Arbres

Thibault PENNING

Définitions

Nous allons voir une nouvelle structure de données appelée arbre.

Définition 0.1 (Nœud)**.** Un nœud est un élément contenant une donnée et une référence vers un ou plusieurs autres nœuds.

Définition 0.2 (Arbre)**.** Un arbre est un ensemble de nœuds ayant plusieurs caractéristiques :

- Tous les nœuds sont connectés entre eux.
- Il n'y a pas de cycle. Cela signifie qu'il n'est pas possible de partir d'un nœud, de traverser plusieurs nœuds (de référence en référence), et retomber sur ce même nœud.
- Un nœud ne peux pas être référencé qu'une seule fois.

Nous pouvons voir un arbre comme une généralisation d'une liste chaînée, où chaque élément ne serait non pas relié à un seul élément suivant, mais à plusieurs.

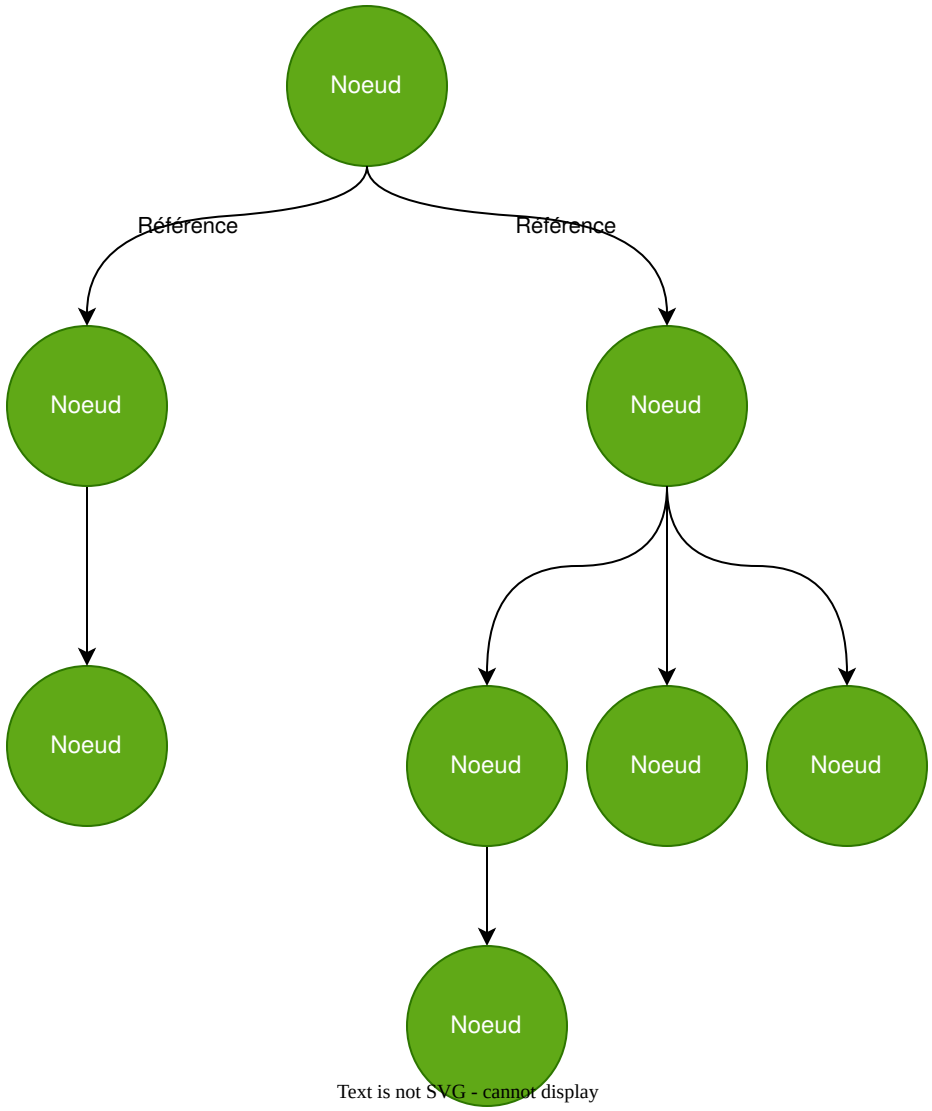

Figure 1: Représentation graphique d'un arbre. Un rond est un noeud, et une flèche représente une référence

