Algorithmique Avancée

Nicolas Labroche

Cours 5 : arbres binaires

Université de Tours

Arbres binaires

Avant-propos

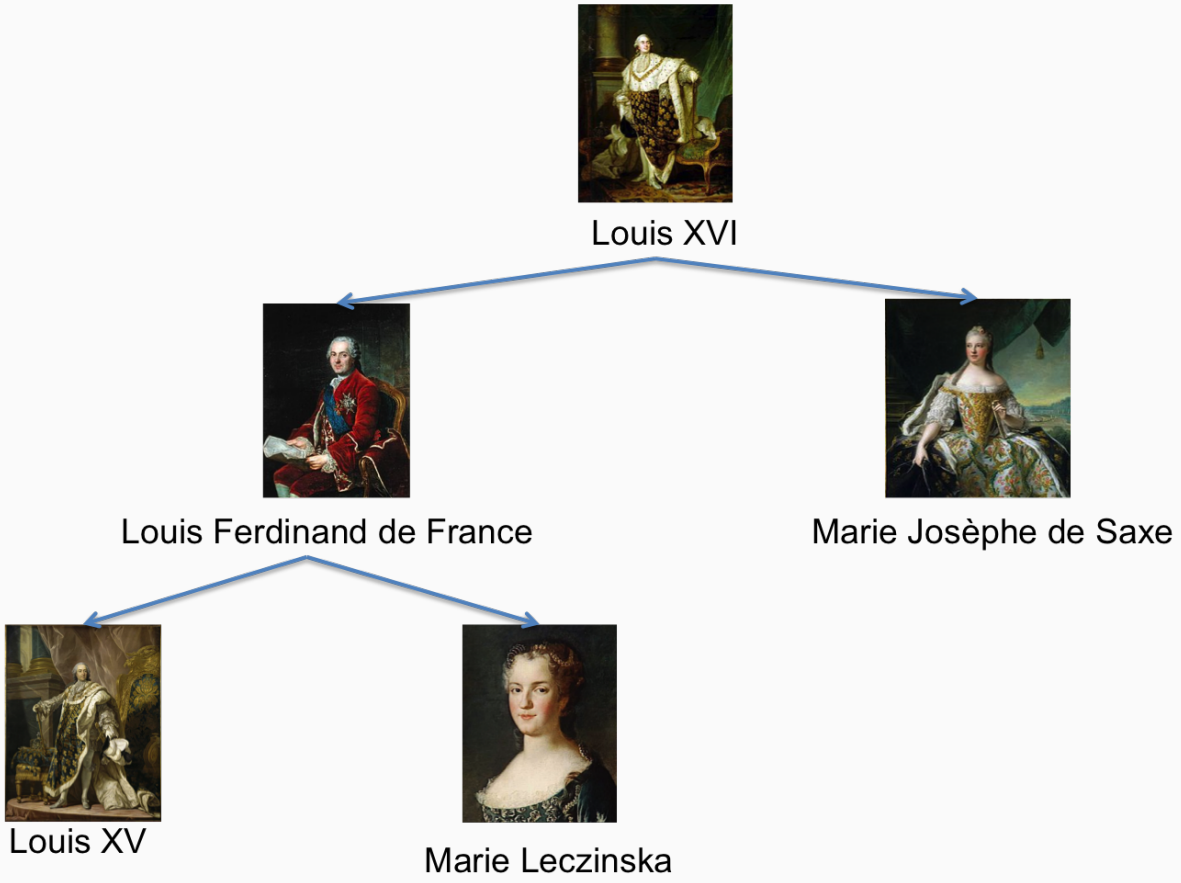
- Contrairement à la partie sur les listes, la partie sur les arbres suit un modèle plus simple qui n'introduit pas de classe de "gestion" ou "conteneur" pour représenter la structure de données comme `SList`
- Il n'est donc pas possible de construire une classe représentant un arbre vide
- Ce n'est pas grave car il n'est pas utile de faire un parallèle avec les structures de base Java (elles n'existent pas)

