



## Aujourd' hui

---

- Chap. 1 suite : récursivité et application aux jeux
- Chap. 2 : arbres binaires
- Chap. 3 : le type union

1



## Récursivité

---

Application aux jeux

2



## Caractérisation des jeux

---

On considère uniquement les jeux

- à deux joueurs (ou plus)
- sans hasard ( $\neq$  dés)
- complètement informés ( $\neq$  cartes)

Exemples: tic-tac-toe, dames, othello, échecs, puissance 4, jeu de go...

3



## Exemple : le jeu de Grundy

---

Une pile de jetons est posée sur la table. Chaque joueur à son tour divise une pile en deux piles non égales. Le premier qui ne peut plus jouer a perdu.

4

## Représentation du jeu

Pour un nombre de pions donné, on peut représenter toutes les parties possibles à l'aide d'une arborescence.

Exemple : deux joueurs A et B jouent avec 7 pions

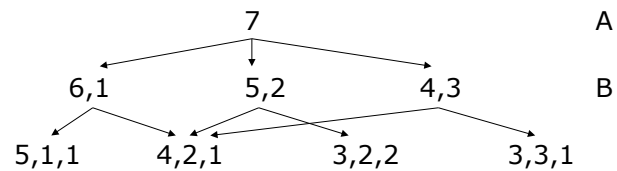
5

## Jeu de Grundy avec 7 pions



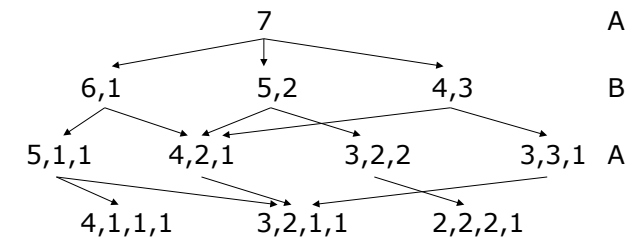
6

## Jeu de Grundy avec 7 pions



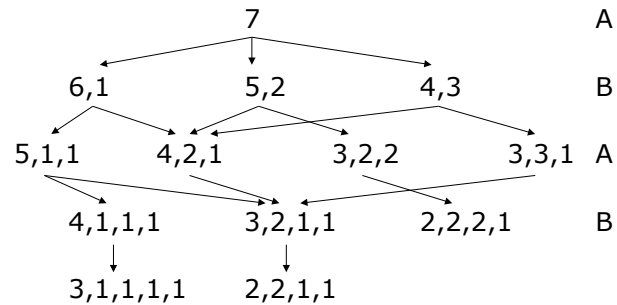
7

## Jeu de Grundy avec 7 pions



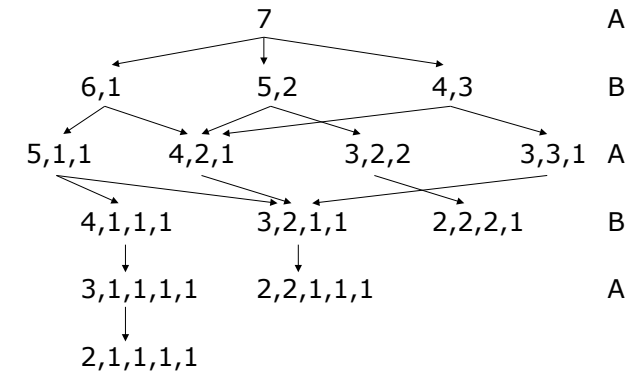
8

## Jeu de Grundy avec 7 pions



9

## Jeu de Grundy avec 7 pions



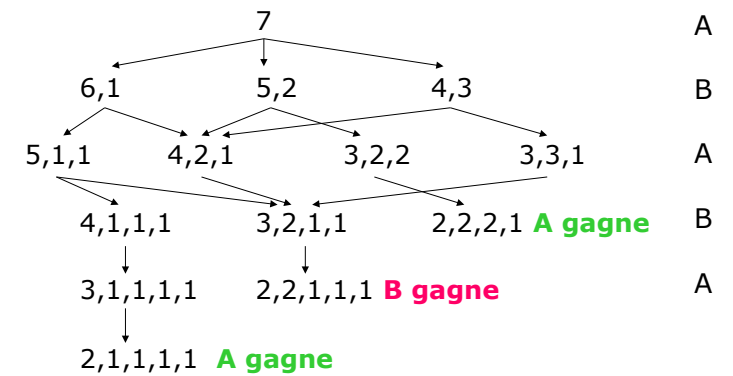
10

## Question

Que se passe-t-il si les deux joueurs jouent de manière optimale?

