Informatique au lycée, 2018-2019

Arbres

Gilles Subrenat[1]
Sylvie Alayrangues[1]

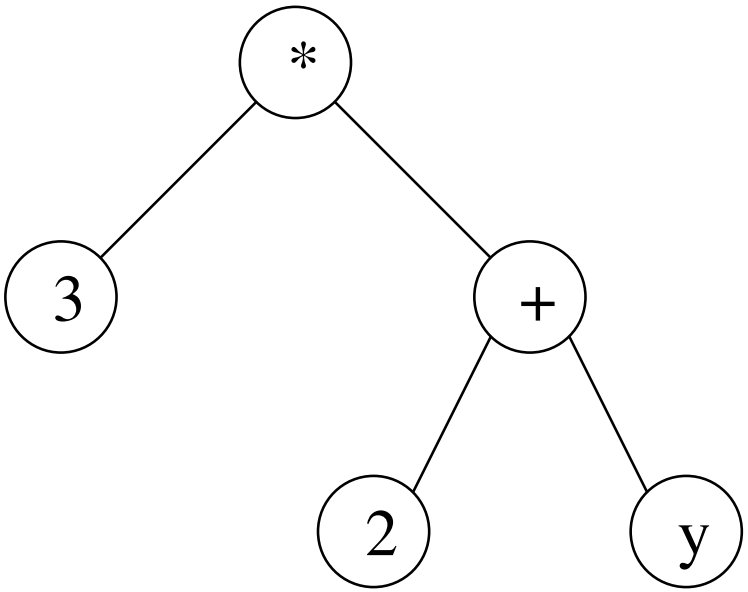
[1]Laboratoire XLIM-SIC
Gilles.Subrenat@univ-poitiers.fr
Sylvie.Alayrangues@univ-poitiers.fr

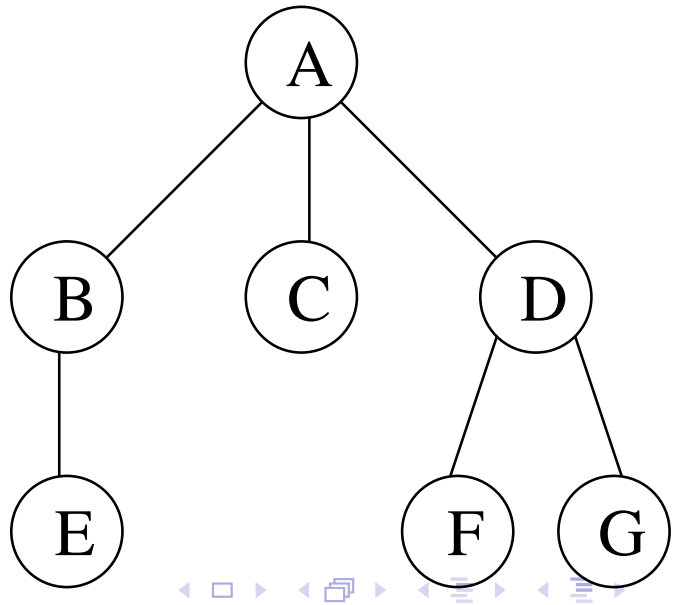
Journées de formation 2018-2019, 27 mars 2019

Plan → A - Arbres

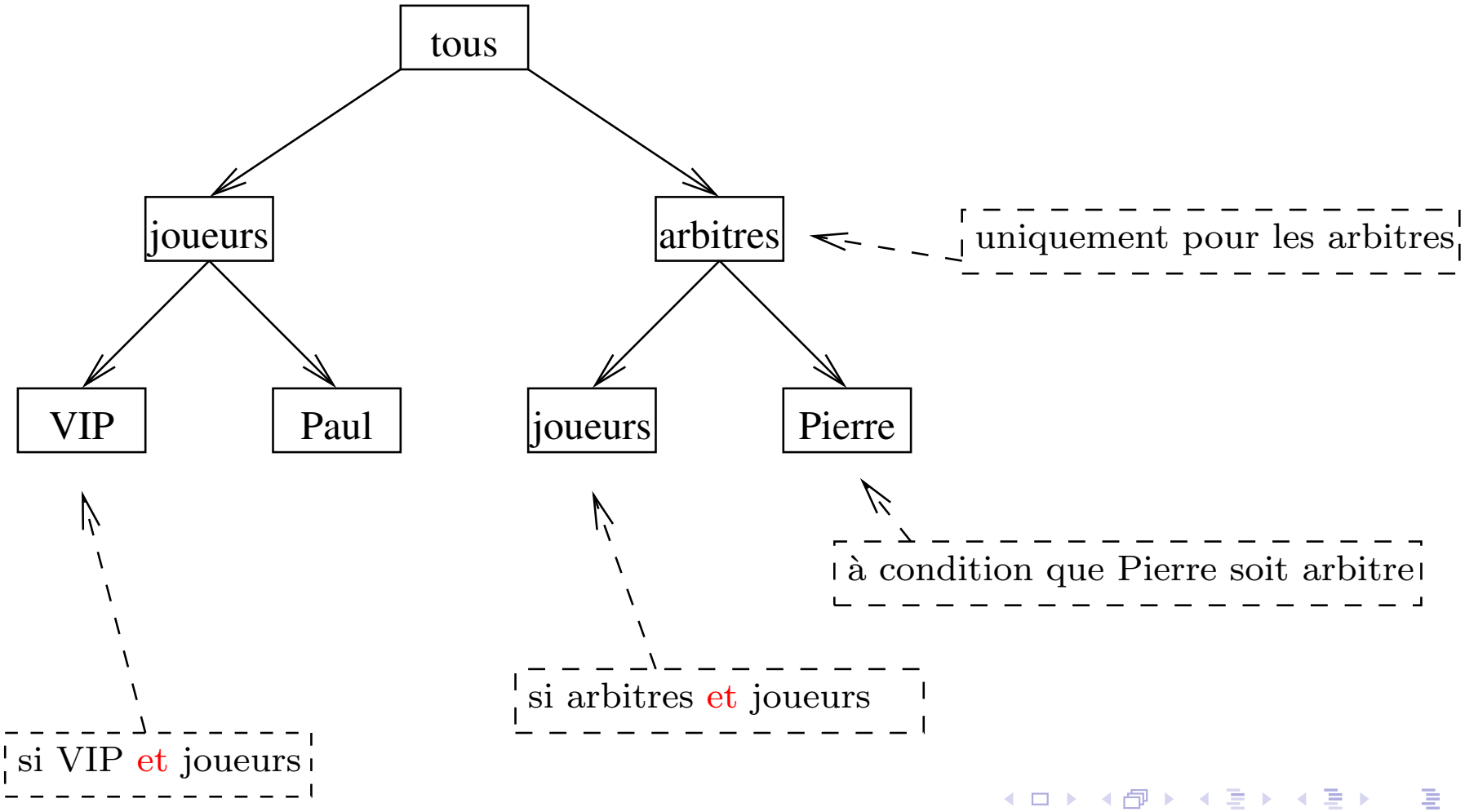
1. Généralités
   - Présentation
   - Exemples
2. Définitions
3. Représentation mémoire
4. Min/Max, élagage α/β