Il faut aussi se rappeler que, tout comme les listes chaînées, un arbre ne stocke non seulement son/ses successeurs, mais aussi une donnée qui sera attribuée à chacun de ses noeuds.

Ici, nous avons choisi de définir les nœuds puis les arbres. Il est intéressant de constater que nous pouvons aussi définir un nœud comme un élément d'un arbre. Ces 2 définitions sont inséparables et il y a en quelque sorte un problème d'œuf ou de poule. Ainsi, il est courant de croiser d'autres définitions équivalentes à celles ci. La vraie définition utilisant des concepts mathématiques que nous ne verrons pas.

> **Note**
>
> Une conséquence d'être à la fois acyclique et entièrement connecté, et qu'il existe une certaine forme d'ordre à un arbre.
> Tout comme une liste, il y a un 1er, 2ème, 3ème, … élément, dans un arbre, il n'y a qu'une seule direction de propagation. Ainsi, contrairement aux listes, il peut exister plusieurs éléments à la 2nde ou 3e position. En revanche, un arbre part toujours d'un nœud et un seul nœud. Et nous pouvons ensuite définir les positions par rapport au nombre de nœuds nécessaires entre ce premier nœud et le nœud que l'on veut juger. Nous verrons cette notion un peu plus tard.
> Ce qui est important, c'est qu'il y a une forme de "direction" à un arbre.

Définition 0.3 (Racine)**.** La racine d'un arbre est le seul nœud de celui-ci qui n'est pas référencé par un autre nœud.

Ainsi, tout arbre part de la racine et continue de nœud en nœuds. La racine est un point de départ menant à **tous** les autres noeuds de l'arbre.

> **Note**
>
> On parle aussi d'arborescence (ou structure arborecentes) pour parler d'un arbre.

Il existe aussi une définition alternative récursive d'un arbre.

Définition 0.4 (Arbre)**.** Un arbre est un ensemble fini de nœuds respectant les 2 propriétés suivantes :

- Elle possède un noeud spécial appelé racine.
- Les autres ne qui ne sont pas la racine sont divisés en ensemble disjoints qui sont eux-mêmes des arbres.

Définition 0.5 (Enfant)**.** L'enfant d'un nœud est un autre nœud tel que le premier nœud référence, le 2nd. Ainsi, l'enfant d'un nœud est aussi l'un de ses successeurs.

On parle aussi de **successeur**.

Définition 0.6 (Parent)**.** Le parent d'un nœud est un autre nœud qui référence le nœud actuel. Ainsi, le parent d'un nœud est un nœud qui a pour enfant le nœud actuel.

On parle aussi de **prédéceseur**.

> **Note**
>
> Dans un arbre, il n'existe qu'un seul (ou 0 pour la racine) parent à un nœud, mais il peut exister en nombre aussi grand que l'on souhaite d'enfants.

Définition 0.7 (Feuille)**.** Une feuille est un nœud n'ayant pas d'enfant.

On note ainsi que une feuille et l'une des fins d'un arbre. Lorsque l'on tomConan, **** get on the way. **** France en Finlande. Non mais ce qui est intéressant, c'est de noter, c'est. Ensuite Je cours. Salut, comment vas-tu So Now in anything on I'm I can't got back here and the say in the what do this. I want that over the end is Love. be dans une feuille, on ne peut pas continuer à parcourir les nœuds de l'arbre.

Définition 0.8 (Sous-arbre)**.** Le sous-arbre d'un nœud est l'arbre représenté par l'un de ses enfants et tout Les enfants en découlant.

