Arbre binaire de recherche

Objectifs

— Manipuler un chaînage hiérarchique

Une structure de données associative peut être implantée en utilisant un arbre binaire de recherche (ABR). La contrainte est de disposer d'une relation d'ordre totale sur les clés. Pour les chaînes de caractères, la relation d'ordre peut être l'ordre lexicographique (celui du dictionnaire).

Dans un ABR, chaque *nœud* contient une clé, la donnée associée, un *sous-ABR gauche* et un *sous-ABR droit*, éventuellement vides. Un ABR est *vide* s'il ne contient aucun nœud. Sa *taille* est son nombre de nœuds.

La *racine* d'un arbre est le nœud situé le plus « haut » dans cet arbre. Par extension, chaque nœud d'un ABR peut donc être considéré comme la racine d'un sous-ABR de l'ABR complet.

Une *feuille* est tout nœud d'un arbre binaire dont les deux sous-ABR sont vides.

Un *arbre binaire de recherche* est construit de façon à toujours respecter la contrainte suivante : *si n est un nœud de l'arbre et n_g est un nœud du sous arbre gauche de n, alors la clé de n_g est strictement inférieure à celle de n ; de même si n_d est un nœud du sous arbre droit de n, alors la clé de n_d est strictement supérieure à celle de n.*

Ainsi, voici le principe pour insérer un nouvel élément (clé + donnée) dans un ABR. Si l'ABR est vide, on créé un nouveau nœud qui devient la racine de cet ABR. Sinon, si la clé du nouvel élément correspond à la clé de la racine, c'est que la clé est déjà présente. Si la clé est différente, on sait qu'il faut insérer dans le sous-ABR gauche (si la nouvelle clé est plus petite que celle de la racine) ou dans le sous-ABR droit (nouvelle clé plus grande).

On constate ainsi que la plupart des opérations sur l'ABR consistent à suivre un chemin dans l'arbre depuis la racine. Si l'arbre est équilibré, le nombre d'opérations sera donc $\log_2(n)$, bien plus efficace qu'avec une structure linéaire. Un arbre est équilibré si toutes les feuilles sont comprises à distance h ou $h-1$ de la racine, $h = \log_2(\text{taille})$ est la hauteur de l'arbre.

Exercice 1 Schématiser l'ABR obtenu suite à l'enregistrement des données suivantes (clé à gauche, donnée à droite). On pourra ne mettre que l'initial de la clé dans le nœud de l'ABR.

Frank	6156
Bob	9278
Henri	1476
Dick	9327
Alice	3890
Eric	9223
Giles	4512
Jack	6843
Irene	0924
Chris	3839

Exercice 2 Définir le type `T_ABR`.

Exercice 3 Écrire un sous-programme qui donne la taille d'un ABR.

Exercice 4 Écrire un sous-programme qui affiche la clé et la donnée associée dans l'ordre croissant des clés.

Exercice 5 Écrire le sous-programme qui fournit la donnée associée à une clé dans un ABR. L'exception `Cle_Absente_Exception` sera levée si la clé n'est pas présente dans l'ABR.

Exercice 6 Écrire le sous-programme qui enregistre un couple clé/donnée dans un ABR.

Exercice 7 Écrire le sous-programme qui supprime une clé (et la donnée associée) d'un ABR.
Indication : Si le nœud à supprimer a moins de deux enfants, il est facile de le supprimer. En revanche, si ce nœud a deux enfants, il faut veiller à ne pas déséquilibrer l'ABR. L'idée est alors de remplacer ce nœud par le plus petit de son sous-ABR droit (qui n'a donc pas de fils gauche).

On pourra définir le sous-programme `decrocher_min` qui extrait d'un ABR le plus petit élément. La figure 1 illustre cette opération. Le plus petit élément de l'ABR (`B`) a été extrait, tout en conservant la structure globale de l'ABR.

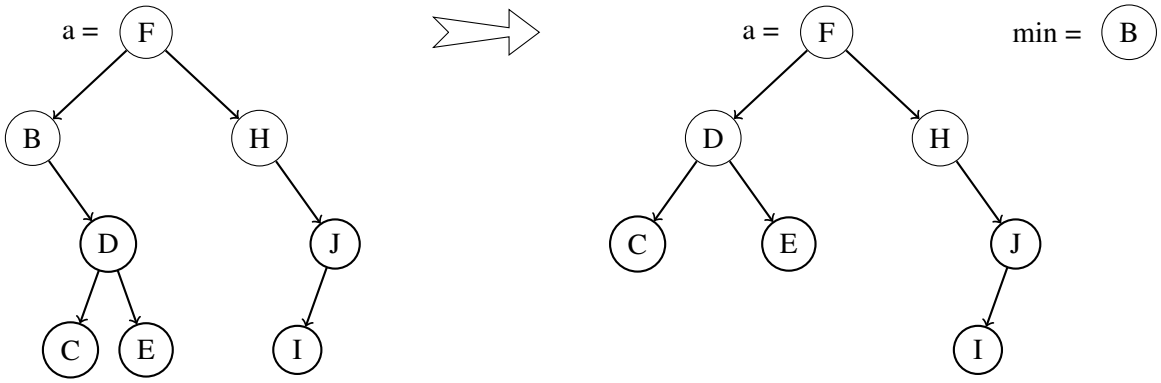

FIGURE 1 – Extraction du plus petit élément