Arbres

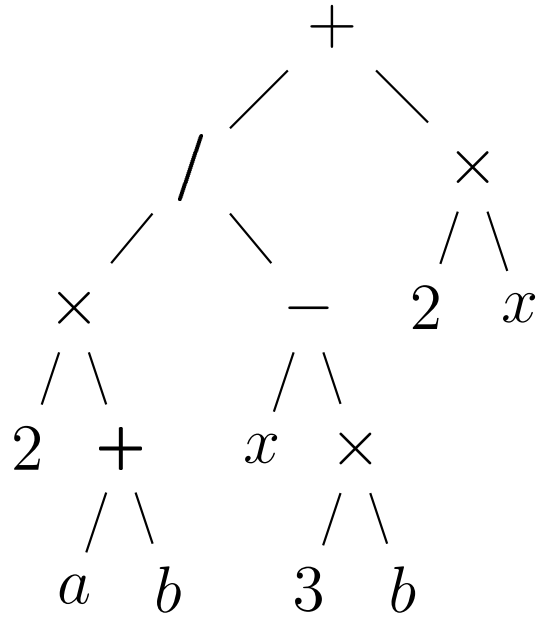
Yann Strozecki
yann.strozecki@uvsq.fr

Novembre 2015

Les arbres

Une des structures les plus importantes et les plus utilisées en informatique

- Arbres généalogiques
- Arbres de classification
- Arbres d'expression
- Données structurées, type XML ou HTML



Représentation de l'expression

$$(2 \times (a + b))/(x - 3 \times b) + 2 \times x$$

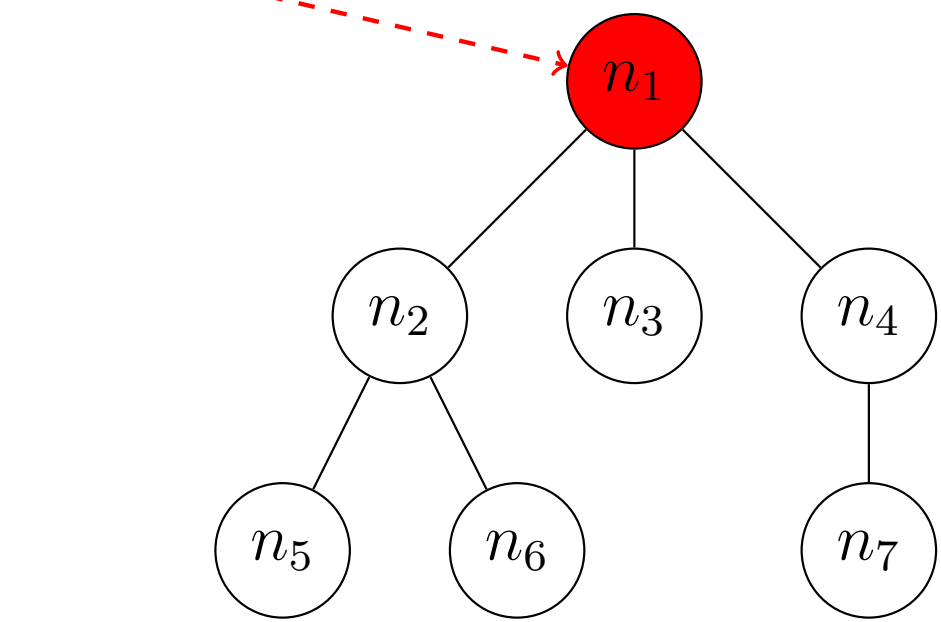
Terminologie

- Un **graphe** : un ensemble de nœuds reliés entre eux par des arêtes.
- Trois propriétés font d'un graphe un arbre **enraciné** :
  1. Il existe un nœud particulier nommé **racine**. Tout nœud c autre que la racine est relié par une arête à un nœud p appelé **père** de c.
  2. Un arbre est **connexe**.
  3. Un arbre est **sans cycle.**

Terminologie

- Un nœud peut avoir 0 ou plusieurs fils.
- Un nœud a exactement un père.

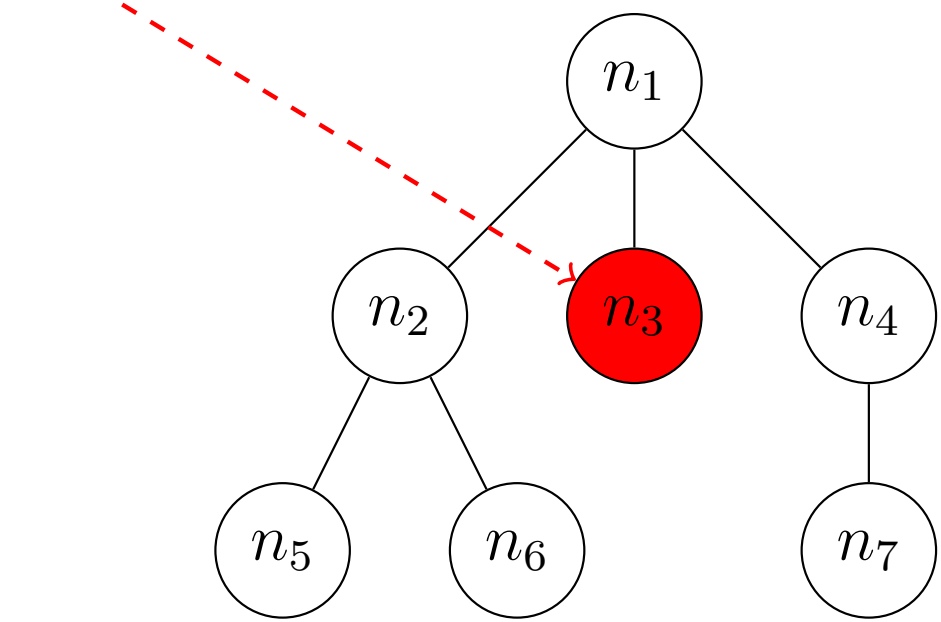
La racine



Terminologie

- Un nœud peut avoir 0 ou plusieurs fils.
- Un nœud a exactement un père.

