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
1828776	Tamer Arar
1827022	Hon-Leung Éric, Chao
1802395	Guy Frank Essome Penda

À :

David Johannès

Le 5 avril 2017

Introduction

Dans le cadre de ce travail pratique, nous avons été en mesure d'appliquer plusieurs notions théoriques de structures discrètes qui portent sur les automates et les langages. De plus, nous avons dû être en mesure d'appliquer des notions d'algorithme et de programmation en C++ et de Qt. Nous avons dû concevoir, à partir d'un lexique (un ensemble de mots d'un langage), un automate à états finis minimal qui doit contenir, entre autres, le nombre minimal d'états finis. Cet automate devait ensuite être utilisé pour faire la complétion de mots, c'est-à-dire suggérer des mots au fur et à mesure que l'on commence à taper un mot, et également d'en donner la correction si le mot entré est différent d'une lettre. Par exemple, un utilisateur qui commence à écrire les lettres « cas », les suggestions : case, cases, caser, casier et casiers doivent lui être suggérés. De plus, si le mot finalement écrit n'est pas écrit de la bonne façon, par exemple, *casier* est écrit « casirr », à l'appui du bouton espace. La correction du mot *casier* doit se faire et « casirr » est aussitôt remplacé par *casier*. Ajoutons que l'alphabet à considérer est celui de la langue française (26 lettres) et que les lettres accentuées ne sont pas prises en compte dans ce programme. Six lexiques ont été fournis dans le cadre de ce TP. Les lexiques 1 à 6, ont, respectivement, 100, 500, 1000, 2000 et 5000 mots. Lors de la réalisation de l'application graphique, il a fallu implémenter cinq composants :

- Composant 1. Création d'un automate à partir d'un lexique donné
- Composant 2. Une fonctionnalité qui suggère à l'utilisateur des mots qui complètent les premières lettres d'un mot que l'utilisateur écrit.
- Composant 3. Une fonctionnalité de correction qui corrige un mot mal écrit par l'utilisateur, dans le contexte particulier où le mot diffère d'une seule lettre du vrai mot.
- Composant 4. Une interface graphique qui donne à l'utilisateur la possibilité de saisir du texte et qu'à l'écriture de ce texte, des suggestions de complétion de mots lui sont montrées et la présence d'autocorrection.
- Composante 5. L'interface qui devrait afficher un menu suivant :
    a) Initialiser le programme
    b) Fonctionnalité suggestion
    c) Fonctionnalité correction
    d) Fonctionnalité suggestion + correction
    e) Retour
    f) Quitter

