Algorithmes et structures de données 2
Laboratoire n°5 : Correcteur orthographique

15.12.2014

Introduction

Dans ce laboratoire, notre intention est de vous persuader que l'utilisation de structures de données adaptées est d'une importance fondamentale lorsque les données à stocker, traiter et rechercher sont nombreuses.

Objectifs

Le but de ce projet est d'implémenter un correcteur orthographique en anglais qui permet de trouver les fautes d'orthographe dans les mots composant un texte donné (allant du mot tout seul au livre complet) à l'aide d'un dictionnaire. Ce laboratoire s'inspire d'un ancien cours de programmation Ada donné l'EPFL [1].

Un mot est considéré comme correctement orthographié, s'il se trouve dans le dictionnaire de référence. Si un mot n'est pas dans le dictionnaire, votre programme devra proposer un ensemble de corrections possibles et les valider à l'aide du dictionnaire.

Durée

- 6 périodes. A rendre le **dimanche 25.01.2015** à minuit, au plus tard.

Donnée

- Vous trouverez le dictionnaire ainsi que des textes de référence dans le dossier du cours :
  `\\eistore1\Profs\OCE\ASD2 - 2014\Labos\Labo 5`
  Contrairement aux laboratoires précédents, vous serez cette fois-ci beaucoup plus libre pour déterminer quelles seront les structures à utiliser ainsi que comment organiser votre code. Mais le langage à utiliser reste bien entendu le C++.

- Votre programme devra lire 2 formats de fichiers différents :
  - Le **dictionnaire** : fichier texte, encodé en UTF-8 sans caractères accentués, comporte un mot par ligne.

- Un **document texte** : fichier texte, encodé en UTF-8, comporte plusieurs lignes de plusieurs mots. Vous devrez être en mesure d'accéder aux mots un par un.

Vous convertirez toutes les lettres en minuscules et supprimerez tous les caractères qui ne sont pas une lettre (a-z) ou une apostrophe (') en milieu de mot.

- Votre solution devra comporter au minimum 2 implémentations différentes, il doit être aisé de passer de l'une à l'autre (constante, switch, booléen à vous de voir mais cela doit être clairement expliqué) :
  - Une solution utilisant une/plusieurs structure STL. Vous devrez justifier votre choix dans les commentaires.
  - Une solution utilisant un *Ternary Search Trie*, que vous devrez implémenter vous-même. De la documentation est disponible dans le dossier du laboratoire sur `eistore1`.

  Vous pouvez bien entendu réutiliser du code des précédents laboratoires. Mais vous devrez dans tous les cas être en mesure de comprendre et de pouvoir expliquer le code rendu, en détail. Celui-ci devra être suffisamment commenté et les éventuelles sources clairement référencées. Votre code devra bien entendu pouvoir être compilé sur l'environnement de développement fixé en début de semestre. L'échange de code entre groupes n'est pas autorisé. Le non-respect de ces consignes sera lourdement sanctionné.

- Si un mot du texte n'est pas présent dans le dictionnaire, il sera considéré comme mal orthographié. Le logiciel générera 4 ensembles de propositions de corrections sur la base des 4 hypothèses suivantes :

  1. L'utilisateur a tapé une lettre supplémentaire : acqueux → aqueux (il y a un c en trop) ;
  2. L'utilisateur a oublié de taper une lettre : aqeux → aqueux (il manque la lettre u) ;
  3. L'utilisateur a mal tapé une lettre : awueux → aqueux (il y a un w à la place du q) ;
  4. L'utilisateur a échangé 2 lettres : auqeux → aqueux (u et q intervertis).

  Ces propositions seront vérifiées avec le dictionnaire et seules celles qui s'y trouvent seront proposées à l'utilisateur.

- Pour chaque texte pour lequel vous vérifierez l'orthographe, vous générerez un fichier texte comportant les mots mal orthographiés (préfixés d'un étoile) suivit immédiatement des propositions de correction vérifiées (préfixée du numéro de l'hypothèse). Les mots doivent rester dans l'ordre dans lequel ils apparaissent dans le texte, en cas de répétition d'un mot les propositions de correction devront à nouveau être présentées.
  Voir un exemple sur la Figure 1.
  Nous utiliserons un script pour vérifier vos résultats, vous serez donc pénalisez si vous ne respectez pas le format demandé.

- Vous afficherez dans la console le temps chargement du dictionnaire ainsi que le temps de correction du texte.

- Si le temps de chargement du dictionnaire en mémoire est supérieur à 1 minute (prétraitement compris), nous vous suggérons d'utiliser une version prétraitée du dictionnaire. Dans ce cas, vous devrez également nous remettre la version prétraitée du dictionnaire.

```
*lates
1:late
2:plates
2:latest
3:fates
3:gates
4:altes
…
*motsuivant
…
…
```

Figure 1 - Exemple de fichier de sortie

Bonus

Vous avez la possibilité d'obtenir jusqu'à 2 fois un bonus de 0.1 point sur la note de votre examen final, vous devrez pour cela réaliser un ou les deux points suivants :

- **Equilibrage de votre *Ternary Search Trie*** : utilisation de la technique AVL.
  Vous pourrez tester l'amélioration de votre implémentation à l'aide du fichier ordered_dictionary.txt. Votre solution devra permettre d'activer ou non l'équilibrage ainsi que d'afficher la hauteur maximum de votre trie.

- **Recherche avec *Wildcard*** :
  std::list<std::string> wildCardMatch(const std::string& pattern);
  Méthode appartenant à votre *Ternary Search Trie*, permettant de retourner la liste des mots du dictionnaire qui correspondent à un pattern. Le caractère « . » est la wildcard et peut être remplacé par tout autre caractère présent à ce niveau dans le trie. Nous n'accepterons pas une solution avec une recherche exhaustive.

Rendu/Evaluation

Il n'y a pas de rapport à rendre pour ce laboratoire. Vous devrez par contre apporter une attention particulière aux commentaires dans votre code, ceux-ci devront permettre d'identifier les étapes principales de vos implémentations ainsi que les choix que vous avez effectués. N'hésitez pas non plus à commenter les corrections obtenues.

Adresse E-Mail de l'assistant : fabien.dutoit@heig-vd.ch

Bonne chance !

Références

[1] Jörg Kienzle, *Programming course: spellchecker project*, EPFL, 2002.