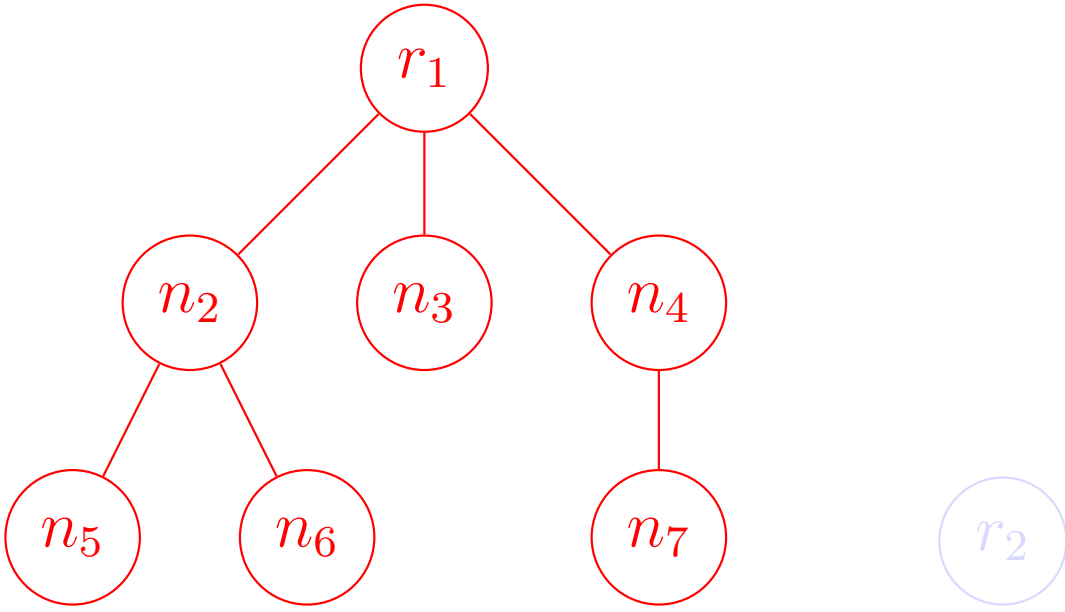
La racine



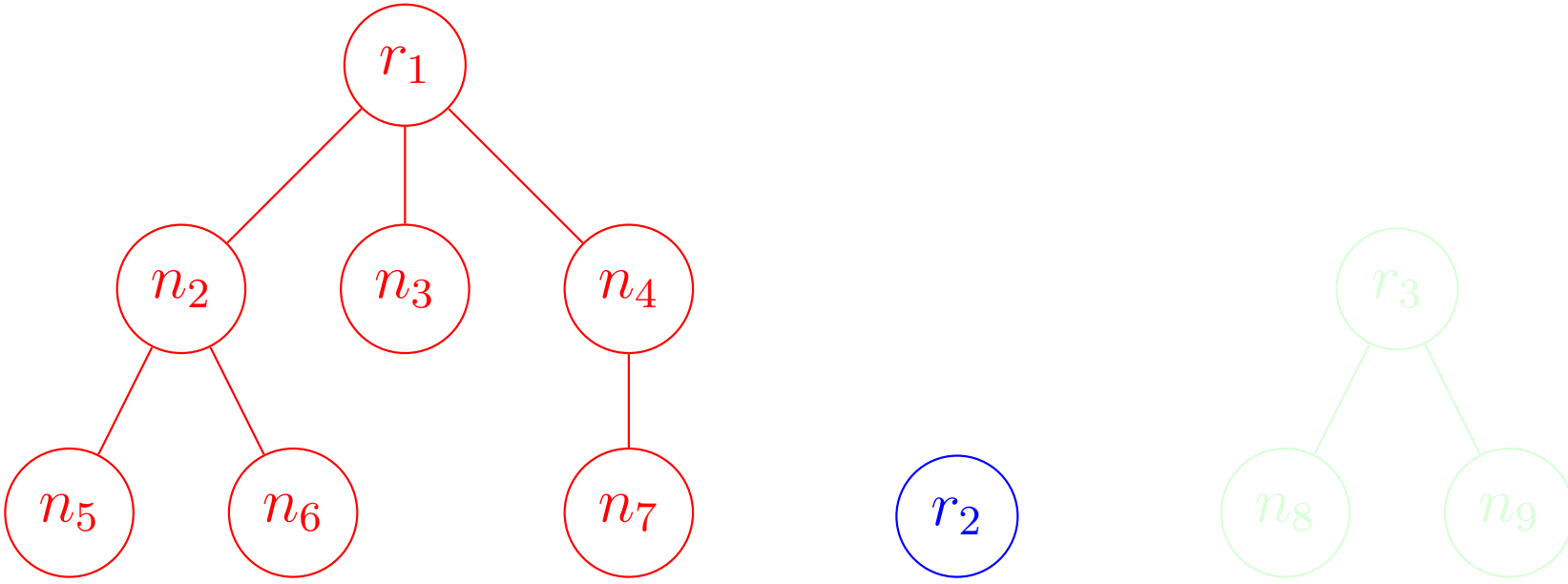
Définition récursive

- **Base :**
  - Un nœud unique n est un arbre
  - n est la racine de cet arbre.
- **Induction :**
  - Soit r un nouveau nœud
  - $T_1, T_2, \ldots, T_k$ sont des arbres ayant pour racine $r_1, r_2, \ldots, r_k$.
  - Création d'un nouvel arbre ayant pour racine r et on ajoute une arête entre r et r_1 r et r_2, ..., r et r_k.