Types d'arbres

- Arbres binaires
  - arbre généalogique des ascendants
  - arbre représentant une expression arithmétique simple
- Arbres généraux
  - arbre généalogique de la descendance
  - arborescence du système de fichier d'un ordinateur
  - noms internet des ordinateurs :
    `sous-domaine.domaine.{fr|edu|com|org}`
  - table des matières d'un livre :
    `/chapitre/section/sous-section/paragraphe`

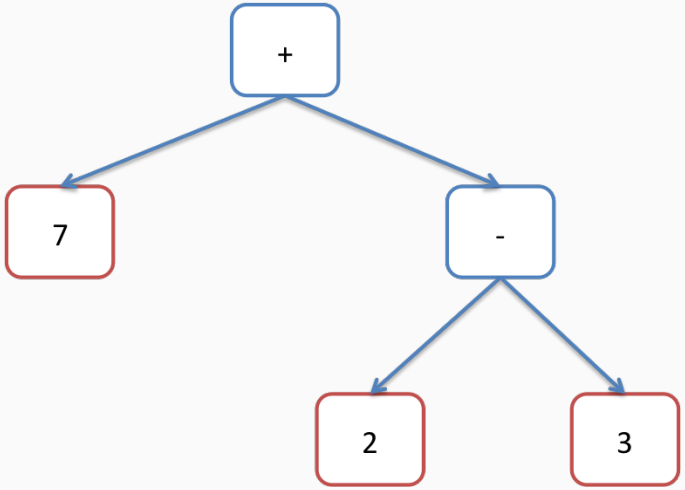
Exemple d'arbre binaire (1)

Arbre généalogique d'ascendance :



Exemple d'arbre binaire (2)

- Expression arithmétique simple : $7 + (2 - 3)$



- Vocabulaire
  - nœuds (en bleu ou rouge), feuilles (en rouge)
  - étiquette, père, fils, sous-arbre . . .

Définition d'un arbre binaire

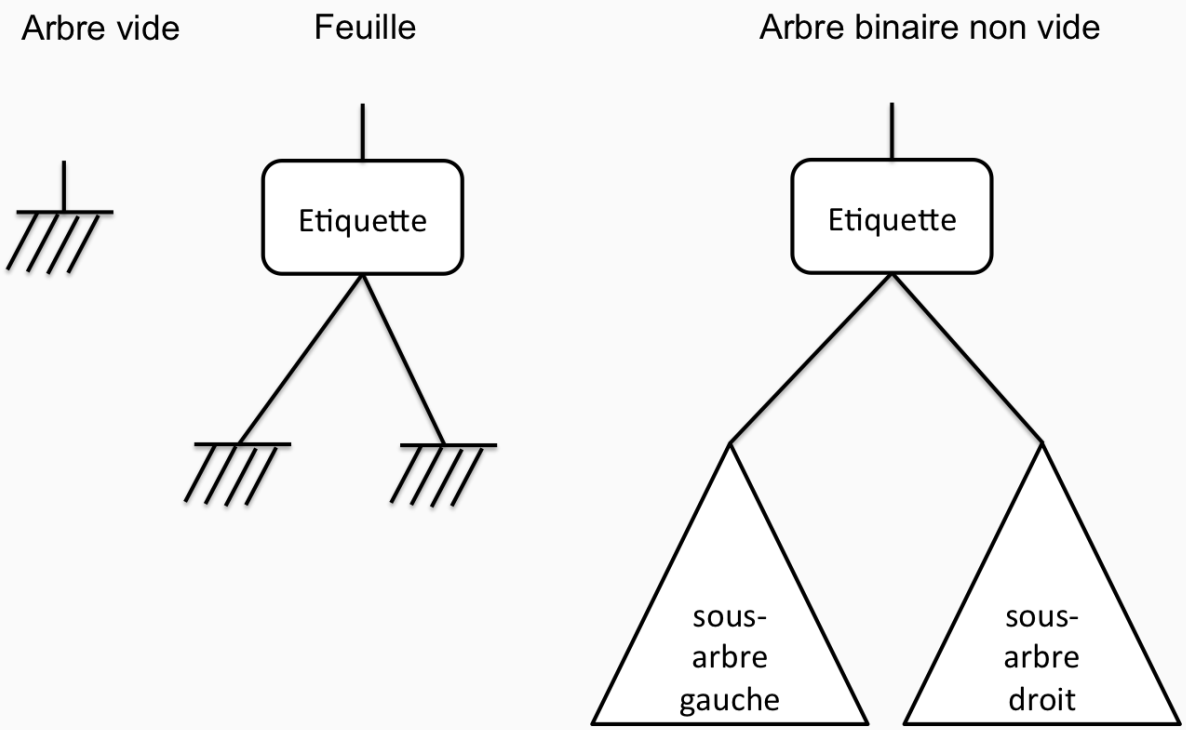
Un arbre binaire est une structure de données