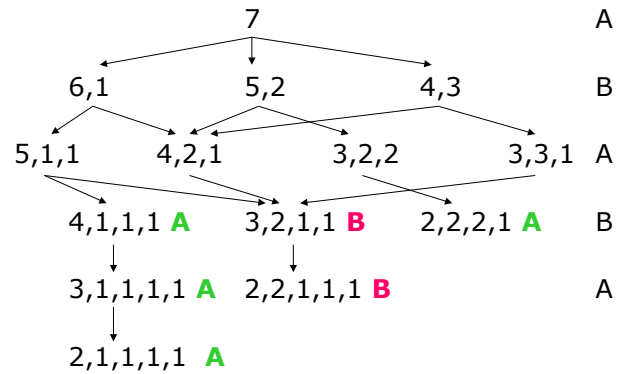
11

## Jeu de Grundy avec 7 pions



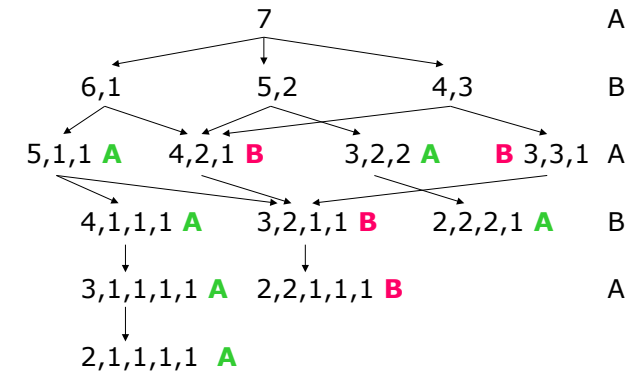
12

## Jeu de Grundy avec 7 pions



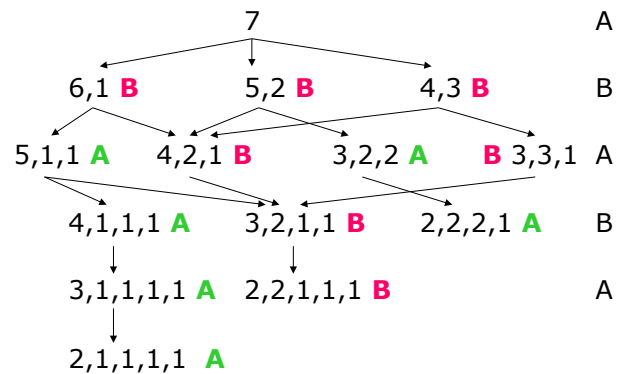
13

## Jeu de Grundy avec 7 pions



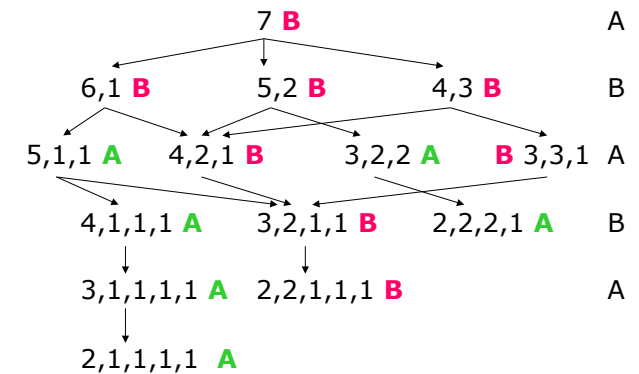
14

## Jeu de Grundy avec 7 pions



15

## Jeu de Grundy avec 7 pions



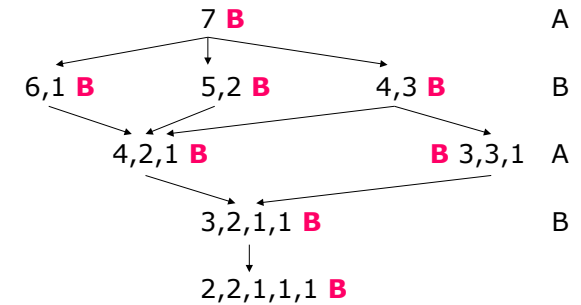
16

## Stratégie gagnante

- Une **stratégie gagnante** est une série de coups qui assure au joueur de gagner quelles que soient les répliques de l'adversaire.
- Il n'existe pas toujours de stratégie gagnante dans les jeux avec match nul (exemple: tic-tac-toe)
- S'il existe une stratégie gagnante pour un joueur, alors il n'en existe pas pour l'adversaire.