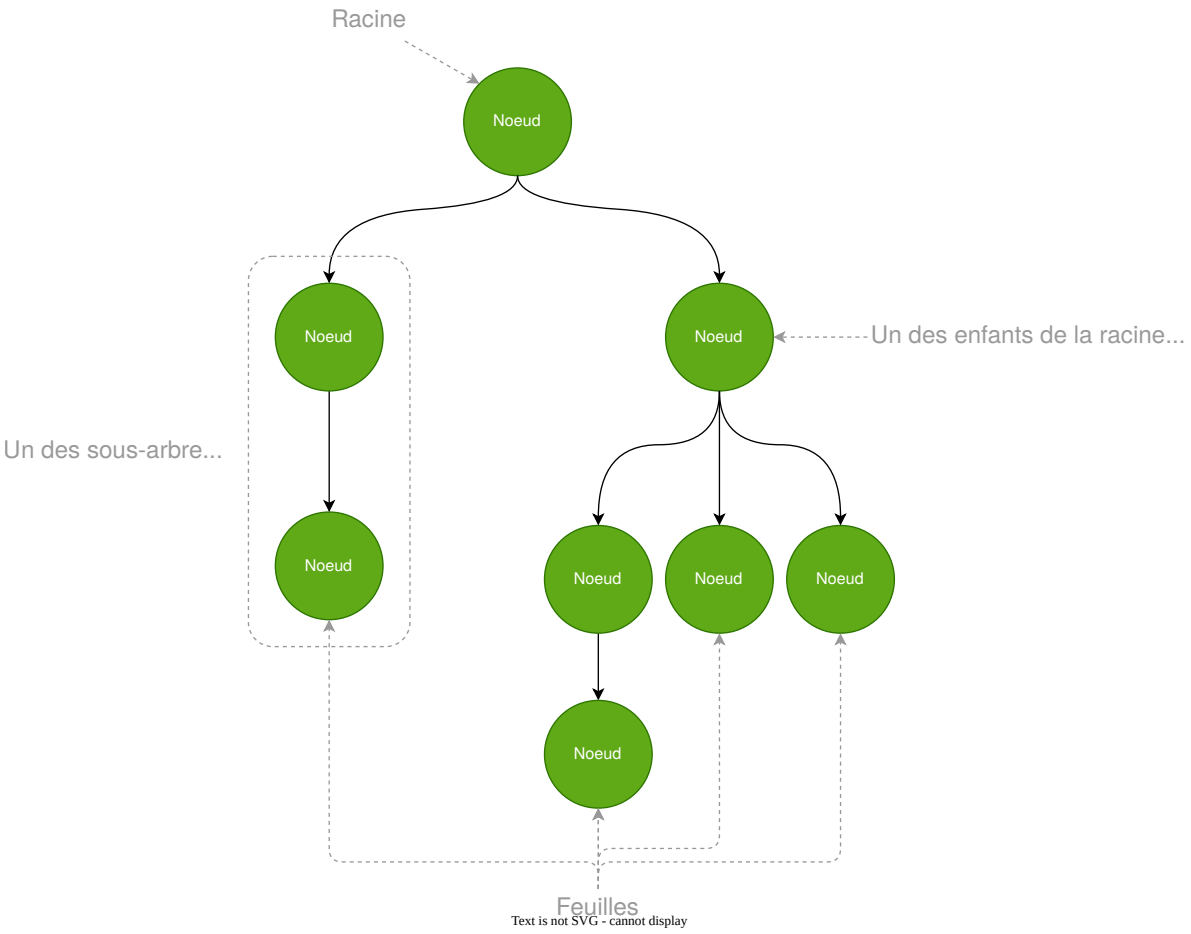

Figure 2: Vocabulaire d'un arbre

Définition 0.9 (Foret)**.** Une foret est un ensemble disjoint d'arbre.