- pour organiser des éléments de même type de façon hiérarchique
- auto-référentielle (récursive) avec deux références

Un arbre binaire est

- soit vide (en Java = `null`)
- soit formé
  - d'un nœud portant une étiquette contenant tous types de valeur
  - d'un sous-arbre gauche qui est un arbre binaire
  - d'un sous-arbre droit qui est un arbre binaire
- au plus formé de 2 fils
- mais peut n'en avoir aucun : on parle alors d'une feuille

Illustration du vocabulaire



Représentation Java d'un arbre binaire

```
public class AB<T> {
  T label;
  AB<T> left;
  AB<T> right;

  public AB(T _label){
    label = _label;
    left = null;
    right = null;
  }

  public AB(T _label, AB<T> _left, AB<T> _right){
    label = _label;
    left = _left;
    right = _right;
  }
}
```

Construction d'arbres binaires

- Constructeurs
  - un arbre vide

```
AB<String> ab = null;
```

  - une feuille à l'aide de la valeur de l'étiquette (les fils sont à `null`)

```
AB<String> ab = new AB<String>(``Hello'');
```

  - un arbre binaire avec un (ou plusieurs) fils

```
AB<String> left = new AB<String>(``Left'');
AB<String> ab = new AB<String>(``Hello'',
    left, null);
```

Feuilles et sous-arbres vides

Mise en place de tests basiques sur la structure de l'arbre binaire

- `isLeaf()` : méthode qui permet de savoir si un arbre binaire est une feuille

```
public boolean isLeaf(){
  return (getLeft() == null && getRight() == null);
}
```

- `hasLeft()` et `hasRight()` : pour savoir si un des 2 fils est vide (et éviter d'appeler une méthode sur un objet vide)

```
public boolean hasLeft() {
  return getLeft() != null;
}
```

Exemple de création d'un arbre binaire

```
public static AB<Character> ab1(){
  AB<Character> u = new AB<Character>('u');
  AB<Character> r = new AB<Character>('r');
  AB<Character> f = new AB<Character>('f', null, u);
  AB<Character> t = new AB<Character>('t', r, null);
  AB<Character> v = new AB<Character>('v');
  AB<Character> s = new AB<Character>('s');
  AB<Character> e = new AB<Character>('e', v, s);
  AB<Character> g = new AB<Character>('g', f, t);
  AB<Character> h = new AB<Character>('h', e, g);
  return h;
}
```

- Question : représenter l'arbre binaire correspondant

Récursion sur les arbres binaires

Un arbre binaire est

- soit vide
- soit constitué d'une étiquette, d'un sous-arbre gauche et d'un sous-arbre droit
- c'est une structure construite récursivement $\Rightarrow$ traitement récursif !

