École Polytechnique de Montréal
Département de Génie Informatique et Génie Logiciel

# LOG2810

# STRUCTURES DISCRETES

## Hiver 2017

### TP2 : Automates et langages

---

## Remis par :



| Matricule | Prénom & Nom |
|---|---|
| 1798345 | Adam Martin-Côté |
| 1786724 | Jonathan Gervais-Ranger |
| 1828978 | Louis Cormier |



## À :
## David Johannès,
## Mariam Tagmouti

**Le 6 avril 2017**

# Table des matières

# Introduction

Dans le cadre du cours *Structures discrètes,* nous avons étudié plusieurs outils mathématiques importants en informatique. Une question que plusieurs personnes se posent dans le domaine informatique est : Comment structurer de l'information afin de parvenir à faire une recherche plus efficace parmi les données?

Parmi toutes les structures de données, il y en a une en particulier que nous avons étudiée dernièrement, les automates. Ces derniers sont très utilisés afin de bâtir des grammaires. Ce sont de petites machines qui indiquent vrai si un mot fait partie d'une certaine grammaire et indique l'inverse s'il n'en fait pas partie. Puisqu'ils sont très efficaces, il nous a été demandé d'implémenter un automate qui est capable d'importer un lexique et qui permet par la suite de faire de la complétion et de la correction de mots.

Dans ce rapport, vous trouverez des informations concernant notre implémentation de ce logiciel. Plus précisément, vous trouverez dans un premier temps, vous trouverez l'explication de notre solution incluant notre diagramme de classes. Dans un deuxième temps, vous trouverez de l'information concernant l'implémentation de nos classes. Dans un troisième et dernier temps, vous trouverez de l'information concernant les difficultés que nous avons rencontrées lors de la création de notre logiciel.

# Présentation de la solution

## Diagramme de classes

Afin de parvenir d'arrivée à notre solution, nous avons dû diviser le programme en quelques classes. La figure 1 présente ces dernières.

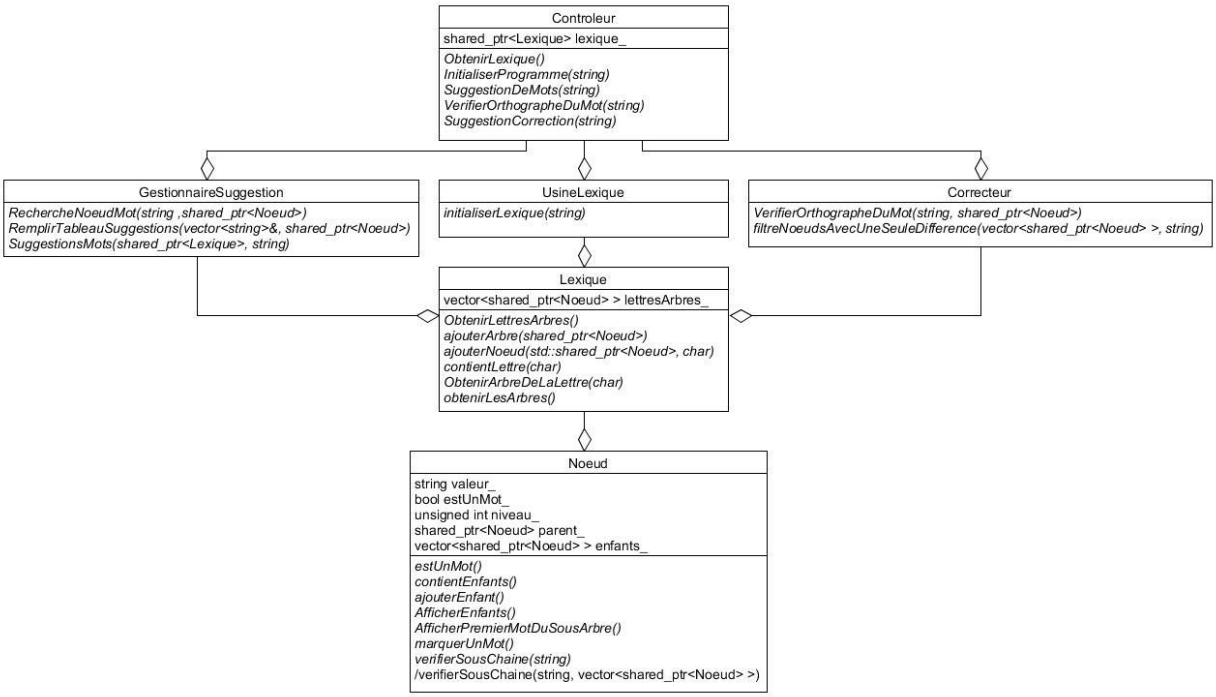


*Figure 1 : Diagramme de classe*

Comme la figure 1 le montre bien, nous avons d'abord implémenté une classe Contrôleur. La fonction de cette dernière est d'appeler les méthodes *initialiserLexique*, *suggestionsMots* et *filtreNoeudsAvecUneDifference*. Pour y parvenir, elle doit inclure les classes UsineLexique, GestionnaireSuggestion et Correcteur.