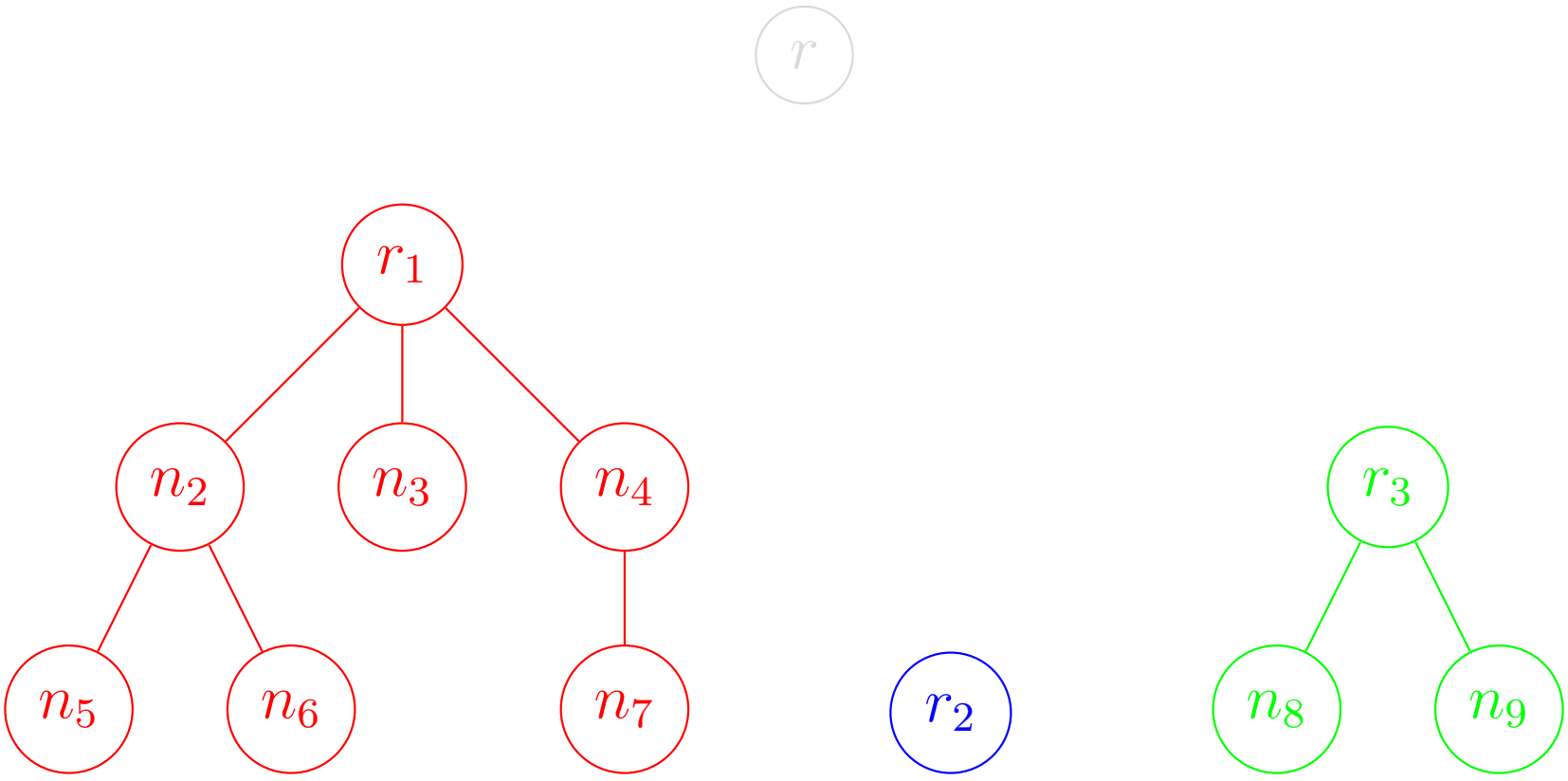
Définition récursive : un exemple



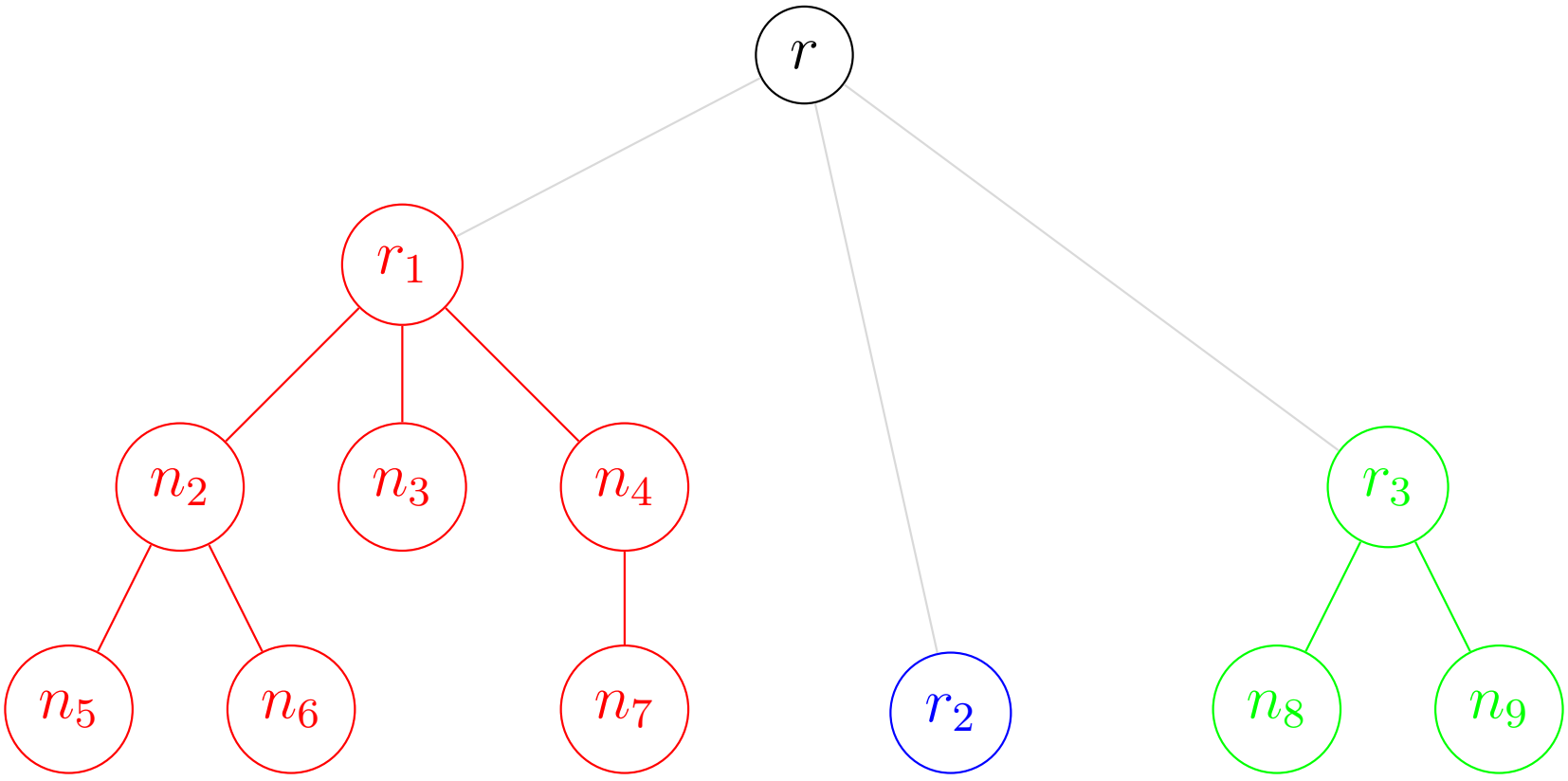
Définition récursive : un exemple



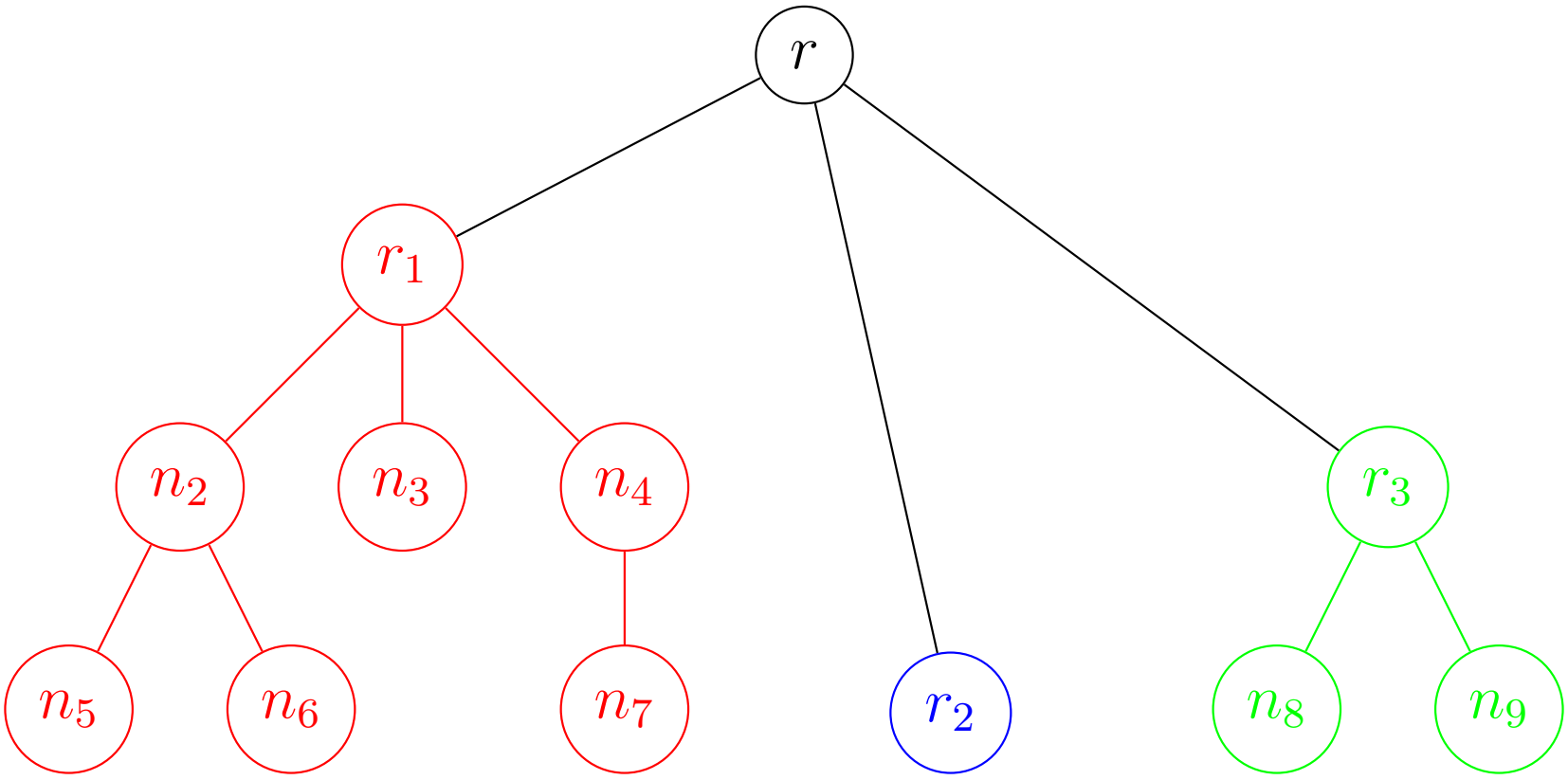
Définition récursive : un exemple



Définition récursive : un exemple



Définition récursive : un exemple



Chemins, ancêtres, descendants, . . .

- Les **ancêtres** d'un nœud : *Nœuds trouvés sur le* ***chemin unique*** *entre ce nœud et la racine*.
- Le nœud d est un **descendant** de a si et seulement si a est un ancêtre de d.
- **Longueur** d'un chemin = nombre d'arêtes parcourues.

Généalogie

- La racine est un ancêtre de tous les nœuds.
- Chaque nœud est un descendant de la racine.
- Les nœuds ayant le même père = **frères**.
- Un nœud n et tous ses descendants = **sous-arbre**

Feuilles et nœuds intérieurs

- Une **feuille** est un nœud qui n'a pas de fils
- Un nœud **intérieur** est un nœud qui a au moins 1 fils.
- Tout nœud de l'arbre est :
  - Soit une feuille
  - Soit un nœud intérieur