Algorithme récursif de traitement

- si l'arbre est réduit à une feuille : gérer l'étiquette (label)
- sinon : il faut traiter tous les nœuds
  - traiter le nœud racine et éventuellement son étiquette
  - traiter récursivement les sous-arbres
    - traiter le sous-arbre gauche
    - traiter le sous-arbre droit

Exemple 1 : nombre de nœuds (1)

- Objectif : compter le nombre de nœuds d'un arbre binaire
- Principe : le nombre de nœuds d'un arbre binaire est
  - 0 si l'arbre vide
  - 1 nœud qui contient l'étiquette de la racine
  - + le nombre de nœuds de son sous-arbre gauche
  - + le nombre de nœuds de son sous-arbre droit

Exemple 1 : nombre de nœuds (2)

```
public int nbNodes(){
  if (isLeaf()) return 1;
  else {
    int left = (hasLeft())? getLeft().nbNodes():0;
    int right = (hasRight())? getRight().nbNodes():0;
    return 1 + left + right;
  }
}
```

- Question : dérouler l'algorithme sur l'arbre binaire précédent

Exemple 2 : hauteur d'un arbre binaire (1)

- Objectif : déterminer la hauteur d'un arbre binaire
  - le nombre de nœuds existants sur sa plus grande branche
- Principe : la hauteur de l'arbre égale
  - 0 lorsque l'arbre est vide
  - 1 pour comptabiliser le nœud racine lorsque l'arbre est non vide
  - + la hauteur maximale entre les sous-arbres gauche et droit

Exemple 2 : hauteur d'un arbre binaire (2)

```
public int height(){
  if (isLeaf()) return 1;
  else {
    int left = (hasLeft())? getLeft().height():0;
    int right = (hasRight())? getRight().height():0;
    return 1 + + Math.max(left, right);
  }
}
```

- Questions : en utilisant l'algorithme précédent
  - quelle est la hauteur d'une feuille ? D'un arbre vide ?
  - quelle est la hauteur de l'arbre binaire précédent ?

Schéma de récursivité

Schéma qui combine (action `combiner`) :

- le traitement (action `traiter`)de l'étiquette avec,
- le traitement des sous-arbres gauche et droit par **récursivité**
- `traiter` et `combiner` dépendent de ce que l'on souhaite faire
- attention : les appels sur `getLeft()`/`getRight()` nécessitent que ces arbres existent !

Schéma de récursivité

```
public <K> K fonc(){
  if (isLeaf()) {
    // cas de base : arbre = feuille
  } else {
    // cas general
    K left = (getLeft() != null)? getLeft().fonc() : null;
    K right = (getRight() != null)? getRight().fonc() : null;
    return combiner(traiter(label(), left, right);
  }
}
```

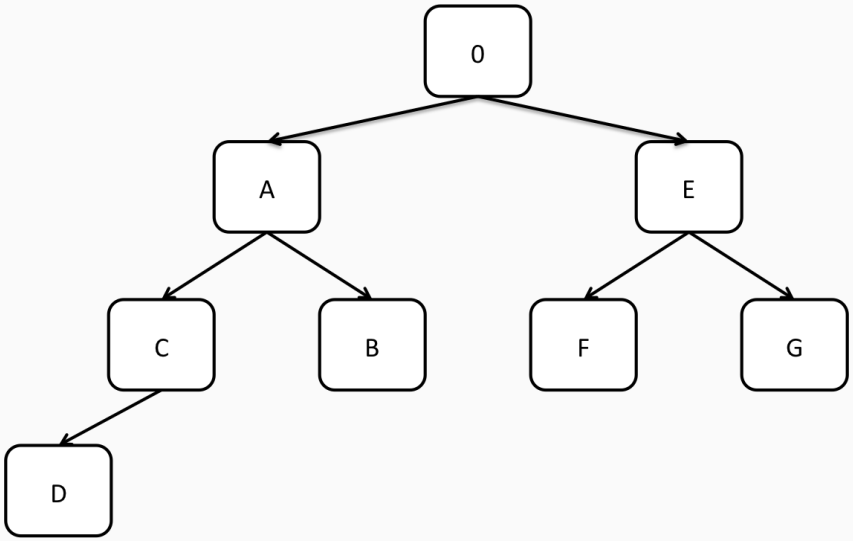
Attention : `null` doit être adapté selon le traitement