Présentation des travaux

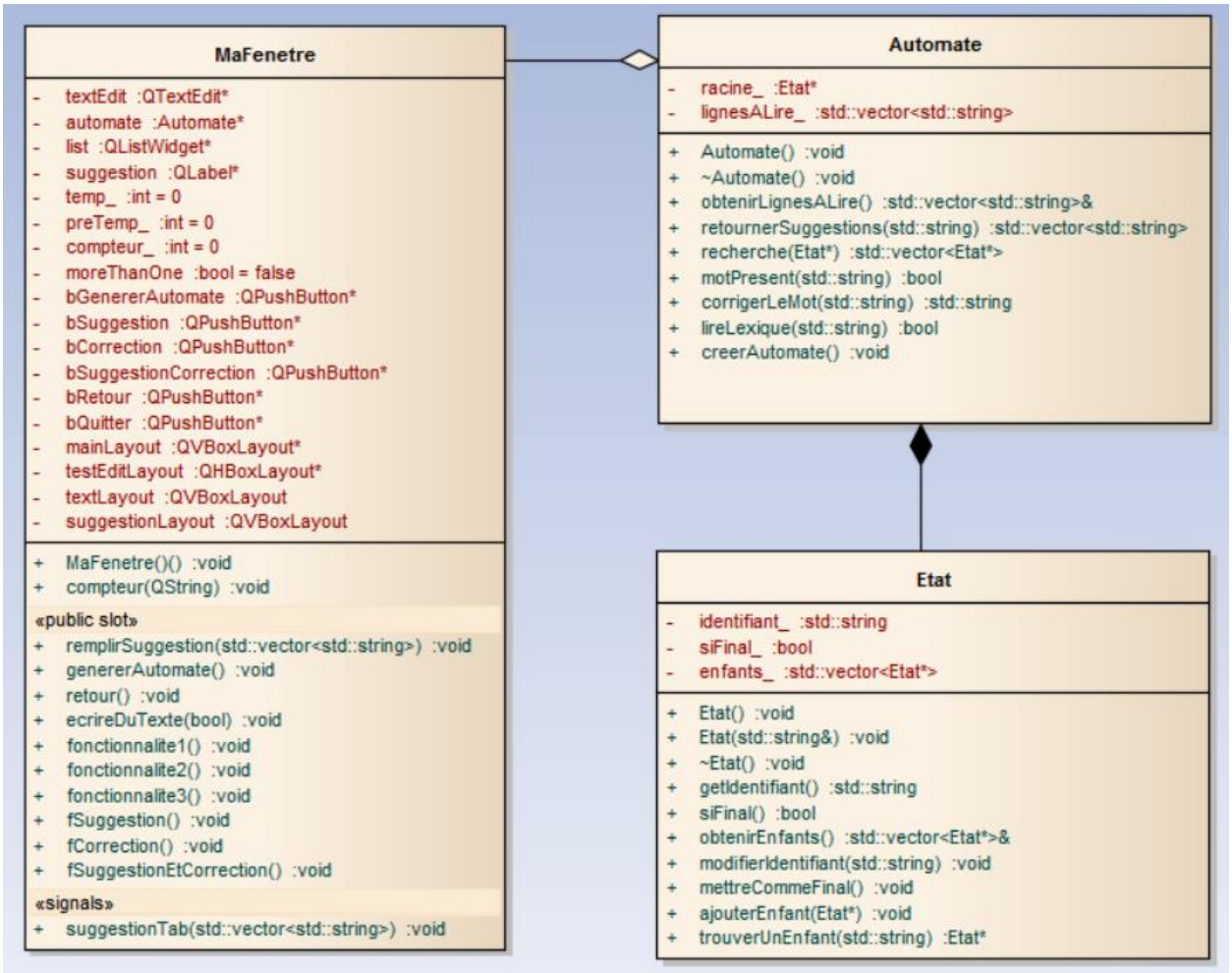

Figure 1 : diagramme de classes de la solution pour l'autocorrection et la suggestion de mot.

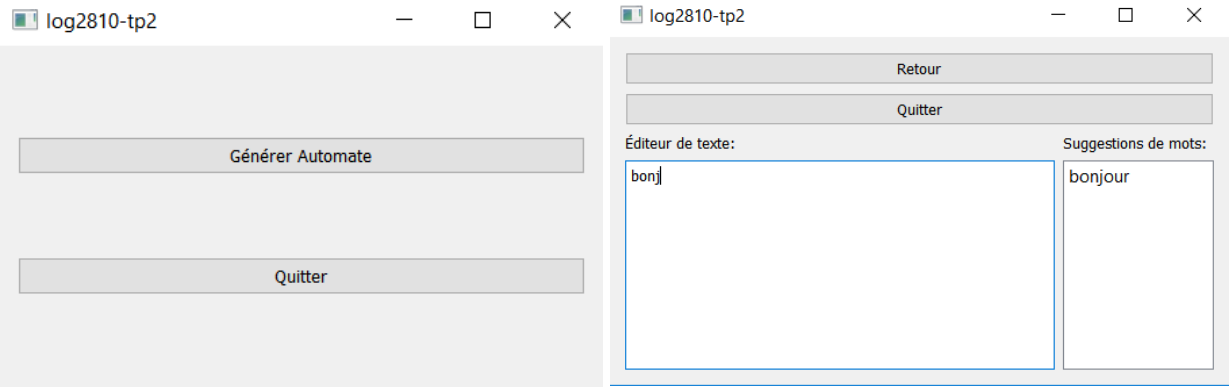

Figure 2 : interface de la solution pour les suggestions de mots.