17

## Stratégie gagnante pour le second joueur avec 7 pions



18

## Recherche d'une stratégie gagnante

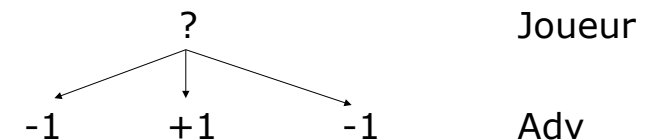
On se place du point de vue de l'un des deux joueurs et on recherche une stratégie qui permet de gagner à coup sûr.

Les fins de parties sont étiquetées +1 (gagné) ou -1 (perdu), puis on fait remonter ces valeurs jusqu'en haut

19

## Recherche d'une stratégie gagnante

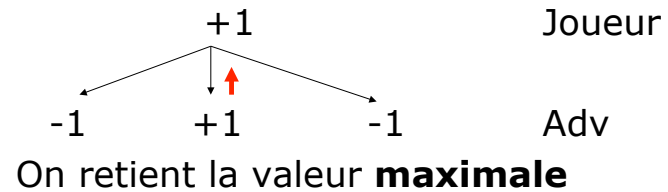
Tour du joueur : il suffit qu'un des coups mène à une victoire



20

## Recherche d'une stratégie gagnante

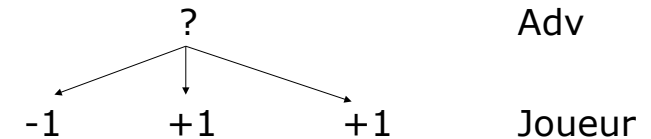
Tour du joueur : il suffit qu'un des coups mène à une victoire



21

## Recherche d'une stratégie gagnante

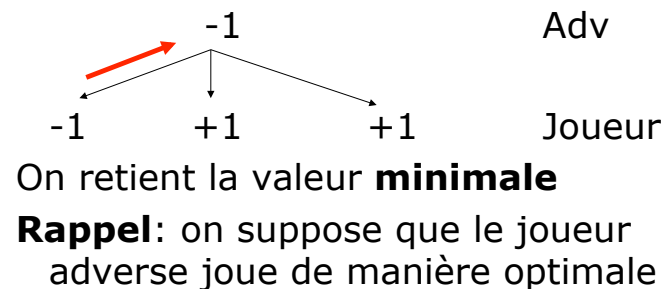
Tour de l'adversaire : il faut que tous les coups mènent à une victoire



22

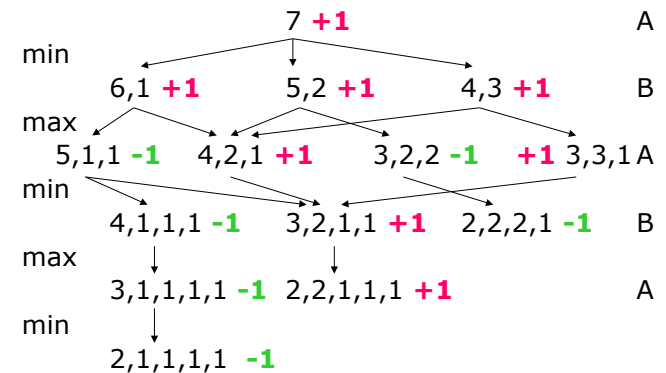
## Recherche d'une stratégie gagnante

Tour de l'adversaire : il faut que tous les coups mènent à une victoire



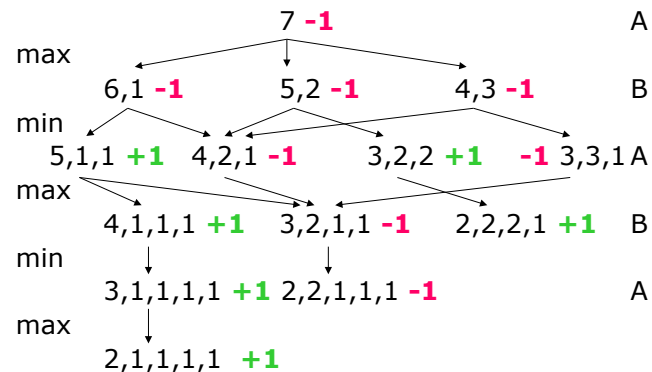
23

## Du point de vue de B



24

## Du point de vue de A



25

## Algorithme: méthode récursive

Plusieurs manière de faire

1) deux fonctions mutuellement récursives :

