Structure de données

Rapport de TP

TP n°3
~
Gestion d’un dictionnaire arborescent

Membres du groupe :
HABERT Eldred
COUMAILLEAU Lilian

Encadré par :
ANTOINE Violaine

Groupe :
G12

29/05/2019

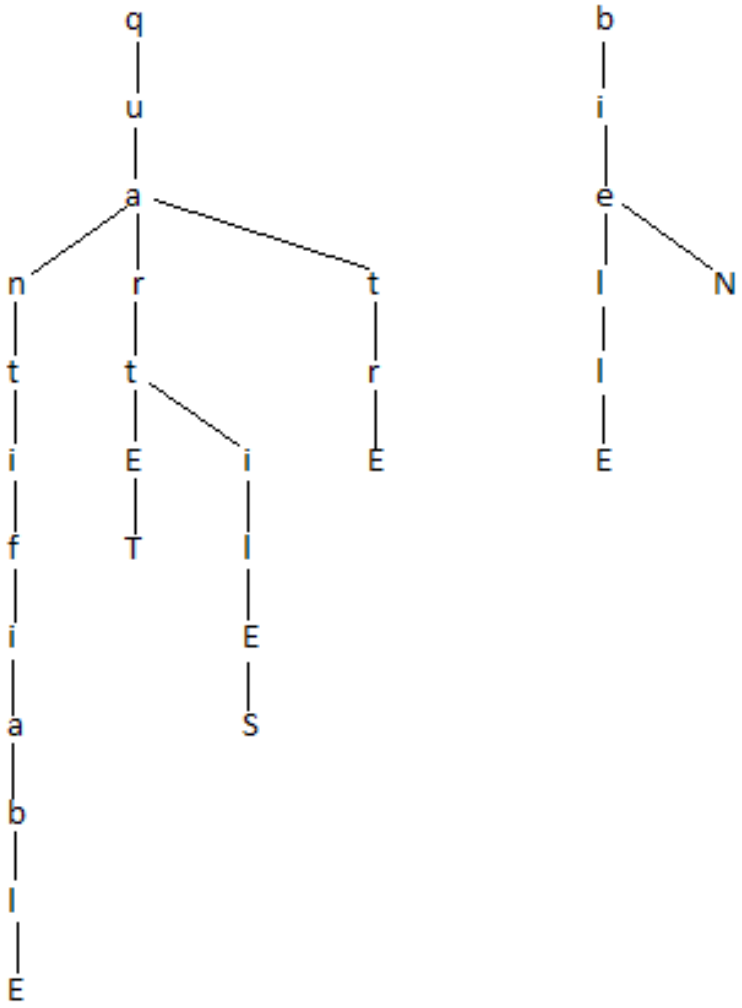
Table des matières

1 Rappel de l’énoncé

On représente un dictionnaire (un ensemble de mots) par une structure arborescente, stockée par lien vertical et lien horizontal chaînés, telle que :

- chaque nœud est une lettre ;
- une lettre ne figure qu’une seule fois dans la liste des successeurs d’un nœud ;
- les lettres appartenant à une même liste chaînée par le lien horizontal sont ordonnées par ordre alphabétique ;
- si un nœud est une majuscule, la suite de lettres depuis sa racine jusqu’à ce point est un mot ;
- si un mot est dans le dictionnaire, il existe un unique chemin depuis une racine jusqu’à une lettre en majuscule dont les valeurs des nœuds forment ce mot.

Nous avons décidé de créer le dictionnaire suivant pour nos tests :



2 Description de l’objet du TP

L’objectif du TP était de créer les sous-programmes suivants :

- Insertion d’un mot dans le dictionnaire à partir d’un fichier texte ;
- Affichage des mots du dictionnaire dans l’ordre alphabétique ;
- Recherche dans le dictionnaire de tout les mots commençant par un motif donné

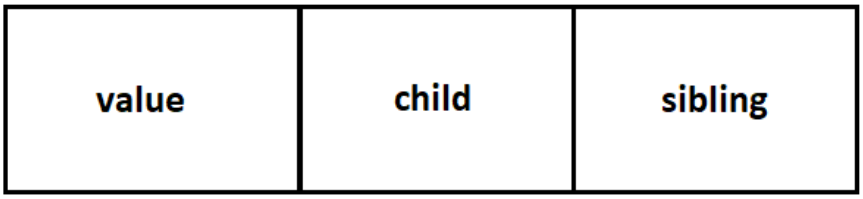
3 Structures de données

Dans notre programme nous utilisons les structures de données suivantes :

- **GrowingArray_t**

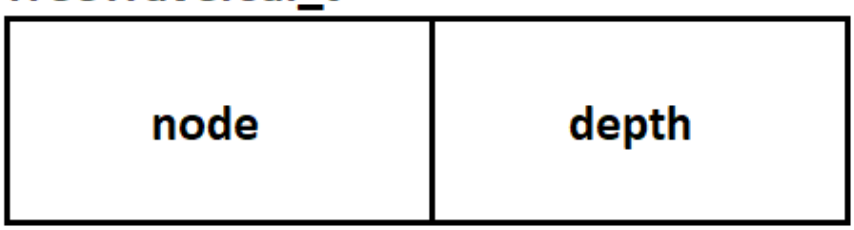
  Il s’agit d’un tableau de taille variable, répliquant le std::vector du C++

- **TreeNode_t**



  Cette structure forme un nœud d’un arbre, utilisé ici pour former un dictionnaire

- **TreeTraversal_t**



  Cette structure est utilisée en interne par la fonction de parcours en profondeur

Notre programme contient aussi un fichier de données passé en paramètre, le fichier dans lequel sont placés les mots du dictionnaire. Dans notre jeu de tests, le fichier s’appelle input.dict.

Ce jeu de tests n’est pas minimal, mais il couvre tous les cas : « création » de l’arbre (insertion du premier nœud), création d’un nouveau frère à un nœud, ajout d’un mot alors qu’un radical est déjà présent, ajout d’un mot radical d’un autre déjà présent, ajout d’un mot déjà présent, et ajout d’une nouvelle racine.

4 Organisation du code source

Chaque sous programme est dans le ficher source correspondant à son objectif. Ainsi nous avons dans les différents fichiers sources :

tree :

- `create_node` : crée un nœud
- `destroy_tree` : détruit un arbre
- `depth_first_traversal` : effectue un parcours préfixe en profondeur de l'arbre en appelant une fonction sur chaque nœud rencontré
- `seek_child` : cherche une lettre dans l'arbre parmi les fils d'un nœud
- `print_tree` : afficher le contenu d'un arbre (utilisé pour le débogage)

dict :

- `find_node` : cherche le dernier nœud correspondant à une chaîne de caractères
- `insert_word` : insère un mot dans le dictionnaire
- `insert_words :` insère plusieurs mots consécutivement
- `insert_words_from_file` : insère une liste de mots à partir d'un fichier texte
- `list_words` : affiche tous les mots du dictionnaire
- `list_words_prefixed` : affiche tous les mots du dictionnaire ayant pour préfixe la chaîne de caractères passée en paramètre

Les fichiers « stack » contiennent une implémentation de pile extraite d'un précédent TP.