Nous avons opté pour un projet sur Qt sur Windows pour faciliter la création d'une interface graphique. L'implémentation des composants logiques a été faite en programmation C++ orientée objet.

Tout d'abord, la classe Automate représente l'automate minimal qui contient des états. La classe Etat représente chacun de ses états. Un état est construit avec un simple nom. Au début, pour construire l'automate, on doit lire le lexique et l'entreposer dans un vecteur de la classe Automate où chaque élément de ce vecteur contient un mot du lexique. Cela est effectué par la méthode lireLexique(std ::string nomFichier). Ensuite, la méthode creerAutomate() se charge de créer l'automate à l'aide du vecteur de mots d'un lexique venant d'être rempli. On prend un premier mot et on sépare ce mot en caractères. On insère alors, sous la racine de l'automate (état vide qui ne contient rien), la première lettre du mot. Ensuite, on insère dans un deuxième état la première et la deuxième lettre du mot. On répète cela, en ajoutant toujours une lettre supplémentaire au prochain état, jusqu'à ce qu'on arrive à la fin du mot et qu'un état aura toutes les lettres de ce mot-ci. Ce dernier état est alors marqué comme état final, car il contient le mot complet. Ensuite, en minimisant cet automate, on devra alors obtenir par exemple quelque chose de similaire à cela pour un lexique contenant les mots; car, cas et case.

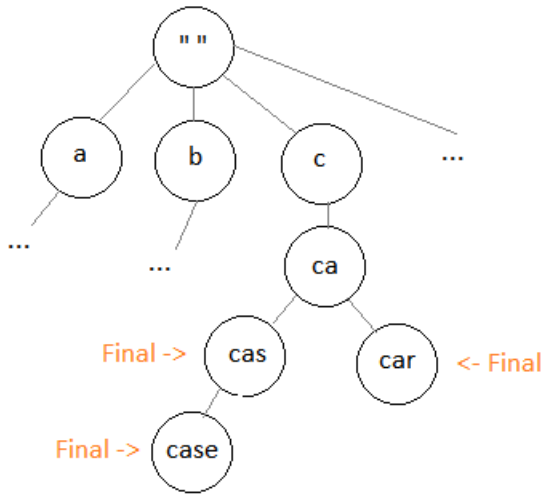

Figure 3 : présentation de l'automate.

Ainsi, lors de la création de l'automate, on s'assure que l'état n'existe pas avant de le créer.

Ensuite, pour la suggestion de mots, nous avons implémenté, dans la classe Automate, deux méthodes. La première est retournerSuggestions(std::string mot) qui prend le préfixe d'un mot pour en trouver des suggestions que l'on place dans un vecteur et qu'on limite le nombre de suggestions à un nombre raisonnable de dix. Cette méthode fait appel à une deuxième méthode recherche(Etat* pointDeDepart) qui commence la recherche à partir d'un point de départ (état de départ).

Puis, pour l'autocorrection, on a implémenté, dans la classe Automate, deux méthodes. La première méthode corrigerLeMot(std ::string mot) prend le mot complet à corriger est regarde, d'abord avec la deuxième méthode implémentée motPresent(std ::string mot) si le mot est bien écrit en faisait le parcours de l'automate et en regardant si le mot y est présent. Dans le cas contraire, on essaye de corriger le mot en remplaçant chacune des lettres du mot par une lettre de l'alphabet, soit parmi les 26 lettres, pour chaque lettre du mot entré et, à chaque fois, on regarde si le mot est présent dans l'automate. Finalement, la méthode corrigerLeMot() renvoi le mot corrigé.