- la fonction **joueur** : valeur de la situation courante = *maximum* des coups possibles
- la fonction **adversaire** : valeur de la situation courante = *minimum* des coups possibles

2) Une seule fonction récursive

- La valeur retournée positive si **joueur** joue, négative si **adversaire** joue.
  - Exploite la profondeur de l'arbre (paire/impair)

26

## Algorithme Min-Max

fonction **joueur** (état courant)

s'il existe un coup gagnant alors  
délivrer (+1)

sinon

pour tous les coups possibles

$C_i = \text{adversaire}(\text{nouvel\_état})$

délivrer le **maximum** des  $C_i$

27

## Algorithme Min-Max

fonction **adversaire** (état courant)

s'il existe un coup gagnant alors  
délivrer (-1)

sinon

pour tous les coups possibles

$C_i = \text{joueur}(\text{nouvel\_état})$

délivrer le **minimum** des  $C_i$

28

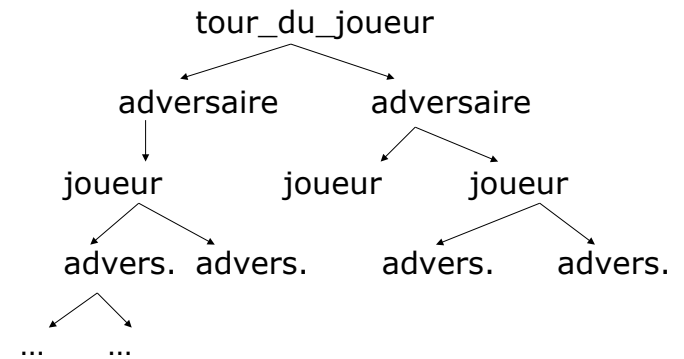
## Algorithme Min-Max

fonction **tour\_du\_joueur** (état courant)

```
s'il existe un coup gagnant alors
  jouer ce coup gagnant
sinon
  pour tous les coups possibles
     $C_i = \mathbf{adversaire}(\text{nouvel\_état})$ 
    s'il existe un  $C_i = +1$  alors
      jouer le coup correspondant
    sinon jouer un coup au hasard
```

29

## Déroulement -> récursivité !



30

## Application

Programmation du jeu de la « course à vingt » en TP, où l'utilisateur joue contre le programme, avec deux versions:

- le programme joue les coups au hasard
- le programme calcule à chaque tour de jeu un coup gagnant (s'il existe)

31

## Limitation de la recherche

- On ne peut pas toujours explorer l'ensemble des coups possibles
  - Échecs : ~20 coups possibles
  - Dames : ~10
  - Jeu de go : ~200
  - complexité exponentielle
- On étudie seulement les  $n$  prochains coups à jouer
  - limitation de la profondeur de l'arbre

32



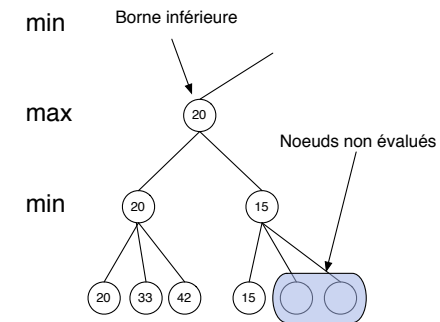
## Fonction d'évaluation

- Définition d'une fonction qui attribue à chaque configuration du jeu une valeur indiquant si elle est favorable au joueur
- Ex : othello
  - Différence de nombre de pions
  - Bonus pour la prise d'un coin
  - Bonus pour la prise d'un bord
  - Nombre de coups possibles restants