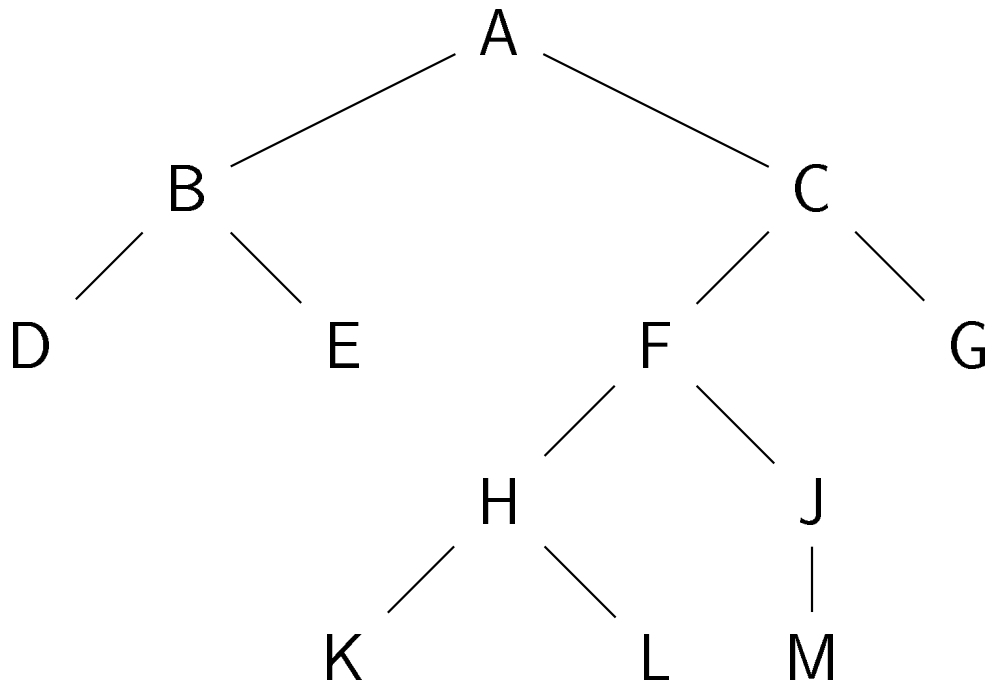
Mesure sur les arbres

- **Taille** de l'arbre T, notée *taille(T)* = nombre de nœuds.
- **Nombre de feuilles** noté *nf(T)*.
- La **profondeur d'un nœud** n, notée $h(n)$, est la longueur du chemin depuis la racine jusqu'à n.
- La **hauteur de l'arbre** T, notée $h(T)$:

$$h(T) = \max_{x \text{ nœud de l'arbre}} h(x)$$

Les arbres binaires

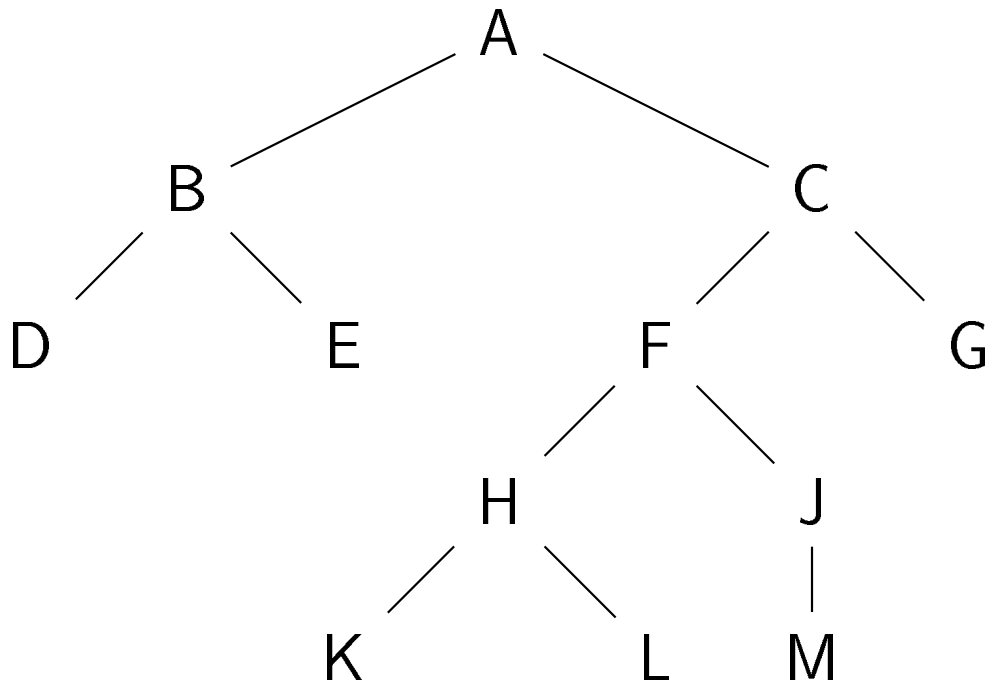
Tous les nœuds d'un arbre binaire ont $0, 1$ ou 2 fils.



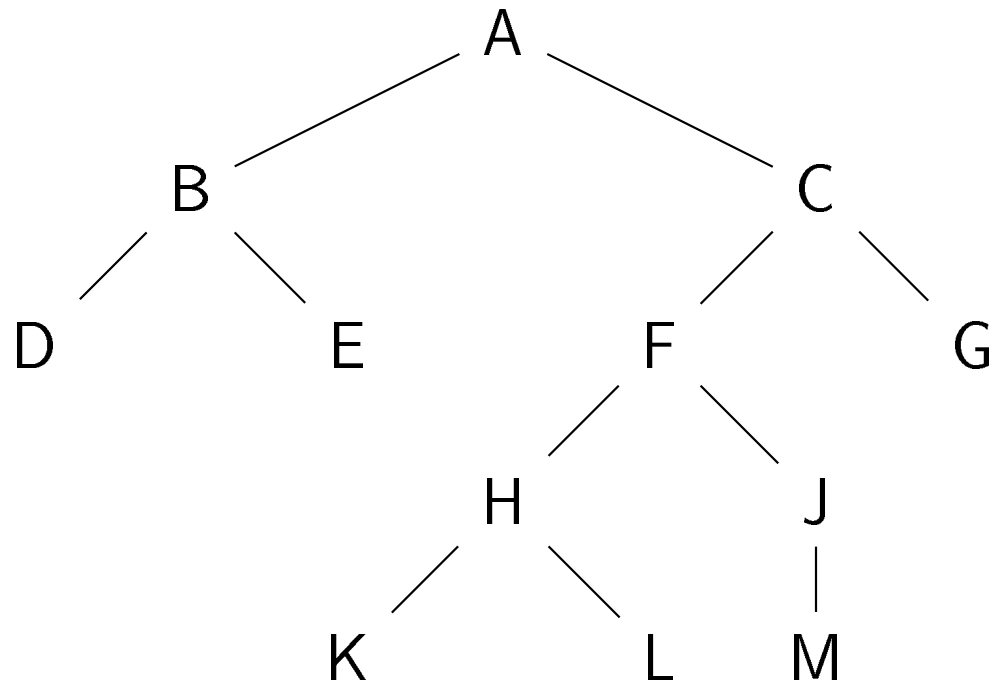
Les arbres binaires : Définitions

- **Fils gauche** de n = racine du sous-arbre gauche de n.
- **Fils droit** de n = racine du sous-arbre droit de n.
- **Bord gauche** de l'arbre = le chemin depuis la racine en ne suivant que des fils gauche.
- **Bord droit** de l'arbre = le chemin depuis la racine en ne suivant que des fils droits.

