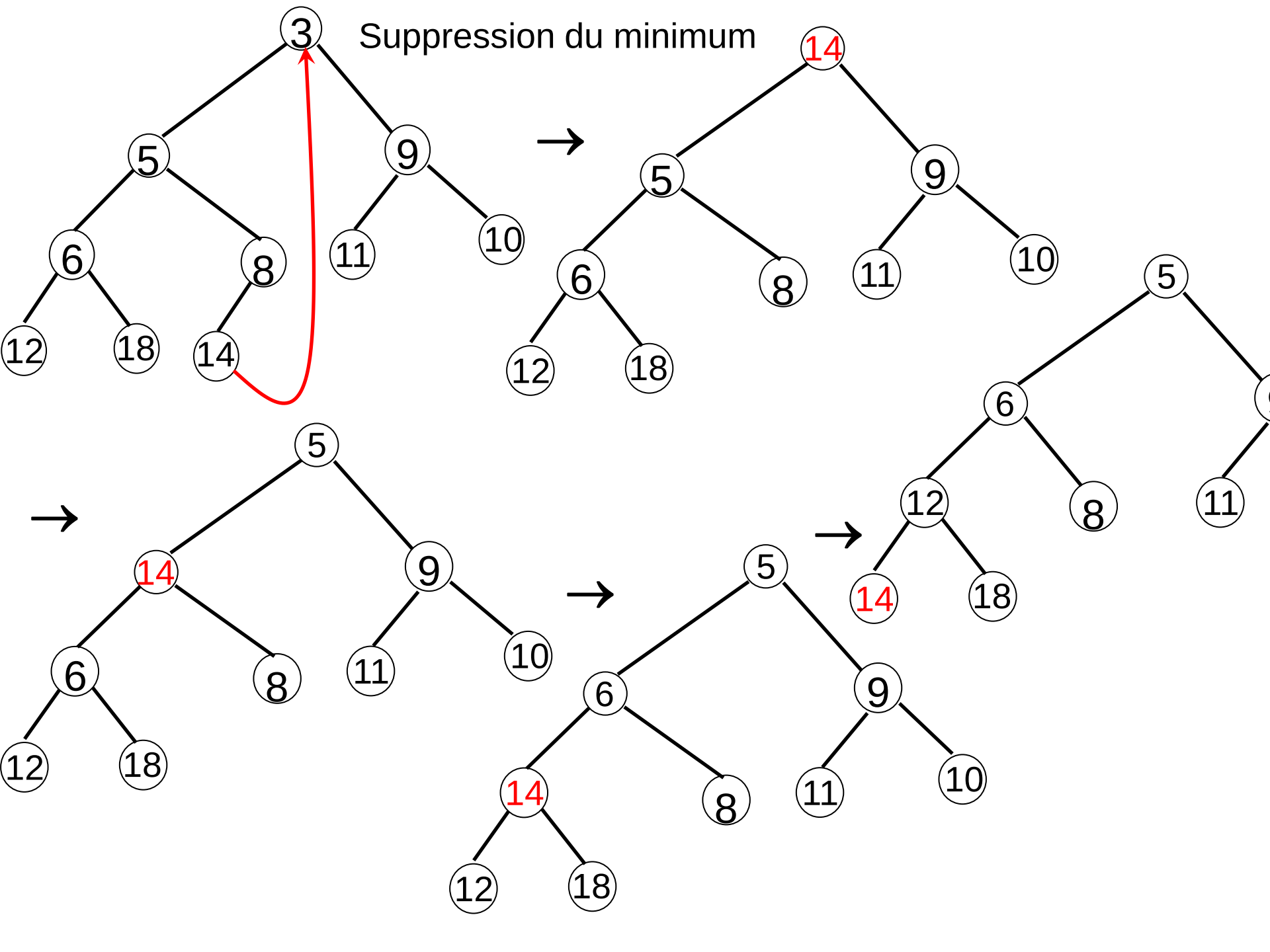
Voici un arbre partiellement ordonné



et voici le tas correspondant

1	2	3	4	5	6	7	8	9	10	11	12
20	27	32	29	30	51	45	45	38	50	85	67


Insertion de 4
3
5
9
6
8
11
10
12
18
14
4
3
5
9
6
4
11
10
12
18
14
8
3
4
9
6
5
11
10
12
18
14
8


Suppression du minimum
3
5
9
6
8
11
10
12
18
14
14
5
9
6
8
11
10
12
18
5
14
9
6
8
11
10
12
18
5
6
12
8
11
14
18
5
6
9
14
8
11
10
12
18

Suite au TD N° 04

Exercice 11 : Programmer l'insertion et la suppression dans un Tas