33

## Élagage alpha-beta

- Il n'est pas toujours nécessaire d'évaluer l'arbre entier



34

## Élagage alpha-beta

- Alpha : borne inférieure d'une valeur à maximiser
  - On peut ignorer les nœuds ayant une valeur inférieure
- Beta : borne supérieure d'une valeur à minimiser
  - On peut ignorer les nœuds ayant une valeur supérieure

35

## Chapitre 2

### Arbres et **Arbres binaires**

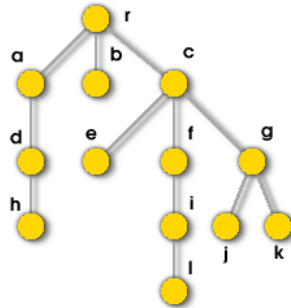
37

## 1) Arbre : définition générale

Un arbre est une hiérarchie formée

- de **nœuds** (a, b...)
- d'**arcs** reliant les nœuds

Un nœud particulier:  
la **racine** (ici, r)



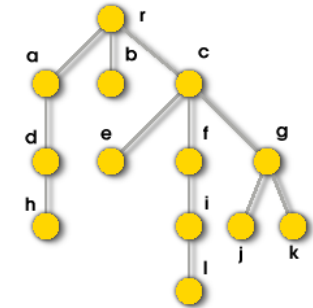
38

## 1) Définitions et exemples

Chaque nœud possède 0, 1 ou n **fil**s et donc autant de sous-arbres

Exemple

- a possède un fils
- b ne possède aucun fils
- c possède trois fils

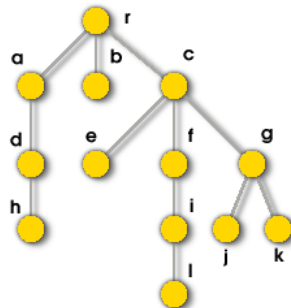


39

## 1) Définitions et exemples

Un nœud sans fils est appelé une **feuille**

Cet arbre possède six feuilles: h, b, e, l, j, et k



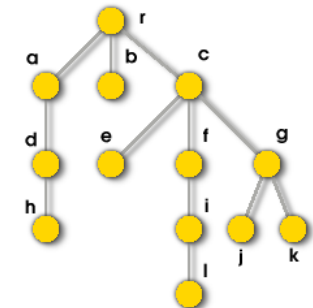
40

## 1) Définitions et exemples

Chaque nœud, sauf la racine, possède exactement un **père**

Exemple

- a est le père de d
- g est le père de j et de k



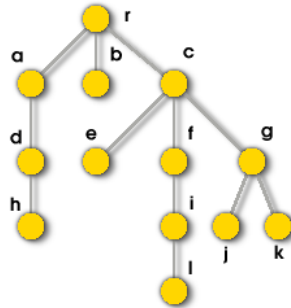
41

## 1) Définitions et exemples

Un **chemin** est une suite d'arcs qui relie deux nœuds

Exemple

- chemin r-a-d-h
- chemin c-f-i
- chemin f-c-g



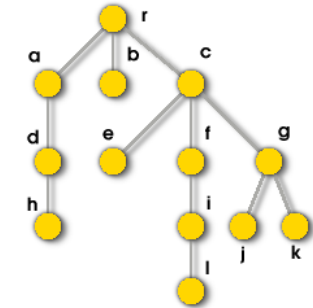
42

## 1) Définitions et exemples

Il existe un chemin allant de la racine à tout nœud n. Les nœuds de ce chemin sont les **ancêtres** de n

Exemple

- ancêtres de h : r, a, d
- ancêtres de g : r, c



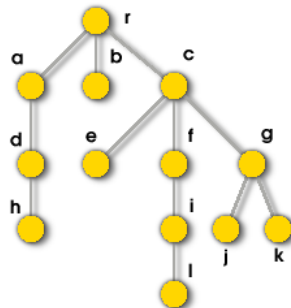
43

## 1) Définitions et exemples

**profondeur** d'un nœud : longueur du chemin entre la racine et ce nœud

Exemple

- profondeur de h : 3
- profondeur de g : 2



44

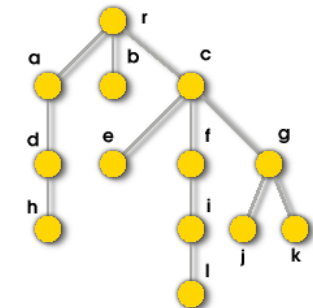
## 1) Définitions et exemples

**Hauteur** d'un nœud : taille du plus long chemin de ce nœud à une feuille située sous lui

**Hauteur** d'un arbre : hauteur de sa racine

Exemple

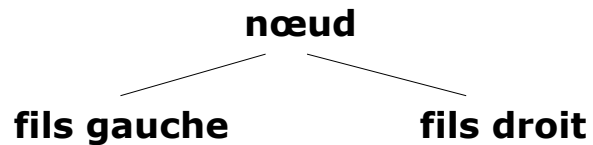
- hauteur de a : 2
- hauteur de c : 3
- hauteur de l'arbre : 4