Définition 0.10 (Taille)**.** On appelle la taille d'un arbre le nombre de nœud de l'arbre

Définition 0.11 (Hauteur)**.** Plus grand nombre de nœuds possible entre la racine et une feuille.

> **Note**
>
> Il existe plusieurs conventions, en fonction de si l'on considère ou pas la racine dans le calcul.
> Dans tout le bac, nous considérerons la racine dans le calcul de la hauteur.
> Ainsi l'arbre vide est le seul d'une hauteur de 0, l'arbre de hauteur 1 est l'arbre compor-

tant seulement la racine, et les autres hauteur (pour les arbres) n'ont pas de structure particulière.

i Encadrement de la hauteur

Dans un arbre quelconque, il n'éxiste pas d'encadrement exact de la hauteur. Nous pouvons simplement dire que :

- hauteur $\leq$ taille, dans le cas d'une liste chainée (qui est aussi un arbre).
- si taille $\geq 2 \Rightarrow$ hauteur ≥ 2 (cas d'une racine avec tout les autres noeuds qui sont son enfant).

Arbre binaire

Définition 0.12 (Arbre binaire)**.** Un arbre est dit binaire lorsque chaque nœud possède au plus enfants.

Définition 0.13 (Sous-arbre gauche/droit)**.** Lorsqu'un arbre est binaire, ses 2 sous-arbres sont appelés sous-arbre gauche et sous-arbre droit.

i Note

Le gauche est droit n'as de sens que lorsque l'on le représente. En général, on parle de gauche pour le premier sous-arbre et droit pour le deuxième, dans l'ordre des références.

Encadrement de la hauteur

Comme précédemment, on peut chercher à encadrer la hauteur par le nombre de nœuds. Contrairement à précédemment, nous savons que le nombre de fils étant limité, l'encadrement de la hauteur peut être beaucoup plus précis.

En particulier, nous avons :

$$\text{hauteur} \leq \text{taille} \leq 2^{\text{hauteur}} - 1$$

La première partie de l'inégalité s'explique de la même manière que précédemment. On utilise pour cela ce que l'on appelle un *peigne*, soit ne forme de liste chaîné où l'arbre est composé seulement de fils gauche ou seulement de fils droit.

L'autre partie de l'inégalité se retrouve en observant le nombre de nœuds dans l'arbre dit complet.

Barton de la racine et regardons le nombre de nœuds maximal que l'on peut avoir sur la dernière couche. On a :

$$1, 2, 4, 8, 16 = 2^{\text{hauteur-1}}$$

Le nombre total de nœuds pour une hauteur donnée est la somme de tous les nombres maximal de nœuds pour chaque hauteur. Rappelez-vous de l'écriture binaire d'un nombre et vous comprendrez ainsi que la somme des puissances de 2 est égale à la puissance de 2 supérieur moins un. Soit ici :

$$2^{\text{hauteur}} - 1$$

Il est facile de comprendre que tout autre cas se situe dans l'un de ces 2 cas extrêmes. D'où l'encadrement.

De même en réécrivant l'inégalité :

$$\log_2(taille) - 1 \leq \text{hauteur} \tag{1}$$

Parcours d'un arbre

On appelle parcours d'un arbre, le faite de visiter tous les nœuds de ce dernier, en passant de noeud en noeud.

En effet, il n'est possible de connaitre tous les noeuds des arbres, seulement en les traversant tous, et en récupérant tout les succésseur.

En revanche, pour parcourir un arbre (binaire), il existe de nombreuse manière.

> **Note**
>
> Dans de nombreux algorithmes, comme le calcule de la taille, ou le calcule de la hauteur, l'ordre importe peu. En revanche, pour l'affichage par exemple, celui-ci est très important.

Parcours en profondeur

Définition 0.14 (Parcours en profondeur)**.** Parcours d'un graphe, qui privilégie le parcours immédiat de ses enfants.

