ET3 INFO Polytech Paris-Saclay
Algorithmique, complexité et graphes

Chapitre 0 : Déroulement du cours et Rappels du premier semestre

Emmanuelle Frenoux, Brigitte Rozoy, Lonni Besançon
Emmanuelle.Frenoux@lisn.fr
Emmanuelle.Frenoux@universite-paris-saclay.fr

Déroulement du cours

- Chargés de TD
  - Marie LAVEAU
  - Laurent ROSAZ (2 séances environ)
  - prenom.nom@universite-paris-saclay.fr
- Notation
  - Une note de TD (1 ou 2 exercices notés)
  - Une note d'exam « terminal »

Retour sur les premiers cours… 1/2

- Qu’est-ce qu’un algorithme ?
- Spécification informelle
- Comment écrit-on un algorithme ?
- Paramètres entrée, sortie, entrée/sortie
- Passage par valeur/par adresse (ou référence)
- Récursivité, récursification/dérécursification

Retour sur les premiers cours… 2/2

- Notion de complexité d'un algorithme
  - Temps
  - Espace
- Comment évaluer la complexité en temps
  - Que prend on en compte ?
  - Comment fait-on les calculs ?
  - Comment diminue-t-on la complexité ?

Retour sur les arbres

- Les arbres, un cas particulier des graphes
  - Dans un graphe (math), les nœuds sont des sommets et les liens sont appelés arêtes (ou arcs)
  - Un graphe est un arbre sans racine, dont les arêtes sont orientées ou non et éventuellement avec des cycles.
  - Si l'on contraint le graphe, on obtient un arbre :
    - Distinguer un sommet => racine
    - Absence de cycle => arbre
    - Arêtes implicitement orientées

Retours sur les arbres

- Un arbre est un ensemble de places (nœuds), organisé de manière hiérarchique à partir de son nœud initial (racine).
- Cette année : Arbre Binaire (décorés/étiquetés)
  - les nœuds ont au plus 2 successeurs (fils) => binaire
  - les nœuds portent une valeur => décoré/étiqueté
- Les arbres sont utilisés pour représenter de multiples choses (relations d'espace, de précédence, de temporalité, etc…)
- Forme algorithmique récursive (en implémentation et en exploitation) :

Arbre = Ø

Racine = …

Arbre = <Racine, Arbre, Arbre>

Retours sur les arbres

- Pour un arbre non vide :
  - Racine est sa racine
  - Fils :
    - **sous-arbre** gauche
    - **sous-arbre** droit
- Si le sous-arbre gauche est non-vide, sa racine est un fils gauche
- Si le sous-arbre droit est non-vide, sa racine est un fils droit
- Le fils gauche et le fils droit sont dits frères ayant pour père la racine initiale
- Les racines des sous-arbres sont appelées des nœuds et ainsi de suite…

Définitions

- Taille de l'arbre :
  nombre de nœuds internes ou externes
  (taille des données contenues dans l'arbre)
- Niveau (profondeur) d'un nœud :
  longueur du chemin le reliant à la racine
- Hauteur de l'arbre (hauteur != taille) :
  le plus grand des niveaux de ses nœuds
- Nœud externe ou feuille :
  nœud dont les fils sont vides
- Nœud interne :
  nœud avec 1 ou plusieurs fils non vides

- En informatique, on utilise des arbres dont les nœuds ont un nombre variable de fils, et éventuellement aucun contenu.

Retour sur les arbres

Type abstrait : Arbre
Utilise : Nœuds, Elément

Fonctionnalités de base :

- **Arbre vide** (arbre sans racines ni nœud) :
  vide?(A) ou "A=vide (ou NULL)"
- **Racine**
  - Place du haut
  - Contenu(Racine) : fournit la valeur contenue ici
- **Fils d'un nœud** (sous-arbre) :
  - fonction gauche (resp. droit)/A.droite/A.gauche/d(A)/g(A)
- **Créer un arbre** : faire_arbre(e, B, C) construit un arbre dont la racine contient e, le fils gauche B et le fils droit C
- **Savoir si un arbre est réduit à une feuille** : feuille?(A) (vérifie un fils puis l'autre)
- **Changer la valeur à la racine** : remplacer_racine(x, var A) (Attention, on modifie le paramètre A de type Arbre)

Retour sur les arbres

Représentation chaînée des arbres

- Représentation la plus naturelle (transcription directe des notions de nœud et de fils)
- Un arbre est un pointeur vers un groupement contenant la valeur de la racine et deux pointeurs vers les fils.

Façon JAVA :

```
Arbre = classe{
Champs
    val : élément
    gauche, droite : Arbre
Méthodes
    contenu_racine() : élément
    Vide?() : booléen
}
```

Façon C :

```
struct node{
    int key_value ;
    struct node *left ;
    struct node *right;
}
struct node* root ;
```

Création :

A = New(Arbre) (réserve la place et met son adresse dans A)

Retour sur les arbres

- Fonctionnalités…
  - arbre vide :
    - Faire_vide : A <- NULL crée un arbre vide
    - vide?(A) : teste un arbre
  - Racine : Place_Racine (A)
  - Contenu : Contenu(N) et Contenu_racine(A)
  - Sous-arbres : gauche(A) et droite(A)
  - Nouvel arbre : faire_arbre(A, x, A')

NB : permettent de manipuler les arbres indépendamment de l'implantation

Retour sur les arbres

Parcours des arbres => accès aux données

- Parcours en profondeur, à main gauche : depuis la racine, descendre successivement dans le sous-arbre gauche puis dans le droit, récursivement (inclure le traitement).
  - Ordre préfixe : on traite chaque sommet la première fois qu'on le rencontre
  - Ordre infixe/symétrique : on traite chaque sommet ayant un fils gauche la seconde fois qu'on le voit et chaque sommet sans fils gauche la première fois qu'on le voit
  - Ordre postfixe/suffixe : on traite chaque sommet la dernière fois qu'on le rencontre
- Parcours en largeur (Breadth First Search (BFS)) : par niveaux, en lisant les nœuds de gauche à droite.
  - Utilisation d'une file
  - Déroulement
    - Le niveau 0 donne la racine
    - A chaque étape, mettre dans la file la liste ordonnée des nœuds de niveau k
    - Prélever les nœuds un à un en les traitant
    - Mettre en queue dans la file le sous-arbre gauche, puis le droit