Généralités sur les Arbres

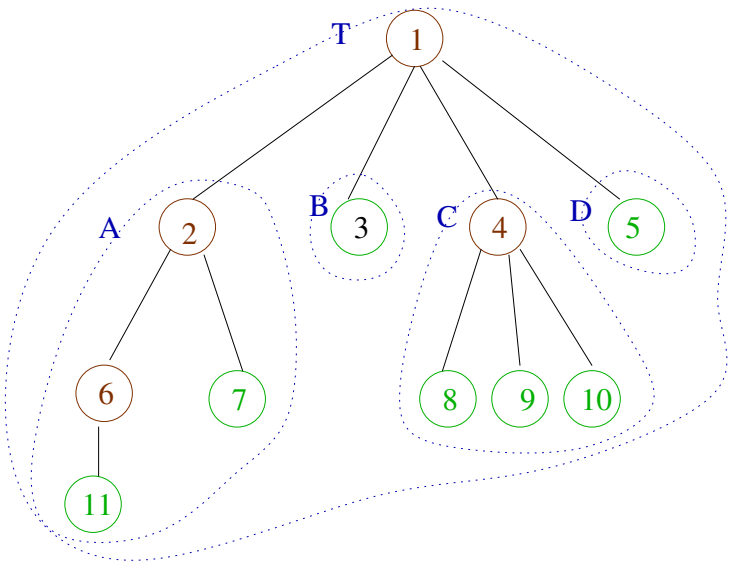
Un arbre est une structure de donnée non linéaire qui induit une organisation hiérarchique des données, présente partout en informatique (bases de données, systèmes de fichiers, site web, analyse syntaxique etc). L'utilisation d'une telle structure de données permet parfois d'obtenir des algorithmes plus efficaces qu'avec des structures linéaires.

1 Définition récursive

Un arbre est soit :
- vide
- composé d'un nombre fini de noeuds, chaque noeud comportant :
  - une valeur
  - un ensemble fini d'arbres successeurs

2 Vocabulaire

- un noeud possède au plus un noeud prédécesseur : son **père**
- la **racine** est le seul noeud qui n'a pas de prédécesseur
- un noeud qui n'a pas de successeur est une **feuille** ou **noeud externe**, tous les autres sont des **noeuds internes**
- une **branche** est une suite finie de noeuds de la racine vers une feuille telle que chaque noeud est le père du noeud suivant dans la suite : $\exists x_0, x_1, \dots x_p$ tels que :
  - x_0 est la racine
  - x_p est une feuille
  - $\forall k \in [1, p]$, père$(x_k) = x_{k-1}$
- la **longueur d'une branche** est le nombre de noeuds qui la compose **-1**
- un noeud y est un **descendant** d'un noeud x si $\exists x_0, x_1, \dots x_p$ avec
  - $x_0 = x$
  - $x_p = y$
  - $\forall k \in [1, p]$, père$(x_k) = x_{k-1}$
- la **profondeur** de y par rapport à x est p
- la **profondeur** d'un noeud dans un arbre est la profondeur de ce noeud par rapport à la racine de l'arbre
- la **hauteur** $h(a)$ d'un arbre a non vide est la longueur maximum d'une de ses branches = profondeur maximale d'un noeud, celle de l'arbre vide est -1
- l'**arité** d'un arbre est le nombre maximum de successeur qu'un noeud peut avoir, l'arbre vide est de toute arité
- la **taille** d'un arbre est son nombre total de noeuds



• Le noeud 1 est la racine de l'arbre T c'est aussi le père des noeuds 2, 3, 4 et 5.

• A est un arbre fils du noeud 1 dont la racine est le noeud 2.

• Les noeuds internes de T sont les noeuds 1, 2, 4 et 6. Les feuilles sont les noeuds 3, 5, 7, 8, 9, 10 et 11.

• Dans T, les noeuds 1, 2, 6 et 11 forment une branche de longueur 3 et les noeuds 1 et 3 en forment une autre de longueur 1.

• Dans T, le noeud 1 est de profondeur 0, le noeud 3 est de profondeur 1 et le noeud 8 est de profondeur 2.

• La hauteur de T est 3 et celle de B et D est 0.

• T est un arbre de taille 11 et d'arité 4. L'arbre A est de taille 4 d'arité 2.

3 Relation taille-arité-hauteur

Soient a l'arité d'un arbre, h sa hauteur, n sa taille et n_p le nombre de noeuds à profondeur $0 \leq p \leq h$, alors :

- $1 \leq n_p \leq a^p$
- taille : $h + 1 \leq n \leq (a^{h+1} - 1)/(a - 1)$
- hauteur : $\log_a(n(a - 1) + 1) - 1 \leq h \leq n - 1$

Pour prouver ces relations on commence par établir une borne inférieure et une borne supérieure du nombre de noeuds à une certaine profondeur dans un arbre. On sait qu'à profondeur 0 il n'y a que la racine. A profondeur 1 il y a les fils de la racine, soit au plus a noeuds. A profondeur 2 il y a les au plus a fils chacun des a fils de la racine, soit a^2 noeuds etc.

Ensuite la taille d'un arbre est simplement la somme du nombre de noeuds présents à chaque profondeur.

Enfin pour obtenir les bornes sur la hauteur d'un arbre il suffit de reprendre celles sur la taille et de les reformuler.

4 Définition récursive d'un arbre binaire

Un arbre binaire est un arbre d'arité 2. C'est soit :

- l'arbre vide *vide*
- un triplet (*Element*, *Gauche*, *Droit*) où *Gauche* et *Droit* sont deux arbres

Comme pour toute définition récursive, si on ne dispose pas de suffisament d'éléments de départ connus, ici l'arbre vide, on ne pourra construire aucun arbre.

On note $AB(E)$ l'ensemble défini récursivement des arbres binaires composés d'éléments appartement à un ensemble E : $AB(E) = \{vide\} \cup \{E \times AB(E) \times AB(E)\}$.

5 Primitives classiques

Toutes les primitives suivantes permettant de manipuler des arbres binaires sont supposées s'exécuter en temps constant ($\Theta(1)$) :

- **Constructeur** : fonction *creerArbre* : E×AB(E)×AB(E)→AB(E) : $e, g, d \mapsto < e, g, d >$
- **Sélecteurs** : condition d'utilisation : l'arbre n'est pas *vide*
  - fonction *racine* : AB(E) → E : $< e, g, d > \mapsto e$
  - fonction *gauche* : AB(E) → AB(E) : $< e, g, d > \mapsto g$
  - fonction *droit* : AB(E) → AB(E) : $< e, g, d > \mapsto d$
- **Prédicat** : fonction *estVide* : AB(E) → Booléen : $a \mapsto$ vrai si l'arbre est vide, faux sinon
- **Modificateurs** : condition d'utilisation : l'arbre n'est pas *vide*
  - procédure *modifierRacine* : AB(E) × E : $< e, g, d >, e' \mapsto < e', g, d >$
  - procédure *modifierGauche* : AB(E) × AB(E) : $< e, g, d >, g' \mapsto < e, g', d >$
  - procédure *modifierDroit* : AB(E) × AB(E) : $< e, g, d >, d' \mapsto < e, g, d' >$
- **Implémentation** : il y a en général une histoire de pointeurs là dedans ...