- Structure arborescente/ hiérarchique
- Notion de père/fils et ancêtre/descendant
- Exemples
  - généalogie
  - hiérarchie de droits d'accès
  - pages HTML
  - ...





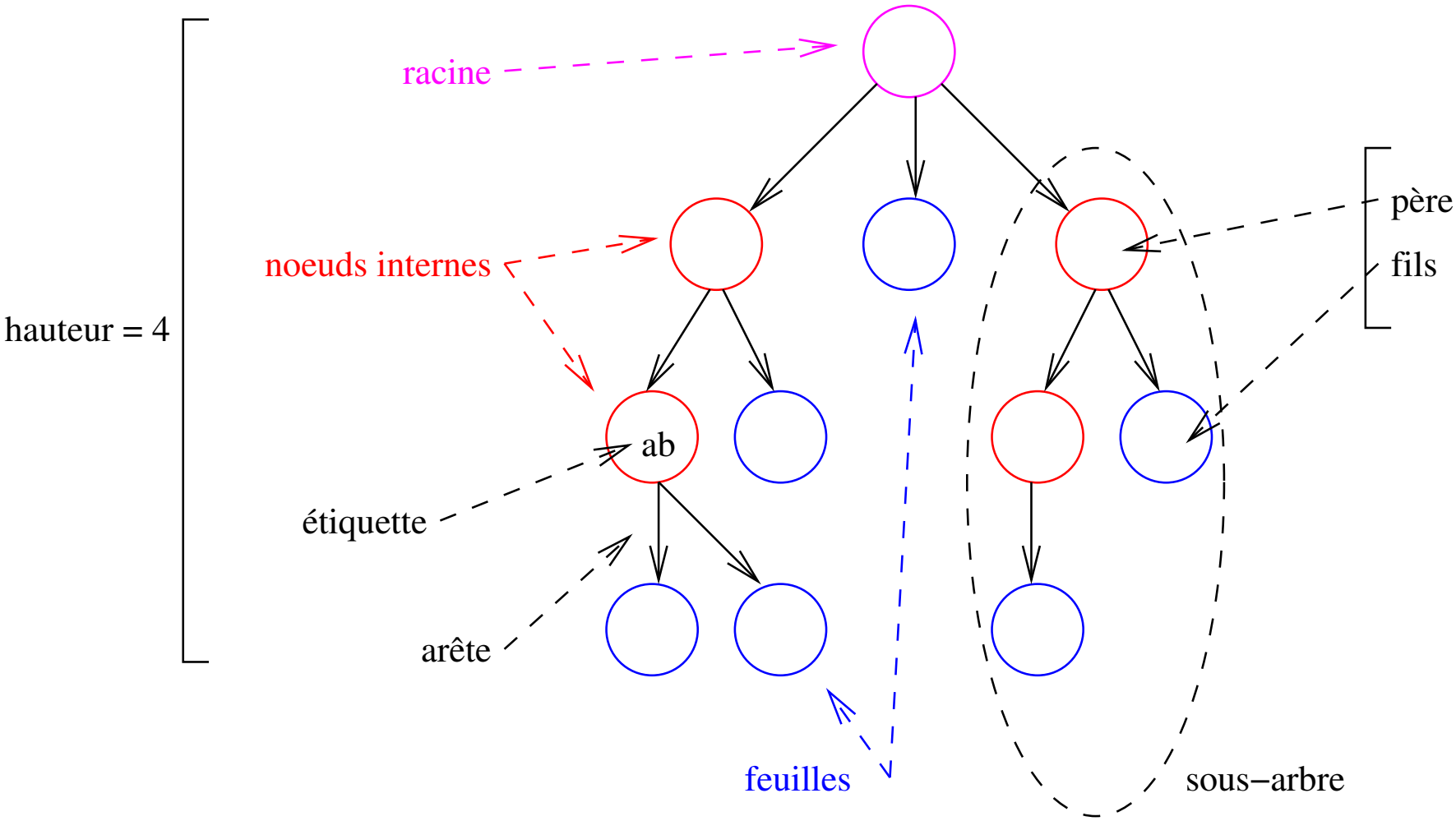
- ACL (Access Control List)



Plan → A - Arbres

1. Généralités
2. Définitions
   - Définitions “formelles”
   - Propriétés des arbres binaires
   - Algorithmes
   - Méthodes
   - Arbre “particuliers
3. Représentation mémoire
4. Min/Max, élagage α/β

- Arbre
  - couple (S,A) de sommets et d'arêtes
  - à un sommet est associée une étiquette (valeur)
  - à un sommet (dit père) sont associés zéro, un ou plusieurs sommets (dits fils)
  - la racine est le seul sommet à ne pas avoir de père
    $\Rightarrow$ tout sommet (sauf la racine) a un et un seul père
- noeud interne : sommet avec au moins un fils
- feuille : sommet sans fils
- sous-arbre
  - un fils et ses descendants forment un sous-arbre rattaché au père
  - un sous-arbre est un arbre (définition récursive)
- hauteur : nombre d'étages dans l'arbre
  - arbre vide : hauteur égale à 0
  - arbre réduit à un seul noeud : hauteur égale à 1
- branche : suite contiguë de sommets entre la racine et une feuille

