Algorithmique et Programmation

Jean-Christophe Engel

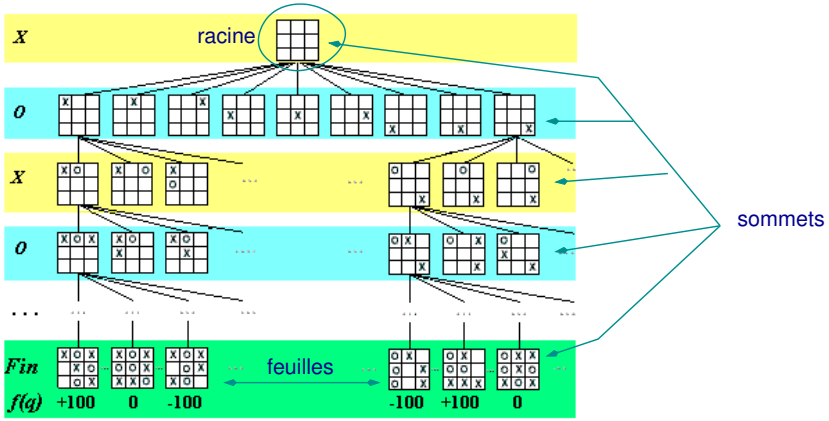
École supérieure d'ingénieurs de Rennes
Université de Rennes 1

