Structures de données arborescentes

1 Motivations

1.1 Un arbre pour un objet

Pour des éléments d'un ensemble construit par induction, il est approprié d'utiliser une représentation par arbre puisque ces objets ont intrinsèquement une structure arborescente

Exemple: Les expressions booléennes, arithmétiques, de type.

1.2 Un arbre pour une collection d'objet

On cherche à stocker une collection d'objets sans multiplicité, et dont l'ordre relatif n'est pas significatif (comme dans le cas d'un ensemble).

On suppose que tous les éléments sont du type `elem`, et qu'ils sont identifiés de manière unique par une clé, c'est à dire une sous-partie permettant l'identification.

On a alors les méthodes suivantes :

- `creer_ens_vide : () -> ens`
- `est_ens_vide : ens -> bool`
- `appartient : ens×elem -> bool`
- `ajoute_elem : ens×elem -> ens`
- `supprime_elem : ens×clé -> ens`
- `trouve_elem : ens×clé -> elem`

Remarque: On peut aussi imaginer des fonctions `ajoute_elem` et `supprime_elem` qui modifieraient directement l'ensemble donné en entrée, plutôt que de renvoyer un nouvel ensemble.

1.3 Implémentations

On peut stocker les éléments dans une liste. On peut aussi associer chaque élément à une clé (pour un ensemble de caractères par exemple, leurs valeurs ascii), qui permet de les comparer rapidement. Si l'on peut ordonner ces clés, on peut alors classer les éléments par ordre croissant dans un tableau.

Opération	Complexité (Liste)	Complexité (Tableau ordonné)
`appartient`	$\theta(n)$	$\theta(\log(n))$ (dichotomie)
`ajoute_elem`	$\theta(1)$	$\theta(n)$
`supprime_elem`	$\theta(n)$	$\theta(n)$
`trouve_elem`	$\theta(n)$	$\theta(\log(n))$

Remarque: On voit selon le contexte qu'une certaine implémentation sera plus efficace qu'une autre : si on doit faire beaucoup d'insertions sans trop chercher d'éléments, le plus efficace sera la liste. Par contre, si on n'insère que rarement des éléments mais que l'on est souvent ammené à chercher dans les éléments, le tableau trié sera à préférer.

ll existe cela dit une structure qui permet d'avoir une complexité d'ajout / suppression et de recherche en $\theta(\log(n))$, sous certaines conditions : il s'agit des **arbres binaires de recherche (ABR)**.