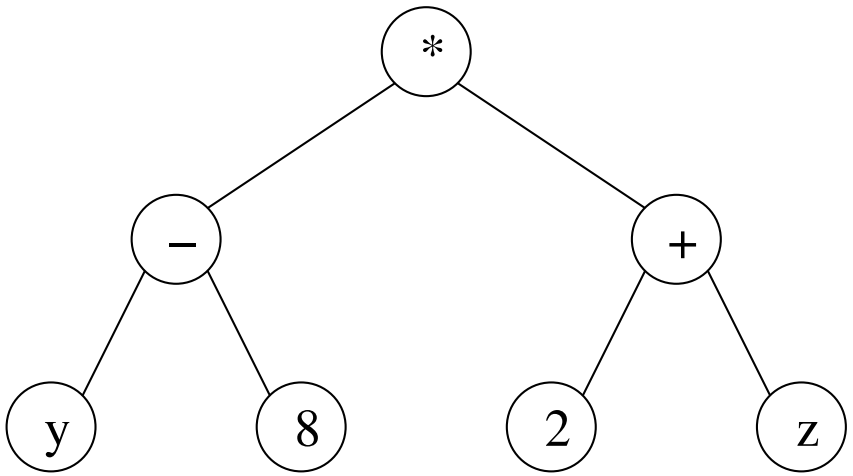

racine
noeuds internes
hauteur = 4
ab
étiquette
arête
père
fils
feuilles
sous−arbre

- Arbre binaire : chaque noeud a deux fils au maximum
- Arbre binaire complet : chaque noeud a zéro ou deux fils
- Arbre binaire équilibré
  - arbre de hauteur minimale en fonction du nombre de noeuds
  - seul le dernier niveau est (éventuellement) incomplet
  - hauteur : $h \approx \log_2(n)$
  - le dernier niveau peut être "tassé" sur la gauche
- Arbre binaire équilibré (autre conception) :
  - pour tout sommet s

    $$\textit{taille(sous-arbre gauche)} = \textit{taille(sous-arbre droit)}$$
    $$\text{ou } \textit{taille(sous-arbre gauche)} = \textit{taille(sous-arbre droit)} + 1$$
- Arbre binaire parfait :
  - définition : arbre équilibré dont le dernier niveau est complet
  - nombre noeuds : $2^h - 1$ (h = hauteur)
  - nombre feuilles : 2^{h-1}
  - hauteur : $\log_2(n+1)$

- Algorithme récursif

```
1  procedure parcours(A : Arbre)
2  {
3     si (A non vide)
4     {
5        //traitement prefixe : afficher étiquette par exemple
6        parcours(a.fils_gauche)
7        //traitement infixe (idem)
8        parcours(a.fils_droit)
9        //traitement postfixe (idem)
10    }
11 }
```



préfixe : * − y 8 + 2 z

infixe : y − 8 * 2 + z

postfixe : y 8 − 2 z + *

- Algorithmes récursifs

```
1  function nbr_noeuds(A : Arbre) : entier
2  {
3     if (A est vide)
4        return 0
5     else
6        return 1 + nbr_noeuds(A.fils_gauche)
7                 + nbr_noeuds(A.fils_droit)
8     }
9  }
```

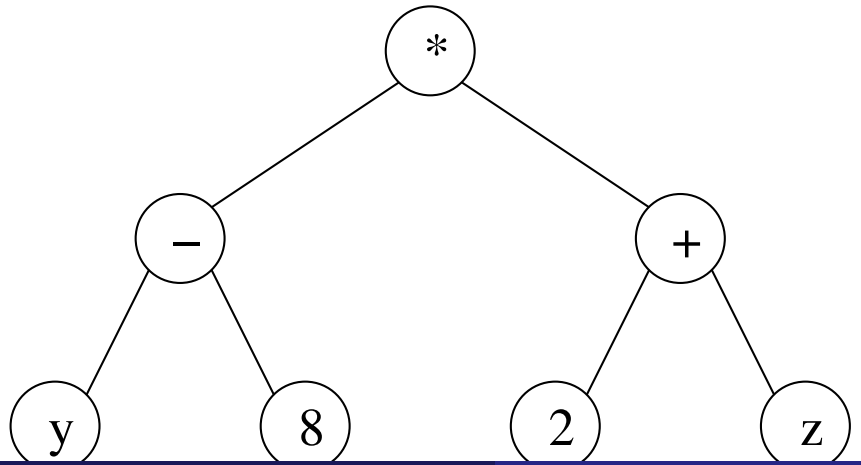
```
1  function nbr_feuilles(A : Arbre) : entier
2  {
3     if (A est vide)
4        return 0
5     elseif (A est une feuille)
6        return 1
7     else
8        return  nbr_feuilles(A.fils_gauche)
9              + nbr_feuilles(A.fils_droit)
10    }
11 }
```

```
1  function hauteur(A : Arbre) : entier
2  {
3     if (A est vide)
4        return 0
5     else
6        return 1 + max(hauteur(A.fils_gauche),
7                       hauteur(A.fils_droit))
8     }
9  }
```

```
1  function somme(A : Arbre) : entier
2  {
3     if (A est vide)
4        return 0
5     else
6        return a.val + somme(A.fils_gauche)
7                     + somme(A.fils_droit)
8     }
9  }
```

- Algorithme itératif

```
1  procedure parcours(A : Arbre)
2  {
3     F : file de Arbre
4     enfiler(F, A)
5     while (not vide(F))
6     {
7        sommet = defiler(F)
8        traiter(sommet)
9        if (not vide(sommet.fils_gauche))
10          enfiler(F, sommet.fils_gauche)
11       if (not vide(sommet.fils_droit))
12          enfiler(F, sommet.fils_droit)
13    }
14 }
```



largeur : * − + y 8 2 z

- Méthodes minimales (approche fonctionnelle)

```
1  //constructeurs
2  creer_vide() : Arbre
3  creer(v : Value, fg : Arbre, fd : Arbre)
4
5  // accesseurs
6  get_val(A : Arbre) : Value
7  get_fg(A : Arbre) : Arbre
8  get_fd(A : Arbre) : Arbre
9
10 // vide ?
11 est_vide(A : Arbre) : booleen
```