Parcours d'arbres binaires : liste des nœuds

- Parcourir un arbre : visiter chacun de ses nœuds
- Exemples d'utilisation
  - récupérer la liste des étiquettes d'un arbre (affichage)
  - rechercher une étiquette
- 3 parcours d'un arbre binaire
  1. parcours préfixe : visite de l'étiquette $\rightarrow$ parcours préfixe du sous-arbre gauche $\rightarrow$ parcours préfixe du sous-arbre droit
  2. parcours infixe : parcours infixe du sous-arbre gauche $\rightarrow$ visite de l'étiquette $\rightarrow$ parcours infixe du sous-arbre droit
  3. parcours postfixe : parcours postfixe du sous-arbre gauche $\rightarrow$ parcours postfixe du sous-arbre droit $\rightarrow$ visite de l'étiquette

Exemples de parcours d'arbre binaire

Soit l'arbre binaire suivant



- Parcours préfixe : `(0, A, C, D, B, E, F, G)`
- Parcours infixe : `(D, C, A, B, 0, F, E, G)`
- Parcours postfixe : `(D, C, B, A, F, G, E, 0)`

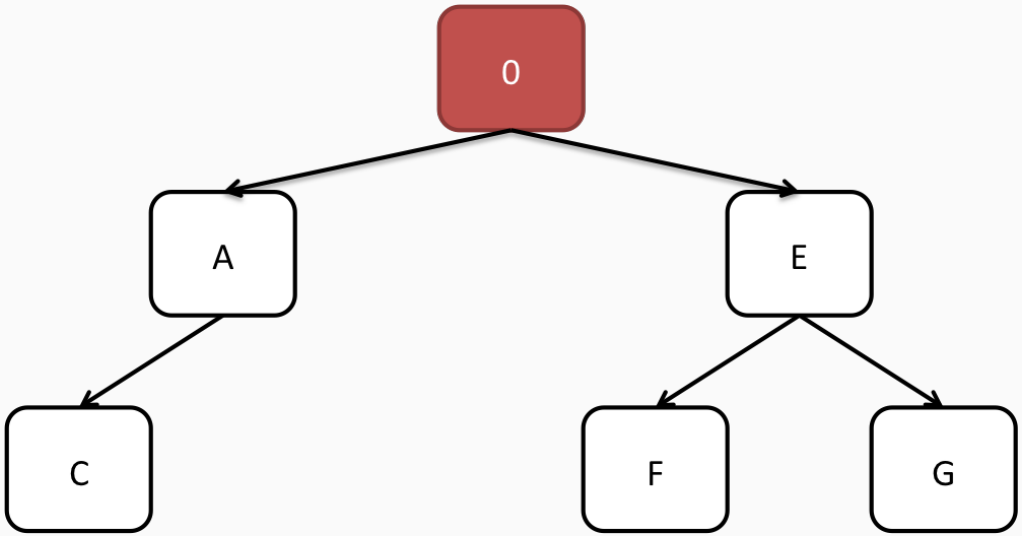
Parcours infixe en Java

On supposera disposer d'une classe `ISList` de liste chaînée implémentant les itérateurs Java.

```
public ISList<T> infix(){
  if (isLeaf()) return new ISList<T>(getLabel(), null);
  else {
    ISList<T> left = (hasLeft())? getLeft().infix() : new ↩
        ISList<>();
    ISList<T> right = (hasRight())? getRight().infix() : new ↩
        ISList<>();
    left.add(getLabel()); // add label between left and right
    Iterator<T> it = right.iterator();
    while (it.hasNext()){
      left.add(it.next());
    }
    return left;
  }
}
```