Exemple : arbre binaire



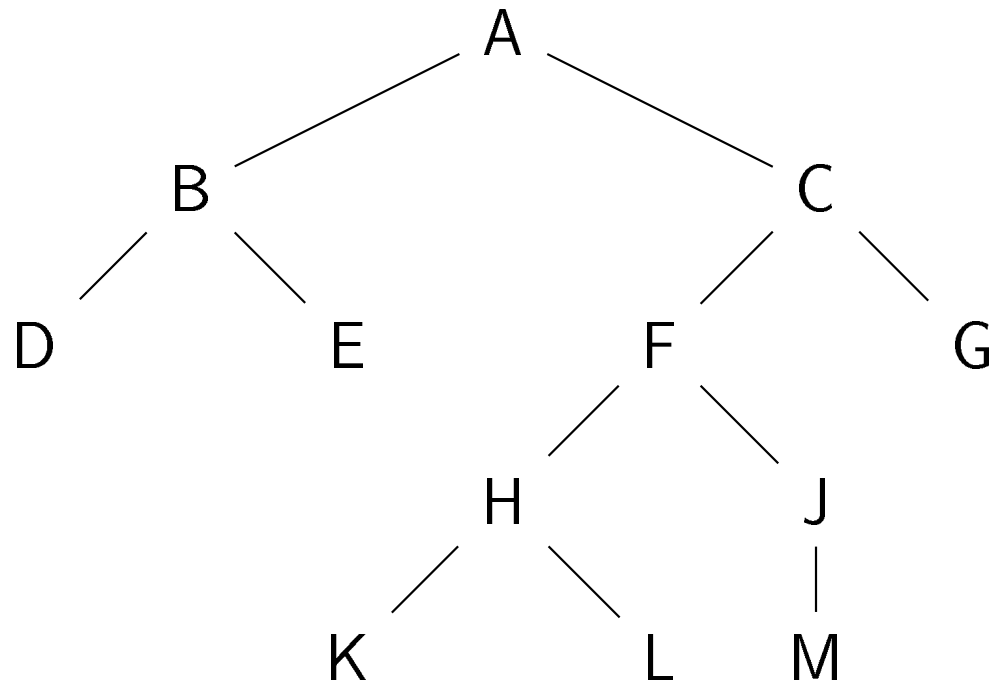
Exemple : arbre binaire



Taille de l'arbre :
Hauteur :
Hauteur moyenne :

Nombre de feuilles :
Longueur de cheminement :
Hauteur moyenne externe :

Exemple : arbre binaire



Taille de l'arbre : 12
Hauteur : 4
Hauteur moyenne : 2.33

Nombre de feuilles : 6
Longueur de cheminement : 28
Hauteur moyenne externe : 3

Quelques arbres binaires particuliers

1. Arbre binaire **filiforme**
2. **Peigne**
3. Arbre binaire **complet**
   - 1 nœud à la hauteur 0
   - 2 nœuds à la hauteur 1
   - 4 nœuds à la hauteur 2
   - ...
   - 2^h nœuds à la hauteur h.
   - Nombre total de nœuds d'un arbre de hauteur h :

$$2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1$$

Quelques arbres binaires particuliers

1. Arbre binaire *parfait* :
   - Tous les niveaux sont remplis sauf le dernier.
   - Les feuilles sont le plus à gauche possible.
2. Arbre binaire *localement complet* : chaque nœud a 0 ou 2 fils.

Propriétés sur les arbres (1)

Lemme

$$h(T) \leq taille(T) - 1$$

Idée de Preuve

Récurrence, égalité obtenue pour un arbre filiforme.

Propriétés sur les arbres (2)

Lemme

Pour tout arbre binaire T de taille n et de hauteur h on a :

$$\lfloor \log_2 n \rfloor \leq h \leq n - 1$$

Idée de Preuve

- Arbre filiforme : arbre de hauteur h ayant le plus petit nombre de nœuds : $n = h + 1$ (seconde inégalité).
- Arbre complet : arbre de hauteur h ayant le plus grand nombre de nœuds : $n = 2^{h+1} - 1$ (première inégalité).

Propriétés sur les arbres (3)

Corollaire

Tout arbre binaire non vide T ayant f feuilles a une hauteur $h(T)$ supérieure ou égale à $\lceil \log_2 f \rceil$.

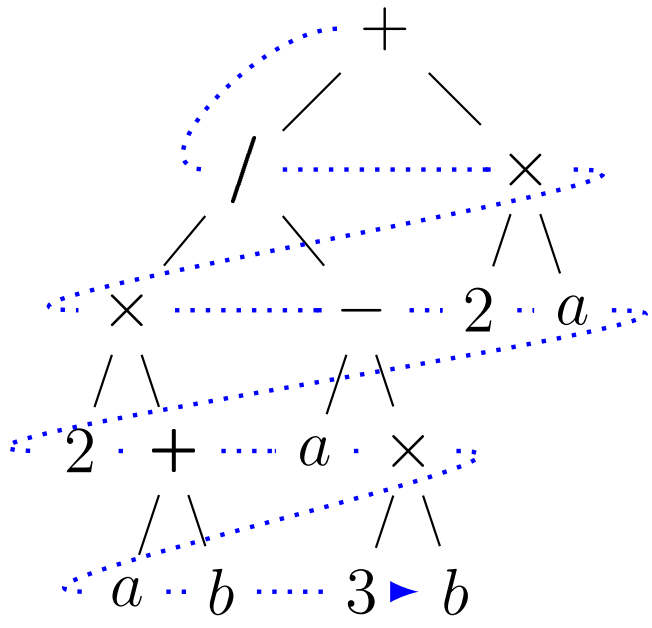
Lemme

Un arbre binaire localement complet ayant n nœuds internes a $(n+1)$ feuilles.

Exploration

- Pas aussi simple que dans le cas des listes.
- Pas d'ordre naturel.
- Deux types de parcours :
  - En largeur d'abord
  - En profondeur d'abord

Parcours en largeur d'abord



Ordre d'évaluation des nœuds :

$$+ / \times \times - 2\ a\ 2 + a \times a\ b\ 3\ b$$

Type de données

Mise en œuvre chaînée

```
Enregistrement Nœud {
    Val      : entier;
    Gauche : ↑ Nœud;
    Droit  : ↑ Nœud;
}
```

Le type Arbre est une référence sur un Noeud.