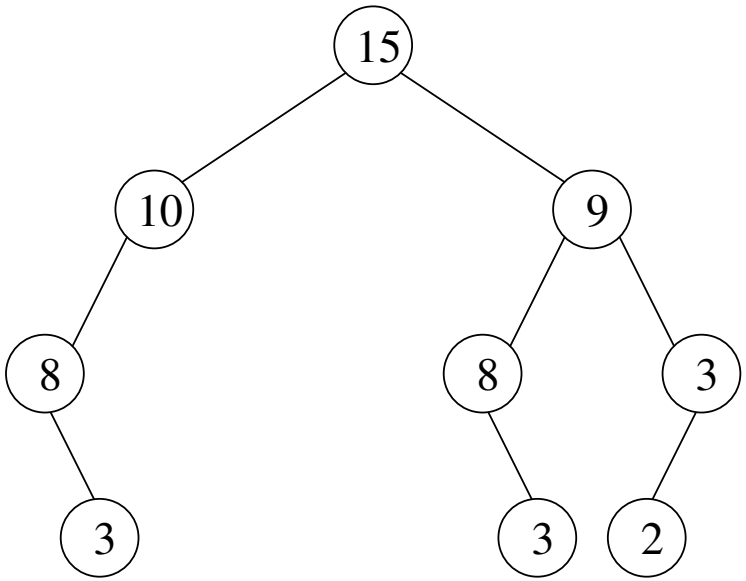
- Méthodes mimales (approche non fonctionnelle)

```
1  // libération mémoire de la racine, pas des sous-arbres
2  detruire_racine(A : Arbre);
3
4  // mutateurs
5  set_val((A : Arbre, v : Value)
6  set_fg(A : Arbre, fg : Arbre) // ancien fg non détruit
7  set_fd(A : Arbre, fd : Arbre) // ancien fd non détruit
```

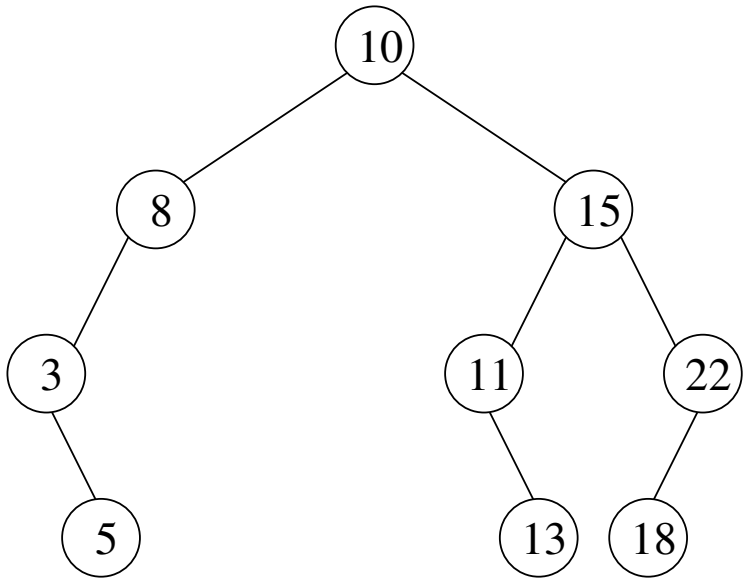
- Destruction complète

```
1  procedure detruire(A : Arbre)
2  {
3     if (not est_vide(A))
4     {
5        detruire(get_fg(A))
6        detruire(get_fd(A))
7        detruire_racine(A)
8     }
9  }
```

- Arbre “maximum” (file à priorité)
  - (la valeur d’)un noeud est supérieure à celles de ses fils
    $\rightarrow$ et donc à celles de ses descendants
- Arbre binaire de recherche (ABR)
  - un noeud est supérieur à ses descendants à gauche
  - un noeud est inférieur à ses descendants à droite



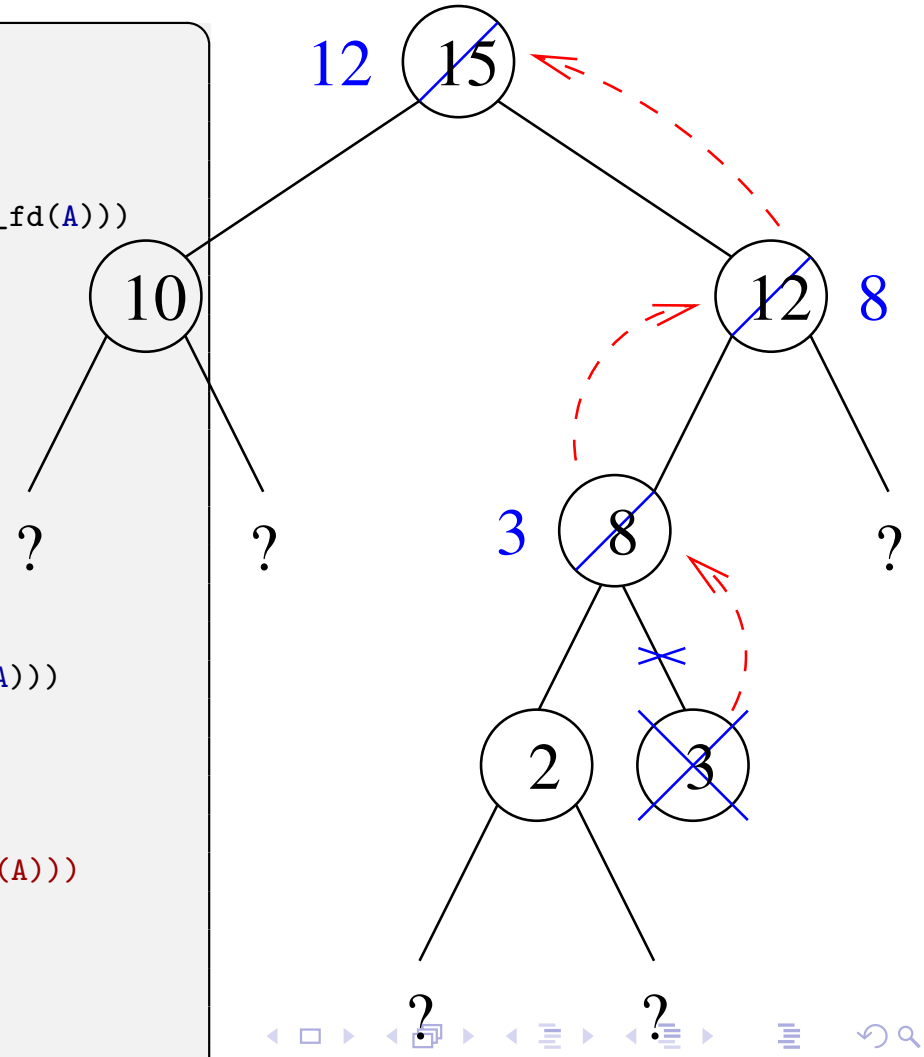
arbre "maximum"