Pour arriver à importer des lexiques, nous avons implémenté une classe qui se nomme UsineLexique. Cette dernière sert à produire un lexique à partir d'un fichier.

De plus, pour la fonctionnalité suggestion, nous avons implémenté une classe qui se nomme GestionnaireSuggestion. Cette dernière a une fonction SuggestionMots qui prend en attribut un mot et qui retourne un vecteur contenant une liste de suggestions. Cette dernière fait appel aux fonctions RechercheNoeudMot, qui vérifie s'il existe un nœud dans le lexique contenant le mot entré, et a RemplirTableauSuggestions qui prend un vecteur en attribut ainsi qu'un nœud de départ et retourne un tableau rempli de suggestions.

En outre, pour la fonctionnalité correction, nous avons créé une classe qui se nomme Correcteur. Dans cette dernière, il y a une fonction qui se nomme VerifierOrthographeDuMot qui vérifie si le mot est bien orthographié en vérifiant s'il existe dans le lexique, sinon il appelle la

fonction filtreNoeudAvecUneSeuleDifference qui va parcourir le lexique jusqu'à trouver un mot qui a une lettre de différence avec le mot entré.

Puis, nous avons créé une classe qui se nomme lexique. Cette dernière contient une forêt d'arbre contenant tous les mots des lexiques importés. Chacun des arbres a les mots commençant par des lettres différentes.

Finalement, nous avons une classe nœud. Cette classe est tout simplement l'implémentation de chacun des nœuds pour les arbres qui sont utilisés par le lexique. Chacun des nœuds a en attribut un mot, son nœud parent, ses nœuds enfants, son niveau et aussi un booléen qui indique si à ce niveau nous avons un mot complet ou non.

## Implémentation

### Implémentation des classes

Les premières classes qui ont été implémentées sont Nœud et Lexique puisque les autres classes ont une relation d'agrégation avec ces derniers. Pour ce faire, nous avons tout simplement mis les attributs que nous jugions pertinents avec les méthodes pour obtenir les différents attributs. Les autres fonctions ont été implémentées par la suite au fur et à mesure que nous réalisions que nous avions besoin d'une nouvelle méthode.

Par la suite, nous avons implémenté la classe UsineLexique pour pouvoir tester nos fonctions à partir des lexiques fournis. Pour y arriver, nous avons tout d'abord travaillé à lire les mots individuellement et arriver à les placer dans un arbre de recherche individuellement. Puis, pour optimiser la recherche des mots, nous avons modifié l'algorithme afin d'éviter d'avoir des doublons pour des préfixes. Par exemple, si nous avons les mots café et cafouillage, nous voulions qu'il y aille un seul chemin pour se rendre à caf et que le reste des lettres se trouvent dans les enfants du nœud « caf ».

Puis, pour implémenter la classe GestionnaireSuggestion, nous avons tout d'abord commencé par créer la fonction SuggestionMots qui prend en paramètre le lexique dans lequel il doit chercher ainsi que le mot entré par l'utilisateur. La première chose que cette fonction fait c'est de vérifier si le mot entré par l'utilisateur fait partie du lexique. S'il ne le trouve pas, il indiquera qu'il n'y a pas de suggestions, sinon, il continue.

Une fois que le nœud contenant le mot entré par l'utilisateur a été trouvé, nous avions besoin d'une fonction récursive capable de trouver à travers le lexique les mots existant dans les nœuds du sous-graphe commençant par le nœud trouvé. Cette fonction, prend en paramètre un vecteur par référence afin que ce soit toujours le même tableau qui est modifié et non une copie à chaque fois et le nœud de départ. Ensuite, il parcourt chacune des branches en profondeur et à chaque fois qu'il rencontre un nœud qui est un mot, il l'ajoute dans le tableau. Finalement, lorsqu'il a fini de parcourir le sous-arbre ou que le tableau contient dix suggestions, il quitte la fonction et les retournes.