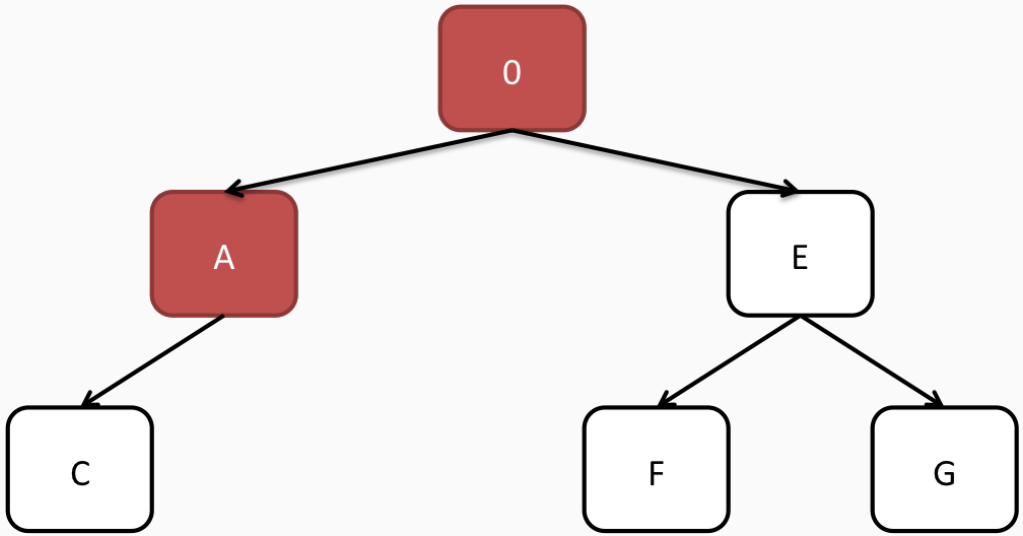
Affichage d'un arbre binaire

- Pas de fonction en Java pour afficher simplement un arbre
- Pour afficher un arbre $\Rightarrow$ parcours préfixe
  - exploration en profondeur de chaque chemin avant de revenir en arrière
- Illustration



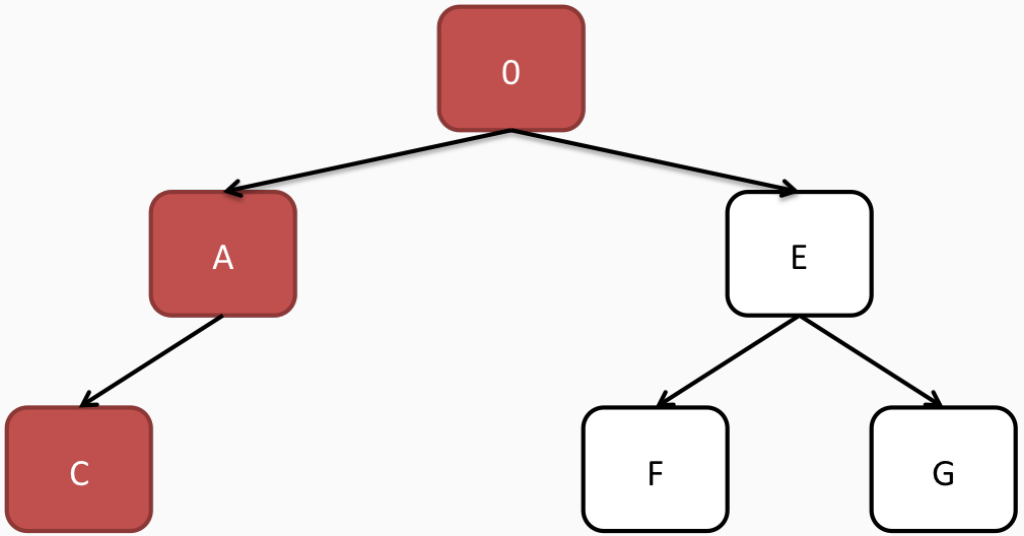
Affichage d'un arbre binaire

- Pas de fonction en Java pour afficher simplement un arbre
- Pour afficher un arbre $\Rightarrow$ parcours préfixe
  - exploration en profondeur de chaque chemin avant de revenir en arrière
- Illustration