45

## 1) Définitions et exemples

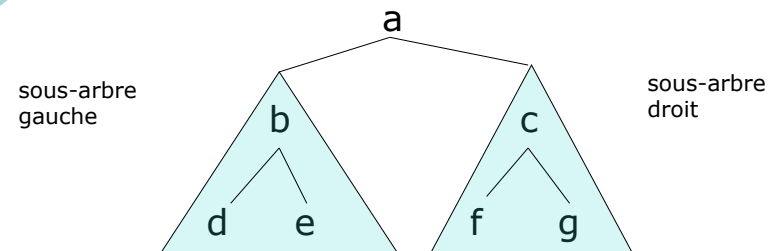
Un **arbre binaire** est un arbre dont tout nœud possède au plus deux fils. Ces fils sont ordonnés : on parlera du **fils gauche** et du **fils droit**



46

## 1) Définitions et exemples

Le sous-arbre qui a pour racine le fils d'un nœud n sera appelé **sous-arbre** de ce nœud.



47

## 2) Parcours standards d'arbres

Il existe plusieurs manières de parcourir tous les nœuds d'un arbre binaire.

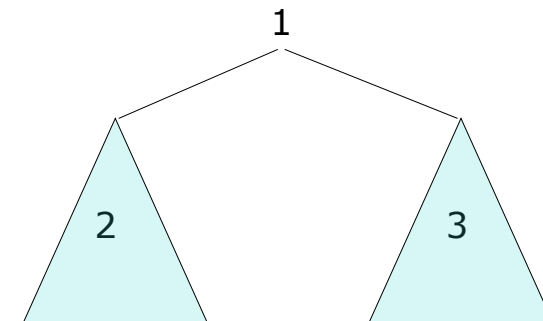
Les parcours standards sont :

- Parcours en profondeur
  - Préfixe ou DGD (descendant gauche droite)
  - Postfixe ou AGD (ascendant gauche droite)
  - Infixe ou symétrique
  - Définition récursive naturelle et simple à mettre en œuvre
- Parcours en largeur
  - Non récursif, faisant intervenir une pile

48

## Parcours préfixe ou DGD

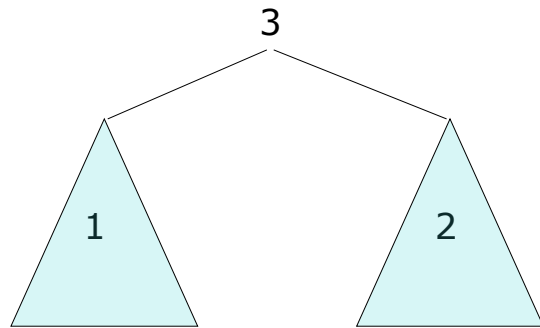
On étudie le nœud courant puis ses sous-arbres



49

## Parcours postfixe ou AGD

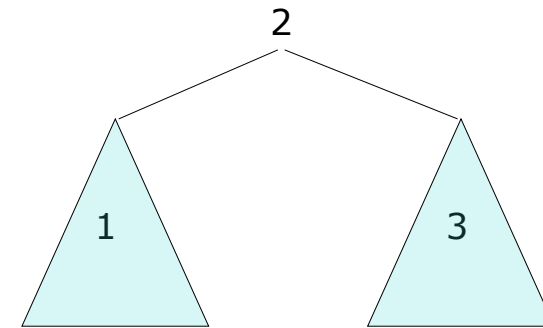
On étudie les sous-arbres puis le nœud courant



