Yael Le Quilleuc
Léo Garreau
Rama Suntharasarma
Maxime Arbelot

Structure de données : Devoir 1 - Rapport de l'exercice 3

Outre le code source, il sera demandé de fournir un rapport expliquant les différentes méthodes implémentées, ainsi que les difficultés rencontrées.

Méthodes implémentées:

- **Ajouter mot**

La méthode `ajouterMot(string mot, Noeud<char> &noeud)` est une fonction récursive : on fournit une string (ce qu'il reste de notre mot) ainsi que le noeud courant. On appelle la fonction initialement sur le mot complet et la racine de notre dictionnaire.

Tant que le mot n'est pas vide, on identifie plusieurs cas possibles :

Si le noeud est vide, on ajoute la lettre dans ce noeud. S'il reste encore des lettres dans le mot après celle-ci, on crée un nouveau noeud gauche et on rappelle la fonction sur avec mot restant sur le fils gauche. S'il ne reste plus de lettre, on place l'attribut fin du noeud courant à vrai pour signaler la fin d'un mot.

Si le noeud n'est pas vide, alors on a deux possibilités :

Soit la prochaine lettre est la même que celle dans le noeud, dans ce cas s'il n'y a pas de noeuds gauche on en crée un puis, si le mot continue, on rappelle la fonction sur la suite du mot et le noeud gauche, sinon on met l'attribut fin à vrai pour signaler la fin d'un mot.

Soit la lettre n'est pas la même que celle dans le noeud. Dans ce cas on crée un noeud à droite s'il n'en existe pas déjà. Puis, comme on veut s'assurer que notre dictionnaire est bien rangé par ordre alphabétique (de a à z), si la lettre courante est plus petite que celle du noeud, on déplace notre noeud courant dans le noeud de droite, et on ajoute la lettre dans le noeud courant. Si le mot n'est pas vide, on rappelle la fonction sur le mot restant et le fils gauche, sinon on signale la fin du mot en mettant fin à vrai. Si la lettre courante est plus grande que celle du noeud, on se contente de rappeler la fonction sur le mot entier (la lettre n'a pas été placée) sur le fils droite.

Il faut être bien vigilant quand on déplace le noeud courant à droite : il faut réinitialiser tous les paramètres aux valeurs par défaut.

- **Supprimer mot**

La méthode *enleverMot(string mot, Noeud<char> *noeud)* se charge de supprimer les mot. Elle prend en paramètre le mot à supprimer ainsi que la racine de notre dictionnaire. La première étape de cette fonction est de lister tous les noeuds qui constituent le mot que l'on veut supprimer. C'est l'objectif de la première boucle *while*.

Dans cette boucle, on parcourt notre arbre.

Soit le noeud courant correspond à la lettre et,

si c'est la fin du mot, on arrête de boucler et si l'attribut fin du noeud est bien à vrai on ajoute ce noeud à notre pile,

si le mot n'est pas fini, on ajoute le noeud à notre pile et s'il y a un noeud gauche, on choisit ce noeud gauche comme noeud courant. S'il n'y en a pas, on arrête de boucler.

Soit le noeud courant ne correspond pas à la lettre et si un noeud droit existe, on positionne notre noeud courant sur ce noeud droit et sinon on arrête de boucler.

A la fin de cette boucle on obtient une pile de noeuds, avec chaque noeud correspondant à une lettre du mot. Pour s'assurer que le mot choisi est bien présent dans l'arbre il suffit de comparer la taille de la liste de noeuds et celle du mot : si elles sont égale le mot est bien présent. Sinon il ne l'est pas et on le signale à l'utilisateur.

Si le mot est bien présent, on veut maintenant le supprimer du dictionnaire. Pour cela on va vider notre pile de noeuds et pour chacun de ses éléments, faire les modifications nécessaires en commençant par le noeud le plus bas dans le dictionnaire (la dernière lettre du mot à supprimer).

Si le noeud courant a un fils gauche, on place le paramètre fin à false. S'il n'a pas de fils gauche mais a un fils droit, on déplace le noeud droit sur le noeud courant. Il faut alors s'assurer que le paramètre fin du noeud courant soit bien égal à celui du noeud droit. S'il n'a ni fils droit, ni fils gauche, on récupère le noeud parent (le noeud suivant de la liste), et on supprime l'accès au noeud courant (droit s'il s'agit du fils droit, gauche sinon).

La difficulté a été de trouver une manière de pouvoir accéder au parent du noeud que l'on veut supprimer : supprimer un noeud du dictionnaire revient enlever le pointeur vers ce noeud chez le parent. Pour autant, la dernière lettre du mot n'est jamais supprimée. Mais comme la paramètre fin de ce noeud est mis à false, la lettre ne sera jamais affichée. C'est pourquoi il a été décidé de prendre une liste des noeuds constituant le mot.

- **Afficher Dictionnaire**

La méthode *affichерDictionnaire(queue<char>&liste,Noeud<char> &noeud)* est une fonction récursive qui se charge d'afficher tous les mots contenus dans le dictionnaire. Au premier appel, elle prend une file vide et la racine de notre dictionnaire. L'idée est que si le noeud courant est la fin d'un mot, on affiche tous les éléments contenus dans la file. Si elle a un noeud gauche, on ajoute la lettre à notre file et on rappelle la fonction cette liste et le noeud gauche. Si elle a un fils droit, on appelle la fonction sur la file actuelle et le fils droit. À chaque appel de fonction, on vérifie ces trois cas.

Avec cette méthode, les difficultés que nous avons rencontré étaient dues à la gestion de la file : à quel moment doit-on en créer une copie, doit-on ajouter la lettre à la copie ou à l'original. Un autre problème est que potentiellement, un très gros dictionnaire fasse une grosse consommation de mémoire.

L'affichage par ordre alphabétique fonctionne car l'arbre a été construit de sorte que dès lors que l'on ajoute un mot, ce-dernier soit bien placé par ordre lexicographique.

- **Chercher Mot**

La méthode *chercherMot(string mot, Noeud<char> &noeud)* est une fonction récursive qui permet de déterminer si un mot est présent ou non dans le dictionnaire. À l'initialisation, elle prend en paramètre le mot recherché et la racine du dictionnaire. Si le mot est présent elle renvoie vrai, sinon faux.

À chaque itération, on récupère la première lettre du mot et on crée un nouveau mot sans cette lettre. Puis on distingue deux cas :

Le noeud courant correspond à la première lettre du mot. Dans ce cas, s'il reste encore des lettre au mot, on regarde si un noeud gauche est présent. SI oui, on rappelle la fonction sur le mot privé de cette lettre et si non, on renvoie faux : le mot n'est pas présent. S'il ne reste plus de lettres, on regarde si le noeud courant est bien la fin d'un mot. Si oui, le mot est présent, on renvoie vrai, sinon il n'est pas présent et on renvoie faux.

Le noeud courant ne correspond pas à la lettre. Dans ce cas on regarde le noeud droit. S'il est présent on rappelle la fonction avec le mot entier (la première lettre n'a pas encore été trouvé) et le noeud droit. S'il n'y a pas de noeud droit, le mot n'est pas présent : on renvoie faux.

Aucune difficulté particulière n'a été rencontré sur cette méthode.