Arbre binaire de recherche

- Arbre max : suppression racine

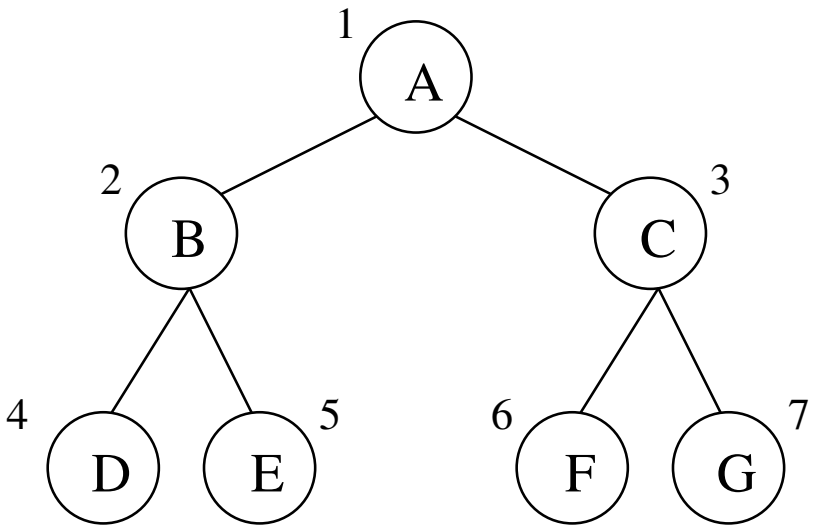
```
1  procedure suppr_max(A : Arbre)
2  {
3     if (est_vide(A))
4        erreur
5     elseif (est_vide(get_fg(A)) and est_vide(get_fd(A)))
6        detruire_racine(A)
7     elseif (est_vide(get_fd(A)))
8     {
9        set_val(A, get_val(get_fg(A)))
10       suppr_max(get_fg(A))
11    }
12    elseif (est_vide(get_fg(A)))
13    {
14       set_val(A, get_val(get_ff(A)))
15       suppr_max(get_fd(A))
16    }
17    elseif (get_val(get_fg(A)) > get_val(get_fd(A)))
18    {
19       set_val(A, get_val(get_fg(A)))
20       suppr_max(get_fg(A))
21    }
22    else // (get_val(get_fd(A)) > get_val(get_fg(A)))
23    {
24       set_val(A, get_val(get_fd(A)))
25       suppr_max(get_fd(A))
26    }
27 }
```



Plan → A - Arbres

1. Généralités
2. Définitions
3. Représentation mémoire
   - Arbre parfait
   - Arbre binaire et pointeurs/références
   - Arbre binaire et tableau
   - Arbre n-aire et liste de fils
   - Arbre n-aire et fils gauche/frère droit
4. Min/Max, élagage α/β

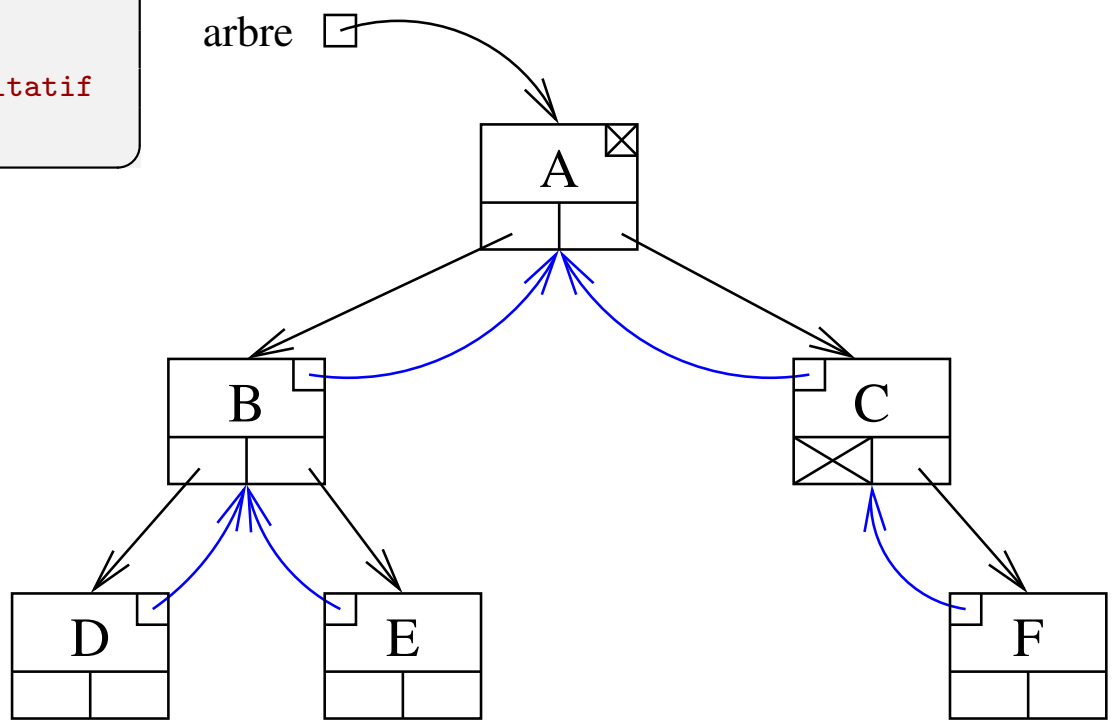
- tableau indicé de 1 à n
- un sommet par case
- racine : case 1
- fils gauche de la case i : $2 * i$
- fils droit de la case i : $2 * i + 1$
- père de la case i : i *div* 2