Plan

1. Introduction
2. Type abstrait
3. Généricité
4. Structures de données (le retour)
   - Arbres binaires
5. Héritage

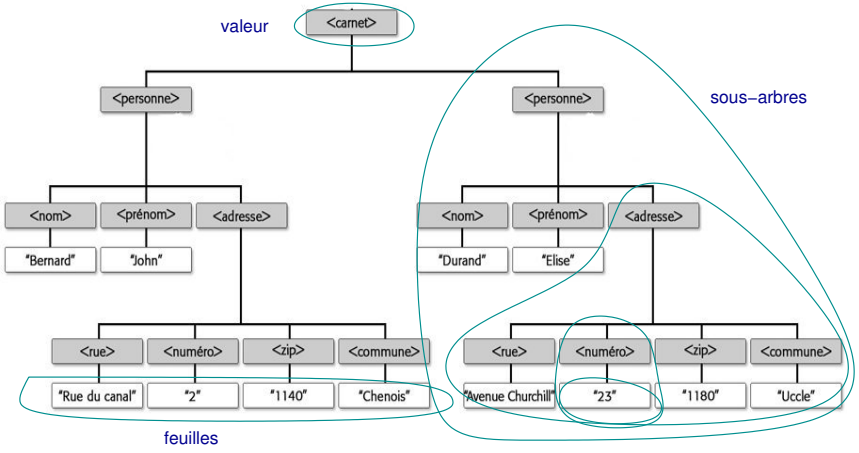
Arbres

- Arbre de décision dans le jeu du morpion

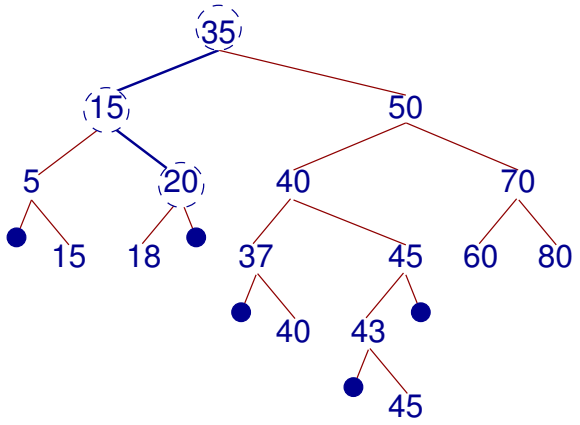


Arbres

- Arbre associé à un document XML

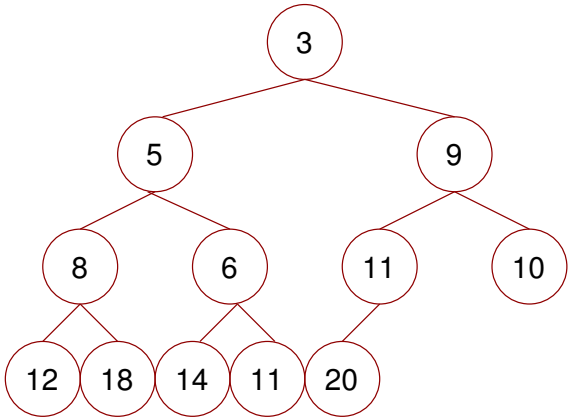


arbre binaire de recherche : TreeSet, TreeMap



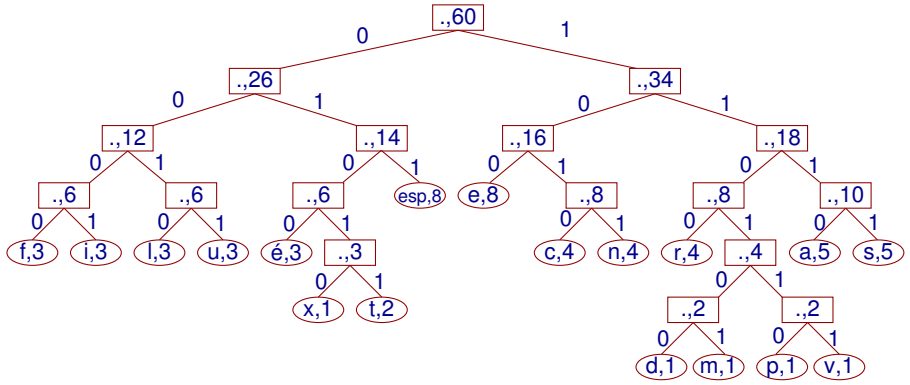
- relation d'ordre
- valeurs du fils gauche inférieures racine
- valeurs du fils droit supérieures racine
- Coût opérations : $O(\log_2(n))$

arbre parfait partiellement ordonné : tri par tas



- relation d'ordre
- racine inférieure aux éléments des fils
- Coût opérations : $O(\log_2(n))$

arbre de Huffman : encodage d'information



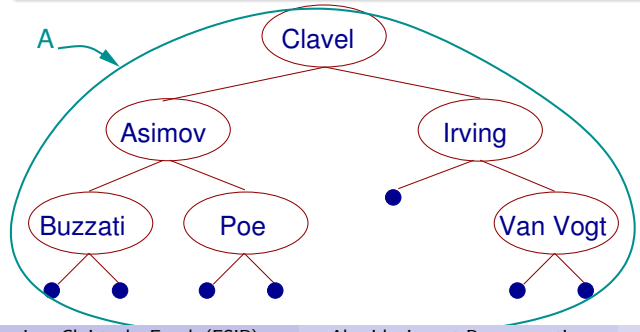
- proche de la racine : octet le plus fréquent
- loin de la racine : octets les moins fréquents

Arbre binaire

Définition

Un arbre binaire est

- soit vide
- soit non vide et possède
  - un élément (valeur)
  - deux fils qui sont des arbres binaires



Arbre binaire : spécification

- Opérations de consultation

```
public boolean estVide();
public T getValeur();                    // @pre arbre non vide
public boolean estFeuille();             // @pre arbre non vide
public boolean existeGauche();           // @pre arbre non vide
public boolean existeDroit();            // @pre arbre non vide
```

- Opérations de parcours

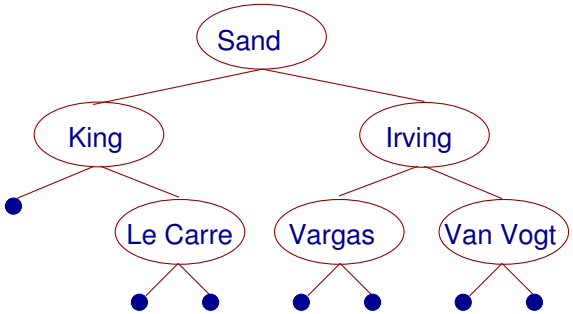
```
public ArbreBinaire<T> filsGauche();   // @pre arbre non vide
public ArbreBinaire<T> filsDroit();    // @pre arbre non vide
```

- Opérations de modification

```
public void setValeur(T x);
public void setArbre(ArbreBinaire<T> a);
public void setGauche(ArbreBinaire<T> a); // @pre arbre non vide
public void setDroit(ArbreBinaire<T> a);  // @pre arbre non vide
public void supprimer();
```


Arbre binaire : stratégies de parcours

en largeur : à partir de la racine descendre « niveau par niveau »