50

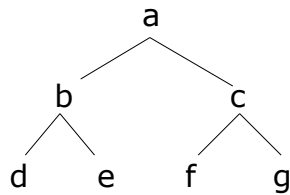
## Parcours infixe ou symétrique

On étudie le nœud courant entre l'étude des sous-arbres



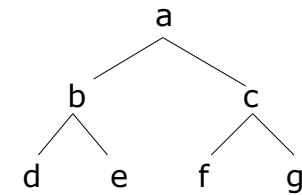
51

## Exemple



52

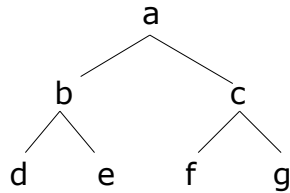
## Exemple



Parcours préfixe : a b d e c f g

53

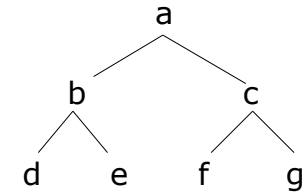
## Exemple



Parcours préfixe : a b d e c f g  
Parcours postfixe : d e b f g c a

54

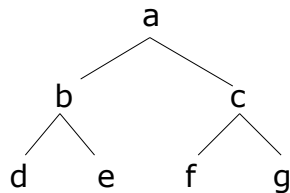
## Exemple



Parcours préfixe : a b d e c f g  
Parcours postfixe : d e b f g c a  
Parcours infixe : d b e a f c g

55

## Exemple



Parcours préfixe : a b d e c f g  
Parcours postfixe : d e b f g c a  
Parcours infixe : d b e a f c g  
Parcours en largeur : a b c d e f g

56

## 3) Spécifications d'un arbre binaire

- Structure récursive par essence
  - caractérisé par sa **racine**
  - un sous-arbre gauche = un arbre caractérisé par sa **racine**
  - Un sous-arbre droit = un arbre caractérisé par sa **racine**
- Sémantiquement : arbre != nœud
- Implémentation :
  - arbre => racine == nœud

57



## Primitives de manipulation d'un arbre binaire (d'entiers)

---

- a. Primitives relatives à la **structure**
  - a. Ajout/modification/suppression
  - b. Test
- b. Primitives relatives aux **valeurs utiles**
  - a. Modification/affichage

58



## Primitives de **structure**

---

- o Modification
- ```
t_arbre creer_arbre(<type> val, t_arbre sag,  
t_arbre sad)  
int supprimer_arbre(t_arbre*) /* supprime l'arbre,  
    met à jour le pere si nécessaire */  
t_arbre ajout_gauche(t_arbre a, int v); /* crée un  
    SAG de valeur v*/  
t_arbre ajout_droit(t_arbre a, int v); /* crée un  
    SAD de valeur v*/
```

59



## Primitives d'accès et de test

---

```
int arbre_vide(t_arbre)  
/* délivre vrai si l'arbre est vide, faux sinon */  
o un arbre vide est caractérisé par la constante NULL  
t_arbre pere(t_arbre)  
/* renvoie le pere s'il existe, NULL sinon */  
t_arbre sag(t_arbre)  
/* renvoie le fils gauche s'il existe, NULL sinon */  
t_arbre sad(t_arbre)  
/* renvoie le fils droit de l'arbre s'il existe, NULL sinon */  
int est_feuille(t_arbre)  
/* renvoie vrai si l'arbre est une feuille, faux sinon */  
o Note: un nœud interne est donc un nœud qui n'est pas  
    une feuille  
int est_racine(t_arbre) /* renvoie vrai si l'arbre est  
    une racine, faux sinon */
```

60




## Primitives de parcours

---

```
void parcours_infixe(t_arbre, void (*fonc)(int*))  
/* effectue un parcours infixe, applique fonc à chaque  
    nœud */  
  
void parcours_prefixe(t_arbre, void (*fonc)(int*))  
/* effectue un parcours prefixe, applique fonc à chaque  
    nœud */  
  
void parcours_postfixe(t_arbre, void (*fonc)(int*))  
/* effectue un parcours postfixe, applique fonc à chaque  
    nœud */
```

Note: on peut en dériver l'affichage de l'arbre !