Par la suite, pour implémenter la classe Correctrice, nous avons créé une méthode qui se nomme VérifierOrthographeDuMot afin de s'assure qu'on n'a pas déjà un mot bien orthographié,

pour éviter de modifier un mot bien écrit. Puis, nous avons développé un algorithme qui vérifie quel mot est à une lettre de différence du mot entré. Pour se faire, l'algorithme vérifie la longueur du mot et ensuite dans l'arbre de la première lettre du mot, il vérifie tous les mots de ce niveau pour un trouver un qui n'a qu'une seule lettre de différence.

## Difficultés rencontrées

Une première difficulté que nous avons eue a été causée par la séparation de tâches. Il fallait que nous trouvions une façon de tester nos fonctions pendant que la personne responsable finit sa fonction pour importer un lexique. Pour pallier cette difficulté, nous avons créé un petit lexique temporaire.

Par ailleurs, nous devions optimiser l'algorithme pour importer les lexiques afin que la recherche des mots soit plus rapide. Il a donc fallu trouver une façon pour que les préfixes qui se répétaient ne deviennent pas des doubles.

Lors de l'implémentation de la fonction suggestion, nous avons rencontré deux difficultés. La première était de trouver une façon de parvenir à ce que le terminal se mette à jour à chaque fois que l'utilisateur change le texte écrit. La stratégie que nous avons utilisée pour contrer cette difficulté a été de créer une fonction où l'utilisateur doit appuyer sur la touche retour pour obtenir ces suggestions. Lorsque nous étions certains que la fonction était fonctionnelle, c'est à ce moment-là que nous nous sommes attardés à la mise à jour dans le terminal. Nous avons pris la décision de faire une interface graphique à l'aide du logiciel QT puisqu'il offre une fonctionnalité qui appelle une fonction lorsqu'une boîte de texte est modifiée.

La deuxième difficulté pour la fonction suggestion a été de trouver une fonction récursive qui se promène dans le lexique. Afin de simplifier la tâche, nous avons tout d'abord implémenté une fonction qui cherche un nœud contenant le terme entré par l'utilisateur. En faisant cela, la recherche de suggestions devenait plus optimale puisqu'il aura juste à chercher des suggestions dans ce sous-graphe. De plus, en faisant cela, on pouvait aussi déjà retourner à l'utilisateur qu'il n'y a pas de suggestions puisque le nœud qu'il est entré n'existe pas. Par la suite, nous avons lancé une autre fonction qui se promène en profondeur dans le sous-graphe afin d'arriver à trouver tous les nœuds terminaux et qu'il arrête au moment qu'il a trouvé dix suggestions ou que le sous-graphe a fini d'être parcouru.

Pour la classe Correcteur, la difficulté que nous avons dû gérer était de trouver un mot qui a exactement une lettre de différence. Il était difficile de trouver une façon de parcourir l'arbre en tenant compte qu'il fallait avoir une lettre de différence avec le mot entré.

Finalement, une grosse difficulté a été de faire l'interface graphique. Initialement, nous voulions l'implémenter dans le terminal. Mais, après discussion, nous avons décidé de le faire avec l'aide du logiciel QT, car il est beaucoup plus facile de gérer les évènements avec un logiciel pour faire une interface graphique que dans le terminal. Par la suite, nous nous sommes rendu compte que l'utilisation du logiciel était légèrement plus compliquée que nous le pensions. Nous avons dû apprendre plusieurs fonctionnalités de QT et aussi comment créer un exécutable qui fonctionne sur des ordinateurs qui n'ont pas QT d'installé. Finalement, plusieurs « bugs » se sont introduits

dans notre logiciel que nous n'eussions pas prévu comme que le logiciel arrêtait de fonctionner si aucun lexique n'était importé ou que quelqu'un fermait la fenêtre pour choisir un lexique.

## Conclusion

En conclusion, nous avons trouvé ce travail très intéressant puisqu'il nous a permis de travailler une structure de données différentes. Nous avons dû trouver une façon d'optimiser la recherche de mots dans un arbre en trouvant une façon différente de stocker les mots. Puis, ça nous a permis de nous familiariser avec la création d'interfaces graphiques avec le logiciel QT.

De plus, comme il a été mentionné dans ce rapport, nous avons dû dans un premier temps nous questionner sur les classes que nous avions besoin de créer pour notre logiciel. Dans un deuxième temps, nous avons dû prendre le temps de nous questionner sur comment nous allions implémenter nos classes et surtout comment nous allions créer nos algorithmes afin qu'ils soient optimaux. Dans un troisième et dernier temps, nous avions dû prendre le temps de nous questionner sur la façon que nous allions nous assurer de faire la connexion entre nos fonctions et l'interface graphique. Finalement, il serait intéressant de se questionner sur une façon pour laquelle nous aurions pu optimiser davantage notre logiciel pour qu'il puisse importer d'immenses lexiques et aussi faire une recherche rapide.