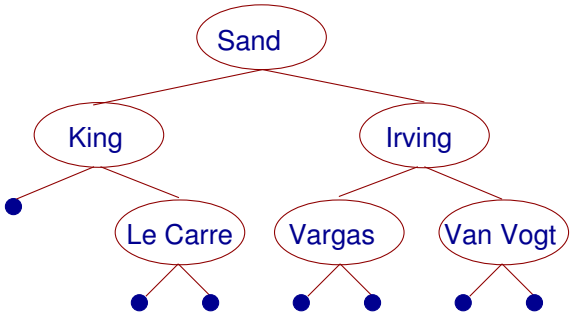


Sand, King, Irving, Le Carre, Vargas, Van Vogt

⇒ parcours itératif avec file d'attente

Arbre binaire : stratégies de parcours

en profondeur : à partir de la racine, parcourir tout le fils gauche puis tout le fils droit

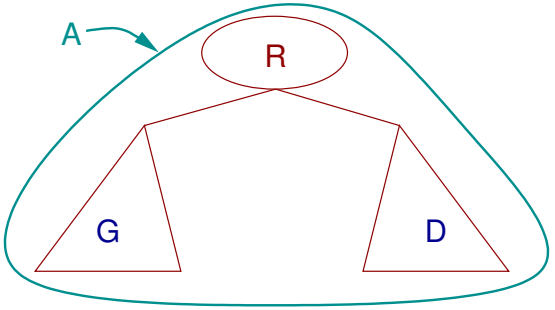


préfixe (RGD) : Sand, King, Le Carre, Irving, Vargas, Van Vogt
infixe (GRD) : King, Le Carre, Sand, Vargas, Irving, Van Vogt
postfixe (GDR) : Le Carre, King, Vargas, Van Vogt, Irving, Sand

Arbre binaire et récursivité

principe général : cas d'un arbre non vide

$$tRecursif(A) = tValeur(R) \oplus tRecursif(G) \otimes tRecursif(D)$$



fonction récursive : le paramètre donne la taille du problème

- le fils gauche de A est « plus petit » que A
- le fils droit de A est « plus petit » que A

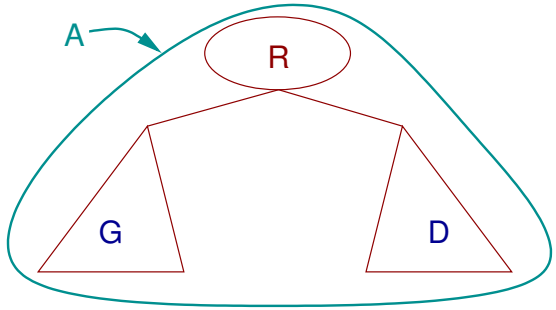
Arbre binaire et récursivité

appel récursif

- descente dans l'arbre (fils gauche ou fils droit)
- paramètres = transmission de valeur de racine vers feuilles

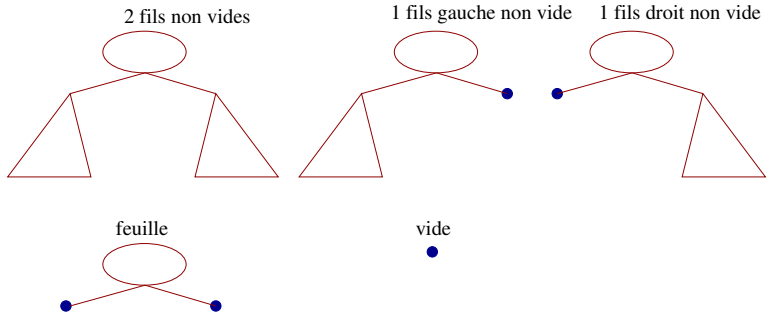
retour appel

- remontée au père
- résultat = transmission valeur des feuilles vers racine



Arbre binaire et récursivité

formes d'arbres



Arbre binaire et récursivité

nombre d'éléments d'un arbre binaire

$$tRecursif(A) = tValeur(R) \oplus tRecursif(G) \otimes tRecursif(D)$$

pour tout arbre non vide :

nombre d'éléments d'un arbre =
1 +
nombre d'éléments du fils gauche +
nombre d'éléments du fils droit

arbre vide :
nombre d'éléments = 0

Arbre binaire et récursivité

nombre d'éléments d'un arbre binaire

$$tRecursif(A) = tValeur(R) \oplus tRecursif(G) \otimes tRecursif(D)$$

```
static <T>
int
nombreElements(ArbreBinaire<T> a)
{
  if (a.estVide()) { return 0; }
  else             { return 1
                          +
                          nombreElements(a.filsGauche())
                          +
                          nombreElements(a.filsDroit());
  }
}
```