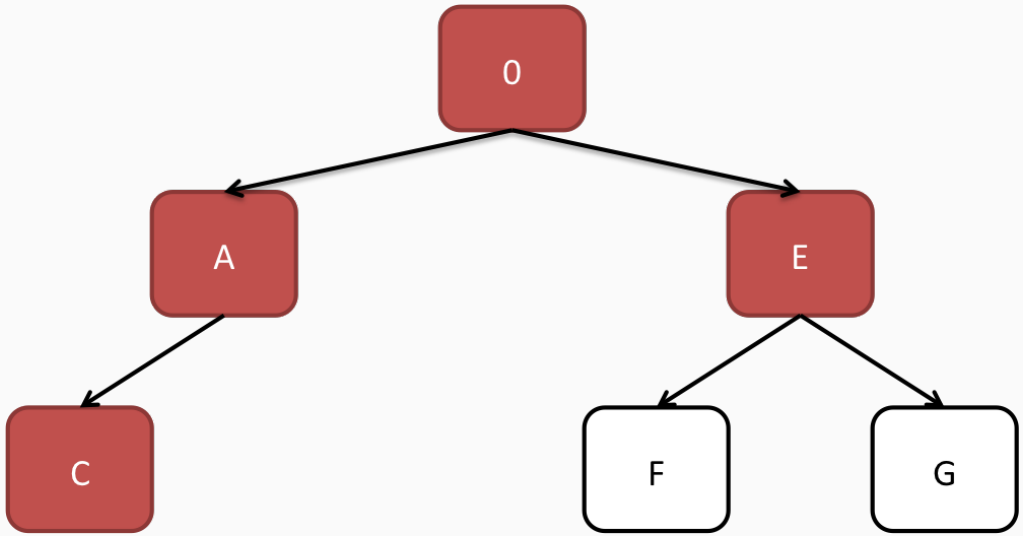
Affichage d'un arbre binaire

- Pas de fonction en Java pour afficher simplement un arbre
- Pour afficher un arbre $\Rightarrow$ parcours préfixe
  - exploration en profondeur de chaque chemin avant de revenir en arrière
- Illustration



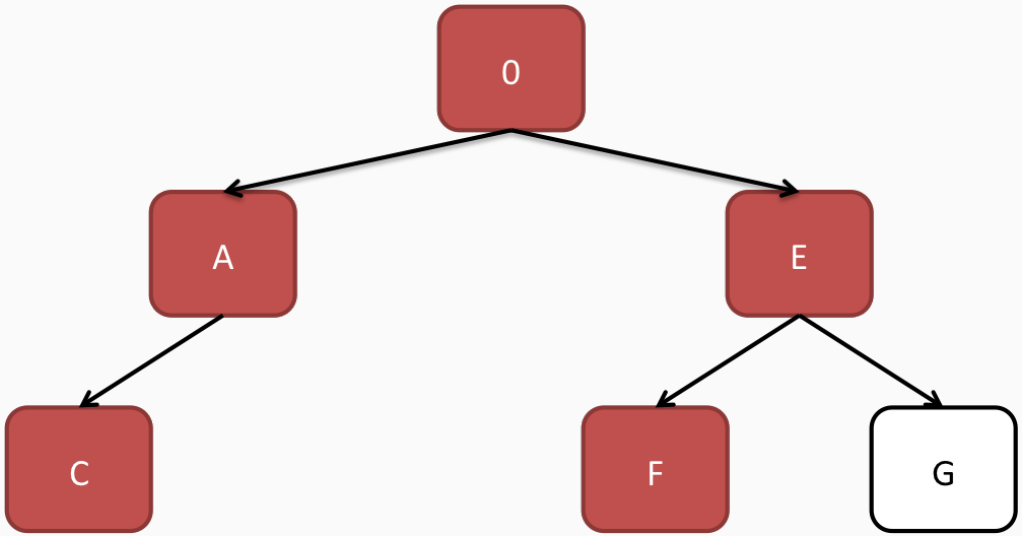
Affichage d'un arbre binaire

- Pas de fonction en Java pour afficher simplement un arbre
- Pour afficher un arbre $\Rightarrow$ parcours préfixe
  - exploration en profondeur de chaque chemin avant de revenir en arrière
- Illustration