Enfin, pour l'interface, il a fallu créer une classe MaFenetre qui contiendra les fenêtres nécessaires à création de l'interface graphique de l'utilisateur lui permettant, ainsi, de saisir du texte et d'avoir, en temps réel, la correction du mot entré et la suggestion des mots qui se fait au fur et à mesure qu'il tape. Un signal est effectué lorsque la saisie du texte change dans la fenêtre, et à ce moment-là, la méthode write() est appelée. C'est elle qui gère l'appel des fonctions de correction et de suggestions de l'automate expliquées précédemment.

De plus, un menu est implémenté pour sélectionner les fonctionnalités désirées, soit activer la suggestion de mots et/ou l'autocorrection.

Difficultés rencontrées lors du TP

Lors de la réalisation de ce travail pratique, nous avons rencontré quelques types de problèmes. Tout d'abord, nous avons dû gérer le travail en équipe de trois. Ainsi, chacun des membres de l'équipe avait une responsabilité qui lui était fixée pour contribuer à l'accomplissement de ce travail dans les temps. Cela ne nous a pas empêchés de résoudre les problèmes que les autres avaient tout au long du TP. C'est-à-dire on ne se tenait pas qu'à notre propre partie du TP, mais on contribuait tous activement à l'accomplissement de celui-ci. On se tenait à jour et la communication était active et présente. Aussi, il a fallu gérer l'écriture du code en ajoutant fréquemment des commentaires pour que les autres membres de l'équipe puissent se repérer facilement à la lecture du code et pour ne pas faire des ajouts de composants logiciels qui nuiraient au code qui est déjà fonctionnel.

Ensuite, du point de vue technique. Il a été d'abord dur de penser à l'implémentation d'un automate minimal. On a d'abord pensé à créer une classe qui représentait les vecteurs de l'automate, soit les transitions entrantes et sortantes, mais finalement, nous avons opté pour une solution plus simple en ne nous basant que sur les états et non leurs transitions. Ainsi, chaque état connait l'état qui le suit. Il a été aussi dur d'implémenter la fonctionnalité de correction de mot. La solution était d'échanger chacune des lettres du mot à corriger par une lettre de l'alphabet et de voir si le mot est présent dans l'automate.

Aussi, il était dur de concevoir une interface graphique en Qt, car on n'a pas assez d'expérience avec ce logiciel. Toutefois, on a appris après des heures de recherche son utilisation. Mais un nouveau problème fait surface: gérer la saisie du texte de l'utilisateur et permettre la continuité dans la saisie du texte, c'est-à-dire de permettre à l'usager de taper du texte en continu et de profiter des fonctionnalités de correction de mots et de suggestions de mots. Nous avons dû implémenter du code qui garde un historique de la saisie du texte et qui place le curseur de Qt à la bonne place après la correction des mots pour permettre à l'usager de toujours rentrer du texte et d'obtenir la suggestion de mots et la correction de chaque mot entrée.

Conclusion

Pour conclure, ce travail pratique nous a permis de comprendre l'utilité des concepts sur les automates appliqués au quotidien et de permettre la prise en compte de son importance dans le travail d'un ingénieur en informatique et en logiciel. Nous permettant de mettre en pratique des notions théoriques en pratique, le travail fut un succès. Ainsi, notre système fait bel et bien la suggestion de mots et l'autocorrection adéquate d'un mot qu'un usager doit entrer dans l'interface graphique Qt. Ce projet de programmation a poussé nos connaissances de programmation en C++ et en Qt un peu plus loin ce qui est bon pour notre avenir d'ingénieur. Il nous a également permis de développer de meilleures approches vis-à-vis les problèmes futurs à résoudre. La méthode d'apprentissage qu'on a établi améliorera notre faculté à apprendre plus facilement lors de la rencontre de nouveaux défis dans le futur, entre autres, lors de notre premier stage. Pour finir, on a trouvé que c'était un travail de taille qui a fait usage de notre créativité et de nos connaissances académiques et personnelles.