61



### c) Primitives de consultation et de modification du nœud courant

---

```
int val_racine(t_arbre a, int* v)
```

```
/* v prend la valeur de la racine si l'arbre n'est pas vide, renvoie vrai si v est exploitable */
```

```
int modif_racine(t_arbre a, int v)
```

```
/* la racine de l'arbre prend la valeur v si l'arbre n'est pas vide, renvoie vrai si modification effectuée */
```

62



### Ex : afficher les ancêtres d'un noeud

---

```
void afficher_ancetres(t_arbre a, char end){  
    a = pere(a);  
    while(!arbre_vide(a)){  
        afficher_val(a, ' ');  
        a = pere(a);  
    }  
    printf("%c", end);  
}
```

64



### 4) Mise en œuvre des arbres binaires

---

- Mise en œuvre par pointeurs
- Mise en œuvre par tableau
  - par calcul
  - par indigage

☞ à faire en TD/TP

73



## Chapitre 3

---

Le type union

74





## 1) Définition et exemples

---

- Le type **structure** permet de regrouper plusieurs objets sous un même nom
- Le type **union** permet d'interpréter de différentes manières une même zone mémoire : le même emplacement mémoire pourra contenir, selon les cas, un entier, un réel, une chaîne, etc.

75



## Définition

---

### Syntaxe

```
typedef union {champ1; ... champn;}  
id_type;
```

où chaque champ est de la forme  
*type ident*

76



## Exemple

---

Un `t_union` est soit un entier, soit un caractère, soit un réel

```
typedef union { int entier ; char lettre ;  
                float reel ; } t_union ;
```

Déclaration de variables `t_union`

```
t_union variable1, variable2;
```

77



## Utilisation

---

Pour manipuler une variable de type union, on indique le nom de la variable et le nom du champ utilisé, reliés par l'opérateur '.'

`variable1.entier` est un entier  
`variable1.lettre` est un caractère  
`variable1.reel` est un réel

78



## Exemple 1

---

variable1  variable2

variable1.entier = 12 ;  
variable2.lettre = 'a' ;

79



## Exemple 1

---

variable1  variable2

variable1.entier = 12 ;  
variable2.lettre = 'a' ;

variable1  variable2

80



## Exemple 2

---

variable1

variable1.entier = 12 ;

81



## Exemple 2

---

variable1

variable1.entier = 12 ;

variable1

82



## Exemple 2

---

variable1

variable1.entier = 12 ;

variable1

variable1.reel = 8.75 ;

83



## Exemple 2

---

variable1

variable1.entier = 12 ;

variable1

variable1.reel = 8.75 ;

variable1

84



## 2) Applications

---

Le type union peut être utilisé notamment :

- dans les types abstraits (piles, files, listes, arbres)
  - ex: une liste d'entiers et de caractères
- dans les types structures, lorsque l'information varie selon l'objet représenté

85



## Exemple

---

Les documents d'une bibliothèque (livres, mensuels) sont caractérisés par un titre et une année de parution.

Les livres sont caractérisés par un nom d'auteur (une chaîne de caractère) tandis que les revues sont caractérisées par le mois de leur parution (un entier)

86

## Représentation d'un document



87

## Définition d'un document

*Info document: soit l'auteur, soit le mois*

```
typedef union {  
    char auteur[20] ;  
    int mois ; } t_info_doc ;
```

*Document = titre + année + info document*

```
typedef struct {  
    char titre[40] ;  
    int annee ;  
    t_info_doc info ; } t_doc ;
```

88

## Problème

On déclare :  
t\_doc mon\_doc;

On veut afficher le contenu de  
mon\_doc

Comment savoir si mon\_doc est un  
livre ou une revue ?

89

## Solution

On ajoute à la structure t\_doc un champ  
indiquant la nature du document, par  
exemple un booléen *livre* qui vaut vrai si  
le document est un livre, faux si c'est une  
revue

```
typedef struct {  
    char titre[40] ;  
    int annee ;  
    int livre ;  
    t_info_doc info ; } t_doc ;
```

90



## Initialisation d'un document

---

```
t_doc mon_doc = {  
    « Guerre et paix »,  
    1865,  
    vrai,  
    « Léon Tolstoï »}
```

(ou bien par affectation de chaque champ)



## Affichage d'un document

---

```
void affiche_doc(t_document doc)  
{  
    printf( 'Titre %s', doc.titre);  
    printf( 'Annee %i', doc.annee);  
    if (doc.livre)  
        printf( 'Auteur %s', doc.info.auteur)  
    else /* c'est une revue */  
        printf( 'Mois %i', doc.info.mois) ;  
}
```