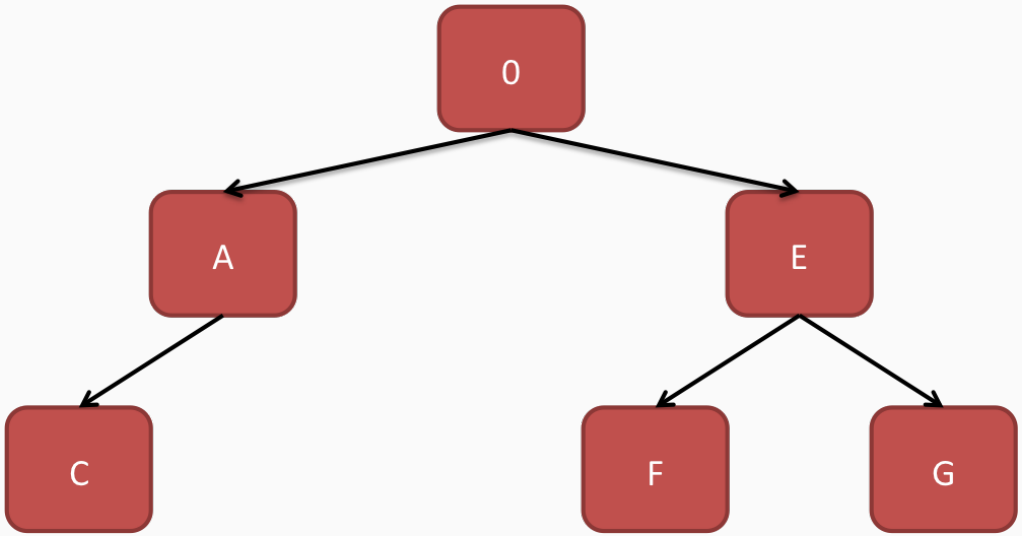
Affichage d'un arbre binaire

- Pas de fonction en Java pour afficher simplement un arbre
- Pour afficher un arbre $\Rightarrow$ parcours préfixe
  - exploration en profondeur de chaque chemin avant de revenir en arrière
- Illustration



Affichage d'un arbre binaire

- Pas de fonction en Java pour afficher simplement un arbre
- Pour afficher un arbre $\Rightarrow$ parcours préfixe
  - exploration en profondeur de chaque chemin avant de revenir en arrière
- Illustration



Affichage d'un arbre binaire (1)

- On souhaite obtenir un affichage comme suit

```
O
|-->A
    |-->C
    |-->[X]
|-->E
    |-->F
    |-->G
```

- Difficultés
  - implémenter le parcours préfixe (en profondeur)
  - afficher chaque nœud avec le bon décalage par rapport à la gauche
- Solution
  - fonction de parcours récursive qui met à jour la profondeur pour adapter l'affichage

Affichage d'un arbre binaire (2)

- Parcours préfixe avec passage de la profondeur en argument pour adapter le décalage dans la présentation

```
private String aff(int level){
  if (isLeaf()) return shift(level) + getLabel();
  else {
    String left = (hasLeft())? getLeft().aff(level + 1): shift↩
        (level+1) + "[X]";
    String right = (hasRight())? getRight().aff(level + 1): ↩
        shift(level+1) + "[X]";
    return shift(level) + getLabel() + "\n" + left + "\n" + ↩
        right;
  }
}
```

Affichage d'un arbre binaire (3)

- Fonction de décalage de l'écriture en fonction de la profondeur

```
private String shift(int shift){
  if (shift == 0) return "";
  StringBuilder buffer = new StringBuilder("");
  for (int i = 0; i < shift - 1; i++) buffer.append("    "↩
      );
  buffer.append("|-->");
  return buffer.toString();
}
```

- Fonction d'affichage de l'arbre binaire avec initialisation de la profondeur à 0

```
public String toString(){
  return aff(0);
}
```