> **Note**
>
> On l'appel parcours en profondeur, car lors du parcours la distance a la racine de l'arbre augmente très vite, pour revenir en arrière et ainsi desuite. On peux le voir comme le parcours allant le plus rapidement aux feuilles.

Définition 0.15 (Parcours préfixe)**.** Le parcours préfixe (ou numérotation préfixe) consiste en l'utilisation de la valeur du noeud courrant, avant l'utilisation de la valeur des sous-arbres du noeud.

```
def prefix(n:Noeud, f:Callable):
    if n is not None:
        f(n.valeur)

        prefix(n.gauche, f)
        prefix(n.droit, f)
```

Définition 0.16 (Parcours suffixe)**.** Le parcours suffixe (ou numérotation suffixe) consiste en l'utilisation de la valeur des sous-arbres du noeud, avant la valeur du noeud courant.

```
def suffixe(n:Noeud, f:Callable):
    if n is not None:
        suffixe(n.gauche, f)
        suffixe(n.droit, f)

        f(n.valeur)
```

Définition 0.17 (Parcours infixe)**.** Le parcours infixe (ou numérotation infixe) consiste en l'utilisation de la valeur du sous arbre gauche, puis le noeud courant, puis le sous arbre droit.

```
def infixe(n:Noeud, f:Callable):
    if n is not None:
        infixe(n.gauche, f)

        f(n.valeur)

        infixe(n.droit, f)
```

! Pile et parcours en profondeur

Le parcours en profondeur peut-être réaliser, soit comme ici grâ à un appel récursif, soit grâce a une pile. On empile le prochain noeud a visiter et on dépile pour visiter. Ici l'ordre est l'inverse des appèle récursif. Ainsi pour l'appel préfixe, on empile d'abord les sous arbre, puis le noeud courrant.

i Note

On remarque qu'ici, la vérification de la fonction récursive est assez légère. Elle conciste simplement en la vérification qu'il s'agis bien d'un neoud. Cela est du a une propriété du graphe : il n'y a pas de cycle. Ainsi l'utilisation de la récursivité n'est ici pas un soucis. Que se passerait-il si cette condition n'était pas présente ?

Parcours en largeur

Définition 0.18 (Parcours en largeur)**.** Parcours d'un graphe, dans lequel les noeuds sont visité par ordre croissant de distance avec la racine.

Il existe 2 implémentation possible du parcours en largeurs :

- Le premier utilise 2 ensemble, un `courant` et un `suivant`. L'ensemble `courant` est celui sur lequel on itère (noeud a une hauteur h), et l'ensemble `suivant` sont composé des enfant de ces noeuds (hauteur $h+1$). Un fois l'ensemble `courant` itéré, on tranvase l'ensemble `suivant` dans courrant et réinitialisé l'ensemble `suivant`.
- Le deuxième consiste simplement en l'utilisation d'une pile. On enfile les enfants quand on les vois et on défile pour intéré sur l'abre. On peut se convaincre d'une propriété : dans la file, les premiers élément sont de hauteur h, puis les dernier de hauteur $h+1$, avec une séparation franche. On comprend ainsi le parcours en profondeur.

On choisis ici de vous montrer le second algorithme, car beaucoup plus simple. Retenez le premier, si le sujet ne vous dote par exemple pas d'une file.

```
def largeur(racine:Noeud, f:Callable):
    fifo = File()
    fifo.enfile(racine)

    while fifo.non_vide():
        n = fifo.defile()
        fifo.enfile(n.gauche)
        fifo.enfile(n.droit)
```

```
        f(n)
```

> **Note**
>
> Changement de la fonction `taille` et `hauteur` avec un arbre binaire.