Exercice

Donner le pseudo-code des fonctions suivantes :

1. estunefeuille(Arbre r) renvoie vrai si r est une feuille, faux sinon
2. hauteur(Arbre r) renvoie la hauteur de l'arbre
3. rechercher(Arbre r, Entier e) renvoie vrai si un noeud de l'arbre r a comme valeur e, faux sinon
4. supprimer(Arbre r, Entier e) supprime tous les sous arbres de r dont la racine est de valeur e
5. allonge(Arbre r) déplace le sous-arbre droit de manière à ce qu'il soit le descendant d'une feuille du sous-arbre gauche

Parcours en largeur d'abord

Algorithme 1 Parcours en largeur d'un arbre binaire

```
ParcoursEnLargeur(r : Nœud)
▷ Entrée : r (la racine d'un arbre)
▷ Sortie : traitement de tous les nœuds de l'arbre enraciné en r
▷ Variables locales :
    ce_niveau, niveau_inférieur : File ;
    o : Nœud ;
Début
    ce_niveau ← { r } ;
    tant que (ce_niveau est non vide) faire
        niveau_inférieur = { } ;
        pour chaque nœud o de ce_niveau faire
            traiter o ;
            niveau_inferieur ← niveau_inférieur ∪ enfants de o.
        fin pour
        ce_niveau ← niveau_inférieur ;
    fin tant que
Fin
```

Parcours en profondeur d'abord

On descend sur le fils gauche tant qu'on peut et sinon on descend sur le fils droit.

Avec cette méthode on voit trois fois chaque nœud donc on peut choisir plusieurs ordres d'évaluation des nœuds.

Parcours en profondeur d'abord

- Parcours infixe :

$$2 \times a + b/a - 3 \times b + 2 \times a$$

- Parcours préfixe :

$$+/ \times 2 + a\ b - a \times 3\ b \times 2\ a$$

- Parcours postfixe

$$2\ a\ b + \times a\ 3\ b \times -/2\ a \times +$$

Parcours en profondeur d'abord

Algorithme 2 Parcours en profondeur d'un arbre binaire

```
ParcoursEnProfondeur(r : Nœud)
▷ Entrée : r (la racine d'un arbre)
▷ Sortie : traitement de tous les nœuds de l'arbre enraciné en r
Début
    si r = ∅
        traitement de l'arbre vide
    sinon
        traitement_prefixe(r) ;
        ParcoursEnProfondeur(r.Gauche) ;
        traitement_infixe(r) ;
        ParcoursEnProfondeur(r.Droit) ;
        traitement_postfixe(r) ;
    fin si
Fin
```

Arbres généraux et forêts

- **Arbre général** : arbre où les nœuds peuvent avoir un nombre quelconque de fils.
- **Forêt** : collection d'arbres en nombre quelconque

Représentation

- Tableau de fils
- Fils aîné et frère droit (bijection avec les arbres binaires).

Arbres généraux : affichage préfixé

Algorithme 3 Parcours préfixe d'un arbre général

```
Prefixe(r : Nœud)
▷ Entrée : r (la racine d'un arbre)
▷ Sortie : traitement de tous les nœuds de l'arbre enraciné en r
▷ Variable locale :
    t : Nœud ;
Début
    traiter le nœud r
    t ← r.FilsAine ;
     tant que t ≠ NIL  faire
        Prefixe(t) ;
        t ← t.FrereDroit ;
    Ftque
Fin
```

Arbres généraux : affichage postfixé

Algorithme 4 Parcours postfixe d'un arbre général

```
Postfixe(r : Nœud)
▷ Entrée : r (la racine d'un arbre)
▷ Sortie : traitement de tous les nœuds de l'arbre enraciné en r
▷ Variable locale :
    t : Nœud ;
Début
    t ← r.FilsAine ;
     tant que t ≠ NIL  faire
        Postfixe(t) ;
        t ← t.FrereDroit ;
    Ftque
    traiter le nœud r
Fin
```

A vous de jouer

On va décrire ici une méthode qui permet de résoudre les jeux à deux joueurs qui jouent alternativement. Je vous donne les étapes et vous me donner leur implémentation pour le jeu du morpion.

1. Proposer une structure de donnée pour stocker l'état du jeu de morpion à n'importe quelle étape de la partie.
2. Une partie est une suite de coups, comment représenter une partie ?

A vous de jouer

On va décrire ici une méthode qui permet de résoudre les jeux à deux joueurs qui jouent alternativement. Je vous donne les étapes et vous me donner leur implémentation pour le jeu du morpion.

1. Proposer une structure de donnée pour stocker l'état du jeu de morpion à n'importe quelle étape de la partie.
2. Une partie est une suite de coups, comment représenter une partie ?
3. Proposer une structure arborescente dont chaque branche représente une partie possible.

A vous de jouer

On va décrire ici une méthode qui permet de résoudre les jeux à deux joueurs qui jouent alternativement. Je vous donne les étapes et vous me donner leur implémentation pour le jeu du morpion.

1. Proposer une structure de donnée pour stocker l'état du jeu de morpion à n'importe quelle étape de la partie.
2. Une partie est une suite de coups, comment représenter une partie ?
3. Proposer une structure arborescente dont chaque branche représente une partie possible.
4. Donner un algorithme qui à partir d'un état du jeu donné et du numéro du joueur actif construit tous les états de jeu atteignable en un coup.

A vous de jouer

On va décrire ici une méthode qui permet de résoudre les jeux à deux joueurs qui jouent alternativement. Je vous donne les étapes et vous me donner leur implémentation pour le jeu du morpion.

1. Proposer une structure de donnée pour stocker l'état du jeu de morpion à n'importe quelle étape de la partie.
2. Une partie est une suite de coups, comment représenter une partie ?
3. Proposer une structure arborescente dont chaque branche représente une partie possible.
4. Donner un algorithme qui à partir d'un état du jeu donné et du numéro du joueur actif construit tous les états de jeu atteignable en un coup.
5. Construire l'arbre de tous les états possibles.

A vous de jouer

On va décrire ici une méthode qui permet de résoudre les jeux à deux joueurs qui jouent alternativement. Je vous donne les étapes et vous me donner leur implémentation pour le jeu du morpion.

