INF3105 – Arbres

Jaël Champagne Gareau

Université du Québec à Montréal (UQAM)

Été 2024

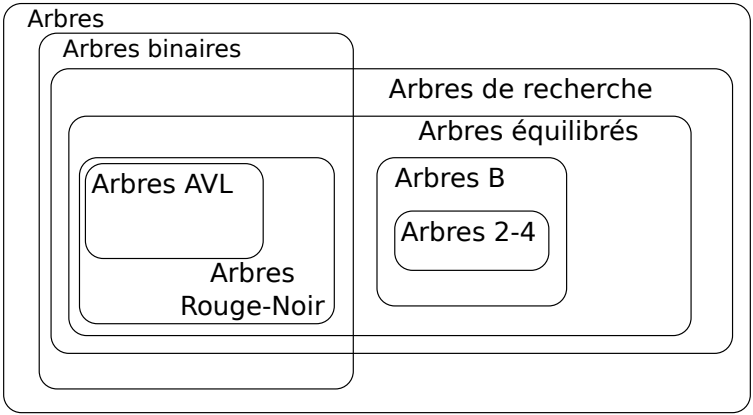
http://cria2.uqam.ca/INF3105/

UQÀM

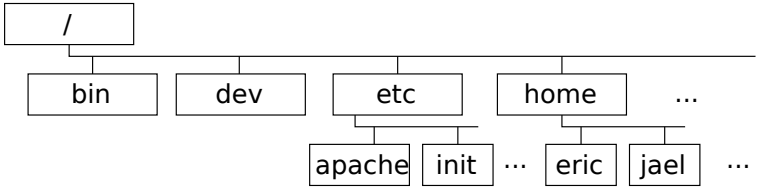
Les arbres

- Structure de données non linéaire.
- Analogie avec arbres dans la nature (plantes).
- Catégories d'applications :
  - Représentation de structures arborescentes.
  - **Recherche efficace d'information** (thème important en INF3105).

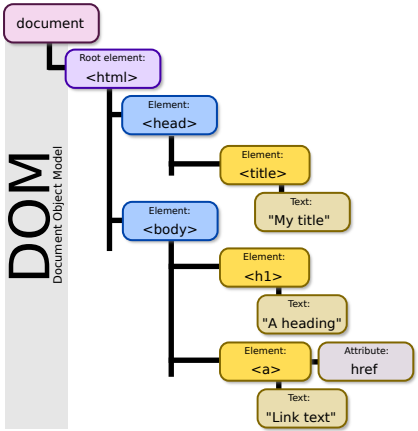
Arbres étudiés en INF3105

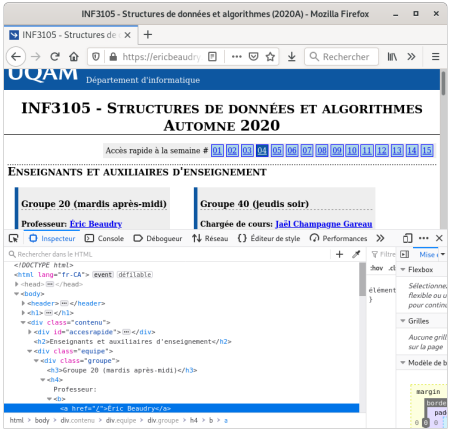


Système de fichiers



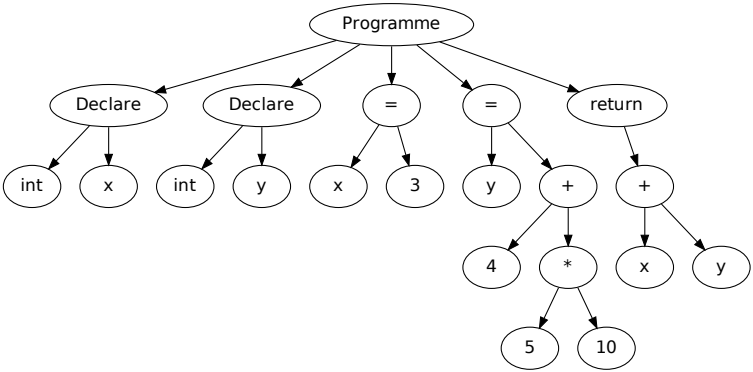
HTML / Arbre DOM (Document Object Model)



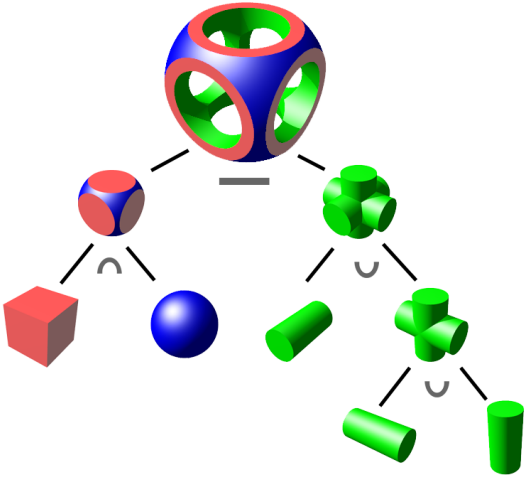


Arbre syntaxique abstrait (*Abstract Syntax Tree*)

```
1  int main() {
2    int x, y;
3    x = 3;
4    y = 4 + 5 * 10;
5    return x + y;
6  }
```

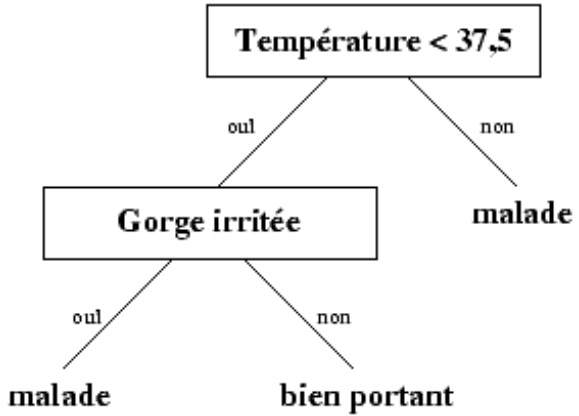


Scène en infographie



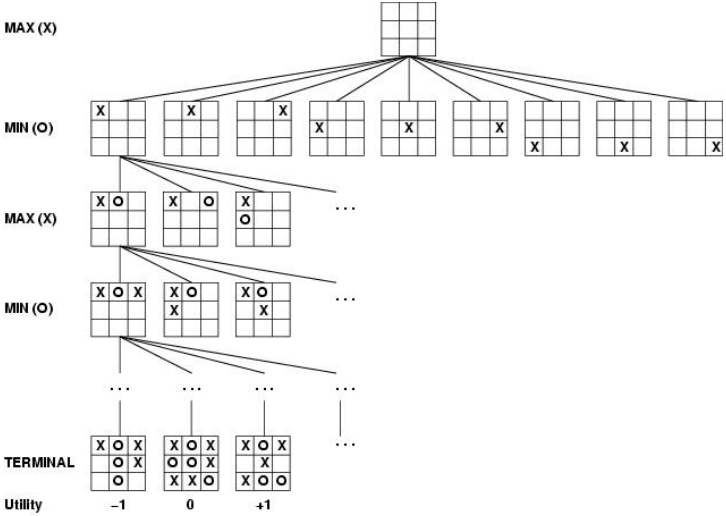
Source image : http://upload.wikimedia.org/wikipedia/commons/8/8b/Csg_tree.png

Arbre de décision



Source image : http://www.grappa.univ-lille3.fr/polys/apprentissage/sortie004.html

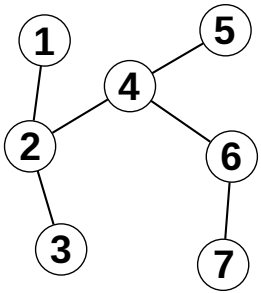
Arbre de recherche (dans un jeu)



Définitions de base

Arbre (*tree*)

Un **arbre** est un ensemble de **nœuds** connexes liés par des arêtes où il n'y a pas de cycles. Pour un arbre de n nœuds, il y a $n-1$ arêtes.



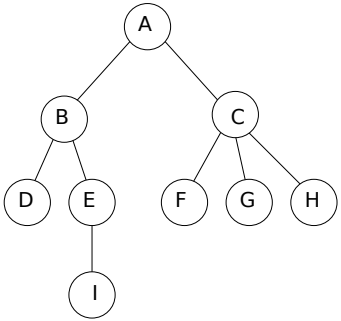
Définitions de base

Arbre (enraciné)

Un **arbre enraciné** est un arbre où l'un des nœuds est désigné comme étant la racine.

Nœud (*node*)

Un **nœud** dans un arbre sert à stocker une valeur ou un objet (des données).



Définitions de base

Racine (*root*)

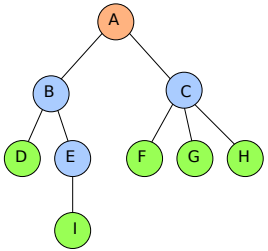
La **racine** d'un arbre est l'unique nœud n'ayant pas de parent. Un arbre vide n'a pas de racine. Tous les nœuds sont accessibles à partir de la racine.

Feuille (*leaf*)

Une **feuille** est un nœud sans enfants. Aussi appelée nœud extérieur (*external node*).

Nœud intérieur (*internal node*)

Un **nœud intérieur** est un nœud qui n'est pas une feuille. La racine peut être considérée comme un nœud intérieur.



Racine : A
Feuilles : $\{D, I, F, G, H\}$
Nœuds intérieurs : $\{B, E, C\}$
ou $\{A, B, E, C\}$ si on considère la racine comme un nœud intérieur.

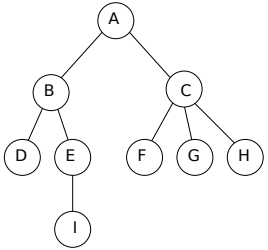
Relations

Parent / enfants

Un nœud **parent** peut avoir plusieurs nœuds **enfants** (*children*).

Frères (*siblings*)

Un nœud **frère** (*sibling*) d'un nœud est un autre nœud ayant le même nœud parent.



parent(A) = `indéfini`.
parent(C) = A
enfants(A) = $\{B, C\}$
enfants(D) = $\{\}$
frères(G) = $\{F, H\}$
frères(I) = $\{\}$

Relations

Ancêtres

Un nœud **ancêtre** d'un nœud est accessible à travers des relations parents :

$$x = \text{parent}(n) \Rightarrow x \in \text{ancetres}(n);$$

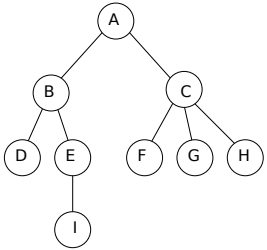
$$x \in \text{ancetres}(n) \text{ et } n = \text{parent}(n_1) \Rightarrow x \in \text{ancetres}(n_1).$$

Descendants

Un nœud **descendant** d'un nœud est accessible à travers des relations enfants :

$$x \in \text{enfants}(n) \Rightarrow x \in \text{descendants}(n);$$

$$x \in \text{descendants}(n) \text{ et } n \in \text{enfants}(n_1) \Rightarrow x \in \text{descendants}(n_1).$$

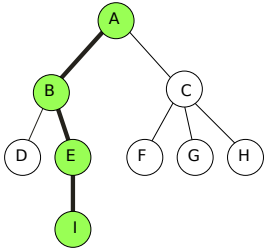


ancêtres(I) = $\{E, B, A\}$
ancêtres(A) = $\{\}$
descendants(C) = $\{F, G, H\}$
descendants(G) = $\{\}$

Hauteur

Définition retenue

La **hauteur** d'un arbre est la longueur (nombre d'arêtes) du plus long chemin dans l'arbre de la racine vers une feuille. Sous cette définition, un arbre vide a une hauteur de -1 et un arbre avec un seul nœud (la racine) a une hauteur de 0.



hauteur = 3

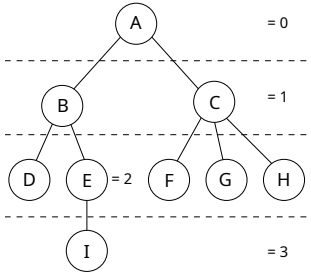
Profondeur

Profondeur d'un nœud

La **profondeur** d'un nœud est la longueur du chemin qui le relie avec la racine.

Profondeur $\neq$ Hauteur

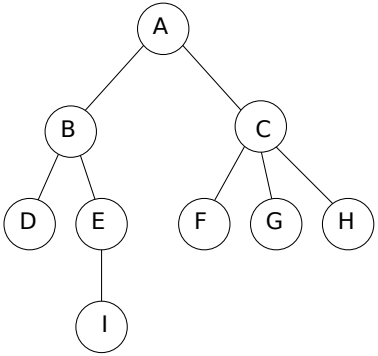
Bien que la profondeur et la hauteur soient des notions reliées, il faut faire attention à ne pas les confondre. La notion de hauteur s'applique à un arbre (incluant un sous-arbre). La notion de profondeur s'applique à un nœud dans un arbre.

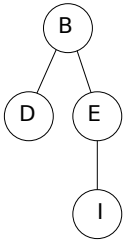


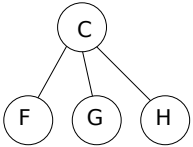
Définitions

Sous-arbre (*subtree*)

Un **sous-arbre** d'un arbre *a* est l'arbre engendré à partir d'un nœud enfant de *a*.



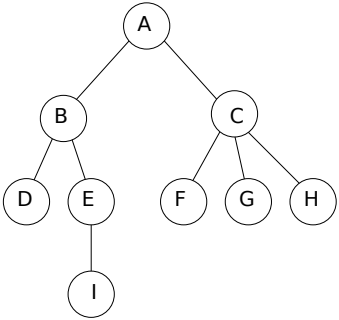


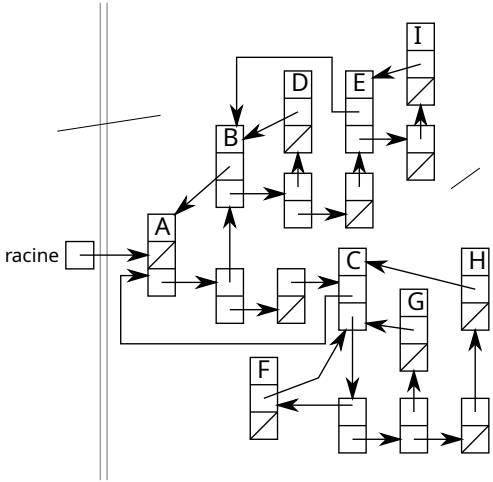


Classe Arbre en C++

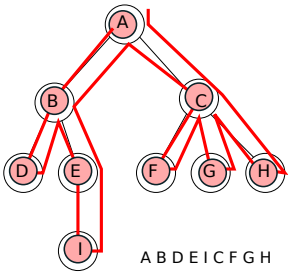
```
1  template <class T>
2  class Arbre {
3    public:
4      // ...
5    private:
6      struct Noeud {
7          T contenu;
8          Noeud* parent;
9          Liste<Noeud*> enfants;
10     };
11     Noeud* racine;
12 };
```

Représentation

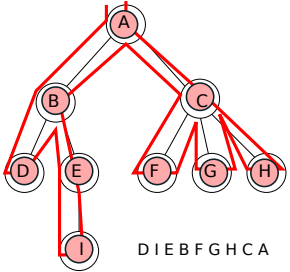




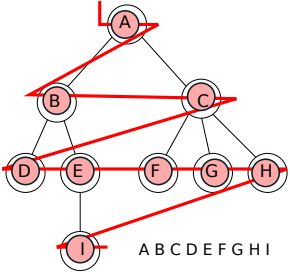
Types de parcours d'un arbre



Préordre



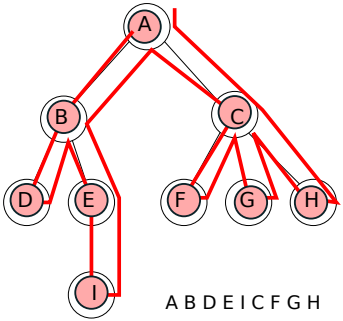
Postordre



Largeur

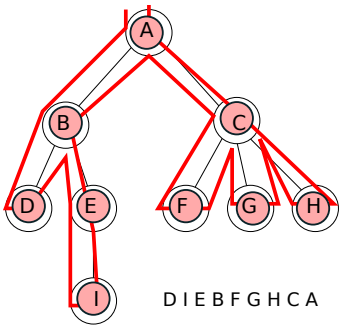
Parcours en préordre

```
1  template <class T>
2  void Arbre<T>::afficherPreOrdre() const {
3      afficherPreOrdre(racine);
4  }
5
6  template <class T>
7  void Arbre<T>::afficherPreOrdre(Noeud* n) const {
8      if(n == nullptr) return;
9      std::cout << n->contenu << " ";
10     Liste<Noeud*>::Iterateur iter = n->enfants.debut();
11     while(iter)
12         afficherPreOrdre(*iter++);
13 }
```



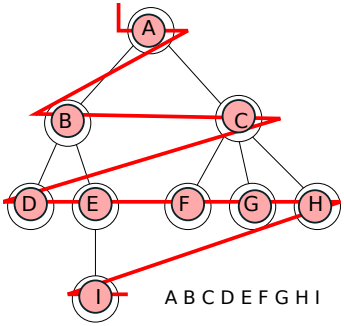
En postordre

```
1  template <class T>
2  void Arbre<T>::afficherPostOrdre() const {
3      afficherPostOrdre(racine);
4  }
5
6  template <class T>
7  void Arbre<T>::afficherPostOrdre(Noeud* n) const {
8      if(n == nullptr) return;
9      Liste<Noeud*>::Iterateur iter = n->enfants.debut();
10     while(iter)
11         afficherPostOrdre(*iter++);
12     std::cout << n->contenu << " ";
13 }
```



En largueur

```
1  template <class T>
2  void Arbre<T>::afficherLargeur() {
3      File<Noeud*> noeuds_a_visiter;
4      if(racine != nullptr) noeuds_a_visiter.enfiler(racine);
5      while(!noeuds_a_visiter.estVide()) {
6          Noeud* n = noeuds_a_visiter.defiler();
7          std::cout << n->contenu << " ";
8          Liste<Noeud*>::Iterateur iter = n->enfants.debut();
9          while(iter)
10             noeuds_a_visiter.enfiler(*iter++);
11     }
12 }
```



arbre.h

```
1  template <class T>
2  class Arbre {
3    public:
4      int hauteur() const;
5    private:
6      struct Noeud {
7          T contenu;
8          Noeud* parent;
9          Liste<Noeud*> enfants;
10     };
11     Noeud* racine
12 };
```

arbre.cpp

```
1  template <class T>
2  int Arbre<T>::hauteur() const {
3
4
5
6
7
8  }
```

Analyse

Complexité temporelle ?

Pire cas ? Cas Moyen ?