1	A
2	B
3	C
4	D
5	E
6	F
7	G

- Type

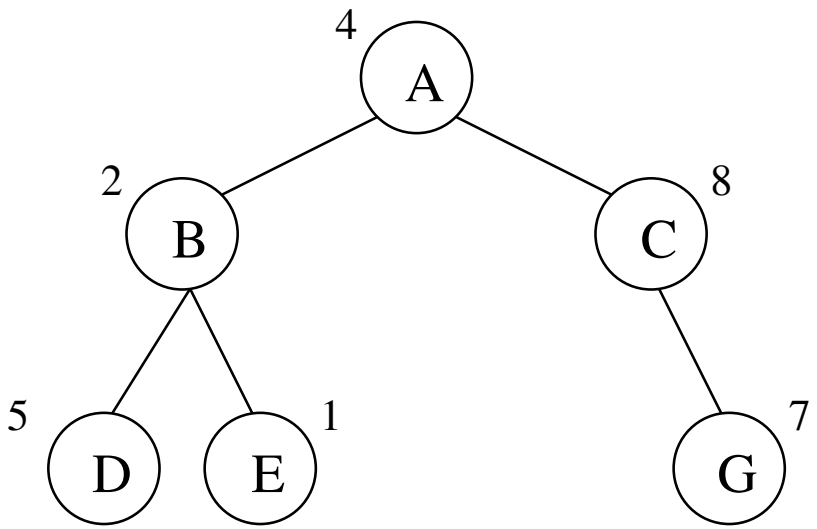
```
1  type Arbre = référence sur Sommet
2
3  type Sommet = structure
4                {
5                   val : Value   // étiquette
6                   fg : Arbre
7                   fd : Arbre
8                   pere : Arbre  // facultatif
9                }
```



- Type

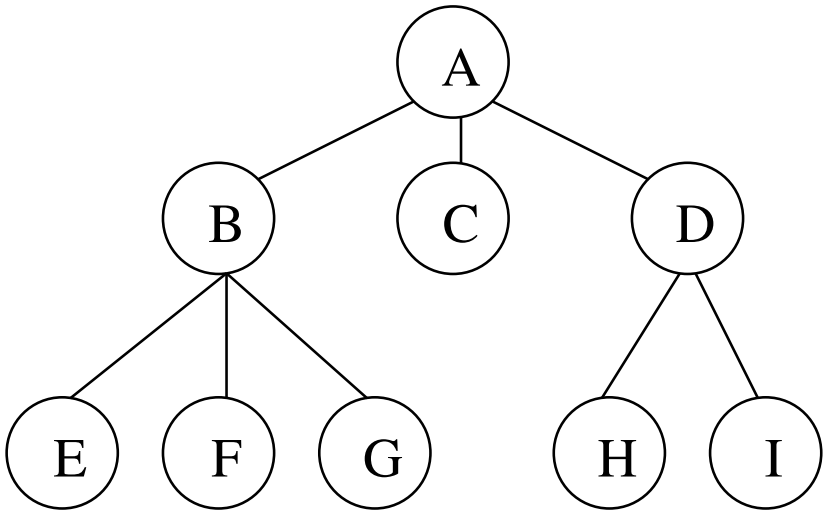
```
1  type Arbre = entier
2
3  type Sommet = structure
4                {
5                   val : Value   // étiquette
6                   fg : Arbre    // i.e. entier !
7                   fd : Arbre
8                   pere : Arbre  // facultatif
9                }
```

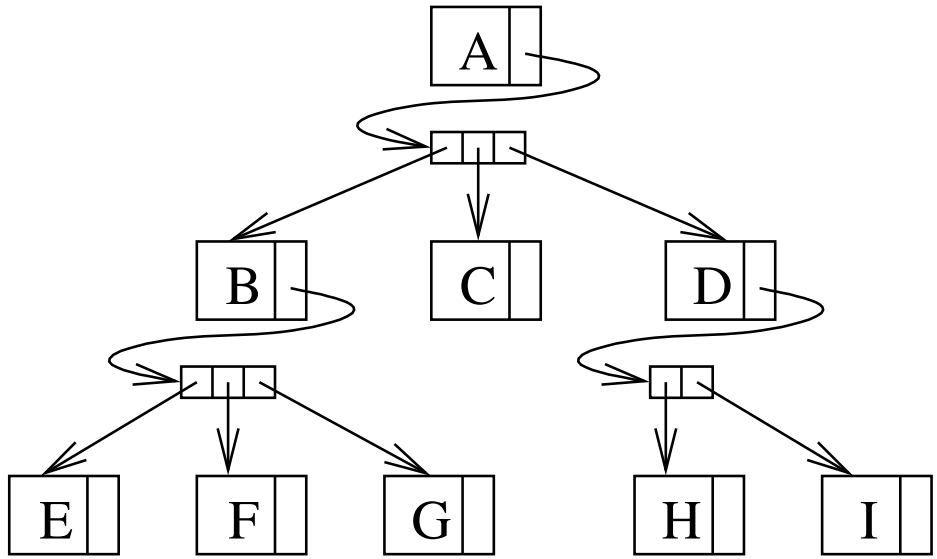
arbre | 4 |



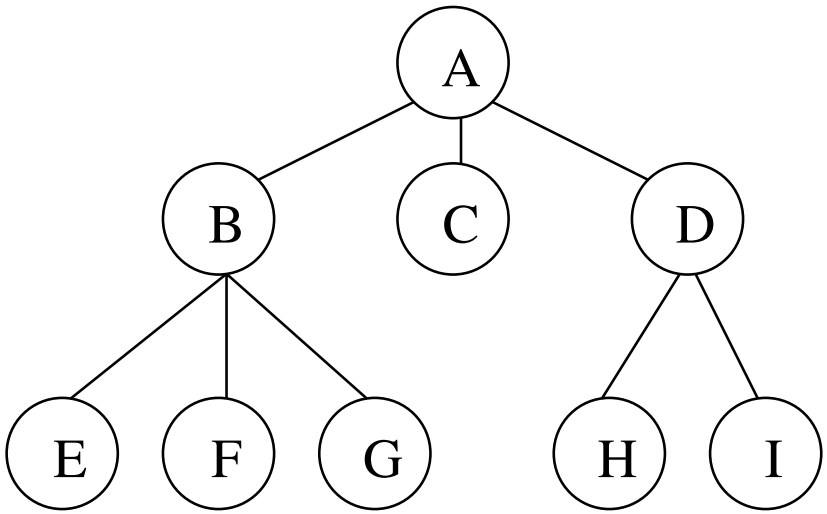
1	E		
2	B	5	1
3			
4	A	2	8
5	D		
6			
7	G		
8	C		