1. Proposer une structure de donnée pour stocker l'état du jeu de morpion à n'importe quelle étape de la partie.
2. Une partie est une suite de coups, comment représenter une partie ?
3. Proposer une structure arborescente dont chaque branche représente une partie possible.
4. Donner un algorithme qui à partir d'un état du jeu donné et du numéro du joueur actif construit tous les états de jeu atteignable en un coup.
5. Construire l'arbre de tous les états possibles.

A vous de jouer

On appelle ces arbres des arbres minimax car on alterne des objectifs contraires.

Comment peut-on utiliser l'arbre qu'on a construit pour le morpion pour calculer une stratégie optimale (qui répond le mieux possible à n'importe quel coup de l'adversaire) ?

A vous de jouer

On appelle ces arbres des arbres minimax car on alterne des objectifs contraires.

Comment peut-on utiliser l'arbre qu'on a construit pour le morpion pour calculer une stratégie optimale (qui répond le mieux possible à n'importe quel coup de l'adversaire) ?

On ajoute à chaque noeud le champ gain. Il vaut 1 si le premier joueur gagne quelque soit la stratégie de l'adversaire à partir de cette configuration. Il vaut -1 si il perd et 0 si il peut assurer un nul. Calculer ce champ sur tout l'arbre.

A vous de jouer

On appelle ces arbres des arbres minimax car on alterne des objectifs contraires.

Comment peut-on utiliser l'arbre qu'on a construit pour le morpion pour calculer une stratégie optimale (qui répond le mieux possible à n'importe quel coup de l'adversaire) ?

On ajoute à chaque noeud le champ gain. Il vaut 1 si le premier joueur gagne quelque soit la stratégie de l'adversaire à partir de cette configuration. Il vaut -1 si il perd et 0 si il peut assurer un nul. Calculer ce champ sur tout l'arbre.

Exemple au tableau + coupes $\alpha\beta$.

A vous de jouer

On appelle ces arbres des arbres minimax car on alterne des objectifs contraires.

Comment peut-on utiliser l'arbre qu'on a construit pour le morpion pour calculer une stratégie optimale (qui répond le mieux possible à n'importe quel coup de l'adversaire) ?

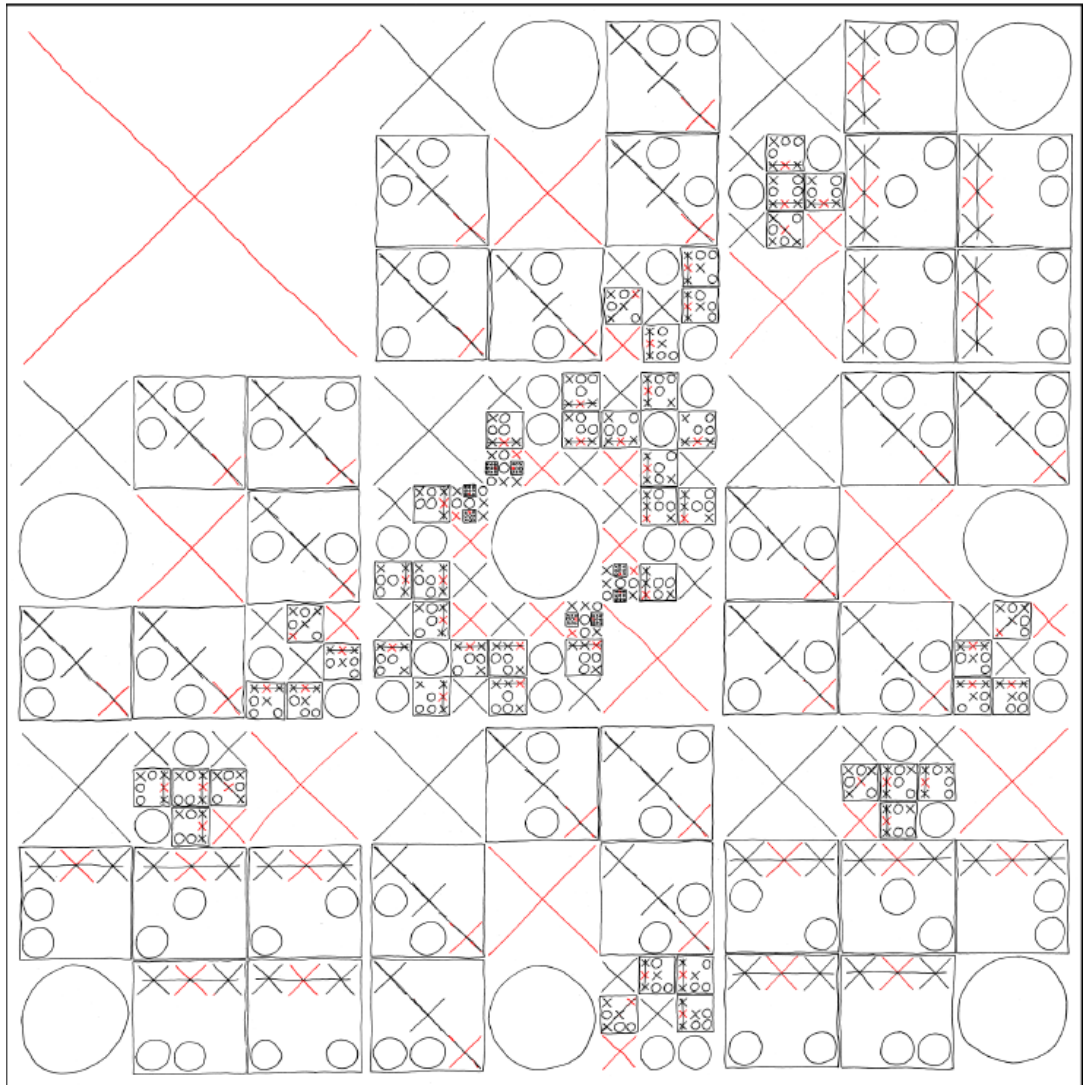
On ajoute à chaque noeud le champ gain. Il vaut 1 si le premier joueur gagne quelque soit la stratégie de l'adversaire à partir de cette configuration. Il vaut -1 si il perd et 0 si il peut assurer un nul. Calculer ce champ sur tout l'arbre.

Exemple au tableau + coupes $\alpha\beta$.

COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

MAP FOR X:



MAP FOR O: