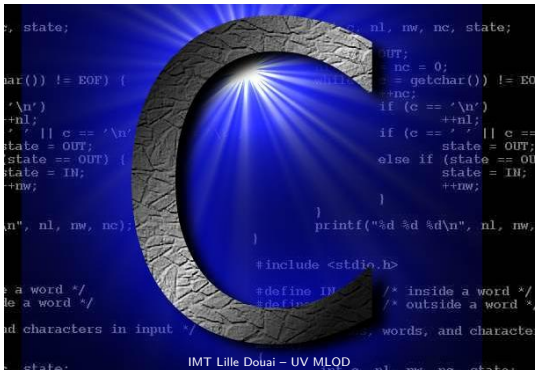
Algorithmique et langage C
— Structures de données dynamiques avancées —



Luc Fabresse
luc.fabresse@imt-nord-europe.fr

version 1.2

Plan

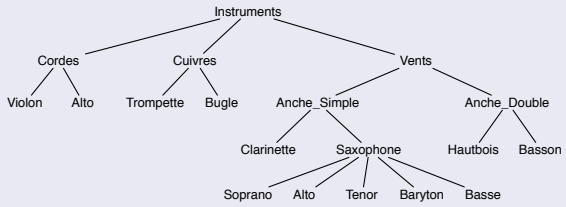
Plan

Notion d'arbre (1)

Définition

- Collection d'informations homogènes organisée en niveaux
- Chaque information d'un niveau donnée peut être reliée à plusieurs informations du niveau inférieur par des branches
- Les branches ne forment pas de cycles

Exemple



Notion d'arbre (2)

Caractéristiques

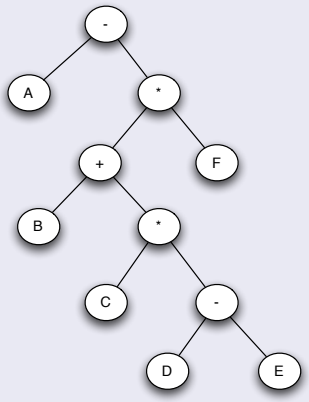
- Un arbre d'éléments de type T est soit vide, soit formé d'une donné de type T appelée **racine** et d'un ensemble fini de taille variable d'arbres de type T appelés sous-arbres
- On appelle **feuille**, la racine d'un arbre n'ayant pas lui-même de sous arbres
- On appelle **noeud**, la racine de tout sous-arbre, la racine et les feuilles étant des nœuds particuliers
- Un sous-arbre d'un arbre est appelé son **fils** et inversement celui-ci est appelé **père** de celui-là
- Deux sous-arbres du même arbre sont des **frères**, et leurs racines respectives des nœuds frères

Définition

- Chaque nœud d'un arbre binaire est la racine d'au plus deux sous-arbres appelés respectivement *sous-arbre gauche* et *sous-arbre droit*
- Les arbres binaires sont un cas particulier des arbres généraux (*n-aire*)

Exemple : représentation d'une expression arithmétique

L'expression `A-(B+C*(D-E))*F` se représente par :



Vocabulaire

- Racine
- Nœud
- Feuille
- Nœud père
- Nœud fils
- Nœud frère
- Sous-arbre gauche (SAG)
- Sous-arbre droit (SAD)
- Ancêtres
- Descendants
- Branches / Chemins
- Hauteur
- Profondeur

Plan

Arbres binaires

Représentation :

Type enregistrement à trois champs :

- `val`
- `filsGauche`
- `filsDroit`

Algorithmes de parcours :

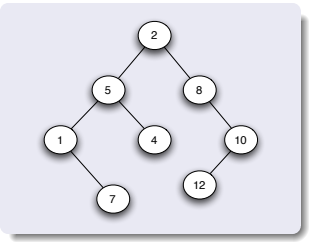
Permet d'accéder une fois et une seule à tous les nœuds de l'arbre. Les parcours les plus utilisés :

- en pré-ordre / préfixé
- en post-ordre / post-fixé
- en ordre / infixe

Arbres binaires : parcours en pré-ordre

Idée

Inspecter la racine, puis parcourir en pré-ordre le SAG (resp. le SAD), et enfin parcourir en pré-ordre le SAD (resp. le SAG)




```
parcoursRGD(a: ArbreBinaire)
begin
    if nonVide(a) then
        affiche(valeur(a));
        parcoursRGD(filsGauche(a));
        parcoursRGD(filsDroit(a));
    end
end
```

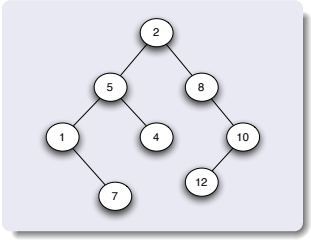
Résultat du parcours :

- Parcours en pré-ordre RACINE, SAG, SAD : 2 5 1 7 4 8 10 12
- Parcours en pré-ordre RACINE, SAD, SAG : 2 8 10 12 5 4 1 7

Arbres binaires : parcours en post-ordre

Idée

Parcourir en post-ordre le SAG (resp. le SAD), puis le SAD (resp. le SAG) et enfin la racine




```
parcoursGDR(a: ArbreBinaire)
begin
    if nonVide(a) then
        parcoursGDR(filsGauche(a));
        parcoursGDR(filsDroit(a));
        affiche(valeur(a));
    end
end
```

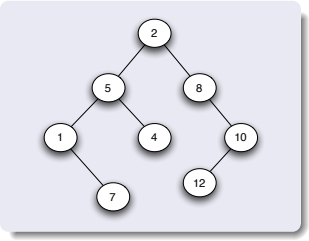
Résultat du parcours :

- Parcours en post-ordre SAG, SAD, RACINE : 7 1 4 5 12 10 8 2
- Parcours en post-ordre SAD, SAG, RACINE : 12 10 8 7 1 4 5 2

Arbres binaires : parcours en ordre

Idée

Parcourir en ordre le SAG (resp. le SAD), puis la racine et enfin le SAD (resp. le SAG)




```
parcoursGRD(a: ArbreBinaire)
begin
    if nonVide(a) then
        parcoursGRD(filsGauche(a));
        affiche(valeur(a));
        parcoursGRD(filsDroit(a));
    end
end
```

Résultat du parcours :

- Parcours en ordre SAG, RACINE, SAD : 1 7 5 4 2 8 12 10
- Parcours en ordre SAD, RACINE, SAG : 10 12 8 2 4 5 7 1

Arbres binaires équilibrés

Définition

Un arbre binaire qui maintient une profondeur équilibrée entre ses branches i.e. la difference de hauteur entre son SAG et SAD est -1, 0 ou 1.

Pros/Cons

- Avantage : temps d'accès moyen aux données est en minimisé (hauteur de l'arbre). Contrairement aux arbres en peigne (a.k.a une liste).
- Inconvénients :
  - maintenir l'équilibrage au fil des insertions/suppressions
  - nécessité de ré-équilibrer (e.g. rotation)

Arbres binaires

Arbre binaire complet

Un arbre binaire complet est un arbre binaire tel que chaque niveau de l'arbre est complètement rempli.

Arbre binaire complet de hauteur h :

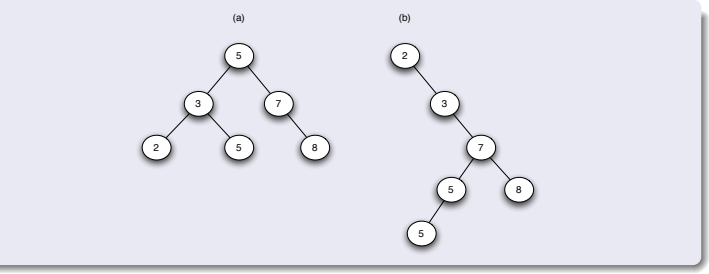
- $2^h - 1$ nœuds
- 2^{h-1} feuilles

Plan

Arbres binaires de recherche (ABR)

Définition

Un arbre binaire est ordonné horizontalement (de gauche à droite) si la valeur (clé) de tout nœud non feuille est supérieure à toutes celles de son sous-arbre gauche et inférieur à toutes celles de son sous-arbre droit.



Arbres binaires de recherche : algorithmes

```
recherche_r(racine,v): Arbre
begin
    if estVide(racine) ou (v=valeur(racine)) then
        return racine;
    else
        if v<valeur(racine) then
            return recherche(filsGauche(racine),v);
        else
            return recherche(filsDroit(racine),v);
        end
    end
end
```

```
recherche_i(racine,v): Arbre
begin
    while nonVide(racine) et v !=valeur(racine) do
        if v<valeur(racine) then
            racine ← filsGauche(racine);
        else
            racine ← filsDroit(racine);
        end
    end
    return racine;
end
```


Insertion d'un élément

On cherche de façon récursive la place de l'élément et on l'insère dans l'arbre. L'élément ajouté sera obligatoirement une feuille.

Suppression d'un élément

- Si l'élément est une feuille, alors on le supprime simplement
- Si l'élément n'a qu'un descendant, alors on le remplace par ce descendant
- Si l'élément a deux descendants, on le remplace au choix, soit par l'élément le plus à droite du SAG, soit par l'élément le plus à gauche du SAD, afin de conserver la propriété d'ordre horizontal

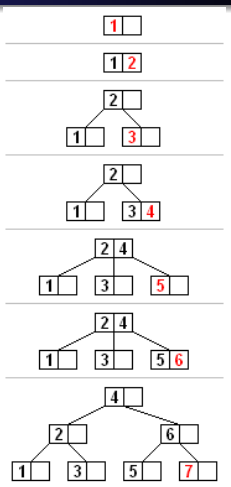
Plan

B-arbre

Exemple : chaque nœud contient 2 valeurs au plus et peut avoir jusqu'à 3 fils.

on insère les entiers de 1 à 7

source : http://fr.wikipedia.org/wiki/Arbre_B



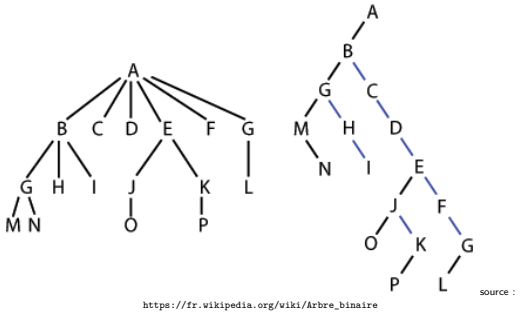
Autres Variantes

- Tas (Heap)
- Arbres Rouge et Noir
- ...

Arbres *n-aire*

Représentation sous forme d'arbre binaire d'un arbre `n-aire`

- `filsGauche` est le premier fils dans l'arbre `n-aire`
- `filsDroit` est un frère dans l'arbre `n-aire`



source : https://fr.wikipedia.org/wiki/Arbre_binaire

Exercice

Compléter le fichier `arbre.c`

Plan

Bilan

- Structures de données statiques (tableau, enregistrements,...)
- Structures de données dynamiques (liste, arbre, ...)
- Tri
- Parcours
- ...

Aspects non abordés

- graphes
- complexité
- ...

Avantages

- Langage structuré et évolué qui permet néanmoins d'effectuer des opérations de bas niveau
- Proche de la machine
- Indépendant de la machine contrairement à l'assembleur
- Bonne portabilité
- De nombreuses bibliothèques
- Code exécutable, rapide, compact, ...
- Père syntaxique de nombreux langages : C++, Java, PHP, ...
- Répandu (Unix, Windows, ...)

- Trop permissif (peu de protection ou de contrôle)
- Pas de gestions d'exceptions
- Programmation spaghetti et autres
- Programmes difficiles à maintenir si :
  - La programmation est trop compacte
  - Les commentaires sont insuffisants
- Portabilité réduite si :
  - non respect des normes
  - utilise des bibliothèques spécifiques
- Nécessite une grande rigueur de programmation
- Langage pauvre : pas d'instructions pour la manipulation de chaînes, de tableaux, de listes chaînées => solution ad-hoc

- Pré-processeur
- Compilation conditionnelle
- Macro-instructions
- Fonctions à nombre de paramètres variables (comme `printf` et `scanf`)
- Construction de bibliothèques ("librairies")
- Fonctions du second ordre (prenant en paramètres d'autres fonctions)
- Utilisation du débogueur gdb

Fin pour programmation en C,
mais ce n'est que le début de l'apprentissage de la programmation...