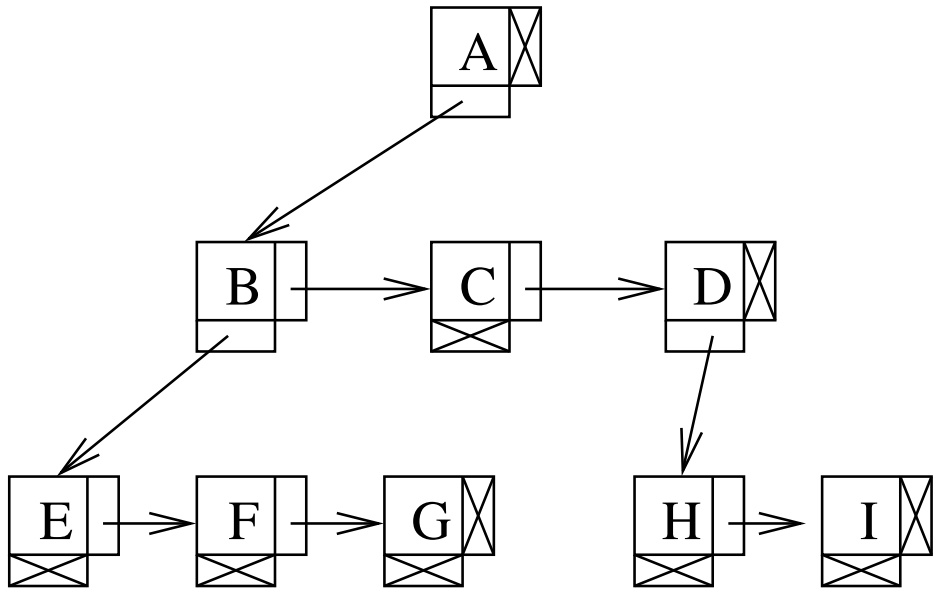
```
1  type Arbre = référence sur Sommet
2
3  type Sommet = structure
4              {
5                 val : Value   // étiquette
6                 fils : Liste_Arbres
7              }
8
9  type Liste_Arbres = à définir
10    // liste chaînée
11    // tableau extensible
12    // ...
```





```
1  type Arbre = référence sur Sommet
2
3  type Sommet = structure
4              {
5                 val : Value   // étiquette
6                 fils : Arbre
7                 frere : Arbre
8              }
```