> **Efficacité**
>
> On a un nombre d'oppération proportionelle aux nombres de noeuds ($\mathcal{O}(n)$)

Arbre binaire de recherche

Définition 0.19 (Arbre binaire de recherche)**.** Un arbre binaire de recherche est un arbre binaire possédant les propriétés suivantes :

- Toutes les valeurs peuvent être comparées entre elle selon un ordre prédéfinie (entier, ordre lexicagraphique pour les chaine de caractère, …)
- Les valeurs situé dans le sous arbre gauche sont inférieurs a la valeur de la racine qui elle même est inférieur au valeurs dans le sous arbres droits.

Exemple 0.1.

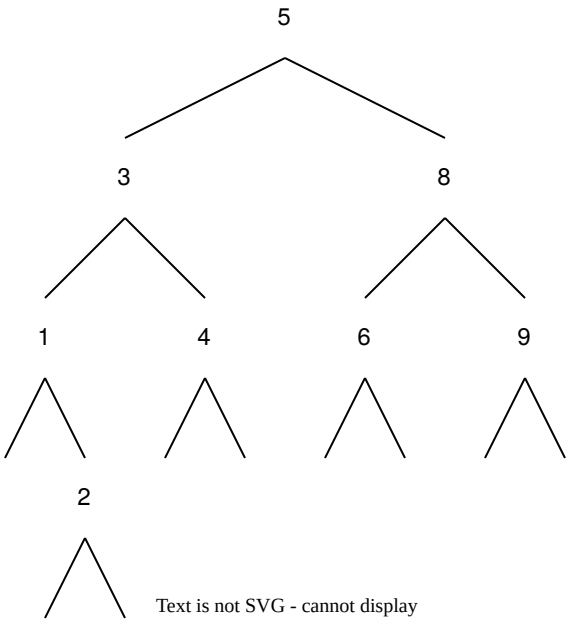

Figure 3: Exemple d'un ABR

> **Note**
>
> On remarque qu'un ABR est un structure de données qui utilise le principe de la dichotomie pour l'organisation des ses noeuds.

Recherche dans un ABR

La recherche dans un ABR suis le même schéma que dans une recherche dichotomique : on compare la valeur souhaitez avec la valeur de la racine. En fonction de la comparaison, on s'arrête, ou on rappel recursivement la fonction sous le sous arbre gauche ou droit.

Efficacité

On peut remarqué un point important : la recherche est **gloutonne**, le choix du sous-arbre n'est jamais remis en cause, et la fonction s'arrête forcément avant ou au moment où une feuille est rencontré. La recherche est donc bornée par la **hauteur**.

Comment cela évolue par rapport à une recherche naïve ?

Cas d'un arbre peigne.

Cas général ?

Ajout dans un arbre binaire de recherche

On se base sur la recherche et ajoute l'élément là où l'on s'attend à le trouver.

Pour une version immuable, on peux utiliser un appel récursif, où l'on construit un nouveau noeud à chaque revoie de valeurs.

> **Doublon**
>
> On peux réaliser 2 version d'un ABR, un où l'on autorise les doublons et l'autre non. Remarquons que les fonction de recherche et d'ajout sont modifié.

Efficacité

On remarque que l'ajout ne diffère pas dans le principe de la recherche on a donc :

$$\mathcal{O}(h) \Rightarrow \mathcal{O}(n)$$

Équilibrage d'un ABR

Il existe des algorithme pour équilibré un arbre. Cela n'est pas au programme. Retenez simplement cette chose : l'équilibrage conciste a remplir au maximum les arbres, de tel sorte qu'il minimise la hauteur.

Il existe la notion d'abre parfaitement équilibré. Dans ce cas, la hauteur est totalement remplis. Rappelé vous de l'Équation 1, on aurait donc dans ce cas une insertion et recherche en :

$$\mathcal{O}(\log_2(n))$$

Soit la recherche et l'insertion proportionnelle au logarithme en base 2 du nombre d'éléments.

Cela signifie qu'il y a autant d'opération que de bit dans le nombre d'éléments du graphe !

Avec cette définition, on remarque que la complexité logarithmique est **bien** meilleur que la complexité linéaire. On comprend maintenant l'utilité d'un arbre binaire !