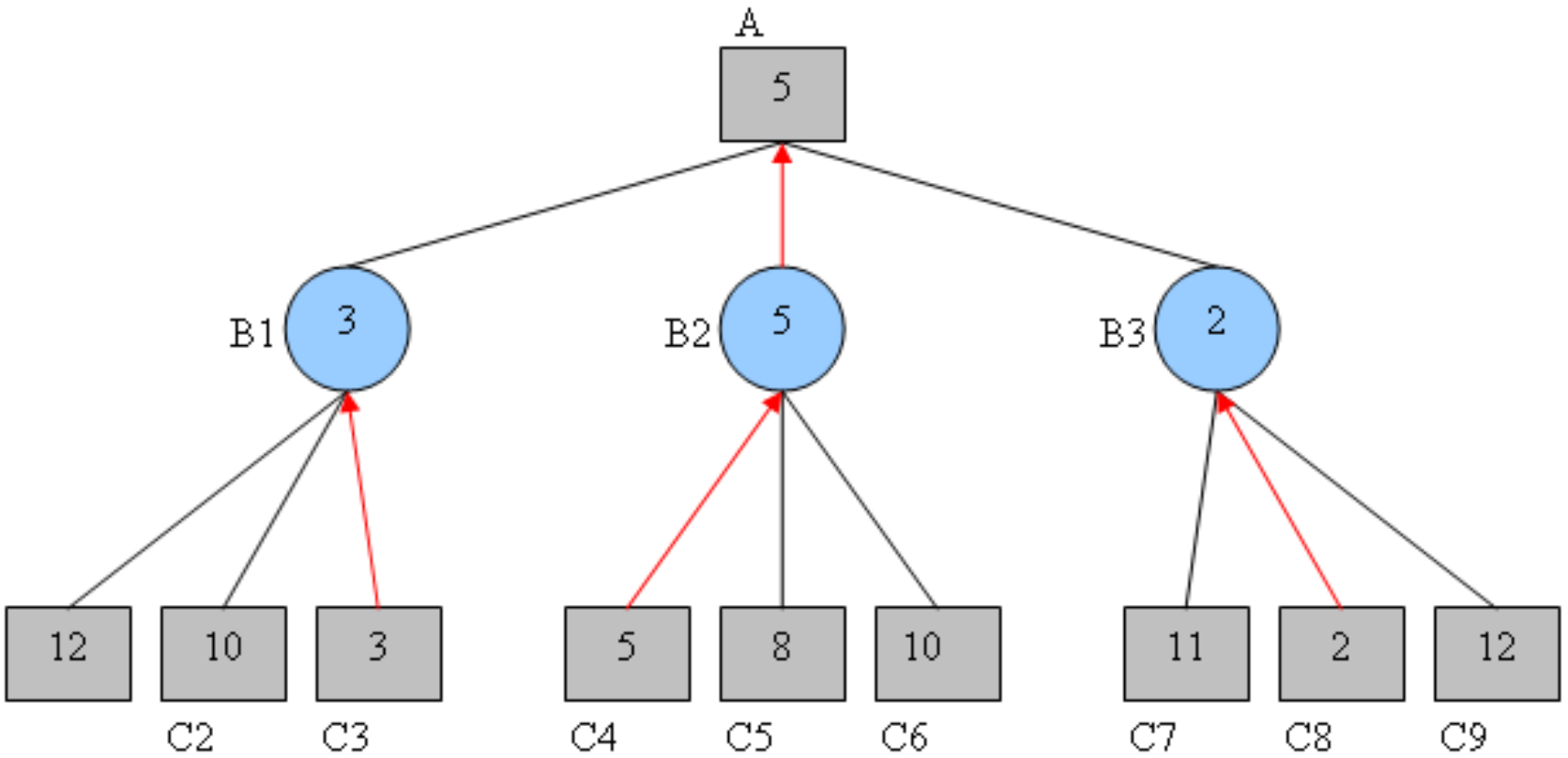


→ c'est un arbre binaire

Plan → A - Arbres

1. Généralités
2. Définitions
3. Représentation mémoire
4. Min/Max, élagage α/β

- Jeu à 2 joueurs
- information complète (i.e. pas de hasard ou d'inconnu)
- jouer tous les coups jusqu'à une profondeur fixée
- noter les positions finales
- déduire le meilleur coup

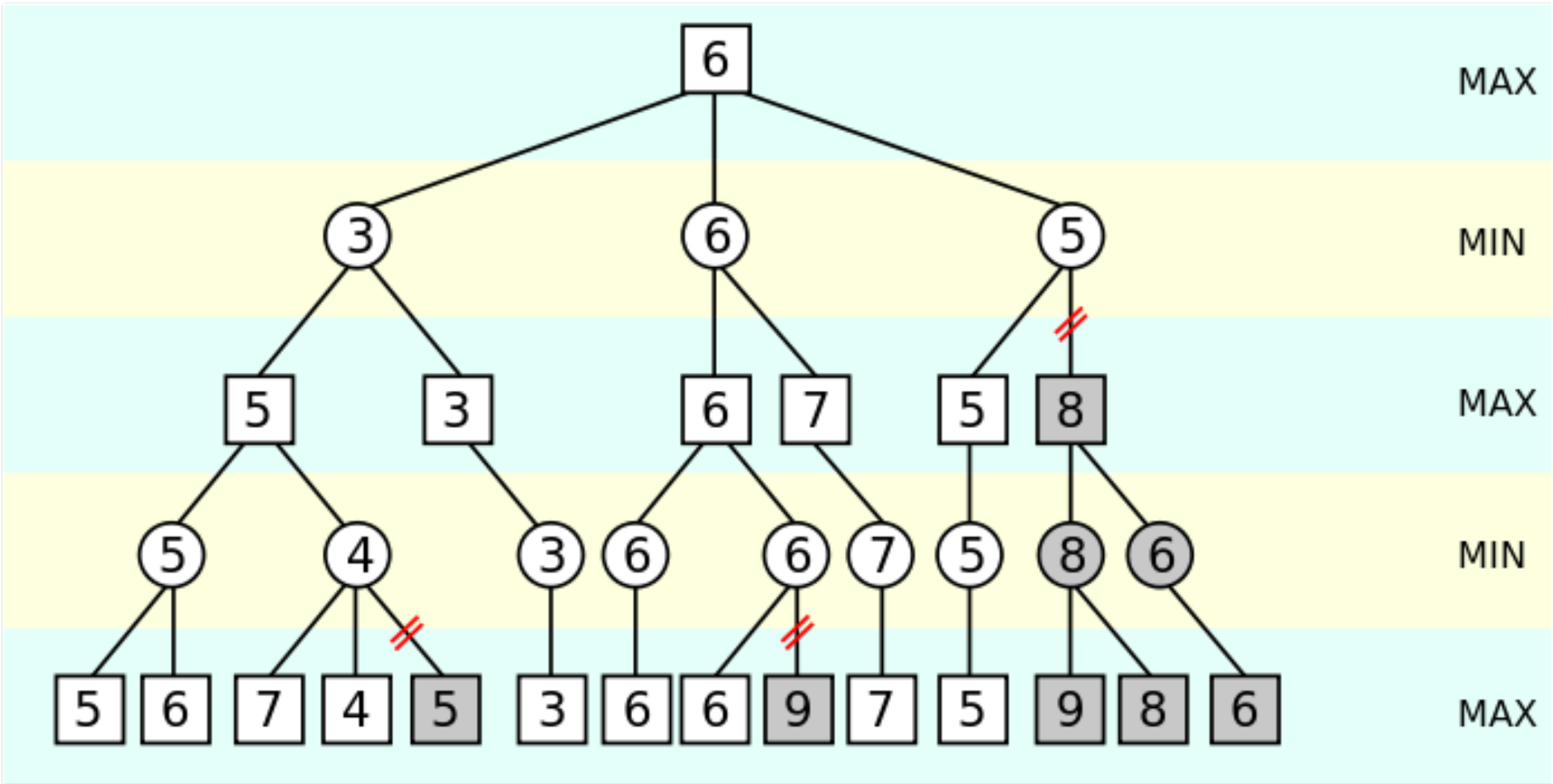

A
5
B1
3
B2
5
B3
2
12
10
3
5
8
10
11
2
12
C2
C3
C4
C5
C6
C7
C8
C9

- Théoriquement il faut jouer tous les coups possibles
- Pratiquement c'est impossible
- Exemple numérique
  - othello reversi
  - 60 coups pour finir la partie
  - hypothèse basse : 2 possibilités par coup

  $\Rightarrow$ 2^{60} cas différents ($\approx$ un milliard de milliard)
  - hypothèse : 1 milliard de possibilités traitées par seconde

  $\Rightarrow$ 36 ans

  $\Rightarrow$ 1300 milliards d'années si 3 possibilités
- Solutions
  - bibliothèques d'ouvertures
  - heuristiques


6
3
6
5
5
3
6
7
5
8
5
4
3
6
6
7
5
8
6
5
6
7
4
5
3
6
6
9
7
5
9
8
6
MAX
MIN
MAX
MIN
MAX